阿里云
专有云Enterprise版

用户指南

产品版本：V3.0.0

文档版本：20171101

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的”现状“、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含”阿里云”、Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

表 1: 格式约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告:** 重启操作将导致业务中断，恢复业务所需时间约10分钟。
	用于警示信息、补充说明等，是用户必须了解的内容。	 **注意:** 导出的数据中包含敏感信息，请妥善保存。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明:** 您也可以通过按**Ctrl** + **A**选中全部文件。
>	多级菜单递进。	**设置** > **网络** > **设置网络类型**
粗体	表示按键、菜单、页面名称等UI元素。	单击 **确定**。
courier字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	bae log list --instanceid *Instance_ID*
[]或者[a|b]	表示可选项，至多选择一个。	ipconfig *[-all|-t]*
{}或者{a|b}	表示必选项，至多选择一个。	swich *{stand | slave}*

目录

法律声明 I

通用约定 I

1 云控制台简介 1

2 配置要求 2

3 登录云控制台 3

4 熟悉界面使用 5

5 角色与权限关系 7

6 管理云监控中心 9

6.1 云监控服务简介 9

6.2 云监控指标说明 9

6.3 管理报警联系人 13

6.3.1 新建报警联系人 13

6.3.2 添加报警联系人到报警联系组 14

6.3.3 查询报警联系人 15

6.3.4 修改报警联系人信息 15

6.3.5 删除报警联系人 15

6.4 管理报警联系组 16

6.4.1 新建报警联系组 16

6.4.2 修改报警通知方式 18

6.5 管理报警规则 18

6.5.1 新建报警规则 18

6.5.2 批量创建报警规则 20

6.6 管理报警项 22

6.6.1 查看报警项 23

6.6.2 查看报警历史 23

6.6.3 修改报警项 23

6.6.4 暂停报警项 24

6.6.5 批量启动报警项 24

6.6.6 查看报警通知对象 24

6.6.7 删除报警项 25

6.7 管理集群告警 26

6.7.1 创建监控组 26

6.7.2 将实例加入监控组 27

6.7.3 创建集群告警规则 28

6.7.4 查看集群告警信息 31

6.8 查看监控图表 31
6.9 查看报警信息 31
7 资源日常管理 33
7.1 查看运行大盘 33
7.1.1 查看区域资源总量 33
7.1.2 查看区域告警信息 34
7.2 查看并导出项目资源概览 34
7.3 配额管理 35
7.3.1 配额参数说明 35
7.3.2 查看配额 36
8 系统维护 37
8.1 管理部门 37
8.1.1 查看部门的项目 37
8.2 管理项目 37
8.2.1 查看项目详情 37
8.2.2 查看项目成员 38
8.2.3 查看项目资源信息 38
8.3 管理角色 39
8.3.1 权限说明 39
8.3.2 查看角色详情及角色下的用户 42
8.4 管理用户 43
8.4.1 查看用户基本信息 43
8.4.2 获取个人账号的AccessKey 43
8.5 管理登录策略 43
8.5.1 查看登录策略 43
9 管理工单 44
9.1 工单申报流程 44
9.2 提交申报 46
9.3 确认故障是否解决 47
9.4 删除工单 47
10 个性化设置 49
10.1 修改登录密码 49
10.2 修改头像 49
11 云服务器ECS 51
11.1 产品概述 51
11.1.1 实例的规格 52
11.1.2 实例的生命周期 54
11.2 使用前须知 55
11.2.1 禁忌 55

11.2.2 建议.....55
11.2.3 限制.....55
11.2.4 Windows 操作系统须知.....56
11.2.5 Linux 操作系统须知.....56
11.3 快速入门.....56
11.3.1 连接实例.....56
11.3.1.1 Linux或Mac OS X环境中使用SSH命令连接Linux实例.....57
11.3.1.2 Windows环境中通过远程连接工具连接Linux实例.....57
11.3.1.3 Windows环境中通过远程桌面连接Windows实例.....58
11.4 管理实例.....61
11.4.1 创建实例.....61
11.4.1.1 创建GPU可视化计算ga1实例.....61
11.4.1.2 创建GPU计算型gn4规格族实例.....62
11.4.1.3 创建GPU通用计算型gn5规格族实例.....63
11.4.1.4 创建GPU通用计算型gn5i规格族实例.....65
11.4.2 查看实例.....66
11.4.3 编辑实例.....66
11.4.4 停止、重启或启动实例.....67
11.4.5 修改实例密码.....67
11.4.6 登录VNC.....68
11.4.7 修改VNC密码.....68
11.4.8 加入安全组.....69
11.4.9 修改私网IP.....70
11.5 管理磁盘.....70
11.5.1 云盘简介.....71
11.5.2 查看磁盘.....71
11.6 管理镜像.....72
11.6.1 查看镜像.....72
11.7 管理快照.....72
11.7.1 查看快照.....73
11.8 管理自动快照策略.....73
11.8.1 查看自动快照策略.....73
11.8.2 为磁盘批量设置自动快照策略.....73
11.9 管理安全组.....74
11.9.1 安全组限制.....74
11.9.2 查看安全组.....74
11.10 安装 FTP 软件.....75
11.10.1 Redhat/CentOS 安装 vsftp 软件.....75
11.10.2 Ubuntu/Debian 安装 vsftp 软件.....75
11.10.3 Windows 2003 通过 IIS 配置 FTP.....76
11.10.4 Windows 2008 安装配置 FTP.....89

11.10.5 Windows 2012 安装配置 IIS+FTP....93
12 容器服务....99
12.1 产品概述....99
12.2 Swarm快速开始....99
12.2.1 创建Swarm集群....99
12.2.2 使用镜像创建wordpress应用....100
12.2.3 使用编排模板创建jenkins应用....103
12.3 Swarm mode快速开始....107
12.3.1 创建Swarm mode集群....107
12.3.2 使用镜像创建tomcat应用....108
12.3.3 使用编排模板创建nginx应用....112
12.4 swarm集群指南....115
12.4.1 管理集群....115
12.4.1.1 创建集群....115
12.4.1.2 添加已有云服务器....116
12.4.1.3 为集群绑定负载均衡....118
12.4.1.4 重新绑定负载均衡....119
12.4.1.5 为集群解绑负载均衡....120
12.4.1.6 搜索集群....121
12.4.1.7 删除集群....122
12.4.1.8 清理集群磁盘....122
12.4.1.9 连接集群....123
12.4.1.10 升级系统服务....125
12.4.2 管理节点....126
12.4.2.1 移除节点....126
12.4.2.2 查看节点上运行的容器....127
12.4.3 安全组....128
12.4.3.1 容器服务安全组规则....128
12.4.3.2 检测安全组....129
12.4.3.3 重新绑定安全组....130
12.4.4 管理编排模板....131
12.4.4.1 创建编排模板....131
12.4.4.2 查看编排模板列表....131
12.4.4.3 更新编排模板....131
12.4.4.4 下载编排模板....132
12.4.4.5 删除编排模板....133
12.4.4.6 另存编排模板....133
12.4.5 服务编排....133
12.4.5.1 标签概览....135
12.4.5.2 probe....136
12.4.5.3 rolling_updates....137

12.4.5.4 depends....139
12.4.5.5 scale....139
12.4.5.6 routing标签....140
12.4.5.7 lb....142
12.4.5.8 global....143
12.4.5.9 服务部署约束（affinity:service）....144
12.4.5.10 external....145
12.4.5.11 dns_options....145
12.4.5.12 oom_kill_disable....146
12.4.5.13 变量替换....146
12.4.5.14 容器重新调度....146
12.4.5.15 不支持的Docker Compose标签....147
12.4.6 管理应用....147
12.4.6.1 使用镜像创建应用....147
12.4.6.2 使用编排模板创建应用....150
12.4.6.3 限制容器资源....152
12.4.6.4 指定节点调度....153
12.4.6.5 指定多节点调度....154
12.4.6.6 删除用户标签....155
12.4.6.7 查看应用详情....156
12.4.6.8 启动或停止应用....157
12.4.6.9 变更应用配置....158
12.4.6.10 重新部署应用....160
查看重新部署是否成功....160
12.4.6.11 删除应用....161
12.4.6.12 运行离线任务....162
12.4.6.13 定时任务....165
12.4.6.14 系统默认应用列表....167
12.4.6.15 创建并调用触发器....167
12.4.6.16 删除触发器....171
12.4.7 配置项....172
12.4.8 管理服务....177
12.4.8.1 查看服务详情....177
12.4.8.2 启动或停止服务....179
12.4.8.3 变更服务配置....179
12.4.8.4 重新调度服务....180
12.4.8.5 重启服务....181
12.4.8.6 删除服务....182
12.4.9 网络管理....183
12.4.9.1 跨主机互联的容器网络....183
12.4.10 管理数据卷....184

12.4.10.1 开通数据卷功能 184
12.4.10.2 创建OSSFS数据卷 184
12.4.10.3 创建NAS数据卷 186
12.4.10.4 查看数据卷 186
12.4.10.5 使用第三方数据卷 187
12.4.10.6 常见问题 189
12.4.11 管理日志 190
12.4.11.1 查看集群日志 190
12.4.11.2 查看应用日志 191
12.4.11.3 查看服务日志 192
12.4.11.4 查看容器日志 193
12.4.12 使用访问控制 194
12.4.13 服务发现和负载均衡 197
12.4.13.1 简单路由（支持 HTTP/HTTPS） 198
12.4.13.2 简单路由-域名配置 202
12.4.13.3 简单路由-HTTP协议变为HTTPS协议 204
12.4.13.4 集群内服务间路由和负载均衡 206
12.4.13.5 负载均衡路由 208
12.4.13.6 容器间的互相发现 212
12.4.14 发布策略 214
12.4.14.1 概述 214
12.4.14.2 使用简单路由蓝绿发布 215
12.4.14.3 使用负载均衡路由蓝绿发布 220
12.5 swarm mode集群指南 226
12.5.1 管理集群 226
12.5.1.1 创建集群 226
12.5.1.2 添加已有云服务器 227
12.5.1.3 为集群绑定负载均衡 229
12.5.1.4 重新绑定负载均衡 230
12.5.1.5 为集群解绑负载均衡 231
12.5.1.6 搜索集群 232
12.5.1.7 删除集群 233
12.5.1.8 清理集群磁盘 233
12.5.1.9 连接集群 234
12.5.1.10 升级系统服务 236
12.5.2 管理节点 237
12.5.2.1 移除节点 237
12.5.2.2 查看节点上运行的容器 238
12.5.2.3 设置为Worker 239
12.5.2.4 设置为Manager 240
12.5.3 安全组 241

12.5.3.1 容器服务安全组规则.....241
12.5.3.2 检测安全组.....242
12.5.3.3 重新绑定安全组.....242
12.5.4 管理编排模板.....243
12.5.4.1 创建编排模板.....243
12.5.4.2 查看编排模板列表.....244
12.5.4.3 更新编排模板.....244
12.5.4.4 下载编排模板.....244
12.5.4.5 删除编排模板.....245
12.5.4.6 另存编排模板.....245
12.5.5 服务编排.....246
12.5.5.1 标签概览.....248
12.5.5.2 健康检查.....249
12.5.5.3 rolling_updates.....252
12.5.5.4 depends.....254
12.5.5.5 服务的容器数量.....255
12.5.5.6 routing标签.....257
12.5.5.7 lb.....260
12.5.5.8 global.....262
12.5.5.9 服务部署约束（affinity:service）.....263
12.5.5.10 external.....265
12.5.5.11 变量替换.....266
12.5.5.12 容器重新调度.....267
12.5.5.13 networks.....268
12.5.5.14 不支持的Docker Compose标签.....269
12.5.6 管理应用.....269
12.5.6.1 使用镜像创建应用.....269
12.5.6.2 使用编排模板创建应用.....272
12.5.6.3 限制容器资源.....274
12.5.6.4 指定节点调度.....276
12.5.6.5 指定多节点调度.....278
12.5.6.6 删除用户标签.....280
12.5.6.7 查看应用详情.....281
12.5.6.8 变更应用配置.....282
12.5.6.9 重新部署应用.....284
查看重新部署是否成功.....285
12.5.6.10 删除应用.....285
12.5.6.11 系统默认应用列表.....287
12.5.6.12 创建并调用触发器.....287
12.5.6.13 删除触发器.....290
12.5.7 配置项.....291

12.5.8 管理服务.....296
12.5.8.1 查看服务详情.....296
12.5.8.2 变更服务配置.....298
12.5.8.3 删除服务.....299
12.5.9 管理网络.....300
12.5.9.1 跨主机互联的容器网络.....301
12.5.9.2 创建自定义网络.....302
12.5.9.3 查看网络列表.....304
12.5.9.4 添加应用到自定义网络.....305
12.5.9.5 查看容器的网络.....306
12.5.9.6 删除自定义网络.....307
12.5.10 管理密钥.....308
12.5.10.1 概述.....308
12.5.10.2 创建密钥.....309
12.5.10.3 查看密钥列表.....310
12.5.10.4 在服务中使用密钥.....311
12.5.10.5 更新服务的密钥.....314
12.5.10.6 删除密钥.....316
12.5.11 管理数据卷.....317
12.5.11.1 开通数据卷功能.....317
12.5.11.2 创建OSSFS数据卷.....318
12.5.11.3 创建NAS数据卷.....319
12.5.11.4 查看数据卷.....320
12.5.12 管理日志.....321
12.5.12.1 查看集群日志.....321
12.5.12.2 查看应用日志.....322
12.5.12.3 查看服务日志.....323
12.5.12.4 查看容器日志.....324
12.5.13 使用访问控制.....325
12.5.14 服务发现和负载均衡.....328
12.5.14.1 简单路由（支持 HTTP/HTTPS）.....328
12.5.14.2 简单路由-域名配置.....332
12.5.14.3 简单路由-HTTP协议变为HTTPS协议.....334
12.5.14.4 集群内服务间路由和负载均衡.....336
12.5.14.5 负载均衡路由.....338
12.5.14.6 服务发现.....342
12.5.15 发布策略.....344
12.5.15.1 概述.....344
12.5.15.2 使用简单路由蓝绿发布.....345
13 对象存储OSS.....351
13.1 什么是 OSS.....351

13.2 基本概念 351
13.3 快速开始 352
13.3.1 上传文件 352
13.3.2 获取文件访问地址 352
13.4 管理存储空间 353
13.4.1 查看存储空间 353
13.4.2 设置访问权限 354
13.4.3 设置静态网站托管 354
13.4.4 设置日志 355
13.4.5 设置 IP 防盗链 356
13.4.6 管理生命周期 356
13.4.7 设置跨域资源共享 357
13.5 管理对象（文件） 359
13.5.1 获取AccessKey 359
13.5.2 新建文件夹 359
13.5.3 搜索文件 360
13.5.4 设置文件 ACL 361
13.5.5 删除文件 361
13.6 图片处理 362
13.6.1 创建样式 362
14 消息服务 365
14.1 产品概述 365
14.2 使用前须知 365
14.3 基本概念 365
14.4 使用子账号访问 MNS 368
14.4.1 创建子账号 368
14.4.2 新建授权策略 369
14.4.3 授权子账号访问 MNS 370
14.4.4 授权策略示例 370
14.4.4.1 示例 1：在 Allow 授权中增加 IP 限制 370
14.4.4.2 示例 2：在 Deny 授权中增加 IP 限制 370
14.4.4.3 示例 3：授予 MNS 控制台可读权限 371
14.4.5 MNS API授权映射表 371
14.5 管理队列 373
14.5.1 创建队列 373
14.5.2 设置队列属性 373
14.5.3 获取队列属性 373
14.5.4 删除队列 373
14.5.5 发送消息 374
14.5.6 消费消息 374
14.5.7 删除消息 374

15 表格存储TableStore..... 375
15.1 什么是表格存储..... 375
15.2 实例介绍..... 375
15.3 快速开始..... 376
15.3.1 创建表..... 376
15.4 管理实例..... 377
15.4.1 查看实例..... 377
15.5 管理数据表..... 377
15.5.1 更新表..... 377
15.5.1.1 调整高性能实例的预留读/写吞吐量..... 377
15.5.2 查看数据表基本详情..... 378
15.5.3 删除表..... 378
15.6 管理VPC..... 379
15.7 附录：限制说明..... 379
16 云数据库RDS版..... 381
16.1 产品概述..... 381
16.2 使用限制..... 381
16.3 使用流程..... 382
16.4 初始化配置..... 383
16.4.1 设置白名单..... 383
16.4.2 创建数据库和账号..... 384
16.5 连接实例..... 387
16.5.1 通过DMS登录实例..... 387
16.5.2 通过客户端连接MySQL实例..... 389
16.6 只读实例..... 393
16.6.1 只读实例简介..... 393
16.6.2 管理只读实例..... 394
16.6.2.1 直接通过只读实例进入管理页面..... 394
16.6.2.2 通过主实例进入只读实例的管理页面..... 395
16.7 实例管理..... 395
16.7.1 查询详情..... 395
16.7.2 重启实例..... 395
16.7.3 设置参数..... 396
16.7.4 修改实例名称..... 396
16.7.5 典型参数设置..... 397
16.7.5.1 MySQL实例可修改参数说明..... 397
16.7.5.2 MySQL实例参数调优最佳实践..... 415
16.7.5.2.1 前言..... 415
16.7.5.2.2 MySQL实例不支持修改的参数..... 415
16.7.5.2.3 MySQL实例支持修改的参数..... 415

16.7.5.2.4 如何设置参数.....415
16.7.5.2.5 MySQL新增参数.....418
16.8 账号管理.....419
16.8.1 创建账号.....419
16.8.2 重置密码.....420
16.8.3 修改账号权限.....421
16.8.4 删除账号.....421
16.8.5 修改描述.....421
16.9 管理数据库.....422
16.9.1 创建数据库.....422
16.9.2 删除数据库.....423
16.10 设置访问模式.....424
16.11 安全管理.....424
16.11.1 设置白名单.....424
16.11.2 审计日志.....426
16.12 性能优化.....426
16.12.1 慢SQL统计.....426
16.12.2 缺失索引.....427
16.13 备份与恢复.....427
16.13.1 备份RDS数据.....427
16.13.1.1 自动备份.....427
16.13.1.2 手动备份.....428
16.13.2 恢复RDS数据.....429
16.13.2.1 通过临时实例恢复到主实例.....429
16.13.2.2 直接恢复到主实例.....433
16.13.3 Binlog日志.....433
16.14 系统资源监控.....433
16.15 迁移本地数据到RDS.....435
16.15.1 压缩数据.....435
16.15.2 迁移MySQL数据.....436
16.15.2.1 使用DTS迁移MySQL数据.....436
16.15.2.2 使用mysqldump迁移MySQL数据.....443
16.16 典型应用.....446
16.16.1 多结构数据存储.....446
17 云数据库Redis版.....448
17.1 产品概述.....448
17.2 管理实例.....448
17.2.1 修改实例密码.....448
17.2.2 查看实例详情.....448
17.2.3 修改实例名称.....449
17.2.4 变更配置.....449

17.2.5 设置IP白名单.....450
17.2.6 设置可运维时间.....450
17.3 导入数据.....451
17.4 备份与恢复.....452
17.4.1 设置自动备份策略.....452
17.4.2 手动备份.....452
17.4.3 备份存档.....453
17.4.4 数据恢复.....453
17.5 设置云数据库参数.....454
17.6 Redis支持的命令.....454
18 数据传输服务DTS.....459
18.1 产品概述.....459
18.2 快速开始.....459
18.2.1 创建数据迁移任务.....459
18.2.2 创建配置文件导入任务.....464
18.2.3 创建实时同步作业.....469
18.3 数据迁移.....474
18.3.1 数据迁移功能简介.....474
18.3.2 预检查项.....476
18.3.2.1 源库连接性检查.....476
18.3.2.2 目标数据库连接性检查.....477
18.3.2.3 源库binlog检查.....478
18.3.2.4 约束完整性检查.....479
18.3.2.5 federated类型的表的存在性检查.....479
18.3.2.6 权限检查.....479
18.3.2.7 同名对象存在性检查.....480
18.3.2.8 schema存在性检查.....480
18.3.2.9 源库server-id检查.....481
18.3.2.10 源库版本号检查.....481
18.3.3 从本地Oracle迁移至RDS for MySQL.....481
18.3.4 从本地MySQL迁移至RDS for MySQL.....487
18.3.5 从本地SQL Server迁移至RDS for SQL Server.....493
18.3.6 SQL Server不停机迁移.....499
18.3.7 Oracle到PPAS不停机数据迁移.....505
18.3.8 从本地PostgreSQL迁移至RDS for PostgreSQL.....513
18.3.9 从本地MongoDB迁移至MongoDB实例.....522
18.3.10 RDS实例间的数据迁移.....527
18.3.11 从ECS上的自建数据库迁移至RDS/MongoDB/PetaData/OceanBase.....532
18.3.12 从MySQL迁移至DRDS.....537
18.3.13 库表列映射.....541
18.3.14 配置SQL条件过滤迁移数据.....545

18.3.15 迁移失败修复.....547
18.4 数据同步.....550
18.4.1 创建RDS实例间数据实时同步作业.....550
18.4.2 预检查失败修复.....557
18.4.3 新增同步对象.....562
18.4.4 减少同步对象.....564
18.4.5 配置监控报警.....565
18.4.6 查看同步性能.....567
18.4.7 跨阿里云账号RDS实时同步.....568
19 数据管理.....575
19.1 关系型数据库.....575
19.1.1 功能总览.....575
19.1.2 SQL操作.....577
19.1.2.1 使用命令窗口.....577
19.1.2.2 使用SQL窗口.....580
19.1.2.2.1 打开空的SQL窗口.....580
19.1.2.2.2 恢复已保存的SQL窗口.....587
19.1.2.2.3 管理常用SQL命令.....589
19.1.2.2.4 使用SQL模板.....590
19.1.2.2.5 生成图表.....590
19.1.2.3 表操作（基于表目录树）.....592
19.1.2.3.1 打开基于表的SQL窗口.....592
19.1.2.3.2 编辑表数据.....593
19.1.3 数据库开发.....594
19.1.3.1 管理表.....594
19.1.3.1.1 新增表.....594
19.1.3.1.2 修改表.....595
19.1.3.1.3 删除表.....596
19.1.3.1.4 创建类似表.....597
19.1.3.1.5 生成模板SQL.....597
19.1.3.1.6 查询表信息.....598
19.1.3.1.7 清空数据.....598
19.1.3.1.8 批量操作表.....599
19.1.3.1.9 表维护.....600
19.1.3.2 管理索引.....600
19.1.3.3 管理外键.....602
19.1.3.4 创建分区.....603
19.1.3.5 创建存储过程.....604
19.1.3.6 创建函数.....606
19.1.3.7 创建视图.....607
19.1.3.8 创建触发器.....608

19.1.3.9 创建事件.....611
19.1.4 数据管理.....613
19.1.4.1 导入数据.....613
19.1.4.2 导出数据.....615
19.1.4.2.1 导出数据库.....615
19.1.4.2.2 导出SQL结果集.....615
19.1.5 性能管理.....616
19.1.5.1 管理锁等待.....616
19.1.5.1.1 查看锁等待.....616
19.1.5.1.2 释放锁等待.....617
19.1.5.2 管理会话.....619
19.1.5.2.1 查看会话.....619
19.1.5.2.2 结束会话.....620
19.1.5.2.3 优化会话.....620
19.1.5.3 查看实时性能.....620
19.1.6 扩展工具.....621
19.1.6.1 表数据量统计.....621
19.1.6.2 E-R 图.....622
20 负载均衡SLB.....624
20.1 产品概述.....624
20.2 规划和准备.....624
20.3 配置监听.....625
20.3.1 配置四层监听.....626
20.3.2 配置七层监听.....627
20.3.3 设置访问控制.....629
20.3.4 配置转发策略.....630
20.4 添加后端服务器.....630
20.5 管理虚拟服务器组.....631
20.6 管理负载均衡实例.....631
20.6.1 启动或停止实例.....631
20.6.2 查看实例详情.....632
20.6.3 修改实例属性.....632
20.7 管理证书.....632
20.7.1 证书格式.....633
20.7.2 生成客户端证书.....634
21 企业级分布式应用服务EDAS.....636
21.1 产品概述.....636
21.2 快速开始.....636
21.2.1 安装EDAS Agent.....637
21.2.2 创建应用.....638

21.2.3 部署和启动应用..... 640
21.3 概览页介绍..... 643
21.4 资源管理..... 644
21.4.1 云服务器ECS..... 644
21.4.1.1 管理ECS实例..... 644
21.4.1.2 安装 Agent..... 646
21.4.1.2.1 EDAS Agent 简介..... 646
21.4.1.2.2 安装 Agent..... 647
21.4.2 负载均衡 SLB..... 648
21.4.3 专有网络 VPC..... 649
21.4.3.1 查看 VPC 实例..... 649
21.4.3.2 安装日志采集器..... 650
21.4.4 集群管理..... 651
21.4.4.1 查看集群..... 651
21.4.4.2 创建集群..... 652
21.4.4.3 管理集群..... 653
21.4.5 资源组管理..... 655
21.5 使用 Docker 部署应用..... 657
21.5.1 准备工作..... 658
21.5.2 镜像定制..... 662
21.5.3 部署镜像..... 663
21.6 应用管理..... 666
21.6.1 应用生命周期管理..... 666
21.6.1.1 概述..... 666
21.6.1.2 应用发布..... 668
21.6.1.2.1 创建应用..... 668
21.6.1.2.2 部署应用..... 672
21.6.1.2.3 启动应用..... 675
21.6.1.3 回滚应用..... 676
21.6.1.4 手动扩容..... 676
21.6.1.5 手动缩容..... 676
21.6.1.6 删除应用..... 677
21.6.2 应用设置..... 677
21.6.2.1 配置JVM参数..... 677
21.6.2.2 配置Tomcat..... 679
21.6.2.3 配置负载均衡..... 681
21.6.2.4 设置健康检查..... 681
21.6.3 运行日志..... 683
21.6.4 应用监控..... 686
21.6.4.1 监控大盘..... 689
21.6.4.2 基础监控..... 691

21.6.4.3 服务监控 693
21.6.5 通知报警 697
21.6.6 弹性伸缩 701
21.6.7 限流降级 703
21.6.7.1 限流管理 705
21.6.7.2 降级管理 707
21.6.8 容器诊室 708
21.7 全局配置推送 711
21.8 账号管理 713
21.8.1 EDAS 账号体系介绍 713
21.8.2 管理主账号 717
21.8.2.1 绑定主账号 717
21.8.2.2 角色管理 719
21.8.2.3 查看所有权限 720
21.8.3 管理子账号 721
21.8.3.1 子账号简介 721
21.8.3.2 创建子账号 722
21.8.3.2.1 创建RAM子账号 723
21.8.3.2.2 RAM 子账号登录 EDAS 控制台 725
21.8.3.2.3 RAM 账号授权 727
21.8.3.2.4 RAM 用户解绑 729
21.8.3.3 主账号对子账号的操作 730
21.9 常见问题 732
21.9.1 开发常见问题 732
21.9.1.1 Ali-Tomcat问题 732
21.9.1.2 轻量配置中心问题 734
21.9.1.3 HSF常见问题 736
21.9.1.4 HSF错误码 738
21.9.1.5 其他开发问题 744
21.9.2 使用常见问题 744
21.9.2.1 账号管理类 744
21.9.2.2 资源管理类 744
21.9.2.3 应用生命周期类 746
21.9.2.4 监控报警类 749
22 分布式关系型数据库DRDS 750
22.1 产品概述 750
22.2 管理DRDS实例 751
22.2.1 创建实例 751
22.2.2 变更配置 752
22.2.3 释放实例 753
22.2.4 实例监控 753

22.2.4.1 预防性能问题.....756
22.2.4.2 示例一.....756
22.2.4.3 示例二.....757
22.2.4.4 示例三.....759
22.3 DRDS账号和权限系统.....761
22.3.1 基本概念.....761
22.3.2 账号权限语句说明.....764
22.4 管理数据库.....767
22.4.1 创建数据库.....767
22.4.2 查看数据库.....771
22.4.3 管理只读账户.....772
22.4.4 设置读写分离.....774
22.4.5 查看监控信息.....775
22.4.6 查看分库信息.....776
22.4.7 管理数据表.....776
22.4.8 设置白名单.....779
22.4.9 导入数据.....780
22.4.10 删除数据库.....783
22.4.11 进行平滑扩容.....783
22.4.12 设置小表广播.....788
22.4.13 设置异构索引.....790
22.5 使用DRDS.....793
22.5.1 使用DDL语句.....793
22.5.2 使用DML语句.....795
22.5.3 使用全局唯一数字序列.....796
22.5.3.1 Sequence显示用法.....799
22.5.3.2 Sequence隐式用法.....803
22.5.3.3 Sequence限制及注意事项.....805
22.5.4 全表扫描.....806
22.5.5 读写分离.....808
22.5.6 DRDS HINT.....810
22.5.7 DRDS自定义控制指令.....811
22.5.7.1 一般帮助语句.....811
22.5.7.2 查看规则和节点拓扑语句.....813
22.5.7.3 SQL调优语句.....820
22.5.7.4 统计信息查询语句.....830
22.5.8 慢SQL明细.....831
22.6 最佳实践.....834
22.6.1 如何选择拆分键.....834
22.6.2 如何选择分片数.....836
22.6.3 SQL优化基本概念.....837

22.6.4 SQL优化方法.....841
22.6.4.1 单表SQL优化.....842
22.6.4.2 JOIN优化.....846
22.6.4.3 子查询优化.....850
22.6.5 如何选择应用连接池.....850
22.6.6 如何排查 DRDS 慢 SQL.....851
22.6.6.1 定位慢SQL.....851
22.6.6.2 定位性能损耗节点.....853
22.6.6.3 处理性能损耗节点.....855
22.6.7 如何处理DDL异常.....855
22.6.8 如何高效扫描DRDS数据.....859
22.7 附录：DRDS术语表.....861
23 消息队列MQ.....869
23.1 产品概述.....869
23.2 名词解释.....870
23.3 快速开始.....871
23.3.1 申请资源.....872
23.3.1.1 创建Topic.....872
23.3.1.2 创建Producer ID.....873
23.3.1.3 创建Consumer ID.....874
23.3.2 发送消息.....874
23.3.2.1 控制台发送消息.....875
23.3.2.2 调用SDK/API发送消息.....875
23.3.2.3 查看消息是否发送成功.....877
23.3.3 订阅消息.....878
23.4 MQ消息类型.....879
23.4.1 事务消息.....879
23.4.2 定时消息和延时消息.....881
23.4.3 顺序消息.....883
23.5 常见使用方式说明.....886
23.5.1 消息过滤.....886
23.5.2 集群消费和广播消费.....889
23.5.3 订阅关系一致.....891
23.5.4 消息重试.....893
23.5.4.1 重试次数.....893
23.5.4.2 配置方式.....894
23.5.5 消费幂等.....896
23.6 使用消息队列控制台.....896
23.6.1 查询消息.....897
23.6.1.1 根据Topic查询.....898
23.6.1.2 根据Message Key查询.....899

23.6.1.3 根据Message ID查询....900
23.6.2 查询资源报表....902
23.6.2.1 查询消费消息....902
23.6.2.2 查询生产消息....903
23.6.3 查询消息轨迹....904
23.6.3.1 消息轨迹简介....904
23.6.3.2 消息轨迹查询....906
23.6.3.3 消息轨迹状态说明....909
23.6.4 监控报警....910
23.7 TCP接入....911
23.7.1 使用Java SDK 接入....911
23.7.1.1 Java SDK使用说明....911
23.7.1.2 准备环境....913
23.7.1.3 日志配置....914
23.7.1.4 发送普通消息（三种方式）....916
23.7.1.4.1 可靠同步发送....916
23.7.1.4.2 可靠异步发送....917
23.7.1.4.3 单向（Oneway）发送....919
23.7.1.5 发送分布式事务消息....921
23.7.1.6 发送延时消息....924
23.7.1.7 发送定时消息....925
23.7.1.8 发送顺序消息....925
23.7.1.9 集群方式订阅消息....927
23.7.1.10 Spring 集成....928
23.7.1.10.1 生产者与 Spring 集成....928
23.7.1.10.2 事务消息生产者与 Spring 集成....929
23.7.1.10.3 消费者与 Spring 集成....931
23.7.2 使用C/C++ SDK 接入....932
23.7.2.1 准备环境....932
23.7.2.1.1 下载SDK....933
23.7.2.1.2 使用 Linux C++ SDK....934
23.7.2.1.3 使用 Windows C++ SDK....935
23.7.2.2 发送普通消息....943
23.7.2.3 发送定时消息....944
23.7.2.4 发送分布式事务消息....946
23.7.2.5 集群方式订阅消息....949
23.7.3 .NET SDK....951
23.7.3.1 环境准备....951
23.7.3.1.1 下载SDK....951
23.7.3.1.2 .NET SDK 配置说明....952
23.7.3.2 发送普通消息....959

23.7.3.3 发送定时消息.....960
23.7.3.4 发送分布式事务消息.....962
23.7.3.5 集群方式订阅消息.....965
23.8 MQTT接入.....967
23.8.1 应用场景和优势.....969
23.8.2 名词解释.....970
23.8.3 MQTT接入准备.....971
23.8.3.1 准备环境.....971
23.8.3.2 申请资源.....972
23.8.3.3 签名计算.....974
23.8.3.4 获取客户端上下线事件.....974
23.8.4 Java接入示例.....977
23.8.4.1 MQTT 客户端收发 MQTT 消息.....977
23.8.4.2 MQ 客户端收发 MQTT 消息.....982
23.8.5 Android接入示例.....985
23.8.6 JavaScript接入示例.....985
23.8.7 iOS接入示例.....987
23.8.8 Python接入示例.....990
23.8.9 JavaScript接入示例.....992
23.8.10 MQTT实例管理.....993
23.8.10.1 查看实例信息.....993
23.8.10.2 设置实例报警.....994
23.8.11 查询MQTT数据.....995
23.8.11.1 查看设备信息.....995
23.8.11.2 查看状态信息.....997
23.8.11.3 查看 Topic 信息.....999
23.8.11.4 查询资源报表.....1001
23.8.12 MQTT常见问题.....1001
23.9 产品使用问题.....1002
23.9.1 常见问题.....1002
23.9.1.1 快速开始.....1002
23.9.1.2 配置相关.....1004
23.9.1.3 消息轨迹.....1004
23.9.1.4 告警处理.....1005
23.9.2 异常类别.....1006
23.9.2.1 使用异常.....1006
23.9.2.2 资源不存在.....1008
23.9.2.3 状态不一致.....1009
24 企业实时监控服务ARMS.....1013
24.1 产品概述.....1013
24.2 名词解释.....1013

24.3 准备数据源.....1014
24.3.1 数据源简介..... 1014
24.3.2 添加数据源..... 1015
24.3.3 安装Agent.....1017
24.4 创建监控任务..... 1018
24.4.1 监控任务概述.....1018
24.4.2 配置数据源..... 1020
24.4.3 日志清洗.....1024
24.4.3.1 智能切分..... 1024
24.4.3.2 自定义切分..... 1025
24.4.4 创建和设置报警控件..... 1027
24.4.5 创建和查询多维数据集..... 1030
24.5 管理监控任务..... 1034
24.5.1 查看监控任务详情.....1034
24.5.2 监控任务操作.....1037
24.6 查询监控结果..... 1040
24.6.1 管理报警..... 1040
24.6.2 查询数据集..... 1042
24.7 定制大盘..... 1044
24.7.1 交互式大盘简介..... 1044
24.7.2 创建交互式大盘..... 1045
24.7.3 管理交互式大盘..... 1050
24.8 管理系统配置..... 1051
24.8.1 映射表管理..... 1051
24.8.2 联系人管理..... 1054
24.8.3 联系人分组管理..... 1056
24.9 最佳实践..... 1058
24.9.1 映射表的使用.....1058
24.9.2 日志清洗.....1063
24.10 产品使用问题..... 1065
24.10.1 自定义清洗常见的问题..... 1065
24.10.2 日志清洗中时间字段（date）的常见问题.....1066
24.10.3 智能切分日期格式.....1071
24.10.4 日志清洗中系统字段介绍.....1073
24.10.5 自定义切分的使用.....1074
24.10.6 常见问题..... 1086
25 云服务总线CSB..... 1088
25.1 产品概述..... 1088
25.2 使用前须知.....1089
25.3 基本概念..... 1090
25.4 应用场景..... 1091

25.5 发布API服务.....1094
25.5.1 创建API分组.....1094
25.5.2 发布服务（旧版）.....1095
25.6 管理发布的API.....1099
25.6.1 查看发布的API列表.....1099
25.6.2 查看与编辑API详情.....1100
25.6.3 授权API订阅.....1101
25.7 订阅 API.....1102
25.7.1 创建消费凭证.....1102
25.7.2 订阅API服务.....1103
25.8 管理订阅的API.....1104
25.8.1 查看及退订API订阅.....1105
25.8.2 查看API订阅详情.....1106
25.9 API调用.....1107
26 MaxCompute.....1109
26.1 前言.....1109
26.2 产品简介.....1110
26.2.1 MaxCompute的定义.....1110
26.2.2 MaxCompute功能组件.....1111
26.3 快速开始.....1112
26.3.1 配置客户端（Console）.....1112
26.3.2 添加删除用户.....1113
26.3.3 用户授权及权限查看.....1114
26.3.3.1 ACL授权.....1114
26.3.3.2 Policy授权.....1115
26.3.3.3 查看权限.....1116
26.3.4 角色创建及授权.....1116
26.3.5 创建删除表.....1117
26.3.5.1 创建表.....1117
26.3.5.2 获取表信息.....1118
26.3.5.3 删除表.....1118
26.3.6 导入导出数据.....1118
26.3.6.1 运行SQL.....1118
26.3.6.1.1 Select语句.....1119
26.3.6.1.2 Insert语句.....1119
26.3.6.1.3 Join操作.....1120
26.3.6.1.4 其他.....1120
26.3.7 编写并使用UDF.....1120
26.3.7.1 UDF示例.....1120
26.3.7.2 UDAF示例.....1121
26.3.7.3 UDTF示例.....1122

26.3.8 编写并运行MapReduce.....1122
26.3.9 编写并运行Graph.....1123
26.4 基本介绍.....1125
26.4.1 基本概念.....1125
26.4.2 常用命令.....1130
26.4.2.1 常用命令简介.....1130
26.4.2.2 项目空间操作.....1130
26.4.2.3 表操作.....1132
26.4.2.4 实例操作.....1135
26.4.2.5 资源操作.....1137
26.4.2.6 函数操作.....1139
26.4.2.7 Tunnel操作.....1140
26.4.2.8 其他操作.....1145
26.5 SQL使用指南.....1148
26.5.1 概要.....1148
26.5.1.1 使用场景.....1148
26.5.1.2 保留字.....1148
26.5.1.3 分区表.....1149
26.5.1.4 类型转换.....1149
26.5.1.4.1 显式类型转换.....1149
26.5.1.4.2 隐式类型转换及其作用域.....1150
26.5.1.4.3 SQL内建函数.....1153
26.5.1.4.4 CASE WHEN.....1153
26.5.1.4.5 分区列.....1154
26.5.1.4.6 UNION ALL.....1154
26.5.1.4.7 String类型与Datetime类型之间的转换.....1154
26.5.2 运算符.....1155
26.5.2.1 关系操作符.....1155
26.5.2.2 算术操作符.....1156
26.5.2.3 位运算操作符.....1157
26.5.2.4 逻辑操作符.....1157
26.5.3 DDL语句.....1158
26.5.3.1 表操作.....1158
26.5.3.1.1 创建表（CREATE TABLE）.....1158
26.5.3.1.2 删除表（DROP TABLE）.....1160
26.5.3.1.3 重命名表（RENAME TABLE）.....1160
26.5.3.1.4 修改表的注释.....1161
26.5.3.1.5 修改表的生命周期属性.....1161
26.5.3.1.6 禁止生命周期.....1162
26.5.3.1.7 修改表的修改时间.....1162
26.5.3.1.8 清空非分区表里的数据.....1162

26.5.3.1.9 备份表的数据.....1163
26.5.3.1.10 强制删除表数据（分区数据）.....1163
26.5.3.2 视图操作.....1164
26.5.3.2.1 创建视图（CREATE VIEW）.....1164
26.5.3.2.2 删除视图（DROP VIEW）.....1165
26.5.3.2.3 重命名视图（RENAME VIEW）.....1165
26.5.3.3 列及分区操作.....1165
26.5.3.3.1 添加分区（ADD PARTITION）.....1165
26.5.3.3.2 删除分区（DROP PARTITION）.....1166
26.5.3.3.3 添加列.....1166
26.5.3.3.4 修改列名.....1167
26.5.3.3.5 修改列、分区注释.....1167
26.5.3.3.6 修改分区的修改时间.....1167
26.5.3.3.7 修改分区值.....1168
26.5.4 DML语句.....1168
26.5.4.1 Insert语句.....1168
26.5.4.1.1 更新表中的数据（INSERT OVERWRITE|INTO）.....1168
26.5.4.1.2 多路输出（MULTI INSERT）.....1169
26.5.4.1.3 输出到动态分区（DYNAMIC PARTITION）.....1170
26.5.4.2 Select语句.....1171
26.5.4.2.1 SELECT操作.....1171
26.5.4.2.2 子查询.....1174
26.5.4.3 Union语句.....1174
26.5.4.3.1 UNION ALL.....1174
26.5.4.4 Join语句.....1175
26.5.4.4.1 JOIN操作.....1175
26.5.4.4.2 MAPJOIN HINT.....1176
26.5.4.5 Explain语句.....1177
26.5.5 内建函数.....1180
26.5.5.1 数学运算函数.....1180
26.5.5.1.1 ABS.....1180
26.5.5.1.2 ACOS.....1181
26.5.5.1.3 ASIN.....1181
26.5.5.1.4 ATAN.....1182
26.5.5.1.5 CEIL.....1182
26.5.5.1.6 CONV.....1183
26.5.5.1.7 COS.....1183
26.5.5.1.8 COSH.....1184
26.5.5.1.9 COT.....1184
26.5.5.1.10 EXP.....1184
26.5.5.1.11 FLOOR.....1185
26.5.5.1.12 LN.....1185

26.5.5.1.13 LOG....1186
26.5.5.1.14 POW....1186
26.5.5.1.15 RAND....1186
26.5.5.1.16 ROUND....1187
26.5.5.1.17 SIN....1187
26.5.5.1.18 SINH....1188
26.5.5.1.19 SQRT....1188
26.5.5.1.20 TAN....1188
26.5.5.1.21 TANH....1189
26.5.5.1.22 TRUNC....1189
26.5.5.1.23 新扩展数字函数说明....1190
26.5.5.1.24 LOG2....1190
26.5.5.1.25 LOG10....1191
26.5.5.1.26 BIN....1191
26.5.5.1.27 HEX....1191
26.5.5.1.28 UNHEX....1192
26.5.5.1.29 RADIANS....1192
26.5.5.1.30 DEGREES....1193
26.5.5.1.31 SIGN....1193
26.5.5.1.32 E....1194
26.5.5.1.33 PI....1194
26.5.5.1.34 FACTORIAL....1194
26.5.5.1.35 CBRT....1195
26.5.5.1.36 SHIFTLEFT....1195
26.5.5.1.37 SHIFTRIGHT....1196
26.5.5.1.38 SHIFTRIGHTUNSIGNED....1196
26.5.5.2 字符串处理函数....1197
26.5.5.2.1 CHAR_MATCHCOUNT....1197
26.5.5.2.2 CHR....1197
26.5.5.2.3 CONCAT....1197
26.5.5.2.4 INSTR....1198
26.5.5.2.5 IS_ENCODING....1199
26.5.5.2.6 KEYVALUE....1199
26.5.5.2.7 LENGTH....1200
26.5.5.2.8 LENGTHB....1201
26.5.5.2.9 MD5....1201
26.5.5.2.10 PARSE_URL....1201
26.5.5.2.11 REGEXP_EXTRACT....1202
26.5.5.2.12 REGEXP_INSTR....1203
26.5.5.2.13 REGEXP_SUBSTR....1203
26.5.5.2.14 REGEXP_COUNT....1204
26.5.5.2.15 SPLIT_PART....1204
26.5.5.2.16 REGEXP_REPLACE....1205
26.5.5.2.17 SUBSTR....1206

26.5.5.2.18 TOLOWER..... 1207
26.5.5.2.19 TOUPPER..... 1207
26.5.5.2.20 TO_CHAR..... 1208
26.5.5.2.21 TRIM..... 1208
26.5.5.2.22 LTRIM..... 1208
26.5.5.2.23 RTRIM..... 1209
26.5.5.2.24 REVERSE..... 1209
26.5.5.2.25 SPACE..... 1210
26.5.5.2.26 REPEAT..... 1210
26.5.5.2.27 ASCII..... 1211
26.5.5.2.28 URL_ENCODE..... 1211
26.5.5.2.29 URL_DECODE..... 1212
26.5.5.2.30 新扩展字符串函数说明..... 1213
26.5.5.2.31 CONCAT_WS..... 1213
26.5.5.2.32 LPAD..... 1213
26.5.5.2.33 RPAD..... 1214
26.5.5.2.34 REPLACE..... 1214
26.5.5.2.35 SOUNDEX..... 1215
26.5.5.2.36 SUBSTRING_INDEX..... 1215
26.5.5.3 日期处理函数..... 1216
26.5.5.3.1 DATEADD..... 1216
26.5.5.3.2 DATEDIFF..... 1217
26.5.5.3.3 DATEPART..... 1218
26.5.5.3.4 DATETRUNC..... 1218
26.5.5.3.5 FROM_UNIXTIME..... 1219
26.5.5.3.6 GETDATE..... 1219
26.5.5.3.7 ISDATE..... 1219
26.5.5.3.8 LASTDAY..... 1220
26.5.5.3.9 TO_DATE..... 1220
26.5.5.3.10 TO_CHAR..... 1221
26.5.5.3.11 UNIX_TIMESTAMP..... 1222
26.5.5.3.12 WEEKDAY..... 1222
26.5.5.3.13 WEEKOFYEAR..... 1222
26.5.5.3.14 新扩展日期函数说明..... 1223
26.5.5.3.15 YEAR..... 1224
26.5.5.3.16 QUARTER..... 1224
26.5.5.3.17 MONTH..... 1224
26.5.5.3.18 DAY..... 1225
26.5.5.3.19 DAYOFMONTH..... 1225
26.5.5.3.20 HOUR..... 1226
26.5.5.3.21 MINUTE..... 1226
26.5.5.3.22 SECOND..... 1226
26.5.5.3.23 FROM_UTC_TIMESTAMP..... 1227
26.5.5.3.24 CURRENT_TIMESTAMP..... 1227

26.5.5.3.25 ADD_MONTHS..... 1228
26.5.5.3.26 LAST_DAY..... 1228
26.5.5.3.27 NEXT_DAY..... 1229
26.5.5.3.28 MONTHS_BETWEEN..... 1229
26.5.5.4 窗口函数..... 1230
26.5.5.4.1 COUNT..... 1230
26.5.5.4.2 AVG..... 1231
26.5.5.4.3 MAX..... 1232
26.5.5.4.4 MIN..... 1232
26.5.5.4.5 MEDIAN..... 1233
26.5.5.4.6 STDDEV..... 1233
26.5.5.4.7 STDDEV_SAMP..... 1234
26.5.5.4.8 SUM..... 1235
26.5.5.4.9 DENSE_RANK..... 1235
26.5.5.4.10 RANK..... 1236
26.5.5.4.11 LAG..... 1238
26.5.5.4.12 LEAD..... 1238
26.5.5.4.13 PERCENT_RANK..... 1239
26.5.5.4.14 ROW_NUMBER..... 1239
26.5.5.4.15 CLUSTER_SAMPLE..... 1240
26.5.5.5 聚合函数..... 1242
26.5.5.5.1 COUNT..... 1242
26.5.5.5.2 AVG..... 1243
26.5.5.5.3 MAX..... 1244
26.5.5.5.4 MIN..... 1244
26.5.5.5.5 MEDIAN..... 1245
26.5.5.5.6 STDDEV..... 1245
26.5.5.5.7 STDDEV_SAMP..... 1245
26.5.5.5.8 SUM..... 1246
26.5.5.5.9 WM_CONCAT..... 1246
26.5.5.5.10 新扩展聚合函数说明..... 1247
26.5.5.5.11 COLLECT_LIST..... 1247
26.5.5.5.12 COLLECT_SET..... 1247
26.5.5.6 其他函数..... 1248
26.5.5.6.1 ARRAY..... 1248
26.5.5.6.2 ARRAY_CONTAINS..... 1248
26.5.5.6.3 CAST..... 1249
26.5.5.6.4 COALESCE..... 1249
26.5.5.6.5 DECODE..... 1249
26.5.5.6.6 EXPLODE..... 1251
26.5.5.6.7 GET_IDCARD_AGE..... 1251
26.5.5.6.8 GET_IDCARD_BIRTHDAY..... 1252
26.5.5.6.9 GET_IDCARD_SEX..... 1252
26.5.5.6.10 GREATEST..... 1252

26.5.5.6.11 INDEX....1253
26.5.5.6.12 MAX_PT....1254
26.5.5.6.13 ORDINAL....1254
26.5.5.6.14 LEAST....1255
26.5.5.6.15 SIZE....1255
26.5.5.6.16 SPLIT....1256
26.5.5.6.17 STR_TO_MAP....1256
26.5.5.6.18 UNIQUE_ID....1257
26.5.5.6.19 UUID....1257
26.5.5.6.20 SAMPLE....1257
26.5.5.6.21 CASE WHEN表达式....1258
26.5.5.6.22 IF....1259
26.5.5.6.23 新扩展其他函数说明....1259
26.5.5.6.24 MAP....1259
26.5.5.6.25 MAP_KEYS....1260
26.5.5.6.26 MAP_VALUES....1260
26.5.5.6.27 SORT_ARRAY....1261
26.5.5.6.28 POSEXPLODE....1261
26.5.5.6.29 STRUCT....1262
26.5.5.6.30 NAMED_STRUCT....1262
26.5.5.6.31 INLINE....1263
26.5.5.6.32 BETWEEN AND 表达式....1263
26.5.5.7 分词函数....1264
26.5.5.7.1 UNIFORM....1264
26.5.5.7.2 UDF_NORMALIZE....1265
26.5.5.7.3 SEGMENT....1265
26.5.5.7.4 SYNONYM....1265
26.5.6 UDF....1266
26.5.6.1 概要....1266
26.5.6.1.1 参数与返回值类型....1266
26.5.6.2 UDF....1268
26.5.6.3 UDAF....1269
26.5.6.4 UDTF....1271
26.5.6.4.1 概要说明....1271
26.5.6.4.2 UDTF使用说明....1273
26.5.6.5 Python UDF....1275
26.5.6.5.1 受限环境....1275
26.5.6.5.2 第三方库....1277
26.5.6.5.3 参数与返回值类型....1277
26.5.6.5.4 UDF....1278
26.5.6.5.5 UDAF....1279
26.5.6.5.6 UDTF....1279
26.5.6.5.7 引用资源....1280

26.5.7 附录..... 1281
26.5.7.1 转义字符..... 1281
26.5.7.2 LIKE字符匹配..... 1282
26.5.7.3 正则表达式规范..... 1282
26.5.7.4 保留字..... 1284
26.5.7.5 限制项汇总..... 1285
26.5.7.6 常见问题及使用建议..... 1286
26.5.7.6.1 数据倾斜..... 1286
26.5.7.6.1.1 group by倾斜..... 1287
26.5.7.6.1.2 distribute by倾斜..... 1287
26.5.7.6.1.3 join倾斜..... 1287
26.5.7.6.1.4 muti-distinct倾斜..... 1288
26.5.7.6.1.5 误用动态分区导致的数据倾斜..... 1288
26.5.7.6.2 Quota及资源使用..... 1288
26.5.7.6.3 MaxCompute储存优化技巧..... 1290
26.5.7.6.4 UDF OOM问题..... 1292
26.5.7.7 MaxCompute常用SQL参数设置..... 1292
26.5.7.7.1 Map设置..... 1292
26.5.7.7.2 Join设置..... 1293
26.5.7.7.3 Reduce设置..... 1293
26.5.7.7.4 UDF设置..... 1294
26.5.7.7.5 MapJoin设置..... 1294
26.5.7.7.6 数据倾斜设置..... 1295
26.6 Tunnel使用指南..... 1295
26.6.1 Tunnel SDK介绍..... 1295
26.6.1.1 TableTunnel..... 1296
26.6.1.2 UploadSession..... 1296
26.6.1.3 DownloadSession..... 1298
26.6.2 Tunnel SDK示例..... 1299
26.6.2.1 简单上传示例..... 1299
26.6.2.2 简单下载示例..... 1300
26.6.2.3 多线程上传示例..... 1301
26.6.2.4 多线程下载示例..... 1303
26.6.3 附录..... 1305
26.6.3.1 Tunnel上传下载常见问题..... 1305
26.6.3.2 Tunnel常见错误码..... 1307
26.7 MapReduce使用指南..... 1309
26.7.1 概要..... 1309
26.7.1.1 MapReduce..... 1309
26.7.1.2 扩展MapReduce..... 1310
26.7.2 功能介绍..... 1311

26.7.2.1 运行命令 1311
26.7.2.2 基本概念 1312
26.7.2.2.1 Map/Reduce 1312
26.7.2.2.2 排序 1313
26.7.2.2.3 哈希 1313
26.7.2.2.4 归并 1313
26.7.2.2.5 输入与输出 1313
26.7.2.2.6 资源读取 1313
26.7.2.2.7 本地运行 1314
26.7.3 SDK介绍 1317
26.7.3.1 核心接口 1317
26.7.3.1.1 MapperBase 1317
26.7.3.1.2 ReducerBase 1318
26.7.3.1.3 TaskContext 1318
26.7.3.1.4 JobConf 1319
26.7.3.1.5 JobClient 1320
26.7.3.1.6 RunningJob 1320
26.7.3.1.7 InputUtils 1321
26.7.3.1.8 OutputUtils 1321
26.7.3.1.9 Pipeline 1321
26.7.3.1.10 数据类型 1322
26.7.4 应用限制 1323
26.7.5 示例程序 1324
26.7.5.1 WordCount示例 1324
26.7.5.2 MapOnly示例 1325
26.7.5.3 多路输入输出示例 1327
26.7.5.4 多任务示例 1329
26.7.5.5 二次排序示例 1331
26.7.5.6 使用资源示例 1332
26.7.5.7 使用counter示例 1334
26.7.5.8 grep示例 1335
26.7.5.9 join示例 1338
26.7.5.10 sleep示例 1340
26.7.5.11 unique示例 1343
26.7.5.12 sort示例 1345
26.7.5.13 分区表输入示例 1346
26.7.5.14 pipeline示例 1347
26.8 Graph使用指南 1349
26.8.1 Graph概要 1349
26.8.1.1 Graph数据结构 1349
26.8.1.2 Graph程序逻辑 1350
26.8.1.2.1 加载图 1350

26.8.1.2.2 迭代计算.....1351
26.8.1.2.3 迭代终止.....1351
26.8.2 Graph功能介绍.....1352
26.8.2.1 运行作业.....1352
26.8.2.2 输入输出.....1353
26.8.2.3 读取资源.....1354
26.8.2.3.1 Graph程序中添加资源.....1354
26.8.2.3.2 Graph程序中使用资源.....1354
26.8.3 Graph SDK介绍.....1355
26.8.4 开发和调试.....1356
26.8.4.1 开发示例.....1356
26.8.4.2 本地调试.....1357
26.8.4.3 本地作业临时目录.....1359
26.8.4.4 集群调试.....1360
26.8.4.5 性能调优.....1360
26.8.4.5.1 配置作业参数.....1360
26.8.4.5.2 运用Combiner.....1362
26.8.4.5.3 减少数据输入量.....1362
26.8.4.5.4 内置jar包.....1362
26.8.5 应用限制.....1363
26.8.6 示例程序.....1363
26.8.6.1 单源最短距离.....1363
26.8.6.2 PageRank.....1365
26.8.6.3 K-均值聚类.....1367
26.8.6.4 BiPartiteMatchiing.....1372
26.8.6.5 强连通分量.....1374
26.8.6.6 连通分量.....1381
26.8.6.7 拓扑排序.....1383
26.8.6.8 线性回归.....1385
26.8.6.9 三角形计数.....1389
26.8.6.10 GraphLoader.....1391
26.9 Java SDK使用指南.....1396
26.9.1 SDK介绍.....1396
26.9.2 AliyunAccount.....1397
26.9.3 Odps.....1397
26.9.4 Projects.....1398
26.9.5 Project.....1398
26.9.6 SQLTask.....1398
26.9.7 Instances.....1399
26.9.8 Instance.....1399
26.9.9 Tables.....1400
26.9.10 Table.....1400

26.9.11 Resources....1400
26.9.12 Resource....1401
26.9.13 Functions....1402
26.9.14 Function....1402
26.10 Java沙箱限制介绍....1403
26.11 Spark使用指南....1406
26.11.1 概要....1406
26.11.2 项目资源....1407
26.11.3 环境设置....1407
26.11.3.1 解压Spark on MaxCompute发布包....1407
26.11.3.2 设置环境变量....1407
26.11.3.3 设置Spark-defaults.conf....1408
26.11.4 常用场景案例演示....1408
26.11.4.1 WordCount案例....1408
26.11.4.2 SparkPi案例....1410
26.11.4.3 MaxCompute Table Rdd案例....1411
26.11.4.4 MaxCompute Table Spark-SQL案例....1415
26.11.4.5 MaxCompute自研Client模式案例....1416
26.11.4.6 MaxCompute Table PySpark案例....1417
26.11.5 Maven依赖....1419
26.11.6 特殊说明....1420
26.11.6.1 Streaming任务....1420
26.11.6.2 Tracking Url....1420
26.11.7 Base支持....1421
26.12 非结构化数据处理使用指南....1423
26.12.1 如何在MaxCompute上访问并处理OSS非结构化数据....1423
26.12.1.1 前言....1423
26.12.1.2 使用内置extractor读取OSS数据....1423
26.12.1.2.1 创建外部表....1424
26.12.1.2.2 查询外部表....1424
26.12.1.3 自定义extractor....1425
26.12.1.3.1 定义StorageHandler....1425
26.12.1.3.2 定义Extractor....1426
26.12.1.3.3 编译打包....1427
26.12.1.3.4 创建外部表....1427
26.12.1.3.5 查询外部表....1427
26.12.1.4 进阶使用....1428
26.12.1.4.1 通过自定义extractor读取外部非结构化数据....1428
26.12.1.4.2 通过STS访问OSS....1430
26.12.1.5 数据的分区....1431
26.12.1.5.1 分区数据在OSS上的标准组织方式和路径格式....1431

26.12.1.5.2 分区数据在OSS上的自定义路径.....1433
26.12.1.5.3 完全自定义的非分区数据子集访问.....1434
26.12.1.6 非结构化数据的输出.....1434
26.12.1.6.1 创建External Table.....1434
26.12.1.6.2 通过对External Table的INSERT操作实现TSV文本文件的写出.....1435
26.12.1.6.3 通过对External Table的INSERT操作实现非结构化文件的写出.....1436
26.12.1.6.4 以MaxCompute为计算介质，实现不同存储介质之间的数据转移.....1437
26.12.2 如何在MaxCompute上访问并处理TableStore（OTS）数据.....1438
26.12.2.1 前言.....1438
26.12.2.2 MaxCompute对TableStore数据进行读取和计算.....1438
26.12.2.2.1 使用前提和假设.....1438
26.12.2.2.2 创建External Table.....1439
26.12.2.2.3 通过External Table访问TableStore数据进行计算.....1439
26.12.2.2.4 使用STS/RAM的方式访问TableStore数据.....1440
26.12.2.3 数据从MaxCompute写出到TableStore.....1441
26.13 安全指南.....1442
26.13.1 目标用户.....1442
26.13.2 快速开始.....1442
26.13.3 用户认证.....1445
26.13.4 项目空间的用户及授权管理.....1445
26.13.4.1 概要.....1445
26.13.4.2 用户管理.....1445
26.13.4.3 角色管理.....1446
26.13.4.4 ACL授权.....1447
26.13.4.5 权限查看.....1449
26.13.5 跨项目空间的资源分享.....1450
26.13.5.1 Package使用方法.....1451
26.13.5.1.1 Package创建者相关操作.....1451
26.13.5.1.2 Package使用者相关操作.....1452
26.13.6 项目空间的数据保护.....1453
26.13.6.1 数据保护机制.....1453
26.13.6.2 开启数据保护后的数据流出方法.....1454
26.13.6.3 资源分享与数据保护.....1456
26.13.7 项目空间的安全配置.....1456
26.13.8 Policy权限策略.....1457
26.13.8.1 Policy概况.....1457
26.13.8.2 Policy基本术语.....1459
26.13.8.3 访问策略语言结构.....1460
26.13.8.3.1 授权语句（Statement）结构.....1460
26.13.8.3.2 条件块结构.....1461
26.13.8.3.3 条件操作类型.....1461

26.13.8.3.4 条件关键字..... 1462
26.13.8.4 访问策略语言规范.....1462
26.13.8.4.1 Principal命名规范..... 1462
26.13.8.4.2 Resource命名规范.....1463
26.13.8.4.3 Action命名规范..... 1463
26.13.8.4.4 Condition Keys命名规范.....1464
26.13.8.4.5 访问策略样例.....1464
26.13.8.5 Policy与ACL的区别..... 1465
26.13.8.6 应用限制..... 1466
26.13.9 安全相关语句汇总.....1466
26.13.9.1 项目空间的安全配置..... 1466
26.13.9.2 项目空间的权限管理..... 1467
26.13.9.3 基于Package的资源分享..... 1468
26.14 工具.....1469
26.14.1 MaxCompute客户端..... 1469
26.14.1.1 安装客户端..... 1469
26.14.1.2 配置及参数说明..... 1470
26.14.2 Eclipse开发插件..... 1474
26.14.2.1 安装Eclipse.....1474
26.14.2.2 创建工程..... 1477
26.14.2.2.1 方式一.....1477
26.14.2.2.2 方式二.....1480
26.14.2.3 MapReduce开发插件介绍..... 1482
26.14.2.3.1 快速运行WordCount示例.....1482
26.14.2.3.2 运行自定义MapReduce程序.....1485
26.14.2.4 UDF开发插件介绍..... 1498
26.14.2.4.1 简介..... 1498
26.14.2.4.2 Local Debug UDF程序..... 1498
26.14.2.4.2.1 菜单栏运行..... 1498
26.14.2.4.2.2 右键单击快速运行.....1500
26.14.2.4.3 运行用户自定义UDF程序.....1502
26.14.2.5 Graph开发插件介绍.....1504
27 大数据开发套件.....1510
27.1 产品概述..... 1510
27.2 大数据开发平台入门..... 1512
27.2.1 准备工作..... 1512
27.2.2 进入阿里云大数据开发平台..... 1515
27.2.3 功能索引..... 1517
27.3 快速开始..... 1518
27.3.1 新建任务..... 1518
27.3.2 测试任务..... 1528

27.3.3 编辑任务.....1532
27.3.4 复制任务.....1532
27.3.5 任务监控告警配置.....1534
27.3.6 删除任务.....1536
27.3.7 新建脚本文件.....1537
27.3.8 删除脚本文件.....1539
27.3.9 添加资源.....1540
27.3.10 删除资源.....1542
27.3.11 注册函数.....1542
27.3.12 注销函数.....1546
27.3.13 导入本地数据.....1547
27.3.14 发布任务.....1550
27.3.15 查看MaxCompute项目名称.....1552
27.3.16 查看数据源配置.....1553
27.4 数据开发.....1553
27.4.1 概述.....1553
27.4.2 权限点划分.....1553
27.4.3 工作流任务设计器.....1555
27.4.3.1 设计器介绍.....1555
27.4.3.2 工作流任务属性配置.....1559
27.4.3.3 节点属性配置.....1564
27.4.4 单节点任务编辑器.....1565
27.4.5 MaxCompute代码编辑器.....1565
27.4.6 系统调度参数.....1569
27.4.7 配置数据同步节点.....1573
27.4.7.1 选择数据来源和目标.....1574
27.4.7.2 字段配置.....1577
27.4.7.3 数据抽取和加载控制.....1579
27.4.7.4 流量与出错控制.....1582
27.4.8 本地数据导入.....1582
27.4.9 跨项目发布.....1582
27.5 数据管理.....1582
27.5.1 概述.....1583
27.5.2 权限点划分.....1583
27.5.3 全局概览.....1584
27.5.4 数据搜索—找表.....1585
27.5.5 申请数据权限.....1586
27.5.6 管理数据表.....1591
27.5.6.1 新建表.....1592
27.5.6.2 收藏表.....1599
27.5.6.3 修改生命周期.....1600

27.5.6.4 修改表结构..... 1602
27.5.6.5 隐藏表..... 1604
27.5.6.6 修改表负责人..... 1605
27.5.6.7 删除表..... 1606
27.5.7 查看表详情..... 1607
27.5.8 权限审批与收回..... 1612
27.5.9 配置类目导航..... 1612
27.6 运维中心..... 1614
27.6.1 概述..... 1614
27.6.2 使用权限说明..... 1614
27.6.2.1 角色的权限点划分..... 1615
27.6.2.2 如果您是一个开发人员..... 1617
27.6.2.3 如果您是一个部署人员..... 1617
27.6.2.4 如果您是一个运维人员..... 1617
27.6.2.5 如果您是一个项目管理员..... 1618
27.6.3 任务管理..... 1618
27.6.4 任务运维..... 1620
27.6.4.1 周期任务运维..... 1620
27.6.4.2 查看执行日志..... 1622
27.6.5 监控报警..... 1623
27.7 组织管理..... 1623
27.8 项目管理..... 1624
28 分析型数据库..... 1625
28.1 产品概述..... 1625
28.2 数据库、表组和表介绍..... 1627
28.3 数据类型介绍..... 1629
28.3.1 支持的数据类型..... 1629
28.3.2 与mysql数据类型对比..... 1630
28.4 特色功能..... 1630
28.5 ECU详解..... 1631
28.6 快速开始..... 1632
28.6.1 开通服务..... 1632
28.6.2 创建数据库..... 1632
28.6.3 分配和配置数据库..... 1634
28.6.4 使用DMS创建和管理表..... 1636
28.6.5 导入数据..... 1638
28.6.6 在应用中连接并使用..... 1640
28.6.7 新增用户和授权..... 1642
28.7 DDL..... 1643
28.7.1 创建数据库与配置..... 1643
28.7.2 创建和修改表组..... 1643

28.7.3 创建和修改表.....1643
28.7.4 列的类型和属性.....1646
28.7.5 索引.....1647
28.7.6 ECU管理.....1647
28.8 DML.....1648
28.8.1 Select语句基本语法.....1648
28.8.2 逻辑表达式和特殊语法.....1655
28.8.3 INSERT/DELETE命令.....1658
28.8.4 多计算引擎和Hint.....1659
28.9 Data Pipeline.....1661
28.9.1 导入数据.....1661
28.9.2 数据导入状态查询.....1662
28.9.3 海量数据导出.....1663
28.10 用户与权限.....1665
28.10.1 用户类型与用户管理.....1665
28.10.2 分析型数据库权限模型.....1665
28.10.3 使用ACL进行权限管理.....1667
28.10.3.1 权限回收.....1668
28.10.3.2 查看用户权限.....1668
28.10.3.3 其他相关内容.....1668
28.11 性能优化和诊断.....1669
28.11.1 使用执行计划调优.....1669
28.11.1.1 explain语句.....1669
28.11.1.2 返回文本格式.....1670
28.11.1.3 Explain逻辑计划详细语义.....1670
28.11.1.4 Explain物理执行计划详细语义.....1672
28.11.2 查看DB性能指标和报告.....1674
28.12 在生产中使用分析型数据库.....1675
28.12.1 业务系统连接并进行查询.....1675
28.12.2 数据导入任务生产指南.....1677
28.12.3 数据模型设计.....1678
28.12.3.1 事实表.....1678
28.12.3.2 维度表.....1679
28.12.3.3 宽表.....1679
28.12.3.4 聚合表.....1680
28.12.3.5 实例分析.....1680
28.12.4 在BI工具中进行连接和使用.....1683
28.13 附录.....1688
28.13.1 附录一：元数据库数据字典.....1688
28.13.1.1 performance_schema库.....1689
28.13.1.2 information_schema库.....1690

28.13.2 附录二：SQL函数表 1697
28.13.2.1 特色函数 1697
28.13.2.2 字符串函数 1701
28.13.2.3 日期函数 1701
28.13.2.4 其他函数 1702
28.13.3 附录三：常见错误码表 1702
29 流计算 1711
29.1 产品概述 1711
29.1.1 什么是流计算 1711
29.1.2 全链路流计算 1712
29.1.3 流计算和批量计算对比 1713
29.1.3.1 批量计算 1713
29.1.3.2 流式计算 1714
29.1.3.3 模型对比 1715
29.2 产品说明 1716
29.2.1 产品历程 1716
29.2.2 产品定位 1717
29.2.3 产品术语 1718
29.2.4 产品特点 1719
29.2.5 产品安全 1721
29.2.5.1 账号安全 1721
29.2.5.2 业务安全 1721
29.2.5.3 数据安全 1722
29.2.6 业务流程 1722
29.2.6.1 数据链路情况 1724
29.2.7 应用场景 1725
29.2.8 应用案例 1726
29.2.8.1 天猫双十一大屏 1726
29.2.8.2 无线数据分析 1728
29.2.8.3 电商运营活动 1730
29.2.8.4 物联网设备监控 1731
29.3 使用准备 1732
29.3.1 开通流计算服务 1732
29.3.2 账号管理 1732
29.3.2.1 账号模型介绍 1732
29.3.2.2 主子账号使用流程 1733
29.3.2.3 对子账号分权 1736
29.3.3 自动化角色授权流程 1736
29.4 热词统计 1738
29.4.1 代码开发 1739
29.4.2 代码调试 1740

29.4.3 数据运维.....1743
29.5 天猫双十一大屏.....1744
29.5.1 问题描述.....1745
29.5.2 前置准备.....1746
29.5.3 注册存储.....1748
29.5.4 开发.....1748
29.5.5 运维.....1750
29.6 WebConsole手册.....1751
29.6.1 数据采集.....1751
29.6.2 数据存储.....1753
29.6.2.1 存储概览.....1753
29.6.2.2 存储类别.....1753
29.6.2.3 存储使用.....1755
29.6.3 大数据总线（DataHub）.....1757
29.6.4 日志服务.....1758
29.6.5 分析型数据库.....1759
29.6.6 表格存储.....1760
29.6.7 云数据库.....1761
29.6.8 分布式数据库.....1763
29.6.9 消息服务.....1764
29.7 数据开发.....1766
29.7.1 开发阶段.....1766
29.7.1.1 SQL辅助.....1766
29.7.1.2 SQL版本管理.....1766
29.7.1.3 数据存储管理.....1767
29.7.1.4 资源（Jar）管理.....1767
29.7.2 调试阶段.....1768
29.7.3 上线阶段.....1771
29.7.4 参数变量.....1771
29.7.4.1 流计算任务静态构成.....1772
29.7.4.2 系统参数.....1772
29.7.5 用户参数.....1773
29.7.5.1 用户参数说明.....1773
29.7.5.2 用户参数约束.....1774
29.7.6 数据运维.....1774
29.7.6.1 作业管理.....1774
29.7.6.2 作业仪表盘.....1778
29.7.6.3 血缘关系图.....1783
29.7.6.3.1 数据拓扑.....1783
29.7.6.3.2 数据抽样.....1784
29.7.6.4 逻辑执行图.....1785

29.7.6.4.1 组件分类.....1785
29.7.6.4.2 关键路径.....1788
29.7.6.4.3 SQL和组件对应关系.....1789
29.7.6.5 作业详情.....1791
29.7.6.5.1 当前运行代码.....1792
29.7.6.5.2 历史运行记录.....1792
29.7.6.6 作业告警.....1792
29.7.6.6.1 延迟监控的原理.....1793
29.7.6.6.2 云监控告警.....1793
29.7.6.7 进阶说明.....1795
29.7.6.7.1 异常排查.....1795
29.7.6.7.1.1 常见作业错误.....1796
29.7.6.7.2 性能调优.....1797
29.7.6.7.2.1 调优指标.....1797
29.7.6.7.2.2 调优参数.....1800
29.7.6.7.2.3 调优思路.....1804
29.7.7 StreamSQL.....1808
29.7.7.1 关键字.....1809
29.7.7.2 数据类型.....1809
29.7.7.3 运算符.....1810
29.7.7.3.1 关系操作符.....1810
29.7.7.3.2 算术操作符.....1811
29.7.7.3.3 位操作符.....1812
29.7.7.3.4 逻辑操作符.....1812
29.7.7.4 DDL语法.....1812
29.7.7.4.1 创建流式源表.....1813
29.7.7.4.2 创建维表.....1814
29.7.7.4.2.1 维表通用配置.....1815
29.7.7.4.2.2 TableStore（源TABLESTORE）维表.....1815
29.7.7.4.3 创建结果表.....1816
29.7.7.4.3.1 DataHub结果表.....1817
29.7.7.4.3.2 RDS（Mysql）结果表.....1818
29.7.7.4.3.3 TABLESTORE结果表.....1818
29.7.7.4.3.4 Analytic DB 结果表.....1819
29.7.7.4.4 创建临时表.....1820
29.7.7.5 DML语法.....1820
29.7.7.6 内建函数.....1823
29.7.7.6.1 数学函数.....1823
29.7.7.6.2 日期函数.....1827
29.7.7.6.3 条件函数.....1831
29.7.7.6.4 字符串函数.....1832

29.7.7.6.5 聚合函数.....1836
29.7.7.6.6 散分函数.....1839
29.7.7.6.7 窗口函数.....1839
29.7.8 SQL最佳实践.....1839
29.7.8.1 DDL声明技巧.....1839
29.7.8.1.1 使用"数据开发"进行创建.....1839
29.7.8.1.2 DDL大小写忽略.....1840
29.7.8.1.3 支持多次声明.....1840
29.7.8.1.4 支持部分字段声明.....1840
29.7.8.2 广告效果分析.....1841
29.7.8.2.1 问题描述.....1841
29.7.8.2.2 解决方案.....1841
29.7.8.3 PV/UV统计.....1842
29.7.8.3.1 问题描述.....1842
29.7.8.3.2 解决方案.....1842
29.7.8.4 实时用户路径追踪.....1842
29.7.8.4.1 问题背景.....1842
29.7.8.4.2 解决方案.....1843
29.7.9 参数说明.....1843
29.7.9.1 资源相关.....1843
29.7.9.2 并发相关.....1843
29.7.9.3 吞吐相关.....1844
29.7.9.4 超时相关.....1844
29.7.10 错误码.....1845
29.7.10.1 910 system.....1845
29.7.10.2 911 project.....1845
29.7.10.3 912 execution.....1846
29.7.10.4 913 folder.....1846
29.7.10.5 914 file.....1847
29.7.10.6 915 node.....1848
29.7.10.7 916 resource.....1848
29.7.10.8 917 instance.....1849
29.7.10.9 918 storage.....1849
29.7.10.10 919 user.....1850
29.7.10.11 920 region.....1850
29.7.10.12 921 debug.....1851
29.7.10.13 922 AUTH.....1851
29.7.10.14 923 alarm.....1852
29.7.10.15 924 galaxy.....1852
29.7.10.16 990 sdk.....1852
30 大数据应用加速器.....1854
30.1 产品概述.....1854

30.2 标签中心 1854
30.2.1 添加云计算资源 1855
30.2.2 实体关系建模 1858
30.2.2.1 配置实体 1859
30.2.2.2 配置标签 1861
30.2.2.3 配置实体关系 1863
30.2.3 数据整合与同步 1865
30.3 整合分析 1868
30.3.1 接口调试 1869
30.3.1.1 接口架构 1869
30.3.1.2 调试界面 1870
30.3.1.3 分析语法表达 1872
30.3.2 界面配置 1873
30.3.2.1 功能介绍 1874
30.3.2.2 配置说明 1875
30.3.2.2.1 筛选条件配置 1875
30.3.2.2.2 分析结果配置 1876
30.3.2.2.3 数据导出配置 1878
30.3.3 分析应用部署 1879
30.4 规则引擎 1879
30.4.1 概览 1880
30.4.2 规则基本操作 1882
30.4.2.1 创建规则 1882
30.4.2.2 配置规则 1886
30.4.2.3 规则列表 1890
30.4.2.4 上线规则 1891
30.4.2.5 下线规则 1892
30.4.3 规则运行状态查看 1893
30.5 常见问题 1893
31 Quick BI 1896
31.1 产品概述 1896
31.2 登录 Quick BI 1896
31.3 数据建模 1898
31.3.1 管理数据源 1898
31.3.1.1 数据源列表 1899
31.3.1.2 新建数据源 1899
31.3.1.2.1 来自云数据源 1900
31.3.1.2.1.1 MaxCompute (ODPS) 数据源 1900
31.3.1.2.1.2 RDS （MySQL） 1902
31.3.1.2.1.3 RDS （SQL Server）数据源 1906
31.3.1.2.1.4 Hybird DB（Greenplum）数据源 1908

31.3.1.2.1.5 Analytic DB 数据源.....1910
31.3.1.2.2 来自ECS数据源.....1911
31.3.1.2.2.1 MySQL.....1911
31.3.1.2.2.2 SQL Server 数据源.....1913
31.3.1.3 编辑数据源.....1915
31.3.1.4 删除数据源.....1915
31.3.1.5 查询数据源.....1915
31.3.1.6 查询数据源中包含的表.....1916
31.3.1.7 查询数据源中表的详情.....1916
31.3.2 管理数据集.....1917
31.3.2.1 创建数据集.....1917
31.3.2.1.1 从数据源创建.....1917
31.3.2.1.2 MaxCompute 数据源下自定义 SQL.....1918
31.3.2.2 设置数据集默认名称.....1919
31.3.2.3 编辑数据集.....1920
31.3.2.3.1 编辑维度.....1921
31.3.2.3.2 编辑度量.....1923
31.3.2.3.3 工具按钮.....1924
31.3.2.3.4 预览数据.....1926
31.3.2.3.5 关联工作表.....1927
31.3.2.3.6 关联工作表示例.....1929
31.3.2.3.7 钻取.....1931
31.3.2.3.8 计算字段.....1939
31.3.2.3.8.1 计算字段的使用规则.....1940
31.3.2.3.8.2 计算字段示例.....1940
31.3.2.3.8.3 计算度量的类型.....1942
31.3.2.3.8.4 新建计算字段.....1942
31.3.2.4 删除数据集.....1944
31.3.2.5 重命名数据集.....1945
31.3.2.6 查询数据集.....1946
31.3.2.7 转让数据集.....1946
31.3.2.8 新建数据集文件夹.....1947
31.3.2.9 重命名数据集文件夹.....1948
31.3.2.10 设置数据集行级权限.....1948
31.4 管理仪表板.....1954
31.4.1 仪表板简介.....1954
31.4.1.1 仪表板特点.....1954
31.4.1.2 仪表板的优化和新增.....1954
31.4.1.3 数据图表类型和使用场景.....1957
31.4.1.4 数据图表的数据要素.....1958
31.4.2 进入仪表板.....1960

31.4.3 仪表板分区..... 1960
31.4.3.1 数据集选择区..... 1961
31.4.3.1.1 切换数据集..... 1961
31.4.3.1.2 搜索维度字段和度量字段..... 1962
31.4.3.2 仪表板配置区 （配置画板）..... 1963
31.4.3.2.1 选择字段..... 1963
31.4.3.2.2 启用颜色图例..... 1965
31.4.3.2.3 排序..... 1966
31.4.3.2.4 过滤字段..... 1967
31.4.3.2.5 多图联动..... 1969
31.4.3.3 仪表板展示区（画布）..... 1971
31.4.3.3.1 工具栏..... 1971
31.4.3.3.2 调整图表位置..... 1971
31.4.3.3.3 查看图表数据..... 1972
31.4.3.3.4 删除图表..... 1973
31.4.3.3.5 切换图表..... 1973
31.4.3.3.6 引导功能..... 1974
31.4.3.3.7 控件..... 1975
31.4.3.3.7.1 查询条件..... 1976
31.4.3.3.7.2 文本框..... 1984
31.4.3.3.7.3 IFRAME..... 1985
31.4.3.3.7.4 TAB..... 1985
31.4.4 创建仪表板..... 1987
31.4.4.1 线图..... 1987
31.4.4.2 柱图..... 1991
31.4.4.3 饼图..... 1997
31.4.4.4 气泡地图..... 1999
31.4.4.5 色彩地图..... 2002
31.4.4.6 交叉表..... 2004
31.4.4.7 仪表盘..... 2010
31.4.4.8 雷达图..... 2014
31.4.4.9 散点图..... 2017
31.4.4.10 漏斗图..... 2019
31.4.4.11 指标看板..... 2021
31.4.4.12 矩阵树图..... 2023
31.4.4.13 极坐标图..... 2026
31.4.4.14 词云图..... 2029
31.4.4.15 旋风漏斗图..... 2031
31.4.4.16 树图..... 2035
31.4.4.17 来源去向图..... 2046
31.4.5 查询仪表板..... 2051

31.4.6 新建仪表板文件夹.....2051
31.4.7 重命名仪表板文件夹.....2052
31.4.8 转让仪表板.....2052
31.4.9 分享仪表板.....2053
31.4.10 公开仪表板.....2054
31.5 电子表格操作.....2055
31.5.1 新建电子表格.....2056
31.5.2 切换数据集.....2057
31.5.3 搜索维度字段和度量字段.....2057
31.5.4 设置字体.....2058
31.5.5 设置对齐方式.....2059
31.5.6 设置文本格式和数字格式.....2060
31.5.7 设置样式、单元格和窗口.....2060
31.5.8 设置图片、链接和下拉框.....2061
31.5.9 设置表格格式.....2063
31.5.10 设置条件规则.....2063
31.5.11 查询电子表格.....2065
31.5.12 新建电子表格文件夹.....2065
31.5.13 重命名电子表格文件夹.....2066
31.5.14 转让电子表格.....2066
31.5.15 分享电子表格.....2067
31.5.16 公开电子表格.....2067
31.6 数据门户搭建.....2068
31.6.1 新建数据门户.....2068
31.6.2 设置模板.....2068
31.6.3 设置菜单.....2069
31.7 组织管理.....2070
31.7.1 创建组织.....2071
31.7.1.1 修改组织信息.....2072
31.7.1.2 退出组织.....2072
31.7.1.3 添加组织成员.....2073
31.7.1.4 编辑组织成员.....2075
31.7.1.5 移除组织成员.....2076
31.7.1.6 查看用户所在的群空间.....2077
31.7.1.7 查询组织成员.....2077
31.7.2 群空间管理.....2078
31.7.2.1 什么是群空间.....2078
31.7.2.2 个人空间和群空间的差异.....2078
31.7.3 新建群空间.....2078
31.7.4 修改群空间.....2079
31.7.5 退出群空间.....2080

31.7.6 设为默认群空间 2081
31.7.7 转让群空间 2081
31.7.8 删除群空间 2082
31.7.9 添加群空间成员 2082
31.7.10 编辑群空间成员 2086
31.7.11 删除群空间成员 2086
31.7.12 查询群空间成员 2087
31.8 权限管理 2088
31.8.1 数据对象管理 2088
31.8.2 行级权限 2090
31.8.3 个人空间下数据对象管理 2090
32 关系网络分析 2092
32.1 产品概述 2092
32.1.1 搜索 2092
32.1.2 关系网络 2093
32.1.3 时空分析 2094
32.2 产品优势 2095
32.3 相关概念 2095
32.4 登录系统 2097
32.5 搜索对象 2099
32.5.1 简单搜索 2100
32.5.2 高级搜索 2101
32.5.3 搜索结果 2103
32.6 关系网络 2107
32.6.1 新建分析 2107
32.6.2 新增节点 2108
32.6.3 关联反查 2110
32.6.4 群体分析 2112
32.6.5 共同邻居分析 2115
32.6.6 路径分析 2116
32.6.7 骨干分析 2120
32.6.8 血缘分析 2122
32.6.9 时序分析 2124
32.6.10 信息立方 2126
32.6.10.1 使用行为分析 2126
32.6.10.2 使用行为明细 2127
32.6.10.3 管理对象信息 2128
32.6.10.4 管理统计信息 2130
32.6.10.5 使用群集统计 2132
32.6.10.6 使用标签统计 2133
32.6.11 图区布局 2134

32.6.12 功能操作.....2145
32.6.12.1 新增节点.....2145
32.6.12.2 画布拖动.....2147
32.6.12.3 图区缩放.....2148
32.6.12.4 节点锁定.....2149
32.6.12.5 标签开关.....2152
32.6.12.6 撤销回退.....2153
32.6.12.7 图区打印.....2154
32.6.12.8 结果保存.....2155
32.6.12.9 删除所选.....2156
32.6.12.10 缩略图.....2159
32.6.13 右键操作.....2160
32.6.14 目录管理.....2175
32.6.15 协作共享.....2177
32.6.15.1 分享分发.....2177
32.6.15.2 初始文件.....2179
32.6.15.3 保存草稿.....2180
32.6.15.4 发布版本.....2182
32.6.15.5 自动合并.....2184
32.6.15.6 手动合并.....2186
32.6.15.7 右键操作.....2188
32.6.16 关系方向.....2192
32.6.17 可视化分层.....2193
32.6.18 二次过滤.....2195
32.7 地图分析.....2199
32.8 用户意见反馈.....2205
32.8.1 提交用户反馈意见.....2205
32.8.2 查看用户反馈意见回复.....2208
32.8.3 系统管理员删除评论.....2210
32.9 应用场景（具体案件模拟）.....2212
32.10 常见问题.....2221
33 采云间（DPC）.....2223
33.1 系统管理.....2223
33.1.1 系统管理概述.....2223
33.1.2 大数据首页.....2224
33.1.3 管理租户.....2225
33.1.4 管理工作空间.....2226
33.1.4.1 设置管理员.....2226
33.1.4.2 管理用户.....2227
33.1.4.3 管理数据源.....2229
33.1.4.4 管理应用.....2231

33.2 数据工厂 2233
33.2.1 ETL 开发服务简介 2233
33.2.2 进入 ETL 开发 2234
33.2.3 ETL 开发界面介绍 2235
33.2.4 管理文件 2237
33.2.4.1 公有文件和私有文件的区别 2237
33.2.4.2 新建文件 2237
33.2.4.2.1 SQL 文件 2237
33.2.4.2.2 其他脚本文件 2239
33.2.4.3 编辑公有文件 2240
33.2.4.4 编辑私有文件 2240
33.2.4.5 查看文件基本信息 2240
33.2.4.6 管理文件夹 2241
33.2.4.6.1 新建文件夹 2241
33.2.4.6.2 移动文件夹 2242
33.2.4.6.3 删除文件夹 2242
33.2.4.7 复制文件 2243
33.2.4.8 移动文件 2243
33.2.4.9 删除文件 2244
33.2.4.10 搜索文件 2244
33.2.4.11 加锁文件 2244
33.2.4.12 解锁文件 2245
33.2.4.13 查看文件版本 2245
33.2.4.14 版本比对 2246
33.2.4.15 版本回滚 2248
33.2.4.16 导出文件 2249
33.2.4.17 导入文件 2249
33.2.4.18 打开文件 2250
33.2.4.19 保存文件 2251
33.2.4.20 格式化文件 2251
33.2.4.21 语法检查 2251
33.2.4.22 编辑文件 2252
33.2.4.22.1 剪切 2252
33.2.4.22.2 复制与粘贴 2253
33.2.4.22.3 撤销与重做 2253
33.2.4.22.4 缩进和取消缩进 2253
33.2.4.22.5 全选 2254
33.2.4.22.6 转到行 2254
33.2.4.22.7 查找 2255
33.2.4.22.8 替换 2256
33.2.4.22.9 代码注释 2256

33.2.4.22.10 代码折叠.....2256
33.2.4.22.11 全屏展示.....2257
33.2.4.23 文件工具.....2257
33.2.4.23.1 文件比较.....2257
33.2.4.23.2 资源球.....2258
33.2.4.23.3 时间函数.....2258
33.2.5 提交 SQL 运行.....2259
33.2.6 设置常用数据源.....2259
33.2.7 终止 SQL 运行.....2260
33.2.8 SQL 变量替换.....2260
33.2.9 查看运行日志.....2261
33.2.10 查看运行结果.....2262
33.2.11 进行结果集操作.....2262
33.2.12 管理数据字典.....2264
33.2.12.1 选择 Project.....2264
33.2.12.2 浏览表.....2265
33.2.12.3 搜索表.....2266
33.2.12.4 查看表详细信息.....2266
33.2.12.5 收藏表.....2269
33.2.13 管理运行记录.....2269
33.2.13.1 查看运行记录.....2269
33.2.13.2 搜索运行记录.....2270
33.2.13.3 打开运行记录.....2270
33.2.13.4 刷新运行记录.....2271
33.2.13.5 删除运行记录.....2271
33.2.13.6 复制运行记录.....2272
33.2.14 管理 MaxCompute 的 UDF.....2272
33.2.14.1 创建 UDF.....2272
33.2.14.2 修改 UDF.....2273
33.2.14.3 查看 UDF.....2273
33.2.14.4 删除 UDF.....2274
33.2.14.5 使用 UDF.....2275
33.2.15 上传本地数据文件.....2275
33.2.16 上传状态和日志.....2276
33.2.17 设置编辑区.....2277
33.2.18 设置脚本格式化.....2278
33.2.19 数据安全.....2278
33.2.19.1 设置最大显示行数.....2278
33.2.19.2 设置隐私数据展示策略.....2279
33.2.19.3 设置结果集下载策略.....2280
33.2.20 设置个人偏好.....2280

33.2.21 FAQ..... 2281
33.3 任务管理..... 2281
33.3.1 任务调度服务简介..... 2281
33.3.2 进入任务管理..... 2282
33.3.3 管理任务..... 2283
33.3.3.1 任务管理界面介绍..... 2283
33.3.3.2 新建同步任务..... 2284
33.3.3.2.1 MySQL 与 Maxcompute 之间的数据同步（单表模式）..... 2285
33.3.3.2.2 MySQL 与 Maxcompute 之间的数据同步（多表模式）..... 2287
33.3.3.2.3 OSS 与 Maxcompute 之间的数据同步..... 2292
33.3.3.2.4 Table Store 与 Maxcompute 之间的数据同步..... 2297
33.3.3.2.5 Oceanbase 与 Maxcompute 之间的数据同步（单表模式）..... 2303
33.3.3.2.6 Oceanbase 与 Maxcompute 之间的数据同步（多表模式）..... 2305
33.3.3.2.7 SQL Server 与 Maxcompute 之间的数据同步..... 2306
33.3.3.3 新建刷新任务..... 2306
33.3.3.4 新建虚任务..... 2308
33.3.3.5 新建检测任务..... 2311
33.3.3.6 触发接口..... 2312
33.3.3.7 搜索任务..... 2313
33.3.3.8 上线任务..... 2314
33.3.3.9 下线任务..... 2316
33.3.3.10 删除任务..... 2318
33.3.3.11 任务执行补数据..... 2318
33.3.3.12 查看任务操作日志..... 2319
33.3.3.13 修改根节点启动时间..... 2320
33.3.3.14 导出任务..... 2321
33.3.3.15 导入任务..... 2322
33.3.3.16 跨周期依赖..... 2323
33.3.3.17 业务类参数替换..... 2325
33.3.3.17.1 通用参数替换..... 2325
33.3.3.17.2 Shell 的特殊处理..... 2325
33.3.3.18 系统上下文..... 2326
33.3.3.18.1 上下文的写入与读取..... 2326
33.3.3.18.2 根据 dataDate 进行加减计算..... 2326
33.3.3.18.3 根据定时的表达式进行加减计算..... 2327
33.3.4 监控任务..... 2328
33.3.4.1 任务监控界面介绍..... 2328
33.3.4.2 查看任务执行历史..... 2329
33.3.4.3 查看同步任务日志..... 2330
33.3.4.4 查看刷新任务日志..... 2332
33.3.4.5 查看检测任务日志..... 2332

33.3.4.6 重跑任务.....2332
33.3.4.7 查看补数据任务状态.....2333
33.3.4.8 导出运行日志.....2335
33.3.4.9 跳过任务.....2337
33.3.4.10 同步最新更改.....2339
33.3.4.11 查看未执行的诊断.....2340
33.3.4.12 新建诊断.....2341
33.3.4.13 使用诊断.....2343
33.3.4.14 修改诊断.....2344
33.3.4.15 删除诊断.....2345
33.3.5 任务报警.....2347
33.3.5.1 配置模板.....2347
33.3.5.1.1 查看模板.....2347
33.3.5.1.2 新增模板.....2347
33.3.5.1.3 修改模板.....2348
33.3.5.1.4 删除模板.....2350
33.3.5.2 配置报警规则.....2350
33.3.5.2.1 配置成功报警规则.....2350
33.3.5.2.2 配置失败报警规则.....2351
33.3.5.2.3 配置超时未开始规则.....2352
33.3.5.2.4 配置超时未结束规则.....2353
33.3.5.2.5 配置运行超时规则.....2354
33.3.5.3 查看报警记录.....2355
33.3.6 FAQ.....2355
33.4 数据分析.....2356
33.4.1 数据分析服务简介.....2356
33.4.2 进入数据分析.....2356
33.4.3 连接数据源.....2357
33.4.3.1 连接MaxCompute数据源.....2357
33.4.3.2 连接 RDS 数据源.....2359
33.4.3.3 通过自定义 SQL 创建数据集.....2360
33.4.4 创作一张工作表.....2361
33.4.5 管理数据集.....2368
33.4.5.1 预览数据集.....2369
33.4.5.2 立即使用数据集.....2370
33.4.5.3 重命名/删除/移动数据集.....2370
33.4.5.4 查看数据集属性.....2370
33.4.5.5 更新数据集结构.....2371
33.4.5.6 构建关联模型.....2371
33.4.6 使用分析面板.....2372
33.4.7 合并单元格.....2376

33.4.8 分类汇总和自动求和.....2376
33.4.9 排序.....2377
33.4.10 交换 XY 轴.....2377
33.4.11 分析还原.....2378
33.4.12 撤销/重做/剪切/复制/粘贴.....2378
33.4.13 导出.....2378
33.4.14 文本格式设置.....2379
33.4.15 表格属性设置.....2379
33.4.16 斜线表头.....2380
33.4.17 锁定首行首列.....2380
33.4.18 图表.....2381
33.4.19 函数.....2382
33.4.20 条件格式.....2383
33.4.21 行级权限.....2384
33.5 报表制作.....2385
33.5.1 报表制作服务简介.....2385
33.5.2 制作报表.....2386
33.5.3 仪表盘.....2398
33.5.4 数据产品.....2399
33.5.4.1 新建数据产品.....2399
33.5.4.2 可编辑功能.....2402
33.5.5 画布.....2403
33.5.6 布局.....2404
33.5.6.1 列布局.....2404
33.5.6.2 TAB 页.....2406
33.5.6.3 WEB 页面.....2408
33.5.7 图表.....2410
33.5.7.1 图表简介.....2410
33.5.7.2 数据源.....2411
33.5.7.3 图表样式.....2416
33.5.7.3.1 交叉表.....2416
33.5.7.3.2 柱状图.....2417
33.5.7.3.3 曲线图.....2418
33.5.7.3.4 饼图.....2419
33.5.7.3.5 油量表.....2420
33.5.7.3.6 雷达图.....2421
33.5.7.3.7 极地坐标图.....2421
33.5.7.3.8 散点图.....2422
33.5.7.3.9 地图气泡图.....2422
33.5.7.3.10 地图色彩图.....2423
33.5.7.3.11 漏斗图.....2423

33.5.7.3.12 指标看板 2424
33.5.8 控件 2425
33.5.8.1 控件简介 2425
33.5.8.2 文本框 2426
33.5.8.3 过滤器 2428
33.5.8.4 日期时间 2430
33.5.8.5 查询条件 2431
33.5.8.6 按钮 2434
33.5.8.7 评论 2437
33.5.9 转让 2438
33.5.10 分享 2439
33.5.11 发布 2440
33.5.12 文档管理 2442
33.5.13 查询加速 2444
33.5.14 租户管理 2445
33.5.14.1 加速设置 2445
33.5.14.2 管理员设置 2446
33.5.14.3 发布导出设置 2446
33.5.14.4 分享有效期设置 2447
33.5.14.5 导入导出 2448
33.5.15 FAQ 2449
33.5.15.1 基于金融云的登录认证体系 2449
33.5.15.2 使用第三方账户体系登录 2449
34 机器学习PAI 2454
34.1 产品概述 2454
34.2 基本概念 2455
34.3 快速开始 2456
34.3.1 数据准备 2456
34.3.2 数据预处理 2459
34.3.3 数据可视化 2462
34.3.4 算法建模 2463
34.3.5 模型评估 2464
34.4 使用教程 2465
34.4.1 源/目标 2465
34.4.2 数据预处理 2466
34.4.2.1 采样与过滤 2466
34.4.2.1.1 加权采样 2466
34.4.2.1.2 随机采样 2468
34.4.2.1.3 过滤与映射 2469
34.4.2.1.4 分层采样 2471
34.4.2.2 数据合并 2472

34.4.2.2.1 合并列.....2472
34.4.2.2.2 拆分.....2475
34.4.2.2.3 UNION.....2476
34.4.2.3 其他.....2479
34.4.2.3.1 增加序号列.....2479
34.4.2.3.2 归一化.....2480
34.4.2.3.3 标准化.....2482
34.4.3 特征工程.....2487
34.4.3.1 特征变换.....2487
34.4.3.1.1 主成分分析 PCA.....2487
34.4.3.2 特征重要性评估.....2489
34.4.3.2.1 随机森林特征重要性.....2489
34.4.3.2.2 线性模型特征重要性.....2490
34.4.3.3 统计分析.....2491
34.4.3.3.1 百分位.....2491
34.4.3.3.2 全表统计.....2493
34.4.3.3.3 皮尔森系数.....2494
34.4.3.3.4 直方图.....2496
34.4.3.3.5 散点图.....2496
34.4.3.3.6 相关系数矩阵.....2502
34.4.3.4 机器学习.....2505
34.4.3.4.1 二分类.....2505
34.4.3.4.1.1 线性支持向量机.....2505
34.4.3.4.1.2 逻辑回归二分类.....2507
34.4.3.4.1.3 GBDT二分类.....2509
34.4.3.4.2 多分类.....2513
34.4.3.4.2.1 逻辑回归多分类.....2513
34.4.3.4.2.2 随机森林.....2515
34.4.3.4.2.3 朴素贝叶斯.....2519
34.4.3.4.3 协同过滤etrec.....2520
34.4.3.4.4 GBDT 回归.....2524
34.4.3.4.5 K 均值聚类.....2527
34.4.3.4.6 评估.....2530
34.4.3.4.6.1 混淆矩阵.....2530
34.4.3.4.6.2 回归模型评估.....2532
34.4.3.4.6.3 多分类评估.....2533
34.4.3.4.6.4 二分类评估.....2534
34.4.3.4.6.5 聚类模型评估.....2535
34.4.3.4.7 预测.....2539
34.4.3.4.8 工具.....2541
34.4.3.4.8.1 SQL脚本.....2541

34.4.3.4.9 文本分析.....2543
34.4.3.4.9.1 阿里分词.....2543
34.4.3.4.9.2 词频统计.....2546
34.4.3.4.9.3 TF-IDF.....2548
34.4.3.4.9.4 PLDA.....2550
34.4.3.4.9.5 Word2Vec.....2552
34.4.3.4.9.6 关键词提取.....2554
34.4.3.4.9.7 文本摘要.....2557
34.4.3.4.9.8 三元组转 kv.....2559
34.4.3.4.9.9 句子拆分.....2561
34.4.3.4.10 网络分析.....2563
34.4.3.4.10.1 k-Core.....2563
34.4.3.4.10.2 单源最短路径.....2565
34.4.3.4.10.3 PageRank.....2566
34.4.3.4.10.4 标签传播聚类.....2569
34.4.3.4.10.5 标签传播分类.....2571
34.4.3.4.10.6 Modularity.....2574
34.4.3.4.10.7 最大联通子图.....2575
34.4.3.4.10.8 点聚类系数.....2577
34.4.3.4.10.9 边聚类系数.....2578
34.4.3.4.10.10 计数三角形.....2581
34.4.3.4.10.11 树深度.....2582
34.4.3.4.11 常见问题.....2584
34.5 使用场景-泰坦尼克号沉船事件.....2591

1 云控制台简介

简单介绍云控制台的定义及使用流程。

概述

云控制台是一种将计算、网络和存储等资源作为基本组成元素，根据系统需求进行选择和预定义的过程，基于现有的硬件模块，并结合虚拟化、软件定义、分布式架构，通过网络聚合多套标准化通用硬件，实现模块化的无缝横向扩展（scale-out），构建出完全依靠软件驱动的环境。通过云控制台，实现与阿里云的无缝衔接，使您既可以享用自建专有云服务，又可以借助阿里云快速扩展自身云计算能力，为您提供最便捷的混合云部署方案。

云控制台致力于打造云计算的服务能力平台，可以为政府、企业、金融、教育、交通、能源等客户提供大规模、成熟稳定的云计算及云业务服务，并提供新的应用开发和运营模式，帮助用户从关注运维向关注开发转变，构建面向下一代互联网及物联网的信息基础设施，是构建政企云的理想基础平台。

总体使用流程

云控制台的操作主要分为如下三个部分：

- 系统初始化：该部分主要是完成系统的基本配置，包括创建区域、部门、项目和用户、创建基础资源VPC以及创建云监控联系人和联系人组。
- 云资源创建：该部分的资源创建由管理员根据需求直接创建。
- 云资源管理：该部分包含了启动资源、使用资源、变更资源配置以及释放资源等资源管理操作。



2 配置要求

访问云控制台前，需要确认您的浏览器和操作系统是否满足配置要求。

当您访问云控制台时，本地 PC 需要满足如表 2: 配置要求表所示要求才可以正常登录。

表 2: 配置要求表

内容	内容要求
浏览器	• Internet Explorer 浏览器：11 及以上版本 • Chrome 浏览器（推荐）：42.0.0 及以上版本 • Firefox 浏览器：30 及以上版本 • Safari 浏览器：9.0.2 版本及以上版本
操作系统	• Windows XP/7 及以上版本 • MAC OS X系统

3 登录云控制台

本节以Chrome浏览器为例，介绍云产品用户如何登录到云控制台。

前提条件

登录云控制台前，需要您确认以下信息：

- 已从管理员处获取DTCenter云控制台的IP地址或服务域名地址。

  云控制台登录网址为：http://x.x.x.x/manage，其中 x.x.x.x 为 IP 地址或者服务域名地址。
- 请确保您的Chrome浏览器已升级至42.0.0及以上版本。

操作步骤

1. 打开Chrome浏览器。
2. 在地址栏中，输入云控制台的网络地址 http:// 云控制台的 *IP* 地址或服务域名地址/manage，按回车键。

   显示登录界面如图 *1:* 登录云控制台所示。

   图 1: 登录云控制台

3. 输入正确的用户名、密码及验证码。

- 首次登录云控制台时，需要修改登录用户名的密码，请按照提示完成密码修改。为提高安全性，密码必须满足最小复杂度要求，即包含英文大/小写字母（A~Z、a~z）、数字（0~9）、特殊符号（！、@、#、$、% 等）中的两种，并且密码长度为 8~20 位。

4. 单击**登录**，进入DTCenter主页面。
5. （可选）进入各云产品控制台。

   在菜单栏上单击**云资源中心** > **云产品管理**，选择对应的云产品，如图 2: 选择云产品所示。

图 2: 选择云产品



4 熟悉界面使用

云控制台的 Web 界面主要分为板块页签、登录用户、菜单导航以及操作区四个区域。

云控制台的 Web 界面主要分为四个区域，如图 *3: Web*页面所示。

图 3: Web页面



说明: 不同角色显示的界面有所不同，请以实际界面为准。

Web 功能区域说明如表 *3:* 区域说明所示。

表 3: 区域说明

区域	说明
板块页签	• 首页：展示云控制台下的资源概览、监控情况以及库存概览。 • 云资源中心：管理各类云产品和资源，设置资源的计费规则和有效期，简化资源使用方式。 • 流程管理：管理云资源的申请流程以及故障申报。 • 系统管理：管理系统中的用户、项目、部门等基本配置，使系统开始运行。 • 联机帮助：用于查看联机帮助信息。
登录用户	：显示当前用户的用户名。

区域	说明
	 ：单击可以选择显示当前用户的个人信息和注销登录。 在个人信息页面，您可以 • 查看基本资料 • 修改信息 • 修改头像 • 修改密码 • 查看AccessKey • 查看第三方访问AccessKey
菜单导航	当选择页签之后，该页签对应的菜单就会显示在左侧的导航树中。选择相应的菜单，就可以开始进行功能配置。
操作区	当选择菜单项后，该菜单项的功能配置界面就会显示在右侧的操作区中。

5 角色与权限关系

角色是一组访问权限的集合，每个角色都有对应的权限范围。

一个用户可以拥有多个角色，即这个用户可以获得这些角色定义的所有权限。权限行为是指权限能做的具体的操作，云控制台权限操作包括页面不可见、查看、创建、删除和配置。

云控制台中有6个默认角色，本文档以全局资源监察员、资源监察员及资源使用人等角色的使用为例，提供操作指导。

全局资源监察员、资源监察员及资源使用人的角色与权限关系如表 *4:* 角色与权限关系图所示。

表 4: 角色与权限关系图

角色	权限	说明
全局资源监察员	• 全局范围的备份管理（读） • 全局范围的大数据产品管理 • 配额管理（读） • 全局范围的云资源管理（读） • 组织管理（读） • 申报使用 • 角色管理（读） • 全局范围的用户和项目管理（读） • 全局范围 VPC 管理（读）	拥有所有部门的资源以及用户中心的查看权限。
资源监察员	部门范围的云资源管理（读）、部门范围的 VPC 管理（读）、申报使用、组织管理（读）、大数据产品部门范围。 • 项目范围的备份管理（读） • 部门范围的大数据产品管理 • 配额管理（读） • 部门范围的云资源管理（读） • 申报使用 • 用户和项目管理（读） • VPC管理（读）	拥有查看所属部门以及下级部门资源的权限。
资源使用人	• 报警人（读、写） • 项目范围的备份管理（读、操作） • 项目范围的大数据产品管理	• 拥有对所属项目资源的操作权限。

角色	权限	说明
	• 项目范围的云资源管理（读、操作） • 申报使用	• 拥有对所属项目资源查看和创建报警项的权限。

6 管理云监控中心

6.1 云监控服务简介

云监控服务是一个开放性的监控平台，提供了资源的实时监控、告警、通知等服务，为用户的产品、业务保驾护航。

目前可以监控云服务器、负载均衡、云数据库和云存储服务的相关指标。用户可以通过这些指标设置告警规则和通知策略，以便及时了解各产品服务的实例资源运行状况和性能，在收到资源不足类型的告警时及时考虑扩容。

云监控主要具有以下功能：

- 自动监控：监控会根据用户创建的云服务器资源或者弹性伸缩组而自动启动，用户无需手动干预，也无需安装任何插件。用户申请资源后就可以在云监控的实例监控中查看到具体实例的监控数据。
- 灵活告警：通过对监控指标设置告警、阈值，暂停、启用告警，实现告警的灵活配置。
- 实时通知：设置告警通知，可以实现用户实时接收短信或邮件通知。当告警规则下产生告警、数据不足或告警恢复正常等状态变化时，系统会向用户的电话发送信息或向邮箱发送邮件通知。

6.2 云监控指标说明

云监控服务通过云资源的监控指标，检测服务的可用性。您可以通过对这些监控指标设置告警规则和通知策略，以便及时了解各产品服务的实例资源运行状况和性能。

云监控可以监控其他服务的资源，目前支持 ECS、负载均衡、RDS 和 OSS。各服务对应的监控指标如 ECS 监控指标表、负载均衡监控指标表、RDS 监控指标表和 OSS 监控指标表所示。

表 5: ECS支持的监控指标

指标名称	指标含义	测量对象	计算公式	相关说明
CPU 使用率	用于统计测量对象的 CPU 使用率，以百分比为单位。	云服务器	单个云服务器 CPU 使用率 /单个云服务器的 CPU 总核数	无
内存使用率	用于统计测量对象的内存使用	云服务器	该云服务器内存使用量 / 该	如果您使用 free 或 top 命令查询 linux 服务器的内存使用情况发现和

指标名称	指标含义	测量对象	计算公式	相关说明
	率，以百分比为单位。		云服务器内存总量	控制台实际看到的内存使用情况不一致，是因为云监控在计算内存使用情况时去除了作为缓存的内存。
磁盘 IO 读	用于统计每秒从测量对象读出的数据量，以千字节/秒为单位。	云服务器	云服务器的磁盘读出的字节数之和 /测量周期	对于系统为 linux 的主机，磁盘 IO 的监控数据获取是通过 iostat 进行的；若您发现 Linux 主机无磁盘 IO 数据，请确认您的机器是否安装了 iostat 指令。Redhat 与CentOS 用户可以使用 yum 进行安装；Ubuntu 与 Debian 用户可以使用 apt-get 进行安装。
磁盘 IO 写	用于统计每秒写到测量对象的数据量，以千字节/秒为单位。	云服务器	云服务器的磁盘写入的字节数之和 /测量周期	无
磁盘使用率	用于统计测量对象的磁盘使用情况，以百分比为单位。	云服务器	云服务器磁盘使用容量 / 云服务器磁盘总容量	无
入流量	用于统计每秒流入测量对象的网络流量，以千比特/秒为单位。	云服务器	无	无
出流量	用于统计每秒流出测量对象的网络流量，以千比特/秒为单位。	云服务器	无	超出购买的带宽时，会出现访问不可达或请求缓慢的情况。监控图上的 eth0是服务器内网网卡，eth1 是服务器外网网卡。
TCP 连接数	服务器建立的 TCP 连接总数。	云服务器	无	无
进程数	添加进程监控后，会对指定关注的进程进行计数，显示正在运行的被关注进程总数。	云服务器	无	如果您想监控服务器上进程运行的情况，在进程挂掉时进行报警，可添加进程数监控，设置当进程数不等于实际数量时进行报警。

指标名称	指标含义	测量对象	计算公式	相关说明
平均负载	针对 Linux 的概念，服务器的 load average 的数值。	云服务器		系统平均负载的数值不大于1 就可以，如果您服务器是多核的，平均负载的值需要除以cpu 核数，得到的值小于 1 就可以，通常平均负载数值超过 1 ，说明目前进程存在队列，服务器运行会慢。



说明:

飞天专有云区域的ECS实例需要安装监控插件，才能采集操作系统级别指标数据。

安装方法：在ECS实例所在行单击**点击安装**，或者选中需要安装的ECS实例，单击**批量安装插件**。

安装监控插件成功，5-10分钟后，监控图表才会显示监控数据。

表 6: RDS服务支持的监控指标

指标名称	指标含义	测量对象	计算公式
CPU 使用率	用于统计测量对象的 CPU 使用率，以百分比为单位。	云数据库	单个云数据库 CPU 使用率 / 单个云数据库的 CPU 总核数
内存使用率	用于统计测量对象的内存使用率，以百分比为单位。	云数据库	该云数据库内存使用量 / 该云数据库内存总量
磁盘使用率	用于统计测量对象磁盘空间的使用率，以百分比为单位。	云数据库	无
IOPS 使用率	用于统计测量对象每秒 IO 请求次数，以次/秒为单位。	云数据库	该云数据库的 IO 请求次数/测量周期
连接数使用率	用于统计每秒应用程序可以连接到测量对象的数量，以个/秒为单位。	云数据库	每秒应用程序可以连接到该云数据库的数量/测量周期

表 7: SLB 服务支持的监控指标

指标名称	指标含义	测量对象	相关说明
端口每秒流出数据包数	负载均衡每秒发出的数据包数量	负载均衡	无

指标名称	指标含义	测量对象	相关说明
端口每秒流入数据包数	负载均衡每秒接到的请求数据包数量	负载均衡	无
端口每秒流入流量	从外部访问负载均衡所需要消耗的流量	负载均衡	无
端口每秒流出流量	负载均衡访问外部所需要消耗的流量	负载均衡	无
端口当前活跃连接数	当时所有 ESTABLISHED 状态的连接	负载均衡	可以理解为并发量，但是不能等同。因为如果您采用的是长连接的情况，一个连接会同时传输多个文件请求。
端口当前非活跃连接数	指除 established 状态的其他所有状态 TCP连接数	负载均衡	无
端口当前新建连接数	统计周期内 TCP 三次握手的第一次 SYN_SENT 状态的数量	负载均衡	统计的是客户端连接到 SLB 的连接请求，活跃连接数、非活跃连接数统计的也是客户端到 SLB的连接请求。

表 8: OSS 服务支持的监控指标

指标名称	指标含义	测量对象
读取次数	用于统计测量对象读取的次数，以次为单位。	云存储
内部错误数	用于统计测量对象出现异常的数量，以个位单位。	云存储
互联网入流量	用于统计每秒流入测量对象的外网网络流量，以字节为单位。	云存储
互联网出流量	用于统计每秒流出测量对象的外网网络流量，以字节为单位。	云存储
内网入流量	用于统计每秒流入测量对象的内网网络流量，以字节为单位。	云存储
内网出流量	用于统计每秒流出测量对象的内网网络流量，以字节为单位。	云存储
写入次数	用于统计测量对象写入的次数，以次为单位。	云存储
存储已用空间	用于统计测量对象的存储空间使用量，以字节为单位。	云存储

6.3 管理报警联系人

6.3.1 新建报警联系人

您可以新建报警联系人，用于接收报警信息。

背景信息

报警联系人是报警消息的接收人，报警消息的发送方式包括手机短信和邮件。当监控数据满足报警规则时会对报警联系人发送告警信息。

操作步骤

1. 选择**云资源中心** > **云监控中心** > **报警人**。
2. 选择**报警联系人**页签。
3. 单击**新建联系人**。

   弹出**设置报警联系人**页面，如图 *4:* 设置报警联系人所示。

   图 4: 设置报警联系人



4. 添加报警人参数，如表 *9:* 设置报警人参数表所示。

表 9: 设置报警人参数表

参数	说明
用户名	输入报警联系人的用户名。
手机	手机输入报警联系人手机号码。手机号码用于系统将报警信息以短信形式通知报警联系人，因此请正确填写您的手机号码。若有变更，请您及时在平台上更新。
邮箱	输入报警联系人的邮箱地址。邮箱地址用于系统将报警信息以邮件形式通知报警联系人，因此请正确填写您的邮箱地址。若有变更，请您及时在平台上更新。
钉钉ID	输入报警联系人的钉钉ID。

5. 单击**确定**。

6.3.2 添加报警联系人到报警联系组

新建报警联系人后，为了便于管理，可以将报警联系人添加到报警联系组。

前提条件

- 已存在报警联系人，创建报警联系人相关操作请参见新建报警联系人。
- 已创建报警联系组，创建报警联系组相关操作请参见新建报警联系组。

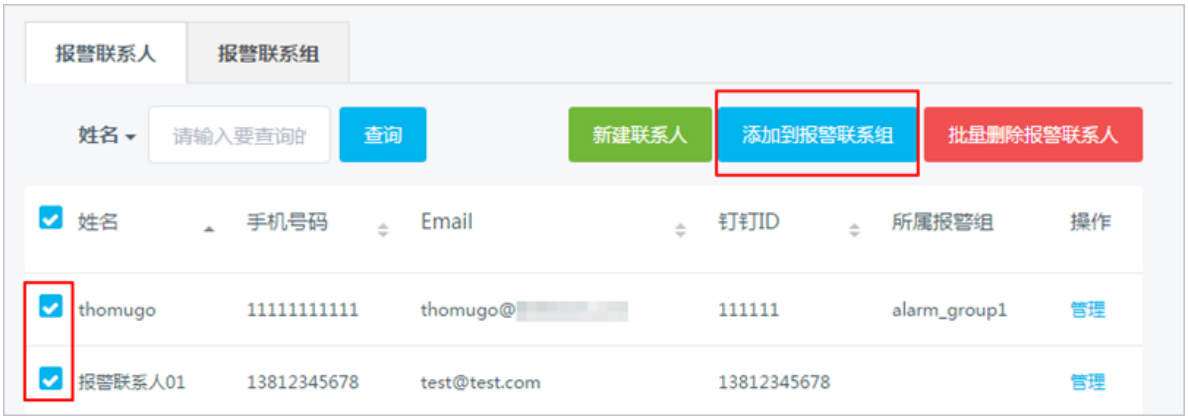
背景信息

一个报警联系人可以添加到多个报警联系组中。

操作步骤

1. 选择**云资源中心** > **云监控中心** > **报警人**。
2. 选择**报警联系人**页签。
3. 选择待添加的报警联系人，单击**添加到报警联系组**，如图 *5:* 添加报警联系人到报警联系组所示。

图 5: 添加报警联系人到报警联系组



4. 在弹出的对话框中，勾选需要加入的报警联系组，单击**确定**。

6.3.3 查询报警联系人

您可以在**报警联系人**页面，查询报警联系人信息及其所属的报警组。

操作步骤

1. 选择**云资源中心** > **云监控中心** > **报警人**。
2. 选择**报警联系人**页签。
3. 选择以姓名、手机、邮箱或钉钉ID为搜索条件，输入搜索关键字，单击**查询**，即可查询报警联系人的相关信息及其所属的报警组。

6.3.4 修改报警联系人信息

当报警联系人的信息发生变更时，可在**报警联系人**页面修改报警联系人的信息。

操作步骤

1. 选择**云资源中心** > **云监控中心** > **报警人**。
2. 选择**报警联系人**页签。
3. 在待修改的报警联系人后，单击，选择**编辑**。
4. 在弹出的对话框中，修改报警联系人的联系方式，包括手机、邮箱及钉钉ID。

6.3.5 删除报警联系人

根据业务需要，可删除不需要的报警联系人。

操作步骤

1. 选择**云资源中心** > **云监控中心** > **报警人**。
2. 选择**报警联系人**页签。

3. 进行如下操作：

   - 删除某个报警联系人

     在待修改的报警联系人后，单击，选择**删除**。

   - 批量删除多个报警联系人

     勾选多个待删除的报警联系人，单击右上角的**批量删除报警联系人**。

4. 在弹出的对话框中，单击**确定**。

6.4 管理报警联系组

6.4.1 新建报警联系组

报警联系组是一组报警联系人，可以包含一个或多个报警联系人。

背景信息

在报警设置中，均通过报警组发送报警通知。对每一个监控点，根据预先设定的报警方式在到达报警阈值时向报警组成员发送报警消息。

操作步骤

1. 选择**云资源中心** > **云监控中心** > **报警人**。
2. 选择**报警联系组**页签。
3. 单击**新建联系组**。

   弹出**报警组管理**页面，如图 *6:* 报警组管理所示。

   图 6: 报警组管理



4. 添加报警组参数，如表 *10:* 报警组参数表所示。

表 10: 报警组参数表

参数	说明
组名	输入报警组组名。组名必须由字母、数字、下划线组成，长度为 2个~20 个字符。
备注	输入报警组的描述信息。描述信息由字母、数字、下划线或 - 组成，长度为 2个~256 个字符。
选择联系人	将联系人加入报警组，操作方法如下： 在左侧的已有联系人中选择联系人，通过单击 -> 添加到右侧已选联系人中。 选择后，如果需要移除某个已选联系人，可以通过单击 <- 来操作。 **说明:** 如果未添加联系人，可先新建空的报警联系组后，再新建报警联系人，最后将报警联系人添加到报警联系组。

5. 单击**确定**。
6. （可选） 如果需要删除报警组中的报警联系人，可以在**报警联系组**页面，单击某个报警联系人后的**删除**，即可将该报警联系人从报警组中移出。

6.4.2 修改报警通知方式

默认已开启手机通知、邮箱通知和钉钉通知，您可以根据需要，关闭不需要的通知方式。

操作步骤

1. 选择**云资源中心** > **云监控中心** > **报警人**。
2. 选择**报警联系组**页签。
3. 在需要修改报警通知方式的报警联系人所在行，通过鼠标点击，选择开启或关闭手机通知、邮箱通知和钉钉通知，如图 7: 修改报警通知方式所示。

图 7: 修改报警通知方式



6.5 管理报警规则

6.5.1 新建报警规则

您可以为某个实例创建报警规则，对其进行监控。

前提条件

创建报警规则需要有云资源管理写或操作权限、报警人读权限。

背景信息

在设置报警规则前，建议先完成报警联系组的创建操作，您也可以在设置报警规则过程中创建。设置报警联系组的具体操作请参见新建报警联系组。

云监控报警规则用于对服务器性能进行监控报警，以便及时发现和处理服务器存在的问题，保障服务器安全、稳定、有效运行。

操作步骤

1. 进入**云资源中心** > **云监控中心** > **监控**页面。
2. 选择对应的云产品。

3. 在对应的实例后面，单击，选择**报警规则**，进入**报警项**页面。

   说明: 您也可以使用查询功能查询特定的实例，为其创建报警规则。

4. 单击**新建报警规则**。

   弹出**设置报警规则**页面，如图 *8:* 设置报警规则所示。

   图 8: 设置报警规则



5. 配置报警规则参数，如表 *11:* 参数说明所示。

   表 11: 参数说明

参数	说明
监控项	从下拉框中选择监控项。 监控项说明请参见x云监控指标说明。
统计周期	从下拉框中选择统计周期。 统计周期是进行一次数据统计的间隔。
统计方法	从下拉框中选择统计方法，统计方法有以下五种： • 平均值：在统计周期内采集的所有监控数据的平均值大于阈值时，才为超过阈值。 • 最大值：统计周期内采集的监控数据中，当最大值超过阈值时，才为超过阈值。 • 最小值：统计周期内采集的监控数据中，当最小值超过阈值时，才为超过阈值。

参数	说明
	• 求和值：对统计周期内采集的监控数据进行求和，求和后的结果超过阈值，才为超过阈值。 • 采样计数值：是指在统计周期内上报的监控数据条数超过阈值时才为超过阈值。

6. 单击**下一步**。
7. 设置通知对象参数，如表 *12:* 报警人参数说明所示。

   通知对象即报警人，报警人设置的详细操作请参见新建报警联系人。

   表 12: 报警人参数说明

参数	说明
重试几次报警	从下拉框中选择重试几次后触发报警。 重试几次后报警是指连续几个统计周期超过阈值后触发报警，只有超出重试次数，才会通知报警对象。
联系人通知组	选择联系人通知组。 对监控项设置报警规则后，当监控数据满足报警规则时会给报警联系人发送告警信息。
通知时间	选择通知时间，通知时间是指发送告警信息的时间段。

8. 单击**确定**。

6.5.2 批量创建报警规则

您可以对多个实例创建相同的报警规则进行监控。

操作步骤

1. 进入**云资源中心** > **云监控中心** > **监控**页面。
2. 选择对应的云产品后，勾选多个实例，如图 *9:* 选择多个实例所示。

   图 9: 选择多个实例



3. 单击右上角的**批量创建报警规则**。

   系统弹出**批量设置报警规则**窗口，如图 *10:* 批量设置报警规则所示。

   图 10: 批量设置报警规则



4. 配置报警规则参数，如表 *13:* 参数说明所示。

   表 13: 参数说明

参数	说明
监控项	从下拉框中选择监控项。 监控项说明请参见x云监控指标说明。
统计周期	从下拉框中选择统计周期。 统计周期是进行一次数据统计的间隔。
统计方法	从下拉框中选择统计方法，统计方法有以下五种：

参数	说明
	• 平均值：在统计周期内采集的所有监控数据的平均值大于阈值时，才为超过阈值。 • 最大值：统计周期内采集的监控数据中，当最大值超过阈值时，才为超过阈值。 • 最小值：统计周期内采集的监控数据中，当最小值超过阈值时，才为超过阈值。 • 求和值：对统计周期内采集的监控数据进行求和，求和后的结果超过阈值，才为超过阈值。 • 采样计数值：是指在统计周期内上报的监控数据条数超过阈值时才为超过阈值。

5. 单击**下一步**。
6. 设置通知对象参数，如表 *14:* 报警人参数说明所示。

   通知对象即报警人，报警人设置的详细操作请参见新建报警联系人。

   表 14: 报警人参数说明

参数	说明
重试几次报警	从下拉框中选择重试几次后触发报警。 重试几次后报警是指连续几个统计周期超过阈值后触发报警，只有超出重试次数，才会通知报警对象。
联系人通知组	选择联系人通知组。 对监控项设置报警规则后，当监控数据满足报警规则时会给报警联系人发送告警信息。
通知时间	选择通知时间，通知时间是指发送告警信息的时间段。

7. 单击**确定**。

6.6 管理报警项

报警项用于显示云监控报警规则的监控项，有 ECS、SLB、RDS、OSS 报警项，以方便快速查看监控项，保障服务器安全、稳定、有效运行。

ECS、SLB、RDS、OSS 的报警项管理操作相同，内容不同。以下以 ECS 为例说明。

6.6.1 查看报警项

报警规则创建后，您可以在**报警项**页面查看您的报警项。

操作步骤

1. 进入**云资源中心** > **云监控中心** > **报警项**页面。
2. 选择**ECS**，显示 ECS 的报警项页面，如图 *11: ECS*报警项所示。

图 11: ECS报警项



3. 您可以选择区域、输入监控资源 ID、输入监控资源名、选择监控项、报警状态及启用状态后，单击**查询**，查询报警项。

6.6.2 查看报警历史

报警发生后，您可以在**报警项**中查看历史报警信息。

操作步骤

1. 进入**云资源中心** > **云监控中心** > **报警项**页面。
2. 选择对应的云产品。
3. 在报警项所在行，单击，选择**报警历史**，查看历史报警信息。

6.6.3 修改报警项

当您需要修改某个报警项的报警规则时，您可以在对应的云产品报警项页面修改该报警项。

操作步骤

1. 进入**云资源中心** > **云监控中心** > **报警项**页面。

2. 选择对应的云产品。
3. 在报警项所在行，单击  ，选择**修改**，修改报警规则。

   修改报警规则相关操作可参考新建报警规则中的设置报警规则。

6.6.4 暂停报警项

您可以根据需要，暂停一个或多个报警项。

操作步骤

1. 进入**云资源中心** > **云监控中心** > **报警项**页面。
2. 选择对应的云产品。
3. 进行如下操作：
   - 暂停某个报警项

     在报警项所在行，单击  ，选择**暂停**。
   - 批量暂停多个报警项

     勾选多个报警项，在右上角单击**批量暂停**。
4. 在弹出的对话框中，单击**确定**。

   暂停报警后将不再收到该报警项的报警通知。

6.6.5 批量启动报警项

报警项被暂停后，如果需要重新启用，可通过**批量启动**功能，启用暂停的报警项。

操作步骤

1. 进入**云资源中心** > **云监控中心** > **报警项**页面。
2. 选择对应的云产品。
3. 勾选需要启用的报警项，单击**批量启动**。

6.6.6 查看报警通知对象

报警规则创建后，您可以查看在**报警项**页面查看各报警项对应的通知对象。

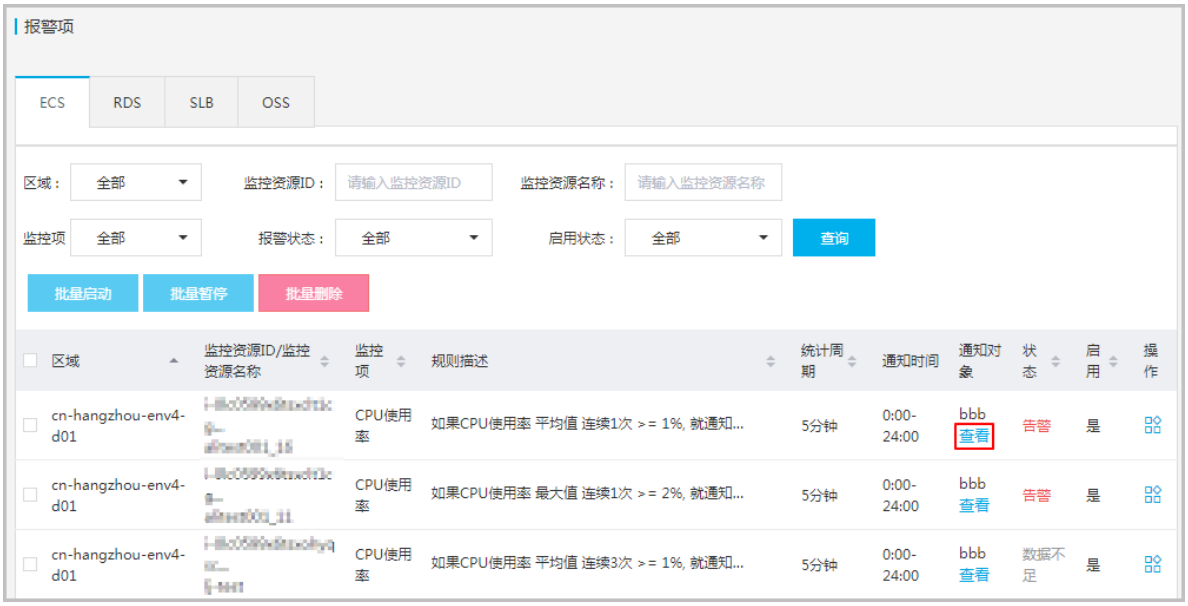
操作步骤

1. 进入**云资源中心** > **云监控中心** > **报警项**页面。
2. 选择对应的云产品。

3. 在报警项对应的**通知对象**下，单击**查看**，如图 *12:* 查看报警通知对象所示。

   弹出一个窗口显示报警联系人信息。

图 12: 查看报警通知对象



6.6.7 删除报警项

当您不需要使用某个或某几个报警项时，您可以删除这些报警项。

操作步骤

1. 进入**云资源中心** > **云监控中心** > **报警项**页面。
2. 选择对应的云产品。
3. 进行如下操作：
   - 删除某个报警项

     在报警信息所在行，单击，选择**删除**。
   - 批量删除多个报警项

     勾选多个报警项，在右上角单击**批量删除**。
4. 在弹出的提示框中，单击**确定**，即可删除。

6.7 管理集群告警

6.7.1 创建监控组

您可以创建弹性监控组，用于统一监控管理。

操作步骤

1. 选择**云资源中心** > **云监控中心** > **集群告警**。
2. 在**弹性监控组列表**页面，单击右上角的**创建监控组**。

   显示**弹性监控组创建**页面，如图 *13:* 创建监控组所示。

   图 13: 创建监控组



3. 输入监控组名称、描述信息并选择专有云的监控区域。
4. 单击**创建**。

   创建成功后，系统会自动生成一个监控组ID。

5. （可选）监控组创建成功后，您还可以进行如下操作：

   - 修改监控组信息

     在**弹性监控组列表**页面对应的监控组后，单击，选择**修改信息**，可以修改监控组的名称和描述信息。

- 删除监控组

  在**弹性监控组列表**页面对应的监控组后，单击，选择**删除监控组**，单击**确定**。

6.7.2 将实例加入监控组

将实例加入监控组，对实例进行监控。

操作步骤

1. 选择**云资源中心** > **云监控中心** > **集群告警**。
2. 在**弹性监控组列表**页面，单击某个监控组ID的链接。
3. 在弹出的**详情**页面，单击**移入实例**，如图 *14:* 详情页面所示。

图 14: 详情页面



4. 在**移入实例**窗口，选择目标实例，单击**提交**，如图 *15:* 选择目标实例所示。

图 15: 选择目标实例



实例加入监控组后，如果不需要监控该实例了，可以在**详情**页面对应的实例后，单击**移出监控组**，将该实例移出。

6.7.3 创建集群告警规则

为监控组设置告警规则，用于实例的监控。

操作步骤

1. 选择**云资源中心** > **云监控中心** > **集群告警**。
2. 在**弹性监控组列表**页面，单击某个监控组ID的链接。
3. 在监控组页面，选择**告警规则**，如图 *16:* 告警规则页面所示。

图 16: 告警规则页面



4. 单击**创建告警规则**。

弹出**创建集群告警规则**对话框，如图 *17:* 创建集群告警规则所示。

图 17: 创建集群告警规则



5. 设置告警规则，相关告警参数说明如表 *15:* 集群告警规则相关参数说明所示。

表 15: 集群告警规则相关参数说明

参数	说明
规则名称	报警规则的名称。

参数	说明
描述	对报警规则的描述，用于识别不同的报警规则。
是否启用	• 启用 选择**启用**，则报警规则创建成功后即可启用。 • 不启用 选择**不启用**，则报警规则创建成功后不启用该规则。
监控项	从下拉框中选择监控项： • CPU使用率 • 内存使用率 • 入流量 • 出流量
统计方法	从下拉框中选择统计方法，统计方法有以下五种： • Average：平均值，在统计周期内采集的所有监控数据的平均值大于阈值时，才为超过阈值。 • Max：最大值，统计周期内采集的监控数据中，当最大值超过阈值时，才为超过阈值。 • Min：最小值，统计周期内采集的监控数据中，当最小值超过阈值时，才为超过阈值。
符号	从下拉框中选择符号： • >=：大于或等于 • >：大于 • <：小于 • <=：小于或等于
阈值	监控项的阈值大小，请填写大于等于0的数值。
统计周期（秒）	统计周期是进行一次数据统计的间隔。 在文本框中输入统计值，为60的整数倍。
连续报警次数（次）	触发报警后，连续报警的次数，为大于或等于1的值。
通知url	触发报警后，用于接收报警信息的地址。 参数取值： • scalingGroupId：监控组ID • scalingAlarmRuleId：集群弹性告警规则ID

参数	说明
	例如：http://notifyUrl/alarm?scalingGroupId=test&scalingAlarmRuleId=test

6. 单击**创建**。

6.7.4 查看集群告警信息

触发告警后，您可以在**告警信息**页面查看各告警信息。

操作步骤

1. 选择**云资源中心** > **云监控中心** > **集群告警**。
2. 在**弹性监控组列表**页面，单击某个监控组ID的链接。
3. 在监控组页面，选择**告警信息**，可以看到已触发的告警信息。

6.8 查看监控图表

通过查看监控图表，了解各实例的运行情况。

操作步骤

1. 进入**云资源中心** > **云监控中心** > **监控**页面。
2. 选择对应的云产品。
3. 在需要查看的实例后，单击，选择**监控图表**。

   在显示的页面中可以看到各监控项的监控数据。

6.9 查看报警信息

通过查看报警信息，您可以清晰了解 ECS、SLB、RDS 以及 OSS 的运行情况，及时获取异常信息。

背景信息

报警信息用于显示不符合报警规则要求的报警项信息。

说明: 本节以查看 ECS 的报警信息为例提供操作指导，其他云资源的操作类似。

操作步骤

1. 选择**云资源中心** > **云监控中心** > **报警信息**，进入**报警信息**页面。

2. 您可以根据区域、监控资源　ID、监控资源名称、监控项、日期等进行过滤查询。查询结果字段说明如下表所示。

表 16: 报警信息字段说明表

字段	说明
区域	监控对象所在的区域名称。
监控实例	监控对象的实例 ID 名称
监控项	监控对象的监控项。
描述	报警信息的详细描述。
触发状态	触发报警的状态，有报警和数据不足两种。
阈值	监控项的阈值大小。
报警值	产生报警开始时对应的监控项的值。
开始时间	产生报警的开始时间。
结束时间	报警结束的时间。

3. （可选）单击**导出**，将当前显示的报警信息以 .xls 格式导出到个人 PC 中。导出文件名称为 ：资源名称+alarm.xls ，固定保存路径为 *C:\Users*\用户名*Downloads*。

7 资源日常管理

7.1 查看运行大盘

云控制台通过图表方式展示了当前系统资源的使用情况和监控情况，方便您了解当前资源的使用情况。

说明: 不同区域类型所呈现的资源类型将有所不同，请以实际页面为准。

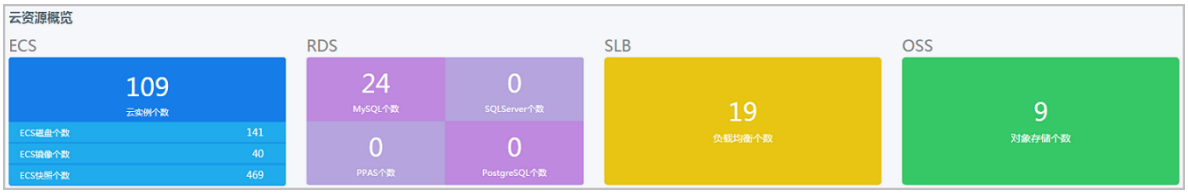
7.1.1 查看区域资源总量

资源总量概览汇总了各个资源的使用情况。

操作步骤

1. 单击**首页**，各个资源的使用情况如图 *18:* 云资源概览所示。

图 18: 云资源概览



资源总量概览各个字段的说明如表 *17:* 资源总量概览字段说明表所示。

表 17: 资源总量概览字段说明表

云产品	内容	说明
ECS	云实例个数	当前用户的 ECS 实例总量。
	ECS 磁盘个数	当前用户的磁盘总量。
	ECS 镜像个数	当前用户的镜像总量。
	ECS 快照个数	当前用户的快照总量。
SLB	负载均衡个数	当前用户的SLB 实例总量。
OSS	对象存储个数	当前用户的OSS 实例总量。
RDS	MySQL个数	当前用户的 MySQL 类型的 RDS 实例总量。

云产品	内容	说明
	SQLServer个数	当前用户的 SQLServer 类型的 RDS 实例总量。
	PPAS个数当	前用户的PPAS类型的RDS实例总量。
	PostgreSQL个数	当前用户的PostgreSQL类型的RDS实例总量。

2. 在**云资源概览**下，单击色块上的参数，系统会跳转到对应的资源页面，在资源页面可以查看详细的资源信息。

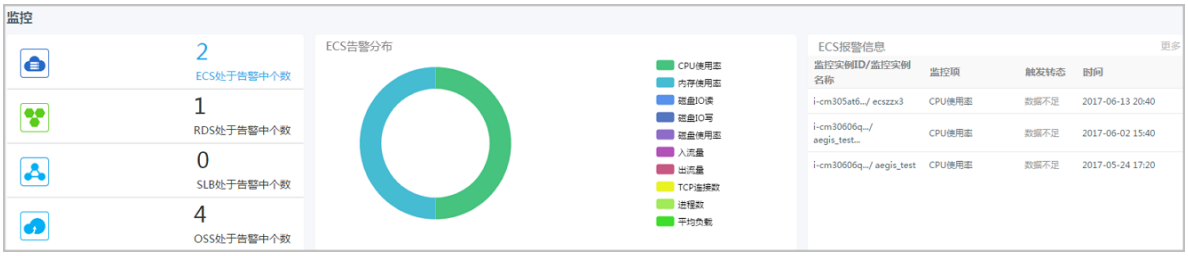
7.1.2 查看区域告警信息

通过告警分布图可以查看各个资源触发告警项的数量，便于您快速了解当前各资源的健康状态。

操作步骤

1. 在**首页**页面，各个资源的告警信息如图 *19:* 监控数据所示。

图 19: 监控数据



2. 分别单击左边的资源栏，查看各资源的告警信息。

   各资源的告警项说明请参见云监控指标说明。

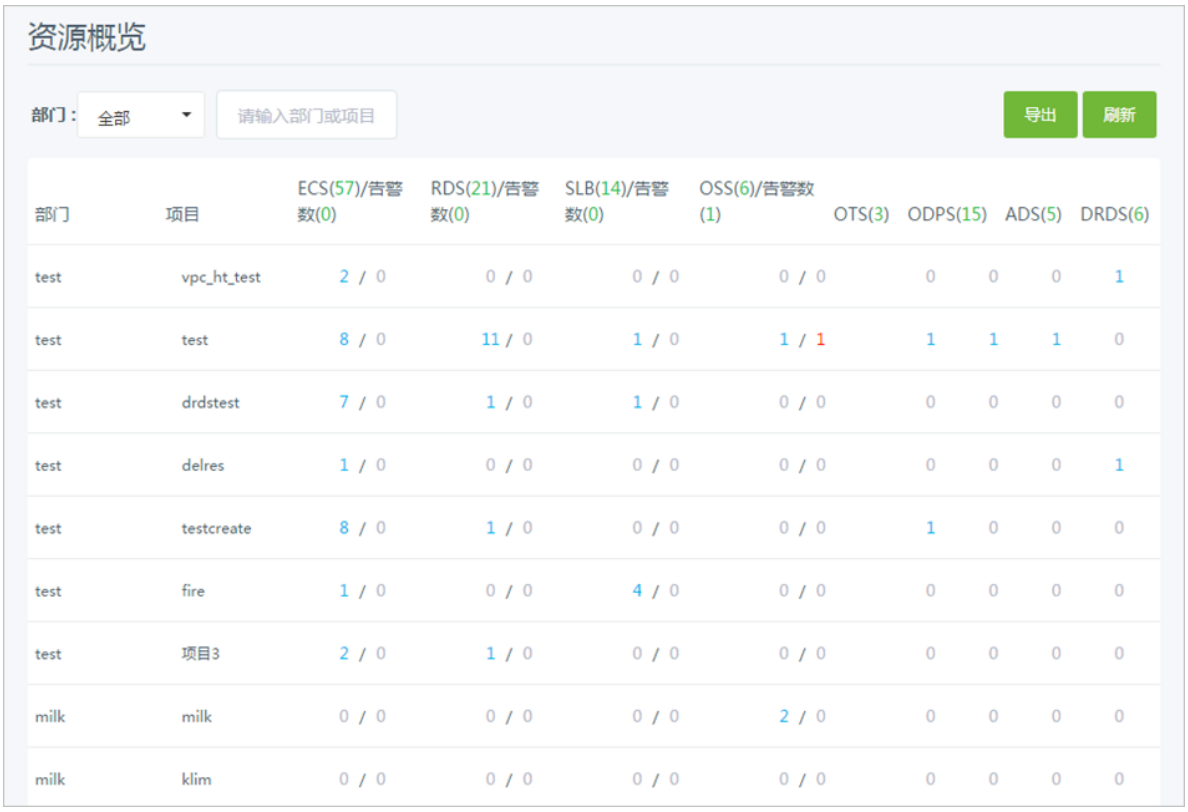
7.2 查看并导出项目资源概览

云资源概览通过表的形式展示云控制台上各部门和项目下的资源数以及告警数。

操作步骤

1. 单击**云资源中心** > **资源概览**，打开云资源概览页面，如图 *20:* 云资源概览所示。

图 20: 云资源概览



部门	项目	ECS(57)/告警数(0)	RDS(21)/告警数(0)	SLB(14)/告警数(0)	OSS(6)/告警数(1)	OTS(3)	ODPS(15)	ADS(5)	DRDS(6)
test	vpc_ht_test	2 / 0	0 / 0	0 / 0	0 / 0	0	0	0	1
test	test	8 / 0	11 / 0	1 / 0	1 / 1	1	1	1	0
test	drdstest	7 / 0	1 / 0	1 / 0	0 / 0	0	0	0	0
test	delres	1 / 0	0 / 0	0 / 0	0 / 0	0	0	0	1
test	testcreate	8 / 0	1 / 0	0 / 0	0 / 0	1	0	0	0
test	fire	1 / 0	0 / 0	4 / 0	0 / 0	0	0	0	0
test	项目3	2 / 0	1 / 0	0 / 0	0 / 0	0	0	0	0
milk	milk	0 / 0	0 / 0	0 / 0	2 / 0	0	0	0	0
milk	klim	0 / 0	0 / 0	0 / 0	0 / 0	0	0	0	0

您可以通过云资源概览快速地查看各部门、项目下的资源数和告警数，便于您了解资源的分布情况和运行情况。

说明： 单击界面上的参数，系统会跳转到对应的资源页面，在资源页面可以查看详细的资源信息。

2. 单击**导出**，可以将云资源列表导出，方便您进行查看。

   导出的文件将以*resource(ByProject).xls*的格式保存。

7.3 配额管理

7.3.1 配额参数说明

ECS

参数	说明
CPU配额总量（核）	ECS产品配额可配置的CPU总数。
内存配额总量（G）	ECS产品配额可配置的内存总数。
数据盘配额总量（G）	ECS产品配额可配置的数据盘总数。

RDS（**包括主实例和只读实例**）

参数	说明
CPU配额总量（核）	RDS（MYSQL/SQLServer/PostgreSQL/PPAS）产品配额可配置的CPU总数。
内存配额总量（M）	RDS（MYSQL/SQLServer/PostgreSQL/PPAS）产品配额可配置的内存总数。
存储配额总量（G）	RDS（MYSQL/SQLServer/PostgreSQL/PPAS）产品配额可配置的存储总数。

SLB

参数	说明
外网IP配额总量（个）	SLB产品配额可配置的内网IP总数。
内网IP配额总量（个）	SLB产品配额可配置的外网IP总数。

OSS

参数	说明
OSS个数配额总量（个）	选择**不限制容量**后，表示不限制容量的OSS产品配额可配置的 bucket总数。
OSS容量配额总量（T）	选择**固定容量**后，表示固定容量的OSS产品配额可配置的 bucket总量。

7.3.2 查看配额

管理员可以查看各部门各区域云资源的总配额及余量情况。

操作步骤

1. 选择**云资源中心** > **配额管理**。
2. 在左边区域框中选择对应的部门，在右边区域框中选择对应的区域，查看对应部门的各云资源总配额及余量情况。

   各资源配额参数的相关说明请参见配额参数说明。

8 系统维护

8.1 管理部门

8.1.1 查看部门的项目

通过查看部门的项目，了解部门内的项目信息。

背景信息

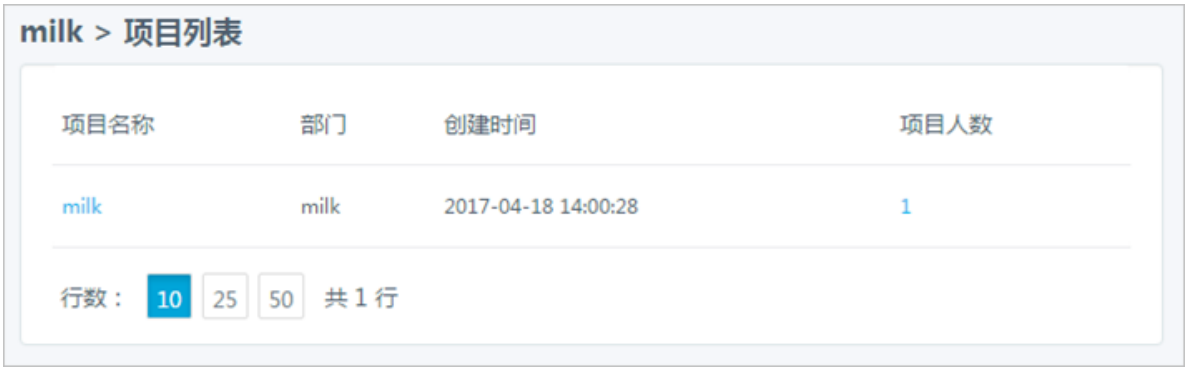
部门体现了企业或单位的组织树结构，一个部门可以有多个项目。

操作步骤

1. 进入**系统管理** > **部门管理**页面。
2. 选择要查看的部门。

   该部门的项目将显示在右侧界面中，如图 *21:* 部门项目列表所示。

   图 21: 部门项目列表



8.2 管理项目

项目是资源存放的容器，所有资源的申请及创建都需要在对应项目下。

8.2.1 查看项目详情

通过查看项目详情，了解项目的名称、ID、所属部门、创建时间及成员数量等基本信息。

操作步骤

1. 进入**系统管理** > **项目管理**页面。
2. 在待查看的项目后，单击，选择**查询详情**。

进入**项目详情**页面，如图 *22:* 项目详情所示。

图 22: 项目详情



8.2.2 查看项目成员

当您需要使用某个项目的资源时，您可以查看自己是否在项目成员列表中。

背景信息

项目下的成员对项目中的资源具有使用权限。

操作步骤

1. 进入**系统管理** > **项目管理**页面。
2. 在待查看的项目后，单击，选择**查询详情**。

   进入**项目详情**页面。
3. 选择**项目成员列表**，可以看到当前项目下所有的项目成员及其联系信息。

8.2.3 查看项目资源信息

当您需要使用某个云资源时，您可以在项目资源列表中查看本项目的资源信息。

背景信息

项目资源列表详细显示了当前项目下所拥有的云资源。

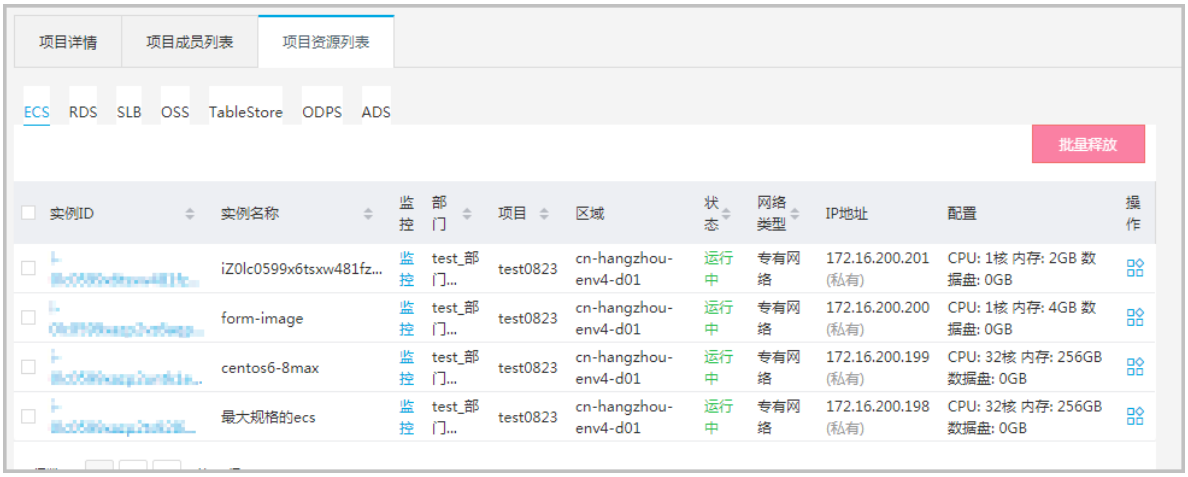
操作步骤

1. 进入**系统管理** > **项目管理**页面。
2. 在待查看的项目后，单击，选择**查询详情**。

   进入**项目详情**页面。
3. 选择**项目资源列表**页签。

4. 在**项目资源列表**中，查看当前项目下所有的云资源，如图 *23:* 项目资源列表所示。

图 23: 项目资源列表



5. 选择对应的云产品。
6. 在待查看的资源后，单击，选择**查询详情**，即可查看该资源的详细信息。

8.3 管理角色

角色是一组访问权限的集合，每个角色都有对应权限范围。一个用户可以拥有多个角色，这意味着该用户可以获得这些角色定义的所有权限。当需要对一组用户赋予相同的权限时，可以使用角色来授权。

8.3.1 权限说明

角色是一组访问权限的集合，在创建用户时，需要为该用户赋予不同的角色，以满足不同用户对系统的访问控制需求。

用户权限可以从以下两个方面进行衡量：

- 权限行为

  权限行为是指权限能做的具体的操作，包括读权限、写权限和操作操作。其中：

  - 读是指拥有的权限涉及模块的访问权限。
  - 写是指拥有的权限涉及模块的管理权限。
  - 操作是指拥有的权限涉及模块的使用权限。
- 维度

  维度是指权限作用的范围，包括全局、部门和项目。其中：

- 全局是指权限作用到涉及模块所有的资源或数据。
- 部门是指权限作用到用户所属部门以及下级部门的资源或数据。
- 项目是指权限作用到用户加入的项目的资源或数据。

云控制台缺省情况下有13种角色，角色中对应的权限说明如表 *18:* 权限说明表所示。

表 18: 权限说明表

权限名称	权限行为	维度	说明
云资源管理	读、写、操作	全局、部门、项目	管理云资源的权限，涉及云资源管理和云监控模块，这里的云资源包括云服务器、云数据库、负载均衡、对象存储、表格存储以及其他产品，不包括虚拟专有网、弹性外网 IP ，这些资源由专门的权限管理。 • 读权限可以查看云资源的信息和云监控（监控、报警项、报警信息）情况。 • 写权限可以创建和删除云资源，可以创建云资源的报警项。 • 操作权限可以修改和操作云资源，比如云服务器的停止和重启操作，也可以创建云资源的报警项。
VPC 管理	读、写	全局、部门	管理虚拟专有网和弹性外网 IP 的权限，涉及模块为虚拟专有网、弹性外网 IP ，其中： • 读权限可以查看虚拟专有网和弹性外网 IP 的信息。 • 写权限可以创建、修改、删除和操作虚拟专有网和弹性外网。 维度为全局可以管理所有的资源，维度为部门只能管理登录用户所属部门以及下级部门的资源。
大数据产品管理	涉及模块所有操作	全局、部门、项目	管理大数据产品的权限。
备份管理	读、写、操作	全局、部门、项目	-
报警人	读、写	默认维度为全局	管理云监控中报警人的权限，涉及模块为云监控中心的报警人。 • 读权限可以查看报警人和报警联系组的信息。 • 写权限可以创建、修改、删除报警人和报警联系组的权限。

权限名称	权限行为	维度	说明
			默认维度为全局。
用户和项目管理	读、写	全局、部门	管理用户以及项目的权限，涉及用户管理和项目管理模块，其中： • 读权限可以查看用户以及项目的信息。 • 写权限可以创建、修改、删除和操作用户和项目。 维度为全局可以管理所有的用户和项目；维度为部门只能管理登录用户所属部门以及下级部门的用户和项目。
角色管理	读、写	默认维度为全局	管理云控制台角色的权限，涉及角色管理模块。 • 读权限可以查看角色的信息。 • 写权限可以创建、修改和删除自定义角色。 默认维度为全局。
部门管理	读、写	全局、部门	管理部门、用户及项目的权限。
系统配置	涉及模块所有操作	默认维度为全局	管理系统配置的权限，涉及系统配置模块。 可以进行系统管理中的系统配置模块的所有操作，默认维度为全局。
日志管理	涉及模块所有操作	默认维度为全局	管理操作日志的权限，涉及日志管理模块。 默认维度为全局。
资源申请单申请	涉及模块所有操作	默认维度为全局	申请资源
资源申请单审核	涉及模块所有操作	默认维度为全局	审核资源申请单
资源申请单审批	涉及模块所有操作	默认维度为全局	审批资源申请单
资源生产	涉及模块所有操作	默认维度为全局	-
运维管理	涉及模块所有操作	默认维度为全局	管理云运维的权限，涉及运维中心模块。 可以进行云运维中心模块所有的操作，默认维度为全局。
报表管理	涉及模块所有操作	维度与云资源管理维度保持一致	管理报表的权限，涉及报表管理模块。 可以导出报表，导出的报表中的资源内容由进行导出操作的用户拥有的云资源管理权限的维度所定。

权限名称	权限行为	维度	说明
申报处理	涉及模块所有操作	默认维度为全局	处理申报的权限，涉及模块为故障申报中的所有申报。 默认维度为全局。
申报使用	涉及模块所有操作	请参考说明	提交申报和查看已提交申报的权限，涉及模块为故障申报中的提交申报以及已提交申报。 这个权限的维度比较特殊，在已提交申报中，只能看到由登录用户自己提交的申报信息。
计费管理	涉及模块所有操作	默认维度为全局	-
计费详情	涉及模块所有操作	默认维度为全局	-
大数据运维	涉及模块所有操作	默认维度为全局	-
大数据工作台	涉及模块所有操作	默认维度为全局	-

8.3.2 查看角色详情及角色下的用户

如果您不清楚某个角色有哪些权限，您可以在**角色管理**中查看各角色对应的角色权限。

操作步骤

1. 选择**系统管理** > **角色管理**。
2. （可选）输入角色关键字，单击**查询**。

   显示查询结果。
3. 选择待查看的角色，查看其角色权限。

   过长的角色权限将会被隐藏，要查看全部角色权限，可将鼠标移动到被隐藏的文字上方，将弹窗显示所有角色权限。

   各角色对应的权限说明请参见权限说明。
4. 单击**角色人数**一栏的数字，进入用户列表，可以查看某个角色下的所有用户。

8.4 管理用户

8.4.1 查看用户基本信息

查看用户基本信息，了解用户所属部门、角色及联系信息。

操作步骤

1. 进入**系统管理** > **用户管理**。
2. 选择**系统用户**页签。
3. 在待查看的用户后，单击，选择**用户信息**，可以查看用户的基本资料。

8.4.2 获取个人账号的AccessKey

如果您开通了OSS并需要访问OSS时，您需要获取个人账号的AccessKey ID和AccessKey Secret，用于登录授权。

操作步骤

1. 在系统界面右上角，单击，选择**个人信息**。
2. 单击**AccessKey**。

   系统弹出**获取AccessKey**的提示框。
3. 在**获取AccessKey**对话框中，单击**确定**。

   系统显示当前用户所属的区域、部门、AccessKey ID及AccessKey Secret信息。

8.5 管理登录策略

8.5.1 查看登录策略

您可以通过查看登录策略，了解某个用户允许登录的时间和地址。

背景信息

云控制台提供服务时，会自动生成一个默认策略，默认不限制登录时间与登录地址。

操作步骤

1. 进入**系统管理** > **登录策略管理**页面。
2. （可选）输入需要查看的策略名，单击**查询**。

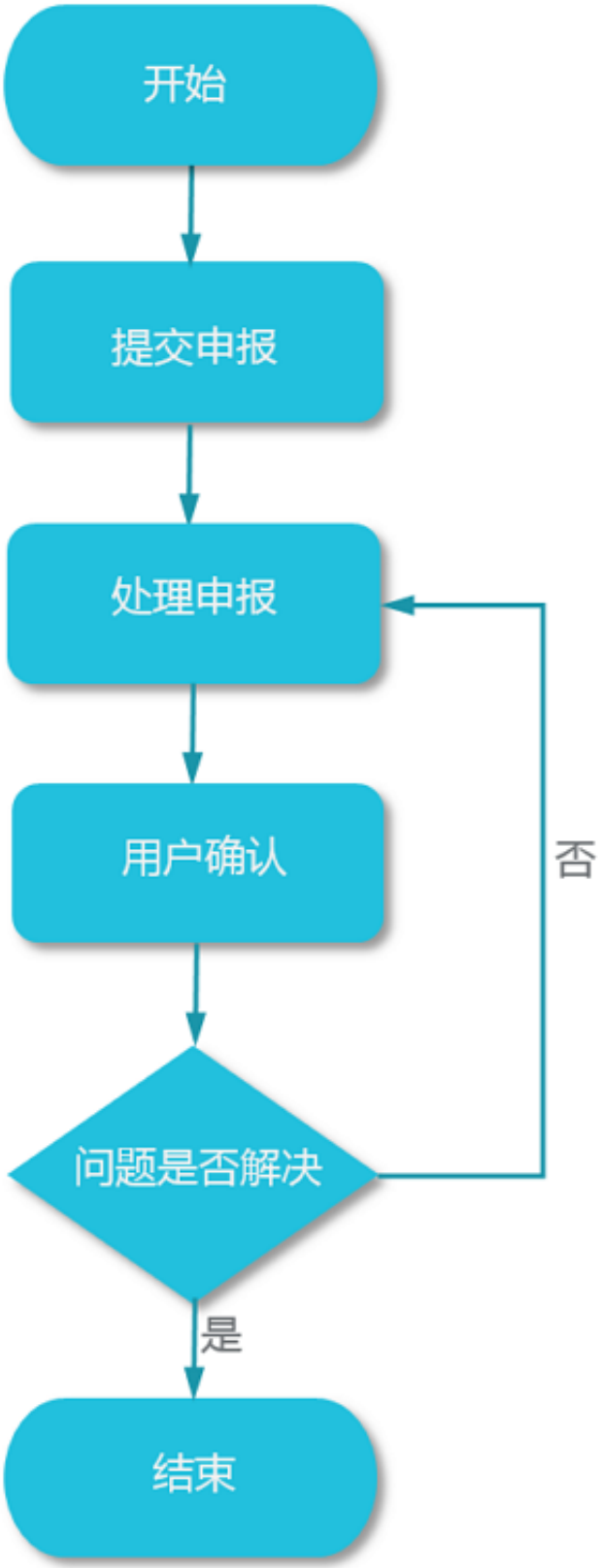
   显示查询结果。
3. 查看对应的登录策略，包括允许用户登录的时间及地址。

9 管理工单

9.1 工单申报流程

当资源出现异常或者有其他问题时，资源使用人可以提交工单申报，以便让系统管理员处理。工单申报流程图如图 *24:* 工单申报流程所示。

图 24: 工单申报流程



说明: 由于管理员无法时时处理问题，如果有紧急问题请直接联系阿里云技术服务支持。

1. 资源使用人提交申报。

2. 系统管理员根据问题描述，通知运维人员处理问题，问题解决后，系统管理员处理申报。
3. 用户确认申报是否已经解决。如果仍未解决则返回给系统管理员继续处理，如果已解决则关闭该申报。

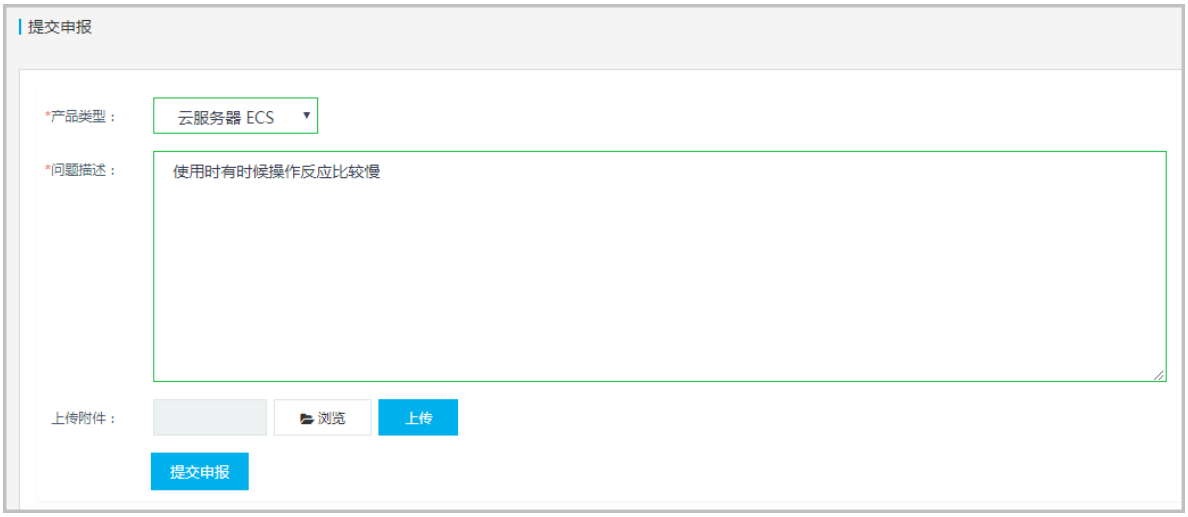
9.2 提交申报

您在使用云控制台过程中，出现的任何问题都可以提交申报将问题反馈给系统管理员解决。

操作步骤

1. 选择**流程管理** > **工单管理** > **提交申报**，进入**提交申报**页面，如图 *25: 提交申报*所示。

图 25: 提交申报



2. 配置提交申报参数，相关参数如表 *19: 参数说明*所示。

表 19: 参数说明

参数	说明
产品类型	表示云资源类型。例如，在使用过程中 ECS 出现了问题，则此处应选择 ECS 。
问题描述	详细清晰的问题描述可以帮助管理员快速定位并解决问题。在填写问题描述时，建议涵盖以下几个内容：谁发现（Who）、什么时候发现（When）、在哪里出现（Where）、什么产品出现了什么问题（What）、问题的影响程度（HowMuch）、问题截图（Evidence）。

3. 单击**浏览**，选择待上传的问题附件。

缺省情况下，不支持上传附件。如需上传附件，请联系管理员先完成附件存储位置的配置，再执行上传附件操作。附件存储路径配置请参见《管理员指南》中的**配置存储服务**。

说明: 上传的附件支持 *.txt*、*.png*、*.doc* 和 *.docx* 格式，且大小不超过 30M 。

4. 单击**提交申报**。

   完成故障申报后，您可以通过选择**流程管理** > **工单管理** > **已提申报**，查看已提交申报的处理情况。

9.3 确认故障是否解决

提交申报后，您可以随时查看申报的处理进度。

背景信息

如果您提交的故障工单已解决，系统将以短信或邮件通知您进行确认。

操作步骤

1. 选择**流程管理** > **工单管理** > **已提申报**，进入**已提申报**页面。

   显示您所申报的所有工单。

   工单主要有以下几种状态：

   - 待运维人员处理：指工单还未受理。
   - 运维人员处理中：指工单已受理。
   - 待您确认：指工单已处理完毕。
   - 工单已关闭

2. （可选）输入工单编号、搜索关键字、工单状态及起始日期，单击**查询**。

   显示查询结果。

3. 选择状态为**待您确认**的工单，单击，选择**详情**。

4. 在**申报详情**页面，查看留言中的问题处理描述和附件，确认故障是否已经排除。

   - 如果故障确认已解决，单击**确认故障已解决**，然后对故障结果进行评价。
   - 如果故障并未解决，单击**继续留言**，将故障现状反馈给系统管理员。

9.4 删除工单

如果您提交的申报有误，您可以删除工单。

背景信息

说明: 如果您的工单已经走到确认故障解决的流程，建议您先确认故障已解决并给出评价，不要因为故障已解决而简单地删除工单。

操作步骤

1. 选择**流程管理** > **工单管理** > **已提申报**，进入**已提申报**页面。
2. 在工单列表页面，选择工单所在行，单击，选择**删除**。

   删除不需要的工单。

   说明: 您也可以在**申报详情**页面，单击**删除工单**，删除不需要的工单。

10 个性化设置

登录云控制台后，您可以修改菜单显示配置及登录密码。

10.1 修改登录密码

为提高安全性，请及时修改系统登录密码。

操作步骤

1. 在系统界面右上角，单击，选择**个人信息**。

   进入**个人信息**页面。

2. 选择**修改密码**，输入原始密码和新密码，如图 *26: 修改密码*所示。

   图 26: 修改密码



3. 单击**提交修改**。

10.2 修改头像

为了便于识别，您可以上传自定义头像。

操作步骤

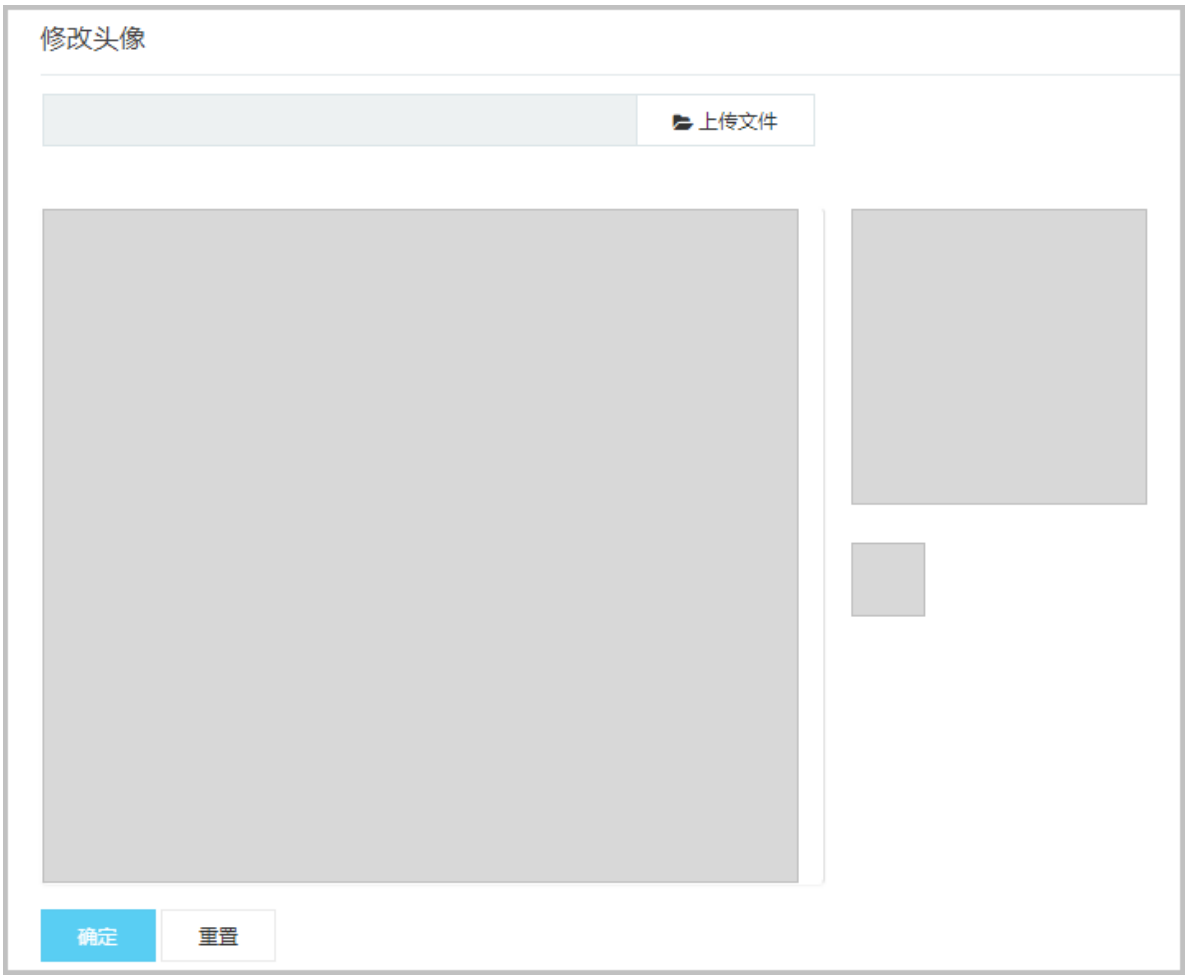
1. 在系统界面右上角，单击，选择**个人信息**。

   进入**个人信息**页面。

2. 单击**修改头像**。

   显示**修改头像**区域框，如图 *27: 修改头像*所示。

   图 27: 修改头像



3. 在**修改头像**区域框中，单击**上传文件**。
4. 选择待上传的图片，单击**打开**，待上传的图片会显示在下面的预览框中。

   如果需要更换图片，可以单击**重置**，重新选择其他图片。
5. 确认后，单击**确认**，头像修改成功。

11 云服务器ECS

11.1 产品概述

本节对云服务器的基本概念进行简单的介绍。

云服务器（Elastic Compute Service，简称ECS）是处理能力可弹性伸缩的计算服务，它的管理方式比物理服务器更简单高效。根据业务需要，您可以随时创建实例、扩容磁盘或释放任意多台云服务器实例。

云服务器ECS实例（以下简称ECS实例）是一个虚拟的计算环境，包含CPU、内存等最基础的计算组件，是云服务器呈献给每个用户的实际操作实体。ECS实例是云服务器最为核心的概念。其他的资源，比如磁盘、IP、镜像、快照等，只有与ECS实例结合后才能使用。

云服务器ECS示意图如图 *28: ECS 示意图*所示。

图 28: ECS 示意图



11.1.1 实例的规格

实例规格见实例规格表。

表 20: 实例规格表

实例规格族	规格类型代码	CPU（Core）	内存（GB）
SN1	ecs.sn1.medium	2	4
	ecs.sn1.large	4	8
	ecs.sn1.xlarge	8	16
	ecs.sn1.3xlarge	16	32
	ecs.sn1.7xlarge	32	64
SN2	ecs.sn2.medium	2	8
	ecs.sn2.large	4	16
	ecs.sn2.xlarge	8	32
	ecs.sn2.3xlarge	16	64
	ecs.sn2.7xlarge	32	128
	ecs.sn2.13xlarge	56	224
N4	ecs.n4.small	1	2
	ecs.n4.large	2	4
	ecs.n4.xlarge	4	8
	ecs.n4.2xlarge	8	16
	ecs.n4.4xlarge	16	32
	ecs.n4.8xlarge	32	64
MN4	ecs.mn4.small	1	4
	ecs.mn4.large	2	8
	ecs.mn4.xlarge	4	16
	ecs.mn4.2xlarge	8	32
	ecs.mn4.4xlarge	16	64
	ecs.mn4.8xlarge	32	128
E4	ecs.e4.small	1	8
	ecs.e4.large	2	16

实例规格族	规格类型代码	CPU（Core）	内存（GB）
	ecs.e4.xlarge	4	32
	ecs.e4.2xlarge	8	64
	ecs.e4.4xlarge	16	128
XN4	ecs.xn4.small	1	1
GN4	ecs.gn4-c4g1.xlarge	4	30
	ecs.gn4-c8g1.2xlarge	8	60
	ecs.gn4.8xlarge	32	48
	ecs.gn4-c4g1.2xlarge	8	60
	ecs.gn4-c8g1.4xlarge	16	60
	ecs.gn4.14xlarge	56	96
GN5	ecs.gn5-c4g1.xlarge	4	30
	ecs.gn5-c8g1.2xlarge	8	60
	ecs.gn5-c4g1.2xlarge	8	60
	ecs.gn5-c8g1.4xlarge	16	120
	ecs.gn5-c28g1.7xlarge	28	112
	ecs.gn5-c8g1.8xlarge	32	240
	ecs.gn5-c28g1.14xlarge	56	224
	ecs.gn5-c8g1.14xlarge	56	480
GN5I	ecs.gn5i-c2g1.large	2	8
	ecs.gn5i-c4g1.xlarge	4	16
	ecs.gn5i-c8g1.2xlarge	8	32
	ecs.gn5i-c16g1.4xlarge	16	64
	ecs.gn5i-c28g1.14xlarge	56	224
GA1	ecs.ga1.2xlarge	8	20
	ecs.ga1.4xlarge	16	40
	ecs.ga1.8xlarge	32	80
	ecs.ga1.14xlarge	56	160

您可以根据业务需要，使用相应规格的云服务器实例。

11.1.2 实例的生命周期

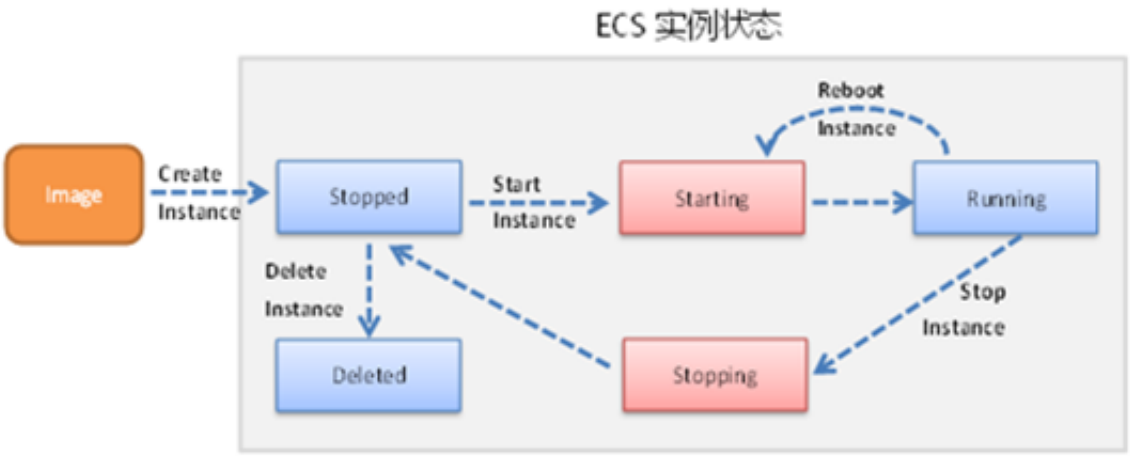
实例的生命周期是从创建开始到最后释放。在这个生命周期中，实例有其固有的几个状态，如下表所示。

表 21: 实例的生命周期

状态	状态属性	解释	API 的对应状态
创建中	中间状态	实例被创建后，在进入运行中之前的状态。如果长时间处于该状态，则说明出现异常。	**Stopped**
启动中	中间状态	实例在控制台或通过 API，被重启、启动等操作后，在进入运行中之前的状态。如果长时间处于该状态，则说明出现异常。	**Starting**
运行中	稳定状态	实例正常运行状态。在这个状态的实例可以运行您的业务。	**Running**
停止中	中间状态	实例在控制台或通过 API，被停止操作后，在进入已停止之前的状态。如果长时间处于该状态，则说明出现异常。	**Stopping**
已停止	稳定状态	实例被正常停止。在这个状态下的实例，不能对外提供业务。	**Stopped**
重置中	中间状态	实例在控制台或通过 API，被重置系统盘或/和数据盘后，在进入运行中之前的状态。如果长时间处于该状态，则说明出现异常。	**Stopped**
更换系统中	中间状态	实例在控制台或通过API，被更换操作系统等操作后，在进入运行中之前的状态。如果长时间处于该状态，则说明出现异常。	**Stopped**

表中描述了控制台中的状态和 API 状态的对应关系，API 状态图如下。

图 29: API 状态图



11.2 使用前须知

11.2.1 禁忌

- 不要随意升级云服务器的内核和操作系统版本。
- 不要开启 SELinux。
- 不要卸载 PVDriver。
- 不要随意修改网卡 MAC 地址。

11.2.2 建议

- 对于 4 GB 及以上内存的云服务器，请选择 64 位操作系统（ 32 位操作系统存在 4 GB 的内存寻址限制）。
- Windows 32 位操作系统支持最高 CPU 核数为 4 核。
- 为保证服务的连续性，避免因宕机迁移而导致服务不可用，建议把相关软件设置成开机启动。

11.2.3 限制

- 暂不支持虚拟化软件安装和再进行虚拟化（如安装使用 VMware）。
- 暂不支持声卡应用、直接加载外接硬件设备（如硬件加密狗、U 盘、外接硬盘、银行 U key 等）。
- 暂不支持 SNAT 等 IP 包地址转换服务。您可以使用自己搭建的 VPN 或者代理方式来实现。
- 暂不支持多播协议。如果需要使用多播，建议改为使用单播点对点方式。

11.2.4 Windows 操作系统须知

- 不要关闭 Windows 系统自带的 shutdownmon.exe 进程。关闭这个进程可能会使服务器重启时间变长。
- 不要重命名、删除或禁用 Windows 下的 Administrator 账号，以免影响服务器使用。
- 不建议使用虚拟内存。
- 控制台修改密码针对的是 Administrator 账号。如果 Administrator 账号被删除或者重命名，单击修改密码按钮会自动创建 Administrator 账号。
- 如果您修改了计算机名，一定要同步注册表中的下列相关键值，否则您无法成功修改计算机名，从而导致无法安装某些第三方程序。注册表中需要修改的键值如下：

```
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\ComputerName
\ActiveComputerName
```

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\ComputerName
\ComputerName
```

11.2.5 Linux 操作系统须知

- 不要修改 Linux 服务器下默认的 */etc/issue* 文件内容。修改该文件会导致云控制台的功能按钮无法正常使用。
- 不要随意更改分区下目录的权限，尤其是 */etc/sbin/bin/boot/dev/usr/lib* 等目录权限。如果权限更改不当会导致系统出现异常。
- 不要重命名、删除或禁用 Linux 下的 root 账号。
- 不要编译 Linux 系统的内核，或对内核进行任何其他操作。
- 不建议使用 Swap 分区。
- 不要开启 NetWorkManager 服务。该服务会与系统内部网络服务发生冲突，导致网络异常。

11.3 快速入门

11.3.1 连接实例

创建 ECS 实例后，您可以连接实例，在实例中安装应用软件。

注意: Windows 实例的用户名是 Administrator，Linux 实例的用户名是 root。

您有 2 种方式连接和管理您的实例：

- 使用远程连接工具连接 ECS 实例。

  说明: 如果您使用远程连接工具，只有拥有外网 IP 的实例才能被远程连接。

- 使用阿里云的远程连接功能 VNC 连接 ECS 实例。

11.3.1.1 Linux或Mac OS X环境中使用SSH命令连接Linux实例

操作步骤

1. 输入命令：ssh root@实例IP。
2. 输入该实例 root 用户的密码，即可完成登录。

11.3.1.2 Windows环境中通过远程连接工具连接Linux实例

前提条件

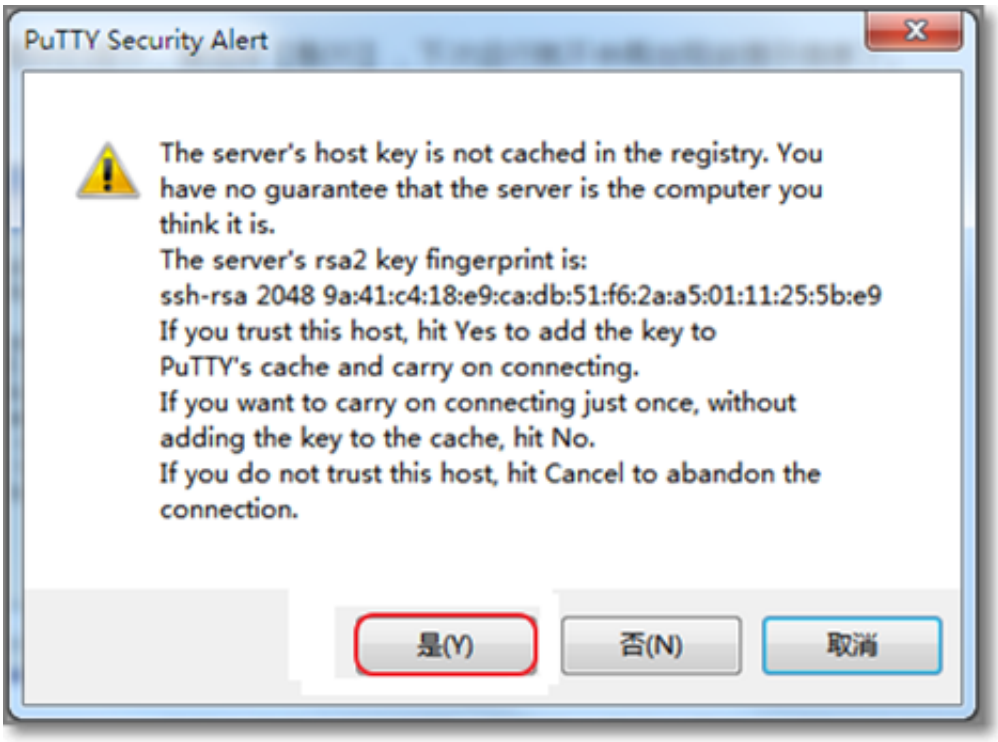
远程连接工具的用法大同小异。本文档以 Putty为例，介绍如何远程连接实例。Putty下载地址：*http://www.chiark.greenend.org.uk/~sgtatham/putty/* 。

操作步骤

1. 下载并安装 Windows SSH 和 Telnet 客户端 Putty。
2. 打开 Putty 客户端，完成如下设置：
   - 在 **Host Name (or IP address)** 中输入您要连接的实例 IP。
   - **Port** 使用默认端口 22。
   - **Connection Type** 选择 **SSH**。
   - **Saved Session** 中输入实例名称，再单击 **Save** 按钮。保存成功后，Putty 软件就能记住实例名称和匹配的 IP 地址，也不需要每次输入 IP 地址。
3. 单击 **Open** 按钮开始连接实例。

   首次连接时，屏幕上会出现*PuTTY安全警示*所示的提示。单击 **是** 按钮。

   图 30: PuTTY安全警示



4. 在屏幕上输入用户名 **root**，按回车键。
5. 输入该实例用户的密码，再按回车键。如果屏幕上显示类似以下所示的语句，表示您已经成功连接实例。

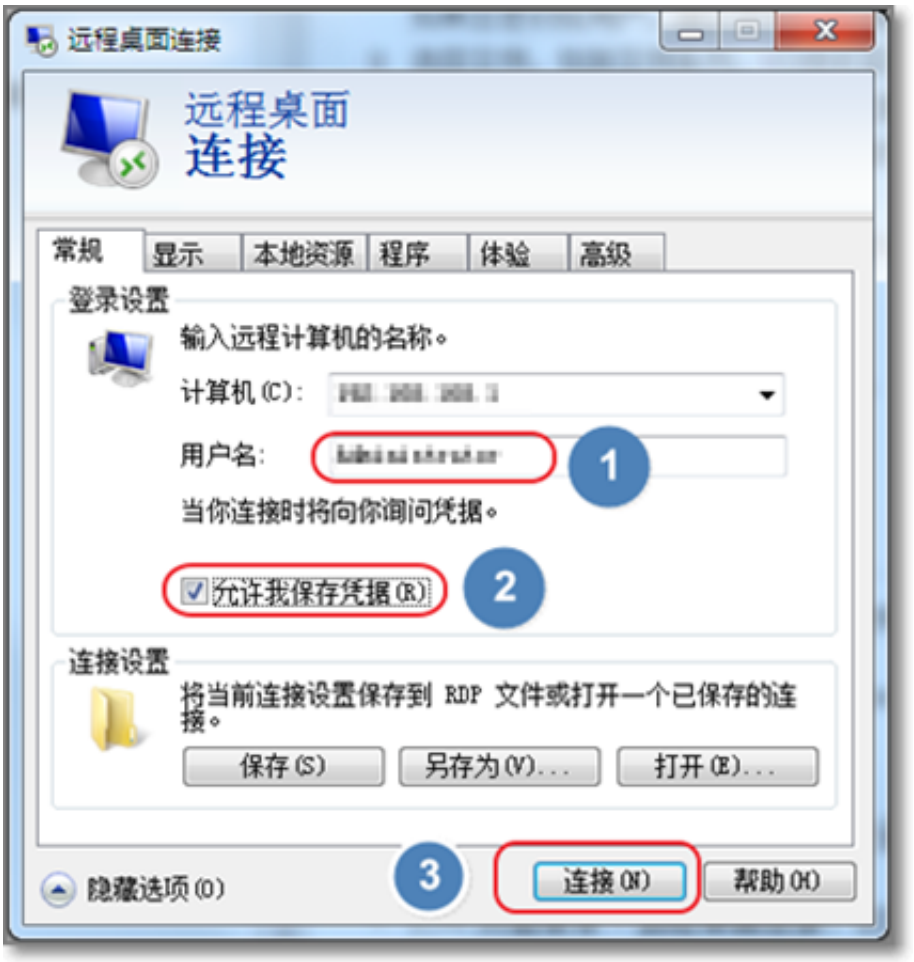
```
Welcome to aliyun Elastic Compute Server!
```

11.3.1.3 Windows环境中通过远程桌面连接Windows实例

本文介绍了如何通过 Windows 自带的远程桌面连接功能连接 Windows 实例。

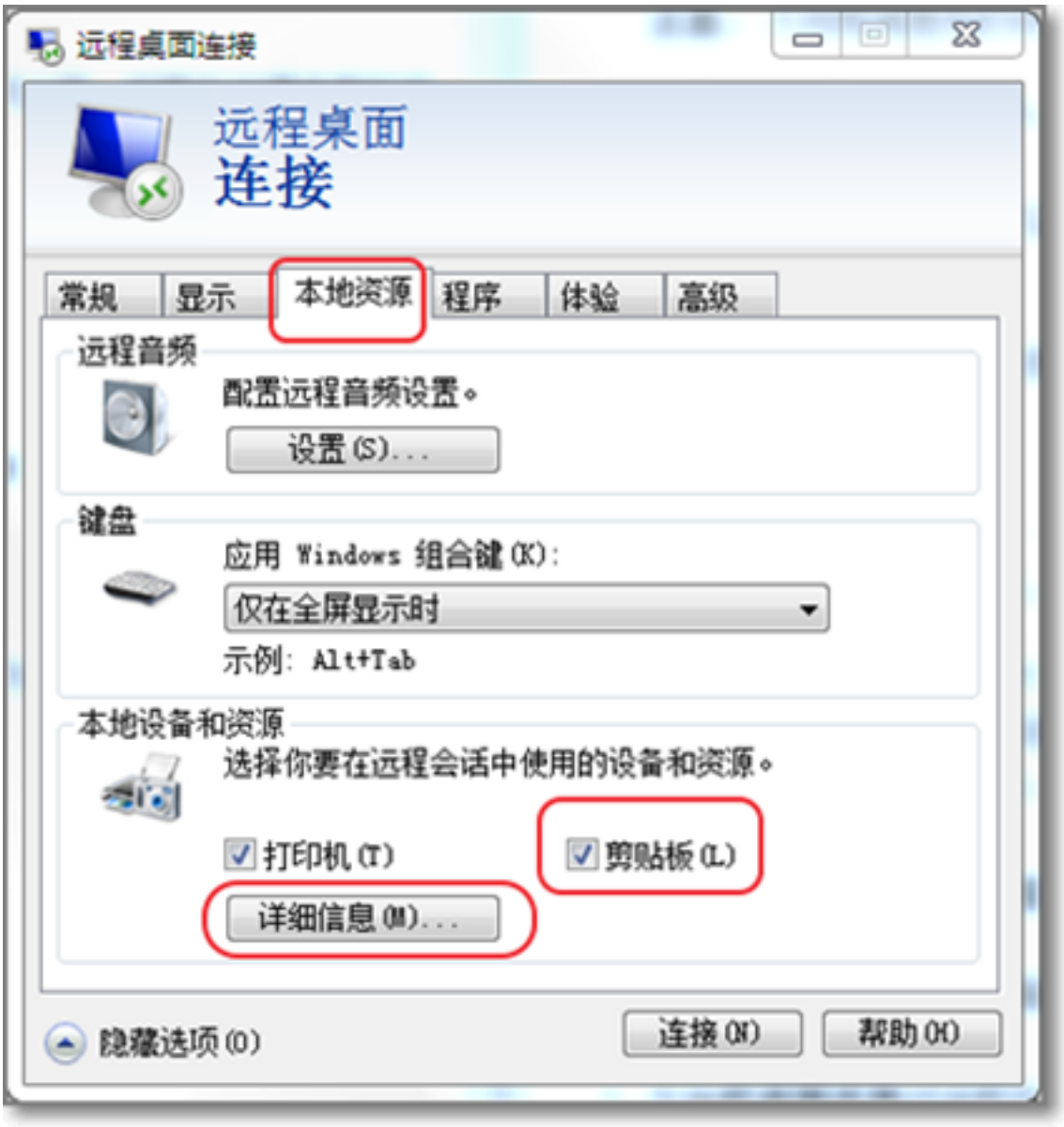
操作步骤

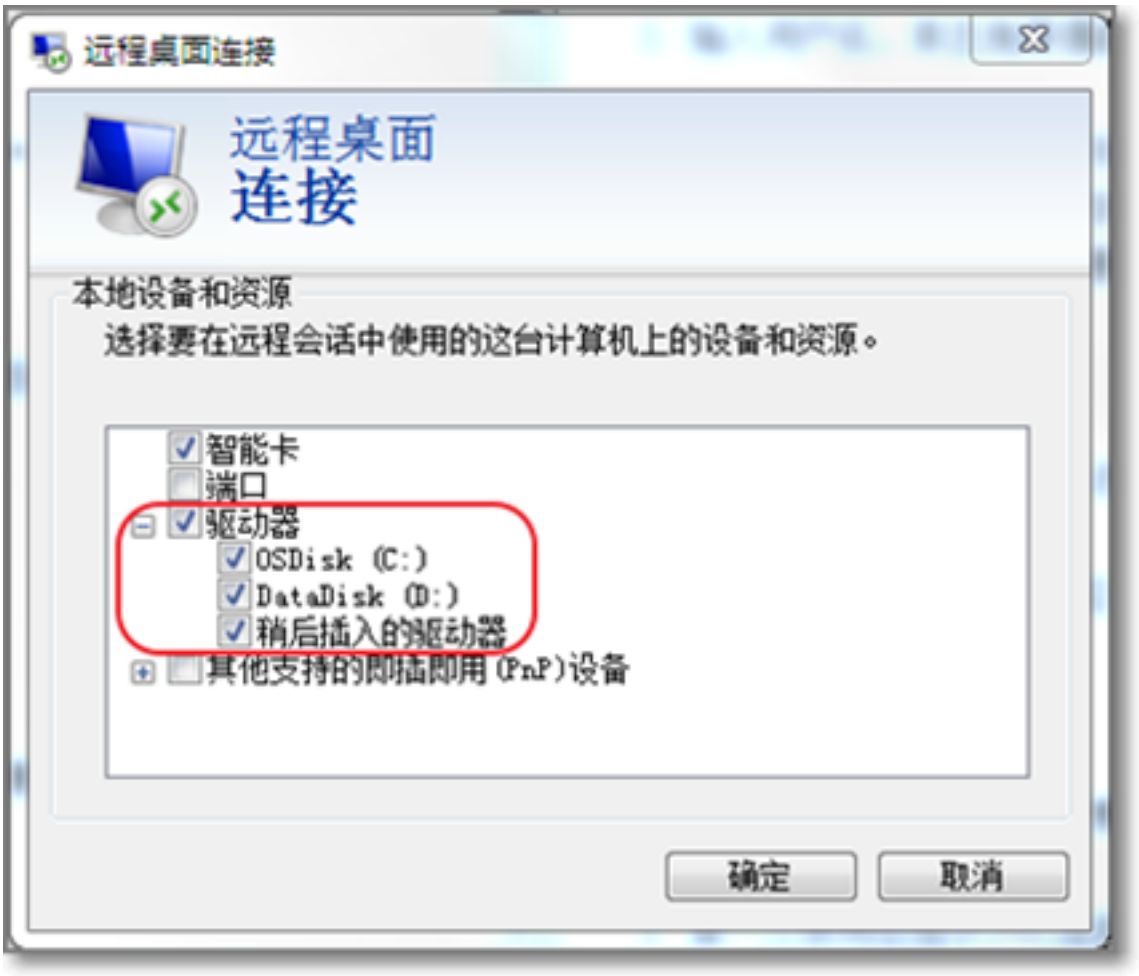
1. 使用其中一种方法启动远程桌面连接：
   - 单击 **开始** 按钮，在搜索框里输入 **远程桌面连接**，在提供的列表里单击 **远程桌面连接**。
   - 在搜索框里输入 **mstsc**，在提供的列表里单击 **mstsc**。
   - 使用快捷键 **Windows 徽标+R**，启动 **运行** 对话框；输入 **mstsc** 后按回车键，启动远程桌面连接。
2. 在 **远程桌面连接** 对话框中，输入实例的外网 IP 地址，并单击 **显示选项(O)**。
3. 输入用户名，默认为 **Administrator**。如果您希望以后不再手动输入密码，可以勾选 **允许我保存凭据(R)**。如果不需要再做其他设置，可以直接单击 **连接** 按钮连接实例。您也可以按以下描述完成其他配置后再连接实例。



您也可以按以下描述完成其他配置后再连接实例：

- 如果您要从本地复制文字信息到实例中，可以单击 **本地资源** 选项卡，并勾选 **剪贴板**。
- 如果您要从本地复制文件到实例中，可以单击 **本地资源** 选项卡；单击 **详细信息** 按钮；并勾选需要的 **驱动器**，选择文件在本地存放的盘符。完成设置后，单击 **确定** 按钮。


远程桌面连接
远程桌面
连接
常规
显示
本地资源
程序
体验
高级
远程音频
配置远程音频设置。
设置(S)...
键盘
应用 Windows 组合键(K):
仅在全屏显示时
示例: Alt+Tab
本地设备和资源
选择你要在远程会话中使用的设备和资源。
打印机(T)
剪贴板(L)
详细信息(M)...
隐藏选项(O)
连接(N)
帮助(H)



- 您还可以单击 **显示** 选项卡，调整远程桌面的大小，一般考虑使用全屏。

4. 在弹出的对话框中输入 Windows 实例账号 **Administrator** 对应的密码后，单击 **确定** 按钮，开始连接实例。

   如果 **远程桌面连接** 窗口显示一个 Windows 桌面，那么表示您已经成功地连接到实例，可以开始操作实例了。

11.4 管理实例

实例是能够为您的业务提供计算服务的最小单位，它以一定的规格来为您提供相应的计算能力。

11.4.1 创建实例

阿里云为您提供了多种创建实例的方法，关于创建实例的详细操作步骤，请参考快速入门 。

11.4.1.1 创建GPU可视化计算ga1实例

阿里云为ga1规格族实例提供以下版本的镜像：

- CentOS 7.3版

- Ubuntu16.04版
- Windows 2008 R2中文版
- Windows 2008 R2英文版

创建实例

您可以按照快速入门中创建实例的描述创建ga1规格族实例。在选择配置时，您需要注意以下几点：

- 网络：选择**专有网络**。因为目前GPU渲染型ga1规格族实例只支持专有网络（VPC）。
- 实例：选择**系列 III**的**GPU 可视化计算ga1**。

注意事项

- 卸载或删除GPU驱动相关组件造成驱动不能正常工作的情况，需要通过更换系统盘的方式恢复GPU 的相关功能。

  说明: 此操作会造成 数据丢失。

- 创建GPU可视化计算ga1实例时，选择其它的镜像会造成实例的驱动不能正常工作，用户需要通过 更换系统盘的方式重新选择预安装相应的镜像。
- 对于Windows系统，GPU驱动安装生效后，阿里云控制台的**远程连接**功能不可用，**管理终端**始终显示黑屏或停留在启动界面。请通过其它协议进入系统，如 Windows 自带的 远程桌面连接#*RDP*#。
- Windows自带的远程连接（RDP）协议不支持DirectX、OpenGL等相关应用，您需要自行安装VNC服务和客户端，或其它支持的协议，例如PCOIP、XenDeskop HDX 3D等。

11.4.1.2 创建GPU计算型gn4规格族实例

实例规格详情

gn4实例规格族的详细规格，参见实例的规格。

创建gn4规格族实例

您可以按照快速入门中的创建实例的描述创建gn4规格族实例，创建时需要注意以下配置：

- 网络：选择 专有网络。gn4实例只支持专有网络。
- 实例类型：选择**异构计算** > **GPU 计算** >gn4。
- 带宽：根据需要选择带宽。



说明: 如果使用Windows 2008 R2镜像，GPU驱动安装生效后，您将不能正常使用控制台的 远程连接 功能连接gn5i实例，所以，您需要将带宽设置为一个非零值或者创建实例后 绑定弹性公网IP地址。

- 镜像：根据需要选择镜像。

下载并安装GPU驱动

您需要为实例安装GPU驱动。您应先根据以下步骤下载GPU驱动。

1. 到NVIDIA官方网址下载实例的镜像版本对应的驱动。下载地址：*http://www.nvidia.com/Download/index.aspx?lang=cn* 。
2. 手动查找适用于实例的驱动程序。选择的项填写说明如下：
   - 产品类型：Tesla
   - 产品系列：M-Class
   - 产品家族：M40
   - 操作系统：根据实例的镜像选择对应的版本。
3. 单击**搜索**按钮。
4. 确认无误后，单击**下载**按钮。
5. 按下载页的**其他信息**描述安装GPU驱动。

注意事项

对于Windows 2008 R2及以下版本，GPU驱动安装生效后，控制台的**远程连接**功能不可用，**管理终端**会始终显示黑屏或停留在启动界面。请您通过其它协议进入系统，如Windows自带的 “远程连接（RDP）” 。

Windows自带的远程连接（RDP）协议不支持DirectX、OpenGL等相关应用，您需自行安装VNC服务和客户端，或其它支持的协议，例如PCOIP、XenDeskop HDX 3D等。

11.4.1.3 创建GPU通用计算型gn5规格族实例

实例规格详情

gn5实例规格族的详细规格，参见实例的规格。

创建gn5规格族实例

您可以按照快速入门中的创建实例的描述创建gn4规格族实例，创建时需要注意以下配置：

- 地域：目前支持的地域包括华东1、华东2、华北2、华南1 、美国东部1（弗吉尼亚）、香港、亚太东南1、欧洲中部1、 美国西部1、 亚太东南2。
- 网络：选择**专有网络**。gn5实例只支持专有网络。
- 带宽：根据需要选择带宽。
- **镜像**：根据需要选择镜像。

下载并安装GPU驱动

您需要为实例安装GPU驱动。您应根据以下步骤下载GPU驱动。

> **说明:** 对于Windows 2008 R2及以下版本 ，GPU驱动安装生效后，如果您通过控制台的**远程连接**功能进入**管理终端**，**管理终端**会始终显示黑屏或停留在启动界面。这时，您需要通过其它协议远程连接ECS实例 ，如Windows自带的远程连接（RDP）。

1. 到NVIDIA官方网址下载实例的镜像版本对应的驱动。下载地址：*http://www.nvidia.com/Download/index.aspx?lang=cn* 。
2. 手动查找适用于实例的驱动程序。选择项填写说明如下：
   - 产品类型：Tesla
   - 产品系列：P-Series
   - 产品家族：Tesla P100
   - 操作系统：根据实例的镜像选择对应的版本。

     > **说明:**
     > - 如果下拉列表中没有显示服务器操作系统，请单击下拉列表底部的**选择所有操作系统**。
     > - 如果实例使用的Linux镜像不在列表中，可以选择**Linux 64-bit**。
3. 单击**搜索**按钮。
4. 确认无误后，单击**下载**按钮。
5. 按下载页的**其他信息**描述安装GPU驱动。

注意事项

如果使用Windows 2008 R2镜像，GPU驱动安装生效后，您将不能正常使用控制台的**远程连接**功能连接gn5实例，所以，您需要将带宽设置为一个非零值或者创建实例后绑定弹性公网IP。

11.4.1.4 创建GPU通用计算型gn5i规格族实例

实例规格详情

gn5i实例规格族的详细规格，参见实例的规格。

创建gn5i规格族实例

您可以按照快速入门中的创建实例的描述创建gn4规格族实例，创建时需要注意以下配置：

- 地域：目前只支持华北2。
- 网络：选择**专有网络**。gn5i实例只支持专有网络。
- 带宽：根据需要选择带宽。

  说明: 如果使用Windows 2008 R2镜像，GPU驱动安装生效后，您将不能正常使用控制台的**远程连接**功能连接gn5实例，所以，您需要将带宽设置为一个非零值或者创建实例后绑定弹性公网IP。
- 镜像：根据需要选择镜像。

下载并安装GPU驱动

您需要为实例安装GPU驱动。您应根据以下步骤下载GPU驱动。

1. 到NVIDIA官方网址下载实例的镜像版本对应的驱动。下载地址：*http://www.nvidia.com/Download/index.aspx?lang=cn* 。
2. 手动查找适用于实例的驱动程序。选择项填写说明如下：
   - 产品类型：Tesla
   - 产品系列：P-Series
   - 产品家族：Tesla P100
   - 操作系统：根据实例的镜像选择对应的版本。

     说明:
     - 如果下拉列表中没有显示服务器操作系统，请单击下拉列表底部的**选择所有操作系统**。
     - 如果实例使用的Linux镜像不在列表中，可以选择**Linux 64-bit**。
3. 单击**搜索**按钮。
4. 确认无误后，单击**下载**按钮。
5. 按下载页的**其他信息**描述安装GPU驱动。

注意事项

对于Windows 2008 R2及以下版本 ，GPU驱动安装生效后，如果您通过控制台的**远程连接**功能进入**管理终端**，**管理终端**会始终显示黑屏或停留在启动界面。这时，您需要通过其它协议远程连接ECS实例 ，如Windows自带的远程连接（RDP）。

11.4.2 查看实例

您可以通过ECS云控制台查看您所拥有的所有实例以及实例的详细信息。

操作步骤

1. 登录*ECS*云控制台，在**概览**页面中查看实例总数和运行实例总数。

   说明:

   - 在云控制台**首页**，您也可以单击**云实例个数**直接进入**实例**页面进行查看。
   - 在**实例**页面右上角，单击**设置**，您可以在**自定义列表项**框中，在需显示的内容项前打钩，单击**确定**。如图 *31:* 设置自定义列表所示。

   图 31: 设置自定义列表

2. 在**实例**页面中，选择**部门**、**区域**或输入**实例名称**，单击**查询**，找到对应的实例。
3. 在相应实例的**操作**列中，选择**查询详情**，进入**实例详情**页面，查看实例具体信息。

   说明: 在**实例**页面下，您也可以直接单击实例ID，进入**实例详情**页面。

11.4.3 编辑实例

针对已创建的实例，您可以在控制台修改实例的ID、名称和具体描述。

操作步骤

1. 登录*ECS*云控制台，找到待编辑的 ECS 实例。
2. 在相应实例的**操作**列中，选择**管理** > **编辑**。
3. 在弹出的对话框中，您可以编辑实例的**ID**、**名称**和**描述**，最后单击**确定**。如图 *32: 编辑ECS*所示。

图 32: 编辑ECS



11.4.4 停止、重启或启动实例

在云控制台中，您可以像操作真实的服务器一样，停止、重启或启动实例。

操作步骤

1. 登录*ECS*云控制台。
2. 在**实例**页面中，选择待操作的实例。
3. 在相应实例的**操作**中，单击**管理**，在下拉菜单中选择**重启**、**停止**或**启动**实例。
   - 停止和重启操作只能在实例处于 **运行中** 时进行，而启动操作只能在实例处于 **已停止** 时进行。

     注意: 停止和重启会造成您的实例停止工作，从而中断您的业务，请谨慎执行这些操作。
   - 您也可以在相应实例后，选择**管理** > **查询详情**，在实例详情页面中，单击右上方的**重启**、**停止**或**启动**按钮，完成重启、停止或启动实例操作。

11.4.5 修改实例密码

操作步骤

1. 登录*ECS*云控制台，找到需修改密码的实例。
2. 在相应实例的 **操作** 列中，选择**管理** > **查询详情**。
3. 在**实例详情**页面中，单击**修改密码**。
4. 在弹出的对话框中，输入新的**登录密码**和**确认密码**，单击**提交**。如图 *33:* 重置密码所示。

图 33: 重置密码



注意: 重置实例密码后，只有在控制台重启实例才能生效。

11.4.6 登录VNC

操作步骤

1. 登录*ECS* 云控制台，找到待登录VNC的实例。
2. 在相应实例的**操作**列中，选择**管理** > **查询详情**。
3. 在**实例详情**页面中，单击**登录VNC**。
4. 在弹出的对话框中，输入**登录密码**，单击**Connect**。

11.4.7 修改VNC密码

操作步骤

1. 登录*ECS*云控制台，找到需修改VNC密码的实例。

2. 在相应实例的 **操作** 列中，选择**管理** > **查询详情**。
3. 在**实例详情**页面中，单击**修改VNC密码**。
4. 在弹出的对话框中，输入新的**登录密码**和**确认密码**，单击**提交**。如图 *34: 修改VNC登录密码*所示。

图 34: 修改VNC登录密码



注意: 修改VNC密码后，只有在控制台重启实例才能生效。

11.4.8 加入安全组

操作步骤

1. 登录*ECS*云控制台，找到待加入安全组的实例。
2. 在相应实例的**操作**列中，选择**管理** > **查询详情**。
3. 在顶部导航栏中选择**实例安全组**，单击**加入安全组**。
4. 在弹出的对话框中，选择待加入的安全组名称，单击**确定**。

说明:

您还可以通过以下步骤将实例加入安全组：

1. 在云控制台中，选择**云资源中心** > **云产品管理** > **云服务器** > **安全组**，进入安全组管理页面。
2. 在相应实例的**安全组ID/名称**中，单击安全组ID/名称。
3. 在**安全组内实例**页面中，单击**ECS移入**加入安全组。

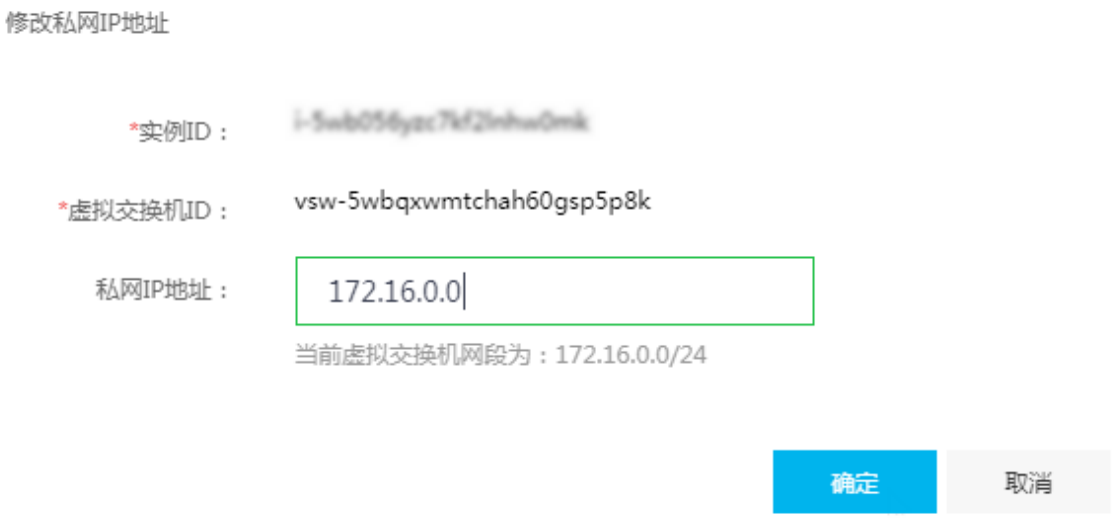
11.4.9 修改私网IP

每个实例会分配一块私网网卡，并绑定一个私网IP。私网 IP 段根据虚拟交换机IP 段。只有处于**停止**状态的实例才可以修改私网IP。

操作步骤

1. 登录*ECS*云控制台，找到待修改私网IP的实例。
2. 完成停止实例。
3. 在相应实例的**操作**列中，选择**管理** > **修改私网IP地址**。
4. 在弹出的图 *35: 修改私网IP*对话框中，输入新的**私网IP地址**，单击**确定**完成更改。

图 35: 修改私网IP



11.5 管理磁盘

ECS 的磁盘种类分为普通云盘、SSD 云盘和高效云盘。

挂载点是阿里云服务器上磁盘在磁盘控制器总线上的位置。

所选配的挂载点，在 Linux 环境下与磁盘设备号对应；在 Windows 环境下与磁盘管理器中的磁盘顺序一致。

11.5.1 云盘简介

云服务器本身提供了云盘用于数据存储。您可以在操作系统中直接识别云盘，进行操作和读写。

ECS 云盘对于云服务器的 ECS 实例而言，您可以认为是一块物理的磁盘。您必须对这块磁盘进行挂载、格式化等基本操作后才能使用。

云服务器共有三种磁盘种类：普通云盘、SSD 云盘和高效云盘。

云盘有以下基本特性：

- 数据安全性高。
- 随机和顺序读写的 IOPS 高。
- 能够单独为实例增加一块或多块云磁盘，至多 5 块磁盘（包括系统盘和数据盘）。
- 当出现宕机迁移时，数据保持在宕机前最后一刻的数据状态。

快照和镜像存放在同一地域中的 OSS 上。为保证服务的灵活性和资源使用效率，当通过快照或者镜像创建磁盘时，分布式存储不会把所有的数据立刻复制到新的磁盘上，而是通过按需的方式进行优化按块加载，即只有当使用到一块数据时，才把用到的整块数据复制过来。

通常一个数据块只有数 MB 大小，下次读写这一块数据时，就直接从磁盘上读取了。

同时为了保证用户整体 I/O 的最优体验，分布式存储会在 I/O 较为空闲时，通过后台复制的方式把快照和镜像的数据逐步逐块地复制到磁盘上。

所以，当首次对云盘进行读写时，云盘的 I/O 性能有可能出现明显的下降，一旦进行过一次访问，I/O 性能会恢复正常。在进行高负荷的运行前，建议您进行全盘访问，比如读操作。

11.5.2 查看磁盘

您可以通过ECS云控制台查看所拥有的磁盘以及相应磁盘的具体信息。

操作步骤

1. 登录*ECS*云控制台，进入**磁盘**页面。
2. 您可以输入 **部门**、**区域**或**磁盘名称**，单击**查询**，找到指定的磁盘。
3. 在相应磁盘的**操作**列中，选择**查看详情**，进入**磁盘详情**页面，查看磁盘的具体信息。

11.6 管理镜像

ECS 镜像是一个包含软件配置（比如操作系统、应用服务器和应用程序）的云服务器的模板。创建实例需要指定一个 ECS 的镜像，该镜像提供的操作系统和软件都会安装在您创建的实例上。您还可以根据创建的实例来创建自定义镜像，您可以根据自定义镜像来创建更多的实例。

11.6.1 查看镜像

您可以通过ECS云控制台直接查看镜像的运行情况。

操作步骤

1. 登录*ECS*云控制台，进入**镜像**页面。
2. 您可以选择**部门**、**区域**或输入**镜像名称**，单击**查询**，查看具体的镜像信息。

11.7 管理快照

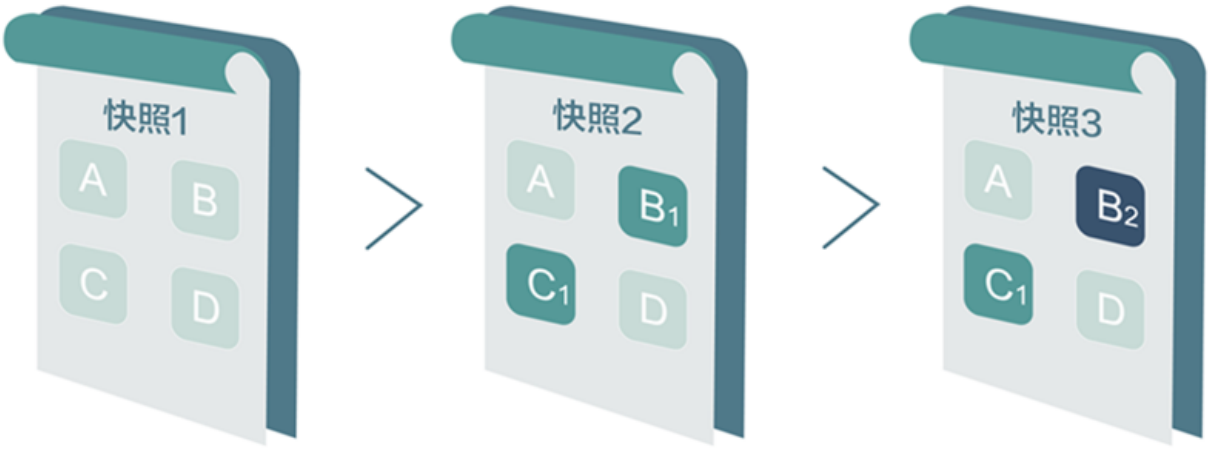
在使用磁盘的过程中，会遇到下面的场景：

- 当您在磁盘上进行数据的写入和存储时，可能希望使用一块磁盘上的数据作为其他磁盘的基础数据。
- 云磁盘虽然提供了安全的存储方式确保您所存储的任何内容都不丢失，但是如果存储在磁盘上的数据本身就是错误的数据，比如因为应用的错误，或黑客利用您的应用的漏洞进行恶意读写，那么就需要其他的机制来保证在您的数据出问题时，能够恢复到您期望的数据状态。

阿里云提供了快照机制，通过创建快照保留某一个时间点上一个磁盘的数据拷贝，有计划地对磁盘创建快照，是一个保证您业务可持续运行的非常好的习惯。

快照使用增量的方式，两个快照之间只有数据变化的部分才会被拷贝，如下图所示：

图 36: 快照工作原理示意图



快照 1 、快照 2 和快照 3 分别是磁盘的第一个、第二个和第三个快照。文件系统对磁盘的数据进行分块检查，当创建快照时，只有数据变化了的块，才会被复制到快照中。

快照 1 由于是磁盘的第一个快照，会把这个磁盘上的所有数据都复制一份；而快照 2 只是复制了有变化的数据块 B1 和 C1，数据块 A 和 D 引用了快照 1 中的 A 和 D；同理，快照 3 则复制了有变化数据块 B2，数据块 A 和 D 继续引用快照 1 中的，而数据块 C1 则引用快照 2 中的。

当磁盘需要恢复到快照 3 的状态，快照回滚会把数据块 A、B2、C1 和 D 复制到磁盘上，从而恢复成快照 3 的状态。

如果快照 2 被删除，快照中的数据块 B1 将被删除，但是数据块 C1 不会被删除。这样，在恢复到快照 3 时，仍可以恢复数据块 C1 的状态。

快照是放在对象存储（OSS）上的，但是您不能在 OSS 上看到这些快照文件，也不能计算到 OSS 的用户的 Bucket 的占用空间，只能通过云服务器 ECS 的控制台或者 API 进行快照操作。

11.7.1 查看快照

您可以通过ECS云控制台查看您所拥有的所有快照。

操作步骤

1. 登录*ECS*云控制台，进入**快照**页面。
2. 您可以选择**部门**、**区域**或输入**快照名称**，单击**查询**，查看快照信息。

11.8 管理自动快照策略

11.8.1 查看自动快照策略

您可以通过ECS云控制台查看您所拥有的所有自动快照策略。

操作步骤

1. 登录*ECS*云控制台，进入**自动快照策略**页面。
2. 您可以选择**区域**或输入**自动快照策略ID**，单击**查询**，查看具体的自动快照策略。

11.8.2 为磁盘批量设置自动快照策略

操作步骤

1. 在**云控制台**首页上，单击**云资源中心** > **云产品管理** > **云服务器** > **自动快照策略**，单击操作**管理** > **批量操作**进入批量管理自动快照策略页面。
2. 选择你想批量应用自动快照策略的磁盘：

- 在**可选磁盘**列表中单击任一磁盘，再单击→即可选中该磁盘；
- 在**已选磁盘**列表中单击任一磁盘，再单击←即可取消选择该磁盘；
- 在**可选磁盘**右侧单击**选择全部**，再单击→即可选中所有磁盘；
- 在**已选磁盘**右侧单击**选择全部**，再单击←即可取消选所有磁盘；

3. 单击**添加**即可。

11.9 管理安全组

安全组主要涉及以下操作：创建安全组、删除安全组、加入安全组、移出安全组、添加安全组规则（IP 授权、端口授权、协议授权、安全组授权）等。

11.9.1 安全组限制

- 单个安全组内的实例个数不能超过 1000。
- 每个实例最多可以加入 5 个安全组。
- 每个安全组规则条目最多 100 条。
- 每个用户的安全组最多 100 个。
- 对安全组的调整操作，对用户的服务连续性没有影响。
- 安全组是有状态的，如果数据包在 outbound 方向是被允许的，那么对应的此连接在 inbound 方向也是允许的。

11.9.2 查看安全组

您可以通过ECS云控制台查看所有安全组的相关信息。

操作步骤

1. 登录*ECS*云控制台，进入**安全组**页面。
2. 选择**部门**、**区域**或输入**安全组名称**，单击**查询**，找到指定的安全组。
3. 在对应安全组的**操作**列中，选择**查看详情**，查看该安全组内实例和安全组规则的详细信息。



说明: 您也可以直接单击安全组ID/名称来，查看安全组相关的信息。

11.10 安装 FTP 软件

使用远程连接工具或阿里云的远程连接功能 VNC 连接 ECS 实例。

11.10.1 Redhat/CentOS 安装 vsftp 软件

操作步骤

1. 安装 vsftp。使用 yum 命令安装 vsftp：yum install vsftpd -y。
2. 添加 ftp 账户和目录。
   1. 检查 *nologin* 的位置。通常在 */usr/sbin/nologin* 或者 */sbin/nologin* 下。
   2. 创建账户。下面的命令指定 */alidata/www/wwwroot* 为用户 pwftp 的家目录，您可以自己定义账户名和目录：useradd -d /alidata/www/wwwroot -s /sbin/nologin pwftp。
   3. 修改账户密码。

      ```
      passwd pwftp
      ```
   4. 修改指定目录的权限。

      ```
      chown -R pwftp.pwftp /alidata/www/wwwroot
      ```
3. 配置 vsftp。
   1. 编辑 vsftp 配置文件：vi /etc/vsftpd/vsftpd.conf。
   2. 修改配置文件中**anonymous_enable=YES** 为 **anonymous_enable=NO**。
   3. 取消如下配置前的注释符号 #：

      ```
      local_enable=YES
            write_enable=YES
            chroot_local_user=YES
      ```
   4. 保存修改，按 ESC 键，输入命令： wq。
4. 修改shell 配置：vi 编辑 */etc/shells*。如果该文件里没有 */usr/sbin/nologin* 或者 */sbin/nologin*（具体看当前系统配置），则追加进去。
5. 启动 vsftp 服务并测试登录。
   1. 启动vsftp服务：service vsftpd start。
   2. 使用账户pwftp测试是否可以登录 FTP。如本实例中的目录是 */alidata/www/wwwroot*。

11.10.2 Ubuntu/Debian 安装 vsftp 软件

操作步骤

1. 使用 apt-get 命令安装vsftp：apt-get install vsftpd -y。
2. 添加 ftp 账户和目录：
   1. 先检查一下 nologin 的位置，通常在 */usr/sbin/nologin* 或者 */sbin/nologin* 下。
   2. 使用下面的命令创建账户，该命令指定了 */alidata/www/wwwroot* 为用户 pwftp 的目录，您可以自己定义账户名和目录：useradd -d /alidata/www/wwwroot -s /sbin/nologin pwftp。
   3. 修改该账户密码：passwd pwftp。
   4. 修改指定目录的权限：chown -R pwftp.pwftp /alidata/www/wwwroot。
3. 配置 vsftp：
   1. 编辑 vsftp 配置文件，命令如下：vi /etc/vsftpd.conf。
   2. 将配置文件中 **anonymous_enable=YES** 改为 **anonymous_enable=NO**。
   3. 取消如下配置前的注释符号：

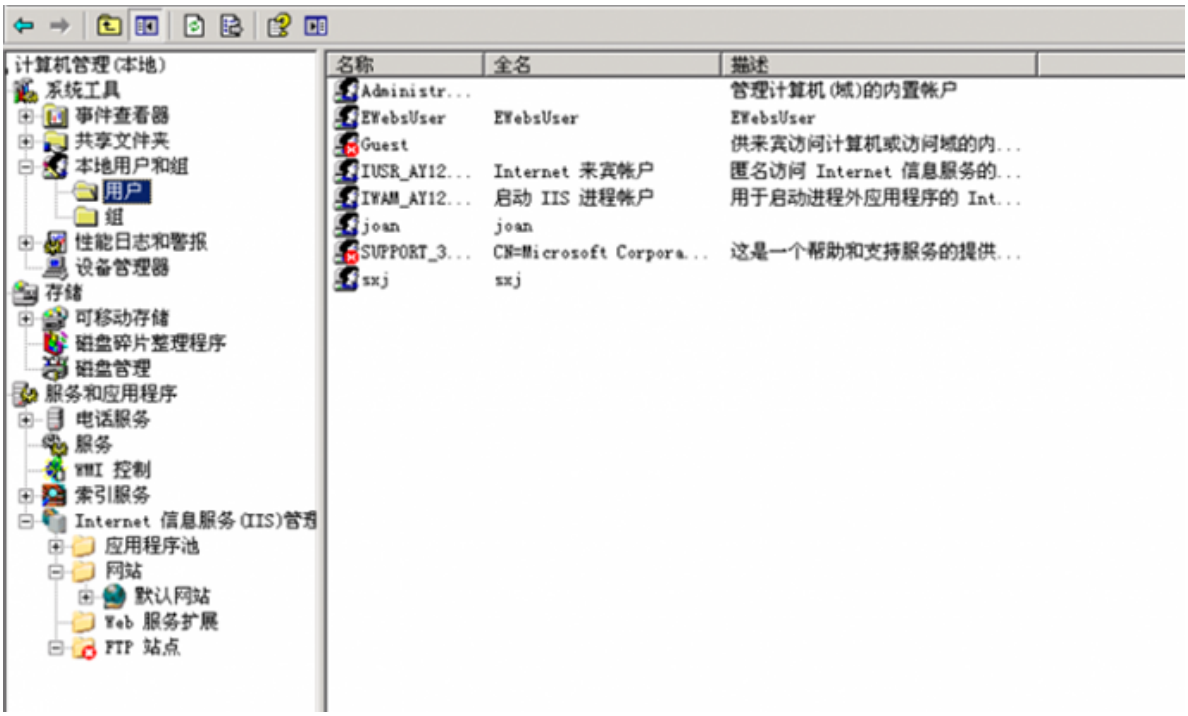
      ```
      local_enable=YES
            write_enable=YES
            chroot_local_user=YES
            chroot_list_enable=YES
            chroot_list_file=/etc/vsftpd.chroot_list
      ```
   4. 保存退出。
   5. 编辑*/etc/vsftpd.chroot_list*文件，将 ftp 的账户名添加进去，保存退出。
4. 修改shell 配置：vi 编辑 */etc/shells*。如果该文件里没有 */usr/sbin/nologin* 或者 */sbin/nologin*（具体看当前系统配置），则追加进去。
5. 重启 vsftp 服务并测试登录：
   1. 使用命令启动 vsftp 服务：service vsftpd restart。
   2. 用账户 pwftp 测试下是否可以登录 ftp。目录是 */alidata/www/wwwroot*。

11.10.3 Windows 2003 通过 IIS 配置 FTP

操作步骤

1. 远程连接 ECS ，右键单击 **我的电脑**，选择 **管理**。
2. 在导航窗口中，双击 **用户** 打开用户列表，并在用户列表空白区域右键单击，并在下拉列表中单击 **新用户**。

   图 37: 打开用户列表



3. 如图输入您的 FTP 用户名和密码后，单击**创建**。

图 38: 创建新用户



4. 在导航窗口中，展开 **Internet 信息服务 (IIS) 管理**，删除默认的 FTP 站点后，右键单击 **FTP 站点**，在弹出菜单中单击**新建 > FTP 站点(F)...**，开始根据**FTP 站点创建向导**指示创建新的 FTP 站点。

图 39: 新建FTP站点

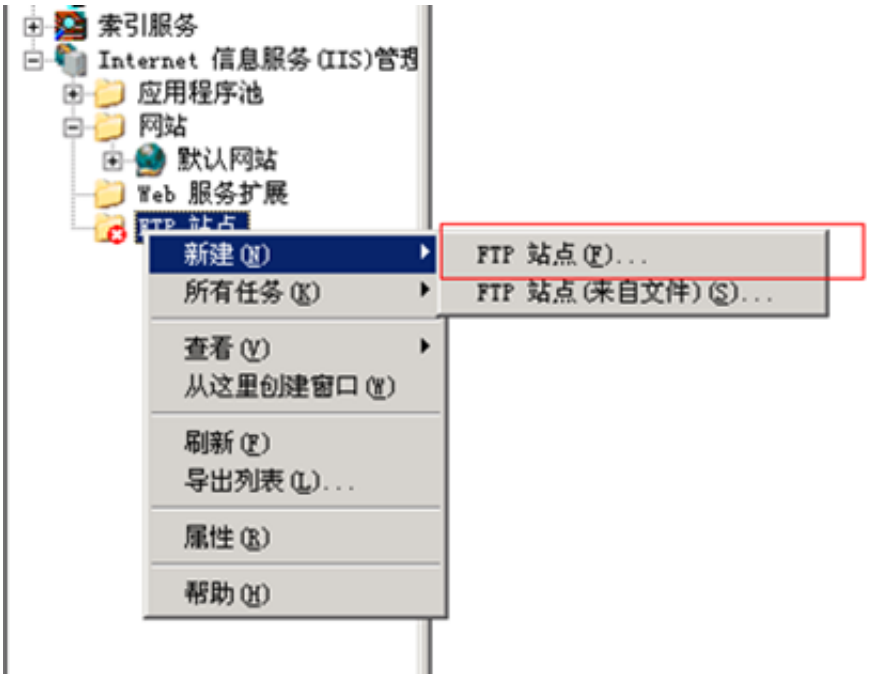


图 40: 创建向导



图 41: FTP站点描述



图 42: IP 地址和端口设置

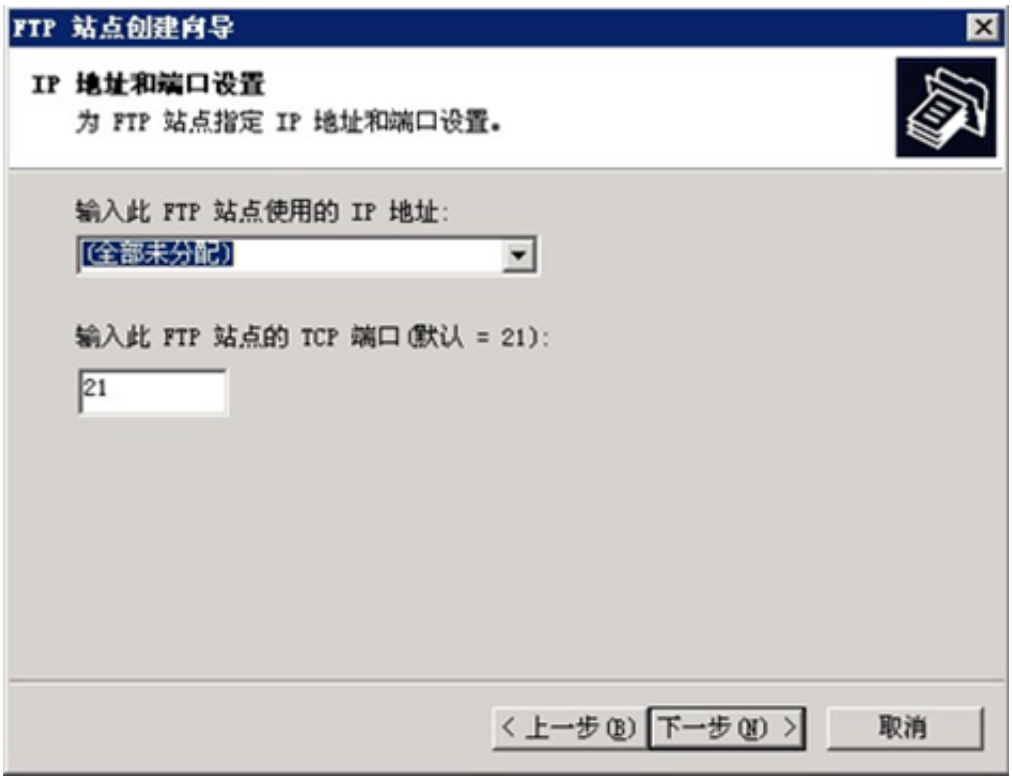
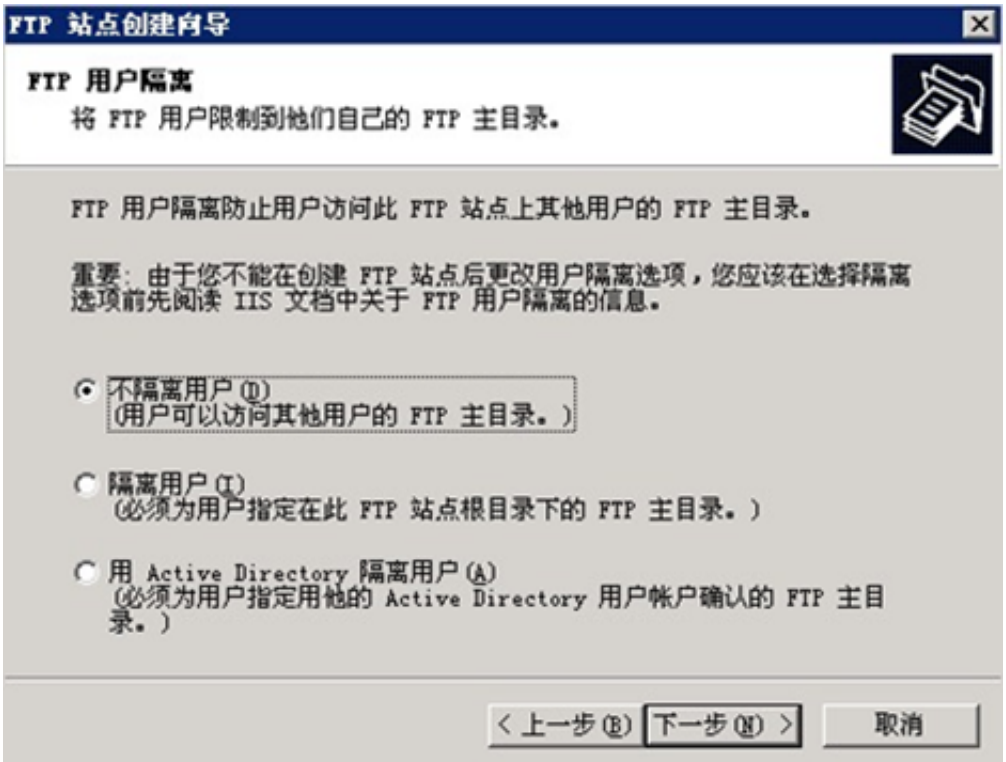

FTP 站点创建向导
IP 地址和端口设置
为 FTP 站点指定 IP 地址和端口设置。
输入此 FTP 站点使用的 IP 地址:
(全部未分配)
输入此 FTP 站点的 TCP 端口(默认 = 21):
21
< 上一步(B)
下一步(N) >
取消

图 43: FTP用户隔离


FTP 站点创建向导
FTP 用户隔离
将 FTP 用户限制到他们自己的 FTP 主目录。
FTP 用户隔离防止用户访问此 FTP 站点上其他用户的 FTP 主目录。
重要: 由于您不能在创建 FTP 站点后更改用户隔离选项,您应该在选择隔离选项前先阅读 IIS 文档中关于 FTP 用户隔离的信息。
不隔离用户(D)
(用户可以访问其他用户的 FTP 主目录。)
隔离用户(I)
(必须为用户指定在此 FTP 站点根目录下的 FTP 主目录。)
用 Active Directory 隔离用户(A)
(必须为用户指定用他的 Active Directory 用户帐户确认的 FTP 主目录。)
< 上一步(B)
下一步(N) >
取消

5. 输入主目录的路径，比如，示例图中输入的路径是*D:\websoft\www*。

图 44: 主目录路径设置

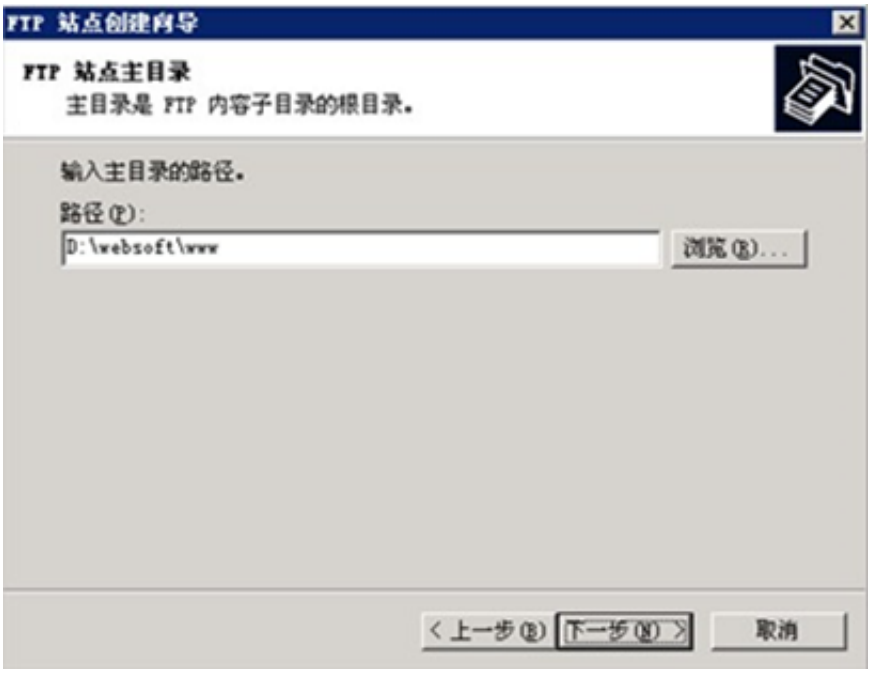


图 45: 访问权限设置



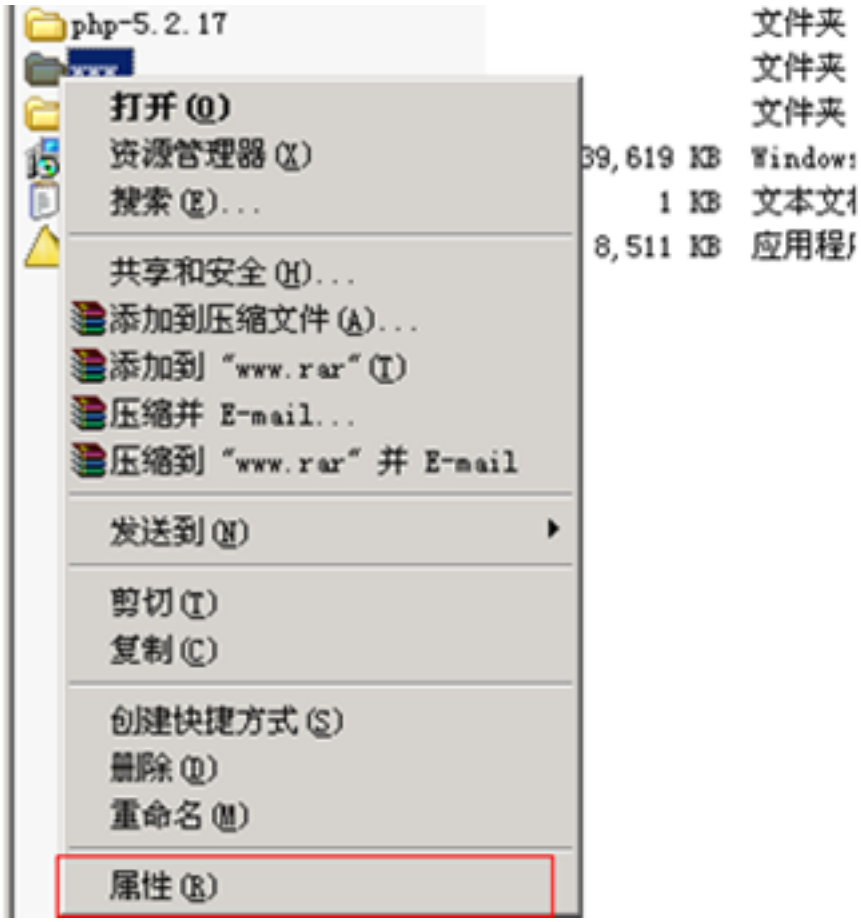
图 46:



6. 设置新建 FTP 站点 www 的权限。

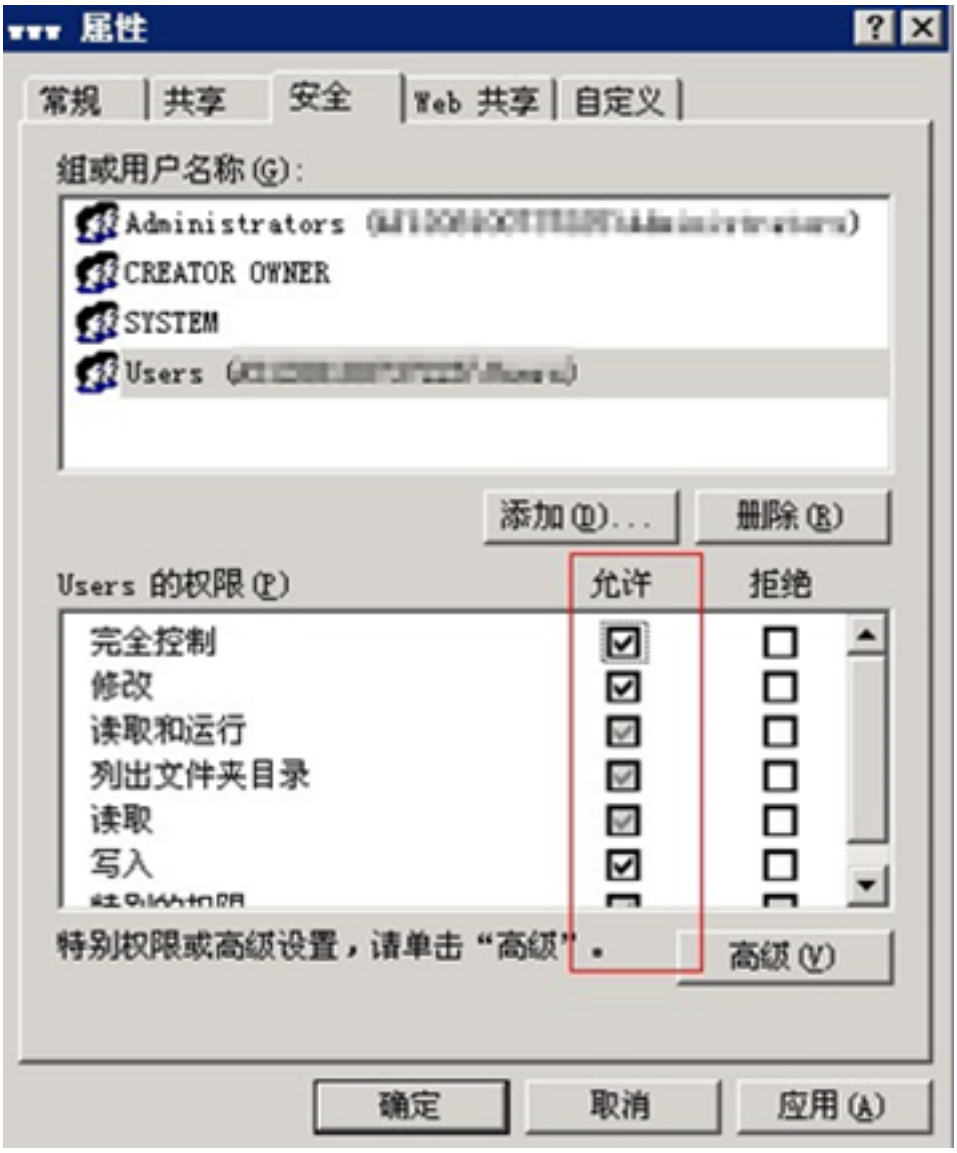
   1. 右键单击文件夹www，在弹出菜单中选择**属性**。

      图 47: 选择属性



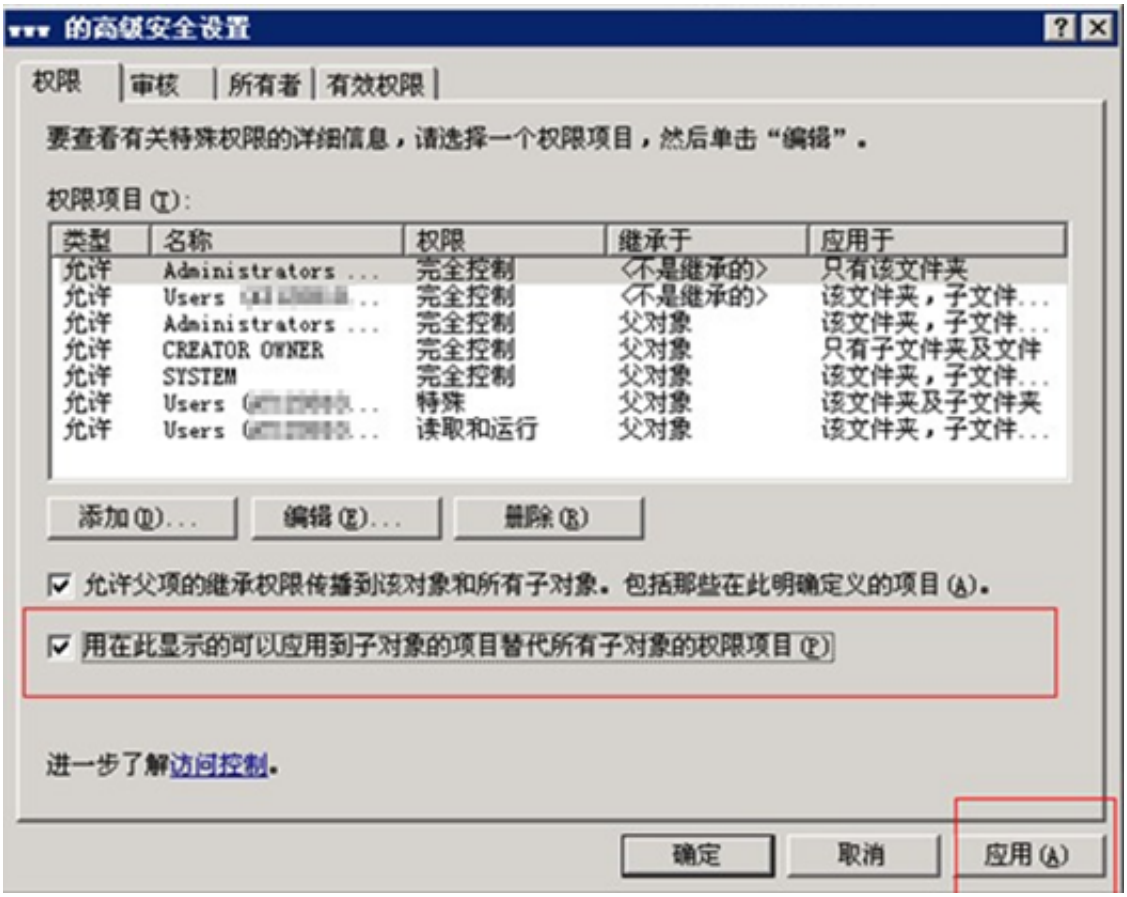
2. 单击**安全**选项卡，在**组或用户名称**中选中**Users**，并设置权限。

图 48: 设置权限



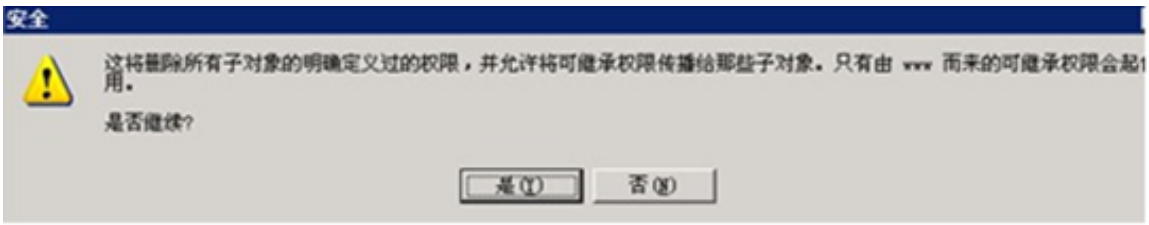
3. 单击**高级**（V），如图所示进行高级安全设置后，单击 **应用**。

图 49: 安全设置



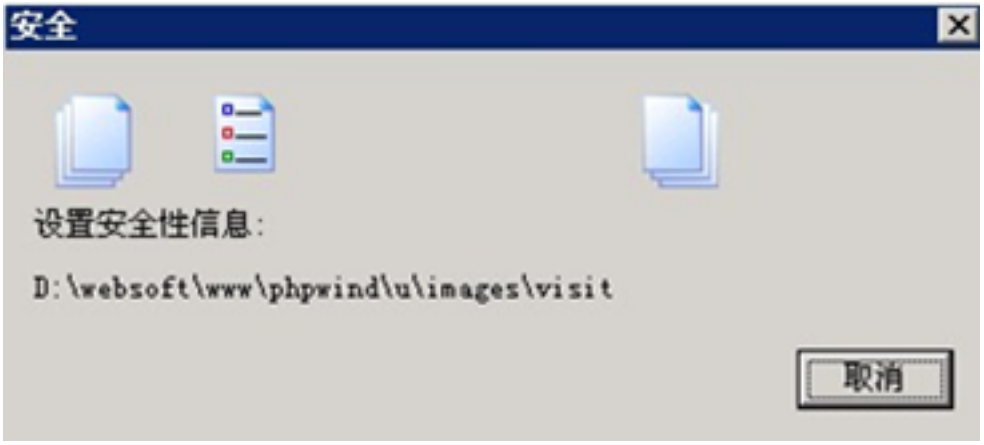
4. 在弹出对话框中，单击**是**。

图 50: 确定



5. 等待系统操作完成。

图 51: 等待完成



6. 在**安全**选项卡中，添加pwftp账号的权限。按如下各图所示完成操作。

图 52: 添加权限

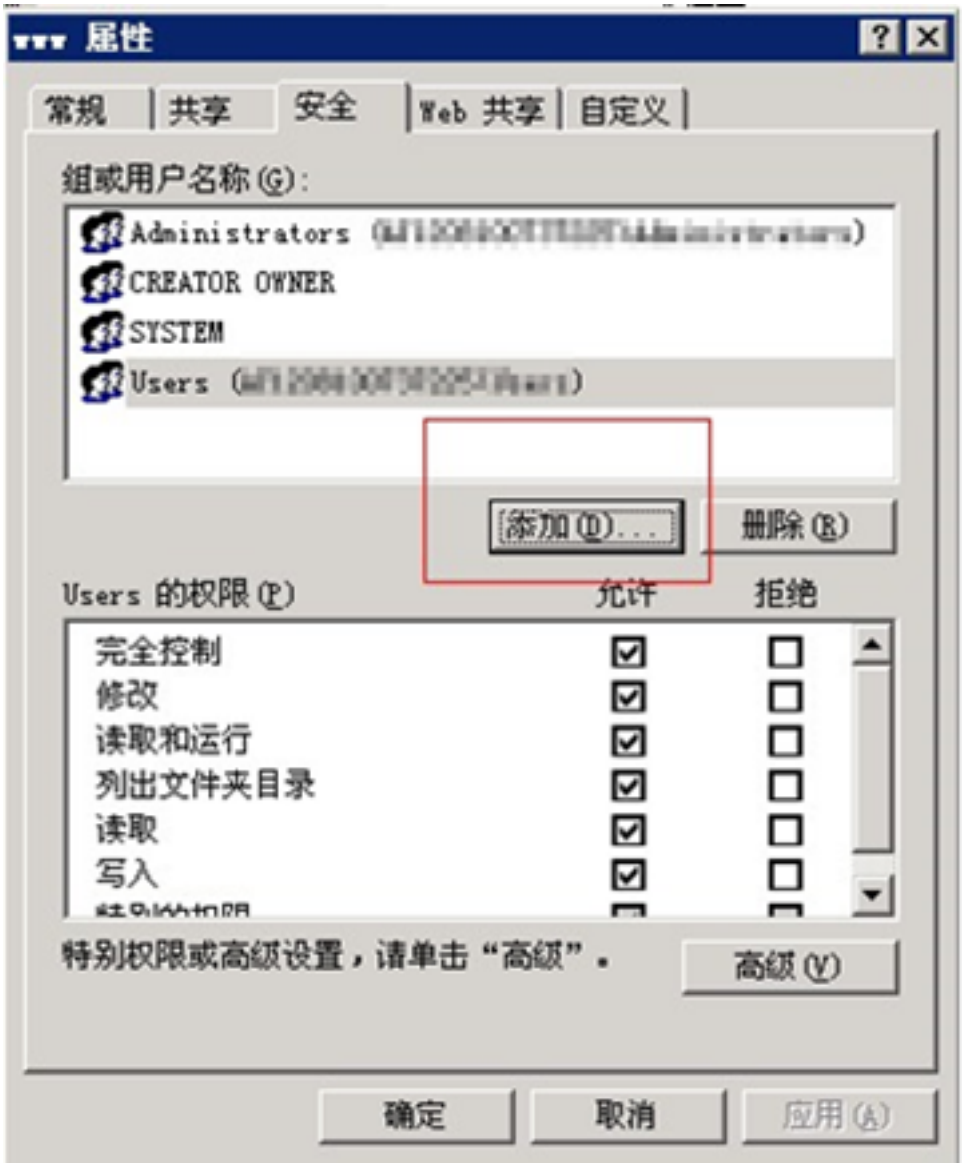
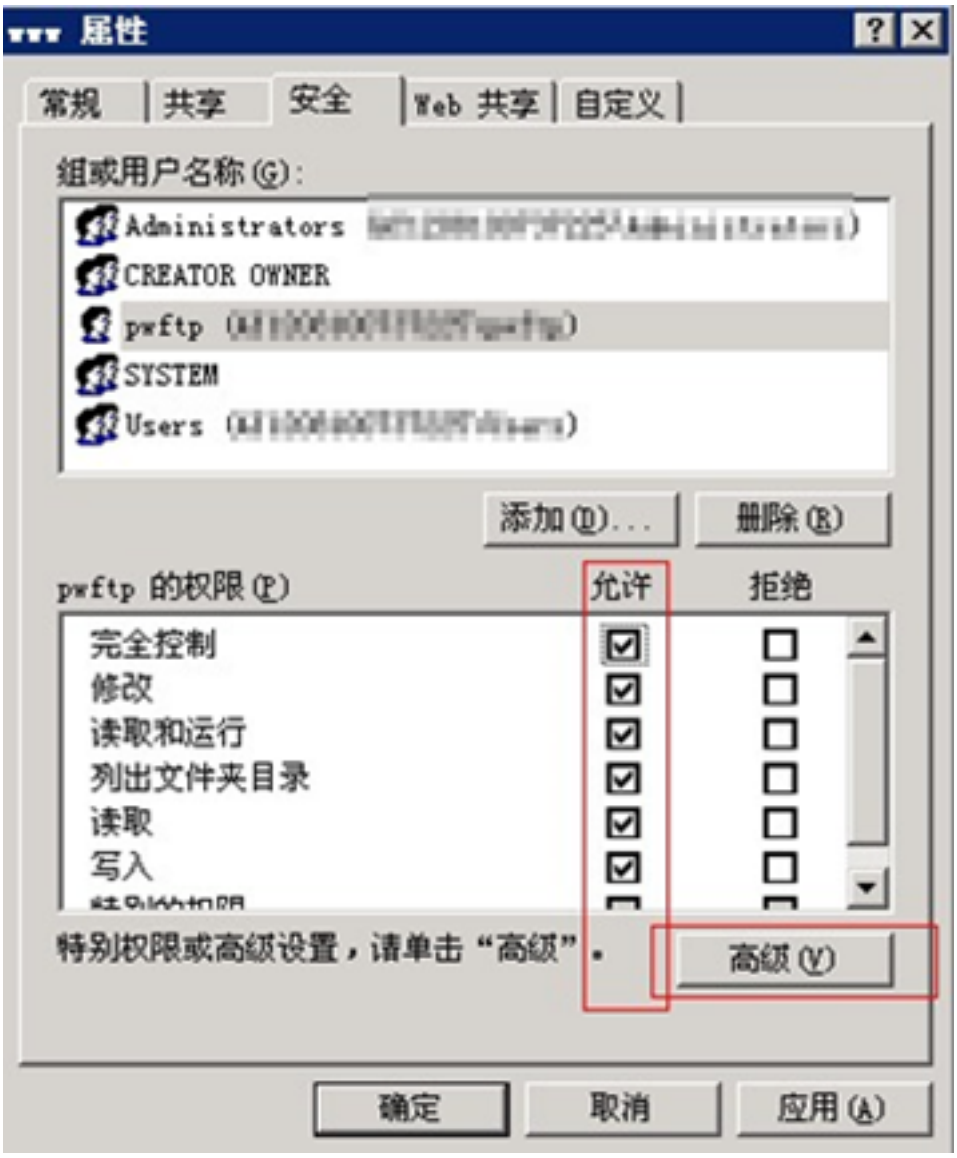

属性
常规
共享
安全
Web 共享
自定义
组或用户名称(G):
Administrators
CREATOR OWNER
SYSTEM
Users
添加(D)...
删除(R)
Users 的权限(P)
允许
拒绝
完全控制
修改
读取和运行
列出文件夹目录
读取
写入
特别权限或高级设置，请单击“高级”。
高级(V)
确定
取消
应用(A)

图 53: 添加用户或组



7. 单击 **确定**后，界面如图 *54:* 权限设置完成所示。单击 **高级**，设置账号 pwftp 的高级安全设置，设置方式与 Users 的设置相同。

图 54: 权限设置完成



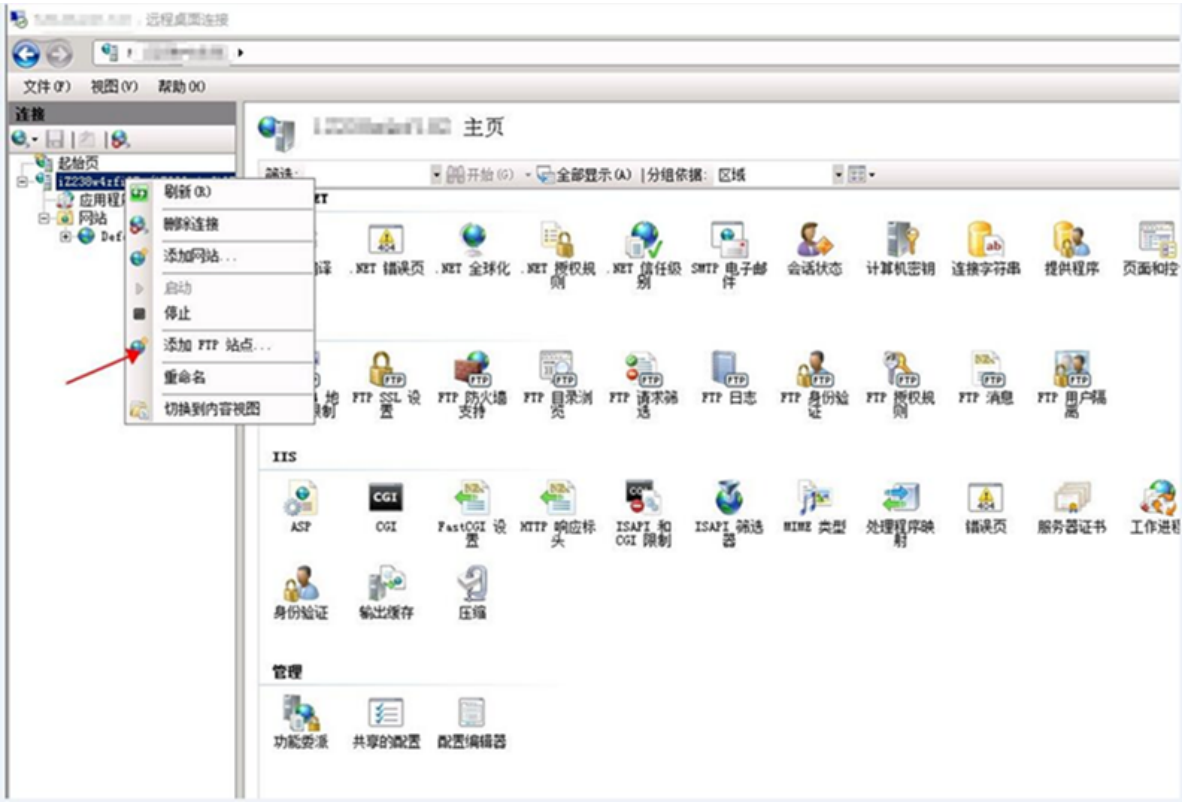
8. 完成设置后，单击 **应用**，再单击 **确定**。

11.10.4 Windows 2008 安装配置 FTP

操作步骤

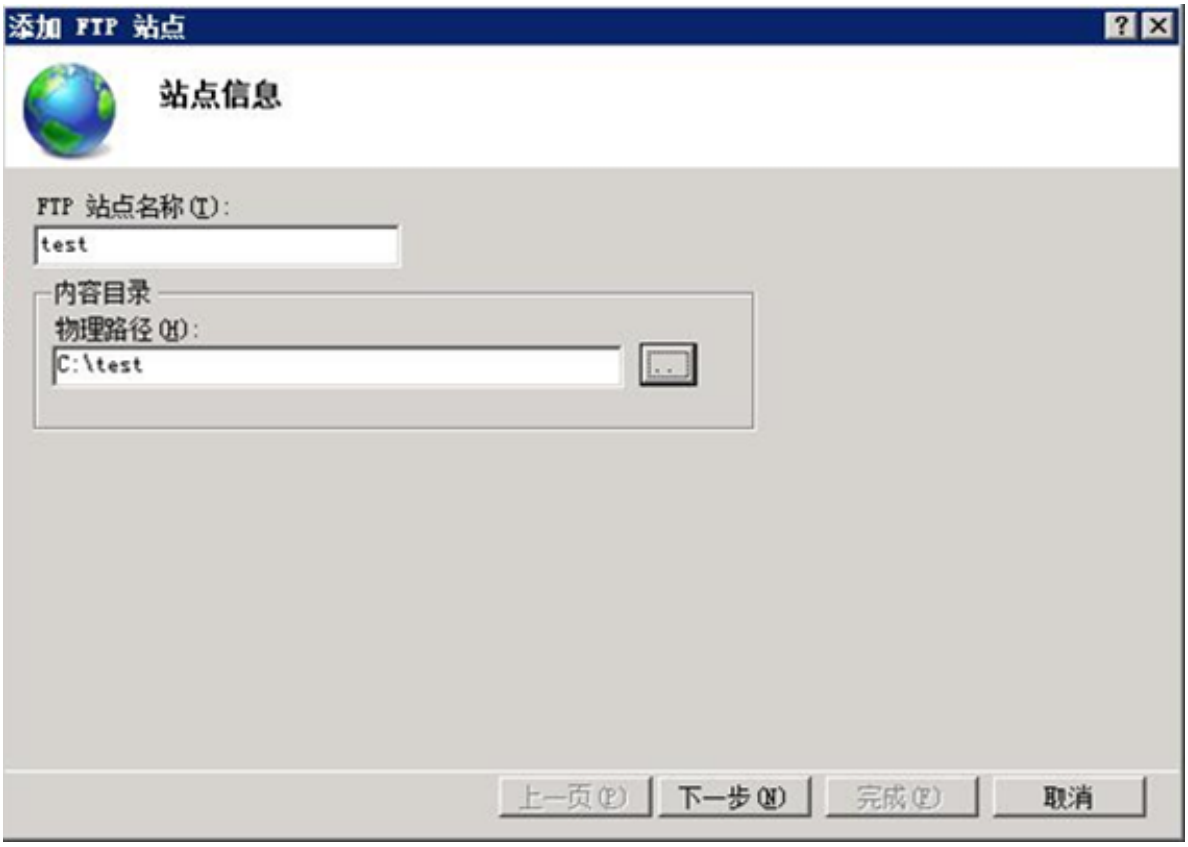
1. 远程登录服务器后，选择**开始** > **管理工具** > **Internet 信息服务**（IIS）**管理器**，再右键单击服务器名称选择 **添加 FTP 站点**。

图 55: 添加FTP站点



2. 输入FTP 站点名称和指定的路径，单击 **下一步** 按钮。

图 56: 添加FTP站点



3. 如图 *57: IP* 地址绑定所示，**IP 地址**选择**全部未分配**，**SSL** 选择**无。**

图 57: IP 地址绑定



4. **身份验证** 选择**基本**，**授权** 选择**所有用户**，**权限**设置为**读取**和**写入**。

图 58: 身份验证和授权信息



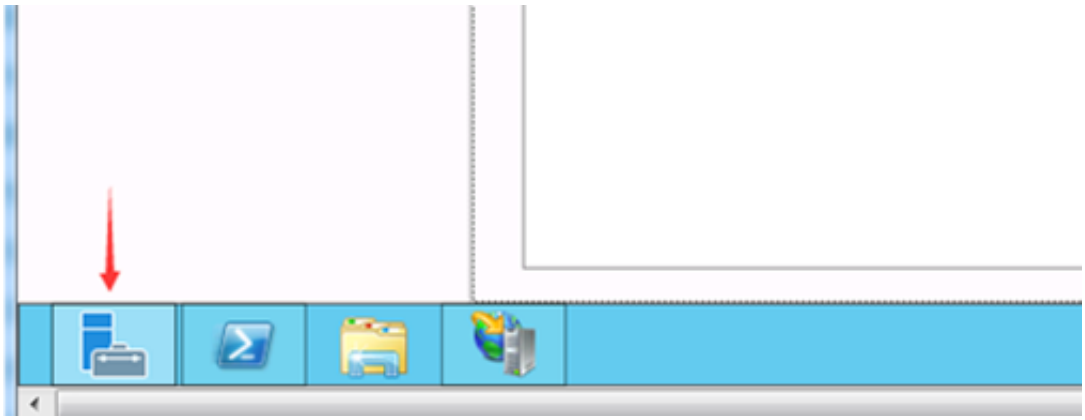
5. 单击**完成**，完成 FTP 配置。之后，您可以使用Administrator账号和密码通过 FTP 连接来上传或下载文件。

11.10.5 Windows 2012 安装配置 IIS+FTP

操作步骤

1. 在服务器操作界面的左下角，单击**服务器管理**图标启动**服务器管理器**。

图 59: 启动服务器管理器



2. 如图所示，打开 IIS 管理器。

图 60: 打开 IIS 管理器



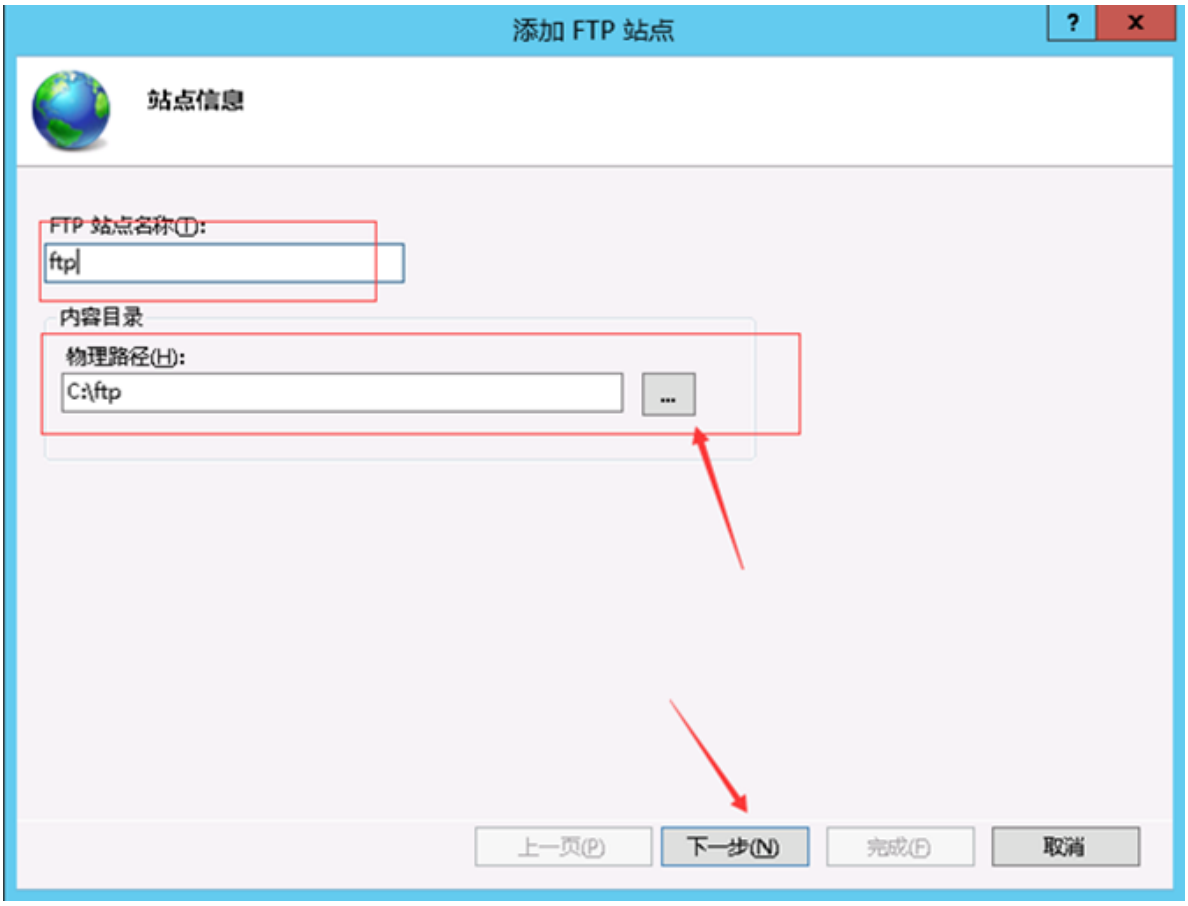
3. 如下图所示，在 IIS 管理器中添加 FTP 站点。

图 61: 添加FTP站点



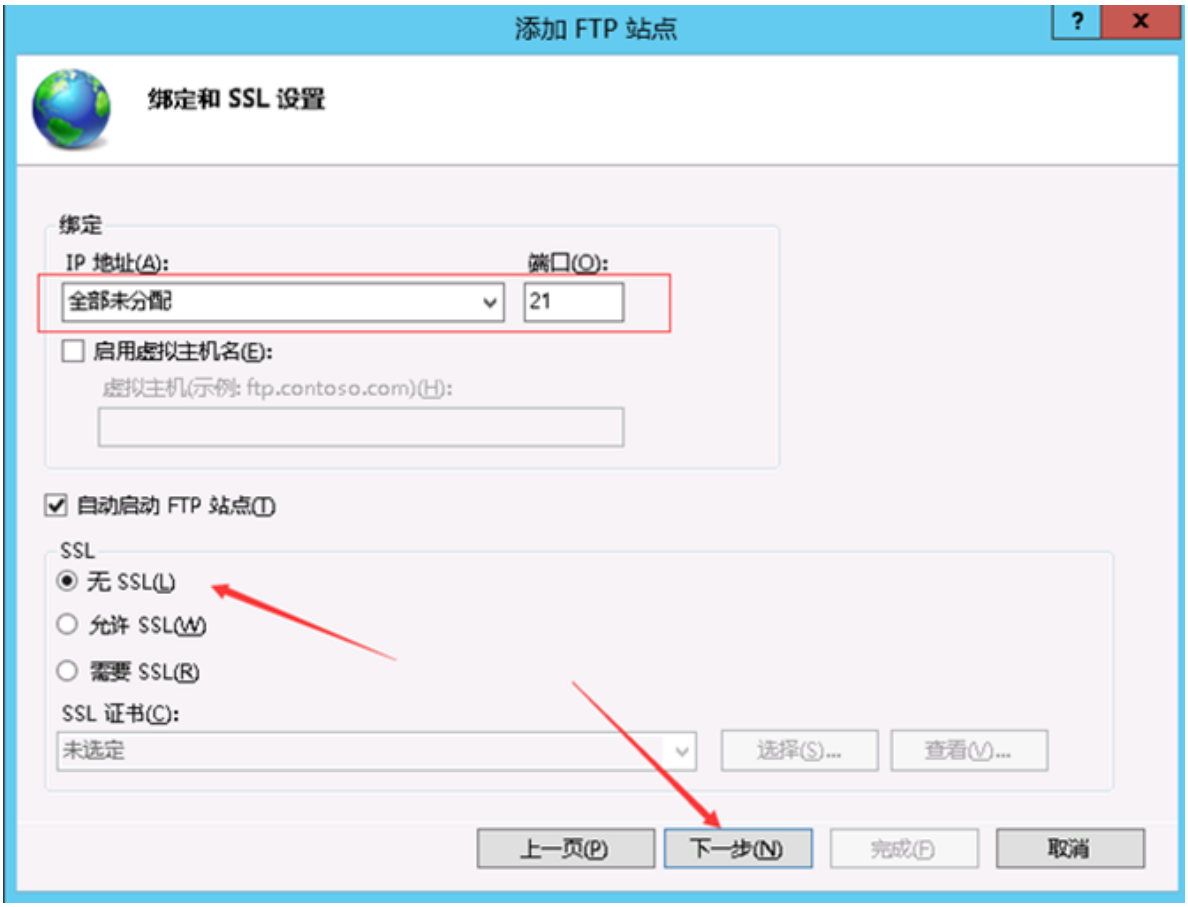
4. 填写 FTP 站点名称，指定 FTP 的路径。

图 62: 填写站点信息



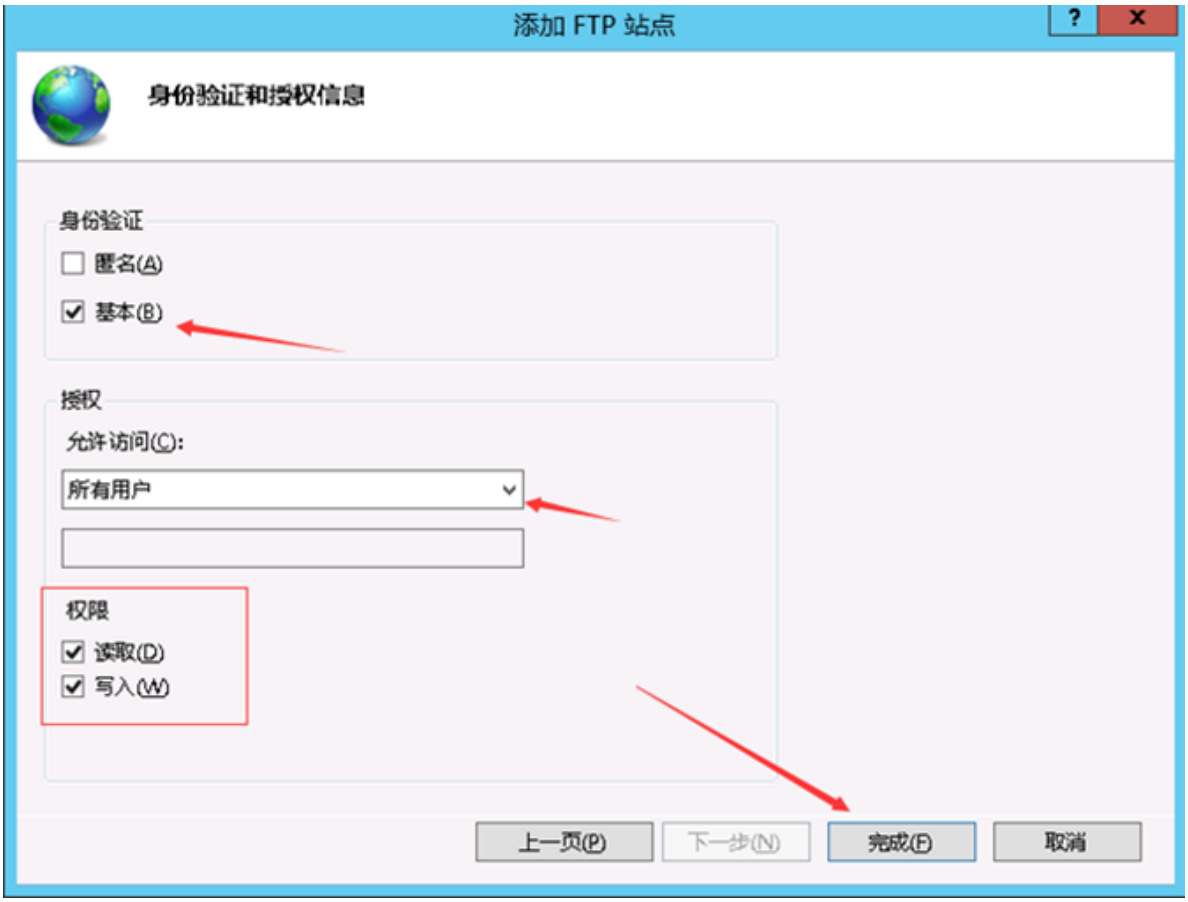
5. 如下图所示，**IP 地址**选择**全部未分配**，**SSL** 选择**无**。

图 63: 绑定和SSL设置



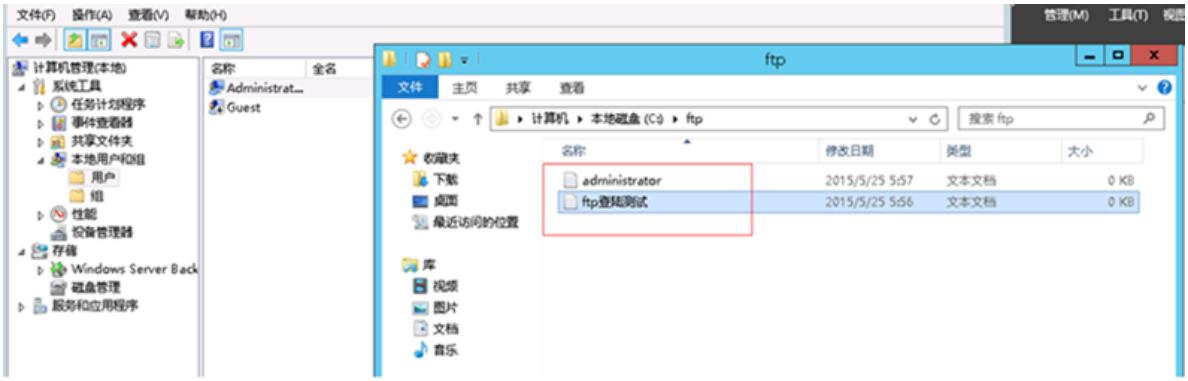
6. **身份验证** 选择**基本**，**授权** 选择**所有用户**，**权限**设置为**读取**和**写入**。

图 64: 身份验证和授权信息



7. FTP 配置完成后，用系统默认的administrator账号和密码，进行登录测试，可以正常上传或下载文件。

图 65: 登录测试



12 容器服务

12.1 产品概述

容器服务（Container Service）是一种高性能可伸缩的容器管理服务，支持在一组阿里云云服务器上通过Docker 容器来运行或编排应用。

容器服务免去了您对容器管理集群的搭建，整合了负载均衡、专有网络等云产品，让您可以通过控制台或简单的API（兼容Docker API）进行容器生命周期管理。

12.2 Swarm快速开始

12.2.1 创建Swarm集群

您可以通过容器服务云控制台，创建一个swarm集群。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**集群**，单击右上角的**创建集群**。

3. 设置集群的基本信息。

   - **集群名称**： 要创建的集群的名称。可以包含1~64个字符，包括数字，中文字符，英文字符和连字符（-）。

     说明: 集群名称在同一个用户下必须唯一。

   - **地域**： 所创建集群将要部署到的地域。
   - **可用区**：集群的可用区。

4. 设置**集群模式**为swarm。

   集群模式有两种：swarm和swarm mode。

5. 选择集群的**网络类型**

   - 选择**经典网络**。
   - 选择**专有网络**，您需要选择一个可用的VPC和VSwitch，并设置容器的起始网段。

   图 66: 集群网络



6. 是否新增节点。

   您只能选择**添加已有节点**，您只能先创建一个零节点集群，并在集群创建成功后再添加已有云服务器。有关如何添加已有云服务器，参见添加已有云服务器。



7. 创建一个负载均衡实例。

   勾选后，系统会在创建集群时创建一个负载均衡实例。您可以通过这个负载均衡实例访问集群内的容器应用。您也可以不随集群常见负载均衡，等集群创建完成后再绑定负载均衡，参见为集群绑定负载均衡。

8. 单击**创建集群**。

   集群创建成功后，如果需要单独配置云服务器或负载均衡，可以去相应的控制台进行相关操作。

后续操作

您可以查看集群创建日志。在**集群列表**页面，选择所创建的集群并单击**查看日志**。

12.2.2 使用镜像创建wordpress应用

您可以通过容器服务云控制台，在swarm集群下使用镜像创建wordpress应用。

操作步骤

1. 登录容器服务云控制台。

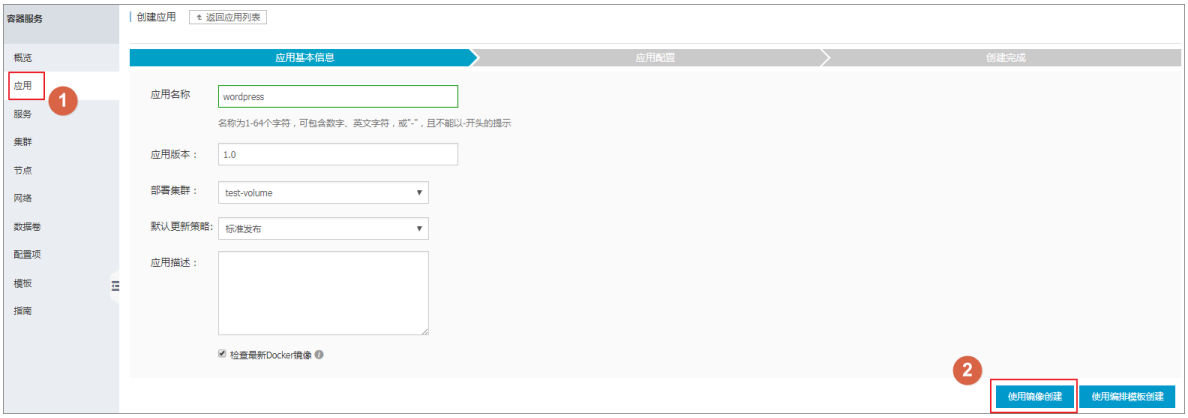
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**，单击右上角的**创建应用**。

3. 设置应用的基本信息并单击**使用镜像创建**。

   - **应用名称**：本示例的应用名称是 wordpress。名称可以包含1~64个字符，包括数字，中文字符，英文字母和连字符（-）。

- **应用版本**：所创建应用的版本。默认为1.0。
- **部署集群**：所创建应用将要部署到的集群。
- **默认更新策略**：应用更新的方式，您可以选择标准发布或蓝绿发布，参见概述。
- **应用描述**：应用信息。可以不填写。该信息不能超过1,024个字符。该信息将显示在 **应用列表** 页面。
- **检查最新Docker镜像**：选中该选项后，表示当镜像Tag不变的情况下，也会去仓库拉取最新的镜像。为了提高效率，容器服务会对镜像进行缓存。部署时，如果发现镜像Tag与本地缓存的一致，则会直接复用而不重新拉取。所以，如果您基于上层业务便利性等因素考虑，在做代码和镜像变更时没有同步修改Tag ，就会导致部署时还是使用本地缓存内旧版本镜像。而勾选该选项后，会忽略缓存，每次部署时重新拉取镜像，确保使用的始终是最新的镜像和代码。



4. 根据您的需要设置以下参数。

   a) 设置**镜像名称**和**镜像版本**。

   您可以直接填写目标Hub的镜像名称，填写的格式为域名*/namespace/imagename:tag*。容器服务会默认使用镜像的最新版本。如果您需要使用镜像的其它版本，单击 **选择镜像版本**，单击所需版本并单击 **确定**。

   b) 设置**容器数量**。

   c) 设置应用的**网络模式**。

   容器服务目前支持 **默认**和 **host** 网络模式。如果您不进行设置，容器服务会默认使用 **默认** 网络模式。

   d) 设置**Restart**。

   即容器是否是否异常重启。

   e) 设置容器的启动命令（**Command**和**Entrypoint**）。

如果设置了会覆盖镜像的配置。

f) 设置容器的资源限制（**CPU 限制** 和 **内存限制**）。

有关容器资源限制的详细信息，参见 限制容器资源。

g) 设置**端口映射**、**简单路由配置**、**负载均衡路由配置**。

本示例中，暴露容器**80**端口，在简单路由配置中配置虚拟域名**wordpress**。

图 67: 配置端口和简单路由



h) 设置容器的**数据卷**。

volumes_from：填写另外一个服务或者容器的名称，将服务或容器上的所有数据卷挂载到所创建的应用上。默认会赋予新创建应用对数据卷的读写权限。

i) 设置**环境变量**。

j) 容器的**labels**。

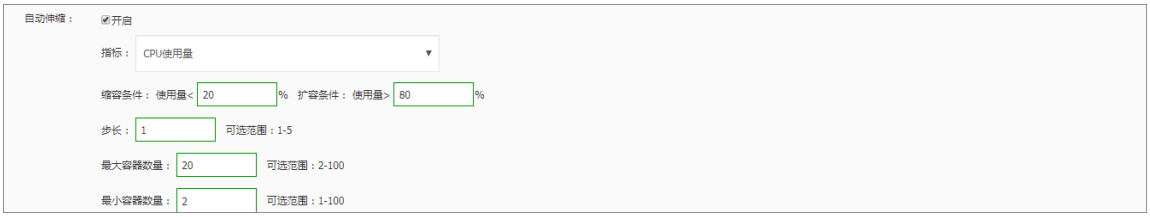
有关容器服务支持的标签，参见 标签概览。

k) 设置是否允许容器进行**平滑升级**。

有关平滑升级的相关信息，参见 *rolling_updates*。

l) 设置容器的**自动伸缩**规则。

图 68: 自动伸缩规则

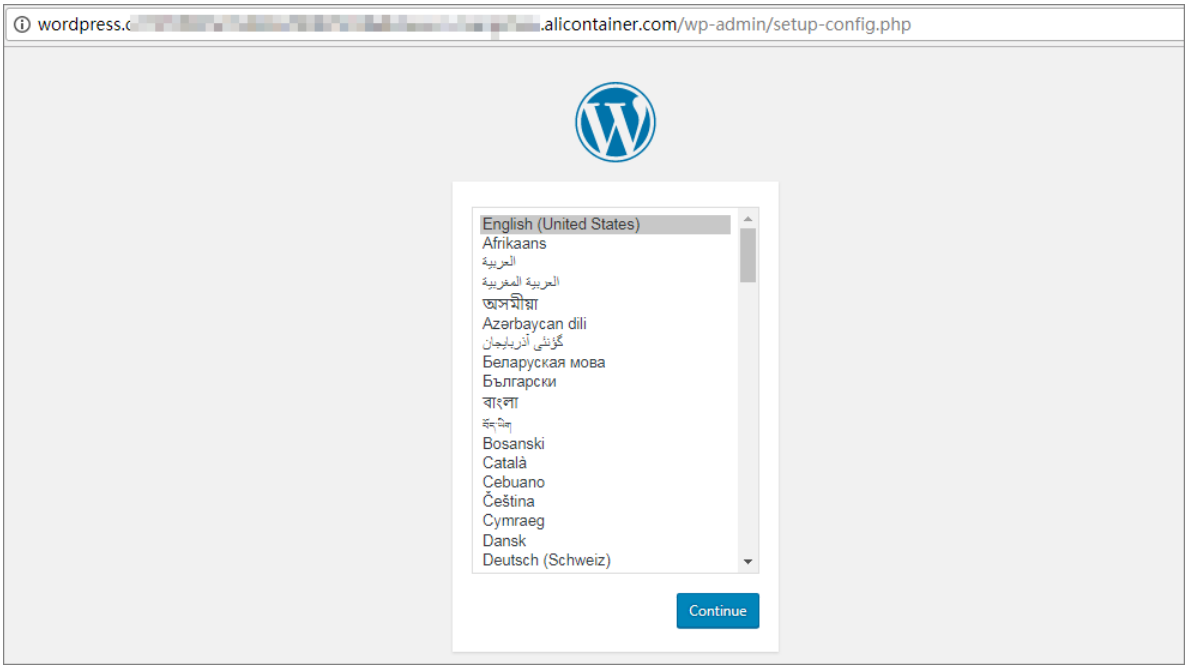


5. 设置完毕后，单击**创建**。

6. 单击**查看应用列表**，**返回应用列表** 或左侧导航栏中的 **应用**返回应用列表。单击应用名称 **wordpress**，查看应用详情。

7. 访问应用详情页面的**路由地址**，进入nginx欢迎页面

图 69: wordpress欢迎页



12.2.3 使用编排模板创建jenkins应用

您可以通过容器服务云控制台，在swarm集群下使用编排模板创建jenkins应用。

使用限制

swarm集群只支持Compose V1/V2版本的编排模板；因此，在使用编排模板创建应用时，请选择Compose V1/V2版本的模板。如果您选择了Compose V3版本的模板，系统会报错。

说明:

在编排模板列表页面，Compose V3版本的模板标注有*composev3*字样。

图 70: 报错信息



操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**，单击右上角的**创建应用**。

3. 设置应用的基本信息并单击**使用编排模板创建**。

   - **应用名称**：本示例创建的是**jenkins**应用。名称可以包含1~64个字符，包括数字，中文字符，英文字母和连字符（-）。
   - **应用版本**：所创建应用的版本。默认为1.0。
   - **部署集群**：所创建应用将要部署到的集群。
   - **默认更新策略**：应用更新的方式，您可以选择标准发布或蓝绿发布，参见发布策略概述。
   - **应用描述**：应用信息。可以不填写。该信息不能超过1,024个字符。该信息将显示在 **应用列表** 页面。
   - **检查最新Docker镜像**：选中该选项后，表示当镜像Tag不变的情况下，也会去仓库拉取最新的镜像。为了提高效率，容器服务会对镜像进行缓存。部署时，如果发现镜像Tag与本地缓存的一致，则会直接复用而不重新拉取。所以，如果您基于上层业务便利性等因素考虑，在做代码和镜像变更时没有同步修改Tag ，就会导致部署时还是使用本地缓存内旧版本镜像。而勾选该选项后，会忽略缓存，每次部署时重新拉取镜像，确保使用的始终是最新的镜像和代码。

4. 编写编排模板。

   - 您可以自己编写编排模板。
   - 或者单击**使用已有编排模板**，选择一个模板并单击**选择**。

     说明: swarm集群只支持Compose V1/V2版本的编排模板。

     您可以根据自己的需要编辑编排模板。您可以直接在编排模板中进行修改，或者在页面右侧选择需要修改的服务，单击**编辑**进行修改或者单击**删除**删除所选的服务

     图 71: 编写编排模板



此外，您还可以单击服务列表下面的**新增服务**添加服务。填写您所需要的镜像的名称，进行参数设置并单击**确定**。



5. 设置完毕后，单击**创建并部署。**

6. 返回应用列表，进入应用详情页面，单击路由地址，进入jenkins欢迎页面

图 72: Jenkins欢迎页面



12.3 Swarm mode快速开始

12.3.1 创建Swarm mode集群

您可以通过容器服务云控制台，创建一个swarm mode集群。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**集群**，单击右上角的**创建集群**。

3. 设置集群的基本信息。

   - **集群名称**： 要创建的集群的名称。可以包含1~64个字符，包括数字，中文字符，英文字符和连字符（-）。

     说明: 集群名称在同一个用户下必须唯一。

   - **地域**： 所创建集群将要部署到的地域。
   - **可用区**：集群的可用区。

4. 设置**集群模式**为swarm mode。

   集群模式有两种：swarm和swarm mode。

5. 选择集群的**网络类型**

   - 选择**经典网络**。
   - 选择**专有网络**，您需要选择一个可用的VPC和VSwitch，并设置容器的起始网段。

   图 73: 集群网络

6. 是否新增节点。

   您只能选择**添加已有节点**，您只能先创建一个零节点集群，并在集群创建成功后再添加已有云服务器。有关如何添加已有云服务器，参见添加已有云服务器。



7. 创建一个负载均衡实例。

   勾选后，系统会在创建集群时创建一个负载均衡实例。您可以通过这个负载均衡实例访问集群内的容器应用。您也可以不随集群常见负载均衡，等集群创建完成后再绑定负载均衡，参见为集群绑定负载均衡。

8. 单击**创建集群**。

   集群创建成功后，如果需要单独配置云服务器或负载均衡，可以去相应的控制台进行相关操作。

后续操作

您可以查看集群创建日志。在**集群列表**页面，选择所创建的集群并单击**查看日志**。

12.3.2 使用镜像创建tomcat应用

你可以通过容器服务云控制台，在swarm mode集群下创建tomcat应用。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**，单击右上角的**创建应用**。
3. 设置应用的基本信息并单击**使用镜像创建**。

   - **应用名称**：本示例创建的应用名称是**tomcat**。名称可以包含1~64个字符，包括数字，中文字符，英文字母和连字符（-）。
   - **应用版本**：所创建应用的版本。默认为1.0。
   - **部署集群**：所创建应用将要部署到的集群。
   - **默认更新策略**：应用更新的方式，您可以选择标准发布或蓝绿发布，参见发布策略。
   - **应用描述**：应用信息。可以不填写。该信息不能超过1,024个字符。该信息将显示在 **应用列表**页面。
   - **检查最新Docker镜像**：选中该选项后，表示当镜像Tag不变的情况下，也会去仓库拉取最新的镜像。为了提高效率，容器服务会对镜像进行缓存。部署时，如果发现镜像Tag与本地缓存的一致，则会直接复用而不重新拉取。所以，如果您基于上层业务便利性等因素考虑，在做代码和镜像变更时没有同步修改Tag ，就会导致部署时还是使用本地缓存内旧版本镜像。而勾

选该选项后，会忽略缓存，每次部署时重新拉取镜像，确保使用的始终是最新的镜像和代码。

图 74: 设置应用基本信息



4. 根据您的需要设置以下参数。

a) 设置**镜像名称**，本示例镜像名称是tomcat。

您可以直接填写目标Hub的镜像名称，填写的格式为域名*/namespace/imagename:tag*。

b) 设置**Mode**。

- **Replicated**：默认值。创建指定数量的容器。选择该选项时，您需要指定容器的数量。
- **Global**：每个节点仅包含一个容器。

c) 设置容器的启动命令（Command和Entrypoint）。

如果设置了会覆盖镜像的配置。

d) 设置服务的更新策略。

设置当服务包含多个容器时应该如何进行更新。

- **更新失败后的动作**：设置如果某一更新失败时，系统如何操作。**继续**表示失败时，继续更新下一批容器。**停止**表示失败时，停止更新服务。
- **更新间隔**：不同批次的容器更新的时间间隔。
- **允许失败比例**：更新服务时允许的更新失败占比。
- **每批更新任务数**：每次批量更新的容器的个数。

e) 设置容器的重启规则。

设置容器退出后，系统如何尝试重新启动容器。

- **重启条件**：在什么情况下重新启动容器。可以选择**不重启**、**失败时**和**总是**。默认为**总是**。
- **重启间隔**：设置尝试重启容器的时间间隔。默认为 0。
- **重启次数**：尝试重启容器的最大次数。默认为一直尝试重启。
- **检测间隔**：检测容器重启成功的时间间隔。默认值为马上检测。

f) 设置**约束条件**。

> **说明:**
>
> 如果您指定的节点约束和*config.json*中设置的**reserved_nodes**有冲突，应用部署会失败，系统提示如下错误信息。
>
> last update error:'Failed to start service xxxxxx in 180 seconds. The service is in pending state. Maybe node constraints can't be satisfied.'

g) 设置**端口映射**。本示例暴露容器的80端口。

h) 设置容器的**数据卷**。

i) 设置容器的资源限制（**CPU**和**内存**）。

- **CPU**：1.0表示整个CPU。
- **内存**：单位为Bytes。

有关容器资源限制的详细信息，参见限制容器资源。

j) 设置容器的资源预留（**CPU**和**内存**）。

k) 设置应用的**网络**。

您可以将应用添加到多个网络中。有关网络的详细信息，参见管理网络。

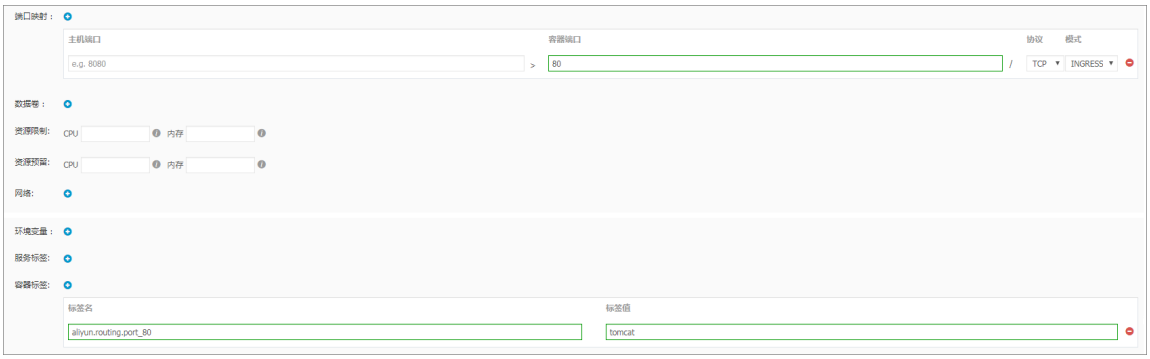
l) 设置**环境变量**。

m) 设置**服务标签**。

n) 设置**容器标签**。swarm mode集群目前仅支持通过容器标签的方式配置路由，本示例添加容器标签aliyun.routing.port_80:tomcat，具体的标签格式请参见*routing*标签

更多有关支持的标签，参见标签概览。

图 75: 设置端口映射和容器标签



o）设置**密钥**。

详细信息，参见概述。

p）设置容器的**自动伸缩**规则。

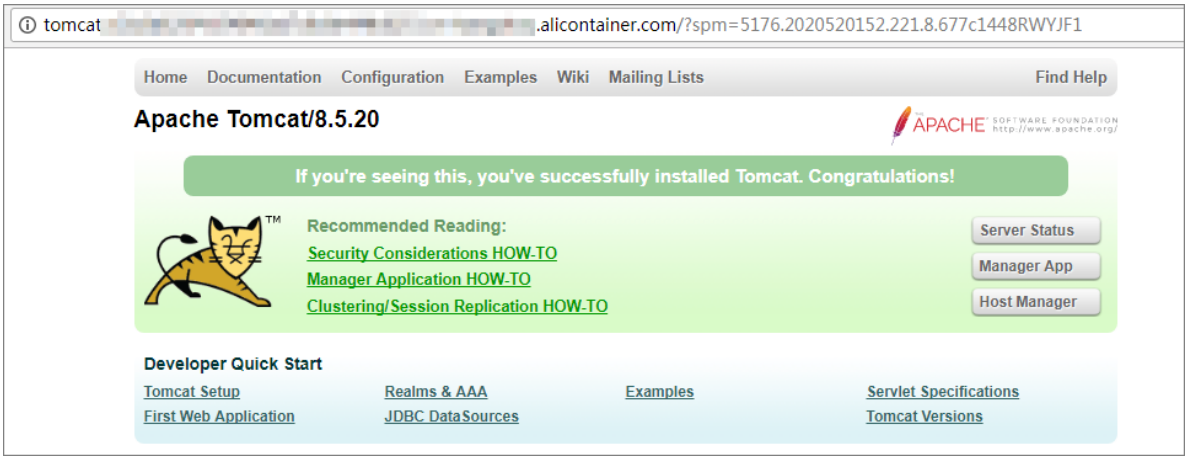
图 76: 自动伸缩规则



5. 设置完毕后，单击**创建并部署**。

6. 在应用列表单击tomcat-test应用名称，进入详情页，访问路由地址，进入tomcat应用欢迎页

图 77: 进入tomact欢迎页



12.3.3 使用编排模板创建nginx应用

你可以通过容器服务云控制台，在swarm mode集群下使用编排模板创建nginx应用。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**，单击右上角的**创建应用**。

3. 设置应用的基本信息并单击**使用编排模板创建**。

   - **应用名称**：本示例创建的应用名称是nginx-test。名称可以包含1~64个字符，包括数字，中文字符，英文字母和连字符（-）。
   - **应用版本**：所创建应用的版本。默认为1.0。
   - **部署集群**：所创建应用将要部署到的集群。
   - **默认更新策略**：应用更新的方式，您可以选择标准发布或蓝绿发布，参见发布策略概述。
   - **应用描述**：应用信息。可以不填写。该信息不能超过1,024个字符。该信息将显示在 **应用列表** 页面。
   - **检查最新Docker镜像**：选中该选项后，表示当镜像Tag不变的情况下，也会去仓库拉取最新的镜像。为了提高效率，容器服务会对镜像进行缓存。部署时，如果发现镜像Tag与本地缓存的一致，则会直接复用而不重新拉取。所以，如果您基于上层业务便利性等因素考虑，在做代码和镜像变更时没有同步修改Tag ，就会导致部署时还是使用本地缓存内旧版本镜像。而勾选该选项后，会忽略缓存，每次部署时重新拉取镜像，确保使用的始终是最新的镜像和代码。

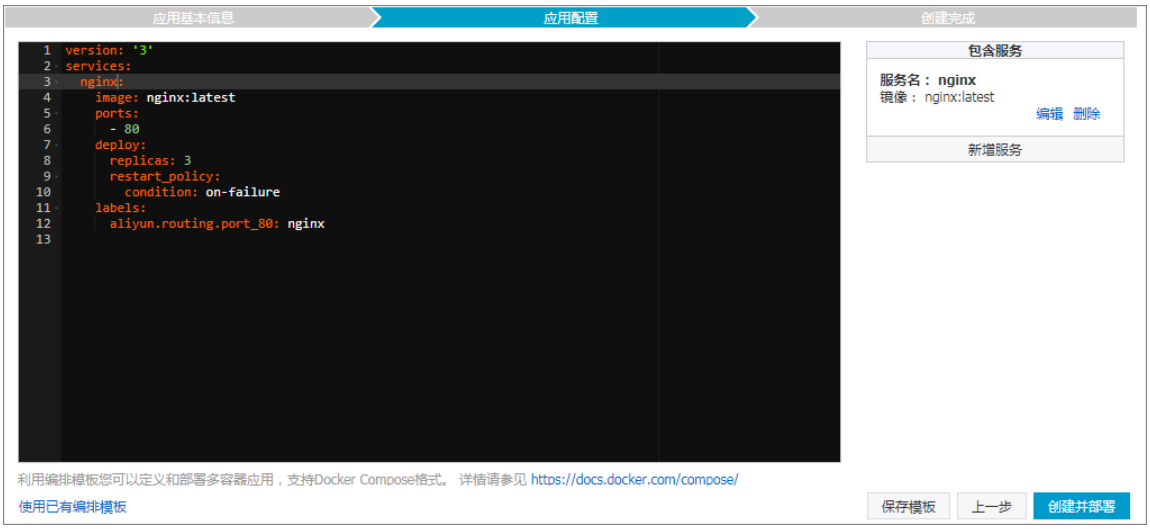
4. 编写编排模板。

   - 您可以自己编写编排模板。
   - 或者单击**使用已有编排模板**，选择一个模板并单击**选择**。

     说明: swarm mode集群支持Compose V1/V2和Compose V3版本的编排模板。

     您可以根据自己的需要编辑编排模板。您可以直接在编排模板中进行修改，或者在页面右侧选择需要修改的服务，单击**编辑**进行修改或者单击**删除**删除所选的服务

     图 78: 编写编排模板



此外，您还可以单击服务列表下面的**新增服务**添加服务。填写您所需要的镜像的名称，进行参数设置并单击**确定**。

图 79: 新增服务



5. 设置完毕后，单击**创建并部署**。
6. 返回应用列表，单击应用名称 nginx-test，进入应用详情页，单击访问路由地址，进入nginx欢迎页面。

图 80: 进入nginx欢迎页面



12.4 swarm集群指南

12.4.1 管理集群

集群是容器运行所需要的云资源组合，关联了若干服务器节点、负载均衡、专有网络等云资源。您可以通过容器服务云控制台创建和管理您的集群以及集群中的节点。

创建集群

如果您已经有若干个云服务器，可以在容器服务上创建一个零节点的集群。详细信息参见添加已有云服务器。然后将已有云服务器添加进集群。

添加已有的云服务器

您可以在云服务器上执行脚本，将云服务器手动加入集群。

详细信息参见添加已有云服务器。

管理集群

支持删除、清理、连接集群等操作。

12.4.1.1 创建集群

您可以创建一个零节点的集群，之后再绑定其他云服务器。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**集群**，单击右上角的**创建集群**。
3. 设置集群的基本信息。

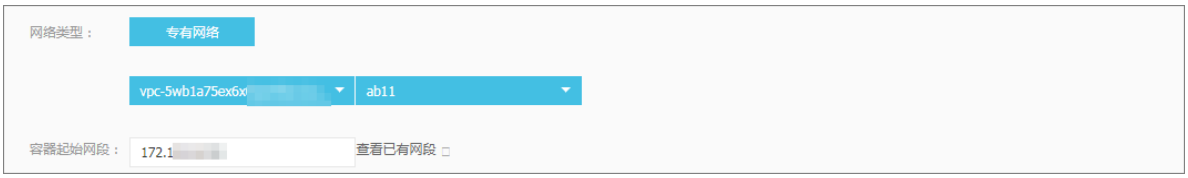
   - **集群名称**： 要创建的集群的名称。可以包含1~64个字符，包括数字，中文字符，英文字符和连字符（-）。

     说明: 集群名称在同一个用户下必须唯一。

   - **地域**： 所创建集群将要部署到的地域。
   - **可用区**：集群的可用区。

4. 集群的网络类型为VPC，您需要选择一个可用的VPC和VSwitch，并设置容器的起始网段。

   图 81: 集群网络



5. 添加节点。

   此项不可选，您只能创建一个零节点集群，并在集群创建成功后再添加已有云服务器。有关如何添加已有云服务器，参见添加已有云服务器。

6. 创建一个负载均衡实例。

   勾选后，系统会在创建集群时创建一个负载均衡实例。您可以通过这个负载均衡实例访问集群内的容器应用。您也可以不随集群常见负载均衡，等集群创建完成后再绑定负载均衡，参见为集群绑定负载均衡。

7. 单击**创建集群**。

   集群创建成功后，如果需要单独配置云服务器或负载均衡，可以去相应的控制台进行相关操作。

后续操作

您可以查看集群创建日志。在**集群列表**页面，选择所创建的集群并单击**查看日志**。

12.4.1.2 添加已有云服务器

背景信息

您可以将已有的云服务器添加到指定集群。如果之前没有创建过集群，您可以先创建一个零节点的集群。有关创建集群的详细信息，参见创建集群。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。

3. 选择所需的集群，单击**添加已有节点**。

   图 82: 添加已有节点



4. 手动添加ECS实例。

手动添加实例，您需要在云服务器上执行脚本。

a) 页面显示实例列表，选择一个ECS实例并单击**下一步**。

您一次只能添加一个实例。

图 83: 选择实例



b) 设置实例信息并单击**下一步**。

图 84: 设置实例信息



c) 页面显示专属这台云服务器的脚本命令。通过终端登录，复制命令并执行。

等待脚本执行成功，显示success。该云服务器即添加成功。

图 85: 脚本命令



图 86: 执行命令



12.4.1.3 为集群绑定负载均衡

您可以在创建集群时自动创建一个负载均衡实例，或者在集群创建完成后为集群绑定一个负载均衡实例。

容器服务集群支持绑定外网负载均衡实例，VPC 负载均衡实例和经典网络下的内网负载均衡实例

使用限制

- 您只能为集群绑定位于同一区域下的负载均衡实例。
- 不支持绑定跨账号的负载均衡实例。
- 不支持跨网络类型绑定，即不能将 VPC 负载均衡实例绑定到经典网络集群上；不支持将经典网络负载均衡实例绑定到 VPC 集群上。
- VPC 集群支持绑定外网负载均衡实例和 VPC 负载均衡实例。
- 经典网络集群支持绑定外网负载均衡实例和经典网络下的内网负载均衡实例
- 一个集群仅支持绑定一个负载均衡实例。
- 两个集群不能共用一个负载均衡实例。

前提条件

您已经在负载均衡云控制台创建了负载均衡实例并且为负载均衡实例配置了监听后端服务器的TCP 9080端口。

操作步骤

1. 登录容器服务云控制台。

有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。
3. 选择要清理的集群，单击**管理**。

图 87: 集群管理



4. 单击左侧导航栏中的**负载均衡** > **绑定SLB**。

图 88: 绑定负载均衡



5. 从下拉列表中选择您要绑定的负载均衡实例并单击**确定**。

> **说明:** 如果您所选择的负载均衡实例已经绑定了后端服务器，系统会提示您“该负载均衡实例已经绑定了后端服务器”，您需要选择其它未绑定后端服务器的负载均衡实例。

12.4.1.4 重新绑定负载均衡

您可以根据您的需要更换集群绑定的负载均衡实例。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的 **集群**。

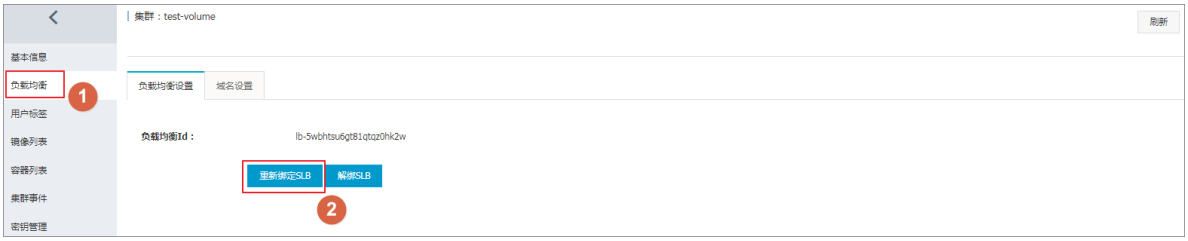
3. 选择要清理的集群，单击 **管理**。

图 89: 集群管理



4. 单击左侧导航栏中的**负载均衡 > 重新绑定SLB**。

图 90: 重新绑定



5. 从下拉列表中选择您要绑定的负载均衡实例并单击**确定**。

> **说明：** 如果您所选择的负载均衡实例已经绑定了后端服务器，系统会提示您**该负载均衡实例已经绑定了后端服务器**，您需要选择其它未绑定后端服务器的负载均衡实例。

12.4.1.5 为集群解绑负载均衡

如果您不再需要负载均衡实例，您可以通过容器服务云控制台进行解绑。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的 **集群**。

3. 选择要清理的集群，单击 **管理**。

图 91: 集群管理



4. 单击左侧导航栏中的**负载均衡 > 解绑SLB**。

图 92: 解绑负载均衡



12.4.1.6 搜索集群

您可以搜索您已经创建的集群。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。

3. 在搜索框中输入要搜索的集群的名称或关键字。名称中带有该关键字的集群将显示在集群列表中。

   说明: 搜索不区分大小写。

图 93: 搜索集群



12.4.1.7 删除集群

您可以从容器服务中删除集群。

背景信息

说明: 删除集群时，会将关联的负载均衡一起删除。请谨慎使用该操作。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。

3. 选择要删除的集群并单击**删除**。

   图 94: 删除集群



4. 在弹出的对话框中，单击**确定**。

12.4.1.8 清理集群磁盘

清理磁盘操作会清理用户集群内每台服务器上的脏数据。

背景信息

脏数据限于：

- 已下载到本地但未使用的 Docker 镜像。
- 曾经挂载到容器，但容器销毁后未清理的数据卷（volumn）目录。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。

3. 选择要清理的集群，单击**管理**。

图 95: 集群管理



4. 在集群管理页面中，单击 **清理磁盘**。

图 96: 清理磁盘



12.4.1.9 连接集群

您可以使用下载的证书通过Docker Swarm API或Docker Client连接集群暴露出来的Endpoint。

操作步骤

1. 获取访问地址。

   a) 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

   b) 单击左侧导航栏中的**集群**，在集群列表中选择一个集群并单击 **管理**。

   您可以查看集群的连接信息。

   图 97: 集群连接信息



2. 下载和保存证书。

   要通过上面的服务地址访问Docker集群，您还需要配置TLS证书。

   在集群管理页面，单击**下载证书**开始下载证书。下载到的文件为*certFile.zip*。在下面的例子中，下载的证书存放在*~/.acs/certs/cluster_name/*目录下。其中，*cluster_name*是您集群的名字。您也可以使用其他目录，但是为了便于管理，推荐您将文件存放在*~/.acs/certs/cluster_name/*目录下。

   ```
   mkdir ~/.acs/certs/cluster_name/  #替换成真正的集群名字
   cd ~/.acs/certs/cluster_name/
   cp /path/to/certFile.zip .
   unzip certFile.zip
   ```

   在这个目录里，您可以看到三个文件 *ca.pem*，*cert.pem* 和 *key.pem*。

3. 管理集群。

   - **使用Docker Client管理集群**

     您可以使用Docker Client访问容器服务上的容器集群。访问集群之前，您需要使用以下两种方法之一配置使用的证书和服务地址。

     - 在命令行参数里配置证书

       ```
       docker --tlsverify --tlscacert=~/.acs/certs/cluster_name/ca.pem --tlscert=~/.acs/certs/ClusterName/cert.pem --tlskey=~/.acs/certs/ClusterName/key.pem \
       -H=集群接入点地址 ps
       ```

       把*cluster_name*和集群接入点地址替换成您实际使用的路径和访问地址。

     - 使用环境变量

       ```
       export DOCKER_TLS_VERIFY="1"
       export DOCKER_HOST="集群接入点地址"
       export DOCKER_CERT_PATH=~/.acs/certs/cluster_name
       ```

```
docker ps
```

把集群接入点地址替换成您实际使用的访问地址，把*cluster_name*替换成您实际使用的路径。

上面的两个例子在集群上执行了docker ps命令，您可以把ps替换成任何其他Docker支持的命令，比如使用docker run启动一个新的容器。

- **使用Docker Compose管理集群**

  目前Docker Compose支持通过环境变量声明服务地址和证书。

```
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="集群接入点地址"
export DOCKER_CERT_PATH=~/.acs/certs/cluster_name
docker-compose up
```

如果在使用过程中不慎导致证书泄露，您需要尽快吊销证书。在集群管理页面，单击**吊销已下载证书**可以吊销您之前下载的证书。吊销之后，您可以重新下载证书。



说明: 单击**吊销已下载证书**将导致您之前下载的证书不可用。

12.4.1.10 升级系统服务

在创建集群时，容器服务会自动为集群创建系统服务。您可以通过容器服务云控制台升级系统服务。

集群的系统服务用来解决应用需要的通用服务，例如日志服务acslogging、路由服务acsrouting、volume服务acsvolumedriver和监控服务acsmonitoring。



说明: 集群的系统服务在升级的过程中会导致您的应用或者服务短暂不可访问或者不能正常工作。请谨慎升级。建议选择访问低谷或者维护时间进行升级。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。
3. 选择所需的集群，单击**管理**。

图 98: 集群管理



4. 单击**升级系统服务**。
5. 在弹出的对话框中，选择要升级的系统服务并单击**升级**。

   例如，本示例中选择的是**简单路由服务**（对应acsrouting，注意升级会短暂影响用户应用的访问），**volume服务**（对应acsvolumedriver，注意升级可能会短暂影响用户相关联应用的功能）。

12.4.2 管理节点

12.4.2.1 移除节点

移除节点可以将服务器从集群中摘除，移除后将不能在节点信息内看到该服务器信息。

说明:

- 在移除节点之前请先做好备份工作。
- 移除节点仅仅是从集群中移出ECS，并不会释放ECS。如果有需要释放ECS，需要到ECS云控制台自行释放。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。
2. 单击左侧导航栏中的**节点**。
3. 选择要移除的节点所在的集群。
4. 选择要移除的节点，单击**更多** > **移除节点**。

图 99: 移除节点



5. 在弹出的确认对话框中，单击**确定**。

12.4.2.2 查看节点上运行的容器

您可以通过节点列表页面查看运行在某个节点上的容器。

操作步骤

1. 登录容器服务云控制台。

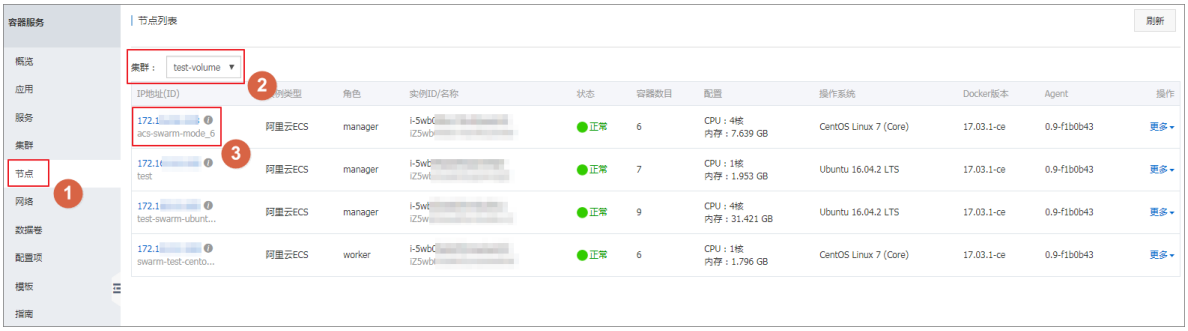
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**节点**。

3. 选择节点所在的集群。

4. 选择所需的节点，单击节点的ID。

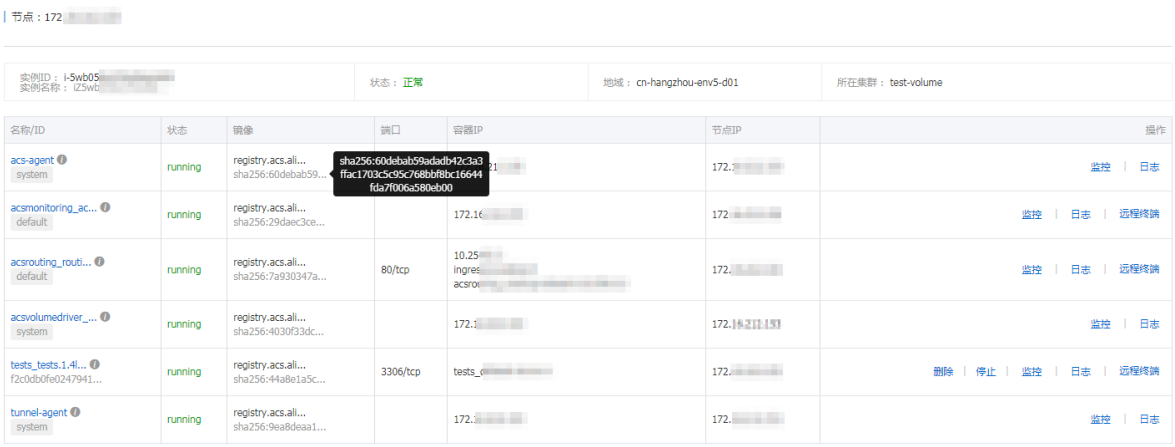
图 100: 查看节点



5. 您可以看到运行在该节点上的容器的列表。

   通过该列表，您可以查看容器的标签（Labels），查看容器的镜像并查看镜像的sha256，查看容器的日志和监控信息，并进行容器相关操作（包括：启停容器、删除容器、通过远程终端操作容器）。

图 101: 容器列表



12.4.3 安全组

12.4.3.1 容器服务安全组规则

容器服务集群安全组开放的规则如下。

VPC安全组

- 为方便用户业务的 Web 服务，默认开启443 端口和 80 端口，可以根据自己的需求选择放开或者关闭。
- ICMP 规则建议保留，用于节点间通信，方便排查问题。有些工具也依赖 ICMP。
- 容器服务依赖 22 端口对机器初始化，方便 SSH 到用户机器进行集群的配置。
- VPC 安全组授权对象设置的是容器网段的基础地址，本示例是 172.18.0.0/16地址段，与您在创建 VPC 集群时选择的容器初始网段有关，也可以选择其他网段，参见创建集群。保证容器之间可以互相通信。

安全配置原则

- 每个集群一个安全组。

  容器服务每个集群都管理了一个安全组。您可以在这个安全组上配置规则。
- 最小权限原则。

  为了您集群的安全性，安全组应该对外开放最小的权限。
- 容器服务创建的安全组添加了一些默认规则。

  为了方便用户操作ECS实例，容器服务创建的安全组添加了一些默认规则，开放了诸如80/443等端口。如果不需要，您可以删除这些规则。

说明: 不要屏蔽22和2376端口。容器服务需要通过这两个端口初始化机器。

- 尽量使用容器内部网络进行通信，不将通信暴露到宿主机上。
- 授权其他ECS实例访问安全组时，授权给安全组，而非单个IP。

  要授权其他ECS实例访问当前安全组，先创建一个新安全组，把要访问当前安全组的ECS实例加入新安全组，再授权新安全组访问当前安全组。
- 优先使用VPC网络。如非必要，节点不要绑定EIP， VPC网络的隔离性更好。
- VPC内网出/入方向里要放开容器的网段。

  如果不放开，会导致容器之间网络不通。

12.4.3.2 检测安全组

您可以通过容器服务云控制台检测集群的安全组规则是否安全。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。
2. 单击左侧导航栏中的**集群**。
3. 选择目标集群，单击**管理**。

   图 102: 集群管理



4. 单击**检测安全组**。系统会检测集群安全组的状态。

   如果安全组规则正常，系统会显示如下提示框，单击**取消**即可。



若果您的安全组规则存在安全风险，系统会弹出提示框列出存在安全风险的安全组规则。单击**修复**，系统会删除存在安全风险的安全组规则并创建新的安全组规则

12.4.3.3 重新绑定安全组

您可以通过容器服务云控制台重新为集群绑定安全组，系统会重新创建一个安全组并将新创建的安全组绑定到集群，同时将集群中的节点添加到新创建的安全组。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。

3. 选择目标集群，单击**管理**。

   图 103: 集群管理



4. 单击**重新绑定安全组**。

5. 在弹出的对话框中单击**确定**。

12.4.4 管理编排模板

您可以通过容器服务云控制台创建并查看编排模板。

12.4.4.1 创建编排模板

您可以通过容器服务云控制台创建您自己的编排模板。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**模板**。
3. 单击右上角的**创建**。
4. 在**创建编排**页面，填写模板信息。

   - **名称**：模板的名称。
   - **描述**：模板的相关信息。
   - **内容**：Docker Compose的yml文件。有关文件的详细信息，参见 *Compose File*的详细说明。

     编排模板中包含的服务会显示在页面的右侧。您可以单击**编辑**，在弹出的**新建服务**对话框中通过参数修改编排模板，或者单击**删除**，删除所选服务。

5. 单击**创建编排**。

   您可以在**编排列表**页面中**我的编排**下查看所创建的编排模板。您可以单击**详情**，查看编排模板的详细信息，或者单击**创建应用**，使用该编排模板创建应用。

12.4.4.2 查看编排模板列表

您可以通过容器服务云控制台查看您所创建的编排模板。

操作步骤

1. 登录容器服务云控制台。
2. 单击左侧导航栏中的**模板**。

   您可以单击**搜索**通过模板的名称前缀搜索模板。

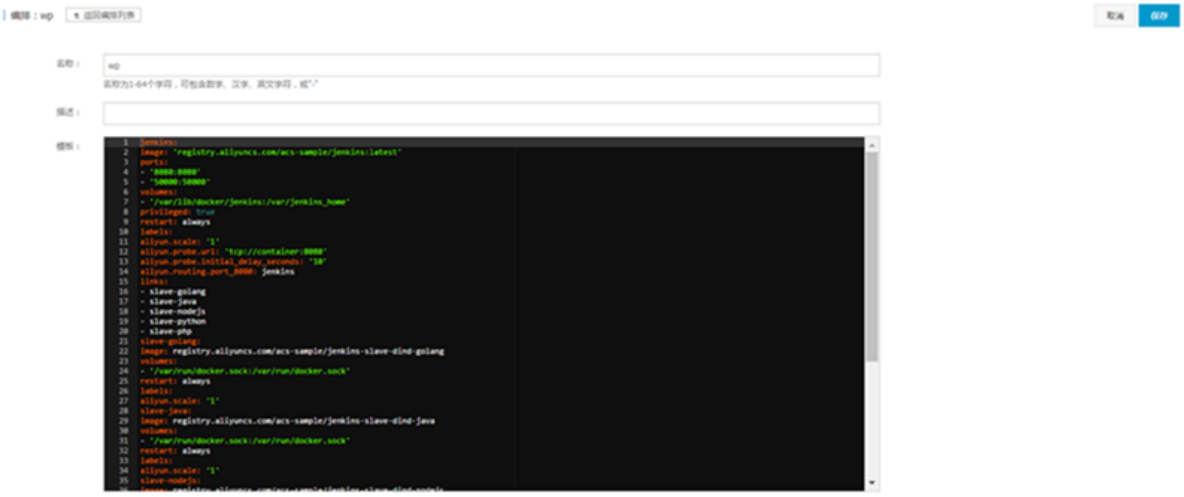
12.4.4.3 更新编排模板

您可以通过容器服务云控制台修改编排模板。

操作步骤

1. 登录容器服务云控制台。

2. 单击左侧导航栏中的**模板**。
3. 选择一个模板并单击**详情**。
4. 单击右上角的**编辑**。
5. 编辑模板内容。



6. 单击右上角的**保存**保存模板。

12.4.4.4 下载编排模板

您可以通过容器服务云控制台将编排模板下载到本地。

操作步骤

1. 登录容器服务云控制台。
2. 单击左侧导航栏中的**模板**。
3. 选择一个模板并单击 **详情**。
4. 单击右上角的**下载**，会立即下载后缀为*yml*格式的模板文件。



12.4.4.5 删除编排模板

您可以通过容器服务云控制台删除不再使用的编排模板。

操作步骤

1. 登录容器服务云控制台。
2. 单击左侧导航栏中的**模板**。
3. 选择一个模板并单击**详情**。
4. 单击右上角的**删除**。



5. 在弹出的确认对话框中，单击**确定**。

12.4.4.6 另存编排模板

您可以将编排模板另存为一个新的编排模板。

操作步骤

1. 登录容器服务云控制台。
2. 单击左侧导航栏中的**模板**。
3. 选择一个模板并单击**详情**。
4. 单击右上角的**另存为**。
5. 填写新的编排模板名称并单击**确定**。

   另存的编排模板会显示在**编排列表**页面中**我的编排**下。

12.4.5 服务编排

容器服务支持Docker Compose编排模板来描述多容器应用。

编排模板允许您描述一个完整的应用，该应用可以由许多个服务组成。例如：一个门户网站应用，由一个Nginx服务、一个Web服务和一个数据库服务组成。

一个服务可能会有多个容器实例，所有容器实例的配置保持一致。例如：上述应用中的Web服务，就可以根据访问量需要启动两个甚至更多的容器。

能力

容器服务支持通过编排模板文件，自动化地部署和管理一个容器应用。

编排模板文件使用的标签兼容大部分Docker Compose 1.5.x到1.7.x版本实现的标签。有关具体兼容的标签，参见 标签概览。

swarm集群支持Compose V1/V2版本的模板格式。更多详细信息，参见 文档。

容器服务也在社区版本之上提供了很多扩展能力：

- 与社区的Docker Compose和Swarm不同，阿里云容器服务支持跨节点的容器连接（link），所以您可以直接将Docker Compose模板描述的应用部署到分布式集群上来提供高可用性和可伸缩性。
- 容器服务也在社区Compose模板描述的基础上提供了一系列扩展来简化Web、微服务应用的部署和运维。更多详细信息，参见 标签概览。

示例

下面是一个WordPress应用，包含了由WordPress镜像提供的Web服务和MySQL镜像提供的db服务。

```
web:
  image: wordpress:4.2
  ports:
    - "80"
  environment:
    - WORDPRESS_AUTH_KEY=changeme
    - WORDPRESS_SECURE_AUTH_KEY=changeme
    - WORDPRESS_LOGGED_IN_KEY=changeme
    - WORDPRESS_NONCE_KEY=changeme
    - WORDPRESS_AUTH_SALT=changeme
    - WORDPRESS_SECURE_AUTH_SALT=changeme
    - WORDPRESS_LOGGED_IN_SALT=changeme
    - WORDPRESS_NONCE_SALT=changeme
  restart: always
  links:
    - db:mysql
  labels:
    aliyun.log_store_wordpress: stdout
    aliyun.probe.url: http://container/license.txt
    aliyun.probe.initial_delay_seconds: "10"
    aliyun.routing.port_80: wordpress;http://www.example.com;https://www.nice.com
    aliyun.scale: "3"
db:
  image: mysql:5.6
  environment:
    MYSQL_ROOT_PASSWORD: password
  restart: always
  labels:
```

```
aliyun.log_store_mysql: stdout
```

12.4.5.1 标签概览

容器服务编排模板文件使用的标签兼容大部分 *Docker Compose*1.5.x到1.7.x版本实现的标签，并在社区版本的基础上提供了很多扩展能力。

扩展能力的标签

容器服务扩展了编排模板的部署和生命周期管理能力，所有扩展能力都被描述在labels标签下面，作为子标签使用。

标签	说明
probe	设置服务的健康性检查。
rolling_updates	设置服务滚动更新。
depends	设置服务的依赖关系。
scale	设置该服务的容器数量，横向扩展服务。
routing	设置该服务的访问域名。
routing.session_sticky	设置routing在做请求路由的时候，是否保持session sticky，即会话保持。
lb	通过自定义阿里云负载均衡nat映射的方式来暴露服务端口到公网或者内网。
global	设置该服务为全局服务。

功能增强的标签

容器服务提供 服务部署约束#affinity:service# 标签用来设置该服务的部署约束条件。

额外支持的标签

标签	说明
external	设置该服务直接链接到外部地址。
dns_options	设置DNS选项，和docker run命令中的--dns-opt参数语义一致。
oom_kill_disable	设置是否禁止OOM Killer，和docker run命令中的--oom-kill-disable参数语义一致。

变量替换

容器服务支持参数化的Docker Compose模板。模板中可以包含环境变量作为参数，当模板部署时会提示输入参数值，并在部署时对模板进行变量替换。

更多详细信息，参见 变量替换。

容器重新调度

容器服务支持对Docker容器的重新调度：当一个节点失效时，容器可以被自动调度到其他可用节点自动运行。

更多详细信息，参见 容器重新调度。

不支持的Docker Compose标签

容器服务暂不支持Docker Compose的部分标签。有关容器服务暂不支持的标签，参见 不支持的*Docker Compose*标签。

12.4.5.2 probe

设置服务的健康性检查。

- 通过URL进行检查，支持HTTP协议、TCP协议。
- 通过shell脚本检查。

健康检查会从容器宿主机上发起，每隔一定时间（默认两秒）向容器发起请求或在容器上执行shell脚本命令。

检查成功的判断条件

- HTTP请求的返回码为2XX/3XX；TCP端口可建立连接。
- shell脚本运行返回值为0。

检查的字段说明

- aliyun.probe.url：HTTP、TCP请求的URL。您不需要填写自己的域名或者IP地址，只需要加上container这个单词，该URL最终会被解析成容器相应的IP去进行健康检查，检查结果返回2XX或者3XX才认为服务是健康的。
  - 例如，容器通过8080端口提供HTTP服务，并提供了/ping作为健康检查的URL，则探测URL的格式为http://container:8080/ping，容器服务会自动通过HTTP GET请求检查URL的返回结果，如果返回结果的返回码为2XX或者3XX，则说明健康检查成功。

  - 例如，MySQL容器侦听3306端口，探测URL的格式为tcp://container:3306，服务会检查容器3306端口是否打开，如果打开则说明健康检查成功。
- aliyun.probe.cmd：健康检查执行的检查shell命令，/check.sh；容器服务会定期在容器内执行该命令，当shell脚本返回值为0时表明健康检查成功。
- aliyun.probe.timeout_seconds：健康检查的超时时间。
- aliyun.probe.initial_delay_seconds：在容器启动后延迟几秒开始健康检查。



说明:

- 一个服务中只能包含aliyun.probe.url和aliyun.probe.cmd其中之一。
- 如果服务不包含aliyun.probe.url或aliyun.probe.cmd，则容器缺省为健康状态，且其他aliyun.probe.xxx标签会被忽略。

示例

利用URL检测容器健康状态。

```
os:
  image: my_nginx
  labels:
    aliyun.probe.url: http://container/ping
    aliyun.probe.timeout_seconds: "10"
    aliyun.probe.initial_delay_seconds: "3"
```

利用shell脚本检测容器健康状态。

```
os:
  image: my_app
  labels:
    aliyun.probe.cmd: health_check.sh
    aliyun.probe.initial_delay_seconds: "3"
```

12.4.5.3 rolling_updates

更新某个服务时，如果该服务包括超过一个以上容器（使用scale标签定义），在第n个容器更新成功后，再去做第n+1个容器的更新，以此来最小化停止服务时间。

编排示例

部署WordPress服务，通过scale标签指定部署2个容器，使用rolling_updates标签可以使WordPress对外停止服务的时间最小化。

```
web:
  image: wordpress
  ports:
    - 80
  restart: always
  links:
```

```
    - 'db:mysql'
  labels:
    aliyun.logs: /var/log
    aliyun.probe.url: http://container/license.txt
    aliyun.probe.initial_delay_seconds: '10'
    aliyun.routing.port_80: http://wordpress
    aliyun.rolling_updates: 'true'
    aliyun.scale: '2'
db:
  image: mariadb
  environment:
    MYSQL_ROOT_PASSWORD: example
  restart: always
  labels:
    aliyun.logs: /var/log/mysql
```

parallelism

您可以使用parallelism标签定义rolling_updates每次并行更新的容器数量。



说明: 此标签必须和rolling_updates配合使用，单独使用无效。

取值：

- 默认值为1，即每次只更新一个容器。
- 当其值大于1的时候，rolling_updates过程中，每次会以parallelism定义的值来并行更新相应个数的容器，实现批量更新。
- 当定义值无效时，默认为1。



说明: 为了确保始终有容器在提供服务，建议parallelism定义的值小于服务包含的容器数。

编排示例

下面的示例部署Nginx服务，通过scale标签部署 3 个容器，使用rolling_updates和parallelism标签定义每次以 2 个容器为单位来进行批量更新。

```
web:
  image: nginx:latest
  restart: always
  environment:
     - "reschedule:on-node-failure"
  ports:
     - 80
  labels:
    aliyun.scale: "3"
    aliyun.probe.url: http://container:80/
    aliyun.probe.timeout_seconds: "10"
    aliyun.probe.initial_delay_seconds: "3"
    aliyun.rolling_updates: 'true'
```

```
    aliyun.rolling_updates.parallelism: "2"
```

12.4.5.4 depends

设置服务的依赖关系。

设置之后，容器服务可以控制容器的启动顺序，逐个启动容器。

编排示例



说明: 多个依赖使用逗号（,）分隔。

```
web:
  image: wordpress:4.2
  ports:
    - 80
  links:
    - db:mysql
  labels:
    aliyun.depends: db,redis
db:
  image: mysql
  environment:
    - MYSQL_ROOT_PASSWORD=password
redis:
  image: redis
```

12.4.5.5 scale

设置该服务的容器数量，横向扩展服务。

在Compose V1/V2中，Docker Compose只能在每一个服务中启动一个容器，如果需要扩展容器数量，需要在启动后手动进行设置。

现在通过scale的扩展标签，支持您在容器启动的时候进行扩展。

此外，在容器被删除之后，您可以在容器服务云控制台对应用进行重新部署（单击左侧导航栏中的**应用** > **选择目标应用** > **重新部署**），容器服务会重启或新建容器使容器恢复到指定数量。

编排示例

```
web:
  image: wordpress:4.2
  ports:
    - 80
  links:
    - db:mysql
  labels:
    aliyun.scale: "3"
db:
  image: mysql
  environment:
```

```
    - MYSQL_ROOT_PASSWORD=password
```

12.4.5.6 routing标签

设置该服务的访问域名。

格式：

aliyun.routing.port_*$contaienr_port*: [http://]*$domain*|*$domain_prefix[:$context_path]*

基本概念：

- $contaienr_port：容器端口，注意该处不是主机的端口。
- $domain：域名，需要用户填写自己的域名。
- $domain_prefix：域名前缀，如果填写域名前缀，容器服务会提供给您一个测试用的域名，域名后缀是wordpress.<*cluster_id*>.<*region_id*>.alicontainer.com。
- $context_path：请求的路径，即可以根据请求的路径来选择区分不同的后端容器。

绑定域名的选择：

- 如果使用HTTP协议暴露服务，可以使用容器服务提供内部域名（顶级域为alicontainer.com），供您测试使用，也可以使用您提供的域名。
- 如果使用HTTPS协议，那么仅支持配置您提供的域名，例如www.example.com。您需要修改DNS设置将域名指定到容器集群提供的负载均衡服务上。

标签声明的格式要求：

- 容器服务为每一个集群分配了子域名，绑定内部域名只需要给出域名的前缀，域名前缀仅表示域名的一级，不能使用点号（.）进行分隔。
- 如果您不指定scheme，则默认使用HTTP协议。
- 域名的长度不能超过128个字符，context root的长度不能超过128个字符。
- 绑定多个域名到服务时，域名之间用分号（;）隔开。
- 一个后端服务可以有多个端口，该端口指的是容器暴露的端口，一个端口只能使用一条label进行声明，带有多个端口的服务需要声明多个label。

编排示例

使用routing标签将容器服务提供的内部域名wordpress.*cluster_id*.*region_id*.alicontainer.com绑定到Web服务的80端口，并且将您提供的自有域名http://wp.sample.com/context绑定到Web服务的80端口。

```
web:
```

```
  image: wordpress:4.2
  links:
    - db:mysql
  labels:
    aliyun.routing.port_80: wordpress;http://wp.sample.com/context
db:
  image: mysql
  environment:
    - MYSQL_ROOT_PASSWORD=password
```

最终您得到的内部域名为 wordpress.*cluster_id*.cn-beijing.alicontainer.com。

Web 服务运行之后，您可以通过 http://wordpress.*cluster_id*.cn-beijing.alicontainer.com 或者 http://wp.sample.com/context 访问相应的 Web 服务。

如果您需要支持 HTTPS 服务，需要自行通过负载均衡云控制台上传 HTTPS 证书，并绑定相应的集群对外访问负载均衡端点。

routing.session_sticky

设置 routing 在做请求路由的时候，是否保持 session sticky，即会话保持。其效果是，在某个会话时间内，请求一直路由到同一个后端的容器，而不是每次请求都随机路由到不同的容器。

> **说明：** 只有当您已经设置了 aliyun.routing.port_$contaienr_port 时，该设置才能起作用。

其设置方法如下：

- 开启会话保持

  aliyun.routing.session_sticky: true

- 关闭会话保持

  aliyun.routing.session_sticky: false

模板编排文件示例：

```
web:
  image: wordpress:4.2
  links:
    - db:mysql
  labels:
    aliyun.routing.port_80: wordpress;http://wp.sample.com/context
    aliyun.routing.session_sticky: true
db:
  image: mysql
  environment:
```

```
    - MYSQL_ROOT_PASSWORD=password
```

12.4.5.7 lb

通过自定义阿里云负载均衡nat映射的方式来暴露服务端口到公网或者到内网。

标签格式如下，带$的变量为占位符。

`aliyun.lb.port_$container_port:$scheme://$[slb_name|slb_id]:$slb_front_port`

编排示例

```
web:
  image: wordpress:4.2
  ports:
    - 7777:80
    - 9999:9999
    - 8080:8080
    - 53:53/udp
  links:
    - db:mysql
  labels:
    aliyun.lb.port_80: http://slb_example_name:8080
    aliyun.lb.port_9999: tcp://slb_example_name:9999
    aliyun.lb.port_8080: https://14a7ba06d3b-cn-hangzhou-dg-a01:80
    aliyun.lb.port_53: udp://14a7ba06d3b-cn-hangzhou-dg-a01:53
db:
  image: mysql
  environment:
    - MYSQL_ROOT_PASSWORD=password
```

要使用好自定义负载均衡的lb标签，您需要理解请求路由过程中的3个端口，即负载均衡的前端端口，负载均衡的后端端口（也就是ECS实例的端口），最后就是容器的端口。以第一个lb标签aliyun.lb.port_80为例，从左往右看，在key中的80端口指的是容器要暴露的端口，后面的8080端口指的是负载均衡要暴露的前端端口。负载均衡的后端端口是ECS实例的端口，可从标签ports的主机：容器端口映射中获取到，由此，您可以查到容器端口80对应的主机端口是7777，因此确定了负载均衡转发的后端端口是7777端口。因此第一个标签说明了当向服务Web发起请求时，首先通过负载均衡前端的8080端口进入，转发到后端主机的7777端口，然后再根据端口映射ports的声明，请求最终从容器端口80进入，交由容器内的WordPress进程提供服务。接下来的标签以此进行相同的解释。该标签配置的负载均衡均不经过集群内置的routing服务，请求的路由由您自己控制。

标签声明的格式要求：

- 指明负载均衡实例时，可以使用负载均衡实例的名称或者负载均衡实例的ID。
- 负载均衡实例名称的限制为1~80个字符，允许包含字母、数字、连字符（-）、正斜杠（/）、点号（.）、下划线（_）。
- 容器端口限制为1~65535。

- 负载均衡前端端口的限制为1~65535。

带有自定义负载均衡nat映射的服务部署限制：

- 您需要自己创建负载均衡实例，对负载均衡实例命名，并创建对应监听端口，然后以扩展标签的方式提供负载均衡实例的名称$slb_name或者$slb_id，以及要暴露的端口，使用的协议$scheme（可能的值有tcp、http、https、udp），映射的容器端口$container_port，指定负载均衡实例的前端端口$slb_front_port。
- 您必须指定服务要暴露端口的主机和容器端口的映射，通过Dockerfile标准的标签ports指定，注意必须指定主机端口，且与其他服务映射的主机端口不能冲突，需要主机的端口用于负载均衡绑定后端的ECS实例。
- 一个服务只能使用一个或者多个负载均衡实例进行服务端口的暴露，因多个服务会分布在不同的后端，多个服务不能共享使用同一个负载均衡实例。
- 部署了带有负载均衡nat映射的服务的主机使用相同的主机：容器端口映射，因此这些服务在每台主机上只有一个实例。
- 支持的负载均衡协议$scheme包括tcp、http、https、udp协议。
- 您需要自行在阿里云负载均衡官方控制台创建监听的端口。
- 请自行登录负载均衡官方控制台对使用在容器服务中的负载均衡实例进行具体的配置修改，例如带宽限制等配置。
- lb标签的价值在于您不需要自行绑定负载均衡后端的ECS服务器，只需要配置好相应的标签，就会帮助您完成绑定后端的操作。因此，除了绑定负载均衡后端的操作，您对负载均衡的设置和修改需要自行在阿里云负载均衡云控制台上完成。
- 容器服务会帮助您生成一个RAM子账户（需要您开通RAM），使用这个具有部分负载均衡权限（没有创建和删除负载均衡的权限）的账号帮助您管理在容器服务中使用的负载均衡实例，例如绑定集群中某些节点作为服务的后端。
- 在服务的整个生命周期内，lb标签会一直生效，除非服务被删除，或者lb标签删除之后重新部署了服务，在此期间，配置在lb标签内的负载均衡实例不能混用。

12.4.5.8 global

设置该服务为全局服务。

有一些服务需要在每一个节点部署，例如监控或是日志类的服务。并且在新的节点建立的时候就对这个节点进行服务的部署。

当一个服务被设置为global时，该服务会在集群中的每一个节点进行部署。当集群中有新增节点时，也会自动部署一个容器实例到新节点之上。

```
monitor:
  image: sample
  labels:
    aliyun.global: true
```

12.4.5.9 服务部署约束（affinity:service）

设置该服务的部署约束条件。

容器服务支持Docker Swarm兼容的容器部署约束条件，您可以通过Docker Swarm Filter控制一个容器的部署。

但是在社区版Docker Compose中，却并没有相关的能力来控制服务直接的部署约束。

在容器服务中，您可以在environment中添加相关affinity:service，来约束服务之间的亲和度（Affinity），达到控制服务部署策略的功能。支持服务之间的Soft affinity和Hard affinity。

编排示例

本示例中，slave服务设置了affinity.service!=master的部署约束。使得slave服务一定会选择没有部署master服务的节点，这样当一个节点失效时，服务可用性不受影响。当您的集群只有一个节点的时候，由于指定的是hard anti-affinity，该部署会失败，因为部署没有办法满足所指定的强约束条件。

```
master:
  image: mysql:5.6
  environment:
    - MYSQL_USER=user
    - MYSQL_PASS=test
    - REPLICATION_MASTER=true
    - REPLICATION_USER=repl
    - REPLICATION_PASS=repl
  ports:
    - 3306
slave:
  image: mysql:5.6
  environment:
    - MYSQL_USER=user
    - MYSQL_PASS=test
    - REPLICATION_SLAVE=true
    - affinity:service!=master
  ports:
    - 3306
  links:
```

```
    - master:mysql
```

12.4.5.10 external

设置该服务直接链接到外部地址。

扩展字段下有以下字段可以使用：

- host：设置链接的域名。
- ports：设置链接的端口。

编排示例

不使用external，直接启动一个MySQL容器。

```
web:
  image: wordpress:4.2
  ports:
    - 80
  links:
    - db:mysql
db:
  image: 10.32.161.160:5000/mysql
  environment:
    - MYSQL_ROOT_PASSWORD=password
```

通过external，描述一个并没有部署在集群中的RDS服务，并提供给部署在集群中的WordPress使用。

```
wordpress:
  image: wordpress:4.2
  ports:
    - 80
  links:
    - db:mysql
  environment:
    - WORDPRESS_DB_USER=cloud
    - WORDPRESS_DB_PASSWORD=MYPASSWORD
    - WORDPRESS_DB_NAME=wordpress
db:
  external:
    host: rdsxxxx.mysql.rds.aliyuncs.com
    ports:
      - 3306
```

12.4.5.11 dns_options

设置DNS选项，和docker run命令中的--dns-opt参数语义一致。

```
wordpress:
  image: wordpress:4.2
  dns_options:
```

```
    - "use-vc"
```

12.4.5.12 oom_kill_disable

设置是否禁止OOM Killer，和docker run命令中的--oom-kill-disable参数语义一致。

```
wordpress:
  image: wordpress:4.2
  oom-kill-disable: true
```

12.4.5.13 变量替换

容器服务支持参数化的Docker Compose模板。模板中可以包含环境变量作为参数，当模板部署时会提示输入参数值，并在部署时对模板进行变量替换。

比如，您可以定义参数POSTGRES_VERSION。

```
db:
  image: "postgres:${POSTGRES_VERSION}"
```

当部署上面的Compose模板的时候，容器服务会提示您输入POSTGRES_VERSION参数值，比如9.3。容器服务会根据参数值对Compose模板进行变量替换。在本示例中，会部署一个postgres:9.3的容器。

容器服务完全兼容Docker Compose的语法，可以在模板中使用$VARIABLE或者${VARIABLE}格式的语法。

在Compose模板中可以使用$$来对需要包含$的字符串进行转义，这样容器服务不会错误地将其作为参数来进行处理。

关于Compose模板支持变量替换的详细信息，参见*Variable substitution*。

12.4.5.14 容器重新调度

容器服务支持对Docker容器的重新调度：当一个节点失效时，容器可以被自动调度到其他可用节点自动运行。

缺省情况下，容器的重新调度策略是关闭的。根据需要，您可以用如下配置来让重调度策略生效。

容器服务提供兼容Docker Swarm的容器重新调度策略，可以通过环境变量方式或者label方式启动。

环境变量：

```
redis:
  image: redis
  environment:
```

```
    - reschedule:on-node-failure
```

Label：

```
redis:
  image: redis
  labels:
    - com.docker.swarm.reschedule-policies=["on-node-failure"]
```

说明： 如果重新调度容器之后，需要恢复Docker容器所需的持久化状态，需要配合支持数据迁移或共享的Docker文件卷。

12.4.5.15 不支持的Docker Compose标签

标签	说明
build	build标签用于使用当前目录中的Dockerfile文件和其他文档进行容器镜像构建。目前容器服务暂不提供构建镜像功能，推荐您将构建和部署的动作分开处理：您可以利用阿里云的镜像仓库 直接从代码源构建镜像，或者将本地构建的镜像推送到镜像仓库；您可以在编排模板中使用image标签引用镜像仓库（包括私有仓库）中的镜像。
dockerfile	说明同build标签。
env_file	容器服务暂不支持以文件方式指定环境变量，您可以通过environment标签添加环境变量。
mac_address	暂时不支持MAC地址的设置。
detach	容器服务的所有镜像都是以detach模式启动的，不允许您指定attach方式执行。
stdin_open	说明同detach标签。
tty	说明同detach标签。
networks	Compose V2中的网络允许服务的容器启动在自定义的网络中，容器服务的容器都是在同一个跨主机互通的容器网络，所以不支持您在 Compose V2中使用networks标签。关于容器服务的网络管理和服务发现，参见跨主机互联的容器网络。

12.4.6 管理应用

您可以创建应用，限制容器的资源，设置应用的部署以及变更应用的配置等。

12.4.6.1 使用镜像创建应用

您可以通过容器服务云控制台使用镜像创建应用。

操作步骤

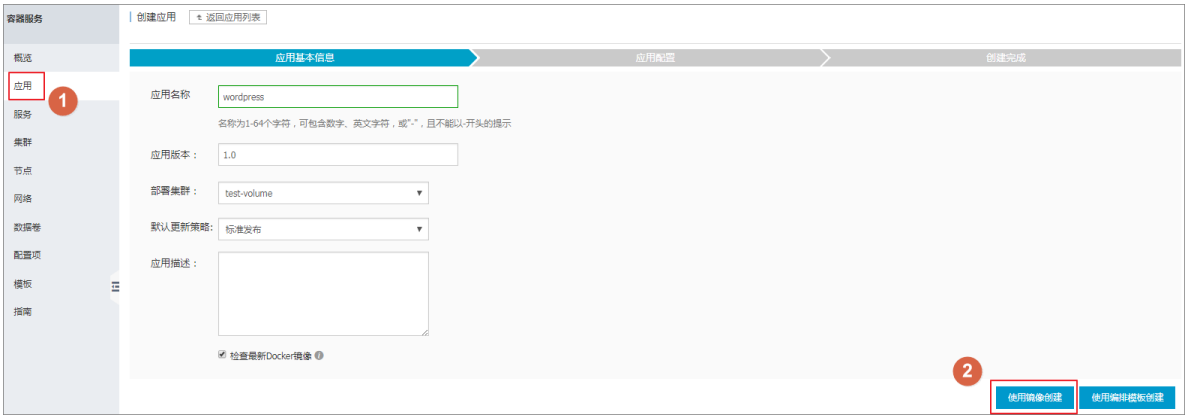
1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**，单击右上角的**创建应用**。

3. 设置应用的基本信息并单击**使用镜像创建**。

   - **应用名称**：所创建应用的名称。名称可以包含1~64个字符，包括数字，中文字符，英文字母和连字符（-）。
   - **应用版本**：所创建应用的版本。默认为1.0。
   - **部署集群**：所创建应用将要部署到的集群。
   - **默认更新策略**：应用更新的方式，您可以选择标准发布或蓝绿发布，参见概述。
   - **应用描述**：应用信息。可以不填写。该信息不能超过1,024个字符。该信息将显示在 **应用列表** 页面。
   - **检查最新Docker镜像**：选中该选项后，表示当镜像Tag不变的情况下，也会去仓库拉取最新的镜像。为了提高效率，容器服务会对镜像进行缓存。部署时，如果发现镜像Tag与本地缓存的一致，则会直接复用而不重新拉取。所以，如果您基于上层业务便利性等因素考虑，在做代码和镜像变更时没有同步修改Tag ，就会导致部署时还是使用本地缓存内旧版本镜像。而勾选该选项后，会忽略缓存，每次部署时重新拉取镜像，确保使用的始终是最新的镜像和代码。



4. 根据您的需要设置以下参数。

   a) 设置**镜像名称**和**镜像版本**。

   您可以直接填写目标Hub的镜像名称，填写的格式为域名/*namespace/imagename:tag*。容器服务会默认使用镜像的最新版本。如果您需要使用镜像的其它版本，单击 **选择镜像版本**，单击所需版本并单击 **确定**。

   b) 设置**容器数量**。

c) 设置应用的**网络模式**。

容器服务目前支持 **默认**和 **host** 网络模式。如果您不进行设置，容器服务会默认使用 **默认** 网络模式。

d) 设置**Restart**。

即容器是否是否异常重启。

e) 设置容器的启动命令（**Command**和**Entrypoint**）。

如果设置了会覆盖镜像的配置。

f) 设置容器的资源限制（**CPU 限制** 和 **内存限制**）。

有关容器资源限制的详细信息，参见 限制容器资源。

g) 设置**端口映射**、**简单路由配置**、**负载均衡路由配置**。

h) 设置容器的**数据卷**。

volumes_from：填写另外一个服务或者容器的名称，将服务或容器上的所有数据卷挂载到所创建的应用上。默认会赋予新创建应用对数据卷的读写权限。

i) 设置**环境变量**。

j) 容器的**labels**。

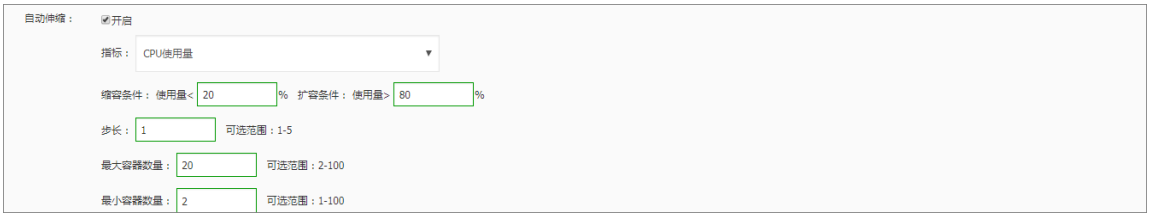
有关容器服务支持的标签，参见 标签概览。

k) 设置是否允许容器进行**平滑升级**。

有关平滑升级的相关信息，参见 *rolling_updates*。

l) 设置容器的**自动伸缩**规则。

图 104: 自动伸缩规则



5. 设置完毕后，单击**创建**。

12.4.6.2 使用编排模板创建应用

您可以通过容器服务云控制台使用编排模板创建应用。

使用限制

swarm集群只支持Compose V1/V2版本的编排模板；因此，在使用编排模板创建应用时，请选择Compose V1/V2版本的模板。如果您选择了Compose V3版本的模板，系统会报错。

说明:

在编排模板列表页面，Compose V3版本的模板标注有*composev3*字样。

图 105: 报错信息



操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**，单击右上角的**创建应用**。
3. 设置应用的基本信息并单击**使用编排模板创建**。

   - **应用名称**：所创建应用的名称。名称可以包含1~64个字符，包括数字，中文字符，英文字母和连字符（-）。
   - **应用版本**：所创建应用的版本。默认为1.0。
   - **部署集群**：所创建应用将要部署到的集群。
   - **默认更新策略**：应用更新的方式，您可以选择标准发布或蓝绿发布，参见发布策略概述。
   - **应用描述**：应用信息。可以不填写。该信息不能超过1,024个字符。该信息将显示在 **应用列表** 页面。
   - **检查最新Docker镜像**：选中该选项后，表示当镜像Tag不变的情况下，也会去仓库拉取最新的镜像。为了提高效率，容器服务会对镜像进行缓存。部署时，如果发现镜像Tag与本地缓存的一致，则会直接复用而不重新拉取。所以，如果您基于上层业务便利性等因素考虑，在做

代码和镜像变更时没有同步修改Tag ，就会导致部署时还是使用本地缓存内旧版本镜像。而勾选该选项后，会忽略缓存，每次部署时重新拉取镜像，确保使用的始终是最新的镜像和代码。

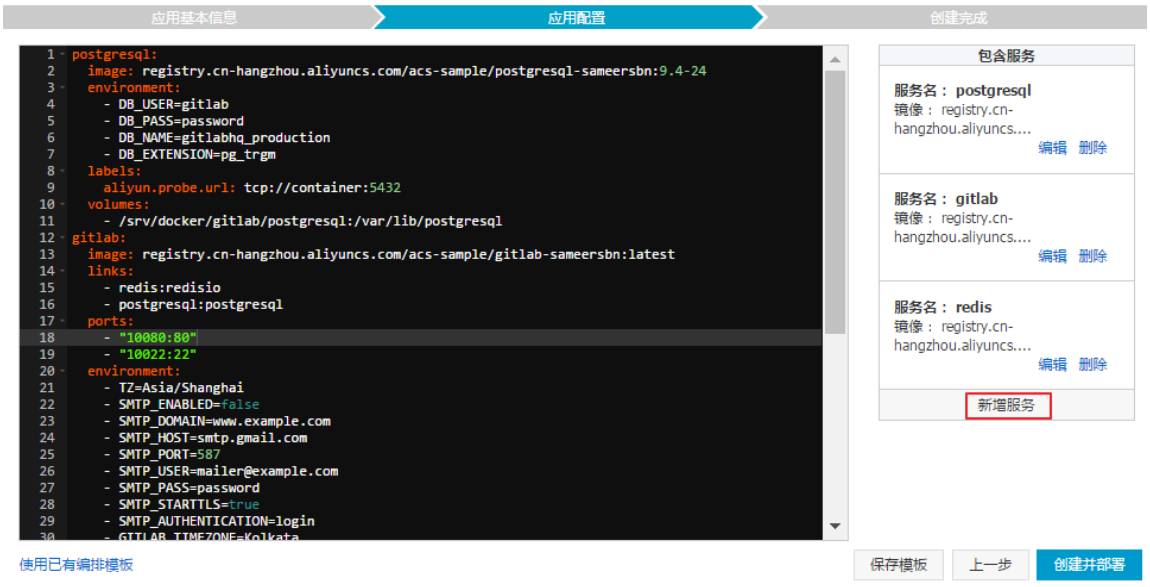
4. 编写编排模板。

- 您可以自己编写编排模板。
- 或者单击**使用已有编排模板**，选择一个模板并单击**选择**。

> **说明:** swarm集群只支持Compose V1/V2版本的编排模板。

您可以根据自己的需要编辑编排模板。您可以直接在编排模板中进行修改，或者在页面右侧选择需要修改的服务，单击**编辑**进行修改或者单击**删除**删除所选的服务

图 106: 编写编排模板



此外，您还可以单击服务列表下面的**新增服务**添加服务。填写您所需要的镜像的名称，进行参数设置并单击**确定**。



5. 设置完毕后，单击**创建并部署**。

12.4.6.3 限制容器资源

Docker容器的一大优势就是可以限制资源，包括CPU、内存、IO等。该设置无法通过容器服务云控制台进行操作，您需要在编排模板中进行设置。

限制CPU

一个CPU核等于100 CPU资源。如果机器配置是4核，则总共可用的CPU资源为400。在编排模板中，可以通过cpu_shares参数指定。cpu_shares: 50表示使用0.5个核。

限制内存

您可以使用mem_limit参数限制内存，单位为byte，最小内存为4MB。如果设置了内存限制，当容器申请的内存超过限制时，容器会因为OOM而停止运行。

编排示例：

```
n1:
  expose:
    - 443/tcp
    - 80/tcp
  image: 'nginx:latest'
  cpu_shares: 50 #0.5核
  mem_limit: 500000000 #500MB
  labels:
    aliyun.scale: '1'
  restart: always
  volumes:
    - /var/cache/nginx
```

资源调度

为了保证容器能获得足量的指定资源，比如上述例子里的0.5核CPU和500MB内存，容器服务会为容器预留资源。比如，一台4核的机器，最多会调度8个cpu_shares=50的容器。但是，如果创建容器时未指定cpu_shares和mem_limit，则默认不占资源。

限制其他资源

其他资源限制请参考*Docker Compose* 的说明。

12.4.6.4 指定节点调度

如果您需要将某个服务部署在指定的节点上，可以使用constraint关键字来实现这个功能。



说明: 部署约束只对新创建容器生效，对老容器变更配置时不起作用。

在下面的示例中，服务指定部署在node1上。

```
web:
    image: 'nginx:latest'
    restart: always
    environment:
        - 'constraint:aliyun.node_index==1'
    ports:
        - 80
    labels:
        aliyun.scale: 2
```

容器服务支持以下表达式：

表达式	说明
constraint:aliyun.node_index==1	指定部署到node1。
constraint:aliyun.node_index!=1	不部署到node1。
constraint:aliyun.node_index==(1|2|3)	指定部署到node1或者node2或者node3。

表达式	说明
constraint:aliyun.node_index!=(1|2|3)	部署到除node1、node2、node3的其他机器上。
affinity:image==~redis	尽量部署到有Redis镜像的机器上。
affinity:service!=~redis	尽量不部署到有Redis容器的机器上。

12.4.6.5 指定多节点调度

如果您希望一个应用在指定的某几个节点上部署，推荐您使用用户标签和*constraint*关键字来进行部署设置。

说明:

- 部署约束只对新创建容器生效，对老容器变更配置时不起作用。
- 使用用户标签部署应用后，如果您删除了用户标签，不会影响到已经部署的应用，但是会影响下次新的部署。请谨慎删除用户标签。

操作步骤

1. 为节点添加用户标签。
   a) 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

   b) 单击左侧导航中的**集群**。
   c) 选择所需集群并单击右侧的**管理**。
   d) 单击左侧导航栏中的**用户标签**。
   e) 勾选您要部署应用的节点并单击**添加标签**。

   图 107: 添加用户标签



   f) 输入您自定义的标签键和标签值并单击**确定**，为所选节点添加用户标签。



2. 创建应用，选择**使用编排模板创建**并在编排模板中配置*constraint*关键字。

   有关如何创建应用，参见创建应用。

```
environment:
  - constraint:group==1 #表示在所有带有 “group:1” 标签的节点上部署
```

12.4.6.6 删除用户标签

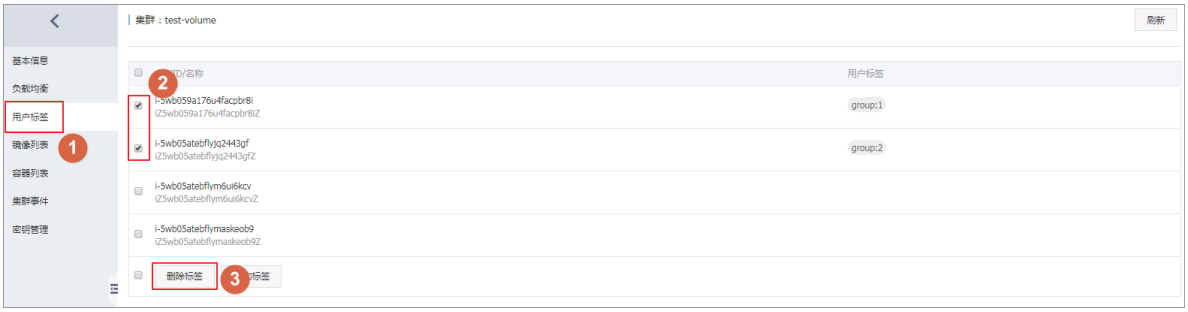
您可以删除已经不再使用的用户标签。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**集群**。
3. 选择所需集群并单击右侧的**管理**。
4. 单击左侧导航栏中的**用户标签**。
5. 勾选您要部署应用的节点并单击**删除标签**。



6. 在弹出的确认对话框中单击**确定**。



12.4.6.7 查看应用详情

您可以通过容器服务云控制台查看已经创建的应用的详细信息。

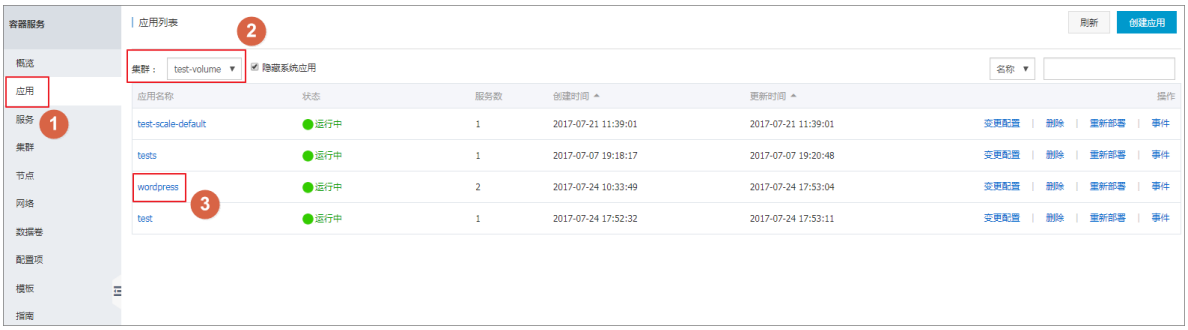
操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

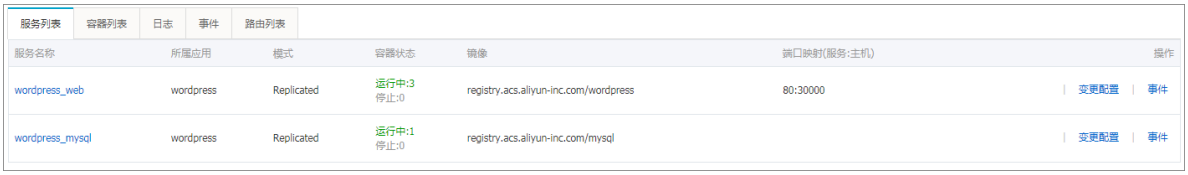
2. 单击左侧导航中的**应用**。
3. 选择所要查看的应用所在的集群。
4. 单击所要查看应用的名称。

图 108: 应用列表



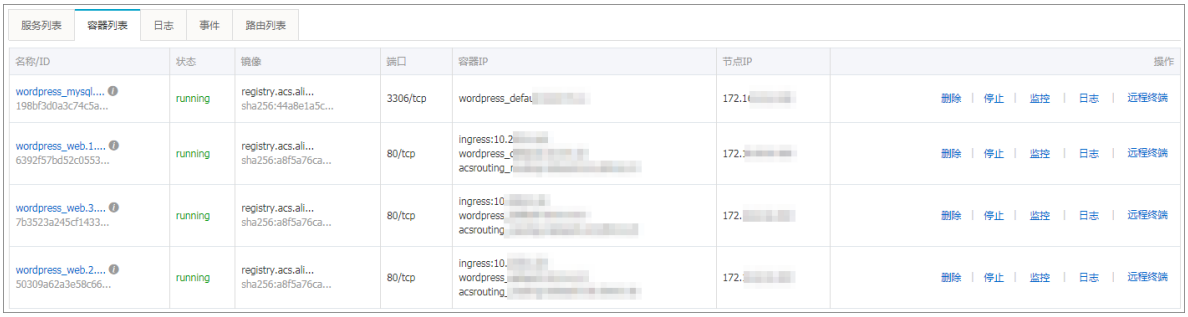
5. 单击**服务列表**查看该应用的服务列表。

图 109: 服务列表



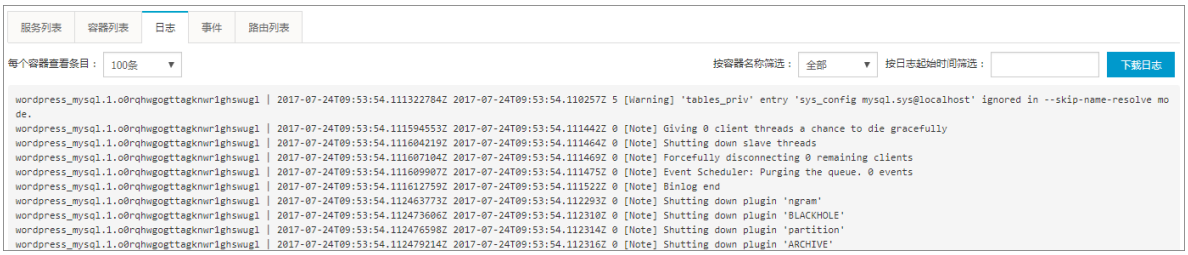
6. 单击**容器列表**查看该应用的容器列表。

图 110: 容器列表



7. 单击 **日志** 查看该应用的日志信息。

图 111: 日志



8. 单击 **事件** 查看该应用的事件信息。

图 112: 事件



9. 单击 **路由列表** 查看该应用的路由地址。

图 113: 路由列表



12.4.6.8 启动或停止应用

您可以通过容器服务云控制台启动或停止应用。

操作步骤

1. 登录容器服务云控制台。

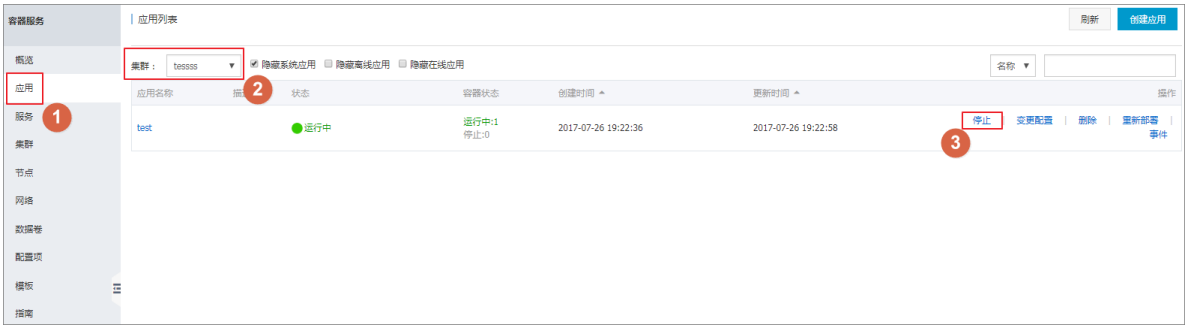
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**应用**。

3. 选择目标应用所在的集群。

4. 选择目标应用并单击**启动**或**停止**。

图 114: 停止应用



5. 在弹出的对话框中，单击**确定**。

12.4.6.9 变更应用配置

您可以通过容器服务云控制台变更应用的配置。

操作步骤

1. 登录容器服务云控制台。

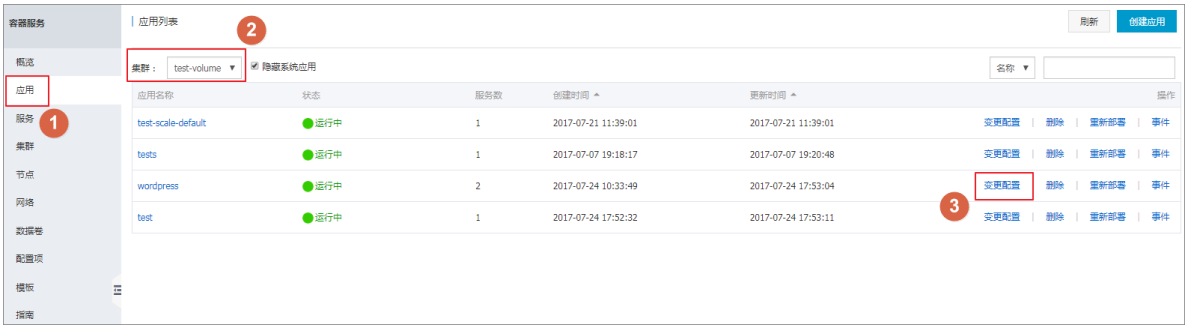
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**。

3. 选择所要查看的应用所在的集群。

4. 选择所需的应用并单击**变更配置**。

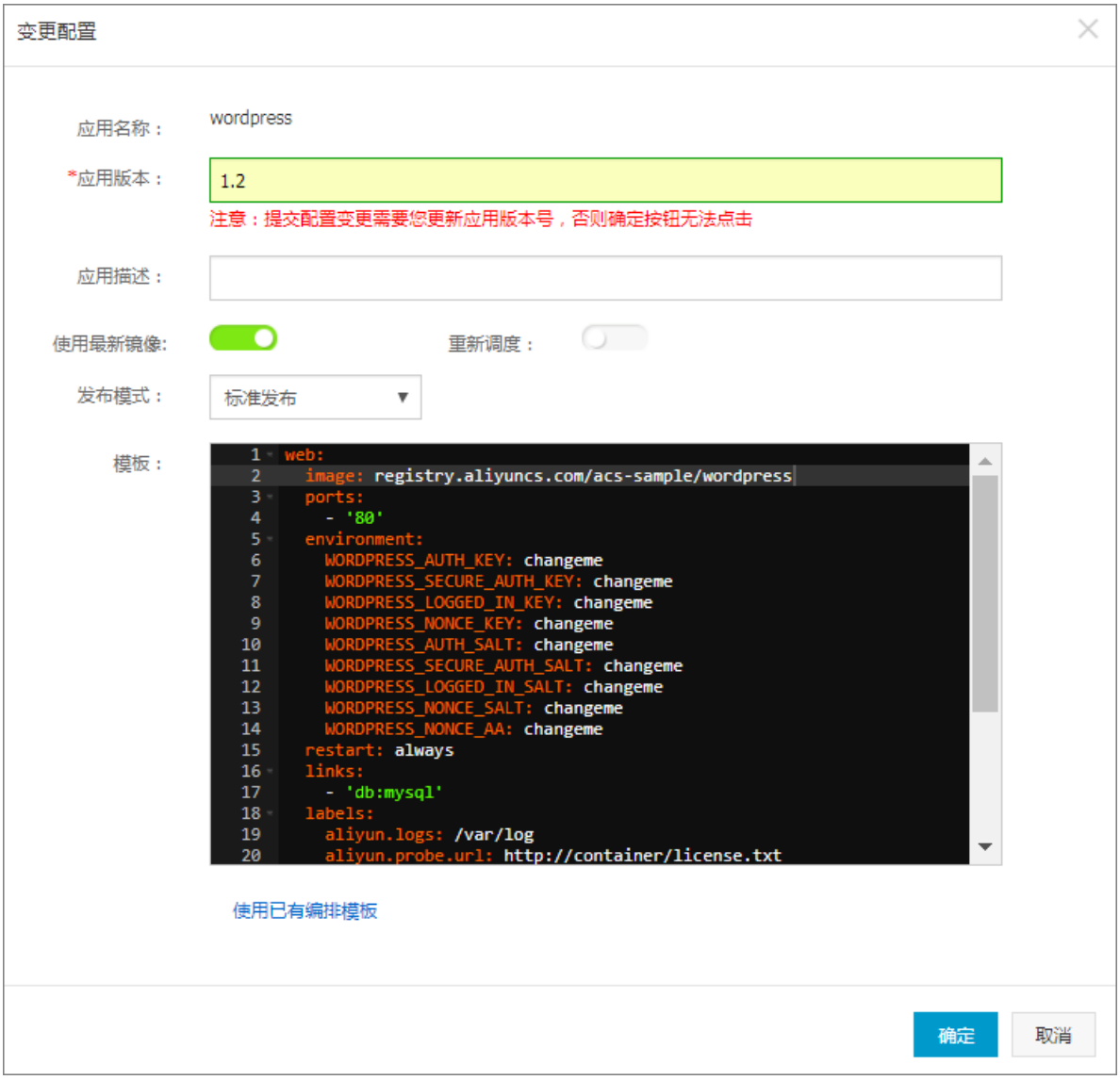
图 115: 应用列表



5. 在弹出的对话框中，修改配置。

说明: 您必须修改**应用版本**；否则，**确定**按钮不可用。

图 116: 变更配置



后续操作

如果您变更配置后，发现应用没有更新，可以尝试重新部署应用。有关重新部署应用的详细信息，参见 重新部署应用。

12.4.6.10 重新部署应用

应用部署之后您可以根据您的需求对应用进行重新部署。重新部署会重新拉取应用使用的镜像，因此如果您部署应用之后更新了镜像，重新部署会使用新的镜像进行应用部署。

背景信息

说明: 重新部署不会更新volume，宿主机上的老volume仍会继续使用。因此，如果您挂载了volume并在新镜像中对volume设置进行了修改，重新部署后新设置不会生效。

在以下情况下，您会用到重新部署功能：

- 部署应用之后，您更新了镜像的内容，需要按照新的镜像部署应用。
- 您停止或删除了某些容器，希望可以启动或重新创建这些容器。重新部署时，容器服务会重新启动已经停止的容器并重新创建已经删除的容器。

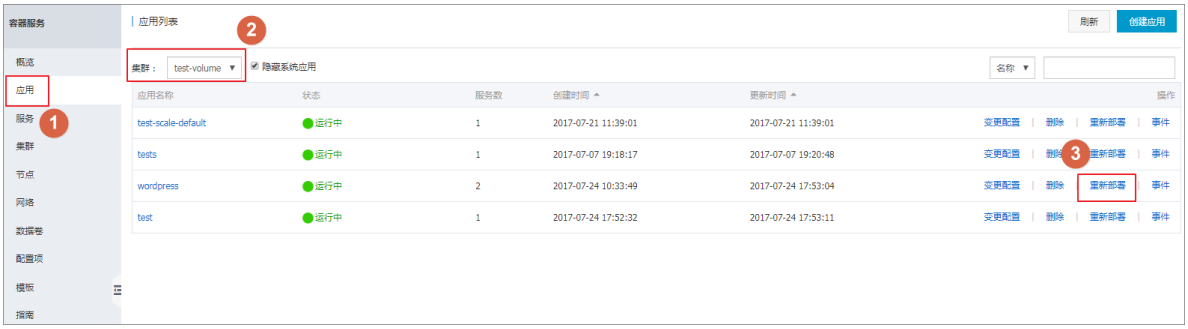
操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**。
3. 选择所要查看的应用所在的集群。
4. 选择要进行重新部署的应用，单击右侧的**重新部署**。

   图 117: 应用列表



5. 在弹出的对话框中，单击**确定**。

查看重新部署是否成功

您可以通过查看镜像的sha256确定重新部署后容器的镜像是否为最新镜像，从而确定重新部署是否成功。

操作步骤

1. 登录容器服务云控制台。

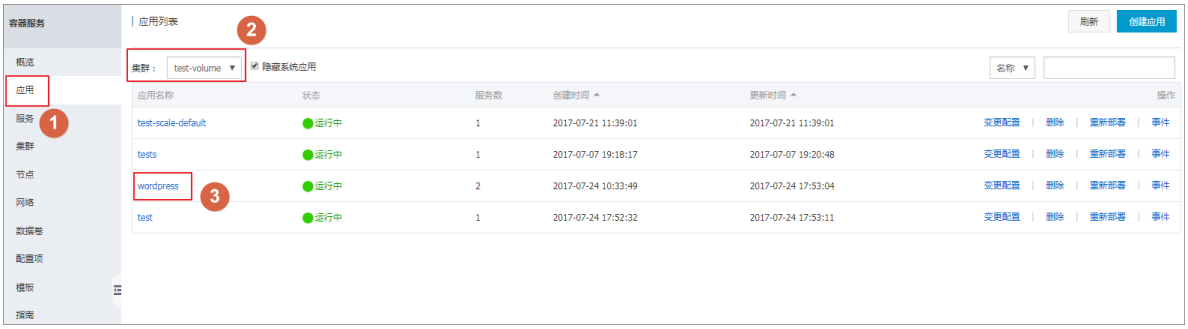
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**。

3. 选择所要查看的应用所在的集群。

4. 单击应用的名称。

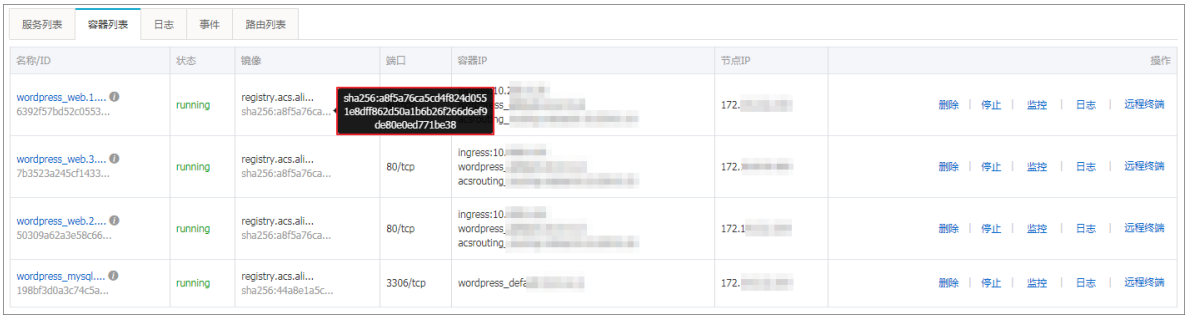
图 118: 应用列表



5. 单击**容器列表**并查看镜像的sha256。

   如果容器的镜像为新镜像，则重新部署成功。

图 119: 查看sha256



12.4.6.11 删除应用

您可以通过容器服务云控制台删除不再使用的应用。

操作步骤

1. 登录容器服务云控制台。

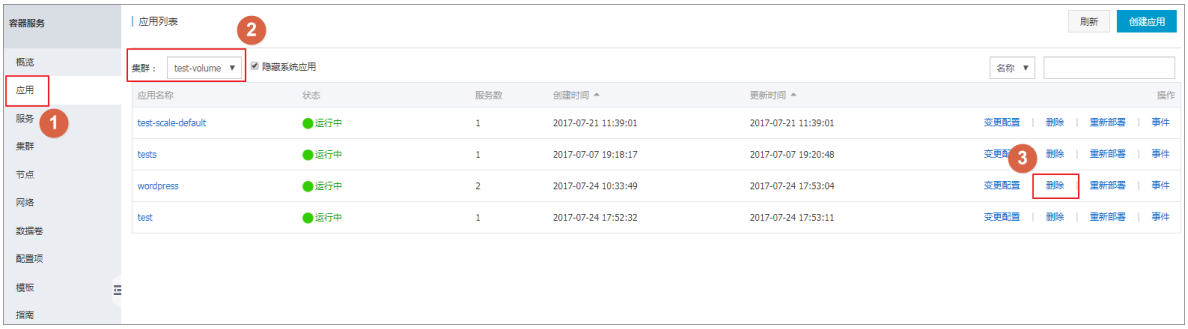
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**。

3. 选择所要查看的应用所在的集群。

4. 选择所要删除的应用并单击**删除**。

图 120: 应用列表



5. 在弹出的对话框中，单击**确定**。

如果您勾选**同时移除数据卷**（volume），系统将删除与应用相关的所有数据卷。但是，已命名的数据卷无法删除。

图 121: 确认删除



12.4.6.12 运行离线任务

容器服务抽象出离线计算的基本模型，推出了基于 Docker 容器的离线计算功能。

背景信息

其核心功能包括：

- 作业编排
- 作业调度与生命周期管理
- 存储与日志等功能的集成

基本概念

下表中列出了离线应用与在线应用的概念对比。

概念	离线应用	在线应用
容器	任务执行单元	服务的执行单元
运行历史	任务出错重试的执行历史	无
服务（任务）	一个特定的功能，可以分割成若干个容器来执行	一组功能相同的容器
应用（作业）	若干个任务的组合	若干个服务的组合

简言之，一个离线作业包含若干个任务，每个任务可以由若干个容器来执行，每个容器可以有多个运行历史；而一个在线应用包含若干个服务，每个服务可以有若干个容器同时服务。

基于Docker Compose的作业编排

和在线应用一样，您可以使用Docker Compose来描述和编排作业。Docker Compose支持Docker的绝大部分功能，比如：

- CPU、内存等资源限制
- 数据卷（Volume）
- 环境变量与标签
- 网络模型、端口暴露

除此之外，阿里云容器服务还扩展了以下功能：

- 容器数量：每个任务分成多少个容器
- 重试次数：每个容器重试多少次
- 移除容器：容器运行完后是否删除，可选策略包括remove_finished（删除完成的容器）、remove_failed（删除失败的容器）、remove_all（删除全部容器）、remove_none（不删除）。
- DAG 模型的任务依赖：同一个作业的任务之间可以有依赖关系，被依赖的任务会先执行。

离线作业的Docker Compose示例：

```
version: "2"
labels:
  aliyun.project_type: "batch"
services:
  s1:
```

```
  image: registry.aliyuncs.com/jimmycmh/testret:latest
  restart: no
  cpu_shares:  10
  mem_limit: 100000000
  labels:
    aliyun.scale: "10"
    aliyun.retry_count: "20"
    aliyun.remove_containers: "remove-all"
s2:
  image: registry.aliyuncs.com/jimmycmh/testret:latest
  cpu_shares:  50
  mem_limit: 100000000
  labels:
    aliyun.scale: "4"
    aliyun.retry_count: "20"
    aliyun.remove_containers: "remove-finished"
    aliyun.depends: "s1"
```

说明:

- 该功能只支持Docker Compose 2.0。
- 您需要在作业级别添加标签aliyun.project_type: "batch"。如果您未添加该标签或标签值不为batch，则认为该应用为在线应用。
- 无论您将restart设置为什么值，都会被修改为no。
- 您可以用aliyun.depends标签指定依赖关系。可以依赖多个任务，用逗号（,）分隔。
- aliyun.retry_count的默认值为3。
- aliyun.remove_containers的默认值为remove_finished。

作业生命周期管理

容器状态由容器的运行及退出状态决定；任务状态由该任务中所有容器的状态决定；作业状态由该作业的所有任务决定。

容器状态

- 运行中（Running）：容器在运行。
- 完成（Finished）：容器退出且ExitCode==0。
- 失败（Failed）：有容器失败次数超过给定值。

任务状态

- 运行中（Running）：有容器在运行。
- 完成（Finished）：所有容器都完成了。
- 失败（Failed）：有容器失败次数超过给定值。

作业状态

- 运行中（Running）：有任务在运行。
- 完成（Finished）：所有任务都完成了。
- 失败（Failed）：有任务失败了。

上述状态都可以通过API 取，方便您自动化运维。

共享存储

容器之间、任务之间会有数据共享和交换，共享存储可以解决这一问题。比如在Hadoop上跑MR作业，是通过HDFS来交换数据的。在容器服务中，可以使用三类共享存储，其特性及应用场景对比如下所示。

存储	优点	缺点	适用范围
OSSFS数据卷	跨主机共享	读写、ls性能低；修改文件会导致文件重写	共享配置文件；附件上传
您自己集成成三方存储，如Portworx	将集群内的云盘虚拟成共享的大磁盘；性能高；snapshot、多拷贝	需要一定运维能力	需要共享数据的重IO应用，如文件服务器等；需要快速迁移的重IO应用，如数据库等

具体使用数据卷的帮助，可以参考以下文档：在阿里云容器服务上使用OSS数据卷#volume#。

12.4.6.13 定时任务

定时任务是常见需求。普遍的做法是，选择一台或几台机器，通过crontab实现定时任务。但是对于大规模或大量的定时任务，这种做法的缺点非常多。容器服务在离线任务的基础上，增加了定时任务的功能，通过简单的描述，解决了这些问题。

背景信息

说明: 关于离线任务的详细信息，参见运行离线任务。

通过crontab实现定时任务的缺点：

- 可靠性低，一台机器宕机，该机器上的定时任务就无法执行了。
- 没有调度功能，机器之间的负载可能不均衡。
- 没有重试机制，任务可能运行失败。
- 无法运行大规模分布式任务。

基于Docker Compose的定时任务描述

同离线任务一样，定时任务也是基于Docker Compose的，您只需要在应用模板里添加aliyun.schedule标签即可实现定时功能。如下面的例子所示。

```
version: "2"
labels:
  aliyun.project_type: "batch"
  aliyun.schedule: "0-59/30 * * * * *"
services:
  s1:
    image: registry.aliyuncs.com/jimmycmh/busybox:latest
    labels:
      aliyun.scale: "5"
      aliyun.retry_count: "3"
      aliyun.remove_containers: "remove-all"
    command: date
```

说明:

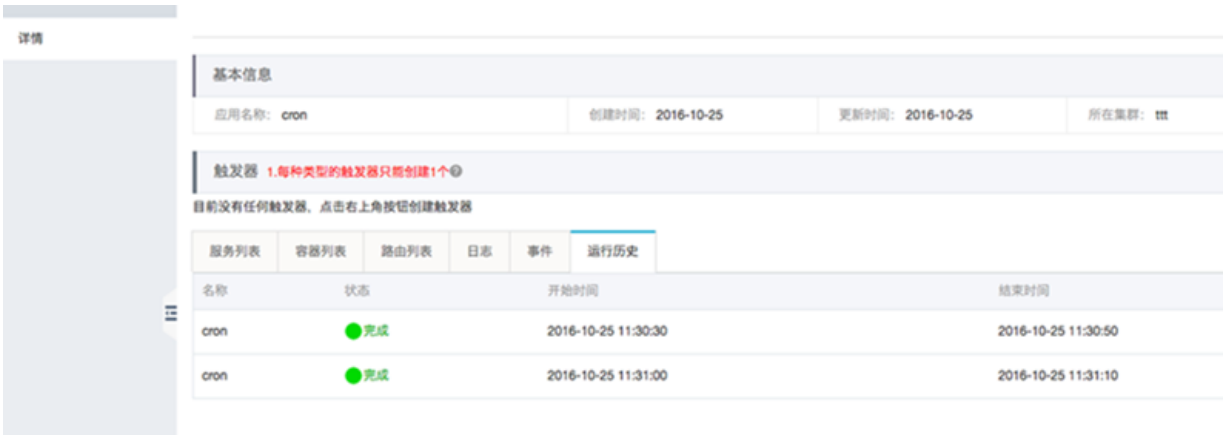
- aliyun.schedule: "0-59/30 * * * * *"表示每 30 秒执行一次该任务；schedule的格式和crontab完全相同（但要注意schedule的格式为秒 分 时 天 月 星期，比Linux上的crontab多了秒这一项），使用的时间为北京时间。
- 因为定时功能只适用于离线任务，所以只要您添加了aliyun.schedule标签，系统会自动添加aliyun.project_type: "batch"标签，因此上述例子中的aliyun.project_type: "batch"可以省略。
- 另外，离线任务中所有的功能，在定时任务中依然可用，比如scale、retry_count、remove_containers等。有关标签的具体含义，参见运行离线任务。

执行过程

定时任务被创建后，应用处于“等待”状态。当任务指定的时间到达时，任务会被启动运行，其后的状态变化和离线应用相同；下一个执行时间到达时，应用状态会重复这一过程。

同一个定时任务同一时刻只会有一个实例在执行，如果任务的执行时间大于其执行周期（比如上述任务的执行时间大于30秒），则下一次执行会进入执行队列；如果执行队列长度大于3，则会丢弃该次执行。

您可以在应用详情页面单击**运行历史**查看运行历史及结果。运行历史列表只保留最后10次的运行历史。



高可用性

定时任务控制器采用主-从备模式。主控制器故障时，控制功能将切换至备用控制器。

如果任务的执行时刻正好在主从切换期间，则会延迟至切换完成后执行。如果主从切换期间同一个任务有多次执行，切换完成后只会执行一次；因此，为了保证任务不丢失，请不要设计重复周期小于一分钟的任务。

12.4.6.14 系统默认应用列表

应用名称	中文名称	包含的服务	简介
acsrouting	路由服务	routing	提供7层协议的请求路由服务。组件包括负载均衡和一个HAProxy容器。域名配置正确后，即可将请求发送到指定的容器内。
acsmonitoring	监控服务	acs-monitoring-agent	与阿里云云监控进行集成，并与目前比较流行的第三方开源监控框架做集成，方便您查询监控信息并配置监控报警。
acsvolumedriv	数据卷服务	volumedriver	与阿里云的存储服务OSS做集成，方便您以volume卷（数据卷）的方式来使用共享式存储，告别有状态的容器运维。使用方法参见数据卷服务。

12.4.6.15 创建并调用触发器

触发器是容器服务提供的简单快捷地进行重新部署和资源伸缩的API。

由于标准的API需要保证安全性，因此需要严格的鉴权。但是对于需要与其他三方系统（例如Jenkins 或者其他持续集成的CI/CD系统）集成的场景来说，所需要的权限是有限的，可能仅仅需要消息通知。因此为了保证安全性与便捷性，带有部分鉴权策略，并且可以灵活调用的API被广泛应用于持续集成持续交付的场景中。

容器服务目前提供重新部署触发器和资源伸缩触发器。

- **重新部署触发器**

  您可以与自己的监控系统进行集成，当发现系统异常时进行重新部署；您也可以与容器Hub进行集成，当容器新镜像构建完成后，可以自动进行部署等。

- **资源伸缩触发器**

  您可以通过调用资源伸缩触发器来实现容器伸缩。

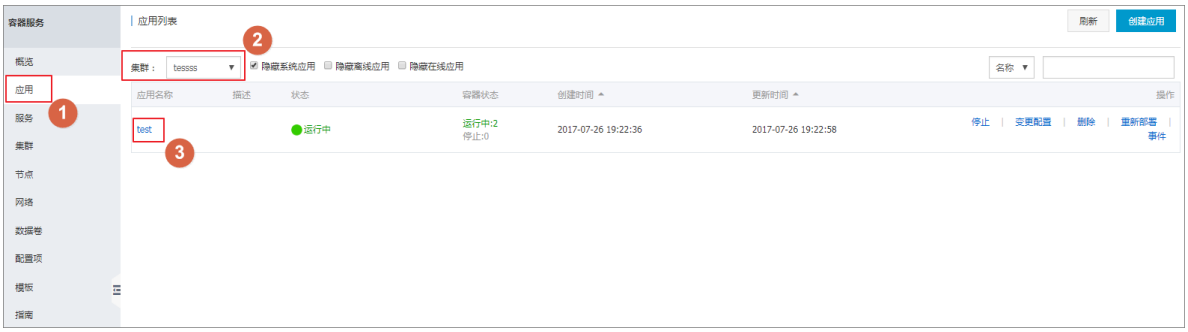
创建触发器

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**应用**。
3. 选择目标应用所在的集群。
4. 选择目标应用并单击应用的名称。

图 122: 应用列表



5. 在应用详情页面，单击**创建触发器**。
6. 在**触发器行为**下拉框中选择**重新部署**或者**资源伸缩**并单击**确定**。

   - **重新部署**

     当您对应用使用的镜像有写权限时，您可以勾选**关联到镜像更新**。勾选后，当容器新镜像构建完成后，可以使用最新镜像自动对应用进行重新部署。

图 123: 创建触发器



- **资源伸缩**

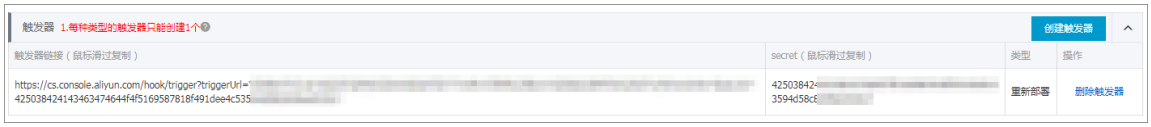
  在**服务**下拉框中选择需要设置资源伸缩触发器的服务。

图 124: 创建触发器



此时生成的触发器地址即为API的地址。

图 125: API地址



调用触发器

您可以通过三方集成系统进行触发，使用GET或者POST都可以进行触发，例如使用curl命令触发

调用重新部署触发器：

```
curl 'https://cs.console.aliyun.com/hook/trigger?triggerUrl=YzI4YTk5NzFkZWZkYzQ2MTJiOWZkNTM1MzY2ZDU1M2NifGNvbGxlY3RkLWJlbmNobWFya3xyZWRlcGxv
```

```
eXwxOGlxbjc1Z25uMmVzfA==&secret=44586c6b466352395143584c3970654ff5323d25
09d546fdc1b33054b0928da8'
```

调用资源伸缩触发器：

调用资源伸缩触发器时，需要在触发器URL中手动添加以下参数：

表 22: 资源伸缩触发器参数

参数名称	必填	语义	可选值
type	是	伸缩类型	缩容：scale_in；扩容：scale_out
step	是	伸缩数量	正整数，1-100

例如，调用下面的触发器会执行扩容五个容器的操作。

```
curl 'https://cs.console.aliyun.com/hook/trigger?triggerUrl=Y2lxZjI5YzhhYjIwMzRl
MjBiYjc2OGUzYTlmZDgyNDAyfHdvcmRwcmVzcy10ZXN0fHNjYWxpbmd8MTkz
ZmEyMXFwZXVwMXw=&secret=53374142724e4e4a626f664a313131556e62c671
6cd0d97d096900b3ad42a9ad&type=scale_out&step=5'
```

12.4.6.16 删除触发器

您可以通过容器服务云控制台删除不再使用的触发器。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**应用**。

3. 选择目标应用所在的集群。

4. 选择目标应用并单击应用的名称。

图 126: 应用列表



5. 选择要删除的触发器并单击右侧的**删除触发器**。
6. 在弹出的确认对话框里，单击**确定**。

12.4.7 配置项

应用分为代码和配置两个部分，当应用容器化之后，通常通过容器环境变量的方式传递配置，从而实现同一个镜像使用不同的配置部署多套应用。

使用限制

目前，在创建应用时关联配置文件的场景仅适用于通过编排模板创建应用。

应用场景

创建应用

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**配置项**，选择需要创建配置项的集群并单击**创建配置项**。

图 127: 创建配置项

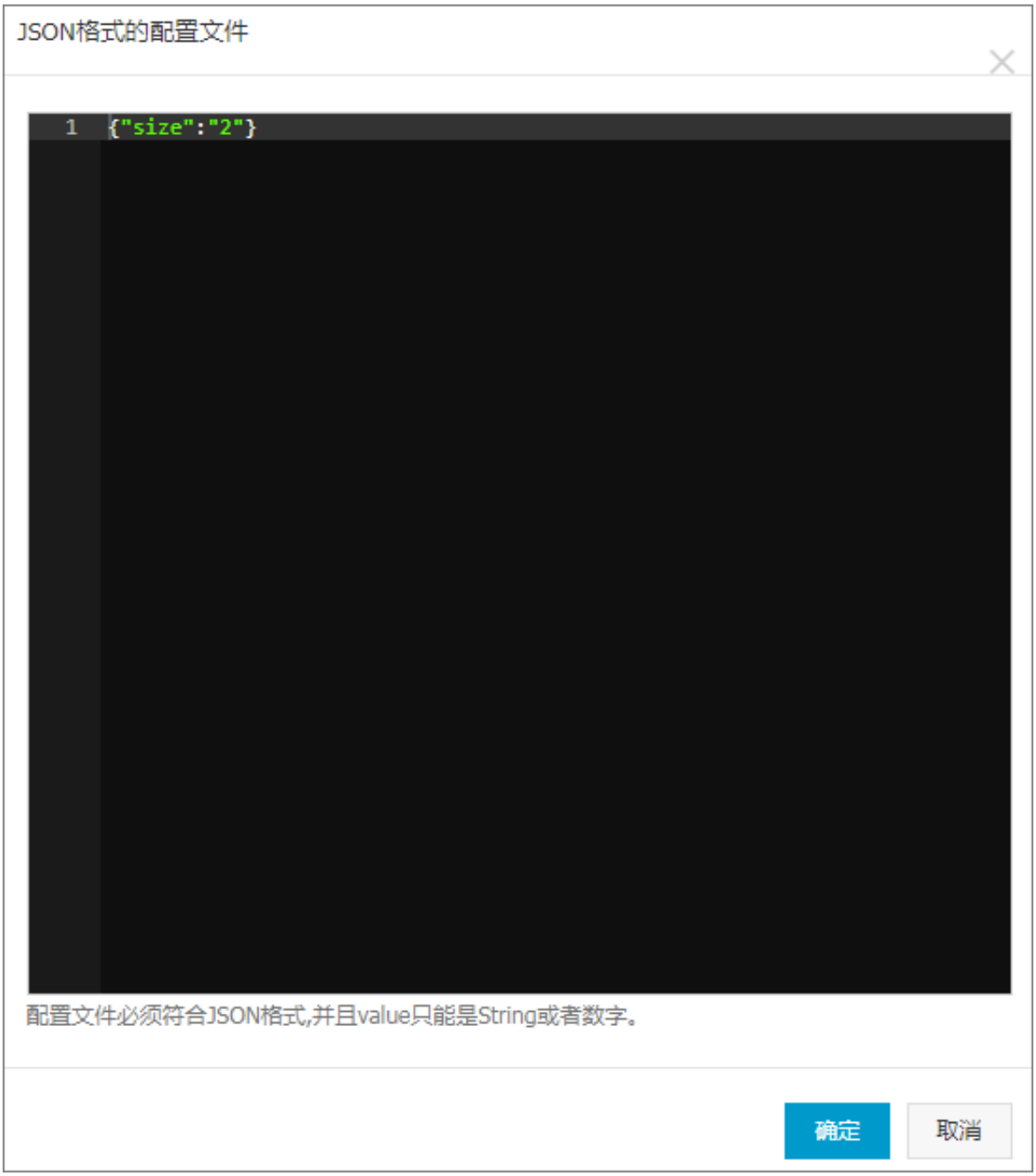


3. 填写配置文件的信息并单击**确定**。

   - **配置文件名**：可包含1~128 个字符。
   - **描述**：最多可包含128个字符。
   - **配置项**：您最多可以设置50个配置项。

     您也可以单击**编辑配置文件**在弹出的对话框里编写配置项并单击**确定**。

图 128: JSON格式文件



本示例中设置了*size*变量。

图 129: 配置文件



4. 单击左侧导航栏中的**应用**，选择与配置项处于相同地域下的集群，并单击**创建应用**。
5. 填写应用的基本信息并单击**使用编排模板创建**。
6. 填写如下所示的编排模板并单击**创建并部署**。

   其中，*size*即为动态的变量，这个变量会被配置项中的数值覆盖。

```
busybox:
 image: 'busybox'
 command: 'top -b'
 labels:
    aliyun.scale: $size
```

7. 在弹出的对话框中，选择要关联的配置文件，单击**替换变量**并单击**确定**。

图 130: 替换变量



更新应用

如果创建应用时关联了配置文件，您可以通过修改配置项并重新部署的方式更新应用。

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**配置项**，选择需要修改的配置项的地域并单击**修改**。

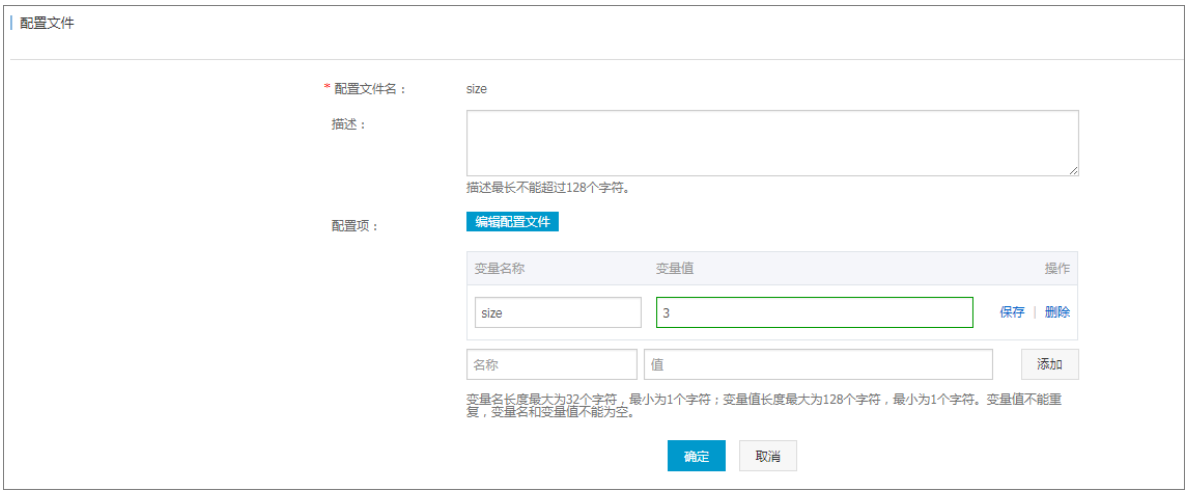
图 131: 修改配置项



3. 在弹出的确认对话框中，单击**确定**。

4. 选择要修改的变量并单击**编辑**（单击后变为**保存**）。修改变量值，单击**保存**并单击**确定**。

图 132: 修改配置项



5. 单击左侧导航栏中的**应用**，选择应用所在的集群，选择应用并单击重**新部署**。

触发更新

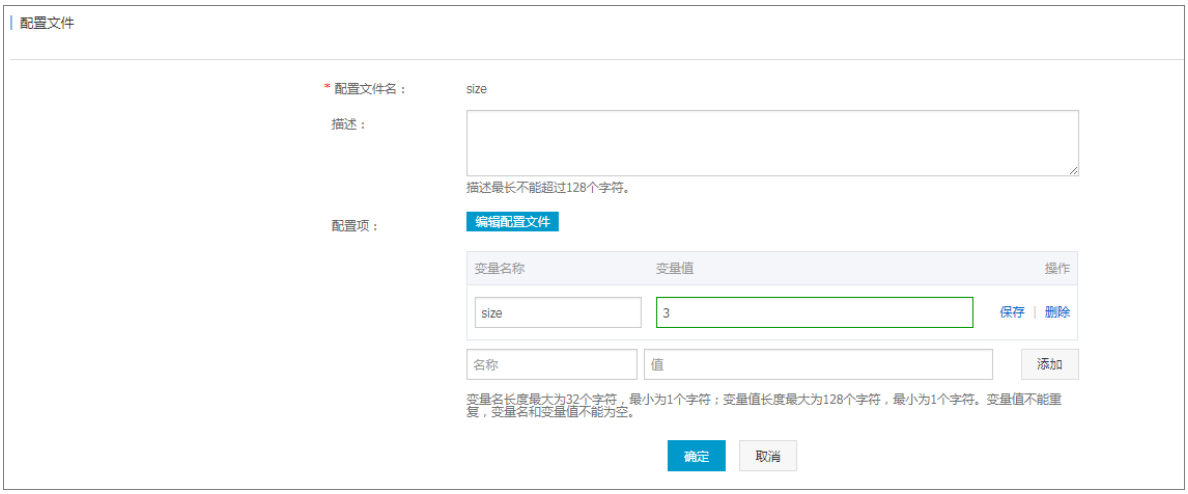
如果创建应用时关联了配置文件，您可以通过触发器触发的方式进行重新部署。

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**配置项**，选择需要修改的配置项的地域并单击**修改**。

图 133: 修改配置项



3. 创建重新部署触发器。

   有关如何创建触发器的详细信息，参见触发器。

图 134: 创建触发器



图 135: 触发器



4. 触发重新部署触发器。

```
curl "https://cs.console.aliyun.com/hook/trigger?triggerUrl=Y2FkZDgyMTUwNzdjYjQ3YThiZWEwMjAyODU1MjBlNDA5fGJ1c3lib3h8cmVkZXBsb3l8MTk0dHVzc29kNGJvcHw=&secret=517a45506f51317652744d774f58416a074b1b75299b83dd91b66150bac7ac15"
```

12.4.8 管理服务

您可以通过容器服务云控制台管理您的服务。

一个应用由一个或多个服务组成，您可以选择变更应用的配置来升级应用，也可以选择单独变更某个服务的配置来独立升级。

12.4.8.1 查看服务详情

您可以通过容器服务云控制台查看服务的详细信息。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**服务**。

3. 选择所要查看的服务所在的集群。

4. 单击所要查看的服务的名称。

图 136: 查看服务详情



5. 您可以在服务详情页面查看该服务的所有容器。

图 137: 容器列表



6. 单击**日志** 查看服务级别的日志信息。

7. 单击 **配置** 查看服务的配置信息。

图 138: 配置信息



8. 单击**事件** 查看服务的事件信息。

图 139: 事件



12.4.8.2 启动或停止服务

您可以通过容器服务云控制台启动或停止服务。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**服务**。
3. 选择所要查看的服务所在的集群。
4. 选择目标服务并单击**启动** 或 **停止**。

   图 140: 停止服务



5. 在弹出的对话框中，单击**确定**。

12.4.8.3 变更服务配置

您可以通过容器服务云控制台变更服务的配置。

操作步骤

1. 登录容器服务云控制台。
2. 单击左侧导航中的**服务**。
3. 选择所要查看的服务所在的集群。

4. 选择目标服务并单击**变更配置**。

图 141: 变更配置



5. 修改服务的配置并单击**更新**。

有关如何配置参数，参见。

12.4.8.4 重新调度服务

您可以重新平衡各个节点运行容器数量，将负载较重节点的容器迁移到新加入的节点和负载较轻的节点上，实现集群负载重新的平衡。

操作步骤

1. 登录容器服务云控制台。

有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**服务**。
3. 选择所要查看的服务所在的集群。
4. 选择目标服务并单击**重新调度**。

图 142: 重新调度服务



5. 在弹出的对话框中，选择**忽略本地数据卷**和**强制重新调度**并单击**确定**。

- **忽略本地数据卷**：对于有本地数据卷的容器，重新调度可能会将容器迁移到其他机器造成数据丢失。如果您需要忽略本地数据卷，请选择此参数；否则，对于有本地数据卷的容器将不进行重新调度。
- **强制重新调度**：目前为了保证线上服务的稳定性，默认只有在机器内存使用率超过 60%，CPU 使用率超过 40% 时才会进行重新调度。如果不想受限于这个限制，请选择此参数，容器服务将忽略使用率限制强制重新调度。

> **说明:** 已使用内存和 CPU 数值以容器配置为准，因此不一定是机器的实际使用情况。

图 143: 确认重新调度



12.4.8.5 重启服务

您可以通过容器服务云控制台重启服务。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。
2. 单击左侧导航中的**服务**。
3. 选择所要查看的服务所在的集群。
4. 选择所要删除的服务并单击**重启**。

图 144: 重启服务



5. 在弹出的对话框中，单击**确定**。

12.4.8.6 删除服务

您可以通过容器服务云控制台删除不再使用的服务。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**服务**。
3. 选择所要查看的服务所在的集群。
4. 选择所要删除的服务并单击**删除**。

图 145: 删除服务



5. 在弹出的对话框中，单击**确定**。

12.4.9 网络管理

12.4.9.1 跨主机互联的容器网络

容器服务为容器创建全局的网络，集群中的容器都可以通过容器的eth0的网络接口访问其他容器。

例如：分别在两台机器上创建容器，并打印出它们的IP地址。

```
cross-host-network-test1:
        image: busybox
        command: sh -c 'ifconfig eth0; sleep 100000'
        tty: true
        environment:
            - 'constraint:aliyun.node_index==(1)'
    cross-host-network-test2:
        image: busybox
        command: sh -c 'ifconfig eth0; sleep 100000'
        tty: true
        environment:
            - 'constraint:aliyun.node_index==(2)'
```

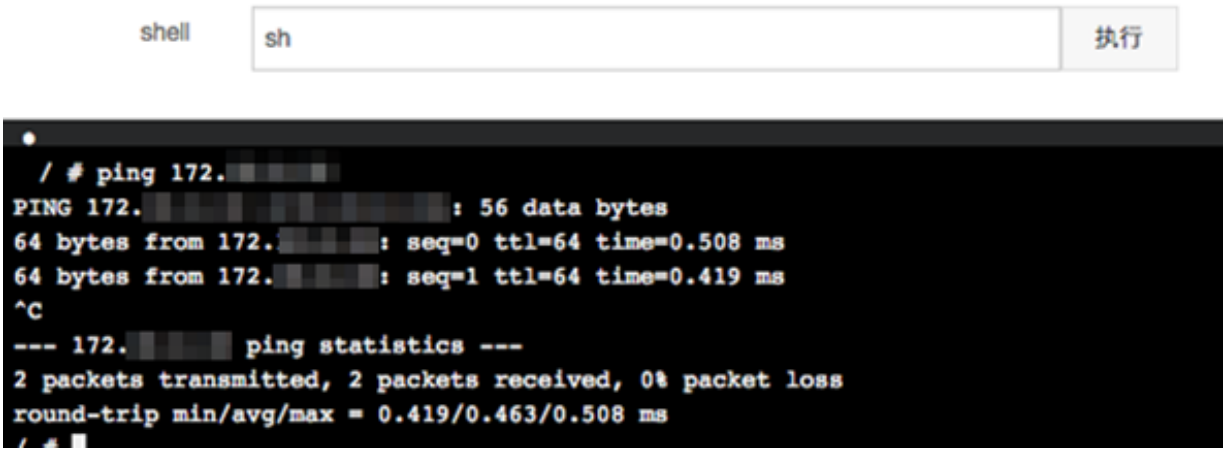
可以看到这两个服务的容器分布在不同的节点上，如下图所示。

图 146: 查看容器列表



通过容器服务云控制台或者通过容器 2 输出的 ifconfig eth0 日志，您可以看到容器 2 的 IP 地址为 172.19.0.10。您可以通过连接远程终端在容器 1 内访问容器 2 的 IP 地址。

图 147: 访问跨主机的容器



12.4.10 管理数据卷

Docker的特性决定了容器本身是非持久化的，容器被删除后其中的数据也会被一并删除。虽然Docker提供的数据卷（Volume）可以通过挂载宿主机上的目录来实现持久存储，但在集群环境中，宿主机上的数据卷有很大的局限性。

- 容器在机器间迁移时，数据无法迁移
- 不同机器之间不能共享数据卷

为了解决这些问题，阿里云容器服务提供第三方数据卷，将各种云存储包装成数据卷，可以直接挂载在容器上，并在容器重启、迁移时自动重新挂载。目前支持OSSFS和NAS存储。

12.4.10.1 开通数据卷功能

前提条件

您的集群里必须部署了acsvolumedriver应用，才可以开通数据卷功能。

操作步骤

1. 登录容器服务云控制台。

   关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**数据卷**，开通数据卷功能。

12.4.10.2 创建OSSFS数据卷

OSSFS是阿里云官方提供的基于FUSE的文件系统。OSSFS数据卷可以将OSS的Bucket包装成数据卷。由于数据需要经过网络同步到云端，OSSFS在性能和功能上与本地文件系统有差距。请不要把数据库等重IO应用、日志等需要不断改写文件的应用运行在OSSFS上。

背景信息

OSSFS和本地文件系统具体差异如下所示：

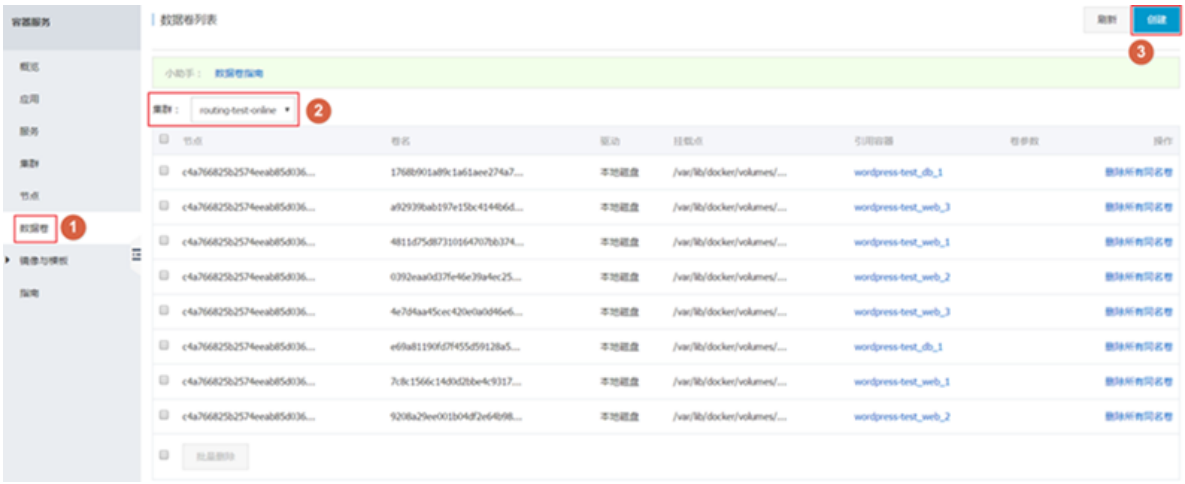
- 随机或者追加写文件会导致整个文件的重写。
- 因为需要远程访问OSS服务器，元数据操作（例如list directory）性能较差。
- 文件/文件夹的重命名操作不是原子的。
- 多个客户端挂载同一个OSS Bucket时，需要您自行协调各个客户端的行为。例如，避免多个客户端写同一个文件等等。
- 不支持硬链接（hard link）。

操作步骤

1. 登录容器服务云控制台。

   关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**数据卷**。
3. 选择需要创建数据卷的集群并单击页面右上角的**创建**。



4. 在弹出的对话框中，设置数据卷参数并单击**创建**。容器服务会在集群的所有节点上创建名称相同的数据卷。

   - **数据卷名**：数据卷的ID。数据卷名在集群内必须唯一。
   - **AccessKey ID**、**AccessKey Secret**：访问OSS所需的AccessKey。您可以从云平台的 **云资源中心** > **云产品管理** >**对象存储**获取账号的Access Key。
   - **访问域名**：如果Bucket和ECS实例位于同一个地域（Region），选择 **内网域名**；否则，选择**外网域名**。
   - **文件缓存**：如果需要在不同机器间同步同一个文件的修改（比如在机器A中修改文件，在机器B中读取修改后的内容），请关闭文件缓存。

说明: 关闭文件缓存将导致ls文件夹变得很缓慢，尤其是同一个文件夹下文件比较多时。因此，没有上述需求时，请打开文件缓存，提高ls的速度。

12.4.10.3 创建NAS数据卷

您可以通过容器服务云控制台将您自己的NAS存储创建为数据卷。

操作步骤

1. 登录容器服务云控制台。

   关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**数据卷**。
3. 选择需要创建数据卷的集群并单击页面右上角的**创建**。
4. 在弹出的对话框中，设置数据卷参数并单击**创建**。容器服务会在集群的所有节点上创建名称相同的数据卷。

   - **数据卷名**：数据卷的名称。数据卷名称在集群内必须唯一。
   - **文件系统ID**：NAS文件系统的ID。
   - **挂载点域名**：集群在NAS文件系统中挂载点的挂载地址。
   - **子目录**：NAS路径下的子目录，以 / 开头，设定后数据卷将挂载到指定的子目录。
     - 如果NAS根目录下没有此子目录，会默认创建后再挂载。
     - 您可以不填此项，默认挂载到NAS根目录。
   - **权限**：设置挂载目录的访问权限，例如：755、644、777等。
     - 只有挂载到NAS子目录时才能设置权限，挂载到根目录时不能设置。
     - 您可以不填此项，默认权限为NAS文件原来的权限。

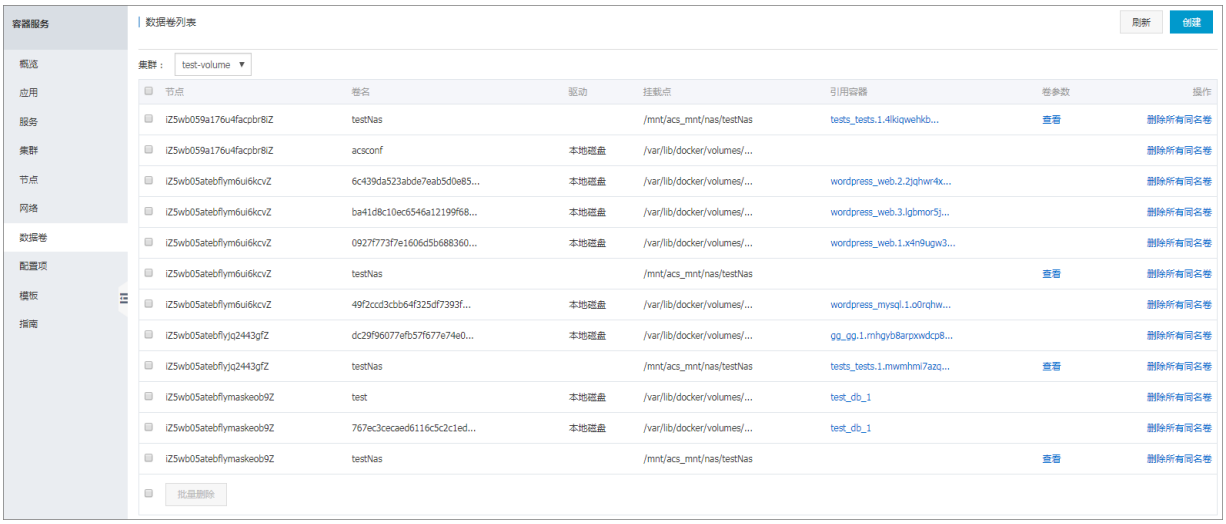
12.4.10.4 查看数据卷

登录容器服务云控制台并单击左侧导航栏中的**数据卷**。

关如何登录云控制台，参见登录云控制台。

数据卷列表页面列出了当前集群中所有的数据卷，包括本地数据卷和第三方数据卷。

图 148: 数据卷列表



对于本地数据卷，数据卷名称的格式为*node_name/volume_name*。

对于第三方数据卷，单击**查看**可以查看数据卷的参数。

创建第三方数据卷时，容器服务会在集群的所有节点上以同一数据卷名称创建该数据卷，便于容器在不同节点间迁移。

您可以单击**删除所有同名卷**删除所有使用该名称的数据卷。

12.4.10.5 使用第三方数据卷

第三方数据卷使用方法跟本地数据卷相同。

背景信息

creat the third volume

您可以在创建应用时设置数据卷的相关信息，或者通过变更已有应用的配置来添加数据卷的设置。

通过镜像创建应用

1. 登录容器服务云控制台。
2. 单击左侧导航栏中的**应用**。
3. 单击页面右上角的**创建应用**。
4. 单击页面右上角的**创建应用**。
5. 单击**使用镜像创建** 。
6. 单击**数据卷** 中的加号图标。
7. 在**主机路径或数据卷名**处填写数据卷名称。

说明: 有关如何使用编排模板创建应用的详细信息，请参见 创建应用。



通过编排文件创建应用

1. 登录容器服务云控制台。
2. 单击左侧导航栏中的 **应用**。
3. 填写应用的基本信息。
4. 单击 **使用编排模板创建** 。
5. 单击 **使用已有编排模板** 。
6. 选择一个编排模板并单击 **选择**。
7. 在 volumes 一节中，第一个冒号前填写数据卷名称。

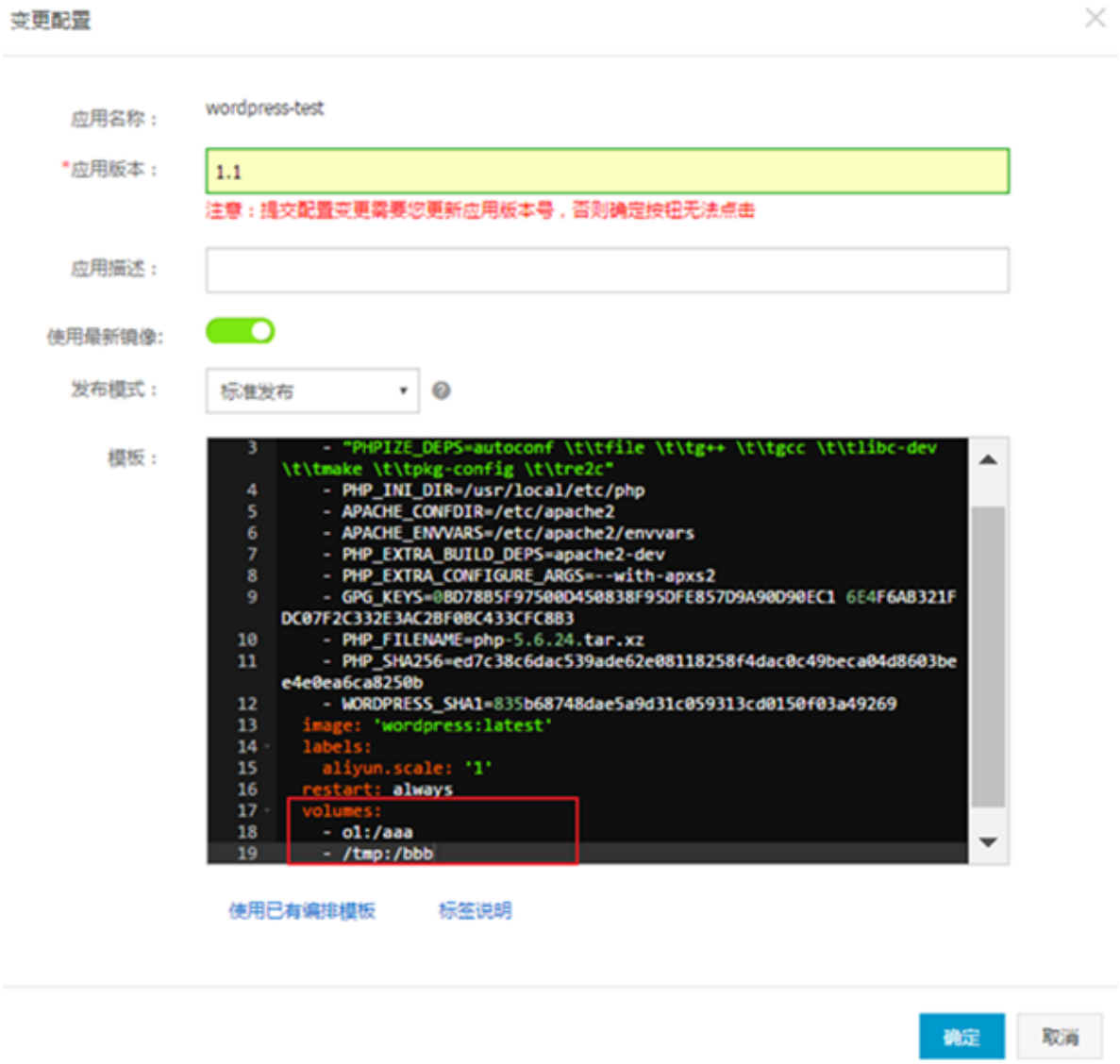
> **说明:** 有关如何使用编排模板创建应用的详细信息，请参见 创建应用。

```
volumes:
   - o1:/aaa
   - /tmp:/bbb
```

变更已有的应用配置

1. 登录容器服务云控制台。
2. 单击左侧导航栏中的 **应用**。
3. 选择需要更新的应用。
4. 单击右侧的 **变更配置** 。
5. 在模板的 volumes 一节中，第一个冒号前填写数据卷名称。

> **说明:** 有关变更配置的详细信息，参见 变更配置。



12.4.10.6 常见问题

问题现象

如果用数据卷名：镜像中已有目录的方式使用第三方数据卷（如 *o1:/data*，而镜像中有 */data* 目录），启动容器会失败，系统会返回类似于 **chown/mnt/acs_mnt/ossfs/XXXX: input/output error** 的错误。

错误原因

产生这个错误的原因是，对于命名数据卷，Docker 会把镜像中已有的文件复制到数据卷中，并用 chown 设置相应的用户权限，而 Linux 禁止对挂载点使用 chown。

解决方法

您可以通过以下两种方法之一解决该问题：

- 升级 Docker 到 1.11 或以上版本，升级 Agent 到最新版本并在编排模板中指定 `nocopy` 选项。Docker 会跳过复制数据的过程，因此不会产生 chown 错误。

```
volumes:
    - o1:/data:nocopy
    - /tmp:/bbb
```

- 如果必须复制数据，您可以不使用数据卷名称，而使用挂载点路径进行设置，比如用 */mnt/acs_mnt/ossfs/XXXX:/data*。但这种方式绕开了 volume driver，在机器重启时，无法保证在 OSSFS 挂载成功之后再启动容器，可能会导致容器挂载了一个本地数据卷。为了避免这种情况，您需要同时使用两个数据卷，其中一个数据卷使用数据卷名称进行设置，另外一个数据卷使用挂载点路径进行设置。使用数据卷名称进行设置的数据卷只起到和 volume driver 同步的功能，并不用于存储。

```
volumes:
    - o1:/nouse
    - /mnt/acs_mnt/ossfs/XXXX:/data
    - /tmp:/bbb
```

12.4.11 管理日志

您可以通过容器服务云控制台查看应用日志，服务日志和容器日志。

12.4.11.1 查看集群日志

您可以查看集群的日志信息。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。

3. 选择要查看的的集群并单击**查看日志**。

   图 149: 选择集群



查看该集群的日志信息。

图 150: 集群日志



12.4.11.2 查看应用日志

您可以查看应用的日志信息。

操作步骤

1. 登录容器服务云控制台。

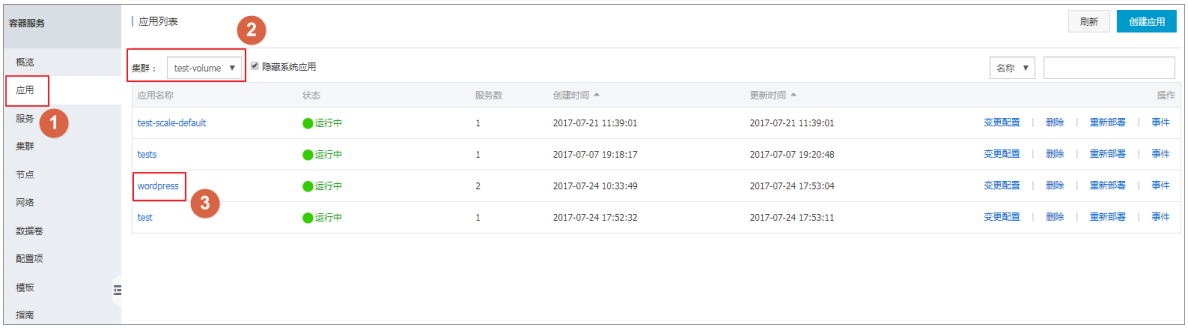
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**应用**。

3. 选择要查看的应用所在的集群。

4. 单击要查看的应用的名称。

图 151: 应用列表



5. 单击**日志**查看该应用的日志信息。

您可以选择显示的日志条目，也可以选择下载所有日志到本地。

图 152: 应用日志



12.4.11.3 查看服务日志

您可以查看服务的日志信息。

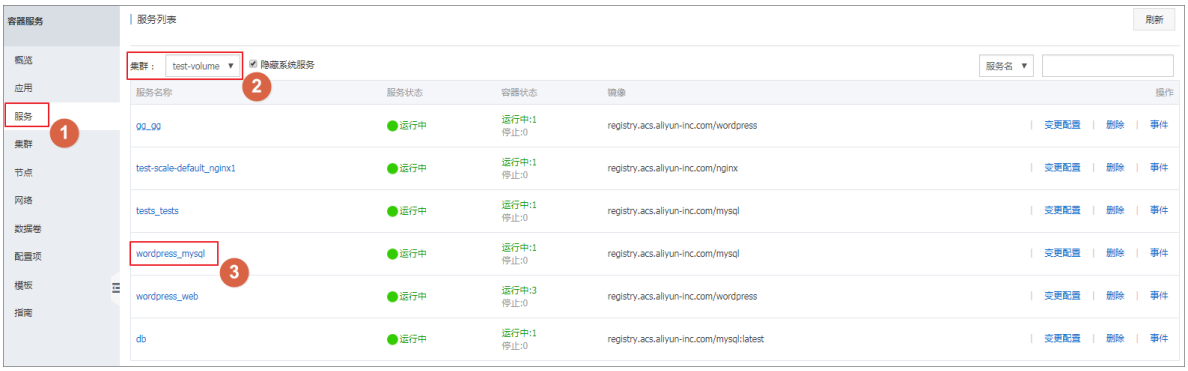
操作步骤

1. 登录容器服务云控制台。

有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**服务**。
3. 选择要查看的服务所在的集群。
4. 单击要查看的服务的名称。

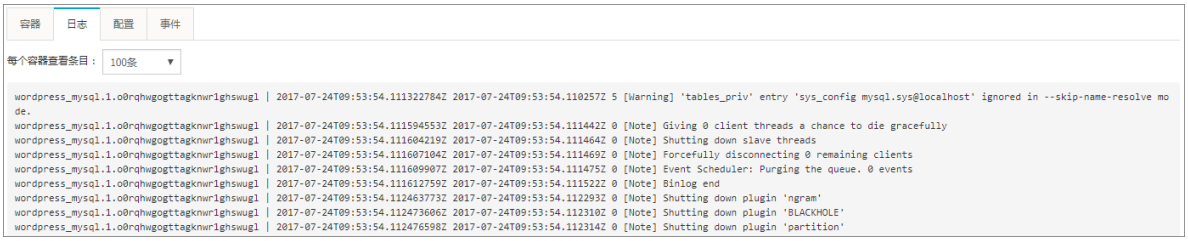
图 153: 查看服务详情



5. 单击**日志**查看该服务的日志信息。

   您可以选择显示的日志条目，也可以选择下载所有日志到本地。

图 154: 服务日志



12.4.11.4 查看容器日志

您可以查看容器的日志信息。

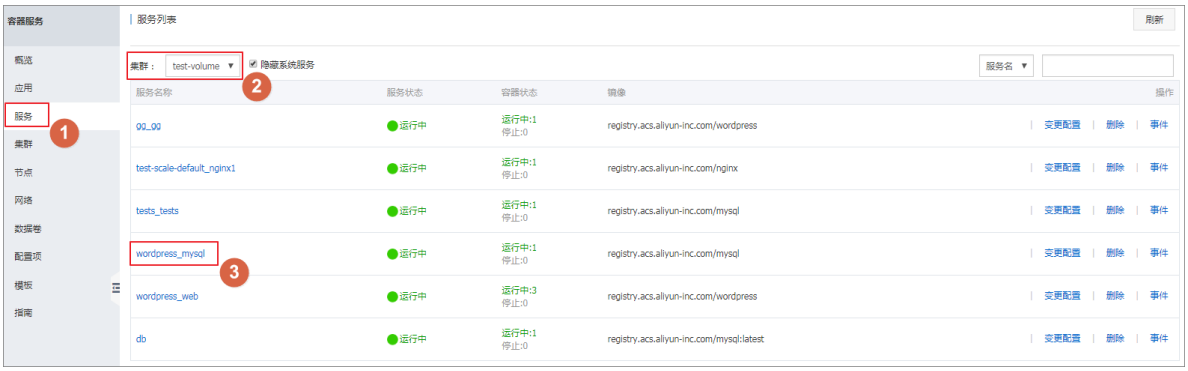
操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

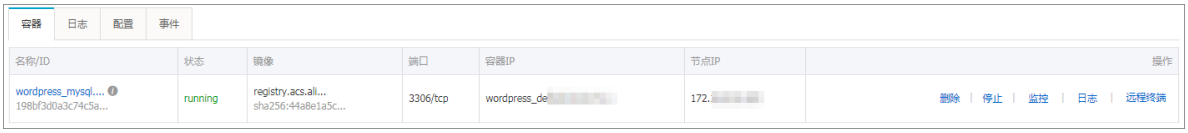
2. 单击左侧导航栏中的 **服务**。
3. 选择要查看的服务所在的集群。
4. 单击要查看的服务的名称。

图 155: 查看服务详情



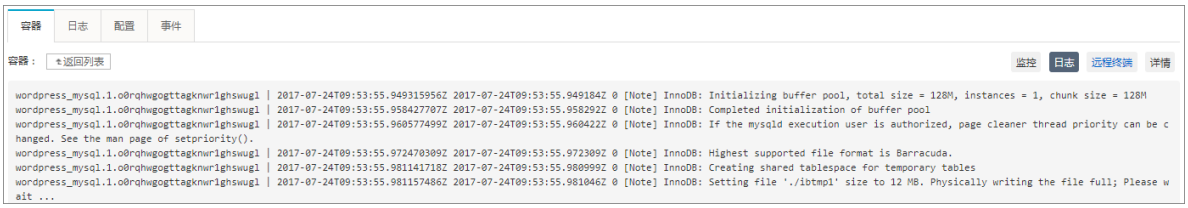
5. 选择所要查看的容器并单击**日志**。

图 156: 容器列表



您可以查看该容器的日志信息。

图 157: 容器日志



12.4.12 使用访问控制

简介

如果您使用容器服务创建了多个集群，您的组织里有多个用户需要使用这些集群。如果这些用户共享使用您的云账号秘钥，就会出现以下的问题：

- 您的秘钥由多人共享，泄密风险高。
- 您无法限制其他人的访问权限，容易出现误操作导致安全风险。

访问控制RAM是阿里云提供的资源访问控制服务。通过RAM，您可以集中管理您的用户，以及控制用户可以访问您名下哪些资源的权限。对于容器服务而言，90%以上的用户主要分配的是集群维度的权限，主要是对集群维度的是否可增删的限制。因此为了更简单地使用权限控制，容器服务提供了两种鉴权策略，一种是资源授权，一种是细粒度API授权。对于90%的用户而言使用资源分配鉴

权策略即可满足需求。对于RAM有深入的了解希望在API级别进行鉴权的用户可以使用API细粒度鉴权。

使用场景

假设是一家APP开发的公司，主要使用阿里云的服务提供APP的后端服务与基础设施和中间件，公司中的成员有后端开发工程师，测试工程师，运维工程师、基础平台工程师。从职责分配上来讲：

- 后端开发工程师负责开发APP的后端服务。
- 测试工程师负责测试APP的后端服务和基础的测试环境维护。
- 运维工程师负责所有的环境的运维（集群的伸缩，监控等）。
- 基础平台工程师负责中间件的维护与基础设计的建设，包括GitLab、Jenkins、Sentry等。

从职责划分上来看，不同的角色对于资源的需求不同，为了保证每个角色有足够的权限完成自己的任务，同时权限不越界。在这个场景中您使用资源分配鉴权即可。

主账号授权

说明: 当您进入RAM控制台的时候，可能会发现一些以acs_开头的RAM子账号，这些子账号是容器服务自动创建的子账号，请不要删除。

1. 登录访问控制云控制台，创建一个子账号并开启控制台登录。

   有关如何登录云控制台，参见登录云控制台。
2. 登录容器服务云控制台，单击左侧导航栏中的**集群**并单击**子账号授权**。
3. 选择相应的子账号并单击**下一步**。

图 158: 选择子账号



4. 选择对应的集群资源和权限，单击**下一步**并在弹出的确认对话框中单击**确定**。

图 159: 子账号授权



5. 单击**完成**完成授权。

使用子账号

使用子账号登录（要主账号进入RAM控制台并将控制台登录地址告知子用户）并进入容器服务云控制台。

此时，集群列表中显示被分配的集群。

深入理解鉴权策略

鉴权策略简单地来讲就是对云资源的使用权限的校验。鉴权分为两种，一种是资源分配鉴权，一种是API细粒度鉴权。

API细粒度鉴权是基于访问控制RAM进行的鉴权，相对来讲更加复杂。

资源分配鉴权是容器服务在API细粒度鉴权之上的包装，主要将API细粒度鉴权无法实现的资源分配在容器服务中进行了实现，并将API细粒度鉴权的鉴权策略封装为只读和读写两个权限。只读权限可以创建应用、删除应用等等，但是无法对集群级别的增删改查进行操作。而读写权限则可以对集群进行完全的管理。

高级用法

API细粒度鉴权可以实现API级别的鉴权。目前提供的后置鉴权条件如下所示。

鉴权 Action 名称	解释
GetClusterById	获取集群描述
GetClusterCerts	获取集群证书
CreateCluster	创建集群
DeleteCluster	删除集群

鉴权 Action 名称	解释
UpdateClusterSizeById	集群扩容

资源列表如下所示。

资源	解释
cluster	集群

参考示例：

```
{
  "Statement": [
    {
      "Action": "cs:*",
      "Effect": "Allow",
      "Resource": "acs:cs:*:*:cluster/cluster_id1"
    },
    {
      "Action": "cs:*",
      "Effect": "Allow",
      "Resource": "acs:cs:*:*:cluster/cluster_id2"
    }
  ],
  "Version": "1"
}
```

本示例表示授予集群的全部权限。

12.4.13 服务发现和负载均衡

服务发现和负载均衡主要解决通信的可靠性问题。为了达到可靠性，容器服务引入了负载均衡机制。通信又可以分为对外暴露服务的通信和内部服务之间的通信。下面根据场景引导您使用不同的解决方案。

场景一

普通且简单的7层协议负载均衡，Web服务的反向代理，推荐使用简单路由服务。更多详细信息，参加 简单路由#支持*HTTP/HTTPS#*， 简单路由-域名配置，简单路由-*HTTP*变成 *HTTPS*协议。

场景二

4　层协议的负载均衡，负载均衡直接负载均衡到多个相同功能的容器，在将传统架构迁移到容器架构过程中非容器集群的服务访问容器集群中容器的服务，推荐使用负载均衡路由。

场景三

同一个集群内，服务间需要相互发现和相互进行通信，且需要负载均衡的能力，推荐使用集群内服务间路由和负载均衡。

场景四

同一个集群内，服务间需要相互发现和相互进行通信，但是不需要负载均衡的能力，推荐使用容器互相发现。

场景五

对负载均衡和服务发现有较高的定制需求，例如需要支持泛域名，自定义错误页面，支持记录访问日志，URL参数值选择后端服务，自定义HAProxy配置文件等等，推荐使用自定义路由。更多详细信息，参见自定义路由使用说明，自定义路由使用示例。

12.4.13.1 简单路由（支持 HTTP/HTTPS）

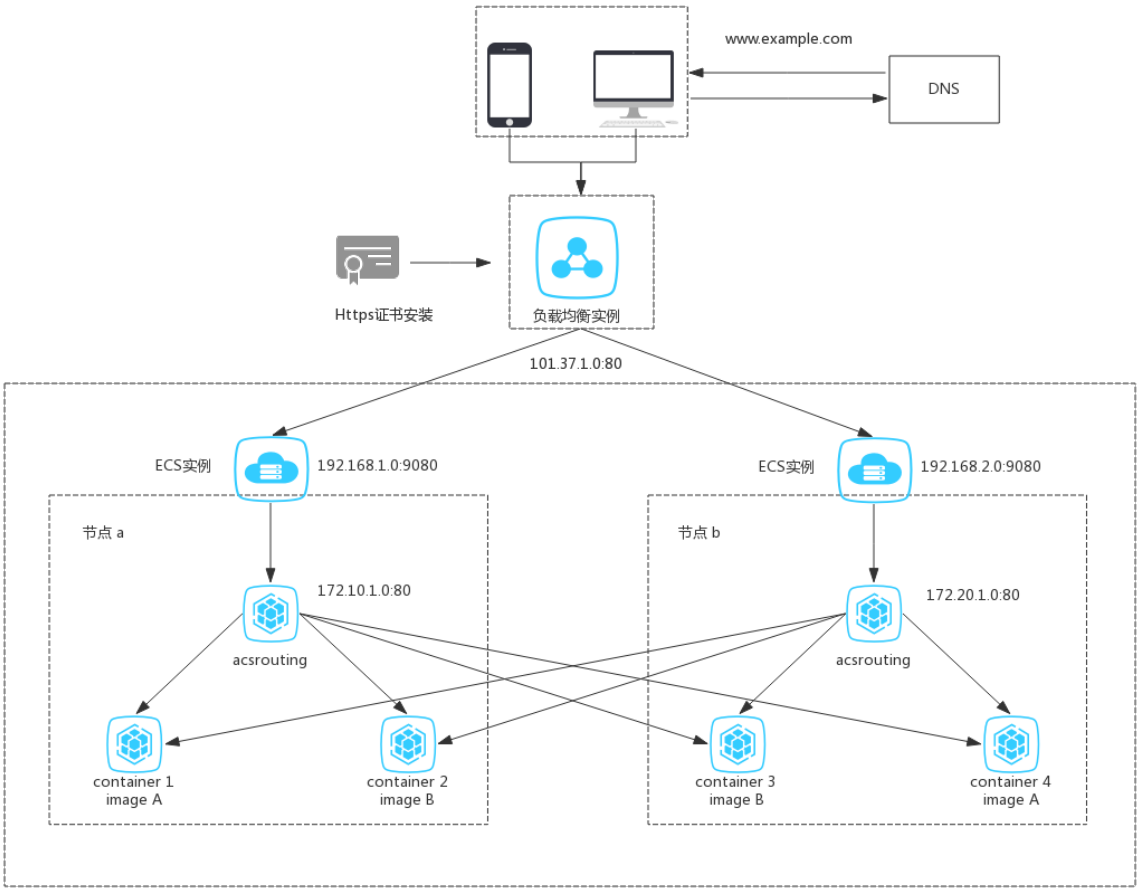
适用场景

普通且简单的7层协议负载均衡，Web路由服务，容器集群内服务之间7层协议互相访问的通信代理和负载均衡。

原理

当您新建一个集群的时候，会默认给这个集群分配一个负载均衡实例。该负载均衡实例会将集群中的所有节点加入作为后端，同时前端会暴露80端口，后端所有节点的机器会暴露9080端口。容器服务会启动一个路由应用acsrouting，即阿里云容器服务路由应用（Alibaba Cloud Container Service Routing）。该路由应用只有一个服务，即路由服务。该服务是全局（global）的，即每个节点（下面说到的主机和节点都是同一个意思，即ECS的VM实例）都部署了该服务（或者说镜像）的一个拷贝，也就是容器。每个节点都由这个容器用来路由HTTP服务或者HTTPS服务。

图 160: 简单路由原理



HTTP服务，负载均衡实例的前后端端口的映射为80:9080，主机与路由容器之间的端口映射为9080:80，即路由的容器暴露80端口，其他用作Web服务的容器可以暴露任意的端口。只要在容器启动的时候设置主机和容器端口的映射，routing服务就能获取到相应的端口进行请求的路由。

通过容器服务云控制台进行设置

通过服务 > 变更配置进行设置

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。
2. 单击左侧导航栏中的**服务**。
3. 选择所要暴露的服务所在的集群。
4. 选择所要暴露的服务（本示例中为nginx1）并单击**变更配置**。

图 161: 变更配置



5. 在变更配置页面，配置主机和容器端口的映射。

   主机端口为空，表示随机暴露一个主机的端口（暴露HTTP/HTTPS服务时，您可以不需要知道主机暴露的具体端口是什么，可以使用overlay网络或者VPC网络来直接访问容器的端口），容器端口为8080。您使用nginx默认暴露Web服务的8080端口来提供HTTP服务，使用的协议是TCP协议。

   图 162: 端口映射



   路由配置通过域名来暴露服务，须标明要暴露的端口，此处为Web服务8080端口。域名字段只填写了域名前缀，如果域名前缀为XXX，获得的域名为XXX.$cluster_id.$region_id.alicontainer.com供测试使用。您也可以填写自己的域名，需要添加解析到相应的负载均衡实例IP。关于配置路由的容器端口和HTTP服务的域名的详细信息，参见 *routing*标签。

通过应用的模板编辑器进行设置

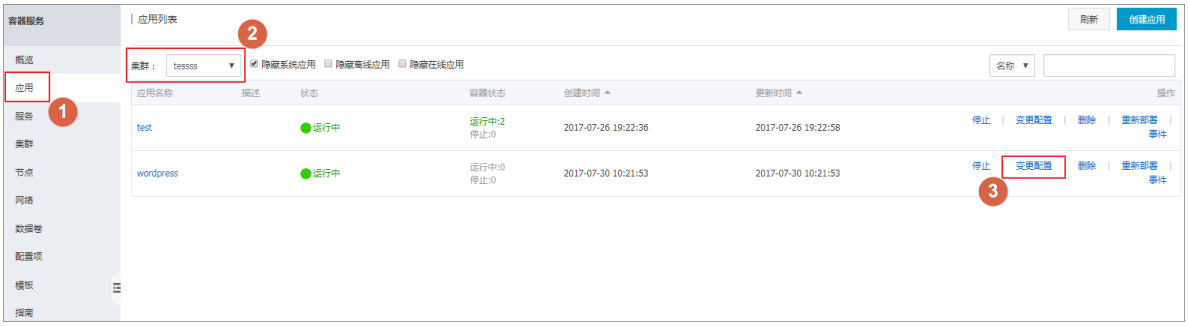
1. 登录容器云云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**应用**。
3. 选择目标应用所在的集群。

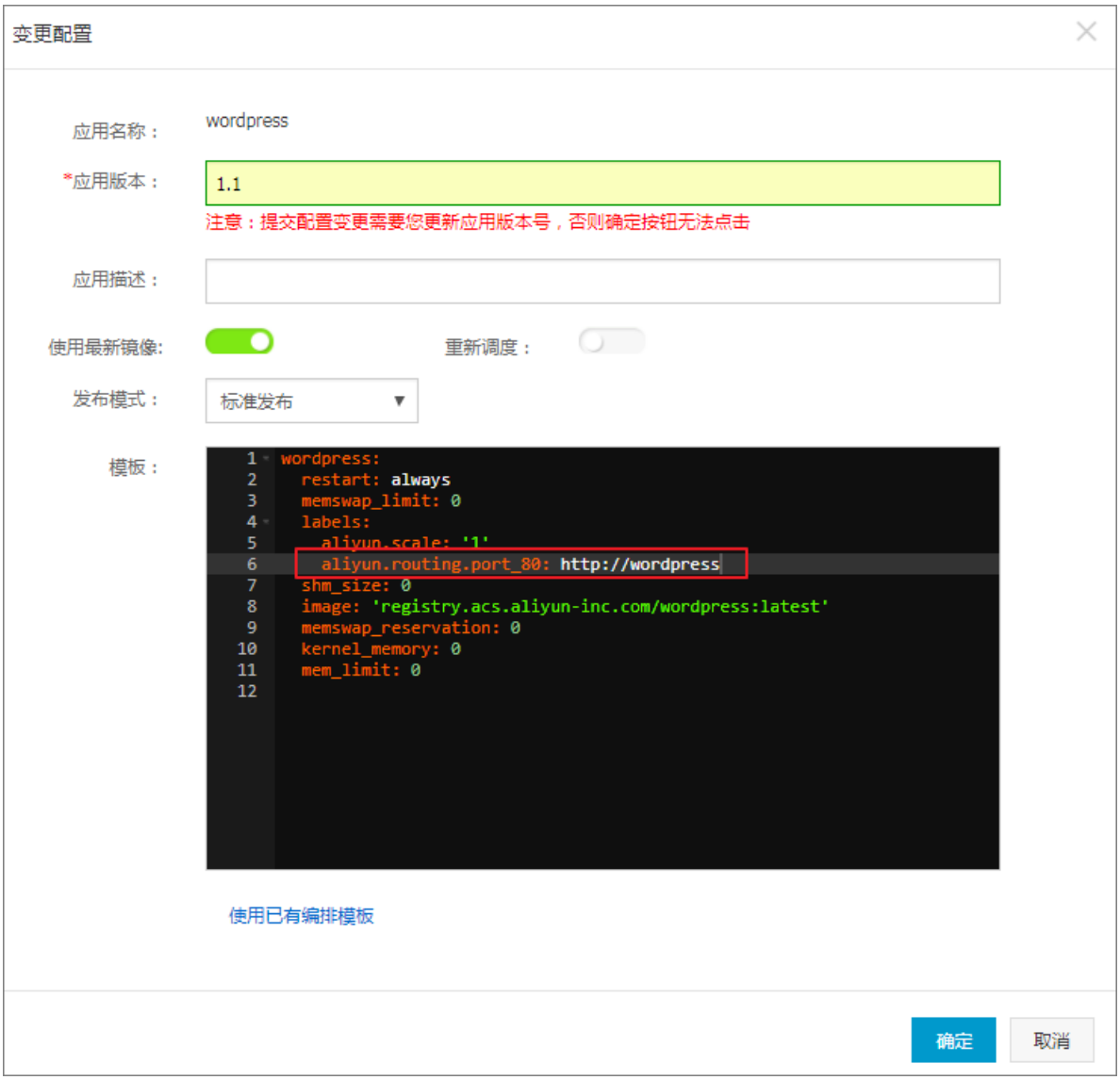
4. 选择目标应用（本示例中为wordpress）并单击**变更配置**。

图 163: 变更配置



5. 在模板编辑器中，添加*routing*标签，定义相应的域名或者域名前缀。注意升级应用的版本，以及确定是否拉取最新的Docker 镜像，最后单击**确定** 更新域名。

图 164: 变更配置



通过客户端工具进行设置

- docker help run：查看使用的“*-p*”选项，路由配置在容器服务云控制台进行。
- docker-compose：查看支持的“*ports*”选项，路由配置规则详情见*routing*标签。

12.4.13.2 简单路由-域名配置

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**服务**。
3. 选择要添加域名的服务所在的集群。

4. 选择要添加域名的服务（本示例中要添加域名的服务为web，所属的应用为wordpress）并单击**变更配置**。

图 165: 变更配置



5. 单击**简单路由配置**右侧的加号图标，输入要添加的域名（本示例中要添加的域名为www.example.com）并单击**确定** 更新配置。

> **说明:** 同一个服务同一个端口的多个域名只能写在同一个条目内，并且域名和域名之间用分号（;）分隔。

图 166: 简单路由配置



此时，服务处于更新中。更新完毕变成就绪状态后，路由服务acsrouting_routing就已经将该域名配置好了。当有请求以域名www.example.com访问服务web 时，就能正确的解析并转发到相应的服务了。

6. 将域名解析到容器服务的集群上。容器服务在创建集群的时候，会给每一个集群分配一个负载均衡实例，该负载均衡实例是属于您自己的。

   a) 单击容器服务云控制台左侧导航栏中的**集群**。

   b) 选择相应的集群，并单击**管理**。

   c) 单击**负载均衡**可以查看负载均衡的ID。

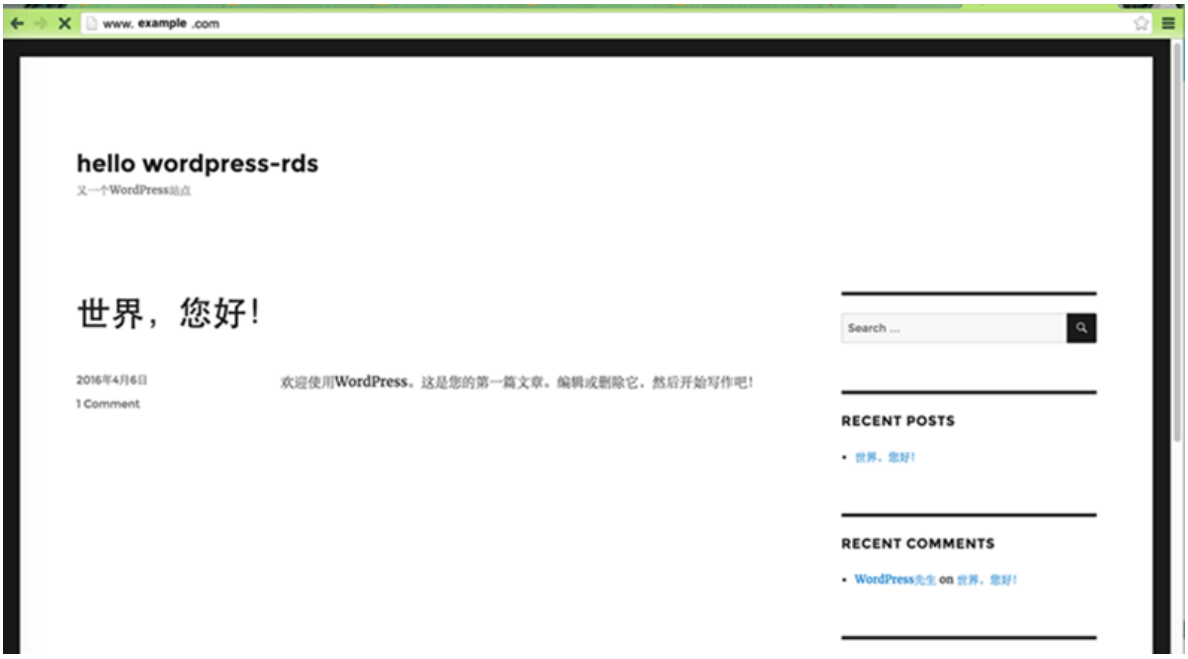
图 167: 负载均衡实例



d) 登录负载均衡云控制台，在实例列表页面找到集群的负载均衡并单击实例的ID。

您可以查看负载均衡实例的**服务地址**。

7. 以万网的域名解析为例，添加A记录，绑定域名www.example.com的地址到负载均衡实例的服务地址。
8. 访问页面www.example.com。



12.4.13.3 简单路由-HTTP协议变为HTTPS协议

前提条件

如果您还没有配置成功HTTP协议的域名访问，请先了解配置HTTP的域名访问。更多详细信息，参见简单路由-域名配置。

操作步骤

1. HTTPS协议是在负载均衡这一层进行支持的。为了支持HTTPS协议，您需要创建负载均衡证书。

a) 登录负载均衡云控制台。

有关如何登录云控制台，参见登录云控制台。

b) 单击**证书管理**页签。

c) 单击**上传安全证书**。

d) 在**上传安全证书**对话框，填写证书信息，然后单击**确认**。

2. 证书创建成功后，找到创建集群时分配的负载均衡实例。

容器服务在创建集群的时候，给每一个集群分配了一个负载均衡实例，该负载均衡实例是属于您自己的。

a) 单击容器服务云控制台左侧导航栏中的**集群**，选择相应的集群，单击**管理**。

b) 单击**负载均衡**，您可以查看负载均衡实例的ID。

图 168: 负载均衡实例



c) 登录负载均衡云控制台，在实例列表页面选择集群的负载均衡实例并单击实例的ID。

您可以查看负载均衡实例的服务地址。

3. 单击**服务监听**页签，打开**服务监听**列表页面。

4. 单击**添加**。

5. 在**添加服务监听**对话框，指定监听配置，然后单击**确认**完成。

a) 前端协议，选择HTTPS。

b) 端口使用443端口，后端端口使用9080端口（该端口为路由服务**acsrouting_routing**在每一台ECS主机上暴露的端口，所有的HTTP请求会在路由服务**acsrouting_routing**上根据HTTP协议的HOST header转发到相应的提供各种服务的容器内）。

c) 选择前面步骤创建的证书www.example.com。

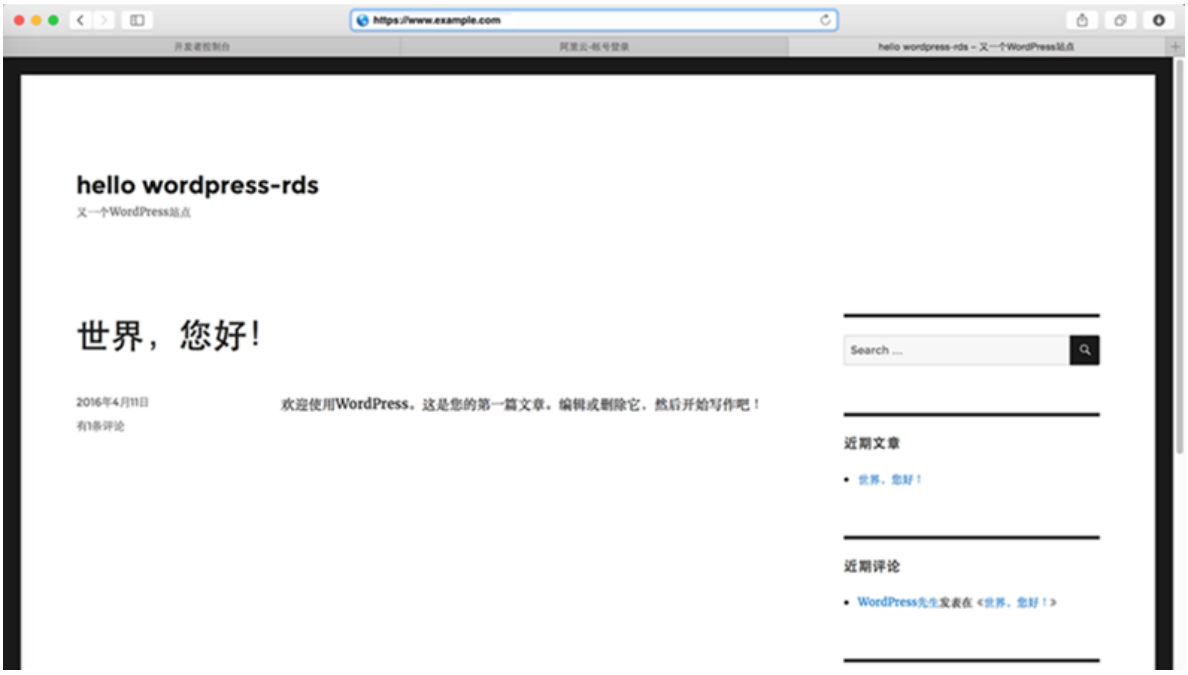
d) 根据需要设置其他选项。

e) 单击**下一步**。

6. 完成**健康检查配置**标签页中的配置。单击**确认**。

您可以选择开启或关闭健康检查。如果您选择开启健康检查，您需要在**域名** 中填写您自己的域名或者在**检查路径**中填写/haproxy-monitor。否则，健康检查会报异常。

7. 配置成功后，单击**确认**。
8. 访问页面https://www.example.com。



12.4.13.4 集群内服务间路由和负载均衡

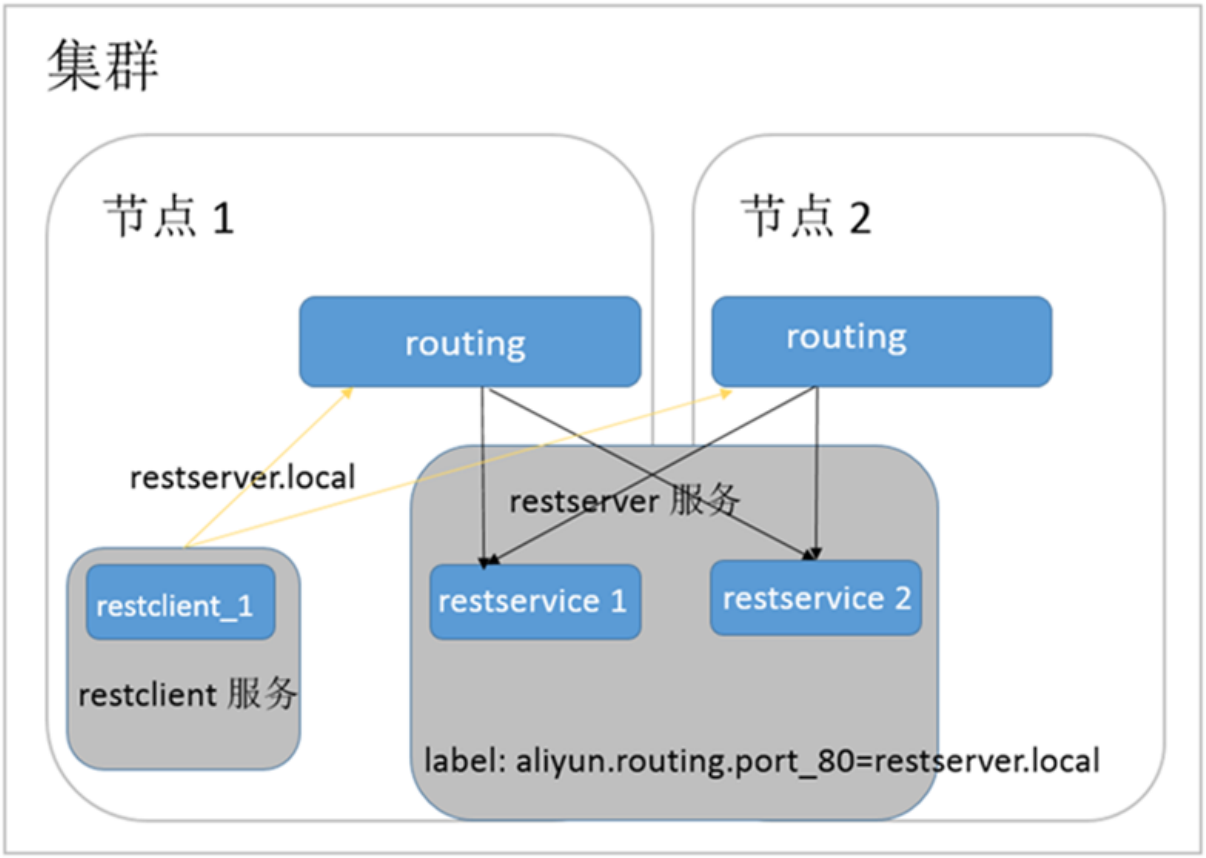
在容器服务上可以通过 简单路由（支持 HTTP/HTTPS）将基于域名的HTTP服务暴露出去，而且能够配合健康检查自动的负载均衡和服务发现，当其中一个容器出现问题之后，routing会自动将健康检查失败的容器从后端摘除，所以能做到自动的服务发现。

背景信息

然而这个例子是将服务暴露到外网，那么服务间如何通过这种方式做到自动的服务发现和负载均衡呢？容器服务引入了负载均衡功能，您只需要使用以.local结尾的域名，并在依赖的服务的external_links中增加这个域名，依赖的服务便可以通过.local的域名访问到依赖的服务，并且能够配合健康检查做到自动的服务发现。

实现原理

- 利用了Docker 1.10之后支持在容器中做别名的方式，在依赖负载于restserver.local的服务的容器中restserver.local的域名解析到的是routing服务的地址，这样可以将HTTP请求转发到routing的容器，并带上HOST为restserver.local的请求头。
- routing会对配置了aliyun.routing_port_xxx: restserver.local的服务监听其容器的健康状态并挂载到HAProxy的后端，HAProxy接收到带有restserver.localHOST头的HTTP请求就能转发到对应了容器了。



优势

- 相对于使用link或者hostname的基于DNS的方式，首先不同客户端对DNS缓存的处理不一致会导致服务发现的延迟性，其次DNS的方案也只有round robin，对于微服务的场景是不够用的。
- 而相对于其他的微服务服务发现的解决方案，提供了一个实现无关的服务发现和负载均衡机制，无需server端和client应用做任何修改即可使用。
- 服务生命周期是解耦的，每个微服务可以采用一个docker-compose模板独立部署、更新。相互之间只是通过一个虚拟域名实现动态绑定即可。

编排实例

```
restserver: #模拟rest服务
  image: registry.acs.aliyun-inc.com/nginx
  labels:
    aliyun.routing.port_80: restserver.local # 使用local的域名，只有集群内的容器可以访问这个域名
    aliyun.scale: "2" # 扩展出两个实例，模拟负载均衡
    aliyun.probe.url: "http://container:80" # 定义容器的健康检查策略是http，端口是80
    aliyun.probe.initial_delay_seconds: "2" # 健康检查在容器起来之后两秒之后再检查
    aliyun.probe.timeout_seconds: "2" # 健康检查超时时间，如果两秒还没返回认为不健康
restclient: # 模拟rest服务消费者
  image: registry.acs.aliyun-inc.com/wordpress
  command: "sh -c 'apk update; apk add curl; while true; do curl --head restserver.local; sleep 1; done'" #访问rest服务，测试负载均衡
```

```
tty: true
external_links:
 - "restserver.local" #指定link的服务的域名。请确保您设置了external_links，否则访问会失败。
```

然后，通过如下的restclient服务的日志，您可以看到restclient的curl的HTTP请求被路由到不同的rest服务的容器上了。

```
yy_restclient_1 | 2017-07-30T04:23:48.354508898Z HTTP/1.1 200 OK

2017-07-30T04:23:48.354544762Z Server: nginx/1.13.1

2017-07-30T04:23:48.354587085Z Date: Sun, 30 Jul 2017 04:23:48 GMT

2017-07-30T04:23:48.354594544Z Content-Type: text/html

2017-07-30T04:23:48.354597850Z Content-Length: 612

2017-07-30T04:23:48.354600909Z Last-Modified: Tue, 30 May 2017 13:03:46 GMT

2017-07-30T04:23:48.354603948Z ETag: "592d6db2-264"

2017-07-30T04:23:48.354607001Z Accept-Ranges: bytes

2017-07-30T04:23:48.354610095Z Set-Cookie: CONTAINERID=a2939e3f1ccexxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxbd820318a395c; path=/

2017-07-30T04:23:48.354613271Z

2017-07-30T04:23:49.368065162Z HTTP/1.1 200 OK

2017-07-30T04:23:49.368104038Z Server: nginx/1.13.1

2017-07-30T04:23:49.368116051Z Date: Sun, 30 Jul 2017 04:23:49 GMT

2017-07-30T04:23:49.368120357Z Content-Type: text/html

2017-07-30T04:23:49.368123361Z Content-Length: 612

2017-07-30T04:23:49.368126572Z Last-Modified: Tue, 30 May 2017 13:03:46 GMT

2017-07-30T04:23:49.368129557Z ETag: "592d6db2-264"

2017-07-30T04:23:49.368132791Z Accept-Ranges: bytes

2017-07-30T04:23:49.368135760Z Set-Cookie: CONTAINERID=f92748xxxxxxxxxxxxxx
xxxxxxxxxxxxxx9623f0659d0976; path=/
```

12.4.13.5 负载均衡路由

暴露HTTP协议或者HTTPS协议的服务

推荐使用简单路由服务（即 routing）的方式来暴露HTTP服务或者HTTPS协议的服务，如果您希望搭建自己的路由链路，可以开通新的内网或者公网负载均衡实例路由到VM的端口（通过label *lb*来实现），并设置主机和容器的映射关系来进行请求的路由。

适用场景：

7层协议负载均衡，自定义各服务的路由，在将传统架构迁移到容器架构过程中非容器集群的服务访问容器集群中容器的服务。

暴露TCP协议或者UDP协议的服务

目前如果要暴露TCP协议的服务，需要您自行设置负载均衡实例或者公网IP，并配置好主机端口与容器端口的映射（通过label *lb*来实现）。

适用场景：

4层协议的负载均衡，自定义各服务的路由，在将传统架构迁移到容器架构过程中非容器集群的服务访问容器集群中容器的服务。

示例：

通过自定义负载均衡的方式来将容器集群内的Redis服务暴露给容器集群外的Python应用。

操作步骤

1. 首先在负载均衡云控制台（单击**创建实例**）创建一个用于路由的负载均衡实例。

   有关如何登录云控制台，参见登录云控制台。

   本示例中选择的是外网实例，您可以根据自己的需要选择外网或者内网。

2. 返回负载均衡云控制台，将创建的负载均衡实例命名为slb_redis_app。容器服务会通过该名称来引用这个负载均衡实例。

   选择所需的实例，编辑实例的名称并单击**确定**。

3. 创建监听端口。

   a) 在负载均衡实例列表中，单击所需实例的ID链接，打开负载均衡实例详情页面。

   b) 单击**服务监听**页签，打开**服务监听**列表页面。

   c) 单击**添加**。

   d) 在**添加服务监听**对话框，创建协议为TCP，端口映射为6379:6379，然后单击**确认**完成。

4. 登录容器服务云控制台，选择一个已有的集群，创建一个名称为 redis-demo 的应用，单击 **使用镜像创建**。

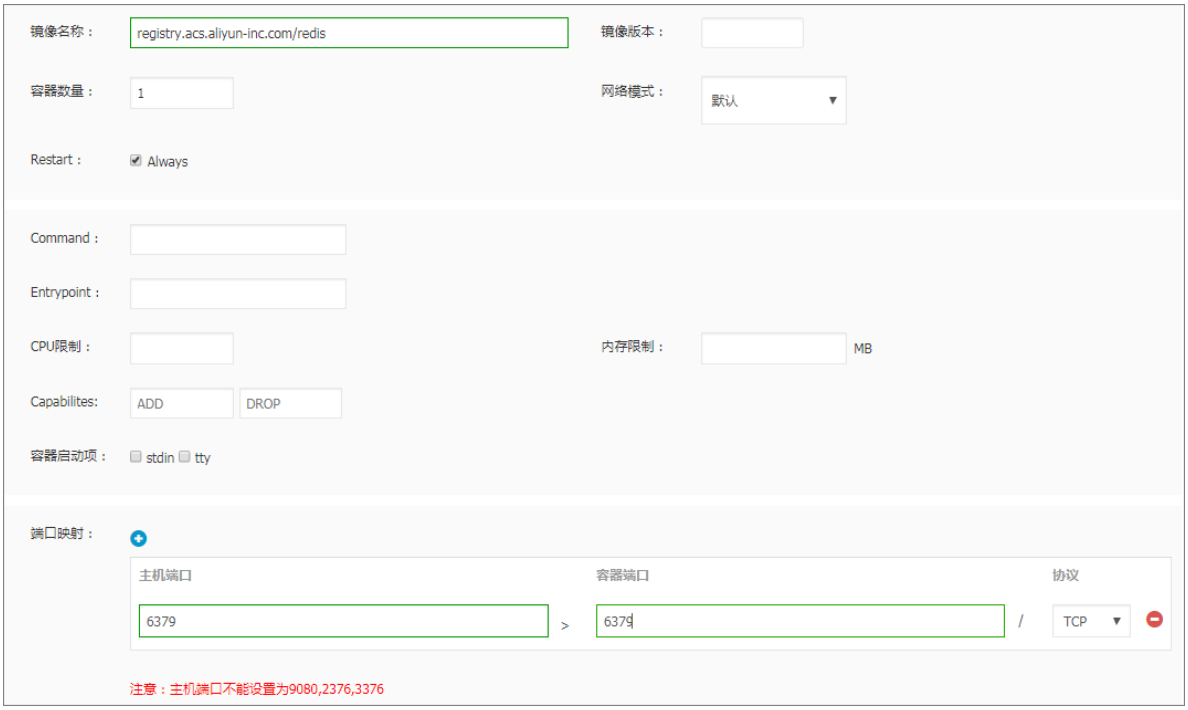
   有关如何创建应用，参见使用镜像创建应用。

5. 填写Redis镜像地址并设置**端口映射**。

   说明: 此处Redis镜像只是开通了容器的6379端口，为了使创建的负载均衡路由到这个容器端口，您必须知道Redis镜像的主机：端口映射。

在**端口映射**　　中，指定主机端口为6379，主机端口6379即为负载均衡实例绑定的后端主机端口，选择使用的协议为TCP。

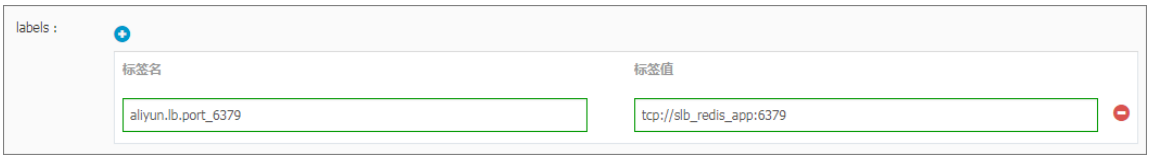
图 169: 端口映射



6. 为了配置自定义负载均衡，需要让Redis服务知道使用的负载均衡实例的信息。您可以通过向服务注入一个标签来实现或者通过设置**负载均衡路由配置**。

   - 向服务注入一个标签。本示例中，标签为aliyun.lb.port_6379: tcp://slb_redis_app:6379。

     图 170: 设置labels

     aliyun.lb.port_$container_port:$scheme://$[slb_name|slb_id]:$front_port

     标签格式如下，带 $ 的变量为占位符。

     - $container_port表示容器要暴露的端口。
     - $scheme表示负载均衡实例监听端口支持的协议，可能的值为tcp、http、https、udp。
     - $[slb_name|slb_id]表示可以填写负载均衡实例的名称或者ID。
     - $front_port表示负载均衡实例要暴露的前端端口。

更多详细信息，参见*lb*。

- 在**创建应用**页面，单击**负载均衡路由配置**右侧的加号图标，设置要配置的负载均衡实例的信息。

  该设置对应的标签内容为 aliyun.lb.port_6379: tcp://slb_redis_app:6379。

  图 171: 设置负载均衡路由



本示例中，路由到的容器端口为6379，引用前面创建的负载均衡实例名称slb_redis_app，与上面主机：容器端口映射设置的TCP协议相呼应，本示例设置监听端口的协议为TCP协议，同时设置负载均衡的前端端口为6379。

说明: 本示例中，同时将负载均衡实例的前端端口、后端端口（即主机的端口）和容器端口均设置为6379，您可以根据自己的需要设置不同的前端端口和主机端口。

7. 单击**创建**，Redis应用即开始创建了。

   Redis应用在创建的过程中会自动将名称为slb_redis_app的负载均衡实例绑定到部署了redis镜像的后端主机。

8. 当应用处于就绪状态后，登录负载均衡云控制台，查看名为slb_redis_app的负载均衡实例的状态。

   在实例列表中，选择所需的实例，单击实例的ID链接 ， 单击**后端服务器**页签，即可查看绑定到负载均衡的后端服务器。

   由健康状态可见，负载均衡已经正确地绑定到了Redis的后端。

9. 您可以在负载均衡云控制台的实例详情页面查看负载均衡实例的IP地址，并使用命令行工具telnet $*server_load_balancer_ip_address* 6379来检查端口的可访问性。

10. 为了测试以上配置，在本地运行一个简单的Python应用来通过slb_redis_app负载均衡实例访问容器集群内的Redis。

说明: Redis主机地址是负载均衡的IP地址。

app.py

```
from flask import Flask
from redis import Redis
app = Flask(__name__)
redis = Redis(host='$server_load_balancer_ip_address', port=6379)
@app.route('/')
def hello():
redis.incr('hits')
return 'Hello World! I have been seen %s times.' % redis.get('hits')
if __name__ == "__main__":
app.run(host="0.0.0.0", debug=True)
```

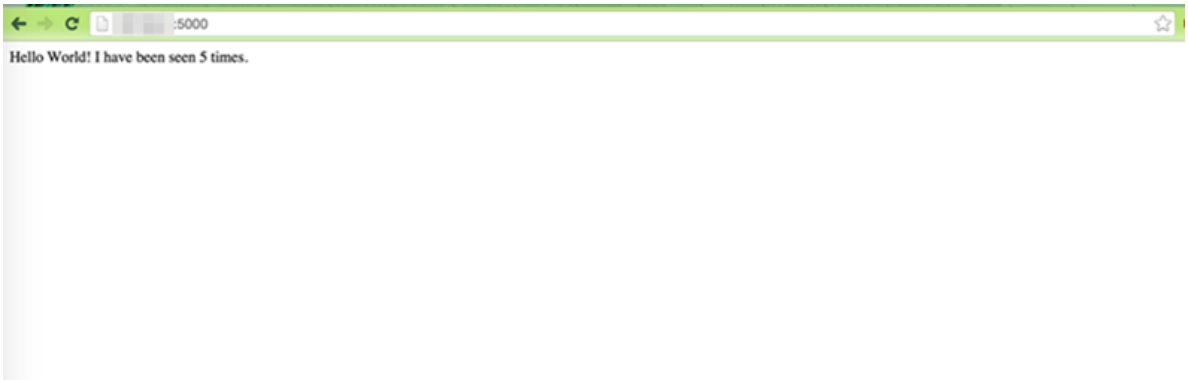
requirements.txt

```
flask
redis
```

shell

```
$ pip install -r requirements.txt
$ python app.py
Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
Restarting with stat
Debugger is active!
Debugger pin code: 243-626-653
```

访问结果如下图所示。



12.4.13.6 容器间的互相发现

容器服务为集群内的服务和容器提供多种服务发现方式，可以通过容器名、link、hostname等进行发现。

通过容器名

容器服务不仅可以通过容器的IP进行访问，还可以通过网络中其他容器的容器名进行访问，通过容器网络互连中的例子，您可以在*cross-host-network-test2*的容器中通过*cross-host-network-test1*的容器名进行访问。

如果在编排文件中不指定container_name的话，默认的容器名为*{project-name}_{service-name}_{container-index}*，所以在连接管理终端后，您可以通过另外一个服务的容器名进行访问。




```
/ # ping testproject_cross-host-network-test1_1
PING testproject_cross-host-network-test1_1 (172.[redacted]): 56 data bytes
64 bytes from 172.[redacted]: seq=0 ttl=64 time=0.037 ms
64 bytes from 172.[redacted]: seq=1 ttl=64 time=0.043 ms
64 bytes from 172.[redacted]: seq=2 ttl=64 time=0.043 ms
^C
--- testproject_cross-host-network-test1_1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.037/0.041/0.043 ms
```

通过link

容器服务支持编排模板服务间的link，服务间的link 可以将一个服务的容器link到另外一个服务的容器中，而容器中可以通过link进来的服务别名访问到依赖的容器，并且在依赖的容器的IP变化时可以动态的更新别名解析的IP。具体的例子可以参考下面的WordPress编排示例，其中WordPress中Web服务link db:mysql的服务到容器内，容器内部就可以通过MySQL的域名访问到db服务的容器。

```
web:
  image: registry.acs.aliyun-inc.com/wordpress:4.5
  ports:
    - '80'
  environment:
    WORDPRESS_AUTH_KEY: changeme
    WORDPRESS_SECURE_AUTH_KEY: changeme
    WORDPRESS_LOGGED_IN_KEY: changeme
    WORDPRESS_NONCE_KEY: changeme
    WORDPRESS_AUTH_SALT: changeme
    WORDPRESS_SECURE_AUTH_SALT: changeme
    WORDPRESS_LOGGED_IN_SALT: changeme
    WORDPRESS_NONCE_SALT: changeme
    WORDPRESS_NONCE_AA: changeme
  restart: always
  links:
    - 'db:mysql'
  labels:
    aliyun.logs: /var/log
    aliyun.probe.url: http://container/license.txt
    aliyun.probe.initial_delay_seconds: '10'
    aliyun.routing.port_80: http://wordpress
    aliyun.scale: '3'
db:
  image: registry.acs.aliyun-inc.com/mysql
  environment:
    MYSQL_ROOT_PASSWORD: password
```

```
  restart: always
  labels:
    aliyun.logs: /var/log/mysql
```

通过hostname

如果在编排模板的服务中定义了*hostname*的配置，则在集群中便可以通过这个hostname访问到这个容器。

例如：

```
testhostname:
        image: registry.acs.aliyun-inc.com/busybox
        hostname: xxserver
        command: sleep 100000
        tty: true
```

那么，集群中就可以通过xxserver解析并访问到这个服务的容器，并当这个服务在有多个容器时，通过这个域名访问还可以做到一定的负载均衡的作用。

另外，如果服务没有配置hostname的话，容器服务会把容器的容器名作为容器内部的hostname；如果有应用需要在容器内知道自己的容器名，用于服务的注册，比如Eureka Client，需要注册一个可被访问的地址到Eureka Server，容器内的进程可以获取到容器名用于服务注册，并让其他的服务调用者通过容器名互相访问。

12.4.14 发布策略

容器服务支持简单路由蓝绿发布和负载均衡蓝绿发布。

12.4.14.1 概述

容器服务提供两种发布策略：蓝绿发布和标准发布。两种发布策略的差异如下表所示。

表 23: 发布策略比较

发布策略	区别
标准发布	在部署新版本应用时删除旧版本应用。发布过程中，您的服务会出现短暂中断。
蓝绿发布	应用更新时生成蓝绿两个版本，两个版本互相热备，通过切换路由权重的方式实现不同版本应用的上下线。 该发布策略具有以下特点： • 服务不会中断 • 应用零宕机

发布策略	区别
	• 发布过程用户无感知重启 • 升级失败自动回滚 • 同一资源栈可以多次升级迭代

标准发布使用场景

标准发布是传统的应用发布策略，在现有环境中，部署新版本应用。除了有特殊需求以外，一般采用此类发布策略，使用场景包括：

- 新版本应用已经过充分测试，无需线上试运行。
- 应用包含数据库服务，且发生过如数据表结构变更等不可逆转的升级情况，标准发布可以免除类似数据迁移、回滚等业务改造的工作。

蓝绿发布使用场景

蓝绿发布适用的应用形态包括前端类服务和后端类服务，常用于应用的增量升级中，面向客户的具体业务场景包括：

保证业务连续性

对业务变化迅速的互联网企业，应用版本迭代频繁是一个现实的挑战，保证线上业务的连续性是必要的。蓝绿发布策略主要的价值是零宕机应用更新，保证服务不中断，能满足云环境下持续发布的应用更新需求。

新版本线上评测

蓝绿发布策略可以让您在线上环境进行版本测试，通过保持一段时间内新旧版本并存，来充分测试新版本服务的功能、性能和安全性。待新版本稳定后，再将旧版本应用下线。

12.4.14.2 使用简单路由蓝绿发布

背景信息

蓝绿发布是一种**零宕机**的应用更新策略。进行蓝绿发布时，应用的旧版本服务与新版本服务会同时并存，同一个应用不同版本的服务之间共享路由，通过调节路由权重的方式，可以实现不同版本服务之间的流量切换。验证无误后，可以通过发布确认的方式将应用的旧版本的服务删除；如果验证不通过，则进行发布回滚，应用的新版本会进行删除。

前置条件

支持蓝绿发布需要将路由服务升级到最新的版本。详细信息参见 升级系统服务。

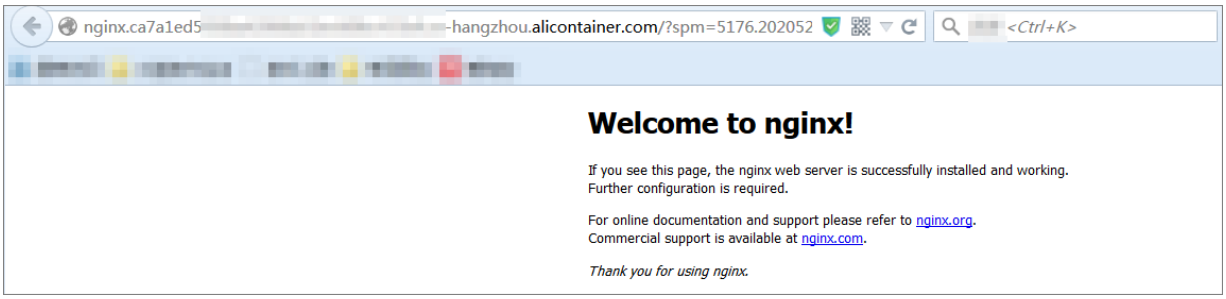
场景介绍

假设您要进行蓝绿发布的应用是一个Nginx的静态页面，初始的应用模板如下。

```
nginx-v1:
  image: registry.acs.aliyun-inc.com/nginx
  labels:
    aliyun.routing.port_80: nginx
  restart: always
```

部署后页面的效果如下。

图 172: 访问Nginx



操作步骤

1. 登录容器服务云控制台。

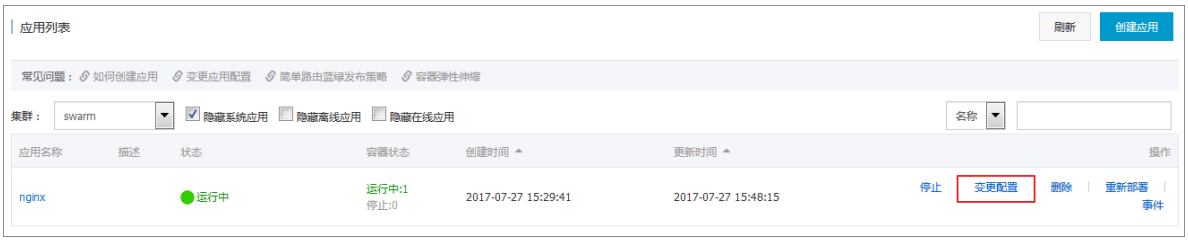
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**应用**。

3. 选择目标应用所在的集群。

4. 选择目标应用并单击**变更配置**。

   图 173: 变更配置



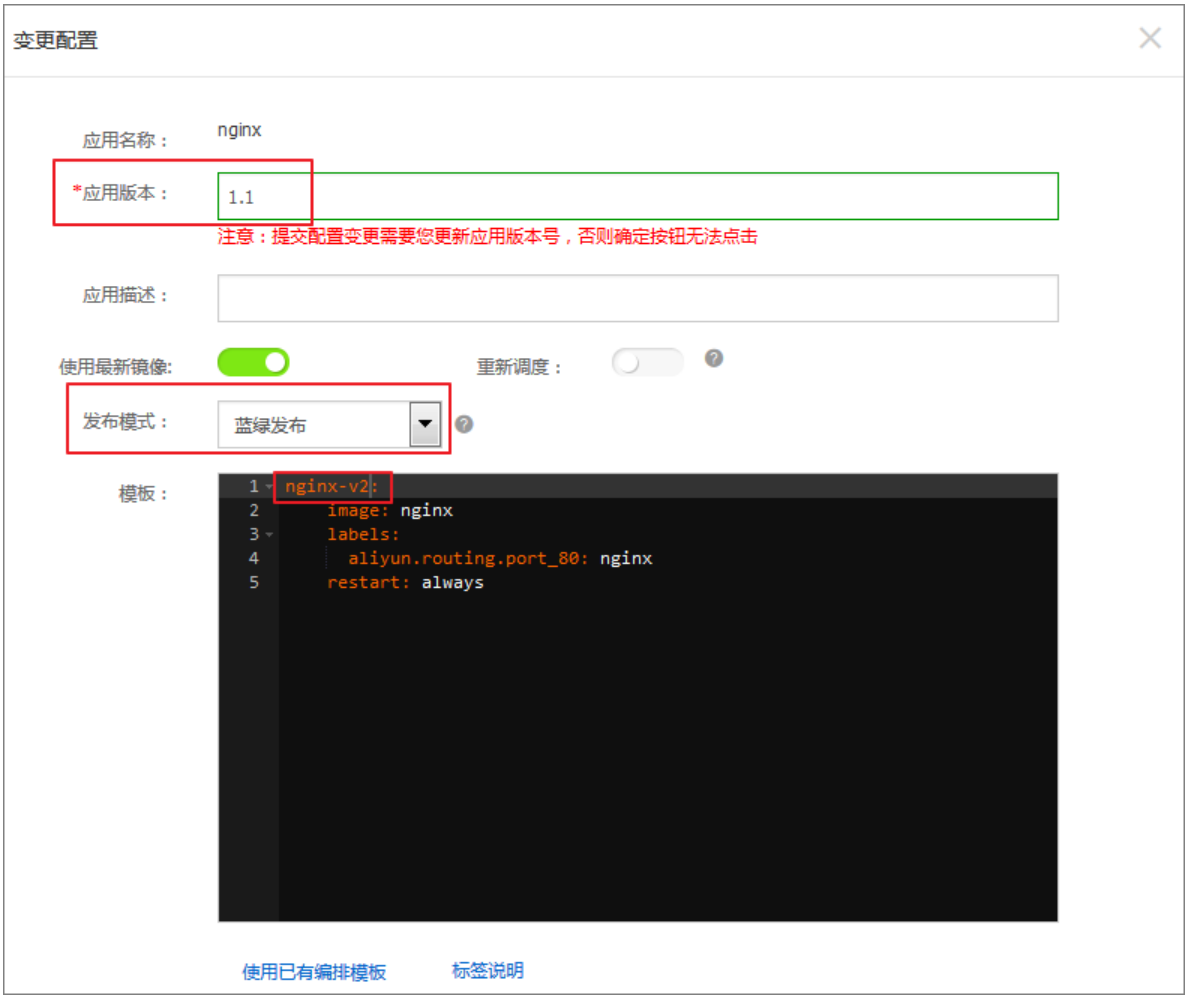
5. 选择变更的发布模式与新版本服务的配置。

   说明:

   - 在蓝绿发布中，新版本与旧版本不能共用同一个名字。

- 在蓝绿发布的场景中，为了保证应用的零宕机切换，新版本的服务的路由权重默认为0，需要通过路由管理页面进行调整，方可进行流量切换。

图 174: 变更配置



模板样例如下所示：

```
nginx-v2:
  image: registry.acs.aliyun-inc.com/nginx
  labels:
    aliyun.routing.port_80: nginx
  restart: always
```

6. 单击**确定**，发布变更。

在发布的过程中，会经历两个状态：

- 蓝绿发布中：表示新版本的服务尚未启动完成。
- 蓝绿发布待确认：表示新版本的服务已经启动完成，此时需要进行发布确认或者发布回滚方可进行下一次发布。

进入应用的详情页面，可以看到新版本的应用和旧版本的应用并存。

图 175: 蓝绿发布



7. 单击 **路由列表** 并单击 **设置服务权重**。

如下图所示，旧版本服务的权重为 100，新版本服务的权重为 0。

图 176: 路由权重



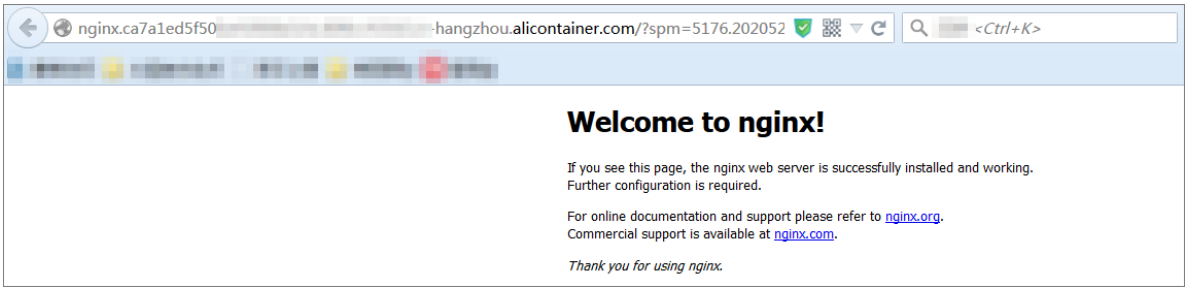
将旧版本服务的权重调整为 0，新版本服务的权重调整为 100。

图 177: 设置路由权重



8. 由于默认路由服务是进行会话保持的，您可以打开一个新的浏览器窗口，访问新的版本。

图 178: 访问Nginx



9. 当整个发布流程验证完毕后，在应用列表页面，单击**确认发布完成**并在弹出的确认对话框中单击**确认**进行发布确认，方可进行下一次发布。

图 179: 确认发布完成

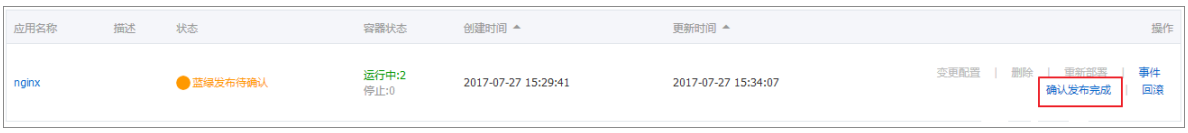


图 180: 发布确认



您可以看到应用的服务列表已经更新了，旧的服务已经完全下线删除了。

图 181: 发布完成



12.4.14.3 使用负载均衡路由蓝绿发布

背景信息

蓝绿发布是一种**零宕机**的应用更新策略。进行蓝绿发布时，应用的旧版本服务与新版本服务会同时并存，同一个应用不同版本的服务之间共享负载均衡，通过调节负载均衡权重的方式，可以实现不同版本服务之间的流量切换。验证无误后，可以通过发布确认的方式将应用的旧版本的服务删除；如果验证不通过，则进行发布回滚，应用的新版本会进行删除。

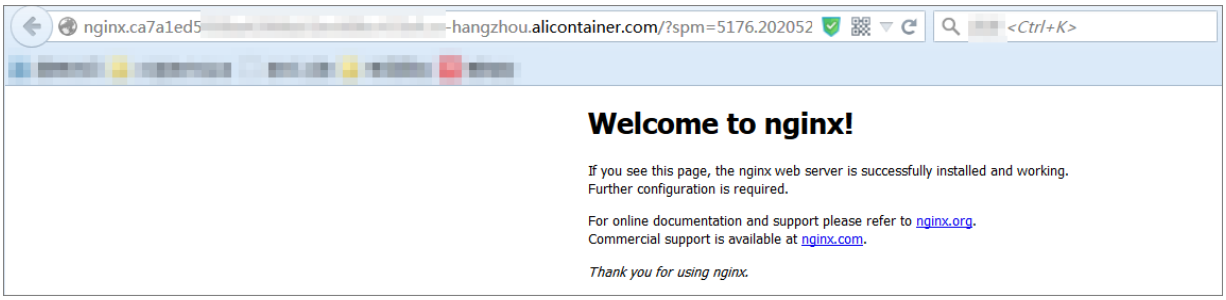
场景介绍

假设您要进行蓝绿发布的应用是一个Nginx的静态页面，初始的应用模板如下。

```
nginx-v1:
  image: registry.acs.aliyun-inc.com/nginx
  ports:
    - 80:80/tcp
  labels:
    aliyun.lb.port_80: tcp://proxy_test:80
  restart: always
```

部署后页面的效果如下。

图 182: 访问Nginx



操作步骤

1. 登录容器服务云控制台。

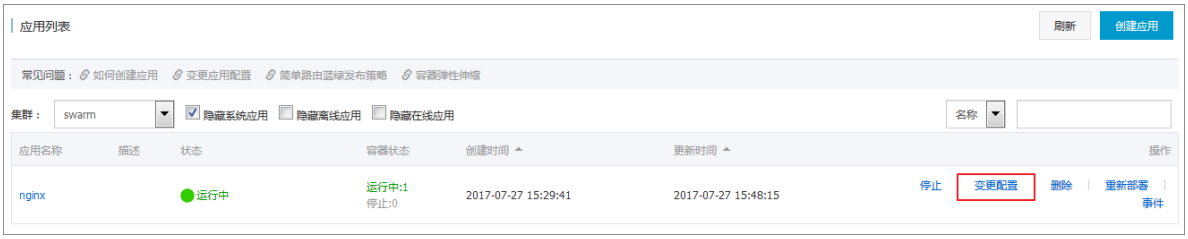
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**应用**。

3. 选择目标应用所在的集群。

4. 选择目标应用并单击**变更配置**。

图 183: 变更配置

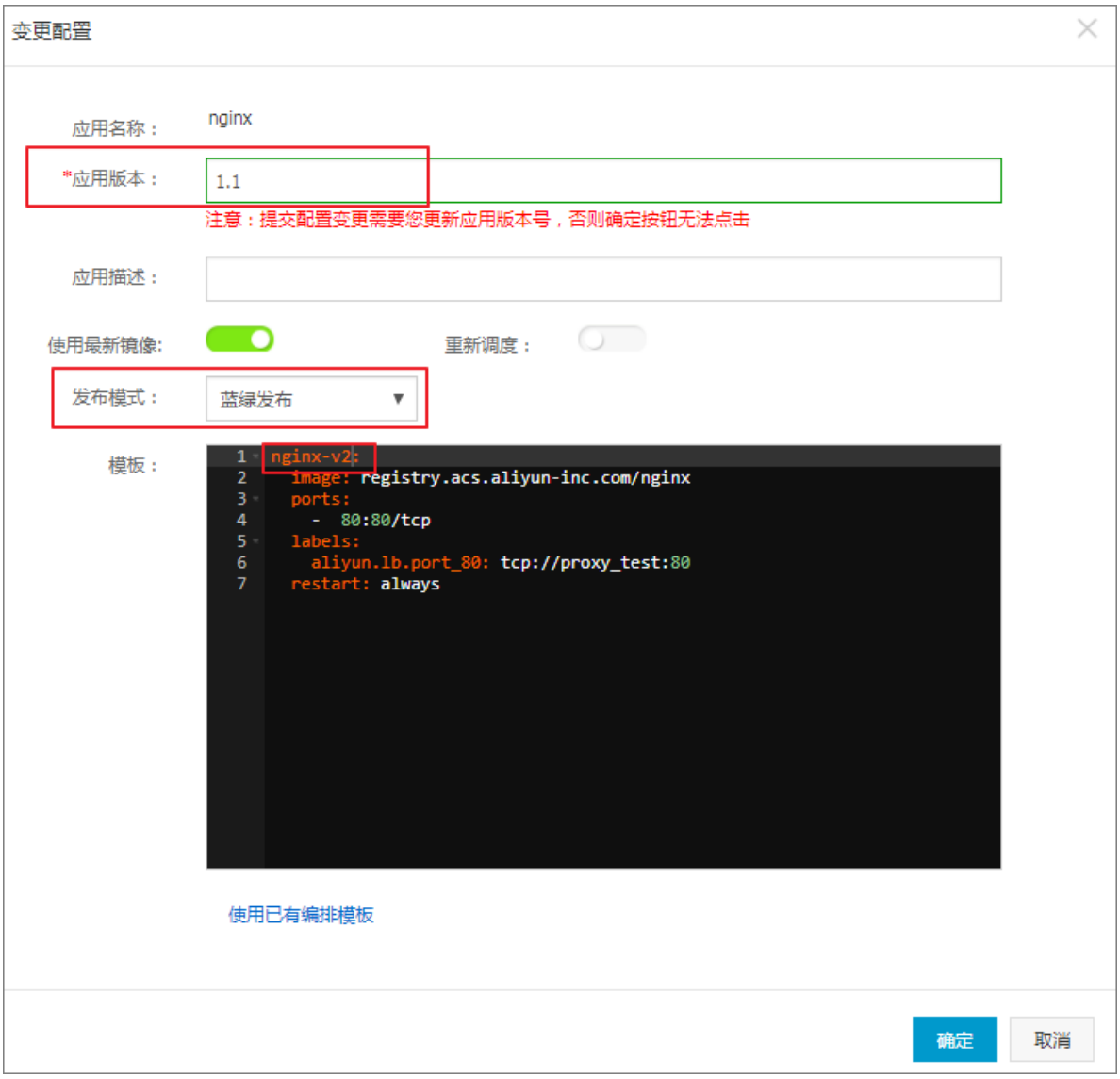


5. 选择变更的发布模式与新版本服务的配置。

说明:

- 在蓝绿发布中，新版本与旧版本不能共用同一个名字。
- 在蓝绿发布的场景中，为了保证应用的零宕机切换，新版本的服务的路由权重默认为 0，需要通过路由管理页面进行调整，方可进行流量切换。

图 184: 变更配置



模板样例如下所示：

```
nginx-v2:
  image: registry.acs.aliyun-inc.com/nginx
  ports:
    - 80:80/tcp
  labels:
    aliyun.lb.port_80: tcp://proxy_test:80
  restart: always
```

6. 单击**确定**，发布变更。

   在发布的过程中，会经历两个状态：

   - 蓝绿发布中：表示新版本的服务尚未启动完成。
   - 蓝绿发布待确认：表示新版本的服务已经启动完成，此时需要进行发布确认或者发布回滚方可进行下一次发布。

进入应用的详情页面，可以看到新版本的应用和旧版本的应用并存。

图 185: 蓝绿发布



7. 单击**路由列表**并单击**设置服务权重**。

旧版本服务的权重为 100，新版本服务的权重为 0。

图 186: 路由权重

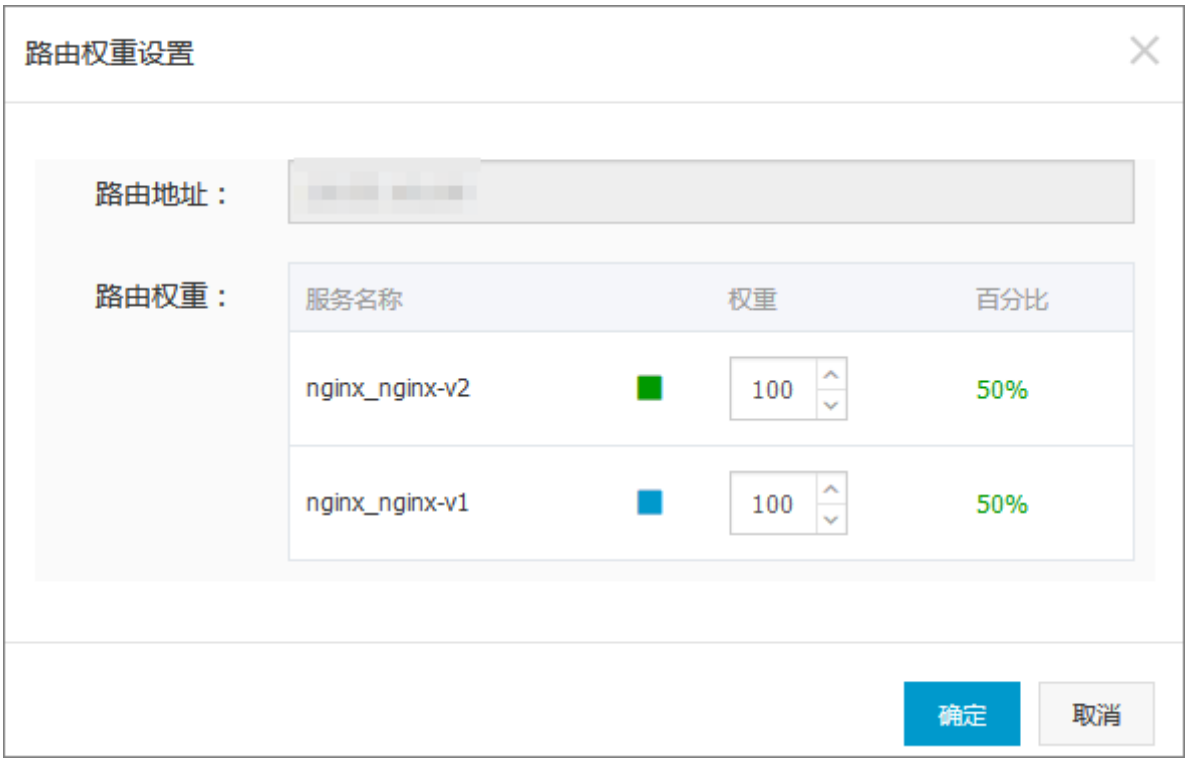
图 187: 初始流量



要做到**零宕机**升级，您需要先把新版本的服务的权重值调整为 100，此时新旧服务的权重各占 50%，测试新旧版本的服务都有稳定的流量。

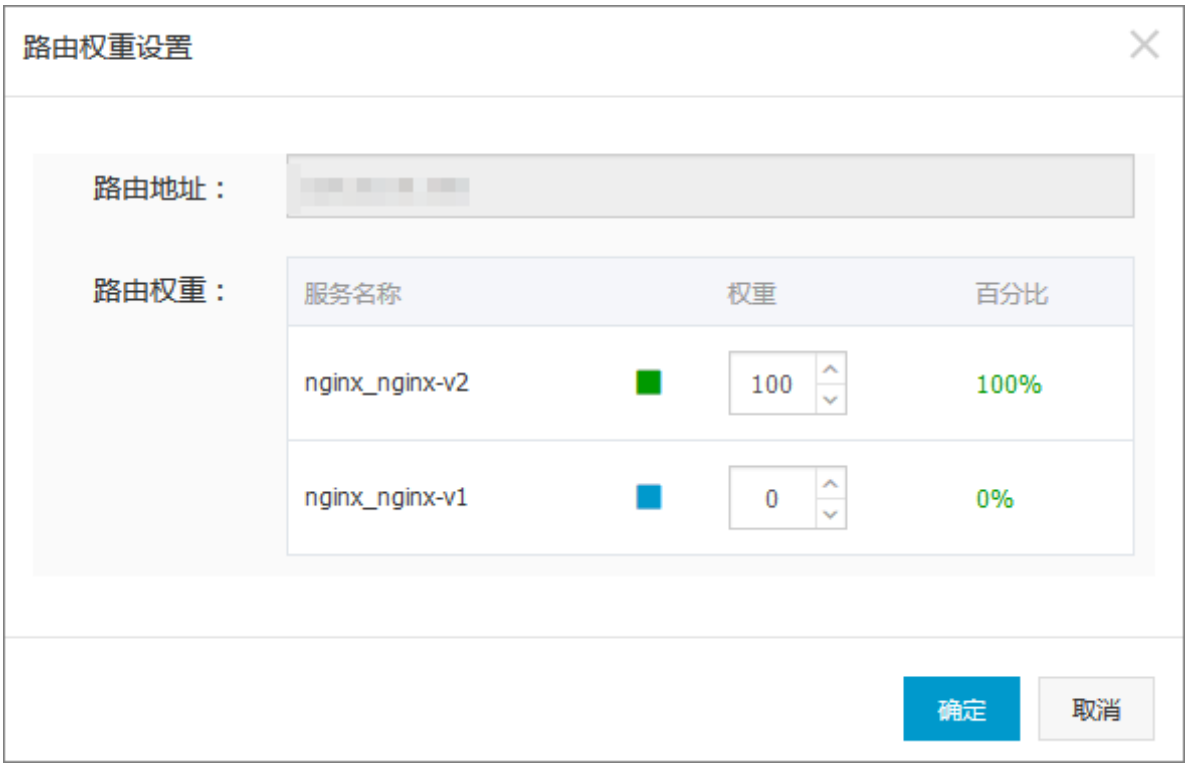
说明: 同时调整两个服务的权重，可能会导致部分请求失败，所以权重需要分为两步进行调整，一步只调整一个服务的权重。例如，先将新服务权重值从 0 调整为 100，待流量稳定后，再将然旧服务权重值从 100 设置为 0。

图 188: 设置路由权重



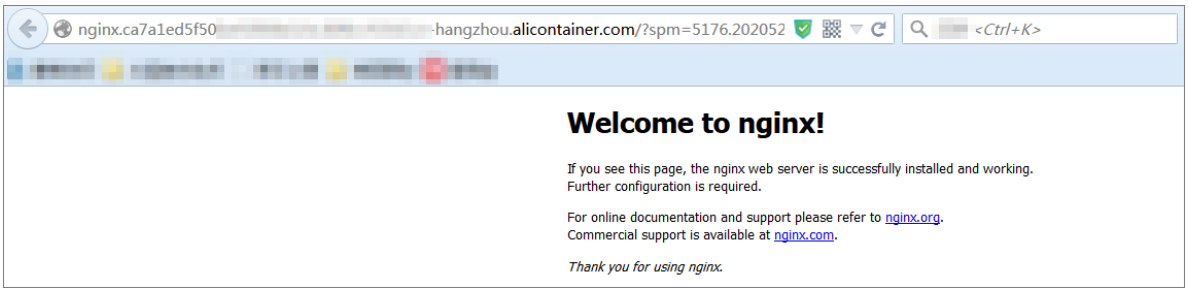
下一步，将旧版本服务的权重调整为 0，新版本服务的权重调整为 100。

图 189: 设置路由权重



8. 可以打开一个新的浏览器窗口，访问新的版本，结果如下。

图 190: 访问Nginx



9. 当整个发布流程验证完毕后，在应用列表页面，单击 **确认发布完成** 并在弹出的确认对话框中单击 **确认** 进行发布确认，方可进行下一次发布。

图 191: 确认发布完成

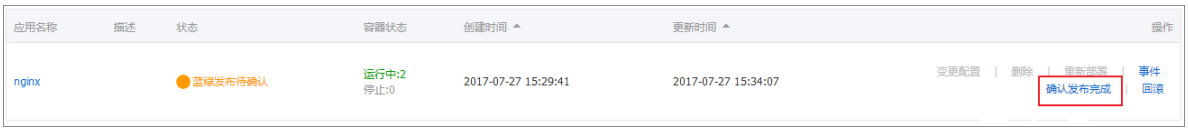


图 192: 发布确认



您可以看到应用的服务列表已经更新了，旧的服务已经完全下线删除了。

图 193: 发布完成



12.5 swarm mode集群指南

12.5.1 管理集群

12.5.1.1 创建集群

您可以创建一个零节点的集群，之后再绑定其他云服务器。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**集群**，单击右上角的**创建集群**。

3. 设置集群的基本信息。

   - **集群名称**：要创建的集群的名称。可以包含 1~64 个字符，包括数字，中文字符，英文字符和连字符（-）。

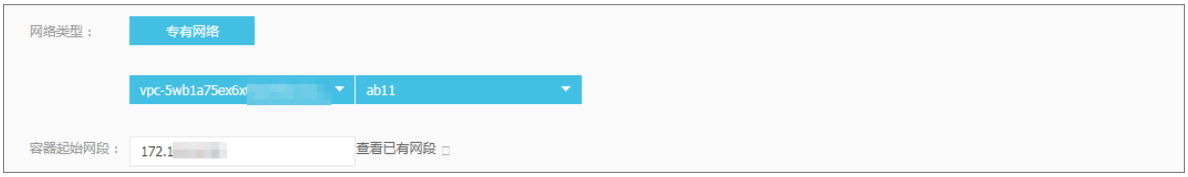
     说明: 集群名称在同一个用户下必须唯一。

   - **地域**：所创建集群将要部署到的地域。

- **可用区**：集群的可用区。

4. 选择集群类型为**Swarm Mode集群**。
5. 集群的网络类型为 VPC，您需要选择一个可用的VPC和VSwitch，并设置容器的起始网段。

图 194: 集群网络



6. 添加节点。

   此项不可选，您只能创建一个零节点集群，并在集群创建成功后再添加已有云服务器。有关如何添加已有云服务器，参见添加已有云服务器。

7. 创建一个负载均衡实例。

   勾选后，系统会在创建集群时创建一个负载均衡实例。您可以通过这个负载均衡实例访问集群内的容器应用。您也可以不随集群常见负载均衡，等集群创建完成后再绑定负载均衡，参见为集群绑定负载均衡。

8. 单击**创建集群**。

   集群创建成功后，如果需要单独配置云服务器或负载均衡，可以去相应的控制台进行相关操作。

后续操作

您可以查看集群创建日志。在**集群列表**页面，选择所创建的集群并单击**查看日志**。

12.5.1.2 添加已有云服务器

您可以将已有的云服务器添加到指定集群。

前提条件

如果之前没有创建过集群，您可以先创建一个零节点的集群。有关创建集群的详细信息，参见创建集群。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。

3. 选择所需的集群，单击**添加已有节点**。

图 195: 添加已有节点



4. 手动添加ECS实例。

手动添加实例，您需要在云服务器上执行脚本。

a) 页面显示实例列表，选择一个ECS实例并单击**下一步**。

您一次只能添加一个实例。

图 196: 选择实例



b) 设置实例信息并单击**下一步**。

图 197: 设置实例信息



c) 页面显示专属这台云服务器的脚本命令。通过终端登录，复制命令并执行。

等待脚本执行成功，显示success。该云服务器即添加成功。

图 198: 脚本命令



图 199: 执行命令



12.5.1.3 为集群绑定负载均衡

您可以在创建集群时自动创建一个负载均衡实例，或者在集群创建完成后为集群绑定一个负载均衡实例。

容器服务集群支持绑定外网负载均衡实例，VPC 负载均衡实例和经典网络下的内网负载均衡实例

使用限制

- 您只能为集群绑定位于同一区域下的负载均衡实例。
- 不支持绑定跨账号的负载均衡实例。
- 不支持跨网络类型绑定，即不能将 VPC 负载均衡实例绑定到经典网络集群上；不支持将经典网络负载均衡实例绑定到 VPC 集群上。
- VPC 集群支持绑定外网负载均衡实例和 VPC 负载均衡实例。
- 经典网络集群支持绑定外网负载均衡实例和经典网络下的内网负载均衡实例
- 一个集群仅支持绑定一个负载均衡实例。
- 两个集群不能共用一个负载均衡实例。

前提条件

您已经在负载均衡云控制台创建了负载均衡实例并且为负载均衡实例配置了监听后端服务器的TCP 9080端口。

操作步骤

1. 登录容器服务云控制台。

有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。
3. 选择要清理的集群，单击**管理**。

图 200: 集群管理



4. 单击左侧导航栏中的**负载均衡** > **绑定SLB**。

图 201: 解绑负载均衡



12.5.1.4 重新绑定负载均衡

您可以根据您的需要更换集群绑定的负载均衡实例。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

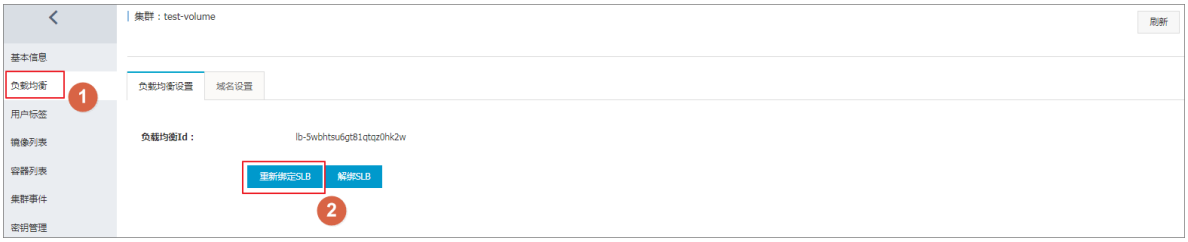
2. 单击左侧导航栏中的**集群**。
3. 选择要清理的集群，单击**管理**。

图 202: 集群管理



4. 单击左侧导航栏中的**负载均衡 > 重新绑定SLB。**

图 203: 重新绑定



5. 从下拉列表中选择您要绑定的负载均衡实例并单击**确定。**

> **说明:** 如果您所选择的负载均衡实例已经绑定了后端服务器，系统会提示您**该负载均衡实例已经绑定了后端服务器**，您需要选择其它未绑定后端服务器的负载均衡实例。

12.5.1.5 为集群解绑负载均衡

如果您不再需要负载均衡实例，您可以通过容器服务云控制台进行解绑。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群。**

3. 选择要清理的集群，单击**管理。**

图 204: 集群管理



4. 单击左侧导航栏中的**负载均衡 > 解绑SLB**。

图 205: 解绑负载均衡



12.5.1.6 搜索集群

您可以搜索您已经创建的集群。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。

3. 在搜索框中输入要搜索的集群的名称或关键字。名称中带有该关键字的集群将显示在集群列表中。

   说明: 搜索不区分大小写。

图 206: 搜索集群



12.5.1.7 删除集群

您可以从容器服务中删除集群。

背景信息

说明: 删除集群时，会将关联的负载均衡一起删除。请谨慎使用该操作。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。

3. 选择要删除的集群并单击**删除**。

   图 207: 删除集群



4. 在弹出的对话框中，单击**确定**。

12.5.1.8 清理集群磁盘

清理磁盘操作会清理用户集群内每台服务器上的脏数据。

背景信息

脏数据限于：

- 已下载到本地但未使用的Docker镜像。
- 曾经挂载到容器，但容器销毁后未清理的数据卷（volumn）目录。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。

3. 选择要清理的集群，单击**管理**。

图 208: 集群管理



4. 在集群管理页面中，单击**清理磁盘**。

图 209: 清理磁盘



12.5.1.9 连接集群

您可以使用下载的证书通过Docker Swarm API或Docker Client连接集群暴露出来的Endpoint。

操作步骤

1. 获取访问地址。

   a) 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

   b) 单击左侧导航栏中的**集群**，在集群列表中选择一个集群并单击 **管理**。

   您可以查看集群的连接信息。

   图 210: 集群连接信息



2. 下载和保存证书。

   要通过上面的服务地址访问Docker集群，您还需要配置TLS证书。

   在集群管理页面，单击**下载证书**开始下载证书。下载到的文件为*certFile.zip*。在下面的例子中，下载的证书存放在*~/.acs/certs/cluster_name/*目录下。其中，*cluster_name*是您集群的名字。您也可以使用其他目录，但是为了便于管理，推荐您将文件存放在*~/.acs/certs/cluster_name/*目录下。

   ```
   mkdir ~/.acs/certs/cluster_name/  #替换成真正的集群名字
   cd ~/.acs/certs/cluster_name/
   cp /path/to/certFile.zip .
   unzip certFile.zip
   ```

   在这个目录里，您可以看到三个文件 *ca.pem*，*cert.pem* 和 *key.pem*。

3. 管理集群。

   - **使用Docker Client管理集群**

     您可以使用Docker Client访问容器服务上的容器集群。访问集群之前，您需要使用以下两种方法之一配置使用的证书和服务地址。

     - 在命令行参数里配置证书

       ```
       docker --tlsverify --tlscacert=~/.acs/certs/cluster_name/ca.pem --tlscert=~/.acs/certs/ClusterName/cert.pem --tlskey=~/.acs/certs/ClusterName/key.pem \
       -H=集群接入点地址 ps
       ```

       把*cluster_name*和集群接入点地址替换成您实际使用的路径和访问地址。

     - 使用环境变量

       ```
       export DOCKER_TLS_VERIFY="1"
       export DOCKER_HOST="集群接入点地址"
       export DOCKER_CERT_PATH=~/.acs/certs/cluster_name
       ```

```
docker ps
```

把集群接入点地址替换成您实际使用的访问地址，把*cluster_name*替换成您实际使用的路径。

上面的两个例子在集群上执行了docker ps命令，您可以把ps替换成任何其他Docker支持的命令，比如使用docker run启动一个新的容器。

- **使用Docker Compose管理集群**

  目前Docker Compose支持通过环境变量声明服务地址和证书。

```
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="集群接入点地址"
export DOCKER_CERT_PATH=~/.acs/certs/cluster_name
docker-compose up
```

如果在使用过程中不慎导致证书泄露，您需要尽快吊销证书。在集群管理页面，单击**吊销已下载证书**可以吊销您之前下载的证书。吊销之后，您可以重新下载证书。



说明: 单击**吊销已下载证书**将导致您之前下载的证书不可用。

12.5.1.10 升级系统服务

在创建集群时，容器服务会自动为集群创建系统服务。您可以通过容器服务云控制台升级系统服务。

集群的系统服务用来解决应用需要的通用服务，例如日志服务acslogging、路由服务acsrouting、volume服务acsvolumedriver和监控服务acsmonitoring。



说明: 集群的系统服务在升级的过程中会导致您的应用或者服务短暂不可访问或者不能正常工作。请谨慎升级。建议选择访问低谷或者维护时间进行升级。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。
3. 选择所需的集群，单击**管理**。

图 211: 集群管理



4. 单击**升级系统服务**。
5. 在弹出的对话框中，选择要升级的系统服务并单击**升级**。

   例如，本示例中选择的是**简单路由服务**（对应acsrouting，注意升级会短暂影响用户应用的访问），**volume服务**（对应acsvolumedriver，注意升级可能会短暂影响用户相关联应用的功能）。

12.5.2 管理节点

12.5.2.1 移除节点

移除节点可以将服务器从集群中摘除，移除后将不能在节点信息内看到该服务器信息。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。
2. 单击左侧导航栏中的**节点**。
3. 选择要移除的节点所在的集群。
4. 选择要移除的节点，单击**更多** > **移除节点**。

图 212: 移除节点



5. 在弹出的确认对话框中，单击**确定**。

12.5.2.2 查看节点上运行的容器

您可以通过节点列表页面查看运行在某个节点上的容器。

操作步骤

1. 登录容器服务云控制台。

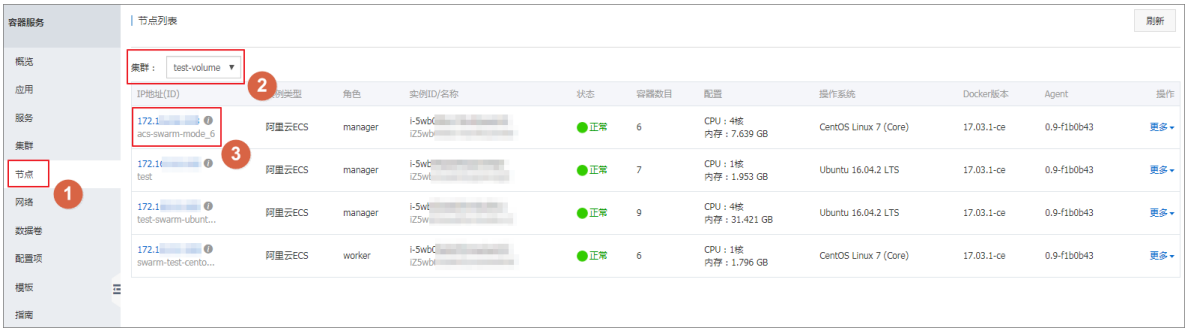
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**节点**。

3. 选择节点所在的集群。

4. 选择所需的节点，单击节点的ID。

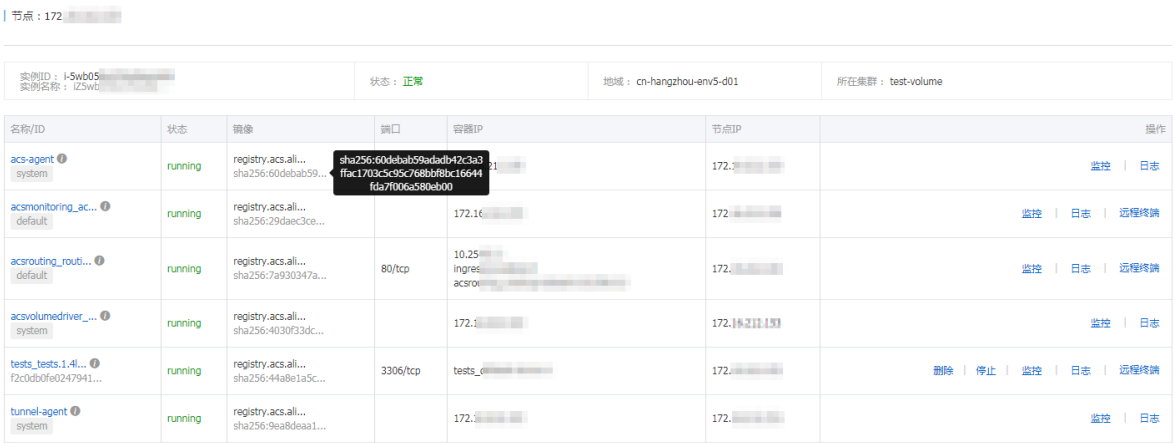
图 213: 查看节点



5. 您可以看到运行在该节点上的容器的列表。

   通过该列表，您可以查看容器的标签（Labels），查看容器的镜像并查看镜像的sha256，查看容器的日志和监控信息，并进行容器相关操作（包括：启停容器、删除容器、通过远程终端操作容器）。

图 214: 容器列表



12.5.2.3 设置为Worker

您可以通过容器服务云控制台将Manager节点设置为Worker。

在swarm mode集群管理过程中，可能存在一些情况，需要将集群中的Manager节点设置为Worker。

- 合理增加负载能力：合理分配Manager和Worker节点的数量占比有助于提高集群的工作负载能力，在小型的集群中，可以适当将多余的Manager节点设置为Worker。
- 集群维护计划： Manager节点下线维护后重新上线后，原Worker从Manager重新设置为Worker节点。

集群在进行扩容时，在Manager节点数量≥3个的情况下，新增节点会在加入后一段时间内自动被设置为Worker。

使用限制

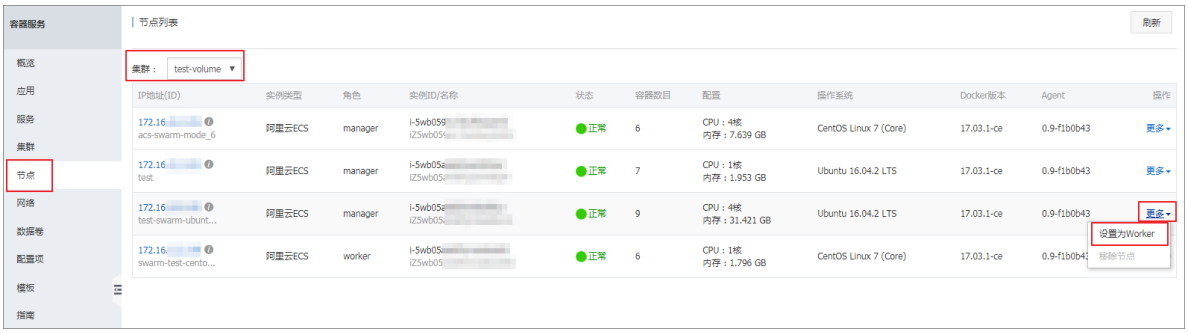
当集群中仅有一个Manager节点时，您无法将该Manager节点设置为Worker。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。
2. 单击左侧导航栏中的**节点**。
3. 选择节点所在的集群。
4. 选择所需的节点，并单击右侧的**更多** > **设置为Worker**。

   图 215: 设置为Worker



5. 在弹出的确认对话框中，单击**确定**。

12.5.2.4 设置为Manager

您可以通过容器服务云控制台将Worker节点设置为Manager。

您在集群管理过程中，可能有以下几种情况，需要将节点设置为Manager。

- 集群维护：您可能需要将某些Manager节点下线进行维护，可以在之前将某个Worker节点设置为Manager，从而保证集群正常运行，不中断业务。
- 资源配置优化：您可能想将一台资源配置更好的ECS服务器设置为Manager，来实现资源配置优化。
- 构建一个更高可用性的集群架构：为了提高集群的容错率，您可能需要对旧集群进行升级，将Worker节点设置为Manager，构建一个拥有更多Manager（ 2n+1个Manager，n≥1 ） 的高可用集群。例如将一个3 Manager的集群扩容至5 Manager的集群，就需要将新加入的2个节点手动设为Manager。

默认情况下，所有节点处于ACTIVE可用性下，这种情况下，Manager节点同样会被分配任务，出于集群业务管理、维护的角度，您可能想让 Manager节点只承担管理调度的任务。此时，可以将该节点设置为Drain，让其成为 “专属管理容器” 。具体如何设置，参见 *Drain a node on the swarm*。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**节点**。
3. 选择节点所在的集群。
4. 选择所需的节点，并单击右侧的**更多** > **设置为Manager**。

   图 216: 设置为Manager



5. 在弹出的确认对话框中，单击**确定**。

12.5.3 安全组

12.5.3.1 容器服务安全组规则

容器服务集群安全组开放的规则如下。

VPC安全组

- 为方便用户业务的 Web 服务，默认开启443 端口和 80 端口，可以根据自己的需求选择放开或者关闭。
- ICMP 规则建议保留，用于节点间通信，方便排查问题。有些工具也依赖 ICMP。
- 容器服务依赖 22 端口对机器初始化，方便 SSH 到用户机器进行集群的配置。
- VPC 安全组授权对象设置的是容器网段的基础地址，本示例是 172.18.0.0/16地址段，与您在创建 VPC 集群时选择的容器初始网段有关，也可以选择其他网段，参见创建集群。保证容器之间可以互相通信。

安全配置原则

- 每个集群一个安全组。

  容器服务每个集群都管理了一个安全组。您可以在这个安全组上配置规则。
- 最小权限原则。

  为了您集群的安全性，安全组应该对外开放最小的权限。
- 容器服务创建的安全组添加了一些默认规则。

  为了方便用户操作ECS实例，容器服务创建的安全组添加了一些默认规则，开放了诸如80/443等端口。如果不需要，您可以删除这些规则。

  说明: 不要屏蔽22和2376端口。容器服务需要通过这两个端口初始化机器。
- 尽量使用容器内部网络进行通信，不将通信暴露到宿主机上。
- 授权其他ECS实例访问安全组时，授权给安全组，而非单个IP。

  要授权其他ECS实例访问当前安全组，先创建一个新安全组，把要访问当前安全组的ECS实例加入新安全组，再授权新安全组访问当前安全组。
- 优先使用VPC网络。如非必要，节点不要绑定EIP， VPC网络的隔离性更好。
- VPC内网出/入方向里要放开容器的网段。

  如果不放开，会导致容器之间网络不通。

12.5.3.2 检测安全组

您可以通过容器服务云控制台检测集群的安全组规则是否安全。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。

3. 选择要目标集群，单击**管理**。

   图 217: 集群管理



4. 单击**检测安全组**。系统会检测集群安全组的状态。

   如果安全组规则正常，系统会显示如下提示框，单击**取消**即可。



   若果您的安全组规则存在安全风险，系统会弹出提示框列出存在安全风险的安全组规则。单击**修复**，系统会删除存在安全风险的安全组规则并创建新的安全组规则

12.5.3.3 重新绑定安全组

您可以通过容器服务云控制台重新为集群绑定安全组，系统会重新创建一个安全组并将新创建的安全组绑定到集群，同时将集群中的节点添加到新创建的安全组。

操作步骤

1. 登录容器服务云控制台。

有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。
3. 选择目标集群，单击**管理**。

图 218: 集群管理



4. 单击**重新绑定安全组**。
5. 在弹出的对话框中单击**确定**。

12.5.4 管理编排模板

您可以通过容器服务云控制台创建并查看编排模板。

12.5.4.1 创建编排模板

操作步骤

1. 登录容器服务云控制台。
2. 单击左侧导航栏中的**模板**。
3. 单击右上角的**创建**。
4. 在**创建编排**页面，填写模板信息。
   - **名称**：模板的名称。
   - **描述**：模板的相关信息。
   - **内容**：Docker Compose的yml文件。有关文件的详细信息，参见 *Compose File*的详细说明。

     编排模板中包含的服务会显示在页面的右侧。您可以单击**编辑**，在弹出的**新建服务**对话框中通过参数修改编排模板，或者单击**删除**，删除所选服务。
5. 单击**创建编排**。

   您可以在**编排列表**页面中**我的编排**下查看所创建的编排模板。您可以单击**详情**，查看编排模板的详细信息，或者单击**创建应用**，使用该编排模板创建应用。

12.5.4.2 查看编排模板列表

您可以通过容器服务云控制台查看您所创建的编排模板。

操作步骤

1. 登录容器服务云控制台。
2. 单击左侧导航栏中的**模板**。

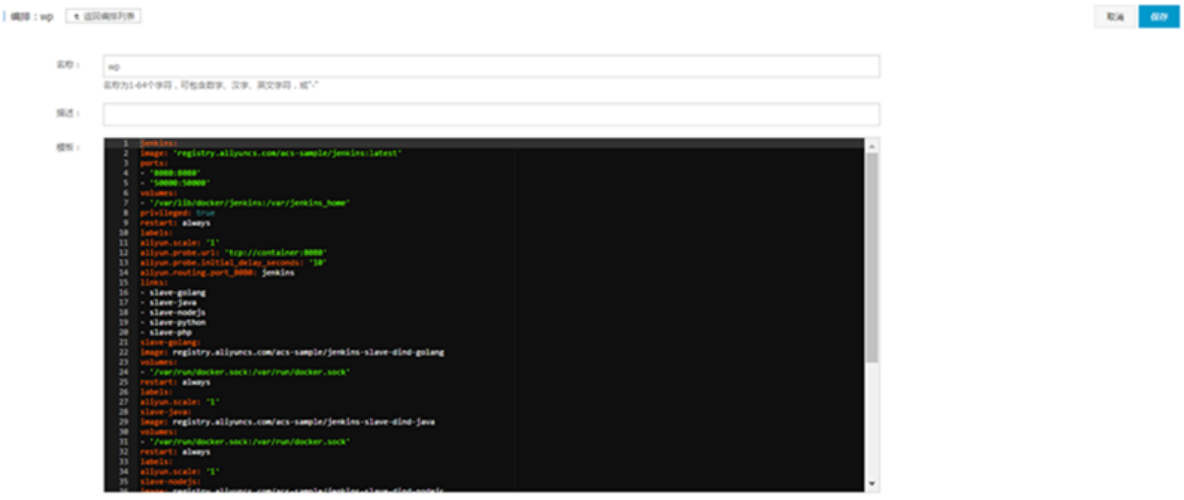
   您可以单击**搜索**通过模板的名称前缀搜索模板。

12.5.4.3 更新编排模板

您可以通过容器服务云控制台修改编排模板。

操作步骤

1. 登录容器服务云控制台。
2. 单击左侧导航栏中的**模板**。
3. 选择一个模板并单击**详情**。
4. 单击右上角的**编辑**。
5. 编辑模板内容。



6. 单击右上角的**保存**保存模板。

12.5.4.4 下载编排模板

您可以通过容器服务云控制台将编排模板下载到本地。

操作步骤

1. 登录容器服务云控制台。
2. 单击左侧导航栏中的**模板**。

3. 选择一个模板并单击 **详情**。
4. 单击右上角的**下载**，会立即下载后缀为*yml*格式的模板文件。



12.5.4.5 删除编排模板

您可以通过容器服务云控制台删除不再使用的编排模板。

操作步骤

1. 登录容器服务云控制台。
2. 单击左侧导航栏中的**模板**。
3. 选择一个模板并单击**详情**。
4. 单击右上角的**删除**。



5. 在弹出的确认对话框中，单击**确定**。

12.5.4.6 另存编排模板

您可以将编排模板另存为一个新的编排模板。

操作步骤

1. 登录容器服务云控制台。
2. 单击左侧导航栏中的**模板**。
3. 选择一个模板并单击**详情**。

4. 单击右上角的**另存为**。
5. 填写新的编排模板名称并单击**确定**。

   另存的编排模板会显示在**编排列表**页面中**我的编排**下。

12.5.5 服务编排

容器服务支持Docker Compose编排模板来描述多容器应用。

编排模板允许您描述一个完整的应用，该应用可以由许多个服务组成。例如：一个门户网站应用，由一个Nginx服务、一个Web服务和一个数据库服务组成。

一个服务可能会有多个容器实例，所有容器实例的配置保持一致。例如：上述应用中的Web服务，就可以根据访问量需要启动两个甚至更多的容器。

能力

容器服务支持通过编排模板文件，自动化地部署和管理一个容器应用。

swarm mode集群支持Compose V1/V2和Compose V3版本的模板格式。

容器服务也在社区Compose模板描述的基础上提供了一系列扩展来简化Web、微服务应用的部署和运维。更多详细信息，参见 标签概览。

示例

下面是一个WordPress应用，包含了由WordPress镜像提供的Web服务和MySQL镜像提供的db服务。

Compose V1/V2模板示例：

```
web:
  image: wordpress:4.2
  ports:
    - "80"
  environment:
    - WORDPRESS_AUTH_KEY=changeme
    - WORDPRESS_SECURE_AUTH_KEY=changeme
    - WORDPRESS_LOGGED_IN_KEY=changeme
    - WORDPRESS_NONCE_KEY=changeme
    - WORDPRESS_AUTH_SALT=changeme
    - WORDPRESS_SECURE_AUTH_SALT=changeme
    - WORDPRESS_LOGGED_IN_SALT=changeme
    - WORDPRESS_NONCE_SALT=changeme
  restart: always
  links:
    - db:mysql
  labels:
    aliyun.log_store_wordpress: stdout
    aliyun.probe.url: http://container/license.txt
    aliyun.probe.initial_delay_seconds: "10"
```

```
    aliyun.routing.port_80: wordpress;http://www.example.com;https://www.nice.com
    aliyun.scale: "3"
db:
  image: mysql:5.6
  environment:
    MYSQL_ROOT_PASSWORD: password
  restart: always
  labels:
    aliyun.log_store_mysql: stdout
```

Compose V3模板示例：

```
    version: '3'
    services:
      web:
        image: wordpress:4
        environment:
          - WORDPRESS_DB_PASSWORD=password
          - WORDPRESS_AUTH_KEY=changeme
          - WORDPRESS_SECURE_AUTH_KEY=changeme
          - WORDPRESS_LOGGED_IN_KEY=changeme
          - WORDPRESS_NONCE_KEY=changeme
          - WORDPRESS_AUTH_SALT=changeme
          - WORDPRESS_SECURE_AUTH_SALT=changeme
          - WORDPRESS_LOGGED_IN_SALT=changeme
          - WORDPRESS_NONCE_SALT=changeme
          - WORDPRESS_NONCE_AA=changeme
        ports:
          - 80
        depends_on:
          - mysql
        deploy:
          replicas: 3
          restart_policy:
            condition: on-failure
        labels:
          aliyun.routing.port_80: wordpress
      mysql:
        image: mysql:5.7
        environment:
          - MYSQL_ROOT_PASSWORD=password
        deploy:
          restart_policy:
```

```
        condition: on-failure
```

12.5.5.1 标签概览

swarm mode集群支持Compose V1/V2和Compose V3版本的编排模板。Compose V1/V2和Compose V3版本的编排模板对于阿里云扩展标签的支持情况不同。

表 24: 支持阿里云扩展能力标签的情况

功能	Compose V1/V2	Compose V3
服务的健康检查	*aliyun.probe*标签	Compose中的*healthcheck* 一节。该功能由Docker Engine原生提供。
滚动更新（rolling update）	*aliyun.rolling_updates*标签、*aliyun.rolling_updates.parallelism*标签	Compose中的*update_config*一节。该功能由swarm mode原生提供，可设置更新的批量、间隔时间、失败后的动作等。
服务依赖	*aliyun.depends*标签	Compose中的*depends_on*一节。服务会按照依赖关系顺序启动，默认等待3分钟，如果服务3分钟后依然没有达到运行状态，会忽略并继续部署其他服务，但最终部署结果是失败。
服务的容器数量	*aliyun.scale*标签	compose中的*replicas*一节。该功能对应于swarm mode的Replicated Service，当容器故障时，会自动创建新容器，始终保证replicas等于指定的数量。
全局服务	*aliyun.global*标签	compose中的*mode*一节。该功能对应于swarm mode的Global Service，当新节点加入集群里，会自动创建相应容器。
负载均衡	*aliyun.routing*、*aliyun.lb*标签	参见路由和负载均衡。
日志	*aliyun.log*标签	没有变化。参见*log 标签*。
数据卷	compose的*volumes*一节	使用方法没有变化。但使用本地数据卷时，如果主机文件夹

功能	Compose V1/V2	Compose V3
		不存在，不会自动创建，需要您手工创建。
监控与自动扩缩容	*aliyun.auto_scaling*、*aliyun.rescheduling*标签	没有支持。
外部服务	compose的external一节。	不支持。
容器重新调度	环境变量中的*reschedule:on-node-failure*。	默认提供，不需要设置。
高可用性调度	环境变量中的*availability:az*。	暂时不支持。
自定义网络	Compose V2中的网络允许服务的容器启动在自定义的网络中，容器服务的容器都是在同一个跨主机互通的容器网络，所以不支持您在 Compose V2中使用networks标签。关于容器服务的网络管理和服务发现，参见跨主机互联的容器网络。	支持使用networks标签将应用添加到自定义网络中。参见*networks*。

不支持的Docker Compose标签

容器服务暂不支持Docker Compose的部分标签。有关容器服务暂不支持的标签，参见 不支持的*Docker Compose*标签。

12.5.5.2 健康检查

设置服务的健康性检查。

模板版本支持

表 25: 模板版本支持

Compose V1/V2	Compose V3
*aliyun.probe*标签	Compose中的*healthcheck* 一节。该功能由Docker Engine原生提供。

Compose V1/V2

设置服务的健康性检查

- 通过URL进行检查，支持HTTP协议、TCP协议。
- 通过shell脚本检查。

健康检查会从容器宿主机上发起，每隔一定时间（默认两秒）向容器发起请求或在容器上执行shell脚本命令。

检查成功的判断条件

- HTTP请求的返回码为2XX/3XX；TCP端口可建立连接。
- shell脚本运行返回值为0。

检查的字段说明

- aliyun.probe.url：HTTP、TCP请求的URL。您不需要填写自己的域名或者IP地址，只需要加上container这个单词，该URL最终会被解析成容器相应的IP去进行健康检查，检查结果返回2XX或者3XX才认为服务是健康的。
  - 例如，容器通过8080端口提供HTTP服务，并提供了/ping作为健康检查的URL，则探测URL的格式为http://container:8080/ping，容器服务会自动通过HTTP GET请求检查URL的返回结果，如果返回结果的返回码为2XX或者3XX，则说明健康检查成功。
  - 例如，MySQL容器侦听3306端口，探测URL的格式为tcp://container:3306，服务会检查容器3306端口是否打开，如果打开则说明健康检查成功。
- aliyun.probe.cmd：健康检查执行的检查shell命令，/check.sh；容器服务会定期在容器内执行该命令，当shell脚本返回值为0时表明健康检查成功。
- aliyun.probe.timeout_seconds：健康检查的超时时间。
- aliyun.probe.initial_delay_seconds：在容器启动后延迟几秒开始健康检查。



说明:

- 一个服务中只能包含aliyun.probe.url和aliyun.probe.cmd其中之一。
- 如果服务不包含aliyun.probe.url或aliyun.probe.cmd，则容器缺省为健康状态，且其他aliyun.probe.xxx标签会被忽略。

编排示例

利用URL检测容器健康状态。

```
os:
  image: my_nginx
  labels:
    aliyun.probe.url: http://container/ping
    aliyun.probe.timeout_seconds: "10"
```

```
    aliyun.probe.initial_delay_seconds: "3"
```

利用shell脚本检测容器健康状态。

```
os:
  image: my_app
  labels:
    aliyun.probe.cmd: health_check.sh
    aliyun.probe.initial_delay_seconds: "3"
```

Compose V3

设置服务的健康性检查。

healthcheck命令包含以下形式：

- test: ["CMD", "curl", "-f", "http://localhost"]：运行命令检查容器是否健康。
- test: ["CMD-SHELL", "curl -f http://localhost && echo 'cool, it works'"]：运行命令检查容器是否健康。
- test: curl -f https://localhost && echo 'cool, it works'：运行命令检查容器是否健康。
- test: ["NONE"]：禁用镜像中所有的健康检查设置。

  或者使用：

  ```
  healthcheck:
        disable: true
  ```

检查成功的判断条件

命令的退出状态显示了容器的健康状态。

- **0**: 成功。容器健康，可以使用。
- **1**: 不健康。容器运行不正常。
- **2**: 保留。不要使用该退出码。

检查的字段解释

- interval：健康检查的时间间隔，默认为30s。
- timeout：健康检查的超时时间，默认为30s。
- start-period：在容器启动后延迟几秒开始健康检查，默认为0s。
- retries：连续几次健康检查失败即认为容器不健康，默认为3。



说明: 一个编排模板中只能包含一个healthcheck设置。如果包含多个，系统仅会执行最后一个healthcheck设置。

编排模板

```
version: '3'
    services:
      web:
        image: wordpress:4
        healthcheck:
          test: ["CMD", "curl", "-f", "http://localhost"]
          interval: 1m30s
          timeout: 10s
          retries: 3
```

12.5.5.3 rolling_updates

设置服务如何更新。

模板版本支持

表 26: 模板版本支持

Compose V1/V2	Compose V3
*aliyun.rolling_updates*标签、*aliyun.rolling_updates.parallelism*标签	Compose中的*update_config*一节。该功能由swarm mode原生提供，可设置更新的批量、间隔时间、失败后的动作等。

Compose V1/V2

rolling_updates

更新某个服务时，如果该服务包括超过一个以上容器（使用scale标签定义），在第n个容器更新成功后，再去做第n+1个容器的更新，以此来最小化停止服务时间。

部署WordPress服务，通过scale标签指定部署2个容器，使用rolling_updates标签可以使WordPress对外停止服务的时间最小化。

```
web:
  image: wordpress
  ports:
    - 80
  restart: always
  links:
    - 'db:mysql'
  labels:
    aliyun.logs: /var/log
    aliyun.probe.url: http://container/license.txt
    aliyun.probe.initial_delay_seconds: '10'
    aliyun.routing.port_80: http://wordpress
    aliyun.rolling_updates: 'true'
    aliyun.scale: '2'
db:
  image: mariadb
  environment:
```

```
    MYSQL_ROOT_PASSWORD: example
  restart: always
  labels:
    aliyun.logs: /var/log/mysql
```

parallelism

您可以使用parallelism标签定义rolling_updates每次并行更新的容器数量。



说明: 此标签必须和rolling_updates配合使用，单独使用无效。

取值：

- 默认值为1，即每次只更新一个容器。
- 当其值大于1的时候，rolling_updates过程中，每次会以parallelism定义的值来并行更新相应个数的容器，实现批量更新。
- 当定义值无效时，默认为1。



说明: 为了确保始终有容器在提供服务，建议parallelism定义的值小于服务包含的容器数。

编排示例

下面的示例部署Nginx服务，通过scale标签部署 3 个容器，使用rolling_updates和parallelism标签定义每次以 2 个容器为单位来进行批量更新。

```
web:
  image: nginx:latest
  restart: always
  environment:
      - "reschedule:on-node-failure"
  ports:
      - 80
  labels:
    aliyun.scale: "3"
    aliyun.probe.url: http://container:80/
    aliyun.probe.timeout_seconds: "10"
    aliyun.probe.initial_delay_seconds: "3"
    aliyun.rolling_updates: 'true'
    aliyun.rolling_updates.parallelism: "2"
```

Compose V3

设置如何更新服务。

字段解释：

- parallelism：每次并行更新的容器数量。
- delay：更新不同组容器之间的时间间隔。

- failure_action：某一更新失败时，系统如何操作。可以为continue （失败后，继续更新下一批容器）或者 pause（失败后，停止更新服务），默认为pause。
- monitor：更新任务完成后监控多长时间来检测更新是否失败。单位可以为ns、us、ms、s、m、h。默认为0s，更新完成后立即检查更新是否失败。
- max_failure_ratio：每次更新允许的失败占比。

编排示例：

```
version: '3'
    services:
      vote:
        image: wordpress:4
        depends_on:
          - mysql
        deploy:
          replicas: 2
          update_config:
            parallelism: 2
            delay: 10s
```

12.5.5.4 depends

设置服务的依赖关系。

模板版本支持

表 27: 模板版本支持

Compose V1/V2	Compose V3
*aliyun.depends*标签	Compose中的*depends_on*一节。服务会按照依赖关系顺序启动，默认等待3分钟，如果服务3分钟后依然没有达到运行状态，会忽略并继续部署其他服务，但最终部署结果是失败。

Compose V1/V2

设置服务的依赖关系。

设置之后，容器服务可以控制容器的启动顺序，逐个启动容器。

编排示例：

说明: 多个依赖使用逗号（,）分隔。

```
web:
  image: wordpress:4.2
  ports:
```

```
    - 80
  links:
    - db:mysql
  labels:
    aliyun.depends: db,redis
db:
  image: mysql
  environment:
    - MYSQL_ROOT_PASSWORD=password
redis:
  image: redis
```

Compose V3

设置服务的依赖关系。

服务会按照依赖关系顺序启动，默认等待3分钟，如果服务3分钟后依然没有达到运行状态，会忽略并继续部署其他服务，但最终部署结果是失败。

编排示例：

```
version: '3'
    services:
      web:
        build: .
        depends_on:
          - db
          - redis
      redis:
        image: redis
      db:
        image: postgres
```

12.5.5.5 服务的容器数量

设置该服务的容器数量，横向扩展服务。

模板版本支持

表 28: 模板版本支持

Compose V1/V2	Compose V3
*aliyun.scale*标签	compose中的*replicas*一节。该功能对应于swarm mode的Replicated Service，当容器故障时，会自动创建新容器，始终保证replicas等于指定的数量。

Compose V1/V2

设置该服务的容器数量，横向扩展服务。

在Compose V1/V2中，Docker Compose只能在每一个服务中启动一个容器，如果需要扩展容器数量，需要在启动后手动进行设置。

现在通过scale的扩展标签，支持您在容器启动的时候进行扩展。

此外，在容器被删除之后，您可以在容器服务云控制台重新部署应用，容器服务会重启或新建容器使容器恢复到指定数量。

编排示例：

```
web:
  image: wordpress:4.2
  ports:
    - 80
  links:
    - db:mysql
  labels:
    aliyun.scale: "3"
db:
  image: mysql
  environment:
    - MYSQL_ROOT_PASSWORD=password
```

Compose V3

该功能对应于swarm mode的Replicated Service（参见*mode*），当容器故障时，会自动创建新容器，始终保证replicas等于指定的数量。

编排示例：

```
version: '3'
    services:
      worker:
        image: dockersamples/examplevotingapp_worker
        networks:
          - frontend
          - backend
        deploy:
          mode: replicated
          replicas: 6
```

mode

可以设置为global（每个swarm mode一个容器）或者replicated（指定数量的容器）。默认为replicated。

```
version: '3'
    services:
      worker:
        image: dockersamples/examplevotingapp_worker
        deploy:
```

```
        mode: global
```

12.5.5.6 routing标签

设置该服务的访问域名。

模板版本支持

Compose V1/V2和Compose V3版本的编排模板均支持routing和session sticky标签，且语法相同。

格式：

aliyun.routing.port_*$contaienr_port*: [http://]*$domain*|*$domain_prefix[:$context_path]*

基本概念：

- $contaienr_port：容器端口，注意该处不是主机的端口。
- $domain：域名，需要用户填写自己的域名。
- $domain_prefix：域名前缀，如果填写域名前缀，容器服务会提供给您一个测试用的域名，域名后缀是wordpress.<*cluster_id*>.<*region_id*>.alicontainer.com。
- $context_path：请求的路径，即可以根据请求的路径来选择区分不同的后端容器。

绑定域名的选择：

- 如果使用HTTP协议暴露服务，可以使用容器服务提供内部域名（顶级域为alicontainer.com），供您测试使用，也可以使用您提供的域名。
- 如果使用HTTPS协议，那么仅支持配置您提供的域名，例如www.example.com。您需要修改DNS设置将域名指定到容器集群提供的负载均衡服务上。

标签声明的格式要求：

- 容器服务为每一个集群分配了子域名，绑定内部域名只需要给出域名的前缀，域名前缀仅表示域名的一级，不能使用点号（.）进行分隔。
- 如果您不指定scheme，则默认使用HTTP协议。
- 域名的长度不能超过128个字符，context root的长度不能超过128个字符。
- 绑定多个域名到服务时，域名之间用分号（;）隔开。
- 一个后端服务可以有多个端口，该端口指的是容器暴露的端口，一个端口只能使用一条label进行声明，带有多个端口的服务需要声明多个label。

编排示例：

使用routing标签将容器服务提供的内部域名wordpress.*cluster_id*.*region_id*.alicontainer.com绑定到Web服务的80端口，并且将您提供的自有域名http://wp.sample.com/context绑定到Web服务的80端口。

Compose V1/V2编排模板：

```
web:
  image: wordpress:4.2
  links:
    - db:mysql
  labels:
    aliyun.routing.port_80: wordpress;http://wp.sample.com/context
db:
  image: mysql
  environment:
    - MYSQL_ROOT_PASSWORD=password
```

Compose V3编排模板：

```
version: '3'
    services:
      web:
        image: wordpress:4
        depends_on:
          - mysql
        deploy:
          replicas: 3
          restart_policy:
            condition: on-failure
        labels:
          aliyun.routing.port_80: wordpress;http://wp.sample.com/context
      db:
        image: mysql
        environment:
          - MYSQL_ROOT_PASSWORD=password
        deploy:
          restart_policy:
            condition: on-failure
```

最终您得到的内部域名为wordpress.*cluster_id*.cn-beijing.alicontainer.com。

Web服务运行之后，您可以通过http://wordpress.*cluster_id*.cn-beijing.alicontainer.com或者http://wp.sample.com/context访问相应的Web服务。

如果您需要支持HTTPS服务，需要自行通过负载均衡云控制台上传HTTPS证书，并绑定相应的集群对外访问负载均衡端点。

routing.session_sticky

设置routing在做请求路由的时候，是否保持session sticky，即会话保持。其效果是，在某个会话时间内，请求一直路由到同一个后端的容器，而不是每次请求都随机路由到不同的容器。

说明: 只有当您已经设置了aliyun.routing.port_$contaienr_port时，该设置才能起作用。

其设置方法如下：

- 开启会话保持

  aliyun.routing.session_sticky: true

- 关闭会话保持

  aliyun.routing.session_sticky: false

编排示例：

Compose V1/V2编排模板：

```
web:
  image: wordpress:4.2
  links:
    - db:mysql
  labels:
    aliyun.routing.port_80: wordpress;http://wp.sample.com/context
    aliyun.routing.session_sticky: true
db:
  image: mysql
  environment:
    - MYSQL_ROOT_PASSWORD=password
```

Compose V3编排模板：

```
version: '3'
    services:
      web:
        image: wordpress:4
        depends_on:
          - mysql
        deploy:
          replicas: 3
          restart_policy:
            condition: on-failure
        labels:
          aliyun.routing.port_80: wordpress;http://wp.sample.com/context
          aliyun.routing.session_sticky: true
      db:
        image: mysql
        environment:
          - MYSQL_ROOT_PASSWORD=password
        deploy:
          restart_policy:
```

```
        condition: on-failure
```

12.5.5.7 lb

通过自定义阿里云负载均衡nat映射的方式来暴露服务端口到公网或者到内网。

模板版本支持

Compose V1/V2和Compose V3版本的编排模板均支持lb标签，且语法相同。

功能概述

标签格式如下，带$的变量为占位符。

```
aliyun.lb.port_$container_port:$scheme://$[slb_name|slb_id]:$slb_front_port
```

编排示例

Compose V1/V2编排示例：

```
web:
  image: wordpress:4.2
  ports:
    - 7777:80
    - 9999:9999
    - 8080:8080
    - 53:53/udp
  links:
    - db:mysql
  labels:
    aliyun.lb.port_80: http://slb_example_name:8080
    aliyun.lb.port_9999: tcp://slb_example_name:9999
    aliyun.lb.port_8080: https://14a7ba06d3b-cn-hangzhou-dg-a01:80
    aliyun.lb.port_53: udp://14a7ba06d3b-cn-hangzhou-dg-a01:53
db:
  image: mysql
  environment:
    - MYSQL_ROOT_PASSWORD=password
```

Compose V3编排示例：

```
version: '3'
    services:
      web:
        image: wordpress:4
        ports:
          - 7777:80
          - 9999:9999
          - 8080:8080
          - 53:53/udp
        depends_on:
          - mysql
        deploy:
          replicas: 3
          restart_policy:
            condition: on-failure
        labels:
```

```
      aliyun.lb.port_80: http://slb_example_name:8080
      aliyun.lb.port_9999: tcp://slb_example_name:9999
      aliyun.lb.port_8080: https://14a7ba06d3b-cn-hangzhou-dg-a01:80
      aliyun.lb.port_53: udp://14a7ba06d3b-cn-hangzhou-dg-a01:53
  db:
    image: mysql
    environment:
      - MYSQL_ROOT_PASSWORD=password
    deploy:
      restart_policy:
        condition: on-failure
```

要使用好自定义负载均衡的lb标签，您需要理解请求路由过程中的3个端口，即负载均衡的前端端口，负载均衡的后端端口（也就是ECS实例的端口），最后就是容器的端口。以第一个lb标签aliyun.lb.port_80为例，从左往右看，在key中的80端口指的是容器要暴露的端口，后面的8080端口指的是负载均衡要暴露的前端端口。负载均衡的后端端口是ECS实例的端口，可从标签ports的主机：容器端口映射中获取到，由此，您可以查到容器端口80对应的主机端口是7777，因此确定了负载均衡转发的后端端口是7777端口。因此第一个标签说明了当向服务Web发起请求时，首先通过负载均衡前端的8080端口进入，转发到后端主机的7777端口，然后再根据端口映射ports的声明，请求最终从容器端口80进入，交由容器内的WordPress进程提供服务。接下来的标签以此进行相同的解释。该标签配置的负载均衡均不经过集群内置的routing服务，请求的路由由您自己控制。

标签声明的格式要求：

- 指明负载均衡实例时，可以使用负载均衡实例的名称或者负载均衡实例的ID。
- 负载均衡实例名称的限制为1~80个字符，允许包含字母、数字、连字符（-）、正斜杠（/）、点号（.）、下划线（_）。
- 容器端口限制为1~65535。
- 负载均衡前端端口的限制为1~65535。

带有自定义负载均衡nat映射的服务部署限制：

- 您需要自己创建负载均衡实例，对负载均衡实例命名，并创建对应监听端口，然后以扩展标签的方式提供负载均衡实例的名称$slb_name或者$slb_id，以及要暴露的端口，使用的协议$scheme（可能的值有tcp、http、https、udp），映射的容器端口$container_port，指定负载均衡实例的前端端口$slb_front_port。
- 您必须指定服务要暴露端口的主机和容器端口的映射，通过Dockerfile标准的标签ports指定，注意必须指定主机端口，且与其他服务映射的主机端口不能冲突，需要主机的端口用于负载均衡绑定后端的ECS实例。
- 一个服务只能使用一个或者多个负载均衡实例进行服务端口的暴露，因多个服务会分布在不同的后端，多个服务不能共享使用同一个负载均衡实例。

- 部署了带有负载均衡nat映射的服务的主机使用相同的主机：容器端口映射，因此这些服务在每台主机上只有一个实例。
- 支持的负载均衡协议$scheme包括tcp、http、https、udp协议。
- 您需要自行在阿里云负载均衡官方控制台创建监听的端口。
- 请自行登录负载均衡官方控制台对使用在容器服务中的负载均衡实例进行具体的配置修改，例如带宽限制等配置。
- lb标签的价值在于您不需要自行绑定负载均衡后端的ECS服务器，只需要配置好相应的标签，就会帮助您完成绑定后端的操作。因此，除了绑定负载均衡后端的操作，您对负载均衡的设置和修改需要自行在阿里云负载均衡云控制台上完成。
- 容器服务会帮助您生成一个RAM子账户（需要您开通RAM），使用这个具有部分负载均衡权限（没有创建和删除负载均衡的权限）的账号帮助您管理在容器服务中使用的负载均衡实例，例如绑定集群中某些节点作为服务的后端。
- 在服务的整个生命周期内，lb标签会一直生效，除非服务被删除，或者lb标签删除之后重新部署了服务，在此期间，配置在lb标签内的负载均衡实例不能混用。

12.5.5.8 global

设置该服务为全局服务。

模板版本支持

表 29: 模板版本支持

Compose V1/V2	Compose V3
*aliyun.global*标签	compose中的*mode*一节。该功能对应于swarm mode的Global Service，当新节点加入集群里，会自动创建相应容器。

Compose V1/V2

设置该服务为全局服务。

有一些服务需要在每一个节点部署，例如监控或是日志类的服务。并且在新的节点建立的时候就对这个节点进行服务的部署。

当一个服务被设置为global时，该服务会在集群中的每一个节点进行部署。当集群中有新增节点时，也会自动部署一个容器实例到新节点之上。

编排示例：

```
monitor:
  image: sample
  labels:
    aliyun.global: true
```

Compose V3

该功能对应于swarm mode的Global Service（参见*mode*），当新节点加入集群里，会自动创建相应容器。

编排示例：

```
version: '3'
    services:
      worker:
        image: dockersamples/examplevotingapp_worker
        deploy:
          mode: global
```

mode

可以设置为global （每个swarm mode一个容器）或者replicated（指定数量的容器）。默认为replicated。

```
version: '3'
        services:
          worker:
            image: dockersamples/examplevotingapp_worker
            deploy:
              mode: replicated
              replicas: 6
```

12.5.5.9 服务部署约束（affinity:service）

设置该服务的部署约束条件。

模板版本支持

Compose V1/V2 和 Compose V3 都支持设置部署约束条件，且语法相同。

功能概述

容器服务支持Docker Swarm兼容的容器部署约束条件，您可以通过Docker Swarm Filter控制一个容器的部署。

但是在社区版Docker Compose中，却并没有相关的能力来控制服务直接的部署约束。

在容器服务中，您可以在environment中添加相关affinity:service，来约束服务之间的亲和度（Affinity），达到控制服务部署策略的功能。支持服务之间的Soft affinity和Hard affinity。

编排示例

Compose V1/V2编排示例：

本示例中，slave服务设置了affinity.service!=master的部署约束。使得slave服务一定会选择没有部署master服务的节点，这样当一个节点失效时，服务可用性不受影响。当您的集群只有一个节点的时候，由于指定的是hard anti-affinity，该部署会失败，因为部署没有办法满足所指定的强约束条件。

```
master:
  image: mysql:5.6
  environment:
    - MYSQL_USER=user
    - MYSQL_PASS=test
    - REPLICATION_MASTER=true
    - REPLICATION_USER=repl
    - REPLICATION_PASS=repl
  ports:
    - 3306
slave:
  image: mysql:5.6
  environment:
    - MYSQL_USER=user
    - MYSQL_PASS=test
    - REPLICATION_SLAVE=true
    - affinity:service!=master
  ports:
    - 3306
  links:
    - master:mysql
```

Compose V3编排示例：

本示例中，mysql服务设置了affinity.service!=web的部署约束。使得mysql服务一定会选择没有部署web服务的节点，这样当一个节点失效时，服务可用性不受影响。当您的集群只有一个节点的时候，由于指定的是hard anti-affinity，该部署会失败，因为部署没有办法满足所指定的强约束条件。

```
version: '3'
    services:
      web:
        image: wordpress:4
        environment:
          - WORDPRESS_DB_PASSWORD=password
          - WORDPRESS_AUTH_KEY=changeme
          - WORDPRESS_SECURE_AUTH_KEY=changeme
          - WORDPRESS_LOGGED_IN_KEY=changeme
          - WORDPRESS_NONCE_KEY=changeme
          - WORDPRESS_AUTH_SALT=changeme
          - WORDPRESS_SECURE_AUTH_SALT=changeme
          - WORDPRESS_LOGGED_IN_SALT=changeme
```

```
      - WORDPRESS_NONCE_SALT=changeme
      - WORDPRESS_NONCE_AA=changeme
    ports:
      - 80
    deploy:
      replicas: 3
      restart_policy:
        condition: on-failure
    labels:
      aliyun.routing.port_80: wordpress
  mysql:
    image: mysql:5.7
    environment:
      - MYSQL_ROOT_PASSWORD=password
      - affinity:service!=web
    deploy:
      restart_policy:
        condition: on-failure
```

12.5.5.10 external

设置该服务直接链接到外部地址。

模板版本支持

表 30: 模板版本支持

Compose V1/V2	Compose V3
compose的external一节。	不支持。

Compose V1/V2

设置该服务直接链接到外部地址。

扩展字段下有以下字段可以使用：

- host：设置链接的域名。
- ports：设置链接的端口。

编排示例：

不使用external，直接启动一个MySQL容器。

```
web:
  image: wordpress:4.2
  ports:
    - 80
  links:
    - db:mysql
db:
  image: 10.32.161.160:5000/mysql
  environment:
```

```
    - MYSQL_ROOT_PASSWORD=password
```

通过external，描述一个并没有部署在集群中的RDS服务，并提供给部署在集群中的WordPress使用。

```
wordpress:
  image: wordpress:4.2
  ports:
    - 80
  links:
    - db:mysql
  environment:
    - WORDPRESS_DB_USER=cloud
    - WORDPRESS_DB_PASSWORD=MYPASSWORD
    - WORDPRESS_DB_NAME=wordpress
db:
  external:
    host: rdsxxxx.mysql.rds.aliyuncs.com
    ports:
      - 3306
```

12.5.5.11 变量替换

容器服务支持参数化的Docker Compose模板。模板中可以包含环境变量作为参数，当模板部署时会提示输入参数值，并在部署时对模板进行变量替换。

比如，您可以定义参数POSTGRES_VERSION。

```
db:
  image: "postgres:${POSTGRES_VERSION}"
```

当部署上面的Compose模板的时候，容器服务会提示您输入POSTGRES_VERSION参数值，比如9.3。容器服务会根据参数值对Compose模板进行变量替换。在本示例中，会部署一个postgres:9.3的容器。

容器服务完全兼容Docker Compose的语法，可以在模板中使用$VARIABLE或者${VARIABLE}格式的语法。

在Compose模板中可以使用$$来对需要包含$的字符串进行转义，这样容器服务不会错误地将其作为参数来进行处理。

关于Compose模板支持变量替换的详细信息，参见*Variable substitution*。

12.5.5.12 容器重新调度

容器服务支持对Docker容器的重新调度：当一个节点失效时，容器可以被自动调度到其他可用节点自动运行。

模板版本支持

表 31: 模板版本支持

Compose V1/V2	Compose V3
环境变量中的*reschedule:on-node-failure*。	默认提供，不需要设置。

Compose V1/V2

缺省情况下，容器的重新调度策略是关闭的。根据需要，您可以用如下配置来让重调度策略生效。

容器服务提供兼容Docker Swarm的容器重新调度策略，可以通过环境变量方式或者label方式启动。

环境变量：

```
redis:
  image: redis
  environment:
    - reschedule:on-node-failure
```

Label：

```
redis:
  image: redis
  labels:
    - com.docker.swarm.reschedule-policies=["on-node-failure"]
```

说明: 如果重新调度容器之后，需要恢复Docker容器所需的持久化状态，需要配合支持数据迁移或共享的Docker文件卷。

12.5.5.13 networks

将服务添加到指定的自定义网络中。

模板版本支持

表 32: 模板版本支持

Compose V1/V2	Compose V3
Compose V2中的网络允许服务的容器启动在自定义的网络中，容器服务的容器都是在同一个跨主机互通的容器网络，所以不支持您在Compose V2中使用networks标签。关于容器服务的网络管理和服务发现，参见跨主机互联的容器网络。	支持使用networks标签将应用添加到自定义网络中。参见*networks*。

Compose V3

您可以在创建应用时通过指定应用的网络将服务添加到自定义的网络中。您可以将服务添加到一个或多个自定义网络中来实现服务间的互相访问。

说明:

- 将应用添加到某一自定义网络后，您无法更改应用的网络。
- 将应用添加到某一自定义网络后，您无法将应用从该网络中移除。只有在删除应用时，应用才会从该网络中移除。

编排示例：

```
version: '3.2'
services:
  test:
    image: 'wordpress:4'
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        failure_action: continue
      restart_policy:
        condition: none
    networks:
      - swarmmode #您的自定义网络的名称
      - testswarm #您的自定义网络的名称
networks: #声明您的自定义网络
  swarmmode:
    external: true
  testswarm:
```

```
    external: true
```

您需要对应用所要使用的自定义网络进行声明，否则部署应用时会报错。其中，external: true表示自定义网络已创建，创建应用时容器服务不会尝试创建该自定义网络，而是查找该自定义网络并将应用添加至该自定义网络。



说明: *external*不能与其它网络配置参数（*driver*、*driver_opts*、*ipam*、*internal*）一起使用。

12.5.5.14 不支持的Docker Compose标签

标签	说明
build	build标签用于使用当前目录中的Dockerfile文件和其他文档进行容器镜像构建。目前容器服务暂不提供构建镜像功能，推荐您将构建和部署的动作分开处理：您可以利用阿里云的镜像仓库 直接从代码源构建镜像，或者将本地构建的镜像推送到镜像仓库；您可以在编排模板中使用image标签引用镜像仓库（包括私有仓库）中的镜像。
dockerfile	说明同build标签。
env_file	容器服务暂不支持以文件方式指定环境变量，您可以通过environment标签添加环境变量。
mac_address	暂时不支持MAC地址的设置。
detach	容器服务的所有镜像都是以detach模式启动的，不允许您指定attach方式执行。
stdin_open	说明同detach标签。
tty	说明同detach标签。

12.5.6 管理应用

您可以创建应用，限制容器的资源，设置应用的部署以及变更应用的配置等。

12.5.6.1 使用镜像创建应用

您可以通过容器服务云控制台使用镜像创建应用。

操作步骤

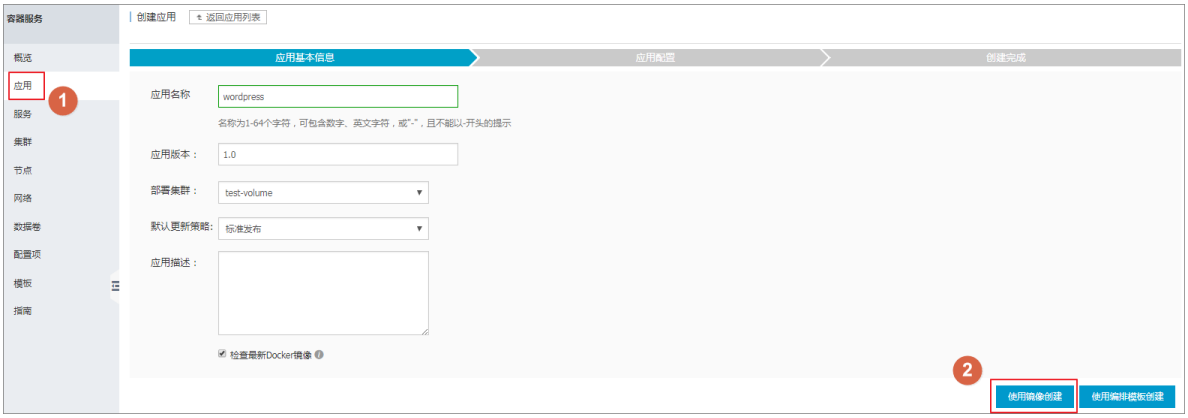
1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**，单击右上角的**创建应用**。

3. 设置应用的基本信息并单击**使用镜像创建**。

- **应用名称**：所创建应用的名称。名称可以包含1~64个字符，包括数字，中文字符，英文字母和连字符（-）。
- **应用版本**：所创建应用的版本。默认为1.0。
- **部署集群**：所创建应用将要部署到的集群。
- **默认更新策略**：应用更新的方式，您可以选择标准发布或蓝绿发布，参见发布策略。
- **应用描述**：应用信息。可以不填写。该信息不能超过1,024个字符。该信息将显示在 **应用列表** 页面。
- **检查最新Docker镜像**：选中该选项后，表示当镜像Tag不变的情况下，也会去仓库拉取最新的镜像。为了提高效率，容器服务会对镜像进行缓存。部署时，如果发现镜像Tag与本地缓存的一致，则会直接复用而不重新拉取。所以，如果您基于上层业务便利性等因素考虑，在做代码和镜像变更时没有同步修改Tag ，就会导致部署时还是使用本地缓存内旧版本镜像。而勾选该选项后，会忽略缓存，每次部署时重新拉取镜像，确保使用的始终是最新的镜像和代码。



4. 根据您的需要设置以下参数。

   a) 设置**镜像名称**。

   您可以直接填写目标Hub的镜像名称，填写的格式为域名*/namespace/imagename:tag*。

   b) 设置**Mode**。

   - **Replicated**：默认值。创建指定数量的容器。选择该选项时，您需要指定容器的数量。
   - **Global**：每个节点仅包含一个容器。

   c) 设置容器的启动命令（Command和Entrypoint）。

   如果设置了会覆盖镜像的配置。

   d) 设置服务的更新策略。

设置当服务包含多个容器时应该如何进行更新。

- **更新失败后的动作**：设置如果某一更新失败时，系统如何操作。**继续**表示失败时，继续更新下一批容器。**停止**表示失败时，停止更新服务。
- **更新间隔**：不同批次的容器更新的时间间隔。
- **允许失败比例**：更新服务时允许的更新失败占比。
- **每批更新任务数**：每次批量更新的容器的个数。

e) 设置容器的重启规则。

设置容器退出后，系统如何尝试重新启动容器。

- **重启条件**：在什么情况下重新启动容器。可以选择**不重启**、**失败时**和**总是**。默认为**总是**。
- **重启间隔**：设置尝试重启容器的时间间隔。默认为 0。
- **重启次数**：尝试重启容器的最大次数。默认为一直尝试重启。
- **检测间隔**：检测容器重启成功的时间间隔。默认值为马上检测。

f) 设置**约束条件**。



说明:

如果您指定的节点约束和*config.json*中设置的**reserved_nodes**有冲突，应用部署会失败，系统提示如下错误信息。

last update error:'Failed to start service xxxxxx in 180 seconds. The service is in pending state. Maybe node constraints can't be satisfied.'

g) 设置**端口映射**。

h) 设置容器的**数据卷**。

i) 设置容器的资源限制（**CPU**和**内存**）。

- **CPU**：1.0表示整个CPU。
- **内存**：单位为Bytes。

有关容器资源限制的详细信息，参见限制容器资源。

j) 设置容器的资源预留（**CPU**和**内存**）。

k) 设置应用的**网络**。

您可以将应用添加到多个网络中。有关网络的详细信息，参见管理网络。

l) 设置**环境变量**。

m) 设置**服务标签**。

n) 设置**容器标签**。

有关支持的标签，参见标签概览。

o) 设置**密钥**。

详细信息，参见概述。

p) 设置容器的**自动伸缩**规则。

图 219: 自动伸缩规则



5. 设置完毕后，单击**创建并部署**。

12.5.6.2 使用编排模板创建应用

您可以通过容器服务云控制台使用编排模板创建应用。

使用限制

swarm mode 集群支持Compose V1/V2 和 Compose V3版本的编排模板。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**，单击右上角的**创建应用**。

3. 设置应用的基本信息并单击**使用编排模板创建**。

   - **应用名称**：所创建应用的名称。名称可以包含1~64个字符，包括数字，中文字符，英文字母和连字符（-）。
   - **应用版本**：所创建应用的版本。默认为1.0。
   - **部署集群**：所创建应用将要部署到的集群。
   - **默认更新策略**：应用更新的方式，您可以选择标准发布或蓝绿发布，参见发布策略概述。
   - **应用描述**：应用信息。可以不填写。该信息不能超过1,024个字符。该信息将显示在 **应用列表** 页面。

- **检查最新Docker镜像**：选中该选项后，表示当镜像Tag不变的情况下，也会去仓库拉取最新的镜像。为了提高效率，容器服务会对镜像进行缓存。部署时，如果发现镜像Tag与本地缓存的一致，则会直接复用而不重新拉取。所以，如果您基于上层业务便利性等因素考虑，在做代码和镜像变更时没有同步修改Tag ，就会导致部署时还是使用本地缓存内旧版本镜像。而勾选该选项后，会忽略缓存，每次部署时重新拉取镜像，确保使用的始终是最新的镜像和代码。

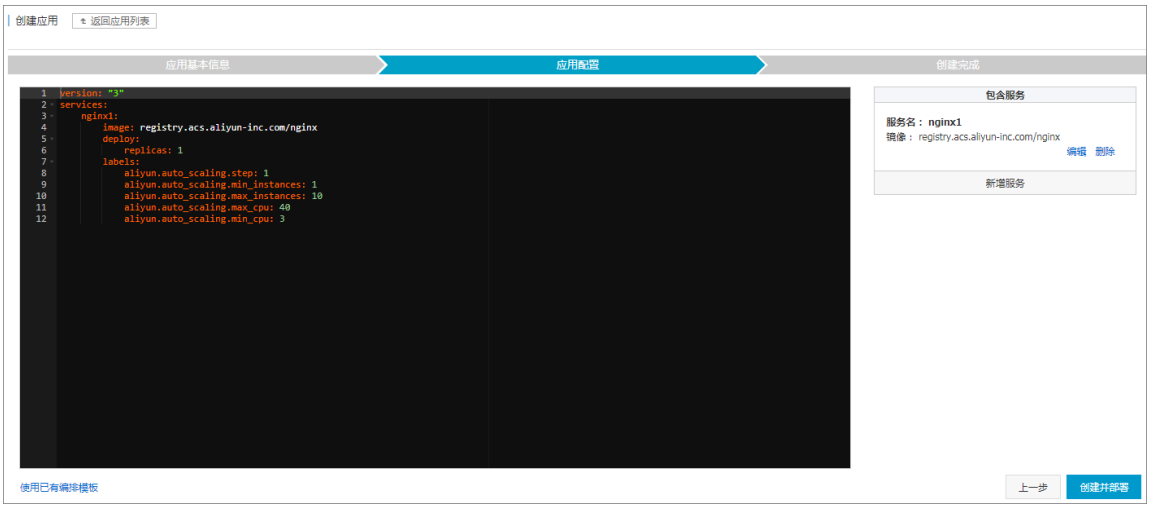
4. 编写编排模板。

- 您可以自己编写编排模板。
- 或者单击**使用已有编排模板**，选择一个模板并单击**选择**。

> **说明:** swarm mode集群支持Compose V1/V2和Compose V3版本的编排模板。

您可以根据自己的需要编辑编排模板。您可以直接在编排模板中进行修改，或者在页面右侧选择需要修改的服务，单击**编辑**进行修改或者单击**删除**删除所选的服务

图 220: 编写编排模板



此外，您还可以单击服务列表下面的**新增服务**添加服务。填写您所需要的镜像的名称，进行参数设置并单击**确定**。

图 221: 新增服务



5. 设置完毕后，单击**创建并部署**。

12.5.6.3 限制容器资源

Docker容器的一大优势就是可以限制资源，包括CPU、内存、IO等。swarm mode集群支持限制资源和预留资源。该设置无法通过容器服务云控制台进行操作，您需要在编排模板中进行设置。

swarm mode集群支持Compose V1/V2和Compose V3。其中，关于资源约束配置的语法，Compose V1/V2和Compse V3不同。

模板版本支持

表 33: 模板版本支持

Compose V1/V2	Compose V3
*mem_limit*标签、*cpu_shares*标签	*limits*标签设置资源限制。*reservations*标签设置资源预留。参见*resources*。

Compose V1/V2

一个CPU核等于100 CPU资源。如果机器配置是4核，则总共可用的CPU资源为400。在编排模板中，可以通过cpu_shares参数指定。cpu_shares: 50表示使用0.5个核。

您可以使用mem_limit参数限制内存，单位为byte，最小内存为4MB。如果设置了内存限制，当容器申请的内存超过限制时，容器会因为OOM而停止运行。

编排示例：

```
n1:
  expose:
    - 443/tcp
    - 80/tcp
  image: 'nginx:latest'
  cpu_shares: 50 #0.5核
  mem_limit: 500000000 #500MB
  labels:
    aliyun.scale: '1'
  restart: always
  volumes:
    - /var/cache/nginx
```

资源调度

为了保证容器能获得足量的指定资源，比如上述例子里的0.5核CPU和500MB内存，容器服务会为容器预留资源。比如，一台4核的机器，最多会调度8个cpu_shares=50的容器。但是，如果创建容器时未指定cpu_shares和mem_limit，则默认不占资源。

Compose V3

在Compose V3编排模板中，可以通过*limits*标签指定资源限制；通过*reservations*标签设置资源预留。

编排示例：

```
version: '3'
n1:
  expose:
    - 443/tcp
```

```
  - 80/tcp
image: 'nginx:latest'
deploy:
  resources:  #注意
    limits:
      cpus: '0.5'
      memory: 500M
    reservations:
      cpus: '0.3'
      memory: 200M
restart: always
volumes:
  - /var/cache/nginx
```

限制其他资源

其他资源限制请参考*Docker Compose* 的说明。

12.5.6.4 指定节点调度

swarm mode集群支持在容器服务云控制台上通过设置界面上的参数或者通过编写编排模板来设置节点的约束条件。

通过界面上的参数进行设置

您可以在 使用镜像创建应用 或者 变更服务配置 时，设置 **约束**。

图 222: 添加约束



通过编排模板进行设置

模板版本支持

表 34: 模板版本支持

Compose V1/V2	Compose V3
*constraint*关键字	Compose 中的*placement*一节。该功能由 swarm mode 原生提供，可设置部署约束

Compose V1/V2

如果您需要将某个服务部署在指定的节点上，可以使用关键字*constraint*来实现这个功能。

说明: 部署约束只对新创建容器生效，对老容器变更配置时不起作用。

在下面的示例中，服务指定部署在node1上。

```
web:
    image: 'nginx:latest'
    restart: always
    environment:
        - 'constraint:aliyun.node_index==1'
    ports:
        - 80
    labels:
        aliyun.scale: 2
```

容器服务支持以下表达式：

表达式	说明
constraint:aliyun.node_index==1	指定部署到node1。
constraint:aliyun.node_index!=1	不部署到node1。
constraint:aliyun.node_index==(1|2|3)	指定部署到node1或者node2或者node3。
constraint:aliyun.node_index!=(1|2|3)	部署到除node1、node2、node3的其他机器上。
affinity:image==~redis	尽量部署到有Redis镜像的机器上。
affinity:service!=~redis	尽量不部署到有Redis容器的机器上。

Compose V3

在 compose V3 中，同样通过关键字*constraint*进行设置，但语法和 compose V1/V2 不同，且支持多种部署限制。

在下面的示例中，服务指定部署到 Manager 节点上。

```
    version: '3'
    services:
      web:
        image: wordpress:4
        ports:
          - 80
        depends_on:
          - mysql
        deploy:
          replicas: 3
          restart_policy:
            condition: on-failure
          placement:  #注意
            constraints:
              - node.role == manager
        labels:
          aliyun.routing.port_80: wordpress
      mysql:
        image: mysql:5.7
```

```
    environment:
      - MYSQL_ROOT_PASSWORD=password
    deploy:
      restart_policy:
        condition: on-failure
```

compose V3 支持以下语法。

表 35: Compose V3语法

节点属性	说明	例子
node.id	Node ID	node.id == 2ivku8v2gvtg4
node.hostname	Node hostname	node.hostname != node-2
node.role	Node role	node.role == manager
node.labels	user-defined node labels 有关如何创建标签，参见创建用户标签。	node.labels.security == high
engine.labels	Docker Engine labels	engine.labels.operatingsystem == ubuntu 14.04

12.5.6.5 指定多节点调度

如果您希望一个应用在指定的某几个节点上部署，推荐您使用用户标签和constraint关键字来进行部署设置。



说明:

- 部署约束只对新创建容器生效，对老容器变更配置时不起作用。
- 使用用户标签部署应用后，如果您删除了用户标签，不会影响到已经部署的应用，但是会影响下次新的部署。请谨慎删除用户标签。

操作步骤

1. 为节点添加用户标签。
   a) 登录容器服务云控制台。

      有关如何登录云控制台，参见登录云控制台。
   b) 单击左侧导航栏中的**集群**。
   c) 选择所需的集群并单击右侧的**管理**。
   d) 单击左侧导航栏中的**用户标签**。

e) 勾选需要添加标签的节点并单击**添加标签**。

图 223: 添加用户标签



f) 输入您自定义的标签键和标签值并单击**确定**，为所选节点添加用户标签。

图 224: 用户标签



2. 创建应用，选择**使用编排模板创建**并在编排模板中配置constraint关键字。

有关如何创建应用，参见使用编排模板创建应用。

Compose V1/V2示例：

```
web:
  image: 'nginx:latest'
  restart: always
  environment:
    - constraint:group==1 #部署到带有用户标签group:1的节点上
  ports:
    - 80
  labels:
    aliyun.scale: 2
```

Compose V3示例：

```
version: '3'
services:
  web:
    image: wordpress:4
    ports:
```

```
  - 8081
deploy:
  replicas: 3
  restart_policy:
    condition: on-failure
  placement:
    constraints:
      - node.labels.group == 1 #部署到带有用户标签group:1的节点上
labels:
  aliyun.routing.port_80: wordpress
```

12.5.6.6 删除用户标签

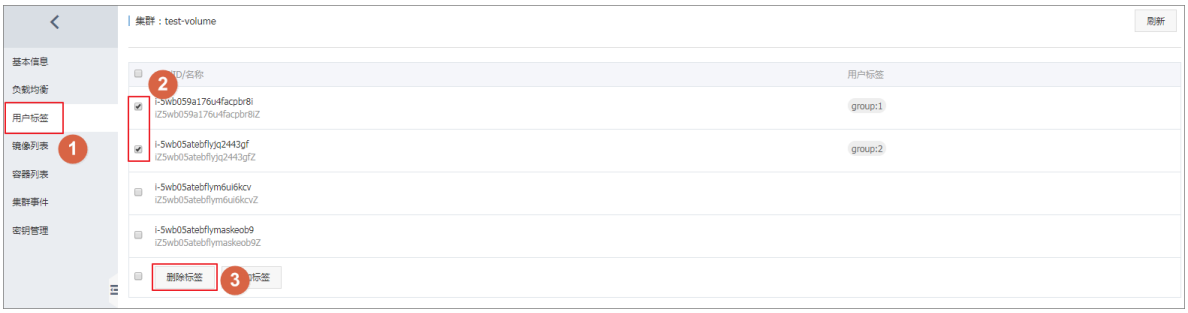
您可以删除已经不再使用的用户标签。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**集群**。
3. 选择所需集群并单击右侧的**管理**。
4. 单击左侧导航栏中的**用户标签**。
5. 勾选您要部署应用的节点并单击**删除标签**。



6. 在弹出的确认对话框中单击**确定**。



12.5.6.7 查看应用详情

您可以通过容器服务云控制台查看已经创建的应用的详细信息。

操作步骤

1. 登录容器服务云控制台。

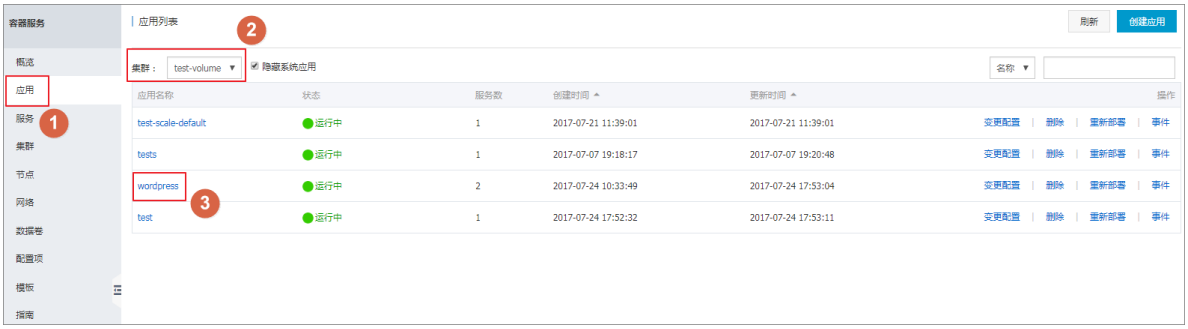
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**。

3. 选择所要查看的应用所在的集群。

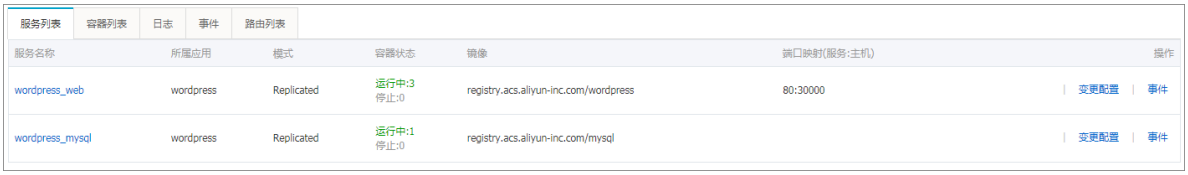
4. 单击所要查看应用的名称。

图 225: 应用列表



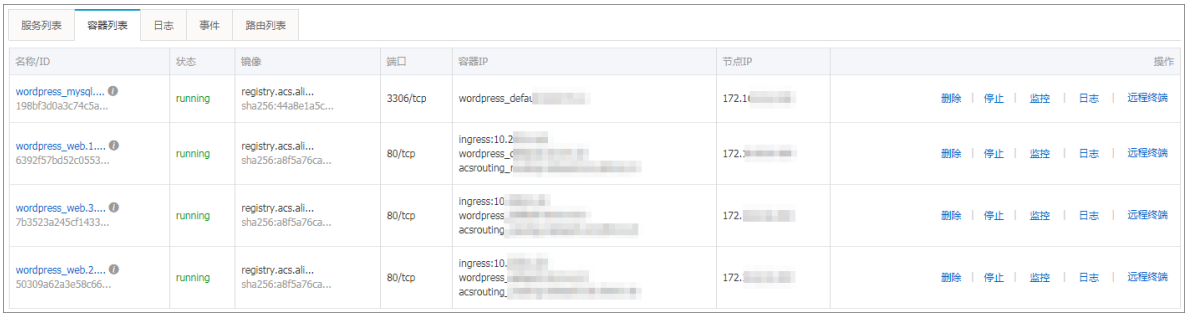
5. 单击**服务列表**查看该应用的服务列表。

图 226: 服务列表



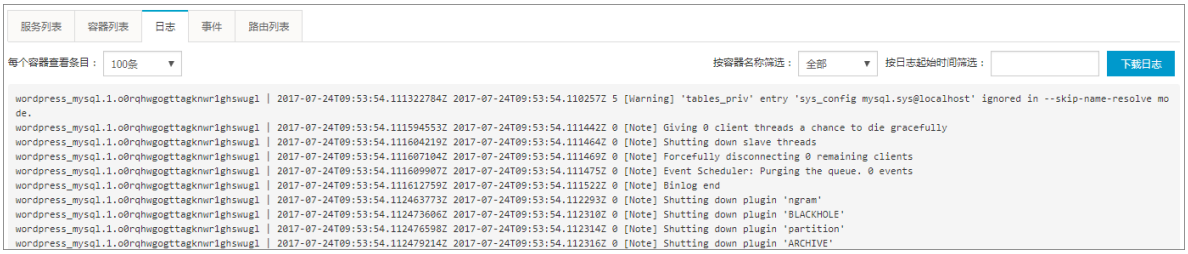
6. 单击**容器列表**查看该应用的容器列表。

图 227: 容器列表



7. 单击 **日志** 查看该应用的日志信息。

图 228: 日志



8. 单击 **事件** 查看该应用的事件信息。

图 229: 事件



9. 单击 **路由列表** 查看该应用的路由地址。

图 230: 路由列表



12.5.6.8 变更应用配置

您可以通过容器服务云控制台变更应用的配置。

操作步骤

1. 登录容器服务云控制台。

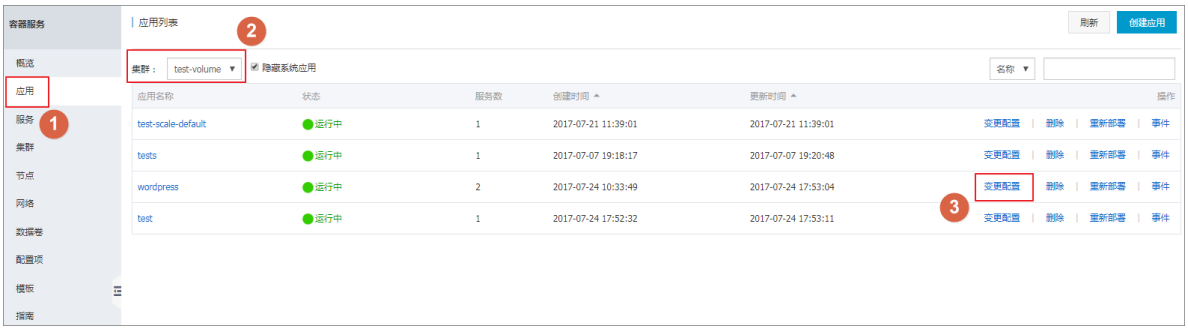
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**。

3. 选择所要查看的应用所在的集群。

4. 选择所需的应用并单击**变更配置**。

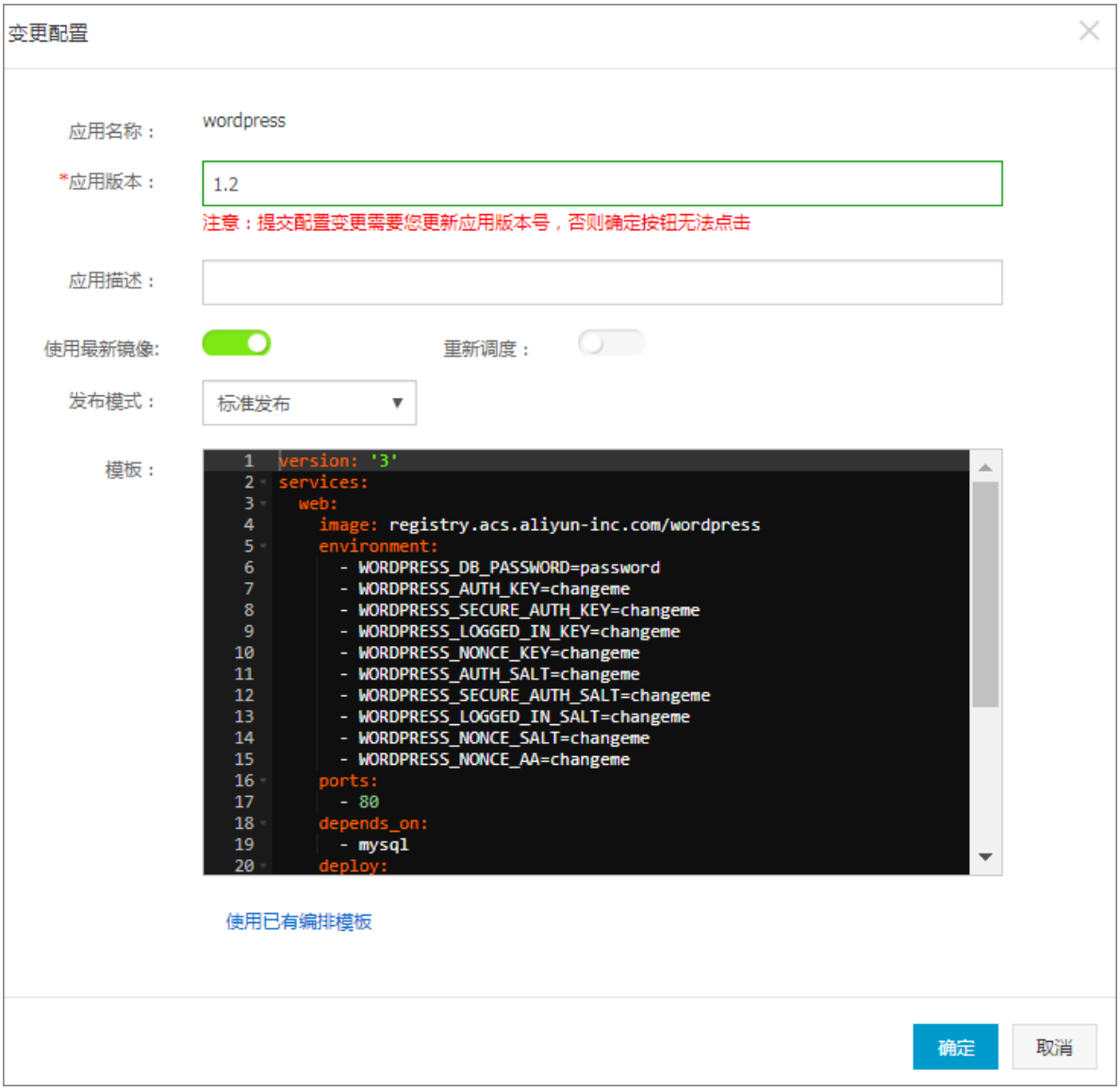
图 231: 应用列表



5. 在弹出的对话框中，修改配置。

说明: 您必须修改**应用版本**；否则，**确定**按钮不可用。

图 232: 变更配置



后续操作

如果您变更配置后，发现应用没有更新，可以尝试重新部署应用。有关重新部署应用的详细信息，参见重新部署应用。

12.5.6.9 重新部署应用

应用部署之后您可以根据您的需求对应用进行重新部署。重新部署会重新拉取应用使用的镜像，因此如果您部署应用之后更新了镜像，重新部署会使用新的镜像进行应用部署。

背景信息

说明: 重新部署不会更新volume，宿主机上的老volume仍会继续使用。因此，如果您挂载了volume并在新镜像中对volume设置进行了修改，重新部署后新设置不会生效。

在以下情况下，您会用到重新部署功能：

- 部署应用之后，您更新了镜像的内容，需要按照新的镜像部署应用。
- 您停止或删除了某些容器，希望可以启动或重新创建这些容器。重新部署时，容器服务会重新启动已经停止的容器并重新创建已经删除的容器。

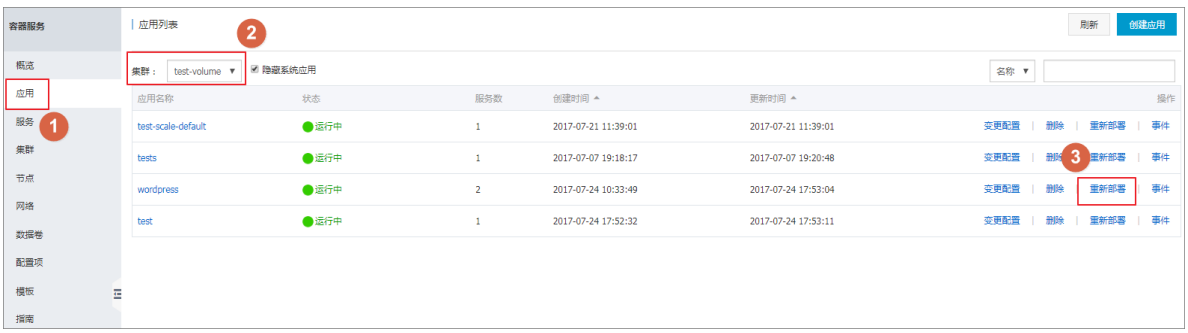
操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**。
3. 选择所要查看的应用所在的集群。
4. 选择要进行重新部署的应用，单击右侧的**重新部署**。

   图 233: 应用列表



5. 在弹出的对话框中，单击**确定**。

查看重新部署是否成功

您可以通过查看镜像的sha256确定重新部署后容器的镜像是否为最新镜像，从而确定重新部署是否成功。

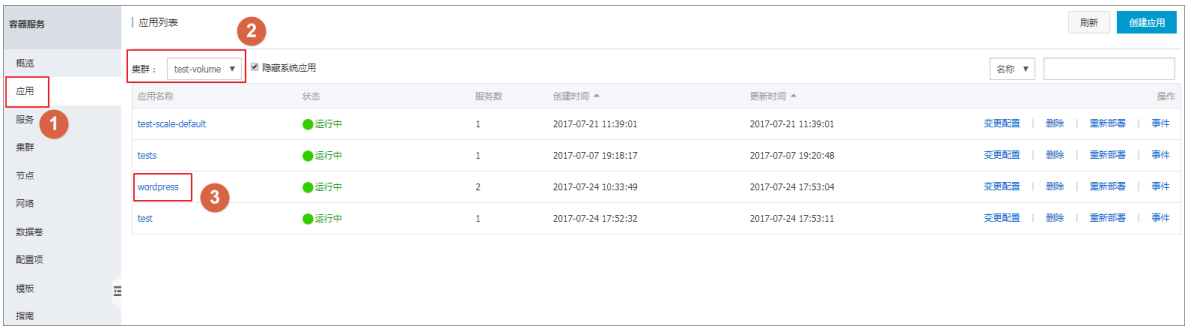
操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**。
3. 选择所要查看的应用所在的集群。
4. 单击应用的名称。

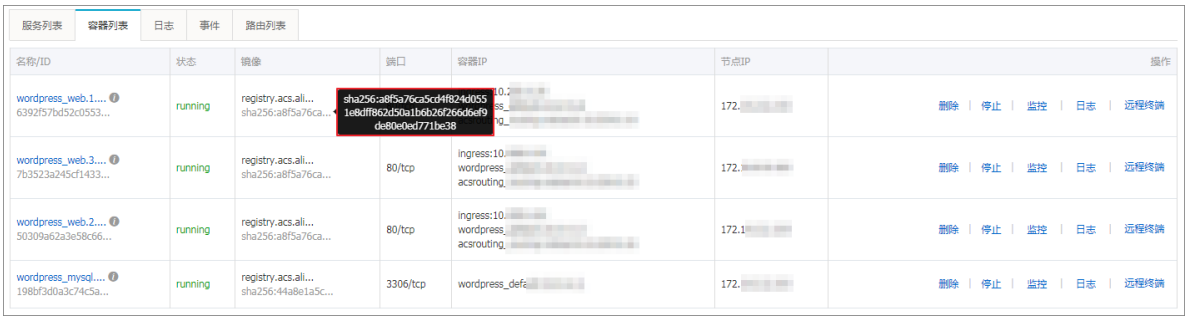
图 234: 应用列表



5. 单击**容器列表**并查看镜像的sha256。

   如果容器的镜像为新镜像，则重新部署成功。

图 235: 查看sha256



12.5.6.10 删除应用

您可以通过容器服务云控制台删除不再使用的应用。

操作步骤

1. 登录容器服务云控制台。

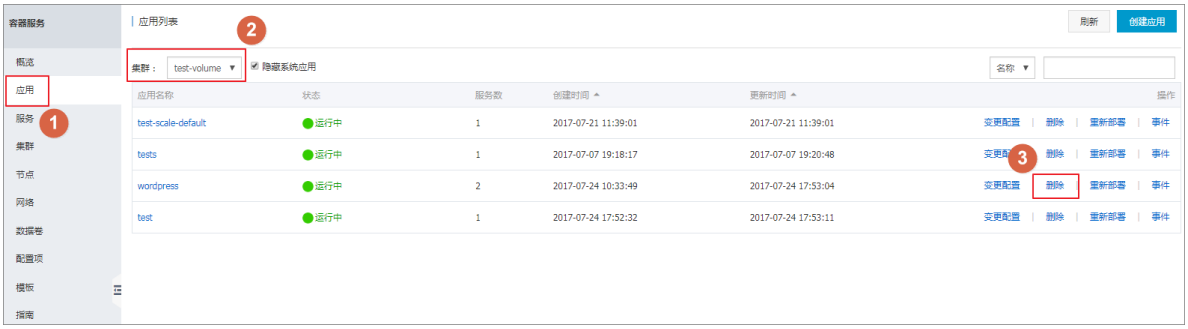
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**应用**。

3. 选择所要查看的应用所在的集群。

4. 选择所要删除的应用并单击**删除**。

图 236: 应用列表



5. 在弹出的对话框中，单击**确定**。

   如果您勾选**同时移除数据卷**（volume），系统将删除与应用相关的所有数据卷。但是，已命名的数据卷无法删除。

图 237: 确认删除



12.5.6.11 系统默认应用列表

应用名称	中文名称	包含的服务	简介
acsrouting	路由服务	routing	提供7层协议的请求路由服务。组件包括负载均衡和一个HAProxy容器。域名配置正确后，即可将请求发送到指定的容器内。
acsmonitoring	监控服务	acs-monitoring-agent	与阿里云云监控进行集成，并与目前比较流行的第三方开源监控框架做集成，方便您查询监控信息并配置监控报警。
acsvolumedriver	数据卷服务	volumedriver	与阿里云的存储服务OSS做集成，方便您以volume卷（数据卷）的方式来使用共享式存储，告别有状态的容器运维。使用方法参见数据卷服务。

12.5.6.12 创建并调用触发器

触发器是容器服务提供的简单快捷地进行重新部署和资源伸缩的API。

由于标准的API需要保证安全性，因此需要严格的鉴权。但是对于需要与其他三方系统（例如Jenkins 或者其他持续集成的CI/CD系统）集成的场景来说，所需要的权限是有限的，可能仅仅需要消息通知。因此为了保证安全性与便捷性，带有部分鉴权策略，并且可以灵活调用的API被广泛应用于持续集成持续交付的场景中。

容器服务目前提供重新部署触发器和资源伸缩触发器。

- **重新部署触发器**

  您可以与自己的监控系统进行集成，当发现系统异常时进行重新部署；您也可以与容器Hub进行集成，当容器新镜像构建完成后，可以自动进行部署等。

- **资源伸缩触发器**

  您可以通过调用资源伸缩触发器来实现容器伸缩。

创建触发器

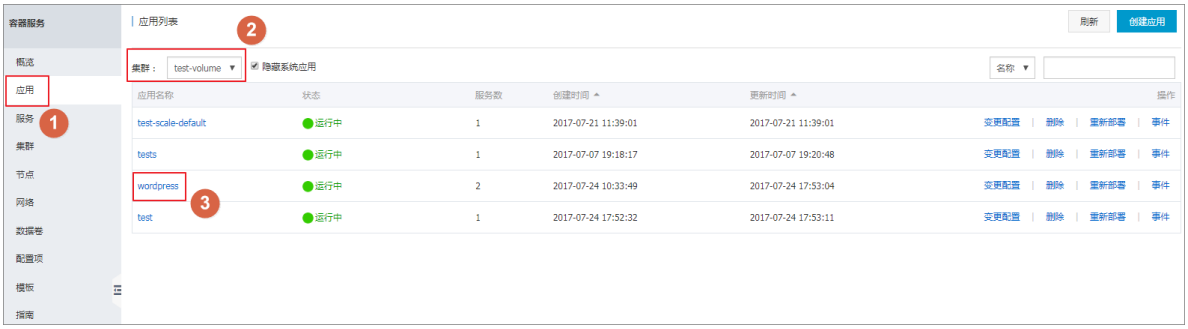
1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**应用**。
3. 选择目标应用所在的集群。

4. 选择目标应用并单击应用的名称。

图 238: 应用列表



5. 在应用详情页面，单击**创建触发器**。
6. 在**触发器行为**下拉框中选择**重新部署**或者**资源伸缩**并单击**确定**。

- **重新部署**

  当您对应用使用的镜像有写权限时，您可以勾选**关联到镜像更新**。勾选后，当容器新镜像构建完成后，可以使用最新镜像自动对应用进行重新部署。

图 239: 创建触发器



- **资源伸缩**

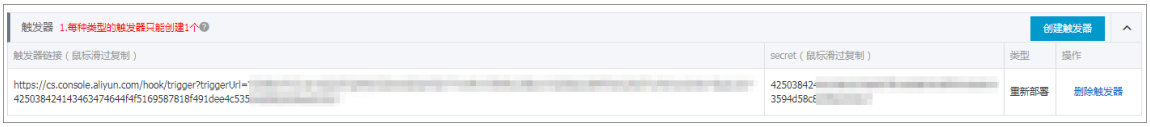
  在**服务**下拉框中选择需要设置资源伸缩触发器的服务。

图 240: 创建触发器



此时生成的触发器地址即为API的地址。

图 241: API地址



调用触发器

您可以通过三方集成系统进行触发，使用GET或者POST都可以进行触发，例如使用curl命令触发

调用重新部署触发器：

```
curl 'https://cs.console.aliyun.com/hook/trigger?triggerUrl=YzI4YTk5NzFkZWZkYzQ2MTJiOWZkNTM1MzY2ZDU1M2NifGNvbGxlY3RkLWJlbmNobWFya3xyZWRlcGxv
```

```
eXwxOGlxbjc1Z25uMmVzfA==&secret=44586c6b466352395143584c3970654ff5323d25
09d546fdc1b33054b0928da8'
```

调用资源伸缩触发器：

调用资源伸缩触发器时，需要在触发器URL中手动添加以下参数：

表 36: 资源伸缩触发器参数

参数名称	必填	语义	可选值
type	是	伸缩类型	缩容：scale_in；扩容：scale_out
step	是	伸缩数量	正整数，1-100

例如，调用下面的触发器会执行扩容五个容器的操作。

```
curl 'https://cs.console.aliyun.com/hook/trigger?triggerUrl=Y2lxZjI5YzhhYjIwMzRl
MjBiYjc2OGUzYTlmZDgyNDAyfHdvcmRwcmVzcy10ZXN0fHNjYWxpbmd8MTkz
ZmEyMXFwZXVwMXw=&secret=53374142724e4e4a626f664a313131556e62c671
6cd0d97d096900b3ad42a9ad&type=scale_out&step=5'
```

12.5.6.13 删除触发器

您可以通过容器服务云控制台删除不再使用的触发器。

操作步骤

1. 登录容器服务云控制台。

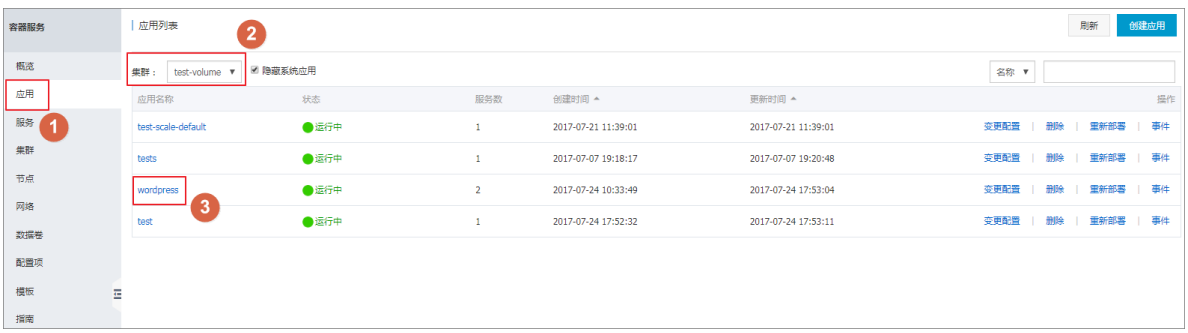
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**应用**。

3. 选择目标应用所在的集群。

4. 选择目标应用并单击应用的名称。

图 242: 应用列表



5. 选择要删除的触发器并单击右侧的**删除触发器**。
6. 在弹出的确认对话框里，单击**确定**。

12.5.7 配置项

应用分为代码和配置两个部分，当应用容器化之后，通常通过容器环境变量的方式传递配置，从而实现同一个镜像使用不同的配置部署多套应用。

使用限制

目前，在创建应用时关联配置文件的场景仅适用于通过编排模板创建应用。

应用场景

创建应用

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**配置项**，选择需要创建配置项的集群并单击**创建配置项**。

图 243: 创建配置项

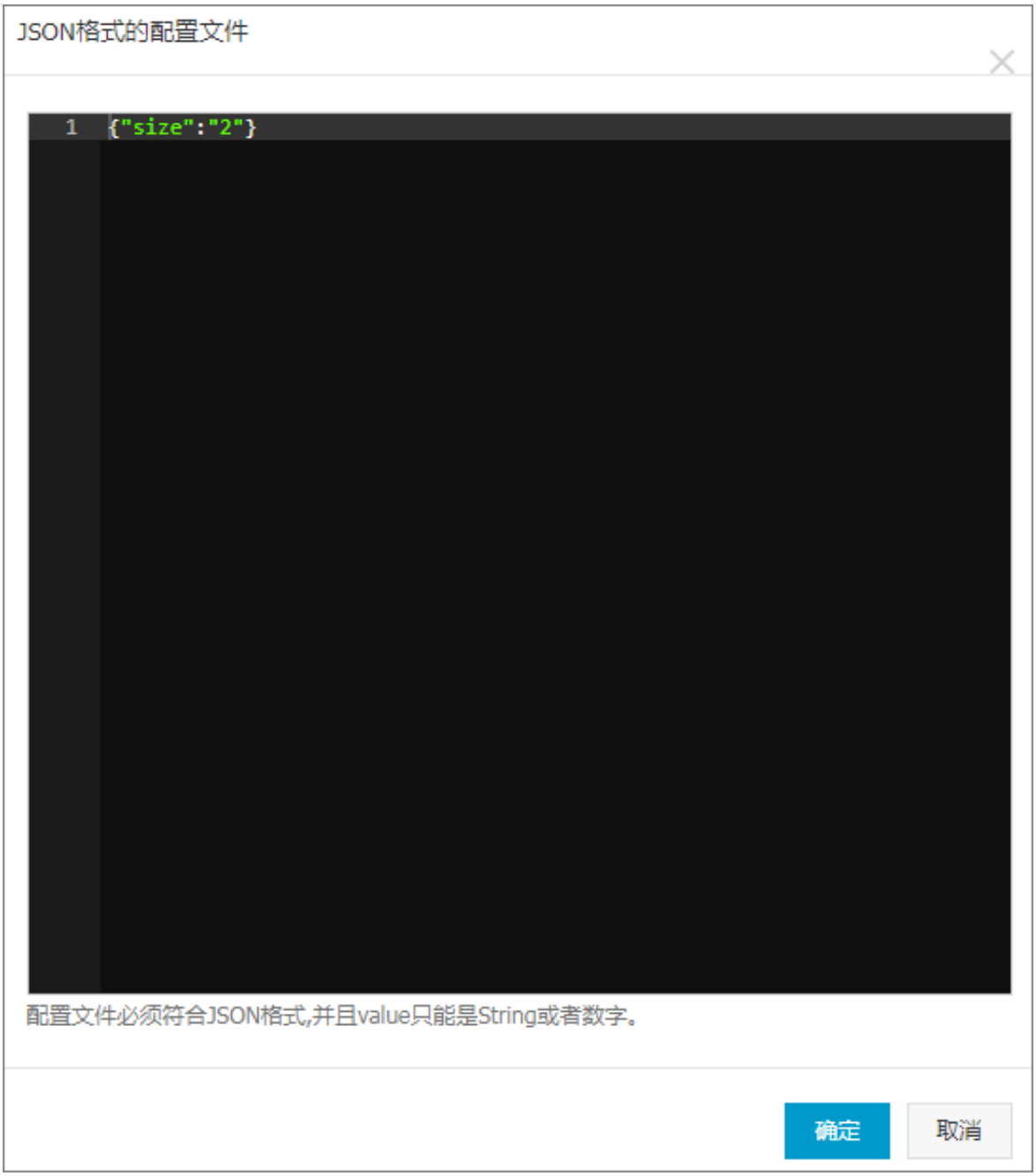


3. 填写配置文件的信息并单击**确定**。

   - **配置文件名**：可包含1~128 个字符。
   - **描述**：最多可包含128个字符。
   - **配置项**：您最多可以设置50个配置项。

     您也可以单击**编辑配置文件**在弹出的对话框里编写配置项并单击**确定**。

图 244: JSON格式文件



本示例中设置了*size*变量。

图 245: 配置文件



4. 单击左侧导航栏中的**应用**，选择与配置项处于相同地域下的集群，并单击**创建应用**。
5. 填写应用的基本信息并单击**使用编排模板创建**。
6. 填写如下所示的编排模板并单击**创建并部署**。

   其中，*size*即为动态的变量，这个变量会被配置项中的数值覆盖。

```
busybox:
 image: 'busybox'
 command: 'top -b'
 labels:
    aliyun.scale: $size
```

7. 在弹出的对话框中，选择要关联的配置文件，单击**替换变量**并单击**确定**。

图 246: 替换变量



更新应用

如果创建应用时关联了配置文件，您可以通过修改配置项并重新部署的方式更新应用。

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**配置项**，选择需要修改的配置项的地域并单击**修改**。

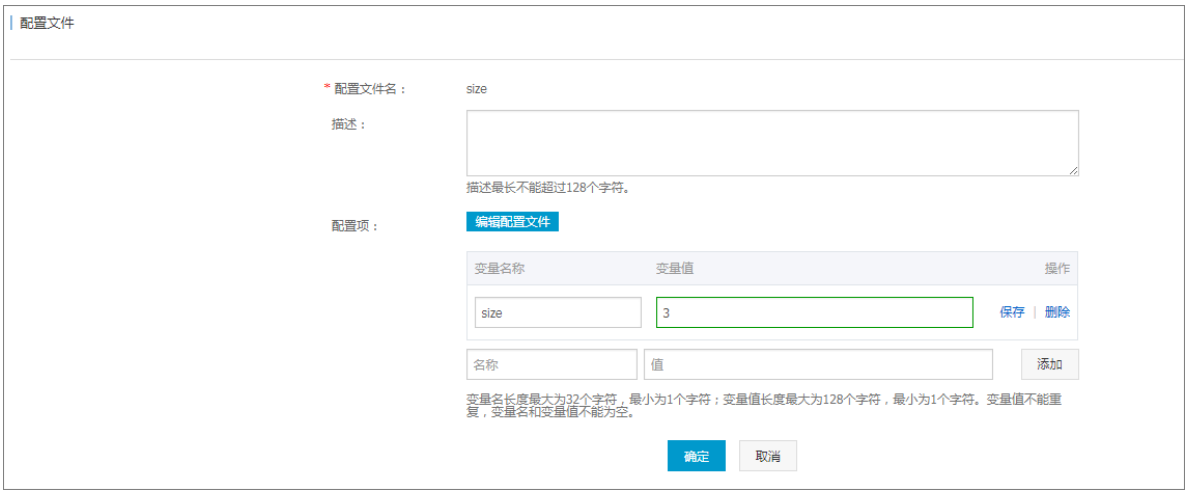
图 247: 修改配置项



3. 在弹出的确认对话框中，单击**确定**。

4. 选择要修改的变量并单击**编辑**（单击后变为**保存**）。修改变量值，单击**保存**并单击**确定**。

图 248: 修改配置项



5. 单击左侧导航栏中的**应用**，选择应用所在的集群，选择应用并单击重**新部署**。

触发更新

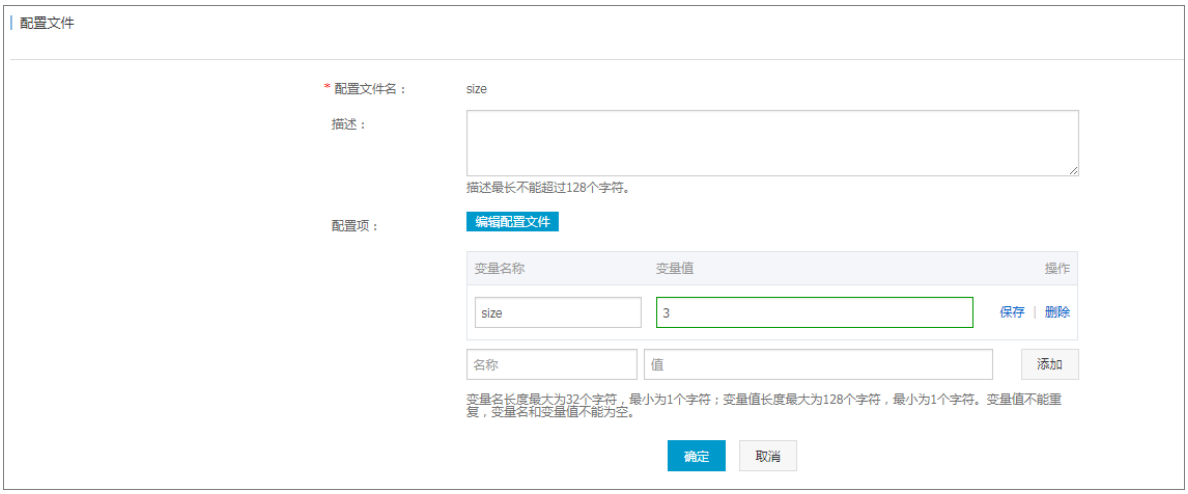
如果创建应用时关联了配置文件，您可以通过触发器触发的方式进行重新部署。

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**配置项**，选择需要修改的配置项的地域并单击**修改**。

图 249: 修改配置项



3. 创建重新部署触发器。

   有关如何创建触发器的详细信息，参见触发器。

图 250: 创建触发器



图 251: 触发器



4. 触发重新部署触发器。

```
curl "https://cs.console.aliyun.com/hook/trigger?triggerUrl=Y2FkZDgyMTUwNzdjYjQ3YThiZWEwMjAyODU1MjBlNDA5fGJ1c3lib3h8cmVkZXBsb3l8MTk0dHVzc29kNGJvcHw=&secret=517a45506f51317652744d774f58416a074b1b75299b83dd91b66150bac7ac15"
```

12.5.8 管理服务

您可以通过容器服务云控制台管理您的服务。

一个应用由一个或多个服务组成，您可以选择变更应用的配置来升级应用，也可以选择单独变更某个服务的配置来独立升级。

12.5.8.1 查看服务详情

您可以通过容器服务云控制台查看服务的详细信息。

操作步骤

1. 登录容器服务云控制台。

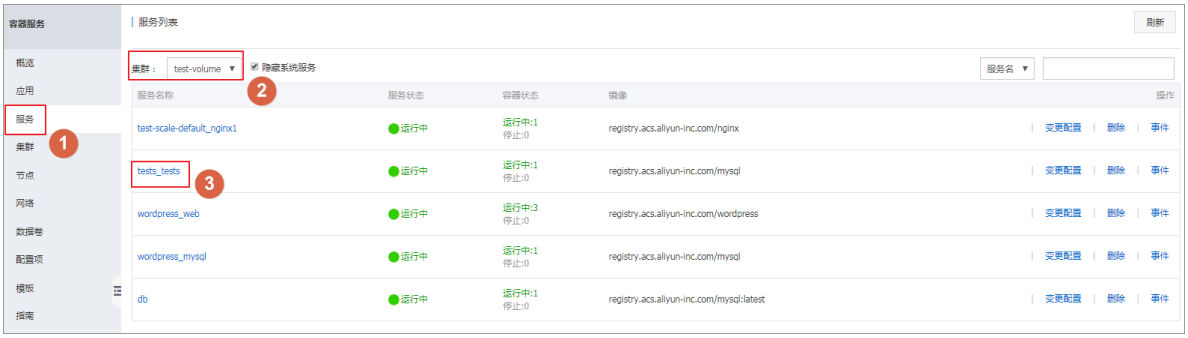
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**服务**。

3. 选择所要查看的服务所在的集群。

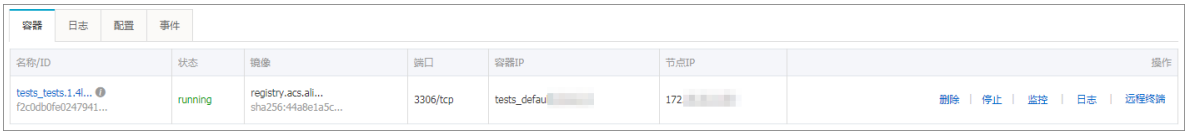
4. 单击所要查看的服务的名称。

图 252: 服务列表



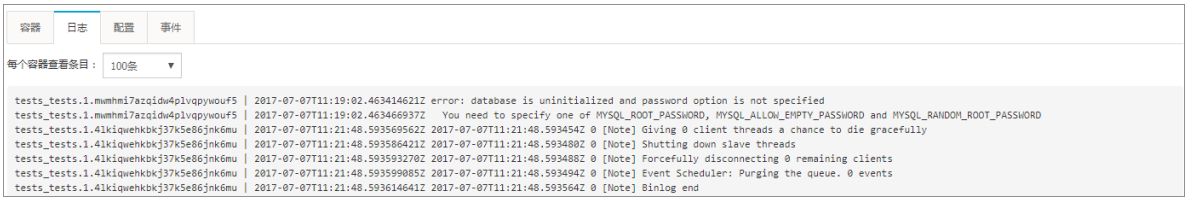
5. 您可以在服务详情页面查看该服务的所有容器。

图 253: 容器



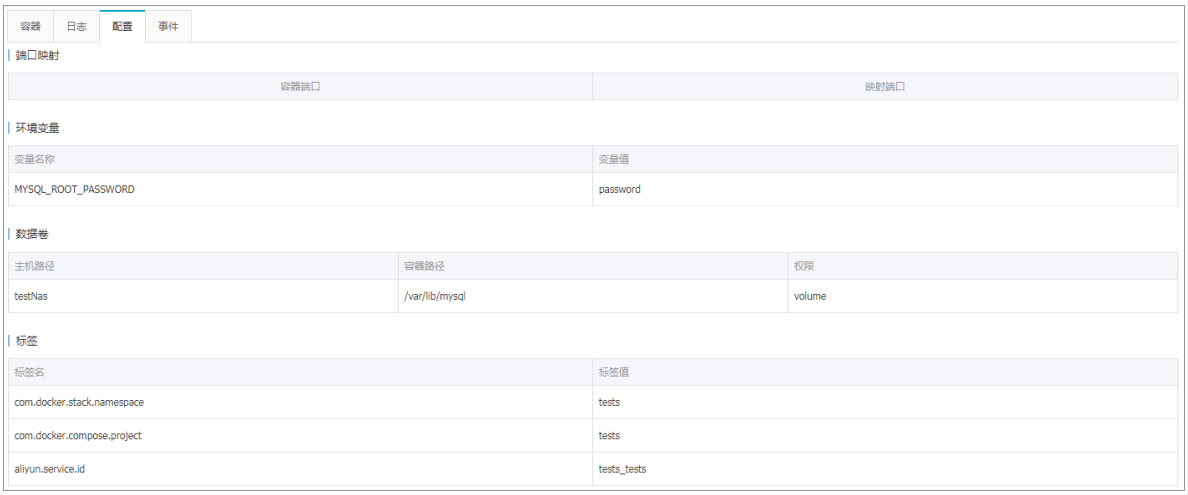
6. 单击**日志**查看服务级别的日志信息。

图 254: 日志



7. 单击**配置**查看服务的配置信息。

图 255: 配置



8. 单击**事件**查看服务的事件信息。

图 256: 事件



12.5.8.2 变更服务配置

您可以通过容器服务云控制台变更服务的配置。

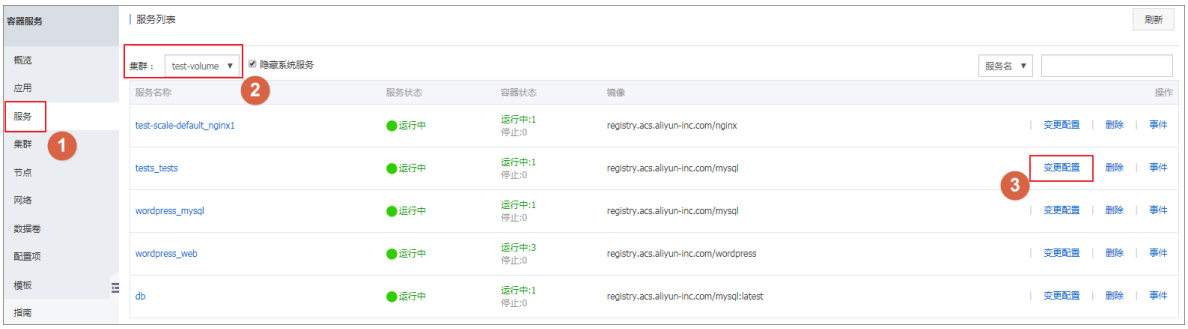
操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**服务**。
3. 选择所要查看的服务所在的集群。
4. 选择目标服务并单击**变更配置**。

图 257: 服务列表



5. 修改服务的配置并单击**更新**。

   有关参数的配置信息，参见使用镜像创建应用。

12.5.8.3 删除服务

您可以通过容器服务云控制台删除不再使用的服务。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**服务**。
3. 选择所要查看的服务所在的集群。
4. 选择所要删除的服务并单击**删除**。

   图 258: 删除服务



5. 在弹出的对话框中，单击**确定**。

12.5.9 管理网络

通过容器服务，您可以查看和管理您的应用的默认网络；此外，您还可以根据自己的需求创建自定义网络并将应用添加到自定义网络。

默认网络

当您创建集群时，容器服务会为系统服务创建默认网络以及ingress网络。

集群使用ingress网络进行负载均衡，将需要对外可用的服务暴露给集群。只有暴露端口的服务才需要ingress网络，不对外暴露端口的后端服务，其对应的容器也不会被加到ingress网络中。

当您创建应用时，如果您不指定将应用添加到自定义网络中，容器服务会为您的应用创建一个默认网络。应用的所有容器均被加入到该默认网络中。网络内的容器间可以通过使用内置的overlay驱动互相访问，也可以通过hostname（与容器的名称相同）进行服务发现。



说明: 您的应用的默认网络的名称为应用名_*default*。

自定义网络

除了使用应用的默认网络外，您还可以创建自定义网络。通过自定义，您可以创建更加复杂的网络拓扑并指定网络驱动和设置。

创建自定义网络后，您可以通过创建应用将应用添加到自定义的网络中。



说明:

- 您可以将应用添加到一个或多个自定义网络中。从而实现服务间的通信。
- 将应用添加到某一自定义网络中后，您无法更改应用的自定义网络。
- 将应用添加到某一自定义网络中后，您无法将应用从自定义网络中移除。

通过容器服务云控制台，您可以进行以下自定义网络的操作。

- 创建自定义网络
- 查看网络列表
- 将应用添加到自定义网络
- 查看容器的网络
- 删除自定义网络

跨主机互联的容器网络

处于同一网络中的应用可以通过容器的IP进行互相访问。您可以通过将应用添加到多个自定义网络来实现服务间的互相访问。

更多详细信息，参见 跨主机互联的容器网络。

12.5.9.1 跨主机互联的容器网络

容器服务为swarm mode集群创建的网络为集群范围的网络。处于同一网络内的容器可以通过容器IP进行互相访问。

示例

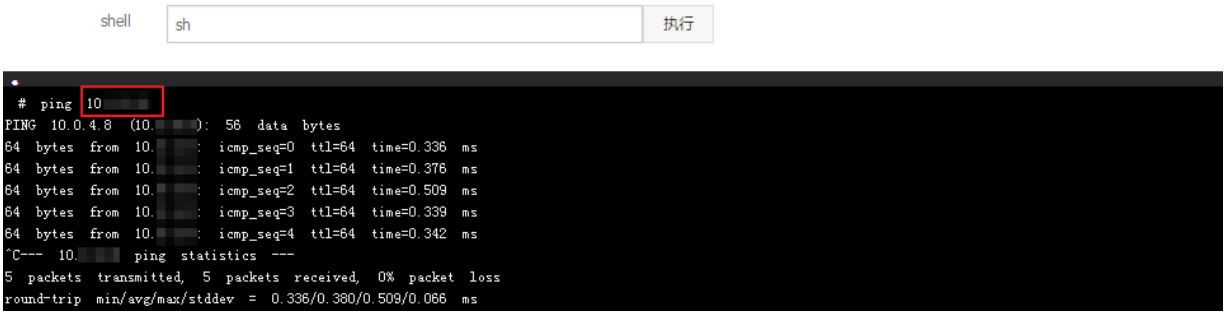
在本示例中，集群**swarmmode**中部署了两个应用**test1**和**test2**，部署在集群内不同的节点上。**test1**和**test2**都加入了网络swarmmode。

图 259: 容器列表



您可以通过连接远程终端在应用**test1**的容器内访问应用**test2**的容器的IP地址。

图 260: 访问



12.5.9.2 创建自定义网络

除了使用应用的默认网络，您还可以创建自定义网络。通过自定义网络您可以创建更加复杂的网络拓扑和网络设置。

操作步骤

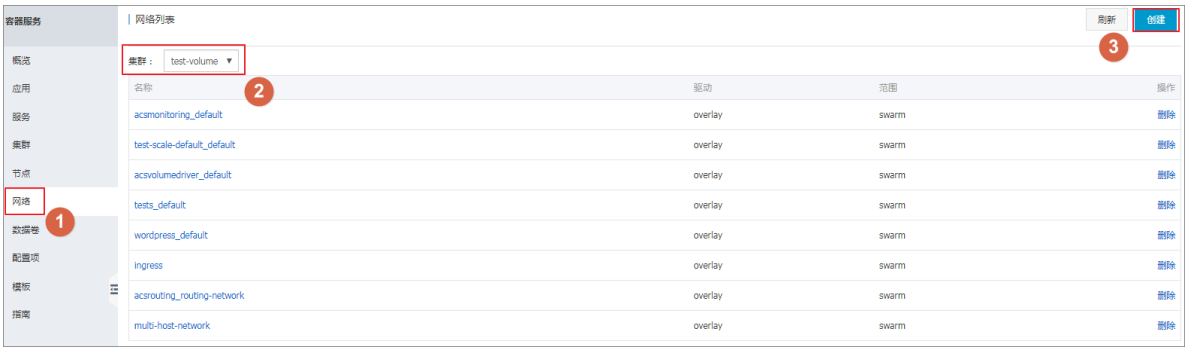
1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**网络**。

3. 选择所需的集群并单击右上角的**创建**。

图 261: 创建网络



4. 填写网络的设置信息并单击**确定**。

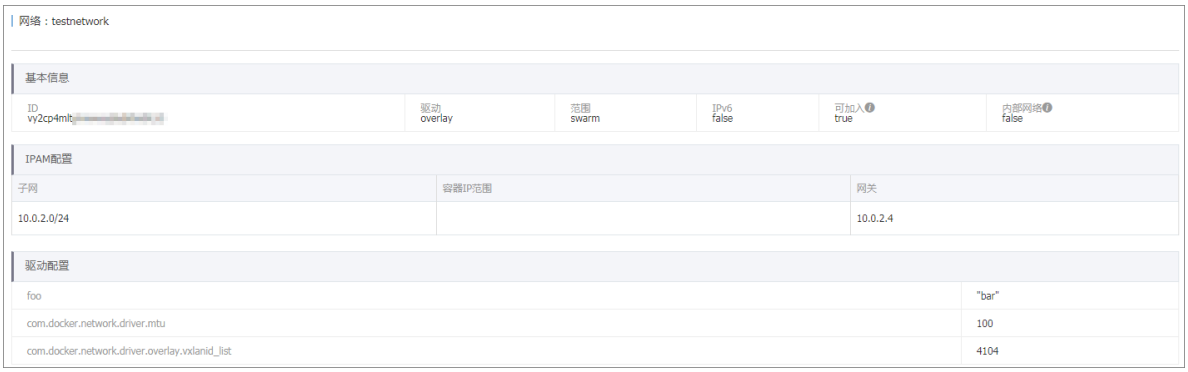
   - **名称**：所创建网络的名称。
   - **驱动**：网络所要使用的驱动。
   - **MTU**：可选项。设置网络包的最大值，单位为 byte。
   - **选项**：
     - **不同节点容器间通信加密**：容器跨节点进行通信时数据是否加密。
     - **允许其它容器加入网络**：是否允许将其它容器加入该网络。
     - **无法访问外部网络**：勾选后，将无法访问外部网络。
   - **IPAM**：可选项。设置自定义 IPAM 配置。
   - **其它配置**：可选项。设置网络驱动的相关参数项。具体可设置的参数参见所使用驱动的相关文档。

图 262: 填写网络参数



创建完成后，您可以在网络列表中单击网络的名称，查看该网络的详细信息。

图 263: 查看网络信息



12.5.9.3 查看网络列表

您可以在容器服务云控制台上查看集群的网络列表。

操作步骤

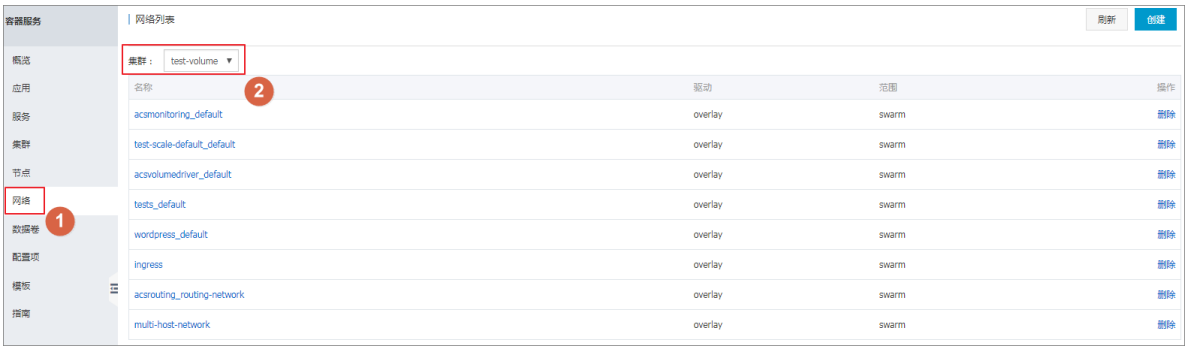
1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**网络**。

3. 选择所需的集群。

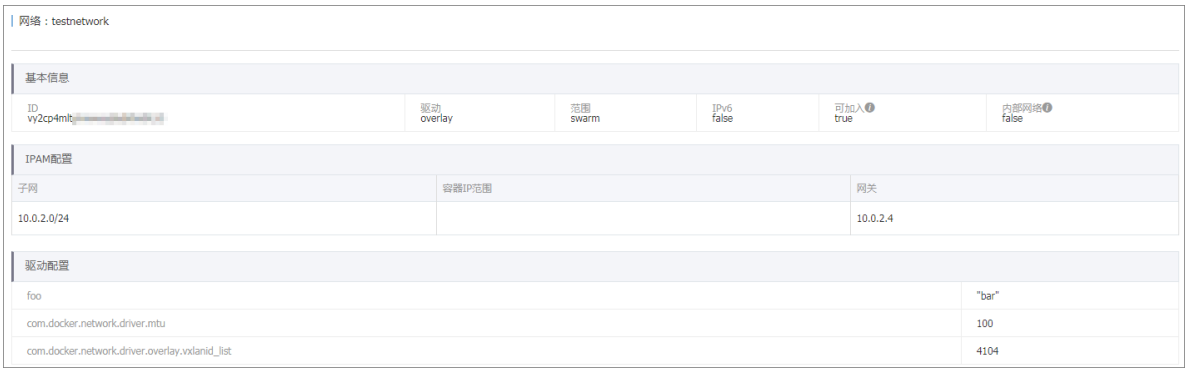
图 264: 网络列表



4. 选择要查看的网络并单击网络名称。

   您可以看到该网络的详细信息。

图 265: 查看网络信息



您可以单击左侧导航栏中的**容器列表**查看该网络下的容器。

图 266: 查看容器



12.5.9.4 添加应用到自定义网络

您可以在创建应用时通过指定应用的网络将服务添加到自定义的网络中。您可以将服务添加到一个或多个自定义网络中来实现服务间的互相访问。

使用限制

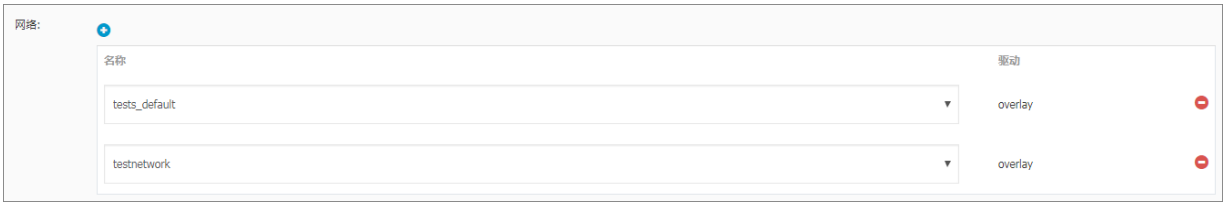
- 将应用添加到某一自定义网络后，您无法更改应用的网络。
- 将应用添加到某一自定义网络后，您无法将应用从该网络中移除。只有在删除应用时，应用才会从该网络中移除。

通过镜像创建应用

有关如何创建应用的详细信息，参见通过镜像创建应用。

设置应用配置时，在**网络**部分选择自定义网络的名称。下面的示例将应用添加到自定义网络tests_default和testnetwork中。

图 267: 添加应用到网络



通过编排模板创建应用

有关如何创建应用的详细信息，参见通过编排模板创建应用。

在编排模板中添加网络的相关信息。下面的示例将应用添加到自定义网络tests_default和testnetwork中。

编排模板：

```
version: '3.2'
services:
  test:
    image: 'wordpress:4'
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        failure_action: continue
      restart_policy:
        condition: none
    networks:
      - tests_default #您的自定义网络的名称
      - testnetwork #您的自定义网络的名称
networks: #声明您的自定义网络
  tests_default:
    external: true
  testnetwork:
    external: true
```

您需要对应用所要使用的自定义网络进行声明，否则部署应用时会报错。其中，external: true 表示自定义网络已创建，创建应用时容器服务不会尝试创建该自定义网络，而是查找该自定义网络并将应用添加至该自定义网络。

> **说明：** **external**不能与其它网络配置参数（**driver**、**driver_opts**、**ipam**、**internal**）一起使用。

12.5.9.5 查看容器的网络

您可以通过容器列表页面查看容器所处的网络。

操作步骤

1. 登录容器服务云控制台。

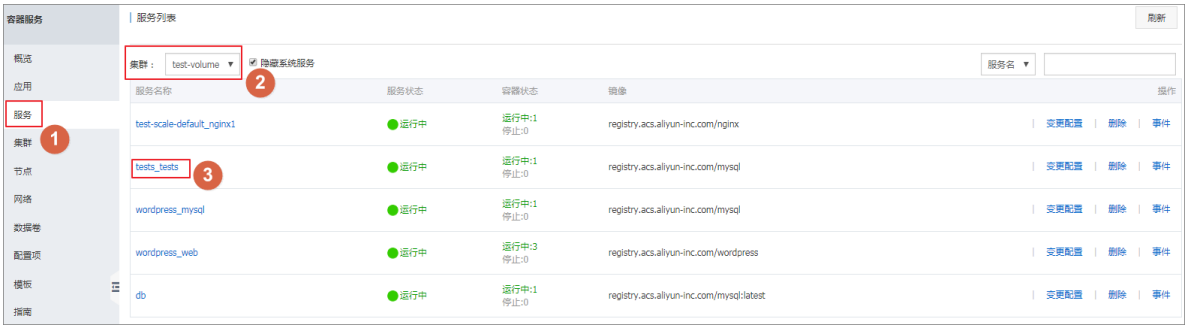
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**服务**。

3. 选择服务所在的集群。

4. 选择所要查看的服务并单击服务名称。

图 268: 选择服务



您可以在容器列表的容器IP一列查看容器所属的网络。

图 269: 查看容器网络



12.5.9.6 删除自定义网络

您可以删除不再使用的自定义网络。

使用限制

- 当自定义网络被应用引用时，您必须首先删除引用了自定义网络的应用，然后再删除自定义网络。
- 删除应用时，应用的默认网络会随应用一起删除，但是应用所引用的自定义网络不会随应用删除。

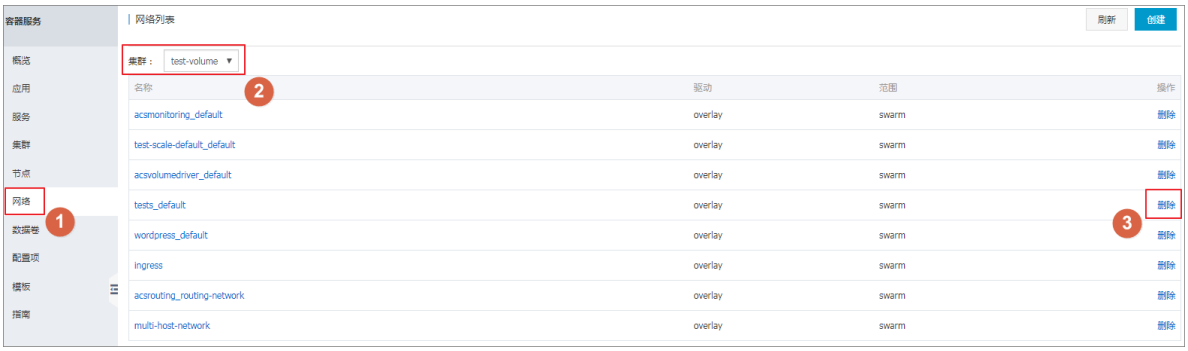
操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航中的**网络**。
3. 选择自定义网络所在的集群。
4. 选择您想要删除的自定义网络并单击右侧的**删除**。

图 270: 删除网络



5. 在弹出的确认对话框中，单击**确定**。

   如果自定义网络被服务引用，系统会报错。您需要先删除引用了该网络的服务然后再删除网络。

   图 271: 报错信息



12.5.10 管理密钥

12.5.10.1 概述

当您在容器服务中部署和编排服务时，您通常需要为服务设置一些敏感信息，例如：密码、TLS证书或者私钥。容器服务提供了一种更加安全的方式让您可以将这些敏感信息（密钥，即Docker中的secrets）存储在容器服务中。

以保存密码为例。与通过配置服务使用存储在环境变量中的明文密码不同，您可以通过容器服务云控制台创建secret 来保存密码。当您创建应用时，您可以配置服务使用所创建的secret。部署应用时，容器服务会基于secret创建一个文件并将该文件挂载在运行该服务的容器的/run/secrets/<secret_name>目录中。您的服务可以访问该文件。

说明: 只有被赋予访问权限并且在运行中的容器可以访问指定的secrets。

您可以赋予一个服务访问一个或多个secrets的权限。此外，您还可以通过变更应用配置来赋予服务访问其他secrets的权限，或者取消服务访问某一secret的权限。

当容器上的服务停止运行时，容器服务会解除secrets和内存文件系统的挂载并从节点的内存上移除secrets。

通过容器服务云控制台，您可以进行以下secret操作。

- 创建密钥
- 查看密钥列表
- 在服务中使用密钥
- 更新服务的密钥
- 删除密钥

12.5.10.2 创建密钥

您可以通过容器服务云控制台创建secret。

使用限制

secrets必须与使用它的应用位于同一集群。

操作步骤

1. 登录容器服务云控制台

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**
3. 选择所需的集群并单机右侧的**管理**

图 272: 集群管理



4. 单击左侧导航栏中的**密钥管理**。

5. 单击页面右上角的**创建**。

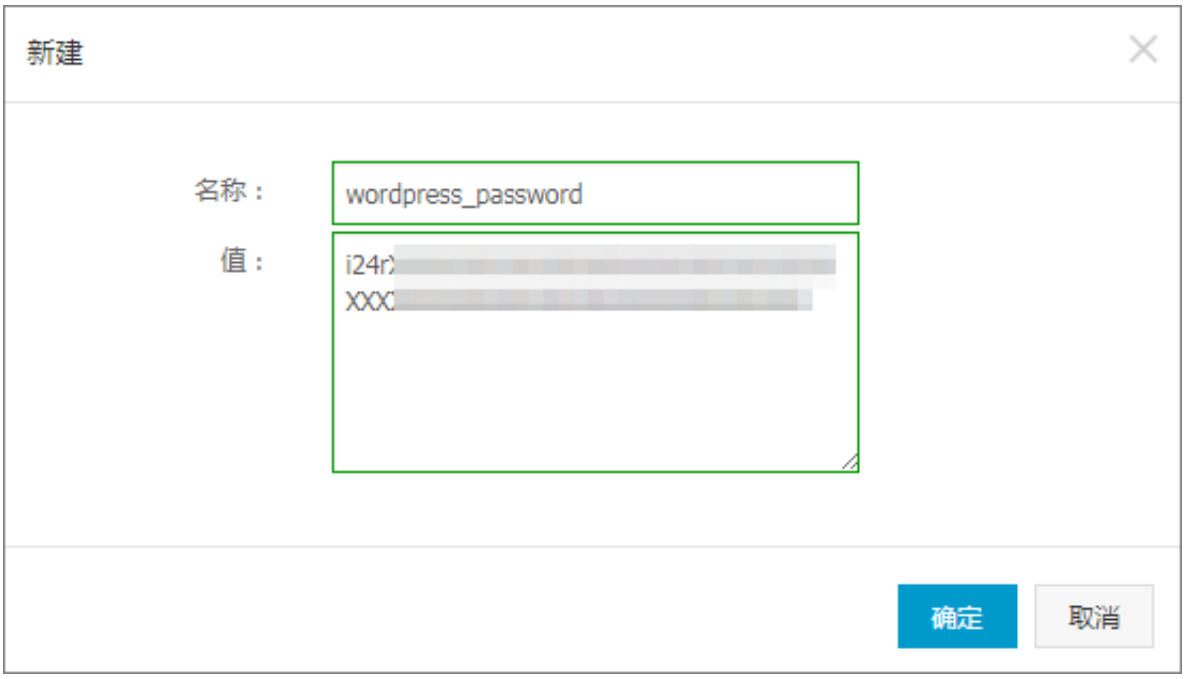
图 273: 创建密钥



6. 填写密钥的名称和值，并单击**确定**。

名称包含大小写字母、数字、连字符（-）和下划线（_），必须以大小写字母或数据开头和结尾，且长度不能超过64个字符。

图 274: 密钥信息



12.5.10.3 查看密钥列表

您可以通过容器服务云控制台查看密钥列表。

操作步骤

1. 登录容器服务云控制台。

有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。

3. 选择所需的集群并单击右侧的**管理**。

图 275: 集群管理



4. 单击左侧导航栏中的**密钥管理**。

您可以看到所选集群中的所有密钥。

图 276: 查看密钥



12.5.10.4 在服务中使用密钥

您可以通过创建应用来指定服务中使用的secrets。您可以为服务赋权使用一个或多个secrets。

前提条件

您已经创建secrets。有关如何创建secrets，参见创建密钥。

使用限制

secrets必须和使用它的应用位于同一集群内。

通过镜像创建应用

有关如何创建应用，参见使用镜像创建应用。

本示例创建名为**mysql** 的MySQL应用。在应用配置的**密钥**部分设置secrets信息。本示例赋权应用使用两个secrets：**my_mysql_password_v1**和**my_secret_v1**。

图 277: 使用密钥

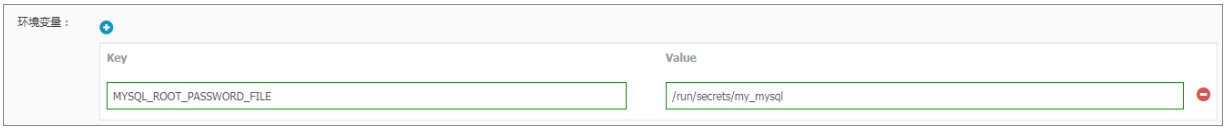


其中：

- **密钥名称**：您在容器服务中所创建的密钥的名称。
- **目标名称**：将要挂载到容器的*/run/secrets/*目录下的文件的名称。默认情况下，与source值相同，即容器在*/run/secrets/secret_name*目录下访问密钥。
- **用户ID**：拥有容器*/run/secrets/*目录下的 secret 文件的用户ID。默认为0。
- **用户组ID**：拥有容器*/run/secrets/*目录下的secret文件的用户组ID。默认为0。
- **权限**：将要挂载到容器*/run/secrets/*目录下的secret文件的权限。格式为八进制，例如：0444表示全局可读。由于secrets是挂载在临时文件系统上的，所以secrets不可写；因此，如果您设置了可写位，系统会忽视您的设置。如果您不熟悉UNIX的权限模式，可以使用*Permission Calculator*。

此外，本示例中您还需要设置环境变量MYSQL_ROOT_PASSWORD_FILE来配置MySQL应用使用*/run/secrets/my_mysql*文件作为root密码。

图 278: 环境变量



通过编排模板创建应用

有关如何创建应用，参见通过编排模板创建应用。

您可以使用short syntax或者long syntax 来在编排模板中进行secrets的设置。

short syntax

使用short syntax，您仅需要指定secret名称。容器服务会赋予容器访问secret的权限并将secret挂载到容器的*/run/secrets/<secret_name>*目录。

本示例创建名为**mysqlshort**的应用。在编排模板中添加密钥的信息。本示例赋权应用使用两个secrets：**my_mysql_password_v1**和**my_secret_v1**。

编排示例：

```
version: '3.2'
services:
  mysqlshort:
    image: 'mysql:5.7'
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        failure_action: continue
      restart_policy:
        condition: none
    environment:
      - MYSQL_ROOT_PASSWORD=/run/secrets/my_mysql_password_v1 #配置 MySQL 应用使用 /run/secrets/my_mysql_password_v1 文件作为 root 密码
    secrets:
      - my_mysql_password_v1 #您所创建的 secret 的名称
      - my_secret_v1 #您所创建的 secret 的名称
secrets: #对 secrets 进行声明
  my_mysql_password_v1:
    external: true
  my_secret_v1:
    external: true
```

您需要在编排模板中对secrets进行声明，否则secrets无法生效。其中，external：true表示密钥已经创建了，容器服务不会尝试创建密钥而是寻找密钥并将其用于服务上。

long syntax

通过使用long syntax，您可以指定secrets的相关参数。

本示例创建名为**mysqllong**的应用。在编排模板中添加密钥的信息。本示例赋权应用使用两个secrets：**my_mysql_password_v1**和**my_secret_v1**

编排示例：

```
version: '3.2'
services:
  mysqllong:
    image: 'mysql:5.7'
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        failure_action: continue
      restart_policy:
        condition: none
    environment:
      - MYSQL_ROOT_PASSWORD=/run/secrets/my_mysql#配置 MySQL 应用使用 /run/secrets/my_mysql_password_v1 文件作为 root 密码
    secrets:
      - source: my_mysql_password_v1 #您所创建的 secret 的名称
        target: my_mysql
        uid: '0'
        gid: '0'
```

```
        mode: 444
      - source: my_secret_v1 #您所创建的 secret 的名称
        target: my_secret
        uid: '0'
        gid: '0'
        mode: 444
secrets: #对 secrets 进行声明
  my_mysql_password_v1:
    external: true
  my_secret_v1:
    external: true
```

其中：

- **source**：您在容器服务中所创建的密钥的名称。
- **target**：将要挂载到容器的*/run/secrets/*目录下的文件的名称。默认情况下，与**source**值相同，即容器在*/run/secrets/secret_name*目录下访问密钥。
- **uid**：拥有容器*/run/secrets/*目录下的secret文件的用户ID。默认为0。
- **gid**：拥有容器*/run/secrets/*目录下的secret文件的用户组ID。默认为0。
- **mode**：将要挂载到容器 */run/secrets/*目录下的secret文件的权限。格式为八进制，例如：0444表示全局可读。由于secrets是挂载在临时文件系统上的，所以secrets不可写；因此，如果您设置了可写位，系统会忽视您的设置。如果您不熟悉UNIX的权限模式，可以使用*Permission Calculator*。

您需要在编排模板中对secrets进行声明，否则secrets无法生效。其中，`external：true`表示密钥已经创建了，容器服务不会尝试创建密钥而是寻找密钥并将其用于服务上。

12.5.10.5 更新服务的密钥

secret创建后，其包含的数据是无法更改的。因此，如果您的secret泄露了或者您需要使用新的数据，您需要重新创建一个secret并通过变更应用配置来更新服务使用新的secret。

本示例以应用**mysqlshort** （目前使用的secrets为**my_mysql_password_v1**和**my_secret_v1**）为例。

操作步骤

1. 在容器服务云控制台创建新的secret **secret my_mysql_password_v2**。

   有关如何登录云控制台，参见登录云控制台。

   有关如何创建 secret，参见创建密钥。

   说明: 所创建的secret必须与应用位于同一集群中。

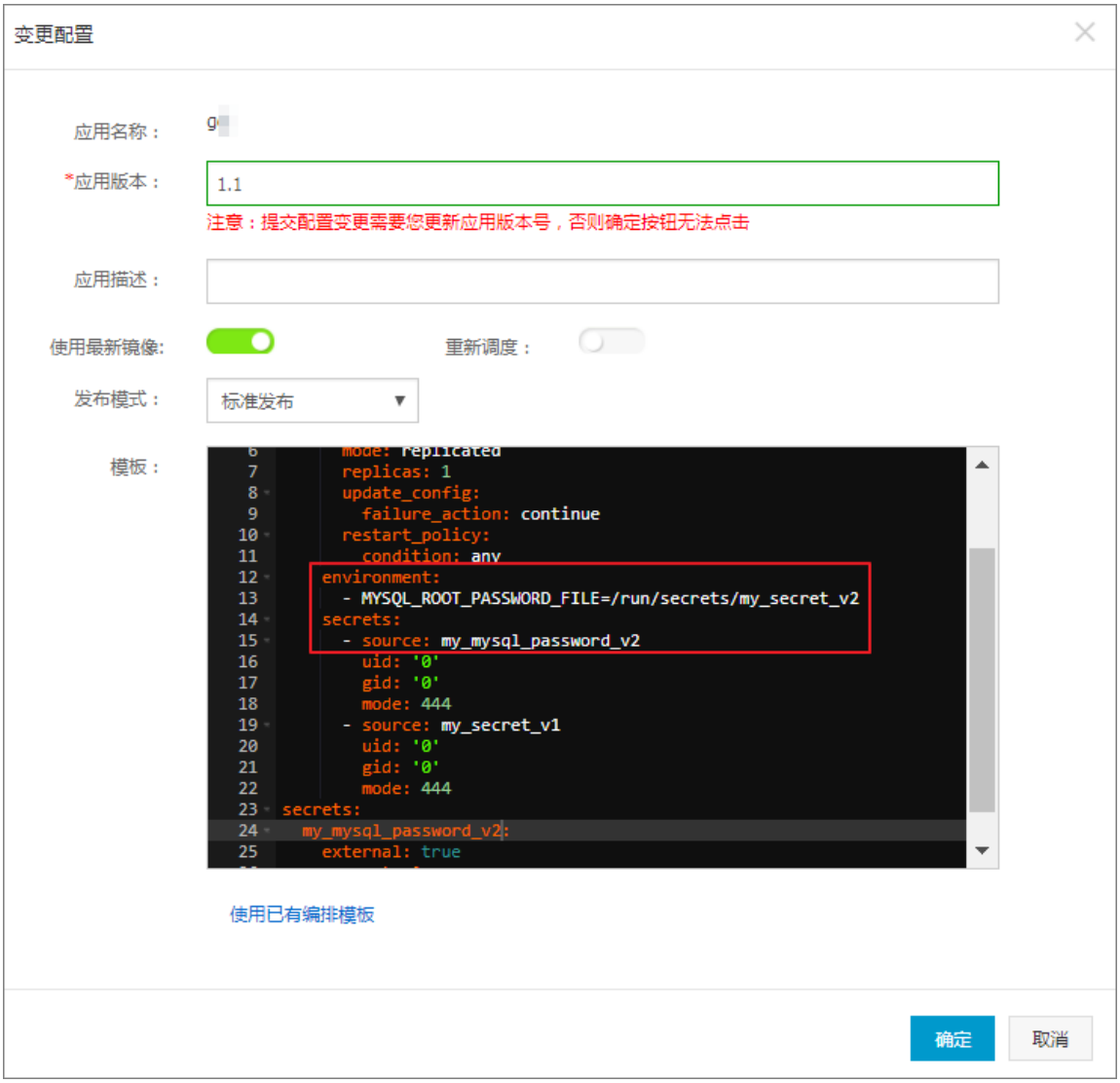
2. 变更应用配置来更新服务使用的secret。

   有关如何变更应用配置，参见变更应用配置。

   您可以通过以下两种方法之一来修改服务的secret信息。

   - 修改secret的名称并修改环境变量MYSQL_ROOT_PASSWORD_FILE。

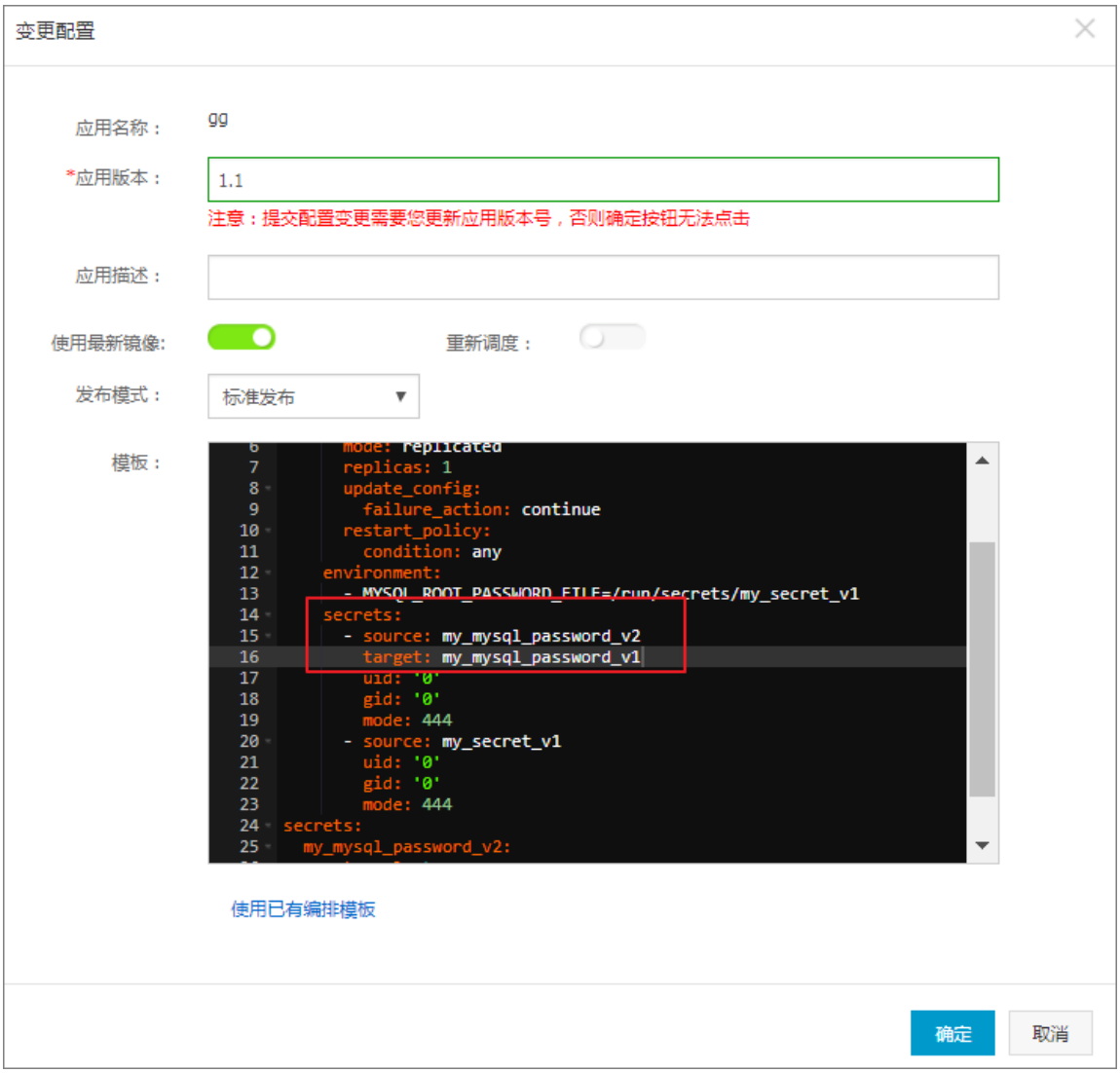
     图 279: 修改服务的secret



   - 修改secret的名称并设置target为my_mysql_password_v1。

     这样设置的话带有secret my_mysql_password_v2数据的文件将会挂载到 */run/secrets/my_mysql_password_v1*，您不需要再修改环境变量MYSQL_ROOT_PASSWORD_FILE。

     图 280: 修改服务的secret



后续操作

将secret **my_mysql_password_v1**从服务中移除后，您可以通过容器服务云控制台删除该secret。

有关操作，参见删除密钥。

12.5.10.6 删除密钥

您可以在容器服务云控制台上删除不再使用的secrets。

使用限制

您无法删除被服务引用的secrets。您需要先把secrets是从服务中移除然后再删除secrets。

操作步骤

1. 登录容器服务云控制台。

   关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。
3. 选择所需的集群并单击右侧的**管理**。

图 281: 集群管理



4. 单击左侧导航栏中的**密钥管理**。
5. 选择需要删除的secret并单击**删除**。

图 282: 删除密钥



6. 在弹出的确认对话框里单击**确定**。

   如果该secret被某些服务引用，删除时会报错，您需要先通过变更应用配置将secret从服务中移除，然后再删除secret。有关如何将secret从服务中移除，参见更新服务的密钥。

12.5.11 管理数据卷

Docker的特性决定了容器本身是非持久化的，容器被删除后其中的数据也会被一并删除。虽然Docker提供的数据卷（Volume）可以通过挂载宿主机上的目录来实现持久存储，但在集群环境中，宿主机上的数据卷有很大的局限性。

- 容器在机器间迁移时，数据无法迁移
- 不同机器之间不能共享数据卷

为了解决这些问题，阿里云容器服务提供第三方数据卷，将各种云存储包装成数据卷，可以直接挂载在容器上，并在容器重启、迁移时自动重新挂载。目前支持OSSFS和NAS存储。

12.5.11.1 开通数据卷功能

前提条件

您的集群里必须部署了acsvolumedriver应用，才可以开通数据卷功能。

操作步骤

1. 登录容器服务云控制台。

   关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**数据卷**，开通数据卷功能。

12.5.11.2 创建OSSFS数据卷

OSSFS是阿里云官方提供的基于FUSE的文件系统。OSSFS数据卷可以将OSS的Bucket包装成数据卷。由于数据需要经过网络同步到云端，OSSFS在性能和功能上与本地文件系统有差距。请不要把数据库等重IO应用、日志等需要不断改写文件的应用运行在OSSFS上。

背景信息

OSSFS和本地文件系统具体差异如下所示：

- 随机或者追加写文件会导致整个文件的重写。
- 因为需要远程访问OSS服务器，元数据操作（例如list directory）性能较差。
- 文件/文件夹的重命名操作不是原子的。
- 多个客户端挂载同一个OSS Bucket时，需要您自行协调各个客户端的行为。例如，避免多个客户端写同一个文件等等。
- 不支持硬链接（hard link）。

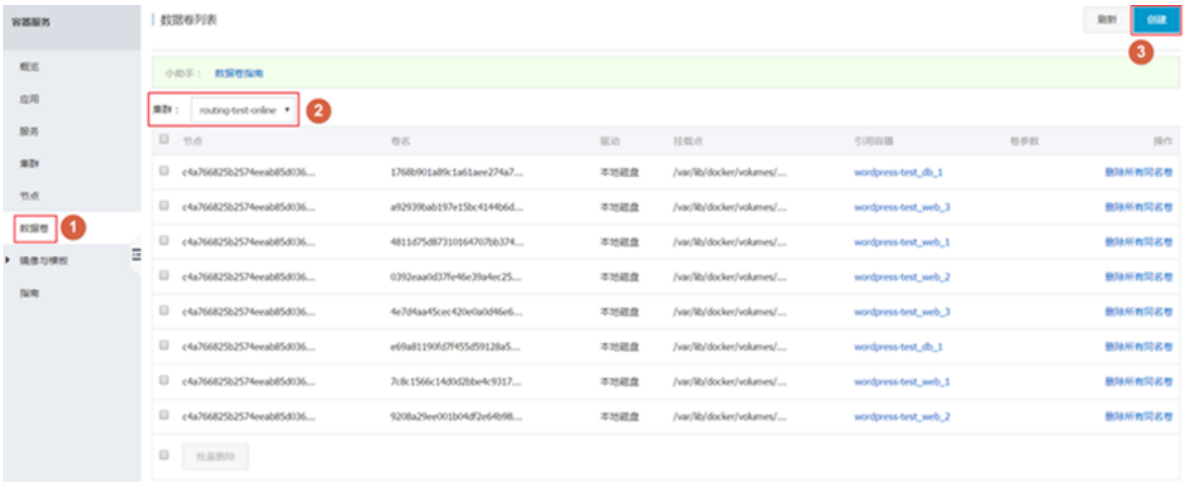
操作步骤

1. 登录容器服务云控制台。

   关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**数据卷**。

3. 选择需要创建数据卷的集群并单击页面右上角的**创建**。



4. 在弹出的对话框中，设置数据卷参数并单击**创建**。容器服务会在集群的所有节点上创建名称相同的数据卷。

   - **数据卷名**：数据卷的ID。数据卷名在集群内必须唯一。
   - **AccessKey ID**、**AccessKey Secret**：访问OSS所需的AccessKey。您可以从云平台的 **云资源中心** > **云产品管理** >**对象存储**获取账号的Access Key。
   - **访问域名**：如果Bucket和ECS实例位于同一个地域（Region），选择 **内网域名**；否则，选择**外网域名**。
   - **文件缓存**：如果需要在不同机器间同步同一个文件的修改（比如在机器A中修改文件，在机器B中读取修改后的内容），请关闭文件缓存。

     说明: 关闭文件缓存将导致ls文件夹变得很缓慢，尤其是同一个文件夹下文件比较多时。因此，没有上述需求时，请打开文件缓存，提高ls的速度。

12.5.11.3 创建NAS数据卷

您可以通过容器服务云控制台将您自己的NAS存储创建为数据卷。

操作步骤

1. 登录容器服务云控制台。

   关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**数据卷**。
3. 选择需要创建数据卷的集群并单击页面右上角的**创建**。
4. 在弹出的对话框中，设置数据卷参数并单击**创建**。容器服务会在集群的所有节点上创建名称相同的数据卷。

   - **数据卷名**：数据卷的名称。数据卷名称在集群内必须唯一。

- **文件系统ID**：NAS文件系统的ID。
- **挂载点域名**：集群在NAS文件系统中挂载点的挂载地址。
- **子目录**：NAS路径下的子目录，以/开头，设定后数据卷将挂载到指定的子目录。
  - 如果NAS根目录下没有此子目录，会默认创建后再挂载。
  - 您可以不填此项，默认挂载到NAS根目录。
- **权限**：设置挂载目录的访问权限，例如：755、644、777等。
  - 只有挂载到NAS子目录时才能设置权限，挂载到根目录时不能设置。
  - 您可以不填此项，默认权限为NAS文件原来的权限。

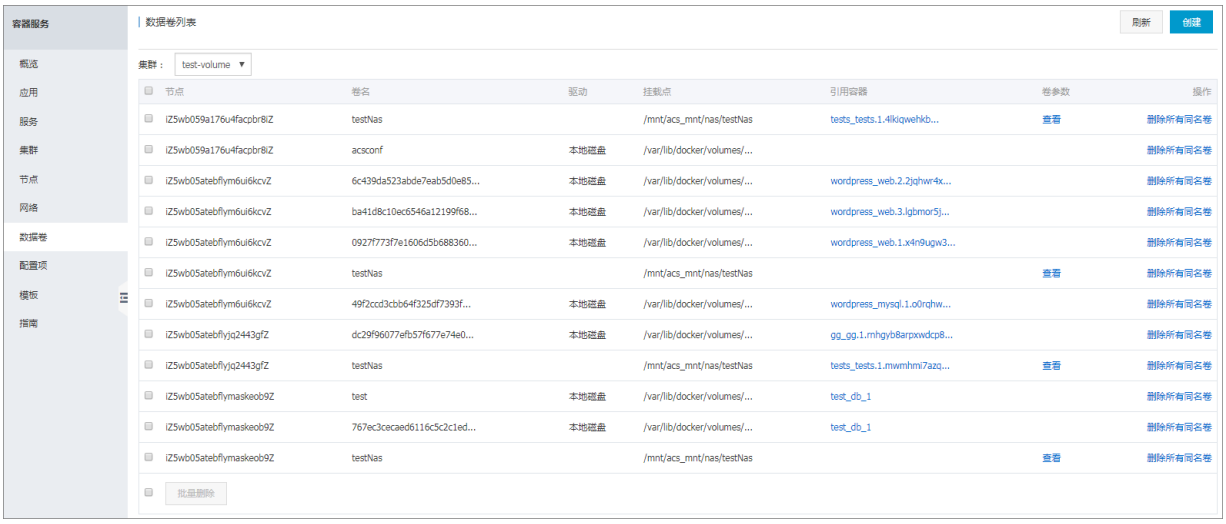
12.5.11.4 查看数据卷

登录容器服务云控制台并单击左侧导航栏中的**数据卷**。

关如何登录云控制台，参见登录云控制台。

数据卷列表页面列出了当前集群中所有的数据卷，包括本地数据卷和第三方数据卷。

图 283: 数据卷列表



对于本地数据卷，数据卷名称的格式为*node_name/volume_name*。

对于第三方数据卷，单击**查看**可以查看数据卷的参数。

创建第三方数据卷时，容器服务会在集群的所有节点上以同一数据卷名称创建该数据卷，便于容器在不同节点间迁移。

您可以单击**删除所有同名卷**删除所有使用该名称的数据卷。

12.5.12 管理日志

您可以通过容器服务云控制台查看应用日志，服务日志和容器日志。

12.5.12.1 查看集群日志

您可以查看集群的日志信息。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**集群**。

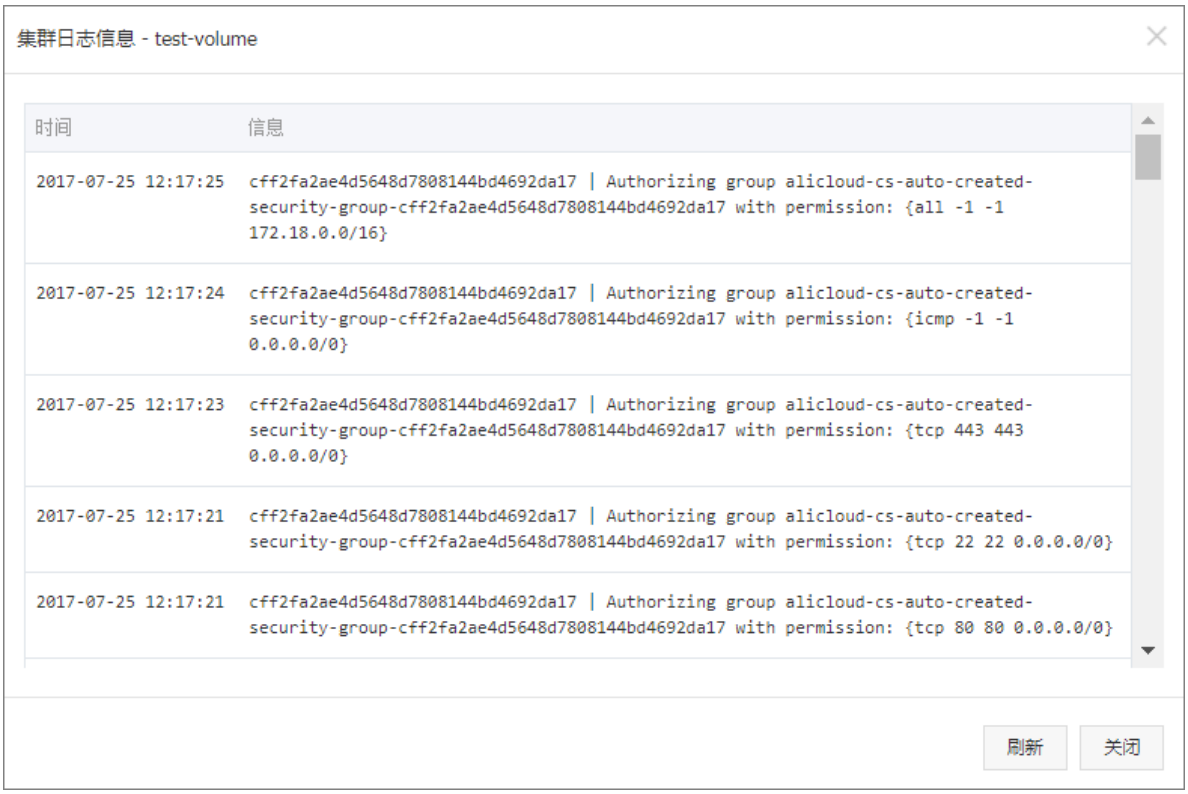
3. 选择要查看的的集群并单击**查看日志**。

图 284: 选择集群



查看该集群的日志信息。

图 285: 集群日志



12.5.12.2 查看应用日志

您可以查看应用的日志信息。

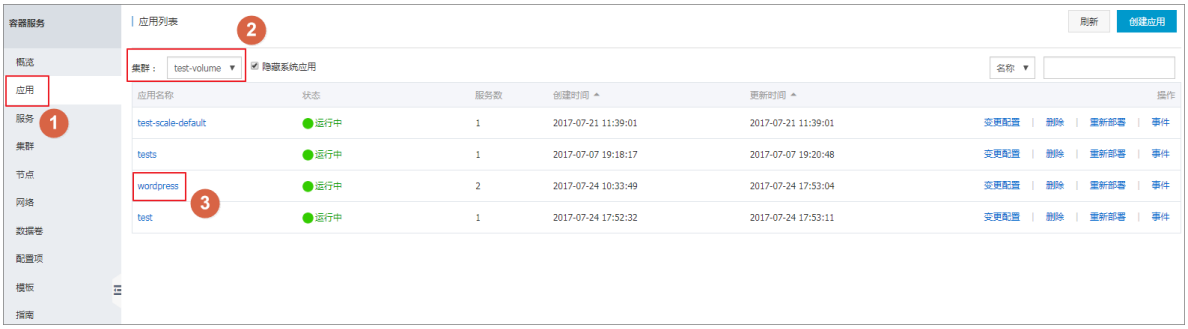
操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**应用**。
3. 选择要查看的应用所在的集群。
4. 单击要查看的应用的名称。

图 286: 应用列表



5. 单击**日志**查看该应用的日志信息。

   您可以选择显示的日志条目，也可以选择下载所有日志到本地。

图 287: 应用日志



12.5.12.3 查看服务日志

您可以查看服务的日志信息。

操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**服务**。

3. 选择要查看的服务所在的集群。

4. 单击要查看的服务的名称。

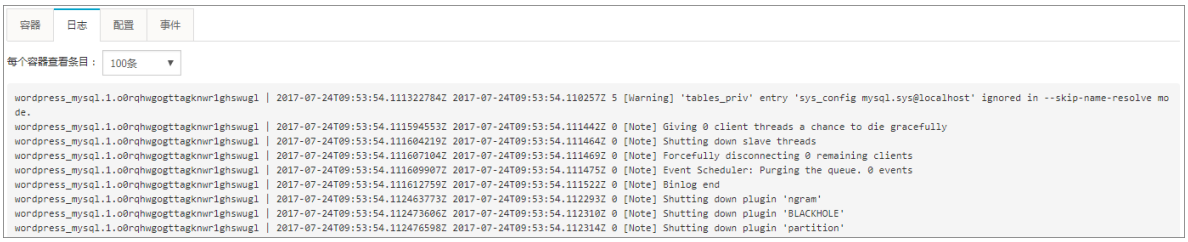
图 288: 查看服务详情



5. 单击**日志**查看该服务的日志信息。

   您可以选择显示的日志条目，也可以选择下载所有日志到本地。

图 289: 服务日志



12.5.12.4 查看容器日志

您可以查看容器的日志信息。

操作步骤

1. 登录容器服务云控制台。

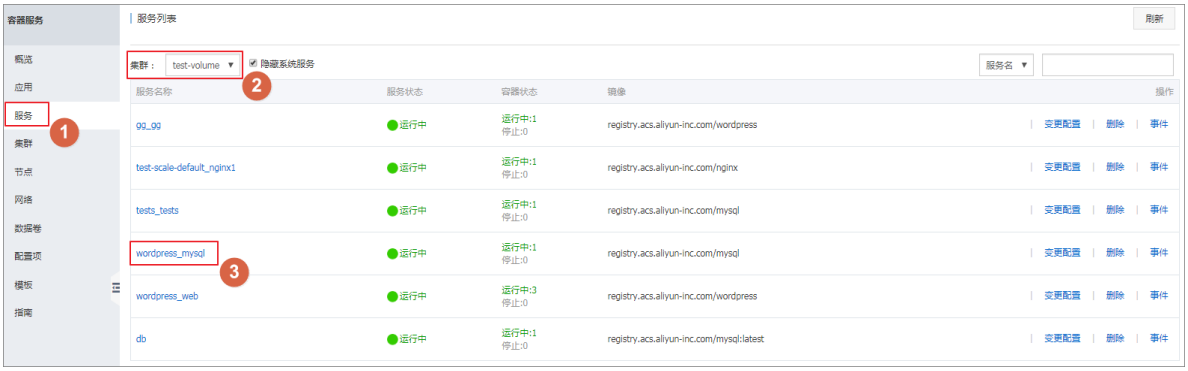
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的 **服务**。

3. 选择要查看的服务所在的集群。

4. 单击要查看的服务的名称。

图 290: 查看服务详情



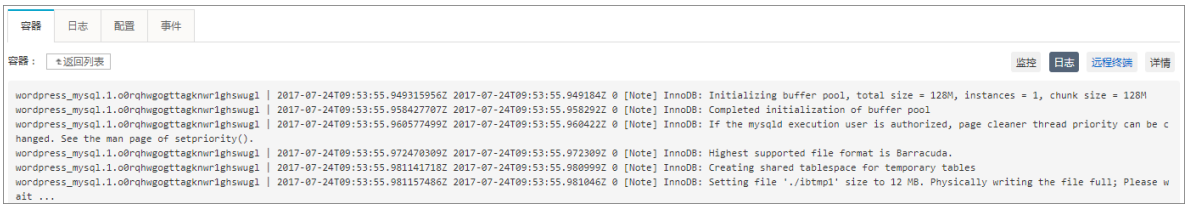
5. 选择所要查看的容器并单击**日志**。

图 291: 容器列表



您可以查看该容器的日志信息。

图 292: 容器日志



12.5.13 使用访问控制

简介

如果您使用容器服务创建了多个集群，您的组织里有多个用户需要使用这些集群。如果这些用户共享使用您的云账号秘钥，就会出现以下的问题：

- 您的秘钥由多人共享，泄密风险高。
- 您无法限制其他人的访问权限，容易出现误操作导致安全风险。

访问控制RAM是阿里云提供的资源访问控制服务。通过RAM，您可以集中管理您的用户，以及控制用户可以访问您名下哪些资源的权限。对于容器服务而言，90%以上的用户主要分配的是集群维度的权限，主要是对集群维度的是否可增删的限制。因此为了更简单地使用权限控制，容器服务提供了两种鉴权策略，一种是资源授权，一种是细粒度API授权。对于90%的用户而言使用资源分配鉴权策略即可满足需求。对于RAM有深入的了解希望在API级别进行鉴权的用户可以使用API细粒度鉴权。

使用场景

假设是一家APP开发的公司，主要使用阿里云的服务提供APP的后端服务与基础设施和中间件，公司中的成员有后端开发工程师，测试工程师，运维工程师、基础平台工程师。从职责分配上来讲：

- 后端开发工程师负责开发APP的后端服务。
- 测试工程师负责测试APP的后端服务和基础的测试环境维护。
- 运维工程师负责所有的环境的运维（集群的伸缩，监控等）。
- 基础平台工程师负责中间件的维护与基础设计的建设，包括GitLab、Jenkins、Sentry等。

从职责划分上来看，不同的角色对于资源的需求不同，为了保证每个角色有足够的权限完成自己的任务，同时权限不越界。在这个场景中您使用资源分配鉴权即可。

主账号授权

说明: 当您进入RAM控制台的时候，可能会发现一些以acs_开头的RAM子账号，这些子账号是容器服务自动创建的子账号，请不要删除。

1. 登录访问控制云控制台，创建一个子账号并开启控制台登录。

   有关如何登录云控制台，参见登录云控制台。

2. 登录容器服务云控制台，单击左侧导航栏中的**集群**并单击**子账号授权**。
3. 选择相应的子账号并单击**下一步**。

图 293: 选择子账号



4. 选择对应的集群资源和权限，单击**下一步**并在弹出的确认对话框中单击**确定**。

图 294: 子账号授权



5. 单击**完成**完成授权。

使用子账号

使用子账号登录（要主账号进入RAM控制台并将控制台登录地址告知子用户）并进入容器服务云控制台。

此时，集群列表中显示被分配的集群。

深入理解鉴权策略

鉴权策略简单地来讲就是对云资源的使用权限的校验。鉴权分为两种，一种是资源分配鉴权，一种是API细粒度鉴权。

API细粒度鉴权是基于访问控制RAM进行的鉴权，相对来讲更加复杂。

资源分配鉴权是容器服务在API细粒度鉴权之上的包装，主要将API细粒度鉴权无法实现的资源分配在容器服务中进行了实现，并将API细粒度鉴权的鉴权策略封装为只读和读写两个权限。只读权限可以创建应用、删除应用等等，但是无法对集群级别的增删改查进行操作。而读写权限则可以对集群进行完全的管理。

高级用法

API细粒度鉴权可以实现API级别的鉴权。目前提供的后置鉴权条件如下所示。

鉴权 Action 名称	解释
GetClusterById	获取集群描述
GetClusterCerts	获取集群证书
CreateCluster	创建集群
DeleteCluster	删除集群
UpdateClusterSizeById	集群扩容

资源列表如下所示。

资源	解释
cluster	集群

参考示例：

```
{
  "Statement": [
    {
      "Action": "cs:*",
      "Effect": "Allow",
      "Resource": "acs:cs:*:*:cluster/cluster_id1"
    },
    {
      "Action": "cs:*",
      "Effect": "Allow",
      "Resource": "acs:cs:*:*:cluster/cluster_id2"
    }
  ],
  "Version": "1"
}
```

本示例表示授予集群的全部权限。

12.5.14 服务发现和负载均衡

服务发现和负载均衡主要解决通信的可靠性问题。为了达到可靠性，容器服务引入了负载均衡机制。通信又可以分为对外暴露服务的通信和内部服务之间的通信。下面根据场景引导您使用不同的解决方案。

场景一

普通且简单的7层协议负载均衡，Web服务的反向代理，推荐使用简单路由服务。更多详细信息，参见简单路由#支持 *HTTP/HTTPS#* ， 简单路由-域名配置，简单路由-*HTTP*协议变为*HTTPS*协议。

场景二

4 层协议的负载均衡，负载均衡直接负载均衡到多个相同功能的容器，在将传统架构迁移到容器架构过程中非容器集群的服务访问容器集群中容器的服务，推荐使用负载均衡路由。

场景三

同一个集群内，服务间需要相互发现和相互进行通信，且需要负载均衡的能力，推荐使用集群内服务间路由和负载均衡。

场景四

同一个集群内，服务间需要相互发现和相互进行通信，但是不需要负载均衡的能力，推荐使用服务发现。

12.5.14.1 简单路由（支持 HTTP/HTTPS）

适用场景

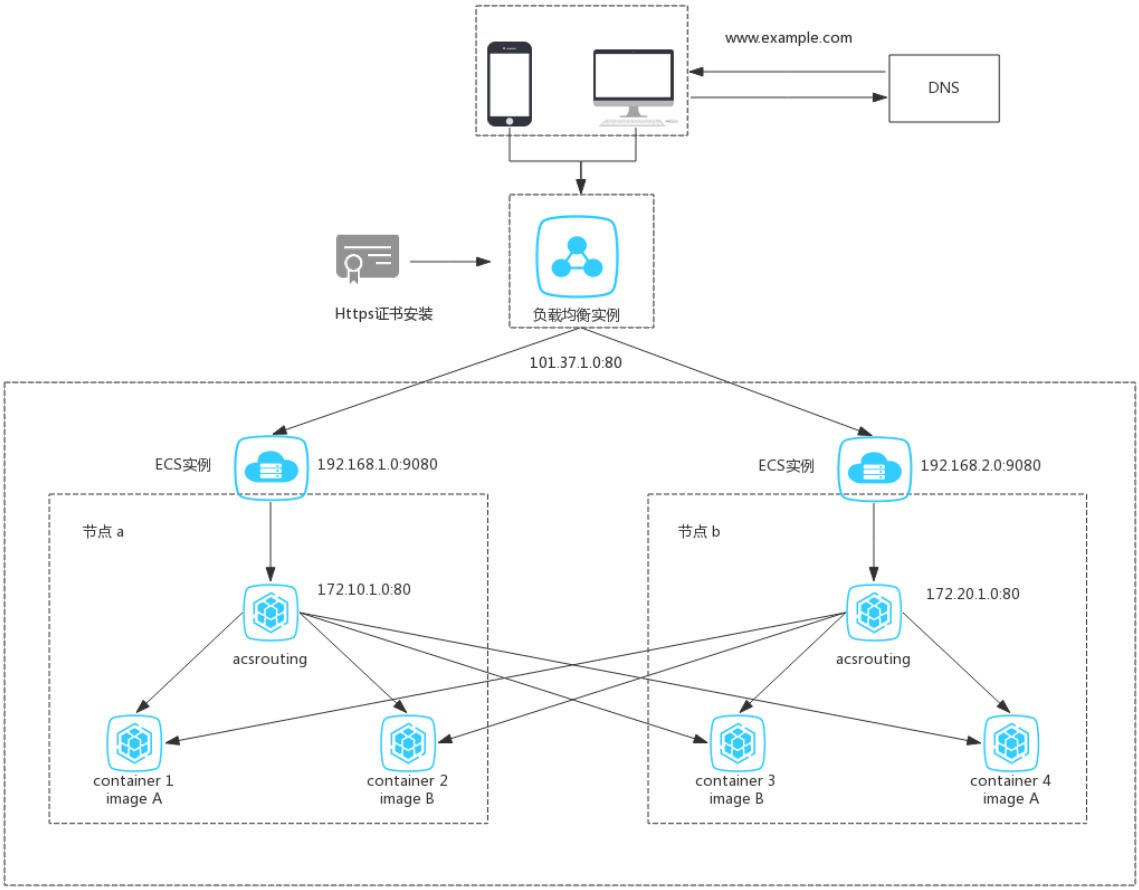
普通且简单的7层协议负载均衡，Web路由服务，容器集群内服务之间7层协议互相访问的通信代理和负载均衡。

原理

当您新建一个集群的时候，会默认给这个集群分配一个负载均衡实例。该负载均衡实例会将集群中的所有节点加入作为后端，同时前端会暴露80端口，后端所有节点的机器会暴露9080端口。容器服务会启动一个路由应用acsrouting，即阿里云容器服务路由应用（Alibaba Cloud Container Service Routing）。该路由应用只有一个服务，即路由服务。该服务是全局（global）的，即每个

节点（下面说到的主机和节点都是同一个意思，即ECS的VM实例）都部署了该服务（或者说镜像）的一个拷贝，也就是容器。每个节点都由这个容器用来路由HTTP服务或者HTTPS服务。

图 295: 简单路由原理



HTTP服务，负载均衡实例的前后端端口的映射为80:9080，主机与路由容器之间的端口映射为9080:80，即路由的容器暴露80端口，其他用作Web服务的容器可以暴露任意的端口。只要在容器启动的时候设置主机和容器端口的映射，routing服务就能获取到相应的端口进行请求的路由。

通过容器服务云控制台进行设置

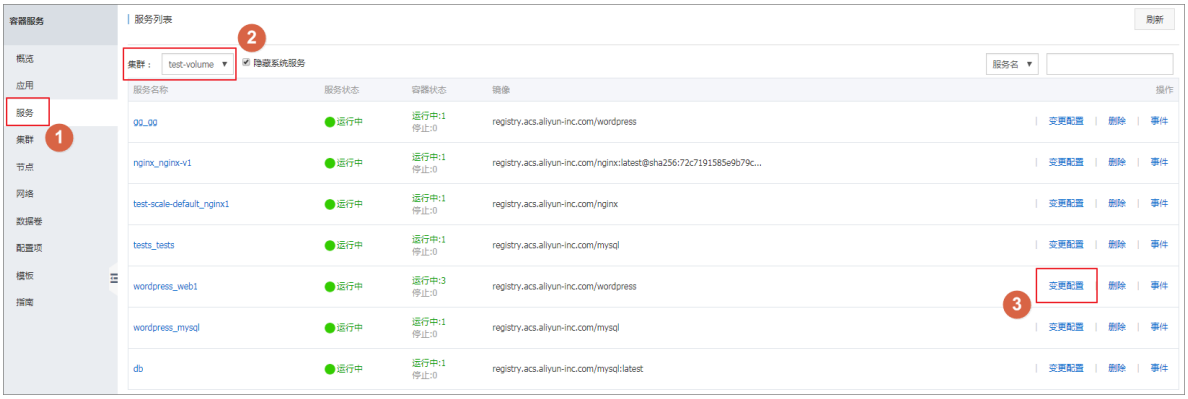
通过服务 > 变更配置进行设置

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。
2. 单击左侧导航栏中的**服务**。
3. 选择所要暴露的服务所在的集群。

4. 选择所要暴露的服务（本示例中为wordpress_web1）并单击**变更配置**。

图 296: 变更配置



5. 在变更配置页面，配置主机和容器端口的映射。

主机端口为空，表示随机暴露一个主机的端口（暴露HTTP/HTTPS服务时，您可以不需要知道主机暴露的具体端口是什么，可以使用overlay网络或者VPC网络来直接访问容器的端口），容器端口为80。您使用wordoress_web1默认暴露Web服务的 80 端口来提供HTTP服务，使用的协议是TCP协议，网络模式是ingress

图 297: 端口映射



6. 通过**容器标签**进行路由配置，本例中通过域名来暴露服务，须标明要暴露的端口，此处为Web服务80端口。

图 298: 路由配置



域名字段只填写了域名前缀，如果域名前缀为XXX，获得的域名为XXX.$cluster_id.$region_id.alicontainer.com供测试使用。您也可以填写自己的域名，需要添加解析到相应的负载均衡实例IP。关于配置路由的容器端口和HTTP服务的域名的详细信息，参见 *routing*标签。

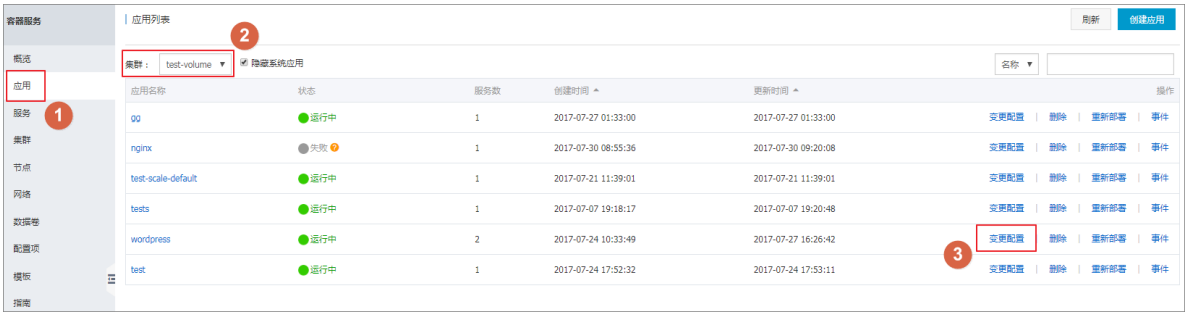
通过应用的模板编辑器进行设置

1. 登录容器云控制台。

   有关如何登录云控制台，参见登录云控制台。

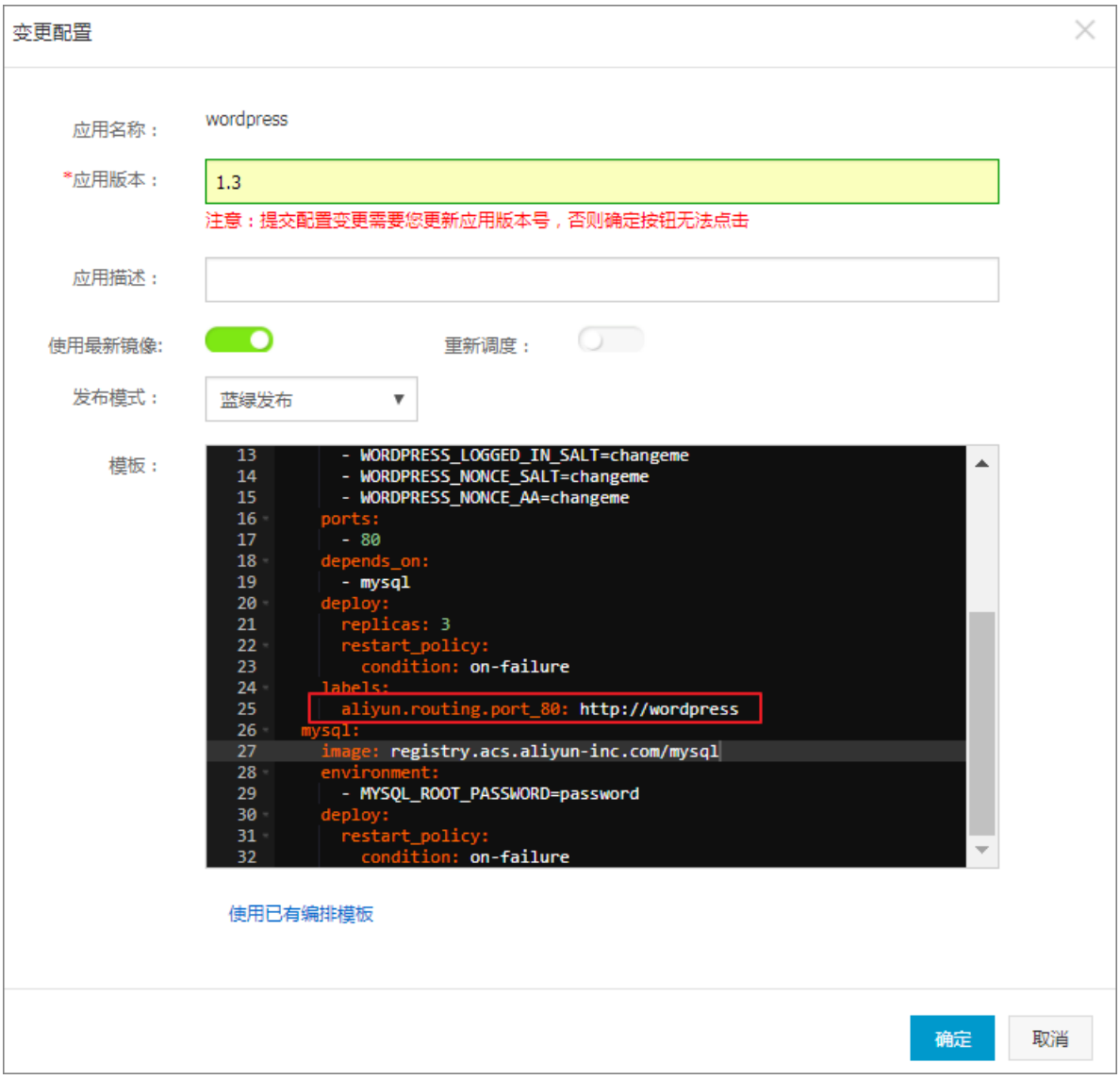
2. 单击左侧导航栏中的**应用**。
3. 选择目标应用所在的集群。
4. 选择目标应用（本示例中为wordpress）并单击**变更配置**。

图 299: 变更配置



5. 在模板编辑器中，添加*routing*标签，定义相应的域名或者域名前缀。注意升级应用的版本，以及确定是否拉取最新的Docker镜像，最后单击**确定** 更新域名。

图 300: 变更配置



通过客户端工具进行设置

- docker help run：查看使用的“*-p*”选项，路由配置在容器服务云控制台进行。
- docker-compose：查看支持的“*ports*”选项，路由配置规则详情见*routing*标签。

12.5.14.2 简单路由-域名配置

操作步骤

1. 登录容器服务云控制台。

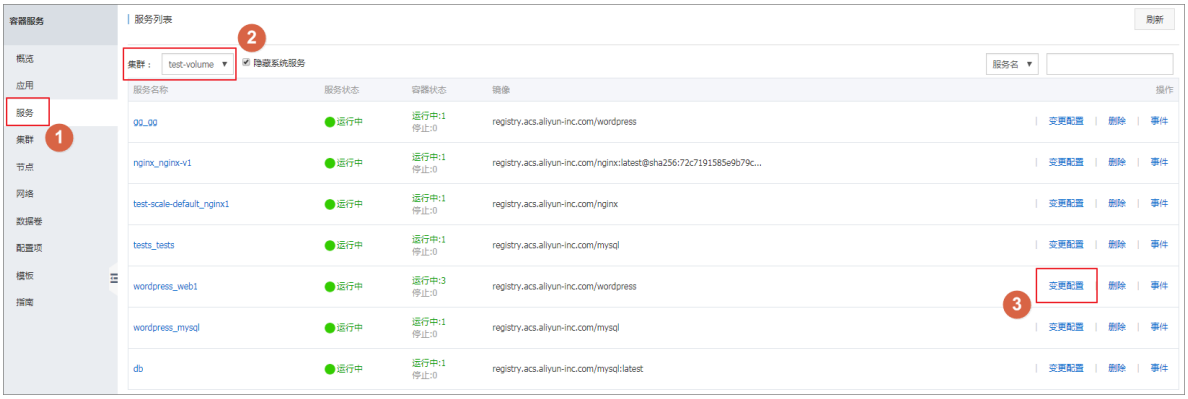
   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**服务**。

3. 选择要添加域名的服务所在的集群。

4. 选择要添加域名的服务（本示例中要添加域名的服务为wordpress_web1，所属的应用为wordpress）并单击**变更配置**。

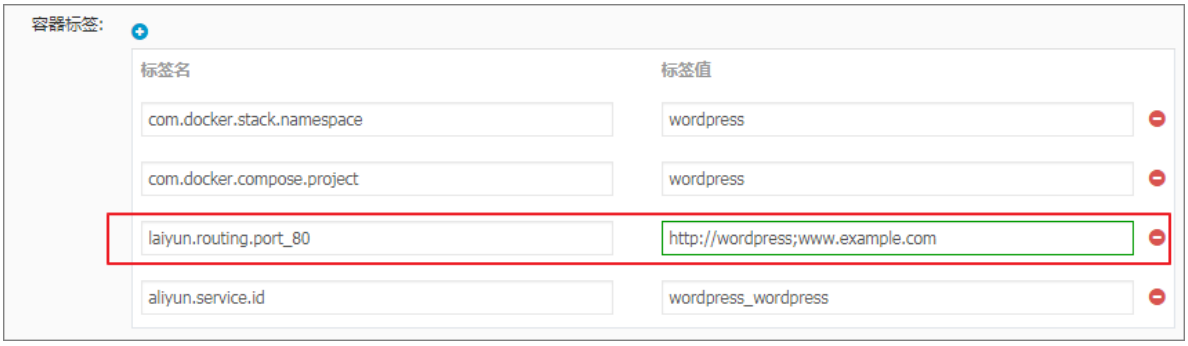
图 301: 变更配置



5. 单击**容器标签**右侧的加号图标，输入要添加的域名（本示例中要添加的域名为www.example.com）并单击**确定** 更新配置。

> **说明:** 同一个服务同一个端口的多个域名只能写在同一个条目内，并且域名和域名之间用分号（;）分隔。

图 302: 简单路由配置



此时，服务处于更新中。更新完毕变成就绪状态后，路由服务acsrouting_routing就已经将该域名配置好了。当有请求以域名www.example.com访问服务web 时，就能正确的解析并转发到相应的服务了。

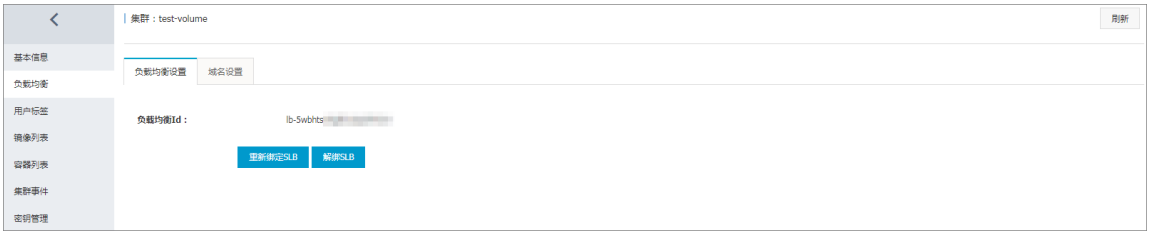
6. 将域名解析到容器服务的集群上。容器服务在创建集群的时候，会给每一个集群分配一个负载均衡实例，该负载均衡实例是属于您自己的。

a) 单击容器服务云控制台左侧导航栏中的**集群**。

b) 选择相应的集群，并单击**管理**。

c) 单击**负载均衡**可以查看负载均衡的ID。

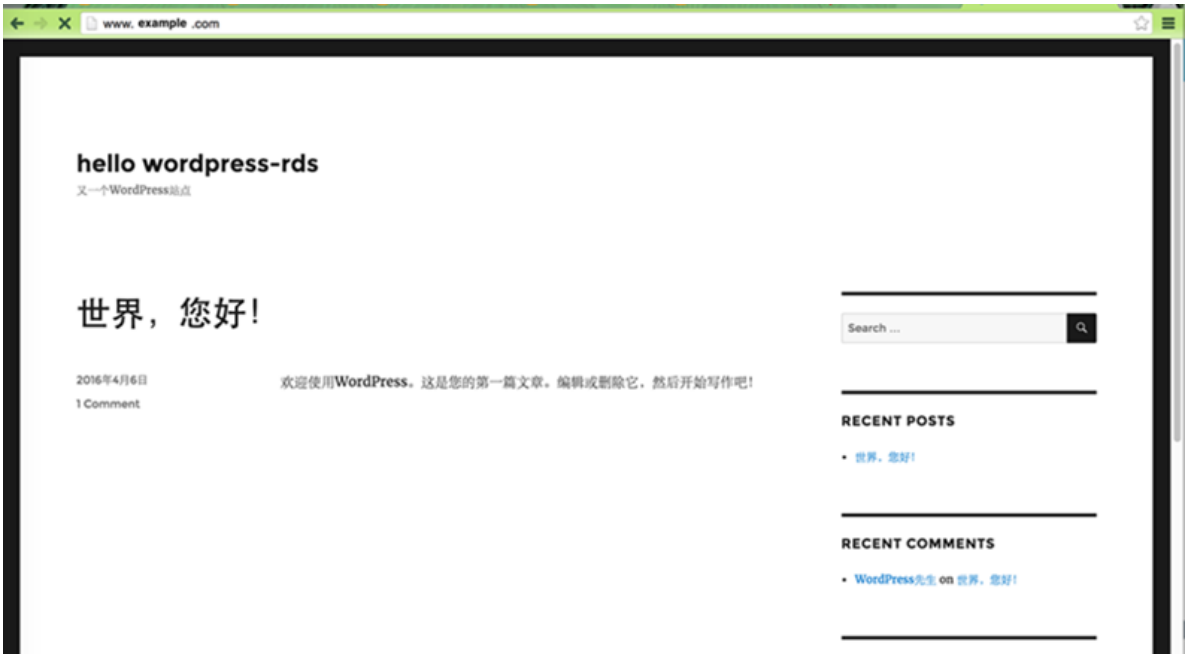
图 303: 负载均衡实例



d) 登录负载均衡云控制台，在实例列表页面找到集群的负载均衡并单击实例的ID。

您可以查看负载均衡实例的**服务地址**。

7. 以万网的域名解析为例，添加A记录，绑定域名www.example.com的地址到负载均衡实例的服务地址。
8. 访问页面www.example.com。



12.5.14.3 简单路由-HTTP协议变为HTTPS协议

前提条件

如果您还没有配置成功HTTP协议的域名访问，请先了解配置HTTP的域名访问。更多详细信息，参见简单路由-域名配置。

操作步骤

1. HTTPS协议是在负载均衡这一层进行支持的。为了支持HTTPS协议，您需要创建负载均衡证书。
   a) 登录负载均衡云控制台。
      有关如何登录云控制台，参见登录云控制台。
   b) 单击**证书管理**页签。
   c) 单击**上传安全证书**。
   d) 在**上传安全证书**对话框，填写证书信息，然后单击**确认**。
2. 证书创建成功后，找到创建集群时分配的负载均衡实例。
   容器服务在创建集群的时候，给每一个集群分配了一个负载均衡实例，该负载均衡实例是属于您自己的。
   a) 单击容器服务云控制台左侧导航栏中的**集群**，选择相应的集群，单击**管理**。
   b) 单击**负载均衡**，您可以查看负载均衡实例的ID。

      图 304: 负载均衡实例

   c) 登录负载均衡云控制台，在实例列表页面选择集群的负载均衡实例并单击实例的ID。
      您可以查看负载均衡实例的服务地址。
3. 单击**服务监听**页签，打开**服务监听**列表页面。
4. 单击**添加**。
5. 在**添加服务监听**对话框，指定监听配置，然后单击**确认**完成。
   a) 前端协议，选择HTTPS。
   b) 端口使用443端口，后端端口使用9080端口（该端口为路由服务**acsrouting_routing**在每一台ECS主机上暴露的端口，所有的HTTP请求会在路由服务**acsrouting_routing**上根据HTTP协议的HOST header转发到相应的提供各种服务的容器内）。
   c) 选择前面步骤创建的证书www.example.com。
   d) 根据需要设置其他选项。
   e) 单击**下一步**。

6. 完成**健康检查配置**标签页中的配置。单击**确认**。

   您可以选择开启或关闭健康检查。如果您选择开启健康检查，您需要在**域名**中填写您自己的域名或者在**检查路径**中填写/haproxy-monitor。否则，健康检查会报异常。

7. 配置成功后，单击**确认**。

8. 访问页面https://www.example.com。



12.5.14.4 集群内服务间路由和负载均衡

swarm mode集群内部自带负载均衡，Docker会在节点间默认创建一个ingress网络，通过ingress LB可以将容器网络中的容器服务暴露到宿主机网络中，从而被外部请求快速发现，并支持通过服务名或别名的访问方式。

swarm mode集群支持healthcheck进行健康检查，当充当访问入口的某个节点的容器服务出现问题时， ingress LB会自动路由到其他节点的active容器拷贝（repic）上，做到容器服务的自动负载均衡，最终实现集群的高可用。请参考*routing mesh*了解更多。

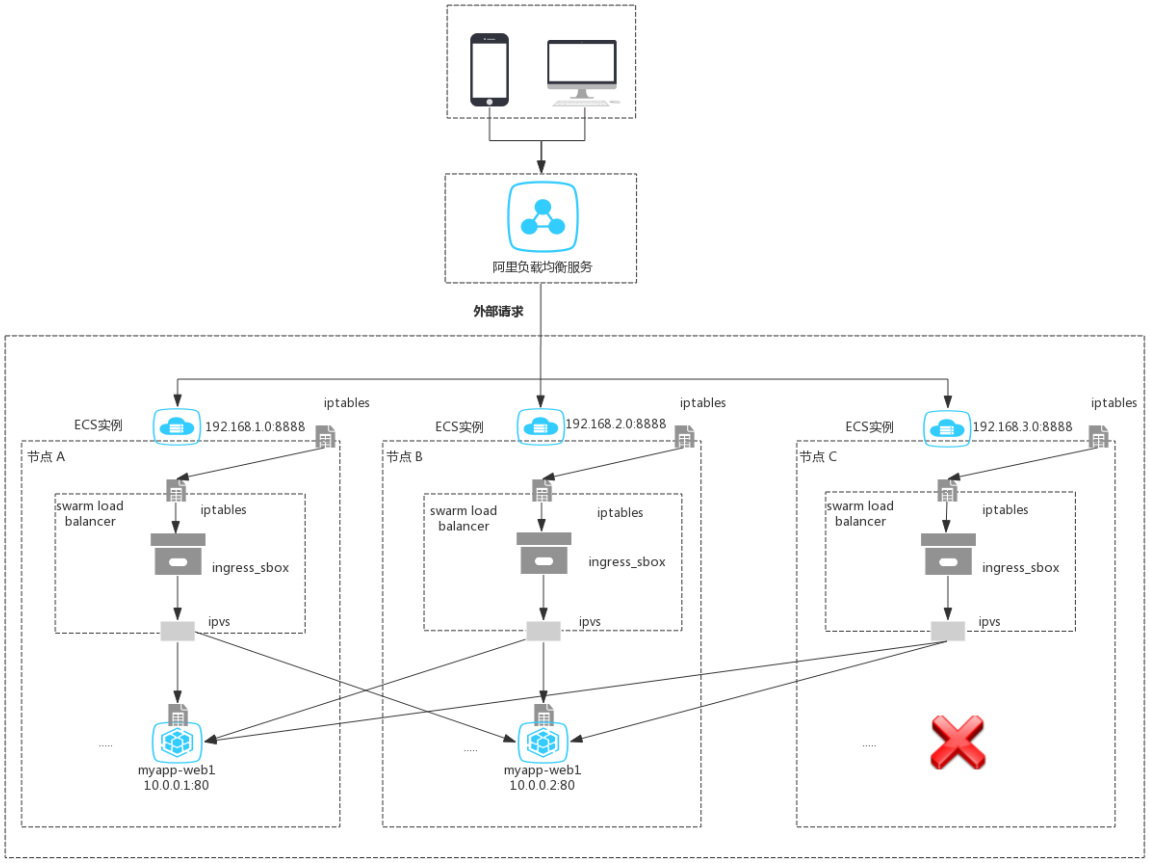
实现原理

swarm mode集群会默认生成一个ingerss网络，在用户请求 > 节点IP: 端口 > ingress > 容器的访问链路中，负责与节点和内部容器间的通讯。其中ingress是由IPVS实现的，具有负载均衡和名字解析功能。

Ingress LB分别在节点iptables，ingress_sbox的iptables和IPVS，service容器的iptables处设置相应规则，从而可以在宿主机网络上通过宿主机IP和published端口来访问容器网络的服务。

您可以使用阿里云负载均衡服务为swarm mode集群配置外部负载均衡服务，将外部请求路由到目标swarm mode集群，实现外网下的服务发现和负载均衡。

图 305: 外网下负载均衡原理



节点published端口 > 容器targeted端口的具体数据流程如下：

1. 宿主机网络通过Worker节点IP和service published port来访问服务。该例中定义publish *8888:80*，可以通过 *worker-ip*:8888访问服务。
2. Worker节点iptables中NAT表定义规则，对于匹配published的宿主机端口（8888）的数据，将其dstIP转换成ingress_sbox中的IP。
3. 节点上的Ingress_sbox会设置manage表，将dst port为8888的数据做标记（fwmark）。通过ipvs，将标记为fwmark的数据包LB到各个实际IP中，默认使用round-robin算法，forware为VS/NAT方式。
4. Service 的各个容器会将dst port为8888的数据的dst port转换成80，从而访问到真实的服务。

优势

- 摆脱传统DNS轮询的时效性问题，同时可以支持UDP等协议，性能比DNS优秀。
- 支持集群内任意节点作为访问入口，服务在任何一台主机的published端口都可到达。

编排示例

有两种方式设置端口：

- 可以手动指定端口（使用*ports*标签）。
- 或者不指定，由Docker swarm在30000-32767中指定一个端口。



说明: 请确保预留7946 TCP/UDP用于发现容器网络和4789 UDP来使ingress网络正常工作。

编排示例：

在该示例中， nginx容器将80端口暴露给集群内所有节点的8888端口，外部请求可以由所有节点上的8888端口自动路由到集群内运行的nginx容器。

```
version: "3"
services:
  ngx:
    image: registry.acs.aliyun-inc.com/nginx
    ports:
      - 8888:80
```

12.5.14.5 负载均衡路由

暴露HTTP协议或者HTTPS协议的服务

推荐使用简单路由服务（即 routing）的方式来暴露HTTP服务或者HTTPS协议的服务，如果您希望搭建自己的路由链路，可以开通新的内网或者公网负载均衡实例路由到VM的端口（通过label *lb*来实现），并设置主机和容器的映射关系来进行请求的路由。

适用场景：

7层协议负载均衡，自定义各服务的路由，在将传统架构迁移到容器架构过程中非容器集群的服务访问容器集群中容器的服务。

暴露TCP协议或者UDP协议的服务

目前如果要暴露TCP协议的服务，需要您自行设置负载均衡实例或者公网IP，并配置好主机端口与容器端口的映射（通过label *lb*来实现）。

适用场景：

4层协议的负载均衡，自定义各服务的路由，在将传统架构迁移到容器架构过程中非容器集群的服务访问容器集群中容器的服务。

示例：

通过自定义负载均衡的方式来将容器集群内的Redis服务暴露给容器集群外的Python应用。

操作步骤

1. 首先在负载均衡云控制台（单击**创建实例**）创建一个用于路由的负载均衡实例。

   有关如何登录云控制台，参见登录云控制台。

   本示例中选择的是外网实例，您可以根据自己的需要选择外网或者内网。

2. 返回负载均衡云控制台，将创建的负载均衡实例命名为slb_redis_app。容器服务会通过该名称来引用这个负载均衡实例。

   选择所需的实例，编辑实例的名称并单击**确定**。

3. 创建监听端口。

   a) 在负载均衡实例列表中，单击所需实例的ID链接，打开负载均衡实例详情页面。

   b) 单击**服务监听**页签，打开**服务监听**列表页面。

   c) 单击**添加**。

   d) 在**添加服务监听**对话框，创建协议为TCP，端口映射为6379:6379，然后单击**确认**完成。

4. 登录容器服务云控制台，选择一个已有的集群，创建一个名称为 redis-demo 的应用，单击 **使用镜像创建**。

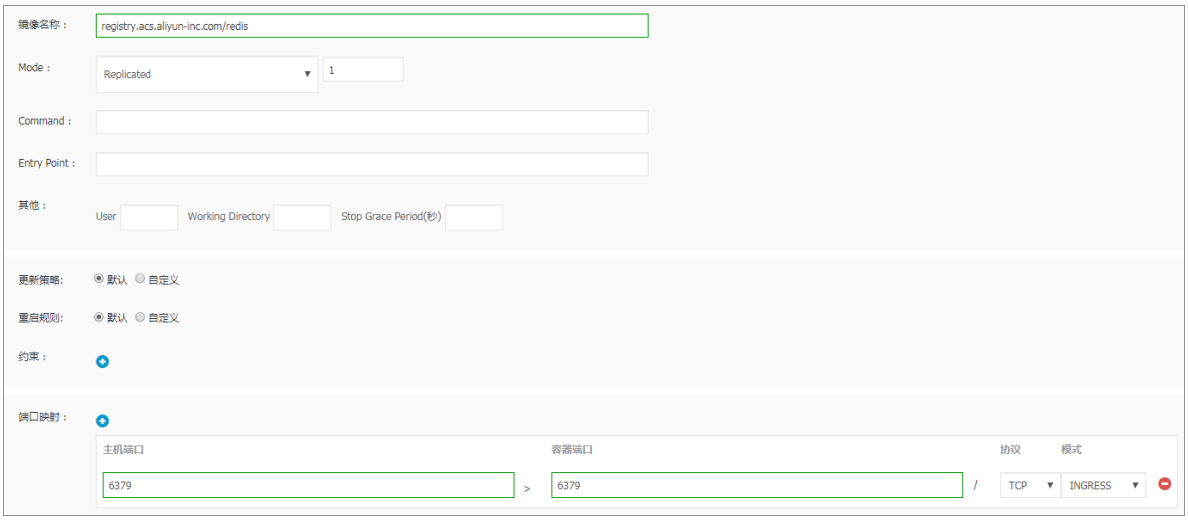
   有关如何创建应用，参见使用镜像创建应用。

5. 填写Redis镜像地址并设置**端口映射**。

   说明: 此处Redis镜像只是开通了容器的6379端口，为了使创建的负载均衡路由到这个容器端口，您必须知道Redis镜像的主机：端口映射。

   在**端口映射** 中，指定主机端口为6379，主机端口6379即为负载均衡实例绑定的后端主机端口，选择使用的协议为TCP。

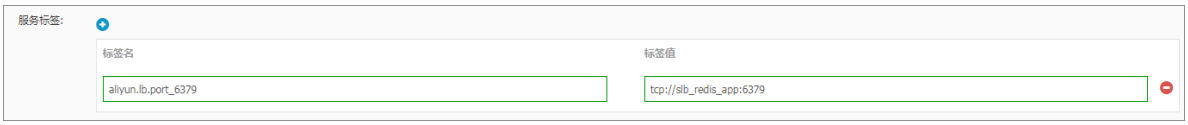
   图 306: 端口映射



6. 为了配置自定义负载均衡，需要让Redis服务知道使用的负载均衡实例的信息。您可以向服务中注入一个标签。

   本示例中，标签为aliyun.lb.port_6379: tcp://slb_redis_app:6379。

   图 307: 设置服务标签



   aliyun.lb.port_$container_port:$scheme://$[slb_name|slb_id]:$front_port

   标签格式如下，带 $ 的变量为占位符。

   - $container_port表示容器要暴露的端口。
   - $scheme表示负载均衡实例监听端口支持的协议，可能的值为tcp、http、https、udp。
   - $[slb_name|slb_id]表示可以填写负载均衡实例的名称或者ID。
   - $front_port表示负载均衡实例要暴露的前端端口。

   更多详细信息，参见*lb*。

   本示例中，路由到的容器端口为6379，引用前面创建的负载均衡实例名称slb_redis_app，与上面主机：容器端口映射设置的TCP协议相呼应，本示例设置监听端口的协议为TCP协议，同时设置负载均衡的前端端口为6379。

   说明: 本示例中，同时将负载均衡实例的前端端口、后端端口（即主机的端口）和容器端口均设置为6379，您可以根据自己的需要设置不同的前端端口和主机端口。

7. 单击**创建并部署**，Redis应用即开始创建了。

   Redis应用在创建的过程中会自动将名称为slb_redis_app的负载均衡实例绑定到部署了redis镜像的后端主机。

8. 当应用处于就绪状态后，登录负载均衡云控制台，查看名为slb_redis_app的负载均衡实例的状态。

   在实例列表中，选择所需的实例，单击实例的ID链接 ， 单击**后端服务器**页签，即可查看绑定到负载均衡的后端服务器。

   由健康状态可见，负载均衡已经正确地绑定到了Redis的后端。

9. 您可以在负载均衡云控制台的实例详情页面查看负载均衡实例的IP地址，并使用命令行工具telnet $*server_load_balancer_ip_address* 6379来检查端口的可访问性。

10. 为了测试以上配置，在本地运行一个简单的Python应用来通过slb_redis_app负载均衡实例访问容器集群内的Redis。

    说明: Redis主机地址是负载均衡的IP地址。

    app.py

    ```
    from flask import Flask
    from redis import Redis
    app = Flask(__name__)
    redis = Redis(host='$server_load_balancer_ip_address', port=6379)
    @app.route('/')
    def hello():
    redis.incr('hits')
    return 'Hello World! I have been seen %s times.' % redis.get('hits')
    if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
    ```

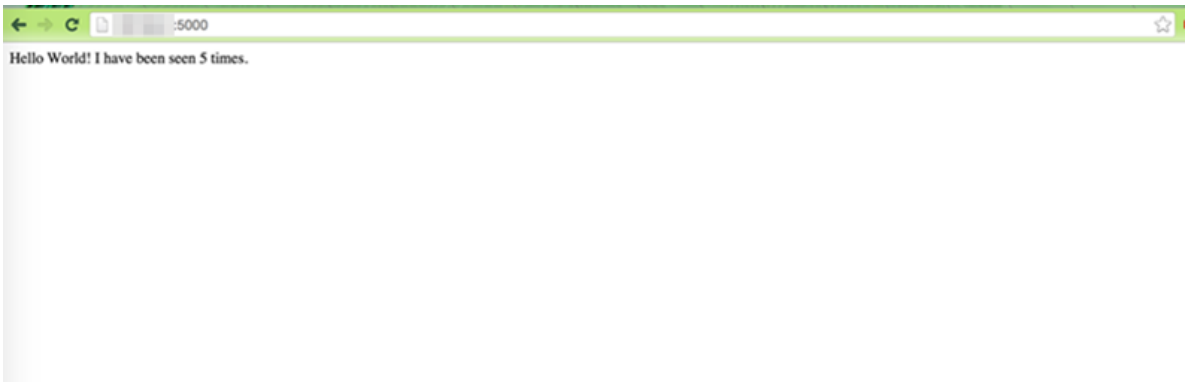
    requirements.txt

    ```
    flask
    redis
    ```

    shell

    ```
    $ pip install -r requirements.txt
    $ python app.py
    Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
    Restarting with stat
    Debugger is active!
    Debugger pin code: 243-626-653
    ```

    访问结果如下图所示。



12.5.14.6 服务发现

集群内服务发现原理

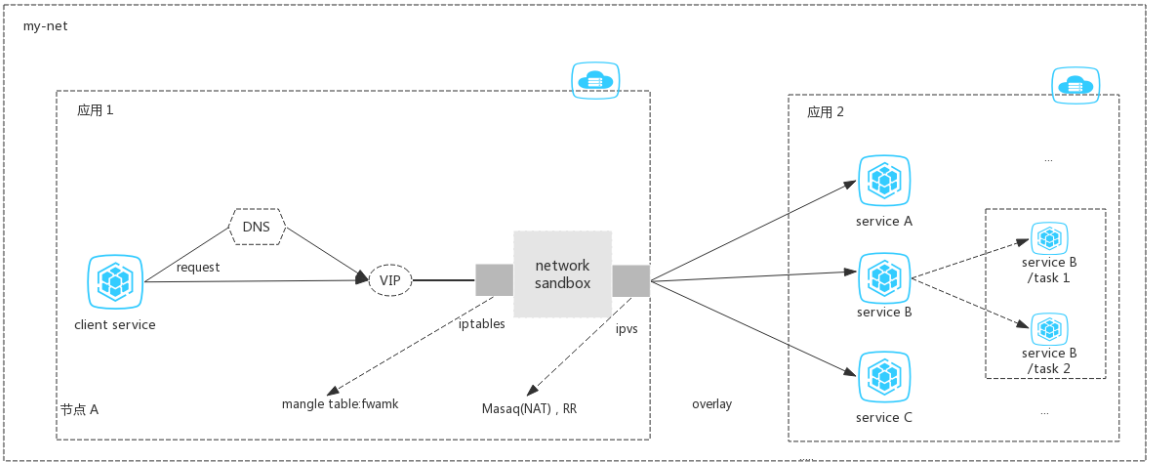
集群内部的服务发现可以通过routing mesh和内置的DNS服务两种方式实现:

- 通过rouging mesh：routing mesh将服务发现和负载均衡结合到一起，参见集群内服务间路由和负载均衡。
- 通过DNS：通过内置的DNS+VIP+ipvs机制，集群会为每一个启动的service分配一个vip，并在DNS中将service name解析为该VIP，发送给该VIP的请求将被自动分发到service下面的诸多active task上（对应容器）。

  说明: 对本场景来说，您可以考虑不暴露节点端口，不加入ingress网络，通过内置DNS服务实现服务发现。

swarm mode提供了基于DNS的服务发现机制。注意这里的DNS server也是分布式的，每一个主机上都有一个DNS server。manager会为该service分配一个VIP（Virtual IP），并在内部DNS上建立一条VIP与service name的映射记录。在每个容器的network sandbox中会配置相应的iptables和ipvs，同一个网络中每增加一个服务，会相应添加对应的iptables和ipvs规则，从而使得服务在整个网络内可访问。

图 308: 服务发现原理



应用1和应用2加入同一个网络my-net，client service向同处于网络my-net中的应用2中的服务发起访问，可以通过service name或者VIP来访问service。通过服务名访问时，容器会访问Docker engine中内置的DNS服务，从而获取VIP。最后由Iptables配合IPVS将VIP请求负载到service下面的一个具体容器上。

服务发现方式

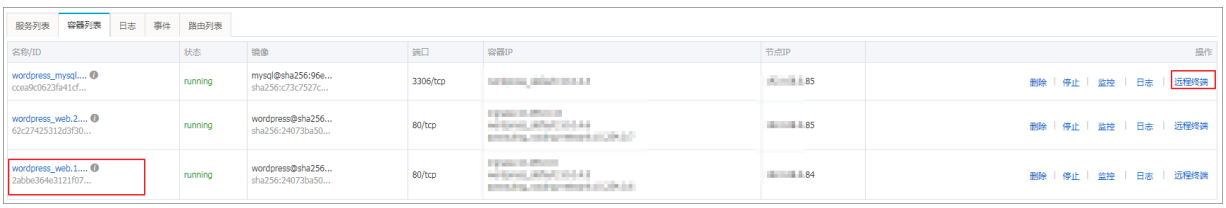
swarm mode集群下，容器服务为集群内的服务发现提供多种方式，推荐使用服务名和容器名。Docker engine内置的DNS服务会将服务名和容器名解析成正确的IP，最终实现服务发现。

说明: 应用需要加入同一个网络，服务间才能互相发现。

通过容器名

容器服务不仅可以通过容器的IP（跨主机互联的容器网络）进行访问，还可以通过网络中其他容器的容器名进行访问。本示例中，您可以在**wordpress_mysql.1.m0m29h9hzkp5mabd32zk0nk8u的**容器中通过**ordpress_web.1.69c1toswifjpr76smxzp2w2jw**的容器名进行访问。

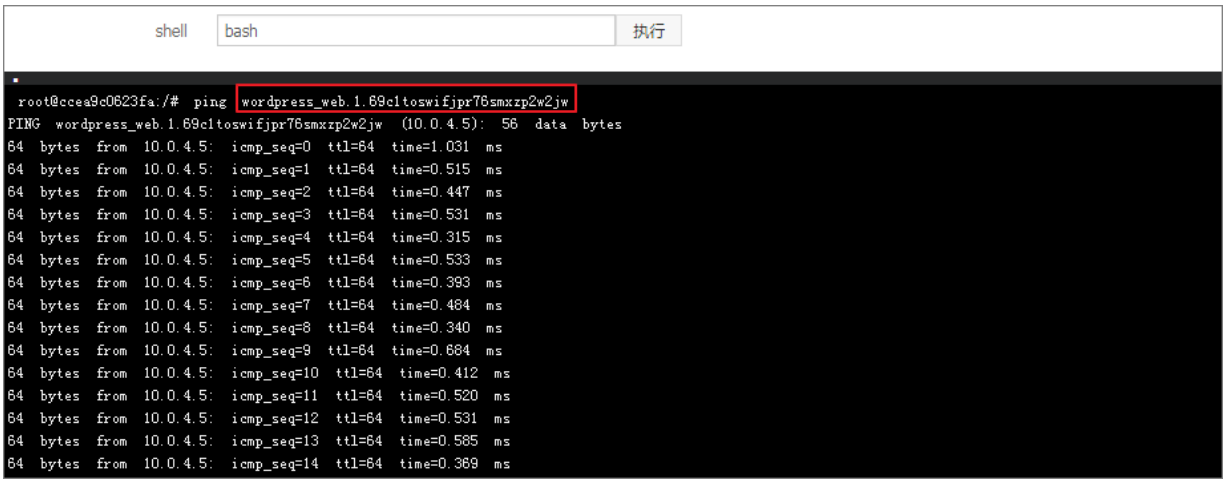
图 309: 容器列表



如果在编排文件中不指定*container_name*的话，默认的命名规范有两种：

- **compose V1/V2**：默认的容器名为*{project-name}_{service-name}_{container-index}*，所以在连接管理终端后，您可以通过另外一个服务的容器名进行访问。
- **compose V3**：默认的容器名为*{project-name}_{service-name}_{container-index}.{*随机字符*}*，可在容器服务云控制台进行查看。

图 310: 访问



通过服务名

swarm mode集群支持通过服务名的发现方式，服务名的默认格式是*{project-name}_{service-name}*。内置的DNS服务会将服务名解析为正确的IP（VIP），从而进行发现。

12.5.15 发布策略

容器服务支持简单路由蓝绿发布和负载均衡蓝绿发布。

12.5.15.1 概述

容器服务提供两种发布策略：蓝绿发布和标准发布。两种发布策略的差异如下表所示。

表 37: 发布策略比较

发布策略	区别
标准发布	在部署新版本应用时删除旧版本应用。发布过程中，您的服务会出现短暂中断。
蓝绿发布	应用更新时生成蓝绿两个版本，两个版本互相热备，通过切换路由权重的方式实现不同版本应用的上下线。 该发布策略具有以下特点：

发布策略	区别
	• 服务不会中断 • 应用零宕机 • 发布过程用户无感知重启 • 升级失败自动回滚 • 同一资源栈可以多次升级迭代

标准发布使用场景

标准发布是传统的应用发布策略，在现有环境中，部署新版本应用。除了有特殊需求以外，一般采用此类发布策略，使用场景包括：

- 新版本应用已经过充分测试，无需线上试运行。
- 应用包含数据库服务，且发生过如数据表结构变更等不可逆转的升级情况，标准发布可以免除类似数据迁移、回滚等业务改造的工作。

蓝绿发布使用场景

蓝绿发布适用的应用形态包括前端类服务和后端类服务，常用于应用的增量升级中，面向客户的具体业务场景包括：

保证业务连续性

对业务变化迅速的互联网企业，应用版本迭代频繁是一个现实的挑战，保证线上业务的连续性是必要的。蓝绿发布策略主要的价值是零宕机应用更新，保证服务不中断，能满足云环境下持续发布的应用更新需求。

新版本线上评测

蓝绿发布策略可以让您在线上环境进行版本测试，通过保持一段时间内新旧版本并存，来充分测试新版本服务的功能、性能和安全性。待新版本稳定后，再将旧版本应用下线。

12.5.15.2 使用简单路由蓝绿发布

背景信息

蓝绿发布是一种**零宕机**的应用更新策略。进行蓝绿发布时，应用的旧版本服务与新版本服务会同时并存，同一个应用不同版本的服务之间共享路由，通过调节路由权重的方式，可以实现不同版本服务之间的流量切换。验证无误后，可以通过发布确认的方式将应用的旧版本的服务删除；如果验证不通过，则进行发布回滚，应用的新版本会进行删除。

前置条件

支持蓝绿发布需要将路由服务升级到最新的版本。详细信息参见 升级系统服务。

场景介绍

假设您要进行蓝绿发布的应用是一个Nginx的静态页面，初始的应用模板如下。

Compose V1/V2编排示例：

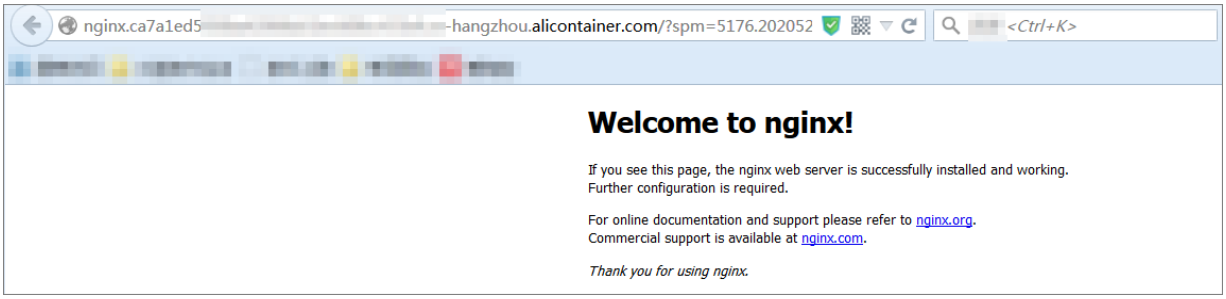
```
nginx-v1:
  image: registry.acs.aliyun-inc.com/nginx
  labels:
    aliyun.routing.port_80: nginx
  restart: always
```

Compose V3编排示例：

```
version: '3.2'
services:
  nginx-v1:
    image: registry.acs.aliyun-inc.com/nginx
    labels:
      aliyun.routing.port_80: nginx
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        failure_action: continue
      restart_policy:
        condition: any
```

部署后页面的效果如下。

图 311: 访问Nginx



操作步骤

1. 登录容器服务云控制台。

   有关如何登录云控制台，参见登录云控制台。

2. 单击左侧导航栏中的**应用**。

3. 选择目标应用所在的集群。
4. 选择目标应用并单击**变更配置**。

图 312: 变更配置

5. 选择变更的发布模式与新版本服务的配置。

> **说明:**
>
> - 在蓝绿发布中，新版本与旧版本不能共用同一个名字。
> - 在蓝绿发布的场景中，为了保证应用的零宕机切换，新版本的服务的路由权重默认为0，需要通过路由管理页面进行调整，方可进行流量切换。

图 313: 变更配置

Compose V1/V2示例：

```
nginx-v2:
  image: nginx
  labels:
    aliyun.routing.port_80: nginx
  restart: always
```

Compose V3示例：

```
nginx-v2:
  image: nginx
  labels:
    aliyun.routing.port_80: nginx
  restart: always
```

6. 单击**确定**，发布变更。

在发布的过程中，会经历两个状态：

- 蓝绿发布中：表示新版本的服务尚未启动完成。
- 蓝绿发布待确认：表示新版本的服务已经启动完成，此时需要进行发布确认或者发布回滚方可进行下一次发布。

进入应用的详情页面，可以看到新版本的应用和旧版本的应用并存。

图 314: 蓝绿发布



7. 单击 **路由列表** 并单击 **设置服务权重**。

   如下图所示，旧版本服务的权重为 100，新版本服务的权重为 0。

   图 315: 路由权重



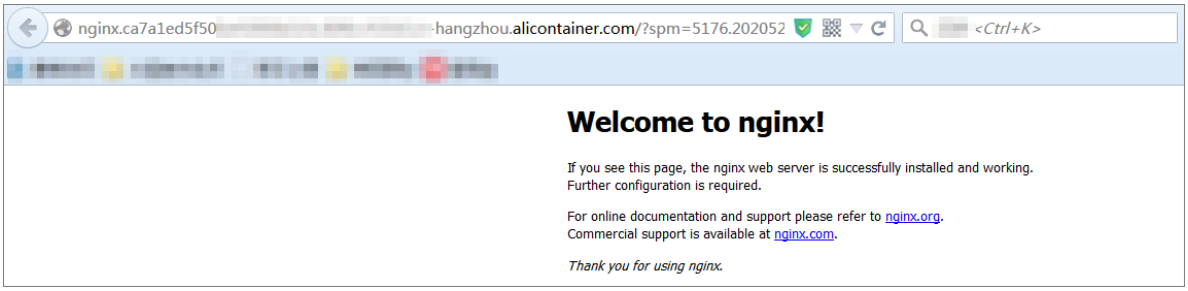
   将旧版本服务的权重调整为 0，新版本服务的权重调整为 100。

   图 316: 设置路由权重



8. 由于默认路由服务是进行会话保持的，您可以打开一个新的浏览器窗口，访问新的版本。

图 317: 访问Nginx



9. 当整个发布流程验证完毕后，在应用列表页面，单击**确认发布完成**并在弹出的确认对话框中单击**确认**进行发布确认，方可进行下一次发布。

图 318: 确认发布完成

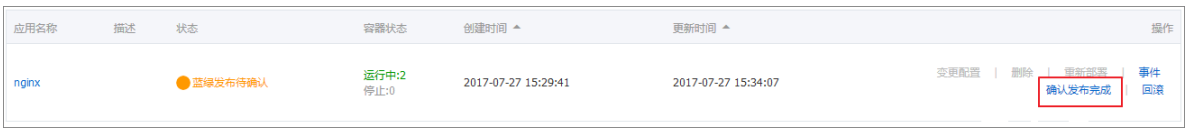


图 319: 发布确认



您可以看到应用的服务列表已经更新了，旧的服务已经完全下线删除了。

图 320: 发布完成



13 对象存储OSS

13.1 什么是 OSS

对象存储服务（Object Storage Service，简称 OSS）提供海量、安全、低成本、高可靠的云存储服务。它可以理解为一个即开即用，无限大空间的存储集群。相比传统自建服务器存储，OSS 在可靠性、安全性、成本和数据处理能力方面都有着突出的优势。使用 OSS，您可以通过网络随时存储和调用包括文本、图片、音频和视频等在内的各种非结构化数据文件。

OSS 将数据文件以对象/文件（object）的形式上传到存储空间（bucket）中。 您可以进行以下操作：

- 创建一个或者多个存储空间
- 每个存储空间中添加一个或多个文件
- 通过获取已上传文件的地址进行文件的分享和下载
- 通过修改存储空间或文件的属性或元信息来设置相应的访问权限
- 通过云控制台执行基本和高级 OSS 任务
- 通过开发工具包 SDK 或直接在应用程序中进行 RESTful API 调用执行基本和高级 OSS 任务。

13.2 基本概念

存储空间

阿里云 OSS 中的所有文件都存储在存储空间（Bucket）中。存储空间是您用来管理所存储文件的单元。所有对象都必须隶属于某个存储空间。您可以设置存储空间属性来控制地域、文件的访问控制、文件的生命周期等，这些属性都是作用在该存储空间下所有的文件上的，因此您可以灵活创建不同的存储空间来完成不同的管理功能。

同一个存储空间内部的空间是扁平的，即没有文件系统的目录等概念，所有的文件都是直接隶属于其对应的存储空间。但您可以使用文件夹对相关文件进行分组归类管理。

对象（文件）

在 OSS 服务中，用户操作的基本数据单元是对象（Object），单个对象的大小限制为 48.8 TB，一个存储空间中可以有无限量的对象。

您必须先拥有存储空间写入权限，才能将对象文件上传到 OSS 中。上传的对象在控制台上的展现形式为文件或文件夹。

访问密钥

访问密钥（AccessKey，简称AK），指的是访问身份验证中用到的 AccessKeyId 和 AccessKeySecret。 OSS 通过使用 AccessKeyId 和 AccessKeySecret 对称加密的方法来验证某个请求的发送者身份。AccessKeyId 用于标示用户，AccessKeySecret 是用户用于加密签名字符串和 OSS 用来验证签名字符串的密钥，其中 AccessKeySecret 必须保密。

对于 OSS 来说，AccessKey 的来源有：

- 存储空间的拥有者申请的 AccessKey
- 被存储空间的拥有者通过资源访问权限控制（Resource Access Management，简称RAM）授权的第三方请求者的 AccessKey
- 被存储空间的拥有者通过临时访问权限控制（Security Token Service，简称STS）授权的第三方请求者的 AccessKey

13.3 快速开始

本操作指引您快速完成基本任务，包括创建存储空间、上传文件和分享文件。

13.3.1 上传文件

背景信息

创建了存储空间之后，您可以上传任意类型文件到存储空间中。通过 OSS 控制台您可以上传小于 500 MB 的文件。如您要上传的文件大于 500 MB，您可以通过 SDK 或 API 进行上传。

操作步骤

1. 登录OSS 云控制台。
2. 单击您要向其中上传文件的存储空间名称，打开该存储空间管理页面。
3. 单击 **Object 管理**，进入该存储空间内所有文件管理页面。
4. 单击**上传文件**，打开**文件选择**对话框。
5. 选择要上传的文件，单击**打开**。
6. 文件上传成功后，请刷新页面查看已上传的文件。

   您可以在**任务管理**中查看上传进度和结果。

13.3.2 获取文件访问地址

背景信息

您已经将文件上传到存储空间中，现在您可以通过获取已上传文件的地址进行文件的分享和下载。

操作步骤

1. 登录OSS 云控制台。
2. 单击您要向其中上传文件的存储空间名称，打开该存储空间管理页面。
3. 单击**Object 管理**，进入该存储空间内所有文件管理页面。
4. 选择目标文件，单击**管理** > **获取地址** ，弹出**获取文件地址**对话框。

图 321: 获取文件地址



说明: 如您的存储空间的读写权限为私有，则在您获取文件访问 URL 时需设置链接有效时间。您可单击**获取地址** 获取文件链接。URL 签名的链接有效时间是基于 NTP 计算的。您可以将此链接给与任何访问者，访问者可以在有效时间内，通过此链接访问该文件。存储空间为私有权限时获得的地址是通过 URL 签名方式生成的。

5. 复制文件链接给访问者进行浏览或者下载。

13.4 管理存储空间

本章节包含的功能与实际项目可能有所不同，请联系客户经理确认。相关操作步骤仅为示例，请您参照云控制台界面。

13.4.1 查看存储空间

操作步骤

1. 登录OSS 云控制台。

2. 单击目标存储空间的名称，进入存储空间管理页面。
3. 在**Bucket信息**页面下您可以查看Bucket的服务地址和创建时间等详细信息。

13.4.2 设置访问权限

背景信息

OSS 提供 ACL（Access Control List）权限控制方法，您可以在创建存储空间的时候设置相应的存储空间权限（ACL），也可以在创建之后修改 ACL。如果不设置 ACL，默认值为私有。

OSS ACL 提供存储空间级别的权限访问控制，存储空间目前有三种访问权限：

- Private（私有权限）：只有该存储空间的创建者或者授权对象可以对该存储空间内的文件进行读写操作，其他人在未经授权的情况下无法访问该存储空间内的文件。
- Public-read（公共读，私有写）：只有该存储空间的创建者可以对该存储空间内的文件进行写操作，任何人（包括匿名访问）可以对该存储空间中的文件进行读操作。
- Public-read-write（公共读写）：任何人（包括匿名访问）都可以对该存储空间中的文件进行读写操作，所有这些操作产生的费用由该存储空间的创建者承担，请慎用该权限。

操作步骤

1. 登录*OSS* 云控制台。
2. 单击目标存储空间的名称，进入存储空间管理页面。
3. 单击 **Bucket 属性** > **读写权限设置**。
4. 在 **读写权限** 框中，下拉选择该存储空间的读写权限。
5. 单击 **设置** 保存对存储空间读写权限的修改。

13.4.3 设置静态网站托管

背景信息

您可以通过使用控制台将自己的存储空间配置成静态网站托管模式，并通过存储空间域名访问该静态网站。

如果默认页面为空，则表示不启用静态网页托管。

直接访问静态网站根域名或者该域名下任何一个以“/”结尾的 URL 都会返回默认首页。

操作步骤

1. 登录*OSS* 云控制台。

2. 单击目标存储空间的名称，进入存储空间管理页面。
3. 单击 **Bucket 属性** > **Website 设置**。
4. **默认首页** 框中用来设置索引页面（相当于网站的 index.html），仅支持使用当前存储空间内已存储的 html 格式的文件。
5. **默认 404 页**框中用来设置访问错误路径时返回的默认 404 页，仅支持使用当前存储空间内已存储 html 格式的文件，如果为空则不启用默认 404 页设置。
6. 单击**设置**保存静态网页信息。

13.4.4 设置日志

背景信息

您可以利用云控制台为存储空间启用和禁用日志记录。您可以在已启用日志记录的同一存储空间中存储日志，也可以另外创建新存储空间来存储日志。

操作步骤

1. 登录OSS 云控制台。
2. 单击目标存储空间的名称，进入存储空间管理页面。
3. 单击 **Bucket 属性** > **Logging 设置** 。
4. 在**日志存储位置**框中，下拉选择选择存储日志记录的存储空间名称，只能选择同一用户下同一数据中心的存储空间。选择**不存储**则直接禁用日志记录功能。
5. 在**日志前缀**框中输入日志名称前缀，即下面的命名规则中的 *<TargetPrefix>*，日志将被记录在根目录下。您也可以在*<TargetPrefix>*前面加文件夹路径，例如：*log/<TargetPrefix>*，日志将被记录在log/目录下

   日志记录命名规则

   存储访问日志记录的对象命名规则示例如下：

   <TargetPrefix><SourceBucket>YYYY-MM-DD-HH-MM-SS-<UniqueString>

   - *<TargetPrefix>*：用户指定的日志前缀
   - *<SourceBucket>*：源存储空间名称
   - *YYYY-MM-DD-HH-MM-SS*：该日志创建时的北京时间，显示内容为“年-月-日-小时-分-秒”，字母位数与最终呈现数字位数一致
   - *<UniqueString>*：OSS 系统生成的字符串

例如一个实际用于存储 OSS 访问日志的对象名称如下：

MyLog-OSS-example2015-09-10-04-00-00-0000

其中，MyLog- 为用户指定的日志前缀，oss-example 是源存储空间的名称，2015-09-10-04-00-00 是该日志被创建时的北京时间，0000 是 OSS 系统生成的字符串。

6. 单击**设置**保存日志设置信息。

13.4.5 设置 IP 防盗链

背景信息

为了减少您存储于 OSS 的数据被其他人盗链，OSS 支持设置基于 HTTP header 中表头字段 referer 的防盗链方法。您可以通过控制台为您的存储空间设置 referer 字段的白名单和是否允许 referer 字段为空的请求访问。

操作步骤

1. 登录*OSS* 云控制台。
2. 单击目标存储空间的名称，进入存储空间管理页面。
3. 单击 **Bucket 属性** > **防盗链设置**。
4. 在**Referer**框中，添加白名单网址。
5. 设置是否允许其为空。
6. 单击**提交**，保存对防盗链的设置。

例如，对于一个名为 oss-example 的存储空间，设置其 referer 白名单为 http://www.aliyun.com。则只有referer 为 http://www.aliyun.com 的请求才能访问 oss-example 这个存储空间中的对象。

13.4.6 管理生命周期

背景信息

您可以通过控制台定义和管理存储空间的生命周期配置规则。您可以为存储空间中所有对象或对象的某个子集（通过指定关键词名称前缀）定义规则。由于配置的规则会自动应用于所有符合该规则的对象，所以您可以利用生命周期管理进行文件的批量管理和自动碎片删除等操作。

- 对于规则匹配的对象，系统会保证在生效日期两日内将数据清除。
- 应用生命周期规则而批量删除的数据将无法恢复，请谨慎配置该规则。

操作步骤

1. 登录OSS 云控制台。
2. 单击目标存储空间的名称，进入存储空间管理页面。
3. 单击 **Bucket 属性** > **LifeCycle设置**。
4. 单击 **添加规则**，打开**增加LifeCycle规则设置** 对话框。

图 322: 增加 Lifecycle规则



5. 配置对应的Lifecycle规则。
   - **状态**：指定本条规则的状态，分别表示启用该规则和停用该规则。
   - **策略**：选择匹配对象策略，可以选择直接匹配到整个存储空间，也可以按照对象名称前缀匹配。
   - **前缀**：假设您在存储空间中存储了图片对象，这些对象使用 img/ 作为名称前缀，要对这些文件进行生命周期管理，则在此键入 img/。
   - **过期**：设置对象文件保留天数，以该文件最后修改时间的天数为起始计算，超过设定天数时即执行规则进行删除。如设置时间为 30 天，最后修改日期为 2016-1-1 的对象会在 1 月 31 号被 OSS 扫描删除。
6. 单击**确定**保存设定的规则。规则保存成功后，您可以在策略列表中查看已设置的生命周期规则，并进行相应的**编辑**或者**删除**操作。

13.4.7 设置跨域资源共享

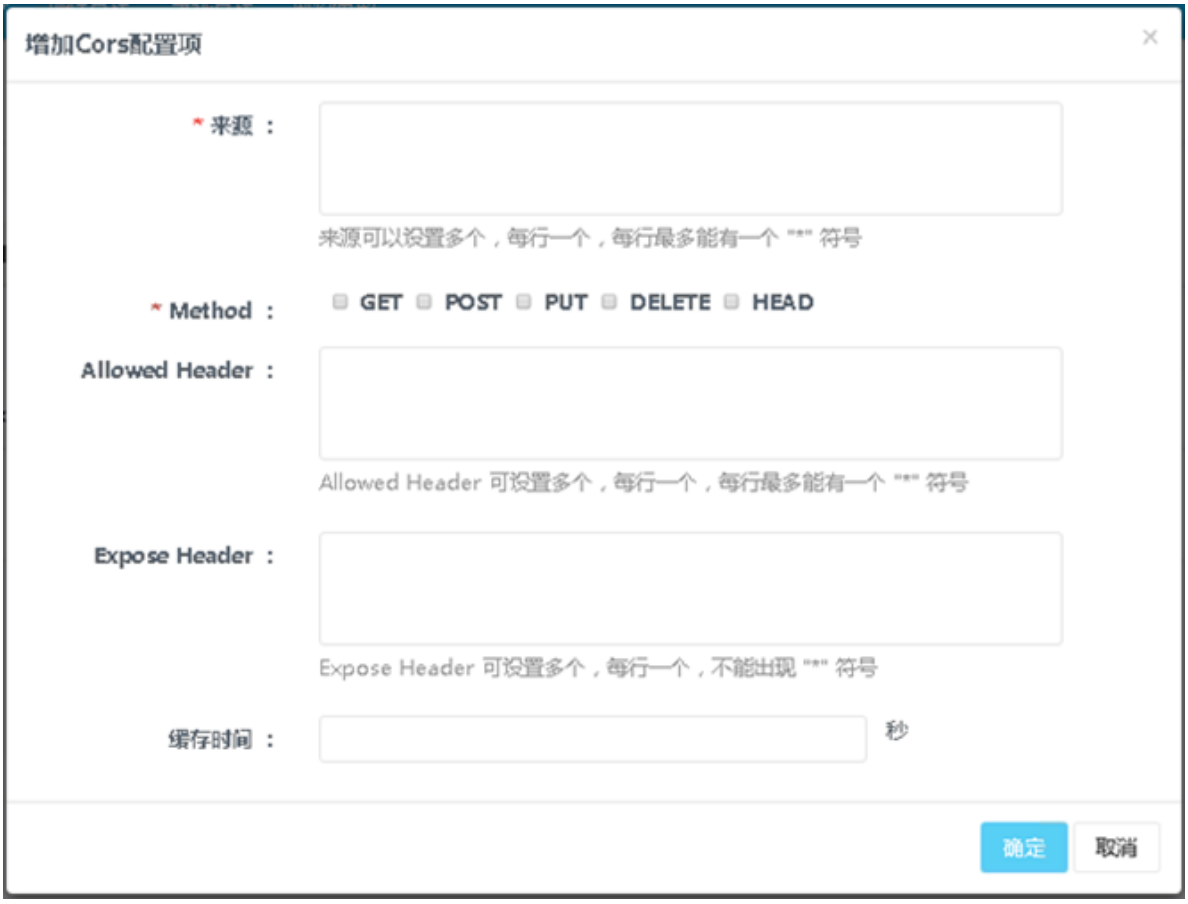
背景信息

OSS 提供 HTML5 协议中的跨域资源共享 CORS 设置，帮助您实现跨域访问。当 OSS 收到一个跨域请求（或者 OPTIONS 请求），会读取存储空间对应的 CORS 规则，然后进行相应的权限检查。OSS 会依次检查每一条规则，使用第一条匹配的规则来允许请求并返回对应的 header。如果所有规则都匹配失败，则不附加任何 CORS 相关的 header。

操作步骤

1. 登录 OSS 云控制台。
2. 单击目标存储空间的名称，进入存储空间管理页面。
3. 单击 **Bucket 属性** > **Cors设置**。
4. 单击**添加规则**，打开**增加CORS配置项**对话框。

图 323: 增加CORS配置项



5. 在对话框中配置对应的规则。每个存储空间最多可以配置 10 条规则。
   - **来源**：指定允许的跨域请求的来源。允许多条匹配规则，以回车为间隔。每个匹配规则允许使用最多一个“*”通配符。
   - **Method**：指定允许的跨域请求方法。

- **Allowed Header**：指定允许的跨域请求 header。允许多条匹配规则，以回车为间隔。每个匹配规则使用最多一个“*”通配符。
- **Expose Header**：指定允许用户从应用程序中访问的响应头。
- **缓存时间**：指定浏览器对特定资源的预取（OPTIONS）请求返回结果的缓存时间。

6. 单击**确定**保存该规则。同时也可以针对已经配置的规则进行修改或者删除。

13.5 管理对象（文件）

13.5.1 获取AccessKey

背景信息

用户的访问密钥AccessKey（AK）相当于登录密码，只是使用场景不同。AccessKey用于程序方式调用云服务API，登录密码用于登录控制台。如果用户不需要调用API，那么就不需要创建AccessKey。

AccessKey包括Access Key ID和Access Key Secret。Access Key ID用于标识用户，Access Key Secret是用来验证用户的密钥。Access Key Secret必须保密。

操作步骤

1. 登录*OSS* 云控制台。
2. 单击您要向其中上传文件的存储空间名称，打开该存储空间管理页面。
3. 单击 **Object 管理**，进入该存储空间内所有文件管理页面。
4. 单击**获取AccessKey**，打开**获取AccessKey**对话框。
5. 单击**显示AK**。系统显示Access Key ID和Access Key Secret。



注意: Access Key ID和Access Key Secret是您访问OSS的密钥，具有该账户完全的权限，请您妥善保管。

13.5.2 新建文件夹

背景信息

OSS 服务是没有文件夹这个概念的，所有元素都是以对象来存储。OSS 控制台中的文件夹本质上来说是创建了一个大小为 **0** 并以/结尾的对象用于同类文件的归类操作和批处理，同时控制台默认将以/结尾的对象作为文件夹形式展现。该对象同样可以上传及下载。用户可以在 OSS 控制台中，采用同 Windows 文件夹的基本操作使用 OSS 文件夹。

说明: 对于任何一个以/结尾的对象，不论该对象是否存有数据，在控制台中都是以文件夹的方式显示，您只能通过 API 或 SDK 的方式来下载该对象。

操作步骤

1. 登录OSS 云控制台。
2. 单击您要向其中上传文件的存储空间名称，打开该存储空间管理页面。
3. 单击**Object 管理**，进入该存储空间内所有文件管理页面。
4. 单击**新建文件夹**，打开**新建文件夹**对话框。

图 324: 新建文件夹



5. 在**文件夹名**框中输入该文件夹名称。
6. 单击**确认**保存已创建的文件夹。

13.5.3 搜索文件

背景信息

本部分介绍如何使用 OSS 云控制台在存储空间或文件夹中搜索具有相同的名称前缀的对象。

按名称前缀进行搜索时，搜索字符串将区分大小写，并且不能包含正斜杠“/”字符。搜索的范围限定为当前存储空间的根级别或当前文件夹中的对象（不包括子文件夹及其所包含的对象）。

操作步骤

1. 登录OSS 云控制台。
2. 单击您要向其中上传文件的存储空间名称，打开该存储空间管理页面。
3. 单击**Object 管理**，进入该存储空间内所有文件管理页面。
4. 在搜索框中输入要搜索的前缀内容，如 aliyun，按回车键或单击 **查询** 按钮开始搜索。系统将列出存储在存储空间的根目录级别且前缀为 aliyun 的对象和文件夹的名称。

如果要在某个文件夹中进行搜索，打开该文件夹后，在搜索框中输入要搜索的前缀内容进行搜索。系统将列出存储在该文件夹的根级别且前缀为搜索内容的对象和文件夹的名称。

13.5.4 设置文件 ACL

背景信息

您可以通过 OSS 控制台设置单个文件的读写权限。

操作步骤

1. 登录OSS 云控制台。
2. 单击目标存储空间的名称，进入存储空间管理页面。
3. 单击**Object 管理**，进入该存储空间内所有文件管理页面。
4. 选择目标文件，单击**管理** > **设置文件ACL** 。
5. 从**设置读写权限**下拉列表中选择合适的读写权限。

13.5.5 删除文件

背景信息

如果您不再需要存储所上传的文件，请将其删除以免占用空间。您可以通过 OSS 控制台删除单个文件或批量删除文件。

控制台批量删除文件，仅限制数量为 50 个文件。如果想更灵活地选择删除的文件，实现更大批量的删除，您可以选择 SDK 和 API 方式。

注意: 文件删除后无法恢复，请谨慎操作。

操作步骤

1. 登录OSS 云控制台。
2. 单击您要向其中上传文件的存储空间名称，打开该存储空间管理页面。
3. 单击**Object 管理**，进入该存储空间内所有文件管理页面。
4. 单击目标文件夹对应的**管理** > **删除** ，打开**删除Object**对话框。

   注意: 如您的文件夹下文件数量过多，可能导致删除文件夹失败。

5. 单击 **确认** 删除该文件夹。

13.6 图片处理

阿里云OSS图片处理服务（Image Service，简称 IMG），是阿里云OSS对外提供的海量、安全、低成本、高可靠的图片处理服务。

您可以将原始图片上传保存在OSS上，通过简单的 RESTful 接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。

图片处理服务提供图片处理接口，图片上传请使用OSS上传接口。基于IMG，您可以搭建出跟图片相关的服务。

13.6.1 创建样式

操作步骤

1. 登录OSS 云控制台。
2. 单击目标存储空间的名称，进入存储空间管理页面。
3. 单击**图片处理**。
4. 单击**创建样式**打开窗口如图：图 *325:* 创建样式所示。

图 325: 创建样式



图片样式窗口说明：

- 规则名：新建图片样式的名称，建议取一个有意义的名称，方便您记住，比如XX水印图旋转。
- 编辑类型：可以选择基本编辑，使用图形化的操作来编辑图片样式。也可以选择高级编辑，使用SDK或者处理参数的方式编辑图片样式。
- 图片预览：选择图片预览的方式。

- 缩略方式：设置图片的缩略方式。
- 缩略宽度：设置图片的缩略尺寸。
- 缩略限制：设置是否限制图片放大。
- 适应方向：设置图片的适应方向。
- 图片处理：设置图片是否需要特殊处理。
- 图片质量：设置图片质量。
- 保存格式：设置图片的保存格式。
- 添加水印：设置图片水印方式。

5. 编辑好图片样式后单击**提交**保存样式。
6. 样式提交后可以单击**导出样式**将样式下载到本地。

14 消息服务

14.1 产品概述

阿里云消息服务（Message Service，简称 MNS）是一种高效、可靠、安全、便捷、可弹性扩展的分布式消息服务。MNS 能够帮助应用开发者在他们应用的分布式组件上自由地传递数据、通知消息，构建松耦合系统。

消息服务提供了队列模型，旨在提供高可靠高并发的一对一消费模型，即队列中的每一条消息都只能够被某一个消费者进行消费。队列模型就如同一家旋转寿司店，寿司店中有多个寿司师傅（生产者）在制作精美的寿司（消息），每一份寿司都是独特的，每位顾客（消费者）同时从传送带上拿取中意的寿司进行食用（消费）。

14.2 使用前须知

阿里云消息服务 MNS 为您提供专业的消息中间件服务，让您轻松实现系统解耦，从容面对高并发的任务处理。

您可以通过MNS 的管理控制台进行基本的队列以及主题订阅操作；同时您还可以通过阿里云开发工具包或者直接在应用程序中进行 RESTful API 调用执行相应的操作。

14.3 基本概念

队列

队列是消息存储的目的地，队列可以分成普通队列和延时队列两类。消息包含数据和属性，可以分成普通消息和延时消息，在一个队列中能唯一标识一个消息的有 MessageId 和 ReceiptHandle 两种。

普通队列

如果发送消息时不指定消息延时参数，消息发送到普通队列后可立即被消费。

延时队列

如果发送消息时不指定消息延时参数，发送到延时队列的消息需要经过一定的时间后才能被取到。您可以通过 CreateQueue 和 SetQueueAttribute 指定队列的延时时长。

生产者、消费者

生产者是向 MNS 的消息队列发送消息的角色。

消费者是从 MNS 的消息队列获取消息的角色。

Endpoint

MNS 的访问域名地址，格式如下：http://$AccountId.mns.$Region.aliyuncs.com/

- mns.<Region>.aliyuncs.com：Region 是 MNS 服务部署的地域。
- AccountId：队列所有者的账号 ID，调用 API 请求时需替换成实际的账号 ID。

普通消息、延时消息

- 普通消息是指被发送到队列马上可以被取到的消息。
- 延时消息是指消息发送到队列后需要经过一定的时间后才能被取到的消息，延时时长由 DelaySeconds 属性决定（请参考 SendMessage 相关属性决定）。

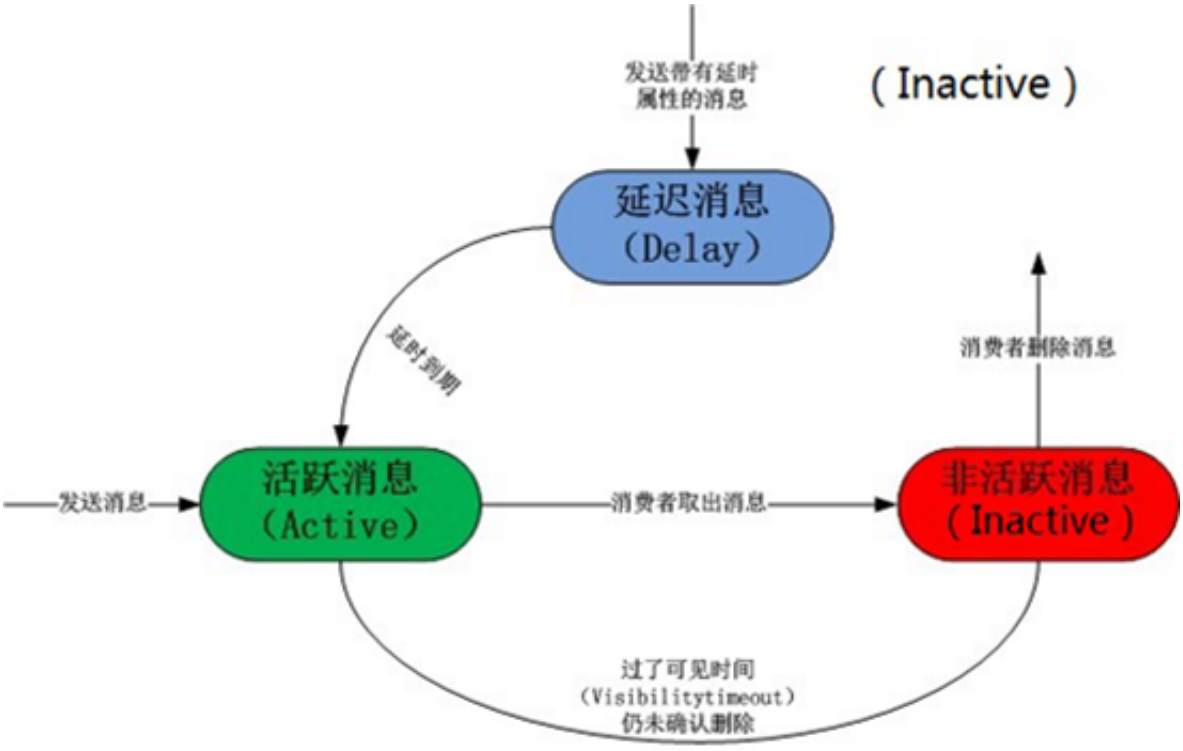
消息 ID（MessageID）

MessageId 用来标识在队列中的一个消息，在一个队列中每个消息都有唯一的 MessageId，但在不同队列之间并不唯一。当消息发送到 MNS 的队列，MNS 会生成一个 MessageId，此 ID 一旦产生就不会改变，并在请求响应中返回，您可以使用此 MessageId 来做数据校对，但是删除消息必须使用 ReceiptHandle。

临时句柄（ReceiptHandle）

ReceiptHandle 是由 MNS 根据当时获取消息的行为产生的临时消息标识，这一点不同于 MessageId。当要删除已消费的消息或者改变其 VisiblityTimeout 属性值，需要用 ReceiptHandle 指定已消费过的消息。ReceiptHandle 只能被使用一次，如果 ReceiptHandle 标识的消息状态改变其就会失效，消费者只有通过获取消息后重新获取 ReceiptHandle 才能进行上述操作。

消息状态（Message Status）



消息状态转换示意图

普通消息被发送到普通消息队列时，初始状态是 Active，当其被取走后在 VisibilityTimeout 的时间内状态为 Inactive，若超过 VisibilityTimeout 时间后消息还未被删除，消息会重新变成 Active 状态；如果在 VisibilityTimeout 时间内被删除，消息状态为 Deleted。

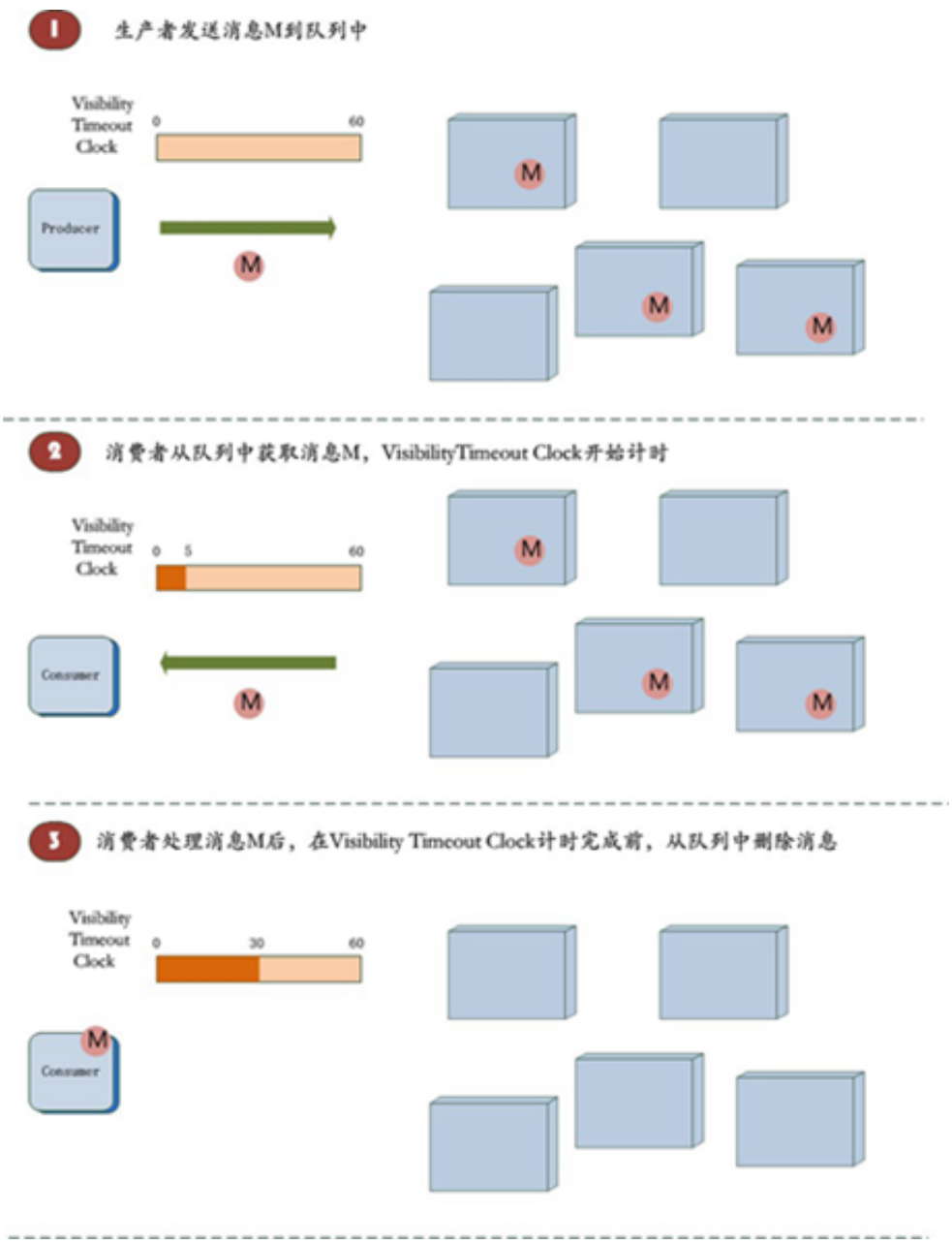
普通消息发送到延时队列时，消息初始状态Delayed，经过延时队列的 DelaySeconds 属性值设定的时间后，消息状态变成 Active。

延时消息发送到队列（普通队列或者延时队列），消息的初始状态为 Delayed，经过消息的 DelaySeconds 属性指定的时间后消息状态变成 Active。

消息的最长存活时间由创建队列时指定的 MessageRetentionPeriod 属性值决定，超过此时间后消息状态会变成 Expired，将被垃圾回收器回收。

消费者只能取到处于 Active 状态的消息。

消息生命周期



14.4 使用子账号访问 MNS

本节介绍如何使用子账号访问消息服务。

14.4.1 创建子账号

操作步骤

1. 进入控制台：选择**产品** > **管理与监控** > **访问控制** 进入访问控制产品页。单击 **获取使用资格** 申请使用权限，然后单击 **管理控制台**进入控制台。

2. 依次单击**用户管理** > **新建用户**（右上角）进入新建用户页面。
3. 在创建用户对话框里输入：登录名（必填）和显示名、备注、邮箱、电话等（可选），同时勾选：**为该用户自动生成AccessKey。**
4. 点击 **确定** ，完成子账号创建。创建成功后，会提示是否下载子账号 AK，请选择下载并妥善保管子账号 AK 文件。

14.4.2 新建授权策略

操作步骤

1. 在 RAM 控制台上依次单击：**授权策略管理** > **自定义授权策略** > **新建授权策略。**
2. 在弹出对话框中选择**授权策略模板**（使用空模板）。
3. 编辑授权策略并提交：修改**授权策略名称**（自定义名称）、**备注**、**策略内容**，并提交。

   一个 MNS 授权策略内容模板示例：

```
{
"Statement":[
{
"Action":"mns:*",
"Effect": "Allow","Resource": "acs:mns:*:*:*"
}
],
"Version":"1"
}
```

   授权策略是 JSON 格式的字符串，其中，

   - **Action**：表示要授权的操作，MNS 操作都以 `mns:` 开头。

     例如：`mns:SendMessage` 表示 MNS 服务的 API：SendMessage/BatchSenMessage。其他请参见 *MNS API* 授权映射表。
   - **Effect**：表示授权类型，例如：Allow、Deny。
   - **Resrouce**：表示要授权的阿里云资源名（ARN），格式为：`acs:<云服务名>: <地域名>: <主账号UID>:<资源URI>`

     例如：`acs:mns:cn-hangzhou:123456789:/queues/MyQueue1/messages` 表示：授权资源是主账号 UID 为 123456789 在 cn-hangzhou 地域的 MyQueue1。用 "*" 表示不指定具体的字段，例如：`"acs:mns:*:*:*"`表示不指定地域名，主账号 ID 和资源 URI，子账号可以访问主账号的所有 MNS 资源。

14.4.3 授权子账号访问 MNS

操作步骤

1. 返回 **用户管理**，找到之前创建的子账号，单击右侧 **授权**。
2. 在弹出的对话框中，选择授权策略名称，并添加到右侧已选授权策略列表，单击 **确定** 提交。
3. 下载最新版本 mnspython SDK 到 ECS 服务器并解压。
4. 进入 SDK 代码目录，修改 sample.cfg，配置步骤 1 中下载的子账号 AK（AccessKeyId，AccessKeySecret ）和主账号的 Endpoint。
5. 运行 sample 命令：`$pythonsample.py` 。如果没有报错，说明子账号 AK 能够成功访问主账号的 queue。

14.4.4 授权策略示例

14.4.4.1 示例 1：在 Allow 授权中增加 IP 限制

允许通过 1.10.88.0/24 和 1.10.66.0/24 两个 IP 段访问 MNS。

```
{
"Statement": [
{
"Action": "mns:*",
"Effect": "Allow", "Resource": "acs:mns:*:*:*"
}
],
"Version": "1", "Condition":{
"IpAddress": {
"acs:SourceIp": ["1.10.88.0/24", "1.10.66.0/24"]
}
}
}
```

14.4.4.2 示例 2：在 Deny 授权中增加 IP 限制

如果源 IP 不在 1.10.88.0/24 中，则禁止对 MNS 执行任何操作。

```
{
"Statement": [
{
"Action": "mns:*",
"Effect": "Deny", "Resource": "acs:mns:*:*:*"
}
],
"Version": "1", "Condition":{ "NotIpAddress": {
"acs:SourceIp": ["1.10.88.0/24"]
}
}
```

```
}
```

说明: 由于Policy 的鉴权规则是 Deny 优先（即如果用户的访问操作命中任意一条 Deny 规则，则禁止访问），故访问者从 1.10.88.0/24 以外的 IP 地址访问时，MNS 服务会报没有权限。

14.4.4.3 示例 3：授予 MNS 控制台可读权限

仅允许子账号拥有查看队列的权限。

```
{
"Version": "1", "Statement": [
{
"Effect": "Allow", "Action": [ "mns:ListQueue",
"mns:GetQueueAttributes",
],
"Resource": "acs:mns:*:*:*"
}
]
}
```

为保证安全，在添加子账号、添加授权策略和增加授权时，需要手机短信或邮件验证。

14.4.5 MNS API授权映射表

ListQueue	mns:ListQueue	acs:mns:$region:$accountid:/queues
CreateQueue	mns:CreateQueue	acs:mns:$region:$accountid:/queues/$queueName
DeleteQueue	mns:DeleteQueue	acs:mns:$region:$accountid:/queues/$queueName
SetQueueAttributes	mns:SetQueueAttributes	acs:mns:$region:$accountid:/queues/$queueName
GetQueueAttributes	mns:GetQueueAttributes	acs:mns:$region:$accountid:/queues/$queueName
SendMessage/ BatchSendMessage	mns:SendMessage	acs:mns:$region:$accountid:/queues/$queueName/messages
ReceiveMessage/ BatchReceiveMessage	mns:ReceiveMessage	acs:mns:$region:$accountid:/queues/$queueName/messages
DeleteMessage	mns:DeleteMessage	acs:mns:$region:$accountid:/queues/$queueName/messages

PeekMessage/ BatchPeekMessage	mns:PeekMessage	acs:mns:$region:$accountid:/ queues/$queueName/messa ges
ChangeMessageVisibility	mns:ChangeMessageVisibility	acs:mns:$region:$accountid:/ queues/$queueName/messs ages
ListTopic	mns:ListTopic	acs:mns:$region:$accountid:/ topics
CreateTopic	mns:CreateTopic	acs:mns:$region:$accountid:/ topics/$topicName
DeleteTopic	mns:DeleteTopic	acs:mns:$region:$accountid:/ topics/$topicName
SetTopicAttributes	mns:SetTopicAttributes	acs:mns:$region:$accountid:/ topics/$topicName
GetTopicAttributes	mns:GetTopicAttributes	acs:mns:$region:$accountid:/ topics/$topicName
ListSubscriptionByTopic	mns:ListSubscriptionByTopic	acs:mns:$region:$accountid:/ topics/$topicName/subscript ions
Subscribe	mns:Subscribe	acs:mns:$region:$accountid:/ topics/$topicName/subscript ions/$subscriptionName
Unsubscribe	mns:Unsubscribe	acs:mns:$region:$accountid:/ topics/$topicName/subscript ions/$subscriptionName
SetSubscriptionAttributes	mns:SetSubscriptionAttributes	acs:mns:$region:$accountid:/ topics/$topicName/subscript ions/$subscriptionName
GetSubscriptionAttributes	mns:GetSubscriptionAttributes	acs:mns:$region:$accountid:/ topics/$topicName/subscript ions/$subscriptionName
PublishMessage	mns:PublishMessage	acs:mns:$region:$accountid:/ topics/$topicName/message s

14.5 管理队列

14.5.1 创建队列

背景信息

首先使用 MNS 创建一个队列，在以下的示例中命名为 **MyQueue**。

操作步骤

1. 单击 **创建队列** 进入创建队列的对话框。
2. 在对话框中填写队列属性，单击 **确定**，队列创建成功。

14.5.2 设置队列属性

背景信息

确定 **MyQueue**创建成功后，对 **MyQueue**的属性进行设置。

操作步骤

1. 单击 **MyQueue** 右侧操作列表中的 **修改设置** 进入修改队列页面。
2. 在弹出的对话框中，修改对应的属性，单击 **确认** 即可修改队列属性。

14.5.3 获取队列属性

背景信息

确定 **MyQueue** 创建成功后，获取 **MyQueue** 的队列属性。

操作步骤

1. 控制台默认显示队列基本属性，直接查看即可。
2. 单击 **MyQueue** 右侧操作列表中的 **获取地址** 可查看队列的公网地址和私网地址。

14.5.4 删除队列

背景信息

确定 **MyQueue** 创建成功后，删除 **MyQueue**。

操作步骤

1. 单击 **MyQueue** 右侧的 **删除** 进入删除页面。
2. 单击 **删除** 即可删除队列。

14.5.5 发送消息

背景信息

确定 **MyQueue** 创建成功后，向 **MyQueue** 发送消息。

操作步骤

1. 单击 **MyQueue** 右侧的 **发送消息** 。
2. 在弹出对话框中，输入要发送的消息，单击 **发送消息**，消息发送成功。

14.5.6 消费消息

背景信息

确定 **MyQueue** 创建成功后，从 **MyQueue** 中取出 message，跟我们发送到 **MyQueue** 的消息进行对比，确定拿到的消息跟我们发送的消息一致。

操作步骤

单击 **MyQueue**右侧的 **接收消息**，弹出的对话框中显示接收消息的相关属性。

14.5.7 删除消息

背景信息

验证消息正确后，删除刚才取出来的 message。

操作步骤

在接收消息的对话框中，选中 **自动删除已接收消息**，单击 **接收消息** 会删除本条消息，同时获取下一条消息。

15 表格存储TableStore

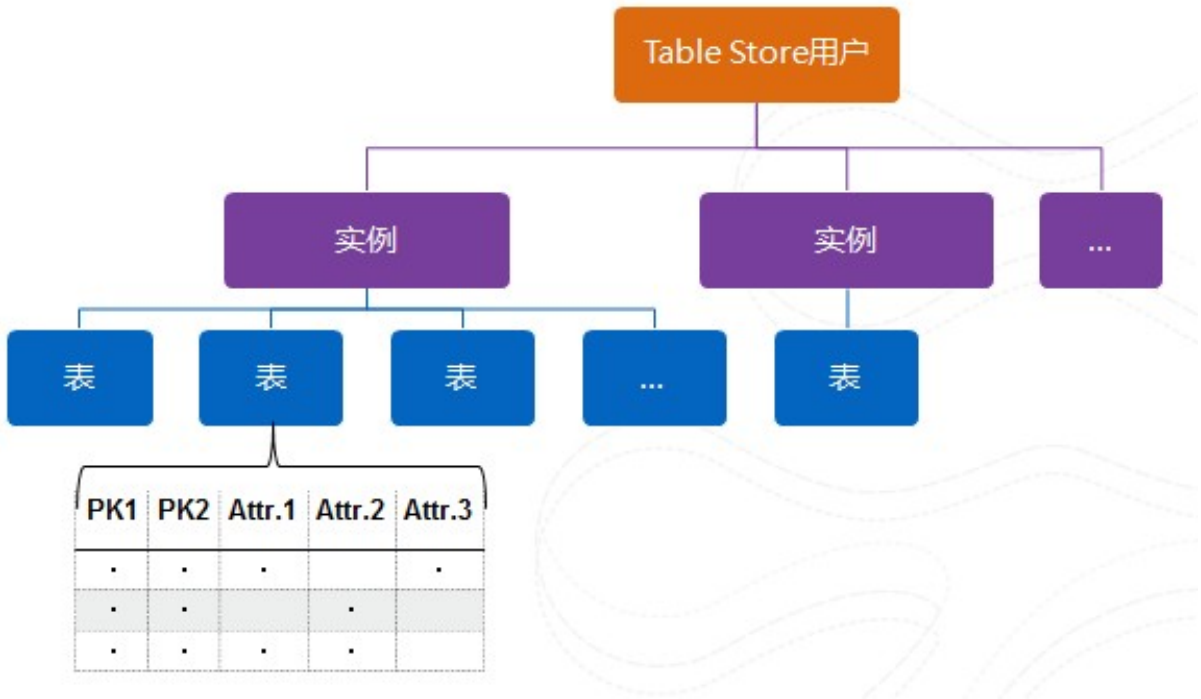
15.1 什么是表格存储

表格存储（Table Store）是构建在阿里云飞天分布式系统之上的 NoSQL 数据存储服务，提供海量结构化数据的存储和实时访问。

- 表格存储以实例和表的形式组织数据，通过数据分片和负载均衡技术，达到规模的无缝扩展。
- 表格存储向应用程序屏蔽底层硬件平台的故障和错误，能自动从各类错误中快速恢复，提供非常高的服务可用性。
- 表格存储管理的数据全部存储在 SSD 中并具有多个备份，提供了快速的访问性能和极高的数据可靠性。

15.2 实例介绍

图 326: 实例



实例是用户使用和管理表格存储服务的实体。用户在开通表格存储服务之后，需要通云控制台来创建实例，然后在实例内进行表的创建和管理。

实例是表格存储资源管理的基础单元，表格存储对应用程序的访问控制和资源计量都在实例级别完成。

15.3 快速开始

本操作指引您快速完成基本任务，包括创建实例和创建表。

15.3.1 创建表

操作步骤

1. 登录表格存储云控制台。
2. 找到要管理的实例，单击该实例的名称，进入**实例详情**页面。
3. 单击**数据表列表**页签。
4. 在**数据表列表**页面，单击**创建数据表**。

   说明: 单个实例最多可以创建 64 张数据表。

5. 填写数据表信息。

   表 38: 数据表参数说明

参数	说明
数据表名称	数据表名称只能由大小写字母、数字、下划线组成，由字母、下划线开头。 数据表名称在实例级别中必须是唯一的。
预留读吞吐量	预留读/写吞吐量可以设置为 0。当预留读/写吞吐量大于 0 时，表格存储根据该配置为表分配和预留相应的资源，从而获得更低的资源使用成本。 取值范围为 0-5000 且取值必须为整数。 容量型实例不支持该参数。
预留写吞吐量	
数据生命周期	最低为 86400 秒（一天）或者 -1（永不过期）。
最大数据版本	非 0 值。 表示该数据表中的属性列能够保留多少个版本的数据。当一个属性列的版本个数超过最大数据版本 时，最早的版本将被异步删除。
数据有效版本偏差	写入数据所有列的版本号与写入时时间的差值需要在数据有效版本偏差范围之内，否者将会写入失败。

参数	说明
	属性列的有效版本范围为：[数据写入时间 - 有效版本偏差，数据写入时间 + 有效版本偏差)。
表主键	最多可设置 4 个主键。第一个主键默认为分区键。 主键可选类型为 **整型Interger** 和 **字符串String**，主键的配置及顺序一旦设置便不可修改。单击**添加主键**加入新的主键。 主键名称只能由大小写字母、数字、下划线组成，由字母、下划线开头。

6. 单击**确定**。

   系统自动返回**实例详情**页面，显示建表结果。建表成功后，表将显示在数据表列表栏中。若新建的表未在列表中显示，可刷新列表页。

15.4 管理实例

15.4.1 查看实例

操作步骤

1. 登录表格存储云控制台。
2. 找到要查看的实例，单击该实例的名称，进入**实例详情**页面。

   您可以查看如下信息：实例状态、所属区域、创建时间、实例的外网访问地址和内网访问地址。

15.5 管理数据表

15.5.1 更新表

高性能实例可以通过表格存储控制台来调整数据表的预留读/写吞吐量，容量型实例可以通过表格存储控制台来调整数据表的数据生命周期、最大数据版本和数据有效版本偏差。

15.5.1.1 调整高性能实例的预留读/写吞吐量

操作步骤

1. 登录表格存储云控制台。
2. 找到要管理的实例，单击该实例的名称，进入**实例详情**页面。
3. 单击**数据表列表**页签，进入**数据表列表**页面。
4. 在数据表列表栏中找到需要更新的表，并单击其右侧操作栏下图标中的**调整读写吞吐量**。

5. 填入新的**预留读吞吐量**及**预留写吞吐量**，取值在 0-5000 之间，且调整操作的间隔应大于 2 分钟。
6. 单击**确定**，返回**实例详情**页面，即可看到新的预留读/写吞吐量。

15.5.2 查看数据表基本详情

背景信息

您可以在表管理页面中查看该表的基本信息及实际使用情况，例如：

- 数据表名称
- 预留读/写吞吐量
- 最后一次调整时间
- 主键（按照建表时主键顺序显示）

操作步骤

1. 登录表格存储云控制台。
2. 找到要查看的实例，单击该实例的名称，进入**实例详情**页面。
3. 单击**数据表列表**页签，进入**数据表列表**页面。
4. 单击要查看的表的名称，进入数据表**基本信息**页面，查看该表的基本信息。

15.5.3 删除表

背景信息

注意: 删除数据表后，表内的数据将无法恢复。

操作步骤

1. 登录表格存储云控制台。
2. 找到要管理的实例，单击该实例的名称，进入**实例详情**页面。
3. 单击**数据表列表**页签，进入**数据表列表**页面。
4. 在数据表列表栏中找到需要删除的表，并单击其右侧操作栏下图标中的**释放**。
5. 单击提示框中的**确定**按钮。确定删除后，表及表内的数据将被永久删除。

15.6 管理VPC

专有网络 VPC（Virtual Private Cloud）是用户基于阿里云构建出的一个隔离的网络环境。用户可以完全掌控自己的虚拟网络，比如选择私有 IP 地址的范围、划分网段、配置路由表和网关等，并可以通过专线 VPN 等连接方式把 VPC 与传统数据中心组成一个按需定制的网络环境，实现用户的平滑上云。

前提条件

- 用户需先创建 VPC，在创建时注意选择合适的节点，后续确保 VPC 和表格存储的实例在相同的节点。关于如何创建VPC，请参见创建*VPC*。
- 专有网络创建成功后，需在 VPC 内创建 ECS 实例。关于如何创建VPC实例，请参见创建*ECS*实例。

操作步骤

1. 登录表格存储云控制台。
2. 找到要管理的实例，单击该实例的名称，进入**实例详情**页面。
3. 单击**绑定 VPC**，进入实例和 VPC 绑定页面。
4. 填写信息，然后单击**确定**。
5. 绑定成功后系统自动返回实例详情页面，即可在 VPC 列表栏中看到已绑定的 VPC 信息。单击 VPC ID 栏中的链接可以查看该 VPC 下绑定的表格存储实例和 VPC 信息列表。

   之后在 VPC 内的 ECS 上就可以使用该实例访问表格存储服务，访问的 endpoint 使用 VPC 访问地址。

后续操作

使用完毕后，可以在 VPC 列表中单击该 VPC 右侧操作栏下的图标中**解除绑定**来删除实例和 VPC 的绑定关系。

解除绑定之后，在该 VPC 内的 ECS 上就无法使用上述地址来访表格存储了，如仍要访问需再次绑定。

15.7 附录：限制说明

以下为表格存储的限制项说明，部分限制范围表明您可以使用的最大值，并不是建议值。为保证更好的性能，请根据实际情况合理设计表结构和单行数据大小，并对下述配置进行调整。

限制项	限制范围	说明
一个用户账号下可以保有实例数	不超过 10	如有需求提高上限，请联系技术支持。
一个实例中表的个数	不超过 64	如有需求提高上限，请联系技术支持。
实例名字长度	3-16 bytes	字符集为 [a-z，A-Z，0-9]和连词符（-），首字符必须是字母且末尾字符不能为连词符（-）。
表名长度	1-255 bytes	字符集为[a-z，A-Z，0-9，_]，首字符必须是字母或下划线（_）。
列名长度限制	1-255 bytes	字符集为[a-z，A-Z，0-9，_]，首字符必须是字母或下划线（_）。
主键包含的列数	1-4	至少 1 列，至多 4 列。
String 类型主键列列值大小	不超过 1 KB	单一主键列String类型的列列值大小上限 1 KB。
String 类型属性列列值大小	不超过 2 MB	单一属性列 String类型的列列值大小上限 2 MB。
Binary 类型主键列列值大小	不超过 1 KB	单一主键列 Binary类型的列列值大小上限 1 KB。
Binary 类型属性列列值大小	不超过 2 MB	单一属性列Binary类型的列列值大小上限 2 MB。
一行中属性列的个数	不限制	不限制单一行拥有的属性列个数。
单行数据大小	不限制	不限制单一行中所有列名与列值总和大小。

16 云数据库RDS版

16.1 产品概述

阿里云关系型数据库（Relational Database Service，简称RDS）是一种稳定可靠、可弹性伸缩的在线数据库服务。基于阿里云分布式文件系统和高性能存储，云数据库提供了容灾、备份、恢复、监控、迁移等方面的全套解决方案，彻底解决数据库运维的烦恼。

云数据库MySQL版基于Alibaba的MySQL源码分支，经过双11高并发、大数据量的考验，拥有优良的性能和吞吐量。除此之外，云数据库MySQL版还拥有经过优化的读写分离、数据压缩、智能调优等高级功能。

MySQL是全球最受欢迎的开源数据库，作为开源软件组合LAMP（Linux + Apache + MySQL + Perl/PHP/Python）中的重要一环，广泛应用于各类应用。

Web2.0时代，风靡全网的社区论坛软件系统Discuz和博客平台Wordpress均基于MySQL实现底层架构。Web3.0时代，阿里巴巴、Facebook、Google等大型互联网公司都采用更为灵活的MySQL构建了成熟的大规模数据库集群。

16.2 使用限制

为保障实例的稳定及安全，云数据库MySQL版有部分使用上的约束，详情如表 *39:* 使用限制所示。

表 39: 使用限制

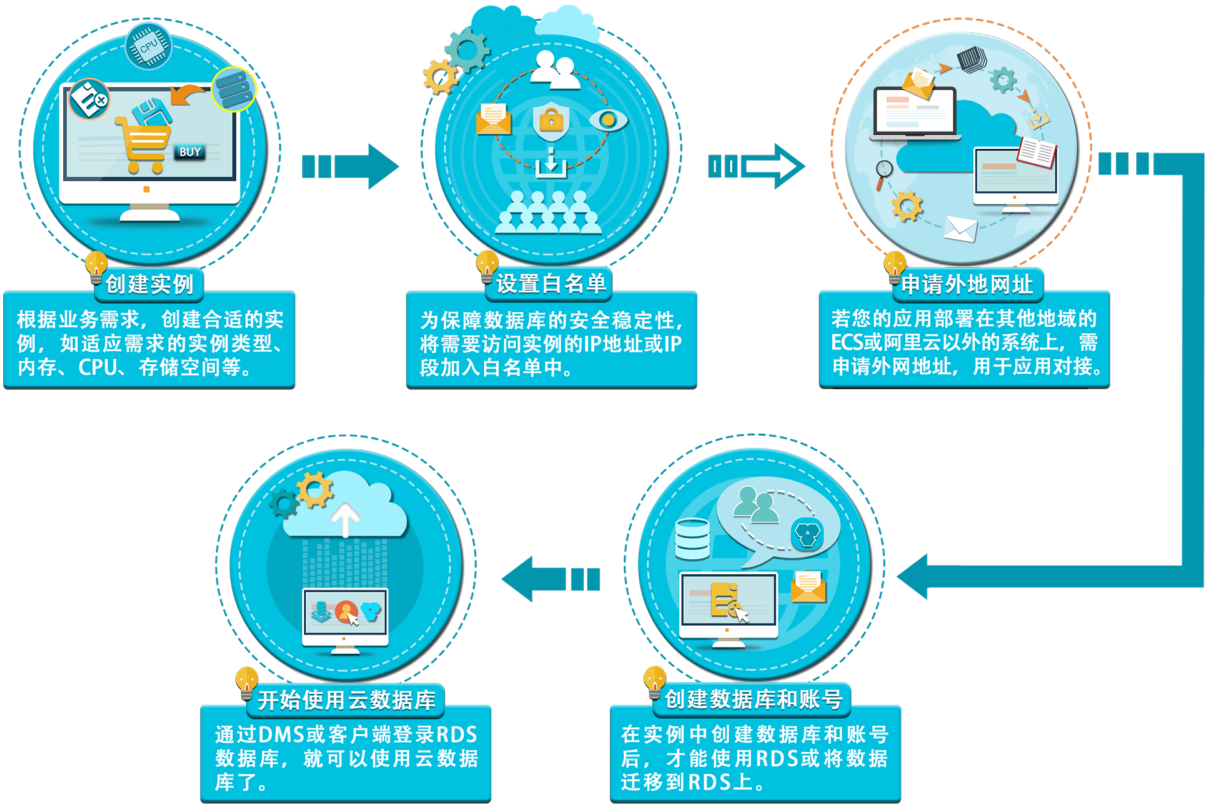
操作	使用限制
修改数据库参数设置	大部分数据库参数须使用RDS控制台或API进行修改，同时出于安全和稳定性考虑，部分参数不支持修改。
数据库的 root 权限	不提供root或者sa权限。
数据库备份	可使用命令行或图形界面进行逻辑备份。仅限通过RDS控制台或API进行物理备份。
数据库还原	可使用命令行或图形界面进行逻辑数据还原。仅限通过RDS控制台或API进行物理还原。
数据迁入	可以使用命令行或图形界面进行逻辑导入。可以使用MySQL命令行工具、数据传输服务等方式迁入数据。

操作	使用限制
MySQL 存储引擎	目前支持InnoDB、TokuDB两种引擎（MyISAM引擎由于自身缺陷，存在数据丢失的风险，因此仅部分存量实例暂时支持，新创建实例的MyISAM引擎表会自动转换为InnoDB引擎表）。出于性能和安全性考虑建议尽量采用InnoDB存储引擎。不支持Memory引擎。如果您创建Memory引擎的表，我们将自动为您转换成InnoDB引擎的表。
搭建数据库复制	云数据库MySQL版本身提供主备复制架构的双节点集群，无需用户手动搭建。其中主备复制架构集群的备（slave）实例不对用户开放，用户应用不可直接访问。
重启 RDS 实例	必须通过RDS控制台或API操作重启实例。
用户、密码管理和数据库管理	云数据库MySQL版默认需要通过RDS控制台进行用户、密码和数据库管理（包括创建、删除、修改权限、修改密码）。

16.3 使用流程

通常，从创建实例到可以开始使用示例，您需要完成如下操作：

图 327: 快速入门流程图



16.4 初始化配置

16.4.1 设置白名单

为了数据库的安全稳定，在开始使用RDS实例前，您需要将访问数据库的IP地址或者IP段加到目标实例的白名单中。正确使用白名单可以让RDS得到高级别的访问安全保护，建议您定期维护白名单。本文将主要介绍设置白名单的操作步骤。

背景信息

注意:

- 系统会给每个实例创建一个默认的**default**白名单分组，该白名单分组只能被修改或清空，但不能被删除。
- 对于新建的RDS实例，系统默认会将0.0.0.0/0添加到**default**白名单分组中，IP地址0.0.0.0/0代表允许任何IP访问该RDS实例。该设置将极大降低数据库的安全性，如非必要请勿使用。所以，在您设置白名单时，需要先将0.0.0.0/0删除，然后再添加您允许访问该RDS实例的IP地址或IP段。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**安全控制** > **白名单设置**，进入**白名单列表**页面。
4. 单击**添加白名单分组**。
5. 在**修改白名单分组**页面的**组内白名单**栏中填写需要访问该实例的IP地址或IP段，如图 *328: 添加白名单分组*所示。然后单击**确定**。

图 328: 添加白名单分组



参数说明：

- **分组名称**：长度为2~32个字符，由小写字母、数字或下划线组成，开头需为小写字母，结尾需为字母或数字。在白名单分组创建成功后，该名称将不能被修改。
- **组内白名单**：填写允许访问RDS实例的IP地址或者IP段。
  - 若填写IP段，如10.10.10.0/24，则表示10.10.10.X的IP地址都可以访问该RDS实例。
  - 若您需要添加多个IP，请用英文逗号隔开（逗号前后都不能加空格），例如192.168.0.1,172.16.213.9。

16.4.2 创建数据库和账号

若要使用云数据库，您需要在实例中创建数据库和账号。本文将介绍在RDS控制台上创建数据库和账号的操作步骤。

背景信息

注意:

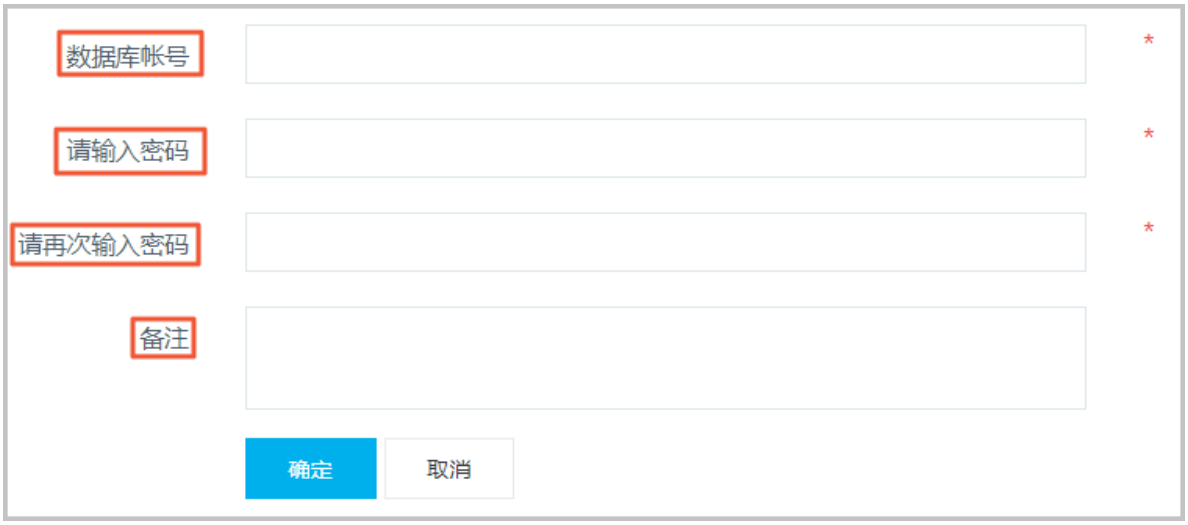
- 同一实例下的数据库共享该实例下的所有资源，每个MySQL 5.6版本的实例最多可以创建500个数据库和500个账号。

- 如果您要迁移本地数据库到RDS，请在RDS实例中创建与本地数据库一致的迁移账号和数据库。
- 分配数据库账号权限时，请按最小权限原则和业务角色创建账号，并合理分配只读和读写权限。必要时可以把数据库账号和数据库拆分成更小粒度，使每个数据库账号只能访问其业务之内的数据。如果不需要数据库写入操作，请分配只读权限。
- 为保障数据库的安全，请将数据库账号的密码设置为强密码，并定期更换。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**数据库管理** > **账号列表**，进入**账号列表**页面。
4. 单击**添加账号**。
5. 输入要创建的账号信息，如图 *329:* 创建账号所示。

图 329: 创建账号



参数说明：

- **数据库账号**：由 2~16 个字符的小写字母，数字或下划线组成、开头需为字母，结尾需为字母或数字，如 *user4example*。
- **请输入密码**：该账号对应的密码，由 6~32 个字符的字母、数字、中划线或下划线组成，如 *password4example*。
- **请再次输入密码**：输入与密码一致的字段，如 *password4example*，以确保密码正确输入。
- **备注**：可以备注该账号的相关信息，便于后续账号管理，最多支持 256 个英文字符。

6. 单击**确定**。
7. 在左侧导航栏中，选择**数据库管理** > **数据库列表**，进入**数据库列表**页面。
8. 单击**添加数据库**。
9. 输入要创建的数据库信息，如图 *330:* 创建数据库所示。

图 330: 创建数据库



参数说明：

- **数据库(DB)名称**：由 2~64 个字符的小写字母、数字、下划线或中划线组成，开头需为字母，结尾需为字母或数字。
- **支持字符集**：设置数据库的字符集，选项有 utf8、gbk、latin1 和 utf8mb4。
- **账号授权**：选择该数据库授权的账号。如果尚未创建账号，该值可以为空。
- **备注说明**：可以备注该数据库的相关信息，便于后续数据库管理，最多支持 256 个英文字符。

10.单击**确定**。

16.5 连接实例

16.5.1 通过DMS登录实例

本章介绍如何通过DMS连接RDS实例。

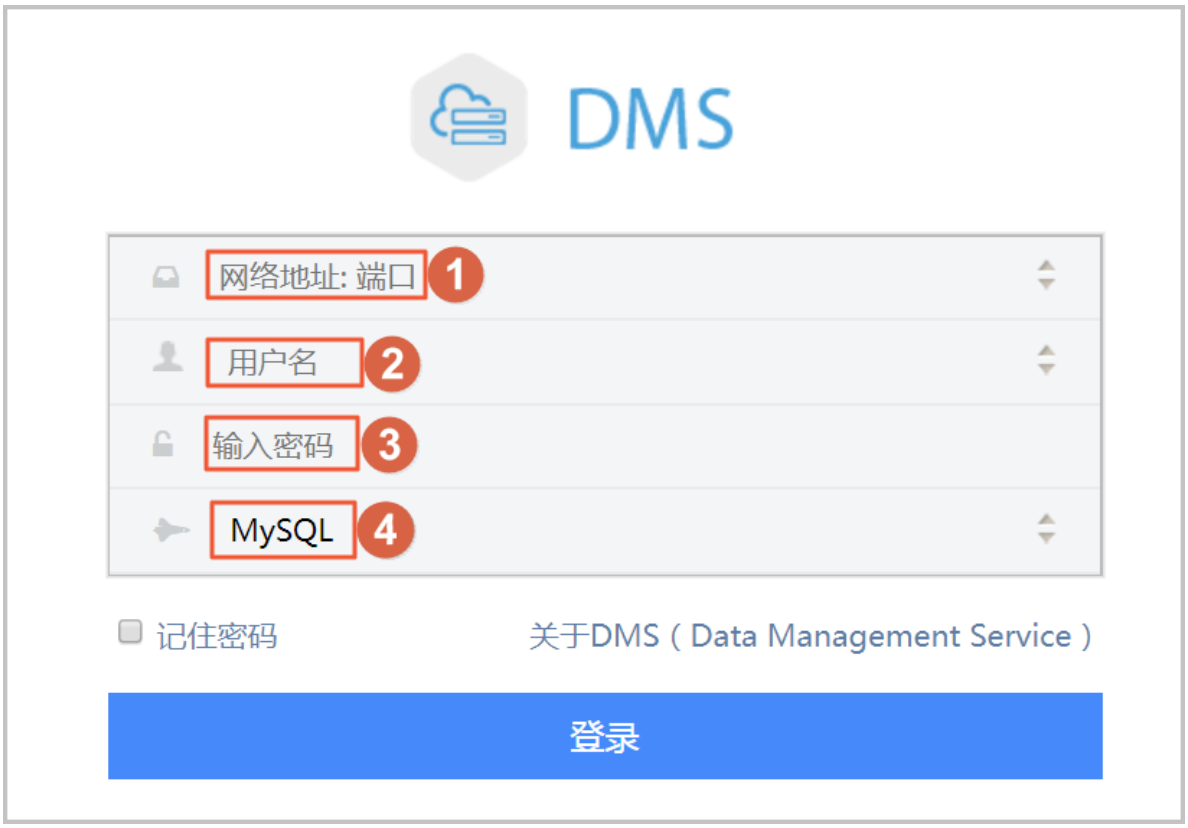
背景信息

您可以通过RDS控制台先登录DMS，然后再连接需要访问的RDS实例。数据管理（Data Management，简称DMS）是一种集数据管理、结构管理、访问安全、BI图表、数据趋势、数据轨迹、性能与优化和服务器管理于一体的数据管理服务。支持MySQL、SQL Server、PostgreSQL、MongoDB、Redis等关系型数据库和NoSQL的数据库管理，同时还支持Linux服务器管理。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的ID，进入**基本信息**页面。
3. 单击页面右上角的**登录DMS**，进入数据管理（DMS）控制台的登录页面。
4. 输入登录信息，如图 *331: DMS*登录页面所示。

图 331: DMS登录页面



参数说明：

- 1：指要连接的实例的内网或外网地址及其多对应的端口号，例如“rm-test00000k012.mysql.aliyun-inc.com:3306”。查找内外网地址及其对应端口号的步骤如下：

  1. 登录*RDS*控制台。
  2. 单击目标实例的 ID，进入**基本信息**页面。
  3. 在**私网连接信息**栏或**公网连接信息**栏中，即可查询相应的网络连接地址及端口号。

- 2：要连接实例所用的账号名称。
- 3：连接实例所用账号对应的密码。
- 4：选择要连接的数据库类型。

5. 单击**登录**。

> **说明：** 若您希望浏览器记住该账号的密码，可以先勾选**记住密码**，然后再单击**登录**。

16.5.2 通过客户端连接MySQL实例

本章介绍如何通过MySQL-Front客户端连接RDS实例。

前提条件

已安装MySQL-Front客户端。

背景信息

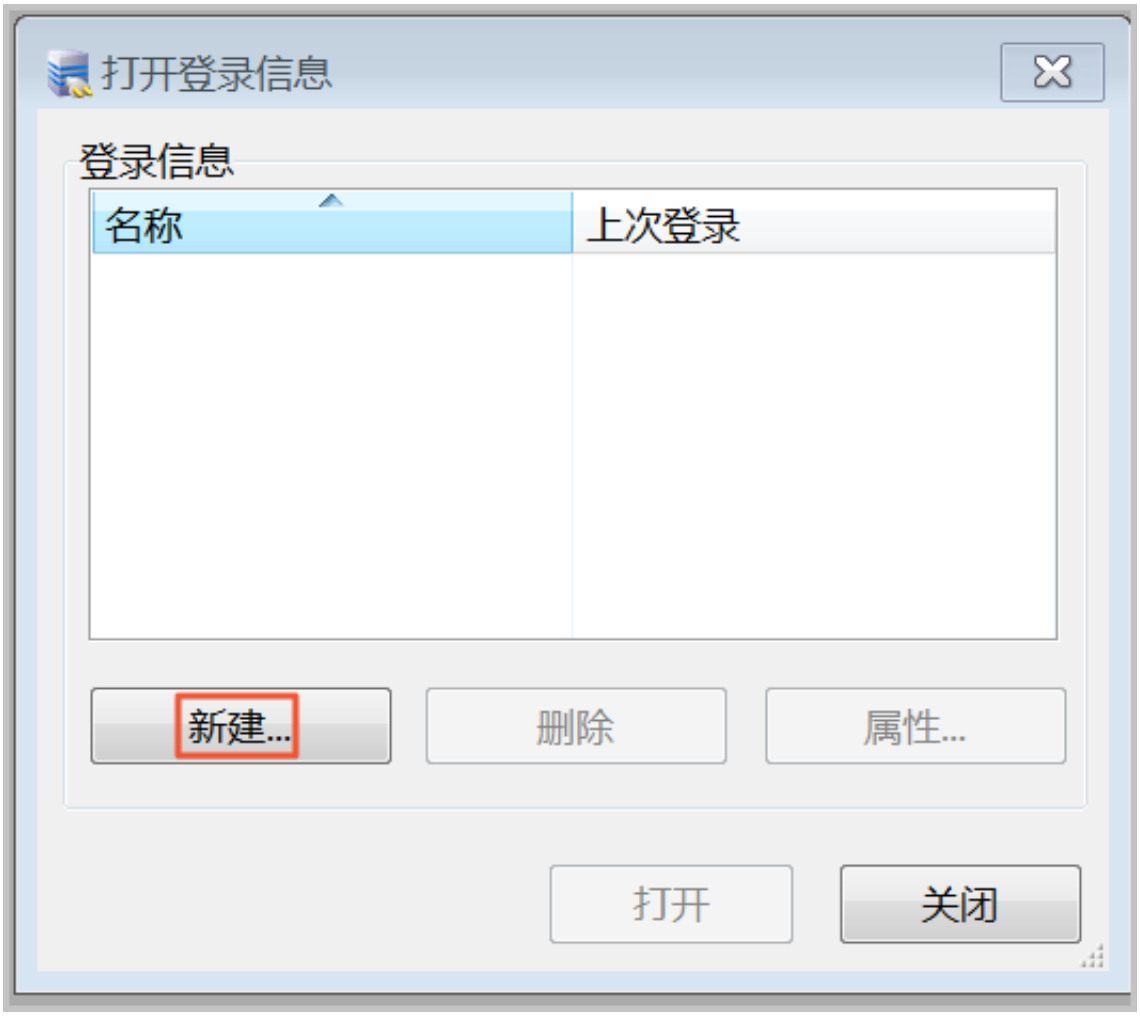
由于RDS提供的关系型数据库服务与原生的数据库服务完全兼容，所以对用户而言，连接数据库的方式也基本类似。本文以MySQL-Front客户端为例介绍RDS实例的连接方法，其它客户端可参见此方法。用客户端连接RDS实例时，请注意选择内外网地址：

- 若您的客户端部署在与要访问的RDS实例在同一地域的ECS上且RDS实例与ECS的网络类型相同时，请使用内网地址。
- 其它情况请使用外网地址。

操作步骤

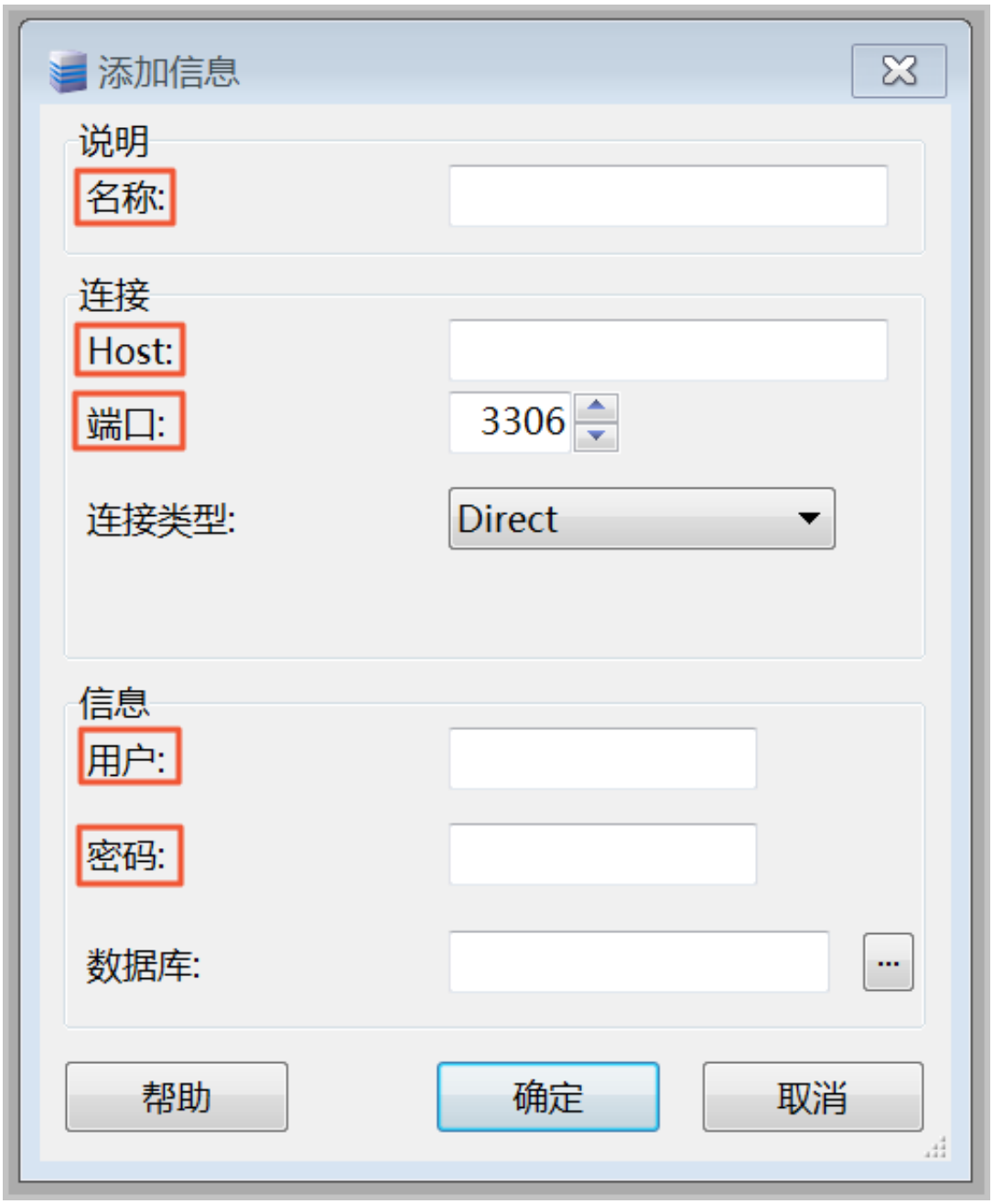
1. 将要访问RDS实例的IP地址加入RDS白名单中。关于如何设置白名单，请参见设置白名单。
2. 启动MySQL-Front客户端。
3. 在**打开登录信息**窗口，单击**新建**，如图 *332:* 新建连接所示。

图 332: 新建连接



4. 输入要连接的RDS实例信息，如图 *333:* 输入连接信息所示。

图 333: 输入连接信息



参数说明：

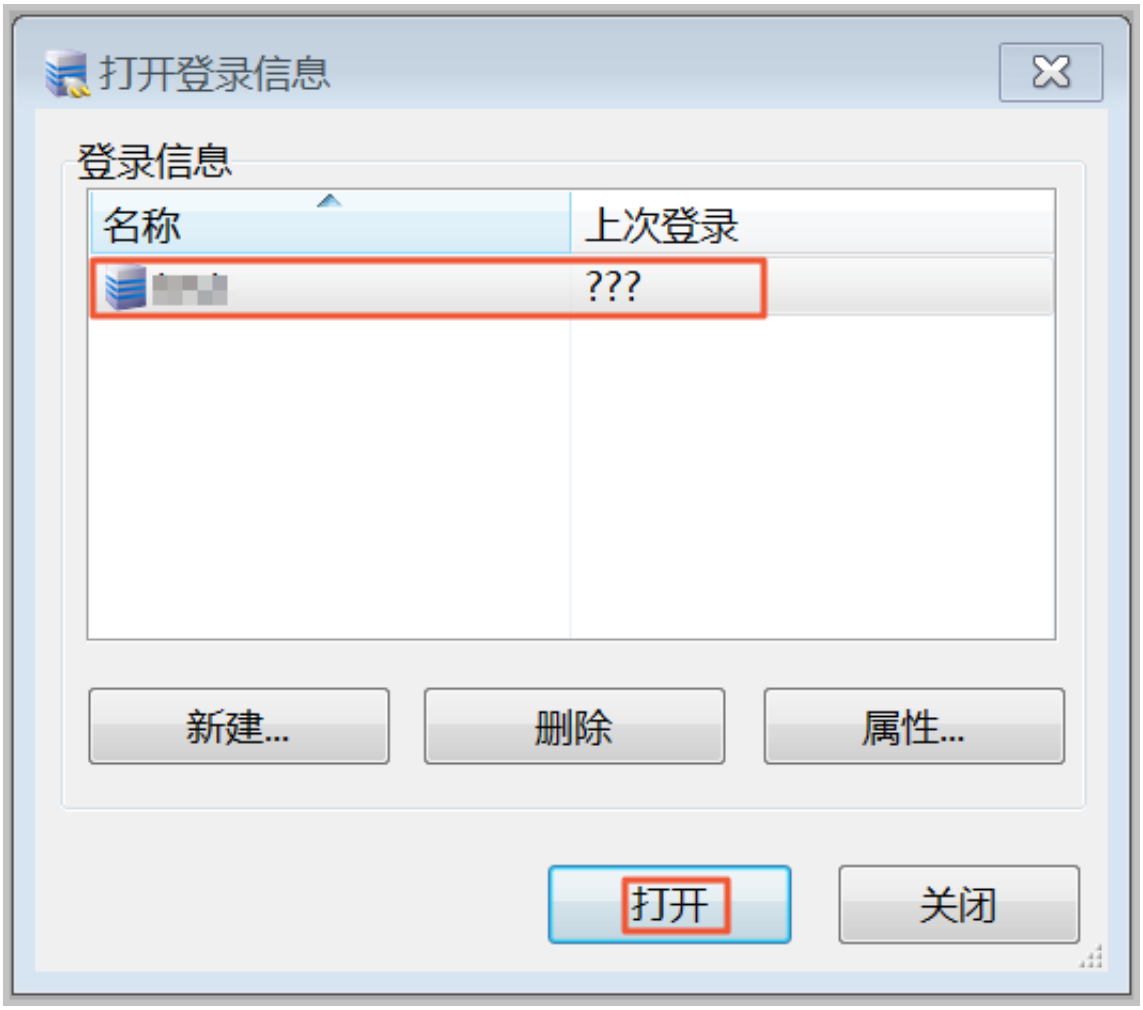
- 名称：MySQL-Front连接数据库的任务名称。若不填，系统会将默认与Host一致。
- Host：若使用内网连接，需输入RDS实例的内网地址。若使用外网连接，需输入RDS实例的外网地址。查看RDS实例的内外网地址及端口信息的步骤如下：

  1. 登录*RDS*控制台。
  2. 单击目标实例的 ID，进入**基本信息**页面。
  3. 在**私网连接信息**栏或**公网连接信息**栏中，即可查询相应的网络连接地址及端口号。

- 端口：若使用内网连接，需输入RDS实例的内网端口。若使用外网连接，需输入RDS实例的外网端口。
- 用户：要访问RDS实例的账号名称。
- 密码：要访问RDS实例的账号所对应的密码。

5. 单击**确定**。
6. 在**打开登录信息**窗口，选中刚才创建的连接，然后单击**打开**，如图 *334: 连接实例*所示。若连接信息无误，即会成功连接实例。

图 334: 连接实例



16.6 只读实例

16.6.1 只读实例简介

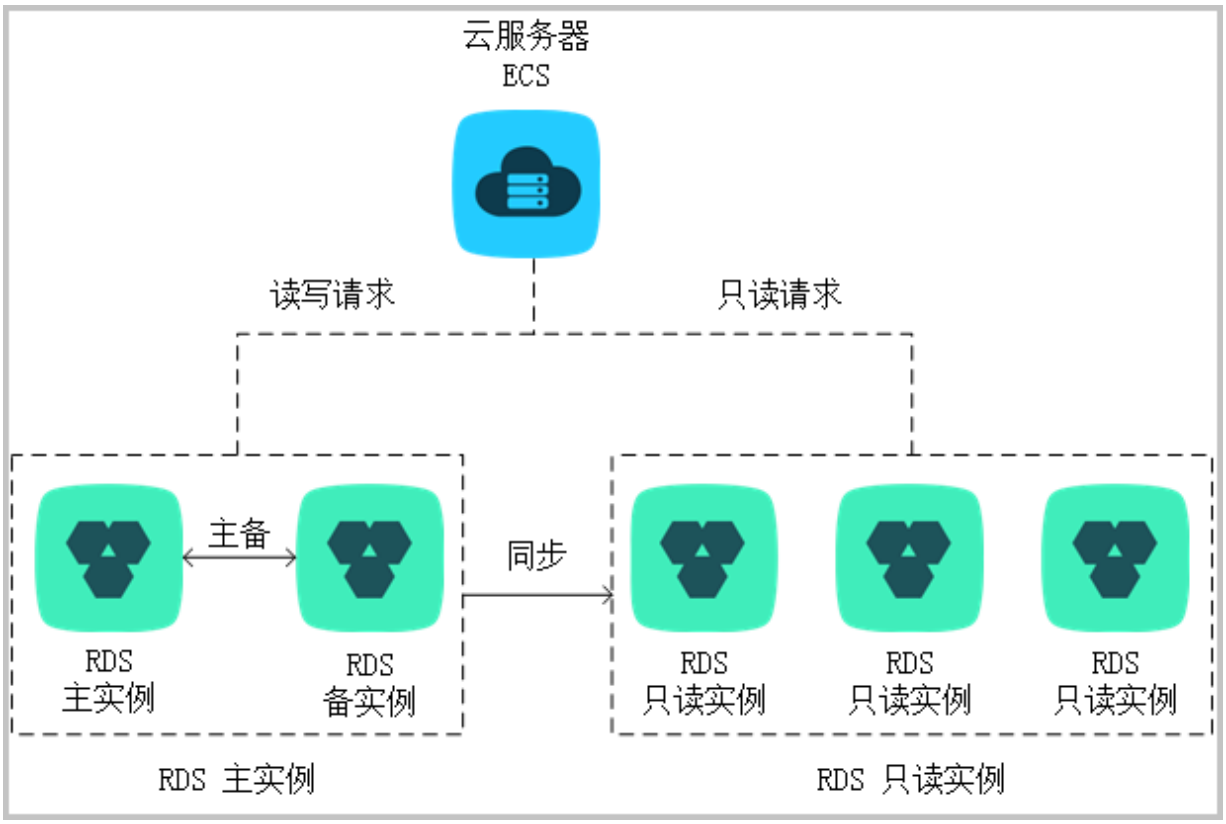
产品简介

目前，只有云数据库MySQL 5.6版的实例支持只读实例。

在对数据库有少量写请求，但有大量读请求的应用场景下，单个实例可能无法抵抗读取压力，甚至对主业务产生影响。为了实现读取能力的弹性扩展，分担数据库压力，您可以在某个地域中创建一个或多个只读实例，利用只读实例满足大量的数据库读取需求，以此增加应用的吞吐量。

只读实例为单个物理节点的架构（没有备节点），采用MySQL的原生复制功能将主实例的更改同步到所有只读实例。只读实例跟主实例在同一地域，但可以在不同的可用区。只读实例拓扑图如所示：

图 335: 只读实例拓扑图



功能特点

只读实例有如下功能特点：

- 规格可以与主实例不一致，并可以随时更改规格（没有时间限制），便于弹性升降级。

- 不需要维护账号与数据库，全部通过主实例同步。
- 独立的白名单配置。
- 提供系统性能监控：

  RDS提供近20个系统性能的监控视图，如磁盘容量、IOPS、连接数、CPU 利用率、网络流量等，用户可以轻松查看实例的负载。
- 提供优化建议：RDS提供多种优化建议，如存储引擎检查、主键检查、大表检查、索引偏多、缺失索引等，用户可以根据优化建议并结合自身的应用来对数据库进行优化。

功能限制

只读实例有如下功能限制：

- 1个主实例最多可以创建5个只读实例。
- 备份设置：不支持备份设置以及临时备份。
- 实例恢复：
  - 不支持通过备份文件或任意时间点创建临时实例，不支持通过备份集覆盖实例。
  - 创建只读实例后，主实例将不支持通过备份集直接覆盖实例来恢复数据。
- 数据迁移：不支持将数据迁移至只读实例。
- 数据库管理：不支持创建和删除数据库。
- 账号管理：不支持创建和删除账号，不支持为账号授权以及修改账号密码功能。

16.6.2 管理只读实例

16.6.2.1 直接通过只读实例进入管理页面

您可以通过实例列表页面直接进入只读实例的管理页面，也可以通过主实例的只读实例列表页面进入只读实例的管理页面。管理方式与常规实例的管理类似，可执行的管理操作请以实际界面为准。本章将介绍如何通过实例列表页面直接进入只读实例的管理页面。

操作步骤

1. 登录*RDS*控制台。
2. 在**RDS实例**列表页面，单击目标只读实例的ID，进入实例的**基本信息**页面，即进入只读实例的管理页面。

   在实例列表中，**实例类型**为**只读实例**的就是只读实例，如图 *336:* 查看只读实例所示。

   图 336: 查看只读实例



16.6.2.2 通过主实例进入只读实例的管理页面

您可以通过实例列表页面直接进入只读实例的管理页面，也可以通过主实例的只读实例列表页面进入只读实例的管理页面。管理方式与常规实例的管理类似，可执行的管理操作请以实际界面为准。本章将介绍如何通过主实例的只读实例列表页面进入只读实例的管理页面。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**性能优化** > **只读实例**，进入**<主实例ID>只读实例列表页面**。
4. 单击目标只读实例的ID，即会进入该只读实例的**基本信息**页面，即进入只读实例的管理页面。

16.7 实例管理

16.7.1 查询详情

您可以查看实例的基本信息、内网连接信息、运行状态、配置信息等详情。本章将介绍如何查询实例详情。

操作步骤

1. 登录*RDS*控制台。
2. 您可以通过如下两种方式进入实例详情页面：
   - 单击目标实例的 ID，进入**基本信息**页面。
   - 在目标实例所对应的操作栏中，单击 > **查询详情**。

16.7.2 重启实例

操作步骤

1. 登录*RDS*控制台。

2. 单击目标实例的 ID，进入**基本信息**页面。
3. 单击实例管理界面右上角的**实例重启**。
4. 在弹出的确认窗口单击**确定**。

16.7.3 设置参数

RDS支持自定义部分实例参数，具体可修改的参数项请参见RDS管理控制台中的 **参数设置**。

背景信息

注意:

- 请按**参数列表**页面规定的**参数范围**设置参数。
- 部分参数修改后需要重启实例，请查看**参数列表**页面中的**是否需要重启**了解，重启前请做好业务安排。

由于RDS提供的云数据库服务与原生的数据库服务完全兼容，所以对用户来说，设置云数据库参数方法和本地数据库参数设置方法基本类似。您可以参照本例，通过RDS管理控制台修改参数，也可以通过API方式，执行相关命令修改参数设置。

关于MySQL 5.6版本数据库的参数说明，请参见官方文档。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的 ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**性能优化** > **参数设置**，进入**参数列表**页面。
4. 在目标参数对应的操作栏中，单击单击 > **修改**。
5. 在弹出的窗口中输入目标值，然后单击**确定**。

16.7.4 修改实例名称

为便于管理，您可以修改实例名称。本章将介绍如何修改实例名称。

背景信息

在实例列表中，**实例ID/实例名称**栏中，上面是实例的ID，下面是实例名称，如图 *337: 实例ID/实例名称*所示。实例ID不能修改，实例名称可以看做是实例的备注名。

图 337: 实例ID/实例名称



操作步骤

1. 登录*RDS*控制台。
2. 在目标实例所对应的操作栏中，单击 > **修改实例名称**。
3. 在**实例名称**栏中输入所需的备注名。
4. 单击**确定**。

16.7.5 典型参数设置

16.7.5.1 MySQL实例可修改参数说明

MySQL目前支持修改的参数如表 *40: MySQL实例可修改参数列表*所示，更多内容还请参考MySQL官方文档：*https://dev.mysql.com/doc/*。

表 40: MySQL实例可修改参数列表

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
auto_increment_increment	1	1	否	[1-65535]	auto_increment_increment and auto_increment_offset are intended for use with master-to-master replication, and can be used to control the operation of AUTO_INCREMENT columns. Both variables have global and session values, and each can assume an integer value between 1 and 65,535 inclusive. Setting the value of either of these two variables to 0 causes its value to be set to 1 instead. Attempting to set the value of either of these

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
					two variables to an integer greater than 65,535 or less than 0 causes its value to be set to 65,535 instead. Attempting to set the value of auto_increment_increment or auto_increment_offset to a noninteger value produces an error, and the actual value of the variable remains unchanged.
auto_increment_offset	1	1	否	[1-65535]	determines the starting point for the AUTO_INCREMENT column value.
back_log	3000	3000	是	[0-65535]	The number of outstanding connection requests that MySQL can have.
binlog_cache_size	2097152	128KB	否	[4096-16777216]	The size of the cache to hold changes to the binary log during a transaction.
binlog_checksum	CRC32	CRC32	是	[CRC32|NONE]	The master to write a checksum for each event in the binary log.
binlog_row_image	full	full	否	[full|minimal]	Binlog save every column or actually required column in binlog images.
binlog_stmt_cache_size	32768	32768	否	[4096-16777216]	The size of the statement cache for updates to non-transactional engines for the binary log.
character_set_server	utf8	utf8	是	[utf8|latin1|gbk|utf8mb4]	The server's default character set.
concurrent_insert	1	1	否	[0|1|2]	NEVER-Disables concurrent inserts; AUTO-(Default) Enables concurrent insert for MyISAM tables that do not have holes; ALWAYS-

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
					Enables concurrent inserts for all MyISAM tables, even those that have holes. For a table with a hole, new rows are inserted at the end of the table if it is in use by another thread. Otherwise, MySQL acquires a normal write lock and inserts the row into the hole.
connect_timeout	10	10	否	[1-3600]	The number of seconds that the mysqld server waits for a connect packet before responding with Bad handshake. The default value is 10 seconds as of MySQL 5.1.23 and 5 seconds before that.Increasing the connect_timeout value might help if clients frequently encounter errors of the form Lost connection to MySQL server at 'XXX', system error: errno.
default_storage_engine	InnoDB	TokuDB	是	[InnoDB|TokuDB|innodb|tokudb]	The default storage engine for new tables.
default_time_zone	SYSTEM	SYSTEM	是	[SYSTEM|-12:00|-11:00|-10:00|-9:00|-8:00|-7:00|-6:00|-5:00|-4:0 +0:00|+1:00|+2:00| +3:00|+4:00|+5:00| +5:30|+6:00|+6:30| +7:00|+8:00|+9:00| +10:00|+11:00| +12:00|+13:00]	The default time zone for the database.
default_week_format	0	0	否	[0-7]	The default mode value to use for the WEEK() function.
delayed_insert_limit	100	100	否	[1-4294967295]	After inserting delayed_insert_limit delayed rows, the INSERT DELAYED handler thread checks whether there are any

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
					SELECT statements pending. If so, it permits them to execute before continuing to insert delayed rows.
delayed_insert_timeout	300	300	否	[1-3600]	How many seconds an INSERT DELAYED handler thread should wait for INSERT statements before terminating.
delayed_queue_size	1000	1000	否	[1-4294967295]	This is a per-table limit on the number of rows to queue when handling INSERT DELAYED statements. If the queue becomes full, any client that issues an INSERT DELAYED statement waits until there is room in the queue again.
delay_key_write	ON	ON	否	[ON|OFF|ALL]	This option applies only to MyISAM tables. It can have one of the following values to affect handling of the DELAY_KEY_WRITE table option that can be used in CREATE TABLE statements.
div_precision_increment	4	4	否	[0-30]	This variable indicates the number of digits by which to increase the scale of the result of division operations performed with the / operator. The default value is 4. The minimum and maximum values are 0 and 30, respectively. The following example illustrates the effect of increasing the default value.

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
eq_range_index_dive_limit	10	10	否	[1-200]	This variable indicates the number of equality ranges in an equality comparison condition when the optimizer should switch from using index dives to index statistics in estimating the number of qualifying rows.
explicit_defaults_for_timestamp	false	false	是	true|false	As indicated by the warning, to turn off the nonstandard behaviors, enable the explicit_defaults_for_timestamp system variable at server startup.
ft_min_word_len	4	4	是	[1-3600]	The minimum length of the word to be included in a FULLTEXT index.
ft_query_expansion_limit	20	20	是	[0-1000]	The number of top matches to use for full-text searches performed using WITH QUERY EXPANSION.
group_concat_max_len	1024	1024	否	[4-18446744073709547552]	maximum permitted result length in bytes for the GROUP_CONCAT() function. The default is 1024, Unit:Byte.
innodb_adaptive_hash_index	ON	ON	否	[ON|OFF]	It may be desirable, depending on your workload, to dynamically enable or disable adaptive hash indexing to improve query performance. The size in bytes of a memory pool InnoDB uses to store data dictionary information and other internal data structures. The more tables you have in your application, the more memory

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
					you allocate here. If InnoDB runs out of memory in this pool, it starts to allocate memory from the operating system and writes warning messages to the MySQL error log. The default value is 8MB.
innodb_additional_mem_pool_size	2097152	2097152	是	[2097152-4048576000]	The locking mode to use for generating auto-increment values. The permissible values are 0, 1, or 2.
innodb_autoinc_lock_mode	1	1	是	[0|1|2]	The number of threads that can enter InnoDB concurrently is determined by the innodb_thread_concurrency variable.
innodb_concurrency_tickets	5000	5000	否	[1-4294967295]	The number of threads that can enter InnoDB concurrently is determined by the innodb_thread_concurrency variable.
innodb_ft_max_token_size	84	84	是	[10-84]	Maximum length of words that are stored in an InnoDB FULLTEXT index.
innodb_ft_min_token_size	3	3	是	[0-16]	Minimum length of words that are stored in an InnoDB FULLTEXT index.
innodb_large_prefix	OFF	OFF	否	[ON|OFF]	Enable this option to allow index key prefixes longer than 767 bytes (up to 3072 bytes), for InnoDB tables that use the DYNAMIC and COMPRESSED row formats.
innodb_lock_wait_timeout	50	50	否	[1-1073741824]	The timeout in seconds an InnoDB transaction may wait for a row lock before giving

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
					up. The default value is 50 seconds. Unit: Second.
innodb_max_dirty_pages_pct	75	75	否	[50-90]	This is an integer in the range from 0 to 100. The default value is 90 for the built-in InnoDB, 75 for InnoDB Plugin. The main thread in InnoDB tries to write pages from the buffer pool so that the percentage of dirty (not yet written) pages will not exceed this value.
innodb_old_blocks_pct	37	37	否	[5-95]	(InnoDB Plugin only) Specifies the approximate percentage of the InnoDB buffer pool used for the old block sublist. The range of values is 5 to 95. The default value is 37 (that is, 3/8 of the pool).
innodb_old_blocks_time	1000	1000	否	[0-1024]	(InnoDB Plugin only) Specifies how long in milliseconds (ms) a block inserted into the old sublist must stay there after its first access before it can be moved to the new sublist. The default value is 0: A block inserted into the old sublist moves immediately to the new sublist the first time it is accessed, no matter how soon after insertion the access occurs. If the value is greater than 0, blocks remain in the old sublist until an access occurs at least that many ms after the first access. Unit: ms.

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
innodb_online_alter_log_max_size	134217728	134217728	否	[134217728-214748364	Specifies an upper limit on the size of the temporary log files used during online DDL operations for InnoDB tables.
innodb_open_files	3000	3000	是	[1-8192]	This variable is relevant only if you use multiple InnoDB tablespaces. It specifies the maximum number of .ibd files that MySQL can keep open at one time. The minimum value is 10. The default value is 300.
innodb_print_all_deadlocks	OFF	OFF	否	[OFF|ON]	When this option is enabled, information about all deadlocks in InnoDB user transactions is recorded in the mysqld error log.
innodb_purge_batch_size	300	300	是	[1-5000]	The granularity of changes, expressed in units of redo log records, that trigger a purge operation, flushing the changed buffer pool blocks to disk.
innodb_purge_threads	1	1	是	[1-32]	The number of background threads devoted to the InnoDB purge operation.
innodb_read_ahead_threshold	56	56	否	[0-64]	(InnoDB Plugin only) Controls the sensitivity of linear read-ahead that InnoDB uses to prefetch pages into the buffer pool. If InnoDB reads at least innodb_read_ahead_threshold pages sequentially from an extent (64 pages), it initiates an asynchronous read for the entire following extent.
innodb_read_io_threads	4	4	是	[1-64]	(InnoDB Plugin only) The number of I/O threads for read

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
					operations in InnoDB. The default value is 4.
innodb_rollback_on_timeout	OFF	OFF	是	[OFF|ON]	InnoDB rolls back only the last statement on a transaction timeout by default. If --innodb_rollback_on_timeout is specified, a transaction timeout causes InnoDB to abort and roll back the entire transaction (the same behavior as in MySQL 4.1). This variable was added in MySQL 5.1.15
innodb_stats_method	nulls_equal	nulls_equal	否	[nulls_equal| nulls_unequal| nulls_ignored]	How the server treats NULL values when collecting statistics about the distribution of index values for InnoDB tables. This variable has three possible values, nulls_equal, nulls_unequal, and nulls_ignored. For nulls_equal, all NULL index values are considered equal and form a single value group that has a size equal to the number of NULL values. For nulls_unequal, NULL values are considered unequal, and each NULL forms a distinct value group of size 1. For nulls_ignored, NULL values are ignored.
innodb_stats_on_metadata	OFF	OFF	否	[ON|OFF]	When this variable is enabled (which is the default, as before the variable was created), InnoDB updates statistics during metadata statements such as SHOW TABLE

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
					STATUS or SHOW INDEX, or when accessing the INFORMATION_SCHEMA tables TABLES or STATISTICS. (These updates are similar to what happens for ANALYZE TABLE.) When disabled, InnoDB does not update statistics during these operations. Disabling this variable can improve access speed for schemas that have a large number of tables or indexes. It can also improve the stability of execution plans for queries that involve InnoDB tables.
innodb_stats_sample_pages	8	8	否	[1-4294967296]	(InnoDB Plugin only) The number of index pages to sample for index distribution statistics such as are calculated by ANALYZE TABLE. The default value is 8.
innodb_strict_mode	OFF	OFF	否	[ON|OFF]	(InnoDB Plugin only) Whether InnoDB returns errors rather than warnings for certain conditions. This is analogous to strict SQL mode. The default value is OFF. See InnoDB Strict Mode for a list of the conditions that are affected.
innodb_table_locks	ON	ON	否	[ON|OFF]	If autocommit = 0, InnoDB honors LOCK TABLES; MySQL does not return from LOCK TABLES ... WRITE until all other threads have released all their locks to the table. The default value

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
					of innodb_table_locks is 1, which means that LOCK TABLES causes InnoDB to lock a table internally if autocommit = 0.
innodb_thread_concurrency	0	0	否	[0-128]	InnoDB tries to keep the number of operating system threads concurrently inside InnoDB less than or equal to the limit given by this variable. Once the number of threads reaches this limit, additional threads are placed into a wait state within a FIFO queue for execution.
innodb_thread_sleep_delay	10000	10000	否	[1-3600000]	How long InnoDB threads sleep before joining the InnoDB queue, in microseconds. The default value is 10,000. A value of 0 disables sleep.Unit:ms
innodb_write_io_threads	4	4	是	[1-64]	(InnoDB Plugin only) The number of I/O threads for write operations in InnoDB. The default value is 4.
interactive_timeout	7200	7200	否	[10-86400]	The number of seconds the server waits for activity on an interactive connection before closing it. An interactive client is defined as a client that uses the CLIENT_INTERACTIVE option to mysql_real_connect(). Unit:second.
key_cache_age_threshold	300	300	否	[100-4294967295]	This value controls the demotion of buffers from the hot sublist of a key cache to the warm sublist. Lower values cause demotion

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
					to happen more quickly. The minimum value is 100. The default value is 300.Unit:Second.
key_cache_block_size	1024	1024	否	[512-16384]	The size in bytes of blocks in the key cache. The default value is 1024.Unit:Byte.
key_cache_division_limit	100	100	否	[1-100]	The division point between the hot and warm sublists of the key cache buffer list. The value is the percentage of the buffer list to use for the warm sublist. Permissible values range from 1 to 100. The default value is 100.
log_queries_not_using_indexes	OFF	OFF	否	[ON|OFF]	If a query takes longer than this many seconds, the server increments the Slow_queries status variable. If the slow query log is enabled, the query is logged to the slow query log file; Unit: Second.
long_query_time	1	1	否	[0.03-10]	If a query takes longer than this many seconds, the server increments the Slow_queries status variable. If the slow query log is enabled, the query is logged to the slow query log file;Unit:Second.
loose_max_statement_time	0	0	否	[0-4294967295]	statement be interrupted if the executing time exceeds this value.
loose_rds_indexstat	OFF	OFF	否	[ON|OFF]	If ON, start to collect index information.
loose_rds_max_tmp_disk_space	10737418240	10737418240	否	[10737418240-1073741824240]	Maximum temp disk space.
loose_rds_tablestat	OFF	OFF	否	[ON|OFF]	RDS table statistics.

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
loose_rds_threads_running_high_watermark	50000	50000	否	[0-50000]	Max concurrency allowed for SELECT.
loose_tokudb_buffer_pool_ratio	0	0	是	[0-100]	TokuDB buffer pool size ratio.
low_priority_updates	0	0	否	[0|1]	If set to 1, all INSERT, UPDATE, DELETE, and LOCK TABLE WRITE statements wait until there is no pending SELECT or LOCK TABLE READ on the affected table. This affects only storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE). This variable previously was named sql_low_priority_updates.
max_allowed_packet	1073741824	1024M	否	[16384-1073741824]	The maximum size of one packet or any generated/ intermediate string. Unit: Byte.
max_connect_errors	100	100	否	[1-4294967295]	If more than this many successive connection requests from a host are interrupted without a successful connection, the server blocks that host from further connections. You can unblock blocked hosts by flushing the host cache.
max_length_for_sort_data	1024	1024	否	[0-838860]	The cutoff on the size of index values that determines which filesort algorithm to use.
max_prepared_stmt_count	16382	16382	否	[0-1048576]	This variable limits the total number of prepared statements in the server.

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
max_write_lock_count	102400	102400	否	[1-102400]	After this many write locks, permit some pending read lock requests to be processed in between.
myisam_sort_buffer_size	262144	262144	否	[262144-16777216]	The size of the buffer that is allocated when sorting MyISAM indexes during a REPAIR TABLE or when creating indexes with CREATE INDEX or ALTER TABLE.
net_read_timeout	30	30	否	[1-31536000]	The number of seconds to wait for more data from a connection before aborting the read.
net_retry_count	10	10	否	[1-4294967295]	If a read or write on a communication port is interrupted, retry this many times before giving up.
net_write_timeout	60	60	否	[1-31536000]	The number of seconds to wait for a block to be written to a connection before aborting the write.
open_files_limit	65535	65535	是	[4000-65535]	The number of files that the operating system permits mysqld to open. The value of this variable at runtime is the real value permitted by the system and might be different from the value you specify at server startup. The value is 0 on systems where MySQL cannot change the number of open files.
performance_schema	OFF	OFF	是	[ON|OFF]	Enable performance_schema or not.

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
query_alloc_block_size	8192	8192	否	[1024-16384]	The allocation size of memory blocks that are allocated for objects created during statement parsing and execution. Unit: Byte.
query_cache_limit	1048576	1048576	否	[1-1048576]	Do not cache results that are larger than this number of bytes. The default value is 1MB.
query_cache_size	3145728	3145728	否	[0-104857600]	The amount of memory allocated for caching query results. The default value is 0, which disables the query cache. The permissible values are multiples of 1024; other values are rounded down to the nearest multiple. Unit: Byte.
query_cache_type	0	0	是	[0|1|2]	Set the query cache type. Setting the GLOBAL value sets the type for all clients that connect thereafter. Individual clients can set the SESSION value to affect their own use of the query cache. Possible values are shown in the following table. • 0: Do not cache results in or retrieve results from the query cache. Note that this does not deallocate the query cache buffer. To do that, you should set query_cache_size to 0. • 1: Cache all cacheable query results except for those that begin with SELECT SQL_NO_CACHE.

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
					• 2: Cache results.
query_cache_wlock_invalidate	OFF	OFF	否	[ON|OFF]	Normally, when one client acquires a WRITE lock on a MyISAM table, other clients are not blocked from issuing statements that read from the table if the query results are present in the query cache. Setting this variable to 1 causes acquisition of a WRITE lock for a table to invalidate any queries in the query cache that refer to the table. This forces other clients that attempt to access the table to wait while the lock is in effect.
query_prealloc_size	8192	8192	否	[8192-1048576]	The size of the persistent buffer used for statement parsing and execution. This buffer is not freed between statements. If you are running complex queries, a larger query_prealloc_size value might be helpful in improving performance, because it can reduce the need for the server to perform memory allocation during query execution operations. Unit: Byte.
rds_reset_all_filter	0	0	否	[0|1]	rds_reset_all_filter=1 ,means reset the rule of filter.
slow_launch_time	2	2	否	[1-1024]	If creating a thread takes longer than this many seconds, the server increments the Slow_launch_threads status variable.

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
sql_mode	\s	\s	否	(支持空格及|REAL_AS_FLOAT|PIPES_AS_CONCAT|ANSI_QUOTES|IGNORE_SPACE|ONLY_FULL_GROUP_BY|NO_UNSIGNED_SUBTRACTION|NO_DIR_IN_CREATE|POSTGRESQL|ORACLE|MSSQL|DB2|MAXDB|NO_KEY_OPTIONS|NO_TABLE_OPTIONS|NO_FIELD_OPTIONS|MYSQL323|MYSQL40|ANSI|NO_AUTO_VALUE_ON_ZERO|NO_BACKSLASH_ESCAPES|STRICT_TRANS_TABLES|STRICT_ALL_TABLES|NO_ZERO_IN_DATE|NO_ZERO_DATE|ALLOW_INVALID_DATES|ERROR_FOR_DIVISION_BY_ZERO|TRADITIONAL|HIGH_NOT_PRECEDENCE|NO_ENGINE_SUBSTITUTION|PAD_CHAR_TO_FULL_LENGTH)(,REAL_AS_FLOAT|,PIPES_AS_CONCAT|,ANSI_QUOTES|,IGNO_UNSIGNED_SUBTRACTION|,NO_DIR_IN_CREATE|,POSTGRESQL|,ORACLE|,MSSQL|NO_AUTO_VALUE_ON_ZERO|,NO_BACKSLASH_ESCAPES|,STRICT_TRANS_TABLES|,STRICT_ALL_TABLES|,NO_ZERO	Modes define what SQL syntax MySQL should support and what kind of data validation checks it should perform.

数名	参数默认值	运行参数值	是否重启	可修改参数值	参数描述
				ERROR_FOR_DIVISION_BY_ZERO|,TRADITIONAL|,HIGH_NOT_PRECEDENCE|,NO_ENGINE_SUBSTITUTION|,PAD_CHAR_TO_FULL_LENGTH)*	
table_definition_cache	512	512	否	[400-80480]	The number of table definitions (from .frm files) that can be stored in the definition cache. If you use a large number of tables, you can create a large table definition cache to speed up opening of tables. The table definition cache takes less space and does not use file descriptors, unlike the normal table cache. The minimum and default values are both 400.
table_open_cache	2000	2000	否	[1-524288]	The stack size of each thread.
thread_stack	262144	262144	是	[131072-18446744073709551615]	The stack size of each thread.
tmp_table_size	2097152	2097152	否	[262144-67108864]	The maximum size of internal in-memory temporary tables.
transaction_isolation	READ-COMMITTED	READ-COMMITTED	是	[READ-COMMITTED|REPEATABLE-READ]	The default transaction isolation level.
wait_timeout	86400	86400	否	[60-259200]	The number of seconds the server waits for activity on a noninteractive connection before closing it.

16.7.5.2 MySQL实例参数调优最佳实践

16.7.5.2.1 前言

RDS用户经常会问如何调优MySQL的参数，本章节将通过最佳实践来介绍MySQL实例支持修改的参数和不支持修改的参数，以及如何将可修改的参数设置到最佳值以提升实例性能。

16.7.5.2.2 MySQL实例不支持修改的参数

不同规格的RDS实例所提供的最大连接数以及内存是不同的，所以与产品规格相关的限制参数，即连接数和内存，不支持修改。若内存或者连接数出现了瓶颈，需通过如下方法解决：

- 内存瓶颈：实例会出现OOM（Out of Memory），然后导致主备发生切换。
- 连接数瓶颈：应用不能新建立连接到数据库，则需要进行应用优化、慢SQL优化或者通过弹性升级实例规格来解决。

另外，为保障主备实例的数据安全，一些涉及数据安全的参数暂不支持修改，例如innodb_flush_log_at_trx_commit、sync_binlog、gtid_mode、semi_sync、binlog_format。

16.7.5.2.3 MySQL实例支持修改的参数

除*MySQL实例不支持修改的参数*中介绍的不支持修改的参数外，RDS绝大部分参数都已经由DBA团队和源码团队优化过，您不需要过多调整线上的参数就可以把数据库比较好的运行起来。关于MySQL实例支持修改的参数详情，请参见文档*MySQL实例可修改参数说明*。但这些参数只适合大多数的应用场景，个别特殊的场景仍需要个别对待。例如：

- 若您使用了tokudb引擎，就需要调整tokudb引擎能使用的内存比例（tokudb_buffer_pool_ratio）。
- 若您的应用本身需要很大的一个锁超时时间，则需要调整innodb_lock_wait_timeout参数的大小以适应应用。

16.7.5.2.4 如何设置参数

本节将介绍可以通过RDS控制台修改的一些比较重要的参数。这些参数如果设置不当，则可能会出现实例性能问题或应用报错。

open_files_limit

作用：该参数用于控制MySQL实例能够同时打开使用的文件句柄数目。

原因：当数据库中的表打开越来越多后，会消耗分配给每个实例的文件句柄数目。RDS在起初初始化实例的时候将open_files_limit的值设置为8192，当打开的表数目超过该参数则会导致所有的数据库请求报错误。



说明: MyISAM 引擎表在被访问的时候需要消耗文件描述符，InnoDB引擎会自己管理已经打开的表，相关参数为table_open_cache。

现象：如果该参数值设置过小可导致应用报错，如下所示。

```
[ERROR] /mysqld: Can't open file: './mysql/user.frm' (errno: 24 -Too many open files);
```

建议：提高open_files_limit的值，RDS目前可以支持的最大值为65535，同时建议替换MyISAM存储引擎为InnoDB引擎。

back_log

作用：MySQL每处理一个连接请求的时候都会创建一个新线程与之对应。在主线程创建新线程期间，如果前端应用有大量的短连接请求到达数据库，MySQL会限制新连接在此时进入请求队列，由参数back_log控制。如果等待的连接数量超过back_log，则将不会接受新的连接请求，所以如果需要MySQL能够处理大量的短连接，需要提高此参数的大小。

现象：如果参数过小，则可能会导致应用报错，如下所示。

```
SQLSTATE[HY000] [2002] Connection timed out;
```

建议：提高back_log的值。较早时期，RDS在初始化时，该参数的默认值是50，现在初始化值已经调至3000。



注意: 调整该参数值后，需要重启实例。

innodb_autoinc_lock_mode

作用：在MySQL 5.1.22后，InnoDB为了解决自增主键锁表的问题，引入了参数innodb_autoinc_lock_mode，用于控制自增主键的锁机制。该参数可以设置的值为0/1/2，RDS默认的参数值为1，表示InnoDB使用轻量级别的mutex锁来获取自增锁，替代最原始的表级锁。但是在load data（包括：INSERT ... SELECT、REPLACE ... SELECT）场景下会使用自增表锁，这样可能会导致应用在并发导入数据时出现死锁。

现象：如果应用并发使用load data（包括：INSERT ... SELECT、REPLACE ... SELECT）导入数据的时候出现死锁，如下所示。

```
RECORD LOCKS space id xx page no xx n bits xx index PRIMARY of table xx.xx trx id xxx lock_mode X insert intention waiting. TABLE LOCK table xxx.xxx trx id xxxx lock mode AUTO-INC waiting；
```

建议：建议将innodb_autoinc_lock_mode的值改为2，表示所有情况插入都使用轻量级别的mutex锁（只针对row模式），这样就可以避免auto_inc的死锁，同时在INSERT ... SELECT的场景下会提升很大的性能。



说明: 若将该参数值设为2，binlog的格式需要设置为row。

query_cache_size

作用：该参数用于控制MySQL query cache的内存大小。如果MySQL开启query cache，在执行每一个query时会先锁住query cache，然后判断是否存在于query cache中，如果存在直接返回结果，如果不存在，则再进行引擎查询等操作。同时，insert、update和delete这样的操作都会将query cahce失效掉，这种失效还包括结构或者索引的任何变化。但cache失效的维护代价较高，会给MySQL带来较大的压力。所以，当数据库不需要频繁更新时，query cache有助于提升实例性能。但是如果反过来，写入非常频繁，并集中在某几张表上时，query cache lock的锁机制会造成很频繁的锁冲突，对于这一张表的写和读会互相等待query cache lock解锁，导致select的查询效率下降。

现象：数据库中有大量的连接状态为checking query cache for query、Waiting for query cache lock、storing result in query cache。

建议：RDS默认关闭query cache功能，如果您的实例打开了query cache，当出现上述情况后可以关闭query cache。当然，有些情况也可以打开query cache，例如巧用query cache解决数据库性能问题。

net_write_timeout

作用：等待将一个block发送给客户端的超时时间。

现象：若该参数设置过小，可能导致客户端报错，如下所示。

```
the last packet successfully received from the server was milliseconds ago，the last packet sent successfully to the server was milliseconds ago.
```

建议：该参数在RDS中的默认值为60S。一般在网络条件比较差或者客户端处理每个block耗时比较长时，由于net_write_timeout设置过小导致的连接中断很容易发生，建议增加该参数的大小。

tmp_table_size

作用：该参数用于决定内部内存临时表的最大值，每个线程都要分配（实际起限制作用的是tmp_table_size和max_heap_table_size的最小值）。如果内存临时表超出了限制，MySQL就会自动把它转化为基于磁盘的MyISAM表。优化查询语句时，要避免使用临时表，如果实在避免不了的话，要保证这些临时表是存在内存中的。

现象：如果复杂的SQL语句中包含了group by/distinct等不能通过索引进行优化而使用了临时表，则会导致SQL执行时间加长。

建议：如果应用中有很多group by/distinct等语句，同时数据库有足够的内存，可以增大tmp_table_size和max_heap_table_size的值，以此来提升查询性能。

16.7.5.2.5 MySQL新增参数

oose_rds_max_tmp_disk_space

作用：用于控制MySQL能够使用的临时文件的大小，RDS初始默认值是10G。

现象：如果临时文件超出此大小，则会导致应用报错，如下所示。

```
The table '/home/mysql/dataxxx/tmp/#sql_2db3_1' is full.
```

建议：需要先分析一下导致临时文件增加的SQL语句是否能够通过索引或者其他方式进行优化。其次，如果确定实例的空间足够大，则可以提升此参数的值，以保证SQL能够正常执行。

注意: 调整该参数值后，需要重启实例。

loose_tokudb_buffer_pool_ratio

作用：用于控制TokuDB引擎能够使用的buffer内存大小。例如，将innodb_buffer_pool_size设置为1000M，将tokudb_buffer_pool_ratio设置为50（代表50%），则TokuDB引擎的表能够使用的buffer内存大小为500M。

建议：该参数在RDS中默认值为0。如果RDS中使用TokuDB引擎，则建议调大该参数，以此来提升TokuDB引擎表的访问性能。

注意: 调整该参数值后，需要重启实例。

loose_max_statement_time

作用：用于控制查询在MySQL的最长执行时间。

现象：如果超过该参数设置的时间，查询将会自动失败，默认是不限制，如下所示。

```
ERROR 3006 (HY000): Query execution was interrupted, max_statement_time exceeded
```

建议：如果用户希望控制数据库中SQL的执行时间，则可以开启该参数，单位是毫秒。

loose_rds_threads_running_high_watermark

作用：用于控制MySQL并发的查询数目。例如，将rds_threads_running_high_watermark的值设置为100，则允许MySQL同时进行的并发查询为100个，超过参数值的查询将会被拒绝掉。该参数与rds_threads_running_ctl_mode配合使用（默认值为select）。

建议：该参数常常在秒杀或者大并发的场景下使用，对数据库具有较好的保护作用。

16.8 账号管理

16.8.1 创建账号

在使用数据库之前，需要在RDS实例中创建账号。在数据库迁移之前，需要在RDS实例和本地数据库中创建一致的账号。本章将介绍如何在RDS实例中创建账号。

背景信息

注意:

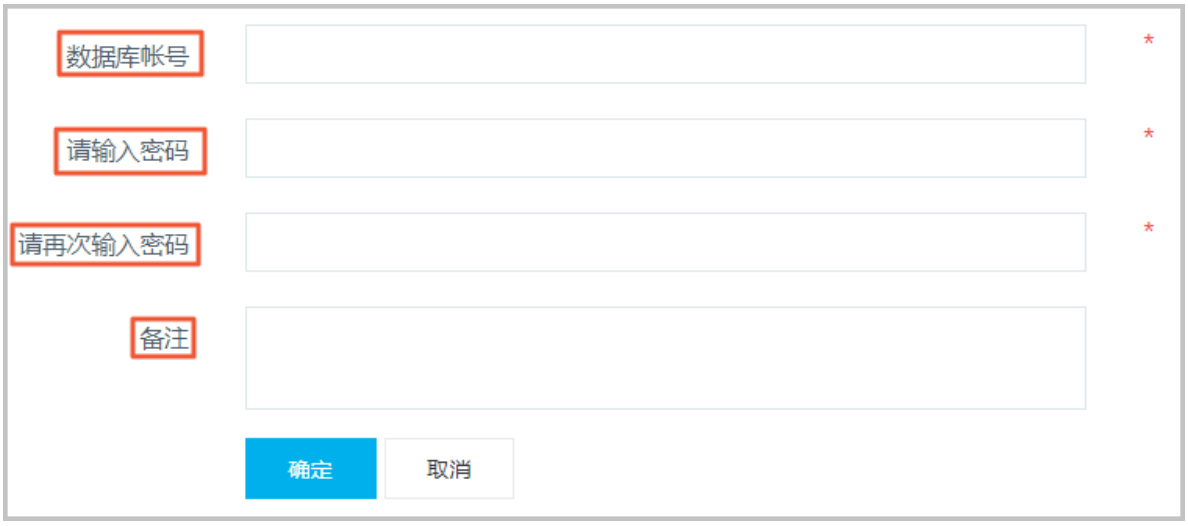
- 同一实例下的数据库共享该实例下的所有资源，每个MySQL 5.6版本的实例最多可以创建500个账号。
- 如果您要迁移本地数据库到RDS，请在RDS实例中创建与本地数据库一致的迁移账号。
- 分配数据库账号权限时，请按最小权限原则和业务角色创建账号，并合理分配只读和读写权限。必要时可以把数据库账号和数据库拆分成更小粒度，使每个数据库账号只能访问其业务之内的数据。如果不需要数据库写入操作，请分配只读权限。
- 为保障数据库的安全，请将数据库账号的密码设置为强密码，并定期更换。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的 ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**数据库管理** > **账号列表**，进入**账号列表**页面。
4. 单击**添加账号**。

5. 输入要创建的账号信息，如图 *338: 创建账号*所示。

图 338: 创建账号



参数说明：

- **数据库账号**：由 2~16 个字符的小写字母，数字或下划线组成、开头需为字母，结尾需为字母或数字，如 *user4example*。
- **请输入密码**：该账号对应的密码，由 6~32 个字符的字母、数字、中划线或下划线组成，如 *password4example*。
- **请再次输入密码**：输入与密码一致的字段，如 *password4example*，以确保密码正确输入。
- **备注**：可以备注该账号的相关信息，便于后续账号管理，最多支持 256 个英文字符。

6. 单击**确定**。

16.8.2 重置密码

在使用RDS 过程中，如果忘记数据库账号密码，可以通过RDS控制台重新设置密码。

背景信息

注意: 为了数据安全，建议您定期更换密码。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的 ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**数据库管理** > **账号列表**，进入**账号列表**页面。

4. 在目标账号所对应的操作栏中，单击 > **重置密码**。
5. 在 **重置密码**页面输入新密码并确认后，单击 **确定**。

> **说明:** 密码由 6个~32 个字符的字母、数字、中划线或下划线组成，建议不要使用曾经用过的密码。

16.8.3 修改账号权限

在使用RDS过程中，您可以根据需要随时调整实例中账号的权限。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的 ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**数据库管理** > **账号列表**，进入**账号列表**页面。
4. 在目标账号所对应的操作栏中，单击 > **修改权限**。
5. 在 **修改授权**页面，调整账号权限，然后单击**确定**。

16.8.4 删除账号

您可以删除不需要的账号。本章将介绍删除账号的操作步骤。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的 ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**数据库管理** > **账号列表**，进入**账号列表**页面。
4. 在目标账号所对应的操作栏中，单击 > **删除**。
5. 在弹出的确认框中单击**确定**。

16.8.5 修改描述

为便于账号管理，在创建账号时您可以添加备注描述，也可以在账号创建成功后修改备注描述。本章将介绍如何修改账号描述。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的 ID，进入**基本信息**页面。

3. 在左侧导航栏中，选择**数据库管理** > **账号列表**，进入**账号列表**页面。
4. 在目标账号所对应的操作栏中，单击 > **修改描述**。
5. 在**描述**中，修改备注信息。
6. 单击**确定**。

16.9 管理数据库

16.9.1 创建数据库

您可以通过RDS控制台创建数据库。数据库名称实例内唯一，实例间互不影响。在数据库迁移之前，需要在 RDS 实例中创建和本地数据库一致的数据库。

背景信息

注意:

- 同一实例下的数据库共享该实例下的所有资源，每个MySQL 5.6版本的实例最多可以创建500个数据库。
- 如果您要迁移本地数据库到RDS，请在RDS实例中创建与本地数据库一致的数据库。
- 分配数据库账号权限时，请按最小权限原则和业务角色创建账号，并合理分配只读和读写权限。必要时可以把数据库账号和数据库拆分成更小粒度，使每个数据库账号只能访问其业务之内的数据。如果不需要数据库写入操作，请分配只读权限。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的 ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**数据库管理** > **数据库列表**，进入**数据库列表**页面。
4. 单击**添加数据库**。
5. 输入要创建的数据库信息，如图 *339:* 创建数据库所示。

图 339: 创建数据库



参数说明：

- **数据库(DB)名称**：由 2~64 个字符的小写字母、数字、下划线或中划线组成，开头需为字母，结尾需为字母或数字。
- **支持字符集**：设置数据库的字符集，选项有 utf8、gbk、latin1 和 utf8mb4。
- **账号授权**：选择该数据库授权的账号。如果尚未创建账号，该值可以为空。
- **备注说明**：可以备注该数据库的相关信息，便于后续数据库管理，最多支持 256 个英文字符。

6. 单击**确定**。

16.9.2 删除数据库

本章介绍如何在RDS控制台上删除实例中的数据库。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的 ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**数据库管理** > **数据库列表**，进入**数据库列表**页面。
4. 在目标数据库所对应的操作栏中，单击 > **删除**。
5. 在弹出的提示框中单击 **确定**。

16.10 设置访问模式

RDS支持**标准模式**和**高安全模式**两种访问模式。本章介绍两种访问模式的区别及设置方法。

前提条件

实例的网络类型为**经典网络**。

背景信息

标准模式和**高安全模式**两种访问模式的区别如下所示：

- 标准模式：RDS使用负载均衡屏蔽了数据库引擎HA切换对应用层的影响，缩短响应时间，但会小幅度增加连接闪断的概率，失去SQL拦截能力。该模式只支持一种连接地址，在实例同时有内网地址和外网地址的情况下，需先释放内网地址或者外网地址后，才能切换到标准模式。
- 高安全访问模式：具有防止90%的连接闪断和SQL拦截的能力（基于SQL语义分析来实现防御SQL注入攻击），但是会增加20%以上响应时间。该模式支持内外网地址并存。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的 ID，进入**基本信息**页面。
3. 单击页面右上角的**切换访问模式**。

> **说明:** 实例在切换访问模式时，实例的**运行状态**会变为为**创建中**，直到**运行状态**变为**运行中**表示切换访问方式成功。

16.11 安全管理

16.11.1 设置白名单

为了数据库的安全稳定，在启用目标实例前，需先修改白名单。您应该将需要访问数据库的IP地址或者IP段加入到RDS实例的白名单中。本章将介绍如何在设置RDS实例的白名单。

背景信息

注意:

- 系统会给每个实例创建一个默认的default白名单分组，该白名单分组只能被修改或清空，但不能被删除。
- 白名单功能支持设置IP地址（如 10.10.10.1）或者IP段（如 10.10.10.0/24，表示10.10.10.X 的IP地址都可以访问数据库）。
- 在每个白名单分组中，MySQL 5.6版本的实例可以添加1000个IP。
- 若将白名单设置为%或者0.0.0.0/0，代表允许任何IP访问RDS实例。该设置将极大降低数据库的安全性，如非必要请勿使用。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**安全控制** > **白名单设置**，进入**白名单列表**页面。
4. 单击**添加白名单分组**。
5. 在**修改白名单分组**页面的**组内白名单**栏中填写需要访问该实例的IP地址或IP段，如图 *340: 添加白名单分组*所示。然后单击**确定**。

图 340: 添加白名单分组



参数说明：

- **分组名称**：长度为2~32个字符，由小写字母、数字或下划线组成，开头需为小写字母，结尾需为字母或数字。在白名单分组创建成功后，该名称将不能被修改。
- **组内白名单**：填写允许访问RDS实例的IP地址或者IP段。
  - 若填写IP段，如10.10.10.0/24，则表示10.10.10.X的IP地址都可以访问该RDS实例。
  - 若您需要添加多个IP，请用英文逗号隔开（逗号前后都不能加空格），例如192.168.0.1,172.16.213.9。

后续操作

正确使用白名单可以让RDS得到高级别的访问安全保护，建议您定期维护白名单。后续操作中，您可以单击  来修改白名单。单击  来清空default白名单或删除自定义白名单。

16.11.2 审计日志

您可以通过RDS控制台查询实例的SQL日志、操作日志和错误日志，来帮助您做故障定位分析。本章将介绍如何查询审计日志。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**安全控制** > **审计日志**，进入**审计日志**页面。
4. 选择要查看的 **SQL 日志**、**错误日志**或**操作日志**标签页。
5. 选择要查询的时间范围，然后单击**确定**。

后续操作

您可以在**操作日志**页面，单击**导出**，导出操作日志，用于线下分析。

16.12 性能优化

16.12.1 慢SQL统计

您可以通过RDS控制台查询慢SQL统计，来帮助您做故障定位分析。

操作步骤

1. 登录*RDS*控制台。

2. 单击目标实例的ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**性能优化** > **慢SQL统计**，进入**慢SQL列表**页面。
4. 选择时间范围，然后单击**查询**。

> **说明:** 该列表中不提供最近两个小时的慢日志，如需查询请通过MySQL库中的slow_log表查询。

16.12.2 缺失索引

根据RDS的SQL语句运行情况及性能，系统将提示缺失索引的数据库表，并提示添加索引的SQL语句。本章将介绍如何查询缺失索引。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**性能优化** > **缺失索引**，进入**索引缺失列表**页面，即可查询所有缺失的索引。

16.13 备份与恢复

16.13.1 备份RDS数据

16.13.1.1 自动备份

自动备份支持全量物理备份。云数据库会执行您设定的备份策略，自动备份数据库。本章将介绍如何设置自动备份的备份策略。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**备份与恢复** > **备份列表**，进入**备份与恢复**页面。
4. 选择**备份设置**标签页，单击**设置**。
5. 在**备份设置**页面设置备份规格，如图 *341:* 设置备份策略所示。

图 341: 设置备份策略



参数说明：

- **保留天数**：数据备份文件保留的天数，默认为 7 天，可以设置 7~730 天。
- **备份周期**：可以设置为一星期中的某一天或者某几天。
- **备份时间**：可以设置为任意时段，以小时为单位。

6. 单击**确定**。

16.13.1.2 手动备份

手动备份支持全量物理备份和全量逻辑备份。本章将介绍如何手动备份 RDS 数据。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的ID，进入**基本信息**页面。
3. 单击页面右上角的**实例备份**。
4. 在**实例备份**页面选择备份方式和备份类型，如图 *342:* 手动备份设置所示。

图 342: 手动备份设置



参数说明：

- **备份方式**：可以选择物理备份或逻辑备份。
- **备份类型**：可以选择全量备份或自动备份。

5. 单击**确定**。

16.13.2 恢复RDS数据

16.13.2.1 通过临时实例恢复到主实例

用户对数据库误操作引起的损失可以通过数据恢复功能最大程度地减少。我们推荐您使用通过临时实例恢复到主实例的方法恢复数据。本章将介绍如何通过临时实例将备份数据恢复到主实例。

前提条件

- 在您选择创建临时实例的时间点之前，RDS有全量的备份。
- 您选择的时间点相邻的两个备份集之间的时间段内没有进行过恢复实例操作。

背景信息

您可以通过临时实例恢复到主实例，创建临时实例并不影响当前的生产实例，而是提供一个临时实例供数据访问。建议您先将数据恢复到临时实例，验证无误后再回迁到主实例，避免数据恢复对业务造成冲击。

说明:

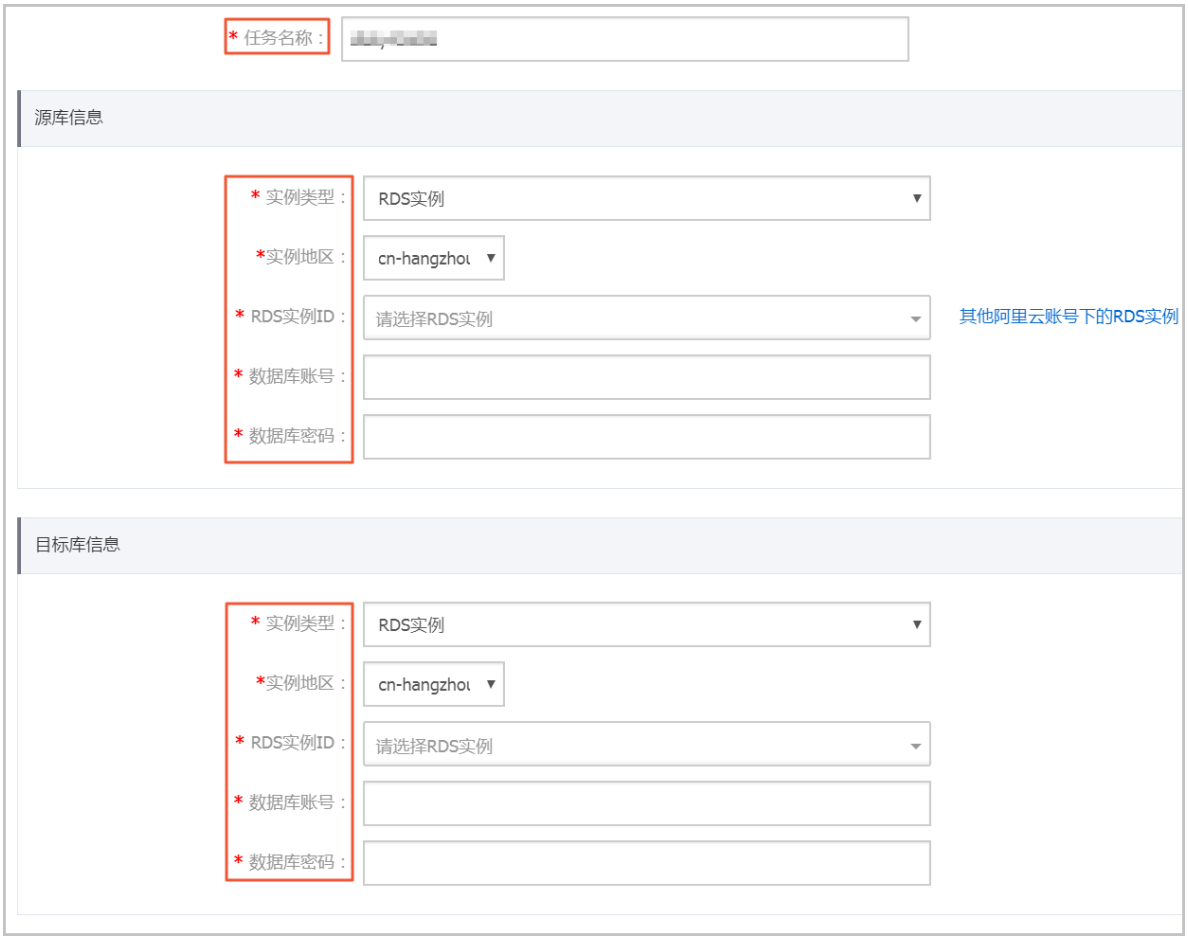
- 创建临时实例时，您只能选择过去7天内的时间点。
- 临时实例会继承备份文件的账号和密码、但沿用当前实例的网络类型。
- 同一时间仅可生成一个临时实例。如果要创建新的临时实例，需先删除已存在的临时实例。
- 临时实例48小时内有效。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**备份与恢复** > **临时实例**，进入**临时实例**页面。
4. 单击**从时间节点创建临时实例**。
5. 在**按时间点创建临时实例**页面选择一个时间点。
6. 单击**确定**。

7. 返回**RDS实例**列表页面。
8. 在左侧导航栏中，选择**云产品管理** > **数据传输**，登录数据传输（DTS）控制台。
9. 单击**dts**，进入**迁移任务列表**页面。
10. 单击**创建迁移任务**，进入**创建迁移任务**页面。
11. 输入任务名称、源数据库信息和目标数据库信息，如图 *343:* 填写源库及目标库信息所示。

图 343: 填写源库及目标库信息



参数说明：

- **任务名称**：默认情况下，DTS为每个任务自动生成一个任务名称，您可以修改这个名称，为任务配置一个具备业务意义的名称，便于后续的任务识别。
- **源库信息**：
  - **实例类型**：数据库的实例类型，选择**RDS实例**。
  - **实例地区**：选择与主实例相同的地域。
  - **RDS实例ID**：单击下拉菜单，选择临时实例的ID。

- **数据库账号**：与主实例账号一致，且该账号必须具有对所有要迁移数据的读/写权限。
- **数据库密码**：与主实例账号的密码一致。

- **目标库信息**
  - **实例类型**：默认为**RDS实例**。
  - **实例地区**：主实例所在地域。
  - **RDS实例ID**：目标RDS实例的ID。单击下拉菜单选择临时实例所对应的主实例ID。
  - **数据库账号**：主实例账号，该账号必须具有对所有要迁移数据的读/写权限。
  - **数据库密码**：主实例账号的密码。

12.单击**授权白名单并进入下一步**，进入**迁移类型及列表**页面。

13.选择迁移类型，并在**迁移对象**框中选择要迁移的对象，单击>将要迁移的对象添加到**已选择对象**框中，如图 *344:* 选择迁移类型及迁移对象所示。

说明: 如果要修改迁移对象在目标数据库上的名字，可以在**已选择对象**框中，将鼠标放在需要修改的数据库上面，即会显示出**编辑**按钮，如图 *344:* 选择迁移类型及迁移对象所示。

图 344: 选择迁移类型及迁移对象

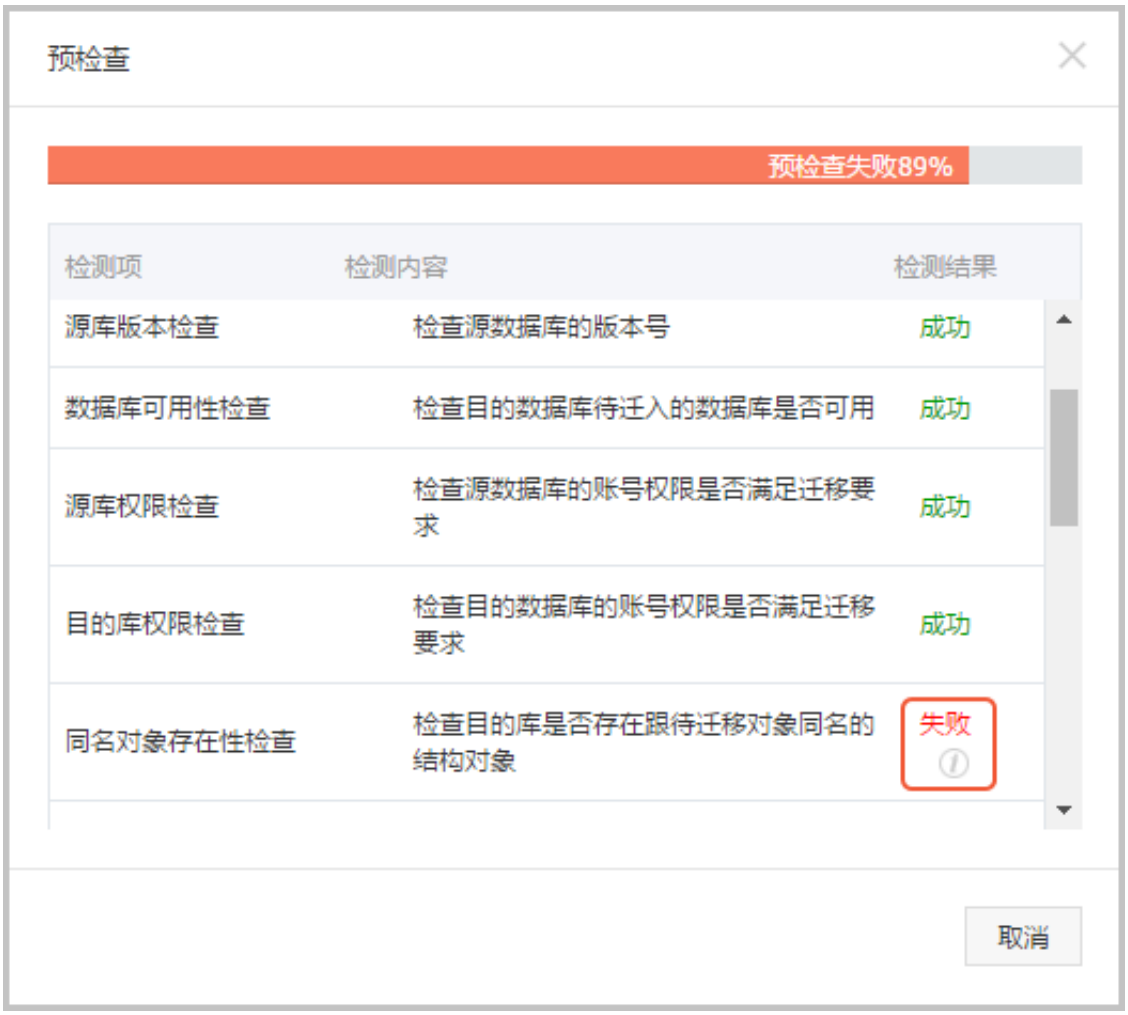


14.单击**预检查并启动**，如图 *345:* 预检查所示。

说明:

- 在迁移任务正式启动之前，会先进行前置预检查，只有预检查通过后，才能成功启动迁移。
- 如果预检查失败，可以单击具体检查项后的按钮，查看具体的失败详情，并根据失败原因修复后，重新进行预检查。
- 当修复完成后，可以在任务列表中，选择这个任务，然后重新启动预检查。

图 345: 预检查



15.当预检查通过后，启动迁移任务。任务启动后，您可以在**迁移任务列表**页面查看具体的迁移状态及进度。

16.13.2.2 直接恢复到主实例

您可以直接将指定备份数据覆盖到主实例，但指定的备份数据创建之后产生的数据将会全部丢失。建议您通过创建临时实例来进行更加安全的数据恢复和回迁。本章将介绍如何直接将制定备份数据覆盖到主实例上。

背景信息

注意： 如果实例中创建了只读实例，将不能直接将指定备份数据覆盖到主实例。请通过创建临时实例进行数据恢复，详情请参见通过临时实例恢复到主实例。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**备份与恢复** > **备份列表**，进入**备份与恢复**页面。
4. 选择**备份列表**标签页。
5. 选择时间范围，然后单击**查询**。
6. 找到要恢复的备份集，然后在其对应的操作栏中，单击 > **覆盖性恢复**。

16.13.3 Binlog日志

本章将介绍如何查询并下载RDS实例的Binlog日志。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的ID，进入**基本信息**页面。
3. 在左侧导航栏中，选择**备份与恢复** > **Binlog列表**，进入**Binlog文件列表**页面。
4. 选择时间范围，然后单击**查询**，即可查询所需要的Binlog日志。
5. 若需要下载Binlog日志，在其所对应的操作栏中，单击 > **下载**。

16.14 系统资源监控

为方便您查看和掌握实例的运行状态，RDS控制台提供了丰富的性能监控项。

操作步骤

1. 登录*RDS*控制台。
2. 单击目标实例的ID，进入**基本信息**页面。

3. 在左侧导航栏中，选择**系统资源监控**。
4. 按需求选择查看数据库的系统资源、数据库性能、InnoDB和MyISAM引擎的监控数据，每个页面具体的监控项如表 *41:* 监控项列表所示。

表 41: 监控项列表

页面名称	监控项	说明	监控频率	监控周期
系统资源	磁盘空间	实例的磁盘空间使用量，包括磁盘空间总体使用量、数据空间使用量、日志空间使用量、临时文件空间使用量和系统文件空间使用。 量单位：MByte	60秒/次 300秒/次	30天
	IOPS	实例的每秒I/O请求次数。 单位：次/秒	60秒/次 300秒/次	30天
	CPU 利用率	实例的 CPU 利用率（不含操作系统占用）。	60秒/次 300秒/次	30天
	网络流量	实例每秒钟的输入、输出流量。 单位：KByte	60秒/次 300秒/次	30天
数据库性能	QPS/TPS	每秒钟 SQL 语句执行次数和事务处理数。	60秒/次 300秒/次	30天
	临时表	数据库执行 SQL 语句时在硬盘上自动创建的临时表的数量。	60秒/次 300秒/次	30天
	COMDML	数据库每秒 SQL 语句执行次数，包括Insert、Delete、Insert_Select、Replace、Replace_Select、Select、Update。	60秒/次 300秒/次	30天
	ROWDML	InnoDB 每秒钟操作执行次数，包括每秒向日志文件的物理写次数、每秒从 InnoDB 表读取、更新、删除、插入的行数。	60秒/次 300秒/次	30天
InnoDB引擎	InnoDB缓冲池	InnoDB缓冲池的读命中率、利用率以及缓冲池脏块的百分率。	60秒/次 300秒/次	30天
	InnoDB读写量	InnoDB 每秒读取和写入的数据量。 单位：KByte	60秒/次 300秒/次	30天
	InnoDB读写次数	InnoDB每秒钟的读取和写入的次数。	60秒/次 300秒/次	30天

页面名称	监控项	说明	监控频率	监控周期
	InnoDB日志	InnoDB每秒向日志文件的物理写入次数、日志写请求、向日志文件完成的fsync写数量。	60秒/次 300秒/次	30天
MyISAM引擎	MyISAM Key Buffer	MyISAM每秒的Key Buffer读命中率、写命中率、利用率。	60秒/次 300秒/次	30天
	MyISAM读写次数	MyISAM每秒从缓冲池中的读、写次数和每秒钟从硬盘上的读、写次数。	60秒/次 300秒/次	30天

16.15 迁移本地数据到RDS

16.15.1 压缩数据

背景信息

云数据库MySQL 5.6版支持通过TokuDB存储引擎压缩数据。经过大量测试表明，数据表从InnoDB存储引擎转到TokuDB存储引擎后，数据量可以减少80%到90%，即2T的数据量能压缩到400G甚至更低。除了数据压缩外，TokuDB存储引擎还支持事务和在线DDL操作，可以很好兼容运行于MyISAM或InnoDB存储引擎上的应用。

TokuDB的限制说明如下：

- TokuDB存储引擎无法支持外键Foreign Key。
- TokuDB存储引擎不适用于频繁大量读取的场景。

操作步骤

1. 执行如下命令，检查MySQL版本。

```
SELECT version();
```

2. 执行如下命令，设置**loose_tokudb_buffer_pool_ratio**的比例，即TokuDB占用TokuDB和InnoDB共用缓存的比例。

```
select sum(data_length) into @all_size from information_schema.tables where engine='innodb';
select sum(data_length) into @change_size from information_schema.tables where engine='innodb' and concat(table_schema, '.', table_name) in ('XX.XXXX', 'XX.XXXX', 'XX.XXXX');
```

```
select round(@change_size/@all_size*100);
```

说明: 上述命令中，**XX.XXXX**为需要转化成TokuDB存储引擎的数据库和表名。

3. 重启实例。操作步骤请参见 重启实例。
4. 执行如下命令，修改存储引擎。您也可以登录 DMS 修改数据表存储引擎，具体操作请参见《*DMS*产品文档》。

```
ALTER TABLE XX.XXXX ENGINE=TokuDB
```

说明: 上述命令中，**XX.XXXX**为需要转化成TokuDB存储引擎的数据库和表名。

16.15.2 迁移MySQL数据

16.15.2.1 使用DTS迁移MySQL数据

使用数据传输服务（DTS）将本地数据库迁移到云数据库MySQL，可以实现应用不停服务的情况下，平滑完成数据库的迁移工作。本章以有公网IP的本地数据库迁移到 RDS上为例，介绍将本地数据迁移到云上的操作步骤。

背景信息

DTS数据迁移支持MySQL的结构迁移、全量迁移和增量迁移。

- 结构迁移

  DTS会将本地数据库的结构定义迁移到目标实例。目前DTS支持结构迁移的对象有：表、视图、触发器、存储过程、存储函数。

- 全量迁移

  DTS会将本地数据库迁移对象的数据全部迁移到目标实例。如果您还选择了增量迁移，那么全量迁移过程中，为了保证数据一致性，无主键的非事务表会被锁定，锁定期间这些表无法写入，锁定时长依赖于这些表的数据量大小，在这些无主键非事务表迁移完成后，锁才会释放。

- 增量迁移

  增量迁移会将迁移过程进行数据变更同步到目标实例，如果迁移期间进行了DDL操作，那么这些结构变更不会迁移到目标实例。

迁移限制

将本地数据库迁移到RDS上有以下限制：

- 迁移过程中，不支持DDL操作。
- 结构迁移不支持event的迁移。
- 如果使用了对象名映射功能后，依赖这个对象的其他对象可能迁移失败。
- 当选择增量迁移时，本地 MySQL 实例需要开启 binlog，且本地库的 binlog_format 要为 row。如果本地 MySQL 为5.6版本时，它的 binlog_row_image 还须设置为 full。

前提条件

已完成 RDS 实例数据库的准备，相关操作请参见 设置白名单 的实例必要配置。

准备本地数据

在正式迁移之前，需要先在本地数据库和RDS实例中创建迁移账号，并在RDS实例中创建要迁移的数据库，并将要迁移的数据库的读写权限授权给迁移账号。不同的迁移类型需要不同的权限，如表 *42:* 迁移类型及权限要求所示。

表 42: 迁移类型及权限要求

迁移类型	结构迁移	全量迁移	增量迁移
本地数据库	select	select	• select • replication slave • replication client
RDS实例	读写权限	读写权限	读写权限

1. 执行如下命令，在本地数据库中创建迁移账号。

   CREATE USER 'username'@'host' IDENTIFIED BY 'password';

   参数说明：

   - username：要创建的账号名称。
   - host：指定该账号登录数据库的主机。如果是本地用户可以使用**localhost**，如果想让该用户从任意主机登录，可以使用通配符**%**。
   - password：该账号的登录密码。

   例如：要创建账号为**William**，密码为**Changme123**的账号，从任意主机登录本地数据库，命令如下：

   CREATE USER 'William'@'%' IDENTIFIED BY 'Changme123';

2. 在本地数据库中给迁移账号授权，本地数据库中迁移账号的权限要求请参见表 *42:* 迁移类型及权限要求。

```
GRANT privileges ON databasename.tablename TO 'username'@'host' WITH GRANT OPTION;
```

参数说明：

- privileges：该账号的操作权限，如 SELECT、INSERT、UPDATE 等。如果要授权该账号所有权限，则使用**ALL**。
- databasename：数据库名。如果要授权该账号所有的数据库权限，则使用通配符*。
- tablename：表名。如果要授权该账号所有的表权限，则使用通配符*。
- username：要授权的账号名称。
- host：授权登录数据库的主机名。如果是本地用户可以使用**localhost**，如果想让该用户从任意主机登录，可以使用通配符**%**。
- WITH GRANT OPTION：授权该账号能使用GRANT命令，该参数为可选。

例如：授权账号**William**对所有数据库和表的所有权限，并可以从任意主机登录本地数据库，命令如下：

```
GRANT ALL ON *.* TO 'William'@'%';
```



说明: 如果需要进行增量迁移，那么需要开启本地数据库的Binlog并正确设置，执行以下步骤。

3. 使用如下命令查询是否开启了Binlog。

```
show global variables like "log_bin";
```

如果查询结果为log_bin=OFF，那么本地数据库没有开启 binlog。为了使迁移过程中产生的增量数据能同步迁移，需要修改配置文件*my.cnf*中的如下参数。

```
log_bin=mysql_bin
binlog_format=row
server_id=大于 1 的整数
binlog_row_image=full //当本地 MySQL 版本大于 5.6 时，则需设置该项
```

4. 修改完成后，执行如下命令，重启MySQL进程。

```
$mysql_dir/bin/mysqladmin -u root -p shutdown
$mysql_dir/bin/safe_mysqld &
```

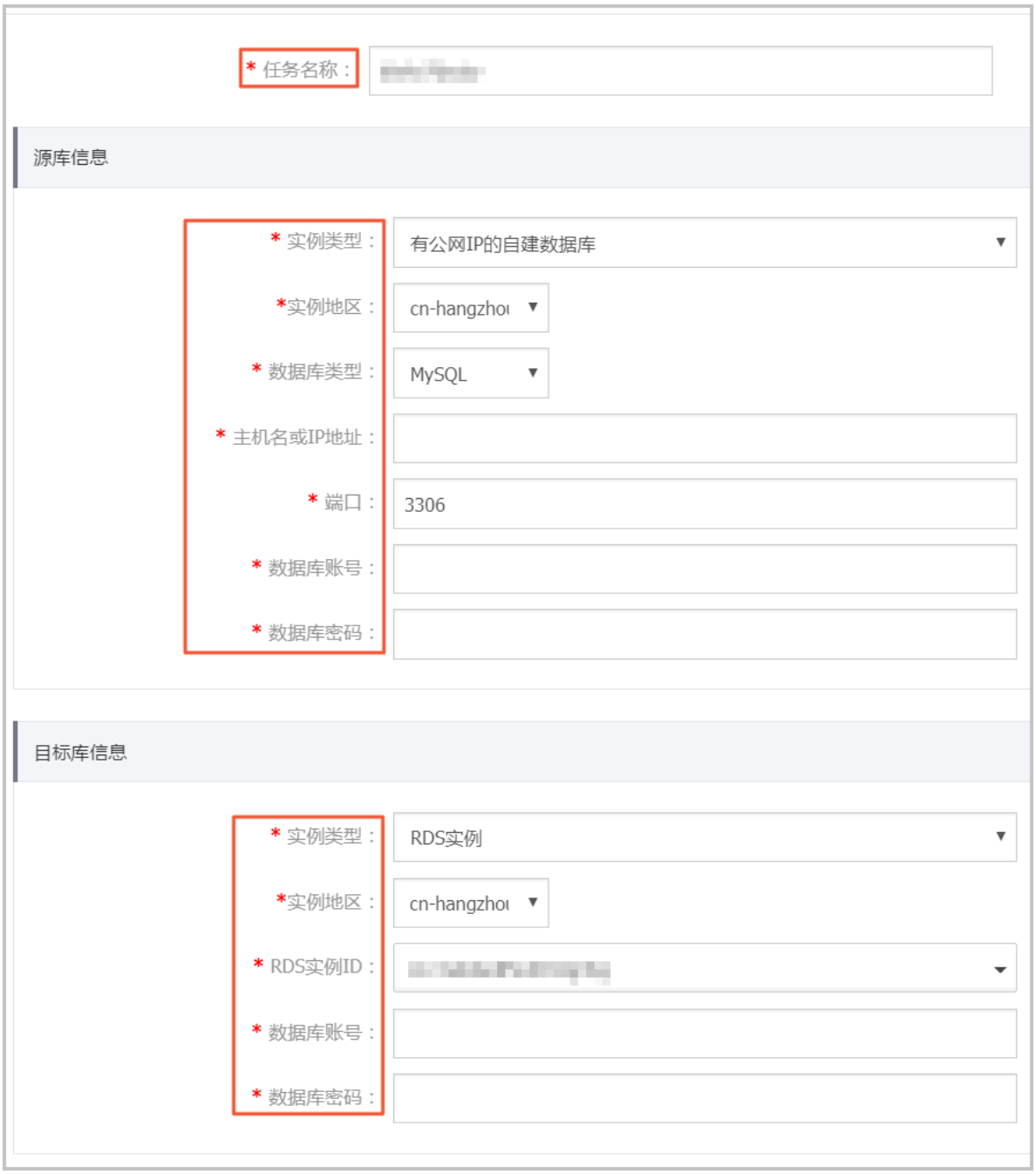
其中，*mysql_dir*为MySQL安装目录。

正式迁移操作

数据准备完毕后，即可进入正式的迁移操作。

1. 登录云资源控制台。
2. 在左侧导航栏中，选择**云产品管理** > **数据传输**，登录数据传输（DTS）控制台。
3. 单击**dts**，进入**迁移任务列表**页面。
4. 单击**创建迁移任务**，进入**创建迁移任务**页面。
5. 输入任务名称、源数据库信息和目标数据库信息，如图 *346:* 填写源库及目标库信息所示。

图 346: 填写源库及目标库信息



参数说明：

- **任务名称**：默认情况下，DTS为每个任务自动生成一个任务名称，您可以修改这个名称，为任务配置一个具备业务意义的名称，便于后续的任务识别。
- **源库信息**：
  - **实例类型**：数据库的实例类型，选择**有公网IP的自建数据库**。
  - **实例地区**：选择与主实例相同的地域。
  - **数据库类型**：本地数据库的类型。

- **主机名或IP地址**：本地数据库的公网地址。
- **端口**：本地数据库的公网端口。
- **数据库账号**：本地数据库的迁移账号，该账号必须具有对所有要迁移数据的读/写权限。
- **数据库密码**：本地数据库迁移账号对应的密码。

- **目标库信息**
  - **实例类型**：默认为**RDS实例**。
  - **实例地区**：主实例所在地域。
  - **RDS实例ID**：目标RDS实例的ID。单击下拉菜单选择临时实例所对应的主实例ID。
  - **数据库账号**：目标 RDS 数据库的迁移账号，该账号必须具有对所有要迁移数据的读/写权限。
  - **数据库密码**：目标 RDS 数据库迁移账号对应的密码。

6. 单击**授权白名单并进入下一步**，进入**迁移类型及列表**页面。
7. 选择迁移类型，并在**迁移对象**框中选择要迁移的对象，单击**>**将要迁移的对象添加到**已选择对象**框中，如图 *347:* 选择迁移类型及迁移对象所示。

   说明: 如果要修改迁移对象在目标数据库上的名字，可以在**已选择对象**框中，将鼠标放在需要修改的数据库上面，即会显示出**编辑**按钮，如图 *347:* 选择迁移类型及迁移对象所示。

图 347: 选择迁移类型及迁移对象

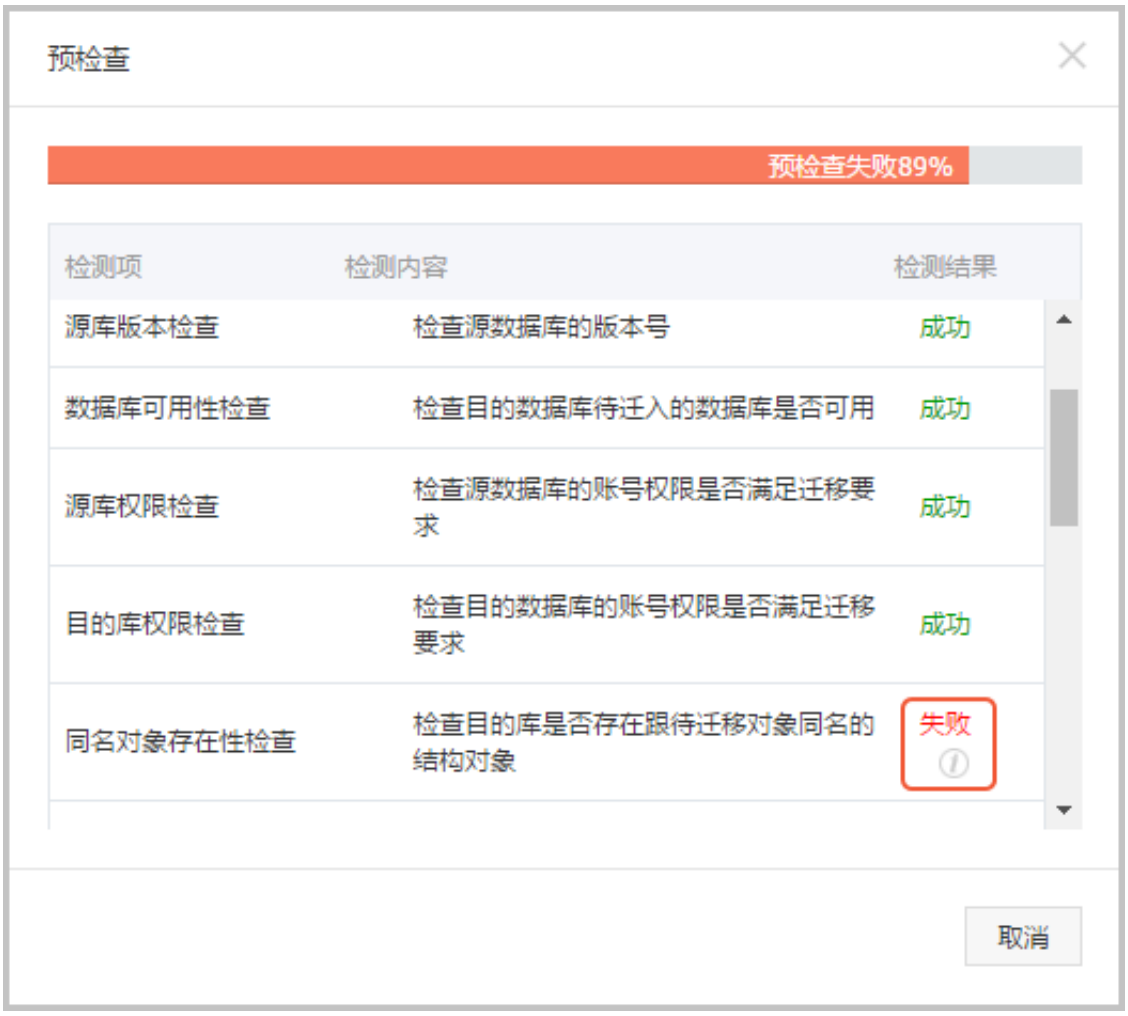


8. 单击**预检查并启动**，如图 *348:* 预检查所示。

说明:

- 在迁移任务正式启动之前，会先进行前置预检查，只有预检查通过后，才能成功启动迁移。
- 如果预检查失败，可以单击具体检查项后的按钮，查看具体的失败详情，并根据失败原因修复后，重新进行预检查。
- 当修复完成后，可以在任务列表中，选择这个任务，然后重新启动预检查。

图 348: 预检查



9. 当预检查通过后，启动迁移任务。任务启动后，您可以在**迁移任务列表**页面查看具体的迁移状态及进度。

后续操作

因迁移账号拥有读写权限，为了保证本地数据库安全，请在数据迁移完成后，删除本地数据库和RDS实例中的迁移账号。

16.15.2.2 使用mysqldump迁移MySQL数据

使用mysqldump工具的优点是简单易用、容易上手，缺点是停机时间较长，因此它适用于数据量不大，或者允许停机的时间较长的情况。

前提条件

已开通云服务器ECS。

背景信息

由于RDS提供的关系型数据库服务与原生的数据库服务完全兼容，故将原有数据库迁移到RDS实例的过程，与从一个MySQL服务器迁移到另外一台MySQL服务器的过程基本类似。

在正式迁移之前，需要先在本地数据库中创建迁移账号，并将要迁移的数据库的读写权限授权给迁移账号。

操作步骤

1. 执行如下命令，在本地数据库中创建迁移账号。

   ```
   CREATE USER 'username'@'host' IDENTIFIED BY 'password';
   ```

   参数说明：

   - username：要创建的账号名称。
   - host：指定该账号登录数据库的主机。如果是本地用户可以使用**localhost**，如果想让该用户从任意主机登录，可以使用通配符**%**。
   - password：该账号的登录密码。

   例如：要创建账号为**William**，密码为**Changme123**的账号，从任意主机登录本地数据库，命令如下：

   ```
   CREATE USER 'William'@'%' IDENTIFIED BY 'Changme123';
   ```

2. 执行如下命令，在本地数据库中给迁移账号授权。

   ```
   GRANT SELECT ON databasename.tablename TO 'username'@'host' WITH GRANT OPTION;
   GRANT REPLICATION SLAVE ON databasename.tablename TO 'username'@'host' WITH GRANT OPTION;

   GRANT REPLICATION SLAVE ON databasename.tablename TO 'username'@'host' WITH GRANT OPTION;
   ```

   参数说明：

   - privileges：该账号的操作权限，如SELECT、INSERT、UPDATE 等。如果要授权该账号所有权限，则使用**ALL**。
   - databasename：数据库名。如果要授权该账号所有的数据库权限，则使用通配符*。
   - tablename：表名。如果要授权该账号所有的表权限，则使用通配符*。
   - username：要授权的账号名。
   - host：授权登录数据库的主机名。如果是本地用户可以使用**localhost**，如果想让该用户从任意主机登录，可以使用通配符**%**。

- WITH GRANT OPTION：授权该账号能使用GRANT命令，该参数为可选。

例如：授权账号**William**对所有数据库和表的所有权限，并可以从任意主机登录本地数据库，命令如下：

```
GRANT ALL ON *.* TO 'William'@'%';
```

3. 使用mysqldump的数据导出工具，将本地数据库数据导出为数据文件。

> **注意：** 导出期间请勿进行数据更新。本步骤仅仅导出数据，不包括存储过程、触发器及函数。

```
mysqldump -h localIp -u userName -p --opt --default-character-set=utf8 --hex-blob dbName --skip-triggers > /tmp/dbName.sql
```

参数说明：

- localIp：本地数据库服务器IP地址。
- userName：本地数据库的迁移账号。
- dbName：需要迁移的数据库名。
- /tmp/dbName.sql：备份生成的文件名。

4. 使用mysqldump导出存储过程、触发器和函数。

> **注意：** 若数据库中没有使用存储过程、触发器和函数，可跳过此步骤。在导出存储过程、触发器和函数时，需要将 definer 去掉，以兼容 RDS。

```
mysqldump -h localIp -u userName -p --opt --default-character-set=utf8 --hex-blob dbName -R | sed -e 's/DEFINER[ ]*=[ ]*[^*]*\*/\*/' > /tmp/triggerProcedure.sql
```

参数说明：

- localIp：本地数据库服务器IP地址。
- userName：本地数据库的迁移账号。
- dbName：需要迁移的数据库名。
- /tmp/triggerProcedure.sql：备份生成的文件名。

5. 将数据文件和存储过程文件上传到ECS上。

本节以文件上传到如下路径为例：

```
/tmp/dbName.sql
```

```
/tmp/triggerProcedure.sql
```

6. 登录ECS，将数据文件和存储过程文件导入到目标RDS中。

   mysql -h intranet4example.mysql.rds.aliyuncs.com –u userName -p dbName < /tmp/dbName.sql

   mysql -h intranet4example.mysql.rds.aliyuncs.com -u userName -p dbName < /tmp/triggerProcedure.sql

   参数说明：

   - intranet4example.mysql.rds.aliyuncs.com：RDS实例连接地址，本节以内网地址为例。
   - userName：RDS数据库的迁移账号。
   - dbName：需要导入的数据库名。
   - /tmp/dbName.sql：要导入的数据文件名。
   - /tmp/triggerProcedure.sql：要导入的存储过程文件名。

16.16 典型应用

16.16.1 多结构数据存储

背景信息

OSS是阿里云对外提供的海量、安全、低成本、高可靠的云存储服务。RDS可以和OSS搭配使用，组成多类型数据存储解决方案。
例如，当业务应用为论坛时，RDS搭配OSS使用，注册用户的图像、帖子内容的图像等资源可以存储在OSS中，以减少RDS的存储压力。

如下代码为OSS与RDS搭配使用的示例。

操作步骤

1. 执行如下命令，初始化OssApi。

```
from oss.oss_api import * endpoint=" oss-cn-hangzhou.aliyuncs.com" accessKeyId, accessKeySecret="your id","your secret" oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
```

2. 执行如下命令，新建Bucket。

```
#设置bucket权限为私有res = oss.create_bucket(bucket,"private") print "%s\n%s" % (res.status, res.read())
```

3. 执行如下命令，上传 Object。

```
res = oss.put_object_from_file(bucket, object, "test.txt") print "%s\n%s" % (res.status, res.getheaders())
```

4. 执行如下命令，获取对应的Object。

```
res = oss.get_object_to_file(bucket, object, "/filepath/test.txt") print "%s\n%s" % (res.status, res.getheaders())
```

在ECS应用代码中，RDS中存储了每个用户的id，而用户对应的头像资源存储于OSS中，以简单的Python代码为例：

```
#!/usr/bin/env python from oss.oss_api import *
endpoint" oss-cn-hangzhou.aliyuncs.com" accessKeyId, accessKeySecret="your id", "your secret" oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
user_id = mysql_client.fetch_one(sql)#从RDS中查找用户的user_id
#获取用户头像并下载到对应路径
oss.get_object_to_file(bucket, object, your_path/user_id+’ .png’ )
#处理用户上传头像
oss.put_object_from_file(bucket, object, your_path/user_id+’ .png’ )
```

17 云数据库Redis版

17.1 产品概述

阿里云数据库 Redis 版（ApsaraDB for Redis）是兼容开源 Redis 协议的 Key-Value 类型在线存储服务。它支持字符串（String）、链表（List）、集合（Set）、有序集合（SortedSet）、哈希表（Hash）等多种数据类型，及事务（Transactions）、消息订阅与发布（Pub/Sub）等高级功能。通过“内存+硬盘”的存储方式，云数据库Redis版在提供高速数据读写能力的同时满足数据持久化需求。

除此之外，云数据库 Redis 版作为云计算服务，其硬件和数据部署在云端，有完善的基础设施规划、网络安全保障、系统维护服务。所有这些都无需用户考虑，确保用户专心致力于自身业务创新。

17.2 管理实例

17.2.1 修改实例密码

如果您忘记密码、需要修改旧密码，或者在创建实例时候没有设置密码，您可以重新设置实例的密码。

操作步骤

1. 登录云控制台。
2. 选择**云资源中心** > **云产品管理** > **键值存储**，进入**实例列表**页面。
3. 定位到目标实例，单击实例ID或者**管理** > **详情**进入**实例信息**页面。
4. 单击**重置密码**。
5. 在**重置密码**窗口，输入登录密码并确认密码，单击**提交**。

17.2.2 查看实例详情

创建实例后，您可以在云控制台上查看实例详细信息。

操作步骤

1. 登录云控制台。
2. 选择**云资源中心** > **云产品管理** > **键值存储**，进入**实例列表**页面。
3. 定位目标实例，单击实例ID或者**管理** > **详情**，在**实例信息**页面查看实例详细信息。

实例信息页包含**基本信息**、**配置信息**以及**连接信息**区域，各个区域包含的信息项如下表所示。

表 43: 实例信息页

区域	信息项
基本信息	• 实例ID • 名称 • 状态 • 区域 • 部门 • 所属项目 • 创建时间 • 所在可用区 • 网络类型
配置信息	• 实例规格 • 最大连接数 • 最大内网带宽 • 可维护时间段 • 白名单
连接信息	• 连接地址 • 端口号

17.2.3 修改实例名称

创建实例后，您可以在云控制台修改实例名称。

操作步骤

1. 登录云控制台。
2. 选择**云资源中心** > **云产品管理** > **键值存储**，进入**实例列表**页面。
3. 定位目标实例，单击**管理** > **编辑**。
4. 在**编辑实例信息**窗口，填写新的实例名称，单击**确定**。

17.2.4 变更配置

云数据库Redis版可执行变更配置操作。

背景信息



注意: 变配过程中，实例会发生秒级闪断。

操作步骤

1. 登录云控制台。
2. 选择**云资源中心** > **云产品管理** > **键值存储**，进入**实例列表**页面。
3. 定位目标实例，单击实例ID或**管理**>**变更实例**。
4. 在**变更实例**窗口，选择需要的**实例规格**，单击**确认**。

   变更实例成功后提示变更配置成功。

17.2.5 设置IP白名单

为了数据库的安全稳定，在开始使用 Redis 实例前，您需要将访问数据库的 IP 地址或者 IP 段加到目标实例的白名单中。

背景信息

正确使用白名单可以让 Redis 得到高级别的访问安全保护，建议您定期维护白名单。

操作步骤

1. 登录云控制台。
2. 选择**云资源中心** > **云产品管理** > **键值存储**，进入**实例列表**页面。
3. 定位目标实例，单击实例ID或者**管理** > **详情**。
4. 在**实例信息**页面，单击**修改白名单**按钮。
5. 在**修改白名单**窗口，填写白名单 IP 列表，单击**确认**。

   IP 白名单设置为 0.0.0.0/0 代表允许所有地址访问，设置为 127.0.0.1 代表禁止所有地址访问。

17.2.6 设置可运维时间

为保障云数据库 Redis 版实例的稳定性，后端系统会不定期对实例、机器进行维护操作。

背景信息

实例维护当天，为保障整个维护过程的稳定性，实例会在所设置的可运维时间前一段时间，进入**实例维护中**的状态，当实例处于该状态时，数据库本身正常的数据访问不会受到任何影响，但控制台上涉及该实例的变更类的功能均暂无法使用（如变更配置），查询类如性能监控等可以正常查阅。



注意: 在进入实例所设置的可运维时间后，实例在维护期间可能会发生闪断，建议您尽量选择业务低峰期为运维时间段。

操作步骤

1. 登录云控制台。
2. 选择**云资源中心** > **云产品管理** > **键值存储**，进入**实例列表**页面。
3. 定位到目标实例，单击实例ID或者**管理** > **详情**进入**实例信息**页面。
4. 单击**修改可运维时间**。
5. 在**编辑实例信息**窗口，选择可运维时间，并单击**确定**。

17.3 导入数据

背景信息

redis-cli 是 Redis 原生的命令行工具。云数据库 Redis版支持通过 redis-cli 来导入数据，方便您将已有的 Redis 数据导入到云数据库 Redis版里，实现数据的无缝迁移。

注意事项：

- 由于云数据库 Redis版仅支持从专有云内网访问，此操作方案仅在专有云 ECS 上执行才生效。若您的 Redis 不在专有云 ECS 服务器上，您需要将原有的 AOF 文件复制到 ECS 上再执行以上操作。
- redis-cli 是 Redis 原生的命令行工具。若您在 ECS 上无法使用 redis-cli，可以先下载安装 Redis 后再使用 redis-cli。

对于在专有云 ECS 上自建的 Redis 实例，执行如下操作。

操作步骤

1. 开启现有 Redis 实例的 AOF 功能（如果实例已经启用 AOF 功能则忽略此步骤）。

   ```
   # redis-cli -h old_instance_ip -p old_instance_port config set appendonly yes
   ```

2. 通过 AOF 文件将数据导入到新的 ApsaraDB for Redis 实例（假定生成的 AOF 文件名为 append.aof）。

   ```
   # redis-cli -h aliyun_redis_instance_ip -p 6379 -a password --pipe < appendonly.aof
   ```

   注意:

如果原有旧的 Redis 实例不需要一直开启 AOF，可在导入完成后通过以下命令关闭：

```
# redis-cli -h old_instance_ip -p old_instance_port config set appendonly no
```

17.4 备份与恢复

随着越来越多的业务开始使用 Redis 作为最终的持久化存储引擎，用户对于数据可靠性就提出了更高的需求。Redis 的备份恢复解决方案全面升级了云数据库 Redis 的数据可靠性。

17.4.1 设置自动备份策略

背景信息

由于越来越多的应用将 Redis 作为持久化存储，所以需要常规的备份机制保证数据误操作之后，具备数据快速恢复的能力。阿里云采用在备节点上执行 RDB 快照备份，备份期间对您的实例访问不会产生性能影响，并且提供了控制台的快捷操作可以让您进行个性化的备份设置。

操作步骤

1. 登录云控制台，定位目标实例。
2. 单击实例 ID 或者**管理** > **详情**进入**实例信息**页面。
3. 选择**备份与恢复**页签。
4. 单击**备份设置**。
5. 单击**设置**，在**备份设置**窗口，自定义自动备份的周期和时间。

   注意: 默认备份数据的保留时间是7天，不可修改。

6. 单击**确定**，完成自动备份设置。

17.4.2 手动备份

除了备份常规设置外，您还可以在控制台上随机发起一次手工备份。

操作步骤

1. 登录云控制台，定位目标实例。
2. 单击实例 ID 或者**管理** > **详情**进入**实例信息**页面。
3. 选择**备份与恢复**页签。
4. 单击**备份数据**。
5. 单击右上角的**创建备份**。

6. 在**备份实例**窗口，单击**确认**，对实例进行立即备份。



说明: 您可以在**备份数据**页，选择时间范围，查询历史备份数据。默认备份数据的保留时间是7天，您可以查询7天以内的历史备份数据。

17.4.3 备份存档

背景信息

由于行业监管需要或者公司制度，需要将 Redis 中的数据定期备份存档，云数据库 Redis 版提供了备份存档功能，云数据库 Redis 服务自动将自动备份或者手动备份文件保存至 OSS 上。目前阿里云将为您在 OSS 保存7天内的备份文件，7天外的备份文件将会自动删除。

如果您需要更长时间的数据存档，可以直接在控制台上复制链接将数据库备份文件自行下载到本地进行长时间存储。

操作步骤

1. 登录云控制台，定位目标实例。
2. 单击实例 ID 或者**管理** > **详情**进入**实例信息**页面。
3. 选择**备份与恢复**页签。
4. 在**备份数据**页，选择要归档的备份数据集，单击**下载**。

17.4.4 数据恢复

您对数据库误操作引起的损失可以通过数据恢复功能最大程度地减少。目前 Redis 提供按照备份数据的恢复方式。

操作步骤

1. 登录云控制台，定位目标实例。
2. 单击实例 ID 或者**管理** > **详情**进入**实例信息**页面。
3. 选择**备份与恢复**页签。
4. 在**备份与恢复**页面，选择**备份数据**页签。
5. 选择要恢复的时间范围，单击**查询**，然后选择目标备份文件，单击**管理** > **数据恢复**。
6. 在数据恢复的窗口，选择**确定**直接对原实例进行数据恢复。



注意: 由于数据恢复操作有较高的风险，如果时间可控还是建议先采用克隆实例的方式，基于一个需要恢复的备份集创建一个实例，验证数据正确性之后，再进行数据恢复操作。

17.5 设置云数据库参数

云数据库Redis版允许用户您部分实例参数，具体可修改的参数项请参见 Redis 云管控平台中的**参数设置**。

背景信息

由于云数据库服务与原生的数据库服务完全兼容，故设置云数据库参数方法和本地数据库参数设置方法基本类似。您可以参照本例，通过 Redis 管理控制台修改参数，也可以通过 Redis CLI 命令行的方式，执行相关命令修改参数设置。

数据库参数说明请单击以下链接，参见各版本数据库的官方文档。

- *redis.conf for Redis 3.0*
- *redis.conf for Redis 2.8*

操作步骤

1. 登录云控制台，定位目标实例。
2. 单击实例 ID 或者**管理** > **详情**进入**实例信息**页面。
3. 选择**参数设置**页签。
4. 选择需要修改的参数，单击**管理** > **修改**。
5. 修改参数值，单击**确定**。

17.6 Redis支持的命令

云数据库 Redis 版兼容 Redis 的3.0版本，支持 Redis 3.0 的 Geo 命令。Redis 命令请参见：*http://redis.io/commands*。

支持的命令操作

Key（**键**）	String（**字符串**）	Hash（**哈希表**）	List（**列表**）	Set（**集合**）	SortedSet（**有序集合**）
DEL	APPEND	HDEL	BLPOP	SADD	ZADD
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD
EXISTS	BITOP	HGET	BRPOPLPUSH	SDIFF	ZCOUNT
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY
EXPIREAT	DECR	HINCRBY	LINSERT	SINTER	ZRANGE
MOVE	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE

Key（键）	String（字符串）	Hash（哈希表）	List（列表）	Set（集合）	SortedSet（有序集合）
PERSIST	GET	HKEYS	LPOP	SISMEMBER	ZRANK
PEXPIRE	GETBIT	HLEN	LPUSH	SMEMBERS	ZREM
PEXPTREAT	GETRANGE	HMGET	LPUSHX	SMOVE	ZREMRANGEBYRANK
PTTL	GETSET	HMSET	LRANGE	SPOP	ZREMRANGEBYSCORE
RANDOMKEY	INCR	HSET	LREM	SRANDMEMBER	ZREVRANGE
RENAME	INCRBY	HSETNX	LSET	SREM	ZREVRANGEBYSCORE
RENAMENX	INCRBYFLOAT	HVALS	LTRIM	SUNION	ZREVRANK
RESTORE	MGET	HSCAN	RPOP	SUNIONSTORE	ZSCORE
SORT	MSET		RPOPLPUSH	SSCAN	ZUNIONSTORE
TTL	MSETNX		RPUSH		ZINTERSTORE
TYPE	PSETEX		RPUSHX		ZSCAN
SCAN	SET				ZRANGEBYLEX
OBJECT	SETBIT				ZLEXCOUNT
	SETEX				ZREMRANGEBYLEX
	SETNX				
	SETRANGE				
	STRLEN				

以及

HyperLogL	Pub/Sub（发布/订阅）	Transaction（务）	Connection（接）	Server（务器）	Scripting（本）	Geo（地理位置）
PFADD	PSUBSCRIBE	DISCARD	AUTH	FLUSHALL	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	FLUSHDB	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	DBSIZE	SCRIPT EXISTS	GEOPOS
	PUNSUBSCRIBE	UNWATCH	QUIT	TIME	SCRIPT FLUSH	GEODIST

HyperLogL	Pub/Sub（发布/订阅）	Transaction（务）	Connection（接）	Server（务器）	Scripting（本）	Geo（地理位置）
	SUBSCRIBE	WATCH	SELECT	INFO	SCRIPT KILL	GEORADIUS
	UNSUBSCRIBE			KEYS	SCRIPT LOAD	GEORADIUSBYMEMBER
				CLIENT KILL		
				CLIENT LIST		
				CLIENT GETNAME		
				CLIENT SETNAME		
				CONFIG GET		
				MONITOR		
				SLOWLOG		

暂未开放的命令

Keys（键）	Server（服务器）
MIGRATE	BGREWRITEAOF
	BGSAVE
	CONFIG REWRITE
	CONFIG SET
	CONFIG RESETSTAT
	COMMAND
	COMMAND COUNT
	COMMAND GETKEYS
	COMMAND INFO
	DEBUG OBJECT
	DEBUG SEGFAULT

Keys（**键**）	Server（**服务器**）
	LASTSAVE
	ROLE
	SAVE
	SHUTDOWN
	SLAVEOF
	SYNC

集群实例不支持的命令

Transaction	Scripting	Connection	Keys	List
DISCARD	EVAL	SELECT	MOVE	BLPOP
EXEC	EVALSHA		SCAN	BRPOP
MULTI	SCRIPT EXISTS			BRPOPLPUSH
UNWATCH	SCRIPT FLUSH			
WATCH	SCRIPT KILL			
	SCRIPT LOAD			

集群实例受限的命令

Keys	Strings	Lists	Sets	Sorted Sets	HyperLogLog
RENAME	MSETNX	RPOPLPUSH	SINTERSTORE	ZUNIONSTORE	PFMERGE
RENAMENX			SINTER	ZINTERSTORE	
			SUNIONSTORE		
			SUNION		
			SDIFFSTORE		
			SDIFF		
			SMOVE		

说明: 受限命令只支持所操作key均分布在单个hash slot中的场景，没有实现多个hash slot数据的合并功能，因此需要用hash tag的方式确保要操作的key均分布在一个hash slot中。比如有key1，aakey，abkey3，那么我们在存储的时候需要用{key}1,

aa{key},ab{key}3的方式存储，这样调用受限命令时才能生效。具体关于hash tag的用法请参见Redis官方文档。

18 数据传输服务DTS

18.1 产品概述

数据传输（Data Transmission，简称DTS）是阿里云提供的一种支持RDBMS（关系型数据库）、NoSQL、OLAP等多种数据源之间数据交互的数据服务。它提供了数据迁移、实时数据订阅及数据实时同步等多种数据传输能力。通过数据传输可实现不停服数据迁移、数据异地灾备、跨境数据同步、缓存更新策略等多种业务应用场景，助您构建安全、可扩展、高可用的数据架构。

- 数据传输服务DTS的目标是帮您将复杂的数据交互工作承担下来，让您可以专注于上层的业务开发，数据传输服务承诺99.5%的链路稳定性及99.999%的数据可靠性。
- 数据传输服务DTS支持多种数据源类型，例如：
  - 关系型数据库：Oracle、MySQL、SQLServer、PostgreSQL、DRDS、PetaData、OceanBase。
  - NoSQL：MongoDB、Redis。
  - OLAP：MaxCompute、AnalyticDB、流计算。
- 数据传输服务DTS支持RAM主子账号体系，您可以使用子账号创建并管理DTS实例，极大程度提升企业安全性。RAM主子账号相关授权方法请参考《用户指南》中**DTS 支持RAM 主子账号**章节内容。

18.2 快速开始

18.2.1 创建数据迁移任务

数据传输 DTS 提供的数据迁移功能，简单易用，只需 3 个步骤，即可完成整个迁移任务的配置。本小节以 MySQL->RDS for MySQL 为例，详细介绍数据迁移任务的配置流程，其他存储引擎的配置流程类似。

前提条件

- 创建 RDS 数据库。

  在数据迁移过程中，如果待迁移的数据库在目标 RDS 实例中不存在，那么 DTS 自动会创建。但是对于如下两种情况，您需要在配置迁移任务之前，手动创建数据库。

  - 数据库名称不符合：RDS 定义规范（由小写字母、数字、下划线、中划线组成，字母开头，字母或数字结尾，最长 64 个字符）。

- 待迁移数据库，在源 MySQL 实例，跟目标 RDS 实例中名称不同。

对于这两种情况，您需要在配置迁移任务之前，先在 RDS 控制台完成数据库创建。

- 创建迁移账号。

创建迁移任务，需要提供源实例及目标 RDS 实例的迁移账号。各种存储引擎具体需要的数据库权限，可以详细参考《产品手册》中的说明。

如果您的源 MySQL 实例迁移账号尚未创建，那么您可以参考 *Grant*语法说明，创建满足权限要求的迁移账号。

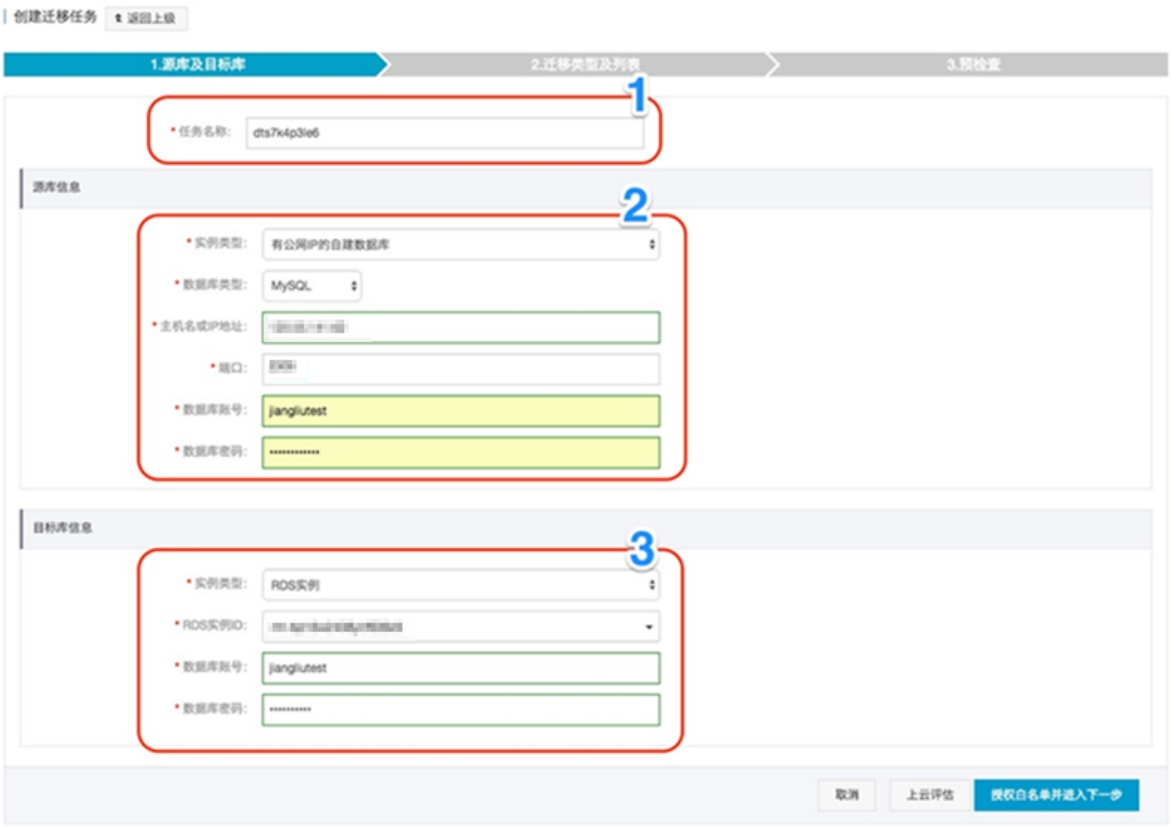
如果您的目标 RDS 实例迁移账号尚未创建，那么您可以参考RDS账号创建流程，创建待迁移数据库有读写权限的迁移账号。

当数据库、迁移账号都创建完成后，就可以开始配置迁移任务了。下面详细介绍下具体的配置步骤。

操作步骤

1. 登录数据传输控制台。
2. 在**数据迁移**页面，单击页面右上角的**创建迁移任务**，开始任务配置。
3. 配置迁移实例连接信息，如图 *349:* 配置迁移实例连接信息所示。

图 349: 配置迁移实例连接信息



在这个步骤中，主要配置迁移任务名称，迁移源实例及目标实例连接信息。其中：

- 任务名称

  默认情况下，DTS 为每个任务自动生成一个任务名称，您可以修改这个名称，为任务配置一个具备业务意义的名称，便于后续的任务识别。

- 源实例连接信息

  本文档以MySQL > RDS For MySQL 迁移为例。

  实例类型：有公网 IP 的自建数据库。

  数据库类型：MySQL。

  主机名或 IP 地址：MySQL。实例连接串。

  端口：MySQL 实例监听端口。

  账号：MySQL 实例访问账号。

  密码：上面 MySQL 访问账号对应的密码。

- 目标 RDS 实例连接信息。

实例类型：RDS 实例。

RDS 实例 ID：选择需要迁移目标 RDS 实例ID。

数据库账号：RDS 实例的访问账号。

数据库密码：上面数据库账号对应的数据库密码。

4. 配置迁移对象及迁移类型，如图 *350:* 配置迁移对象及类型所示。

图 350: 配置迁移对象及类型



在这个步骤中，需要配置迁移类型及迁移对象。其中：

- 迁移类型

  迁移类型包括结构迁移、全量数据迁移及增量数据迁移。

  如果要做全量迁移，那么选择结构迁移 + 全量数据迁移。

  如果要做不停机迁移，那么选择结构迁移 + 全量数据迁移 + 增量数据迁移。

- 迁移对象

  迁移对象，需要选择您要迁移的对象。迁移对象选择的粒度可以为：库、表、列三个粒度。

默认情况下，迁移对象迁移到目标库后，对象名跟源库一致。如果您迁移的对象在源实例跟目标实例上名称不同，那么需要使用 DTS 提供的对象名映射功能，详细使用方式请参考库表列映射。

5. 预检查，如图 *351:* 预检查所示。

在迁移任务正式启动之前，会先进行前置预检查，只有预检查通过后，才能成功启动迁移。具体的预检查内容请参见预检查简介。

如果预检查失败，那么可以单击具体检查项后的按钮，查看具体的失败详情，并根据失败原因修复后，重新进行预检查。

图 351: 预检查



当单击查看后，可以查看到具体的预检查失败原因及修复方法，如图 *352:* 查看失败原因所示。

图 352: 查看失败原因



当修复完成后，可以在任务列表中，选择这个任务，然后重新启动预检查。

6. 启动迁移任务。

   当预检查通过后，我们可以启动迁移任务，任务启动后，可以到任务列表中查看具体的迁移状态及进度。

   说明： 以上为数据迁移任务完整的创建流程。其他的实例类型或者存储引擎的配置流程类似。

18.2.2 创建配置文件导入任务

数据传输服务支持通过离线文件导入的方式进行数据库迁移。这种迁移方式主要是为解决封闭网络环境下的数据库的上云迁移。当数据传输服务器无法连通源数据库时，那么数据传输服务提供的数据迁移方式，无法从云端连接本地 IDC 机房的数据库，进行数据上云迁移。

本小节简单介绍利用文件导入功能实现封闭网络环境下的数据库的上云迁移。

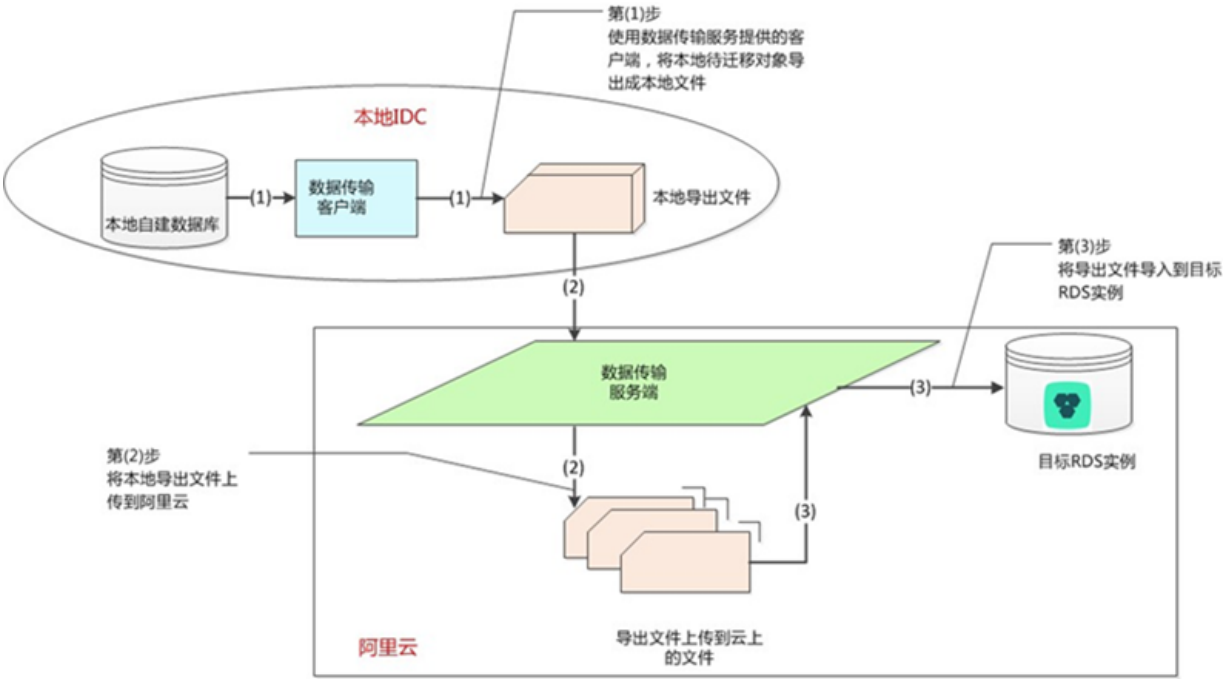
支持数据源

目前文件导入方式仅支持 SQLServer 的上云迁移。

迁移步骤简介

迁移步骤如图 *353:* 迁移步骤所示。

图 353: 迁移步骤



使用文件导入的实现的迁移方案，整个迁移过程分为三个步骤：

1. 数据导出。

   这个步骤是使用数据传输服务提供的客户端将源数据库中需要迁移的数据导出成文件。这个文件的数据存储格式为数据传输 DTS 定义的中间格式。

   为了保证迁移数据的完整性，当使用客户端进行数据导出时，源库最好停写，防止迁移过程中源实例新增写入数据未被迁移到目标库，导致迁移两边数据源数据不一致。

2. 导出文件上传。

   这个步骤是，使用数据传输服务提供的客户端，将本地导出的文件上传到阿里云。

3. 数据导入。

   这个步骤是将第二步中上传上来的文件，导入到目标 RDS 实例中。

迁移方法

下面简单介绍使用文件导入功能进行上云迁移的详细步骤。

1. 导出本地数据。

   a. 下载客户端。

      如果您已经下载过DTS客户端软件，那么这个步骤可以忽略。

如果您尚未下载过 DTS 客户端软件，那么首先需要下载数据传输 DTS 提供的客户端软件，后续将使用这个客户端软件，进行本地待迁移数据的导出，及导出文件的上传操作。

客户端提供 Window64、Linux 两个版本，需要根据本地服务器运行环境选择不同的版本。

客户端的下载入口，在文件导入界面的新手指南，或者是下方的提示链接中，请根据您本地运行服务的操作系统，选择下载不同的版本，如图 *354:* 下载客户端所示。

图 354: 下载客户端



b. 安装客户端。

Window版本

1. 解压客户端。

   使用解压软件，将客户端解压到本地。

2. 启动客户端。

   进入解压目录，双击*start.bat*启动客户端。

3. 连接客户端，进入客户端控制台。

   使用客户端部署服务器 IP:8081 启动客户端 WEB 控制台。其中 8081 为 WEB 服务的监听端口。

Linux版本

A. 解压客户端。

   在本地的 linux 服务器上将客户端解压到本地目录。

B. 启动客户端。

   进入解压目录，执行如下命令，启动客户端。

   `sh start.sh`

   启动完成后，使用如下命令确认客户端是否启动成功。

   `ps -ef|grep portal`

如果输出结果如下，则说明客户端启动成功。

```
[root@dm00 jiangliutest]# ps -ef|grep portal
root      4159     1 50 15:15 pts/0    00:00:36 java -Dmeta.dir=/jiangliutest -Dfile.encoding=UTF-8 -Xms1g -Xmx1g -server -jar dts-portal.war
```

C. 连接客户端。

客户端为 WEB 应用程序，所以需要找一台部署了浏览器，且能够连通客户端的服务器。在浏览器地址中，输入 IP:8081（其中IP为部署客户端的服务器的 IP 地址，8081 为客户端的监听端口），进入客户端 WEB 控制台。

c. 导出数据到本地文件。

A. 进入客户端控制台，单击右上角的**数据导出**，开始数据导出的后续配置。

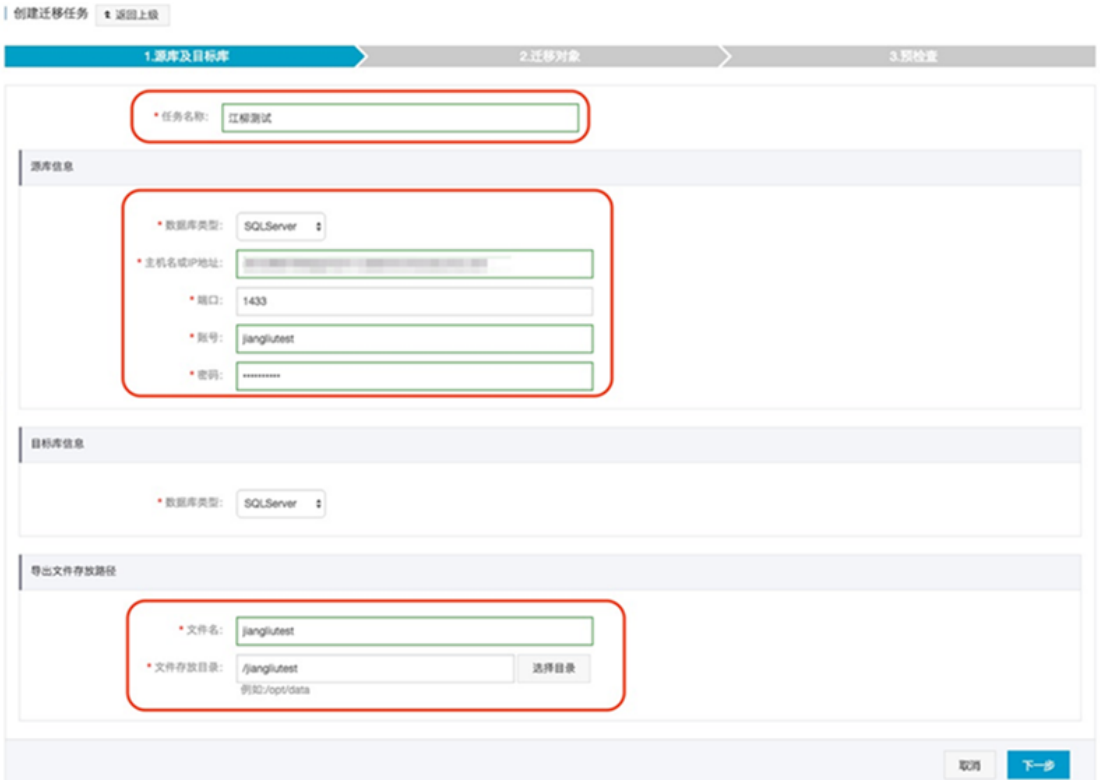
B. 数据库连接信息及导出文件路径配置。

这一步主要配置任务名，导出数据库的连接方式以及导出文件的存储路径。其中：

- 数据导出任务名称：任务名称为导出任务的别名，没有唯一性要求。默认情况下，DTS 为每个导出任务自动生成一个名称，可以修改这个名称，为其配置一个具备业务意义的名称，便于后续的任务识别。
- 待导出数据库的连接方式：目前只能支持 SQLServer，需要填写 SQLServer 数据库的主机名或 IP 地址， 监听端口，账号密码。

C. 导出文件的名称及存放路径，如图 *355:* 导出文件名称及存放路径所示。

图 355: 导出文件名称及存放路径



d. 选择导出对象。

这个步骤主要选择导出对象。 目前导出对象可以支持库、表、列三种粒度。

默认情况下，导出对象在目标实例中的对象名跟源实例相同。如果希望对象在目标实例中的名称跟源实例不同，那么需要使用 DTS 提供的对象名映射功能。具体使用方式请参考库表列映射。

e. 预检查。

在迁移任务正式启动之前，会先进行前置预检查，只有预检查通过后，才能成功启动数据导出。

如果预检查失败，那么可以单击具体检查项后的按钮，查看具体的失败详情，并根据失败原因修复后，重新进行预检查。

当预检查通过后，即可启动导出任务。启动完成后，可以到任务列表中查看导出的具体进度。

2. 上传导出文件。

当数据导出完成后，即可以将导出文件上传。需要使用数据传输客户端进行导出文件上传。

3. 数据导入。

当文件上传完成后，即可登录数据传输 DTS 控制台，将上传文件导入到目标 RDS 实例。

a. 导入文件选择。

进入数据传输 DTS 控制台的数据迁移页面，单击右上角的**文件导入**，开始文件导入的配置。

b. 在文件导入的第一步，选择需要导入的文件。在**迁移对象**框中，选择刚才上传的文件，将其添加到右边的**已选择对象**框中。

c. 选择导入 RDS 实例。

这个步骤主要配置，需要导入的目标 RDS 实例的信息，包括 RDS 实例 ID ，RDS 实例访问账号及密码。这里配置的连接账号必须对待导入对象有读写权限。

d. 预检查。

当导入文件和目标RDS实例配置完成后，即进行导入前的预检查。只有预检查通过后，才能启动数据导入。当数据导入启动后，可以到任务列表查看导入任务的详细进度。

18.2.3 创建实时同步作业

数据传输 DTS 提供的数据实时同步功能，简单易用。只需 3 个步骤，即可完成整个同步链路的配置。本小节介绍如何使用数据传输服务快速创建两个RDS（MySQL）实例之间的实时同步作业，实现 RDS 增量数据的实时同步。

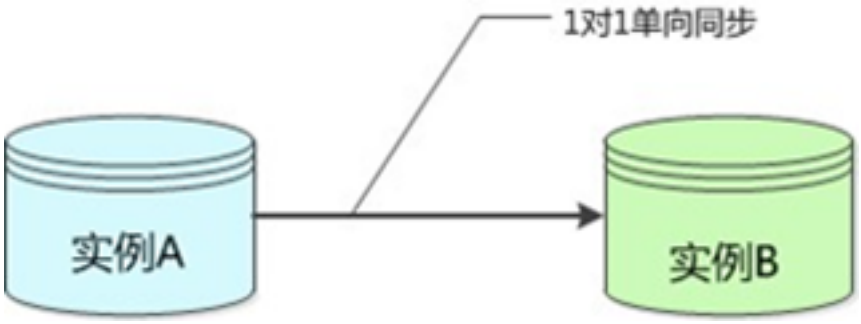
同步限制

- 数据源
  - 目前实时同步只能支持 RDS MySQL 实例，暂不支持其他数据源类型。
  - 目标实例不支持访问模式为标准模式且只有外网连接地址的 RDS 实例。
  - 不支持香港可用区 A 的 RDS 实例的实时同步。
- 同步架构

  目前数据传输服务提供的实时同步功能支持的同步架构有限，其仅能支持如下三种架构：

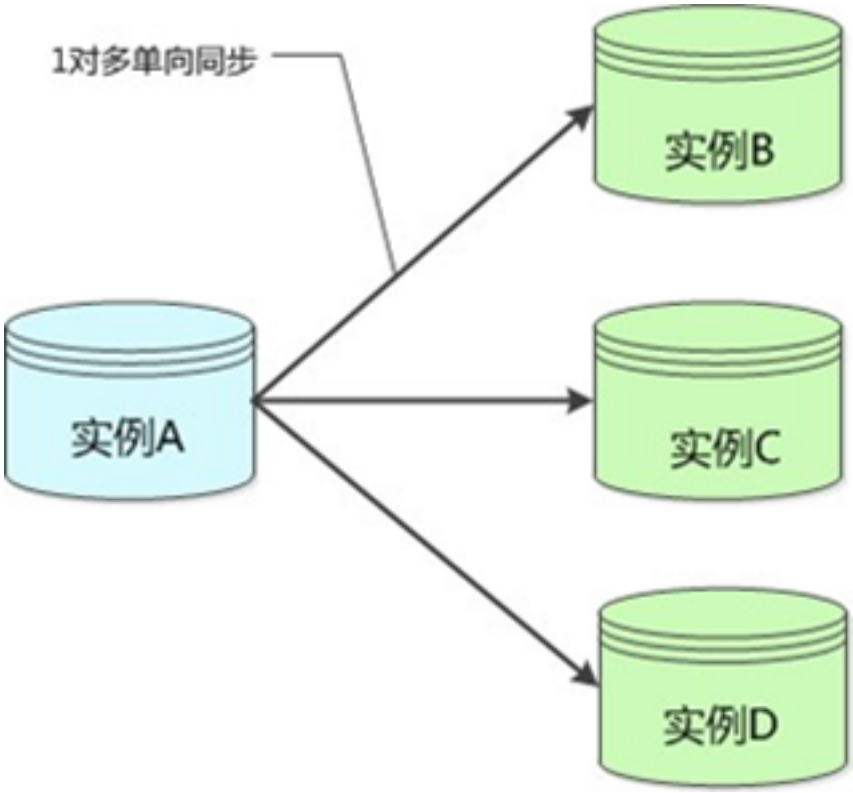
  - A->B 即两个实例之间的单向同步，如图 *356: 1对1*单向同步所示，且要求 B 中同步的对象必须为只读，否则可能导致同步链路异常。

    图 356: 1对1单向同步



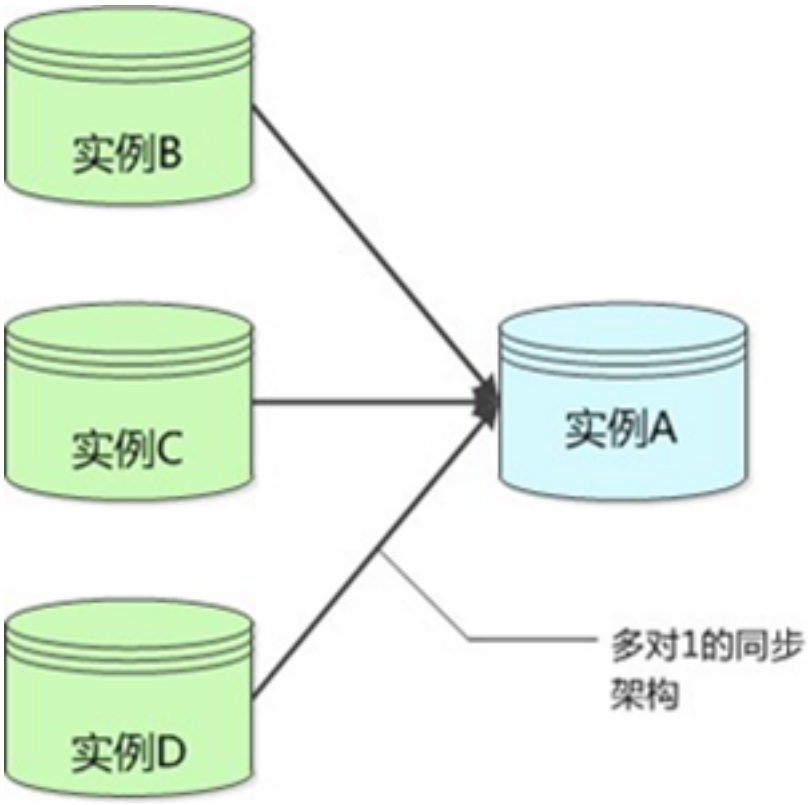
- A->B/C/D 即 1对多的分发式同步架构，如图 *357: 1对多单向同步*所示。这个架构对目标 RDS 实例个数没有限制，但是要求目标实例中的同步对象必须为只读，否则可能导致同步链路异常。

图 357: 1对多单向同步



- B/C/D->A 即多对1的数据汇总架构，如图 *358: 多对1的同步架构*所示。对于多对1 的同步架构，要求每个同步链路的同步对象不相同，保证同步完整性。

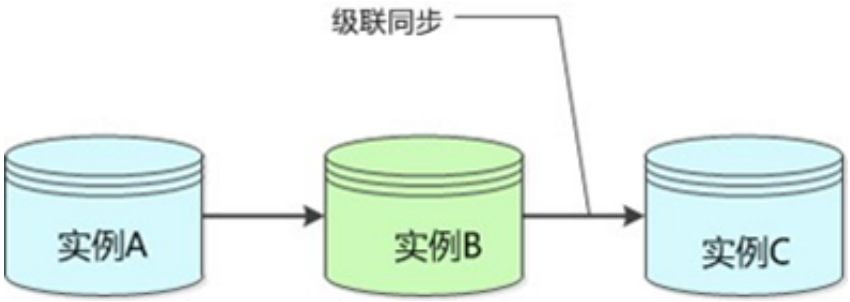
图 358: 多对1的同步架构



对于下面几种架构，暂时不支持：

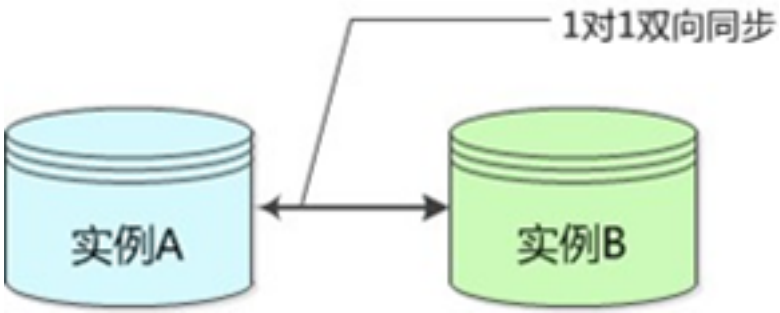
- A->B->C 即级联架构，如图 *359:* 级联架构所示。

图 359: 级联架构



- A->B->A 即实例 A 和实例 B 之间的双向同步架构，如图 *360: 1*对*1*双向同步所示。

图 360: 1对1双向同步



如果您在配置同步链路时，配置了这些不支持的同步架构，那么预检查中的**复杂拓扑**检查项会检查失败。

- 不兼容触发器

  如果同步对象为整个库且这个库中包含了会更新同步表内容的触发器，那么可能导致同步数据不一致。

  触发器内容为在 insert 一条数据到 a 之后，在 b 中插入一条数据。这种情况下，在同步过程中，如果源实例有 a 上的 insert 操作，就会导致 b 表在源实例跟目标实例数据不一致。

  为了解决这个问题，只能将目标实例中的对应触发器删除掉。b 表的数据由源实例同步过去。

前提条件

在配置同步作业前，要确保同步作业的源及目标 RDS 实例都已经存在。如果不存在，那么请先创建 RDS 实例。

操作步骤

下面我们详细介绍下创建任意两个 RDS 实例之间的同步链路的具体步骤。

1. 登录数据传输控制台，进入**数据同步**页面。
2. 单击页面右上角的**创建同步作业**，开始作业配置。
3. 在**选择同步通道的源及目标实例**步骤中，配置同步链路连接信息。

   在这一步主要配置：

   - 同步作业名称

     同步作业名称没有唯一性要求，主要为了更方便识别具体的作业，建议选择一个有业务意义的作业名称，方便后续的链路查找及管理。

   - 源实例信息

- 实例类型：源 RDS 实例的实例类型，目前只支持 RDS For MySQL，这里选择**RDS实例**（MySQL）。
- 实例地区：源 RDS 实例所在区域。
- 实例ID：源 RDS 实例的实例ID。

- 目标实例信息
  - 实例类型：目标RDS 实例的实例类型，这里选择**RDS实例**（MySQL）。
  - 实例地区：目标RDS 实例所在区域。
  - 实例ID：目标RDS 实例的实例ID。



说明: 源跟目标 RDS 实例必须为两个不同的实例，选择**实例ID**时，下拉菜单中只列出对应阿里云账号下的 RDS For MySQL 实例。

当这些内容配置完成后，可以单击**授权白名单并进入下一步**。

4. 授权 RDS 实例白名单。

   这个步骤，主要是将数据传输服务器 IP 添加到同步 RDS 实例的白名单中。避免因为 RDS 设置了白名单，数据传输服务器连接不上 RDS 导致同步作业创建失败。

   为了保证同步作业的稳定性，在同步过程中，请勿将这些服务器 IP 从 RDS 实例的白名单中删除。

5. 当白名单授权后，单击**下一步**，进入同步账号创建。
6. 创建目标库上的同步账号。

   这个步骤主要是在目标 RDS 实例上创建一个同步账号，账号名字为：dtssyncwriter，在同步过程中，不能删除这个账号，否则会导致同步链路中断。

7. 单击**下一步**，选择同步对象。

   当创建完目标 RDS 实例的同步账号后，即进入同步对象的选择步骤。实时同步的同步对象的选择粒度可以支持到表级别，即您可以选择同步某些库或是同步某几张表。

   如果选择的同步对象为整个库，那么这个库中所有对象的结构变更操作（例如 create table，drop view 等），都会同步到目标库。

   如果选择的某张表，那么只有这个表的 drop/alter/truncate/rename table和create/drop index的操作会同步到目标库。

需要注意的是 rename table 操作可能导致同步数据不一致。例如同步对象只包含表 A，不包含表 B，如果同步过程中源实例执行了rename A to B的操作，那么改名后的B表的操作不会被同步到目标库。为了解决这个问题，可以选择同步表 A、B 对应的整个数据库。

当选择完同步对象后，即进入同步初始化配置。

8. 单击**下一步**，进行同步初始化配置。

   同步初始化配置，初始化是同步链路启动的第一步，它会将源实例中已经存在同步对象的结构及数据在目标实例中初始化，作为后续增量同步数据的基线数据。

   同步初始化类型细分为：结构初始化和全量数据初始化。默认情况下，需要选择**结构初始化**及**全量数据初始化**。

9. 单击**预检查并启动**，启动预检查。

   当上面所有选项配置完成后，即进入启动之前的预检查。

   当同步作业配置完成后，数据传输服务会进行限制预检查，当预检查通过后，可以单击**启动**，启动同步作业。

   当同步作业启动之后，即进入同步作业列表。此时刚启动的作业处于**同步初始化**状态。初始化的时间长度依赖于源实例中同步对象的数据量大小。当初始化完成后同步链路即进入**同步中**的状态，此时源跟目标实例的同步链路才真正建立完成。

18.3 数据迁移

18.3.1 数据迁移功能简介

本小节简单介绍数据传输服务提供的数据迁移功能支持的实例类型及数据源类型。

数据迁移支持实例类型

数据迁移支持的实例类型如表 *44:* 数据迁移支持实例类型所示。

表 44: 数据迁移支持实例类型

源实例	目的实例
本地 IDC 自建数据库	• 经典网络 RDS、VPC 网络 RDS 实例。 • 经典网络 MongoDB、VPC 网络 MongoDB 实例。 • 经典网络 Redis 实例、VPC 网络 Redis 实例。 • 经典网络 PetaData。 • 经典网络OceanBase。

源实例	目的实例
	• ECS 自建数据库。
ECS 自建数据库	• 经典网络 RDS、VPC 网络 RDS 实例。 • 经典网络 MongoDB 实例。 • 经典网络 PetaData。 • 经典网络 OceanBase。 • ECS 自建数据库。
RDS实例	• 经典网络RDS、VPC 网络 RDS 实例。 • 经典网络 PetaData。 • 经典网络 OceanBase。 • ECS 自建数据库。
经典网络 MongoDB 实例	• 经典网络 MongoDB 实例。 • ECS 自建数据库

数据迁移支持数据源

数据迁移功能，目前支持的迁移功能如表 *45:* 数据迁移支持数据源所示。

表 45: 数据迁移支持数据源

源数据库	结构迁移	全量迁移	增量迁移
Oracle->RDS For MySQL	支持	支持	不支持
Oracle->RDS For PPAS	支持	支持	支持
MySQL ->RDS For MySQL	支持	支持	支持
SQLServer->RDS For SQLServer	支持	支持	支持
PostgreSQL-> RDS For PostgreSQL	支持	支持	支持
MongoDB -> MongoDB	支持	支持	支持
Redis -> Redis	支持	支持	支持
MySQL->PetaData	不支持	支持	不支持

源数据库	结构迁移	全量迁移	增量迁移
MySQL->OCeanBase	支持	支持	支持

18.3.2 预检查项

本小节简单介绍数据迁移启动之间的预检查阶段检查的内容，失败原因及具体的修复方法。

18.3.2.1 源库连接性检查

检测数据传输服务器是否能够连通要迁移的源数据库。数据传输服务通过 JDBC 协议创建到源数据库的连接，当创建连接失败时，这个检查项即失败。

当源库连接性预检查失败时，可能是由于如下几种原因：

- 创建迁移任务时，提供的账号或密码不正确。

  检测方法：

  在任何一台网络能够跟源数据库互通的服务器上，使用创建迁移任务的账号/密码，通过客户端软件连接源数据库，确认是否能够正常连接。如果连接报错，且报错信息中含有Access deny，那么说明账号或密码有误。

  修复方法：

  在数据传输服务控制台修改迁移任务，将账号密码修改成正确的配置后重新预检查。
- 源库迁移账号限制了来源IP。

  检测方法：

  - 在一台网络能够跟源数据库连通的服务器上，使用创建迁移任务使用的账号/密码，通过客户端软件连接源数据库，能够正常连接。此时说明源数据库限制了来源 IP，只有允许的服务器可以连接，而数据传输服务器的 IP 不在授权列表，所以不能正常连接。
  - 如果源数据库为 MySQL 数据库，那么直接使用 mysql 客户端连接源数据库，执行`select host from mysql.user where user= ‘迁移账号’ ，password='迁移账号密码';`，如果查询结果不为%，那么说明数据传输服务器 IP 不在授权 IP 列表中，所以不能正常连接。
  - 如果是 SQL Server，那么检查 SQL Server 安装服务器上是否设置了防火墙，或者检查源数据库中有 endpoint 或 trigger 限制了访问来源 IP。
  - 如果是 Oracle，那么检查源数据库的配置文件*sqlnet.ora*，确认配置项**TCP.VALIDNODE_CHECKING**是否为**yes**，如果为**yes**，说明源数据库限制了访问来源 IP。

修复方法：

- 如果源数据库为 MySQL，那么在源数据库中执行grant all on . to '迁移账号'@'%' identified by '迁移密码';，给迁移账号重新授权。这个命令中的迁移账号用真实的迁移账号替换，迁移密码用真实的迁移密码替换。授权后重新进行预检查。
- 如果源数据库为 SQLServer，那么需要先关闭掉防火墙或禁用掉 trigger 后，重新进行预检查。
- 如果源数据库为 Oracle，那么先修改**TCP.VALIDNODE_CHECKING=no**并重启监听后，重新进行预检查。

- 源数据库安装服务器设置了防火墙。

  检测方法：如果源数据库安装服务器为 Linux，那么在 shell 中运行iptables -L，检查服务器是否配置了防火墙。如果安装服务器为window，那么在控制面板中查找到 Windows 防火墙，查看是否配置了防火墙。

  修复方法：

  先将防火墙关闭后，重新进行预检查。

- 数据传输服务器与源数据库网络不通

  如果不是上面三种情况中的任何一种，那么可能是由于数据传输服务器跟源数据库网络不连通，当处于这种情况时，请联系数据传输服务值班同学协助解决。

18.3.2.2 目标数据库连接性检查

检测数据传输服务器是否能够连通要迁移的目标数据库。数据传输服务通过 JDBC 协议创建到目标数据库的连接，当创建连接失败时，这个检查项即失败。

当目标库连接性预检查失败时，可能是由于如下的两种原因：

- 创建迁移任务时，提供的账号或密码不正确。

  检测方法：

  在任何一台网络能够跟目标数据库互通的服务器上，使用创建迁移任务使用的账号/密码，通过客户端软件连接源数据库，确认是否能够正常连接。如果连接报错，且报错信息中含有 Access deny，那么说明账号或密码有误。

  修复方法：

  在数据传输服务控制台修改迁移任务，将账号密码修改成正确的配置后重新预检查。

- 数据传输服务器与目标数据库网络不通。

  如果不是迁移账号密码不正确的话，那么可能是由于数据传输服务器跟目标数据库网络不连通，当处于这种情况时，请联系数据传输服务值班同学协助解决。

18.3.2.3 源库binlog检查

源库binlog是否开启检查

这个配置项只有当进行 MySQL->MySQL 增量迁移时，才会进行检查。这个检查项主要检查源数据库是否开启 binlog 日志。如果检查项失败，那么说明源数据库没有开启 binlog 。

修复方式：如果binlog没有打开，那么需要修改源数据库配置文件中的log_bin，开启二进制日志log_bin=mysql_bin。配置完成后重启源数据库后，重新进行预检查。

源库binlog模式检查

这个配置项只有当进行 MySQL->MySQL 增量迁移时，才会检查。这个检查项主要检查源数据库的 binlog 模式是否为ROW。如果检查失败，那么说明源数据库的binlog格式不为ROW。

如果检查项失败，那么在源数据库中执行set global binlog_format=ROW后，重新进行预检查。建议修改后重启 MySQL，否则连接中的 session 可能以非 ROW 模式继续写入，这会导致数据丢失。

源库binlog存在性检查

这个配置项只有当进行 MySQL->MySQL 增量迁移时，才会检查。这个检查项主要检查源数据库存在的binlog文件是否被删除。如果检查失败，那么说明源数据库的 binlog 文件不完整。

如果检查失败，那么在源数据库执行PURGE BINARY LOGS TO “未被删除的第一个binlog文件名”后，重新进行预检查。

具体 purge 的文件名参考预检查的修复方法。

Mysql源库binlog_row_image是否为FULL

这个配置项只有当进行 MySQL->MySQL 增量迁移时，才会检查。这个检查项主要检查源数据库的binlog_row_image是否为full，即是否记录全镜像。如果检查失败，那么说明源数据库的binlog未记录全镜像。

如果检查失败，那么在源库执行set global binlog_row_image=FULL命令后，重新进行预检查。

18.3.2.4 约束完整性检查

这个检查项主要检查迁移的对象列表中，有外键依赖关系的父子表是否都被迁移，防止破坏外键约束完整性。

当这个检查项失败时，失败原因为：待迁移表子表名字依赖的父表父表名字没有迁移。

对于这种错误的修复方式如下：

- 不进行约束完整性检查失败的子表的迁移，修改迁移任务配置，将这些约束完整性检查失败的子表从迁移列表中删除后，重新进行预检查。
- 进行约束完整性检查失败的子表对应的父表的迁移，修改迁移任务配置，将这些约束完整性检查失败的子表对应的父表添加到迁移列表中后，重新进行预检查。
- 删除约束完整性检查失败的子表的外键依赖，修改源数据库将这些子表的外键依赖删除掉，重新进行预检查。

18.3.2.5 federated类型的表的存在性检查

这个配置项只有当进行进行 MySQL->MySQL 增量迁移时，才会检查。这个检查项主要检查员数据库中是否存在增量迁移不支持的存储引擎。目前增量迁移不支持 federated和mrg_myisam 两种存储引擎。

当检查失败，且失败原因为**源数据库中存在引擎为 FEDERATED 的表**时，说明源数据库中要迁移的部分表的存储引擎为 federated ，所以导致检查失败。

当检查失败，且失败原因为**源数据库中存在引擎为 MRG_MYISAM 的表**时，说明源数据库中要迁移的部分表的存储引擎为 mrg_myisam，所以导致检查失败。

修复方法：

当检查失败时，对于这些存储引擎为 federated 或 mrg_myisam 的表，需要修改迁移任务，将这些表从迁移列表删除掉。然后单独创建一个迁移任务，对这些表进行结构迁移+全量数据迁移。

18.3.2.6 权限检查

源库权限检查

这个检查项主要检查源数据库的迁移账号是否有足够的权限可以进行数据迁移。每种数据库需要的迁移权限请参见迁移手册。

目标库权限检查

这个检查项主要检查源数据库的迁移账号是否有足够的权限可以进行数据迁移。每种数据库需要的迁移权限请参见迁移手册。

18.3.2.7 同名对象存在性检查

这个检查项主要检查目标库是否已经存在跟待迁移对象同步的对象。当这个检查项失败时，说明目标 RDS 实例中已经存在了跟待迁移的对象一样名字的对象，这会导致迁移失败。

当这个检查项失败时，失败原因为：目标库存在跟待迁移对象同名的结构对象迁移库名。

当这个检查项失败时，修复方式如下：

- 使用数据传输服务提供的库表映射功能，将待迁移对象迁移到目标库中的另外一个对象名的对象中。
- 删除或重命名目标库中跟迁移对象同名的对象。
- 修改迁移任务配置，将同名对象从迁移对象中删除掉，暂时不迁移这些同名对象。

18.3.2.8 schema存在性检查

主要检查待迁移的数据库在目标 RDS 实例中是否存在。如果不存在，那么数据传输服务会自动创建。但是对于如下几种情况，自动创建数据库会失败，所以直接报预检查失败。

- 数据库名中含有小写字母、数字、下划线、中划线的其他特殊字符。

  当遇到这种情况时，预检查的失败原因为：迁移的数据库**迁移库名**的名称不符合rds规范。

  修复方法：在 RDS 控制台的数据库管理页面中，创建满足rds规范的数据库，将新建库的读写权限赋给迁移账号。然后使用数据传输服务提供的库名映射功能将要迁移的数据库映射至新创建的数据库后，然后重新进行预检查。
- 数据库的字符集不为utf8、gbk、latin1、utf8mb4 中的一种。

  当遇到这种情况时，预检查的失败原因为：迁移的数据库**迁移库名**的字符集不满足 rds 的要求。

  修复方法：到 RDS 控制台的数据库管理页面中，创建满足 rds 规范的数据库，然后将新建库的读写权限赋给迁移账号。如果新创建的库名与待迁移的库名不同，那么使用数据传输服务提供的库名映射功能将要迁移的数据库映射至新创建的数据库后，重新进行预检查。
- 目标库迁移账号无迁移库的读写权限。

  当遇到这种情况时，预检查的失败原因为：您没有权限使用库**迁移库名**。

修复方法：先到 RDS 控制台的数据库管理页面中账号管理页面，修改权限，将迁移库的读写权限授予迁移账号后，重新进行预检查。

18.3.2.9 源库server-id检查

这个配置项只有当进行 MySQL->MySQL 增量迁移时，才会进行检查。这个检查项主要检查源数据库的**server-id**是否设置为大于 1 的整数。

当检查失败时，只要在源数据库执行set global server_id= ‘某个不为1的数’ ，然后重新进行预检查。

18.3.2.10 源库版本号检查

这个检查项主要检查源数据库的版本号，数据传输服务是否支持。数据传输服务支持的各种源数据库的版本如表 *46:* 源库类型及其版本号所示。

表 46: 源库类型及其版本号

源数据库类型	支持的版本号
MySQL	5.0、5.1、5.5、5.6，当进行增量迁移时，仅支持5.1、5.5、5.6。
SQL Server	2005、2008、2008R2、2012。
Oracle	10g、11g。

当版本检查失败时，只能将源数据库升降级到数据传输服务支持的版本范围后，再进行预检查。

18.3.3 从本地Oracle迁移至RDS for MySQL

本小节简单介绍使用 DTS 进行 Oracle->RDS For MySQL 数据迁移的任务配置流程。

使用数据传输DTS可以将本地的Oracle 的数据迁移至 RDS MySQL 实例。DTS 可以实现结构定义及全量数据的迁移，为保证迁移数据一致性，建议迁移过程中，本地的Oracle 停止写入。

迁移类型

对于 Oracle->RDS For MySQL 的迁移，目前还不支持不停服迁移。只能支持结构定义及全量数据的迁移。

结构迁移及全量数据迁移的限制如下：

- 结构迁移

DTS 会将迁移对象的结构定义迁移到目标实例。目前 DTS 支持结构迁移的对象有：表。其他对象如视图、同义词、触发器、存储过程、存储函数、包、自定义类型等暂不支持。

- 全量迁移

  DTS 会将源数据库迁移对象在的数据全部迁移到目标实例。如果迁移过程中，本地 Oracle 库有数据写入的话，那么这些增量数据不一定能够被迁移到 MySQL 中。所以，为保证迁移数据一致性，在迁移过程中，Oracle 实例需停止写入。

迁移权限要求

当使用 DTS 进行 Oracle 迁移时，在不同迁移类型情况下，源跟目标数据库的迁移账号权限要求如表 *47:* 迁移类型及权限要求所示。

表 47: 迁移类型及权限要求

迁移类型	结构迁移	全量迁移
本地 Oracle 实例	schema 的 owner	schema 的 owner
目的 RDS MySQL 实例	待迁入 db 的读写权限	待迁入 db 的读写权限

迁移流程

数据传输服务在进行 Oracle->RDS For MySQL 的数据迁移时，为了解决对象间的依赖，提高迁移成功率。结构对象及数据的迁移顺序如下：

1. 结构对象迁移：表的迁移。
2. 全量数据迁移。
3. 结构对象迁移：外键迁移。

全量数据迁移完成后，任务列表中的迁移进度为：**结构迁移**100%，**全量迁移100%**，但迁移状态为**迁移中**。此时迁移任务正在进行步骤 （3）外键的迁移，请勿手动结束任务，否则会造成迁移数据不一致。

数据类型映射关系

由于 Oracle 跟 MySQL 的数据类型并不是一一对应的，所以 DTS 在进行结构迁移时，会根据两种数据库类型的数据类型定义，进行类型映射，数据类型映射关系如表 *48:* 数据类型映射关系所示。

表 48: 数据类型映射关系

Oracle数据类型	MySQL数据类型	DTS是否支持
varchar2(n [char/byte])	varchar(n)	支持
nvarchar2[(n)]	national varchar[(n)]	支持
char[(n [byte/char])]	char[(n)]	支持
nchar[(n)]]	national char[(n)]	支持
number[(p[,s])]	decimal[(p[,s])]	支持
float(p)]	double	支持
long	longtext	支持
date	datetime	支持
binary_float	decimal(65,8)	支持
binary_double	double	支持
timestamp[(fractional_seconds_precision)]	datetime[(fractional_seconds_precision)]	支持
timestamp[(fractional_seconds_p recision)]with local time zone	datetime[(fractional_seconds_precision)]	支持
timestamp[(fractional_seconds_p recision)]with local time zone	datetime[(fractional_seconds_precision)]	支持
clob	longtext	支持
nclob	longtext	支持
blob	longblob	支持
raw	varbinary(2000)	支持
long raw	longblob	支持
bfile	-	不支持
interval year(year_precision) to mongth	-	不支持
interval day(day_precision) to second[(fractional_seconds_p recision)]	-	不支持

说明:

- 对于 char 类型，当 char(n) 的定义长度 n 超过 255 时，DTS 会自动将类型转换为 varchar(n) 。
- 由于 MySQL 本身不支持类似 Oracle 中的 bfile、interval year to month、interval day to second 这种数据类型，所以 DTS 在进行结构迁移时，这三种类型不会进行转化，迁移时如果表中含有这三种类型，会导致结构迁移失败，您可以根据业务需求，将这三种类型修改成 MySQL 中对应的数据类型即可。
- 由于 MySQL 的 timestamp 类型不包含时区，而 Oracle 的 timestamp with [local] time zone 带时区，所以 DTS在迁移这两种类型的数据时，会将其转换成 UTC 时区后，存入MySQL。

迁移步骤

下面详细介绍如何使用 DTS 将本地的 Oracle 迁移到 RDS for MySQL 的任务配置流程。

创建 RDS 实例数据库

在数据迁移过程中，如果待迁移的数据库在目标 RDS 实例中不存在，那么 DTS 自动会创建。但是对于如下两种情况，您需要在配置迁移任务之前，手动创建数据库。

- 数据库名称不符合：RDS 定义规范（由小写字母、数字、下划线、中划线组成，字母开头，字母或数字结尾，最长 64 个字符）。
- 待迁移数据库，在 Oracle 跟目标 RDS 实例中名称不同。

对于这两种情况，您需要在配置迁移任务之前，先在 RDS 控制台完成数据库创建。具体请参考 RDS 用户指南中的创建 RDS 实例。

创建迁移账号

迁移任务配置，需要提供 Oracle 数据库及目标 RDS 实例的迁移账号。迁移账号所需权限请参见上文的迁移权限要求。

如果您的 Oracle 实例的迁移账号尚未创建，那么您可以参考 *Oracle Grant*语法说明，创建满足要求的迁移账号。

RDS For MySQL 迁移账号的创建及授权操作请参见《*RDS*使用手册》。

配置迁移任务

当上面的所有前置条件都配置完成后，就可以开始迁移任务配置。下面详细介绍下具体的迁移步骤。

1. 登录数据传输控制台，单击右上角的**创建迁移任务**，正式开始任务配置。
2. 配置本地 Oracle 及目标 RDS 实例的连接信息。

   这个步骤主要配置迁移任务名称，Oracle 连接信息及目标 RDS 实例连接信息。其中：

   - 任务名称

     DTS 为每个任务自动生成一个任务名称，任务名称没有唯一性要求。您可以根据需要修改任务名称，建议为任务配置具有业务意义的名称，便于后续的任务识别。
   - 源实例信息
     - 实例类型：选择**有公网IP的自建数据库**。
     - 数据库类型：选择**Oracle**。
     - 主机名或IP地址：配置 Oracle 访问地址，这个地址必须为公网访问方式。
     - 端口：Oracle 实例的监听端口。
     - SID：Oracle 实例的 SID。
     - 账号：Oracle 实例的连接账号。
     - 密码：上面 Oracle 连接账号对应的密码。
   - 目标实例信息
     - 实例类型：选择**RDS实例**
     - RDS 实例 ID：配置迁移的目标 RDS 实例的实例 ID。 DTS 支持经典网络、VPC 网络的 RDS 实例。
     - 账号：RDS 实例的连接账号。
     - 密码：上面 RDS 实例连接账号对应的密码。

   当配置完连接信息后，单击右下角**授权白名单并进入下一步**，进行白名单授权。这个步骤 DTS 会将 DTS 服务器的 IP 地址添加到目标 RDS 实例的白名单中，避免因为 RDS 实例设置了白名单，导致 DTS 服务器连接不上 RDS 实例导致迁移失败。
3. 迁移对象及迁移类型配置。

   迁移类型包括：结构迁移、全量数据迁移。默认选择**结构迁移+全量数据迁移**。

   迁移对象选择您需要迁移的对象。迁移对象选择的粒度可以为：库、表、列三个粒度。

默认情况下，对象迁移到 RDS 实例后，对象名跟 Oracle 一致。如果您迁移的对象在源实例跟目标实例上名称不同，那么需要使用 DTS 提供的对象名映射功能，详细使用方式请参考库表列映射。

当配置完迁移对象及迁移类型后，即进入任务启动前的预检查步骤。

4. 任务预检查。

   在迁移任务正式启动之前，会先进行前置预检查，只有预检查通过后，才能成功启动迁移。具体的预检查内容请参见预检查简介。

   如果预检查失败，那么可以单击具体检查项后的按钮，查看具体的失败详情，并根据失败原因修复后，重新进行预检查。

   图 361: 预检查



5. 启动迁移任务。

   当预检查通过后，我们可以启动迁移任务，任务启动后，可以到任务列表中查看具体的迁移状态及进度。

至此，完成将本地 Oracle 数据库到 RDS For MySQL 的数据迁移任务配置。

18.3.4 从本地MySQL迁移至RDS for MySQL

本小节简单介绍使用 DTS 进行MySQL->RDS For MySQL 数据迁移的任务配置流程。

使用数据传输 DTS 可以将本地 MySQL 实例的数据迁移到 RDS for MySQL 实例。DTS 提供的增量数据同步功能，可以实现在本地应用不停服的情况下，平滑完成 MySQL 数据的迁移工作。

迁移类型

对于 MySQL->RDS For MySQL 数据迁移，DTS 支持结构迁移、全量数据迁移及增量数据迁移，这个迁移类型的功能及限制如下：

- 结构迁移

  数据传输会将迁移对象的结构定义迁移到目标实例。目前 DTS 支持结构迁移的对象有：表、视图、触发器、存储过程、存储函数。

- 全量数据迁移

  全量迁移将 MySQL 实例的存量数据全部迁移到目标 RDS 实例。

  如果您只进行全量数据迁移，那么迁移过程中本地 MySQL 实例新增的业务写入不会被同步到目标RDS实例。

  如果您还选择了增量数据迁移，那么全量迁移过程中，为了保证迁移数据一致性，无主键的非事务表会被锁定，锁定期间这些表无法写入，锁定时长依赖于这些表的数据量大小，在这些无主键非事务表迁移完成后，锁才会释放。

- 增量数据迁移

  增量迁移会将迁移过程中，本地 MySQL 实例的增量变更数据同步到目标 RDS 实例，增量数据迁移可以实现本地 MySQL 实例在迁移过程中产生的增量数据被实时同步到目标库，最终实现本地 MySQL 同目标 RDS 实例进入动态数据同步的过程。

迁移限制

从本地 MySQL 迁移到 RDS for MySQL 的迁移限制如下：

- 迁移过程中，不支持 DDL 操作。
- 结构迁移不支持 event 的迁移。
- 如果使用了对象名映射功能后，依赖这个对象的其他对象可能迁移失败。
- 当选择增量迁移时，源 MySQL 实例需要开启 binlog。
- 当选择增量迁移时，源库的 binlog_format 要为row。

- 当选择增量迁移且源 MySQL 如果为 5.6 版本时，它的 binlog_row_image 必须为 full。

迁移权限要求

当使用 DTS 进行 MySQL->RDS For MySQL 迁移时，在不同迁移类型，对源跟目标 MySQL 实例的迁移账号权限要求如表 *49:* 迁移类型及权限要求所示。

表 49: 迁移类型及权限要求

迁移类型	结构迁移	全量迁移	增量迁移
本地 MySQL 实例	select	select	superselectreplication slavereplication client
目的 RDS 实例	读写权限	读写权限	读写权限

迁移流程

DTS 在进行 MySQL->RDS For Mysql 数据迁移时，为了解决对象间的依赖关系，提高迁移成功率。结构对象及数据的迁移顺序如下：

1. 结构对象：表、视图的迁移。
2. 全量数据迁移。
3. 结构对象：存储过程、函数、触发器、外键的迁移。
4. 增量数据迁移。

如果任务没有选择增量数据迁移，那么当全量数据迁移完成后，任务列表中的迁移进度为：**结构迁移**100%，**全量迁移100%**，迁移状态为**迁移中**。此时迁移任务正在进行步骤（3）中的对象的迁移。此时，请勿手动结束任务，否则会造成迁移数据丢失。

迁移步骤

下面详细介绍使用 DTS 将本地的 MySQL 迁移到 RDS for MySQL 的任务配置流程。

RDS 实例数据库创建

在数据迁移过程中，如果待迁移的数据库在目标 RDS 实例中不存在，那么 DTS 自动会创建。但是对于如下两种情况，您需要在配置迁移任务之前，手动创建数据库。

- 数据库名称不符合：RDS　定义规范（由小写字母、数字、下划线、中划线组成，字母开头，字母或数字结尾，最长 64 个字符）。
- 待迁移数据库，在 Oracle 跟目标 RDS 实例中名称不同。

- 数据库名称不符合：RDS 定义规范（由小写字母、数字、下划线、中划线组成，字母开头，字母或数字结尾，最长 64 个字符）。
- 待迁移数据库，在 Oracle 跟目标 RDS 实例中名称不同。

对于这两种情况，您需要在配置迁移任务之前，先在 RDS 控制台完成数据库创建。具体请参考 RDS 用户指南中的创建 RDS 实例。

迁移账号创建

迁移任务配置，需要提供本地 MySQL 数据库及目标 RDS 实例的迁移账号。迁移账号所需权限请参见上文的迁移权限要求。

如果源 MySQL 实例迁移账号尚未创建，那么您可以参考 *Grant*语法说明，创建满足权限要求的迁移账号。

如果目标 RDS 实例迁移账号尚未创建，那么您可以参考 *RDS*账号创建流程，创建对待迁移数据库有读写权限的迁移账号。

其他准备工作

如果需要进行增量迁移，那么需要确认源库的 binlog 是否开启，格式是否满足要求。具体确认项如下。

1. 确认源库的 binlog 是否开启，如图 *362:* 确认*binlog*是否开启所示。

   图 362: 确认binlog是否开启

   ```
   mysql> show global variables like "log_bin";
   +---------------+-------+
   | Variable_name | Value |
   +---------------+-------+
   | log_bin       | ON    |
   +---------------+-------+
   1 row in set (0.04 sec)
   ```

   如果这里面查询出来的**log_bin=OFF**，那么说明源库没有开启 binlog ，这个时候如果需要使用增量迁移的话，需要参考如下方法修改本地 MySQL 的 binlog 配置，打开 binlog 日志。

   a. 开启二进制日志，修改**log_bin=mysql_bin**。
   b. 设置 binlog 模式为 row，修改**binlog_format=row**。
   c. 配置 server_id 大于 1，即 server_id = 某一个大于 1 的整数。

d. 如果版本为 5.6 ，修改**binlog_row_image=full**。

e. 修改完成后，重启 MySQL 进程。

2. 确认源库的 binlog 格式为 row 模式，如图 *363: 确认binlog格式*所示。

图 363: 确认binlog格式

```
mysql> show global variables like "binlog_format";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| binlog_format | ROW   |
+---------------+-------+
1 row in set (0.04 sec)
```

使用上面的命令确认本地 MySQL 的 binlog 格式是否为 row ，如果查询出来的结果不为 ROW 的话，那么通过如图 *364: 设置binlog格式*所示的参数设置命令将格式修改为 row。

图 364: 设置binlog格式

```
mysql> set global binlog_format=ROW;
Query OK, 0 rows affected (0.04 sec)
```

当修改完成后，最好将已有的老连接 KILL 掉重连，否则可能出现其他的连接依然使用 statement 格式记录 binlog 日志。

3. 当本地 MySQL 版本大等于 5.6.2 时，确认源库的**binlog_row_image=full**，如图 *365: 查询binlog_row_image*所示。

图 365: 查询binlog_row_image

```
mysql> show global variables like "binlog_row_image";
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| binlog_row_image | FULL  |
+------------------+-------+
1 row in set (0.04 sec)
```

如果查询出来的结果不为 FULL 的话，那么通过如图 *366: 设置binlog_row_image*所示的参数设置命令将其修改为full。

图 366: 设置binlog_row_image

```
mysql> set global binlog_row_image=full;
Query OK, 0 rows affected (0.03 sec)
```

迁移任务配置

当数据库、迁移账号都创建完成后，就可以开始配置迁移任务了。下面详细介绍下具体的配置步骤。

1. 登录数据传输控制台，单击右上角的**创建迁移任务**，开始任务配置。
2. 配置本地 MySQL 及目标 RDS 实例连接信息，如图 *367: 配置相关信息*所示。

   在这个步骤中，主要配置迁移任务名称，迁移源实例及目标实例连接信息。其中：

   - 任务名称

     DTS 为每个任务自动生成一个任务名称，任务名称没有唯一性要求。您可以根据需要修改任务名称，建议为任务配置具有业务意义的名称，便于后续的任务识别。
   - 源实例连接信息
     - 实例类型：选择**有公网IP的自建数据库**。
     - 数据库类型：选择**MySQL**。
     - 主机名或 IP 地址：配置本地 MySQL 数据库访问地址，这个地址必须为公网访问方式。
     - 端口：MySQL 实例监听端口。
     - 账号：MySQL 实例访问账号。
     - 密码：上面 MySQL 访问账号对应的密码。
   - 目标 RDS 实例连接信息
     - 实例类型：选择**RDS实例**
     - RDS 实例 ID：配置迁移的目标 RDS 实例的实例 ID。 DTS 支持经典网络、VPC 网络的 RDS 实例。
     - 账号：RDS 实例的连接账号。

- 密码：上面 RDS 实例连接账号对应的密码。

图 367: 配置相关信息



3. 配置迁移对象及迁移类型。

   - 迁移类型

     迁移类型包括结构迁移、全量数据迁移及增量数据迁移。

     如果要做全量迁移，那么选择结构迁移 + 全量数据迁移。

     如果要做不停机迁移，那么选择结构迁移 + 全量数据迁移 + 增量数据迁移。

   - 迁移对象

     迁移对象，需要选择您要迁移的对象。迁移对象选择的粒度可以为：库、表、列三个粒度。

     默认情况下，迁移对象迁移到目标库后，对象名跟源库一致。如果您迁移的对象在源实例跟目标实例上名称不同，那么需要使用 DTS 提供的对象名映射功能，详细使用方式请参考库表列映射。

   当配置完迁移对象及迁移类型后，即进入任务启动前的预检查步骤。

4. 预检查。

在迁移任务正式启动之前，会先进行前置预检查，只有预检查通过后，才能成功启动迁移。具体的预检查内容请参见预检查简介。

如果预检查失败，那么可以单击具体检查项后的按钮，查看具体的失败详情，并根据失败原因修复后，重新进行预检查。

图 368: 预检查



5. 启动迁移任务。

当预检查通过后，我们可以启动迁移任务，任务启动后，可以到任务列表中查看具体的迁移状态及进度。

增量数据迁移是个动态同步的过程，所以建议在增量迁移达到无延迟状态时，在目标数据库上进行业务验证，如果验证成功，那么可以停掉迁移任务，然后将业务切换到目标数据库。

至此，完成将本地 MySQL 数据库到 RDS For MySQL 的数据迁移任务配置。

18.3.5 从本地SQL Server迁移至RDS for SQL Server

本小节简单介绍使用 DTS 进行 SQLServer->RDS For SQLServer 数据迁移的任务配置流程。

使用数据传输 DTS 可以将本地 SQLServer 实例的数据迁移到 RDS For SQLServer 实例。SQLServer 迁移支持增量数据同步功能，可以实现在本地应用不停服的情况下，平滑完成 SQLServer 数据库的迁移工作。

迁移类型

对于 SQL Server->RDS for SQL Server 数据迁移，DTS 支持结构迁移、全量数据迁移及增量数据迁移，这个迁移类型的功能及限制如下：

- 结构迁移

  DTS 将迁移对象的结构定义迁移到目标实例。目前 DTS 支持结构迁移的对象有：表、视图、表触发器、同义词、SQL 存储过程、SQL 函数、plan guid、自定义类型、rule、default。

- 全量数据迁移

  全量迁移将 SQLServer 实例的存量数据全部迁移到目标 RDS 实例。如果您只进行全量数据迁移，那么迁移过程中本地 SQLServer 实例新增的业务写入不会被同步到目标 RDS 实例。

  如果您还选择了增量数据迁移，那么迁移过程中，本地 SQLServer 实例的增量写入数据会被同步到目标RDS实例。

迁移限制

从本地 SQL Server 迁移到 RDS for SQL Server 的迁移限制如下：

- 迁移过程中，不支持 DDL 操作。
- 结构迁移不支持 assemblies、库级存储过程、service broker、全文索引、全文目录、分布式schema、分布式函数、CLR 标量函数、CLR 标值函数、内部表、聚合函数、系统的迁移。
- 如果使用了对象名映射功能后，依赖这个对象的其他对象可能迁移失败。
- 增量迁移，只支持含有聚簇索引，且聚簇索引为唯一键的表。
- 增量迁移，不支持只更新大字段的 update 语句的同步。
- 增量迁移，不支持含有计算列的表。

迁移权限要求

当使用 DTS 进行 SQLServer 迁移时，不同迁移类型，对本地 SQLServer 数据库及目标 RDS For SQL Server 实例的迁移账号权限要求如表 *50:* 迁移类型及要求所示。

表 50: 迁移类型及要求

迁移类型	结构迁移	全量迁移
本地 SQLServer 实例	select	select
目的 RDS 实例	读写权限	读写权限

迁移流程

数据传输服务在进行 SQLServer 上云迁移时，为了解决对象间的依赖，提高迁移成功率。

结构对象及数据的迁移顺序如下：

1. 进行结构对象：表、视图、同义词、自定义类型、rule、default、plan guid 的迁移。
2. 全量数据迁移。
3. 进行结构对象 SQL 存储过程、SQL 函数、触发器、外键的迁移。
4. 增量数据迁移。

如果任务没有选择增量数据迁移，那么当全量数据迁移完成后，任务列表中的迁移进度为：**结构迁移**100%，**全量迁移100%**，迁移状态为**迁移中**。此时迁移任务正在进行步骤（3）中的对象的迁移。此时，请勿手动结束任务，否则会造成迁移数据丢失。

如果任务没有选择增量数据迁移，那么当全量数据迁移完成后，任务列表中的迁移进度为：**结构迁移100%**，**全量迁移100%**，迁移状态为**迁移中**。此时迁移任务正在进行步骤（3）中的对象的迁移。此时，请勿手动结束任务，否则会造成迁移数据丢失。

迁移步骤

下面详细介绍使用 DTS 将本地的 SQLServer 迁移到 RDS forSQLServer 的任务配置流程。

创建RDS 实例数据库

在数据迁移过程中，如果待迁移的数据库在目标 RDS 实例中不存在，那么 DTS 自动会创建。但是对于如下两种情况，您需要在配置迁移任务之前，手动创建数据库。

- 数据库名称不符合：RDS 定义规范（由小写字母、数字、下划线、中划线组成，字母开头，字母或数字结尾，最长 64 个字符）。
- 待迁移数据库，在 Oracle 跟目标 RDS 实例中名称不同。

对于这两种情况，您需要在配置迁移任务之前，先在 RDS 控制台完成数据库创建。具体请参考 RDS 用户指南中的创建 RDS 实例。

创建迁移账号

迁移任务配置，需要提供本地 SQLServer 数据库及目标 RDS 实例的迁移账号。迁移账号所需权限请参见上文的迁移权限要求。

如果本地 SQLServer 实例迁移账号尚未创建，那么您可以参考*SQLServer User*创建，创建满足权限要求的迁移账号。

如果目标 RDS 实例迁移账号尚未创建，那么您可以参考 RDS账号创建流程，创建对待迁移数据库有读写权限的迁移账号。

其他准备工作

如果您需要进行不停机迁移，那么还需要设置本地 SQLServer 日志格式为 full。

如果本地 SQLServer 数据库的日志格式不为 full，那么需要通过下面两个步骤设置：

1. 在源数据库执行alter database database_name set recovery_model_desc='full';，其中**database_name**为需要迁移的数据库名。
2. 为了保证开启完整日志生效，需要在源数据库进行一次日志备份，在源数据库执行BACKUP LOG database_name to DISK=backup_place WITH init;，其中，**database_name** 为待迁移的数据库名，**backup_place**为备份文件存储的地址。

迁移任务配置

当数据库、迁移账号都创建完成后，就可以开始配置迁移任务了。下面详细介绍下具体的配置步骤。

1. 登录数据传输控制台，单击右上角的**创建迁移任务**，开始任务配置。
2. 配置本地 SQLServer 及目标 RDS 实例连接信息，如图 *369:* 配置连接信息所示。

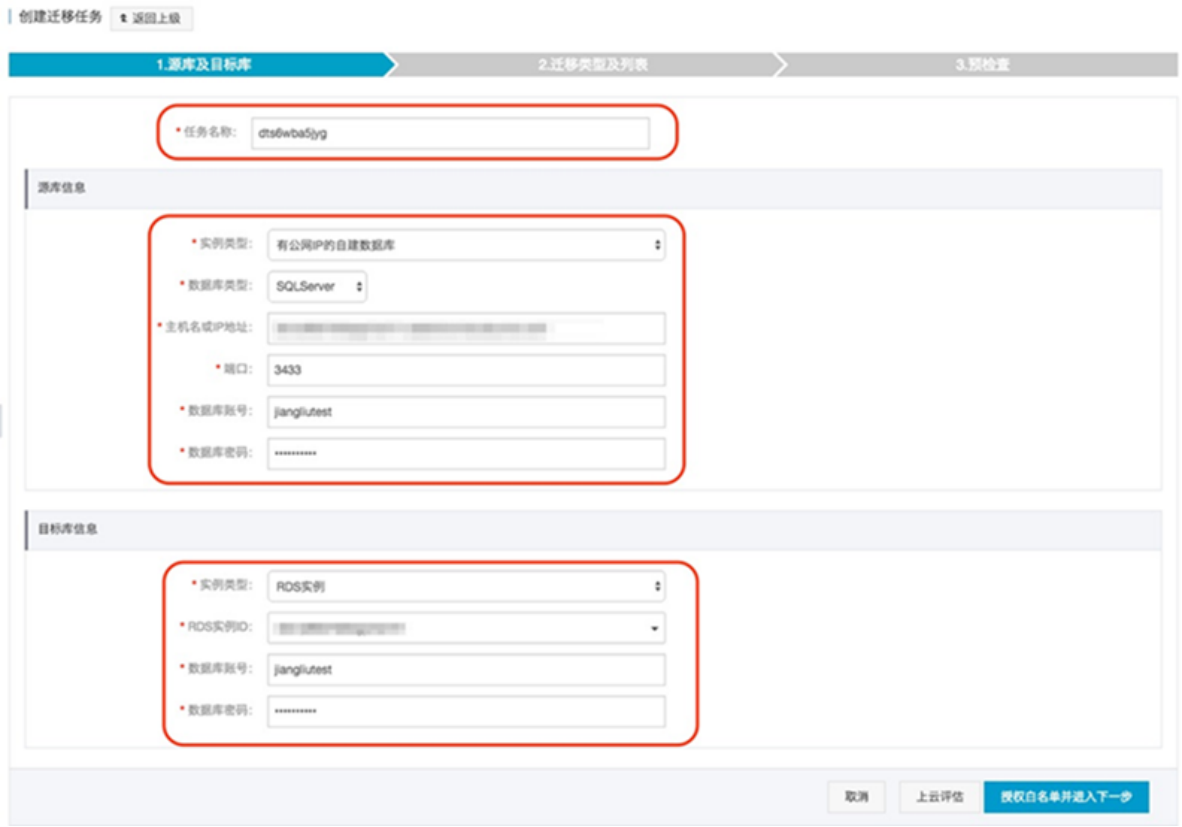
   在这个步骤中，主要配置迁移任务名称，迁移源实例及目标实例连接信息。其中：

   - 任务名称

     DTS 为每个任务自动生成一个任务名称，任务名称没有唯一性要求。您可以根据需要修改任务名称，建议为任务配置具有业务意义的名称，便于后续的任务识别。
   - 源实例连接信息
     - 实例类型：选择**有公网IP的自建数据库**。
     - 数据库类型：选择**SQLServer**。
     - 主机名或 IP 地址：配置本地 SQLSerever 数据库访问地址，这个地址必须为公网访问方式。
     - 端口：SQLServer 实例监听端口。
     - 数据库账号：SQLServer 实例访问账号。
     - 数据库密码：上面 SQLServer 访问账号对应的密码。
   - 实例类型：选择**RDS实例**

- RDS 实例 ID：配置迁移的目标 RDS 实例的实例 ID。 DTS 支持经典网络、VPC 网络的 RDS 实例。
- 账号：RDS 实例的连接账号。
- 密码：上面 RDS 实例连接账号对应的密码。

图 369: 配置连接信息



3. 配置迁移对象及迁移类型。

- 迁移类型

  迁移类型包括结构迁移、全量数据迁移及增量数据迁移。

  如果要做全量迁移，那么选择结构迁移 + 全量数据迁移。

  如果要做不停机迁移，那么选择结构迁移 + 全量数据迁移 + 增量数据迁移。

- 迁移对象

  迁移对象，需要选择您要迁移的对象。迁移对象选择的粒度可以为：库、表、列三个粒度。

默认情况下，迁移对象迁移到目标库后，对象名跟源库一致。如果您迁移的对象在源实例跟目标实例上名称不同，那么需要使用 DTS 提供的对象名映射功能，详细使用方式请参考库表列映射。

当配置完迁移对象及迁移类型后，即进入任务启动前的预检查步骤。

4. 任务预检查。

在迁移任务正式启动之前，会先进行前置预检查，只有预检查通过后，才能成功启动迁移。具体的预检查内容请参见预检查简介。

如果预检查失败，那么可以单击具体检查项后的按钮，查看具体的失败详情，并根据失败原因修复后，重新进行预检查。

图 370: 预检查



5. 启动迁移任务。

当预检查通过后，我们可以启动迁移任务，任务启动后，可以到任务列表中查看具体的迁移状态及进度。

增量数据迁移是个动态同步的过程，所以建议在增量迁移达到无延迟状态时，在目标数据库上进行业务验证，如果验证成功，那么可以停掉迁移任务，然后将业务切换到目标数据库。

至此，完成将本地 SQLServer 数据库到 RDS For SQLServer 的数据迁移任务配置。

18.3.6 SQL Server不停机迁移

本小节简单介绍使用 DTS 进行 SQLServer->RDS For SQLServer 不停机迁移的任务配置流程。

数据传输 DTS 支持 SQLServer 增量数据迁移，通过增量数据迁移可以实现本地 SQLServer 业务不停服的情况下，完成 SQLServer 到 RDS For SQLServer 数据库的迁移。

迁移类型

对于 SQLserver->RDS For SQLserver 数据迁移，DTS 支持结构迁移、全量数据迁移及增量数据迁移，这个迁移类型的功能及限制如下：

- 结构迁移

  DTS 将迁移对象的结构定义迁移到目标实例。目前 DTS 支持结构迁移的对象有：表、视图、表触发器、同义词、SQL 存储过程、SQL 函数、plan guid、自定义类型、rule、default。

- 全量数据迁移

  全量迁移将 SQLServer 实例的存量数据全部迁移到目标 RDS 实例。

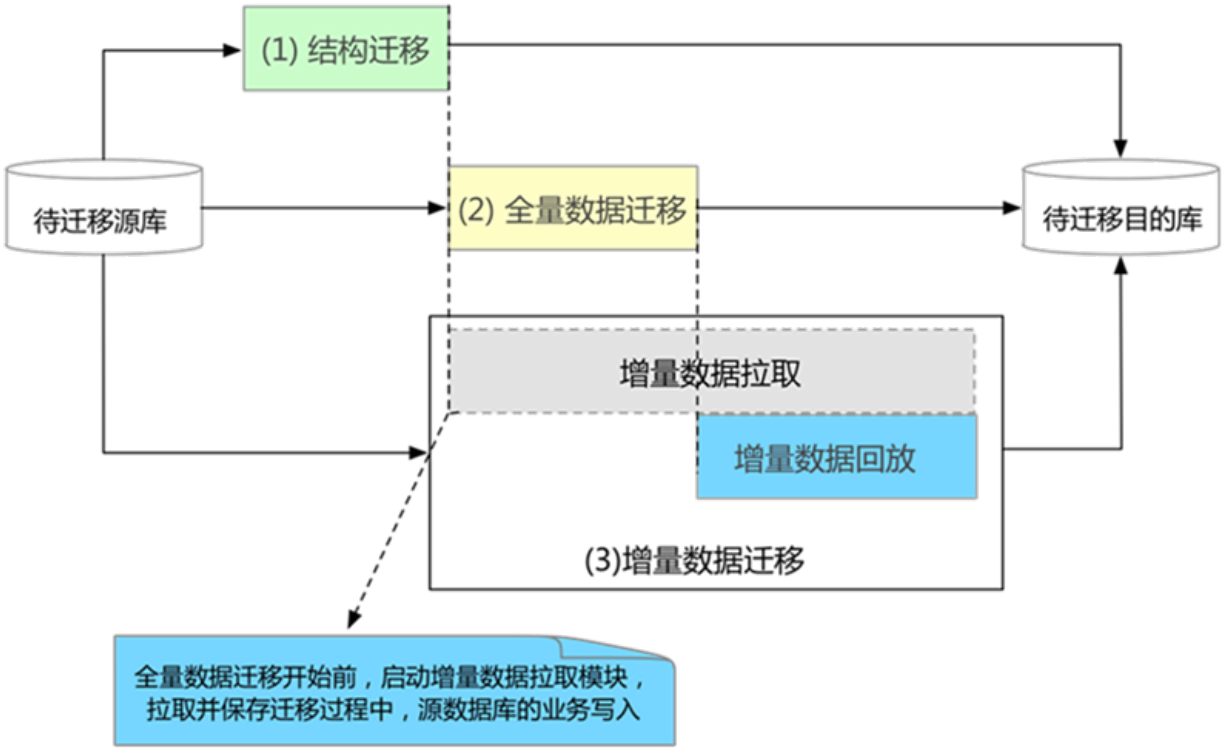
- 增量数据迁移

  将迁移过程中产生的增量数据迁移到目标库，通过增量数据迁移可以保证源数据库迁移过程产生的增量数据可以被同步到目标数据库，最终保证两个数据库的数据一致。

不停服迁移原理

不停服迁移原理如图 *371:* 不停服迁移原理所示。

图 371: 不停服迁移原理



为保证迁移过程中，源数据库业务写入的所有变更数据能够被迁移到目标库，在全量迁移之前，数据传输后台就会开启增量数据拉取模块，拉取并保存迁移过程中，源数据库的业务写入。当全量数据迁移完成后，会启动增量写入模块，开始回放这些增量数据，从而实现源跟目标实例增量数据实时同步。

当迁移进入增量迁移阶段时，增量数据迁移是一个动态的过程，除非您手动停止迁移，否则会一直保持两边数据实时同步，所以建议当增量迁移无延迟时，可以进行源数据库和目标数据库的数据校验，并在目标数据库进行业务测试，当这些验证都通过后，可以停止迁移任，然后将业务切换到目标数据库上。通过这种迁移方式，可实现在数据迁移过程中，应用停服时间降低到分钟级别。

迁移限制

迁移限制如下：

- 结构迁移不支持 assemblies、库级存储过程、service broker、全文索引、全文目录、分布式schema、分布式函数、CLR 标量函数、CLR 标值函数、内部表、聚合函数、系统的迁移。
- 如果使用了对象名映射功能后，依赖这个对象的其他对象可能迁移失败。
- 增量迁移，不支持 DDL 同步。
- 增量迁移，只支持含有聚簇索引，且聚簇索引为唯一键的表。
- 增量迁移，不支持只更新大字段的 update 语句的同步。
- 增量迁移，不支持含有计算列的表。

- 一个增量迁移任务，只支持一个数据库的迁移。如果同时有多个数据库需要进行增量数据迁移，那么需要创建多个迁移任务。

迁移权限要求

当使用数据传输进行 SQLServer 迁移时，在不同迁移类型情况下，源跟目标数据库的迁移账号权限要求如表 *51:* 迁移类型及要求所示。

表 51: 迁移类型及要求

迁移类型	结构迁移	全量数据迁移	增量数据迁移
本地 SQLServer 实例	select	select	sysadmin
目的 RDS 实例	读写权限	读写权限	读写权限

迁移准备

源数据库的日志格式必须为 full，如果源数据库的日志格式不为 full，那么需要通过下面两个步骤设置：

1. 在源数据库执行alter database database_name set recovery_model_desc='full';，其中**database_name**为需要迁移的数据库名。
2. 为了保证开启完整日志生效，需要在源数据库进行一次日志备份，在源数据库执行BACKUP LOG database_name to DISK=backup_place WITH init;，其中，**database_name** 为待迁移的数据库名，**backup_place**为备份文件存储的地址。

迁移步骤

创建RDS实例数据库

在数据迁移过程中，如果待迁移的数据库在目标 RDS 实例中不存在，那么 DTS 自动会创建。但是对于如下两种情况，您需要在配置迁移任务之前，手动创建数据库。

- 数据库名称不符合：RDS 定义规范（由小写字母、数字、下划线、中划线组成，字母开头，字母或数字结尾，最长 64 个字符）。
- 待迁移数据库，在 Oracle 跟目标 RDS 实例中名称不同。

- 数据库名称不符合：RDS 定义规范（由小写字母、数字、下划线、中划线组成，字母开头，字母或数字结尾，最长 64 个字符）。
- 待迁移数据库，在 Oracle 跟目标 RDS 实例中名称不同。

对于这两种情况，您需要在配置迁移任务之前，先在 RDS 控制台完成数据库创建。具体请参考 RDS 用户指南中的创建 RDS 实例。

创建迁移账号

迁移任务配置，需要提供本地 SQLServer 数据库及目标 RDS 实例的迁移账号。迁移账号所需权限请参见上文的迁移权限要求。

如果本地 SQLServer 实例迁移账号尚未创建，那么您可以参考*SQLServer User*创建，创建满足权限要求的迁移账号。

如果目标 RDS 实例迁移账号尚未创建，那么您可以参考 RDS账号创建流程，创建对待迁移数据库有读写权限的迁移账号。

配置迁移任务

当数据库、迁移账号都创建完成后，就可以开始配置迁移任务了。下面详细介绍下具体的配置步骤。

1. 登录数据传输控制台，单击右上角的**创建迁移任务**，开始任务配置。
2. 配置本地 SQLServer 及目标 RDS 实例连接信息，如图 *372:* 配置连接信息所示。

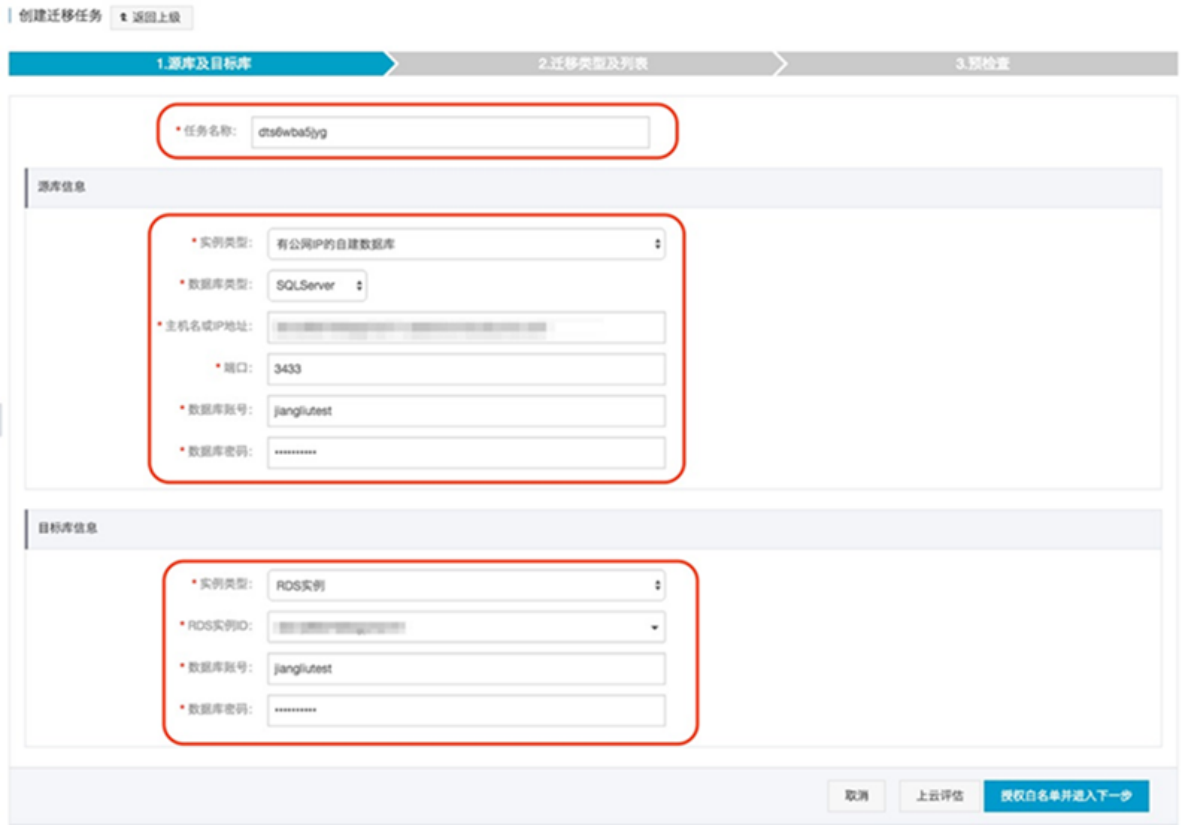
   在这个步骤中，主要配置迁移任务名称，迁移源实例及目标实例连接信息。其中：

   - 任务名称

     DTS 为每个任务自动生成一个任务名称，任务名称没有唯一性要求。您可以根据需要修改任务名称，建议为任务配置具有业务意义的名称，便于后续的任务识别。
   - 源实例连接信息
     - 实例类型：选择**有公网IP的自建数据库**。
     - 数据库类型：选择**SQLServer**。
     - 主机名或 IP 地址：配置本地 SQLSerever 数据库访问地址，这个地址必须为公网访问方式。
     - 端口：SQLServer 实例监听端口。
     - 数据库账号：SQLServer 实例访问账号。
     - 数据库密码：上面 SQLServer 访问账号对应的密码。
   - - 实例类型：选择**RDS实例**
     - RDS 实例 ID：配置迁移的目标 RDS 实例的实例 ID。 DTS 支持经典网络、VPC 网络的 RDS 实例。

- 账号：RDS 实例的连接账号。
- 密码：上面 RDS 实例连接账号对应的密码。

图 372: 配置连接信息



3. 配置迁移对象及迁移类型。

- 迁移类型

  迁移类型包括结构迁移、全量数据迁移及增量数据迁移。

  如果要做全量迁移，那么选择结构迁移 + 全量数据迁移。

  如果要做不停机迁移，那么选择结构迁移 + 全量数据迁移 + 增量数据迁移。

- 迁移对象

  迁移对象，需要选择您要迁移的对象。迁移对象选择的粒度可以为：库、表、列三个粒度。

  默认情况下，迁移对象迁移到目标库后，对象名跟源库一致。如果您迁移的对象在源实例跟目标实例上名称不同，那么需要使用 DTS 提供的对象名映射功能，详细使用方式请参考库表列映射。

当配置完迁移对象及迁移类型后，即进入任务启动前的预检查步骤。

4. 任务预检查。

   在迁移任务正式启动之前，会先进行前置预检查，只有预检查通过后，才能成功启动迁移。具体的预检查内容请参见预检查简介。

   如果预检查失败，那么可以单击具体检查项后的按钮，查看具体的失败详情，并根据失败原因修复后，重新进行预检查。

   图 373: 预检查



5. 启动迁移任务。

   当预检查通过后，我们可以启动迁移任务，任务启动后，可以到任务列表中查看具体的迁移状态及进度。

   增量数据迁移是个动态同步的过程，所以建议在增量迁移达到无延迟状态时，在目标数据库上进行业务验证，如果验证成功，那么可以停掉迁移任务，然后将业务切换到目标数据库。

   至此，完成将本地 SQLServer 数据库到 RDS For SQLServer 的数据迁移任务配置。

6. 数据同步验证。

   为了验证迁移过程中，源数据库变更是否可以同步到目标库，我们在迁移过程中，往源库插入一条记录，如图 *374: 源库导入*所示。

   图 374: 源库导入



可以查看此时源库表 sbtest1中新增了一条记录，此时到目标库中查询这条记录，如图 *375:* 目标库验证所示。

图 375: 目标库验证



所以，通过这种方式可以将迁移过程中源库产生的增量数据实时同步到目标库，从而实现数据迁移过程中，应用停机时间降低到分钟级别。

致此，完成本地 SQLServer 数据库到 RDS For SQLServer 不停服迁移的任务配置。

18.3.7 Oracle到PPAS不停机数据迁移

本小节简单介绍使用 DTS 进行 Oracle->RDS For PostgreSQL Plus Advanced Server 不停机数据迁移的任务配置流程。

使用数据传输*DTS*可以将本地的Oracle 的数据迁移至 RDS For PostgreSQL Plus Advanced Server。同时，DTS 提供的增量数据迁移功能，可以实现Oracle在提供服务的过程中，进行数据迁移，当 Oracle 跟 PPAS 两边数据达到动态同步过程中，将业务切换到 PPAS，使得依赖 Oracle 的应用停机时间最短。

为便于编写，下文用 RDS For PPAS 表示 RDS For PostgreSQL Plus Advanced Server。

迁移类型

Oracle->RDS For PPAS 的迁移，支持结构迁移、全量数据迁移及增量数据迁移。各个迁移阶段的支持范围如下：

- 结构迁移

  DTS 将迁移对象的结构定义迁移到目标实例。目前DTS支持的对象包括：表、视图、同义词、触发器、存储过程、存储函数、包、自定义类型。

- 全量数据迁移

  DTS 将源数据库迁移对象的存量数据迁移到目标实例。

  如果未选择增量数据迁移，那么 Oracle 数据库有数据写入的话，新增增量数据不一定能够被迁移到 PPAS 中。所以，如果要保证迁移数据一致性，那么尽量选择在业务低峰期进行停机迁移。

- 增量数据迁移

  增量数据迁移会将迁移过程中，源 Oracle 实例的增量变更数据同步到目标 PPAS 实例，增量数据迁移可以实现Oracle 实例在迁移过程中产生的增量数据被实时同步到目标库，最终 Oracle 同目标 RDS PPAS 实例进入动态数据同步的过程。

迁移限制

迁移限制如下：

- 迁移过程中，不支持 DDL 操作。
- 不支持物化视图的迁移。
- 结构迁移时，reverse index 迁移到 RDS For PPAS 中，存储成普通索引。
- 结构迁移时，位图索引迁移到 RDS For PPAS，存储成普通索引。
- 结构迁移时，分区索引迁移到 RDS For PPAS，在每个分区上创建独立的索引。
- 增量数据迁移只支持有主键，或有非空唯一索引的表。
- 增量数据迁移不支持 long 类型。

迁移权限要求

当使用 DTS 进行 Oracle->RDS For PPAS 迁移时，不同迁移类型，对源及目标数据库的迁移账号权限要求如表 *52:* 迁移类型及权限要求所示。

表 52: 迁移类型及权限要求

迁移类型	结构迁移	全量迁移	增量迁移
本地 Oracle 实例	schema 的 owner	schema 的 owner	SYSDBA
目的 RDS for PPAS 实例	schema 的 owner	schema 的 owner	schema 的 owner

迁移前置条件

- Oracle 版本为10g，11g，12c 。
- Oracle 开启 supplement log，且要求 supplemental_log_data_pk和supplemental_log_data_ui 开启。
- Oracle 要求开启 archive log。

数据类型映射关系

由于 Oracle 跟 RDS For PPAS 的数据类型不是一一对应的，所以数据传输服务在进行结构迁移时，会根据两种数据库类型的数据类型定义，进行类型映射，表 *53:* 数据类型及其映射关系为数据传输服务定义的数据类型映射关系。

表 53: 数据类型及其映射关系

Oracle数据类型	PPAS数据类型	数据传输服务是否支持
varchar2(n [char/byte])	varchar2[(n)]	支持
nvarchar2[(n)]	nvarchar2[(n)]	支持
char[(n [byte/char])]	char[(n)]	支持
nchar[(n)]]	nchar[(n)]	支持
number[(p[,s])]	number[(p[,s])]	支持
float(p)]	double precision	支持
long	long	支持
date	date	支持
binary_float	real	支持
binary_double	double precision	支持

Oracle数据类型	PPAS数据类型	数据传输服务是否支持
timestamp[(fractional_seconds_precision)]	timestamp[(fractional_seconds_precision)]	支持
timestamp[(fractional_seconds_precision)]with time zone	timestamp[(fractional_seconds_precision)]with time zone	支持
timestamp[(fractional_seconds_precision)]with local time zone	timestamp[(fractional_seconds_precision)]with time zone	支持
clob	clob	支持
nclob	nclob	支持
blob	blob	支持
raw	raw(size)	支持
long raw	long raw	支持
bfile	—	不支持
interval year(year_precision) to mongth	interval year to mongth	不支持
interval day(day_precision) to second[(fractional_seconds_precision)]	interval day to second[(fractional_seconds_precision)]	不支持

由于 RDS For PPAS 不支持数据类型 timestamp[(fractional_seconds_precision)]with local time zone，所以数据传输服务在迁移这种类型的数据时，会将其转换成 UTC 时区后，存入 RDS For PPAS 的数据类型 timestamp[(fractional_seconds_precision)]with time zone 中。

迁移步骤

下面详细介绍下使用 DTS 将 Oracle 数据库迁移到 RDS for PPAS 的任务配置流程。

迁移账号创建

迁移任务配置，需要提供 Oracle 数据库及目标 RDS 实例的迁移账号。迁移账号所需权限请参见上文的迁移权限要求。

如果您的 Oracle 实例的迁移账号尚未创建，那么您可以参考 *Oracle Grant*语法说明，创建满足要求的迁移账号。

RDS For PPAS 迁移账号的创建及授权操作如下：

1. 通过 PostgreSQL 客户端，在本地数据库中创建迁移账号。

   ```
   CREATE USER username IDENTIFIED BY password;
   ```

   其中：

   - username：要创建的账号。
   - password：该账号的登录密码。

2. 在本地数据库中给迁移账号授权，本地数据库中迁移账号的权限要求如表 *52:* 迁移类型及权限要求所示。

   ```
   GRANT privileges ON tablename TO username;
   ```

   其中：

   - privileges：该账号的操作权限，如 SELECT、INSERT、UPDATE 等。如果要授权该账号所有权限，则使用 ALL。
   - tablename：表名。如果要授权该账号所有的表权限，则使用通配符*。
   - username：要授权的账号名。

   例如：

   ```
   GRANT ALL ON * TO myuser;
   ```

迁移任务配置

当上面的所有前置条件都配置完成后，就可以开始迁移任务配置。下面详细介绍下具体的迁移步骤。

1. 登录数据传输控制台，单击右上角的**创建迁移任务**，正式开始任务配置。
2. 配置本地 Oracle 及目标 RDS 实例的连接信息。

   这个步骤主要配置迁移任务名称，Oracle 连接信息及目标 RDS 实例连接信息。其中：

   - 任务名称

     DTS 为每个任务自动生成一个任务名称，任务名称没有唯一性要求。您可以根据需要修改任务名称，建议为任务配置具有业务意义的名称，便于后续的任务识别。

   - 源实例信息
     - 实例类型：选择**有公网IP的自建数据库**。
     - 数据库类型：选择**Oracle**。

- 主机名或IP地址：配置 Oracle 访问地址，这个地址必须为公网访问方式。
- 端口：Oracle 实例的监听端口。
- SID：Oracle 实例的 SID。
- 账号：Oracle 实例的连接账号。
- 密码：上面 Oracle 连接账号对应的密码。

- 目标实例信息
  - 实例类型：选择**RDS实例**
  - RDS 实例 ID：配置迁移的目标 RDS 实例的实例 ID。 DTS 支持经典网络、VPC 网络的 RDS 实例。
  - 账号：RDS 实例的连接账号。
  - 密码：上面 RDS 实例连接账号对应的密码。

  当配置完连接信息后，单击右下角**授权白名单并进入下一步**，进行白名单授权。这个步骤 DTS 会将 DTS 服务器的 IP 地址添加到目标 RDS 实例的白名单中，避免因为 RDS 实例设置了白名单，导致 DTS 服务器连接不上 RDS 实例导致迁移失败。

3. 配置迁移对象及迁移类型，如图 *376:* 配置迁移类型和迁移对象所示。

   - 迁移类型

     迁移类型包括结构迁移、全量数据迁移及增量数据迁移。

     如果要做全量迁移，那么选择结构迁移 + 全量数据迁移。

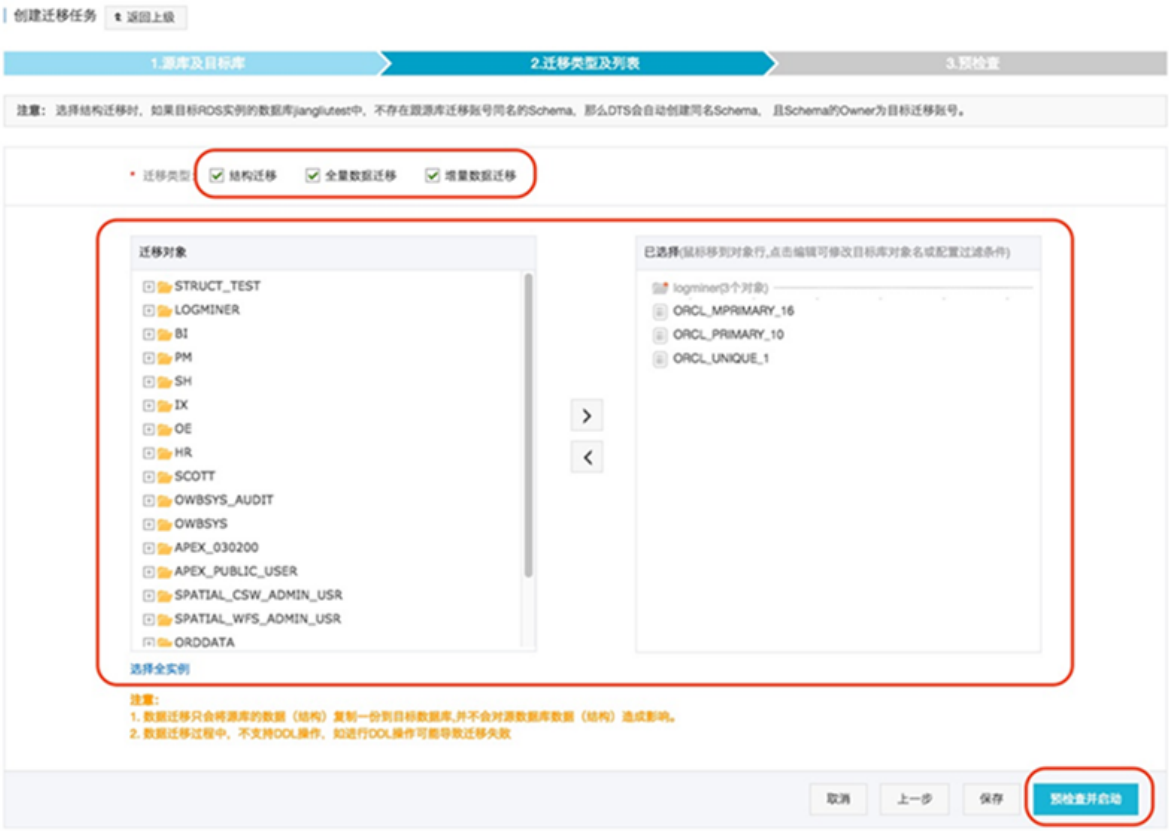
     如果要做不停机迁移，那么选择结构迁移 + 全量数据迁移 + 增量数据迁移。

   - 迁移对象

     迁移对象，需要选择您要迁移的对象。迁移对象选择的粒度可以为：库、表、列三个粒度。

     默认情况下，迁移对象迁移到目标库后，对象名跟源库一致。如果您迁移的对象在源实例跟目标实例上名称不同，那么需要使用 DTS 提供的对象名映射功能，详细使用方式请参考库表列映射。

图 376: 配置迁移类型和迁移对象



4. 预检查。

   在迁移任务正式启动之前，会先进行前置预检查，只有预检查通过后，才能成功启动迁移。具体的预检查内容请参见预检查简介。

   如果预检查失败，那么可以单击具体检查项后的按钮，查看具体的失败详情，并根据失败原因修复后，重新进行预检查。

   图 377: 预检查



5. 启动任务。

   当预检查通过后即启动迁移任务。

   当任务启动后，可以到任务列表中查看任务具体的迁移状态及进度。

   增量数据迁移是个动态同步的过程，所以建议在增量迁移达到无延迟状态时，在目标数据库上进行业务验证，如果验证成功，那么可以停掉迁移任务，然后将业务切换到目标数据库。

   至此，完成本地 Oracle 数据库到 RDS for PPAS 实例的不停机数据迁移的任务配置。

附：预检查内容

DTS在启动迁移之前，会进行前置预检查，本小节简单介绍 Oracle->RDS For PPAS 的预检查内容，如表 *54:* 预检查内容所示。

表 54: 预检查内容

预检查项	检查内容	备注
源库连接性检查	检查 DTS 服务同 Oracle 实例的连通性	(1) 填写信息是否有误？如果填写信息有误，请修改后重新预检查(2) 检查 Oracle 是否开启监听端口。
目的库连接性检查	检查 DTS 服务同目的 RDS For PPAS 实例的连通性	检查填写信息是否有误，如果有误请先修改后重新预检查。
源库版本检查	检查 Oracle 实例的版本，DTS是否支持	DTS 目前只支持10g，11g，12c三个版本。

预检查项	检查内容	备注
源库权限检查	检查 Oracle 实例访问账号的权限是否满足要求	如果权限不足，请参照上面的权限要求一节授权后，重新预检查。
目的库权限检查	检查 RDS For PPAS 实例访问账号的权限是否满足要求	如果权限不足，请参照上面的权限要求一节授权后，重新预检查。
同名对象存在性检查	检查待迁移对象在目标 RDS For PPAS 是否已经存在	如果检查失败，请将目标库中这些已经存在的对象删除后，重新进行预检查。
源端同名对象存在性检查	检查待迁移对象中，要迁入目标同一个 schema 的对象是否同名	如果检查失败，请参考库表列映射，将重名对象进行重命名。
源库日志模式检查	检查源库是否开启 archive log	如果未开启，请启用后，重新预检查。
约束完整性检查	检查待迁移对象依赖的父对象是否迁移	如果检查失败，那么可以修改迁移对象，同时迁移依赖的父对象后，重新预检查。
DBLINK 存在性检查	检查源库是否存在 DBLINK	如果存在，那么需要修改迁移对象，不选择 DBLINK。
增量拓扑冲突检查	检查同一个迁移对象是否已经存在迁移链路	如果存在冲突链路，那么需要删除掉冲突链路后，重新预检查。
字段类型检查	检查待迁移表的是否存在数据类型为 long 类型的字段	如果存在那么对应的表只能进行全量数据迁移，不能选择增量数据迁移。
表是否存在主键或者唯一性非空索引检查	检查待迁移表是否包含主键或非空唯一键	如果存在那么对应的表只能进行全量数据迁移，不能选择增量数据迁移。
补偿日志开启检查	检查是否开启 supplemental_log	如果未开启，请启用后，重新预检查。

18.3.8 从本地PostgreSQL迁移至RDS for PostgreSQL

本小节简单介绍使用 DTS 进行 PostgreSQL->RDS for PostgreSQL 数据迁移的任务配置流程。

使用数据传输DTS可以将本地 PostgreSQL 实例的数据迁移到RDS For PostgreSQL 实例。PostgreSQL 迁移支持增量数据同步功能，可以实现在本地应用不停止服务的情况下，平滑完成 PostgreSQL 数据库的迁移工作。

迁移类型

- 结构迁移

数据传输 DTS 将迁移对象的结构定义迁移到目标实例。对于 PostgreSQL ,数据传输服务支持结构迁移的对象包含：Table、trigger、view、sequence、function、user defined type、rule、domain、operation、aggregate。

- 全量数据迁移

  数据传输 DTS 将源数据库迁移对象的存量数据全部迁移到目标实例。

- 增量数据迁移

  增量数据迁移将迁移过程中，本地 PostgreSQL 实例的增量更新数据同步到目标 RDS 实例，最终实现本地 PostgreSQL 同目标 RDS 实例进入动态数据同步的过程。使用增量数据迁移，可以实现在本地 PostgreSQL 正常提供服务的时候，平滑完成 PostgreSQL->RDS For PostgreSQL 的数据迁移。

迁移限制

迁移限制如下：

- 源实例，支持 PostgreSQL 9.2、9.3、9.4。
- 迁移过程中，不支持 DDL 操作。
- 不支持迁移使用 C 语言编写的 function。
- 如果使用了对象名映射功能后，依赖这个对象的其他对象可能迁移失败。

迁移权限要求

当使用数据传输服务进行 PostgreSQL 迁移时，在不同迁移类型情况下，源跟目标数据库的迁移账号权限要求如表 *55:* 迁移类型及权限要求所示。

表 55: 迁移类型及权限要求

迁移类型	结构迁移	全量迁移	增量数据迁移
本地 PostgreSQL 实例	pg_catalog 的 usage 权限	迁移对象的 select	superuser
目的 RDS 实例	迁移对象的 create、usage 权限	schema 的 owner	schema 的owner

迁移流程

数据传输服务在进行 PostgreSQL->RDS For PostgreSQL 数据迁移时，为了解决对象间的依赖，提高迁移成功率。

结构对象及数据的迁移顺序如下：

1. 进行结构对象：Table、view、sequence、function、user defined type、rule、domain、operation、aggregate的迁移。
2. 全量数据迁移
3. 进行结构对象：trigger、foreign key的迁移。

全量数据迁移完成后，任务列表中的迁移进度为：**结构迁移**100%，**全量迁移100%**，但迁移状态为**迁移中**。此时迁移任务正在进行步骤 （3）外键的迁移，请勿手动结束任务，否则会造成迁移数据不一致。

迁移步骤

下面详细介绍下使用 DTS 将本地的 PostgreSQL 数据库迁移到 RDS for PostgreSQL 的任务配置流程。

RDS 实例数据库创建

在数据迁移过程中，如果待迁移的数据库在目标 RDS 实例中不存在，那么 DTS 自动会创建。但是对于如下两种情况，您需要在配置迁移任务之前，手动创建数据库。

- 数据库名称不符合：RDS 定义规范（由小写字母、数字、下划线、中划线组成，字母开头，字母或数字结尾，最长 64 个字符）。
- 待迁移数据库，在 Oracle 跟目标 RDS 实例中名称不同。
- 数据库名称不符合：RDS 定义规范（由小写字母、数字、下划线、中划线组成，字母开头，字母或数字结尾，最长 64 个字符）。
- 待迁移数据库，在本地 PostgreSQL 跟目标 RDS For PostgreSQL 实例中存储名称不同。

对于这两种情况，您需要在配置迁移任务之前，先在 RDS 控制台完成数据库创建。具体请参考 RDS 用户指南中的创建 RDS 实例。

迁移账号创建

迁移任务配置，需要提供本地 PostgreSQL 数据库及目标 RDS 实例的迁移账号。迁移账号所需权限请参见上文的迁移权限要求。

如果您的本地 PostgreSQL 或 RDS For PostgreSQL 实例的迁移账号尚未创建，那么可以参考如下流程创建迁移账号：

1. 通过 PostgreSQL 客户端，在 PostgreSQL中创建迁移账号。

   ```
   create user username password 'password';
   ```

如果您要使用增量迁移，那么创建的账号必须是 superuser，所以创建账号语句调整为：

```
create user username with superuser password 'password';
```

其中：

- username：要创建的账号。
- password：该账号的登录密码。

2. 给迁移账号授权，本地 PostgreSQL 及 RDS For PostgreSQL 的迁移账号权限要求如表 *55:* 迁移类型及权限要求所示。

```
GRANT privileges ON tablename TO username;
```

其中：

- privileges：该账号的操作权限，如 SELECT、INSERT、UPDATE 等。如果要授权该账号所有权限，则使用 ALL。
- tablename：表名。如果要授权该账号所有的表权限，则使用通配符*。
- username：要授权的账号名。

逻辑流复制插件安装

如果您需要使用增量数据迁移进行不停机迁移，那么在任务配置之前，需要在本地 PostgreSQL 中安装 DTS 提供的逻辑流复制插件。

1. 插件下载。
2. 插件安装。
   a. 解压下载的压缩包。
   b. 将*ali_decoding.so*文件拷贝到 PostgreSQL 安装路径的 lib 目录下。

      如果用 rpm 包安装，那么这个绝对路径为*/usr/pgsql-{$version}/lib/*，其中*$version*为版本号，例如版本为 9.5，那么绝对路径为*/usr/pgsql-9.5/lib*。
   c. 将*ali_decoding.contorl*文件拷贝到 PostgreSQL 安装路径下的*share/extension*目录下。

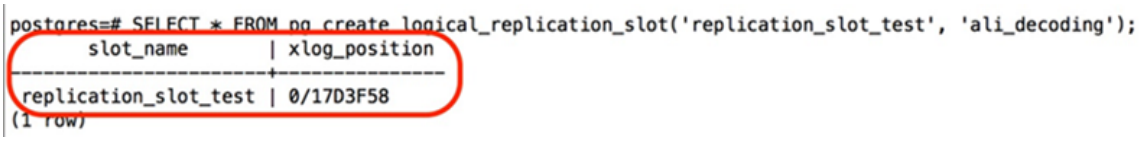
      如果用rpm包安装，那么这个绝对路径为*/usr/pgsql-${version}/share/extension/*，其中*${version}*为 PostgreSQL 版本号；如果版本为 9.5 ，那么绝对路径为*/usr/pgsql-9.5/share/extension/*。
   d. 测试安装是否成功。

使用superuser账号登录 PostgreSQL 客户端，运行如下 SQL，看是否能够成功创建replication slot。如果成功创建，则说明插件安装成功。

```
SELECT * FROM pg_create_logical_replication_slot('replication_slot_test', 'ali_decoding');
```

如果输出如图 *378: 测试结果*所示结果，说明插件安装成功。

图 378: 测试结果



测试成功后，使用如下 SQL 将 replication slot 删除掉。

```
SELECT pg_drop_replication_slot('replication_slot_test');
```

迁移任务配置

当上面的所有前置条件都配置完成后，就可以开始正式的数据迁移了。下面详细介绍迁移任务配置流程。

1. 登录数据传输控制台，单击右上角的**创建迁移任务**，开始迁移任务配置。
2. 配置本地 PostgreSQL 跟目标 RDS For PostgreSQL 实例连接信息，如图 *379: 配置信息*所示。

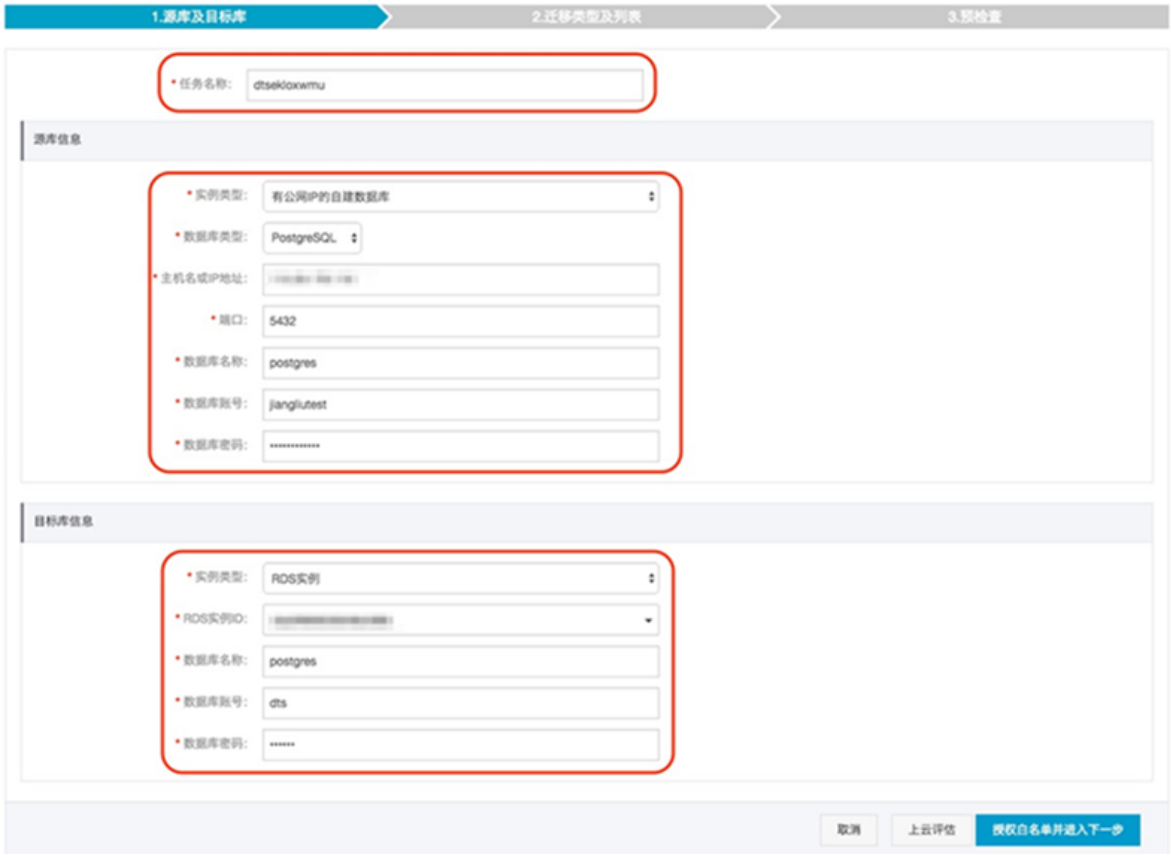
   这个步骤主要配置迁移任务名称、PostgreSQL 连接信息及目标 RDS For PostgreSQL 实例连接信息。其中：

   - 任务名称
   - 任务名称

     DTS 为每个任务自动生成一个任务名称，任务名称没有唯一性要求。您可以根据需要修改任务名称，建议为任务配置具有业务意义的名称，便于后续的任务识别。
   - 源实例信息
     - 实例类型：选择**有公网IP的自建数据库**。
     - 数据库类型：选择**PostgreSQL**。
     - 主机名或 IP 地址：配置 PostgreSQL 访问地址，这个地址必须为公网访问方式。
     - 端口：本地 PostgreSQL 实例的监听端口。
     - 数据库名称：连接 PostgreSQL 的默认数据库名。

  - 数据库账号：本地 PostgreSQL 实例的连接账号。
  - 数据库密码：本地 PostgreSQL 实例连接账号对应的密码。
- - 实例类型：选择**RDS实例**
  - RDS 实例 ID：配置迁移的目标 RDS 实例的实例 ID。 DTS 支持经典网络、VPC 网络的 RDS 实例。
  - 账号：RDS 实例的连接账号。
  - 密码：上面 RDS 实例连接账号对应的密码。

图 379: 配置信息



3. 选择迁移对象及迁移类型，如图 *380:* 选择迁移对象及类型所示。
   - 迁移类型

     对于 PostgreSQL->RDS For PostgreSQL，支持 结构迁移、全量数据迁移、增量数据迁移。

     如果只需要进行全量迁移，那么迁移类型选择：结构迁移 + 全量数据迁移。

     如果需要进行不停机迁移，那么迁移类型选择：结构迁移 + 全量数据迁移 + 增量数据迁移。
   - 迁移对象

选择您要迁移的对象。迁移对象选择的粒度可以为：库、表、列三个粒度。

默认情况下，对象迁移到 RDS for PostgreSQL 实例后，对象名跟本地 PostgreSQL 实例一致。如果您迁移的对象在源实例跟目标实例上名称不同，那么需要使用 DTS 提供的对象名映射功能，详细使用方式请参考库表列映射。

图 380: 选择迁移对象及类型

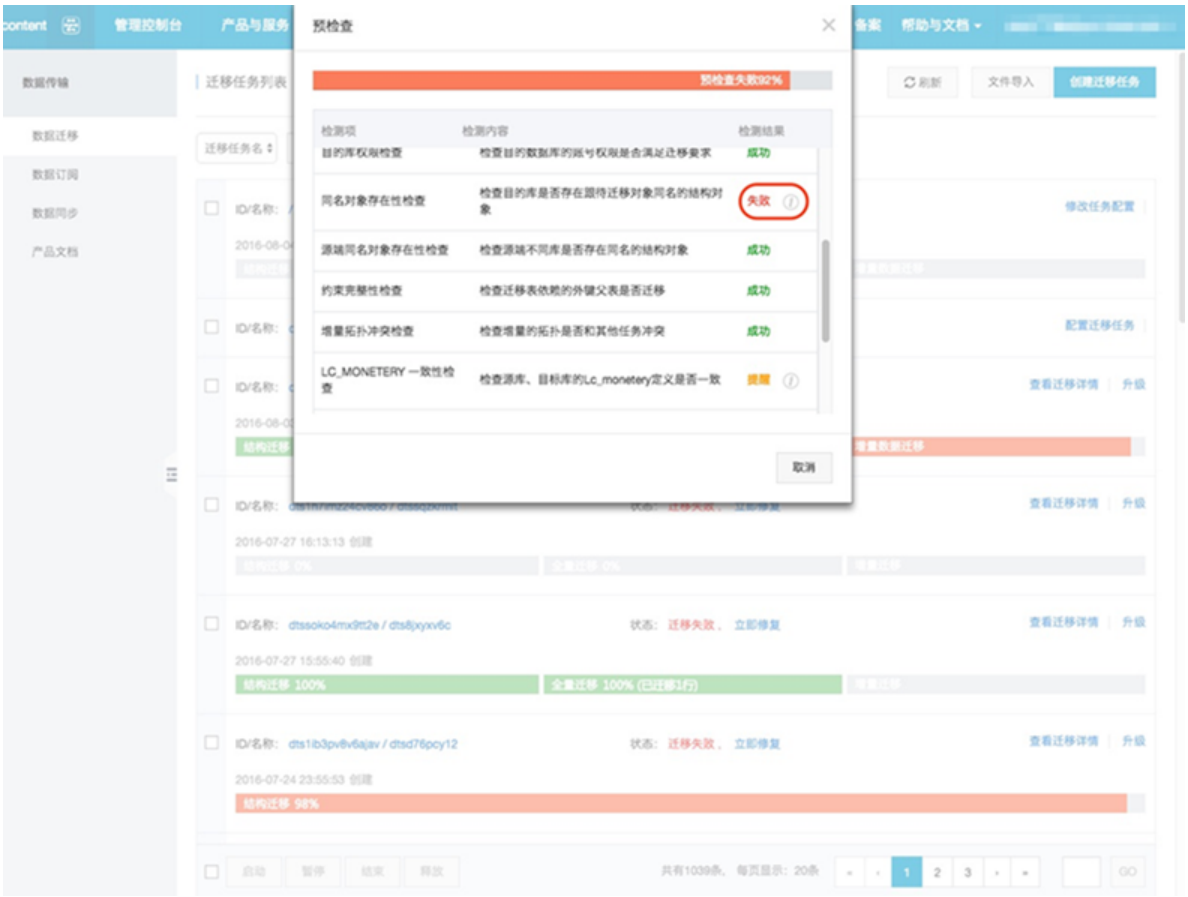


4. 预检查，如图 *381:* 预检查所示。

   在迁移任务正式启动之前，会先进行前置预检查，只有预检查通过后，才能成功启动迁移。具体的预检查内容请参见预检查简介。

   如果预检查失败，那么可以单击具体检查项后的按钮，查看具体的失败详情，并根据失败原因修复后，重新进行预检查。

   图 381: 预检查



5. 启动迁移任务。

   当预检查通过后，可以启动迁移任务，任务启动成功后，可以在任务列表中查看迁移的具体状态及迁移进度。

   至此，完成本地 PostgreSQL 数据库到 RDS for PostgreSQL 实例的数据迁移任务配置。

附：预检查内容

DTS在启动迁移之前，会进行前置预检查，如表 *56:* 预检查内容所示为 PostgreSQL->RDS for PostgreSQL 的预检查内容。

表 56: 预检查内容

预检查项	检查内容	备注
源库连接性检查	检查 DTS 服务同本地 PostgreSQL 实例的连通性。	1. 填写信息是否有误？如果填写信息有误，请修改后重新预检查。 2. 检查端口是否配置从其他服务器连接。

预检查项	检查内容	备注
目的库连接性检查	检查DTS服务同目的RDS For PostgreSQL实例的连通性。	检查填写信息是否有误，如果有误请先修改后重新预检查。
目的库连接性检查	检查 DTS 服务同目的 RDS For PostgreSQL 实例的连通性。	检查填写信息是否有误，如果有误请先修改后重新预检查。
源库版本检查	检查本地 PostgreSQL 版本跟目标 RDS For PostgreSQL 是否一致。	如果版本不一致，预检查会有提醒。可以根据提醒对本地 PostgreSQL 进行升级或降级，也可以继续迁移。
数据库可用性检查	检查待迁移数据库在目标 RDS For PostgreSQL 是否已存在。	如果待迁移数据库命名规范不满足 RDS 实例要求，那么 DTS 在目标 RDS 实例创建待迁移数据库会报错失败，即数据库可用性检查会失败。此时请参考库表列映射，对迁移数据库进行重命名。
源库权限检查	检查任务配置时，提供的本地PostgreSQL数据库账号的权限是否满足要求	如果检查失败，那么请参考上文迁移账号创建一节对迁移账号进行授权后，重新进行预检查。
目的库权限检查	检查任务配置时，提供的目的RDS For PostgreSQL 数据库账号的权限是否满足要求。	如果检查失败，那么请参考上文迁移账号创建一节对迁移账号进行授权后，重新进行预检查。
同名对象存在性检查	检查待迁移对象在目标 RDS For PostgreSQL 实例中是否已经存在。	如果检查失败，请将目标库中这些已经存在的对象删除后，重新进行预检查。
源端同名对象存在性检查	检查要迁移到同一个数据库中的多个对象是否重名。	如果检查失败，请参考库表列映射，将对象进行重命名。
约束完整性检查	检查待迁移对象依赖的父对象是否迁移。	如果检查失败，那么可以修改迁移对象，同时迁移依赖的父对象后，重新预检查。
增量拓扑冲突检查	检查同一个迁移对象是否已经存在迁移链路。	如果存在冲突链路，那么需要删除掉冲突链路后，重新预检查。
LC_MONETERY 一致性检查	检查源库、目标库的 Lc_monetery 定义是否一致。	如果检查失败，可以修改目标**RDS For PostgreSQL的LC_MONETERY**定义，或者继续迁移。

预检查项	检查内容	备注
PostgreSQL逻辑流复制插件检查	检查本地 PostgreSQL 实例是否安装了逻辑流复制插件。	如果检查失败，可以参考下面的逻辑流复制插件安装一节安装逻辑流复制插件后重新预检查。
PostgreSQL逻辑流slot存在性检查	检查本地 PostgreSQL 实例存在跟 DTS 创建的 replication slot 重名的replication slot。	如果检查失败，可以删除本地 PostgreSQL 中已经存在的同名 replication slot 后，重新预检查。

18.3.9 从本地MongoDB迁移至MongoDB实例

本小节简单介绍使用 DTS 进行 MongoDB->RDS For MongoDB 数据迁移的任务配置流程。

使用数据传输 DTS 可以将本地的 MongoDB 实例的数据迁移至 RDS For MongoDB Mongodb 迁移支持增量数据同步功能，可以实现在本地应用不停服的情况下，平滑完成 MongoDB 数据库的迁移工作。

迁移类型

对于 MongoDB->云数据库 MongoDB 数据迁移，DTS 支持全量数据迁移及增量数据迁移，这个迁移类型的功能及限制如下。

- 全量数据迁移

  数据传输 DTS 将源数据库迁移对象的存量数据全部迁移到目标实例。

- 增量数据迁移

  增量数据迁移将迁移过程中，本地 MongoDB 实例的增量更新数据同步到云数据库 MongoDB，最终本地 MongoDB 同云数据库 MongoDB 进入动态数据同步的过程。使用增量数据迁移，可以实现在本地 MongoDB 正常提供服务的时候，平滑完成 MongoDB->RDS For MongoDB 的数据迁移。

迁移功能

- MongoDB 全量数据迁移支持：
  - 支持 database 迁移
  - 支持 collection 迁移
  - 支持 index 迁移
- MongoDB 增量数据迁移支持：
  - 支持 document 新增、删除、更新操作的同步。

- 支持collection 新建、删除操作的同步。
- 支持database 新建、删除操作的同步。
- 支持index 新建、删除操作的同步。

迁移权限要求

当使用 DTS 进行 MongoDB-> 云数据库MongoDB 迁移时，不同迁移类型，对源跟目标 MySQL 实例的迁移账号权限要求如表 *57:* 迁移类型及权限要求所示。

表 57: 迁移类型及权限要求

迁移类型	全量数据迁移	增量数据迁移
本地 MongoDB	read	待迁移库的 readadmin 的 read权限 local 的 read 权限
目的 MongoDB 实例	readWrite	readWrite

迁移步骤

下面详细介绍使用 DTS 将本地的 MongoDB 迁移到云数据库 MongoDB 的任务配置流程。

迁移账号创建

迁移任务配置，需要提供本地自建 MongoDB 实例及目标 MongoDB 云数据库的迁移账号，迁移账号所需权限请参见上面的迁移权限要求一节。

如果您的本地 MongoDB 或 MongoDB 云数据库的迁移账号尚未创建，可以参考如下流程创建迁移账号：

```
db.createUser({user:"username",pwd:"password",roles:
[{role:"rolename1",db:"database_name1"},{role:"rolename2",db:"database_name2"}]});
```

其中：

- username：要创建的账号。
- password：上面账号对应的密码。
- rolename1/rolename2：待授权给 username 的角色名，例如上面的 read， readWrite。
- database_name1/database_name2：只将 database_name1/database_name2 上的角色 role1/role2 授权给 username。

  关于MongoDB的角色授权请参考 *MongoDB Create User*说明。

迁移任务配置

当上面的所有前置条件都配置完成后，就可以开始正式的数据迁移了。下面详细介绍迁移任务配置流程。

1. 进入数据传输 DTS 控制台，单击右上角的创建迁移任务，开始迁移任务配置。
2. 配置本地 MongoDB 跟目标 MongoDB 实例的连接信息，如图 *382:* 配置信息所示。

   这个步骤主要配置迁移任务名称，本地 MongoDB 连接信息及目标 MongoDB 云数据库连接信息。其中：

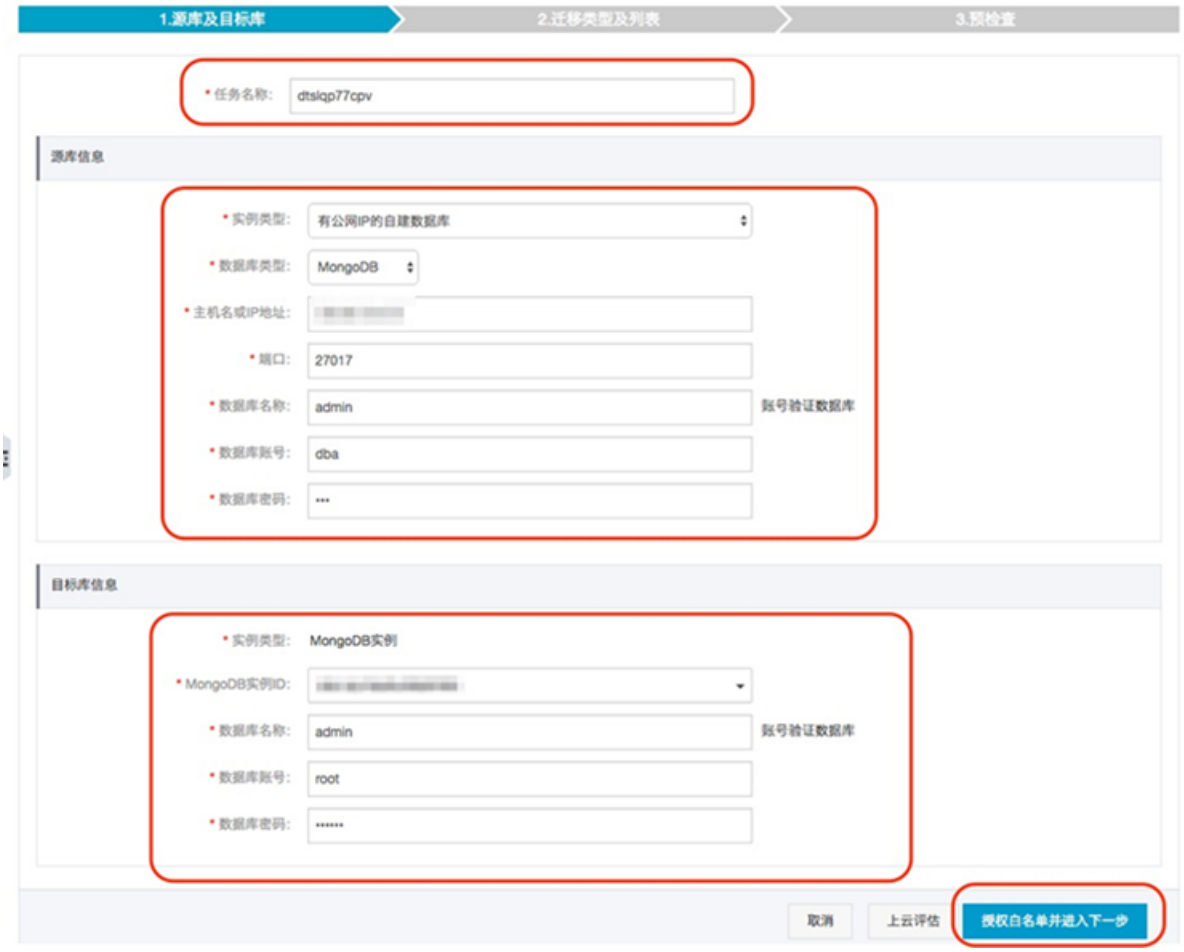
   - 任务名称

     DTS 为每个任务自动生成一个任务名称，任务名称没有唯一性要求。您可以根据需要修改任务名称，建议为任务配置具有业务意义的名称，便于后续的任务识别。

   - 源实例信息
     - 实例类型：选择**有公网IP的自建数据库**
     - 数据库类型：选择**MongoDB**
     - 主机名或 IP 地址：配置 MongoDB 访问地址，这个地址必须为公网访问方式。
     - 端口：本地 MongoDB 实例的监听端口。
     - 数据库名称：连接 MongoDB 实例的默认数据库名。
     - 数据库账号：本地 MongoDB 实例的连接账号。
     - 数据库密码：本地 MongoDB 实例连接账号对应的密码。
   - 目标实例信息
     - 实例类型：选择**MongoDB实例**。
     - MongoDB 实例 ID：配置迁移的目标 MongoDB 实例的实例 ID。 DTS 支持经典网络的 MongoDB 实例。如果您的 MongoDB 实例为VPC 网络的实例，那么需要切换到经典网络模式后，再使用 DTS 进行迁移。
     - 数据库名称：连接 MongoDB 的默认数据库名。
     - 数据库账号：连接 MongoDB 实例的连接账号。
     - 数据库密码：上面数据库账号对应的密码。

     当配置完连接信息后，单击右下角**授权白名单并进入下一步**进行白名单授权。这个步骤 DTS 会将 DTS 服务器的 IP：地址添加到目标 MongoDB 云数据库的白名单中，避免因为 MongoDB 实例设置了白名单，DTS 服务器连接不上 MongoDB 实例导致迁移失败。

图 382: 配置信息



3. 选择迁移对象及迁移类型，如图 *383:* 选择迁移对象及类型所示。

   - 迁移类型

     对于 MongoDB，支持 全量数据迁移、增量数据迁移。

     如果只需要进行全量迁移，那么迁移类型选择：全量数据迁移。

     如果需要进行不停机迁移，那么迁移类型选择：全量数据迁移 + 增量数据迁移。

   - 迁移对象

     这个步骤需要选择要迁移的对象。迁移对象选择的粒度可以为：库、collection/function 两个粒度。默认情况下，对象迁移到 MongoDB 实例后，对象名跟本地 MongoDB 实例一致。如果您迁移的对象在源实例跟目标实例上名称不同，那么需要使用 DTS 提供的对象名映射功能，详细使用方式请参考库表列映射。

     图 383: 选择迁移对象及类型

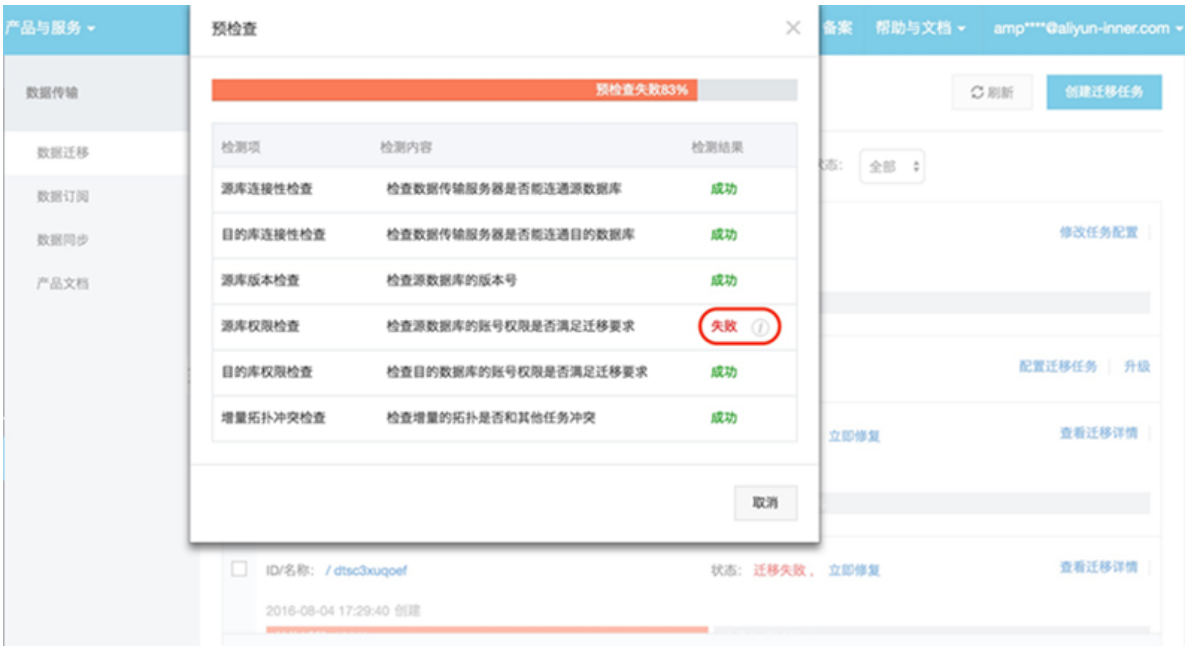


4. 预检查，如图 *384:* 预检查所示。

   在迁移任务正式启动之前，会先进行前置预检查，只有预检查通过后，才能成功启动迁移。具体的预检查内容请参见预检查简介。

   如果预检查失败，那么可以单击具体检查项后的按钮，查看具体的失败详情，并根据失败原因修复后，重新进行预检查。

   图 384: 预检查



5. 启动迁移任务。

当预检查通过后，可以启动迁移任务，任务启动成功后，可以在任务列表中查看迁移的具体状态及迁移进度。

至此，完成本地 MongoDB 数据库到 MongoDB 实例的数据迁移任务配置。

附：预检查内容

DTS 在启动迁移之前，会进行前置预检查，如表 *58:* 预检查内容所示为 MongoDB->RDS For MongoDB 的预检查内容。

表 58: 预检查内容

检查项	检查内容	备注
源库连接性检查	检查 DTS 服务器跟本地 MongoDB 实例的连通性。	1. 填写信息是否有误？如果填写信息有误，请修改后重新预检查。 2. 检查端口是否配置从其他服务器连接访问。
目标库连接性检查	检查 DTS 服务器跟目标 MongDB 实例的连通性。	检查填写信息是否有误，如果有误请先修改后重新预检查。
源库版本检查	检查本地 MongoDB 的版本，DTS 是否支持	请先升级版本到 3.2 后，重新预检查。
源库权限检查	检查本地 MongoDB 提供的迁移账号的权限是否满足需求。	如果检查失败，请参照迁移账号创建一节授权后，重新预检查。
目的库权限检查	检查目的 MongoDB 数据库提供的迁移账号的权限是否满足需求。	如果检查失败，请参照迁移账号创建一节授权后，重新预检查。
增量拓扑冲突检查	检查 MongoDB 实例上是否有其他增量迁移任务正在运行。	如果检查失败，那么需要结束或删除其他的增量迁移任务后，重新预检查。

18.3.10 RDS实例间的数据迁移

本小节简单介绍使用 DTS 进行两个 RDS 实例之间的数据迁移的任务配置流程。

使用数据传输 DTS 可以实现两个 RDS 实例间的数据迁移。对于支持增量迁移的存储引擎，还可以使用 DTS 在源 RDS 实例不停服的情况下，将数据迁移到目标 RDS 实例。目前对于 RDS 不同存

储引擎，只支持同构迁移(例如R DS For MySQL 迁移到 RDS For MySQL)，不支持异构迁移（例如 RDS For MySQL 迁移到 RDS For SQLServer）。

迁移权限要求

当使用 DTS 进行两个 RDS 实例间的数据迁移时，不同迁移类型，对源跟目标数据库的迁移账号权限要求如表 *59:* 迁移类型及权限要求所示。

表 59: 迁移类型及权限要求

迁移类型	结构迁移	全量迁移	增量迁移
源 RDS 实例	读写权限	读写权限	读写权限
目的 MySQL	读写权限	读写权限	读写权限

迁移步骤

下面详细介绍如何使用 DTS 实现两个 RDS MySQL 实例间的数据迁移。源 RDS 跟目标 RDS 实例可以不同也可以相同，即您可以使用 DTS 实现一个 RDS 实例内部的数据迁移。

RDS 实例数据库创建

在数据迁移过程中，如果待迁移的数据库在目标 RDS 实例中不存在，那么 DTS 自动会创建。但是对于如下两种情况，您需要在配置迁移任务之前，手动创建数据库。

- 数据库名称不符合：RDS 定义规范（由小写字母、数字、下划线、中划线组成，字母开头，字母或数字结尾，最长 64 个字符）。
- 待迁移数据库，在 Oracle 跟目标 RDS 实例中名称不同。
- 数据库名称不符合：RDS 定义规范（由小写字母、数字、下划线、中划线组成，字母开头，字母或数字结尾，最长 64 个字符）。
- 待迁移数据库，在本地 RDS MySQL 跟目标 RDS MySQL 实例中存储名称不同。

对于这两种情况，您需要在配置迁移任务之前，先在 RDS 控制台完成数据库创建。具体请参考 RDS 用户指南中的创建 RDS 实例。

迁移账号创建

迁移任务配置时，需要提供源 RDS 实例及目的 RDS 实例的迁移账号。迁移账号的相关权限请参见上面的迁移权限要求 一节。如果尚未创建迁移账号，可以参考 RDS 实例账号创建。需要先在源及目的 RDS 实例中创建迁移账号，并将要迁移的库表的读写权限授权给上面创建的账号。在 RDS 控制台中创建账号，并将迁移库的读写权限授权给这个账号。

迁移任务配置

当上面的所有前置条件都配置完成后，就可以开始正式的数据迁移了。下面详细介绍迁移任务配置流程。

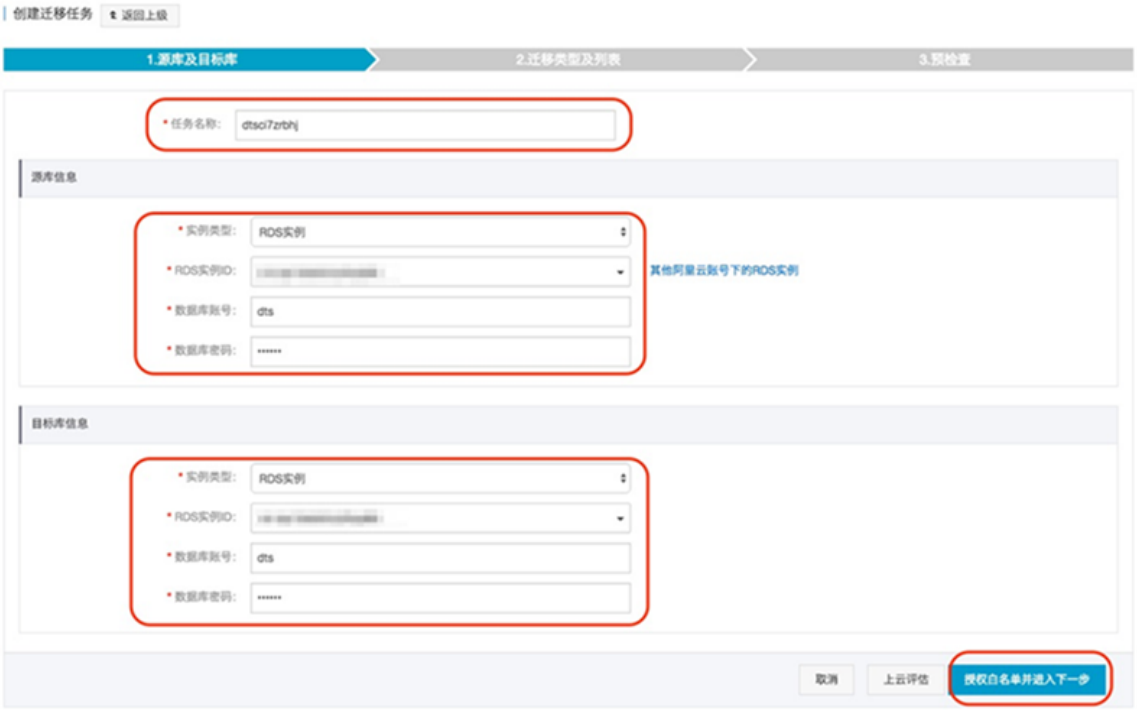
1. 登录数据传输控制台，单击右上角的**创建迁移任务**，开始迁移任务配置。
2. 配置源及目的实例连接信息，如图 *385:* 配置连接信息所示。

   这个步骤主要配置迁移任务名称、源 RDS 连接信息及目标 RDS 实例连接信息。其中：

   - 任务名称

     DTS 为每个任务自动生成一个任务名称，任务名称没有唯一性要求。您可以根据需要修改任务名称，建议为任务配置具有业务意义的名称，便于后续的任务识别。
   - 源实例信息
     - 实例类型：选择 RDS 实例。
     - RDS 实例 ID：配置迁移的源 RDS 实例的实例 ID。DTS支持经典网络、VPC 网络的 RDS 实例。
     - 数据库名称：如果 RDS 实例的存储引擎为 PostgreSQL 或 PPAS，那么需要配置数据库名称。这个数据库名称为连接 RDS 实例使用的默认数据库。
     - 数据库账号：连接 RDS 实例的账号。
     - 数据库密码：上面数据账号对应的密码。
   - 目标实例信息
     - 实例类型：选择**RDS实例**
     - RDS 实例 ID：配置迁移的目标 RDS 实例的实例 ID。 DTS 支持经典网络、VPC 网络的 RDS 实例。
     - 账号：RDS 实例的连接账号。
     - 密码：上面 RDS 实例连接账号对应的密码。

   图 385: 配置连接信息



3. 选择迁移对象及迁移类型，如图 *386:* 选择迁移对象及类型所示。

   - 迁移类型

     迁移类型包括结构迁移、全量数据迁移及增量数据迁移。

     如果要做全量迁移，那么选择结构迁移 + 全量数据迁移。

     如果要做不停机迁移，那么选择结构迁移 + 全量数据迁移 + 增量数据迁移。

   - 迁移对象

     迁移对象，需要选择您要迁移的对象。迁移对象选择的粒度可以为：库、表、列三个粒度。

     默认情况下，迁移对象迁移到目标库后，对象名跟源库一致。如果您迁移的对象在源实例跟目标实例上名称不同，那么需要使用 DTS 提供的对象名映射功能，详细使用方式请参考库表列映射。

图 386: 选择迁移对象及类型



4. 预检查，如图 *387:* 预检查所示。

   在迁移任务正式启动之前，会先进行前置预检查，只有预检查通过后，才能成功启动迁移。具体的预检查内容请参见预检查简介。

   如果预检查失败，那么可以单击具体检查项后的按钮，查看具体的失败详情，并根据失败原因修复后，重新进行预检查。

   图 387: 预检查



5. 启动迁移任务。

   当预检查通过后，可以启动迁移任务，任务启动成功后，可以在任务列表中查看迁移的具体状态及迁移进度。

   如果选择了增量迁移，那么进入增量迁移阶段后，源库的更新写入都会被 DTS 同步到目标 RDS 实例。迁移任务不会自动结束。如果您只是为了迁移，那么建议在增量迁移无延迟的状态时，源库停写几分钟，等待增量迁移再次进入无延迟状态后，停止掉迁移任务，直接将业务切换到目标 RDS 实例上即可。

   至此，完成两个 RDS 实例的数据迁移任务配置。

18.3.11 从ECS上的自建数据库迁移至RDS/MongoDB/PetaData/OceanBase

本小节简单介绍使用 DTS 进行 ECS 自建数据库迁移到 RDS 实例的数据迁移任务的配置流程。

使用数据传输 DTS 可以实现 ECS 自建数据库到 RDS 实例的数据迁移。对于支持增量迁移的存储引擎，还可以使用 DTS 在 ECS 自建数据库不停服的情况下，将数据迁移到目标 RDS 实例。

下面详细介绍使用 DTS 将 ECS 上的自建数据库迁移到 RDS 实例的任务配置流程。

RDS实例数据库创建

在数据迁移过程中，如果待迁移的数据库在目标 RDS 实例中不存在，那么 DTS 自动会创建。但是对于如下两种情况，您需要在配置迁移任务之前，手动创建数据库。

- 数据库名称不符合：RDS 定义规范（由小写字母、数字、下划线、中划线组成，字母开头，字母或数字结尾，最长 64 个字符）。
- 待迁移数据库，在 Oracle 跟目标 RDS 实例中名称不同。
- 数据库名称不符合：RDS 定义规范（由小写字母、数字、下划线、中划线组成，字母开头，字母或数字结尾，最长 64 个字符）。
- 待迁移数据库，在 ECS 自建数据库跟目标 RDS 实例中存储名称不同。

对于这两种情况，您需要在配置迁移任务之前，先在 RDS 控制台完成数据库创建。具体请参考 RDS 用户指南中的创建 RDS 实例。

迁移账号创建

迁移任务配置时，需要提供 ECS自建数据库及目的 RDS 实例的迁移账号。

自建数据库的存储引擎不同，账号创建及权限授权的方式不同，具体授权方式请参考各存储引擎的官方文档。

目标 RDS 迁移账号的创建流程请参考 RDS账号创建说明。

迁移任务配置

当上面的所有前置条件都配置完成后，就可以开始正式的数据迁移了。下面详细介绍迁移任务配置流程。

1. 登录数据传输控制台，单击右上角的**创建迁移任务**，开始迁移任务配置。
2. 配置源及目的实例连接信息，如图 *388:* 配置连接信息所示。

   这个步骤主要配置 迁移任务名称，源 ECS 连接信息及目标 RDS 实例连接信息。其中：

   - 任务名称

     DTS 为每个任务自动生成一个任务名称，任务名称没有唯一性要求。您可以根据需要修改任务名称，建议为任务配置具有业务意义的名称，便于后续的任务识别。
   - 源库信息
     - 实例类型：选择**ECS上的自建数据库**。
     - 实例地区：选择对应的实例区域。
     - ECS 实例 ID：配置迁移的源 ECS 实例的实例 ID。DTS 支持经典网络、VPC 网络的 ECS 实例。如果 ECS 网络模式为 VPC 网络，那么 ECS 上面必须挂载公网 EIP。
     - 数据库类型：选择**ECS上自建数据库**的数据库类型，例如 MySQL、SQLServer 等。

- 端口号：默认值。
- 数据库名称：如果 RDS 实例的存储引擎为 PostgreSQL/PPAS/MongoDB，那么需要配置数据库名称。这个数据库名称为连接 ECS 自建数据库时使用的默认数据库。
- 数据库账号：连接自建数据库的账号。
- 数据库密码：上面数据账号对应的密码。

- 目标库信息
  - 实例类型：选择要迁入的实例对应的实例类型，包括 RDS/MongoDB/PetaData/OceanBase。
  - 实例地区：选择要迁入的实例所在的地区。
  - 实例 ID：配置迁移的目标实例的实例 ID。
  - 数据库名称：如果数据库类型为 PostgreSQL/PPAS/MongoDB，那么需要配置数据库名称。这个数据库名称为连接实例时使用的默认数据库。
  - 数据库账号：连接实例的账号。
  - 数据库密码：上面数据账号对应的密码。

图 388: 配置连接信息



3. 选择迁移对象及迁移类型，如图 *389:* 选择迁移对象及类型所示。

- 迁移类型

  对于不同的数据库类型，DTS 支持的迁移类型不同。

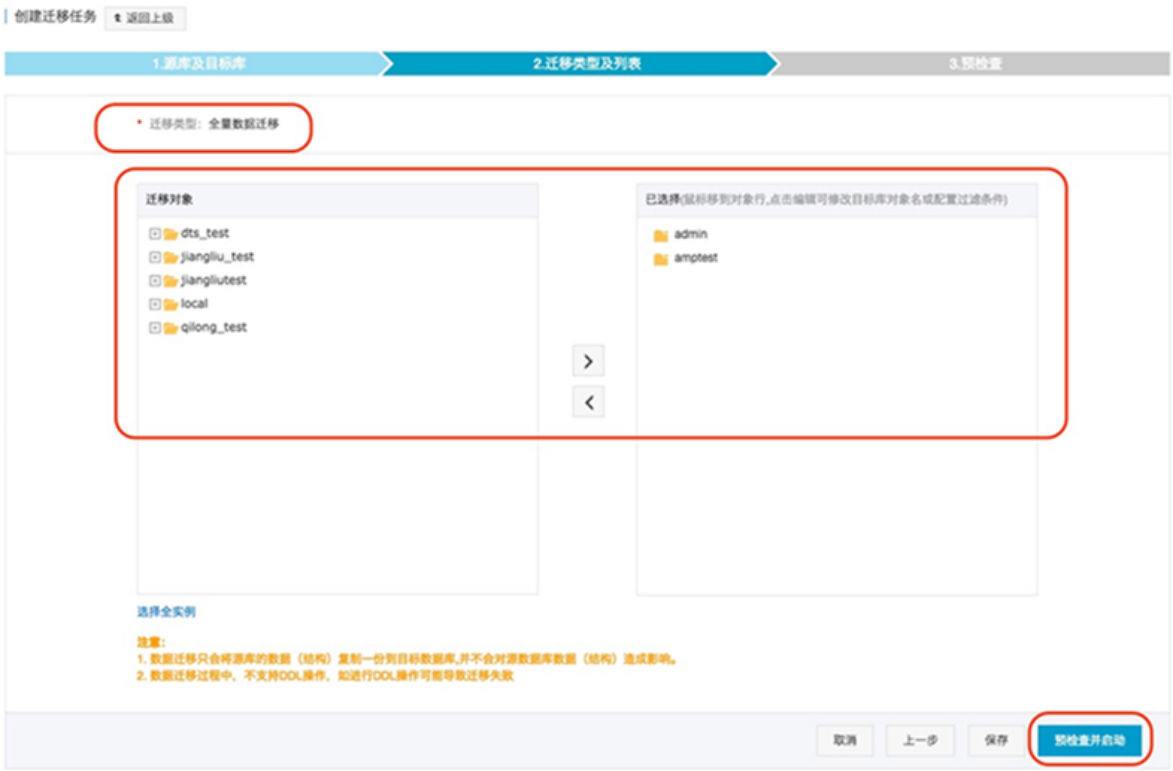
  如果只需要进行全量迁移，那么迁移类型选择：结构迁移 + 全量数据迁移。

  对于支持增量迁移的数据库类型，如果需要进行不停机迁移，迁移类型选择：结构迁移 + 全量数据迁移 + 增量数据迁移。

- 迁移对象

  这个步骤选择要迁移的对象。迁移对象的选择粒度细化为：库、表、列三个粒度。默认情况下，对象迁移到目标 RDS 实例后，对象名跟源实例一致。如果您迁移的对象在源实例跟目标实例上名称不同，那么需要使用 DTS 提供的对象名映射功能，详细使用方式请参考库表列映射。

图 389: 选择迁移对象及类型



4. 预检查，如图 *390:* 预检查所示。

   在迁移任务正式启动之前，会先进行前置预检查，只有预检查通过后，才能成功启动迁移。具体的预检查内容请参见预检查简介。

如果预检查失败，那么可以单击具体检查项后的按钮，查看具体的失败详情，并根据失败原因修复后，重新进行预检查。

图 390: 预检查



5. 启动迁移任务。

   当预检查通过后，可以启动迁移任务，任务启动成功后，可以在任务列表中查看迁移的具体状态及迁移进度。

   如果选择了增量迁移，那么进入增量迁移阶段后，源库的更新写入都会被 DTS 同步到目标 RDS 实例。迁移任务不会自动结束。如果您只是为了迁移，那么建议在增量迁移无延迟的状态时，源库停写几分钟，等待增量迁移再次进入无延迟状态后，停止掉迁移任务，直接将业务切换到目标 RDS 实例上即可。

   至此，完成 ECS 自建数据库到 RDS 实例数据迁移任务的配置。

至此，完成将本地 ECS 自建数据库迁移到 RDS 的数据迁移工作。如果您选择了增量迁移，那么增量迁移过程中，如果不结束迁移，任务会一直处于迁移中。建议在增量迁移无延迟的状态时，源库停写几分钟，等待增量迁移再次进入无延迟状态后，停止掉迁移任务，将业务切换到目标 RDS 实例上即可。

18.3.12 从MySQL迁移至DRDS

本小节简单介绍使用 DTS 将 MySQL 数据迁移到 DRDS 的数据迁移任务配置流程。

使用数据传输 DTS 可以实现 MySQL 到 DRDS 的数据迁移。同时，通过增量数据迁移，可在 MySQL 不停服的情况下，将数据平滑迁移到目标 DRDS 实例。

数据源

DTS能够支持下面几种实例类型的 MySQL 到 DRDS 的迁移：

- 本地 IDC 自建 MySQL->DRDS
- ECS 自建 MySQL->DRDS
- RDS->DRDS

迁移对象

由于 DRDS 只支持表，所以DTS 支持的迁移对象包括：

- 单表（包括所有索引定义）
- 分区表

迁移权限要求

当使用 DTS 进行 MySQL 到 DRDS 实例的数据迁移时，不同迁移类型，对源跟目标数据库的迁移账号权限要求不同。

- 本地 IDC 自建 MySQL->DRDS 实例

  本地 IDC 自建 MySQL 到 DRDS 实例的数据迁移，迁移账号权限要求如表 *60:* 迁移类型及权限要求所示。

  表 60: 迁移类型及权限要求

迁移类型	结构迁移	全量迁移	增量迁移
本地 IDC 自建 MySQL	select	select	superselectreplication slavereplication client
目的 DRDS	读写权限	读写权限	读写权限

- ECS 自建 MySQL->DRDS实例

  ECS 自建 MySQL 到 DRDS 实例的数据迁移，迁移账号权限要求如表 *61:* 迁移类型及权限要求所示。

表 61: 迁移类型及权限要求

迁移类型	结构迁移	全量迁移	增量迁移
ECS 上自建 MySQL	select	select	superselectreplication slavereplication client
目的 DRDS	读写权限	读写权限	读写权限

- RDS 实例->DRDS实例

  RDS 实例到 DRDS 实例的数据迁移，迁移账号权限要求如表 *62:* 迁移类型及权限要求所示。

表 62: 迁移类型及权限要求

迁移类型	结构迁移	全量迁移	增量迁移
源 RDS 实例	读写权限	读写权限	读写权限
目的 DRDS	读写权限	读写权限	读写权限

迁移步骤

下面以 RDS 到 DRDS 数据迁移为例，介绍 DTS 支持 DRDS 数据迁移的任务配置。自建 MySQL 到 DRDS 的迁移，除了源实例连接方式不同外，其他配置均相同。

迁移账号创建

迁移任务配置时，需要提供源 RDS 实例及目的 DRDS 实例的迁移账号。迁移账号的相关权限请参见上面的迁移权限要求一节。

如果 RDS 尚未创建迁移账号，请参考 RDS实例账号创建创建账号，并将待迁移库的读写权限授权给迁移账号。

迁移任务配置

当上面的所有前置条件都配置完成后，就可以开始正式的数据迁移了。下面详细介绍迁移任务配置流程。

1. 登录数据传输控制台，单击右上角的**创建迁移任务**，开始迁移任务配置。
2. 配置源及目录实例连接信息。

   这个步骤主要配置迁移任务名称、源 RDS 连接信息及目标 DRDS 实例的连接信息。其中：

   - 任务名称

DTS 为每个任务自动生成一个任务名称，任务名称没有唯一性要求。您可以根据需要修改任务名称，建议为任务配置具有业务意义的名称，便于后续的任务识别。

- 源实例信息
  - 实例类型：这个示例中选择 RDS 实例；如果是本地 IDC 的自建 MySQL，那么选择**有公网IP的自建数据库**；如果是 ECS 上的自建 MySQL ，那么选择**ECS上的自建数据库**。
  - 实例地域：选择 RDS 实例所在地域；如果是本地 IDC 的自建 MySQL，那么选择离自建 MySQL 物理距离最近的地域；如果是 ECS 上的自建 MySQL，那么选择 ECS 所在地域。
  - RDS 实例 ID：配置源 RDS 实例的实例 ID。DTS 支持经典网络、VPC 网络的 RDS 实例。
  - 数据库账号：连接 RDS 实例的访问账号。
  - 数据库密码：上面数据账号对应的密码。
- 目标实例信息
  - 实例类型：选择**DRDS实例**
  - 实例地区：选择 DRDS 实例所在地域
  - DRDS 实例 ID：配置迁移的目标 DRDS 实例的实例 ID。 DTS 支持经典网络、VPC 网络的 DRDS 实例。
  - 数据库名称：待迁入数据的 DRDS 数据库名称。
  - 数据库账号：连接 DRDS 数据库的账号。
  - 数据库密码：上面数据库账号对应的密码。

当配置完连接信息后，单击右下角**授权白名单并进入下一步**进行白名单授权。这个步骤 DTS 会将 DTS 服务器的 IP 地址添加到 RDS 及 DRDS 实例的白名单中，避免因为 RDS 或 DRDS 实例设置了白名单，DTS 服务器连接不上实例导致迁移失败。

3. 选择迁移对象及迁移类型。
   - 迁移类型

     迁移类型包括结构迁移、全量数据迁移及增量数据迁移。

     如果要做全量迁移，那么选择结构迁移 + 全量数据迁移。

     如果要做不停机迁移，那么选择结构迁移 + 全量数据迁移 + 增量数据迁移。
   - 迁移对象

     迁移对象，需要选择您要迁移的对象。迁移对象选择的粒度可以为：库、表、列三个粒度。

默认情况下，迁移对象迁移到目标库后，对象名跟源库一致。如果您迁移的对象在源实例跟目标实例上名称不同，那么需要使用 DTS 提供的对象名映射功能，详细使用方式请参考库表列映射。

4. 预检查，如图 *391:* 预检查所示。

   在迁移任务正式启动之前，会先进行前置预检查，只有预检查通过后，才能成功启动迁移。具体的预检查内容请参见预检查简介。

   如果预检查失败，那么可以单击具体检查项后的按钮，查看具体的失败详情，并根据失败原因修复后，重新进行预检查。

   图 391: 预检查



5. 启动迁移任务。

   当预检查通过后，可以启动迁移任务，任务启动成功后，可以在任务列表中查看迁移的具体状态及迁移进度。

   如果选择了增量迁移，那么进入增量迁移阶段后，源库的更新写入都会被 DTS 同步到目标 DRDS 实例。迁移任务不会自动结束。如果您只是为了迁移，那么建议在增量迁移无延迟的状态时，源实例停写几分钟，等待增量迁移再次进入无延迟状态后，停止掉迁移任务，直接将业务切换到目标 DRDS 实例上即可。

   至此，完成 MySQL 到 DRDS 实例的数据迁移任务配置。

18.3.13 库表列映射

本文介绍在数据迁移任务配置过程中，如何使用对象名映射功能。

数据传输 DTS 数据迁移支持对象名映射，即支持源跟目标实例的迁移对象(包括数据库、表或列)的名称不同。

数据库名映射

如果迁移的数据库在源跟目标实例中命名不同，那么可以使用 DTS 提供的对象名映射功能进行数据库名映射。

数据库名映射功能在配置迁移任务的第二个步骤**选择迁移类型及迁移对象**中配置。下面介绍详细的配置步骤。

1. 在**已选择**框中，将鼠标挪到需要进行数据库名映射的数据库对象行上，右侧会浮出**编辑**入口，如图 *392: 编辑入口*所示。

   图 392: 编辑入口



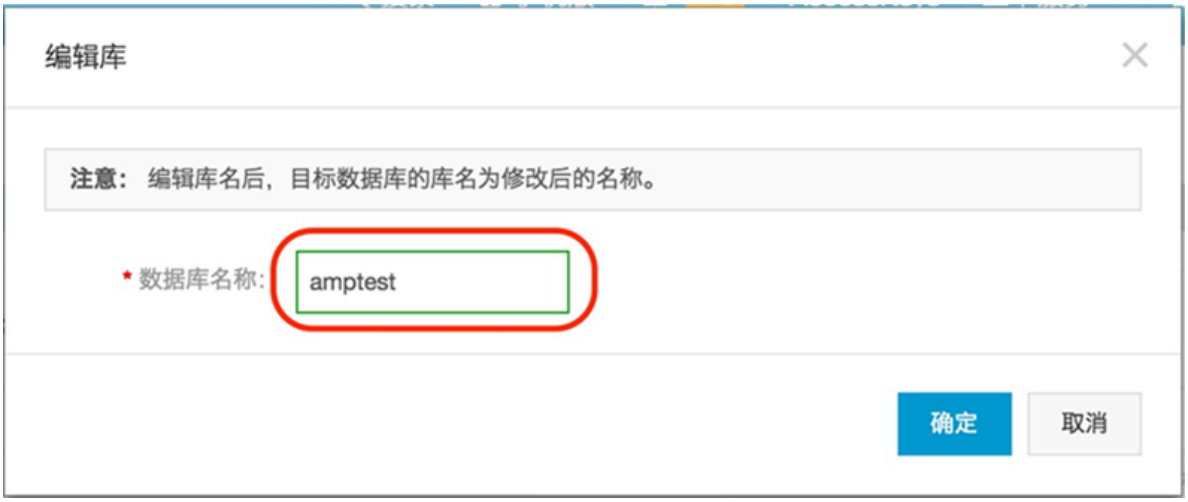
2. 修改库名。

   如果这个数据库迁移到目标实例中库名要修改为 jiangliutest，单击**编辑** ，进入数据库名修改页面。

在数据库名编辑页，数据库名称可以直接修改，修改后的名称即为数据库在目标实例中存储的名称。

假设修改前，数据库名为 amptest，如图 *393: 修改前*所示。

图 393: 修改前



由于数据库迁移到目标实例中库名需要修改成 jiangliutest，所以直接修改界面中的 amptest 为 jiangliutest，如图 *394: 修改后*所示。

图 394: 修改后



数据库名修改完之后，这个数据库迁移到目标实例使用的库名即为修改后的库名。

表名映射

如果迁移的表在源跟目标实例中命名不同，那么可以使用 DTS 提供的对象名映射功能进行数据库名映射。

如果要使用表名映射功能，那么选择迁移对象时，不能直接选择整个库，必须选择具体的表。

除了表，其他的结构对象（例如视图、存储过程、存储函数、同义词等），也可以进行对象名映射，使用相同的方式进行对象名映射。

表名映射功能在配置迁移任务的第二个步骤**选择迁移类型及迁移对象**中配置。下面介绍详细的配置步骤。

1. 在**已选择**框中，将鼠标挪到需要进行表名映射的对象行上，右侧会浮出**编辑** 入口，如图 *392:* 编辑入口所示。
2. 修改表名。

   例如需将待迁移表 amptest 迁移到目标实例中的 jiangliutest，那么 amptest 后面的**编辑**，进入表名修改页面。

   在表编辑页，表名可以直接修改，修改后的名称即为数据库在目标实例中存储的名称。

   假设修改前，数据库名为 amptest，如图 *393:* 修改前所示。

   表迁移到目标实例中库名需要修改成 jiangliutest，所以直接修改界面中的 amptest 为jiangliutest，如图 *394:* 修改后所示。

列名映射

如果要迁移的源跟目标实例的某个表中对应的列名不同，那么可以使用 DTS 的对象名映射功能。

列名映射是在创建迁移任务的第二个步骤**选择迁移类型及迁移对象**中。如果要修改迁移列的名称，那么选择迁移对象时，必须选择这个列对应的表，不能直接选择数据库。下面简单介绍下修改迁移列对应列名的方法。

1. 假设要修改列对应的表为 sbtest1，那么将鼠标挪到界面已选择框中的 sbtest1 对应的行上，此时右边会出现**编辑**按钮，如图 *395:* 表编辑入口所示。

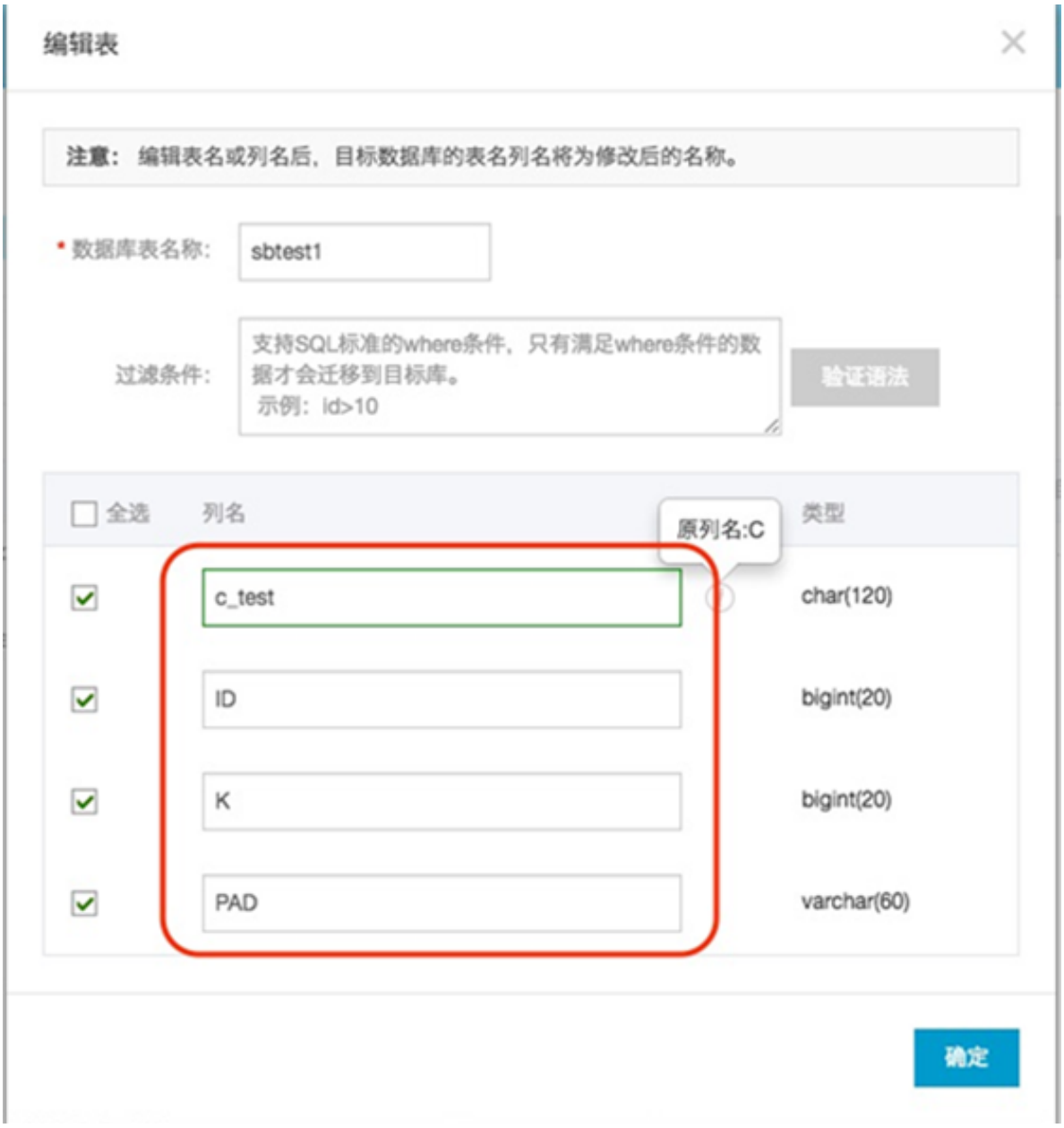
   图 395: 表编辑入口



2. 单击**编辑**，进入**编辑表**界面，如图 *396:* 编辑表界面所示。

   在这个步骤中，对于需要修改列名的列，直接编辑列名即可。修改后，目标库中存储的列名为修改后的列名。

图 396: 编辑表界面



至此，完成列名映射的相关配置。

18.3.14 配置SQL条件过滤迁移数据

本小节简单介绍如何在创建迁移任务时，配置 SQL 条件来过滤需要的迁移数据。

DTS 支持通过配置 SQL 条件来过滤表中需要迁移的数据。这个 SQL 条件只能作用在配置的表上，DTS 会通过这个 SQL 条件来过滤源库中这个表的数据，只有满足这个条件的数据才会被迁移到目标库。这个功能可以应用于定期增量迁移、拆表等多种应用场景。

功能限制

SQL 过滤条件只能作用于全量数据迁移阶段，如果任务的迁移类型选择了**增量数据迁移**，那么 SQL 过滤条件不会应用于增量数据。

配置SQL过滤条件

SQL 过滤条件配置，在迁移任务配置的**选择迁移类型及列表**这个步骤。

如果要配置表的迁移 SQL 过滤条件，那么选择迁移对象时，必须选择表，不能直接选择整个数据库。下面简单介绍下配置 SQL 过滤条件的方法。

配置入口

1. 对于需要配置过滤条件的表，在**选择迁移类型及列表**步骤的**已选择**对象框中，将鼠标挪到需要配置过滤条件的表上，会浮现**编辑**按钮，如图 *397: 编辑入口*所示。

图 397: 编辑入口



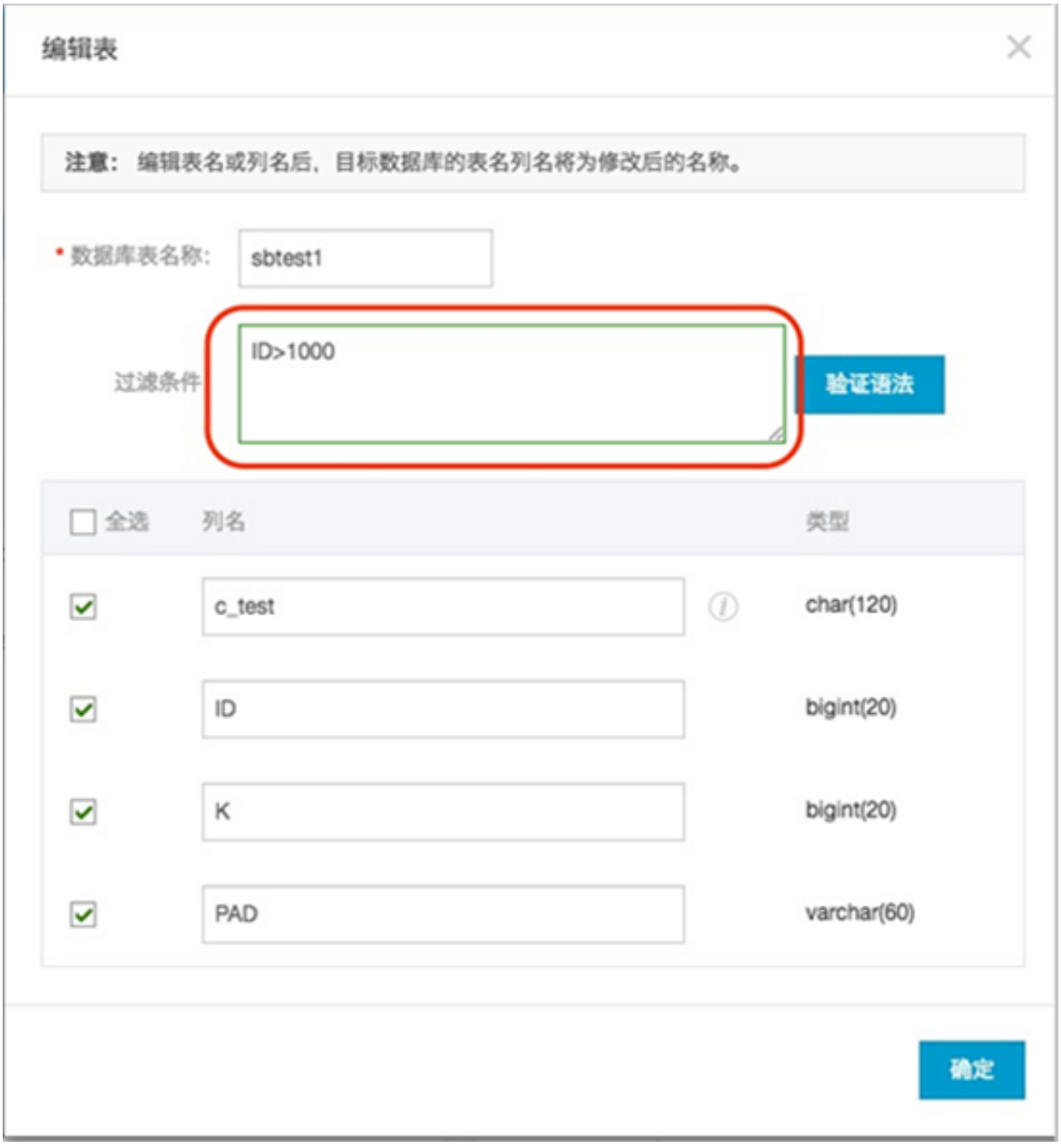
2. 单击**编辑**，即进入过滤条件配置页面。

修改 SQL 过滤条件

DTS 的过滤条件同数据库标准的 SQL WHERE 条件，可以支持运算及简单的函数。

根据需求在过滤条件的编辑框内，填写 SQL 过滤条件，例如我们只需要将表中 id>1000 的记录迁移到目标实例，那么具体配置如图 *398: 修改SQL过滤条件*所示。

图 398: 修改SQL过滤条件



至此，完成数据过滤条件的配置流程。

18.3.15 迁移失败修复

DTS 数据迁移功能在多个阶段都提供了失败在线修复功能。主要包括：

- 结构迁移

由于 DTS 支持异构数据源之间的迁移，例如对于 Oracle->MySQL，对于一些不支持的数据类型，在结构迁移时，直接导入到目标实例，会导致迁移失败。

- 全量迁移

  全量迁移过程中，可能因为目标 RDS 实例空间满、RDS 实例白名单被删除等问题导致迁移失败，这些失败任务都可以通过手动修改后，重启任务来自助恢复任务。

为了解决这些问题，DTS 提供了在线修复功能，当您在迁移过程遇到这些问题时，可以修复续传。下面简单介绍结构迁移和全量迁移过程中的失败修复流程。

结构迁移—失败修复

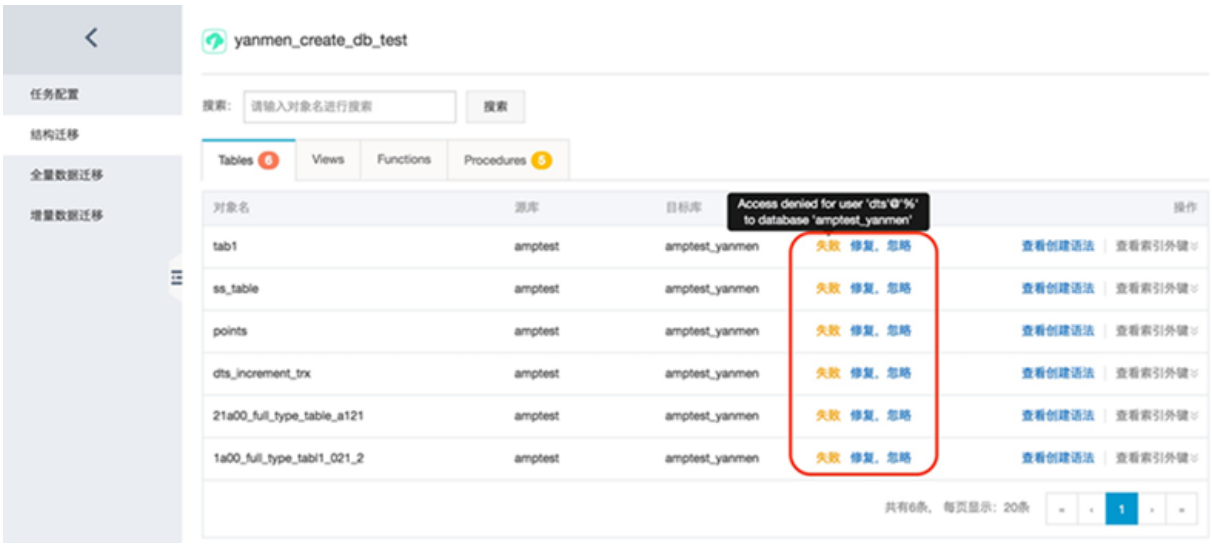
如果结构迁移失败了，任务状态会显示迁移失败，同时状态后，提供**立即修复**入口，如图 *399: 结构迁移异常*所示。

图 399: 结构迁移异常



此时，单击**立即修复**，进入结构对象修复界面，如图 *400: 结构迁移详情页*所示。

图 400: 结构迁移详情页



对于每个对象，如果失败了，单击**修复**，进入对象修复页面，如图 *401: 结构修复页面*所示。

图 401: 结构修复页面



根据失败原因提示，修改结构定义语法，修改完成后，单击**修复**，将修改后的语法重新导入到目标实例。

如果单击**修复**，失败原因变成**修复失败**，且显示修复失败原因，那么说明没有修复彻底。需要根据修复失败原因继续修复，直至修复成功。

如果修复成功，那么会回到结构迁移详情页，且刚才修复的对象的状态变成完成。

当所有的对象全部修复成功后，这个任务就会继续往下执行，例如进入全量迁移阶段。

全量迁移—失败修复

对于下列几种失败原因，DTS 提供了修复重试的功能：

- 源或目标数据库连接失败，确认连接没有问题后，重试任务。
- 源或目标数据库连接超时，确认连接没有问题后，重试任务。
- 目标 RDS 实例空间满，实例被锁定，需要升级 RDS 实例规格或清理实例日志空间后，重试任务。
- 源数据库中部分 Myisam 被损坏，需要 repair 后，重试任务。

其他情况下，如果全量迁移失败，DTS 只提供忽略操作，只能忽略失败对象后，进行其他正常对象的迁移。

如果全量迁移失败了，那么迁移任务的状态为**迁移失败**，状态后面提供**立即修复**入口，如图 *402: 全量迁移失败*所示。

图 402: 全量迁移失败



当出现失败时，单击**立即修复**，进入修复页面。

如果遇到上面可以重试的错误，那么根据错误提示修复问题后，单击全量迁移详情页面的重试按钮，触发任务断点续传。

对于其他失败原因，DTS 只提供**忽略**操作，即忽略掉这个对象的全量数据迁移。一旦单击忽略，那么这个对象的数据不会迁移，但是其他正常对象的数据会迁移到目标实例。

18.4 数据同步

18.4.1 创建RDS实例间数据实时同步作业

本小节介绍如何使用数据传输服务快速创建两个 RDS（MySQL）实例之间的实时同步作业，实现RDS增量数据的实时同步。

支持功能

- 支持阿里云账号下两个 RDS MySQL 实例间的实时同步。
- 支持不同阿里云账号下的 RDS MySQL 实例间的实时同步。

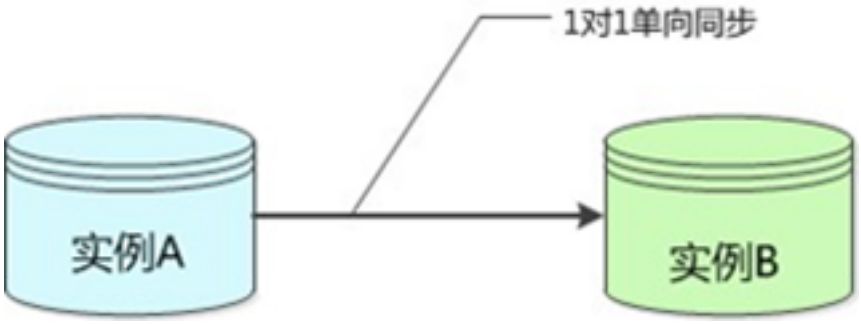
同步限制

- 数据源
  - 目前实时同步只能支持 RDS MySQL 实例，暂不支持其他数据源类型。
  - 目标实例不支持访问模式为标准模式且只有外网连接地址的 RDS 实例。
  - 不支持香港可用区 A 的 RDS 实例的实时同步。
- 同步架构

  目前数据传输服务提供的实时同步功能支持的同步架构有限，其仅能支持如下三种架构：

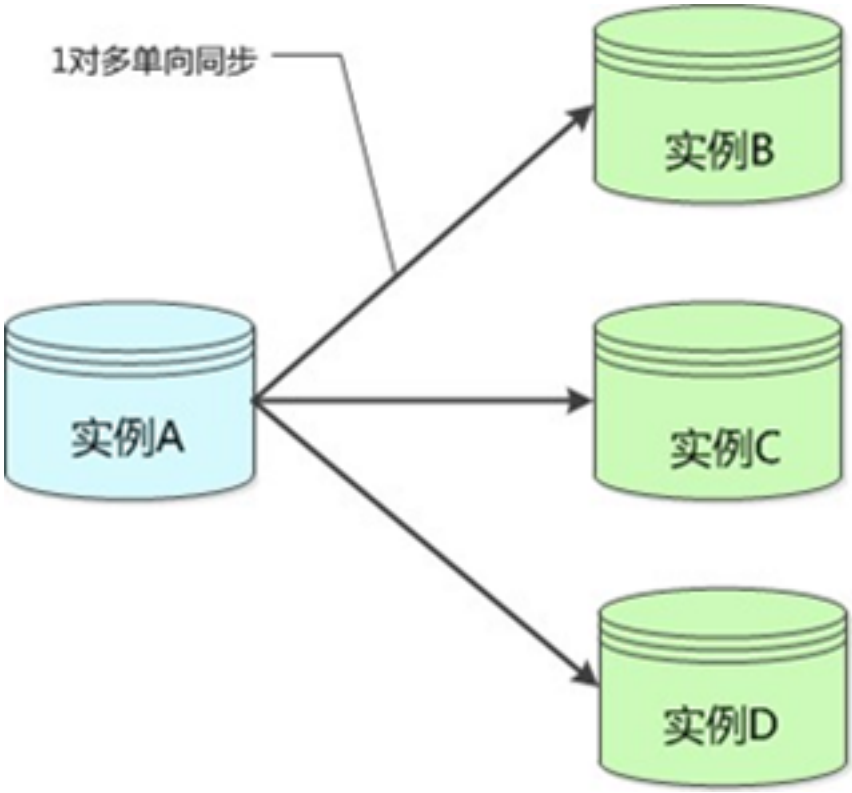
  - A->B 即两个实例之间的单向同步，如图 *403: 1对1单向同步*所示，且要求 B 中同步的对象必须为只读，否则可能导致同步链路异常。

    图 403: 1对1单向同步



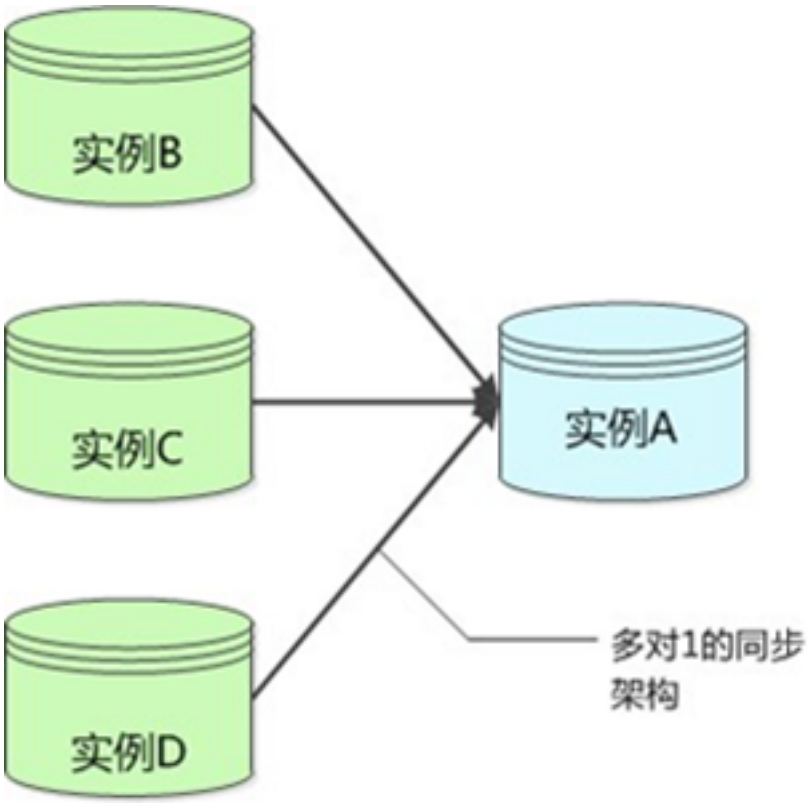
- A->B/C/D 即 1对多的分发式同步架构，如图 *404: 1对多单向同步*所示。这个架构对目标RDS 实例个数没有限制，但是要求目标实例中的同步对象必须为只读，否则可能导致同步链路异常。

图 404: 1对多单向同步



- B/C/D->A 即多对1的数据汇总架构，如图 *405: 多对1的同步架构*所示。对于多对1 的同步架构，要求每个同步链路的同步对象不相同，保证同步完整性。

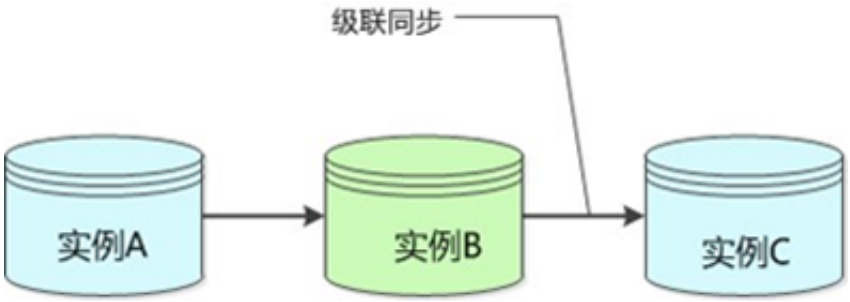
图 405: 多对1的同步架构



对于下面几种架构，暂时不支持：

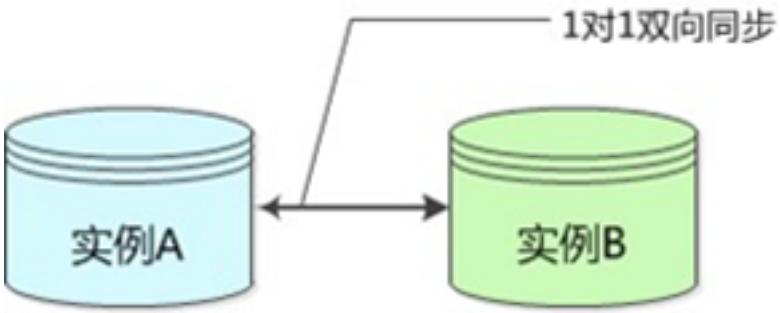
- A->B->C 即级联架构，如图 *406:* 级联架构所示。

图 406: 级联架构



- A->B->A 即实例 A 和实例 B 之间的双向同步架构，如图 *407: 1*对*1*双向同步所示。

图 407: 1对1双向同步



如果您在配置同步链路时，配置了这些不支持的同步架构，那么预检查中的**复杂拓扑**检查项会检查失败。

- 功能限制
  - 不兼容触发器

    如果同步对象为整个库且这个库中包含了会更新同步表内容的触发器，那么可能导致同步数据不一致。

    例如同步库为 A，这个库中存在了两个表 a，b。a 上有一个触发器，触发器内容为在 insert 一条数据到 a 之后，在 b 中插入一条数据。这种情况下，在同步过程中，如果源实例有 a 上的 insert 操作，就会导致b表在源实例跟目标实例数据不一致。

    为了解决这个问题，只能将目标实例中的对应触发器删除掉。b 表的数据由源实例同步过去。
  - rename table 限制

    rename table 操作可能导致同步数据不一致。例如同步对象只包含表 A，不包含表 B，如果同步过程中源实例执行了rename A to B的操作，那么改名后的 B 表的操作不会被同步到目标库。为了解决这个问题，可以选择同步表 A、B 对应的整个数据库。

准备事项

在配置同步作业前，要确保同步作业的源及目标 RDS 实例都已经存在。如果不存在，那么请先创建 RDS 实例。

配置步骤

下面我们详细介绍下创建任意两个 RDS 实例之间的同步链路的具体步骤。

1. 登录数据传输控制台，进入**数据同步**页面。
2. 单击页面右上角的**创建同步作业**，开始作业配置。
3. 在**选择同步通道的源及目标实例**步骤中，配置同步链路连接信息，如图 *408: 配置同步链路信息*所示。

在这一步主要配置：

- 同步作业名称

  同步作业名称没有唯一性要求，主要为了更方便识别具体的作业，建议选择一个有业务意义的作业名称，方便后续的链路查找及管理。

- 源实例信息
  - 实例类型：源 RDS 实例的实例类型，目前只支持 RDS For MySQL，这里选择**RDS实例**（MySQL）。
  - 实例地区：源 RDS 实例所在区域。
  - 实例ID：源 RDS 实例的实例ID。
- 目标实例信息
  - 实例类型：目标RDS 实例的实例类型，这里选择**RDS实例**（MySQL）。
  - 实例地区：目标RDS 实例所在区域。
  - 实例ID：目标RDS 实例的实例ID。

图 408: 配置同步链路信息



说明: 源跟目标 RDS 实例必须为两个不同的实例，选择**实例ID**时，下拉菜单中只列出对应阿里云账号下的 RDS For MySQL 实例。

当这些内容配置完成后，可以单击**授权白名单并进入下一步**。

4. 授权 RDS 实例白名单。

这个步骤，主要是将数据传输服务器 IP 添加到同步 RDS 实例的白名单中。避免因为 RDS 设置了白名单，数据传输服务器连接不上 RDS 导致同步作业创建失败。

为了保证同步作业的稳定性，在同步过程中，请勿将这些服务器 IP 从 RDS 实例的白名单中删除。

5. 当白名单授权后，单击**下一步**，进入同步账号创建。
6. 创建目标库上的同步账号。

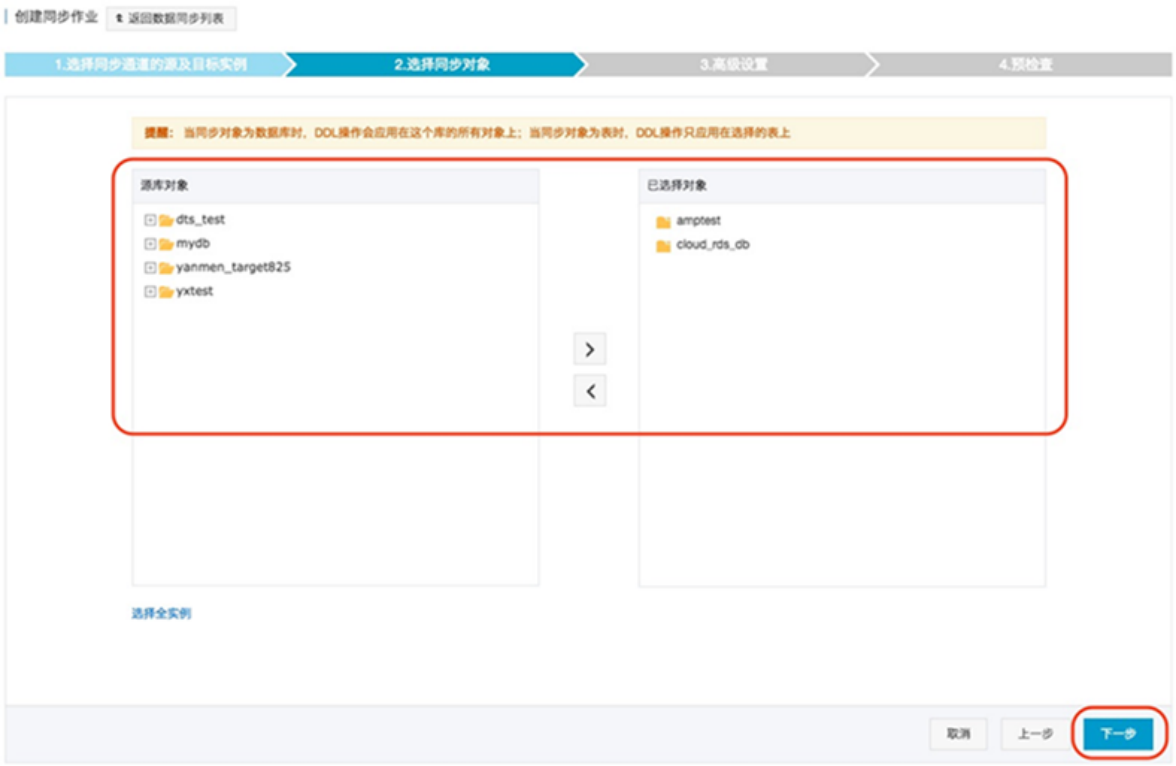
   这个步骤主要是在目标 RDS 实例上创建一个同步账号，账号名字为：dtssyncwriter，在同步过程中，不能删除这个账号，否则会导致同步链路中断。
7. 单击**下一步**，选择同步对象，如图 *409:* 选择同步对象所示。

   当创建完目标 RDS 实例的同步账号后，即进入同步对象的选择步骤。实时同步的同步对象的选择粒度可以支持到表级别，即您可以选择同步某些库或是同步某几张表。

   如果选择的同步对象为整个库，那么这个库中所有对象的结构变更操作（例如 create table，drop view 等），都会同步到目标库。

   如果选择的某张表，那么只有这个表的 drop/alter/truncate/rename table和create/drop index的操作会同步到目标库。

   图 409: 选择同步对象



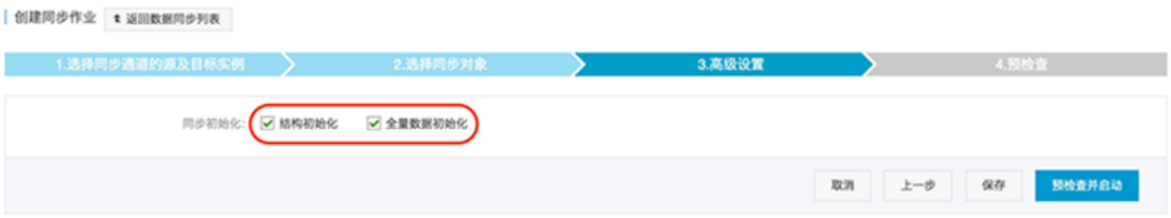
当选择完同步对象后，即进入同步初始化配置。

8. 单击**下一步**，进行同步初始化配置，如图 *410: 同步初始化配置*所示。

   同步初始化配置，初始化是同步链路启动的第一步，它会将源实例中已经存在同步对象的结构及数据在目标实例中初始化，作为后续增量同步数据的基线数据。

   同步初始化类型细分为：结构初始化和全量数据初始化。默认情况下，需要选择**结构初始化**及**全量数据初始化**。

   图 410: 同步初始化配置



9. 单击**预检查并启动**，启动预检查。

   当上面所有选项配置完成后，即进入启动之前的预检查。

   当同步作业配置完成后，数据传输服务会进行限制预检查，当预检查通过后，可以单击**启动**，启动同步作业。

当同步作业启动之后，即进入同步作业列表。此时刚启动的作业处于**同步初始化**状态。初始化的时间长度依赖于源实例中同步对象的数据量大小。当初始化完成后同步链路即进入**同步中**的状态，此时源跟目标实例的同步链路才真正建立完成。

18.4.2 预检查失败修复

实时同步链路在启动之前，会先进行简单的预检查，本小节简单介绍预检查的内容及失败时的修复方式。

源库连接性检查

- 检查内容

  这个检查项主要检查数据传输服务器同源 RDS 实例的连接性。数据传输服务使用 JDBC 协议尝试连接源 RDS实例，如果连接失败，那么这个检查项即检查失败。

- 失败原因
  - 数据传输服务暂不支持源实例所在地区的 RDS 实例的实时同步。
  - 源实例账号密码有误。
- 修复方法

  提交工单，联系阿里云技术售后，协助解决。

目标库连接性检查

- 检查内容

  这个检查项主要检查数据传输服务器同目标 RDS 实例的连通性。数据传输服务使用 JDBC 协议尝试连接目标RDS 实例，如果连接失败，那么这个检查项即检查失败。

- 失败原因
  - 数据传输服务暂不支持目标实例所在地区的 RDS 实例的实时同步。
  - 目标实例账号密码有误。
- 修复方法

  提交工单，联系阿里云技术售后，协助解决。

源库版本检查

- 检查内容

  这个检查项主要检查：

1. 源 RDS 实例的版本号，实时同步功能是否支持。
2. 目标 RDS 实例的版本号是否高等于源 RDS 实例的版本。

- 失败原因
  - 源 RDS 实例的版本低于数据传输服务支持的版本，实时同步暂时只支持 MySQL5.1、5.5、5.6 三个版本。
  - 目标 RDS 实例版本低于源 RDS 实例版本。
- 修复方法
  - 如果源 RDS 版本过低，那么请首先到 RDS 控制台，将源 RDS 升级到 5.6 版本后，重新创建同步链路。
  - 如果是目标 RDS 实例版本低于源 RDS 实例，那么请先到 RDS 控制台，将目标 RDS 实例升级到 5.6 版本后，重新创建同步链路。

数据库存在性检查

这个检查项主要检查同步 DB，在目标库是否已经存在。如果同步 DB 在目标库不存在，那么数据传输服务会自动创建，但是在如下情况下，自动创建 DB 会出现失败，此时这个检查项还是会提示检查失败。

- 数据库名中含有小写字母、数字、下划线、中划线的其他特殊字符。
- 数据库的字符集不为 utf8、gbk、latin1、utf8mb4 中的一种。
- 目标库迁移账号无同步库的读写权限。

  当源为 RDS 实例时，这个检查项不会失败。

源库权限检查

这个检查项主要检查源库同步账号的权限是否满足要求。当权限不满足要求时，这个检查项会失败。当源为RDS实例时，这个检查项不会出现失败。

目标库权限检查

- 检查内容

  这个检查项主要检查目标库同步账号的权限是否满足要求，当权限不满足时，这个检查项会失败。
- 失败原因
  - 数据传输服务自动在目标 RDS 实例创建账号失败。

- 数据传输服务对目标 RDS 实例的同步账号进行读写权限授权失败。

- 修复方法

  提交工单，联系技术同学协助处理。

同名对象存在性检查

- 检查内容

  这个检查项只有选择了同步初始化的同步链路时会进行检查，它主要检查目标 RDS 实例中是否已经存在跟同步对象同名的对象。

- 失败原因

  当目标 RDS 实例中已经存在跟同步对象相同名称的对象时，这个检查项会失败。

- 修复方法

  - 先将目标库中跟同步对象相同名称的对象删除掉。
  - 重新创建同步链路，同步初始化选择：结构初始化+全量数据初始化。

源库server_id检查

这个检查主要检查源数据库的 server_id 是否设置为大等于 2 的整数。如果源为 RDS 实例，那么这个检查项不会失败。

源库binlog是否开启检查

这个检查项主要检查源数据库是否开启 binlog 日志。如果检查项失败，那么说明源数据库没有开启 binlog。当源为 RDS 实例时，那么这个检查项不会失败。

源库binlog模式检查

这个检查项主要检查源数据库的 binlog 模式是否为 ROW。如果检查失败，那么说明源数据库的 binlog 格式不为 ROW。当源为 RDS 实例时，这个检查项不会失败。

约束完整性检查

- 检查内容

  这个检查项主要检查同步的对象列表中，有外键依赖关系的父子表是否都被同步，防止破坏外键约束完整性。

- 失败原因

当同步的对象中，有部分对象为外键依赖的子表，但是它对应的父表没有同步，破坏了外键约束完整性。

- 修复方法

  对于这种情况，有如下三种修复方式：

  - 不进行约束完整性检查失败的子表的同步，重新创建同步作业，不进行这些约束完整性检查失败的子表的同步。
  - 进行约束完整性检查失败的子表对应的父表的同步，重新创建同步作业，将这些约束完整性检查失败的子表对应的父表添加到同步列表中。
  - 删除约束完整性检查失败的子表的外键依赖，修改源数据库将这些子表的外键依赖删除掉，重新创建同步作业。

存储引擎检查

- 检查内容

  这个检查项主要检查同步的对象中，是否存在实时同步不支持的存储引擎。如federated, mrg_myisam,tokudb。

- 失败原因

  当同步对象中，有表的存储引擎为 federated, mrg_myisam 或 tokudb 时，这个检查项会失败。

- 修复方法

  先将不支持存储引擎的表的存储引擎修改为 Innodb 后，重新创建同步作业。

字符集检查

- 检查内容

  这个检查项主要检查同步的对象中，是否存在实时同步不支持的字符集，例如 ucs2。

- 失败原因

  当同步对象的字符集为不支持的字符集时，这个检查项会失败。

- 修复方法

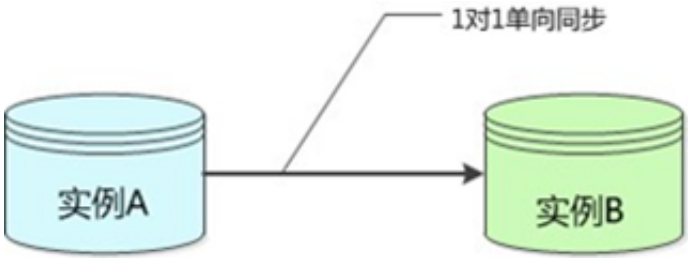
  先将不支持字符集的表的字符集修改为 utf8、gbk、latin1 中的一种后，重新创建同步作业。

复杂拓扑存在性检查

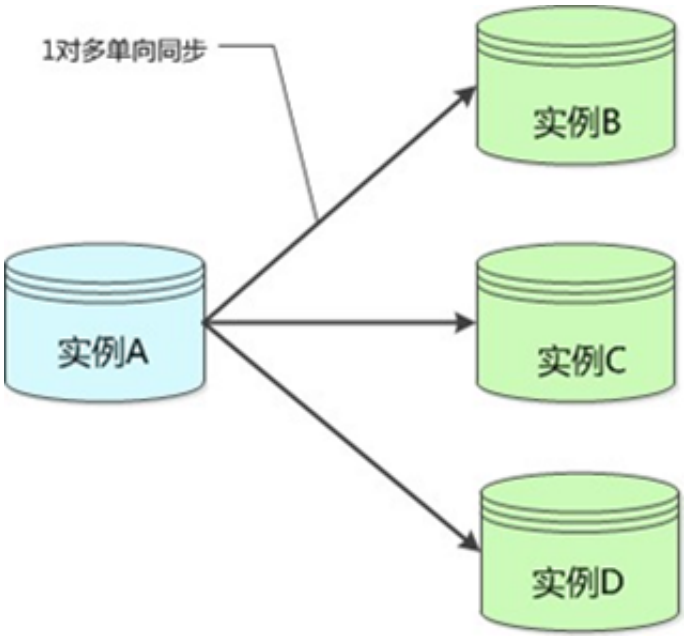
- 检查内容

这个检查项主要检查同步作业的源跟目标 RDS 实例上是否存在不支持的同步架构。目前实时同步只能支持以下两种架构：

- 1 对 1



- 1对多



对于多对1，级联，双向等同步架构暂不支持。

- 失败原因
  - 源 RDS 实例上已经存在同步作业，且其为同步作业的目标实例。
  - 目标 RDS 实例上已经存在同步作业。
  - 源跟目标 RDS 存在正在运行的迁移任务，且这个迁移任务迁移的对象跟要创建的同步作业重叠。
- 修复方法
  - 如果源跟目标 RDS 实例同已存在同步作业相同，创建新同步作业是为了同步新的对象，那么直接修改已经存在的同步链路，修改同步对象，将要添加对象添加进去即可。

- 如果这个同步链路同已经存在的迁移任务冲突，那么必须等迁移任务完成后，才能重新创建同步作业。
- 如果新建同步作业会跟老的同步作业构成：级联架构或双向同步或多对 1 架构，那么暂不支持。

MySQL老密码格式检查

这个检查项，主要检查源实例使用的密码是否为 old password。当源为 RDS 实例时，这个检查项不会失败。

18.4.3 新增同步对象

数据传输服务支持在同步过程中，动态修改同步对象。本小节简单介绍下，如何在同步过程中增加同步对象。

修改限制

只有当同步作业处于**同步中**、**同步失败**的状态时，才可以修改同步对象。

同步起始时间

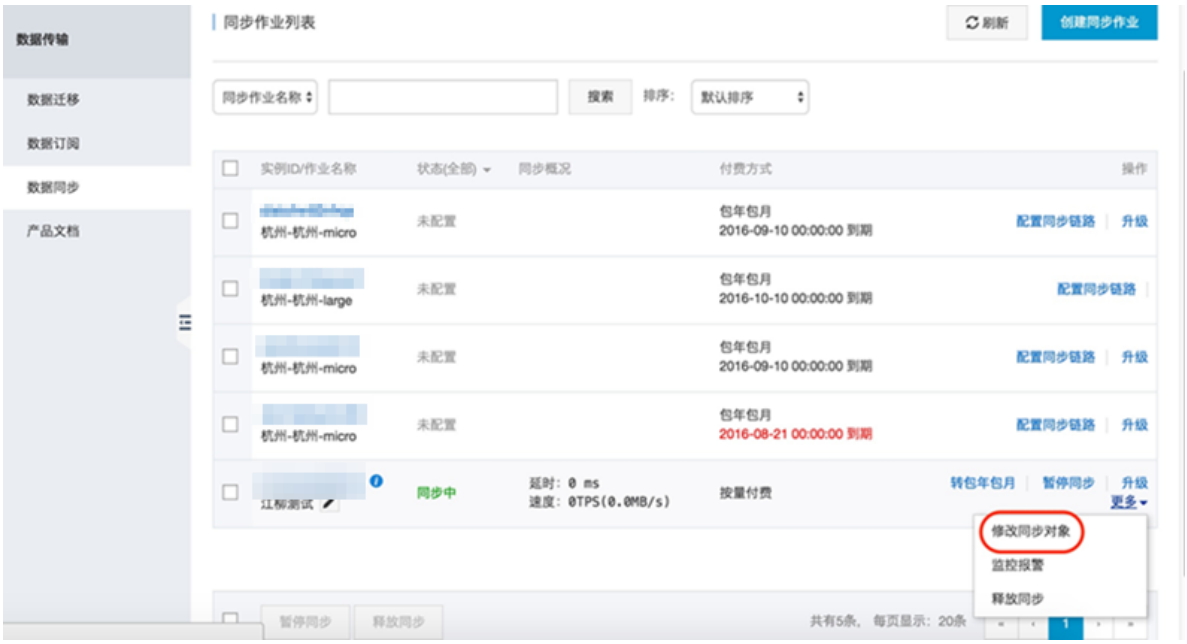
当新增同步对象时，根据同步作业是否选择同步初始化，同步起始时间不同，具体如下：

- 如果同步任务选择了同步初始化，那么新增对象也会先进行同步初始化，然后开始增量同步。
- 如果同步任务未选择同步初始化，那么同步对象会从当前时间点源实例产生的增量数据开始同步。

修改步骤

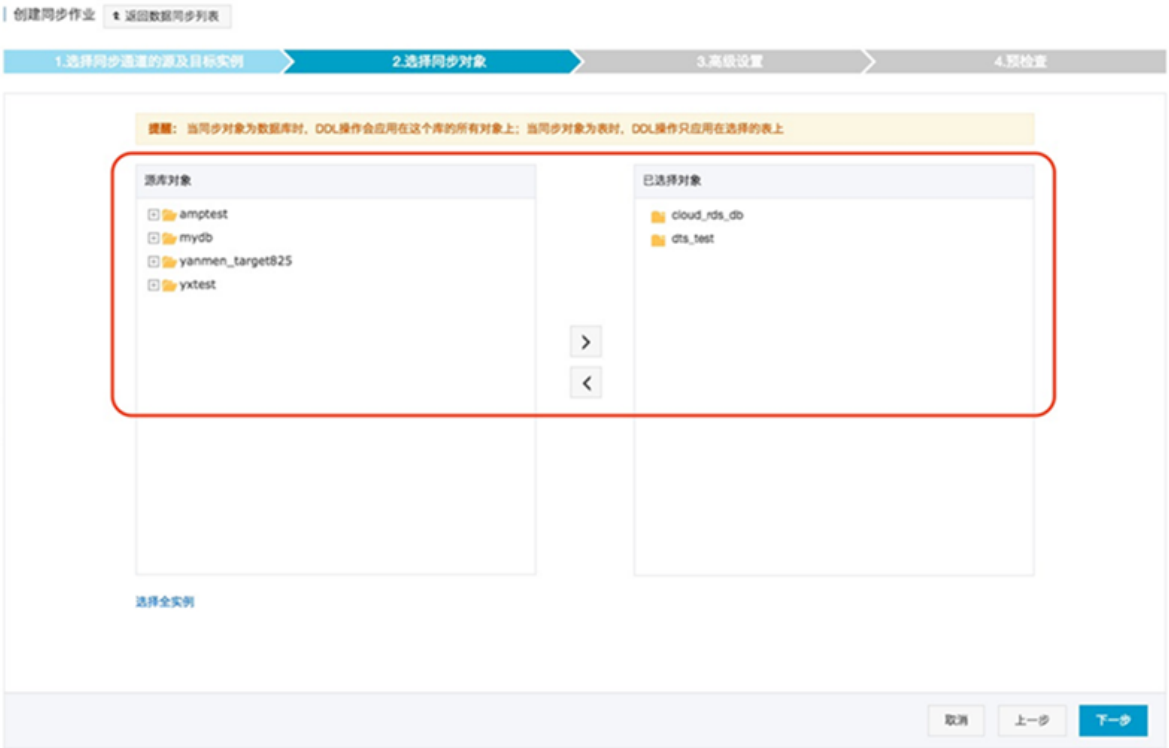
1. 登录数据传输控制台，进入**数据同步**页面。
2. 选择待修改的同步作业，单击**更多** > **修改同步对象**，修改同步对象，如图 *411:* 修改同步对象入口所示。

图 411: 修改同步对象入口



3. 在**选择同步对象**界面中，根据需要添加需要同步的对象，如图 *412:* 添加同步对象所示。

图 412: 添加同步对象



4. 单击**下一步**，进行预检查，如图 *413:* 预检查所示。

图 413: 预检查



当预检查通过后，即可以单击**启动**完成同步对象添加。

当完成同步对象添加后，如果同步作业需要进行初始化，那么这个任务的状态会由原来的**同步中**变成**同步中**（**新增对象初始化中详情**）。此时，后台会重新启动同步作业通道，所以同步延迟会变成 -1s，当重启成功后，延迟及同步速度恢复正常。

在同步作业列表中，可以单击**详情**，查看新增对象初始化的具体进度详情。当新增同步对象完成初始化后，同步作业状态回到同步中。

18.4.4 减少同步对象

数据传输服务支持在同步过程中，动态修改同步对象。本小节简单介绍下，如何在同步过程中减少同步对象。

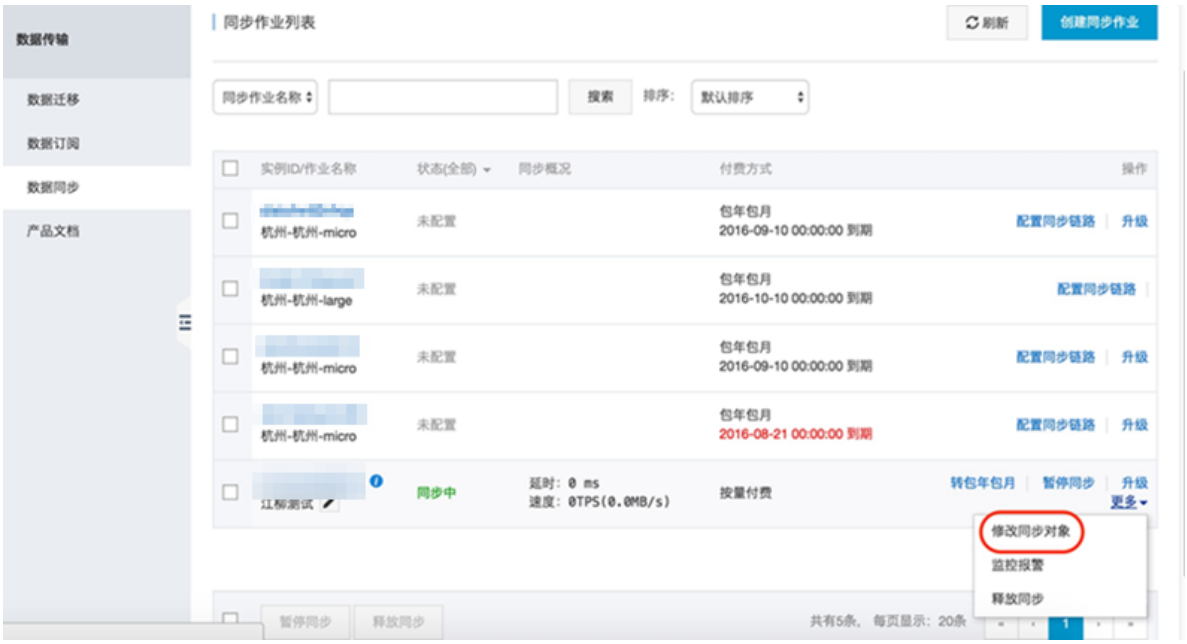
修改限制

只有当同步作业处于**同步中**、**同步失败**的状态时，才可以修改同步对象。

修改步骤

1. 登录数据传输控制台，进入**数据同步**页面。
2. 选择待修改的同步作业，单击**更多** > **修改同步对象**，修改同步对象，如图 *414:* 修改同步对象入口所示。

图 414: 修改同步对象入口



3. 在**选择同步对象**选择界面中，根据需要，删除不需要同步的对象。

   至此，完成减少同步对象的整个配置流程。

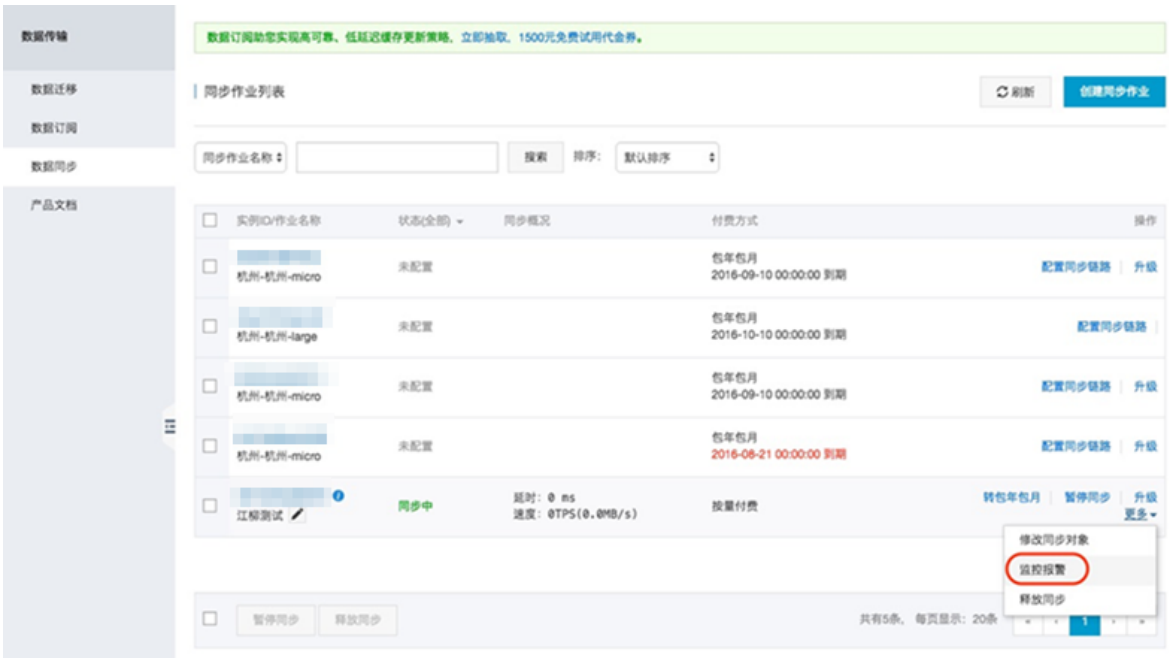
18.4.5 配置监控报警

数据传输服务提供了监控报警的功能，可以监控同步作业的状态及同步延迟。同步作业在任何状态下都可以配置监控报警功能。本小节简单介绍如何在数据传输服务控制台配置同步作业状态及延迟的监控报警。

配置步骤

1. 登录数据传输控制台，进入**数据同步**页面。
2. 选择要配置监控报警的同步作业，单击**更多** > **监控报警**，即可以开始配置，如图 *415:* 监控报警入口所示。

图 415: 监控报警入口



3. 在**监控报警**界面，选择对应的监控项，在后面单击**修改**，如图 *416:* 监控报警界面所示。

   默认情况下，监控项都是开启的状态。如果需要修改报警触发条件及手机联系人，可以单击每个报警项后的**修改**，进行修改。

图 416: 监控报警界面



4. 修改报警策略及联系人。

   如图 *417:* 同步延迟报警项所示，可以修改同步延迟的延迟阈值及延迟报警的手机联系人。

   报警规则：同步延迟 > 10s ，是指当这个同步作业的延迟大于 10s 时，才会向报警手机联系人发送报警短信。您可以根据业务延迟敏感度，配置延迟阈值。

   实时同步支持向多个手机联系人发送报警短信，您可以根据需要添加报警联系人。

图 417: 同步延迟报警项



对于同步状态，不能修改报警规则，如图 *418: 同步状态报警项*所示。只有在同步作业状态异常的情况下才会发送报警短信。同步作业状态异常包括：同步初始化失败、同步失败和新增对象初始化失败。

图 418: 同步状态报警项



当配置完成后，一旦监控项触发报警规则，数据传输服务会自动向报警联系人发送报警短信。

至此，完成监控报警的相关配置。

18.4.6 查看同步性能

数据传输服务提供了同步延迟、同步 TPS、同步流量三种性能趋势图，您可以实时查看同步作业的运行性能状态。

1. 登录数据传输控制台，进入**数据同步**页面。
2. 在同步作业列表中，单击待查看的同步作业的ID。

   进入作业的详情页。
3. 在详情页中，单击左侧导航栏中的**同步性能**。
4. 查看同步性能趋势图。

数据传输服务了同步延迟、同步 TPS、同步流量三种性能趋势图。

- 同步流量：是指每秒钟数据传输服务中的数据写入模块从数据拉取模块中拉取的数据流量，单位为 MB/s。
- 同步 TPS：是指每秒钟数据传输服务同步到目标 RDS 实例的 query 数。
- 同步延迟：是指目标 RDS 实例同步的最新数据的时间戳跟源 RDS 实例当前时间戳的时间差，单位为 s。

18.4.7 跨阿里云账号RDS实时同步

本小节介绍跨阿里云账号 RDS 实例间数据实时同步作业的配置流程。

数据传输支持不同阿里云账号下的两个 RDS 实例之间的实时同步。

支持功能

1. 支持不同阿里云账号下的两个 RDS MySQL 实例间的实时同步。
2. 支持公共云与金融云账号下的 RDS 实例间的数据实时同步。

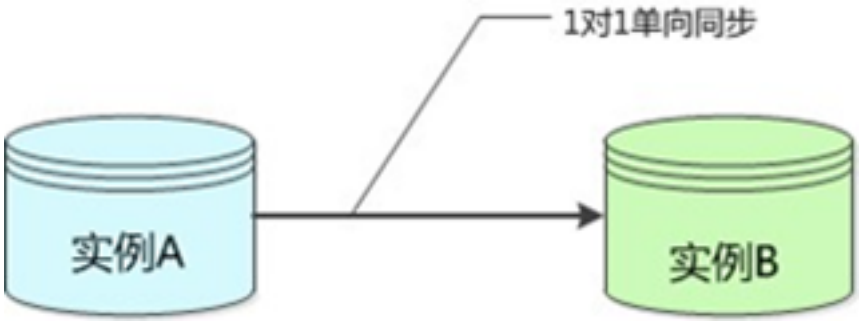
同步限制

- 数据源
  - 目前实时同步只能支持 RDS MySQL 实例，暂不支持其他数据源类型。
  - 目标实例不支持访问模式为标准模式且只有外网连接地址的 RDS 实例。
  - 不支持香港可用区 A 的 RDS 实例的实时同步。
- 同步架构

  目前数据传输服务提供的实时同步功能支持的同步架构有限，其仅能支持如下三种架构：

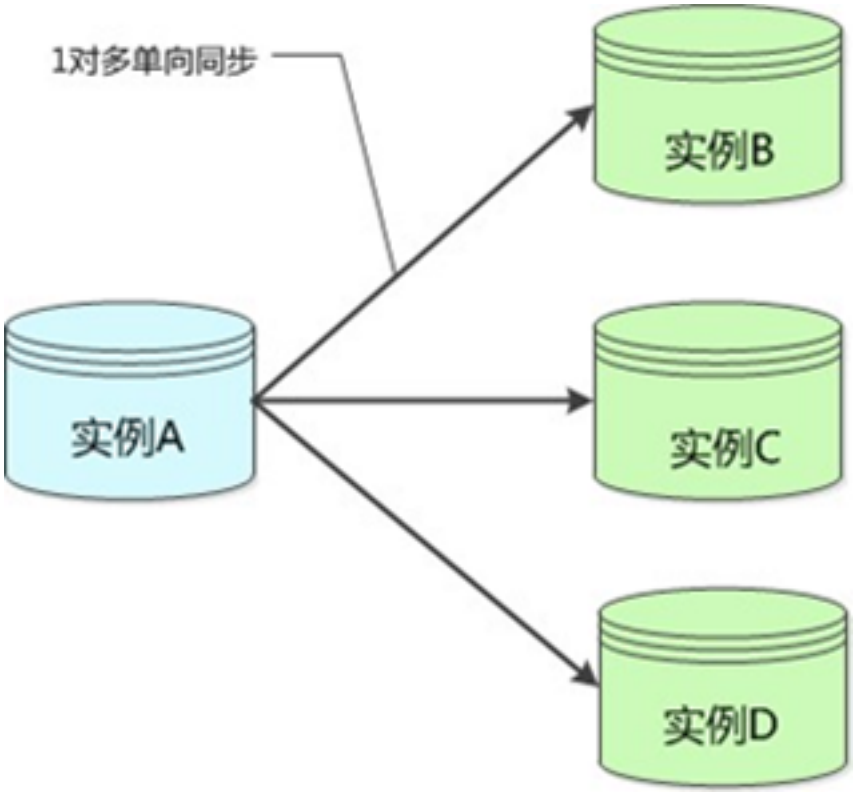
  - A->B 即两个实例之间的单向同步，如图 *419: 1对1*单向同步所示，且要求 B 中同步的对象必须为只读，否则可能导致同步链路异常。

    图 419: 1对1单向同步



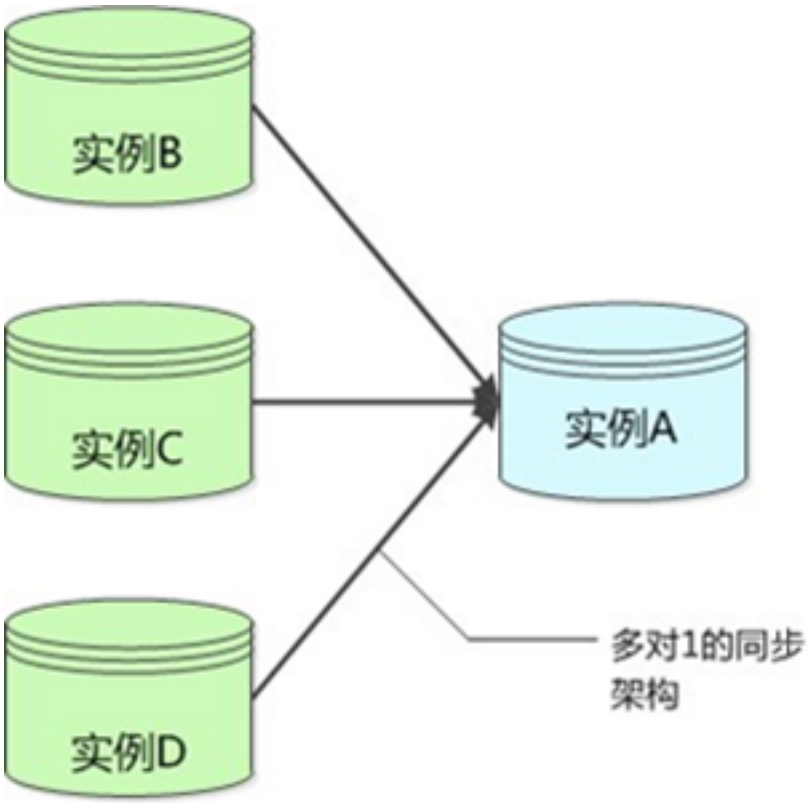
- A->B/C/D 即 1对多的分发式同步架构，如图 *420: 1对多单向同步*所示。这个架构对目标RDS 实例个数没有限制，但是要求目标实例中的同步对象必须为只读，否则可能导致同步链路异常。

图 420: 1对多单向同步



- B/C/D->A 即多对1的数据汇总架构，如图 *421: 多对1的同步架构*所示。对于多对1 的同步架构，要求每个同步链路的同步对象不相同，保证同步完整性。

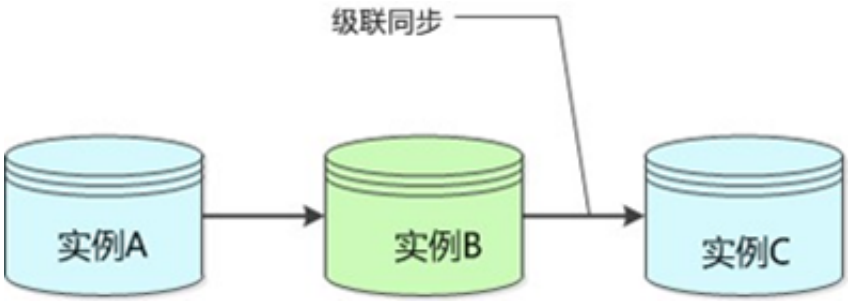
图 421: 多对1的同步架构



对于下面几种架构，暂时不支持：

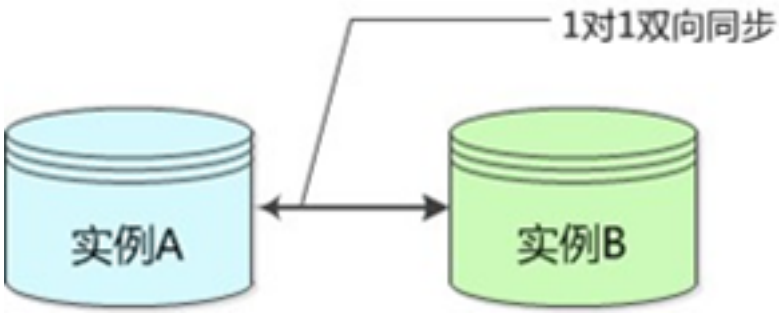
- A->B->C 即级联架构，如图 *422: 级联架构*所示。

  图 422: 级联架构



- A->B->A 即实例 A 和实例 B 之间的双向同步架构，如图 *423: 1对1双向同步*所示。

  图 423: 1对1双向同步



如果您在配置同步链路时，配置了这些不支持的同步架构，那么预检查中的**复杂拓扑**检查项会检查失败。

- 功能限制
  - 不兼容触发器

    如果同步对象为整个库且这个库中包含了会更新同步表内容的触发器，那么可能导致同步数据不一致。

    例如同步库为 A，这个库中存在了两个表 a，b。a 上有一个触发器，触发器内容为在 insert 一条数据到 a 之后，在 b 中插入一条数据。这种情况下，在同步过程中，如果源实例有 a 上的 insert 操作，就会导致b表在源实例跟目标实例数据不一致。

    为了解决这个问题，只能将目标实例中的对应触发器删除掉。b 表的数据由源实例同步过去。
  - rename table 限制

    rename table 操作可能导致同步数据不一致。例如同步对象只包含表 A，不包含表 B，如果同步过程中源实例执行了rename A to B的操作，那么改名后的 B 表的操作不会被同步到目标库。为了解决这个问题，可以选择同步表 A、B 对应的整个数据库。

准备事项

在配置同步作业前，要确保同步作业的源及目标 RDS 实例都已经存在。如果不存在，那么请先创建 RDS 实例。

配置步骤

1. 登录数据传输控制台，进入**数据同步**页面。
2. 单击页面右上角的**创建同步作业**，开始作业配置。
3. 在**选择同步通道的源及目标实例**步骤中，配置同步链路连接信息。

   在这一步主要配置：

- 同步作业名称

  同步作业名称没有唯一性要求，主要为了更方便识别具体的作业，建议选择一个有业务意义的作业名称，方便后续的链路查找及管理。

- 源实例信息
  - 实例类型：源 RDS 实例的实例类型，目前只支持 RDS For MySQL，这里选择**RDS实例**（MySQL）。
  - 实例地区：源 RDS 实例所在区域。
  - RDS实例ID：源 RDS 实例的实例ID。

    由于源实例不属于登录的阿里云账号，所以源实例配置时，单击界面右侧的**其他阿里云账号下的RDS实例**，然后配置。
  - 数据库账号：源RDS实例的数据库账号。
  - 数据库密码：源RDS实例的数据库密码。
- 目标实例信息
  - 实例类型：目标RDS 实例的实例类型，这里选择**RDS实例**（MySQL）。
  - 实例地区：目标RDS 实例所在区域。
  - RDS实例ID：目标RDS 实例的实例ID。

当这些内容配置完成后，可以单击**授权白名单并进入下一步**。

4. 授权 RDS 实例白名单。

   这个步骤，主要是将数据传输服务器 IP 添加到同步 RDS 实例的白名单中。避免因为 RDS 设置了白名单，数据传输服务器连接不上 RDS 导致同步作业创建失败。

   为了保证同步作业的稳定性，在同步过程中，请勿将这些服务器 IP 从 RDS 实例的白名单中删除。
5. 当白名单授权后，单击**下一步**，进入同步账号创建。
6. 创建目标库上的同步账号。

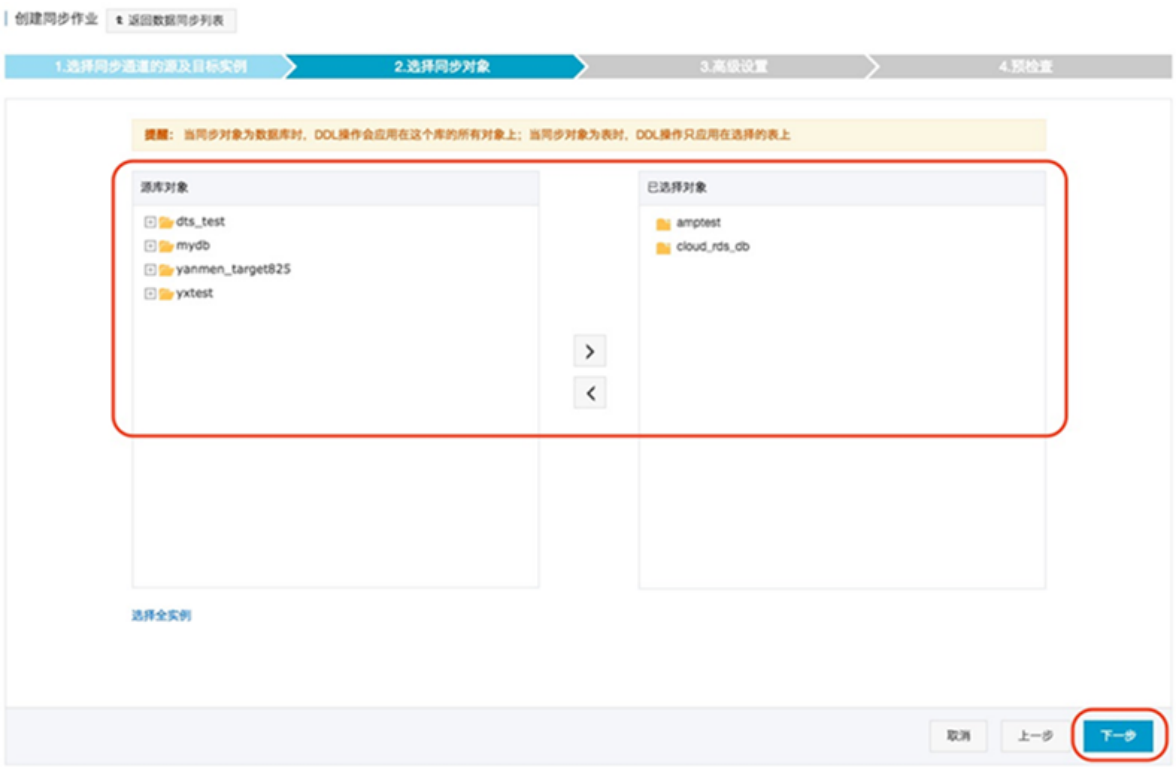
   这个步骤主要是在目标 RDS 实例上创建一个同步账号，账号名字为：dtssyncwriter，在同步过程中，不能删除这个账号，否则会导致同步链路中断。
7. 单击**下一步**，选择同步对象，如图 *424:* 选择同步对象所示。

   当创建完目标 RDS 实例的同步账号后，即进入同步对象的选择步骤。实时同步的同步对象的选择粒度可以支持到表级别，即您可以选择同步某些库或是同步某几张表。

如果选择的同步对象为整个库，那么这个库中所有对象的结构变更操作（例如 create table，drop view 等），都会同步到目标库。

如果选择的某张表，那么只有这个表的 drop/alter/truncate/rename table和create/drop index的操作会同步到目标库。

图 424: 选择同步对象



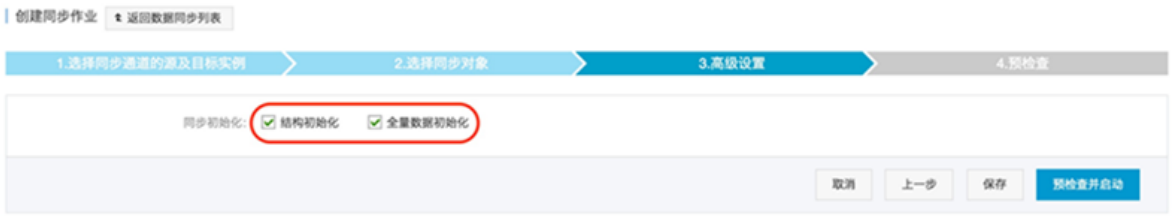
当选择完同步对象后，即进入同步初始化配置。

8. 同步初始化配置，如图 *425:* 同步初始化配置所示。

同步初始化配置，初始化是同步链路启动的第一步，它会将源实例中已经存在同步对象的结构及数据在目标实例中初始化，作为后续增量同步数据的基线数据。

同步初始化类型细分为：结构初始化和全量数据初始化。默认情况下，需要选择**结构初始化**及**全量数据初始化**。

图 425: 同步初始化配置



9. 单击**预检查并启动**，启动预检查。

   当上面所有选项配置完成后，即进入启动之前的预检查。

   当同步作业配置完成后，数据传输服务会进行限制预检查，当预检查通过后，可以单击**启动**，启动同步作业。

   当同步作业启动之后，即进入同步作业列表。此时刚启动的作业处于**同步初始化**状态。初始化的时间长度依赖于源实例中同步对象的数据量大小。当初始化完成后同步链路即进入**同步中**的状态，此时源跟目标实例的同步链路才真正建立完成。

19 数据管理

19.1 关系型数据库

19.1.1 功能总览

功能界面

关系型数据库的界面如图 *426: 关系型数据库*所示。

图 426: 关系型数据库



各功能模块如表 *63: 各功能模块说明*所示。

表 63: 各功能模块说明

编号	名称	内容说明
1	顶部菜单栏	DMS各个功能模块的主要入口。
2	数据库切换下拉框	通过切换数据库，访问不同库的表及其他数据对象。
3	数据库对象导航按钮	根据需要在表、视图、可编程对象（函数，存储过程，触发器，事件）
4	表搜索框	通过模糊搜索来快速找到您需要表，进行操作。

编号	名称	内容说明
5	DMS对象列表	可查看数据库对象（如表）的详细信息。
6	常用数据库操作的快捷导航	方便进行数据库的操作。
7	实例健康状态报告	反映数据库服务当前的健康状态。

支持的数据库类型

- DMS for MySQL
- DMS for SQLServer
- DMS for PostgreSQL/PPAS

支持的数据库操作

- SQL操作
  - SQL窗口
  - SQL命令窗口
  - 工作环境保存
  - SQL执行
  - SQL优化
  - SQL格式化（SQL语句美化）
  - 执行计划查看
  - SQL输入智能提示
- 数据库对象操作
  - 数据表操作。
  - 表结构操作：增加表、修改结构、删除表。
  - 表数据变更：插入数据、更新数据、删除数据。
  - 表数据查询与可视化编辑。
- 视图和可编程对象操作（视图、函数、存储过程、触发器、事件）
  - 新增
  - 修改
  - 删除

  - 启用/禁用
- 数据处理
  - 数据导入
  - 数据导出
- 性能与诊断
  - 实时性能
  - 实时会话
  - 锁等待分析
- 数据处理工具
  - E-R 图
  - 表数据量统计
  - 批量操作表

良好的用户交互体验

操作简单，对于操作中出现的错误，会给出操作改进文案，以协助完成数据操作。

19.1.2 SQL操作

19.1.2.1 使用命令窗口

背景信息

这里以MySQL数据库为例进行说明。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在顶部菜单栏上，选择**SQL操作** > **命令窗口**，打开命令窗口。

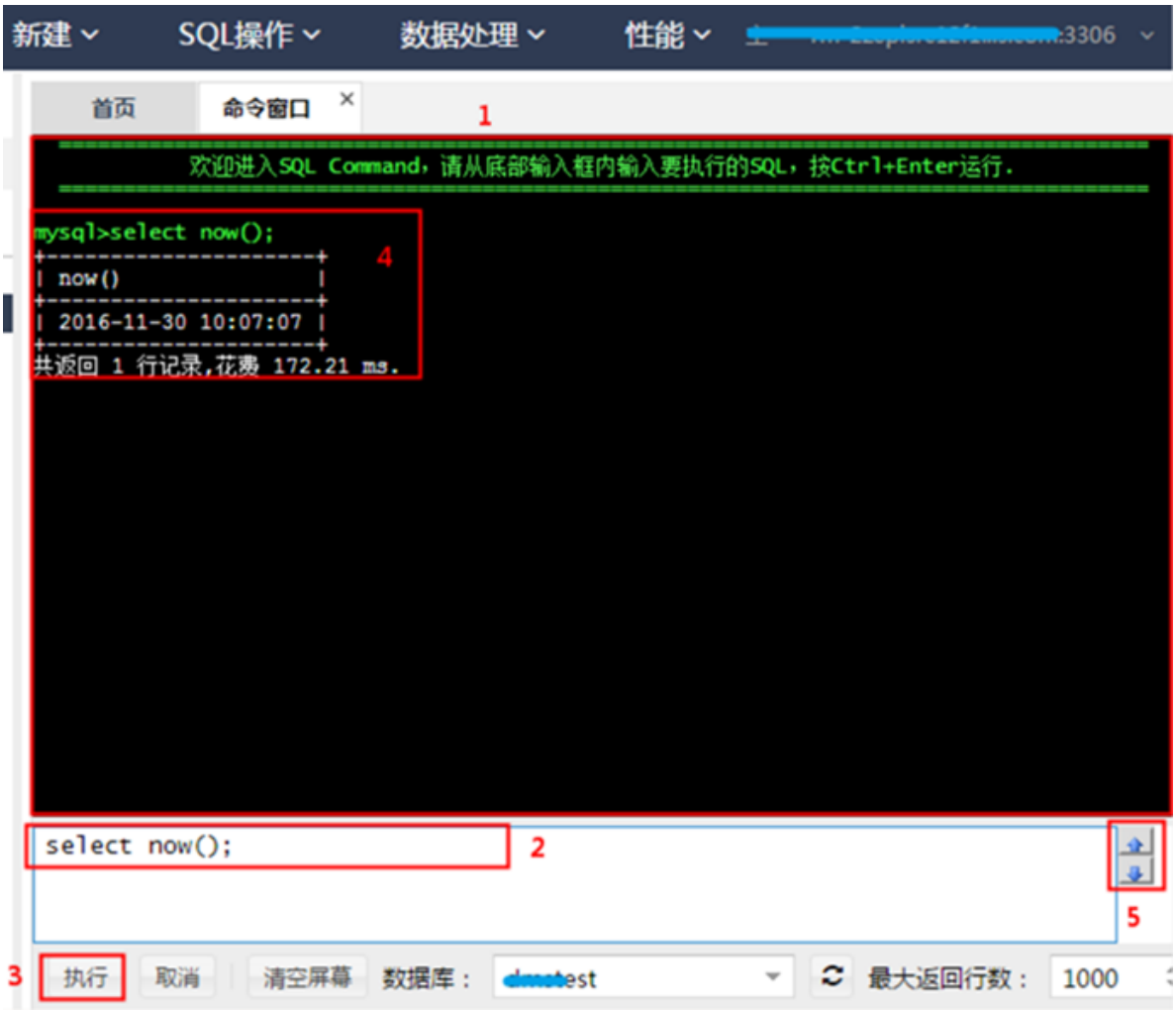
   界面将切换成一个空的命令窗口，如图 *427:* 命令窗口所示。

   图 427: 命令窗口



3. 输入SQL，单击**执行**，使用命令窗口，如图 *428: 执行SQL语句*所示。

图 428: 执行SQL语句



图中各编号说明如表 *64:* 各编号说明所示。

表 64: 各编号说明

编号	名称	说明
1	命令窗口	显示命令执行结果。
2	SQL输入窗口	用于输入SQL命令。
3	**执行**按钮	执行输入的SQL命令。
4	结果显示区	DMS将执行的结果追加到**结果区域**。
5	**上下键**按钮	您可以通过进行历史输入的导航，来快速执行之前执行过的SQL。

4. （可选）如果执行时间过长，不符合预期，您可以通过单击**取消**按钮来取消执行。
5. （可选）单击**清空屏幕**，可以清除结果，方便后续的结果展示。

如果想使用别的数据库，可以通过**数据库**下拉框来切换。

19.1.2.2 使用SQL窗口

19.1.2.2.1 打开空的SQL窗口

背景信息

- 这里以MySQL数据库为例进行说明。
- DMS最多支持打开20个SQL窗口，建议打开的SQL窗口不多于 5 个。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在顶部菜单栏上，选择**SQL操作** > **SQL窗口**，打开SQL窗口。

   打开空的**SQL窗口**后，如图 *429: SQL*空窗口所示。

图 429: SQL空窗口



SQL窗口各编号说明如表 *64:* 各编号说明所示。

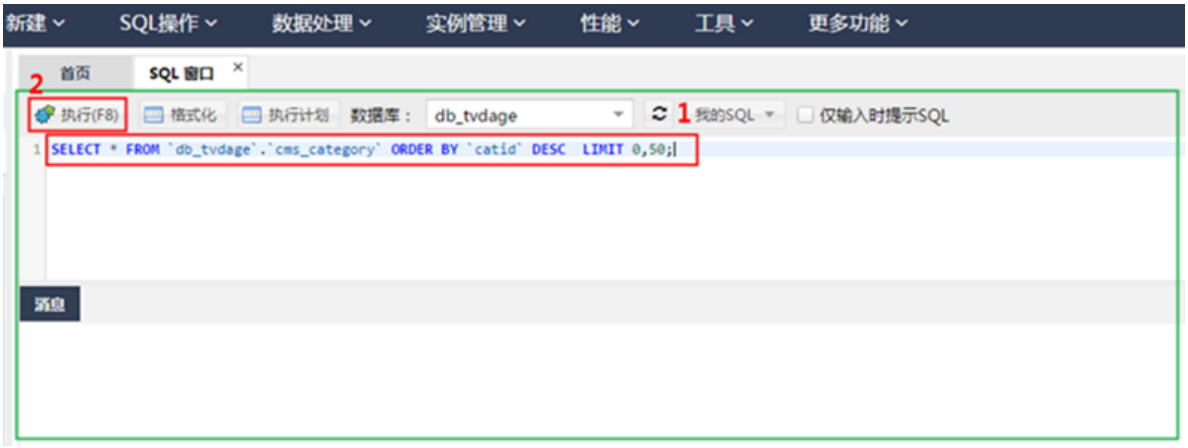
表 65: SQL窗口各编号说明

编号	名称	说明
1	SQL窗口	绿框部分表示的是SQL窗口的主体。

编号	名称	说明
2	**执行(F8)**按钮	用于执行输入的SQL语句。
3	**格式化**按钮	用于格式化输入的SQL语句，格式化后的SQL语句更清晰易读。
4	**执行计划**按钮	用于查看选中的SQL的执行计划，便于优化SQL，提升SQL处理性能。

3. 输入需要执行的SQL，单击**执行**，完成SQL查询或者更新，如图 *430: 执行SQL*所示。

图 430: 执行SQL



4. 查看SQL执行的结果集，如图 *431: 查看结果集*所示。

图 431: 查看结果集

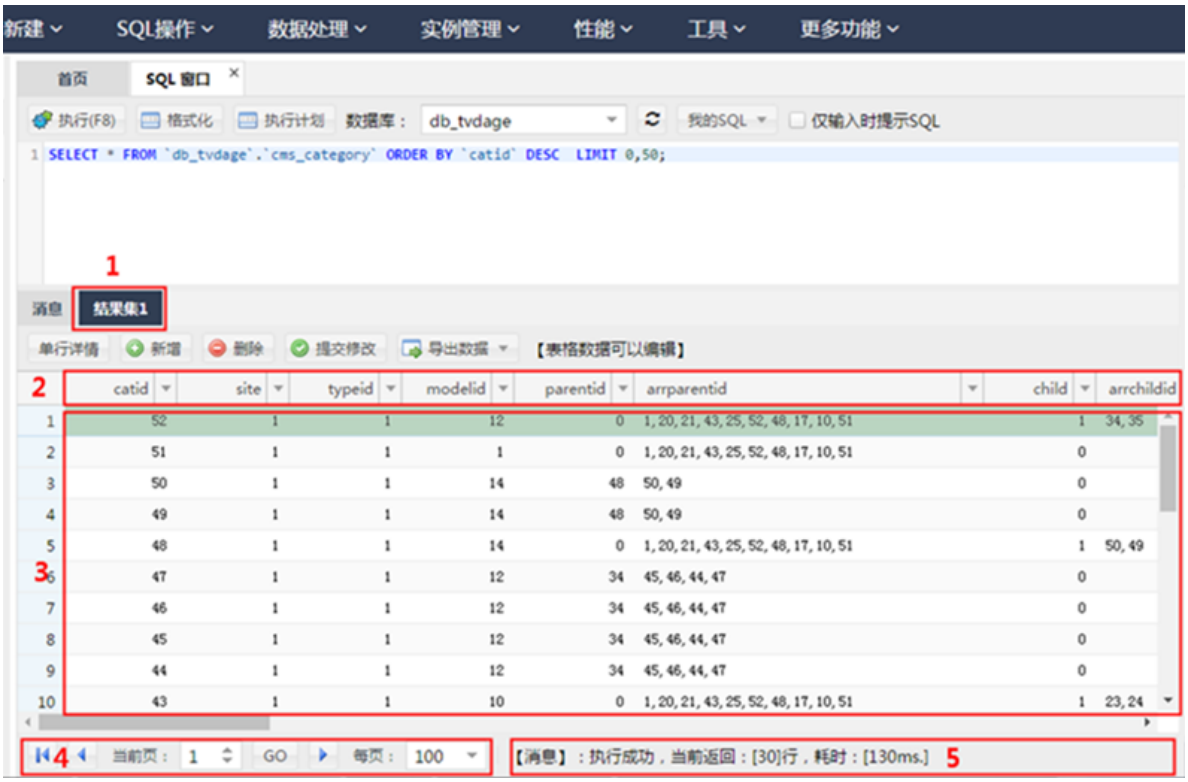


表 66: 结果集窗口各编号说明

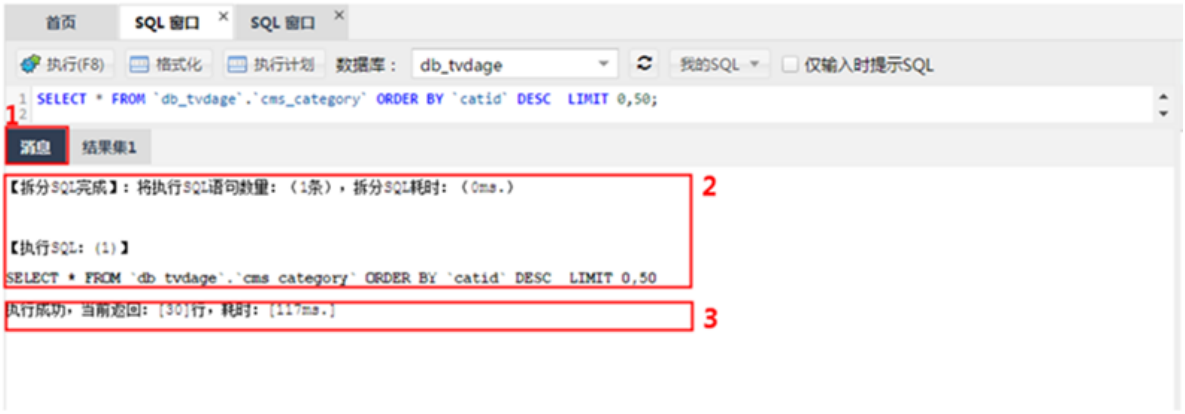
编号	说明
1	**结果集**Tab页面展示SQL查询语句返回的结果集。
2	表格第一行展示字段的名称，如果SQL中为字段指定了别名，则显示字段的别名。
3	表格的数据部分，按查询结果的行顺序展示数据。如果表格太小，数据显示不完全，则会自动出现水平滚动条和垂直滚动条。
4	结果集**分页**和按需**翻页**。 • 查询结果默认每页展示100条，查看多于100条结果需要进行翻页。 • 您可以按需自行设置**每页**展示的数据条数。 • 每次获取到的下一页的数据会追加在**编号3表格数据**的后面。
5	获取数据的进度及耗时信息。

5. 查看SQL执行的消息。

   进行数据查询（select）或者数据订正（insert、update、delete）后DMS都会反馈执行的操作的消息，方便您了解具体的执行情况和执行影响范围。

数据查询操作的消息如图 *432: 数据查询*所示。

图 432: 数据查询



数据订正的操作消息如图 *433: 数据订正*所示。

图 433: 数据订正



数据订正窗口各编号说明如表 *67: 数据订正窗口各编号说明*所示。

表 67: 数据订正窗口各编号说明

编号	说明
1	执行SQL后单击**消息** 标签页即可查看执行的消息。数据订正没有结果集，所以数据订正执行后DMS就直接为您展示消息。
2	DMS执行您输入的SQL的具体步骤。 • 分析您输入的SQL。 • 将分析好的SQL逐个执行到数据库。 • 展示查询到的数据。

编号	说明
	• 统计查询或者影响的数据行数等信息。
3	DMS执行输入的SQL的执行结果。 • 执行是否成功。 • 查询到的行数，或者增加/删除/修改的影响行数。 • 执行SQL消耗的时间。

6. 批量执行多条SQL语句。

   DMS支持一次性批量执行多条SQL语句，如图 *434:* 批量执行所示。

   图 434: 批量执行



   - 编号1：第一条SQL执行的结果展示。
   - 编号2：第二条SQL执行的结果展示。

   a) 在SQL窗口输入要执行的多个SQL，注意每个SQL后面需要带分隔符号，如分号（；）。

   b) 如果只执行部分SQL语句，则选中要执行的SQL语句；如果全部执行，则不选中或者选中所有的SQL语句后，单击**执行** 。

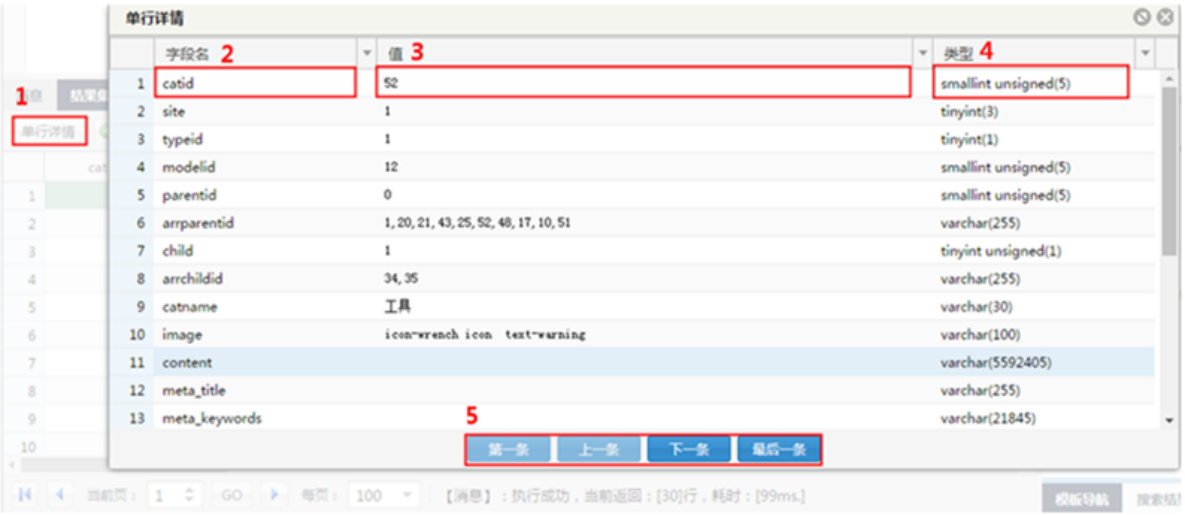
      等待SQL执行操作完成。

   c) 查看执行的结果。

如果是选择语句（select）DMS会显示它的结果集；如果是其他语句，DMS会展示执行的结果，例如：影响行数。

7. 单击**单行详情**，查看结果集的单条记录详细内容，如图 *435:* 单行详情 所示。

图 435: 单行详情



单行详情窗口各编号说明如所示。

表 68: 单行详情窗口各编号

编号	说明
1	在**结果集**表格中选择要展示的单行记录，单击**单行详情** 查看单条数据记录。弹出的**单行详情**对话框展示记录的每个**字段名称**，**字段值**，**字段类型**。
2	**字段名称**，如果为字段指定了别名，则展示字段的别名。
3	**字段的值**，DMS会自动帮您解析好字段的结果并展示，对时间，二进制等数据，会将数据格式化为易读的字符串后展示给您。
4	**字段类型**，包括字段的类型名称和字段长度。
5	记录导航区。通过**上一条**、**下一条**、**第一条**、**最后一条** 按钮，可以方便地查看前面和后面数据的单行详情。

8. （可选）在结果集中对查询到的数据进行编辑。

- 单击 **新增**，可以增加一行数据到当前查询的表中。
- 单击 **删除**，可以删除结果集表格您选中的数据行。

- 选中数据行，对表数据进行操作前需要选中要操作的行。
- 直接更新选中行的字段值。

修改数据后，单击**提交修改** ，将结果保存到数据库。

单击**提交修改** 后，DMS会为您展示保存您的修改需要执行的SQL语句，以便您进行确认，防止误操作造成数据的损失。

单击**确认**，将修改执行到数据库中，达到您预期的结果。

9. 单击**格式化**，对选中的SQL语句进行格式化，转化成易于读写的SQL语句。

- 如果选中了SQL，则只会格式化选中的部分。没有选中则会格式化您输入的所有SQL语句。
- 格式化SQL不会修改SQL的执行逻辑和语义，不影响您的执行，只是将您的SQL转换成标准的易读的SQL。

示例：

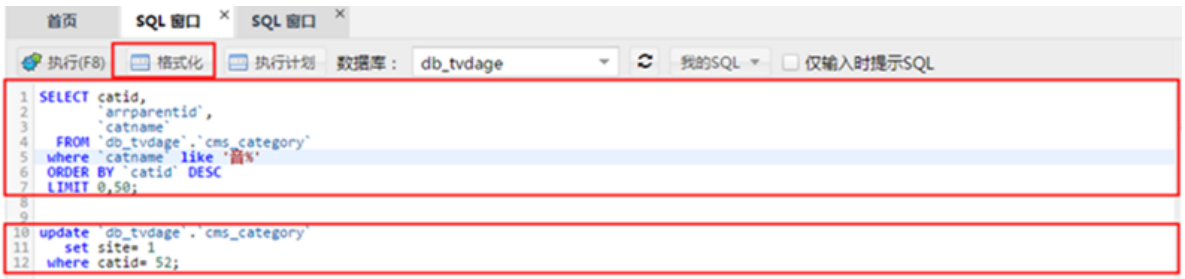
格式化前的SQL，如图 *436: 格式化前的SQL*所示。

图 436: 格式化前的SQL



格式化后的SQL，如图 *437: 格式化后的SQL*所示。

图 437: 格式化后的SQL

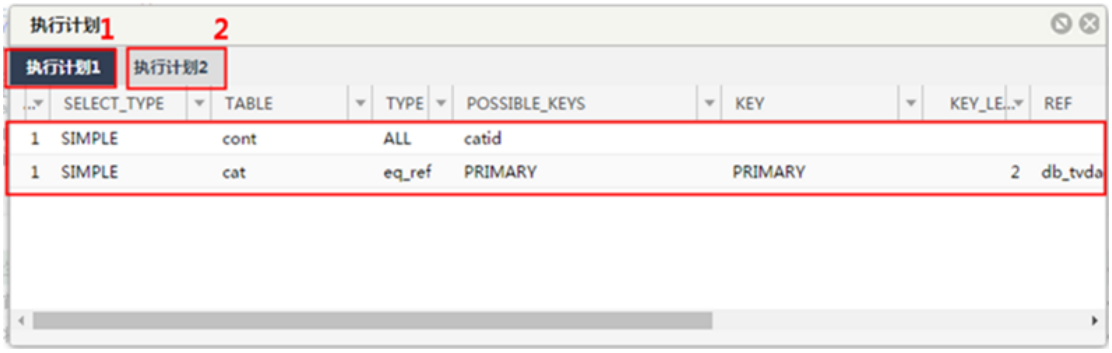


10. 单击**执行计划** ，查看SQL的执行计划，方便排查SQL问题和优化SQL性能。

- 单击**执行计划**，可以查看选中SQL的执行计划。如果没有选中SQL语句，则是查看所有SQL的执行计划。

- 执行计划详情展示。展示执行计划的类型，可能的键等信息。
  - 每种数据库的执行计划展示方式均不太相同，实际展示的执行计划内容跟具体的数据库有关。
  - 如果有多条SQL语句要查询执行计划，DMS会通过不同的Tab页面来展示每一条SQL的执行计划详情，如图 *438:* 执行计划所示。

图 438: 执行计划



- 编号1：第一条SQL语句的执行计划详情。
- 编号2：第二条SQL语句的执行计划详情。

19.1.2.2.2 恢复已保存的SQL窗口

背景信息

- 这里以MySQL数据库为例进行说明。
- DMS最多支持打开（保存）20 个SQL窗口，建议打开的SQL窗口不多于 5 个。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

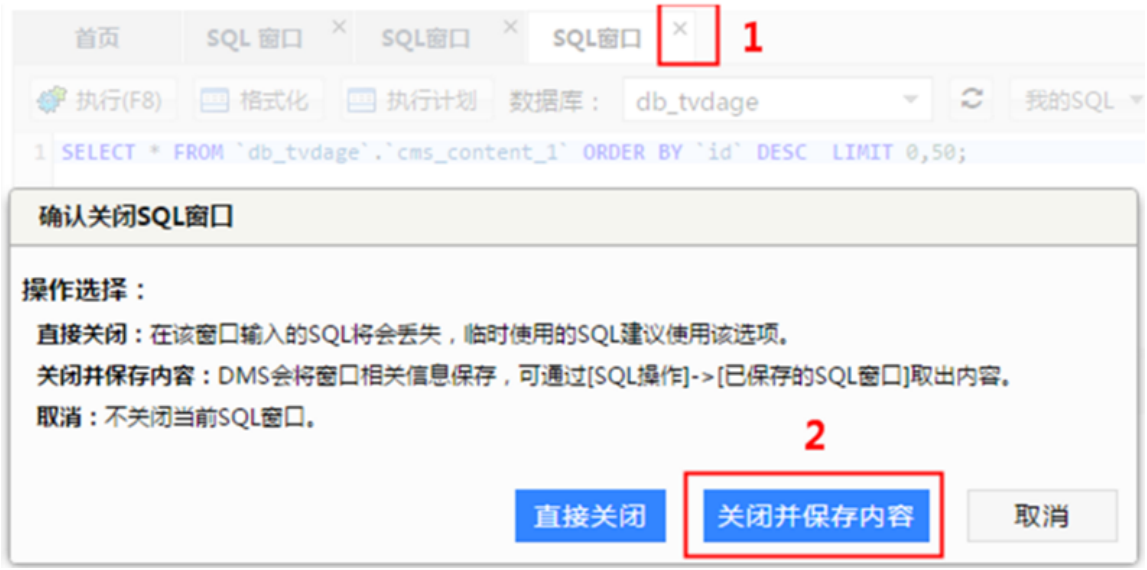
   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在顶部菜单栏上，选择**SQL操作** > **SQL窗口**。
3. 保存当前的SQL窗口操作环境。
   - DMS会在关闭操作页面时自动保存工作环境。
   - 下次登录到DMS时，DMS会自动恢复上次的工作环境，包括上次使用的数据库、打开的SQL窗口和SQL窗口中您上次输入的SQL语句。

- 如果关闭一个SQL窗口， DMS会自动提示保存SQL窗口的内容，如图 *439: 确认关闭SQL窗口*所示。

图 439: 确认关闭SQL窗口



  - 编号1：单击SQL窗口右上角的关闭图标来关闭SQL窗口。
  - 编号2：DMS提醒您保存工作内容。单击 **关闭并保存内容** ，DMS会保存SQL窗口中的工作内容，保存成功后窗口会关闭。

    若直接单击**直接关闭**按钮，DMS不保存SQL窗口工作内容。

4. 恢复已经保存的SQL窗口。

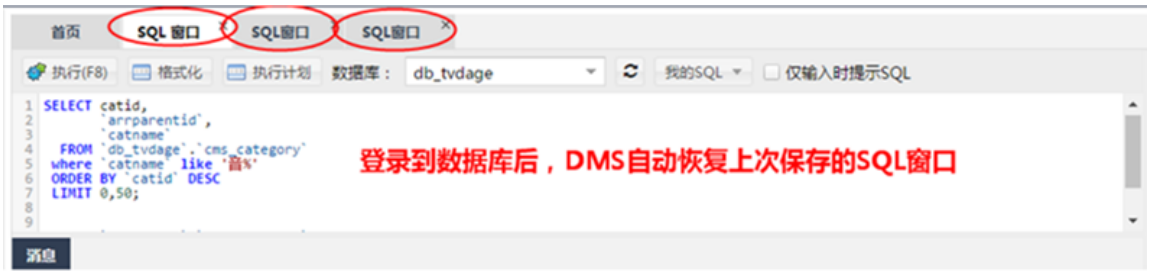
   a) 选择菜单**SQL操作** > **已保存的SQL窗口**。

   DMS展示所有已经保存的SQL窗口。

   b) 单击**新开SQL窗口**，即可恢复已经保存的 SQL窗口。

   c) 当您在DMS上登录到数据库时，DMS会自动恢复上次保存的SQL窗口工作内容，如图 *440: SQL恢复窗口*所示。

图 440: SQL恢复窗口



19.1.2.2.3 管理常用SQL命令

背景信息

这里以MySQL数据库为例进行说明。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 登录到数据库后，在顶部菜单栏上，选择**SQL操作** > **SQL窗口**，打开SQL窗口。

3. 进行如下操作：

   - 增加一条常用SQL

     选择**我的SQL** > **添加我的SQL**，可以增加一条常用的SQL。

     - 适用范围：指在任何情况下都能使用这个自定义的常用SQL。
     - 所有库：无论通过DMS登录哪个数据库，都可以看到这个自定义的SQL。
     - 当前实例：只有通过使用目前连接的这个实例（ip+port）才能看到这个自定义的SQL。
     - 当前库：只有通过使用目前连接的这个库才能看到这个自定义的SQL。如果切换了数据库，则选择**我的SQL** > **选择我的SQL**，无法看到这个SQL。

   - 查看已保存的SQL

     选择**我的SQL** > **选择我的SQL**，可以查看您已保存的常用SQL。

   - 管理您的SQL

     选择**我的SQL** > **管理我的SQL**，可以对常用的SQL进行管理。

     - 在**管理我的SQL**页面，单击**编辑**或**删除**可以对我的SQL进行操作。
     - 在**管理我的SQL**页面，单击**新增**，新增我的SQL。

- 双击其中一条**我的SQL**可以将我的SQL插入到**SQL窗口**中并选中。

19.1.2.2.4 使用SQL模板

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

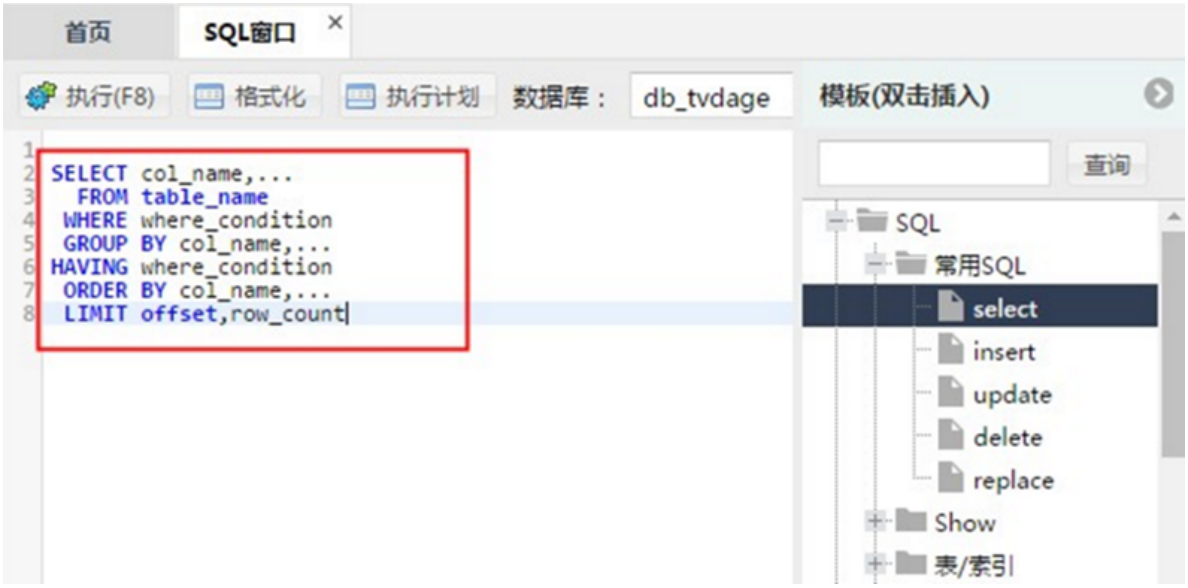
   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 登录数据库后，在顶部菜单栏上，选择**SQL操作** > **SQL窗口**，打开SQL窗口。

   SQL模板在SQL窗口的最右侧。

3. 双击命令或者单击将命令拖拽到SQL窗口中后，可以使用或参考该SQL命令，如图 *441: SQL*窗口所示。

   图 441: SQL窗口



   修改模板中的命令，便于您在不熟悉命令的情况下使用该命令。

19.1.2.2.5 生成图表

前提条件

只有登录到RDS数据库才能使用该功能。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

登录数据库。

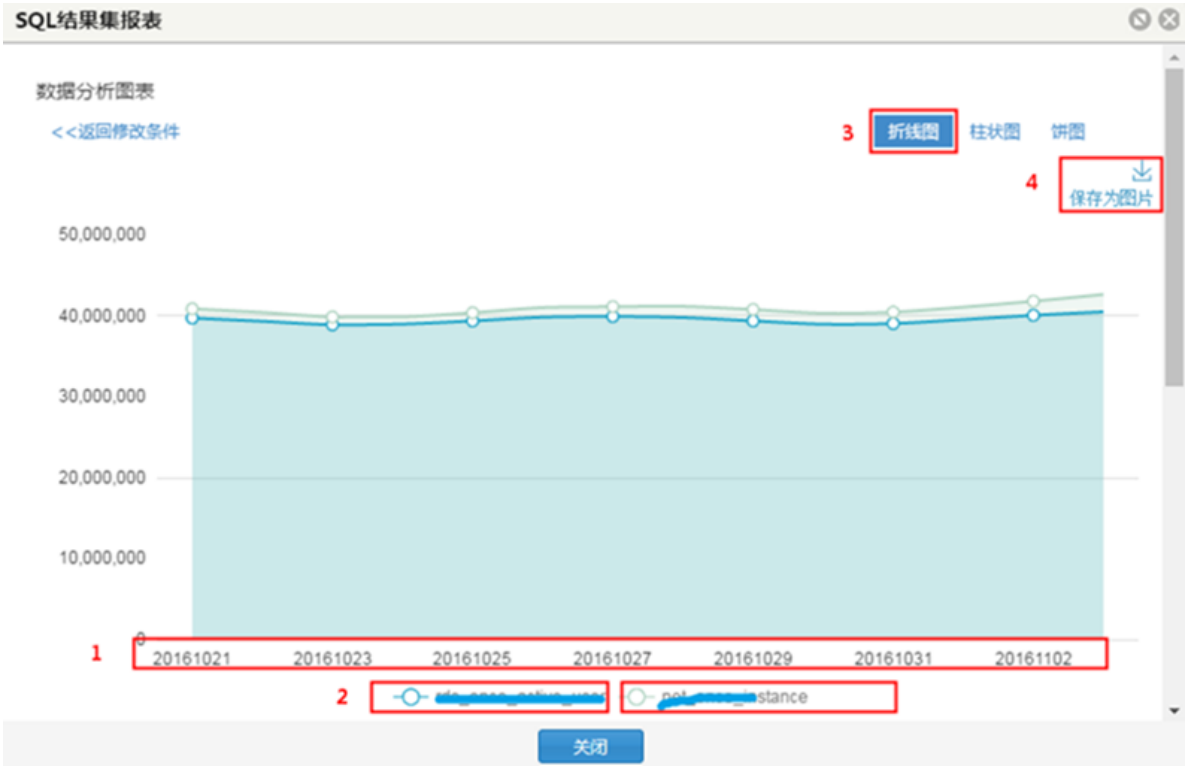
通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在顶部菜单栏上，选择**SQL操作** > **SQL窗口**，打开SQL窗口。
3. 输入SQL查询语句。
4. 单击**生成报表**。

   系统显示**SQL结果集报表**页面。
5. 设置报表维度。
   - 选择进行统计的维度，如用户ID，时间等。
   - 选择要度量的对象，如访问次数，商品销量等。
6. 单击**分析并生成图表**。

   DMS会为刚刚查询出来的数据结果生成图表，如图 *442:* 图表所示。

图 442: 图表



结果集报表窗口各编号说明如表 *69:* 结果集报表窗口各编号说明所示。

表 69: 结果集报表窗口各编号说明

编号	说明
1	横坐标是图表的统计维度，如时间、用户ID。
2	图标的统计指标，单击指定即可显示或者隐藏该指标的度量数值。
3	可以根据需要自由选择折线图、柱状图、饼图来生成图表。
4	单击**保存为图片**，可以将生成的图片保存到您的计算机器上。

19.1.2.3 表操作（基于表目录树）

19.1.2.3.1 打开基于表的SQL窗口

背景信息

这里以MySQL数据库为例进行说明。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

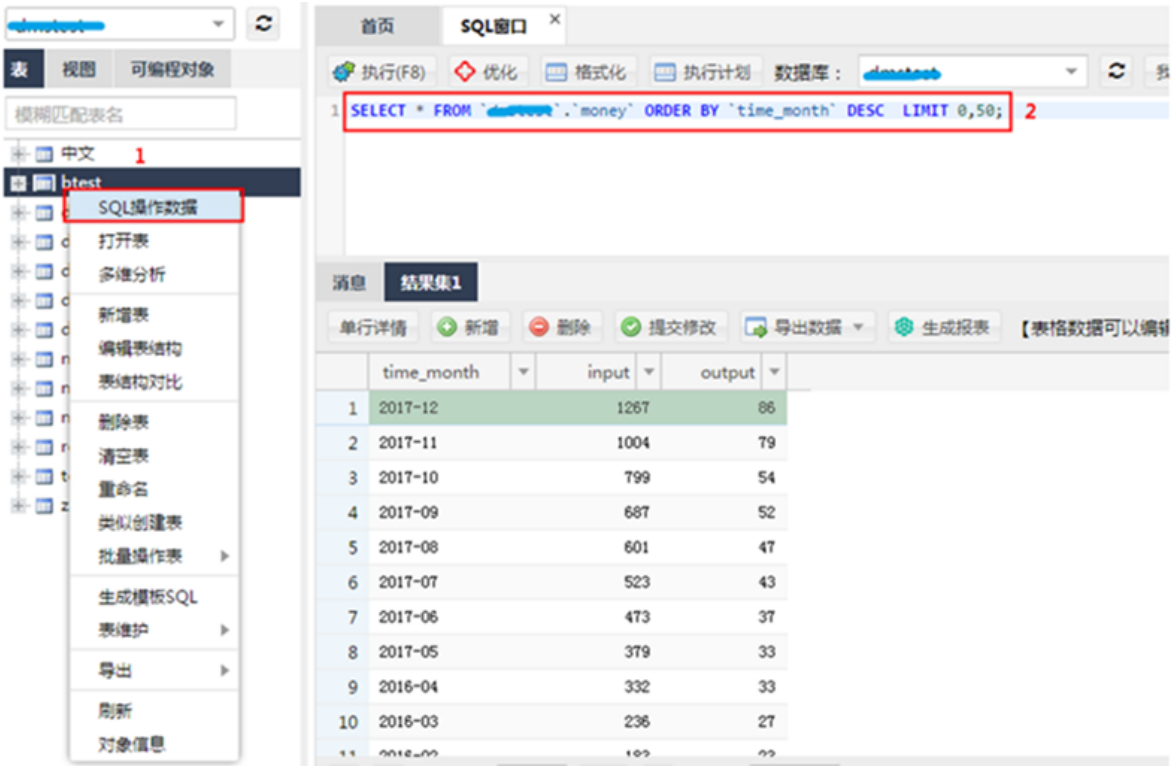
   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在DMS左边的目标树上，单击表，选择**SQL操作数据**，打开SQL窗口。

   工具将自动查询表中前50条数据的SQL，如图 *443: SQL*数据所示。

   图 443: SQL数据



19.1.2.3.2 编辑表数据

背景信息

- 这里以MySQL为例进行说明。
- 本功能适用数据不多的表。若表数据太多，需先定位至数据才能编辑。定位数据可能需花费较多时间。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

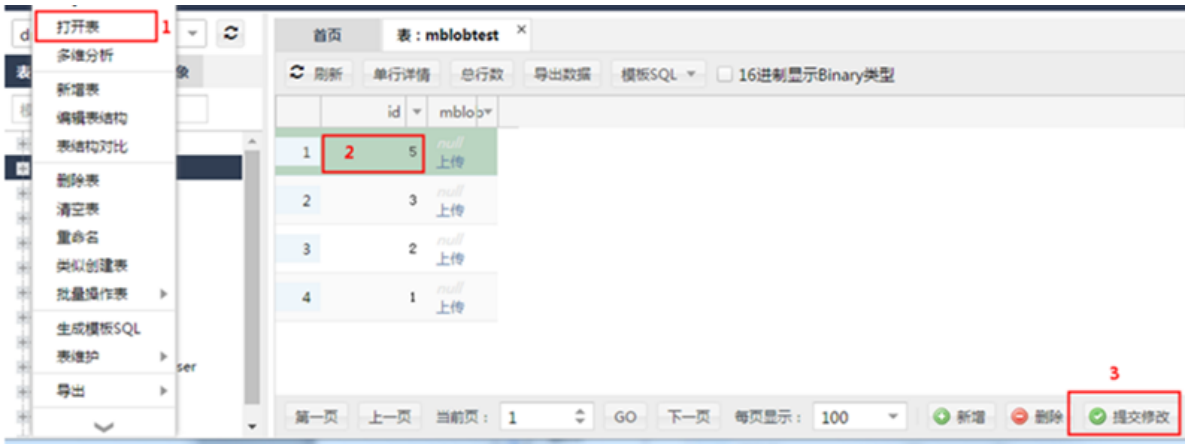
   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在DMS左右的目标树上，单击表，选择**打开表**。

   将打开一个表数据窗口，并展示该表中的数据，如图 *444:* 表数据窗口所示。

   图 444: 表数据窗口



- 编号1：在目标树的表上单击**打开表** 菜单，将打开表数据编辑界面。
- 编号2：您可以修改表中字段的数值。
- 编号3：修改完成后，单击**提交修改**按钮来提交要修改的数据。

19.1.3 数据库开发

19.1.3.1 管理表

19.1.3.1.1 新增表

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 通过如下三种方式新建表。

   - 在顶部菜单栏上，选择**新建** > **表**。
   - 在左侧的**表**目录树中，单击表，选择**新增表**。
   - 在首页的**常用操作**区域中，单击**建表**。

3. 编辑列。

   进入**新建**：**表**页面，默认位于列信息Tab页。

   您可以根据需要，编辑字段的基本信息和扩展信息。

   也可以单击**列信息**来编辑表信息。

4. 单击**索引**Tab页，进入索引编辑页。

- 单击**新增**，可以增加一个索引。
- 单击**删除**，可以删除一个索引。
- 直接编辑索引行就可修改索引信息。

5. 单击**外键**Tab页，进入**编辑外键**页。
   - 单击**新增**，增加一个外键，并进入编辑状态。
   - 单击**删除**，可以删除一个外键。
   - 直接编辑索引行即可修改索引信息。编辑时需要填写外键的名称、列及引用的库、表、列信息。
6. 单击**分区**Tab页，进入分区编辑页，填写分区的SQL信息即可。
7. 单击**基本信息**Tab页，进行表的基本信息编辑。
   - 可编辑表名、存储引擎、字符集、备注等信息。
   - 用户可以单击**更多**选项来修改表参数。
8. 单击**保存**，DMS会生成创建表SQL。

   如果用户确认无误，单击**确认**，DMS会将该表增加到用户的数据库中。

19.1.3.1.2 修改表

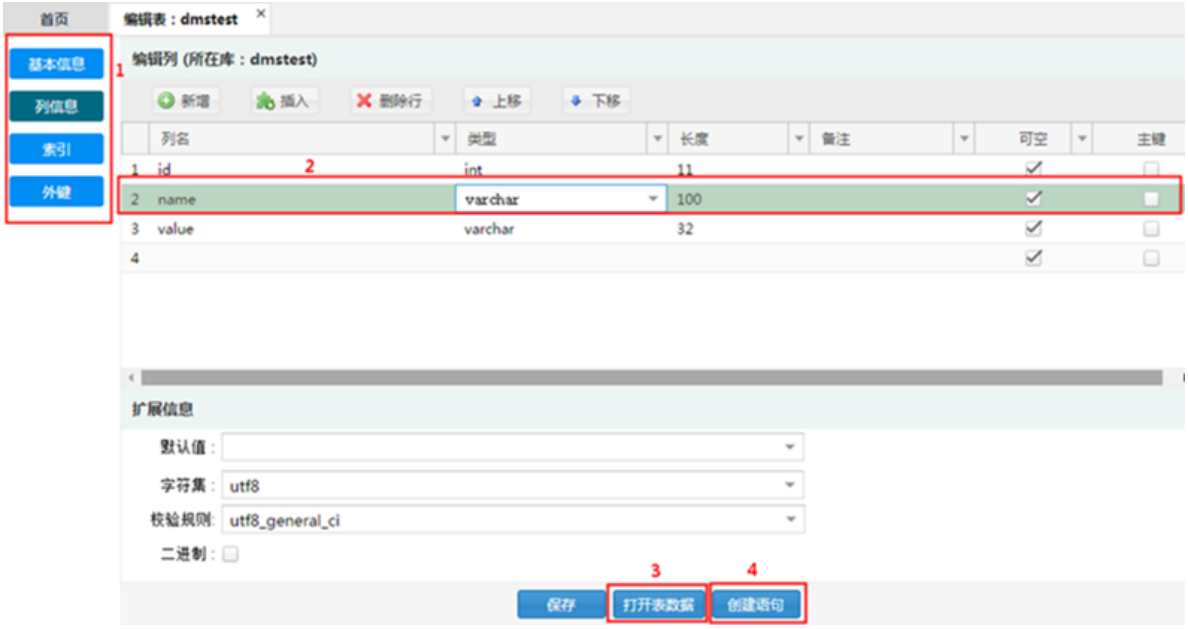
操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。
2. 在左侧的**表**目录树中，单击表，选择**编辑表结构**，对表结构进行编辑。
3. 打开的表编辑界面与新建表的界面类似，DMS会自动将表的结构加载到界面中，如图 *445:* 表编辑界面所示。

   图 445: 表编辑界面



- 编号1：单击表对象类型中选择一个类型，如**列信息**、**索引** 等。
- 编号2：单击表对象的具体操作，与新建表编辑表属性类似。
- 编号3：单击**打开表数据**，可以查看和修改表数据。
- 编号4：单击**创建语句**，可以查看表的创建语句，如图 *446: 创建SQL语句*所示。

图 446: 创建SQL语句



4. 单击**保存**，DMS会展示此次表结构变更待执行的SQL语句。

   确认无误后，单击**确认**，DMS将表结构变更保存到您的数据库中。

19.1.3.1.3 删除表

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在左侧的**表**目录树中，单击要删除的表，选择**删除表**。

   警告: 删除表是高危操作，请谨慎操作。

3. 如果确认需要删除，单击**Yes**，删除该表。

19.1.3.1.4 创建类似表

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在左侧的**表**目录树中，单击要复制的表，选择**类似创建表**。

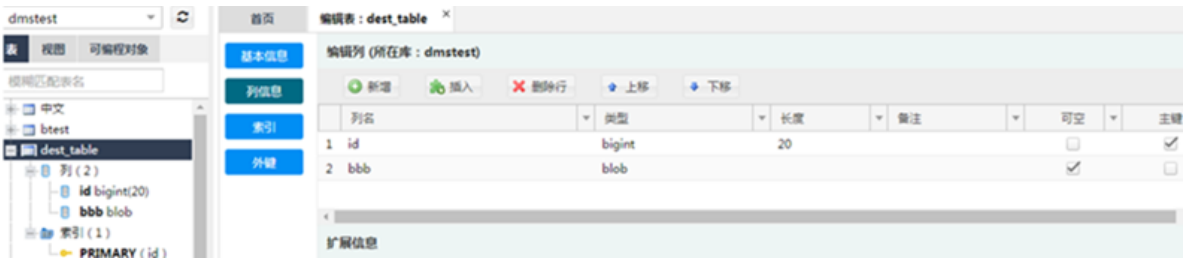
   DMS弹出类似创建表的窗口。

3. 填写目的表名，单击**确认**，DMS将创建一个类似于当前选中的表。

4. 查看目标表结构，与选中的源表相同。

   创建类似表操作完成，如图 *447:* 目标表结构所示。

   图 447: 目标表结构



19.1.3.1.5 生成模板SQL

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在左侧的**表**目录树中，单击要复制的表，选择**生成模板SQL**。

3. DMS生成该表的模板SQL，包括insert模板、update模板、select模板和create table模板，供您做SQL操作时参考使用，如图 *448: 模板SQL*所示。

图 448: 模板SQL



19.1.3.1.6 查询表信息

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在左侧的**表**目录树中，单击要复制的表，选择**对象信息**。
3. DMS获取表对象的信息，单击**基本属性**Tab页可查看表的基本信息。
4. 单击**创建语句**Tab页，可查看表的创建语句。

19.1.3.1.7 清空数据

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在左侧的**表**目录树中，单击要复制的表，选择**清空表**。

   注意: 清空表数据是一个高危操作，可能会影响您后续的数据使用，DMS会询问您是否真的要清空表。

3. 如果确认要进行清空表操作，单击**Yes**，DMS会执行表数据的清空。
4. 打开表数据查看，验证表数据是否已经被清空。

19.1.3.1.8 批量操作表

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 批量删除表。
   a) 在左侧的**表**目录树中，单击表，选择**批量操作表** > **批量删除表**。

      系统显示批量删除表窗口。

   b) 选择多个待删除的表。
   c) 单击**确定**。

      DMS会让您确认是否进行批量操作。

   d) 单击**Yes**。

      DMS 执行批量删除操作。

3. 批量操作表。

   支持批量清空数据、批量删除、批量表维护和批量修改表前缀。

   a) 在左侧目标树上，单击表，选择**批量操作表** > **更多批量操作**。

      系统显示更多批量操作窗口。

   b) 选择多个待操作的表，单击**清空数据**、**删除**、**表维护**或**表名前缀**。
   c) 单击**确定**。

      DMS会让您确认是否进行批量操作。

   d) 单击**Yes**。

DMS将执行批量操作。

19.1.3.1.9 表维护

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在左侧的**表**目录树中，单击要维护的表，选择**表维护** > **优化表**。

3. 单击**Yes**。

   如果您确认需要优化一个表，单击**Yes**后，DMS会进行表优化操作。

   表优化包括在数据库内部重新利用表中的空间、整理文件碎片等。

   说明: 检查表、修复表、分析表的操作方式与优化表相同。

19.1.3.2 管理索引

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在左侧的**表**目录树中，展开要修改的表，单击**索引** > **添加索引** 。

   系统显示**添加索引**页面，如图 *449:* 添加索引所示。

   图 449: 添加索引



3. 设置索引参数。

   - 编号1：填写索引名称，选择索引类型。
   - 编号2： 单击+或 - 为索引添加或者删除字段。
   - 编号3：编辑索引中的字段，您可以自行填写或者从下拉列表中获取。对于可变长度的数据类型，如varchar，还可以为字段指定前缀长度，以节省索引空间。

4. 单击**保存**。

   DMS会生成添加索引的SQL语句。您需要确认变更是否符合预期，如图 *450: SQL语句*所示。

   图 450: SQL语句



5. 单击**执行**。
6. 执行完添加索引后，可以再次查看表中的索引，确认添加的索引是否生效。

   添加索引后，根据需要，您可以修改或删除索引。

   - 单击表目录树上的表的索引，选择菜单上的 **修改索引** ，进入**修改索引**窗口。

     修改索引与添加索引流程类似，但是执行的SQL是先删除旧索引再增加新索引。
   - 单击表目录树上的表索引，单击菜单上的**删除** 索引，进入**删除索引**确认窗口，删除索引。

19.1.3.3 管理外键

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。
2. 在左侧的**表**目录树中，单击要修改的表，选择**编辑表结构**。
3. 在打开的**编辑表**界面上，单击**外键**Tab页，编辑外键。
4. 填写外键信息、外键字段、引用的库表字段信息。
5. 单击**保存**。

19.1.3.4 创建分区

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

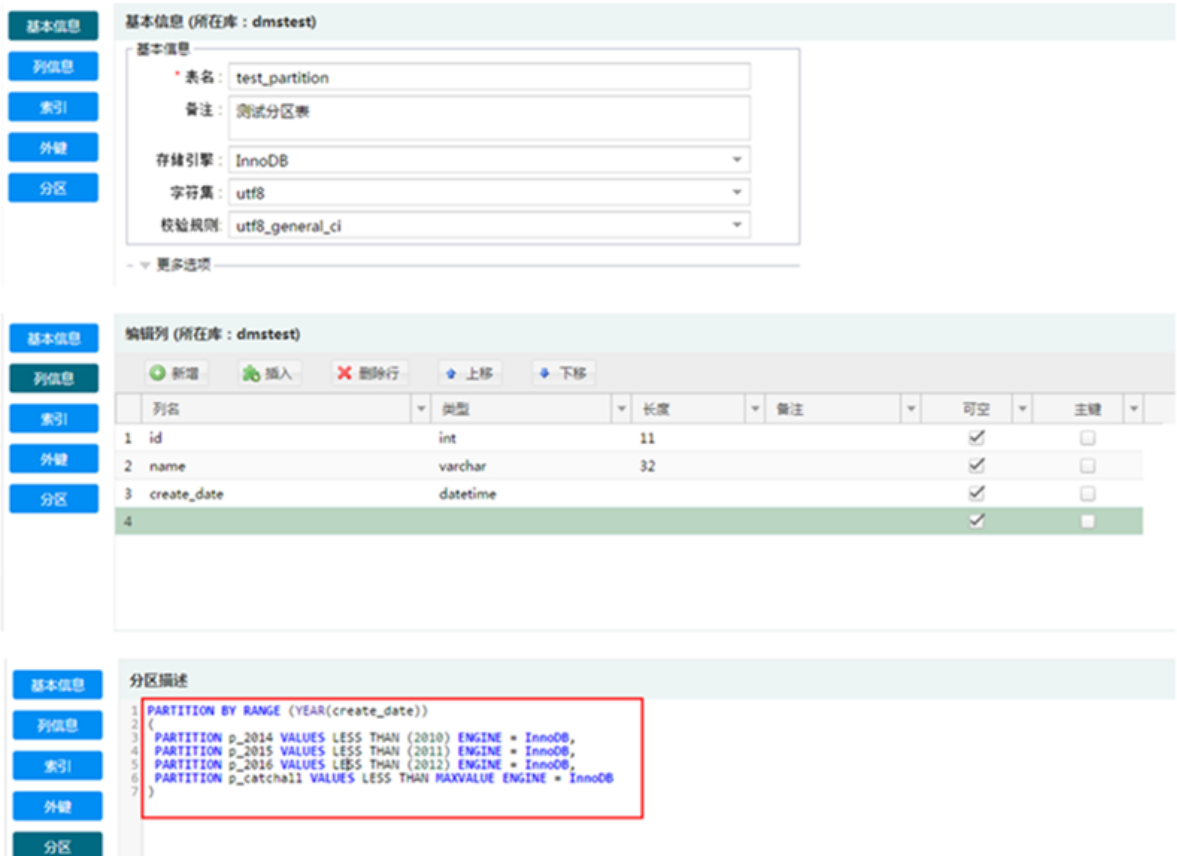
   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在左侧的**表**目录树中，单击表，选择**新增表**。

   系统显示**新建**：**表**页面，如图 *451:* 创建分区所示。

   图 451: 创建分区



3. 分别填写表基本信息、表字段信息、表分区信息。
4. 单击**保存**，保存创建的表结构。

   DMS会弹出窗口以确认创建表的SQL语句。

5. 单击**确定**，DMS将创建分表区。其中分区的字段和分区逻辑是之前填写的分区SQL。
6. 执行后查看表结构，确认分区表是否已成功创建。

19.1.3.5 创建存储过程

背景信息

这里以MySQL数据库为例进行说明。

存储过程、函数、触发器、事件在DMS中统称为可编程对象。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 单击左侧的**可编程对象**目录树，选择**存储过程** > **新建**（**存储过程**）。

   系统显示存储过程创建页面，如图 *452:* 创建存储过程所示。

   图 452: 创建存储过程



3. 填写存储过程名称说明。
4. 单击**确定**。
5. DMS提供存储过程模板，您只需编辑存储过程主体部分。
6. 单击**保存**，将存储过程保存到数据库中。

   如果存在语句错误，DMS会提示出错原因。

7. 单击**执行**，可以执行该存储过程。

   系统显示设置存储过程的入口参数值页面，如图 *453:* 设置存储过程的入口参数值所示。

   图 453: 设置存储过程的入口参数值



   填写入口参数，如本例中**cnt**参数的值设置为**80**，目标是将**Value=80**的记录查询出来。

8. 单击**开始执行**，DMS将会执行该存储过程。

   如果存储过程中有输出参数或者中间结果集，则DMS会展示这些数据。

   - **消息**Tab页将展示存储过程执行的消息。如输出变量，中间结果集等等。
   - **中间结果集1**Tab页将展示存储过程执行的过程中输出的结果集。如果有多个结果集，DMS会产生多个中间结果集Tab页， 如中间结果集1、中间结果集2、中间结果集3等等。

9. 选择**中间结果集1**页签。

   系统显示value值等于80的记录。

10. 在创建存储过程时，您可以指定创建选项。单击**选项设置** ，可以为创建存储过程指定不同的选项。

11. DMS的存储过程创建工作就完成后，您可以在可编程对象中查看到该存储过程。

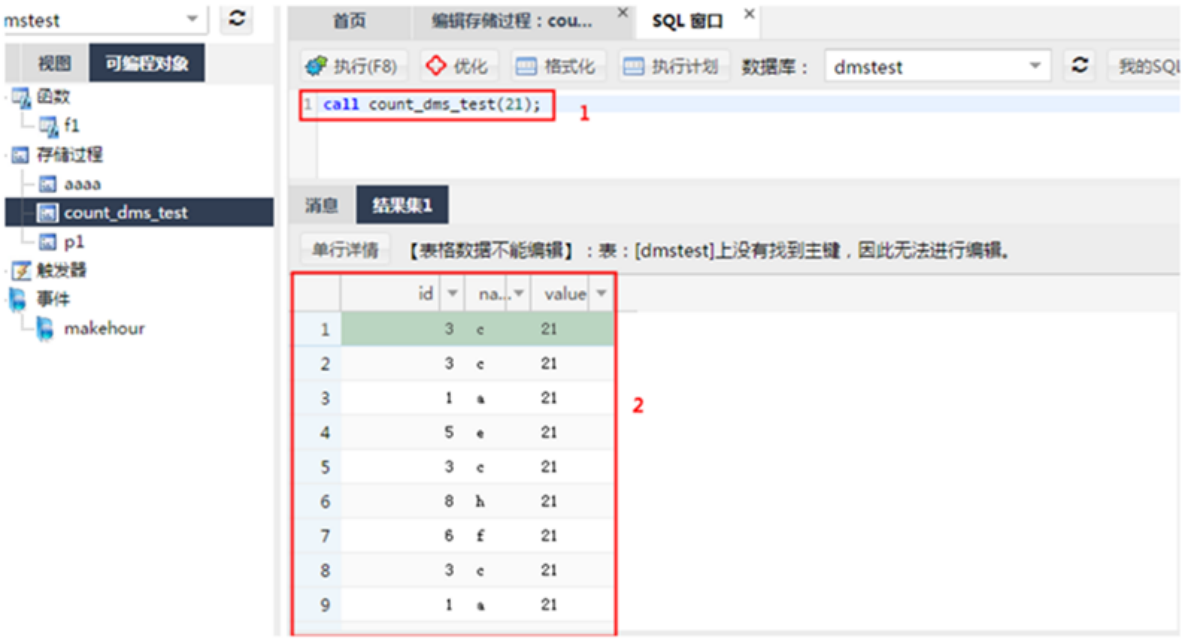
    您还可以通过菜单进行存储过程的其他操作。

    - 新建
    - 编辑
    - 删除

- 执行

12.您可以通过SQL窗口来执行存储过程，如图 *454:* 存储过程所示。

图 454: 存储过程



- 编号1：使用call存储过程名（参数）来调用存储过程。
- 编号2：对于有结果集的存储过程，SQL窗口会展示结果集。

19.1.3.6 创建函数

背景信息

函数、存储过程、触发器、事件在DMS中统称为可编程对象。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在左侧的**可编程对象**目录树中，选择**函数** > **新建**（**函数**）。

   系统显示**请填写新增函数的基本信息**页面，如图 *455:* 新增函数所示。

图 455: 新增函数



3. 设置新增函数的基本信息。
4. 单击**确定**。

   进入函数编辑界面，DMS将生成函数创建模板，您只需填写函数体部分。
5. 填写函数体部分。
6. 单击**保存**，DMS将检查函数定义是否正确，不正确的话会给出提示。

   DMS会将函数定义执行到您的数据库中，并提示保存成功。
7. 单击**执行**，进行该函数的执行操作。
8. 输入参数，如**wednesday**，单击**开始执行**后，DMS执行函数。
9. 单击**选项设置** ，可以对函数进行设置，类似于存储过程的选项设置。

   您也可以通过SQL窗口来执行函数。

19.1.3.7 创建视图

操作步骤

创建视图

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。
2. 单击左侧目录树的**视图**，可查看当前库中的视图。

3. 右键单击空白处，选择**新增视图**。

   系统显示**新建**：**视图**页面，如图 *456:* 新建视图所示。

   图 456: 新建视图



4. 设置视图的基本信息。

   本例中我们将dmstest表中value值为偶数的记录过滤出来，并输出id和name字段。

5. 单击**保存修改**，DMS会按您填写的内容，生成创建视图的SQL语句。

6. 您确认准确无误后，单击**确定**，DMS会将您定义的视图保存到数据库中。

7. 保存后，可以在左侧的目录树的**视图**页签中看到该视图，并且可以查看刚刚保存的视图的定义。

查看视图

8. 在视图上右击**查看视图**，可以通过刚刚创建的视图来查询数据。

9. 通过视图上的其他菜单，还可以进行视图的其他操作。

   包括以下选项：

   - 查看数据
   - 新建视图
   - 编辑视图
   - 删除视图
   - 刷新视图列表

19.1.3.8 创建触发器

背景信息

触发器、函数、存储过程、事件在DMS中统称为可编程对象。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 单击左侧的**可编程对象**目录树，选择**触发器** > **新建**（**触发器**）。

   系统显示**新建**：**触发器**页面，如图 *457:* 新建触发器所示。

   图 457: 新建触发器

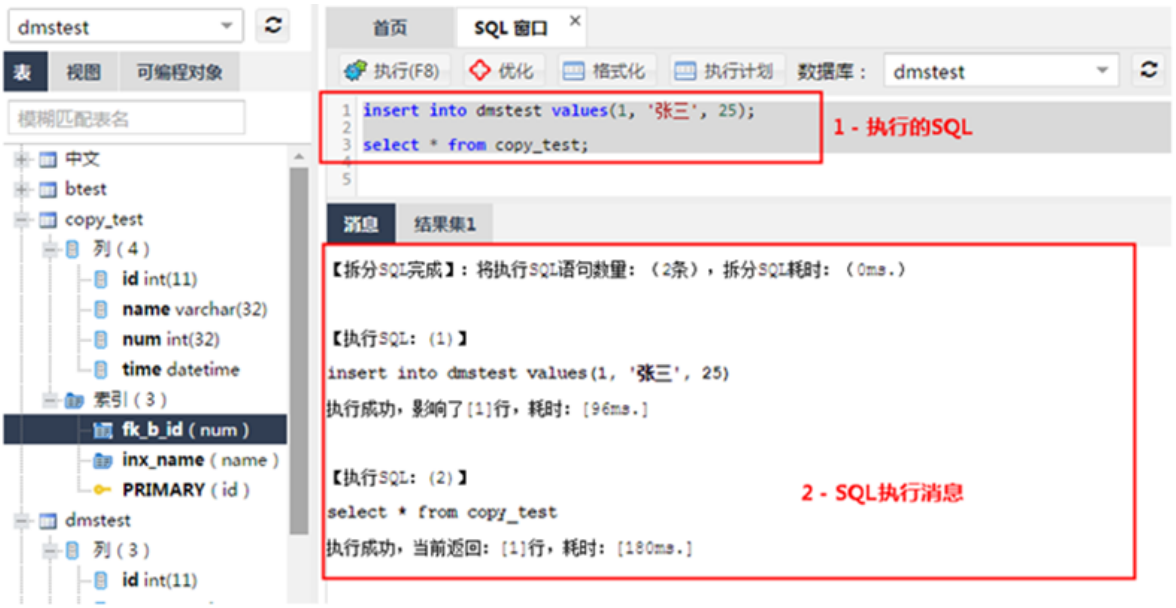


   - 编号1：触发表。
   - 编号2：进行触发器设置。
     - 填写触发器名称。
     - 选择触发表，例子中选择编号1中的dmstest表。
     - 选择触发时间，例子中选择在事件发生后。
     - 选择触发事件，例子中选择插入事件。
   - 编号3：设置触发器语句。
     - 设置在上述指定事件发生后，触发器将要进行的操作。
     - 本例中，当有数据被插入到dmstest表中时，通过触发器将数据自动插入copy_test表中，并且记录下数据被插入的时间（copy_test.time字段）。

3. 当设置好触发器选项后，单击**保存**，DMS会按您的配置生成触发器来创建SQL。您需要确认创建语句是否符合预期。

4. 单击**确定**，DMS会将触发器保存到您的数据库中。DMS会提示您触发器保存成功。在**可编程对象 > 触发器**列表中，可以查看到刚刚保存的触发器。
5. 您可以向触发表**dmstest**中插入数据，观察数据是否被记录到表**copy_test**中，如图 *458:* 触发表中添加数据所示。

图 458: 触发表中添加数据

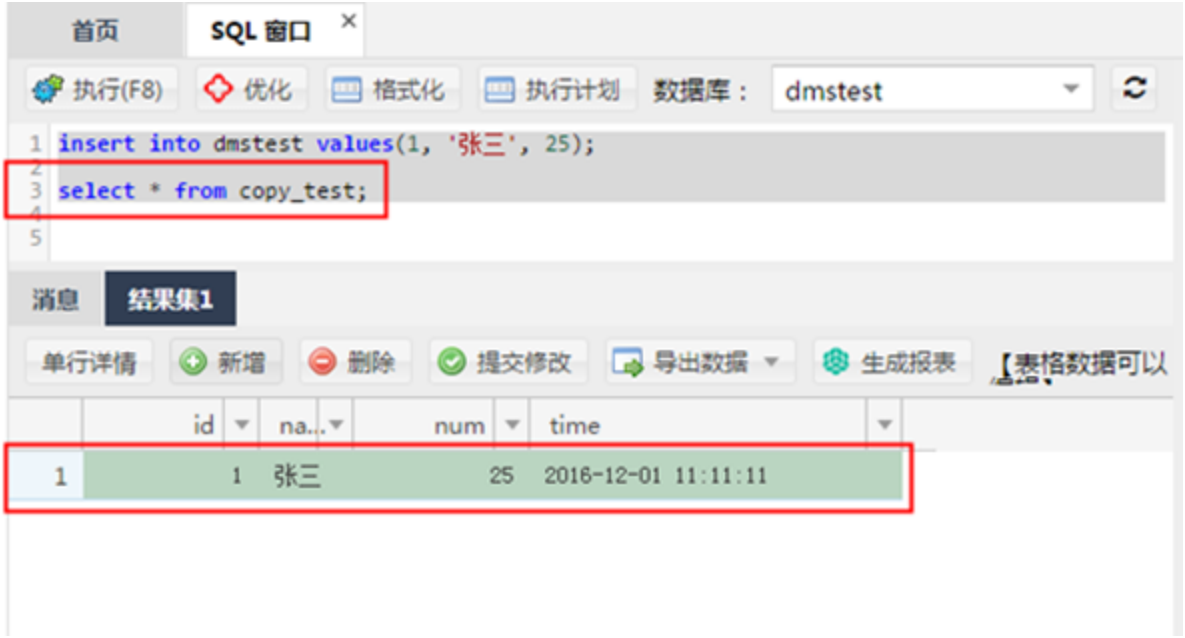


编号1：我们插入数据到dmstest表，并从copy_test表中查询数据。

编号2：SQL窗口输出了执行SQL的消息。告诉我们插入了一行到了表dmstest，并且从copy_test表中查询到一条数据。

6. 查看SQL窗口执行的结果集，验证插入操作是否正确地被触发器处理，如图 *459:* 触发验证所示。

图 459: 触发验证



由图可知，数据已正确地被触发器处理。

7. 选择**可编辑对象** > **触发器**，还可对触发器进行其他操作：
   - 新建（触发器）
   - 编辑（触发器）
   - 删除（触发器）

19.1.3.9 创建事件

前提条件

您登录的数据库必须开启事件支持。

- 通过执行SELECT @@event_scheduler;语句来查看数据库是否支持事件，若结果返回“ON”，说明数据库开启了事件支持。
- 若结果返回**OFF**，说明数据库未开启事件支持。需要您修改配置文件或者执行SET GLOBAL event_scheduler = ON;语句来开启事件支持。

背景信息

事件、触发器、函数、存储过程在DMS中统称为可编程对象。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 单击左侧的**可编程对象**目录树，选择**事件** > **新建**（**事件**）。

   系统显示新建事件页面，如图 *460:* 新建事件所示。

   图 460: 新建事件



   编号1：事件设置：主要设置好事件的名称，是否循环执行，循环周期，开始时间，结束时间，状态，注释等信息。

   编号2：事件执行语句：填写当事件定时触发时需要执行的操作。

3. 设置好事件的触发规则，填写事件的执行SQL语句。
4. 单击**保存**，DMS生成待创建事件的SQL语句。
5. 确认SQL语句无误后，单击**确定**，DMS将刚刚编辑的事件在您的数据库执行。
   - 如果执行成功，DMS会提示您保存成功。
   - 左侧的**可编程对象** > **事件**中可以看到刚刚创建的事件。
6. 通过SQL窗口来验证事件是否正常执行。

   本例中，事件的执行语句是每分钟插入一条数据到**copy_test**表中。通过查询该表，查看数据是否正常插入即可。

7. DMS通过事件上的菜单，可以进行事件的其他操作。

   包含以下选项：

   - 新建（事件）
   - 编辑（事件）
   - 删除（事件）

19.1.4 数据管理

19.1.4.1 导入数据

背景信息

这里以MySQL数据库为例进行说明。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在顶部菜单栏上，选择**数据处理** > **导入**。

   系统显示**导入**页面，如图 *461:* 导入所示。

   图 461: 导入



   数据导入窗口包括数据导入操作工具栏和导入操作历史。

   如果已经进行过数据导入操作，**导入历史列表**中会展示已有的操作历史。

3. 单击**新增任务**。

   系统显示导入任务页面，如图 *462:* 导入任务所示。

   图 462: 导入任务



- 选择文件类型，目前支持SQL和CSV两种文件类型。
- 如果数据文件使用了特定的字符集，可以手动指定文件字符集。默认情况下，DMS能探测文件字符集。
- 执行某条SQL失败时，DMS 会中断导入。您可以选择**忽略报错**，但是可能出现错误将影响后续操作。
- 您可以填写任务描述，简单介绍导入内容和导入原因等，方便后续查看。

4. 单击**开始**，导入任务开始执行。

   如果导入的数据中存在错误，DMS会中断导入，并提示错误信息。出错后，您可以修改数据文件，重新进行导入。

   如果导入的数据和SQL均正确，DMS会展示导入进度，已导入的数据量和导入耗时等信息。

   成功后，您可以在**导入历史列表**中查看到刚刚创建的导入任务。

   单击**任务编号**，可以打开任务的执行详情。

19.1.4.2 导出数据

19.1.4.2.1 导出数据库

背景信息

这里以MySQL数据库为例进行说明。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在顶部菜单栏上，选择**数据处理** > **导出** ，进入**导出**页面。
3. 在**导出**页面中，单击**新增任务** > **导出数据库**。
4. 在**新增导出任务**页面，选择数据库、文件类型（SQL/CSV）、导出内容（结构和数据/数据/结构），并在右侧列表中勾选需要导出的表和附加内容。
5. 单击**确定**，开始执行导出任务。

   DMS每隔2秒会更新导出进度，方便您随时掌握导出进度。

   您可以关闭该导出窗口，下次通过数据导出历史列表来查看和下载。

   导出完成后，DMS会将导出文件自动下载到本地，您可以单击**下载文件** ，下载本次导出文件。

   通过数据导出历史，您还可以查看之前提交的数据导出任务。单击具体任务名称条目可以查看任务详情，并进行导出数据的下载。

19.1.4.2.2 导出SQL结果集

背景信息

这里以MySQL数据库为例进行说明。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在顶部菜单上，选择**数据处理** > **导出**，进入**导出**页面。

3. 在**导出**页面中，选择**新增任务** > **SQL结果集导出**。
4. 在**新增SQL导出任务**页面，根据需要进行配置。

   选择文件类型（CSV/SQL_Insert）、结果集行数限制和数据库，并输入执行SQL。
5. 单击**确定**，DMS在后台开始执行SQL结果集导出任务。

   执行成功后，导出文件会自动下载到本地，您也可以单击**下载文件**，下载本次导出文件。

   导出完成后，DMS会进行导出结果汇总，并自动下载已经的SQL结果集文件。

   在导出任务历史列表中，查看之前提交的SQL结果集导出任务，并进行SQL结果集文件的下载。

19.1.5 性能管理

19.1.5.1 管理锁等待

19.1.5.1.1 查看锁等待

背景信息

这里以MySQL数据库为例进行说明。

操作步骤

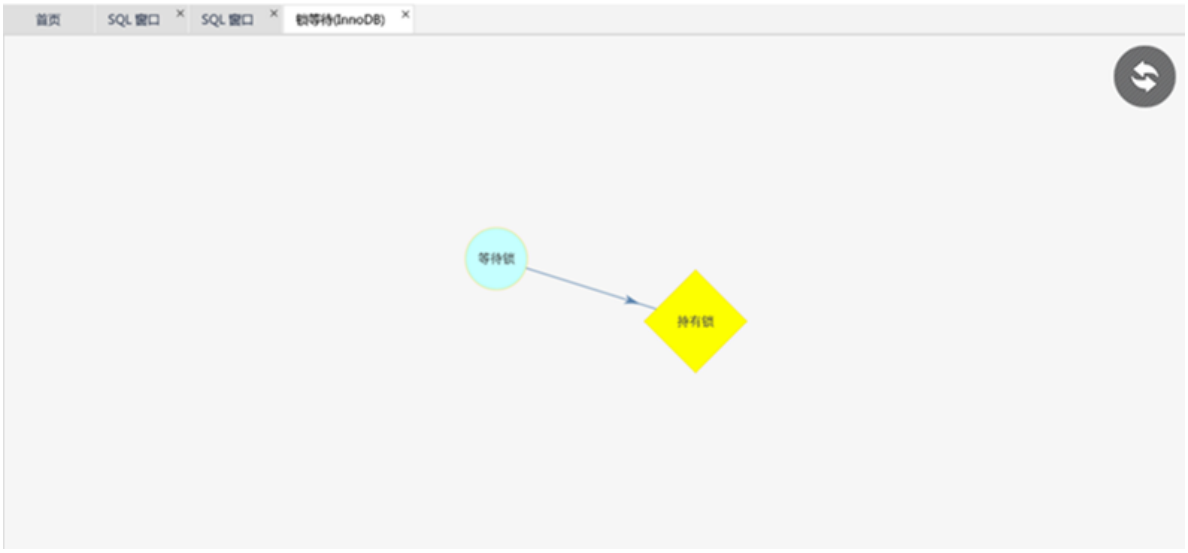
1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。
2. 在顶部栏菜单栏上，选择**性能** > **InnoDB锁等待**。

   系统显示**锁等待(InnoDB)**页面，如图 *463: InnoDB*锁等待所示。

   图 463: InnoDB锁等待



当前实例如有事务正在等待锁，将展示持有锁和等待锁。

3. 将鼠标箭头移至**持有锁**或**等待锁**图标上，页面将展示持有锁或等待锁清单及相应的会话ID。
4. 单击 ，可重新加载数据。

19.1.5.1.2 释放锁等待

背景信息

这里以MySQL数据库为例进行说明。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

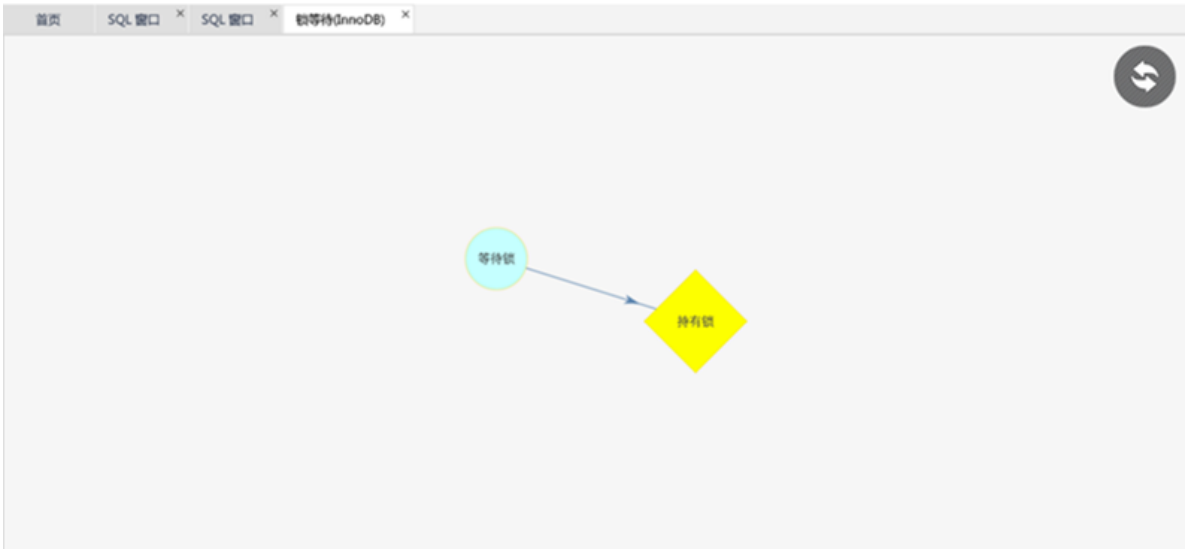
   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在顶部菜单栏上，选择**性能** > **InnoDB锁等待**。

   系统显示**锁等待(InnoDB)**页面，如图 *464: InnoDB*锁等待所示。

   图 464: InnoDB锁等待



当前实例如有事务正在等待锁，将展示持有锁和等待锁。

3. 将鼠标箭头移至**持有锁**或**等待锁**图标上，页面将展示持有锁或等待锁清单及相应会话ID。
4. 单击**持有锁**或**等待锁**图标。

   界面将弹出**确认删除会话**提示框。

   没有事务等待锁时，页面显示如图 *465:* 锁等待所示。

图 465: 锁等待



5. 单击**Yes**，即可释放当前会话。

19.1.5.2 管理会话

19.1.5.2.1 查看会话

背景信息

这里以MySQL数据库为例进行说明。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

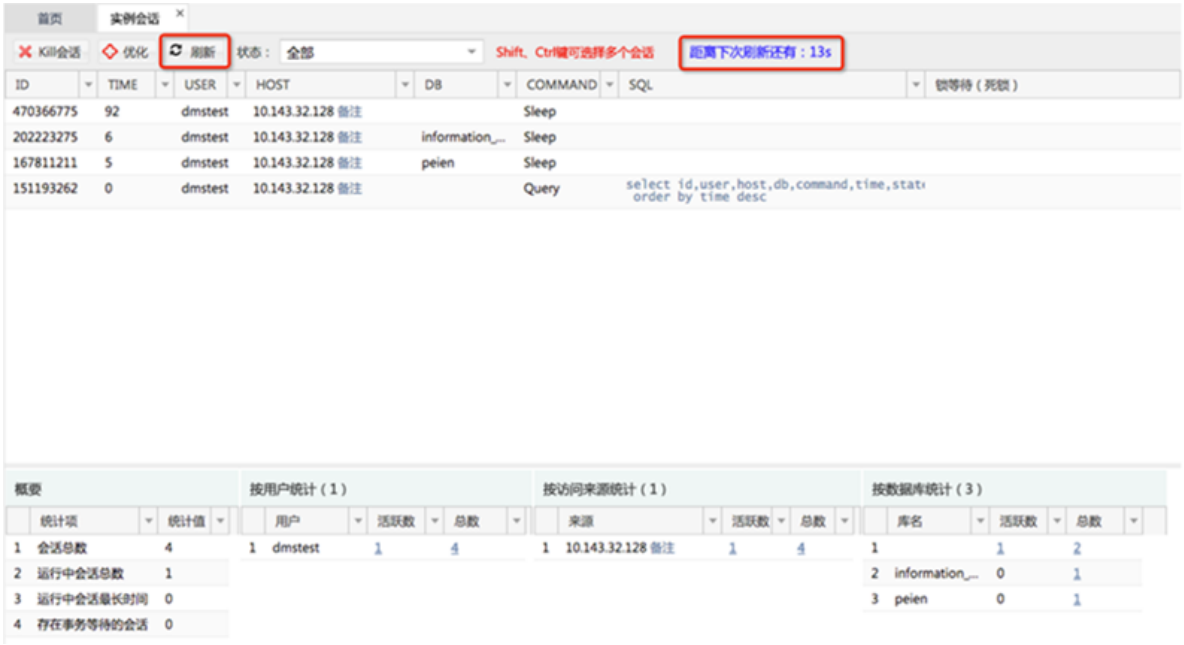
   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在顶部菜单栏上，选择**性能** > **实例会话**。

   系统显示**实例会话**页面，如图 *466:* 实例会话所示。

   图 466: 实例会话



   会话列表默认每30s刷新一次，您也可以直接单击**刷新**，进行手动刷新。

3. 单击具体的SQL文案，可查看当前会话所执行的SQL详情。

   实例列表下方提供概要、用户统计、访问来源统计、数据库统计，从不同维度展示实例会话的内容。

19.1.5.2.2 结束会话

背景信息

这里以MySQL数据库为例进行说明。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在顶部菜单栏上，选择**性能** > **实例会话**。
3. 选中单条会话记录，或通过Shift/Ctrl快捷键选中多条会话记录。
4. 单击**Kill会话**。

   界面将弹出相应会话记录的Kill会话确认框。

5. 单击**确定**。

   确认Kill会话，结束所选会话。

   您可以通过实例列表下方提供的概要、用户统计、访问来源统计及数据库统计，结束会话。

19.1.5.2.3 优化会话

背景信息

这里以MySQL数据库为例进行说明。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在顶部菜单栏上，选择**性能** > **实时会话**。
3. 选中单条会话记录，单击**优化**，执行并优化当前会话SQL。

19.1.5.3 查看实时性能

背景信息

这里以MySQL数据库为例进行说明。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

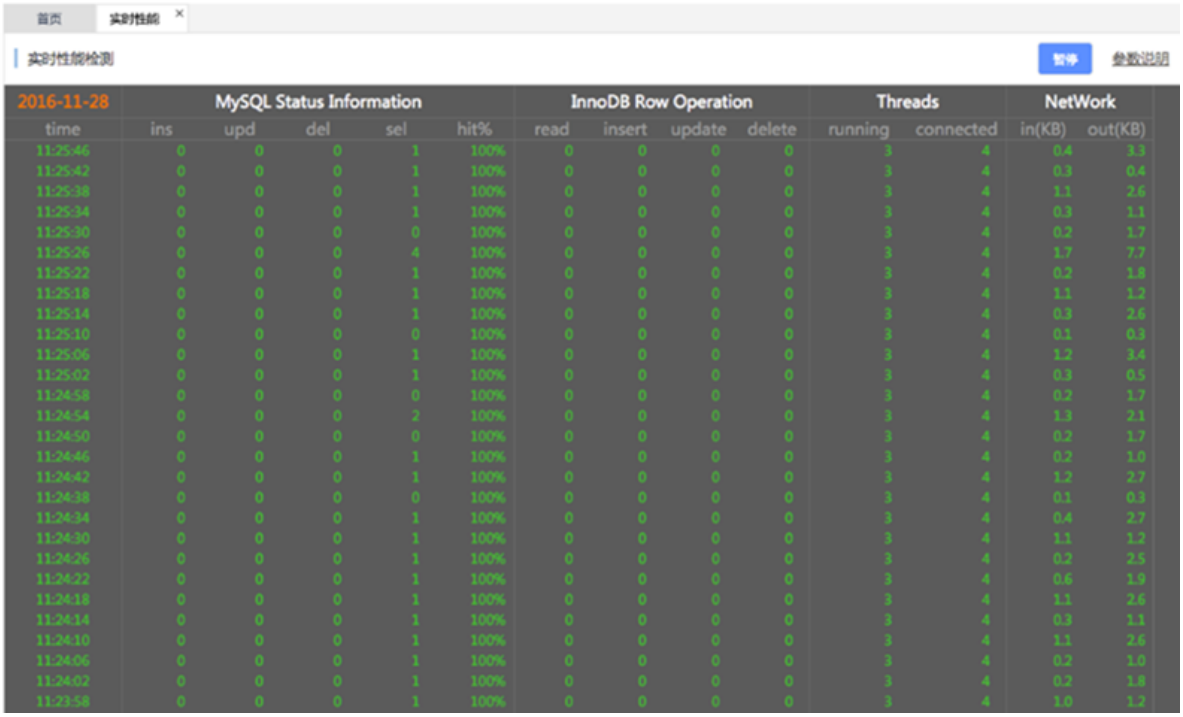
   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在顶部菜单栏上，选择**性能 > 实时性能**。

   进入实时性能页面，如图 *467:* 实时性能所示。

   图 467: 实时性能



3. 单击**暂停**或**开始**，可暂停或开始实时性能检测。

   说明: 将鼠标移至**参数说明**上，可查看实时性能参数说明。

19.1.6 扩展工具

19.1.6.1 表数据量统计

背景信息

这里以MySQL数据库为例进行说明。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

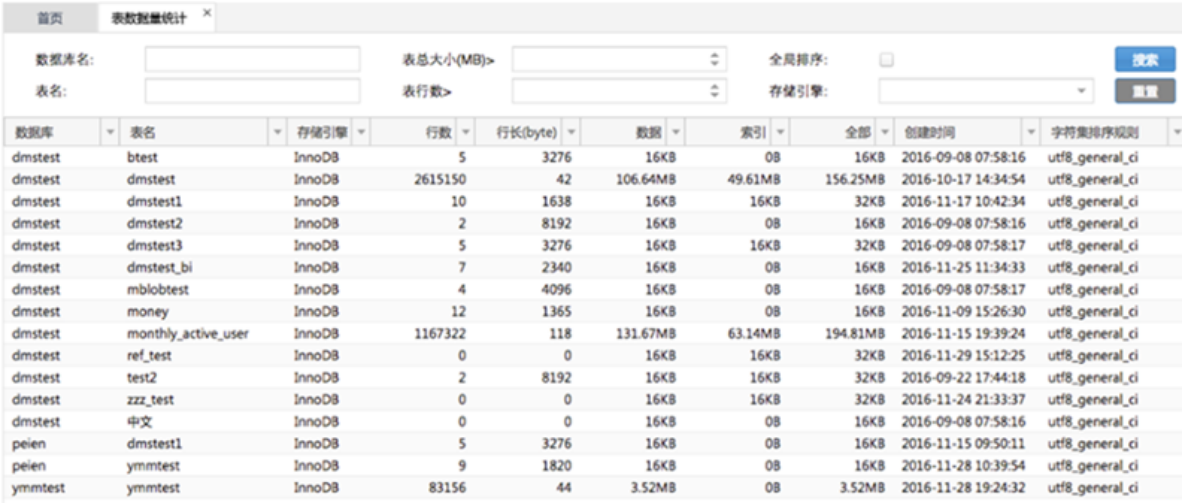
   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在顶部菜单栏上，选择**工具** > **表数据量统计**。

   系统显示**表数据量统计**页面，如图 *468:* 表数据量统计所示。

   图 468: 表数据量统计



数据库	表名	存储引擎	行数	行长(byte)	数据	索引	全部	创建时间	字符集排序规则
dmstest	btest	InnoDB	5	3276	16KB	0B	16KB	2016-09-08 07:58:16	utf8_general_ci
dmstest	dmstest	InnoDB	2615150	42	106.64MB	49.61MB	156.25MB	2016-10-17 14:34:54	utf8_general_ci
dmstest	dmstest1	InnoDB	10	1638	16KB	16KB	32KB	2016-11-17 10:42:34	utf8_general_ci
dmstest	dmstest2	InnoDB	2	8192	16KB	0B	16KB	2016-09-08 07:58:16	utf8_general_ci
dmstest	dmstest3	InnoDB	5	3276	16KB	16KB	32KB	2016-09-08 07:58:17	utf8_general_ci
dmstest	dmstest_bi	InnoDB	7	2340	16KB	0B	16KB	2016-11-25 11:34:33	utf8_general_ci
dmstest	mblobtest	InnoDB	4	4096	16KB	0B	16KB	2016-09-08 07:58:17	utf8_general_ci
dmstest	money	InnoDB	12	1365	16KB	0B	16KB	2016-11-09 15:26:30	utf8_general_ci
dmstest	monthly_active_user	InnoDB	1167322	118	131.67MB	63.14MB	194.81MB	2016-11-15 19:39:24	utf8_general_ci
dmstest	ref_test	InnoDB	0	0	16KB	16KB	32KB	2016-11-29 15:12:25	utf8_general_ci
dmstest	test2	InnoDB	2	8192	16KB	16KB	32KB	2016-09-22 17:44:18	utf8_general_ci
dmstest	zzz_test	InnoDB	0	0	16KB	16KB	32KB	2016-11-24 21:33:37	utf8_general_ci
dmstest	中文	InnoDB	0	0	16KB	0B	16KB	2016-09-08 07:58:16	utf8_general_ci
peien	dmstest1	InnoDB	5	3276	16KB	0B	16KB	2016-11-15 09:50:11	utf8_general_ci
peien	ymmtest	InnoDB	9	1820	16KB	0B	16KB	2016-11-28 10:39:54	utf8_general_ci
ymmtest	ymmtest	InnoDB	83156	44	3.52MB	0B	3.52MB	2016-11-28 19:24:32	utf8_general_ci

3. 在当前页面查看当前实例下所有用户表信息，包含数据库、表名、存储引擎、行数、行长（byte）、数据、索引、全部、创建时间及字符集排序规则。

   您可以在表数据量统计中，按数据库名、表名、表总大小（MB）、表行数、全局排序、存储引擎多条件筛选，同时还可完成翻页、刷新和重置操作。

19.1.6.2 E-R 图

背景信息

这里以MySQL数据库为例进行说明。

操作步骤

1. 在DMS登录页面，输入数据库登录信息，单击**登录**。

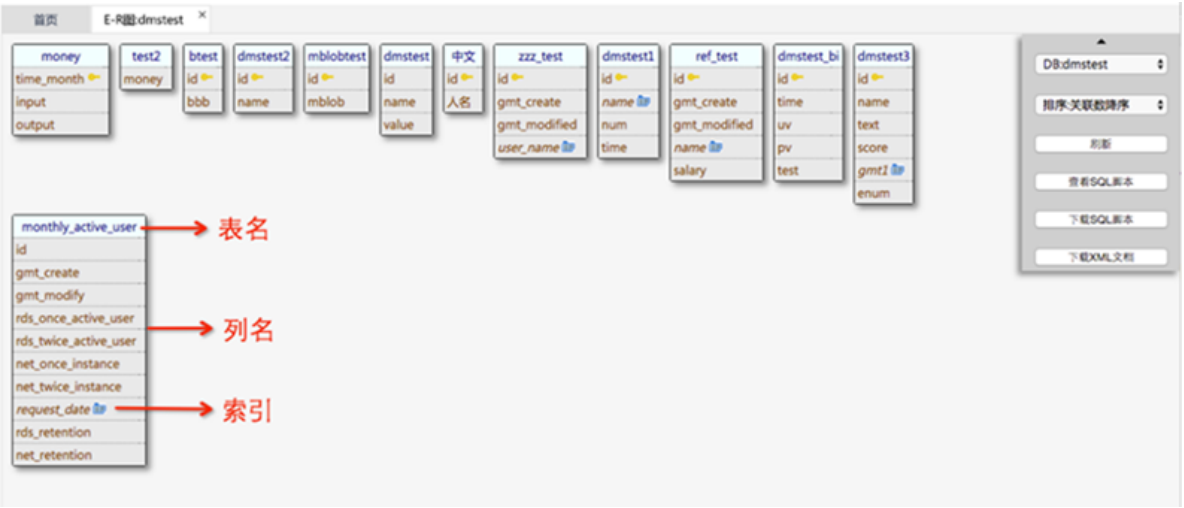
   登录数据库。

   通过DMS登录RDS实例的操作请参见通过*DMS*登录实例。

2. 在顶部菜单栏上，选择**工具** > **E-R图**。

系统显示E-R图页面，如图 *469: E-R*图所示。

图 469: E-R图



E-R图展示当前数据库下所有表的实体-联系图，提供了表示表名、列名、索引和联系的方法。

3. 进行如下操作：

- 选择**DB:mysql**下拉菜单，可切换数据库。
- 选择**排序**：**排序选项**下拉菜单，可按表名升序、关联数降序、字段数升序、字段数降序排列表。
- 单击**刷新**，可刷新当前E-R图。
- 单击**查看SQL脚本**，可查看当前数据库下所有表的建表SQL。
- 单击**下载SQL脚本**，可下载当前数据库下所有表的建表SQL。
- 单击**下载XML文档**，可将当前数据库下建表SQL转成XML文件。
- 双击表的列名，可展示当前列的定义信息。
- 双击表名，会跳转到编辑当前表页面。

20 负载均衡SLB

20.1 产品概述

负载均衡（Server Load Balancer）是将访问流量根据调度算法和转发策略分发到后端多台云服务器（Elastic Compute Service，简称ECS）的流量分发控制服务。

负载均衡可以通过流量分发扩展应用系统对外的服务能力。通过检查云服务器池中ECS实例的健康状态，自动隔离异常状态的ECS实例，从而解决了单台ECS实例的单点问题，提高了应用的整体服务能力。

负载均衡主要有如下几个功能：

- 负载均衡服务提供四层（UDP和TCP协议）和七层（HTTP和HTTPS协议）监听。
- 负载均衡服务会检查云服务器池中ECS的健康状态，自动隔离异常状态的ECS，从而解决了单台ECS的单点问题，同时提高了应用的整体服务能力。
- 负载均衡服务是ECS面向多机方案的一个配套服务，需要同ECS结合使用。

20.2 规划和准备

在搭建负载均衡服务之前，您需要做如下准备：

- 创建ECS实例

  在使用负载均衡服务前，您需要创建好ECS实例，并搭建好需要的应用服务。ECS实例将用来接收、处理负载均衡监听转发的请求。
- 规划负载均衡实例的类型
  - 外网：如果您的负载均衡实例需要对外提供服务或允许外网访问，选择外网。
  - 内网：如果您的负载均衡实例只需要对内提供服务，选择内网。此时您还需要进一步选择实例的网络类型，经典网络还是专有网络。如果选择专有网络，确保您已经创建了专有网络和交换机。
- 规划负载均衡的监听服务

  阿里云提供四层（TCP/UDP协议）和七层（HTTP/HTTPS）负载均衡服务。您可以根据不同的业务场景，选择配置不同的监听。

由于七层监听比四层监听在底层实现上多了一个Tengine处理环节，因此，七层性能没有四层性能好。此外，客户端端口不足，后端服务器连接过多等场景，也可能导致七层服务性能不高，如果您对性能有很高的要求，建议您使用四层监听。

20.3 配置监听

负载均衡监听负责监听负载均衡实例上的请求，根据转发规则将请求流量分发至后端ECS上。每个监听对应后端的一个应用服务。

背景信息

负载均衡提供四层（TCP 协议和 UDP 协议）和七层（HTTP协议 和 HTTPS 协议）监听。每个协议的特点和使用场景如下表所示。

表 70: 监听协议

协议	特点	使用场景
TCP	• 面向连接的协议。在正式收发数据前，必须和对方建立可靠的连接 • 基于源地址进行会话保持 • 在网络层可直接看到来源地址 • 数据传输快	• 适用于注重可靠性，对数据准确性要求高，速度相对较慢的场景，如文件传输、发送或接收邮件、远程登录等和无特殊要求的Web应用
UDP	• 面向非连接的协议。在数据发送前不与对方进行三次握手，直接进行数据包发送，不提供差错恢复和数据重传 • 可靠性相对低 • 数据传输快	适用于关注实时性而相对不注重可靠性的场景，如视频聊天、金融实时行情推送
HTTP	• 应用层协议，主要解决如何包装数据 • 基于Cookie进行会话保持 • 使用X-Forward-For获取源地址	适用于需要对数据内容进行识别的应用，如Web应用、小的手机游戏等
HTTPS	• 加密传输数据，可以阻止未经授权的访问 • 统一的证书管理服务，您可以将证书上传到负载均衡，解密操作直接在负载均衡上完成	适用于需要加密传输的应用

20.3.1 配置四层监听

创建完负载均衡实例后，您需要添加至少一个监听。通过添加监听配置转发端口、转发规则和健康检查。

背景信息

阿里云提供四层TCP协议和UDP协议的监听服务。负载均衡四层监听将请求直接转发到后端ECS实例，而且不修改表头。关于UDP和TCP的特点和使用场景，参考配置监听。

操作步骤

1. 在负载均衡实例列表中，单击目标实例的ID链接，打开负载均衡实例详情页面。
2. 单击**服务监听**页签，打开**服务监听**列表页面。
3. 单击**添加**。
4. 在**添加服务监听**对话框，指定以下监听配置，然后单击**确认**完成。

表 71: 监听配置

配置	说明
SLB协议[端口]	选择用来接收请求并向后端服务器进行请求转发的负载均衡系统前端协议和端口。 四层监听选择**TCP**或**UDP**。
后端协议[端口]	输入后端ECS开放用来接收请求的端口。
转发规则	选择一个转发规则。 • **轮询模式**：按照访问次数依次将请求分发到后端ECS上。 • **最小连接数**：将请求转发至当前连接数最小的后端ECS上。
会话保持	选择是否开启会话保持。如果开启会话保持，指定会话的超时时间。 开启会话保持功能后，负载均衡会把来自同一客户端的访问请求分发到同一台后端ECS上进行处理。 针对四层（TCP协议和UDP协议）监听，负载均衡系统提供基于IP地址的会话保持。负载均衡会将来自同一IP地址的访问请求转发到同一台后端ECS进行处理。
检查端口	健康检查服务访问后端时的探测端口。
响应超时时间	每次健康检查响应的最大超时时间。 后端ECS在相应时间内没有正确响应，则判定为健康检查失败。
健康检查间隔	进行健康检查的时间间隔。

配置	说明
	LVS集群内所有节点，都会独立、并行地遵循该属性去对后端ECS进行定期健康检查。
不健康阈值	同一LVS节点服务器针对后端同一ECS，从成功到失败的连续健康检查失败次数。
健康阈值	同一LVS节点服务器针对后端同一ECS，从失败到成功的连续健康检查成功次数。

20.3.2 配置七层监听

创建完负载均衡实例后，您需要添加至少一个监听。通过添加监听配置转发端口、转发规则和健康检查。

背景信息

阿里云提供七层（HTTP协议和HTTPS协议）的负载均衡服务，负载均衡七层监听原理上是反向代理的一种实现。客户端HTTP请求到达负载均衡监听后，负载均衡服务器会通过与后端ECS建立TCP连接，即再次通过新TCP连接HTTP协议访问后端，而不是直接转发报文到后端ECS。

关于HTTP和HTTPS的特点和使用场景，参考配置监听。

操作步骤

1. 在负载均衡实例列表中，单击目标实例的ID链接，打开负载均衡实例详情页面。
2. 单击**服务监听**页签，打开**服务监听**列表页面。
3. 单击**添加**。
4. 在**添加服务监听**对话框，指定以下监听配置，然后单击**确认**完成。

表 72: 监听配置

配置	说明
SLB协议[端口]	选择用来接收请求并向后端服务器进行请求转发的负载均衡系统前端协议和端口。 七层监听，选择**HTTP**或**HTTPS**。
后端协议[端口]	输入后端ECS开放用来接收请求的端口。
转发规则	选择一个转发规则。 • **轮询模式**：按照访问次数依次将请求分发到后端ECS上。 • **最小连接数**：将请求转发至当前连接数最小的后端ECS上。

配置	说明
双向认证	选择是否开启双向认证。开启双向认证后，除了服务器证书，您还需要上传CA证书。 默认未开启，进行单向认证。如果还未上传所需证书，单击**上传证书**。详情查看上传证书。 **说明:** 该选项只适用于HTTPS协议监听。
选择Server证书	选择服务器证书。 服务器证书用于用户浏览器检查服务器发送的证书是否是由自己信赖的中心签发的。 **说明:** 该选项只适用于HTTPS协议监听。
选择CA证书	选择CA证书。 服务器收到用户浏览器发送的客户端证书后，用CA证书验证客户端证书。如果没有通过验证，拒绝连接。
会话保持	选择是否开启会话保持。如果开启会话保持，指定会话的超时时间。 开启会话保持功能后，负载均衡会把来自同一客户端的访问请求分发到同一台后端ECS上进行处理。 针对七层（HTTP协议和HTTPS协议）监听，负载均衡系统提供基于cookie的会话保持。
Cookie处理方式	选择cookie的处理方式。 • **SLB植入Cookie**：客户端第一次访问时，负载均衡服务在返回请求中植入cookie（即在HTTP/HTTPS响应报文中插入SERVERID字串）。下次客户端携带此cookie访问，负载均衡服务会将请求定向转发给之前记录到的ECS实例上。 此种方式，您需要指定cookie的过期时间。 • **SLB重写Cookie**：负载均衡服务发现自定义了cookie，将会对原来的cookie进行重写。下次客户端携带新的cookie访问，负载均衡服务会将请求定向转发给之前记录到的ECS实例上。 此种方式，您需要指定HTTPS/HTTP响应中插入的cookie并在后端ECS上维护cookie的过期时间。
域名和检查路径	HTTP健康检查默认由负载均衡系统通过后端ECS内网IP地址向应用服务器的缺省首页发起http head请求。

配置	说明
	• 如果您用来进行健康检查的页面并不是应用服务器的缺省首页，需要指定域名和具体的检查路径。 • 如果您对http head请求限定了host字段的参数，您只需要指定检查路径，即用于健康检查页面文件的URI。
检查端口	健康检查服务访问后端时的探测端口。
响应超时时间	每次健康检查响应的最大超时时间。 后端ECS在相应时间内没有正确响应，则判定为健康检查失败。
健康检查间隔	进行健康检查的时间间隔。 LVS集群内所有节点，都会独立、并行地遵循该属性去对后端ECS进行定期健康检查。
不健康阈值	同一LVS节点服务器针对后端同一ECS，从成功到失败的连续健康检查失败次数。
健康阈值	同一LVS节点服务器针对后端同一ECS，从失败到成功的连续健康检查成功次数。

20.3.3 设置访问控制

您可以通过添加白名单，控制哪些IP可以访问负载均衡监听。

背景信息

在设置白名单时，请注意：

- 设置白名单存在一定业务风险，一旦设置白名单，就只有白名单中的IP可以访问负载均衡监听。
- 如开启访问控制而不设置白名单列表，则这个负载均衡监听就无人可以访问。
- 设置白名单的过程中可能会引起访问负载均衡监听短时中断。

操作步骤

1. 在负载均衡实例列表中，单击目标实例的ID链接，打开负载均衡实例详情页面。
2. 单击**服务监听**页签，打开服务监听列表页面。
3. 在目标监听的**操作**列下，选择**管理 > 配置访问控制**。
4. 在弹出的对话框内，开启访问控制开关，然后输入允许访问该监听的IP地址。

   IP以逗号隔开，不可重复，最多支持300条；支持输入CIDR block形式的网段，如10.23.12.0/24。
5. 单击**确认**。

20.3.4 配置转发策略

负载均衡为 HTTP 或 HTTPS 监听提供监听维度上的域名URL转发功能，您可配置域名或URL，及对应的虚拟服务器组，系统根据相应的命中转发规则进行流量转发。

背景信息

更多信息对于来自负载均衡实例前端的相关请求，转发判断规则如下：

- 如果能匹配到相应监听关联的按域名、URL 转发策略，则按转发策略，将请求转发到对应的虚拟服务器组；
- 如果未匹配，而对应监听启用并配置了虚拟服务器组，则将请求转发到对应的虚拟服务器组；
- 如果均未匹配，则转发到负载均衡实例后端所有 ECS 。端口为监听配置时指定的后端端口。

操作步骤

1. 登录负载均衡云控制台。
2. 单击目标实例的ID链接，打开负载均衡实例详情页面。
3. 单击 **服务监听** 页签，打开服务监听列表页面。
4. 选择创建好的 7 层监听操作列内的 **管理** > **配置转发规则** 。
5. 在**配置转发规则**对话框，单击**添加转发规则**。
6. 参考界面提示填写转发规则，然后单击 **保存**。

20.4 添加后端服务器

创建完负载均衡实例后，您需要为其添加ECS实例用来处理前端转发的连接请求。

操作步骤

1. 在负载均衡实例列表中，单击目标实例的ID链接，打开负载均衡实例详情页面。
2. 单击**后端服务器**页签，打开后端服务器列表页面。
3. 单击**添加后端服务器**。
4. 在弹出的对话框内，选择需要添加为后端服务器的ECS实例，并修改服务器的权重。

   默认情况下，采取平均的转发规则，每台ECS的权重均为100。
5. 单击**添加**。

后续操作

如果您需要修改或删除已添加的后端服务器，在后端服务器列表页面，进行如下操作：

- 修改后端服务器权重

找到目标ECS实例，单击右侧的**管理** > **修改**修改ECS的权重。

- 移出后端服务器

  找到目标ECS实例，单击右侧的**管理** > **移出**。建议您在移除ECS实例前，先将其权重修改为0。待无请求转发给该ECS实例时，再将其移除，避免影响业务。

20.5 管理虚拟服务器组

虚拟服务器组可满足需要在监听级别设置后端服务器和端口以及需要使用域名和URL转发的需求。

说明:

- 虚拟服务器组只能添加监听所在区域的后端服务器。
- 一个后端服务器可以属于多个虚拟服务器组。
- 一个虚拟服务器组可绑定在一个实例的多个监听上。
- 虚拟服务器组由服务器+端口组成。

操作步骤

1. 登录负载均衡云控制台。
2. 单击 **虚拟服务器组** 页签，打开虚拟服务器组列表页。
3. 单击 **创建虚拟服务器组**，弹出 **创建虚拟服务器组** 对话框。
4. 填写分组名称，在 **可选服务器列表** 区域选择相应的服务器，在已选服务器列表配置端口和权重。
5. 单击 **确认**，完成虚拟服务器组的添加。

后续操作

创建好虚拟服务器组后，您还可以在虚拟服务器组列表页，查看虚拟服务器组详情、编辑虚拟服务器组、以及删除虚拟服务器组。

20.6 管理负载均衡实例

通过负载均衡云控制台，您可以编辑、删除、启动、停止负载均衡实例。

20.6.1 启动或停止实例

您可以随时启动或停止您的负载均衡服务。

操作步骤

1. 登录负载均衡云控制台。

2. 单击目标实例的ID链接，进入负载均衡实例详情页面。
3. 还可以通过单击右侧的**管理** > **启动**或**管理** > **停止**来启动或停止实例。

20.6.2 查看实例详情

登录负载均衡云控制台后，即可查看已创建的负载均衡实例。

操作步骤

1. 登录负载均衡云控制台。
2. 在实例列表页，单击实例的ID链接查看实例详情，或单击右侧的**管理** > **查询详情** 。

20.6.3 修改实例属性

您可编辑负载均衡实例的名称和描述信息。

操作步骤

1. 登录负载均衡云控制台。
2. 在负载均衡实例列表中，单击目标实例的**操作**列下的**管理** > **编辑**。
3. 在弹出的对话框内，编辑负载均衡实例的名称以及描述信息，然后单击**确定**。

20.7 管理证书

为了满足数据传输的安全需求，负载均衡支持HTTPS监听。为了配置HTTPS监听，您需要上传所需的证书。

如果是HTTPS双向认证，需要上传CA证书和服务器证书；如果是HTTPS单向认证，仅需要上传服务器证书。证书上传到负载均衡系统后，无需再在后端ECS上部署证书。上传到证书管理系统的私钥都会加密存储。

- 服务器证书：用于用户浏览器检查服务器发送的证书是否是由自己信赖的中心签发的，服务器证书可以到阿里云云盾证书服务购买，也可以到其它服务商处购买。
- 客户端证书：用于证明客户端用户的身份，使得客户端用户在与服务器端通信时可以证明其真实身份。您可以用自签名的CA证书为客户端证书签名。
- CA证书：服务器收到用户浏览器发送的客户端证书后，用CA证书验证客户端证书。如果没有通过验证，拒绝连接。

在证书管理页面，您可以：

- 单击**上传安全证书**，上传证书。详情查看上传证书。
- 单击**批量删除**，删除所有选中的证书。

- 在目标证书的**操作**列中，单击**管理** > **编辑证书**修改证书的名称。
- 在目标证书的**操作**列中，单击**管理** > **删除证书**删除证书。

20.7.1 证书格式

上传到负载均衡的证书必须是Linux环境下PEM格式的证书。

证书格式要求

确保您上传的证书满足如下证书要求：

- 根证书机构颁发的证书

  如果是通过根证书（root CA）机构颁发的证书，您拿到的证书是唯一的一份，不需要额外的证书。配置的站点即可被浏览器等访问设备认为可信。证书规则如下：

  - 以 -----BEGIN CERTIFICATE-----, -----END CERTIFICATE----- 开头和结尾；请将这些内容一并上传。
  - 每行64个字符，最后一行长度可以不足64个字符。
  - 证书内容不要包含空格。
- 中级机构颁发的证书

  如果是通过中级CA机构颁发的证书，您拿到的证书文件包含多份证书，需要将服务器证书与中间证书合并在一起上传。拼接规则如下：

  - 服务器证书放第一位，中间证书放第二位，中间不要有空行。
  - 证书内容不要包含空格。
  - 证书之间不能有空行，并且每行64字节，详情参考*https://www.ietf.org/rfc/rfc1424.txt*。
  - 符合证书的格式要求。一般情况下，机构在颁发证书的时候会有对应说明，注意规则说明。

RSA私钥格式要求

在上传服务器证书时，您也需要上传证书的私钥，RSA私钥规则如下：

- 以-----BEGIN RSA PRIVATE KEY-----, -----END RSA PRIVATE KEY----- 开头和结尾，请将这些内容一并上传。
- 字串之间不能有空行，每行64字符，最后一行长度可以不足64字符。

> **说明:**
>
> 如果您的私钥是加密的，比如私钥的开头和结尾是 -----BEGIN PRIVATE KEY-----, -----END PRIVATE KEY----- 或 -----BEGIN ENCRYPTED

PRIVATE KEY-----, -----END ENCRYPTED PRIVATE KEY----- ，或者私钥中包含 Proc-Type: 4,ENCRYPTED ，需要先运行以下命令进行转换，然后将*new_server_key.pem*的内容与服务器证书一起上传。

```
openssl rsa -in old_server_key.pem -out new_server_key.pem
```

20.7.2 生成客户端证书

客户端证书用于证明客户端用户的身份，使得客户端用户在与服务器端通信时可以证明其真实身份。

前提条件

本操作使用CA证书为客户端证书签名，确保您已经生成*CA*证书。

操作步骤

1. 运行以下命令在*ca*目录创建存放key的目录*users*。

   ```
   $ sudo mkdir users
   ```

2. 运行以下命令为客户端创建一个key。

   ```
   $ sudo openssl genrsa -des3 -out /root/ca/users/client.key 1024
   ```

   创建key时需要输入pass phrase，这个是当前key的口令，以防止本密钥泄漏后被人盗用。两次输入同一个密码。

3. 运行以下命令为key创建一个证书签名请求*csr*文件。

   ```
   $ sudo openssl req -new -key /root/ca/users/client.key -out /root/ca/users/client.csr
   ```

   根据提示输入上一步保存的pass phrase后并按回车键，然后根据提示输入要求提供的信息。

   > **说明:** A challenge password是客户端证书口令（请注意将它和client.key的口令区分开），可以与服务器端证书或者根证书口令一致。

4. 运行以下命令使用您的CA Key为刚才的客户端key签名。

   ```
   $ sudo openssl ca -in /root/ca/users/client.csr -cert /root/ca/private/ca.crt -keyfile /root/ca/private/ca.key -out /root/ca/users/client.crt -config "/root/ca/conf/openssl.conf"
   ```

   当出现确认是否签名的提示时，两次都输入y。

5. 运行以下命令将证书转换为大多数浏览器都能识别的*PKCS12*文件。

   ```
   $ sudo openssl pkcs12 -export -clcerts -in /root/ca/users/client.crt -inkey /root/ca/users/client.key -out /root/ca/users/client.p12
   ```

按照提示输入客户端client.key的pass phrase后按回车键，然后根据提示输入用于导出证书的密码。这个是客户端证书的保护密码，在安装客户端证书时需要输入这个密码。

6. 运行以下命令查看生成的客户端证书。

```
cd users
ls
```

21 企业级分布式应用服务EDAS

21.1 产品概述

企业级分布式应用服务（Enterprise Distributed Application Service，简称 EDAS）是企业级互联网架构解决方案的核心产品。EDAS 充分利用阿里云现有资源管理和服务体系，引入中间件成熟的整套分布式计算框架（包括HSF 或 Dubbo 分布式服务化框架、服务治理、运维管控、链路追踪和稳定性组件等），以应用为中心，帮助企业级客户轻松构建并托管分布式应用服务体系。

图 470: EDAS产品示意图



21.2 快速开始

主要介绍如何创建和部署一个简单的 Java Web 应用的完整流程，帮助您快速上手 EDAS。

EDAS 基础版涵盖了应用生命周期管理、资源及容器管理、报警监控、账号体系管理等功能。

完成一个简单应用部署的流程如下：



21.2.1 安装EDAS Agent

用 EDAS 部署应用之前，需要在 ECS 上先安装 EDAS Agent 客户端。EDAS Agent 是 EDAS 服务集群与部署在相应的 ECS 上的应用程序之间进行通信的 Daemon 程序，更多详情请参考*EDAS Agent*简介。

操作步骤

1. 登录EDAS控制台，在左侧菜单栏选择**资源管理** > **云服务器ECS**。
2. 在**实例列表**页面，选择相应的分区。
3. 单击页面右上角**同步ECS** ，将资源同步到EDAS中。

   该操作会花费一些时间，同步完成后，单击刷新即可看到您的ECS资源。
4. 在实例列表页面中，在右上角单击**安装Agent** ，在弹出的对话框中复制安装脚本。

   图 471: Agent安装脚本



5. 以 root 身份登录您的ECS机器，把复制的脚本在这台机器上粘贴执行。安装后，在控制台单击**刷新**，即可看到安装状态。

   图 472: 实例安装状态



21.2.2 创建应用

完成 EDAS Agent 安装后，您就可以开始创建应用了。

操作步骤

1. 在 EDAS 控制台左侧菜单栏选择**应用管理**，单击页面右上方的**创建应用**。
2. 在**创建应用**对话框输入相应信息，单击**下一步**。

图 473: 创建应用



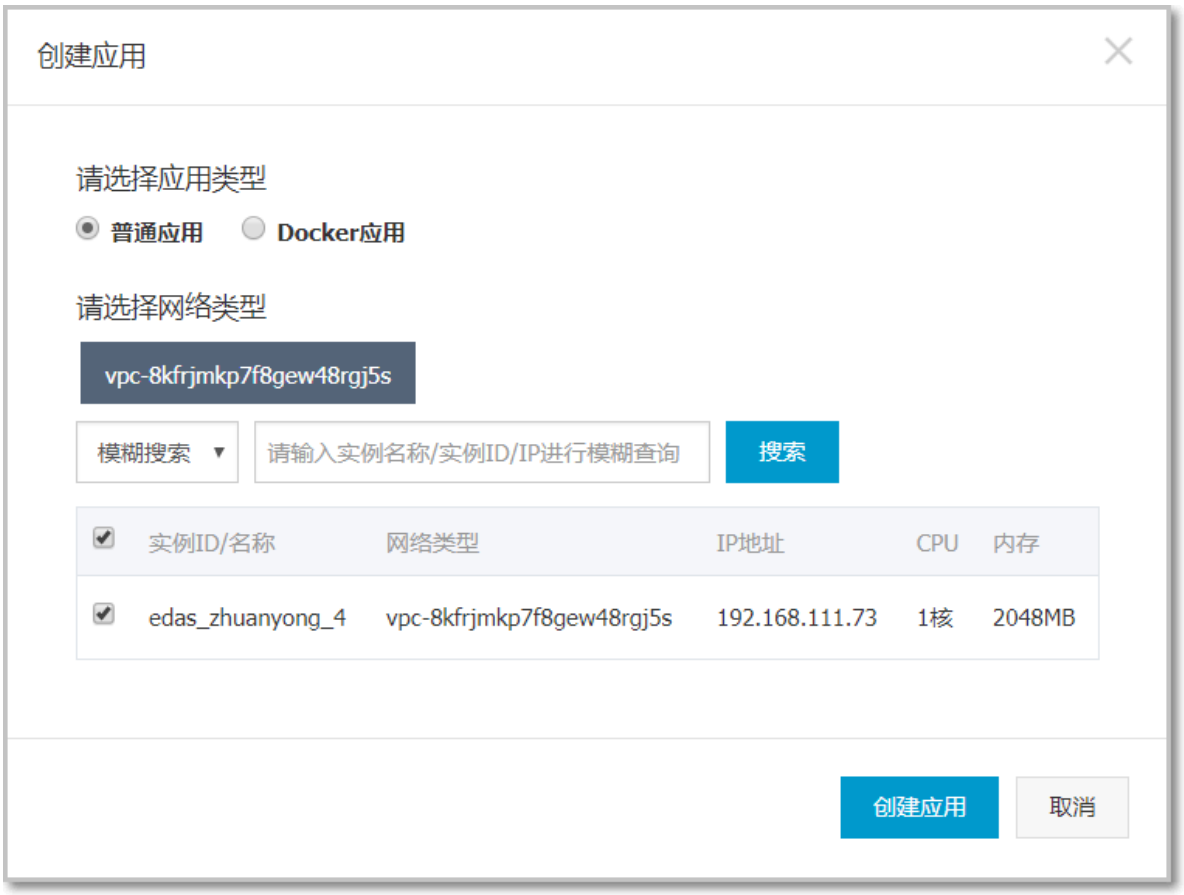
表 73: 创建应用参数说明

参数	说明
应用运行环境	如无特殊需求，请选择最新版本。必选项。
应用名	请尽量选择有意义的名称。必选项。
应用所在区域	请选择您的 ECS 机器所在的分区。必选项。
应用健康检查	EDAS定期访问该设置的URL，根据其响应状态，确定应用的存活状态。若不配置，EDAS不进行应用的健康检查，但不影响应用的正常运行。可选项。
备注	该应用的备注信息。可选项。

3. 选择应用类型、网络类型及相应实例，单击**创建应用**完成创建。

   - 应用类型：普通应用或Docker应用。Docker应用即在Docker中部署应用。
   - 网络类型：要创建的应用所在的分区。
   - 实例：选择一个至少拥有一台 ECS 机器的环境，并且确保该 ECS 已经安装 EDAS Agent。

图 474: 应用类型和网络类型



应用创建完成后会出现在应用列表里。

21.2.3 部署和启动应用

创建好应用后，就可以上传示例 WAR 包进行部署，并启动应用。

操作步骤

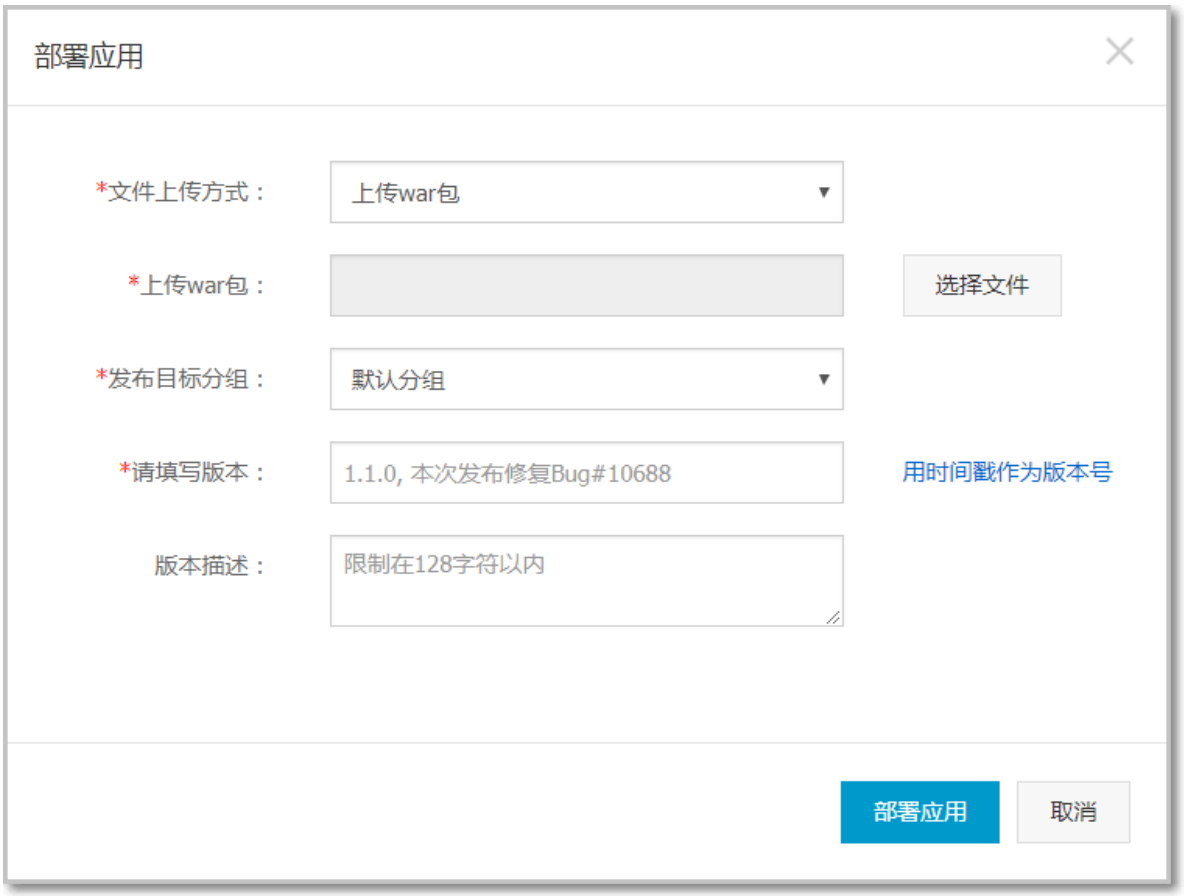
1. 点击此处下载示例 WAR 包。

   说明: 如您想了解 WAR 包开发的规范，请参考 《*EDAS* 开发指南》。

2. 在控制台左侧菜单栏选择**应用管理**。
3. 在**应用列表**页面单击刚刚创建的应用名称，进入应用基本信息页面，然后单击页面右上角的**部署应用**。
4. 在**部署应用**对话框中，设置部署参数，然后单击**部署应用**。

说明: 部署普通应用和部署 Docker 应用的参数稍有不同，本文档以部署普通应用为例进行说明，部署 Docker 应用的参数说明请参见图 *502:* 部署 *Docker* 应用图 *502:* 部署 *Docker* 应用

图 475: 部署应用



部署应用参数说明：

- 文件上传方式：
  - 上传 WAR 包：选择上传 WAR 包后，在下面上传 WAR 包右侧单击 **选择文件**，打开本地文件夹，选择要部署的 WAR 包 。

    如果是选择上传 WAR 包方式，可能需要等待片刻，等待 WAR 包上传，待进度条完成100%后，则文件上传成功。
  - WAR 包地址：选择 WAR 包地址后，在下面 WAR 包地址右侧的文本框中输入定存放 WAR 包且可以访问的 URL 地址。
  - 使用历史版本：选择使用历史版本后，在历史版本右侧的下拉框中选择具体历史版本。

说明： 如果选择使用历史版本，对话框会发生变化，无需再设置请填写版本和版本描述信息。

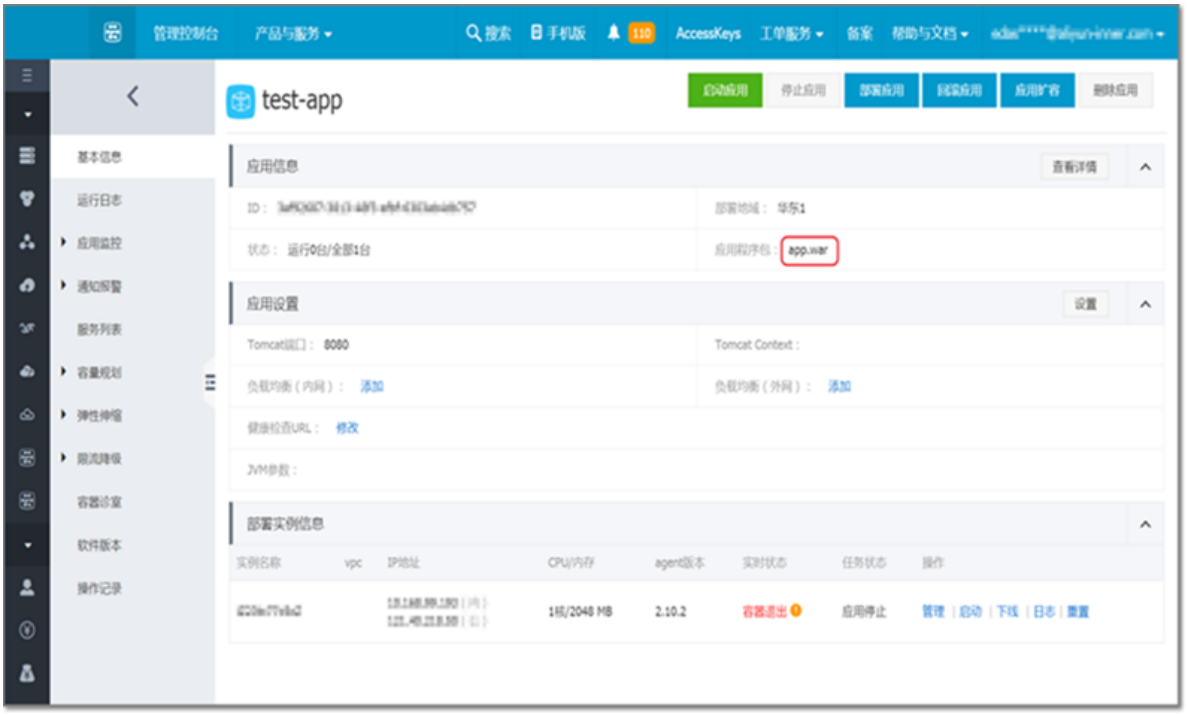
- 发布目标分组：部署此应用的分组。
- 请填写版本（适用于上传 WAR 包和 WAR 包地址两种文件上传方式）：填写 WAR 包的版本。

  应用版本用于标识一次应用发布所使用的部署包的版本，能够帮助用户很好的区分每一次应用发布的部署包版本，并在回滚操作的时候，能够精准的跟踪到某一次发布。

  注意：部署应用的时候，可以添加一个版本号或者文字描述，不建议使用 **用时间戳作为版本号。**
- 版本描述：对此 WAR 包版本进行描述。
- 历史版本（仅适用于使用历史版本的文件上传方式）：在下拉框中选择要使用的历史版本。

部署完成后，该应用的基本信息页签中的应用程序包会更新为刚刚部署WAR包信息。

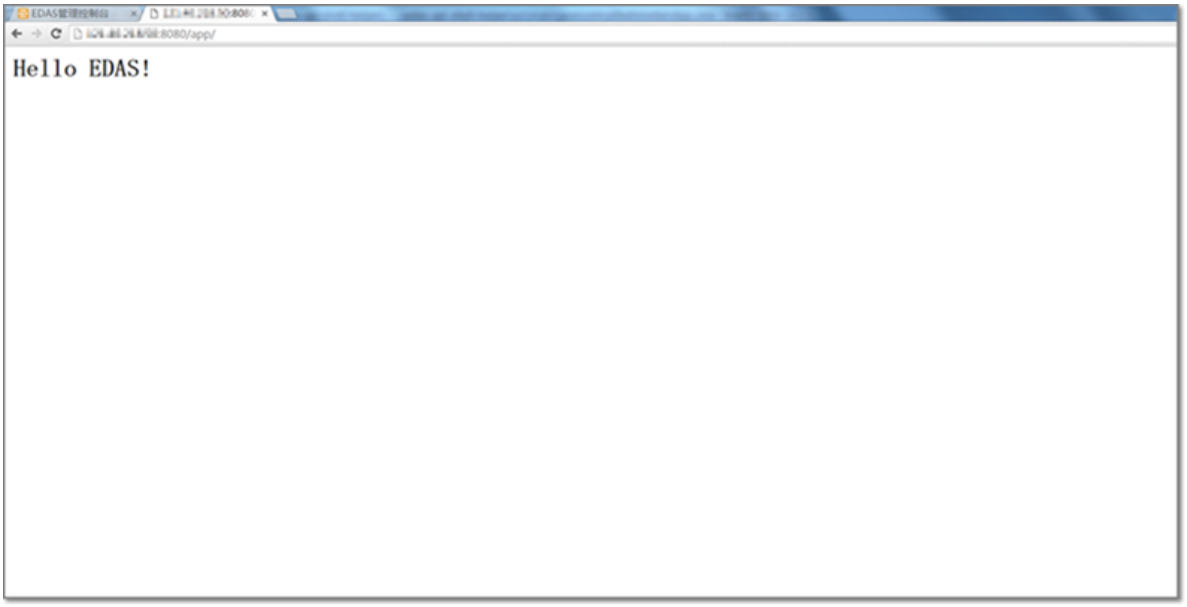
图 476: 查看应用程序包



5. 在**应用详情**页面单击右上方的**启动应用**。

应用启动成功后，在浏览器地址栏输入*http://ECS IP:8080/app* 并回车，即可进入以下EDAS的欢迎页面。

图 477: EDAS欢迎页面



21.3 概览页介绍

EDAS 控制台概览页主要提供了账户的统计信息。

概览页中包括：

- **套餐类型**：当前用户所使用的套餐类型。
- **运行概况**：当前用户拥有的 ECS 资源个数、应用个数、应用实例（包括独占实例和 Docker 实例）个数，方便了解当前账户的资源情况。其中各个数字表达的含义如下：
  - ECS 资源个数：代表您一共拥有多少台 ECS 机器。
  - 应用个数：您一共创建了多少个应用。
  - 应用实例个数：您的所有应用在 EDAS 节点上一共部署了多少个实例。

图 478: 概览页面



21.4 资源管理

介绍 EDAS 涉及的资源，以及如何使用、管理这些资源。

您可以通过 EDAS 控制台，查看和使用ECS、SLB等资源。 EDAS 的资源管理只是在应用层面使用这些资源，而不会对这些资源进行新购，释放等操作。 此外 EDAS 提供了资源分组管理的能力，可以在多人或者多部门使用 EDAS 的情况下，利用主子账号来控制资源使用权限。

21.4.1 云服务器ECS

介绍如何通过EDAS控制台使用和管理ECS。

21.4.1.1 管理ECS实例

ECS 实例同步到 EDAS 控制台后，可以通过 EDAS 控制台，对这些 ECS 实例进行查看和管理。

操作步骤

1. 登录 EDAS 控制台。
2. 在左侧导航栏中，选择**资源管理** > **云服务器 ECS** > ，进入（ECS）实例列表页面。

图 479: 实例列表页面



3. 选择具体区域（Region），查看该区域中的 ECS 实例的信息和状态。

   ECS 信息/状态说明

   - 实例 ID/名称：实例 ID 为系统自动生成，名称是创建实例时用户定义的。
   - VPC（全部）：该实例所在VPC的ID。
   - IP 地址：该实例在VPC中的IP地址。
   - 状态：实例当前的状态，包括运行中、已停止等。
   - 配置：实例的 CPU 和内存配置。
   - 部署应用：该实例上部署的应用。单击应用名称，可以跳转到该应用的详情页面。
   - 已安装 Agent：该实例上是否已经安装了 Agent。
   - Agent 状态：
     - 如果该实例上已经安装了 Agent，其状态包括在线、Docker 在线等。
     - 如果未安装 Agent，状态为未知。

4. 单击实例列表右上方的按钮，可以对 ECS 进行如下操作：

操作	说明
刷新	刷新实例列表
同步 ECS	将 ECS 实例同步到 EDAS
安装 Agent	在 ECS 实例上安装 EDAS Agent

21.4.1.2 安装 Agent

介绍 EDAS Agent 及其安装方法。

21.4.1.2.1 EDAS Agent 简介

介绍EDAS Agent的定义、功能及作用。

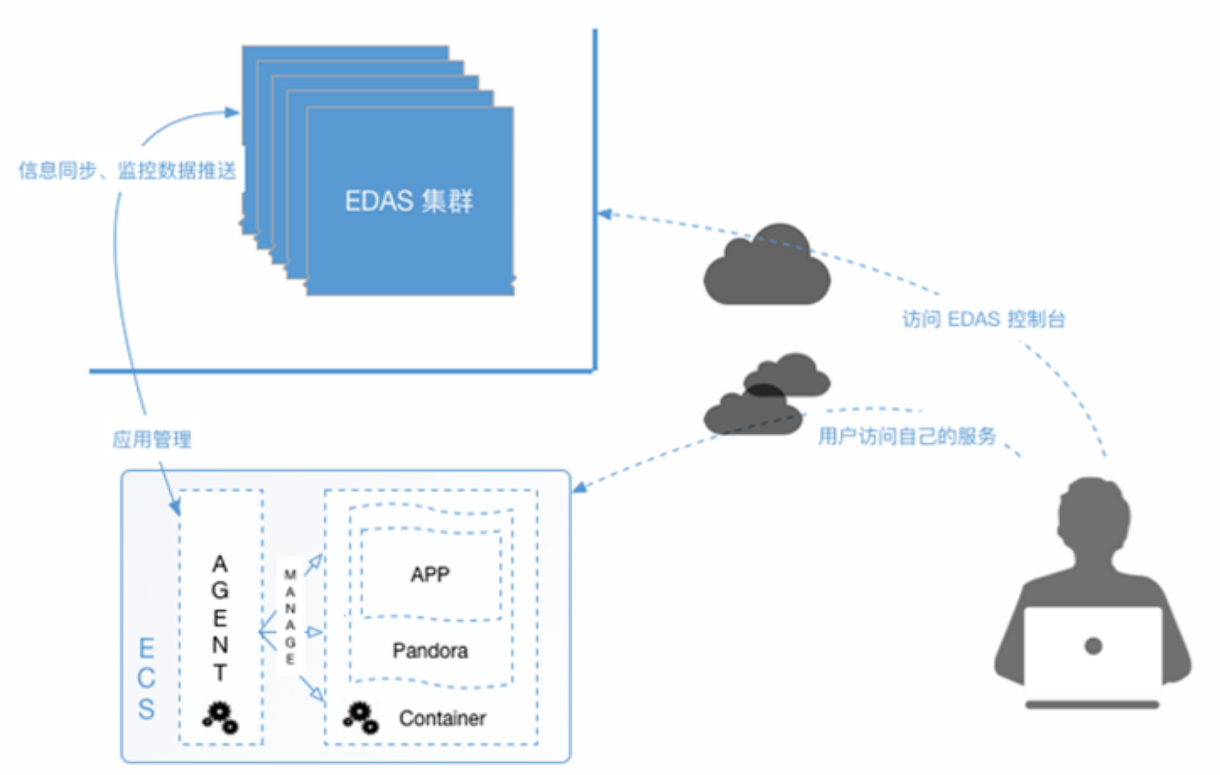
EDAS Agent（以下简称 Agent）是 EDAS 中安装在用户 ECS 上，主要用于 EDAS 服务集群与部署在相应的 ECS 上的应用程序之间进行通信的 Daemon 程序，在运行的过程中主要承担以下的角色：

- 应用管理：包括应用部署、启动、停止等。
- 状态回报：包括应用存活状态、健康检查状态、AliTomcat 容器状态等。
- 信息获取：如获取 ECS 和容器的监控信息。

除了以上基于应用的管控功能，Agent 还是 EDAS 控制台与用户应用程序之间信息沟通的主要桥梁；简单的说，一个应用发布的服务是否在某台 ECS 上正确及时的发布，这个简单的信息获取就需要 Agent 的参与和协调。

说明: 上述 Agent 所涉及的功能对用户都是透明的，作为用户，您只需要安装 Agent 即可。

图 480: Agent在EDAS中的位置及作用示意



21.4.1.2.2 安装 Agent

EDAS 只会选择安装了 Agent 的 ECS 进行应用的部署（包括第一次安装、扩容）。EDAS 计费系统中的节点的概念，也是指安装了 Agent 并且部署了应用的 ECS。所以使用 EDAS 的第一个步骤就是安装 Agent。

背景信息

说明:

- 该脚本需要先以 root 身份登入至用户的 ECS 终端手动执行。
- 目前 Agent 的安装及其运行环境仅支持 CentOS 6.5 64 位与 Ali-Linux 5.7 64 位。
- 该脚本可以重复执行，重复执行的结果是覆盖安装，所以 Agent 没有提供单独的升级脚本，即当需要升级 Agent 时，需要执行和安装时同样的脚本。
- 目前不同 Region 的 ECS 机器所使用的安装脚本是不同的，在选择安装脚本之前一定要切换到 ECS 所对应的 Region 列表，然后再单击 **安装Agent** 。

操作步骤

1. 登录EDAS 控制台，在左侧导航栏中选择**资源管理** > **云服务 ECS** 。
2. 在页面的左上角选择 ECS 所在区域。
3. 在 ECS 实例列表页面，选择相应的 ECS，单击右上角的**安装Agent**，如下图所示：
4. 在 **EDAS Agent安装**对话框，复制安装脚本。

图 481: EDAS Agent安装页面



在默认的列表中，您可能无法找到对应的 ECS，造成这种情况一般存在以下原因：

- 实例没有在相应的区域中，可在左上角的区域列表中进行切换后确认。
- 后台的 ECS 列表没有同步完成，由于自动同步方式方式会存在一定时延，如切换相应的区域后仍未发现相应的 ECS ，可点击右上角的**同步ECS**触发手动同步。
- 如果手动同步依然失败，请检查阿里云上主账户的 AK/SK 是否启用。

5. 在 ECS 上安装 Agent。

a) 以 root 账户登录该 ECS。

b) 在 ECS 终端中，粘贴刚刚拷贝命令并执行。

c) 稍等片刻，脚本执行成功后会出来如下信息：

```
[root@iHostname]# wget -q -O /root/install.sh http://edas-hz.oss-cn-hangzhou-internal.
 aliyuncs.com/install.sh && sh /root/install.sh -ak {your-access-key} -sk {your-secret- k ey}
....
....
..
Edas Install Success!
```

d) 在机器上确认 Agent 是否运行正常。

```
[root@iHostname]# pgrep -f 'AgentDaemon' && egrep 'Register ecu' /home/ admin/ edas-
agent/logs/agent.log
14:25:36 [main] INFO com.alibaba.edas.agent.AgentDaemon - Register ecu into edas
 successfully
```

e) 在 EDAS 控制台中确认相应的 ECS 是否已经安装了正确的版本，如下图：

图 482: 确认ECS安装版本



后续操作

升级EDAS Agent

升级EDAS Agent的步骤与安装步骤一致，Agent会自动覆盖之前安装的版本并自动重启。

21.4.2 负载均衡 SLB

介绍如何通过EDAS控制台查看SLB实例相关信息。

操作步骤

1. 登录 EDAS 控制台。
2. 在左侧导航栏中，选择**资源管理** > **负载均衡 SLB** > ，进入（SLB）实例列表页面。

图 483: SLB 实例列表



3. 选择具体区域，查看该区域中的SLB实例信息和状态。

   SLB 信息/状态说明

   - 实例 ID/名称：SLB实例的 ID，是系统自动生成的。
   - 服务地址：您拥有的SLB服务的内部IP地址。
   - 后端服务器：在 SLB 控制台添加的，用于接收负载均衡实例分发的请求的 ECS 实例。
   - 状态：SLB 服务器的状态，包含两种：运行中和已停止。 已过期的 SLB 不会显示。

21.4.3 专有网络 VPC

专有网络 VPC 是一个可以自定义隔离专有网络， 您可以自定义这个专有网络的拓扑和 IP 地址，适用于对网络安全性要求较高和有一定的网络管理能力的客户。

本节主要介绍如何通过EDAS控制台查看、管理VPC，以及如何在VPC实例上安装日志采集器。

21.4.3.1 查看 VPC 实例

介绍如何通过EDAS控制台查看VPC实例信息。

操作步骤

1. 登录 EDAS 控制台。
2. 在左侧导航栏中，选择**资源管理** > **专有网络 VPC** > ，进入（ VPC ）实例列表页面。

   图 484: VPC 实例列表



3. 选择具体区域，查看该区域中的VPC实例信息和状态。

   1. VPC 实例信息说明：

      - VPC ID：VPC 创建时系统自动生成的。
      - 名称：VPC 创建时用户定义的。
      - 网段：VPC 的网段，在创建 VPC 时设置。
      - 状态：VPC 的状态包含两种：运行中和已停止。 已过期的 VPC 不会显示。
      - ECS 实例：显示该 VPC 网络下创建的 ECS 的实例个数。 单击实例个数可以跳转到 ECS 页面，查看该 VPC 内的所有 ECS。

后续操作

在 VPC 网络内，ECS 实例与 EDAS 服务器隔离。所以需要安装日志采集器来收集 ECS 的信息。单击实例列表操作列中的 **安装日志采集器** 进行安装。安装日志采集器的具体步骤，请参考 安装日志采集器。

21.4.3.2 安装日志采集器

在 VPC 网络内，ECS 实例与 EDAS 服务器隔离。所以需要安装日志采集器来收集 ECS 的信息。单击实例列表操作列中的 **安装日志采集器** 进行安装。

背景信息

必须在 VPC 内部选取一台机器（假设为机器 A）来安装日志采集器。具体安装步骤如下：

操作步骤

1. 登录 EDAS 控制台。
2. 在左侧导航栏中，选择**资源管理** > **专有网络 VPC** 。
3. 进入机器 A 所在的 区域，在VPC 列表中找到机器 A 所在的 VPC ID，单击操作列的 **安装日志采集器** 。
4. 在弹出的对话框中，复制安装脚本命令。
5. 以 root 身份登录机器 A，粘贴之前复制的安装脚本命令，并单击 **回车** 执行。
6. 在安装过程退出之后，手动执行 **netstat -ant|grep 8000** 命令。

   - 如果有连接建立，说明日志采集器安装成功.
   - 如果没有连接建立，说明安装存在问题，请提交工单与我们联系。

21.4.4 集群管理

集群由Docker主机组成，用于部署应用。创建应用后，集群不可更改。应用的扩容缩容都在应用所属集群内完成。

21.4.4.1 查看集群

介绍如何查看集群的信息和状态。

操作步骤

查看集群列表

1. 登录EDAS控制台。
2. 在左侧导航栏中选择**资源管理 > 集群管理**。
3. 在集群列表页面，选择区域（Region），查看该区域中的集群信息。

图 485: 集群列表

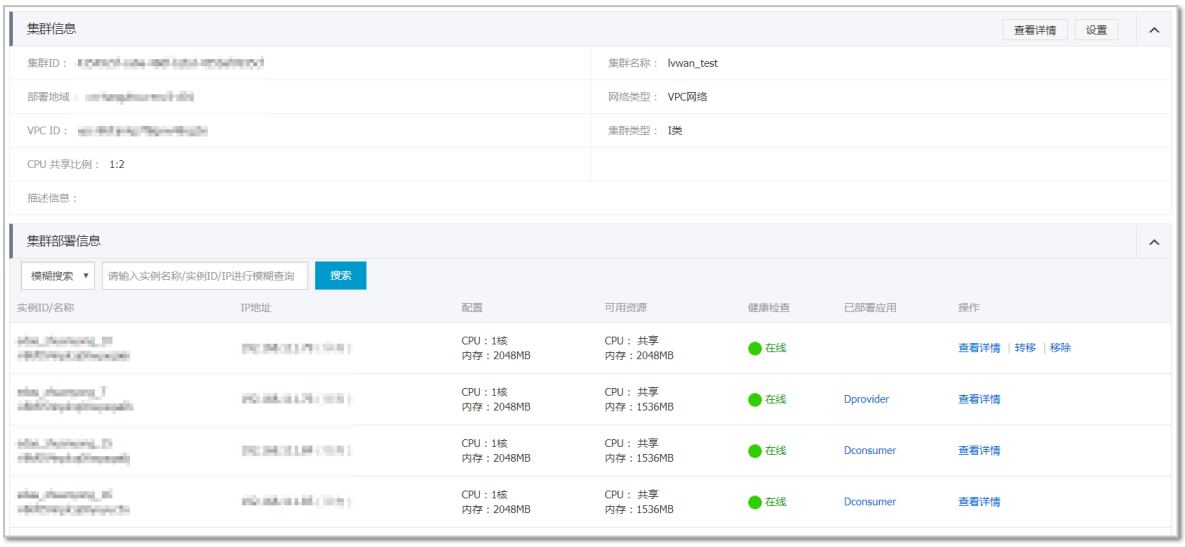


集群列表中包含个集群的名称/ID、网络类型、节点数量、资源量和状态等信息。

查看集群详细信息

4. 在集群列表页面，单击集群名称。
5. 在集群详情页面，查看集群详细信息。

图 486: 集群详细信息



集群详情页面主要包括两部分：集群信息和集群部署信息。

- 集群信息可以查看集群详情，修改集群信息。
- 集群部署信息展示集群 Docker 主机列表。

21.4.4.2 创建集群

介绍如何创建集群。

操作步骤

1. 登录EDAS控制台。
2. 在左侧导航栏中选择**资源管理 > 集群管理**。
3. 在集群列表页面右上角，单击**创建集群**。
4. 在**添加集群**对话框中，设置集群参数。

图 487: 添加集群



集群参数说明：

- 集群名称：名字仅支持字母、数字、下划线和点且长度不能超过64个字符。
- 集群网络类型：分为经典网络和 VPC 网络，选择 VPC 网络，需要用户提前创建 VPC 网络。

5. 设置完成后，单击**添加**。

   集群创建完成后，可以返回集群列表，查看集群是否创建成功以及相关信息和状态。

21.4.4.3 管理集群

集群主要由 Docker 主机组成，所以对集群的管理是对 Docker 主机的管理，包括添加Docker 主机、转移 Docker 主机和删除 Docker 主机。

操作步骤

添加 Docker 主机

1. 登录EDAS控制台。
2. 在左侧导航栏中选择**资源管理** > **集群管理**，在集群列表页面中单击具体集群名称。
3. 在集群详情页面的右上角，单击**添加集群主机**。
4. 在转化 Docker 主机页面的实例列表中，选择要转化的实例，然后单击 **下一步**。
   - 如果有已经转化过的 Docker 主机（但不属于任何集群），页面右上角会提示“您有docker主机可以直接加入集群，点击加入”。单击此链接，可以直接将该 Docker 主机加入集群。

- 如果没有已完成转化的 Docker 主机，页面会提示 “没有查询到任何的Docker主机。跳转云服务器ECS菜单为ECS安装Docker” 。单击此链接，跳转到ECS实例列表页面，为ECS实例安装 Docker。

5. 系统对 Docker 主机进行健康检查。

   健康检查分为转化中、转化失败、在线和离线四种状态。

   - Docker 主机转化开始，状态为**转化中**。
   - 根据转化结果，有**转化失败**和**在线**两种情况。
   - **离线**状态是指 EDAS 不能检测到 Docker 主机的心跳。

   Docker 主机开始转化流程后，转化状态会在集群部署信息列表中显示。

6. 输入重装操作系统后 ECS 的登录密码，单击 **下一步**。
7. 在弹出的提示页面，确认并单击 **转化**，转化任务将提交给 EDAS 服务器来执行。

   此时，转化操作已完成，返回集群页面看到相应的节点处于 **转化中** 的状态，整个转化过程大约需要5分钟左右。在此期间，您可以在 EDAS 控制台做别的操作。

   转化完成后，返回集群列表页面，检查转化结果。

   - 如果转化成功，节点状态为 **在线** 状态，此时，这些节点资源已经可以用来部署应用了。
   - 如果转化中遇到了异常情况导致 ECS 转化失败，则主机节点则处于 **转化失败** 状态。此时，需要定位、排查失败的原因，然后单击主机节点后面的 **重试**。

转移 Docker 主机

8. 在集群详情页面的集群部署信息区域的 Docker 主机列表的操作列，单击 **转移**。
9. 选择 Docker 主机要转移的目标集群，单击 **确定** 。

   Docker 主机集群间转移成功后，查询集群列表，可以看到集群的资源在 Docker 主机转移成功时同步更新。

删除 Docker 主机

10. 在集群详情页面的集群部署信息区域的 Docker 主机列表的操作列，单击 **移除**，可以移除 Docker 主机，移除的 Docker 主机在重新添加集群时不需要再重新转化。

21.4.5 资源组管理

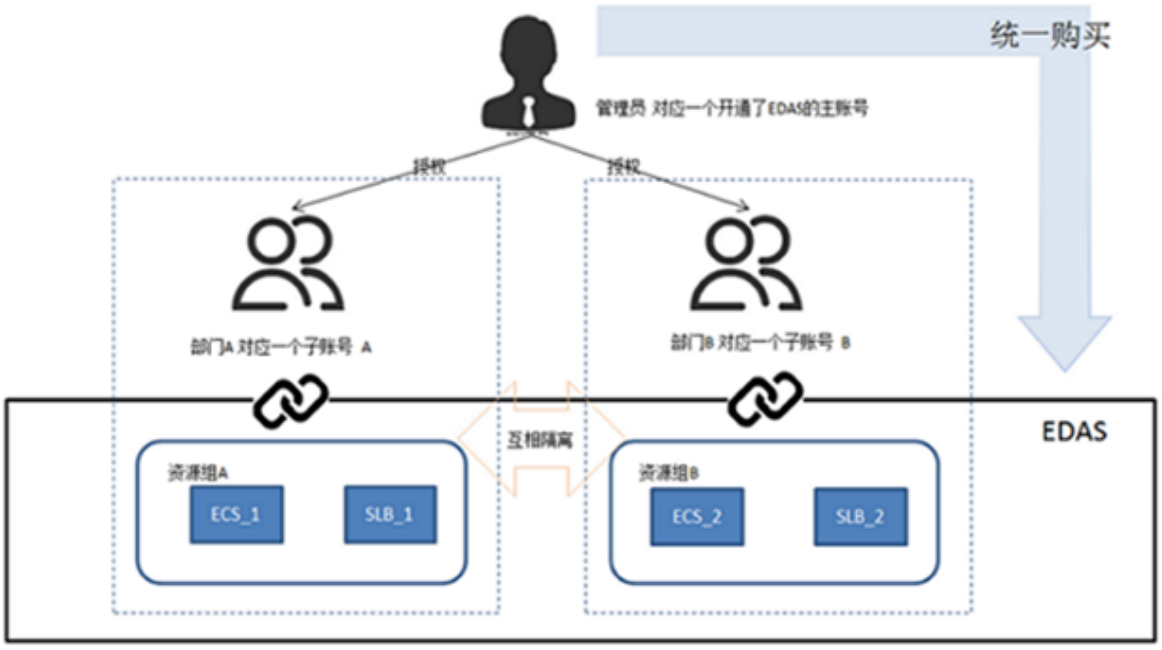
资源组是 EDAS 对资源的一种划分，资源组只能绑定 ECS 或者 SLB，不能绑定 VPC。您可以使用资源组对账号的权限进行控制。 EDAS 可以对子账号进行资源组授权，被授权的子账号拥有操作这个资源组中所有资源的权限。

背景信息

典型应用场景

- 某公司使用了 EDAS 搭建自己的业务应用。其中部门 A 负责用户中心的应用，部门 B 负责商品中心的应用。
- 两个部门共同使用了一个EDAS主账号开通的 EDAS 服务，并分别有一个子账号。部门 A 有几台专用 ECS和SLB实例，用于部署用户中心的应用。
- 该公司在使用 EDAS 的时候创建了一个资源组，向这个资源组中添加部门 A 的 ECS 和 SLB 实例，然后将这个资源组授权给部门 A 的使用的子账号。
- 部门 A 在使用这个子账号时只能操作这个资源组中的资源。部门 A 和部门 B 之间不会产生冲突。

图 488: 资管组管理应用场景示意图



操作步骤

查看资源组

1. 登录EDAS控制台。

2. 在左侧导航栏中选择**资源管理** > **资源组管理** 。
3. 选择区域（Region），可以查看该区域内的资源组以及各资源组中的 ECS 和 SLB 实例。
4. 在实例列表中可以查看资源组的信息，包括资源组名称、描述、ECS 实例信息（内网 IP 地址），和 SLB 实例信息。

图 489: 资源组实例列表



创建资源组

5. 在实例列表页面右上角单击 **创建资源组**。
6. 输入资源组名称和描述，单击 **确定**。

编辑资源组

7. 在在实例列表要编辑的资源组的操作列中，单击 **编辑**。
8. 在编辑资源组对话框中，对资源组名称和描述信息进行编辑，然后单击 **确定**。

添加 ECS 到资源组

9. 在实例列表需要添加 ECS 的资源组的操作列中，单击 **绑定 ECS**。
10. 在弹出的绑定 ECS 对话框中，选择要绑定的 ECS 实例，单击 **确定**。

添加 SLB 到资源组

11. 在实例列表要添加 SLB 的资源组的操作列中，单击 **绑定 SLB**。
12. 在弹出的绑定 SLB 对话框中，选择要绑定的 SLB 实例，单击 **确定**。

添加集群到资源组

13. 在实例列表要添加 SLB 的资源组的操作列中，单击 **绑定集群**。
14. 在弹出的绑定集群对话框中，选定要绑定的集群名称，单击 **确定**。

对子账号授权资源组

15. 以主账号身份登录 EDAS 控制台。
16. 在控制台左侧菜单栏选择 **账号管理** > **子账号管理**。

17.在要授权的用户的操作列，单击 **授权资源组**。

18.在授权资源组对话框中选择资源组，单击 **确定**。

删除资源组

19.在资源组列表中要删除的资源组的操作列中，单击 **删除**。

20.在弹出的确认框中单击 **确定**，即可删除一个资源组。

21.5 使用 Docker 部署应用

EDAS 支持使用 Docker 来部署应用，从而提升客户的资源使用率。本文档将简单介绍在 EDAS 使用 Docker 部署应用的流程。

操作步骤

1. 创建 Docker 宿主机集群，并将 Docker 实例添加到该集群中。

   详细步骤，请参考创建集群 和 管理集群。

   说明:

   - 安装 Docker 应用的 ECS 上不需要手动安装 EDAS Agent。
   - 目前 EDAS 所部属的 Docker 应用使用的 host 网络模式为共享宿主机的 IP 及网络，后期在 VPC 环境 Docker 应用将拥有独立IP地址及独立网络。

2. 创建应用。

   使用 Docker 创建应用，请参考创建应用 中 Docker 相关内容。

3. 部署应用。

   使用 Docker 部署应用支持上传 WAR 包和 Docker 镜像两种方式。

   - 上传 WAR 包

     部署方式与 EDAS 普通应用一致。详细步骤，请参考部署应用 。WAR 包上传后，将运行在 Docker 容器内部。

   - Docker 镜像

     1. 准备工作
     2. 镜像定制
     3. 部署镜像

   本节重点介绍如何通过 Docker 镜像部署应用。

EDAS 还提供了 Docker 应用部署的 Open API。用户可以自行搭建 Jenkins 持续集成平台，通过调用 EDAS 的 Open API 来实现代码提交后，自动打包成 Docker 镜像，并部署在 EDAS 平台上。

注意：

- 应用部署后，可以通过 EDAS 控制台的应用管理页面进行监控和管理。
- Docker 可以通过 EDAS 控制台的资源管理下的集群管理进行监控和管理。

21.5.1 准备工作

使用 Docker 镜像部署应用前，需要先 开通镜像仓库 和 准备打包环境。

操作步骤

开通镜像仓库

1. 登录镜像仓库控制台。
2. 根据提示，创建默认的 namespace，并设置默认密码。

准备打包环境

3. 准备 Docker。

   当前打包方式，需要 Docker 服务并可正常访问阿里云镜像仓库。

   Linux 用户，请采用 Centos 7.0/Ubuntu 12.04 以上版本操作系统，确保操作系统内核版本满足 Docker 运行基本条件。

   - 方式一：（推荐）ECS 通过 EDAS 控制台转化为 Docker 主机。

     选择任意一台集群中的机器，登录主机即可。

   - 方式二：本地安装（官方参考文档）。

     - Linux 安装
       - 文档：*https://docs.docker.com/engine/installation/linux/*
       - 命令行：

         ```
         curl -sSL http://acs-public-mirror.oss-cn-hangzhou.aliyuncs.com/docker-engine/internet | sh -
         ```

     - Mac 安装： *https://docs.docker.com/docker-for-mac/*

4. 配置镜像加速。

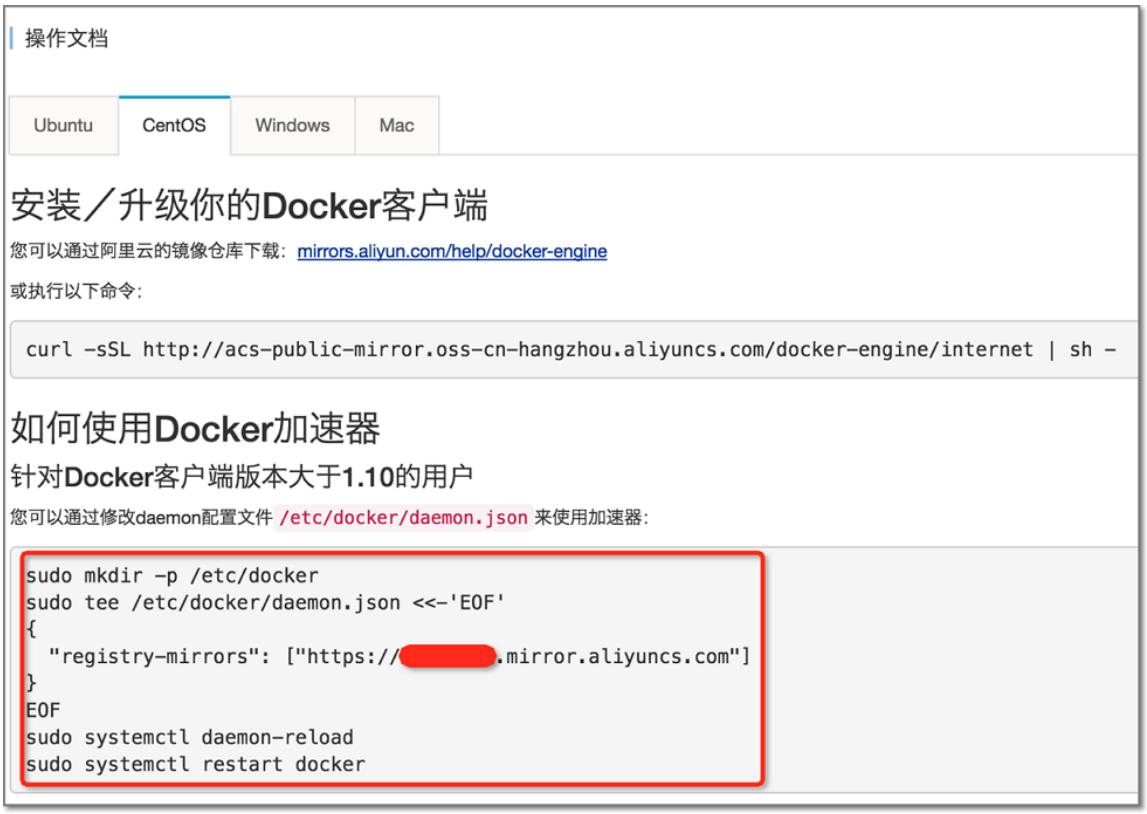
   a) 访问镜像仓库控制台，获取个人专属镜像加速通道。

b) 配置镜像加速。

根据本地打包环境，参考页面对应操作系统的操作文档，根据提示命令，配置镜像加速地址。

如果打包环境操作系统为 CentOS，则拷贝下方命令，在安装了 Docker 准备打包镜像的机器上粘贴执行。

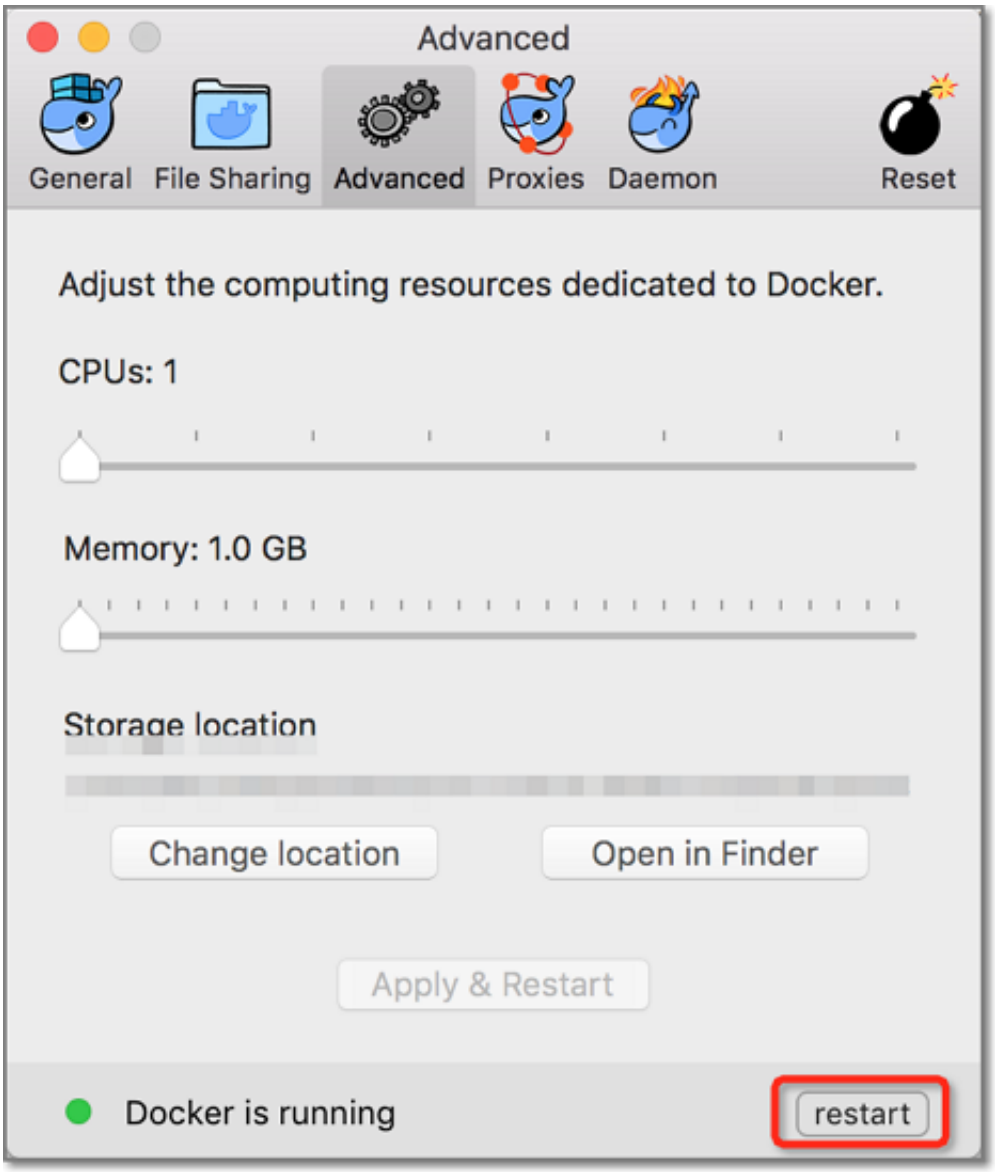
图 490: 配置镜像加速地址



5. 重启 Docker 服务。

- Linux 系统：上一步命令中，会自动重启服务。
- Mac 系统：若采用 Docker Engine，单击下方 **restart** 按钮即可。

图 491: 在 Mac系统中重启 Docker服务



6. 下载 EDAS 基础镜像。

   a) 为了制作镜像，需要下载应用对应版本的基础镜像。

   此版本号为创建应用时选择的“应用运行环境”。如下图中的 EDAS-Container 3.2。

   图 492: 创建应用对话框



b) 确定应用版本号。

点击已创建的 Docker 应用，在左侧导航栏中单击软件版本，确认应用版本号。

图 493: 容器版本页面



c) 拉取镜像。

执行如下命令，拉取镜像。

其中，需要将 3.2 替换为实际需要拉取的镜像版本号。若发现右侧更新时间变更，建议更新基础镜像，重新打包。

```
docker pull registry.aliyuncs.com/edas/edas-container:3.2
```

21.5.2 镜像定制

介绍如何制作定制镜像。

约束要求和建议

- 请勿更改 Tomcat 的默认启动脚本、目录与日志路径，确保可以获得完整的 EDAS 产品功能。
- 随着 EDAS 新应用版本发布，需要重新打包发布镜像，拥有 EDAS 新功能特性。若距离上次打包镜像发布的时间较长，建议更新基础镜像，重新打包发布。详情参考 下载 *EDAS* 基础镜像。
- 目录清单
  - Tomcat 目录：*/home/admin/taobao-tomcat-production-7.0.59.3/*
  - 日志目录： */root/logs/*
  - WAR 包目录：*/home/admin/taobao-tomcat-production-7.0.59.3/deploy/*

基础镜像定制

基础镜像定制有以下3种方式：

- 方式一：编写 Dockerfile，向容器内复制本地文件，或执行命令

  在拉取所需的基础镜像后，可通过编写 Dockerfile 方式，制作标准镜像。如下为 **发布 WAR 包** 与 **修改 Tomcat 配置** 两种示例。

  注意: 请勿使用 CMD 命令自定义镜像启动脚本。

  例 1：将本地应用 /tmp/edas-app.war 放入镜像，以镜像方式发布应用。

  ```
  FROM registry.aliyuncs.com/edas/edas-container:3.2
  ADD /tmp/edas-app.war /home/admin/taobao-tomcat-production-7.0.59.3/deploy/
  ```

  例 2：替换 server.xml，并安装系统监控工具 sar。

  ```
  FROM registry.aliyuncs.com/edas/edas-container:3.2
  ADD /tmp/server.xml /home/admin/taobao-tomcat-production-7.0.59.3/conf/server.xml
  RUN yum install -y sysstat
  ```

  Dockerfile 编写完成后，可直接发布镜像

- 方式二：使用本地 Docker 命令，向容器内复制本地文件。

与文件复制 cp 命令基本一致，执行如下命令，将复制本机 /tmp/app.war 到打包环境 WAR 包目录中。

格式：

docker cp	本地文件路径	edas-build-package:容器文件路径
docker cp	/tmp/app.war	edas-build-package:/home/admin/taobao-tomcat-production-7.0.59.3/deploy/

上方命令为示例，实际执行命令如下。作用为，复制本地文件 /tmp/app.war 至容器内 WAR 包部署目录：

```
docker cp /tmp/app.war edas-build-package:/home/admin/taobao-tomcat-production-7.0.59.3/deploy/
```

- 方式三：进入 Docker 容器，下载远程文件

  1. 执行命令 **docker exec -it edas-build-package bash**，进入容器。
  2. 执行命令 **wget** *http://anything-you-want/*本地保存路径,下载文件。

- 执行命令 docker exec -it edas-build-package bash，进入容器。
- 执行命令 wget *http://anything-you-want/*本地保存路径,下载文件。

部署 WAR 包

- 默认部署目录，请勿更改。
- 参考 向镜像内复制本地文件/在镜像内下载远程文件 步骤，复制 WAR 包到部署目录。

21.5.3 部署镜像

介绍如何部署镜像。

背景信息

基础镜像环境清单

EDAS Docker 容器环境，内置了以下基础服务，为 EDAS 应用运行的基线要求。

- CentOS 7
- JDK 7（默认）或 JDK 8（可选）
- Ali-Tomcat 7

操作步骤

1. 创建容器仓库镜像。

a) 登录 Docker 镜像仓库控制台，在左侧导航栏中单击 **镜像列表**，在镜像仓库列表页面右上方单击 **创建镜像仓库**。

b) 根据项目特性，设置 **仓库名称**，**仓库类型**，**代码源** 为 **本地仓库** 等必填信息。

图 494: 创建镜像仓库



c) 点击 **创建镜像仓库**。

注意: 代码若需要保密性，请设置“仓库类型”为“私有”。

后续示例，将默认使用此配置。若需要更换配置，请参考文章最后速查部分，进行替换。

2. 登录远程仓库。

a) 登录镜像仓库控制台，在镜像列表页面，单击 **修改 docker 登录密码**，设置登录密码。

b) 找到刚刚创建的镜像仓库，点击 **管理**。

c) 找到页面中提供的，登录到阿里云 docker registry 的代码，复制到终端执行，并输入密码。

图 495: Demo-frontend-service




```
> docker login --username=**********@aliyun.com registry.cn-hangzhou.aliyuncs.com
< Password:
< Login Succeeded
```

提示 Login Succeeded 则登录成功。

3. 更新/创建基础镜像容器。

若需要重新打包基础镜像，请重新初始化整个打包环境，清空数据。执行前请注意备份数据。

a) （首次创建请跳过）备份代码与变更文件。

b) （首次创建请跳过）参考 下载 EDAS 基础镜像，更新本地基础镜像。

```
docker pull registry.aliyuncs.com/edas/edas-container:3.0.0
```

c) 执行命令，初始化打包环境，清空已有数据（注意替换镜像版本号）。

```
docker rm -f -v edas-build-package
docker run --name edas-build-package -d --restart=always registry.aliyuncs.com/edas/edas-container:3.0.0
```

4. 打包本地镜像。

完成基础镜像定制后，根据打包方式不同，分别执行以下 shell 进行镜像打包。此时，打包后的镜像保存在本地。

注意: 将最后的镜像地址，替换成之前在 创建容器仓库镜像 中创建的镜像仓库名，并填写一个易于辨识的版本号。

以下方命令为示例，打包服务 *registry.cn-hangzhou.aliyuncs.com/edas/demo-frontend-service*，版本号20161111

- 针对 方式一：编写 Dockerfile，向容器内复制本地文件，或执行命令

```
docker build -t demo-frontend-service:20161111 -f /tmp/Dockerfile .
docker tag demo-frontend-service:20161111 registry.cn-hangzhou.aliyuncs.com/edas/demo-frontend-service:20161111
```

- 针对 方式二：本地 docker 命令，向容器内复制本地文件 / 方式三：进入 docker 容器，下载远程文件

```
export IMAGE_ID=`docker ps -a -f name=edas-build-package --format {{.ID}}` && docker commit $IMAGE_ID registry.cn-hangzhou.aliyuncs.com/edas/demo-frontend-service:20161111
```

5. 上传镜像仓库。

   执行如下命令，将打包本地镜像，推送到远程服务器，待部署。若未登录或提示无权限，请参考登录远程仓库 进行账号登录。

   以下方命令为示例，推送远程服务仓库 *registry.cn-hangzhou.aliyuncs.com/edas/demo-frontend-service*，版本号*20161111*。

```
docker push registry.cn-hangzhou.aliyuncs.com/edas/demo-frontend-service:20161111
```

21.6 应用管理

通过EDAS控制台，可以发布应用以及对应用进行一系列管理操作。

21.6.1 应用生命周期管理

应用发布以后，可以在 EDAS 控制台对发布的应用进行回滚、扩容缩容、删除等一系列管理操作。

21.6.1.1 概述

应用是 EDAS 管理的基本单位，由一组部署相同服务的机器组成。EDAS 提供了完整的应用生命周期管理机制，可以完成对一个应用从发布到运行过程的全面管理，包括应用的创建、部署、启动、回滚、扩容和停止下线等。

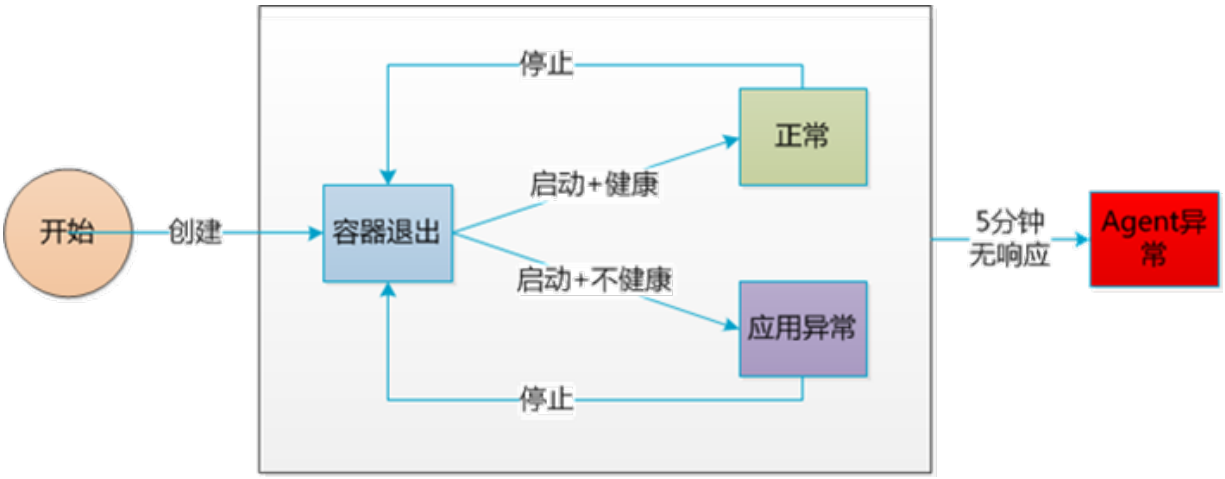
应用生命周期管理主要包含以下几个环节：

图 496: 应用生命周期示意



应用下的每台机器包含了**容器退出**、**正常**、**应用异常**、**Agent 异常** 4个状态。各状态转换如下图所示：

图 497: 状态转换示意



如果 EDAS Agent 长时间与 EDAS 控制台无心跳信息（例如：网络断开、Agent 退出、服务器升级），应用状态会变为 Agent 异常。

应用生命周期管理包含应用发布、管理、配置三大类功能：

- 应用发布包含：应用创建、启动、停止、部署。
- 应用管理包含：应用回滚、手动扩容、删除，某台机器重置、下线。
- 应用配置包含：容器、JVM参数、负载均衡、健康检查等配置。

说明:

- 应用运行或者停止时都可以进行应用部署、扩容、回滚、重置、应用配置等一系列的操作。
- Tomcat 容器、JVM 参数配置保存后仅改变相关配置文件，不会重启应用，需要应用重启后才会真正生效。

21.6.1.2 应用发布

本节主要介绍在 EDAS 控制台进行应用发布的过程，旨在引导您快速上手 EDAS 并成功发布应用。

说明: 如果发布 HSF 服务，请在启动应用前创建服务分组，否则会因为鉴权失败导致应用发布失败。请在 EDAS 控制台左侧菜单栏选择**服务市场** > **服务分组**，单击页面右上角**创建服务分组**进行创建。服务分组名称必须是全局唯一。创建好分组后需重新启动应用才能生效。

21.6.1.2.1 创建应用

创建应用分为创建普通应用和创建 Docker 应用。

前提条件

创建应用之前，请确保已经有成功安装了 EDAS Agent 的实例。

操作步骤

创建普通应用

1. 登录EDAS控制台，在左侧导航栏单击**应用管理**，进入应用列表页面。
2. 在**应用列表**页面右上角单击**创建应用**。
3. 在创建应用对话框输入相关信息，然后单击**下一步**。

图 498: 创建应用



表 74: 创建应用参数说明

参数	说明
应用运行环境	运行应用的容器（Ali-Tomcat）版本，请在下拉菜单中选择最新版本。**必选项。**
应用名	请尽量选择有含义且易识别的名称。**必选项。**
应用所在区域	请选择 ECS 所在的区域。**必选项。**
应用健康检查	EDAS定期访问该设置的URL，根据其响应状态，确定应用的存活状态。若不配置，EDAS不进行应用的健康检查，但不影响应用的正常运行。**可选项。**
备注	该应用的备注信息。**可选项。**

4. 选择应用类型（普通应用）、网络类型及具体实例，然后单击**创建应用**。

创建 Docker 应用

说明：Docker 应用创建时需要指定集群。创建应用之前，请确保该集群中已经有转化为 Docker 主机的 ECS。

5. 登录EDAS控制台，在左侧导航栏单击**应用管理**，进入应用列表页面。
6. 在**应用列表**页面右上角单击**创建应用**。
7. 在创建应用对话框输入相关信息，然后单击**下一步**。

图 499: 创建应用

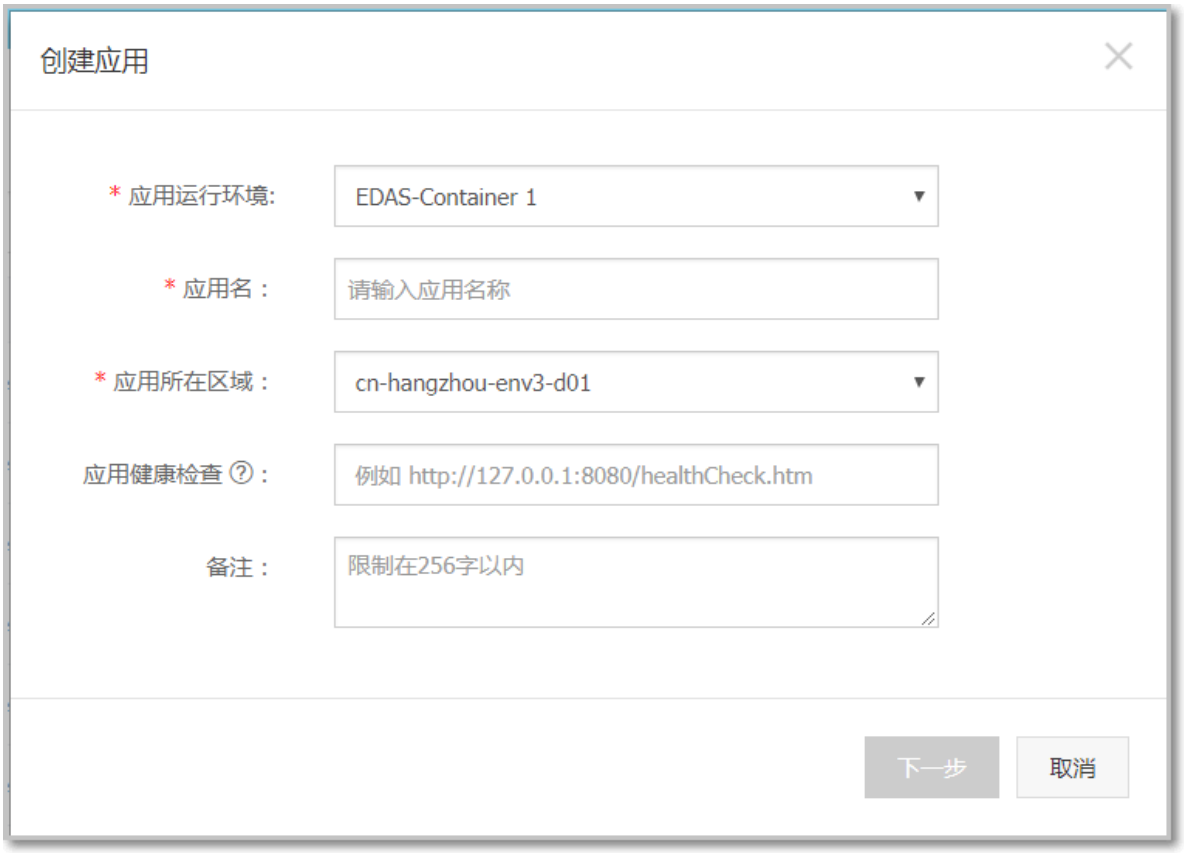


表 75: 创建应用参数说明

参数	说明
应用运行环境	运行应用的容器（Ali-Tomcat）版本，请在下拉菜单中选择最新版本。**必选项。**
应用名	请尽量选择有含义且易识别的名称。**必选项。**
应用所在区域	请选择 ECS 所在的区域。**必选项。**
应用健康检查	EDAS定期访问该设置的URL，根据其响应状态，确定应用的存活状态。若不配置，EDAS不进行应用的健康检查，但不影响应用的正常运行。**可选项。**
备注	该应用的备注信息。**可选项。**

8. 选择应用类型（Docker 应用）、网络类型及具体实例，然后单击**创建应用**。

图 500: 创建 Docker 应用



- 应用类型： 选择 Docker 应用。
- WEB 端口： Docker 应用采用 HOST 网络模式，在创建 Docker 应用时需要指定 Tomcat 的 WEB 端口。打开页面时系统会自动为您分配一个可用端口。
- 预分配端口： 为了避免端口冲突，当您的应用使用了除 Tomcat 之外的其他端口时，可以在这里指定这个端口为保留端口。
- 集群：应用部署的集群，创建应用后不可更改。应用的扩容缩容都在应用所属集群内完成。**一个应用不能同时部署在多个集群上。**
- Docker 配额： 采用 Docker 部署的应用，可以声明运行应用时所需要的资源实现资源隔离。可声明的资源包括 CPU 和内存，选择的资源不能超过集群中可用的 Docker 主机的可用资源。

- 实例个数： 创建 Docker 应用的实例个数。 **一个应用在一个 Docker 主机上只能创建一个实例。**
- 实例列表： 当您确定了应用的配额和实例个数后，系统会自动为您分配部署这些应用实例的 ECS 主机。

9. 设置完成后，单击 **创建应用。**

   应用成功创建后，页面右上角会出现**创建成功**的提示。

21.6.1.2.2 部署应用

创建应用后，就可以上传 WAR 包来进行应用部署。

操作步骤

1. 登录EDAS控制台，在左侧导航栏中单击应用管理，然后在应用列表页面单击具体应用的名称。
2. 在应用详情页面右上角单击 **部署应用。**
3. 设置部署应用参数。

   说明: 应用有普通应用和 Docker 应用两种类型，在部署应用页面中，参数有所不同，下面分别介绍。

- 部署普通应用

  图 501: 部署普通应用

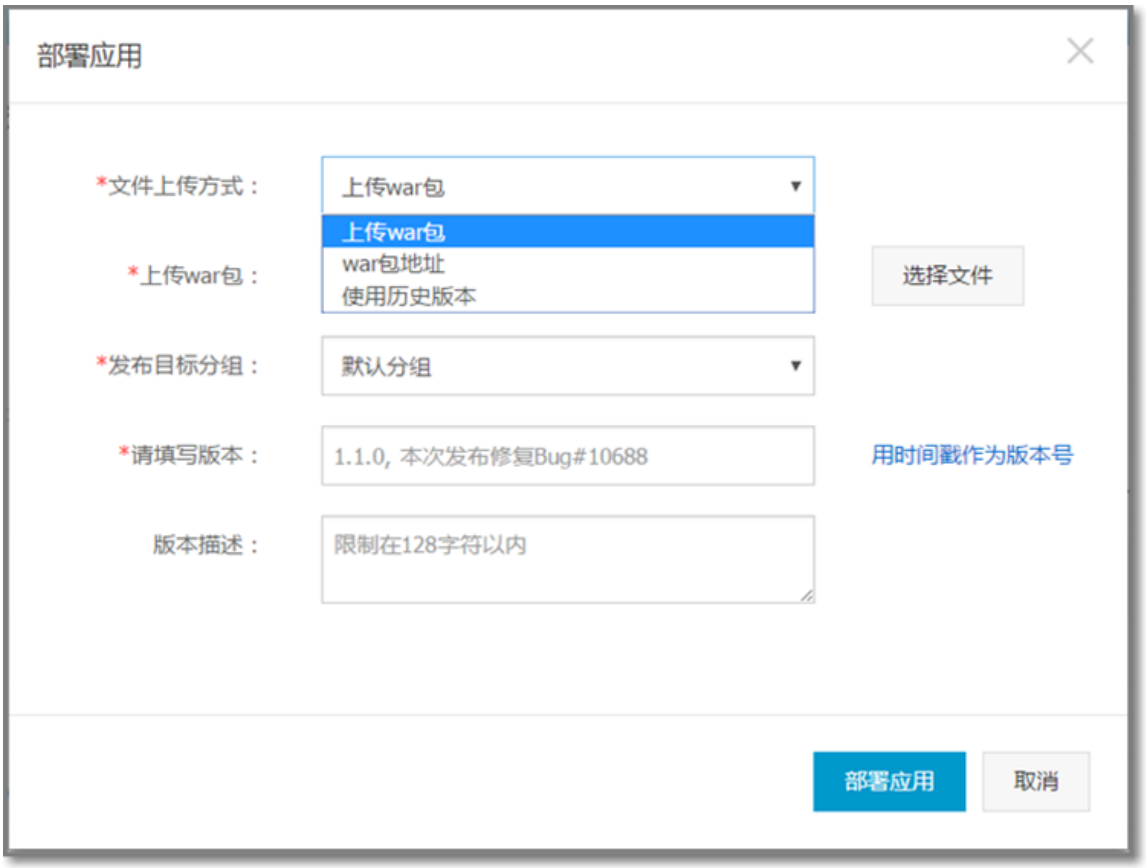


表 76: 部署应用参数说明

参数	说明
文件上传方式	• 上传 WAR 包：选择上传 WAR 包后，在下面上传 WAR 包右侧单击 **选择文件**，打开本地文件夹，选择要部署的 WAR 包。 如果是选择上传 WAR 包方式，可能需要等待片刻，等待 WAR 包上传，待进度条完成100%后，则文件上传成功。 • WAR 包地址：选择 WAR 包地址后，在下面 WAR 包地址右侧的文本框中输入定存放 WAR 包且可以访问的 URL 地址。 • 使用历史版本：选择使用历史版本后，在历史版本右侧的下拉框中选择具体历史版本。 **说明:** 如果选择使用历史版本，对话框会发生变化，无需再设置请填写版本和版本描述信息。

参数	说明
发布目标分组	需要发布此应用版本的分组。
请填写版本（适用于上传WAR包和WAR包地址两种文件上传方式）	填写WAR包的版本。
版本描述	对此WAR包版本进行描述。
历史版本（仅适用于使用历史版本的文件上传方式）	在下拉框中选择要使用的历史版本。

- 部署 Docker 应用

图 502: 部署 Docker 应用

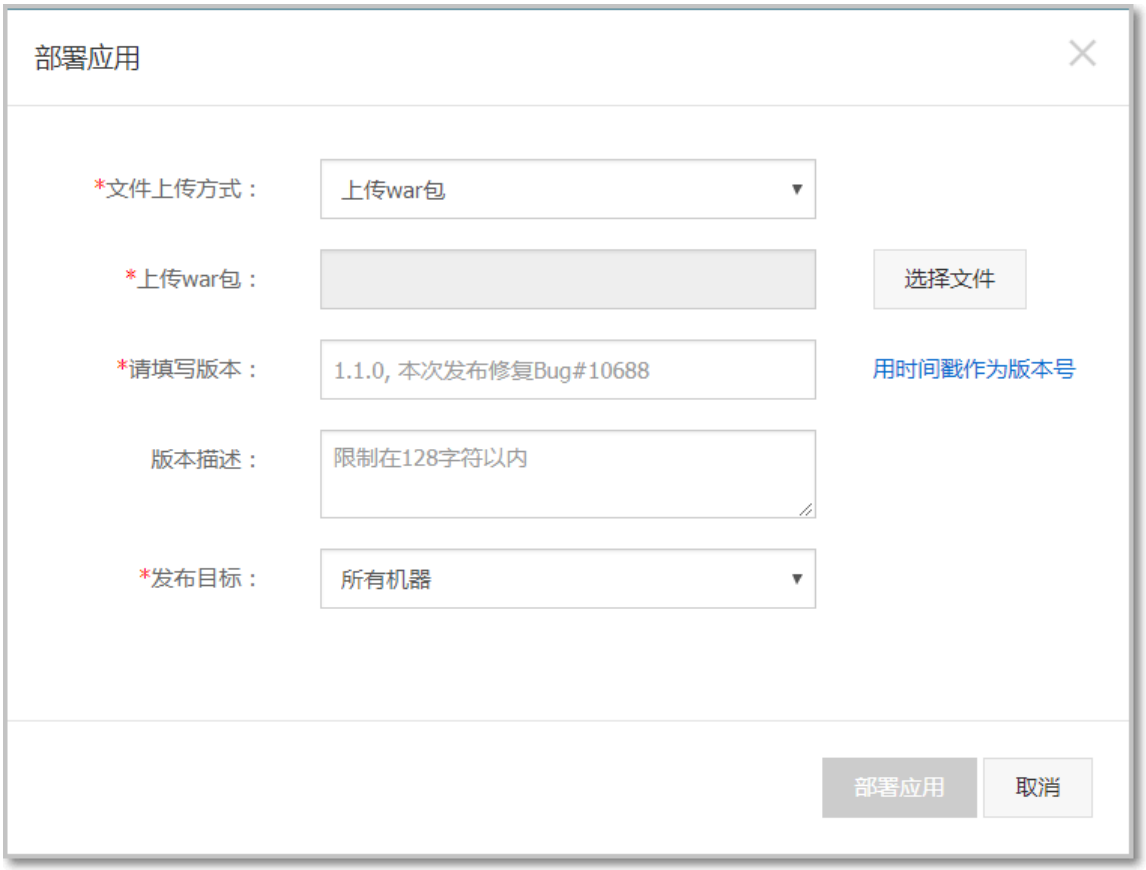


表 77: 部署应用参数说明

参数	说明
文件上传方式	• 上传 WAR 包：选择上传 WAR 包后，在下面上传 WAR 包右侧单击 **选择文件**，打开本地文件夹，选择要部署的 WAR 包 。 如果是选择上传 WAR 包方式，可能需要等待片刻，等待 WAR 包上传，待进度条完成100%后，则文件上传成功。 • WAR 包地址：选择 WAR 包地址后，在下面 WAR 包地址右侧的文本框中输入定存放 WAR 包且可以访问的 URL 地址。 • 使用历史版本：选择使用历史版本后，在历史版本右侧的下拉框中选择具体历史版本。 **说明:** 如果选择使用历史版本，对话框会发生变化，无需再设置请填写版本和版本描述信息。
请填写版本（适用于上传WAR包和WAR包地址两种文件上传方式）	填写WAR包的版本。
版本描述	对此WAR包版本进行描述。
发布目标	需要发布此应用版本的分组或所有机器。
历史版本（仅适用于使用历史版本的文件上传方式）	在下拉框中选择要使用的历史版本。

4. 设置完成后，单击 **部署应用**。

文件推送成功后，页面右上角出现推送成功或者开始推送告知应用开始部署。

21.6.1.2.3 启动应用

应用部署完成后，需要启动应用。

操作步骤

1. 在EDAS控制台左侧的导航栏中，单击 **应用管理**。
2. 在应用列表中，单击部署成功、需要发布的应用。
3. 在应用详情页面的右上角，单击 **启动应用**。

应用启动后，会有成功的提示，且应用实例的任务状态显示为**运行中**。

21.6.1.3 回滚应用

应用发布之后，如果有回滚到之前发布的版本的需求，可以使用“回滚应用”的功能，选择想要回滚的版本，就可以将应用回滚至该版本。

操作步骤

1. 登录 EDAS 控制台，在左侧菜单栏选择**应用管理**。
2. 在应用列表页面，单击要操作的应用名字进入应用详情页面，单击右上角的**回滚应用**。
3. 在**回滚应用**页面参考发布的 WAR 包名称和发布时间，选择需要回滚的版本，单击**回滚应用**。



说明: 只有部署了 Beta 机器，才会出现回滚目标的选择。如果没有部署 Beta 机器，则默认是回滚应用下的全部机器。**最多只能看到5 个回滚版本。**

21.6.1.4 手动扩容

当应用中机器负载过高，需要扩容机器分担压力，就可以使用应用扩容功能进行手动扩容。

操作步骤

1. 在应用详情页面的右上角单击**应用扩容**。
2. 在**扩容**页面，选择目标分组、实例个数，以及扩容的目标实例，然后单击**扩容**。



说明: 扩容进来的机器的运行状态是根据所在应用的运行状态来决定的，即如果扩容的时候应用处于运行中，那么扩容进来的机器会自动部署应用，自动启动运行应用；如果扩容的时候应用处于停止状态，那么扩容进来的机器就会部署应用，不会启动运行应用，而是处于停止状态。

21.6.1.5 手动缩容

当应用不需要太多资源的时候， EDAS 也可以通过停止并下线某个实例，实现应用手动缩容。

操作步骤

1. 在应用详情页面单击**实例部署信息**。
2. 下线不再需要的实例。
   - 当实例的任务状态为**运行中**，先单击**停止**，待任务状态变为**停止**，再单击**下线**。

图 503: 停止实例



- 当实例的任务状态为**停止**，直接单击**下线**即可将这台机器从该应用中删除掉。

图 504: 下线实例



21.6.1.6 删除应用

应用删除表示删除这个应用相关的所有信息，释放该应用下的所有机器，并将机器上的应用 WAR 包、容器文件全部删除。

前提条件

在删除应用前，请确保应用下的所有实例上的日志、WAR 包、配置等都已另外保存。

操作步骤

1. 在应用详情页面单击**停止应用**。
2. 单击**删除应用**进行删除。

   删除成功后，页面右上角会出现删除成功的提示。

21.6.2 应用设置

应用设置用于设置应用的JVM参数、Tomcat、负载均衡及健康检查。

21.6.2.1 配置JVM参数

如果需要在启动应用的时候配置容器的参数，可以通过**配置JVM参数**这个功能实现。正确调整这些参数有助于降低垃圾回收开销，从而缩短服务器响应时间并提高吞吐量。如果没有设置容器参数，那么默认由JVM分配。

操作步骤

1. 登录 EDAS 控制台，在左侧菜单栏单击**应用管理**，单击需要设置的应用名称进入应用详情页。
2. 在应用详情页面**应用设置**栏右侧单击**设置**。

图 505: 应用设置



3. 在应用设置对话框的 JVM 参数页签设置相应的参数。

表 78: JVM参数说明

参数	说明
初始 Heap Size	对应参数 -Xms
最大 Heap Size	对应参数 -Xmx
最大持久代大小	对应参数 -XX:MaxPermSize
自定义	对应 -D 参数，用来在启动一个 Java 程序时设置系统属性值。

图 506: 应用设置-JVM参数页签



4. 设置完成后，单击**配置 JVM 参数**，页面右上角会提示**配置 JVM 成功**。

> **说明:** EDAS JVM 参数配置会被写到容器目录下的 *bin/setenv.sh* 文件中，需要重启应用才能使参数配置生效。

21.6.2.2 配置Tomcat

EDAS 提供 Tomcat 容器参数配置功能，您可以在 EDAS 控制台修改容器（Tomcat）的端口、应用访问路径以及连接池的连接数。

前提条件

> **说明:**
>
> - 配置 Tomcat 容器参数后，只有重启容器才能生效。
> - 容器配置只在 EDAS Agent 2.8.0 版本及以上才有效。

操作步骤

1. 登录 EDAS 控制台，在左侧菜单栏选择应用管理，单击需要设置的应用名称进入应用详情页。
2. 在应用详情页面**应用设置**栏右侧单击**设置**。

3. 在**应用设置**对话框单击 **Tomcat** 页签，设置 Tomcat 参数。

图 507: 应用设置-Tomcat页签



表 79: Tomcat参数说明

参数	说明
Tomcat 端口	端口配置限定是(1024，65535)，由于 1024 以下的端口需要 Root 权限才能操作，而容器配置的权限为 Admin。所以请填写 1024 以上的端口。如果不配置，则默认是 8080。
Tomcat Context	选择应用的访问路径。 • 选择 WAR **包名字**，则自定义路径不需填写，应用的访问路径是 WAR 包名称。 • 选择**根目录**，自定义路径也不需填写，应用的访问路径是/。 • 选择**自定义**，则需要在下面的自定义路径中填写自定义的一个路径，访问路径就是自定义的路径。如果不配置此项，则默认是 WAR 包名字为路径。

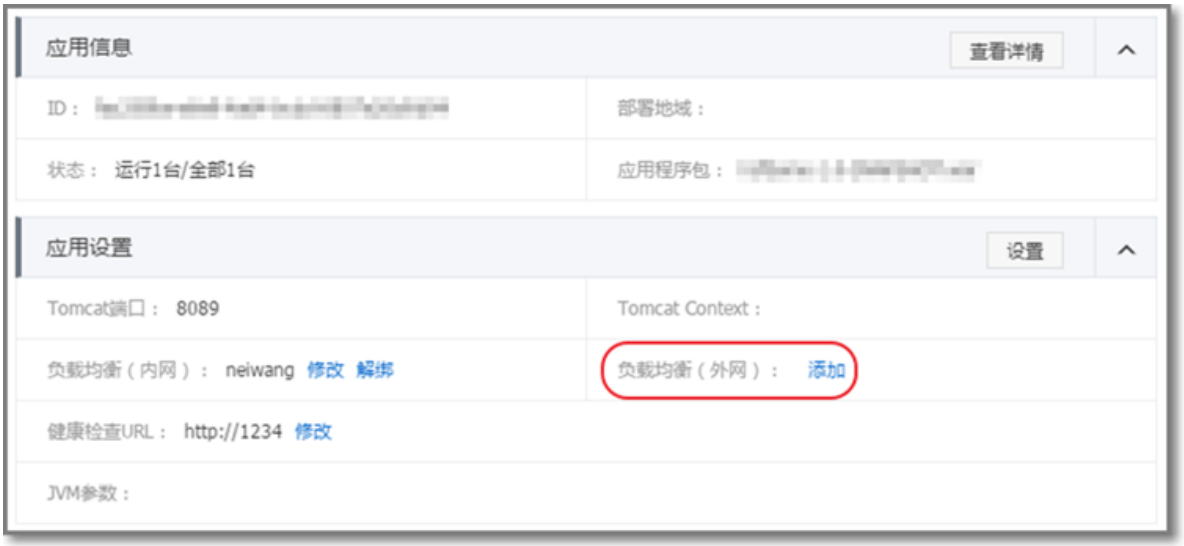
参数	说明
最大线程数	配置连接池的连接数大小，对应参数 maxThreads，不填默认是 400。此项配置对应用性能有很大影响，建议由专业人士配置。

21.6.2.3 配置负载均衡

如果您开通了阿里云负载均衡服务，EDAS 将会将负载均衡实例同步到 EDAS 控制台，为您提供负载均衡配置的功能。

操作步骤

1. 在应用详情页面，单击**负载均衡**（**外网**）右侧的**添加**。



2. 根据 IP 和名称，选择需要绑定的负载均衡服务器（SLB）。
3. 单击**绑定**。页面右上角出现**绑定成功**，即此次操作成功。

后续操作

绑定 SLB 之后，也可以通过**修改**和**解绑**来更换 SLB 和取消绑定 SLB。

21.6.2.4 设置健康检查

EDAS 提供健康检查功能，可以用于查看部署的应用是否正常运行。您可以通过设置健康检查 URL 实现此功能。

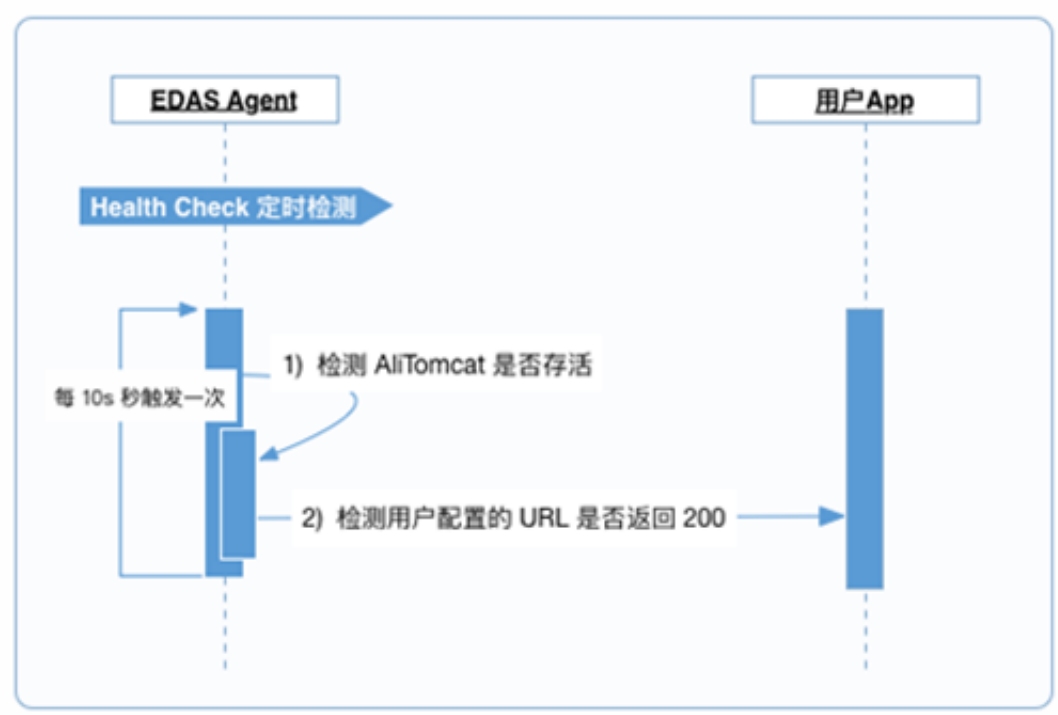
背景信息

EDAS 中的健康检查，是指由 EDAS Agent （以下简称为 Agent）针对容器与应用进行定时检测与汇报，然后将结果反馈在控制台上的过程。健康检查能帮助您在集群环境下了解整个服务的运行状

态，从而为一些常见的问题审查与定位提供帮助。EDAS 提供配置健康检查 URL，来检查部署的应用是否在正常运行。

健康检查机制中 Agent 的检测流程图如下：

图 508: Agent检测流程图



上图的流程为每隔十秒触发一次。图中的 1）和 2）具体说明如下：

1. 检测 AliTomcat 是否存活：如果运行用户 APP 指定的 AliTomcat 进程存活，则认为检测通过；如果该进程没有存活，则不会发生第 2 步检测。
2. 检测配置的 URL 是否返回200：如果用户没有配置任何的 URL，将不会发生检测；否则将检测配置的 URL 是否返回 HTTP 200。

在不配置健康检查的情况下，使用 EDAS Agent 2.8.0 以上版本的容器会自动分配一个健康检查路径 *http://127.0.0.1:8080/WAR包名称/_ehc.html*。 如果需要手动配置健康检查链接，请按以下步骤操作：

操作步骤

1. 登录 EDAS 控制台，单击左侧菜单栏单击**应用管理**。
2. 在应用列表页面单击需要查看的应用名称进入应用详情页。
3. 在应用详情页面，点击**健康检查 URL** 右侧的**修改**。

4. 填写正确的健康检查 URL 并单击**保存**。页面右上角会提示**修改健康检查URL 成功**。

   如果做了相应的容器配置，请根据配置信息设置健康检查链接 *http://127.0.0.1:*自定义端口/配置的路径/*_ehc.html* 。 “_ehc.html” 是默认路径，如有需要可以更改为自定义路径。请确保健康检查 URL 是应用内一个可被访问的路径，且该路径必须返回 200-500 HTTP 代码。

   【例】如果 WAR 包名称是 order.war，在没有配置其他容器设置的前提下，则健康检查的 URL 可以设置为*http://127.0.0.1:8080/order/_ehc.html*；若配置了容器路径为根路径，端口设置为 “8081” ，WAR 包中包含一个可以标识健康状态的文件是 “healthcheck.html” ，则健康检查URL为*http://127.0.0.1:8081/healthchec k.html* 。

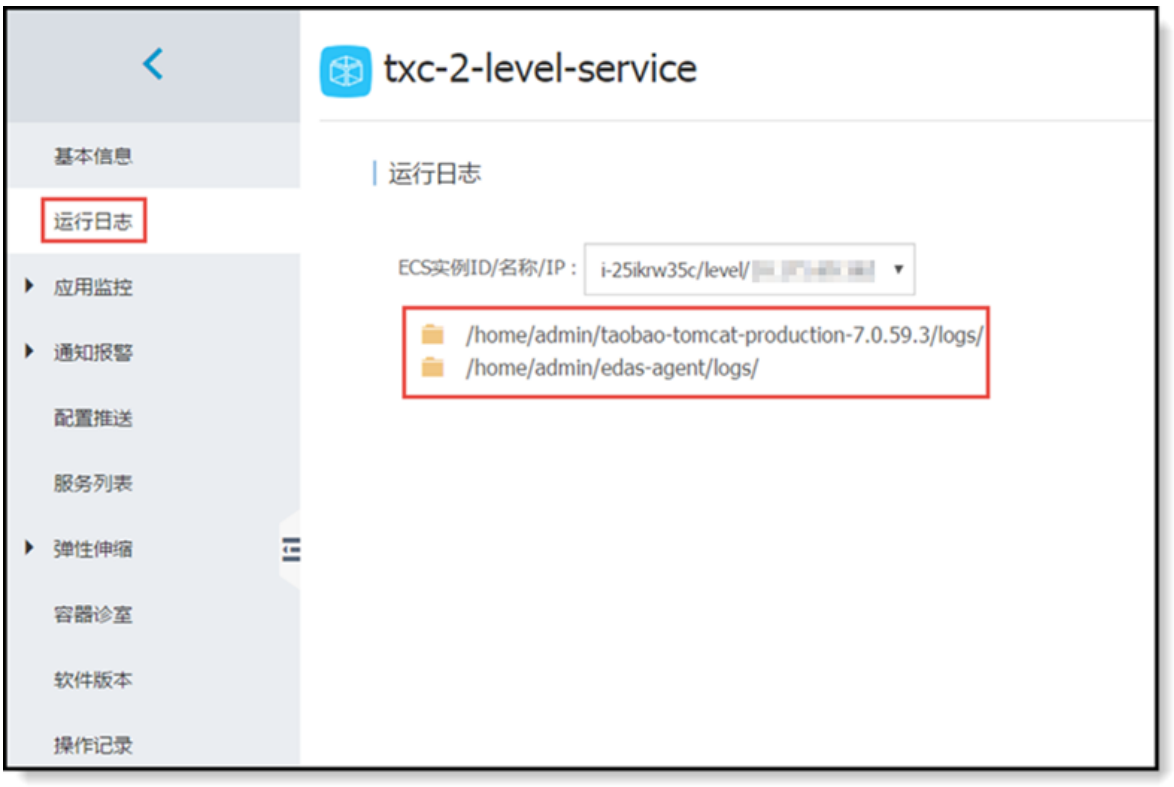
21.6.3 运行日志

EDAS 控制台提供日志浏览功能，无需登录服务器就可以查看服务器上所部属的应用运行日志。当应用出现异常情况的时候，可以通过查看日志来排查问题。

操作步骤

1. 在 EDAS 控制台左侧菜单栏单击**应用管理**，在**应用列表**页面中单击需要查看的应用。
2. 在应用详情页面左侧菜单栏中单击**运行日志**或者选择**实例部署信息**下的**日志**。

图 509: 运行日志页面



运行日志页面默认包含两个日志路径：Tomcat 容器的 logs 目录和 EDAS Agent 的 logs 目录。

Tomcat 容器日志的具体路径会根据实际使用的版本而定。

3. 单击 **ECS实例ID/名称/IP** 右侧的下拉箭头，在下拉菜单中选择该应用下的某个ECS实例的ID、名称或者IP。
4. 单击日志文件夹或者路径即可展开该文件夹包含的日志文件。

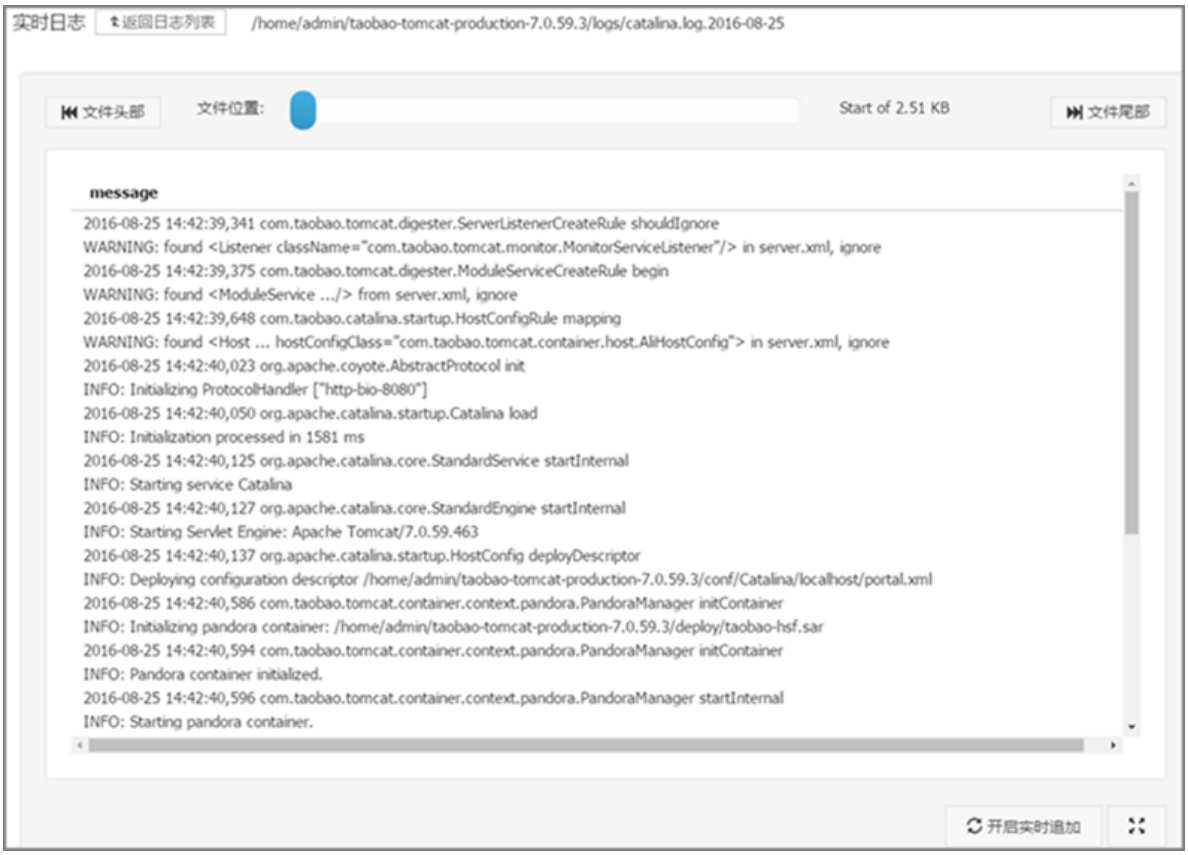
图 510: 运行日志展开页面



说明: 展示的文件目录中，只有可读的文件，不会展示文件夹。

5. 双击想要查看的日志文件，查看具体日志信息。

图 511: 日志详情页面



单击页面右下角的**开启实时追加**，可以一直加载文件的最新追加内容（类似于 tailf 命令的效果）。

6. （可选）收藏路径。

   a) 在运行日志页面右上角，单击**收藏路径**。

   b) 在添加应用日志路径对话框中输入**应用日志路径**。

   注意:

   - 此路径必须在 /home/admin 目录下，而且完整路径中必须包含“log”或者“logs”。
   - 文件最后必须以斜杠“/”结尾，表示添加的是一个文件夹。

   c) 单击**添加**。

如果要取消收藏，单击选中文件目录中的一个**文件夹**名称，单击页面右上角**取消收藏选中路径**。

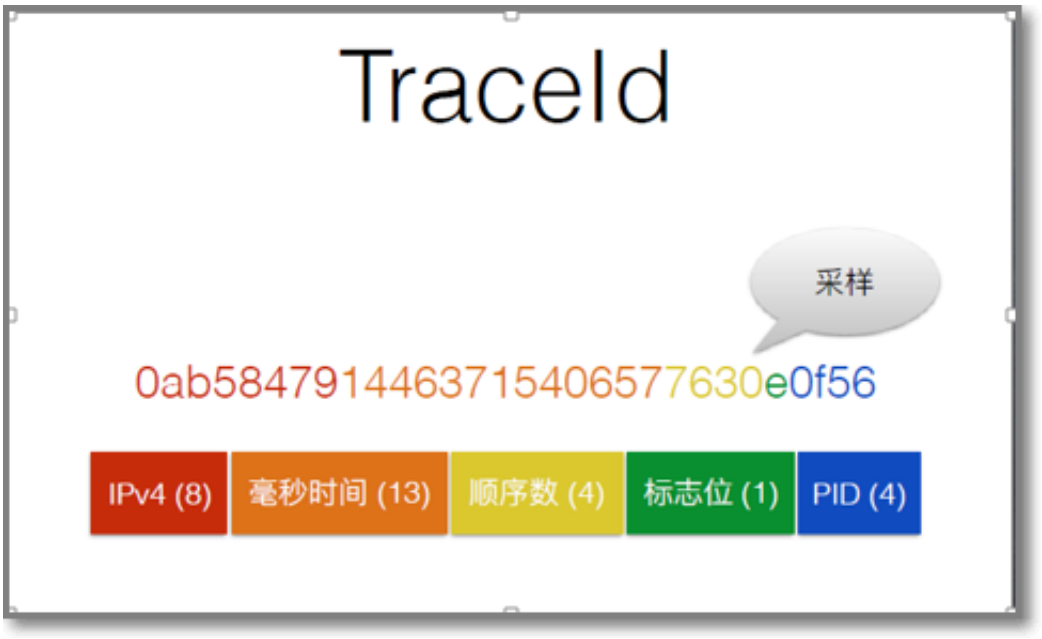
该路径将不再显示在日志查看页面。此操作不会删除或改变服务器上的任何文件。

21.6.4 应用监控

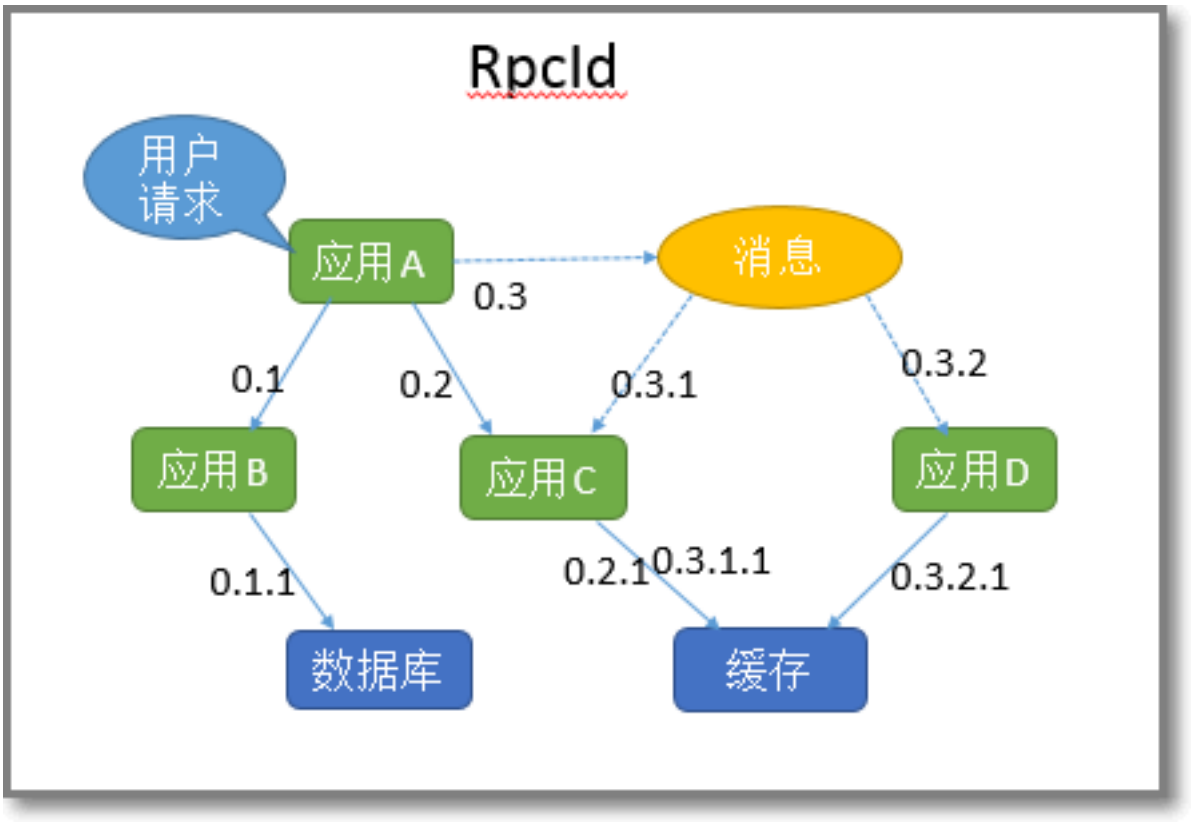
应用监控可以准确的反映应用的实时流量情况和历史信息，用户可以通过该信息监控应用的健康状态，并且可以快速发现、定位问题。

名词解释

- TraceId: 与一次请求相对应，全局唯一，在各个系统间传递。



  - IP 地址：创建 TraceId 的机器的 IP（十六进制）。
  - 创建时间：调用链别创建的时间。
  - 顺序数：用于链路采样。
  - 标志位：可选，用于调试和标记。
  - 进程号：可选，单机多进程的应用使用。
- RpcId: 调用标识日志埋点顺序和嵌套关系，也在各个系统间传递。



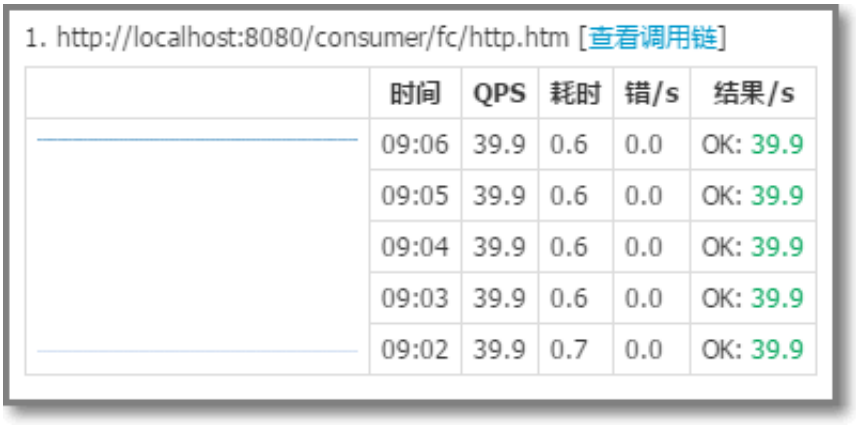
- 服务维度: 服务监控的监控数据分为应用维度和服务维度，应用维度的数据按照应用聚合，而服务维度的数据按照用户自定义的服务聚合，例如用户有一个应用 A，这个 A 提供了几个服务a,b,c。
- 下钻应用: 查看目标指标关联的上（下）游应用的相关指标情况。

监控数据类型

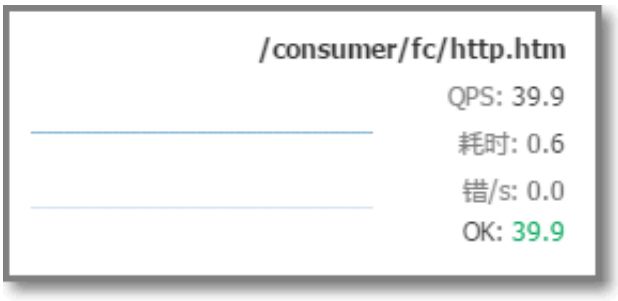
- **应用概要**：包含应用提供服务和调用服务的整体情况。
- **我的入口**：应用提供入口的情况（入口即为一次前端请求，大部分情况下是基于 HTTP 协议）。
- **提供的 RPC 服务**：应用作为服务端提供 RPC 服务的情况（包含 HSF 类型和其他自定义类型）。
- **RPC 调用来源**：应用作为服务端被客户端调用的情况。
- **RPC 调用依赖**：应用作为客户端调用 RPC 服务的情况（包含 HSF 类型和其他自定义类型）。
- **访问数据库**：应用作为客户端访问数据库的情况。
- **消息类型**：应用生成、消费消息的情况。

监控报表类型

- **豆腐块（默认）**：豆腐块类型报表以“表格+图”的形式展现数据，包含的信息有监控对象、时间、QPS、耗时、服务端耗时、错和结果，其中图中展现信息默认为一小时的数据，表格展现的数据默认为五分钟的数据；



- **多图**：多图类型报表以“图”的形式展现数据，包含的信息有监控对象、时间、QPS、耗时、错和结果，其中图中展现信息默认为一小时的数据，并会单独列出最近时刻的数据；



- **表格**：表格类型报表以“表格”的形式展现数据，包含的信息有监控对象、QPS、耗时、错和结果，其展示的信息为最近一分钟的数据；



指标说明

- **时间**：以分钟为维度，展示当前分钟内的数据，如08:00指的是8点0分0秒至8点0分59秒的数据。
- **QPS**：某一分钟的每秒访问量的平均值，计算的方式为：QPS=该分钟内的总访问量/60。

- **耗时**：客户端记录的一分钟内的平均访问耗时，单位为毫秒，计算方式为耗时=该分钟内所有访问的总耗时/总访问量。
- **服务端耗时**：服务端记录的一分钟内的平均耗时，单位为毫秒，计算方式与“耗时”一致。
- **错**：记录一分钟内 RPC 调用的出错速率，计算方式为为错=该分钟内总错误数/60。
- **结果**：记录返回结果，形式以“结果：QPS”展示，其中“结果”表示的是 RPC 的结果，HTTP 的结果与 HTTP 协议的 ErrorCode 一致。

21.6.4.1 监控大盘

监控大盘基于不同分组，以图表的形式展示整体服务提供、服务消费和基础监控相关项目。

背景信息

- 服务提供：包括 RPC 服务和 HTTP 服务。
- 服务消费：数据库访问。
- 基础监控：包括 CPU、负载、内存、磁盘及网络。

操作步骤

1. 登录 EDAS 控制台。
2. 在左侧导航栏，单击 **应用管理**，在应用列表中，单击需要监控的 **应用名称**。
3. 在应用详情页面左侧导航栏中，选择**应用监控** > **监控大盘** 。

图 512: 监控大盘



在该页面中，可以查看服务提供、服务消费和基础监控的具体监控图。

- 将鼠标悬停到监控图- 横坐标的某个点，可以查看该时间点的信息和状态数据。

- 单击监控图上方的项目名称，如 RPC 服务，可以跳转到服务监控的页签中，查看详细数据。监控参数的含义，请参见 应用监控。

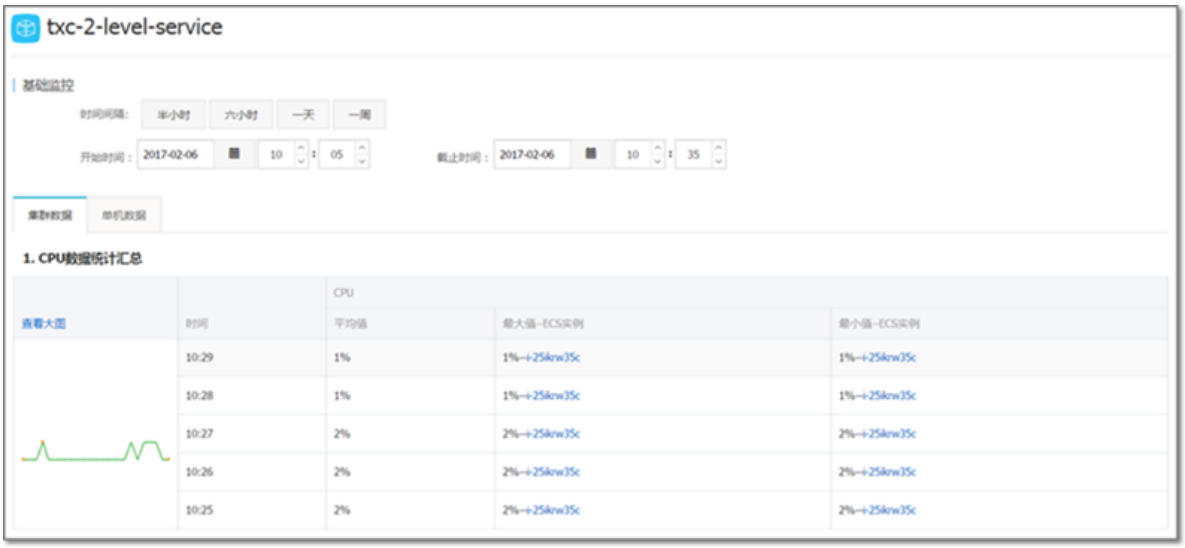
21.6.4.2 基础监控

EDAS 从应用所运行的机器上采集数据，然后基于分析的结果提供 CPU、内存、负载、网络、磁盘五个主要指标的单机与集群视图。由于从数据的采集到分析存在一定的延时，所以 EDAS 无法提供百分之百的实时监控视图，目前的时延策略为两分钟。所有监控均以应用为单位进行数据的统计和处理。

操作步骤

1. 登录EDAS 控制台，在左侧导航栏中单击**应用管理**，在应用列表中单击具体的应用。
2. 在应用详情页面左侧的导航栏中选择**应用监控** > **基础监控**，进入基础监控页面。

图 513: 基础监控页面



基础监控页面默认监控最近半小时的集群数据。也可以通过选择**时间间隔**及**单机数据**页签监控其他时间间隔的集群数据和单击数据。

3. 选择监控数据类型。

监控的数据类型包括集群数据和单机数据两种。

这两种数据类型从不同维度监控相同的一些列指标，包括：

1. CPU 数据统计汇总：代表 CPU 的占用率，为 user 和 sys 占用率的和，其中集群数据的图示为该应用集群内所有机器占用率的平均值。

2. 内存数据统计汇总：物理内存的总大小与实际使用大小，其中集群数据的图示为该应用集群内所有机器的总大小与总使用大小。
3. 负载数据统计汇总：系统负载中的 1 min load 字段，其中集群数据中的图示为该应用集群内所有机器 1 min load 的平均值。
4. 网络速度数据统计汇总： 网卡的写出与读入速度，如果机器上有多块网卡，该数据代表所有网卡名以“eth”开头的写出与读入速度的总和，其中集群数据中的图示为该应用集群内所有机器的平均值。
5. 磁盘数据统计汇总：机器上所有挂载磁盘的总大小与实际使用大小，其中集群数据的图示为该应用集群内所有机器的总大小与总使用大小。
6. 磁盘读写速度数据统计汇总：机器上所有挂载磁盘的读写速度大小总和，其中集群数据的图示为应用集群内所有机器改数据的平均值。
7. 磁盘读写次数数据统计汇总：机器上所有挂载磁盘的读写次数（IOPS）总和，其中集群数据的图示为应用集群内所有机器改数据的平均值。

4. 设置时间间隔。

   时间间隔包括半小时、六小时、一天和一周四种。

   - **半小时**：统计当前时间之前半个小时内的监控数据，是登录基础监控页面时的默认方式。数据统计点的时间间隔为1分钟，是 EDAS 所提供的最细的查询粒度。
   - **六小时**：统计当前时间之前 6 个小时内的监控数据，数据统计点的时间间隔为5分钟。
   - **一天**：统计当前时间之前 24 个小时内的监控数据，数据统计点的时间间隔为15分钟。
   - **一周**：统计当前时间之前 7 天内的监控数据，数据统计点的时间间隔为1小时，是 EDAS 提供的最长统计周期。

   说明:

   页面中的**开始时**间与**结束时间**，是当前视图所呈现的时间跨度；当选择其中一项时，对应的项会自动调整其值，如：选择**半小时**，结束时间选择 **2016年5月20日 12点00分00秒**时，开始时间会自动设置为**2016年5月20日 11点30分00秒**。

   设置完成后，监控的数据随选择的时间间隔自动进行更新。

5. （可选）查看具体指标的监控大图。

   在监控时，还可以单击具体指标下方的**查看大图**来查看具体指标的监控大图，并可以在大图中设置时间间隔。

   图 514: 监控大图



21.6.4.3 服务监控

通过收集和分析在不同的网络调用中间件上的日志埋点，可以得到同一次请求上的各个系统的调用链关系，有助于梳理应用的请求入口与服务的调用来源、依赖关系，同时，也对分析系统调用瓶颈、估算链路容量、快速定位异常有很大帮助。

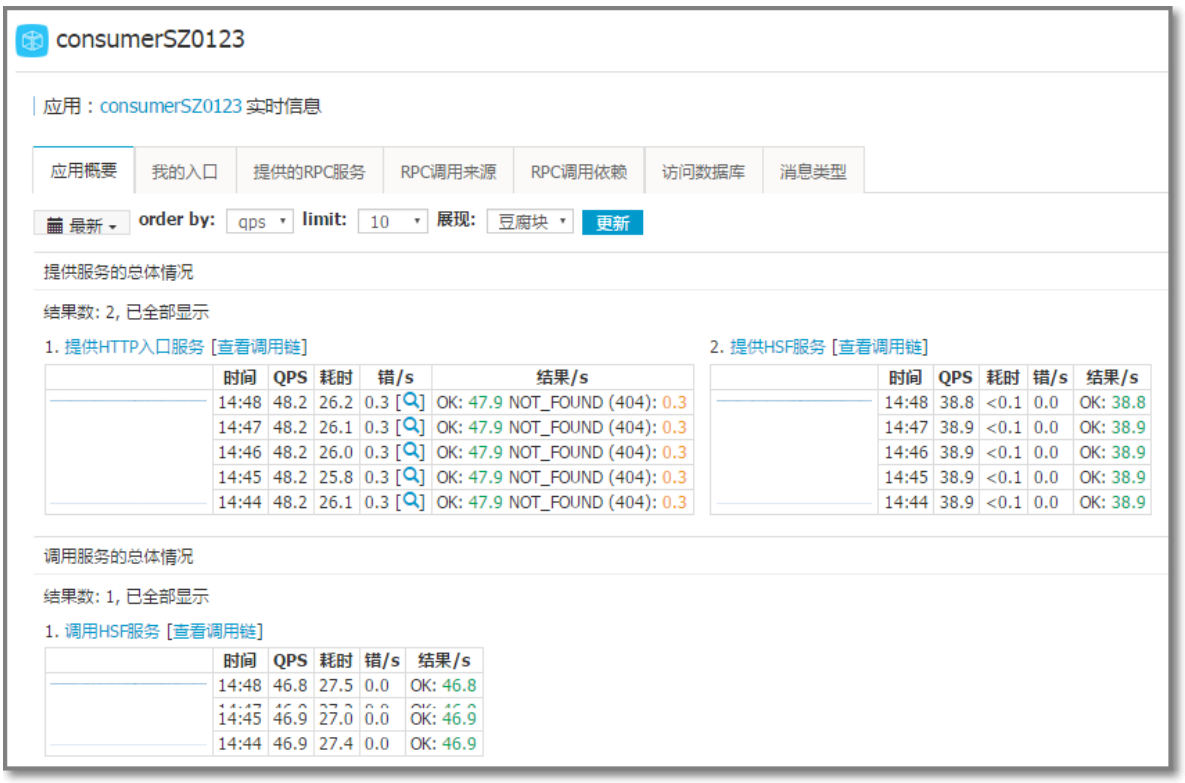
操作步骤

监控服务

1. 登录 EDAS 控制台。
2. 在左侧导航栏中，单击 **应用管理**。

3. 在应用列表中，单击要监控的 **应用名称**。
4. 在具体应用页面左侧的导航栏中选择**应用监控** > **服务监控** 。

图 515: 服务监控



服务监控页面包含七个页签：

- 应用概要：展示所提供服务的总体情况和调用的服务情况。
- 我的入口：展示对外提供的 HTTP 入口列表。
- 提供的 RPC 服务：展示当前应用所提供的 RPC 的调用情况。
- RPC 调用来源：展示当前应用所提供的 RPC 服务被哪些应用调用。
- RPC 调用依赖：展示当前应用调用了哪些应用的服务。
- 访问数据库：展示对数据库的请求情况。
- 消息类型：MQ 消息等。

默认显示默认分组下的应用概要页签。

5. （可选）设置监控条件，单击 **更新**，更新监控数据。
   - **最新**：默认展示当前时间的数据情况，点击下拉箭头，可以手工选择时间。

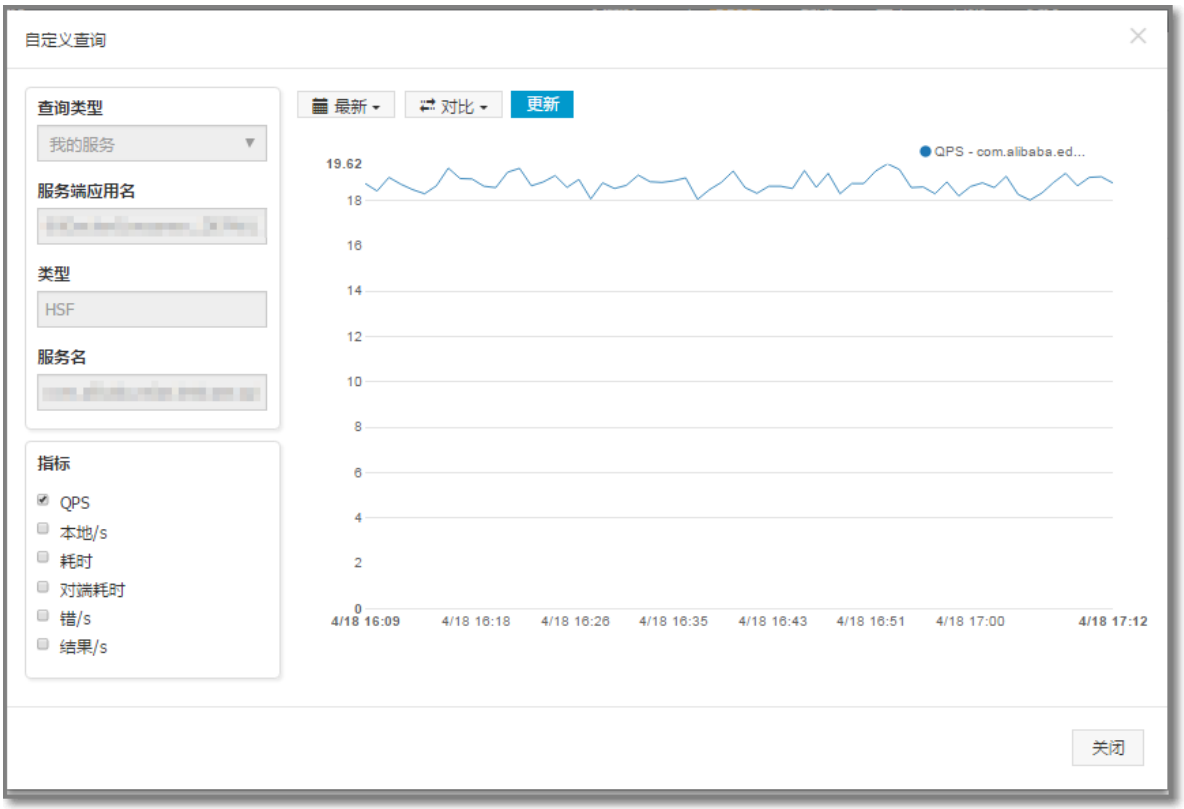
- **order by**：默认以 QPS 排序，点击下拉箭头，可以按耗时、错/S(一分钟平均出错的 QPS)进行排序。
- **结果数**：默认显示10条结果，点击下拉箭头，可以设置展示数量，包括1、5、30、50、100和不限。
- **展现**：默认以豆腐块图展现，也可以选择支持多图和表格作为展现形式。

6. 查看监控数据。

   监控指标的说明，请参见 应用监控。

7. 单击监控图中某一列的具体指标，弹出自定义查询页面，查看该指标项的监控数据。

图 516: 指标监控



8. 在指标区域中，选择不同指标，也可以监控不同组合的数据。

查看调用链

在监控服务过程中，可以监控该应用和其它应用的附件间的调用关系，同时也可以查看详细的调用链路。

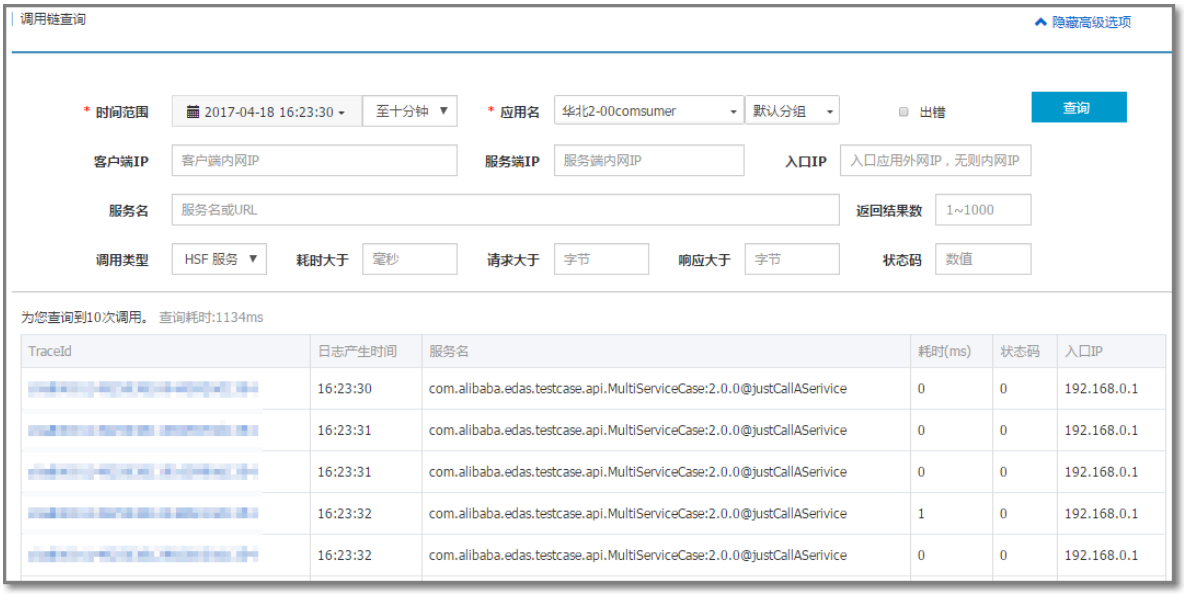
9. 在监控图中，单击调用的或被调用的服务右侧的 **查看调用链**，跳转到**链路分析** > **调用链查询**页面。

图 517: 调用链查询



10. 在调用链查询页面，可以查看该应用和（被）调用的服务间的调用链路。

图 518: 查看调用链路



监控下钻应用

服务监控页面除了可以查询和该应用相关的调用链路，还可以下钻到相互依赖的应用上进行监控。

11. 在 **提供的 RPC 服务**、**RPC 调用来源** 或 **RPC 调用依赖** 等页签中，单击监控图上方 **下钻** 右侧的 **来源应用**、**被调用的服务** 或 **调用的服务** 按钮，弹出下钻应用的监控页面。

12.监控下钻应用的数据。

监控方法和监控本应用的方法一致。

21.6.5 通知报警

EDAS 提供报警功能，当某些资源使用过度时，可以利用此功能提醒开发者及时处理线上问题。EDAS 中的报警是根据您配置的一些规则，基于后台采集到的数据进行分析和判断之后，通过短信与邮件的方式通知给相应的联系人。

背景信息

通知报警主要包括3部分：配置报警规则、设置报警联系人和查看报警记录。

说明: 目前 EDAS 中的报警方式只有短信与邮件通知的方式，暂不支持自行定制其他方式。

操作步骤

配置报警规则。

1. 登录 EDAS 控制台，在左侧导航栏中单击 **应用管理**，在应用列表中单击具体的应用。
2. 在应用详情页面左侧的导航栏中选择 **通知报警** > **报警规则**。
3. 在规则列表页面的右上角单击**创建规则**。
4. 在创建规则页面，设置规则参数。

图 519: 设置报警规则参数



表 80: 报警规则参数说明

参数	说明
规则名称	代表此规则的可以易于理解的名字，由数字，字母或下划线组成。
监控对象	基于监控的指标（基础监控、HTTP、HSF、应用容器）与设定的阈值，创建可比较的规则，监控对象不能少于一个，可增加多条。
触发条件	可选择“任一指标”与“所有指标”： • 任一指标：只要有监控对象的监控项中的一项符合规则，则触发报警。 • 所有指标：需要监控对象的监控项中所有规则均符合，才产生报警。
统计周期	可选值有1分钟、5分钟、15分钟、30分钟、1小时。由于有的时候系统会出现短时间的抖动，最典型的例子是当服务启动时，CPU 会占用特别多，但是一般来说在两分钟之内就会恢复，此时如果报警其实是误报。针对这种情况，EDAS 允许您选择一个统计周期，意味着如果在该周期内一直**持续的符合某些规则**，才选择报警。例如：选择监控项为 CPU

参数	说明
	大于 30%，统计周期选择五分钟，那么只有系统的 CPU 占用**持续**五分钟超过30%时，EDAS 才认为这是一次异常行为。
重试几次后报警	可选值有 1、3、5；此字段与统计周期类似，指的是连续多少个周期符合监控对象的规则，才真正触发报警。

5. 设置完成后，单击**确定**。

创建好的规则即时生效。如需废弃该规则，在规则列表中可选择删除，同样即时生效。

设置报警联系人

注意:

- **报警联系人的来源**：**当前系统支持通知与该账户有过主子账户关系的联系人。具体情况如下**：
  - **被主账户绑定过子账户的其他阿里云账户**
  - **登录过 EDAS 的 RAM（阿里云角色访问控制台，以下简称** RAM **）子账号**
- **报警联系人联系方式(邮箱与手机号)**：**邮箱与联系人默认从阿里云统一获取，但出于保护隐私考虑，只有此用户登录的时候才会获取此信息。如果不想使用在阿里云注册的手机号接收报警信息，请在个人资料处修改。**

6. 登录 EDAS 控制台，在左侧导航栏中单击 **应用管理**，在应用列表中单击具体的应用。
7. 在应用详情页面左侧的导航栏中选择**通知报警** > **报警联系人** 。
8. 在报警联系人页面的右上角单击**添加报警联系人**。
9. 在联系人列表中勾选要添加的联系人，然后单击 **确定** 。

设置企业员工作为报警联系人

说明: 如果需要接收通知报警的企业员工从来没有使用过 EDAS，**假设当前的 EDAS 的主账户为** master@aliyun.com，**需要通知的账户为** employee@company.com，**请按以下步骤进行操作。**

10. 添加 RAM 子账户。

a) 用 master@aliyun.com 登录阿里云控制台，在菜单栏中选择**产品与服务** > **访问控制**。

b) 在RAM页面左侧菜单栏中单击 **用户管理** ，在用户管理页面右上角单击 **新建用户**，在创建用户对话框中设置用户（Employee）的参数。

图 520: 设置用户参数



11.用刚刚创建的子账号登录 EDAS，修改资料。

a) Employee 通过 RAM 提供的子账户链接登录 EDAS 控制台。

b) 在左侧菜单栏选择**账号管理** > **个人资料** ，填入自己的手机号与邮箱。

12.按照 **设置报警联系人** 描述的步骤将账户 master@aliyun.com 添加成为报警联系人。

查看报警记录

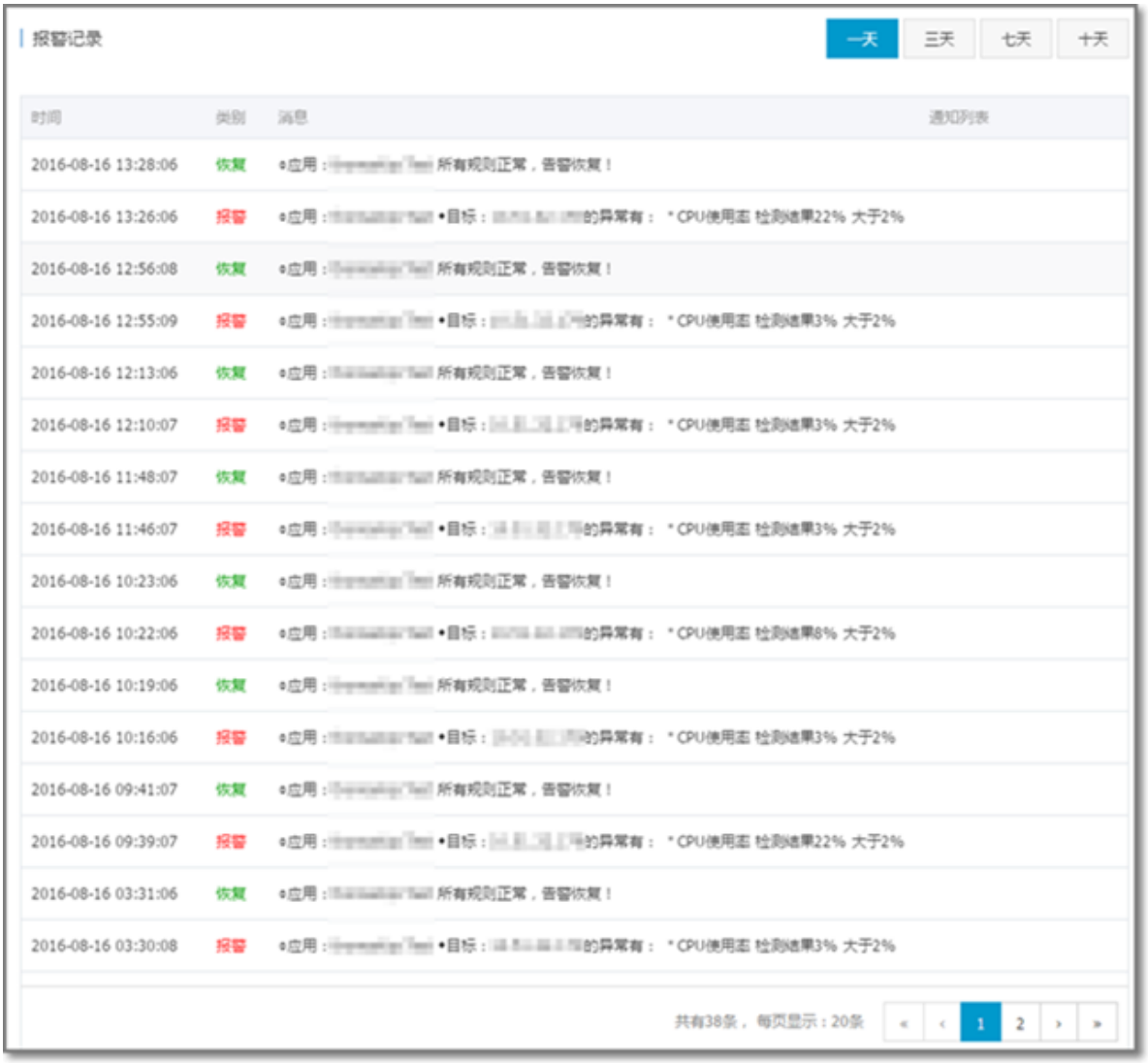
说明: 当有报警通知时，除了通知相应的报警联系人之外，系统也会产生纪录。

13.登录 EDAS 控制台，在左侧导航栏中单击 **应用管理**，在应用列表中单击具体的应用。

14.在应用详情页面左侧的导航栏中选择 **通知报警** > **报警记录**。

报警纪录提供最多十天之内的报警详细信息的查询，且当报警恢复时，也会有通知信息，并且此通知也会随邮件和短信告知。

图 521: 报警记录页面



21.6.6 弹性伸缩

EDAS 分布式集群管理中，很重要的一个运维能力是能敏锐的感知集群内各个服务器的状态，并根据状态实时的实现集群扩容、缩容，在保证服务质量的同时，提升集群系统的可用率。

背景信息

EDAS 提供弹性伸缩功能来根据集群内服务器的 CPU、RT 和 Load 三个指标实现自动的扩容或者缩容。

指标说明：

- CPU：服务器上 CPU 的使用率，以百分比表示，如果应用内有多个服务器，则是所有服务器的平均值。
- RT：系统对请求作出的响应时间，以 ms 为单位。
- Load：系统负载大小，以正整数表示。

以上指标均以正整数填写，不支持浮点数；当应用中有多台服务器时，以上指标均指应用中所有服务器的平均值。弹性伸缩包括自动扩容和自动缩容，可以分别单独配置扩容规则或者缩容规则。

操作步骤

1. 登录 EDAS 控制台。
2. 左侧导航栏中，单击 **应用管理**，在应用管理页面，单击具体的应用名称。
3. 在应用详情页面左侧的导航栏中，选择 **弹性伸缩** > **伸缩规则**。
4. 勾选 **扩容规则**，设置扩容规则参数。

图 522: 扩容规则



- 触发指标，包括 CPU、RT 和 Load。
- 触发条件
  - **任一指标**：表示设定的指标中任意一个指标被触发都会引起扩容。
  - **所有指标**：表示设定所有指标必须全部被触发才能引发自动扩容操作。

- 持续时间：指标持续被触发的时间，表示在持续时间内，指标的平均值达到设置的阈值，就会引起扩容操作，用户可根据集群服务能能力的灵敏度酌情配置。
- 每次扩容的实例数：此参数表示每次触发扩容操作后，自动增加的服务器个数，用户可根据应用的单个服务器的服务能力酌情配置。
- 最大实例数：表示当集群内服务器数量达到配置的最大实例数后，不再继续扩容，此配置请用户根据自己的资源限额配置。

自动缩容的配置方式和自动扩容类似，指标含义及设定方式请参考 **自动扩容**。

说明: 同时配置扩容和缩容规则的时候，缩容规则的指标不能大于扩容规则的指标，否则，在单击 **保存** 的时候会弹出错误提示：

设置了弹性伸缩规则后，如果发生了自动扩容或者自动缩容操作后，有两种方式可以查看弹性伸缩结果：

- 通过应用的 **基本信息** 中的 **实例部署信息** 页签来查看实例是否增加或者是否减少。
- 在 **弹性伸缩** 的 **历史记录** 中查看扩容、缩容的历史记录，如图所示：

图 523: 历史记录



21.6.7 限流降级

EDAS 中的限流降级主要用于解决后端核心服务因压力过大造成系统反应过慢或者崩溃问题，通常用于例如商品秒杀、抢购、大促、防刷单等大流量场景。

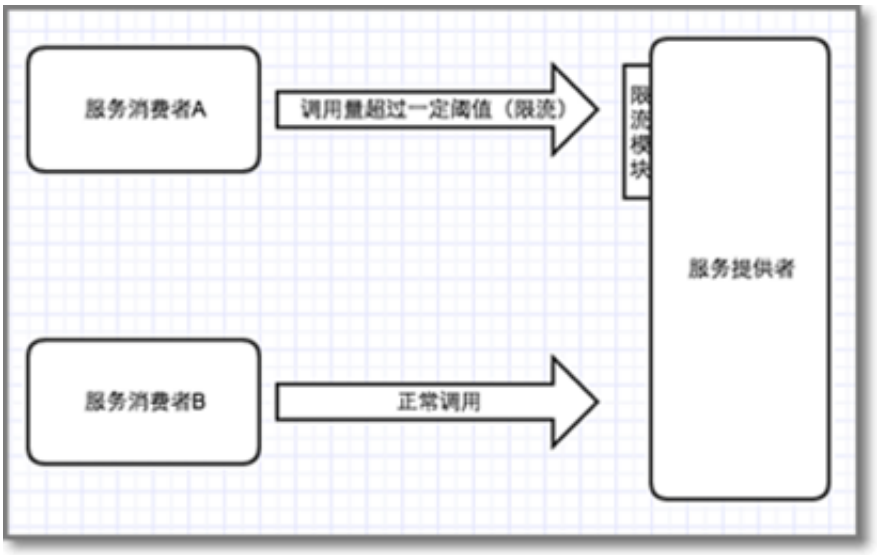
限流

可以理解为是一个控制流量阈值或调节比例的功能，在前端网站面对大流量访问的时候，可以对流量进行控制，防止大流量对后端核心系统造成破坏，导致服务不可用的情况。即通过调节流量阈值来控制通过系统的最大流量值，保证系统安全可靠运行。

基本原理

通过在服务提供者端配置限流模块代码并在 EDAS 中配置限流策略后，使服务提供者具备限流功能。此时服务消费者去调用服务提供者时，所有的访问请求都会通过限流模块进行计算，若服务消费者调用量在一定时间内容超过了预设阈值，则会触发限流策略进行限流处理。

图 524: 限流



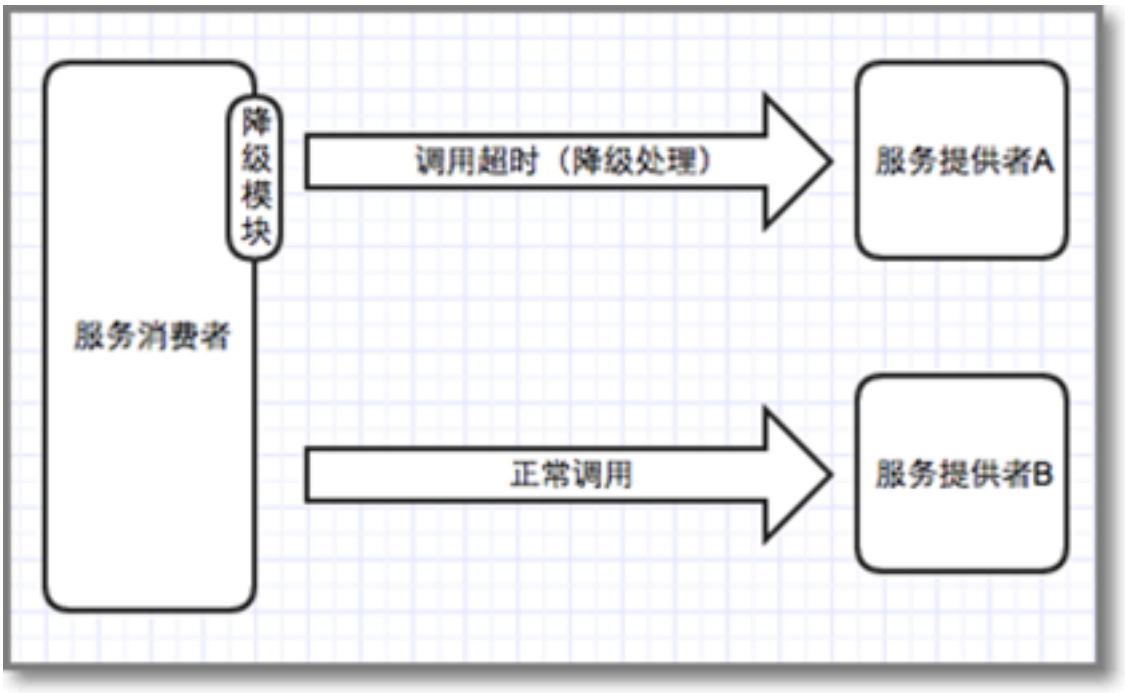
降级

在 EDAS 中，降级通常用于对下游出现超时的非核心服务提供者进行低优先级调用，确保上游核心应用（服务消费者）不被影响。

基本原理

通过在服务消费者端配置降级模块代码并在 EDAS 中配置降级策略，使服务消费者具备降级功能。此时服务消费者去调用服务提供者时，若服务提供者服务响应时间超过了预设阈值，则会触发降级策略进行降级处理。如下图所示：

图 525: 降级



21.6.7.1 限流管理

每一个应用都会提供很多服务。对于这些服务，可以配置限流规则以实现对服务的流控，确保服务能够稳定运行，拒绝超出服务能力的流量。EDAS 可以从 QPS 和线程两个维度对限流规则进行配置，确保系统在面对流量高峰时以最大的支撑能力平稳运行。

背景信息

- **HSF 限流**：出现流量高峰时，超出流控规则所定义的流量上限时，一部分调用方将出现 BlockException 错误。根据设定的阈值，在1秒内会有与设置的阈值相同个数的服务调用成功。
- **HTTP 限流**：出现流量高峰时，一部分调用方将被重定向到一个出错页面，实际访问时会跳转到淘宝首页。根据阈值设定，这里也有成功访问到服务的请求。

> **注意：** 限流规则仅适用于服务提供者，对于服务消费者是不能配置的。在配置之前请先确认该应用是否为服务提供者。

操作步骤

1. 加入限流规则代码。
   a) 登录 EDAS 控制台，单击左侧菜单的**应用管理**，进入应用列表页面，选择一个已部署的服务提供者应用进入应用管理控制台。
   b) 在应用管理控制台的左侧菜单中选择**限流降级 > 限流规则**。单击页面右上角**应用配置说明**，根据示例添加限流规则代码。
2. 编译并发布应用，具体流程请参考发布应用。

3. 回到应用管理控制台，在左侧菜单栏选择**限流降级 > 限流规则**，在页面右上角单击**添加限流规则**。
4. 设置限流规则参数。

图 526: 添加限流规则界面

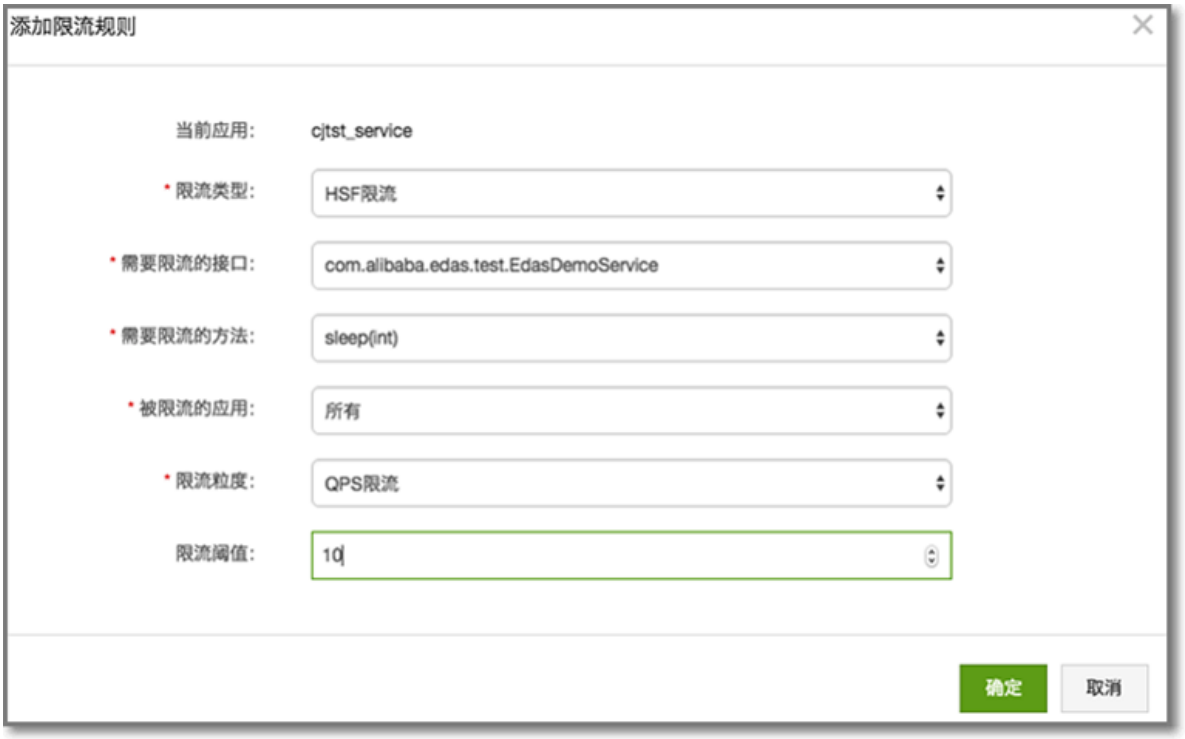


表 81: 添加限流规则参数说明

参数	说明
限流类型：	分为 HSF 限流和 HTTP 限流。请根据应用的访问类型，选择具体的限流类型。
需要限流的接口：	列出了应用中所有的接口。请根据实际情况选择需要限流的接口。
需要限流的方法：	针对选择的接口，自动加载接口内所有的方法。可以根据实际情况选择某个方法进行限流，也可以选择对全部方法都进行限流。
被限流的应用：	此时列表中将加载除当前应用以外的所有应用，因为每一个应用都可能会访问当前应用。请根据实际情况，选择被限流的应用。
限流粒度：	主要分为 QPS 限流和 Thread 限流。QPS 限流表示对请求中每秒请求次数的限制；Thread 限流是对线程数的限制。一般来讲线程越大则 QPS 越大，但是一个线程的 QPS 一般都会 >1，因为一个线程会持续发请求，请求的响应时间一般就几十毫秒。

参数	说明
限流阈值：	高于阈值时将触发限流动作。

后续操作

在**限流降级**规则页面，在相应的限流规则右侧单击**编辑**/**停用**/**启用**/**删除**等选项进行相关操作。

21.6.7.2 降级管理

每一个应用会调用许多外部服务。对于这些服务，可以配置降级规则，以实现对劣质服务的精准屏蔽，确保应用自身能够稳定运行，避免由于依赖的服务质量劣化，影响了应用自身的服务能力。

背景信息

EDAS 提供了从响应时间维度对降级规则的配置，帮助用户在应对流量高峰时，合理的屏蔽劣质依赖。触发降级规则的消费者，在指定的时间窗口内，不会再发起实际的远程调用，而是以抛出 DegradeException 异常替代。在时间窗口结束后，才会恢复原来的远程服务调用。

说明： 降级规则仅仅适合服务消费者，对于服务提供者是不能配置的，故在配置之前请先确认是否为服务消费者。

操作步骤

1. 加入降级规则代码。
   a) 登录 EDAS 控制台，单击左侧菜单的**应用管理**，进入应用列表页面，选择一个已部署的服务提供者应用进入应用管理控制台。
   b) 在应用管理控制台的左侧菜单选择**限流降级** > **降级规则**，单击页面有上角**应用配置说明**，根据示例添加降级规则代码。
2. 编译并发布应用，具体请参考发布应用。
3. 回到应用管理控制台，单击页面右上角**添加降级规则**。
4. 设置降级规则参数。

图 527: 添加降级规则



表 82: 降级规则参数说明

参数	说明
需要降级的接口：	这里将列出所有消费者正在消费的接口，根据情况，选择需要降级的接口。
需要降级的方法：	根据您上面选择的接口，自动加载所有的方法，可根据情况选择是全部降级还是限制某个方法。
RT 阈值：	触发降级的服务响应时间的阈值，超过后，将对您选择的接口或方法进行降级处理，单位毫秒。
时间窗口：	触发降级后规则持续的时长。

后续操作

在**降级规则**规则页面，单击相应的降级规则右侧的**编辑/停用/启用/删除**等选项进行相关操作。

21.6.8 容器诊室

由于应用是部署并运行在 Tomcat 容器中，EDAS 提供了容器监控功能，为您提供相应数据来判断内存，类冲突等应用运行问题。EDAS 针对应用容器提供了详细的统计功能，可以为开发者提供当前应用运行机器上的 JVM 堆 / 非堆内存、类加载（Class Loader）、线程、Tomcat 连接器的统计数据。与基础监控一样，可查看应用下的单机数据。

背景信息

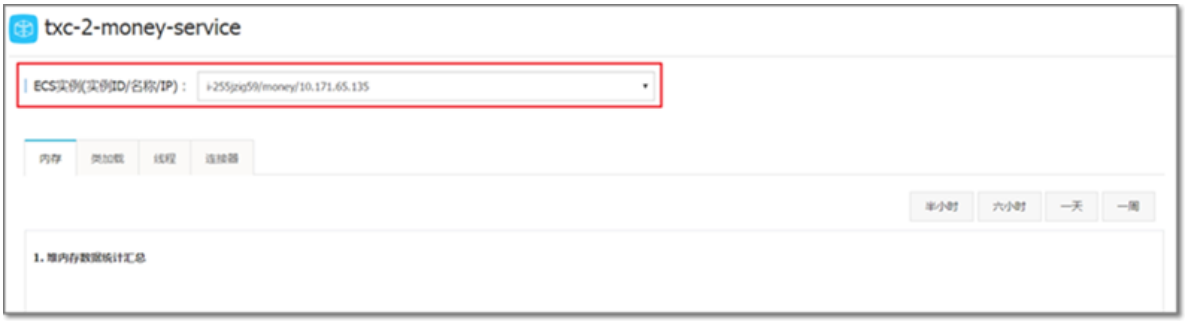
容器监控和基础监控的不同之处如下：

- 监控对象不一样，基础监控面向的是机器，容器监控面向是应用所在的容器。
- 容器诊室不支持“集群”方式查看诊断信息，仅支持单机查看。
- 基础监控存在一定的时延，容器监控除了内存监控外，其他数据由于不需要后续的统计计算，所以几乎是实时的。

操作步骤

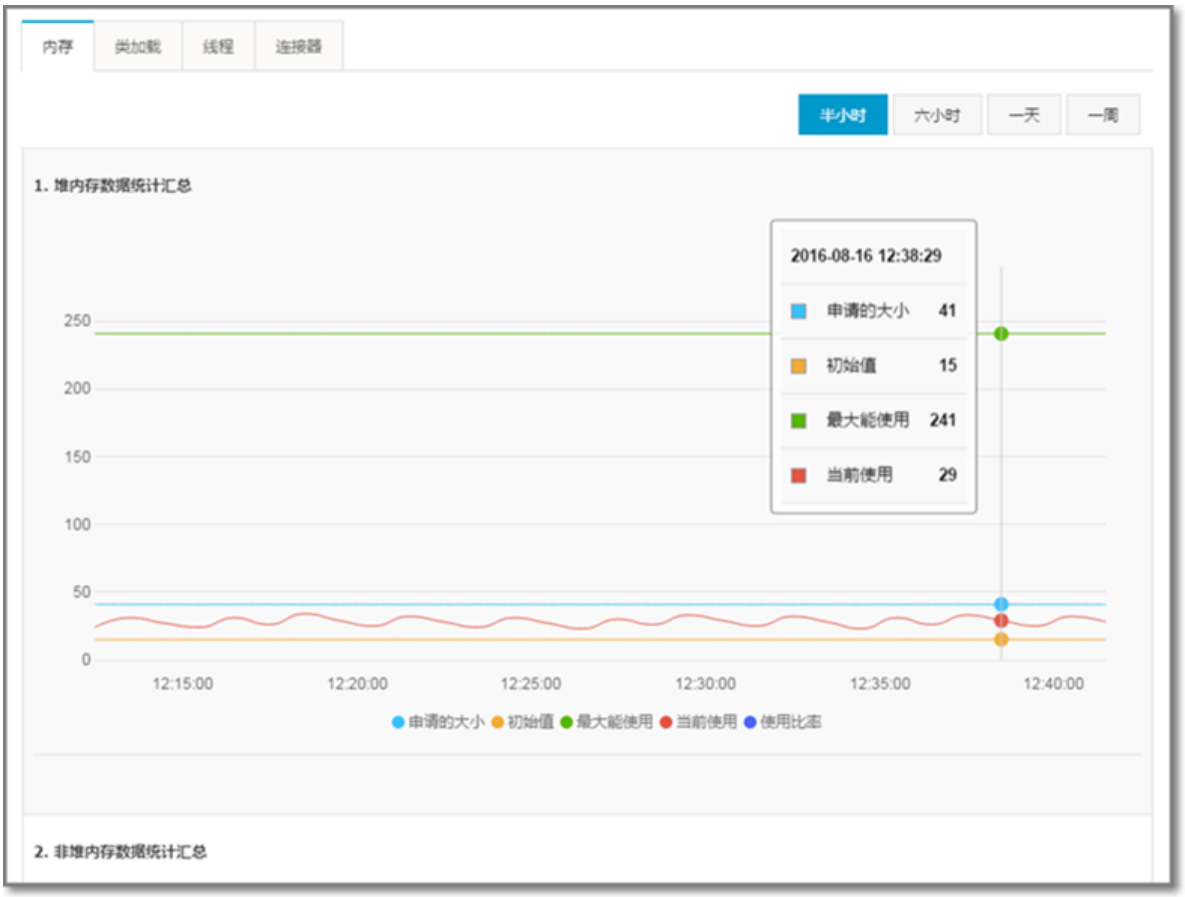
1. 登录 EDAS 控制台，在左侧导航栏中单击**应用管理**，在应用列表中单击具体的应用。
2. 在应用详情页面左侧的导航栏中单击**容器诊室** ，进入容器诊室页面。
3. 单击 **ECS 实例(实例ID/名称/IP)**右侧的下拉箭头，选择机器。

   图 528: 选择ECS实例

4. 单击具体页签，监控容器的相关指标。

   图 529: 监控容器指标



容器诊室页面中包含以下4个页签：

- **内存**：内存监控以机器为单位，EDAS 能提供相应应用中 Tomcat 容器所在 JVM 进程的堆（heap） 与非堆（non heap）的统计信息。EDAS 提供了**半小时**、**六小时**、**一天**和**一周**4种时间间隔。

  该页面也是用户进入容器诊室的默认页面。
- **JAR**：提供实时的 JAR 包的加载情况。开发者有时会碰到应用中存在 JAR 包版本冲突的问题，此功能可以很直观的体现出相应的 JAR 加载的路径，从而简化此类问题排查的成本。
- **线程**：当前 JVM 进程中所有线程的基本信息，包括 ID、状态、线程名，且统计字段均为 JVM 原生信息。
- **连接器**： Tomcat 连接器指的是 Tomcat 的 XML 配置中的<Connector />，可理解为每一个<Connector />的配置为一行拉取出来的信息。该视图提供最近十分钟之内对应连接器的运行情况。

从连接器的角度每个连接器会有一定数量的线程（既线程池）服务于该连接器，以处理到来的请求；有时遇到并发或者吞吐的瓶颈时，需要知道相应连接器中线程池处理情况，以 HTTP 连接器为例，当配置如下的 XML 时：

```
<Connector port="8080" protocol="HTTP/1.1" maxThreads="250" .... />
```

单击连接器右侧操作选项中的**线程池信息**，会看到比较详细的信息，如下图：

图 530: 监控线程池信息



上图说明该应用的几乎没有负载，如果“忙碌线程数”快接近所配置的线程池最大值时，说明系统正面临着严重的并发问题，此时应该考虑将应用扩容或者优化业务代码。

21.7 全局配置推送

通过全局配置推送，用户可以实时的增加、修改、删除配置，来动态的使得配置生效，无需更改代码、重新发布服务、重启服务。

背景信息

配置 在 EDAS 中是由三元组（group、DataId、Content）组成的一个配置信息，配置信息中的三个要素 group、DataId、Content 的意义如下

- group：分组名称，是命名空间，如 Java 中的 package，最大 128 个字符。
- DataId：配置名称，如 Java 中的类名，最大 256 个字符。group+DataId 确定一个配置，对应一个值。对于特殊字符，group 和 DataId 命名中仅允许包含”.”、”:”、”-“、”_” 这 4 种特殊字符。

- Content：配置中值的内容，最大 1024 个字符。

> **说明:** 配置推送中的元素 **分组** 是在服务分组中创建。如果没有创建任何服务，进入配置页面，会有一条系统自动产生的配置，用户可以忽略这条系统配置。

操作步骤

1. 登录 EDAS 控制台。
2. 在左侧导航栏中，选择 **服务市场** > **服务分组**。

> **说明:** 弹出升级容器提示，选择 **暂不升级**。

3. 单击页面右上角的 **创建服务组**，在弹出的 **创建 HSF 服务组** 对话框中输入**服务组名称**，单击 **创建**。
4. 在控制台左侧导航栏中，单击 **全局配置推送**。
5. 在配置列表中选择配置所在的 Region，并选择刚刚创建的服务分组，在页面右上角单击 **创建配置**。
6. 在创建配置对话框中，输入 DataId 和 Content，单击 **确定**。

> **说明:** group 在**全局配置推送**页面已经选择，该对话框中不可修改。

图 531: 创建配置



配置完成后，可以返回全局配置推送页面查看刚刚创建的配置。

后续操作

也可以对全局配置进行更新、删除等操作。

21.8 账号管理

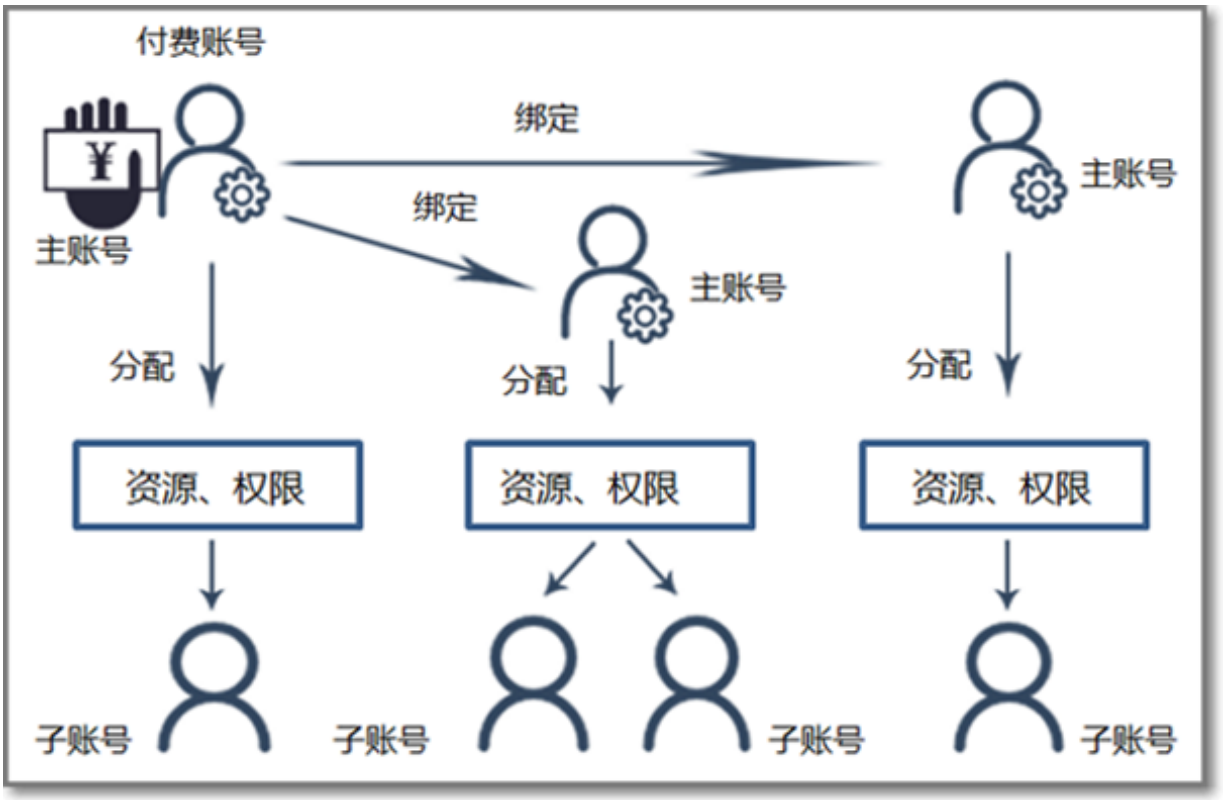
为了更好地保护企业信息安全，实现企业级的账号管理，包含主、子账号体系。

21.8.1 EDAS 账号体系介绍

EDAS 提供了完善的主子账号管理体系。主账号可以分配权限和资源给多个子账号，实现按需为用户分配最小权限，从而降低企业信息安全风险，减轻主账号的工作负担。

接入阿里云访问控制（Resource Access Management，RAM）的账号体系之前，EDAS 本身拥有一套严谨的主子账号体系，实现了人权分离。2016年7月份升级后，EDAS 也同时支持了 RAM 主子账号体系。

EDAS的账号体系



付费账号是指用于购买 EDAS 产品的账号，该账号同时也是一个主账号。针对同一个企业内通常有多个部门同时使用 EDAS 的场景，EDAS 支持用户用付费账号购买 EDAS，然后将此付费账号绑定多个主账号，即一人付费多人使用，实现了客户利益最大化。

说明: 一个付费账号最多只能绑定 5 个主账号。

付费账号和主账号的关系

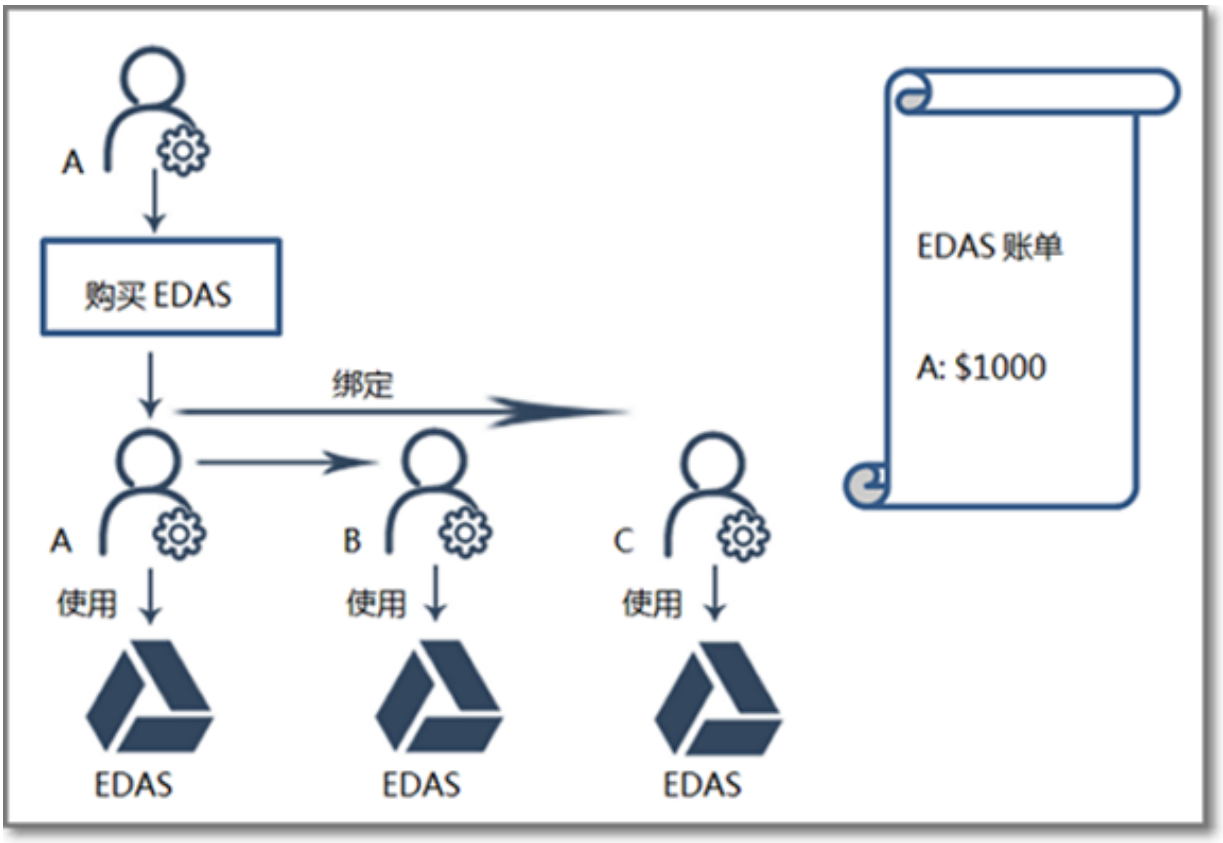
- 所有的付费账号都是主账号，但不是所有的主账号都是付费账号。
- 所有的主账号都是一个独立的个体，拥有该主账号购买的所有资源，且对 EDAS 拥有所有的权限（除了没有绑定主账号权限）。
- 付费账号和主账号是相互独立的两个阿里云账号，付费账号与主账号之间的绑定付费关系只在购买 EDAS 的时候有效，即付费账号只能替主账号购买 EDAS，不能代替购买其他资源。被绑定的主账号如果需要使用 EDAS 中使用的其他资源，如 ECS、SLB等（具体的资源详情请参考资源管理），则需要用本账号购买。

下文以几个典型场景为例来进一步说明 EDAS 账号体系的运用。

场景一

某公司用账号 A 购买了 EDAS，那么 A 就是一个付费账号，同时也是一个主账号。该公司还有其他两个部门都需要使用 EDAS，于是账号 A 将这两个部门的账号 B、C 绑定为主账号。那么 B、C 账号不需要付费购买 EDAS 就可以正常访问 EDAS了。如下图所示：

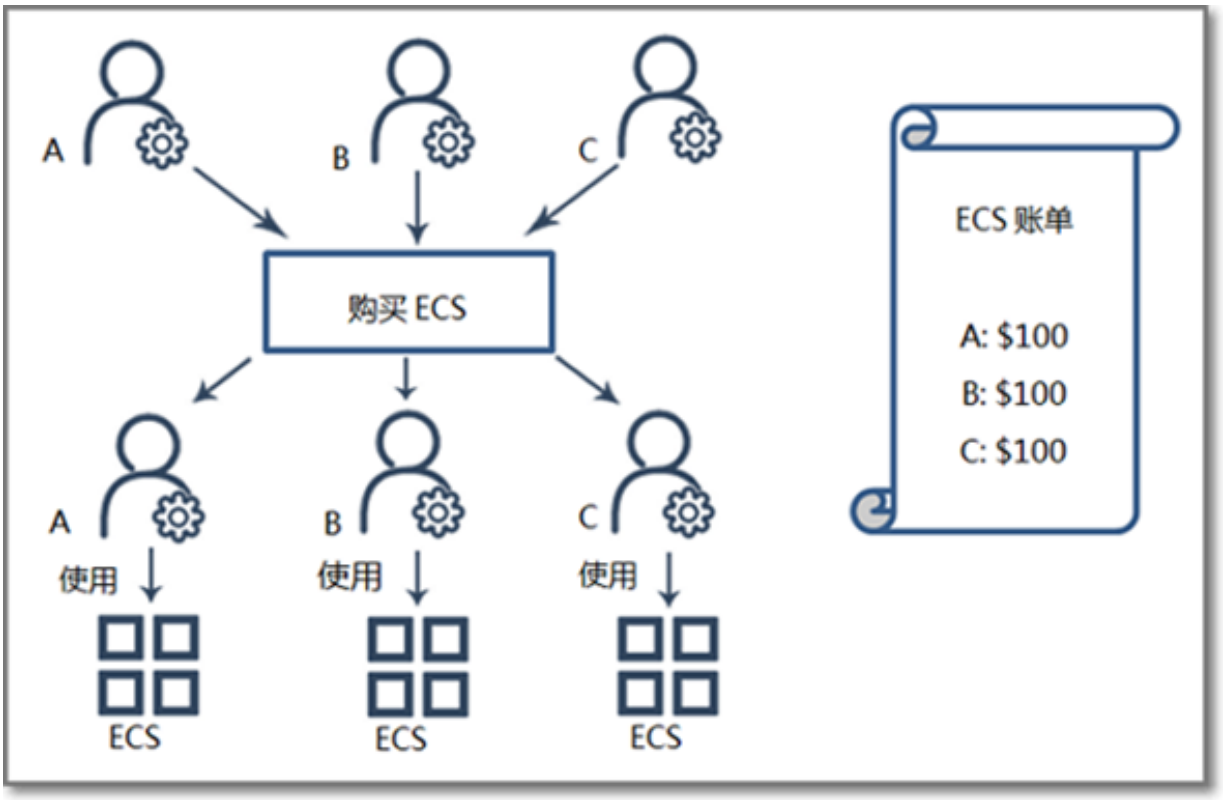
图 532: EDAS账号体系场景示意一



场景二

假设 B、C 需要使用 EDAS 的完整功能，如创建应用、运行应用等，则必须自行购买 ECS 等资源，而不能用账号 A 进行购买，如下图所示：

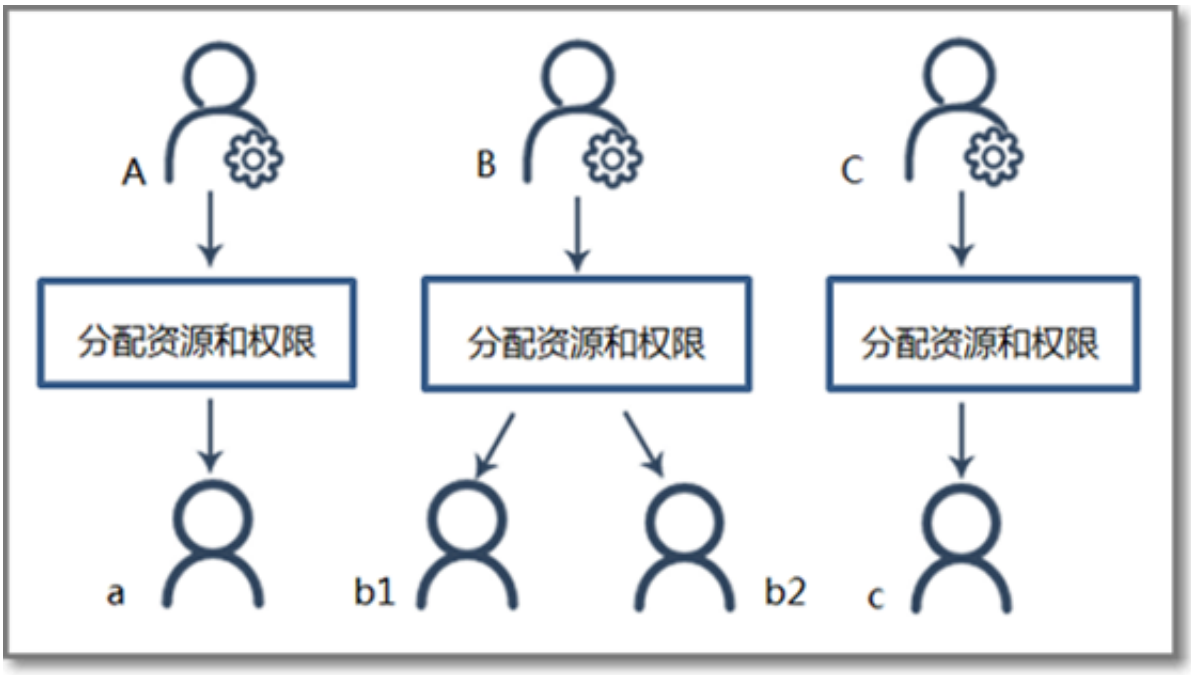
图 533: EDAS账号体系场景示意二



场景三

准备好资源后，三个主账号对应的部门分别创建了各自的子账号，用于权限及资源的具体分配和管理。A 账号给子账号 a 授予了所有的 ECS 资源和所有权限；B 账号创建了应用管理员和运维管理员的两个角色，并将这两个角色分别授予给了子账号 b1，b2；C 账号创建了一个查看应用的角色，授权给了 c。如下图所示：

图 534: EDAS账号体系场景示意三



21.8.2 管理主账号

EDAS 中，主账号拥有其下所有资源和 EDAS 的所有操作的权限。购买 EDAS 的主账号也是付费账号。付费账号在购买 EDAS 服务后即可绑定企业内其他主账号， 这些主账号就可以在不付费购买 EDAS 的情况下使用 EDAS，从而为客户节省资源成本。

21.8.2.1 绑定主账号

付费账号在购买 EDAS 服务后即可绑定企业内其他主账号，这些主账号就可以在不付费购买 EDAS 的情况下使用 EDAS，从而为客户节省资源成本。

操作步骤

1. 在 EDAS 控制台左侧菜单栏选择**账号管理** > **主账号管理**。
2. 单击页面右上角**绑定主账号**。
3. 输入主账号、应用实例数限额，选择产品套餐，单击**确定**。

图 535: 绑定主账号界面



- **主账号**：主账号必须是一个没有购买过 EDAS 服务的合法的阿里云账号。
- **应用实例数限额**：表示该主账号及其名下的子账号所拥有的所有应用下面的实例数上限。付费账号在给主账号分配此限额的时候，其下的所有主账号的限额之和不能大于付费账号的总应用实例数限额。
- **产品套餐**：付费账号给绑定的主账号分配的套餐与自身的套餐类型相同。

说明:

- 单个付费账号最多能绑定 5 个主账号。
- 如需解绑主账号，请采用工单方式提交。
- 对于绑定的主账号，每个用户的实际应用实例数≤该用户的应用实例数限额。
- 对于绑定的主账号，**付费账号**的**实际应用实例数** + 该付费账号绑定的所有**其他主账号**的应用实例数**限额之和**≤付费账号的应用实例数限额。

图 536: 主账号管理界面



21.8.2.2 角色管理

主账号可以通过创建不同的角色，为子账号自定义不同的操作权限。

操作步骤

1. 在 EDAS 控制台左侧菜单选择**角色管理**。
2. 在页面右上角单击**添加角色**。
3. 输入角色名称，将左边栏的权限添加到右边，单击**确定**。

图 537: 添加角色界面



角色添加成功后，还可以对该角色进行**查看权限**，**管理权限**及**删除**操作。

21.8.2.3 查看所有权限

您也可以在控制台查看 EDAS 系统所拥有的所有权限。

操作步骤

1. 在 EDAS 控制台左侧菜单栏选择**账号管理** > **所有权限**。
2. 点开各个层级即可以查看具体权限信息。

图 538: 权限信息页面



21.8.3 管理子账号

子账号管理主要包括主账号和子账号的关系、子账号管理和主账号对子账号的操作三部分内容。

21.8.3.1 子账号简介

主账号在使用EDAS过程中，通常需要不同身份的用户来做不同类型的工作，如应用管理员（创建应用、启动应用、查看应用、删除应用等）、运维管理员（查看资源、查看应用监控，管理报警规则、管理限流降级规则等）等。主账号可以通过给子账号分配不同的角色和资源，来实现分权分责。这种主子账号的权限模式，类似于Linux系统中系统用户和普通用户，系统用户可以给普通用户授权、撤销权限。

RAM子账号说明

- RAM 账号由主账号在 RAM 系统中创建，不需要校验合法性，名字在主账号内唯一即可。
- 不同于一般的阿里云账号登录，RAM 账号有独特的登录入口，入口在 RAM 控制台中有说明。

主账号和子账号的关系

- EDAS系统中，主账号可以通过绑定子账号，避免与其他用户共享账号秘钥，按需为子账号分配最小权限，实现各谋其位，各司其职的高效企业管理。
- 主账号绑定子账号，创建主子关系，这种主子关系只在EDAS内部有效，在EDAS以外的环境主账号和子账号是相互独立的两个阿里云账号。
- 子账号必须是合法的阿里云账号。

- 一个主账号既可以是绑定子账号的主账号，也可以是另一个主账号的子账号。

21.8.3.2 创建子账号

介绍如何在没有RAM（Resource Access Management，资源接入管理）账号体系的环境中创建子账号。

背景信息

在子账号管理页面可以通过绑定子账号的功能，将一个其他的阿里云账号绑定为子账号。

操作步骤

1. 在EDAS控制台左侧菜单栏选择**账号管理** > **子账号管理**。
2. 单击页面右上角**绑定用户**。

图 539: 子账号管理页面



3. 输入合法的阿里云账号，作为子账号。可以通过添加账号来批量绑定。

图 540: 绑定用户界面



21.8.3.2.1 创建RAM子账号

EDAS 系统支持阿里云访问控制（Resource Access Management，RAM）的账号体系，主账号可以通过创建 RAM 子账号，避免与其他用户共享账号秘钥，按需为子账号分配最小权限，实现各司其职的高效企业管理。

操作步骤

1. 在 EDAS 控制台左侧菜单栏依次单击 **账号管理** > **子账号管理**。

   说明: 在 RAM 控制台中创建的 RAM 用户默认就是 EDAS 产品中该主账号的的子账号。

2. 单击页面右上角的 **绑定用户**，会跳转到 RAM 控制台。
3. 在 *RAM* 控制台单击 **用户管理**。
4. 单击页面右上角的 **新建用户**。
5. 在创建用户对话框填写登录名及其他信息，单击**确定**。

   注意: 登录名要求在主账号内唯一。



在用户管理页面能看到新建的用户名，则表示一个 RAM 用户创建成功。

6. 在用户管理页面单击用户的**登录名/显示名**，进入用户信息页面。
7. 在 **Web 控制台登录管理**区域，单击 **启用控制台登录**。



8. 在重置密码对话框中输入**新密码**和**确认密码**，选择是否**要求该账号下次登录成功后重置密码**。设置完成后，单击**确定**。



至此，一个可以登录控制台的 RAM 用户就创建成功了。

21.8.3.2.2 RAM 子账号登录 EDAS 控制台

介绍如何通过RAM 子账号登录 EDAS 控制台。

操作步骤

1. 在 RAM 控制台的 **概览**页，找到 **RAM 用户登录链接**。



说明: RAM 用户的登录链接，是根据主账号不同而不同的，所以不同的主账号创建的 RAM 账号的登录链接是不同的。

2. 在 RAM 用户登录页面填写字用户名称和子用户密码，然后单击**登录**进入 RAM 账号的 RAM 控制台。



说明:

- 企业别名：企业别名不用子用户填写，在登录链接中已经存在。
- 子用户名称：主账号在创建 RAM 用户的时候设置的登录名。

- 子用户密码：主账号在给 RAM 账号启用控制台登录的时候设置的密码。如果主账号当时选择了**要求该账号下次登录成功后重置密码**，那么 RAM 账号在第一次登录控制台后，会被要求重置密码，以后登录都用重置之后的密码。

3. 进入 RAM 控制台后，单击页面最上方导航栏的**产品与服务**，找到互联网中间件类目下的**企业级分布式应用服务 EDAS** 进入 EDAS 控制台。



21.8.3.2.3 RAM 账号授权

介绍如何为RAM账号授权。

背景信息

RAM 账号有两种授权方式：

- RAM 授权
- EDAS 授权

这两种方式是互斥的，即当 RAM 账号有了 RAM 授权，那么就不能再在 EDAS 中授权。只有在 RAM 控制台上解除了 RAM 授权，才可以在 EDAS 控制台授权 EDAS 权限。如果 RAM 账号要在 EDAS 授权，那么就不能有 RAM 授权。

RAM 授权的粒度是 EDAS 服务级别的，即 RAM 授权表示允许用户拥有 EDAS 的所有权限。RAM 授权或者解除授权只能在 RAM 控制台上操作。

操作步骤

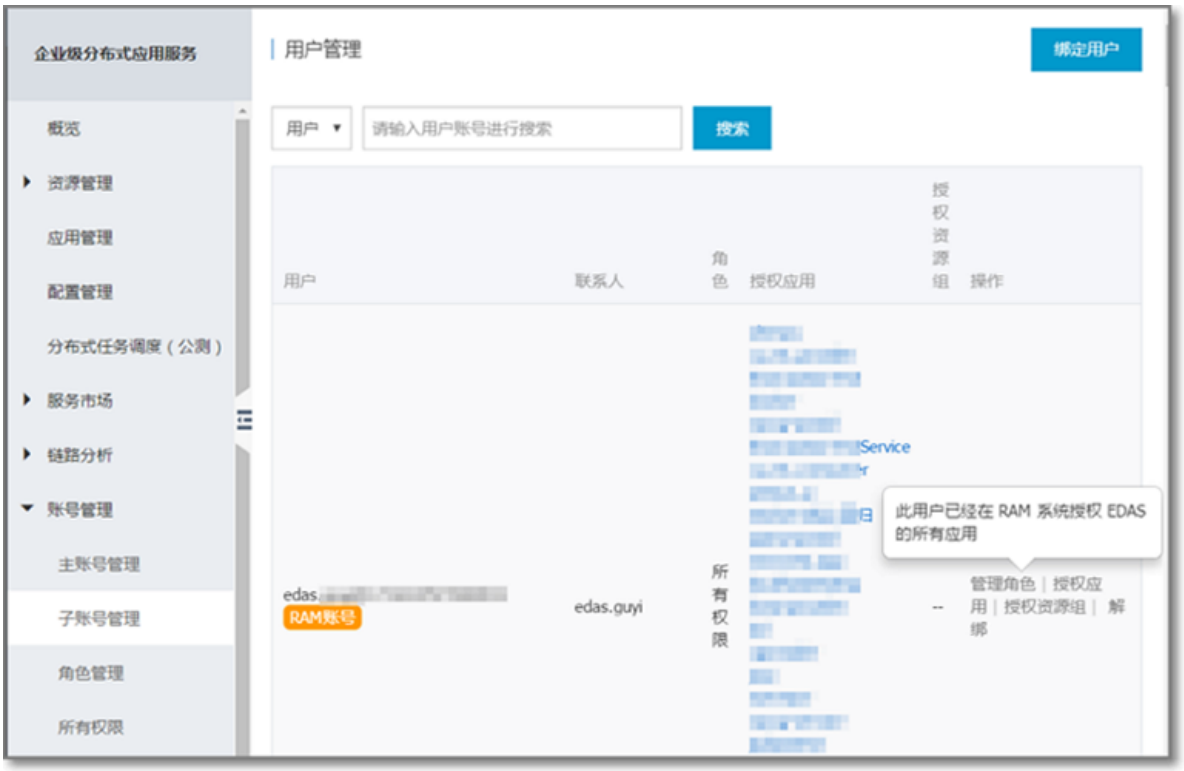
1. 单击 RAM 控制台左侧菜单栏的**用户管理**，选择需要授权的用户，单击该用户右侧操作选项中的**授权**。



2. 在对话框左侧搜索框中输入 **EDAS**，选择 **AliyunEDASFullAccess** 加入到右侧**已选授权策略名称**，单击**确定**授权该账号 EDAS 的所有权限。



授权完成后，用主账号登录 EDAS，在左侧菜单栏选择**账号管理>子账号管理**，就可以看到这个账号被授予了所有权限、资源、应用。此时在 EDAS 上将无法对此账号进行授权操作。



后续操作

解除 RAM 授权的步骤如下：

1. 在 RAM 控制台左侧菜单栏单击**用户管理**，选择需要授权的用户，单击该用户的**操作**中的**授权**。
2. 将右边栏中的 **AliyunEDASFullAccess** 移至左边，单击**确定**即可。

授权解除后，用主账号登录 EDAS，在左侧菜单栏选择**账号管理>子账号管理**可以看到这个账号被取消了所有资源和权限。此时就可以在 EDAS 上对此账号进行授权操作了。



21.8.3.2.4 RAM 用户解绑

RAM用户的解绑，即将此用户删除。

操作步骤

1. 登录 *RAM控制台*。
2. 单击左侧菜单栏的 **用户管理**，在要解绑的用户对应行右侧单击操作选项中的**删除**。



21.8.3.3 主账号对子账号的操作

主账号对子账号的操作包括管理角色、授权应用、授权资源组和解绑。这四类操作步骤类似，以管理角色为例进行描述。另外三种操作不再详细描述，仅做简要说明。

背景信息

主账号可以通过将角色授权给子账号，从而使得子账号拥有该角色里面的权限。

操作步骤

1. 在EDAS控制台左侧菜单栏选择**账号管理** > **子账号管理**。
2. 单击要授权的子账号右侧操作选项中的**管理角色**。

图 541: 子账号管理页面



3. 勾选需要授权的角色，单击**确定**。

图 542: 管理角色界面



设置完成后，在子账号管理页面，子账号的角色一列就可以看到授权的角色名称。

说明:

- **授权应用**

  主账号可以通过将应用授权给子账号，从而使得子账号拥有对该应用的所属权限。

  授权应用后，只表明子账号对应用有所属权限，至于对该应用的具体操作，如应用启动、删除应用等，需要通过授权角色来授予操作的权限。所以授权应用后，一般需要授权相应角色，才能使子账号对该应用有操作权限。

- **授权资源组**

  主账号通过授权资源组，可以使子账号拥有对相应资源的使用权。资源组的概念请参考资源管理。

- **解绑**

  主账号通过解绑，可以解除子账号和主账号的主子关系。解绑后，之前分配的角色、应用、资源组等关系也都被解除。如果解绑的子账号没有购买过 EDAS 服务，解绑后将不再能登录 EDAS 控制台。

21.9 常见问题

介绍产品在开发、使用过程中的常见问题及解决方案。

21.9.1 开发常见问题

开发常见问题包括Ali-Tomcat、轻量配置中心、HSF、HSF错误码及其他开发问题。

21.9.1.1 Ali-Tomcat问题

介绍Ali-Tomcat开发过程中的常见问题及解决方案。

- **服务认证失败**

  错误信息：

  java.lang.Exception: Service authentication failed

  说明：此问题只会发生在 EDAS 生产环境。

  解决方案：

  - 安装 EDAS Agent 使用的 AccessKey (AK)，SecretKey (SK) 错误，或者安装时使用 Web 页面终端安装导致 AK，SK 错误。

    1. 执行cat /home/admin/.spas_key/default ；
    2. 登录 EDAS 控制台，在左侧菜单选择**资源管理>云服务器 ECS**，点击**安装 Agent**按钮；
    3. 比较弹出的界面中 AK，SK 是否与上述值一致（注意大小写，Web 终端会导致大小写不一致）。

  - ECS 的时间不准确，比准确时间慢 30s 以上，请调整 ECS 时间。

  执行 date 命令，查看是否时间不准确。

- **未知主机异常**

  错误信息：

  Caused by: java.net.UnknownHostException：iZ25ax7xuf5Z

  解决方案：**iZ25ax7xuf5Z** 是当前主机名，确定 /etc/hosts 存在本机 IP 及当前主机名的配置。如果没有请添加，例如：192.168.1.10 iZ25ax7xuf5Z

- **如何进行问题定位**

Ali-Tomcat 启动时会因为各种错误导致启动失败，需要查看 catalina.out 及 localhost.log 这两个日志文件进行问题定位。如果使用 Tomcat4E 插件，则会在 Eclipse 的控制台看到具体的问题描述。

- **异常出现时，如何判断是自身代码错误，还是 EDAS 出现错误**

  解决方案：查看错误堆栈的最后部分，是否是自身代码。例如：

  Caused by: com.yourcompany.yourpack

- **端口被占用**

  错误信息：

  Caused by: java.net.BindException: Address already in use: JVM_Bind

  解决方案：端口被占用，处理方式同轻量配置中心端口冲突。

- **com.ali.unit.rule.Router 初始化失败**

  错误信息：

  SEVERE: Context initialization failed java.lang.NoClassDefFoundError: Could not initialize class com.ali.unit.rule.Router

  解决方案：不能链接地址服务器 jmenv.tbsite.net，请进行域名绑定。

  在hosts文件中添加以下内容进行地址服务器域名绑定：

  192.168.1.10 jmenv.tbsite.net

  其中 192.168.1.10 改为您使用的轻量配置中心的地址，hosts文件的路径如下：

  - Windows： C:\Windows\System32\drivers\etc\hosts
  - Linux： /etc/hosts

- Pandora 启动失败，QoS 端口绑定异常

  错误信息：

  Cannot start pandora qos due to qos port bind exception

  解决方案：QoS 端口被占用，处理方式同轻量配置中心端口冲突。

- JVM 内存不足

  错误信息：

  java.lang.OutOfMemoryError

解决方案：设置内存大小，具体方案请网络搜索 **JVM 内存设置。**

- 部署 WAR 包出现空指针异常

  错误信息：

  deployWAR NullPointException

  解决方案：查看 WAR包是否正常，执行 jar xvf xxx.war 确定 WAR 可以正常解压。

- com.taobao.diamond.client.impl.DiamondEnvRepo 初始化失败

  错误信息：

  Could not initialize class com.taobao.diamond.client.impl.DiamondEnvRepo

  解决方案：地址服务器中 Diamond 数据为空，需要确信地址服务器配置正确且稳定，访问 *http://jmenv.tbsite.net:8080/diamond-server/diamond* ，能正常返回。

21.9.1.2 轻量配置中心问题

- **执行 startup.bat 和 startup.sh 不能正常启动。**

  这个问题的原因可能有几种，请按照提示的错误信息解决。

  - **提示错误**：

    Java version not supported， must be 1.6 or 1.6+

    解决方案：

    检查 Java 是否正确安装。如未安装，请先正确安装；

    确保安装 Java 1.6 及以上版本。

  - **提示错误**：

    Unable to start embedded Tomcat servlet container

    或者

    Tomcat connector in failed state

    解决方案：

    检查 8080 端口是否被占用，如果端口已经被其他程序占用，请先停止该占用端口的程序，再执行启动脚本。具体操作如下：

    Windows

1. 进入 cmd，执行 netstat -aon|findstr “8080”，记录查询到数据的最后一列数字，即 PID (Process ID)，例如 “2720”；
2. 执行 tasklist|findstr “2720”，会显示当前 PID 对应的程序，例如 javaw.exe;
3. 执行 taskkill /PID 2720 /T /F;
4. 再次启动轻量配置中心。

Linux

1. 执行 netstat -antp|grep 8080，会显示占用 8080 端口的进程号及程序，例如“2720”；
2. 执行 kill -9 2720;
3. 再次启动轻量配置中心。

- **提示错误**：

  Caused by: java.net.UnknownHostException：iZ25ax7xuf5Z

  解决方案：

  iZ25ax7xuf5Z 是当前主机名，确定 /etc/hosts 存在本机 IP 及当前主机名的配置。如果没有请添加，例如：192.168.1.10 iZ25ax7xuf5Z

- **多网卡的机器如何指定启动的** IP？

  解决方案：

  在启动脚本 startup.bat 或 startup.sh 中添加启动参数：

  -Daddress.server.ip={可以访问的IP}

- **自定义服务发布端IP**

  发布服务过程中，有时候需要将服务发布在虚拟网卡，或者是一个跟本机相关联的非物理存在的 IP（例如 ECS 的弹性 IP）上。如果通过 -Dhsf.server.ip 指定了该虚拟 IP，启动服务的时候就会报错，导致服务发布失败，因为发布过程中在本机网卡中找不到这个虚拟 IP。

  为了解决这个问题，EDAS 推出了自定义服务发布端 IP 的功能，允许服务端发布服务到配置中心的时候不指定任何 IP。服务发布成功后，再对 IP 地址进行修改，重新发布服务即可。服务调用方则不用做任何更改。

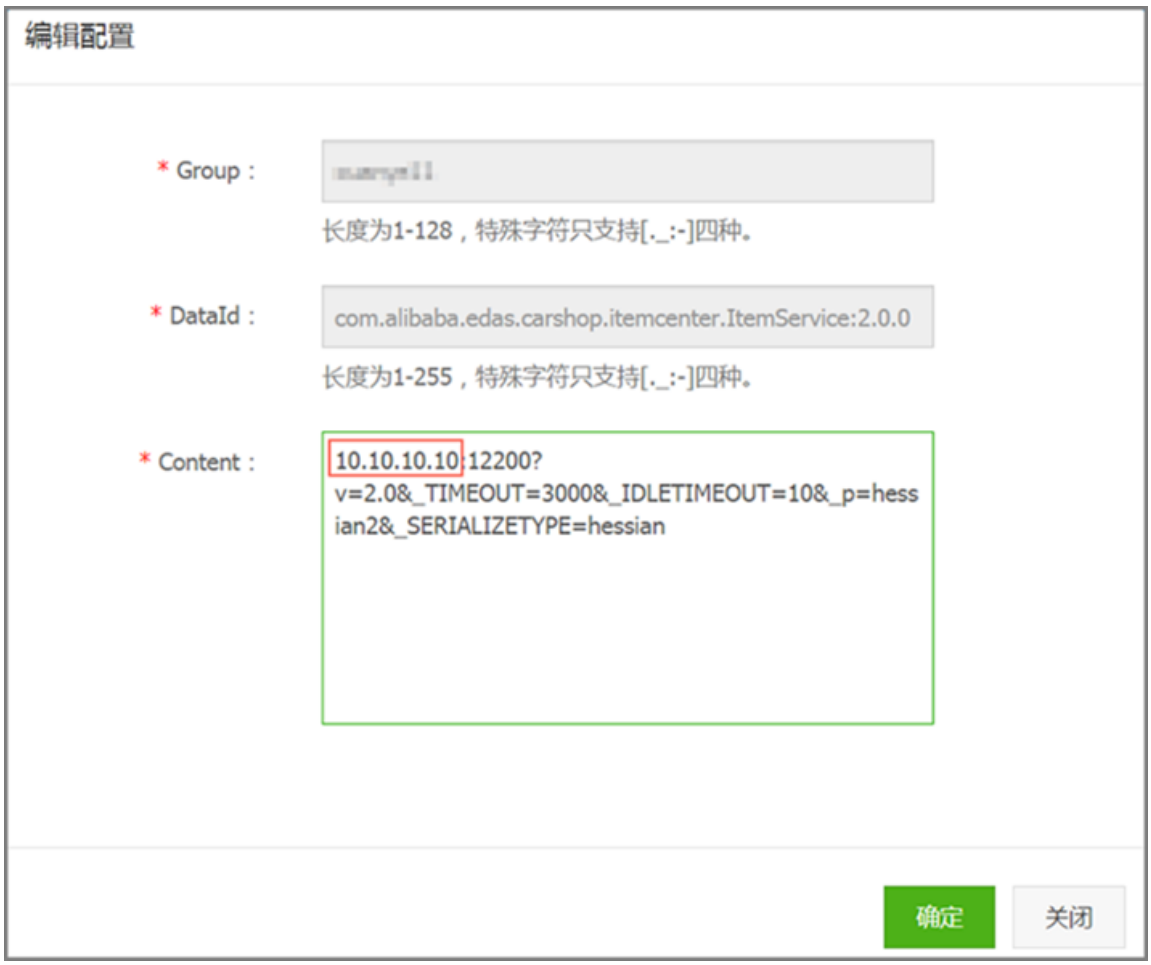
  具体操作如下：

  1. 服务发布成功后，在**配置列表**中找到发布的服务，单击右侧**更新**。

     您也可以在**服务列表**中找到发布的服务。

2. 在**编辑配置**页面的 Content 输入框中修改 IP 地址。

> **注意:** 没有特殊需求时不要随意更改 IP 地址后面的内容，以免发生服务调用错误。



3. 单击**确定**保存。
4. 重启服务。带有新地址的服务会被重新注册，使更改生效。

修改完成后，服务调用方不用做任何更改，按照正常调用方式调用服务。您可以通过 *{user.home}\logs\configclient\config-client.log* 路径下的日志观察到服务调用的真正 IP。查看日志中关键字 **[Data-received]** 后的内容，可以看到调用的服务的完整信息。

21.9.1.3 HSF常见问题

- **定位及解决 HSF 问题**

  HSF 的问题描述会记录在 /home/admin/logs/hsf/hsf.log 中，如果出现与 HSF 相关的问题，请查询此文件进行错误定位。HSF 的错误一般都会有对应的错误码，根据错误码速查即可找到对应的解决方案。

- **HSF 如何设置超时时间**？

  可通过 HSF 标签：methodSpecials 和 clientTimeout 进行配置。

  - methodSpecials: 为方法单独配置超时时间（单位 ms）
  - clientTimeout: 为接口中所有方法设置统一的超时时间（单位 ms）

  超时时间配置的优先级由高到低是：

  客户端 methodSpecials > 客户端 clientTimeout > 服务端 methodSpecials > 服务端 clientTimeout

  Consumer 标签配置示例：

  <**hsf:consumer**id="service"interface="com.taobao.edas.service.SimpleService"

  version="1.1.0"group="test1"clientTimeout="3000"

  target="10.1.6.57:12200?_TIMEOUT=1000"maxWaitTimeForCsAddress="5000">

  <**hsf:methodSpecials**>

  <**hsf:methodSpecial**name="sum"timeout="2000"></**hsf:methodSpecial**>

  </**hsf:methodSpecials**>

  </**hsf:consumer**>

- **HSF invalid call is removed**

  报错信息:

  invalid **callis** removed because **ofconnection** closed

  报错原因：

  - 网络闪断：客户端和服务端建立的连接后，客户端发起调用请求，若服务端还在处理该请求，此时同时没有达到客户端超时，因网络等各类问题导致了客户端主动关闭了连接，此时会报错。
  - 服务端重启：客户端发起请求后，等待客户端响应，但此时客户端重启，导致socket断开，客户端收到操作系统connection closed回调，将报错。

  解决方案

  如果业务是幂等性的，可以由业务做重试机制处理。同时检查服务器网络，一般情况是网络断开（闪断）导致。

- **HSF 启动时绑定 IP、端口失败**

问题： HSF 启动时报错，错误如下。

Java.net.BindException: Can’t assign requested address

原因：获取当前 IP 端口失败。

解决方案： 通过设置 JVM 参数解决。

-Dhsf.server.ip=您本地网卡的IP -Dhsf.server.port=12200

- **如何设置用户日志不被覆盖**

  问题：使用EDAS后，log4j日志无法输出。

  原因：log4j 日志被覆盖会导致日志无法输出。

  解决方案：设置 JVM 参数 -Dlog4j.defaultInitOverride=false，则会输出用户日志。

- **HSF 其他**

  报错信息：启动时出现如下报错。

  java.lang.IllegalArgumentException: HSFApiConsumerBean.ServiceMetadata.ifClazz is null。

  解决方案：接口对应的类没有加载成功，请确认接口类在 class path 中可以被加载到。

  报错信息：failure to connect 10.10.1.1

  解决方案：

  确认是否在同一个 VPC、同一个地区。如果不是那不能连通。

  确认是否在同一个安全组，如果不在同一个安全组则需要开通对应的 12200 端口。

  执行 telnet 10.10.1.1 12200 查看端口是否可以连通。如果不能连通，请确认 10.10.1.1 机器的防火墙设置。

21.9.1.4 HSF错误码

错误编码：**HSF-0001**

报错信息：

HSFServiceAddressNotFoundException 未找到需要调用的服务目标地址

描述信息：

需要调用的目标服务为：xxxx；组别为：xxxx。

解决方案：

1. 名称不匹配，检查发布者和消费者配置的服务名字、版本号、组别，这3个要一致（注意大小写也要一样，同时要注意前后不能有空格）。
2. Tomcat 容器是否启动报错，查看 Tomcat 安装目录 /logs/catalina.out localhost.log.2016-07-01（当前日期）这2个日志文件是否有异常，如果有异常先解决异常。
3. 服务分组未创建，登录 EDAS 控制台，在左侧菜单选择**服务市场>服务分组** 查看应用的服务分组是否已经创建。例如：

   ```
   <hsf:provider
           id="sampleServiceProvider" interface="com.alibaba.edas.SampleService" ref="target"
           version="for-test" group="your-namespace" ></hsf:provider>
   ```

   对应的 group：*your-namespace*需要在服务分组列表中。
4. 鉴权失败，查看**服务提供者**机器的日志 /home/admin/configclient/logs/config.client.log，是否存在 spas-authentication-failed 错误。如果存在此类型错误：
   - 同上：服务分组没有创建。
   - 安装 EDAS Agent 使用的 AK，SK 错误，或者安装时使用 Web 页面终端安装导致 AK，SK 错误。
     1. 执行 cat /home/admin/.spas_key/default；
     2. 登录 EDAS 控制台，在左侧菜单选择**资源管理>云服务器 ECS**，点击**安装 Agent**按钮；
     3. 比较弹出的界面中 AK，SK 是否与安装 EDAS Agent 所使用的 AK, SK 一致（注意大小写，Web 终端会导致大小写不一致）。
     4. 服务提供者地址 ping 不通，由于多网卡，将一个客户端无法访问的 IP 发布出来。可以使用 -Dhsf.server.ip 来指定服务端的 IP 。
5. 服务调用太快，在 configserver 将地址推送过来之前就发起调用导致出错。在服务消费者的配置项里添加 maxWaitTimeForCsAddress 的相关配置（参见《开发者指南》）。
6. 数据推送异常，此问题请联系开发人员。

错误编码：**HSF-0002**

报错信息：

消费端报错：HSFTimeOutException

解决方案：

- 机器的网络是否健康。ping 一下服务端的 IP，查看是否连通。

- 服务端处理耗时>3s，在服务端的 hsf.log 里查找业务执行超时的日志，定位具体的类及方法：
  - 服务端出现序列化错误。请检查代码：流类型、File、超大的对象等都会导致序列化错误，请不要传递。
  - 代码性能较低，请进行代码优化。
  - 服务端逻辑复杂，必须处理时间>3s，则可以修改超时时间（参见《开发者指南》）。
- 偶然出现超时，两端出现 GC 。检查服务端和客户端 GC 日志，耗时很长的 GC，会导致超时。网络搜索 **Java gc 优化**。
- 客户端负荷很高，导致客户端请求发送失败，造成超时。增加客户端机器。

错误编码：**HSF-0003**

报错信息：

消费端报错：java.io.FileNotFoundException: /home/admin/logs/hsf.log （系统找不到指定的路径）。

解决方案：由于默认的 HSF 日志路径找不到或没权限，通过在启动时加 -DHSF.LOG.PATH=xxx 来修改默认的路径

错误编码：**HSF-0005**

报错信息：

启动时报错：

java.lang.IllegalArgumentException: 未配置需要发布为服务的 Object，服务名为：com.taobao.hsf.jar.test.HelloWorldService:1.0.zhouli

解决方案：

provider 的 bean 缺少 target 属性，请检查配置文件。

target 指定服务的实现类不存在，请检查配置文件。

错误编码：**HSF-0007**

报错信息：

启动时报错：java.lang.IllegalArgumentException: unsurpport serialize type

解决方案：provider 的bean 错误配置了serializeType 属性或者 preferSerializeType 属性，请检查配置文件。建议使用 hessian，hessian2。

错误编码：**HSF-0008**

报错信息：java.lang.IllegalArgumentException: ProviderBean 中指定的服务类型不是接口[com.taobao.hsf.jar.test.HelloWorldServiceImpl]。

解决方案：provider 的 bean 配置的 serviceInterface 不是接口，需要配置成接口名，请检查配置文件。

错误编码：**HSF-0009**

报错信息：java.lang.IllegalArgumentException: 真实的服务对象[com.taobao.hsf.jar.test.HelloWorldServiceImpl@10f0a3e8] 没有实现指定接口[com.taobao.hsf.jar.test.HelloWorldService]。

解决方案：provider 的 target 指定的 bean 没有实现接口，请确定实现类实现了对应接口。

错误编码：**HSF-0014**

报错信息：java.lang.IllegalArgumentException: ProviderBean 中指定的接口类不存在[com.taobao.hsf.jar.test.HelloWorldService1]。

解决方案：provider 的 serviceInterface 属性配置错误，指定的接口不存在。

错误编码：**HSF-0016**

报错信息：

启动时报错：启动 HSF SERVER 失败。

解决方案：

- 检查12200端口是否有冲突，一般 Server Bind 失败会造成启动失败。
- 多网卡且存在外网 IP 机器，通过-Dhsf.server.ip 来指定本地 IP。

错误编码：**HSF-0017**

报错信息：

启动时报错：java.lang.RuntimeException: [ThreadPool Manager] Thread pool allocated failed for service [com.taobao.hsf.jar.test.HelloWorldService:1.0.zhouli]: balance [600] require [800]

解决方案：分配的线程池不够用，HSF 默认的最大线程池是 600，可以通过配置 jvm 参数 -Dhsf.server.max.poolsize=xxx 来修改默认的全局最大线程池大小。

错误编码：**HSF-0020**

报错信息：

WARN taobao.hsf - HSF服务：com.taobao.hsf.jar.test.HelloWorldService:1.0.zhouli 重复初始化！

解决方案：HSF 在一个进程中，由服务名和版本号唯一区分一个服务。不支持同一个进程发布或订阅同个服务名、同个版本号，但不同组别的服务，请检查配置文件。例如配置文件中同时存在：

com.taobao.hsf.jar.test.HelloWorldService 1.0 groupA

com.taobao.hsf.jar.test.HelloWorldService 1.0 groupB

错误编码：**HSF-0021**

报错信息：

启动时报错：

java.lang.IllegalArgumentException: ProviderBean 中指定的接口类不存在[com.taobao.hsf.jar.test.HelloWorldService1]。

java.lang.IllegalArgumentException: ConsumerBean中指定的接口类不存在[com.taobao.hsf.jar.test.HelloWorldService1]。

解决方案：HSFSpringProviderBean 的 serviceInterface 属性配置错误，指定的接口确实不存在（HSF-0014），或者 HSFSpringConsumerBean 中字段 interfaceName 指定的接口不存在（HSF-0021）。

错误编码：**HSF-0027**

报错信息：[HSF-Provider] HSF thread pool is full

解决方案：

服务器端某个服务处理速度过慢，不能及时处理客户端的请求，导致服务端业务执行的线程池达到最大值。HSF 默认会 dump 文件：/home/admin/logs/hsf/HSF_JStack.log （默认路径），查看此文件的 HSFBizProcessor-xxx 线程堆栈信息，分析性能瓶颈。

HSF 默认的最大线程数是600，如果确实要增大这个数，配置 JVM 参数 -Dhsf.server.max.poolsize=xxx 进行修改。

错误编码：**HSF-0030**

报错信息：[HSF-Provider] 未找到需要调用的方法。

解决方案：

- 服务端未提供此方法，登录 EDAS 控制台，在左侧菜单选择**应用管理>点击服务提供者应用名称**，跳转到应用详情页面，点击左侧菜单**服务列表**查看对应的服务是否已经发布成功。
- 新旧版本并存，调用到错误版本的服务。同上查看。
- 服务方和消费方接口不一致，例如：服务方的参数是 java.lang.Double，而消费方调用时使用double。
- 服务端\客户端加载了不一致的接口类，请比较服务端和客户端接口JAR 包的 MD5 是否一致。

错误编码：**HSF-0031**

报错信息：[HSF-Provider] 执行 HSF 服务xxx的方法xxx耗时xxxms，接近超时时间。

解决方案：服务端会在**超时时间-实际耗时<100ms**时，打印此日志，默认超时时间为3s。

- 如果超时时间很短，比如小于100ms，每次调用都会打印这个日志，请忽略这个日志。
- 如果超时时间已经很长，打印此日志，则说明业务执行慢，需要分析业务执行的性能瓶颈。

错误编码：**HSF-0032**

报错信息：please check log on server side that unknown server error happens。

解决方案：服务端在处理请求时，发生了未捕获的异常，请查看服务端的 hsf.log 日志解决。

错误编码：**HSF-0033**

报错信息：Serilization error during serialize response。

解决方案：

服务端在序列化返回数据时发生异常，请查看服务端的 hsf.log 日志解决。

如果包含“must implement java.io.Serializable”字样，请在 DO 上实现一下可序列化的接口。

错误编码：**HSF-0038**

报错信息：服务器在多网卡的场景下，HSF服务端绑定在了错误的 IP 上。

解决方案：JVM 启动参数添加 -Dhsf.server.ip=xxx.xxx.xx.xx 指定到期望的 IP 地址。

错误编码：**HSF-0035**

报错信息：RPCProtocolTemplateComponent invalid address。

解决方案：当前机器与对应的地址无法建立 TCP 连接，查看对应远端地址及端口是否可以连通。

21.9.1.5 其他开发问题

- 问：开发框架未使用 Spring 如何进行开发？

  回答：我们建议使用 Spring 进行 HSF 的相关开发，如果使用其他框架，可以使用 lightApi 的方式进行开发，相关内容请参见《开发者指南》。

- 问：开发环境能直接访问生产环境的服务吗？

  回答：不能，生产环境是安全隔离的。

- 问：EDAS 提供 Open API 吗？都有什么功能？

  EDAS 提供 Open API，可以提供资源查询，应用生命周期管理和账号管理功能。

- 问：EDAS 除了支持 Java 还支持别的语言吗？

  回答：HSF 默认是基于 Java 开发，同时 HSF 也提供 C++ 版本的客户端、PHP 版本的客户端，可以访问 Java 提供的后端 HSF 服务。

21.9.2 使用常见问题

开发常见问题包括账号问题、资源问题、应用生命周期和监控告警。

21.9.2.1 账号管理类

- 问：是否允许创建多个子账号？

  回答：可以。需要申请阿里云账号作为 RAM 主账号（即付费账号），然后 RAM 主账号可以创建多个 RAM 子账号。

- 问：谁能对子账号进行应用的授权？

  回答： EDAS只支持由主账号对子账号进行应用的授权。

21.9.2.2 资源管理类

- 问：Agent 安装之后没有上报，界面未显示 Agent 的版本号。

  回答：请按以下操作排除问题：

  1. 登录到 ECS，查看*/home/admin/edas-agent/logs/agent.log* 内容，如果出现 UnAuthorizedException，请确认如下问题：

     - 安装 EDAS Agent 使用的 AK，SK 错误，或者安装时使用 Web 页面终端安装导致 AK，SK 错误。

       1. 执行 `cat /home/admin/.spas_key/default`；

2. 登录 EDAS 控制台，在左侧菜单选择**资源管理>云服务器 ECS**，点击**安装 Agent** 按钮；
3. 比较弹出的界面中 ak，sk 是否与上述值一致（注意大小写， Web 终端会导致大小写不一致）。

- 安装时使用的地区脚本不正确。

2. 查看*/home/admin/edas-agent/logs/std.log*，如果出现 Java not found 或其他异常，请执行 java —version 确认是否是 1.7 版本。如果是Java1.5，请先使用rpm -e 对应安装rpm名称 命令移除之后，再重新安装Agent。

- 问：Agent 安装后控制状态“未知”或“异常”

  回答：请查看 ECS 上 */home/admin/edas-agent/logs* 目录下的 std.log 和 agent.log 日志文件：

  - std.log 是 Agent 安装过程中的日志
  - agent.log 是 Agent 的运行日志

  根据错误排查问题，一般有如下原因：

  - 如果日志中有“*Permission denied*”或“*Not such file*”字样，这可能是没有相关文件目录权限导致，需要确认 admin 用户是否具有 /home/admin目录下所有文件的权限，然后重新安装 agent。
  - 检查 ECS 主机名和 /etc/hosts 文件中的是否一致，如果不一致，修改后重启 agent 即可：

    ```
    /home/admin/edas-agent/bin/shutdown.sh
    /home/admin/edas-agent/bin/startup.sh
    ```

- 问：EDAS 的 Java 版本是哪个版本，我能够选择吗？

  回答：EDAS 提供7，8 两个版本的 Java，默认使用的是 Java 7，用户也可以在安装 EDAS Agent 过程中，进行 Java 版本的选择。具体用法如下：

  ```
  install.sh -ak -sk [-java <7(default)|8>]
  ```

- 问：什么是 Docker 实例？和 ECS 独占实例有何区别？

  回答：EDAS 的应用部署有两种类型：

  - **ECS 独占实例**

    在一台独立的 ECS 机器上，仅允许部署单独一个应用。

  - **Docker 实例**

    使用 Docker 技术，EDAS 实现了在单个 ECS 实例上部署多个应用。在一台 ECS 机器上创建多个 Docker 实例，每一个 Docker 实例中可部署一个应用。



注意: 单个应用在同一 ECS 上只能部署一个实例。

这对商业化计费有何影响？

目前 EDAS Docker 化处于公测期，在应用实例数统计上，对通过 Docker 实例部署所占用的节点暂不收取费用。

- 问：如果 Agent 的心跳进程停了，会有什么样的后果？

  回答：如果相应的 ECS 上没有安装应用程序，则不会有任何的影响。如果已经安装了应用，则在应用的机器列表中（在 **应用管理** 中选择对应的应用进入**基本信息**之后，屏幕下方会列出该应用下的所部署的 ECS 列表），相应的 ECS 的 实时状态 会变成 **AGENT 异常** ；且此时，任何针对该 ECS 的部署、启动、停止等命令均是无效的。

  登录到该机器执行`sudo -u admin /home/admin/edas-agent/bin/startup.sh` 将 Agent 启动。排查 Agent 异常退出的原因：

  查看*/home/admin/edas-agent/logs/agent.log*中是否有错误信息。

  确定系统内存是否足够，如果系统内存紧张，会导致系统 oom **kill**。具体请搜索：linux oom **kill**。如果出现 oom **kill**，建议查看系统内存使用情况，调整内存分配。

- 问：如果 Ali-Tomcat 突然容器退出，该如何处理？

  回答：登录 EDAS 控制台启动对应的应用。排查Ali-Tomcat异常退出的原因：

  - 查看*/home/admin/tomcat#安装目录#/logs/catalina.out* 中是否有错误信息。
  - 确定系统内存是否足够，如果系统内存紧张，会导致系统 oom kill。具体请搜索：linux oom kill。如果出现 oom kill，建议查看系统内存使用情况，调整内存分配策略。

- 问：系统重启后，EDAS Agent 未自动启动。

  回答：目前 CentOS 6.5版本可以自动启动，其他系统暂未测试。如未启动，执行以下程序：

  ```
  sudo -u admin /home/admin/edas-agent/bin/startup.sh
  /usr/alisys/dragoon/bin/DragoonAgent
  ```

21.9.2.3 应用生命周期类

- 问：重启服务器后 EDAS Agent 是否能自动重启？

  回答：是的，重启服务器后 EDAS Agent 会自动重启，但是您的 Tomcat 并不会自动重启。

- 问：EDAS Agent 为什么无法启动？

回答：首先在服务器上使用 ping 命令检查 EDAS Console服务器是否可达：

```
ping edas-internal.console.aliyun.com
```

其次请检查您的安全令牌文件是否被正确设置：

```
cat /home/admin/.spas_key/default
```

- 问：现在 EDAS 是否支持在同一台服务器上安装多个应用？

  回答：在 EDAS 产品中，应用的部署有两种类型：

  - ECS 独占实例：在一台独立的 ECS 机器上，仅允许部署单独一个应用。
  - Docker 实例：EDAS使用 Docker 技术，在一台独立的 ECS 机器上创建多个 Docker 实例，允许在每一个 Docker 实例上部署一个应用。

- 问：应用部署时提供的发布包 URL 是不是可以随意设置？

  回答：应用部署包的 URL 必须保证您的服务器是可以下载的。

- 问：为什么应用操作（如启动应用、停止应用、部署应用等）会失败？

  回答：请在出现失败情况的服务器上检查 EDAS Agent 是否正常运行，通常而言，应用操作失败都是由于 Agent 没有正常运行导致。

- 问：在创建应用的时候，为什么我名下的服务器没有出现在服务器选择对话框中？

  回答：请检查在服务器上是否已正确安装 EDAS Agent，请参考资源管理->ECS列表->安装Agent 中的步骤完成 Agent 安装。

  注意: 安装 Agent 时一定要选择正确的区域（Region）。

- 问：为什么在 EDAS Console上面机器实例的状态是“未知”？

  回答：EDAS Agent会定期汇报心跳数据给EDAS Console，如果Agent停止汇报状态，则某段时间后该机器将会被判定为未知状态。通常而言，该问题是由于Agent停止导致。

- 问：服务可以正常调用，但是服务列表不显示。

  回答：接口 API 中存在泛型，但是泛型没有指定具体的类型，导致解析服务列表失败，请修改对应的代码。

- 问：服务列表显示正常，但是调用失败怎么办？

  回答：

  1. 查看服务提供者对应的分组是否已经创建，如果没有创建则会鉴权失败。

2. 需要查看 */home/admin/logs/hsf/hsf.log*，确认具体的错误码，根据错误码查询*HSF常见问题*。

- 问：删除应用后是否可以恢复？

  回答：不可以，删除应用操作不可逆，所有的数据都会被清除。

- 问：如何进行分批发布或者beta发布

  回答：

  - 如果一个应用有多台机器，可以选择分批发布，设定分批的批次>1即可。
  - 如果一个应用有多台机器，可以设定其中几台为 beta 机器，发布时可以针对 beta 机器单独进行发布。发布时仅会更新标记为beta的机器，其他机器不做更改。

- 问：我的应用部署在多个机器上之后，集群中会话如何共享呢？

  回答：目前 EDAS 尚未提供分布式 Session 管理的功能，用户可以借助一些分布式缓存系统（比如：OCS，Redis 等）来进行管理。

- 问：健康检查 URL 如何设置？

  回答：应用发布时会自动将您提供的 WAR 文件部署到 Tomcat 目录，所以缺省情况下健康检查 URL 需要附加 WAR包名称，而且 WAR 包中的文件必须返回200-400 HTTP 代码。比如，如果您的 WAR 包名称是 *order.war,* 其中包含一个文件 *index.jsp*，则健康检查的 URL 可以设置为http://127.0.0.1:8080/order/index.jsp。

- 问：我的应用部署在多个机器上之后，如何实现负载均衡，可以使用 SLB 吗？

  回答：

  1. EDAS 中提供 HTTP 服务的 Web 应用，需要使用 SLB 来进行负载均衡，用户可以在 EDAS 的应用配置页面，进行 SLB 的配置。
  2. 对于 EDAS 的 RPC 服务提供者对应的应用，用户无需考虑负载均衡问题，EDAS 原生支持对 RPC 服务提供者的负载均衡。

- 问：什么是 Docker 实例？和 ECS 独占实例有何区别？对商业化计费有何影响？

  回答：在 EDAS 产品中，应用的部署有两种类型：

  - ECS 独占实例：在一台独立的 ECS 机器上，仅允许部署单独一个应用。
  - Docker 实例：EDAS 使用 Docker 技术，在一台独立的 ECS 机器上创建多个 Docker 实例，允许在每一个 Docker 实例上部署一个应用。

  这对商业化计费有何影响？

目前 EDAS Docker 化处于公测期，在应用实例数统计上，暂不收取通过 Docker 实例部署所占用的节点。

21.9.2.4 监控报警类

- 问：基础监控可以提供应用维度的数据吗？

  回答：基础监控可以提供以应用为维度的数据，可以汇总应用所在 ECS 的指标，提供给用户一个聚合结果。同时可以查看单机指标。

- 问：基础监控可以提供什么监控指标？

  回答：基础监控可以提供的监控指标包括机器的 CPU、内存、网络、磁盘空间占用、磁盘读写速度、磁盘 IOPS。

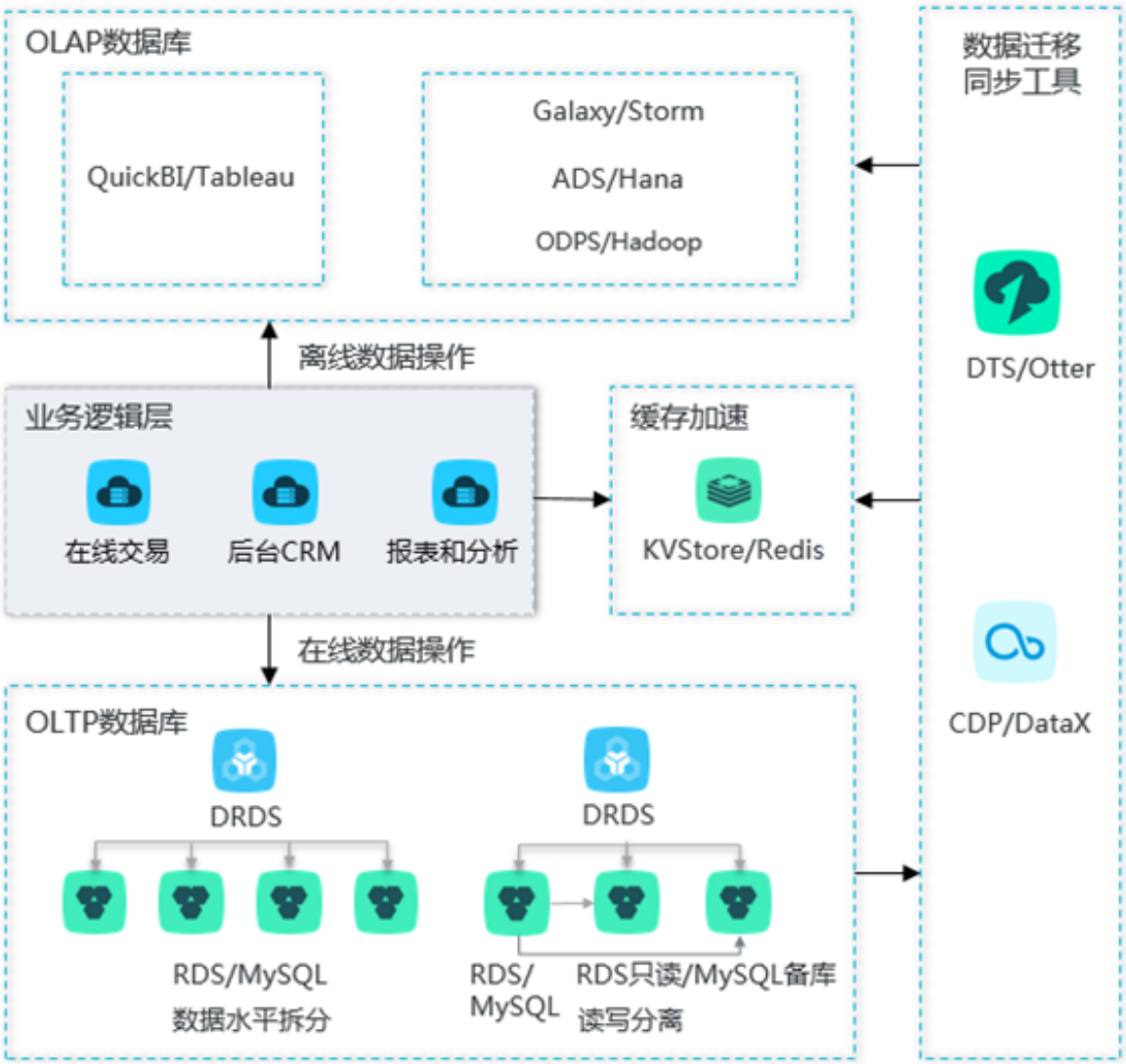
22 分布式关系型数据库DRDS

22.1 产品概述

分布式关系型数据库服务（Distributed Relational Database Service ，简称DRDS）是阿里巴巴集团自主研发的中间件产品，专注于解决单机关系型数据库扩展性问题。DRDS兼容MySQL协议和语法，支持分库分表、平滑扩容、服务升降配、透明读写分离和分布式事务等特性。产品具备轻量(无状态)、灵活、稳定、高效等特点，可以为您提供分布式数据库全生命周期的运维管控能力。

DRDS主要应用场景在大规模在线数据操作上。通过贴合业务的拆分方式，将操作效率提升到极致，有效满足在线业务对关系性数据库要求。

图 543: DRDS产品结构示意图



解决的问题

- **单机数据库容量瓶颈**： 随着数据量和访问量的增长，传统单机数据库会遇到很大的挑战，依赖硬件升级并不能完全解决问题。
- **单机数据库扩展困难**：传统单机数据库容量扩展往往意味着服务中断，很难做到业务无感知或者少感知。
- **单机数据库使用成本高**： 当业务数据和访问量增加到一定量时，单机数据库需要依赖特定的高端存储和小型机设备，成本曲线快速上升。

22.2 管理DRDS实例

DRDS实例的生命周期管理主要包括创建实例、变更实例配置、释放实例等操作。

DRDS实例有以下几个状态：

- 运行中
- 升配中
- 降配中
- 锁定中，一般为欠费
- 已释放

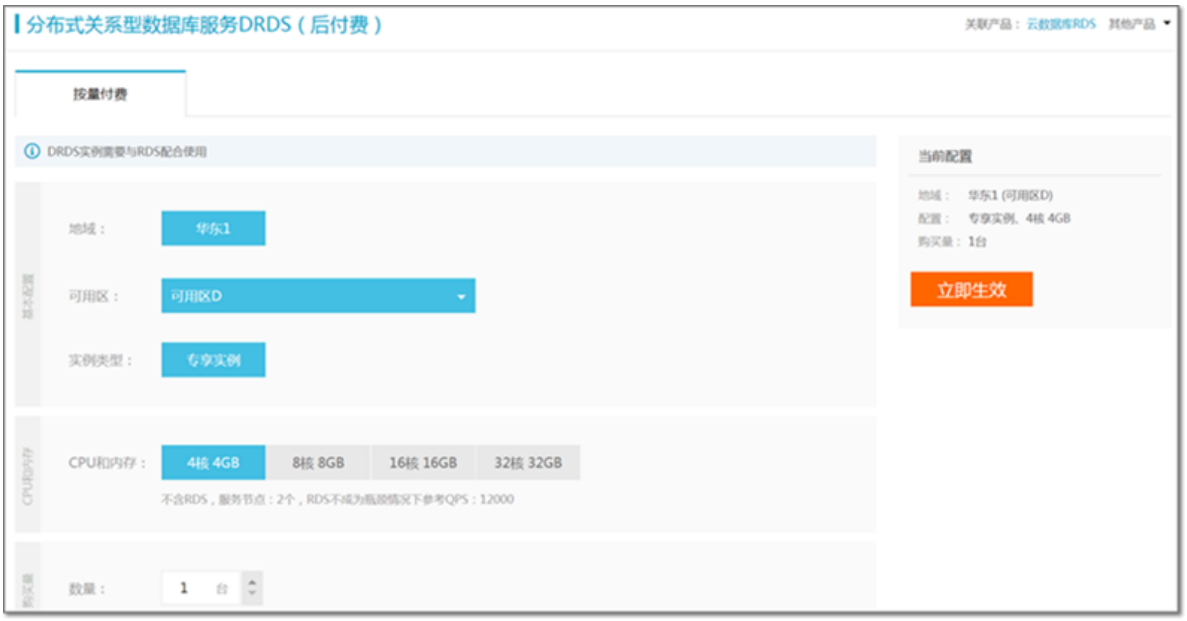
22.2.1 创建实例

使用DRDS的第一步即创建实例。

操作步骤

1. 登录DRDS控制台，单击右上角**创建实例**。
2. 在**创建实例**页面，选择实例所在的域、实例类型以及规格等，单击**立即生效**，进入订单确认页面。

图 544: 创建实例



实例创建成功后，在 DRDS Web 控制台的实例列表即能看见所创建的实例。实例名称是实例的唯一标识，可以通过该标识唯一确定一个DRDS实例。

图 545: 实例列表



22.2.2 变更配置

在使用DRDS的过程中，如有需要可以对DRDS实例规格进行变更。

操作步骤

1. 在DRDS Web控制台的实例列表中选择对应实例，单击右侧的**变更配置**按钮。
2. 在实例变更页面，选择所需规格，单击**确定变更**。

图 546: 实例变更



实例配置变更成功后，在DRDS Web控制台的实例列表中会显示出变更后的实例规格。

22.2.3 释放实例

如需释放某个实例，同样可以在控制台操作。

前提条件

释放实例前请确保已经删除了该实例下的所有数据库。

操作步骤

在DRDS控制台的实例列表页，单击右侧的**释放**按钮进行释放。

注意: 实例释放后不可恢复，请谨慎操作。

22.2.4 实例监控

本文介绍 DRDS 实例的性能监控功能，如何分析性能指标并根据指标排查 DRDS 性能问题。

监控指标技术说明

DRDS 在控制台上提供了丰富的监控项，从指标类型上主要分为资源监控和 DRDS 引擎监控两类指标。引擎层面的监控指标又可以分成 DRDS 实例级别和 DRDS 库级别两个维度的监控，当某些引擎类的监控指标出现异常的时候，可以直接查看各个数据库的监控指标，从而定位到有性能问题的数据库。

目前 DRDS 提供的监控指标如下表：

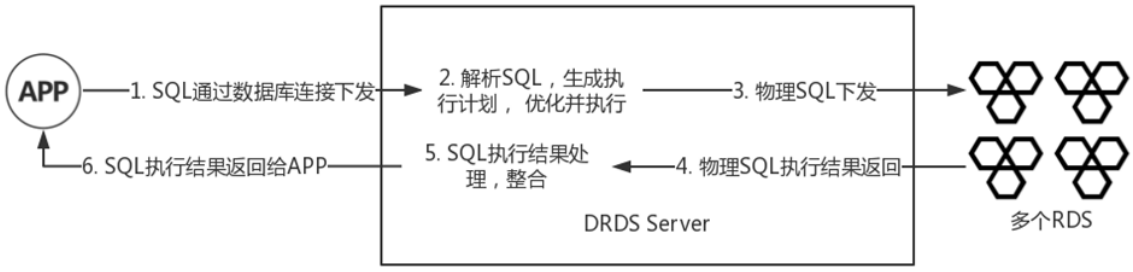
监控项	监控项分类	含义	数据采集周期	数据保留时长	说明
CPU 利用率	资源监控	DRDS 服务节点的 CPU 利用率的平均值	5分钟	3天	
网络输入流量	资源监控	DRDS 服务节点的网络输入流量的总和	5分钟	3天	RDS 返回数据到 DRDS，会产生网络输入流量
网络输出流量	资源监控	DRDS 服务节点的网络输出流量的总和	5分钟	3天	DRDS 发送物理 SQL 到 RDS, DRDS 返回数据到应用，均会产生网络输出流量
逻辑 QPS	引擎监控	DRDS 服务节点每秒处理的 SQL 语句数目的总和	5分钟	7天	
物理 QPS	引擎监控	DRDS 服务节点每秒发送到 RDS 的 SQL 操作数总和	5分钟	7天	一条逻辑 SQL 可能会拆分成多条物理 SQL
逻辑 RT	引擎监控	DRDS 对于每条 SQL 的平均响应时间	5分钟	7天	如果逻辑 SQL 会变成物理 SQL 下发，那么此条 SQL 的逻辑 RT 会包含物理 SQL 的 RT
物理 RT	引擎监控	DRDS 发送到 RDS 的 SQL 的平均响应时间	5分钟	7天	
连接数	引擎监控	应用到 DRDS 的连接总数	5分钟	7天	不包括 DRDS 到 RDS 的连接

监控项	监控项分类	含义	数据采集周期	数据保留时长	说明
活跃线程数	引擎监控	DRDS 用来执行 SQL 的线程数	5分钟	7天	

监控指标原理

在分析监控指标之前，需要对 SQL 语句在 DRDS 上的执行流程进行了解。

图 547: DRDS SQL执行流程



在整个 SQL 的执行链路中，第2到第4步的执行状况都会在 DRDS 的各个监控指标上有所体现。

- **第2步**：SQL 解析、优化、执行，主要消耗的是 CPU 资源。越是复杂的 SQL（结构复杂或者超长），消耗的 CPU 资源就越大。通过 **TRACE** 指令跟踪SQL的执行过程，可以看到一条 SQL 在优化 阶段的耗时。这部分的耗时越高，表示 CPU 资源消耗的就越高。
- **第3步**：物理 SQL 下发和执行，主要消耗的是 IO 资源，可以通过逻辑、物理的 QPS 和 RT 等指标分析出这一部分的执行状况。例如，如果物理 QPS 很低同时 物理 RT 很高，表示当前 RDS 处理 SQL 很慢，需要关注 RDS 的性能。
- **第5步**：SQL 执行结果处理、整合，这部分操作主要是用来对物理 SQL 的执行结果进行转换。多数情况下，此类转换只会进行一些 SQL 元信息的转换，资源消耗很小。但是当出现 heap sort等执行步骤的时候，则会消耗非常高的 CPU 资源。关于如何确定 SQL 在此阶段的消耗，可以参考如何排查 *DRDS* 慢 *SQL*文档中关于**TRACE**指令的说明。

22.2.4.1 预防性能问题

对于常见的数据库性能问题，结合性能监控指标，可以得到比较有效的处理方法。

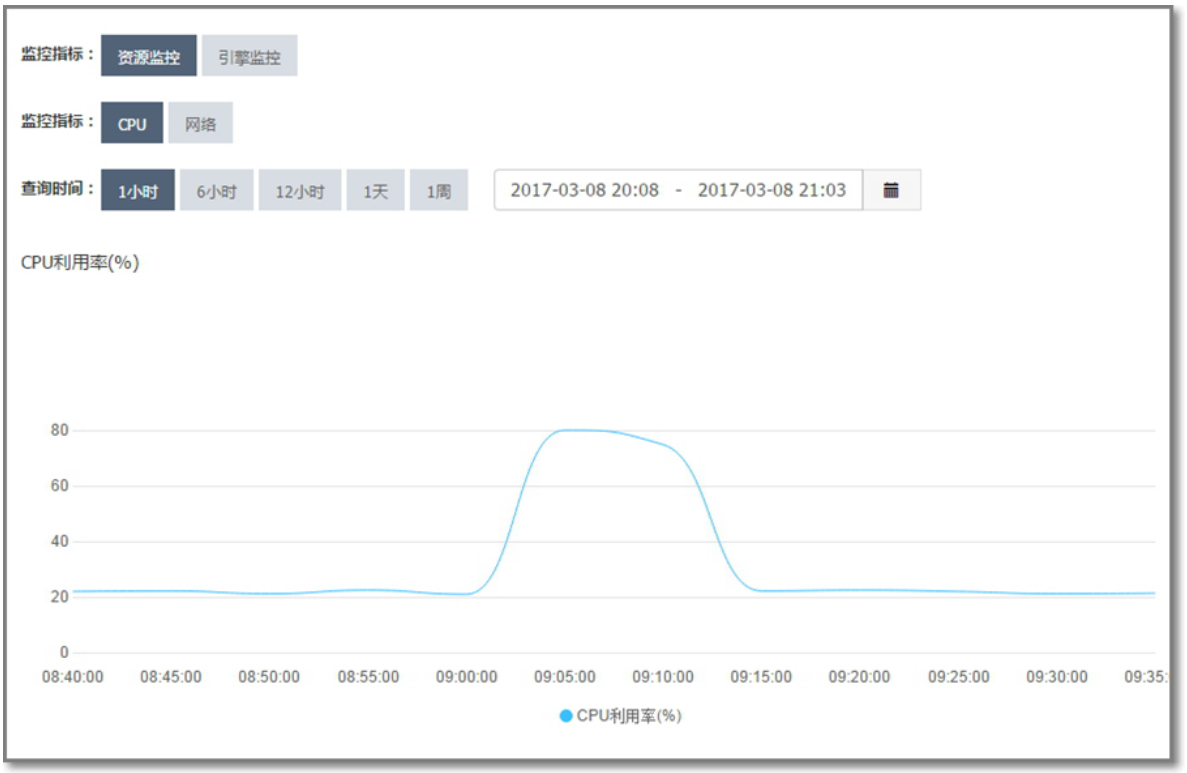
22.2.4.2 示例一

性能指标往往会随着当前系统业务量的波动而波动。

以下是常见的两种情况：

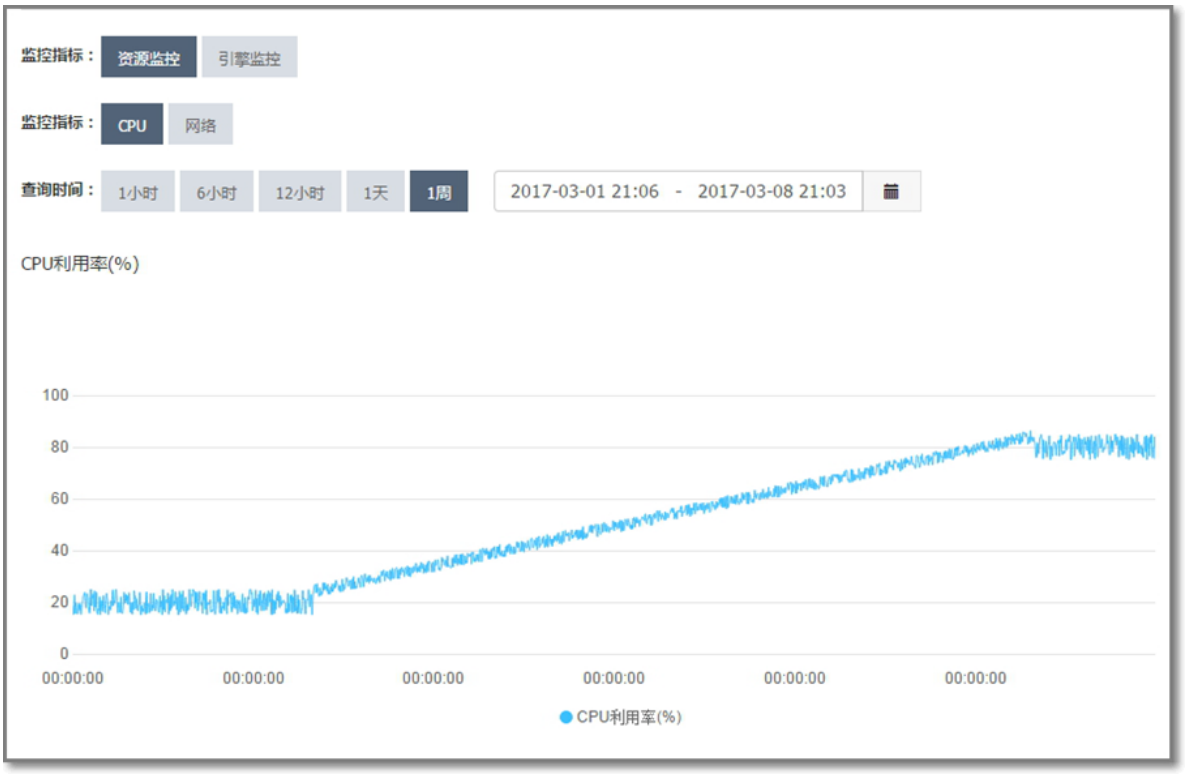
- 某应用在每天早上9点会有抢购活动，所以这个时间点整个系统的业务流量会有一个明显的增长。从监控数据上来看，DRDS 的 CPU 利用率从9点开始从20%增长到80%左右，整个高峰持续10分钟左右。

图 548: CPU利用率-1



- 某应用业务一直在增长，系统的业务流量也随着一直增长，直到稳定到一个水平线。DRDS 的 CPU 监控数据也基本上反应了这一变化。

图 549: CPU利用率-2



在 DRDS 的压力随着业务变化而变化的时候，应该密切关注监控指标的变化。如果超过预设的阈值，则**应该通过进行 DRDS 升配缓解性能压力。**

在 DRDS 控制台可以针对实例设置报警规则，当 CPU 的平均值超过预设的阈值的时候，系统会方发送短信到对应的联系人。CPU 的阈值可以根据实际情况设定，推荐设置成80%。

22.2.4.3 示例二

观察逻辑 RT 和物理 RT 的差值。

逻辑 RT 指的是 DRDS 从收到逻辑 SQL 到返回数据给应用的响应时间，物理 RT 指的是 DRDS 从发出物理 SQL 到 RDS 至收到 RDS 返回数据的时间。

如果一条逻辑 SQL 被拆分成了一条或者多条物理 SQL 那么逻辑 RT 会大于等于物理 RT。理想情况下，DRDS 只会对于 RDS 返回过来的数据进行少量的操作，所以一般情况下逻辑 RT 只会略高于物理 RT 一点。但是某些情况下，物理的 SQL 执行很快，而逻辑 SQL 处理时间很久，则逻辑 RT 和物理 RT 会分别出现如下的走向：

图 550: 逻辑RT

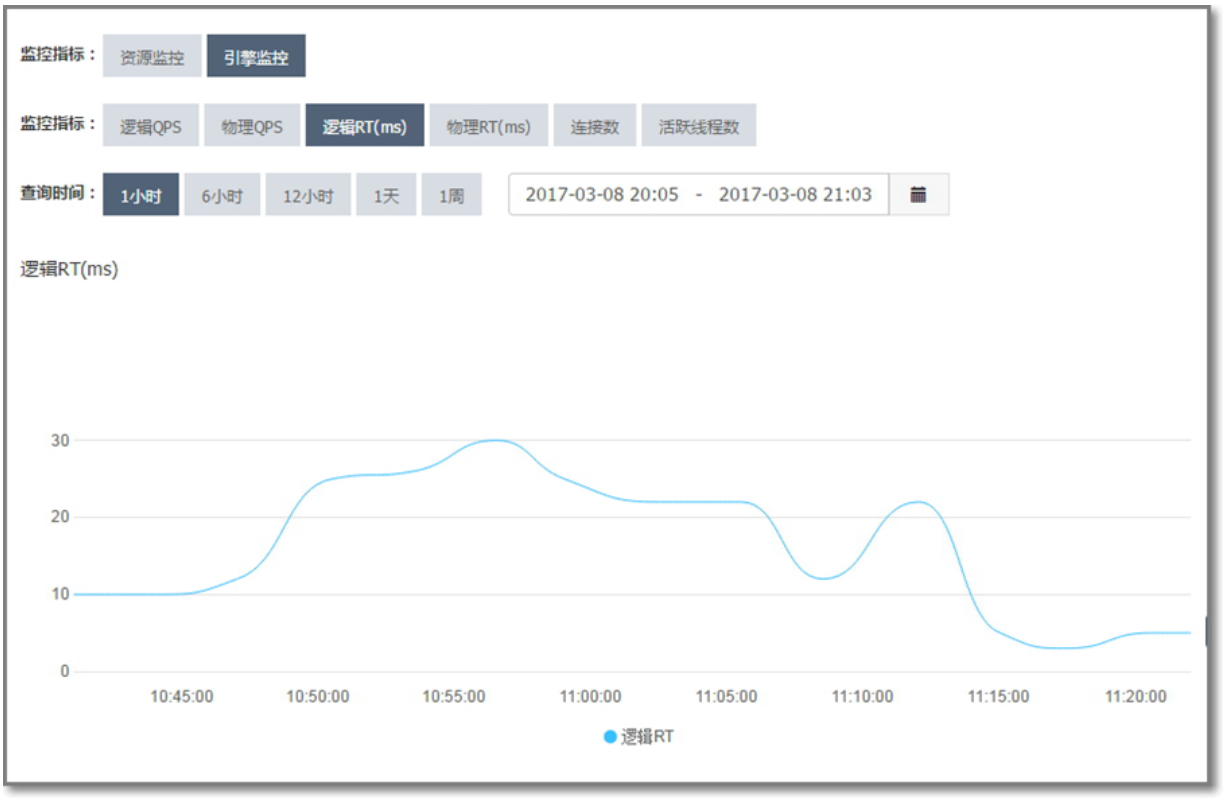

监控指标：
资源监控
引擎监控
监控指标：
逻辑QPS
物理QPS
逻辑RT(ms)
物理RT(ms)
连接数
活跃线程数
查询时间：
1小时
6小时
12小时
1天
1周
2017-03-08 20:05 - 2017-03-08 21:03
逻辑RT(ms)
30
20
10
0
10:45:00
10:50:00
10:55:00
11:00:00
11:05:00
11:10:00
11:15:00
11:20:00
逻辑RT

图 551: 物理RT


监控指标：
资源监控
引擎监控
监控指标：
逻辑QPS
物理QPS
逻辑RT(ms)
物理RT(ms)
连接数
活跃线程数
查询时间：
1小时
6小时
12小时
1天
1周
2017-03-08 20:03 - 2017-03-08 21:03
物理RT(ms)
2017-03-08 10:47:03
物理RT 2.6
6
4
2
0
10:45:00
10:50:00
10:55:00
11:00:00
11:05:00
11:10:00
11:15:00
11:20:00
物理RT

从上面两个监控图可以看到，逻辑 RT 和物理 RT 的变化趋势大致一样，逻辑 RT 在10毫秒到20毫秒范围内波动，物理 RT 在2毫秒到5毫秒范围内波动。这说明 DRDS 层面已经有较大的压力。这种情况可以通过升级 DRDS 配置来解决性能问题。相反，如果逻辑 RT 和 物理 RT 都很高，那么可以通过升级 RDS 的配置或者在 RDS 层面优化 SQL 来解决性能问题。

22.2.4.4 示例三

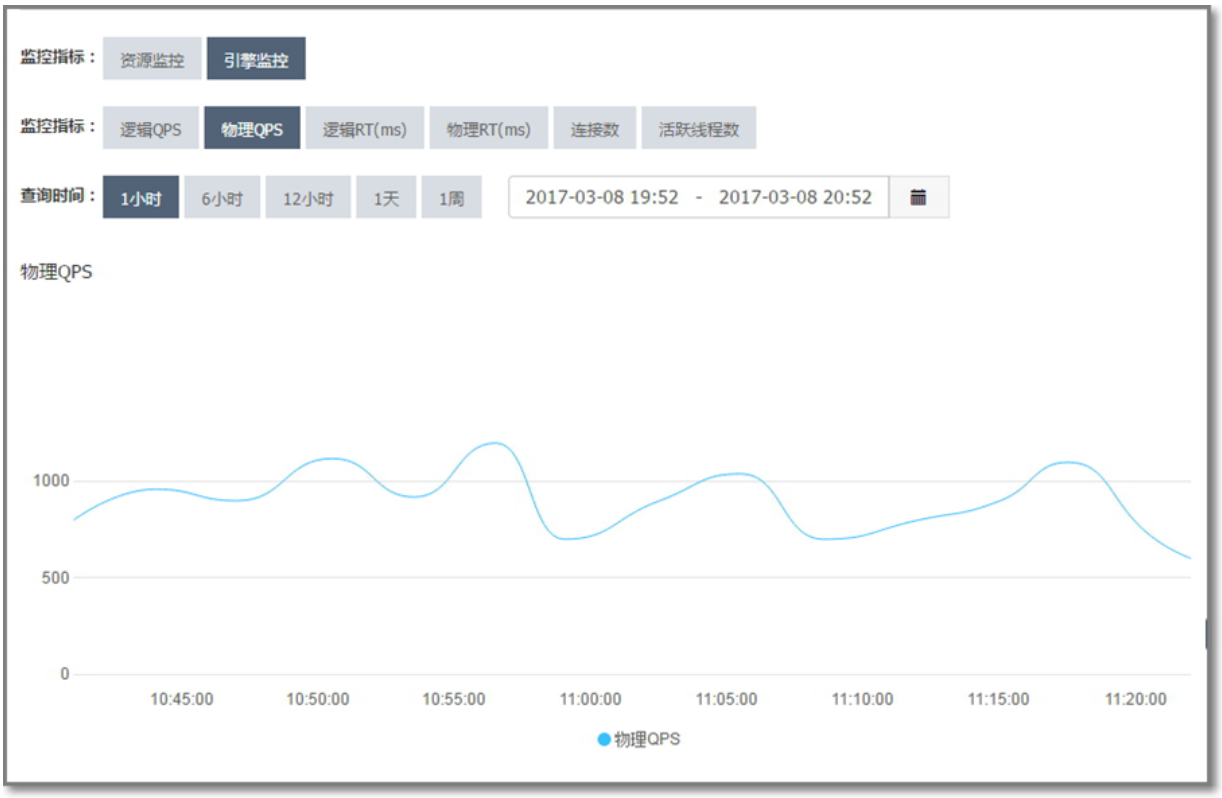
观察逻辑 QPS 和物理 QPS 的差值。

从监控数据上来看，逻辑 QPS 和物理 QPS 的趋势相同，但是两者有差值比较大，且成一定的比例。

图 552: 逻辑QPS



图 553: 物理QPS



根据监控指标显示，逻辑 QPS 在80至150这个范围内波动，物理 QPS 在700至1200这个范围内波动。

造成这种现象的原因是DRDS 会根据逻辑 SQL 生成物理 SQL，逻辑 SQL 和物理 SQL 的数量比例并不是1：1的关系。例如有一个 DRDS 逻辑表，建表语句如下：

```
CREATE TABLE drds_user
(id int,
name varchar(30))
dbpartition by hash(id);
```

当查询条件带上分库键id，DRDS 会将这条逻辑 SQL 直接下推到 RDS 去执行，从执行计划上可知，物理 SQL 的数目是1条：

```
mysql> explain select name from drds_user where id = 1;
+-------------------------+-----------------------------------------------------------------------------+--------+
| GROUP_NAME              | SQL                                                                         | PARAMS |
+-------------------------+-----------------------------------------------------------------------------+--------+
|         SANGUAN_BSQT_0001_RDS | select `drds_user`.`name` from `drds_user` where (`drds_user`.`id` = 1) | {}     |
+-------------------------+-----------------------------------------------------------------------------+--------+
```

当查询没有带上分库键，DRDS 会将逻辑 SQL 拆分成多条条物理 SQL。从下面的执行计划可知，物理 SQL 的数目是8条。

```
mysql> explain select name from drds_user where name = 'LiLei';
+---------- --------------+-----------------------------------------------------------------------------+--------+
```

```
| GROUP_NAME              | SQL                                                
```

- 管理员账号的名字跟数据库名一致，比如数据库名是 easydb，管理员账号的名字就叫 easydb。
- 只读账号的名字是数据库名加上 _RO 后缀，比如数据库名是 easydb，只读账号的名字就叫 easydb_RO。

假设有两个数据库 dreamdb、andordb，根据上面的规则可知，库 dreamdb 下面包含管理员账号 dreamdb，只读账号 dreamdb_RO；库 andordb 下面包含管理员账号 andordb，只读账号 andordb_RO。

账号规则

- 管理员账号具有所有权限；
- 只有管理员账号具有创建账号和授权功能；其它账号只能由管理员账号创建，其权限只能由管理员账号授予；
- 管理员账号是跟数据库绑定的，没有其它数据库的权限，连接的时候只能连接自己对应的那个数据库，不能授予其它数据库的权限给某个账号。例如 easydb 这个管理员账号只能连接 easydb 数据库，只能授予 easydb 数据库权限或者 easydb 数据库中表的权限给某个账号；
- 只读账号只具有 SELECT 权限。

用户名规则

- 大小写不敏感；
- 长度必须大于等于4个字符，小于等于20个字符；
- 必须以字母开头；
- 字符可以包括大写字母、小写字母、数字。

密码规则

- 长度必须大于等于6个字符，小于等于20个字符；
- 字符可以包括大写字母、小写字母、数字、特殊字符（@#$%^&+=）。

HOST匹配规则

- HOST 必须是纯 IP 地址，可以包含 _ 和 % 通配符（ _ 代表一个字符，% 代表0个或多个字符）。含有通配符的 HOST 需要加上单引号，例如 lily@'30.9.%.%'，david@’ %’ ；
- 假设系统中有两个用户都符合当前欲登录的用户，则以最长前缀匹配（不包含通配符的最长 IP 段）的那个用户为准。例如系统有两个用户 david@'30.9.12_.234' 和 david@'30.9.1%.234'，在主机 30.9.127.234 上面登录 david，则使用的是 david@'30.9.12_.234'这个用户。

- 开启 VPC 时，主机的 IP 地址会发生变化。**为避免账号和权限系统中的配置无效，请将 HOST 配置为'%' 来匹配任意 IP。**

权限

权限级别支持情况

- 全局层级（暂不支持）
- 数据库层级（支持）
- 表层级（支持）
- 列层级（暂不支持）
- 子程序层级（暂不支持）

权限项

目前支持和表相关联的8个基本权限
项：**CREATE**、**DROP**、**ALTER**、**INDEX**、**INSERT**、**DELETE**、**UPDATE**、**SELECT**。

- **TRUNCATE** 操作需要有表上的 DROP 权限；
- **REPLACE** 操作需要有表上的 INSERT 和 DELETE 权限；
- **CREATE INDEX** 和 **DROP INDEX** 操作需要有表上的 INDEX 权限；
- **CREATE SEQUENCE** 需要有数据库级的创建表（CREATE）权限；
- **DROP SEQUENCE** 需要有数据库级的删除表（DROP）权限；
- **ALTER SEQUENCE** 需要有数据库级的更改表（ALTER）权限；
- **INSERT ON DUPLICATE UPDATE** 语句需要有表上的 INSERT 和 UPDATE 权限。

权限规则

- 权限是绑定到账号（username@’ host’ ），不是绑定到用户名（username）；
- 授权的时候会判断表是否存在，不存在则报错；
- 数据库权限级别从高到低依次是：全局级别权限（暂不支持）、数据库级别权限、表级别权限、列级别权限。高级别权限的授予会覆盖低级别权限，移除高级别权限的同时也会移除低级别权限；
- 不支持 USAGE 授权。

22.3.2 账号权限语句说明

下文介绍如何创建、删除账号，修改账号密码等语句。

创建账号语句（CREATE USER）

语法规则：

```
CREATE USER user_specification [, user_specification] ...
  user_specification: user [ auth_option ]
  auth_option: IDENTIFIED BY 'auth_string'
```

例如：

创建一个名为 lily，只能从 30.9.73.96 登录的账号，密码为123456。

```
CREATE USER lily@30.9.73.96 IDENTIFIED BY '123456';
```

创建一个名为 david，可以从任意主机登录的账号，密码为空。

```
CREATE USER david@'%';
```

删除账号语句（DROP USER）

语法规则：

```
DROP USER user [, user] ...
```

例如：

移除账号 lily@30.9.73.96。

```
DROP USER lily@30.9.73.96;
```

修改账号密码语句（SET PASSWORD）

语法规则：

```
  SET PASSWORD FOR user = password_option
  password_option: {
      PASSWORD('auth_string')
  }
```

例如：

修改账号 lily@30.9.73.96 的密码为123456。

```
SET PASSWORD FOR lily@30.9.73.96 = PASSWORD('123456')
```

授权权限语句（GRANT）

语法规则：

```
GRANT
    priv_type[, priv_type] ...
    ON priv_level
    TO user_specification [, user_specification] ...
    [WITH GRANT OPTION]
priv_level: {
  | db_name.*
  | db_name.tbl_name
  | tbl_name
}
user_specification:
    user [ auth_option ]
auth_option: {
    IDENTIFIED BY 'auth_string'
}
```



注意: GRANT 语句里面的账号如果不存在，同时又没有提供 IDENTIFIED BY 信息，则报账号不存在异常；如果提供了 IDENTIFIED BY 信息，则会创建该账号同时授权。

例如：

在数据库 easydb 下面，创建一个用户名为 david，可以在任意主机登录，具有 easydb 数据库所有权限的账号。

方法1：先创建账号，再授权。

```
CREATE USER david@'%' IDENTIFIED BY 'your_password';
```

```
GRANT ALL PRIVILEGES ON easydb.* to david@'%';
```

方法2：一条语句完成创建账号和授权两个操作。

```
GRANT ALL PRIVILEGES ON easydb.* to david@'%' IDENTIFIED BY 'your_password';
```

在数据库 easydb 下面，创建一个用户名为 hanson，可以在任意主机登录，具有 easydb.employees 表所有权限的账号。

```
GRANT ALL PRIVILEGES ON easydb.employees to hanson@'%' IDENTIFIED BY 'your_password';
```

在数据库 easydb 下面，创建一个用户名为 hanson，只能在 192.168.3.10 登录，具有 easydb.emp 表的 INSERT 和 SELECT 权限的账号。

```
GRANT INSERT,SELECT ON easydb.emp to hanson@'192.168.3.10' IDENTIFIED BY 'your_password';
```

在数据库 easydb 下面创建一个只读账号 actro，可以在任意主机登录。

```
GRANT SELECT ON easydb.* to actro@'%' IDENTIFIED BY 'your_password';
```

回收权限语句（REVOKE）

语法规则：

- 删除账号的权限：删除账号在某个权限级别下的权限项，具体权限级别由 priv_level 指定。

```
REVOKE
    priv_type
      [, priv_type] ...
    ON priv_level
    FROM user [, user] ...
```

- 删除账号的所有权限：删除账号在系统内（数据库级别和表级别的）的所有权限项。

```
REVOKE ALL PRIVILEGES, GRANT OPTION
    FROM user [, user] ...
```

例如：

删除 hanson@’ %’ 在 easydb.emp 表的 CREATE、DROP、INDEX 权限。

```
REVOKE CREATE,DROP,INDEX ON easydb.emp FROM hanson@'%';
```

删掉账号 lily@30.9.73.96 的所有权限。

```
REVOKE ALL PRIVILEGES,GRANT OPTION FROM lily@30.9.73.96;
```

注意: 为了兼容 MySQL，需同时写上 GRANT OPTION。

查看授权语句（SHOW GRANTS）

语法规则：

```
SHOW GRANTS[ FOR user@host];
```

查询所有的授权

```
SHOW GRANTS
```

查询针对某个账号的授权情况

```
SHOW GRANTS FOR user@host;
```

22.4 管理数据库

数据库管理包括创建数据库、数据库信息查询、读写分离、数据表管理、分库管理、白名单设置、数据导入、只读实例管理、监控信息等。

22.4.1 创建数据库

创建数据库分为4个步骤。

操作步骤

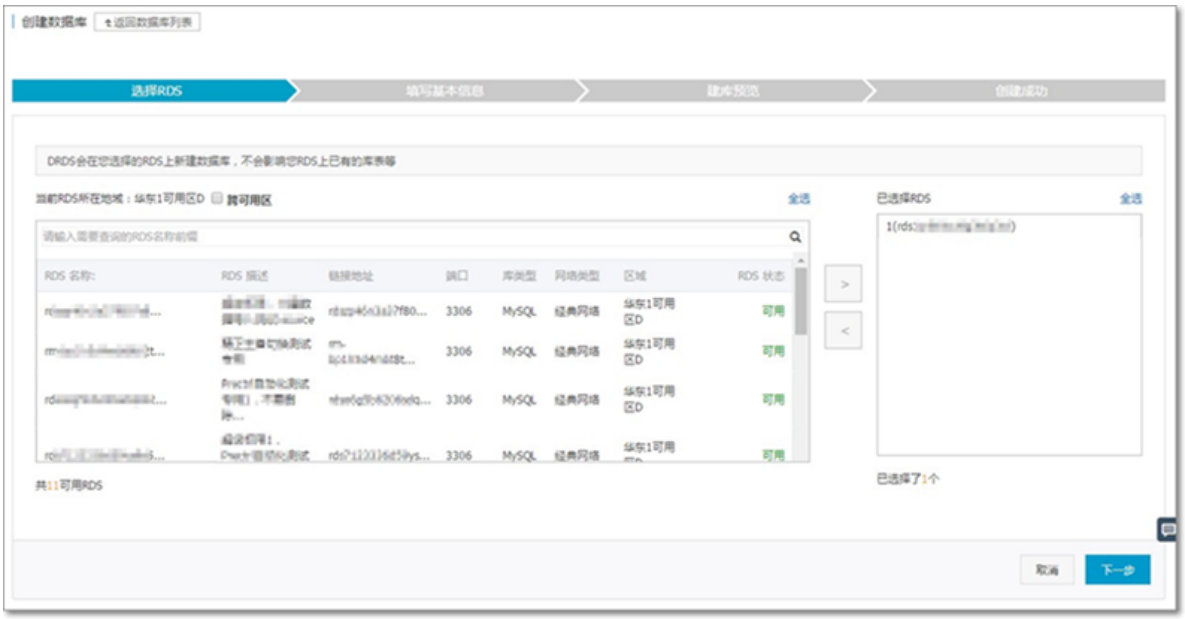
1. 登录DRDS Web控制台，在左侧菜单栏单击**实例列表**。
2. 定位到需要操作的实例，单击实例名称或者实例右侧的**管理**操作进入实例基本信息页。
3. 单击右上角的**创建数据库**按钮进入创建流程。

图 554: 实例基本信息



4. 在**选择MySQL(RDS)**页面，选择相应的数据库实例并单击**下一步**。

图 555: 选择MySQL(RDS)实例



说明:

- 左边的选择框中列出了所有可供选择的MySQL(RDS)实例。在左框选择MySQL(RDS)实例后，可以点击中间向右的箭头按钮，将该实例移动到右边的选择框。
- 如果将错误的MySQL(RDS)实例加入到了右边的选择框，可以在右边的选择框中选中对应的MySQL(RDS)实例后，点击向左的箭头取消选择该MySQL(RDS)实例。

5. 在**填写基本信息**页面输入相关信息，单击**下一步**。

数据库类型分为拆分和非拆分。选择不同需要输入的信息也不同。

- 拆分：是指在一个MySQL实例上创建多个分库，多个分库共同组成一个DRDS数据库，该模式下可以完整的使用DRDS的所有功能。DRDS 由每个分库负责每一份数据的读写操作，从

而有效的分散了整体访问压力。在系统扩容时，只需要水平增加分库的数量，并且迁移相关数据，就可以提高 DRDS 系统的总体容量。

- 非拆分：是指使用一个已经在MySQL(RDS)上建好的数据库作为一个DRDS数据库，该模式下不能使用DRDS的分库分表等功能，仅仅能使用DRDS的读写分离的功能。

图 556: 拆分模式输入项



图 557: 非拆分模式输入项



说明: 导入库是指选择一个当前选择的MySQL(RDS)实例中的已经建好的数据库。

6. 在**建库预览**页，确认信息无误后单击**下一步**进行数据库创建。

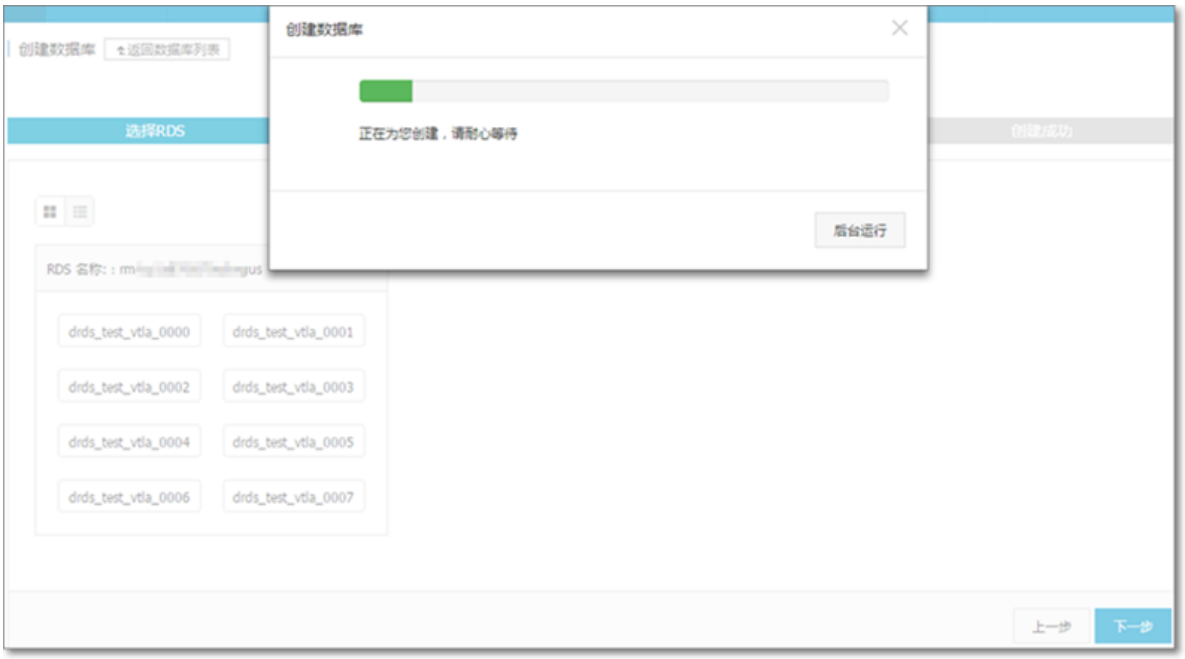
图 558: 建库预览



说明: 建库预览页面展示了即将在MySQL/RDS下创建的分库名。 DRDS建库时，会默认在其中一个分库中创建一个与DRDS数据库同名的分库存储元数据信息。

数据库创建为异步操作，通常需要几分钟，如下图所示。

图 559: 创建数据库



创建成功后或者直接在以上页面单击**后台运行**，会跳转到数据库列表页。

22.4.2 查看数据库

数据库创建成功后，可以在控制台查看数据库的基本信息。

操作步骤

在**数据库列表**页，单击数据库名称进入数据库基本信息页面。

图 560: 数据库管理



说明:

- 其中**命令行链接地址**是指登录到数据库的链接信息。由于DRDS完全兼容MySQL协议，所以您可以将地址直接复制粘贴到MySQL客户端中，输入密码即可登录DRDS数据库。
- 在该页面中您可以对数据库进行管理，比如执行DDL语句、对数据库存储进行扩容、数据导入、删除数据库等。
- 一些旧版 MySQL 客户端对用户名长度有限制，不能大于16个字符。 DRDS 使用的库名和用户名相同，建库时库名超过16个字符，则会报错。
- 由于使用 MySQL 客户端，在使用 HINT 时，在命令行上务必添加 -c 参数。DRDS 的 HINT 是通过注释实现(类似/*...*/)。如果没有加 -c 参数，会丢失注释，从而导致 DRDS 的 HINT 丢失。

22.4.3 管理只读账户

每个 DRDS 数据库可以创建一个只读用户，在 DRDS 数据库中该只读用户只能执行 SELECT、SET、SHOW 等读操作不能执行写操作，以便进行权限限制。

操作步骤

1. 在DRDS控制台左侧菜单栏单击**数据库列表**。
2. 在**数据库列表**页选择需要操作的数据库。
3. 在数据库**基本信息**页，单击右上角**只读用户管理**按钮。
4. 在**只读用户管理**对话框，输入账户密码并单击**创建只读用户**。

图 561: 创建只读账户



说明: 只读账户的账户名是固定的，用户只需要输入账户密码后即可创建只读账户。

只读账户创建成功后可对只读账户进行密码重置和删除操作。

图 562: 只读账户管理



版本号高于 5.1.26-1100253 的 DRDS 支持更完善的账户体系功能，具体请参考文档*DRDS账号和权限系统*。

22.4.4 设置读写分离

本文介绍读写分离的具体操作步骤。

前提条件

如果没有建立只读实例或者只读实例数量不足，需要先添加只读实例。

背景信息

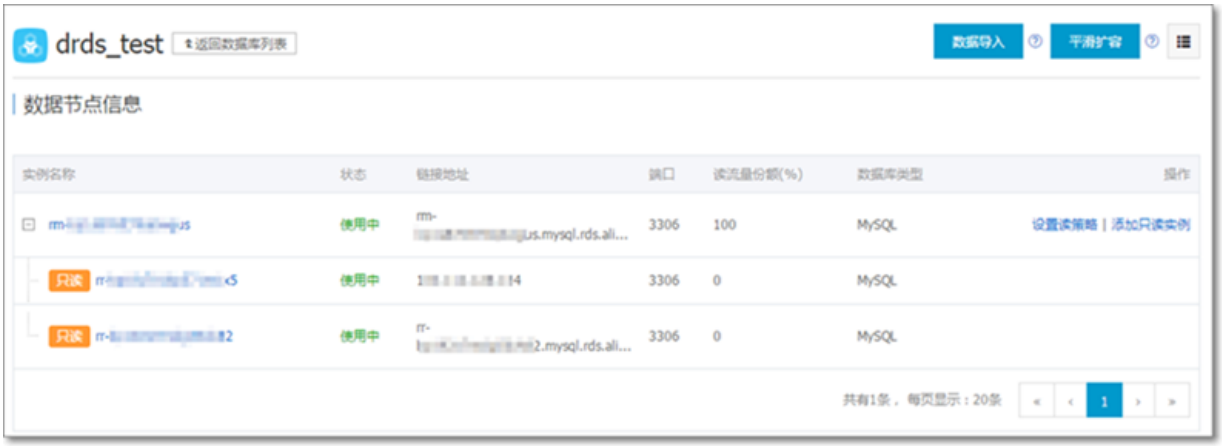
读写分离是对DRDS存储的管理以及DRDS数据库的读写流量的分配。

DRDS的读写分离功能对应用透明，即应用在不需要修改任何代码的情况下，只需要在DRDS控制台中调整读权重，即可将读流量按照需要的比例在主实例与只读实例之间调整。写权重则统一走主实例，不分流。

落到只读实例上的读操作所读取的数据都是从主实例上异步同步的，延迟在毫秒级别，所以个别要求实时性的SQL请通过*DRDS Hint* 指定主库执行。

设置读写比例在容量管理页面是以MySQL(RDS)实例为单位设置的，如果一个DRDS数据库在建库时选择了多个MySQL(RDS)实例，则需要针对每个MySQL(RDS)实例设置读写比例。

图 563: 容量管理



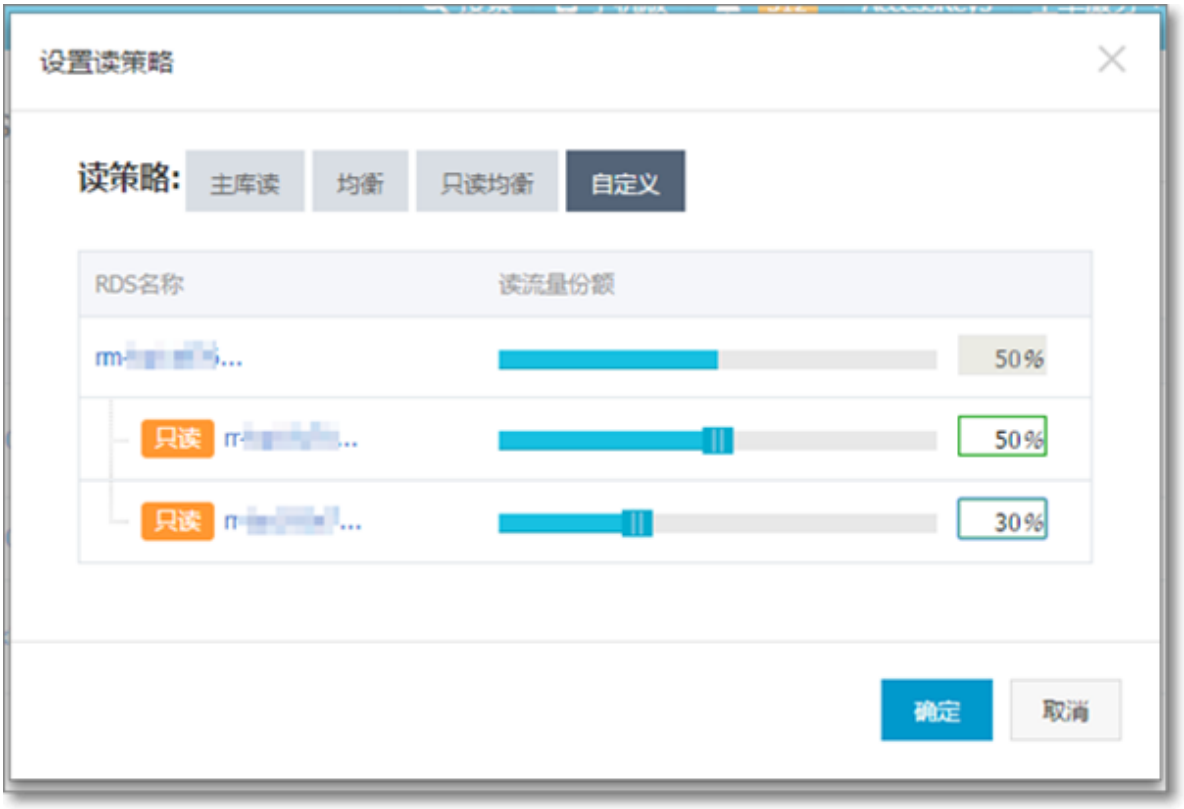
操作步骤

1. 在DRDS数据库列表页，单击需要操作的数据库名称或者右侧的**管理**操作。
2. 在数据库管理页面，单击左侧菜单栏的**读写分离**。
3. 单击右侧的**设置读策略**。

> **说明:** 如果发现只读实例数量不足，可以单击右上角**添加只读实例**先进行添加。

4. 在**设置读策略**对话框，选择所需的读策略并单击**确定**。

图 564: 设置读策略



说明:

- 不同的读策略意味着在应用访问DRDS数据库的过程中，主库和读库会分配对应比例的读流量。
- 设置完成后，可以连接 DRDS，执行 **SHOW NODE** 指令查看实际读流量分布。
- 通过 DRDS HINT 可以指定在主实例或只读实例执行 SQL，具体可以参考文档*DRDS HINT*。

22.4.5 查看监控信息

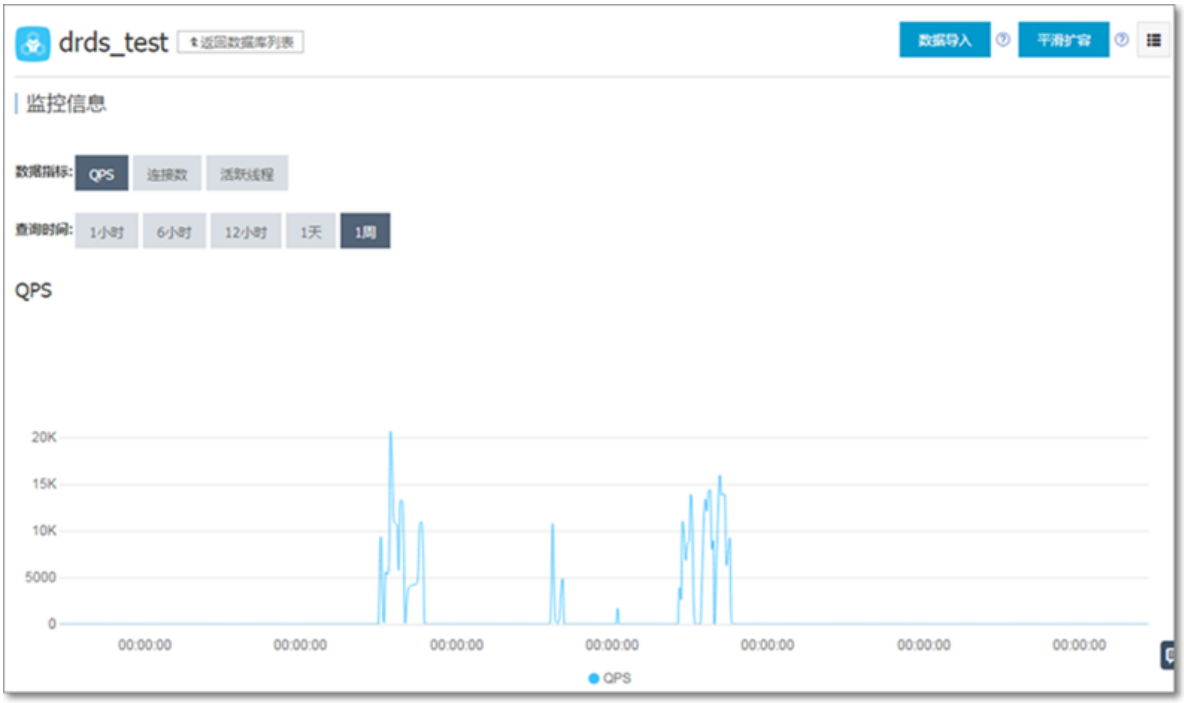
DRDS从两个维度展示了DRDS数据库的历史监控信息，即数据指标和查询时间。

操作步骤

1. 在DRDS控制台单击左侧菜单栏的**数据库列表**，选择需要查看的数据库。
2. 在数据库基本信息页，单击左侧菜单栏的**监控信息**。

3. 选择**数据指标**和需要查询的**时间维度**，即可看到相应的监控信息。

图 565: 查看监控信息



说明: 实例级别的监控请参考文档实例监控。

22.4.6 查看分库信息

分库管理页列出了当前的DRDS数据库中，各个分库与MySQL(RDS)实例的对应关系。

操作步骤

1. 在DRDS控制台单击左侧菜单栏的**数据库列表**，选择需要查看的数据库。
2. 在数据库基本信息页，单击左侧菜单栏的**分库管理**，即可看到该数据库下的所有分库。
3. 单击数据库名或者右侧的**查看库信息**进入详情页。

   在分库详情页面可以看到该分库下有哪些分表，以及DRDS连接该分库时所使用的连接参数等信息。

22.4.7 管理数据表

数据表管理主要包括DDL语句执行，查看表结构，设置全表扫描，设置小表广播属性，表删除等功能。

背景信息

在数据表管理页面列出了所有数据表的列表，并展示了表名、表属性，以及可以对表进行的管理操作。如果是分表则会展示表的拆分键。相关概念如下：

- 拆分键：即分库/分表字段。 DRDS 根据拆分键的值将数据表水平拆分到后端的每一个 MySQL(RDS) 分库里。换句话说，拆分键值相同的数据，一定会位于同一个 MySQL(RDS) 数据库里。DRDS 里除了可以定义分库键以外，每一张逻辑表都可以定义自己的拆分键。拆分键只能是单个字段。如果分库键与分表键相同，那么在插入时只需要指定该分库/分表键。如果分库键与分表键不同则在插入时需要同时指定分库键和分表键。
- 全表扫描：复杂的SQL语句会分发到所有库上执行，并且在 DRDS 中进行比较和合并。全表扫描相当消耗性能，应该在业务中尽量避免。
- 小表广播：是指将数据量较小且数据更新少的数据单表广播到需要联合查询分表对应的分库上，以小表作为驱动表加速分布式联合查询。

操作步骤

1. 在DRDS控制台**实例列表**页选择需要操作的实例，进入实例管理页面。
2. 在实例管理页单击左侧菜单栏的**数据库列**表，选择需要查看的数据库。
3. 在数据库概览页，单击左侧菜单栏的**数据表管理**，即可看到数据表的信息。

图 566: 数据表管理



在数据表信息页面，可以**查看表结构**，或者单击**高级设置**进行全表扫描设置。

对已经切分的表做查询的时候，如果where条件里不带分库键，那么就需要开启全表扫描。

执行DDL语句

如需执行DDL语句，请按以下步骤操作。

4. 如需执行DDL语句，请单击右上角**新建表/分布式DDL按钮**。

5. 在以下对话框上传文件或者直接输入DDL语句，并单击**开始DDL**。

图 567: 新建表/分布式DDL



6. 在新建表对话框，单击表名称右侧的**设置分库策略**选项，选择一个字段对表进行拆分。

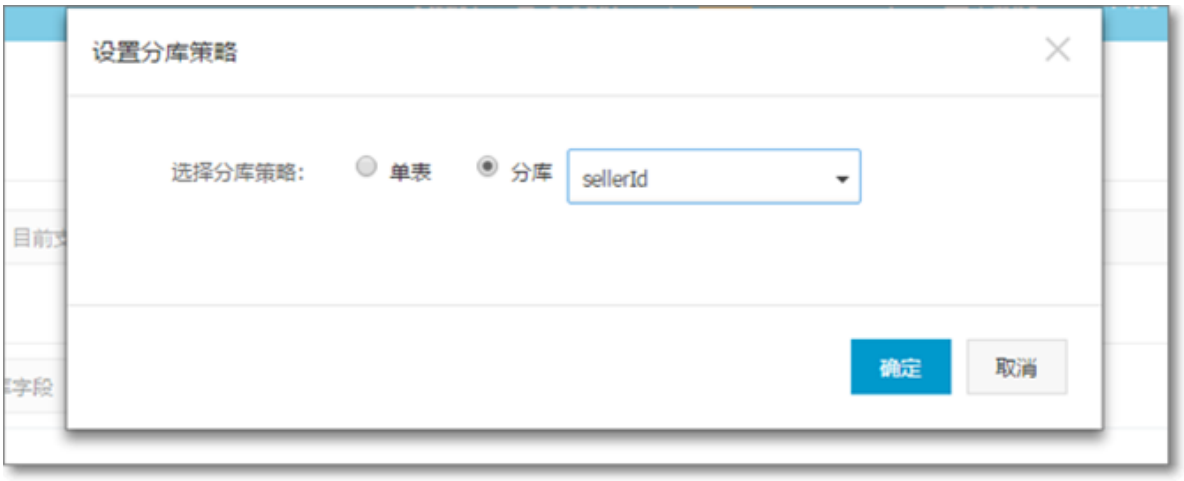
拆分字段的作用是在执行DML语句时按照拆分字段将SQL路由到对应的分库。

图 568: 设置分表字段



7. 在设置分库策略对话框选择分库及拆分字段，单击**确定**。

图 569: 设置分库策略



8. 重复步骤6-7，直到所有的表都设置完成。单击**下一步**提交DDL任务。

   建表完成后，在数据表列表页就可以看到新建的数据表了。

   图 570: 数据表列表



   在数据表列表页面，除了可以查看表结构，还可以单击**高级设置**按钮进行设置。对于单表，可以通过高级设置将表设置为广播表；对于分表，则可以将表设置为全表扫描。

22.4.8 设置白名单

白名单功能用于限制能够访问该DRDS数据库的机器IP。

操作步骤

1. 在DRDS控制台单击左侧菜单栏的**数据库列表**，选择需要查看的数据库。
2. 在数据库基本信息页，单击左侧菜单栏的**白名单设置**。
3. 在白名单设置页面，单击**手动修改**，输入允许访问该数据库的IP并单击**确定**。

   IP 地址支持全IP、 CIDR 模式和 * 号占位符模式。当配置为 *.*.*.* 或空时，为不设置 IP 访问限制。

可以单击**查看DRDS机器列表**按钮查看机器IP。

22.4.9 导入数据

DRDS支持将现有的MySQL(RDS)实例上的数据导入到DRDS数据库。

背景信息

如果待导入的数据量比较小，比如总数据量少于500万条，推荐使用Navicat或者sqldump导出数据，再使用mysql source命令将数据导入到DRDS数据库。如果源MySQL(RDS)实例中的数据量较大，那么可以使用DRDS Web控制台提供的数据导入功能。使用该功能的前提是源MySQL(RDS)实例已经由DRDS运维系统管理（具体请参见DRDS运维文档）。

DRDS Web控制台的数据导入流程分为**建表** > **配置** > **迁移** > **切换**四个步骤。

操作步骤

建表

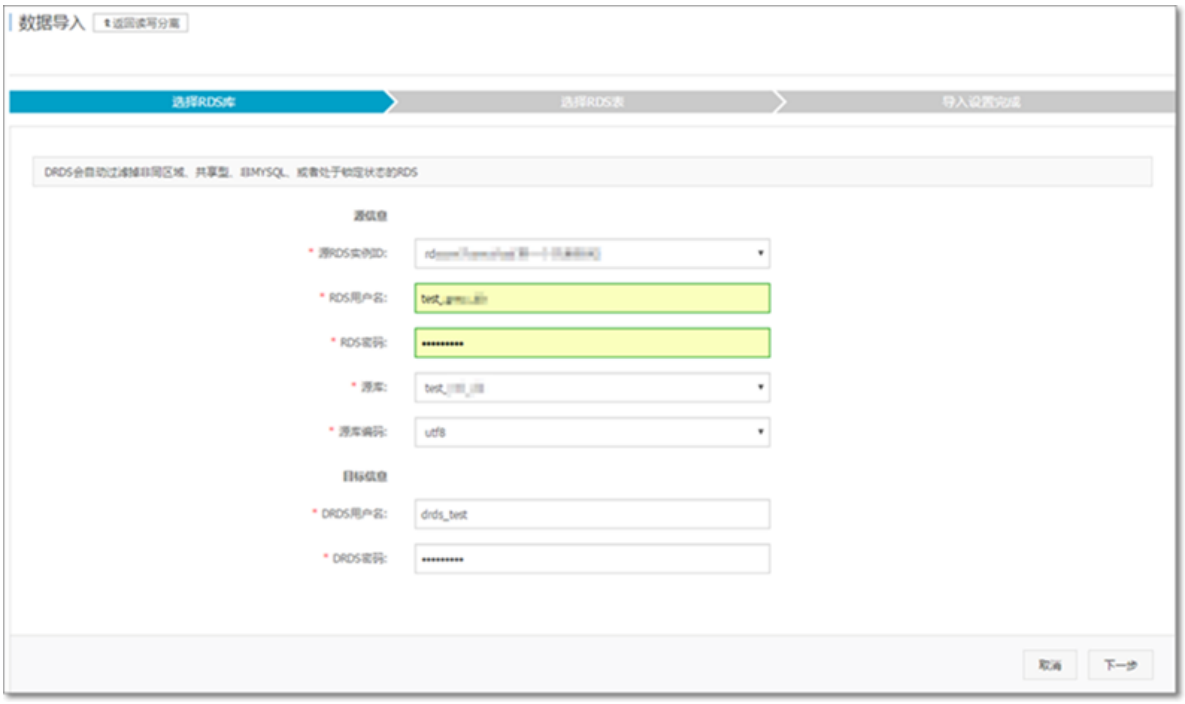
在数据导入之前，首先需要在DRDS数据库上创建相同表结构的表，字段名和字段类型必须一致。

1. 请参考创建数据表完成建表操作。

配置

2. 在DRDS控制台左侧菜单栏单击**数据库列表**，选择需要操作的数据库。
3. 在数据库基本信息页，单击右上角的**数据导入**按钮，进入数据导入向导。
4. 在**选择**MySQL（RDS）**库**页面，输入需要的信息，单击**下一步**。

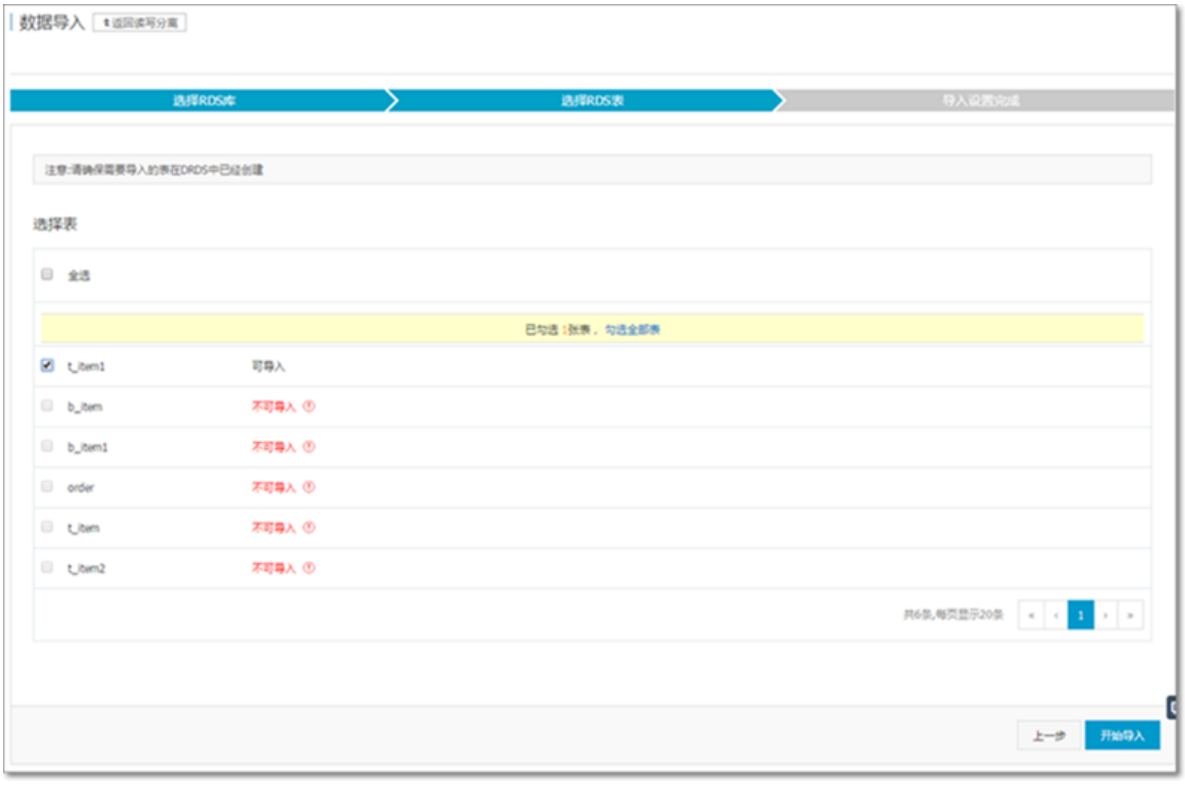
图 571: 选择MySQL数据库



DRDS控制台将对比源数据库和DRDS数据库所有表的表结构，并展示可导入的表和不可导入的表。

5. 在**选择**MySQL（RDS）**表**页面，勾选需要导入的表，单击**开始导入**。

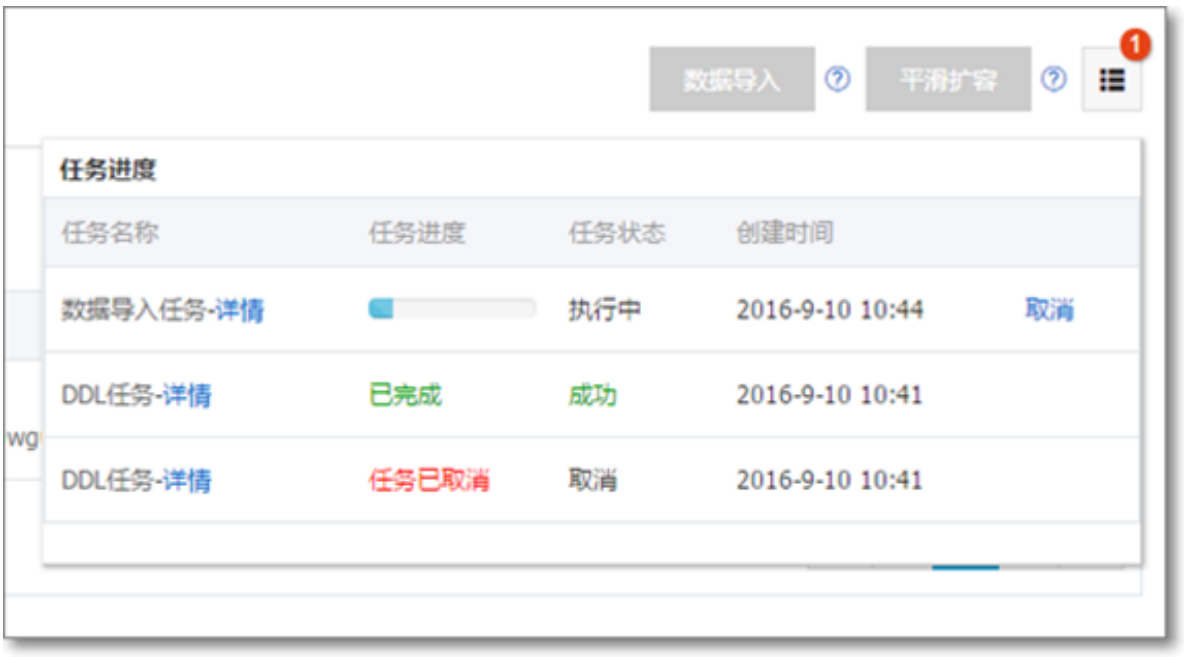
图 572: 选择 MySQL（RDS）**表**



迁移

6. 单击右上角任务列表可以查看数据导入的状态，如下图。

图 573: 数据导入任务状态



切换

7. 数据迁移任务完成后，单击右侧的**切换**按钮进行切换。

图 574: 数据导入成功



说明: 导入的数据包含全量（所有历史数据）和增量（全量迁移期间产生的数据）两部分，执行顺序是导入全量再导入增量。导入任务显示成功时，代表目前DRDS中数据已经和 MySQL(RDS) 实例中数据同步。此时后台数据迁移进程还在运行，以保证维持同步状态。当点击切换才会停止增量数据的迁移。切换后，就停止了从源库中读取增量数据，数据导入完成。

22.4.10 删除数据库

如果需要删除数据库，可以在**数据库列表**或者**数据库基本信息**页面进行。

操作步骤

1. 在数据库列表页，找到需要删除的数据库，单击右侧**删除**选项。
2. 在**删除数据库**对话框，单击**确定**。

图 575: 删除数据库



说明: 只有在公共云环境需要验证手机号码，专有云环境下则不需要。

22.4.11 进行平滑扩容

对于拆分表，如果发现已经达到存储瓶颈，需要进行水平扩展，那么可以通过平滑扩容来实现。

背景信息

DRDS 是一个两层结构，上层提供 SQL 路由和数据获取、聚合等功能，底层由 MySQL（RDS）提供数据储存功能。随着业务数据量的增加，会出现底层存储无法承受业务压力的情况。这时可以通过升级MySQL（RDS）规格来提高单个MySQL（RDS）实例的处理能力（垂直扩展），也可通过增加 MySQL（RDS）实例数量来增加系统总体处理能力（水平扩展）。

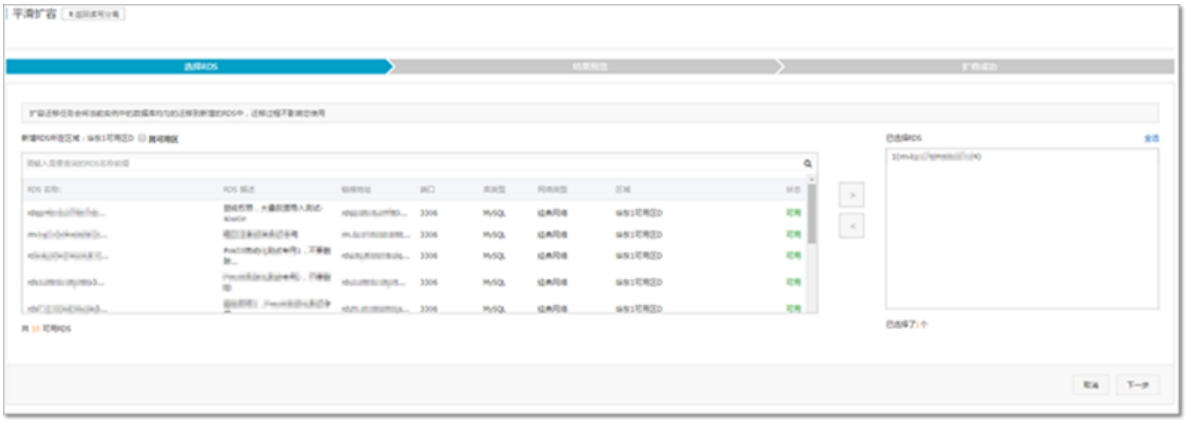
平滑扩容属于后者，是一种水平扩容的实现方式。对于单机数据库水平扩展是无法做到的，而垂直扩展有自身瓶颈，扩展范围有限。对于快速发展的业务，平滑扩容是一种高效、稳定、易用的解决方案。

平滑扩容分为**配置** > **迁移** > **切换** > **清理**四个步骤。

操作步骤

1. 在DRDS控制台数据库列表页，单击需要操作的数据库名称或者右侧的**管理**操作，进入数据库管理页面。
2. 单击右上角的**平滑扩容**按钮，进入DRDS平滑扩容向导。

图 576: 平滑扩容向导



配置

即选择存储节点。

3. 在**选择MySQL(RDS)**页，选择需要新增的实例，添加到右侧已选实例列表中，并单击**下一步**。
4. 在**结果预览**页，确认信息无误后单击**开始扩容**。

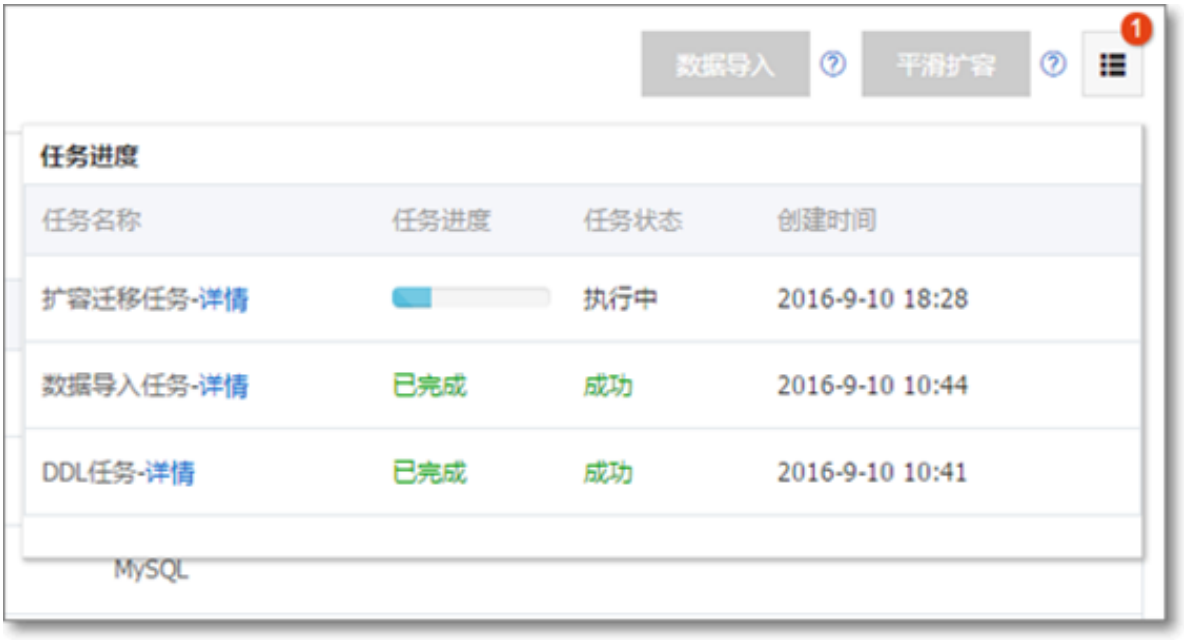
   扩容任务开始异步执行。在数据库管理页面的右上角能看到扩容执行状态。

迁移

迁移是指将待迁移数据库中的历史数据迁移至新增的MySQL/RDS中。

5. 在任务列表中查看，单击右上角的按钮可以查看任务执行的状态。

图 577: 扩容任务状态



说明: 在执行切换前本次扩容还没有对您的DRDS产生任何影响，因此在切换前可以通过回滚来放弃本次扩容。

历史数据迁移完成后，迁移任务进度会达到100%，此时可以进行切换。

图 578: 数据迁移完成



切换

切换任务会将您的读写流量切换到新增的MySQL(RDS)实例上，切换过程大约持续2~5分钟。请在业务低谷期执行切换。

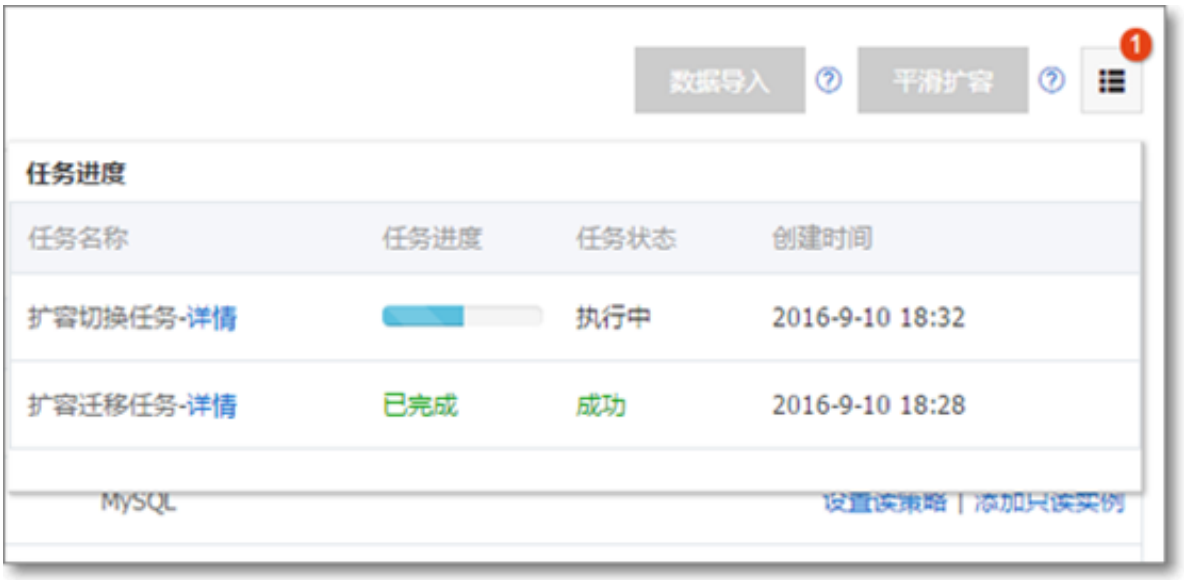
6. 在上图显示任务进度对话框，单击**切换**按钮。
7. 在弹出的**扩容切换**对话框单击**确定**。

图 579: 扩容切换



切换开始后会生成一个切换任务，任务也会显示在任务栏中。

图 580: 扩容切换任务



清理

清理任务是指删除本次扩容后不再使用的数据库。可以考虑备份后再执行此操作。本操作对数据库有一定压力，请在业务低谷期执行切换。

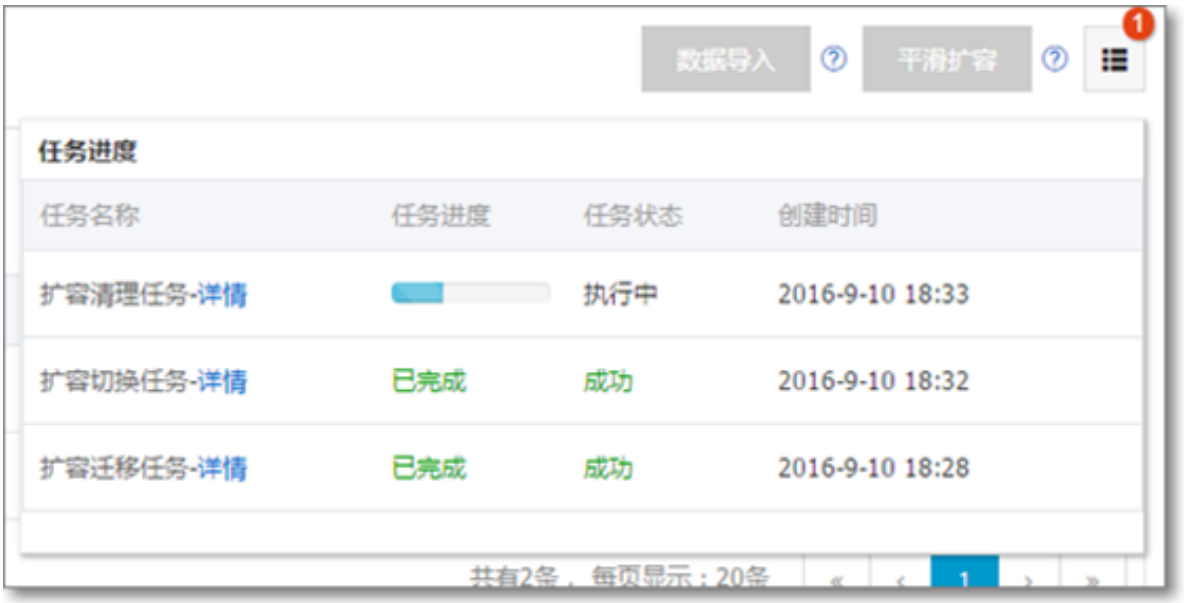
8. 切换任务完成后，单击任务窗口右上角的**清理**按钮。

图 581: 切换完成



9. 在弹出的对话框右下角单击**确定**进行清理。

图 582: 扩容清理



清理任务完成后，平滑扩容就完成了。新增的MySQL(RDS)实例会成为DRDS数据库的存储节点。

22.4.12 设置小表广播

DRDS 将一些数据量小且更新频度不高的数据表存储为单表模式，这些数据表称为小表。通过数据同步将小表复制到与之JOIN 的分库上进而提升JOIN效率的解决方案称为小表广播或者小表复制。

背景信息

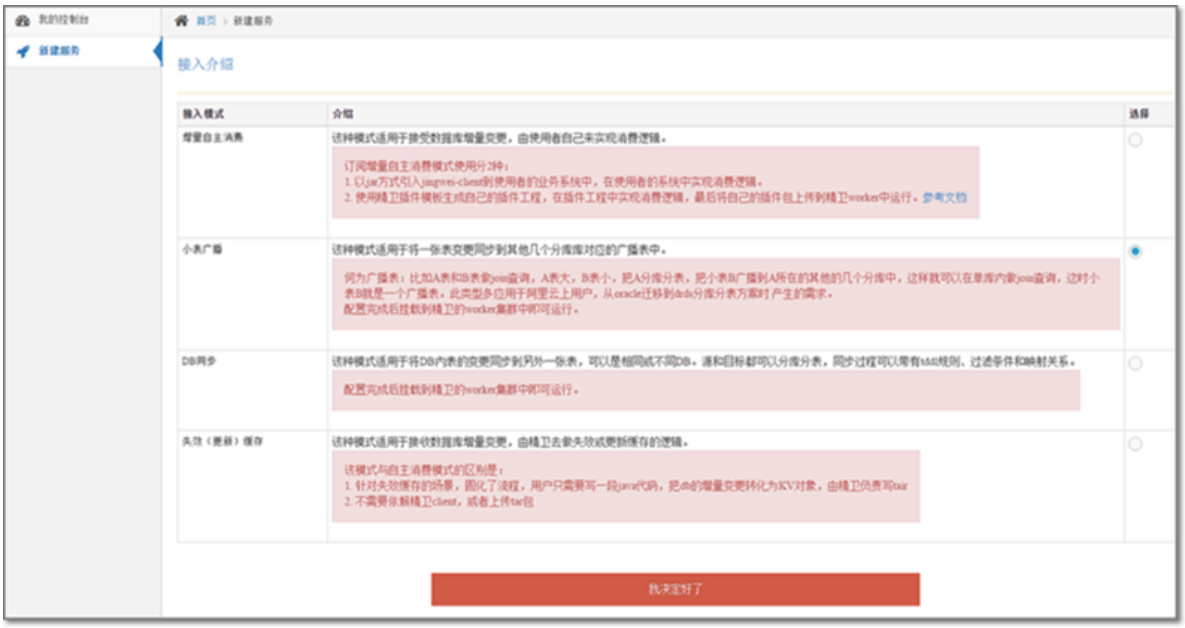
一些大的业务表进行切分后，总会存在一些数据量不大，更新量也不大的原始信息表。这些表往往会与切分后的大表进行JOIN操作。这种操作物理上就会造成分布式JOIN查询，效率相对比较低下。

针对这种分布式JOIN的场景，DRDS提供了专用工具来进行小表广播，将原信息表的所有数据（包括增量更新）全部自动的广播到大表的机器上。这样，原来的分布式JOIN查询就变成单机本地查询，从而大大提高了效率。

操作步骤

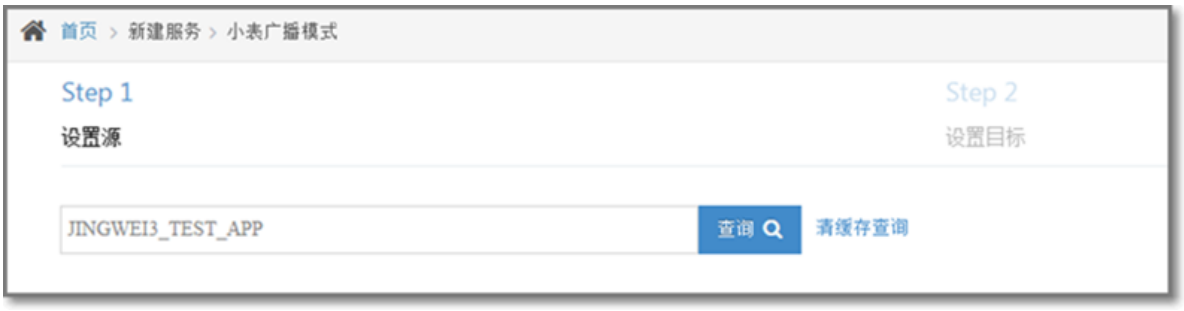
1. 打开精卫控制台，新建服务并选择小表广播模式。

图 583: 选择小表广播



2. 在新建服务第一步**设置源**对话框输入**DRDS appname**查询源表信息。

图 584: 设置源



说明: appname是一个DRDS数据库的唯一标识。

3. 选择需要做广播的表，并勾选下方的**不需要全量迁移**，链路类型请选择**直连DB BinLog**。
4. 在**设置目标**页面确认源到目标的映射并单击**完成**。

   页面会自动跳转到服务详情页。
5. 单击增量同步服务名称进入任务详情页。

图 585: 服务详情



6. 在任务详情页面单击**挂载集群**选择任务运行的集群，然后点击**启动**运行任务。

图 586: 启动任务



等待几分钟后刷新页面，确认任务启动成功（即上图的**运行状态**变成绿色且显示RUNNING），则小表广播设置成功。

22.4.13 设置异构索引

如需查询数据库的分表，请在SQL查询语句的where条件中带上分库分表键，以便将查询路由到具体的分库中，从而提高查询效率。如果SQL的where条件中没有分库分表键，DRDS会进行一次全表扫描，造成查询效率低下。DRDS提供了异构索引功能来解决这个问题。

操作步骤

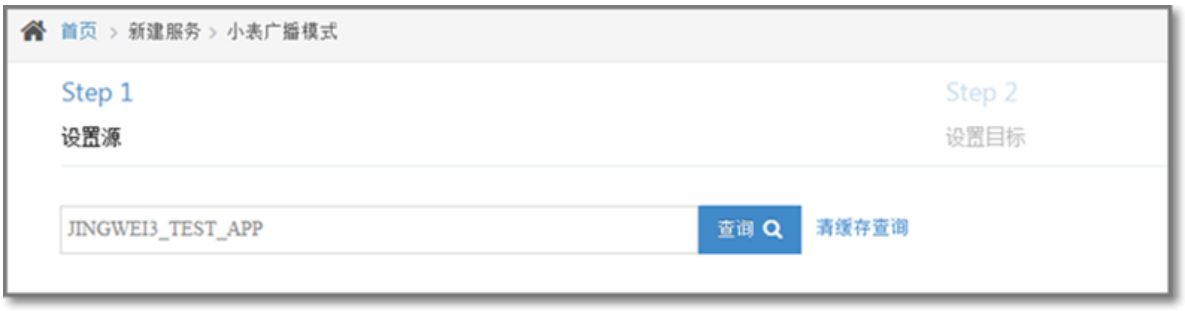
1. 登录精卫控制台，在左侧菜单栏选择**新建服务**，并选择**DB同步**模式。

图 587: 选择模式



2. 在新建服务第一步**设置源**对话框输入**DRDS appname**查询源表信息。

图 588: 设置源

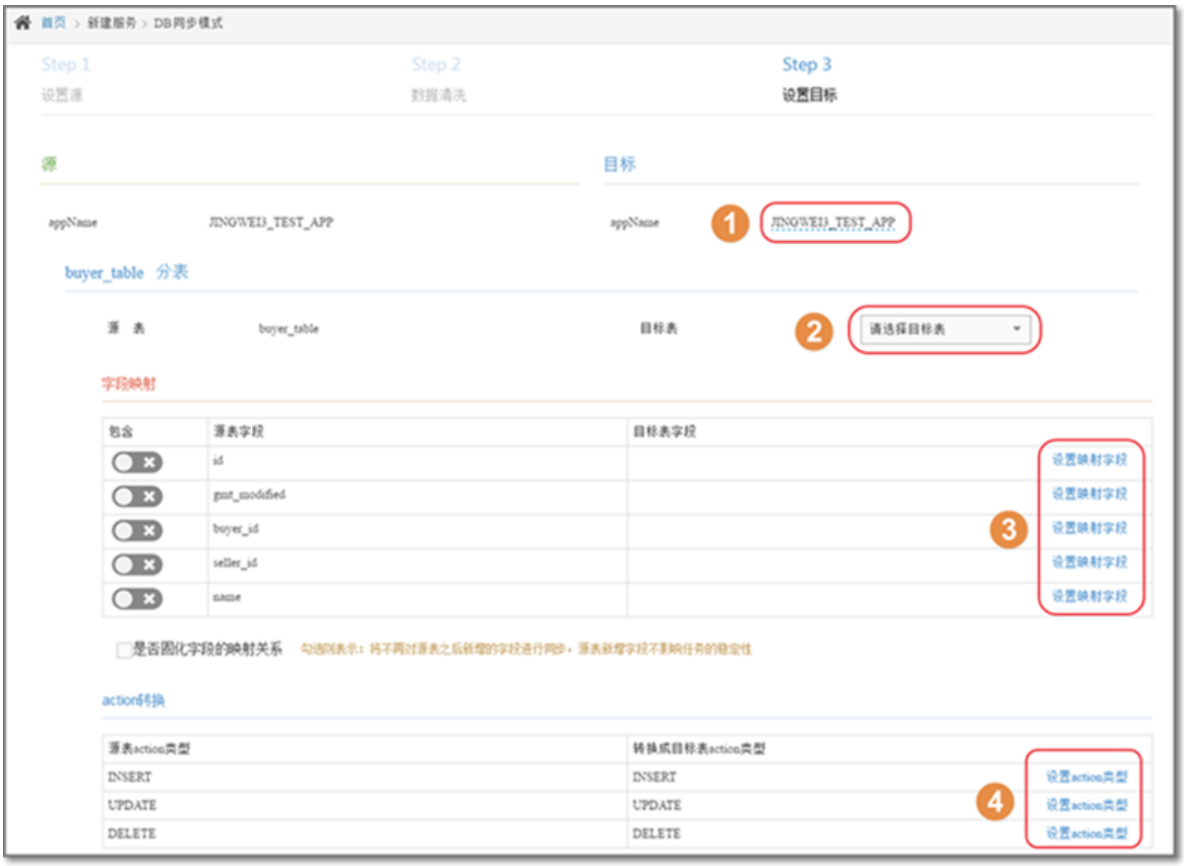


3. 选择源app中需要同步的表和操作，链路类型选项请选择**直连DB binlog**。
4. 在数据清洗页面直接单击**下一步**。

> **说明:** 数据清洗是指通过Java代码的方式过滤数据，正常情况下可不用。

5. 在**DB同步模式**页面，根据需要设置相关选项并单击**完成**。

图 589: 选择功能



说明:

- 目标app：通常无需设置，选择默认就好。
- 目标表：选择目标表。
- 字段映射：如果源表和目标表的字段不一样，可以在此处设置映射关系。
- action类型转换：适用于INSERT转成UPDATE这类需求。

6. 在任务详情页面单击**挂载集群**选择任务运行的集群，然后点击**启动**运行任务。

图 590: 启动任务



等待几分钟后刷新页面，确认任务启动成功（即上图的**运行状态**变成绿色且显示RUNNING），异构索引就设置成功了。

22.5 使用DRDS

DRDS数据库成功创建之后，就可以在创建的数据库中使用SQL语句执行各种操作了。

DRDS完全兼容MySQL协议，因此接入方式与MySQL数据库完全相同。通过*DRDS数据库的连接串*就可以使用MySQL客户端登录到DRDS数据库进行操作，同时也可以在应用中连接DRDS数据库。

22.5.1 使用DDL语句

DRDS除了支持在Web控制台执行DDL以外，还支持登录到数据库执行DDL。

控制台建表和登陆到数据库建表的区别在于：

- 在Web控制台执行建表DDL语句只能建单表或者分库的表，而不能建分库分表。登录到DRDS数据库后支持完整的DRDS DDL语句。
- 由于控制台执行建表语句有一个步骤可以设置分库键，所以控制台的建表语句不支持下文将要介绍到的**DBPARTITION BY**，**TBPARTITION BY**，**TBPARTITIONS**，**BROADCAST**等语句。

建表DDL

登录到DRDS数据库建表的时候需要指定几个关键参数：

- DBPARTITION BY hash(partition_key)：指定分库键和分库算法。
- TBPARTITION BY（可选）：默认与DBPARTITION相同，指定数据与物理表使用什么方式进行映射（指定分表键）。
- TBPARTITIONS（可选）：每个库上的物理表数目（默认为1）。如无需分表，则无需指定该字段。
- BROADCAST（与DBPARTITION BY互斥）：指定建广播表。

下面以创建数据库表的例子来介绍DRDS数据库中创建表的方式。

- 建立单表：SQL语句与MySQL中使用的SQL一样。

```
CREATE TABLE single_tbl
   (id int, name varchar(30), primary key(id));
```

- 分库不分表，且切分方式为根据以id列的值进行hash运算后的结果将表中数据分布在多个子库中，SQL语句是：

```
CREATE TABLE multi_db_single_tbl
   (id int, name varchar(30), primary key(id)) dbpartition by hash(id);
```

- 如果希望建一个分表，每个分库含有3张物理表，切分方式都为按照id列进行hash运算（会根据id列的值进行hash运算后的结果将表中数据分布在多个子库中，每个子库中的数据再根据运算结果分布在3个物理表中），则SQL语句是：

```
CREATE TABLE multi_db_multi_tbl
   (id int, name varchar(30), primary key(id))
   dbpartition by hash(id) tbpartition by hash(id) tbpartitions 3;
```

- 如果希望建一个分表，每个库含有3张物理表，切分方式为库按照id列进行hash运算，物理表按照id2列进行hash运算（会根据id列的值进行hash运算后的结果将表中数据分布在多个子库中，每个子库中的数据再根据id2列值的hash运算结果分布在3个物理表中），则SQL语句是：

```
CREATE TABLE multi_db_multi_tbl
   (id int auto_increment, id2 int,
name varchar(30), primary key(id))
   dbpartition by hash(id) tbpartition by hash(id2) tbpartitions 3;
```

除了可以使用hash来做映射策略，还可以使用日期类策略MM/DD/WEEK/MMDD来进行映射，但只限于对物理表的映射。这种方式通常用于将不同的日期的数据进行物理表集中的场景，如业务员操作日志。

- 公司A的日志系统记录营业员进行的所有操作，按照营业员id以及一周七天来切分，SQL语句是：

```
CREATE TABLE user_log
   (userId int, name varchar(30), operation varchar(30), actionDate DATE)
   dbpartition by hash(userId) tbpartition by WEEK(actionDate) tbpartitions;
```

- 如果按照一年12个月进行日志记录，上面的例子可以改为：

```
CREATE TABLE user_log
   (userId int, name varchar(30), operation varchar(30), actionDate DATE)
   dbpartition by hash(userId) tbpartition by MM(actionDate) tbpartitions 12;
```

- 如果按照一个月31天进行日志记录，上面的例子可以改为：

```
CREATE TABLE user_log
   (userId int, name varchar(30), operation varchar(30), actionDate DATE)
   dbpartition by hash(userId) tbpartition by DD(actionDate) tbpartitions 31;
```

- 如果按照一年365天进行日志记录，上面的例子可以改为，但这样将在每个分库中创建365个物理表，那么SQL语句是：

```
CREATE TABLE user_log
   (userId int, name varchar(30), operation varchar(30), actionDate DATE)
   dbpartition by hash(userId) tbpartition by MMDD(actionDate) tbpartitions 365;
```

- 如果您希望数据可以在每个分库中只分布到10个物理表：

```
CREATE TABLE user_log
   (userId int, name varchar(30), operation varchar(30), actionDate DATE)
   dbpartition by hash(userId) tbpartition by MMDD(actionDate) tbpartitions 10;
```

- 建广播表，SQL语句是：

```
CREATE TABLE brd_tbl(id int, name varchar(30), primary key(id))
   ENGINE=InnoDB DEFAULT CHARSET=utf8 BROADCAST;
```

在DRDS中，增加了建表DDL语句的选项以外，其他的DDL语句与MySQL没有区别，使用时与MySQL的语法保持一致。

22.5.2 使用DML语句

DRDS的DML语句与MySQL语句一样。

需要注意的是，在DRDS的**SELECT**、**UPDATE**语句中应尽量带上拆分建；在DRDS的**INSERT**语句必须带上拆分键与键值，且值不能为空。

22.5.3 使用全局唯一数字序列

DRDS 全局唯一数字序列（64位数字，对应 MySQL 中 Signed BIGINT 类型，以下简称为 Sequence）的主要目标是为了生成全局唯一和有序递增的数字序列，常用于主键列、唯一索引列等值的生成。

DRDS中的Sequence主要有两类用法：

- 显式 Sequence，通过 Sequence DDL 语法创建和维护，可以独立使用；通过select seq.nextval;获取序列值，seq 是具体 Sequence 的名字；
- 隐式 Sequence，在为主键定义 AUTO_INCREMENT 后，用于自动填充主键，由 DRDS 自动维护。

> **注意:** 仅拆分表和广播表指定了 AUTO_INCREMENT 后，DRDS 才会创建隐式的 Sequence。非拆分表并不会，非拆分表的 AUTO_INCREMENT 的值由底层 MySQL（RDS）自己生成。

DRDS Sequence 类型与特性说明

目前共支持3种 Sequence 类型：

类型（缩写）	全局唯一	连续	单调递增	同一连接内单调递增	非单点	数据类型	可读性
Group Sequence（GROUP）	是	否	否	是	是	所有整型	好
Time-based Sequence（TIME）	是	否	宏观上单调递增，微观上非单调递增。	是	是	仅支持 BIGINT	差
Simple Sequence（SIMPLE）	是	是	是	是	否	所有整型	好

概念解释：

- **连续**：如果本次取值为 n，下一次取值一定是 n + 1，则是连续的；如果下一次取值不能保证为 n + 1，则是非连续的；
- **单调递增**：如果本次取值为 n，下一次取值一定是一个比 n 大的数，则是单调递增的；
- **单点**：存在单点故障风险；

- **宏观上单调递增**，**微观上非单调递增**：类似于1、3、2、4、5、7、6、8、.....这样的序列，这个序列从宏观是看是递增的，微观上非单调递增。

Group Sequence（GROUP，**默认使用**）

特性

全局唯一的 Sequence，产生的值是自然数序列，但是不保证连续和单调递增。如果未指定 Sequence 类型，DRDS 默认使用 Group Sequence。

- 优点：**全局唯一、不会产生单点问题、性能非常好**；
- 缺点：**产生的序列不连续、可能会有跳跃段**；**不会严格从起始值开始取值**；**不能循环。**

实现机理

采用多个节点产生值来保证高可用，每次取出一段值，如果该段值没有取完（比如连接断掉等情形），就会产生跳跃段。

Time-based Sequence（TIME）

特性

基于**时间戳 + 节点编号 + 序列号**组合而成的一种 Sequence，保证全局唯一和宏观自增；这种 Sequence 值的更新不依赖于数据库，也不需要持久化到数据库，仅在数据库中保留名称和类型信息，性能很好；产生的是类似于776668092129345536、776668098018148352、776668111578333184、776668114812141568、.....这样的序列值。



注意: 用于表中自增列时，**必须使用 BIGINT 类型。**

- 优点：**全局唯一、性能很好。**
- 缺点：产生的序列不连续，起始值、步长、最大值、是否循环这些参数对于 Time-based Sequence 无意义。

Simple Sequence（SIMPLE）

特性

仅 Simple Sequence 支持自定义步长、最大值和循环/非循环利用。

- 优点：**全局唯一、连续、单调递增。**
- 缺点：**单点**，**性能较差**，**存在瓶颈**，**需要谨慎使用。**

实现机理

每产生一个值都要进行一次持久化操作。

使用场景

这三种 Sequence 都保证全局唯一，均可以应用在**主键列**和**唯一索引列**。

- 大部分场景下建议选用 **Group Sequence**；
- 如果业务强依赖连续的 Sequence 值，此时只能使用 Simple Sequence（注意 Simple Sequence 的性能问题）；
- 对于 Sequence 的性能要求比较高，表的插入量很小，Sequence 序列可以接受很大的值，优先考虑使用 Time-based Sequence。它是纯 CPU 计算型的，计算过程中不需要加锁，不依赖于数据库，也不需要进行持久化操作。

以创建一个起始值是 100000，步长为 1 的 Sequence 为例。

- 如果采用 **Simple Sequence**，则会严格产生100000、100001、100002、100003、100004、.....、200000、200001、200002、200003、.....这样的序列（全局唯一、连续、单调递增）。Simple Sequence 会保证持久化，即使发生单点问题，服务重启后依然会在断点继续产生 Sequence 值，中间不会产生跳跃段。Simple Sequence 的机理是每产生一个值都要进行一次持久化操作，因此性能并不是很好。
- 如果采用 **Group Sequence**，产生的序列有可能是200000、200001、200002、200003、201001、201002、201003、.....这样的序列。

  注意:

  - Group Sequence 起始值并不会严格从设定的参数（本例中是100000）开始，但是保证比该参数大。本例中是从200000开始取值的。
  - Group Sequence 保证全局唯一，但是会有跳跃段。比如 Group Sequence 的某个节点挂掉，或者某个连接只取了一部分值，然后该连接被关闭了，都会产生跳跃段。该例中200003和201001之间的值丢掉了，即产生了跳跃段。
- 如果采用 **Time-based Sequence**，产生的序列有可能是7766680921293455536、7766680980181483352、7766681115783331184、7766681148121415688、.....这样的序列。

22.5.3.1 Sequence显示用法

本文介绍如何用 DDL 语句创建、修改、删除、查询 Sequence， 以及如何获取显式 Sequence 的值。

操作步骤

创建Sequence

1. 创建Sequence语句如下：

```
CREATE [ GROUP | SIMPLE | TIME ] SEQUENCE <name>
[ START WITH <numeric value> ] [ INCREMENT BY <numeric value> ]
[ MAXVALUE <numeric value> ] [ CYCLE | NOCYCLE ]
```

参数说明：

参数	说明	使用范围
START WTIH	Sequence 的起始值，若未指定，则默认值为1。	Simple Sequence、Group Sequence
INCREMENT BY	Sequence 每次增长时的增量值（或称为间隔值或步长），若未指定，则默认值为1。	Simple Sequence
MAXVALUE	Sequence 允许的最大值，若未指定，则默认值为有符号长整型（Signed BIGINT）的最大值。	Simple Sequence
CYCLE或NOCYCLE	两者只能选择其一，代表当 Sequence 值增长到最大值后，是否允许继续循环（即仍从 START WITH 开始）使用 Sequence 值。若未指定，则默认值为 NOCYCLE。	Simple Sequence

注意:

- 如果未指定类型关键字，则默认类型为 Group Sequence；
- **INCREMENT BY**、**MAXVALUE**、**CYCLE 或 NOCYCLE**这三个参数对于 Group Sequence 是无意义的；

- **START WITH**、**INCREMENT BY**、**MAXVALUE**、**CYCLE** 或 **NOCYCLE**这4个参数对于 Time-based Sequence 是无意义的。
- Group Sequence 是非连续的。**START WITH** 参数对于 Group Sequence 仅具有指导意义，**Group Sequence 不会严格按照该参数作为起始值，但是保证起始值比该参数大。**

以下为创建不同类型 Sequence 的几个示例。

示例一：**创建一个 Group Sequence。**

- 方法一：

```
mysql> CREATE SEQUENCE seq1;
Query OK, 1 row affected (0.27 sec)
```

- 方法二：

```
mysql> CREATE GROUP SEQUENCE seq1;
Query OK, 1 row affected (0.27 sec)
```

示例二：**创建一个 Time-based Sequence。**

```
mysql> CREATE TIME SEQUENCE seq1;
Query OK, 1 row affected (0.27 sec)
```

示例三： **创建一个 Simple** Sequence，**起始值是** 1000，**步长为** 2，**最大值为** 99999999999，**增长到最大值后不继续循环。**

```
mysql> CREATE SIMPLE SEQUENCE seq2 START WITH 1000 INCREMENT BY 2
 MAXVALUE 99999999999 NOCYCLE;
Query OK, 1 row affected (0.03 sec)
```

2. 修改Sequence

DRDS 支持对 Sequence 的以下几个方面进行修改：

- 修改 Simple Sequence 的参数：起始值、步长、最大值、循环或非循环；
- 修改 Group Sequence 的参数：起始值；
- 不同类型 Sequence 间的转换。

```
ALTER SEQUENCE <name> [ CHANGE TO GROUP | SIMPLE | TIME ]
START WITH <numeric value> [ INCREMENT BY <numeric value> ]
[ MAXVALUE <numeric value> ] [ CYCLE | NOCYCLE ]
```

参数说明：

参数	说明	适用范围
START WITH	Sequence 的起始值，若未指定，则默认值为1。	Simple Sequence、Group Sequence
INCREMENT BY	Sequence 每次增长时的增量值（或称为间隔值或步长），若未指定，则默认值为1。	Simple Sequence
MAXVALUE	Sequence 允许的最大值，若未指定，则默认值为有符号长整型（Signed BIGINT）的最大值。	Simple Sequence
CYCLE或NOCYCLE	两者只能选择其一，代表当 Sequence 值增长到最大值后，是否允许继续循环（即仍从 START WITH 开始）使用 Sequence 值。若未指定，则默认值为 NOCYCLE。	Simple Sequence

注意:

- Group Sequence 是非连续的。**START WITH** 参数对于 Group Sequence 仅具有指导意义，**Group Sequence 不会严格按照该参数作为起始值，但是保证起始值比该参数大。**
- 对于 Simple Sequence，如果修改 Sequence 时指定了 **START WITH**，则会立即生效，下次取 Sequence 值时会从新的 **START WITH** 值开始。比如原先 Sequene 增长到 100 了，这时把 **START WITH** 值改成了200，那么下一次获取的 Sequence 值就是从 200 开始的。
- 修改 **START WITH** 的参数值，需要仔细评估已经产生的 Sequence 值，以及生成新 Sequence 值的速度，防止产生冲突。如非必要，请谨慎修改 **START WITH** 参数值。

将 Simple Sequence seq2 的起始值改为 3000，步长改为 5，最大值改为 1000000，增长到最大值后改为继续循环。

```
mysql> ALTER SEQUENCE seq2 START WITH 3000 INCREMENT BY 5 MAXVALUE
 1000000 CYCLE;
```

```
Query OK, 1 row affected (0.01 sec)
```

不同类型 Sequence 间的转换

- 通过**ALTER SEQUENCE**的CHANGE TO <sequence_type>子句实现；
- **ALTER SEQUENCE** 如果指定了 CHANGE TO子句，则强制必须加上 **START WITH** 参数，避免忘记指定起始值而造成取值时得到重复值；若没有 CHANGE TO（可选参数），则不强制。

示例：

将 Group Sequence 转换为 Simple Sequence。

```
mysql> ALTER SEQUENCE seq1 CHANGE TO SIMPLE START WITH 1000000;
Query OK, 1 row affected (0.02 sec)
```

3. 删除Sequence

   语法：

   ```
   DROP SEQUENCE <name>
   ```

   ```
   mysql> DROP SEQUENCE seq3;
   Query OK, 1 row affected (0.02 sec)
   ```

4. 查询Sequence

   语法：

   ```
   DROP SEQUENCE < sequence name >
   ```

   结果集中的 TYPE 列，显示的是 Sequence 类型的缩写。

   ```
   mysql> SHOW SEQUENCES;
   +------------+--------------------+-------------+------------+---------------------+-------+--------+
   | NAME       | VALUE              | INCREMENT_BY | START_WITH | MAX_VALUE
    | CYCLE | TYPE   |
   +------------+--------------------+-------------+------------+---------------------+-------+--------+
   | AUTO_SEQ_1 | 91820513           | 1           | 91820200   | 9223372036854775807 | N
       | SIMPLE |
   | AUTO_SEQ_4 | 91820200           | 2           | 1000       | 9223372036854775807 | Y     | SIMPLE |
   | seq_test   | N/A                | N/A         | N/A        | N/A                 | N/A   | TIME   |
   | AUTO_SEQ_2 | 100000             | N/A         | N/A        | N/A                 | N/A   | GROUP  |
   | AUTO_SEQ_3 | 200000             | N/A         | N/A        | N/A                 | N/A   | GROUP  |
   +------------+--------------------+-------------+------------+---------------------+-------+--------+
   5 rows in set (0.01 sec)
   ```

5. 获取显示Sequence的值

语法：

```
< sequence name >.NEXTVAL
```

```
SELECT sample_seq.nextVal FROM dual;
+--------------------+
| SAMPLE_SEQ.NEXTVAL |
+--------------------+
|             101001 |
+--------------------+
1 row in set (0.04 sec)
```

或者可以把这个 SAMPLE_SEQ.nextVal 当做一个值写入 SQL 中：

```
mysql> INSERT INTO some_users (name,address,gmt_create,gmt_modified,intro) VALUES
 ('sun',SAMPLE_SEQ.nextVal,now(),now(),'aa');
Query OK, 1 row affected (0.01 sec)
```

注意: 如果建表时已经指定了 **AUTO_INCREMENT** 参数，insert 时不需要指定自增列，让 DRDS 自动维护。

22.5.3.2 Sequence隐式用法

在为主键定义 **AUTO_INCREMENT** 后，Sequence 可以用于自动填充主键，由 DRDS 自动维护。

CREATE TABLE

扩展标准建表语法，增加了自增列的 Sequence 类型，如果未指定类型关键字，则默认类型为 GROUP。DRDS 自动创建的跟表相关联的 Sequence 名称，都是以 AUTO_SEQ_ 为前缀，后面加上表名。

```
CREATE TABLE <name> (
  <column> ... AUTO_INCREMENT [ BY GROUP | SIMPLE | TIME ],
  <column definition>,
  ...
) ... AUTO_INCREMENT=<start value>
```

SHOW CREATE TABLE

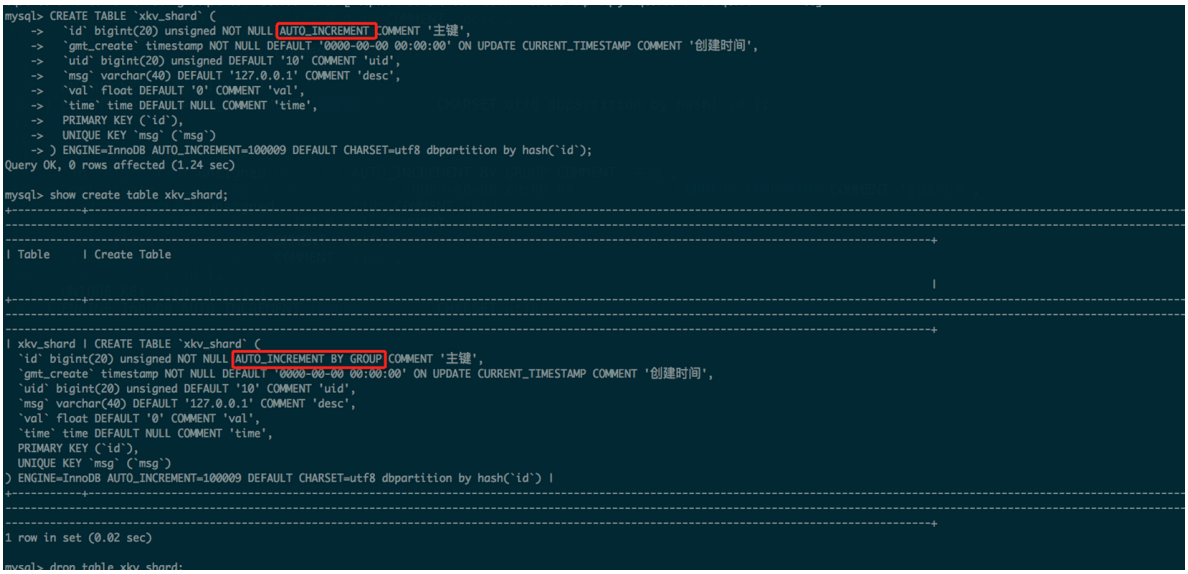
当表为拆分表或者广播表时，显示自增列 Sequence 的类型。

```
SHOW CREATE TABLE <name>
```

示例

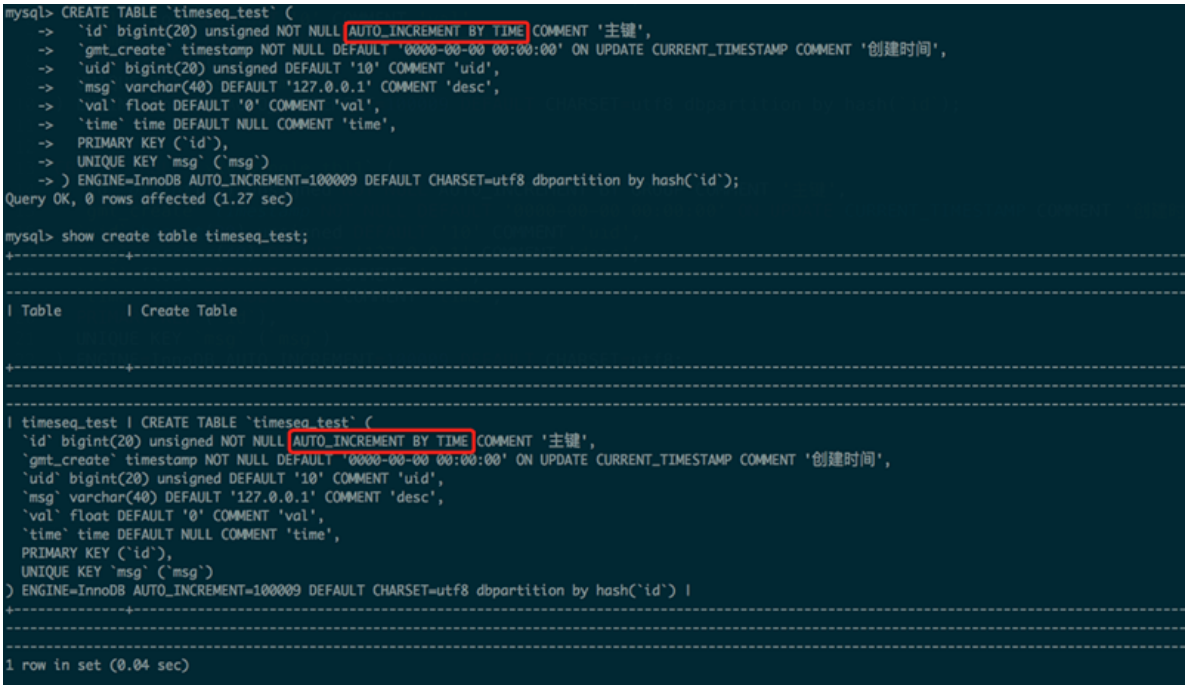
- 建表时指定 **AUTO_INCREMENT**，但是没有指定 Sequence 类型关键字，则默认使用 Group Sequence。

图 591: 示例一



- 建表时指定 **AUTO_INCREMENT**，同时指定使用Time-based Sequence作为主键值。

图 592: 示例二



ALTER TABLE

暂不支持通过**ALTER TABLE**来修改对应Sequence的类型，但可修改起始值。如果想要修改表中带的隐式 Sequence 的类型，需要通过**SHOW SEQUENCES** 指令查找出Sequence的具体名称和类型，然后再用**ALTER SEQUENCE**指令去修改。

注意: 使用 DRDS Sequence 后，请谨慎修改 **AUTO_INCREMENT** 的起始值（仔细评估已经产生的 Sequence 值，以及生成新 Sequence 值的速度，防止产生冲突）。

22.5.3.3 Sequence限制及注意事项

本文介绍Sequence操作的限制及注意事项。

限制与注意事项

- Time-based Sequence 用于表中自增列时，该列必须使用 BIGINT 类型；
- 转换 Sequence 类型时，必须指定 **START WITH** 起始值；
- 对于 DRDS 仅连接一个MySQL(RDS)的场景（即所有表都是单库单表），那么执行 Insert 时，DRDS 会直接下推，绕过优化器中分配 Sequence 值的部分。此时 `insert into tab values（seq.nextval, ...)`这种用法不支持，建议使用 MySQL 自增列代替。

如何处理主键冲突

比如直接在MySQL(RDS)中写入了数据，而对应的主键值不是 DRDS 生成的 Sequence 值，那么后续让 DRDS 自动生成主键写入数据库，可能会和这些数据发生主键冲突，可以通过以下步骤解决问题：

1. 通过 DRDS 指定 SQL 来查看当前已有 Sequence。**AUTO_SEQ_** 开头的 Sequence 是隐式 Sequence（创建表时加上 AUTO_INCREMENT 参数的表产生的 Sequence）：

```
mysql> SHOW SEQUENCES;
+------------------------+-------+-------------+------------+-----------+-------+-------+
| NAME                   | VALUE | INCREMENT_BY | START_WITH | MAX_VALUE | CYCLE | TYPE  |
+------------------------+-------+-------------+------------+-----------+-------+-------+
| AUTO_SEQ_timeseq_test  | N/A   | N/A          | N/A        | N/A       | N/A   | TIME  |
| AUTO_SEQ_xkv_shard_tbl1 | 0     | N/A          | N/A        | N/A       | N/A   | GROUP |
| AUTO_SEQ_xkv_shard     | 0     | N/A          | N/A        | N/A       | N/A   | GROUP |
+------------------------+-------+-------------+------------+-----------+-------+-------+
3 rows in set (0.04 sec)
```

2. 比如 t_item 表有冲突，并且 t_item 表主键是 ID，那么从 DRDS 获取这个表最大主键值：

```
mysql> SELECT MAX(id) FROM t_item;
+-----------+
| max(id)   |
+-----------+
| 8231 |
+-----------+
1 row in set (0.01 sec)
```

3. 更新 DRDS Sequence 表中对应的值，这里更新成比 8231 要大的值，比如 9000，更新完成后，后续插入语句生成的自增主键将不再报错：

```
mysql> ALTER SEQUENCE AUTO_SEQ_USERS START WITH 9000;
```

```
Query OK, 1 row affected (0.01 sec)
```

22.5.4 全表扫描

DRDS在全表扫描时支持聚合函数进行统计汇总。由于全表扫描相当消耗性能，因此默认为关闭状态，如需使用需要手动打开。

具体的配置方式请参考数据表管理，找到对应的表后进行配置。

- 如果目标表没有分库分表，那么DRDS可以支持任何聚合函数，因为实际上 DRDS 是直接把原SQL传递到后端 MySQL 执行。
- 非全表扫描：SQL语句在经过 DRDS 路由后，直接发送到后端单个 MySQL 库上执行。如果拆分键在 WHERE 条件中都是 = 关系，常会出现这种情况。在非全表扫描的情形下，同样可以支持任何聚合函数。
- 全表扫描：目前支持的聚合函数有 COUNT、MAX、MIN、SUM。同时支持 LIKE、ORDER BY 与 LIMIT 语法，但是不支持 GROUP BY 语法。
- 并行全表扫描：如果希望直接从所有库上面dump数据到其他地方，DRDS也提供了方法，可以感知到当前下层有多少个库，并允许单独针对这些库进行操作。

查看当前的库表个数：

```
mysql> show topology from users\G
*************************** 1. row ******************
       ID: 0
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS
TABLE_NAME: users_00
*************************** 2. row ***************
       ID: 1
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS
TABLE_NAME: users_01
*************************** 3. row ******************
       ID: 2
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS
TABLE_NAME: users_02
*************************** 4. row ******************
       ID: 3
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0001_RDS
TABLE_NAME: users_03
*************************** 5. row ***************
       ID: 4
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0001_RDS
TABLE_NAME: users_04
*************************** 6. row ****************
       ID: 5
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0001_RDS
TABLE_NAME: users_05
*************************** 7. row ****************
       ID: 6
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0002_RDS
TABLE_NAME: users_06
*************************** 8. row ****************
```

```
       ID: 7
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0002_RDS
TABLE_NAME: users_07
*************************** 9. row ***************
       ID: 8
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0002_RDS
TABLE_NAME: users_08
*************************** 10. row ***************
       ID: 9
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0003_RDS
TABLE_NAME: users_09
*************************** 11. row ***************
       ID: 10
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0003_RDS
TABLE_NAME: users_10
*************************** 12. row ***************
       ID: 11
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0003_RDS
TABLE_NAME: users_11
*************************** 13. row ***************
       ID: 12
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0004_RDS
TABLE_NAME: users_12
*************************** 14. row ***************
       ID: 13
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0004_RDS
TABLE_NAME: users_13
*************************** 15. row ***************
       ID: 14
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0004_RDS
TABLE_NAME: users_14
*************************** 16. row ***************
       ID: 15
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0005_RDS
TABLE_NAME: users_15
*************************** 17. row ***************
       ID: 16
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0005_RDS
TABLE_NAME: users_16
*************************** 18. row ***************
       ID: 17
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0005_RDS
TABLE_NAME: users_17
*************************** 19. row ***************
       ID: 18
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0006_RDS
TABLE_NAME: users_18
*************************** 20. row ***************
       ID: 19
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0006_RDS
TABLE_NAME: users_19
*************************** 21. row ***************
       ID: 20
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0006_RDS
TABLE_NAME: users_20
*************************** 22. row ***************
       ID: 21
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0007_RDS
TABLE_NAME: users_21
*************************** 23. row ***************
       ID: 22
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0007_RDS
```

```
TABLE_NAME: users_22
*************************** 24. row ******************
       ID: 23
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0007_RDS
TABLE_NAME: users_23
24 rows in set (0.01 sec)
```

针对topology进行单表遍历：

```
/*TDDL:node='DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS'*/  select  *
 from users\G
```

这条SQL语句查询分库**DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS**中的逻辑表**users**的所有分表。

注意: 这个库的个数可能会因为进行扩容等情况随时发生变化。DRDS不保证这些GROUPS的名字永远固定不变，所以请每次都先运行`show topology from table`语句来获得最新的表拓扑。

22.5.5 读写分离

本文从DRDS数据库的使用角度介绍读写分离。控制台操作请参见设置读写分离。

在控制台设置读流量之后，备库就会有读流量分配进来。登录到DRDS数据库后，执行`SHOW NODE`指令查看实际读流量分布。

页面上的读权重修改，会准实时影响这个读流量的分布，如：

```
mysql> show node\G
*************************** 1. row ******************
           ID: 0
         NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS
  MASTER_READ_COUNT: 170
   SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 2. row ******************
           ID: 1
         NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0001_RDS
  MASTER_READ_COUNT: 11
   SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 3. row ******************
           ID: 2
         NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0002_RDS
  MASTER_READ_COUNT: 11
   SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 4. row ******************
           ID: 3
         NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0003_RDS
  MASTER_READ_COUNT: 11
```

```
   SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 5. row ******************
            ID: 4
          NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0004_RDS
  MASTER_READ_COUNT: 11
   SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 6. row ******************
            ID: 5
          NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0005_RDS
  MASTER_READ_COUNT: 11
   SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 7. row ******************
            ID: 6
          NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0006_RDS
  MASTER_READ_COUNT: 11
   SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 8. row ******************
            ID: 7
          NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0007_RDS
  MASTER_READ_COUNT: 11
   SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
8 rows in set (0.06 sec)
```

查询结果中，各列分别表示一组读写库一段时间内的主库读计数、从库读计数、主库读占比、从库读占比。一组读写库和一组读写实例(控制台展示的) 是多对一的关系，命令行查到的信息比控制台更加细致(数据分片级别)。

如果希望特定的SQL语句强制到主库或者备库，可以通过DRDS提供的HINT来完成，如：

```
/*TDDL:SLAVE*/select * from users where id=123
```

那么这条SQL语句会路由到备库执行。



说明:

- 如果MySQL备实例不可用，需要手动将其读比例设置为0，否则流量会继续走到改节点并且失败。
- 读写分离是以DRDS数据库为基本单位的。如果同一个MySQL只读实例在多个DRDS数据库上使用，需要在每个DRDS数据库上都将其权重设置为0。

22.5.6 DRDS HINT

DRDS的HINT是DRDS提供的一种特殊的SQL语法，它允许您通过相关的语法影响SQL的默认执行方式。

对于没有使用DRDS HINT的SQL，DRDS会经过SQL解析、SQL优化、执行这三个步骤，而DRDS HINT则会绕过SQL优化器执行，从而加快SQL的执行效率。

例如，已知目的数据在某个分库的分表中了，需要直接将SQL下发到该分库执行，可以使用DRDS HINT语法来完成：

```
/*TDDL:node in(分库名)*/SELECT * FROM 分表
```



注意: 由于DRDS HINT语句是以MySQL注释的形式使用的，在使用MySQL官方命令行客户端执行带有DRDS HINT的SQL时，请在登录命令中加上-c 参数。否则，该客户端会将注释删掉后发送到服务端执行，导致DRDS自定义注释失效。

下文为DRDS HINT使用示例。

- 强制SQL读写主库：

  ```
  /*TDDL:MASTER*/select * from users;
  ```

- 强制读SQL在备库执行：

  ```
  /*TDDL:SLAVE*/select * from users;
  ```

  说明: 此HINT仅当SQL是读操作才会有效。执行时会从备库(只读实例)中按照权重随机选取一个进行读取操作。若备库（只读实例）不存在时，不会报错，会选取主库。

- 指定特定节点执行SQL：

  ```
  /*TDDL:node in ('DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0005_RDS',’ ...’ )*/delete from users;
  ```

  说明: node后面跟节点的名字。 节点名字可以通过show node指令获取。

- 调整SQL的超时时间：

DRDS到MySQL之间会有超时时间。对于一些必须做的慢查询，可以通过以下HINT调整这个SQL的超时时间。

```
/*TDDL:socket_timeout=10000*/select count(*) from users;
```

说明： **socket_timeout**的单位是毫秒。如果socket_timeout=0，则该SQL不会超时。

- 将SQL下发到所有分库执行：

```
/*TDDL:scan*/show tables;
```

说明: 比如有4个分库，则show tables会在每个分库上执行一次。

22.5.7 DRDS自定义控制指令

为方便您使用和维护DRDS ，DRDS 提供特有的辅助语句。

语法说明：[] 中为用户提供的标识符，( ) 中为可选内容。此外，此文档适用于当前版本，如一些语句不可用，说明使用的版本过低。

22.5.7.1 一般帮助语句

SHOW HELP语句用于获取DRDS 提供的维护SQL。

对于不清楚DRDS 提供的维护SQL，可以通过此语句获取。

使用方式如下：

```
mysql> show help\G
*************************** 1. row ******************
  STATEMENT: show rule
DESCRIPTION: Report all table rule
    EXAMPLE:
*************************** 2. row ******************
  STATEMENT: show rule from TABLE
DESCRIPTION: Report table rule
    EXAMPLE: show rule from user
*************************** 3. row ******************
  STATEMENT: show full rule from TABLE
DESCRIPTION: Report table full rule
    EXAMPLE: show full rule from user
*************************** 4. row ******************
  STATEMENT: show topology from TABLE
DESCRIPTION: Report table physical topology
    EXAMPLE: show topology from user
*************************** 5. row ******************
  STATEMENT: show partitions from TABLE
DESCRIPTION: Report table dbPartition or tbPartition columns
    EXAMPLE: show partitions from user
*************************** 6. row ******************
```

```
  STATEMENT: show broadcasts
DESCRIPTION: Report all broadcast tables
    EXAMPLE:
*************************** 7. row ******************
  STATEMENT: show datasources
DESCRIPTION: Report all partition db threadPool info
    EXAMPLE:
*************************** 8. row ******************
  STATEMENT: show node
DESCRIPTION: Report master/slave read status
    EXAMPLE:
*************************** 9. row ******************
  STATEMENT: show slow
DESCRIPTION: Report top 100 slow sql
    EXAMPLE:
*************************** 10. row ******************
  STATEMENT: show physical_slow
DESCRIPTION: Report top 100 physical slow sql
    EXAMPLE:
*************************** 11. row ******************
  STATEMENT: clear slow
DESCRIPTION: Clear slow data
    EXAMPLE:
*************************** 12. row ******************
  STATEMENT: trace SQL
DESCRIPTION: Start trace sql, use show trace to print profiling data
    EXAMPLE: trace select count(*) from user; show trace
*************************** 13. row ******************
  STATEMENT: show trace
DESCRIPTION: Report sql execute profiling info
    EXAMPLE:
*************************** 14. row ******************
  STATEMENT: explain SQL
DESCRIPTION: Report sql plan info
    EXAMPLE: explain select count(*) from user
*************************** 15. row ******************
  STATEMENT: explain detail SQL
DESCRIPTION: Report sql detail plan info
    EXAMPLE: explain detail select count(*) from user
*************************** 16. row ******************
  STATEMENT: explain execute SQL
DESCRIPTION: Report sql on physical db plan info
    EXAMPLE: explain execute select count(*) from user
*************************** 17. row ******************
  STATEMENT: show sequences
DESCRIPTION: Report all sequences status
    EXAMPLE:
*************************** 18. row ******************
  STATEMENT: create sequence NAME [start with COUNT]
DESCRIPTION: Create sequence
    EXAMPLE: create sequence test start with 0
*************************** 19. row ******************
  STATEMENT: alter sequence NAME [start with COUNT]
DESCRIPTION: Alter sequence
    EXAMPLE: alter sequence test start with 100000
*************************** 20. row ******************
  STATEMENT: drop sequence NAME
DESCRIPTION: Drop sequence
    EXAMPLE: drop sequence test
```

```
20 rows in set (0.00 sec)
```

22.5.7.2 查看规则和节点拓扑语句

本文介绍用于查看路由规则及节点拓扑的语句。

请参考以下示例。

- SHOW RULE语句用于查看对应逻辑库下，使用分库分表的表所采用的路由规则。可用于查看所有采用分表的逻辑表路由规则，包含是否为广播表，分库分表方案，分库分表数等信息。

```
mysql> show rule\G
*************************** 1. row ******************
               ID: 2
        TABLE_NAME: users
         BROADCAST: 0
   DB_PARTITION_KEY: id
DB_PARTITION_POLICY: hash
 DB_PARTITION_COUNT: 8
   TB_PARTITION_KEY: id
TB_PARTITION_POLICY: hash
 TB_PARTITION_COUNT: 3
1 rows in set (0.00 sec)
```

- SHOW RULE FROM [tablename]语句用于查看对应逻辑库下，表的分库分表规则。此语句类似SHOW RULE，但是可以查看所有表的信息，不单是分库分表的表。

```
mysql> show rule from users\G
*************************** 1. row ******************
               ID: 0
        TABLE_NAME: users
         BROADCAST: 0
   DB_PARTITION_KEY: id
DB_PARTITION_POLICY: hash
 DB_PARTITION_COUNT: 8
   TB_PARTITION_KEY: id
TB_PARTITION_POLICY: hash
 TB_PARTITION_COUNT: 3
1 row in set (0.01 sec)
```

- SHOW FULL RULE FROM [tablename]语句用于获取更多规则信息，如可否全分表扫描，分库、分表具体规则等。此语句比SHOW RULE 更加详细，包含是否可全表扫描，分库、分表具体规则，但对于非分库分表的表，无法获取对应信息。

```
mysql> show full rule from users\G
*************************** 1. row ******************
                ID: 0
          TABLE_NAME: users
           BROADCAST: 0
          JOIN_GROUP: NULL
ALLOW_FULL_TABLE_SCAN: 1
                DB_NAME_PATTERN:   DRDS_TEST_1473471355140LRPRDRDS_TEST_VTL
A_{0000}_RDS
        DB_RULES_STR: ((#id,1,24#).longValue().abs() % 24).intdiv(3)
     TB_NAME_PATTERN: users_{00}
```

```
        TB_RULES_STR: ((#id,1,24#).longValue().abs() % 24)
      PARTITION_KEYS: id
             DEFAULT_DB_INDEX:   DRDS_TEST_1473471355140LRPRDRDS_TEST_VTL
A_0000_RDS
1 row in set (0.00 sec)
```

- SHOW TOPOLOGY FROM [tablename]语句用于查看分库与分表对应关系，适用于所有的逻辑表，即包含分表的逻辑表，也包含不分表的逻辑表。

```
mysql> show topology from users\G
*************************** 1. row ******************
       ID: 0
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS
TABLE_NAME: users_00
*************************** 2. row ******************
       ID: 1
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS
TABLE_NAME: users_01
*************************** 3. row ******************
       ID: 2
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS
TABLE_NAME: users_02
*************************** 4. row ******************
       ID: 3
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0001_RDS
TABLE_NAME: users_03
*************************** 5. row ******************
       ID: 4
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0001_RDS
TABLE_NAME: users_04
*************************** 6. row ******************
       ID: 5
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0001_RDS
TABLE_NAME: users_05
*************************** 7. row ******************
       ID: 6
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0002_RDS
TABLE_NAME: users_06
*************************** 8. row ******************
       ID: 7
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0002_RDS
TABLE_NAME: users_07
*************************** 9. row ******************
       ID: 8
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0002_RDS
TABLE_NAME: users_08
*************************** 10. row ******************
       ID: 9
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0003_RDS
TABLE_NAME: users_09
*************************** 11. row ******************
       ID: 10
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0003_RDS
TABLE_NAME: users_10
*************************** 12. row ******************
       ID: 11
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0003_RDS
TABLE_NAME: users_11
*************************** 13. row ******************
       ID: 12
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0004_RDS
```

```
TABLE_NAME: users_12
*************************** 14. row ***************
      ID: 13
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0004_RDS
TABLE_NAME: users_13
*************************** 15. row ***************
      ID: 14
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0004_RDS
TABLE_NAME: users_14
*************************** 16. row ***************
      ID: 15
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0005_RDS
TABLE_NAME: users_15
*************************** 17. row ***************
      ID: 16
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0005_RDS
TABLE_NAME: users_16
*************************** 18. row ***************
      ID: 17
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0005_RDS
TABLE_NAME: users_17
*************************** 19. row ***************
      ID: 18
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0006_RDS
TABLE_NAME: users_18
*************************** 20. row ***************
      ID: 19
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0006_RDS
TABLE_NAME: users_19
*************************** 21. row ***************
      ID: 20
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0006_RDS
TABLE_NAME: users_20
*************************** 22. row ***************
      ID: 21
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0007_RDS
TABLE_NAME: users_21
*************************** 23. row ***************
      ID: 22
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0007_RDS
TABLE_NAME: users_22
*************************** 24. row ***************
      ID: 23
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0007_RDS
TABLE_NAME: users_23
24 rows in set (0.00 sec)
```

- SHOW PARTITIONS FROM [tablename]语句用于查看查看分库分表键，针对所有的逻辑表。

```
mysql> show partitions from users\G
*************************** 1. row ***************
KEYS: id
```

```
1 row in set (0.00 sec)
```

- SHOW BROADCASTS语句用于查看广播表信息。

```
mysql> show broadcasts;
```

- SHOW DATASOURCES用于查看底层存储信息，包含schema，数据库分组名，JDBC 信息，用户名，底层存储类型，读写权重等。

```
mysql> show datasources\G
*************************** 1. row ******************
          ID: 0
      SCHEMA: drds_test_1473471355140lrpr
        NAME: rm-bp1s876tt70q0wgus_drds_test_vtla_0000_qgmr_681
  GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS
         URL:  jdbc:mysql://rm-xxxx.mysql.rds.aliyuncs.com:3306/drds_test_vtla_0000?
autoReconnect=false&connectTimeout=3000&rewriteBatchedStatem
ents=true&allowMultiQueries=true&socketTimeout=900000&clobberStreamingResults=true&
netTimeoutForStreamingResults=0&failOverReadOnly=false&readOnlyPropagatesTo
Server=false&characterEncoding=utf8
        USER: zbfunaik0
        TYPE: mysql
        INIT: 0
         MIN: 1
         MAX: 30
 IDLE_TIMEOUT: 1
    MAX_WAIT: 5000
 ACTIVE_COUNT: 0
POOLING_COUNT: 1
        ATOM: rm-bp1s876tt70q0wgus_drds_test_vtla_0000_qgmr
  READ_WEIGHT: 10
 WRITE_WEIGHT: 10
*************************** 2. row ******************
          ID: 1
      SCHEMA: drds_test_1473471355140lrpr
        NAME: rm-bp1s876tt70q0wgus_drds_test_vtla_0001_qgmr_682
  GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0001_RDS
         URL:  jdbc:mysql://rm-xxxx.mysql.rds.aliyuncs.com:3306/drds_test_vtla_0001?
autoReconnect=false&connectTimeout=3000&rewriteBatchedStatem
ents=true&allowMultiQueries=true&socketTimeout=900000&clobberStreamingResults=true&
netTimeoutForStreamingResults=0&failOverReadOnly=false&readOnlyPropagatesTo
Server=false&characterEncoding=utf8
        USER: zbfunaik0
        TYPE: mysql
        INIT: 0
         MIN: 1
         MAX: 30
 IDLE_TIMEOUT: 1
    MAX_WAIT: 5000
 ACTIVE_COUNT: 0
POOLING_COUNT: 1
        ATOM: rm-bp1s876tt70q0wgus_drds_test_vtla_0001_qgmr
  READ_WEIGHT: 10
 WRITE_WEIGHT: 10
*************************** 3. row ******************
          ID: 2
      SCHEMA: drds_test_1473471355140lrpr
        NAME: rm-bp1s876tt70q0wgus_drds_test_vtla_0002_qgmr_683
  GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0002_RDS
```

```
                    URL:  jdbc:mysql://rm-xxxx.mysql.rds.aliyuncs.com:3306/drds_test_vtla_0002?
autoReconnect=false&connectTimeout=3000&rewriteBatchedStatem
ents=true&allowMultiQueries=true&socketTimeout=900000&clobberStreamingResults=true&
netTimeoutForStreamingResults=0&failOverReadOnly=false&readOnlyPropagatesTo
Server=false&characterEncoding=utf8
        USER: zbfunaik0
        TYPE: mysql
        INIT: 0
         MIN: 1
         MAX: 30
 IDLE_TIMEOUT: 1
    MAX_WAIT: 5000
 ACTIVE_COUNT: 0
POOLING_COUNT: 1
        ATOM: rm-bp1s876tt70q0wgus_drds_test_vtla_0002_qgmr
  READ_WEIGHT: 10
 WRITE_WEIGHT: 10
*************************** 4. row ******************
          ID: 3
      SCHEMA: drds_test_1473471355140lrpr
        NAME: rm-bp1s876tt70q0wgus_drds_test_vtla_0003_qgmr_684
  GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0003_RDS
                    URL:  jdbc:mysql://rm-xxxx.mysql.rds.aliyuncs.com:3306/drds_test_vtla_0003?
autoReconnect=false&connectTimeout=3000&rewriteBatchedStatem
ents=true&allowMultiQueries=true&socketTimeout=900000&clobberStreamingResults=true&
netTimeoutForStreamingResults=0&failOverReadOnly=false&readOnlyPropagatesTo
Server=false&characterEncoding=utf8
        USER: zbfunaik0
        TYPE: mysql
        INIT: 0
         MIN: 1
         MAX: 30
 IDLE_TIMEOUT: 1
    MAX_WAIT: 5000
 ACTIVE_COUNT: 0
POOLING_COUNT: 1
        ATOM: rm-bp1s876tt70q0wgus_drds_test_vtla_0003_qgmr
  READ_WEIGHT: 10
 WRITE_WEIGHT: 10
*************************** 5. row ******************
          ID: 4
      SCHEMA: drds_test_1473471355140lrpr
        NAME: rm-bp1i7td4edkd37x34_drds_test_vtla_0004_etnd_689
  GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0004_RDS
                  URL:  jdbc:mysql://rm-bxxxx.mysql.rds.aliyuncs.com:3306/drds_test_vtla_0004?
autoReconnect=false&connectTimeout=3000&rewriteBatchedStatem
ents=true&allowMultiQueries=true&socketTimeout=900000&clobberStreamingResults=true&
netTimeoutForStreamingResults=0&failOverReadOnly=false&readOnlyPropagatesTo
Server=false&characterEncoding=utf8
        USER: zbfunaik0
        TYPE: mysql
        INIT: 0
         MIN: 1
         MAX: 30
 IDLE_TIMEOUT: 1
    MAX_WAIT: 5000
 ACTIVE_COUNT: 0
POOLING_COUNT: 1
        ATOM: rm-bp1i7td4edkd37x34_drds_test_vtla_0004_etnd
  READ_WEIGHT: 10
 WRITE_WEIGHT: 10
*************************** 6. row ******************
```

```
          ID: 5
      SCHEMA: drds_test_1473471355140lrpr
        NAME: rm-bp1i7td4edkd37x34_drds_test_vtla_0005_etnd_690
  GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0005_RDS
         URL:  jdbc:mysql://rm-xxxx.mysql.rds.aliyuncs.com:3306/drds_test_vtla_0005?
autoReconnect=false&connectTimeout=3000&rewriteBatchedStatem
ents=true&allowMultiQueries=true&socketTimeout=900000&clobberStreamingResults=true&
netTimeoutForStreamingResults=0&failOverReadOnly=false&readOnlyPropagatesTo
Server=false&characterEncoding=utf8
        USER: zbfunaik0
        TYPE: mysql
        INIT: 0
         MIN: 1
         MAX: 30
 IDLE_TIMEOUT: 1
    MAX_WAIT: 5000
 ACTIVE_COUNT: 0
POOLING_COUNT: 1
        ATOM: rm-bp1i7td4edkd37x34_drds_test_vtla_0005_etnd
  READ_WEIGHT: 10
 WRITE_WEIGHT: 10
*************************** 7. row ******************
          ID: 6
      SCHEMA: drds_test_1473471355140lrpr
        NAME: rm-bp1i7td4edkd37x34_drds_test_vtla_0006_etnd_691
  GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0006_RDS
         URL:  jdbc:mysql://rm-xxxx.mysql.rds.aliyuncs.com:3306/drds_test_vtla_0006?
autoReconnect=false&connectTimeout=3000&rewriteBatchedStatem
ents=true&allowMultiQueries=true&socketTimeout=900000&clobberStreamingResults=true&
netTimeoutForStreamingResults=0&failOverReadOnly=false&readOnlyPropagatesTo
Server=false&characterEncoding=utf8
        USER: zbfunaik0
        TYPE: mysql
        INIT: 0
         MIN: 1
         MAX: 30
 IDLE_TIMEOUT: 1
    MAX_WAIT: 5000
 ACTIVE_COUNT: 0
POOLING_COUNT: 1
        ATOM: rm-bp1i7td4edkd37x34_drds_test_vtla_0006_etnd
  READ_WEIGHT: 10
 WRITE_WEIGHT: 10
*************************** 8. row ******************
          ID: 7
      SCHEMA: drds_test_1473471355140lrpr
        NAME: rm-bp1i7td4edkd37x34_drds_test_vtla_0007_etnd_692
  GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0007_RDS
         URL:  jdbc:mysql://rm-xxxx.mysql.rds.aliyuncs.com:3306/drds_test_vtla_0007?
autoReconnect=false&connectTimeout=3000&rewriteBatchedStatem
ents=true&allowMultiQueries=true&socketTimeout=900000&clobberStreamingResults=true&
netTimeoutForStreamingResults=0&failOverReadOnly=false&readOnlyPropagatesTo
Server=false&characterEncoding=utf8
        USER: zbfunaik0
        TYPE: mysql
        INIT: 0
         MIN: 1
         MAX: 30
 IDLE_TIMEOUT: 1
    MAX_WAIT: 5000
 ACTIVE_COUNT: 0
POOLING_COUNT: 1
```

```
        ATOM: rm-bp1i7td4edkd37x34_drds_test_vtla_0007_etnd
  READ_WEIGHT: 10
 WRITE_WEIGHT: 10
8 rows in set (0.00 sec)
```

- SHOW NODE用于查看物理库的读写信息，包括存储层分库信息，读写比例分配等。

```
mysql> show node\G
*************************** 1. row ******************
            ID: 0
          NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS
 MASTER_READ_COUNT: 212
  SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 2. row ******************
            ID: 1
          NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0001_RDS
 MASTER_READ_COUNT: 29
  SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 3. row ******************
            ID: 2
          NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0002_RDS
 MASTER_READ_COUNT: 29
  SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 4. row ******************
            ID: 3
          NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0003_RDS
 MASTER_READ_COUNT: 29
  SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 5. row ******************
            ID: 4
          NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0004_RDS
 MASTER_READ_COUNT: 29
  SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 6. row ******************
            ID: 5
          NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0005_RDS
 MASTER_READ_COUNT: 29
  SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 7. row ******************
            ID: 6
          NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0006_RDS
 MASTER_READ_COUNT: 29
  SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 8. row ******************
            ID: 7
          NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0007_RDS
 MASTER_READ_COUNT: 29
  SLAVE_READ_COUNT: 0
```

```
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
8 rows in set (0.02 sec)
```

22.5.7.3 SQL调优语句

本文介绍SQL调优语句的用法。

请参考以下示例。

- SHOW SLOW (WHERE)语句用于查看最近的100条DRDS 慢SQL。包含DRDS 执行IP，具体慢SQL和执行时间。单位为毫秒，执行时间超过 1秒的SQL语句会被记录为慢SQL。

  此外支持添加WHERE 条件和LIMIT 限制。

```
mysql> show slow where execute_time >1000 limit 1\G
*************************** 1. row ******************
    TRACE_ID: 9e0d7808b803000
        HOST: 10.200.248.14
  START_TIME: 2016-09-11 22:16:02.721
EXECUTE_TIME: 1436
  AFFECT_ROW: 0
                 SQL: create table users(id bigint(20) unsigned NOT NULL primary key
 auto_increment,name varchar(32),address varchar(20),gmt_create datetime,gmt_modified
 datetime, intro varchar(255)) dbpartition by hash(id) tbpartition by hash(id) tbpartitions 3
1 row in set (0.01 sec)
```

- SHOW PHYSICAL_SLOW (WHERE)语句用于查看对应底层存储的最近100条慢SQL。

  SHOW SLOW 与 SHOW PHYSICAL_SLOW 的区别是针对的层次不一样，一个是DRDS 层级，一个是物理存储层级。包含物理库分组名，DBKEY等内部信息）。

  注意其中的下划线。

  此外支持添加WHERE 条件和LIMIT 限制。

```
mysql> show physical_slow\G
Empty set (0.02 sec)
```

- CLEAR SLOW用于清理慢SQL信息，针对所有慢SQL信息，DRDS和底层存储的都会清除。

```
mysql> clear slow;
Query OK, 0 rows affected (0.00 sec)
```

- TRACE [SQL] 和SHOW TRACE组合语句用于查看具体SQL的执行情况。

  TRACE [SQL]和SHOW TRACE要结合使用，对分表与不分表都可以使用。包含数据库分组，物理SQL等信息。

```
mysql> trace select 1\G
*************************** 1. row ******************
1: 1
1 row in set (0.00 sec)
```

```
mysql> show trace\G
*************************** 1. row ******************
                  ID: 0
           TIMESTAMP: 0.000
                TYPE: Optimize
          GROUP_NAME: DRDS
          DBKEY_NAME: DRDS
       TIME_COST(MS): 0
CONNECTION_TIME_COST(MS): 0.00
                ROWS: 0
           STATEMENT: select 1
              PARAMS: NULL
*************************** 2. row ******************
                  ID: 1
           TIMESTAMP: 0.360
                TYPE: Query
      GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS
         DBKEY_NAME: rm-bp1s876tt70q0wgus_drds_test_vtla_0000_qgmr
       TIME_COST(MS): 2
CONNECTION_TIME_COST(MS): 0.88
                ROWS: 1
           STATEMENT: select 1
              PARAMS: NULL
2 rows in set (0.00 sec)
```

- EXPLAIN [SQL]语句用于查看语句对应的分库，物理语句，和整体参数。

  不同于底层存储的EXPLAIN 语句，要实现底层存储EXPLAIN 功能，要使用EXPLAIN EXECUTE [SQL] 语句。

```
mysql> explain select * from t_users\G
                           *************************** 1. row ******************
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS
       SQL: select * from t_users
    PARAMS: {}
1 row in set (0.01 sec)
```

- EXPLAIN DETAIL [SQL]语句用于查询语句的DRDS 执行信息。针对所有SQL，返回信息包含执行语句类型、并发程度、返回列信息、物理表、库分组等。

```
mysql> explain detail select * from t_users\G
*************************** 1. row ******************
GROUP_NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS
       SQL: Query
    queryConcurrency:SEQUENTIAL
    sql: select * from t_users
    executeOn:DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS

    PARAMS: NULL
1 row in set (0.00 sec)
```

- EXPLAIN EXECUTE [SQL]语句用于查看底层存储的执行计划，等同于MYSQL 的EXPLAIN 语句。

```
mysql> explain execute select * from t_users limit 1\G
*************************** 1. row ******************
```

```
           id: 1
  select_type: SIMPLE
        table: t_users
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
        Extra: NULL
1 row in set (0.00 sec)
```

- CHECK TABLE [name]语句用于对数据表进行检查。

  对于分表，检查底层表的一致性和完整性。

  对于所有表，检查表的存在性。

```
mysql> check table users\G
*************************** 1. row ******************
   TABLE: drds_test_1473471355140lrpr.users
      OP: check
MSG_TYPE: status
MSG_TEXT: OK
1 row in set (0.07 sec)
```

- SHOW TABLE STATUS (LIKE 'pattern' | WHERE expr)语句用于对数据表进行检查。

  该数据聚合了各个分片的数据。

  和DRDS的SCAN Hint结合，还可以查看每个分片的数据量（如果用MySQL命令行，启动时需要加上-c参数，否则Hint会被MySQL命令行过滤掉），使用方式如下。

```
mysql> /*TDDL:scan='users'*/show table status like 'users'\G
*************************** 1. row ******************
           Name: users_05
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 1
 Avg_row_length: 16384
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 9006
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
   Block_format: Original
*************************** 2. row ******************
           Name: users_04
         Engine: InnoDB
        Version: 10
```

```
     Row_format: Compact
           Rows: 1
 Avg_row_length: 16384
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 9005
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
  Block_format: Original
*************************** 3. row ******************
           Name: users_03
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 1
 Avg_row_length: 16384
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 9004
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
  Block_format: Original
*************************** 4. row ******************
           Name: users_11
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 1
 Avg_row_length: 16384
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 9012
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
  Block_format: Original
*************************** 5. row ******************
           Name: users_10
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 1
```

```
 Avg_row_length: 16384
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 9011
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
  Block_format: Original
*************************** 6. row ******************
          Name: users_09
        Engine: InnoDB
       Version: 10
    Row_format: Compact
          Rows: 1
 Avg_row_length: 16384
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 9010
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
  Block_format: Original
*************************** 7. row ******************
          Name: users_19
        Engine: InnoDB
       Version: 10
    Row_format: Compact
          Rows: 0
 Avg_row_length: 0
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 1
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
  Block_format: Original
*************************** 8. row ******************
          Name: users_18
        Engine: InnoDB
       Version: 10
    Row_format: Compact
          Rows: 0
 Avg_row_length: 0
    Data_length: 16384
```

```
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 1
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
   Block_format: Original
*************************** 9. row ******************
           Name: users_20
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 0
 Avg_row_length: 0
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 1
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
   Block_format: Original
*************************** 10. row *****************
           Name: users_23
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 0
 Avg_row_length: 0
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 1
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
   Block_format: Original
*************************** 11. row *****************
           Name: users_22
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 0
 Avg_row_length: 0
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
```

```
      Data_free: 0
 Auto_increment: 1
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
   Block_format: Original
*************************** 12. row ***************
          Name: users_21
        Engine: InnoDB
       Version: 10
    Row_format: Compact
          Rows: 0
 Avg_row_length: 0
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 1
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
   Block_format: Original
*************************** 13. row ***************
          Name: users_08
        Engine: InnoDB
       Version: 10
    Row_format: Compact
          Rows: 1
 Avg_row_length: 16384
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 9009
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
   Block_format: Original
*************************** 14. row ***************
          Name: users_07
        Engine: InnoDB
       Version: 10
    Row_format: Compact
          Rows: 1
 Avg_row_length: 16384
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 9008
```

```
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
   Block_format: Original
*************************** 15. row ***************
          Name: users_06
        Engine: InnoDB
       Version: 10
    Row_format: Compact
          Rows: 1
 Avg_row_length: 16384
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 9007
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
   Block_format: Original
*************************** 16. row ***************
          Name: users_02
        Engine: InnoDB
       Version: 10
    Row_format: Compact
          Rows: 1
 Avg_row_length: 16384
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 9003
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
   Block_format: Original
*************************** 17. row ***************
          Name: users_01
        Engine: InnoDB
       Version: 10
    Row_format: Compact
          Rows: 1
 Avg_row_length: 16384
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 9002
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
```

```
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
   Block_format: Original
*************************** 18. row ***************
           Name: users_00
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 1
 Avg_row_length: 16384
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 9001
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
   Block_format: Original
*************************** 19. row ***************
           Name: users_17
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 0
 Avg_row_length: 0
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 1
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
   Block_format: Original
*************************** 20. row ***************
           Name: users_16
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 0
 Avg_row_length: 0
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 1
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
```

```
       Checksum: NULL
 Create_options:
        Comment:
   Block_format: Original
*************************** 21. row ***************
          Name: users_15
        Engine: InnoDB
       Version: 10
    Row_format: Compact
          Rows: 0
 Avg_row_length: 0
   Data_length: 16384
Max_data_length: 0
  Index_length: 0
     Data_free: 0
 Auto_increment: 1
   Create_time: 2016-09-11 22:16:02
   Update_time: NULL
    Check_time: NULL
     Collation: utf8_general_ci
      Checksum: NULL
 Create_options:
        Comment:
   Block_format: Original
*************************** 22. row ***************
          Name: users_13
        Engine: InnoDB
       Version: 10
    Row_format: Compact
          Rows: 1
 Avg_row_length: 16384
   Data_length: 16384
Max_data_length: 0
  Index_length: 0
     Data_free: 0
 Auto_increment: 9014
   Create_time: 2016-09-11 22:16:02
   Update_time: NULL
    Check_time: NULL
     Collation: utf8_general_ci
      Checksum: NULL
 Create_options:
        Comment:
   Block_format: Original
*************************** 23. row ***************
          Name: users_12
        Engine: InnoDB
       Version: 10
    Row_format: Compact
          Rows: 1
 Avg_row_length: 16384
   Data_length: 16384
Max_data_length: 0
  Index_length: 0
     Data_free: 0
 Auto_increment: 9013
   Create_time: 2016-09-11 22:16:02
   Update_time: NULL
    Check_time: NULL
     Collation: utf8_general_ci
      Checksum: NULL
 Create_options:
```

```
        Comment:
  Block_format: Original
*************************** 24. row ******************
           Name: users_14
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 0
 Avg_row_length: 0
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 1
    Create_time: 2016-09-11 22:16:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
  Block_format: Original
24 rows in set (0.06 sec)
```

22.5.7.4 统计信息查询语句

本文介绍用于查询统计信息的语句。

- SHOW STATS语句用于查看整体的统计信息，主要包含QPS 、RT、慢SQL统计等信息。

```
mysql> show stats\G
*************************** 1. row ***************************
                  QPS: 0.00
              RDS_QPS: 0.00
             SLOW_QPS: 0.00
     PHYSICAL_SLOW_QPS: 0.00
      ERROR_PER_SECOND: 0.00
MERGE_QUERY_PER_SECOND: 0.00
    ACTIVE_CONNECTIONS: 1
               RT(MS): 0.00
           RDS_RT(MS): 0.00
         NET_IN(KB/S): 0.00
        NET_OUT(KB/S): 0.00
       THREAD_RUNNING: 1
1 row in set (0.02 sec)
```

统计指标具体说明如下表：

指标	说明
QPS	逻辑QPS
RDS_QPS	物理QPS
ERROR_PER_SECOND	每秒的错误数，包含语法错误，主键冲突等等所有异常
VIOLATION_PER_SECOND	每秒的主键或者唯一键冲突

指标	说明
MERGE_QUERY_PER_SECCOND	通过分库分表，从多表中进行的查询
ACTIVE_CONNECTIONS	当前连接上的连接
CONNECTION_CREATE_PER_SECCOND	每秒创建的连接数
RT(MS)	逻辑RT（响应时间）
RDS_RT(MS)	物理RT
NET_IN(KB/S)	DRDS收到的网络流量
NET_OUT(KB/S)	DRDS发送的网络流量
THREAD_RUNNING	正在运行的线程数
HINT_USED_PER_SECOND	每秒带HINT的查询的数量
HINT_USED_COUNT	启动到现在带HINT的查询总量
AGGREGATE_QUERY_PER_SECCOND	每秒走AGGREGATECURSOR的次数
AGGREGATE_QUERY_COUNT	启动到现在走AGGREGATECURSOR的次数
TEMP_TABLE_CREATE_PER_SECCOND	每秒创建的临时表的数量
TEMP_TABLE_CREATE_COUNT	启动到现在创建的临时表总数量
MULTI_DB_JOIN_PER_SECCOND	每秒跨库JOIN的数量
MULTI_DB_JOIN_COUNT	启动到现在跨库JOIN的总量

22.5.8 慢SQL明细

DRDS 将执行时间超过1秒的 SQL 定义为慢 SQL。DRDS 中的慢 SQL 分为两种：逻辑慢 SQL 和物理慢 SQL。

- 逻辑慢 SQL：应用发送到 DRDS 的 慢SQL；
- 物理慢 SQL：DRDS 发送到 MySQL 的 慢SQL。

语法

```
SHOW FULL {SLOW | PHYSICAL_SLOW} [WHERE where_condition]
                    [ORDER BY col_name [ASC | DESC], ...]
                    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

解释

SHOW FULL SLOW显示的是逻辑慢 SQL，即应用发送到 DRDS 的 SQL。

其中SHOW FULL SLOW的结果集会包含以下列，其含义如下：

- TRACE_ID：该 SQL 的唯一标记，同一个逻辑 SQL 以及该逻辑 SQL 产生的物理 SQL 的 TRACE_ID 相同，同时TRACE_ID 也会以注释的形式发送到 MySQL，在 MySQL 的 SQL 明细中可以根据 TRACE_ID 找到该 SQL；
- HOST：发送该 SQL 的客户端的 IP。

  注意: 在 VPC 模式下可能无法获取客户端 IP。

- START_TIME：DRDS 收到这个 SQL 的时间；
- EXECUTE_TIME：DRDS 执行该 SQL 消耗的时间；
- AFFECT_ROW：该 SQL 返回的记录数或者影响的行数；
- SQL：执行的语句。

SHOW FULL PHYSICAL_SLOW指的是物理慢 SQL，即 DRDS 发送到 MySQL 的 SQL。

SHOW FULL PHYSICAL_SLOW 的结果集会包含以下列，其含义如下：

- TRACE_ID：该 SQL 的唯一标记，同一个逻辑 SQL 以及该逻辑 SQL 产生的物理 SQL 的 TRACE_ID 相同，同时 TRACE_ID 也会以注释的形式发送到 MySQL，在 MySQL 的 SQL 明细中可以根据 TRACE_ID 找到该 SQL；
- GROUP_NAME：数据库分组名，分组的目标是管理多组数据完全相同的数据库，比如通过MySQL进行数据复制后的主备数据库，主要用来解决读写分离，主备切换的问题；
- DBKEY_NAME：执行的分库信息；
- START_TIME：DRDS 开始执行这个 SQL 的时间；
- EXECUTE_TIME：DRDS 执行该 SQL 消耗的时间；
- SQL_EXECUTE_TIME：DRDS 调用 MySQL 执行该 SQL 消耗的时间；
- GETLOCK_CONNECTION_TIME：DRDS 从连接池获取连接消耗的时间，该值如果很大，说明 MySQL 的连接已被耗尽，一般是慢 SQL 比较多引起，登录到相应的 MySQL，结合SHOW PROCESSLIST指令来排查。
- CREATE_CONNECTION_TIME：DRDS 建立 MySQL 连接消耗的时间，该值如果很大，很大原因是底层的 MySQL 压力比较大或者挂掉了；
- AFFECT_ROW：该 SQL 返回的记录数或者影响的行数；
- SQL：执行的语句。

示例一

下面的例子展示了如何一步步定位慢 SQL 在 DRDS 上、DRDS 跟 MySQL 之间的执行情况

1. 通过一些条件，例如执行时间，SQL 字符串匹配等方式来获取我们想要的慢 SQL；

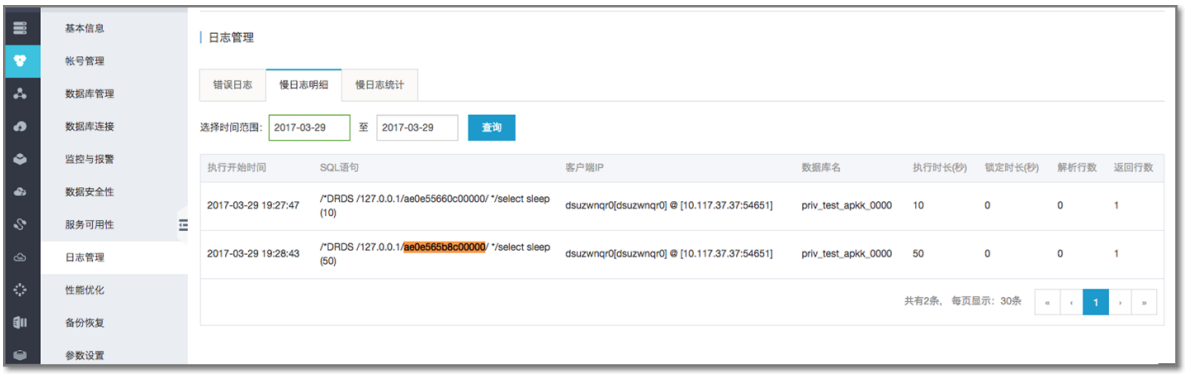
```
mysql> show full slow where `SQL` like '%select sleep(50)%';
+----------------+-----------+-------------------------+-------------+-----------+-----------------+
| TRACE_ID       | HOST      | START_TIME              | EXECUTE_TIME | AFFECT_ROW
 | SQL           |
+----------------+-----------+-------------------------+-------------+-----------+-----------------+
| ae0e565b8c00000 | 127.0.0.1 | 2017-03-29 19:28:43.028 |         50009 |           1 | select
 sleep(50) |
+----------------+-----------+-------------------------+-------------+-----------+-----------------+
1 row in set (0.02 sec)
```

2. 根据逻辑慢 SQL 中获取到的**TRACE_ID**，执行SHOW FULL PHYSICAL_SLOW指令获取这个 SQL 的物理执行情况；

```
mysql> show full physical_slow where trace_id = 'ae0e565b8c00000';
+----------------+-----------------------------------------------------+-----------------------------------------------
+-----------------------+-------------+-----------------+-----------------------+-----------------------
+-----------+-----------------+
| TRACE_ID       | GROUP_NAME                                                | DBKEY_NAME
                        | START_TIME              | EXECUTE_TIME | SQL_EXECUTE_TIME |
GETLOCK_CONNECTION_TIME | CREATE_CONNECTION_TIME | AFFECT_ROW | SQL
          |
+----------------+-----------------------------------------------------+-----------------------------------------------
+-----------------------+-------------+-----------------+-----------------------+-----------------------
+-----------+-----------------+
| ae0e565b8c00000 | PRIV_TEST_1489167306631PJAFPRIV_TEST_APKK_0000_RDS |
rdso6g5b6206sdq832ow_priv_test_apkk_0000_nfup | 2017-03-29 19:27:53.02 |       50001 |
         50001 |                    0 |                    0 |          1 | select sleep(50) |
+----------------+-----------------------------------------------------+-----------------------------------------------
+-----------------------+-------------+-----------------+-----------------------+-----------------------
+-----------+-----------------+
1 row in set (0.01 sec)
```

3. 可以在 MySQL 的 SQL 明细与慢 SQL 中，根据这个**TRACE_ID**查看这个 SQL 在 MySQL 上的执行情况。

图 593: 慢日志明细



示例二

在 MySQL 中找到一个慢 SQL，如何找到 DRDS 中的原始 SQL。

1. 通过MySQL 慢查询日志，观察到慢 SQL 的**TRACE_ID**为 ae0e55660c00000。
2. 根据第一步获取到的**TRACE_ID**，执行SHOW FULL PHYSICAL_SLOW指令获取这个 SQL 的物理执行情况；

```
mysql> show full physical_slow where trace_id = 'ae0e55660c00000';
+-----------------+-----------------------------------------------+-----------------------------------------
+-------------------------+-------------+-----------------+-----------------------+----------------------
+------------+------------------+
| TRACE_ID         | GROUP_NAME                                      | DBKEY_NAME
                          | START_TIME              | EXECUTE_TIME | SQL_EXECUTE_TIME |
GETLOCK_CONNECTION_TIME | CREATE_CONNECTION_TIME | AFFECT_ROW | SQL
          |
+-----------------+-----------------------------------------------+-----------------------------------------
+-------------------------+-------------+-----------------+-----------------------+----------------------
+------------+------------------+
| ae0e55660c00000 | PRIV_TEST_1489167306631PJAFPRIV_TEST_APKK_0000_RDS |
rdso6g5b6206sdq832ow_priv_test_apkk_0000_nfup | 2017-03-29 19:27:37.308 |        10003
 |           10001 |                     0 |                     0 |          1 | select sleep(10) |
+-----------------+-----------------------------------------------+-----------------------------------------
+-------------------------+-------------+-----------------+-----------------------+----------------------
+------------+------------------+
1 row in set (0.02 sec)
```

22.6 最佳实践

本章节介绍DRDS使用过程中的一些最佳实践。

22.6.1 如何选择拆分键

拆分键即分库/分表字段，是在水平拆分过程中用于生成拆分规则的数据表字段。DRDS 根据拆分键的值将数据表水平拆分到每个 RDS 实例上的物理分库中。

数据表拆分的首要原则，就是要尽可能找到数据表中的数据在业务逻辑上的主体，并确定大部分（或核心的）数据库操作都是围绕这个主体的数据进行，然后可使用该主体对应的字段作为拆分键，进行分库分表。

业务逻辑上的主体，通常与业务的应用场景相关，下面的一些典型应用场景都有明确的业务逻辑主体，可用于拆分键：

- 面向用户的互联网应用，都是围绕用户维度来做各种操作，那么业务逻辑主体就是用户，可使用用户对应的字段作为拆分键；
- 侧重于卖家的电商应用，都是围绕卖家维度来进行各种操作，那么业务逻辑主体就是卖家，可使用卖家对应的字段作为拆分键；
- 游戏类的应用，是围绕玩家维度来做各种操作，那么业务逻辑主体就是玩家，可使用玩家对应的字段作为拆分键；

- 车联网方面的应用，则是基于车辆信息进行操作，那么业务逻辑主体就是车辆，可使用车辆对应的字段作为拆分键；
- 税务类的应用，主要是基于纳税人的信息来开展前台业务，那么业务逻辑主体就是纳税人，可使用纳税人对应的字段作为拆分键。

以此类推，其它类型的应用场景，大多也能找到合适的业务逻辑主体作为拆分键的选择。

例如，某面向卖家的电商应用，需要对下面的一张单表进行水平拆分：

```
CREATE TABLE sample_order (
  id INT(11) NOT NULL,
  sellerId INT(11) NOT NULL,
  trade_id INT(11) NOT NULL,
  buyer_id INT(11) NOT NULL,
  buyer_nick VARCHAR(64) DEFAULT NULL,
  PRIMARY KEY (id)
)
```

由于确定了业务逻辑主体为卖家，那么选择对应的字段 sellerId 作为拆分键，且只分库不分表，则分布式 DDL 建表语句为：

```
CREATE TABLE sample_order (
  id INT(11) NOT NULL,
  sellerId INT(11) NOT NULL,
  trade_id INT(11) NOT NULL,
  buyer_id INT(11) NOT NULL,
  buyer_nick VARCHAR(64) DEFAULT NULL,
  PRIMARY KEY (id)
) DBPARTITION BY HASH(sellerId)
```

如果确实找不到合适的业务逻辑主体作为拆分键，那么可以考虑下面的方法来选择拆分键：

- 根据数据分布和访问的均衡度来考虑拆分键，尽量将数据表中的数据相对均匀地分布在不同的物理分库/分表中，适用于大量分析型查询的应用场景（查询并发度大部分能维持为1）；
- 按照数字（字符串）类型与时间类型字段相结合作为拆分键，进行分库和分表，适用于日志检索类的应用场景。

例如，某日志系统记录用户的所有操作，需要对下面的日志单表进行水平拆分：

```
CREATE TABLE user_log (
  userId INT(11) NOT NULL,
  name VARCHAR(64) NOT NULL,
  operation VARCHAR(128) DEFAULT NULL,
  actionDate DATE DEFAULT NULL
```

```
)
```

可以选择用户标识与时间字段相结合作为拆分键，并按照一周七天进行分表，则分布式 DDL 建表语句为：

```
CREATE TABLE user_log (
  userId INT(11) NOT NULL,
  name VARCHAR(64) NOT NULL,
  operation VARCHAR(128) DEFAULT NULL,
  actionDate DATE DEFAULT NULL
) DBPARTITION BY HASH(userId) TBPARTITION BY WEEK(actionDate) TBPARTITIONS 7
```

更多拆分键的选择和分表形式，请参考 使用*DDL*语句。

注意: 无论选择什么拆分键，采用何种拆分策略，都要注意拆分值是否存在热点的问题，尽量规避热点数据来选择拆分键。

22.6.2 如何选择分片数

DRDS 中的水平拆分有两个层次：分库和分表。每个 RDS 实例上默认会创建8个物理分库，每个物理分库上可以创建一个或多个物理分表。分表数通常也被称为分片数。

一般情况下，建议单个物理分表的容量不超过500万行数据。通常可以预估1到2年的数据增长量，用估算出的总数据量除以总的物理分库数，再除以建议的最大数据量500万，即可得出每个物理分库上需要创建的物理分表数：

```
物理分库上的物理分表数 = 向上取整(估算的总数据量 / (RDS 实例数 * 8) / 5,000,000)
```

因此，当计算出的物理分表数等于1时，分库即可，无需再进一步分表，即每个物理分库上一个物理分表；若计算结果大于1，则建议既分库又分表，即每个物理分库上多个物理分表。

例如，某用户预估一张表在2年后的总数据量大概是1亿行，有4个 RDS 实例，那么按照上述公式计算：

```
物理分库上的物理分表数 = CEILING(100,000,000 / ( 4 * 8 ) / 5,000,000) = CEILING(0.625) = 1
```

结果为1，那么只分库即可，即每个物理分库上1个物理分表。

若上述例子中仅有1个 RDS 实例，那么按照上述公式计算：

```
物理分库上的物理分表数 = CEILING(100,000,000 / ( 1 * 8 ) / 5,000,000) = CEILING(2.5) = 3
```

结果为3，那么建议既分库又分表，即每个物理分库上3个物理分表。

22.6.3 SQL优化基本概念

DRDS 是一个高效、稳定的分布式关系数据库服务，处理的是分布式关系运算。DRDS 对 SQL 的优化方法与单机关系数据库（例如 MySQL）有所不同，侧重考虑分布式环境中的网络 IO 开销，会尽量将 SQL 中的运算下推到底层各个分库（例如 RDS/MySQL）执行，从而减少网络 IO 开销、提升 SQL 执行效率。

DRDS 提供了一些指令来获取 SQL 的执行信息、辅助 SQL 的优化，例如获取 SQL 执行计划的 EXPLAIN 系列指令、获取 SQL 执行过程和开销的 TRACE 指令等。本文档介绍 DRDS 中 SQL 优化相关的基本概念和常用指令。

执行计划

为了访问数据而产生的一组有序的操作步骤集合，被称为 SQL 执行计划（简称执行计划）。在 DRDS 中，执行计划分为两个层次：DRDS 层的执行计划与 RDS/MySQL 层的执行计划。对执行计划的分析是进行 SQL 优化的有效方法，可以了解 DRDS 或 RDS/MySQL 是否对 SQL 语句生成了最优化的执行计划，是否有优化的空间等，从而为 SQL 优化提供重要的参考信息。

在 SQL 语句执行期间，DRDS 优化器会根据 SQL 语句和相关表的基本信息，判断该 SQL 语句应该在哪些分库上执行，决定在分库上执行的具体 SQL 语句形式，采用何种执行策略、数据合并与计算策略等。这个过程会尽可能达到优化 SQL 语句执行的目的，并产生 DRDS 层的执行计划。而 RDS/MySQL 层的执行计划就是原生的 MySQL 执行计划。

DRDS 提供了一组 EXPLAIN 指令来查看不同层面或不同详尽程度的执行计划。

下表是 DRDS 中 EXPLAIN 指令的简要说明，详细信息请参考*DRDS自定义控制指令* 。

表 83: EXPLAIN 指令说明

控制指令	说明	示例
EXPLAIN { SQL }	查看 DRDS 层 SQL 语句的概要执行计划，包括执行的分库、物理语句和整体参数。	EXPLAIN SELECT * FROM test
EXPLAIN DETAIL { SQL }	查看 DRDS 层 SQL 语句的详细执行计划，包括执行语句类型、并发度、返回字段信息、物理表和库分组等。	EXPLAIN DETAIL SELECT * FROM test

控制指令	说明	示例
EXPLAIN EXECUTE { SQL }	查看底层 RDS/MySQL 的执行计划，等同于 MySQL 的 EXPLAIN 语句。	EXPLAIN EXECUTE SELECT * FROM test

DRDS 层执行计划

DRDS 层执行计划的返回结果中，字段含义如下表所示：

表 84: DRDS层执行计划字段含义

字段	说明
GROUP_NAME	DRDS 分库的名字，可以根据后缀识别出是哪个分库，其值与 SHOW NODE 指令的结果一致。
SQL	在该分库上执行的 SQL 语句。
PARAMS	当 DRDS 使用 Prepare 协议与 MySQL 通信时，SQL 语句的参数列表。

其中 SQL 字段的内容有两种形式：

1. 如果 SQL 语句不包含以下部分，则以 SQL 语句的形式显示执行计划：
   - 涉及多个分库的聚合函数；
   - 涉及多个分片的分布式 Join；
   - 复杂子查询。

   例如：

```
mysql> EXPLAIN SELECT * FROM test;
+-----------------------------------------------+-----------------------------------------------+--------+
| GROUP_NAME                                    | SQL                                           | PARAMS |
+-----------------------------------------------+-----------------------------------------------+--------+
| TESTDB_1478746391548CDTCTESTDB_OXGJ_0000_RDS | select `test`.`c1`,`test`.`c2` from `test` | {}     |
| TESTDB_1478746391548CDTCTESTDB_OXGJ_0001_RDS | select `test`.`c1`,`test`.`c2` from `test` | {}     |
| TESTDB_1478746391548CDTCTESTDB_OXGJ_0002_RDS | select `test`.`c1`,`test`.`c2` from `test` | {}     |
| TESTDB_1478746391548CDTCTESTDB_OXGJ_0003_RDS | select `test`.`c1`,`test`.`c2` from `test` | {}     |
| TESTDB_1478746391548CDTCTESTDB_OXGJ_0004_RDS | select `test`.`c1`,`test`.`c2` from `test` | {}     |
| TESTDB_1478746391548CDTCTESTDB_OXGJ_0005_RDS | select `test`.`c1`,`test`.`c2` from `test` | {}     |
| TESTDB_1478746391548CDTCTESTDB_OXGJ_0006_RDS | select `test`.`c1`,`test`.`c2` from `test` | {}     |
```

```
| TESTDB_1478746391548CDTCTESTDB_OXGJ_0007_RDS | select `test`.`c1`,`test`.`c2`
 from `test` | {}     |
+-----------------------------------------+-----------------------------------------+--------+
8 rows in set (0.04 sec)
```

GROUP_NAME 字段中显示的 GROUP 名称可以在 SHOW NODE 的结果中找到：

```
mysql> SHOW NODE;
+----+-----------------------------------------+------------------+-----------------+--------------------
+-------------------+
| ID | NAME                                    | MASTER_READ_COUNT | SLAVE_READ_COUNT
 | MASTER_READ_PERCENT | SLAVE_READ_PERCENT |
+----+-----------------------------------------+------------------+-----------------+--------------------
+-------------------+
| 0  | TESTDB_1478746391548CDTCTESTDB_OXGJ_0000_RDS |               69 |
   0 | 100%                | 0%                  |
| 1  | TESTDB_1478746391548CDTCTESTDB_OXGJ_0001_RDS |               45 |
   0 | 100%                | 0%                  |
| 2  | TESTDB_1478746391548CDTCTESTDB_OXGJ_0002_RDS |               30 |
   0 | 100%                | 0%                  |
| 3  | TESTDB_1478746391548CDTCTESTDB_OXGJ_0003_RDS |               29 |
   0 | 100%                | 0%                  |
| 4  | TESTDB_1478746391548CDTCTESTDB_OXGJ_0004_RDS |               11 |
   0 | 100%                | 0%                  |
| 5  | TESTDB_1478746391548CDTCTESTDB_OXGJ_0005_RDS |                1 |
   0 | 100%                | 0%                  |
| 6  | TESTDB_1478746391548CDTCTESTDB_OXGJ_0006_RDS |                8 |
   0 | 100%                | 0%                  |
| 7  | TESTDB_1478746391548CDTCTESTDB_OXGJ_0007_RDS |               50 |
   0 | 100%                | 0%                  |
+----+-----------------------------------------+------------------+-----------------+--------------------
+-------------------+
8 rows in set (0.10 sec)
```

2. 无法使用 SQL 语句表示的执行计划，DRDS 使用自定义格式的执行计划来表示。

   例如：

```
mysql> EXPLAIN DETAIL SELECT COUNT(*) FROM test;
+-----------------------------------------+-------------------------------+--------+
| GROUP_NAME                              | SQL                           | PARAMS |
+-----------------------------------------+-------------------------------+--------+
| TEST_DB_1478746391548CDTCTEST_DB_OXGJ_0000_RDS | Merge as test
    queryConcurrency:GROUP_CONCURRENT
    columns:[count(*)]
    executeOn: TEST_DB_1478746391548CDTCTEST_DB_OXGJ_0000_RDS
        Query from test as test
            queryConcurrency:SEQUENTIAL
            columns:[count(*)]
            tableName:test
            executeOn: TEST_DB_1478746391548CDTCTEST_DB_OXGJ_0000_RDS
        Query from test as test
            queryConcurrency:SEQUENTIAL
            columns:[count(*)]
            tableName:test
            executeOn: TEST_DB_1478746391548CDTCTEST_DB_OXGJ_0001_RDS
        ... ...
        Query from test as test
            queryConcurrency:SEQUENTIAL
            columns:[count(*)]
```

```
        tableName:test
        executeOn: TEST_DB_1478746391548CDTCTEST_DB_OXGJ_0007_RDS
| NULL  |
+-----------------------------------------------+-------------------------------+--------+
1 row in set (0.00 sec)
```

其中，SQL 字段内容中的 executeOn 表示下推的 SQL 语句在哪个分库上执行，分库执行后返回的结果最终由 DRDS 进行合并。

RDS/MySQL 层执行计划

RDS/MySQL 层执行计划的结果与原生 MySQL 执行计划一致，请参考 *MySQL* 官方文档。

一个 DRDS 逻辑表可能由多个分布在不同分库上的分片所组成，所以查看 RDS/MySQL 层执行计划也有多种方法。

1. 查看一个 RDS/MySQL 分片的执行计划。

   如果查询条件中带有拆分键，则直接使用 EXPLAIN EXECUTE 指令来查看对应分片上的执行计划。例如：

```
mysql> EXPLAIN EXECUTE SELECT * FROM test WHERE c1 = 1;
+----+-------------+-------+-------+---------------+---------+---------+-------+------+-------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref   | rows | Extra |
+----+-------------+-------+-------+---------------+---------+---------+-------+------+-------+
|  1 | SIMPLE      | test  | const | PRIMARY       | PRIMARY | 4       | const |    1 | NULL  |
+----+-------------+-------+-------+---------------+---------+---------+-------+------+-------+
1 row in set (0.04 sec)
```

   注意: 如果 SQL 语句出现了跨分片的情况（例如 SQL 语句的条件中没有带拆分键），则 EXPLAIN EXECUTE 会随机返回一个 RDS/MySQL 分片上的执行计划。

   如果要查看一条 SQL 语句在指定分片上的执行计划，可以使用 Hint 的方式来实现。例如：

```
mysql>          /*TDDL:node='TESTDB_1478746391548CDTCTESTDB_OXGJ_0000_RDS'*/
EXPLAIN SELECT * FROM test;
+----+-------------+-------+------+---------------+-------+---------+-------+------+-------+
| id | select_type | table | type | possible_keys |  key  | key_len |  ref  | rows | Extra |
+----+-------------+-------+------+---------------+-------+---------+-------+------+-------+
|  1 | SIMPLE      | test  | ALL  | NULL          | NULL  | NULL    | NULL  |    2 | NULL  |
+----+-------------+-------+------+---------------+-------+---------+-------+------+-------+
1 row in set (0.04 sec)
```

2. 查看所有 RDS/MySQL 分片的执行计划。

   如果确实需要查看 SQL 语句在所有分片上的执行计划，可以利用 SCAN Hint 来实现：

```
mysql> /*TDDL:scan='test'*/EXPLAIN SELECT * FROM test;
+----+-------------+-------+------+---------------+-------+---------+-------+------+-------+
| id | select_type | table | type | possible_keys |  key  | key_len |  ref  | rows | Extra |
+----+-------------+-------+------+---------------+-------+---------+-------+------+-------+
```

```
| 1 | SIMPLE      | test  | ALL  | NULL          | NULL  | NULL    | NULL  |    2 | NULL  |
| 1 | SIMPLE      | test  | ALL  | NULL          | NULL  | NULL    | NULL  |    3 | NULL  |
+----+-------------+-------+------+---------------+-------+---------+-------+------+-------+
2 rows in set (0.08 sec)
```

注意:

1. 使用 Hint 方式时，除了分库分表情况下的表名替换，DRDS 不会对 SQL 语句做其它处理，会直接将逻辑 SQL 语句发送到 RDS/MySQL 上执行，结果也不会做任何处理。
2. 通过 EXPLAIN 获取的执行计划是静态分析产生的，并没有真正在数据库中执行。

TRACE指令

DRDS 中的 TRACE 指令可以跟踪 SQL 的执行过程和各个阶段的执行开销，与执行计划相结合，更有助于对 SQL 进行优化。

TRACE 指令包含两条相关的指令：TRACE 和 SHOW TRACE，需要在一起配合使用。详细的用法信息请参考 *DRDS自定义控制指令*。

22.6.4 SQL优化方法

本文介绍 DRDS 中 SQL 优化的原则和不同类型 SQL 的优化方法。

SQL 优化的基本原则

在 DRDS 中，可由 RDS/MySQL 执行的 SQL 计算称为可下推计算。可下推计算能够减少数据传输，减少网络层和 DRDS 层的开销，提升 SQL 语句的执行效率。

因此，DRDS SQL 语句优化的基本原则为：**尽量让更多的计算可下推到 RDS/MySQL 上执行。**

可下推计算主要包括：

- JOIN 连接；
- 过滤条件，如 WHERE 或 HAVING 中的条件；
- 聚合计算，如 COUNT，GROUP BY 等；
- 排序，如 ORDER BY；
- 去重，如 DISTINCT；
- 函数计算，如 NOW() 函数等；
- 子查询。

注意: 上述列表只是列出可下推计算的各种可能形式，并不代表所有的子句/条件或者子句/条件的组合一定是可下推计算。

不同类型和条件的 SQL 优化有不同的侧重点和方法，下面将针对以下几种情况介绍 SQL 优化的具体方法：

- 单表 SQL 优化
  - 过滤条件优化
  - 查询返回行数优化
  - 分组及排序优化
- JOIN 优化
  - 可下推的 JOIN 优化
  - 分布式 JOIN 优化
- 子查询优化

22.6.4.1 单表SQL优化

单表 SQL 优化有以下几个原则：

- SQL 语句尽可能带有拆分键；
- 拆分键的条件尽可能是等值条件；
- 如果拆分键的条件是 IN 条件，则 IN 后面的值的数目应尽可能少（需要远少于分片数，并且数目不会随业务的增长而增多）；
- 如果 SQL 语句不带有拆分键，那么 DISTINCT、GROUP BY 和 ORDER BY 在同一个 SQL 语句中尽量只出现一种。

过滤条件优化

DRDS 的数据是按拆分键水平切分的，过滤条件中应尽量包含带有拆分键的条件，可以让 DRDS 根据拆分键对应的值将查询直接下推到特定的分库，避免 DRDS 做全表扫描。

例如，表 test 的拆分键是 c1，过滤条件中若不带有拆分键，会做全表扫描：

```
mysql> SELECT * FROM test WHERE c2 = 2;
+----+----+
| c1 | c2 |
+----+----+
|  2 |  2 |
+----+----+
```

```
1 row in set (0.05 sec)
```

对应的执行计划为：

```
mysql> EXPLAIN SELECT * FROM test WHERE c2 = 2;
+-----------------------------------------------+-----------------------------------------------------------------------------
+--------+
| GROUP_NAME                                    | SQL                                                                         | PARAMS |
+-----------------------------------------------+-----------------------------------------------------------------------------
+--------+
| SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0004_RDS | select `test`.`c1`,`test`.`c2`
 from `test` where (`test`.`c2` = 2) | {}     |
| SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0007_RDS | select `test`.`c1`,`test`.`c2`
 from `test` where (`test`.`c2` = 2) | {}     |
| SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0005_RDS | select `test`.`c1`,`test`.`c2`
 from `test` where (`test`.`c2` = 2) | {}     |
| SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0002_RDS | select `test`.`c1`,`test`.`c2`
 from `test` where (`test`.`c2` = 2) | {}     |
| SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0003_RDS | select `test`.`c1`,`test`.`c2`
 from `test` where (`test`.`c2` = 2) | {}     |
| SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0006_RDS | select `test`.`c1`,`test`.`c2`
 from `test` where (`test`.`c2` = 2) | {}     |
| SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0000_RDS | select `test`.`c1`,`test`.`c2`
 from `test` where (`test`.`c2` = 2) | {}     |
| SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0001_RDS | select `test`.`c1`,`test`.`c2`
 from `test` where (`test`.`c2` = 2) | {}     |
+-----------------------------------------------+-----------------------------------------------------------------------------
+--------+
8 rows in set (0.00 sec)
```

含拆分键的过滤条件的取值范围越小，越有助于提高 DRDS 的查询速度。

例如，对表 test 查询时包含带有拆分键 c1 的范围过滤条件：

```
mysql> SELECT * FROM test WHERE c1 > 1 AND c1 < 4;
+----+----+
| c1 | c2 |
+----+----+
|  2 |  2 |
|  3 |  3 |
+----+----+
2 rows in set (0.04 sec)
```

对应的执行计划为：

```
mysql> EXPLAIN SELECT * FROM test WHERE c1 > 1 AND c1 < 4;
+-----------------------------------------------
+-----------------------------------------------------------------------------------+--------+
| GROUP_NAME                                    | SQL
                                                                                    | PARAMS |
+-----------------------------------------------
+-----------------------------------------------------------------------------------+--------+
| SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0002_RDS | select `test`.`c1`,`test`.`c2`
 from `test` where ((`test`.`c1` > 1) AND (`test`.`c1` < 4)) | {}     |
| SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0003_RDS | select `test`.`c1`,`test`.`c2`
 from `test` where ((`test`.`c1` > 1) AND (`test`.`c1` < 4)) | {}     |
+-----------------------------------------------
+-----------------------------------------------------------------------------------+--------+
```

```
2 rows in set (0.00 sec)
```

等值条件会比范围条件执行得更快。例如：

```
mysql> SELECT * FROM test WHERE c1 = 2;
+----+----+
| c1 | c2 |
+----+----+
|  2 |  2 |
+----+----+
1 row in set (0.03 sec)
```

对应的执行计划为：

```
mysql> EXPLAIN SELECT * FROM test WHERE c1 = 2;
+-----------------------------------------------+-------------------------------------------------------------------------
+--------+
| GROUP_NAME                                    | SQL                                                                     | PARAMS |
+-----------------------------------------------+-------------------------------------------------------------------------
+--------+
| SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0002_RDS | select `test`.`c1`,`test`.`c2`
 from `test` where (`test`.`c1` = 2) | {}     |
+-----------------------------------------------+-------------------------------------------------------------------------
+--------+
1 row in set (0.00 sec)
```

此外，在向拆分表中插入数据时，插入字段中必须带有拆分键。

例如，向表 test 中插入数据时带有拆分键 c1：

```
mysql> INSERT INTO test(c1,c2) VALUES(8,8);
Query OK, 1 row affected (0.07 sec)
```

查询返回行数优化

DRDS 在执行带有 **LIMIT** [*offset*,] *row_count* 的查询时，实际上是依次将 *offset* 之前的记录读取出来并直接丢弃，这样当 *offset* 非常大的时候，即使 *row_count* 很小，也会导致查询非常缓慢。例如以下的 SQL：

```
SELECT *
FROM sample_order
ORDER BY sample_order.id
LIMIT 10000, 2
```

它虽然只返回第10000与10001两条记录，可它的执行时间为12秒左右，这是因为 DRDS 实际读取的记录数为10002条：

```
mysql> SELECT * FROM sample_order ORDER BY sample_order.id LIMIT 10000,2;
+--------------+------------+--------------+--------------+------------+
| id           | sellerId   | trade_id     | buyer_id     | buyer_nick |
+--------------+------------+--------------+--------------+------------+
| 242012755468 | 1711939506 | 242012755467 | 244148116334 | zhangsan   |
| 242012759093 | 1711939506 | 242012759092 | 244148138304 | wangwu     |
```

```
+--------------+------------+--------------+--------------+------------+
2 rows in set (11.93 sec)
```

对应的执行计划为：

```
mysql> EXPLAIN SELECT * FROM sample_order ORDER BY sample_order.id LIMIT 10000,2;
+-----------------+-------------+------------+
| GROUP_NAME      | SQL         | PARAMS     |
+-----------------+-------------+------------+
|          SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0004_RDS          |          select

 from `sample_order` order by `sample_order`.`id` asc  limit 0,10002 | {}     |
|          SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0007_RDS          |          select

 from `sample_order` order by `sample_order`.`id` asc  limit 0,10002 | {}     |
|          SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0005_RDS          |          select

 from `sample_order` order by `sample_order`.`id` asc  limit 0,10002 | {}     |
|          SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0002_RDS          |          select

 from `sample_order` order by `sample_order`.`id` asc  limit 0,10002 | {}     |
|          SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0003_RDS          |          select

 from `sample_order` order by `sample_order`.`id` asc  limit 0,10002 | {}     |
|          SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0006_RDS          |          select

 from `sample_order` order by `sample_order`.`id` asc  limit 0,10002 | {}     |
|          SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0000_RDS          |          select

 from `sample_order` order by `sample_order`.`id` asc  limit 0,10002 | {}     |
|          SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0001_RDS          |          select

 from `sample_order` order by `sample_order`.`id` asc  limit 0,10002 | {}     |
+------------------+--------------------------+--------+
8 rows in set (0.01 sec)
```

针对上述情况，SQL 优化方向是先查出 id 集合，再通过 IN 匹配真正的记录内容，改写后的 SQL 查询如下：

```
SELECT *
FROM sample_order o
WHERE o.id IN (
      SELECT id
      FROM sample_order
      ORDER BY id
      LIMIT 10000, 2 )
```

这样改写的目的是先用内存缓存 id（前提是 id 数目不多），如果 sample_order 表的拆分键是 id，那么 DRDS 还可以将这样的 IN 查询通过规则计算下推到不同的分库来查询，避免全表扫描和不必要的网络 IO。观察改写后的 SQL 查询效果：

```
mysql> SELECT *
    -> FROM sample_order o
    -> WHERE o.id IN ( SELECT id FROM sample_order ORDER BY id LIMIT 10000,2 );
+--------------+------------+--------------+--------------+------------+
| id           | sellerId   | trade_id     | buyer_id     | buyer_nick |
```

```
+--------------+------------+--------------+--------------+------------+
| 242012755468 | 1711939506 | 242012755467 | 244148116334 | zhangsan   |
| 242012759093 | 1711939506 | 242012759092 | 244148138304 | wangwu     |
+--------------+------------+--------------+--------------+------------+
2 rows in set (1.08 sec)
```

执行时间由原来的12秒减少到1.08秒，缩减了一个数量级。

对应的执行计划为：

```
mysql> EXPLAIN SELECT *
    -> FROM sample_order o
    -> WHERE o.id IN ( SELECT id FROM sample_order ORDER BY id LIMIT 10000,2 );
+-----------------------------------------------
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
+--------+
| GROUP_NAME                                    | SQL
                                                | PARAMS |
+-----------------------------------------------
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
+--------+
|          SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0002_RDS          |          select
 `o`.`id`,`o`.`sellerId`,`o`.`trade_id`,`o`.`buyer_id`,`o`.`buyer_nick` from `sample_order` `o` where
 (`o`.`id` IN (10002)) | {}     |
|          SEQPERF_1478746391548CDTCSEQPERF_OXGJ_0001_RDS          |          select
 `o`.`id`,`o`.`sellerId`,`o`.`trade_id`,`o`.`buyer_id`,`o`.`buyer_nick` from `sample_order` `o` where
 (`o`.`id` IN (10001)) | {}     |
+-----------------------------------------------
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
+--------+
2 rows in set (0.03 sec)
```

分组及排序优化

在 DRDS 中，如果在一条 SQL 查询中必须同时使用 DISTINCT、GROUP BY 与 ORDER BY，应尽可能保证 DISTINCT、GROUP BY 与 ORDER BY 语句后所带的字段相同，且尽量为拆分键，使最终的 SQL 查询只返回少量数据。这样能够让分布式查询中消耗的网络带宽最小，并且不需要取出大量数据在临时表内进行排序，系统的性能能够达到最优状态。

22.6.4.2 JOIN优化

DRDS 的 JOIN 查询分为可下推的 JOIN 和不可下推的 JOIN（即分布式 JOIN）两类，其优化策略各不相同。

可下推的 JOIN 优化

可下推的 JOIN 主要分为以下几类：

- 单表（即非拆分表）之间的 JOIN；
- 参与 JOIN 的表在过滤条件中均带有拆分键作为条件，并且拆分算法相同（即通过拆分算法计算的数据分布在相同分片上）；

- 参与 JOIN 的表均按照拆分键作为 JOIN 条件，并且拆分算法相同；
- 广播表（也称为小表广播）与拆分表之间的 JOIN。

使用 DRDS 时，应尽可能将 JOIN 查询优化成能够在分库上执行的可下推的 JOIN 形式。

以广播表与拆分表之间的 JOIN 为例，应将广播表作为 JOIN 驱动表（将 JOIN 中的左表称为驱动表）。DRDS 的广播表在各个分库都会存放一份同样的数据，当作为 JOIN 驱动表时，该表与分表的 JOIN 可以转化为单库的 JOIN 并进行合并计算，提高查询性能。

例如，有以下的三个表做 JOIN 查询（其中表 sample_area 是广播表，sample_item 和 sample_buyer 是拆分表），查询执行时间约15秒：

```
mysql> SELECT sample_area.name
    -> FROM sample_item i JOIN sample_buyer b ON i.sellerId = b.sellerId JOIN sample_area
 a ON b.province = a.id
    -> WHERE a.id < 110107
    -> LIMIT 0, 10;
+------+
| name |
+------+
|  BJ  |
|  BJ  |
|  BJ  |
|  BJ  |
|  BJ  |
|  BJ  |
|  BJ  |
|  BJ  |
|  BJ  |
|  BJ  |
+------+
10 rows in set (14.88 sec)
```

如果调整一下 JOIN 的顺序，将广播表放在最左边作为 JOIN 驱动表，则整个 JOIN 查询在 DRDS 中会被下推为单库 JOIN 查询：

```
mysql> SELECT sample_area.name
    -> FROM sample_area a JOIN sample_buyer b ON b.province = a.id JOIN sample_item i ON
 i.sellerId = b.sellerId
    -> WHERE a.id < 110107
    -> LIMIT 0, 10;
+------+
| name |
+------+
|  BJ  |
|  BJ  |
|  BJ  |
|  BJ  |
|  BJ  |
|  BJ  |
|  BJ  |
|  BJ  |
|  BJ  |
|  BJ  |
```

```
+------+
10 rows in set (0.04 sec)
```

查询执行时间从15秒减少到0.04秒，性能提升非常明显。



注意: 广播表在分库上通过同步机制实现数据一致，有秒级延迟。

分布式 JOIN 优化

如果一个 JOIN 查询不可下推（即 JOIN 条件和过滤条件中均不带有拆分键），则需要由 DRDS 完成查询中的部分计算，即分布式 JOIN。

通常将分布式 JOIN 中的表按照数据量大小分为两类：

- 小表：经过条件过滤后，参与 JOIN 计算的中间结果的数据量比较少（一般少于 100 条，或者相较于其它表数据更少）的表；
- 大表：经过条件过滤后，参与 JOIN 计算的中间结果的数据量比较大（一般多于 100 条，或者相较于其它表数据更多）的表。

在 DRDS 层的 JOIN 计算中，大多数情况下采用的 JOIN 算法都是 Nested Loop 及其派生算法（若 JOIN 有排序要求，则使用 Sort Merge 算法）。采用 Nested Loop 算法时，如果 JOIN 中左表的数据量越少，那么 DRDS 对右表做查询的次数就越少，如果右表上建有索引或者表中的数据量也很少，则 JOIN 的速度会更快。因此，在 DRDS 中，分布式 JOIN 的左表被称为驱动表，对分布式 JOIN 的优化应将小表作为驱动表，且让驱动表带有尽可能多的过滤条件。

以下面的分布式 JOIN 为例，查询约需要24秒：

```
mysql> SELECT t.title, t.price
    -> FROM sample_order o,
    ->      ( SELECT * FROM sample_item i WHERE i.id = 242002396687 ) t
    -> WHERE t.source_id = o.source_item_id AND o.sellerId < 1733635660;
+---------------------------------+--------+
| title                           | price  |
+---------------------------------+--------+
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
+---------------------------------+--------+
```

```
10 rows in set (23.79 sec)
```

通过初步分析，上述 JOIN 查询是一个 INNER JOIN，并不知道参与 JOIN 计算的中间结果的实际数据量，可以对 o 表与 t 表分别做 COUNT() 查询得到实际数据。

对于 o 表，观察到 WHERE 条件中的 o.sellerId < 1733635660 只与 o 表相关，可以将其提取出来，附加到 o 表的 COUNT() 查询条件中，得到如下的查询结果：

```
mysql> SELECT COUNT(*) FROM sample_order o WHERE o.sellerId < 1733635660;
+----------+
| count(*) |
+----------+
|   504018 |
+----------+
1 row in set (0.10 sec)
```

o 表的中间结果约有50万条记录。类似地，t 表是一个子查询，直接将其抽取出来进行 COUNT() 查询：

```
mysql> SELECT COUNT(*) FROM sample_item i WHERE i.id = 242002396687;
+----------+
| count(*) |
+----------+
|        1 |
+----------+
1 row in set (0.01 sec)
```

t 表的中间结果只有1条记录，所以可确定 o 表为大表，t 表为小表。根据尽量将小表作为分布式 JOIN 驱动表的原则，将 JOIN 查询调整后的查询结果为：

```
mysql> SELECT t.title, t.price
    -> FROM ( SELECT * FROM sample_item i WHERE i.id = 242002396687 ) t,
    ->      sample_order o
    -> WHERE  t.source_id = o.source_item_id AND o.sellerId < 1733635660;
+----------------------------------+--------+
| title                            | price  |
+----------------------------------+--------+
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
| Sample Item for Distributed JOIN | 239.00 |
+----------------------------------+--------+
10 rows in set (0.15 sec)
```

查询时间从约24秒减少到0.15秒，性能提升非常明显。

22.6.4.3 子查询优化

在包含子查询的 SQL 优化中，应尽可能将查询下推到具体的分库上执行，并减少 DRDS 层的计算量。

要达到这一目标，可以尝试两个方面的优化：

- 将子查询的形式改写为多表 JOIN 形式，并参照 JOIN 优化方法进一步优化；
- 尽量在 JOIN 条件或过滤条件中带上拆分键，有利于 DRDS 将查询下推到特定的分库，避免全表扫描。

以下面的子查询为例：

```
SELECT o.*
FROM sample_order o
WHERE NOT EXISTS
    (SELECT sellerId FROM sample_seller s WHERE o.sellerId = s.id)
```

可将其改写为 JOIN 的形式：

```
SELECT o.*
FROM sample_order o LEFT JOIN sample_seller s ON o.sellerId = s.id
WHERE s.id IS NULL
```

22.6.5 如何选择应用连接池

数据库连接池是对数据库连接进行统一管理的技术，主要目的是提高应用性能，减轻数据库负载。

- **资源复用**：连接可以重复利用，避免了频繁创建、释放连接引起的大量性能开销。在减少系统消耗的基础上，同时增进了系统的平稳性。
- **提高系统响应效率**：连接的初始化工作完成后，所有请求可以直接利用现有连接，避免了连接初始化和释放的开销，提高了系统的响应效率。
- **避免连接泄漏**：连接池可根据预设的回收策略，强制回收连接，从而避免了连接资源泄漏。

将应用和数据库连接进行业务操作，建议使用连接池。如果是 Java 程序，推荐使用 *Druid* 连接池。

Druid 的 Spring 标准配置：

```
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init"
destroy-method="close">
      <property name="driverClassName" value="com.mysql.jdbc.Driver" />
      <!-- 基本属性 URL、user、password -->
            <property name="url" value="jdbc:mysql://ip:port/db?autoReconnect=true&amp;
rewriteBatchedStatements=true&amp;socketTimeout=30000&amp;connectTimeout=3000" />
      <property name="username" value="root" />
      <property name="password" value="123456" />
      <!-- 配置初始化大小、最小、最大 -->
      <property name="maxActive" value="20" />
      <property name="initialSize" value="3" />
```

```
        <property name="minIdle" value="3" />
        <!-- maxWait 获取连接等待超时的时间 -->
        <property name="maxWait" value="60000" />
      <!-- timeBetweenEvictionRunsMillis 间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->
        <property name="timeBetweenEvictionRunsMillis" value="60000" />
        <!-- minEvictableIdleTimeMillis 一个连接在池中最小空闲的时间，单位是毫秒-->
        <property name="minEvictableIdleTimeMillis" value="300000" />
        <!-- 检测连接是否可用的 SQL -->
        <property name="validationQuery" value="SELECT 'z'" />
        <!-- 是否开启空闲连接检查 -->
        <property name="testWhileIdle" value="true" />
        <!-- 是否在获取连接前检查连接状态 -->
        <property name="testOnBorrow" value="false" />
        <!-- 是否在归还连接时检查连接状态 -->
        <property name="testOnReturn" value="false" />
    </bean>
```

22.6.6 如何排查 DRDS 慢 SQL

在 DRDS 中，一条 SQL 语句会在 DRDS 和 MySQL/RDS 节点上逐步执行。任意节点上的执行损耗过大都会导致慢 SQL。

慢 SQL 的一般排查步骤为：

1.定位慢 SQL；

2.定位性能损耗节点；

3.定位性能损耗原因并处理。

> **说明:** 排查过程中，建议通过 MySQL 命令行进行连接：`mysql -hIP -PPORT -uUSER -pPASSWORD -c` 。请务必加上 **-c**，防止 MySQL 客户端过滤掉注释（默认）从而影响 HINT 的执行。

22.6.6.1 定位慢SQL

定位慢 SQL 一般有两种场景：历史信息可从慢 SQL 记录中查询；实时慢 SQL 执行信息可使用SHOW PROCESSLIST 指令展示。

- 查看慢 SQL 记录

  执行以下指令查询慢 SQL Top 10。此查询针对 DRDS 层面的逻辑 SQL 。一个逻辑 SQL 对应一个或者多个 MySQL/RDS 库表的 SQL 执行。详情见慢*SQL*明细。

```
mysql> SHOW SLOW limit 10;
+-----------------+--------------+------------------------+-------------+------------
+-------------------------------------------------------------+
| TRACE_ID        | HOST         | START_TIME             | EXECUTE_TIME | AFFECT_ROW
 | SQL                                                         |
```

```
+----------------+-------------+-----------------------+-------------+------------
+-------------------------------------------------------------+
| ac3133132801001 | 42.120.74.97 | 2017-03-06 15:48:32.330 |       900392 |         -1 | select
 detail_url, sum(price) from t_item group by detail_url;  |
......
+----------------+-------------+-----------------------+-------------+------------
+-------------------------------------------------------------+
10 rows in set (0.01 sec)
```

- 查看当前实时 SQL 执行信息

  如果当前服务器中正在执行的 SQL 比较慢，可以使用 **SHOW PROCESSLIST** 指令来查看当前 DRDS 数据库中实时的执行信息。其中 TIME 列代表的是该 SQL 已经执行的时间。

```
mysql> SHOW PROCESSLIST WHERE COMMAND != 'Sleep';
+--------------+-----------+-------------------+-------------+---------
+-------------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------------+-----------
+--------------+-----------+
| ID             | USER      | DB                | COMMAND     | TIME    | STATE
                                  | INFO
          | ROWS_SENT | ROWS_EXAMINED | ROWS_READ |
+--------------+-----------+-------------------+-------------+---------
+-------------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------------+-----------
+--------------+-----------+
| 0-0-352724126 | ifisibhk0 | test_123_wvvp_0000 | Query       |      13  | Sending data
                                                  | /*DRDS /42.120.74.88/ac47e5a72801000/ */select
 `t_item`.`detail_url`,SUM(`t_item`.`price`) from `t_i |      NULL |         NULL |      NULL |
| 0-0-352864311 | cowxhthg0 | NULL              | Binlog Dump |     17  | Master has sent all binlog
 to slave; waiting for binlog to be updated | NULL
                                  |      NULL |         NULL |      NULL |
| 0-0-402714795 | ifisibhk0 | test_123_wvvp_0005 | Alter       |     114 | Sending data
                                              | /*DRDS /42.120.74.88/ac47e5a72801000/ */ALTER TABLE
 `Persons` ADD `Birthday` date                  |      NULL |         NULL |      NULL |
......
+--------------+-----------+-------------------+-------------+---------
+-------------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------------+-----------
+--------------+-----------+
12 rows in set (0.03 sec)
```

  各列的信息如下：

  - ID：连接标识。
  - USER：执行该 SQL 的分库用户名。
  - DB：指定的数据库，如果没有指定则为 NULL。
  - COMMAND：正在执行的命令类型。SLEEP 代表空闲连接。其它命令详情见 *MySQL* 线程信息文档。
  - TIME：SQL 已执行的时间，单位是秒。
  - STATE：当前的执行状态，请参见 *MySQL* 线程状态文档。

- INFO：正在执行的 SQL 语句，有可能因为过长而无法完全显示，此时可以结合业务参数等信息把完整 SQL 推导出来。

在当前的示例中定位到以下的慢 SQL：

```
ALTER TABLE `Persons` ADD `Birthday` date
```

22.6.6.2 定位性能损耗节点

从慢 SQL 记录或者实时 SQL 执行信息中定位到慢 SQL 后，可以执行 TRACE 指令跟踪该 SQL 在 DRDS 和 MySQL/RDS 上的运行时间，以便定位瓶颈。

TRACE 命令会**实际执行** SQL，在执行过程中记录所有节点消耗的时间，并返回执行结果。TRACE 及其他控制指令详情见*DRDS自定义控制指令*。



说明： DRDS TRACE 命令需要保持连接的上下文信息，某些 GUI 客户端可能会使用连接池，导致命令不正常。因此建使用 MySQL 命令行执行。

针对上文定位的慢 SQL，可以执行以下指令：

```
mysql> trace select detail_url, sum(distinct price) from t_item group by detail_url;
   +---------------+---------------+
   | detail_url    | sum(price)    |
   +---------------+---------------+
   | www.xxx.com   | 1084326800.00 |
   | www.xx1.com   | 1084326800.00 |
   | www.xx2.com   | 1084326800.00 |
   | www.xx3.com   | 1084326800.00 |
   | www.xx4.com   | 1084326800.00 |
   | www.xx5.com   | 1084326800.00 |
   ......
   +---------------+---------------+
   1 row in set (7 min 2.72 sec)
```

TRACE 指令执行完毕后，可以执行 SHOW TRACE 命令查看结果，根据每个组件的时间消耗来判断慢 SQL 的瓶颈。

```
mysql> SHOW TRACE;
                              +------+------------+-------------+-------------------------------------------------------------
+-------------------------------------------+--------------+------------------------+------
+-------------------------------------------------------------------------------------------------------------------------------
+--------+
   | ID   | TIMESTAMP  | TYPE        | GROUP_NAME                                  | DBKEY_NAME
                        | TIME_COST(MS) | CONNECTION_TIME_COST(MS) | ROWS | STATEMENT
                                                                                                                    | PARAMS |
                              +------+------------+-------------+-------------------------------------------------------------
+-------------------------------------------+--------------+------------------------+------
+-------------------------------------------------------------------------------------------------------------------------------
+--------+
   |    0 |      0.000 | Optimize     | DRDS                                        | DRDS
                    | 2             | 0.00                     |    0 | select detail_url, sum(price) from t_item group
 by detail_url                                                                                                    | NULL   |
```

```
|    1 | 423507.342 | Merge Sorted | DRDS                                                
```

22.6.6.3 处理性能损耗节点

慢节点可能存在于DRDS或者底层的MySQL/RDS，判断明确后需要分别进行处理。

DRDS 慢节点处理

当慢 GROUP_NAME 是 DRDS 时，请检查执行过程中是否存在 Merge Sorted、Temp Table Merge、Aggregate 等计算耗时操作。如果存在的话请参考 *SQL优化方法*文档进行优化。

MySQL/RDS 慢节点处理

当慢节点在 MySQL/RDS 时，请检查该 SQL 语句在 MySQL/RDS 上的**执行计划**。

在 DRDS 中，可以使用 /*TDDL:node={GROUP_NAME}*/ EXPLAIN 来查看某个 MySQL/RDS 的执行计划。执行计划展示了执行该 SQL 的过程信息，包括表间关联及索引信息等。

详细过程如下：

1. 依据 GROUP_NAME 组装 HINT：/*TDDL:node=’TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0005_RDS’ */
2. 将组装好的 HINT 及带 EXPLAIN 前缀的 STATEMENT 拼装成新的 SQL 并执行。EXPLAIN 指令不会真正执行，而只是显示该 SQL 的执行计划信息。

以上文定位的慢节点为例查询执行计划：

```
mysql> /*TDDL:node='TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0005_RDS'*/ EXPLAIN select `t_item`.`detail_url`,SUM(distinct `t_item`.`price`) from `t_item` group by `t_item`.`detail_url` order by `t_item`.`detail_url` asc;
+----+-------------+--------+------+---------------+-----+---------+-----+---------+----------------------------------------------+
| id | select_type | table  | type | possible_keys | key | key_len | ref | rows    | Extra                                        |
+----+-------------+--------+------+---------------+-----+---------+-----+---------+----------------------------------------------+
|  1 | SIMPLE      | t_item | ALL  | NULL          |NULL | NULL    | NULL| 1322263 | Using temporary; Using filesort |
+----+-------------+--------+------+---------------+-----+---------+-----+---------+----------------------------------------------+
1 row in set (0.01 sec)
```

根据观察发现以上 SQL 在 RDS 中执行时，出现了 Using temporary; Using filesort 现象，说明没有正确的使用索引从而导致执行缓慢。此时可以修正索引问题后重新执行。

22.6.7 如何处理DDL异常

DRDS 的 DDL 指令会在所有分表上执行对应的 DDL 操作。

失败的情况可以分为两类：

1. DDL在分库执行失败，DDL 在任意分库执行出错都可能导致各分表结构不一致。

2. 执行长时间无响应。在对大表执行 DDL 操作时，有可能由于分库的执行时间过长导致 DDL 长时间无响应。

分库执行报错的原因多种多样，如建表时表已存在、加列时列已存在等各类冲突、磁盘空间不足等。

长时间无响应一般是由分库的执行时间过长导致的。以 MySQL（RDS 原理与之类似）为例，DDL 的耗时大部分取决于该操作是 In-Place（直接在源表修改）还是 Copy Table（拷贝表数据）。In-Place 只需要修改元数据就可以了，而 Copy Table 需要重建整张表数据，此外还涉及日志和 buffer 操作。

各类操作与这两项因素的关系请参见 MySQL 官方文档 *Summary of Online Status for DDL Operations*。

判断 DDL 操作是 In-place 还是 Copy Table 操作，可以查看操作结束后 “rows affected” 这一项的返回值。

示例：

- 改变某列的默认值（非常快，完全不会影响表数据）：

```
Query OK, 0 rows affected (0.07 sec)
```

- 增加一个索引（需要花些时间 ，但是 0 rows affected 说明表数据没有被复制）：

```
Query OK, 0 rows affected (21.42 sec)
```

- 改变某列的数据类型（耗费大量时间并且需要重建表中的所有数据行）：

```
Query OK, 1671168 rows affected (1 min 35.54 sec)
```

因此，执行一个大表 DDL 操作前，可以先通过以下步骤判定这是一个快速或慢速操作：

1. 复制表结构生成一张克隆表。
2. 插入一些数据。
3. 在克隆表上执行这个 DDL 操作。
4. 检查操作完成后 “rows affected” 值是否是0。一个非0的值意味着该操作需要重建整张表，这时可能需要考虑在流量低谷去执行该操作。

失败的处理

DRDS DDL 操作会将所有 SQL 分发到所有分库上并行执行。任一分库上执行失败不会影响其他分库。另外，DRDS 还提供了 CHECK TABLE 指令来检测分表结构的一致性。因此，失败的

DDL 操作可以重新执行，已经执行成功的分库上失败报错并不会影响其他分库。只要保证最终所有分表结构一致即可。

DDL 的失败处理步骤

1. 使用 **CHECK TABLE** 指令检查表结构。如果返回结果只有一行且为状态正常则可认为**表状态一致**。此时进行步骤2，否则进行步骤3。
2. 使用 **SHOW CREATE TABLE** 指令检查表结构。如果显示的表结构符合 DDL 执行后的预期则可认为 DDL 执行成功，否则继续进行步骤3。
3. 使用 **SHOW PROCESSLIST** 指令观察所有当前执行的 SQL 状态。如有仍在执行的 DDL 操作，请等候其执行完成后再进行步骤1、2，检查表结构是否符合预期，否则进行步骤4。
4. 在 DRDS 上重新执行 DDL 操作。如果出现 **Lock conflict** 的报错请进行步骤5，否则进行步骤3。
5. 使用 **RELEASE DBLOCK** 指令释放 DDL 操作锁，然后进行步骤4。

详细操作如下：

1. 检查表结构一致性

   使用 CHECK TABLE 指令检查表结构，当返回结果只有一行且显示状态 OK 时，表明**表结构一致**。

   注意: 如果在 DMS 上执行 CHECK TABLE 没有返回结果，请在命令行下重试。

```
mysql> check table `xxxx`;
+---------------------------+-------+----------+----------+
| TABLE                     | OP    | MSG_TYPE | MSG_TEXT |
+---------------------------+-------+----------+----------+
| TDDL5_APP.xxxx            | check | status   | OK       |
+---------------------------+-------+----------+----------+
1 row in set (0.05 sec)
```

2. 检查表结构

   使用 SHOW CREATE TABLE 指令检查表结构，如果**表结构一致**且**表结构无误**时，可认为 DDL 已执行成功。

```
mysql> show create table `xxxx`;
+---------
+-----------------------------------------------------------------------------------------------------------------------+
| Table   | Create Table                                                                                                  |
+---------
+-----------------------------------------------------------------------------------------------------------------------+
|  xxxx   | CREATE TABLE `xxxx` (
`id` int(11) NOT NULL DEFAULT '0',
`NAME` varchar(1024) NOT NULL DEFAULT '',
PRIMARY KEY (`id`)
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`id`) tbpartition by hash(`id`)
 tbpartitions 3                    |
+---------
+-----------------------------------------------------------------------------------------------------------+
1 row in set (0.05 sec)
```

3. 观察当前正在执行的 SQL 语句

有些 DDL 执行速度过慢，发现 DDL 长时间无响应后，可执行 SHOW PROCESSLIST 指令观察所有当前执行的 SQL 状态。

```
mysql> SHOW PROCESSLIST WHERE COMMAND != 'Sleep';
+---------------+-----------+-------------------+-------------+---------
+-------------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------------+-----------
+---------------+-----------+
| ID            | USER      | DB                | COMMAND     | TIME    | STATE
                                  | INFO
          | ROWS_SENT | ROWS_EXAMINED | ROWS_READ |
+---------------+-----------+-------------------+-------------+---------
+-------------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------------+-----------
+---------------+-----------+
| 0-0-352724126 | ifisibhk0 | test_123_wvvp_0000 | Query       |       15 | Sending data
                                                  | /*DRDS /42.120.74.88/ac47e5a72801000/ */select
 `t_item`.`detail_url`,SUM(`t_item`.`price`) from `t_i |      NULL |         NULL |      NULL |
| 0-0-352864311 | cowxhthg0 | NULL              | Binlog Dump |     13 | Master has sent all binlog
 to slave; waiting for binlog to be updated | NULL
                                 |      NULL |         NULL |      NULL |
| 0-0-402714566 | ifisibhk0 | test_123_wvvp_0005 | Query       |       14 | Sending data
                                                  | /*DRDS /42.120.74.88/ac47e5a72801000/ */select
 `t_item`.`detail_url`,`t_item`.`price` from `t_i      |      NULL |         NULL |      NULL |
| 0-0-402714795 | ifisibhk0 | test_123_wvvp_0005 | Alter       |      114 | Sending data
                                                | /*DRDS /42.120.74.88/ac47e5a72801000/ */ALTER TABLE
 `Persons` ADD `Birthday` date                    |      NULL |         NULL |      NULL |
......
+---------------+-----------+-------------------+-------------+---------
+-------------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------------+-----------
+---------------+-----------+
12 rows in set (0.03 sec)
```

TIME 列代表该指令已经执行的秒数。发现过慢指令后，例如图中，可使用 **KILL '0-0-402714795'** 指令来取消慢指令。



注意: DRDS 中一个逻辑 SQL 对应多条分库指令，因此为了停止一个逻辑 DDL 可能需要 Kill 多条指令。从 SHOW PROCESSLIST 结果集的 INFO 列判断该指令归属的逻辑 SQL。

4. Lock conflict 报错处理

DRDS 执行 DDL 操作先会加库级锁，操作完后再释放掉。KILL DDL 操作很可能会导致该锁没有释放，此时再执行 DDL 会有以下报错：

```
Lock conflict , maybe last DDL is still running
```

此时执行 **RELEASE DBLOCK** 释放该锁即可。指令取消及锁释放后，选择业务低谷甚至停止期间，重新执行该 DDL。

其他问题

DMS 或其它客户端无法显示修改后的表结构

为了兼容某些客户端从系统表（如 COLUMNS 或 TABLES）中获取表结构的功能，DRDS 在用户0分库 RDS 里建立了一个影子库，影子库名与用户的 DRDS 逻辑库名一致。存储了所有用户库里的表结构等信息。

DMS 获取 DRDS 表结构是从影子库系统表中获取的。在处理异常 DDL 过程中，由于种种原因可能会出现用户库表结构正常修改，但是影子库中的表结构却未修改的现象。此时需要用户连接到影子库，在该库中重新对表进行一次 DDL 操作即可。

注意: CHECK TABLE 不会检测影子库表结构与用户库是否一致。

22.6.8 如何高效扫描DRDS数据

DRDS 支持高效的数据扫描方式，并支持在全表扫描时使用聚合函数进行统计汇总。

常见的扫描场景如下：

- **没有分库分表**： DRDS 会把原 SQL 传递到后端 MySQL 执行。这种情况下 DRDS 支持任何聚合函数。
- **非全表扫描**：SQL 经过 DRDS 路由后，发送到单个 MySQL 库上执行。比如说拆分键在 WHERE 中是等于关系时，就会出现非全表扫描。此时同样可以支持任何聚合函数。
- **全表扫描**：目前支持的聚合函数有 COUNT、MAX、MIN、SUM。另外在全表扫描时同样支持 LIKE、ORDER BY 、LIMIT 以及 GROUP BY 语法。
- **并行的全表扫描**：如果需要从所有库导出数据，可以通过 SHOW 指令查看表拓扑结构，针对分表并行处理。请参见下文。

通过 HINT 来进行表遍历

1. 执行 SHOW TOPOLOGY FROM TABLE_NAME 指令获取表拓扑结构。

```
mysql:> SHOW TOPOLOGY FROM DRDS_USERS;
+------+------------------+--------------+
| ID   | GROUP_NAME       | TABLE_NAME   |
+------+------------------+--------------+
|    0 | DRDS_00_RDS      | drds_users   |
|    1 | DRDS_01_RDS      | drds_users   |
+------+------------------+--------------+
2 rows in set (0.06 sec)
```

   非分库分表的表默认存储在第0个分库。

2. 针对 TOPOLOGY 进行单表遍历。

   a. 第0个分库运行当前 SQL

```
/*TDDL:node='DRDS_00_RDS'*/ SELECT * FROM DRDS_USERS;
```

   b. 第1个分库运行当前 SQL

```
/*TDDL:node='DRDS_01_RDS'*/ SELECT * FROM DRDS_USERS;
```



注意: 推荐每次扫描前执行 SHOW TOPOLOGY FROM TABLE_NAME 获取最新的表拓扑结构。

并行扫描

DRDS 支持 mysqldump 指令导出数据。但如果想要更快地扫描数据，可以针对每个分表开启多个会话的方式并行加速扫描。

```
mysql> SHOW TOPOLOGY FROM LJLTEST;
+------+---------------+------------+
| ID   | GROUP_NAME    | TABLE_NAME |
+------+---------------+------------+
|    0 | TDDL5_00_GROUP | ljltest_00 |
|    1 | TDDL5_00_GROUP | ljltest_01 |
|    2 | TDDL5_00_GROUP | ljltest_02 |
|    3 | TDDL5_01_GROUP | ljltest_03 |
|    4 | TDDL5_01_GROUP | ljltest_04 |
|    5 | TDDL5_01_GROUP | ljltest_05 |
|    6 | TDDL5_02_GROUP | ljltest_06 |
|    7 | TDDL5_02_GROUP | ljltest_07 |
|    8 | TDDL5_02_GROUP | ljltest_08 |
|    9 | TDDL5_03_GROUP | ljltest_09 |
|   10 | TDDL5_03_GROUP | ljltest_10 |
|   11 | TDDL5_03_GROUP | ljltest_11 |
+------+---------------+------------+
```

```
12 rows in set (0.06 sec)
```

如上所示该表有四个分库，每个分库有三个分表。使用以下的 SQL 对 TDDL5_00_GROUP 库上的分表进行操作：

```
/*TDDL:node='TDDL5_00_GROUP'*/ select * from ljltest_00;
```



说明： HINT 中的 TDDL5_00_GROUP 与 SHOW TOPOLOGY 指令结果中的 GROUP_NAME 列相对应。另外 SQL 中的表名为分表名。

此时可开启最多12个会话（分别对应12张分表）并行处理数据。

22.7 附录：DRDS术语表

本文列出DRDS常见术语作为参考。

术语	含义	备注
分布式关系型数据库服务DRDS	DRDS（Distributed Relational Database Service）是阿里巴巴自主研发的致力于解决单机数据库服务瓶颈问题而推出的分布式数据库产品。DRDS高度兼容MySQL协议和语法、支持自动化水平拆分、在线平滑扩缩容、弹性扩展、透明读写分离、具备数据库全生命周期运维管控能力。	
TDDL	淘宝分布式数据层（Taobao Distributed Data Layer），诞生于阿里巴巴，已成为内部近千核心应用首选组件。	
DRDS Console	普通用户使用的DRDS运维系统。	
DRDS Manager	专业运维人员使用的DRDS运维系统。	
DRDS Server	DRDS的服务节点，无状态，对外提供MySQL协议的服务。解析SQL并到RDS上执行，通过MySQL协议返回正确的执行结果。	
负载均衡器	由于DRDS Server节点无状态，请求可以随机路由到任意一台DRDS Server节点。负载均衡器用于完成此任务。在专有云上，负载均衡器一般为VIPServer。	
Diamond	DRDS主要元数据的存储服务。	

术语	含义	备注
精卫（Jing Wei）	通过解析Binlog提供异构索引、广播表等服务。	
愚公（Yu Gong）	提供数据导入、扩容缩容等服务。	
DRDS实例（DRDS Instance）	若干个DRDS Server节点构成一个DRDS实例，一个DRDS实例中可以存在多个DRDS数据库。	
DRDS实例ID（DRDS Instance ID）	一个唯一ID来标明一个DRDS实例。	
DRDS 实例节点数	DRDS实例中DRDS Server的节点数。	
VIP	负载均衡器（Virtual Internet Protocol），分为： 1 公网VIP，可公网访问，一般用于测试； 2 私网VIP，仅限于阿里云内网访问。	
VPC	专有网络（Virtual Private Cloud），一般见于公共云。	
Region	地域信息，一般见于公共云，如华东。	
Azone	可用区信息，一般见于公共云，如华东1可用区。	
逻辑SQL	由应用端发送到DRDS的SQL称为逻辑SQL。	
物理SQL	由DRDS对逻辑SQL进行解析之后发送到RDS上执行的SQL称为物理SQL。	逻辑SQL和物理SQL可能一样，也可能不一样。 逻辑SQL和物理SQL可能是一对一的关系，也可能是一对多的关系。
每秒查询数（QPS）	一个统计周期内，平均每秒钟DRDS执行的逻辑SQL数。	并非事务数。大多数控制语句如COMMIT操作，SET语句等，不计入其中。

术语	含义	备注
平均响应时间（RT）	一个统计周期内，DRDS执行逻辑SQL的平均响应时间，单位毫秒。 一个SQL的响应时间计算方式为：（DRDS写出结果集的最后一个包的时间点）-（DRDS接收到SQL的时间点）	
物理每秒查询数（物理QPS）	一个统计周期内，平均每秒钟DRDS在RDS上执行的物理SQL数。	
物理平均响应时间（物理RT）	一个统计周期内，DRDS在RDS上执行逻辑物理SQL的平均响应时间，单位毫秒。 一个物理SQL的响应时间计算方式为：（DRDS收到RDS返回结果集的时间点）-（DRDS开始获取RDS连接的时间点）	这中间包含了与RDS建立连接或者从连接池获取连接的时间、网络传输的时间、RDS执行SQL的时间。
连接数	应用与DRDS所建立的连接数。	并非DRDS与RDS建立的连接数。
接收流量	应用发往DRDS的SQL的网络流量。	此数据与DRDS同RDS的交互无关。
发送流量	DRDS发往应用的结果集的网络流量。	此数据与DRDS同RDS的交互无关。
活跃线程数（ThreadRunning）	DRDS正在运行的线程数，一般可以用于表示DRDS的负载情况。	
Global	一个DRDS实例中所有库的监控数据的累加。	
内存利用率	DRDS Server进程的JVM内存利用率。	
总内存利用率	DRDS Server所在机器的内存利用率。	目前仅当DRDS Server部署在ECS中才有此指标，一般为公共云。
CPU利用率	DRDS Server所在机器的CPU利用率。	目前仅当DRDS Server部署在ECS中才有此指

术语	含义	备注
		标，一般为公共云。
系统负载（Load）	DRDS Server所在机器的负载情况。	目前仅当DRDS Server部署在ECS中才有此指标，一般为公共云。
服务端口	DRDS Server对外提供MySQL协议的端口。	一般情况为3306，但是当一台机器部署多个DRDS节点时（多为物理机），端口会有所变化。
管理端口	DRDS Server提供管理接口的端口。	一般情况为服务端口+100。
启动时间	DRDS Server节点启动的时间。	
运行时间	DRDS Server节点从上次启动的持续运行时间。	
总内存	DRDS Server节点JVM内存的最大值。	
使用内存	DRDS Server节点已使用的JVM内存。	
节点数目	实例的机器数目，必填。一个DRDS实例本质就是一个DRDS集群，节点数目是指集群中的机器数。	
实例类型	实例的服务类型，分为专享与共享，必选。专享实例是用户独占的工作模式；共享实例是使用多租户的工作模式，一般用于公共云。	
机器类型	实例的机器类型，分为自动选择、物理机（PHY）与虚拟机（ECS），必选。DRDS的库存按部署的机器类型分为物理机库存与虚拟机库存，两个类型一般不可以混用（因为部署和运维方式不一样）。	

术语	含义	备注
AliUid	实例的所属UID，必填。在专有云下，在专有云环境下，UID由部署环境的账号系统提供。	
后端端口	VIP的后端端口，对于DRDS Server来说，这个端口就是Server机器的服务端口。	
前端端口	VIP的前端端口，供用户访问的端口。每个VIP都有一组前端端口与后端端口，VIP会将前端端口的数据转发到后端端口。	
私网/公网	VIP的网络类型。分为： 1 公网：公网的VIP，支持公网访问。 2 私网：私网的VIP（包括VPC VIP），支持私网的访问。	
lbId	负载均衡实例ID，是VIP的唯一标识，VIP的管理都基于这个标识。	
转发模式	VIP的转发模式。有以下几种： 1 Fnat：当后端机器是虚拟机或需要支持VPC，需选此模式，建议使用模式； 2 Nat：当后端机器是物理机，可选此模式，目前仅用于公共云； 3 OpenFnat：仅用于公共云，专有云请忽略。	
专有网络ID（VpcID）	目标的VPC，即需要进入的VPC。	
虚拟交换机ID（VSwitchID）	目标的虚拟交换机ID，它决定实例的VPC VIP所处网段。	
APPName	目标DRDS DB的APP NAME。每一个DRDS DB 都有一个与这相对应的APP NAME，用于加载配置。	
UserName	目标DRDS DB的登录数据库的用户名。	
DBName	目标DRDS DB的登录数据库的名字。	
IP白名单	只允许IP白名单中指定的IP地址访问DRDS实例。	
只读实例	物理数据库实例按照是否允许写入分为两种：	

术语	含义	备注
	1. 主实例：允许读写，专有云环境中为MySQL主备中的主实例，公共云环境中为RDS主实例； 2. 只读实例：只允许读取，专有云环境中为MySQL主备中的备实例，公共云环境中为RDS只读实例。	
读SQL	读取数据的SQL分类，如SELECT语句。只有当SQL语句不在事务中时，DRDS才会判断是否为读SQL，否则在读写分离时按照写SQL处理。	
读写分离	如果物理数据库实例带有只读实例，可以在DRDS Console上配置，将读SQL按比例分配到主实例和只读实例上。DRDS将自动识别SQL类型，按比例分配执行。	
平滑扩缩容	在水平拆分的基础上，通过动态调整数据在物理数据库实例上的分布，达到扩缩容的目的。通常扩缩容过程异步完成，并且不需要业务代码做任何修改。	
小表广播	通过预先将一些单库单表的数据同步到所有分库上，达到将跨库联合查询转化为在物理数据库上分别完成的可下推联合查询。	
水平拆分	按照指定规则，将原本保存在一张表中的数据行，分散到多张表中，实现水平线性扩展。	
拆分	指在一个MySQL实例上创建多个分库，多个分库共同组成一个DRDS数据库，该模式下可以完整的使用DRDS的所有功能。	
非拆分	指使用一个已经在MySQL(RDS)上建好的数据库作为一个DRDS数据库，该模式下不能使用DRDS的分库分表等功能，仅仅能使用DRDS的读写分离的功能。	
导入库	创建DRDS DB过程中特有的概念，指MySQL(RDS)实例中的一个已经建好的数据库。	

术语	含义	备注
读策略	DRDS分配读SQL到主实例和只读实例的比例。	
全表扫描	如果SQL语句中没有指定拆分字段，DRDS将在所有分表上执行SQL并汇总结果，性能开销较大，可以选择关闭这一功能。	
拆分键	逻辑表中的一列，DRDS按照这一列将数据和SQL语句路由到物理表上。	
数据导入	从现有的MySQL(RDS)实例上的数据导入到DRDS数据库中。	
全量迁移	迁移数据库中已有的所有记录到DRDS中。全量迁移开始前会记录一个位点。	
位点	MySQL Binlog文件中，每一行代表一个数据变更操作，Binlog文件中一行的位置称为位点。	
增量迁移	从记录的位点开始读取MySQL Binlog的所有记录，转换成SQL后在DRDS中执行。执行切换操作前，增量迁移将持续运行。	
切换	数据导入和平滑扩容的一个步骤，将MySQL Binlog中剩余的所有增量记录写入DRDS。	
清理	平滑扩容的最后一个步骤，清理平滑扩容过程中产生的冗余数据和配置。	
异构索引	对于DRDS的数据库的分表，查询时SQL中的WHERE条件尽量会带上分库分表键，这样DRDS会将这个查询路由到具体的分库中，以提高查询效率。如果SQL的WHERE条件中没有分库分表键，DRDS会进行一次全表扫描。针对这种场景，DRDS提供了异构索引来解决这个问题。将DRDS某分库分表中的数据以不同的分库分表键同步或部分同步到另一张表，同步到的目标表叫做异构索引表。	
全局唯一数字序列（Sequence）	DRDS全局唯一数字序列（64位数字，对应MySQL中BIGINT类型）的主要目标是为了保证所定义唯一字段中的数据的全局唯	

术语	含义	备注
	一（比如PRIMARY KEY，UNIQUE KEY等）和有序递增。	
DRDS自定义注释（Hint）	为了方便使用DRDS，DRDS定义了一些注释，用于指定一些特殊行为。	

23 消息队列MQ

23.1 产品概述

消息队列（Message Queue，简称 MQ）是阿里巴巴集团中间件技术部自主研发的专业消息中间件。产品基于高可用分布式集群技术，提供消息发布订阅、消息轨迹查询、定时（延时）消息、资源统计、监控报警等一系列消息云服务，是企业级互联网架构的核心产品。MQ 历史超过 9 年，为分布式应用系统提供异步解耦、削峰填谷的能力，同时具备海量消息堆积、高吞吐、可靠重试等互联网应用所需的特性，是阿里巴巴双11使用的核心产品，每年天猫双十一全天提供 99.99% 可用性。

MQ 目前提供 TCP、HTTP、MQTT 等协议层面的接入方式，支持 Java、C++ 以及 .NET 不同语言，方便不同编程语言开发的应用快速接入 MQ 消息云服务。您可以将应用部署在阿里云 ECS、企业自建云，或者嵌入到移动端、物联网设备中与 MQ 建立连接进行消息收发，同时本地开发者也可以通过公网接入 MQ 服务进行消息收发。

图 594: MQ产品示意图



23.2 名词解释

为了方便您更好地理解相关概念并使用 MQ ，在介绍如何快速上手使用MQ前，您可以先了解下相关的专有名词及术语。

Message Queue	消息队列，阿里云企业级互联网架构的核心产品，提供基于高可用分布式集群技术搭建的消息发布订阅、轨迹查询、资源统计、定时（延时）、监控报警等一系列消息云服务。
Message	消息，消息队列中信息传递的载体。
Message ID	消息的全局唯一标识，由 MQ 系统自动生成，唯一标识某条消息。
Message Key	消息的业务标识，由消息生产者（Producer）设置，唯一标识某个业务逻辑。
Topic	消息主题，一级消息类型，通过 Topic 对消息进行分类。
Tag	消息标签，二级消息类型，用来进一步区分某个 Topic 下的消息分类。
Producer	消息生产者，也称为消息发布者，负责生产并发送消息。
Producer ID	一类 Producer 的标识。属于同一个 Producer ID的消息生产者通常生产并发送一类消息，且发送逻辑一致。
Producer 实例	Producer 的一个对象实例，不同的 Producer 实例可以运行在不同进程内或者不同机器上。Producer 实例线程安全，可在同一进程内多线程之间共享。
Consumer	消息消费者，也称为消息订阅者，负责接收并消费消息。
Consumer ID	一类 Consumer 的标识，这类 Consumer 通常接收并消费一类消息，且消费逻辑一致。
Consumer 实例	Consumer 的一个对象实例，不同的 Consumer 实例可以运行在不同进程内或者不同机器上。一个 Consumer 实例内配置线程池消费消息。
集群消费	一个 Consumer ID 所标识的所有 Consumer 平均分摊消费消息。例如某个 Topic 有 9 条消息，一个 Consumer ID 有 3 个 Consumer 实例，那么在集群消费模式下每个实例平均分摊，只消费其中的 3 条消息。
广播消费	一个 Consumer ID 所标识的所有 Consumer 都会各自消费某条消息一次。例如某个 Topic 有 9 条消息，一个 Consumer ID 有 3 个 Consumer 实例，那么在广播消费模式下每个实例都会各自消费 9 条消息。

定时消息	Producer 将消息发送到 MQ 服务端，但并不期望这条消息立马投递，而是推迟到在当前时间点之后的某一个时间投递到 Consumer 进行消费，该消息即定时消息。
延时消息	Producer 将消息发送到 MQ 服务端，但并不期望这条消息立马投递，而是延迟一定时间后才投递到 Consumer 进行消费，该消息即延时消息。
顺序消息	MQ 提供的一种按照顺序进行发布和消费的消息类型，分为全局顺序消息和分区顺序消息。
消息堆积	Producer 已经将消息发送到 MQ 服务端，但由于 Consumer 消费能力有限，未能在短时间内将所有消息正确消费掉，此时在 MQ 服务端保存着未被消费的消息，该状态即消息堆积。
消息过滤	订阅者可以根据消息标签（Tag）对消息进行过滤，确保订阅者最终只接收被过滤后的消息类型。消息过滤在 MQ 服务端完成。
消息轨迹	在一条消息从发布者发出到订阅者消费处理过程中，由各个相关节点的时间、地点等数据汇聚而成的完整链路信息。通过消息轨迹，用户能清晰定位消息从发布者发出，经由 MQ 服务端，投递给消息订阅者的完整链路，方便定位排查问题。
重置消费位点	以时间轴为坐标，在消息持久化存储的时间范围内（默认3天），重新设置消息订阅者对其订阅 Topic 的消费进度，设置完成后订阅者将接收设定时间点之后由消息发布者发送到 MQ 服务端的消息。

23.3 快速开始

本节主要描述快速接入 MQ以及用 MQ 进行消息收发的完整流程。主要目的在于用简单明了的方式引导您快速上手 MQ，为进一步使用和熟悉 MQ 的功能提供入门。

本章节主要内容如下：

- 如何申请 MQ 资源
- 如何通过 MQ 进行消息收发

MQ快速接入流程图：

图 595: MQ接入流程



MQ支持的接入协议和开发语言一览表

接入方式	Java	C/C++	.NET	PHP	Python
TCP(SDK)	支持	支持	支持	不支持	不支持
MQTT	支持	不支持	不支持	不支持	不支持

23.3.1 申请资源

本节介绍使用MQ的第一步，即资源申请。

在 MQ 消息系统中，消息发布者将消息发送到某个指定的消息主题（Topic），而消息订阅者则通过订阅该指定的 Topic 来获取和消费消息。因此一个新的应用接入 MQ 需要先申请相关的 MQ 资源，包括消息主题（Topic）的创建，以及为 Topic 建立消息生产关系（Producer ID）和消息消费关系（Consumer ID）。

同时，在调用 SDK/API 进行消息发送和订阅的时候，除了需要指定创建的 Topic， Producer ID 以及 Consumer ID 以外，还需输入您在阿里云服务器管理控制台创建的身份验证信息，即 AccessKey 和 SecretKey。

如果还没创建访问秘钥，您可以登录阿里云服务器管理控制台进行创建。

23.3.1.1 创建Topic

消息主题（Topic）是 MQ 里对消息进行的一级归类，例如可以创建*Topic_Trade*来识别交易类消息。使用MQ的第一步需要先为您的应用申请 Topic 资源，即创建 Topic。

操作步骤

1. 登录 MQ 控制台，在左侧菜单栏选择**Topic 管理**。
2. 在页面左上角选择相应的域（比如**VPCabcd**），然后单击页面右上角的**发布 Topic**按钮。
3. 在**发布Topic**对话框输入 Topic 名称及备注，单击**确定**。

   您创建的 Topic 将出现在 Topic 列表中。

   注意: Topic名称必须全局唯一。如果名称已经被其他用户使用，您将无法创建相同名称的Topic。

图 596: 发布Topic



23.3.1.2 创建Producer ID

创建好 Topic 后，要为这个 Topic 申请消息生产端的资源，即创建 Producer ID。一个 Topic 只能对应一个 Producer ID。

操作步骤

1. 在 **Topic 管理**页面，找到您刚刚创建的 Topic。
2. 单击操作选项中的**申请发布**。
3. 在**申请发布**对话框输入 Producer ID，单击**确定**。

说明:

- Producer ID 必须全局唯一。如果名称已存在，您将无法创建相同名称的 Producer ID。
- Topic对应的生产端必须和这个 Topic 在同一个域，比如您在“VPCabc”域创建了“*Topic_open*”，那么和“*Topic_open*”对应的 Producer ID 也必须在同一个域。
- Producer ID 和 Topic 的关系是 1:N，即一个 Topic 只能绑定一个 Producer ID，但是同一个 Producer ID 可以对应多个 Topic。

23.3.1.3 创建Consumer ID

创建好消息生产关系后，您需要为 Topic 申请相应的消息消费资源，即创建 Consumer ID。

操作步骤

1. 在 **Topic管理**页面，找到您刚刚创建的 Topic，单击右侧操作选项里的**申请订阅**。。
2. 在**订阅Topic**对话框输入 Consumer ID，单击**确定**。

图 597: 订阅Topic



注意:

- Consumer ID 必须全局唯一。如果名称已存在，您将无法创建相同名称的 Consumer ID。
- Consumer ID 必须和对应的 Topic 在同一个域，比如给“公网测试”域的“*Topic_open*”绑定 Consumer ID “*CID_123*”后，“华北1”域内的“*Topic_huabei1*”将不能绑定同一个 Consumer ID。
- Consumer ID 和 Topic 的关系是 N:N。同一个 Consumer ID 可以订阅多个 Topic，同一个 Topic 也可以对应多个 Consumer ID。

23.3.2 发送消息

在控制台申请 Topic 资源后，就可以进行消息发送了。MQ 提供两种发送消息的方式，即通过控制台发送消息或者调用 SDK/API 发送消息。

控制台发送消息通常用于快速验证 Topic 资源的可用性。在生产环境下使用 MQ 则建议您调用相关的 SDK/API 进行消息发送。

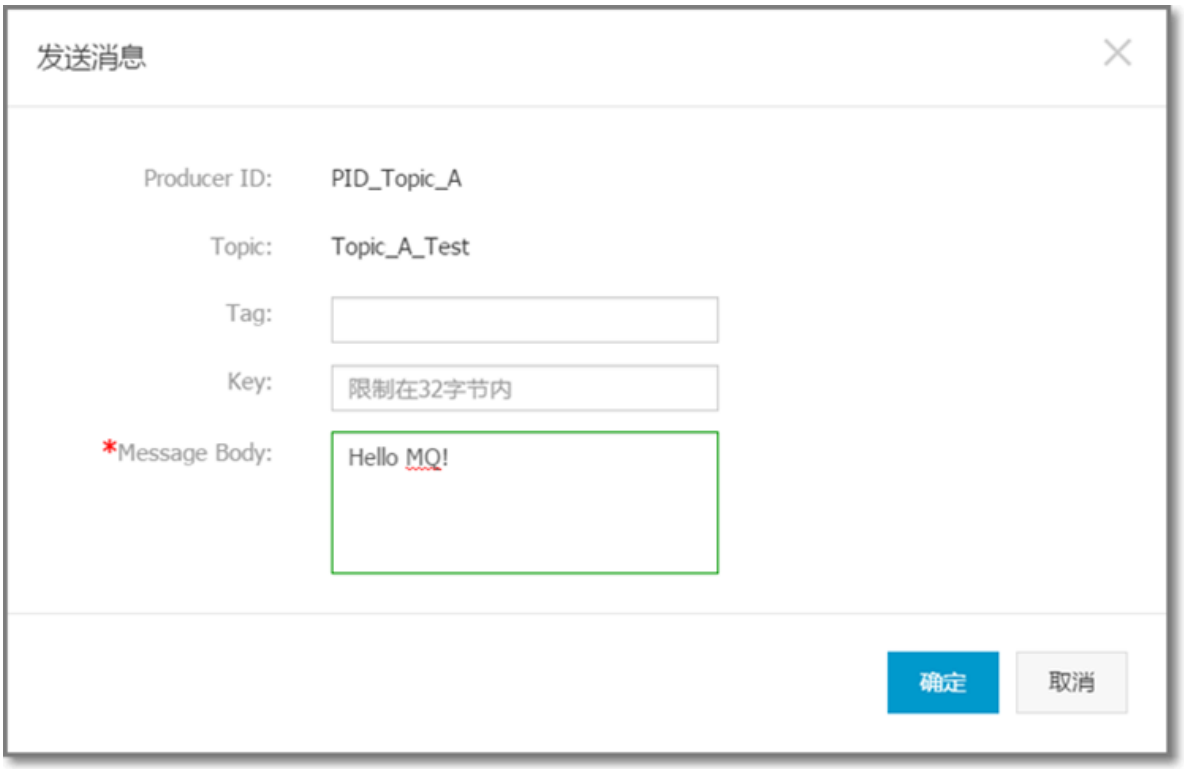
23.3.2.1 控制台发送消息

MQ 提供两种发送消息的方式，即通过控制台发送消息或者调用 SDK/API 发送消息。在控制台申请 Topic 资源后，您可以在控制台发送消息来快速验证 Topic 资源的可用性。在生产环境下使用 MQ 则建议您调用相关的 SDK/API 进行消息发送。

操作步骤

1. 在 MQ 控制台的左侧菜单栏依次单击**发布订阅管理** > **发布管理**。
2. 在消息列表中找到您刚刚创建的 Topic，单击右侧操作选项里的**发送**。
3. 在**发送消息**对话框输入消息的具体内容，比如“Hello MQ！”，单击**确定**。控制台会返回消息发送成功通知以及相应的 Message ID。

图 598: 发送消息



23.3.2.2 调用SDK/API发送消息

在生产环境使用 MQ，建议您调用 SDK/API 来进行消息发送。本文以 TCP 协议下调用 Java SDK 为例进行说明。

操作步骤

1. 通过下面两种方式可以引入依赖（任选一种）：

- Maven 方式引入依赖：

```
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>ons-client</artifactId>
<version>1.2.1</version>
</dependency>
```

- 下载依赖*JAR*包

2. 根据以下说明设置相关参数，运行示例代码：

```
public class ProducerTest {
 public static void main(String[] args) {
    Properties properties = new Properties();
    properties.put(PropertyKeyConst.ProducerId, "XXX");// 您在MQ控制台创建的Producer ID
     properties.put(PropertyKeyConst.AccessKey,"XXX");// 鉴权用AccessKey，在阿里云服务器管理控制台创建
    properties.put(PropertyKeyConst.SecretKey, "XXX");// 鉴权用SecretKey，在阿里云服务器管理控制台创建
    Producer producer = ONSFactory.createProducer(properties);
    // 在发送消息前，必须调用start方法来启动Producer，只需调用一次即可
    producer.start();

    //循环发送消息
    while(true){
       Message msg = new Message( //
          // Message Topic
          "TopicTestMQ",
          // Message Tag,
         // 可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条件在MQ服务器过滤
          "TagA",
          // Message Body
          // 任何二进制形式的数据， MQ不做任何干预，
          // 需要Producer与Consumer协商好一致的序列化和反序列化方式
          "Hello MQ".getBytes());
      // 设置代表消息的业务关键属性，请尽可能全局唯一，以方便您在无法正常收到消息情况下，可通过MQ控制台查询消息并补发
       // 注意：不设置也不会影响消息正常收发
       msg.setKey("ORDERID_100");
       // 发送消息，只要不抛异常就是成功
       // 打印Message ID，以便用于消息发送状态查询
       SendResult sendResult = producer.send(msg);
                    System.out.println("Send Message success. Message ID is: " + sendResult.getMessageId());
    }

    // 在应用退出前，可以销毁Producer对象
    // 注意：如果不销毁也没有问题
    producer.shutdown();
}
```

```
}
```

23.3.2.3 查看消息是否发送成功

消息发送后，您可以在控制台查看消息发送状态。

操作步骤

1. 在 MQ 控制台左侧菜单栏依次单击**消息查询** > **Message ID 查询**。
2. 在 Message ID 查询页面输入发送消息后返回的 Message ID，单击**搜索**查询消息发送状态，如下图所示。

图 599: 消息查询



说明:

- StoreTime 表示 MQ 服务端存储这条消息的时间。如果查询到此消息，表示消息已经成功发送到服务端。
- 此例子中，订阅端从未启动过，所以消息状态显示“暂无消费数据”。更多消息状态，请参考消息查询。

23.3.3 订阅消息

消息发送成功后，订阅方就可以对消息进行订阅。订阅消息需要通过调用相关协议及开发语言的SDK/API 来完成。本文以调用TCP Java SDK订阅消息为例进行说明。

操作步骤

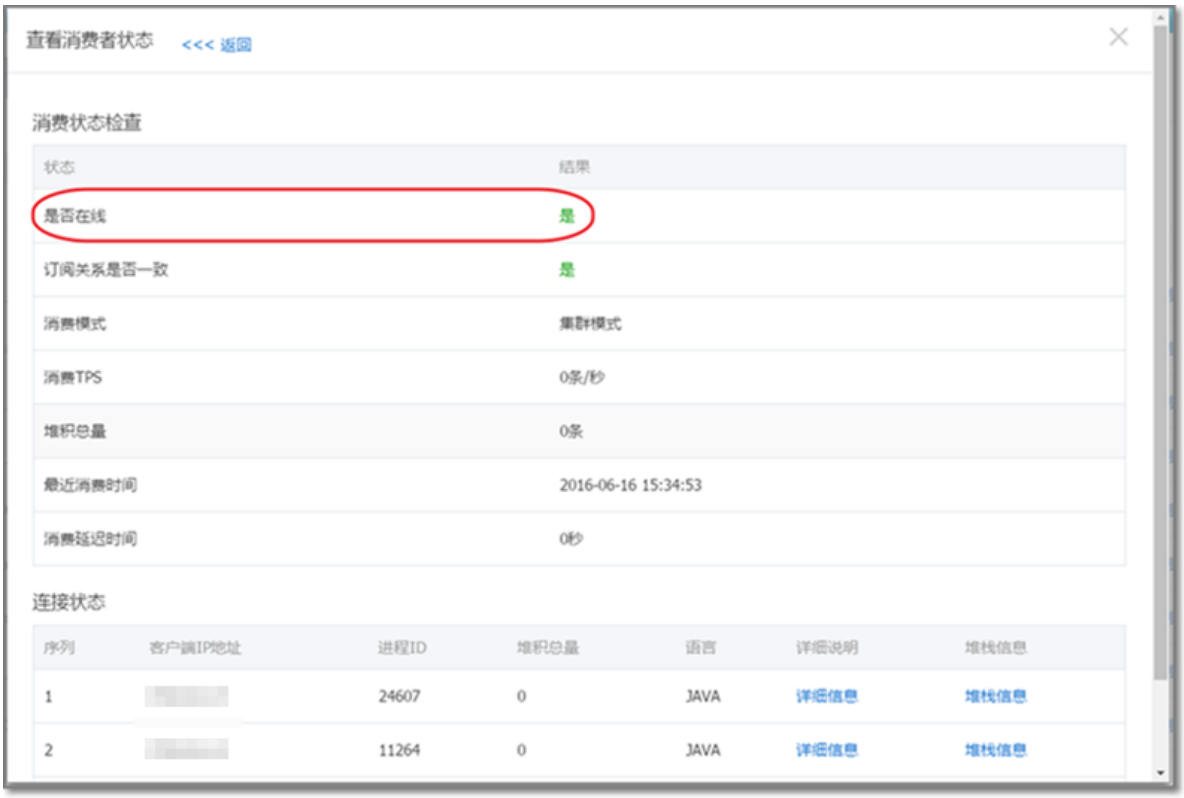
1. 运行以下示例代码来测试订阅消息的功能。运行代码前请按照说明正确设置相关参数。

```
public class ConsumerTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
       properties.put(PropertyKeyConst.ConsumerId, "XXX");// 您在MQ控制台创建的Consumer
 ID
        properties.put(PropertyKeyConst.AccessKey, "XXX");// 鉴权用AccessKey，在阿里云服
务器管理控制台创建
        properties.put(PropertyKeyConst.SecretKey, "XXX");// 鉴权用SecretKey，在阿里云服务
器管理控制台创建
        Consumer consumer = ONSFactory.createConsumer(properties);
        consumer.subscribe("TopicTestMQ", "*", new MessageListener() {
            public Action consume(Message message, ConsumeContext context) {
                System.out.println("Receive: " + message);
                return Action.CommitMessage;
            }
        });
        consumer.start();
        System.out.println("Consumer Started");
    }
}
```

完成后，可以在MQ控制台查看订阅端是否成功启动，即消息订阅是否完成。

2. 在 MQ 控制台左侧菜单栏依次单击**发布订阅管理** > **订阅管理**。
3. 找到要查看的 Topic，单击右侧操作选项里的**消费者状态**。
   - 如果**是否在线**显示为**是**，则说明订阅端已成功启动。
   - 如果消费者状态**是否在线**显示为**否**，请参考消费者状态问题排查。

图 600: 消费端状态



完成以上所有步骤后，您就成功接入了 MQ 服务，可以用 MQ 进行消息发送和订阅了。

23.4 MQ消息类型

本节介绍MQ支持的消息类型。

23.4.1 事务消息

本节主要介绍 MQ 事务的概念、适用场景以及使用过程中的注意事项。

基本概念

MQ 事务消息用于帮助您实现类似 X/Open XA 的分布事务功能，通过 MQ 事务消息能达到分布式事务的最终一致。MQ的事务消息包含以下概念：

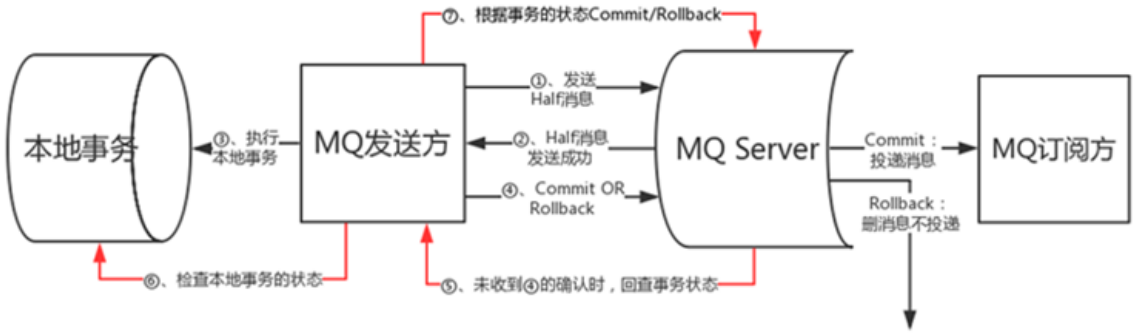
- **事务消息**：MQ 提供类似 X/Open XA 的分布事务功能，通过 MQ 事务消息能达到分布式事务的最终一致。
- **半消息**：暂不能投递的消息，发送方已经将消息成功发送到了 MQ 服务端，但是服务端未收到生产者对该消息的二次确认，此时该消息被标记成“暂不能投递”状态，处于该种状态下的消息即半消息。

- **消息回查**：由于网络闪断、生产者应用重启等原因，导致某条事务消息的二次确认丢失，MQ 服务端通过扫描发现某条消息长期处于“半消息”时，需要主动向消息生产者询问该消息的最终状态（Commit 或是 Rollback），该过程即消息回查。

原理介绍

MQ 事务消息交互流程如下所示：

图 601: 事务消息交互流程图



其中：

1. 发送方向 MQ Server发送消息。
2. MQ Server 将消息持久化成功之后，向发送方 ACK 确认消息已经发送成功，此时消息为半消息。
3. 发送方开始执行本地事务逻辑。
4. 发送方根据本地事务执行结果向 MQ Server 提交二次确认（Commit 或是 Rollback），MQ Server 收到 Commit 状态则将半消息标记为可投递，订阅方最终将收到该消息；MQ Server 收到 Rollback 状态则删除半消息，订阅方将不会接受该消息。
5. 在断网或者是应用重启的特殊情况下，上述步骤4提交的二次确认最终未到达 MQ Server，经过固定时间后 MQ Server 将对该消息发起消息回查。
6. 发送方收到消息回查后，需要检查对应消息的本地事务执行的最终结果。
7. 发送方根据检查得到的本地事务的最终状态再次提交二次确认，MQ Server 仍按照步骤4对半消息进行操作。

事务消息发送对应步骤1、2、3、4，事务消息回查对应步骤5、6、7，具体接入手册请参考不同开发语言相关章节。

注意事项

1. 通过 ONSFactory.createTransactionProducer 创建事务消息的 Producer 时必须指定 LocalTransactionChecker 的实现类，处理异常情况下事务消息的回查。
2. 事务消息发送完成本地事务后，可在 execute 方法中返回如下三种状态：
   - TransactionStatus.CommitTransaction 提交事务，允许订阅方消费该消息。
   - TransactionStatus.RollbackTransaction 回滚事务，消息将被丢弃不允许消费。
   - TransactionStatus.Unknow 暂时无法判断状态，期待固定时间以后 MQ Server 向发送方进行消息回查。
3. 可通过如下方式给每条消息设定第一次消息回查的最快时间。

```
Message message = new Message();
//在消息属性中添加第一次消息回查的最快时间，单位秒，如下设置实际第一次回查时间为
 120 ~ 125 秒之间
message.putUserProperties(PropertyKeyConst.CheckImmunityTimeInSeconds,"120");
```

注意: 以上方式只确定事务消息的第一次回查的最快时间，实际回查时间向后浮动0~5秒；如第一次回查后事务仍未提交，后续每隔5秒回查一次。

23.4.2 定时消息和延时消息

本节主要介绍 MQ 定时消息和延时消息的概念、适用场景以及使用过程中的注意事项。

基本概念

- 定时消息：Producer 将消息发送到 MQ 服务端，但并不期望这条消息立马投递，而是推迟到在当前时间点之后的某一个时间投递到 Consumer 进行消费，该消息即定时消息。
- 延时消息：Producer 将消息发送到 MQ 服务端，但并不期望这条消息立马投递，而是延迟一定时间后才投递到 Consumer 进行消费，该消息即延时消息。

定时消息与延迟消息在代码配置上存在一些差异，但是最终达到的效果相同：消息在发送到 MQ 服务端后并不会立马投递，而是根据消息中的属性延迟固定时间后才投递给消费者。

适用场景

定时/延时消息适用于如下一些场景：

- 消息生产和消费有时间窗口要求：比如在电商交易中超时未支付关闭订单的场景，在订单创建时会发送一条 MQ 延时消息，这条消息将会在30分钟以后投递给消费者，消费者收到此消息后需要判断对应的订单是否已完成支付。如支付未完成，则关闭订单，如已完成支付则忽略。
- 通过消息触发一些定时任务，比如在某一固定时间点向用户发送提醒消息。

使用方式

定时消息、延时消息的使用在代码编写上存在略微的区别：

- 发送**定时消息**需要明确指定消息发送时间点之后的某一时间点作为消息投递的时间点。

发送**延时消息**时需要设定一个延时时间长度，消息将从当前发送时间点开始延迟固定时间之后才开始投递。

- 发送定时消息

```
public class ProducerDelayTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
     properties.put(PropertyKeyConst.ProducerId, "XXX");//您在 MQ 控制台创建的 Producer ID
         properties.put(PropertyKeyConst.AccessKey, "XXX");// 阿里云身份验证，在阿里云服务器管理控制台创建
          properties.put(PropertyKeyConst.SecretKey, "XXX");// 阿里云身份验证，在阿里云服务器管理控制台创建
        Producer producer = ONSFactory.createProducer(properties);
        // 在发送消息前，必须调用 start 方法来启动 Producer，只需调用一次即可。
        producer.start();
        Message msg = new Message( //
                // Message Topic
                "Topic",
                 // Message Tag 可理解为 Gmail 中的标签，对消息进行再归类，方便 Consumer 指定过滤条件在 MQ 服务器过滤
                "tag",
            // Message Body 可以是任何二进制形式的数据， MQ 不做任何干预，需要 Producer 与 Consumer 协商好一致的序列化和反序列化方式
                "Hello MQ".getBytes());
        // 设置代表消息的业务关键属性，请尽可能全局唯一
        // 以方便您在无法正常收到消息情况下，可通过 MQ 控制台查询消息并补发。
        // 注意：不设置也不会影响消息正常收发
        msg.setKey("ORDERID_100");
        /**
         * 定时消息投递，设置投递的具体时间戳，单位毫秒例如2016-03-07 16:21:00投递
        */
        long timeStamp =new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse("2016-03-07 16:21:00").getTime();
        msg.setStartDeliverTime(timeStamp);

        // 发送消息，只要不抛异常就是成功
        SendResult sendResult = producer.send(msg);
        System.out.println("Message Id:" + sendResult.getMessageId());
        // 在应用退出前，销毁 Producer 对象<br>
        // 注意：如果不销毁也没有问题
        producer.shutdown();
    }
}
```

- 发送延时消息

```
public class ProducerDelayTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
```

```
        properties.put(PropertyKeyConst.ProducerId, "XXX");// 您在控制台创建的 Producer ID
        properties.put(PropertyKeyConst.AccessKey, "XXX");// AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
        properties.put(PropertyKeyConst.SecretKey, "XXX");// SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建
        Producer producer = ONSFactory.createProducer(properties);
        // 在发送消息前，必须调用 start 方法来启动 Producer，只需调用一次即可。
        producer.start();
        Message msg = new Message( //
                // Message Topic
                "Topic",
                // Message Tag, 可理解为 Gmail 中的标签，对消息进行再归类，方便 Consumer 指定过滤条件在 MQ 服务器过滤
                "tag",
            // Message Body 可以是任何二进制形式的数据， MQ 不做任何干预，需要 Producer 与 Consumer 协商好一致的序列化和反序列化方式
                "Hello MQ".getBytes());
        // 设置代表消息的业务关键属性，请尽可能全局唯一。
        // 以方便您在无法正常收到消息情况下，可通过 MQ 控制台查询消息并补发。
        // 注意：不设置也不会影响消息正常收发
        msg.setKey("ORDERID_100");
      // deliver time 单位 ms，指定一个时刻，在这个时刻之后才能被消费，这个例子表示 3s 后才能被消费
        long delayTime = 3000;
        msg.setStartDeliverTime(System.currentTimeMillis() + delayTime);
        // 发送消息，只要不抛异常就是成功
        SendResult sendResult = producer.send(msg);
        System.out.println("Message Id:" + sendResult.getMessageId());
        // 在应用退出前，销毁 Producer 对象
        // 注意：如果不销毁也没有问题
        producer.shutdown();
    }
}
```

注意:

- 定时/延时消息 msg.setStartDeliverTime 的参数需要设置成当前时间戳之后的某个时刻（单位毫秒），如果被设置成当前时间戳之前的某个时刻，消息将立刻投递给消费者。
- 定时/延时消息 msg.setStartDeliverTime 的参数可设置40天内的任何时刻（单位毫秒），超过40天消息发送将失败。
- 消费者存在消息堆积时，定时/延时消息将不能严格按照配置的时间进行消费。

23.4.3 顺序消息

本文主要介绍 MQ 顺序的概念、适用场景以及使用过程中的注意事项。

SDK支持

请使用 Java SDK 1.2.7 及以上版本，具体示例代码请参考发送顺序消息文档。

概念介绍

顺序消息是 MQ 提供的一种按照顺序进行发布和消费的消息类型。

顺序消息由两个部分组成：顺序发布和顺序消费。

- **顺序发布**：对于指定的一个 Topic，客户端将按照一定的先后顺序进行发送消息。
- **顺序消费**：对于指定的一个 Topic，按照一定的先后顺序进行接收消息，即先发送的消息一定会先被客户端接收到。

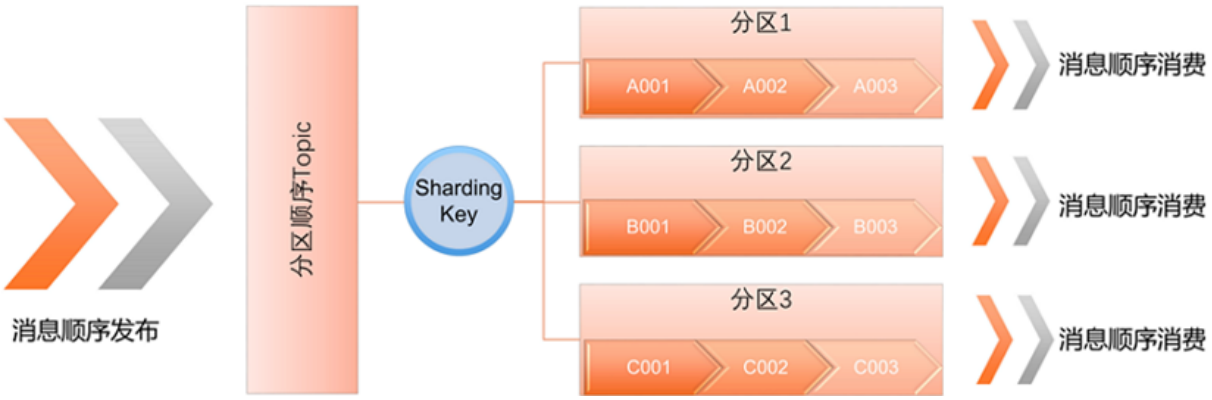
顺序消息类型分为两种：全局顺序和分区顺序。

- **全局顺序**：对于指定的一个 Topic，所有消息按照严格的先入先出（FIFO）的顺序进行发布和消费。
- **分区顺序**：对于指定的一个 Topic，所有消息根据 sharding key 进行区块分区。同一个分区内的消息按照严格的 FIFO 顺序进行发布和消费。Sharding key 是顺序消息中用来区分不同分区的关键字段，和普通消息的 key 是完全不同的概念。

图 602: 全局顺序消息示意图



图 603: 分区顺序消息示意图



适用场景

MQ **全局顺序消息**适用于以下场景：

- 性能要求不高，所有的消息严格按照 FIFO 原则进行消息发布和消费的场景。

MQ **分区顺序消息**适用于如下场景：

- 性能要求高，以 sharding key 作为分区字段，在同一个区块中严格的按照 FIFO 原则进行消息发布和消费的场景。

  举例说明：

  - 【例一】

    用户注册需要发送发验证码，以用户 ID 作为 sharding key，那么同一个用户发送的消息都会按照先后顺序来发布和订阅。

  - 【例二】

    电商的订单创建，以订单 ID 作为 sharding key，那么同一个订单相关的创建订单消息、订单支付消息、订单退款消息、订单物流消息都会按照先后顺序来发布和订阅。

  阿里巴巴集团内部电商系统均使用此种分区顺序消息，既保证业务的顺序，同时又能保证业务的高性能。

顺序消息对比

在控制台创建顺序消息使用的 Topic，各种类型 Topic 对比如下。

表 85: 消息类型对比

Topic 类型	支持事务消息	支持定时消息	性能
无序消息	是	是	最高
分区顺序	否	否	高
全局顺序	否	否	一般

表 86: 发送方式对比

消息类型	支持可靠同步发送	支持可靠异步发送	支持 oneway 发送
无序消息	是	是	是
分区顺序	是	否	否
全局顺序	是	否	否

23.5 常见使用方式说明

本章介绍MQ使用过程中的最常见的使用方式及注意事项。

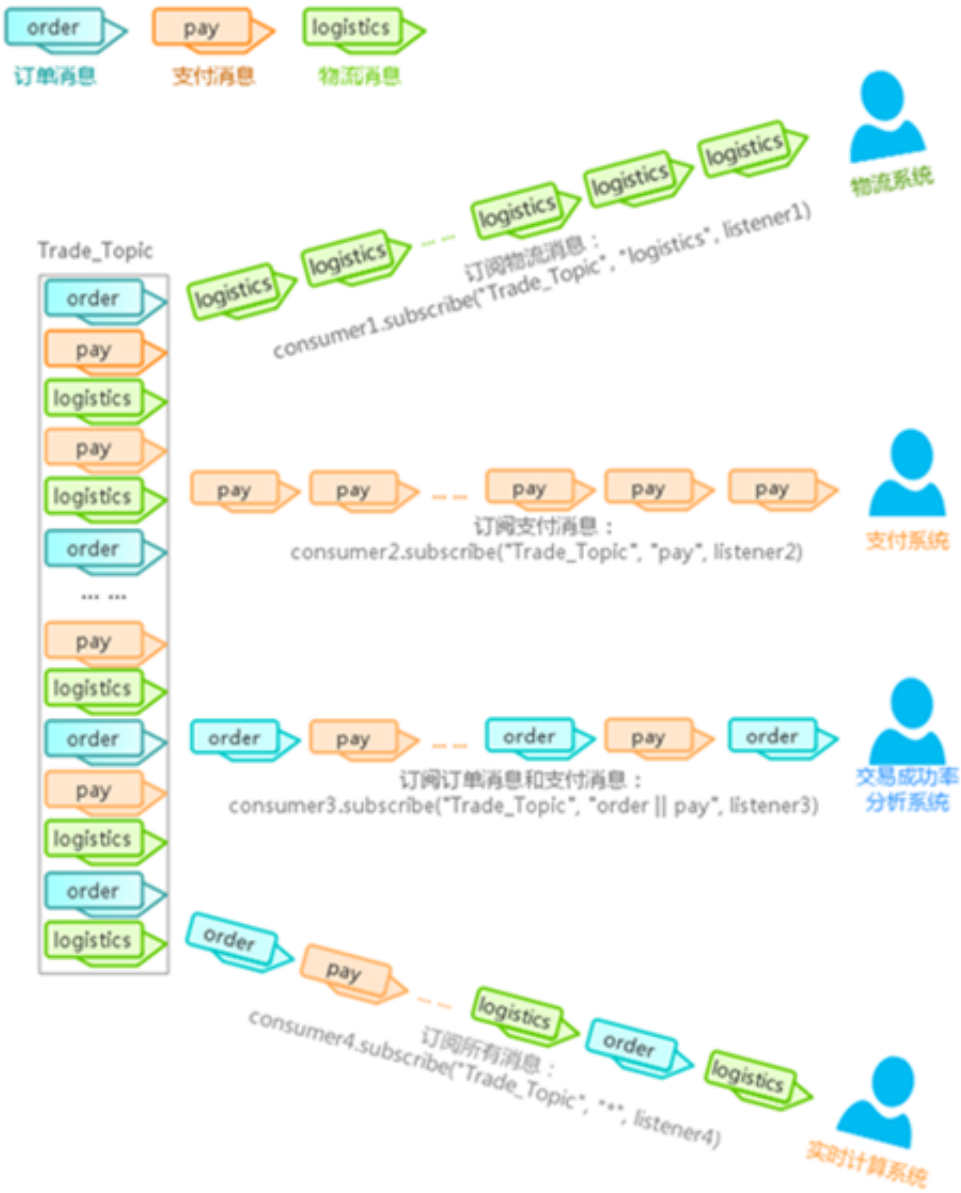
23.5.1 消息过滤

MQ提供消息过滤功能，方便MQ消费者如何根据Tag在MQ服务端完成消息过滤。

Tag，即消息标签、消息类型，用来区分某个MQ的Topic下的消息分类。MQ允许消费者按照Tag对消息进行过滤，确保消费者最终只消费到他关心的消息类型。

以下图所示电商交易场景为例，从客户下单到收到商品这一过程会生产一系列消息，比如订单创建消息（order）、支付消息（pay）、物流消息（logistics）。这些消息会发送到Topic为Trade_Topic的队列中，被各个不同的系统所接收，比如支付系统、物流系统、交易成功率分析系统、实时计算系统等。其中，物流系统只需接收物流类型的消息（logistics），而实时计算系统需要接收所有和交易相关（order、pay、logistics）的消息。

图 604: 消息过滤



说明: 针对消息归类，您可以选择创建多个Topic， 或者在同一个Topic下创建多个Tag。但通常情况下，不同的Topic之间的消息没有必然的联系，而Tag则用来区分同一个Topic下相互关联的消息，比如全集和子集的关系、流程先后的关系。

参考示例：

- 发送消息

```
Message msg = new Message("MQ_TOPIC","TagA","Hello MQ".getBytes());
```

说明: 发送消息时，每条消息必须指明消息类型Tag：

- 订阅消息
  - 消费方式-1：消费者如需订阅某Topic下所有类型的消息，Tag用 * 符号表示。

```
consumer.subscribe("MQ_TOPIC", "*", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

  - 消费方式-2：消费者如需订阅某Topic下某一种类型的消息，请明确标明Tag。

```
consumer.subscribe("MQ_TOPIC", "TagA", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

  - 消费方式-3：消费者如需订阅某Topic下多种类型的消息，请在多个Tag之间用 || 分隔。

```
consumer.subscribe("MQ_TOPIC", "TagA||TagB", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

  - 消费方式-4（错误示例）：同一个消费者多次订阅某Topic下的不同Tag，后者会覆盖前者（错误示例)。

```
//如下错误代码中，consumer只能接收到MQ_TOPIC下TagB的消息，而不能接收TagA的消息。
consumer.subscribe("MQ_TOPIC", "TagA", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
consumer.subscribe("MQ_TOPIC", "TagB", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
```

```
});
```

23.5.2 集群消费和广播消费

本节主要介绍 MQ 集群消费和广播消费的基本概念，适用场景以及注意事项。

基本概念

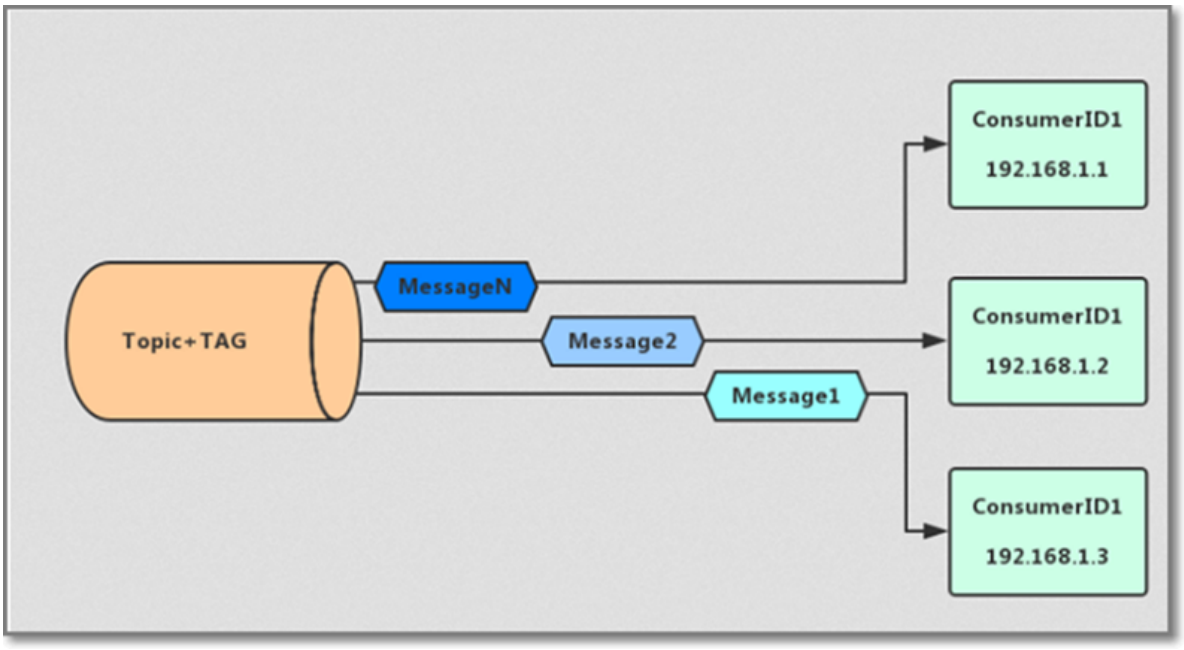
MQ 是基于发布订阅模型的消息系统。在 MQ 消息系统中消息的订阅方订阅关注的 Topic，以获取并消费消息。由于订阅方应用一般是分布式系统，以集群方式部署有多台机器。因此 MQ 约定以下概念。

- **集群**：MQ 约定使用相同 Consumer ID 的订阅者属于同一个集群，同一个集群下的订阅者消费逻辑必须完全一致（包括 Tag 的使用），这些订阅者在逻辑上可以认为是一个消费节点。
- **集群消费**：使用集群消费模式时，MQ 认为任意一条消息只需要被集群内的任意一个消费者处理即可。
- **广播消费**：使用广播消费模式时，MQ 会将每条消息推送给集群内所有注册过的客户端，保证消息至少被每台机器消费一次。

场景对比

- 集群消费模式

图 605: 集群消费模式

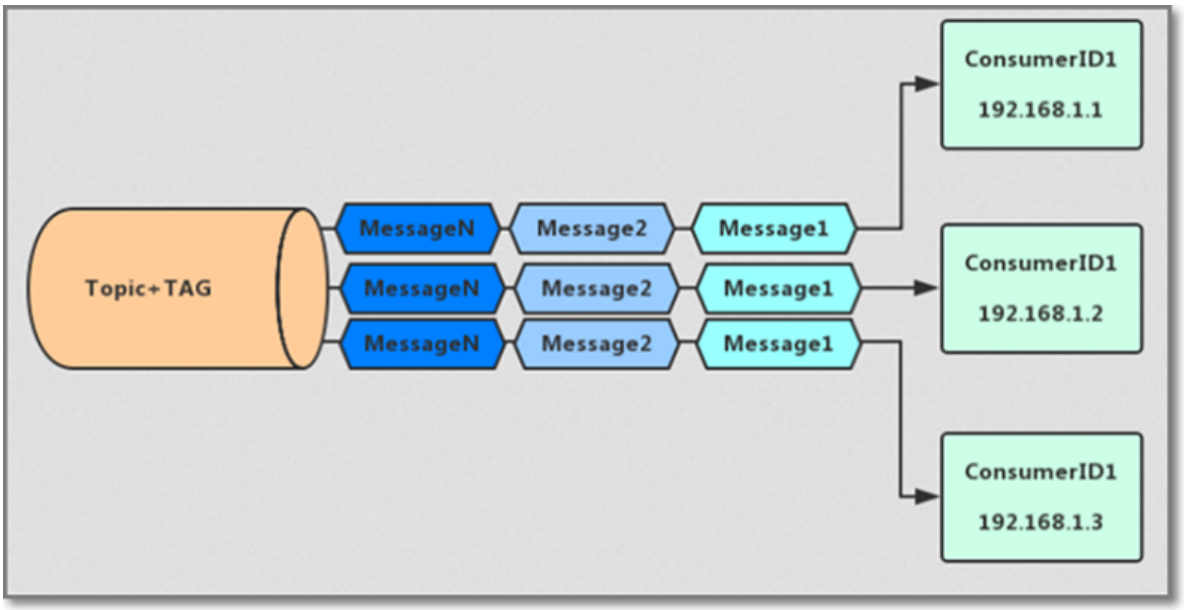


集群消费模式适用场景及注意事项如下。

- 消费端集群化部署，每条消息只需要被处理一次。
- 由于消费进度在服务端维护，可靠性更高。
- 集群消费模式下，每一条消息都只会被分发到一台机器上处理，如果需要被集群下的每一台机器都处理，请使用广播模式。
- 集群消费模式下，不保证消息的每一次失败重投等逻辑都能路由到同一台机器上，因此处理消息时不应该做任何确定性假设。

- 广播消费模式

图 606: 广播消费模式

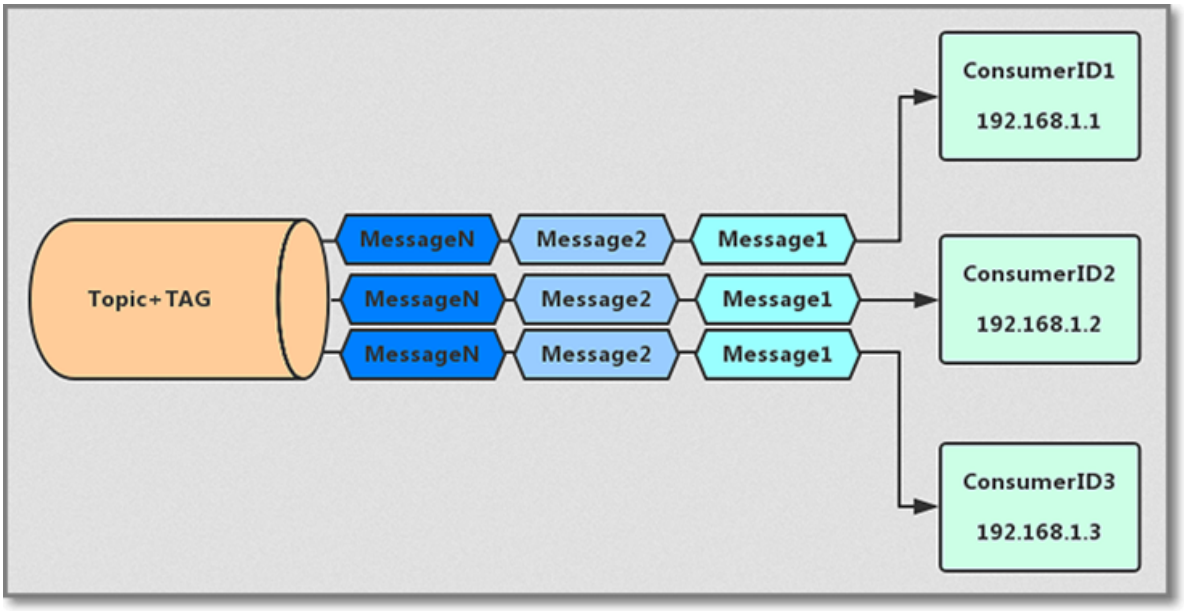


广播消费模式适用场景及注意事项如下。

- 每条消息都需要被相同逻辑的多台机器处理。
- 消费进度在客户端维护，出现重复的概率稍大于集群模式。
- 广播模式下，MQ 保证每条消息至少被每台客户端消费一次，但是并不会对消费失败的消息进行失败重投，因此业务方需要关注消费失败的情况。
- 广播模式下，第一次启动时默认从最新消息消费，客户端的消费进度是被持久化在客户端本地的隐藏文件中，因此不建议删除该隐藏文件，否则会丢失部分消息。
- 广播模式下，每条消息都会被大量的客户端重复处理，因此推荐尽可能使用集群模式。
- 目前仅 Java 客户端支持广播模式。
- 广播模式下服务端不维护消费进度，所以服务端不提供堆积查询和报警功能。

- 适用集群模式模拟广播模式

图 607: 集群方式模拟广播模式



此方式的使用场景及注意事项如下。

- 每条消息都需要被多台机器处理，每台机器的逻辑可以相同也可以不一样。
- 消费进度在服务端维护，可靠性高于广播模式。

23.5.3 订阅关系一致

MQ 里的一个 Consumer ID 代表一个 Consumer 实例群组。对于大多数分布式应用来说，一个 Consumer ID 下通常会挂载多个 Consumer 实例。订阅关系一致指的是同一个 Consumer ID 下所有 Consumer 实例的处理逻辑必须完全一致。一旦订阅关系不一致，消息消费的逻辑就会混乱，甚至导致消息丢失。

由于 MQ 的订阅关系主要由 Topic+Tag 共同组成，因此，保持订阅关系一致意味着同一个 Consumer ID 下所有的实例需在以下两方面均保持一致：

- 订阅的 Topic 必须一致。
- 订阅的 Topic 中的 Tag 必须一致。

下面举几个反面例子：

【例一】以下例子中，同一个 Consumer ID 下的两个实例订阅的 Topic 不一致。

Consumer 实例 1-1：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.ConsumerId, "CID_jodie_test_1");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("jodie_test_A", "*", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

Consumer 实例1-2：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.ConsumerId, " CID_jodie_test_1");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("jodie_test_B ", "*", new MessageListener() {
public Action consume(Message message, ConsumeContext context) {
System.out.println(message.getMsgID());
return Action.CommitMessage;
}
});
```

【例二】以下例子中，同一个 Consumer ID 下订阅 Topic 的 Tag 不一致。Consumer 实例2-1 订阅了 TagA，而 Consumer 实例2-2 未指定 Tag。

Consumer 实例2-1：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.ConsumerId, "CID_jodie_test_2");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("jodie_test_A", "TagA", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

Consumer 实例2-2：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.ConsumerId, " CID_jodie_test_2");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("jodie_test_A ", "*", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

【例三】此例中，错误的原因有两个：

- 同一个 Consumer ID 下订阅 Topic 个数不一致。
- 同一个 Consumer ID 下订阅 Topic 的 Tag 不一致。

Consumer 实例3-1：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.ConsumerId, "CID_jodie_test_3");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("jodie_test_A", "TagA", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
consumer.subscribe("jodie_test_B", "TagB", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

Consumer 实例3-2：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.ConsumerId, " CID_jodie_test_3");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("jodie_test_A ", "TagB", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

23.5.4 消息重试

本节介绍 MQ 消费者的消费逻辑失败时，通过设置返回状态可达到消息重试的结果。

说明： MQ 消息重试只针对集群消费方式生效；广播方式不提供失败重试特性，即消费失败后，失败消息不再重试，继续消费新的消息。

23.5.4.1 重试次数

本文介绍消息发送失败后的重试次数及时间间隔。

MQ 默认允许每条消息最多重试 16 次，每次重试的间隔时间如下：

第几次重试	每次重试间隔时间	第几次重试	每次重试间隔时间
1	10 秒	9	7 分钟
2	30 秒	10	8 分钟
3	1 分钟	11	9 分钟
4	2 分钟	12	10 分钟
5	3 分钟	13	20 分钟
6	4 分钟	14	30 分钟
7	5 分钟	15	1 小时

第几次重试	每次重试间隔时间	第几次重试	每次重试间隔时间
8	6 分钟	16	2 小时

如果消息重试16次后仍然失败，消息将不再投递。如果严格按照上述重试时间间隔计算，某条消息在一直消费失败的前提下，将会在接下来的4小时46分钟之内进行16次重试，超过这个时间范围消息将不再重试投递。

23.5.4.2 配置方式

本文介绍消息发送失败后，重试的配置方式及示例代码。

消费失败后，重试配置方式

集群消费方式下，消息消费失败后期望消息重试，需要在消息监听器接口的实现中明确进行配置（三种方式任选一种）：

- 返回 Action.ReconsumeLater （推荐）
- 返回 Null
- 抛出异常

代码示例如下：

```
public class MessageListenerImpl implements MessageListener {

    @Override
    public Action consume(Message message, ConsumeContext context) {
        //方法3：消息处理逻辑抛出异常，消息将重试
        doConsumeMessage(message);
        //方式1：返回 Action.ReconsumeLater, 消息将重试
        return Action.ReconsumeLater;
        //方式2：返回 null, 消息将重试
        return null;
        //方式3：直接抛出异常，消息将重试
        throw new RuntimeException("Consumer Message exceotion");
    }
}
```

消费失败后，不重试配置方式

集群消费方式下，消息失败后期望消息不重试，需要捕获消费逻辑中可能抛出的异常，最终返回 Action.CommitMessage，此后这条消息将不会再重试。代码示例如下：

```
public class MessageListenerImpl implements MessageListener {

    @Override
    public Action consume(Message message, ConsumeContext context) {
        try {
            doConsumeMessage(message);
        } catch (Throwable e) {
```

```
            //捕获消费逻辑中的所有异常，并返回 Action.CommitMessage;
            return Action.CommitMessage;
        }
        //消息处理正常，直接返回 Action.CommitMessage;
        return Action.CommitMessage;
    }
}
```

自定义消息最大重试次数

自定义 MQ 客户端日志配置，请升级 TCP Java SDK 版本到1.2.2及以上。

MQ 允许 Consumer 启动的时候设置最大重试次数，重试时间间隔将按照如下策略：

- 最大重试次数小于等于16次，则重试时间间隔同上表描述。
- 最大重试次数大于16次，超过16次的重试时间间隔均为每次2小时。

配置方式如下：

```
Properties properties = new Properties();
//配置对应 Consumer ID 的最大消息重试次数为20 次
properties.put(PropertyKeyConst.MaxReconsumeTimes,20);
Consumer consumer =ONSFactory.createConsumer(properties);
```



说明:

- 消息最大重试次数的设置对相同 Consumer ID 下的所有 Consumer 实例有效。
- 如果只对配置了相同 Consumer ID 的两个 Consumer 实例中的其中一个设置了 MaxReconsumeTimes，那么该配置对两个 Consumer 实例均生效。
- 配置采用覆盖的方式生效，即最后启动的 Consumer 实例会覆盖之前的启动实例的配置。

获取消息重试次数

消费者收到消息后，可按照如下方式获取消息的重试次数：

```
public class MessageListenerImpl implements MessageListener {

    @Override
    public Action consume(Message message, ConsumeContext context) {
        //获取消息的重试次数
        System.out.println(message.getReconsumeTimes());
        return Action.CommitMessage;
    }
}
```

23.5.5 消费幂等

本文强调 MQ 消费者在接收到消息以后根据业务上的唯一 Key 对消息做幂等处理的必要性。

消费幂等的必要性

在互联网应用中，尤其在网络不稳定的情况下，MQ 的消息有可能会出现重复，这个重复简单可以概括为以下两种情况：

- 发送时消息重复【消息 Message ID 不同】：

MQ Producer 发送消息场景下，消息已成功发送到服务端并完成持久化，此时网络闪断或者客户端宕机导致服务端应答给客户端失败；如果此时 MQ Producer 意识到消息发送失败并尝试再次发送消息，MQ 消费者后续会收到两条内容相同但是 Message ID 不同的消息。

- 投递时消息重复【消息 Message ID 相同】。

MQ Consumer 消费消息场景下，消息已投递到消费者并完成业务处理，当客户端给服务端反馈应答的时候网络闪断，为了保证消息至少被消费一次，MQ 服务端将在网络恢复后再次尝试投递之前已被处理过的消息，MQ 消费者后续会收到两条内容相同并且 Message ID 也相同的消息。

处理建议

基于上述第一种原因，内容相同的消息 Message ID 可能会不同，真正安全的幂等处理，不建议以 Message ID 作为处理依据；最好的方式是以业务唯一标识作为幂等处理的关键依据，而业务的唯一标识可以通过 Key 设置到消息里面：

```
Message message = new Message();
message.setKey("ORDERID_100");
SendResult sendResult = producer.send(message);
```

订阅方收到消息时可以根据消息的 key 进行幂等处理：

```
consumer.subscribe("ons_test", "*", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        String key = message.getKey()
        // 根据业务唯一标识的 key 做幂等处理
    }
});
```

23.6 使用消息队列控制台

本章节介绍MQ控制台的基本功能及使用方式。

MQ控制台主要提供以下几大功能：

- 发布、订阅管理

- 消息查询
- MQTT管理
- 资源报表
- 消息轨迹
- 报警监控

23.6.1 查询消息

MQ 控制台提供几种消息查询方式用于问题排查。

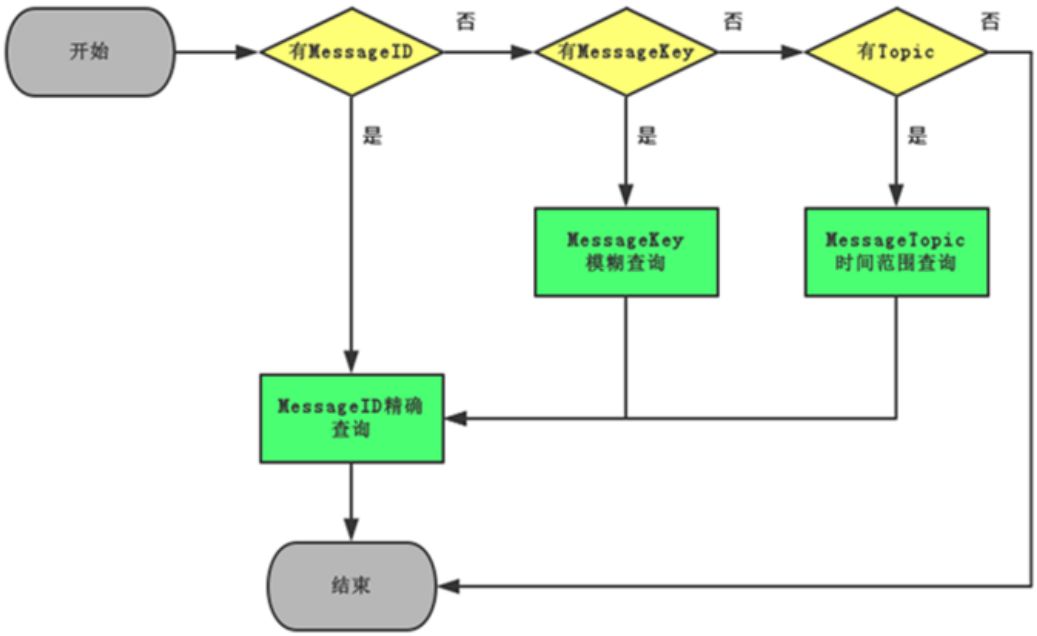
MQ 提供了三种消息查询的方式，分别是根据 Messag ID，Message Key 以及 Topic 查询。三种方式的特点和对比如下。

表 87: 查询方式对比

查询方式	查询条件	查询类别	说明
Message ID 查询	Topic +Message ID	精确查询	根据 Message ID 可以精确定位任意一条消息，获取消息的属性。
Message Key 查询	Topic +Message Key	模糊查询	根据 Message Key 可以匹配到包含指定 Key 的最近 64 条消息，建议消息生产方为每条消息设置尽可能唯一的 Key，以确保相同的 Key 的消息不会超过 64 条，否则消息会漏查。
Topic 查询	Topic+时间段	范围查询	根据 Topic 和时间范围，批量获取符合条件的所有消息，查询量大，不易匹配。

建议查询过程如下：

图 608: 消息查询流程



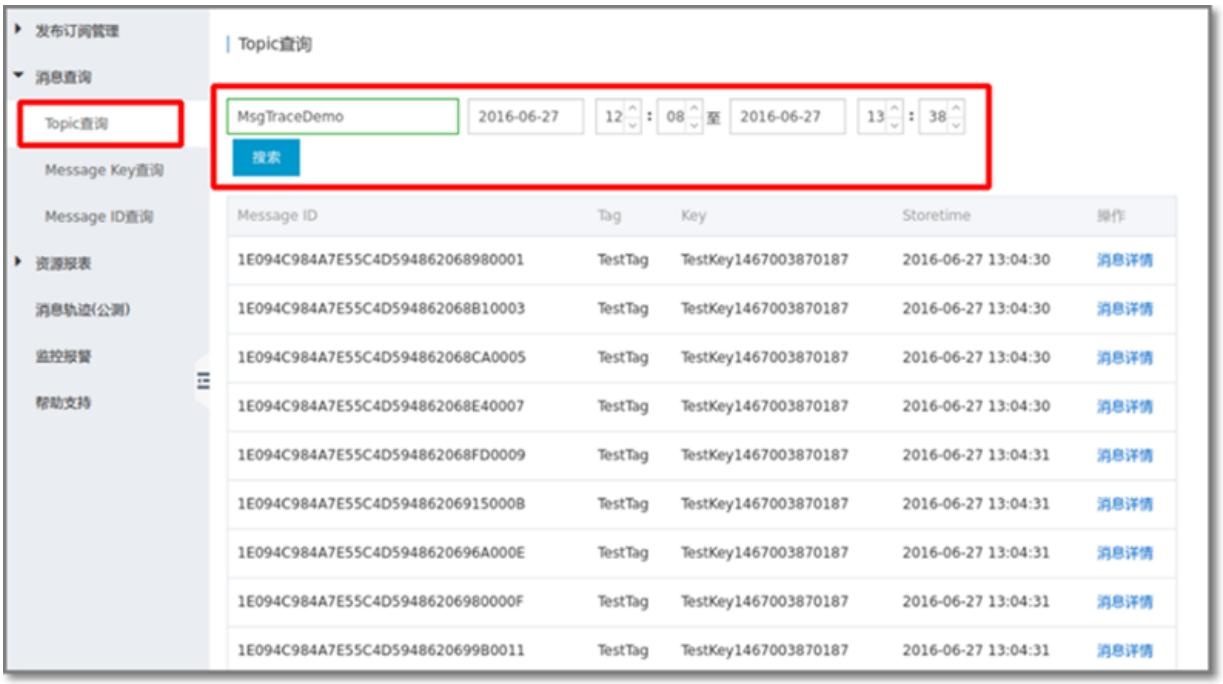
23.6.1.1 根据Topic查询

根据 Topic 查询一般用在前两种查询条件都无法获得的情况下，根据 Topic 和消息的发送时间范围，批量获取该时间范围内的所有消息，然后再找到关心的数据。

Topic 查询属于范围查询，获取 Topic 下符合时间条件的所有消息，消息量大，建议尽量缩短查询区间。Topic 查询数据量大，采用分页展示。

查询样例如下图：

图 609: 查询样例



23.6.1.2 根据Message Key查询

Message Key 查询消息的原理是，MQ 根据设置的 Message Key 信息建立消息的索引信息，当您输入 Key 进行查询时，根据该索引即可匹配相关的消息返回。

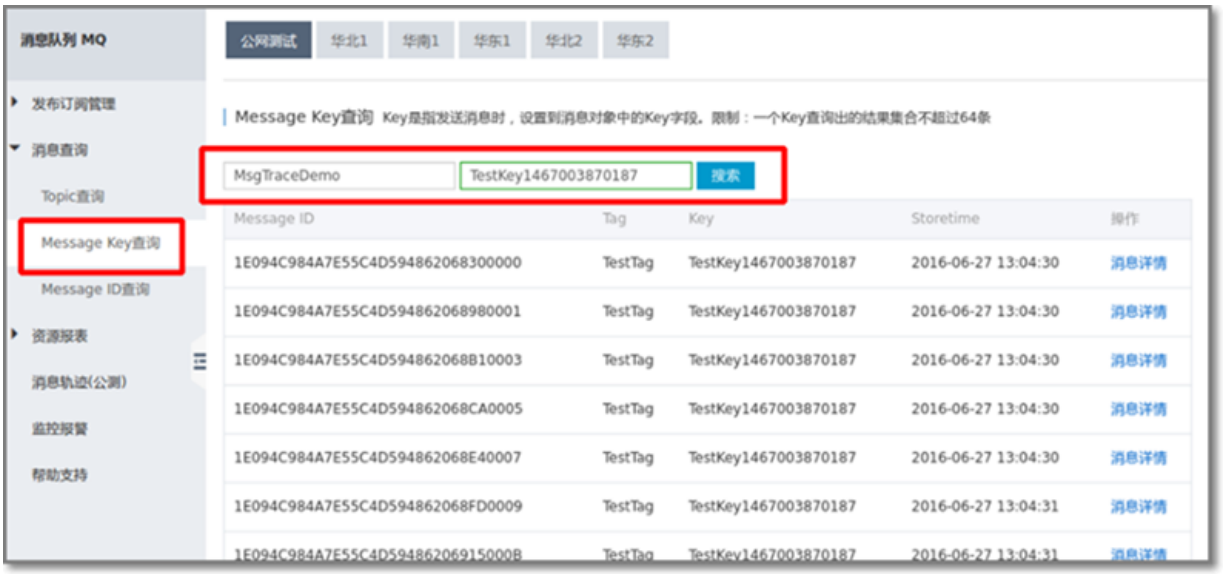
Message Key 查询条件是您设置的 Message Key 属性。Message Key 查询仅仅返回符合条件的最近的 64 条消息，因此建议业务方设置 Key 时尽可能唯一，并具有业务区分度。

设置Message Key的方法如下：

```
Message msg = new Message("Topic","*","Hello MQ".getBytes());
/**
* 对每条消息设置其检索的Key，该Key值设置代表消息的业务关键属性，请尽可能全局唯一。
* 以方便您在无法正常收到消息情况下，可通过MQ控制台查询消息。不设置也不会影响消息正常收发
*/
msg.setKey("TestKey"+System.currentTimeMillis());
```

查询样例如下图：

图 610: 查询样例



23.6.1.3 根据Message ID查询

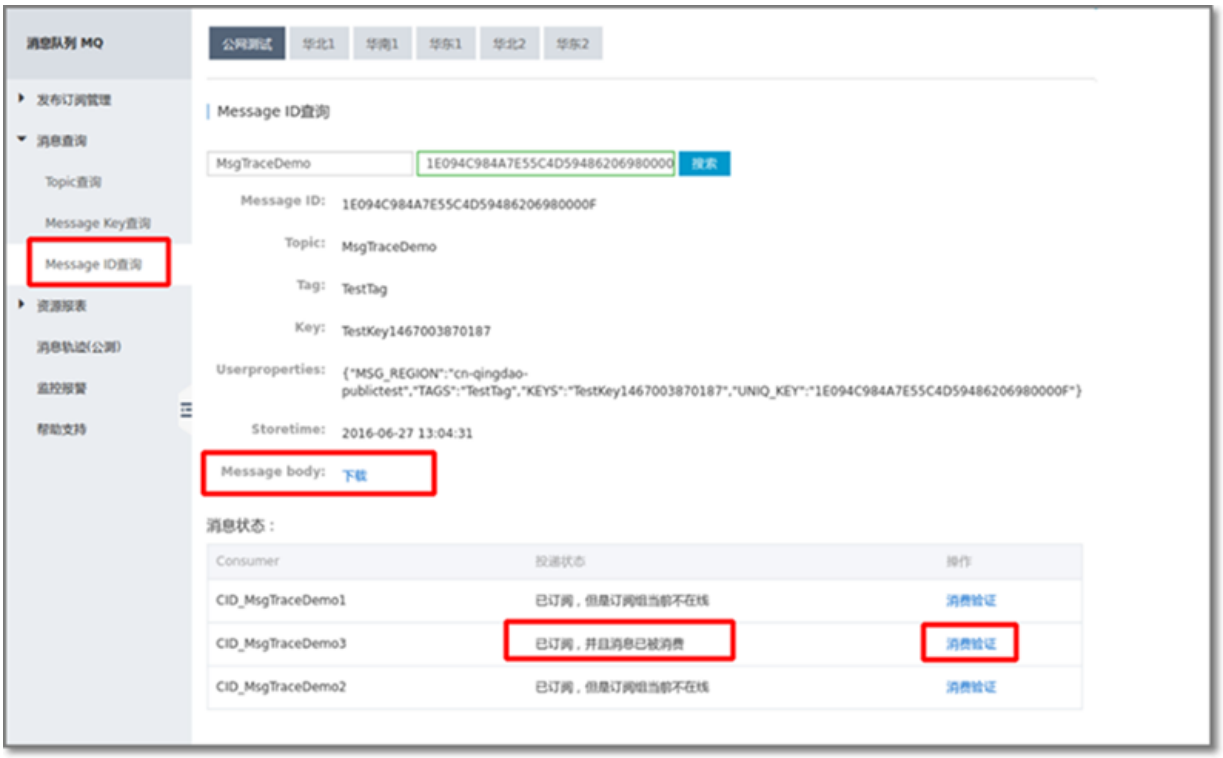
根据 Message ID 查询消息属于精确查询，输入 Message ID 信息即可精确查询到任意一条消息。因此，为了尽可能精确地查询，建议发送方调用 Producer 发送消息时，在发送消息成功后将 Message ID 信息打印到日志中，方便问题排查。

以 Java SDK 为例，获取 Message ID 的方法如下：

```
SendResult sendResult = producer.send(msg);
String msgId = sendResult.getMessageId();
```

查询到消息后，您可以在消息详情页面看到每条消息的基本属性，消息体下载链接以及消息的消费状态，如下图所示：

图 611: 查询结果



其中，投递状态是 MQ 根据各个 Consumer ID 的消费进度计算出的结果，投递状态的信息参考以下表格。

注意:

投递状态仅仅是根据消费进度估算的结果，如果需要详细的消费信息，请使用消息轨迹工具查询。消息轨迹可以展示该条消息的完整链路，具体请参见消息轨迹使用指南。

表 88: 消息投递状态说明

投递状态	可能的原因
已订阅，并且消息已被消费	该订阅组已经正常消费过这条消息
已订阅，但消息被过滤表达式过滤，请查看 Tag	该消息的 Tag 不符合消费方的订阅关系，消息被过滤，可以在控制台>订阅管理>消费者状态查询菜单查询订阅方的订阅关系
已订阅，但消息未被消费	该订阅组订阅了该消息，但还未消费，有可能是消费过慢，或者消费出现异常导致阻塞
已订阅，但是订阅组当前不在线	该订阅组订阅了该消息，但是不在线，请检查消费者端应用不在线的原因

投递状态	可能的原因
未知异常	出现未收录异常，可以进工单系统提工单解决

MQ提供了消费验证功能，该功能可以将指定消息推送给指定的在线客户端，以检测客户端消费该消息的逻辑和结果是否符合预期。

注意： 消费验证功能仅仅是用于验证客户端的消费逻辑是否正常，并不会影响正常的收消息流程，因此消息的消费状态等信息在单击消费验证后并不会改变。

23.6.2 查询资源报表

资源报表是 MQ 提供的消息发送和消费的统计数据查询功能。通过资源报表可以对某个 Topic 在一段时间范围内的发送总量或者 TPS 进行查询，也可以对一个 Consumer ID 消费某个 Topic 的总量或者 TPS 进行查询。

本节主要介绍如何对消息生产和消费信息进行查询。

23.6.2.1 查询消费消息

消息消费查询用于对一个 Consumer ID 消费某个 Topic 的总量或者 TPS 进行查询。

操作步骤

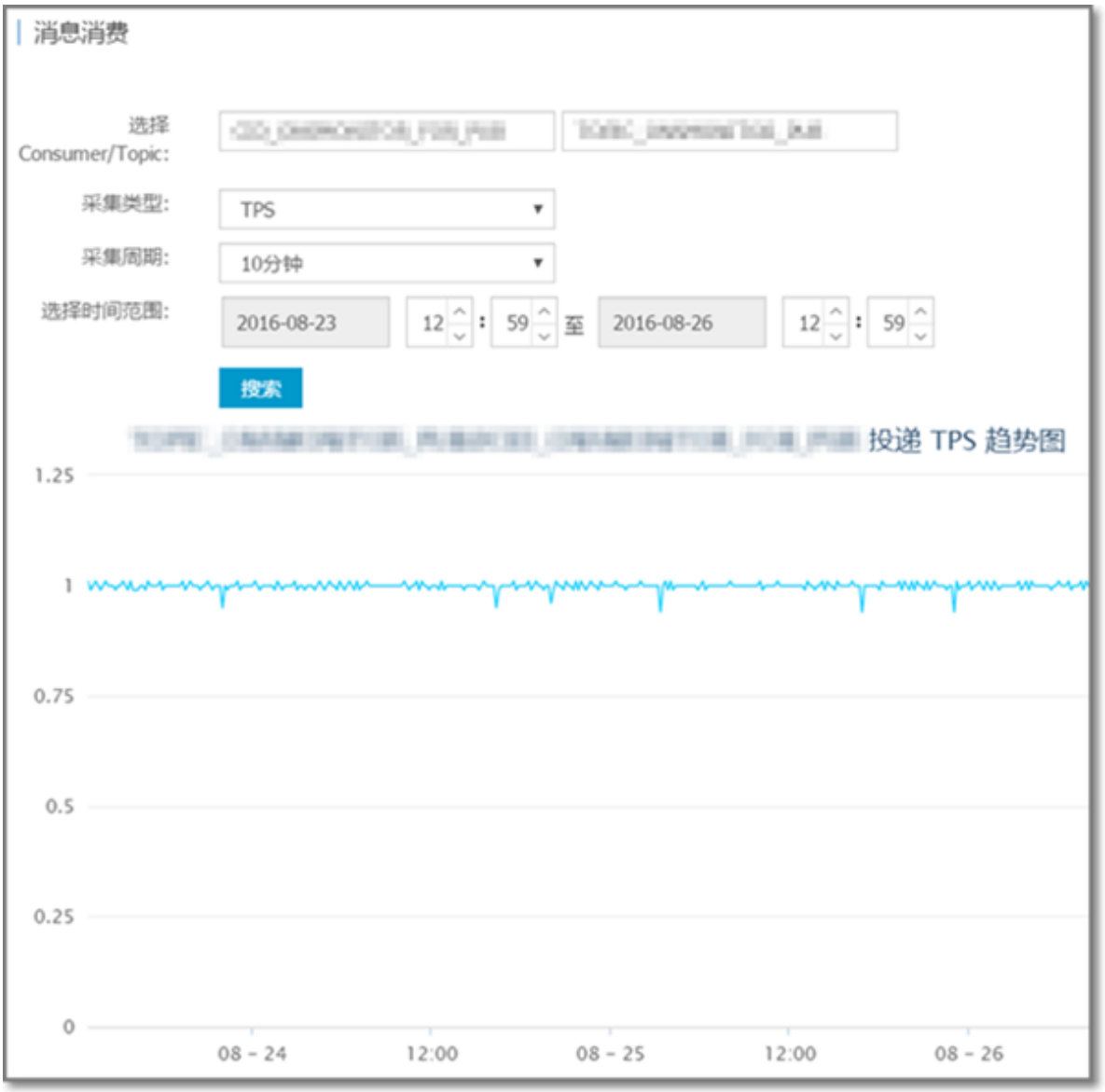
1. 在 MQ 控制台左侧菜单栏选择**资源报表** > **消息消费**。
2. 选择您要查询的 Consumer ID 所在的域。
3. 在消息消费页面输入相应信息，单击**搜索**。

   选项说明：

   - **采集类型**：分为TPS 和总量，总量提供该周期内消费该 Topic 数据总量，TPS 提供消费该 Topic 的平均 TPS。
   - **采集周期**：包括 1 分钟、10 分钟后、30 分钟、1 小时，采集周期决定了数据采集的时间间隔，周期越短，采集点越密集，消息消费数据越详细。
   - **时间范围**：MQ 最多可以提供最近三天之内的消息消费查询。

   查询结果以图表的形式返回。

   图 612: 消费查询



23.6.2.2 查询生产消息

消息生产查询用于对某个 Topic 在一个时间段内的消息发送总量或者 TPS 进行查询。

操作步骤

1. 在 MQ 控制台左侧菜单栏选择**资源报表** > **消息生产**。
2. 选择您要查询的 Topic 所在的域。
3. 在消息消费页面输入相应信息，单击搜索。

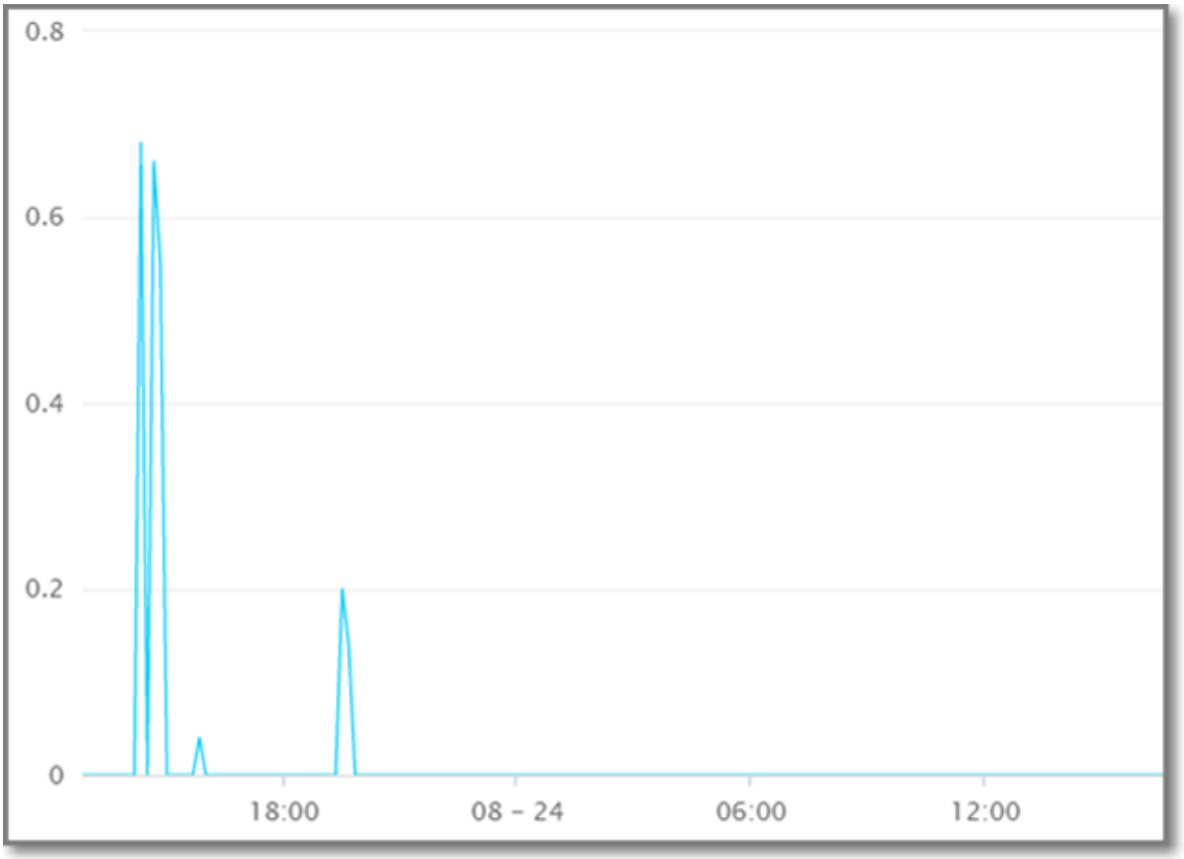
   选项说明：

   - **采集类型**：分为 TPS 和总量，总量提供该周期内 Topic 的发送总量，TPS 提供发送平均 TPS。

- **采集周期**：包括 1 分钟、10 分钟后、30 分钟、1 小时，采集周期决定了数据采集的时间间隔，周期越短，采集点越密集，消息消费数据越详细。
- **时间范围**：MQ 最多可以提供最近三天之内的消息消费查询。

查询结果以图表的形式返回。

图 613: 查询结果



23.6.3 查询消息轨迹

23.6.3.1 消息轨迹简介

本节介绍 MQ 消息轨迹的基本原理、使用场景、以及使用案例。

基本原理

定义：消息轨迹指的是一条消息从生产方发出到消费方消费处理，整个过程中的各个相关节点的时间地点等数据汇聚而成的完整链路信息。

原理：MQ 系统中，一条消息的完整链路包含生产方、服务方、消费方三个角色，每个角色处理消息的过程中都会在轨迹链路中增加相关的信息，将这些信息汇聚即可获取任意消息当前的状态，从而为生产环境中的问题排查提供强有力的数据支持。

消息轨迹的数据包含：

表 89: 消息轨迹数据

生产方信息	消费方信息	服务方信息
生产客户端信息	消费客户端信息	消息存储位置信息
发送时间	投递时间，投递轮次	消息存储时间
发送成功与否	消费成功与否	消息本身的属性
发送耗时	消费耗时	-

使用场景

在生产环境的消息收发不符合预期时可以使用消息轨迹工具排查问题。通过消息的属性（Message ID、Message Key、Topic）搜索相关的消息轨迹，找到消息的实际收发状态，帮助诊断问题。

图 614: 消息轨迹



案例

业务上有一条消息一直没有收到怎么办？

业务方如果根据业务日志里的信息判断某条消息一直没有收到，此时可以使用消息轨迹工具来确认该情况。

1. 收集怀疑的消息的信息，Message ID，Message Key，Topic 以及大概的发送时间范围。
2. 进入 MQ 控制台，根据已有的信息新建查询任务，查询相关的消息的轨迹。

3. 查看结果。并分析判断原因，如果轨迹显示尚未消费，则可以去订阅管理页面查询，确认是否有堆积导致消息尚未消费。

如果发现已经消费，请根据消费端的信息，找到对应的客户端机器和时间，登录查看相关日志。

23.6.3.2 消息轨迹查询

消息轨迹的使用对于业务方不会增加额外的接入成本，仅仅需要确保客户端 SDK 版本支持该特性。正常收发消息后以消息的相关属性在 MQ 控制台上查询即可。

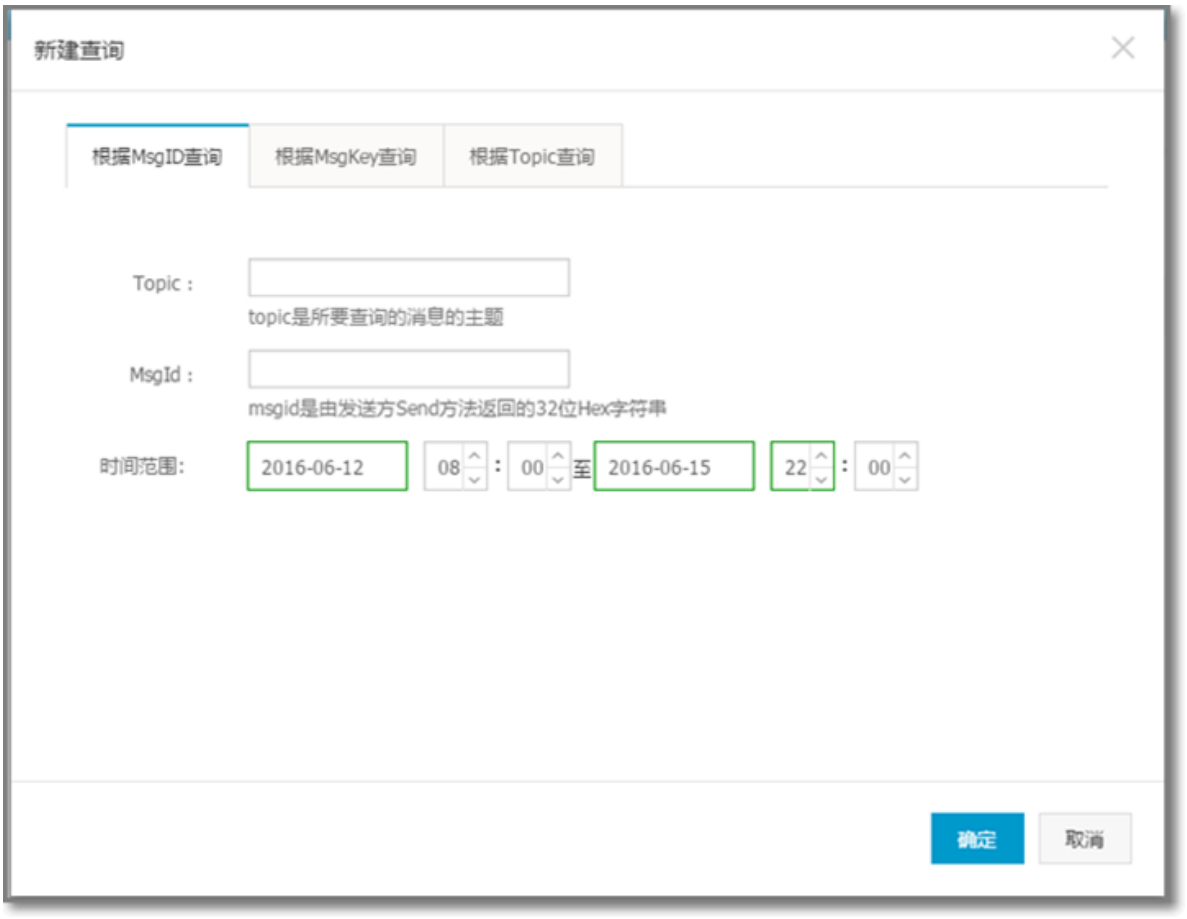
前提条件

要使用消息轨迹功能，必须确保 Java 客户端版本至少是1.2.2。如果版本太低，请首先升级客户端。详细的客户端信息，请参考控制台给出的升级提示，获取最新的 SDK。

操作步骤

1. 在 MQ 控制台左侧菜单栏选择**消息轨迹**（现处于公测阶段），选择所在的区域，并单击右上角**新建查询**按钮。

图 615: 新建查询



消息轨迹查询功能支持三种查询方式，请按照对应方式输入查询条件，新建查询。

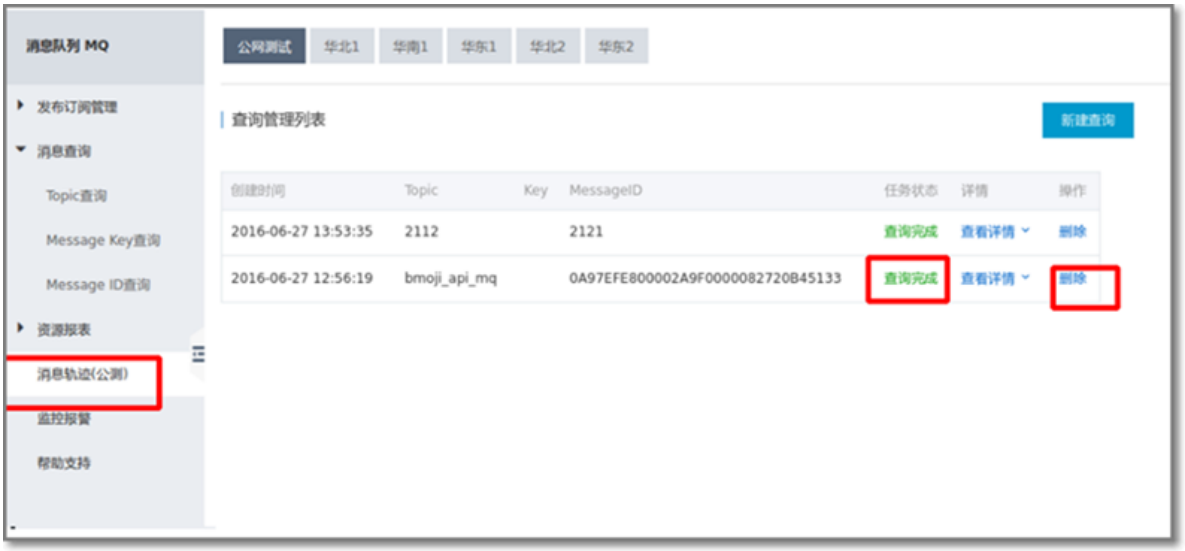
- 根据 Message ID 查询：需要输入消息的唯一 Message ID，Topic 名称以及消息的大致发送时间。
- 根据 Message Key 查询：需要输入消息的 Message Key 和 Topic 以及大致发送时间，适用于没有记录 Message ID，但记录了 Message Key 的场景。
- 根据 Topic 查询：仅仅输入 Topic 和时间段，批量查询，适用于没有上述 Message ID 和 Message Key，而且消息量比较小的场景。

注意:

- 查询时，尽可能设置最为精确的时间区间，以便缩小查询范围，提高速度。
- 根据 MsgID 查询属于精确查询，速度快，精确匹配，推荐用户使用。
- 根据 MsgKey 查询属于模糊查询，仅适用于业务方没有记录 Message ID 但是设置了 Message Key，同时 Message Key 具有区分度的情况，Message Key 查询最多查询 1000 条轨迹。
- 根据 Topic 分段查询属于范围查询，不推荐使用，因为时间范围内消息很多，不具备区分度。

新建查询后，会生成一个查询任务，MQ 后台会异步执行，并将任务状态反馈到管理页面，查询结束时，任务状态显示查询完成，否则显示查询中。

图 616: 查询任务

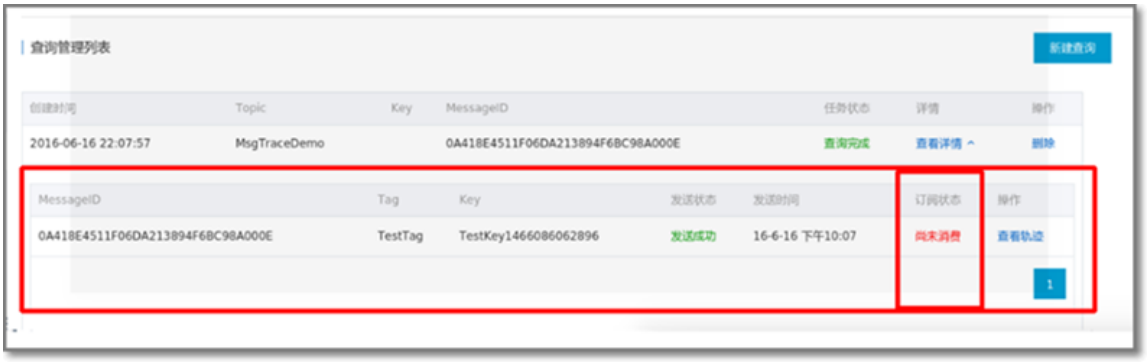


根据任务的状态可以选择查看轨迹，或者删除查询任务。

2. 完成查询后，单击右侧操作选项里的**查看详情**按钮查看轨迹。

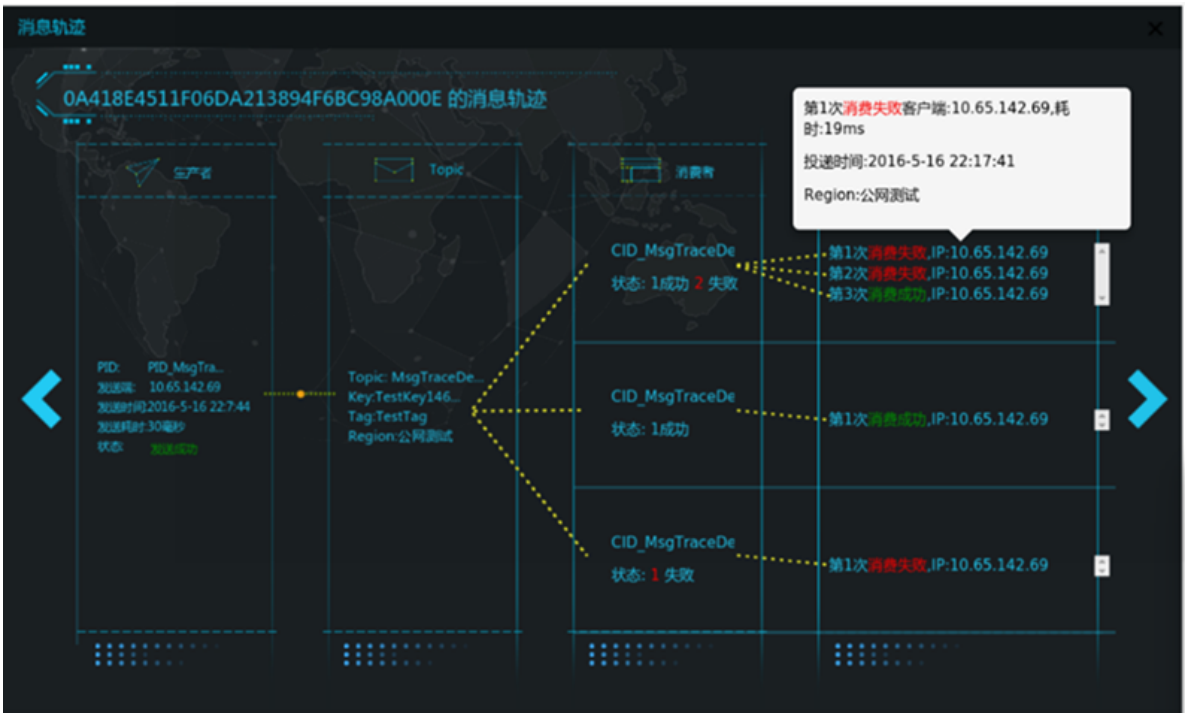
   - 如果发现没有结果，请参考弹窗链接，排查原因。
   - 如果查询到轨迹信息，可以看到轨迹的简要信息，主要是消息本身的属性以及接收状态的统计，如下图所示：

   图 617: 消息轨迹信息



3. 单击**查看轨迹**按钮即可查看完整的链路图，如图所示：

   图 618: 消息链路图



消息链路图包含4个部分：

- 生产者信息

- Topic 信息
- 消费者信息
- 详情信息

各个字段区域均可以通过鼠标悬停的方式获取详细信息。

对于 MsgKey 和 Topic 查询方式，如果匹配到多条轨迹，可以进行上下翻页，查看比对轨迹数据。

如果对消息轨迹的查询结果有疑问，也可以参考相关的消息轨迹。

23.6.3.3 消息轨迹状态说明

消息轨迹查询页面中涉及到的名词概念列表如下。

表 90: 消息轨迹名词说明

相关概念	含义
发送成功	消息发送成功
发送失败	消息发送失败
消息定时中	该消息是定时或者延时消息，且尚未到达投递时间
事务未提交	该消息是事务消息，且尚未提交状态
事务回滚	该消息是事务消息，并且已经回滚
全部成功	该消息所有投递都已成功消费
部分成功	该消息投递中存在消费失败并重试成功的情况
尚未消费	该消息尚未投递给任何消费方
发送时间	记录消息从发送端发送时的客户端时间戳
发送耗时	记录发送端调用 send 方法发送消息的毫秒耗时
Region	记录消息存储的 Region 信息，或者消费方机器所在的 Region 信息
消费耗时	记录消息推送到客户端之后执行 consumeMessage 方法的耗时
投递时间	记录客户端执行 consumeMessage 方法开始消费消息时的时间戳

23.6.4 监控报警

MQ 可以对您所订阅 Topic 的消费状态进行监控并且发送短信报警的功能，帮助您实时掌握消息消费状态并及时处理消费异常等情况。

背景信息

使用监控报警功能，首先需要在 MQ 上通过 Consumer ID 对 Topic 进行订阅，然后对相关监控项进行设置。当消费者在线，并且消息堆积数或者延迟时间超过阈值之后，MQ 会对报警接收人发送短信进行报警。

通过新建监控项，可以对一个 Consumer ID 的消费状态进行监控。具体操作如下：

操作步骤

1. 登录 MQ 控制台，在左侧菜单栏选择**监控报警**。
2. 选择您要的 Topic 所在的域。
3. 在监控项管理页面，单击右上角**新增监控项**。
4. 在**新增监控项**对话框输入相关信息，单击**确定**。新建的监控项会出现在**监控项管理**页面。

   选项说明：

   - **Consumer ID**：选择要监控的 Consumer ID。
   - **Topic**：该 Consumer ID 对应的 Topic。
   - **报警阈值**：是指消费堆积的阈值，堆积报警阈值的范围是 1 到 100，000，000。如果您选择的 Consumer ID 在消费对应的 Topic 时产生了消息堆积，并且堆积超过了报警阈值，MQ就会给报警接收人发送短信通知。
   - **消费延迟阈值**：是指该 Consumer ID 最后一次消费该 Topic 消息的时间和该 Topic 最近发送时间之间的差值，消费延迟阈值的最小值是 1 分钟。
   - **报警时间**：范围精确到分钟，最大范围为 00:00-23:59，您只会在设置的接收时间范围内才会收到报警短信。
   - **报警频率**：根据需要选择 5 分钟、15 分钟或者 30 分钟。
   - **报警接收人**：填写接收人的昵称和手机号码，昵称范围为 32 个字。

   在监控项管理页面，您可以对已经建好的监控项进行编辑、禁用、删除等操作。对于禁用的监控项，也可以在监控项管理页面单击启用重新启用。

23.7 TCP接入

本节介绍如何通过TCP方式接入使用MQ。

23.7.1 使用Java SDK 接入

介绍如何使用Java SDK进行TCP接入。

23.7.1.1 Java SDK使用说明

消息队列（MQ）提供 Java SDK 实现消息发布与消息订阅。本文将详细介绍各接口的相关参数以及各接口的使用说明。

表 91: 通用参数说明

参数名	参数说明
ONSAddr	设置 MQ TCP 协议接入点，参考交付文档（推荐）
NAMESRV_ADDR	设置 Name Server 列表（不推荐），与 ONSAddr 二选一
AccessKey	您在阿里云账号管理控制台中创建的 AccessKey，用于身份认证
SecretKey	您在阿里云账号管理控制台中创建的 SecretKey，用于身份认证
OnsChannel	用户渠道，默认为：ALIYUN

表 92: 发送消息参数说明

参数名	参数说明
ProducerId	您在控制台创建的 Producer ID
SendMsgTimeoutMillis	设置消息发送的超时时间，单位（毫秒），默认：3000
CheckImmunityTimeInSeconds（事务消息）	设置事务消息第一次回查的最快时间，单位（秒）
shardingKey（顺序消息）	顺序消息中用来计算不同分区的值

图 619: Java SDK发布消息接口及参数说明

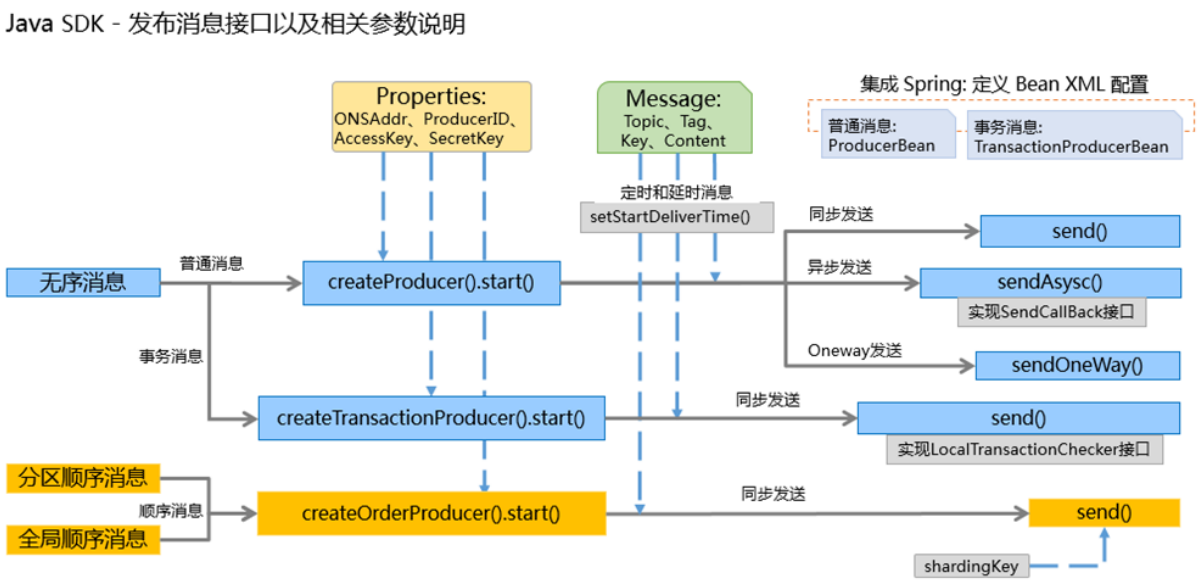
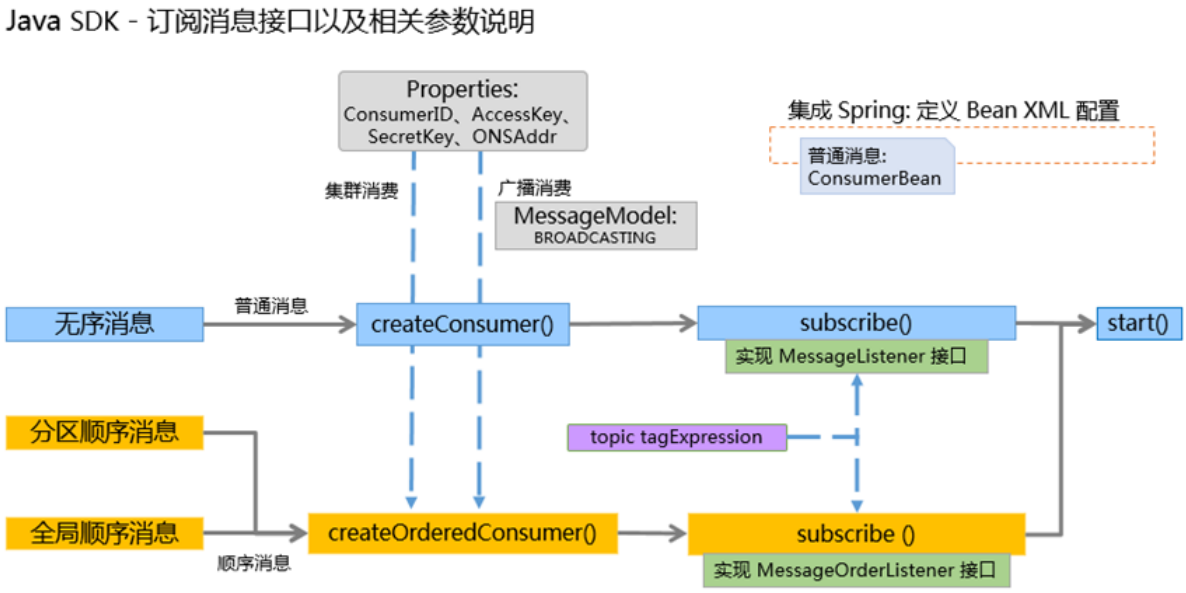


表 93: 订阅消息参数说明

参数名	参数说明
ConsumerId	您在 MQ 控制台上申请的 Consumer ID
MessageModel	设置 Consumer 实例的消费模式，默认为集群消费（值：CLUSTERING）；广播消费（BROADCASTING）
ConsumeThreadNums	设置 Consumer 实例的消费线程数，默认：64
MaxReconsumeTimes	设置消息消费失败的最大重试次数，默认：16
ConsumeTimeout	设置每条消息消费的最大超时时间，超过设置时间则被视为消费失败，等下次重新投递再次消费。每个业务需要设置一个合理的值，单位（分钟）。默认：15
suspendTimeMillis（顺序消息）	只适用于顺序消息，设置消息消费失败的重试间隔时间

图 620: Java SDK订阅消息接口及参数说明



23.7.1.2 准备环境

运行 Java 代码之前，请先准备好环境。

操作步骤

1. 通过下面两种方式（任选一种）引入依赖。
   - Maven 方式引入依赖：

     ```
     <dependency>
       <groupId>com.aliyun.openservices</groupId>
       <artifactId>ons-client</artifactId>
       <version>1.2.7.Final</version>
     </dependency>
     ```

   - 下载依赖 JAR包：

     下载链接

2. 代码里涉及到的 Topic， Producer ID， Consumer ID，需要到 MQ 控制台上创建。 Message Tag 可以完全由应用自定义，具体创建过程可参考申请资源。
3. 使用 MQ 服务的应用程序需要部署在 ECS 上。

23.7.1.3 日志配置

介绍MQ 客户端日志的正常打印方式，MQ 客户端日志格式解析以及如何自定义 MQ 客户端日志配置。

打印 MQ 客户端日志

MQ 客户端日志在问题定位排查中扮演着非常重要的角色，通过日志记录客户端运行过程中的异常，能够帮助尽可能真实的还原某个时间点的异常场景，最终达到快速定位、修复 Bug 的目的。

TCP Java SDK 打印 MQ 客户端日志

MQ 的 TCP Java SDK 基于 SLF4J 接口编程，客户端日志的打印依赖您在配置文件中指定的日志实现。目前支持 log4j（暂不支持log4j2）、logback，可在 pom.xml 或者 lib 中添加对应的日志实现依赖即可：

方式一：**依赖 log4j 作为日志实现**

```
<dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>jcl-over-slf4j</artifactId>
   <version>1.7.7</version>
</dependency>
<dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>slf4j-log4j12</artifactId>
   <version>1.7.7</version>
</dependency>
<dependency>
   <groupId>log4j</groupId>
   <artifactId>log4j</artifactId>
   <version>1.2.17</version>
</dependency>
```

方式二：**依赖 logback 作为日志实现**

```
<dependency>
   <groupId>ch.qos.logback</groupId>
   <artifactId>logback-core</artifactId>
   <version>1.1.2</version>
</dependency>
<dependency>
   <groupId>ch.qos.logback</groupId>
   <artifactId>logback-classic</artifactId>
<version>1.1.2</version>
</dependency>
```



说明: 应用中同时依赖 log4j 和 logback 的日志实现会造成日志冲突导致客户端日志打印混乱。确保应用只依赖其中一个日志实现，是正确打印 MQ 客户端日志的前提条件，建议通过 mvn clean dependency:tree | grep log 命令排查。

MQ 客户端日志默认配置

在应用中添加了唯一的日志实现后启动 MQ 客户端，MQ 客户端将会按照如下的配置生成日志文件：

- 日志保存路径：/{user.home}/logs/ons.log，其中{user.home}是指启动当前 Java 进程的用户的根目录。
- 单个日志文件大小：64MB
- 保存历史日志文件的最大个数：10个
- 日志级别：INFO

自定义日志配置

MQ 客户端支持自定义**日志保存路径**、**日志级别**以及**保存历史日志文件的最大个数**；考虑到日志传输以及阅读的便利性，暂不允许自定义**单个日志文件大小**，仍保持默认64MB；可通过配置**保存历史日志文件的最大个数**来自定义日志文件保存的时间范围。

各个参数配置说明如下：

- 日志保存路径：请确保应用进程有对该路径写的权限，否则日志不会打印。
- 保存历史日志文件的最大个数：支持1到100之前的数值，超出范围或者格式错误默认保存10个。
- 日志级别：支持 ERROR、WARN、INFO、DEBUG 中任何一种，不匹配默认 INFO。

TCP Java SDK 自定义 MQ 客户端日志

自定义 MQ 客户端日志配置，请升级 TCP Java SDK 版本到1.2.5及以上。

在 TCP Java SDK 中自定义 MQ 客户端日志配置，请设置如下系统参数：

- ons.client.logRoot：日志保存路径
- ons.client.logFileMaxIndex：保存历史日志文件的最大个数
- ons.client.logLevel：日志级别

举例说明，可在启动脚本中或者 IDE 的 VM options 中添加如下系统参数：

```
-Dons.client.logRoot=/home/admin/logs -Dons.client.logLevel=WARN -Dons.client.logFileMaxIndex=20
```

23.7.1.4 发送普通消息（三种方式）

MQ 发送普通消息有三种实现方式：可靠同步发送、可靠异步发送、单向(Oneway)发送。本节介绍每种实现的原理、使用场景以及三种实现的异同，同时提供了代码示例以供参考。

23.7.1.4.1 可靠同步发送

介绍可靠同步发送的实现原理和应用场景。

原理

同步发送是指消息发送方发出数据后，会在收到接收方发回响应之后才发下一个数据包的通讯方式。

图 621: 同步发送



应用场景

此种方式应用场景非常广泛，例如重要通知邮件、报名短信通知、营销短信系统等。

示例代码

```
public class ProducerTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
        properties.put(PropertyKeyConst.ProducerId, "XXX");//您在控制台创建的Producer ID
        properties.put(PropertyKeyConst.AccessKey,"XXX");// AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
         properties.put(PropertyKeyConst.SecretKey, "XXX");// SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建
           properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");//设置发送超时时间，单位毫秒
        properties.put(PropertyKeyConst.ONSAddr,
```

```
        "http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal");//此处以公共云生产环境为例，专有云环境请参考交付文档
        Producer producer = ONSFactory.createProducer(properties);
        // 在发送消息前，必须调用start方法来启动Producer，只需调用一次即可。
        producer.start();

        //循环发送消息
        for (int i = 0; i < 100; i++){
            Message msg = new Message( //
                // Message Topic
                "TopicTestMQ",
                  // Message Tag 可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条件在MQ服务器过滤
                "TagA",
                // Message Body 可以是任何二进制形式的数据， MQ不做任何干预，
                // 需要Producer与Consumer协商好一致的序列化和反序列化方式
                "Hello MQ".getBytes());
            // 设置代表消息的业务关键属性，请尽可能全局唯一。
            // 以方便您在无法正常收到消息情况下，可通过阿里云服务器管理控制台查询消息并补发。
            // 注意：不设置也不会影响消息正常收发
            msg.setKey("ORDERID_" + i);

            // 同步发送消息，只要不抛异常就是成功
            SendResult sendResult = producer.send(msg);
            System.out.println(sendResult);
        }

        // 在应用退出前，销毁Producer对象
        // 注意：如果不销毁也没有问题
        producer.shutdown();
    }
}
```

23.7.1.4.2 可靠异步发送

介绍可靠异步发送的实现原理和使用场景。

原理

异步发送是指发送方发出数据后，不等接收方发回响应，接着发送下个数据包的通讯方式。MQ 的异步发送，需要您实现异步发送回调接口（SendCallback），在执行消息的异步发送时，应用不需要等待服务器响应即可直接返回，通过回调接口接收务器响应，并对服务器的响应结果进行处理。

图 622: 异步发送



应用场景

异步发送一般用于链路耗时较长，对 RT 响应时间较为敏感的业务场景，例如视频上传后通知启动转码服务，转码完成后通知推送转码结果等。

示例代码

```
   public static void main(String[] args) {
       Properties properties = new Properties();
      properties.put(PropertyKeyConst.AccessKey, "DEMO_AK");// AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
        properties.put(PropertyKeyConst.SecretKey, "DEMO_SK");// SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建
       properties.put(PropertyKeyConst.ProducerId, "DEMO_PID");//您在控制台创建的Producer ID
          properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");//设置发送超时时间，单位毫秒

       Producer producer = ONSFactory.createProducer(properties);
       // 在发送消息前，必须调用start方法来启动Producer，只需调用一次即可。
       producer.start();

       Message msg = new Message(
               // Message Topic
               "TopicTestMQ",
                 // Message Tag,可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条件在MQ服务器过滤
               "TagA",
           // Message Body，任何二进制形式的数据，MQ不做任何干预，需要Producer与Consumer协商好一致的序列化和反序列化方式
               "Hello MQ".getBytes());

      // 设置代表消息的业务关键属性，请尽可能全局唯一。以方便您在无法正常收到消息情况下，可通过MQ控制台查询消息并补发。
       // 注意：不设置也不会影响消息正常收发
       msg.setKey("ORDERID_100");

       // 异步发送消息，发送结果通过callback返回给客户端。
       producer.sendAsync(msg, new SendCallback() {
```

```
        @Override
        public void onSuccess(final SendResult sendResult) {
            // 消费发送成功
            System.out.println("send message success. topic=" + sendResult.getTopic() + ", msgId="
+ sendResult.getMessageId());
        }

        @Override
        public void onException(OnExceptionContext context) {
            // 消息发送失败
                System.out.println("send message failed. topic=" + context.getTopic() + ", msgId=" +
context.getMessageId());
        }
    });

    // 在callback返回之前即可取得msgId。
            System.out.println("send message async. topic=" + msg.getTopic() + ", msgId=" +
msg.getMsgID());

    // 在应用退出前，销毁Producer对象。注意：如果不销毁也没有问题
    producer.shutdown();
  }
```

23.7.1.4.3 单向（Oneway）发送

介绍单向（Oneway）发送的原理、应用场景和示例代码。

原理

单向（Oneway）发送特点为只负责发送消息，不等待服务器回应且没有回调函数触发，即只发送请求不等待应答。此方式发送消息的过程耗时非常短，一般在微秒级别。

图 623: Oneway发送



应用场景

适用于某些耗时非常短，但对可靠性要求并不高的场景，例如日志收集。

下表概括了三者的特点和主要区别。

发送方式	发送TPS	发送结果反馈	可靠性
同步发送	快	有	不丢失
异步发送	快	有	不丢失
单向发送	最快	无	可能丢失

示例代码

```
  public static void main(String[] args) {
      Properties properties = new Properties();
     properties.put(PropertyKeyConst.AccessKey, "DEMO_AK");// AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
      properties.put(PropertyKeyConst.SecretKey, "DEMO_SK");// SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建
      properties.put(PropertyKeyConst.ProducerId, "DEMO_PID");//您在控制台创建的Producer ID
        properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");//设置发送超时时间，单位毫秒
      Producer producer = ONSFactory.createProducer(properties);
      // 在发送消息前，必须调用start方法来启动Producer，只需调用一次即可。
      producer.start();
      //循环发送消息
      for (int i = 0; i < 100; i++){
          Message msg = new Message(
                  // Message Topic
                  "TopicTestMQ",
                  // Message Tag,
                   // 可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条件在MQ服务器过滤
                  "TagA",
                  // Message Body
                    // 任何二进制形式的数据，MQ不做任何干预，需要Producer与Consumer协商好一致的序列化和反序列化方式
                  "Hello MQ".getBytes());

          // 设置代表消息的业务关键属性，请尽可能全局唯一。
          // 以方便您在无法正常收到消息情况下，可通过阿里云服务器管理控制台查询消息并补发。
          // 注意：不设置也不会影响消息正常收发
          msg.setKey("ORDERID_" + i);

          // oneway发送消息，只要不抛异常就是成功
          producer.sendOneway(msg);
      }

      // 在应用退出前，销毁Producer对象
      // 注意：如果不销毁也没有问题
      producer.shutdown();
  }
```

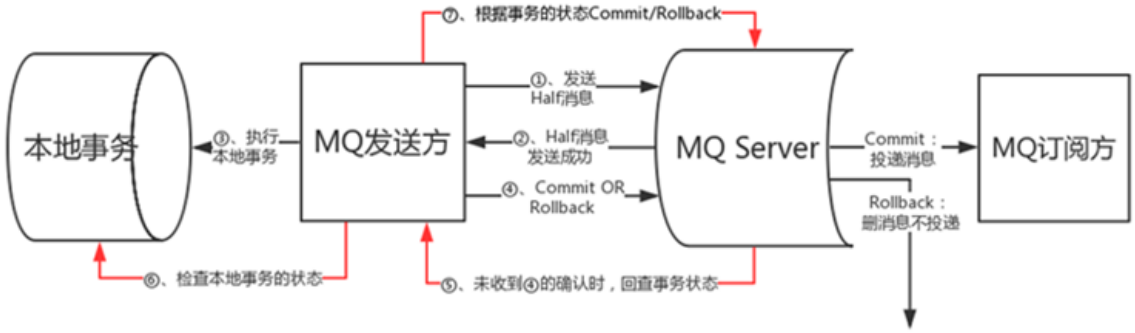
23.7.1.5 发送分布式事务消息

介绍如何发布分布式事务消息。

背景信息

MQ事务消息交互流程如下：

图 624: MQ事务消息交互流程



操作步骤

1. 发送半消息及执行本地事务。

```
package com.alibaba.webx.TryHsf.app1;

import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.transaction.LocalTransactionExecuter;
import com.aliyun.openservices.ons.api.transaction.TransactionProducer;
import com.aliyun.openservices.ons.api.transaction.TransactionStatus;
import java.util.Properties;
import java.util.concurrent.TimeUnit;

public class TransactionProducerClient {
 private final static Logger log = ClientLogger.getLog(); // 用户需要设置自己的log, 记录日志便于排查问题

 public static void main(String[] args) throws InterruptedException {
    final BusinessService businessService = new BusinessService(); // 本地业务Service
    Properties properties = new Properties();
    properties.put(PropertyKeyConst.ProducerId, ""); // 您在控制台创建的Producer ID
    properties.put(PropertyKeyConst.AccessKey, ""); // 阿里云身份验证，在阿里云服务器管理控制台创建
     properties.put(PropertyKeyConst.SecretKey, ""); // 阿里云身份验证，在阿里云服务器管理控制台创建
    properties.put(PropertyKeyConst.ONSAddr,
        "http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal");//此处以公共云生产环境为例，专有云环境请参考交付文档
    TransactionProducer producer = ONSFactory.createTransactionProducer(properties,
        new LocalTransactionCheckerImpl());
```

```
    producer.start();
    Message msg = new Message("Topic", "TagA", "Hello MQ transaction===".getBytes());
    // 输入您在控制台创建的Topic
    SendResult sendResult = producer.send(msg, new LocalTransactionExecuter() {
        @Override
        public TransactionStatus execute(Message msg, Object arg) {
            // 消息ID(有可能消息体一样，但消息ID不一样，当前消息ID在控制台无法查询)
            String msgId = msg.getMsgID();
            // 消息体内容进行crc32, 也可以使用其他的如MD5
            long crc32Id = HashUtil.crc32Code(msg.getBody());
            // 消息ID和crc32id主要是用来防止消息重复
            // 如果业务本身是幂等的，可以忽略，否则需要利用msgId或crc32Id来做幂等
            // 如果要求消息绝对不重复，推荐做法是对消息体body使用crc32或md5来防止重复消息
            Object businessServiceArgs = new Object();
            TransactionStatus transactionStatus = TransactionStatus.Unknow;
            try {
                boolean isCommit =
                        businessService.execbusinessService(businessServiceArgs);
                if (isCommit) {
                    // 本地事务成功、提交消息
                    transactionStatus = TransactionStatus.CommitTransaction;
                } else {
                    // 本地事务失败、回滚消息
                    transactionStatus = TransactionStatus.RollbackTransaction;
                }
            } catch (Exception e) {
                log.error("Message Id:{}", msgId, e);
            }
            System.out.println(msg.getMsgID());
            log.warn("Message Id:{}transactionStatus:{}", msgId, transactionStatus.name());
            return transactionStatus;
        }
    }, null);
    // demo example 防止进程退出(实际使用不需要这样)
    TimeUnit.MILLISECONDS.sleep(Integer.MAX_VALUE);
}
}
```

2. 提交事务消息状态。

   当本地事务执行完成（执行成功或执行失败），需要通知服务器当前消息的事务状态。通知方式有以下两种：

   - 执行本地事务完成后提交；
   - 执行本地事务一直没提交状态，等待服务器回查消息的事务状态。

   事务状态有以下三种：

   - TransactionStatus.CommitTransaction 提交事务，允许订阅方消费该消息。
   - TransactionStatus.RollbackTransaction 回滚事务，消息将被丢弃不允许消费。
   - TransactionStatus.Unknow 无法判断状态，期待MQ Broker向发送方再次询问该消息对应的本地事务的状态。

```
import com.alibaba.rocketmq.client.producer.LocalTransactionState;
```

```
public class LocalTransactionCheckerImpl implements LocalTransactionChecker {
private final static Logger log = ClientLogger.getLog();
final  BusinessService businessService = new BusinessService();

@Override
public TransactionStatus check(Message msg) {
    //消息ID(有可能消息体一样，但消息ID不一样，当前消息属于Half 消息，所以消息ID在控制台无法查询)
    String msgId = msg.getMsgID();
    //消息体内容进行crc32, 也可以使用其他的方法如MD5
    long crc32Id = HashUtil.crc32Code(msg.getBody());
    //消息ID、消息本 crc32Id主要是用来防止消息重复
    //如果业务本身是幂等的，可以忽略，否则需要利用msgId或crc32Id来做幂等
    //如果要求消息绝对不重复，推荐做法是对消息体使用crc32或md5来防止重复消息。
    //业务自己的参数对象，这里只是一个示例，实际需要用户根据情况来处理
    Object businessServiceArgs = new Object();
    TransactionStatus transactionStatus = TransactionStatus.Unknow;
    try {
        boolean isCommit = businessService.checkbusinessService(businessServiceArgs);
        if (isCommit) {
            //本地事务已成功、提交消息
            transactionStatus = TransactionStatus.CommitTransaction;
        } else {
            //本地事务已失败、回滚消息
            transactionStatus = TransactionStatus.RollbackTransaction;
        }
    } catch (Exception e) {
        log.error("Message Id:{}", msgId, e);
    }
    log.warn("Message Id:{}transactionStatus:{}", msgId, transactionStatus.name());
    return transactionStatus;
}
}
```

后续操作

事务回查机制说明

1. 发送事务消息为什么必须要实现回查机制？

   当步骤（1）中半消息发送完成，但本地事务返回状态为TransactionStatus.Unknow时，或者应用退出导致本地事务未提交任何状态时，从MQ Broker的角度看，这条Half状态的消息的状态是未知的，因此MQ Broker会定期要求发送方能Check该Half状态消息，并上报其最终状态。

2. Check被回调时，业务逻辑都需要做些什么？

   MQ事务消息的check方法里面，应该写一些检查事务一致性的逻辑。MQ发送事务消息时需要实现LocalTransactionChecker接口，用来处理MQ Broker主动发起的本地事务状态回查请求；因此在事务消息的Check方法中，需要完成两件事情：

   - 检查该半消息对应的本地事务的状态（commited or rollback）；
   - 向MQ Broker提交该半消息本地事务的状态。

23.7.1.6 发送延时消息

本文介绍发送延时消息的场景及代码示例。

延时消息用于指定消息发送到MQ服务器端后，延时一段时间才被投递到客户端进行消费（例如3秒后才被消费），适用于解决一些消息生产和消费有时间窗口要求的场景，或者通过消息触发延迟任务的场景，类似于延迟队列。

代码示例

```
public class ProducerDelayTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
        properties.put(PropertyKeyConst.ProducerId, "XXX");// 您在控制台创建的 Producer ID
          properties.put(PropertyKeyConst.AccessKey, "XXX");// AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
         properties.put(PropertyKeyConst.SecretKey, "XXX");// SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建
        properties.put(PropertyKeyConst.ONSAddr,
            "http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal");//此处以公共云生产环境为例，专有云环境请参考交付文档
        Producer producer = ONSFactory.createProducer(properties);
        // 在发送消息前，必须调用 start 方法来启动 Producer，只需调用一次即可。
        producer.start();
        Message msg = new Message( //
                // Message Topic
                "Topic",
                 // Message Tag, 可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条件在MQ服务器过滤
                "tag",
                 // Message Body 可以是任何二进制形式的数据， MQ 不做任何干预，需要 Producer 与 Consumer 协商好一致的序列化和反序列化方式
                "Hello MQ".getBytes());
        // 设置代表消息的业务关键属性，请尽可能全局唯一。
        // 以方便您在无法正常收到消息情况下，可通过 MQ 控制台查询消息并补发。
        // 注意：不设置也不会影响消息正常收发
        msg.setKey("ORDERID_100");
       // 延时时间单位为毫秒（ms），指定一个时刻，在这个时刻之后才能被消费，这个例子表示3秒后才能被消费
        long delayTime = 3000;
        msg.setStartDeliverTime(System.currentTimeMillis() + delayTime);
        // 发送消息，只要不抛异常就是成功
        SendResult sendResult = producer.send(msg);
        System.out.println("Message Id:" + sendResult.getMessageId());
        // 在应用退出前，销毁Producer对象<br>
        // 注意：如果不销毁也没有问题
        producer.shutdown();
    }
}
```

23.7.1.7 发送定时消息

本文介绍发送定时消息的场景及代码示例。

定时消息可以做到在指定时间戳之后才可被消费者消费，用于解决一些消息生产和消费有时间窗口要求的场景，或者通过消息触发定时任务的场景。

代码示例

```
public class ProducerDelayTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
        properties.put(PropertyKeyConst.ProducerId, "XXX");//您在 MQ 控制台创建的Producer ID
        properties.put(PropertyKeyConst.AccessKey, "XXX");// 阿里云身份验证，在阿里云服务器管理控制台创建
        properties.put(PropertyKeyConst.SecretKey, "XXX");// 阿里云身份验证，在阿里云服务器管理控制台创建
        properties.put(PropertyKeyConst.ONSAddr,
          "http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal");//此处以公共云生产环境为例，专有云环境请参考交付文档
        Producer producer = ONSFactory.createProducer(properties);
        // 在发送消息前，必须调用 start 方法来启动 Producer，只需调用一次即可。
        producer.start();
        Message msg = new Message( //
                // Message Topic
                "Topic",
                  // Message Tag 可理解为 Gmail 中的标签，对消息进行再归类，方便Consumer指定过滤条件在MQ服务器过滤
                "tag",
                 // Message Body 可以是任何二进制形式的数据， MQ 不做任何干预，需要 Producer 与 Consumer 协商好一致的序列化和反序列化方式
                "Hello MQ".getBytes());
        // 设置代表消息的业务关键属性，请尽可能全局唯一
        // 以方便您在无法正常收到消息情况下，可通过 MQ 控制台查询消息并补发。
        // 注意：不设置也不会影响消息正常收发
        msg.setKey("ORDERID_100");
        /**
         * 定时消息投递，设置投递的具体时间戳，单位毫秒例如2016-03-07 16:21:00投递
        */
             long timeStamp =new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse("2016-03-07 16:21:00").getTime();
        msg.setStartDeliverTime(timeStamp);

        // 发送消息，只要不抛异常就是成功
        SendResult sendResult = producer.send(msg);
        System.out.println("Message Id:" + sendResult.getMessageId());
        // 在应用退出前，销毁 Producer 对象<br>
        // 注意：如果不销毁也没有问题
        producer.shutdown();
    }
}
```

23.7.1.8 发送顺序消息

顺序消息是 MQ 提供的一种按照顺序进行发布消费的消息类型，适用由于需要严格按照先进先出的原则进行消息发布和消费的场景。详情请参考顺序消息文档。

发送和订阅顺序消息，请使用 Java SDK 1.2.7 及以上版本。

全局顺序消息和分区顺序消息的收发方式基本一样，具体请参考以下示例代码。

发送消息示例代码

```
package com.aliyun.openservices.ons.example.order;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.order.OrderProducer;
import java.util.Properties;
public class ProducerClient {
    public static void main(String[] args) {
        Properties properties = new Properties();
        properties.put(PropertyKeyConst.ProducerId, "XXX");// 您在控制台创建的 Producer ID
         properties.put(PropertyKeyConst.AccessKey, "XXX");// AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
         properties.put(PropertyKeyConst.SecretKey, "XXX");// SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建
        properties.put(PropertyKeyConst.ONSAddr,
                properties.put(PropertyKeyConst.ONSAddr,  "http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal");//此处以公共云生产环境为例，专有云环境请参考交付文档
        OrderProducer producer = ONSFactory.createOrderProducer(properties);
        // 在发送消息前，必须调用 start 方法来启动 Producer，只需调用一次即可。
        producer.start();
        for (int i = 0; i < 1000; i++) {
            String orderId = "biz_" + i % 10;
            Message msg = new Message(//
                    // Message所属的Topic
                    "Order_global_topic",
                       // Message Tag, 可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条件在MQ服务器过滤
                    "TagA",
                  // Message Body 可以是任何二进制形式的数据， MQ 不做任何干预，需要 Producer 与 Consumer 协商好一致的序列化和反序列化方式
                    "send order global msg".getBytes()
            );
            // 设置代表消息的业务关键属性，请尽可能全局唯一。
            // 以方便您在无法正常收到消息情况下，可通过 MQ 控制台查询消息并补发。
            // 注意：不设置也不会影响消息正常收发
            msg.setKey(orderId);
              // 分区顺序消息中区分不同分区的关键字段，sharding key于普通消息的key是完全不同的概念。
            // 全局顺序消息，该字段可以设置为任意非空字符串。
            String shardingKey = String.valueOf(orderId);
            // 发送消息，只要不抛异常就是成功
            SendResult sendResult = producer.send(msg, shardingKey);
            System.out.println("Message Id:" + sendResult.getMessageId());
        }
        // 在应用退出前，销毁Producer对象
        // 注意：如果不销毁也没有问题
        producer.shutdown();
    }
```

```
}
```

订阅消息示例代码

```
package com.aliyun.openservices.ons.example.order;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.order.ConsumeOrderContext;
import com.aliyun.openservices.ons.api.order.MessageOrderListener;
import com.aliyun.openservices.ons.api.order.OrderAction;
import com.aliyun.openservices.ons.api.order.OrderConsumer;
import java.util.Properties;
public class ConsumerClient {
    public static void main(String[] args) {
        Properties properties = new Properties();
        properties.put(PropertyKeyConst.ConsumerId, "XXX");// 您在控制台创建的 Consumer ID
         properties.put(PropertyKeyConst.AccessKey, "XXX");// AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
         properties.put(PropertyKeyConst.SecretKey, "XXX");// SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建
                properties.put(PropertyKeyConst.ONSAddr,
                  "http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal");//此处以公共云生产环境为例，专有云环境请参考交付文档
         properties.put(PropertyKeyConst.SuspendTimeMillis, "100"); // 顺序消息消费失败进行重试前的等待时间单位(毫秒)
         properties.put(PropertyKeyConst.MaxReconsumeTimes, "20"); // 消息消费失败时的最大重试次数
        // 在订阅消息前，必须调用 start 方法来启动 Consumer，只需调用一次即可。
        OrderConsumer consumer = ONSFactory.createOrderedConsumer(properties);
        consumer.subscribe(
                // Message所属的Topic
                "Jodie_Order_Topic",
                // 订阅指定Topic下的Tags：
                // 1. * 表示订阅所有消息
                // 2. TagA || TagB || TagC 表示订阅TagA 或 TagC 或 TagD 的消息
                "*",
                new MessageOrderListener() {
                    /**
                     * 1. 消息消费处理失败或者处理出现异常，返回OrderAction.Suspend<br>
                     * 2. 消息处理成功，返回与返回OrderAction.Success
                     */
                    @Override
                    public OrderAction consume(Message message, ConsumeOrderContext context) {
                        System.out.println(message);
                        return OrderAction.Success;
                    }
                });
        consumer.start();
    }
}
```

23.7.1.9 集群方式订阅消息

本文介绍集群方式订阅消息的定义及代码示例。

集群订阅即某个消费者集群只消费指定的 Topic，而不是消费所有 Topic。

```
public class ConsumerTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
        properties.put(PropertyKeyConst.ConsumerId, "XXX");// 您在控制台创建的 Consumer ID
         properties.put(PropertyKeyConst.AccessKey, "XXX");// AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
         properties.put(PropertyKeyConst.SecretKey, "XXX");// SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建
        properties.put(PropertyKeyConst.ONSAddr,
          "http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal");//此处以公共云生产环境为例，专有云环境请参考交付文档
        Consumer consumer = ONSFactory.createConsumer(properties);
        consumer.subscribe("TopicTestMQ", "*", new MessageListener() {
            public Action consume(Message message, ConsumeContext context) {
                System.out.println("Receive: " + message);
                return Action.CommitMessage;
            }
        });
        consumer.start();
        System.out.println("Consumer Started");
    }
}
```

23.7.1.10 Spring 集成

介绍如何在 Spring 框架下用 MQ 收发消息。主要包括三部分内容：普通消息生产者和 Spring 集成，事务消息生产者和 Spring 集成，消息消费者与 Spring 集成。

23.7.1.10.1 生产者与 Spring 集成

生产者与Spring集成操作如下。

操作步骤

1. 在 producer.xml 中定义生产者 Bean 等信息。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                          xsi:schemaLocation="http://www.springframework.org/schema/beans    http://www.springframework.org/schema/beans/spring-beans.xsd">

         <bean  id="producer"  class="com.aliyun.openservices.ons.api.bean.ProducerBean"  init-method="start" destroy-method="shutdown">
        <property name="properties" > <!--生产者配置信息-->
          <props>
            <prop key="ProducerId">PID_DEMO</prop> <!--请替换XXX-->
            <prop key="AccessKey">XXX</prop>
            <prop key="SecretKey">XXX</prop>
          </props>
        </property>
    </bean>

</beans>
```

2. 通过已经与 Spring 集成好的生产者生产消息。

```
package demo;
```

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.exception.ONSClientException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ProduceWithSpring {
    public static void main(String[] args) {
        /**
          * 生产者Bean配置在producer.xml中，可通过ApplicationContext获取或者直接注入到其他类(比如具体的Controller)中。
         */
        ApplicationContext context = new ClassPathXmlApplicationContext("producer.xml");

        Producer producer = (Producer) context.getBean("producer");
        //循环发送消息
        for (int i = 0; i < 100; i++) {
            Message msg = new Message( //
                    // Message Topic
                    "TopicTestMQ",
                        // Message Tag 可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条件在MQ服务器过滤
                    "TagA",
                    // Message Body 可以是任何二进制形式的数据， MQ不做任何干预
                    // 需要Producer与Consumer协商好一致的序列化和反序列化方式
                    "Hello MQ".getBytes());
            // 设置代表消息的业务关键属性，请尽可能全局唯一
            // 以方便您在无法正常收到消息情况下，可通过MQ 控制台查询消息并补发
            // 注意：不设置也不会影响消息正常收发
            msg.setKey("ORDERID_100");
            // 发送消息，只要不抛异常就是成功
            try {
                SendResult sendResult = producer.send(msg);
                assert sendResult != null;
                System.out.println("send success: " + sendResult.getMessageId());
            }catch (ONSClientException e) {
                System.out.println("发送失败");
            }

        }
    }
}
```

23.7.1.10.2 事务消息生产者与 Spring 集成

事务消息生产者与 Spring 集成步骤如下。

背景信息

有关事务消息的概念请查看发布分布式事务消息。

操作步骤

1. 实现 LocalTransactionChecker，如下所示。

```
package demo;
```

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.transaction.LocalTransactionChecker;
import com.aliyun.openservices.ons.api.transaction.TransactionStatus;

public class DemoLocalTransactionChecker implements LocalTransactionChecker {
   public TransactionStatus check(Message msg) {
      System.out.println("开始回查本地事务状态");
            return  TransactionStatus.CommitTransaction;  //根据本地事务状态检查结果返回不同
的TransactionStatus
   }
}
```

2. 在 transactionProducer.xml 中定义事务消息生产者 Bean 等信息。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                        xsi:schemaLocation="http://www.springframework.org/schema/beans    http://
www.springframework.org/schema/beans/spring-beans.xsd">

   <bean id="localTransactionChecker" class="demo.DemoLocalTransactionChecker"></bean>

               <bean    id="transactionProducer"    class="com.aliyun.openservices.ons.api.bean.
TransactionProducerBean" init-method="start" destroy-method="shutdown">
      <property name="properties" > <!--事务消息生产者配置信息-->
         <props>
            <prop key="ProducerId">PID_DEMO</prop> <!--请替换XXX-->
            <prop key="AccessKey">AKDEMO</prop>
            <prop key="SecretKey">SKDEMO</prop>
         </props>
      </property>
      <property name="localTransactionChecker" ref="localTransactionChecker"></property>
   </bean>

</beans>
```

3. 通过已经与 Spring 集成好的生产者生产事务消息。

```
package demo;

import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.transaction.LocalTransactionExecuter;
import com.aliyun.openservices.ons.api.transaction.TransactionProducer;
import com.aliyun.openservices.ons.api.transaction.TransactionStatus;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ProduceTransMsgWithSpring {

   public static void main(String[] args) {
      /**
         * 事务消息生产者Bean配置在transactionProducer.xml中，可通过ApplicationContext获取
或者直接注入到其他类(比如具体的Controller)中。
       * 请结合例子"发送事务消息"
       */
                                 ApplicationContext    context    =    new    ClassPathXmlApplicat
ionContext("transactionProducer.xml");

                           TransactionProducer    transactionProducer    =    (TransactionProducer)
 context.getBean("transactionProducer");
```

```
        Message msg = new Message("XXX", "TagA", "Hello MQ transaction===".getBytes());

       SendResult sendResult = transactionProducer.send(msg, new LocalTransactionExecuter() {
            @Override
            public TransactionStatus execute(Message msg, Object arg) {
                System.out.println("执行本地事务");
                    return TransactionStatus.CommitTransaction; //根据本地事务执行结果来返回不同
的TransactionStatus
            }
        }, null);
    }
}
```

23.7.1.10.3 消费者与 Spring 集成

消费者与 Spring 集成步骤如下。

操作步骤

1. 创建 MessageListener，如下所示。

```
package demo;

import com.aliyun.openservices.ons.api.Action;
import com.aliyun.openservices.ons.api.ConsumeContext;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.MessageListener;

public class DemoMessageListener implements MessageListener {

    public Action consume(Message message, ConsumeContext context) {

        System.out.println("Receive: " + message.getMsgID());
        try {
            //do something..
            return Action.CommitMessage;
        }catch (Exception e) {
            //消费失败
            return Action.ReconsumeLater;
        }
    }
}
```

2. 在 consumer.xml 中定义消费者 Bean 等信息。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                           xsi:schemaLocation="http://www.springframework.org/schema/beans    http://
www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="msgListener" class="demo.DemoMessageListener"></bean> <!--Listener配置-->

        <bean  id="consumer"  class="com.aliyun.openservices.ons.api.bean.ConsumerBean"  init-
method="start" destroy-method="shutdown">
        <property name="properties" > <!--消费者配置信息-->
            <props>
                <prop key="ConsumerId">CID_DEMO</prop> <!--请替换XXX-->
                <prop key="AccessKey">AKDEMO</prop>
```

```
            <prop key="SecretKey">SKDEMO</prop>
            <!--将消费者线程数固定为50个
            <prop key="ConsumeThreadNums">50</prop>
            -->
        </props>
    </property>
    <property name="subscriptionTable">
        <map>
            <entry value-ref="msgListener">
                <key>
                    <bean class="com.aliyun.openservices.ons.api.bean.Subscription">
                        <property name="topic" value="TopicTestMQ"/>
                        <property name="expression" value="*"/><!--expression即Tag，可以设置成具体的Tag，如 taga||tagb||tagc，也可设置成*。 *仅代表订阅所有Tag，不支持通配-->
                    </bean>
                </key>
            </entry>
            <!--更多的订阅添加entry节点即可-->
        </map>
    </property>
</bean>

</beans>
```

3. 运行已经与 Spring 集成好的消费者，如下所示。

```
package demo;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ConsumeWithSpring {
    public static void main(String[] args) {

        /**
         * 消费者Bean配置在consumer.xml中，可通过ApplicationContext获取或者直接注入到其他类(比如具体的Controller)中
         */
        ApplicationContext context = new ClassPathXmlApplicationContext("consumer.xml");
        System.out.println("Consumer Started");
    }
}
```

23.7.2 使用C/C++ SDK 接入

本章节介绍如何使用C/C++ SDK进行MQ的TCP接入，主要包括以下内容：

23.7.2.1 准备环境

用 C++ SDK 方式接入 MQ，需要完成以下准备工作。

说明:

- 代码里涉及到的 Topic， Producer ID， Consumer ID，需要到 MQ 控制台上创建。

  Message Tag 可以完全由应用自定义，具体创建过程可参考申请资源申请资源。
- 使用 MQ 服务的应用程序需要部署在阿里云 ECS 上。

23.7.2.1.1 下载SDK

CPP 支持 Windows 和 Linux 两个跨平台的 SDK，而且接口完全一致。

操作步骤

1. 下载依赖 SDK 包链接。
   - *Linux CPP* 下载链接
   - *Windows CPP* 下载链接
2. 下载完成后，解压压缩包。

   解压完成后，会有如下目录结构：
   - example/
   - include/
   - lib/
   - SDK_GUIDE.pdf
   - changelog

   上面的目录和文件的作用如下：
   - demo: 包含了一个创建好的 Windows C++ 演示 Demo。
   - example：包含了普通消息发送、Oneway 消息发送、顺序消息发送、普通消息消费、顺序消息消费等例子，Linux 下还包含了 Makefile 用于 example 的编译和管理。
   - include：自己编写的程序需要 include 的头文件。
   - lib：
     - Linux SDK 子目录如下，分别是 64 的静态库和动态库。

       ```
       lib-boost-share/
           libonsclient4cpp.so
       lib-boost-static/
           libonsclient4cpp.a
       ```
     - Windows SDK 子目录如下，是 64 位系统下 SDK 的 dll 库。如果没有安装 Visual Studio 2015 环境下，需要拷贝安装vc_redist.x64。这是 Visual C++ 2015的运行时环境。

       ```
       64/
       vc_redist.x64
       ```
   - SDK_GUIDE.pdf： SDK 环境准备文档和 FAQ。
   - changelog：新版本发布解决的问题和引入的新特性列表。

23.7.2.1.2 使用 Linux C++ SDK

2016.12.02开始，Linux CPP 版本依赖了高性能 boost 库(1.62.0版本)，不仅降低了 CPU 资源占用率，而且提高了运行效率。目前主要依赖了 boost_system，boost_thread，boost_chrono，boost_filesystem 四个库。我们提供了静态库和动态库两种解决方案。

静态解决方案

MQ 库文件在 lib/lib-boost-static 目录下，boost 库静态链接到 libonsclient4cpp.a 中。对于没有依赖 boost 库的业务方，可以直接选用静态库方案。静态库方案中，相应的boost库已经链接到 ibonsclient4cpp.a，编译时只需要链接 libonsclient4cpp.a 即可，无需执行其他操作。使用方式如下：

```
cd aliyun-mq-linux-cpp-sdk //下载的SDK解压后的路径
cd example //进入demo目录，修改demo文件，填入自己申请的topic，key相关的信息
g++ -static -I ../include -L ../lib/lib-boost-static ProducerExampleForEx.cpp -lonsclient4cpp -lpthread –ldl
```

完全的静态链接请确保机器上安装了 libstdc++，pthread 等相关的静态库，默认安装的 libstdc++ 是没有安装静态库的，所以需要通过 yum 或者 > apt-get 来安装相关的静态库。此外使用如上方式会出现一些警告信息如下：

```
warning: Using 'gethostbyaddr' in statically linked applications requires at runtime the shared libraries from the glibc version used for linking
```

建议最佳的方式，不要使用完全的静态链接，而是只静态链接 lonsclient4cpp，其他库动态链接即可。使用方式如下：

```
g++ -I ../include -L ../lib/lib-boost-static ProducerExampleForEx.cpp –Wl,-dn -lonsclient4cpp –Wl,-dy -lpthread –ldl
```

动态解决方案

MQ 库文件在 lib/lib-boost-share 目录下，需要业务方生成可执行文件时链接 boost 动态库和 libonsclient4cpp.so。对于业务方已经依赖了 boost 库，需要选择动态库方案的情况，对 boost 库的依赖需要做如下工作：

1. 下载 boost 1.62.0 版本：

   boost 1.62.0

2. 解压 boost 1.62.0:

   tar —bzip2 -xf /path/to/boost_1_62_0.tar.bz2

3. 安装 boost 1.62.0 版本：

   a. cd path/to/boost_1_62_0

   b. 配置 boost：./bootstrap.sh

   c. 编译 boost: ./b2 link=shared runtime-link=shared

   d. 安装 boost: ./b2 install

4. 执行 ldconfig -v|grep libboost。如果有相关的输出表明 boost 动态库在动态库搜索路径中。

5. 生成可执行文件时，需要链接 boost 动态库和 MQ 动态库。方法如下：

```
cd aliyun-mq-linux-cpp-sdk  //下载的 SDK 解压后的路径
cd example                  //进入 demo 目录，修改 demo 文件，填入自己在 MQ 控制台申请的 Topic，key 相关的信息
g++ -Wall -Wno-deprecated -L ../lib/lib-boost-share/ -I ../include/ ProducerExampleForEx.cpp -lonsclient4cpp -lboost_system -lboost_thread -lboost_chrono -lboost_filesystem -lpthread
export LD_LIBRARY_PATH="../lib/lib-boost-share/"  //添加动态载入的搜索路径
./a.out  //运行程序
```

23.7.2.1.3 使用 Windows C++ SDK

本文介绍如何在Windows中使用C++ SDK 。

背景信息

Windows中又细分为两种使用环境：

- Visual Studio 2015 环境下使用 C++ SDK
- 非 Visual Studio 2015 环境下使用 C++ SDK

操作步骤

Visual Studio 2015 环境下使用 C++ SDK

1. 使用 Visual Studio 2015 创建自己的项目。

   图 625: VS环境-1

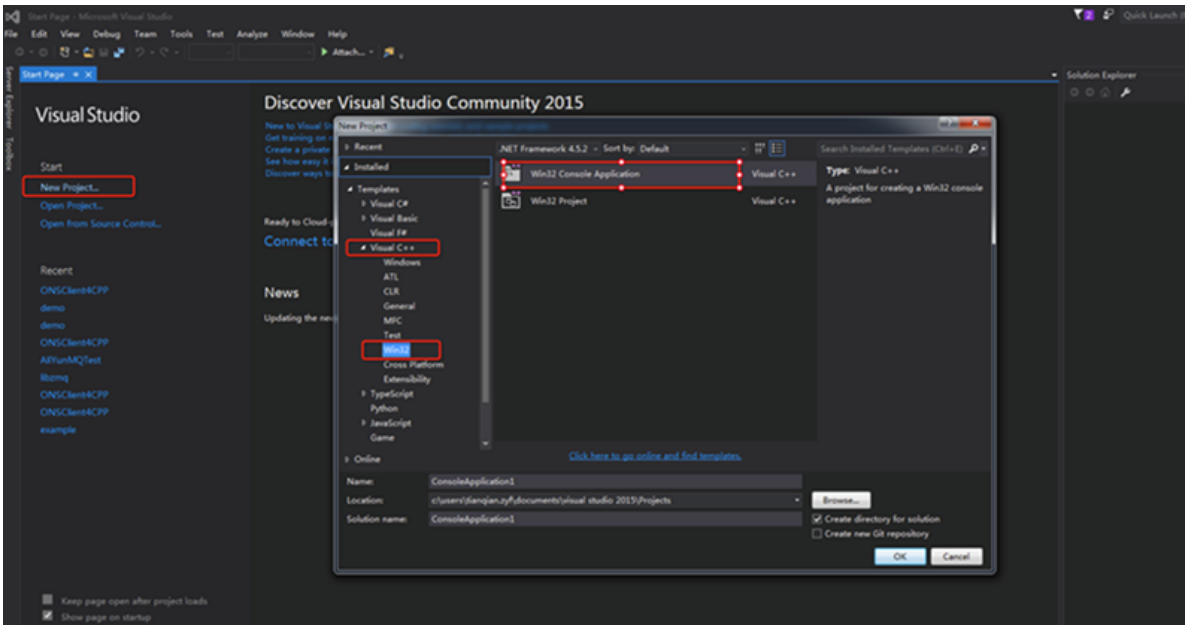


图 626: VS环境-2

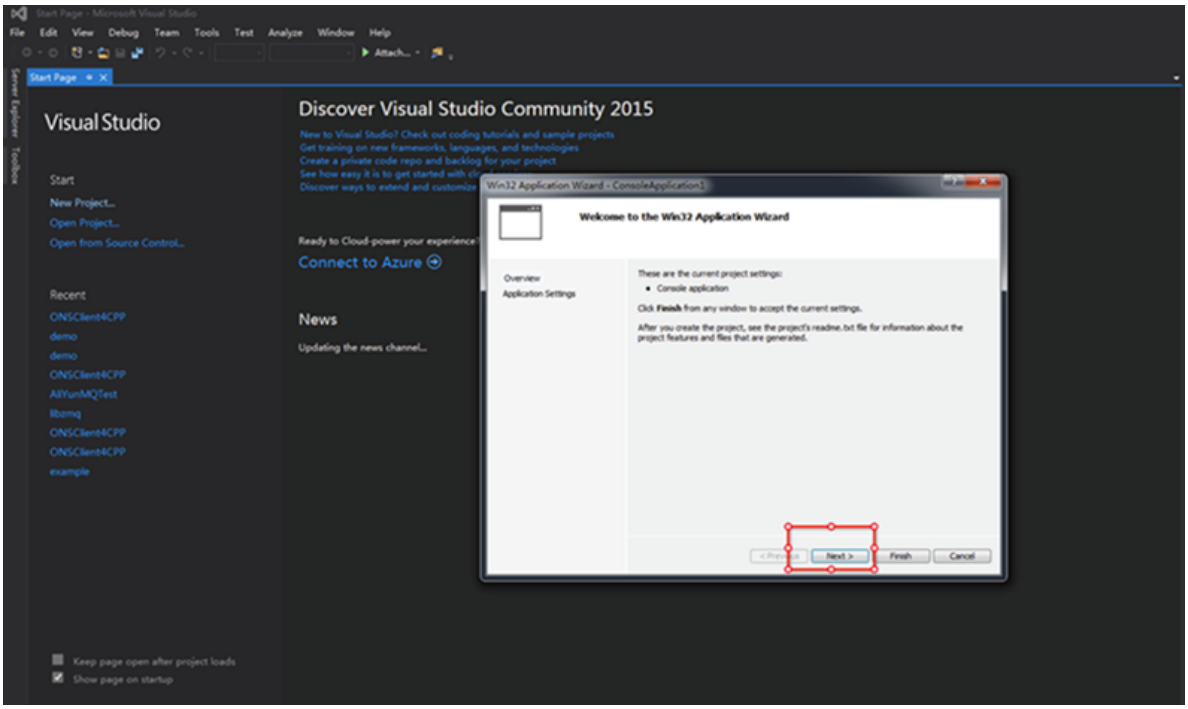
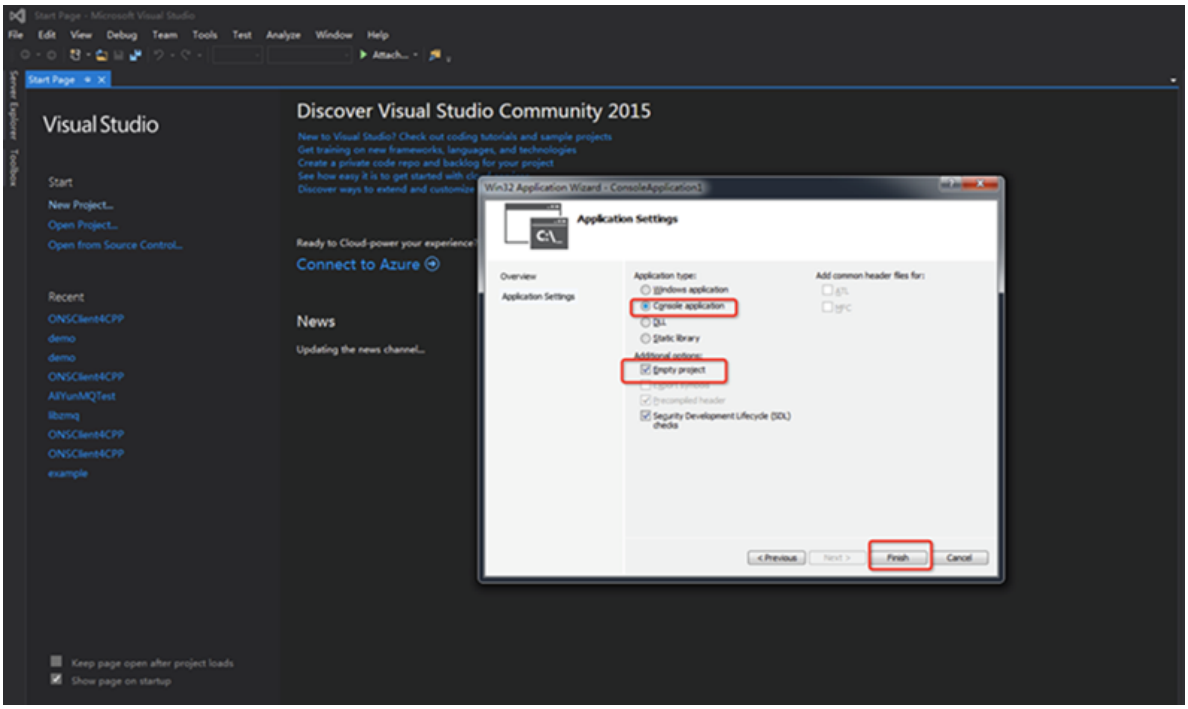
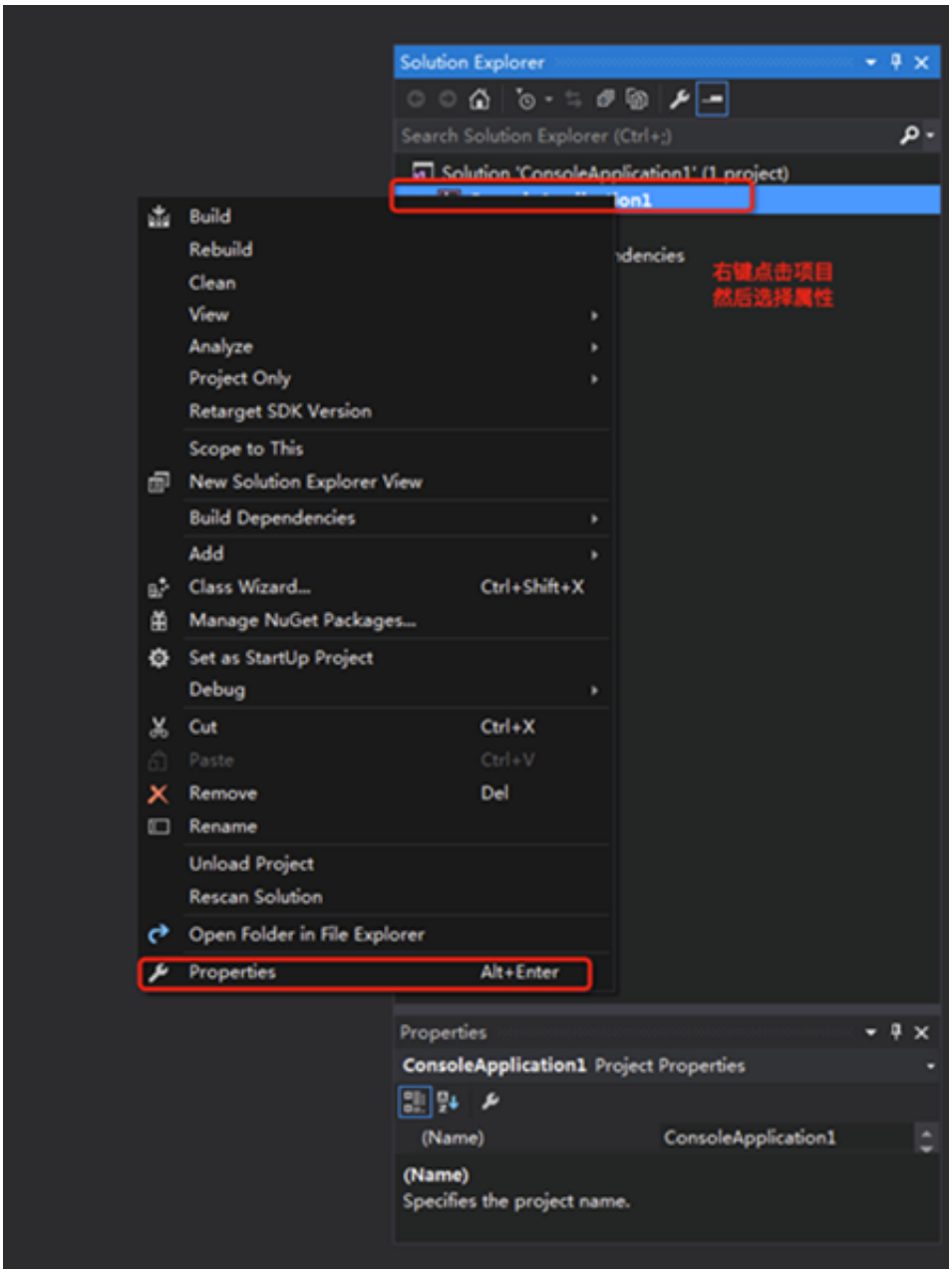
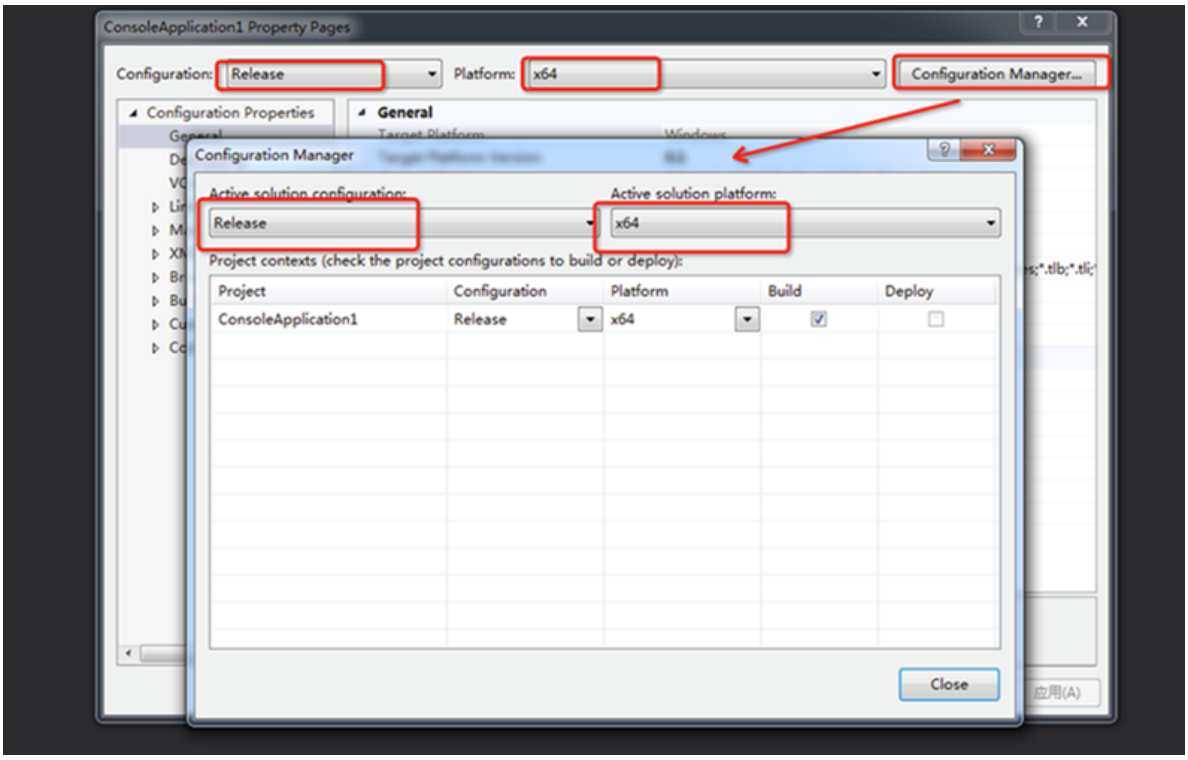


图 627: VS环境-3



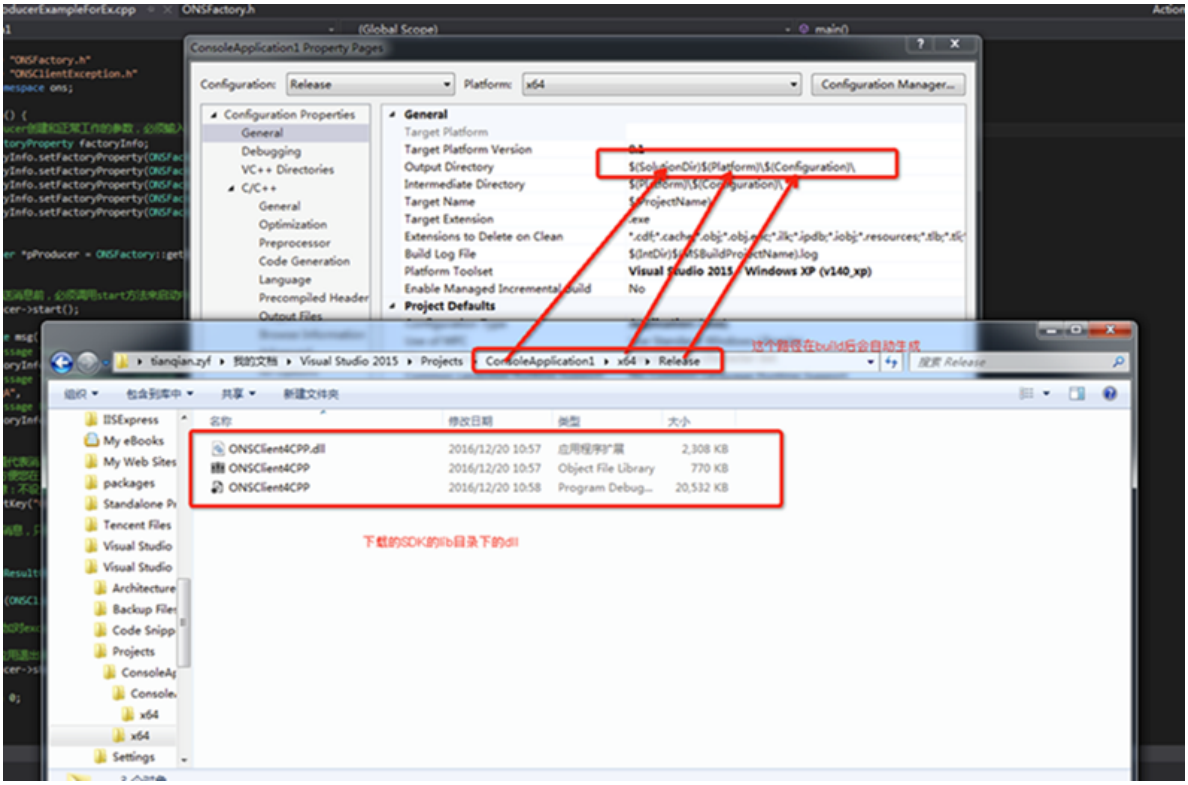
2. 右键单击项目，选择**属性>配置管理器**，设置活动解决方案配置为 **release**，设置活动解决方案平台为 **x64**。


Solution Explorer
Search Solution Explorer (Ctrl+;)
Build
Rebuild
Clean
View
Analyze
Project Only
Retarget SDK Version
Scope to This
New Solution Explorer View
Build Dependencies
Add
Class Wizard... Ctrl+Shift+X
Manage NuGet Packages...
Set as StartUp Project
Debug
Cut Ctrl+X
Paste Ctrl+V
Remove Del
Rename
Unload Project
Rescan Solution
Open Folder in File Explorer
Properties Alt+Enter
右键点击项目
然后选择属性
Properties
ConsoleApplication1 Project Properties
(Name) ConsoleApplication1
(Name)
Specifies the project name.



3. 右键单击项目，依次选择**属性>配置属性>常规>输出目录**：**/A**。按照**活动解决方案平台**的设置，拷贝 64 位 lib 目录下的所有文件到输出目录 **/A**。

图 628: VS环境-4



4. 右键单击项目，依次选择**属性>配置属性>C/C++-常规>附加包含目录**：**/B**。拷贝 include 目录下的头文件到包含目录：**/B**。

图 629: VS环境-5

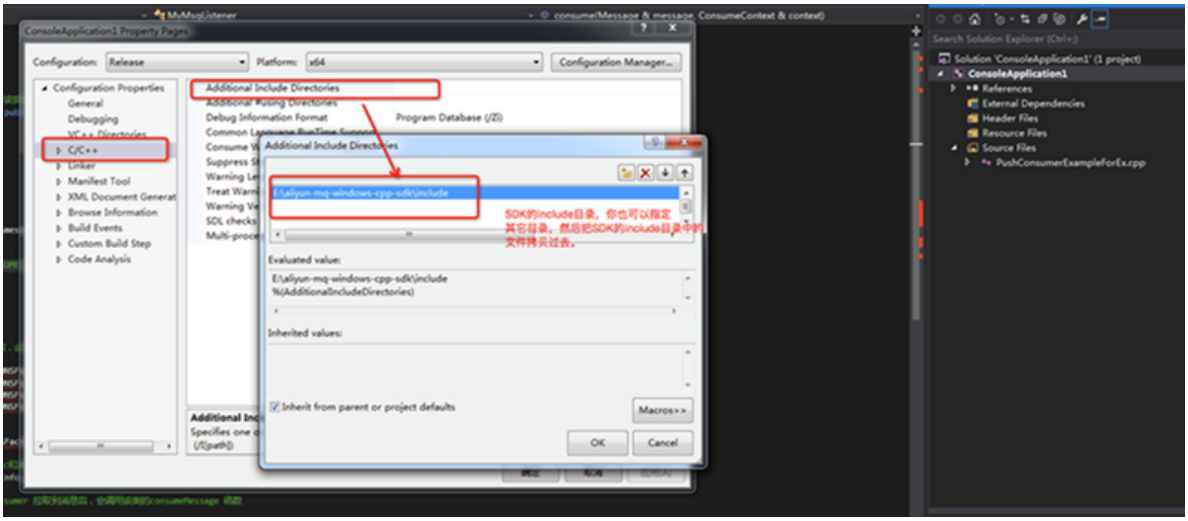
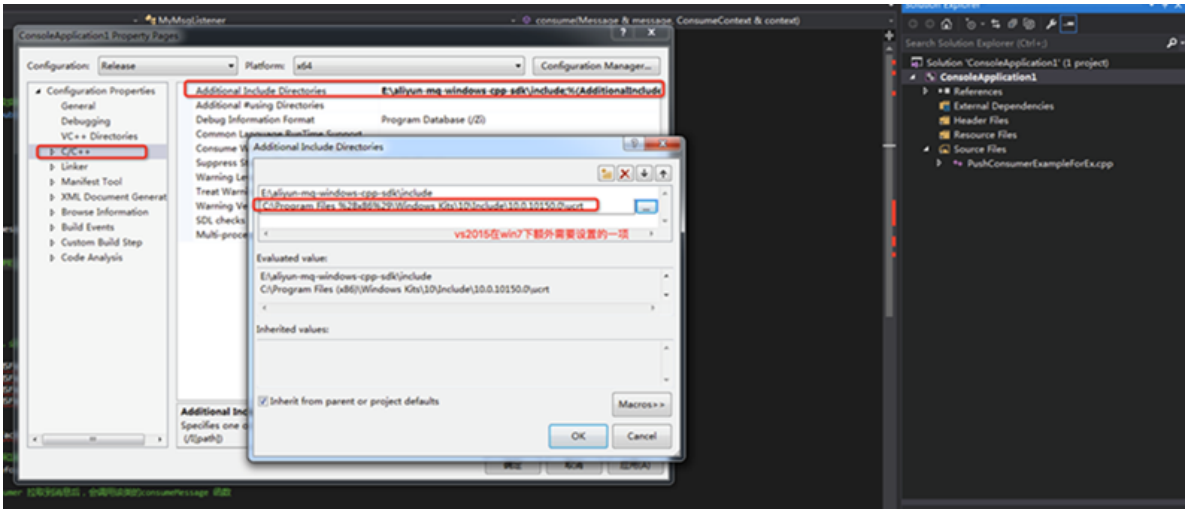
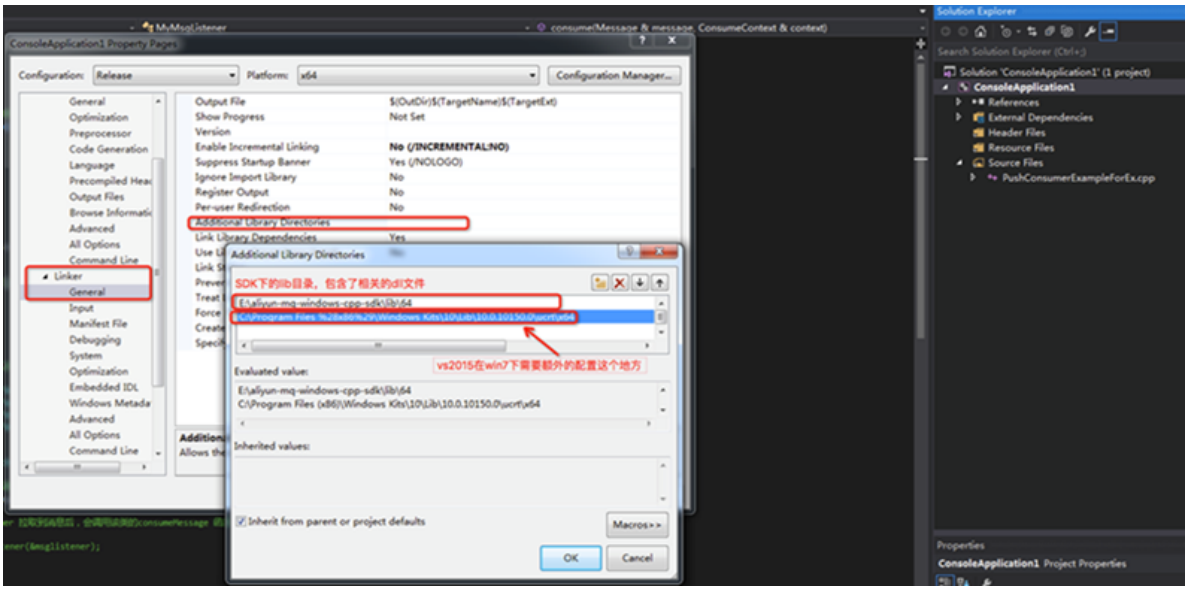


图 630: VS环境-6



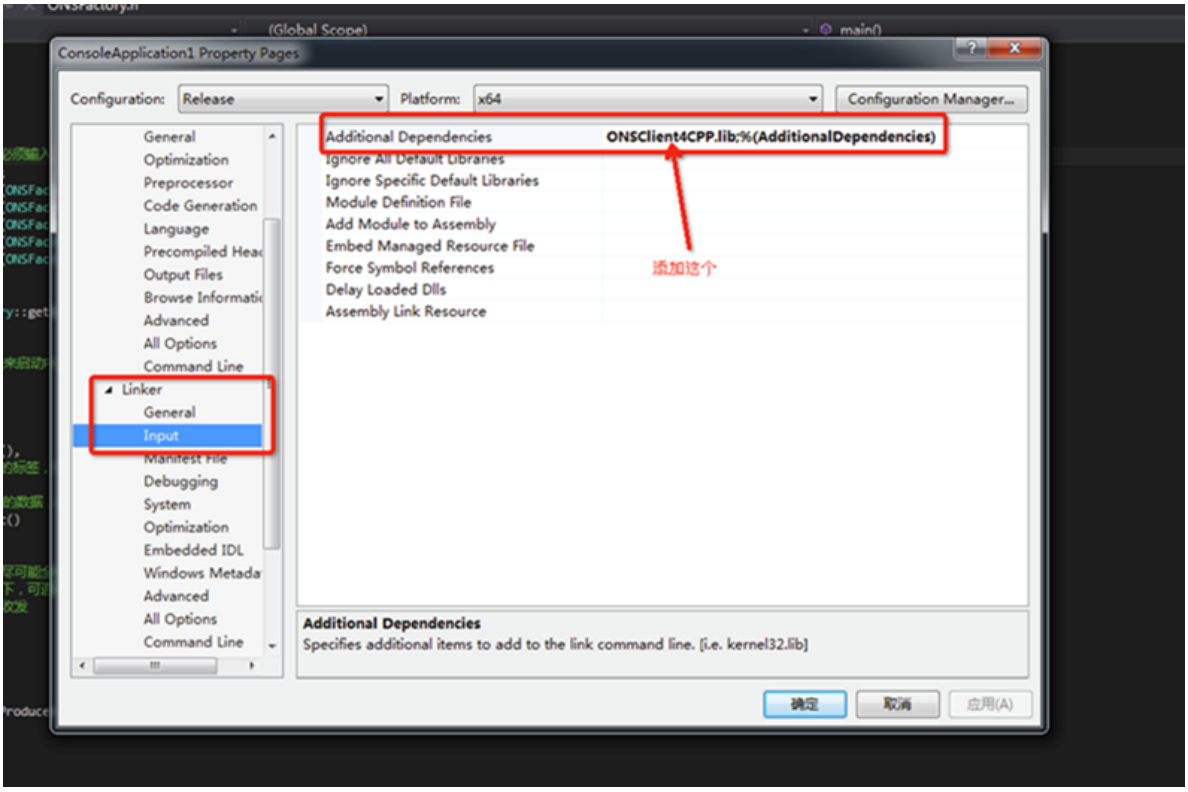
5. 右键单击项目，依次选择**属性>配置属性>链接>常规>附加库目录**：**/A**。

图 631: VS环境-7



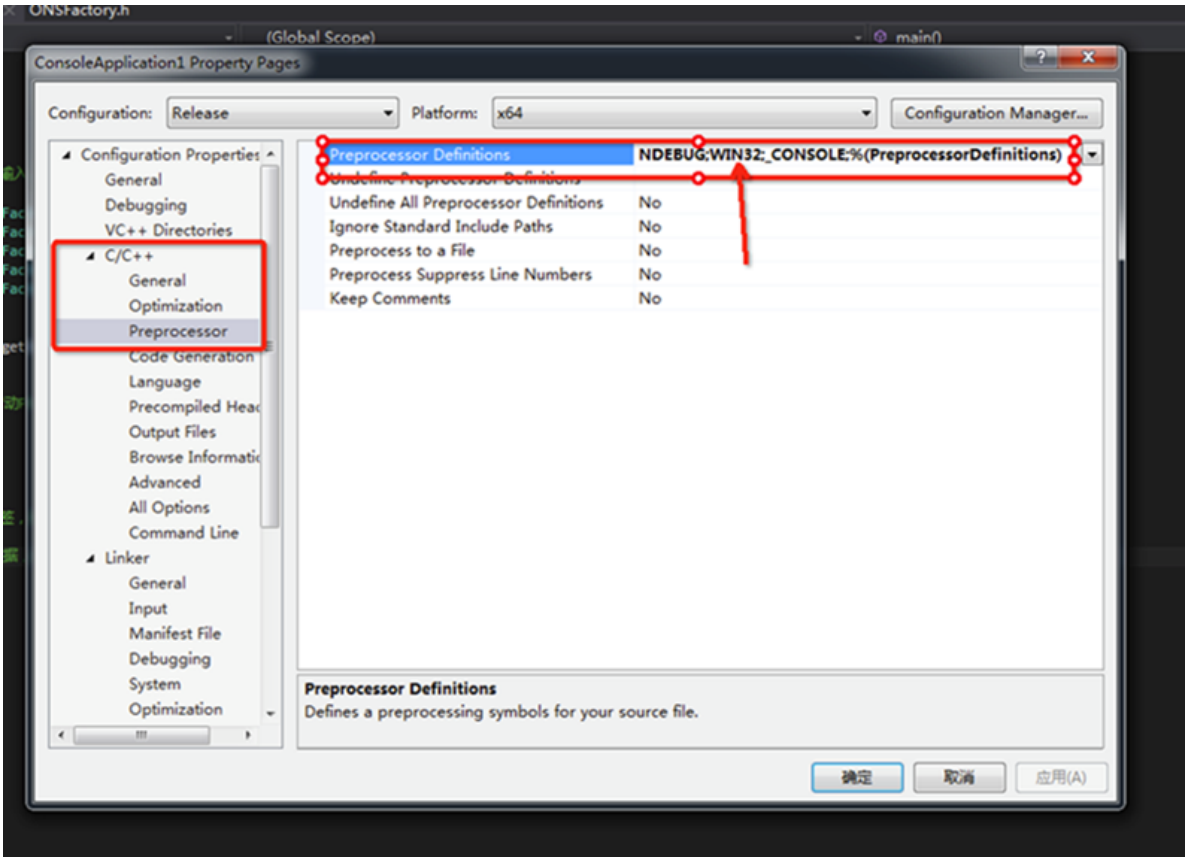
6. 右键单击项目，依次选择**属性>配置属性>链接>输入>附加依赖项**：**ONSClient4CPP.lib。**

图 632: VS环境-8



7. 右键单击项目，依次选择**属性>配置属性>>C/C++-常规>>预处理器定义**: 添加 **WIN32**宏。

图 633: VS环境-9



非 Visual Studio 2015 环境下使用 C++ SDK

8. 首先需要按照 Visusal Studio 2015 的环境来配置，配置过程同上。
9. 安装 **vc_redist.x64**。这是 **Visual C++ 2015** 的运行时环境。

> **说明:** 如果不想进行繁琐的设置，您也可以使用设置好的 SDK Demo，下载 SDK 后进行解压，然后进入 Demo 目录，使用 VS2015 打开工程。

图 634: 非VS环境-1

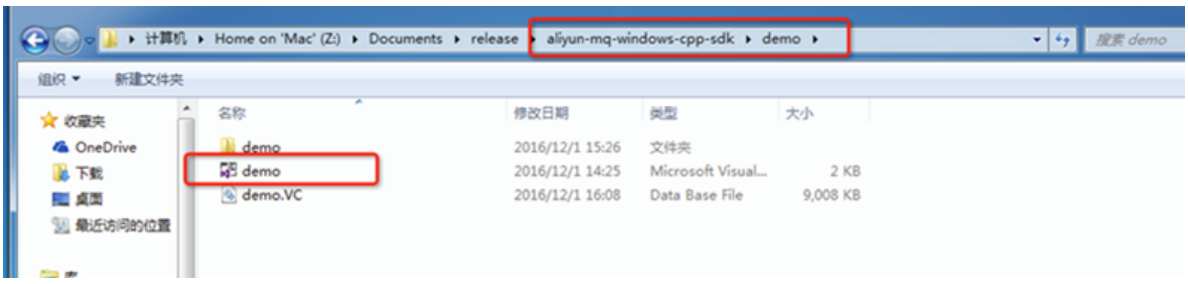


图 635: 非VS环境-2

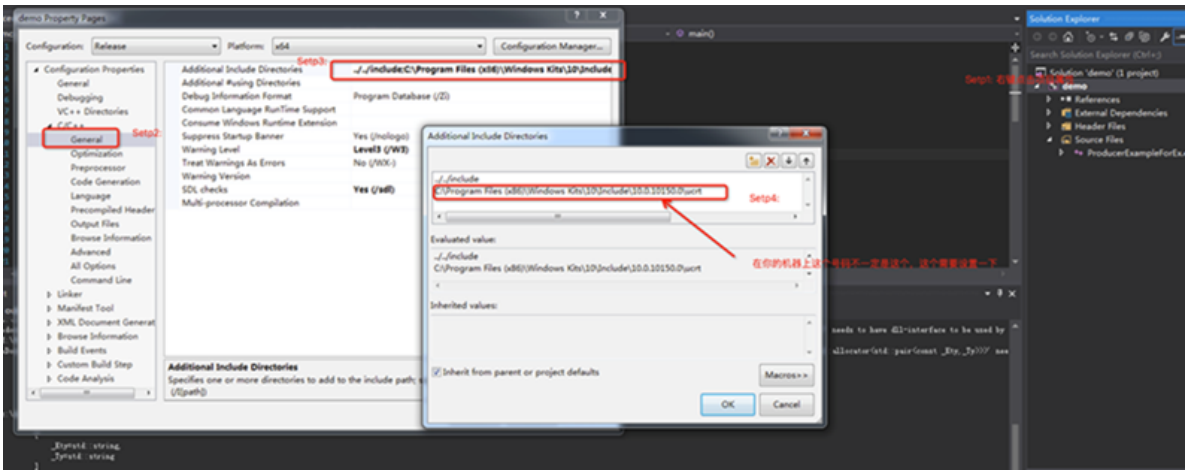
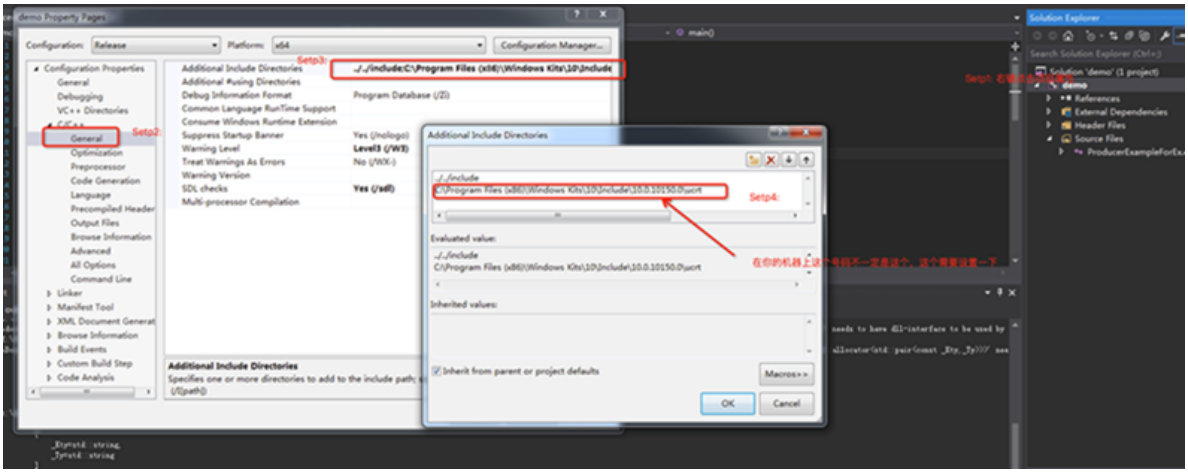


图 636: 非VS环境-3



到此为此就设置好编译环境了。点击 Build，即可编译出可执行的程序，然后拷贝 dll 到可执行程序的目录下即可运行，或者拷贝 dll 到系统目录下。

23.7.2.2 发送普通消息

本文介绍发送普通消息的代码示例。

请参考以下示例代码进行消息发送。

```
#include "ONSFactory.h"
#include "ONSClientException.h"

using namespace ons;

int main()
{

    //创建producer和发送消息所必需的信息；
    ONSFactoryProperty factoryInfo;
```

```
        factoryInfo.setFactoryProperty(ONSFactoryProperty::ProducerId,   "XXX");//您在控制台创建的Producer ID
    factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );// 消息内容
    factoryInfo.setFactoryProperty(ONSFactoryProperty::MsgContent, "XXX");//消息内容
     factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "XXX");//AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
    factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "XXX" );//SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建

    //create producer;
    Producer *pProducer = ONSFactory::getInstance()->createProducer(factoryInfo);

    //在发送消息前，必须调用start方法来启动Producer，只需调用一次即可；
    pProducer->start();

    Message msg(
            //Message Topic
            factoryInfo.getPublishTopics(),
             //Message Tag,可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条件在MQ服务器过滤
            "TagA",
              //Message Body,不能为空，MQ不做任何干预，需要Producer与Consumer协商好一致的序列化和反序列化方式
            factoryInfo.getMessageContent()
    );

    // 设置代表消息的业务关键属性，请尽可能全局唯一
    // 以方便您在无法正常收到消息情况下，可通过 MQ 控制台查询消息并补发
    // 注意：不设置也不会影响消息正常收发
    msg.setKey("ORDERID_100");

    //发送消息，只要不抛出异常，就代表发送成功
    try
    {
        SendResultONS sendResult = pProducer->send(msg);
    }
    catch(ONSClientException & e)
    {
        //自定义处理exception的细节
    }
    // 在应用退出前，必须销毁Producer对象，否则会导致内存泄露等问题
    pProducer->shutdown();

    return 0;
}
```

23.7.2.3 发送定时消息

本文介绍发送定时消息的代码示例。

定时消息可以做到在指定时间之后才可被消费者消费，用于解决一些消息生产和消费有时间窗口要求的场景，或者通过消息触发定时任务的场景，类似于延迟队列。代码示例如下：

```
#include "ONSFactory.h"
#include "ONSClientException.h"
using namespace ons;
int main()
{
```

```
    //创建producer和发送消息所必需的信息；
    ONSFactoryProperty factoryInfo;
      factoryInfo.setFactoryProperty(ONSFactoryProperty::ProducerId,  "XXX");//您在MQ控制台创建的producer
    factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );//您在MQ控制台申请的topic
    factoryInfo.setFactoryProperty(ONSFactoryProperty::MsgContent, "xxx");//msg content
    factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "xxx");//阿里云身份验证，在阿里云服务器管理控制台创建
    factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "xxx" );//阿里云身份验证，在阿里云服务器管理控制台创建

    //create producer;
    Producer *pProducer = ONSFactory::getInstance()->createProducer(factoryInfo);

    //在发送消息前，必须调用start方法来启动Producer，只需调用一次即可；
    pProducer->start();

    Message msg(
            //Message Topic
            factoryInfo.getPublishTopics(),
             //Message Tag,可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条件在 MQ 服务器过滤
            "TagA",
             //Message Body, 不能为空，MQ不做任何干预，需要Producer与Consumer协商好一致的序列化和反序列化方式
            factoryInfo.getMessageContent()
    );

    // 设置代表消息的业务关键属性，请尽可能全局唯一。
    // 以方便您在无法正常收到消息情况下，可通过 MQ 控制台查询消息并补发。
    // 注意：不设置也不会影响消息正常收发
    msg.setKey("ORDERID_100");

     // deliver time 单位 ms，指定一个时刻，在这个时刻之后才能被消费，这个例子表示3s后才能被消费
    long deliverTime = 获取系统当前时间(ms) + 3000;
    msg.setStartDeliverTime(deliverTime);

    //发送消息，只要不抛出异常，就代表发送成功
    try
    {
        SendResultONS sendResult = pProducer->send(msg);
    }
    catch(ONSClientException & e)
    {
        //自定义处理exception的细节
    }

    // 在应用退出前，必须销毁Producer对象，否则会导致内存泄露等问题
    pProducer->shutdown();

    return 0;
}
```

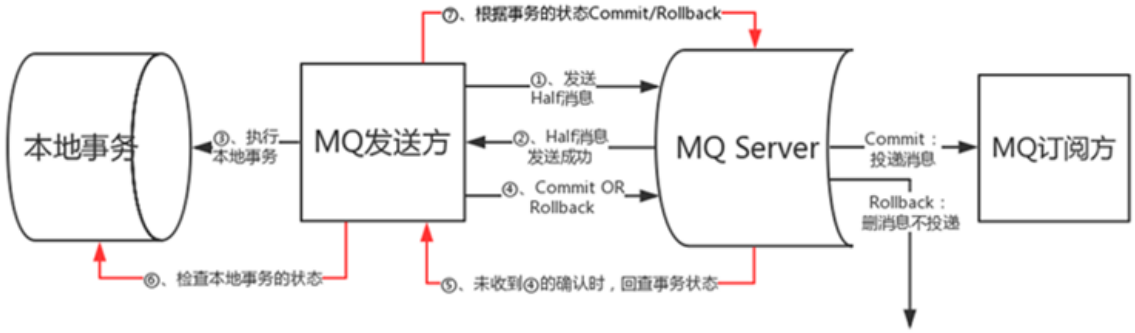
23.7.2.4 发送分布式事务消息

本文介绍如何发送分布式事务消息。

背景信息

MQ 事务消息交互流程如下。

图 637: MQ事务消息交互流程



操作步骤

1. 发送半消息（Half Message）及执行本地事务。

```
using System;
 using System.Collections.Generic;
 using System.Linq;
 using System.Text;
 using System.Runtime.InteropServices;
 using ons;

 namespace ons
 {
 public class MyLocalTransactionExecuter : LocalTransactionExecuter
 {
    public MyLocalTransactionExecuter()
    {
    }

    ~MyLocalTransactionExecuter()
    {
    }
    public override TransactionStatus execute(ref Message value)
    {
                    Console.WriteLine("execute topic: {0}, tag:{1}, key:{2}, msgId:{3},msgbody:{4},
userProperty:{5}",
                value.getTopic(), value.getTag(), value.getKey(), value.getMsgID(), value.getBody(),
value.getUserProperty("VincentNoUser"));

            // 消息ID(有可能消息体一样，但消息ID不一样，当前消息ID在控制台控制不可能查询)
            string msgId = value.getMsgID();
            // 消息体内容进行crc32, 也可以使用其他的如MD5
            // 消息ID和crc32id主要是用来防止消息重复
```

```
            // 如果业务本身是幂等的，可以忽略，否则需要利用msgId或crc32Id来做幂等
          // 如果要求消息绝对不重复，推荐做法是对消息体body使用crc32或md5来防止重复消息。
            TransactionStatus transactionStatus = TransactionStatus.Unknow;
            try {
                boolean isCommit = 本地事务执行结果;
                if (isCommit) {
                    // 本地事务成功、提交消息
                    transactionStatus = TransactionStatus.CommitTransaction;
                } else {
                    // 本地事务失败、回滚消息
                    transactionStatus = TransactionStatus.RollbackTransaction;
                }
            } catch (Exception e) {
                //exception handle
            }
            return transactionStatus；
    }
}
class onscsharp
{

    static void Main(string[] args)
    {
        ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
         factoryInfo.setFactoryProperty(factoryInfo.getProducerIdName(), "");//您在MQ控制台申请的Producer ID
          factoryInfo.setFactoryProperty(factoryInfo.getPublishTopicsName(), "");// 您在MQ控制台申请的Topic
        factoryInfo.setFactoryProperty(factoryInfo.getMsgContentName(), "");//message body
       factoryInfo.setFactoryProperty(factoryInfo.getAccessKeyName(), "");//AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
        factoryInfo.setFactoryProperty(factoryInfo.getSecretKeyName(), "");//SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建

        //create transaction producer
        ONSFactory onsfactory = new ONSFactory();
        LocalTransactionChecker myChecker = new MyLocalTransactionChecker();
                TransactionProducer pProducer = onsfactory.getInstance().createTransactionProducer(factoryInfo,ref myChecker);

        //在发送消息前，必须调用start方法来启动Producer，只需调用一次即可，启动之后可以多线程并发发送消息
        pProducer.start();

            Message msg = new Message(
            //Message Topic
            factoryInfo.getPublishTopics(),
            //Message Tag
            "TagA",
            //Message Body
            factoryInfo.getMessageContent()
        );

        // 设置代表消息的业务关键属性，请尽可能全局唯一。
        // 以方便您在无法正常收到消息情况下，可通过MQ K控制台查询消息并补发。
        // 注意：不设置也不会影响消息正常收发
        msg.setKey("ORDERID_100");

        //发送消息，只要不抛出异常，就代表发送成功
        try
```

```
        {
            LocalTransactionExecuter myExecuter = new MyLocalTransactionExecuter();
            SendResultONS sendResult = pProducer.send(msg, ref myExecuter);
        }
        catch(ONSClientException e)
        {
            Console.WriteLine("\nexception of sendmsg:{0}",e.what() );
        }

        // 在应用退出前，必须销毁Producer对象，否则会导致内存泄露等问题；
        // shutdown之后不能重新start此producer
        pProducer.shutdown();
    }
}
}
```

2. 提交事务消息状态。

当本地事务执行完成（执行成功或执行失败），需要通知服务器当前消息的事务状态。通知方式有以下两种：

- 执行本地事务完成后提交。
- 执行本地事务一直没提交状态，等待服务器回查消息的事务状态。

事务状态有以下三种：

- TransactionStatus.CommitTransaction 提交事务，允许订阅方消费该消息。
- TransactionStatus.RollbackTransaction 回滚事务，消息将被丢弃不允许消费。
- TransactionStatus.Unknow 无法判断状态，期待 MQ Broker 向发送方再次询问该消息对应的本地事务的状态。

```
public class MyLocalTransactionChecker : LocalTransactionChecker
{
    public MyLocalTransactionChecker()
    {
    }

    ~MyLocalTransactionChecker()
    {
    }

    public override TransactionStatus check(ref Message value)
    {
                    Console.WriteLine("check topic: {0}, tag:{1}, key:{2}, msgId:{3},msgbody:{4},
userProperty:{5}",
            value.getTopic(), value.getTag(), value.getKey(), value.getMsgID(), value.getBody(),
value.getUserProperty("VincentNoUser"));
            // 消息ID(有可能消息体一样，但消息ID不一样，当前消息ID在控制台控制不可能查询)
            string msgId = value.getMsgID();
            // 消息体内容进行crc32，也可以使用其他的如MD5
            // 消息ID和crc32id主要是用来防止消息重复
            // 如果业务本身是幂等的，可以忽略，否则需要利用msgId或crc32Id来做幂等
                // 如果要求消息绝对不重复，推荐做法是对消息体body使用crc32或md5来防止重复
消息。
            TransactionStatus transactionStatus = TransactionStatus.Unknow;
```

```
        try {
            boolean isCommit = 本地事务执行结果;
            if (isCommit) {
                // 本地事务成功、提交消息
                transactionStatus = TransactionStatus.CommitTransaction;
            } else {
                // 本地事务失败、回滚消息
                transactionStatus = TransactionStatus.RollbackTransaction;
            }
        } catch (Exception e) {
            //exception handle
        }
        return transactionStatus；
    }
    }
```

后续操作

事务回查机制说明

1. 发送事务消息为什么必须要实现check机制？

   当步骤1发送半消息完成，但本地事务返回状态为 TransactionStatus.Unknow 时，亦或是应用退出导致本地事务未提交任何状态时，从 MQ Broker 的角度看，这条半状态的消息的状态是未知的，因此 MQ Broker 会定期要求发送方能 check 该半状态消息，并上报其最终状态。

2. Check 被回调时，业务逻辑都需要做些什么？

   MQ 事务消息的 check 方法里面，应该写一些检查事务一致性的逻辑。MQ 发送事务消息时需要实现 LocalTransactionChecker 接口，用来处理 MQ Broker主动发起的本地事务状态回查请求；因此在事务消息的 check 方法中，需要完成两件事情：

   a. 检查该半消息对应的本地事务的状态（commited or rollback）。

   b. 向 MQ Broker 提交该半消息本地事务的状态。

3. 本地事务的不同状态对半消息的影响？

   - TransactionStatus.CommitTransaction 提交事务，允许订阅方消费该消息。
   - TransactionStatus.RollbackTransaction 回滚事务，消息将被丢弃不允许消费。
   - TransactionStatus.Unknow 无法判断状态，期待 MQ Broker 向发送方再次询问该消息对应的本地事务的状态。

   具体代码请参见 MyLocalTransactionChecker 的实现。

23.7.2.5 集群方式订阅消息

本文介绍集群方式订阅消息的定义及代码示例。

集群订阅即某个消费者集群只消费指定的Topic，而不是消费所有Topic。

```
#include "ONSFactory.h"
using namespace ons;

// MyMsgListener：创建消费消息的实例
//pushConsumer拉取到消息后，会主动调用该实例的consume 函数
class MyMsgListener : public MessageListener
{

    public:

        MyMsgListener()
        {
        }

        virtual ~MyMsgListener()
        {
        }

        virtual Action consume(Message &message, ConsumeContext &context)
        {
            //自定义消息处理细节
            return CommitMessage; //CONSUME_SUCCESS;
        }
};


int main(int argc, char* argv[])
{

    //pushConsumer创建和工作需要的参数，必须输入
    ONSFactoryProperty factoryInfo;
     factoryInfo.setFactoryProperty(ONSFactoryProperty::ConsumerId, "XXX");//您在MQ控制台申请
的consumerId
   factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );//您在MQ控制台申请
的msg topic
     factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "XXX");//阿里云身份验证，在
阿里云服务器管理控制台创建
   factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "XXX");//阿里云身份验证，在阿
里云服务器管理控制台创建

    //create pushConsumer
                        PushConsumer*       pushConsumer       =       ONSFactory::getInstance()-
>createPushConsumer(factoryInfo);

    //指定pushConsumer 订阅的消息topic和tag, 注册消息回调函数
    MyMsgListener  msglistener;
    pushConsumer->subscribe(factoryInfo.getPublishTopics(), "*",&msglistener );

    //start pushConsumer
    pushConsumer->start();

   //NOTE:直到不再接收消息，才能调用shutdown；调用shutdown之后，consumer退出，不能接收
到任何消息

    //销毁pushConsumer, 在应用退出前，必须销毁Consumer 对象，否则会导致内存泄露等问题
    pushConsumer->shutdown();
    return 0;

}
```

23.7.3 .NET SDK

本节介绍如何使用.NET SDK进行MQ的TCP接入。

23.7.3.1 环境准备

用 .NET SDK 方式接入 MQ，需要完成以下准备工作。

说明:

- 代码里涉及到的 Topic， Producer ID， Consumer ID，需要到 MQ 控制台上创建。Message Tag 可以完全由应用自定义，具体创建过程可参考申请资源。
- 使用 MQ 服务的应用程序需要部署在阿里云 ECS 上。

23.7.3.1.1 下载SDK

MQ提供的.NET 版本是基于 MQ CPP 版本的托管封装，这样能保证 .NET 完全不依赖于 Windows .NET 公共库，内部采用 C++ 多线程并发处理，保证.NET版本的高效稳定。

背景信息

在使用 VS 开发 .NET 的应用程序和类库时，默认的目标平台为“Any CPU”，即运行时可根据 CPU 类型自动选择 X86 或 X64。拥有这样的能力是因为 .NET 编译后的程序集是基于 IL 的。在运行时，CLR 才会将其 JIT 发射为 X86 或 X64 的机器码。而 C 或 C++ 编译生成的 DLL 就是机器码。所以，其平台的决策是在编译时决定的。通过编译选项的设置，我们将 C/C++ 项目编译为 X86 的32位 dll 或者 X64 的 64 位 dll，因此我们提供了包含 VS2015 编译的 release64 位版本 DLL。其他 VS 版本也可以使用。

操作步骤

1. 下载依赖 SDK 包链接。

   SDK有新、旧两个版本。

   - 旧版 *Windows .NET SDK* 下载

     说明:

     - 基于托管封装的 SDK 存在诸多问题，并且无法稳定在 ASP.NET 上正常工作，因此在2016/12/29号开始推出新版本的 SDK。
     - 新版 SDK 基于 C# PINVOKE 调用底层的 dll，利用开源软件 SWIG 生成 PINVOKE 封装代码。新版 SDK 相比于托管版本的 SDK 更稳定，部署安装更简单。
     - 托管版本的 SDK 不再维护，只提供最后一次的稳定版本。

- 新版 *Windows .NET SDK* 下载

2. 下载完成后，解压压缩包。

   解压完成后，会有如下目录结构：

   - example/
   - lib/
   - demo/
   - interface/
   - SDK_GUIDE.pdf
   - changelog

   上面的目录和文件的作用如下：

   - example：包含了普通消息发送、Oneway 消息发送、顺序消息发送、普通消息消费、顺序消息消费等例子。
   - lib：底层的 C++ DLL 相关文件，以及**Virtual C++ 2015**运行时环境安装包。

     ```
     64/
        NSClient4CPP.lib
        ONSClient4CPP.dll
        ONSClient4CPP.pdb
     vc_redist.x64.exe
     ```

   - SDK_GUIDE.pdf： SDK 环境准备文档和 FAQ。
   - changelog：新版本发布解决的问题和引入的新特性列表。
   - inteface: 封装 PINVOKE 调用的代码，需要包含到项目代码中。

23.7.3.1.2 .NET SDK 配置说明

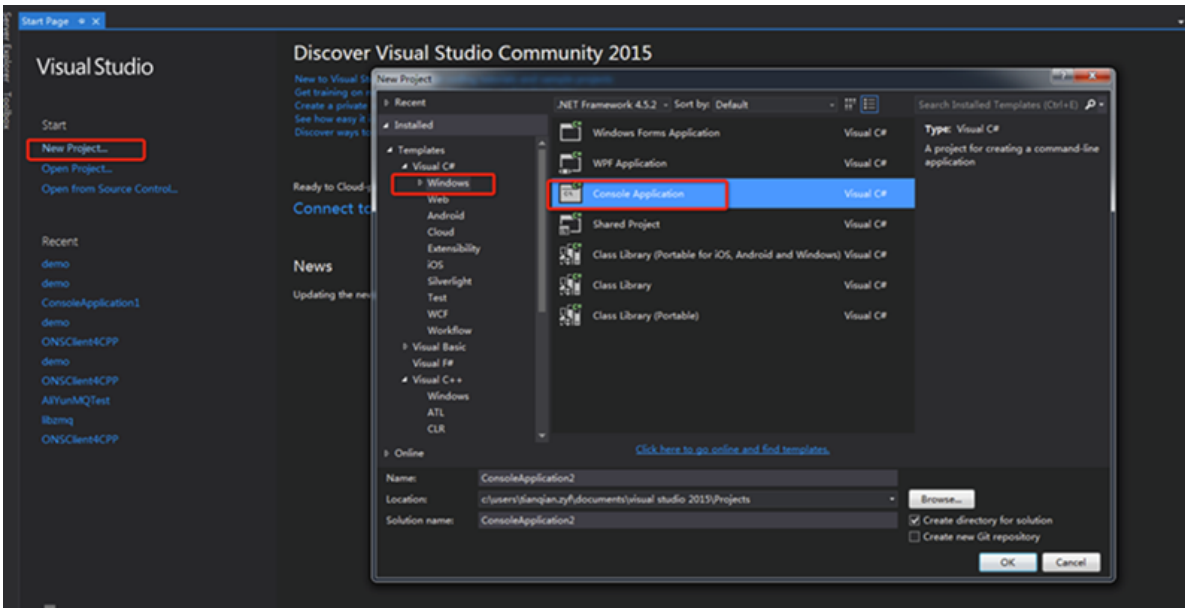
本文介绍如何在Windows中使用.NET SDK 。

操作步骤

Visual Studio 2015 环境下使用 NET SDK

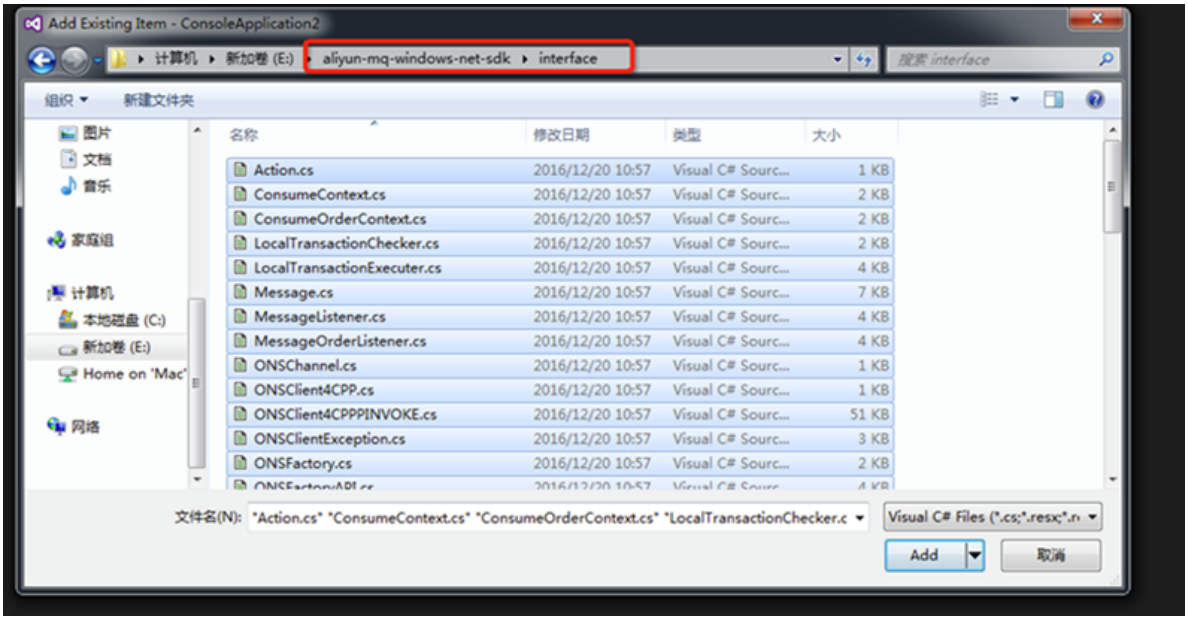
1. 使用 Visual Studio 2015 创建自己的项目。

   图 638: .NET SDK-1



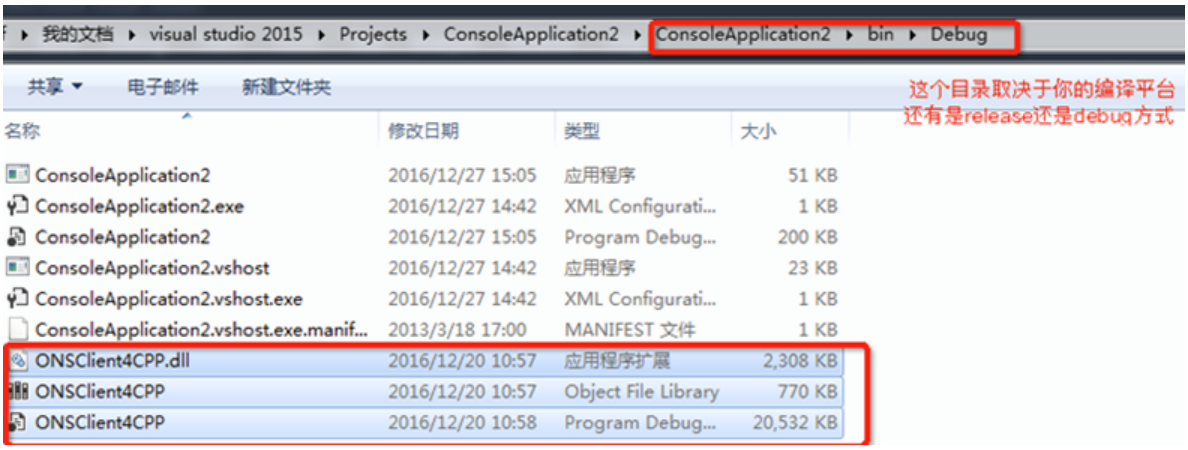
2. 右键单击项目选择**添加>现存在项** 将下载的 SDK 中的 **interface** 目录下的所有文件都添加进去。

图 639: .NET SDK-2



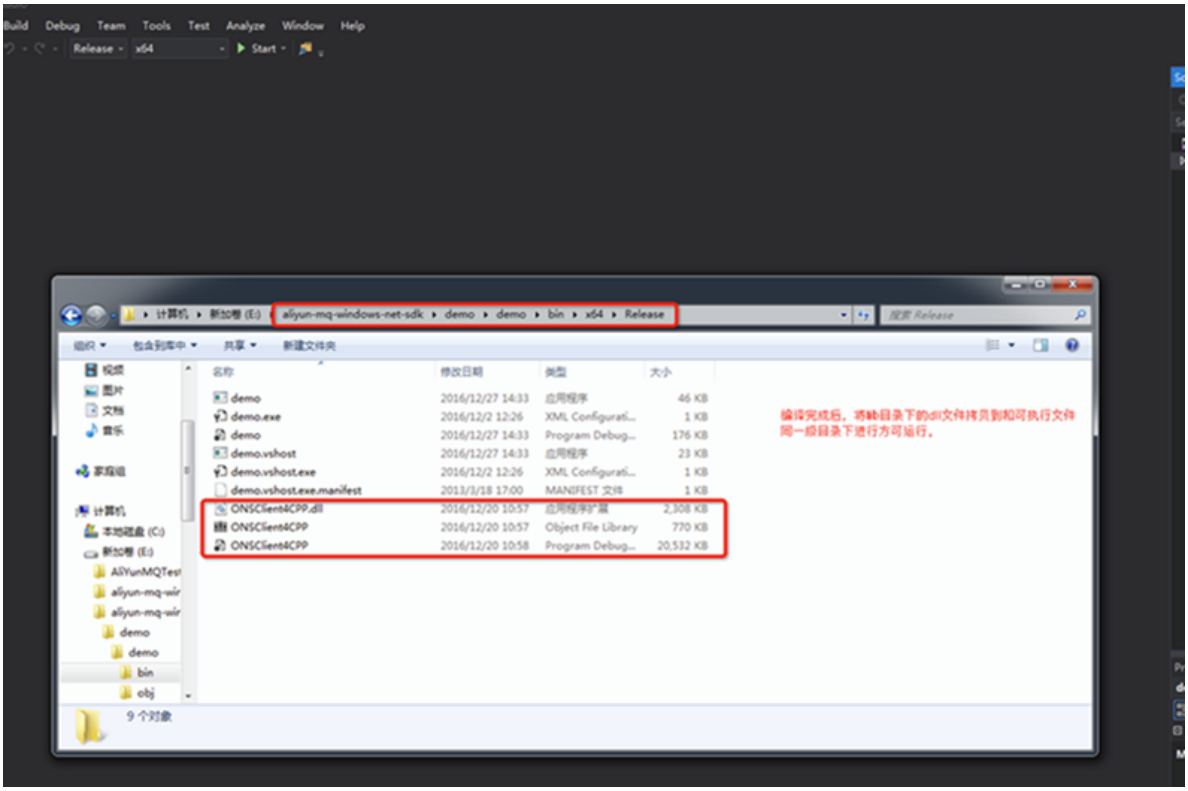
3. 编写测试程序，然后进行编译，最后将 SDK 下的 dll 放到和可执行文件同一级目录下，或者系统目录下即可运行。

图 640: .NET SDK-3



说明: SDK 提供了设置好的 Demo，直接打开工程进行编译即可。运行的时候将相关的 DLL 文件拷贝到和可执行文件同级目录下，如下图：

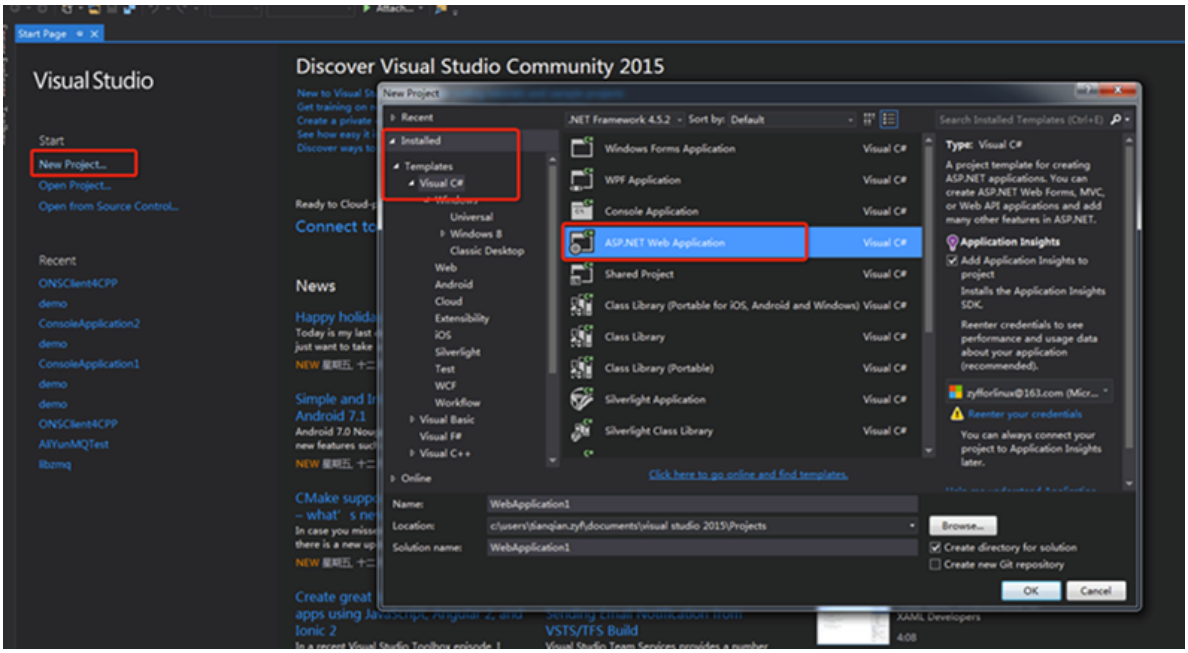
图 641: .NET SDK-4



Visual Studio 2015 配置 ASP.NET 使用 MQ SDK

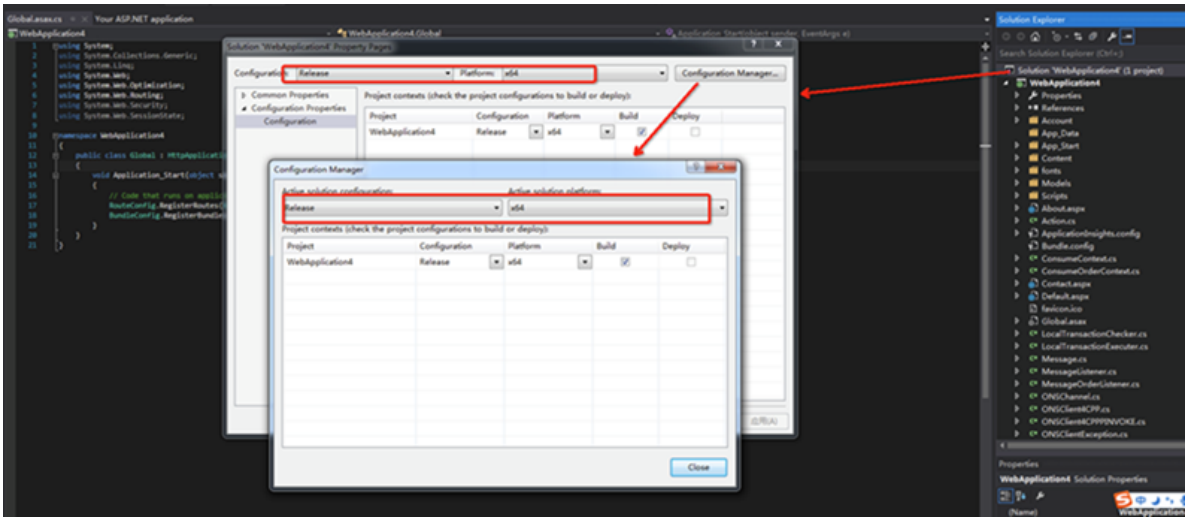
4. 使用 VS2015 创建一个 ASP.NET 的 **Web Forms** 项目。

图 642: .NET SDK-5



5. 右键单击项目选择属性，选择配置管理器。设置活动解决方案配置为 release；设置活动解决方案平台为 x64。

图 643: .NET SDK-6



6. 右键单击项目选择**添加>现存在项**，将下载的 SDK 中的 interface 目录下的所有文件都添加进去。

   请参考上文中配置普通的 .net 项目的步骤 2。

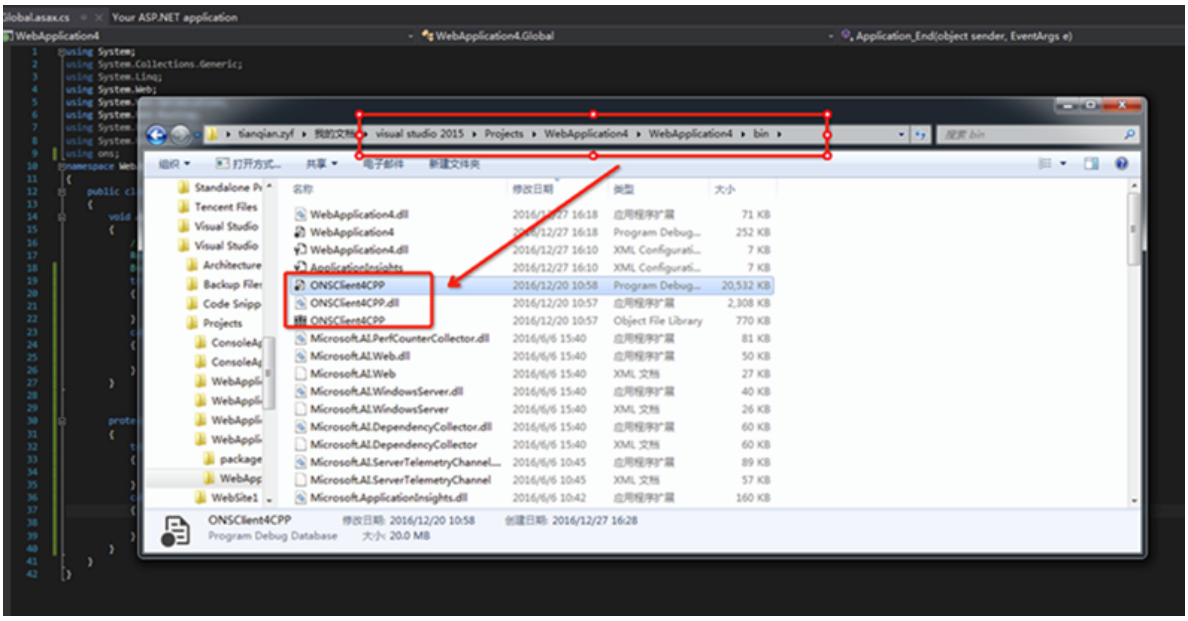
7. 在 **Global.asax.cs** 文件中添加启动和关闭 SDK 的代码。

说明: 建议将 SDK 的代码封装成一个单例类，这样可以避免因为作用域的问题被垃圾回收器回收。SDK 的 example 目录下提供了一个 Example.cs，实现了一个简单的单例实现。您需要把 Example.cs 包含到自己的项目中才能使用。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Optimization;
using System.Web.Routing;
using System.Web.Security;
using System.Web.SessionState;
using ons;    // 这个命名空间是SDK所在的命名空间
using test;   // 这是一个简单封装SDK的类所在命名空间，见SDK的example目录下的Example.cs
namespace WebApplication4
{
    public class Global : HttpApplication
    {
        void Application_Start(object sender, EventArgs e)
        {
            // Code that runs on application startup
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
            try
            {
                // 启动SDK的代码，下面是简单封装SDK后的代码
                OnscSharp.CreateProducer();
                OnscSharp.StartProducer();
            }
            catch (Exception ex)
            {
                    //处理异常
            }
        }
        protected void Application_End(object sender, EventArgs e)
        {
            try
            {
                // 关闭SDK的代码
               OnscSharp.ShutdownProducer();
            }
            catch (Exception ex)
            {
              // 处理异常
            }
        }
    }
}
```

8. 编写测试程序，然后进行编译。
9. 将 SDK 下的 dll 放到和可执行文件同一级目录下，或者系统目录下即可运行。

图 644: .NET SDK-7



10.依次点击**工具>选项>项目和解决方案>Web 项目**，然后勾选对网站和项目使用 **IIS Express** 的 64 位版。

图 645: .NET SDK-8

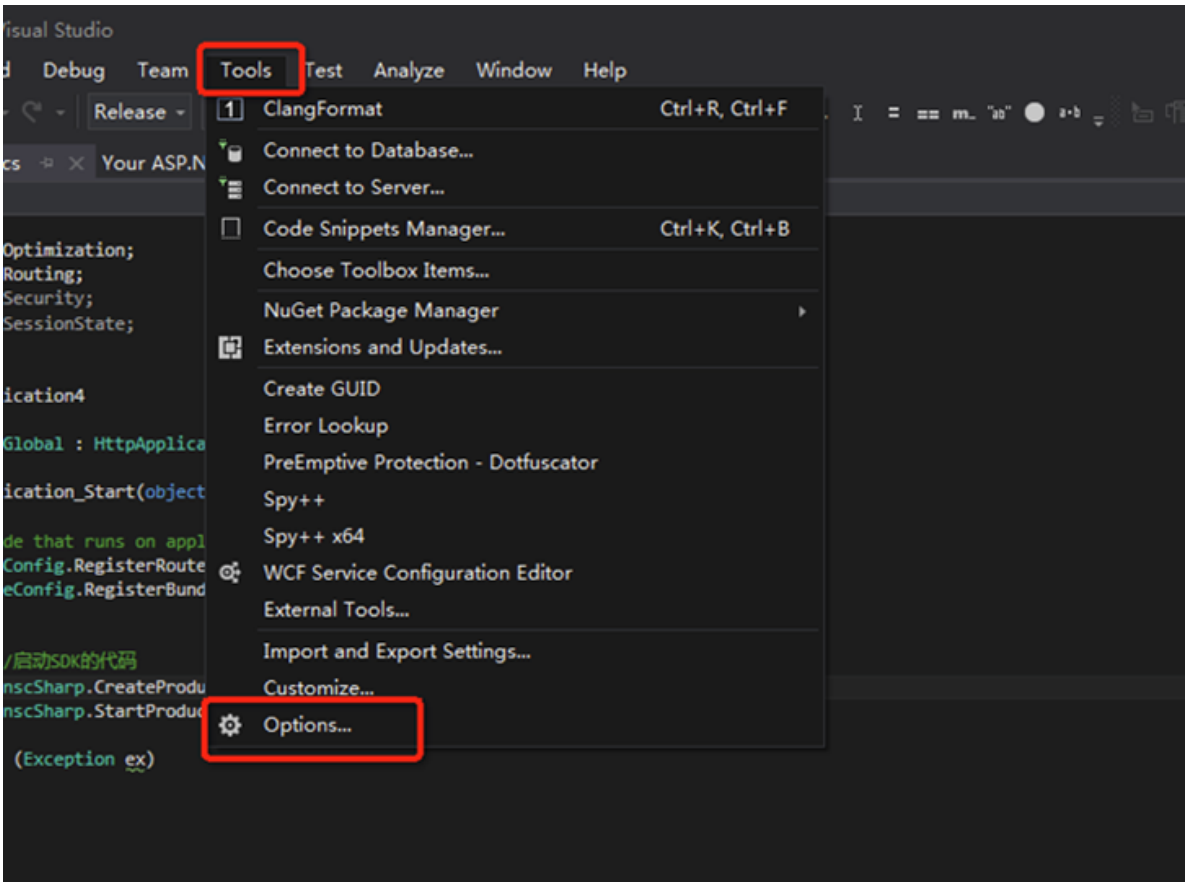
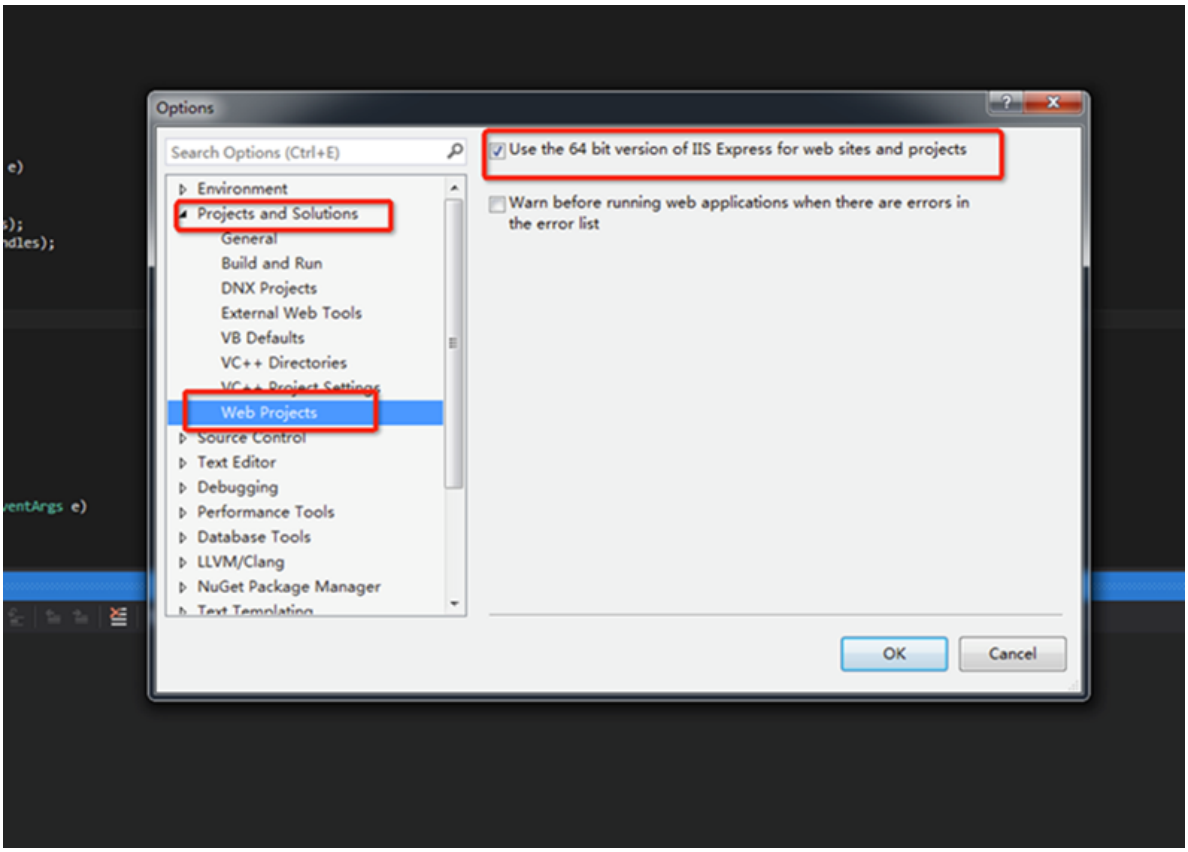


图 646: .NET SDK-9



23.7.3.2 发送普通消息

本文介绍发送普通消息的代码示例。

请参考以下示例代码进行消息发送。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using ons;

namespace ons
{
    class onscsharp
    {
        static void Main(string[] args)
        {
            //Producer创建和正常工作的参数，必须输入
            ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
             factoryInfo.setFactoryProperty(factoryInfo.getProducerIdName(), "PID_xxxx ");//您在MQ控制台申请的Producer ID
             factoryInfo.setFactoryProperty(factoryInfo.getPublishTopicsName(), "xxx");//您在MQ控制台申请的Topic
            factoryInfo.setFactoryProperty(factoryInfo.getMsgContentName(), "xxx");//msg content
              factoryInfo.setFactoryProperty(factoryInfo.getAccessKeyName(), "xxx");//AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
```

```
            factoryInfo.setFactoryProperty(factoryInfo.getSecretKeyName(),"xxx");//SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建

        //创建producer
        ONSFactory onsfactory = new ONSFactory();
        Producer pProducer = onsfactory.getInstance().createProducer(factoryInfo);

        //在发送消息前，必须调用start方法来启动Producer，只需调用一次即可
        pProducer.start();

        Message msg = new Message(
            //Message Topic
            factoryInfo.getPublishTopics(),
            //Message Tag
            "TagA",
            //Message Body
            factoryInfo.getMessageContent()
        );

        // 设置代表消息的业务关键属性，请尽可能全局唯一。
        // 以方便您在无法正常收到消息情况下，可通过 MQ 控制台查询消息并补发。
        // 注意：不设置也不会影响消息正常收发
        msg.setKey("ORDERID_100");

        //发送消息，只要不抛出异常，就代表发送成功
        try
        {
            SendResultONS sendResult = pProducer.send(msg);
        }
        catch(ONSClientException e)
        {
            //发送失败处理
        }

        // 在应用退出前，必须销毁Producer对象，否则会导致内存泄露等问题
        pProducer.shutdown();

    }
  }
}
```

23.7.3.3 发送定时消息

定时消息可以做到在指定时间之后才可被消费者消费，用于解决一些消息生产和消费有时间窗口要求的场景，或者通过消息触发定时任务的场景，类似于延迟队列。

代码示例如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using ons;

namespace ons
{
  class onscsharp
  {
```

```
    static void Main(string[] args)
    {
        //producer创建和正常工作的参数，必须输入
        ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
         factoryInfo.setFactoryProperty(factoryInfo.getProducerIdName(), "XXX ");//您在MQ控制台申请的Producer ID
          factoryInfo.setFactoryProperty(factoryInfo.getPublishTopicsName(), "XXX");//您在MQ控制台申请的Topic
        factoryInfo.setFactoryProperty(factoryInfo.getMsgContentName(), "XXX");//消息内容
         factoryInfo.setFactoryProperty(factoryInfo.getAccessKeyName(), "XXX");//AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
         factoryInfo.setFactoryProperty(factoryInfo.getSecretKeyName(),"XXX");//SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建

        //创建producer
        ONSFactory onsfactory = new ONSFactory();
        Producer pProducer = onsfactory.getInstance().createProducer(factoryInfo);

        //在发送消息前，必须调用start方法来启动Producer，只需调用一次即可
        pProducer.start();

        Message msg = new Message(
            //Message Topic
            factoryInfo.getPublishTopics(),
            //Message Tag
            "TagA",
            //Message Body
            factoryInfo.getMessageContent()
        );

        // 设置代表消息的业务关键属性，请尽可能全局唯一。
        // 以方便您在无法正常收到消息情况下，可通过MQ Console查询消息并补发。
        // 注意：不设置也不会影响消息正常收发
        msg.setKey("ORDERID_100");

          // deliver time 单位 ms，指定一个时刻，在这个时刻之后才能被消费，这个例子表示3s后才能被消费
        long deliverTime = 获取系统当前时间(ms) + 3000;
        msg.setStartDeliverTime(deliverTime);

        //发送消息，只要不抛出异常，就代表发送成功
        try
        {
            SendResultONS sendResult = pProducer.send(msg);
        }
        catch(ONSClientException e)
        {
            //发送失败处理
        }

        // 在应用退出前，必须销毁Producer对象，否则会导致内存泄露等问题
        pProducer.shutdown();

    }
  }
}
```

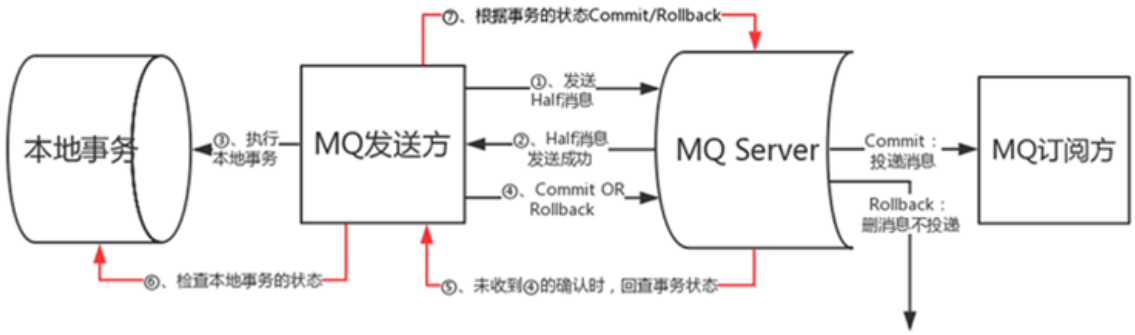
23.7.3.4 发送分布式事务消息

本文介绍如何发送分布式事务消息。

背景信息

MQ 事务消息交互流程如下。

图 647: 事务消息流程图



操作步骤

1. 发送半消息（Half Message）及执行本地事务。

```
using System;
 using System.Collections.Generic;
 using System.Linq;
 using System.Text;
 using System.Runtime.InteropServices;
 using ons;

 namespace ons
 {
 public class MyLocalTransactionExecuter : LocalTransactionExecuter
 {
    public MyLocalTransactionExecuter()
    {
    }

    ~MyLocalTransactionExecuter()
    {
    }
    public override TransactionStatus execute(ref Message value)
    {
                    Console.WriteLine("execute topic: {0}, tag:{1}, key:{2}, msgId:{3},msgbody:{4},
userProperty:{5}",
                value.getTopic(), value.getTag(), value.getKey(), value.getMsgID(), value.getBody(),
value.getUserProperty("VincentNoUser"));

            // 消息ID(有可能消息体一样，但消息ID不一样，当前消息ID在控制台控制不可能查询)
            string msgId = value.getMsgID();
            // 消息体内容进行crc32, 也可以使用其他的如MD5
            // 消息ID和crc32id主要是用来防止消息重复
```

```
            // 如果业务本身是幂等的，可以忽略，否则需要利用msgId或crc32Id来做幂等
          // 如果要求消息绝对不重复，推荐做法是对消息体body使用crc32或md5来防止重复消息。
            TransactionStatus transactionStatus = TransactionStatus.Unknow;
            try {
                boolean isCommit = 本地事务执行结果;
                if (isCommit) {
                    // 本地事务成功、提交消息
                    transactionStatus = TransactionStatus.CommitTransaction;
                } else {
                    // 本地事务失败、回滚消息
                    transactionStatus = TransactionStatus.RollbackTransaction;
                }
            } catch (Exception e) {
                //exception handle
            }
            return transactionStatus；
    }
}
class onscsharp
{

    static void Main(string[] args)
    {
        ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
         factoryInfo.setFactoryProperty(factoryInfo.getProducerIdName(), "");//您在MQ控制台申请的Producer ID
          factoryInfo.setFactoryProperty(factoryInfo.getPublishTopicsName(), "");// 您在MQ控制台申请的Topic
        factoryInfo.setFactoryProperty(factoryInfo.getMsgContentName(), "");//message body
       factoryInfo.setFactoryProperty(factoryInfo.getAccessKeyName(), "");//AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
        factoryInfo.setFactoryProperty(factoryInfo.getSecretKeyName(), "");//SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建

        //create transaction producer
        ONSFactory onsfactory = new ONSFactory();
        LocalTransactionChecker myChecker = new MyLocalTransactionChecker();
                TransactionProducer pProducer = onsfactory.getInstance().createTransactionProducer(factoryInfo,ref myChecker);

        //在发送消息前，必须调用start方法来启动Producer，只需调用一次即可，启动之后可以多线程并发发送消息
        pProducer.start();

            Message msg = new Message(
            //Message Topic
            factoryInfo.getPublishTopics(),
            //Message Tag
            "TagA",
            //Message Body
            factoryInfo.getMessageContent()
        );

        // 设置代表消息的业务关键属性，请尽可能全局唯一。
        // 以方便您在无法正常收到消息情况下，可通过MQ K控制台查询消息并补发。
        // 注意：不设置也不会影响消息正常收发
        msg.setKey("ORDERID_100");

        //发送消息，只要不抛出异常，就代表发送成功
        try
```

```
        {
            LocalTransactionExecuter myExecuter = new MyLocalTransactionExecuter();
            SendResultONS sendResult = pProducer.send(msg, ref myExecuter);
        }
        catch(ONSClientException e)
        {
            Console.WriteLine("\nexception of sendmsg:{0}",e.what() );
        }

        // 在应用退出前，必须销毁Producer对象，否则会导致内存泄露等问题；
        // shutdown之后不能重新start此producer
        pProducer.shutdown();
    }
}
```

2. 提交事务消息状态。

   当本地事务执行完成（执行成功或执行失败），需要通知服务器当前消息的事务状态。通知方式有以下两种：

   - 执行本地事务完成后提交。
   - 执行本地事务一直没提交状态，等待服务器回查消息的事务状态。

   事务状态有以下三种：

   - TransactionStatus.CommitTransaction 提交事务，允许订阅方消费该消息。
   - TransactionStatus.RollbackTransaction 回滚事务，消息将被丢弃不允许消费。
   - TransactionStatus.Unknow 无法判断状态，期待 MQ Broker 向发送方再次询问该消息对应的本地事务的状态。

```
public class MyLocalTransactionChecker : LocalTransactionChecker
{
    public MyLocalTransactionChecker()
    {
    }

    ~MyLocalTransactionChecker()
    {
    }

    public override TransactionStatus check(ref Message value)
    {
                    Console.WriteLine("check topic: {0}, tag:{1}, key:{2}, msgId:{3},msgbody:{4},
userProperty:{5}",
            value.getTopic(), value.getTag(), value.getKey(), value.getMsgID(), value.getBody(),
value.getUserProperty("VincentNoUser"));
            // 消息ID(有可能消息体一样，但消息ID不一样，当前消息ID在控制台控制不可能查询)
            string msgId = value.getMsgID();
            // 消息体内容进行crc32, 也可以使用其他的如MD5
            // 消息ID和crc32id主要是用来防止消息重复
            // 如果业务本身是幂等的，可以忽略，否则需要利用msgId或crc32Id来做幂等
                // 如果要求消息绝对不重复，推荐做法是对消息体body使用crc32或md5来防止重复
消息。
            TransactionStatus transactionStatus = TransactionStatus.Unknow;
            try {
```

```
            boolean isCommit = 本地事务执行结果;
            if (isCommit) {
                // 本地事务成功、提交消息
                transactionStatus = TransactionStatus.CommitTransaction;
            } else {
                // 本地事务失败、回滚消息
                transactionStatus = TransactionStatus.RollbackTransaction;
            }
        } catch (Exception e) {
            //exception handle
        }
        return transactionStatus；
    }
    }
```

后续操作

事务回查机制说明

1. 发送事务消息为什么必须要实现check机制？

   当步骤1发送半消息完成，但本地事务返回状态为 TransactionStatus.Unknow 时，亦或是应用退出导致本地事务未提交任何状态时，从 MQ Broker 的角度看，这条半状态的消息的状态是未知的，因此 MQ Broker 会定期要求发送方能 check 该半状态消息，并上报其最终状态。

2. Check 被回调时，业务逻辑都需要做些什么？

   MQ 事务消息的 check 方法里面，应该写一些检查事务一致性的逻辑。MQ 发送事务消息时需要实现 LocalTransactionChecker 接口，用来处理 MQ Broker主动发起的本地事务状态回查请求；因此在事务消息的 check 方法中，需要完成两件事情：

   a. 检查该半消息对应的本地事务的状态（commited or rollback）。

   b. 向 MQ Broker 提交该半消息本地事务的状态。

3. 本地事务的不同状态对半消息的影响？

   - TransactionStatus.CommitTransaction 提交事务，允许订阅方消费该消息。
   - TransactionStatus.RollbackTransaction 回滚事务，消息将被丢弃不允许消费。
   - TransactionStatus.Unknow 无法判断状态，期待 MQ Broker 向发送方再次询问该消息对应的本地事务的状态。

   具体代码请参见 MyLocalTransactionChecker 的实现。

23.7.3.5 集群方式订阅消息

集群订阅即某个消费者集群只消费指定的Topic，而不是消费所有Topic。

示例代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using ons;

namespace ons
{

   //pushConsumer拉取到消息后，会主动调用该实例的consume 函数
   public class MyMsgListener : MessageListener
   {
      public MyMsgListener()
      {
      }

      ~MyMsgListener()
      {
      }

      public override  Action  consume(ref Message value)
      {
         /*
               所有中文编码相关问题都在sdk压缩包包含的文档里做了说明，请仔细阅读
        */
         return ons.Action.CommitMessage;
      }
   }

   class onscsharp
   {
        static void Main(string[] args)
      {
         //pushConsumer创建和工作需要的参数，必须输入
         ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
          factoryInfo.setFactoryProperty(factoryInfo.getConsumerIdName(), "XXX");//您在MQ控制台申请的Consumer ID
           factoryInfo.setFactoryProperty(factoryInfo.getPublishTopicsName(), "XXX");//您在MQ控制台申请的Topic
          factoryInfo.setFactoryProperty(factoryInfo.getAccessKeyName(),"xx");//AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
         factoryInfo.setFactoryProperty(factoryInfo.getSecretKeyName(), "xxxx");//SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建

         //create consumer
         ONSFactory onsfactory = new ONSFactory();
         PushConsumer pConsumer = onsfactory.getInstance().createPushConsumer(factoryInfo);

         //register msg listener and subscribe msg topic
         MessageListener msgListener = new MyMsgListener();
         pConsumer.subscribe(factoryInfo.getPublishTopics(), "*",  ref msgListener);

         //start consumer
         pConsumer.start();
               //consumer启动后，会自动拉取消息，拉取到消息后，会自动调用MyMsgListener实例的consume函数；

          //确定消费完成后，调用shutdown函数；在应用退出前，必须销毁Consumer 对象，否则会导致内存泄露等问题
         pConsumer.shutdown();
```

```
    }
  }
}
```

23.8 MQTT接入

本节介绍MQTT的接入简介、名词、接入准备、各种接入方式的示例以及MQTT数据查询和常见问题。

MQ 针对用户在移动互联网以及物联网领域的存在的特殊消息传输需求，开放了 MQTT 协议的完整支持。

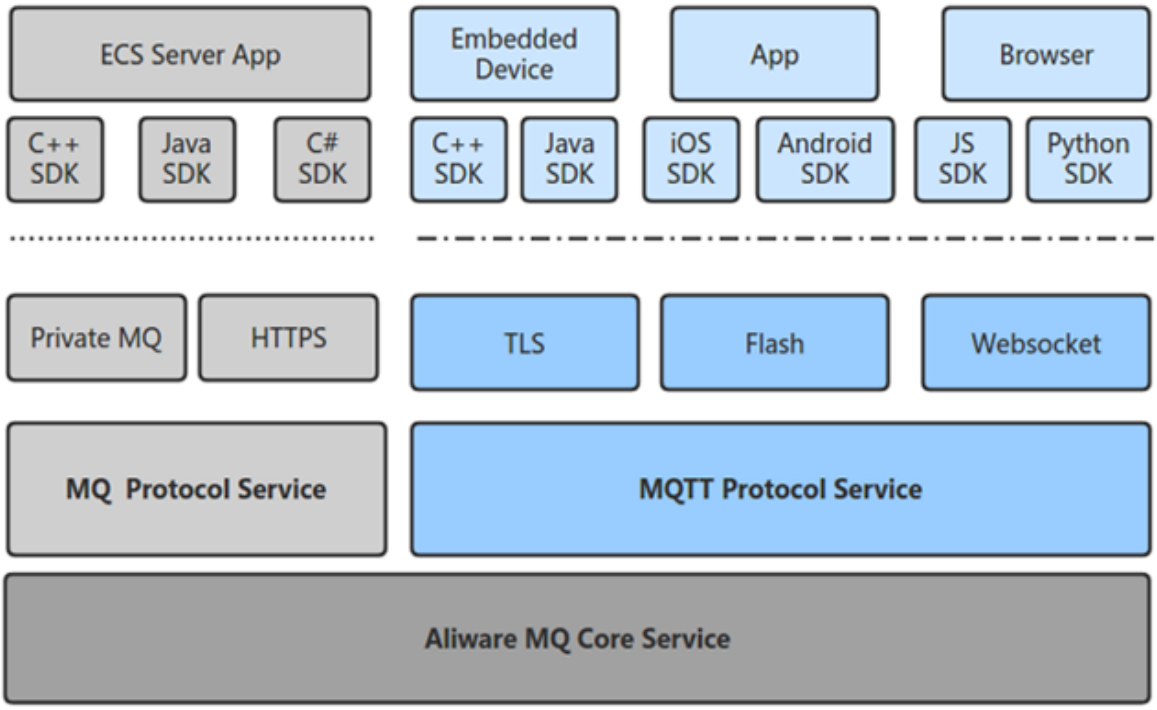
MQTT（Message Queuing Telemetry Transport，消息队列遥测传输）是一种轻量的，基于发布订阅模型的即时通讯协议。该协议设计开放，协议简单，平台支持丰富，几乎可以把所有联网物品和外部连接起来，因此在移动互联网和物联网领域拥有众多优势。

该协议的特点包括：

- 使用发布/订阅消息模式，提供一对多的消息分发，解除了应用程序之间的耦合。
- 对负载内容屏蔽的消息传输。
- 使用 TCP/IP 提供基础的网络连接。
- 有三种级别的消息传递服务。
- 小型传输，开销很小（固定长度的头部是2字节），协议交换最小化，以降低网络流量。

如上文所述，MQ 完整支持 MQTT3.1.1协议，通过在 MQ Core 基础上增加 MQTT 协议网关的方式对互联网上的客户端提供服务。整个系统架构如下图所示：

图 648: MQTT 系统结构图



其中，MQ Core Service 负责底层的消息存储和分发，上层支持 MQ 私有协议服务器以及 MQTT 协议网关服务器。MQTT 网关负责对用户的 MQTT 客户端提供服务，同时负责 MQTT 协议和后端 MQ 私有协议的转换。主要工作如下：

1. 提供 MQTT 服务，将用户的 MQTT 消息转换成后端 MQ 消息存储到 MQ 核心系统中，实现数据上行采集。
2. 接收来自 MQ 的消息，并将消息转换成对应的 MQTT 消息，推送给 MQTT 客户端，实现数据下行推送。

客户端分布：

1. 使用 MQTT 协议的客户端作为移动端接入 MQ，一般分布在公网环境，比如嵌入式设备、移动手机、平板、浏览器之类的平台上。
2. 使用 MQ 协议的客户端一般作为业务上的服务端接入 MQ，应该部署在阿里云的 ECS 环境。

23.8.1 应用场景和优势

MQTT 作为移动互联网以及物联网领域的主流协议，应用场景丰富。

应用场景

目前 MQ 提供的 MQTT 服务已经在各个领域有所应用，覆盖了直播互动、车联网、金融支付、即时聊天等多个场景。同时，MQTT 服务端采用分布式理念进行设计，系统无单点瓶颈，各个组件都可以无限水平扩展，保证容量可以跟随用户在线使用量进行调整。

图 649: MQTT应用场景



优势

相比其他移动端消息服务，基于 Aliware MQ 的 MQTT 服务具有以下优势：

- MQ 支持标准的 MQTT 协议，应用方无技术捆绑，可以无缝迁移。
- 可以支持移动端 MQTT 消息和服务端 MQ 消息的相互流转，实现服务端和移动端的双向打通。
- 支持设备级权限控制，并支持 SSL/TLS 加密通信，数据传输更安全可靠。
- 兼容任何支持 MQTT 3.1.1 协议的 SDK，覆盖绝大多数移动端开发语言和平台。

23.8.2 名词解释

本文介绍 MQTT 协议的一些基本术语以及在阿里云环境使用 MQTT 涉及到的主要概念和术语，包括资源类、权限类和网络类名词。

资源类

Instance 用户申请购买 MQTT 服务的实体单元，每个 MQTT 实例都对应一个全局唯一的服务接入点URL。使用 MQTT 前都需要在对应的 Region 申请一个实例，并使用对应的接入点来访问服务。如何申请 MQTT 实例请参考申请资源。

Parent Topic MQTT 协议基于 Pub/Sub 模型，因此任何消息都属于一个 Topic。根据 MQTT 协议，Topic 存在多级，定义第一级 Topic 为父 Topic（Parent Topic），使用 MQTT 前，该 Parent Topic 需要先在 MQ 控制台申请。

Subtopic MQTT 的二级 Topic，甚至三级 Topic 都是父 Topic 下的子类。使用时，直接在代码里设置，无需申请。

Client ID MQTT 的 Client ID 是每个客户端的唯一标识，要求全局唯一，使用相同的 Client ID 连接 MQTT 服务会被拒绝。

Client ID 由两部分组成，组织形式为 GroupID@@@DeviceID。

- **Group ID**: 用于指定一组逻辑功能完全一致的节点共用的组名，代表一类相同功能的设备。Group ID 需要在 MQ 控制台申请方可使用，申请链接参考申请资源。
- **Device ID**: 每个设备独一无二的标识，由业务方自己指定。需要保证全局唯一，例如每个传感器设备的序列号。

权限类

Username 使用 MQTT 客户端收发消息时，MQ 会根据用户设置的 Username 和 Password 来进行鉴权。鉴权逻辑遵循阿里云统一的权限规范。此处 Username 设置为阿里云的 AccessKey 即可。

Password MQ 要求用户将 GroupID 作为签名字段，SecretKey 作为秘钥，使用 HmacSHA1 算法计算签名字符串，并将签名字符串设置到 Password 参数中用于鉴权。关于 Password 的签名可以参考签名计算文档。具体的代码实现可以参考各个语言版本的Demo程序。

网络类

ServerUrl MQ 提供的 MQTT 服务的接入点 URL，都是公网 URL，目前 MQTT 的接入除了支持标准协议的1883端口，同时还支持加密 SSL，WebSocket，Flash 等方式。接入点URL是在申请实例之后自动分配，请妥善保管。申请实例请参考申请资源章节。

协议相关

cleanSession cleanSession 标志是 MQTT 协议中对一个客户端建立 TCP 连接后是否关心之前状态的定义。具体语义如下：

- cleanSession=true：客户端再次上线时，将不再关心之前所有的订阅关系以及离线消息。
- cleanSession=false：客户端再次上线时，还需要处理之前的离线消息，而之前的订阅关系也会持续生效。

注意:

- MQTT 要求每个客户端每次连接时的 cleanSession 标志必须固定，不允许动态变化，否则会导致离线消息的判断有误。
- Aliware MQTT 目前对外 QoS2 消息不支持非 cleanSession ，如果客户端以 QoS2 方式订阅消息，即使设置 cleanSession=false 也不会生效。
- P2P 消息的 cleanSession 判断以发送方客户端的配置为准。

23.8.3 MQTT接入准备

本节介绍MQTT的接入准备工作，包括环境准备、资源申请、签名计算以及客户端上下线等。

23.8.3.1 准备环境

MQTT 接入需要首先完成 MQTT 客户端适配、服务接入配置等操作。

客户端适配

MQ 提供的 MQTT 服务严格遵循 MQTT3.1.1 协议设计，理论上能够适配所有的 MQTT 客户端，但不排除部分客户端存在细节上的兼容性问题。针对 MQTT 用户常用的平台，推荐对应的三方包如下：

使用平台	推荐的第三方SDK	相关链接
Java	Eclipse Paho SDK	*http://www.eclipse.org/paho/clients/java/*

使用平台	推荐的第三方SDK	相关链接
iOS	MQTT-Client-Framework	*https://github.com/ckrey/MQTT-Client-Framework*
Android	Eclipse Paho SDK	*https://github.com/eclipse/paho.mqtt.android*
JavaScript	Eclipse Paho JavaScript	*http://www.eclipse.org/paho/clients/js/*

其他语言的客户端 SDK 如 PHP、Python 等暂时没有提供测试。如有需要可以访问 *http://www.eclipse.org/paho/downloads.php* 进行下载。

服务接入配置

MQ目前开放的MQTT服务除了支持标准的 MQTT 协议还支持如 WebSocket、Flash。

标准协议端口	Websocket端口	Flash端口
1883	80	843

23.8.3.2 申请资源

使用 MQTT 收发消息需要首先在控制台申请 Topic 和 MQTT Group ID 资源，否则服务端会拒绝非法的 Client ID 的连接。

操作步骤

1. 申请MQTT Topic。

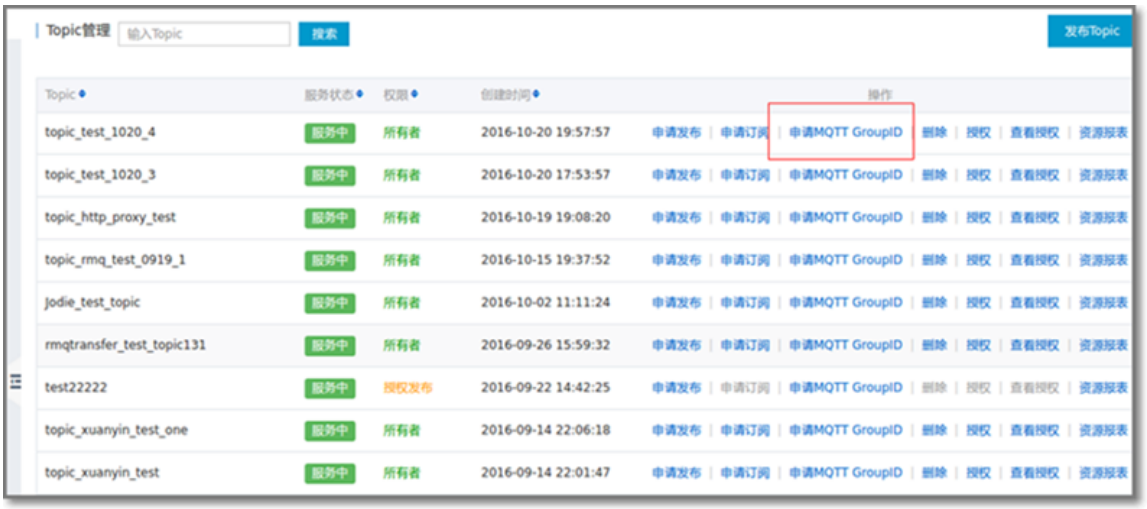
   使用 MQTT 协议收发消息，需要申请 MQTT 的 Parent Topic（一级父 Topic），多级子 Topic 不需要申请，直接在代码中使用即可。

   - 如果没有申请过Parent Topic，请参考申请资源。
   - 如果已经申请过Parent Topic，请直接进入下一步。

2. 申请 MQTT Group ID。

   a) 进入 MQ 控制台 Topic 管理页面，在右侧工具按钮栏单击申**请 MQTT Group ID**。

   图 650: MQTT Group ID-1



b) 在弹出框中输入需要申请的 Group ID。

说明: MQTT Group ID 的命名必须以“GID_”开头，长度限制在7-32个字节，内容仅限于数字和字母。如果输入不合法字符或者长度不符合，将显示参数校验错误。

图 651: MQTT Group ID-2



c) 输入正确的 Group ID，单击**确定**，将完成申请并跳转到 MQTT Group ID 管理页面。该页面显示当前 Region 下当前账号拥有的 MQTT Group ID。

图 652: MQTT Group ID-3



说明:

- 申请 MQTT Group ID 必须拥有特定 Topic 的读写权限。
- 如果 Group ID 不再使用，请及时删除。

23.8.3.3 签名计算

使用 MQTT 收发消息，服务端需要对客户端的身份进行权限校验，因此客户端请求中都需要带上签名以便比对身份。

MQTT SDK 访问消息服务器

MQTT 客户端实际连接 MQTT 消息服务器时，在 connect 报文中需要上传 username 和 password。其中 username 就是 AccessKey，password 则是将 Group ID 作为待签名字符串，用 SecretKey 作为秘钥计算得到的签名。

比如客户端的 Client ID 是 GID_AAA@@@BBB001。

此时待签名字符串就是取 Client ID 的前缀，即 Group ID，“GID_AAA”。

然后用 SecretKey 作为秘钥，使用 HmacSHA1 方法对上面的待签名字符串做签名得到 password 签名字符串，即“eqweq+adwe23fssf”。

Hmac 的算法实现，各个语言都有现成的函数库，请自行搜索。

23.8.3.4 获取客户端上下线事件

本文介绍如何在服务端获取 MQTT 客户端的上下线动作等状态信息。

背景信息

在某些特定场景中，服务端需要了解MQTT客户端的上下线信息，例如：

- 服务端需要关注特定的 ClientID 什么时候上线，什么时候下线；
- 服务端需要根据特定 ClientID 的在线状态做逻辑切换；

- 服务端需要在客户端上线或者下线时触发一些预定义的动作。

MQTT 服务端在客户端上线和下线事件触发时，会向后端 MQ 推送一条上下线消息。用户的服务端应用一般部署在阿里云的服务器上，可以通过订阅这条 MQ 消息来获取所有客户端的上下线动作。

其中，上下线动作的这个事件对应的 Topic 命名规范是：**GroupID+后缀_MQTT**

例如您需要关注 GroupID_AAA@@@XXXX 类型的 MQTT 客户端，那么对应的事件 Topic 就是 GroupID_AAA_MQTT。

操作步骤

1. 申请事件Topic。根据上文原理介绍，用户关注哪些 GroupID 分组的设备，就申请对应的事件 Topic，申请 Topic 方法请参考 申请资源。
2. 服务端订阅消息。使用 MQ 的 SDK 订阅**步骤1**中申请的 Topic，即可收到关注的 MQTT 客户端上下线事件。MQ 的接收程序请参考 订阅消息。

   其中，事件类型放在 MQ 的 Tag 中，代表是上线还是下线，具体格式为：`MQTag：connect/disconnect/tcpclean`

   - connect 事件代表客户端上线动作。
   - disconnect 事件代表客户端主动断开连接。按照 MQTT 协议，客户端主动断开 TCP 连接之前应该发送 disconnect 报文，服务端在收到 disconnect 报文后触发该类型消息。如果某些客户端 SDK 没有按照协议发送 disconnect 报文，相应地会无法收到该消息。
   - tcpclean 事件代表实际的 TCP 连接断开。无论客户端是否显式发送过 disconnect 报文，只要当前 TCP 连接断开就会触发 tcpclean 事件。

     注意: tcpclean 消息代表客户端网络层连接的真实断开。对应的，disconnect 消息仅仅代表客户端是主动发送了下线报文。受限于客户端的实现，有时候客户端异常退出会导致 disconnect 消息并没有正常发送。因此判断客户端下线请使用 tcpclean 事件。

   数据内容为 JSON 类型，相关的 Key 如下：

   - ClientId 代表具体设备；
   - time 代表本次事件的时间；
   - eventType 代表事件类型，供 MQTT 客户端区分事件类型；
   - channelId 代表每个 TCP 连接的唯一标识；

- eventIndex 代表事件产生的递增顺序。

```
clientId：GID_XXX@@@XXXXX
time:1212121212
eventType:connect/disconnect/tcpclean
channelId:2b9b1281046046faafe5e0b458e4f553
eventIndex:1493772130197
```

后续操作

判断客户端当前是否在线不能仅仅根据收到的最后一条消息的状态，而需要结合上下线消息的前后关联来判断。具体判断规则如下：

- 同一个 clientId 的客户端，产生上下线事件的先后顺序以 eventIndex 为准。越大则越新，不能以时间来做判断（粒度太大）。
- 同一个 clientId 的客户端，可能存在多次闪断，因此，当收到下线消息时，一定要根据 channelId 字段判断是否是当前的 TCP 连接。简而言之，下线消息只能覆盖相同 channelId 的状态，如果channelId 不一样，尽管 eventIndex 较新，也不能覆盖。

获取状态的 Demo 如下：

```
public static void main(String[] args) throws InterruptedException {
        /**
         * 设置阿里云的AccessKey，用于鉴权
         */
        final String acessKey ="XXXXXX";
        /**
         * 设置阿里云的SecretKey，用于鉴权
         */
        final String secretKey ="XXXXXXX";
        /**
         *  上述步骤中涉及的事件Topic，需要先在MQ控制台里申请
         */
        final String topic ="GID_XXX_MQTT";
        /**
         * ConsumerID,需要先在MQ控制台里申请
         */
        final String consumerID ="CID_XXXX";
        Properties properties =new Properties();
          //PropertyKeyConst.ONSAddr地址请根据实际情况进行输入：
           "http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal");//此处以公共云生产环境为例，专有云环境请参考交付文档
        properties.put(PropertyKeyConst.ConsumerId, consumerID);
        properties.put(PropertyKeyConst.AccessKey, acessKey);
        properties.put(PropertyKeyConst.SecretKey, secretKey);
        Consumer consumer =ONSFactory.createConsumer(properties);
        /**
         * 处理收到的事件，根据Tag区分是上线还是下线，body是个JSON字符串
         */
        consumer.subscribe(topic, "*", new MessageListener() {
            public Action consume(Message message, ConsumeContext consumeContext) {
                String event = message.message.getTag();
                  String event = message.getTag();
```

```
            if(event.equals("connect")){
                // this is connect event
            }else if(event.equals("disconnect")){
                // this is client disconnect event
            }eles if(event.equals("tcpclean")){
                // this is tcp disconnect
            }
            String body = new String(message.getBody());
            JSONObject object = JSON.parseObject(body);
            String clientId = object.getString("clientId");
            long time  =object.getLong("time");
            return Action.CommitMessage;
        }
    });
    consumer.start();
    System.out.println("[Case Normal Consumer Init]   Ok");
    Thread.sleep(Integer.MAX_VALUE);
    consumer.shutdown();
    System.exit(0);
}
```

23.8.4 Java接入示例

本节介绍Java接入MQTT示例，包括MQ 客户端收发 MQTT 消息、MQTT 客户端收发 MQTT 消息和SSL 方式接入 MQTT 客户端。

23.8.4.1 MQTT 客户端收发 MQTT 消息

本文介绍如何使用 MQTT 客户端收发 MQTT 消息，并给出示例代码供前期开发测试参考，包括资源申请、环境准备、示例代码、注意事项等。

前提条件

资源申请

使用 MQ 提供的 MQTT 服务，首先需要核实应用中使用的 Topic 资源是否已经申请。如果没有，请先去控制台申请 Topic，Group ID 等资源。

申请资源时需要根据需求选择对应的 Region，例如 MQTT 需要使用华北2的接入点，那么 Topic 等资源就在华北2 申请，资源申请具体请参见申请资源。

说明: MQTT 使用的多级子 Topic 不需要申请，代码里直接使用即可，没有限制。

背景信息

说明: 本文给出的示例均基于 Eclipse Paho Java SDK 实现，SDK 下载请参见*MQTT*接入准备。如使用其他第三方的客户端，请适当修改。

操作步骤

1. 准备环境。

使用 MQTT 协议来收发消息，需要根据应用平台选择合适的客户端。本示例运行在 Java 平台，使用 Eclipse Paho Java SDK 构建。首先引入 Maven 依赖，POM 文件配置如下：

```
<dependencies>
   <dependency>
      <groupId>org.eclipse.paho</groupId>
      <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
      <version>1.0.2</version>
   </dependency>
</dependencies>
<repositories>
      <repository>
         <id>Eclipse Paho Repo</id>
         <url>https://repo.eclipse.org/content/repositories/paho-releases/</url>
      </repository>
      <repository>
         <id>snapshots-repo</id>
         <url>https://oss.sonatype.org/content/repositories/snapshots</url>
         <releases>
            <enabled>false</enabled>
         </releases>
         <snapshots>
            <enabled>true</enabled>
         </snapshots>
      </repository>
 </repositories>
```

2. MQTT 发送消息。

本段示例代码演示如何使用 MQTT 客户端发送普通消息和 P2P 的点对点消息，其中用到的工具 MacSignature 参考下文。

```
public class MQTTSendMsg {
   public static void main(String[] args) throws IOException {
      /**
     * 设置当前用户私有的MQTT的接入点。例如此处示意使用XXX，实际使用请替换用户自己的接入点。接入点的获取方法是，在控制台申请MQTT实例，每个实例都会分配一个接入点域名。
       */
      final String broker ="tcp://XXXX.mqtt.aliyuncs.com:1883";
      /**
       * 设置阿里云的AccessKey，用于鉴权
       */
      final String acessKey ="XXXXXX";
      /**
       * 设置阿里云的SecretKey，用于鉴权
       */
      final String secretKey ="XXXXXXX";
      /**
       * 发消息使用的一级Topic，需要先在MQ控制台里申请
       */
      final String topic ="XXXX";
      /**
       * MQTT的ClientID，一般由两部分组成，GroupID@@@DeviceID
       * 其中GroupID在MQ控制台里申请
     * DeviceID由应用方设置，可能是设备编号等，需要唯一，否则服务端拒绝重复的ClientID连接
       */
```

```
        final String clientId ="GID_XXX@@@ClientID_XXXX";
        String sign;
        MemoryPersistence persistence = new MemoryPersistence();
        try {
            final MqttClient sampleClient = new MqttClient(broker, clientId, persistence);
            final MqttConnectOptions connOpts = new MqttConnectOptions();
            System.out.println("Connecting to broker: " + broker);
            /**
             * 计算签名，将签名作为MQTT的password。
                     *  签名的计算方法，参考工具类MacSignature，第一个参数是ClientID的前半部
分，即GroupID
             * 第二个参数阿里云的SecretKey
             */
            sign = MacSignature.macSignature(clientId.split("@@@")[0], secretKey);
            connOpts.setUserName(acessKey);
            connOpts.setServerURIs(new String[] { broker });
            connOpts.setPassword(sign.toCharArray());
            connOpts.setCleanSession(true);
            connOpts.setKeepAliveInterval(90);
            sampleClient.setCallback(new MqttCallback() {
                public void connectionLost(Throwable throwable) {
                    System.out.println("mqtt connection lost");
                    while(!sampleClient.isConnected()){
                        try {
                            Thread.sleep(1000);
                            sampleClient.connect(connOpts);
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }
                }
                public void messageArrived(String topic, MqttMessage mqttMessage) throws Exception {
                                        System.out.println("messageArrived:" + topic + "------" + new
String(mqttMessage.getPayload()));
                }
                public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
                    System.out.println("deliveryComplete:" + iMqttDeliveryToken.getMessageId());
                }
            });
            sampleClient.connect(connOpts);
            for (int i = 0; i < 10; i++) {
                try {
                    String scontent = new Date()+"MQTT Test body" + i;
                    final MqttMessage message = new MqttMessage(scontent.getBytes());
                    message.setQos(0);
                    System.out.println(i+" pushed at "+new Date()+" "+ scontent);
                    /**
                     *消息发送到某个主题Topic，所有订阅这个Topic的设备都能收到这个消息。
                 * 遵循MQTT的发布订阅规范，Topic也可以是多级Topic。此处设置了发送到二级Topic
                     */
                    sampleClient.publish(topic+"/notice/", message);
                    /**
                     * 如果发送P2P消息，二级Topic必须是“p2p” ，三级Topic是目标的ClientID
                     * 此处设置的三级Topic需要是接收方的ClientID
                     */
                    String p2pTopic =topic+"/p2p/GID_mqttdelay3@@@DEVICEID_001";
                    sampleClient.publish(p2pTopic,message);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        } catch (Exception me) {
```

```
            me.printStackTrace();
        }
    }
}
```

3. MQTT 接收消息。

   本段代码演示如何使用 MQTT 客户端订阅消息，接收普通的消息以及点对点消息。

```
public class MQTTRecvMsg {
    public static void main(String[] args) throws IOException {
        /**
        * 设置当前用户私有的MQTT的接入点。例如此处示意使用XXX，实际使用请替换用户自己的接入点。接入点的获取方法是，在控制台申请MQTT实例，每个实例都会分配一个接入点域名。
         */
        final String broker ="tcp://XXXX.mqtt.aliyuncs.com:1883";
        /**
         * 设置阿里云的AccessKey，用于鉴权
         */
        final String acessKey ="XXXXXX";
        /**
         * 设置阿里云的SecretKey，用于鉴权
         */
        final String secretKey ="XXXXXXX";
        /**
         * 发消息使用的一级Topic，需要先在MQ控制台里申请
         */
        final String topic ="XXXX";
        /**
         * MQTT的ClientID，一般由两部分组成，GroupID@@@DeviceID
         * 其中GroupID在MQ控制台里申请
        * DeviceID由应用方设置，可能是设备编号等，需要唯一，否则服务端拒绝重复的ClientID连接
         */
        final String clientId ="GID_XXXX@@@ClientID_XXXXXX";
        String sign;
        MemoryPersistence persistence = new MemoryPersistence();
        try {
            final MqttClient sampleClient = new MqttClient(broker, clientId, persistence);
            final MqttConnectOptions connOpts = new MqttConnectOptions();
            System.out.println("Connecting to broker: " + broker);
            /**
             * 计算签名，将签名作为MQTT的password
                        *  签名的计算方法，参考工具类MacSignature，第一个参数是ClientID的前半部分，即GroupID
             * 第二个参数阿里云的SecretKey
             */
            sign = MacSignature.macSignature(clientId.split("@@@")[0], secretKey);
            /**
                    * 设置订阅方订阅的Topic集合，此处遵循MQTT的订阅规则，可以是一级Topic，二级Topic，P2P消息请订阅/p2p
             */
            final String[] topicFilters=new String[]{topic+"/notice/",topic+"/p2p"};
            final int[]qos={0，0};
            connOpts.setUserName(acessKey);
            connOpts.setServerURIs(new String[] { broker });
            connOpts.setPassword(sign.toCharArray());
            connOpts.setCleanSession(true);
            connOpts.setKeepAliveInterval(90);
            sampleClient.setCallback(new MqttCallback() {
```

```
            public void connectionLost(Throwable throwable) {
                System.out.println("mqtt connection lost");
                throwable.printStackTrace();
                while(!sampleClient.isConnected()){
                    try {
                        Thread.sleep(1000);
                        sampleClient.connect(connOpts);
                            //客户端每次上线都必须上传自己所有涉及的订阅关系，否则可能会导致消息接收延迟
                        sampleClient.subscribe(topicFilters,qos);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
            public void messageArrived(String topic, MqttMessage mqttMessage) throws Exception {
                                System.out.println("messageArrived:" + topic + "------" + new String(mqttMessage.getPayload()));
            }
            public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
                System.out.println("deliveryComplete:" + iMqttDeliveryToken.getMessageId());
            }
        });
        //客户端每次上线都必须上传自己所有涉及的订阅关系，否则可能会导致消息接收延迟
        sampleClient.connect(connOpts);
        //每个客户端最多允许存在30个订阅关系，超出限制可能会丢弃导致收不到部分消息
        sampleClient.subscribe(topicFilters,qos);
        Thread.sleep(Integer.MAX_VALUE);
    } catch (Exception me) {
        me.printStackTrace();
    }
  }
}
```

上文代码用到的工具类 MacSignature.java 如下：

```
public class MacSignature {
    /**
     * @param text      要签名的文本
     * @param secretKey 阿里云MQ SecretKey
     * @return 加密后的字符串
     * @throws InvalidKeyException
     * @throws NoSuchAlgorithmException
     */
    public static String macSignature(String text, String secretKey) throws InvalidKeyException, NoSuchAlgorithmException {
        Charset charset = Charset.forName("UTF-8");
        String algorithm = "HmacSHA1";
        Mac mac = Mac.getInstance(algorithm);
        mac.init(new SecretKeySpec(secretKey.getBytes(charset), algorithm));
        byte[] bytes = mac.doFinal(text.getBytes(charset));
        return new String(Base64.encodeBase64(bytes), charset);
    }
    /**
     * 发送方签名方法
     *
     * @param clientId  MQTT ClientID
     * @param secretKey 阿里云MQ SecretKey
     * @return 加密后的字符串
     * @throws NoSuchAlgorithmException
     * @throws InvalidKeyException
```

```
    */
        public static String publishSignature(String clientId, String secretKey) throws
NoSuchAlgorithmException, InvalidKeyException {
      return macSignature(clientId, secretKey);
   }
   /**
    * 订阅方签名方法
    *
    * @param topics    要订阅的Topic集合
    * @param clientId  MQTT ClientID
    * @param secretKey 阿里云MQ SecretKey
    * @return 加密后的字符串
    * @throws NoSuchAlgorithmException
    * @throws InvalidKeyException
    */
   public static String subSignature(List<String> topics, String clientId, String secretKey) throws
NoSuchAlgorithmException, InvalidKeyException {
      Collections.sort(topics); //以字典顺序排序
      String topicText = "";
      for (String topic : topics) {
         topicText += topic + "\n";
      }
      String text = topicText + clientId;
      return macSignature(text, secretKey);
   }
   /**
    * 订阅方签名方法
    *
    * @param topic     要订阅的Topic
    * @param clientId  MQTT ClientID
    * @param secretKey 阿里云MQ SecretKey
    * @return 加密后的字符串
    * @throws NoSuchAlgorithmException
    * @throws InvalidKeyException
    */
     public static String subSignature(String topic, String clientId, String secretKey) throws
NoSuchAlgorithmException, InvalidKeyException {
      List<String> topics = new ArrayList<String>();
      topics.add(topic);
      return subSignature(topics, clientId, secretKey);
   }
}
```

23.8.4.2 MQ 客户端收发 MQTT 消息

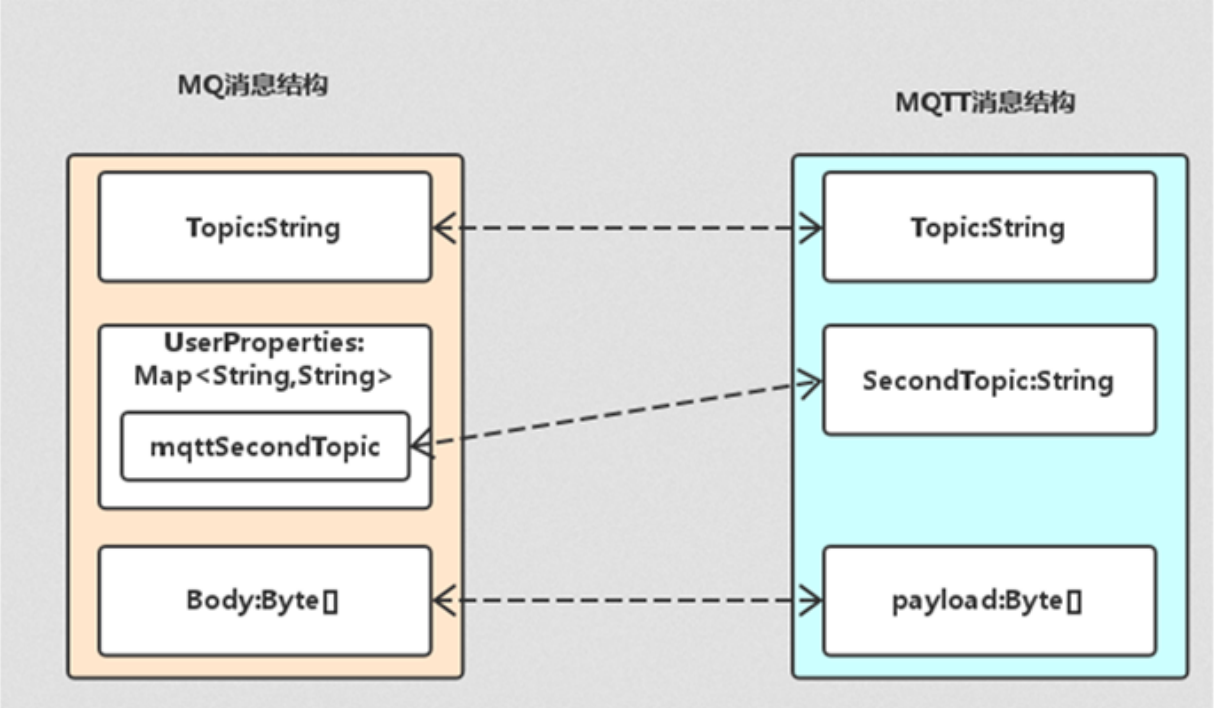
MQ 除了提供标准的 MQTT 服务之外，还支持使用 MQ 客户端来收发 MQTT 消息，实现 MQ 私有协议和 MQTT 协议之间的数据互通。

MQTT 是面向移动端的消息协议，一般单个客户端的处理能力都比较弱。因此 MQTT 协议适用于大量在线客户端，但是每个客户端消息很少的场景。而 MQ 是面向服务端的消息引擎，单个客户端处理能力强，TPS 高，适用于服务端进行大批量处理分析的场景。因此，一般推荐在移动端设备上使用 MQTT，而在服务端应用则使用 MQ 。

例如，业务上有千万数量级的传感器在使用 MQTT 客户端上传数据，此时服务端就可以用 MQ 客户端搭建，用少量的机器来完成这些传感器数据的处理。

MQ 消息和 MQTT 消息的对应关系如下图所示：

图 653: MQ-MQTT消息对应图



使用 MQ 客户端的相关说明，请参见*Java SDK*。

MQ 客户端发送 MQTT 消息

使用 MQ 客户端向 MQTT 设备发消息的方式和普通 MQ 客户端发消息没有区别，只是要根据需求，将 MQTT 的二级 Topic 设置到 MQ 的消息属性中，同时指定消息的 Tag 为 MQ2MQTT 即可。

```
public class ONSSendMsg {
    public static void main(String[] args) throws InterruptedException {
        /**
         * 设置阿里云的AccessKey，用于鉴权
         */
        final String acessKey ="XXXXXX";
        /**
         * 设置阿里云的SecretKey，用于鉴权
         */
        final String secretKey ="XXXXXXX";
        /**
         * 发消息使用的一级Topic，需要先在MQ控制台里申请
         */
        final String topic ="MQTTTestTopic";
        /**
         * ProducerID，需要先在MQ控制台里申请
         */
        final String producerId ="PID_MQTTTestTopic";
        Properties properties =new Properties();
        properties.put(PropertyKeyConst.ProducerId, producerId);
        properties.put(PropertyKeyConst.AccessKey, acessKey);
```

```
        properties.put(PropertyKeyConst.SecretKey, secretKey);
        Producer producer = ONSFactory.createProducer(properties);
        producer.start();
        byte[] body=new byte[1024];
        final Message msg = new Message(
                topic,//MQ消息的Topic，需要事先申请
                        "MQ2MQTT",//MQ Tag，通过MQ向MQTT客户端发消息时，必须指定MQ2MQTT作为Tag，其他Tag或者不设都将导致MQTT客户端收不到消息
                body);//消息体，和MQTT的body对应
        /**
         * 使用MQ客户端给MQTT设备发送P2P消息时，需要在MQ消息中设置mqttSecondTopic属性
         * 设置的值是 “/p2p/” +目标ClientID
         */
        String targetClientID="GID_MQTTTestTopic@@@DeviceID_0001";
        msg.putUserProperties("mqttSecondTopic", "/p2p/"+targetClientID);
        //发送消息，只要不抛异常就是成功。
        SendResult sendResult = producer.send(msg);
        System.out.println(sendResult);
        /**
         * 如果仅仅发送Pub/Sub消息，则只需要设置实际MQTT订阅的Topic即可，支持设置二级Topic
         */
        msg.putUserProperties("mqttSecondTopic", "/notice/");
        SendResult result =producer.send(msg);
        producer.shutdown();
        System.exit(0);
    }
}
```

MQ 客户端接收 MQTT 消息

使用 MQ 客户端接收 MQTT 设备的消息和普通 MQ 客户端收消息没有区别，MQ 客户端只需要订阅 MQTT 的一级 Topic 即可。

```
public class ONSRecvMsg {
    public static void main(String[] args) throws InterruptedException {
        /**
         * 设置阿里云的AccessKey，用于鉴权
         */
        final String acessKey ="XXXXXX";
        /**
         * 设置阿里云的SecretKey，用于鉴权
         */
        final String secretKey ="XXXXXXX";
        /**
         * 收消息使用的一级Topic，需要先在MQ控制台里申请
         */
        final String topic ="MQTTTestTopic";
        /**
         * ConsumerID ,需要先在MQ控制台里申请
         */
        final String consumerID ="CID_MQTTTestTopic";
        Properties properties =new Properties();
        properties.put(PropertyKeyConst.ConsumerId, consumerID);
        properties.put(PropertyKeyConst.AccessKey, acessKey);
        properties.put(PropertyKeyConst.SecretKey, secretKey);
        Consumer consumer =ONSFactory.createConsumer(properties);
        /**
         * 此处MQ客户端只需要订阅MQTT的一级Topic即可
         */
```

```
    consumer.subscribe(topic, "*", new MessageListener() {
        public Action consume(Message message, ConsumeContext consumeContext) {
            System.out.println("recv msg:"+message);
            return Action.CommitMessage;
        }
    });
    consumer.start();
    System.out.println("[Case Normal Consumer Init]   Ok");
    Thread.sleep(Integer.MAX_VALUE);
    consumer.shutdown();
    System.exit(0);
}
```

23.8.5 Android接入示例

本文介绍如何使用 Android 客户端收发 MQTT 消息。因为 Android 和 Java 语法上互通，所以本文仅给出 Android 开发环境的配置，收发程序代码请参考 Java 语言示例。

操作步骤

1. 添加Android 依赖。

   说明: 本文给出的 Android 示例工程使用 Gradle 包管理器管理依赖。

   在 Android 工程中找到 app 下的配置文件 build.gradle。添加依赖如下。

```
dependencies {
    compile 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.0.2'
    compile group: 'commons-codec', name: 'commons-codec', version: '1.5'
}
```

2. 设置收发消息Demo。

   由于 Android 和 Java 语法互通，所以收发消息的代码可以复用 Java 的代码，具体请参考*Demo*工程。

23.8.6 JavaScript接入示例

本文介绍如何使用 JavaScript 客户端收发 MQTT 消息，并给出示例代码供前期开发测试参考。

前提条件

资源申请

使用 MQ 提供的 MQTT 服务，首先需要核实应用中使用的 Topic 资源是否已经申请，如果没有请先去控制台申请 Topic，Group ID 等资源。

申请资源时需要根据需求选择对应的 Region，例如，MQTT 需要使用华北2的接入点，那么 Topic 等资源就在华北2 申请，资源申请具体请参见申请资源。

注意：MQTT 使用的多级子 Topic 不需要申请，代码里直接使用即可，没有限制。

背景信息

说明: 本文给出的实例均基于 Eclipse Paho JavaScript SDK 实现，该 SDK 下载请参见*MQTT*接入准备。如使用其他第三方的客户端，请适当修改。

操作步骤

1. 设置config.js 文件。

```
host = 'mqtt-test.cn-qingdao.aliyuncs.com';// MQTT公测Region的接入点域名，参考MQTT环境准备章节
port = 80;//WebSocket协议服务端口
topic = 'XXXXXXX';//需要操作的Topic
useTLS = false;//是否走加密TLS协议
accessKey = 'XXXXXX';//账号的AccessKey，在阿里云控制台查看。
secretKey = 'XXXXXX=';//账号的的SecretKey，在阿里云控制台查看。
cleansession = true;
groupId='GID_XXXX';
clientId=groupId+'@@@XXXX';//GroupId@@@DeviceId，由控制台申请的Group ID和自己指定的Device ID组合构成。
```

2. 发送程序。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <title>Aliyun Mqtt Websockets</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <script src="mqttws31.js" type="text/javascript"></script>
  <script src="config.js" type="text/javascript"></script>
  <script src="crypto-js.js" type="text/javascript"></script>
  <script type="text/javascript">
  var mqtt;
  var reconnectTimeout = 2000;
  var username =accessKey;
  var password=CryptoJS.HmacSHA1(groupId,secretKey).toString(CryptoJS.enc.Base64);
  function MQTTconnect() {
  mqtt = new Paho.MQTT.Client(
          host,//MQTT域名
          port,//WebSocket端口
          clientId//客户端ClientId
      );
      var options = {
          timeout: 3,
          onSuccess: onConnect,
          onFailure: function (message) {
              setTimeout(MQTTconnect, reconnectTimeout);
          }
      };

      mqtt.onConnectionLost = onConnectionLost;
      mqtt.onMessageArrived = onMessageArrived;

      if (username != null) {
          options.userName = username;
          options.password = password;
```

```
        }
        mqtt.connect(options);
    }

    function onConnect() {
        // Connection succeeded; subscribe to our topic
        mqtt.subscribe(topic, {qos: 0});
        message = new Paho.MQTT.Message("Hello mqtt!!");//set body
        message.destinationName =topic;// set topic
        mqtt.send(message);
    }

    function onConnectionLost(response) {
        setTimeout(MQTTconnect, reconnectTimeout);
    };

    function onMessageArrived(message) {

        var topic = message.destinationName;
        var payload = message.payloadString;
        console.log("recv msg : "+topic+"    "+payload);
    };
    MQTTconnect();
    </script>
  </head>
</html>
```

23.8.7 iOS接入示例

本文介绍如何使用 iOS 客户端收发 MQTT 消息，并给出示例代码供前期开发测试参考，包括资源申请、环境准备、示例代码、注意事项等。

前提条件

资源申请

使用 MQ 提供的 MQTT 服务，首先需要核实应用中使用的 Topic 资源是否已经申请。如果没有，请先去控制台申请 Topic，Group ID 等资源。

MQTT 使用的多级子 Topic 不需要申请，代码里直接使用即可，没有限制。

背景信息

说明: 本文给出的实例均基于第三方框架 MQTT-Client-Framework 实现，SDK 下载请参见 *MQTT*接入准备。如使用其他第三方的客户端，请自行修改。

操作步骤

1. 准备环境。

   iOS 环境下开发 MQTT 客户端程序，一般依赖稳定的第三方 FrameWork。由于涉及网络数据传输，建议选择 Object-c 原生的框架，比如 MQTT-Client-Framework 。

   第三方 FrameWork 一般可以用 CocoaPods 来管理资源包依赖。具体配置流程如下：

a) 安装、配置CocoaPods。

CocoaPods 依赖 Ruby 等基础环境。请确保安装过 Ruby，然后将国内的镜像源更新为淘宝源，提高资源包下载速度。

```
//查看软件源
gem sources -l
//清理掉默认的软件源
gem sources --remove https://rubygems.org/
//加入淘宝的源
gem sources -a https://ruby.taobao.org/
```

更新完地址后，运行以下命令，等待一段时间后即可运行 pod 工具。

```
sudo gem install cocoapods
```

2. 更新项目依赖。

MQTT-Client-FrameWork 第三方框架支持 pod 管理方式，因此添加依赖只需要在项目根目录的 Podfile 中加入依赖即可，内容如下：

```
pod 'MQTTClient'
target '${yourprojectname}' do
end
```

其中 target 填入项目的名称，然后在 Podfile 目录下运行以下命令，更新依赖即可。

```
pod install
```

3. 初始化客户端。

MQTT-Client-FrameWork 包提供的客户端类有 MQTTSession 和 MQTTSessionManager，建议使用后者维持静态资源，而且已经封装好自动重连等逻辑。初始化时需要传入相关的网络参数。具体如下：

```
    self.manager = [[MQTTSessionManager alloc] init];
    self.manager.delegate = self;
            self.manager.subscriptions = [NSDictionary dictionaryWithObject:[NSNumber
numberWithInt:self.qos]
                                    forKey:[NSString stringWithFormat:@"%@/#", self.rootTopic]];
          //password的计算方式是，使用secretkey对groupId做hmac签名算法，具体实现参
考macSignWithText方法
                    NSString *passWord = [[self class] macSignWithText:self.groupId
secretKey:self.secretKey];
    [self.manager connectTo:self.mqttSettings[@"host"]
                       port:[self.mqttSettings[@"port"] intValue]
                        tls:[self.mqttSettings[@"tls"] boolValue]
                  keepalive:60  //心跳间隔不得大于120s
                      clean:true
                       auth:true
                       user:self.accessKey
                       pass:passWord
```

```
                                will:false
                            willTopic:nil
                              willMsg:nil
                              willQos:0
                        willRetainFlag:FALSE
                         withClientId:self.clientId];
```

4. 添加回调接口。

   针对连接当前状态，添加对应的回调接口，可以进行相关的业务逻辑处理。

```
[self.manager addObserver:self
                  forKeyPath:@"state"
                     options:NSKeyValueObservingOptionInitial | NSKeyValueObservingOptionNew
                     context:nil];
-    (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary *)change context:(void *)context {
    switch (self.manager.state) {
        case MQTTSessionManagerStateClosed:
            break;
        case MQTTSessionManagerStateClosing:
            break;
        case MQTTSessionManagerStateConnected:
            break;
        case MQTTSessionManagerStateConnecting:
            break;
        case MQTTSessionManagerStateError:
            break;
        case MQTTSessionManagerStateStarting:
        default:
            break;
    }
}
```

5. 设置userName 和 passWord。

   由于服务端需要对客户端进行鉴权，因此需要传入合法的 userName 和 passWord。userName 设置为当前用户的 AccessKey，password 则设置为 MQTT 客户端 GroupID 的签名字符串，签名计算方式是使用 SecretKey 对 GroupID 做 HmacSHA1 散列加密。具体方法请参考 Demo 中的 macSignWithText 函数。

```
+ (NSString *)macSignWithText:(NSString *)text secretKey:(NSString *)secretKey
{
    NSData *saltData = [secretKey dataUsingEncoding:NSUTF8StringEncoding];
    NSData *paramData = [text dataUsingEncoding:NSUTF8StringEncoding];
    NSMutableData* hash = [NSMutableData dataWithLength:CC_SHA1_DIGEST_LENGTH ];
          CCHmac(kCCHmacAlgSHA1, saltData.bytes, saltData.length, paramData.bytes, paramData.length, hash.mutableBytes);
    NSString *base64Hash = [hash base64EncodedStringWithOptions:0];

    return base64Hash;
}
```

6. 下载Demo工程。

上文描述的客户端代码具体实现请参考 *Demo* 工程。请根据业务需求适当修改后再用于生产环境。

23.8.8 Python接入示例

本文主要介绍如何使用 Python 客户端收发 MQTT 消息，并给出示例代码供前期开发测试参考。包括资源申请、环境准备、示例代码、注意事项等。

前提条件

资源申请

使用 MQ 提供的 MQTT 服务，首先需要核实应用中使用的 Topic 资源是否已经申请。如果没有，请先去控制台申请 Topic，Group ID 等资源。

MQTT 使用的多级子 Topic 不需要申请，代码里直接使用即可，没有限制。

背景信息



说明： 本文给出的示例均基于 Eclipse Paho Python SDK 实现，SDK 下载请参见 *MQTT*接入准备。如使用其他第三方的客户端，请适当修改。

操作步骤

1. 准备环境

   本示例使用 Python 语言编写， Python 客户端依赖的添加可以自行搜索。一般采用 pip 包管理器来处理，具体安装步骤请参考 *Paho* 官方文档。

   官方版本目前不支持 SNI。如果需要使用 SSL 加密的连接方式，请参考以下步骤安装 patch 以支持 SNI，后续等官方更新 SDK 的版本：

   ```
   pip install git+https://github.com/exosite/paho.mqtt.python.git@openssl_sni_support --upgrade
   ```

2. 收发MQTT消息

   本段示例代码演示如何使用 MQTT Python 客户端发送和接收消息。

   ```
   #!/usr/bin/env python
   import hmac
   import base64
   from hashlib import sha1
   import time
   from paho.mqtt.client import MQTT_LOG_INFO, MQTT_LOG_NOTICE, MQTT_LOG_WARNING, MQTT_LOG_ERR, MQTT_LOG_DEBUG
   from paho.mqtt import client as mqtt
   #accessKey get from aliyun account console
   accessKey = 'XXXXXX'
   #secretKey get from aliyun account console
   secretKey = 'XXXXX'
   ```

```
#MQTT GroupID,get from mq console
groupId = 'GID_XXX'
client_id=groupId+'@@@'+'TEST00001'
# Topic
topic = 'XXXX/python/test'
#MQTT endPoint get from mq console
brokerUrl='XXXXXXXXX'
def on_log(client, userdata, level, buf):
    if level == MQTT_LOG_INFO:
        head = 'INFO'
    elif level == MQTT_LOG_NOTICE:
        head = 'NOTICE'
    elif level == MQTT_LOG_WARNING:
        head = 'WARN'
    elif level == MQTT_LOG_ERR:
        head = 'ERR'
    elif level == MQTT_LOG_DEBUG:
        head = 'DEBUG'
    else:
        head = level
    print('%s: %s' % (head, buf))
def on_connect(client, userdata, flags, rc):
    print('Connected with result code ' + str(rc))
def on_message(client, userdata, msg):
    print(msg.topic + ' ' + str(msg.payload))
def on_disconnect(client, userdata, rc):
    if rc != 0:
        print('Unexpected disconnection %s' % rc)
client = mqtt.Client(client_id, protocol=mqtt.MQTTv311)
client.on_log = on_log
client.on_connect = on_connect
client.on_message = on_message
client.on_disconnect = on_disconnect
password = base64.b64encode(hmac.new(secretKey.encode(), groupId.encode(),
 sha1).digest()).decode()
client.username_pw_set(accessKey, password)
client.connect(brokerUrl, 1883, 60)
client.subscribe(topic, 0)
for i in range(1, 11):
    print(i)
    rc = client.publish(topic, str(i), qos=0)
    print ('rc: %s' % rc)
    time.sleep(0.1)
client.loop_forever()
```

3. SSL 方式连接服务

   使用 Python 客户端通过 SSL 方式来连接服务，需要客户端支持 SNI。

   如果是 Paho 提供的客户端，请根据 Release Note 确认是否支持 SNI。如果没有，请按照本文**第1步-准备环境**的说明安装 patch。

   相比标准方式，使用 SSL 方式时需要注意以下两点：

   - 加密端口应设置为8883。
   - 需要导入服务端公钥证书，并配置 TLS 参数。证书下载链接如下：*ca.crt*

   同理，代码上需要在连接之前导入证书文件，并设置连接的 broker 地址。

```
client.tls_set('XXX.crt', server_hostname='XXXXXXX')
client.connect(brokerUrl, 8883, 60)
```

23.8.9 JavaScript接入示例

本文主要介绍如何使用 JavaScript 客户端收发 MQTT 消息，并给出示例代码供前期开发测试参考。

前提条件

资源申请

使用 MQ 提供的 MQTT 服务，首先需要核实应用中使用的 Topic 资源是否已经申请。如果没有，请先去控制台申请 Topic，Group ID 等资源。

MQTT 使用的多级子 Topic 不需要申请，代码里直接使用即可，没有限制。

背景信息

说明: 本文给出的实例均基于 Eclipse Paho JavaScript SDK 实现，该 SDK 下载请参见*MQTT*接入准备。如使用其他第三方的客户端，请适当修改。

操作步骤

1. config.js 文件配置

```
host = 'XXX.mqtt.aliyuncs.com';// 设置当前用户的接入点域名，接入点获取方法请参考接入准备章节文档，先在控制台申请实例
port = 80;//WebSocket协议服务端口，如果是走HTTPS，设置443端口
topic = 'XXXXXXX';//需要操作的Topic
useTLS = false;//是否走加密HTTPS，如果走HTTPS，设置为true
accessKey = 'XXXXXX';//账号的AccessKey，在阿里云控制台查看
secretKey = 'XXXXXX=';//账号的的SecretKey，在阿里云控制台查看
cleansession = true;
groupId='GID_XXXX';
clientId=groupId+'@@@XXXX';//GroupId@@@DeviceId，由控制台申请的Group ID和自己指定的Device ID组合构成
```

2. 收发消息示例程序

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Aliyun Mqtt Websockets</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="mqttws31.js" type="text/javascript"></script>
    <script src="config.js" type="text/javascript"></script>
    <script src="crypto-js.js" type="text/javascript"></script>
    <script type="text/javascript">
    var mqtt;
    var reconnectTimeout = 2000;
    var username =accessKey;
    var password=CryptoJS.HmacSHA1(groupId,secretKey).toString(CryptoJS.enc.Base64);
```

```
    function MQTTconnect() {
    mqtt = new Paho.MQTT.Client(
            host,//MQTT域名
            port,//WebSocket端口
            clientId//客户端ClientId
        );
        var options = {
            timeout: 3,
            onSuccess: onConnect,
            onFailure: function (message) {
                setTimeout(MQTTconnect, reconnectTimeout);
            }
        };
        mqtt.onConnectionLost = onConnectionLost;
        mqtt.onMessageArrived = onMessageArrived;
        if (username != null) {
            options.userName = username;
            options.password = password;
        }
        mqtt.connect(options);
    }
    function onConnect() {
        // Connection succeeded; subscribe to our topic
        mqtt.subscribe(topic, {qos: 0});
        message = new Paho.MQTT.Message("Hello mqtt!!");//set body
        message.destinationName =topic;// set topic
        mqtt.send(message);
    }
    function onConnectionLost(response) {
        setTimeout(MQTTconnect, reconnectTimeout);
    };
    function onMessageArrived(message) {
        var topic = message.destinationName;
        var payload = message.payloadString;
        console.log("recv msg : "+topic+"    "+payload);
    };
    MQTTconnect();
    </script>
  </head>
</html>
```

23.8.10 MQTT实例管理

实例运行期间可以查看当前实例的历史水位曲线，包括消息收发 TPS、同时在线连接数、订阅关系数等信息，也可以设置实例报警，实时监控实例使用情况。

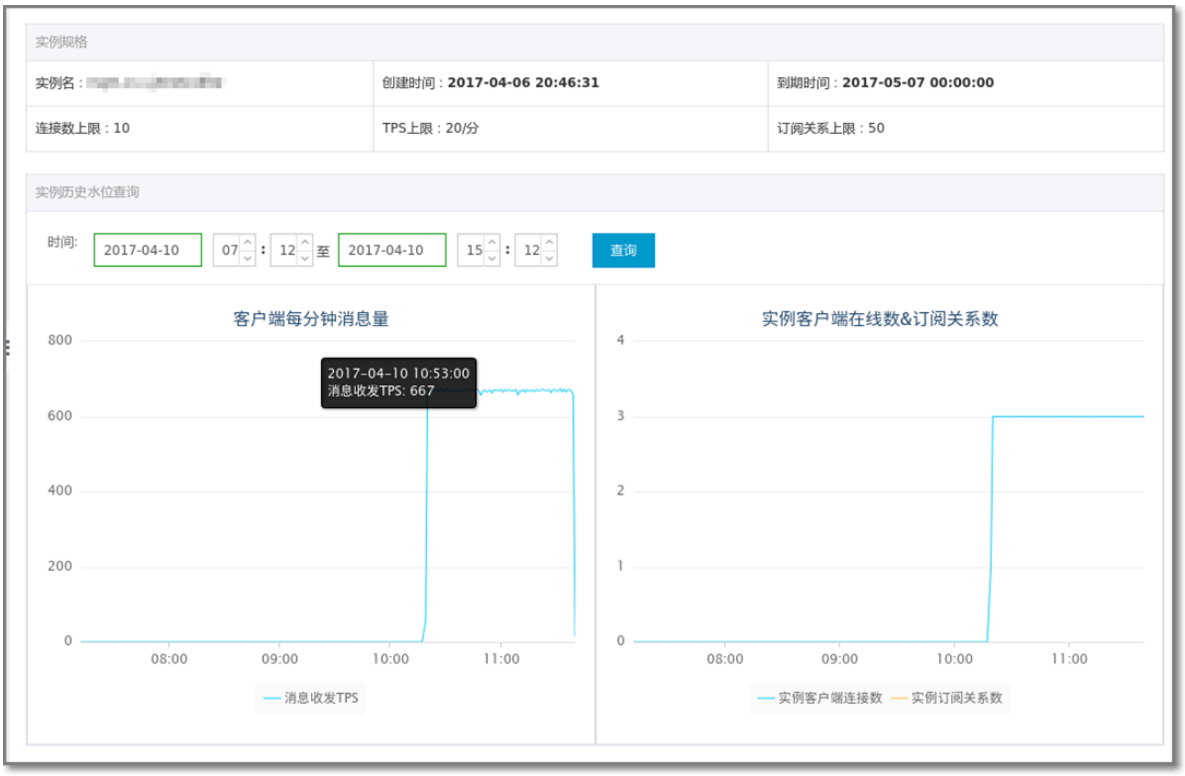
23.8.10.1 查看实例信息

您可以通过MQ控制台随时关注MQTT实例信息。

操作步骤

1. 在控制台左侧的导航栏选择**MQTT 管理** > **MQTT 实例管理** 。
2. 选择需要查看的实例进入实例详情页面，即可查看具体信息。

图 654: 实例信息



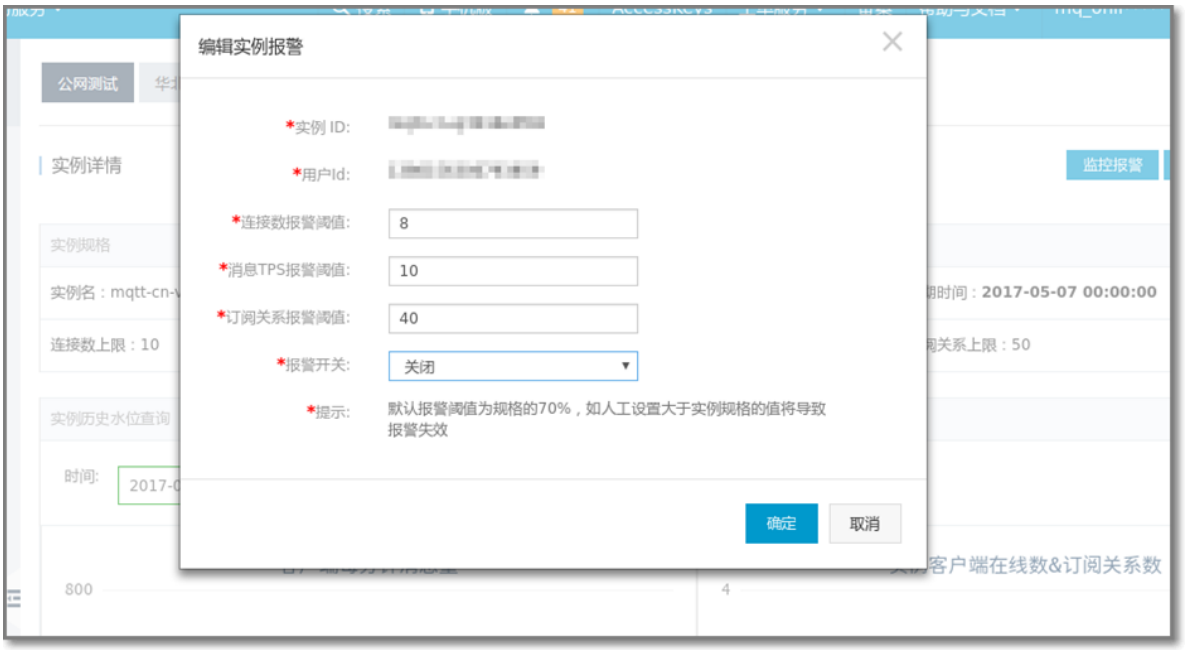
23.8.10.2 设置实例报警

除了人工查询历史水位，MQ 还提供了监控报警功能。在监测到实际使用量超过设置的报警阈值后，MQ 会向联系人手机发送报警短信，提示尽快升级规格。报警阈值默认为规格的70%，报警阈值和报警开关都可以自定义设置。

操作步骤

1. 在实例详情页，单击右上角**监控报警**按钮。
2. 根据需要设置各项报警阈值，单击**确定**。

图 655: 实例报警



23.8.11 查询MQTT数据

MQTT 控制台目前提供多维度的数据查询功能，包括设备查询，状态查询，Topic 查询，以及资源报表查询。

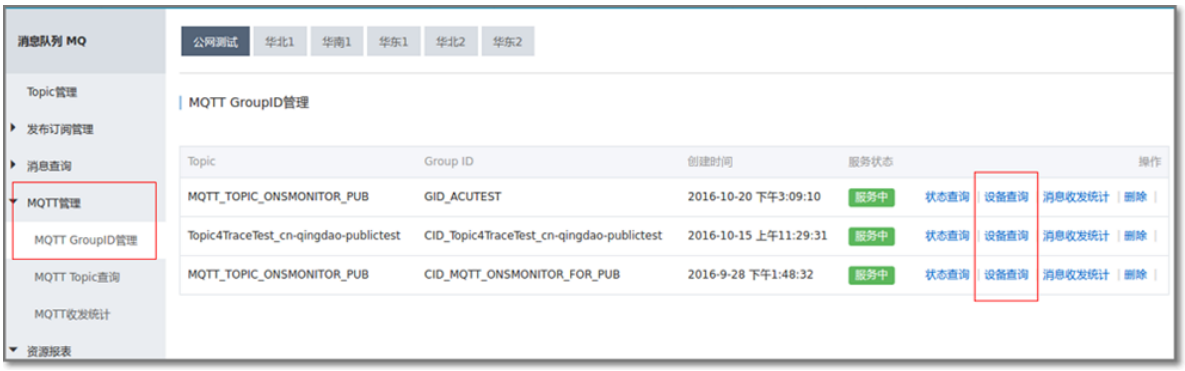
23.8.11.1 查看设备信息

设备查询指根据单个设备独一无二的 Client ID 查询该设备的在线状态以及订阅关系等信息。设备查询入口位于 MQTT Group ID 管理页面。

操作步骤

1. 登录MQ控制台。
2. 在左侧导航栏中选择**MQTT管理** > **MQTT GroupID管理**。
3. 在Topic列表中，选择需要查询的Topic，在操作列单击**设备查询**。

图 656: 设备查询

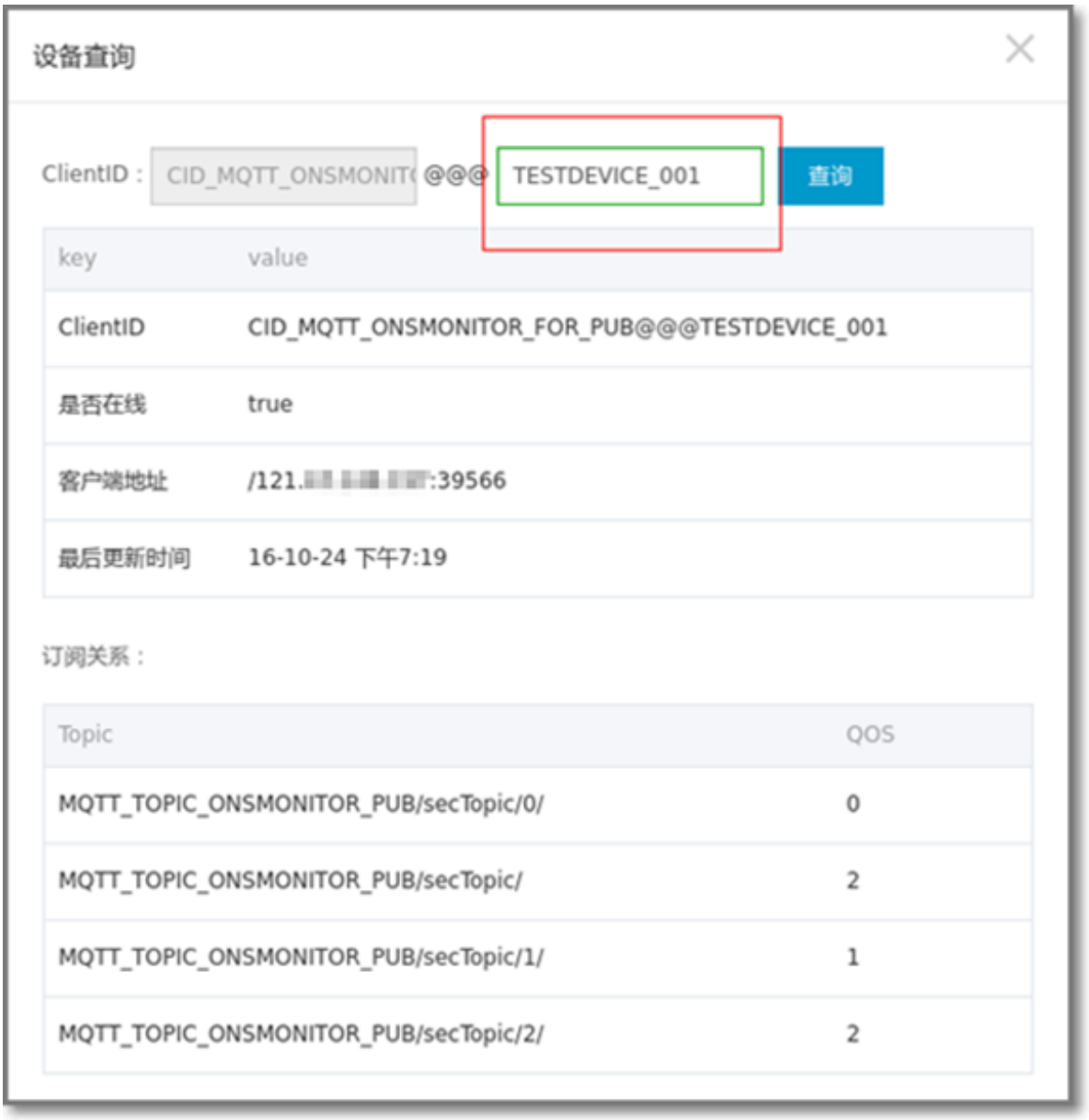


4. 查看设备信息。

   查询结果包含两部分：

   - 在线状态：包含设备的 IP 地址以及最后连接的时间。
   - 订阅关系：显示当前 Client ID 管理的所有 Topic 以及 QoS 级别信息。

   图 657: 设备查询结果



23.8.11.2 查看状态信息

状态查询指根据特定的 Group ID 查询当前 Group ID 下实际的设备在线统计信息，一般用于宏观统计。状态查询的入口位于 MQTT Group ID 管理页面。

操作步骤

1. 登录MQ控制台。
2. 在左侧导航栏中选择**MQTT管理** > **MQTT GroupID管理** 。
3. 在Topic列表中，选择需要查询的Topic，在操作列单击**状态查询**。
4. 查看状态信息。

- 在线连接曲线显示最近15分钟，以该 Group ID 为前缀的设备在线统计数据，该统计值基于分布式数据采集组件收集，可能存在延迟和少量的误差。

图 658: 状态查询



- 消息堆积曲线显示的是最近15分钟内该 Group ID 下设备消费消息的堆积数据。该堆积数据仅仅计算一个采集周期内符合订阅关系的推送值和实际推送返回 ACK 信息的数量之差，并不是绝对的堆积消息数曲线，一般用于评估当前状态的消费速度，是否存在堆积等。

图 659: 消息堆积状态查询



23.8.11.3 查看 Topic 信息

Topic 查询指根据 MQTT 的 Topic 信息查询订阅该 Topic 的所有设备信息，以及该 Topic 最近15分钟内的消息收发情况。

操作步骤

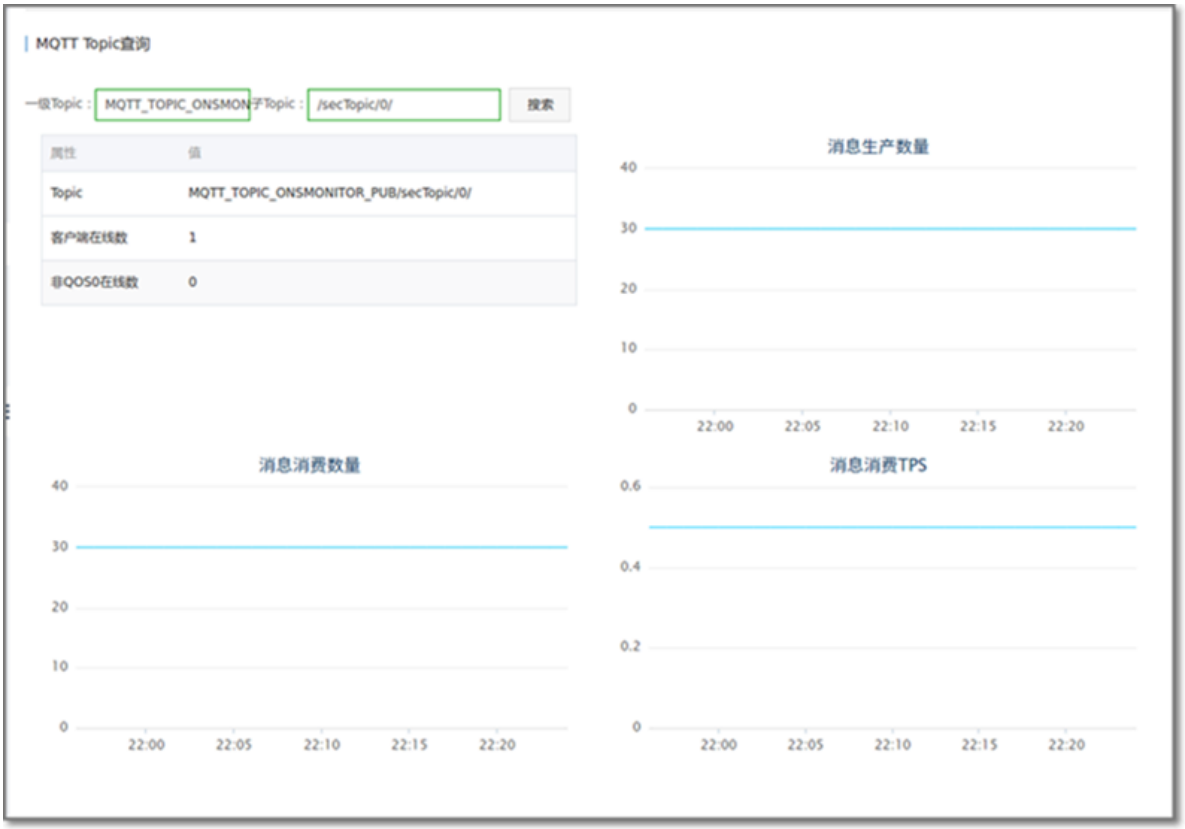
1. 登录MQ控制台。
2. 在左侧导航栏中选择**MQTT管理** > **MQTT Topic查询**。
3. 输入Topic信息，单击**搜索**。

图 660: Topic查询-1



4. 查看Topic信息。

图 661: Topic查询-2



23.8.11.4 查询资源报表

资源报表功能用于统计 MQTT 所有的 Topic 的收发消息数，支持多维度筛选，以及自定义时间范围查询。

操作步骤

1. 登录MQ控制台。
2. 在左侧导航栏中选择**MQTT管理** > **MQTT 收发统计**。
3. 设置查询条件，单击**搜索**。

图 662: MQTT收发消息统计



23.8.12 MQTT常见问题

介绍MQTT接入时可能遇到的常见问题及解决办法。

问题1：连接被异常断开

- MQTT 服务端在客户端发送 Publish 和 Subscribe 报文的时候进行权限验证，如果权限验证失败则会断开连接。
- 不同的客户端使用相同的 Client ID 连接 MQTT 服务，会被强制断开。

问题2：之前订阅过的 Topic 消息还在继续推送

MQTT 协议中订阅关系是持久化的，因此如果不需要订阅某些 Topic，需要调用 unsubscribe 方法取消订阅关系。

问题3：为什么第一次开启消费端，消息接收有很大延迟

MQTT 服务端对于订阅关系的匹配处理是周期定时操作的，目前周期5分钟。因此第一次订阅某个新的 Parent Topic，则这个新的 Topic 从加载到收到第一条消息的延迟最多5分钟，之后的消息接收一般都是毫秒级延迟。

问题4：为什么有的 Topic 的消息能收到，有的收不到

每个 MQTT 客户端允许持有的订阅关系数量有限制，目前是30个。如果一个客户端试图订阅超过该限制数量的 Topic ，则会被丢弃，导致收不到这部分消息。

23.9 产品使用问题

本节介绍MQ使用过程中有可能遇到的使用问题，包括常见问题及异常处理两大类。

23.9.1 常见问题

本节针对MQ使用过程中的常见问题进行分类说明。

23.9.1.1 快速开始

本文简要概括了新手用户使用MQ过程中可能遇到的问题。

1. 新创建的 Consumer ID从哪里开始消费？
   - 如果订阅的 Topic 第一次发送消息是在 3 天内，那么这个新创建的 Consumer ID默认从服务器保存的最早消息开始消费。
   - 如果订阅的 Topic 第一次发送消息是在 3 天前，那么这个新创建的 Consumer ID 默认从服务器保存的最晚消息开始消费，也就是队列尾部开始消费，历史消息不会再消费。
   - 如果这个 Consumer ID 是第二次启动，那么从上次消费的位置开始消费。
   - 如果您想从特定位置开始消费，可以通过 MQ 控制台的消费位点重置功能，指定到具体的时间开始消费。每次重置只针对特定 Consumer ID 下的特定 Topic，不会影响其他 Consumer ID。
2. MQ 消费失败如何重新消费消息？
   - **集群消费方式**

     消费业务逻辑代码如果返回 Action.ReconsumerLater，或者 NULL，或者抛出异常，消息都会走重试流程，至多重试 16 次，如果重试 16 次后，仍然失败，则消息丢弃。每次重试的间隔时间如下：

第几次重试	每次重试间隔时间
1	10 秒
2	30 秒
3	1 分钟
4	2 分钟
5	3 分钟
6	4 分钟
7	5 分钟
8	6 分钟
9	7 分钟
10	8 分钟
11	9 分钟
12	10 分钟
13	20 分钟
14	30 分钟
15	1 小时
16	2 小时

可以通过调用message.getReconsumeTimes()方法来获取消息的重试次数。

- **广播消费方式**

  广播消费方式仍然能保证一条消息至少被消费一次，但消费失败后不做重试操作。

3. 消息发送了，但是没有收到怎么办？

   MQ 提供了多种消息查询方式：

   - 使用 Topic 按时间范围进行查询，可以查询到一段时间内某 Topic 收到的所有消息。
   - 使用 Topic 和 Message Id 对消息进行精准查询。
   - 使用 Topic 和 Message Key 较为精准地查询具有相同 Message Key 的一类消息。

   上述方式可以查询到消息的具体内容以及消费情况，如果需要追踪一条消息从生产者发出到被消费者消费的整个链路中各个相关节点的时间地点，可以使用 MQ 最新的消息轨迹查询功能，具体使用方式请参考4.5.2消息轨迹使用指南。

4. MQ 是否能保证消息不重复？

   绝大多数情况下，消息是不重复的。作为一款分布式消息中间件，在网络抖动、应用处理超时等异常情况下，无法保证消息不重复，但是能保证消息不丢失。

23.9.1.2 配置相关

MQ配置相关的常见问题如下。

1. MQ 消息在服务器保存多长时间？

   存储的消息最多保存 3 天，超过 3 天未消费的消息会被删除。建议配置监控项实时监控消费进度，并根据报警信息人工介入处理。

2. MQ 消息Body的长度限制是多少？

   256KB。

3. MQ 客户端如何设置消费线程数？

   在启动 Consumer 时，设置一个 ConsumeThreadNums 属性即可。具体示例如下所示：

```
public static void main(String[] args) {
    Properties properties = new Properties();
    properties.put(PropertyKeyConst.ConsumerId, "CID_001");
    properties.put(PropertyKeyConst.AccessKey, "xxxxxxxxxxx");
    properties.put(PropertyKeyConst.SecretKey, "xxxxxxxxxxx");
    /**
     * 设置消费端线程数固定为20
     */
    properties.put(PropertyKeyConst.ConsumeThreadNums,20);
    Consumer consumer =ONSFactory.createConsumer(properties);
    consumer.subscribe("TestTopic", "*", new MessageListener() {
        public Action consume(Message message, ConsumeContext context) {
            System.out.println("Receive: " + message);
            return Action.CommitMessage;
        }
    });
    consumer.start();
    System.out.println("Consumer Started");
}
```

4. 由于.NET 客户端的配置问题，导致 dll 加载失败或者其他运行错误时如何处理？

   请参考.NET SDK 压缩包中的文档，确认工程配置与文档说明一致：SDK_GUIDE.pdf。

23.9.1.3 消息轨迹

MQ消息轨迹常见问题如下。

1. 为什么查询不到轨迹数据？

   当输入查询条件查不到轨迹数据时，请参考是否属于以下无法查询情况。

- 消息轨迹目前支持 Java 客户端，且版本至少为1.2.2，C++，.NET，HTTP 尚在开发中，敬请等待。
- 查询条件是否正确，Topic，Message ID，Message Key 是否输入正确。
- 查询时间区间是否正确，为了提高查询速度，需要您输入消息的发送时间范围，如果查询不到，请尝试扩大时间范围重试。
- 如果确认上述情况无误还是无法查询到结果，请提交工单获得技术支持，并附带上日志文件，日志文件位于/home/{user}/logs/ons.log。

2. 确定已经消费了消息，但轨迹却显示没有消费，而且客户端 IP 和 Producer ID 不符合实际情况？

   出现该现象是由于客户端本身并没有升级到支持轨迹功能的版本，因此 MQ 轨迹查询后端仅仅能够拿到部分不完整的轨迹数据，展现结果不正常，建议尽快升级客户端，参考3.5章节。

3. 消费者列表中为什么没有我的 Consumer ID？

   出现该情况的原因可能是消息的下游订阅方太多，轨迹图中空间有限，请将鼠标放在滚动条位置，滚动查看完整数据。

4. 之前的查询任务怎么看不到了？

   为了防止历史查询任务太多，影响展示效果，MQ 会对您的历史查询任务做定期清理，默认只保留7天以内的查询任务，如果出现历史任务看不到，请重新查询即可。

23.9.1.4 告警处理

MQ告警处理部分的常见问题如下。

1. MQ 如何添加消费堆积告警？

   登录 *MQ* 控制台，在左侧菜单栏选择**监控报警**，单击页面右上角**新增监控项**，选择需要报警的 Consumer ID 与 Topic，设置报警阈值以及报警人信息即可完成报警项设置。

2. MQ 用户收到堆积告警后怎么办？

   请按以下步骤操作：

   a. 登录 *MQ*控制台, 在左侧菜单栏单击**发布订阅管理> 订阅管理**，找到相应的 Consumer ID 以及 Topic，然后单击**消费者状态**查看消息堆积情况。

   b. 如果发现消息堆积量很多，按以下步骤进行处理：

      - 首先排查下当前消息处理速度是不是比较慢，即检查消息写入速度是否大于消费速度，可以通过资源报表查询历史消费状态。如果确定消费速度跟不上，可以调整业务代码或者对消费者进行扩容。

- 检查消费是否有“卡住”现象（消费线程阻塞），可以用 jstack 输出程序堆栈进行分析。

c. 如果消息已经没有堆积，可以看看是否阈值设置过小导致消息堆积。可以在监控管理页面找到对应监控项进行编辑。

3. MQ 用户收到订阅关系不一致告警后怎么办？

请按以下步骤操作：

- 登录 *MQ* 控制台，在左侧菜单栏单击**发布订阅管理> 订阅管理**，查看同一个 Consumer ID 是否订阅多个 Topic。
- 检查相同 Consumer ID 客户端订阅关系是否一致（相同 Consumer ID 的不同实例订阅的 Topic 和 Tag 必须保持一致）。

23.9.2 异常类别

本节概括了MQ使用过程中可能遇到的异常类别及处理方法。

23.9.2.1 使用异常

MQ使用过程中可能遇到的异常问题如下。

1. 启动 Producer、Consumer 失败，Group Name 重复。

可能产生的原因：

在同一个 JVM 里面使用同一个 Producer ID/Consumer ID 配置了多个 Producer/Consumer 实例，导致客户端启动失败。

建议解决方案：

按如下步骤操作：

a. 确保一个 JVM 中对应的一个 Producer ID/Consumer ID 只配置一个 Producer/Consumer 实例。

b. 重启应用。

2. 广播模式下，消费者启动加载 JSON 文件异常。

可能产生的原因：

Fastjson 版本太低导致广播消费者加载本地的 offsets.json 文件异常，导致启动失败。

建议解决方案：

将 Fastjson 的版本升级到 ons-client 所依赖的版本，保证本地的 offsets.json 能够被正常加载。默认情况下 offsets.json 在 /home/{user}/.rocketmq_offsets/ 下。

3. 主动订阅消息，获取队列列表失败。

   可能产生的原因：

   可能未在控制台上创建该 Topic，导致订阅方启动时获取 Topic 的队列信息失败。

   建议解决方案：

   按如下步骤操作：

   a. 发布 Topic，*MQ 控制台 > 发布订阅管理 > 发布 Topic*。
   b. 订阅 Topic，*MQ 控制台 > 发布订阅管理 > 订阅 Topic*。
   c. 重启应用。

4. 发送消息尝试多次失败。

   可能产生的原因：

   a. MQ Broker 向 Producer 返回了错误码，具体错误形式参见这个异常对应的嵌套异常。
   b. MQ Broker 宕机后，Producer 还未发现最新的 Broker 列表，会短暂报此异常。
   c. Producer 发送消息超时，一般由服务器压力过大或者网络不稳定导致。

   建议解决方案：

   按如下步骤操作：

   a. 报此异常是暂时的，可能是 MQ Broker 正在重启维护或者是服务器压力过大产生的短暂超时，请稍等一下再尝试。
   b. 如果一直尝试未果，请提交工单获得技术支持。

5. 未收录异常解决办法。

   问题描述：

   暂未收录任何异常。

   建议解决方案：

   请提交工单获得技术支持。

23.9.2.2 资源不存在

资源不存在相关的异常问题如下。

1. 订阅组（Consumer Group）不存在

   可能产生的原因：

   未在控制台上创建该订阅组，导致使用该订阅组与 MQ Broker 创建连接的时候，服务器校验不通过。

   建议解决方案：

   按如下步骤操作：

   a. 前往 MQ 控制台创建订阅组，*MQ 控制台 > 发布订阅管理 > 订阅 Topic*。

   b. 重启应用。

2. Topic 不存在

   可能产生的原因：

   未在控制台上创建该 Topic，导致 Producer 向 MQ 发送消息时，服务器校验不通过。

   建议解决方案：

   按如下步骤操作：

   a. 前往 MQ 控制台发布 Topic，*MQ 控制台 > Topic管理 > 发布 Topic*。

   b. 重启应用。

3. 主机名不存在

   可能产生的原因：

   可能是无法正确获取主机名或者主机 IP 地址导致，请尝试使用此命令来证实：hostname

   如果无法正常输出，就说明确实是此原因；如果可以正常输出，可能是其他原因，请请提交工单获得技术支持。

   建议解决方案：

   按如下步骤操作：

   a. 在报错机器上执行查看主机名命令。

   ```
   [root@iZ231wxgt6mZ ~]# hostname
   iZ231wxgt6mZ
   ```

如果执行命令报错，请检查是否给 hostname 定义了别名，比如在 .bash_profile 或者 .bashrc 中 alias xxx= ‘hostname’ ； 或者命令路径不在 $PATH 下面。

b. Ping主机。

```
[root@iZ231wxgt6mZ ~]# ping iZ231wxgt6mZ
```

如果无法正常 ping 通主机名，则需要将本机地址绑定到 /etc/hosts 文件中。默认 ECS 机器都会有一 个本地地址和主机名的绑定关系，切勿手动的将其去除。

c. 确认系统配置。

检查 /etc/sysconfig/network 中的记录的 hostname 是否和 /etc/hosts 中的主机名绑定一致，如果不一致请修改。如果需要修改 /etc/sysconfig/network 中的内容，修改后需要重启机器才能生效。 推荐不要随意修改系统文件里的配置，可以会引发一些其他异常问题。

以上三个步骤确认完毕后，客户端启动就不在会报 UnknownHostException 的异常了。

23.9.2.3 状态不一致

状态不一致相关的异常问题如下。

1. 消息不合法

可能产生的原因：

一般是消息属性、消息内容不合法，不合法的情况有：

- 消息为空
- 消息内容为空
- 消息内容长度为0
- 消息内容超过限定长度

建议解决方案：

请确保消息没有以上的不合法情况，并根据异常提示进行解决。

2. 参数不合法

可能产生的原因：

参数不合法的情况有以下几种：

嵌套的异常说明	异常描述
consumeThreadMin Out of range [1, 1000]	消费端线程数设置不合理

嵌套的异常说明	异常描述
consumeThreadMax Out of range [1, 1000]	消费端线程数设置不合理
messageListener is null	未设置messageListener
consumerGroup is null	未设置Consumer ID
msg delay time more than 40 day	定时消息延时不能超过40天

建议解决方案：

按照如下步骤操作：

a. 按照异常提示修改客户端对应参数的配置，确保其在合理范围内。

b. 重启应用。

3. 客户端状态异常

可能产生的原因：

a. 创建 Consumer、Producer 之后未显示调用 start() 方法来启动客户端。

b. 创建 Consumer、Producer 之后 start() 过程有异常导致客户端启动失败。

c. 创建 Consumer、Producer 并成功调用 start() 方法后，显示调用了 shutdown() 方法关闭了客户端。

建议解决方案：

按如下步骤操作：

a. 确保创建 Consumer、Producer 之后调用 start() 保证客户端处于启动状态。

b. 查看 ons.log 判断在客户端在启动过程中是否有异常。

4. 订阅关系不一致

问题描述：

在不同的 JVM 中启动了多个 Consumer，并且给相同的 Consumer ID 配置了不同的 Topic，或者是相同的 Topic 但 Tag 不同，最终导致订阅关系不一致，消息不符合预期。

错误代码示例：

- 错误示例一：同一个订阅组（CID-MQ-FAQ）订阅的 Topic 不同（分别是 MQ-FAQ-TOPIC-1、MQ-FAQ-TOPIC-2）。

  JVM-1上的代码：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.ConsumerId, "CID-MQ-FAQ");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("MQ-FAQ-TOPIC-1", "NM-MQ-FAQ", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println("Receive: " + message);
        return Action.CommitMessage;
    }
});
consumer.start();
```

JVM-2上的代码：

```
 Properties properties = new Properties();
properties.put(PropertyKeyConst.ConsumerId, "CID-MQ-FAQ");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("MQ-FAQ-TOPIC-2", "NM-MQ-FAQ", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println("Receive: " + message);
        return Action.CommitMessage;
    }
});

consumer.start();
```

- 错误示例二：同一订阅组（CID-MQ-FAQ）订阅的 Topic 相同，但 Tag 不同（分别NM-MQ-FAQ-1、NM-MQ-FAQ-2）。

  JVM-1上的代码：

  ```
  Properties properties = new Properties();
  properties.put(PropertyKeyConst.ConsumerId, "CID-MQ-FAQ");
  Consumer consumer = ONSFactory.createConsumer(properties);
  consumer.subscribe("MQ-FAQ-TOPIC-1", "NM-MQ-FAQ-1", new MessageListener() {
      public Action consume(Message message, ConsumeContext context) {
          System.out.println("Receive: " + message);
          return Action.CommitMessage;
      }
  });
  consumer.start();
  ```

  JVM-2上的代码：

  ```
   Properties properties = new Properties();
  properties.put(PropertyKeyConst.ConsumerId, "CID-MQ-FAQ");
  Consumer consumer = ONSFactory.createConsumer(properties);
  consumer.subscribe("MQ-FAQ-TOPIC-1", "NM-MQ-FAQ-2", new MessageListener() {
      public Action consume(Message message, ConsumeContext context) {
          System.out.println("Receive: " + message);
          return Action.CommitMessage;
      }
  });

  consumer.start();
  ```

建议解决方案：

请确保在不同 JVM 中使用相同的 Consumer ID 启动多个 Consumer 时，配置的 Topic 和 Tag 是一致的。

24 企业实时监控服务ARMS

24.1 产品概述

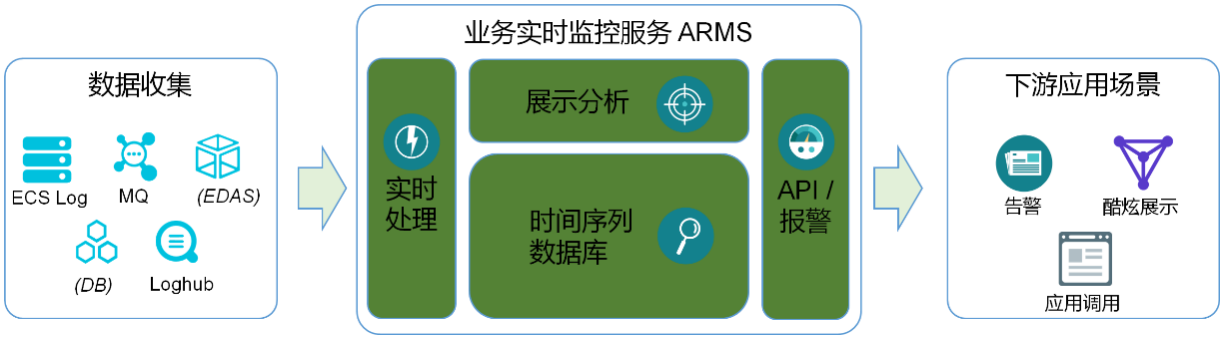
业务实时监控服务（Application Real-Time Monitoring Service，简称 ARMS ）是一款端到端一体化实时监控解决方案的 PaaS 级阿里云产品。

基于ARMS，您可以利用海量数据迅速定制秒级响应的业务监控能力。在技术架构上，ARMS 整合和封装了数据收集、消息通道、实时计算、列式存储，以及在线报表等多种先进互联网技术组件。在用户体验方面，ARMS 为您屏蔽了复杂的监控计算逻辑，针对不同行业提供了快速搭建基于系统监控、商品销售、网站分析等各种场景监控方案的能力。

通过 ARMS，您只需要拖拽式操作三个步骤，即可搭建出一套监控服务：

- 日志和数据接入
- 实时计算任务编排
- 告警和数据集定制

图 663: ARMS示意图



24.2 名词解释

本文介绍ARMS常见的名词和术语。

监控任务

一个任务代表ARMS从数据抓取，数据处理，数据存储到结果展示的一个实例。

数据源

ARMS获取数据的方式，如服务器日志、MQ。

清洗规则

清洗规则定义在一个监控任务中如何解析采集到的数据。清洗规则在监控任务中必须定义。

数据集

数据集定义了监控任务中如何基于采集到的数据做聚合计算，持久化存储，以及 Open API 访问输出。数据集分为下钻数据集和普通数据集，目前默认为下钻数据集，普通数据集暂时不支持，敬请期待。

报警规则

报警规则定义了如何基于数据集生成报警以及通知方式。报警的优先级目前暂分为三种：警告、错误、致命。

24.3 准备数据源

24.3.1 数据源简介

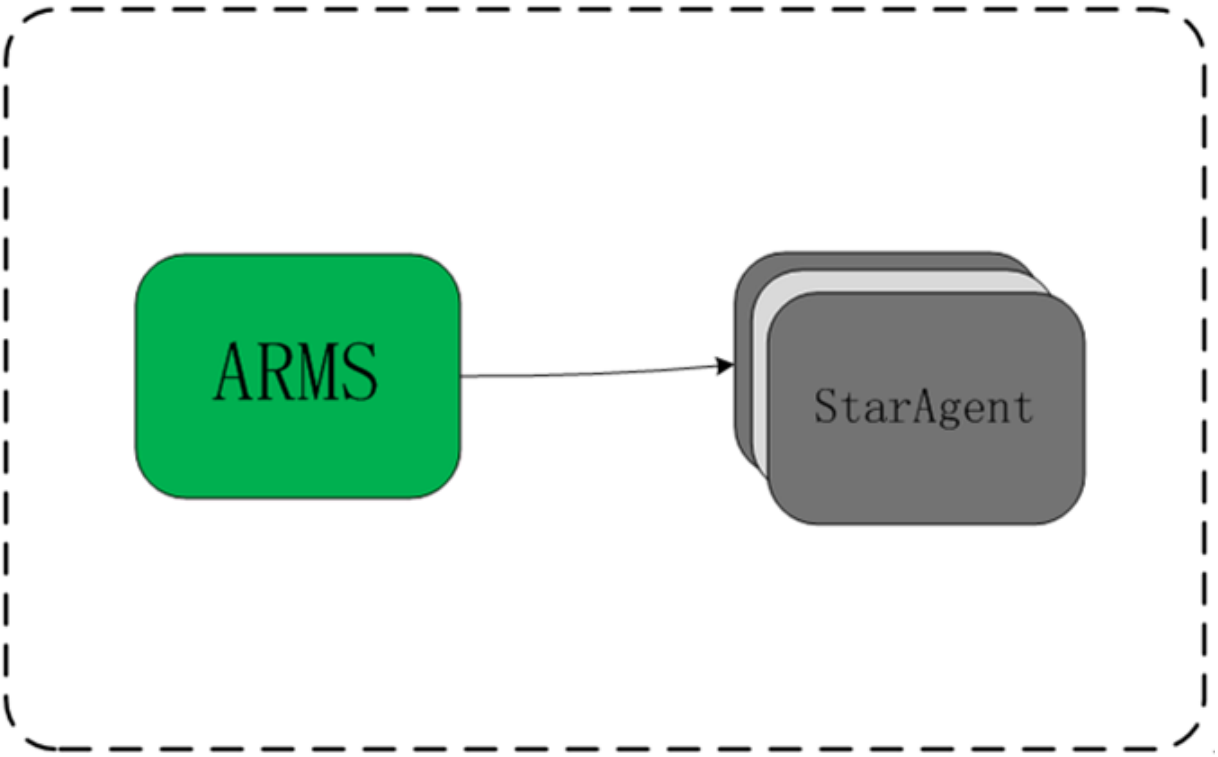
数据源（日志源）为 ARMS 提供数据流入，您可以通过各种方式将数据推送到 ARMS 实时计算引擎。在专有云环境下，ARMS支持通过StarAgent 完成在 VM/Docker实例上的日志文件的增量推送，同时支持消费MQ中消息数据。适用场景主要包括专有云监控场景。

ARMS可以通过以下方式与StarAgent进行连接。

服务器直连

通过 StarAgent 完成在服务器上日志的增量拉取，例如日志文件。适用所有可以网络直连互通的网络环境。

图 664: 服务器直连



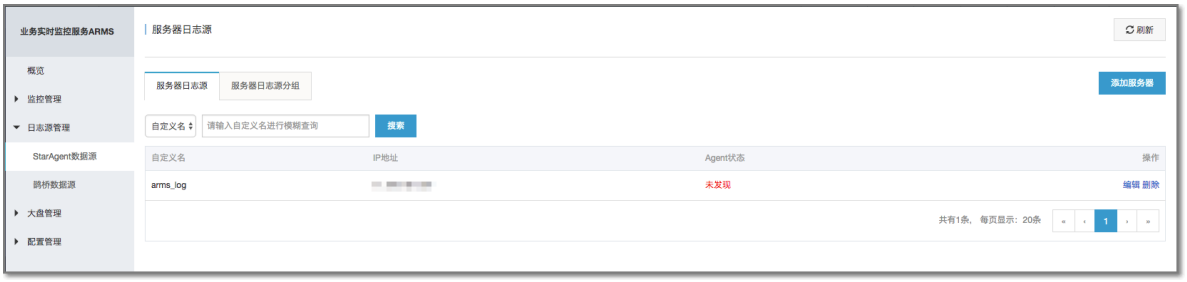
24.3.2 添加数据源

本文以StarAgent数据源为例描述添加数据源的具体步骤。鹊桥数据源的操作步骤与此类似。

操作步骤

1. 在ARMS控制台左侧菜单栏选择**日志源管理** > **StarAgent数据源**。

图 665: StarAgent数据源



2. 单击页面右上角的**添加服务器**，在**添加日志源服务器**对话框输入日志源名称和服务器IP地址。

图 666: 添加日志源服务器

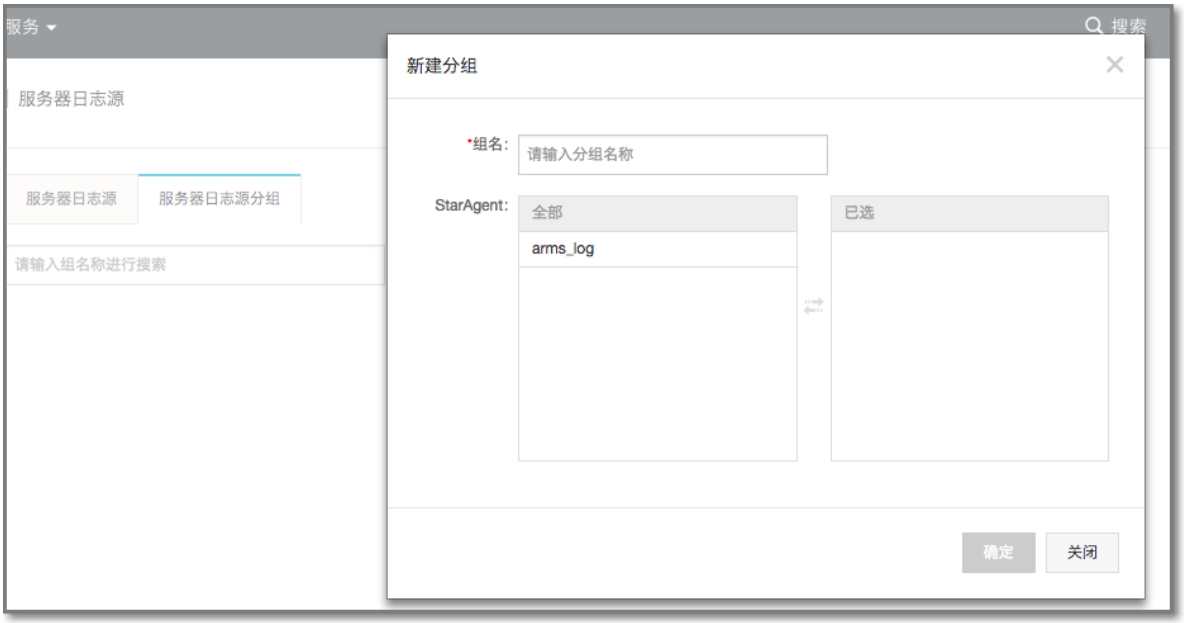


3. 单击**确定**。

添加好日志源后，也可以为日志源创建服务器分组，将服务器按功能进行分类，方便使用时直接选择分组即可。

在服务器日志源页面，单击**服务器日志源分组**Tab，并单击页面右上角的**创建服务器分组**。

图 667: 服务分组



24.3.3 安装Agent

Agent是日志系统中处于数据源头的关键组件，为了使ARMS能抓取日志，需要在每一台日志源服务器上安装Agent。

操作步骤

1. 联系ARMS运维人员获取安装资源 *dragoon.tar.gz*。
2. 确保将dragoon.tar.gz下载到本地目录，并执行安装脚本：

```
tar -zxvf ./dragoon.tar.gz -C /root
mkdir -p /usr/alisys
ln -s  /root/dragoon/  /usr/alisys
```

3. 配置白名单。

   白名单格式分两种：IP白名单和文件目录白名单。

   IP白名单配置说明如下：

   - IP白名单配置文件路径为：*/usr/alisys/dragoon/conf/white_ip.conf*。
   - Client的IP必须在白名单里才允许访问。
   - IP 可以用以下符号分隔：换行符、逗号、分号、空格。
   - 最大支持10000个。
   - 如果需要所有机器都可访问，可以利用默认配置文件，即**1~255**，具体写法可以参考安装包默认配置。
   - 如果是网络直连模式，白名单直接填写ARMS服务器的IP地址。具体地址请联系ARMS运维人员；如果是通过鹊桥方式连接，白名单直接填写sProxy的IP地址。

   文件白名单配置说明如下：

   - Agent要求的文件目录为："/alidata/", "/apsara/", "/home/", "/opt/", "/tmp/", "/usr/", "/var/"，文件末尾包含log或LOG。
   - 具体可以在agent机器上执行命令查看： `/usr/alisys/dragoon/bin/DragoonAgent d -e GetConfig`
   - 也可通过修改*/usr/alisys/dragoon/conf/staragent.conf*修改配置，配置内容和GetConfig命令返回的内容一致。
   - 最大权限是将目录设置为 "/"。

   说明:

   - 在不重启的情况下，修改或者增加白名单文件后，默认一小时内生效。

- 通过执行命令可以使白名单立即生效：/usr/alisys/dragoon/bin/DragoonAgent -e ReloadConfig。
- 重启Agent会使白名单立即生效。

Agent安装完毕后，可以执行以下命令进行启停：

- 启动：/usr/alisys/dragoon/bin/DragoonAgent -H 0.0.0.0
- 停止：/usr/alisys/dragoon/bin/agent.sh stop

24.4 创建监控任务

24.4.1 监控任务概述

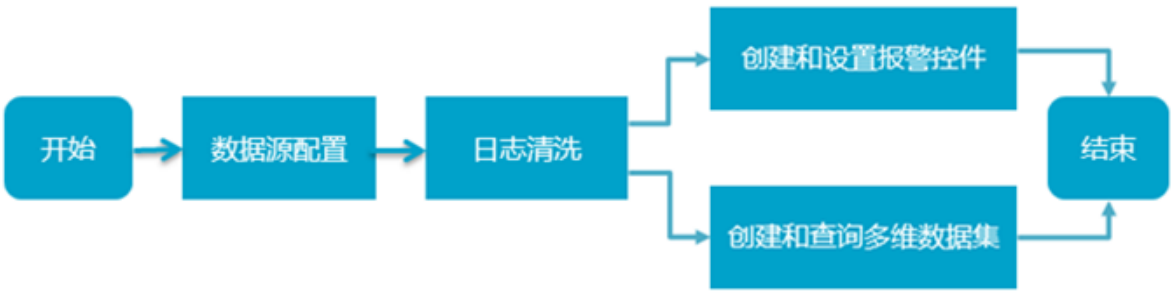
目前 ARMS 目前支持的监控任务类型包括：自定义监控、导入模板、行业模板。

创建监控任务流程

在 ARMS 中创建监控任务主要包含**数据源配置**、**日志清洗**、**数据集与报警配置**三大步骤。

在第三步**数据集与报警配置**里，可以根据需要进行不同的配置，具体流程如下图所示。

图 668: 创建监控任务



目前 ARMS 目前支持三种任务类型：自定义监控、导入模板和行业模板。

自定义监控

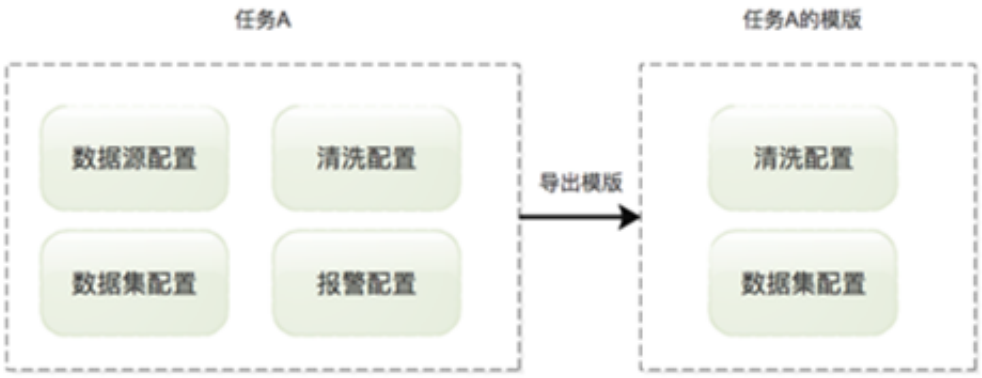
自定义监控是 ARMS 提供的通用监控类型。对日志进行清洗之后，您可以自由地统计所需指标，产生需要的数据与报表，灵活地配置报警。

导入模板

如果需要跨 Region 或者在同一 Region 下复制一个监控任务，则可以使用导入模板的方式。

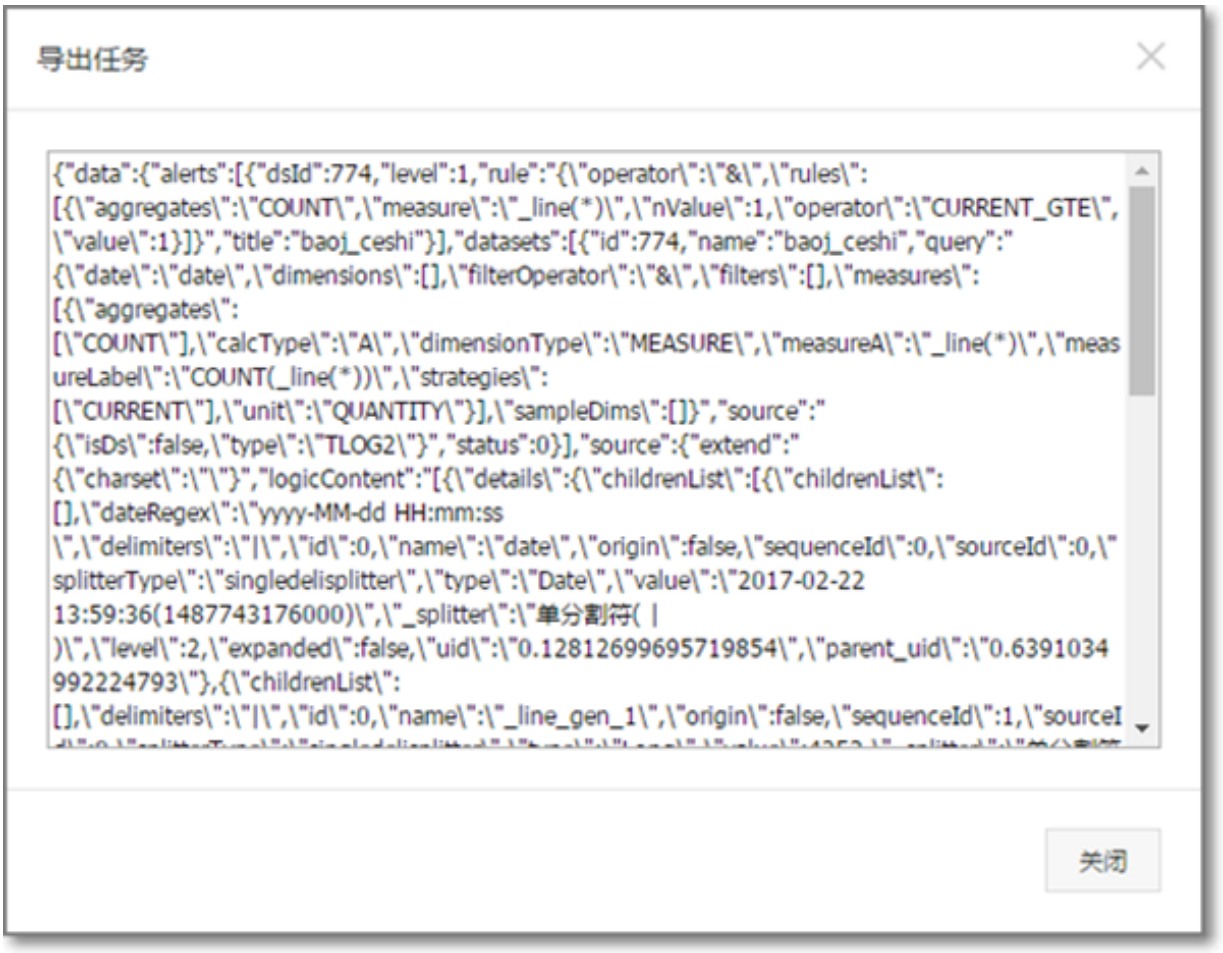
一个完整的监控任务包含数据源配置、清洗配置、数据集或报警配置，而导出的模板中包含的是该任务的通用配置，即清洗配置（日志样例与清洗规则）和数据集配置。导入模板后，新建的任务需要重新配置数据源。

图 669: 监控任务模板



导出的模板格式为 JSON 文件。

图 670: 导出模板



行业模板

行业模板是指针对某个行业或者某类问题的监控任务解决方案。一个行业模板可能包含多个监控任务和交互式大盘。行业模板为系统内置，后续会陆续开放相关模板，如 Tsar 系统监控、Nginx 监控等。

根据行业模板创建任务与交互式大盘后，只需要到对应的任务与交互式大盘进行相应的修改即可，减少了很多繁琐的配置过程。

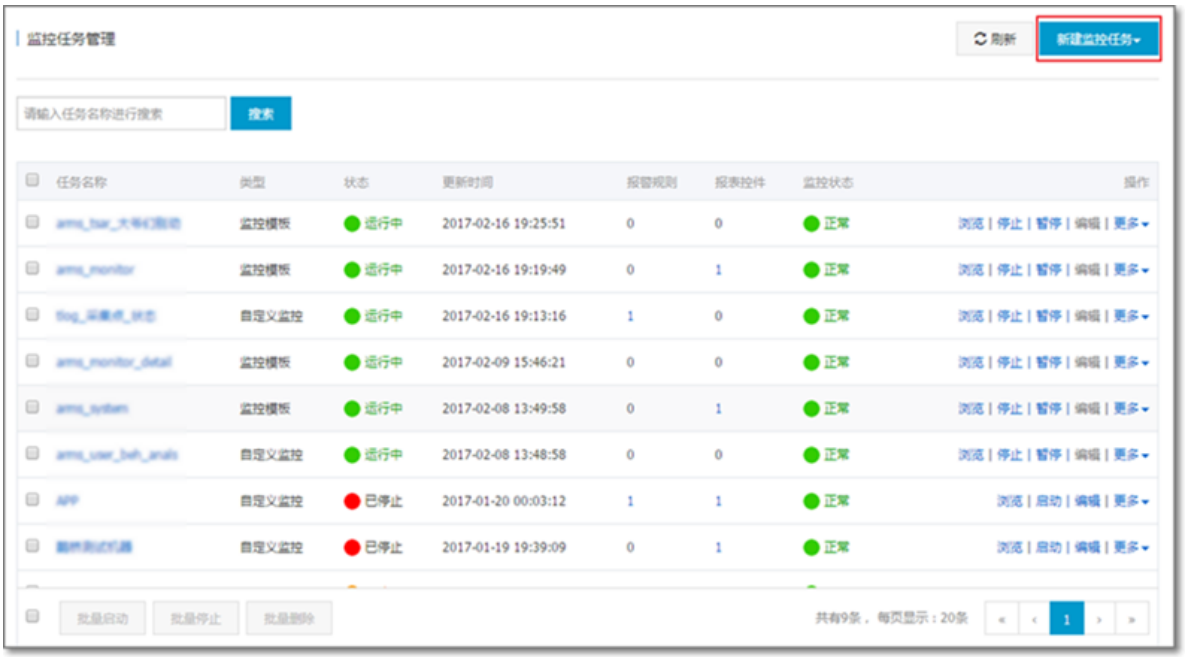
24.4.2 配置数据源

监控任务配置分为三个步骤：数据源配置、日志切分、数据集与报警配置。本文介绍第一步数据源配置。

操作步骤

1. 在控制台左侧菜单栏选择**监控管理** > **监控任务管理**，单击页面右上角**新建监控任务**。

图 671: 新建监控任务



2. 在新建监控任务下拉列表选择监控类型。

   目前支持三种新建方式：自定义、导入模板、行业模板。

   - 如果选择自定义创建任务，需在对话框输入监控任务名称并选择监控任务类型，单击**新建并进行配置**（后续以此方式进行介绍）。

     图 672: 自定义任务



   - 如果选择导入模板创建任务，需要从现有监控任务中先导出模板。导出模板操作如下：

     1. 在监控任务管理页，找到需要导出模板的监控任务。
     2. 单击右侧**更多** > **导出模板**选项。

3. 拷贝导出的JSON文件，回到新建导入模板任务对话框，输入监控任务名称与模板配置。

图 673: 导入模板



- 如果选择行业模板创建任务，需修改任务名称与交互式大盘的名称，并单击**新建并进行配置**。

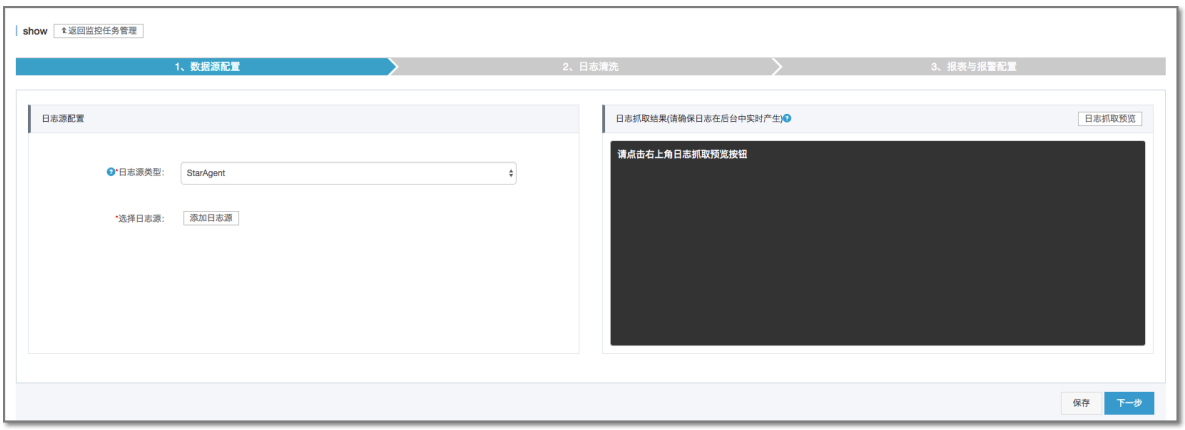
图 674: 行业模板



选择好监控类型后，需要进行数据源配置。以下步骤以选择自定义监控类型为例。

3. 在日志源配置页面，添加StarAgent日志源，并设置编码方式。

图 675: 日志源配置



注意:

- 请输入日志的绝对路径，如**/apps/arms.log**；如果需要配置多个文件，以逗号分隔，如**/root/arms01.log,/apps/arms02.log**。

- 编码设置：默认为“自动探测”模式，也可以根据自己的日志编码格式灵活选择。

4. 设置完成后**保存**，并单击右上角**日志抓取预览**，进行日志抓取。
   - ARMS 会从选择的机器日志中抓取部分数据（最多20条），抓取时间一般为30秒左右。
   - 如果预抓取日志未成功，请检查之前的日志路径配置和日志源选择。在日志预抓取成功之后再进行下一步操作。

   完成以上日志源配置操作之后，就可以进入下一步**日志清洗**了。

24.4.3 日志清洗

日志清洗是指对日志进行切分、静态 JOIN 等，最终转化标准的 Key-Value(KV) 格式。目前日志清洗支持两种模式：智能切分、自定义切分，请根据需要进行选择。

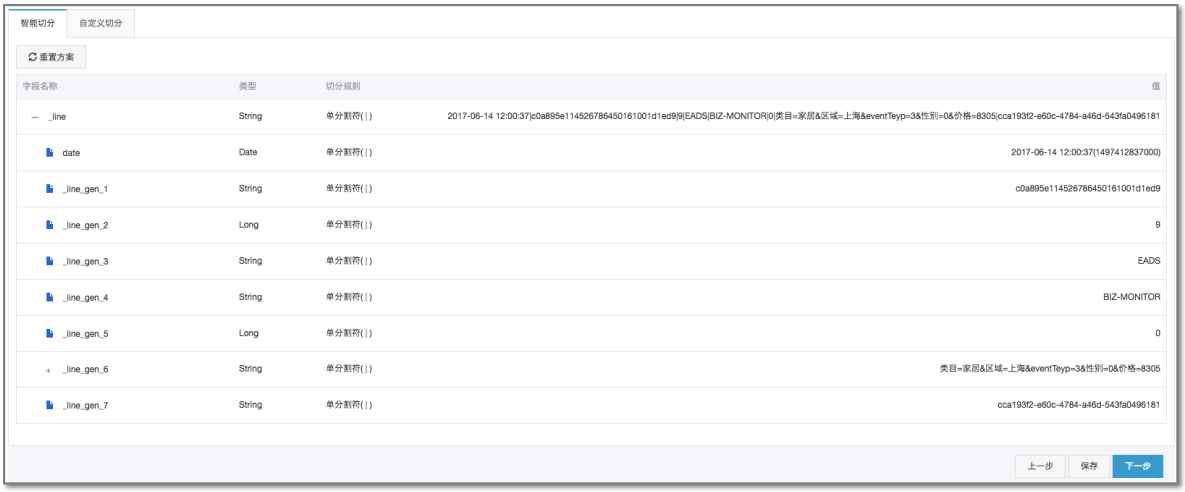
24.4.3.1 智能切分

ARMS会对基于日志抓取的结果进行智能切分，转化为标准的 KV 格式，并提供几种可选方案。对选择的方案也可以进行调整，包括删除和修改。

操作步骤

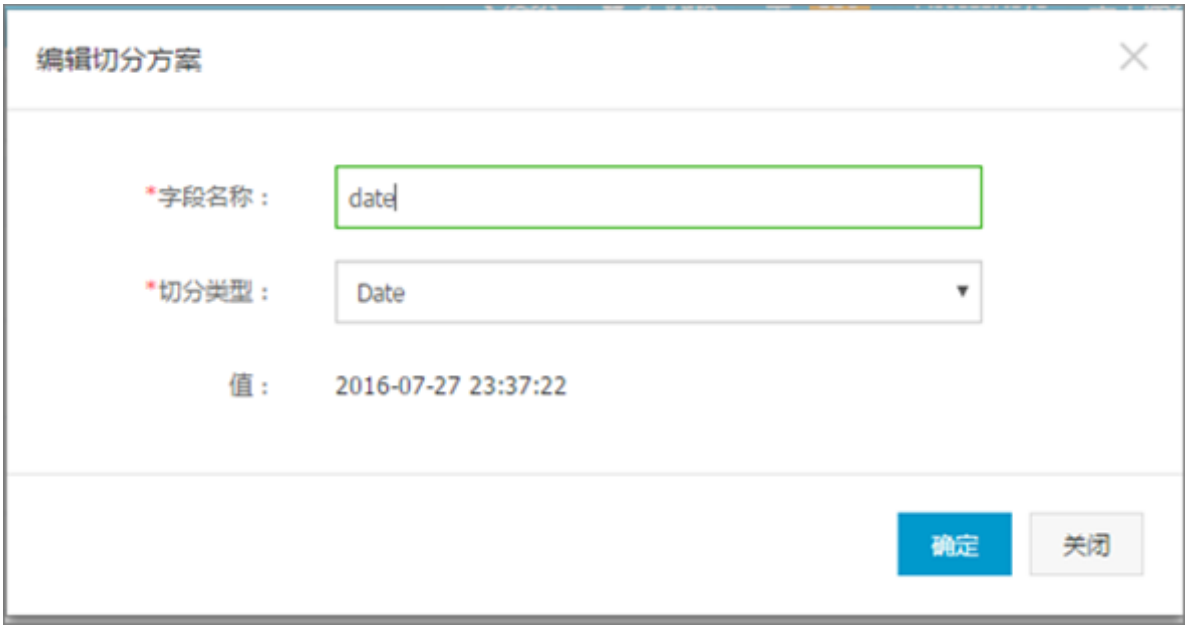
1. 在**日志清洗**页面，单击**智能切分**。
2. 单击右上角**重置方案**按钮。

图 676: 智能切分



3. 将鼠标放在需要修改的字段对应的一行，会显示编辑和删除图标。可以根据需要对字段进行微调。例如将方案1中的“Date1”字段修改为“date” ，如下图。

图 677: 修改方案



说明: 方案被修改后，如果想还原推荐的方案，重新点击**重置方案**即可。

24.4.3.2 自定义切分

您也可以选择自定义切分模式，具体信息请参考自定义切分的使用。

操作步骤

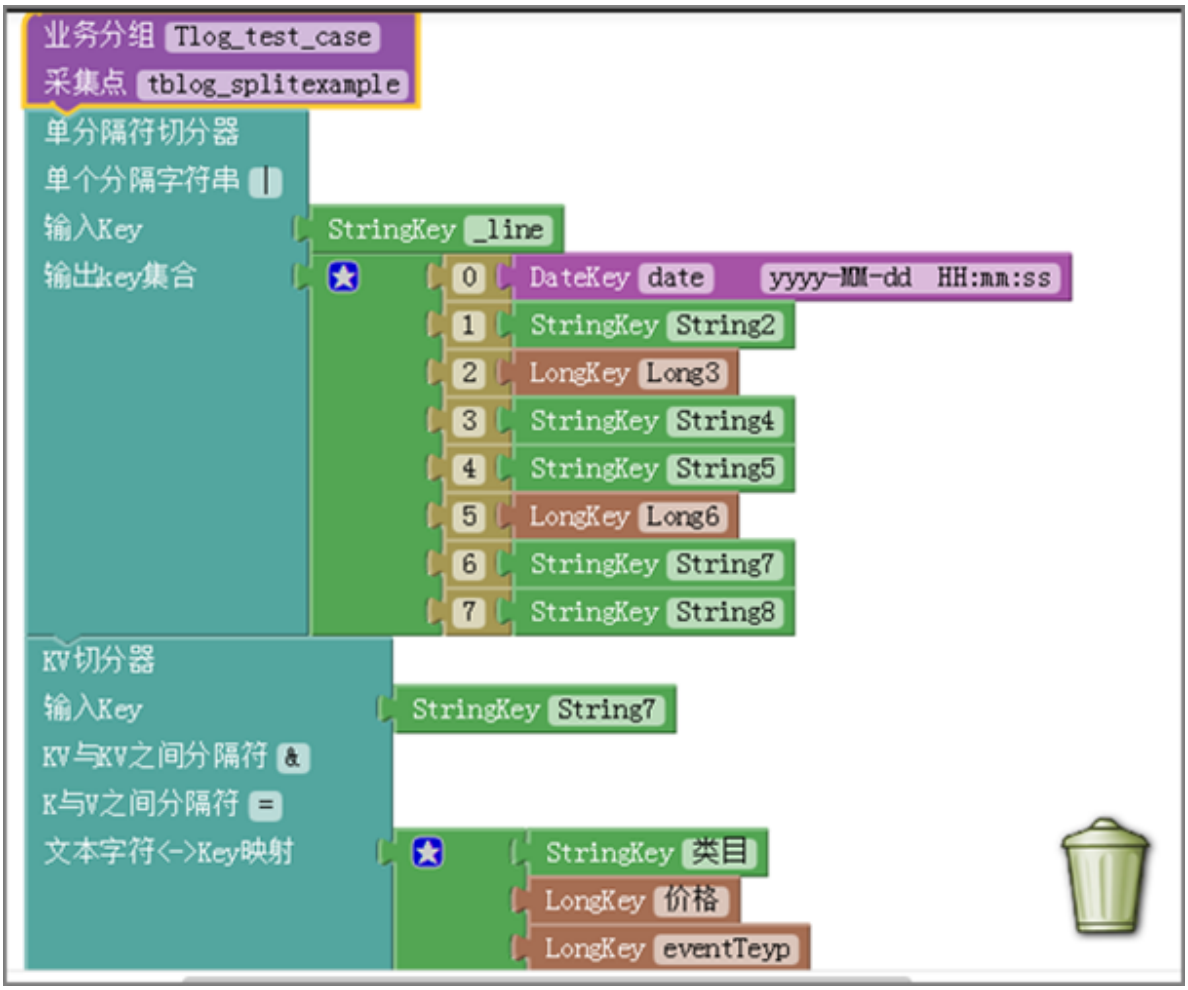
1. 在数据清洗页面，单击**自定义切分**，进入自定义切分模式。

   以下为范例日志。

```
2016-07-27 23:37:23|c0a895e114526786450161001d1ed9|9|EADS|BIZ-MONITOR|0|类目=女装&区域=上海&eventTeyp=3&性别=0&价格=20|iZ28ql4lx29Z
2016-07-27 23:37:23|c0a895e114526786450161001d1ed9|9|EADS|BIZ-MONITOR|0|类目=童装&区域=深圳|&eventTeyp=3&性别=1&价格=30|iZ28ql4lx29Z
```

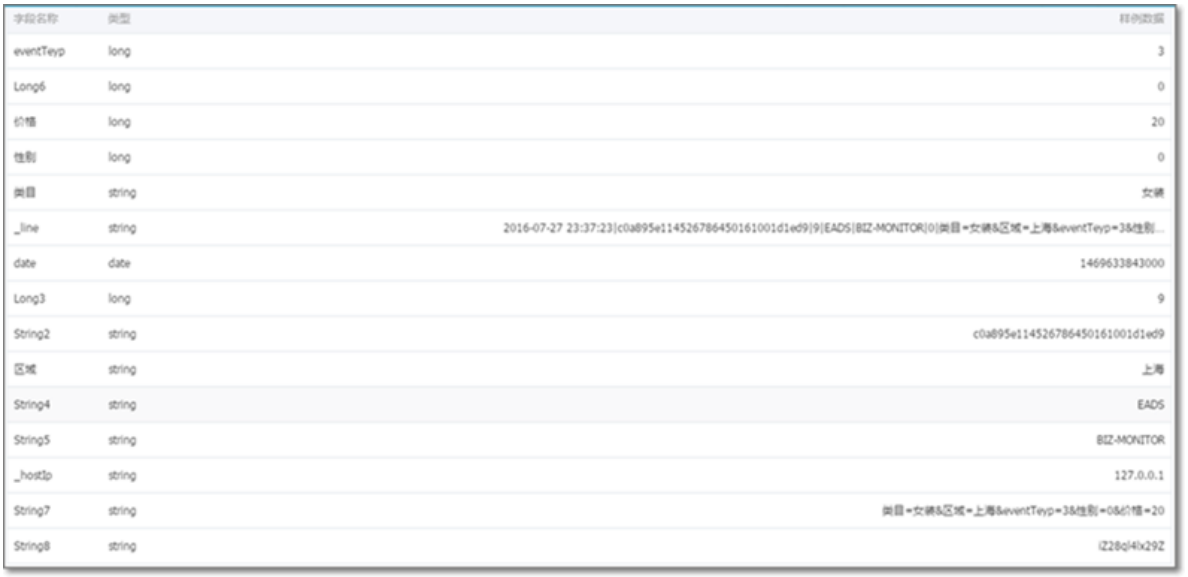
2. 拖拉形成如下所示积木块。

图 678: 自定义切分



3. 单击**日志切分预览**，结果如下图所示：

图 679: 切分预览



字段名称	类型	样例数据
eventTeyp	long	3
Long6	long	0
价格	long	20
性别	long	0
类目	string	女裤
_line	string	2016-07-27 23:37:23|c0a895e114526786450161001d1ed9|9|EADS|BIZ-MONITOR|0|类目=女裤&区域=上海&eventTeyp=3&性别...
date	date	1469633843000
Long3	long	9
String2	string	c0a895e114526786450161001d1ed9
区域	string	上海
String4	string	EADS
String5	string	BIZ-MONITOR
_hostIp	string	127.0.0.1
String7	string	类目=女裤&区域=上海&eventTeyp=3&性别=0&价格=20
String8	string	iZ28ql4lx29Z

4. 在进行数据清洗后，可以执行**保存**操作并单击**下一步**，即可进行数据集配置。

24.4.4 创建和设置报警控件

在满足设置的报警条件时，ARMS 可以通过邮件或短信的方式实时地进行报警。ARMS 的报警控件还支持**历史查询和删除**，可以查询日志中的数据在某个历史时间点出现过的异常。

背景信息

默认报警条件：

- 为了避免报警的信息爆炸，相同报警半小时内只发送两次报警消息。
- 如果5分钟内没有重复报警则会发送恢复邮件，通知数据恢复正常。
- 发送恢复邮件之后，报警的状态会重置。如果该报警再次出现，会认定为新的报警。

在 ARMS 中，报警控件本质是数据集的数据展示方式，所以在创建报警控件的同时，会创建一个对应的数据集，该数据集对应了报警控件的底层数据存储。

在控制台左侧菜单栏选择**监控管理** > **数据集管理**，找到相应的数据集可以查询具体数据，并获得 API 调用代码提示。

操作步骤

添加与配置报警信息

假设日志如下：

```
2016-11-14 15:56:46|9|EADS|BIZ-MONITOR|0|类目=男装&区域=杭州&eventTeyp=1&性别=1&价格=2140
2016-11-14 15:56:46|9|EADS|BIZ-MONITOR|0|类目=家居&区域=上海&eventTeyp=3&性别=0&价格=8305
```

以此日志作为数据源进行数据统计分析，制作一分钟内销售额低于最小设置值或高于最大设置值的报警控件。具体步骤如下：

1. 在控制台左侧菜单栏选择**监控管理** > **监控任务管理**。
2. 在任务列表找到前面一步创建的监控任务并单击右侧的**编辑**选项，单击**下一步**直到进入**数据集与报警配置**页面。

图 680: 数据集与报警配置



3. 在报警设置栏右上角单击**添加报警**。

图 681: 添加报警



4. 在**添加报警**对话框输入相关信息，点击**保存**，并单击**完成配置**。

图 682: 编辑报警



注意: 请仔细选择报警规则，**满足下述一条规则**和**同时满足下述规则**分别对应了报警条件的**或**与**且**的关系。

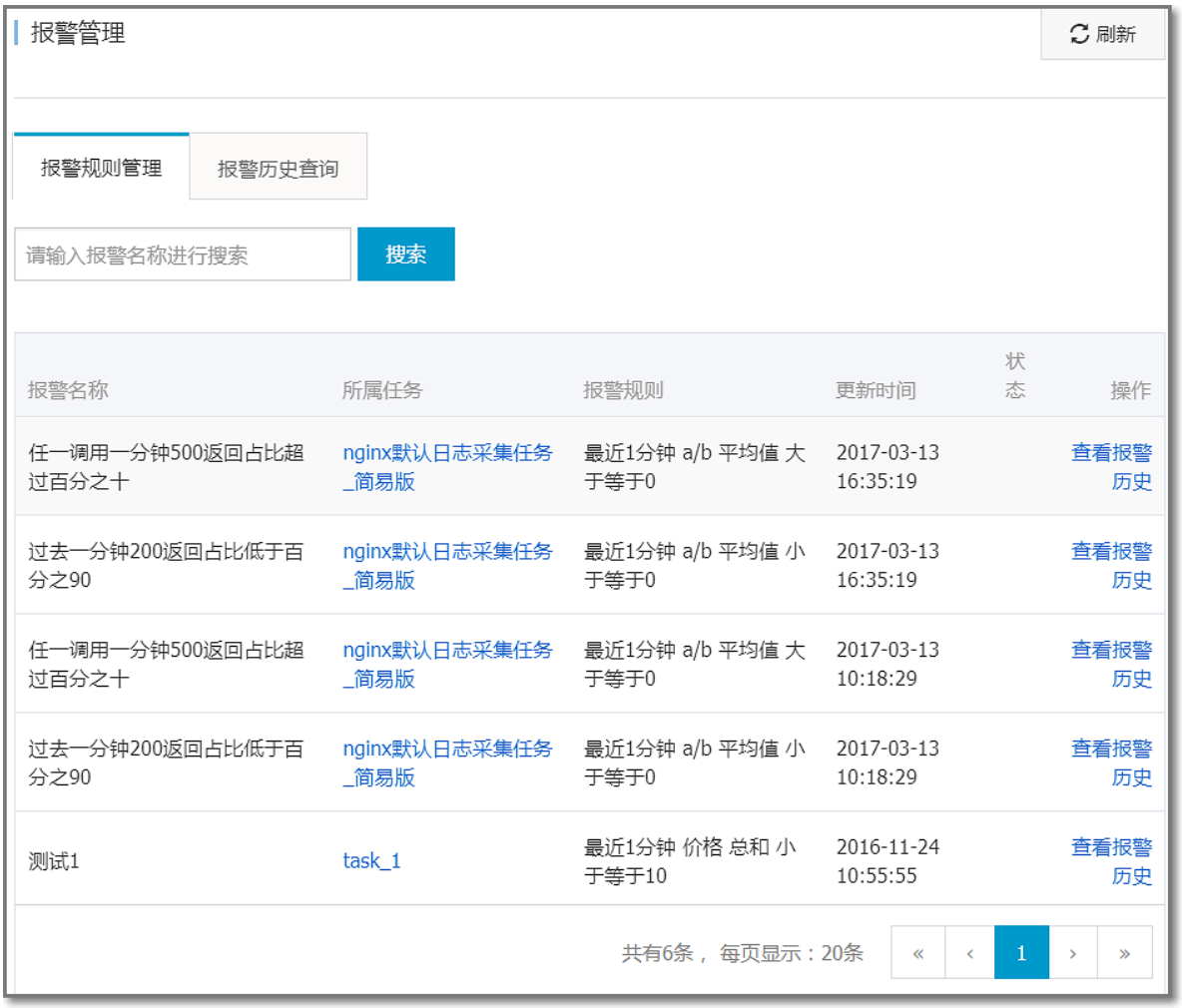
5. 回到监控任务列表页，单击**启动**选项，则此报警控件生效。
6. 在控制台左侧菜单栏选择**报警管理**，找到刚刚创建好的报警控件，单击右侧**查看报警历史**，即可查询所有报警数据。

图 683: 报警详情



7. 单击**搜索**查询报警数据。

图 684: 报警历史查询



说明: 在查询页面选择报警事件，可以批量删除报警历史记录。

24.4.5 创建和查询多维数据集

背景信息

多维数据集功能，是指将日志中的流式数据转化为结构化的模型数据存储，并提供页面以及 API 方式的查询。在 ARMS 中进行简单的操作，就可以获得 ARMS 系统关于多维数据分析优化的数据组织和存储模式。

多维数据集在概念上为一个超立方块（hypercube），而在物理存储上可以理解为一个多维数组。维的属性值被映射为多维数组的下标值或下标值的范围，而指标数据作为多维数的值存储在数据的单元中。维可以看作为自变量，指标数据为因变量。

- ARMS 中最多可以设置三个维度的数据集。
- 维度和维度之间是有层级关系的。比如，要查看第二个维度，必须要先选择第一个维度的属性。维度有点像一个树状结构。维度的定义需要规划，例如，第一维度可以是省，第二维度可以是市，第三个维度可以是区，指标数据则是市民消费情况。

图 685: 数据集



- 除非有特殊场景需求，两个完全不相关地维度最好不要放在一起定义，比如“地域”和“物品类型”。
- ARMS 提供下钻功能，指从汇总数据深入到细节数据进行观察或新增维度。钻取的目的是改变维度层次，变换分析粒度。

多维数据集的创建和查询需要一个自定义的监控任务，具体的监控任务创建可以参考配置数据源。

假设日志如下。以此日志作为数据源进行数据统计分析和存储，可以从**类目**和**区域**两个维度进行统计报表的价格指标查询。

```
2016-11-14 15:56:46|9|EADS|BIZ-MONITOR|0|类目=男装&区域=杭州&eventTeyp=1&性别=1&价格=2140
2016-11-14 15:56:46|9|EADS|BIZ-MONITOR|0|类目=家居&区域=上海&eventTeyp=3&性别=0&价格=8305
```

操作步骤

1. 在控制台左侧菜单栏选择**监控管理** > **监控任务管理**。
2. 在任务列表找到需要操作的监控任务并单击右侧的**编辑**选项，单击**下一步**直到进入**数据集与报警配置**页面。

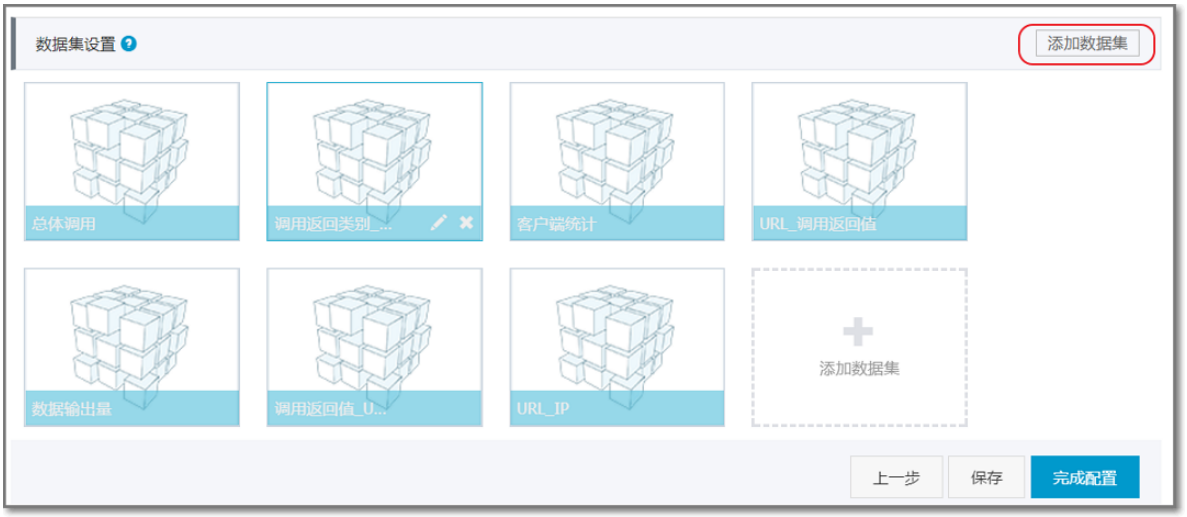
图 686: 数据集与报警配置



说明: 在日志的切分阶段，用户可以获得**类目**和**区域**字段。

3. 在页面右下方单击**添加数据集**。

图 687: 数据集设置



4. 在**添加数据集**对话框，输入相关信息，单击**保存**，并单击**完成配置**。

图 688: 添加数据集



5. 在监控任务列表页**启动**自定义监控。

   完成以上步骤后，多维数据集就创建完成了。更多关于多维数据集的高级使用和查询，请参考查询数据集文档。

24.5 管理监控任务

本节介绍如何查看监控任务详情并对监控任务进行管理。

24.5.1 查看监控任务详情

ARMS提供三个维度的监控任务信息：监控任务详情、数据集详情、任务运行详情。通过查看相关信息可以进行监控任务的运维及问题判断。

在控制台左侧选择**监控管理** > **监控任务管理**，进入**监控任务管理**页面，点击相应的监控任务名称右侧的**浏览**，即可查看任务详情。

监控任务详情

监控任务详情主要显示任务相关的配置信息，包括数据源的配置信息、样例日志、日志切分模型、数据集与报警配置。

配置信息只能查看，不能修改。

数据集详情

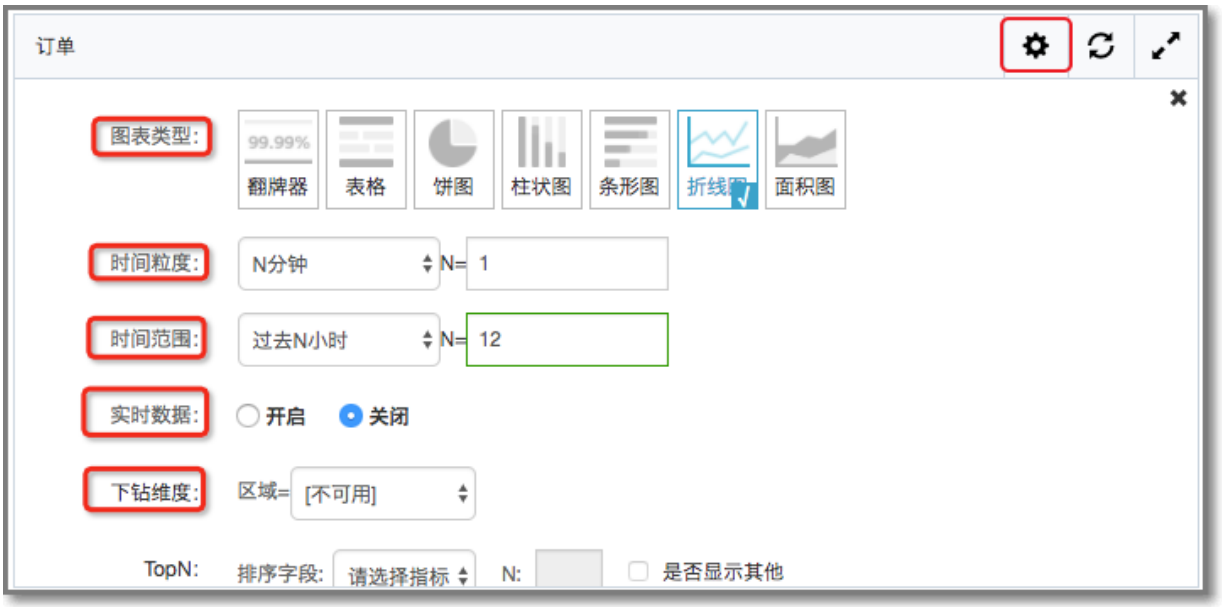
数据集详情展示该任务下所有的数据集。在该页面数据集都以数据图表的形式进行展现。



说明: 每配置一个报警也会生成一个数据集。

每一个数据集默认以折线图的形式进行展现。如果需要进行数据集的探索，可以点击右上角齿轮进行配置：

图 689: 数据集图表配置



- 图表类型：目前支持7种类型的图表。
- 时间粒度：默认为1分钟。
- 时间范围：可以理解为图表的时间跨度。
- 实时数据：默认为关闭。如果开启，图表数据将会定时刷新，周期为1分钟。
- 指标：用于显示的指标。

下图为配置的效果：

图 690: 配置效果图



任务运行详情

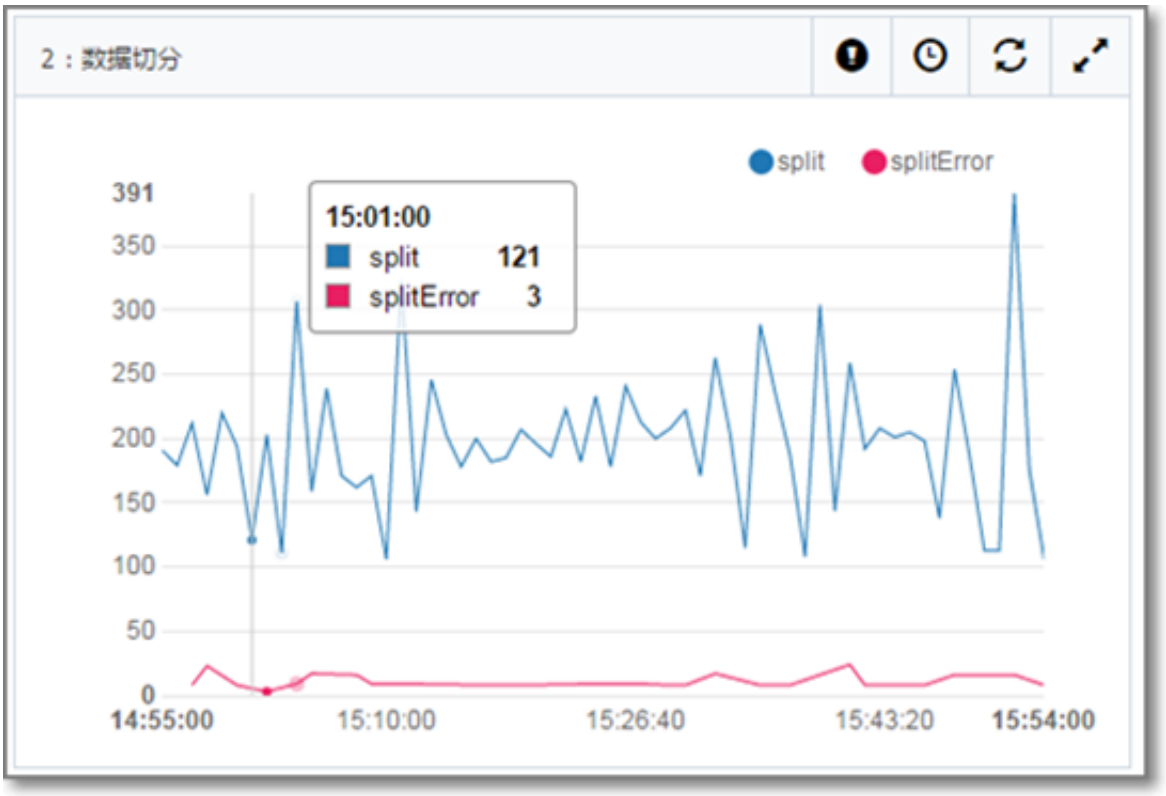
任务运行详情展示了该任务的详细运行过程，任务的运行过程包含：数据抓取、数据切分、数据聚合、数据存储。目前监控已经覆盖了除数据存储之外的三大过程，如下图所示：

图 691: 任务运行详情



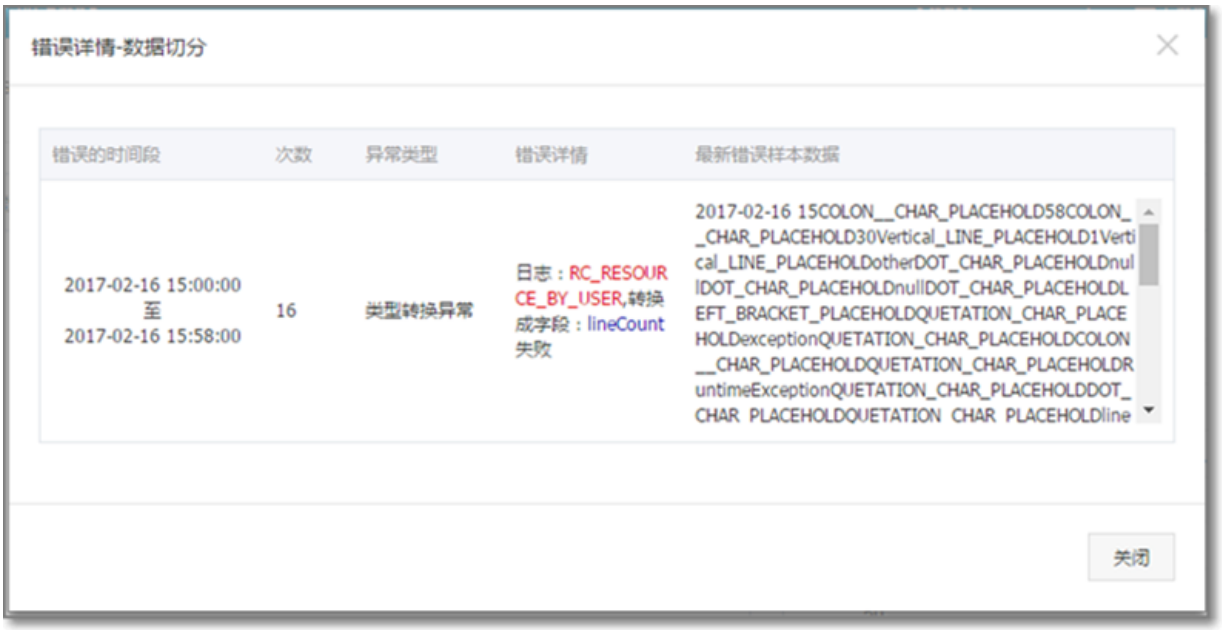
通过数据抓取、数据切分、数据聚合图表，可以详细的看到每分钟成功处理的行数和失败的行数，如下图所示为数据切分的过程：

图 692: 切分报错



上图显示日志切分过程存在错误，可以通多点击图形右上角的“！”显示错误详情。

图 693: 报错详情



24.5.2 监控任务操作

在监控任务管理页面，可以对每个监控任务进行启动、编辑、删除、复制等操作。

在控制台左侧菜单栏选择**监控管理** > **监控任务管理**，进入**监控任务管理**页面。

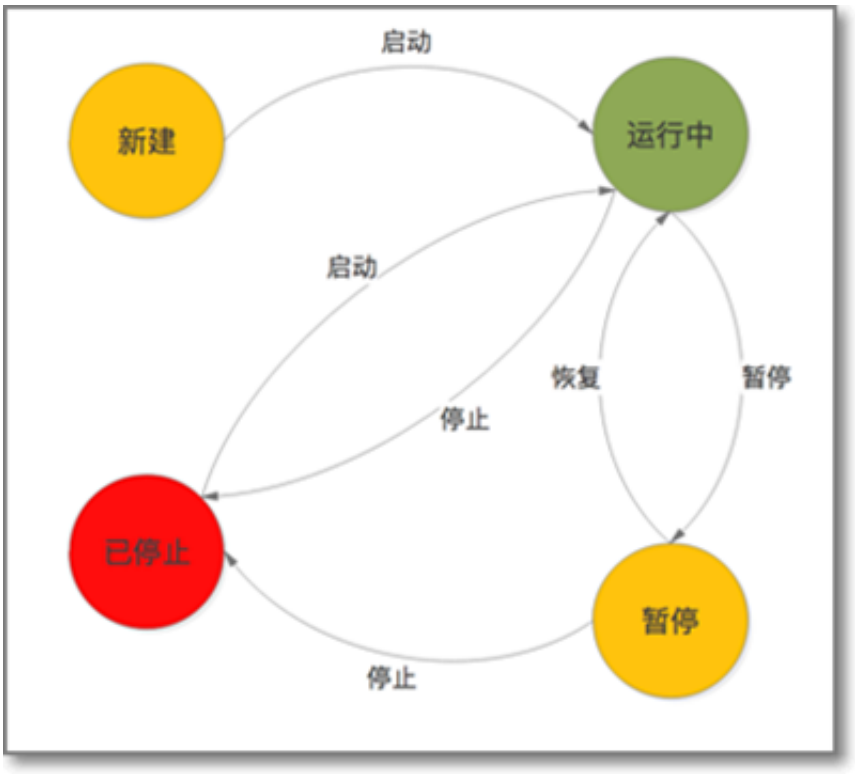
监控任务状态说明

监控任务一共四种状态：新建、运行中、已停止、暂停。

状态	描述
新建	首次建立任务显示该状态，新建的任务可以单击**启动**。
运行中	表示当前任务正在运行，“运行中”的任务可以单击**停止**与**暂停**。
已停止	表示该任务停止运行，已停止的任务可以单击**启动**。
暂停	单击**暂停**之后任务变为该状态，“暂停”的任务可以单击**恢复**与**停止**。

下图为任务状态转化关系：

图 694: 监控任务状态



监控任务操作说明

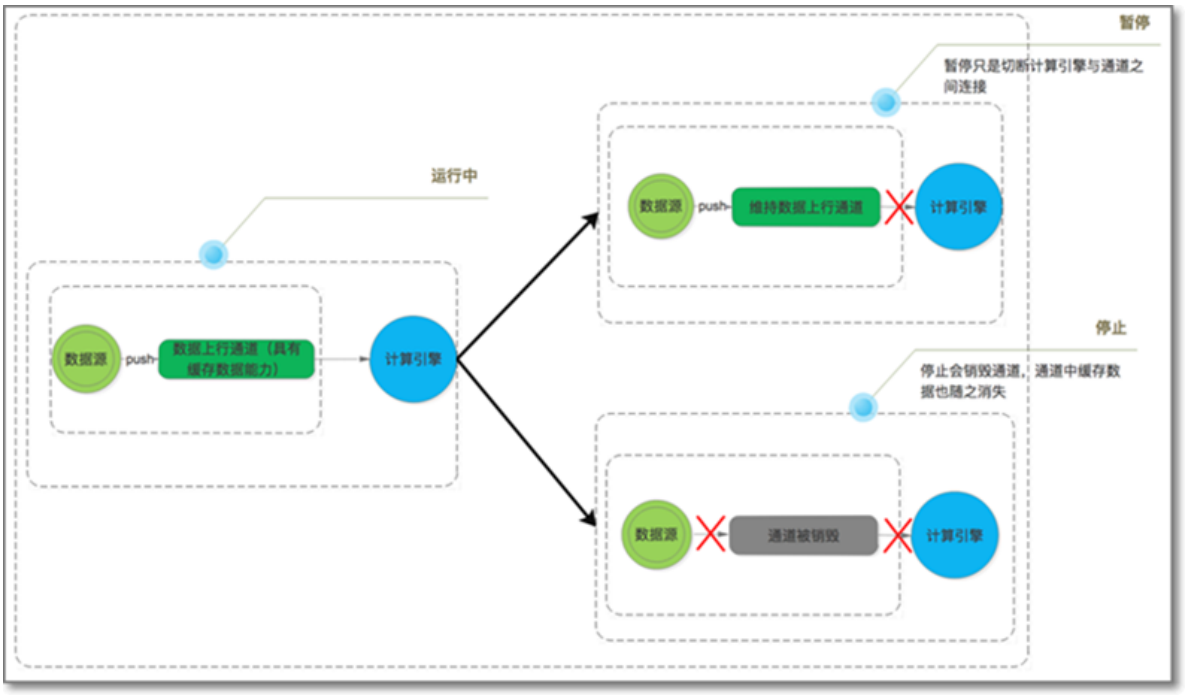
监控任务的操作主要包括：启动、停止、暂停、恢复、删除、复制任务、导出模版等操作。

- 启动、停止、暂停、恢复

  暂停操作将会切断计算引擎与通道之间的连接，单击**恢复**将会重新建立连接，并继续上次的消费位点继续消费数据。

  停止操作销毁了通道。

  图 695: 暂停和停止



- 删除

  任务在“运行中”、“暂停”是不允许删除的，“新建”与“已停止”的任务可以删除。

  删除任务的同时将删除任务相关的配置、数据集、报警。

- 复制任务

  对当前任务进行拷贝用于新建的监控任务。进行拷贝之前需要填写新生成任务的名称。

- 导出模板

  一个完整的任务包含数据源配置、清洗配置、数据集或报警配置；导出模板中包含了清洗配置（日志样例与清洗规则）、数据集配置。导出的模板为 JSON 文件。

操作错误提示说明

在进行任务的启动、停止、暂停与恢复操作时，可能会抛出异常。以下为错误提示和对应的解决方法：

提示	详细描述	解决方法
成功	操作成功	无
数据源配置不能为空	当前任务没有配置数据源，如未选择数据源类型、未选择数据源等。	进行数据源配置

提示	详细描述	解决方法
切分配置为空	样例日志不存在或者没有对样例日志进行切分	检查切分配置并进行配置
数据集或者报警不能为空	用户没有配置任何业务逻辑	在任务下配置报警或者数据集即可。
操作频繁	系统默认限定操作间隔为15s	等待15s之后重试
数据集中字段存在与切分模型之中	数据集中选择的字段在切分模型中无法找到	检查切分模型或者数据集
无法分配资源	系统无法为该任务分配资源	请联系管理员
错误	其他原因，如网络原因、系统故障等。	请联系管理员

24.6 查询监控结果

本节描述数据集以及报警监控相关的管理操作。

24.6.1 管理报警

ARMS控制台提供了查询监控任务中配置的报警规则、报警历史等功能。

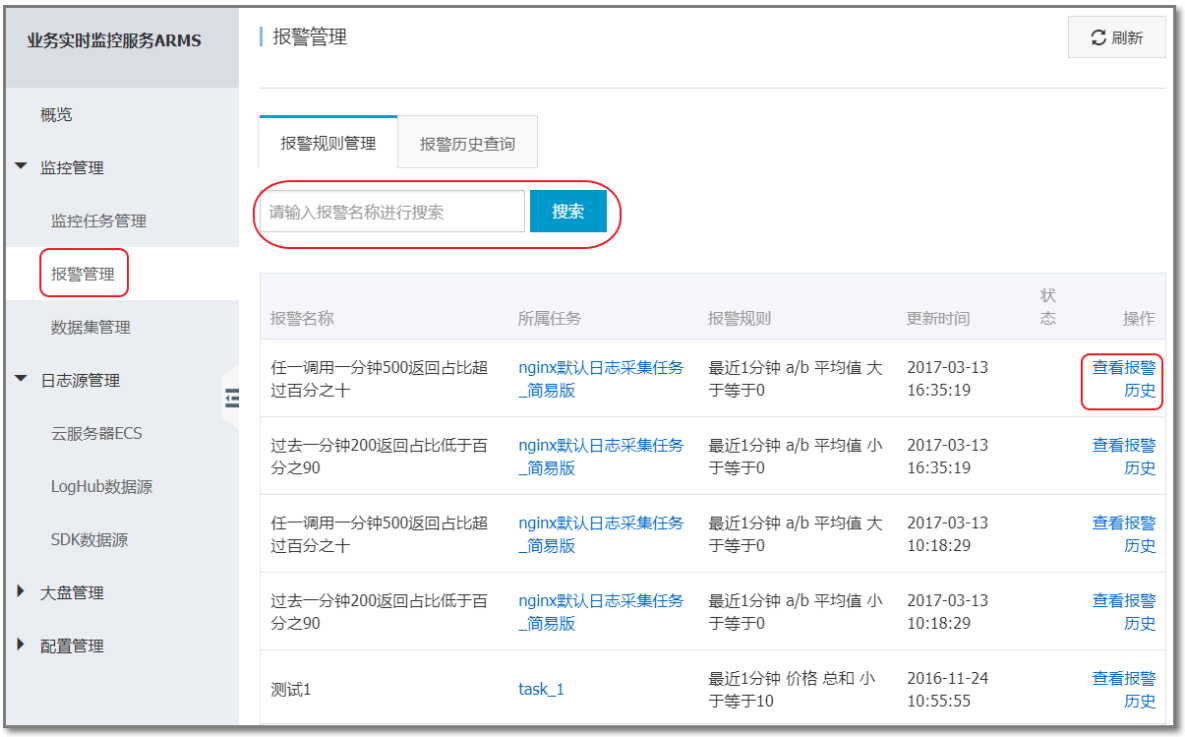
操作步骤

查询报警规则

1. 在控制台左侧选择**监控管理** > **报警管理**，进入**报警管理**页面。
2. 在**报警规则管理**页面，可以输入报警名称进行搜索。

   报警列表展示了每一个报警的名称、所属的监控任务、报警的规则等信息。单击报警信息右侧的**详情**可以进入报警历史栏查看该报警下的报警历史信息。

   图 696: 报警管理



查询报警历史

3. 单击**报警历史**Tab切换到报警历史页面。
4. 选择监控任务以及对应的报警，单击**搜索**。

图 697: 报警历史



24.6.2 查询数据集

报表、报警和数据查询是数据集（DataSet）的展示，而数据集是上述控件的数据本质。ARMS 以数据集的方式将数据进行展示，屏蔽了底层数据存储结构、压缩存储、数据备份、数据查询机制等细节，因此您不需要了解数据是如何存储在存储介质，而只需关注数据本身价值的实现。

操作步骤

1. 在控制台左侧菜单栏选择**数据集管理**，选择需要查询的数据集名称，单击**数据查询**。

图 698: 数据集管理



说明: 这里的数据集管理不仅有之前创建的**多维数据集**，还有**报警控件**相关的数据集，数据集名称与控件名称相同。这里以多维数据集做为举例说明。

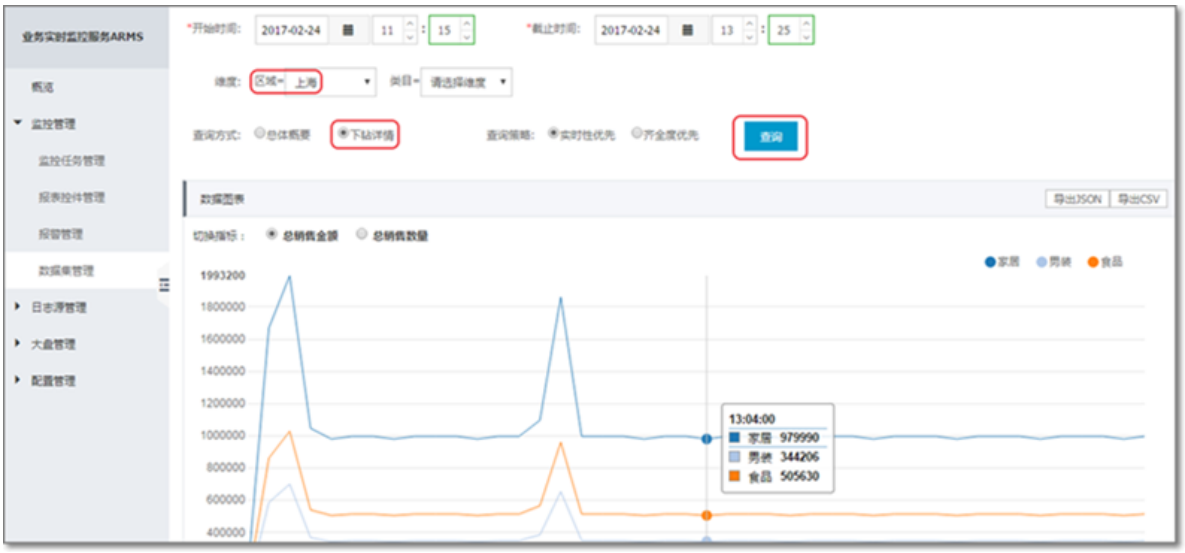
2. 在**数据集查询设置**页面，选择需要的维度，单击**查询**，数据会以**数据图表**的方式进行展示。

图 699: 数据集查询



3. 如果存储的是**多维度**数据，可以选择**下钻详情**并单击**查询**，查看各个维度数据详情。

图 700: 下钻详情



4. 也可以选择查询策略进行查询，查询策略分为**实时性优先**和**齐全度优先**。

图 701: 查询策略



说明: 由于现在的聚合数据按秒级写入HBase，所以当前一分钟的数据不是该分钟的完整数据。选择**齐全度优先**时，ARMS会忽略该分钟数据。未来**齐全度优先**会允许通过设置数据到达水位的方式进行个性化定制；当您选择**实时性优先**时，ARMS会将该分钟的当前值（完全数据，实际的到达数据值）进行展示。

24.7 定制大盘

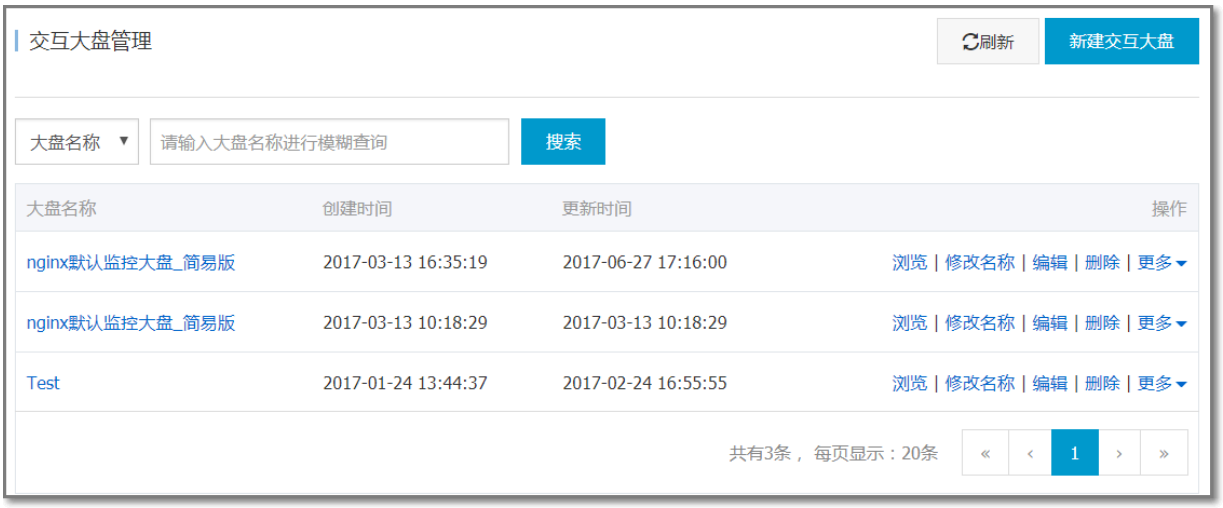
本章节介绍如何定制交互式大盘。

24.7.1 交互式大盘简介

本节介绍交互式大盘的界面。

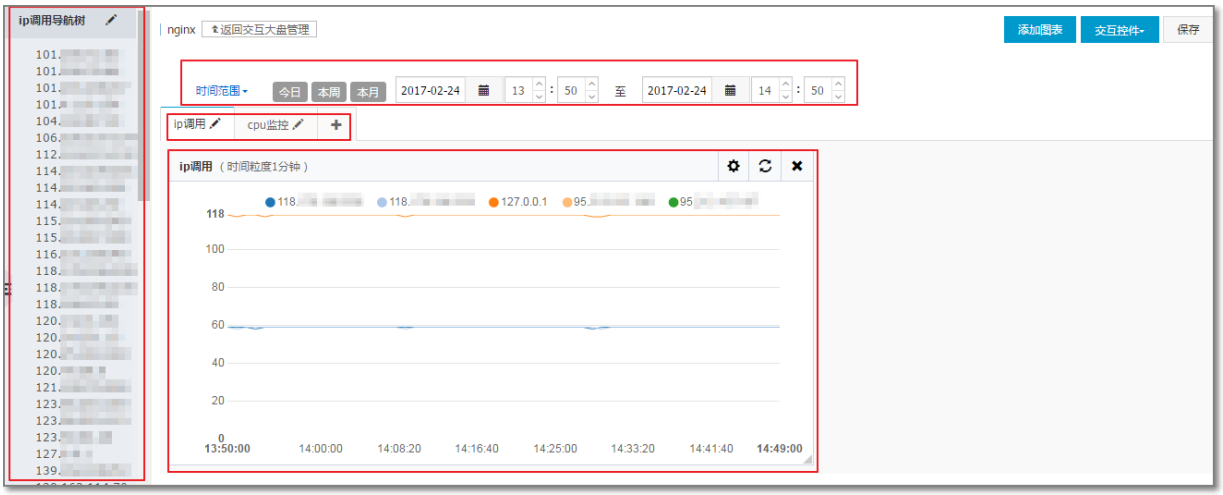
在控制台左侧选择**大盘管理** > **交互大盘管理**，进入交互大盘管理页面，在大盘列表中可以看到创建的所有交互大盘。

图 702: 交互大盘



交互大盘主要分为导航树、时间选择、标签栏和数据集展示四个模块。下图是一个完整的交互大盘示例。

图 703: 大盘示例



- 左侧是**导航树**，根据用户数据自动生成，还支持按照相应的 JSON 格式手动导入导航树。最多可以设置三层的导航树，逐级下钻选择查询维度，然后单击，就可以在右侧的数据集展示模块看到相应数据。
- 右侧由上至下分别为时间选择模块、标签栏模块和数据集展示模块。
  - **时间选择模块**：支持选择时间范围，也提供了类似今日、本周和本月的常用时间范围。
  - **标签栏模块**：允许添加多个不同的标签。本示例中添加了“IP 调用”和“CPU 监控”两个标签栏，方便在查询不同数据时快速切换。
  - **数据集展示模块**：是交互大盘的核心模块，会根据选择的导航树维度、时间范围显示对应的标签栏下的所有数据图表。

24.7.2 创建交互式大盘

创建交互式大盘包含两个步骤：新建交互式大盘和配置大盘数据。

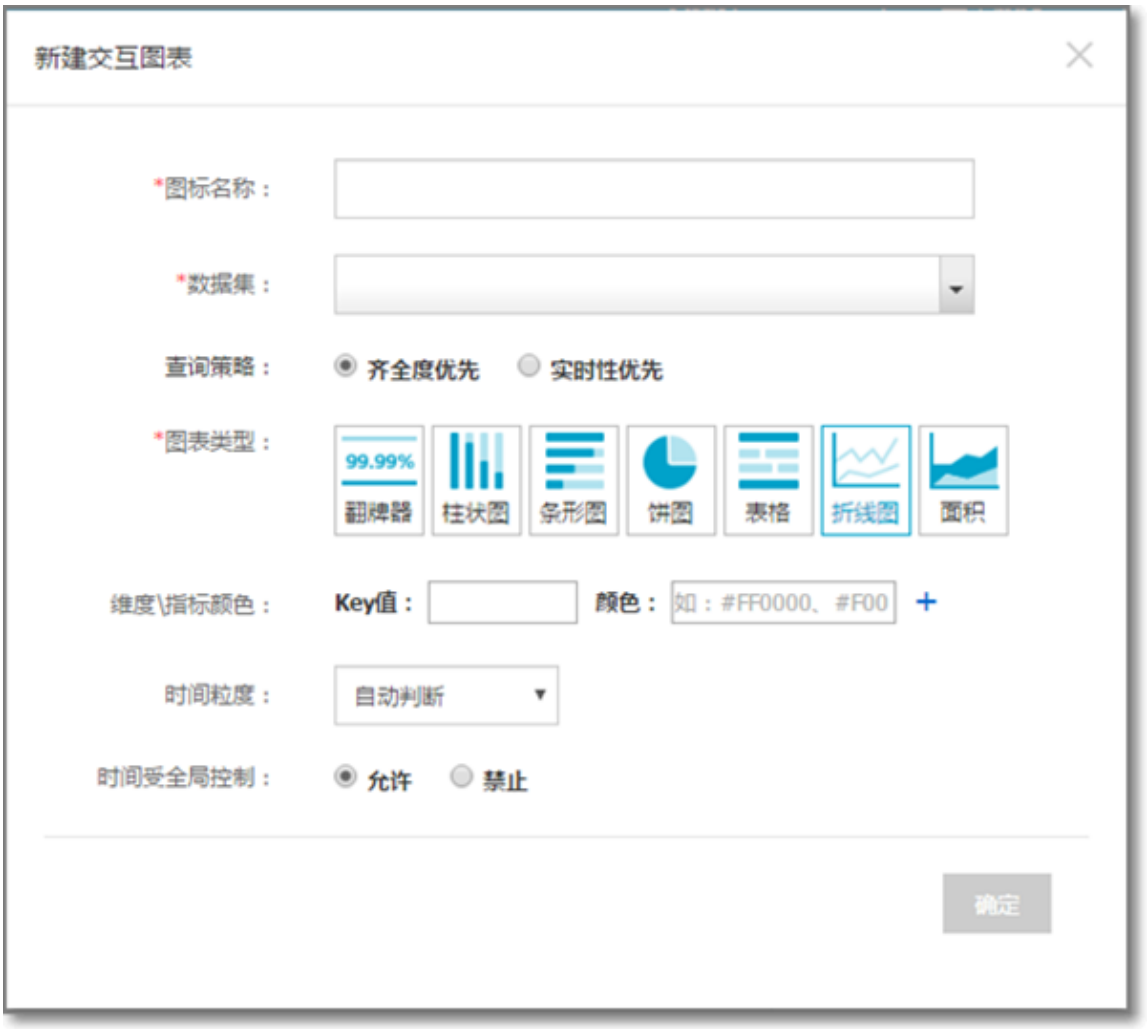
操作步骤

1. 在 ARMS 控制台左侧菜单栏选择**大盘管理** > **交互大盘管理**。
2. 单击右上角**新建交互大盘**。
3. 在**新建交互大盘**对话框输入交互大盘名称并单击**确定**。

   说明: 新建的交互大盘没有任何内容。

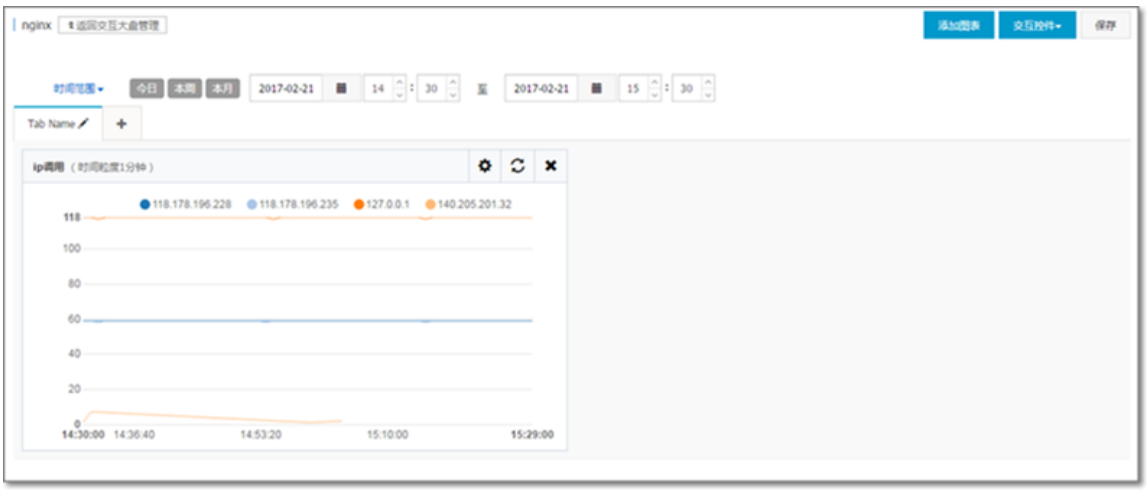
4. 添加数据集。
   a) 在交互大盘管理页，单击大盘名称右侧的**编辑**按钮，进入编辑页面。
   b) 单击右上角**添加图表**。
   c) 在**新建交互图表**对话框，输入图表名称、在**数据集**的下拉选项中选择相应数据集，选择图表类型为折线图，单击**确定**。

   图 704: 新建交互图表



这时候就能在数据集展示模块中看到响应的数据集折线图展示。

图 705: 数据集



5. 添加导航树。

a) 在大盘编辑页面，单击右上角**交互控件**按钮，选择导航树控件，弹出**导航树窗口**，如下图。

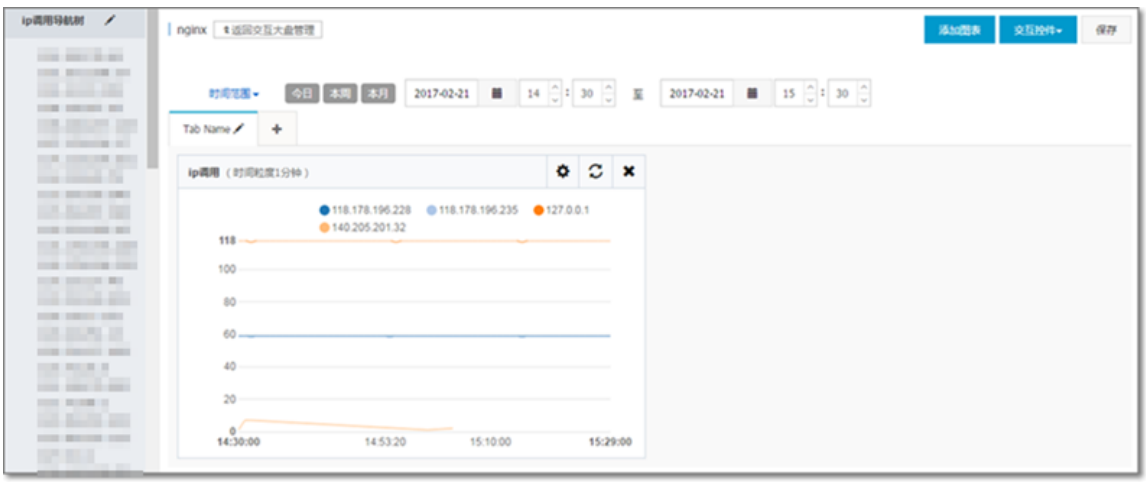
交互控件

图 706: 导航树



b) 填标题名，选择数据集，这时候会自动把该数据集的多维度遍历值导入到数据文本框。单击**确定**就可以在页面左侧看到对应的导航树。

图 707: 导入导航树



6. 关联数据集和导航树。

   a) 在数据集展示模块找到对应的数据集，单击齿轮的设置按钮，弹出**编辑交互图表**窗口，如下图所示。

   图 708: 编辑交互式图标

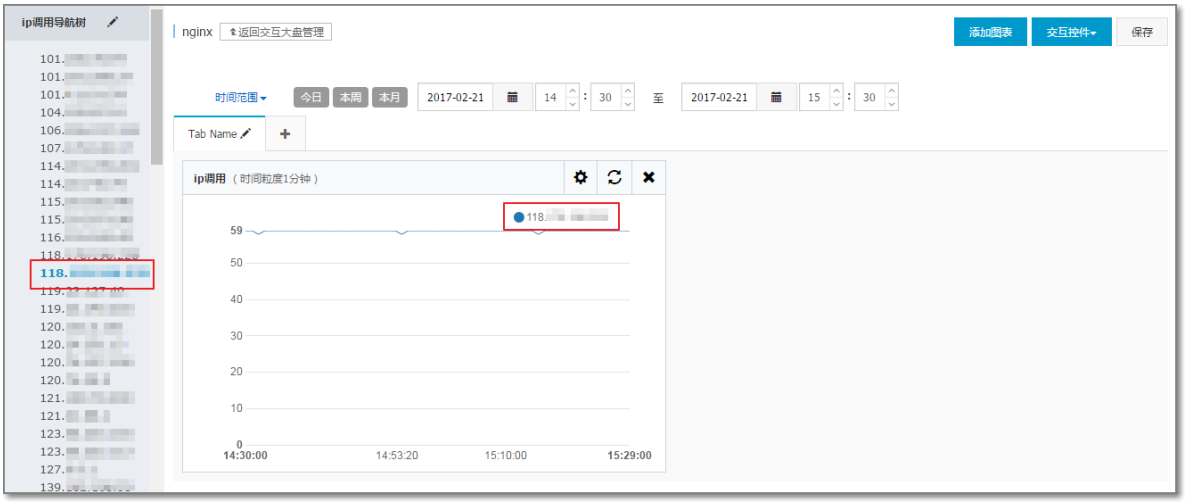


   b) 找到维度，选择关联，在下一下拉框选择对应的导航树，然后在最后一个下拉框选择关联的导航树的维度。单击确定，这时候数据集的维度和导航树的维度已经关联起来。

7. 查看数据集对应的显示。

选择导航树中的不同维度值，就可以在数据集中看到对应的数据展示。

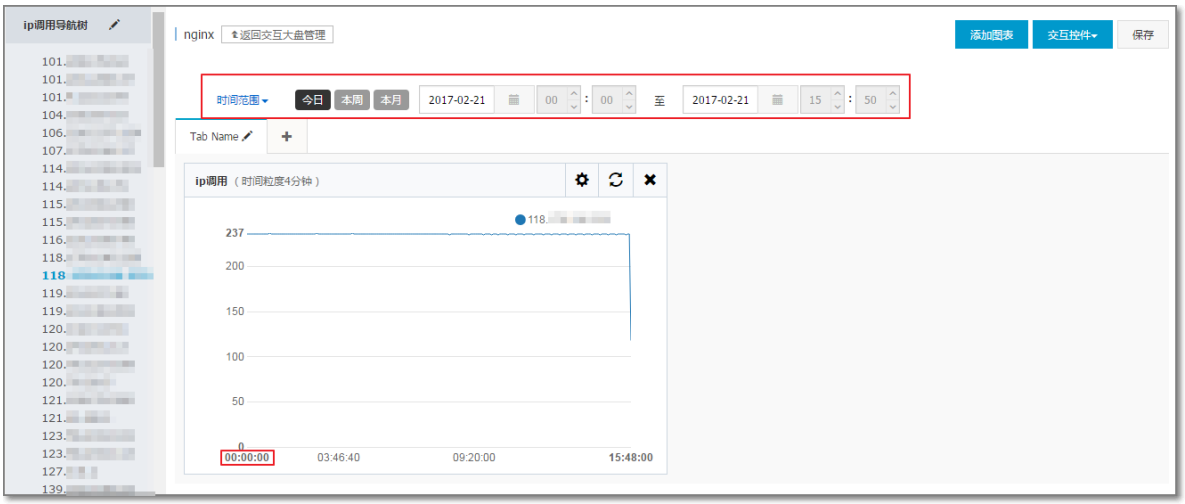
图 709: 数据展示



8. 查看不同时间的数据集数据。

在时间选择模块，可以选择今日、本周、本月，或者调整开始时间与结束时间，就可以看到不同时间范围的数据。

图 710: 时间模块



9. 大盘保存。

配置好大盘之后，单击右上角的**保存**按钮，可以保存当前的配置。此外，每隔十秒钟，系统还会自动保存一次，防止丢失正在编辑的数据。

说明: 大盘中可以对图表的位置、大小等进行灵活地配置。

- 大小调整

  添加的图表右下角，按住拖动即可调整大小。

- 位置调整

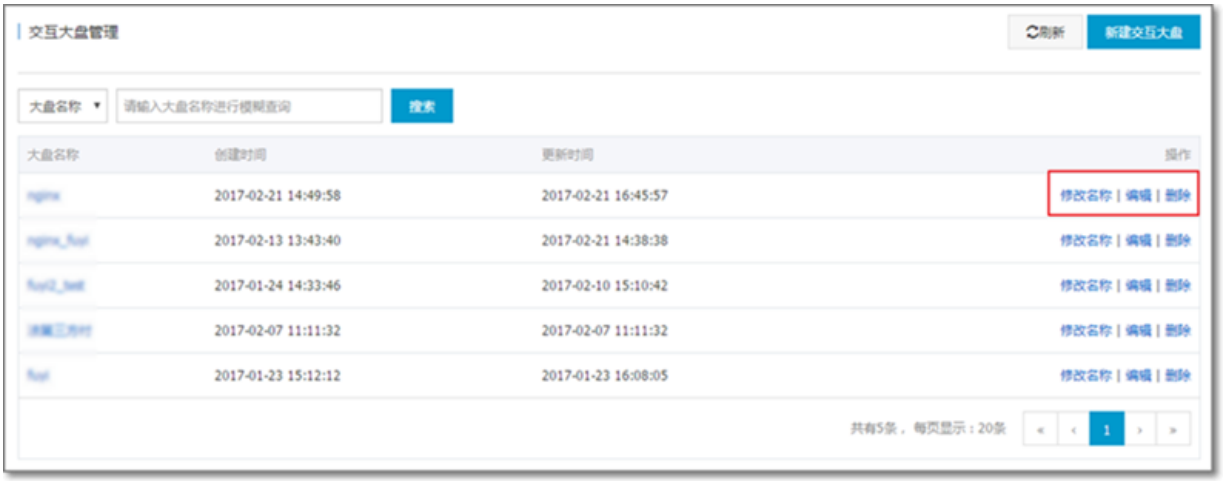
  对需要移动的图表按下鼠标左键，移动鼠标，释放到需要移动的位置。

24.7.3 管理交互式大盘

本文描述如何对交互式大盘进行编辑、删除、播放等操作。

ARMS 支持对每一个大盘进行修改名称、编辑与删除操作，同时支持基于大盘名称的搜索。

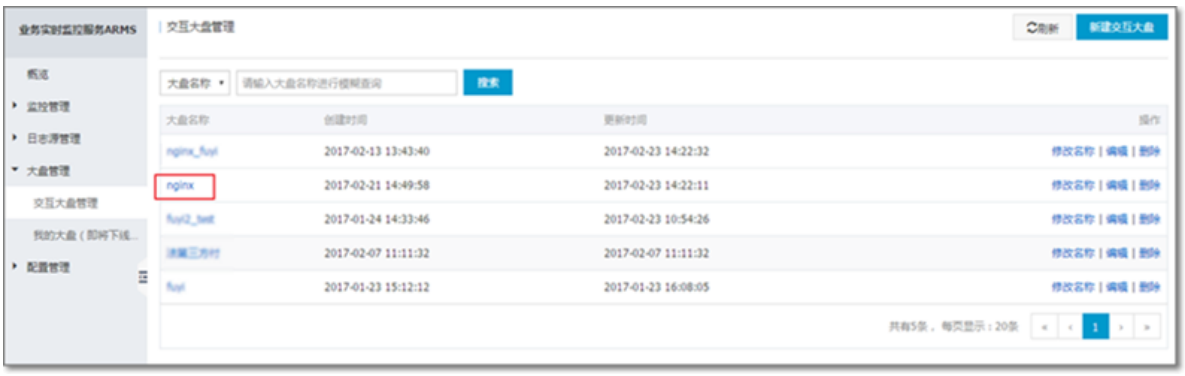
图 711: 大盘管理



播放模式与全屏模式

1. 在大盘列表单击需要操作的大盘名称，进入播放模式。这时只允许查看大盘，不允许编辑。

图 712: 播放模式



2. 在播放模式下，单击右上角的全屏，弹出**全屏设置**窗口，如下图所示。

图 713: 全屏设置



3. 选择全屏的 Tab，单击**确定**，进入全屏。

目前每分钟刷新一次，如果有特殊需求，欢迎联系我们。

24.8 管理系统配置

本节介绍如何进行ARMS映射表管理以及联系人管理。

24.8.1 映射表管理

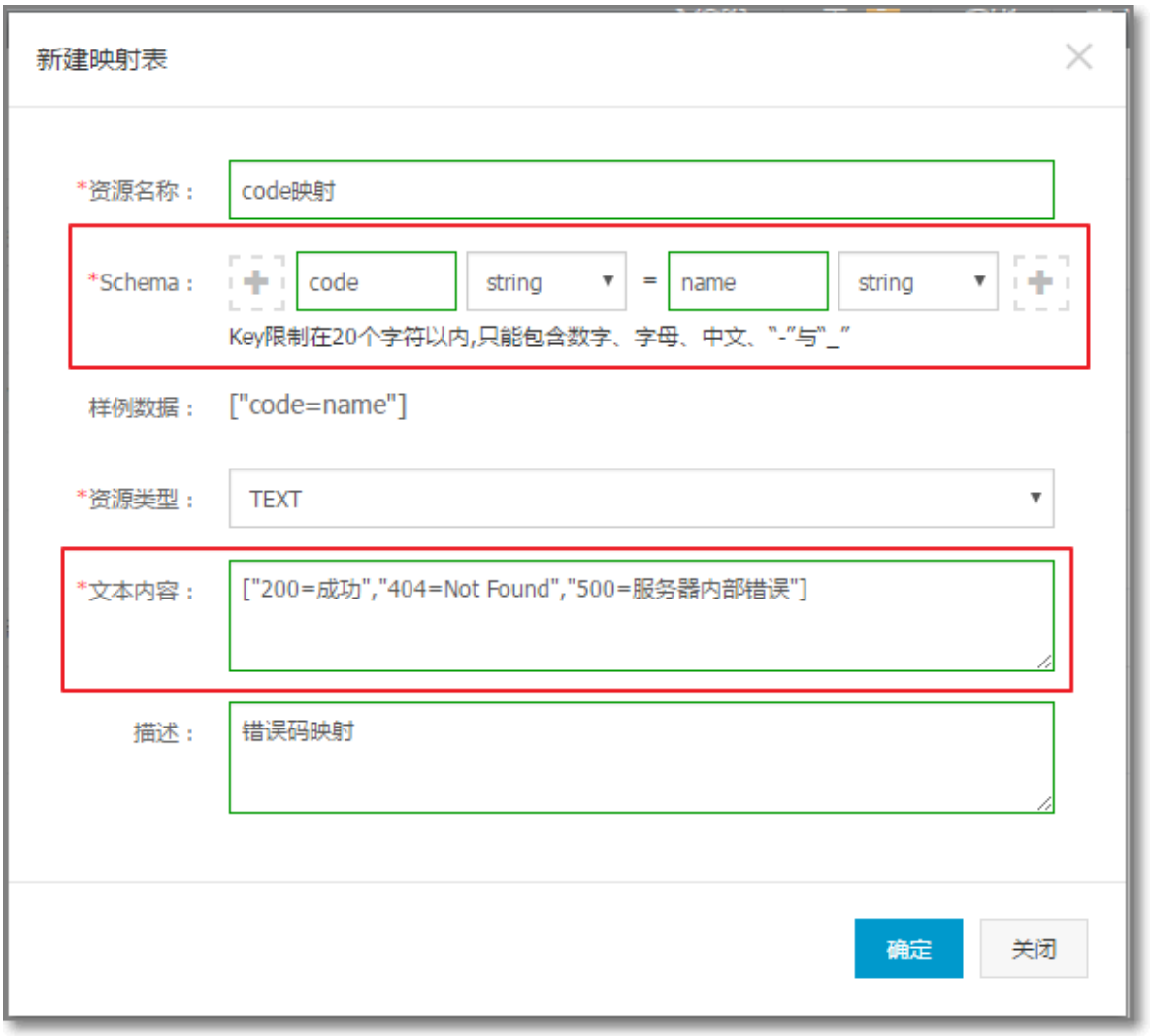
映射表主要用于用户存储静态数据，其主要功能包含：用户在进行日志清洗后，将清洗出的字段通过进行静态 Join 获取需要的目标字段;用户在进行数据集查询时，可以结合维表进行组合查询。下文介绍如何创建、编辑映射表。如何更好地使用映射表请参考映射表的使用。

操作步骤

1. 在控制台左侧选择**配置管理** > **映射表管理**，进入**映射表管理**页面。
2. 在页面右上角，单击**新建映射表**。

如下图新建了一个名为“code映射”的映射表，其 Schema 为一对一的映射关系，即将一个为 code 的字段转化为一个 name 的字段：

图 714: 映射表管理



下图的映射表为多对1的映射关系，利用经度+纬度转化为地址：

图 715: 映射关系



3. 编辑映射表。

   在映射表管理界面中的映射表列表中，单击需要编辑的任务的**编辑**操作选项，即可进入编辑模式。

图 716: 编辑映射表



编辑模式下不能修改 Schema 和资源类型。如果资源类型是 TEXT，可以修改文本内容（红色区域）。修改之后，依赖映射表的任务或者其他模块将立即生效。

24.8.2 联系人管理

在ARMS控制台可以对联系人信息进行管理。

背景信息

联系人管理包括添加联系人、编辑联系人、删除联系人以及如何搜索联系人。主要介绍如何添加联系人，其他管理操作仅做说明。

操作步骤

1. 在控制台左侧选择**配置管理** > **联系人管理**，进入**联系人管理列表**页面。

2. 选中**联系人**页签，单击右上角**新建联系人**。

图 717: 联系人



3. 输入用户姓名、手机、邮箱。

图 718: 新建联系人



4. 单击**确定**。

5. 编辑联系人。单击对应联系人操作栏的**编辑**选项，即可进入编辑模式。

图 719: 编辑联系人



6. 删除联系人，支持单个联系人的删除和批量删除。
7. 联系人搜索，只支持按联系人名称进行模糊搜索。

24.8.3 联系人分组管理

本文介绍如何通过控制台管理联系人分组。

背景信息

联系人分组管理包括添加、编辑、查看、删除及搜索联系人分组。主要介绍添加联系人分组，其他管理操作仅做说明。

操作步骤

1. 在控制台左侧选择**配置管理** > **联系人管理**，进入联系人管理页面。
2. 选中**联系人组**页签，单击右上角**新建联系组**。

图 720: 联系人分组



3. 填写组名、选择联系人。

图 721: 新建分组



4. 单击**确定**。

后续操作

编辑联系人分组

单击对应分组栏右侧的画笔按钮，即可进入编辑模式。

查看联系人分组

单击对应分组栏右侧的展开按钮，即可查看对应分组下的联系人。

图 722: 查看分组



说明: 如上图所示，您可以在展开模式下删除分组下的联系人，也可以直接删除整个组。

注意: 请谨慎删除联系人组，删除联系人组请确保没有正在运行的任务，否则可能导致报警等功能失效。

24.9 最佳实践

24.9.1 映射表的使用

映射表主要用于用户存储静态数据，其主要功能包含：

1. 用户在进行日志清洗后，将清洗出的字段通过进行静态 Join 获取需要的目标字段。
2. 用户在进行数据集查询时，可以结合维表进行组合查询。

静态 Join 案例

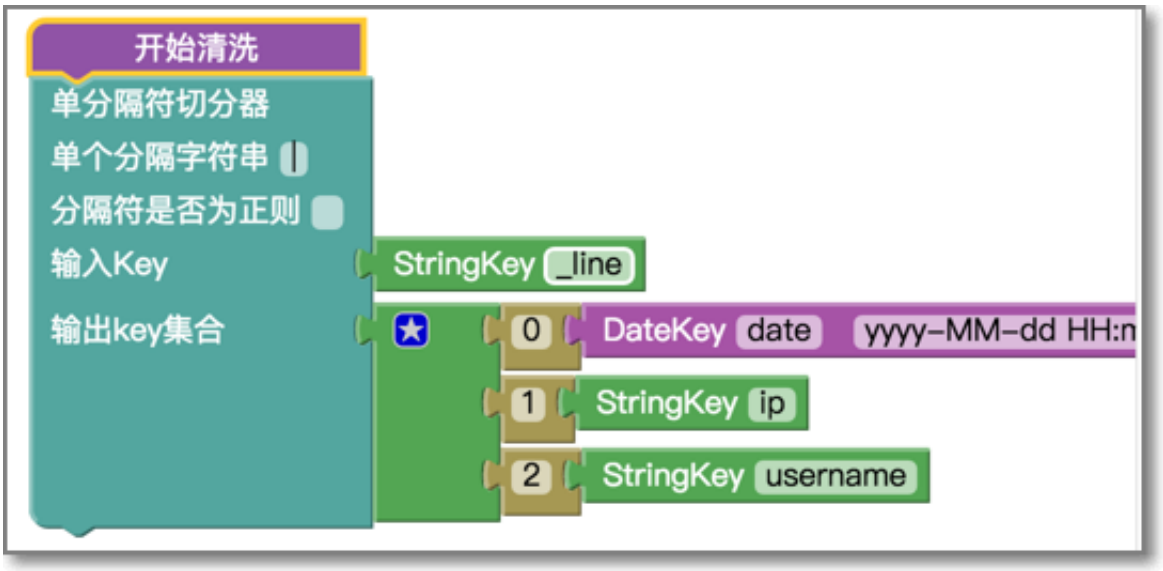
映射表的新建与使用

假设用户日志格式如下：

```
2017-01-09 16:02:49|42.**.**.**|Kevin.yang
```

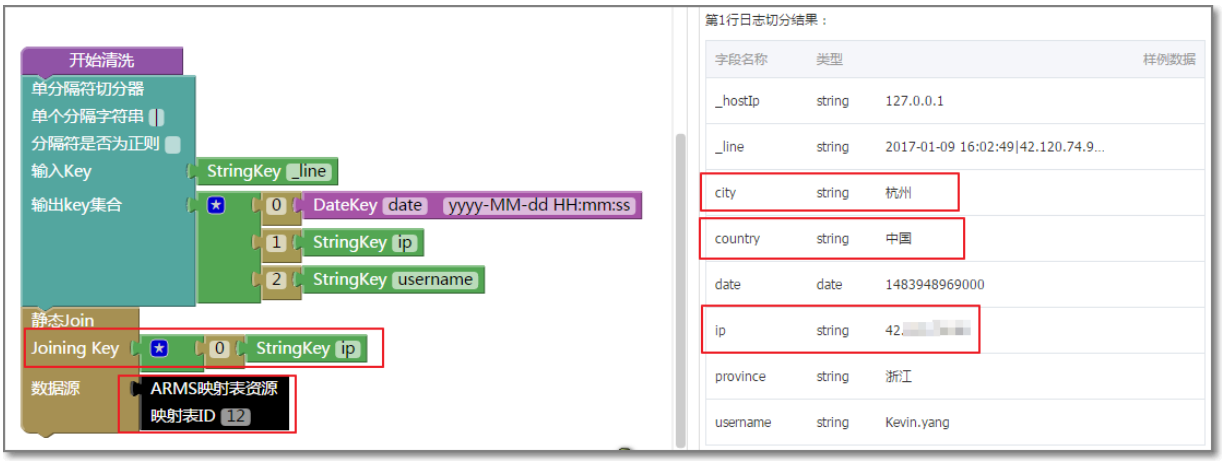
如下图为原始切分逻辑，切分后的字段分别为 date、ip、username。

图 723: 原始切分逻辑



现在业务需求是统计每分钟每个国家的用户访问量，但是切分模型没有相关的国家字段。这时候通过映射表可以进行 Join ，映射表中保存了 IP 与国家、省份、城市的映射关系，如下图所示为结合映射表之后的逻辑：

图 724: 结合映射表切分逻辑



通过上图可以发现，国家、省份、城市这三个字段根据原始的 IP 字段获得了，那么上图中“映射表 ID”是怎么设置的呢？

1. 在控制台左侧菜单栏选择**配置管理>映射表管理**，单击右侧**新建映射表**。
2. 填写映射表名称。

图 725: 新建映射表



3. 填写映射表的 schema 信息：针对上述需求 ip 转化为国家、省份与城市，其 schema 信息详相当于源字段信息（名称、类型）与目标字段信息（名称、类型）的映射关系。

   注意: 类型仅仅支持 string、long、double 类型。

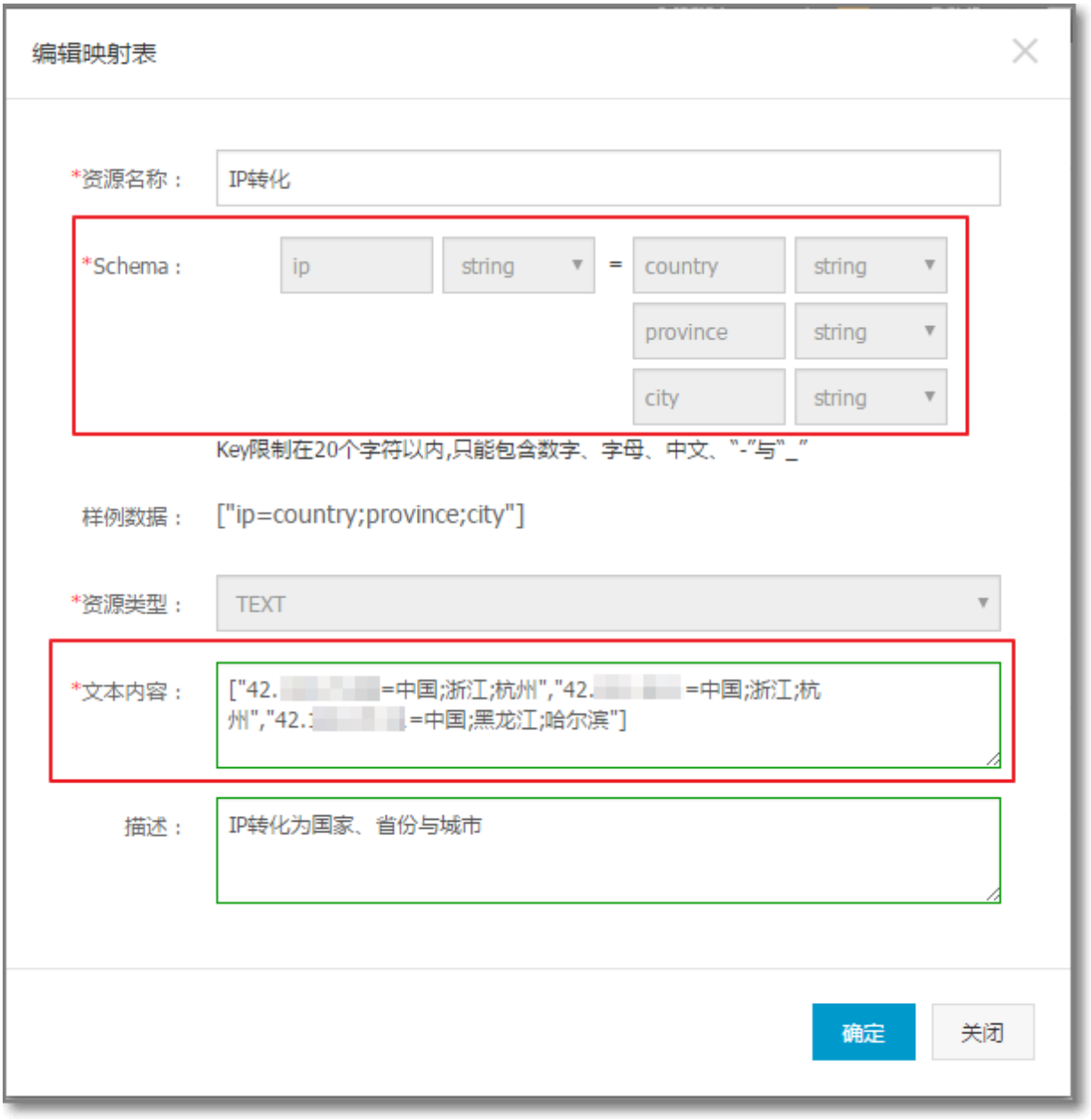
4. 选择映射表的资源类型（目前只支持文本类型 TEXT，后续会支持其他接入方式）。
5. 如果映射表的类型为文本类型，需要添加文本内容，文本内容可以参考样例数据进行录入，如图针对上述需求文本中只录入了一个 ip 的映射关系，文本内容必须严格按照 schema 设定的形式，否则无法保存。
6. 映射表保存后系统生成一个唯一的维表 ID ，填入上述的积木块中即可。

映射表数据更新

针对上述的案例，刚刚映射表中的文本中只配置了 IP 为42...** 的映射关系，上述的任务已经运行起来了，但是实际日志中包含其他 IP，利用刚才建立的映射表进行静态 Join 岂不是有问题的？

不用担心，您无需停止任务，只需要更新一下映射表里面的文本数据即可。

图 726: 编辑映射表



- 修改映射表时，映射表的 schema 与资源类型不能修改。
- 如上图所示当资源类型为文本类型时，修改文本内容即可。

基于映射表进行数据集的组合查询

在控制台左侧菜单栏选择**监控管理 > 数据集管理**，单击右侧**数据查询**。下面演示一个简单的示例：

图 727: 数据查询



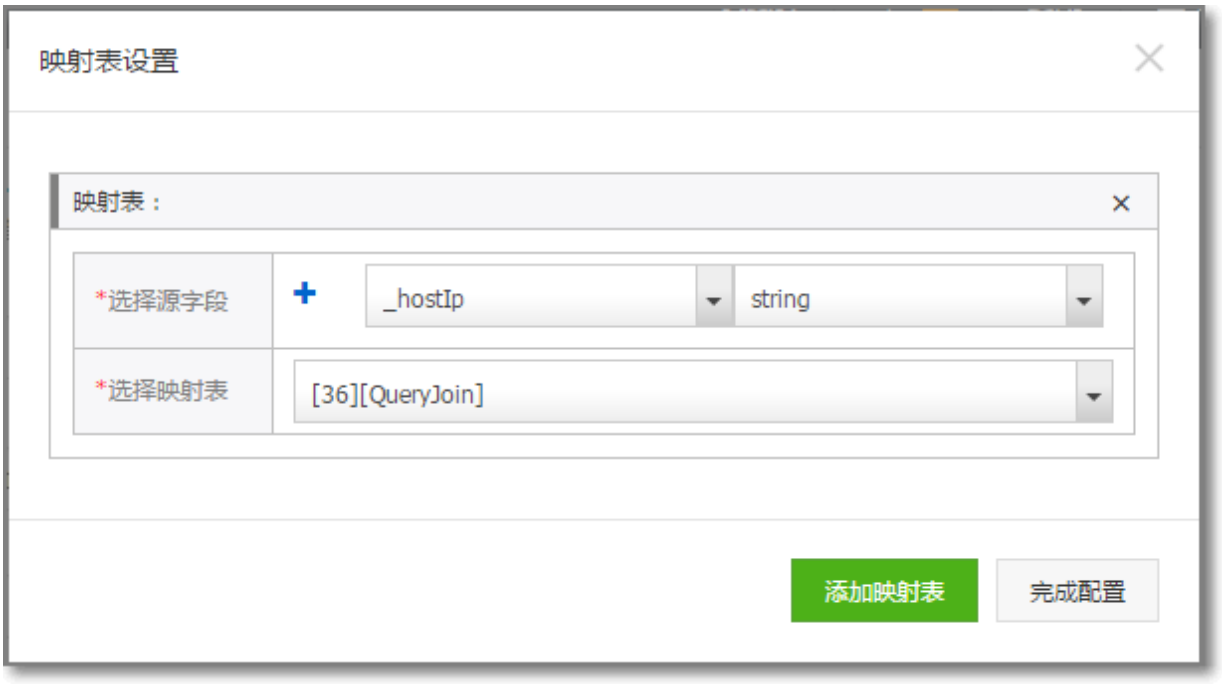
如上数据集存在维度 _hostIp ，选择下钻，其结果如下所示：

图 728: 查询结果

date	MAX_partition_bfree	free	_hostIp	MAX_partition_bused	disk	use	MAX_partition_util
2017-04-14 10:29:00	237782245376	2377822453.76	11	8821133312	3.58	88211333.12	358
2017-04-14 10:30:00	237781696512	2377816965.12	11	8821682176	3.58	88216821.76	358
2017-04-14 10:31:00	237781188608	2377811886.08	11	8822190080	3.58	88221900.8	358
2017-04-14 10:32:00	237780652032	2377806520.32	11	8822726656	3.58	88227266.56	358
2017-04-14 10:33:00	237780099072	2377800990.72	11	8823279616	3.58	88232796.16	358
2017-04-14 10:34:00	237779591168	2377795911.68	11	8823787520	3.58	88237875.2	358
2017-04-14 10:35:00	237779025920	2377790259.2	11	8824352768	3.58	88243527.68	358
2017-04-14 10:36:00	237778460672	2377784606.72	11	8824918016	3.58	88249180.16	358
2017-04-14 10:37:00	237777948672	2377779486.72	11	8825430016	3.58	88254300.16	358
2017-04-14 10:38:00	237777420288	2377774202.88	11	8825958400	3.58	88259584	358
2017-04-14 10:39:00	237776879616	2377768796.16	11	8826499072	3.58	88264990.72	358

现在需要查询各个省份、地址详细的数据咋办？只需要对 _hostIp 进行 Join 查询，我们已经在自己的维表中配置 IP 到地区的详细映射关系了。

图 729: 映射表配置



其查询后的详细数据如下：

图 730: 详细数据

date	MAX_partition_bfree	free	_hostIp	MAX_partition_bused	disk	province	use	MAX_partition_util	city	country
2017-04-14 10:29:00	237782245376	2377822453.76	11.	8821133312	3.58	浙江	88211333.12	358	杭州	中国
2017-04-14 10:30:00	237781696512	2377816965.12	11.	8821682176	3.58	浙江	88216821.76	358	杭州	中国
2017-04-14 10:31:00	237781188608	2377811886.08	11.	8822190080	3.58	浙江	88221900.8	358	杭州	中国
2017-04-14 10:32:00	237780652032	2377806520.32	11.	8822726656	3.58	浙江	88227266.56	358	杭州	中国
2017-04-14 10:33:00	237780099072	2377800990.72	11.	8823279616	3.58	浙江	88232796.16	358	杭州	中国
2017-04-14 10:34:00	237779591168	2377795911.68	11.	8823787520	3.58	浙江	88237875.2	358	杭州	中国
2017-04-14 10:35:00	237779025920	2377790259.2	11.	8824352768	3.58	浙江	88243527.68	358	杭州	中国
2017-04-14 10:36:00	237778460672	2377784606.72	11.	8824918016	3.58	浙江	88249180.16	358	杭州	中国
2017-04-14 10:37:00	237777948672	2377779486.72	11.	8825430016	3.58	浙江	88254300.16	358	杭州	中国
2017-04-14 10:38:00	237777420288	2377774202.88	11.	8825958400	3.58	浙江	88259584	358	杭州	中国
2017-04-14 10:39:00	237776879616	2377768796.16	11.	8826499072	3.58	浙江	88264990.72	358	杭州	中国

24.9.2 日志清洗

ARMS 中数据清洗的作用和功能

- 为什么要数据清洗

  ARMS 向用户提供低学习成本实时监控解决方案。不同用户拥有不同的日志格式，以及不同日志的不同字段信息。ARMS 为适配用户日志，需要先对用户日志进行数据清洗工作。

例如，一行日志,包含日期格式，商品 ID 以及交易额如下所示：

```
2016-08-24-13:32:33|itemID=abc|amount=100
```

通过日志清洗，ARMS 需要知道，在这行日志中，时间为2016-08-24-13:32:33，商品 ID 为abc，以及交易额为100。通过清洗出以上属性以后，ARMS 就可以在后续聚合计算编排中，针对这些属性进行相关场景的计算。

- ARMS 中智能切分和手动切分的区别

  ARMS 抽象数据清洗过程为对日志数据的切分，形成字段（时间、字符串、数字等字段）。

智能切分 ： 快速、便捷、智能

智能生成数据清洗切分方案。对用户提供的部分日志样例进行智能分析，并推荐最优方案供用户选择。智能切分方案可转换为手动切分拖拽式界面进行微调，大大节省手动切分配置时间。

手动切分：手动、全面、可控

通过拖拽式界面配置数据清洗切分逻辑，全图形化配置流程，不需要编写代码即可完成大部分的监控配置任务。

日志格式主要建议

- **尽量保证格式一致**

  同一日志源日志尽量保证格式一致，若存在多种日志在同一日志源，需要使用filter 功能过滤出需要的日志内容。

- **使用可以识别的时间戳**

  ARMS 支持多种常用时间格式：（需要精确到年月日时分秒，毫秒可选）

  手动切分中时间戳格式支持所有能转化为常用 SimpleDateFormat 时间正则表达式（如yyyy-MM-dd HH:mm:ss）的时间格式，时间正则请参见*SimpleDateFormat*官方文档说明。

  智能切分能识别的时间格式：

  - 支持常见能转化为常用 SimpleDateFormat 时间正则表达式（如 2016-8-21 23:12:53 等）的时间格式
  - 支持 Ngix 时间格式，如 28/Nov/2014:11:56:09 +0800
  - 支持其他 Java SimpleDateFormat 常用格式请参见智能切分日期格式。

- **使用清晰无歧义的分隔符** (常用分隔符为 空格、”|”、”;”、”,”等)

推荐使用同种类的分隔符，更容易被切分器切分。

- **字串子串等更推荐使用 JSON 格式**

  日志串中含有 JSON 字串，使用 JSON 切分器处理，更易于数据清洗。

  日志中 JSON 串需为有引号的标准 JSON 格式，且不能含有换行符。

自定义切分的使用请参考文档自定义切分的使用。

24.10 产品使用问题

介绍ARMS在使用过程中遇到的常见问题及解决办法。

24.10.1 自定义清洗常见的问题

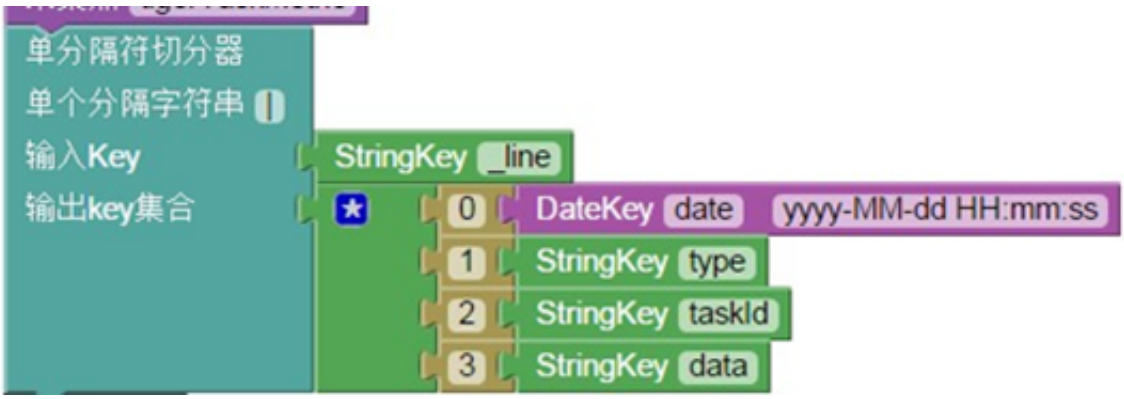
1. 特殊字符（不可见字符）作为分隔符如何切分？

   假如不可见字符是ASCII 码的1，则对应的分隔符为\u0001；如果是2，则是\u0002...以此类推，tab 作为分隔符的话请使用\t作为分隔符。

2. 特殊字符在流程配置页面上如何测试？

   因为无法在浏览器里粘贴不可见字符，因此您只能暂时将测试文本的分隔符替换为可见字符进行测试，全部流程测试通过后，再将分隔符替换为不可见字符进行部署。

3. 下图中 key 的列表中如何添加更多的 key?



   试试点击积木块上的蓝底白色五角星。

4. JSON 格式的日志可以切分吗？

   可以，不过没有直接的专属 JSON 切分器来支持，需要使用逻辑表达式+KV 切分器配合使用。请阅读完下面 KV 切分器的说明，凭借您的智慧，一定可以成功地切分 JSON 格式的日志。

5. 跨行日志（如异常堆栈）支持吗？

暂不支持。

24.10.2 日志清洗中时间字段（date）的常见问题

任务设置中“日志清洗”是非常重要的环节，“日志清洗”将用户日志转化为标准的Key-Value格式，由于日志计算按照时间维度聚合，所以要求用户在进行日志清洗后产生的Key-Value中必须包含时间字段，系统默认采用字段名称为date。所以如果日志中不存在时间字段或者时间字段切分后名称不为date，系统会提示**切分后的字段中不包含时间字段date!**。

下面就上述遇到的问题列举几种常见的案例和解决方法：

日志中不存在时间字段

建议修改日志格式，添加相应的时间字段即可；

如原有日志为：

```
北京|电子产品|211
```

修改为：

```
2016-07-28 12:12:12|北京|电子产品|211
```

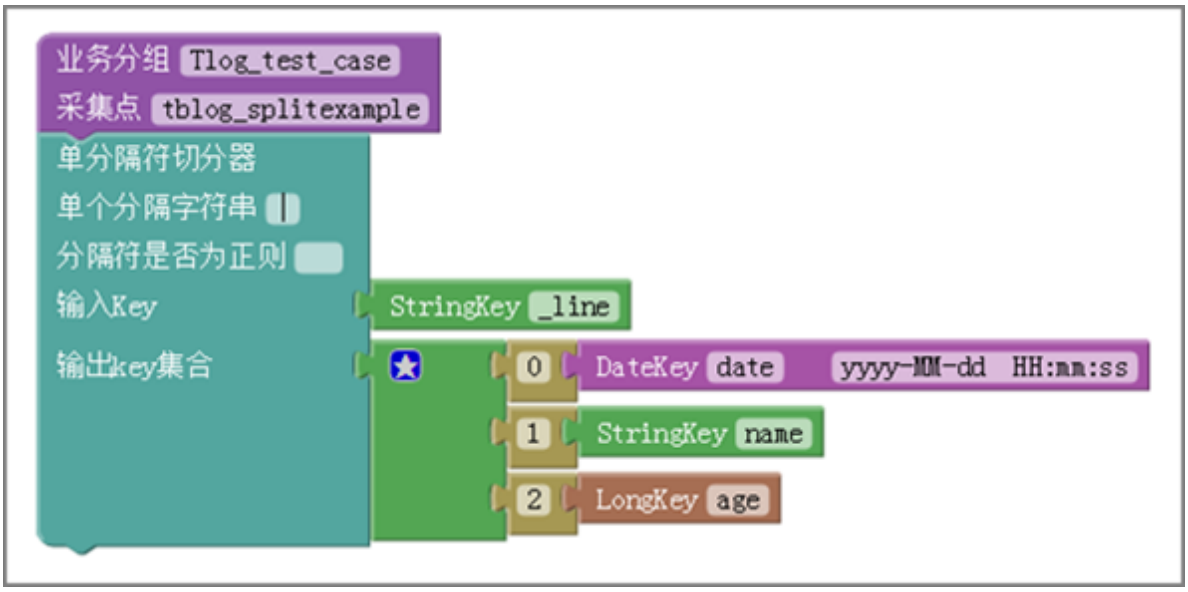
日志中存在时间

1. 案例（1）：日志中时间字段为Date型，该种情况比较简单，样例日志如下：

   ```
   2016-08-09 11:21:17|kevin.yang|26
   ```

   该情况通常会被“智能切分”自动识别出来，其自定义切分形式如下：

   图 731: 切分方案-1



1. 案例（2）：日志中时间字段为long型10位

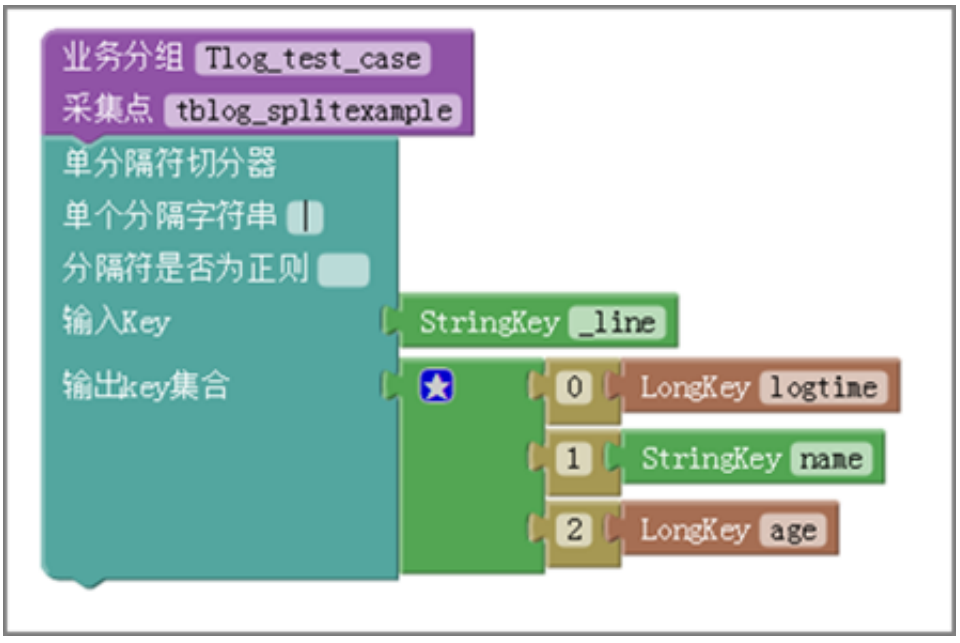
```
1477480004|kevin.yang|26
```

说明:

1. 第一个字段为时间，第二个字段为名称name，第三个字段为年龄age。
2. 最终的时间字段date转化后为13位long类型。

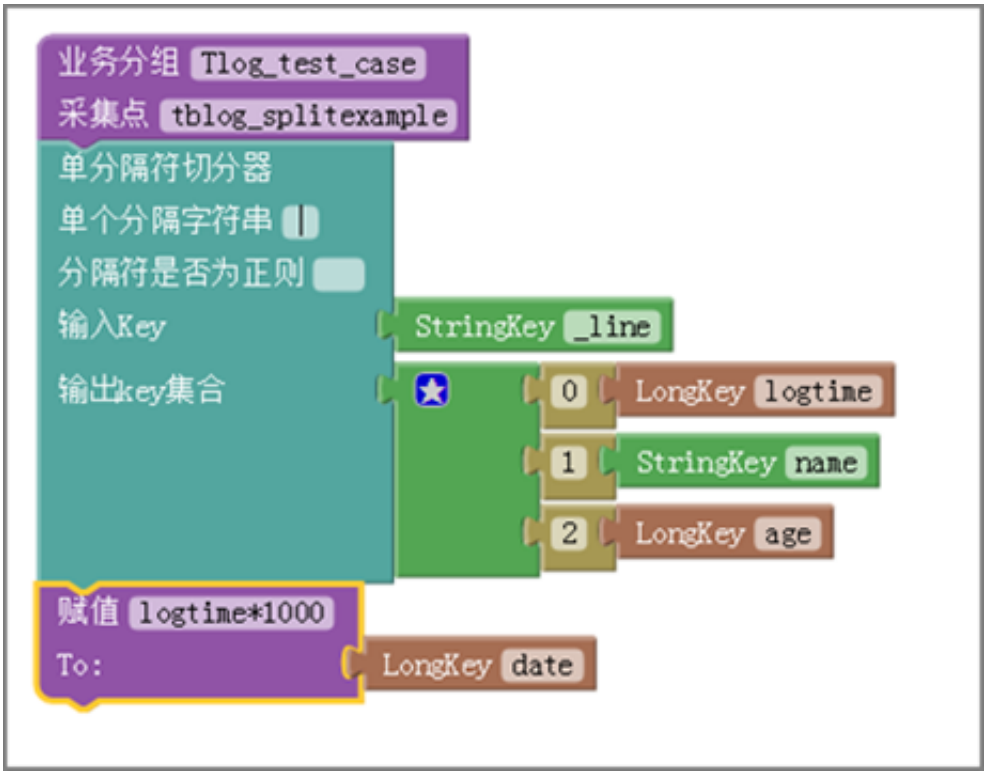
原有自定义切分方案为：

图 732: 切分方案-2



解决方法：需要将logtime变为13位的long。

图 733: 切分方案-3

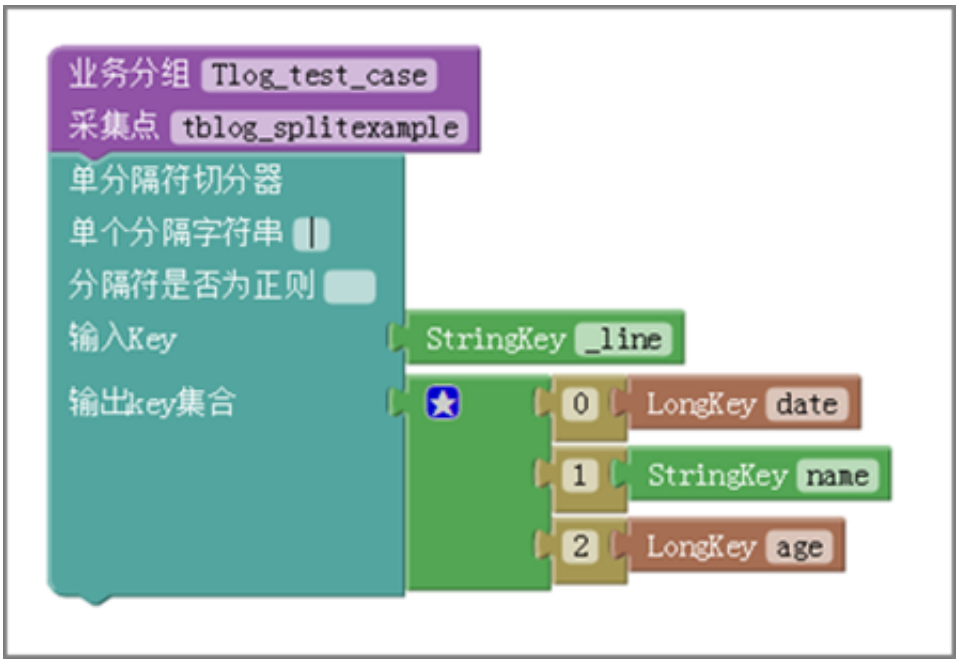


2. 案例（3）：日志中时间字段为long型13位。

```
1477480004000|kevin.yang|26
```

该种情况处理也比较简单，自定义切分时将其命名为date即可（如果使用智能切分，将系统生成的该字段对应的名称重命名为date）：

图 734: 切分方案-4

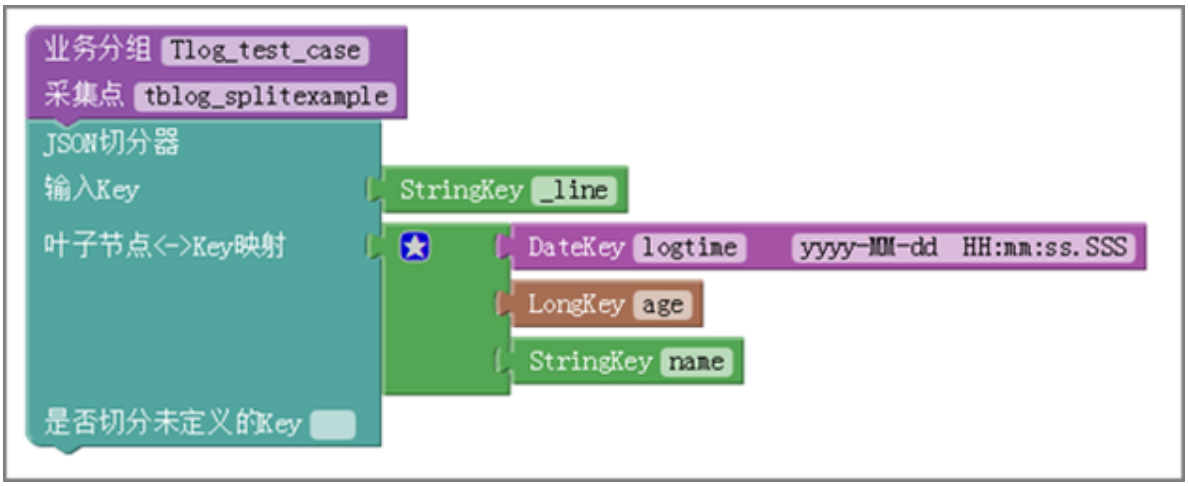


3. 案例（4）：日志中时间字段存在于JSON字符串中（时间字段为Date类型），样例日志如下：

```
"name":"kevin.yang",
```

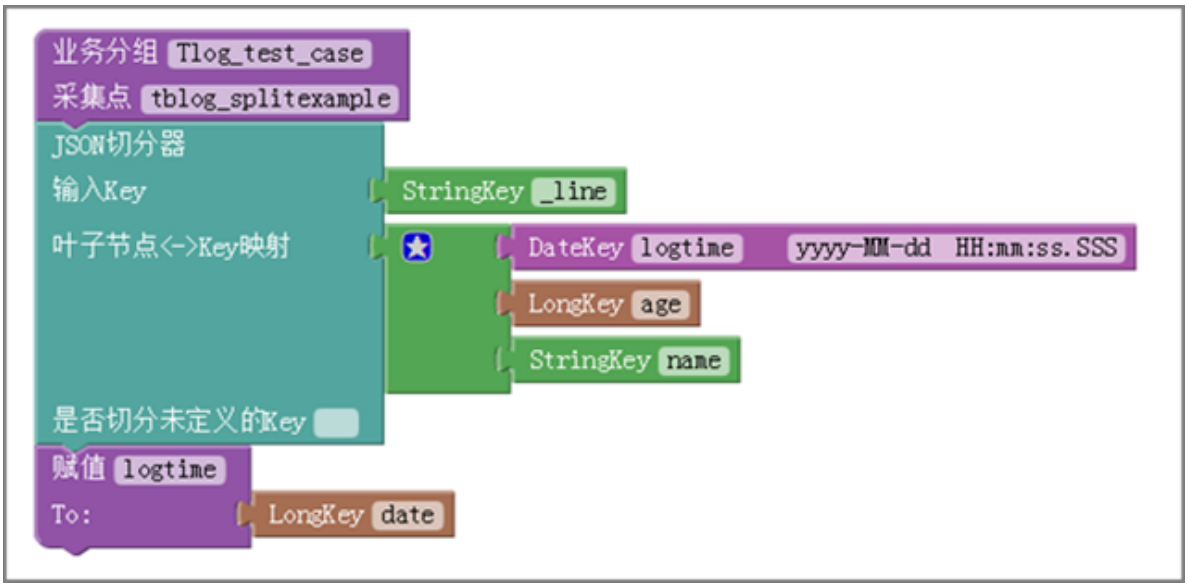
其自定义切分的方案通常为：

图 735: 切分方案-5



解决方法：将logtime赋值给date。

图 736: 切分方案-6

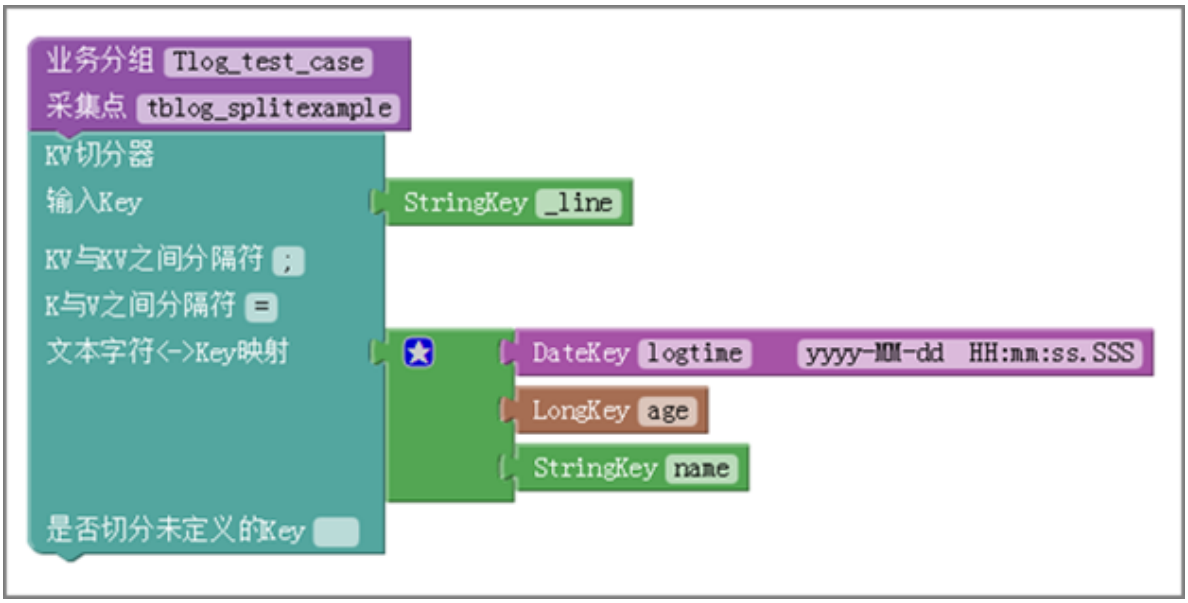


4. 案例（5）：日志中时间字段存在于KV中（时间字段为Date类型），样例日志如下：

```
name=kevin;logtime=2016-10-26 11:28:51.103;age=26
```

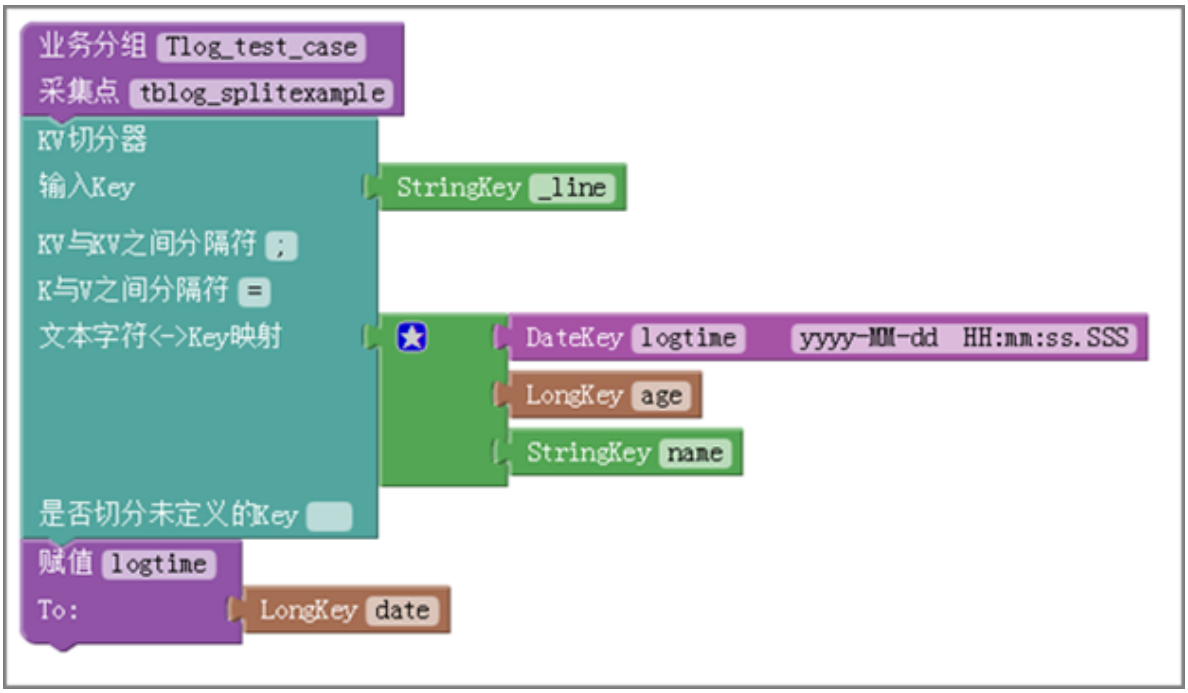
其自定义切分的方案通常为：

图 737: 切分方案-7



解决方法：将logtime赋值给date。

图 738: 切分方案-8



24.10.3 智能切分日期格式

智能切分会自动识别特定格式的日期字段，此文档说明具体定义智能切分适配的日期格式。

时间必须有精确的年月日时分秒字段。可根据以下说明格式可选添加毫秒字段、可选添加时区字段。格式规范说明参考 *SimpleDateFormat*。

1. 常见日期格式

   常见日期格式为日志中常见的时间格式，通常为由规律分隔符隔开的年元日时分秒数据，年月日与时分秒中可用分隔符或者大写字母‘T’隔开。秒字段后可用分隔符‘,’或‘.’后接毫秒字段。最后可加上标准时区字段。

   - **yyyy-MM-ddTHH:mm:ss**

     其中‘T’可替换为空格或其他分隔字符，其中‘:’可替换为其他分割字符，年月日时、时分秒间分隔符可替换为如‘/’‘、’‘-’等常用分隔符，如：

     ```
     1963-04-17T23:51:12
     1963/04/17 12:12:53
     ```

   - **yyyy-MM-dd HH:mm:ss,SSS**

     其中‘,’可替换为‘.’，其他位置可替换字符同上，如：

     ```
     1963/04/17 12:12:53,172
     1963/04/17T12:12:53.172
     ```

   - **yyyy-MM-dd HH:mm:ssZ**

如：

```
1963/04/17 12:12:53+0800
1963/04/17T12:12:53-0700
```

- **yyyy-MM-dd HH:mm:ss,SSS** 如：

```
1963/04/17 12:12:53,999+0800
1963/04/17T12:12:53.878-0700
```

2. 固定标准时间格式

固定时间表示格式，SimpleDateFormat中能翻译成日期正则式的较常用日期格式，

例如年月日中没有分隔符的时间格式，以及月日年形式的常用时间格式等。**

- 类型一：日期时间各自中无分隔符
  - **yyyyMMddHHmmss**

```
19551020233546
```

  - **yyyyMMdd HHmmss**

```
19551020 233546
```

- 类型二：日期格式倒序日在前月在中年在后

dd-MM-yyyy HH:mm:ss

```
17-04-1963 23:59:23
```

3. SimpleDateFormat中部分常见日期格式

SimpleDateFormat文档中出现的部分常见日期格式，详细信息请参见*SimpleDateFormat*官方文档说明。

时间字段必须精确到年月日时分秒的日期。

- **yyyy.MM.dd G ‘at’ HH:mm:ss z**

```
2001.07.04 AD at 12:08:56 PDT
```

- **EEE, MMM d, ‘’ yy**

```
Wed, Jul 4, '01
```

- **yyyyy.MMMMM.dd GGG hh:mm aaa**

```
02001.July.04 AD 12:08 PM
```

- **EEE, d MMM yyyy HH:mm:ss Z**

```
Wed, 4 Jul 2001 12:08:56 -0700
```

- **yyMMddHHmmssZ**

  010704120856-0700

- **yyyy-MM-dd’ T’ HH:mm:ss.SSSZ**

  2001-07-04T12:08:56.235-0700

- **yyyy-MM-dd’ T’ HH:mm:ss.SSSXXX**

  2001-07-04T12:08:56.235-07:00

4. 其他特定日期格式

   如Ngix日期格式

   25/Aug/2012:16:41:07 +0800

24.10.4 日志清洗中系统字段介绍

在**日志清洗**环节除了用户定义的字段，通常系统会添加一些默认的字段，主要包括_line字段与_hostIp字段。

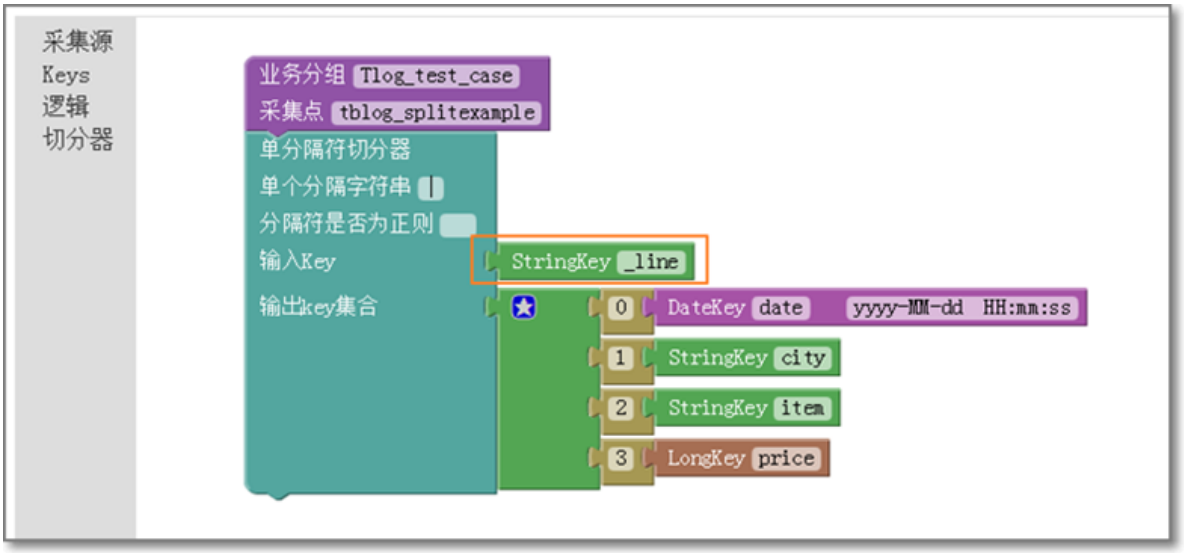
- **_line**

  _line字段表示每一行日志

  如日志格式如下：

  ```
  2016-07-28 12:12:12|北京|电子产品|211
  2016-07-28 12:13:12|北京|电子产品|211
  2016-07-28 12:14:12|北京|电子产品|211
  ```

  进行第一次切分时，其输入key为每一行日志”_line” ，其自定义切分形式如下：



- **_hostIp**

  _hostIp字段表示每一行日志的来源IP，目前ARMS支持三种数据源StarAgent、鹊桥和MQ数据源，其支持关系如下表所示。

数据源	是否支持
StarAgent数据源	支持
鹊桥数据源	支持
MQ数据源	不支持

  目前智能切分不提供_hostIp，只有在自定义切分模式下才提供_hostIp，以上面的日志为例来进行说明，点击**日志切分预览**：



  上图中_hostIp字段为127.0.0.1（点击**日志切分预览**后无论什么数据源_hostIp字段均为127.0.0.1），是因为local模式，任务真正运行时会产生真实数据。

24.10.5 自定义切分的使用

除了智能切分器，ARMS 还提供了自定义清洗配置器，用于自定义日志清洗流程来处理复杂的切分需求。

自定义清洗配置器采用所见即所得的可视化配置方式，本文档将详细介绍如何使用自定义清洗配置器。

配置界面简介

自定义清洗配置界面包含三个部分：

- 原始日志区域
- 可视化清洗流程编辑区域
- 清洗结果预览区域

图 739: 自定义清洗配置



在可视化流程编辑区域（以下简称“编辑区”），可以使用一个或多个“积木块”组装出日志清洗的逻辑，将原始日志区域中的日志“清洗”为结构化的键值对（Key-Value Pair)。

在编辑区编辑的过程中，可以随时点击清洗结果预览区域上方的**日志切分预览**按钮，预览每一行原始日志通过当前编辑区的流程清洗后的键值对结果。

在通过预览结果确认清洗流程准确无误后，单击页面下方的**下一步**按钮启动任务。ARMS 将使用在编辑区保存的流程来处理每一条从数据源中消费的日志。

示例一

本示例演示如何配置一个简单的日志清洗流程。

假如某系统的样例交易日志如下：

```
2016-11-08 11:00:01|user_abc|123456|下单
2016-11-08 11:00:02|user_abc|123456|支付
2016-11-08 11:10:01|user_abc|123456|退货
```

以第一行数据为例，假设目标切分结果为如下几个键值对：

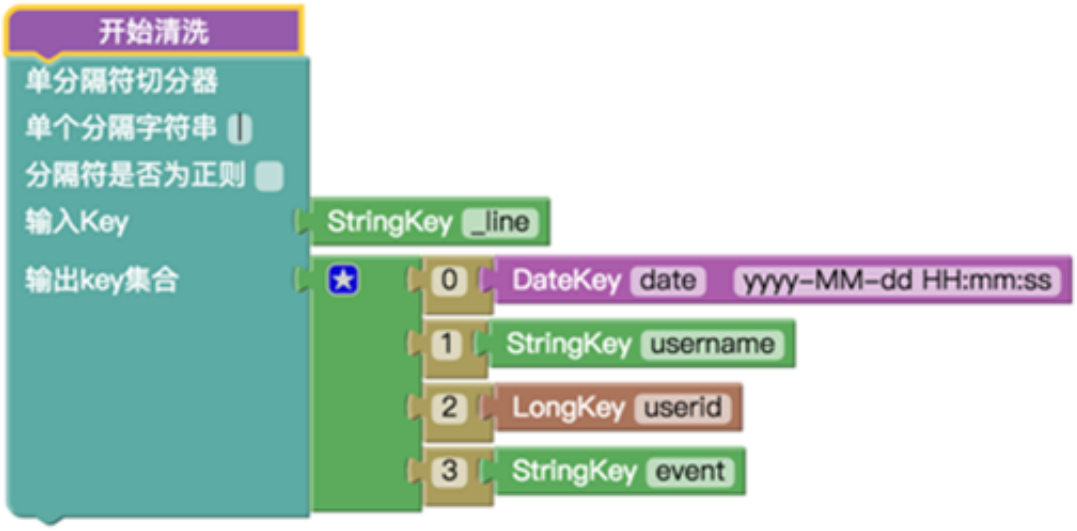
Key 类型	Key	Value
Date	date	2016-11-08 11:00:01
String	username	user_abc
Long	userid	123456
String	event	下单

通过观察发现，键值对之间均使用竖线（|）进行分隔，因此可以使用**单分隔符切分器**来切分键值对。

1. 在编辑区左侧的工具栏中，将**单分隔符切分器**拖拽至编辑区，如下图所示。

   注意: 切分器必须连接在**开始清洗**模块之后才能够生效。

   图 740: 日志切分-1

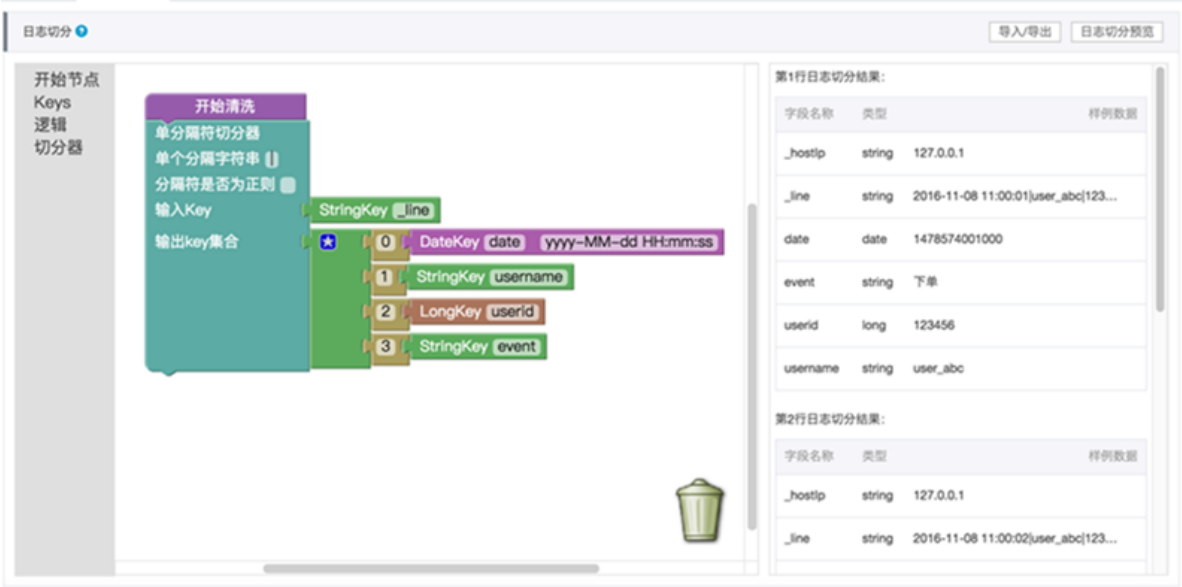


   配置切分说明：

   - 以“|”为分隔符，“_line”作为入参（_line代表原始日志行本身）。
   - 切分后0号位置的字符串按照 yyyy-MM-dd HH:mm:ss 的格式转为 date（内部实际将以 long 型保存）。
   - 1号位置的字符串转为保存为 key 为 username 的 string。
   - 2号位置转为 key 为 userid 的 long。
   - 3号位置保存为 key 为 event 的 string。

2. 点击**日志切分预览**，可以观察到每行日志的切分结果如下：

   图 741: 切分预览



最终生成的切分模型如下所示：

图 742: 切分模型-1



3. 单击**保存**或**下一步**，清洗流程生效。

到此为止，一个简单日志切分就完成了。

示例二

本示例演示如何对复杂的日志进行切分清洗。

```
2016-11-08 11:00:01|下单|user_abc|123456
2016-11-08 11:00:02|支付|200
2016-11-08 11:10:01|退货
```

在此样例中，不同的日志由于操作类型不一样，字段的个数、同位置各字段的含义也不同（这在业务系统中非常常见）。这时可以通过 if-then/if-else 逻辑对不同类型的日志进行区分。

图 743: 日志切分-2



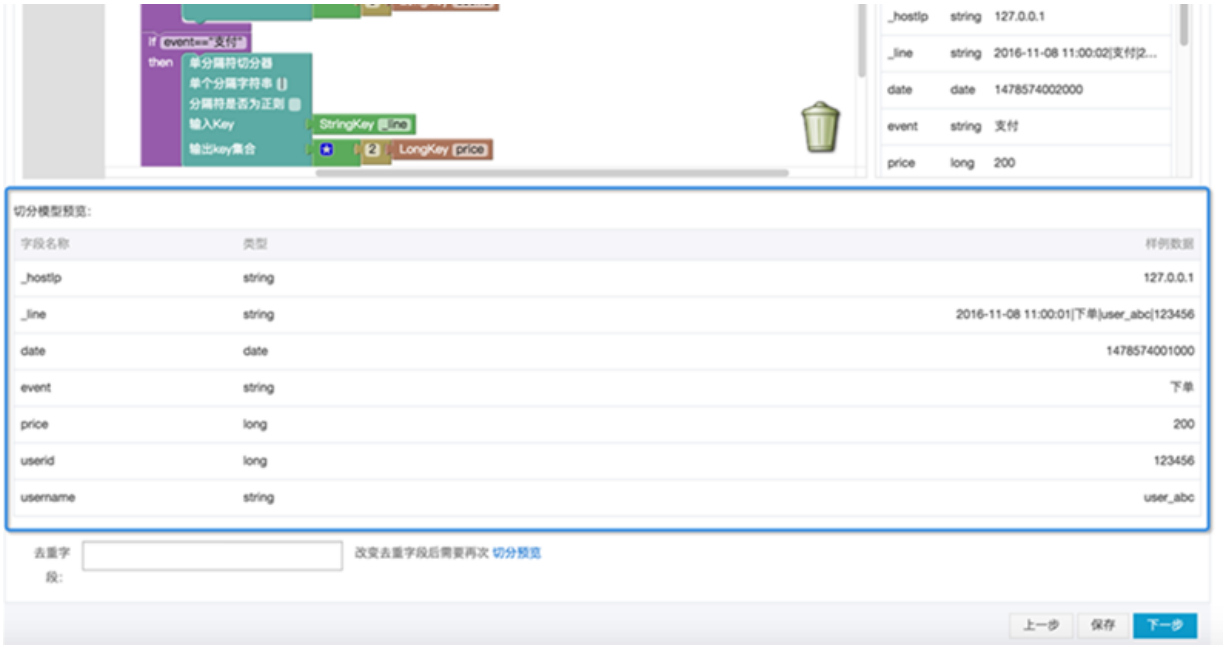
配置切分说明：

- 首先使用单分隔符切分器切分出 date 与 event 字段；
- 接着使用逻辑模块中的 if-then 模块，在 event 为不同值时分别使用不同的单分隔符切分器配置切分出不同的字段。

  从右侧的预览结果中也可以看到，第一行与第二行切分出的键值对组合是不一样的。第一行中有 username 与 userid，而第二行中有 price。

  注意： 在自定义切分最下方的**最终切分模型预览**，包含的将是上图中右侧单行切分预览区域中**所有字段的并集**。

图 744: 字段并集



了解更多切分方式

以上两个例子仅使用了一种切分器（单分隔符切分器）。ARMS 内置了单分隔符、多分隔符、顺序、K-V、JSON 等多种切分器，针对不同的场景可以单独或组合使用。每个切分器的详细说明如下：

1. 单分隔符切分器

   说明

   将单一字符串作为输入，当匹配用户指定分隔字符串时进行切分，切分结果为多个 Key-Values。

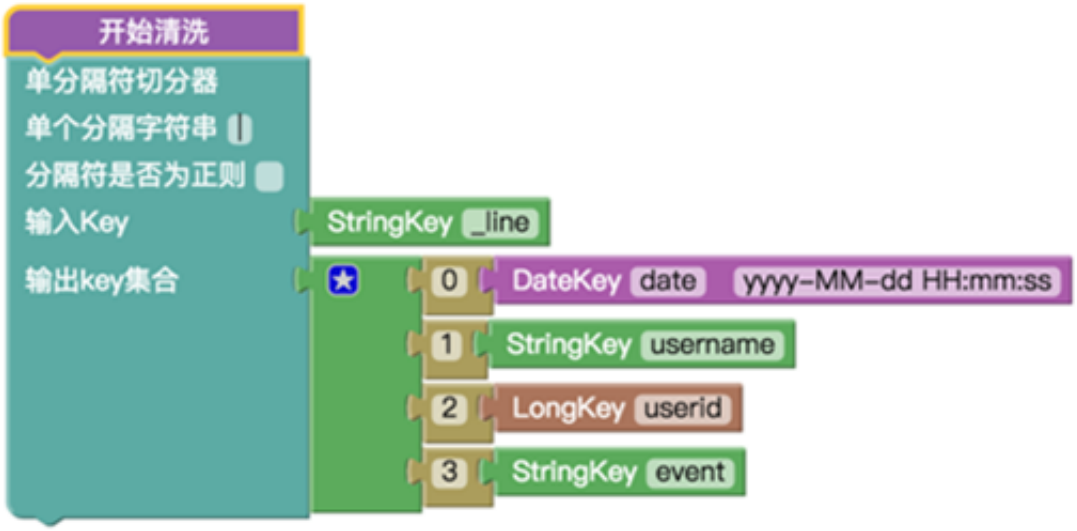
   实例

   样例日志如下所示：

   ```
   2016-11-08 11:00:01|user_abc|123456|下单
   2016-11-08 11:00:02|user_abc|123456|支付
   2016-11-08 11:10:01|user_abc|123456|退货
   ```

   其自定义切分如下所示：

   图 745: 日志切分-3

   配置说明：

   - 以|为分隔符。
   - 切分后0号位置的字符串按照 yyyy-MM-dd HH:mm:ss的格式转为 date（内部实际将以 long 型保存）。

- 1号位置的字符串转为保存为 key 为 username 的 string。
- 2号位置转为 key 为 userid 的 long。
- 3号位置保存为 key 为 event 的 string。

说明:

- 对不可见字符的支持：本切分器支持不可见字符的切分，例如 \t，\u001f(ascii char=31）。一些不可见字符的切分将无法在页面测试中被正确验证（因为页面文本框内无法正确输入不可见字符），所以建议在测试页面先使用可见字符进行调试，然后换成不可见字符保存后进行部署。
- 多字符分隔符：本切分器分隔符可以为多字符，如 a::b::c 为输入，以::为分隔符切分。
- 切分模型包含 _line 字段与_hostIp 字段，具体参阅日志清洗中系统字段介绍。

2. 多分隔符切分器

说明

将单一字符串作为输入，当匹配任意一个用户指定分隔字符时进行切分，切分结果为多个 Key-Values。

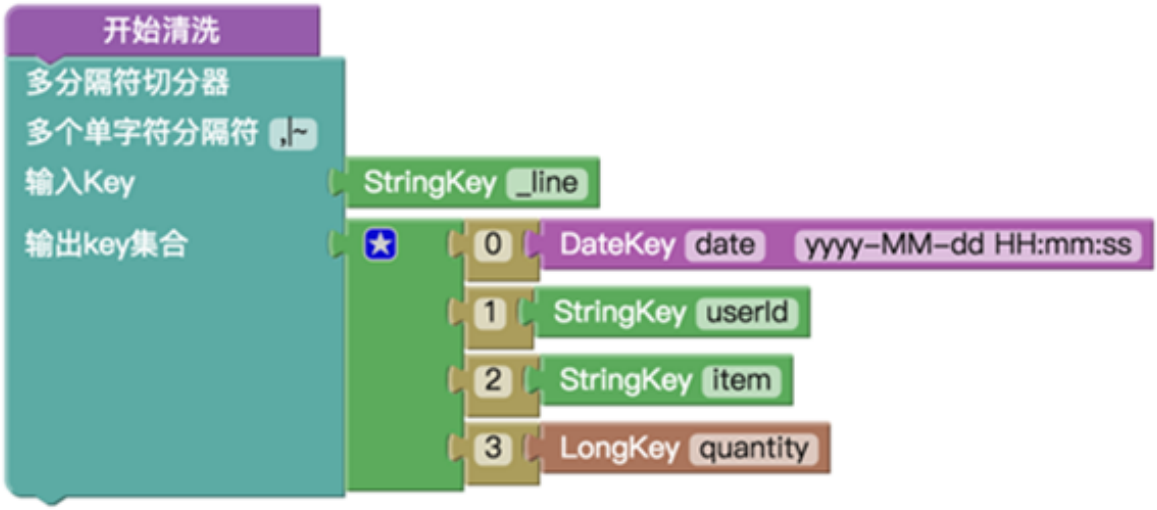
实例

样例日志：

```
2014-07-25 17:25:00,aaa|b~1
2014-07-25 17:26:00,aaa|b~1
```

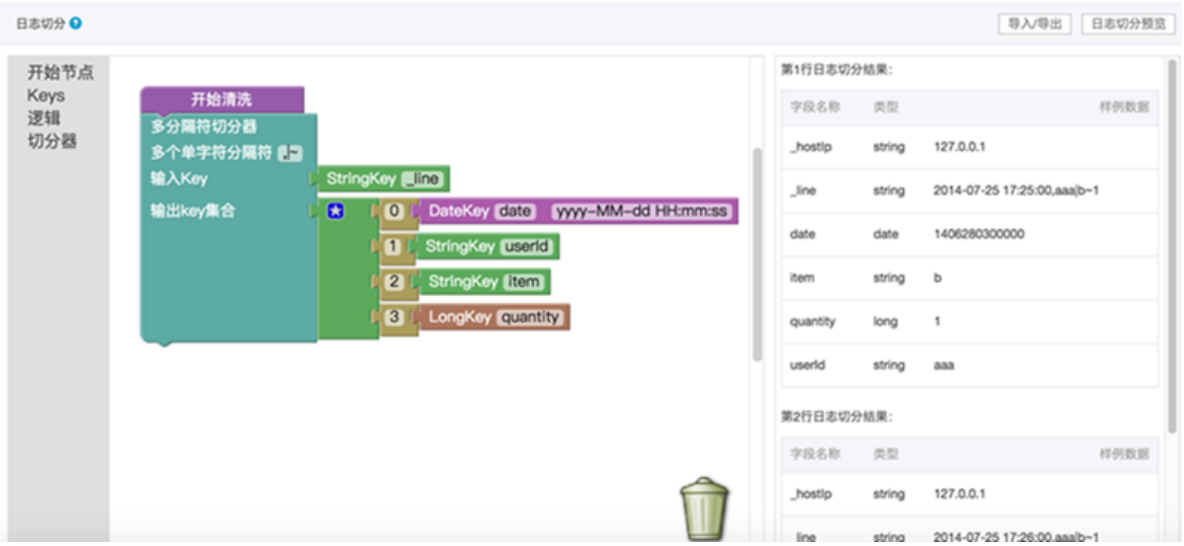
自定义切分如下所示：

图 746: 日志切分-4



每行日志的切分如下：

图 747: 日志切分-5



切分模型如下所示：

图 748: 切分模型-2



字段名称	类型	样例数据
_hostIp	string	127.0.0.1
item	string	b
userId	string	aaa
_line	string	2014-07-25 17:25:00,aaa|b=1
date	date	1406280300000
quantity	long	1

3. 顺序分隔符切分器

对于较复杂的切分场景（如 accesslog），ARMS 提供**顺序分隔符切分器**，以 O(n) 的复杂度将日志进行切分。

实例

样例日志：

```
117.xx.xx.xx 835158 - [26 May 2014:14:05:28 +0800] "GET http://trade.taobao.com/trade/detail/trade_snap.htm?spm=a1z0f.2.100003.9.8BpicQ&trade_id=664793864233811" 302 0
117.xx.xx.xx 835158 - [26 May 2014:14:05:28 +0800] "GET http://trade.taobao.com/trade/detail/trade_snap.htm?spm=a1z0f.2.100003.9.8BpicQ&trade_id=664793864233811" 302 0
```

我们希望切出 IP 字段、平均响应时间、时间、请求的 URL:

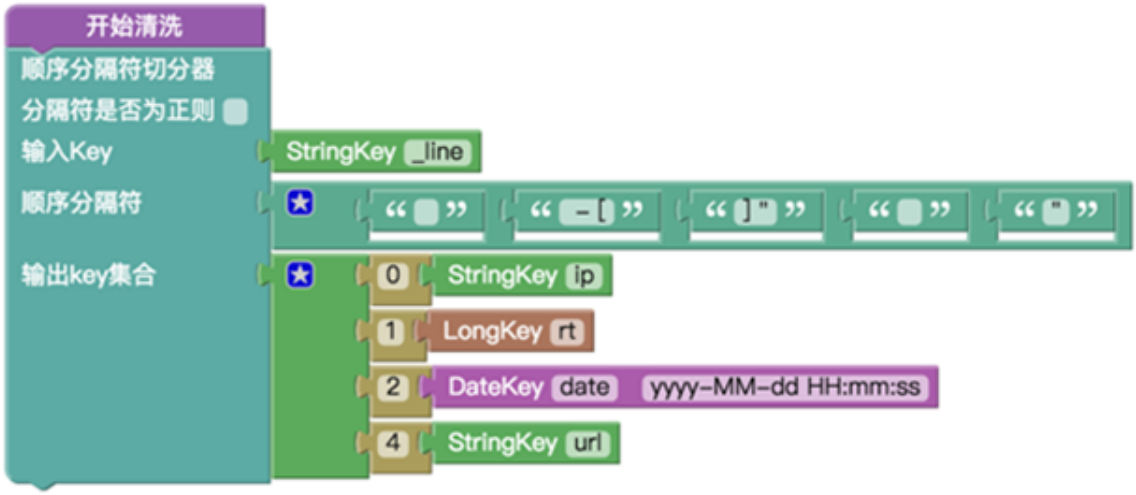
=> [url - *http://trade.taobao.com/trade/detail/trade_snap.htm?spm=a1z0f.2.100003.9.8BpicQ&trade_id=664793864233811*] [_hostIp - 127.0.0.1] [ip - 117.xx.xx.xx] [date - 1401084328000] [rt - 835158]

我们会发现：

- 用单分隔符切分器显然无法满足需求；
- 用多分隔符切分器也不能满足需求：我们如果想切出 rt=835158 这组 KV，必须以空格作为切分字符之一，但是这样做会错误地把26 May 2014:14:05:28 +0800这个字符串给切成几段，因此我们将无法得到 date - 1401084328000；
- 如果使用正则表达式来匹配 KV，效率太低，需要遍历多次。

ARMS 提供了”顺序分隔符切分”的方式，使用如下配置，就可以得到本示例正确的结果。具体配置方式如下图：

图 749: 顺序分隔符切分



先定义了5个分隔符字符串，并依次将输入字符串切分为了6段子串，随后将指定的子串分别赋给不同的 key，即可完成目标切分效果。

4. KV 切分器

如果需要切分如下多组 key=value 形式的日志：

```
key1=aaaa;key2=bbbb;key3=cccc;key4=dddd;....
key1=aaaa;key2=bbbb;key3=cccc;key4=dddd;....
```

且各个 key-value 的位置不固定时，可以使用 KV 切分器：

切分配置如下：

图 750: 日志切分-6



注意:

- 在没有配置文本字符<->Key 映射时，KV 切分器将视所有的 key 为“未定义key”，所有 key 对应的 value 将被切为 StringKey 类型；
- 如果选择取消勾选**是否切分未定义的 Key**，那么未定义 Key 将不会被切分。也就是说如果没有配置文本字符 <->Key 映射，并且**未勾选**是否切分未定义的 Key，本切分器将不做任何事。

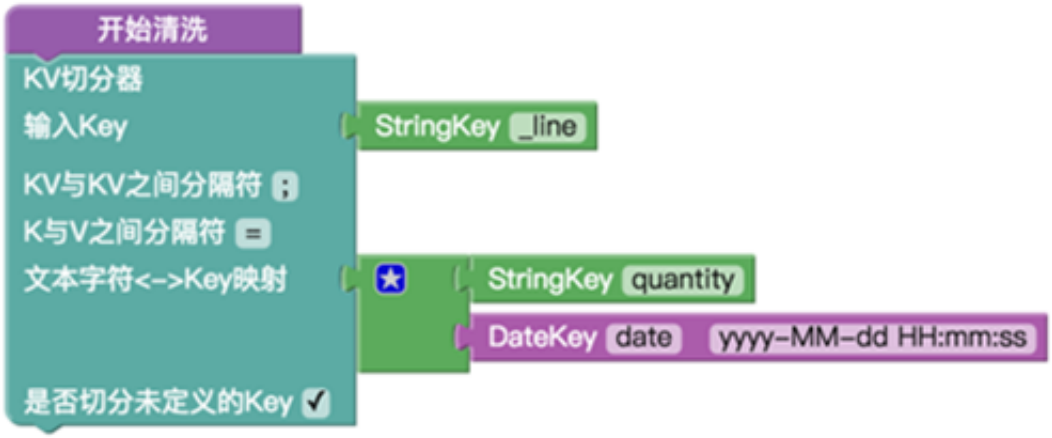
这里展示一个需要将制定的 Key 转换成对应类型的例子：

假设有日志如下：

```
name=abc;item=iphone6;quantity=15;date=2014-07-25 17:25:00;....
name=abc;item=iphone6;quantity=15;date=2014-07-25 17:25:00;....
```

需要将 quantity 变为 LongKey，date 变为 DateKey：

图 751: 日志切分-7



5. JSON 切分器

如果日志中包含 JSON 字符串，使用 JSON 切分器可以快速切分出该 JSON 中的“叶子节点”。

例一：**全自动切分**

```
{
"title": "Example Schema",
"type": "object",
"quantity": 200,
"date":"2015-12-12 12:12:12",
"properties": {
        "firstName": {
          "type": "string"
        },
        "lastName": {
          "type": "string"
        },
```

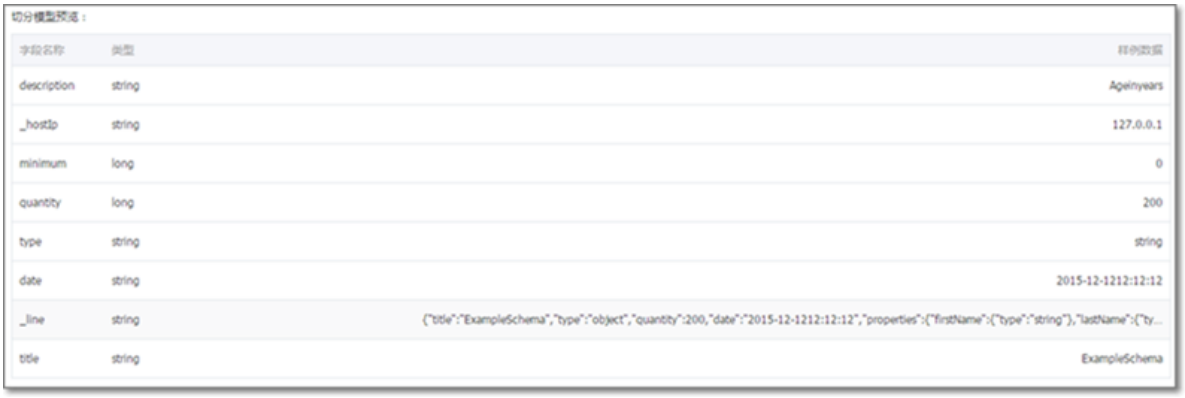
```
        "age": {
            "description": "Age in years",
            "type": "integer",
            "minimum": 0
        }
    },
"required": ["firstName", "lastName"]
}
```

将该 JSON 放在一行文本中，使用 JSON 切分器进行切分：

图 752: 日志切分-8



切分结果如下：



字段名称	类型	样例数据
description	string	Ageinyears
_hostIp	string	127.0.0.1
minimum	long	0
quantity	long	200
type	string	string
date	string	2015-12-1212:12:12
_line	string	{"title":"ExampleSchema","type":"object","quantity":200,"date":"2015-12-1212:12:12","properties":{"firstName":{"type":"string"},"lastName":{"ty...
title	string	ExampleSchema

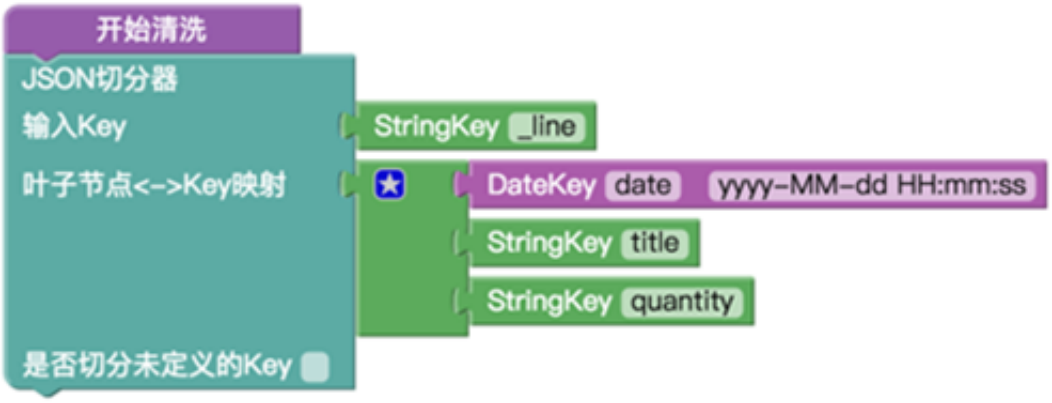
由于勾选了**切分未定义字段**，JSON 切分器将自动切出所有的“非 array 类型的叶子节点”，并自动转成对应的类型（stringkey 或 longkey）。

> **说明:** 本例中的 required 字段由于是 array 类型所以没有切分。另外，date 这个字段默认转化为 string 类型。

例二：**自定义切分**

如果只想切分 JSON 中指定的某些字段，可以这样配置：

图 753: 日志切分-9



其切分模型如下：

图 754: 切分模型-3



字段名称	类型	样例数据
_hostIp	string	127.0.0.1
_line	string	{"title":"Example5chema","type":"object","quantity":200,"date":"2015-12-12 12:12:12","properties":{"firstName":{"type":"string"},"lastName":{"t...
date	date	1449893532000
quantity	string	200
title	string	Example5chema

本例中取消了对**切分未定义字段**选项的勾选，自定义了 title、date 和 quantity 三个字段，并指定了类型。ARMS 会只切分自定义的字段，并转成对应的类型（本例中 date 字段被转成了 ARMS 内置的 long 型）。

6. 自定义切分其他问题

   1）自定义清洗中的系统字段请参阅日志清洗中系统字段介绍。

   2）自定义切分遇到的问题请参阅自定义清洗常见的问题。

24.10.6 常见问题

1. ARMS报表数据在服务器保存多长时间？

   数据默认保存时间为半年。

2. ARMS支持监控除了日志以外的数据来源吗？

   目前ARMS不仅提供日志作为数据来源，同时也支持MQ作为数据源。后期我们将推出数据库据的BinLog作为来 数据来源

3. ARMS业务监控报表数据的延迟有多久？

   在日志通道不延迟的情况下，统计计算的延迟在10秒之内。也就是说，假如您在ARMS上定制了一个统计日志某字段出现次数的报表，我们将在北京时间12点10分10秒前统计出12点10分0秒前的数据。

4. 除了在ARMS控制台内查看报表以外，我可以通过API的方式获取到某张报表所对应的数据吗？

   可以。

5. ARMS会修改或移动我本机上的日志文件吗？

   不会，ARMS使用的StarAgent客户端，只会按照您的配置对您机器上的文件进行读取操作，不会对其进行任何形式的修改操作。

6. ARMS的实时计算引擎JStorm能开发给用户直接使用么？

   我们目前不支持直接开发JStorm给客户直接使用。主要原因也是因为ARMS基于JStorm做了很多性能和QoS的优化。但是不排除未来我们经过优化以后暴露给您JStorm接口直接运行。

7. ARMS的列示存储能直接暴露给用户直接使用么？

   不能。目前我们仅支持Open API的方式取数据。主要原因也是因为我们的存储方式是经过特定优化。

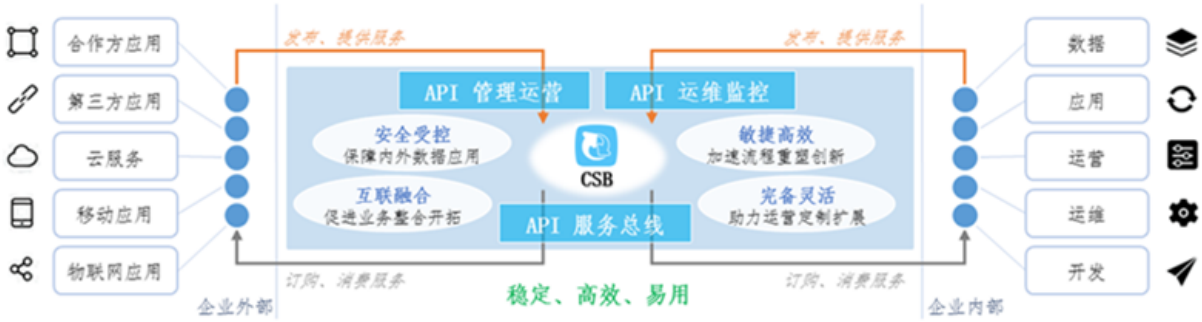
25 云服务总线CSB

25.1 产品概述

云服务总线（Cloud Service Bus，简称CSB）是一个基于高可用分布式集群技术构建的服务API开放平台。

随着业务模式的创新，越来越多的企业组织需要以API方式把自己的核心业务资产贯通整理并开放给合作伙伴，或者与第三方的应用整合，以发掘业务模式、提高服务水平、拓展合作空间。CSB作为一款PaaS级阿里云产品，正是基于此需求，将企业各个系统提供的服务以发布、订阅API的形式进行开放，并对这些互通服务进行一致的组织和管控，从而打通内外新旧系统，实现跨技术平台、应用系统、企业组织的服务能力互通。通过CSB，您可以轻松构建企业内部以及与上下游、第三方企业之间灵活、开放、安全、稳定、高效的合作平台，实现业务能力的进一步融合、重塑与创新。

图 755: CSB产品示意图

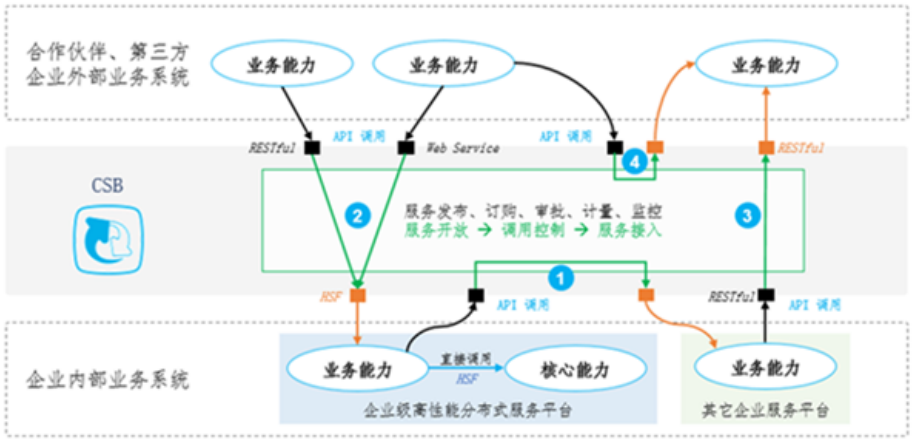


CSB基本功能如下：

- 企业内部不同架构平台的服务之间通过CSB实现互通。
- 企业内部服务开放给外部，允许通过CSB以不同的协议同时访问。
- 企业内部通过CSB访问外部以不同协议提供的服务。
- 外部不同架构平台的服务之间通过CSB实现互通。

具体如下图所示：

图 756: CSB功能示意图



25.2 使用前须知

本文介绍使用CSB前，需要了解的信息。

角色与主要操作

CSB的用户角色及其对应的主要操作有：

- 服务发布者：服务组管理、发布服务、发布管理
- 服务订阅者：消费凭证管理、订阅服务、订阅管理
- 系统管理员：实例管理、用户管理

在CSB中，用户是对等的，没有从属概念，只有授权关系。

用户与实例

用户与实例的关系如下：

- 每个用户都可以拥有属于自己的一个或多个CSB实例，拥有这些CSB实例的管理员权限。
- 用户可以定义这些CSB实例之间的服务路由规则，并控制其他用户对这些CSB实例的访问权限，包括发布服务、订阅服务，甚至实例管理（代理）的权限。
- 每个用户和他所拥有的所有CSB实例，即构成该用户的CSB租户域。
- 用户在取得某个CSB实例的访问权限后，即可以在该实例上发布或者订阅服务。服务的发布者就是该服务的拥有者，可以审批授权其他用户的订阅申请，未来还将支持授权其他用户管理该用户的某个服务组。CSB用户在这里同样是对等的，CSB只关心两个用户之间针对某个服务或服务组的授权关系，并不关心用户的实际组织层级和隶属关系。

群组

CSB里的群组概念如下：

- CSB有群组的概念，对应于相对隔离的管理环境。例如企业的内部数据中心和阿里云公共云的某个地域（region）即是不同的群组。
- 每个群组有CSB群组管理员的角色。与CSB实例管理员不同，只有群组管理员可以应用户请求创建CSB实例。例如在阿里云公共云环境中，CSB产品运维团队即是该群组的管理员。

级联式服务发布

CSB的级联式服务发布是指：

- 级联发布是指跨CSB实例的服务发布，也就是在一个CSB实例上接入已有服务，而在另外一个CSB实例上开放出来，供订阅者消费。级联链路可以跨3个或更多节点。
- 级联发布的链路并不是随意的，需要群组管理员在出发节点即服务的接入端实例进行定义。链路规则就是指明接连链路中各个CSB实例的先后链接关系，例如A接入的服务要在C上发布，必须依次经过实例B和C。
- 级联服务经过的CSB实例可能归属不同的用户，即隶属不同的租户域，因此需要群组管理员首先进行实例间授信。在阿里云公共云上，级联发布链路的定义和管理，包括实例间授信操作，都由群组管理员即CSB产品运维团队完成。

25.3 基本概念

本文介绍CSB常见的名词及相关定义。

表 94: 基本概念

名词	解释
API消费方应用	指直接调用CSB上开放的服务API的消费者应用。
API消费者	代表API消费方应用在CSB上订购服务和管理订购的用户。
API消费凭证	API消费方应用需要使用API消费凭证（简称凭证）来调用CSB上开放的服务API。API消费者使用凭证来订购API。凭证具体表现为一对AccessKey和SecretKey，通常简称AK/SK，在API调用时用来做签名信息计算，CSB接收到API调用请求时对签名信息做验证以确定调用者的身份。 API消费者可以创建多个API消费凭证，每个凭证可被一个或一组API消费方应用使用，通常可以把凭证作为API消费方应用的分组。
API提供方应用	和消费方相对，指接入CSB开放出服务API的提供者应用。

名词	解释
API发布者	即API提供方应用在CSB上发布服务和管理服务的代表。体现为开放平台上具有发布者角色的账号。
CSB实例	每一组CSB服务总线节点（Broker）构成的集群被视为一个独立的CSB实例，通常负责一个业务域内能力的对外开放，也可以把外部的能力发布开放给业务域的内部。
CSB群组	一个CSB群组就是多个CSB实例的集合，这些CSB实例通过同一个CSB管理中心管理，使用同一套用户账号系统。 如何界定群组的边界取决于实际业务需求以及网络拓扑限制，而其中CSB管控中心与CSB实例的部署模式也应根据具体网络拓扑限制确定。
服务接入	对CSB来说，或者说将服务接入到CSB上，指的是在CSB上注册这个服务并且提供足够的信息让CSB可以访问这个已有的服务。例如要接入一个RESTful服务，需要提供这个服务的基础URL或称端点，要使用的HTTP方法等信息。
服务开放	对CSB来说，服务在CSB上开放就是把一个已接入的服务在某个CSB上提供对应的API调用入口，可以用各种协议（包括原协议）来访问这个服务API。注意API的发布都在特定CSB实例上。
服务授权	API的发布者赋予API消费方应用使用指定API消费凭证来调用某个API的权限，就是授权。授权除了定性的允许、禁止之外，还可以包含更广泛的限制如访问频度控制、优先级等等。
服务组	服务组（API组）是业务上的原子粒度分组，每一个API都归属且仅归属一个API组。也就是说，在发布新的API时，必须指定其所属的服务组。 API组在API数量少时可视作简单分类，在数量多时通常需要更多层次结构的分类目录树，这个特性将在后续版本推出。

25.4 应用场景

CSB可以应用于专有云、公共云、混合云等多种场景。CSB的主要目标是帮助企业实现各个系统间的服务访问以及能力的对外开放，并对其进行安全授权、流量限制等管理和控制，实现跨系统跨协议的服务互通。

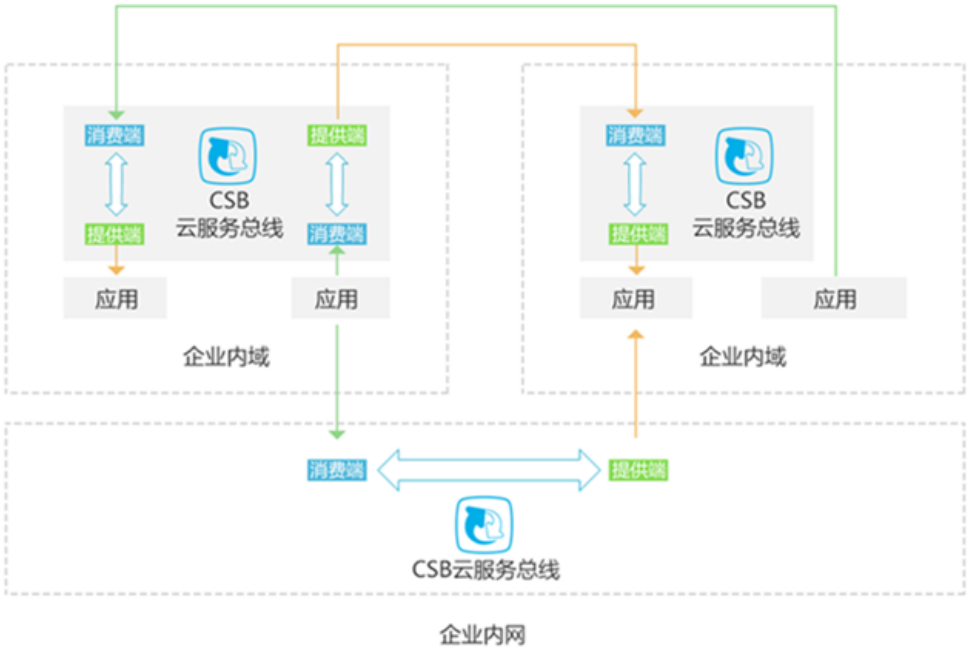
内部互通

企业内部服务能力通过CSB可管可控地开放互通。

- 一个域的应用通过另一个域的CSB实例访问其内部应用服务；
- 两个域的CSB实例构成的桥接通道实现互通；

- 各个域之间通过企业统一的CSB实例实现互通。

图 757: 内部互通

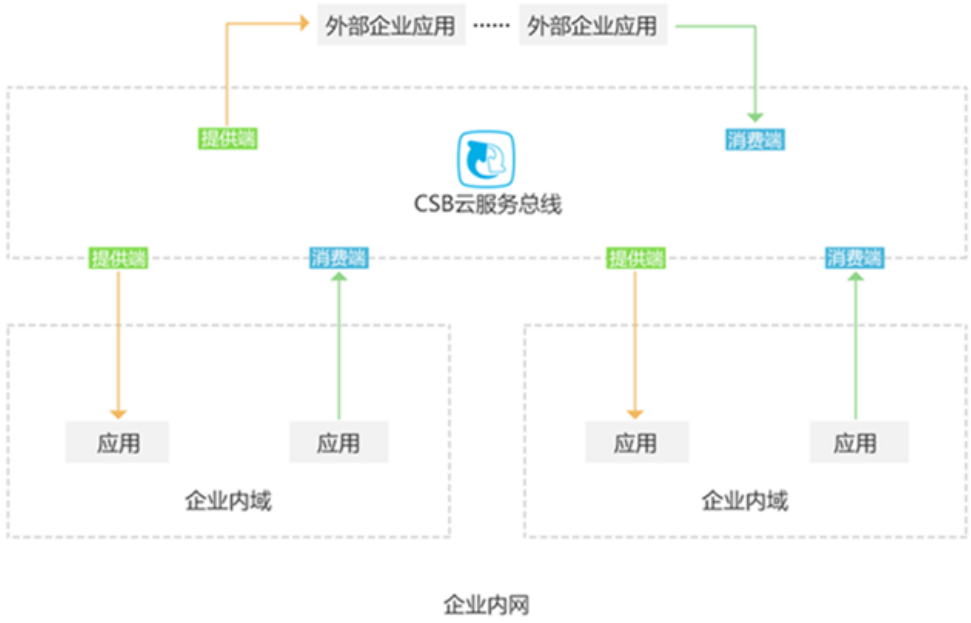


内外互通

企业内部以及与合作伙伴和第三方的系统通过CSB可管可控地开放互通。

- 企业内部以及合作伙伴和第三方都可以在CSB上开放订购服务；
- 各自的服务通过授权做访问控制（包括流量控制等）；
- 所有开放的服务在CSB上形成由企业自主管理的综合服务集市。

图 758: 内外互通

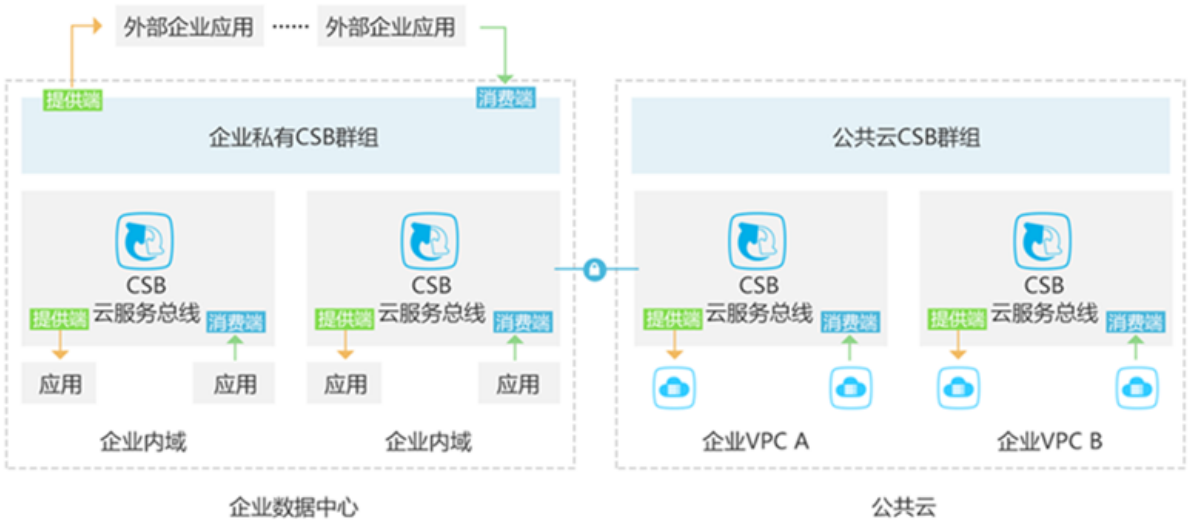


混合云、多群组

企业内部以及企业在公共云上的虚拟专有云（VPC）之间互通。

- 企业内部某个CSB与其在公共云上的某个VPC内的CSB通过专线互通；
- 公共云上的多个企业VPC之间通过各自的CSB互通；
- 每个CSB（实例）上的用户和服务由CSB实例拥有者自主管理。

图 759: 混合云与多群组



25.5 发布API服务

发布API服务就是把一个已有的后端服务在某个CSB实例上注册，并以选定的一种或多种协议开放成API供消费者使用，同时对服务的消费做一定的访问控制。

发布API服务包括三个关键概念：

- 接入服务：提供API对应的后端服务信息，让CSB能访问到这个已有的服务。
- 开放服务：指明API开放的协议，以及开放的接口和后端服务接口如何对应。
- 访问控制：指明API开放的策略，是否限流，对谁可见，访问是否需要授权。

图 760: 发布API服务



25.5.1 创建API分组

API组是业务上的原子粒度分组，每一个API都归属且仅归属一个API组。发布API之前必须指定归属API组（服务组）。

背景信息

如果当前用户还没有任何API组，请按以下步骤进行创建。

操作步骤

1. 在控制台**实例列表**页，单击需要操作的实例名称进入实例详情页。
2. 在左侧菜单栏单击**我的服务组**，并单击页面右上角的**新建服务组**。

图 761: 新建服务组



3. 在**新建服务组**对话框，按照提示输入相关信息并单击**确定**。

说明:

- 您可以上传一个服务接口定义文件，以方便发布服务时直接从这个接口定义文件选择接口方法，自动生成入参出参列表。
- 上传的接口定义文件需要以JAR方式提供，包含定义API所用的接口类以及所有依赖的参数类。
- 上传的JAR包编译Target版本必须为1.7或以下版本。
- JAR文件需要提供一个接口声明列表文件META-INF/api.list。内容格式如下面的示例：

```
# 包含2个服务接口ServiceInterfaceC和ServiceInterfaceD
a.b.c.ServiceInterfaceC
a.b.d.ServiceInterfaceD
# 注释掉的服务接口
#a.b.e.ServiceInterfaceE
```

- 您也可以创建多个API组，每个API组都应提供负责人信息。

25.5.2 发布服务（旧版）

发布API服务包括几个步骤：命名服务、选择接入和开放协议、设定参数、限制访问、发布服务。

背景信息

请根据控制台的引导完成服务发布。

操作步骤

1. 在实例详情页，单击左侧菜单栏的**发布服务**。
2. 在**命名服务**页输入服务全名等信息，并单击**下一步**。

图 762: 命名服务



说明:

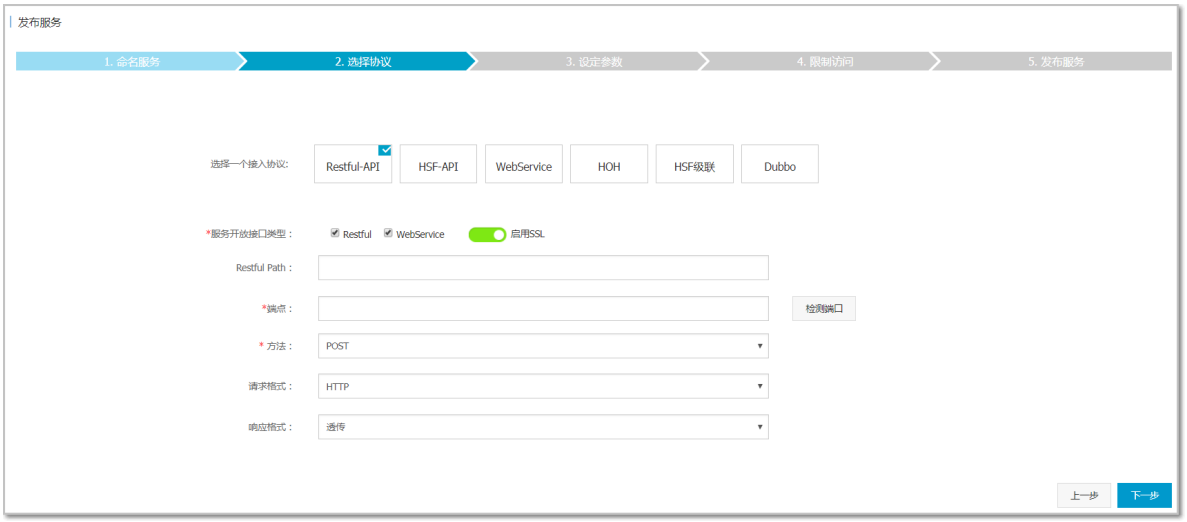
输入项说明：

- **服务全名**：必须为CSB群组上唯一。在注销服务后可以复用。
- **服务版本**：指该API在CSB上发布时指定的版本，即CSB发布版本，而非现有服务本身版本。CSB访问后端服务时不会使用该信息。同一个API可以有多个发布版本，但只能指定其中一个版本为当前激活版本。目前只有以特定协议（如HTTP API）调用该API时需要明确指明调用的API发布版本。
- **服务别名**：无格式要求。
- **所属服务组**：即服务所归属的服务组（API组），在API数量少时API组可视作简单分类。
- **服务说明**：选填，可以填写API的详细描述或者其他说明信息。

3. 在**选择协议**页面，选择要接入服务的协议，以便CSB可以访问要接入的已有后端服务，并确定以什么协议开放成API。

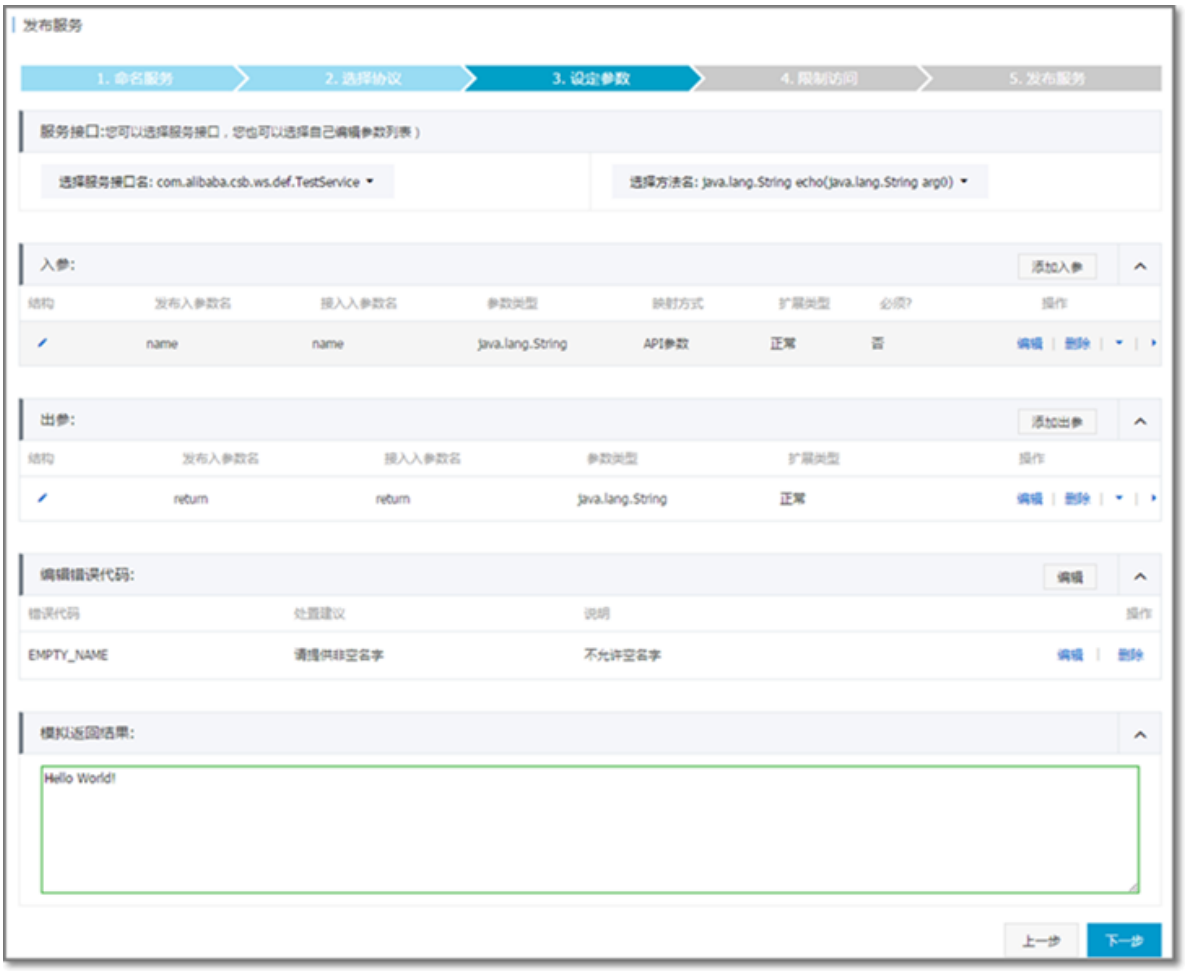
说明: 不同的接入协议类型，CSB需要的服务接入信息也不同。开放的协议可以多选，针对不同的接入协议类型，CSB支持的可开放协议类型有一定的限制。

图 763: 选择协议



4. 单击**下一步**，进入**设定参数**页并设定相关参数。

图 764: 设定参数



说明:

- 在参数设置中，除了一条一条地录入，后端服务的入参和出参列表可以通过之前在API组（服务组）中上传的接口定义文件中选择对应的接口和方法来生成，然后再做适当修改。
- 每一个入参都可以选择是否在API中开放。如果不开放但又是后端服务必要的入参，需要指定缺省值。
- 参数的编辑支持平级插入（向下的三角箭头）和缩进插入（向右的三角箭头），也支持数组、列表、集合等类型，可以定义复杂的多级结构。
- 可以输入服务本身的错误代码列表，给服务的使用者参考。
- 指定服务的模拟返回结果是指当用户以模拟方式调用时不会真的调用后端接口，而是直接返回模拟结果。

5. 单击**下一步**，进入**限制访问**页，指定该API的受访限制信息。

图 765: 限制访问



说明:

- **允许不授权访问**是指允许消费者在订阅该API的时候无需发布者审批即可访问。
- **可见域限制**是指谁可以看到并订购消费该API，即所谓的API可见域（可见域的功能暂未生效）。

6. 单击下一步进入**发布服务**页，确认信息无误后单击**发布**。

完成以上步骤后，一个API服务就发布完成了。

25.6 管理发布的API

API发布以后，您可以在CSB控制台进行一系列管理操作。

针对已发布的API，可以在API管理页面进行查看、启动、停止、注销等操作，也可以返回编辑API详情。服务发布者还可以对订阅者发起的订阅请求进行授权审批。

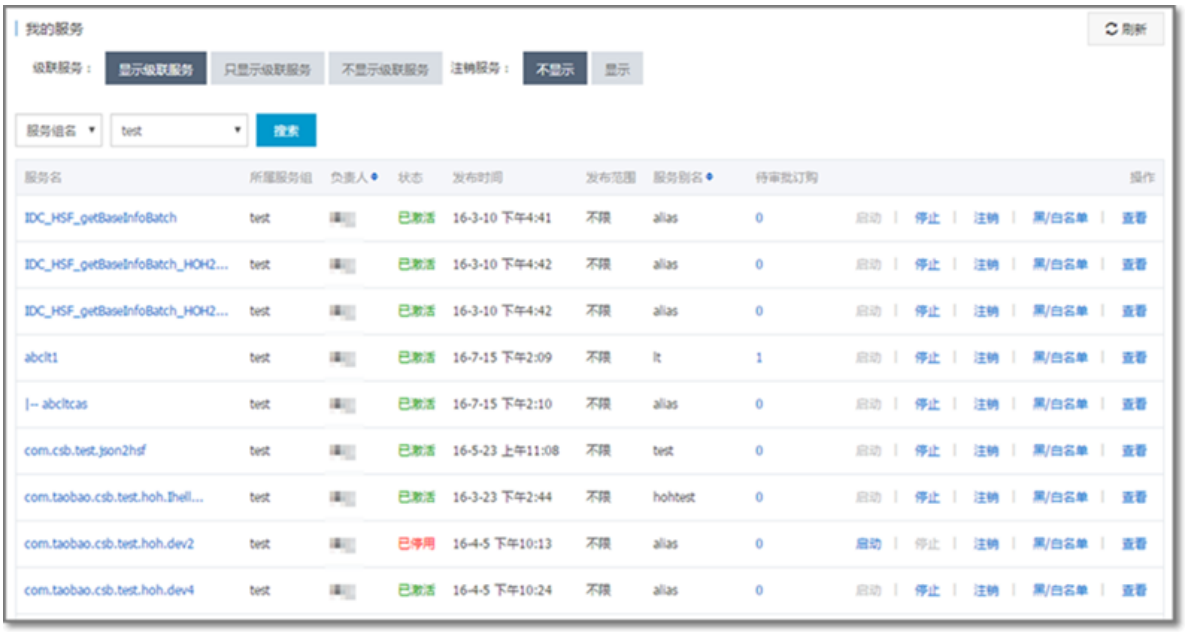
25.6.1 查看发布的API列表

在CSB控制台可以查看当前用户发布的所有API，包括已经注销的服务。同时还可以进行服务启动、停止、注销等操作。

操作步骤

1. 在CSB控制台的**实例列表**页，单击需要操作的实例名称。
2. 在实例概览页，单击左侧菜单栏的**发布者** > **我的服务**，可以看到该实例下的服务列表，包括已注销的服务。

图 766: API服务列表



说明:

- 通过列表上方的过滤开关可以选择是否显示已注销的服务，以及是否列出级联类型的服务。列表中服务名以“|--”、“—-”、“--|”开头的服务都是级联发布的服务，分别代表当前实例为该级联的起始端（即已有服务的接入端）、接力端，和结束端（即提供给消费者调用的开放端）。
- 如果列表太长，可以通过服务名、服务别名、服务组名以及项目名搜索服务。

3. （可选）单击右侧的操作选项启动、停止、或者注销服务。

说明: 服务注销意味着针对该服务的所有订购都将无法访问，订购者在自己的订购列表可以看到该服务已处于注销状态。

4. （可选）单击**黑/白名单**为每个API设定黑白名单，对可以访问该API的IP进行限定。

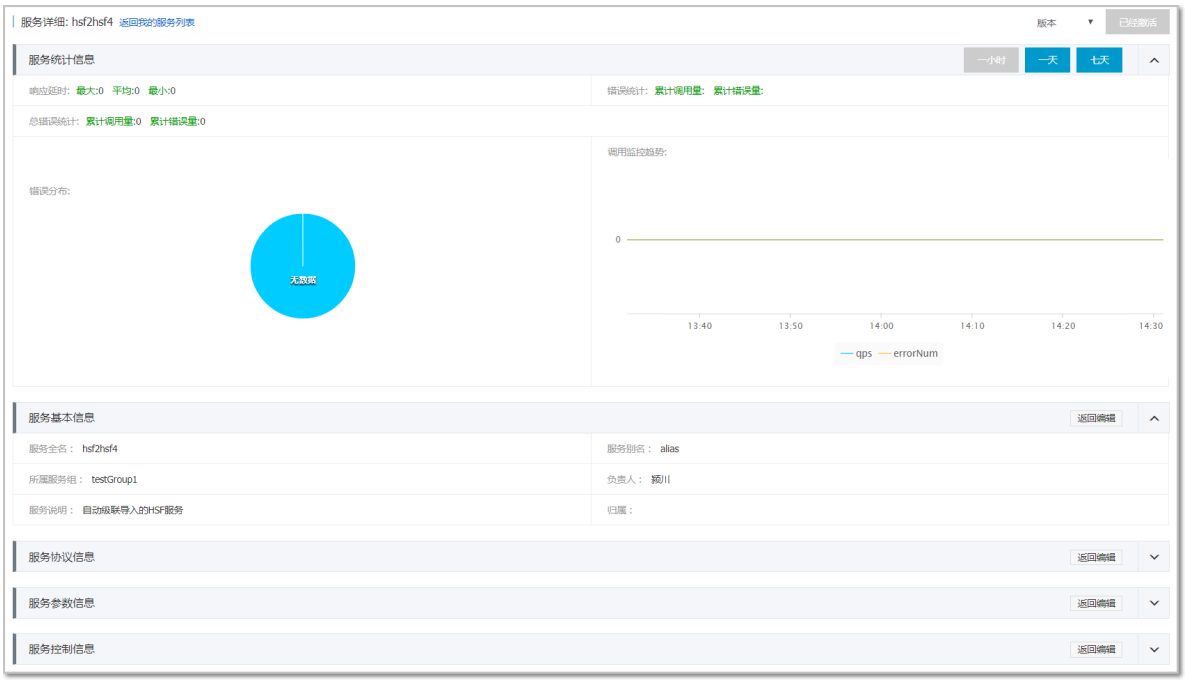
25.6.2 查看与编辑API详情

在服务详情页面，可以看到所发布服务的调用统计情况，服务的基本信息、协议信息、参数信息以及控制信息。

操作步骤

1. 在CSB控制台，具体实例的详情页面左侧的导航栏中，选择**发布者** > **我的服务**。
2. 在**我的服务**页面的服务列表中，单击某个服务右侧的**查看**按钮，进入服务详情页。

图 767: API详情



说明: 在详情页可以看到所发布服务的调用统计情况，服务的基本信息、协议信息、参数信息以及控制信息。各部分的信息都可以折叠或者展开查看。

3. （可选）如需修改API信息，可以单击**返回编辑**按钮进行更新。
4. （可选） 如需查看并激活某个特定的版本，单击页面右上角的下拉列表可以选择当前API的指定版本进行展示。单击**激活该版本**可以使当前版本进入激活状态。

25.6.3 授权API订阅

如果没有特别指定，服务的订阅需要发布者审批才会生效。

背景信息

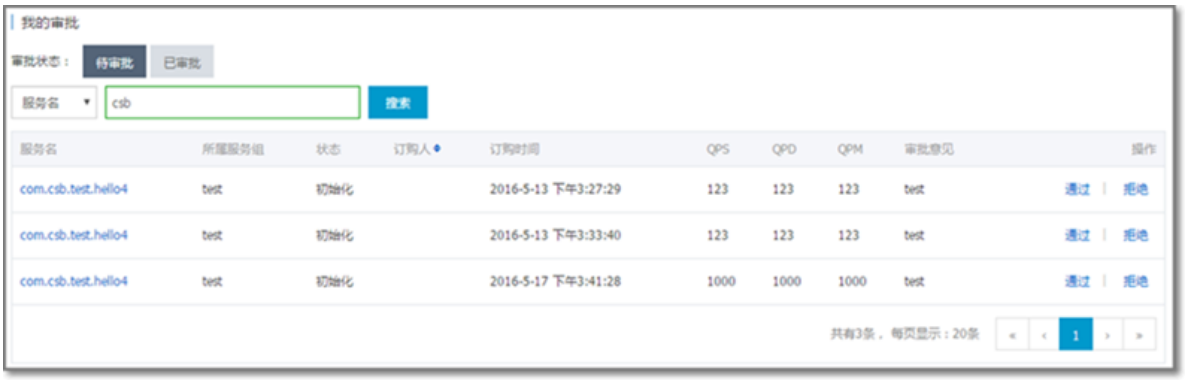
服务发布者可以按照以下步骤进行API授权。

操作步骤

1. 在CSB控制台**实例列表**页单击需要操作的实例名，进入实例概览页面。
2. 在实例概览页，单击左侧菜单栏**发布者** > **我的审批**，可以查看当前用户所发布服务的授权状况。

说明: 可以通过列表上方的过滤开关选择是否只列出待审批的授权。

图 768: API授权



3. 单击对应服务名右侧的**通过**或者**拒绝**进行API访问的授权。

25.7 订阅 API

服务发布者发布某个API以后，服务消费者就可以进行订阅。

服务消费者需要先创建消费凭证，用于服务订阅时的身份验证。消费者提交服务订阅请求后，服务发布者可以针对订阅请求进行授权，并设定访问频度、访问优先级等。

25.7.1 创建消费凭证

凭证代表服务API的消费方应用。如果订阅方还没有任何凭证，需要先创建一个。一个用户可以建多个凭证代表不同的应用或者应用组。

操作步骤

1. 在CSB控制台**实例列表**页单击需要操作的实例名称，进入实例管理页面。
2. 在实例概览页，单击左侧菜单栏的**订阅者** > **我的凭证**。在此页面可以看到当前用户所有的消费凭证列表。
3. 在**我的凭证**页面，单击右上角的**新建凭证**按钮进行创建。

图 769: 创建凭证



说明:

- 创建凭证只需要指定凭证名即可，凭证的命名通常可以用该凭证要代表的应用名、应用组名、系统名等。
- 凭证命名可以包括英文字符和“.”、“-”、“_”三个特殊字符。

凭证使用时，如果怀疑当前凭证已经泄露，可以更新（AK/SK）。由于使用该凭证的应用需要一段时间才能更新为新的凭证信息，通常需要一个时间窗口让新旧的凭证同时生效。此时可以先单击凭证右侧的**更新**操作，生成新的凭证信息，这时新旧凭证信息都是可用的状态。等旧的凭证信息确认不再使用后，再单击**取代**将旧的凭证信息废止，新的凭证会取代成为当前凭证。

25.7.2 订阅API服务

创建好消费凭证后，服务消费者就可以订阅API。

操作步骤

1. 在**实例列表**页单击实例名称进入实例管理页面。
2. 在左侧菜单栏选择**订阅者** > **订购服务**，可以看到当前用户可见的API列表。

   说明: 如果列表太长，可以通过API名称、别名以及服务组名进行查找和过滤。

3. 定位到需要订阅的API后，单击右侧的**查看详情与订购**，会弹出**服务订购**对话框。
4. 在对话框输入相关信息，单击**订购服务**发起订购请求。

图 770: 订阅API



订阅完成后，通常需要服务发布者进行授权。具体请参考*API授权*。

25.8 管理订阅的API

对应发布API管理，CSB同样提供了服务订阅者的管理平台。

在CSB控制台，您可以查看当前用户订阅的API列表以及每个API的详细信息，以及相应凭证消费API的统计信息，同时还可以进行API退订等操作。

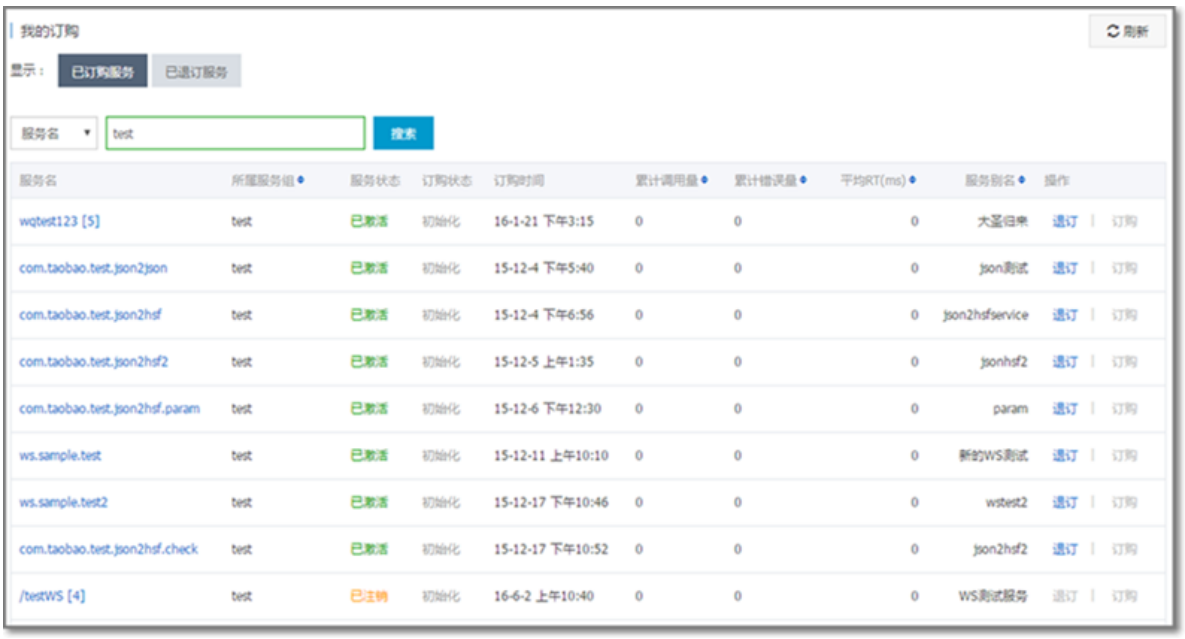
25.8.1 查看及退订API订阅

在CSB控制台可以查看当前用户订阅的所有API，并根据需要进行退订。

操作步骤

1. 在CSB控制台的**实例列表**页，单击需要操作的实例名称。
2. 在实例概览页，单击左侧菜单栏的**订阅者** > **我的订购**，可以看到当前用户订购的API列表，以及所订阅的API的当前状态，如**已激活**、**已停用**、**已注销**等。

图 771: API订购列表



说明:

- 通过列表上方的过滤按钮可以选择显示用户的**已订购服务**或者**已退订服务**。
- 当前用户可以用多个凭证订购同一API。如果订阅状态是**已退订**，则表示当前用户用于订阅该API的所有凭证都已退订。

3. 在API列表页，单击该API右侧对应的**退订**按钮，则所有订阅该API的用户凭证都退订。

说明: 如需指定某个凭证进行退订，请在API列表中单击该API的名称，并在弹出的对话框中选定某个凭证并单击右侧的退订。

图 772: 服务-凭证订阅关系



已退订的API也可以在订购列表页或者上述凭证对话框选择再次订阅。

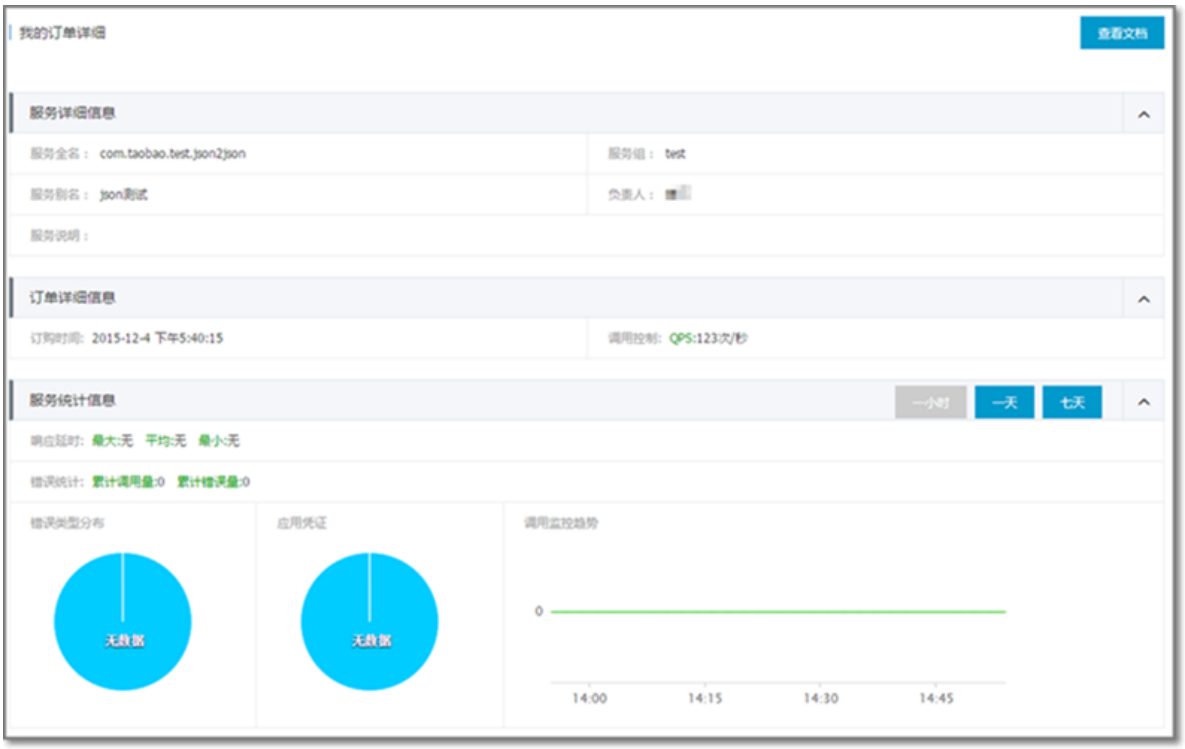
25.8.2 查看API订阅详情

在CSB控制台可以查看所订阅API的详细信息，包括当前凭证消费该API的统计情况。

操作步骤

1. 在CSB控制台**实例列表**页单击需要查看的实例，进入实例概览页。
2. 在左侧菜单栏单击**订阅者** > **我的订购**。
3. 在订购的API列表页，单击需要查看的API服务名称，会弹出该服务-凭证订阅关系对话框。
4. 单击某个凭证名称即可查看该凭证的订阅详情。

图 773: 已订阅的API详情



说明: 单击右上角的查看文档可以查看该API更详细的信息。

25.9 API调用

API消费方应用调用API有多种方式，例如HTTP API、HSF API、Web Service API等。

不同方式API调用说明如下：

- HSF API的消费调用沿用HSF原有的服务调用方式，无需任何专用SDK，如有必要可以指定CSB的服务IP进行HSF服务调用。
- 调用HTTP API或者Web Service API则需要使用CSB Client SDK，目前提供了Java和PHP版本的Client SDK。

对于开放成Web Service协议有两点说明：

- 如果接入是HTTP协议，则开放出的WSDL地址是：http://broker-vip:9081/$api_name/$api_version/ws2restful?wsdl。

  注意: **ws2restful**为固定值。

- 如果接入是HSF协议，则开放出来的WSDL地址是：http://broker-vip:9081/$api_name/$api_version/$method?wsdl。

详细的SDK使用说明，包括调用请求的签名方式，SDK与示例的下载，以及SDK具体使用方式，请参考《阿里云云服务总线*CSB SDK*参考手册》。

26 MaxCompute

26.1 前言

本文档针对不同的阅读对象，给出不同的建议。

如果您是一个初学者

如果您是一个初学者，建议您从如下章节开始读起：

- 产品简介：MaxCompute产品的总体介绍以及包含的主要功能。通过阅读该章节，您会对MaxCompute有一个总体的认识。
- 快速开始：通过实例一步一步指引您如何申请账号，如何安装客户端，如何创建表，如何授权，如何导入导出数据，如何运行SQL任务，如何运行UDF，如何运行Mapreduce程序等。
- 基本介绍：MaxCompute基本概念、MaxCompute常用命令介绍。您可以进一步熟悉如何操作MaxCompute。
- 工具：在分析数据之前，您可能需要掌握MaxCompute常用工具的下载、配置以及使用方法。

如果您是一个数据分析师

如果您是一个数据分析师，您可能需要熟读如下章节：

SQL使用指南：您可以查询并分析存储在MaxCompute上的大规模数据。包含的主要功能如下：

- 支持DDL语句，您可以通过Create、Drop和Alter对表和分区进行管理。
- 您可以通过Select选择表中的某几条记录，通过Where语句查看满足条件的记录，实现过滤功能。
- 您可以通过等值连接Join实现两张表的关联。
- 您可以通过对某些列Group By，实现聚合操作。
- 您可以通过Insert overwrite/into把结果记录插入到另一张表中。
- 你可以通过内置函数和自定义函数（UDF）来实现一系列的计算。
- 支持收集表的统计信息和设置表生命周期。
- 支持正则表达式。

如果您是一个有开发经验的用户

如果您是一个有开发经验的用户，并且对分布式概念有一定程度的了解。同时，由于某些数据分析无法使用SQL来实现，此时建议您学习MaxCompute更高级的功能模块：

- MapReduce：MaxCompute提供的Java MapReduce编程模型。您可以使用MapReduce提供的接口（Java API）编写MapReduce程序，用于处理MaxCompute的中的数据。
- Graph：一套面向迭代的图计算处理框架。使用图进行建模，图由点（Vertex）和边（Edge）组成，点和边包含权（Value）。通过迭代对图进行编辑、演化，最终求解出结果。
- Eclipse Plugin：方便用户使用MapReduce，UDF以及Graph的Java SDK进行开发工作。
- SDK：提供给开发者的工具包。
- Tunnel：您可以使用Tunnel服务向MaxCompute批量上传离线数据，或者从MaxCompute下载离线数据。
- DataHub服务：您可以使用DataHub服务实现实时数据的上传和下载。

如果您是项目Owner或者管理员

如果您是一个项目空间的Owner或者管理员，您需要熟读如下章节：

安全指南：您可以通过阅读该章节，了解如何给用户授权，如何进行跨项目空间的资源共享，如何设置项目空间的数据保护功能，如何进行policy授权等。

26.2 产品简介

26.2.1 MaxCompute的定义

大数据计算服务（MaxCompute）是阿里巴巴自主研发的海量数据处理平台。主要服务于批量结构化数据的存储和计算，可以提供海量数据仓库的解决方案以及针对大数据的分析建模服务。

随着社会数据收集手段的不断丰富及完善，越来越多的行业数据被积累下来。数据规模已经增长到了传统软件行业无法承载的海量数据（百GB、TB、乃至PB）级别。 在分析海量数据场景下，由于单台服务器的处理能力限制，数据分析者通常采用分布式计算模式。但分布式的计算模型对数据分析人员的能力要求较高， 且不宜维护。使用分布式模型，数据分析人员不仅需要了解业务需求，同时还需要熟悉底层计算模型。

MaxCompute的目的是为用户提供一种便捷的分析处理海量数据的手段。用户可以不必关心分布式计算细节，从而达到分析大数据的目的。MaxCompute已经在阿里巴巴集团内部得到大规模的应用，例如：大型互联网企业的数据仓库和BI分析、网站的日志分析、电子商务网站的交易分析、用户特征和兴趣挖掘等。

26.2.2 MaxCompute功能组件

MaxCompute提供了数据上传和下载通道，提供了SQL及MapReduce等多种计算分析服务，同时还提供了完善的安全解决方案。其包括的功能有：

- 数据通道。
  - Tunnel：提供高并发的离线数据上传和下载服务。用户可以使用Tunnel服务向MaxCompute批量上传数据，或者将数据从MaxCompute下载到本地。Tunnel仅提供Java编程接口供用户使用。
  - DataHub：向用户提供数据的实时上传和下载的功能。与Tunnel服务不同，通过DataHub上传的数据会即刻在用户数据中体现。
- 计算及分析任务。
  - SQL：MaxCompute只能以表的形式存储数据，并且对外提供了SQL查询功能。用户可以将MaxCompute作为传统的数据库软件操作，但其却能处理TB、PB级别的海量数据。需要注意，MaxCompute SQL不支持事务、索引及Update/Delete等操作。同时MaxCompute的SQL语法与Oracle，MySQL有一定差别，用户无法将其他数据库中的SQL语句无缝迁移到MaxCompute上来。此外，在使用方式上，MaxCompute SQL最快可以在分钟或者秒级别完成查询，无法在毫秒级别返回查询结果。MaxCompute SQL的优点体现在用户的学习成本低，用户不需要了解分布式概念，具备数据库操作经验的用户可以快速熟悉MaxCompute SQL的使用。
  - MapReduce：MapReduce最早是由Google提出的分布式数据处理模型，随后受到了业内的广泛关注，并被大量应用到各种商业场景中。在本文档中，会对MapReduce模型做简要介绍，以便于用户快速熟悉、了解该模型。使用MapReduce的用户需要对分布式概念有基本了解，并有对应的编程经验。MapReduce为用户提供Java编程接口。
  - Graph：MaxCompute提供的Graph功能是一套面向迭代的图计算处理框架。图计算作业使用图进行建模，图由点（Vertex）和边（Edge）组成，点和边包含权值（Value）。通过迭代对图进行编辑、演化，最终求解出结果，典型应用有：*PageRank*，单源最短距离算法，*K-* 均值聚类算法等。
- SDK：提供给开发者的工具包，SDK的相关介绍请参考：*MaxCompute SDK* 介绍。
- 安全：MaxCompute提供了功能强大的安全服务，为用户的数据安全提供保护，详情请参考：*MaxCompute* 安全指南。

各个功能模型的详细描述请参考用户手册的对应章节。如果想快速了解如何使用MaxCompute，请参考：快速开始。

26.3 快速开始

26.3.1 配置客户端（Console）

背景信息

MaxCompute的各项功能都可以通过客户端#Console#来访问。由于客户端是使用Java开发的，因此要确保本地PC上安装有JRE1.8。同时，用户需要提前准备好云账号并获取**Access Key ID**和**Access Key Secret**。



说明: 用户在开始配置客户端前，请确认已创建项目空间，且获取其**Access ID**和**Access Key**。

操作步骤

1. 下载客户端压缩包到本地PC。
2. 将下载好的客户端压缩包解压到一个文件夹中，在该文件夹中可以看到如下4个文件夹。

```
bin/
conf/
lib/
plugins/
```

3. 编辑conf文件夹中的**odps_config.ini**文件，配置如下相关信息。

```
project_name=my_project
access_id=**********************
access_key=**********************
end_point= <MaxCompute服务地址>
```



说明:

- Access ID和Access Key是用户的云账号信息。
- Project_name=my_project的作用是指定用户想进入的项目空间。一旦配置此项，每次登录客户端，将会默认进入到该项目空间。如果不配置，登录客户端后，必须使用use project_name命令进入项目空间。
- end_point项需要配置成MaxCompute提供给用户的服务地址。针对不同的用户，服务地址也不同。
- 关于客户端的详细介绍请参考：*MaxCompute客户端*。

4. 修改好配置文件后，运行bin目录下的**odps**文件（在Linux 系统下运行*./bin/odpscmd*，在Windows下运行*./bin/odpscmd.bat*），此时就可以开始运行SQL语句了，示例如下。

```
create table tbl1(id bigint);
insert overwrite table tbl1 select count(*) from tbl1;
select 'welcome to MaxCompute!' from tbl1;
```

说明: 更多SQL语句的介绍请参考：*SQL* 使用指南。

26.3.2 添加删除用户

任意非项目空间Owner用户必须被加入MaxCompute项目空间中，并被授予对应的权限，才能在MaxCompute中的对应项目空间中进行操作。

本章节将介绍项目空间Owner如何将其他用户加入对应的项目空间或者从对应的项目空间中删除。

如果您是项目空间Owner，建议您仔细阅读本章节；如果您是普通用户，建议您向项目空间Owner提出申请，被加入对应的项目空间后再阅读后续章节。

- **添加用户**

  添加用户的命令为：

  ```
  ADD USER <full_username>;
  ```

  通过客户端向项目空间添加用户的示例如下：

  ```
  add user bob@aliyun.com;
  ```

  如果不确定该用户是否已经存在于此项目空间，可以通过如下命令来查看：

  ```
  LIST USERS;
  ```

  说明:

  - 添加用户成功后，被添加的用户不能立刻在MaxCompute中对应的项目空间进行操作，需要项目空间Owner对用户授予一定权限，该用户才能在所拥有的权限范围内进行操作。
  - 授权的相关操作请参考：用户授权及权限查看。

- **删除用户**

删除用户的命令为：

```
REMOVE USER <full_username> ;
```

通过客户端在项目空间删除用户的示例如下：

```
remove user bob@aliyun.com;
```

说明:

- 在删除用户前，请确保该用户的所有权限已经被取消。若用户被删除，但该用户的权限未被取消，则该用户的权限仍然保留，当该用户再次被加入对应项目空间时，原有权限将被恢复。
- 添加、删除用户的更多信息请参考：项目空间的用户管理。

26.3.3 用户授权及权限查看

在添加用户后，需要给用户授权。只有用户获得权限后，才能实行相关操作。所谓授权，即授予用户对MaxCompute中的Object（或称之为客体，例如：表，任务，资源等）某种操作权限，包括：读、写、查看等。

本章节主要针对项目空间的管理员用户。如果您只是MaxCompute的普通用户，请确认已经获取足够权限，快速浏览本章节即可。

MaxCompute 的授权分为 *ACL* 授权及 *Policy* 授权两种不同的授权策略。

26.3.3.1 ACL授权

ACL授权中，MaxCompute的客体包括：

Project，Table，Function，Resource，Instance，Task。每种客体具有不同的操作权限，详细介绍请参考：*ACL* 授权。

授权的命令为：

```
GRANT privileges ON project_object TO project_subject
REVOKE privileges ON project_object FROM project_subject
privileges ::= action_item1, action_item2, ...
project_object ::= PROJECT project_name | TABLE schema_name |
INSTANCE inst_name | FUNCTION func_name |
RESOURCE res_name | JOB job_name
project_subject ::= USER full_username | ROLE role_name
```

说明: 上述授权命令中有关角色（ROLE）的相关部分用户可以暂时不予关注，待后续章节进行介绍。

授权的示例如下：

```
grant CreateTable on PROJECT $user_project_name to USER bob@aliyun.com;
-- 向bob@aliyun.com授予名为“$user_project_name”的project的CreateTable（创建表）权限
grant Describe to Table $user_table_name to USER bob@aliyun.com;
-- 向bob@aliyun.com授予名为“$user_table_name”的Table的Describe（获取表信息）权限
grant Execute on Function $user_function_name to USER bob@aliyun.com;
-- 向bob@aliyun.com授予名为"$user_function_name"的Function的Execute（执行）权限
```

26.3.3.2 Policy授权

Policy授权是一种基于主体的授权，Policy授权的相关介绍请参考 *Policy* 权限策略。

授权的命令为：

```
GET POLICY;
PUT POLICY <policyFile>;
GET POLICY ON ROLE <roleName>;
PUT POLICY <policyFile> ON ROLE <roleName>;
```

授权的示例如下：

```
{
"Version": "1",
"Statement": [
{
"Effect": "Deny",
"Action": [
"odps:*"
],
"Resource": "acs:odps:*:projects/$user_project_name/tables/*",
 "Condition": {
"StringEquals": {
"odps:TaskType": [
"DT"
]
}
}
},
{
"Effect": "Allow",
 "Action": [
"odps:List",
"odps:Read",
"odps:Describe",
"odps:Select"
],
"Resource": [
 "acs:odps:*:projects/$user_project_name/tables/a",
"acs:odps:*:projects/$user_project_name"
],
"Condition": {
 "StringEquals": {
"odps:TaskType": [
                "SQL"
                 ]
        }
  }
```

```
}
]
}
```

说明: 上述示例的目的是禁用$user_project_name的Tunnel功能，并且仅授予用户对此Project及该项目中表a的List，Read，Describe及Select权限。

26.3.3.3 查看权限

如果想查看某个用户的权限，可以执行如下命令：

```
show grants for  $user_name;
```

说明: 关于查看权限的详细介绍请参考：权限查看。

26.3.4 角色创建及授权

由于所有被加入MaxCompute的用户只有被授予相应的权限后才能使用MaxCompute。当项目空间下有大量用户存在时，这个授权动作会非常繁琐。在这种情况下，项目空间管理员可以对拥有相同权限的用户进行分类，每类用户有相同的权限，我们称之为角色。多个用户可以同时存在于一个角色下，一个用户也可以隶属于多个角色。给角色授权后，该角色下的所有用户拥有相同的权限。

- **创建角色**

  创建角色的命令为：

  ```
  CREATE ROLE <roleName>;
  ```

  创建角色的示例如下：

  ```
  create role player;
  ```

- **向角色中添加用户**

  向角色中添加用户的命令为：

  ```
  GRANT <roleName> TO <full_username>;
  ```

  向角色中添加用户的示例如下：

  ```
  grant player to bob@aliyun.com;
  ```

- **删除角色**

将角色删除的命令为：

```
DROP ROLE <roleName>;
```

将角色删除的示例如下：

```
drop role player;
```

说明: 删除弃用的角色前，请确认所有该角色下的用户都已经被移出该角色。

给角色授权的语句与对用户授权类似，可以参考用户授权的示例对角色进行授权操作。更多有关角色授权的介绍请参考：角色管理。

26.3.5 创建删除表

用户加入项目空间并被授予权限之后，即可开始操作MaxCompute。本章内容将主要介绍用户如何进行表操作。

26.3.5.1 创建表

创建表的命令为：

```
CREATE TABLE [IF NOT EXISTS] table_name
[(col_name data_type [COMMENT col_comment], ...)] [COMMENT table_comment]
[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)] [LIFECYCLE days]
[AS select_statement]
CREATE TABLE [IF NOT EXISTS] table_name
LIKE existing_table_name
```

创建表的示例如下：

```
create table test1 (key string);
-- 创建非分区表
create table test2 (key bigint) partitioned by (pt string, ds string);
--创建分区表
create table test3 (key boolean) partitioned by (pt string, ds string) lifecycle 100;
-- 创建带有生命周期的表
create table test4 like test3;
-- 除生命周期属性外，test3的其他属性（字段类型，分区类型等）均与test4完全一致
create table test5 as select * from test2;
--  创建test5，但分区，生命周期信息不会被拷贝到目标表中。且仅会将test2的数据复制到test5中。
```

MaxCompute的表支持分区、生命周期。创建表的详细介绍请参考：创建表#*CREATE TABLE*#。修改分区请参考：添加及删除分区。生命周期的修改请参考：修改表的生命周期属性。

26.3.5.2 获取表信息

获取表信息的命令为：

```
desc <table_name>;
```

获取表信息的示例如下：

```
desc test3;
-- 获取test3信息。
desc test4;
-- 获取test4信息。
```

26.3.5.3 删除表

删除表的命令为：

```
DROP TABLE [IF EXISTS] table_name;
```

删除表的示例如下：

```
drop table test2;
```



说明: 更多介绍请参见：*删除表#DROP TABLE#*。

26.3.6 导入导出数据

MaxCompute的数据导入导出可以通过MaxCompute Tunnel提供的SDK自行编写Java工具。示例程序请参考：*Tunnel SDK示例*。

26.3.6.1 运行SQL

由于大多数用户对SQL的语法并不陌生，此处只简要介绍MaxCompute SQL使用中需要注意的问题，不再做操作示例。

注意事项如下：

- MaxCompute SQL不支持事务、索引及Update/Delete等操作。
- MaxCompute SQL语法与Oracle，MySQL有一定差别，无法将其他数据库中的SQL语句无缝迁移到MaxCompute中。
- 在使用方式上，MaxCompute SQL无法满足实时查询，查询计算时间在分钟级别，无法在秒乃至毫秒级别返回用户结果。

26.3.6.1.1 Select语句

Select语句的使用有如下限制：

- group by语句的key可以是输入表的列名，也可以是由输入表的列构成的表达式，不可以是select语句的输出列。

```
select substr(col2, 2) from tbl group by substr(col2, 2);
-- 可以，group by的key可以是输入表的列构成的表达式。
select col2 from tbl group by substr(col2, 2);
-- 不可以，group by的key不在select语句的列中。
select substr(col2, 2) as c from tbl group by c;
-- 不可以，group by的key不可以是列的别名，即select语句的输出列。
```

> **说明:** 之所以有上述限制，是因为通常在SQL解析中，group by的操作是先于select操作的，因此group by只能接受输入表的列或表达式为key。

- sort by前必须加distribute by。
- order by/sort by/distribute by的key必须是select语句的输出列，即列的别名。

```
select col2 as c from tbl order by col2 limit 100 -- 不可以，order by的key不是select语句的输出列，即列的别名。
select col2 from tbl order by col2 limit 100; -- 可以，当select语句的输出列没有别名时，使用列名作为别名。
```

> **说明:** 之所以有上述限制，是因为通常在SQL解析中，order by/sort by/distribute by的操作是后于select操作的，因此它们只能接受select语句的输出列为key。

更多Select语句详情请参考：*Select*语句。

26.3.6.1.2 Insert语句

Insert语句的使用有如下限制：

- 向某个分区插入数据时，分区列不可以出现在select列表中。

```
insert overwrite table sale_detail_insert partition (sale_date='2013', region='china') select shop_name, customer_id, total_price, sale_date, region from sale_detail;
-- 返回失败，sale_date, region为分区列，不可以出现在静态分区的insert语句中。
```

- 动态分区插入时，动态分区列必须在select列表中。

```
insert overwrite table sale_detail_dypart partition (sale_date='2013', region) select shop_name,customer_id,total_price from sale_detail;
-- 返回失败，动态分区插入时，动态分区列必须在select列表中。
```

更多Insert语句详情请参考：*Insert*语句。

26.3.6.1.3 Join操作

Join操作有如下使用限制：

- MaxCompute SQL支持的Join操作类型包括：{LEFT OUTER|RIGHT OUTER|FULL > OUTER|INNER} JOIN。
- MaxCompute SQL目前最多支持16个并发Join操作。
- 在mapjoin中，最多支持6张小表的mapjoin。

Join操作的更多详情请参考：*Join*语句。

26.3.6.1.4 其他

- MaxCompute SQL目前最多支持256个并发union操作。
- MaxCompute SQL目前最多支持256个并发insert overwrite/into操作。

26.3.7 编写并使用UDF

MaxCompute的UDF包括：UDF，UDAF，UDTF三种函数。通常情况下，此三种函数被统称为UDF。

说明:

- UDF目前只支持Java语言接口，用户如果想编写UDF程序，可以通过添加资源的方式将UDF代码上传到项目空间中，使用注册函数语句创建UDF。
- 本章节中会分别给出UDF，UDAF，UDTF的代码示例。

26.3.7.1 UDF示例

背景信息

本章节将以实现一个字符小写转换功能的UDF为例，给出完整的UDF开发流程示例，具体操作如下。

操作步骤

1. 代码编写。

   按照MaxCompute UDF框架的规定，实现函数功能，并进行编译。

```
package org.alidata.odps.udf.examples; import com.aliyun.odps.udf.UDF;
public final class Lower extends UDF { public String evaluate(String s) {
if (s == null) { return null; } return s.toLowerCase();
}
```

```
}
```

将上述jar包命名为*my_lower.jar*。

2. 添加资源。

   在运行UDF之前，必须指定引用的UDF代码。用户代码需要通过资源的形式添加到MaxCompute中。Java UDF必须被打成jar包，以jar资源添加到MaxCompute中，UDF框架会自动加载jar包，运行用户自定义的UDF。

   添加jar资源的示例命令如下：

   ```
   add jar my_lower.jar;
   ```

   说明: 如果存在同名的资源请将jar包重命名，并注意修改下面示例命令中相关jar包的名字；也可以直接使用-f选项覆盖原有的jar资源。

3. 注册UDF函数。

   用户的jar包被上传后，由于MaxCompute中并没有关于这个UDF的任何信息。因此需要在MaxCompute中注册一个唯一的函数名，并指定这个函数名与哪个jar资源的哪个函数对应。

   注册UDF函数的示例命令如下：

   ```
   CREATE FUNCTION test_lower AS org.alidata.odps.udf.examples.Lower USING my_lower.jar;
   ```

   在sql中使用此函数的示例如下：

   ```
   select test_lower('A') from my_test_table;
   ```

26.3.7.2 UDAF示例

UDAF的注册方式与UDF基本相同，使用方式与内建函数中的聚合函数相同。

UDAF示例代码如下：

```
package org.alidata.odps.udf.examples;
import com.aliyun.odps.io.LongWritable; import com.aliyun.odps.io.Text;
import com.aliyun.odps.io.Writable; import com.aliyun.odps.udf.Aggregator;
import com.aliyun.odps.udf.UDFException;
/**
project: example_project
table: wc_in2
partitions: p2=1,p1=2
columns: colc,colb,cola
*/
public class UDAFExample extends Aggregator {
@Override
public void iterate(Writable arg0, Writable[] arg1) throws UDFException { LongWritable result = (LongWritable) arg0;
for (Writable item : arg1) { Text txt = (Text) item;
```

```
result.set(result.get() + txt.getLength());
}
}
@Override
public void merge(Writable arg0, Writable arg1) throws UDFException { LongWritable result =
 (LongWritable) arg0;
LongWritable partial = (LongWritable) arg1; result.set(result.get() + partial.get());
}
@Override
public Writable newBuffer() { return new LongWritable(0L);
}
@Override
public Writable terminate(Writable arg0) throws UDFException { return arg0;
}
}
```

26.3.7.3 UDTF示例

UDTF的注册方式与UDF基本相同，使用方式也与UDF相同。

UDTF示例代码如下：

```
package org.alidata.odps.udtf.examples;
import com.aliyun.odps.udf.UDTF;
import com.aliyun.odps.udf.UDTFCollector; import com.aliyun.odps.udf.annotation.Resolve;
 import com.aliyun.odps.udf.UDFException;
// TODO define input and output types, e.g., "string,string->string,bigint".
@Resolve({"string,bigint->string,bigint"}) public class MyUDTF extends UDTF {
@Override
public void process(Object[] args) throws UDFException { String a = (String) args[0];
Long b = (Long) args[1];
for (String t: a.split("\\s+")) { forward(t, b);
}
}
}
```

26.3.8 编写并运行MapReduce

本章节将介绍安装好MaxCompute客户端后，如何快速运行MapReduce程序。

背景信息

以WordCount程序为例，具体操作如下。

编写并运行MapReduce程序前，请确认如下信息。

- 编译、运行MapReduce需要安装JDK1.8版本，请确认是否已按照对应版本。
- 需要配置好MaxCompute客户端，具体操作请参考：配置客户端。

操作步骤

1. 创建输入输出表。

建表语句示例如下：

```
create table wc_in (key string, value string);
create table wc_out (key string, cnt bigint);
```



说明: 创建表的语句请参考：创建表。

2. 使用数据上传下载工具上传数据。

   上传数据的示例命令如下：

```
odpscmd -e  “tunnel upload kv.txt wc_in”
```

3. 编写MaxCompute程序并编译。

   - MaxCompute为用户提供了便捷的Eclipse开发插件，方便用户快速开发MapReduce程序，并提供了本地调试MapReduce的功能。
   - 用户需要先在Eclipse中创建一个MaxCompute项目工程，而后在此工程中编写MapReduce程序。本地调试通过后，将编译好的程序上传至MaxCompute，最后进行base调试。



说明: 如果在java程序中使用了任何资源，请务必将此资源加入-resources参数。

26.3.9 编写并运行Graph

Graph作业的提交方式与MapReduce基本相同，本章节将举例说明如何提交Graph作业。

背景信息

以sssp算法为例，具体操作如下。

操作步骤

1. 创建输入输出表。

   建表语句示例如下：

```
create table sssp_in (v bigint, es string);
create table sssp_out (v bigint, l bigint);
```



说明: 创建表的语句请参考：创建表。

2. 使用数据上传下载工具上传数据。

上传数据的示例命令如下：

```
tunnel u -fd " " sssp.txt sssp_in;
```

3. 编写sssp示例。

   说明: 根据Graph开发插件的介绍，本地编译、调试sssp算法示例，仅需将sssp代码打包即可，不需同时将SDK打入*odps-graph-example-sssp.jar*中。

4. 添加jar资源。

   添加jar资源的示例命令如下：

   ```
   add jar $LOCAL_JAR_PATH/odps-graph-example-sssp.jar odps-graph-example-sssp.jar
   ```

   说明: 添加资源的详细介绍请参考：添加资源。

5. 运行sssp。

   运行sssp的示例命令如下：

   ```
   jar -libjars odps-graph-example-sssp.jar -classpath $LOCAL_JAR_PATH/odps-graph-example-sssp.jar com.aliyun.odps.graph.examples.SSSP 1 sssp_in sssp_out;
   ```

   说明:

   jar命令用于运行MaxCompute Graph作业，用法与MapReduce作业的运行命令完全一致。Graph作业执行时命令行会打印作业实例ID，执行进度，结果Summary 等，输出示例如下所示。

   ID = 20130730160742915gl205u3 2013-07-31 00:18:36 SUCCESS

   Summary：

   ```
   Graph Input/Output Total input bytes=211 Total input records=5 Total output bytes=161
   Total output records=5 graph_input_[bsp.sssp_in]_bytes=211


    graph_output_[bsp.sssp_out]_records=5 Graph Statistics
   Total edges=14
   Total halted vertices=5 Total sent messages=28 Total supersteps=4 Total vertices=5
   Total workers=1 Graph Timers
   Average superstep time (milliseconds)=7 Load time (milliseconds)=8
   Max superstep time (milliseconds) =14 Max time superstep=0
   Min superstep time (milliseconds)=5 Min time superstep=2
   Setup time (milliseconds)=277 Shutdown time (milliseconds)=20
   Total superstep time (milliseconds)=30 Total time (milliseconds)=344
   ```

```
OK
```

26.4 基本介绍

26.4.1 基本概念

- **项目空间**（Project）

  项目空间（Project）是MaxCompute的基本组织单元，它类似于传统数据库的Database或Scheme的概念，是进行多用户隔离和访问控制的主要边界。一个用户可以同时拥有多个项目空间的权限。通过安全授权，可以在一个项目空间中访问另一个项目空间中的对象，例如：表（Table）、资源（Resource）、函数（Function）、实例（Instance）。

  用户可以通过Use Project命令进入一个项目空间，示例如下：

  ```
  use my_project
  -- 进入一个名为my_project的项目空间。
  ```

  说明: 运行上述命令后，用户会进入一个名为my_project的项目空间，从而可以操作该项目空间下的对象，而不需要关心操作对象所在的项目空间。Use Project是MaxCompute客户端提供的命令，相关命令的具体说明请参考：*MaxCompute*常用命令。

- **表**

  表是MaxCompute的数据存储单元。它在逻辑上也是由行和列组成的二维结构，每行代表一条记录，每列表示相同数据类型的一个字段，一条记录可以包含一个或多个列，各个列的名称和类型构成这张表的schema。在MaxCompute中，所有的数据都被存储在表中，表中的列可以是MaxCompute支持的任意一种数据类型。MaxCompute中的各种不同类型计算任务的操作对象（输入、输出）都是表。用户可以创建表，删除表以及向表中导入数据。

  为了提高处理效率，可以在创建表时指定表的分区（Partition），即指定表内的某几个字段作为分区列。在大多数情况下，用户可以将分区类比为文件系统下的目录。 MaxCompute将分区列的每个值作为一个分区（目录）。用户可以指定多级分区，即将表的多个字段作为表的分区，分区之间类似多级目录的关系。在使用数据时如果指定了需要访问的分区名称，则只会读取相应的分区，避免全表扫描，提高处理效率，降低费用。相关的详细介绍请参考：列及分区操作。

- **数据类型**

  MaxCompute表中的列必须是下列描述的任意一种类型，MaxCompute支持的基本数据类型的描述及取值范围，如下表所示：

类型	常量定义	描述	取值范围
Tinyint	1Y，-127Y	8位有符号整形。	-128 ~ 127
Smallint	32767S， -100S	16位有符号整形。	-32768 ~ 32767
Int	1000，-15645787	32位有符号整形。	-2^{31} ~ $2^{31}-1$
Bigint	100000000000L， -1L	64位有符号整形。	$-2^{63}+1$ ~ $2^{63}-1$
String	“abc” ，’ bcd’ ，” all be a ‘inc’	字符串，支持UTF-8编码。其他编码的字符行为未定义。	单个String列最长允许8MB。
Float	无	32位二进制浮点型。	/
Boolean	True，False	布尔型。	True或False
Double	3.1415926 1E+7	64位二进制浮点型。	-1.0 10 308 ~ 1.0 10308
Datetime	Datetime ‘2017-11-11 00:00:00’	日期时间类型。使用东八区时间作为系统标准时间。	0001-01-01 00:00:00 000 ~ 9999- 12-31 23:59:59 999
Decimal	3.5BD，99999999999.9999999BD	10进制精确数字类型。	整形部分：$-10^{36}+1$ ~ $10^{36}-1$ 小数部分：精确到10^{-18}
Varchar	无	变长字符类型，n为长度。	1 ~ 65535
Binary	无	二进制数据类型。	单个Binary列最长允许8MB。
Timestamp	Timestamp ‘2017-11-11 00:00:00.123456789’	与时区无关的时间戳类型。	0001-01-01 00:00:00 000000000 ~ 9999- 12-31 23:59:59 999999999

注意:

- 上述的各种数据类型均可为NULL。

- 对于Int类型常量，如果超过Int类型取值范围，会转换为Bigint类型；如果超过Bigint类型取值范围，会转换为Double类型。MaxCompute在未设定**odps.sql.type.system.odps2**为**true**的情况下，保留此转换，但会报告一个警告，提示Int类型被当作Bigint类型处理了。如果用户的脚本有此种情况，建议全部改写为Bigint类型，避免混淆。
- Varchar类型常量可通过String类型常量的隐式转换表示。
- String类型常量支持连接，例如 ‘abc’ ‘xyz’ 会解析为 ‘abcxyz’ ，不同部分可以写在不同行上。

MaxCompute支持的复杂数据类型的定义及构造方法，如下表所示：

类型	定义方法	构造方法

Array	array< int >;
	array< struct< a:int, b:string >>
		array(1, 2, 3);
		array(array(1, 2);
		array(3, 4))

Map	map< string, string >; map< smallint, array< string>>	map(“k1”, “v1”, “k2”, “v2”); map(1S, array(‘a’, ‘b’), 2S, array(‘x’, ‘y))
Struct	struct< x:int, y:int>; struct< field1:bigint, field2:array< int>, field3:map< int, int>>	named_struct(‘x’, 1, ‘y’ , 2); named_struct(‘field1’, 100L, ‘field2’ , array(1, 2), ‘field3’ , map(1, 100, 2, 200)

- **资源**

  资源（Resource）是MaxCompute的特有概念。用户如果想使用MaxCompute的自定义函数（UDF）或MapReduce功能需要依赖资源来完成。

  - MaxCompute SQL UDF：用户在编写UDF后，需要将编译好的jar包作为资源上传到MaxCompute。运行UDF时，MaxCompute会自动下载这个jar包，获取用户代码，从而运行UDF，无需用户干预。上传jar包的过程就是在MaxCompute上创建资源的过程，这个jar包是MaxCompute资源的一种。

- MaxCompute MapReduce：用户编写MapReduce程序后，将编译好的jar包作为资源上传到MaxCompute。运行MapReduce作业时，MapReduce框架会自动下载这个jar资源，获取用户代码，从而运行MapReduce作业，无需用户干预。

  说明:

  - MaxCompute的自定义函数（UDF）或MapReduce对资源的读取有一定的限制，详情请参考：应用限制。
  - 用户也可以将文本文件以及MaxCompute中的表作为不同类型的资源上传到MaxCompute。用户可以在UDF及MapReduce的运行过程中读取、使用这些资源， MaxCompute提供了读取、使用资源的接口。详细示例请查看资源使用示例及*UDTF*使用说明中的描述。

MaxCompute资源的类型包括：

- File类型。
- Table类型：MaxCompute中的表。
- Jar类型：编译好的Java Jar包。
- Archive类型：通过资源名称中的后缀识别压缩类型，支持的压缩文件类型包括：.zip/.tgz/.tar.gz/.tar/jar。
- Py类型：Python脚本，供Python UDF使用。

说明: 有关资源的操作介绍请参考：资源操作。

- **自定义函数**

MaxCompute为用户提供了SQL计算功能，用户可以在MaxCompute SQL中使用系统的内建函数完成一定的计算和计数功能。但当内建函数无法满足要求时，用户可以使用MaxCompute提供的Java编程接口开发自定义函数（User Defined Function，以下简称UDF）。自定义函数（ UDF）又可以进一步分为标量值函数（UDF）、自定义聚合函数（UDAF）和自定义表值函数（UDTF）三种。

用户在开发完成UDF代码后，需要将代码编译成jar包，并将此jar包以jar资源的形式上传到MaxCompute，最后在MaxCompute中注册此UDF。在使用UDF时，只需要在SQL中指明UDF的函数名及输入参数即可，使用方式与MaxCompute提供的内建函数相同。

说明: 有关UDF的操作介绍请参考：函数操作。

- **任务**

  任务（Task）是MaxCompute的基本计算单元。SQL及MapReduce功能都是通过任务（Task）完成的。对于用户提交的大多数任务，特别是计算型任务（例如SQL DML语句、MapReduce等），MaxCompute会对其进行解析，得出任务的执行计划。

  执行计划是由具有依赖关系的多个执行阶段（Stage）构成的。目前，执行计划逻辑上可以被看做一个有向图，图中的点是执行阶段，各个执行阶段的依赖关系是图的边。MaxCompute会依照图（执行计划）中的依赖关系执行各个阶段。在同一个执行阶段内，会有多个进程，也称之为Worker，共同完成该执行阶段的计算工作。同一个执行阶段的不同Worker只是处理的数据不同，执行逻辑完全相同。

  计算型任务在执行时，会被实例化，用户可以操作这个实例（Instance）的信息，例如：获取实例状态（ Status Instance ）、终止实例运行（Kill Instance ）等。

  另一方面，部分MaxCompute任务并不是计算型的任务（例如SQL中的DDL语句），这些任务本质上仅需要读取、修改MaxCompute中的元数据信息。因此，这些任务无法被解析出执行计划。

  > **注意:** 在MaxCompute中，并不是所有的请求都会被转化为任务（Task），例如项目空间（Project）、资源（Resource）、自定义函数（UDF）及实例（Instance）的操作就不需要通过MaxCompute的任务来完成。

- **任务实例**

  在MaxCompute中，部分任务（Task）在执行时会被实例化，以MaxCompute实例（下文简称实例或 Instance）的形式存在。实例会经历运行（Running）及结束（Terminated）两个阶段。运行阶段的状态为Running（运行中），而结束阶段的状态将会是Success（成功）、Failed（失败）或Canceled（被取消）。用户可以根据运行任务时MaxCompute给出的实例ID查询、改变任务的状态，示例如下：

  ```
  status <instance_id>;
  --查看某实例的状态
  kill <instance_id>;
  --停止某实例，将其状态设置为Canceled
  ```

- **资源配额**

  配额（Quota）分为存储和计算两种。对于存储，在MaxCompute中可以设置一个project中允许使用的存储上限，在接近上限到一定程度时会触发报警。对于计算资源的限制，有内存和CPU两方面，即在project 中同时运行的进程所占用的内存和CPU资源不可以超过指定的上限。

26.4.2 常用命令

26.4.2.1 常用命令简介

MaxCompute提供了对项目空间、表、资源及实例等对象的一系列操作。用户可以通过客户端命令及SDK来操作这些对象。

为了帮助用户快速了解MaxCompute，本章节将详细介绍如何通过客户端使用这些命令。

说明:

- 如果想了解如何安装、配置客户端请参考：配置客户端。
- 对于SDK的更多介绍请参考：*SDK介绍*。

26.4.2.2 项目空间操作

- **进入项目空间**

  命令格式如下：

  ```
  use <project_name>;
  ```

  用途：进入指定项目空间。进入该空间后可以直接操作该项目空间下的所有对象。

  说明: 项目空间不存在或当前用户不在此项目空间中时，返回异常并退出。

  示例如下：

  ```
  odps@ myproject> use my_project;
  -- my_project是用户有权限访问的一个project。
  ```

  说明:

  - 上述示例是在客户端中运行此命令时的示例。
  - 所有的MaxCompute命令关键字、项目空间名、表名、列名均对大小写不敏感。
  - 在命令运行后，用户可以直接访问该项目空间下的对象。例如：

    假设my_project项目空间下有表test_src，用户运行如下命令：

    ```
    odps@ myproject>select * from test_src;
    ```

    MaxCompute会自动搜索项目空间my_project下的表。如果存在此表，返回表中的数据，如果不存在此表，则返回异常并退出。

如果用户在my_project下想要访问另一项目空间my_project2下的表test_src，则需要指定项目空间名，如下所示：

```
odps@ myproject>select * from my_project2.test_src;
```

此时返回my_project2项目空间下的数据结果，而不是my_project下的test_src表数据。

MaxCompute没有提供创建及删除项目空间的命令，用户可以通过管理控制台对各自的项目空间完成更多的配置及操作。

- **查询项目空间**

  命令格式如下：

  ```
  list projects;
  ```

  用途：查看MaxCompute中的所有project。

- **回收项目空间垃圾**

  查看空间垃圾的命令格式如下：

  ```
  show recyclebin;
  ```

  用途：列出本project中的对象明细。

  说明: 只有Project Owner有权限执行此命令。

  清空回收站中所有项目的命令格式如下：

  ```
  purge all;
  ```

  用途：清空本project下回收站中所有对象，释放存储空间。

  说明: 只有Project Owner有权限执行此命令。

  回收某张表的命令格式如下：

  ```
  purge table tblname;
  ```

  用途：清空指定表名的对象在回收站中的记录，释放存储空间。

  说明:

- 当指定名称的表存在时，Project Owner或者对表有写权限的用户有权限执行此命令。
- 当表已被drop时，只有Project Owner有权限执行此命令。

26.4.2.3 表操作

- **Create Table**

  命令格式如下：

```
CREATE TABLE [IF NOT EXISTS] table_name
[(col_name data_type [COMMENT col_comment], ...)] [COMMENT table_comment]
[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)] [LIFECYCLE days]
[AS select_statement];
CREATE TABLE [IF NOT EXISTS] table_name
LIKE existing_table_name;
```

  用途：创建一张表。

  示例如下：

```
CREATE TABLE IF NOT EXISTS sale_detail( shop_name STRING,
customer_id STRING, total_price DOUBLE)
PARTITIONED BY (sale_date STRING,region STRING);
-- 如果没有同名表存在，创建一张分区表sale_detail
```

  说明:

  - 表名与列名均对大小写不敏感。
  - 表名，列名中不能有特殊字符，只能使用英文的a-z，A-Z及数字和下划线’_’，且以字母开头，名称的长度不超过128字节，否则报错。
  - 注释内容是长度不超过1024 字节的有效字符串，否则报错。
  - 对于该命令更详细的介绍请参考：创建表。

- **Drop Table**

  命令格式如下：

```
DROP TABLE [IF EXISTS] table_name;
```

  用途：删除一张表。

  说明: 如果不指定IF EXISTS选项而表不存在，则返回异常；若指定此选项，无论表是否存在，皆返回成功。

  参数说明：

table_name：要删除的表名。

示例如下：

```
DROP TABLE sale_detail;
-- 若表存在，成功返回。
DROP TABLE IF EXISTS sale_detail;
-- 无论是否存在sale_detail表，均成功返回。
```

- **Describe Table**

  命令格式如下：

  ```
  DESC <table_name>;
  ```

  用途：返回指定表的信息，包括Owner（表的属主）、Project（表所属的项目空间）、CreateTime（创建时间）、LastDDLTime（最后一次DDL操作时间）、LastModifiedTime（表中的数据最后一次被改动的时间）、InternalTable（表示被描述的对象是表，总是显示 YES）、Size（表数据所占存储容量的大小，单位Byte）、Native Columns（非分区列的信息，包括：列名，类型，备注）以及Partition Columns （分区列信息，包括：分区名，类型，备注）。

  参数说明：

  table_name：表名或视图名称。

  示例如下：

  ```
  odps@ project_name>DESC sale_detail;
  -- 描述一张分区表
  +------------------------------------------------------------------------------------+
  | Owner: ALIYUN$odpsuser@aliyun.com | Project: test_project |
  |TableComment: |
  +------------------------------------------------------------------------------------+
  |CreateTime: 2014-01-01 17:32:13 |
  |LastDDLTime: 2014-01-01 17:57:38 |
  |LastModifiedTime: 1970-01-01 08:00:00 |
  +------------------------------------------------------------------------------------+
  |InternalTable: YES | Size: 0 |
  +------------------------------------------------------------------------------------+
  |Native Columns: |
  +------------------------------------------------------------------------------------+
  |Field | Type | Comment |
  +------------------------------------------------------------------------------------+
  |shop_name | string | |
  |customer_id | string | |
  |total_price | double | |
  +------------------------------------------------------------------------------------+
  |Partition Columns: |
  +------------------------------------------------------------------------------------+
  |sale_date | string | |
  |region | string | |
  ```

```
+------------------------------------------------------------------------------+
```

说明:

- 上述示例是在客户端中运行此命令时的示例。
- 如果是不带分区的表，将不会显示Partition Columns相关信息。
- 如果描述的是一个视图（View），将不显示InternalTable选项，而是显示VirtualView选项，其值总是为YES。与此类似的，Size选项将会被ViewText选项替代，表示View的定义，例如：select * > from src。关于视图的介绍请参考：创建视图。

- **Show Tables**

  命令格式如下：

  ```
  SHOW TABLES;
  ```

  用途：列出当前项目空间下所有的表。

  示例如下：

  ```
  odps@ project_name>show tables;
  ALIYUN$odps_user@aliyun.com:table_name
  ......
  ```

  说明:

  - 上述示例是在客户端中运行此命令时的示例。
  - odps_user@aliyun.com是用户名，表示该表的创建者。
  - table_name是表名。

- **Show Partitions**

  命令格式如下：

  ```
  SHOW PARTITIONS <table_name>;
  ```

  用途：列出一张表的所有分区。

  参数说明：

  table_name：指定查询的表名称，表不存在或为非分区表时报错。

  示例如下：

  ```
  SHOW PARTITIONS table_name;
  partition_col1=col1_value1/partition_col2=col2_value1
  ```

```
partition_col1=col1_value2/partition_col2=col2_value2
```

说明:

- 上述示例是在客户端中运行此命令时的示例。
- partition_col1和partition_col2表示该表的分区列。
- col1_value1，col2_value1，col1_value2，col2_value2表示分区列对应的值。

26.4.2.4 实例操作

- **Show Instances**

  命令格式如下：

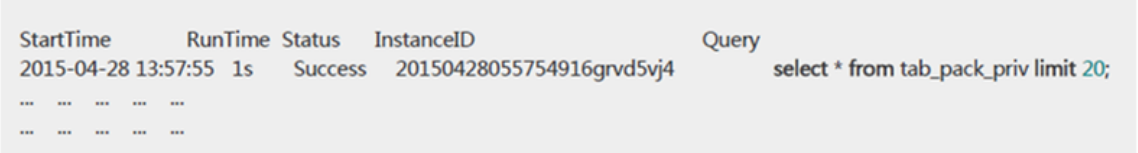
  ```
  SHOW INSTANCES [FROM startdate TO enddate] [number]
  ```

  用途：返回由当前用户创建的实例信息。

  参数说明：

  - startdate To enddate：返回指定时间段内的实例，即从起始时间startdate到结束时间enddate的实例信息。需满足如下格式：yyyy-mm-dd，精度到天。可选参数，若不指定，返回用户三天内提交的实例。
  - number：指定返回实例的数量。依照时间排序，返回距离当前时间最近的number个实例信息。若不指定number，返回满足要求的所有实例信息。
  - 输出项包括：StartTime（时间精确到秒）、RunTime（s）、Status（实例状态，包括Waiting、Success、 Failed、Running、Cancelled、 Suspended）、InstanceID以及实例对应的SQL，如下图所示：

    图 774: 输出项截图

  Instance的状态有如下6种：

  - Running：正在运行。
  - Success：成功结束。

- Waiting：等待中。
- Failed：作业失败，但是尚未改写目标表数据。
- Suspended：挂起。
- Cancelled：被中止。

- **Status Instance**

  命令格式如下：

  ```
  STATUS <instance_id>;
  ```

  用途：返回指定实例的状态，状态包括Success、Failed、 Running、 Cancelled。

  > **说明:** 如果此实例并非当前用户创建，返回异常。

  参数说明：

  instance_id：实例的唯一标识符。指定查询哪个实例状态。

  示例如下：

  ```
  status 20131225123302267gk3u6k4y2; Success
  -- 查看ID为20131225123302267gk3u6k4y2的实例的状态，查询结果为Success。
  ```

  > **说明:** 上述示例是在客户端中运行此命令时的示例。

- **Kill Instance**

  命令格式如下：

  ```
  kill <instance_id>;
  ```

  用途：停止用户指定的实例，此实例的状态必须为Running。需要注意，这是一个异常的过程，在系统接受此请求并返回后，并不意味着分布式的作业已经停止，而是只代表系统已接收到请求，因此还需要使用status命令查看此instance的状态后才可以确定。

  参数说明：

  instance_id，实例的唯一标识符。必须是状态为Running的实例的ID，否则报错。

  示例如下：

  ```
  kill 20131225123302267gk3u6k4y2;
  ```

```
-- 停止ID为20131225123302267gk3u6k4y2的实例。
```

说明: 上述示例是在客户端中运行此命令时的示例。

- **Desc Instance**

  命令格式如下：

  ```
  desc instance <instance_id>;
  ```

  用途：根据具体的实例ID获得作业信息，包括具体的sql、owner、startime、endtime、status等信息。

  参数说明：

  instance_id：实例的唯一标识符。

  示例如下：

  ```
  desc instance 20150715103441522gond1qa2;
  ID 20150715103441522gond1qa2
  Owner ALIYUN$maojing.mj@alibaba-inc.com
  StartTime 2015-07-15 18:34:41
  EndTime 2015-07-15 18:34:42
  Status Terminated
  console_select_query_task_1436956481295 Success Query select * from mj_test;
  -- 查询ID为20150715103441522gond1qa2的实例对应的作业信息。
  ```

  说明: 上述示例是在客户端中运行此命令时的示例。

26.4.2.5 资源操作

- **添加资源**

  命令格式如下：

  ```
  add file <local_file> [as alias] [comment 'cmt'][-f];
  add archive <local_file> [as alias] [comment 'cmt'][-f];
  add table <table_name> [partition <(spec)>] [as alias] [comment 'cmt'][-f]; add jar <local_file.jar> [comment 'cmt'][-f];
  add py <local_file.py> [comment 'cmt'][-f];
  ```

  参数说明：

  - file/archive/table/jar/py：表明资源类型。资源类型的介绍请参考基本概念章节中的资源。
  - local_file：表示本地文件所在路径。并以此文件名作为该资源名，资源名是资源的唯一标识。

- table_name：表示MaxCompute中的表名。
- [PARTITION (spec)]：当添加的资源为分区表时，MaxCompute仅支持将某个分区作为资源，不支持将整张分区表作为资源。
- alias：指定资源名，不添加该参数时默认文件名为资源名。Jar及Py类型资源不支持此功能。
- [comment ‘cmt’]：给资源添加注释。
- [-f]：当存在同名的资源时，此操作会覆盖原有资源。若不指定此选项，存在同名资源时，操作将失败。

示例如下：

```
odps@ odps_public_dev>add table sale_detail partition (ds='20150602') as sale.res comment
 'sale detail on 201506 02' -f;
OK: Resource 'sale.res' have been updated.
-- 添加一个别名为sale.res的表资源到MaxCompute。
```

说明: 每个资源文件的大小不能超过64M，单个SQL， MapReduce任务所引用的资源总大小不能超过512M。

- **删除资源**

命令格式如下：

```
DROP RESOURCE <resource_name>;
```

参数说明：

resource_name：创建资源时指定的资源名。

- **查看资源列表**

命令格式如下：

```
LIST RESOURCES;
```

用途：查看当前项目空间下所有的资源。

示例如下：

```
odps@ $project_name>list resources;

Resource Name     Comment     Last Modified Time      Type
1234.txt                  2014-02-27 07:07:56     file
mapred.jar                2014-02-27 07:07:57     jar
```

26.4.2.6 函数操作

- **注册函数**

  命令格式如下：

  ```
  CREATE  FUNCTION <function_name> AS <package_to_class> USING<resource_list>;
  ```

  参数说明：

  - function_name：UDF函数名。这个名字就是SQL中引用该函数所使用的名字。
  - package_to_class：如果是java UDF，这个名字就是从顶层包名一直到实现UDF类名的fully qualified class name。如果是python UDF，这个名字就是python脚本名.类名。这个名字必须用引号引起来。
  - resource_list：UDF所用到的资源列表，这个里面必须包括UDF代码所在的资源。如果用户代码中通过distributedcache接口读取资源文件，这个列表中还得包括UDF所读取的资源文件列表。资源列表由多个资源名组成，资源名之间由逗号（”,” ）分隔。资源列表必须用引号引起来。

  示例如下：

  假设Java UDF类org.alidata.odps.udf.examples.Lower在my_lower.jar中，创建函数my_lower：

  ```
  CREATE FUNCTION test_lower AS 'org.alidata.odps.udf.examples.Lower'
  USING 'my_lower.jar';
  ```

  假设Python UDF MyLower在脚本pyudf_test.py中，创建函数my_lower：

  ```
  create function my_lower as 'pyudf_test.MyLower';using 'pyudf_test.py';
  ```

  说明:

  - 与资源文件一样，同名函数只能注册一次。
  - 一般情况下用户自建函数无法覆盖系统内建函数，只有项目空间的Owner才有权利覆盖内建函数。如果用户使用了覆盖内建函数的自定义函数，在SQL执行结束后，会在Summary中打印出warning信息。

- **注销函数**

命令格式如下：

```
DROP FUNCTION  <function_name>;
```

示例如下：

```
DROP FUNCTION test_lower;
```

26.4.2.7 Tunnel操作

- **功能简介**
  - Upload：支持文件或目录（指一级目录）的上传，每次上传只支持数据上传到一个表或表的一个分区，有分区的表一定要指定上传的分区。

    ```
    tunnel upload log.txt test_project.test_table/p1="b1",p2="b2";
    tunnel upload log.txt test_table --scan=only;
    ```

  - Download：只支持下载到单个文件，每次下载只支持下载一个表或分区到一个文件，有分区的表一定要指定下载的分区。

    ```
    tunnel download test_project.test_table/p1="b1",p2="b2" log.txt;
    ```

  - Resume：因为网络或tunnel服务的原因出错，dship支持文件或目录的续传。

    ```
    tunnel resume;
    ```

  - Show：显示历史任务信息。

    ```
    tunnel show history -n 5
    tunnel show log
    ```

  - Purge：清理session目录，默认清理3天内的。

    ```
    tunnel purge 5
    ```

- **Tunnel命令使用说明**

  通过help子命令获取帮助信息，每个命令和选择支持短命令格式如下：

```
odps@ project_name>tunnel help;
Usage: tunnel <subcommand> [options] [args]
Type 'tunnel help <subcommand>' for help on a specific subcommand.

Available subcommands:
 upload (u)
 download (d)
 resume (r)
 show (s)
 purge (p)
 help (h)
```

```
tunnel is a command for uploading data to / downloading data from ODPS.
```

参数说明：

- upload：帮助用户上传数据到MaxCompute的表中。
- download：帮助用户从MaxCompute的表中下载数据。
- resume：如果上传数据失败，通过resume命令进行断点续传，目前仅支持上传数据的续传。

  每次上传、下载数据被称为一个session。在resume命令后指定session id完成续传。
- show：查看历史运行信息。
- purge：清理session目录。
- help：输出tunnel帮助信息。

- **Upload**

  将本地文件的数据导入MaxCompute的表中，以追加模式导入。子命令的命令格式如下：

```
usage: tunnel upload [options] <path> <[project.]table[/partition]>
            upload data from local file
 -bs,-block-size <ARG>             block size in MiB, default 100
 -c,-charset <ARG>                 specify file charset, default ignore.
                                   set ignore to download raw data
 -cp,-compress <ARG>               compress, default true
 -dbr,-discard-bad-records <ARG>   specify discard bad records
                                   action(true|false), default false
 -dfp,-date-format-pattern <ARG>   specify date format pattern, default
                                   yyyy-MM-dd HH:mm:ss
 -fd,-field-delimiter <ARG>        specify field delimiter, support
                                   unicode, eg \u0001. default ","
 -h,-header <ARG>                  if local file should have table header,
                                   default false
 -mbr,-max-bad-records <ARG>       max bad records, default 1000
 -ni,-null-indicator <ARG>         specify null indicator string, default
                                   ""(empty string)
 -rd,-record-delimiter <ARG>       specify record delimiter, support
                                   unicode, eg \u0001. default "\n"
 -s,-scan <ARG>                    specify scan file
                                   action(true|false|only), default true
 -sd,-session-dir <ARG>            set session dir, default /D:/console/plugins/dship/
 -te,-tunnel_endpoint <ARG>        tunnel endpoint
   -threads <ARG>                  number of threads, default 1
 -tz,-time-zone <ARG>              time zone, default local timezone:
                                   Asia/Shanghai
Example:
   tunnel upload log.txt test_project.test_table/p1="b1",p2="b2"
```

参数说明：

- -bs，block-size：每次上传至Tunnel的数据块大小。默认值：100MiB (MiB = 1024*1024B)。
- -c，-charset：指定本地数据文件编码。默认为UTF-8，不设定时，默认下载源数据。
- -cp，-compress：指定是否在本地压缩后再上传，减少网络流量，默认开启。

- -dbr：是否忽略脏数据（多列、少列、列数据类型不匹配等情况）。值为true时，将全部不符合表定义的数据忽略。值为false，若遇到脏数据，则给出错误提示信息，目标表内的原始数据不会被污染。
- -dfp：DateTime类型数据格式。默认为yyyy-MM-dd HH:mm:ss。
- -fd：本地数据文件的列分割符，默认为逗号。
- -h：数据文件是否包括表头。如果为true，则dship会跳过表头从第二行开始上传数据。
- -mbr，-max-bad-records：默认情况下，当上传的脏数据超过1000条时，上传动作终止。通过此参数，可以调整可容忍的脏数据量。
- -ni：NULL数据标志符，默认为" "（空字符串）。
- -rd：本地数据文件的行分割符，linux默认为 ‘\n’ ，windows默认为 ‘\r\n’ 。
- -s：是否扫描本地数据文件，默认值为false 。值为true时，先扫描数据，若数据格式正确，再导入数据。值为false，不扫描数据，直接进行数据导入。值为only时，仅进行扫描本地数据，扫描结束后不继续导入数据。
- -sd，-session-dir：session目录所在的路径，默认为*/D:/console/plugins/dship/lib/..*（plugins/dship/lib所在具体路径）。
- -te：指定tunnel的Endpoint。
- -tz：指定时区。默认为本地时区：Asia/Shanghai。

示例如下：

创建目标表：

```
CREATE TABLE IF NOT EXISTS sale_detail(
 shop_name STRING,
 customer_id STRING,
  total_price DOUBLE)
PARTITIONED BY (sale_date STRING,region STRING);
```

添加分区：

```
alter table sale_detail add partition (sale_date='201312', region='hangzhou');
```

准备数据文件data.txt，其内容为：

```
shop9,97,100
shop10,10,200
```

```
shop11,11
```

这份文件的第三行数据与sale_detail的表定义不符。sale_detail定义了三列，但数据只有两列，此时导入数据。

```
odps@ project_name>tunnel u d:\data.txt  sale_detail/sale_date=201312,region=hangzhou -s false Upload session:  20150610163922488 0870a002ec60c
Start upload:d:\data.txt
Total bytes:41 Split input to 1 blocks
2015-06-10 16:39:22 upload block: '1'
ERROR: column mismatch -,expected 3 columns, 2 columns found, please check  data or  delimiter
```

由于data.txt 中有脏数据，数据导入失败。并给出session id及错误提示信息。数据验证如下：

```
odps@ odpstest_ay52c_ay52> select * from sale_detail where sale_date='201312'; ID = 20150610084135370gyvc61z5
+-----------+------------+------------+-----------+--------+
shop_name | customer_id | total_price | sale_date | region |
+-----------+------------+------------+-----------+--------+
+-----------+------------+------------+-----------+--------+
```

- **Show**

  显示历史记录。子命令的命令格式如下：

  ```
  usage: tunnel show history [options]
  ```

  示例如下：

  ```
  odps@ project_name>tunnel show history;
   usage: tunnel show history [options]
              show session information
   -n,-number <ARG>   lines
  Example:
     tunnel show history -n 5
     tunnel show log
  ```

- **Resume**

  修复执行历史记录，仅对上传数据有效。子命令的命令格式如下：

  ```
  usage: tunnel resume [session_id] [--force]
              resume an upload session
   -f,-force   force resume
  Example:
     tunnel resume
  ```

  示例如下：

  修改data.txt文件为：

  ```
  shop9,97,100
  ```

```
shop10,10,200
```

修复执行上传数据：

```
odps@ project_name>tunnel resume 20150610163922488 0a002ec60c --  force;
start resume
20150610163922488 0a002ec60c
Upload session: 20150610163922488 0a002ec60c
Start upload:d:\data.txt
Resume 1 blocks
2015-06-10 16:46:42 upload block: '1'
2015-06-10 16:46:42 upload block complete, blockid=1
upload complete, average > speed is 0 KB/s
OK
```

其中，20150610163922488 0a002ec60c为上传失败的session id。数据验证如下：

```
odps@ project_name>select * from sale_detail where sale_date='201312';
ID = 20150610084801405g0a741z5
+-----------+-------------+-------------+-----------+--------+
shop_name | customer_id | total_price | sale_date | region |
+-----------+-------------+-------------+-----------+--------+
+-----------+-------------+-------------+-----------+--------+
```

- **Download**

  子命令的命令格式如下：

```
odps@ project_name>tunnel help download
usage: tunnel download [options] <[project.]table[/partition]> <path>
              download data to local file
 -c,-charset <ARG>                  specify file charset, default ignore.
                                    set ignore to download raw data
 -cp,-compress <ARG>                compress, default true
 -dfp,-date-format-pattern <ARG>    specify date format pattern, default
                                    yyyy-MM-dd HH:mm:ss
 -e,-exponential <ARG>              When download double values, use
                                    exponential express if necessary.
                                    Otherwise at most 20 digits will be
                                    reserved. Default false
 -fd,-field-delimiter <ARG>         specify field delimiter, support
                                    unicode, eg \u0001. default ","
 -h,-header <ARG>                   if local file should have table header,
                                    default false
    -limit <ARG>                    specify the number of records to
                                    download
 -ni,-null-indicator <ARG>          specify null indicator string, default
                                    ""(empty string)
 -rd,-record-delimiter <ARG>        specify record delimiter, support
                                    unicode, eg \u0001. default "\n"
 -sd,-session-dir <ARG>             set session dir, defa            /D:/console/plugins/dship/
-te,-tunnel_endpoint <ARG>          tunnel endpoint
    -threads <ARG>                  number of threads, default 1
 -tz,-time-zone <ARG>               time zone, default local timezone:Asia/Shanghai
Example:
    tunnel download test_project.test_table/p1="b1",p2="b2" log.txt
```

参数说明：

- -fd：本地数据文件的列分割符，默认为逗号 ‘，’ 。
- -rd：本地数据文件的行分割符，默认为 ‘\r\n’ 。
- -dfp：DateTime类型数据格式，默认为 ‘yyyy-MM-dd HH:mm:ss’ 。
- -ni：NULL数据标志符，默认为" "（空字符串）。
- -c：本地数据文件编码，默认为 ‘UTF-8’ 。

示例如下：

下载数据到result.txt文件中：

```
$ ./tunnel download sale_detail/sale_date=201312,region=hangzhou result.txt; Download
 session: 20150610165824528387 0a002ed0b9
Total records: 2
2015-06-10 16:58:24 download records: 2
2015-06-10 16:58:24 file size: 30 bytes
OK
```

数据验证如下，result.txt的文件内容为：

```
shop9,97,100.0
shop10,10,200.0
```

- **Purge**

清除session目录，默认清除距离当前日期3天内的。子命令的命令格式如下：

```
usage: tunnel purge [n]
            force session history to be purged.([n] days before, default
            3 days)

Example:
   tunnel purge 5
```

26.4.2.8 其他操作

- **ALIAS命令**

Alias功能主要为了满足在不修改代码的前提下，在MaxCompute的M/R参考手册或自定义函数（UDF）代码中，通过某个固定的资源名读取不同资源（数据）的需求。

命令格式如下：

```
ALIAS <alias>=<real>;
```

用途：为资源创建别名。

示例如下：

```
ADD TABLE src_part PARTITION (ds='20121208') AS res_20121208; ADD TABLE src_part
 PARTITION (ds='20121209') AS res_20121209;
ALIAS resName=res_20121208;
jar -resources resName -libjars work.jar -classpath ./work.jar com.company.MainClass
 args ...; // 作业一 ALIAS resName=res_20121209;
jar -resources resName -libjars work.jar -classpath ./work.jar com.company.MainClass
 args ...; // 作业二
```



说明： 上面的资源别名resName在两个作业里引用到不同的资源表，代码可以不做修改也能读取到不同的数据。

- **Set**

  命令格式如下：

  ```
  set ["<KEY>=<VALUE>"]
  ```

  用途：用户可以使用set命令设置MaxCompute或用户自定义的系统变量影响MaxCompute的行为。

  目前，MaxCompute支持的系统变量包括：

  ```
  --MaxCompute SQL及新版本Mapreduce支持的Set命令
  set odps.stage.mapper.mem= --设置每个map worker的内存大小，单位是M，默认值1024M
  set odps.stage.reducer.mem= --设置每个reduce worker的内存大小，单位是M，默认值1024M
  set odps.stage.joiner.mem= --设置每个join worker的内存大小，单位是M，默认值1024M
  set odps.stage.mem =--设置ODPS指定任务下所有worker的内存大小。优先级低于以上三个set key，单位M，无默认值
  set odps.stage.mapper.split.size=
  -- 修改每个map worker的输入数据量，即输入文件的分片大小，
  -- 从而间接控制每个map阶段下worker的数量，单位M，默认值256M
  set odps.stage.reducer.num= --修改每个reduce阶段worker数量，无默认值。
  set odps.stage.joiner.num= --修改每个join阶段worker数量，无默认值。
  set odps.stage.num= --修改ODPS指定任务的所有阶段的worker的并发度，优先级低于以上三者，无默认值。
  --MaxCompute老版本MapReduce支持的Set命令
  set odps.mapred.map.memory= --设置每个map worker的内存大小，单位是M，默认值1024M
  set odps.mapred.reduce.memory= --设置每个reduce worker的内存大小，单位是M，默认值1024M
  setodps.mapred.map.split.size=
  -- 修改每个map worker的输入数据量，即输入文件的分片大小，
  -- 从而间接控制每个map阶段下worker的数量，单位M，默认值256M
  set odps.mapred.reduce.tasks= --修改每个reduce阶段worker数量，无默认值。
  ```

- **SetProject**

命令格式如下：

```
setproject ["<KEY>=<VALUE>"];
```

用途：用户可以使用setproject命令设置Project属性，当不指定< KEY >=< VALUE >时，显示当前Project属性配置。

Project 属性的详细说明如下：

属性名称	设置权限	属性描述	取值范围
odps.table.drop.ignoreno...	所有用户	当删除不存在的表时，是否报错，true时不报错	true(不报错)/false
odps.instance.priority.au...	ProjectOwner	是否自动调整任务优先级，让小任务优先调	true(开启)/false
odps.instance.priority.level	ProjectOwner	调整Project内任务的优先级	1(优先级高) ~ 3
odps.security.ip.whitelist	ProjectOwner	指定访问Project的IP白名单	ip列表，逗号分隔
odps.table.lifecycle	ProjectOwner	optional：创建表时，lifecycle子句为可选，如果用户不设置生命周期，则此表永久有效 mandatory：lifecycle子句为必选 inherit：如果用户不指定生命周期，该表的生命周期为odps.table.lifecycle.value的值	optional /mandatory/ inherit
odps.table.lifecycle.value	ProjectOwner	默认的生命周期值	1 ~ 37231(默认)
odps.instance.remain.days	ProjectOwner	Instance信息保留时间	3 ~ 30
odps.task.sql.outerjoin.pp...	ProjectOwner	在full outer join中过滤条件是否下推	true/false
odps.function.strictmode	ProjectOwner	内置函数遇到脏数据时，返	true/false

		回NULL(false)，或者报错(true)	
odps.task.sql.write.str2null	ObjectOwner	是否将空字符串视为NULL(true)	true/false

- **导出project meta**

  命令格式如下：

  ```
  export <projectname> <local_path>;
  ```

  用途：导出project的meta到本地文件。Meta以odpscmd接受的语句表示，利用该meta文件可以重建一个project。

- **Show Flags**

  命令格式如下：

  ```
  show flags;
  ```

  用途：显示Set设置的参数。

  说明: 运行Use Project命令会清除掉set命令设置的配置。

26.5 SQL使用指南

26.5.1 概要

26.5.1.1 使用场景

MaxCompute SQL离线计算适用于海量数据（TB级别），实时性要求不高的场合。它的每个作业的准备，提交等阶段要花费较长时间，因此要求每秒处理几千至数万笔事务的业务是不能用MaxCompute SQL完成的。准实时场景可以使用MaxCompute SQL的在线模式处理。

MaxCompute SQL采用的是类似于SQL的语法，可以看作是标准SQL的子集，但不能因此简单的把MaxCompute SQL等价成一个数据库，它在很多方面并不具备数据库的特征，如事务、主键约束、索引等。目前在MaxCompute中允许的最大SQL长度是2MB。

26.5.1.2 保留字

MaxCompute将SQL语句的关键字作为保留字。在对表、列或是分区命名时请不要使用，否则会报错。保留字不区分大小写。

下面只给出常用的保留字列表，完整的保留字列表请参见：*MaxCompute SQL 保留字*。

```
% & && ( ) * + - . / ; < <= <>
= > >= ? ADD ALL ALTER
AND AS ASC BETWEEN BIGINT BOOLEAN BY
CASE CAST COLUMN COMMENT CREATE DESC DISTINCT
DISTRIBUTE DOUBLE DROP ELSE FALSE FROM FULL
GROUP IF IN INSERT INTO IS JOIN
LEFT LIFECYCLE LIKE LIMIT MAPJOIN NOT NULL
ON OR ORDER OUTER OVERWRITE PARTITION RENAME
REPLACE RIGHT RLIKE SELECT SORT STRING TABLE
THEN TOUCH TRUE UNION VIEW WHEN WHERE
```

26.5.1.3 分区表

指定分区列会给用户带来诸多便利，例如：提高SQL运行效率，减少计费等。在如下场景下使用分区表将会带来较大的收益：在select语句的where条件过滤中使用分区列作为过滤条件。与此同时，部分对分区操作的SQL的运行效率则较低，例如：动态分区生成的数据量过多（大于 2048）且执行集中到一个MaxCompute的instance中时会执行失败，多级分区时会容易忽略分区数目过大的问题，在有大量分区生成需要根据原数据的信息做初步评估。

目前用户最多可创建 6 级分区。对于部分MaxCompute操作命令，处理分区表和非分区表时的语法有差别，详情请参考： *DDL 语句*及 *DML 语句*部分的说明。

建表语句请参考：创建表。

26.5.1.4 类型转换

MaxCompute SQL允许数据类型之间的转换，类型转换方式包括：显式类型转换和隐式类型转换。

26.5.1.4.1 显式类型转换

显式类型转换是用cast将一种数据类型的值转换为另一种类型的值的行为，在MaxCompute SQL中支持的显式类型转换如下：

From/To	Bigint	Double	String	Datetime	Boolean	Decimal
Bigint	–	Y	Y	N	N	Y
Double	Y	–	Y	N	N	Y
String	Y	Y	–	Y	N	Y
Datetime	N	N	Y	–	N	N
Boolean	N	N	N	N	–	N
Decimal	Y	Y	Y	N	N	-

其中，Y表示可以转换，N表示不可以转换，–表示不需要转换。

说明:

- 将double类型转换为bigint类型时，小数部分会被截断，例如：cast(1.6 as bigint) = 1。
- 满足double类型格式的string类型转换为bigint时，会先将string类型转换为double类型，再将double类型转换为bigint类型，因此，小数部分会被截断，例如：cast(“1.6” as bigint) = 1。
- 满足bigint类型格式的string类型可以被转换为double类型，小数点后保留一位，例如：cast(“1” as double) = 1.0。
- 常量字符串转为decimal类型时，需要加上引号，不加引号数值会当做double类型处理，例如：cast(“1.234567890123456789” as decimal)；
- 不支持的显式类型转换会导致异常。
- 如果在执行时转换失败，报错退出。
- 日期类型转换时采用默认格式**yyyy-mm-dd hh:mi:ss**，详细说明信息请参考：*String 类型与 Datetime 类型之间的转换*。
- 部分类型之间不可以通过显式的类型转换，但可以通过SQL内建函数进行转换，例如：从boolean类型转换到string类型，可使用函数to_char，详细介绍请参考：*TO_CHAR* ，而to_date函数同样支持从string类型到datetime类型的转换，详细介绍请参考： *TO_DATE* 。
- 关于cast的介绍请参考： *CAST*。
- Decimal类型超出值域，cast string to > decimal可能会出现最高位溢出报错，最低位溢出截断等情况。

26.5.1.4.2 隐式类型转换及其作用域

隐式类型转换是指在运行时，由MaxCompute依据上下文使用环境及类型转换规则自动进行的类型转换。MaxCompute支持的隐式类型转换规则如下所示：

From/To	Boolean	Tinyint	Smallint	Int	Bigint	Float	Double	Decimal	String	Varchar	Timestamp	Binary
Boolean	T	F	F	F	F	F	F	F	F	F	F	F
Tinyint	F	T	T	T	T	T	T	T	T	T	F	F
Smallint	F	F	T	T	T	T	T	T	T	T	F	F

Int	F	F	F	T	T	T	T	T	T	T	F	F
Bigint	F	F	F	F	T	T	T	T	T	T	F	F
Float	F	F	F	F	F	T	T	T	T	T	F	F
Double	F	F	F	F	F	F	T	T	T	T	F	F
Decimal	F	F	F	F	F	F	F	T	T	T	F	F
String	F	F	F	F	F	F	T	T	T	T	F	F
Varchar	F	F	F	F	F	F	T	T	T	T	F	F
Timestamp	F	F	F	F	F	F	F	F	T	T	T	F
Binary	F	F	F	F	F	F	F	F	F	F	F	T

其中，T表示可以转换，F表示不可以转换。

说明:

- 不支持的隐式类型转换会导致异常。
- 如果在执行时转换失败，报错退出。
- 由于隐式类型转换是MaxCompute依据上下文使用环境自动进行的类型转换，因此，推荐在类型不匹配时显式的用cast进行转换。
- 隐式类型转换规则是有发生作用域的。在某些作用域中，只有一部分规则可以生效。详细信息请参考隐式类型转换的作用域。

关系运算符作用下的隐式转换

关系运算符包括：=， <>， <， <=， >， >=， IS NULL， IS NOT NULL， LIKE，RLIKE和IN。由于LIKE， RLIKE 和IN的隐式类型转换规则不同于其他关系运算符，将单独拿出章节对这三种关系运算符做出说明。此处的说明不包含这三种特殊的关系运算符。当不同类型的数据共同参与关系运算时，按照下述原则进行隐式类型转换。

From/To	Bigint	Double	String	Datetime	Boolean	Decimal
Bigint	–	Double	Double	N	N	Decimal
Double	Double	–	Double	N	N	Decimal
String	Double	Double	–	Datetime	N	Decimal
Datetime	N	N	Datetime	–	N	N
Boolean	N	N	N	N	–	N
Decimal	Decimal	Decimal	Decimal	N	N	–

说明:

- 如果需要比较的两个类型间不能进行隐式类型转换，则该关系运算不能完成，报错退出。
- 关系运算符介绍，请参考：关系操作符。

特殊的关系运算符作用下的隐式转换

特殊的关系运算符包括LIKE，RLIKE，IN。

LIKE及RLIKE的使用方式如下所示：

```
source like pattern;
source rlike pattern;
```

此二者在隐式类型转换中的注意事项如下：

- LIKE和RLIKE的source和pattern参数均仅接受string类型。
- 其他类型不允许参与运算，也不能进行到string类型的隐式类型转换。
- 如果字符串source或者pattern为NULL，则返回NULL。

IN的使用方式如下所示：

```
key in (value1, value2,...)
```

IN的隐式转换规则如下所示：

- IN右侧的value值列表中的数据类型必须一致。
- 当key与values之间比较时，若bigint类型，double类型，string类型之间比较，统一转换为double类型；若datetime类型和string类型之间比较，统一转换为datetime类型。除此之外不允许其他类型之间的转换。

IN的注意事项：

IN如果参数过多，会造成编译时的压力，在5000个参数时，gcc编译会占用到17G的内存。建议限制到1024个参数，参数1024个时，内存峰值占用1G，编译时间39秒。

算术运算符作用下的隐式转换

算术运算符包括：+，-，*，/，%，其隐式转换规则如下所示：

- 只有string类型，bigint类型，decimal类型以及double类型才能参与算术运算。
- String在参与运算前会进行隐式类型转换到double类型。
- Bigint类型和double类型共同参与计算时，会将bigint隐式转换为double类型。

- 日期型和布尔型不允许参与算数运算。

  说明: 算术运算符的更多详情请参考：算术操作符。

逻辑运算符作用下的隐式转换

逻辑运算符包括：and，or和not，其隐式转换规则如下所示：

- 只有boolean类型才能参与逻辑运算。
- 其他类型不允许参与逻辑运算，也不允许其他类型的隐式类型转换。

  说明: 逻辑运算符的更多详情请参考：逻辑操作符。

26.5.1.4.3 SQL内建函数

MaxCompute SQL提供了大量的系统函数，方便用户对任意行的一列或多列进行计算，输出任意种的数据类型。其隐式转换规则如下：

- 在调用函数时，如果输入参数的数据类型与函数定义的参数数据类型不一致，把输入参数的数据类型转换为函数定义的数据类型。
- 每个MaxCompute SQL内建函数的参数对于允许的隐式类型转换的要求不同，详细信息请参考：内建函数部分的说明。

26.5.1.4.4 CASE WHEN

Case when的隐式转换规则如下：

- 如果返回类型只有bigint类型和double类型，统一转换为double类型。
- 如果返回类型中有string类型，统一转换为string类型；如果不能转换，则报错（如boolean类型）。
- 除此之外不允许其他类型之间的转换。

26.5.1.4.5 分区列

MaxCompute SQL支持分区表，对于分区表的定义，请参考： *DDL* 语句及 *DML* 语句部分的说明。目前，MaxCompute分区支持tinyint类型，smallint类型，int类型，bigint类型，varchar类型和string类型。

26.5.1.4.6 UNION ALL

参与UNION ALL运算的所有列的数据类型、列个数、列名称必须完全一致，否则报错。

26.5.1.4.7 String类型与Datetime类型之间的转换

MaxCompute支持string类型和datetime类型之间的相互转换。转换时使用的格式为**yyyy-mm-dd hh:mi:ss.ff3**，其中：

单位	字符串(忽略大小写)	有效值域
年	yyyy	0001 ~ 9999
月	mm	01 ~ 12
日	dd	01 ~ 28,29,30,31
时	hh	00 ~ 23
分	mi	00 ~ 59
秒	ss	00 ~ 59
毫秒	ff3	00-999



说明:

- 各个单位的值域中，如果首位为0，不可省略，例如：2014-1-9 12:12:12 就是非法的datetime格式，无法从这个string类型数据转换为datetime类型， 必须写为2014-01-09 12:12:12。
- 只有符合上述格式描述的string类型才能够转换为datetime类型，例如：cast(“2013-12-31 02:34:34” as datetime) 将会把string类型“2013-12-31 02:34:34”转换为datetime类型。同理，datetime类型转换为string类型时，默认转换为**yyyy-mm-dd hh:mi:ss**的格式。 类似于下面的转换尝试，将会失败导致异常，例如：

```
cast("2013/12/31 02/34/34" as datetime)
cast("20131231023434" as datetime)
```

```
cast("2013-12-31 2:34:34" as datetime)
```

MaxCompute提供了to_date函数，用以将不满足日期格式的string类型数据转换为datetime类型。详细信息请参考：*TO_DATE* 。

26.5.2 运算符

26.5.2.1 关系操作符

操作符	说明
A=B	如果A或B为NULL，返回NULL；如果A等于B，返回TRUE，否则返回FALSE。
A<>B	如果A或B为NULL，返回NULL；如果A不等于B，返回TRUE，否则情况返回FALSE。
A<B	如果A或B为NULL，返回NULL；如果A小于B，返回TRUE，否则返回FALSE。
A<=B	如果A或B为NULL，返回NULL；如果A小于等于B，返回TRUE，否则返回FALSE。
A>B	如果A或B为NULL，返回NULL；如果A大于B，返回TRUE，否则返回FALSE。
A>=B	如果A或B为NULL，返回NULL；如果A大于等于B，返回TRUE，否则返回FALSE。
A IS NULL	如果A为NULL，返回TRUE，否则返回FALSE。
A IS NOT NULL	如果A不为NULL，返回TRUE，否则返回FALSE。
A LIKE B	如果A或B为NULL，返回NULL；A为字符串，B为要匹配的模式，如果匹配成功，返回TRUE，否则返回 FALSE。 “%”匹配任意多个字符， “_”匹配单个字符，要匹配“%”或“_”需要用转义符表示“\%” ， “_”。 ‘aaa’ like ‘a’ = TRUE ‘aaa’ like ‘a%’ = TRUE ‘aaa’ like ‘aab’ = FALSE ‘a%b’ like ‘a\%b’ = TRUE ‘axb’ like ‘a\%b’ = FALSE
A RLIKE B	如果A或B为NULL，返回NULL；A是字符串，B是字符串常量正则表达式， 如果匹配成功，返回TRUE，否则返回FALSE；如果B为空串会报错退出。
A IN B	B是一个集合，如果A为NULL，返回NULL，如A在B中则返回TRUE，否则返回FALSE；若B仅有一个元素NULL，即A IN （NULL），则返回NULL；若B含有NULL元素，将NULL视为B集合中其他元素的类型。 B必须是常数并且至少有一项，所有类型要一致。

由于double值存在一定的精度差，因此，不建议直接使用等号对两个double类型数据进行比较。用户可以使用两个double类型相减，而后取绝对值的方式判断。当绝对值足够小时，认为两个double数值相等，示例如下：

```
abs(0.9999999999 - 1.0000000000) < 0.000000001
-- 0.9999999999和1.0000000000为10位精度，而0.000000001为9位精度。
-- 此时可以认为0.9999999999和1.0000000000相等。
```



说明:

- Abs是MaxCompute提供的内建函数，意为取绝对值，详细介绍可参考： *ABS* 。
- 通常情况下，MaxCompute的double类型能够保障16位有效数字。

26.5.2.2 算术操作符

操作符	说明
A + B	如果A或B为NULL，返回NULL；否则返回A + B的结果。
A–B	如果A或B为NULL，返回NULL；否则返回A–B的结果。
A * B	如果A或B为NULL，返回NULL；否则返回A * B的结果。
A / B	如果A或B为NULL，返回NULL；否则返回A / B的结果。如果A和B为bigint类型，结果为double类型。
A % B	如果A或B为NULL，返回NULL；否则返回A % B的结果。
+A	仍然返回A。
-A	如果A为NULL，返回NULL；否则返回-A。



说明:

- 只有string类型，bigint类型，double类型，decimal类型才能参与算术运算，日期型和布尔型不允许参与运算。
- String类型在参与运算前会进行隐式类型转换到double类型。
- bigint类型和double类型共同参与运算时，会将bigint类型隐式类型转换为double类型再进行运算，返回结果为double类型。

- A和B都是bigint类型，进行A / B运算，返回结果为double类型，进行上述其他运算仍然返回bigint类型。

26.5.2.3 位运算操作符

操作符	说明
A & B	返回A与B进行按位与的结果。例如：1&2返回0，1&3返回1。NULL与任何值按位与都为NULL。 A和B必须为bigint 类型。
A | B	返回A与B进行按位或的结果。例如：1 | 2返回3，1 | 3返回3，NULL与任何值按位或都为NULL。 A和B必须为bigint 类型。



注意: 位运算符只允许bigint类型参与运算，不支持隐式类型转换。

26.5.2.4 逻辑操作符

操作符	说明
A and B	TRUE and TRUE = TRUE
	TRUE and FALSE = FALSE
	FALSE and TRUE = FALSE
	FALSE and NULL = FALSE
	FALSE and FALSE = FALSE
	NULL and FALSE = FALSE
	TRUE and NULL = NULL
	NULL and TRUE = NULL
	NULL and NULL = NULL
A or B	TRUE or TRUE = TRUE
	TRUE or FALSE = TRUE
	FALSE or TRUE = TRUE
	FALSE or NULL = NULL
	NULL or FALSE = NULL
	TRUE or NULL = TRUE
	NULL or TRUE = TRUE

操作符	说明
	NULL or NULL = NULL
NOT A	如果A是NULL，返回NULL。
	如果A是TRUE，返回FALSE。
	如果A是FALSE，返回TRUE。

说明: 逻辑操作符只允许boolean类型参与运算，不支持隐式类型转换。

26.5.3 DDL语句

26.5.3.1 表操作

26.5.3.1.1 创建表（CREATE TABLE）

命令格式如下：

```
create table [if not exists] table_name
[(col_name data_type [comment col_comment], ...)] [comment table_comment]
[partitioned by (col_name data_type [comment col_comment], ...)] [lifecycle days]
[as select_statement]
create table [if not exists] table_name like existing_table_name
```

说明:

- 表名与列名均对大小写不敏感；
- 在创建表时，如果不指定if not exists选项而存在同名表，则返回报错；若指定此选项，则无论是否存在同名表，即使原表结构与要创建的目标表结构不一致， 均返回成功。已存在的同名表的元信息不会被改动；
- 数据类型只能是：bigint，double，boolean，datetime，decimal 及string,Array < T >,Map < T1, T2 > ；
- 表名，列名中不能有特殊字符，只能用英文的 a-z，A-Z及数字和下划线_，且以字母开头，名称的长度不超过128字节；
- Partitioned by指定表的分区字段，目前仅支持string类型。分区值不可以有双字节字符（如中文），必须是以英文字母 a-z、A-Z开始后可跟字母数字， 名称的长度不超过 128 字节。允许的字符包括：空格 ‘ ‘，冒号’ :’ ，下划线’ _’ ，美元符’ $’ ，井号’ #’ ，点’ .’ ，感叹号’ !’ 和’ @’ ，出现其他字符行为未定义，例如：” \t” ，” \n” ，” /” 等。当利用分区字段对表进行分区时，新增分区、更新分区内数据和读取分区数据均不需要做全表扫描，可以提高处理效率；

- 注释内容是长度不超过1024字节的有效字符串；
- lifecycle指明此表的生命周期，单位：天。create table like语句不会复制源表的生命周期属性；
- 理论上源表分区最多只能6级，但考虑极限存储的分区膨胀方式，请尽可能少用分区；
- 一个表允许的分区个数支持按照具体的project配置，默认60000个。

示例如下：

创建表sale_detail保存销售记录，该表使用销售时间（sale_date）和销售区域（region）作为分区列：

```
create table if not exists sale_detail( shop_name string,
customer_id string,
total_price double)
partitioned by (sale_date string,region string);
-- 创建一张分区表sale_detail
```

也可以通过create table ... as select ..语句创建表，并在建表的同时将数据复制到新表中：

```
create table sale_detail_ctas1 as select * from sale_detail;
```

说明: 此时，如果sale_detail中存在数据，上述示例会将sale_detail的数据全部复制到sale_detail_ctas1表中。需要注意，此处的sale_detail是一张分区表，而通过create table ... as select ... 语句创建的表不会复制分区属性，只会把源表的分区列作为目标表的一般列处理，即sale_detail_ctas1是一个含有5列的非分区表。

在create table ... as select ...语句中，如果在select子句中使用常量作为列的值，建议指定列的名字：

```
create table sale_detail_ctas2 as select shop_name,
customer_id, total_price,
'2013' as sale_date,
'China' as region from sale_detail;
```

说明:

如果不加列的别名，则如下示例中创建的表sale_detail_ctas3的第四、五列会是类似“_c3”、“_c4”这样的系统自动生成的名字：

```
create table sale_detail_ctas3 as select shop_name,
customer_id, total_price, '2013',
'China'
```

```
from sale_detail;
```

此时，再次使用表sale_detail_ctas3需要加上如下示例中的反引号才能正确引用，直接执行SQL“select c3, _c4 from sale_detail_ctas3”会报错退出。因为MaxCompute SQL的列名不能够以“”开头，因此必须加反引号“`”做特殊处理。建议用户使用别名，避免出现上述情况。

```
select `_c3`, `_c4` from sale_detail_ctas3;
```

如果希望源表和目标表具有相同的表结构，可以尝试使用create table ... like操作，示例如下：

```
create table sale_detail_like like sale_detail;
```

说明: 此时，sale_detail_like的表结构与sale_detail完全相同。除生命周期属性外，列名、列注释以及表注释等均相同。但sale_detail中的数据不会被复制到sale_detail_like表中。

26.5.3.1.2 删除表（DROP TABLE）

命令格式如下：

```
drop table [if exists] table_name;
```

说明: 如果不指定if exists选项而表不存在，则返回异常；若指定此选项，无论表是否存在，皆返回成功。

示例如下：

```
create table sale_detail_drop like sale_detail; drop table sale_detail_drop;
-- 若表存在，返回成功；若不存在，返回异常。
drop table if exists sale_detail_drop2;
-- 无论是否存在sale_detail_drop2表，均成功返回。
```

26.5.3.1.3 重命名表（RENAME TABLE）

命令格式如下：

```
alter table table_name rename to new_table_name;
```

说明:

- rename操作仅修改表的名字，不改动表中的数据。
- 如果已存在与new_table_name同名表，则会报错。
- 如果table_name不存在，则会报错。

示例如下：

```
create table sale_detail_rename1 like sale_detail;
alter table sale_detail_rename1 rename to sale_detail_rename2;
```

26.5.3.1.4 修改表的注释

命令格式如下：

```
alter table table_name set comment 'tbl comment';
```



说明:

- table_name必须是已存在的表。
- comment最长为1024字节。

示例如下：

```
alter table sale_detail set comment 'new coments for table sale_detail';
```

通过MaxCompute命令desc可以查看表中comment的修改，具体请参考：获取表信息。

26.5.3.1.5 修改表的生命周期属性

MaxCompute提供数据生命周期管理功能，方便用户释放存储空间，简化回收数据的流程。

命令格式如下：

```
alter table table_name set lifecycle days;
```



说明:

- days参数为生命周期时间，只接受正整数。单位：天。
- 如果表table_name是非分区表，自最后一次数据被修改开始计算，经过days天后数据仍未被改动，则此表无需用户干预，将会被MaxCompute自动回收（类似drop table操作）。在MaxCompute中，每当表的数据被修改后，表的LastDataModifiedTime将会被更新，因此，MaxCompute会根据每张表的LastDataModifiedTime以及lifecycle的设置来判断是否要回收此表。
- 如果表table_name是分区表，则根据各分区的LastDataModifiedTime判断该分区是否该被回收。不同于非分区表，分区表的最后一个分区被回收后，该表不会被删除。
- 生命周期只能设定到表级别，不能在分区级设置生命周期。
- 创建表时即可指定生命周期。

示例如下：

```
create table test_lifecycle(key string) lifecycle 100;
-- 新建test_lifecycle表，生命周期为100天。
alter table test_lifecycle set lifecycle 50;
-- 修改test_lifecycle表，将生命周期设为50天。
```

26.5.3.1.6 禁止生命周期

在某些情况下，有些特定的分区可能不希望被生命周期功能自动回收掉。此时，可以禁止该分区被生命周期功能回收。

命令格式如下：

```
ALTER TABLE table_name partition[partition_spec] ENABLE|DISABLE LIFECYCLE;
```

示例如下：

```
ALTER TABLE trans PARTITION(dt='20141111') DISABLE LIFECYCLE;
```

26.5.3.1.7 修改表的修改时间

MaxCompute SQL提供touch操作，用来修改表的LastDataModifiedTime。效果是会将表的LastDataModifiedTime修改为当前时间。

命令格式如下：

```
alter table table_name touch;
```

说明:

- table_name不存在，则报错返回。
- 此操作会改变表的LastDataModifiedTime 的值。此时，MaxCompute会认为表的数据有变动，生命周期的计算会重新开始。

修改分区表的修改时间的相关操作，请参考：修改分区的修改时间。

26.5.3.1.8 清空非分区表里的数据

命令格式如下：

```
TRUNCATE TABLE table_name;
```

说明: 作用是将指定的非分区表中的数据清空。该命令不支持分区表，对于分区表，可以用ALTER TABLE table_name DROP PARTITION (partition_spec) 的方式将分区里的数据清除。

26.5.3.1.9 备份表的数据

如果project里的空间比较紧张，需要进行删除数据或者压缩数据，可以考虑MaxCompute里对表的archive功能，效果是可以将存储空间压缩50%左右。Archive功能将数据存为raid file，数据不再简单的存三份，而是6份数据+3份校验块的方式，这样有效的存储比约为从1:3提高到1:1.5，可以节省一半的物理空间，另外也采用了更高压缩比的压缩算法。

上述的操作也是有代价的，如果某个数据块损坏或某台机器损坏，恢复数据块的时间要比原来的方式更长，读的性能也会有一定损失。 所以现在这种功能可以用在一些冷数据的压缩存储上，例如一些非常大的日志数据，超过一定时间期限后使用的频率非常低，但是又需要长期保存，则可以考虑用raid file来存储。

命令格式如下：

```
ALTER TABLE [table_name] <PARTITION(partition_name='partition_value')> ARCHIVE;
```

示例如下：

```
alter table my_log partition(ds='20140101') archive;
```

输出信息：

```
Summary:
table name: test0128 /pt=a instance count: 1 run time: 21
before merge, file count: 1 file size: 456 file physical size: 1368
after merge, file count: 1 file size: 512 file physical size: 768
```



说明:

在输出信息中可以看到，在archive前后的logical和physical size的变化情况，并且在这个过程中会将多个小文件自动的合并。在archive后，可以用desc extended命令检查该分区是否archive以及物理空间占用情况：

```
desc extended my_log partition(ds='20140101');
+------------------------------------------------------------------------------+
PartitionSize: 512 |
+------------------------------------------------------------------------------+
CreateTime: 2015-01-28 07:05:20 |
LastDDLTime: 2015-01-28 07:05:20 |
LastModifiedTime: 2015-01-28 07:05:21 |
+------------------------------------------------------------------------------+
```

26.5.3.1.10 强制删除表数据（分区数据）

如果用户需要强制删除表数据或者分区数据，并确定该删除操作不需要恢复，需要即时的释放存储空间，可以在执行删除操作时加上purge选项。

命令格式如下：

```
DROP TABLE tblname PURGE;
ALTER TABLE tblname DROP PARTITION(part_spec) PURGE;
```

示例如下：

```
drop table my_log purge;
alter table my_log drop partition (ds='20150618') purge;
```

26.5.3.2 视图操作

26.5.3.2.1 创建视图（CREATE VIEW）

命令格式如下：

```
create [or replace] view [if not exists] view_name
[(col_name [comment col_comment], ...)]
[comment view_comment]
[as select_statement]
```



说明:

- 创建视图时，必须有对视图所引用表的读权限。目前提供的视图不是物化视图，执行时会访问视图引用表的数据。需要注意，引用表的权限的变更会影响视图的访问。
- 视图只能包含一个有效的select语句。
- 视图可以引用其他视图，但不能引用自己，也不能循环引用。
- 不可以向视图写入数据。例如：使用insert into或者insert overwrite操作视图。
- 当视图建好以后，如果视图的引用表发生了变更，有可能导致视图无法访问。例如：删除被引用表。用户需要自己维护引用表及视图之间的对应关系。
- 如果没有指定if not exists，在视图已经存在时用create view会导致异常。这种情况可以用create or replace view来重建视图，重建后视图本身的权限保持不变。

示例如下：

```
create view if not exists sale_detail_view
(store_name, customer_id, price, sale_date, region)
comment 'a view for table sale_detail'
```

```
as select * from sale_detail;
```

26.5.3.2.2 删除视图（DROP VIEW）

命令格式如下：

```
drop view [if exists] view_name;
```



说明: 如果视图不存在且没有指定if exists，则报错。

示例如下：

```
drop view if exists sale_detail_view;
```

26.5.3.2.3 重命名视图（RENAME VIEW）

命令格式如下：

```
alter view view_name rename to new_view_name;
```



说明: 如果已存在同名视图，则报错。

示例如下：

```
create view if not exists sale_detail_view
(store_name, customer_id, price, sale_date, region)
comment 'a view for table sale_detail'
as select * from sale_detail;
alter view sale_detail_view rename to market;
```

26.5.3.3 列及分区操作

26.5.3.3.1 添加分区（ADD PARTITION）

命令格式如下：

```
alter table table_name add [if not exists] partition partition_spec
partition_spec:(partition_col1 = partition_col_value1, partition_col2 = partiton_col_value2, ...)
```



说明:

- 如果未指定if not exists而同名的分区已存在，则报错返回。
- 目前MaxCompute单表支持的分区数量上限为6万。
- 对于多级分区的表，如果想添加新的分区，必须指明全部的分区值。

示例如下：

以给sale_detail表添加一个新的分区为例。

```
alter table sale_detail add if not exists partition (sale_date='201312', region='hangzhou');
-- 成功添加分区，用来存储2013年12月杭州地区的销售记录。
alter table sale_detail add if not exists partition (sale_date='201312', region='shanghai');
-- 成功添加分区，用来存储2013年12月上海地区的销售记录。
alter table sale_detail add if not exists partition(sale_date='20111011');
-- 仅指定一个分区sale_date，报错返回。
alter table sale_detail add if not exists artition(region='shanghai');
-- 仅指定一个分区region，报错返回。
```

26.5.3.3.2 删除分区（DROP PARTITION）

命令格式如下：

```
alter table table_name drop [if exists] partition_spec;

partition_spec:: (partition_col1 = partition_col_value1, partition_col2 = partiton_col_value2, ...)
```



说明: 如果分区不存在且未指定if exists，则报错返回。

示例如下：

以从表sale_detail中删除一个分区为例。

```
alter table sale_detail drop partition(sale_date='201312',region='hangzhou');
-- 成功删除2013年12月杭州分区的销售。
```

26.5.3.3.3 添加列

命令格式如下：

```
alter table table_name add columns (col_name1 type1, col_name2 type2...)
```



说明:

- 列的数据类型只能是：

  bigint，double，boolean，datetime，decimal，string，tinyint，smallint，int，float，varchar，

  及struct类型
- 目前MaxCompute单表的列数上限为1200。

26.5.3.3.4 修改列名

命令格式如下：

```
alter table table_name change column old_col_name rename to new_col_name;
```



说明:

- old_col_name必须是已存在的列。
- 表中不能有名为new_col_name的列。

26.5.3.3.5 修改列、分区注释

命令格式如下：

```
alter table table_name change column col_name comment 'comment';
```



说明:

- comment内容最长1024字节。
- 列的数据类型和位置不能修改。

26.5.3.3.6 修改分区的修改时间

MaxCompute SQL提供touch操作用来修改分区的LastDataModifiedTime。效果是会将分区的LastDataModifiedTime修改为当前时间。

命令格式如下：

```
alter table table_name touch partition(partition_col='partition_col_value', ...);
```



说明:

- table_name或partition_col不存在，则报错返回。
- 指定的partition_col_value不存在，则报错返回。
- 此操作会改变表的LastDataModifiedTime
  的值。此时，MaxCompute会认为表或分区的数据有变动，生命周期的计算会重新开始。

修改表的修改时间的相关操作，请参考：修改表的修改时间。

26.5.3.3.7 修改分区值

MaxCompute SQL支持通过rename操作修改对应表的分区值。

命令格式如下：

```
ALTER TABLE table_name PARTITION (partition_col1 = partition_col_value1, partition_col2 =
 partiton_col_value2, . ..)
RENAME TO PARTITION (partition_col1 = partition_col_newvalue1, partition_col2 =
partiton_col_newvalue2, ...);
```



说明:

- 不支持修改分区列列名，只能修改分区列对应的值。
- 修改多级分区的一个或者多个分区值，多级分区的每一级的分区值都必须写上。

26.5.4 DML语句

26.5.4.1 Insert语句

26.5.4.1.1 更新表中的数据（INSERT OVERWRITE|INTO）

在MaxCompute SQL处理数据的过程中，insert overwrite|into是最常用到的语句，它们用于将计算的结果保存于目标表中，供下一步计算使用。insert into与insert overwrite的区别是：insert into会向表或表的分区中追加数据，而insert overwrite则会在向表或分区中插入数据前，清空表中的原有数据。

命令格式如下：

```
insert overwrite|into table tablename [partition (partcol1=val1, partcol2=val2 ...)] select_statement
from from_statement;
```



说明: MaxCompute的Insert语法与通常使用的MySQL或Oracle的Insert语法有差别，在insert overwrite|into后需要加入table关键字，而不是直接使用tablename。

示例如下：

以计算sale_detail表中不同地区的销售额为例。

```
create table sale_detail_insert like sale_detail;
alter table sale_detail_insert add partition(sale_date='2013', region='china');
insert overwrite table sale_detail_insert partition (sale_date='2013', region='china') select
 shop_name, customer_id, total_price from sale_detail;
```



说明: 在进行insert更新数据操作时，源表与目标表的对应关系依赖于在select子句中列的顺序，而不是表与表之间列名的对应关系。

如下的SQL语句同样合法。

```
insert overwrite table sale_detail_insert partition (sale_date='2013', region='china')
select customer_id, shop_name, total_price from sale_detail;
-- 在创建sale_detail_insert表时，列的顺序为： shop_name string, customer_id string, total_price
 bigint
--         而从sale_detail向sale_detail_insert插入数据是，sale_detail的插入顺序为：customer_id,
 shop_name, total_price
-- 此时，会将sale_detail.customer_id的数据插入sale_detail_insert.shop_name
-- 将sale_detail.shop_name的数据插入sale_detail_insert.customer_id
```

向某个分区表插入数据时，分区列不可以出现在select列表中。

```
insert overwrite table sale_detail_insert partition (sale_date='2013', region='china') select
 shop_name, customer_id, total_price, sale_date, region from sale_detail;
-- 报错返回，sale_date, region为分区列，不可以出现在静态分区的insert语句中。
```

26.5.4.1.2 多路输出（MULTI INSERT）

MaxCompute SQL支持在一个语句中插入不同的结果表或者分区。

命令格式如下：

```
from from_statement
insert overwrite | into table tablename1 [partition (partcol1=val1, partcol2=val2 ...)]
 select_statement1
[insert overwrite | into table tablename2 [partition ...] select_statement2]
```

说明:

- 一般情况下，单个SQL里最多可以写256路输出，超过256路报语法错误。
- 在一个multi insert中，对于分区表，同一个目标分区不可以出现多次；对于非分区表，该表不能出现多次。
- 对于同一张分区表的不同分区，不能同时有insert overwrite和insert into操作，否则报错返回。

示例如下：

```
create table sale_detail_multi like sale_detail;
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2010', region='china' ) select
 shop_name, customer_id, total_price
insert overwrite table sale_detail_multi partition (sale_date='2011', region='china' ) select
 shop_name, customer_id, total_price;
-- 成功返回，将sale_detail的数据插入到sales里的2010年及2011年中国大区的销售记录中。
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2010', region='china' ) select
 shop_name, customer_id, total_price
insert overwrite table sale_detail_multi partition (sale_date='2010', region='china' ) select
 shop_name, customer_id, total_price;
-- 报错返回，同一分区出现多次。
```

```
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2010', region='china' )
select shop_name, customer_id, total_price
insert into table sale_detail_multi partition (sale_date='2011', region='china' ) select shop_name,
 customer_id, total_price;
-- 报错返回，同一张表的不同分区，不能同时有insert overwrite和insert into操作。
```

26.5.4.1.3 输出到动态分区（DYNAMIC PARTITION）

在insert overwrite到一张分区表时，可以在语句中指定分区的值。也可以用另外一种更加灵活的方式，在分区中指定一个分区列名，但不给出值。相应的，在select子句中的对应列来提供分区的值。

命令格式如下：

```
insert overwrite table tablename partition (partcol1, partcol2 ...) select_statement from
 from_statement;
```

说明:

- 目前，在使用动态分区功能的SQL中，在分布式环境下，单个进程最多只能输出512个动态分区，否则引发运行时异常。
- 在现阶段，任意动态分区SQL不可以生成超过2000个动态分区，否则引发运行时异常。
- 动态生成的分区值不可以为NULL，否则会引发异常。
- 如果目标表有多级分区，在运行insert语句时允许指定部分分区为静态，但是静态分区必须是高级分区。

动态分区的示例如下：

```
create table total_revenues (revenue bigint) partitioned by (region string); insert overwrite table
 total_revenues partition(region)
select total_price as revenue, region from sale_detail;
```

说明: 按照上述示例，在SQL运行之前，是不知道会产生哪些分区的，只有在select运行结束后，才能由region字段产生的值确定会产生哪些分区，这也是为什么叫做动态分区的原因。

其他示例如下：

```
create table sale_detail_dypart like sale_detail;
insert overwrite table sale_detail_dypart partition (sale_date, region) select * from sale_detail;
-- 成功返回。
insert overwrite table sale_detail_dypart partition (sale_date='2013', region) select
 shop_name,customer_id,total_price,region from sale_detail;
-- 成功返回，多级分区，指定一级分区。
```

```
insert overwrite table sale_detail_dypart partition (sale_date='2013', region) select
 shop_name,customer_id,total_price from sale_detail;
-- 失败返回，动态分区插入时，动态分区列必须在select列表中。
insert overwrite table sales partition (region='china', sale_date) select
 shop_name,customer_id,total_price,region from sale_detail;
-- 失败返回，不能仅指定低级子分区，而动态插入高级分区。
```

26.5.4.2 Select语句

26.5.4.2.1 SELECT操作

命令格式如下：

```
select [all | distinct] select_expr, select_expr, ... from table_reference
[where where_condition] [group by col_list]
[order by order_condition]
[distribute by distribute_condition [sort by sort_condition] ] [limit number]
```

在使用select语句时需要了解和注意如下几点：

- select 操作从表中读取数据，要读的列可以用列名指定，或者用 * 代表所有的列。

  示例如下：

  ```
  select * from sale_detail;
  -- 读取sale_detail表中所有列。
  select shop_name from sale_detail;
  -- 只读取sale_detail表中的一列shop_name。
  ```

  说明: 当使用Select语句屏显时，目前最多只能显示1000行结果。当select作为子句时，无此限制，select 子句会将全部结果返回给上层查询。若需要获取超过1000行select的数据结果，需要使用tunnel下载该表或者select出的临时表，详细信息请参考： *tunnel* 使用指南。

- 在where子句中可以指定过滤的条件。

  示例如下：

  ```
  select * from sale_detail where shop_name like 'hang%';
  ```

  where子句支持的过滤条件，如下表所示：

过滤条件	描述
>, < , =, >=, <=, <>	/
like, rlike	/
in, not in	如果在in/not in条件后加子查询，子查询只能返回一列值，且返回值的数量不能超过1000。

在select语句的where子句中可以指定分区范围，这样可以仅扫描表的指定部分，避免全表扫描。

示例如下：

```
select sale_detail.* from sale_detail
where sale_detail.sale_date >= '2008' and sale_detail.sale_date <= '2014';
```



说明: MaxCompute SQL的where子句不支持between条件查询，Where子句中可使用条件个数不超过256个。

- 在table_reference中支持使用嵌套子查询。

  示例如下：

  ```
  select * from (select region from sale_detail) t where region = 'shanghai';
  ```

- distinct：如果有重复数据行时，在字段前使用distinct，会将重复字段去重，只返回一个值，而使用all将返回字段中所有重复的值。不指定此选项时默认效果和all相同，使用distinct则只返回一行记录。

  示例如下：

  ```
  select distinct region from sale_detail;
  select distinct region, sale_date from sale_detail;
  -- distinct多列，distinct的作用域是select的列集合，不是单个列。
  ```

- group by：分组查询，一般group by是和聚合函数配合使用。在select中包含聚合函数时，group by的key可以是输入表的列名，也可以是由输入表的列构成的表达式，不可以是select语句的输出列的别名。

  示例如下：

  ```
  select region from sale_detail group by region;
  -- 直接使用输入表列名作为group by的列，可以运行
  select sum(total_price) from sale_detail group by region;
  -- 以region值分组，返回每一组的销售额总量，可以运行
  select region, sum(total_price) from sale_detail group by region;
  -- 以region值分组，返回每一组的region值(组内唯一)及销售额总量，可以运行
  select region as r from sale_detail group by r;
  -- 使用select列的别名运行，报错返回
  select 'China-' + region as r from sale_detail group by 'China-' + region;
  -- 必须使用列的完整表达式
  select region, total_price from sale_detail group by region;
  -- 报错返回，select的所有列中，没有使用聚合函数的列，必须出现在group by中
  select region, total_price from sale_detail group by region, total_price;
  ```

```
-- 可以运行
```



说明: 有上述的限制是因为在SQL解析中，group by操作通常是先于select操作的，因此group by只能接受输入表的列或表达式为key。关于聚合函数的详细信息，请参考：聚合函数。

- order by：对所有数据按照某几列进行全局排序。如果用户希望按照降序对记录进行排序，可以使用DESC关键字。由于是全局排序，order by必须与limit共同使用。在使用order by排序时，NULL会被认为比任何值都小，这个行为与Mysql一致，但是与Oracle不一致。与group by不同，order by后面必须加select列的别名，当select某列时，如果没有指定列的别名，会默认将列名作为列的别名。

  示例如下：

  ```
  select * from sale_detail order by region;
  -- 报错返回，order by没有与limit共同使用
  select * from sale_detail order by region limit 100;
  select region as r from sale_detail order by region;
  -- 报错返回，order by后面必须加列的别名。
  select region as r from sale_detail order by r;
  ```

  说明: [limit number]中的number是常数，限制输出行数。当使用无limit的select语句直接从屏幕输出查看结果时，最多只输出5000行。每个项目空间的屏显最大限制可能不同，可以通过控制台面板控制。

- distribute by：对数据按照某几列的值做hash分片，必须使用select的输出列别名。

  示例如下：

  ```
  select region from sale_detail distribute by region;
  -- 列名即是别名，可以运行
  select region as r from sale_detail distribute by region;
  -- 报错返回，后面必须加列的别名。
  select region as r from sale_detail distribute by r;
  ```

- sort by：局部排序，语句前必须加distribute by。实际上sort by是对distribute by的结果进行局部排序。必须使用select的输出列别名。

  示例如下：

  ```
  select region from sale_detail distribute by region sort by region; select region as r from
   sale_detail sort by region;
  -- 没有distribute by，报错退出。
  ```

- order by不和distribute by/sort by共用，同时group by也不和distribute by/sort by共用，必须使用select的输出列别名。

说明:

- order by/sort by/distribute by的key必须是select语句的输出列，即列的别名。
- 在MaxCompute SQL解析中，order by/sort by/distribute by是后于select操作的，因此它们只能接受select语句的输出列为key。

26.5.4.2.2 子查询

普通的select是从几张表中读数据，如select column_1, column_2 ... from table_name，但查询的对象也可以是另外一个 select操作，即子查询。

命令格式如下：

```
select * from (select shop_name from sale_detail) a;
```

注意: 子查询必须要有别名。

示例如下：

```
create table shop as select * from sale_detail;
select a.shop_name, a.customer_id, a.total_price from
(select * from shop) a join sale_detail on a.shop_name = sale_detail.shop_name;
```

说明: 在from子句中，子查询可以当作一张表来使用，与其他的表或子查询进行join操作。

26.5.4.3 Union语句

26.5.4.3.1 UNION ALL

命令格式如下：

```
select_statement union all select_statement
```

说明: 将两个或多个select操作返回的数据集联合成一个数据集，如果结果有重复行时，会返回所有符合条件的行，不进行重复行的去重处理。

MaxCompute SQL不支持顶级的两个查询结果合并，要改写为一个子查询的形式。

改写前错误示例如下：

```
select * from sale_detail where region = 'hangzhou'
union all
```

```
select * from sale_detail where region = 'shanghai';
```

改写后正确示例如下：

```
select * from (
select * from sale_detail where region = 'hangzhou' union all
select * from sale_detail where region = 'shanghai') t;
```

注意:

- union all操作对应的各个子查询的列个数、名称和类型必须一致。如果列名不一致时，可以使用列的别名加以解决。
- 一般情况下，MaxCompute最多允许256路union all，超过此限制时报语法错误。

26.5.4.4 Join语句

26.5.4.4.1 JOIN操作

MaxCompute的JOIN支持多路连接，但不支持笛卡尔积，即无on条件的连接。

命令格式如下：

```
join_table:
table_reference join table_factor [join_condition]
| table_reference {left outer|right outer|full outer|inner} join table_reference join_condition
table_reference: table_factor
join_table
table_factor: tbl_name [alias]
table_subquery alias
( table_references )
join_condition:
on equality_expression ( and equality_expression )*
```

说明: equality_expression是一个等式表达式。

在使用join语句时需要了解和注意如下几点：

- left outer join：左连接，返回左表（即如下示例中的shop）中的所有记录，即使在右表（即如下示例中的sale_detail）中没有匹配的行。

  示例如下：

  ```
  select a.shop_name as ashop, b.shop_name as bshop from shop a left outer join sale_detail
   b on a.shop_name=b.shop_name;
  -- 由于表shop及表sale_detail中都有shop_name列，因此需要在select子句中使用别名进行区分。
  ```

- right outer join：右连接，返回右表（即如下示例中的sale_detail）中的所有记录，即使在左表（即如下示例中的shop）中没有匹配的行。

示例如下：

```
select a.shop_name as ashop, b.shop_name as bshop from shop a right outer join sale_detail
 b on a.shop_name=b.shop_name;
-- 由于表shop及表sale_detail中都有shop_name列，因此需要在select子句中使用别名进行区
分。
```

- full outer join：全连接，返回左右表中的所有记录。

  示例如下：

  ```
  select a.shop_name as ashop, b.shop_name as bshop from shop a full outer join sale_detail
   b on a.shop_name=b.shop_name;
  ```

- inner join：在表中存在至少一个匹配时，inner join返回行，关键字inner可省略。

  示例如下：

  ```
  select a.shop_name from shop a inner join sale_detail b on a.shop_name=b.shop_name;
   select a.shop_name from shop a join sale_detail b on a.shop_name=b.shop_name;
  ```

- 连接条件：只允许and连接的等值条件，并且最多支持16路join操作。只有在MAPJOIN中，可以使用不等值连接或者使用or连接多个条件。

  示例如下：

  ```
  select a.* from shop a full outer join sale_detail b on a.shop_name=b.shop_name full outer
   join sale_detail c on a.shop_name=c.shop_name;
  -- 支持多路join链接，最多支持16路join
  select a.* from shop a join sale_detail b on a.shop_name <> b.shop_name;
  -- 不支持不等值Join链接条件，报错返回。
  ```

26.5.4.4.2 MAPJOIN HINT

当一个大表和一个或多个小表做join时，可以使用mapjoin，性能比普通的join要快很多。

Mapjoin的基本原理是：在小数据量情况下， SQL会将用户指定的小表全部加载到执行join 操作的程序的内存中，从而加快join的执行速度。

示例如下：

```
select /* + mapjoin(a) */ a.shop_name, b.customer_id, b.total_price
from shop a join sale_detail b
on a.shop_name = b.shop_name;
```



注意:

在使用mapjoin时，需要注意如下限制：

- left outer join的左表必须是大表。

- right outer join的右表必须是大表。
- inner join的左表或右表均可以作为大表。
- full outer join不能使用mapjoin。
- mapjoin支持小表为子查询。
- 使用mapjoin时需要引用小表或子查询时，需要引用别名。
- 在mapjoin中，可以使用不等值连接或者使用or连接多个条件。
- MaxCompute在mapjoin中最多支持指定8张小表，否则报语法错误。
- 如果使用mapjoin，则所有小表占用的内存总和不得超过512MB。该限制值可以用参数*odps.sql.mapjoin.memory.max*调整，最大为2048M。

> **注意:** 该大小指的是数据解压后的大小，而通过desc命令查询相关表显示的是数据压缩后的大小，需要在计算时按照压缩格式乘以压缩比。

MaxCompute SQL不支持在普通join的on条件中使用不等值表达式或or逻辑等复杂的join条件，但是在mapjoin中可以进行如上操作，示例如下：

```
select /*+ mapjoin(a) */ a.total_price, b.total_price
from shop a join sale_detail b
on a.total_price < b.total_price or a.total_price + b.total_price < 500;
```

26.5.4.5 Explain语句

MaxCompute SQL提供Explain操作，用来显示对应于DML语句的最终执行计划结构的描述。执行计划就是最终用来执行SQL语义的程序。

命令格式如下：

```
EXPLAIN <DMLquery>;
```

> **说明:**
>
> Explain的执行结果包含如下内容：
>
> - 对应于该DML语句的所有Task的依赖结构。
> - Task中所有Task的依赖结构。
> - Task中所有Operator的依赖结构。

示例如下：

```
EXPLAIN
```

```
SELECT abs(a.key), b.value FROM src a JOIN src1 b ON a.value = b.value;
```

Explain的输出结果包含如下三个部分：

- 首先是Job间的依赖关系。

  输出结果示例如下：

  ```
  job0 is root job
  ```

  说明: 因为该query只需要一个Job（job0），所以只需要一行信息。

- 其次是Task间的依赖关系。

  输出结果示例如下：

  ```
  In Job job0:
  root Tasks: M1_Stg1, M2_Stg1
  J3_1_2_Stg1 depends on: M1_Stg1, M2_Stg1
  ```

  说明:

  - job0包含三个Task，M1_Stg1和M2_Stg1两个Task会先执行，两者执行完成后J3_1_2_Stg1执行。
  - Task的命名规则：在MaxCompute中，共有四种Task类型：MapTask，ReduceTask，JoinTask和LocalWork。Task名称的第一个字母表示了当前Task的类型（如“M2Stg1”就是一个MapTask）；紧跟着第一个字母后的数字代表了当前Task的ID，这个ID在所有对应当前query的Task中是唯一的；之后用下划线“ \”分隔的数字代表了当前Task的直接依赖（如“J3_1_2_Stg1”意味着当前Task（ID为3）依赖ID为1和ID为2的两个Task）。

- 第三部分是Task中的Operator结构，Operator串描述了一个Task的执行语义。

  输出结果示例如下：

  ```
  In Task M1_Stg1:
  Data source: yudi_2.src #### "Data source"描述了当前Task的输入内容 TS: alias: a #### TableScanOperator
  RS: order: + #### ReduceSinkOperator keys:
  a.value values:
  a.key partitions:
  a.value
  In Task J3_1_2_Stg1:
  JOIN: a INNER JOIN b #### JoinOperator
  SEL: Abs(UDFToDouble(a._col0)), b._col5 #### SelectOperator FS: output: None #### FileSinkOperator
  In Task M2_Stg1:
  ```

```
Data source: yudi_2.src1 TS: alias: b
RS: order: + keys:
b.value values:
b.value partitions:
b.value
```

说明:

- 各个Operator的含义如下所示。
  - TableScanOperator：描述了query语句中“FROM”语句块的逻辑，explain结果中会显示输入表名称（alias）。
  - SelectOperator：描述了query语句中的“SELECT”语句块的逻辑，explain结果中会显示向下一个operator传递的列，多个列由逗号分隔。如果是列的引用，会显示成"< alias >.< column_name >"；如果是表达式的结果，会显示函数的形式，如 “func1(arg1_1, arg1_2, func2(arg2_1, arg2_2))” ；若是常量，则直接显示值内容。
  - FilterOperator：描述了query语句中的“WHERE”语句块的逻辑，explain结果中会显示一个WHERE条件表达式，形式类似SelectOperator的显示规则。
  - JoinOperator：描述了query语句中的“JOIN”语句块的逻辑，explain结果中会显示哪些表用哪种方式join在一起。
  - GroupByOperator：描述了聚合操作的逻辑，如果query中使用了聚合函数，就会出现该结构，explain结果中会显示聚合函数的内容。
  - ReduceSinkOperator：描述了Task间数据分发操作的逻辑，如果当前Task的结果会传递给另一个Task，则必然需要在当前Task的最后使用ReduceSinkOperator来执行数据分发操作。explain结果中会显示输出结果的排序方式、分发的key、value以及用来求hash值的列。
  - FileSinkOperator：描述了最终数据的存储操作，如果query中有insert语句块，explain结果中会显示目标表名称。
  - LimitOperator：描述了query语句中的“LIMIT”语句块的逻辑，explain结果中会显示limit数。
  - MapjoinOperator：类似JoinOperator，描述了大表的Join操作。
- 如果query足够复杂，explain的结果太多，会导致触发API的限制，使得用户看到的explain结果不完整。此时可以通过拆分query，各部分分别explain，来了解job的结构。

- 最多查询的partition数目不超过10000个，输入的partition过多，会导致**Data source**的内容过长。此时可以通过在query中加入partition的过滤条件，过滤掉大部分的partition，以此规避这个限制。

26.5.5 内建函数

26.5.5.1 数学运算函数

26.5.5.1.1 ABS

函数声明如下：

```
double abs(double number)
bigint abs(bigint number)
decimal abs(decimal number)
```

用途：**返回绝对值。**

参数说明：

number：Double或bigint或Decimal　类型。输入为bigint类型时返回bigint类型，输入为double类型时返回double类型。若输入为string类型会隐式转换到double类型后参与运算，为其他类型时报错。

返回值：Double或者bigint或者decimal类型，取决于输入参数的类型。若输入参数为NULL，返回NULL。



说明: 当输入bigint类型的值超过bigint的最大表示范围时，会返回double类型，这种情况下可能会损失精度。

示例如下：

```
abs(null) = null
abs(-1) = 1
abs(-1.2) = 1.2
abs("-2") = 2.0
abs(12232083745629837659238745692 3748) = 1.2232083745629837e32
```

下面是一个完整的abs函数在SQL中使用的示例，其他内建函数(除窗口函数、聚合函数外)的使用方式与其类似，不再一一举例。

```
select abs(id) from tbl1;
```

```
-- 取tbl1表内id字段的绝对值
```

26.5.5.1.2 ACOS

函数声明如下：

```
double acos(double number)
decimal acos(decimal number)
```

用途：**计算 number 的反余弦函数。**

参数说明：

number：Double类型或Decimal类型，-1≤number≤1。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。

返回值：Double类型或Decimal类型，值域在0 ~ π之间。若number为NULL，返回NULL。

示例如下：

```
acos("0.87") = 0.5155940062460905
acos(0) = 1.5707963267948966
```

26.5.5.1.3 ASIN

函数声明如下：

```
double asin(double number)
decimal asin(DECIMAL number)
```

用途：**反正弦函数。**

参数说明：

number：Double类型或Decimal类型，-1≤number≤1。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。

返回值：Double类型或Decimal类型，值域在-π/2 ~π/2之间。若 number为NULL，返回NULL。

示例如下：

```
asin(1) = 1.5707963267948966
```

```
asin(-1) = -1.5707963267948966
```

26.5.5.1.4 ATAN

函数声明如下：

```
double atan(double number)
```

用途：反正切函数。

参数说明：

number：Double类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。

返回值：Double类型，值域在-π/2 ~π/2之间。若number为NULL，返回NULL。

示例如下：

```
atan(1) = 0.7853981633974483;
atan(-1) = -0.7853981633974483
```

26.5.5.1.5 CEIL

函数声明如下：

```
bigint ceil(double value)
bigint ceil(decimal value)
```

用途：返回不小于输入值value的最小整数。

参数说明：

value：Double 类型或 Decimal 类型，若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。

返回值：Bigint类型。若任意一个输入参数为NULL，返回NULL。

示例如下：

```
ceil(1.1) = 2
```

```
ceil(-1.1) = -1
```

26.5.5.1.6 CONV

函数声明如下：

```
string conv(string input, bigint from_base, bigint to_base)
```

用途：进制转换函数。

参数说明：

- input：以string类型表示的要转换的整数值，接受bigint类型及double类型的隐式转换。
- from_base，to_base：以十进制表示进制的值，可接受的值为2，8，10，16。接受string类型及double类型的隐式转换。

返回值：String 类型。若任意一个输入参数为NULL，返回NULL。转换过程以64位精度工作，溢出时报错。输入如果是负值，即以“-”开头，并报错。 如果输入的是小数，则会转为整数值后进行进制转换，小数部分会被舍弃。

示例如下：

```
conv('1100', 2, 10) = '12'
conv('1100', 2, 16) = 'c'
conv('ab', 16, 10) = '171'
conv('ab', 16, 16) = 'ab'
```

26.5.5.1.7 COS

函数声明如下：

```
double cos(double number)
decimal cos(decimal number)
```

用途：余弦函数，输入为弧度值。

参数说明：

number：Double类型或Decimal类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。

返回值：Double类型或Decimal类型。若number为NULL，返回NULL。

示例如下：

```
cos(3.1415926/2) = 2.6794896585028633e-8
```

```
cos(3.1415926) = -0.9999999999999986
```

26.5.5.1.8 COSH

函数声明如下：

```
double cosh(double number)
decimal cosh(decimal number)
```

用途：双曲余弦函数。

参数说明：

number：Double类型或Decimal类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。

返回值：Double类型或Decimal类型。若number为NULL，返回NULL。

26.5.5.1.9 COT

函数声明如下：

```
double cot(double number)
decimal cot(decimal number)
```

用途：余切函数，输入为弧度值。

参数说明：

number：Double或Decimal类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。

返回值：Double类型或Decimal类型。若number为NULL，返回NULL。

26.5.5.1.10 EXP

函数声明如下：

```
double exp(double number)
decimal exp(decimal number)
```

用途：指数函数。返回number的指数值。

参数说明：

number：Double或Decimal 类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。

返回值：Double或Decimal类型，若number为NULL，返回NULL。

26.5.5.1.11 FLOOR

函数声明如下：

```
bigint floor(double number)
bigint floor(decimal number)
```

用途：向下取整，返回比number小的整数值。

参数说明：

number：Double或Decimal类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。

返回值：Bigint类型。若number为NULL，返回NULL。

示例如下：

```
floor(1.2) = 1
floor(1.9) = 1
floor(0.1) = 0
floor(-1.2) = -2
floor(-0.1) = -1
floor(0.0) = 0
floor(-0.0) = 0
```

26.5.5.1.12 LN

函数声明如下：

```
double ln(double number)
decimal ln(decimal number)
```

用途：返回number的自然对数。

参数说明：

number：Double或Decimal类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。

返回值：Double或Decimal类型。若number为NULL返回NULL，若number为负数或零，返回NULL。

26.5.5.1.13 LOG

函数声明如下：

```
double log(double base, double x)
decimal log(decimal base, DECIMAL x)
```

用途：返回以base为底的x的对数。

参数说明：

- base：Double类型或Decimal类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。
- x：Double类型或Decimal类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。

返回值：Double类型或Decimal类型的对数值。若base和x中存在NULL，返回NULL；若base和x中某一个值为负数或零，返回NULL；若base为1（会引发一个除零行为）也会返回NULL。

26.5.5.1.14 POW

函数声明如下：

```
double pow(double x, double y)
decimal pow(decimal x, DECIMAL y)
```

用途：返回x的y次方，即 x^y。

参数说明：

- X：Double类型或Decimal 类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。
- Y：Double类型或Decimal 类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。

返回值：Double类型或Decimal类型。若x或y为NULL，则返回NULL。

26.5.5.1.15 RAND

函数声明如下：

```
double rand(bigint seed)
```

用途：以seed为种子返回double类型的随机数，返回值的区间是0 ~ 1。

参数说明：

seed：可选参数，Bigint类型。随机数种子，决定随机数序列的起始值。

返回值：Double类型。

示例如下：

```
select rand() from dual;
select rand(1) from dual;
```

26.5.5.1.16 ROUND

函数声明如下：

```
double round(double number, [bigint decimal_places])
decimal round(decimal number, [bigint decimal_places])
```

用途：四舍五入到指定小数点位置。

参数说明：

- number：Double类型或Decimal类型。若输入为string类型或bigint类型会隐式转换到double 类型后参与运算，为其他类型时报错。
- decimal_place：Bigint类型常量。四舍五入计算到小数点后的位置，为其他类型参数会报错。如果省略表示四舍五入到个位数。默认值为0。

返回值：Double类型或Decimal类型。若number或decimal_places为NULL，返回 NULL。



说明： decimal_places可以是负数。负数会从小数点向左开始计数，并且不保留小数部分；如果decimal_places超过了整数部分长度，返回0。

示例如下：

```
round(125.315) = 125.0
round(125.315, 0) = 125.0
round(125.315, 1) = 125.3
round(125.315, 2) = 125.32
round(125.315, 3) = 125.315
round(-125.315, 2) = -125.32
round(123.345, -2) = 100.0
round(null) = null
round(123.345, 4) = 123.345
round(123.345, -4) = 0.0
```

26.5.5.1.17 SIN

函数声明如下：

```
double sin(double number)
```

```
decimal sin(decimal number)
```

用途：正弦函数，输入为弧度值。

参数说明：

number：Double类型或Decimal类型。若输入为string类型或bigint类型会隐式转换到double 类型后参与运算，为其他类型时报错。

返回值：Double类型或Decimal类型。若number为NULL，返回NULL。

26.5.5.1.18 SINH

函数声明如下：

```
double sinh(double number)
decimal sinh(decimal number)
```

用途：双曲正弦函数。

参数说明：

number：Double类型或Decimal 类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。

返回值：Double类型或Decimal类型。若number为NULL，返回NULL。

26.5.5.1.19 SQRT

函数声明如下：

```
double sqrt(double number)
decimal sqrt(decimal number)
```

用途：计算平方根。

参数说明：

number：Double类型或Decimal类型。必须大于0，小于0时报错。若输入为string类型或bigint类型会隐式转换到double 类型后参与运算，为其他类型时报错。

返回值：Double类型或Decimal类型。若number为NULL，返回NULL。

26.5.5.1.20 TAN

函数声明如下：

```
double tan(double number)
```

```
decimal tan(decimal number)
```

用途：正切函数，输入为弧度值。

参数说明：

number：Double类型或Decimal类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。

返回值：Double类型或Decimal类型。若number为NULL，返回NULL。

26.5.5.1.21 TANH

函数声明如下：

```
double tanh(double number)
decimal tanh(decimal number)
```

用途：双曲正切函数。

参数说明：

number：Double类型或Decimal类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。

返回值：Double类型或Decimal类型。若number为NULL，返回NULL。

26.5.5.1.22 TRUNC

函数声明如下：

```
double trunc(double number[, bigint decimal_places])
decimal trunc(decimal number[, bigint decimal_places])
```

用途：将输入值 number 截取到指定小数点位置。

参数说明：

- number：Double类型或Decimal 类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。
- decimal_places：Bigint类型常量。要截取到的小数点位置，为其他类型参数会隐式转为bigint类型。如果省略表示默认截取到个位数。

返回值：Double类型或Decimal类型。若number或decimal_places为NULL，返回NULL。



说明:

- 截取掉的部分补0。
- decimal_places可以是负数，负数会从小数点向左开始截取，并且不保留小数部分；如果decimal_places超过了整数部分长度，返回0。

示例如下：

```
trunc(125.815) = 125.0
trunc(125.815, 0) =125.0
trunc(125.815, 1) = 125.80000000000001
trunc(125.815, 2) = 125.81
trunc(125.815, 3) = 125.815
trunc(-125.815, 2) = -125.81
trunc(125.815, -1) = 120.0
trunc(125.815, -2) = 100.0
trunc(125.815, -3) = 0.0
trunc(123.345, 4) = 123.345
trunc(123.345, -4) = 0.0
```

26.5.5.1.23 新扩展数字函数说明

升级到MaxCompute2.0以后，产品扩展了部分数字函数，其中**UNHEX函数**在使用函数的sql前，需要添加如下的set语句。

```
set odps.sql.type.system.odps2=true;
```

本章节后介绍的数字函数即为新扩展的数字函数。

26.5.5.1.24 LOG2

函数声明如下：

```
Double log2(DOUBLE number)
Double log2(DECIMAL number)
```

用途：以2为底返回number的对数。

参数类型：

number：Double类型或Decimal类型。

返回值：Double类型。若输入参数为0或NULL，返回NULL。

示例如下：

```
log2(null) = null
log2(0) = null
```

```
log2(8) = 3.0
```

26.5.5.1.25 LOG10

函数声明如下：

```
Double log10(DOUBLE number)
Double log10(DECIMAL number)
```

用途：以10为底返回number的对数。

参数类型：

number：Double类型或Decimal类型。

返回值：Double类型。若输入参数为0或NULL，返回NULL。

示例如下：

```
log10(null) = null
log10(0) = null
log10(8) = 0.9030899869919435
log10('abc') = null
```

26.5.5.1.26 BIN

函数声明如下：

```
string bin(BIGINT number)
```

用途：返回number的二进制代码表示。

参数类型：

number：Bigint类型。

返回值：String类型。若输入参数为0，返回0；若输入参数为NULL，返回NULL。

示例如下：

```
bin(0) = '0'
bin(null) = 'null'
bin(12) = '1100'
```

26.5.5.1.27 HEX

函数声明如下：

```
STRING hex(BIGINT number)
STRING hex(STRING number)
```

```
STRING hex(BINARY number)
```

用途：将整数或字符转换为十六进制格式。

参数类型：

number：如果number是Bigint类型，那么返回number的十六进制表示；如果变量是string类型，则返回该字符串的十六进制表示。

返回值：String类型。若输入参数为0，返回0；若输入参数为NULL，返回NULL。

示例如下：

```
hex(0) = '0'
hex('abc') = '616263'
hex(17) = '11'
hex('17') = '3137'
hex(null) = 'null'
```

26.5.5.1.28 UNHEX

函数声明如下：

```
BINARY unhex(STRING number)
```

用途：返回十六进制字符串所代表的字符串。

参数类型：

number：为十六进制字符串。

返回值：Binary类型。若输入参数为0，返回失败；若输入参数为NULL，返回NULL。

示例如下：

```
unhex('616263') = 'abc'
unhex(616263) = 'abc'
```

26.5.5.1.29 RADIANS

函数声明如下：

```
DOUBLE radians(DOUBLE number)
```

用途：将角度转换为弧度。

参数类型：

number：Double类型。

返回值：Double类型。若输入参数为NULL，返回NULL。

示例如下：

```
radians(90) = 1.5707963267948966
radians(0) = 0.0
radians(null) = null
```

26.5.5.1.30 DEGREES

函数声明如下：

```
DOUBLE degrees(DOUBLE number)
DOUBLE degrees(DECIMAL number)
```

用途：将弧度转换为角度。

参数类型：

number：Double或Decimal类型。

返回值：Double类型。若输入参数为NULL，返回NULL。

示例如下：

```
degrees(1.5707963267948966) = 90.0
degrees(0) = 0.0
degrees(null) = null
```

26.5.5.1.31 SIGN

函数声明如下：

```
DOUBLE sign(DOUBLE number)
DOUBLE sign(DECIMAL number)
```

用途：取输入数据的符号，“1.0”表示正，“-1.0”表示负，否则“0.0”。

参数类型：

number：Double或Decimal类型。

返回值：Double类型。若输入参数为0，返回0.0；若输入参数为NULL，返回NULL。

示例如下：

```
sign(-2.5) = -1.0
sign(2.5) = 1.0
sign(0) = 0.0
```

```
sign(null) = null
```

26.5.5.1.32 E

函数声明如下：

```
DOUBLE e()
```

用途：返回e的值。

返回值：Double类型。

示例如下：

```
e() = 2.718281828459045
```

26.5.5.1.33 PI

函数声明如下：

```
DOUBLE pi()
```

用途：返回π的值。

返回值：Double类型。

示例如下：

```
pi() = 3.141592653589793
```

26.5.5.1.34 FACTORIAL

函数声明如下：

```
BIGINT factorial(INT number)
```

用途：返回number的阶乘。

参数类型：

number：Int类型，并且值域在[0..20]之间。

返回值：Bigint类型。若输入参数为0，返回1；若输入参数为NULL或[0..20]之外的数值，返回NULL。

示例如下：

```
factorial(5) = 120 --5!= 5*4*3*2*1 = 120
```

26.5.5.1.35 CBRT

函数声明如下：

```
DOUBLE cbrt(DOUBLE number)
```

用途：返回立方根。

参数类型：

number：Double类型。

返回值：Double类型。若输入参数为NULL，返回NULL。

示例如下：

```
cbrt(8) = 2
cbrt(null) = null
```

26.5.5.1.36 SHIFTLEFT

函数声明如下：

```
INT shiftleft(TINYINT|SMALLINT|INT number1, INT number2)
BIGINT shiftleft(BIGINT number1, INT number2)
```

用途：按位左移（<<）。

参数类型：

- number1：Tinyint|Smallint|Int|Bigint整形数据。
- number2：Int整形数据。

返回值：Int类型或Bigint类型。

示例如下：

```
shiftleft(1,2) = 4
-- 1的二进制左移2位（1<<2,0001左移两位是0100）
shiftleft(4,3) = 32
```

```
-- 4的二进制左移3位（4<<3,0100左移3位是100000）
```

26.5.5.1.37 SHIFTRIGHT

函数声明如下：

```
INT shiftright(TINYINT|SMALLINT|INT number1, INT number2)
BIGINT shiftright(BIGINT number1, INT number2)
```

用途：按位右移（>>）。

参数类型：

- number1：Tinyint|Smallint|Int|Bigint整形数据。
- number2：Int整形数据。

返回值：Int类型或Bigint类型。

示例如下：

```
shiftright(4,2) = 1
-- 4的二进制右移2位（4>>2,0100右移两位是0001）
shiftright(32,3) = 4
-- 32的二进制右移3位（32>>3,100000右移3位是0100）
```

26.5.5.1.38 SHIFTRIGHTUNSIGNED

函数声明如下：

```
INT shiftrightunsigned(TINYINT|SMALLINT|INT number1, INT number2)
BIGINT shiftrightunsigned(BIGINT number1, INT number2)
```

用途：无符号按位右移（>>>）。

参数类型：

- number1：Tinyint|Smallint|Int|Bigint整形数据。
- number2：Int整形数据。

返回值：Int类型或Bigint类型。

示例如下：

```
shiftrightunsigned(8,2) = 2
-- 8的二进制无符号右移2位(8>>>2,1000右移两位是0010)
shiftrightunsigned(-14,2) = 1073741820
```

```
-- -14的二进制右移2位(-14>>>2，11111111 11111111 11111111 11110010右移2位是 00111111
 11111111 11111111 11111100)
```

26.5.5.2 字符串处理函数

26.5.5.2.1 CHAR_MATCHCOUNT

函数声明如下：

```
bigint char_matchcount(string str1, string str2)
```

用途：用于计算str1中有多少个字符出现在str2中。

参数说明：

str1，str2：String类型。必须为有效的UTF-8字符串，如果对比中发现有无效字符则函数返回负值。

返回值：Bigint类型。若任意一个输入参数为NULL，返回NULL。

示例如下：

```
char_matchcount('abd', 'aabc') = 2
-- str1中的两个字符串'a','b'在str2中出现过
```

26.5.5.2.2 CHR

函数声明如下：

```
string chr(bigint ascii)
```

用途：将给定ASCII码ascii转换成字符。

参数说明：

ascii：Bigint类型ASCII值。若输入为string类型，double类型或decimal类型会隐式转换到bigint类型后参与运算，为其他类型时报错。

返回值：String类型。参数范围是0~255，超过此范围会报错。若输入参数为NULL，返回NULL。

26.5.5.2.3 CONCAT

函数声明如下：

```
string concat(string a, string b...)
```

用途：返回值是将参数中的所有字符串连接在一起的结果。

参数说明：

a，b...：String 类型，若输入为bigint类型，decimal类型，double类型或datetime类型会隐式转换为string类型后参与运算，为其他类型时报错。

返回值：String类型。若没有输入参数为NULL或任意一个输入参数为NULL，均返回NULL。

示例如下：

```
concat('ab', 'c') = 'abc'
concat() = null
concat('a', null, 'b') = null
```

26.5.5.2.4 INSTR

函数声明如下：

```
bigint instr(string str1, string str2[, bigint start_position[, bigint nth_appearance]])
```

用途：计算子串str2在字符串str1中的位置。

参数说明：

- str1：String类型。搜索的字符串。若输入为bigint类型，decimal类型，double类型或datetime类型会隐式转换为string后参与运算，为其他类型时报错。
- str2：String类型。要搜索的子串。若输入为bigint类型，decimal类型，double类型或datetime类型会隐式转换为string后参与运算，为其他类型时报错。
- start_position：Bigint类型。为其他类型会报错。表示从str1的第几个字符开始搜索，默认起始位置是第一个字符位置1。
- nth_appearance：Bigint类型。大于0，表示子串在字符串中的第nth_appearance次匹配的位置，如果nth_appearance为其他类型或小于等于0会报错。

返回值：Bigint类型。



说明:

- 若在str1中未找到str2，返回0。
- 若任意一个输入参数为NULL，返回NULL。
- 若str2为空串时总是能匹配成功，因此instr(‘abc’ ， ‘’)会返回1。

示例如下：

```
instr('Tech on the net', 'e') = 2
instr('Tech on the net', 'e', 1, 1) = 2
instr('Tech on the net', 'e', 1, 2) = 11
```

```
instr('Tech on the net', 'e', 1, 3) = 14
```

26.5.5.2.5 IS_ENCODING

函数声明如下：

```
boolean is_encoding(string str, strin from_encoding, string to_encoding)
```

用途：判断输入字符串str是否可以从指定的一个字符集from_encoding转为另一个字符集to_encoding。可用于判断输入是否为“乱码”，通常的用法是将from_encoding设为“utf-8”，to_encoding设为“gbk”。

参数说明：

- str：String类型。若输入参数为NULL，返回NULL。空字符串则可以被认为属于任何字符集。
- from_encoding，to_encoding：String类型。源及目标字符集。若输入参数为NULL，返回NULL。

返回值：Boolean类型，如果str能够成功转换，则返回true，否则返回false。

示例如下：

```
is_encoding('测试', 'utf-8', 'gbk') = true
is_encoding('測試', 'utf-8', 'gbk') = true
-- gbk字库中有这两个繁体字
is_encoding('測試', 'utf-8', 'gb2312') = false
-- gb2312库中不包括这两个字
```

26.5.5.2.6 KEYVALUE

函数声明如下：

```
KEYVALUE(STRING srcStr，STRING split1，STRING split2， STRING key)
KEYVALUE(STRING srcStr， STRING key) //split1 = ";"，split2 = ":"
```

用途：将srcStr（源字符串）按split1分成“key-value”对，按split2将key-value对分开，返回“key”所对应的value 。

参数说明：

- srcStr输入需要拆分的字符串。
- key：String类型。源字符串按照split1和split2拆分后，根据该key值的指定，返回其对应的value。

- split1，split2：用来作为分隔符的字符串，按照指定的这两个分隔符拆分源字符串。如果表达式中没有指定这两项，默认split1为’;’ ，split2为’:’ 。当某个被split1拆分后的字符串中有多个split2时，返回结果未定义。

返回值：String 类型。

- Split1或split2为NULL时，返回NULL。
- srcStr，key为NULL或者没有匹配的key时，返回NULL。
- 如果有多个key-value匹配，返回第一个匹配上的key对应的value。

示例如下：

```
keyvalue('0:1\;1:2', 1) = '2'
--源字符串为“0:1\;1:2” ，因为没有指定split1和split2，默认split1为";"，split2为 “:” 。经过split1拆分后，key-value对为：
0:1\,1:2
经过split2拆分后变成：
0 1/
1 2
返回key为1所对应的value值，为2。
keyvalue("\;decreaseStore:1\;xcard:1\;isB2C:1\;tf:21910\;cart:1\;shipping:2\;pf:0\;market:shoes\;instPayAmount:0\;", "\;",":","tf") = "21910"
--源字符串
为 “\;decreaseStore:1\;xcard:1\;isB2C:1\;tf:21910\;cart:1\;shipping:2\;pf:0\;market:shoes\;instPayAmount:0\;” ，按照 split1 “\;” 拆分后，得出的key-value对为：
decreaseStore:1，xcard:1，isB2C:1，tf:21910，cart:1，shipping:2，pf:0，market:shoes，instPayAmount:0

按照split2":"拆分后变成：
decreaseStore 1
xcard 1
isB2C 1
tf 21910
cart 1
shipping 2
pf 0
market shoes
instPayAmount 0
key值为 “tf” ,返回其对应的value:21910。
```

26.5.5.2.7 LENGTH

函数声明如下：

```
bigint length(string str)
```

用途：返回字符串str的长度。

参数说明：

str：String类型。若输入为bigint类型，decimal类型，double类型或datetime类型会隐式转换为string类型后参与运算，为其他类型时报错。

返回值：Bigint类型。若str为NULL ，返回NULL。如果str为非UTF-8编码格式，返回-1。

示例如下：

```
length('hi! 中国') = 6
```

26.5.5.2.8 LENGTHB

函数声明如下：

```
bigint lengthb(string str)
```

用途：返回字符串str中以字节为单位的长度。

参数说明：

str：String类型。若输入为bigint类型，double类型，decimal类型或datetime类型会隐式转换为string类型后参与运算，为其他类型时报错。

返回值：Bigint类型。若str为NULL，返回NULL。

示例如下：

```
lengthb('hi! 中国') = 10
```

26.5.5.2.9 MD5

函数声明如下：

```
string md5(string value)
```

用途：计算输入字符串value的md5值。

参数说明：

value：String类型。若输入类型是bigint类型，decimal类型，double类型或datetime会隐式转换成string类型参与运算，为其他类型时报错。

返回值：String类型。若输入参数为NULL，返回NULL。

26.5.5.2.10 PARSE_URL

函数声明如下：

```
STRING PARSE_URL(STRING url, STRING part[,STRING key])
```

用途：对url的解析，按key提取信息。

参数说明：

- url或part为NULL，则返回NULL，url为无效报错。
- part：String类型。支持HOST， PATH， QUERY， REF， PROTOCOL，AUTHORITY，FILE和USERINFO，不区分大小写，不在此范围报错。
- 当part为QUERY时，根据key的值取出在query string中的value值，否则忽略key参数。

返回值：String类型。

示例如下：

```
url = file://username:password@example.com:8042/over/there/index.dtb?type=animal&name=narwhal#nose
parse_url('url', 'HOST') = "example.com"
parse_url('url', 'PATH') = "/over/there/index.dtb"
parse_url('url', 'QUERY') = "type=animal&name=narwhal"
parse_url('url', 'QUERY', 'name') = "narwhal"
parse_url('url', 'REF') = "nose"
parse_url('url', 'PROTOCOL') = "file"
parse_url('url', 'AUTHORITY') = "username:password@example.com:8042"
parse_url('url', 'FILE') = "/over/there/index.dtb?type=animal&name=narwhal"
parse_url('url', 'USERINFO') = "username:password"
```

26.5.5.2.11 REGEXP_EXTRACT

函数声明如下：

```
string regexp_extract(string source, string pattern[, bigint occurrence])
```

用途：将字符串source按照pattern正则表达式的规则拆分，返回第occurrence个group的字符。

参数说明：

- source：String类型。需要搜索的字符串。
- pattern：String类型常量。pattern为空串时报错，pattern中如果没有指定group，报错。
- occurrence：Bigint类型常量。必须 >=0，为其他类型或小于0时报错，不指定时默认为1，表示返回第一个group。若occurrence = 0，返回满足整个pattern 的子串。

返回值：String类型。若任意一个输入参数为NULL，返回NULL。

示例如下：

```
regexp_extract('foothebar', 'foo(.*?)(bar)', 1) = the
regexp_extract('foothebar', 'foo(.*?)(bar)', 2) = bar
regexp_extract('foothebar', 'foo(.*?)(bar)', 0) = foothebar
regexp_extract('8d99d8', '8d(\\d+)d8') = 99
-- 如果是在 MaxCompute 客户端上提交正则计算的SQL，需要使用两个"\"作为转移字符
regexp_extract('foothebar', 'foothebar')
```

```
-- 报错，pattern中没有指定group
```

26.5.5.2.12 REGEXP_INSTR

函数声明如下：

```
bigint regexp_instr(string source, string pattern[,bigint start_position[, bigint nth_occurrence[, bigint return_option]])
```

用途：返回字符串source从start_position开始，和pattern第n次（nth_occurrence）匹配的子串的起始/结束位置。

参数说明：

- source：String类型。待搜索的字符串。
- pattern：String类型常量。pattern为空串时报错。
- start_position：Bigint类型常量。搜索的开始位置，不指定时默认值为1，为其他类型或小于等于0的值会报错。
- nth_occurrence：Bigint类型常量。不指定时默认值为1，表示搜索第一次出现的位置。为其他类型或小于等于0的值会报错。
- return_option：Bigint类型常量。值为0或1，为其他类型或不允许的值会报错。0表示返回匹配的开始位置，1表示返回匹配的结束位置。

返回值：Bigint类型。是return_option指定的类型返回匹配的子串在source中的开始或结束位置。若任意一个输入参数为NULL，返回NULL。

示例如下：

```
regexp_instr("i love www.taobao.com", "o[[:alpha:]]{1}", 3, 2) = 14
```

26.5.5.2.13 REGEXP_SUBSTR

函数声明如下：

```
string regexp_substr(string source, string pattern[, bigint start_position[, bigint nth_occurrence]])
```

用途：从start_position位置开始，source中第nth_occurrence次匹配指定模式pattern的子串。

参数说明：

- source：String类型。搜索的字符串。
- pattern：String类型常量。要匹配的模型，pattern为空串时报错。

- start_position：Bigint常量。必须大于0。为其他类型或小于等于0时报错，不指定时默认为1，表示从source的第一个字符开始匹配。
- nth_occurrence：Bigint常量。必须大于0，为其他类型或小于等于0时报错，不指定时默认为1，表示返回第一次匹配的子串。

返回值：String类型。若任意一个输入参数为NULL，返回NULL。没有匹配时返回NULL。

示例如下：

```
regexp_substr ("I love aliyun very much", "a[[:alpha:]]{5}") = "aliyun"
regexp_substr('I have 2 apples and 100 bucks!', '[[:blank:]][[:alnum:]]*', 1, 1) = " have"
regexp_substr('I have 2 apples and 100 bucks!', '[[:blank:]][[:alnum:]]*', 1, 2) = " 2"
```

26.5.5.2.14 REGEXP_COUNT

函数声明如下：

```
bigint regexp_count(string source, string pattern[, bigint start_position])
```

用途：计算source中从start_position开始，匹配指定模式pattern的子串的次数。

参数说明：

- source：String类型。搜索的字符串，为其他类型时报错。
- pattern：String类型常量。要匹配的模型，pattern为空串时报错，为其他类型时报错。
- start_position：Bigint类型常量。必须大于0。为其他类型或小于等于0时报错，不指定时默认为1，表示从source的第一个字符开始匹配。

返回值：Bigint类型。若任意一个输入参数为NULL，返回NULL。没有匹配时返回0。

示例如下：

```
regexp_count('abababc', 'a.c') = 1
regexp_count('abcde', '[[:alpha:]]{2}', 3) = 1
```

26.5.5.2.15 SPLIT_PART

函数声明如下：

```
string split_part(string str, string separator, bigint start[, bigint end])
```

用途：依照分隔符separator拆分字符串str，返回从第start部分到第end部分的子串（闭区间）。

参数说明：

- Str：String类型。要拆分的字符串。如果是bigint类型，decimal类型，double类型或datetime类型会隐式转换到string类型后参加运算，为其他类型时报错。
- Separator：String类型常量。拆分用的分隔符，可以是一个字符，也可以是一个字符串，为其他类型时报错。
- start：Bigint类型常量。必须大于0。非常量为其他类型报错。返回段的开始编号（从1开始），如果没有指定end，则返回start指定的片段。
- end：Bigint类型常量。大于等于start，否则报错。返回片段的截止编号，非常量或为其他类型会报错。可省略，缺省时表示最后一部分。

返回值：String类型。若任意一个输入参数为NULL，返回NULL。若separator为空串，返回原字符串str。



说明:

- 如果separator不存在于str中，且start指定为1，返回整个str。若输入为空串，输出为空串。
- 如果start的值大于切分后实际的分段数，例如：字符串拆分完有6个片段，但start大于 6，返回空串。
- 若 end 大于片段个数，按片段个数处理。

示例如下：

```
split_part('a,b,c,d', ',', 1) = 'a'
split_part('a,b,c,d', ',', 1, 2) = 'a,b'
split_part('a,b,c,d', ',', 10) = ''
```

26.5.5.2.16 REGEXP_REPLACE

函数声明如下：

```
string regexp_replace(string source, string pattern, string replace_string[, bigint occurrence])
```

用途：将source字符串中第occurrence次匹配pattern的子串替换成指定字符串replace_string后返回。

参数说明：

- source：String类型。要替换的字符串。
- pattern：String类型常量。要匹配的模式，pattern为空串时报错。
- replace_string：String类型。将匹配的pattern替换成的字符串。

- occurrence：Bigint类型常量。必须大于等于0，表示将第几次匹配替换成replace_string，为0时表示替换掉所有的匹配子串。为其他类型或小于0报错。可缺省，默认值为0。

返回值：String类型，当引用不存在的组时，不进行替换。当输入source，pattern，occurrence参数为NULL时，返回NULL。若replace_string为NULL且pattern匹配，返回NULL；replace_string为NULL但pattern不匹配，则返回原字符串。



说明: 当引用不存在的组时，行为未定义。

示例如下：

```
regexp_replace("123.456.7890", "([[:digit:]]{3})\\.([[:digit:]]{3})\\.([[:digit:]]{4})", "(\\1)\\2-\\3", 0) =
 "(123)456-7890"
regexp_replace("abcd", "(.)", "\\1 ", 0) = "a b c d "
regexp_replace("abcd", "(.)", "\\1 ", 1) = "a bcd"
regexp_replace("abcd", "(.)", "\\2", 1) = "abcd"
-- 因为pattern中只定义了一个组，引用的第二个组不存在，
-- 请避免这样使用，引用不存在的组的结果未定义。
regexp_replace("abcd", "(.*)(.)$", "\\2", 0) = "d"
regexp_replace("abcd", "a", "\\1", 0) = "bcd"
-- 因为在pattern中没有组的定义，所以\1引用了不存在的组，
-- 请避免这样使用，引用不存在的组的结果未定义。
```

26.5.5.2.17 SUBSTR

函数声明如下：

```
string substr(string str, bigint start_position[, bigint length])
```

用途：返回字符串str从start_position开始长度为length的子串。

参数说明：

- str：String类型。若输入为bigint类型，decimal类型，double类型或datetime类型会隐式转换为string类型后参与运算，为其他类型时报错。
- start_position：Bigint类型。起始位置为1。当start_position为负数时表示开始位置是从字符串的结尾往前倒数，最后一个字符是-1，为其他类型时报错。
- length：Bigint类型。子串的长度，大于0，为其他类型或小于等于0报错。

返回值：String类型。若任意一个输入参数为NULL，返回NULL。



说明: 当length被省略时，返回到str结尾的子串。

示例如下：

```
substr("abc", 2) = "bc"
substr("abc", 2, 1) = "b"
substr("abc",-2,2) = "bc"
substr("abc",-3) = "abc"
```

26.5.5.2.18 TOLOWER

函数声明如下：

```
string tolower(string source)
```

用途：输出英文字符串source对应的小写字符串。

参数说明：

source：String类型。若输入为bigint类型，decimal类型，double类型或datetime类型会隐式转换为string类型后参与运算，为其他类型时报错。

返回值：String类型。若输入参数为NULL，返回NULL。

示例如下：

```
tolower("aBcd") = "abcd"
tolower("哈哈Cd") = "哈哈cd"
```

26.5.5.2.19 TOUPPER

函数声明如下：

```
string toupper(string source)
```

用途：输出英文字符source串对应的大写字符串。

参数说明：

source：String类型。若输入为bigint类型，decimal类型，double类型或datetime类型会隐式转换为string类型后参与运算，为其他类型时报错。

返回值：String类型。若输入参数为NULL，返回NULL。

示例如下：

```
toupper("aBcd") = "ABCD"
```

```
toupper("哈哈Cd") = "哈哈CD"
```

26.5.5.2.20 TO_CHAR

函数声明如下：

```
string to_char(boolean value)
string to_char(bigint value)
string to_char(double value)
string to_char(decimal value)
```

用途：将Boolean类型，bigint类型，decimal类型或double类型转为对应的string类型表示。

参数说明：

value：可以接受boolean类型，bigint类型或double类型输入，为其他类型时报错。对datetime类型的格式化输出请参见： 日期处理函数—*TO_CHAR* 。

返回值：String类型。若输入参数为NULL，返回NULL。

示例如下：

```
to_char(123) = '123'
to_char(true) = 'TRUE'
to_char(1.23) = '1.23'
to_char(null) = 'null'
```

26.5.5.2.21 TRIM

函数声明如下：

```
string trim(string str)
```

用途：将输入字符串str的左右空格去除。

参数说明：

str：String类型。若输入为bigint类型，decimal类型，double类型或datetime类型会隐式转换为string类型后参与运算，为其他类型时报错。

返回值：String类型。若输入参数为NULL，返回NULL。

26.5.5.2.22 LTRIM

函数声明如下：

```
string ltrim(string str)
```

用途：将输入的字符串str的左边空格去除。

参数类型：

str：String类型。若输入为bigint类型，decimal类型，double类型或datetime类型会隐式转换为string类型后参与运算，为其他类型时报错。

返回值：String类型。若输入参数为NULL，返回NULL。

示例如下：

```
select ltrim(' abc ') from dual;
返回结果：
+-----+
| _c0 |
+-----+
| abc  |
+-----+
```

26.5.5.2.23 RTRIM

函数声明如下：

```
string rtrim(string str)
```

用途：将输入的字符串str的右边空格去除。

参数类型：

str：String类型。若输入为bigint类型，decimal类型，double类型或datetime类型会隐式转换为string类型后参与运算，为其他类型时报错。

返回值：String类型。若输入参数为NULL，返回NULL。

示例如下：

```
select rtrim('a abc ') from dual;
返回结果：
+-----+
| _c0 |
+-----+
| a abc |
+-----+
```

26.5.5.2.24 REVERSE

函数声明如下：

```
STRING REVERSE(string str)
```

用途：返回倒序字符串。

参数类型：

str：String类型。若输入为bigint类型，decimal类型，double类型或datetime类型会隐式转换为string类型后参与运算，为其他类型时报错。

返回值：String类型。若输入参数为NULL，返回NULL。

示例如下：

```
select reverse('abcedfg') from dual;
返回结果：
+-----+
| _c0 |
+-----+
| gfdecba |
+-----+
```

26.5.5.2.25 SPACE

函数声明如下：

```
STRING SPACE(bigint n)
```

用途：空格字符串函数，返回长度为n的字符串。

参数类型：

n：Bigint类型。长度不超过2M。如果为空，报错。

返回值：String类型。

示例如下：

```
select length(space(10)) from dual;
-- 返回10。
select space(400000000000) from dual；
-- 报错，长度超过2M。
```

26.5.5.2.26 REPEAT

函数声明如下：

```
STRING REPEAT(string str, bigint n)
```

用途：返回重复n次后的str字符串。

参数类型：

- str：String类型。若输入为bigint类型，decimal类型，double类型或datetime类型会隐式转换为string类型后参与运算，为其他类型时报错。

- n：Bigint类型。长度不超过2M。如果为空，报错。

返回值：String类型。

示例如下：

```
select repeat('abc',5) from lxw_dual;
-- 返回：abcabcabcabcabc
```

26.5.5.2.27 ASCII

函数声明如下：

```
Bigint ASCII(string str)
```

用途：返回字符串str第一个字符的ascii码。

参数类型：

str：String类型。若输入为bigint类型，decimal类型，double类型或datetime类型会隐式转换为string类型后参与运算，为其他类型时报错。

返回值：Bigint类型。

示例如下：

```
select ascii('abcde') from dual;
-- 返回结果：97
```

26.5.5.2.28 URL_ENCODE

函数声明如下：

```
STRING URL_ENCODE(STRING input[, STRING encoding])
```

用途：将输入字符串编码为application/x-www-form-urlencoded MIME格式：

- a-z, A-Z 保持不变。
- “.”，“-”，“*”，“_”保持不变。
- 空格转为”+”。
- 其余字符根据指定的encoding转为字节值， encoding不输入默认为UTF-8，然后将每个字节值表示为%xy的格式，xy是该字符值的十六进制表示方式。

参数说明：

- input：要输入的字符串。

- encoding：指定的编码格式，不输入默认为UTF-8。

返回值：String类型。若输入参数为NULL，返回NULL。

示例如下：

```
url_encode('示例for url_encode:// (fdsf)') = "%E7%A4%BA%E4%BE%8Bfor+url_encode%3A%2F%2F+%28fdsf%29"
url_encode('Exaple for url_encode :// dsf(fasfs)', 'GBK') = "Exaple+for+url_encode+%3A%2F%2F+dsf%28fasfs%29"
```

26.5.5.2.29 URL_DECODE

函数声明如下：

```
STRING URL_DECODE(STRING input[, STRING encoding])
```

用途：将输入字符串从application/x-www-form-urlencoded MIME格式转为正常字符串，是url_encoding的逆过程：

- a-z, A-Z保持不变.
- “.”，“-”，“*”，“_”保持不变。
- “+”转为空格。
- %xy格式的序列转为对应的字节值，连续的字节值根据输入的encoding名称解成对应的字符串。
- 其余的字符保持不变。
- 函数最终的返回值是UTF-8编码的字符串。

参数说明：

- input：要输入的字符串。
- encoding：指定的编码格式，不输入默认为UTF-8。

返回值：String类型。若输入参数为NULL，返回NULL。

示例如下：

```
url_decode('%E7%A4%BA%E4%BE%8Bfor+url_encode%3A%2F%2F+%28fdsf%29')= "示例for url_encode:// (fdsf)"
```

```
url_decode('Exaple+for+url_encode+%3A%2F%2F+dsf%28fasfs%29', 'GBK') = "Exaple for url_encode :// dsf(fasfs)" ```
```

26.5.5.2.30 新扩展字符串函数说明

升级到MaxCompute2.0以后，产品扩展了部分字符串函数，其中**LPAD函数**和**RPAD函数**在使用函数的sql前，需要添加如下的set语句。

```
set odps.sql.type.system.odps2=true;
```

本章节后介绍的字符串函数即为新扩展的字符串函数。

26.5.5.2.31 CONCAT_WS

函数声明如下：

```
string concat_ws(string SEP, string a, string b...)
```

用途：返回值是将参数中的所有字符串安装指定的分隔符连接在一起的结果。

参数说明：

- SEP：String类型。分隔符，若不指定，返回的值报错。
- a，b...：String类型。若输入为bigint类型，decimal类型，double类型或datetime类型会隐式转换为string类型后参与运算，为其他类型时报错。

返回值：String类型。若没有输入参数或者某个输入参数为NULL，结果均返回NULL。

示例如下：

```
concat_ws(':','name','hanmeimei') = 'name:hanmeimei'
concat_ws(':','avg',null,'34')= 'null'
```

26.5.5.2.32 LPAD

函数声明如下：

```
string lpad(string a, int len, string b)
```

用途：用b字符串将a字符串向左补足到len位。

参数说明：

- len：Int整型。
- a，b：String类型。

返回值：String类型。若len小于a的位数，则返回a从左开始截取len位字符；若len为0则返回空。

示例如下：

```
lpad('abcdefgh',10,'12')='12abcdefgh'
lpad('abcdefgh',5,'12')='abcde'
lpad('abcdefgh',0,'12')
-- 返回空
```

26.5.5.2.33 RPAD

函数声明如下：

```
string rpad(string a, int len, string b)
```

用途：用b字符串将a字符串向右补足到len位。

参数说明：

- len：Int整型。
- a，b：String类型。

返回值：String类型。若len小于a的位数，则返回a从左开始截取len位字符；若len为0则返回空。

示例如下：

```
rpad('abcdefgh',10,'12')='abcdefgh12'
rpad('abcdefgh',5,'12')='abcde'
rpad('abcdefgh',0,'12')
-- 返回空
```

26.5.5.2.34 REPLACE

函数声明如下：

```
string replace(string a, string OLD, string NEW)
```

用途：用NEW字符串替换a字符串中与OLD字符串完全重合的部分并返回a。

参数说明：

所有参数均为string类型。

返回值：String类型。若任意一个输入参数为NULL，返回NULL。

示例如下：

```
replace('abababab','abab','12') = '12ab'
replace('abababab','cdf','123') = 'abababab'
```

```
replace('123abab456ab',null,'abab') = 'null'
```

26.5.5.2.35 SOUNDEX

函数声明如下：

```
string soundex(string a)
```

用途：将普通字符串转换成soundex字符串。

参数说明：

所有参数均为string类型。

返回值：String类型。若输入参数为NULL，返回NULL。

示例如下：

```
soundex('hello') = 'H400'
```

26.5.5.2.36 SUBSTRING_INDEX

函数声明如下：

```
string substring_index(string a, string SEP, int count))
```

用途：截取字符串a第count分隔符之前的字符串，如count为正则从左边开始截取，如果为负则从右边开始截取。

参数说明：

- a，sep：String类型。
- count：Int类型。

返回值：String类型。若输入参数为NULL，返回NULL。

示例如下：

```
substring_index('https://help.aliyun.com', '.', 2) = 'https://help.aliyun'
substring_index('https://help.aliyun.com', '.', -2) = 'aliyun.com'
```

```
substring_index('https://help.aliyun.com', null, 2) = 'null'
```

26.5.5.3 日期处理函数

26.5.5.3.1 DATEADD

函数声明如下：

```
datetime dateadd(datetime date, bigint delta, string datepart)
```

用途：按照指定的单位datepart和幅度delta修改date的值。

参数说明：

- date：Datetime类型，日期值。若输入为string类型会隐式转换为datetime类型后参与运算，为其他类型时报错。
- delta：Bigint类型，修改幅度。若输入为string类型或double类型会隐式转换为bigint类型后参与运算，为其他类型时报错。若delta大于 0，修改幅度为加；否则为减。
- datepart：String类型常量。此字段的取值遵循string类型与datetime类型转换的约定，即**yyyy**表示年，**mm**表示月。关于类型转换的规则请参考：*String类型与Datetime类型之间的转换*。此外也支持扩展的日期格式：年**year**，月**month**或**mon**，日**day**，小时**hour**。非常量、不支持的格式或为其他类型时报错。

返回值：Datetime类型。若任意一个输入参数为NULL，返回NULL。

说明:

- 按照指定的单位增减delta时导致的对更高单位的进位或退位，年、月、时、分、秒分别按照10进制、12进制、24进制、60进制计算。当delta的单位是月时，计算规则如下：若datetime的月部分在增加delta值后不造成 day溢出，则保持day值不变，否则把day值设置为结果月份的最后一天。
- datepart的取值遵循string类型与datetime类型转换的约定，即**yyyy**表示年，**mm**表示月， datetime相关的内建函数如无特殊说明均遵守此约定。 同时，如果没有特殊说明，所有datetime相关的内建函数的part部分也同样支持扩展的日期格式：年**year**，月**month**或**mon**，日**day**，小时**hour**。

示例如下：

```
若trans_date = 2005-02-28 00:00:00：
dateadd(trans_date, 1, 'dd') = 2005-03-01 00:00:00
-- 加一天，结果超出当年2月份的最后一天，实际值为下个月的第一天
dateadd(trans_date, -1, 'dd') = 2005-02-27 00:00:00
-- 减一天
```

```
dateadd(trans_date, 20, 'mm') = 2006-10-28 00:00:00
-- 加20个月，月份溢出，年份加1
若trans_date = 2005-02-28 00:00:00, dateadd(transdate, 1, 'mm') = 2005-03-28 00:00:00
若trans_date = 2005-01-29 00:00:00, dateadd(transdate, 1, 'mm') = 2005-02-28 00:00:00
-- 2005年2月没有29日，日期截取至当月最后一天，
若trans_date = 2005-03-30 00:00:00, dateadd(transdate, -1, 'mm') = 2005-02-28 00:00:00
```

此处对trans_date的数值表示仅作示例使用，在文档中有关datetime的介绍会经常使用到这种简易的表达方式。在MaxCompute SQL中，datetime类型没有直接的常数表示方式，如下使用方式是错误的：

```
select dateadd(2005-03-30 00:00:00, -1, 'mm') from tbl1;
```

如果一定要描述datetime类型常量，请尝试如下方法：

```
select dateadd(cast("2005-03-30 00:00:00" as datetime), -1, 'mm') from tbl1;
-- 将String类型常量显式转换为Datetime类型
```

26.5.5.3.2 DATEDIFF

函数声明如下：

```
bigint datediff(datetime date1, datetime date2, string datepart)
```

用途：计算两个时间date1，date2在指定时间单位datepart的差值。

参数说明：

- datet1，date2：Datetime类型，被减数和减数。若输入为string类型会隐式转换为datetime类型后参与运算，为其他类型时报错。
- datepart：String类型常量，支持扩展的日期格式。若datepart不符合指定格式或者为其他类型，则会报错。

返回值：Bigint类型。若任意一个输入参数为NULL，返回NULL。若date1小于date2，返回值可以为负数。



说明: 计算时会按照datepart切掉低单位部分，然后再计算结果。

示例如下：

```
若start = 2005-12-31 23:59:59，end = 2006-01-01 00:00:00:
datediff(end, start, 'dd') = 1
datediff(end, start, 'mm') = 1
datediff(end, start, 'yyyy') = 1
datediff(end, start, 'hh') = 1
datediff(end, start, 'mi') = 1
datediff(end, start, 'ss') = 1
```

```
datediff('2013-05-31 13:00:00', '2013-05-31 12:30:00', 'ss') = 1800
datediff('2013-05-31 13:00:00', '2013-05-31 12:30:00', 'mi') = 30
```

26.5.5.3.3 DATEPART

函数声明如下：

```
bigint datepart(datetime date, string datepart)
```

用途：提取日期date中指定的时间单位datepart的值。

参数说明：

- date：Datetime类型。若输入为string类型会隐式转换为datetime类型后参与运算，为其他类型时报错。
- datepart：String类型常量，支持扩展的日期格式。若datepart不符合指定格式或者为其他类型，则会报错。

返回值：Bigint类型。若任意一个输入参数为NULL，返回NULL。

示例如下：

```
datepart('2013-06-08 01:10:00', 'yyyy') = 2013
datepart('2013-06-08 01:10:00', 'mm') = 6
```

26.5.5.3.4 DATETRUNC

函数声明如下：

```
datetime datetrunc (datetime date,string datepart)
```

用途：返回日期date被截取指定时间单位datepart后的日期值。

参数说明：

- date：Datetime类型。若输入为string类型会隐式转换为datetime类型后参与运算，为其他类型时报错。
- datepart：String类型常量，支持扩展的日期格式。若datepart不符合指定格式或者为其他类型，则会报错。

返回值：Datetime类型。若任意一个输入参数为NULL，返回NULL。

示例如下：

```
datetrunc('2011-12-07 16:28:46', 'yyyy') = 2011-01-01 00:00:00
datetrunc('2011-12-07 16:28:46', 'month') = 2011-12-01 00:00:00
```

```
datetrunc('2011-12-07 16:28:46', 'DD') = 2011-12-07 00:00:00
```

26.5.5.3.5 FROM_UNIXTIME

函数声明如下：

```
datetime from_unixtime(bigint unixtime)
```

用途：将数字型的unix时间日期值unixtime转为日期值。

参数说明：

unixtime：Bigint类型，秒数，unix格式的日期时间值。若输入为string类型，decimal类型或double类型会隐式转换为bigint类型后参与运算。

返回值：Datetime类型的日期值。unixtime为NULL时，返回NULL。

示例如下：

```
from_unixtime(123456789) = 1973-11-30 05:33:09;
```

26.5.5.3.6 GETDATE

函数声明如下：

```
datetime getdate()
```

用途：获取当前系统时间。使用东八区时间作为MaxCompute标准时间。

返回值：返回当前日期和时间，datetime类型。



说明: 在一个MaxCompute SQL任务中（以分布式方式执行），getdate总是返回一个固定的值。返回结果会是MaxCompute SQL执行期间的任意时间，时间精度精确到秒（后续版本将精确到毫秒）。

26.5.5.3.7 ISDATE

函数声明如下：

```
boolean isdate(string date, string  format)
```

用途：判断一个日期字符串能否根据对应的格式串转换为一个日期值，如果转换成功返回TRUE，否则返回FALSE。

参数说明：

- date：String类型，日期值。若输入为bigint类型，decimal类型，double类型或datetime类型会隐式转换为string类型后参与运算，为其他类型时报错。
- format：String类型常量，不支持日期扩展格式。若为其它类型或不支持的格式，则会报错。如果format中出现多余的格式串，则只取第一个格式串对应的日期数值，其余的会被视为分隔符。如isdate(“1234-yyyy “, “yyyy-yyyy “)，会返回TRUE。

返回值：Boolean类型。若任意一个输入参数为NULL，返回NULL。

26.5.5.3.8 LASTDAY

函数声明如下：

```
datetime lastday(datetime date)
```

用途：取date当月的最后一天，截取到天，时分秒部分为00:00:00。

参数说明：

date：Datetime类型。若输入为string类型会隐式转换为datetime类型后参与运算，为其他类型时报错。

返回值：Datetime类型。若输入参数为NULL，返回NULL。

26.5.5.3.9 TO_DATE

函数声明如下：

```
datetime to_date(string date, string format)
```

用途：将一个format格式的字符串date转成日期值。

参数说明：

- date：String类型，要转换的字符串格式的日期值。若输入为bigint类型，decimal类型，double类型或datetime类型会隐式转换为String类型后参与运算， 为空串或为其他类型时报错。
- format：String 类型常量，日期格式。非常量或为其他类型会报错。format不支持日期扩展格式，其他字符作为无用字符在解析时忽略。 format参数至少包含**yyyy**，否则报错。如果format中出现多余的格式串，则只取第一个格式串对应的日期数值，其余的会被视为分隔符。如to_date(‘1234-2234 ‘, ‘yyyy-yyyy ‘)会返回1234-01-01 00:00:00。

返回值：Datetime类型。格式为*yyyy-mm-dd hh:mi:ss*。若任意一个输入参数为NULL，返回NULL。

示例如下：

```
to_date('阿里巴巴2010-12*03', '阿里巴巴yyyy-mm*dd') = 2010-12-03 00:00:00
to_date('20080718', 'yyyymmdd') = 2008-07-18 00:00:00
to_date('200807182030','yyyymmddhhmi')=2008-07-18 20:30:00
to_date('2008718', 'yyyymmdd')
-- 格式不符合，返回NULL
to_date('阿里巴巴2010-12*3', '阿里巴巴yyyy-mm*dd')
-- 格式不符合，返回NULL
to_date('2010-24-01', 'yyyy')
-- 格式不符合，返回NULL
```

26.5.5.3.10 TO_CHAR

函数声明如下：

```
string to_char(datetime date, string format)
```

用途：将日期类型date按照format指定的格式转成字符串。

参数类型：

- date：Datetime类型，要转换的日期值。若输入为string类型会隐式转换为datetime类型后参与运算，为其他类型时报错。
- format：String类型常量。非常量或为其他类型会报错。format中的日期格式部分会被替换成相应的数据，其他字符直接输出。

返回值：String类型。若任意一个输入参数为NULL，返回NULL。

示例如下：

```
to_char('2010-12-03 00:00:00', '阿里金融yyyy-mm*dd') = '阿里金融2010-12*03'
to_char('2008-07-18 00:00:00', 'yyyymmdd') = '20080718'
to_char('阿里巴巴2010-12*3', '阿里巴巴yyyy-mm*dd')
-- 返回NULL
to_char('2010-24-01', 'yyyy')
-- 返回NULL
to_char('2008718', 'yyyymmdd')
-- 返回NULL
```



说明: 关于其他类型向string类型转换请参考：字符串函数*TO_CHAR* 。

26.5.5.3.11 UNIX_TIMESTAMP

函数声明如下：

```
bigint unix_timestamp(datetime date)
```

用途：将日期date转化为整型的unix格式的日期时间值。

参数说明：

date：Datetime类型，日期值。若输入为string类型会隐式转换为datetime类型后参与运算，为其他类型时报错。

返回值：Bigint类型。表示unix格式日期值，date为NULL时，返回NULL。

26.5.5.3.12 WEEKDAY

函数声明如下：

```
bigint weekday (datetime date)
```

用途：返回date日期当前周的第几天。

参数说明：

date：Datetime类型。若输入为string类型会隐式转换为datetime类型后参与运算，为其他类型时报错。

返回值：Bigint类型。若输入参数为NULL，返回NULL。周一作为一周的第一天，返回值为0。其他日期依次递增，周日返回6。

26.5.5.3.13 WEEKOFYEAR

函数声明如下：

```
bigint weekofyear(datetime date)
```

用途：返回日期date位于那一年的第几周。周一作为一周的第一天。

> **说明:** 如果这一周跨年，关于这一周属于上一年， 还是下一年，主要是看这一周大多数日期（4天以上）在哪一年多。 属于前一年，就是前一年的最后一周。属于后一年，就是后一年的第一周。

参数说明：

date：Datetime类型，日期值。若输入为string类型会隐式转换为datetime类型后参与运算，为其他类型时报错。

返回值：Bigint类型。若输入参数为NULL，返回NULL。

示例如下：

```
select weekofyear(to_date("20141229", "yyyymmdd")) from dual;
返回结果：
+------------+
| _c0        |
+------------+
| 1          |
+------------+
--    虽然20141229属于2014年，但是这一周的大多数日期是在2015年，因此返回结果为1，表示是2015年的第一周。
select weekofyear(to_date("20141231", "yyyymmdd")) from dual；
-- 返回结果为1。
select weekofyear(to_date("20151229", "yyyymmdd")) from dual；
-- 返回结果为53。
```

26.5.5.3.14 新扩展日期函数说明

升级到MaxCompute2.0以后，产品扩展了部分日期函数，在使用新函数的sql前，需要添加如下的set语句。

```
set odps.sql.type.system.odps2=true;
```



说明:

- 上述set语句和新函数的sql语句需要同时提交运行，不要单独先选择set语句执行，再选择sql执行。
- **ADD_MONTHS**，**LAST_DAY**，**NEXT_DAY**，**MONTHS_BETWEEN**四个函数使用sql前，不需要执行上述set语句。

正确使用示例如下。

```
set odps.sql.type.system.odps2=true;
select year('1970-01-01 12:30:00') = 1970 from dual;
```

本章节后介绍的日期函数即为新扩展的日期函数。

26.5.5.3.15 YEAR

函数声明如下：

```
INT year(string date)
```

用途：返回一个日期的年。

参数类型：

date：String类型，日期值。格式至少包含**yyyy-mm-dd**且不含多余的字符串，否则返回NULL。

返回值：Int类型。

示例如下：

```
year('1970-01-01 12:30:00') = 1970
year('1970-01-01') = 1970
year('70-01-01') = 70
year(1970-01-01) = null
year('1970/03/09') = null
year(null) = null
```

26.5.5.3.16 QUARTER

函数声明如下：

```
INT quarter(datetime/timestamp/string date)
```

用途：返回一个日期的季度，范围是1-4。

参数类型：

date：Datetime类型或Timestamp类型或String类型，日期值。格式至少包含**yyyy-mm-dd**且不含多余的字符串，否则返回NULL。

返回值：Int类型。如输入参数为NULL，返回NULL。

示例如下：

```
quarter('1970-11-12 10:00:00') = 4
quarter('1970-11-12') = 4
```

26.5.5.3.17 MONTH

函数声明如下：

```
INT month(string date)
```

用途：返回一个日期的月份。

参数类型：

date：String类型，日期值。为其他类型时报错。

返回值：Int类型。

示例如下：

```
month('2014-09-01') = 9
month('20140901') = null
```

26.5.5.3.18 DAY

函数声明如下：

```
INT day(string date)
```

用途：返回一个日期的天。

参数类型：

date：String类型，日期值。为其他类型时报错。

返回值：Int类型。

示例如下：

```
day('2014-09-01') = 1
day('20140901') = null
```

26.5.5.3.19 DAYOFMONTH

函数声明如下：

```
INT  dayofmonth(date)
```

用途：返回一个日期的天。

参数类型：

date：String类型，日期值。为其他类型时报错。

返回值：Int类型。

示例如下：

```
dayofmonth('2014-09-01') = 1
```

```
dayofmonth('20140901') = null
```

26.5.5.3.20 HOUR

函数声明如下：

```
INT hour(string date)
```

用途：返回一个日期的小时。

参数类型：

date：String类型，日期值。为其他类型时报错。

返回值：Int类型。

示例如下：

```
hour('2014-09-01 12:00:00') = 12
hour('12:00:00') = 12
hour('20140901120000') = null
```

26.5.5.3.21 MINUTE

函数声明如下：

```
INT minute(string date)
```

用途：返回一个日期的分钟。

参数类型：

date：String类型，日期值。为其他类型时报错。

返回值：Int类型。

示例如下：

```
minute('2014-09-01 12:30:00') = 30
minute('12:30:00') = 30
minute('20140901120000') = null
```

26.5.5.3.22 SECOND

函数声明如下：

```
INT second(string date)
```

用途：返回一个日期的秒钟。

参数类型：

date：String类型，日期值。为其他类型时报错。

返回值：Int类型。

示例如下：

```
second('2014-09-01 12:30:45') = 45
second('12:30:45') = 45
second('20140901123045') = null
```

26.5.5.3.23 FROM_UTC_TIMESTAMP

函数声明如下：

```
timestamp from_utc_timestamp({any primitive type}*, string timezone)
```

用途：将一个UTC时区的时间戳转换成一个指定时区的时间戳。

参数类型：

- {any primitive type}*：时间戳，类型包含timestamp/datetime/tinyint/smallint/int/bigin。
- timezone：指定需要转换的目标时区，如PST。

返回值：Timestamp类型。

示例如下：

```
from_utc_timestamp(1501557840,'PST') = '1970-01-19 01:05:57.84'
from_utc_timestamp('1970-01-30 16:00:00','PST') = '1970-01-31 00:00:00'
from_utc_timestamp('1970-01-30','PST') = '1970-01-30 08:00:00'
```

26.5.5.3.24 CURRENT_TIMESTAMP

函数声明如下：

```
timestamp current_timestamp()
```

用途：返回当前timestamp类型的时间戳，值不固定。

返回值：Timestamp类型。

示例如下：

```
select current_timestamp() from dual;
```

```
-- 返回'2017-08-03 11:50:30.661'
```

26.5.5.3.25 ADD_MONTHS

函数声明如下：

```
string add_months(string startdate, int nummonths)
```

用途：返回开始日期startdat增加nummonths个月后的日期。

参数类型：

- startdate：String类型，日期值。格式至少包含**yyyy-mm-dd**的日期，否则返回NULL。
- num_months：Int型数值。

返回值：String类型的日期值，格式为**yyyy-mm-dd**。

示例如下：

```
add_months('2017-02-14',3) = '2017-05-14'
add_months('17-2-14',3) = '0017-05-14'
add_months('2017-02-14 21:30:00',3) = '2017-05-14'
add_months('20170214',3) = null
```

26.5.5.3.26 LAST_DAY

函数声明如下：

```
string last_day(string date)
```

用途：返回该日期所在月份的最后一天日期。

参数类型：

date：String类型。格式为**yyyy-MM-dd HH:mi:ss**或**yyyy-mm-dd**。

返回值：String类型的日期值，格式为**yyyy-mm-dd**。

示例如下：

```
last_day('2017-03-04') = '2017-03-31'
last_day('2017-07-04 11:40:00') = '2017-07-31'
```

```
last_day('20170304') = null
```

26.5.5.3.27 NEXT_DAY

函数声明如下：

```
string next_day(string startdate, string week)
```

用途：返回大于指定日期startdate并且与week相匹配的第一个日期，即下周几的具体日期。

参数类型：

- startdate：String类型。格式为**yyyy-MM-dd HH:mi:ss**或**yyyy-mm-dd**。
- week：String类型。一个星期前2个或3个字母，或者一个星期的全名，如Mo、TUE、FRIDAY。

返回值：String类型的日期值，格式为**yyyy-mm-dd**。

示例如下：

```
next_day('2017-08-01','TU') = '2017-08-08'
next_day('2017-08-01 23:34:00','TU') = '2017-08-08'
next_day('20170801','TU') = null
```

26.5.5.3.28 MONTHS_BETWEEN

函数声明如下：

```
double months_between(datetime/timestamp/string date1, datetime/timestamp/string date2)
```

用途：返回日期date1和date2之间的月数。

参数类型：

- date1：Datetime或Timestamp或String类型。格式为**yyyy-MM-dd HH:mi:ss**或**yyyy-mm-dd**。
- date2：Datetime或Timestamp或String类型。格式为**yyyy-MM-dd HH:mi:ss**或**yyyy-mm-dd**。

返回值：double类型。

- 当date1晚于date2，返回值为正；当date2晚于date1，返回值为负。
- 当date1和date2分别对应两个月的最后一天，返回整数月，否则计算方式为date1-date2的天数除以31天。

示例如下：

```
months_between('1997-02-28 10:30:00', '1996-10-30') = 3.9495967741935485
months_between('1996-10-30','1997-02-28 10:30:00' ) = -3.9495967741935485
```

```
months_between('1996-09-30','1996-12-31') = -3.0
```

26.5.5.4 窗口函数

MaxCompute SQL中可以使用窗口函数进行灵活的分析处理工作，窗口函数只能出现在select子句中，窗口函数中不要嵌套使用窗口函数和聚合函数，窗口函数不可以和同级别的聚合函数一起使用。

目前在一个MaxCompute SQL语句中，最多可以使用5个窗口函数。

窗口函数语法如下：

```
window_func() over (partition by col1, [col2...]
[order by col1 [asc|desc][, col2[asc|desc]...]] windowing_clause)
```

语法说明：

- partition by部分用来指定开窗的列。分区列的值相同的行被视为在同一个窗口内。现阶段，同一窗口内最多包含1亿行数据（建议不超过500万行），否则运行时报错。
- order by用来指定数据在一个窗口内如何排序。
- windowing_clause部分可以用rows指定开窗方式，目前有如下两种方式：
  - **rows between x preceding|following and y preceding|following**：表示窗口范围是从前或后x行到前或后y行；
  - **rows x preceding|following**：表示窗口范围是从前或后第x行到当前行。

    说明:

    - x，y必须为大于等于0的整数常量，限定范围0 ~ 10000，值为0时表示当前行。
    - 必须指定order by才可以用rows方式指定窗口范围 。
    - 并非所有的窗口函数都可以使用rows指定开窗方式，支持这种用法的窗口函数有avg，count，max，min，stddev和sum。

26.5.5.4.1 COUNT

函数声明如下：

```
bigint count([distinct] expr) over(partition by col1[, col2...]
[order by col1 [asc|desc][, col2[asc|desc]...]] [windowing_clause])
```

用途：计算计数值。

参数说明：

- expr：任意类型。当值为NULL时，该行不参与计算。当指定distinct关键字时表示取唯一值的计数值；
- partition by col1[, col2...]：指定开窗口的列；
- order by col1 [asc|desc],col2[asc|desc]：不指定order by时，返回当前窗口内expr的计数值，指定order by时返回结果以指定的顺序排序，并且值为当前窗口内从开始行到当前行的累计计数值。

返回值：Bigint类型。



说明: 当指定distinct关键字时不能写order by。

示例如下：

假设存在表test_src，表中存在bigint类型的列user_id。

```
select user_id,count(user_id) over (partition by user_id) as count from test_src;
+---------+------------+
| user_id |  count     |
+---------+------------+
| 1       | 3          |
| 1       | 3          |
| 1       | 3          |
| 2       | 1          |
| 3       | 1          |
+---------+------------+
+---------+------------+
-- 不指定order by时，返回当前窗口内user_id的计数值
select user_id,count(user_id) over (partition by user_id order by user_id) as count from test_src;
+---------+------------+
| user_id | count      |
+---------+------------+
| 1       | 1          |     -- 窗口起始
| 1       | 2          |     -- 到当前行共计两条记录，返回2
| 1       | 3          |
| 2       | 1          |
| 3       | 1          |
+---------+------------+
-- 指定order by时，返回当前窗口内从开始行到当前行的累计计数值。
```

26.5.5.4.2 AVG

函数声明如下：

```
avg([distinct] expr) over(partition by col1[, col2...]
[order by col1 [asc|desc] [, col2[asc|desc]...]] [windowing_clause])
```

用途：计算平均值。

参数说明：

- distinct：当指定distinct关键字时表示取唯一值的平均值。
- expr：Double类型。若输入为string类型，bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。当值为NULL时，该行不参与计算。 Boolean类型不允许参与计算。
- partition by col1[, col2]...：指定开窗口的列。
- order by col1 [asc|desc], col2[asc|desc]：不指定order by时返回当前窗口内所有值的平均值，指定order by时返回结果以指定的方式排序，并且返回窗口内从开始行到当前行的累计平均值。

返回值：Double类型。

说明: 指明distinct关键字时不能写order by。

26.5.5.4.3 MAX

函数声明如下：

```
max([distinct] expr) over(partition by col1[, col2...]
[order by col1 [asc|desc][, col2[asc|desc]...]] [windowing_clause])
```

用途：计算最大值。

参数说明：

- expr：除Boolean类型外的任意类型。当值为NULL时，该行不参与计算。当指定distinct关键字时表示取唯一值的最大值（指定该参数与否对结果没有影响）。
- partition by col1[, col2...]：指定开窗口的列。
- order by col1 [asc|desc], col2[asc|desc]：不指定order by时，返回当前窗口内的最大值。指定order by时，返回结果以指定的方式排序，并且值为当前窗口内从开始行到当前行的最大值。

返回值：同expr类型。

说明: 当指定distinct关键字时不能写order by。

26.5.5.4.4 MIN

函数声明如下：

```
min([distinct] expr) over(partition by col1[, col2...]
[order by col1 [asc|desc][, col2[asc|desc]...]] [windowing_clause])
```

用途：计算最小值。

参数说明：

- expr：除boolean类型外的任意类型。当值为NULL时，该行不参与计算。当指定distinct关键字时表示取唯一值的最小值（指定该参数与否对结果没有影响）。
- partition by col1[, col2..]：指定开窗口的列。
- order by col1 [asc|desc], col2[asc|desc]：不指定order by时，返回当前窗口内的最小值。指定order by时，返回结果以指定的方式排序，并且值为当前窗口内从开始行到当前行的最小值。

返回值：同expr类型。



说明: 当指定distinct关键字时不能写order by。

26.5.5.4.5 MEDIAN

函数声明如下：

```
double median(double number) over(partition by col1[, col2...])
decimal median(decimal number) over(partition by col1[,col2...])
```

用途：计算中位数。

参数说明：

- number：Double类型或decimal类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。当输入值为NULL时，返回NULL。
- partition by col1[, col2...]：指定开窗口的列。

返回值：Double类型。

26.5.5.4.6 STDDEV

函数声明如下：

```
double stddev([distinct] expr) over(partition by col1[, col2...] [order by col1 [asc|desc][, col2[asc|
desc]...]] [windowing_clause])
decimal stddev([distinct] expr) over(partition by col1[,col2...] [order by col1 [asc|desc][, col2[asc|
desc]...]] [windowi ng_clause])
```

用途：总体标准差。

参数说明：

- expr：Double类型或Decimal类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。当输入值为NULL时，返回NULL。当指定distinct关键字时表示计算唯一值的总体标准差。
- partition by col1[, col2..]：指定开窗口的列。
- order by col1 [asc|desc], col2[asc|desc]：不指定order by时，返回当前窗口内的总体标准差。指定order by时，返回结果以指定的方式排序，并且值为当前窗口内从开始行到当前行的总体标准差。

返回值：输入为Decimal类型时返回Decimal类型，否则返回Double类型。



说明: 当指定distinct关键字时不能写order by。

26.5.5.4.7 STDDEV_SAMP

函数声明如下：

```
double stddev_samp([distinct] expr) over(partition by col1[, col2...] [order by col1 [asc|desc][,
 col2[asc|desc]...]] [windowing_clause])
decimal stddev_samp([distinct] expr) over(partition by col1[,col2...] [order by col1 [asc|desc][,
 col2[asc|desc]...]] [windowing_clause])
```

用途：样本标准差。

参数说明：

- expr：Double类型或Decimal类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。当输入值为NULL时，返回NULL。当指定distinct关键字时表示计算唯一值的样本标准差。
- partition by col1[, col2..]：指定开窗口的列。
- order by col1 [asc|desc], col2[asc|desc]：不指定order by时，返回当前窗口内的样本标准差。指定order by时，返回结果以指定的方式排序，并且值为当前窗口内从开始行到当前行的样本标准差。

返回值：输入为Decimal类型时返回Decimal类型，否则返回Double类型。



说明: 当指定distinct关键字时不能写order by。

26.5.5.4.8 SUM

函数声明如下：

```
sum([distinct] expr) over(partition by col1[, col2...]
[order by col1 [asc|desc][, col2[asc|desc]...]] [windowing_clause])
```

用途：计算汇总值。

参数说明：

- expr：Double类型或Decimal类型或Bigint类型，当输入为string类型时隐式转换为double类型参与运算，为其它类型时报错。当值为NULL时，该行不参与计算。指定distinct关键字时表示计算唯一值的汇总值。
- partition by col1[, col2..]：指定开窗口的列。
- order by col1 [asc|desc], col2[asc|desc]：不指定order by时，返回当前窗口内expr的汇总值。指定order by时， 返回结果以指定的方式排序，并且返回当前窗口从首行至当前行的累计汇总值。

返回值：输入是Bigint类型时，返回Bigint类型；输入为Double类型或String类型时，返回Double 类型。



说明: 当指定distinct关键字时不能写order by。

26.5.5.4.9 DENSE_RANK

函数声明如下：

```
bigint dense_rank() over(partition by col1[, col2...]
order by col1 [asc|desc][, col2[asc|desc]...])
```

用途：计算连续排名。col2相同的行数据获得的排名相同。

参数说明：

- partition by col1[, col2..]：指定开窗口的列。
- order by col1 [asc|desc], col2[asc|desc]：指定排名依据的值。

返回值：Bigint类型。

示例如下：

如表emp中数据如下：

```
| empno | ename | job | mgr | hiredate| sal| comm | deptno |
7369,SMITH,CLERK,7902,1980-12-17 00:00:00,800,,20
```

```
7499,ALLEN,SALESMAN,7698,1981-02-20 00:00:00,1600,300,30
7521,WARD,SALESMAN,7698,1981-02-22 00:00:00,1250,500,30
7566,JONES,MANAGER,7839,1981-04-02 00:00:00,2975,,20
7654,MARTIN,SALESMAN,7698,1981-09-28 00:00:00,1250,1400,30
7698,BLAKE,MANAGER,7839,1981-05-01 00:00:00,2850,,30
7782,CLARK,MANAGER,7839,1981-06-09 00:00:00,2450,,10
7788,SCOTT,ANALYST,7566,1987-04-19 00:00:00,3000,,20
7839,KING,PRESIDENT,,1981-11-17 00:00:00,5000,,10
7844,TURNER,SALESMAN,7698,1981-09-08 00:00:00,1500,0,30
7876,ADAMS,CLERK,7788,1987-05-23 00:00:00,1100,,20
7900,JAMES,CLERK,7698,1981-12-03 00:00:00,950,,30
7902,FORD,ANALYST,7566,1981-12-03 00:00:00,3000,,20
7934,MILLER,CLERK,7782,1982-01-23 00:00:00,1300,,10
7948,JACCKA,CLERK,7782,1981-04-12 00:00:00,5000,,10
7956,WELAN,CLERK,7649,1982-07-20 00:00:00,2450,,10
7956,TEBAGE,CLERK,7748,1982-12-30 00:00:00,1300,,10
```

现在需要将所有职工根据部门分组，每个组内根据SAL做降序排序，获得职工自己组内的序号。

```
SELECT deptno
       , ename
       , sal
       , DENSE_RANK() OVER (PARTITION BY deptno ORDER BY sal DESC) AS nums
-- deptno(部门)作为开窗列，sal（薪水）作为结果返回时需要排序的值。
   FROM emp;
-- 执行结果如下：
+------------+-------+------------+------------+
| deptno     | ename | sal        | nums       |
+------------+-------+------------+------------+
| 10         | JACCKA | 5000.0     | 1          |
| 10         | KING  | 5000.0     | 1          |
| 10         | CLARK | 2450.0     | 2          |
| 10         | WELAN | 2450.0     | 2          |
| 10         | TEBAGE | 1300.0     | 3          |
| 10         | MILLER | 1300.0     | 3          |
| 20         | SCOTT | 3000.0     | 1          |
| 20         | FORD  | 3000.0     | 1          |
| 20         | JONES | 2975.0     | 2          |
| 20         | ADAMS | 1100.0     | 3          |
| 20         | SMITH | 800.0      | 4          |
| 30         | BLAKE | 2850.0     | 1          |
| 30         | ALLEN | 1600.0     | 2          |
| 30         | TURNER | 1500.0     | 3          |
| 30         | MARTIN | 1250.0     | 4          |
| 30         | WARD  | 1250.0     | 4          |
| 30         | JAMES | 950.0      | 5          |
+------------+-------+------------+------------+
```

26.5.5.4.10 RANK

函数声明如下：

```
bigint rank() over(partition by col1[, col2...] order by col1 [asc|desc][, col2[asc|desc]...])
```

用途：计算排名。col2相同的行数据获得排名顺序下降。

参数说明：

- partition by col2[, col2..]：指定开窗口的列。
- order by col1 [asc|desc], col2[asc|desc]：指定排名依据的值。

返回值：Bigint类型。

示例如下：

如表emp中数据如下：

```
| empno | ename | job | mgr | hiredate| sal| comm | deptno |
7369,SMITH,CLERK,7902,1980-12-17 00:00:00,800,,20
7499,ALLEN,SALESMAN,7698,1981-02-20 00:00:00,1600,300,30
7521,WARD,SALESMAN,7698,1981-02-22 00:00:00,1250,500,30
7566,JONES,MANAGER,7839,1981-04-02 00:00:00,2975,,20
7654,MARTIN,SALESMAN,7698,1981-09-28 00:00:00,1250,1400,30
7698,BLAKE,MANAGER,7839,1981-05-01 00:00:00,2850,,30
7782,CLARK,MANAGER,7839,1981-06-09 00:00:00,2450,,10
7788,SCOTT,ANALYST,7566,1987-04-19 00:00:00,3000,,20
7839,KING,PRESIDENT,,1981-11-17 00:00:00,5000,,10
7844,TURNER,SALESMAN,7698,1981-09-08 00:00:00,1500,0,30
7876,ADAMS,CLERK,7788,1987-05-23 00:00:00,1100,,20
7900,JAMES,CLERK,7698,1981-12-03 00:00:00,950,,30
7902,FORD,ANALYST,7566,1981-12-03 00:00:00,3000,,20
7934,MILLER,CLERK,7782,1982-01-23 00:00:00,1300,,10
7948,JACCKA,CLERK,7782,1981-04-12 00:00:00,5000,,10
7956,WELAN,CLERK,7649,1982-07-20 00:00:00,2450,,10
7956,TEBAGE,CLERK,7748,1982-12-30 00:00:00,1300,,10
```

现在需要将所有职工根据部门分组，每个组内根据SAL做降序排序，获得职工自己组内的序号。

```
SELECT deptno
       , ename
       , sal
       , RANK() OVER (PARTITION BY deptno ORDER BY sal DESC) AS nums
-- deptno(部门)作为开窗列，sal（薪水）作为结果返回时需要排序的值。
   FROM emp;
-- 执行结果如下：
+------------+-------+------------+------------+
| deptno     | ename | sal        | nums       |
+------------+-------+------------+------------+
| 10         | JACCKA | 5000.0     | 1          |
| 10         | KING  | 5000.0     | 1          |
| 10         | CLARK | 2450.0     | 3          |
| 10         | WELAN | 2450.0     | 3          |
| 10         | TEBAGE | 1300.0     | 5          |
| 10         | MILLER | 1300.0     | 5          |
| 20         | SCOTT | 3000.0     | 1          |
| 20         | FORD  | 3000.0     | 1          |
| 20         | JONES | 2975.0     | 3          |
| 20         | ADAMS | 1100.0     | 4          |
| 20         | SMITH | 800.0      | 5          |
| 30         | BLAKE | 2850.0     | 1          |
| 30         | ALLEN | 1600.0     | 2          |
| 30         | TURNER | 1500.0     | 3          |
| 30         | MARTIN | 1250.0     | 4          |
| 30         | WARD  | 1250.0     | 4          |
| 30         | JAMES | 950.0      | 6          |
```

```
+------------+-------+------------+------------+
```

26.5.5.4.11 LAG

函数声明如下：

```
lag(expr，bigint offset, default) over(partition by col1[, col2...] [order by col1 [asc|desc][, col2[asc|desc]...]])
```

用途：按偏移量取当前行之前第几行的值，如当前行号为rn，则取行号为rn-offset的值。

参数说明：

- expr：任意类型。
- offset：Bigint 类型常量。输入为string类型，double类型会隐式转换到bigint类型后参与运算，offset > 0。
- default：常量。当offset指定的范围越界时的缺省值，默认值为NULL。
- partition by col1[, col2..]：指定开窗口的列。
- order by col1 [asc|desc], col2[asc|desc]：指定返回结果的排序方式。

返回值：同expr类型。

26.5.5.4.12 LEAD

函数声明如下：

```
lead(expr, bigint offset, default) over(partition by col1[, col2...][order by col1 [asc|desc][, col2[asc|desc]...]])
```

用途：按偏移量取当前行之后第几行的值，如当前行号为rn则取行号为rn+offset的值。

参数说明：

- expr：任意类型。
- offset：Bigint类型常量。输入为string类型，double类型会隐式转换到bigint类型后参与运算，offset > 0。
- default：常量。当offset指定的范围越界时的缺省值。
- partition by col1[, col2..]：指定开窗口的列。
- order by col1 [asc|desc], > col2[asc|desc]：指定返回结果的排序方式。

返回值：同expr类型。

示例如下：

```
select c_double_a,c_string_b,c_int_a,lead(c_int_a,1) over(partition by c_double_a order by c_string_b) from dual;
select c_string_a,c_time_b,c_double_a,lead(c_double_a,1) over(partition by c_string_a order by c_time_b) from dual;
select c_string_in_fact_num,c_string_a,c_int_a,lead(c_int_a) over(partition by c_string_in_fact_num order by c_string_a) from dual;
```

26.5.5.4.13 PERCENT_RANK

函数声明如下：

```
percent_rank() over(partition by col1[, col2...] order by col1 [asc|desc][, col2[asc|desc]...])
```

用途：计算一组数据中某行的相对排名。

参数说明：

- partition by col1[, col2..]：指定开窗口的列。
- order by col1 [asc|desc], col2[asc|desc]：指定排名依据的值。

返回值：Double类型。值域为[0, 1]，相对排名的计算方式为：**(rank-1)/(number of rows -1)**。



说明: 目前限制单个窗口内的行数不超过10,000,000条。

26.5.5.4.14 ROW_NUMBER

函数声明如下：

```
row_number() over(partition by col1[, col2...] order by col1 [asc|desc][, col2[asc|desc]...])
```

用途：计算行号，从1开始。

参数说明：

- partition by col1[, col2..]：指定开窗口的列。
- order by col1 [asc|desc], > col2[asc|desc]：指定结果返回时的排序的值。

返回值：Bigint类型。

示例如下：

如表emp中数据如下：

```
| empno | ename | job | mgr | hiredate| sal| comm | deptno |
7369,SMITH,CLERK,7902,1980-12-17 00:00:00,800,,20
7499,ALLEN,SALESMAN,7698,1981-02-20 00:00:00,1600,300,30
7521,WARD,SALESMAN,7698,1981-02-22 00:00:00,1250,500,30
```

```
7566,JONES,MANAGER,7839,1981-04-02 00:00:00,2975,,20
7654,MARTIN,SALESMAN,7698,1981-09-28 00:00:00,1250,1400,30
7698,BLAKE,MANAGER,7839,1981-05-01 00:00:00,2850,,30
7782,CLARK,MANAGER,7839,1981-06-09 00:00:00,2450,,10
7788,SCOTT,ANALYST,7566,1987-04-19 00:00:00,3000,,20
7839,KING,PRESIDENT,,1981-11-17 00:00:00,5000,,10
7844,TURNER,SALESMAN,7698,1981-09-08 00:00:00,1500,0,30
7876,ADAMS,CLERK,7788,1987-05-23 00:00:00,1100,,20
7900,JAMES,CLERK,7698,1981-12-03 00:00:00,950,,30
7902,FORD,ANALYST,7566,1981-12-03 00:00:00,3000,,20
7934,MILLER,CLERK,7782,1982-01-23 00:00:00,1300,,10
7948,JACCKA,CLERK,7782,1981-04-12 00:00:00,5000,,10
7956,WELAN,CLERK,7649,1982-07-20 00:00:00,2450,,10
7956,TEBAGE,CLERK,7748,1982-12-30 00:00:00,1300,,10
```

现在需要将所有职工根据部门分组，每个组内根据SAL做降序排序，获得职工自己组内的序号。

```
SELECT deptno
       , ename
       , sal
       , ROW_NUMBER() OVER (PARTITION BY deptno ORDER BY sal DESC) AS nums
-- deptno(部门)作为开窗列，sal（薪水）作为结果返回时需要排序的值。
   FROM emp;
-- 执行结果如下：
+------------+-------+------------+------------+
| deptno     | ename | sal        | nums       |
+------------+-------+------------+------------+
| 10         | JACCKA | 5000.0     | 1          |
| 10         | KING  | 5000.0     | 2          |
| 10         | CLARK | 2450.0     | 3          |
| 10         | WELAN | 2450.0     | 4          |
| 10         | TEBAGE | 1300.0     | 5          |
| 10         | MILLER | 1300.0     | 6          |
| 20         | SCOTT | 3000.0     | 1          |
| 20         | FORD  | 3000.0     | 2          |
| 20         | JONES | 2975.0     | 3          |
| 20         | ADAMS | 1100.0     | 4          |
| 20         | SMITH | 800.0      | 5          |
| 30         | BLAKE | 2850.0     | 1          |
| 30         | ALLEN | 1600.0     | 2          |
| 30         | TURNER | 1500.0     | 3          |
| 30         | MARTIN | 1250.0     | 4          |
| 30         | WARD  | 1250.0     | 5          |
| 30         | JAMES | 950.0      | 6          |
+------------+-------+------------+------------+
```

26.5.5.4.15 CLUSTER_SAMPLE

函数声明如下：

```
boolean cluster_sample(bigint x[, bigint y]) over(partition by col1[, col2..])
```

用途：分组抽样。

参数说明：

- x：Bigint类型常量。x>=1。若指定参数y，x表示将一个窗口分为x份；否则，x表示在一个窗口中抽取x行记录（即有x行返回值为true）。x为NULL时，返回值为NULL。
- y：Bigin t类型常量。y>=1，y<=x。表示从一个窗口分的x份中抽取y份记录（即y份记录返回值为true）。y为NULL时，返回值为NULL。
- partition by col1[, col2]：指定开窗口的列。

返回值：Boolean类型。

示例如下：

如表test_tbl中有key，value两列，key为分组字段，值有groupa，groupb两组，value为值，如下所示：

```
+------------+-------------------+
| key        | value             |
+------------+-------------------+
| groupa     | -1.34764165478145  |
| groupa     | 0.740212609046718  |
| groupa     | 0.167537127858695  |
| groupa     | 0.630314566185241  |
| groupa     | 0.0112401388646925 |
| groupa     | 0.199165745875297  |
| groupa     | -0.320543343353587 |
| groupa     | -0.273930924365012 |
| groupa     | 0.386177958942063  |
| groupa     | -1.09209976687047  |
| groupb     | -1.10847690938643  |
| groupb     | -0.725703978381499 |
| groupb     | 1.05064697475759   |
| groupb     | 0.135751224393789  |
| groupb     | 2.13313102040396   |
| groupb     | -1.11828960785008  |
| groupb     | -0.849235511508911 |
| groupb     | 1.27913806620453   |
| groupb     | -0.330817716670401 |
| groupb     | -0.300156896191195 |
| groupb     | 2.4704244205196    |
| groupb     | -1.28051882084434  |
+------------+-------------------+
```

想要从每组中抽取约10%的值，可以使用以下MaxCompute SQL完成：

```
select key, value from (select key, value, cluster_sample(10, 1) over(partition by key) as flag from
 tbl) sub where flag = true;
-- 返回结果：
+--------+-------------------+
| key    | value             |
+--------+-------------------+
| groupa | -0.273930924365012  |
| groupb | -1.11828960785008 |
```

```
+-----+-----------------------
```

26.5.5.5 聚合函数

聚合函数，其输入与输出是多对一的关系，即将多条输入记录聚合成一条输出值。可以与 SQL 中的 group by 语句联用。

26.5.5.5.1 COUNT

函数声明如下：

```
bigint count([distict|all] value)
```

用途：计算记录数。

参数说明：

- distinct|all：指明在计数时是否去除重复记录，默认是all，即计算全部记录。如果指定distinct，则可以只计算唯一值数量。
- value：可以为任意类型，当value值为NULL时，该行不参与计算。value可以为*，当count(*)时，返回所有行数。

返回值：Bigint 类型。

示例如下：

```
-- 如表tbla有列col1类型为bigint
+------+
| COL1 |
+------+
| 1    |
+------+
| 2    |
+------+
| NULL |
+------+
select count(*) from tbla;
-- 值为3
select count(col1) from tbla;
-- 值为2
```

聚合函数可以和group by一同使用，例如：假设存在表test_src，存在如下两列：key string类型，value double类型。

```
-- test_src的数据为
+-----+-------+
| key | value |
+-----+-------+
| a   | 2.0   |
+-----+-------+
| a   | 4.0   |
```

```
+-----+-------+
| b   | 1.0   |
+-----+-------+
| b   | 3.0   |
+-----+-------+
-- 此时执行如下语句，结果为：
select key, count(value) as count from test_src group by key;
+-----+-------+
| key | count |
+-----+-------+
| a   | 2     |
+-----+-------+
| b   | 2     |
+-----+-------+
```

聚合函数将对相同key值的value值做聚合计算。下面介绍的其他聚合函数使用方法均与此例相同，不一一举例。

26.5.5.5.2 AVG

函数声明如下：

```
double avg(double value)
decimal avg(decimal value)
```

用途：计算平均值。

参数说明：

value：Double类型或Decimal类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。当value值为NULL时，该行不参与计算。Boolean类型不允许参与计算。

返回值：若输入Decimal类型，返回Decimal类型，其他合法输入类型返回Double类型。

示例如下：

```
-- 如表tbla有一列value，类型为bigint
+-------+
| value |
+-------+
| 1     |
| 2     |
| NULL  |
+-------+
-- 则对该列计算avg结果为(1+2)/2=1.5
select avg(value) as avg from tbla;
+------+
| avg  |
+------+
| 1.5  |
```

```
+------+
```

26.5.5.5.3 MAX

函数声明如下：

```
max(value)
```

用途：计算最大值。

参数说明：

value：可以为任意类型，当列中的值为NULL时，该行不参与计算。Boolean类型不允许参与运算。

返回值：与value类型相同。

示例如下：

```
-- 如表tbla有一列col1，类型为bigint
+------+
| col1 |
+------+
| 1    |
+------+
| 2    |
+------+
| NULL |
+------+
select max(value) from tbla;
-- 返回值为2
```

26.5.5.5.4 MIN

函数声明如下：

```
MIN(value)
```

用途：计算最小值。

参数说明：

value：可以为任意类型，当列中的值为NULL时，该行不参与计算。Boolean类型不允许参与计算。

返回值：与value类型相同。

示例如下：

```
-- 如表tbla有一列value，类型为bigint
+------+
```

```
| value|
+------+
| 1    |
+------+
| 2    |
+------+
| NULL |
+------+
select min(value) from tbla;
-- 返回值为1
```

26.5.5.5.5 MEDIAN

函数声明如下：

```
double median(double number)
decimal median(decimal number)
```

用途：计算中位数。

参数说明：

number：Double类型或Decimal类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。当输入值为NULL时返回失败。

返回值：Double类型或Decimal类型。

26.5.5.5.6 STDDEV

函数声明如下：

```
double stddev(double number)
decimal stddev(decimal number)
```

用途：计算总体标准差。

参数说明：

number：Double类型或Decimal类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。当输入值为NULL时返回失败。

返回值：Double类型或Decimal类型。

26.5.5.5.7 STDDEV_SAMP

函数声明如下：

```
double stddev_samp(double number)
decimal stddev_samp(decimal number)
```

用途：计算样本标准差。

参数说明：

number：Double类型或Decimal类型。若输入为string类型或bigint类型会隐式转换到double类型后参与运算，为其他类型时报错。当输入值为NULL时返回失败。

返回值：Double类型或Decimal类型。

26.5.5.5.8 SUM

函数声明如下：

```
sum(value)
```

用途：计算汇总值。

参数说明：

value：double类型、decimal类型或Bigint类型。若输入为string会隐式转换到double类型后参与运算，当列中的值为NULL 时，该行不参与计算。Boolean类型不允许参与计算。

返回值：输入为bigint类型时返回bigint类型，输入为double类型或string类型时返回double类型。

示例如下：

```
-- 如表tbla有一列value，类型为bigint
+------+
| value|
+------+
| 1    |
+------+
| 2    |
+------+
| NULL |
+------+
select sum(value) from tbla;
-- 返回值为3
```

26.5.5.5.9 WM_CONCAT

函数声明如下：

```
string wm_concat(string separator, string str)
```

用途：用指定的spearator做分隔符，连接str中的值。

参数说明：

- separator：String类型常量，分隔符。为其他类型或非常量将报错。

- str：String类型。若输入为bigint类型，double类型或datetime类型会隐式转换为string类型后参与运算，为其他类型时报错。

返回值：String 类型。



说明: 对语句“select wm_concat(',', name) from > test_src;”，若test_src为空集合，这条MaxCompute SQL语句返回NULL。

26.5.5.5.10 新扩展聚合函数说明

升级到MaxCompute2.0以后，产品扩展了部分聚合函数。

本章节后介绍的聚合函数即为新扩展的聚合函数。

26.5.5.5.11 COLLECT_LIST

函数声明如下：

```
ARRAY  collect_list(col)
```

用途：在给定group内，将col指定的表达式聚合为一个数组。

参数说明：

col：表的某列，可为任意数据类型。

返回值：ARRAY类型。

26.5.5.5.12 COLLECT_SET

函数声明如下：

```
ARRAY  collect_set(col)
```

用途：在给定group内，将col指定的表达式聚合为一个无重复元素的集合数组。

参数说明：

col：表的某列，可为任意数据类型。

返回值：ARRAY类型。

26.5.5.6 其他函数

26.5.5.6.1 ARRAY

函数声明如下：

```
array(value1,value2, ...)
```

用途：使用给定value构造ARRAY。

参数说明：

value：value可为任意类型，但是所有value必须类型一致。

返回值：ARRAY类型。

示例如下：

```
select array(123,456,789) from dual;
-- 返回结果：
[123, 456, 789]
```

26.5.5.6.2 ARRAY_CONTAINS

函数声明如下：

```
array_contains(ARRAY<T> a, value v)
```

用途：检测给定ARRAY a中是否包含v。

参数说明：

- a：Array类型数据。
- v：给出的v必须与array数组中的数据类型一致。

返回值：Boolean类型。

示例如下：

```
select array_contains(array('a','b'), 'a') from dual;
-- 返回true
select array_contains(array(456,789),123) from dual;
```

```
-- 返回false
```

26.5.5.6.3 CAST

函数声明如下：

```
cast(expr as <type>)
```

用途：将表达式的结果转换成目标类型，如 cast(‘1’ as bigint) 将字符串 “1” 转为整数类型的1，如果转换不成功或不支持的类型转换会报错。



说明:

- cast(double as bigint)，将 double类型的值转换成bigint类型。
- cast(string as bigint) ，在将字符串转为bigint类型时，如果字符串中是以整型表达的数字，会直接转为bigint类型。 如果字符串中是以浮点数或指数形式表达的数字，则会先转为double类型，再转为bigint类型。
- cast(string as datetime)或cast(datetime as > string) 时，会采用默认的日期格式**yyyy-mm-dd hh:mi:ss**。

26.5.5.6.4 COALESCE

函数声明如下：

```
coalesce(expr1, expr2, ...)
```

用途：返回列表中第一个非NULL的值，如果列表中所有的值都是NULL则返回NULL。

参数说明：

expr：要测试的值。所有这些值类型必须相同或为NULL，否则会报错。

返回值：返回值类型和参数类型相同。



说明: 参数至少要有一个，否则报错。

26.5.5.6.5 DECODE

函数声明如下：

```
decode(expression, search, result[, search, result]...[, default])
```

用途：实现if-then-else分支选择的功能。

参数说明：

- expression：要比较的表达式。
- search：和expression进行比较的搜索项。
- result：search和expression的值匹配时的返回值。
- default：可选项，如果所有的搜索项都不匹配，则返回此default值，如果未指定，则会返回NULL。

返回值：返回匹配的search；如果没有匹配，返回default；如果没有指定default，返回NULL。



说明:

- 至少要指定三个参数。
- 所有的result类型必须一致，或为NULL，不一致的数据类型会报错。所有的search和expression类型必须一致，否则报错。
- 如果decode中的search选项有重复且匹配时，会返回第一个值。

示例如下：

```
select decode(customer_id,
1, 'Taobao',
2, 'Alipay',
3, 'Aliyun',
NULL, 'N/A',
'Others') as result from sale_detail;
```

上面的decode函数实现了下面if-then-else语句中的功能：

```
if customer_id = 1 then result := 'Taobao';
elsif customer_id = 2 then result := 'Alipay';
elsif customer_id = 3 then result := 'Aliyun';
...
else
result := 'Others';
end if;
```



注意: 通常情况下MaxCompute SQL在计算NULL = NULL时返回NULL，但在decode函数中，NULL与NULL的值是相等的。在上述事例中，当customer_id的值为NULL时，decode函数返回N/A。

26.5.5.6.6 EXPLODE

函数声明如下：

```
explode (var)
```

用途：用于将一行数据转为多行的UDTF，如果var是array类型，则将列中存储的array转为多行；如果var是map类型，则将列中存储的map的每个key-value转换为包含两列的行，其中一列存储key，另一列存储value。

参数说明：

var： array < T > 类型或map < K,V > 类型。

返回值：转置后的行。



说明:

UDTF 使用上有以下限制：

- 在一个select中只能有一个udtf，不可以再出现其他的列。
- 不可以与group by/cluster by/distribute by/sort by一起使用。

示例如下：

```
explode(array(null, 'a', 'b', 'c'))  col
```

26.5.5.6.7 GET_IDCARD_AGE

函数声明如下：

```
get_idcard_age(idcardno)
```

用途：根据身份证号返回当前的年龄，当前年份减去身份证中标识的出生年份的差值。

参数说明：

idcardno：String类型，15位或18位身份证号。在计算时会根据省份代码以及最后一位校验码检查身份证的合法性，如果校验不通过会返回NULL。

返回值：Bigint 类型。若输入参数为NULL，返回NULL。如果当前年份减去出生年份差值大于100，返回NULL。

26.5.5.6.8 GET_IDCARD_BIRTHDAY

函数声明如下：

```
get_idcard_birthday(idcardno)
```

用途：根据身份证号返回出生日期。

参数说明：

idcardno：String类型，15位或18位身份证号。在计算时会根据省份代码以及最后一位校验码检查身份证的合法性，如果校验不通过会返回NULL。

返回值：Datetime 类型。若输入参数为NULL，返回NULL。

26.5.5.6.9 GET_IDCARD_SEX

函数声明如下：

```
get_idcard_sex(idcardno)
```

用途：根据身份证号返回性别，值为M（男）或F（女）。

参数说明：

idcardno：String类型，15位或18位身份证号。在计算时会根据省份代码以及最后一位校验码检查身份证的合法性，如果校验不通过会返回NULL。

返回值：String 类型。若输入参数为NULL，返回NULL。

26.5.5.6.10 GREATEST

函数声明如下：

```
greatest(var1, var2, ...)
```

用途：返回输入参数中最大的一个。

参数说明：

var：可以为bigint类型，double类型，datetime类型或string类型。若所有值都为NULL则返回NULL。

返回值：

- 输入参数中的最大值，当不存在隐式转换时返回同输入参数类型。
- NULL为最小值。

- 当输入参数类型不同时，double类型，bigint类型，string类型之间的比较转为double类型；string类型，datetime类型的比较转为datetime类型。不允许其他的隐式转换。

26.5.5.6.11 INDEX

函数声明如下：

```
index(Var1[var2])
```

用途：如果Var1是array类型，函数返回第var2个元素值；如果Var1是map类型，函数返回Var1中key为var2的value。

参数说明：

- Var1：arrray < T >类型或map < K,V >类型。
- var2：如果Var1是array < T >类型，则var2是bigint类型并且var2 >= 0；如果Var1是map < K,V >类型，var2是K类型。

返回值：

- 如果Var1是array < T >类型，函数返回T类型，在var2超出array < T >元素数目范围的情况下返回NULL。
- 如果Var1是map < K,V >类型，函数返回V类型，在map < K,V >中不存在key为var2的情况下返回NULL。

示例如下：

```
--当Var1为数组时，执行如下sql:
select array('a','b','c')[2] from dual;
-- 返回结果：
+-----+
| _c0 |
+-----+
| c   |
+-----+
--Var1为map类型，执行如下sql：
select str_to_map("test1=1,test2=2")["test1"] from dual;
-- 返回结果：
+-----+
| _c0 |
+-----+
| 1   |
+-----+
```

注意:

- 使用的时候，需要去掉index，直接执行Var1[Var2]，否则会报语法错误。

- 当Var1为NULL时，返回NULL。

26.5.5.6.12 MAX_PT

函数声明如下：

```
max_pt(table_full_name)
```

用途：对于分区的表，此函数返回该分区表的一级分区的最大值，按字母排序，且该分区下有对应的数据文件。

参数说明：

table_full_name：String 类型。指定表名（必须带上project名，例如：prj.src），用户必须对此表有读权限。

返回值：最大的一级分区的值。

示例如下：

如tbl是分区表，该表对应的分区如下，且都有数据文件pt='20120901'，pt='20120902' ，则以下语句中max_pt返回值为'20120902'，MaxCompute SQL语句读出pt='20120902'分区下的数据。

```
select * from tbl where pt=max_pt('myproject.tbl');
```



说明: 如果只是用alter table的方式新加了一个分区，但是此分区中并无任何数据文件，则此分区不会做为返回值。

26.5.5.6.13 ORDINAL

函数声明如下：

```
ordinal(bigint nth, var1, var2, ...)
```

用途：将输入变量按从小到大排序后，返回nth指定的位置的值。

参数说明：

- nth：Bigint类型。指定要返回的位置，为NULL时返回NULL。
- var：可以为bigint类型，double类型，datetime类型或string类型。

返回值：

- 排在第nth位的值，当不存在隐式转换时返回同输入参数类型。

- 有类型转换时，double类型，bigint类型，string类型之间的转换返回double类型。string类型，datetime类型之间的转换返回datetime类型。不允许其他的隐式类型转换。
- NULL为最小。

示例如下：

```
ordinal(3, 1, 3, 2, 5, 2, 4, 6) =  2
```

26.5.5.6.14 LEAST

函数声明如下：

```
least(var1, var2, ...)
```

用途：返回输入参数中最小的一个。

参数说明：

var：可以为bigint类型，double类型，datetime类型或string类型。若所有值都为NULL则返回NULL。

返回值：

- 输入参数中的最小值，当不存在隐式转换时返回同输入参数类型。
- 有类型转换时，double类型，bigint类型，string类型之间的转换返回double类型。string类型，datetime类型之间的转换返回datetime类型。不允许其他的隐式类型转换。
- NULL为最小。

26.5.5.6.15 SIZE

函数声明如下：

```
size(map<K, V>)
size(array<T>)
```

用途：size(map)返回给定MAP中K/V对数;size(array)返回给定的array中的元素数目。

参数说明：

- map：Map类型数据。
- array：Array类型数据。

返回值：Int 类型。

示例如下：

```
select size(map('a',123,'b',456)) from dual;
-- 返回2
select size(map('a',123,'b',456,'c',789)) from dual;
-- 返回3
select size(array('a','b')) from dual;
-- 返回2
select size(array(123,456,789)) from dual;
-- 返回3
```

26.5.5.6.16 SPLIT

函数声明如下：

```
split(str, pat)
```

用途：使用pat分隔str。

参数说明：

- str：String 类型。指被分隔的字符串。
- pat:：String 类型。分隔符，支持正则。

返回值：array <string >，元素是str被pat分隔后的结果。

示例如下：

```
select split("a,b,c",",") from dual;
-- 返回结果：
+------+
| _c0  |
+------+
| [a, b, c] |
+------+
```

26.5.5.6.17 STR_TO_MAP

函数声明如下：

```
str_to_map(text [, delimiter1 [, delimiter2]])
```

用途：使用delimiter1将text分隔成K-V对，然后使用delimiter2分隔每个K-V对。

参数说明：

ext：String类型。指被分隔的字符串。

delimiter1：String类型。分隔符，不指定时默认为' ,'。

delimiter2：String类型。分隔符，不指定时默认为'='。

返回值：map < string, string >，元素是text被delimiter1和delimiter2分隔后的K-V结果。

示例如下：

```
select str_to_map("test1=1,test2=2") from dual;
-- 返回结果：
+------------+
| a          |
+------------+
| {test1:1, test2:2} |
```

26.5.5.6.18 UNIQUE_ID

函数声明如下：

```
STRING UNIQUE_ID()
```

用途：返回一个随机的唯一id，形式示例："29347a88-1e57-41ae-bb68-a9edbdd94212_1"，相对于uuid运行效率比较高。

26.5.5.6.19 UUID

函数声明如下：

```
string uuid()
```

用途：返回一个随机ID，形式示例：” 29347a88-1e57-41ae-bb68-a9edbdd94212” 。

26.5.5.6.20 SAMPLE

函数声明如下：

```
boolean sample(x, y,column_name)
```

用途：对所有读入的column_name的值，sample根据x，y的设置做采样，并过滤掉不满足采样条件的行。

参数说明：

- x，y：Bigint 类型。表示哈希为x份，取第y份。y可省略，省略时取第一份，如果省略参数中的y，则必须同时省略column_name。x，y为整型常量，大于 0，为其他类型或小于等于0时报错，若 y>x 也报错。x，y任意一个输入参数为NULL，返回NULL。
- column_name：是采样的目标列。column_name可以省略，省略时根据x，y的值随机采样。任意类型，列的值可以为NULL。不做隐式类型转换。如果column_name为常量NULL会报错。

返回值：Boolean 类型。

说明: 为了避免NULL值带来的数据倾斜，因此对于column_name中为NULL的值，会在x份中进行均匀哈希。如果不加column_name，则数据量比较少时输出不一定均匀，在这种情况下建议加上column_name，以获得比较好的输出结果。

示例如下：

假定存在表tbla，表内有列名为cola 的列：

```
select * from tbla where sample (4, 1 , cola) = true;
-- 表示数值会根据cola hash为4份，取第1份
select * from tbla where sample (4, 2) = true;
-- 表示数值会对每行数据做随机哈希分配为4份，取第2份
```

26.5.5.6.21 CASE WHEN表达式

MaxCompute 提供两种case when语法格式，如下所述：

```
case value
when (_condition1) then result1
when (_condition2) then result2
...
else resultn
end

case
when (_condition1) then result1
when (_condition2) then result2
when (_condition3) then result3
...
else resultn
end
```

case when表达式可以根据表达式value的计算结果灵活返回不同的值，如以下语句根据shop_name的不同情况得出所属区域：

```
select
case
when shop_name is null then 'default_region'
when shop_name like 'hang%' then 'zj_region'
end as region
from sale_detail;
```

说明:

- 如果result的类型只有bigint类型和double类型，统一转换为double类型再返回。
- 如果result的类型中有string类型，统一转换为string类型再返回，如果不能转换则报错（如boolean型）。
- 除此之外不允许其他类型之间的转换。

26.5.5.6.22 IF

函数声明如下：

```
if(testCondition, valueTrue, valueFalseOrNull)
```

用途：判断testCondition是否为真，如果为真，返回valueTrue，如果不满足则返回另一个值(valueFalse或者Null)。

参数说明：

testCondition：Boolean类型。要判断的表达式。

valueTrue：表达式testCondition为True的时候，返回的值。

valueFalseOrNull：不满足表达式testCondition时，返回的值，可以设为NULL。

返回值：返回值类型和参数valueTrue或者valueFalseOrNul的类型一致。

示例如下：

```
select if(1=2,100,200) from dual;
-- 返回结果：
+------------+
| _c0        |
+------------+
| 200        |
+------------+
```

26.5.5.6.23 新扩展其他函数说明

升级到MaxCompute2.0以后，产品扩展了部分其他函数。

本章节后介绍的其他函数即为新扩展的其他函数。

26.5.5.6.24 MAP

函数声明如下：

```
map(K key1, V value1, K key2, V value2, ...)
```

用途：使用给定key/value对建立map 。

参数说明：

key/value：所有key类型一致，必须是基本类型；所有value类型一致，可为任意类型。

返回值：MAP类型。

示例如下：

```
select  map('a',123,'b',456) from dual;
-- 返回结果：
{a:123, b:456}
```

26.5.5.6.25 MAP_KEYS

函数声明如下：

```
map_keys(map<K, V> )
```

用途：将参数map中的所有key作为数组返回。

参数说明：

map：Map类型数据。

返回值：ARRAY类型。若输入参数为NULL，返回NULL。

示例如下：

```
select map_keys(map('a',123,'b',456)) from dual;
-- 返回结果：
[a, b]
```

26.5.5.6.26 MAP_VALUES

函数声明如下：

```
map_values(map<K, V>)
```

用途：将参数map中的所有values作为数组返回。

参数说明：

map：Map类型数据。

返回值：ARRAY类型。若输入参数为NULL，返回NULL。

示例如下：

```
select map_keys(map('a',123,'b',456)) from dual;
-- 返回结果：
```

```
[123, 456]
```

26.5.5.6.27 SORT_ARRAY

函数声明如下：

```
sort_array(ARRAY<T>)
```

用途：对给定数组排序。

参数说明：

ARRAY：Array类型数据。数组中的数据可为任意类型。

返回值：ARRAY类型。

示例如下：

```
select    sort_array(array('a','c','f','b')),sort_array(array(4,5,7,2,5,8)),sort_array(array('你','我','他'))
 from dual;
-- 返回结果：
[a, b, c, f] [2, 4, 5, 5, 7, 8] [他, 你, 我]
```

26.5.5.6.28 POSEXPLODE

函数声明如下：

```
posexplode(ARRAY<T>)
```

用途：将给定ARRAY展开，每个value一行，每行两列分别对应数组从0开始的下标和数组元素。

参数说明：

ARRAY：Array类型数据。数组中的数据可为任意类型。

返回值：表生成函数。

示例如下：

```
select posexplode(array('a','c','f','b')) from dual;
-- 返回结果：
+------------+-----+
| pos        | val |
+------------+-----+
| 0          | a   |
| 1          | c   |
| 2          | f   |
| 3          | b   |
```

```
+------------+-----+
```

26.5.5.6.29 STRUCT

函数声明如下：

```
struct(value1,value2, ...)
```

用途：使用给定value列表建立struct。

参数说明：

value：各value可为任意类型。

返回值：STRUCT类型。生成struct的field的名称依次为col1，col2，...。

示例如下：

```
select struct('a',123,'ture',56.90) from dual;
-- 返回结果：
{col1:a, col2:123, col3:ture, col4:56.9}
```

26.5.5.6.30 NAMED_STRUCT

函数声明如下：

```
named_struct(string name1, T1 value1, string name2, T2 value2, ...)
```

用途：使用给定name/value列表建立struct。

参数说明：

- value：各value可为任意类型。
- name：指定的string类型的field名称。

返回值：STRUCT类型。生成struct的field的名称依次为name1，name2，...。

示例如下：

```
select named_struct('user_id',10001,'user_name','LiLei','married','F','weight',63.50) from dual;
-- 返回结果：
```

```
{user_id:10001, user_name:LiLei, married:F, weight:63.5}
```

26.5.5.6.31 INLINE

函数声明如下：

```
inline(ARRAY<STRUCT<f1:T1, f2:T2, ...>>)
```

用途：将给定struct数组展开，每个元素对应一行，每行每个struct元素对应一列。

参数说明：

STRUCT：数组中的value可为任意类型。

返回值：表生成函数。

示例如下：

```
select inline(array(named_struct('user_id',10001,'user_name','LiLei','married','F','weight',63.50)))
 from dual;
-- 返回结果：
+------------+-----------+---------+------------+
| user_id    | user_name | married | weight     |
+------------+-----------+---------+------------+
| 10001      | LiLei     | F       | 63.5       |
+------------+-----------+---------+------------+
```

26.5.5.6.32 BETWEEN AND 表达式

格式定义如下：

```
A [NOT] BETWEEN B AND C
```

如果A、B或C为空，则为空；如果A大于或等于B且小于或等于C，则为true，否则为false。

示例如下：

如表emp中数据如下：

```
| empno | ename | job | mgr | hiredate| sal| comm | deptno |
7369,SMITH,CLERK,7902,1980-12-17 00:00:00,800,,20
7499,ALLEN,SALESMAN,7698,1981-02-20 00:00:00,1600,300,30
7521,WARD,SALESMAN,7698,1981-02-22 00:00:00,1250,500,30
7566,JONES,MANAGER,7839,1981-04-02 00:00:00,2975,,20
7654,MARTIN,SALESMAN,7698,1981-09-28 00:00:00,1250,1400,30
7698,BLAKE,MANAGER,7839,1981-05-01 00:00:00,2850,,30
7782,CLARK,MANAGER,7839,1981-06-09 00:00:00,2450,,10
7788,SCOTT,ANALYST,7566,1987-04-19 00:00:00,3000,,20
7839,KING,PRESIDENT,,1981-11-17 00:00:00,5000,,10
7844,TURNER,SALESMAN,7698,1981-09-08 00:00:00,1500,0,30
7876,ADAMS,CLERK,7788,1987-05-23 00:00:00,1100,,20
7900,JAMES,CLERK,7698,1981-12-03 00:00:00,950,,30
7902,FORD,ANALYST,7566,1981-12-03 00:00:00,3000,,20
7934,MILLER,CLERK,7782,1982-01-23 00:00:00,1300,,10
```

```
7948,JACCKA,CLERK,7782,1981-04-12 00:00:00,5000,,10
7956,WELAN,CLERK,7649,1982-07-20 00:00:00,2450,,10
7956,TEBAGE,CLERK,7748,1982-12-30 00:00:00,1300,,10
```

查询sal在大于等于1000 小于等于1500之间的数据。

```
select * from emp where sal BETWEEN 1000 and 1500;
-- 返回结果：
+-------+-------+-----+------------+------------+------------+------------+------------+
| empno | ename | job | mgr        | hiredate   | sal        | comm       | deptno     |
+-------+-------+-----+------------+------------+------------+------------+------------+
| 7521  | WARD  | SALESMAN | 7698  | 1981-02-22 00:00:00 | 1250.0     | 500.0      | 30  |
| 7654  | MARTIN | SALESMAN | 7698 | 1981-09-28 00:00:00 | 1250.0     | 1400.0     | 30 |
| 7844  | TURNER | SALESMAN | 7698 | 1981-09-08 00:00:00 | 1500.0     | 0.0        | 30 |
| 7876  | ADAMS | CLERK | 7788  | 1987-05-23 00:00:00 | 1100.0     | NULL       | 20  |
| 7934  | MILLER | CLERK | 7782  | 1982-01-23 00:00:00 | 1300.0     | NULL       | 10  |
| 7956  | TEBAGE | CLERK | 7748  | 1982-12-30 00:00:00 | 1300.0     | NULL       | 10  |
+-------+-------+-----+------------+------------+------------+------------+------------+
```

26.5.5.7 分词函数

MaxCompute封装了AliWS的功能，在MaxCompute中封装实现了四个函数：

uniform，normalize，segment及synonym。



说明:

- 这四个函数并不是每个project下都有打开，如果没有这几个函数请联系project owner，必须由project owner创建后才可使用。
- 在使用这四个函数的sql前，需要添加如下的set语句。

```
set odps.sql.type.system.odps2=true;
```

26.5.5.7.1 UNIFORM

函数声明如下：

```
string uniform(string)
```

用途：分词，重组，归一化（小写，半角，简体，去特殊符号）。

示例如下：

```
uniform('abc A 電話')
```

```
-- 返回 'abca 电话'
```

26.5.5.7.2 UDF_NORMALIZE

函数声明如下：

```
string udf_normalize(string)
```

用途：归一化（小写，半角，简体，去特殊符号）。

示例如下：

```
udf_normalize('abc A 電話')
-- 返回 'abca电话'
```

26.5.5.7.3 SEGMENT

函数声明如下：

```
string segment(string str, bigint grainLevel = 1, boolean posTag = false)
```

用途：简单分词后用空格分开，不排序，不重组，不替换，不归一。

参数说明：

- grainLevel：分词粒度。1是最大粒度， 2是最小粒度， 4是中间粒度，其他值报错，默认行为是最大粒度。
- posTag：是否添加词性标注。默认为false，不添加词性标注。

> **说明:** 添加词性标注功能时，AliTokenizer.conf中的enable_pos_tagger必须置为true。

示例如下：

```
segment('abc A 電話')
-- 返回 'abc A 電話'
```

26.5.5.7.4 SYNONYM

函数声明如下：

```
string synonym(string)
```

用途：同义词替换，主要包括错别字纠正，品牌中英转换。

示例如下：

```
synonym('nokia, 诺基亚, motorola')
```

```
-- 返回 '诺基亚, 诺基亚, 摩托罗拉'
```

26.5.6 UDF

26.5.6.1 概要

UDF全称User Defined Function，即用户自定义函数。MaxCompute提供了很多内建函数来满足用户的计算需求，同时用户还可以通过创建自定义函数来满足不同的计算需求。UDF在使用上与普通的内建函数类似。本章节只对SQL UDF的使用方法做简单的介绍，关于 SQL UDF的详细说明请参考官方提供的文档。

在MaxCompute中，用户可以扩展的UDF如下所示：

UDF 分类	描述
User Defined Scalar Function，通常也称之为UDF	用户自定义标量值函数（User Defined Scalar Function）通常也称之为UDF。其输入与输出是一对一的关系，即读入一行数据，写出一条输出值。
UDTF（User Defined Table Valued Function）	自定义表值函数，是用来解决一次函数调用输出多行数据场景的，也是唯一能返回多个字段的自定义函数。 而UDF只能一次计算输出一条返回值。
UDAF（User Defined Aggregation Function）	自定义聚合函数，其输入与输出是多对一的关系，即将多条输入记录聚合成一条输出值。可以与SQL中的Group By语句联用。具体语法请参考聚合函数。



说明： UDF广义的说法代表了自定义标量函数，自定义聚合函数及自定义表函数三种类型的自定义函数的集合。狭义来说，仅代表用户自定义标量函数。文档会经常使用这一名词，请读者根据文档上下文判断具体含义。

26.5.6.1.1 参数与返回值类型

UDF支持MaxCompute SQL的数据类型有：

- 基本数据类型：bigint类型，double类型，boolean类型，datetime类型，decimal类型，string类型，tinyint类型，smallint类型，int类型，float类型，varchar类型，binary类型以及timestamp类型。
- 复杂数据类型：array类型，map类型以及struct类型。

其中，Java UDF使用tinyint，smallint，int，float，varchar，binary和timestamp等几个基本数据类型的方法，如下所示：

- UDAF和UDTF通过@Resolve注解来获取signature，例如：@Resolve("smallint->varchar(10)")。
- UDF通过反射分析evaluate来获取signature，此时MaxCompute内置类型与Java类型符合一一映射关系。

JAVA UDF使用array，map以及struct三个复杂数据类型的方法，如下所示：

- UDAF和UDTF通过@Resolve annotation来指定sinature，例如：@Resolve("array<string>,struct<a1:bigint,b1:string>,string->map<string,bigint>,struct<b1:bigint>")。
- UDF通过evaluate方法的signature来映射UDF的输入输出类型，此时参考MaxCompute类型与Java类型的映射关系。其中array对应java.util.List，map对应java.util.Map，struct对应com.aliyun.odps.data.Struct。

  注意:

  - com.aliyun.odps.data.Struct从反射看不出field name和field type，所以需要用@Resolve annotation来辅助。即如果需要在UDF中使用struct，要求在UDF class上也标注上@Resolve注解，这个注解只会影响参数或返回值中包含com.aliyun.odps.data.Struct的重载。
  - 目前class上只能提供一个@Resolve annotation，因此一个UDF中带有struct参数或返回值的重载只能有一个。

MaxCompute数据类型与Java类型的对应关系，如下所示：

MaxCompute Type	Java Type
Tinyint	java.lang.Byte
Smallint	java.lang.Short
Int	java.lang.Integer
Bigint	java.lang.Long
Float	java.lang.Float
Double	java.lang.Double
Decimal	java.math.BigDecimal
Boolean	java.lang.Boolean
String	java.lang.String
Varchar	com.aliyun.odps.data.Varchar

MaxCompute Type	Java Type
Binary	com.aliyun.odps.data.Binary
Datetime	java.util.Date
Timestamp	java.sql.Timestamp
Array	java.util.List
Map	java.util.Map
Struct	com.aliyun.odps.data.Struct

说明:

- Java中对应的数据类型以及返回值数据类型是对象，首字母请务必大写。
- SQL中的NULL值通过Java中的NULL引用表示，因此Java primitive type是不允许使用的，因为无法表示SQL中的NULL值。
- 此处的Array类型对应的java类型是List，而不是数组。

两种语言接口feature对比情况见下表所示：

支持语言	UDF	UDAF	UDTF	Datetime 类型	读资源文件	读资源表
Python	Yes	Yes	Yes	Yes	Yes	Yes
Java	Yes	Yes	Yes	yes	Yes	Yes

26.5.6.2 UDF

实现UDF需要继承com.aliyun.odps.udf.UDF类，并实现evaluate方法。Evaluate方法必须是非static的public方法。Evaluate方法的参数和返回值类型将作为SQL中UDF的函数签名。这意味着用户可以在UDF中实现多个evaluate方法，在调用UDF时，框架会依据UDF调用的参数类型匹配正确的evaluate方法。

UDF示例如下：

```
package org.alidata.odps.udf.examples;
import com.aliyun.odps.udf.UDF;
public final class Lower extends UDF { public String evaluate(String s) { if (s == null) { return null; }
 return s.toLowerCase();
}
}
```

说明: 可以通过实现void setup(ExecutionContext ctx)和void close()来分别实现UDF的初始化和结束代码。

UDF的使用方式与MaxCompute SQL中普通的内建函数相同，详情请参考：内建函数

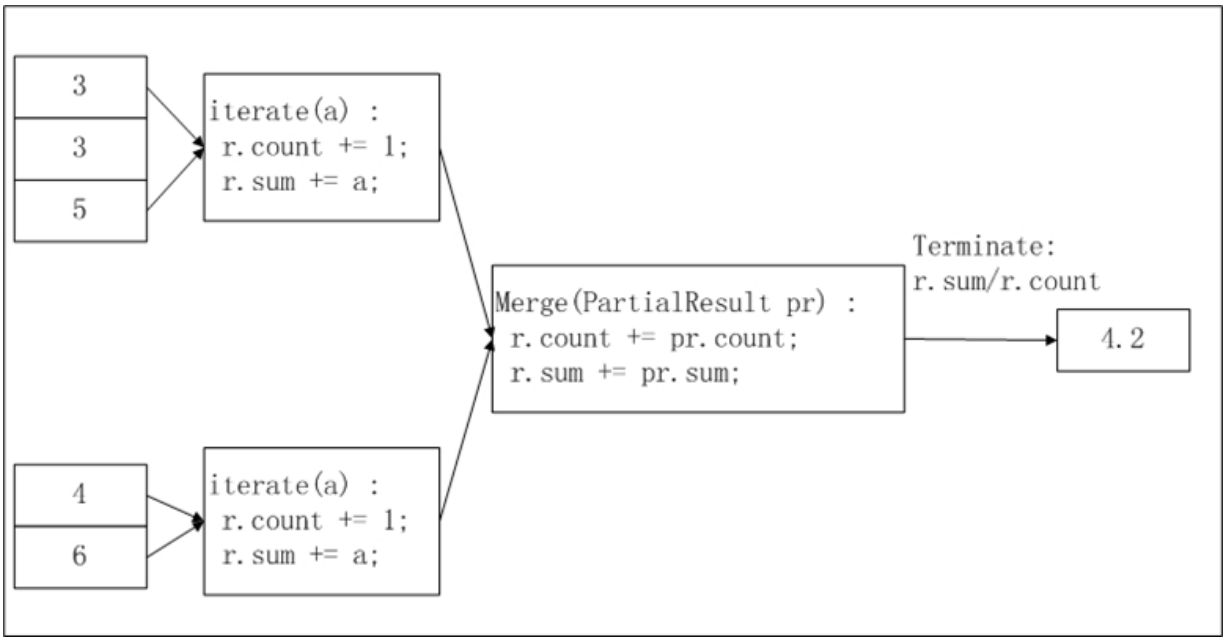
26.5.6.3 UDAF

实现Java UDAF类需要继承com.aliyun.odps.udf.UDAF，并实现如下几个接口：

```
public abstract class Aggregator implements ContextFunction {
@Override
public void setup(ExecutionContext ctx) throws UDFException {
}
@Override
public void close() throws UDFException {
}
/**
* 创建聚合Buffer
* @return Writable 聚合buffer
*/
abstract public Writable newBuffer();
/**
* @param buffer 聚合buffer
* @param args SQL中调用UDAF时指定的参数
* @throws UDFException
*/
abstract public void iterate(Writable buffer, Writable[] args) throws UDFException;
/**
* 生成最终结果
* @param buffer
* @return Object UDAF的最终结果
* @throws UDFException
*/
abstract public Writable terminate(Writable buffer) throws UDFException;
abstract public void merge(Writable buffer, Writable partial) throws UDFException;
}
```

其中最重要的是iterate，merge和terminate三个接口，UDAF的主要逻辑依赖于这三个接口的实现。

此外，还需要用户实现自定义的Writable buffer。以实现求平均值avg为例，下图简要说明了在MaxCompute UDAF中这一函数的实现逻辑及计算流程。



在上图中，输入数据被按照一定的大小进行分片（有关分片的描述可参考：*MapReduce*章节），每片的大小适合一个worker在适当的时间内完成。这个分片大小的设置需要用户手动配置完成。

UDAF 的计算过程分为如下两个阶段：

- 第一阶段：每个worker统计分片内数据的个数及汇总值，用户可以将每个分片内的数据个数及汇总值视为一个中间结果。
- 第二阶段：worker汇总上一个阶段中每个分片内的信息。在最终输出时，r.sum / r.count即是所有输入数据的平均值。

以计算平均值为例，UDAF**示例如下**：

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.udf.Aggregator;
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.annotation.Resolve;
@Resolve({"double->double"})
public class AggrAvg extends Aggregator {
private static class AvgBuffer implements Writable { private double sum = 0;
private long count = 0;
@Override
public void write(DataOutput out) throws IOException { out.writeDouble(sum);
out.writeLong(count);
}
@Override
public void readFields(DataInput in) throws IOException { sum = in.readDouble();
count = in.readLong();
}
}
```

```
private DoubleWritable ret = new DoubleWritable();
@Override
public Writable newBuffer() { return new AvgBuffer();
}
@Override
public void iterate(Writable buffer, Writable[] args) throws UDFException { DoubleWritable arg =
 (DoubleWritable) args[0];
AvgBuffer buf = (AvgBuffer) buffer; if (arg != null) {
buf.count += 1; buf.sum += arg.get();
}
}
@Override
public Writable terminate(Writable buffer) throws UDFException { AvgBuffer buf = (AvgBuffer)
 buffer;
if (buf.count == 0) { ret.set(0);
} else {
ret.set(buf.sum / buf.count);
}
return ret;
}
@Override
public void merge(Writable buffer, Writable partial) throws UDFException { AvgBuffer buf =
 (AvgBuffer) buffer;
AvgBuffer p = (AvgBuffer) partial; buf.sum += p.sum;
buf.count += p.count;
}
}
```

注意:

- UDAF在SQL中的使用语法与普通的内建聚合函数相同，详情请参考 聚合函数。
- 关于如何运行UDTF的方法与UDF 类似，详情请参考 运行*UDF*。

26.5.6.4 UDTF

26.5.6.4.1 概要说明

Java UDTF需要继承com.aliyun.odps.udf.UDTF类。这个类需要实现4个接口。接口定义描述如下：

接口	描述
public void setup(ExecutionContext ctx) throws UDFException	初始化方法，在UDTF处理输入数据前，调用用户自定义的初始化行为。在每个Worker内setup会被先调用一次。
public void process(Object[] args) throws UDFException	这个方法由框架调用，SQL中每一条记录都会对应调用一次process，process的参数为SQL语句中指定的UDTF输入参数。输入参数以Object[]的形式传入，输出结果通过forward函数输出。用户需要在process函数内自行调用forward，以决定输出数据。

接口	描述
public void close() throws UDFException	UDTF的结束方法，此方法由框架调用，并且只会被调用一次，即在处理完最后一条记录之后。
public void forward(Object ...o) throws UDFException	用户调用forward方法输出数据，每次forward代表输出一条记录。对应SQL语句UDTF的as子句指定的列。

UDTF示例如下：

```
package org.alidata.odps.udtf.examples;
import com.aliyun.odps.udf.UDTF;
import com.aliyun.odps.udf.UDTFCollector;
import com.aliyun.odps.udf.annotation.Resolve;
import com.aliyun.odps.udf.UDFException;
// TODO define input and output types, e.g., "string,string->string,bigint".
@Resolve({"string,bigint->string,bigint"}) public class MyUDTF extends UDTF {
@Override public void process(Object[] args) throws UDFException { String a = (String) args[0];
Long b = (Long) args[1];
for (String t: a.split("\\s+")) { forward(t, b);
}
}
}
```

在SQL中可以这样使用这个UDTF，假设在ODPS上创建UDTF时注册函数名为user_udtf：

```
select user_udtf(col0, col1) as (c0, c1) from my_table;
```

假设my_table的col0， col1的值如下所示：

```
+------+------+
| col0 | col1 |
+------+------+
| A B  | 1    |
| C D  | 2    |
+------+------+
```

则select出的结果为：

```
+----+----+
| c0 | c1 |
+----+----+
| A  | 1  |
| B  | 1  |
| C  | 2  |
| D  | 2  |
```

```
+----+----+
```

26.5.6.4.2 UDTF使用说明

UDTF在SQL中的常用方式如下：

```
select user_udtf(col0, col1) as (c0, c1) from my_table;
select user_udtf(col0, col1) as (c0, c1) from (select * from my_table distribute by col1 sort by col1) t;
```

注意:

使用UDTF有如下使用限制。

- 同一个select子句中不允许有其他表达式。

  ```
  select col0, user_udtf(col0, col1) as (c0, c1) from mytable;
  ```

- UDTF不能嵌套使用。

  ```
  select user_udtf(mp_udtf(col0,col1)) as (c0,c1)from mytable;
  ```

UDTF相关示例

在UDTF中，用户可以读取MaxCompute的资源。下面将介绍利用UDTF读取MaxCompute资源的相关示例。

1. 编写UDTF程序，编译成功后导出jar包（udtfexample1.jar）。

```
package com.aliyun.odps.examples.udf;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Iterator;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.UDTF;
import com.aliyun.odps.udf.annotation.Resolve;
/**
* project: example_project
* table: wc_in2
* partitions: p2=1,p1=2
* columns: colc,colb
*/
@Resolve({ "string,string->string,bigint,string" }) public class UDTFResource extends UDTF { ExecutionContext ctx;
long fileResourceLineCount;
long tableResource1RecordCount;
long tableResource2RecordCount;
@Override
public void setup(ExecutionContext ctx) throws UDFException { this.ctx = ctx;
try {
InputStream in = ctx.readResourceFileAsStream("file_resource.txt");
BufferedReader br = new BufferedReader(new InputStreamReader(in));
```

```
String line;
fileResourceLineCount = 0;
while ((line = br.readLine()) != null) { fileResourceLineCount++;
}
br.close();
Iterator<Object[]> iterator = ctx.readResourceTable("table_resource1").iterator();
tableResource1RecordCount = 0;
while (iterator.hasNext()) { tableResource1RecordCount++; iterator.next();
}
iterator = ctx.readResourceTable("table_resource2").iterator();
tableResource2RecordCount = 0;
while (iterator.hasNext()) { tableResource2RecordCount++;
iterator.next();
}
} catch (IOException e) { throw new UDFException(e);
}
}
@Override
public void process(Object[] args) throws UDFException { String a = (String) args[0];
long b = args[1] == null ? 0 : ((String) args[1]).length();
forward(a, b, "fileResourceLineCount=" + fileResourceLineCount + "|tableResource1Record
Count=" + tableResource1RecordCount + "|tableResource2RecordCount=" +
tableResource2RecordCount);
}
}
```

2. 添加资源到MaxCompute。

```
Add file file_resource.txt;
Add jar udtfexample1.jar;
Add table table_resource1 as table_resource1;
Add table table_resource2 as table_resource2;
```

3. 在MaxCompute中创建UDTF函数（mp_udtf ）。

```
create function mp_udtf as com.aliyun.odps.examples.udf.UDTFResource using
 'udtfexample1.jar, file_resource.txt, table_resource1, table_resource2';
```

4. 在MaxCompute创建资源表table_resource1、table_resource2，并插入相应的数据。

5. 运行该UDTF。

```
select mp_udtf("10","20") as (a, b, fileResourceLineCount) from table_resource1;
-- 返回结果：
+-------+------------+-------+
| a | b      | fileResourceLineCount |
+-------+------------+-------+
|  10         |  2                      |  fileResourceLineCount=3|tableResource1RecordCount=0|
tableResource2RecordCount=0 |
|  10         |  2                      |  fileResourceLineCount=3|tableResource1RecordCount=0|
tableResource2RecordCount=0 |
+-------+------------+-------+
```

说明: 用户同样可以使用相同的方式获取资源，详情请参考：*MapReduce使用示例*。

UDTF相关示例——复杂数据类型示例

如下示例中的代码，定义了一个有三个overloads的UDF。其中第一个使用了array作为参数，第二个使用了map作为参数，第三个使用了struct作为参数。由于第三个overloads使用了struct作为参数或者返回值，因此要求必须要对UDF class打上@Resolve annotation，来指定struct的具体类型。

```
@Resolve("struct<a:bigint>,string->string")
public class UdfArray extends UDF {
  public String evaluate(List<String> vals, Long len) {
    return vals.get(len.intValue());
  }
  public String evaluate(Map<String,String> map, String key) {
    return map.get(key);
  }
  public String evaluate(Struct struct, String key) {
    return struct.getFieldValue("a") + key;
  }
}
```

用户可以直接将复杂数据类型传入UDF中，如下所示：

```
create function my_index as 'UdfArray' using 'myjar.jar';
select id, my_index(array('red', 'yellow', 'green'), colorOrdinal) as color_name from colors;
```

26.5.6.5 Python UDF

26.5.6.5.1 受限环境

MaxCompute　UDF的Python版本为2.7，并以沙箱模式执行用户代码，即代码是在一个受限的运行环境中执行的。在这个环境中，被禁止的行为包括：

- 读写本地文件。
- 启动子进程。
- 启动线程。
- 使用socket通信。
- 其他系统调用。

基于上述原因，用户上传的代码必须都是纯Python实现，C扩展模块是被禁止的。

此外，Python的标准库中也不是所有模块都可用，涉及到上述功能的模块都会被禁止。具体标准库可用模块说明如下：

1. 所有纯Python实现（不依赖扩展模块）的模块都可用。
2. C实现的扩展模块中下列模块可用。
   - array
   - audioop

- binascii
- _bisect
- cmath
- _codecs_cn
- _codecs_hk
- _codecs_iso2022
- _codecs_jp
- _codecs_kr
- _codecs_tw
- _collections
- cStringIO
- datetime
- _functools
- future_builtins
- _hashlib
- _heapq
- itertools
- _json
- _locale
- _lsprof
- math
- _md5
- _multibytecodec
- operator
- _random
- _sha256
- _sha512
- _sha
- _struct
- strop

- time
- unicodedata
- _weakref
- cPickle

3. 部分模块功能受限。例如沙箱限制了用户代码最多能往标准输出和标准错误输出写出数据的大小，即sys.stdout/sys.stderr最多能写20Kb，多余的字符会被忽略。

26.5.6.5.2 第三方库

运行环境中还安装了除标准库以外比较常用的三方库，做为标准库的补充。支持的三方库还包括：numpy。



警告: 三方库的使用同样受到禁止本地、网络IO或其他在受限环境下的限制，因此三方库中涉及到相关功能的API 也是被禁止的。

26.5.6.5.3 参数与返回值类型

参数与返回值通过如下方式指定：

```
@odps.udf.annotate(signature)
```

Python UDF目前支持的MaxCompute SQL数据类型有：bigint类型，string类型，double类型，boolean类型和datetime类型。SQL语句在执行之前，所有函数的参数类型和返回值类型必须确定。因此对于Python这一动态类型语言，需要通过对UDF类加decorator的方式指定函数签名。

函数签名signature通过字符串指定，语法如下：

```
arg_type_list '->' type_list
arg_type_list: type_list | '*' | ''
type_list: [type_list ','] type
type: 'bigint' | 'string' | 'double' | 'boolean' | 'datetime'
```



说明:

- 箭头左边表示参数类型，右边表示返回值类型。
- 只有UDTF的返回值可以是多列，UDF和UDAF只能返回一列。
- ‘*’ 代表变长参数，使用变长参数，UDF/UDTF/UDAF可以匹配任意输入参数。

下面是合法的signature的示例：

```
'bigint,double->string'          # 参数为bigint类型、double类型，返回值为string类型
'bigint,boolean->string,datetime'    #  UDTF参数为bigint类型、boolean类型，返回值为string类型,datetime类型
'*->string'                      # 变长参数，输入参数任意，返回值为string类型
```

```
'->double'                    # 参数为空，返回值为double类型
```

Query语义解析阶段会将检查到不符合函数签名的用法，报错禁止执行。执行期间，UDF函数的参数会以函数签名指定的类型传给用户。用户的返回值类型也要与函数签名指定的类型一致，否则检查到类型不匹配时也会报错。MaxCompute SQL数据类型对应Python类型如下所示：

表 95:

MaxCompute SQL Type	Python Type
Bigint	int
String	str
Double	float
Boolean	bool
Datetime	int

说明:

- Datetime类型是以int的形式传给用户代码的，值为epoch utc time起始至今的毫秒数。用户可以通过Python标准库中的datetime模块处理日期时间类型。
- NULL值对应Python里的None。

此外，**odps.udf.int(value[, silent=True])**的参数也做了调整。增加了参数silent 。当silent为true时，如果 value无法转换为int ，不会报错，而是返回None 。

26.5.6.5.4 UDF

实现Python UDF非常简单，只需要定义一个new-style class，并实现evaluate方法。

示例如下：

```
from odps.udf import annotate
@annotate("bigint,bigint->bigint")
class MyPlus(object):
   def evaluate(self, arg0, arg1):
      if None in (arg0, arg1):
         return None
      return arg0 + arg1
```

注意: Python UDF必须通过annotate指定函数签名。

26.5.6.5.5 UDAF

参数说明：

- class odps.udf.BaseUDAF：继承此类实现Python UDAF。
- BaseUDAF.new_buffer()：实现此方法返回聚合函数的中间值的buffer。buffer必须是mutable object（例如list、dict），并且buffer的大小不应该随数据量递增。在极限情况下，buffer marshal过后的大小不应该超过2Mb。
- BaseUDAF.iterate(buffer[, args, ...])：实现此方法将args聚合到中间值buffer中。
- BaseUDAF.merge(buffer, pbuffer)：实现此方法将两个中间值buffer聚合到一起，即pbuffer merge到buffer中。
- BaseUDAF.terminate(buffer)：实现此方法将中间值buffer转换为MaxCompute SQL基本类型。

以UDAF求平均值为例，示例如下：

```
#coding:utf-8
from odps.udf import annotate
from odps.udf import BaseUDAF
@annotate('double->double')
class Average(BaseUDAF):
    def new_buffer(self):
        return [0, 0]
    def iterate(self, buffer, number):
        if number is not None:
            buffer[0] += number
            buffer[1] += 1
    def merge(self, buffer, pbuffer):
        buffer[0] += pbuffer[0]
        buffer[1] += pbuffer[1]
    def terminate(self, buffer):
        if buffer[1] == 0:
            return 0.0
        return buffer[0] / buffer[1]
```

26.5.6.5.6 UDTF

参数说明：

- class odps.udf.BaseUDTF：Python UDTF的基类，用户继承此类，并实现process，close 等方法。
- BaseUDTF.init()：初始化方法，继承类如果实现这个方法，则必须在一开始调用基类的初始化方法super(BaseUDTF, self).init() 。init方法在整个UDTF生命周期中只会被调用一次，即在处理第一条记录之前。当UDTF需要保存内部状态时，可以在这个方法中初始化所有状态。
- BaseUDTF.process([args, ...])：这个方法由MaxCompute SQL框架调用，SQL中每一条记录都会对应调用一次process ，process的参数为SQL语句中指定的UDTF输入参数。

- BaseUDTF.forward([args, ...])：UDTF的输出方法，此方法由用户代码调用。每调用一次forward，就会输出一条记录。forward的参数为SQL语句中指定的UDTF的输出参数。
- BaseUDTF.close()：UDTF的结束方法，此方法由MaxCompute SQL框架调用，并且只会被调用一次，即在处理完最后一条记录之后。

示例如下：

```
#coding:utf-8
# explode.py
from odps.udf import annotate
from odps.udf import BaseUDTF
@annotate('string -> string')
class Explode(BaseUDTF):
  """将string按逗号分隔输出成多条记录
  """
  def process(self, arg):
    props = arg.split(',')
    for p in props:
      self.forward(p)
```

注意： Python UDTF也可以不加annotate指定参数类型和返回值类型。此时，函数在SQL中使用时可以匹配任意输入参数，但返回值类型无法推导，所有输出参数都将认为是string类型。因此在调用forward时，就必须将所有输出值转换成str类型。

26.5.6.5.7 引用资源

Python UDF可以通过odps.distcache模块引用资源文件，目前支持引用文件资源和表资源。

引用文件资源的语法如下：

```
odps.distcache.get_cache_file(resource_name)
```

说明：

- 参数说明：返回指定名字的资源内容。resource_name为str类型，对应当前Project中已存在的资源名。如果资源名非法或者没有相应的资源，会报错。
- 返回值：返回值为file-like object，在使用完这个object后，调用者有义务调用close方法释放打开的资源文件。

使用get_cache_file的示例如下：

```
from odps.udf import annotate
from odps.distcache import get_cache_file
@annotate('bigint->string')
class DistCacheExample(object):
def __init__(self):
  cache_file = get_cache_file('test_distcache.txt')
  kv = {}
  for line in cache_file:
```

```
        line = line.strip()
        if not line:
            continue
        k, v = line.split()
        kv[int(k)] = v
    cache_file.close()
    self.kv = kv
def evaluate(self, arg):
    return self.kv.get(arg)
```

引用表资源的语法如下：

```
odps.distcache.get_cache_table(resource_name)
```

说明:

- 参数说明：返回指定资源表的内容。resource_name为str类型，对应当前Project中已存在的资源表名。如果资源名非法或者没有相应的资源，会报错。
- 返回值：返回值为generator类型，调用者通过遍历获取表的内容，每次遍历得到的是以tuple形式存在的表中的一条记录。

使用get_cache_table的示例如下：

```
from odps.udf import annotate
from odps.distcache import get_cache_table
@annotate('->string')
class DistCacheTableExample(object):
    def __init__(self):
        self.records = list(get_cache_table('udf_test'))
        self.counter = 0
        self.ln = len(self.records)
    def evaluate(self):
        if self.counter > self.ln - 1:
            return None
        ret = self.records[self.counter]
        self.counter += 1
        return str(ret)
```

26.5.7 附录

26.5.7.1 转义字符

在MaxCompute SQL中的字符串常量可以用单引号或双引号表示，可以在单引号括起的字符串中包含双引号，或在双引号括起的字符串中包含单引号，否则要用转义符来表达，如下的表达方式都是正确的："I'm a happy manong!" 'I\'m a happy manong!'。

在MaxCompute SQL中，反斜线“\”即为转义符，用来表达字符串中的特殊字符，或将其后跟随的字符解释为其本身。当读入字符串常量时，如果反斜线后跟随三位有效的8进制数字，范围在001

~177之间，系统会根据ASCII值转换为相应的字符。对于如下表所示的情况，则会将其解释为特殊字符。

转义	字符
\b	backspace
\t	tab
\n	newline
\r	carriage-return
\'	单引号
\"	双引号
\ \	反斜线
\;	分号
\Z	control-Z
\0或\00	结束符

示例如下：

```
select length('a\tb') from dual;
-- 结果为3，表示字符串里实际有三个字符， “\t”被视为一个字符。在转义符后的其他字符被解释为其本身。
```

```
select 'a\ab',length('a\ab') from dual;

-- 结果为’ aab’ ，3。 “\a”被解释为普通的“a” 。
```

26.5.7.2 LIKE字符匹配

在LIKE匹配时， “%”表示匹配任意多个字符， “_”表示匹配单个字符，如果要匹配“%”或“_”本身，则要对其进行转义， “\%”匹配字符“%” ， “_”匹配字符“_” 。

说明: 关于字符串的字符集，目前MaxCompute SQL支持UTF-8的字符集。如果数据是以其他格式编码，可能会导致计算出的结果不正确。

26.5.7.3 正则表达式规范

MaxCompute SQL中的正则表达式采用的是PCRE的规范，匹配时是按字符进行，支持的元字符如下所示：

- ^ ：行首

- $ ：行尾
- . ：任意字符
- * ：匹配零次或多次
- + ：匹配1次或多次
- ? ：匹配修饰符，当该字符紧跟在任何一个其他限制符 (* ,+, ?, {n}, {n,}, {n,m}) 后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串，匹配零次或1次
- A|B ：A或B
- (abc)* ：匹配abc序列零次或多次
- {n}或{m,n} ：匹配的次数
- [ab] ：匹配括号中的任一字符
- [^ab] ：^表示非，匹配任一非a非b的字符
- \ ：转义符
- \n ：n为数字1-9，后向引用
- \d ：数字
- \D ：非数字
- [::] ：POSIX字符组
  - [[:alnum:]] ：字母字符和数字字符，范围[a-zA-Z0-9]
  - [[:alpha:]] ：字母，范围[a-zA-Z]
  - [[:ascii:]] ：ASCII字符，范围[\x00-\x7F]
  - [[:blank:]] ：空格字符和制表符，范围[\t]
  - [[:cntrl:]] ：控制字符，范围[\x00-\x1F\x7F]
  - [[:digit:]] ：数字字符，范围[0-9]
  - [[:graph:]] ：空白字符之外的字符，范围[\x21-\x7E]
  - [[:space:]] ：空白字符，范围[\t\r\n\v\f]
  - [[:print:]] ：[:graph:]和[:space:]，范围[\x20-\x7E]
  - [[:lower:]] ：小写字母字符，范围[a-z]
  - [[:punct:]] ：标点符号，范围[][!" #$%&()*+,./:;<=>? @\^_`{|}~-]
  - [[:upper:]] ：大写字母字符，范围[A-Z]
  - [[:xdigit:]] ：十六进制字符，范围[A-Fa-f0-9]

由于系统采用反斜线“\”作为转义符，因此正则表达式的模式中出现的“\”都要进行二次转义。例如：正则表达式要匹配字符串“a+b”，其中“+”是正则中的一个特殊字符，因此要用转义的方式表达，在正则引擎中的表达方式是“a+b”，由于系统还要解释一层转义，因此能够匹配该字符串的表达式是“a\\+b”。

假设存在表test_dual，示例如下：

```
select 'a+b' rlike 'a\\+b' from test_dual;
+------+
_c1 |
+------+
true |
+------+
```

极端的情况，如果要匹配字符“\”，由于在正则引擎中“\”是一个特殊字符，因此要表示为“\”，而系统还要对表达式进行一次转义，因此写成“\\”。

```
select 'a\\b', 'a\\b' rlike 'a\\\\b' from test_dual;
+-----+------+
_c0 | _c1 |
+-----+------+
a\b | true |
+-----+------+
```

说明:

- 在MaxCompute SQL中写“a\b”，而在输出结果中显示’ a\b’ ，同样是因为MaxCompute 会对表达式进行转义。
- 如果字符串中有制表符TAB，系统在读入’ \t’ 这两个字符的时候，即已经将其存为一个字符，因此在正则的模式中也是一个普通的字符。

  ```
  select 'a\tb', 'a\tb' rlike 'a\tb' from test_dual;
  +---------+------+
  _c0 | _c1 |
  +---------+------+
  a b | true |
  +---------+------+
  ```

-

26.5.7.4 保留字

以下是MaxCompute SQL的全部保留字，在对表、列或是分区命名时请不要使用，否则会报错。保留字不区分大小写。

```
% & && ( ) * +. / ; < <= <> = > >= ? ADD AFTER ALL ALTER ANALYZE AND ARCHIVE ARRAY
AS ASC BEFORE BETWEEN BIGINT BINARY BLOB BOOLEAN BOTH BUCKET BUCKETS
BY CASCADE CASE CAST CFILE CHANGE CLUSTER CLUSTERED CLUSTERSTATUS
COLLECTION COLUMN COLUMNS COMMENT COMPUTE CONCATENATE CONTINUE
CREATE CROSS CURRENT CURSOR DATA DATABASE DATABASES DATE DATETIME
```

DBPROPERTIES DEFERRED DELETE DELIMITED DESC DESCRIBE DIRECTORY DISABLE DISTINCT DISTRIBUTE DOUBLE DROP ELSE ENABLE END ESCAPED EXCLUSIVE EXISTS EXPLAIN EXPORT EXTENDED EXTERNAL FALSE FETCH FIELDS FILEFORMAT FIRST FLOAT FOLLOWING FORMAT FORMATTED FROM FULL FUNCTION FUNCTIONS GRANT GROUP HAVING HOLD_DDLTIME IDXPROPERTIES IF IMPORT IN INDEX INDEXES INPATH INPUTDRIVER INPUTFORMAT INSERT INT INTERSECT INTO IS ITEMS JOIN KEYS LATERAL LEFT LIFECYCLE LIKE LIMIT LINES LOAD LOCAL LOCATION LOCK LOCKS LONG MAP MAPJOIN MATERIALIZED MINUS MSCK NOT NO_DROP NULL OF OFFLINE ON OPTION OR ORDER OUT OUTER OUTPUTDRIVER OUTPUTFORMAT OVER OVERWRITE PARTITION PARTITIONED PARTITIONPROPERTIES PARTITIONS PERCENT PLUS PRECEDING PRESERVE PROCEDURE PURGE RANGE RCFILE READ READONLY READS REBUILD RECORDREADER RECORDWRITER REDUCE REGEXP RENAME REPAIR REPLACE RESTRICT REVOKE RIGHT RLIKE ROW ROWS SCHEMA SCHEMAS SELECT SEMI SEQUENCEFILE SERDE SERDEPROPERTIES SET SHARED SHOW SHOW_DATABASE SMALLINT SORT SORTED SSL STATISTICS STORED STREAMTABLE STRING STRUCT TABLE TABLES TABLESAMPLE TBLPROPERTIES TEMPORARY TERMINATED TEXTFILE THEN TIMESTAMP TINYINT TO TOUCH TRANSFORM TRIGGER TRUE UNARCHIVE UNBOUNDED UNDO UNION UNIONTYPE UNIQUEJOIN UNLOCK UNSIGNED UPDATE USE USING UTC UTC_TMESTAMP VIEW WHEN WHERE WHILE

26.5.7.5 限制项汇总

以下为汇总的MaxCompute的限制项说明：

边界名	最大值/限制条件	分类	说明
表名长度	128字节	长度限制	表名，列名中不能有特殊字符，只能用英文的 a-z,A-Z 及数字和下划线_，且以字母开头。
注释长度	1024字节	长度限制	注释内容是长度不超过1024 字节的有效字符串。
表的列定义	1200个	数量限制	单表的列定义个数最多1200个。
单表分区数	60000	数量限制	一张表最多允许60000个分区。
表的分区层级	6级	数量限制	在表中建的分区层次不能超过6级。
表统计定义个数	100个	数量限制	表统计定义个数。
表统计定义长度	64000	长度限制	表统计项定义长度。
屏显	10000行	数量限制	SELECT语句屏显默认最多输出10000 行。
insert目标个数	256个	数量限制	multiins同时insert的数据表数量。
UNION ALL	256个表	数量限制	最多允许256个表的UNION ALL。
join源	16个	数量限制	join的源表个数最多运行16 个。
MAPJOIN	8个小表	数量限制	MAPJOIN最多允许8张小表。

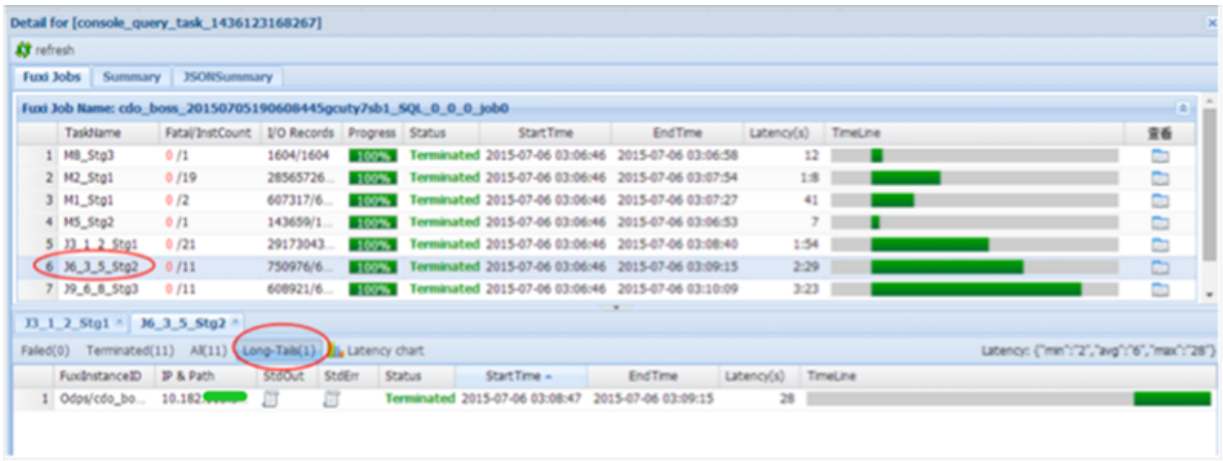
边界名	最大值/限制条件	分类	说明
MAPJOIN内存限制	512M	数量限制	MAPJOIN所有小表的内存限制不能超过512M（解压到内存）。
窗口函数	5个	数量限制	一个SELECT中最多允许5个窗口函数。
ptinsubq	1000行	数量限制	pt in subquery返回的结果不可以超过1000行。
sql语句长度	2M	长度限制	允许的sql语句的最大长度。
wherer子句条件个数	256个	数量限制	where子句中可使用条件个数。
列记录长度	8M	数量限制	表中一条记录的最大长度。
in的参数个数	1024	数量限制	in的最大参数限制，如 in (1,2,3....,10240)。in(...)如果参数过多，会造成编译时的压力；1024是建议值、不是限制值。
jobconf.json	1M	长度限制	jobconf.json的大小为1M。当表包含的Partition数量太多时，可能超过jobconf.json超过1M。
视图	不可写	操作限制	视图不可以写，不可以使用insert操作。
列的数据类型和位置	不允许修改	操作限制	不允许修改列的数据类型、列位置。
java udf函数	不能是abstract或者static	操作限制	java udf函数不能是abstract或者static。
最多查询分区个数	10000	数量限制	最多查询分区个数不能超过10000。

注意: 上述的MaxCompute SQL限制项均不可以被人为修改配置。

26.5.7.6 常见问题及使用建议

26.5.7.6.1 数据倾斜

通常，运行Job的instance的time、input records和output records参数的min、max和avg三个值不平衡时（例如max远大于avg 时），都可以判定已经出现了数据倾斜问题。通过查看Log view，也可以定位数据倾斜问题，如下图所示：



每个task的tab“Long Tails”展示了该task中出现数据倾斜的instance。产生数据倾斜的根本原因是有个别instance处理的数据量远远超过其他instance，以致于个别instance的运行时间也远远超过其他instance的平均时间，从而导致整个job的运行速度变慢。

SQL中不同类型的数据倾斜可以采用不同的方式来处理。

26.5.7.6.1.1 group by倾斜

可能原因：由于group by的key分布不均匀，从而导致reduce端的数据倾斜。

解决方法：可以在SQL执行前设置防group倾斜的参数：

```
set odps.sql.groupby.skewindata=true
```

说明: 一旦该参数设置为true，系统会在进行Shuffle hash算法时自动加入随机因素，并通过引入一个新增task来防止数据倾斜问题。

26.5.7.6.1.2 distribute by倾斜

可能原因：例如想对全表做全排序，而采用常量来进行distribute by，从而导致reduce端的数据倾斜。

解决方法：需要避免上述这种操作。

26.5.7.6.1.3 join倾斜

可能原因：由于join on所在的key分布不均匀（例如有个别key在join的多个表中有大量重复），从而导致个别join instance中的数据量以接近笛卡尔积式的数据量暴增，造成数据倾斜。

解决方法：根据场景，有如下相关方案来解决join倾斜：

- 如果join的两边有一个是小表，可以把join改成map join来处理。

- 倾斜的key用单独的逻辑来处理。例如经常出现的一种情况是两边表on的key里有大量null数据导致了倾斜，此时需要在join前先过滤掉null的数据或者通过case when将null值替换为某种随机值，然后再进行join。
- 如果不想更改SQL，可以通过设置如下参数让MaxCompute自动做优化。

```
set odps.sql.skewinfo=tab1:(col1，col2)[(v1，v2)，(v3，v4)，...]
set odps.sql.skewjoin=true;
```

26.5.7.6.1.4 muti-distinct倾斜

可能原因：多个distinct会放大group by数据倾斜问题。

解决方法：通常避免使用muti-distinct，可以采用两层group by来平缓数据倾斜问题。

26.5.7.6.1.5 误用动态分区导致的数据倾斜

可能原因：在使用动态分区的场景下，如果有K个Map instance，N个目标分区，可能产生K * N的小文件数，过多的小文件会对文件系统造成巨大的管理压力。所以，在默认情况下如下设置是生效的：

```
set odps.sql.reshuffle.dynamicpt=true;
```

即它会引入额外一级ReduceTask，相同的目标分区交由同一个（或少量几个）Reduce instance来写入，从避免小文件过多。但是 dynamic partition shuffle可能会导致数据倾斜。

解决方法：如果目标分区数比较少，根本就不会造成小文件过多的困扰，可以通过如下命令关闭上述功能；又或者不使用动态分区。

```
set odps.sql.reshuffle.dynamicpt=false;
```

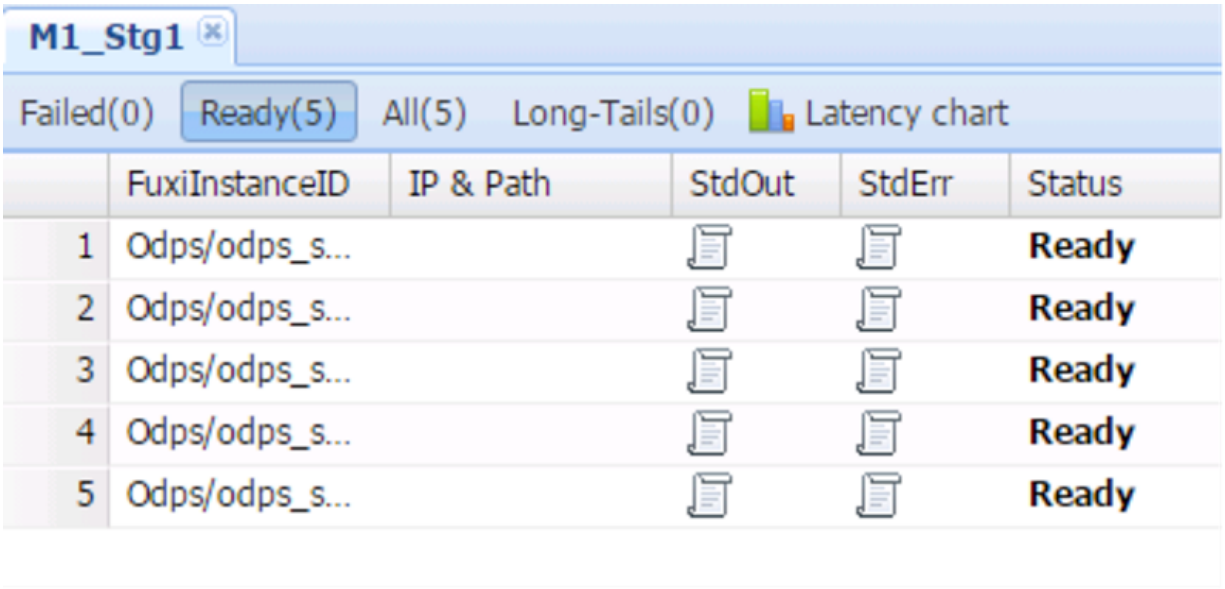
26.5.7.6.2 Quota及资源使用

在使用MaxCompute的时候，有时候会遇到一些计算资源不足的问题，涉及到对集群资源的规划和使用方式。

计算资源不足的作业一般有如下两个特征：首先，作业输出一直停留在某个阶段，进度停滞不前。例如在下图中，作业中的M1_Stg1 task 的完成百分比一直停留在0%（R2_1_Stg1由于依赖于M1_Stg1，所以在M1_Stg1完成前，其进度也会一直停留在0%）。

```
2016-01-29 13:52:09 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:52:14 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:52:19 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:52:24 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:52:29 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:52:34 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:52:39 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:52:44 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:52:49 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:52:54 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:52:59 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:53:04 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:53:09 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:53:15 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:53:20 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:53:25 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:53:30 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:53:35 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:53:40 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:53:45 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:53:50 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
2016-01-29 13:53:55 M1_Stg1_job0:0/0/5[0%]      R2_1_Stg1_job0:0/0/1[0%]
```

其次，通过Logview（如下图所示），可以看到task的instance会一直处于“Ready”状态（注意区别于Waiting，Waiting表明在等待依赖任务的完成，而Ready是在等待资源的分配）。该状态表明，目前没有资源来运行这些随时待命的task实例。一旦实例获得资源，就会转变为“Running”状态开始运行。



每个作业都会根据执行计划拆分成由多个task组成的DAG图，而每个task又会起多个instance来并发执行计算任务。通常起一个instance需要的资源是1核CPU和2G内存。为了合理分配资源，会为

每个project分配一个quota组，所属quota组决定了该project 中所有作业同时可以使用的资源上限（CPU 和内存）。一旦同时运行的作业的资源使用量达到了quota组的可用上限，就会出现因资源不足而导致的作业“不动了”的情况。

要解决该问题有如下两种方式：

- 让作业避开高峰，在空闲时间段运行。
- 扩大project的quota组资源量（需要找运维人员进行处理）。

26.5.7.6.3 MaxCompute储存优化技巧

- **合理设置分区表**

  MaxCompute支持分区表的概念，分区表指的是在创建表时指定的partition的分区空间，即指定表内的某几个字段作为分区列。在大多数情况下，用户可以将分区类比为文件系统下的目录。MaxCompute将分区列的每个值作为一个分区（目录）。用户可以指定多级分区，即将表的多个字段作为表的分区，分区之间类似多级目录的关系。在使用数据时如果指定了需要访问的分区名称，则只会读取相应的分区，避免全表扫描，提高处理效率，降低费用。

  分区示例：`create table src (key string, value bigint) partitioned by (pt string);`，使用`select * from src where pt='20160901';`指定正确的分区格式，MaxCompute在生成查询计划时只会将‘20151201’分区的数据纳入输入中。

  未分区示例：`select * from src where key = ‘MaxCompute’;`，查询计划会扫描全表数据。

  常见的分区设置方式有根据日期或者地区（国家），也可以根据业务需要自行设置。示例如下：

```
create table if not exists sale_detail(
shop_name     string,
customer_id   string,
total_price   double)
partitioned by (sale_date string,region string);
-- 创建一个二级分区表，第一级分区sale_date是日期，第二级分区region是地区。
```

- **合理设置表的生命周期**

  MaxCompute平台中存储资源是非常宝贵的。可以根据数据本身的使用情况，对表设置生命周期，MaxCompute会及时删除超过生命周期的数据，达到节省存储空间的目的。

  例如：`create table test3 (key boolean) partitioned by (pt string, ds string) lifecycle 100;`，创建一张生命周期为100的表。如果这张表或者分区的最后修改时间超过了100天将会被删除。



注意： 生命周期是以分区为最小单位的，所以针对一个分区表，如果部分分区达到了生命周期的阈值，那么这些分区会被直接删掉，未达到生命周期阈值的分区不受影响。

另外可以通过命令 alter table table_name set lifecycle days;修改已经创建好的表的生命周期。

- **归档冷数据**

  有些数据需要永久或者较长时间保存下来，但是随着时间的推移，使用频率会越来越低。当数据使用频率降低到一定程度后，可以考虑将数据进行归档操作。归档功能将数据存为raid file，数据不再简单的存三份，而是采用Cauchy Reed Solomon算法，将数据存为6份数据+3份校验块的方式，这样有效的存储比约为从1:3提高到1:1.5。另外 MaxCompute归档表的压缩算法采用了压缩率更高的bzip2算法。综合两种算法，存储空间可以节省70%以上。

  归档命令格式如下：

  ```
  ALTER TABLE table_name [PARTITION(partition_name='partition_value')] ARCHIVE;
  ```

  示例如下：

  ```
  alter table my_log partition(ds='20140101') archive;
  ```

- **合并小文件**

  在reduce计算过程或者实时tunnel数据采集过程，会产生大量小文件。小文件过多会带来以下问题：

  - 小文件越多，单个instance可以处理的文件数有限制，所以需要instance资源也越多。因此造成的浪费资源，影响整体的执行性能。
  - 过多的小文件给文件系统带来压力，并且影响磁盘空间的利用率。

  目前提供两种可供选择的小文件合并的方法：ALTER合并模式和SQL合并模式：

  - ALTER合并模式通过console命令行进行合并，命令格式如下：

    ```
    ALTER TABLE tablename [PARTITION] MERGE SMALLFILES;
    ```

  - SQL合并模式是在执行SQL结束之后，通过设置控制参数 odps.task.merge.enabled=true;进行是否需要小文件合并的判断，如果需要小文件合并，则另外启动Fuxi Job进行小文件合并。

26.5.7.6.4 UDF OOM问题

一些job在运行时会报OOM的问题，报错信息如下：

```
FAILED: ODPS-0123144: Fuxi job failed - WorkerRestart errCode:9,errMsg:SigKill(OOM), usually caused by OOM(out of memory)
```

这个问题可以尝试通过设置UDF运行时参数解决，如下所示：

```
odps.sql.mapper.memory=3072;
set odps.sql.udf.jvm.memory=2048;
set odps.sql.udf.python.memory=1536;
```

26.5.7.7 MaxCompute常用SQL参数设置

26.5.7.7.1 Map设置

```
set odps.sql.mapper.cpu=100
```

作用：设定处理map task每个instance的cpu 数目，默认为100。[50，800]之间调整。

```
set odps.sql.mapper.memory=1024
```

作用：设定map task每个instance的memory大小，单位M，默认1024M。[256，12288]之间调整。

```
set odps.sql.mapper.merge.limit.size=64
```

作用：设定控制文件被合并的最大阈值，单位M，默认64M。用户可以通过控制这个变量，从而达到对map端输入的控制。[0，Integer.MAX_VALUE]之间调整。

```
set odps.sql.mapper.split.size=256
```

作用：设定一个map的最大数据输入量，单位M，默认256M。用户可以通过控制这个变量，从而达到对map端输入的控制。[1，Integer.MAX_VALUE]之间调整。

26.5.7.7.2 Join设置

```
set odps.sql.joiner.instances=-1
```

作用：设定Join task的instance数量，默认为-1。[0，2000]之间调整。

```
set odps.sql.joiner.cpu=100
```

作用：设定Join task每个instance的cpu 数目，默认为100。[50，800]之间调整。

```
set odps.sql.joiner.memory=1024
```

作用：设定Join task每个instance的memory大小，单位为M，默认为1024M。[256 ，12288]之间调整。

26.5.7.7.3 Reduce设置

```
set odps.sql.reducer.instances=-1
```

作用：设定reduce task的instance数量，默认为-1。[0，2000]之间调整。

```
set odps.sql.reducer.cpu=100
```

作用：设定处理reduce task每个instance的cpu 数目，默认为100。[50，800]之间调整。

```
set odps.sql.reducer.memory=1024
```

作用：设定reduce task每个instance的memory大小，单位M， 默认1024M。[256 ，12288]之间调整。

26.5.7.7.4 UDF设置

```
set odps.sql.udf.jvm.memory=1024
```

作用：设定UDF jvm heap使用的最大内存，单位M，默认1024M。[256，12288]之间调整。

```
set odps.sql.udf.timeout=600
```

作用：设置UDF超时时间，默认为600 秒，单位秒。[0，3600]之间调整。

```
set odps.sql.udf.python.memory=256
```

作用：设定UDF python使用的最大内存，单位M，默认256M。[64，3072]之间调整。

```
set odps.sql.udf.optimize.reuse=true/false
```

作用：开启后，相同的UDF函数表达式，只计算一次，可以提高性能，默认为true。

```
set odps.sql.udf.strict.mode=false/true
```

作用：控制有些函数在遇到脏数据时是返回NULL还是报错，true是报错，flase是返回null。

26.5.7.7.5 MapJoin设置

```
set odps.sql.mapjoin.memory.max=512
```

作用：设置mapjoin时小表的最大内存，默认512，单位M，[128，2048]之间调整动态分区设置。

```
set odps.sql.reshuffle.dynamicpt=true/false
```

作用：

- 动态分区某些场景很慢，关闭可以加快SQL速度。
- 如果动态分区值很少，关闭后可以避免出现数据倾斜。

26.5.7.7.6 数据倾斜设置

```
set odps.sql.groupby.skewindata=true/false
```

作用：开启group by优化。

```
set odps.sql.skewjoin=true/false
```

作用：开启join优化，必须设置odps.sql.skewinfo才有效。

```
set odps.sql.skewinfo
```

作用：设置join优化具体信息，具体命令格式如下：

```
set odps.sql.skewinfo=skewed_src:(skewed_key)[("skewed_value")]
```

示例如下：

针对单个字段单个倾斜数值。

```
set odps.sql.skewinfo=src_skewjoin1:(key)[("0")]
-- 输出结果为explain select a.key c1, a.value c2, b.key c3, b.value c4 from src a join src_skewjoin1
 b on a.key = b.key;
```

针对单个字段多个倾斜数值。

```
set odps.sql.skewinfo=src_skewjoin1:(key)[("0")("1")]
-- 输出结果为explain select a.key c1, a.value c2, b.key c3, b.value c4 from src a join src_skewjoin1
 b on a.key = b.key;
```

26.6 Tunnel使用指南

MaxCompute Tunnel是MaxCompute的数据通道，用户可以通过Tunnel向MaxCompute中上传或者下载数据。目前Tunnel仅支持表（不包括视图 View）数据的上传下载。

26.6.1 Tunnel SDK介绍

MaxCompute提供的数据上传下载工具即是基于Tunnel SDK编写的。本章节将介绍Tunnel SDK的主要接口，不同版本的SDK在使用上有差别，准确信息以SDK Java Doc为准。

主要接口	描述
TableTunnel	访问MaxCompute Tunnel服务的入口类。
TableTunnel.UploadSession	表示一个向MaxCompute表中上传数据的会话。
TableTunnel.DownloadSession	表示一个向MaxCompute表中下载数据的会话。

注意: 对于tunnel endpoint，支持根据MaxCompute Endpoint自动路由，用户无需进行配置。

26.6.1.1 TableTunnel

接口定义：

```
public class TableTunnel {
public DownloadSession createDownloadSession(String projectName, String tableName);
public DownloadSession createDownloadSession(String projectName, String tableName,
 PartitionSpec partitionSpe c);
public UploadSession createUploadSession(String projectName, String tableName);
public UploadSession createUploadSession(String projectName, String tableName,
 PartitionSpec partitionSpec);
public DownloadSession getDownloadSession(String projectName, String tableName,
 PartitionSpec partitionSpec, String id);
public DownloadSession getDownloadSession(String projectName, String tableName, String id);
public UploadSession getUploadSession(String projectName, String tableName, PartitionSpec
 partitionSpec, String id);
public UploadSession getUploadSession(String projectName, String tableName, String id); public
 void setEndpoint(String endpoint);
}
```

TableTunnel接口说明：

- 生命周期：从TableTunnel实例被创建开始，一直到程序结束。
- 作用：提供创建Upload对象和Download对象的方法。

26.6.1.2 UploadSession

接口定义：

```
public class UploadSession {
UploadSession(Configuration conf, String projectName, String tableName, String partitionSpec)
 throws TunnelException;
UploadSession(Configuration conf, String projectName, String tableName, String partitionSpec,
 String uploadId) throws TunnelException;
public void commit(Long[] blocks); public Long[] getBlockList();
public String getId();
public TableSchema getSchema();
public UploadSession.Status getStatus(); public Record newRecord();
public RecordWriter openRecordWriter(long blockId);
public RecordWriter openRecordWriter(long blockId, boolean compress);
}
```

UploadSession接口说明：

- 生命周期：从Upload实例被创建开始，一直到上传结束。
- 作用：创建Upload实例，可以通过调用构造方法创建，也可以通过TableTunnel创建。
  - 请求方式：同步。

  - Server端会为该Upload创建一个session，生成唯一的uploadId标识该Upload，客户端可以通过getId 获取。
- 上传数据：
  - 请求方式：异步。
  - 调用openRecordWriter方法，生成RecordWriter实例，其中参数blockId用于标识此次上传的数据，也描述数据在整个表中的位置。取值范围：[0,20000]。当数据上传失败，可以根据blockId重新上传。
- 查看上传：
  - 请求方式：同步。
  - 调用getStatus可以获取当前Upload状态。
  - 调用getBlockList可以获取成功上传的blockid list，可以和上传的blockid list对比，对失败的blockId重新上传。
- 结束上传：
  - 请求方式：同步。
  - 调用commit(Long[] blocks)方法，参数blocks列表表示已经成功上传的block列表，server端会对该列表进行验证。
  - 该功能是加强对数据正确性的校验，如果提供的block列表与server端存在的block列表不一致，则报错。
- 状态说明：
  - UNKNOWN：server端刚创建一个session时设置的初始值。
  - NORMAL：创建upload对象成功。
  - CLOSING：当调用complete方法（结束上传）时，服务端会先把状态设置为CLOSING。
  - CLOSED：完成结束上传（即把数据移动到结果表所在目录）后。
  - EXPIRED：上传超时。
  - CRITICAL：服务出错。



注意:

- 同一个UploadSession里的blockId不能重复。即对于同一个UploadSession，用一个blockId打开RecordWriter，写入一批数据后，调用close，然后再commit完成后，不可以重新再用该blockId打开另一个RecordWriter写入数据。

- 每一个Block的大小上限为100GB，强烈建议每个Block写入大于64M的数据，否则会严重影响计算性能。
- 每个Session在服务端的生命周期为24小时。
- 上传数据时，建议准备好数据后再调用openRecordWriter开始写入。Writer每写入8KB数据会触发一次网络动作，如果120秒内没有网络动作，服务端将主动关闭连接，届时Writer将不可用，需重新打开一个新的Writer写入。

26.6.1.3 DownloadSession

接口定义：

```
public class DownloadSession {
DownloadSession(Configuration conf, String projectName, String tableName, String partitionSpec) throws TunnelException
DownloadSession(Configuration conf, String projectName, String tableName, String partitionSpec, String downloadId) throws TunnelException
public String getId()
public long getRecordCount() public TableSchema getSchema()
public DownloadSession.Status getStatus()
public RecordReader openRecordReader(long start, long count)
public RecordReader openRecordReader(long start, long count, boolean compress)
}
```

DownloadSession接口说明：

- 生命周期：从Download实例被创建开始，一直到上传结束。
- 作用：创建Download实例，可以通过调用构造方法创建，也可以通过TableTunnel创建。
  - 请求方式：同步。
  - Server端会为该Download创建一个session，生成唯一downloadId标识该Download，客户端可以通过getId获取。该操作开销较大，server端会对数据文件创建索引，当文件数很多时，该时间会比较长；同时server端会返回总Record数，可以根据总Record数启动多个并发同时下载。
- 下载数据：
  - 请求方式：异步。
  - 调用openRecordReader方法，生成RecordReader实例，其中参数Start用于标识本次下载的record的起始位置，从0开始，取值范围是大于等于0；参数Count用于标识本次下载的记录数，取值范围是大于0。
- 查看下载：
  - 请求方式：同步。

- 调用getStatus可以获取当前Download状态。

- 状态说明：
  - UNKNOWN：server端刚创建一个session时设置的初始值。
  - NORMAL：创建Download对象成功。
  - CLOSED：下载结束后。
  - EXPIRED：下载超时。

26.6.2 Tunnel SDK示例

26.6.2.1 简单上传示例

示例如下：

```
import java.io.IOException;
import java.util.Date;
import com.aliyun.odps.Column;
import com.aliyun.odps.Odps;
import com.aliyun.odps.PartitionSpec;
import com.aliyun.odps.TableSchema;
import com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.RecordWriter;
import com.aliyun.odps.tunnel.TableTunnel;
import com.aliyun.odps.tunnel.TunnelException;
import com.aliyun.odps.tunnel.TableTunnel.UploadSession;
public class UploadSample {
  private static String accessId = "<your access id>";
  private static String accessKey = "<your access Key>";
  private static String tunnelUrl = "<your tunnel endpoint>";
  private static String odpsUrl = "<your odps endpoint>";
  private static String project = "<your project>";
  private static String table = "<your table name>";
  private static String partition = "<your partition spec>";
  public static void main(String args[]) {
    Account account = new AliyunAccount(accessId, accessKey);
    Odps odps = new Odps(account);
    odps.setEndpoint(odpsUrl);
    odps.setDefaultProject(project);
    try {
      TableTunnel tunnel = new TableTunnel(odps);
      tunnel.setEndpoint(tunnelUrl);
      PartitionSpec partitionSpec = new PartitionSpec(partition);
      UploadSession uploadSession = tunnel.createUploadSession(project,
          table, partitionSpec);
      System.out.println("Session Status is : "
          + uploadSession.getStatus().toString());
      TableSchema schema = uploadSession.getSchema();
      // 准备数据后打开Writer开始写入数据，准备数据后写入一个Block
     // 单个Block内写入数据过少会产生大量小文件严重影响计算性能，强烈建议每次写入64MB以
上数据(100GB以内数据均可写入同一Block)
      // 可通过数据的平均大小与记录数量大致计算总量即 64MB < 平均记录大小*记录数 < 100GB
      RecordWriter recordWriter = uploadSession.openRecordWriter(0);
```

```
      Record record = uploadSession.newRecord();
      for (int i = 0; i < schema.getColumns().size(); i++) {
        Column column = schema.getColumn(i);
        switch (column.getType()) {
          case BIGINT:
            record.setBigint(i, 1L);
            break;
          case BOOLEAN:
            record.setBoolean(i, true);
            break;
          case DATETIME:
            record.setDatetime(i, new Date());
            break;
          case DOUBLE:
            record.setDouble(i, 0.0);
            break;
          case STRING:
            record.setString(i, "sample");
            break;
          default:
            throw new RuntimeException("Unknown column type: "
                + column.getType());
        }
      }
      for (int i = 0; i < 10; i++) {
        // Write数据至服务端，每写入8KB数据会进行一次网络传输
        // 若120s没有网络传输服务端将会关闭连接，届时该Writer将不可用，需要重新写入
        recordWriter.write(record);
      }
      recordWriter.close();
      uploadSession.commit(new Long[]{0L});
      System.out.println("upload success!");
    } catch (TunnelException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

26.6.2.2 简单下载示例

示例如下：

```
import java.io.IOException; import java.util.Date;
import com.aliyun.odps.Column; import com.aliyun.odps.Odps;
import com.aliyun.odps.PartitionSpec; import com.aliyun.odps.TableSchema; import
 com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount; import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.RecordReader; import com.aliyun.odps.tunnel.TableTunnel;
import com.aliyun.odps.tunnel.TableTunnel.DownloadSession; import
 com.aliyun.odps.tunnel.TunnelException;
public class DownloadSample {
private static String accessId = "<your access id>"; private static String accessKey = "<your
 access Key>";
private static String tunnelUrl = "<your tunnel endpoint>";
private static String odpsUrl = "<your odps endpoint>";
private static String project = "<your project>"; private static String table = "<your table name>";
private static String partition = "<your partition spec>";
public static void main(String args[]) {
```

```
Account account = new AliyunAccount(accessId, accessKey); Odps odps = new Odps(account);
 odps.setEndpoint(odpsUrl);
odps.setDefaultProject(project);
TableTunnel tunnel = new TableTunnel(odps); tunnel.setEndpoint(tunnelUrl);
PartitionSpec partitionSpec = new PartitionSpec(partition); try {
DownloadSession downloadSession = tunnel.createDownloadSession(project, table,
 partitionSpec);
System.out.println("Session Status is : "
+ downloadSession.getStatus().toString());
long count = downloadSession.getRecordCount(); System.out.println("RecordCount is: " + count);
RecordReader recordReader = downloadSession.openRecordReader(0, count);
Record record;
while ((record = recordReader.read()) != null) { consumeRecord(record,
 downloadSession.getSchema());
}
recordReader.close();
} catch (TunnelException e) { e.printStackTrace();
} catch (IOException e1) { e1.printStackTrace();
}
}
private static void consumeRecord(Record record, TableSchema schema) { for (int i = 0; i <
 schema.getColumns().size(); i++) {
Column column = schema.getColumn(i); String colValue = null;
switch (column.getType()) { case BIGINT: {
Long v = record.getBigint(i);
colValue = v == null ? null : v.toString(); break;
}
case BOOLEAN: {
Boolean v = record.getBoolean(i); colValue = v == null ? null : v.toString(); break;
}
case DATETIME: {
Date v = record.getDatetime(i); colValue = v == null ? null : v.toString(); break;
}
case DOUBLE: {
Double v = record.getDouble(i); colValue = v == null ? null : v.toString(); break;
}
case STRING: {
String v = record.getString(i);
colValue = v == null ? null : v.toString(); break;
}
default:
throw new RuntimeException("Unknown column type: "
+ column.getType());
}
System.out.print(colValue == null ? "null" : colValue); if (i != schema.getColumns().size())
System.out.print("\t");
}
System.out.println();
}
}
```

26.6.2.3 多线程上传示例

示例如下：

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Date;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
```

```
import com.aliyun.odps.Column;
import com.aliyun.odps.Odps;
import com.aliyun.odps.PartitionSpec;
import com.aliyun.odps.TableSchema;
import com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.RecordWriter;
import com.aliyun.odps.tunnel.TableTunnel;
import com.aliyun.odps.tunnel.TunnelException;
import com.aliyun.odps.tunnel.TableTunnel.UploadSession;
class UploadThread implements Callable<Boolean> {
  private long id;
  private RecordWriter recordWriter;
  private Record record;
  private TableSchema tableSchema;
     public UploadThread(long id, RecordWriter recordWriter, Record record, TableSchema
 tableSchema) {
    this.id = id;
    this.recordWriter = recordWriter;
    this.record = record;
    this.tableSchema = tableSchema;
  }
  @Override
  public Boolean call() {
    for (int i = 0; i < tableSchema.getColumns().size(); i++) {
      Column column = tableSchema.getColumn(i);
      switch (column.getType()) {
        case BIGINT:
          record.setBigint(i, 1L);
          break;
        case BOOLEAN:
          record.setBoolean(i, true);
          break;
        case DATETIME:
          record.setDatetime(i, new Date());
          break;
        case DOUBLE:
          record.setDouble(i, 0.0);
          break;
        case STRING:
          record.setString(i, "sample");
          break;
        default:
          throw new RuntimeException("Unknown column type: "
              + column.getType());
      }
    }
    try {
      for (int i = 0; i < 10; i++) {
        // Write数据至服务端，每写入8KB数据会进行一次网络传输
        // 若120s没有网络传输服务端将会关闭连接，届时该Writer将不可用，需要重新写入
        recordWriter.write(record);
      }
      recordWriter.close();
    } catch (IOException e) {
      e.printStackTrace();
      return false;
    }
    return true;
  }
}
```

```
public class UploadThreadSample {
  private static String accessId = "<your access id>";
  private static String accessKey = "<your access Key>";
  private static String tunnelUrl = "<your tunnel endpoint>";
  private static String odpsUrl = "<your odps endpoint>";
  private static String project = "<your project>";
  private static String table = "<your table name>";
  private static String partition = "<your partition spec>";
  private static int threadNum = 10;:
  public static void main(String args[]) {
    Account account = new AliyunAccount(accessId, accessKey);
    Odps odps = new Odps(account);
    odps.setEndpoint(odpsUrl);
    odps.setDefaultProject(project);
    try {
      TableTunnel tunnel = new TableTunnel(odps);
      tunnel.setEndpoint(tunnelUrl);
      PartitionSpec partitionSpec = new PartitionSpec(partition);
      UploadSession uploadSession = tunnel.createUploadSession(project,
          table, partitionSpec);
      System.out.println("Session Status is : "
          + uploadSession.getStatus().toString());
      ExecutorService pool = Executors.newFixedThreadPool(threadNum);
      ArrayList<Callable<Boolean>> callers = new ArrayList<Callable<Boolean>>();
      // 准备数据后打开Writer开始多线程写入数据，准备数据后写入
     // 单个Block内写入数据过少会产生大量小文件严重影响计算性能，强烈建议每次写入64MB以
上数据(100GB以内数据均可写入同一Block)
      // 可通过数据的平均大小与记录数量大致计算总量即 64MB < 平均记录大小*记录数 < 100GB
      for (int i = 0; i < threadNum; i++) {
        RecordWriter recordWriter = uploadSession.openRecordWriter(i);
        Record record = uploadSession.newRecord();
        callers.add(new UploadThread(i, recordWriter, record, uploadSession.getSchema()));
      }
      pool.invokeAll(callers);
      pool.shutdown();
      Long[] blockList = new Long[threadNum];
      for (int i = 0; i < threadNum; i++)
        blockList[i] = Long.valueOf(i);
      uploadSession.commit(blockList);
      System.out.println("upload success!");
    } catch (TunnelException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}
```

26.6.2.4 多线程下载示例

示例如下：

```
import java.io.IOException;
import java.util.ArrayList; import java.util.Date; import java.util.List;
import java.util.concurrent.Callable;
import  java.util.concurrent.ExecutionException;  import  java.util.concurrent.ExecutorService;
 import java.util.concurrent.Executors;
import java.util.concurrent.Future;
```

```
import com.aliyun.odps.Column; import com.aliyun.odps.Odps;
import com.aliyun.odps.PartitionSpec; import com.aliyun.odps.TableSchema; import com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount; import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.RecordReader; import com.aliyun.odps.tunnel.TableTunnel;
import com.aliyun.odps.tunnel.TableTunnel.DownloadSession; import com.aliyun.odps.tunnel.TunnelException;
class DownloadThread implements Callable<Long> { private long id;
private RecordReader recordReader; private TableSchema tableSchema;
public DownloadThread(int id,
RecordReader recordReader, TableSchema tableSchema) { this.id = id;
this.recordReader = recordReader; this.tableSchema = tableSchema;
}
@Override
public Long call() {
Long recordNum = 0L; try {
Record record;
while ((record = recordReader.read()) != null) { recordNum++;
System.out.print("Thread " + id + "\t"); consumeRecord(record, tableSchema);
}
recordReader.close();
} catch (IOException e) { e.printStackTrace();
}
return recordNum;
}
private static void consumeRecord(Record record, TableSchema schema) { for (int i = 0; i < schema.getColumns().size(); i++) {
Column column = schema.getColumn(i); String colValue = null;
switch (column.getType()) { case BIGINT: {
Long v = record.getBigint(i);
colValue = v == null ? null : v.toString(); break;
}
case BOOLEAN: {
Boolean v = record.getBoolean(i); colValue = v == null ? null : v.toString(); break;
}
case DATETIME: {
Date v = record.getDatetime(i); colValue = v == null ? null : v.toString(); break;
}
case DOUBLE: {
Double v = record.getDouble(i); colValue = v == null ? null : v.toString(); break;
}
case STRING: {
String v = record.getString(i);
colValue = v == null ? null : v.toString(); break;
}
default:
throw new RuntimeException("Unknown column type: "
+ column.getType());
}
System.out.print(colValue == null ? "null" : colValue); if (i != schema.getColumns().size())
System.out.print("\t");
}
System.out.println();
}
}
public class DownloadThreadSample {
private static String accessId = "<your access id>"; private static String accessKey = "<your access Key>";
private static String tunnelUrl = "<your tunnel endpoint>";
private static String odpsUrl = "<your odps endpoint>";
private static String project = "<your project>"; private static String table = "<your table name>";
private static String partition = "<your partition spec>";
```

```
private static int threadNum = 10; public static void main(String args[]) {
Account account = new AliyunAccount(accessId, accessKey);
Odps odps = new Odps(account); odps.setEndpoint(odpsUrl); odps.setDefaultProject(project);
TableTunnel tunnel = new TableTunnel(odps); tunnel.setEndpoint(tunnelUrl);
PartitionSpec partitionSpec = new PartitionSpec(partition); DownloadSession downloadSession;
try {
downloadSession = tunnel.createDownloadSession(project, table, partitionSpec);
System.out.println("Session Status is : "
+ downloadSession.getStatus().toString());
long count = downloadSession.getRecordCount(); System.out.println("RecordCount is: " + count);
ExecutorService pool = Executors.newFixedThreadPool(threadNum);
 ArrayList<Callable<Long>> callers = new ArrayList<Callable<Long>>();
long start = 0;
long step = count / threadNum;
for (int i = 0; i < threadNum - 1; i++) {
RecordReader recordReader = downloadSession.openRecordReader( step * i, step);
callers.add(new DownloadThread( i, recordReader, downloadSession.getSchema()));
}
RecordReader recordReader = downloadSession.openRecordReader(step * (threadNum - 1),
 count
+ ((threadNum - 1) * step));
callers.add(new DownloadThread( threadNum - 1, recordReader,
 downloadSession.getSchema()));
Long downloadNum = 0L;
List<Future<Long>> recordNum = pool.invokeAll(callers); for (Future<Long> num : recordNum)
downloadNum += num.get(); System.out.println("Record Count is: " + downloadNum);
 pool.shutdown();
} catch (TunnelException e) { e.printStackTrace();
} catch (IOException e) { e.printStackTrace();
} catch (InterruptedException e) { e.printStackTrace();
} catch (ExecutionException e) { e.printStackTrace();
}
}
}
```

26.6.3 附录

26.6.3.1 Tunnel上传下载常见问题

- MaxCompute Tunnel是什么？

  Tunnel是MaxCompute的数据通道，用户可以通过Tunnel向MaxCompute中上传或者下载数据。目前Tunnel仅支持表（不包括视图View）数据的上传下载。

- BlockId是否可以重复？

  同一个UploadSession里的blockId不能重复。即对于同一个UploadSession，用一个blockId打开RecordWriter，写入一批数据后，调用close，然后再commit完成后，写入成功后不可以重新再用该blockId打开另一个RecordWriter写入数据。 Block默认最多20000个，即0-19999。

- Block大小是否存在限制？

  每个block的大小上限为100GB，强烈建议每个Block写入大于64M的数据，每一个Block对应一个文件，小于64MB的文件统称为小文件，小文件过多将会影响使用性能。

- Session是否可以共享使用，存在生命周期吗？

  每个Session在服务端的生命周期为24小时，创建后24小时内均可使用，也可以跨进程/线程共享使用，但是必须保证同一个BlockId没有重复使用。分布式上传可以按照如下步骤进行：**创建Session** > **数据量估算** > **分配**Block（**例如线程1使用**0-100，**线程2使用**100-200） > **准备数据** > **上传数据** > **Commit所有写入成功的Block**。
- 遇到Write/Read超时或IOException怎么处理？

  上传数据时：Writer每写入8KB数据会触发一次网络动作，如果120秒内没有网络动作，服务端将主动关闭连接，届时Writer将不可用，需要重新打开一个新的Writer写入。

  下载数据时：Reader也有类似机制，若长时间没有网络IO会被断开连接，建议Read过程连续进行，中间不穿插其他系统的接口。
- MaxCompute Tunnel目前有哪些语言的SDK？

  MaxCompute Tunnel目前有Java及C++版的SDK。
- MaxCompute Tunnel是否支持多个客户端同时上传同一张表？

  支持。
- MaxCompute Tunnel适合批量上传还是流式上传？

  MaxCompute Tunnel适用于批量上传，不适合流式上传。
- MaxCompute Tunnel上传数据时一定要先存在分区吗？

  是的，Tunnel不会自动创建分区。
- Dship与MaxCompute Tunnel的关系？

  Dship是一个工具，通过MaxCompute Tunnel来上传下载。
- Tunnel upload数据的行为是追加还是覆盖？

  追加的模式。
- Tunnel路由功能是怎么回事？

  路由功能指的是Tunnel SDK通过设置MaxCompute获取Tunnel Endpoint的功能。因此，SDK可以只设置MaxCompute的endpoint来正常工作。
- 用MaxCompute Tunnel上传数据时，每个block的数据量大小多大比较合适？

需要综合考虑网络情况，实时性要求，数据如何使用以及集群小文件等因素。一般情况下，如果数据量较大，是持续上传的模式，可以在64M - 256M； 如果是每天传一次的批量模式，可以设置到1G左右。

- 使用MaxCompute Tunnel 下载，总是提示timeout是什么问题？

  一般是endpoint错误，需要检查Endpoint配置，简单的判断方法是通过telnet等方法检测网络连通性。

- 通过MaxCompute Tunnel下载，报出如下的错误的原因？

  ```
  You have NO privilege ‘odps:Select ‘ on {acs:odps:*:projects/XXX/tables/XXX}. project ‘XXX ‘ is protected
  ```

  该project开启了数据保护功能，用户的操作是从一个项目的数据导向另一个项目，此操作需要该project的owner进行操作。

- 通过MaxCompute Tunnel上传，报出如下的错误的原因？

  ```
  ErrorCode=FlowExceeded, ErrorMessage=Your flow quota is exceeded.**
  ```

  Tunnel对请求的并发进行了控制，默认上传和下载的并发Quota为2000，任何相关的请求发出到结束过程中均会占用一个Quota单位。若出现类似错误，有如下几种建议的解决方案：

  - sleep一下再重试。
  - 将project的tunnel并发quota调大，需要联系管理员评估流量压力。
  - 报告project owner，调查谁占用了大量并发quota并进行控制。

26.6.3.2 Tunnel常见错误码

Tunnel常见的错误码如下所示：

表 96: 常见错误码

ErrorCode	错误原因	处理建议
NoSuchPartition	分区不存在	Tunnel不会创建分区，需要用户自行创建分区后再上传下载
InvalidProjectTable	项目或表名称不合法	检查相关名称
NoSuchProject	项目不存在	检查相关名称
NoSuchTable	表不存在	检查相关名称

StatusConflict	Session过期或者已经Commit过	重新创建Session上传
MalformedDataStream	数据格式错误	通常是网络断开或Schema与Table的不一样导致
InvalidPartitionSpec	分区信息不合法	请检查分区信息，正确值示例类似如下所示：pt='1',ct='2016'
InvalidRowRange	指定行数不合法，通常是超出了数据最大size或者为0	检查相关参数
Unauthorized	账号信息错误	通常是AccessId或AccessKey有误，或本地机器时间与服务端时间差距15分钟以上
DataStoreError	存储错误	建议联系管理员
NoPermission	权限不足通常原因是无相关权限或者设置了IP白名单	检查权限是否正确
MissingPartitionSpec	缺失分区信息，分区表操作必须携带分区信息	补充分区信息
TableModified	表数据在上传下载期间被其他任务改动	重新创建Session重试
FlowExceeded	超出并发Quota限制	建议检查控制并发量，若确实需要增加并发，请联系Project owner或管理员评估流量压力
InvalidResourceSpec	通常为Project/Table/Partition信息与Session不一致	建议检查相关信息并重试
MethodNotAllowed	方法不支持，通常是尝试导出View	目前不支持View导出
InvalidColumnSpec	列信息错误	通常是指定列下载时列名错误
DataVersionConflict	服务正在进行跨集群复制	请稍等后重试
InternalServerError	内部错误	建议重试或联系管理员

26.7 MapReduce使用指南

MaxCompute提供了MapReduce编程接口。用户可以使用MapReduce提供的接口（Java API）编写MapReduce程序来处理MaxCompute中的数据。

26.7.1 概要

26.7.1.1 MapReduce

MapReduce最早是由Google提出的分布式数据处理模型，随后受到了业内的广泛关注，并被大量应用到各种商业场景中。

MapReduce处理数据过程主要分成2个阶段：Map阶段和Reduce阶段。首先执行Map阶段，再执行Reduce阶段。Map和Reduce的处理逻辑由用户自定义实现，但要符合MapReduce框架的约定。

MapReduce处理数据的具体流程如下所示：

1. 在正式执行Map前，需要将输入数据进行**分片**。即将输入数据切分为大小相等的数据块，每一块作为单个Map Worker的输入被处理，以便于多个Map Worker同时工作。
2. 分片完毕后，多个Map Worke即可以开始同时工作。每个Map Worker在读入各自的数据后，进行计算处理，最终输出给Reduce。

   说明: Map Worker在输出数据时，需要为每一条输出数据指定一个Key。这个Key值决定了这条数据将会被发送给哪一个Reduce Worker。Key值和Reduce Worker是多对一的关系，具有相同Key的数据会被发送给同一个Reduce Worker，单个Reduce Worker有可能会接收到多个Key值的数据。
3. 在进入Reduce阶段之前，MapReduce框架会对数据按照Key值排序，使得具有相同Key的数据彼此相邻。如果用户指定了**合并操作**（Combiner），框架会调用Combiner，将具有相同Key的数据进行聚合。

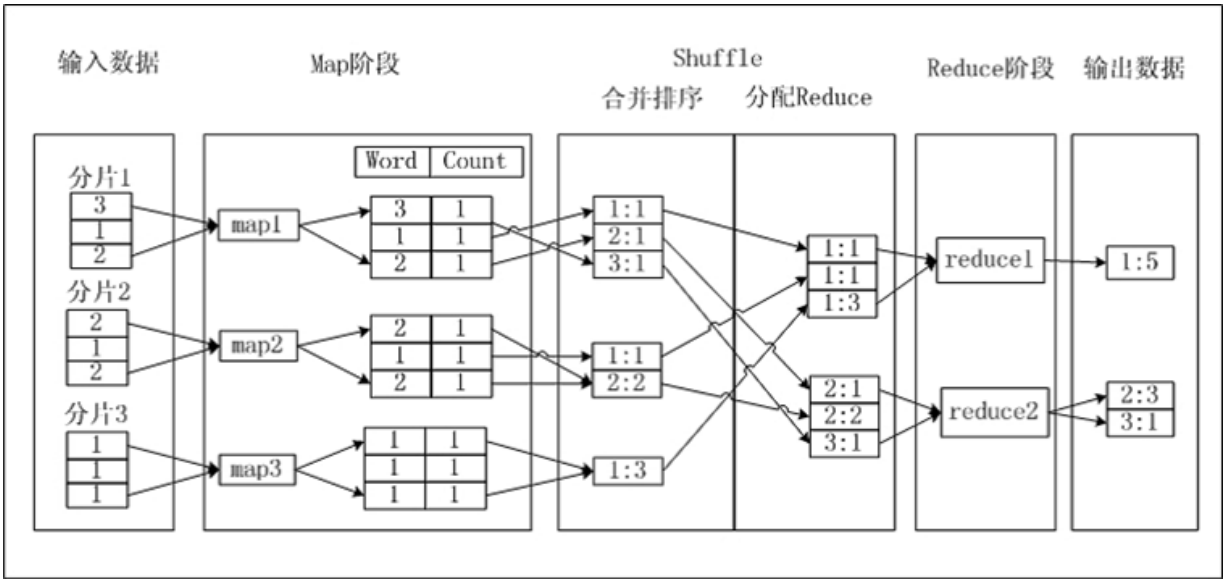
   说明: Combiner的逻辑可以由用户自定义实现。与经典的MapReduce框架协议不同，在MaxCompute中，Combiner的输入、输出参数必须与Reduce保持一致。这部分的处理通常也叫做**洗牌**（Shuffle）。
4. 进入Reduce阶段，相同的Key的数据会到达同一个Reduce Worker。同一个Reduce Worker会接收来自多个Map Worker的数据。每个Reduce Worker会对Key相同的多个数据进行Reduce操作。最后，一个Key的多条数据经过Reduce的作用后，将变成一个值。



说明: 上述内容仅是对MapReduce框架的简单介绍，更多相关信息请查阅其他资料。

下面将以WordCount为例，解释MaxCompute MapReduce各个阶段的相关概念。

假设存在一个文本a.txt，文本内每行是一个数字，用户要统计每个数字出现的次数。 文本内的数字称为Word，数字出现的次数称为Count。如果MaxCompute Mapreduce完成这一功能，需要经历下图描述的几个步骤：



1. 对文本a.txt进行分片，将每片内的数据作为单个Map Worker的输入。
2. Map处理输入，每获取一个数字，将数字的Count设置为1，并将**Word|Count**对输出，此时以Word作为输出数据的Key。
3. 在Shuffle阶段前期，先对每个Map Worker的输出，按照Key值，即Word值排序。排序后进行Combine操作，即将Key值（Word值）相同的Count累加， 构成一个新的**Word|Count**对。此过程被称为合并排序。
4. 在Shuffle阶段后期，数据被发送到Reduce端。Reduce Worker收到数据后依赖Key值再次对数据排序。
5. Reduce阶段，每个Reduce Worker对数据进行处理时，采用与Combiner相同的逻辑，将Key值（Word 值）相同的Count累加，得到输出结果。

> **说明:** 由于MaxCompute的所有数据都被存放在表中，因此MaxCompute MapReduce的输入、输出只能是表，不允许用户自定义输出格式，不提供类似文件系统的接口。

26.7.1.2 扩展MapReduce

传统的MapReduce模型要求每一轮MapReduce操作之后，数据必须落地到分布式文件系统上（例如HDFS或MaxCompute表）。而一般的MapReduce应用通常由多个MapReduce作业组成，每个作

业结束之后需要写入磁盘。接下去的Map任务很多情况下只是读一遍数据，为后续的Shuffle阶段做准备，这样其实造成了冗余的IO操作。

MaxCompute的计算调度逻辑可以支持更复杂编程模型，针对上面的情况，可以在Reduce后面直接执行下一次的Reduce操作，而不需要中间插入一个Map操作。 基于此，MaxCompute提供了扩展的MapReduce模型，即可以支持Map后连接任意多个Reduce操作，例如Map-Reduce-Reduce。

Hadoop Chain Mappper/Reducer也支持类似的串行化Map或Reduce操作，但和MaxCompute的扩展MapReduce（MR2）模型有本质的区别。 因为Chain Mapper/Reducer还是基于传统的MapReduce模型，只是可以在原有的Mapper或Reducer后面再增加一个或多个Mapper操作（不允许增加Reducer）。带来的好处是用户可以复用之前的Mapper业务逻辑，可以把一个Map或Reduce拆成多个Mapper阶段，但本质上并没有改变底层的调度和IO模型。

26.7.2 功能介绍

26.7.2.1 运行命令

MaxCompute客户端提供一个jar命令用于运行MapReduce作业。

命令格式如下：

```
Usage: jar [<GENERIC_OPTIONS>] <MAIN_CLASS> [ARGS]
-conf <configuration_file> Specify an application configuration file
-classpath <local_file_list> classpaths used to run mainClass
-D <name>=<value> Property value pair, which will be used to run mainClass
-local Run job in local mode
-resources <resource_name_list> file/table resources used in mapper or reducer, seperate by
 comma
```

参数说明：

- -conf <configuration file> ：指定JobConf配置文件。
- -classpath <local_file_list>：本地执行时的classpath，主要用于指定main函数所在的jar包的本地路径（包含相对路径和绝对路径）。
- -D <prop_name>=<prop_value> ：本地执行时，<mainClass>的java属性，可以定义多个。
- -local：以本地模式执行MapReduce作业，主要用于程序调试。
- -resources <resource_name_list>： MapReduce作业运行时使用的资源声明。一般情况下，resource_name_list中需要指定Map/Reduce函数所在的资源名称。

> **注意:** 如果用户在Map/Reduce函数中读取了其他MaxCompute资源，这些资源名称也需要被添加到resource_name_list中。资源之间使用逗号分隔，跨项目空间

使用资源时，需要在前面加上：PROJECT_NAME/resources/，例如：-resources otherproject/resources/resfile。

说明: 上述的可选参数均包含在**<GENERIC_OPTIONS>**中。

用户可以通过-conf选项指定JobConf配置文件。该文件可以影响SDK中JobConf的设置。关于JobConf 的介绍请参考MapReduce核心接口的介绍。下面将给出一个JobConf配置文件的示例。

示例如下：

```
<configuration>
<property>
<name>import.filename</name>
<value>resource.txt</value>
</property>
</configuration>
```

在上述示例中，通过JobConf配置文件定义一个名为import.filename的变量，该变量的值为resource.txt。用户可以在MapReduce程序中通过JobConf接口获取该变量的值。用户通过SDK中JobConf接口可以达到相同的目的。

示例如下：

```
jar -resources mapreduce-examples.jar -classpath mapreduce-examples.jar
org.alidata.odps.mr.examples.WordCount wc_in wc_out
add file data/src.txt
jar -resources src.txt,mapreduce-examples.jar -classpath mapreduce-examples.jar
 org.alidata.odps.mr.examples.WordCount wc_in wc_out
add file data/a.txt
add table wc_in as test_table add jar work.jar
jar -conf odps-mapred.xml -resources a.txt,test_table,work.jar
-classpath work.jar:otherlib.jar
-D import.filename=resource.txt org.alidata.odps.mr.examples.WordCount args
```

26.7.2.2 基本概念

26.7.2.2.1 Map/Reduce

Map及Reduce分别支持对应的map/reduce方法，setup及cleanup方法。setup方法在map/reduce方法之前调用，每个Worker调用且仅调用一次。 cleanup方法在map/reduce方法之后调用，每个Worker调用且仅调用一次。

说明: 详细使用示例请参考：示例程序。

26.7.2.2.2 排序

支持将map输出的key record中的某几列作为排序（Sort）列，不支持用户自定义的比较器（comparator）。用户可以在排序列中选择某几列作为Group列，不支持用户自定义的Group比较器。Sort列一般用来对用户数据进行排序，而Group列一般用来进行二次排序。

说明: 详细使用示例请参考：二次排序源代码。

26.7.2.2.3 哈希

支持设置哈希（partition）列及用户自定义哈希函数（partitioner）。哈希列的使用优先级高于自定义哈希函数。哈希函数用于将map端的输出数据按照哈希逻辑分配到不同的Reduce Worker上。

26.7.2.2.4 归并

归并（Combiner）函数将Shuffle阶段相邻的Record进行归并。用户可以根据不同的业务逻辑选择是否使用归并函数。归并函数是MapReduce计算框架的一种优化，通常情况下Combiner的逻辑与reduce相同。当map输出数据后，框架会在map端对相同key值的数据进行本地的归并操作。

26.7.2.2.5 输入与输出

- MaxCompute MapReduce的输入、输出支持MaxCompute内置类型的Bigint、Double、String、Datetime、Boolean、Decimal、Tinyint、Smallint、 Int、Float、Varchar、Timestamp、Binary、Array、Map及Struct类型，不支持用户自定义类型。
- 接受多表输入，且输入表的Schema可以不相同。在map函数中用户可以获取当前Record对应的Table信息。
- 输入可以为空，不支持视图（View）作为输入。
- Reduce接受多路输出，可以输出到不同表，或者同一张表的不同分区。不同输出的Schema可以不同。不同输出间通过label进行区分， 默认输出不必加 lablel，但目前不接受没有输出的情况。

说明: 备注：详细使用示例请参考 ：示例程序。

26.7.2.2.6 资源读取

允许用户在map/reduce中读取MaxCompute资源。map/reduce的任意Worker都会将资源加载到内存中供用户代码使用。

说明: 详细使用示例请参考：资源使用示例。

备注：

26.7.2.2.7 本地运行

用户通过在jar命令中设置–local参数，在本地模拟MapReduce的运行过程，从而进行本地调试。本地运行时，客户端会从MaxCompute中下载本地调试所需要的输入表的元信息、数据，所需要的资源以及输出表的元信息，并将这些信息保存到一个名为warehouse的本地目录中。在程序运行结束后，会将计算结果输出到warehouse目录内的一个文件中。如果本地的warehouse目录下已经下载了输入表及被引用的资源，在下一次运行时，会直接引用warehouse下的数据及文件，而不会重复下载。

在本地运行过程中，仍然会启动多个Map及Reduce进程处理数据，但这些进程不是并发运行，而是依次串行运行。此外这个模拟运行过程与真正的分布式运行有如下差别：

- 输入表行数限制：目前，最多只会下载100行数据。
- 资源的使用：在分布式环境中，MaxComput会限制引用资源的大小，详情请参考应用限制。但在本地运行环境中，不会有资源大小的限制。
- 安全限制：MaxCompute MapReduce及UDF程序在分布式环境中运行时受到Java沙箱的限制。但在本地运行时，则没有此限制。

下面将给出一个简单的本地运行示例。

示例如下：

```
odps@my_project> jar -l com.aliyun.odps.mapred.example.WordCount wc_in wc_out;
 Summary:
counters: 10
map-reduce framework combine_input_groups=2 combine_output_records=2
map_input_bytes=4 map_input_records=1 map_output_records=2
map_output_[wc_out]_bytes=0 map_output_[wc_out]_records=0 reduce_input_groups=2
reduce_output_[wc_out]_bytes=8 reduce_output_[wc_out]_records=2
OK
```

说明: 关于WordCount示例的介绍请参考： *WordCount*代码示例。

如果用户是第一次运行本地调试命令，命令成功结束后，会在当前路径下看到一个名为warehouse的路径。 warehouse的目录结构如下所示：

```
<warehouse>
  |___my_project(项目空间目录)
      |___ <__tables__>
      |     |_wc_in(表数据目录)
      |     |     |___ data(文件)
      |     |     |
      |     |     |___ <__schema__> (文件)
      |     |_wc_out ( 表数据目录 )
      |           |___ data(文件)
      |           |
      |           |___ <__schema__> (文件)
      |
      |___ <__resources__>
            |
            |__table_resource_name (表资源)
            |       |___<__ref__>
            |
            |__ file_resource_name ( 文件资源 )
```

my_project的同级目录表示项目空间。wc_in及wc_out表示数据表，用户在jar命令中读写的表文件数据会被下载到这一级的目录下。 schema文件中的内容表示表的元信息，其文件格式定义为：

```
project=local_project_name
table=local_table_name
columns=col1_name:col1_type,col2_name:col2_type
partitions=p1:STRING,p2:BIGINT
-- 本示例中不需要此字
```

说明: 其中，列名与列类型使用冒号“:”分隔，列与列之间使用逗号“,”分隔。schema文件的最前面需要声明Project名字及Table名字，即projectname.tablename，使用逗号与列的定义做分隔。data 文件表示表的数据。列的数量及数据必须与schema文件的定义相符，不能多列或者少列，列之间使用逗号分隔。

wc_in的schema文件内容示例：

```
my_project.wc_in,key:STRING,value:STRING
```

对应的data文件内容示例：

```
0,2
```

说明: 客户端会从MaxCompute中下载表的元信息及部分数据内容并保存到上述两个文件中。如果再次运行这个示例，将直接使用wc_in目录下的数据，不会再次下载。需要特殊声明的是，从MaxCompute中下载数据的功能只在MapReduce的本地运行模式下

才支持，在Eclipse开发插件中进行本地调试时，不支持将MaxCompute的数据下载到本地。

wc_out的schema文件内容示例：

```
my_project.wc_out,key:STRING,cnt:BIGINT
```

对应的data文件内容示例：

```
0,1
2,1
```

说明: 客户端会从MaxCompute下载wc_out表的元信息，并保存到schema文件中。而data文件的内容是在本地运行后，生成的结果数据文件。用户也可以自行编辑schema及data文件，而后将这两个文件放置在对应的表目录下。在本地运行时，客户端检测到表目录已经存在， 则不会从MaxCompute中下载这个表的信息。本地的表目录可以是MaxCompute中不存在的表。

功能对比：

功能描述	Hadoop MapReduce	MaxCompute MapReduce
任务进度汇报	Map通过计算数据的读入量，Reduce通过不同phase的进度估计任务总体进度。客户端可实时获取当前进度。不受用户控制。	支持任务进度实时汇报。但实现方式不同。
统计信息	允许用户自定义统计信息，并进行实时总更新。信息实时更新行为不受用户控制。	不支持运行时更新统计信息。在任务结束时，统一进行计算汇总。
文件压缩	支持指定是否进行压缩存储。	不支持。MaxCompute中用户无法直接操作文件存储。
推测执行	/	默认开启，不可设置。
任务结束通知	当任务结束时，服务端会通知客户端。	不支持。因此用户在使用SDK提交任务时，只能不断轮询任务状态，直到任务结束。

26.7.3 SDK介绍

在本章节，会对较为常用的MapReduce核心接口做简短介绍。

26.7.3.1 核心接口

表 97: 核心接口

主要接口	描述
MapperBase	用户自定义的Map函数需要继承自此类。处理输入 表的记录对 象，加工处理成键值对集合输出到Reduce阶段，或者不经过Reduce阶段直接输出结果记录到结果表。不经过Reduce阶段而直接输出计算结果的作业，也可称之为Map-Only作业。
ReducerBase	用户自定义的Reduce函数需要继承自此类。对与一个键（Key）关联的一组数值集（Values）进行归约计算。
TaskContext	是MapperBase及ReducerBase多个成员函数的输入参数之一。含有任务运行的上下文信息。
JobClient	用于提交和管理作业，提交方式包括阻塞（同步）方式及非阻塞（异步）方式。
RunningJob	作业运行时对象，用于跟踪运行中的MapReduce作业实例。
JobConf	描述一个MapReduce任务的配置，通常在主程序（main函数）中定义JobConf对象，然后通过JobClient提交作业给MaxCompute服务。

26.7.3.1.1 MapperBase

主要函数接口如表所示：

主要接口	描述
void cleanup(TaskContext context)	在Map阶段结束时，map方法之后调用。
void map(long key, Record record, TaskContext context)	map方法，处理输入表的记录。
void setup(TaskContext context)	在Map阶段开始时，map方法之前调用。

26.7.3.1.2 ReducerBase

主要函数接口如表所示：

主要接口	描述
void cleanup(TaskContext context)	在Reduce阶段结束时，reduce方法之后调用。
void reduce(Record key, Iterator<Record > values, TaskContext context)	reduce方法，处理输入表的记录。
void setup(TaskContext context)	在Reduce阶段开始时，reduce方法之前调用。

26.7.3.1.3 TaskContext

主要函数接口如表所示：

主要接口	描述
TableInfo[] getOutputTableInfo()	获取输出的表信息。
Record createOutputRecord()	创建默认输出表的记录对象。
Record createOutputRecord(String label)	创建给定label输出表的记录对象。
Record createMapOutputKeyRecord()	创建Map输出Key的记录对象。
Record createMapOutputValueRecord()	创建Map输出Value的记录对象。
void write(Record record)	写记录到默认输出，用于Reduce端写出数据，可以在Reduce端多次调用。
void write(Record record, String label)	写记录到给定标签输出，用于Reduce端写出数据 ，可以在Reduce端多次调用。
void write(Record key, Record value)	写记录到中间结果，用于Map端写出数据，可以在Map端多次调用。
BufferedInputStream readResourceFileAsStream(String resourceName)	读取文件类型资源。
Iterator<Record > readResourceTable(String resourceName)	读取表类型资源。
Counter getCounter(Enum<? > name)	获取给定名称的Counter对象。
Counter getCounter(String group, String name)	获取给定组名和名称的Counter对象。
void progress()	向MapReduce框架报告心跳信息。如果用户方法处理时间很长，且中间没有调用框架，可以调用这个方法避免task超时，框架默认600秒超时。

说明： TaskContext接口中提供了progress功能，但此功能是防止Worker长时间运行未结束，被框架误认为超时而强制停止的情况出现。 这个接口更类似于向框架发送心跳信息，并不是用来汇报Worker进度。MaxCompute MapReduce默认Worker超时时间为10分钟（系统默认配置，不受用户控制）， 如果超过10分钟，Worker仍然没有向框架发送心跳（调用progress接口），框架会强制停止该Worker，MapReduce任务失败退出。因此， 建议用户在Mapper/Reducer函数中，定期调用progress接口，防止框架认为Worker超时，误杀任务。

26.7.3.1.4 JobConf

主要函数接口如表所示：

主要接口	描述
void setResources(String resourceNames)	声明本作业使用的资源。只有声明的资源才能在运行 Mapper/Reducer时通过TaskContext对象读取。
void setMapOutputKeySchema(Column[] schema)	设置Mapper输出到Reducer的Key属性。
void setMapOutputValueSchema(Column[] schema)	设置Mapper输出到Reducer的Value属性。
void setOutputKeySortColumns(String[] cols)	设置Mapper输出到Reducer的Key排序列。
void setOutputGroupingColumns(String[] cols)	设置Key分组列。
void setMapperClass(Class<? extends Mapper > > > theClass)	设置作业的Mapper函数。
void setPartitionColumns(String[] cols)	设置作业指定的分区列，默认是Mapper输出Key的所有列。
void setReducerClass(Class<? extends Reducer > theClass)	设置作业的Reducer。
void setCombinerClass(Class<? extends Reducer > theClass)	设置作业的combiner。在Map端运行，作用类似于单个Map 对本地的相同Key值做Reduce。
void setSplitSize(long size)	设置输入分片大小，单位 MB，默认值640。
void setNumReduceTasks(int n)	设置Reducer任务数，默认为Mapper任务数的1/4。
void setMemoryForMapTask(int mem)	设置Mapper任务中单个Worker的内存大小，单位：MB，默认值2048。

主要接口	描述
void setMemoryForReduceTask(int mem)	设置Reducer任务中单个Worker的内存大小，单位：MB，默认值 2048。
void setOutputSchema(Column[] schema, String label)	设置指定label的输出属性。多路输出时，每一路输出对应一个label。

说明:

- 通常情况下，GroupingColumns包含在KeySortColumns中，KeySortColumns包含在Key中。
- 在Map端，Mapper输出的Record会根据设置的PartitionColumns计算哈希值，决定分配到哪个Reducer，会根据KeySortColumns对Record进行排序。
- 在Reduce端，输入Records在按照KeySortColumns排序好后，会根据GroupingColumns指定的列对输入的Records进行分组，即会顺序遍历输入的Records，把GroupingColumns所指定列相同的Records作为一次reduce函数调用的输入。

26.7.3.1.5 JobClient

主要函数接口如表所示：

主要接口	描述
static RunningJob runJob(JobConf job)	阻塞（同步）方式提交MapReduce作业后立即返回。
static RunningJob submitJob(JobConf job)	非阻塞（异步）方式提交MapReduce作业后立即返回。

26.7.3.1.6 RunningJob

主要函数接口如表所示：

主要接口	描述
String getInstanceID()	获取作业运行实例ID，用于查看运行日志和作业管理。
boolean isComplete()	查询作业是否结束。
boolean isSuccessful()	查询作业实例是否运行成功。

主要接口	描述
void waitForCompletion()	等待直至作业实例结束。一般使用于异步方式提交的作业。
JobStatus getJobStatus()	查询作业实例运行状态。
void killJob()	结束此作业。
Counters getCounters()	获取Conter信息。

26.7.3.1.7 InputUtils

主要函数接口如表所示：

主要接口	描述
static void addTable(TableInfo table, JobConf conf)	添加表table到任务输入，可以被调用多次，新加入的表以append方式添加到输入队列中。
static void setTables(TableInfo [] tables, JobConf conf)	添加多张表到任务输入中。

26.7.3.1.8 OutputUtils

主要函数接口如表所示：

主要接口	描述
static void addTable(TableInfo table, JobConf conf)	添加表table到任务输出，可以被调用多次，新加入的表以append方式添加到输出队列中。
static void setTables(TableInfo [] tables, JobConf conf)	添加多张表到任务输出中。

26.7.3.1.9 Pipeline

Pipeline是MR2的主体类。可以通过Pipeline.builder构建一个Pipeline。Pipeline的主要接口如下：

```
public Builder addMapper(Class<? extends Mapper> mapper)
public Builder addMapper(Class<? extends Mapper> mapper,Column[] keySchema, Column[]
 valueSchema, String[] sortCols, SortOrder[] order, String[] partCols,Class<? extends Partitioner>
 theClass, String[] groupCols)
public Builder addReducer(Class<? extends Reducer> reducer)
public Builder addReducer(Class<? extends Reducer> reducer,Column[] keySchema, Column[]
 valueSchema, String[] sortCols, SortOrder[] order, String[] partCols,Class<? extends Partitioner>
 theClass, String[] groupCols)
public Builder setOutputKeySchema(Column[] keySchema)
public Builder setOutputValueSchema(Column[] valueSchema)
public Builder setOutputKeySortColumns(String[] sortCols)
```

```
public Builder setOutputKeySortOrder(SortOrder[] order)
public Builder setPartitionColumns(String[] partCols)
public Builder setPartitionerClass(Class<? extends Partitioner> theClass)
public Builder setOutputGroupingColumns(String[] cols)
```

示例如下：

```
Job job = new Job();
Pipeline pipeline = Pipeline.builder()
.addMapper(TokenizerMapper.class)
.setOutputKeySchema(
new Column[] { new Column("word", OdpsType.STRING) })
.setOutputValueSchema(
new Column[] { new Column("count", OdpsType.BIGINT) })
.addReducer(SumReducer.class)
.setOutputKeySchema(
new Column[] { new Column("count", OdpsType.BIGINT) })
.setOutputValueSchema(
new Column[] { new Column("word", OdpsType.STRING),
new Column("count", OdpsType.BIGINT) })
.addReducer(IdentityReducer.class).createPipeline();
job.setPipeline(pipeline); job.addInput(...)
job.addOutput(...) job.submit();
```

如上所示，用户可以在main函数中构建一个Map后连续接两个Reduce的MapReduce任务。如果用户比较熟悉MapReduce的基础功能，可以轻松的使用MR2 。我们也建议用户在使用MR2 功能之前，先了解MapReduce的基础用法，即通过JobConf 完成MapReduce任务的配置。当然，JobConf仅能够配置Map后接单Reduce的MapReduce任务。

26.7.3.1.10 数据类型

MapReduce支持的数据类型有：

Bigint、Double、String、Datetime、Boolean、Decimal、Tinyint、Smallint、 Int、Float、Varchar、Timestamp、Binary、Array、Map及Struct类型。MaxCompute数据类型与Java类型的对应关系如下：

MaxCompute Type	Java Type
Tinyint	java.lang.Byte
Smallint	java.lang.Short
Int	java.lang.Integer
Bigint	java.lang.Long
Float	java.lang.Float
Double	java.lang.Double
Decimal	java.math.BigDecimal

MaxCompute Type	Java Type
Boolean	java.lang.Boolean
String	java.lang.String
Varchar	com.aliyun.odps.data.Varchar
Binary	com.aliyun.odps.data.Binary
Datetime	java.util.Date
Timestamp	java.sql.Timestamp
Array	java.util.List
Map	java.util.Map
Struct	com.aliyun.odps.data.Struct

26.7.4 应用限制

目前，MapReduce的使用限制包括：

- 单个Map或Reduce Worker占用memory默认为2048MB，范围[256MB, 12GB]。
- 单个任务引用的资源数量不超过256个，分区表按照一个单位计算。
- 单个任务的输入路数不能超过1024，单个任务的输出路数不能超过256路。
- 单个任务中自定义Counter的数量不能超过64。
- 单个job的map instance个数由框架根据split size计算得出，如果没有输入表，可以通过odps.stage.mapper.num直接设置，范围[1, 100000]。
- 单个job默认reduce instance个数为map instance个数的1/4，用户可以设置reduce instance个数，范围[0, 2000]。可能出现如下的情况：reduce处理的数据量会比map大很多倍，导致reduce阶段比较慢，而reduce只能最多起2000个。
- 单个map instance或reduce instance失败重试次数为3，一些不可重试的异常会直接导致作业失败。
- 本地运行模式下，Map Worker个数不能超过100；Reduce Worker个数不能超过100；默认一路输入下载记录数100。
- 单个Map或Reduce Worker重复读一个资源次数限制<=64次。
- 单个任务引用的资源总计字节数大小不超过2G。
- 框架会参考设置的split size值来划分map，决定map的个数。
- MaxCompute表string列内容长度不允许超过8MB。

- map或者reduce worker在无数据读写且没有通过context.progress()主动发送心跳的情况下的超时时间，默认值是600s。

26.7.5 示例程序

26.7.5.1 WordCount示例

示例如下：

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException; import java.util.Iterator;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
public class WordCount {
public static class TokenizerMapper extends MapperBase {
private Record word;
private Record one;
@Override
public void setup(TaskContext context) throws IOException {
word = context.createMapOutputKeyRecord();
one = context.createMapOutputValueRecord();
one.set(new Object[] { 1L });
System.out.println("TaskID:" + context.getTaskID().toString());
}
@Override
public void map(long recordNum, Record record, TaskContext context) throws IOException {
for (int i = 0; i < record.getColumnCount(); i++) {
word.set(new Object[] { record.get(i).toString() });
context.write(word, one);
}
}
}
/**
*A combiner class that combines map output by sum them.
**/
public static class SumCombiner extends ReducerBase {
private Record count;
@Override
public void setup(TaskContext context) throws IOException {
count = context.createMapOutputValueRecord();
}
@Override
public void reduce(Record key, Iterator<Record> values, TaskContext context)
throws IOException {
long c = 0;
while (values.hasNext()) {
Record val = values.next();
c += (Long) val.get(0);
}
count.set(0, c);
```

```
context.write(key, count);
}
}
/**
* A reducer class that just emits the sum of the input values.
**/
public static class SumReducer extends ReducerBase {
private Record result = null;
@Override
public void setup(TaskContext context) throws IOException { result =
 context.createOutputRecord();
}
@Override
public void reduce(Record key, Iterator<Record> values, TaskContext context)
throws IOException {
long count = 0;
while (values.hasNext()) {
Record val = values.next();
count += (Long) val.get(0);
}
result.set(0, key.get(0));
result.set(1, count);
context.write(result);
}
}
public static void main(String[] args)
throws Exception {
if (args.length != 2) {
System.err.println("Usage: WordCount <in_table> <out_table>");
System.exit(2);
}
JobConf job = new JobConf();
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(SumCombiner.class);
job.setReducerClass(SumReducer.class);
job.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
job.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
JobClient.runJob(job);
}
}
```

26.7.5.2 MapOnly示例

对于MapOnly的作业，Map直接将 < Key, Value > 信息输出到MaxCompute的表中。用户只需要指定输出表即可，不再需要指定Map输出的Key/Value元信息。

示例如下：

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.SchemaUtils;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
```

```
import com.aliyun.odps.data.TableInfo;
public class MapOnly {
public static class MapperClass extends MapperBase {
@Override
public void setup(TaskContext context)
throws IOException {
boolean is = context.getJobConf().getBoolean("option.mapper.setup", false);
if (is) {
Record result = context.createOutputRecord();
result.set(0, "setup");
result.set(1, 1L); context.write(result);
}
}
@Override
public void map(long key, Record record, TaskContext context) throws IOException {
boolean is = context.getJobConf().getBoolean("option.mapper.map", false);
if (is) {
Record result = context.createOutputRecord();
result.set(0, record.get(0));
result.set(1, 1L); context.write(result);
}
}
@Override
public void cleanup(TaskContext context) throws IOException {
boolean is = context.getJobConf().getBoolean("option.mapper.cleanup", false);
if (is) {
Record result = context.createOutputRecord();
result.set(0, "cleanup");
result.set(1, 1L); context.write(result);
}
}
}
public static void main(String[] args) throws Exception {
if (args.length != 2 && args.length != 3) {
System.err.println("Usage: OnlyMapper <in_table> <out_table> [setup|map|cleanup]");
System.exit(2);
}
JobConf job = new JobConf();
job.setMapperClass(MapperClass.class);
job.setNumReduceTasks(0);
InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
if (args.length == 3) {
String options = new String(args[2]);
if (options.contains("setup")) {
job.setBoolean("option.mapper.setup", true);
}
if (options.contains("map")) {
job.setBoolean("option.mapper.map", true);
}
if (options.contains("cleanup")) {
job.setBoolean("option.mapper.cleanup", true);
}
}
JobClient.runJob(job);
}
```

```
}
```

26.7.5.3 多路输入输出示例

目前MaxCompute支持多表的输入及输出。但在多表输入中，多张表的列数量及类型定义必须相同。多表输出中列数量及列类型则可以不同。

示例如下：

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException; import java.util.Iterator;
import java.util.LinkedHashMap;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
*Multi input & output example.
*
*To run: jar -resources odps-mapred-example-0.12.0.jar
 com.aliyun.odps.mapred.open.example.MultipleInOut
*mr_src,mr_src1,mr_srcpart|pt=1,mr_srcpart|pt=2/ds=2
*mr_multiinout_out1,mr_multiinout_out2|a=1/b=1|out1,mr_multiinout_out2|a=2/b=2|out2;
*
**/
public class MultipleInOut {
public static class TokenizerMapper extends MapperBase {
private Record word;
private Record one;
@Override
public void setup(TaskContext context) throws IOException {
word = context.createMapOutputKeyRecord();
one = context.createMapOutputValueRecord();
one.set(new Object[] { 1L });
}
@Override
public void map(long recordNum, Record record, TaskContext context)
throws IOException {
for (int i = 0; i < record.getColumnCount(); i++) {
word.set(new Object[] { record.get(i).toString() });
context.write(word, one);
}
}
}
public static class SumReducer extends ReducerBase {
private Record result;
private Record result1;
private Record result2;
@Override
public void setup(TaskContext context) throws IOException {
result = context.createOutputRecord();
result1 = context.createOutputRecord("out1");
result2 = context.createOutputRecord("out2");
```

```
}
@Override
public void reduce(Record key, Iterator<Record> values, TaskContext context)
throws IOException { long count = 0;
while (values.hasNext()) {
Record val = values.next();
count += (Long) val.get(0);
}
long mod = count % 3;
if (mod == 0) {
result.set(0, key.get(0));
result.set(1, count);
//不指定label，输出的默认(default)输出
context.write(result);
} else if (mod == 1) {
result1.set(0, key.get(0));
result1.set(1, count);
context.write(result1, "out1");
} else {
result2.set(0, key.get(0));
result2.set(1, count);
context.write(result2, "out2");
}
}
@Override
public void cleanup(TaskContext context) throws IOException {
Record result = context.createOutputRecord();
result.set(0, "default");
result.set(1, 1L);
context.write(result);
Record result1 = context.createOutputRecord("out1");
result1.set(0, "out1");
result1.set(1, 1L); context.write(result1, "out1");
Record result2 = context.createOutputRecord("out2");
result2.set(0, "out1");
result2.set(1, 1L); context.write(result2, "out2");
}
}
public static LinkedHashMap<String, String> convertPartSpecToMap( String partSpec) {
LinkedHashMap<String, String> map = new LinkedHashMap<String, String>();
if (partSpec != null && !partSpec.trim().isEmpty()) {
String[] parts = partSpec.split("/");
for (String part : parts) {
String[] ss = part.split("=");
if (ss.length != 2) {
throw new RuntimeException("ODPS-0730001: error part spec format: "+ partSpec);
}
map.put(ss[0], ss[1]);
}
}
return map;
}
public static void main(String[] args) throws Exception {
String[] inputs = null;
String[] outputs = null;
if (args.length == 2) {
inputs = args[0].split(",");
outputs = args[1].split(",");
} else {
System.err.println("MultipleInOut in... out...");
System.exit(1);
}
```

```
JobConf job = new JobConf();
job.setMapperClass(TokenizerMapper.class);
job.setReducerClass(SumReducer.class);
job.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
job.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
//解析用户的输入表字符串
for (String in : inputs) { String[] ss = in.split("\\|");
if (ss.length == 1) {
InputUtils.addTable(TableInfo.builder().tableName(ss[0]).build(), job);
} else if (ss.length == 2) {
LinkedHashMap<String, String> map = convertPartSpecToMap(ss[1]);
InputUtils.addTable(TableInfo.builder().tableName(ss[0]).partSpec(map).build(), job);
} else {
System.err.println("Style of input: " + in + " is not right");
System.exit(1);
}
}
//解析用户的输出表字符串
for (String out : outputs) { String[] ss = out.split("\\|");
if (ss.length == 1) {
OutputUtils.addTable(TableInfo.builder().tableName(ss[0]).build(), job);
} else if (ss.length == 2) {
LinkedHashMap<String, String> map = convertPartSpecToMap(ss[1]);
OutputUtils.addTable(TableInfo.builder().tableName(ss[0]).partSpec(map).build(), job);
} else if (ss.length == 3) {
if (ss[1].isEmpty()) {
LinkedHashMap<String, String> map = convertPartSpecToMap(ss[2]);
OutputUtils.addTable(TableInfo.builder().tableName(ss[0]).partSpec(map).build(), job);
} else {
LinkedHashMap<String, String> map = convertPartSpecToMap(ss[1]);
OutputUtils.addTable(TableInfo.builder().tableName(ss[0]).partSpec(map).label(ss[2]).build(),
 job);
}} else {
System.err.println("Style of output: " + out + " is not right");
System.exit(1);
}
}
JobClient.runJob(job);
}
}
```

26.7.5.4 多任务示例

示例如下：

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.RunningJob;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
* MultiJobs
```

```
*
* Running multiple job
*
* To run: jar -resources > multijobs_res_table,odps-mapred-example-0.12.0.jar
* com.aliyun.odps.mapred.open.example.MultiJobs mr_multijobs_out;
*
**/
public class MultiJobs {
public static class InitMapper extends MapperBase {
@Override
public void setup(TaskContext context) throws IOException { Record record = context.createOutputRecord();
long v = context.getJobConf().getLong("multijobs.value", 2);
record.set(0, v);
context.write(record);
}
}
public static class DecreaseMapper extends MapperBase {
@Override
public void cleanup(TaskContext context) throws IOException {
//从JobConf中获取main函数中定义的变量值
long expect = context.getJobConf().getLong("multijobs.expect.value", -1);
long v = -1;
int count = 0;
Iterator<Record> iter = context.readResourceTable("multijobs_res_table");
while (iter.hasNext()) {
Record r = iter.next();
v = (Long) r.get(0);
if (expect != v) {
throw new IOException("expect: " + expect + ", but: " + v);
}
count++;
}
if (count != 1) {
throw new IOException("res_table should have 1 record, but: " + count);
}
Record record = context.createOutputRecord();
v--;
record.set(0, v);
context.write(record);
context.getCounter("multijobs", "value").setValue(v);
}
}
public static void main(String[] args) throws Exception { if (args.length != 1) {
System.err.println("Usage: TestMultiJobs <table>");
System.exit(1);
}
String tbl = args[0];
long iterCount = 2;
System.err.println("Start to run init job.");
JobConf initJob = new JobConf();
initJob.setLong("multijobs.value", iterCount);
initJob.setMapperClass(InitMapper.class);
InputUtils.addTable(TableInfo.builder().tableName("mr_empty").build(), initJob);
OutputUtils.addTable(TableInfo.builder().tableName(tbl).build(), initJob);
initJob.setMapOutputKeySchema(SchemaUtils.fromString("key:string"));
initJob.setMapOutputValueSchema(SchemaUtils.fromString("value:string"));
initJob.setNumReduceTasks(0); JobClient.runJob(initJob);
while (true) {
System.err.println("Start to run iter job, count: " + iterCount);
JobConf decJob = new JobConf();
decJob.setLong("multijobs.expect.value", iterCount);
```

```
decJob.setMapperClass(DecreaseMapper.class);
InputUtils.addTable(TableInfo.builder().tableName("mr_empty").build(), decJob);
OutputUtils.addTable(TableInfo.builder().tableName(tbl).build(), decJob);
decJob.setNumReduceTasks(0);
RunningJob rJob = JobClient.runJob(decJob); iterCount--;
if (rJob.getCounters().findCounter("multijobs", "value").getValue() == 0) { break;
}
}
if (iterCount != 0) {
throw new IOException("Job failed.");
}
}
}
```

26.7.5.5 二次排序示例

示例如下：

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.SchemaUtils;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.data.TableInfo;
/**
* This is an example ODPS Map/Reduce application. It reads the input > table that
* must contain two integers per record. The output is sorted by the > first and
* second number and grouped on the first number.
*
* To run: jar -resources odps-mapred-example-0.12.0.jar
* com.aliyun.odps.mapred.open.example.SecondarySort mr_sort_in > mr_secondarysort_out;
*
**/
public class SecondarySort {
/**
* Read two integers from each line and generate a key, value pair as > ((left,
* right), right).
**/
public static class MapClass extends MapperBase { private Record key;
private Record value;
@Override
public void setup(TaskContext context) throws IOException { key = context.
createMapOutputKeyRecord();
value = context.createMapOutputValueRecord();
}
@Override
public void map(long recordNum, Record record, TaskContext context) throws IOException {
long left = 0;
long right = 0;
if (record.getColumnCount() > 0) { left = (Long) record.get(0);
if (record.getColumnCount() > 1) { right = (Long) record.get(1);
}
key.set(new Object[] { (Long) left, (Long) right });
value.set(new Object[] { (Long) right });
```

```
context.write(key, value);
}
}
}
/**
* A reducer class that just emits the sum of the input values.
**/
public static class ReduceClass extends ReducerBase {
private Record result = null;
@Override
public void setup(TaskContext context) throws IOException { result = context.createOutputRecord();
}
@Override
public void reduce(Record key, Iterator<Record> values, TaskContext context) throws IOException {
result.set(0, key.get(0));
while (values.hasNext()) {
Record value = values.next();
result.set(1, value.get(0));
context.write(result);
}
}
}
public static void main(String[] args) throws Exception {
if (args.length != 2) {
System.err.println("Usage: secondarysrot <in> <out>");
System.exit(2);
}
JobConf job = new JobConf();
job.setMapperClass(MapClass.class);
job.setReducerClass(ReduceClass.class);
//将多列设置为Key
// compare first and second parts of the pair
job.setOutputKeySortColumns(new String[] { "i1", "i2" });
// partition based on the first part of the pair
job.setPartitionColumns(new String[] { "i1" });
// grouping comparator based on the first part of the pair
job.setOutputGroupingColumns(new String[] { "i1" });
// the map output is LongPair, Long
job.setMapOutputKeySchema(SchemaUtils.fromString("i1:bigint,i2:bigint"));
job.setMapOutputValueSchema(SchemaUtils.fromString("i2x:bigint"));
InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
JobClient.runJob(job); System.exit(0);
}
}
```

26.7.5.6 使用资源示例

示例如下：

```
package com.aliyun.odps.mapred.open.example;
import java.io.BufferedInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
```

```
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
* Upload
*
* Import data from text file into table
*
* To run: jar -resources > odps-mapred-example-0.12.0.jar,mr_join_src1.txt
* com.aliyun.odps.mapred.open.example.Upload mr_join_src1.txt > mr_join_src1;
*
**/
public class Upload {
public static class UploadMapper extends MapperBase {
@Override
public void setup(TaskContext context) throws IOException { Record record = context.createOutputRecord();
StringBuilder importdata = new StringBuilder();
BufferedInputStream bufferedInput = null;
try {
byte[] buffer = new byte[1024]; int bytesRead = 0;
String filename = context.getJobConf().get("import.filename");
bufferedInput = context.readResourceFileAsStream(filename);
while ((bytesRead = bufferedInput.read(buffer)) != -1) {
String chunk = new String(buffer, 0, bytesRead);
importdata.append(chunk);
}
String lines[] = importdata.toString().split("\n"); for (int i = 0; i < lines.length; i++) {
String[] ss = lines[i].split(",");
record.set(0, Long.parseLong(ss[0].trim()));
record.set(1, ss[1].trim()); context.write(record);
}
} catch (FileNotFoundException ex) { throw new IOException(ex);
} catch (IOException ex) { throw new IOException(ex);
} finally {
}
}
@Override
public void map(long recordNum, Record record, TaskContext context) throws IOException {
}
}
public static void main(String[] args) throws Exception { if (args.length != 2) {
System.err.println("Usage: Upload <import_txt> <out_table>");
System.exit(2);
}
JobConf job = new JobConf();
job.setMapperClass(UploadMapper.class);
job.set("import.filename", args[0]);
job.setNumReduceTasks(0);
job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint"));
job.setMapOutputValueSchema(SchemaUtils.fromString("value:string"));
InputUtils.addTable(TableInfo.builder().tableName("mr_empty").build(), job);
OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
JobClient.runJob(job);
}
}
```

说明:

用户有多种手段设置JobConf：

- 通过SDK中JobConf的接口设置，本示例即是通过此方法。此方法的优先级最高。
- 在jar命令行中通过–conf参数指定新的JobConf文件。此种方式的优先级最低。–conf的使用方式请参考运行命令。

26.7.5.7 使用counter示例

代码示例中定义了三个Counter：map_outputs，reduce_outputs和global_counts。用户可以在Map/Reduce的setup，map/reduce及cleanup接口中获取任意自定义Counter，并进行操作。

示例如下：

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException; import java.util.Iterator;
import com.aliyun.odps.counter.Counter;
import com.aliyun.odps.counter.Counters;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.RunningJob;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.SchemaUtils;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.data.TableInfo;
/**
* User Defined Counters
*
* To run: jar -resources odps-mapred-example-0.12.0.jar
* com.aliyun.odps.mapred.open.example.UserDefinedCounters mr_src > mr_testcounters_out;
*
**/
public class UserDefinedCounters {
enum MyCounter {
TOTAL_TASKS, MAP_TASKS, REDUCE_TASKS
}
public static class TokenizerMapper extends MapperBase {
private Record word;
private Record one;
@Override
public void setup(TaskContext context) throws IOException { super.setup(context);
Counter map_tasks = context.getCounter(MyCounter.MAP_TASKS);
Counter total_tasks = context.getCounter(MyCounter.TOTAL_TASKS);
map_tasks.increment(1);
total_tasks.increment(1);
word = context.createMapOutputKeyRecord();
one = context.createMapOutputValueRecord();
one.set(new Object[] { 1L });
}
@Override
public void map(long recordNum, Record record, TaskContext context) throws IOException {
for (int i = 0;
i < record.getColumnCount();
i++) {
```

```
word.set(new Object[] { record.get(i).toString() });
context.write(word, one);
}
}
}
public static class SumReducer extends ReducerBase {
private Record result = null;
@Override
public void setup(TaskContext context) throws IOException { result = context.createOutputRecord();
Counter reduce_tasks = context.getCounter(MyCounter.REDUCE_TASKS);
Counter total_tasks = context.getCounter(MyCounter.TOTAL_TASKS);
reduce_tasks.increment(1);
total_tasks.increment(1);
}
@Override
public void reduce(Record key, Iterator<Record> values, TaskContext context) throws IOException {
long count = 0;
while (values.hasNext()) {
Record val = values.next();
count += (Long) val.get(0);
}
result.set(0, key.get(0));
result.set(1, count);
context.write(result);
}
}
public static void main(String[] args) throws Exception { if (args.length != 2) {
System.err.println("Usage: TestUserDefinedCounters <in_table> <out_table>");
System.exit(2);
}
JobConf job = new JobConf();
job.setMapperClass(TokenizerMapper.class);
job.setReducerClass(SumReducer.class);
job.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
job.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
RunningJob rJob = JobClient.runJob(job);
Counters counters = rJob.getCounters();
long m = counters.findCounter(MyCounter.MAP_TASKS).getValue();
long r = counters.findCounter(MyCounter.REDUCE_TASKS).getValue();
long total = counters.findCounter(MyCounter.TOTAL_TASKS).getValue();
System.exit(0);
}
}
```

26.7.5.8 grep示例

示例如下：

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.Mapper;
```

```
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.RunningJob;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
* Extracts matching regexs from input files and counts them.
*
* To run: jar -resources odps-mapred-example-0.12.0.jar
* com.aliyun.odps.mapred.open.example.Grep mr_src mr_grep_tmp > mr_grep_out val;
**/
public class Grep {
/**
* RegexMapper
**/
public class RegexMapper extends MapperBase { private Pattern pattern;
private int group;
private Record word;
private Record one;
@Override
public void setup(TaskContext context) throws IOException {
JobConf job = (JobConf) context.getJobConf();
pattern = Pattern.compile(job.get("mapred.mapper.regex"));
group = job.getInt("mapred.mapper.regex.group", 0);
word = context.createMapOutputKeyRecord();
one = context.createMapOutputValueRecord();
one.set(new Object[] { 1L });
}
@Override
public void map(long recordNum, Record record, TaskContext context) throws IOException {
for (int i = 0; i < record.getColumnCount(); ++i) {
String text = record.get(i).toString();
Matcher matcher = pattern.matcher(text);
while (matcher.find()) {
word.set(new Object[] { matcher.group(group) });
context.write(word, one);
}
}
}
}
/**
* LongSumReducer
**/
public class LongSumReducer extends ReducerBase {
private Record result = null;
@Override
public void setup(TaskContext context) throws IOException { result =
 context.createOutputRecord();
}
@Override
public void reduce(Record key, Iterator<Record> values, TaskContext context) throws
 IOException {
long count = 0;
while (values.hasNext()) {
Record val = values.next();
count += (Long) val.get(0);
}
result.set(0, key.get(0));
result.set(1, count);
```

```
context.write(result);
}
}
/**
* A {@link Mapper} that swaps keys and values.
**/
public class InverseMapper extends MapperBase {
private Record word;
private Record count;
@Override
public void setup(TaskContext context) throws IOException {
word = context.createMapOutputValueRecord();
count = context.createMapOutputKeyRecord();
}
/**
* The inverse function. Input keys and values are swapped.
**/
@Override
public void map(long recordNum, Record record, TaskContext context) throws IOException {
word.set(new Object[] { record.get(0).toString() });
count.set(new Object[] { (Long) record.get(1) });
context.write(count, word);
}
}
/**
* IdentityReducer
**/
public class IdentityReducer extends ReducerBase {
private Record result = null;
@Override
public void setup(TaskContext context) throws IOException {
result = context.createOutputRecord();
}
/** Writes all keys and values directly to output. **/
@Override
public void reduce(Record key, Iterator<Record> values, TaskContext context) throws
 IOException {
result.set(0, key.get(0));
while (values.hasNext()) {
Record val = values.next();
result.set(1, val.get(0));
context.write(result);
}
}
}
public static void main(String[] args) throws Exception {
if (args.length < 4) {
System.err.println("Grep <inDir> <tmpDir> <outDir> <regex> [<group>]"); System.exit(2);
}
JobConf grepJob = new JobConf();
grepJob.setMapperClass(RegexMapper.class);
grepJob.setReducerClass(LongSumReducer.class);
grepJob.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
grepJob.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), grepJob);
OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), grepJob);
grepJob.set("mapred.mapper.regex", args[3]);
if (args.length == 5) {
grepJob.set("mapred.mapper.regex.group", args[4]);
}
@SuppressWarnings("unused")
RunningJob rjGrep = JobClient.runJob(grepJob);
```

```
JobConf sortJob = new JobConf();
sortJob.setMapperClass(InverseMapper.class);
sortJob.setReducerClass(IdentityReducer.class);
sortJob.setMapOutputKeySchema(SchemaUtils.fromString("count:bigint"));
sortJob.setMapOutputValueSchema(SchemaUtils.fromString("word:string"));
InputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), sortJob);
OutputUtils.addTable(TableInfo.builder().tableName(args[2]).build(), sortJob);
sortJob.setNumReduceTasks(1); // write a single file
sortJob.setOutputKeySortColumns(new String[] { "count" });
 // sort by
// decreasing
// freq
@SuppressWarnings("unused")
RunningJob rjSort = JobClient.runJob(sortJob);
}
}
```

26.7.5.9 join示例

MaxCompute MapReduce框架自身并不支持Join逻辑。但用户可以在自己的Map/Reduce函数中实现数据的Join，这需要用户做一些额外的工作。

假设需要Join两张表mr_join_src1(key bigint, value string)及mr_join_src2(key bigint, value string)，输出表是mr_join_out(key bigint, value1 string, value2 string)，其中value1是mr_join_src1的value值，value2是mr_join_src2的value值。

示例如下：

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import com.aliyun.odps.counter.Counter;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
* Join
*
* To run: jar -resources odps-mapred-example-0.12.0.jar
* com.aliyun.odps.mapred.open.example.Join mr_join_src11 > mr_join_src22 mr_join_out;
*
**/
public class Join {
public static class JoinMapper extends MapperBase {
private Record mapkey;
private Record mapvalue;
@Override
public void setup(TaskContext context) throws IOException {
```

```
mapkey = context.createMapOutputKeyRecord();
mapvalue = context.createMapOutputValueRecord();
}
@Override
public void map(long key, Record record, TaskContext context) throws IOException {
/*              根据value的值判断这个record来自那张表，此处属于用户代码逻辑。如果无法通过表中value字段的值来判断Record属于哪张表，可以考虑向输入表中添加一个字段。通过value值产生的tag，将在Reduce中执行连接操作时用到。 */
long tag = 1;
String val = record.get(1).toString();
if (val.startsWith("valb_")) {
tag = 2;
}
mapkey.set(0, Long.parseLong(record.get(0).toString()));
mapkey.set(1, tag);
mapvalue.set(0, tag);
for (int i = 1; i < record.getColumnCount(); i++) {
mapvalue.set(i, record.get(i));
}
context.write(mapkey, mapvalue);
}
}
public static class JoinReducer extends ReducerBase {
private Record result = null;
@Override
public void setup(TaskContext context) throws IOException {
result = context.createOutputRecord();
}
@Override
public void reduce(Record key, Iterator<Record> values, TaskContext context) throws IOException {
long k = (Long) key.get(0);
List<Object[]> list1 = new ArrayList<Object[]>();
Counter cnt = context.getCounter("MyCounters", "reduce_outputs");
cnt.increment(1);
while (values.hasNext()) {
Record value = values.next();
long tag = (Long) value.get(0);
if (tag == 1) {
//如果数据来自第一张表，将数据缓存到list中。
//由于数据是按照Key, tag来进行排序的，因此tag==1的Value会被排到前面。
//在实际计算中，建议用户慎重使用这种方式，如果某个key的value过多，
//将会导致Reduce所需内存增加，超过JobConf::setMemoryForReduceTask的设置时，
//Reduce有可能会因为超内存而被系统结束掉。
list1.add(value.toArray().clone());
} else {
//如果数据来自第二张表，按照Key, tag来进行排序，
//此时第一张表的所有value都已经被存放在list1中。
//对于第一张表的所有value
for (Object r1: list1) { int index = 0;
//首先设置Key
result.set(index++, k);
Object[] s_arr = (Object[])r1;
result.set(index++, s_arr[1].toString());
result.set(index++, value.get(1).toString());
context.write(result);
}
}
}
}
}
```

```
public static void main(String[] args) throws Exception {
if (args.length != 3) {
System.err.println("Usage: Join <input table1> <input table2> <out>");
System.exit(2);
}
JobConf job = new JobConf();
job.setMapperClass(JoinMapper.class);
job.setReducerClass(JoinReducer.class);
job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint,tag:bigint"));
job.setMapOutputValueSchema(SchemaUtils.fromString("tagx:bigint,value:string"));
job.setPartitionColumns(new String[] { "key" });
//使用key，tag排序。只有key有序后，在Reduce端才能做join
//tag用来区分当前Record来自那张表。
job.setOutputKeySortColumns(new String[] { "key", "tag" });
job.setOutputGroupingColumns(new String[] { "key" });
//由于Reduce端使用list缓存数据，建议加大Reduce Worker内存
job.setMemoryForReduceTask(4096);
job.setInt("table.counter", 0);
InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
InputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
OutputUtils.addTable(TableInfo.builder().tableName(args[2]).build(), job);
JobClient.runJob(job);
}
}
```

26.7.5.10 sleep示例

示例如下：

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
importjava.util.LinkedHashMap;
import java.util.Map;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import com.aliyun.odps.OdpsException;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
* Dummy class for testing MR framefork. Sleeps for a defined period of > time in
* mapper and reducer. Generates fake input for map / reduce jobs. Note > that
* generated number of input pairs is in the order of
* <code>numMappers
* mapSleepTime / 100</code>, so the > job uses some disk
* space.
*  To run: jar -resources odps-mapred-example-0.12.0.jar >
 com.aliyun.odps.mapred.open.example.SleepJob
* -m 1 -r 1 -mt 1 -rt 1;
*
**/
public class SleepJob {
```

```
private static Log LOG = LogFactory.getLog(SleepJob.class);
public static class SleepMapper extends MapperBase {
private LinkedHashMap<Integer, Integer> inputs = new LinkedHashMap<Integer, Integer>();
private long mapSleepDuration = 100;
private int mapSleepCount = 1;
private int count = 0;
private Record key;
@Override
public void setup(TaskContext context) throws IOException {
LOG.info("map setup called");
JobConf conf = (JobConf) context.getJobConf();
mapSleepCount = conf.getInt("sleep.job.map.sleep.count", 1);
if (mapSleepCount < 0)
throw new IOException("Invalid map count: " + mapSleepCount);
mapSleepDuration = conf.getLong("sleep.job.map.sleep.time", 100)
/ mapSleepCount;
LOG.info("mapSleepCount = " + mapSleepCount + ", mapSleepDuration = " +
 mapSleepDuration);
final int redcount = conf.getInt("sleep.job.reduce.sleep.count", 1);
if (redcount < 0)
throw new IOException("Invalid reduce count: " + redcount);
final int emitPerMapTask = (redcount * conf.getNumReduceTasks());
int records = 0;
int emitCount = 0;
while (records++ < mapSleepCount) {
int key = emitCount;
int emit = emitPerMapTask / mapSleepCount;
if ((emitPerMapTask) % mapSleepCount > records) {
++emit;
}
emitCount += emit;
int value = emit;
inputs.put(key, value);
}
key = context.createMapOutputKeyRecord();
}
@Override
public void cleanup(TaskContext context) throws IOException {
// it is expected that every map processes mapSleepCount number of
// records.
LOG.info("map run called");
for (Map.Entry<Integer, Integer> entry : inputs.entrySet()) {
LOG.info("Sleeping... (" + (mapSleepDuration * (mapSleepCount - count)) + ") ms left");
try {
Thread.sleep(mapSleepDuration);
} catch (InterruptedException e) { throw new IOException(e);
}
++count;
// output reduceSleepCount * numReduce number of random values, so that
// each reducer will get reduceSleepCount number of keys.
int k = entry.getKey();
int v = entry.getValue();
for (int i = 0; i < v; ++i) {
key.set(new Object[] { (Long) ((long) (k + i)) });
context.write(key, key);
}
}
}
}
public static class SleepReducer extends ReducerBase {
private long reduceSleepDuration = 100;
private int reduceSleepCount = 1;
```

```
private int count = 0;
@Override
public void setup(TaskContext context) throws IOException {
LOG.info("reduce setup called");
JobConf conf = (JobConf) context.getJobConf();
reduceSleepCount = conf.getInt("sleep.job.reduce.sleep.count",
reduceSleepCount);
reduceSleepDuration = conf.getLong("sleep.job.reduce.sleep.time", 100)
/ reduceSleepCount;
LOG.info("reduceSleepCount = " + reduceSleepCount
+ ", reduceSleepDuration = " + reduceSleepDuration);
}
@Override
public void reduce(Record key, Iterator<Record> values, TaskContext context) throws
 IOException {
LOG.info("reduce called");
LOG.info("Sleeping... ("
+ (reduceSleepDuration * (reduceSleepCount - count)) + ") ms left"); try {
Thread.sleep(reduceSleepDuration);
} catch (InterruptedException e) {
throw new IOException(e);
}
count++;
}
}
public static int run(int numMapper, int numReducer, long mapSleepTime, int mapSleepCount,
 long reduceSleepTime, int reduceSleepCount)
throws OdpsException {
JobConf job = setupJobConf(numMapper, numReducer, mapSleepTime, mapSleepCount,
 reduceSleepTime, reduceSleepCount);
JobClient.runJob(job);
return 0;
}
public static JobConf setupJobConf(int numMapper, int numReducer, long mapSleepTime, int
 mapSleepCount, long reduceSleepTime, int reduceSleepCount) {
JobConf job = new JobConf();
InputUtils.addTable(TableInfo.builder().tableName("mr_empty").build(), job);
OutputUtils.addTable(TableInfo.builder().tableName("mr_sleep_out").build(), job);
job.setNumReduceTasks(numReducer);
job.setMapperClass(SleepMapper.class);
job.setReducerClass(SleepReducer.class);
job.setMapOutputKeySchema(SchemaUtils.fromString("int1:bigint"));
job.setMapOutputValueSchema(SchemaUtils.fromString("int2:bigint"));
job.setPartitionColumns(new String[] { "int1" });
job.setLong("sleep.job.map.sleep.time", mapSleepTime);
job.setLong("sleep.job.reduce.sleep.time", reduceSleepTime);
job.setInt("sleep.job.map.sleep.count", mapSleepCount);
job.setInt("sleep.job.reduce.sleep.count", reduceSleepCount);
return job;
}
private static void printUsage() {
System.err.println("SleepJob [-m numMapper] [-r numReducer]"
+ " [-mt mapSleepTime (msec)] [-rt reduceSleepTime (msec)]"
+ " [-recordt recordSleepTime (msec)]");
}
public static void main(String[] args) throws Exception {
if (args.length < 1) {
printUsage();
return;
}
int numMapper = 1, numReducer = 1;
long mapSleepTime = 100, reduceSleepTime = 100, recSleepTime = 100;
```

```
int mapSleepCount = 1, reduceSleepCount = 1;
for (int i = 0; i < args.length; i++) { if (args[i].equals("-m")) {
numMapper = Integer.parseInt(args[++i]);
} else if (args[i].equals("-r")) {
numReducer = Integer.parseInt(args[++i]);
} else if (args[i].equals("-mt")) {
mapSleepTime = Long.parseLong(args[++i]);
} else if (args[i].equals("-rt")) {
reduceSleepTime = Long.parseLong(args[++i]);
} else if (args[i].equals("-recordt")) { recSleepTime = Long.parseLong(args[++i]);
}
}
// sleep for *SleepTime duration in Task by recSleepTime per record
mapSleepCount = (int) Math.ceil(mapSleepTime / ((double) recSleepTime));
reduceSleepCount = (int) Math.ceil(reduceSleepTime
/ ((double) recSleepTime));
run(numMapper, numReducer, mapSleepTime, mapSleepCount, reduceSleepTime,
 reduceSleepCount);
}
}
```

26.7.5.11 unique示例

示例如下：

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
* Unique Remove duplicate words
*
* To run: jar -resources odps-mapred-example-0.12.0.jar
* com.aliyun.odps.open.mapred.example.Unique mr_sort_in > mr_unique_out
* key|value|all;
*
**/
public class Unique {
public static class OutputSchemaMapper extends MapperBase {
private Record key;
private Record value;
@Override
public void setup(TaskContext context) throws IOException {
key = context.createMapOutputKeyRecord();
value = context.createMapOutputValueRecord();
}
@Override
public void map(long recordNum, Record record, TaskContext context) throws IOException {
long left = 0;
long right = 0;
if (record.getColumnCount() > 0) { left = (Long) record.get(0);
if (record.getColumnCount() > 1) { right = (Long) record.get(1);
```

```
}
key.set(new Object[] { (Long) left, (Long) right });
value.set(new Object[] { (Long) left, (Long) right });
context.write(key, value);
}
}
}
public static class OutputSchemaReducer extends ReducerBase {
private Record result = null;
@Override
public void setup(TaskContext context) throws IOException {
result = context.createOutputRecord();
}
@Override
public void reduce(Record key, Iterator<Record> values, TaskContext context) throws IOException {
result.set(0, key.get(0));
while (values.hasNext()) {
Record value = values.next();
result.set(1, value.get(1));
}
context.write(result);
}
}
public static void main(String[] args) throws Exception {
if (args.length > 3 || args.length < 2) {
System.err.println("Usage: unique <in> <out> [key|value|all]");
System.exit(2);
}
String ops = "all";
if (args.length == 3) { ops = args[2];
}
// Key Unique
if (ops.equals("key")) {
JobConf job = new JobConf();
job.setMapperClass(OutputSchemaMapper.class);
job.setReducerClass(OutputSchemaReducer.class);
job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint,value:bigint"));
job.setMapOutputValueSchema(SchemaUtils.fromString("key:bigint,value:bigint"));
job.setPartitionColumns(new String[] { "key" });
job.setOutputKeySortColumns(new String[] { "key", "value" });
job.setOutputGroupingColumns(new String[] { "key" });
job.set("tablename2", args[1]); job.setNumReduceTasks(1);
job.setInt("table.counter", 0);
InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
JobClient.runJob(job);
}
// Key&Value Unique
if (ops.equals("all")) {
JobConf job = new JobConf();
job.setMapperClass(OutputSchemaMapper.class);
job.setReducerClass(OutputSchemaReducer.class);
job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint,value:bigint"));
job.setMapOutputValueSchema(SchemaUtils.fromString("key:bigint,value:bigint"));
job.setPartitionColumns(new String[] { "key" });
job.setOutputKeySortColumns(new String[] { "key", "value" });
job.setOutputGroupingColumns(new String[] { "key", "value" });
job.set("tablename2", args[1]); job.setNumReduceTasks(1);
job.setInt("table.counter", 0);
InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
```

```
JobClient.runJob(job);
}
// Value Unique
if (ops.equals("value")) { JobConf job = new JobConf();
job.setMapperClass(OutputSchemaMapper.class);
job.setReducerClass(OutputSchemaReducer.class);
job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint,value:bigint"));
job.setMapOutputValueSchema(SchemaUtils.fromString("key:bigint,value:bigint"));
job.setPartitionColumns(new String[] { "value" });
job.setOutputKeySortColumns(new String[] { "value" });
job.setOutputGroupingColumns(new String[] { "value" });
job.set("tablename2", args[1]); job.setNumReduceTasks(1);
job.setInt("table.counter", 0);
InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
JobClient.runJob(job);
}
}
}
```

26.7.5.12 sort示例

示例如下：

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException; import java.util.Date;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
 import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.example.lib.IdentityReducer;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
* This is the trivial map/reduce program that does absolutely nothing > other
* than use the framework to fragment and sort the input values.
* <p>
*          To          run:          jar          -resources          odps-mapred-example-0.12.0.jar          >
 com.aliyun.odps.mapred.open.example.Sort
* mr_sort_in mr_sort_out;
*
**/
public class Sort {
static int printUsage() {
System.out.println("sort <input> <output>"); return -1;
}
/**
* Implements the identity function, mapping record's first two columns > to
* outputs.
**/
public static class IdentityMapper extends MapperBase {
private Record key;
private Record value;
@Override
public void setup(TaskContext context) throws IOException {
key = context.createMapOutputKeyRecord();
value = context.createMapOutputValueRecord();
}
```

```
@Override
public void map(long recordNum, Record record, TaskContext context) throws IOException {
key.set(new Object[] { (Long) record.get(0) });
value.set(new Object[] { (Long) record.get(1) });
context.write(key, value);
}
}
/**
* The main driver for sort program. Invoke this method to submit the
* map/reduce job.
*
* @throws IOException
* When there is communication problems with the job tracker.
**/
public static void main(String[] args) throws Exception {
JobConf jobConf = new JobConf();
jobConf.setMapperClass(IdentityMapper.class);
jobConf.setReducerClass(IdentityReducer.class);
jobConf.setNumReduceTasks(1);
jobConf.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint"));
jobConf.setMapOutputValueSchema(SchemaUtils.fromString("value:bigint"));
InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), jobConf);
OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), jobConf);
Date startTime = new Date(); System.out.println("Job started: " + startTime);
JobClient.runJob(jobConf);
Date end_time = new Date(); System.out.println("Job ended: " + end_time);
 System.out.println("The job took " + (end_time.getTime() - startTime.getTime()) / 1000 + "
 seconds.");
}
}
```

26.7.5.13 分区表输入示例

以Partition作为输入输出为例。

示例如下：

```
public static void main(String[] args) throws Exception {
JobConf job = new JobConf();
...
LinkedHashMap<String, String> input = new LinkedHashMap<String, String>();
input.put("pt", "123456");
InputUtils.addTable(TableInfo.builder().tableName("input_table").partSpec(input).build(), job);
LinkedHashMap<String, String> output = new LinkedHashMap<String, String>(); output.put("ds",
 "654321");
OutputUtils.addTable(TableInfo.builder().tableName("output_table").partSpec(output).build(),
 job);
JobClient.runJob(job);
}
```

示例2：

```
package com.aliyun.odps.mapred.open.example;
...
public static void main(String[] args) throws Exception {
if (args.length != 2) {
System.err.println("Usage: WordCount <in_table> <out_table>");
System.exit(2);
}
```

```
JobConf job = new JobConf();
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(SumCombiner.class);
job.setReducerClass(SumReducer.class);
job.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
job.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
odps.setEndpoint("odps_endpoint_url");
odps.setDefaultProject("my_project");
Table table = odps.tables().get(tblname);
TableInfoBuilder builder = TableInfo.builder().tableName(tblname);
for (Partition p : table.getPartitions()) { if (applicable(p)) {
LinkedHashMap<String, String> partSpec = new LinkedHashMap<String, String>();
for (String key : p.getPartitionSpec().keys()) {
partSpec.put(key, p.getPartitionSpec().get(key));
}
InputUtils.addTable(builder.partSpec(partSpec).build(), conf);
}
}
OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
JobClient.runJob(job);
}
```



说明: 这是一段使用MaxCompute SDK和MapReduce SDK组合实现MapReduce任务读取范围Partitoin的示例。此段代码不能够编译执行，仅给出了main函数的示例。示例中applicable函数是用户逻辑，用来决定该Partition是否符合作为该MapReduce作业的输入。

26.7.5.14 pipeline示例

示例如下：

```
package com.aliyun.odps.mapred.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.Column;
import com.aliyun.odps.OdpsException;
import com.aliyun.odps.OdpsType;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.Job;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.pipeline.Pipeline;
public class Histogram {
public static class TokenizerMapper extends MapperBase {
Record word;
Record one;
@Override
public void setup(TaskContext context) throws IOException {
word = context.createMapOutputKeyRecord();
one = context.createMapOutputValueRecord();
one.setBigint(0, 1L);
}
@Override
public void map(long recordNum, Record record, TaskContext context)
```

```
throws IOException {
for (int i = 0; i < record.getColumnCount(); i++) {
String[] words = record.get(i).toString().split("\\s+");
for (String w : words) {
word.setString(0, w);
context.write(word, one);
}
}
}
}
public static class SumReducer extends ReducerBase { private Record num;
private Record result;
@Override
public void setup(TaskContext context) throws IOException {
num = context.createOutputKeyRecord();
result = context.createOutputValueRecord();
}
@Override
public void reduce(Record key, Iterator<Record> values, TaskContext context) throws
 IOException {
long count = 0;
while (values.hasNext()) {
Record val = values.next();
count += (Long) val.get(0);
}
result.set(0, key.get(0));
num.set(0, count);
context.write(num, result);
}
}
public static class IdentityReducer extends ReducerBase {
@Override
public void reduce(Record key, Iterator<Record> values, TaskContext context) throws
 IOException {
while (values.hasNext()) {
context.write(values.next());
}
}
}
public static void main(String[] args) throws OdpsException {
if (args.length != 2) {
System.err.println("Usage: orderedwordcount <in_table> <out_table>");
System.exit(2);
}
Job job = new Job();
/**
*      构造Pipeline的过程中，如果不指定Mapper的OutputKeySortColumns，PartitionColumns，
OutputGroupingColumns，
* 框架会默认使用其OutputKey作为此三者的默认配置
**/
Pipeline pipeline = Pipeline.builder()
.addMapper(TokenizerMapper.class)
.setOutputKeySchema(
new Column[] { new Column("word", OdpsType.STRING) })
.setOutputValueSchema(
new Column[] { new Column("count", OdpsType.BIGINT) })
.setOutputKeySortColumns(new String[] { "word" })
.setPartitionColumns(new String[] { "word" })
.setOutputGroupingColumns(new String[] { "word" })
.addReducer(SumReducer.class)
.setOutputKeySchema(
new Column[] { new Column("count", OdpsType.BIGINT) })
```

```
.setOutputValueSchema(
new Column[] { new Column("word", OdpsType.STRING)})
.addReducer(IdentityReducer.class).createPipeline();
job.setPipeline(pipeline);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
job.submit(); job.waitForCompletion();
System.exit(job.isSuccessful() == true ? 0 : 1);
}
}
```

26.8 Graph使用指南

26.8.1 Graph概要

MaxCompute Graph是一套面向迭代的图计算处理框架。图计算作业使用图进行建模，图由点（Vertex）和边（Edge）组成，点和边包含权值（Value）。MaxCompute Graph支持下述图编辑操作：

- 修改点或边的权值。
- 增加/删除点。
- 增加/删除边。

> **说明:** 编辑点和边时，点与边的关系需要用户维护。

通过迭代对图进行编辑、演化，最终求解出结果，典型应用有*PageRank*，单源最短距离算法，*K*-均值聚类算法等。用户可以使用MaxCompute Graph提供的接口Java SDK编写图计算程序。

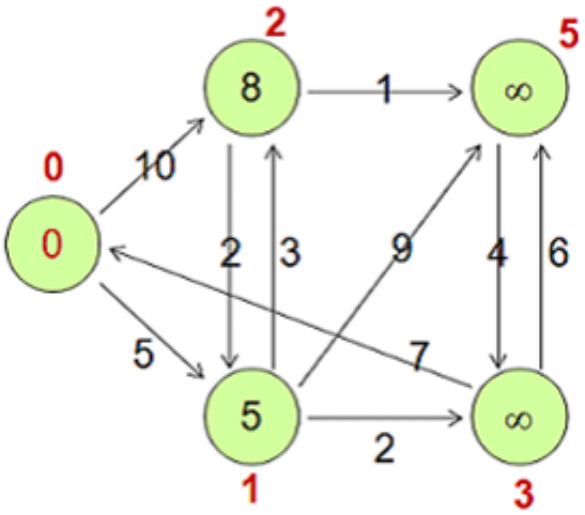
26.8.1.1 Graph数据结构

MaxCompute Graph能够处理的图必须是一个由点（Vertex）和边（Edge）组成的有向图。由于MaxCompute仅提供二维表的存储结构， 因此需要用户自行将图数据分解为二维表格式存储在MaxCompute中，在进行图计算分析时，使用自定义的GraphLoader将二维表数据转换为MaxCompute Graph引擎中的点和边。至于如何将图数据分解为二维表格式，用户可以根据自身的业务场景做决定。在后面的示例程序章节中，给出的示例分别使用了不同的表格式来表达图的数据结构，仅供用户参考。

点的结构可以简单表示为< ID, Value, Halted, Edges >，分别表示点标识符（ID）、权值（Value）、状态（Halted，表示是否要停止迭代）以及出边集合（Edges，以该点为起始点的所

有边列表）。边的结构可以简单表示为<DestVertexID, Value >，分别表示目标点（DestVertexID）和权值（Value）。Graph数据结构如图所示：

图 775: Graph数据结构



以上图为例，上图由下面的点组成：

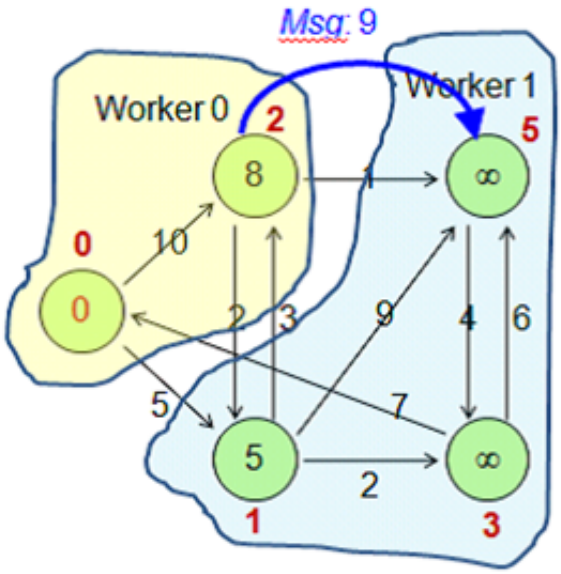
Vertex	<ID, Value, Halted, Edges>
v0	<0, 0, false, [<1, 5 >, <2, 10 >] >
v1	<1, 5, false, [<2, 3>, <3, 2>, <5, 9>]>
v2	<2, 8, false, [<1, 2>, <5, 1 >]>
v3	<3, Long.MAX_VALUE, false, [<0, 7>, <5, 6>]>
v5	<5, Long.MAX_VALUE, false, [<3, 4 >]>

26.8.1.2 Graph程序逻辑

26.8.1.2.1 加载图

- 图加载：框架调用用户自定义的GraphLoader，将输入表的记录解析为点或边。
- 分布式化：框架调用用户自定义的Partitioner，对点进行分片（默认分片逻辑：点ID哈希值，然后对Worker数取模），分配到相应的Worker。

图 776: 加载图



以上图为例，假设Worker数是2，那么v0， v2会被分配到Worker0，因为ID对2取模结果为0，而v1，v3，v5将被分配到Worker1，ID对2取模结果为1。

26.8.1.2.2 迭代计算

一次迭代为一个超步（SuperStep），遍历所有非结束状态（Halted值为false）的点或者收到消息的点（处于结束状态的点收到信息会被自动唤醒），并调用其compute（ComputeContext context， Iterable messages）方法。

在用户实现的compute（ComputeContext context，Iterable messages）方法中：

- 处理上一个超步发给当前点的消息（Messages）。
- 根据需要对图进行编辑：
  - 修改点/边的取值。
  - 发送消息给某些点。
  - 增加/删除点或边。
- 通过Aggregator汇总信息到全局信息。
- 设置当前点状态为结束或非结束状态。
- 迭代进行过程中，框架会将消息以异步的方式发送到对应Worker并在下一个超步进行处理，用户无需关心。

26.8.1.2.3 迭代终止

满足以下任意一个条件，迭代终止。

- 所有点处于结束状态（Halted值为true）且没有新消息产生。
- 达到最大迭代次数。
- 某个Aggregator的terminate方法返回true。

伪代码描述如下：

```
// 1. load
for each record in input_table { GraphLoader.load();
}
// 2. setup
WorkerComputer.setup();
for each aggr in aggregators { aggr.createStartupValue();
}
for each v in vertices { v.setup();
}
// 3. superstep
for (step = 0; step < max; step ++) { for each aggr in aggregators { aggr.createInitialValue();
}
for each v in vertices { v.compute();
}
}
// 4. cleanup
for each v in vertices { v.cleanup();
}
WorkerComputer.cleanup();
```

26.8.2 Graph功能介绍

26.8.2.1 运行作业

MaxCompute客户端提供一个jar命令用于运行MaxCompute Graph作业，其使用方式与MapReduce中的jar命令相同。

命令格式如下：

```
Usage: jar [<GENERIC_OPTIONS>] <MAIN_CLASS> [ARGS]
-conf <configuration_file> Specify an application configuration file
-classpath <local_file_list> classpaths used to run mainClass
-D <name>=<value> Property value pair, which will be used to run mainClass
-local Run job in local mode
-resources <resource_name_list> file/table resources used in graph, seperate by comma
```

参数说明：

- -conf <configuration file> ：指定JobConf配置文件。
- -classpath <local_file_list>：本地执行时的classpath，主要用于指定main函数所在的jar包的本地路径（包含相对路径和绝对路径）。
- -D <prop_name>=<prop_value> ：本地执行时，<mainClass>的java属性，可以定义多个。
- -local：以本地模式执行Graph作业，主要用于程序调试。

- -resources <resource_name_list>： MapReduce作业运行时使用的资源声明。一般情况下，resource_name_list中需要指定Graph作业所在的资源名称。

  注意: 如果用户在Graph作业中读取了其他MaxCompute资源，这些资源名称也需要被添加到resource_name_list中。资源之间使用逗号分隔，跨项目空间使用资源时，需要在前面加上：PROJECT_NAME/resources/，例如：-resources otherproject/resources/resfile。

  说明: 上述的可选参数均包含在**<GENERIC_OPTIONS>**中。

用户也可以直接运行Graph作业的main函数直接将作业提交到MaxCompute，而不是通过MaxCompute客户端提交作业。以PageRank算法为例。

示例如下：

```
public static void main(String[] args) throws IOException {
if (args.length < 2)
printUsage();
GraphJob job = new GraphJob();
job.setGraphLoaderClass(PageRankVertexReader.class);
job.setVertexClass(PageRankVertex.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
// 将作业中使用的资源添加到cache resource，对应于jar命令中 -resources 和 -libjars 中指定的资源
job.addCacheResource("mapreduce-examples.jar");
// 将使用的jar及其他文件添加到class cache resource，对应于jar命令中 -libjars 中指定的资源
job.addCacheResourceToClassPath("mapreduce-examples.jar");
// 设置console中，odps_config.ini对应的配置项，使用时替换为自己的配置
Account account = new AliyunAccount(accessId, accessKey);
Odps odps = new Odps(account);
odps.setDefaultProject(project);
odps.setEndpoint(endpoint);
SessionState.get().setOdps(odps);
SessionState.get().setLocalRun(false); // default max iteration is 30
job.setMaxIteration(30);
if (args.length >= 3)
job.setMaxIteration(Integer.parseInt(args[2]));
long startTime = System.currentTimeMillis(); job.run();
System.out.println("Job Finished in "
+ (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
}
```

26.8.2.2 输入输出

MaxCompute Graph作业的输入输出限制为表，不允许用户自定义输入输出格式。

作业输入定义如下：

```
GraphJob job = new GraphJob();
job.addInput(TableInfo.builder().tableName( “tblname” ).build()); //表作为输入
```

```
job.addInput(TableInfo.builder().tableName( “tblname” ).partSpec("pt1=a/pt2=b").build()); //分区作为输入
//只读取输入表的 col2 和 col0 列，在 GraphLoader 的 load 方法中，record.get(0) 得到的是col2列，顺序一致
job.addInput(TableInfo.builder().tableName( “tblname” ).partSpec("pt1=a/pt2=b").build(), new String[]{"col2", "col0 "});
```

说明:

- 支持多路输入。
- 暂时不支持分区过滤条件。更多应用限制请参考：应用限制。
- 关于作业输入定义，更多的信息请参考GraphJob的addInput相关方法说明。
- 框架读取输入表的记录传给用户自定义的GraphLoader载入图数据。

作业输出定义如下：

```
GraphJob job = new GraphJob();
//输出表为分区表时需要给到最末一级分区
job.addOutput(TableInfo.builder().tableName("table_name").partSpec("pt1=a/pt2=b").build());
// 下面的参数 true 表示覆盖tableinfo指定的分区，即INSERT OVERWRITE语义，false表示INSERT INTO语义
job.addOutput(TableInfo.builder().tableName("table_name").partSpec("pt1=a/
pt2=b").lable("output1").build(), true);
```

说明:

- 支持多路输出，通过label标识每路输出。
- Graph作业在运行时可以通过WorkerContext的write方法写出记录到输出表，多路输出需要指定标识。
- 关于作业输出定义，更多的信息请参见GraphJob的addOutput相关方法说明。

26.8.2.3 读取资源

26.8.2.3.1 Graph程序中添加资源

除了通过jar命令指定Graph读取的资源外，还可以通过GraphJob的如下两个方法指定：

```
void addCacheResources(String resourceNames)
void addCacheResourcesToClassPath(String resourceNames)
```

26.8.2.3.2 Graph程序中使用资源

在Graph程序中可以通过相应的上下文对象**WorkerContext**的下述方法读取资源：

```
public byte[] readCacheFile(String resourceName) throws IOException;
public Iterable<byte[]> readCacheArchive(String resourceName) throws IOException;
public Iterable<byte[]> readCacheArchive(String resourceName, String relativePath)throws IOException;
```

```
public Iterable<WritableRecord> readResourceTable(String resourceName);
public BufferedInputStream readCacheFileAsStream(String resourceName) throws IOException;
public Iterable<BufferedInputStream> readCacheArchiveAsStream(String resourceName) throws IOException;
public Iterable<BufferedInputStream> readCacheArchiveAsStream(String resourceName, String relativePath) throws IOException;
```

说明:

- 通常在WorkerComputer的setup方法里读取资源，然后保存在WorkerValue中，之后通过getWorkerValue的方法获取。
- 建议使用上面的流接口，边读边处理，降低内存耗费。
- 更多应用限制请参考：应用限制。

26.8.3 Graph SDK介绍

主要接口	说明
GraphJob	GraphJob继承自JobConf，用于定义、提交和管理一个MaxCompute Graph作业。
Vertex	Vertex是图的点的抽象，包含属性：id，value，halted，edges，通过GraphJob的setVertexClass接口提供Vertex实现。
Edge	Edge是图的边的抽象，包含属性：destVertexId， value，图数据结构采用邻接表，点的出边保存在点的edges中。
GraphLoader	GraphLoader用于载入图，通过GraphJob的setGraphLoaderClass接口提供GraphLoader实现。
VertexResolver	VertexResolver用于自定义图拓扑修改时的冲突处理逻辑，通过GraphJob的setLoadingVertexResolverClass和setComputingVertexResolverClass接口提供图加载和迭代计算过程中的图拓扑修改的冲突处理逻辑。
Partitioner	Partitioner用于对图进行划分使得计算可以分片进行，通过GraphJob的setPartitionerClass接口提供Partitioner实现，默认采用HashPartitioner，即对点ID求哈希值然后对Worker数目取模。
WorkerComputer	WorkerComputer允许在Worker开始和退出时执行用户自定义的逻辑，通过GraphJob的setWorkerComputerClass接口提供WorkerComputer实现。
Aggregator	通过Aggregator的setAggregatorClass(Class ...)接口定义一个或多个Aggregator。
Combiner	通过Combiner的setCombinerClass接口设置Combiner。
Counters	计数器，在作业运行逻辑中，可以通过WorkerContext接口取得计数器并进行计数，框架会自动进行汇总。

主要接口	说明
WorkerContext	上下文对象，封装了框架的提供的功能，如修改图拓扑结构，发送消息，写结果，读取资源等等。

26.8.4 开发和调试

MaxCompute没有为用户提供Graph开发插件，但用户仍然可以基于Eclipse开发MaxCompute Graph程序，建议的开发流程如下：

1. 编写Graph代码，使用本地调试进行基本的测试。
2. 进行集群调试，验证结果。

26.8.4.1 开发示例

背景信息

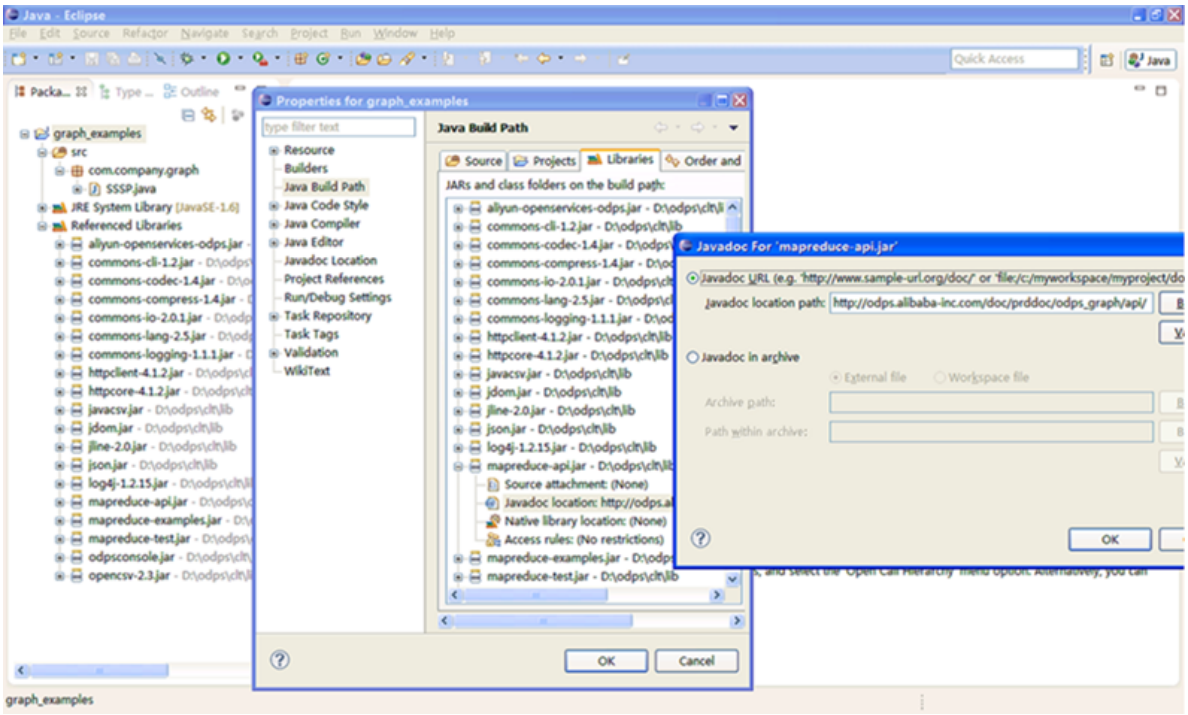
本章节以SSSP算法为例介绍如何使用Eclipse开发和调试Graph程序。具体的开发步骤如下。

操作步骤

1. 创建Java工程。

   例如：graph_examples；将MaxCompute客户端lib目录下的jar包添加到Eclipse工程的Build Path里。一个配置好的Eclipse工程如下图所示。

   图 777: 创建 Java 工程



2. 开发MaxCompute Graph程序。

   实际开发过程中，一般会先拷贝一个示例（例如SSSP），然后再做修改。在本示例中，仅修改了package的路径为：*package com.aliyun.odps.graph.example*。

3. 编译打包，在Eclipse环境中，右键单击源代码目录（上图中的src目录），选择**Export** > **Java** > **JAR file**生成jar包，并选择目标jar包的保存路径，例如：*D:\odps\clt\odps-graph-example-sssp.jar*。

4. 使用MaxCompute客户端运行SSSP，相关操作请参考：编写并运行*Graph*。

   说明: 相关的开发步骤请参考Graph开发插件介绍。

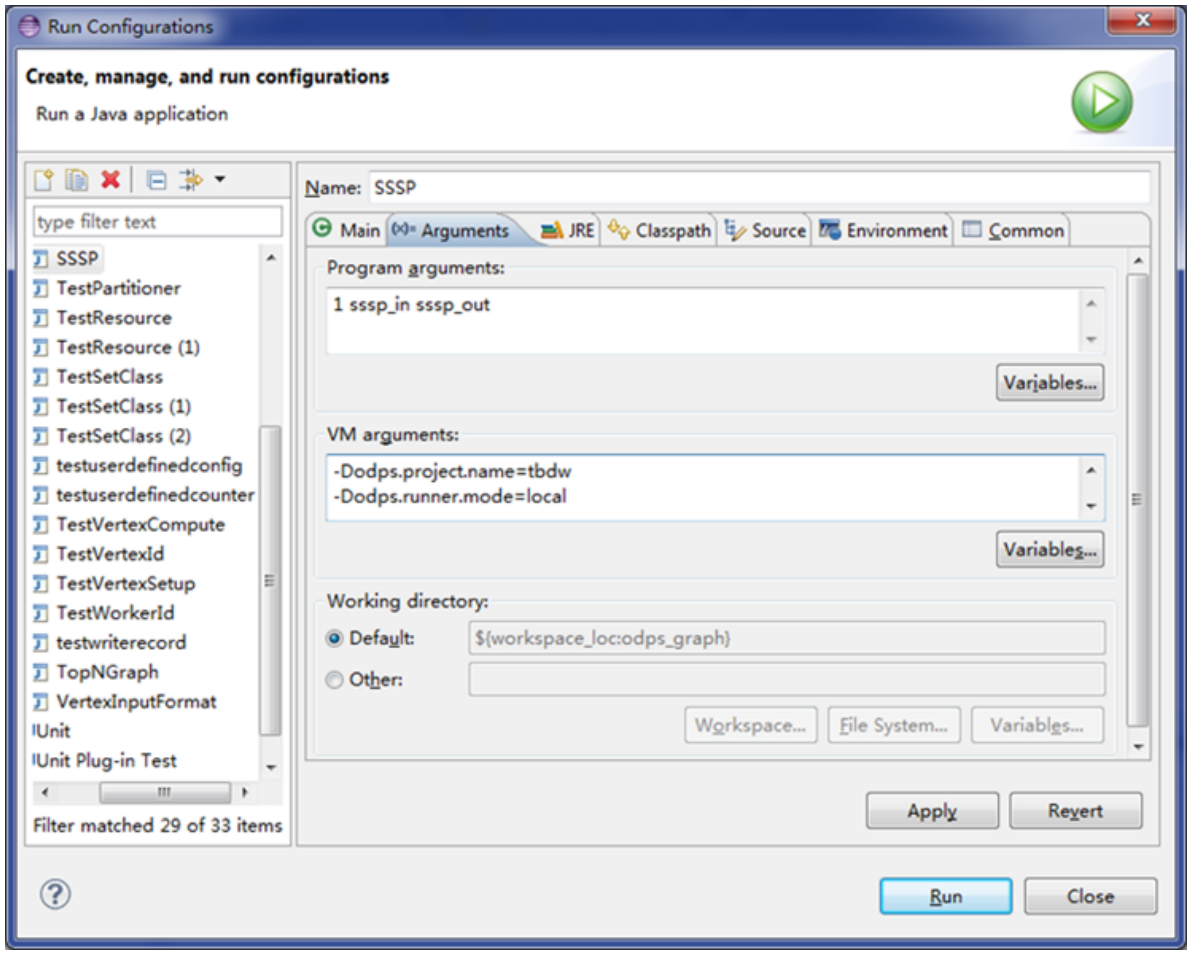
26.8.4.2 本地调试

MaxCompute Graph支持本地调试模式，可以使用Eclipse进行断点调试。断点调试的具体步骤如下。

操作步骤

1. 选择Eclipse工程，右键单击Graph作业主程序（包含 main 函数）文件，配置其运行参数，如下图所示。

   图 778: 本地调试示意图



2. 在**Arguments tab**页签中，做如下设置，作为主程序的输入参数。

   - Program arguments参数为：1 sssp_in sssp_out。
   - VM arguments参数为：Dodps.runner.mode=local ，Dodps.project.name=<project.name> ，Dodps.end.point=<end.point> ，Dodps.access.id=<access.id> ，Dodps.access.key=<access.key>。
   - 对于本地模式（即odps.end.point参数不指定），需要在warehouse创建sssp_in，sssp_out表，为输入表sssp_in添加数据，输入数据如下。

     ```
     1,"2:2,3:1,4:4"
     2,"1:2,3:2,4:1"
     3,"1:1,2:2,5:1"
     4,"1:4,2:1,5:1"
     5,"3:1,4:1"
     ```

     说明: 关于warehouse的介绍请参考：*MapReduce本地运行*。

3. 单击**Run**，在本地运行SSSP。

其中，参数设置可参考MaxCompute客户端中conf/odps_config.ini的设置，上述是几个常用参数，其他参数的说明如下：

- odps.runner.mode：取值为local，本地调试功能必须指定。
- odps.project.name：指定当前project，必须指定。
- odps.end.point：指定当前MaxCompute服务的地址，可以不指定。如果不指定，只从warehouse读取表或资源的meta 和数据，不存在则报错；如果指定，会先从warehouse读取，不存在时会远程连接odps读取。
- odps.access.id：连接MaxCompute服务的id，只在指定odps.end.point时有效。
- odps.access.key：连接MaxCompute服务的 key，只在指定odps.end.point时有效。
- odps.cache.resources：指定使用的资源列表，效果与jar命令的-resources相同。
- odps.local.warehouse：本地warehouse路径，不指定时默认为./warehouse。

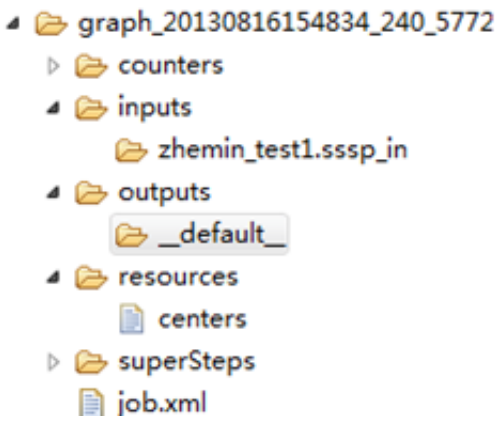
在Eclipse中本地运行SSSP的调试输出信息如下：

```
Counters: 3
com.aliyun.odps.graph.local.COUNTER
TASK_INPUT_BYTE=211
TASK_INPUT_RECORD=5
TASK_OUTPUT_BYTE=161
TASK_OUTPUT_RECORD=5
graph task finish
```

注意: 在上述的示例中，需要本地warehouse下有sssp_in及sssp_out表。sssp_in及sssp_out表的详细信息请参考：编写并运行*Graph*。

26.8.4.3 本地作业临时目录

每运行一次本地调试，都会在Eclipse工程目录下新建一个临时目录，如下图所示：



一个本地运行的Graph作业临时目录包括如下几个目录和文件：

- counters：存放作业运行的一些计数信息。
- inputs：存放作业的输入数据，优先取自本地的warehouse，如果本地没有，会通过MaxCompute SDK从服务端读取（如果设置了odps.end.point），默认一个input只读10条数据，可以通过Dodps.mapred.local.record.limit参数进行修改，但是不能超过1万条记录。
- outputs ：存放作业的输出数据，如果本地warehouse中存在输出表，outputs里的结果数据在作业执行完后会覆盖本地warehouse中对应的表。
- resources：存放作业使用的资源，与输入类似，优先取自本地的warehouse，如果本地没有，会通过MaxCompute SDK从服务端读取（如果设置了odps.end.point）。
- job.xml：作业配置。
- superstep：存放每一轮迭代的消息持久化信息。

> **注意:** 如果需要本地调试时输出详细日志，需要在src目录下放一个log4j的配置文件：**log4j.properties_odps_graph_cluster_debug**。

26.8.4.4 集群调试

在通过本地的调试之后，可以提交作业到集群进行测试，相关步骤如下：

1. 配置MaxCompute客户端。
2. 使用add jar /path/work.jar -f;命令更新jar包。
3. 使用jar命令运行作业，查看运行日志和结果数据。

> **注意:** 集群运行Graph的详细介绍请参考：编写并运行*Graph*。

26.8.4.5 性能调优

下面主要从MaxCompute Graph框架角度介绍常见性能优化的几个方面。

26.8.4.5.1 配置作业参数

对性能有所影响的GraphJob配置项包含如下参数：

- setSplitSize(long) ：输入表切分大小，单位MB，大于0，默认64。
- setNumWorkers(int) ：设置作业worker数量，范围[1, 1000]，默认值-1。 worker数由作业输入字节数和split size决定。
- setWorkerCPU(int)： MapCPU资源，100为1cpu核，范围[50，800]之间，默认200。
- setWorkerMemory(int)： Map内存资源，单位MB，范围[256M，12G]之间，默认4096。

- setMaxIteration(int) ：设置最大迭代次数，默认-1，小于或等于0时表示最大迭代次数不作为作业终止条件。
- setJobPriority(int) ：设置作业优先级，范围[0, 9]，默认9，数值越大优先级越小。

通常情况下：

1. 可以考虑使用setNumWorkers方法增加worker数目。
2. 可以考虑使用setSplitSize方法减少切分大小，提高作业载入数据速度。
3. 加大worker的cpu或内存。
4. 设置最大迭代次数，有些应用如果结果精度要求不高，可以考虑减少迭代次数，尽快结束。

接口setNumWorkers与setSplitSize配合使用，可以提高数据的载入速度。假设setNumWorkers为workerNum， setSplitSize为splitSize，总输入字节数为inputSize，则输入被切分后的块数splitNum = inputSize / splitSize，workerNum和splitNum之间的关系如下：

1. 若splitNum = workerNum，每个worker负责载入一个split。
2. 若splitNum > workerNum，每个worker负责载入一个或多个split。
3. 若splitNum < workerNum，每个worker负责载入零个或一个split。

因此，应调节workerNum和splitSize，在满足前两种情况时，数据载入比较快。迭代阶段只调节workerNum即可。如果设置runtime partitioning为false，则建议直接使用setSplitSize控制worker数量，或者保证满足前两种情况。在出现第三种情况时，部分worker上点数会为0。可以在jar命令前使用set odps.graph.split.size=<m>; set odps.graph.worker.num=<n>;与setNumWorkers和setSplitSize等效。

另外一种常见的性能问题是数据倾斜，反应到Counters就是某些worker处理的点或边数量远远超过其他worker。

数据倾斜的原因通常是某些key对应的点、边，或者消息的数量远远超出其他key，这些key被分到少量的worker处理，从而导致这些worker相对于其他运行时间长很多，解决方法如下：

- 可以尝试Combiner，把这些key对应点的消息进行本地聚合，减少消息发生。
- 改进业务逻辑。

26.8.4.5.2 运用Combiner

开发人员可定义Combiner来减少存储消息的内存和网络数据流量，缩短作业的执行时间。详细介绍请参考：*Graph SDK介绍*中的Combiner部分。

26.8.4.5.3 减少数据输入量

数据量大时，读取磁盘中的数据可能会耗费一部分处理时间，因此，减少需要读取的数据字节数可以提高总体的吞吐量，从而提高作业性能。可供选择的方法有如下几种：

- 减少数据输入量：对某些决策性质的应用，处理数据采样后子集所得到的结果只可能影响结果的精度，而并不会影响整体的准确性，因此可以考虑先对数据进行特定采样后再导入输入表中进行处理。
- 避免读取用不到的字段：MaxCompute Graph框架的TableInfo类支持读取指定的列（以列名数组方式传入），而非整个表或表分区，这样也可以减少数据的输入量，提高作业性能。

26.8.4.5.4 内置jar包

如下的jar包会默认加载到运行Graph程序的JVM中，用户可以不必上传这些资源，也不必在命令行的- libjars带上这些jar包。

- commons-codec-1.3.jar
- commons-io-2.0.1.jar
- commons-lang-2.5.jar
- commons-logging-1.0.4.jar
- commons-logging-api-1.0.4.jar
- guava-14.0.jar
- json.jar
- log4j-1.2.15.jar
- slf4j-api-1.4.3.jar
- slf4j-log4j12-1.4.3.jar
- xmlenc-0.52.jar

> **注意：** 在运行JVM的CLASSPATH里，上述内置jar包会放在用户jar包的前面，所以可能产生版本冲突。例如：用户的程序中使用了commons-codec-1.5.jar某个类的函数，但是这个函数不在commons-codec-1.3.jar中，此时只能看1.3版本里是否有满足用户需求的实现，如果没有，则需要等待MaxCompute升级新版本。

26.8.5 应用限制

- 单个job引用的resource数量不超过256个，table、archive按照一个单位计算。
- 单个job引用的resource总计字节数大小不超过512M。
- 单个job的输入路数不能超过1024（输入表的个数不能超过64），单个job的输出路数不能超过256。
- 多路输出中指定的label不能为null或者为空字符串，长度不能超过256，只能包括A-Z，a-z，0-9，_，#，.，-等。
- 单个job中自定义counter的数量不能超过64，counter的group name和counter name中不能带有#，两者长度和不能超过100。
- 单个job的worker数由框架计算得出，最大为1000，超过报错。
- 单个worker占用cpu 默认为200，范围[50, 800]。
- 单个worker占用memory默认为4096，范围[256M, 12G]。
- 单个worker重复读一个resource次数限制不大于64 次。
- plit size默认为64M，用户可设置，范围：0 < split_size <= (9223372036854775807 > 20)。
- MaxCompute Graph程序中的GraphLoader/Vertex/Aggregator等在集群运行时，受到Java沙箱的限制（Graph作业的主程序则不受此限制），具体限制请参考：*Java* 沙箱限制介绍。

26.8.6 示例程序

26.8.6.1 单源最短距离

Dijkstra算法是求解有向图中单源最短距离（Single Source Shortest Path，简称SSSP）的经典算法。

最短距离：对一个有权重的有向图G=(V,E)，从一个源点s到汇点v有很多路径，其中边权和最小的路径，称从s到v的最短距离。算法基本原理如下：

- 初始化：源点s到s自身的距离为零（d[s]=0），其他点u到s的距离为无穷（d[u]=∞）。
- 迭代：若存在一条从u到v的边，那么从s到v的最短距离更新为： d[v]=min(d[v], d[u]+weight(u, v))，直到所有的点到s的距离不再发生变化时迭代结束。



说明： 从算法基本原理可以看出，这个算法非常适合使用MaxCompute Graph程序进行求解：每个点维护到源点的当前最短距离值，当这个值变化时，将新值加上边的权值发送消息通知其邻接点，下一轮迭代时，邻接点根据收到的消息更新其当前最短距离，当所有点当前最短距离不再变化时，迭代终止。

示例如下：

```
import java.io.IOException;
import com.aliyun.odps.io.WritableRecord;
import com.aliyun.odps.graph.Combiner;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.Edge;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.data.TableInfo;
public class SSSP {
public static final String START_VERTEX = "sssp.start.vertex.id";
/**定义 SSSPVertex ，其中：
* 点值表示该点到源点 startVertexId 的当前最短距离；
* compute()方法使用迭代公式：d[v]=min(d[v], d[u]+weight(u, v)) 更新点值；
* cleanup() 方法把点及其到源点的最短距离写到结果表中；
 **/
public static class SSSPVertex extends
Vertex<LongWritable, LongWritable, LongWritable, LongWritable> {
private static long startVertexId = -1;
public SSSPVertex() {
this.setValue(new LongWritable(Long.MAX_VALUE));
}
public boolean isStartVertex(
ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable> context) {
if (startVertexId == -1) {
String s = context.getConfiguration().get(START_VERTEX);
startVertexId = Long.parseLong(s);
}
return getId().get() == startVertexId;
}
@Override
public void compute(
ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable> context,
 Iterable<LongWritable> messages) throws IOException {
long minDist = isStartVertex(context) ? 0 : Integer.MAX_VALUE;
for (LongWritable msg : messages) { if (msg.get() < minDist) {
minDist = msg.get();
}
}
if (minDist < this.getValue().get()) {
this.setValue(new LongWritable(minDist));
if (hasEdges()) {
for (Edge<LongWritable, LongWritable> e : this.getEdges()) {
context.sendMessage(e.getDestVertexId(), new LongWritable(minDist + e.getValue().get()));
}
}
} else {
voteToHalt(); //当点值没发生变化时，调用 voteToHalt() 告诉框架该点进入 halt 状态，当所有点都进入 halt 状态时计算结束；
}
}
@Override
public void cleanup(
WorkerContext<LongWritable, LongWritable, LongWritable, LongWritable> context) throws
 IOException {
context.write(getId(), getValue());
```

```
}
}
/** 定义 MinLongCombiner，对发送给同一个点的消息进行合并，优化性能，减少内存占用； **/
public static class MinLongCombiner extends
Combiner<LongWritable, LongWritable> {
@Override
public void combine(LongWritable vertexId, LongWritable combinedMessage, LongWritable
 messageToCombine) throws IOException {
if (combinedMessage.get() > messageToCombine.get()) {
combinedMessage.set(messageToCombine.get());
}
}
}
/**定义 SSSPVertexReader 类，加载图，将表中每一条记录解析为一个点，记录的第一列是点标
识，第二列存储该点起始的所有的边集，内容如：2:2,3:1,4:4；**/
public static class SSSPVertexReader extends
GraphLoader<LongWritable, LongWritable, LongWritable, LongWritable> {
@Override
public void load(LongWritable recordNum, WritableRecord record,
MutationContext<LongWritable, LongWritable, LongWritable, LongWritable> context) throws
 IOException {
SSSPVertex vertex = new SSSPVertex();
vertex.setId((LongWritable) record.get(0));
String[] edges = record.get(1).toString().split(",");
for (int i = 0; i < edges.length; i++) {
String[] ss = edges[i].split(":");
vertex.addEdge(new LongWritable(Long.parseLong(ss[0])), new
 LongWritable(Long.parseLong(ss[1])));
}
context.addVertexRequest(vertex);
}
}
public static void main(String[] args) throws IOException { if (args.length < 2) {
System.out.println("Usage: <startnode> <input> <output>");
System.exit(-1);
}
GraphJob job = new GraphJob(); //定义 GraphJob，指定 Vertex/GraphLoader/Combiner 等的实
现，指定输入输出表。
job.setGraphLoaderClass(SSSPVertexReader.class);
job.setVertexClass(SSSPVertex.class);
job.setCombinerClass(MinLongCombiner.class);
job.set(START_VERTEX, args[0]);
job.addInput(TableInfo.builder().tableName(args[1]).build());
job.addOutput(TableInfo.builder().tableName(args[2]).build());
long startTime = System.currentTimeMillis(); job.run();
System.out.println("Job Finished in " + (System.currentTimeMillis() - startTime) / 1000.0 + "
 seconds");
}
}
```

26.8.6.2 PageRank

PageRank算法是计算网页排名的经典算法，详细介绍请参考： *PageRank*。该算法的输入是一个有向图G，其中顶点表示网页，如果存在网页A到网页B的链接， 那么存在联接A到B的边。算法基本原理如下：

- 初始化：点值表示PageRank的rank值（double 类型），初始时，所有点取值为1/TotalNumVertices。
- 迭代公式：PageRank(i)=0.15/TotalNumVertices+0.85*sum，其中sum为所有指向i点的点（设为j）PageRank(j)/out_degree(j) 的累加值。

> **说明:** 从算法基本原理可以看出，这个算法非常适合使用MaxCompute Graph程序进行求解：每个点j维护其PageRank值，每一轮迭代都将PageRank(j)/out_degree(j) 发给其邻接点（向其投票），下一轮迭代时，每个点根据迭代公式重新计算PageRank取值。

示例如下：

```
import java.io.IOException;
import org.apache.log4j.Logger;
import com.aliyun.odps.io.WritableRecord;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.io.Writable;
public class PageRank {
private final static Logger LOG = Logger.getLogger(PageRank.class);
/**
* 定义 PageRankVertex ，其中：
* 点值表示该点（网页）的当前 PageRank 取值；
* compute()方法使用迭代公式：PageRank(i)=0.15/TotalNumVertices+0.85*sum > 更新点值；
* cleanup() 方法把点及其 PageRank 取值写到结果表中；
**/
public static class PageRankVertex extends
Vertex<Text, DoubleWritable, NullWritable, DoubleWritable> {
@Override
public void compute(
ComputeContext<Text, DoubleWritable, NullWritable, DoubleWritable> context,
 Iterable<DoubleWritable> messages) throws IOException {
if (context.getSuperstep() == 0) {
setValue(new DoubleWritable(1.0 / context.getTotalNumVertices()));
} else if (context.getSuperstep() >= 1) { double sum = 0;
for (DoubleWritable msg : messages) { sum += msg.get();
}
DoubleWritable vertexValue = new DoubleWritable( (0.15f / context.getTotalNumVertices()) +
 0.85f * sum);
setValue(vertexValue);
}
if (hasEdges()) {
context.sendMessageToNeighbors(this, new DoubleWritable(getValue()
.get() / getEdges().size()));
}
}
@Override
```

```
public void cleanup(
WorkerContext<Text, DoubleWritable, NullWritable, DoubleWritable> context) throws IOException {
context.write(getId(), getValue());
}
}
/**定义 PageRankVertexReader 类，加载图，将表中每一条记录解析为一个点，记录的第一列是起点，其他列为终点；**/
public static class PageRankVertexReader extends
GraphLoader<Text, DoubleWritable, NullWritable, DoubleWritable> {
@Override public void load(
LongWritable recordNum, WritableRecord record,
MutationContext<Text, DoubleWritable, NullWritable, DoubleWritable> context) throws IOException {
PageRankVertex vertex = new PageRankVertex();
vertex.setValue(new DoubleWritable(0));
vertex.setId((Text) record.get(0));
System.out.println(record.get(0));
for (int i = 1; i < record.size(); i++) {
Writable edge = record.get(i);
System.out.println(edge.toString());
if (!(edge.equals(NullWritable.get()))) {
vertex.addEdge(new Text(edge.toString()), NullWritable.get());
}
}
LOG.info("vertex edgs size: " + (vertex.hasEdges() ? vertex.getEdges().size() : 0));
context.addVertexRequest(vertex);
}
}
private static void printUsage() {
System.out.println("Usage: <in> <out> [Max iterations (default 30)]");
System.exit(-1);
}
public static void main(String[] args) throws IOException { if (args.length < 2)
printUsage();
GraphJob job = new GraphJob(); //定义 GraphJob，指定 Vertex/GraphLoader等的实现，以及最大迭代次数（默认 > 30），并指定输入输出表。
job.setGraphLoaderClass(PageRankVertexReader.class);
job.setVertexClass(PageRankVertex.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
// default max iteration is 30
job.setMaxIteration(30); if (args.length >= 3)
job.setMaxIteration(Integer.parseInt(args[2]));
long startTime = System.currentTimeMillis(); job.run();
System.out.println("Job Finished in "
+ (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
}
}
```

26.8.6.3 K-均值聚类

k-均值聚类（Kmeans）算法是非常基础并大量使用的聚类算法。算法基本原理如下：以空间中k个点为中心进行聚类，对最靠近他们的点进行归类。通过迭代的方法，逐次更新各聚类中心的值，直至得到最好的聚类结果。

假设要把样本集分为k个类别，算法实现如下：

- 适当选择k个类的初始中心。
- 在第i次迭代中，对任意一个样本，求其到k个中心的距离，将该样本归到距离最短的中心所在的类。
- 利用均值等方法更新该类的中心值。
- 对于所有的k个聚类中心，如果利用上两步的迭代法更新后，值保持不变或者小于某个阈值，则迭代结束，否则继续迭代。

示例如下：

```
import java.io.DataInput; import java.io.DataOutput;
import java.io.IOException;
importorg.apache.log4j.Logger;
import com.aliyun.odps.io.WritableRecord;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.io.Tuple;
import com.aliyun.odps.io.Writable;
public class Kmeans {
private final static Logger LOG = Logger.getLogger(Kmeans.class);
/**定义 KmeansVertex，compute() 方法非常简单，只是调用上下文对象的 aggregate 方法传入当前点的取值（Tuple 类型，向量表示）**/
public static class KmeansVertex extends
Vertex<Text, Tuple, NullWritable, NullWritable> {
@Override
public void compute(
ComputeContext<Text, Tuple, NullWritable, NullWritable> context, Iterable<NullWritable> messages) throws IOException { context.aggregate(getValue());
}
}
/**定义 KmeansVertexReader 类，加载图，将表中每一条记录解析为一个点，点标识无关紧要，这里取传入的 recordNum 序号作为标识，点值为记录的所有列组成的 Tuple **/
public static class KmeansVertexReader extends
GraphLoader<Text, Tuple, NullWritable, NullWritable> {
@Override
public void load(LongWritable recordNum, WritableRecord record, MutationContext<Text, Tuple, NullWritable, NullWritable> context) throws IOException {
KmeansVertex vertex = new KmeansVertex();
vertex.setId(new Text(String.valueOf(recordNum.get())));
vertex.setValue(new Tuple(record.getAll()));
context.addVertexRequest(vertex);
}
}
public static class KmeansAggrValue implements Writable {
Tuple centers = new Tuple();
Tuple sums = new Tuple();
Tuple counts = new Tuple();
```

```
@Override
public void write(DataOutput out) throws IOException {
centers.write(out);
sums.write(out); counts.write(out);
}
@Override
public void readFields(DataInput in) throws IOException {
centers = new Tuple();
centers.readFields(in);
sums = new Tuple();
sums.readFields(in);
counts = new Tuple();
counts.readFields(in);
}
@Override
public String toString() {
return "centers " + centers.toString() + ", sums " + sums.toString()
+ ", counts " + counts.toString();
}
}
/**
* 定义 KmeansAggregator，这个类封装了 Kmeans 算法的主要逻辑，其中：
*          createInitialValue          为每一轮迭代创建初始值（k          类中心点），若是第一轮迭代（superstep=0），该取值为初始中心点，否则取值为上一轮结束时的新中心点；
*    aggregate    方法为每个点计算其到各个类中心的距离，并归为距离最短的类，并更新该类的 sum 和 count；
* merge 方法合并来自各个 worker 收集的 sum 和 count；
* terminate 方法根据各个类的 sum 和 count 计算新的中心点，若新中心点与之前的中心点距离小于某个阈值或者迭代次数到达最大迭代次数设置，则终止迭代（返回 false），写最终的中心点到结果表
**/
public static class KmeansAggregator extends Aggregator<KmeansAggrValue> {
@SuppressWarnings("rawtypes")
@Override
public KmeansAggrValue createInitialValue(WorkerContext context)
throws IOException {
KmeansAggrValue aggrVal = null;
if (context.getSuperstep() == 0) {
aggrVal = new KmeansAggrValue();
aggrVal.centers = new Tuple();
aggrVal.sums = new Tuple();
aggrVal.counts = new Tuple();
byte[] centers = context.readCacheFile("centers");
String lines[] = new String(centers).split("\n");
for (int i = 0;
i < lines.length; i++) { String[] ss = lines[i].split(",");
Tuple center = new Tuple();
Tuple sum = new Tuple();
for (int j = 0; j < ss.length; ++j) {
center.append(new DoubleWritable(Double.valueOf(ss[j].trim())));
sum.append(new DoubleWritable(0.0));
}
LongWritable count = new LongWritable(0);
aggrVal.sums.append(sum); aggrVal.counts.append(count);
aggrVal.centers.append(center);
}
} else {
aggrVal = (KmeansAggrValue) context.getLastAggregatedValue(0);
}
return aggrVal;
}
```

```
@Override
public void aggregate(KmeansAggrValue value, Object item) {
int min = 0;
double mindist = Double.MAX_VALUE;
Tuple point = (Tuple) item;
for (int i = 0;
i < value.centers.size();
i++) { Tuple center = (Tuple) value.centers.get(i);
// use Euclidean Distance, no need to calculate sqrt
double dist = 0.0d;
for (int j = 0; j < center.size(); j++) {
double v = ((DoubleWritable) point.get(j)).get()
- ((DoubleWritable) center.get(j)).get();
dist += v * v;
}
if (dist < mindist) { mindist = dist; min = i;
}
}
// update sum and count
Tuple sum = (Tuple) value.sums.get(min);
for (int i = 0;
i < point.size(); i++) {
DoubleWritable s = (DoubleWritable) sum.get(i); s.set(s.get() + ((DoubleWritable)
 point.get(i)).get());
}
LongWritable count = (LongWritable) value.counts.get(min);
count.set(count.get() + 1);
}
@Override
public void merge(KmeansAggrValue value, KmeansAggrValue partial) {
for (int i = 0; i < value.sums.size(); i++) {
Tuple sum = (Tuple) value.sums.get(i);
Tuple that = (Tuple) partial.sums.get(i);
for (int j = 0; j < sum.size(); j++) {
DoubleWritable s = (DoubleWritable) sum.get(j);
s.set(s.get() + ((DoubleWritable) that.get(j)).get());
}
}
for (int i = 0; i < value.counts.size(); i++) {
LongWritable count = (LongWritable) value.counts.get(i);
count.set(count.get() + ((LongWritable) partial.counts.get(i)).get());
}
}
@SuppressWarnings("rawtypes")
@Override
public boolean terminate(WorkerContext context, KmeansAggrValue value) throws IOException {
// compute new centers
Tuple newCenters = new Tuple(value.sums.size());
for (int i = 0; i < value.sums.size(); i++) {
Tuple sum = (Tuple) value.sums.get(i);
Tuple newCenter = new Tuple(sum.size());
LongWritable c = (LongWritable) value.counts.get(i);
for (int j = 0; j < sum.size(); j++) {
DoubleWritable s = (DoubleWritable) sum.get(j);
double val = s.get() / c.get();
newCenter.set(j, new DoubleWritable(val));
// reset sum for next iteration
s.set(0.0d);
}
// reset count for next iteration
c.set(0);
newCenters.set(i, newCenter);
```

```
}
// update centers
Tuple oldCenters = value.centers; value.centers = newCenters;
LOG.info("old centers: " + oldCenters + ", new centers: " + newCenters);
// compare new/old centers
boolean converged = true;
for (int i = 0; i < value.centers.size() && converged; i++) {
Tuple oldCenter = (Tuple) oldCenters.get(i);
Tuple newCenter = (Tuple) newCenters.get(i); double sum = 0.0d;
for (int j = 0; j < newCenter.size(); j++) {
double v = ((DoubleWritable) newCenter.get(j)).get() - ((DoubleWritable) oldCenter.get(j)).get();
sum += v * v;
}
double dist = Math.sqrt(sum);
LOG.info("old center: " + oldCenter + ", new center: " + newCenter + ", dist: " + dist);
// converge threshold for each center: 0.05
converged = dist < 0.05d;
}
if (converged || context.getSuperstep() == context.getMaxIteration() - 1) {
// converged or reach max iteration, output centers
for (int i = 0; i < value.centers.size(); i++) { context.write(((Tuple) value.centers.get(i)).toArray());
}
// true means to terminate iteration
return true;
}
// false means to continue iteration
return false;
}
}
private static void printUsage() {
System.out.println("Usage: <in> <out> [Max iterations (default 30)]");
System.exit(-1);
}
/**定义 GraphJob，指定 Vertex/GraphLoader/Aggregator 等的实现，以及最大迭代次数（默认 30），并指定输入输出表。**/
public static void main(String[] args) throws IOException {
if (args.length < 2)
printUsage();
GraphJob job = new GraphJob();
job.setGraphLoaderClass(KmeansVertexReader.class);
job.setRuntimePartitioning(false);
// job.setRuntimePartitioning(false)，对于 Kmeans 算法，加载图是不需要进行点的分发，设置 RuntimePartitioning 为false，提升加载图时的性能。
job.setVertexClass(KmeansVertex.class);
job.setAggregatorClass(KmeansAggregator.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
// default max iteration is 30
job.setMaxIteration(30); if (args.length >= 3)
job.setMaxIteration(Integer.parseInt(args[2]));
long start = System.currentTimeMillis(); job.run();
System.out.println("Job Finished in " + (System.currentTimeMillis() - start) / 1000.0 + " seconds");
}
```

```
}
```

26.8.6.4 BiPartiteMatchiing

二分图是指图的所有顶点可分为两个集合，每条边对应的两个顶点分别属于这两个集合。对于一个二分图G，M是它的一个子图，如果M的边集中任意两条边都不依附于同一个顶点，则称M为一个匹配。二分图匹配常用于有明确供需关系场景（如交友网站等）下的信息匹配行为。

算法实现如下：

- 从左边第1个顶点开始，挑选未匹配点进行搜索，寻找增广路。
- 如果经过一个未匹配点，说明寻找成功。
- 更新路径信息，匹配边数+1，停止搜索。
- 如果一直没有找到增广路，则不再从这个点开始搜索。

示例如下：

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.util.Random;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.io.WritableRecord;
public class BipartiteMatching {
private static final Text UNMATCHED = new Text("UNMATCHED");
public static class TextPair implements Writable {
public Text first; public Text second;
public TextPair() { first = new Text();
second = new Text();
}
public TextPair(Text first, Text second) {
this.first = new Text(first);
this.second = new Text(second);
}
@Override
public void write(DataOutput out) throws IOException {
first.write(out);
second.write(out);
}
@Override
public void readFields(DataInput in) throws IOException {
first = new Text();
first.readFields(in);
second = new Text();
```

```
second.readFields(in);
}
@Override
public String toString() { return first + ": " + second;
}
}
public static class BipartiteMatchingVertexReader extends
GraphLoader<Text, TextPair, NullWritable, Text> {
@Override
public void load(LongWritable recordNum, WritableRecord record, MutationContext<Text, TextPair, NullWritable, Text> context) throws IOException {
BipartiteMatchingVertex vertex = new BipartiteMatchingVertex();
vertex.setId((Text) record.get(0));
vertex.setValue(new TextPair(UNMATCHED, (Text) record.get(1)));
String[] adjs = record.get(2).toString().split(",");
for (String adj:adjs) {
vertex.addEdge(new Text(adj), null);
}
context.addVertexRequest(vertex);
}
}
public static class BipartiteMatchingVertex extends
Vertex<Text, TextPair, NullWritable, Text> {
private static final Text LEFT = new Text("LEFT");
private static final Text RIGHT = new Text("RIGHT");
private static Random rand = new Random();
@Override
public void compute(
ComputeContext<Text, TextPair, NullWritable, Text> context, Iterable<Text> messages) throws IOException {
if (isMatched()) { voteToHalt();
return;
}
switch ((int) context.getSuperstep() % 4) {
case 0:
if (isLeft()) {
context.sendMessageToNeighbors(this, getId());
}
break;
case 1:
if (isRight()) {
Text luckyLeft = null;
for (Text message : messages) { if (luckyLeft == null) {
luckyLeft = new Text(message);
} else {
if (rand.nextInt(1) == 0) { luckyLeft.set(message);
}
}
}
if (luckyLeft != null) { context.sendMessage(luckyLeft, getId());
}
}
break;
case 2:
if (isLeft()) {
Text luckyRight = null;
for (Text msg : messages) { if (luckyRight == null) {
luckyRight = new Text(msg);
} else {
if (rand.nextInt(1) == 0) { luckyRight.set(msg);
}
}
```

```
}
if (luckyRight != null) {
setMatchVertex(luckyRight);
context.sendMessage(luckyRight, getId());
}
}
break; case 3:
if (isRight()) {
for (Text msg : messages) { setMatchVertex(msg);
}
}
break;
}
}
@Override
public void cleanup(
WorkerContext<Text, TextPair, NullWritable, Text> context) throws IOException {
context.write(getId(), getValue().first);
}
private boolean isMatched() {
return !getValue().first.equals(UNMATCHED);
}
private boolean isLeft() {
return getValue().second.equals(LEFT);
}
private boolean isRight() {
return getValue().second.equals(RIGHT);
}
private void setMatchVertex(Text matchVertex) { getValue().first.set(matchVertex);
}
}
private static void printUsage() {
System.err.println("BipartiteMatching <input> <output> [maxIteration]");
}
public static void main(String[] args) throws IOException { if (args.length < 2) {
printUsage();
}
GraphJob job = new GraphJob();
job.setGraphLoaderClass(BipartiteMatchingVertexReader.class);
job.setVertexClass(BipartiteMatchingVertex.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
int maxIteration = 30;
if (args.length > 2) {
maxIteration = Integer.parseInt(args[2]);
}
job.setMaxIteration(maxIteration);
job.run();
}
}
```

26.8.6.5 强连通分量

在有向图中，如果从任意一个顶点出发，都能通过图中的边到达图中的每一个顶点，则称之为强连通图。一张有向图的顶点数极大的强连通子图称为强连通分量。此算法示例基于 parallel Coloring algorithm。

每个顶点包含两个部分：

- colorID：在向前遍历过程中存储顶点v的颜色，在计算结束时，具有相同 colorID 的顶点属于一个强连通分量。
- transposeNeighbors：存储输入图的转置图中顶点v的邻居 ID。

算法实现如下：

- 生成转置图：包含两个超步，首先每个顶点发送ID到其出边对应的邻居，这些ID在第二个超步中会存为transposeNeighbors值。
- 修剪：一个超步，每个只有一个入边或出边的顶点，将其colorID设置为自身ID，状态设为不活跃，后面传给该顶点的信号被忽略。
- 向前遍历：顶点包括两个子过程（超步），启动和休眠。在启动阶段，每个顶点将其colorID设置为自身ID，同时将其ID传给出边对应的邻居。 休眠阶段，顶点使用其收到的最大colorID更新自身colorID，并传播其colorID，直到colorID收敛。当colorID收敛，master进程将全局对象设置为向后遍历。
- 向后遍历：同样包含两个子过程，启动和休眠。启动阶段，每一个ID等于colorID的顶点将其ID传递给其转置图邻居顶点， 同时将自身状态设置为不活跃，后面传给该顶点的信号可忽略。在每一个休眠步骤，每个顶点接收到与其colorID匹配的信号，并将其colorID在转置图中传播，随后设置自身状态为不活跃。该步骤结束后如果仍有活跃顶点，则回到修剪步骤。

示例如下：

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.BooleanWritable;
import com.aliyun.odps.io.IntWritable;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.Tuple;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.io.WritableRecord;
/**
* Definition from Wikipedia:
* In the mathematical theory of directed graphs, a graph is said
* to be strongly connected if every vertex is reachable from every
* other vertex. The strongly connected components of an arbitrary
* directed graph form a partition into subgraphs that are themselves
* strongly connected.
*
* Algorithms with four phases as follows.
```

```
* 1. Transpose Graph Formation: Requires two supersteps. In the first
* superstep, each vertex sends a message with its ID to all its > outgoing
* neighbors, which in the second superstep are stored > in transposeNeighbors.
*
* 2. Trimming: Takes one superstep. Every vertex with only in-coming or
* only outgoing edges (or neither) sets its colorID to its own ID and
* becomes inactive. Messages subsequently sent to the vertex > are ignored.
*
* 3. Forward-Traversal: There are two sub phases: Start and Rest. In the
* Start phase, each vertex sets its colorID to its own ID and > propagates
* its ID to its outgoing neighbors. In the Rest phase, vertices update
* their own colorIDs with the minimum colorID they have seen, and > propagate
* their colorIDs, if updated, until the colorIDs converge.
* Set the phase to Backward-Traversal when the colorIDs converge.
*
* 4. Backward-Traversal: We again break the phase into Start and Rest.
* In Start, every vertex whose ID equals its colorID propagates its ID > to
* the vertices in transposeNeighbors and sets itself inactive. > Messages
* subsequently sent to the vertex are ignored. In each of the Rest > phase supersteps,
* each vertex receiving a message that matches its colorID: (1) > propagates
* its colorID in the transpose graph; (2) sets itself inactive. > Messages
* subsequently sent to the vertex are ignored. Set the phase back to Trimming
* if not all vertex are inactive.
*
* http://ilpubs.stanford.edu:8090/1077/3/p535-salihoglu.pdf
**/
public class StronglyConnectedComponents {
public final static int STAGE_TRANSPOSE_1 = 0;
public final static int STAGE_TRANSPOSE_2 = 1;
public final static int STAGE_TRIMMING = 2;
public final static int STAGE_FW_START = 3;
public final static int STAGE_FW_REST = 4;
public final static int STAGE_BW_START = 5;
public final static int STAGE_BW_REST = 6;
/**
* The value is composed of component id, incoming neighbors,
* active status and updated status.
**/
public static class MyValue implements Writable {
LongWritable sccID;// strongly connected component id
Tuple inNeighbors; // transpose neighbors
BooleanWritable active; // vertex is active or not
BooleanWritable updated; // sccID is updated or not
public MyValue() {
this.sccID = new LongWritable(Long.MAX_VALUE);
this.inNeighbors = new Tuple();
this.active = new BooleanWritable(true);
this.updated = new BooleanWritable(false);
}
public void setSccID(LongWritable sccID) {
this.sccID = sccID;
}
public LongWritable getSccID() {
return this.sccID;
}
public void setInNeighbors(Tuple inNeighbors) {
this.inNeighbors = inNeighbors;
}
public Tuple getInNeighbors() {
return this.inNeighbors;
}
public void addInNeighbor(LongWritable neighbor) {
```

```
this.inNeighbors.append(new LongWritable(neighbor.get()));
}
public boolean isActive() {
return this.active.get();
}
public void setActive(boolean status) {
this.active.set(status);
}
public boolean isUpdated() {
return this.updated.get();
}
public void setUpdated(boolean update) {
this.updated.set(update);
}
@Override
public void write(DataOutput out) throws IOException {
this.sccID.write(out);
this.inNeighbors.write(out);
this.active.write(out);
this.updated.write(out);
}
@Override
public void readFields(DataInput in) throws IOException {
this.sccID.readFields(in);
this.inNeighbors.readFields(in);
this.active.readFields(in);
this.updated.readFields(in);
}
@Override
public String toString() {
StringBuilder sb = new StringBuilder();
sb.append("sccID: " + sccID.get());
sb.append(" inNeighbores: " + inNeighbors.toDelimitedString(','));
sb.append(" active: " + active.get());
sb.append(" updated: " + updated.get());
return sb.toString();
}
}
public static class SCCVertex extends
Vertex<LongWritable, MyValue, NullWritable, LongWritable> {
public SCCVertex() {
this.setValue(new MyValue());
}
@Override
public void compute(
ComputeContext<LongWritable, MyValue, NullWritable, LongWritable> context,
 Iterable<LongWritable> msgs) throws IOException {
// Messages sent to inactive vertex are ignored.
if (!this.getValue().isActive()) {
this.voteToHalt(); return;
}
int stage = ((SCCAggrValue)context.getLastAggregatedValue(0)).getStage(); switch (stage) {
case STAGE_TRANSPOSE_1:
context.sendMessageToNeighbors(this, this.getId());
break;
case STAGE_TRANSPOSE_2:
for (LongWritable msg: msgs) {
this.getValue().addInNeighbor(msg);
}
case STAGE_TRIMMING:
this.getValue().setSccID(getId());
if (this.getValue().getInNeighbors().size() == 0 || this.getNumEdges() == 0) {
```

```
this.getValue().setActive(false);
}
break;
case STAGE_FW_START: this.getValue().setSccID(getId());
context.sendMessageToNeighbors(this, this.getValue().getSccID());
break;
case STAGE_FW_REST:
long minSccID = Long.MAX_VALUE;
for (LongWritable msg : msgs) {
if (msg.get() < minSccID) { minSccID = msg.get();
}
}
if (minSccID < this.getValue().getSccID().get()) {
this.getValue().setSccID(new LongWritable(minSccID));
context.sendMessageToNeighbors(this, this.getValue().getSccID());
this.getValue().setUpdated(true);
} else {
this.getValue().setUpdated(false);
}
break;
case STAGE_BW_START:
if (this.getId().equals(this.getValue().getSccID())) {
for (Writable neighbor : this.getValue().getInNeighbors().getAll()) {
context.sendMessage((LongWritable)neighbor, this.getValue().getSccID());
}
this.getValue().setActive(false);
}
break;
case STAGE_BW_REST: this.getValue().setUpdated(false);
for (LongWritable msg : msgs) {
if (msg.equals(this.getValue().getSccID())) {
for (Writable neighbor : this.getValue().getInNeighbors().getAll()) {
context.sendMessage((LongWritable)neighbor, this.getValue().getSccID());
}
this.getValue().setActive(false);
this.getValue().setUpdated(true);
break;
}
}
break;
}
context.aggregate(0, getValue());
}
@Override
public void cleanup(
WorkerContext<LongWritable, MyValue, NullWritable, LongWritable> context)
throws IOException {
context.write(getId(), getValue().getSccID());
}
}
/**
* The SCCAggrValue maintains global stage and graph updated and > active status.
* updated is true only if one vertex is updated.
* active is true only if one vertex is active.
*/
public static class SCCAggrValue implements Writable {
IntWritable stage = new IntWritable(STAGE_TRANSPOSE_1);
BooleanWritable updated = new BooleanWritable(false);
BooleanWritable active = new BooleanWritable(false);
public void setStage(int stage) { this.stage.set(stage);
}
public int getStage() { return this.stage.get();
```

```
}
public void setUpdated(boolean updated) {
this.updated.set(updated);
}
public boolean getUpdated() {
return this.updated.get();
}
public void setActive(boolean active) {
this.active.set(active);
}
public boolean getActive() {
return this.active.get();
}
@Override
public void write(DataOutput out) throws IOException {
this.stage.write(out);
this.updated.write(out);
this.active.write(out);
}
@Override
public void readFields(DataInput in) throws IOException {
this.stage.readFields(in);
this.updated.readFields(in);
this.active.readFields(in);
}
}
/**
* The job of SCCAggregator is to schedule global stage in > every superstep.
*/
public static class SCCAggregator extends Aggregator<SCCAggrValue> {
@SuppressWarnings("rawtypes")
@Override
public SCCAggrValue createStartupValue(WorkerContext context) throws IOException { return
 new SCCAggrValue();
}
@SuppressWarnings("rawtypes")
@Override
public SCCAggrValue createInitialValue(WorkerContext context) throws IOException {
return (SCCAggrValue) context.getLastAggregatedValue(0);
}
@Override
public void aggregate(SCCAggrValue value, Object item) throws IOException { MyValue v =
 (MyValue)item;
if ((value.getStage() == STAGE_FW_REST || value.getStage() == STAGE_BW_REST)&&
 v.isUpdated()) { value.setUpdated(true);
}
// only active vertex invoke aggregate()
value.setActive(true);
}
@Override
public void merge(SCCAggrValue value, SCCAggrValue partial) throws IOException {
boolean updated = value.getUpdated() || partial.getUpdated();
value.setUpdated(updated);
boolean active = value.getActive() || partial.getActive();
value.setActive(active);
}
@SuppressWarnings("rawtypes")
@Override
public boolean terminate(WorkerContext context, SCCAggrValue value) throws IOException {
// If all vertices is inactive, job is over.
if (!value.getActive()) { return true;
}
```

```
// state machine
switch (value.getStage()) {
case STAGE_TRANSPOSE_1:value.setStage(STAGE_TRANSPOSE_2);
break;
case STAGE_TRANSPOSE_2:value.setStage(STAGE_TRIMMING);
break;
case STAGE_TRIMMING:value.setStage(STAGE_FW_START);
break;
case STAGE_FW_START: value.setStage(STAGE_FW_REST);
break;
case STAGE_FW_REST:if (value.getUpdated()) {
value.setStage(STAGE_FW_REST);
} else {
value.setStage(STAGE_BW_START);
}
break;
case STAGE_BW_START: value.setStage(STAGE_BW_REST);
break;
case STAGE_BW_REST:if (value.getUpdated()) { value.setStage(STAGE_BW_REST);
} else { value.setStage(STAGE_TRIMMING);
}
break;
}
value.setActive(false);
value.setUpdated(false);
return false;
}
}
public static class SCCVertexReader extends
GraphLoader<LongWritable, MyValue, NullWritable, LongWritable> {
@Override public void load(
LongWritable recordNum, WritableRecord record,
MutationContext<LongWritable, MyValue, NullWritable, LongWritable> context) throws
 IOException {
SCCVertex vertex = new SCCVertex();
vertex.setId((LongWritable) record.get(0));
String[] edges = record.get(1).toString().split(",");
for (int i = 0; i < edges.length; i++) { try {
long destID = Long.parseLong(edges[i]);
vertex.addEdge(new LongWritable(destID), NullWritable.get());
} catch(NumberFormatException nfe) { System.err.println("Ignore " + nfe);
}
}
context.addVertexRequest(vertex);
}
}
public static void main(String[] args) throws IOException {
if (args.length < 2) {
System.out.println("Usage: <input> <output>");
System.exit(-1);
}
GraphJob job = new GraphJob();
job.setGraphLoaderClass(SCCVertexReader.class);
job.setVertexClass(SCCVertex.class);
job.setAggregatorClass(SCCAggregator.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
long startTime = System.currentTimeMillis();
job.run();
System.out.println("Job Finished in " + (System.currentTimeMillis() - startTime) / 1000.0 + "
 seconds");
}
```

```
}
```

26.8.6.6 连通分量

两个顶点之间存在路径，称两个顶点为连通的。如果无向图G中任意两个顶点都是连通的，则称G为连通图，否则称为非连通图， 其顶点个数极大的连通子图称为连通分量。本算法计算每个点的连通分量成员，最后输出顶点值中包含最小顶点id的连通分量。将最小顶点id沿着边传播到连通分量的所有顶点。

示例如下：

```
import java.io.IOException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.graph.examples.SSSP.MinLongCombiner;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.WritableRecord;
/**
* Compute the connected component membership of each vertex and output
* each vertex which's value containing the smallest id in the > connected
* component containing that vertex.
*
* Algorithm: propagate the smallest vertex id along the edges to all
* vertices of a connected component.
*
*/
public class ConnectedComponents {
public static class CCVertex extends
Vertex<LongWritable, LongWritable, NullWritable, LongWritable> {
@Override
public void compute(
ComputeContext<LongWritable, LongWritable, NullWritable, LongWritable> context,
 Iterable<LongWritable> msgs) throws IOException {
if (context.getSuperstep() == 0L) {
this.setValue(getId());
context.sendMessageToNeighbors(this, getValue());
return;
}
long minID = Long.MAX_VALUE;
for (LongWritable id : msgs) {
if (id.get() < minID) { minID = id.get();
}
}
if (minID < this.getValue().get()) {
this.setValue(new LongWritable(minID));
context.sendMessageToNeighbors(this, getValue());
} else {
this.voteToHalt();
}
}
/**
```

```
* Output Table Description:
* +-----------------+----------------------------------------+
* Field | Type | Comment |
* +-----------------+----------------------------------------+
* v | bigint | vertex id |
* minID | bigint | smallest id in the connected component |
* +-----------------+----------------------------------------+
*/
@Override
public void cleanup(
WorkerContext<LongWritable, LongWritable, NullWritable, LongWritable> context) throws
 IOException {
context.write(getId(), getValue());
}
}
/**
* Input Table Description:
* +-----------------+----------------------------------------------------+
* Field | Type | Comment |
* +-----------------+----------------------------------------------------+
* v | bigint | vertex id |
* es | string | comma separated target vertex id of outgoing edges |
* +-----------------+----------------------------------------------------+
*
* Example:
* For graph:
* 1 ----- 2
* | |
* 3 ----- 4
* Input table:
* +-----------+
* v | es |
* +-----------+
* | 1 | 2,3 |
* | 2 | 1,4 |
* | 3 | 1,4 |
* | 4 | 2,3 |
* +-----------+
*/
public static class CCVertexReader extends
GraphLoader<LongWritable, LongWritable, NullWritable, LongWritable> {
@Override
public void load(
LongWritable recordNum, WritableRecord record,
MutationContext<LongWritable, LongWritable, NullWritable, LongWritable> context) throws
 IOException {
CCVertex vertex = new CCVertex();
vertex.setId((LongWritable) record.get(0));
String[] edges = record.get(1).toString().split(",");
for (int i = 0; i < edges.length; i++) {
long destID = Long.parseLong(edges[i]);
vertex.addEdge(new LongWritable(destID), NullWritable.get());
}
context.addVertexRequest(vertex);
}
}
public static void main(String[] args) throws IOException {
if (args.length < 2) {
System.out.println("Usage: <input> <output>");
System.exit(-1);
}
GraphJob job = new GraphJob();
```

```
job.setGraphLoaderClass(CCVertexReader.class);
job.setVertexClass(CCVertex.class);
job.setCombinerClass(MinLongCombiner.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
long startTime = System.currentTimeMillis();
job.run();
System.out.println("Job Finished in " + (System.currentTimeMillis() - startTime) / 1000.0 + "
 seconds");
}
}
```

26.8.6.7 拓扑排序

对于有向边（u,v），定义所有满足u<v的顶点序列为拓扑序列，拓扑排序就是求一个有向图的拓扑序列的算法。

算法实现如下：

- 从图中找到一个没有入边的顶点，并输出。
- 从途中删除该点，及其所有出边。
- 重复以上步骤，直到所有点都已输出。

示例如下：

```
import java.io.IOException;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.odps.graph.Combiner;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.BooleanWritable;
import com.aliyun.odps.io.WritableRecord;
public class TopologySort {
private final static Log LOG = LogFactory.getLog(TopologySort.class);
public static class TopologySortVertex extends
Vertex<LongWritable, LongWritable, NullWritable, LongWritable> {
@Override
public void compute(
ComputeContext<LongWritable, LongWritable, NullWritable, LongWritable> context,
 Iterable<LongWritable> messages) throws IOException {
// in superstep 0, each vertex sends message whose value is 1 to its
// neighbors
if (context.getSuperstep() == 0) { if (hasEdges()) {
context.sendMessageToNeighbors(this, new LongWritable(1L));
}
} else if (context.getSuperstep() >= 1) {
// compute each vertex's indegree
```

```
long indegree = getValue().get();
for (LongWritable msg : messages) {
indegree += msg.get();
}
setValue(new LongWritable(indegree));
if (indegree == 0) {
voteToHalt();
if (hasEdges()) {
context.sendMessageToNeighbors(this, new LongWritable(-1L));
}
context.write(new LongWritable(context.getSuperstep()), getId());
LOG.info("vertex: " + getId());
}
context.aggregate(new LongWritable(indegree));
}
}
}
public static class TopologySortVertexReader extends
GraphLoader<LongWritable, LongWritable, NullWritable, LongWritable> {
@Override public void load(
LongWritable recordNum, WritableRecord record,
MutationContext<LongWritable, LongWritable, NullWritable, LongWritable> context) throws
 IOException {
TopologySortVertex vertex = new TopologySortVertex();
vertex.setId((LongWritable) record.get(0));
vertex.setValue(new LongWritable(0));
String[] edges = record.get(1).toString().split(",");
for (int i = 0; i < edges.length; i++) {
long edge = Long.parseLong(edges[i]);
if (edge >= 0) {
vertex.addEdge(new LongWritable(Long.parseLong(edges[i])), NullWritable.get());
}
}
LOG.info(record.toString());
context.addVertexRequest(vertex);
}
}
public static class LongSumCombiner extends
Combiner<LongWritable, LongWritable> {
@Override
public void combine(LongWritable vertexId, LongWritable combinedMessage, LongWritable
 messageToCombine) throws IOException {
combinedMessage.set(combinedMessage.get() + messageToCombine.get());
}
}
public static class TopologySortAggregator extends
Aggregator<BooleanWritable> {
@SuppressWarnings("rawtypes")
@Override
public BooleanWritable createInitialValue(WorkerContext context) throws IOException {
return new BooleanWritable(true);
}
@Override
public void aggregate(BooleanWritable value, Object item) throws IOException {
boolean hasCycle = value.get();
boolean inDegreeNotZero = ((LongWritable) item).get() == 0 ? false : true;
value.set(hasCycle && inDegreeNotZero);
}
@Override
public void merge(BooleanWritable value, BooleanWritable partial) throws IOException {
value.set(value.get() && partial.get());
}
```

```
@SuppressWarnings("rawtypes")
@Override
public boolean terminate(WorkerContext context, BooleanWritable value) throws IOException {
if (context.getSuperstep() == 0) {
// since the initial aggregator value is true, and in superstep we don't
// do aggregate
return false;
}
return value.get();
}
}
public static void main(String[] args) throws IOException { if (args.length != 2) {
System.out.println("Usage : <inputTable> <outputTable>");
System.exit(-1);
}
// 输入表形式为
// 0 1，2
// 1 3
// 2 3
// 3 -1
// 第一列为vertexid，第二列为该点边的destination vertexid，若值为-1，表示该点无出边
// 输出表形式为
// 0 0
// 1 1
// 1 2
// 2 3
// 第一列为supstep值，隐含了拓扑顺序，第二列为vertexid
// TopologySortAggregator用来判断图中是否有环
// 若输入的图有环，则当图中active的点入度都不为0时，迭代结束
// 用户可以通过输入表和输出表的记录数来判断一个有向图是否有环
GraphJob job = new GraphJob();
job.setGraphLoaderClass(TopologySortVertexReader.class);
job.setVertexClass(TopologySortVertex.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
job.setCombinerClass(LongSumCombiner.class);
job.setAggregatorClass(TopologySortAggregator.class);
long startTime = System.currentTimeMillis(); job.run();
System.out.println("Job Finished in " + (System.currentTimeMillis() - startTime) / 1000.0 + "
 seconds");
}
}
```

26.8.6.8 线性回归

在统计学中，线性回归是用来确定两种或两种以上变量间的相互依赖关系的统计分析方法，与分类算法处理离散预测不同， 回归算法可对连续值类型进行预测。线性回归算法定义损失函数为样本集的最小平方误差之和，通过最小化损失函数求解权重矢量。

常用的解法是梯度下降法，算法实现如下：

- 初始化权重矢量，给定下降速率以及迭代次数（或者迭代收敛条件）。
- 对每个样本，计算最小平方误差。
- 对最小平方误差求和，根据下降速率更新权重。

- 重复迭代直到收敛。

示例如下：

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.Tuple;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.io.WritableRecord;
/**
* LineRegression input: y,x1,x2,x3,.....
*
* @author shiwan.ch
* @update jiasheng.tjs running parameters are like: tjs_lr_in > tjs_lr_out 1500 2
* 0.07
*/
public class LinearRegression {
public static class GradientWritable implements Writable {
Tuple lastTheta;
Tuple currentTheta;
Tuple tmpGradient;
LongWritable count;
DoubleWritable lost;
@Override
public void readFields(DataInput in) throws IOException {
lastTheta = new Tuple();
lastTheta.readFields(in);
currentTheta = new Tuple();
currentTheta.readFields(in);
tmpGradient = new Tuple();
tmpGradient.readFields(in);
count = new LongWritable();
count.readFields(in);
/* update 1: add a variable to store lost at every iteration */
lost = new DoubleWritable();
lost.readFields(in);
}
@Override
public void write(DataOutput out) throws IOException {
lastTheta.write(out);
currentTheta.write(out);
tmpGradient.write(out);
count.write(out);
lost.write(out);
}
}
public static class LinearRegressionVertex extends
Vertex<LongWritable, Tuple, NullWritable, NullWritable> {
@Override
```

```
public void compute(
ComputeContext<LongWritable, Tuple, NullWritable, NullWritable> context,
 Iterable<NullWritable> messages) throws IOException {
context.aggregate(getValue());
}
}
public static class LinearRegressionVertexReader extends
GraphLoader<LongWritable, Tuple, NullWritable, NullWritable> {
@Override
public void load(LongWritable recordNum, WritableRecord record,
 MutationContext<LongWritable, Tuple, NullWritable, NullWritable> context)
throws IOException {
LinearRegressionVertex vertex = new LinearRegressionVertex();
vertex.setId(recordNum);
vertex.setValue(new Tuple(record.getAll())); context.addVertexRequest(vertex);
}
}
public static class LinearRegressionAggregator extends
Aggregator<GradientWritable> {
@SuppressWarnings("rawtypes")
@Override
public GradientWritable createInitialValue(WorkerContext context) throws IOException {
if (context.getSuperstep() == 0) {
/* set initial value, all 0 */
GradientWritable grad = new GradientWritable();
grad.lastTheta = new Tuple();
grad.currentTheta = new Tuple();
grad.tmpGradient = new Tuple();
grad.count = new LongWritable(1);
grad.lost = new DoubleWritable(0.0);
int n = (int) Long.parseLong(context.getConfiguration().get("Dimension"));
for (int i = 0; i < n; i++) { grad.lastTheta.append(new DoubleWritable(0));
grad.currentTheta.append(new DoubleWritable(0));
grad.tmpGradient.append(new DoubleWritable(0));
}
return grad;
} else
return (GradientWritable) context.getLastAggregatedValue(0);
}
public static double vecMul(Tuple value, Tuple theta) {
/* perform this partial computing: y(i)−hθ(x(i)) for each sample */
/* value denote a piece of sample and value(0) is y */
double sum = 0.0;
for (int j = 1; j < value.size(); j++)
sum += Double.parseDouble(value.get(j).toString()) *
 Double.parseDouble(theta.get(j).toString());
Double tmp = Double.parseDouble(theta.get(0).toString()) + sum -
Double.parseDouble(value.get(0).toString());
return tmp;
}
@Override
public void aggregate(GradientWritable gradient, Object value) throws IOException {
/*
* perform on each vertex--each sample i：set theta(j) for each sample > i
* for each dimension
*/
double tmpVar = vecMul((Tuple) value, gradient.currentTheta);
/*
* update 2:local worker aggregate(), perform like merge() below. This
* means the variable gradient denotes the previous aggregated value
*/
```

```
gradient.tmpGradient.set(0, new DoubleWritable( ((DoubleWritable)
 gradient.tmpGradient.get(0)).get() + tmpVar));
gradient.lost.set(Math.pow(tmpVar, 2));
/*
* calculate (y(i)−hθ(x(i)))x(i)(j) for each sample i for each
* dimension j
*/
for (int j = 1; j < gradient.tmpGradient.size(); j++) gradient.tmpGradient.set(j, new DoubleWritable(
((DoubleWritable) gradient.tmpGradient.get(j)).get() + tmpVar * Double.parseDouble(((Tuple)
 value).get(j).toString())));
}
@Override
public void merge(GradientWritable gradient, GradientWritable partial) throws IOException {
/* perform SumAll on each dimension for all samples. */
Tuple master = (Tuple) gradient.tmpGradient;
Tuple part = (Tuple) partial.tmpGradient;
for (int j = 0; j < gradient.tmpGradient.size(); j++) {
DoubleWritable s = (DoubleWritable) master.get(j);
s.set(s.get() + ((DoubleWritable) part.get(j)).get());
}
gradient.lost.set(gradient.lost.get() + partial.lost.get());
}
@SuppressWarnings("rawtypes")
@Override
public boolean terminate(WorkerContext context, GradientWritable gradient) throws IOException
 {
/*
* 1. calculate new theta 2. judge the diff between last step and this
* step, if smaller than the threshold, stop iteration
*/
gradient.lost = new DoubleWritable(gradient.lost.get()/ (2 * context.getTotalNumVertices()));
/*
* we can calculate lost in order to make sure the algorithm is running > on
* the right direction (for debug)
*/
System.out.println(gradient.count + " lost:" + gradient.lost);
Tuple tmpGradient = gradient.tmpGradient;
System.out.println("tmpGra" + tmpGradient);
Tuple lastTheta = gradient.lastTheta;
Tuple tmpCurrentTheta = new Tuple(gradient.currentTheta.size());
System.out.println(gradient.count + " terminate_start_last:" + lastTheta);
double alpha = 0.07; // learning rate
// alpha =
// Double.parseDouble(context.getConfiguration().get("Alpha"));
/* perform theta(j) = theta(j)-alpha*tmpGradient */
long M = context.getTotalNumVertices();
/*
* update 3: add (/M) on the code. The original code forget this step
*/
for (int j = 0; j < lastTheta.size(); j++) { tmpCurrentTheta
.set( j,
new DoubleWritable(Double.parseDouble(lastTheta.get(j)
.toString()) – alpha / M * Double.parseDouble(tmpGradient.get(j).toString())));
}
System.out.println(gradient.count + " terminate_start_current:" + tmpCurrentTheta);
// judge if convergence is happening.
double diff = 0.00d;
for (int j = 0; j < gradient.currentTheta.size(); j++)
diff += Math.pow(((DoubleWritable) tmpCurrentTheta.get(j)).get() - ((DoubleWritable)
 lastTheta.get(j)).get(), 2);
if (
/*
```

```
* Math.sqrt(diff) < 0.00000000005d ||
*/
Long.parseLong(context.getConfiguration().get("Max_Iter_Num")) == gradient.count
.get()) { context.write(gradient.currentTheta.toArray());
return true;
}
gradient.lastTheta = tmpCurrentTheta;
gradient.currentTheta = tmpCurrentTheta;
gradient.count.set(gradient.count.get() + 1);
int n = (int) Long.parseLong(context.getConfiguration().get("Dimension"));
/*
* update 4: Important!!! Remember this step. Graph won't reset the
* initial value for global variables at the beginning of each iteration
*/
for (int i = 0; i < n; i++) {
gradient.tmpGradient.set(i, new DoubleWritable(0));
}
return false;
}
}
public static void main(String[] args) throws IOException { GraphJob job = new GraphJob();
job.setGraphLoaderClass(LinearRegressionVertexReader.class);          job.setRuntimePartitioni
ng(false);
job.setNumWorkers(3);
job.setVertexClass(LinearRegressionVertex.class);
job.setAggregatorClass(LinearRegressionAggregator.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
job.setMaxIteration(Integer.parseInt(args[2])); // Numbers of Iteration
job.setInt("Max_Iter_Num", Integer.parseInt(args[2]));
job.setInt("Dimension", Integer.parseInt(args[3])); // Dimension
job.setFloat("Alpha", Float.parseFloat(args[4])); // Learning rate
long start = System.currentTimeMillis(); job.run();
System.out.println("Job Finished in " + (System.currentTimeMillis() - start) / 1000.0 + " seconds");
}
}
```

26.8.6.9 三角形计数

计算通过每个顶点的三角形个数。算法实现如下：

- 每个顶点将其ID发送给所有出边邻居。
- 存储入边和出边邻居并发送给出边邻居。
- 对每条边计算其终点的交集数量，并求和，结果输出到表。
- 将表中的输出结果求和并除以三，即得到三角形个数。

示例如下：

```
import java.io.IOException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.Edge;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
```

```
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.Tuple;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.io.WritableRecord;
/**
* Compute the number of triangles passing through each vertex.
*
* The algorithm can be computed in three supersteps:
* I. Each vertex sends a message with its ID to all its outgoing
* neighbors.
* II. The incoming neighbors and outgoing neighbors are stored and
* send to outgoing neighbors.
* III. For each edge compute the intersection of the sets at destination
* vertex and sum them, then output to table.
*
* The triangle count is the sum of output table and divide by three > since
* each triangle is counted three times.
*
**/
public class TriangleCount {
public static class TCVertex extends
Vertex<LongWritable, Tuple, NullWritable, Tuple> {
@Override
public void setup(
WorkerContext<LongWritable, Tuple, NullWritable, Tuple> context) throws IOException {
// collect the outgoing neighbors
Tuple t = new Tuple();
if (this.hasEdges()) {
for (Edge<LongWritable, NullWritable> edge : this.getEdges()) {
t.append(edge.getDestVertexId());
}
}
this.setValue(t);
}
@Override
public void compute(
ComputeContext<LongWritable, Tuple, NullWritable, Tuple> context, Iterable<Tuple> msgs)
 throws IOException {
if (context.getSuperstep() == 0L) {
// sends a message with its ID to all its outgoing neighbors
Tuple t = new Tuple(); t.append(getId());
context.sendMessageToNeighbors(this, t);
} else if (context.getSuperstep() == 1L) {
// store the incoming neighbors
for (Tuple msg : msgs) {
for (Writable item : msg.getAll()) {
if (!this.getValue().getAll().contains((LongWritable)item)) {
this.getValue().append((LongWritable)item);
}
}
}
// send both incoming and outgoing neighbors to all outgoing neighbors
context.sendMessageToNeighbors(this, getValue());
} else if (context.getSuperstep() == 2L) {
// count the sum of intersection at each edge
long count = 0;
for (Tuple msg : msgs) {
for (Writable id : msg.getAll()) {
if (getValue().getAll().contains(id)) { count ++;
}
```

```
}
}
// output to table
context.write(getId(), new LongWritable(count));
this.voteToHalt();
}
}
}
public static class TCVertexReader extends
GraphLoader<LongWritable, Tuple, NullWritable, Tuple> {
@Override public void load(
LongWritable recordNum, WritableRecord record,
MutationContext<LongWritable, Tuple, NullWritable, Tuple> context) throws IOException {
TCVertex vertex = new TCVertex();
vertex.setId((LongWritable) record.get(0));
String[] edges = record.get(1).toString().split(",");
for (int i = 0; i < edges.length; i++) { try {
long destID = Long.parseLong(edges[i]);
vertex.addEdge(new LongWritable(destID), NullWritable.get());
} catch(NumberFormatException nfe) { System.err.println("Ignore " + nfe);
}
}
context.addVertexRequest(vertex);
}
}
public static void main(String[] args) throws IOException { if (args.length < 2) {
System.out.println("Usage: <input> <output>"); System.exit(-1);
}
GraphJob job = new GraphJob();
job.setGraphLoaderClass(TCVertexReader.class);
job.setVertexClass(TCVertex.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
long startTime = System.currentTimeMillis();
job.run();
System.out.println("Job Finished in " + (System.currentTimeMillis() - startTime) / 1000.0 + "
 seconds");
}
}
```

26.8.6.10 GraphLoader

示例如下：

```
import java.io.IOException;
import com.aliyun.odps.conf.Configuration;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.VertexResolver;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.VertexChanges;
import com.aliyun.odps.graph.Edge;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.WritableComparable;
import com.aliyun.odps.io.WritableRecord;
/**
```

```
*              本示例是用于展示，对于不同类型的数据类型，如何编写图作业程序载入数据。主要展示GraphLoader和
* VertexResolver的配合完成图的构建。
*
* MaxCompute Graph的作业输入都为 MaxCompute 的Table，假设作业输入有两张表，一张存储点的信息，一张存储边的信息。
* 存储点信息的表的格式，如：
* +-----------------------+
* VertexID | VertexValue |
* +-----------------------+
* | id0| 9|
* +-----------------------+
* | id1| 7|
* +-----------------------+
* | id2| 8|
* +-----------------------+
*
* 存储边信息的表的格式，如
* +---------------------------------+
* VertexID | DestVertexID| EdgeValue|
* +---------------------------------+
* | id0| id1| 1|
* +---------------------------------+
* | id0| id2| 2|
* +---------------------------------+
* | id2| id1| 3|
* +---------------------------------+
*
*   结合两张表的数据，表示id0有两条出边，分别指向id1和id2；id2有一条出边，指向id1；id1没有出边。
*
*对于此种类型的数据，在GraphLoader::load(LongWritable, Record, MutationContext)，可以使用 > MutationContext#addVertexRequest(Vertex)向图中请求添加点，使用link

后，在link VertexResolver#resolve(WritableComparable, Vertex, VertexChanges, boolean)中，将load 方法中添加的点和边，合并到一个Vertex对象中，作为返回值，添加到最后参与计算的图中。
*
**/
public class VertexInputFormat {
private final static String EDGE_TABLE = "edge.table";
/**
* 将Record解释为Vertex和Edge，每个Record根据其来源，表示一个Vertex或者一条Edge。
* <p>
*类似于com.aliyun.odps.mapreduce.Mapper#map ，输入Record，生成键值对，此处的键是Vertex的ID，值是Vertex或Edge，通过上下文Context写出，这些键值对会在LoadingVertexResolver出根据Vertex的ID汇总。
*
*   注意：此处添加的点或边只是根据Record内容发出的请求，并不是最后参与计算的点或边，只有在随后的VertexResolver中添加的点或边才参与计算。
**/
public static class VertexInputLoader extends
GraphLoader<LongWritable, LongWritable, LongWritable, LongWritable> {
private boolean isEdgeData;
/**
* 配置VertexInputLoader。
*
* @param conf
* 作业的配置参数，在main中使用GraphJob配置的，或者在console中set的
* @param workerId
```

```
* 当前工作的worker的序号，从0开始，可以用于构造唯一的vertex id
* @param inputTableInfo
* 当前worker载入的输入表信息，可以用于确定当前输入是哪种类型的数据，即Record的格式
**/
@Override
public void setup(Configuration conf, int workerId, TableInfo inputTableInfo) {
isEdgeData = conf.get(EDGE_TABLE).equals(inputTableInfo.getTableName());
}
/**
* 根据Record中的内容，解析为对应的边，并请求添加到图中。
*
* @param recordNum
* 记录序列号，从1开始，每个worker上单独计数
* @param record
* 输入表中的记录，三列，分别表示初点、终点、边的权重
* @param context
* 上下文，请求将解释后的边添加到图中
**/
@Override public void load(
LongWritable recordNum, WritableRecord record,
MutationContext<LongWritable, LongWritable, LongWritable, LongWritable> context) throws
 IOException {
if (isEdgeData) {
/**
* 数据来源于存储边信息的表。
*
* 1、第一列表示初始点的ID
**/
LongWritable sourceVertexID = (LongWritable) record.get(0);
/**
* 2、第二列表示终点的ID
**/
LongWritable destinationVertexID = (LongWritable) record.get(1);
/**
* 3、地三列表示边的权重
**/
LongWritable edgeValue = (LongWritable) record.get(2);
/**
* 4、创建边，由终点ID和边的权重组成
**/
Edge<LongWritable, LongWritable> edge = new Edge<LongWritable,
 LongWritable>( destinationVertexID, edgeValue);
/**
* 5、请求给初始点添加边
**/
context.addEdgeRequest(sourceVertexID, edge);
/**
* 6、如果每条Record表示双向边，重复4与5 Edge<LongWritable, > LongWritable> edge2 = new
* Edge<LongWritable, LongWritable>( sourceVertexID, edgeValue);
* context.addEdgeRequest(destinationVertexID, edge2);
**/
} else {
/**
* 数据来源于存储点信息的表。
*
* 1、第一列表示点的ID
**/
LongWritable vertexID = (LongWritable) record.get(0);
/**
* 2、第二列表示点的值
**/
```

```
LongWritable vertexValue = (LongWritable) record.get(1);
/**
* 3、创建点，由点的ID和点的值组成
**/
MyVertex vertex = new MyVertex();
/**
* 4、初始化点
**/
vertex.setId(vertexID); vertex.setValue(vertexValue);
/**
* 5、请求添加点
**/
context.addVertexRequest(vertex);
}
}
}
/**
* 汇总GraphLoader::load(LongWritable, Record, > MutationContext)生成的键值对，类似于
* com.aliyun.odps.mapreduce.Reducer中的reduce。对于唯一的Vertex > ID，所有关于这个ID上
* 添加\删除、点\边的行为都会放在VertexChanges中。
*
*              注意：此处并不只针对load方法中添加的有冲突的点或边才调用（冲突是指添加多个相
同Vertex对象，添加重复边等），
* 所有在load方法中请求生成的ID都会在此处被调用。
**/
public static class LoadingResolver extends
VertexResolver<LongWritable, LongWritable, LongWritable, LongWritable> {
/**
* 处理关于一个ID的添加或删除、点或边的请求。
*
* VertexChanges有四个接口，分别与MutationContext的四个接口对应：
* VertexChanges::getAddedVertexList()与
* MutationContext::addVertexRequest(Vertex)对应，
* 在load方法中，请求添加的ID相同的Vertex对象，会被汇总在返回的List中
* VertexChanges::getAddedEdgeList()与
* MutationContext::addEdgeRequest(WritableComparable, Edge)
* 对应，请求添加的初始点ID相同的Edge对象，会被汇总在返回的List中
* VertexChanges::getRemovedVertexCount()与
* MutationContext::removeVertexRequest(WritableComparable)
* 对应，请求删除的ID相同的Vertex，汇总的请求删除的次数作为返回值
* VertexChanges#getRemovedEdgeList()与
* MutationContext#removeEdgeRequest(WritableComparable, WritableComparable)
* 对应，请求删除的初始点ID相同的Edge对象，会被汇总在返回的List中
*
*       用户通过处理关于这个ID的变化，通过返回值声明此ID是否参与计算，如果返回的Vertex不
为null，
* 则此ID会参与随后的计算，如果返回null，则不会参与计算。
*
* @param vertexId
* 请求添加的点的ID，或请求添加的边的初点ID
* @param vertex
* 已存在的Vertex对象，数据载入阶段，始终为null
* @param vertexChanges
* 此ID上的请求添加\删除、点\边的集合
* @param hasMessages
* 此ID是否有输入消息，数据载入阶段，始终为false
**/
@Override
public Vertex<LongWritable, LongWritable, LongWritable, LongWritable> resolve( LongWritable
 vertexId,
```

```
Vertex<LongWritable, LongWritable, LongWritable, LongWritable> vertex,
 VertexChanges<LongWritable, LongWritable, LongWritable, LongWritable> vertexChanges,
 boolean hasMessages) throws IOException {
/**
* 1、获取Vertex对象，作为参与计算的点。
**/
MyVertex computeVertex = null;
if (vertexChanges.getAddedVertexList() == null
|| vertexChanges.getAddedVertexList().isEmpty()) { computeVertex = new MyVertex();
 computeVertex.setId(vertexId);
} else {
/**
* 此处假设存储点信息的表中，每个Record表示唯一的点。
**/
computeVertex = (MyVertex) vertexChanges.getAddedVertexList().get(0);
}
/**
*  2、将请求给此点添加的边，添加到Vertex对象中，如果数据有重复的可能，根据算法需要决定是否去重。
**/
if (vertexChanges.getAddedEdgeList() != null) {
for (Edge<LongWritable, LongWritable> edge : vertexChanges.getAddedEdgeList())
 { computeVertex.addEdge(edge.getDestVertexId(), edge.getValue());
}
}
/**
* 3、将Vertex对象返回，添加到最终的图中参与计算。
**/
return computeVertex;
}
}
/**
* 确定参与计算的Vertex的行为。
*
**/
public static class MyVertex extends
Vertex<LongWritable, LongWritable, LongWritable, LongWritable> {
/**
* 将vertex的边，按照输入表的格式再写到结果表。输入表与输出表的格式和数据都相同。
*
* @param context
* 运行时上下文
* @param messages
* 输入消息
**/
@Override
public void compute(
ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable> context,
 Iterable<LongWritable> messages) throws IOException {
/**
* 将点的ID和值，写到存储点的结果表
**/
context.write("vertex", getId(), getValue());
/**
* 将点的边，写到存储边的结果表
**/
if (hasEdges()) {
for (Edge<LongWritable, LongWritable> edge : getEdges()) { context.write("edge", getId(),
 edge.getDestVertexId(),
edge.getValue());
}
```

```
}
/**
* 只迭代一轮
**/
voteToHalt();
}
}
/**
* @param args
* @throws IOException
*/
public static void main(String[] args) throws IOException {
if (args.length < 4) { throw new IOException(
"Usage: VertexInputFormat <vertex input> <edge input> <vertex output> <edge output>");
}
/**
* GraphJob用于对Graph作业进行配置
*/
GraphJob job = new GraphJob();
/**
* 1、指定输入的图数据，并指定存储边数据所在的表。
*/
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addInput(TableInfo.builder().tableName(args[1]).build());
job.set(EDGE_TABLE, args[1]);
/**
*  2、指定载入数据的方式，将Record解释为Edge，此处类似于map，生成的  >
 key为vertex的ID，value为edge。
*/
job.setGraphLoaderClass(VertexInputLoader.class);
/**
* 3、指定载入数据阶段，生成参与计算的vertex。此处类似于reduce，将map 生成的edge合并成
一个vertex。
*/
job.setLoadingVertexResolverClass(LoadingResolver.class);
/**
* 4、指定参与计算的vertex的行为。每轮迭代执行vertex.compute方法。
*/
job.setVertexClass(MyVertex.class);
/**
* 5、指定图作业的输出表，将计算生成的结果写到结果表中。
*/
job.addOutput(TableInfo.builder().tableName(args[2]).label("vertex").build());
job.addOutput(TableInfo.builder().tableName(args[3]).label("edge").build());
/**
* 6、提交作业执行。
*/
job.run();
}
}
```

26.9 Java SDK使用指南

26.9.1 SDK介绍

MaxCompute提供了一整套Java SDK供用户开发，提供了丰富灵活的API接口用于支持用户基于MaxCompute平台进行开发。本章节将会简要介绍较为常用的MaxCompute核心接口。

包名	描述
odps-sdk-core	MaxCompute基础功能。 封装了基础的MaxCompute概念及其编程接口，包括MaxCompute、project、table等，tunnel相关的功能也在此包。
odps-sdk-core-internal	MaxCompute扩展功能。 封装了一些不常用的MaxCompute概念和操作，例如Event、XFlow等。
odps-sdk-commons	MaxCompute基础设施。 包含TableSchema、Column、Record、OdpsType等基础设施和一些util的封装。
odps-sdk-udf	MaxCompute UDF编程接口。
odps-sdk-mapred	MaxCompute MapReduce作业编程接口。

26.9.2 AliyunAccount

阿里云认证账号。

输入参数为accessId及accessKey，是阿里云用户的身份标识和认证密钥。此类用来初始化MaxCompute。

26.9.3 Odps

MaxCompute SDK的入口，用户通过此类来获取项目空间下的所有对象集合，包括：Projects ，Tables ，Resources，Functions，Instances。 用户可以通过传入AliyunAccount实例来构造MaxCompute对象。

示例如下：

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
odps.setDefaultProject("my_project");
for (Table t : odps.tables()) {
....
}
```

说明: MaxCompute提供了两个服务地址供用户选择。详细说明请参考：*Tunnel SDK简介*。

26.9.4 Projects

MaxCompute中所有项目空间的集合。集合中的元素为Project 。

示例如下：

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps  odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Project p = odps.projects().get("my_exists");
p.reload();
Map<String, String> properties = prj.getProperties();
...
```



说明: MaxCompute提供了两个服务地址供用户选择。详细说明请参考：*Tunnel SDK简介*。

26.9.5 Project

对项目空间信息的描述，可以通过Projects获取相应的项目空间。

26.9.6 SQLTask

用于运行、处理SQL任务的接口。可以通过run接口直接运行SQL。run接口返回Instance实例，通过Instance获取SQL的运行状态及运行结果。

示例如下：

```
Account account = new AliyunAccount("my_access_id", "my_access_key"); Odps odps = new
 Odps(account);
String  odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Instance instance = SQLTask.run(odps, "my_project", "select ...");
String id = instance.getId();
instance.waitforsuccess();
Set<String> taskNames = instance.getTaskNames();
for (String name
taskNames) {
TaskSummary summary = instance.getTaskSummary(name);
String s = summary.getSummaryText();
}
Map<String, String> results = instance.getTaskResults();
Map<String, TaskStatus> taskStatus = instance.getTaskStatus();
for (Entry<String, TaskStatus> status : taskStatus.entrySet()) {
String result = results.get(status.getKey());
}
```



说明:

- 如果用户想创建表，需要通过SQLTask接口，而不是Table接口来完成。用户需要将创建表的语句传入SQLTask。

- MaxCompute提供了两个服务地址供用户选择。详细说明请参考：*Tunnel SDK简介*。

26.9.7 Instances

MaxCompute中所有实例（Instance）的集合。集合中的元素为Instance 。

示例如下：

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
odps.setDefaultProject("my_project");
for (Instance i : odps.instances ()) {
....
}
```



说明: MaxCompute提供了两个服务地址供用户选择。详细说明请参考：*Tunnel SDK简介*。

26.9.8 Instance

对实例信息的描述，可以通过Instances获取相应的实例。

示例如下：

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps (account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Instance ins = odps.instances().get("instance id");
Date startTime = instance.getStartTime();
Date endTime = instance.getEndTime();
...
Status instanceStatus = instance.getStatus();
String instanceStatusStr = null;
if (instanceStatus == Status.TERMINATED) {
instanceStatusStr = TaskStatus.Status.SUCCESS.toString();
Map<String, TaskStatus> taskStatus = instance.getTaskStatus();
for (Entry<String, TaskStatus> status : taskStatus.entrySet()) {
if (status.getValue().getStatus() != TaskStatus.Status.SUCCESS) {
instanceStatusStr = status.getValue().getStatus().toString();
break;
}
}
} else {
instanceStatusStr = instanceStatus.toString();
}
...
TaskSummary summary = instance.getTaskSummary("instance name");
```

```
String s = summary.getSummaryText();
```



说明: MaxCompute提供了两个服务地址供用户选择。详细说明请参考：*Tunnel SDK简介*。

26.9.9 Tables

MaxCompute中所有表的集合。集合中的元素为Table 。

示例如下：

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
odps.setDefaultProject("my_project");
for (Table t : odps.tables()) {
....
}
```



说明: MaxCompute提供了两个服务地址供用户选择。详细说明请参考：*Tunnel SDK简介*。

26.9.10 Table

对表信息的描述，可以通过Tables获取相应的表。

示例如下：

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Table t = odps.tables().get("table name");
t.reload();
Partition part = t.getPartition(new PartitionSpec(tableSpec[1]));
part.reload();
...
```



说明: MaxCompute提供了两个服务地址供用户选择。详细说明请参考：*Tunnel SDK简介*。

26.9.11 Resources

MaxCompute中所有资源的集合。集合中的元素为Resource。

示例如下：

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
```

```
odps.setEndpoint(odpsUrl);
odps.setDefaultProject("my_project");
for (Resource r : odps.resources()) {
....
}
```



说明: MaxCompute提供了两个服务地址供用户选择。详细说明请参考：*Tunnel SDK*简介。

26.9.12 Resource

对资源信息的描述，可以通过Resources获取相应的资源。

示例如下：

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Resource r = odps.resources().get("resource name");
r.reload();
if (r.getType() == Resource.Type.TABLE) { TableResource tr = new TableResource(r);
String tableSource = tr.getSourceTable().getProject() + "." + tr.getSourceTable().getName();
if (tr.getSourceTablePartition() != null) {
tableSource += " partition(" + tr.getSourceTablePartition().toString() + ")";
}
....
}
```



说明: MaxCompute提供了两个服务地址供用户选择。详细说明请参考：*Tunnel SDK*简介。

附录：

创建文件资源的示例

```
String projectName = "my_porject";
String source = "my_local_file.txt";
File file = new File(source);
InputStream is = new FileInputStream(file);
FileResource resource = new FileResource();
String name = file.getName();
resource.setName(name);
odps.resources().create(projectName, resource, is);
```

创建表资源的示例

```
TableResource resource = new TableResource(tableName, tablePrj, partitionSpec);
//resource.setName(INVALID_USER_TABLE);
resource.setName("table_resource_name");
```

```
odps.resources().update(projectName, resource);
```

26.9.13 Functions

MaxCompute中所有函数的集合。集合中的元素为Function。

示例如下：

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
odps.setDefaultProject("my_project");
for (Function f : odps.functions()) {
....
}
```



说明: MaxCompute提供了两个服务地址供用户选择。详细说明请参考：*Tunnel SDK*简介。

26.9.14 Function

对函数信息的描述，可以通过Functions获取相应的函数。

示例如下：

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Function f = odps.functions().get("function name");
List<Resource> resources = f.getResources();
```



说明: MaxCompute提供了两个服务地址供用户选择。详细说明请参考：*Tunnel SDK*简介。

附录：

创建函数的示例

```
String resources = "xxx:xxx";
String classType = "com.aliyun.odps.mapred.open.example.WordCount";
ArrayList<String> resourceList = new ArrayList<String>();
for (String r : resources.split(":")) {
resourceList.add(r);
}
Function func = new Function();
func.setName(name);
func.setClassType(classType);
func.setResources(resourceList);
```

```
odps.functions().create(projectName, func);
```

26.10 Java沙箱限制介绍

MaxCompute MapReduce及UDF程序在分布式环境中运行时受到Java沙箱的限制（MapReduce作业的主程序则不受此限制），具体限制如下：

- 不允许直接访问本地文件，只能通过MaxCompute MapReduce/Graph提供的接口间接访问：读取resources选项指定的资源，包括文件、jar包和资源表等；通过System.out和System.err输出日志信息，可以通过MaxCompute客户端的Log命令查看日志信息。
- 不允许直接访问分布式文件系统，只能通过MaxCompute MapReduce/Graph访问到表的记录。
- 不允许JNI调用限制。
- 不允许创建Java线程，不允许运行子进程执行Linux命令。
- 不允许访问网络，包括获取本地IP地址等都会被禁止。
- Java反射限制，suppressAccessChecks权限被禁止，无法setAccessible某个private的属性或方法，以达到读取private属性或调用private方法的目的。

  具体的说，在用户代码中直接使用下面的方法会报access denied异常。

  访问本地文件相关方法。

  java.io.File：

```
public boolean delete()
public void deleteOnExit()
public boolean exists()
public boolean canRead()
public boolean isFile()
public boolean isDirectory()
public boolean isHidden()
public long lastModified()
public long length()
public String[] list()
public String[] list(FilenameFilter filter)
public File[] listFiles()
public File[] listFiles(FilenameFilter filter)
public File[] listFiles(FileFilter filter)
public boolean canWrite()
public boolean createNewFile()
public static File createTempFile(String prefix, String suffix)
public static File createTempFile(String prefix, String suffix,File directory)
public boolean mkdir()
public boolean mkdirs()
public boolean renameTo(File dest)
public boolean setLastModified(long time)
public boolean setReadOnly()
java.io.RandomAccessFile:
RandomAccessFile(String name, String mode)
RandomAccessFile(File file, String mode)
java.io.FileInputStream:
```

```
FileInputStream(FileDescriptor fdObj)
FileInputStream(String name) FileInputStream(File file)
java.io.FileOutputStream:
FileOutputStream(FileDescriptor fdObj)
FileOutputStream(File file)
FileOutputStream(String name)
FileOutputStream(String name, boolean append)
java.lang.Class:
public ProtectionDomain getProtectionDomain()
java.lang.ClassLoader:
ClassLoader()
ClassLoader(ClassLoader parent)
java.lang.Runtime:
public Process exec(String command)
public Process exec(String command, String envp[])
public Process exec(String cmdarray[])
public Process exec(String cmdarray[], String envp[])
public void exit(int status)
public static void runFinalizersOnExit(boolean value)
public void addShutdownHook(Thread hook)
public boolean removeShutdownHook(Thread hook)
public void load(String lib)
public void loadLibrary(String lib)
java.lang.System:
public static void exit(int status)
public static void runFinalizersOnExit(boolean value)
public static void load(String filename)
public static void loadLibrary( String libname)
public static Properties getProperties()
public static void setProperties(Properties props)
public static String getProperty(String key) // 只允许部分key可以访问
public static String getProperty(String key, String def) // 只允许部分key可以访问
public static String setProperty(String key, String value)
public static void setIn(InputStream in)
public static void setOut(PrintStream out)
public static void setErr(PrintStream err)
public static synchronized void setSecurityManager(SecurityManager s)
```

System.getProperty允许的key列表如下：

```
java.version
java.vendor
java.vendor.url
java.class.version
os.name os.version
os.arch
file.separator
path.separator
line.separator
java.specification.version
java.specification.vendor
java.specification.name
java.vm.specification.version
java.vm.specification.vendor
java.vm.specification.name
java.vm.version
java.vm.vendor
java.vm.name
file.encoding
user.timezone
java.lang.Thread:
```

```
Thread()
Thread(Runnable target)
Thread(String name)
Thread(Runnable target, String name)
Thread(ThreadGroup group, ...)
public final void checkAccess()
public void interrupt()
public final void suspend()
public final void resume()
public final void setPriority (int newPriority)
public final void setName(String name)
public final void setDaemon(boolean on)
public final void stop()
public final synchronized void stop(Throwable obj)
public static int enumerate(Thread tarray[])
public void setContextClassLoader(ClassLoader cl)
java.lang.ThreadGroup:
ThreadGroup(String name)
ThreadGroup(ThreadGroup parent, String name)
public final void checkAccess()
public int enumerate(Thread list[])
public int enumerate(Thread list[], boolean recurse)
public int enumerate(ThreadGroup list[])
public int enumerate(ThreadGroup list[], boolean recurse)
public final ThreadGroup getParent()
public final void setDaemon(boolean daemon)
public final void setMaxPriority(int pri)
public final void suspend()
public final void resume()
public final void destroy()
public final void interrupt()
public final void stop()
java.lang.reflect.AccessibleObject:
public static void setAccessible(...)
public void setAccessible(...)
java.net.InetAddress:
public String getHostName()
public static InetAddress[] getAllByName(String host)
public static InetAddress getLocalHost()
java.net.DatagramSocket:
public InetAddress getLocalAddress()
java.net.Socket:
Socket(...)
java.net.ServerSocket:
ServerSocket(...)
public Socket accept()
protected final void implAccept(Socket s)
public static synchronized void setSocketFactory(...)
public static synchronized void setSocketImplFactory(...)
java.net.DatagramSocket:
DatagramSocket(...)
public synchronized void receive(DatagramPacket p)
java.net.MulticastSocket:
MulticastSocket(...)
java.net.URL:
URL(...)
public static synchronized void setURLStreamHandlerFactory(...)
java.net.URLConnection
public static synchronized void setContentHandlerFactory(...)
public static void setFileNameMap(FileNameMap map)
java.net.HttpURLConnection:
public static void setFollowRedirects(boolean set)
```

```
java.net.URLClassLoader
URLClassLoader(...)
java.security.AccessControlContext:
public AccessControlContext(AccessControlContext acc, DomainCombiner combiner)
public DomainCombiner getDomainCombiner()
```

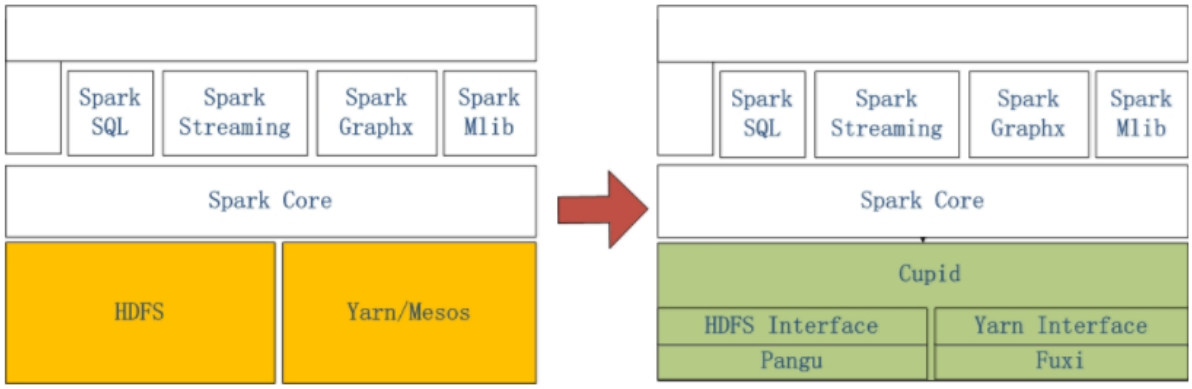
26.11 Spark使用指南

26.11.1 概要

Spark on MaxCompute是阿里云开发的大数据分析引擎，为阿里内部用户、政企客户提供大数据处理能力。**Spark on MaxCompute**为用户提供原生Spark的使用体验以及Spark原生的组件及API；提供存取MaxCompute数据源的能力；提供多租户场景更好的安全能力；提供管控平台，让Spark作业与MaxCompute作业共享资源、存储、用户体系，保证高性能和低成本。 Spark配合MaxCompute产品能够构建更加完善高效的数据处理解决方案，同时社区Spark的应用可以无缝的运行在**Spark on MaxCompute**之上。

Spark on MaxCompute与**原生Spark**的架构图比对如下所示。

图 779: Spark on MaxCompute与原生Spark架构比对



说明: 左侧是原生spark的架构图，右侧是**Spark on MaxCompute**的架构图。

Spark on MaxCompute的优势。

- **Spark on MaxCompute**运行在Cupid平台上，而Cupid平台提供强大的平台能力和统一接口，不需要入侵开源框架源代码。
- **Spark on MaxCompute**天然运行在MaxCompute集群内部，是目前能够提供的处理数据本地化最好的解决方案。

26.11.2 项目资源

使用**Spark on MaxCompute**时，你可能需要关注或者下载如下的相关项目资源。

- **Spark on MaxCompute**发布包：请下载最新的发布包。下载地址如下：*Spark on MaxCompute*。
- **Spark on MaxCompute**插件：本项目已开源，链接地址如下：*Aliyun Cupid SDK*。

26.11.3 环境设置

准备好上述的项目资源后，需要通过如下几个步骤的环境设置才可以运行GitHub项目中的相关Demo。

26.11.3.1 解压Spark on MaxCompute发布包

对下载好的**Spark on MaxCompute**最新发布包进行解压，解压后的文件夹结构如下。

```
.
..
|-- RELEASE
|-- bin
|-- conf
|-- examples
|-- jars
|-- python
|-- sbin
|-- yarn
|-- __spark_libs__.zip
```

26.11.3.2 设置环境变量

根据用户的实际情况，设置所需的环境变量。



说明: 主要是**JAVA_HOME**和**SPARK_HOME**的设置。

JAVA_HOME设置

```
export JAVA_HOME=/path/to/jdk
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
export PATH=$JAVA_HOME/bin:$PATH
```

SPARK_HOME设置

```
export SPARK_HOME=/path/to/spark_extracted_package
```

```
export PATH=$SPARK_HOME/bin:$PATH
```

26.11.3.3 设置Spark-defaults.conf

在*$SPARK_HOME/conf* 路径下存在**spark-defaults.conf**文件，需要在该文件中设置MaxCompute相关的账号信息后，才可以提交Spark任务到MaxCompute。

```
# 默认配置内容如下，将空白部分根据实际的账号信息填上即可
# OdpsAccount Info Setting
spark.hadoop.odps.project.name =
spark.hadoop.odps.access.id =
spark.hadoop.odps.access.key =
spark.hadoop.odps.end.point =
# Cupid Longtime Job
# spark.hadoop.odps.cupid.engine.running.type = longtime
# spark.hadoop.odps.cupid.job.capability.duration.hours = 8640
# spark.hadoop.odps.moye.trackurl.dutation = 8640
# Sql catalog for odps table
spark.sql.catalogImplementation = odps
spark.hadoop.odps.cupid.bearer.token.enable = false
# odps table split size, Unit(MB)
spark.hadoop.odps.input.split.size = 256
# Resource settings
spark.executor.instances = 1
spark.executor.cores = 2
spark.executor.memory = 6g
spark.driver.cores = 2
spark.driver.memory = 4g
spark.master = yarn-cluster
```

26.11.4 常用场景案例演示

本章节将给出一些常用场景的案例演示，帮助用户更快的了解**Spark on MaxCompute**的相关使用。

用户需要下载GitHub项目并对项目进行编译后，才能够运行相关Demo。

```
# 下载GitHub项目
git clone https://github.com/aliyun/aliyun-cupid-sdk.git
# 编译GitHub项目
cd aliyun-cupid-sdk
mvn -T 1C clean install -DskipTests
```

执行完上述步骤后，会产生相应的jar包，这些jar包会在下面具体的案例演示Demo中使用。

26.11.4.1 WordCount案例

Spark运行简单的WordCount。

案例代码如下。

```
/**
  * Licensed to the Apache Software Foundation (ASF) under one
```

```
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 * <p>
 * http://www.apache.org/licenses/LICENSE-2.0
 * <p>
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.aliyun.odps.spark.examples

import org.apache.spark.sql.SparkSession

/**
 * Computes an approximation to pi
 * 1. build aliyun-cupid-sdk
 * 2. properly set spark.defaults.conf
 * 3. bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.WordCount
       *     /path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-
SNAPSHOT-shaded.jar
 */
object WordCount {
  /**
   * To run this demo in IDEA, you need to do following:
   *   1. add spark-assembly jar into Project dependency
   *   2. use the local mode statement
   */
  def main(args: Array[String]) {
    // local mode statement
    // val spark = SparkSession
    //   .builder()
    //   .appName("WordCount")
    //   .master("local[4]")
    //   .getOrCreate()

    val spark = SparkSession
      .builder()
      .appName("WordCount")
      .getOrCreate()
    val sc = spark.sparkContext
    try {
                  sc.parallelize(1 to 100, 10).map(word => (word, 1)).reduceByKey(_ + _,
 10).take(100).foreach(println)
    } finally {
      sc.stop()
    }
  }
}
```

提交作业的运行命令如下。

```
bin/spark-submit \
 --master yarn-cluster \
 --class com.aliyun.odps.spark.examples.WordCount \
```

```
            /path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-
SNAPSHOT-shaded.jar
```

26.11.4.2 SparkPi案例

Spark运行SparkPi。

案例代码如下。

```
/**
  * Licensed to the Apache Software Foundation (ASF) under one
  * or more contributor license agreements.  See the NOTICE file
  * distributed with this work for additional information
  * regarding copyright ownership.  The ASF licenses this file
  * to you under the Apache License, Version 2.0 (the
  * "License"); you may not use this file except in compliance
  * with the License.  You may obtain a copy of the License at
  * <p>
  * http://www.apache.org/licenses/LICENSE-2.0
  * <p>
  * Unless required by applicable law or agreed to in writing, software
  * distributed under the License is distributed on an "AS IS" BASIS,
  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  * See the License for the specific language governing permissions and
  * limitations under the License.
  */

package com.aliyun.odps.spark.examples

import org.apache.spark.sql.SparkSession

import scala.math.random

/**
  * Computes an approximation to pi
  * 1. build aliyun-cupid-sdk
  * 2. properly set spark.defaults.conf
  * 3. bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.SparkPi
         *     /path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-
SNAPSHOT-shaded.jar
  */
object SparkPi {
  def main(args: Array[String]) {
    val spark = SparkSession
      .builder()
      .appName("SparkPi")
      .getOrCreate()
    try {
      val slices = if (args.length > 0) args(0).toInt else 2
      val n = math.min(100000L * slices, Int.MaxValue).toInt // avoid overflow
      val count = spark.sparkContext.parallelize(1 until n, slices).map { i =>
        val x = random * 2 - 1
        val y = random * 2 - 1
        if (x * x + y * y < 1) 1 else 0
      }.reduce(_ + _)
      println("Pi is roughly " + 4.0 * count / n)
    } finally {
      spark.stop()
    }
  }
```

```
}
```

提交作业的运行命令如下。

```
bin/spark-submit \
 --master yarn-cluster \
 --class com.aliyun.odps.spark.examples.SparkPi \
        /path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-
SNAPSHOT-shaded.jar
```

26.11.4.3 MaxCompute Table Rdd案例

读写MaxCompute Table，转化为Spark RDD。



注意: 案例Demo中的Projct/Table必须存在，或者更改为对应的Project/Table。

案例代码如下。

```
/**
  * Licensed to the Apache Software Foundation (ASF) under one
  * or more contributor license agreements.  See the NOTICE file
  * distributed with this work for additional information
  * regarding copyright ownership.  The ASF licenses this file
  * to you under the Apache License, Version 2.0 (the
  * "License"); you may not use this file except in compliance
  * with the License.  You may obtain a copy of the License at
  * <p>
  * http://www.apache.org/licenses/LICENSE-2.0
  * <p>
  * Unless required by applicable law or agreed to in writing, software
  * distributed under the License is distributed on an "AS IS" BASIS,
  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  * See the License for the specific language governing permissions and
  * limitations under the License.
  */

package com.aliyun.odps.spark.examples

import com.aliyun.odps.data.Record
import com.aliyun.odps.{PartitionSpec, TableSchema}
import org.apache.spark.odps.OdpsOps
import org.apache.spark.sql.SparkSession

import scala.util.Random

/**
  * 1. build aliyun-cupid-sdk
  * 2. properly set spark.defaults.conf
                    *           3.           bin/spark-submit           --master           yarn-cluster           --class
 com.aliyun.odps.spark.examples.OdpsTableReadWrite \
        *     /path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-
SNAPSHOT-shaded.jar
  */
object OdpsTableReadWrite {
  /**
    * This Demo represent the ODPS Table Read/Write Usage in following scenarios
    * This Demo will fail if you don't have such Table inside such Project
```

```
   *   1. read from normal table via rdd api
   *   2. read from single partition column table via rdd api
   *   3. read from multi partition column table via rdd api
   *   4. read with multi partitionSpec definition via rdd api
   *   5. save rdd into normal table
   *   6. save rdd into partition table with single partition spec
   *   7. dynamic save rdd into partition table with multiple partition spec
   *
   */
  def main(args: Array[String]) {

    val spark = SparkSession
      .builder()
      .appName("OdpsTableReadWrite")
      .getOrCreate()

    val sc = spark.sparkContext
    val projectName = sc.getConf.get("odps.project.name")

    try {
      val odpsOps = new OdpsOps(sc)

      /**
        * read from normal table via rdd api
        * desc cupid_wordcount;
        * +------------------------------------------------------------------------------------+
        * | Field           | Type       | Label | Comment                                     |
        * +------------------------------------------------------------------------------------+
        * | id              | string     |       |                                             |
        * | value           | string     |       |                                             |
        * +------------------------------------------------------------------------------------+
        */
      val rdd_0 = odpsOps.readTable(
        projectName,
        "cupid_wordcount",
        (r: Record, schema: TableSchema) => (r.getString(0), r.getString(1))
      )

      /**
        * read from single partition column table via rdd api
        * desc dftest_single_parted;
        * +------------------------------------------------------------------------------------+
        * | Field           | Type       | Label | Comment                                     |
        * +------------------------------------------------------------------------------------+
        * | id              | string     |       |                                             |
        * | value           | string     |       |                                             |
        * +------------------------------------------------------------------------------------+
        * | Partition Columns:                                                                 |
        * +------------------------------------------------------------------------------------+
        * | pt              | string     |                                                     |
        * +------------------------------------------------------------------------------------+
        */
      val rdd_1 = odpsOps.readTable(
        projectName,
        "dftest_single_parted",
        Array("pt=20160101"),
        (r: Record, schema: TableSchema) => (r.getString(0), r.getString(1), r.getString("pt"))
      )

      /**
        * read from multi partition column table via rdd api
        * desc dftest_parted;
```

```
    * +------------------------------------------------------------------------------+
    * | Field          | Type       | Label | Comment                              |
    * +------------------------------------------------------------------------------+
    * | id             | string     |       |                                      |
    * | value          | string     |       |                                      |
    * +------------------------------------------------------------------------------+
    * | Partition Columns:                                                           |
    * +------------------------------------------------------------------------------+
    * | pt             | string     |                                              |
    * | hour           | string     |                                              |
    * +------------------------------------------------------------------------------+
    */
  val rdd_2 = odpsOps.readTable(
    projectName,
    "dftest_parted",
    Array("pt=20160101,hour=12"),
      (r: Record, schema: TableSchema) => (r.getString(0), r.getString(1), r.getString("pt"),
r.getString(3))
  )

  /**
    * read with multi partitionSpec definition via rdd api
    * desc cupid_partition_table1;
    * +------------------------------------------------------------------------------+
    * | Field          | Type       | Label | Comment                              |
    * +------------------------------------------------------------------------------+
    * | id             | string     |       |                                      |
    * | value          | string     |       |                                      |
    * +------------------------------------------------------------------------------+
    * | Partition Columns:                                                           |
    * +------------------------------------------------------------------------------+
    * | pt1            | string     |                                              |
    * | pt2            | string     |                                              |
    * +------------------------------------------------------------------------------+
    */
  val rdd_3 = odpsOps.readTable(
    projectName,
    "cupid_partition_table1",
    Array("pt1=part1,pt2=part1", "pt1=part1,pt2=part2", "pt1=part2,pt2=part3"),
      (r: Record, schema: TableSchema) => (r.getString(0), r.getString(1), r.getString("pt1"),
r.getString("pt2"))
  )

  /**
    * save rdd into normal table
    * desc cupid_wordcount_empty;
    * +------------------------------------------------------------------------------+
    * | Field          | Type       | Label | Comment                              |
    * +------------------------------------------------------------------------------+
    * | id             | string     |       |                                      |
    * | value          | string     |       |                                      |
    * +------------------------------------------------------------------------------+
    */
  val transfer_0 = (v: Tuple2[String, String], record: Record, schema: TableSchema) => {
    record.set("id", v._1)
    record.set(1, v._2)
  }
  odpsOps.saveToTable(projectName, "cupid_wordcount_empty", rdd_0, transfer_0, true)

  /**
    * save rdd into partition table with single partition spec
    * desc cupid_partition_table1;
```

```
      * +------------------------------------------------------------------------+
      * | Field          | Type       | Label | Comment                          |
      * +------------------------------------------------------------------------+
      * | id             | string     |       |                                  |
      * | value          | string     |       |                                  |
      * +------------------------------------------------------------------------+
      * | Partition Columns:                                                     |
      * +------------------------------------------------------------------------+
      * | pt1            | string     |                                          |
      * | pt2            | string     |                                          |
      * +------------------------------------------------------------------------+
      */
    val transfer_1 = (v: Tuple2[String, String], record: Record, schema: TableSchema) => {
      record.set("id", v._1)
      record.set("value", v._2)
    }
      odpsOps.saveToTable(projectName, "cupid_partition_table1", "pt1=test,pt2=dev", rdd_0,
transfer_1, true)

    /**
      * dynamic save rdd into partition table with multiple partition spec
      * desc cupid_partition_table1;
      * +------------------------------------------------------------------------+
      * | Field          | Type       | Label | Comment                          |
      * +------------------------------------------------------------------------+
      * | id             | string     |       |                                  |
      * | value          | string     |       |                                  |
      * +------------------------------------------------------------------------+
      * | Partition Columns:                                                     |
      * +------------------------------------------------------------------------+
      * | pt1            | string     |                                          |
      * | pt2            | string     |                                          |
      * +------------------------------------------------------------------------+
      */
      val transfer_2 = (v: Tuple2[String, String], record: Record, part: PartitionSpec, schema:
TableSchema) => {
      record.set("id", v._1)
      record.set("value", v._2)

      val pt1_value = if (new Random().nextInt(10) % 2 == 0) "even" else "odd"
      val pt2_value = if (new Random().nextInt(10) % 2 == 0) "even" else "odd"
      part.set("pt1", pt1_value)
      part.set("pt2", pt2_value)
    }
        odpsOps.saveToTableForMultiPartition(projectName, "cupid_partition_table1", rdd_0,
transfer_2, true)
  } catch {
    case ex: Exception => {
      throw ex
    }
  } finally {
    sc.stop
  }
 }
}
```

提交作业的运行命令如下。

```
bin/spark-submit \
 --master yarn-cluster \
 --class com.aliyun.odps.spark.examples.OdpsTableReadWrite \
```

```
        /path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

MaxCompute Table读取并发度可以通过如下两个方法进行调整。

- 调整**spark.hadoop.odps.input.split.size**，该值越大，map task越少，默认为256MB。
- 通过**OdpsOps.readTable**中的**numPartition**进行设置，该值可直接决定map task个数，模式按照**spark.hadoop.odps.input.split.size**进行计算。

26.11.4.4 MaxCompute Table Spark-SQL案例

使用sqlContext读写MaxCompute Table。

注意:

- 案例Demo中的Projct/Table必须存在，或者更改为对应的Project/Table。
- **Spark-defaults.conf**里必须设置*spark.sql.catalogImplementation = odps*。

案例代码如下。

```
/**
  * Licensed to the Apache Software Foundation (ASF) under one
  * or more contributor license agreements.  See the NOTICE file
  * distributed with this work for additional information
  * regarding copyright ownership.  The ASF licenses this file
  * to you under the Apache License, Version 2.0 (the
  * "License"); you may not use this file except in compliance
  * with the License.  You may obtain a copy of the License at
  * <p>
  * http://www.apache.org/licenses/LICENSE-2.0
  * <p>
  * Unless required by applicable law or agreed to in writing, software
  * distributed under the License is distributed on an "AS IS" BASIS,
  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  * See the License for the specific language governing permissions and
  * limitations under the License.
  */

package com.aliyun.odps.spark.examples

import org.apache.spark.sql.SparkSession

/**
  * 1. build aliyun-cupid-sdk
  * 2. properly set spark.defaults.conf
      * 3. bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.OdpsTableReadWriteViaSQL \
     *        /path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
  */
object OdpsTableReadWriteViaSQL {

  def main(args: Array[String]) {

    // please make sure spark.sql.catalogImplementation=odps in spark-defaults.conf
```

```
    // to enable odps catalog
    val spark = SparkSession
      .builder()
      .appName("OdpsTableReadWriteViaSQL")
      .getOrCreate()

    val projectName = spark.sparkContext.getConf.get("odps.project.name")
    val tableName = "cupid_wordcount"

    // get a ODPS table as a DataFrame
    val df = spark.table(tableName)
    println(s"df.count: ${df.count()}")

    // Just do some query
    spark.sql(s"select * from $tableName limit 10").show(10, 200)
    spark.sql(s"select id, count(id) from $tableName group by id").show(10, 200)

    // any table exists under project could be use
    // productRevenue
    spark.sql(
      """
        |SELECT product,
        |       category,
        |       revenue
        |FROM
        |  (SELECT product,
        |          category,
        |          revenue,
        |          dense_rank() OVER (PARTITION BY category
        |                             ORDER BY revenue DESC) AS rank
        |   FROM productRevenue) tmp
        |WHERE rank <= 2
      """.stripMargin).show(10, 200)

    spark.stop()
  }
}
```

提交作业的运行命令如下。

```
bin/spark-submit \
 --master yarn-cluster \
 --class com.aliyun.odps.spark.examples.OdpsTableReadWrite \
            /path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-
SNAPSHOT-shaded.jar
```

26.11.4.5 MaxCompute自研Client模式案例

由于安全考虑，MaxCompute里的机器不能进行直连。所以原生Spark中的**yarn-client**模式无法使用。为了解决交互式的需求，MaxCompute团队自主研发了Client模式。

案例代码如下。

```
/**
  * Licensed to the Apache Software Foundation (ASF) under one
  * or more contributor license agreements.  See the NOTICE file
  * distributed with this work for additional information
  * regarding copyright ownership.  The ASF licenses this file
```

```
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 * <p>
 * http://www.apache.org/licenses/LICENSE-2.0
 * <p>
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.aliyun.odps.spark.examples

import com.aliyun.odps.cupid.client.spark.client.CupidSparkClientRunner

/**
 * 1. build aliyun-cupid-sdk
 * 2. properly set spark.defaults.conf
 * 3. set $SPARK_HOME
         *     4.     java     -cp     /path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-
examples_2.11-1.0.0-SNAPSHOT-shaded.jar:$SPARK_HOME/jars/\* \
  *   com.aliyun.odps.spark.examples.SparkClientNormalFT /path/to/aliyun-cupid-sdk/examples/
client-jobexamples/target/client-jobexamples_2.11-1.0.0-SNAPSHOT.jar
 */
object SparkClientNormalFT {
  def main(args: Array[String]) {
    val cupidSparkClient = CupidSparkClientRunner.getReadyCupidSparkClient()
    val jarPath = args(0) //client-jobexamples jar path
    val sparkClientNormalApp = new SparkClientNormalApp(cupidSparkClient)
    sparkClientNormalApp.runNormalJob(jarPath)
    cupidSparkClient.stopRemoteDriver()
  }
}
```

提交作业的运行命令如下。

```
java -cp \
/path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-
SNAPSHOT-shaded.jar:$SPARK_HOME/jars/* \
com.aliyun.odps.spark.examples.SparkClientNormalFT        /path/to/aliyun-cupid-sdk/examples/
client-jobexamples/target/client-jobexamples_2.11-1.0.0-SNAPSHOT.jar
```

26.11.4.6 MaxCompute Table PySpark案例

PySpark读写MaxCompute Table。

案例代码如下。

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from odps.odps_sdk import OdpsOps
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext, DataFrame

if __name__ == '__main__':
    """
```

```
    跑通pyspark on odps的例子需要满足以下条件
    1. SPARK on odps client已经下载完毕，并配置好SPARK_HOME以及对应的ODPS配置
    2. 找到odps.zip, 就在同一个目录下
    3. 下载好cupid-datasource_2.11-x.y.z.jar, 请使用最新版本的jar包
    4. 运行命令如下:
       spark-submit --master yarn-cluster --jars cupid-datasource_2.11-x.y.z.jar --py-files odps.zip
 odps_table_rw.py
    """

    conf = SparkConf().setAppName("odps_pyspark")
    sc = SparkContext(conf=conf)
    sql_context = SQLContext(sc)

    project_name = "cupid_testa1"
    in_table_name = "cupid_wordcount"
    out_table_name = "cupid_wordcount_py"

    # 读取odps table 返回Python DataFrame
    # OdpsOps.read_odps_table签名如下:
    #   sql_context: Python SQLContext
    #   project_name: odps project name(string)
    #   table_name: odps table name(string)
    #   partitions: partition definitions(array[string]), for example
    #       ["pt1='part1',pt2='part1'", "pt1='part1',pt2='part2'"], default []
    #   num_partition: spark partition(int), default 0
    normal_df = OdpsOps.read_odps_table(sql_context, project_name, in_table_name)

                          #       df的operation请参考       https://spark.apache.org/docs/1.5.1/api/python/
 pyspark.sql.html#pyspark.sql.DataFrame
    # 为了不让console打印太多，用了sample函数
    for i in normal_df.sample(False, 0.01).collect():
        print i
    print "Read normal odps table finished"

    # 将Python DataFrame写入odps table
    # 不接受其他类型参数，必须是Python DataFrame，并且schema必须与odps table一致
    # OdpsOps.write_odps_table签名如下:
    #   sql_context: Python SQLContext
    #   py_df: Python DataFrame
    #   project_name: odps project name(string)
    #   table_name: odps table name(string)
    #   partition_str: partition definitions(string), for example
    #       "pt1='part1',pt2='part3'", default ""
    #   is_overwrite: is_overwrite(boolean), default False
        OdpsOps.write_odps_table(sql_context,  normal_df.sample(False,  0.001),  project_name,
 out_table_name)
    print "Write normal odps table finished"
```

提交作业的运行命令如下。

```
spark-submit \
 --master yarn-cluster \
  --jars   /path/to/aliyun-cupid-sdk/external/cupid-datasource/target/cupid-datasource_2.11-1.0.0-
SNAPSHOT.jar \
 --py-files /path/to/aliyun-cupid-sdk/examples/spark-examples/src/main/python/odps.zip \
```

```
/path/to/aliyun-cupid-sdk/examples/spark-examples/src/main/python/odps_table_rw.py
```

26.11.5 Maven依赖

上述章节中的GitHub项目可以作为用户QuickStart开发的模版，如果用户需要自定义开发，请确认pom.xml文件如下所示。

```
# spark的模块的版本使用社区2.1.0的版本即可，并且保证scope是provided
<dependency>
 <groupId>org.apache.spark</groupId>
   <artifactId>spark-core_2.11</artifactId>
   <version>2.1.0</version>
   <scope>provided</scope>
</dependency>
# MaxCompute插件已经发布到Maven仓库，需添加以下依赖
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>cupid-core_2.11</artifactId>
    <version>1.0.0</version>
    <scope>provided</scope>
</dependency>

<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>cupid-client_2.11</artifactId>
    <version>1.0.0</version>
</dependency>

<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>cupid-datasource_2.11</artifactId>
    <version>1.0.0</version>
</dependency>
```

maven库的文件列表如下所示：

- Cupid平台core代码，包括cupid task提交接口封装，以及父子进程读写表相关接口。

  ```
  <dependency>
        <groupId>com.aliyun.odps</groupId>
        <artifactId>cupid-core_2.11</artifactId>
        <version>1.0.0</version>
    </dependency>
  ```

- datasource，封装了spark相关的MaxCompute Table读写的用户接口。

  ```
  <dependency>
        <groupId>com.aliyun.odps</groupId>
        <artifactId>cupid-datasource_2.11</artifactId>
        <version>1.0.0</version>
  ```

- 封装了Cupid Client模式的用户SDK。

  ```
  <dependency>
        <groupId>com.aliyun.odps</groupId>
        <artifactId>cupid-client_2.11</artifactId>
  ```

```
        <version>1.0.0</version>
</dependency>
```

26.11.6 特殊说明

本章节将对使用中的一些特殊情况进行相关说明。

26.11.6.1 Streaming任务

MaxCompute也支持Spark Streaming，为了支持长任务运行，需要在**spark-defaults.conf**中添加如下的特殊配置。

```
# 不同于离线作业，Streaming作业有一些特殊配置
# 这些配置也是配置在spark-defaults.conf即可生效
# 配置成longtime才不会被回收
spark.hadoop.odps.cupid.engine.running.type=longtime
# 时间长度
spark.hadoop.odps.cupid.job.capability.duration.hours=25920
# failOver最大重试次数
spark.yarn.maxAppAttempts=10
# 是否启用writeAheadLog模式，能够保证数据不丢失，但是效率会降低
spark.streaming.receiver.writeAheadLog.enable=true
```

26.11.6.2 Tracking Url

提交完成作业后，一般会有如下输出。

```
17/08/28 14:53:26 INFO Client:
client token: N/A
diagnostics: N/A
ApplicationMaster host: 11.137.199.2
ApplicationMaster RPC port: 57524
queue: queue
start time: 1503903179541
final status: SUCCEEDED
tracking URL: http://jobview.odps.aliyun-inc.com/proxyview/jobview/?h=http://service.
odps.aliyun-inc.com/api&p=odps_public_dev&i=20170828065141675g5h4t6u1&t=spark&id=
application_1503903039442_1185611255&metaname=20170828065141675g5h4t6u1&token=
L0dSMHRkSlNXS2ZkeFE1UkVsckthTTZQWHV3PSxPRFBTX09CTzoxMDU5NTgy
NzI0MzIyOT          k5LDE1MDQxNjIzODMseyJTdGF0ZW1lbnQiOlt7IkFjdGlvbiI6WyJvZHBzOl
JlYWQiXSwi      RWZmZWN0IjoiQWxsb3ciLCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnByb2pl
Y3RzL29kcH      NfcHVibGljX2Rldi9pbnN0YW5jZXMvMjAxNzA4MjgwNjUxNDE2NzVnNWg0dD
Z1MSJdfV0s IlZlcnNpb24iOiIxIn0=
user: user
```

在输出示例中看到TrackingUrl，表示用户的作业已经提交到MaxCompute集群。

说明: 上面输出示例中的TrackingUrl非常关键，它既是SparkWebUI，也是HistoryServer的Url。

26.11.7 Base支持

用户可以在base中运行spark离线作业（cluster模式），便于与MaxCompute其他类型的执行节点进行集成和调度。进行操作前，用户需要首先上传以下几个resource。

1. **$SPARK_HOME/__spark_libs__.zip**，就是**$SPARK_HOME/jars**的压缩包。
2. 用户编译的shaded包，需要注意这个shaded包里必须依赖**odps-spark-datasource jar**。
3. **spark-defaults.conf**。



注意: 在base上commit完才能算资源上传完成。

上传完成之后，新建一个MaxCompute SQL类型的节点，添加如下代码。

```
jar -classpath
http://schedule@{env}inside.cheetah.alibaba-inc.com/scheduler/res?id=**__spark_libs__.zip
    resource    id**,http://schedule@{env}inside.cheetah.alibaba-inc.com/scheduler/res?id=**spark-
examples-2.0.0-SNAPSHOT-shaded.jar**
com.aliyun.odps.cupid.tools.OdpsCltWrapper
"--class" com.aliyun.odps.spark.examples.SparkPi
"--properties-file"                http://schedule@{env}inside.cheetah.alibaba-inc.com/scheduler/res?
id=**spark_2_1_0_conf**
"--conf"       spark.yarn.archive=http://schedule@{env}inside.cheetah.alibaba-inc.com/scheduler/
res?id=**__spark_libs__.zip resource id**
"--master" yarn-cluster
http://schedule@{env}inside.cheetah.alibaba-inc.com/scheduler/res?id=**spark-
examples-1.0.0-SNAPSHOT-shaded.jar**
#          请注意，这个          http://schedule@{env}inside.cheetah.alibaba-inc.com/scheduler/res?
id=**__spark_libs__.zip** 需要用户在 **** 中间替换成用户空间该资源真正的resource id
#示例代码如下：
jar -classpath
http://schedule@{env}inside.cheetah.alibaba-inc.com/scheduler/res?id=7189035,http://
schedule@{env}inside.cheetah.alibaba-inc.com/scheduler/res?id=7189038
com.aliyun.odps.cupid.tools.OdpsCltWrapper
"--class" com.aliyun.odps.spark.examples.SparkPi
"--properties-file"                http://schedule@{env}inside.cheetah.alibaba-inc.com/scheduler/res?
id=7189067
"--conf"       spark.yarn.archive=http://schedule@{env}inside.cheetah.alibaba-inc.com/scheduler/
res?id=7189035
"--master" yarn-cluster
http://schedule@{env}inside.cheetah.alibaba-inc.com/scheduler/res?id=7189038
```



说明:

在base中提交作业，不同于**spark-submit**，不需要指定用户的MaxCompute account信息，只需要提供一个工号即可，示例如下。

```
# spark-submit中需要的:
# spark.hadoop.odps.project.name
# spark.hadoop.odps.access.id
# spark.hadoop.odps.access.key
# spark.hadoop.odps.end.point
# 都不应该出现
```

```
# 正确示例如下 biz_id的尖括号是必须的
spark.hadoop.com.aliyun.biz_id=130400^
spark.hadoop.odps.cupid.bearer.token.enable = false
spark.hadoop.fs.file.impl = org.apache.hadoop.fs.LocalFileSystem
spark.executor.instances=6
spark.executor.cores=2
spark.executor.memory=6g
spark.driver.cores=2
spark.driver.memory=4g
spark.master=yarn-cluster
```

获取命令中resource id的url的操作步骤如下所示。

1. 打开**新建资源**对话框，选择需要上传的文件后，单击**提交**。



2. 进入如下界面后，单击如图所示左上角的按钮，上传资源完成后提示上传成功。



3. 右键单击成功上传的相关base资源，选择**查看历史版本** > **版本信息**，即可查看到对应的url ，如下图所示。



26.12 非结构化数据处理使用指南

26.12.1 如何在MaxCompute上访问并处理OSS非结构化数据

26.12.1.1 前言

MaxCompute作为阿里云大数据平台的核心计算组件，拥有强大的计算能力，能够调度大量的节点进行并行计算，同时对分布式计算中的failover，重试等均有一套行之有效的处理管理机制。而MaxCompute SQL能在简明的语义上实现各种数据处理逻辑，在集团内外更是广为应用，在其上实现与各种数据源的互通，对于打通整个阿里云的数据生态具有重要意义。

下面将通过几个简单的示例，演示如何在MaxCompute上访问并处理OSS非结构化数据。

26.12.1.2 使用内置extractor读取OSS数据

使用MaxCompute内置的extractor，可以非常方便的读取按照约定格式存储的OSS数据。用户只需要创建一个外部表，就能以这张表作为源表进行查询操作。假设有一份csv数据保存在OSS上，**endpoint**为oss-cn-shanghai-internal.aliyuncs.com，**bucket**为oss-odps-test，数据文件放在*/demo/SampleData/CSV/AmbulanceData/vehicle.csv*中。相关操作示例如下。

26.12.1.2.1 创建外部表

执行如下命令，创建外部表。

```
CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_csv_external
(
vehicleId bigint,
recordId bigint,
patientId bigint,
calls bigint,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
STORED BY 'com.aliyun.odps.CsvStorageHandler'
LOCATION  'oss://<*your-id*>:<*your-secret-key*>@oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/Demo/SampleData/CSV/AmbulanceData/';
```

说明:

- com.aliyun.odps.CsvStorageHandler是内置的处理csv格式文件的StorageHandler，它定义了如何读写csv文件。用户只需要指明这个名字即可，相关逻辑已经由系统实现。
- LOCATION必须指定一个OSS目录，默认系统会读取这个目录下所有的文件。
- 外部表只是在系统中记录了与OSS目录的关联，当DROP这张表时，对应的LOCATION数据不会被删除。
- 为了便于演示，此处明文传入了访问OSS的credential。如果考虑信息安全风险，需要部署STS服务，并使用STS方式访问OSS，具体操作请参考进阶使用#通过*STS*访问*OSS*。

26.12.1.2.2 查询外部表

外部表创建成功后，用户可以像使用普通表一样使用这个外部表。

假设*/demo/SampleData/CSV/AmbulanceData/vehicle.csv*的数据如下：

```
1,1,51,1,46.81006,-92.08174,9/14/2014 0:00,S
1,2,13,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,3,48,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,4,30,1,46.81006,-92.08174,9/14/2014 0:00,W
1,5,47,1,46.81006,-92.08174,9/14/2014 0:00,S
1,6,9,1,46.81006,-92.08174,9/14/2014 0:00,S
1,7,53,1,46.81006,-92.08174,9/14/2014 0:00,N
1,8,63,1,46.81006,-92.08174,9/14/2014 0:00,SW
1,9,4,1,46.81006,-92.08174,9/14/2014 0:00,NE
```

```
1,10,31,1,46.81006,-92.08174,9/14/2014 0:00,N
```

执行如下命令，提交一个作业，调用内置csv extractor，从OSS读取数据进行处理。

```
SELECT recordId, patientId, direction
FROM ambulance_data_csv_external
WHERE patientId > 25;
```

输出结果如下：

```
+------------+------------+-----------+
| recordId | patientId | direction |
+------------+------------+-----------+
| 1 | 51 | S |
| 3 | 48 | NE |
| 4 | 30 | W |
| 5 | 47 | S |
| 7 | 53 | N |
| 8 | 63 | SW |
| 10 | 31 | N |
+------------+------------+-----------+
```



说明: 目前系统内置提供了CsvStorageHandler，TsvStorageHandler以及TextStorageHandler。

26.12.1.3 自定义extractor

当OSS中数据格式比较复杂，内置的extractor无法满足需求时，需要自定义extractor来读取OSS文件中的数据。假设有一份text数据文件保存在OSS上，并不是csv格式，记录之间的列通过|分割。数据文件放在*/demo/SampleData/CustomTxt/AmbulanceData/vehicle.csv*中。相关操作示例如下。

26.12.1.3.1 定义StorageHandler

此时用户可以提供自己的数据解析逻辑。 而StorageHandler就是作为这些自定义逻辑的统一入口，用来指定用户自己实现的extractor和outputer的类型。这个入口通常只是提供一个非常简单的定义，例如可以实现如下示例中的一个**SpeicalTextStorageHandler**。

```
package com.aliyun.odps.udf.example.text;
public class SpeicalTextStorageHandler extends OdpsStorageHandler {
@Override
public Class<? extends Extractor> getExtractorClass() {
return TextExtractor.class;
}
@Override
public Class<? extends Outputer> getOutputerClass() {
return TextOutputer.class;
}
```

```
}
```



说明: 需要注意，对于上述的这种数据格式（用“|”分割的文本数据），MaxCompute内置的TextStorageHandler是可以处理的。此处只是作为一个简单的示例，用来介绍使用SDK定制自定义storage handler（尤其是extractor）来处理特殊数据的方法。

26.12.1.3.2 定义Extractor

下面示例中的**TextExtractor**可以将分隔符作为参数传输进来，可以处理所有类似格式的文本文件。

```
/**
* Text extractor that extract schematized records from formatted plain-
text(csv, tsv etc.)
**/
public class TextExtractor extends Extractor {
private InputStreamSet inputs;
private String columnDelimiter;
private DataAttributes attributes;
private BufferedReader currentReader;
private boolean firstRead = true;
public TextExtractor() {
// default to ",", this can be overwritten if a specific delimiter is
provided (via DataAttributes)
this.columnDelimiter = ",";
}
// no particular usage for execution context in this example
@Override
public void setup(ExecutionContext ctx, InputStreamSet inputs,
DataAttributes attributes) {
this.inputs = inputs;
--inputs是一个InputStreamSet，每次调用next()返回一个InputStream，这个InputStream可以读取一个OSS文件的所有内容。
this.attributes = attributes;
// check if "delimiter" attribute is supplied via SQL query
String columnDelimiter = this.attributes.getValueByKey("delimiter");
--delimiter通过DDL语句传输参数。
if ( columnDelimiter != null)
{
this.columnDelimiter = columnDelimiter;
}
// note: more properties can be inited from attributes if needed
}
@Override
public Record extract() throws IOException {
String line = readNextLine();
if (line == null) {
return null;
--返回null表示这个表中已经没有记录可读了。
}
return textLineToRecord(line);
--textLineToRecord将一行数据按照delimiter分割为多个列。完整实现可以参考完整的TextExtractor实现。
--extactor()调用返回一条Record，代表从OSS数据中抽取出的一条Record。
}
@Override
public void close(){
```

```
// no-op
}
}
```

26.12.1.3.3 编译打包

类似于普通Java UDF的使用方式，用户可以将Java代码实现编译及打包，并通过如下命令上传到MaxCompute。

add jar odps-udf-example.jar;

26.12.1.3.4 创建外部表

在代码jar包上传完成后，与使用内置extractor类似，用户同样需要执行如下命令，建立一个外部表。不同的是，此时在指定外部表访问数据的时候，需要使用自定义的StorageHandler。

```
CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_txt_external
(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
STORED BY 'com.aliyun.odps.udf.example.text.SpeicalTextStorageHandler'
--STORED BY指定自定义StorageHandler的类名。
WITH SERDEPROPERTIES('delimiter'='|')
--SERDEPROPERITES可以指定参数，这些参数会通过DataAttributes传递到Extractor代码中。
LOCATION 'oss://<*your-id*>:<*your-secret-key*>@oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/Demo/SampleData/CustomTxt/AmbulanceData/'
USING 'odps-udf-example.jar';
--同时需要指定类定义所在的jar包。
```

26.12.1.3.5 查询外部表

假设*/demo/SampleData/CustomTxt/AmbulanceData/vehicle.csv*的数据如下：

```
1|1|51|1|46.81006|-92.08174|9/14/2014 0:00|S
1|2|13|1|46.81006|-92.08174|9/14/2014 0:00|NE
1|3|48|1|46.81006|-92.08174|9/14/2014 0:00|NE
1|4|30|1|46.81006|-92.08174|9/14/2014 0:00|W
1|5|47|1|46.81006|-92.08174|9/14/2014 0:00|S
1|6|9|1|46.81006|-92.08174|9/14/2014 0:00|S
1|7|53|1|46.81006|-92.08174|9/14/2014 0:00|N
1|8|63|1|46.81006|-92.08174|9/14/2014 0:00|SW
1|9|4|1|46.81006|-92.08174|9/14/2014 0:00|NE
1|10|31|1|46.81006|-92.08174|9/14/2014 0:00|N
```

执行如下命令，提交一个作业，调用自定义的extractor，从OSS读取数据进行处理。

```
SELECT recordId, patientId, direction
```

```
FROM ambulance_data_txt_external
WHERE patientId > 25;
```

输出结果如下：

```
+------------+------------+-----------+
| recordId | patientId | direction |
+------------+------------+-----------+
| 1 | 51 | S |
| 3 | 48 | NE |
| 4 | 30 | W |
| 5 | 47 | S |
| 7 | 53 | N |
| 8 | 63 | SW |
| 10 | 31 | N |
+------------+------------+-----------+
```

26.12.1.4 进阶使用

相关的进阶使用示例如下。

26.12.1.4.1 通过自定义extractor读取外部非结构化数据

在前面的章节中，给出了通过内置与自定义的extractor处理存储在OSS上的csv等文本数据的方法。本章节将以语音数（wav格式文件）为例，介绍如何通过自定义的extractor来访问处理OSS上的非文本文件。

此处以如何处理存放在OSS上的语音文件作为操作示例。

1. 自定义SpeechSentenceSnrExtractor主体逻辑。用户在setup接口中读取参数，进行初始化，并且导入语音处理模型（通过resource引入）。

```
public SpeechSentenceSnrExtractor(){
this.utteranceLabels = new HashMap<String, UtteranceLabel>();
}
@Override
public void setup(ExecutionContext ctx, InputStreamSet inputs,
DataAttributes attributes){
this.inputs = inputs;
this.attributes = attributes;
this.mlfFileName = this.attributes.getValueByKey(MLF_FILE_ATTRIBUTE_KEY);
String sampleRateInKHzStr =
this.attributes.getValueByKey(SPEECH_SAMPLE_RATE_KEY);
this.sampleRateInKHz = Double.parseDouble(sampleRateInKHzStr);
try {
// read the speech model file from resource and load the model into
memory
BufferedInputStream inputStream =
ctx.readResourceFileAsStream(mlfFileName);
loadMlfLabelsFromResource(inputStream);
inputStream.close();
} catch (IOException e) {
throw new RuntimeException("reading model from mlf failed with exception
" + e.getMessage());
}
```

```
}
@Override
public Record extract() throws IOException {
SourceInputStream inputStream = inputs.next();
if (inputStream == null){
return null;
}
// process one wav file to extract one output record [snr, id]
String fileName = inputStream.getFileName();
fileName = fileName.substring(fileName.lastIndexOf('/') + 1);
logger.info("Processing wav file " + fileName);
// infer id from speech file name
String id = fileName.substring(0, fileName.lastIndexOf('.'));
// read speech file into memory buffer
long fileSize = inputStream.getFileSize();
byte[] buffer = new byte[(int)fileSize];
int readSize = inputStream.readToEnd(buffer);
inputStream.close();
// compute the avg sentence snr from speech file
double snr = computeSnr(id, buffer, readSize);
// construct output record [snr, id]
Column[] outputColumns = this.attributes.getRecordColumns();
ArrayRecord record = new ArrayRecord(outputColumns);
record.setDouble(0, snr);
record.setString(1, id);
return record;
}
private void loadMlfLabelsFromResource(BufferedInputStream fileInputStream)
throws IOException {
// loading MLF label from resource, skipped here
}
// compute the snr of the speech sentence, assuming the input buffer
contains the entire content of a wav file
private double computeSnr(String id, byte[] buffer, int validBufferLen){
// computing the snr value for the wav file (supplied as byte buffer
array), skipped here
}
```

说明: extract()接口中，实现了对语音文件的具体读取和处理逻辑，对读取的数据根据语音模型进行信噪比的计算，并且将结果填充成[snr, id]格式的Record。

2. 执行如下命令，创建外部表。

```
CREATE EXTERNAL TABLE IF NOT EXISTS speech_sentence_snr_external
(
sentence_snr double,
id string
)
STORED BY 'com.aliyun.odps.udf.example.speech.SpeechStorageHandler'
WITH SERDEPROPERTIES (
'mlfFileName'='sm_random_5_utterance.text.label' ,
'speechSampleRateInKHz' = '16'
)
LOCATION    'oss://<*your-id*>:<*your-secret-key*>@oss-cn-shanghai-internal.aliyuncs.com/
oss-odps-test/dev/SpeechSentenceTest/'
USING 'odps-udf-example.jar,sm_random_5_utterance.text.label';
```

3. 执行如下命令，从OSS读取数据进行处理。

```
SELECT sentence_snr, id
```

```
FROM speech_sentence_snr_external
WHERE sentence_snr > 10.0;
```

4. 处理完成，输出如下结果。

```
-----------------------------------------------------------
| sentence_snr | id |
-----------------------------------------------------------
| 34.4703 | J310209090013_H02_K03_042 |
-----------------------------------------------------------
| 31.3905 | tsh148_seg_2_3013_3_6_48_80bd359827e24dd7_0 |
-----------------------------------------------------------
| 35.4774 | tsh148_seg_3013_1_31_11_9d7c87aef9f3e559_0 |
-----------------------------------------------------------
| 16.0462 | tsh148_seg_3013_2_29_49_f4cb0990a6b4060c_0 |
-----------------------------------------------------------
| 14.5568 | tsh_148_3013_5_13_47_3d5008d792408f81_0 |
-----------------------------------------------------------
```

说明： 通过自定义extractor，用户在SQL语句上即可分布式地处理多个OSS上语音数据文件。同样的，使用类似的方法，用户可以方便的利用MaxCompute的大规模计算能力，完成对图像、视频等各种类型非结构化数据的处理。

26.12.1.4.2 通过STS访问OSS

在前面的章节中介绍创建外部表的语句时，传入了明文accessId和secretAccessKey。如果考虑信息安全风险，需要部署STS服务，使用MaxCompute提供的更为安全的方式来访问OSS。

MaxCompute结合了阿里云的访问控制服务（RAM）和令牌服务（STS）来解决账号的安全问题，具体操作示例如下。

1. 在RAM中授权MaxCompute访问OSS的权限。登录RAM控制台，创建角色**AliyunODPSDefaultRole**，并将策略内容设置为如下所示。

```
{
"Statement": [
{
"Action": "sts:AssumeRole",
"Effect": "Allow",
"Principal": {
"Service": [
"odps.aliyuncs.com"
]
}
}
],
"Version": "1"
}
```

2. 编辑该角色的授权策略，将权限AliyunODPSRolePolicy授权给该角色。

3. 授权完成，进行后续相关操作，示例如下。

```
CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_csv_external
(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
STORED BY 'com.aliyun.odps.CsvStorageHandler'
LOCATION    'oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/Demo/SampleData/CSV/AmbulanceData/';
```



说明: 此时可以看到，在进行授权后，不再需要在LOCATION传入明文账号信息。

26.12.1.5 数据的分区

在前面的章节中，一个外部表关联的数据通过LOCATION上指定的OSS**目录**来实现。而在处理的时候，MaxCompute读取**目录**下面的所有数据， 包括子目录中的所有文件。对于随着时间不断积累的数据目录，由于数据量过大，进行这样的全目录扫描，可能会带来不必要的额外IO以及数据处理时间。 解决这个问题通常有两种做法。

- 减少访问数据量：用户自己负责对数据存放地址做好规划，同时考虑使用多个EXTERNAL TABLE来描述不同部分的数据，让每个EXTERNALTABLE的LOCATION指向数据的一个子集。
- 数据分区：EXTERNAL TABLE与内部表一样， 支持分区表的功能，用户可以利用分区功能对数据做系统化的管理。

本章节将主要介绍EXTERNAL TABLE的分区功能。

26.12.1.5.1 分区数据在OSS上的标准组织方式和路径格式

与MaxCompute内部表不同，对于存放在外部存储上（如OSS）上面的数据，MaxCompute不拥有数据的管理权，因此用户如果需要在系统中使用分区表功能，在OSS上的数据文件的存放路径需要符合一定的格式，路径格式如下所示。

```
partitionKey1=value1\partitionKey2=value2\...
```

相关示例如下。

1. 假设用户每天的LOG文件存放在OSS上面，并希望能在通过MaxCompute处理的时候，能够按照粒度为**天**来访问一部分数据。假设这些LOG文件是CSV的格式（复杂自定义格式用法也类似），此时可以使用如下的分区外部表来定义数据。

```
CREATE EXTERNAL TABLE log_table_external (
click STRING,
ip STRING,
url STRING,
)
PARTITIONED BY (
year STRING,
month STRING,
day STRING
)
STORED BY 'com.aliyun.odps.CsvStorageHandler'
LOCATION  'oss://<ak_id>:<ak_key>@oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/log_data/';
```

> **说明:** 与前面章节中的外部表示例不同，在上述的三层分区示例中，用户在定义EXTERNAL TABLE的时候，通过PARTITIONED BY的语法，指定该外部表为分区表，分区的key分别是year，month和day。

2. 为了让类似上述的分区能够生效，在OSS上面存储数据的时候需要遵循前面提到的路径格式。一个有效的路径存储layout的示例如下。

```
osscmd ls oss://oss-odps-test/log_data/
2017-01-14 08:03:35 128MB Standard oss://oss-odps-test/log_data/year=2016/month=06/day=01/logfile
2017-01-14 08:04:12 127MB Standard oss://oss-odps-test/log_data/year=2016/month=06/day=01/logfile.1
2017-01-14 08:05:02 118MB Standard oss://oss-odps-test/log_data/year=2016/month=06/day=02/logfile
2017-01-14 08:06:45 123MB Standard oss://oss-odps-test/log_data/year=2016/month=07/day=10/logfile
2017-01-14 08:07:11 115MB Standard oss://oss-odps-test/log_data/year=2016/month=08/day=08/logfile
...
```

> **说明:** 如果用户准备了离线数据，即通过osscmd或者其他OSS工具自行将离线数据上载到OSS存储服务上，此时数据路径格式是由用户决定的。如果想要EXTERNAL TABLE的分区表功能正常工作，推荐用户上载数据的时候遵循如上路径格式。

3. 在如上的前提下，通过ALTER TABLE ADD PARTITION语句，将分区信息引入MaxCompute。对应的DDL语句示例如下。

```
ALTER TABLE log_table_external ADD PARTITION (year = '2016', month = '06', day = '01')
ALTER TABLE log_table_external ADD PARTITION (year = '2016', month = '06', day = '02')
ALTER TABLE log_table_external ADD PARTITION (year = '2016', month = '07', day = '10')
ALTER TABLE log_table_external ADD PARTITION (year = '2016', month = '08', day = '08')
```

```
...
```

4. 在数据准备好并且PARTITION信息引入系统之后，可以通过SQL语句对OSS上的外部表数据的分区进行操作，相关示例如下。

   - 假设用户只想分析2016年6月1号当天，有多少不同的IP出现在LOG里面，可以通过如下命令实现：

     ```
     SELECT count(distinct(ip)) FROM log_table_external
     WHERE year = '2016' AND month = '06' AND day = '01';
     ```

     说明: 此时，对于**log_table_external**这个外部表对应的目录，将只访问*log_data/year=2016/month=06/day=01*子目录下的文件（logfile和logfile.1），而不会对整个*log_data/*目录作全量数据扫描，避免了大量无用的IO操作。

   - 同样，假设用户只希望对2016年下半年的数据做分析，则可以通过如下命令实现：

     ```
     SELECT count(distinct(ip)) FROM log_table_external
     WHERE year = '2016' AND month > '06';
     ```

     说明: 此时，只访问OSS上面存储的下半年的LOG。

26.12.1.5.2 分区数据在OSS上的自定义路径

如果用户有事先保存在OSS上面的历史数据，但是该数据没有按照partitionKey1=value1\partitionKey2=value2\...的路径格式存放，此时同样可以通过MaxCompute的分区方式进行访问计算，MaxCompute同样提供了通过自定义路径来引入partition的方法。

相关示例如下。

1. 假设用户的数据路径上只有简单的分区值（而无分区key信息），此时数据的路径存储layout的示例如下。

   ```
   osscmd ls oss://oss-odps-test/log_data_customized/
   2017-01-14 08:03:35 128MB Standard oss://oss-odps-test/log_data_customized/2016/06/01/logfile
   2017-01-14 08:04:12 127MB Standard oss://oss-odps-test/log_data_customized/2016/06/01/logfile.1
   2017-01-14 08:05:02 118MB Standard oss://oss-odps-test/log_data_customized/2016/06/02/logfile
   2017-01-14 08:06:45 123MB Standard oss://oss-odps-test/log_data_customized/2016/07/10/logfile
   2017-01-14 08:07:11 115MB Standard oss://oss-odps-test/log_data_customized/2016/08/08/logfile
   ```

```
...
```

2. 此时要绑定不同的子目录到不同的分区上，可以通过类似如下自定义分区路径的命令实现。

```
ALTER TABLE log_table_external ADD PARTITION (year = '2016', month = '06', day = '01')
LOCATION  'oss://<ak_id>:<ak_key>@oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/
log_data_customized/2016/06/01/';
```



说明: 在ADD PARTITION的时候增加了LOCATION信息，从而实现自定义分区数据路径后，即使数据存放不符合推荐的partitionKey1=value1\partitionKey2=value2\...格式，也能正确的实现对子目录数据的分区访问。

26.12.1.5.3 完全自定义的非分区数据子集访问

一些特殊情况下，用户可能会有访问某个OSS路径上的任意文件子集的需求，而这个文件子集中的文件在路径格式上 没有明显的规律性。在这方面MaxCompute非结构化数据处理框架也提供了相应的支持，此处不再展开做详细介绍。

如果用户对于相关高阶的特殊用法，有相关的具体需求，可以联系MaxCompute技术团队获取相关支持。

26.12.1.6 非结构化数据的输出

相关的操作示例如下。

26.12.1.6.1 创建External Table

与读取OSS数据类似，对OSS数据进行写操作，同样是先通过CREATE EXTERNAL TABLE语句创建出一个外部表，再通过标准MaxCompute SQL的INSERT INTO/OVERWRITE等语句实现的。此处使用MaxCompute内置的TsvStorageHandler为例进行操作说明。

```
DROP TABLE IF EXISTS tpch_lineitem_tsv_external;
CREATE EXTERNAL TABLE IF NOT EXISTS tpch_lineitem_tsv_external
(
orderkey BIGINT,
suppkey BIGINT,
discount DOUBLE,
tax DOUBLE,
shipdate STRING,
linestatus STRING,
shipmode STRING,
comment STRING
)
STORED BY 'com.aliyun.odps.TsvStorageHandler'
```

```
LOCATION     'oss://<AK_id>:<AK_secret>@oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/
tsv_output_folder/' ;
```

说明:

上述的DDL语句建立了一个外部表tpch_lineitem_tsv_external，并将两个维度的外部数据信息关联到这个外部表上。

- 数据存储介质：LOCATION将一个OSS上的地址关联到外部表上，对这个外部表进行的读写操作都会反映到这个OSS地址上。
- 数据存储格式：StorageHandler用来表明对数据的读写操作方式，此处使用了MaxCompute内置的com.aliyun.odps.TsvStorageHandler，用户可以使用这个由系统自带的实现来读取和写出TSV文件。 同时用户也可以通过MaxCompute的SDK来自定义StorageHandler，此处不再详细介绍。

26.12.1.6.2 通过对External Table的INSERT操作实现TSV文本文件的写出

将OSS数据通过External Table关联上之后，可以通过对External Table执行标准的SQL INSERT OVERWRITE/INSERT INTO操作进行OSS文件的写出。 具体输出数据的来源有**MaxCompute的内部表**和**通过External Table引入MaxCompute计算体系的外部数据**两种。

说明:

- MaxCompute的内部表： 可以通过对外部表的INSERT操作实现MaxCompute内部表数据到外部存储介质的写出。
- 通过External Table引入MaxCompute计算体系的外部数据： 通过External Table将外部数据引入MaxCompute进行计算，然后再存储到不同的外部存储地址，或者是不同的外部存储介质（例如将TableStore数据经由MaxCompute导出到OSS）。

此处假设用户已经有一个名为tpch_lineitem的MaxCompute内部表，并需要将其中的一部分数据以TSV格式导出到OSS上面。可以在完成上述的创建External Table操作之后，使用类似如下的INSERT OVERWRITE操作进行实现。

```
INSERT OVERWRITE TABLE tpch_lineitem_tsv_external
SELECT l_orderkey, l_suppkey, l_discount, l_tax, l_shipdate, l_linestatus,
l_shipmode, l_comment
FROM tpch_lineitem
```

```
WHERE l_discount = 0.07 and l_tax = 0.01;
```

上述操作是从内部的tpch_lineitem表中，在符合l_discount = 0.07 并 l_tax = 0.01的行中选出8个列（对应tpch_lineitem_tsv_external这个外部表的schema）按照TSV的格式写到OSS上。在上述操作成功完成后，可以看到在OSS上产生的对应TSV数据文件。

注意:

从MaxCompute向OSS上输出数据时，存在特有的文件结构。

- 通过MaxCompute对一个OSS地址，使用INSERT INTO/OVERWRITE外部表做写出操作时，所有的数据将在指定的LOCATION下的.odps文件夹产生。
- .odps文件夹中的.meta文件是MaxCompute额外写出的宏数据文件，用于记录当前文件夹中有效的数据。 正常情况下，如果INSERT操作成功，可以认为当前文件夹的所有数据均是有效数据。只有在存在作业失败的情况下需要对这个宏数据进行解析。
- 对于中途失败或者被结束进程的作业，针对INSERT OVERWRITE操作，再次完成一次成功操作即可避免对.meta文件这个宏数据的解析。
- 对于高级用户，如果一定需要解析.meta文件，可以联系MaxCompute技术团队获取相关支持。

对于MaxCompute内置的TSV/CSV处理来说，产生的文件数目与对应SQL stage的并发度是相同的。如果需要控制产生文件的数目，可以配合MaxCompute的各种灵活语义和配置来实现。例如在上述示例中，如果需要强制产生一个TSV文件，可以在INSERT OVERWRITE操作的最后加上一个DISTRIBUTE BY l_discount，即可在最后插入仅有一个Reducer的Reduce stage，实现只输出一个TSV文件。

26.12.1.6.3 通过对External Table的INSERT操作实现非结构化文件的写出

同样对于数据的输出，MaxCompute还提供了Outputer接口，允许用户的数据被通过OutputStream写出成自由的非结构化数据文件。此处不再展开做详细介绍。

如果用户有相关的具体需求，可以联系MaxCompute技术团队获取相关支持。

26.12.1.6.4 以MaxCompute为计算介质，实现不同存储介质之间的数据转移

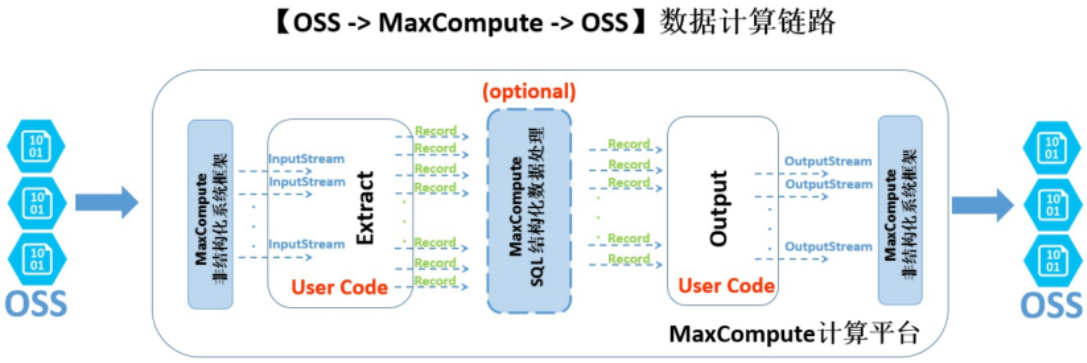
External Table作为一个MaxCompute与外部存储介质的切入点，提供了对多种外部数据（包括OSS，TableStore等）的读取和写出的功能。依赖于接入点，我们可以实现各种各样的数据计算/存储链路，例如：

1. 读取External Table A关联的OSS数据，在MaxCompute上做复杂计算处理，并输出到External Table B关联的OSS地址。
2. 读取External Table A关联的TableStore数据，在MaxCompute上做复杂计算处理，并输出到External Table B关联的OSS地址。

> **说明:** 上述示例与数据源为MaxCompute内部表的场景， 唯一的区别在于SELECT的来源变成External table，而不是MaxCompute内置表。

相关示例如下。

以MaxCompute为中心计算平台，用户实现数据从OSS输入，再输出到OSS（例如另一个location，或者不同的OSS账号）的整个流程，如下图所示。



从上图可以看出，从数据的流动和处理逻辑上理解，用户可以简单地把非结构化处理框架理解成在MaxCompute计算平台两端有机耦合的数据导入（Ingres）以及导出（Egress），相关步骤如下。

1. 外部的（OSS）数据经过非结构化框架转换，会使用Java用户容易理解的InputStream类提供给自定义代码接口。用户自实现Extract逻辑只需要负责对输入的InputStream做读取/解析/转化/计算，最终返回MaxCompute计算平台通用的Record格式。
2. 返回的Record可以自由的参与MaxCompute的SQL逻辑运算，这一部分计算是基于MaxCompute内置的强大结构化SQL运算引擎，并可能产生新的Record。

3. 运算过后的Record中再传递给用户自定义的Output逻辑，用户可以进行进一步的计算转换，并最终将Record里面需要输出的信息通过系统提供的OutputStream输出，由系统负责写到OSS。



注意: 上述的所有的步骤都是可以由用户根据需要进行自由的选择与拼接的。

26.12.2 如何在MaxCompute上访问并处理TableStore（OTS）数据

26.12.2.1 前言

MaxCompute作为阿里云大数据平台的核心计算组件，承担了集团内外大部分的分布式计算需求。而MaxCompute SQL作为分布式数据处理的主要入口，为快速方便处理/存储EB级别的离线数据提供了强有力的支持。随着大数据业务的不断扩展，新的数据使用场景在不断产生，在这样的背景下，MaxCompute计算框架也在不断的演化，原来主要面对内部特殊格式数据的强大计算能力，也正在一步步的开放给不同的外部数据。在这里我们将详细介绍如何将来自TableStore(OTS)的数据纳入MaxCompute上的计算生态，实现多种数据源之间的无缝连接。

对于在线服务等应用场景，NoSQL KV Store（e.g.，BigTable，HBase）相比传统数据库，具有schema灵活，易扩展，实时性强等优点。阿里云TableStore(OTS) 是在飞天分布式系统之上实现的大规模的NoSQL数据存储服务，提供海量KV数据的存储和实时访问。在集团内部各BU以及外部阿里云生态圈中具有广泛的应用，尤其是TableStore在行级别上的实时更新和可覆盖性写入等特性，相对于MaxCompute内置表append-only批量操作，提供了一个很好的补充。但是TableStore作为一个偏存储的服务，对于存储在其上的海量数据，在大规模批量并行处理的计算能力上有所欠缺。基于这样的背景，打通MaxCompute的计算和TableStore存储之间的数据链路就变的至关重要。

下面将通过几个简单的示例，演示如何在MaxCompute上访问并处理TableStore（OTS）数据。

26.12.2.2 MaxCompute对TableStore数据进行读取和计算

相关的操作示例如下。

26.12.2.2.1 使用前提和假设

此处将假设用户对于TableStore已经有了一定的了解。如果对于TableStore不熟悉或者对于整个KV table的概念比较陌生，可以通过*TableStore*文档先了解一些基本概念（例如主键，分区键，属性列等）。

26.12.2.2.2 创建External Table

MaxCompute通过EXTERNAL TABLE的方式对接TableStore的数据：用户通过一个CREATE EXTERNAL TABLE的DDL语句，把对TableStore表数据的描述引入到MaxCompute的meta系统内部后，即可如同使用一个普通TABLE一样实现对TableStore数据的处理。

示例如下：

```
DROP TABLE IF EXISTS ots_table_external;
CREATE EXTERNAL TABLE IF NOT EXISTS ots_table_external
(
odps_orderkey bigint,
odps_orderdate string,
odps_custkey bigint,
odps_orderstatus string,
odps_totalprice double
)
STORED BY 'com.aliyun.odps.TableStoreStorageHandler'
--com.aliyun.odps.TableStoreStorageHandler是MaxCompute内置的处理TableStore数据
的StorageHandler，定义了MaxCompute和TableStore的交互，相关逻辑由MaxCompute实现。
WITH SERDEPROPERTIES (
--SERDEPROPERITES可以理解成提供参数选项的接口，在使
用TableStoreStorageHandler时，有两个必须指定的选项，分别是下方
的tablestore.columns.mapping和tablestore.table.name。
'tablestore.columns.mapping'=':o_orderkey, :o_orderdate, o_custkey,
o_orderstatus，o_totalprice',
--tablestore.columns.mapping选项：必需选项，用来描述需要通过MaxCompute来访问
的TableStore表的列，包括主键和属性列。其中以:开头的用来表示TableStore主键，例如本示例
中的:o_orderkey和:o_orderdate。其他的均为属性列。TableStore支持的主键数最少为1个，最多
为4个，主键类型为bigint或string，其中第一个主键为分区键。在指定映射的时候，用户必须提
供指定TableStore表的所有主键，对于属性列无特殊要求，可以只提供需要通过MaxCompute来
访问的属性列。
'tablestore.table.name'='ots_tpch_orders'
--tablestore.table.name选项：必需选项，用来描述需要通过MaxCompute来访问的TableStore表
的表名。如果指定的TableStore表名错误（不存在），则会报错，MaxCompute不会主动创
建TableStore表。
)
LOCATION 'tablestore://<*your AK id*>:<*your AK secret key*>@odps-ots-dev.cn-
hangzhou.ots.aliyuncs.com';
--LOCATION clause用来指定TableStore的具体信息，包括instance名字，endpoint等。需要注
意，在本示例中采用了明文AK，如果专有云集群上部署有STS服务，可以通过本章节后面
的RAM/STS鉴权方式进行访问。
```

说明: 上述示例将一个TableStore表映射到了MaxCompute的一个External Table上。在此基础上，后继对TableStore数据的操作可以直接通过External Table进行。

26.12.2.2.3 通过External Table访问TableStore数据进行计算

在参考上述示例创建出External Table之后，TableStore的数据被引入到了MaxCompute的生态中。此时，用户即可以通过正常的MaxCompute SQL语法来访问TableStore数据。

示例如下：

```
SELECT odps_orderkey, odps_orderdate, SUM(odps_totalprice) AS totalprice
FROM ots_table_external
WHERE odps_orderkey > 5000 AND odps_orderdate >20170725 AND odps_orderdate
 <20170910
GROUP BY odps_orderkey, odps_orderdate
HAVING totalprice> 2000;
```

说明: 在本示例中，直接使用的是用户所熟悉的MaxCompute SQL语法，访问TableStore的所有细节由MaxCompute内部处理。

如果用户需要对一份数据做多次计算，相较于每次从TableStore远程读数据，更高效的做法是一次性把需要的数据导入到MaxCompute内部，成为一个MaxCompute（内部）表。

示例如下：

```
CREATE TABLE internal_orders AS
SELECT odps_orderkey, odps_orderdate, odps_custkey, odps_totalprice
FROM ots_table_external
WHERE odps_orderkey > 5000 ;
```

说明: 此时，internal_orders即为一个用户熟悉的MaxCompute表，并且拥有所有MaxCompute内部表的特性，包括高效的压缩列存储数据格式，完整的meta等。同时因为存储在MaxCompute内部，访问速度比访问外部的TableStore更快，尤其适用于需要进行多次计算的热点数据。

26.12.2.2.4 使用STS/RAM的方式访问TableStore数据

在部署有RAM/STS服务的专有云集群，MaxCompute可以使用其来解决账号的安全问题，而无需明文提供AK。具体操作示例如下。

1. 在RAM中授权MaxCompute访问OSS的权限。创建角色**AliyunODPSDefaultRole**，并将策略内容设置为如下所示。

```
{
"Statement": [
{
"Action": "sts:AssumeRole",
"Effect": "Allow",
"Principal": {
"Service": [
"odps.aliyuncs.com"
]
}
}
],
"Version": "1"
```

```
}
```

2. 编辑该角色的授权策略，将权限AliyunODPSRolePolicy授权给该角色。
3. 授权完成，进行后续相关操作，示例如下。

```
DROP TABLE IF EXISTS ots_table_external;
CREATE EXTERNAL TABLE IF NOT EXISTS ots_table_external
(
odps_orderkey bigint,
odps_orderdate string,
odps_custkey bigint,
odps_orderstatus string,
odps_totalprice double
)
STORED BY 'com.aliyun.odps.TableStoreStorageHandler'
WITH SERDEPROPERTIES (
'tablestore.columns.mapping'=':o_orderkey, :o_orderdate, o_custkey,
o_orderstatus，o_totalprice',
'tablestore.table.name'='ots_tpch_orders'
)
LOCATION 'tablestore://odps-ots-dev.cn-hangzhou.ots.aliyuncs.com'
```



说明: 此时可以看到，在进行授权后，不再需要在LOCATION传入明文信息。

26.12.2.3 数据从MaxCompute写出到TableStore

打通MaxCompute和TableStore的数据生态，除了将TableStore作为批量数据处理的数据来源以外，另一个重要的场景是将MaxCompute的数据处理结果输出到TableStore，利用TableStore可实时更新和可单行覆盖等特点，迅速的将离线计算的结果反馈给在线应用。这种对TableStore的数据输出可以使用MaxCompute SQL的INSERT OVERWRITE来实现。



说明: 由于MaxCompute不会主动创建外部的TableStore表，因此对TableStore表进行数据输出之前，必须保证该表已经在TableStore上创建过（否则将报错）。

假设用户已经创建了上一章节示例中的**ots_table_external**这个外部表来打通MaxCompute与TableStoreb数据表**ots_tpch_orders**的链路， 并且在MaxCompute内部存储有一个名为**internal_orders**的数据。此时，用户希望对**internal_orders**中的数据进行一定处理后写回到TableStore上，可以通过对外部表做INSERT OVERWITE TABLE操作进行实现，命令示例如下。

```
INSERT OVERWRITE TABLE ots_table_external
SELECT odps_orderkey, odps_orderdate, odps_custkey, CONCAT(odps_custkey,
'SHIPPED'), CEIL(odps_totalprice)
FROM internal_orders;
```



说明: 对于TableStore这种KV数据的NoSQL存储介质，从MaxCompute的输出将只影响相对应主键所在的行，并且在这些TabeleStore行上面，同样只会更新在创建External Table时指定的属性列，不会修改未在External Table里面出现的数据列。

26.13 安全指南

26.13.1 目标用户

本文档主要面向MaxCompute项目空间所有者（Owner）、管理员以及对MaxCompute多租户数据安全体系感兴趣的用户。MaxCompute多租户数据安全体系主要包括如下内容：

- 用户认证。
- 项目空间的用户与授权管理。
- 跨项目空间的资源分享。
- 项目空间的数据保护。

26.13.2 快速开始

- **添加用户并授权**

  场景描述：Jack是项目空间prj1的管理员，一个新加入的项目组成员Alice （已拥有云账号：alice@aliyun.com）申请加入项目空间prj1，并申请如下权限：查看Table列表，提交作业，创建表。

  操作如下：（由项目空间管理员进行操作）

  ```
  use prj1
  add user aliyun$alice@aliyun.com
  -- 添加用户。
  grant List, CreateTable, CreateInstance on project prj1 to user aliyun$alice@aliyun.com
  -- 使用grant语句对用户授权。
  ```

- **添加角色并通过ACL授权**

  场景描述：Jack是项目空间prj1的管理员，有三个新加入的项目组成员：
  Alice，Bob，Charlie，他们的角色是数据审查员。他们要申请如下权限：查看Table列表，提交作业，读取表userprofile。

  对于这个场景的授权，项目空间管理员可以使用基于对象的ACL授权机制来完成。

  操作如下：

  ```
  use prj1
  add user aliyun$alice@aliyun.com
  -- 添加用户。
  add user aliyun$bob@aliyun.com
  add user aliyun$charlie@aliyun.com
  create role tableviewer
  -- 创建角色。
  grant List, CreateInstance on project prj1 to role tableviewer
  -- 对角色赋权。
  ```

```
grant Describe, Select on table userprofile to role tableviewer
grant tableviewer to aliyun$alice@aliyun.com
-- 对用户赋予角色tableviewer。
grant tableviewer to aliyun$bob@aliyun.com
grant tableviewer to aliyun$charlie@aliyun.com
```

- **资源打包分享**

  场景描述：Jack是项目空间prj1的管理员。John是项目空间prj2的管理员。由于业务需要，Jack希望将其项目空间prj1中的某些资源（例如datamining.jar及sampletable表）分享给John的项目空间prj2。如果项目空间prj2的用户Bob需要访问这些资源，那么prj2管理员可以通过ACL或Policy自主授权，无需Jack参与。

  操作如下：

  1. 项目空间prj1的管理员Jack在项目空间prj1中创建资源包（package）。

     ```
     use prj1
     create package datamining
     -- 创建一个package。
     add resource datamining.jar to package datamining
     -- 添加资源到package。
     add table sampletable to package datamining
     -- 添加table到package。
     allow project prj2 to install package datamining
     -- 将package分享给项目空间prj2。
     ```

  2. 项目空间prj2管理员Bob在项目空间prj2中安装package。

     ```
     use prj2
     install package prj1.datamining
     -- 安装一个package。
     describe package prj1.datamining
     -- 查看package中的资源列表。
     ```

  3. Bob对package进行自主授权。

     ```
     use prj2
     grant Read on package prj1.datamining to user aliyun$bob@aliyun.com
     -- 通过ACL授权Bob使用package。
     ```

     说明： 关于跨项目空间资源分享的详细介绍请参考：跨项目空间的资源分享。

- **设置项目保护模式**

  项目空间的数据保护设置

  场景描述：Jack是项目空间prj1的管理员。该项目空间有很多敏感数据，例如用户身份号码和购物记录。而且还有很多具有自主知识产权的数据挖掘算法。Jack希望能将项目空间中的这些敏感

数据和算法保护好，项目中用户只能在项目空间中访问，数据只能在项目空间内流动，不允许流出到项目空间之外。

操作如下：

```
use prj1
set ProjectProtection=true
-- 开启项目空间的数据保护机制。
```

一旦当项目空间开启项目空间的数据保护机制后，无法将项目空间中的数据转移到项目空间之外，所有的数据只能在项目空间内部流动。但是在某些情况下，可能由于业务需要，用户Alice需要将某些数据表导出到项目空间之外，并且也经过空间管理员的审核通过。针对这类情况，MaxCompute提供了两种机制来支持受保护项目空间的数据流出。

方法1：设置ExceptionPolicy。详情请参考：开启数据保护后的数据流出方法。

1. 创建Policy文件。例如创建一个*/tmp/exception_policy.txt*，它只允许Alice 使用SQL任务将表t1从项目空间prj1导出。policy内容如下：

```
{
"Version": "1",
"Statement": [{
"Effect":"Allow",
"Principal":"ALIYUN$alice@aliyun.com",
"Action":["odps:Describe","odps:Select"],
"Resource":"acs:odps:*:projects/prj1/tables/t1",
"Condition":{
"StringEquals": { "odps:TaskType":"SQL"
}
}
}]
```

2. 设置Exception policy。

```
use prj1
-- 开启项目空间的数据保护机制，并设置数据导出的例外。
set ProjectProtection=true with exception /tmp/exception_policy.txt
```

> **说明:** 设置Exception policy时，应确保导出人（Principal）对数据资源（Resource）不具有更新（Update）或重建同名对象（Drop、CreateTable）权限，以防止TOC2TOU（time-of-check to time-of-use）数据泄露。

方法2：设置TrustedProject。将prj2设置为prj1的可信项目空间，设置后将允许prj1中的所有数据流出到prj2。详情请参考：开启数据保护后的数据流出方法。

```
use prj1
```

```
add trustedproject prj2
```

说明： 基于Package的资源分享机制和项目空间的数据保护机制是正交的两种安全机制。

MaxCompute规定：资源分享拥有更高的优先级。即在一个受保护的项目空间中，如果一个对象是通过Package机制分享给其他项目空间，那么跨项目空间访问该对象时将不受ProjectProtection规则的限制。

26.13.3 用户认证

用户认证（Authentication）的主要功能是检查请求（Request）发送者的真实身份。一般包括如下两个方面：

- 正确验证消息发送方的真实身份。
- 正确验证接收到的消息在途中是否被篡改。

26.13.4 项目空间的用户及授权管理

26.13.4.1 概要

项目空间（Project）是MaxCompute实现多租户体系的基础，是用户管理数据和计算的基本单位。当用户申请创建一个项目空间之后，该用户就是这个空间的所有者（Owner）。也就是说，这个项目空间内的所有对象（eg、表、实例、资源、UDF 等）都属于该用户。除了Owner之外，任何人都无权访问此项目空间内的对象，除非有Owner的授权许可。

本章节主要介绍项目空间的用户、角色及授权管理。其主要适用于如下场景：假设用户Alice是项目空间test_project的Owner，如果有其他人需要申请访问项目空间test_project的资源，并且这个申请人是属于Alice的项目团队。那么可以采用本章节介绍的方法来进行用户及授权管理。如果申请人不属于Alice的项目团队，建议使用跨项目空间的资源分享功能，详情请参考：跨项目空间的资源分享。

26.13.4.2 用户管理

当项目空间的Owner Alice决定对另一个用户授权时，Alice需要先将该用户添加到自己的项目空间中来。只有添加到项目空间中的用户才能够被授权。

添加用户的命令如下：

```
add user <full_username>
```

```
-- 在项目空间中添加用户。
```

当一个用户离开此项目团队时，Alice需要将该用户从项目空间中移除。用户一旦从项目空间中被移除，该用户将不再拥有任何访问项目空间资源的权限。

移除用户的命令如下：

```
remove user <full_username>
-- 在项目空间中移除用户。
```

说明:

- 当一个用户被移除后，该用户不再拥有访问该项目空间资源的任何权限。
- 移除一个用户之前，如果该用户已被赋予某些角色，则需要先撤销该用户的所有角色。关于角色的介绍请参考：角色管理。
- 当一个用户被移除后，与该用户有关的ACL授权仍然会被保留，但Role级别的Policy授权会失效，项目空间级别的会被保留。一旦该用户以后被再添加到该项目空间时， 该用户的历史的ACL授权访问权限将被重新激活。
- MaxCompute目前不支持在项目空间中彻底移除一个用户及其所有权限数据。

26.13.4.3 角色管理

角色（Role）是一组访问权限的集合。当需要对一组用户赋予相同的权限时，可以使用角色来授权。基于角色的授权可以大大简化授权流程，降低授权管理成本。当需要对用户授权时，应当优先考虑是否应该使用角色来完成。

每一个项目空间在创建时，会自动创建一个admin的角色，并且为该角色授予了确定的权限：能访问项目空间内的所有对象，能进行用户与角色管理，能对用户或角色进行授权。与项目空间Owner相比，admin角色不能将admin权限指派给用户，不能设定项目空间的安全配置，不能修改项目空间的鉴权模型。Admin角色所对应的权限不能被修改。

角色管理的相关命令如下：

```
create role <rolename>
-- 创建角色。
drop role <rolename>
-- 删除角色。
grant <rolename> to <username>
-- 给用户指派某种角色。
revoke <rolename> from <username>
```

```
-- 撤销角色指派。
```



说明: 删除一个角色时，MaxCompute会检查该角色内是否还存在其他用户。若存在，则删除该角色失败。只有在该角色的所有用户都被撤销时，删除角色才会成功。

26.13.4.4 ACL授权

授权操作一般涉及到三个要素：主体（Subject）、客体（Object）和操作（Action）。

在MaxCompute中，主体是指用户，客体是指项目空间中的各种类型对象，操作则与特定对象类型有关，不同类型的对象支持的操作也不尽相同。

MaxCompute项目空间支持如下的对象类型及操作：

客体（Object）	**操作**（Action）	**说明**
Project	Read	查看项目空间自身（不包括项目空间的任何对象）的信息，如CreateTime等。
Project	Write	更新项目空间自身（不包括项目空间的任何对象）的信息，如Comments。
Project	List	查看项目空间所有类型的对象列表。
Project	CreateTable	在项目空间中创建Table。
Project	CreateInstance	在项目空间中创建Instance。
Project	CreateFunction	在项目空间中创建Function。
Project	CreateResource	在项目空间中创建Resource。
Project	CreateJob	在项目空间中创建Job。
Project	CreateVolume	在项目空间中创建Volume。
Project	All	具备上述所有权限。
Table	Describe	读取Table的元信息。
Table	Select	读取Table的数据。
Table	Alter	修改Table的元信息，添加删除分区。
Table	Update	覆盖或添加Table的数据。
Table	Drop	删除Table。
Table	All	具备上述所有权限。
Function	Read	读取，及执行权限。

客体（Object）	操作（Action）	说明
Function	Write	更新
Function	Delete	删除
Function	All	具备上述所有权限。
Resource，Instance，Job，Volume	Read	读取
Resource，Instance，Job，Volume	Write	更新
Resource，Instance，Job，Volume	Delete	删除
Resource，Instance，Job，Volume	All	具备上述所有权限。

说明: 上述权限描述中，Project类型对象的CreateTable权限操作，Table类型对象的Select、Alter、Update、Drop权限操作需要与Project对象的CreateInstance权限操作配合使用。单独使用上述几种权限而没有指派CreateInstance权限是无法完成对应操作的。

在添加用户或创建角色之后，需要对用户或角色进行授权。MaxCompute ACL授权是一种基于对象的授权，通过授权的权限数据（即访问控制列表， Access Control List） 被看做是该对象的一种子资源，只有当对象已经存在时，才能进行授权操作。当对象被删除时，通过授权的权限数据会被自动删除。

MaxCompute ACL支持的授权方法是采用类似SQL92定义的GRANT/REVOKE命令进行授权。通过对应的授权命令来完成对已存在的项目空间对象的授权或撤销授权。

命令格式如下：

```
grant actions on object to subject
revoke actions on object from subject
actions ::= action_item1, action_item2, ...
object ::= project project_name | table schema_name | instance inst_name | function func_name
 | resource res_name
subject ::= user full_username | role role_name
```

说明:

MaxCompute ACL的授权命令并不支持[WITH GRANT OPTION]授权参数。即当用户A授权用户B访问某个对象时，用户B无法将权限进一步授权给用户C。因此，所有的授权操作都必须由具有如下三种身份之一的用户完成：

- 项目空间Owner。

- 项目空间中拥有admin角色的用户。
- 项目空间中对象创建者。

使用ACL授权的示例如下：

场景说明：云账号用户alice@aliyun.com和bob@aliyun.com是新加入到项目空间test_project的成员。在test_project中，该用户需要提交作业、创建数据表、查看项目空间已存在的对象。管理员执行的授权操作如下：

```
use test_project
-- 打开项目空间。
security
add user aliyun$alice@aliyun.com
-- 添加用户。
add user aliyun$bob@aliyun.com
-- 添加用户。
create role worker
-- 创建角色。
grant worker TO aliyun$alice@aliyun.com
-- 角色指派。
grant worker TO aliyun$bob@aliyun.com
-- 角色指派。
grant CreateInstance, CreateResource, CreateFunction, CreateTable, List ON PROJECT
 test_project TO ROLE worke r
-- 对角色授权。
```

26.13.4.5 权限查看

MaxCompute支持从多种维度查看权限，具体包括：查看指定用户的权限，查看指定角色的权限，以及查看指定对象的授权列表。

查看指定用户的权限

```
show grants
-- 查看当前用户自己的访问权限。
show grants for <username>
-- 查看指定用户的访问权限，仅Project Owner和Admin可以操作。
```

查看指定角色的权限

```
describe role <rolename>
-- 查看指定角色的访问权限角色指派。
```

查看指定对象的授权列表

```
show acl for <objectName> [on type <objectType>]
-- 查看指定对象上的用户和角色授权列表。
```



说明: 当省略[on type <objectType>]时，默认的type为Table。

在展现用户权限或角色权限时，MaxCompute使用了如下的标记字符：A、C、D、G，它们的含义如下：

- A：表示Allow，即允许访问。
- D：表示Deny，即拒绝访问。
- C：表示with Condition，即为带条件的授权，只出现在policy授权体系中，详细信息请参考：条件块结构。
- G：表示with Grant option，即可以对object进行授权。

展现权限的示例如下：

```
odps@test_project> show grants for aliyun$odpstest1@aliyun.com
[roles]
dev
Authorization Type: ACL
[role/dev]
A projects/test_project/tables/t1: Select [user/odpstest1@aliyun.com]
A projects/test_project: CreateTable | CreateInstance | CreateFunction | List
A projects/test_project/tables/t1: Describe | Select
Authorization Type: Policy
[role/dev]
AC projects/test_project/tables/test_*: Describe
DC projects/test_project/tables/alifinance_*: Select [user/odpstest1@aliyun.com]
A projects/test_project: Create* | List
AC projects/test_project/tables/alipay_*: Describe | Select
Authorization Type: ObjectCreator
AG projects/test_project/tables/t6: All
AG projects/test_project/tables/t7: All
```

26.13.5 跨项目空间的资源分享

假设用户是项目空间的Owner或管理员（admin角色），有人需要申请访问用户的项目空间资源。如果申请人属于用户的项目团队，此时建议用户使用项目空间的用户与授权管理功能，详情请参考：项目空间的用户及授权管理。但是如果申请人并不属于用户的项目团队， 此时用户可以使用本章节介绍的基于Package的跨项目空间的资源分享功能。

Package是一种跨项目空间共享数据及资源的机制，主要用于解决跨项目空间的用户授权问题。对于如下场景出现的问题，如果不使用Package，将无法有效的解决。

假设，Alifinance项目空间的成员若要访问Alipay项目空间的数据，此时需要Alipay项目空间管理员先将Alifiance项目空间中的用户添加到Alipay项目空间中，然后再分别对这些新加入的用户进行普通授权。但实际上，Alipay项目空间管理员出于安全因素的考虑，并不希望对Alifiance项目空间中的每个用户都进行授权管理，此时就需要有一种机制能够使Alifiance项目空间管理员能对许可的对象进行自主授权控制。

使用Package之后，Alipay项目空间管理员可以对Alifinance需要使用的对象进行打包授权（也就是创建一个Package），然后许可Alifinance项目空间可以安装这个Package。在Alifinance项目空间管理员安装Package之后，就可以自行管理Package是否需要进一步授权给自己Project下的用户。

26.13.5.1 Package使用方法

Package的使用涉及到两个主体：Package创建者和Package使用者。Package创建者项目空间是资源提供方，它将需要分享的资源及其访问权限进行打包，然后许可Package使用方来安装使用。Package使用者项目空间是资源使用方，它在安装资源提供方发布的Package之后，便可以直接跨项目空间访问资源。

下面将分别介绍Package创建者和Package使用者所涉及的相关操作。

26.13.5.1.1 Package创建者相关操作

- **创建Package**

  创建Package的命令为：

  ```
  create package <pkgname>
  ```

- **删除Package**

  删除Package的命令为：

  ```
  delete package <pkgname>
  ```

- **将需要分享的资源添加到Package**

  添加资源到Package的命令为：

  ```
  add project_object to package package_name [with privileges <privileges>]
  remove project_object from package package_name
  project_object ::= table table_name | instance inst_name | function func_name | resource
   res_name
  privileges ::= action_item1, action_item2, ...
  ```

  说明:

  - 目前支持的对象类型不包括Project 类型，也就是不允许通过Package在其他Project中创对象。
  - 添加到Package中的不仅仅是对象本身，还包括相应的操作权限。当没有通过[with privileges privileges]来指定操作权限时，默认为只读权限，即Read/Describe/Select。“对象及其权限”被看作一个整体，package内的资源可以添加和删除，一旦添加和删除后原先授予的权限也会被回收掉。

- **许可其他项目空间使用Package**

  许可其他项目空间使用Package的命令为：

  ```
  allow project <prjname> to install package <pkgname> [using label <number>]
  ```

- **撤销其他项目空间使用Package的许可**

  撤销其他项目空间使用Package的许可的命令为：

  ```
  disallow project <prjname> to install package <pkgname>
  ```

- **查看已创建和已安装的Package列表**

  查看已创建及安装的Package列表的命令为：

  ```
  show packages
  ```

- **查看Package详细信息**

  查看Package详细信息的命令为：

  ```
  describe package <pkgname>
  ```

26.13.5.1.2 Package使用者相关操作

- **安装Package**

  安装Package的命令为：

  ```
  install package <pkgname>;
  ```

  说明: 对于安装Package来说，要求pkgName的格式为：<projectName>.<packageName>。

- **卸载Package**

  卸载Package的命令为：

  ```
  uninstall package <pkgname> ;
  ```

  说明: 对于卸载Package来说，要求pkgName的格式为：<projectName>.<packageName>。

- **查看Package**

  查看Package的命令为：

  ```
  show packages
  -- 查看已创建和已安装的package列表。
  ```

```
describe package <pkgname>
-- 查看package详细信息。
```

被安装的Package是一种独立的MaxCompute对象类型。若要访问Package里的资源（即其他项目空间分享给用户的资源），用户必须拥有对该Package的Read权限。如果请求者没有Read权限，则需要向Project Owner或 Admin申请。Project Owner或Admin 可以通过ACL授权或Policy授权机制来完成。

例如，如下的ACL授权允许云账号用户odps_test@aliyun.com访问Package里的资源：

```
use prj2 security
install package prj1.testpkg
grant read on package prj1.testpackage to user aliyun$odps_test@aliyun.com
```

如下的Policy授权允许项目空间prj2中的任何用户都可以访问Package里的资源：

```
use prj2
install package prj1.testpkg
put policy /tmp/policy.txt
```

/tmp/policy.txt内容如下：

```
{
"Version": "1", "Statement": [{
"Effect":"Allow",
"Principal":"*",
"Action":"odps:Read", "Resource":"acs:odps:*:projects/prj2/packages/prj1.testpkg"
}]
}
```

26.13.6 项目空间的数据保护

在现实中，有一些公司（比如金融机构、军工企业）对数据安全非常敏感，例如不允许员工将工作带回家，而只允许在公司内部进行操作，而且公司的所有电脑上的 USB 存储接口也都是禁用的。这样做的目的是禁止员工将敏感数据泄漏出去。

那么，作为MaxCompute项目空间管理员，如果用户对项目空间中的数据非常敏感，绝对不允许流出到其他项目空间中去，希望MaxCompute能将所有可能导致数据流出的操作禁止掉，则需要进行如下的相关设置及操作。

26.13.6.1 数据保护机制

MaxCompute提供的项目空间保护机制可以将所有可能导致数据流出的操作禁止掉。用户只需要在其项目空间中做如下设置：

```
set security.ProjectProtection=true
```

```
-- 设置ProjectProtection规则：数据只能流入，不能流出。
```

设置ProjectProtection后，用户的项目空间中的数据流向就会得到控制，即数据只能流入，不能流出。

默认时，ProjectProtection不会被设置，取值为false。同时在多个项目空间中拥有访问权限的用户可以自由地使用任意支持跨 Project的数据访问操作来转移项目空间的数据。如果对项目空间中的数据高度敏感，则需要管理员自行设置ProjectProtection保护机制。

26.13.6.2 开启数据保护后的数据流出方法

在用户的项目空间被设置了ProjectProtection之后，用户可能很快就会遇到这样的需求：Alice向用户提出申请，其需要将某张表的数据导出用户的项目空间。而且经过用户的审查之后，这张表的确没有泄漏用户关心的敏感数据。为了不影响 Alice的正常业务需要，MaxCompute为用户提供了在ProjectProtection被设置之后的两种数据导出途径。

- **设置ExceptionPolicy**

  Project Owner在设置ProjectProtection时可以附带一个exception策略，命令如下：

  ```
  SET ProjectProtection=true WITH EXCEPTION <policyFile>
  ```

  说明: 上述这种policy不同于Policy授权（尽管它与 Policy 授权语法完全一样），它只是对项目空间保护机制的例外情况的一种描述，即所有符合policy中所描述的访问情形都可以打破ProjectProtection规则。

  下面给出一个exception policy的示例。

  假设，允许云账号Alice@aliyun.com可以通过SQL任务对表alipay.table_test执行Select操作时将数据流出到alipay项目空间之外。

  ```
  {
  "Version": "1", "Statement": [{
  "Effect":"Allow", "Principal":"ALIYUN$Alice@aliyun.com",
  "Action":["odps:Select"],
  "Resource":"acs:odps:*:projects/alipay/tables/table_test",
  "Condition":{
  "StringEquals": { "odps:TaskType":["DT", "SQL"]
  }
  }
  }]
  }
  ```

  说明:

- Exception policy并不是一种普通的授权。如果云账号Alice并没有对表alipay.table_test的Select操作权限，即使设置了上述exception policy，Alice仍然是无法导出数据。ProjectProtection是一种数据流向的控制，而不是访问控制。只有在用户能访问数据的前提下，控制数据流向才是有意义的。
- TOC2TOU（time-of-check to time-of-use）数据泄露问题（又称Race Condition问题）。问题描述如下：

  1. [TOC阶段]用户A向Project Owner申请将t1导出， Project Owner对t1的数据敏感程度进行评估，PASS后通过exception policy授权A可以导出t1。
  2. 恶意用户修改了t1的内容，将敏感数据写入到t1。
  3. [TOU阶段]用户A将t1的内容导出。此时导出的t1已经不是Project Owner审查的那个t1了。

  防止TOC2TOU问题的建议：对于用户申请导出的表， Project Owner需要确保没有任何其他用户（含admin）能对该表进行更新（Update）操作或重建同名表操作（Drop + CreateTable）。对于TOC2TOU问题描述中的场景，建议Project Owner在第（1）步中以Project Owner身份创建t1的一个snapshot，设置exception policy时使用这个snapshot，并且不要授予admin角色给任何用户。

- **设置TrustedProject**

  若当前项目空间处于受保护状态，如果将数据流出的目标空间设置为当前空间的TrustedProject，那么向目标项目空间的数据流向将不会被视为触犯ProjectProtection规则。如果多个项目空间之间两两互相设置为TrustedProject，那么这些项目空间就形成了一个TrustedProject Group，数据可以在这个Project Group内流动，但禁止流出到Project Group之外。

  管理TrustedProject 的命令如下：

```
ist trustedprojects
-- 查看当前project中的所有TrustedProjects。
add trustedproject <projectname>
-- 在当前project中添加一个TrustedProject。
remove trustedproject <projectname>
```

```
-- 在当前project中移除一个TrustedProject。
```

26.13.6.3 资源分享与数据保护

在MaxCompute中，基于package的资源分享机制与项目空间的数据保护机制是正交的两种安全机制，但在功能上却是相互制约的。

MaxCompute规定：资源分享优先于数据保护。即如果一个数据对象是通过资源分享方式授予其他项目空间的用户访问， 那么该数据对象将不受ProjectProtection规则的限制。

如果要防止数据从项目空间的流出，在设置ProjectProtection=true之后，还需检查如下配置：

- 确保没有添加trustedproject，如果有设置，则需要评估可能的风险。
- 确保没有设置exception policy，如果有设置，则需要评估可能的风险，尤其要考虑TOC2TOU数据泄露风险。
- 确保没有使用package数据分享，如果有设置，则需要确保package中没有敏感数据。

26.13.7 项目空间的安全配置

MaxCompute是一个支持多租户的数据处理平台，不同的租户对数据安全需求不尽相同。为了满足不同租户对数据安全的灵活需求，MaxCompute支持项目空间级别的安全配置，Project Owner可以定制适合自己的外部账号支持和鉴权模型。

MaxCompute支持了多种正交的授权机制，如 ACL授权，Policy授权，隐式授权（如对象创建者自动被赋予访问对象的权限）。但是并非所有用户都需要使用这些安全机制。 用户可以根据自己的业务安全需求或使用习惯，合理设置本项目空间的鉴权模型。

```
show SecurityConfiguration
-- 查看项目空间的安全配置。
set security.CheckPermissionUsingACL=true/false
-- 激活/冻结ACL授权机制，默认为true。
set security.CheckPermissionUsingPolicy=true/false
-- 激活/冻结Policy授权机制，默认为true。
set security.ObjectCreatorHasAccessPermission=true/false
-- 允许/禁止对象创建者默认拥有访问权限，默认为true。
set security.ObjectCreatorHasGrantPermission=true/false
-- 允许/禁止对象创建者默认拥有授权权限，默认为true。
set security.LabelSecurity=true/false
-- 开启/关闭LabelSecurity安全策略。
set security.ProjectProtection=true/false
```

```
-- 开启/关闭项目空间的数据保护机制，禁止/允许数据流出项目空间。
```

26.13.8 Policy权限策略

26.13.8.1 Policy概况

Policy授权是一种基于主体的授权。通过Policy授权的权限数据（即访问策略）被看做是授权主体的一种子资源。只有当主体（用户或角色）存在时，才能进行Policy授权操作。当主体被删除时，通过Policy授权的权限数据会被自动删除。Policy授权使用MaxCompute自定义的一种访问策略语言来进行授权，允许或禁止主体对项目空间对象的访问权限。

Policy授权是一种新的授权机制，它主要解决ACL授权机制无法解决的一些复杂授权场景，例如：

- 一次操作对一组对象进行授权，如所有的函数、所有以**taobao**开头的表。
- 带限制条件的授权，如授权只会在指定的时段内才会生效、当请求者从指定的IP地址发起请求时授权才会生效、或者只允许用户使用SQL（而不允许其它类型的Task）来访问某张表。

Policy授权命令格式如下：

```
GET POLICY;
-- 读取项目空间的Policy。
PUT POLICY <policyFile>;
-- 设置（覆盖）项目空间的Policy。
GET POLICY ON ROLE <roleName>;
-- 读取项目空间中某个角色的Policy。
PUT POLICY <policyFile> ON ROLE <roleName>;
-- 设置（覆盖）项目空间中某个角色的Policy。
```



说明： MaxCompute目前支持的Policy类型有Project Policy和Role Policy。Project Policy对Project中的所有用户有效，而Role Policy只对已赋予角色的用户有效。在Policy格式上，Project Policy必须指定Principal（用户），而Role Policy则不能指定Principal（因为用户与角色的指派关系已经存在）。

使用Project Policy授权的示例如下：

场景说明：授权用户alice@aliyun.com只能在“2013-11-11 23:59:59”这个时间点之前，只能从“10.32.180.0~23”这个IP段提交请求，只允许在项目空间test_project中执行CreateInstance,CreateTable和List操作，禁止删除test_project下的任何表。

编写Policy如下：

- ```
  {
  "Version": "1", "Statement": [{
  "Effect":"Allow", "Principal":"ALIYUN$alice@aliyun.com",
  "Action":["odps:CreateTable","odps:CreateInstance","odps:List"],
  "Resource":"acs:odps:*:projects/test_project",
  ```
  ```

```
"Condition":{ "DateLessThan": {
"acs:CurrentTime":"2013-11-11T23:59:59Z"
},
"IpAddress": { "acs:SourceIp":"10.32.180.0/23"
}
}
},
{
"Effect":"Deny",  "Principal":"ALIYUN$alice@aliyun.com",  "Action":"odps:Drop",
 "Resource":"acs:odps:*:projects/test_project/tables/*"
}]
}
```json
```

- ```json
  {
  "Version": "1", "Statement": [{
  "Effect":"Allow", "Principal":"ALIYUN$alice@aliyun.com",
  "Action":["odps:CreateTable","odps:CreateInstance","odps:List"],
  "Resource":"acs:odps:*:projects/test_project",
  "Condition":{ "DateLessThan": {
  "acs:CurrentTime":"2013-11-11T23:59:59Z"
  },
  "IpAddress": { "acs:SourceIp":"10.32.180.0/23"
  }
  }
  },
  {
  "Effect":"Deny", "Principal":"ALIYUN$alice@aliyun.com",
  "Action":"odps:Drop",
  "Resource":"acs:odps:*:projects/test_project/tables/*"
  }]
  }
  ```

- ```
  {
  "Version": "1", "Statement": [{
  "Effect":"Allow", "Principal":"ALIYUN$alice@aliyun.com",
  "Action":["odps:CreateTable","odps:CreateInstance","odps:List"],
  "Resource":"acs:odps:*:projects/test_project",
  "Condition":{ "DateLessThan": {
  "acs:CurrentTime":"2013-11-11T23:59:59Z"
  },
  "IpAddress": { "acs:SourceIp":"10.32.180.0/23"
  }
  }
  },
  {
  "Effect":"Deny", "Principal":"ALIYUN$alice@aliyun.com",
  "Action":"odps:Drop",
  "Resource":"acs:odps:*:projects/test_project/tables/*"
  }]
  }
  ```

说明:

- 目前仅支持Role Policy和Project Policy，暂不支持User Policy。
- 每种Policy只支持一个Policy文件。由于Put Policy操作会覆盖已有的Policy，当用户需要修改Policy时，应按如下顺序分步操作：
  ```

1. Get Policy。
2. 用户自己Merge Policy Statements。
3. Put Policy。

26.13.8.2 Policy基本术语

权限（Permission）是访问控制的一个基本概念，它表示允许或拒绝一个请求者（Requester）对资源（Resource）执行某种操作（Action）。用语句（Statement）来表示单个权限的形式化描述，而用策略（Policy）来表示语句的集合。

访问策略（Access Policy）主要包括如下的访问控制元素：主体（Principal）、操作（Action）、资源（Resource）、访问限制（Access Restriction）和效力（Effect）。下面分别对这几个基本实体进行简要描述。

- **主体**（Principal）

  主体（Principal）是指访问策略中的权限被指派的对象。例如，访问策略允许张三在2011年12月31日之前对资源SampleBucket执行CreateObject操作中的主体是张三。

- **操作**（Action）

  操作（Action）是指主体对资源的访问方法。例如，访问策略允许张三在2011年12月31日之前对资源SampleBucket执行CreateObject操作中的操作是CreateObject。

- **资源**（Resource）

  资源（Resource）是指主体请求访问的对象。例如，访问策略允许张三在2011年12月31日之前对资源SampleBucket执行CreateObject操作中的资源是SampleBucket。

- **访问限制**（Access Restriction）

  访问限制（Access Restriction）是指权限生效的限制条件。例如，访问策略允许张三在2011年12月31日之前对资源SampleBucket执行CreateObject操作中的限制条件是在2011年12月31日之前。

- **效力**（Effect）

  授权效力包括两个方面：允许操作（Allow）和拒绝操作（Deny）。通常，Deny有更高的效力，在权限检查时会优先使用。

  注意: 拒绝操作和撤销授权是完全独立的两个概念，撤销授权通常包括撤销对Allow和Deny这两种不同效力的授权。例如传统数据库一般支持Revoke和Revoke Deny两种操作。

26.13.8.3 访问策略语言结构

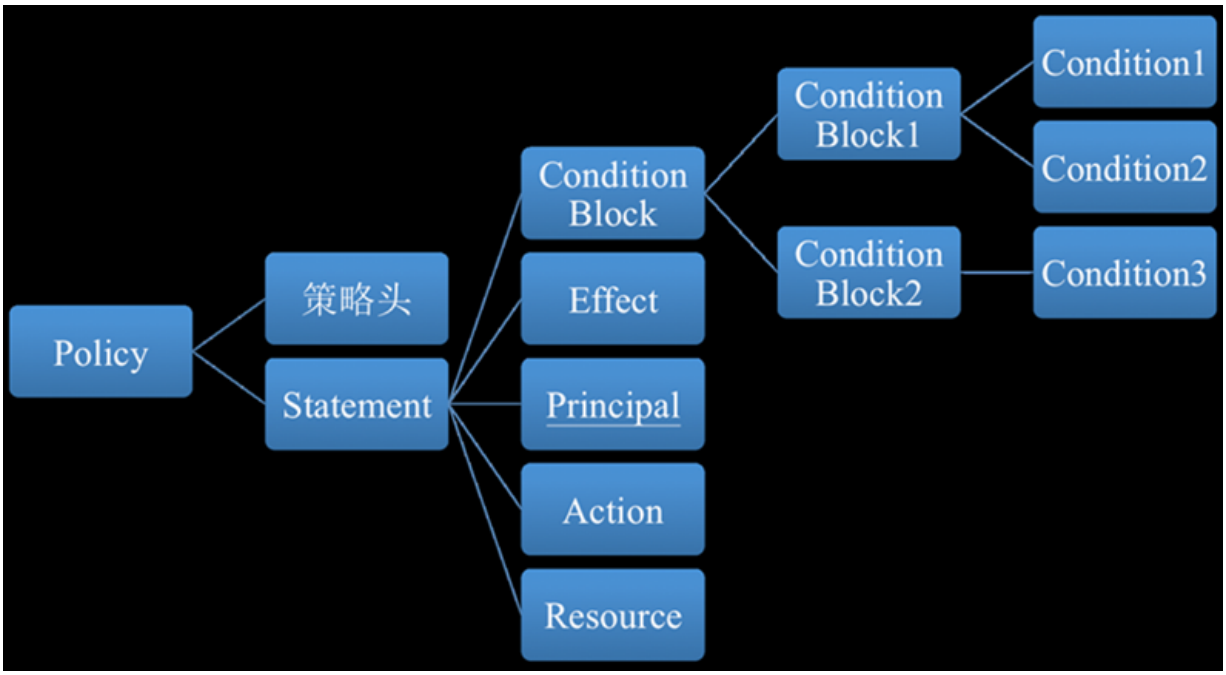
下图描述了访问策略的基本结构。一个策略（Policy）由如下部分组成：

- 可选的策略头部元素。
- 一条或多条语句（Statement）。

策略头部元素是可选的，它主要包括策略的版本信息。策略的主体是语句的集合。

Policy结构如下图所示：

图 780: Policy 结构图



26.13.8.3.1 授权语句（Statement）结构

一条授权语句（Statement）包括如下条目：

- Effect：指明该条语句的权限类型，取值必须为Allow或Deny。
- Principal：如果Policy在授权时是与用户或角色绑定，那么就不允许再指定Principal，例如MaxCompute的Role Policy。如果 Policy在授权时是与项目空间或项目空间内的对象绑定，那么必须指定Principal，例如MaxCompute的Project Policy。
- Action： 表示授权操作，可以是一个或多个操作名，可支持通配符号*和?。例如，Action = *表示所有的操作。
- Resource：表示授权对象，可以是一个或多个对象名，可支持通配符号*和?。例如，Resource = * 表示所有的对象。

- Condition Block：条件块是该条授权语句所述权限得以生效的条件。条件块结构请参靠下节的描述。

26.13.8.3.2 条件块结构

条件块由一个或多个条件子句构成。一个条件子句由条件操作类型，条件关键字和条件值组成。条件操作类型和条件关键字在下面的章节中会有详细描述。

条件块的满足性判断原则如下所示：

- 一个条件关键字可以指定一个或多个值，在条件检查时，如果条件关键字的值与指定值中的某一个相等，即可判定条件满足。
- 同一种条件操作类型的条件子句下的多个条件关键字同时满足的情况下，才能判定该条件子句满足。
- 条件块下的所有条件子句同时满足的情况下，才能判定该条件块满足。

26.13.8.3.3 条件操作类型

当前支持如下条件操作类型：字符串类型、数字类型、日期类型、布尔类型和IP地址类型。每种条件操作类型分别支持如下的方法：

String：

```
StringEquals
StringNotEquals
StringEqualsIgnoreCase
StringNotEqualsIgnoreCase
StringLike
StringNotLike
```

Numeric：

```
NumericEquals
NumericEquals
NumericLessThan
NumericLessThanEquals
NumericGreaterThan
NumericGreaterThanEquals
```

Date and time：

```
DateEquals
DateNotEquals
DateLessThan
DateLessThanEquals
DateGreaterThan
```

```
DateGreaterThanEquals
```

Boolean：

```
Bool
```

IP address：

```
IpAddress NotIpAddress
```

26.13.8.3.4 条件关键字

MaxCompute支持开放云服务ACS（Aliyun Cloud Service）保留的如下条件关键字：

ACS保留条件关键字	类型	说明
acs:CurrentTime	Date and time	Web Server 接收到请求的时间。以ISO 8601格式表示，如2012-11-11T23:59:59Z。
acs:SecureTransport	Boolean	发送请求是否使用了安全信道，如HTTPS。
acs:SourceIp	IP address	发送请求时的客户端IP地址。
acs:UserAgent	String	发送请求时的客户端UserAgent。
acs:Referer	String	发送请求时的HTTP referer。

说明: acs:SourceIp是指http connection的remote_ip，而不是http header中的x-forwarded- for所指的ip list中的ip。举例来说，如果10.230.205.105是一个局域网Ip，acs:SourceIp则为该网络的网关出口Ip。如果该网络出口使用了代理服务器，则acs:SourceIp为该代理服务器Ip；如果有多级代理服务器，则为最后一级代理服务器Ip。如果代理服务器设置的代理规则不同，则对acs:SourceIp的值也有影响。

26.13.8.4 访问策略语言规范

26.13.8.4.1 Principal命名规范

Principal是指请求发送者的身份。目前仅支持阿里云账号、域账号和淘宝账号的身份表示。对于云账号，可以支持ID或DisplayName的表示。

示例如下：

```
"Principal": "43274"
"Principal": "ALIYUN$jun.li@aliyun.com"
```

```
"Principal": ["ALIYUN$jun.li@aliyun.com","ALIYUN$jack@aliyun.com", "TAOBAO$alice"]
```

26.13.8.4.2 Resource命名规范

目前使用如下格式来命名MaxCompute所提供的资源。格式如下：

```
acs:<service-name>:<namespace>:<relative-id>
```

说明:

- acs：保留的Resource头部。
- service-name：云开放服务名称，如maxcompute、oss、table store等。
- namespace：命名空间，用于资源隔离。如果以云账号ID来做资源隔离，那么可取值为云账号ID。如果不支持该项，可以使用通配符*号来代替。
- relative-id：与service相关的资源描述部分，其语义由具体service指定。这部分的格式描述支持类似于文件路径的树状结构。以MaxCompute为例，relative-id的格式为：

  ```
  projects/<project_name>/<object_type>/<object_name>
  ```

MaxCompute Resource命名实例如下：

Resource命名实例	说明
*	项目空间中的所有对象。
projects/prj1/tables/t1	项目空间prj1中的t1表。
projects/prj1/instances/*	项目空间prj1中的所有的instances。
projects/prj1/tables/*	项目空间prj1中的所有表。
projects/prj1/tables/taobao*	项目空间prj1中的所有以taobao开头的表。

26.13.8.4.3 Action命名规范

Action命名规范如下：

```
<service-name>:<action-name>
```

说明:

- service-name：开放服务名称，如maxcompute 、oss、table store等。
- action-name：service相关的操作接口名称。

MaxCompute Action命名实例如下：

Action命名实例	说明
*	所有操作。
odps:*	MaxCompute的所有操作。
odps:CreateTable	MaxCompute的CreateTable操作。
odps:Create*	MaxCompute的所有以Create开头的操作。

26.13.8.4.4 Condition Keys命名规范

开放云服务ACS保留的条件关键字命名格式为：

```
acs:<condition-key>
```

说明: condition-key：ACS保留了5种条件关键字，所有的开放服务可以共用。它们是：

acs:CurrentTime，acs:SecureTransport，acs:SourceIp，acs:UserAgent，acs:Referer。

具体服务相关的条件关键字的命名格式为：

```
<service-name>:<condition-key>
```

说明: condition-key：service自定义的条件关键字。

26.13.8.4.5 访问策略样例

Policy Sample：

```
{
"Version": "1",
"Statement": [{
"Effect":"Allow",
"Principal":"ALIYUN$alice@aliyun.com",
"Action":["odps:CreateTable","odps:CreateInstance","odps:List"],
"Resource":"acs:odps:*:projects/prj1",
"Condition":{ "DateLessThan": {
"acs:CurrentTime":"2013-11-11T23:59:59Z"
},
"IpAddress": { "acs:SourceIp":"10.32.180.0/23"
}
}
},
{
"Effect":"Deny", "Principal":"ALIYUN$alice@aliyun.com",
"Action":"odps:Drop",
"Resource":"acs:odps:*:projects/prj1/tables/*"
}]
```

```
}
```



说明： 授权用户alice@aliyun.com只能在2013-11-11T23:59:59Z这个时间点之前，只能从10.32.180.0/23这个IP段提交请求，只允许在项目空间prj1中执行CreateInstance，CreateTable和List操作，禁止删除prj1下的任何table。

26.13.8.5 Policy与ACL的区别

ACL授权的特点：

- 授权或撤销授权时，要求:Grantee（如User或Role）和Object （如Table）必须已经存在。这点与Oracle授权特性相似，可以避免删除并重建同名对象所带来的安全风险。
- 删除一个对象时，自动撤销与该对象关联的所有授权。
- 仅支持Allow（白名单）授权，不支持Deny（黑名单）授权。
- 使用经典的Grant/Revoke授权命令进行授权。命令简单，使用时不易出错。不支持带限制条件的授权。
- 适合于简单的授权需求：授权不带限制条件，不需要Deny，并只需要对已存在对象进行授权。

Policy授权的特点：

- 授权或撤销授权时，不关心Grantee或Object存在与否。授权对象可以支持以通配符""来表达，例如，*projects/tbproj/tables/taobao*，它表示项目空间tbproj中所有以taobao开头的表。这点与Mysql授权特性相似，它允许对不存在的对象授权，授权者应考虑到删除并重建同名对象所带来的安全风险。
- 删除一个对象时，与该对象关联的Policy授权不会被删除。
- 同时支持Allow（白名单）和Deny（黑名单）授权。当Allow和Deny授权同时存在时，遵循Deny优先原则。
- 支持带限制条件的授权。授权者可以对Allow或Deny授权施加条件限制（目前支持20种条件操作）。例如，允许请求者的IP为指定的IP地址范围，同时访问时间必须在2012-11-11 23:59:59之前。
- 适合于相对复杂的授权需求：带授权限制条件，有Deny授权需求，希望支持对未来的对象授权。
- 使用Policy授权命令进行授权，命令较为复杂。

26.13.8.6 应用限制

名称	限制数	说明
ACCESS_POLICY_SIZE_LIMIT	32 KB	AccessPolicy的文本长度限制。
USER_NUMBER_LIMIT_IN_ON E_PROJECT	1000	项目空间允许添加的最大用户数量。
ROLE_NUMBER_LIMIT_IN_ON E_PROJECT	500	项目空间允许创建的最大角色数量。
ROLE_NAME_LENGTH_LIMIT	64	角色命名的字符长度。
SECURITY_COMMENT_SIZE_L IMIT	1 KB	注释的字符长度。
PACKAGE_NAME_LENGTH_LI MIT	128	Package命名的字符长度。
ALLOW_PROJECT_NUMBER_L IMIT_IN_ONE_PACKAGE	1024	一个Package允许被多少Project安装。
RESOURCE_NUMBER_LIMIT_I N_ONE_PACKAGE	256	一个Package允许包含的最大资源数量。
PACKAGE_NUMBER_LIMIT_IN _ONE_PROJECT	512	一个Project允许创建的Package数量。
INSTALLED_PACKAGE_NUMB ER_LIMIT_IN_ONE_PROJECT	64	一个Project允许安装的Package数量。

26.13.9 安全相关语句汇总

26.13.9.1 项目空间的安全配置

鉴权配置

语句	说明
show SecurityConfiguration	查看项目空间的安全配置。
set CheckPermissionUsingACL=true/false	激活/冻结 ACL 授权机制。
set CheckPermissionUsingPolicy=true/fal se	激活/冻结 Policy 授权机制。
set ObjectCreatorHasAccessPermission=t rue/false	允许/禁止对象创建者默认拥有访问权限。
set ObjectCreatorHasGrantPermission=tr ue/false	允许/禁止对象创建者默认拥有ACL授权权限。

数据保护

语句	说明
set ProjectProtection=false	关闭数据保护机制。
set ProjectProtection=true [with exception <policy>]	开启数据保护机制。
list TrustedProjects	查看可信项目空间列表。
add TrustedProject <projectName>	添加可信项目空间。
remove TrustedProject <projectName>	移除可信项目空间。

26.13.9.2 项目空间的权限管理

用户管理

语句	说明
list users	查看所有已添加进来的用户.
add user <username>	添加一个用户。
remove user <username>	移除一个用户。

角色管理

语句	说明
list roles	查看所有已创建的角色。
create role <rolename>	创建一个角色。
drop role <rolename>	删除建一个角色。
grant <rolelist> to <username>	撤销对用户的角色指派。
revoke <rolelist> from <username>	对用户指派一个或多个角色。

ACL授权

语句	说明
grant <privList> on <objType> <objName> to user <username>	对用户授权。
grant <privList> on <objType> <objName> to role <rolename>	对角色授权。

语句	说明
revoke <privList> on <objType> <objName> from user <username>	撤销对用户的授权。
revoke <privList> on <objType> <objName> from role <rolename>	撤销对角色的授权。

Policy授权

语句	说明
get policy	查看项目空间级别的Policy设置。
put policy <policyFile>	设置项目空间级别的Policy。
get policy on role <roleName>	查看某个角色的Policy设置。
put policy <policyFile> on role <roleName>	设置某个角色的Policy。

权限审查

语句	说明
whoami	查看当前用户信息。
show grants [for <username>] [on type <objectType>]	查看用户权限和角色。
show acl for <objectName> [on type <objectType>]	查看具体对象的授权信息。
describe role <roleName>	查看角色的授权信息和角色指派。

26.13.9.3 基于Package的资源分享

分享资源

语句	说明
create package <pkgName>	创建一个Package。
delete package <pkgName>	删除一个Package。
add <objType> <objName> to package<pkgName> [with privileges privs]	向Package中添加需要分享的资源。
remove <objType> <objName> from package <pkgName>	从Package中删除已分享的资源。

语句	说明
allow project <prjName> to install package <pkgName> [using label <num>]	许可某个项目空间使用用户Package。
disallow project <prjName> to install package <pkgName>	禁止某个项目空间使用用户的Package。

使用资源

语句	说明
install package <pkgName>	安装Package。
uninstall package <pkgName>	卸载Package。

查看Package

语句	说明
show packages	列出所有创建和安装的Packages。
describe package <pkgName>	查看package的详细信息。

26.14 工具

26.14.1 MaxCompute客户端

本章节是MaxCompute客户端用户手册的一部分，在此说明如何借助客户端命令行工具，使用MaxCompute服务的基础功能。

注意:

- 请用户不要依赖客户端的输出格式来做任何的解析工作。客户端的输出格式不承诺向前兼容，不同版本间的客户端命令格式及行为有差异。
- MaxCompute客户端是一个java程序，需要JRE环境才能运行，请下载并安装JRE1.8 版本。

如果想快速了解客户端的使用方式，请参考：快速开始中关于配置客户端的介绍。

26.14.1.1 安装客户端

1. 下载客户端压缩包到本地PC。
2. 将下载好的客户端压缩包解压到一个文件夹中，在该文件夹中可以看到如下4个文件夹。

```
bin/
```

```
conf/
lib/
plugins/
```

3. 编辑conf文件夹中的**odps_config.ini**文件，配置如下相关信息。

```
project_name=
access_id=********************
access_key=**********************
end_point= <MaxCompute服务地址>
```



说明:

- 将其中的access_id和access_key替换为从www.aliyun.com 网站上申请到的access_id和access_key。
- 如果用户经常使用某个project，可以将该project的名字添加到project_name=后。这样可以避免每次进入客户端后均需要执行use project_name;命令。

4. 修改好配置文件后，运行bin目录下的**odps**文件（在Linux 系统下运行*./bin/odpscmd*，在Windows下运行*./bin/odpscmd.bat*），此时就可以开始运行MaxCompute命令了，示例如下。

```
create table tbl1(id bigint);
insert overwrite table tbl1 select count(*) from tbl1;
select 'welcome to MaxCompute!' from tbl1;
```

26.14.1.2 配置及参数说明

- **获取帮助**

  显示客户端的帮助信息，命令格式如下：

```
odps@ >./bin/odpscmd -h;
```



说明: 也可以在交互模式下键入h;或 help;（不区分大小写）。

- **启动参数**

  在启动时，可以指定一系列参数，命令格式如下：

```
Usage: odpscmd [OPTION]... where options include: --help (-h)for help --project=<prj_name>
 use project --endpoint=<http://host:port> set endpoint -u <user_name> -p <password>
 user name and password --instance-priority=<priority> priority scope[0-9] -M read machine
 readable data -k <n> will skip begining queries and start from specified position -r <n> set
```

```
retry times -f <"file_path;"> execute command in file -e <"command;[command;]..."> execute command, include sql command -C will display job counters
```

示例如下：（以-f 参数为例）

1. 准备本地脚本文件script.txt，假设在D 盘，文件内容如下。

```
DROP TABLE IF EXISTS test_table_mj;
CREATE TABLE test_table_mj (id string, name string);
DROP TABLE test_table_mj;
```

2. 运行如下命令。

```
odpscmd\bin>odpscmd -f d:/script.txt;
```

3. 显示执行结果。

```
ID = 20150528122432906gux77io3
Log view:
http://webconsole.odps.aliyun-inc.com:8080/logview/?h=http://service-corp.odps.aliyun-
                                inc.com/api&podps_public_dev&i2015052                                8122432906g
tokenRnlrSzJoL242YW43dFFlc1dmb1ZWZzFxQ1RFPSxPRFBTX09CTzoxMDc
wMDI1NjI3ODA1                                NjI5LDE0MzM0MjA2NzMseyJTdGF0ZW1lbnQiOlt7
IkFjdGlvbiI6WyJvZHBzOlJlYWQiXSwiRWZmZWN0IjoiQWxsb3ciLCJSZXNv
dXJjZSI6WyJhY3M6b2RwczoqOnB                yb2plY3RzL29kcHNfcHVibGljX2Rld
i9pbnN0YW5jZXMvMjAxNTA1MjgxMjI0MzI5MDZndXg3N2lvMyJdfV0sIlZlc
nNpb24iOiIxIn0=
OK
ID = 20150528122439318gcmkk6u1
Log view:
http://webconsole.odps.aliyun-inc.com:8080/logview/?h=http://service-corp.odps.aliyun-
                                inc.com/api&podps_public_dev&i2015052                                8122439318gcm
tokendSt0RXdlV0M5YjZET2I1MnJuUFkzWDN1aWpzPSxPRFBTX09CTzoxMDc
wMDI1NjI3ODA1NjI5LDE0MzM0MjA2ODAseyJTdGF0ZW1lbnQiOlt7IkFjdGl
vbiI6WyJvZHBzOlJlYWQiXSwiRWZmZWN0IjoiQWxsb3ciLCJSZXNvdXJjZSI
6WyJhY3M6b2RwczoqOnB                yb2plY3RzL29kcHNfcHVibGljX2Rld
i9pbnN0YW5jZXMvMjAxNTA1MjgxMjI0MzkzMThnY21razZ1MSJdfV0sIlZlc
nNpb24iOiIxIn0=
OK
ID = 20150528122440389g98cmlmf
Log view:
http://webconsole.odps.aliyun-inc.com:8080/logview/?h=http://service-corp.odps.aliyun-
                                inc.com/api&podps_public_dev&i2015052                                8122440389g9
tokenNWlwL0EvQThxUXhzcTRERDc5NFg0b2IxZ3QwPSxPRFBTX09CTzoxMDc
wMDI1NjI3OD                                A1NjI5LDE0MzM0MjA2ODAseyJTdGF0ZW1lbnQiOl
t7IkFjdGlvbiI6WyJvZHBzOlJlYWQiXSwiRWZmZWN0IjoiQWxsb3ciLCJSZX
NvdXJjZSI6WyJhY3M6b2RwczoqOnB                yb2plY3RzL29kcHNfcHVibGljX2Rld
i9pbnN0YW5jZXMvMjAxNTA1MjgxMjI0NDAzODlnOThjbWxtZiJdfV0sIlZlc
nNpb24iOiIxIn0=
OK
```

- **交互模式**

直接运行客户端将进入到交互模式，格式如下。

```
[admin: ~]$odpscmd Aliyun ODPS Command Line Tool Version 1.0 @Copyright 2012 Alibaba Cloud Computing Co., Ltd. All rights reserved. XXX@ XXX> INSERT OVERWRITE TABLE DUAL SELECT * FROM DUAL;
```



说明: 上面的第一个XXX会显示odps的标识符，第二个XXX会显示用户当前所在的项目空间。在光标位置输入命令（以分号作为语句结束标志），回车即可运行。

- **输出消息**

  提交SQL语句后的返回值有两种输出格式：HumanReadable格式（默认）和MachineReadable格式。如果用户运行odpscmd时加了“-M”参数，则以CSV格式输出。

  说明: 此功能当前只对应select语句和read语句，在读取数据时有效。

- **续跑**

  在用-e或-f模式运行时，如果有多个语句，并且想从中间某个语句开始运行，可以指定参数-k，表示忽略前面的语句，从指定位置的语句开始运行。当指定参数<= 0时，从第一条语句开始执行。每个以分号分隔的语句被视为一个有效语句，在运行时会打印出当前运行成功或者失败的是第几个语句。

  例如文件*/tmp/dual.sql*中有如下三条sql语句：

  ```
  drop table dual;
  create table dual (dummy string);
  insert overwrite table dual select count(*) from dual;
  ```

  想忽略前面两条语句，需执行如下命令：

  ```
  odpscmd -k 3 -f dual.sql
  ```

- **获取当前登录用户**

获取当前登录用户云账号及使用的end point配置，命令格式如下：

```
whoami
```

示例如下：

```
odps@ hiveut>whoami; Name: odpstest@aliyun.com ID: 1090142773636588 End_Point: <MaxCompute服务地址> Project: lijunsecuritytest
```

- **退出**

  命令格式如下：

  ```
  odps@ > quit;
  ```

  或

  ```
  q;
  ```

- **设置作业优先级**

  命令格式如下：

  ```
  Admin@ > ./bin/odpscmd --instance-priority=<PRIORITY>;
  ```

  配置文件：odps_config.ini

  `instance_priority=<PRIORITY>`

  > **注意:**
  >
  > - <PRIORITY > 的取值范围为0-9，0最大，9最小。
  > - 在配置文件中配置后，对在 CLT 中提交的所有Instance有效。
  > - 用户如果没有设置Priority，优先级默认值为9。

- **DryRun模式**

  DryRun是指提交一个有效的SQL语句时，MaxCompute只分析其语法正确性，生成相应的执行计划，但是并不提交分布式作业即返回。命令格式如下：

  ```
  ./bin/odpscmd -y
  ```

- **sql可靠性**

  在客户端中执行INSERT或者CREATE TABLE AS等SQL语句时。如果作业执行中间过程出现了异常，客户端会尽可能根据已知的信息将数据和元数据都自动恢复到执行SQL QUERY之前的状态，包括：

- INSERT OVERWRTIE在QUERY执行中覆盖的数据，会从临时备份目录中恢复到原目录。
- INSERT INTO在QUERY执行过程中产生的新增数据，会全部删除。
- QUERY执行中创建的表，或动态生成的原来不存在的分区信息，全部删除。

如果MaxCompute无法完成恢复过程，则会输出一个特定的出错码，告诉用户不要再进行尝试，否则可能导致部分数据无法恢复。在这种情况下的出错信息如下：

```
ODPS-
0110999: Critical! Internal error happened in commit operation and rollback failed, possible
 breach of atomicity
```

26.14.2 Eclipse开发插件

26.14.2.1 安装Eclipse

背景信息

为了方便用户使用MapReduce及UDF的Java SDK进行开发工作，MaxCompute提供了Eclipse开发插件。该插件能够模拟MapReduce及UDF的运行过程，为用户提供本地调试手段，并提供简单的模板生成功能。

说明： 与MapReduce提供的本地运行模式不同，Eclipse插件不能够与MaxCompoute同步数据。用户使用的数据需要手动拷贝到Eclipse插件的**warehouse**目录下。

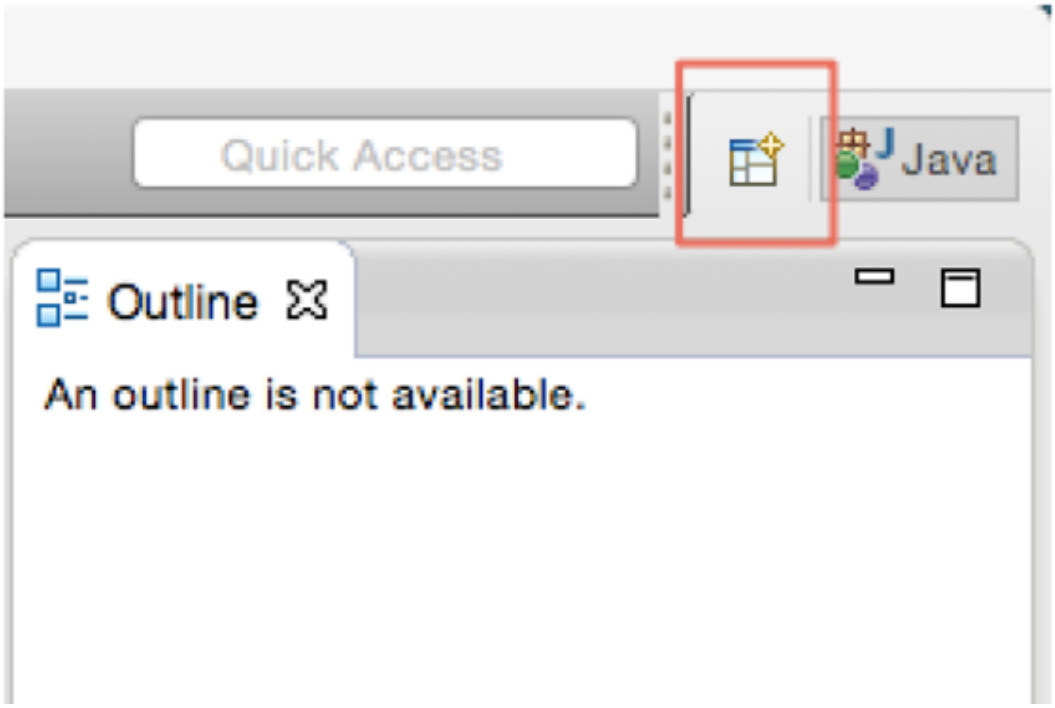
操作步骤

1. 将Eclipse软件包解压。解压后将看到如下的jar内容。

   odps-eclipse-plugin-bundle-0.15.0.jar

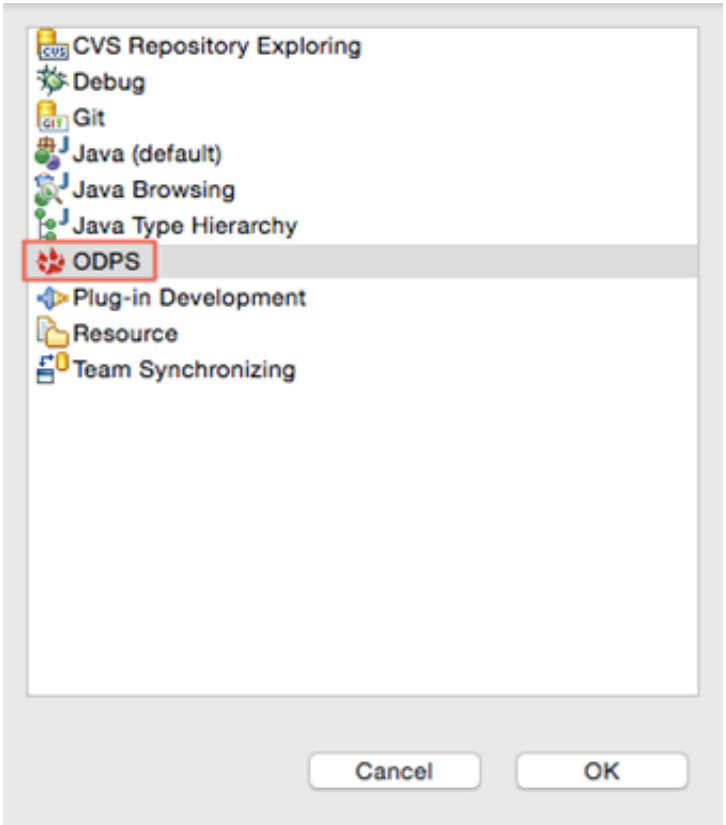
2. 将插件放置在Eclipse安装目录的**plugins**子目录下。
3. 打开Eclipse，单击右上角的打开透视图（Open Perspective），页面如下图所示。

   图 781: Eclipse安装图1



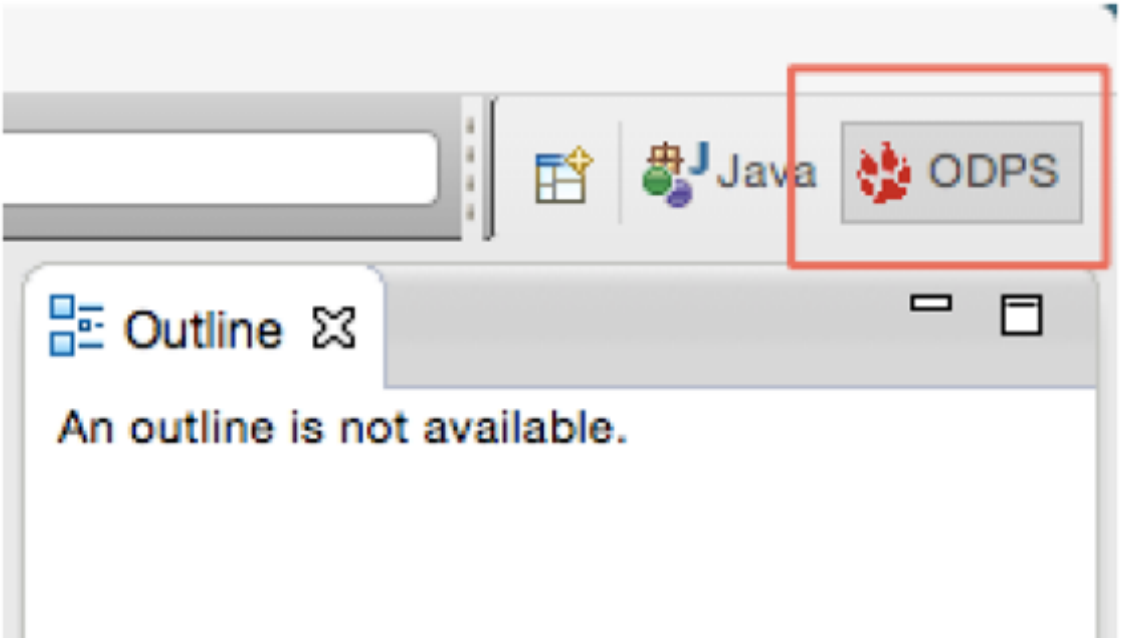
4. 在弹出的如下对话框中单击**ODPS**。

图 782: Eclipse安装图2



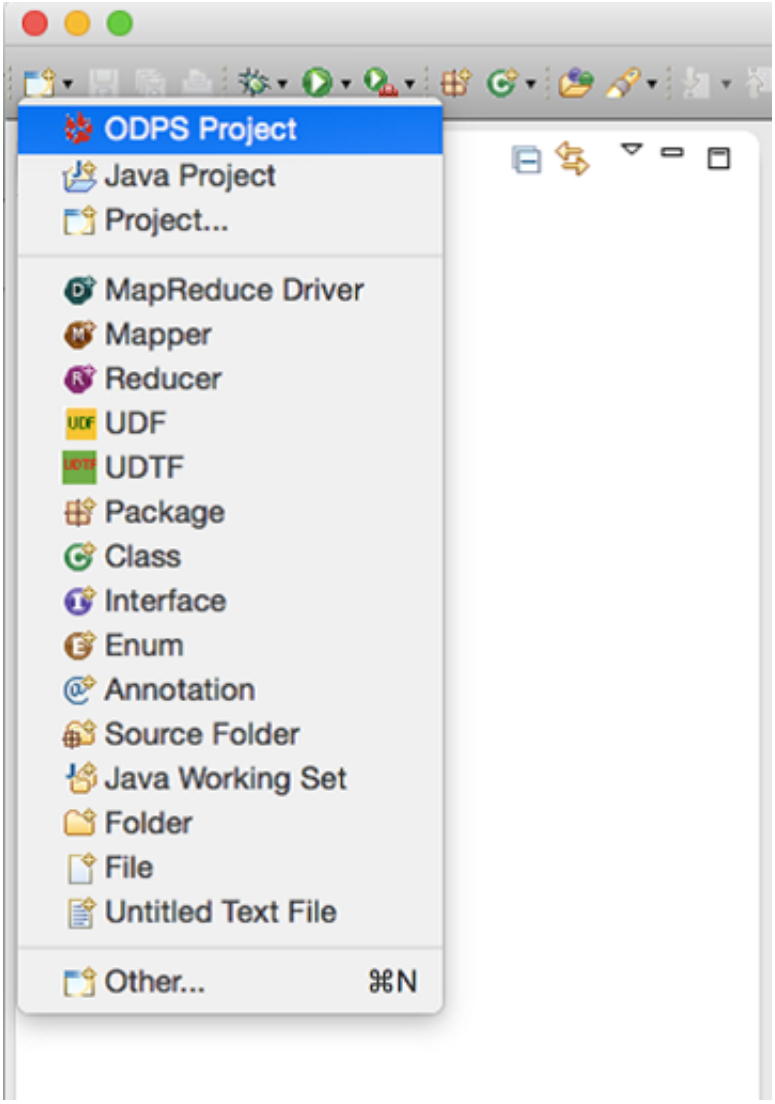
5. 单击**ODPS**后出现如下图的对话框。

图 783: Eclipse安装图3



6. 选择**ODPS**，并单击**OK**。同样在右上角会出现ODPS图标，表示插件生效，如下图所示。

图 784: Eclipse安装图4



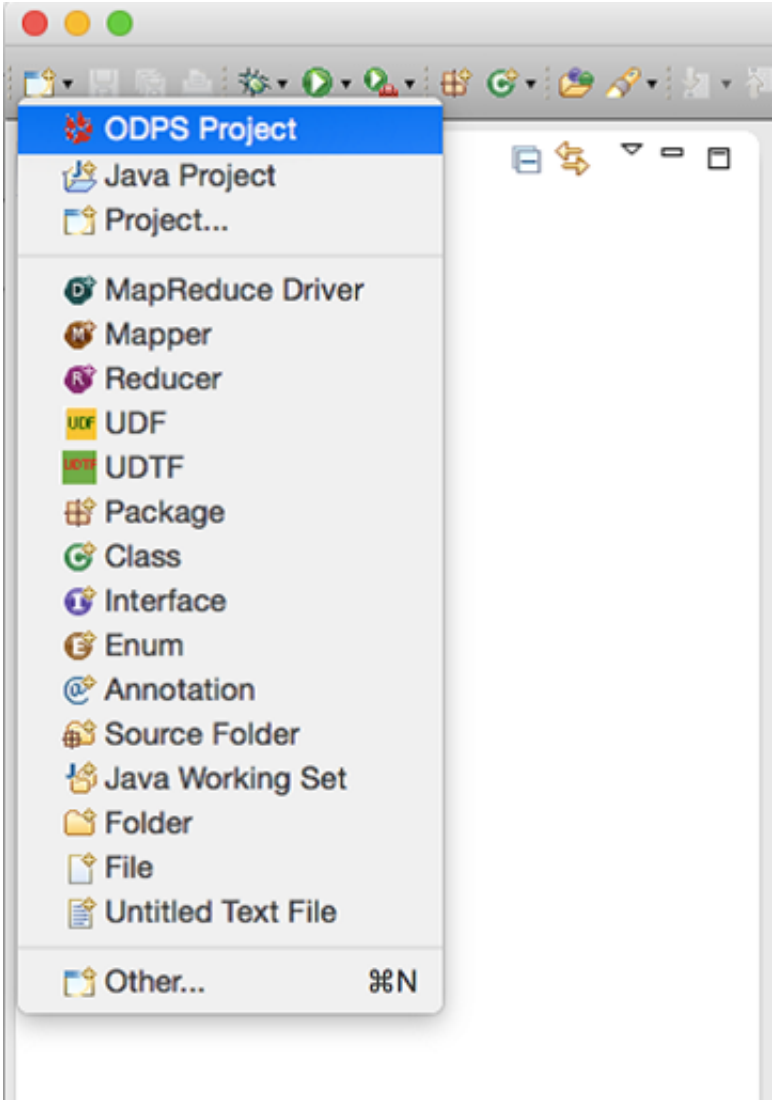
26.14.2.2 创建工程

26.14.2.2.1 方式一

操作步骤

1. 在Eclipse左上角选择**文件(File)** > **新建(New)** > **Project** > **ODPS** > **ODPS Project**创建工程，如下图所示。

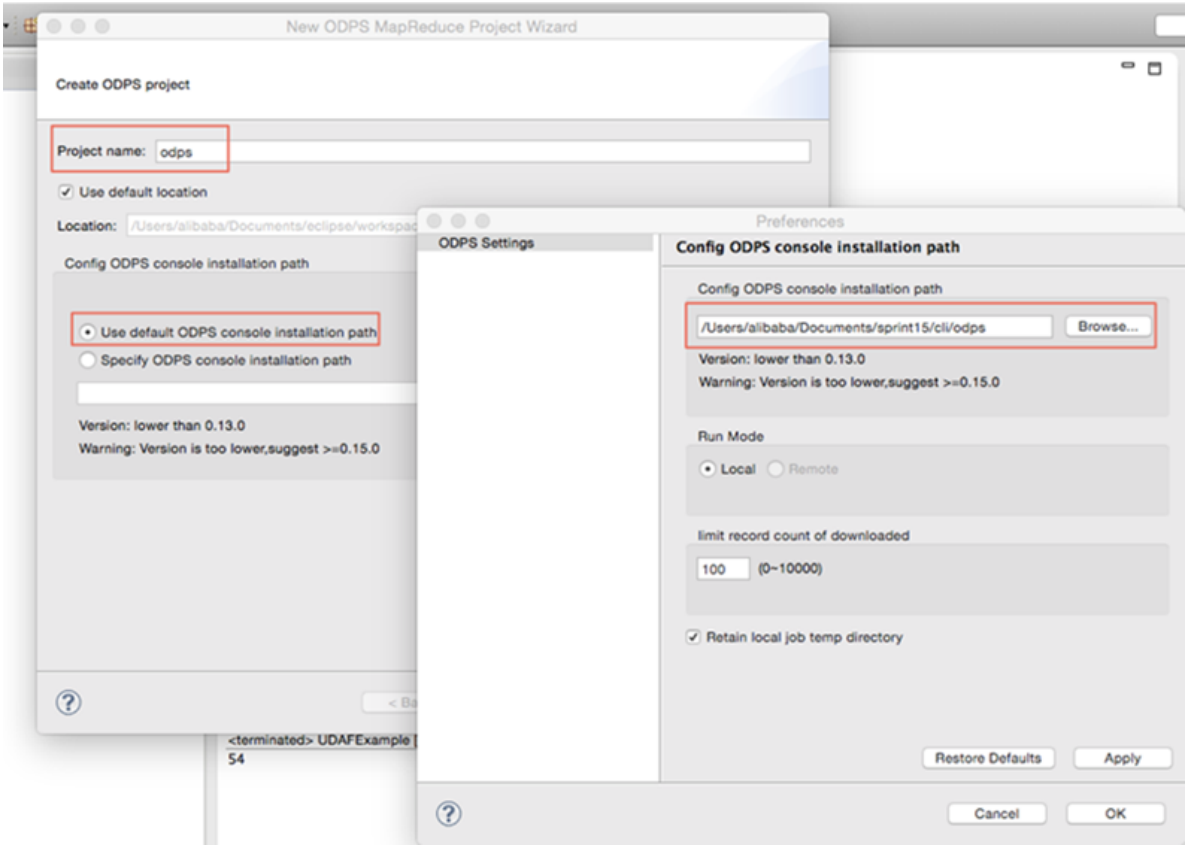
   图 785: Eclipse创建数据库步骤1



说明: 示例中使用ODPS作为工程名。

2. 创建ODPS工程后会出现如下图所示对话框。输入**Project name**，选择MaxCompute客户端路径，单击**Finish**并确认。

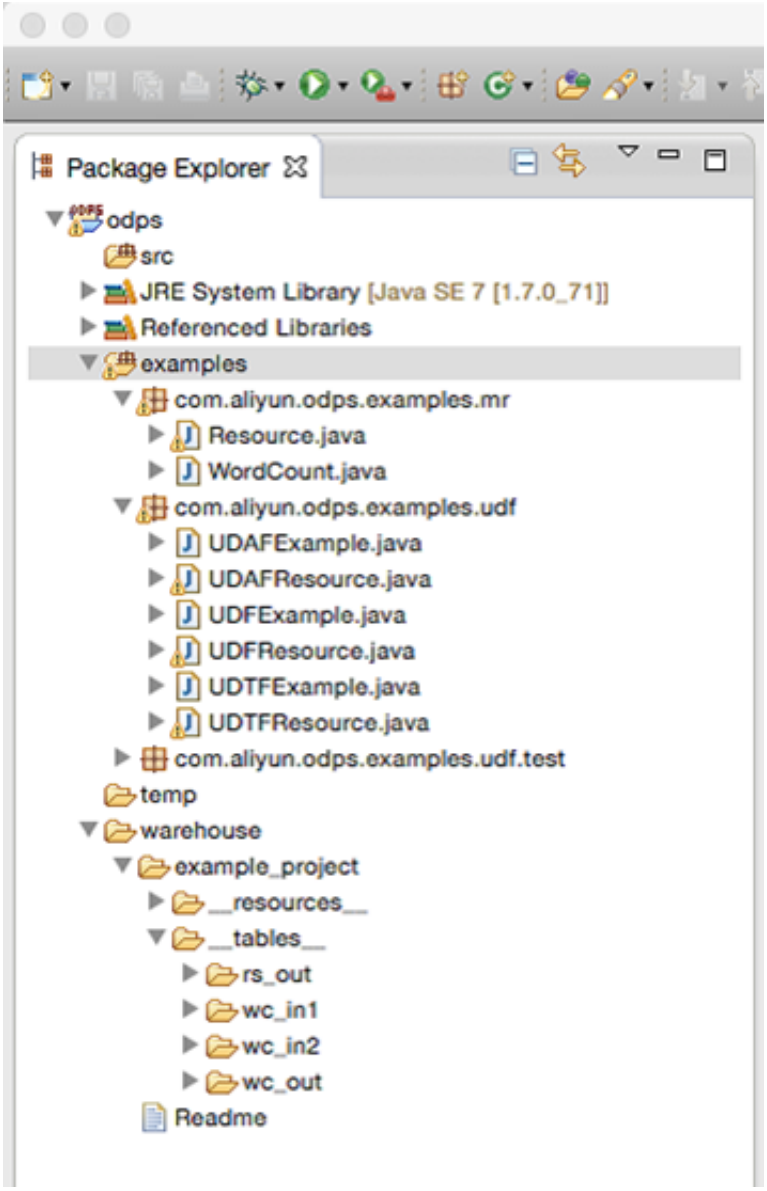
图 786: Eclipse创建数据库步骤2



说明: 客户端需要提前安装。

3. 创建好工程后，在左侧包资源管理器（Package Explorer）中可以看到如下图所示的目录结构。

图 787: Eclipse创建数据库步骤3

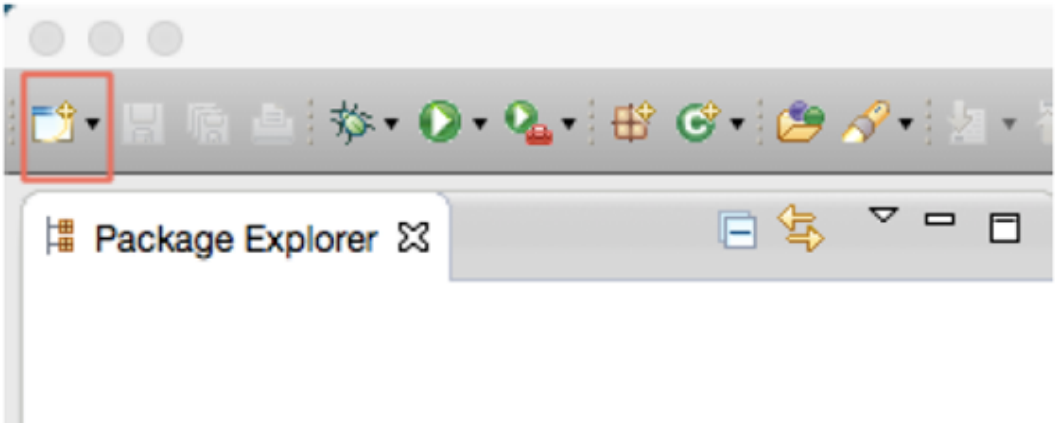


26.14.2.2.2 方式二

操作步骤

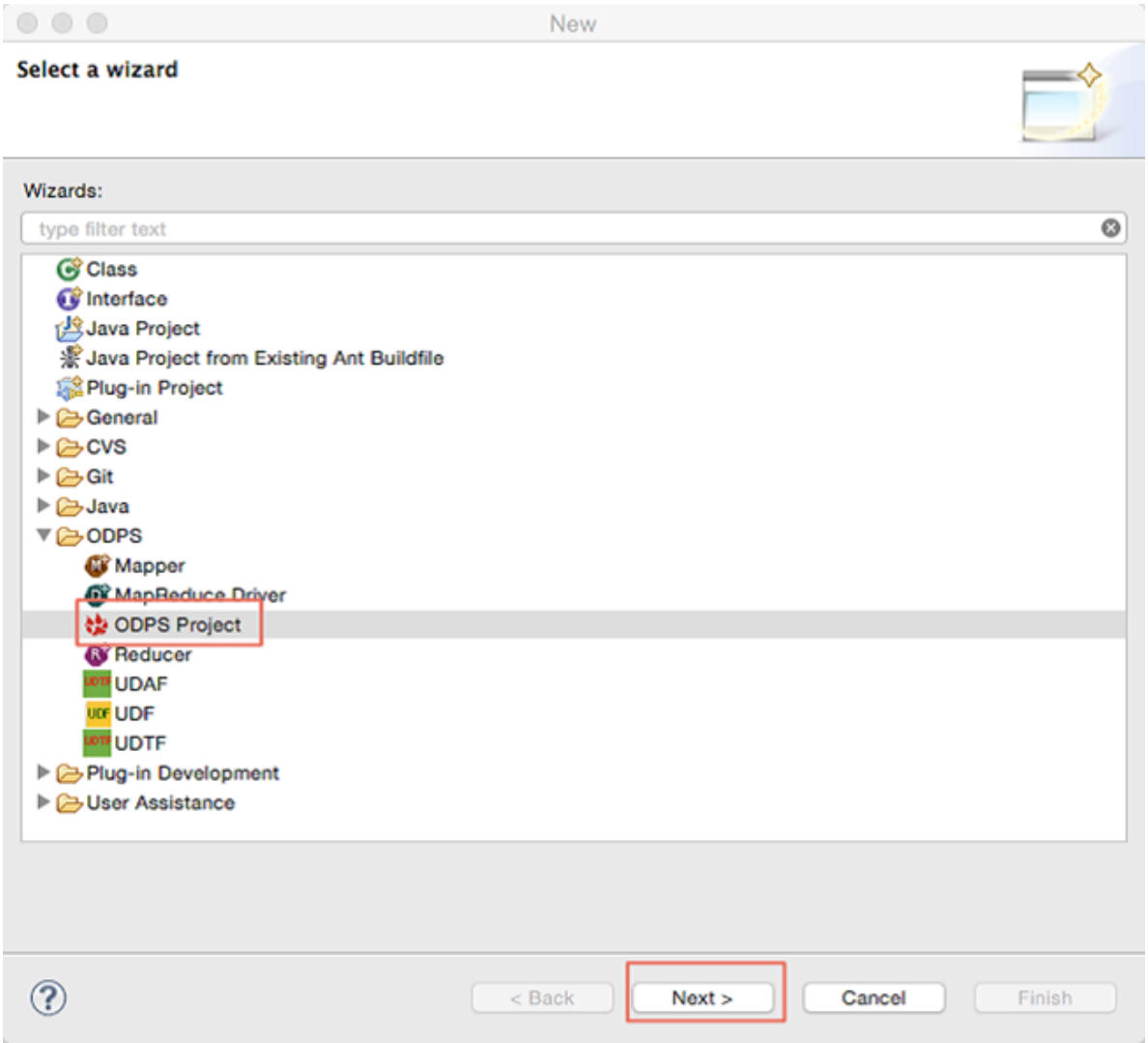
1. 在Eclipse直接单击左上角的**新建**，如下图所示。

图 788: Eclipse创建工程步骤1



2. 在弹出的对话框中，选择**ODPS Project**，单击**下一步**，如下图所示。

图 789: Eclipse创建工程步骤2



说明: 示例中使用ODPS作为工程名。

3. 后续操作同方式一中的相关步骤。Eclipse插件安装结束后。用户即可以使用此插件编写MapReduce或者UDF程序。

说明: 关于插件中对MapReduce的功能介绍请参考：*MapReduce开发插件介绍*，UDF的程序编写请参考：*UDF开发插件介绍*。

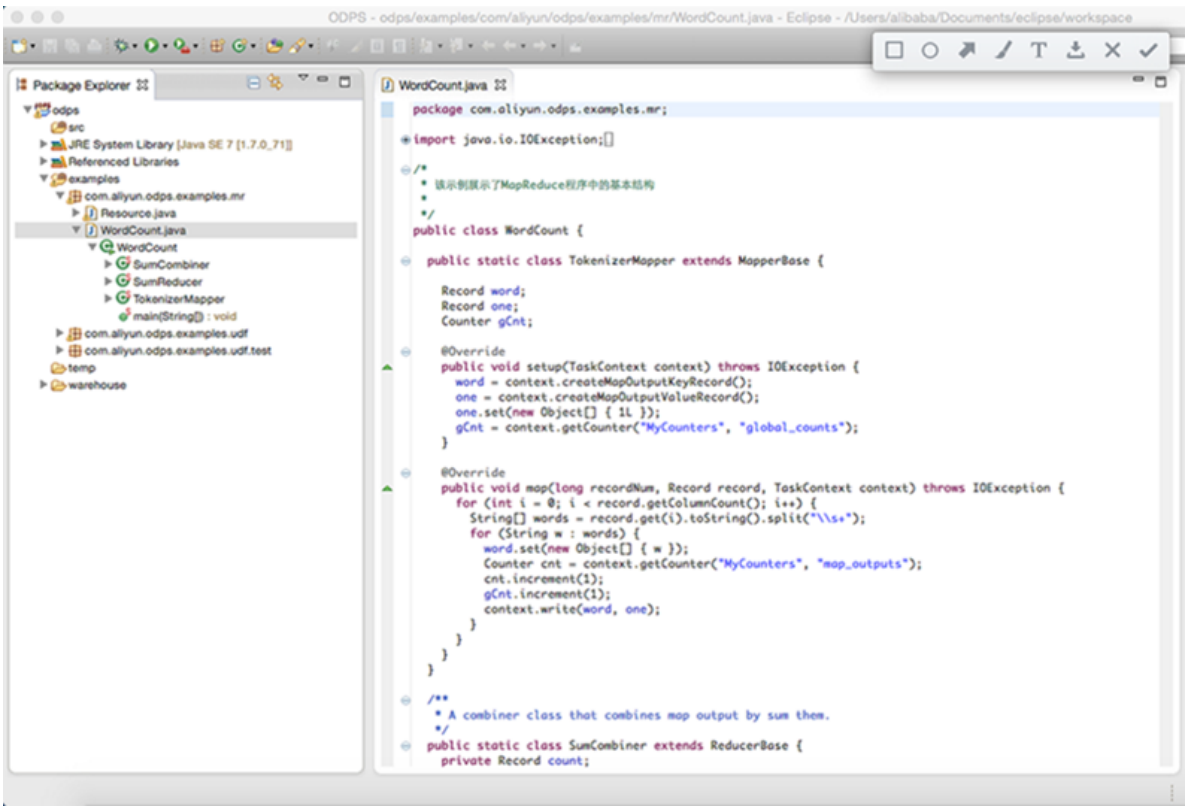
26.14.2.3 MapReduce开发插件介绍

26.14.2.3.1 快速运行WordCount示例

操作步骤

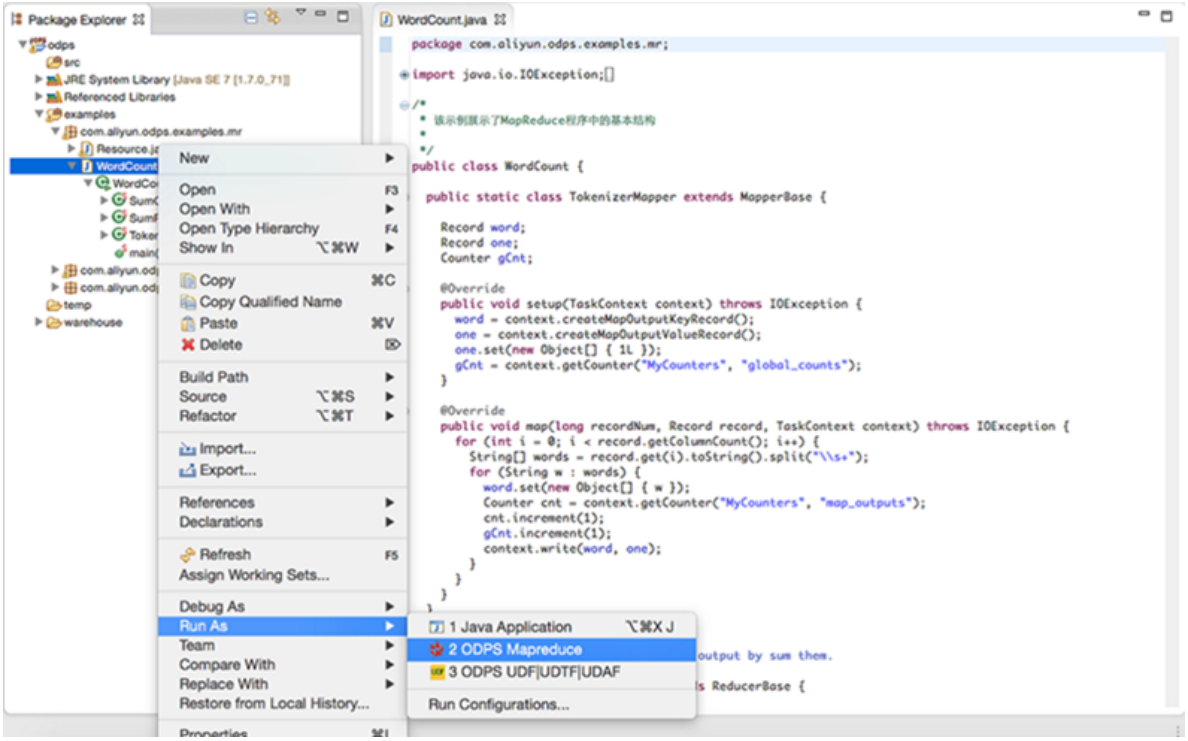
1. 选择ODPS项目中的WordCount示例，如下图所示。

图 790: WordCount 示例



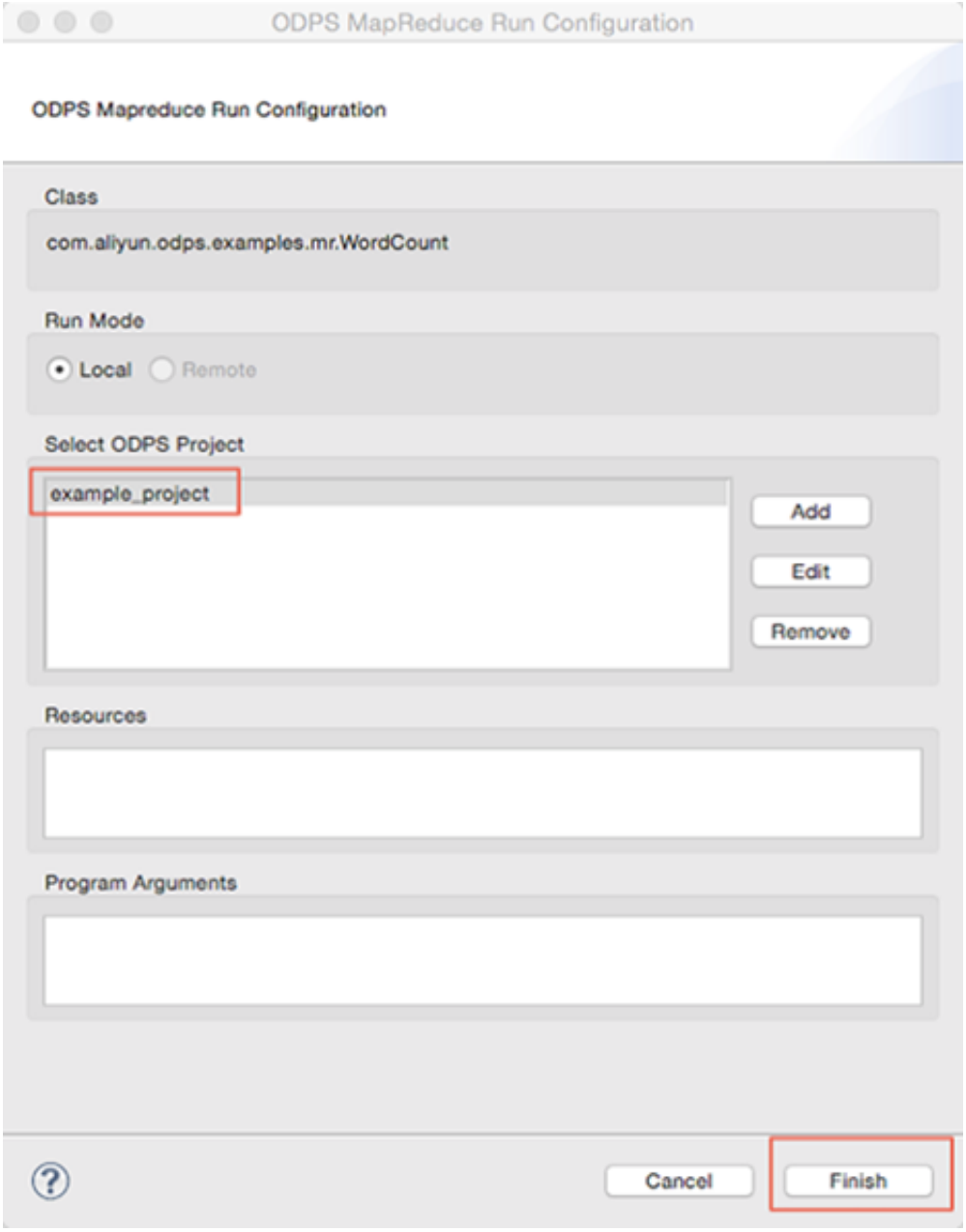
2. 右键单击**WordCount.java**，依次选择**Run AsODPS MapReduce**，如下图所示。

图 791: 运行WordCount示例



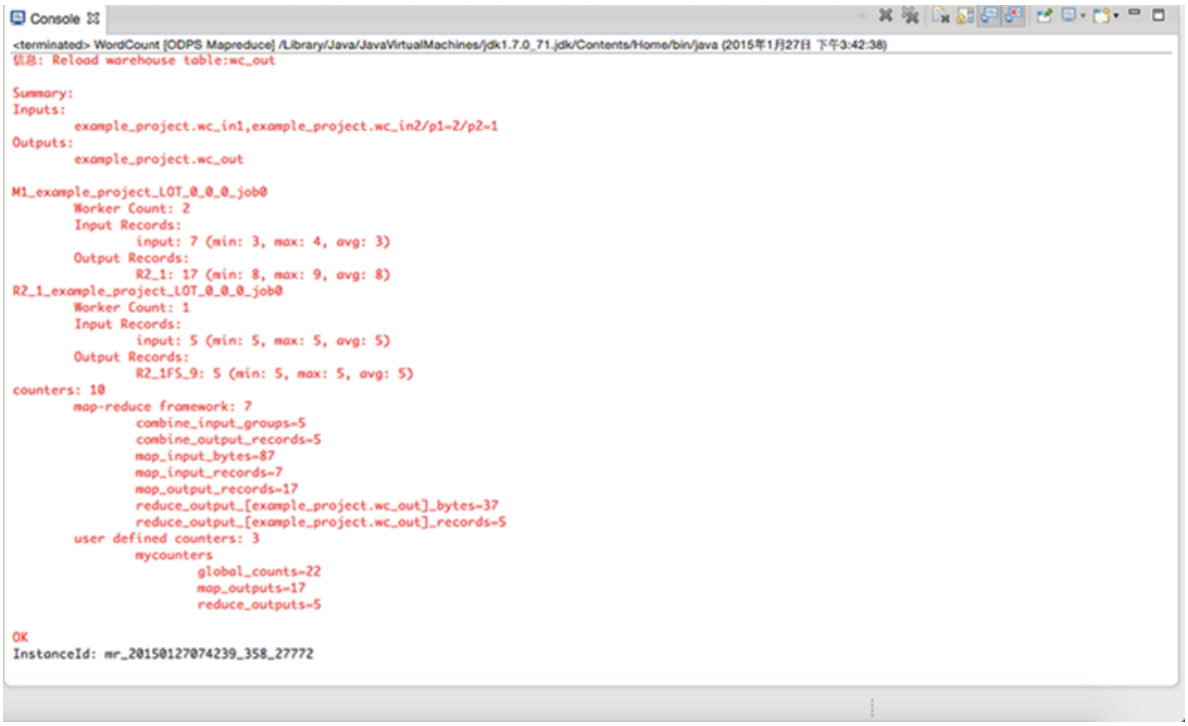
3. 在弹出的对话框中，选择**example_project**，单击**确认**，如下图所示。

图 792: 运行WordCount示例



4. 运行成功后，会出现如下图所示的结果提示。

图 793: WordCount示例运行结果

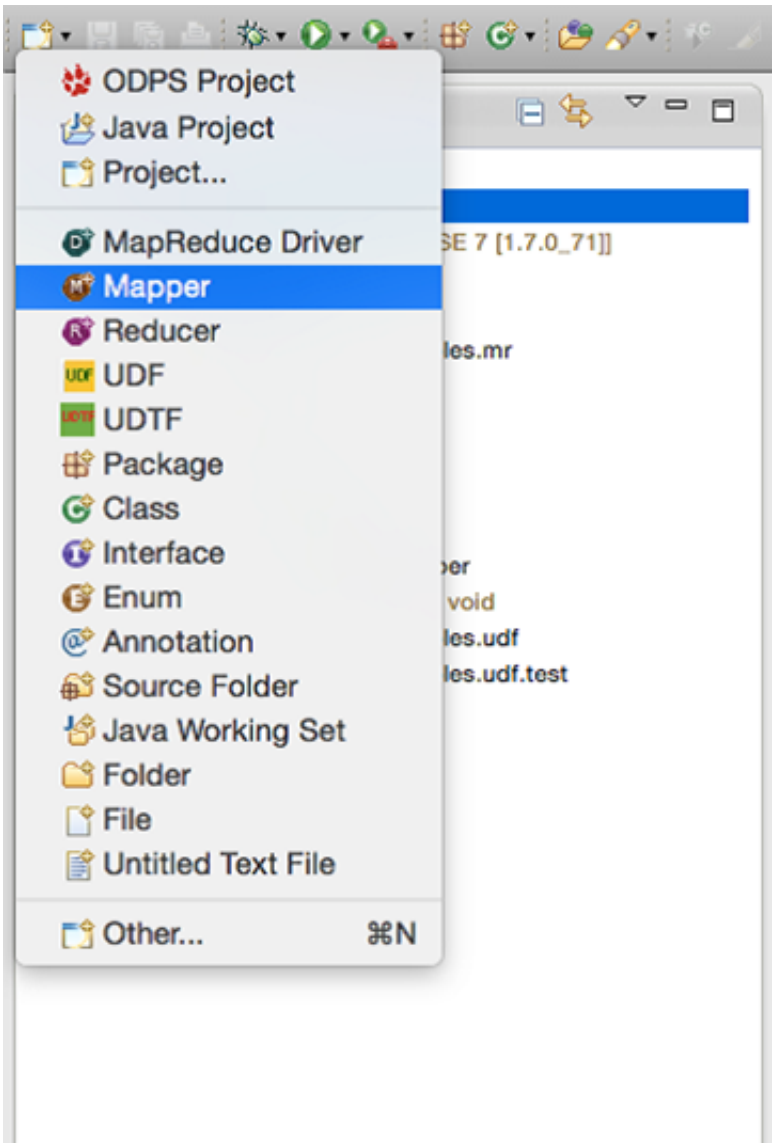


26.14.2.3.2 运行自定义MapReduce程序

操作步骤

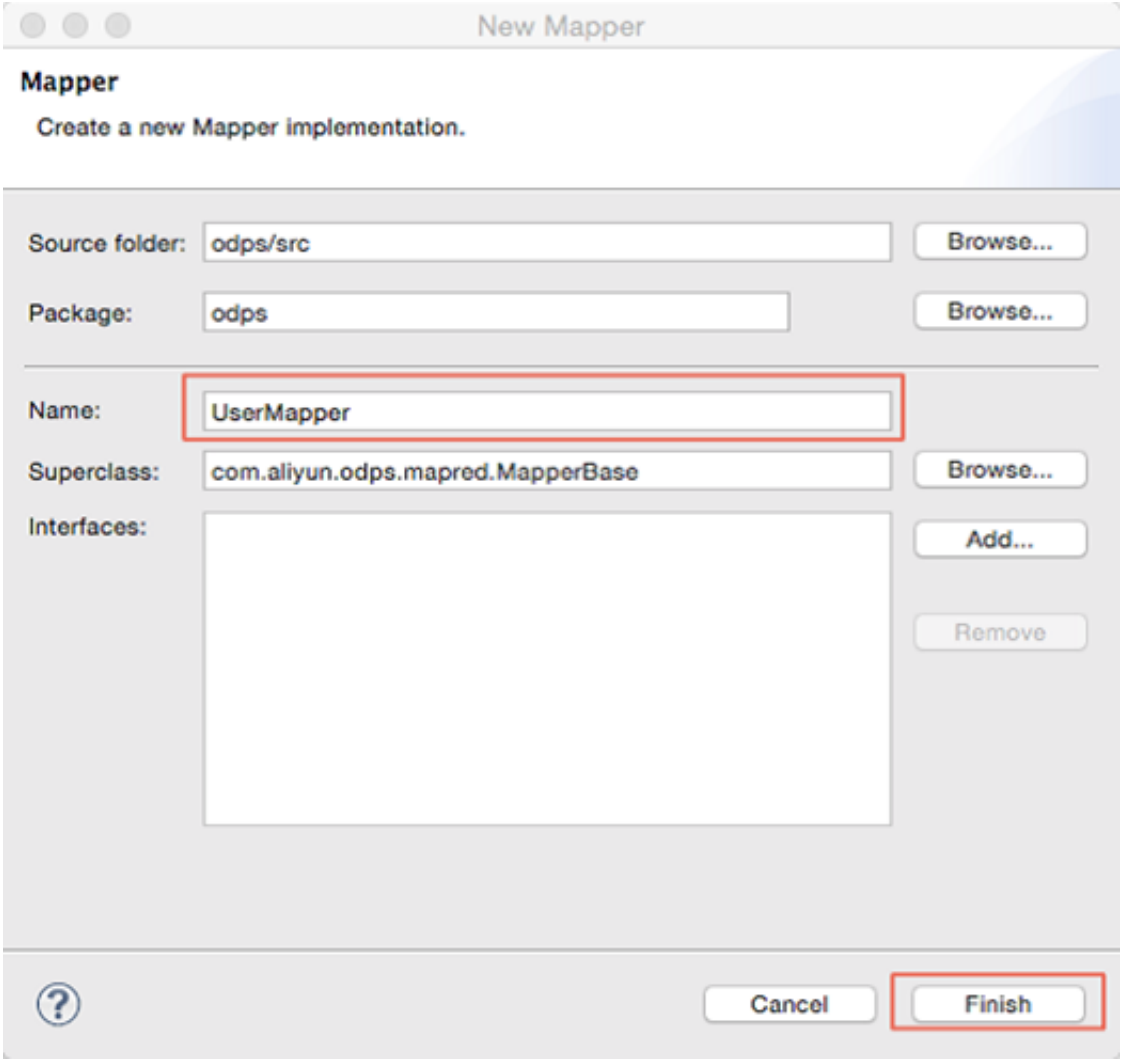
1. 在Eclipse中右键选择**src**目录，选择**New** > **Mapper**，如下图所示。

图 794: 运行自定义MapReduce程序步骤1



2. 选择Mapper后出现如下的对话框。输入Mapper类的名字，并确认。

图 795: 运行自定义MapReduce程序步骤2



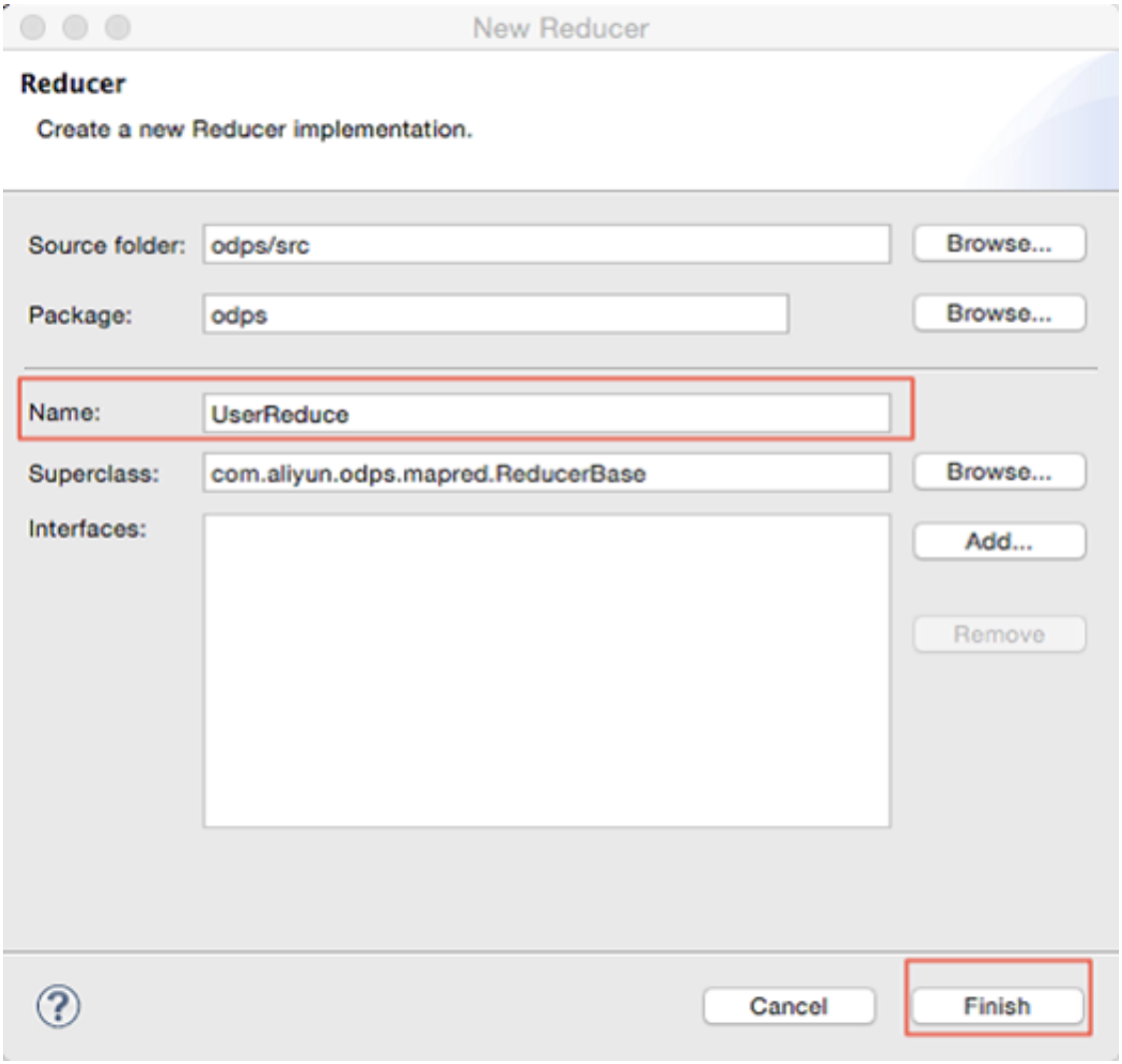
3. 在左侧包资源管理器（Package Explorer）中，src目录下生成文件UserMapper.java。该文件的内容即是一个Mapper类的模板。将模板中package名称默认配置为odps，编写模板内容如下。

```
package odps;
import java.io.IOException;
import com.aliyun.odps.counter.Counter; import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.MapperBase;
public class UserMapper extends MapperBase {
Record word; Record one; Counter gCnt;
@Override
public void setup(TaskContext context) throws IOException {
word = context.createMapOutputKeyRecord(); one = context.createMapOutputValueRecord();
 one.set(new Object[] { 1L });
gCnt = context.getCounter("MyCounters", "global_counts");
}
@Override
public void map(long recordNum, Record record, TaskContext context) throws IOException {
for (int i = 0; i < record.getColumnCount(); i++) { String[] words = record.get(i).toString().split("\
\s+"); for (String w : words) {
word.set(new Object[] { w });
Counter cnt = context.getCounter("MyCounters", "map_outputs"); cnt.increment(1);
gCnt.increment(1); context.write(word, one);
}
```

```
}
}
@Override
public void cleanup(TaskContext context) throws IOException {
}
}
```

4. 在Eclipse中右键选择**src**目录，选择**New** > **Reduce**，如下图所示。

图 796: Reducer



5. 在弹出的对话框中输入Reduce类的名字，并确认。

说明: 本示例使用UserReducer。

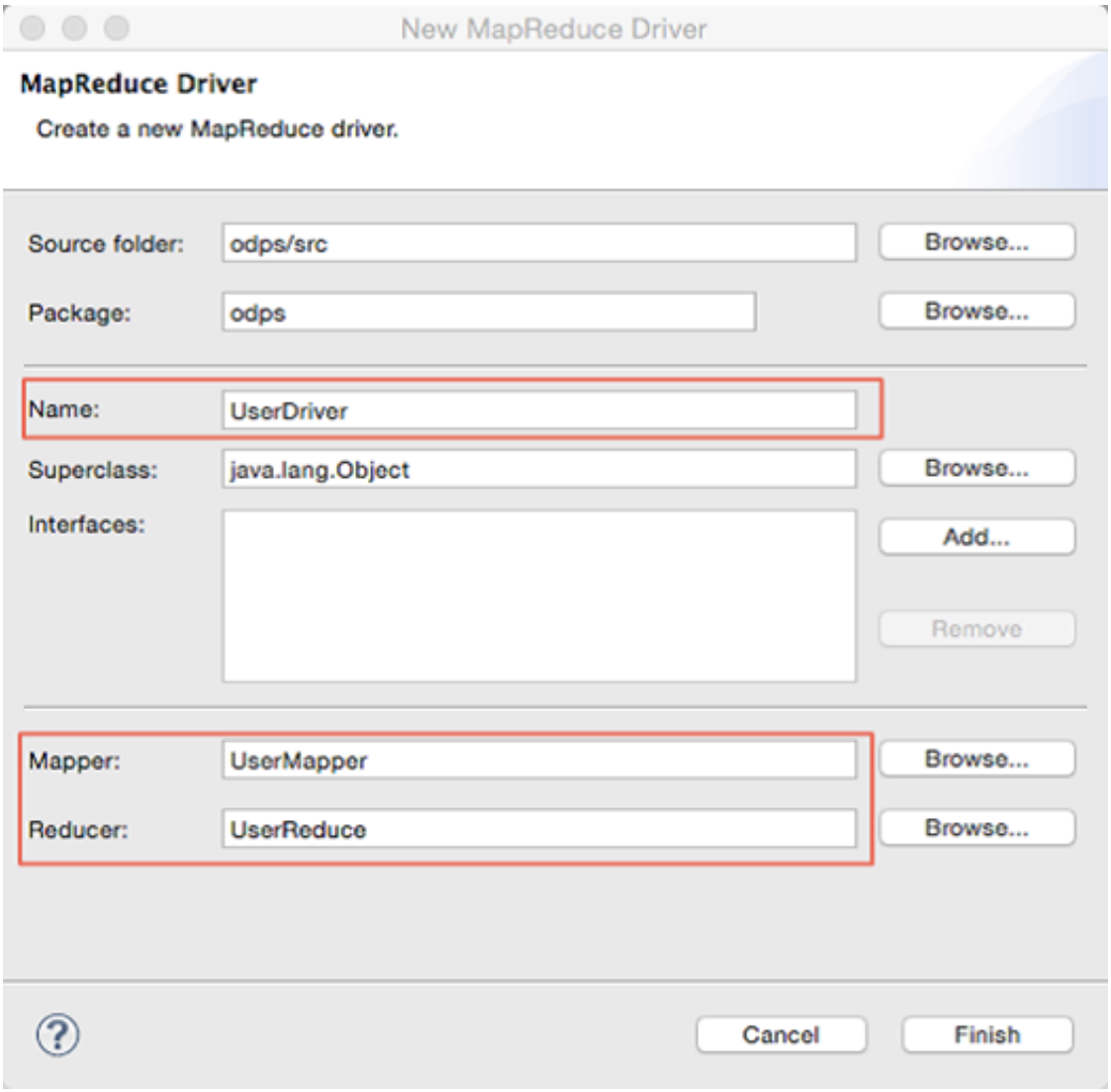
6. 在左侧包资源管理器（Package Explorer）中，src目录下生成文件UserReducer.java。该文件的内容即是一个Reduce类的模板，将模板中package名称默认配置为odps，编写模板内容如下。

```
package odps;
import java.io.IOException;
```

```
import java.util.Iterator;
import com.aliyun.odps.counter.Counter;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.ReducerBase;
public class UserReducer extends ReducerBase {
private Record result; Counter gCnt;
@Override
public void setup(TaskContext context) throws IOException { result = context.createOutputRecord();
gCnt = context.getCounter("MyCounters", "global_counts");
}
@Override
public void reduce(Record key, Iterator<Record> values, TaskContext context) throws IOException {
long count = 0;
while (values.hasNext()) { Record val = values.next(); count += (Long) val.get(0);
}
result.set(0, key.get(0)); result.set(1, count);
Counter cnt = context.getCounter("MyCounters", "reduce_outputs"); cnt.increment(1);
gCnt.increment(1);
context.write(result);
}
@Override
public void cleanup(TaskContext context) throws IOException {
}
}
```

7. 在Eclipse中右键选择**src**目录，选择**New** > **MapReduce Driver**。
8. 选择MapReduce Driver后出现如下的对话框。输入Driver Name，Mapper及Recducer类，并确认。

图 797: MapReduce Driver



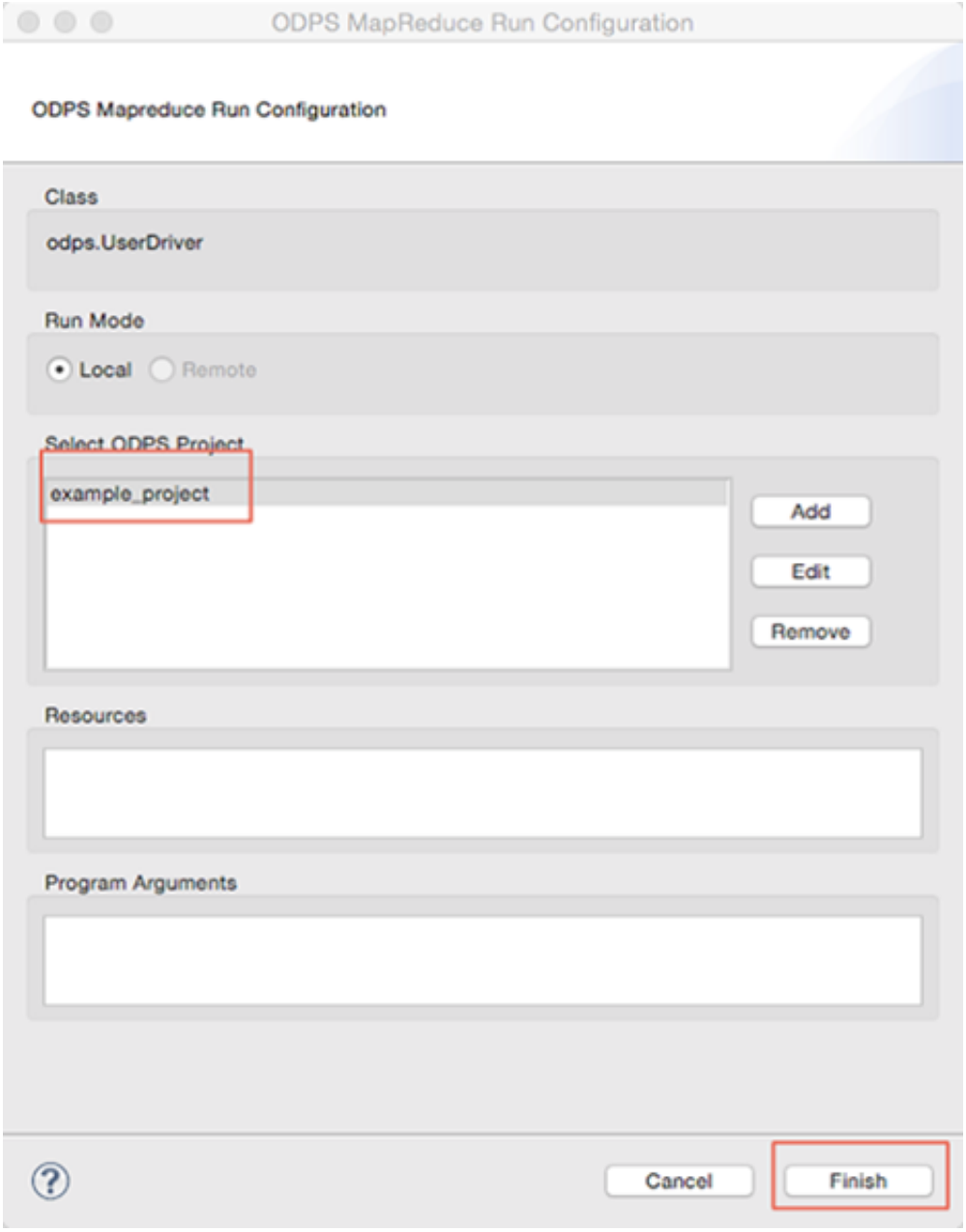
9. 在左侧包资源管理器（Package Explorer）中，src目录下生成文件MyDriver.java。该文件的内容即是一个MapReduce Driver的模板。将模板中package名称默认配置为odps，编写模板内容如下。

```
package odps;
import com.aliyun.odps.OdpsException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.examples.mr.WordCount.SumCombiner;
import com.aliyun.odps.examples.mr.WordCount.SumReducer;
import com.aliyun.odps.examples.mr.WordCount.TokenizerMapper;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.RunningJob;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
public class UserDriver {
public static void main(String[] args) throws OdpsException { JobConf job = new JobConf();
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(SumCombiner.class);
```

```
job.setReducerClass(SumReducer.class);
job.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
job.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
InputUtils.addTable(
TableInfo.builder().tableName("wc_in1").cols(new String[] { "col2", "col3" }).build(), job);
InputUtils.addTable(TableInfo.builder().tableName("wc_in2").partSpec("p1=2/p2=1").build(),
 job);
OutputUtils.addTable(TableInfo.builder().tableName("wc_out").build(), job);
RunningJob rj = JobClient.runJob(job); rj.waitForCompletion();
}
}
```

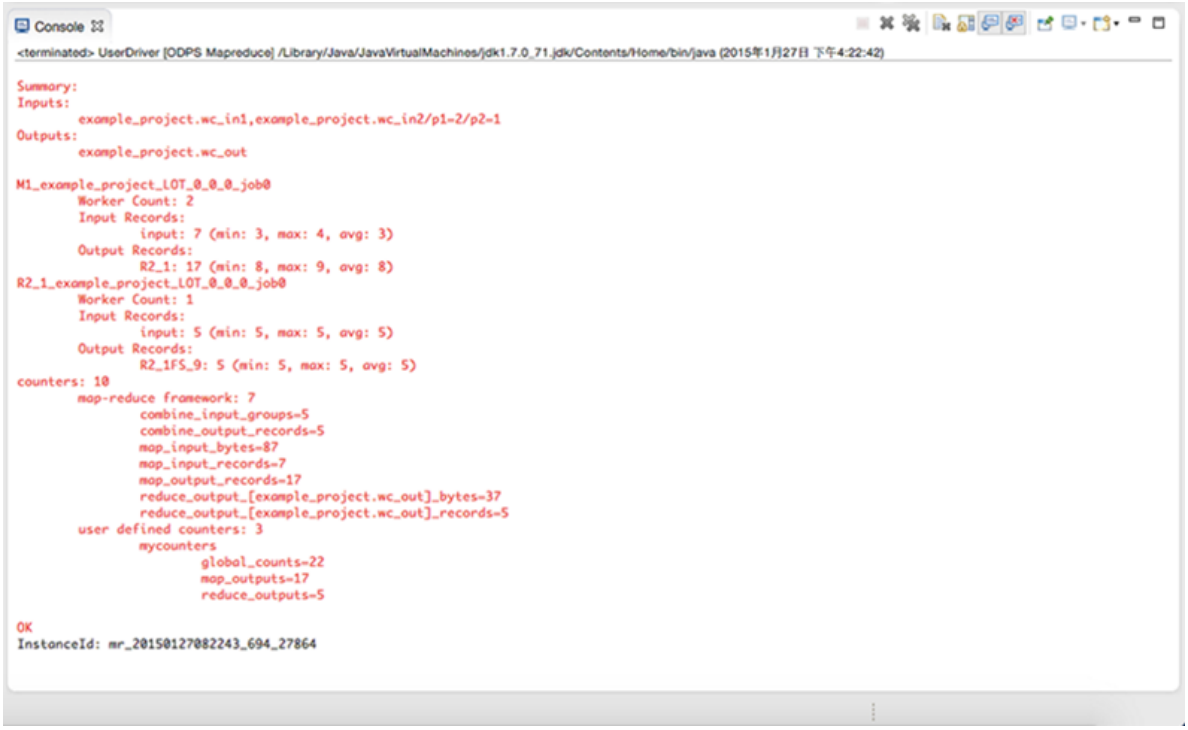
10. 运行MapReduce程序，选中**UserDriver.java**，右键选择**Run As** > **ODPS MapReduce** ，单击**确认**。出现如下图所示对话框。

图 798: ODPS MapReduce Run Configuration



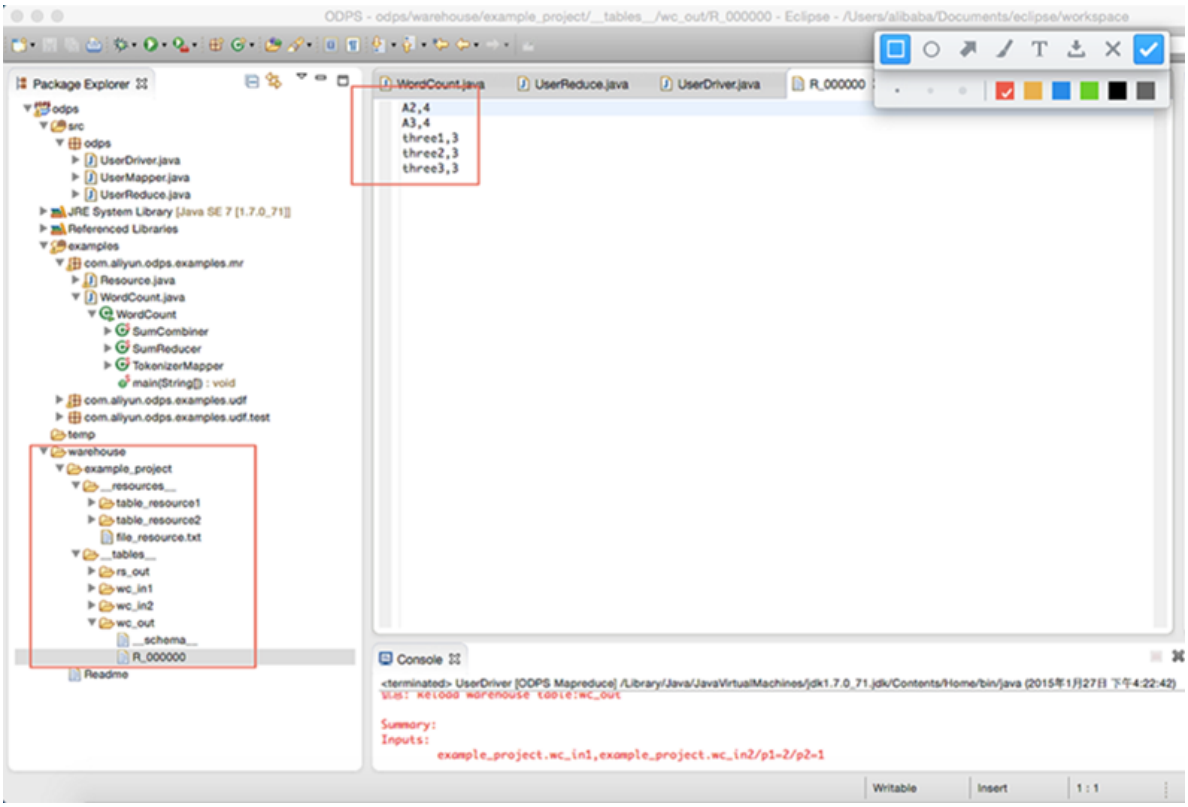
11.选择ODPS　Project为**example_project**，单击**Finish**开始本地运行MapReduce程序，有如下图所示的输出信息，说明本地运行成功。

图 799: Console



12.运行的输出结果在warehouse目录下，刷新ODPS工程，如下图所示。

图 800: 输出结果

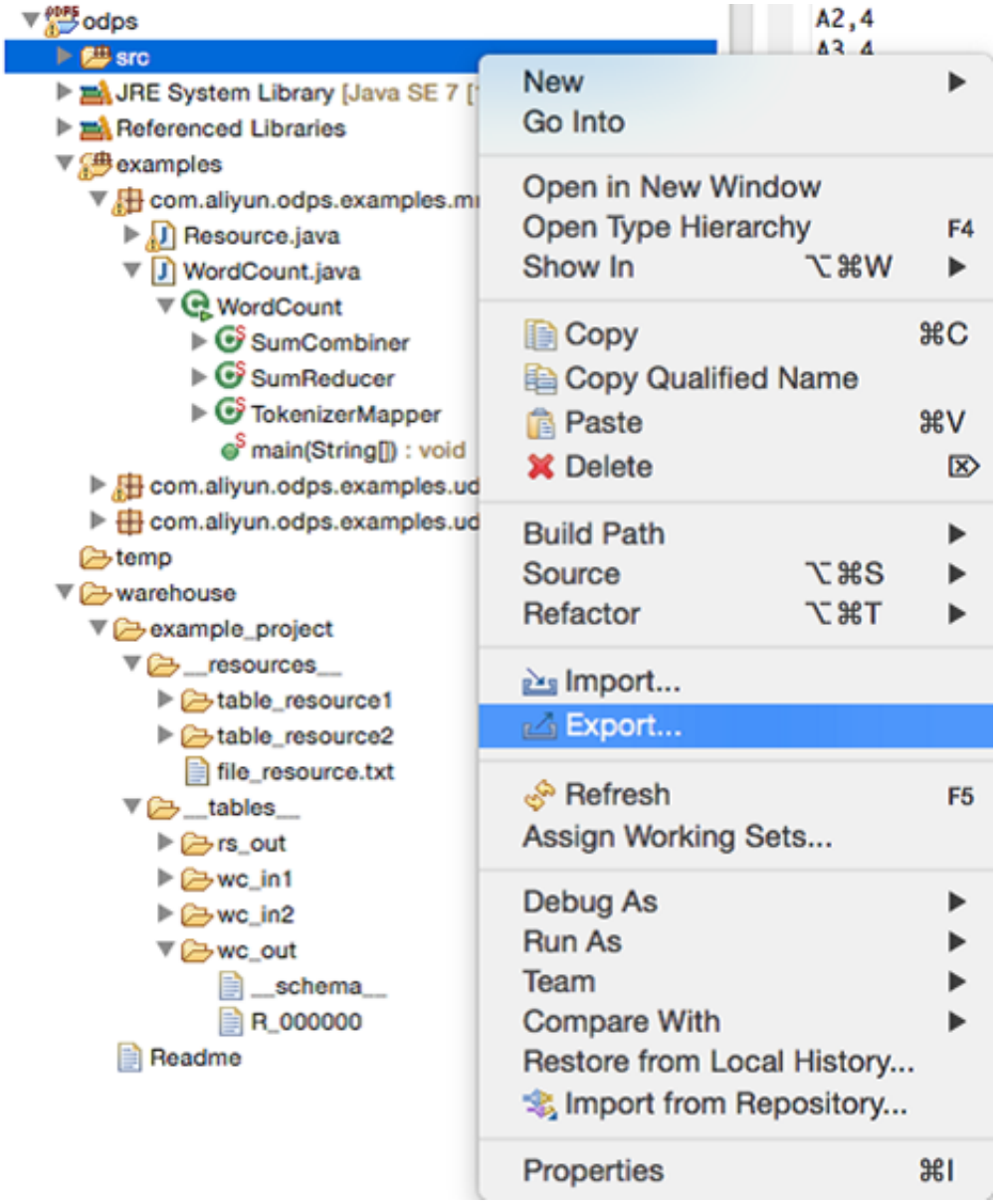


说明: wc_out即是输出目录，R_000000即是结果文件。通过本地调试，确定输出结果正确后，可以通过Eclipse导出（Export）功能将MapReduce打包，供后续分布式环境使用。

13.导出步骤如下所示。

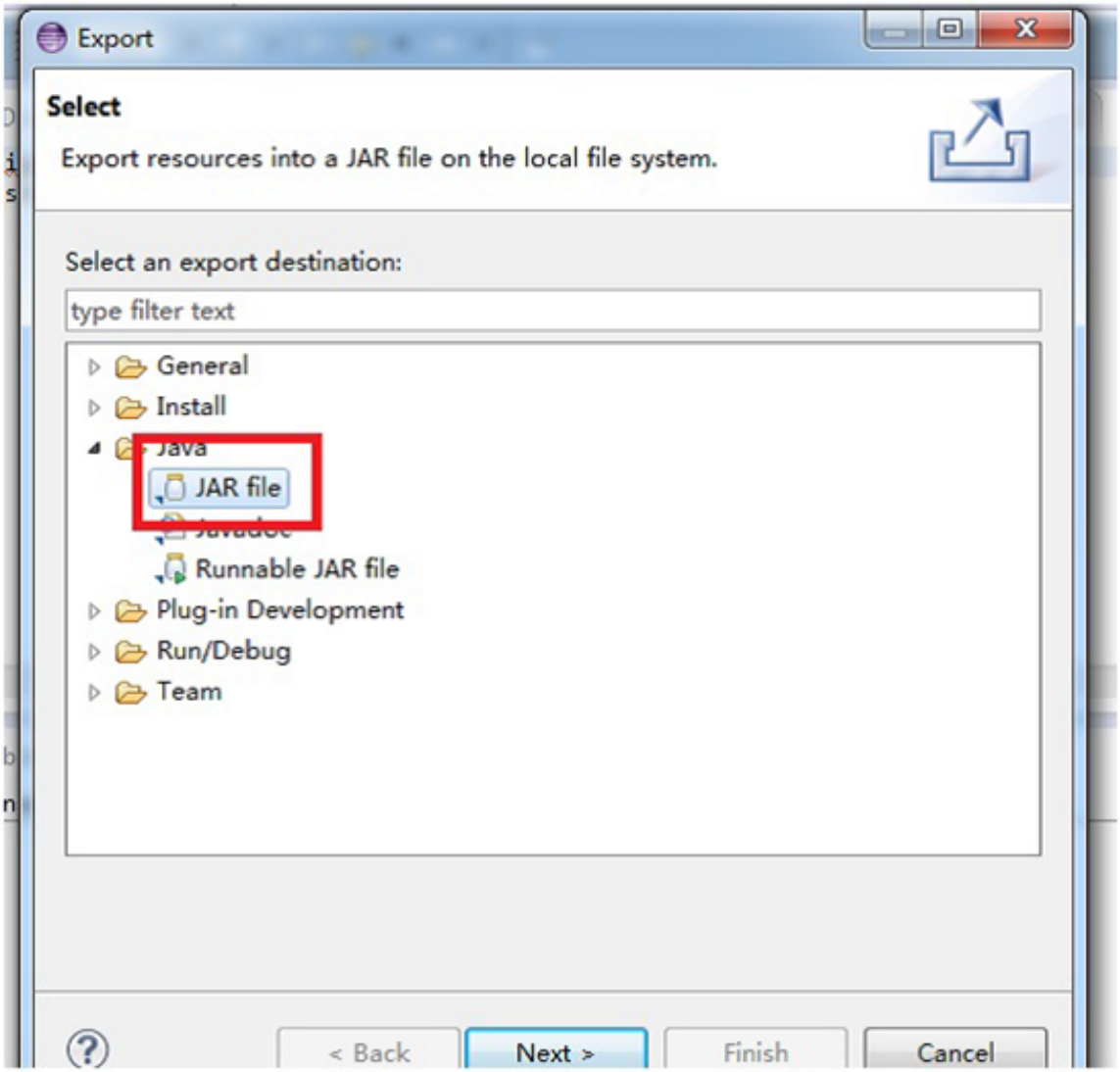
1. 选择**src**目录，单击**Export**，如下图所示。

图 801: 导出文件步骤1



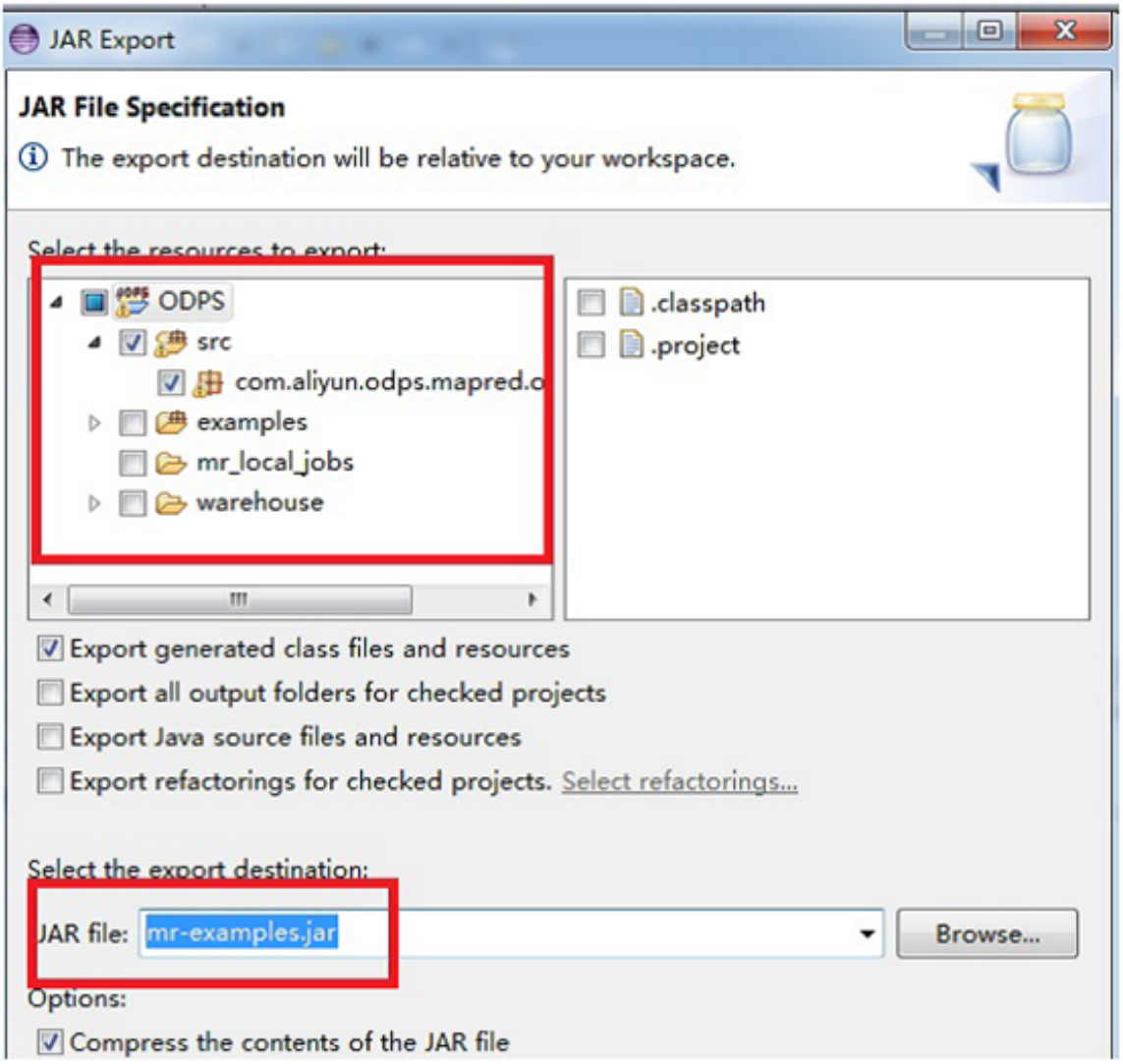
2. 选择导出模式为**Jar File**，如下图所示。

图 802: 导出文件步骤2



3. 仅需要导出**src**目录下的package（com.aliyun.odps.mapred.open.example），Jar File名称指定为mr- examples.jar ，如下图所示。

图 803: 导出文件步骤3



说明: 在本示例中，将程序包命名为mr-examples.jar，实际操作时可根据用户实际需求而定。

4. 确认后，导出成功。

14.如果用户想在本地模拟新建Project，可以在warehouse下面，创建一个新的子目录（与example_project平级的目录），目录层次结构如下图所示。

```
<warehouse>
|___example_project(项目空间目录)
    |___ <_tables_>
    |    |_table_name1(非分区表)
    |    |    |___ data(文件)
    |    |    |
    |    |    |___ <_schema_> (文件)
    |    |
    |    |_table_name2(分区表)
    |        |___ partition_name=partition_value(分区目录)
    |        |        |___ data(文件)
    |        |
    |        |___ <_schema_> (文件)
    |
    |___ <_resources_>
        |
        |__table_resource_name (表资源)
        |     |___<_ref_>
        |
        |__ file_resource_name ( 文件资源 )
```

schema文件示例如下：

非分区表：

```
project=project_name table=table_name
columns=col1:BIGINT,col2:DOUBLE,col3:BOOLEAN,col4:DATETIME,col5:STRING
分区表：project=project_name table=table_name
columns=col1:BIGINT,col2:DOUBLE,col3:BOOLEAN,col4:DATETIME,col5:STRING
 partitions=col1:BIGINT,col2:DOUBLE,col3:BOOLEAN,col4:DATETIME,col5:STRING
```

注：当前支持5种数据格式：bigint,double,boolean,datetime,string，对应到java中的数据类型- long,double,boolean,java.util.Date,java.lang.String。

data文件示例如下：

1,1.1,true,2015-06-04 11:22:42 896,hello world
\N,\N,\N,\N,\N
注：时间格式精确到毫秒级别，所有类型用\N表示null。

说明:

- 本地模式运行MapReduce程序，默认情况下先到warehouse下查找相应的数据表或资源，如果表或资源不存在会到服务器上下载相应的数据存入warehouse目录下，再以本地模式运行。
- 运行完MapReduce后，需刷新warehouse目录，才能看到生成的结果。

26.14.2.4 UDF开发插件介绍

26.14.2.4.1 简介

MaxCompute Eclipse插件可用于辅助用户调试UDF|UDTF|UDAF（以下简称UDF）程序，验证UDF程序逻辑的正确性。在本章节将介绍如何使用Eclipse插件开发并在本地运行UDF。UDAF，UDTF的编写执行过程与UDF类似，均可参考UDF的示例完成。

26.14.2.4.2 Local Debug UDF程序

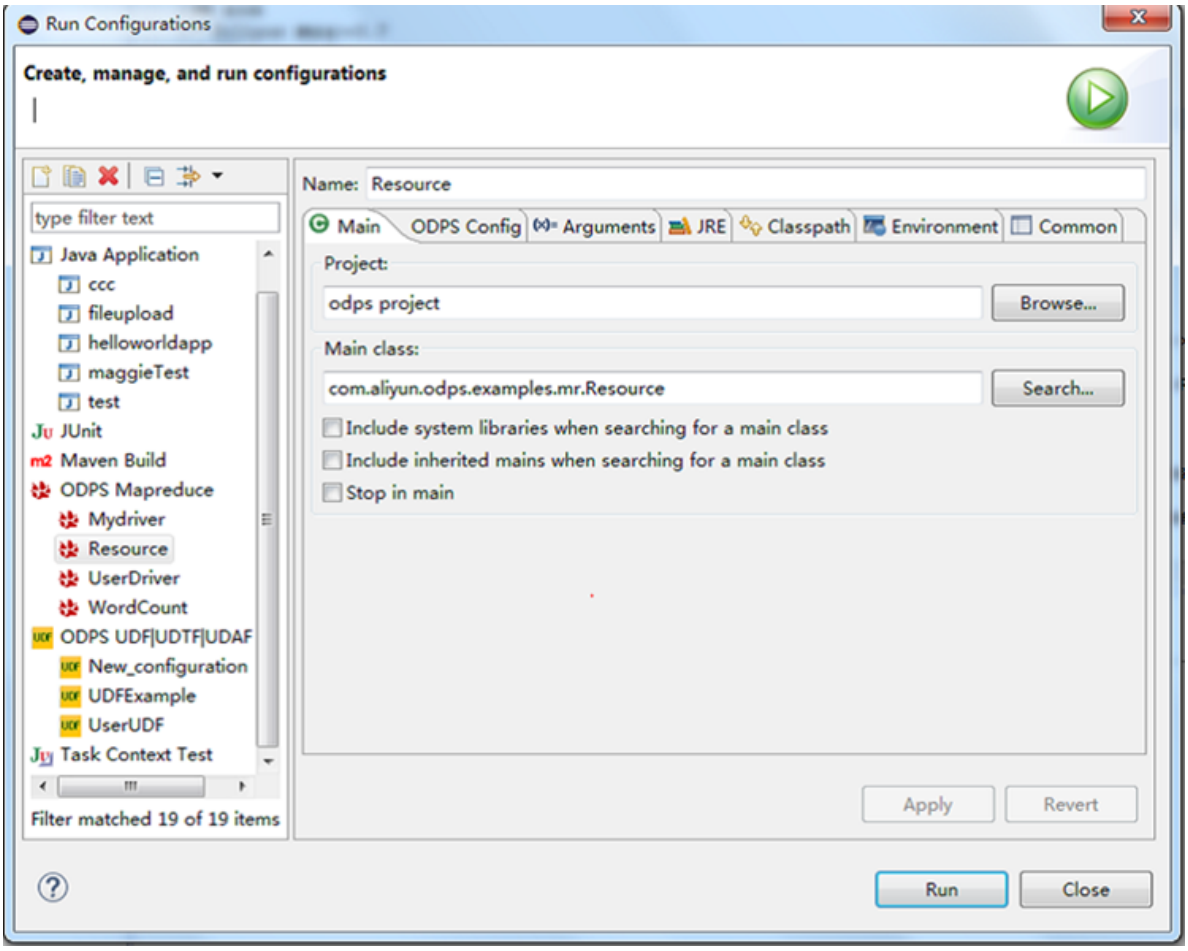
MaxCompute Eclipse插件提供两种运行UDF的方式：菜单栏和右键单击快速运行方式。

26.14.2.4.2.1 菜单栏运行

操作步骤

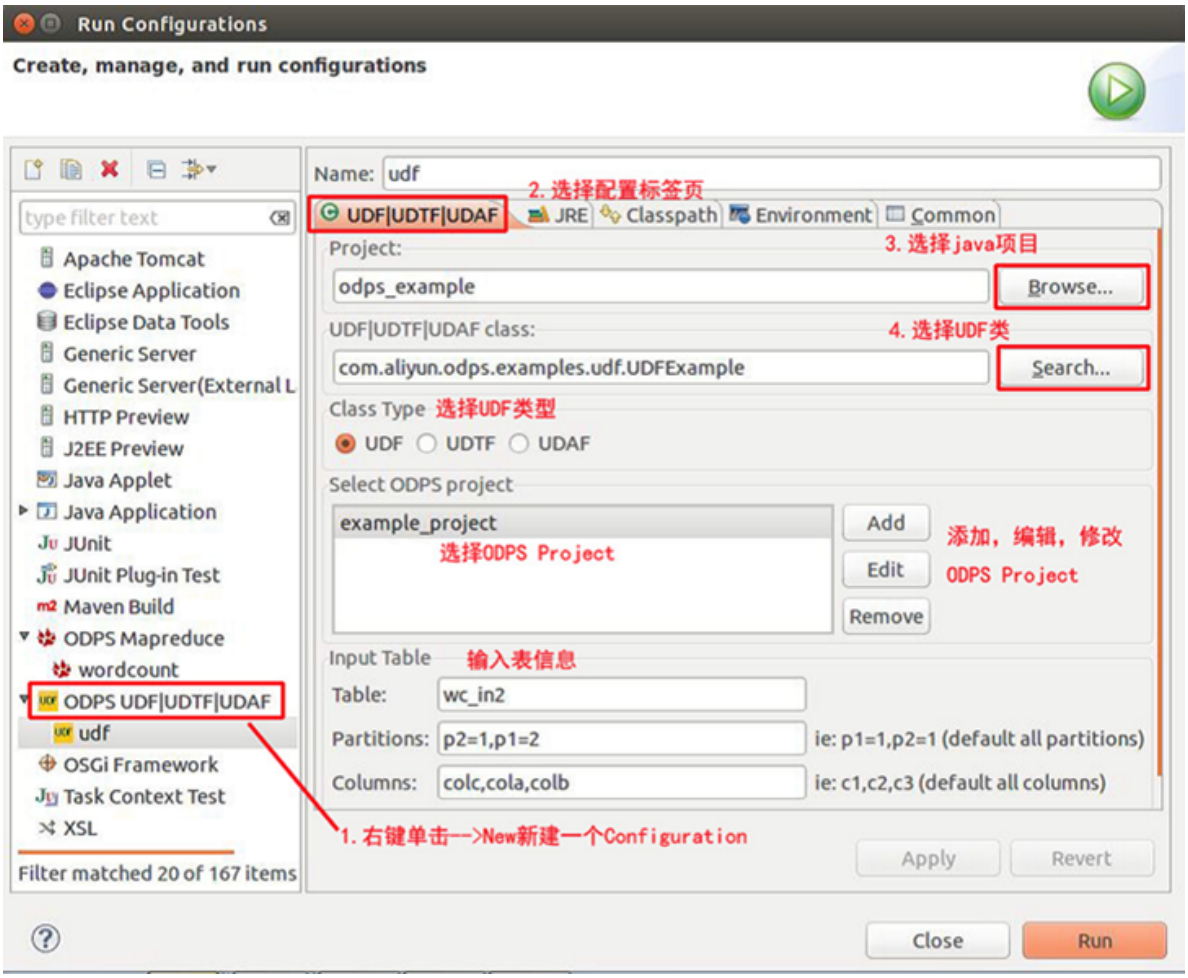
1. 从菜单栏选择**Run** > **Run Configurations**，弹出如下图所示的对话框。

图 804: Run Configurations



2. 用户可以新建一个Run Configuration，选择运行的UDF类及类型、选择ODPS Project、填写输入表信息，如下图所示。

图 805: Run Configurations 2



说明: 上述配置中，Table表示UDF的输入表，Partitions表示读取某个分区下的数据，分区由逗号分隔，Columns表示列，将依次作为UDF函数的参数被传入，列名由逗号分隔。

3. 单击**Run**，运行结果将显示在控制台中，如下图所示。

图 806: console

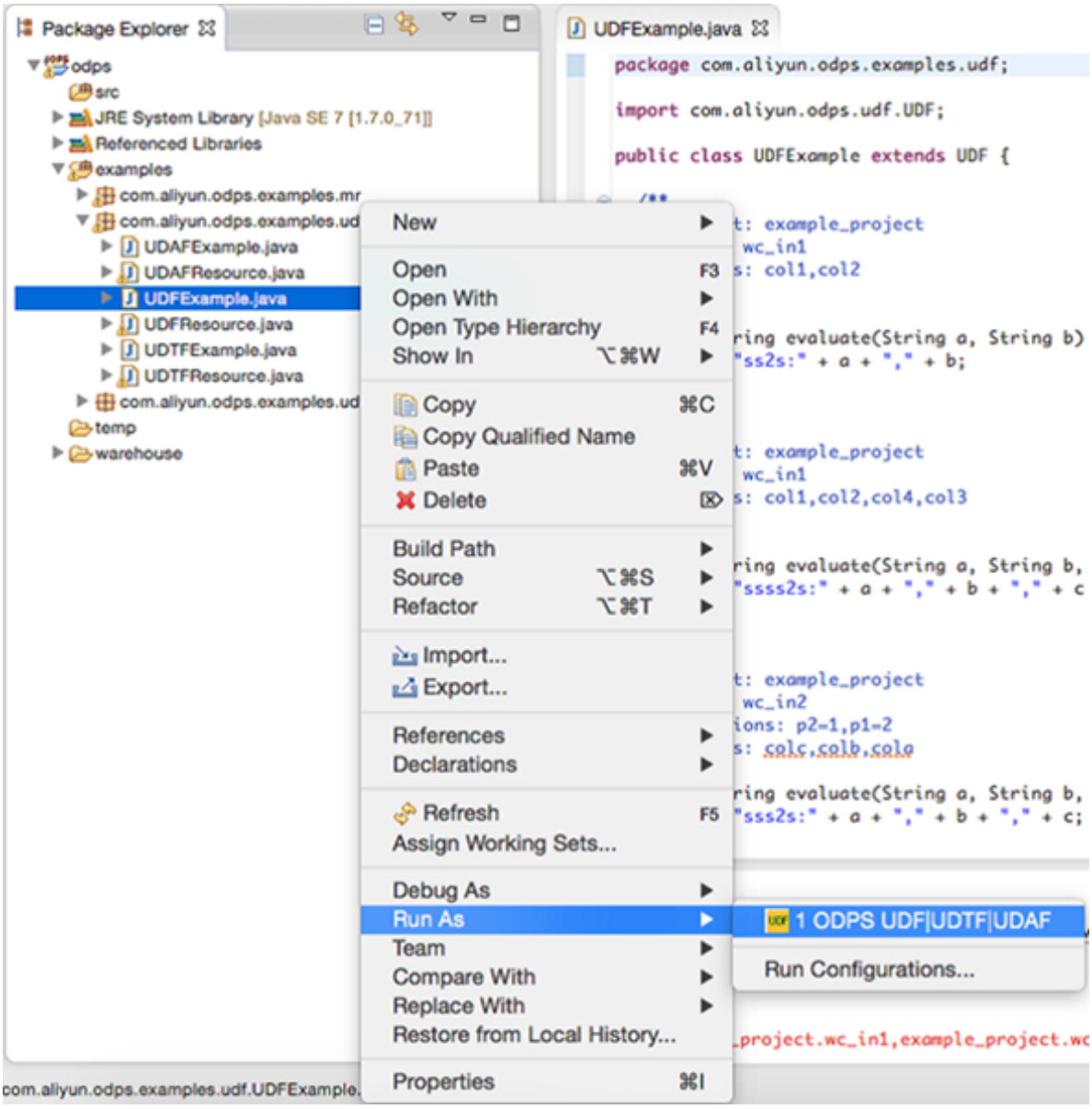


26.14.2.4.2.2 右键单击快速运行

操作步骤

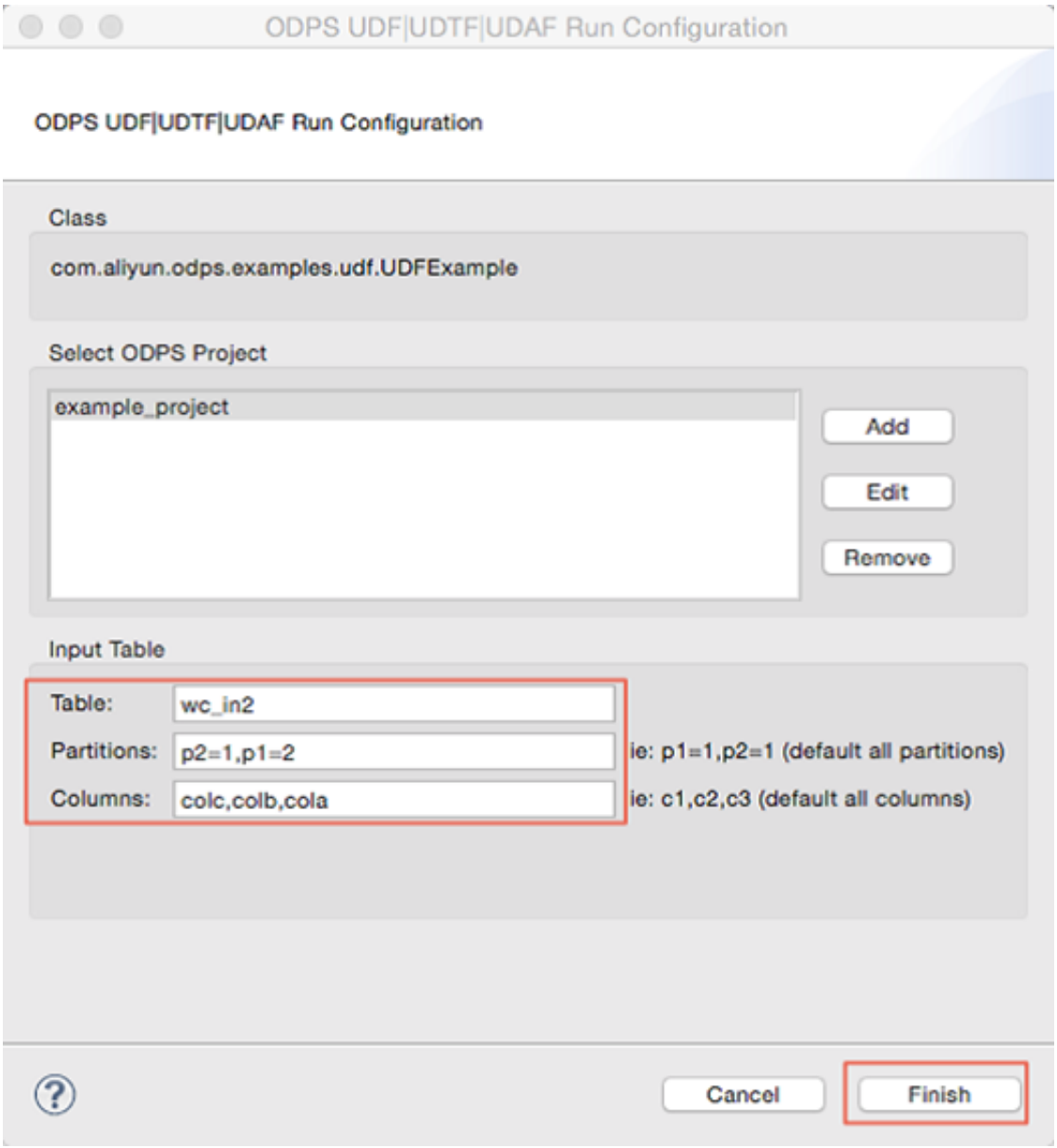
1. 选中一个udf.java文件（例如：UDFExample.java）并单击鼠标右键，选择**Run As > Run UDF|UDAF|UDTF**，如下图所示。

图 807: 右键单击快速运行步骤1



2. 在弹出的对话框中，配置相关参数，配置信息如下图所示。

图 808: 右键单击快速运行步骤2



说明: 上述配置中，Table表示UDF的输入表，Partitions表示读取某个分区下的数据，分区由逗号分隔，Columns表示列，将依次作为UDF 函数的参数被传入，列名由逗号分隔。

3. 单击**Finish**，运行UDF，获得输出结果。

26.14.2.4.3 运行用户自定义UDF程序

操作步骤

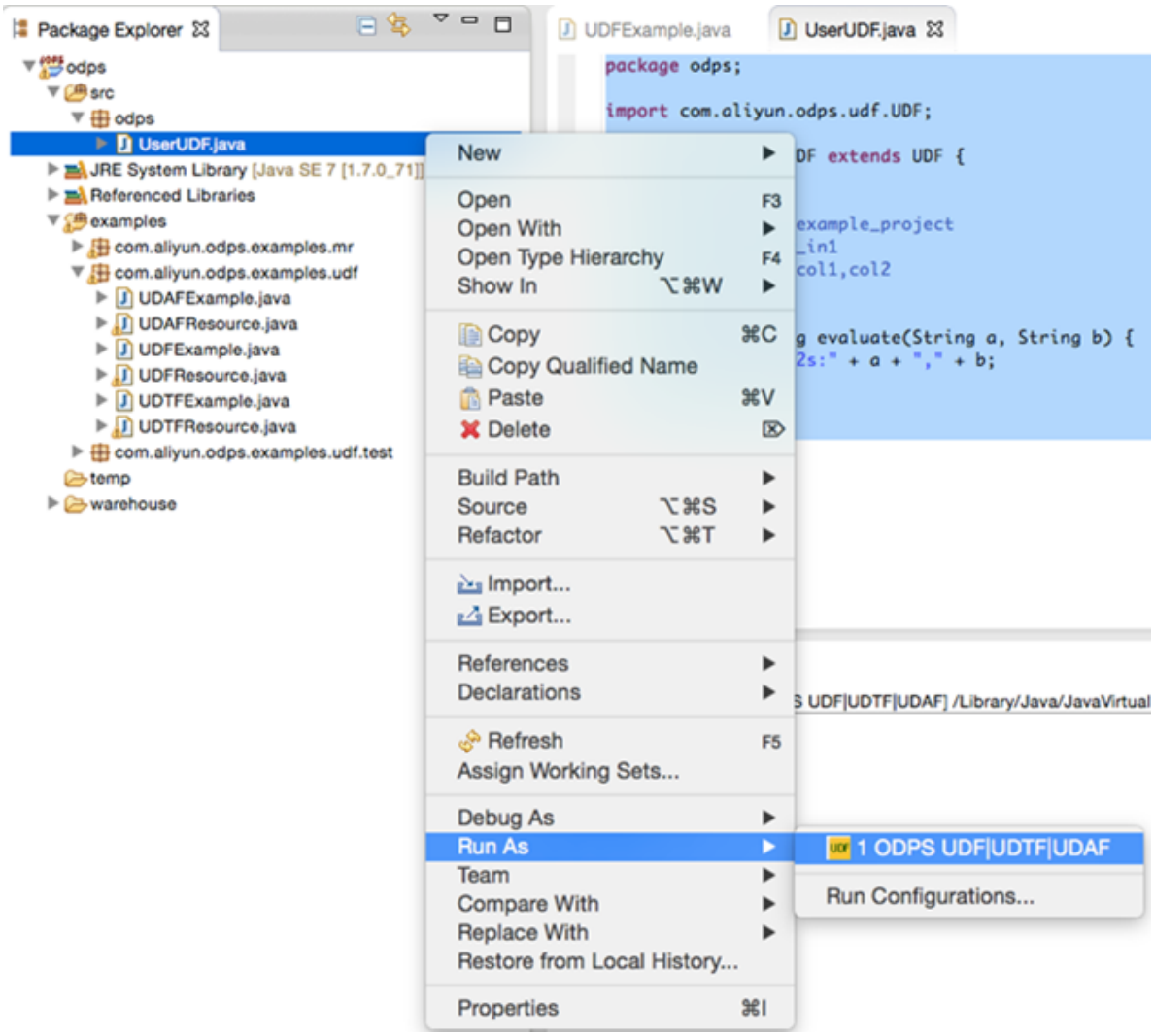
1. 右键单击一个工程并选择**New** > **UDF**（或者在菜单栏选择**File** > **New** > **UDF**）。填写UDF类名后单击**Finish**。在对应的**src**目录下生成与UDF类名同名的Java文件，编辑该java文件内容如下。

```
package odps;
```

```
import com.aliyun.odps.udf.UDF;
public class UserUDF extends UDF {
/**
* project: example_project
* table: wc_in1
* columns: col1,col2
*
*/
public String evaluate(String a, String b) { return "ss2s:" + a + "," + b;
}
}
```

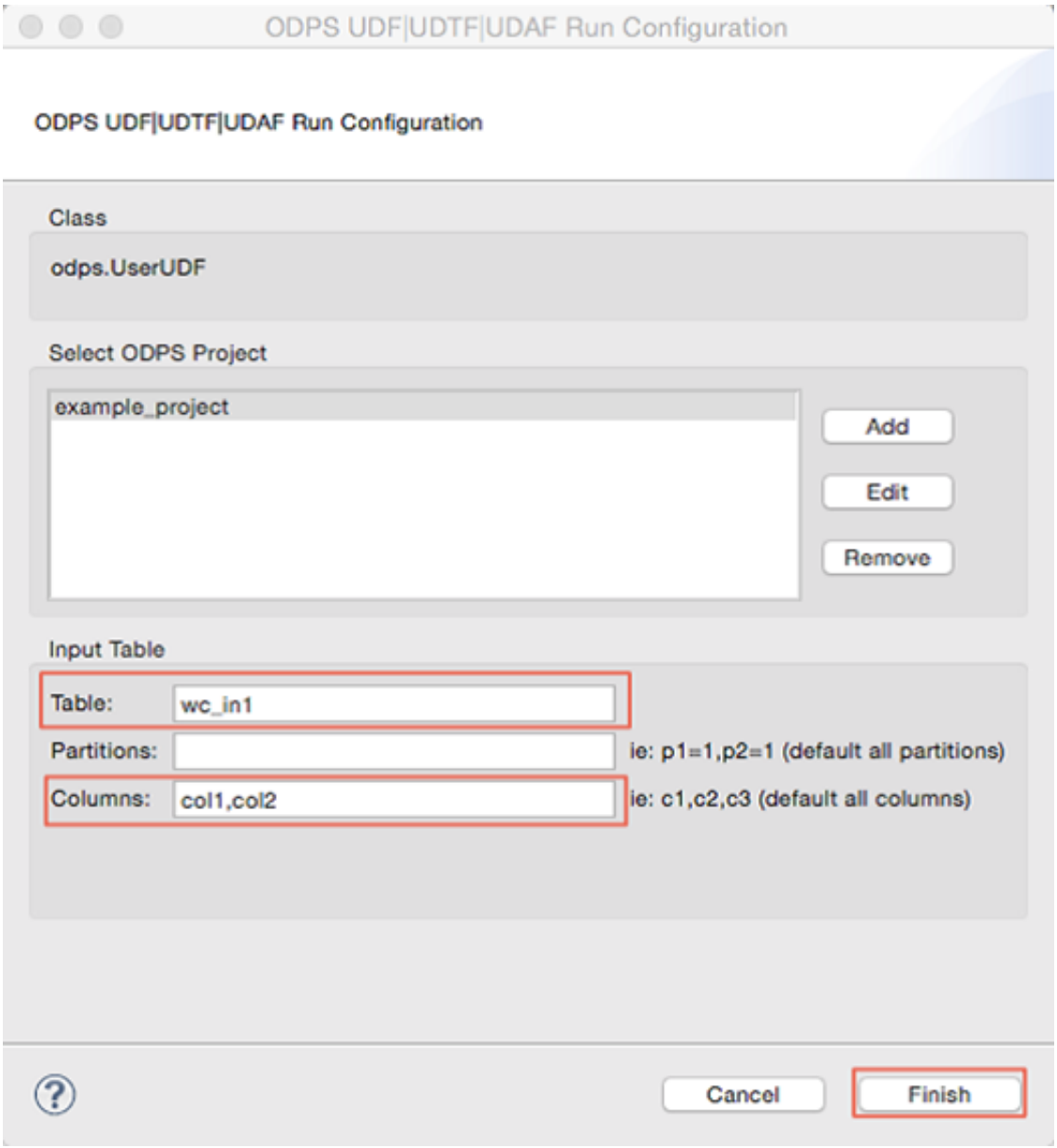
2. 右键选择该java文件（如UserUDF.java），选择**Run As** > **ODPS UDF|UDTF|UDAF**，如下图所示。

图 809: 运行用户自定义UDF程序步骤1



3. 在弹出的对话框中，配置如下图所示的信息。

图 810: 运行用户自定义UDF程序步骤3-1



4. 单击**finish**，获得输出结果。

```
ss2s:A1,A2
ss2s:A1,A2
ss2s:A1,A2
ss2s:A1,A2
```

说明: 本示例中仅给出UDF的运行示例，UDTF的运行方式与UDF基本相同，不做特殊说明。

26.14.2.5 Graph开发插件介绍

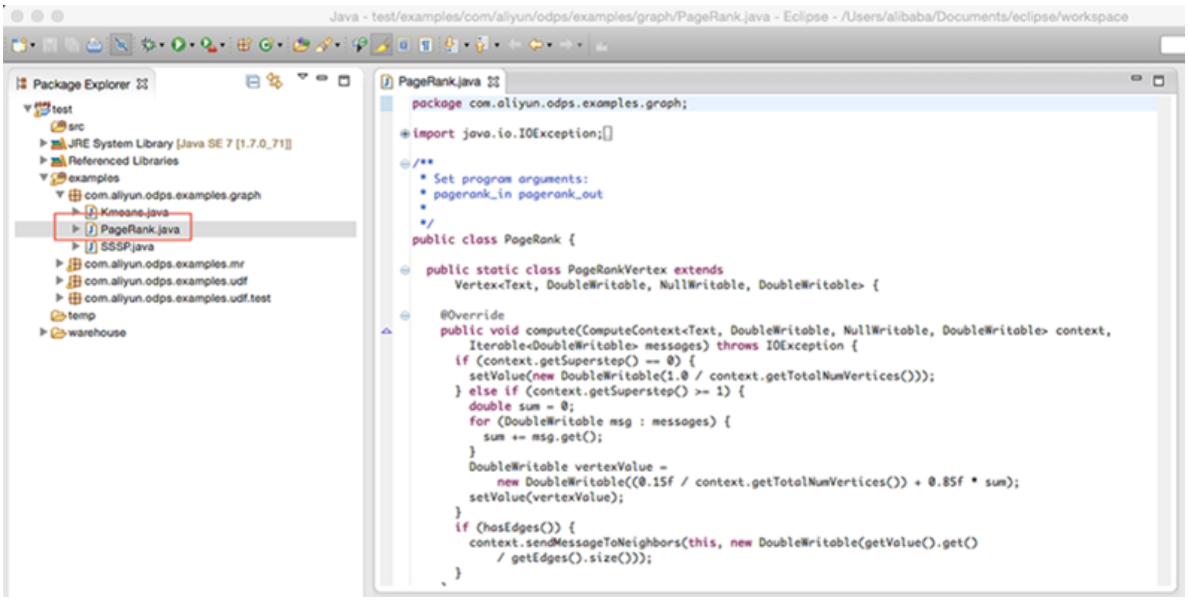
背景信息

创建MaxCompute项目后，用户可以编写自己的Graph程序，并依照如下操作步骤完成本地调试。在此示例中，选用插件提供的PageRank.java来完成本地调试工作。

操作步骤

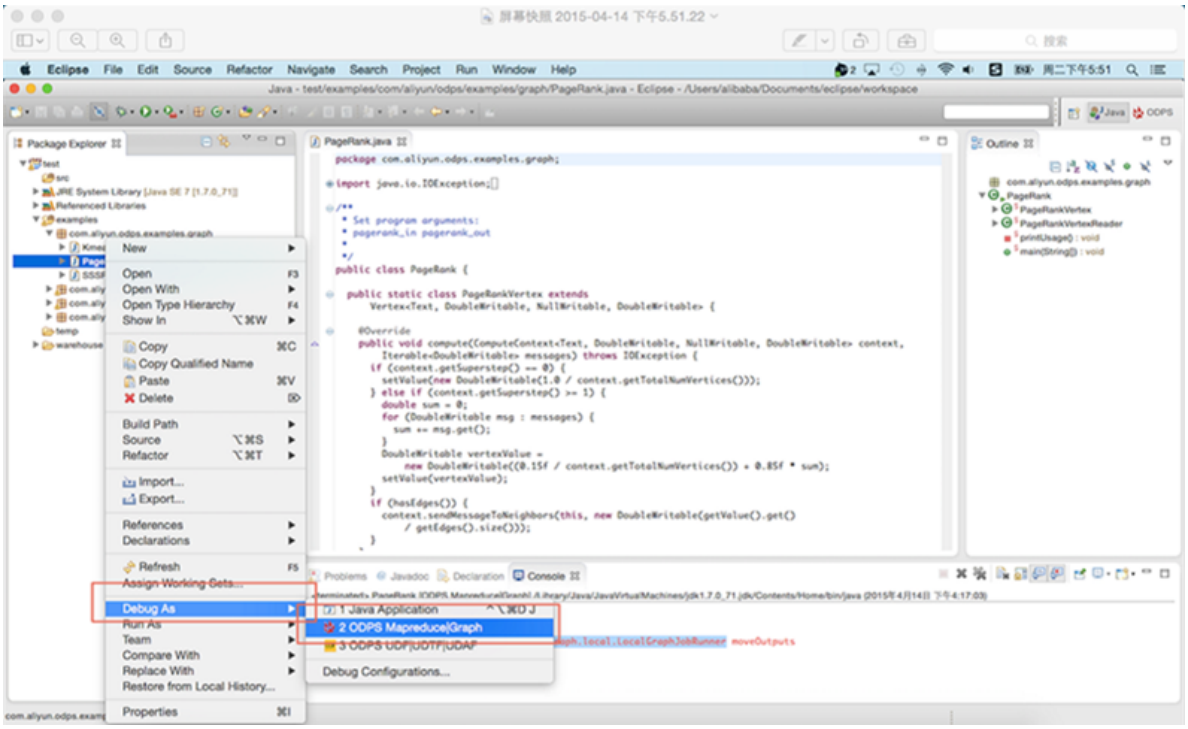
1. 选择**examples** > **PageRank.java**，如下图所示。

图 811: PageRank.java 代码图1



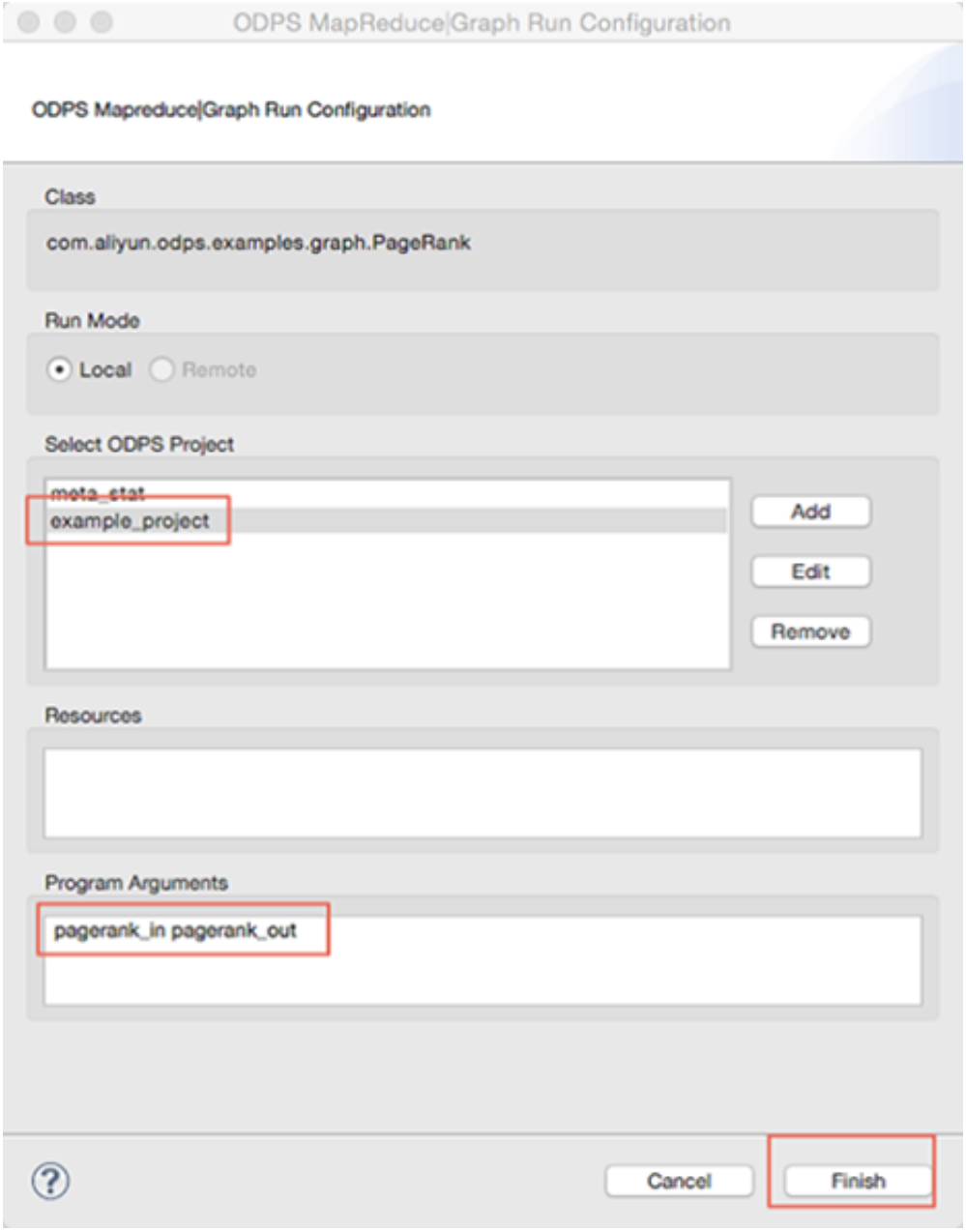
2. 右键单击，选择**Debug As** > **ODPS MapReduce|Graph**，如下图所示。

图 812: PageRank.java 代码图2



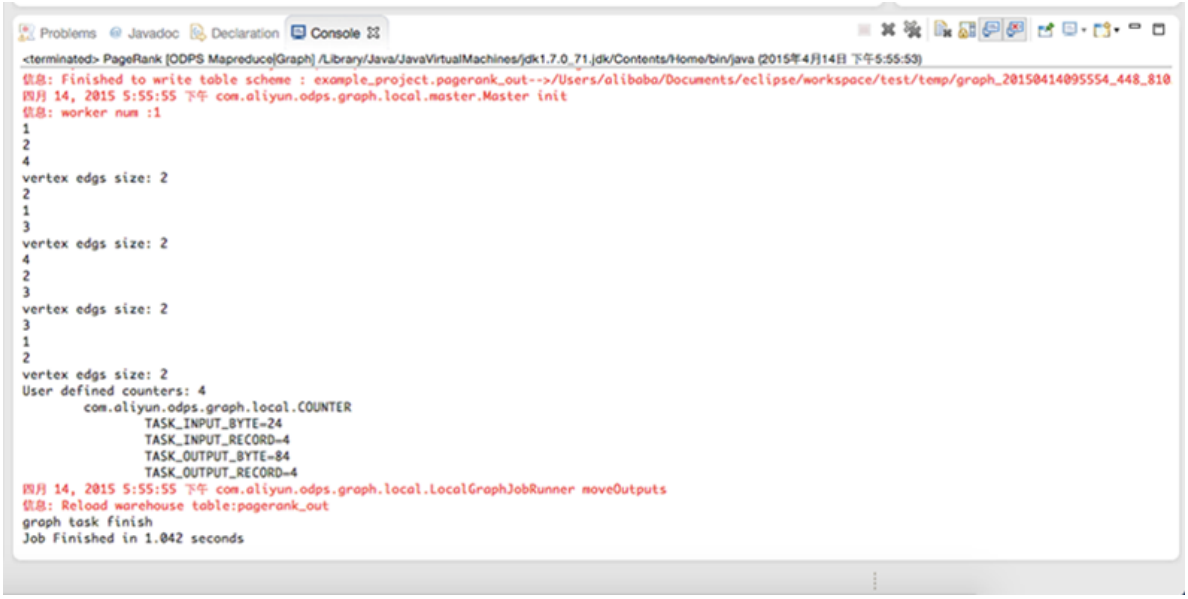
3. 在弹出的对话框中，按照如下图所示的信息进行配置。

图 813: 配置图



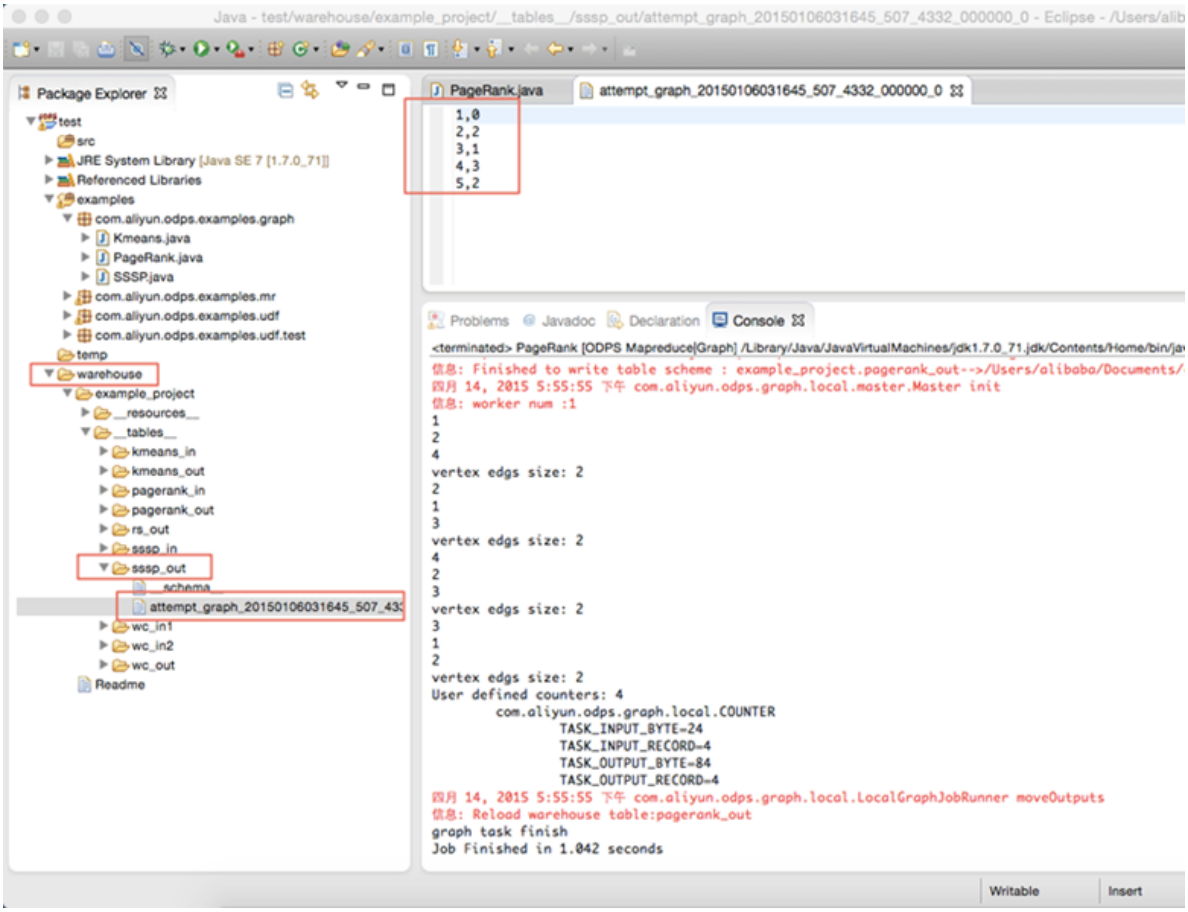
4. 单击**Finish**，查看作业运行结果，如下图所示。

图 814: 运行结果



可以查看在本地的计算结果，如下图所示。

图 815: 本地计算结果



调试通过后，用户可以将程序打包，并以jar资源的形式上传到MaxCompute，并提交Graph 作业。

说明:

- 打包过程可参考：*MapReduce*开发插件介绍。
- 有关本地结果的目录结构介绍，也可以参考：*MapReduce*开发插件介绍。
- 关于上传jar资源的详细介绍，可参考：编写并运行*Graph*中的相关部分。

27 大数据开发套件

27.1 产品概述

产品架构

阿里云大数据开发平台是阿里巴巴集团推出的大数据领域平台级产品，提供一站式大数据开发、管理、分析、挖掘、共享、交换等端到端的解决方案，其利用MaxCompute（原ODPS）在几分钟内可将原始数据转变为业务洞察的海量数据处理能力，而无需关心集群的搭建和运维。

产品架构如图 *816:* 阿里云大数据开发平台架构图所示：

图 816: 阿里云大数据开发平台架构图



由上图可以看出，阿里云大数据开发平台底层是基于MaxCompute（原 ODPS）的集成开发环境，包括数据开发、数据管理、数据分析、数据挖掘和管理控制台。其中数据分析和数据挖掘属于阿里云大数据开发平台高级组件，需要根据您所在项目购买情况而定是否具有这两大高级组件。

主要特性

阿里云大数据开发平台引入全新的工作流任务设计理念，较之前版本具有如下几大特性：

- 拖拽式的操作界面

系统数据开发模块提供丰富的可视化组件，包括SQL（MaxCompute SQL）、数据同步、MR（MaxCompute MR）、机器学习、SHELL、虚拟节点。您可通过向开发画布拖拽组件的方式来完成新建工作流任务，组件配置即开发。

- 个性化数据收藏与管理

  系统数据管理模块提供个性化的数据收藏与管理功能，您可轻松收藏所关注的数据表，同时可对数据表的生命周期、基本信息、负责人等信息进行管理，也可查看数据表存储信息、分区信息、产出信息、血缘信息等内容。

- 一键式跨项目任务发布

  数据开发模块也提供一键式将开发环境项目空间中的任务发布至测试环境、预发环境或生产环境。

- 可视化任务监控

  运维中心提供可视化的任务监控管理工具，支持以DAG图的形式展示任务运行时的全局情况。异常管理便捷化，支持重跑、恢复、暂停和终止等操作。

开发流程

通常情况下，数据开发需经由如下开发流程来完成：

图 817: 总体开发流程



从上图可以看出，数据开发的总体流程可包括数据产生、数据收集与存储、分析与计算、数据提取、数据展现与分享。其中虚线框内的开发流程都可基于阿里云大数据平台来完成。说明如下：

- 数据产生：业务系统每天会产生大量结构化的数据，这些数据都存储在业务系统所对应的数据库中，包括MySQL、Oracle、RDS等。
- 数据收集与存储：若想利用MaxCompute（原ODPS）的海量数据存储与处理能力来分析这些已有的数据，首先需要将不同业务系统的数据进行同步至MaxCompute（原 ODPS）中。阿里大数据开发平台提供数据同步服务，可支持多种数据源类型，将业务系统数据按照预设的调度周期同步到MaxCompute（原ODPS）。
- 数据分析与处理：随之可对MaxCompute（原ODPS）上的数据进行加工（MaxCompute SQL、MaxCompute MR）、分析与挖掘（数据分析、数据挖掘）等处理，从而发现其价值。
- 数据提取：分析与处理后的结果数据，需同步导出至业务系统，以供业务人员使用其分析的价值。
- 数据展现和分享：可通过报表、地理信息系统等多种展现方式来展示与分享大数据分析、处理后的成果。

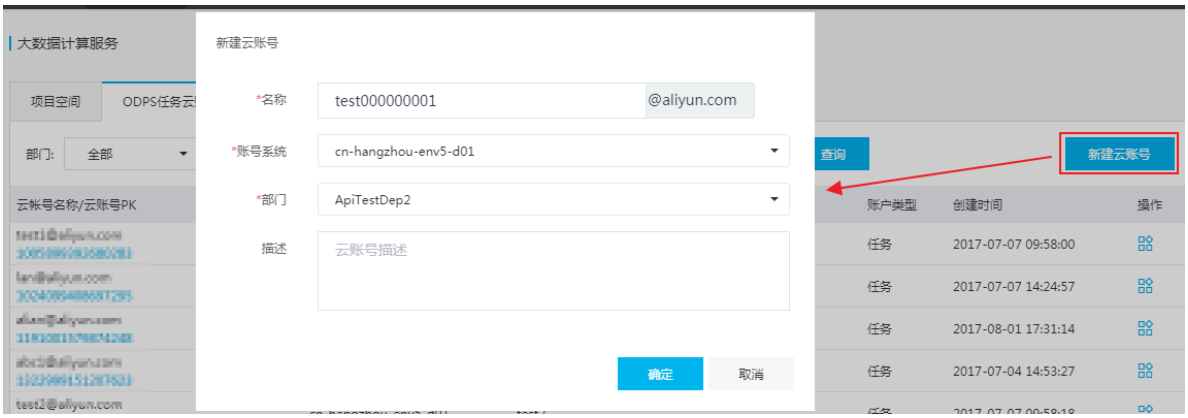
27.2 大数据开发平台入门

27.2.1 准备工作

操作步骤

1. 登录*DTCenter*云控制台。
2. 进入**系统管理** > **部门管理**页面，单击**添加部门**，详情请参见创建部门。
3. 进入**系统管理** > **用户管理**页面，单击**添加**，创建用户。
4. 进入**系统管理** > **项目管理**页面，单击右上角的**新增项目**，创建*DTCenter*项目。
5. 单击顶部菜单栏云资源中心，导航至**云产品管理** > **大数据计算服务** > **ODPS任务云账号**，单击**新建云账号**。

   图 818: 新建云账号



6. 导航至**项目空间**页面，单击右上角的**新建ODPS项目**，填写创建大数据计算服务页面信息。

图 819: 新建ODPS项目



7. 单击顶部菜单栏云资源中心，导航至**云产品管理** > **大数据开发套件**，选择部门后，进入阿里云大数据开发平台，进行相关操作。

> **注意:** 若账号只被组织管理员添加至组织，可以访问阿里云大数据开发平台，但不能够正常使用阿里云大数据开发平台的相关服务。此时，还需联系**项目管理员**将账号信息添加至项目空间并赋予开发角色。

27.2.2 进入阿里云大数据开发平台

成功登录至阿里云大数据平台，首页如下：

图 820: 进入阿里云大数据平台



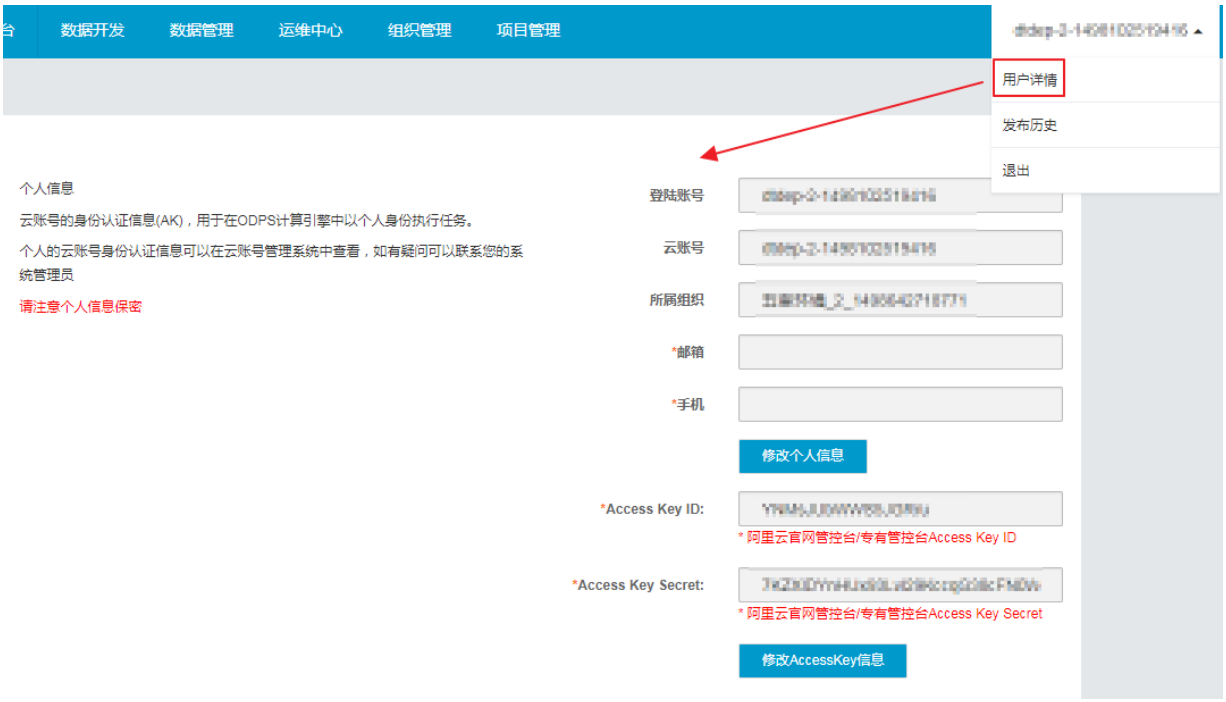
针对上图中的标注说明如下：

1-项目空间列表：当前用户加入的项目空间列表，即只对列表中的项目空间有操作权限。用户可在此处切换项目空间。

2-顶部功能菜单栏：当前用户有权可见的功能模块，包括数据开发、数据管理、运维中心、组织管理、项目管理。

3-用户信息：当前登录用户，可查看并编辑用户信息，包括邮箱、手机以及AK，如下图所示：

图 821: 用户详情页



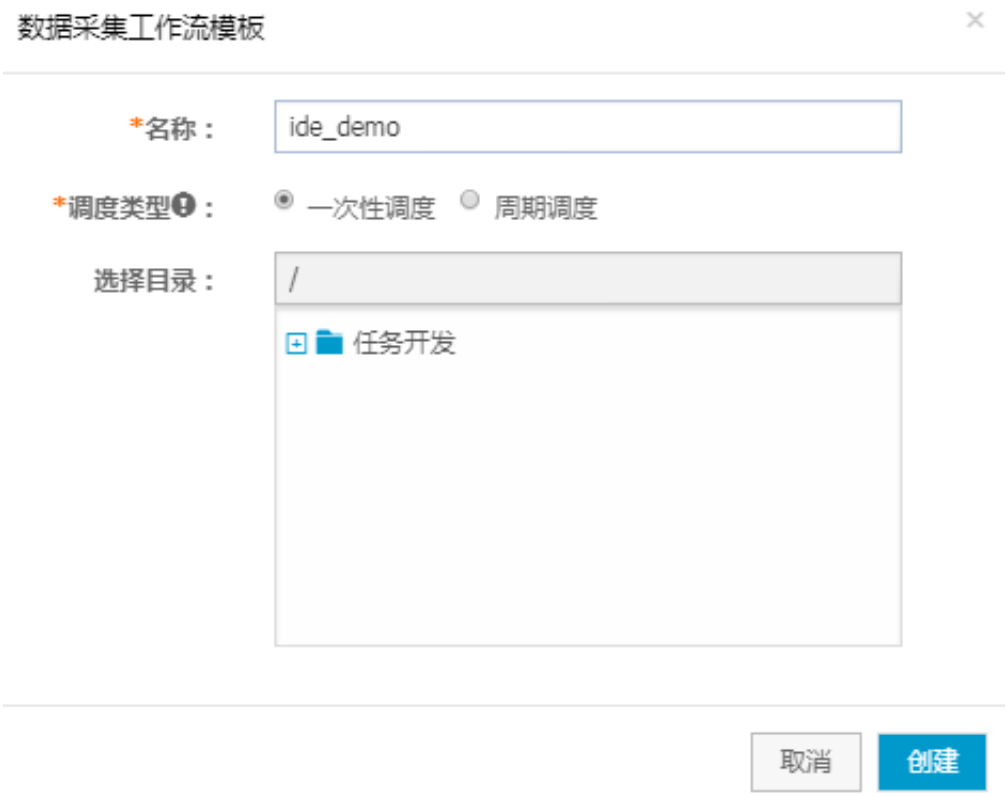
4-工具栏：可通过“新建”按钮进行新建工作流任务、脚本文件、上传资源、新建函数；可通过**更多操作**进行快速关闭其他和全部标签。

5-导航栏：对应顶部功能菜单的导航栏，不同的功能模块对应不同的左侧导航栏。

6-数据开发首页：数据开发模块首页提供了今日任务监控（运维中心）、项目成员数、项目管理员数、数据源连接数以及新建采集模型工作流任务的快捷入口。

- 今日任务监控：主要展示调度系统中今日等待时间/等待资源的任务数、未运行/运行中的任务数、失败/已完成的任务数；
- 当前项目空间中项目成员数、项目管理员数和已经配置的数据源数。
- 快捷入口/业务模板：目前提供一种新建数据采集模型的工作流任务的快捷入口，单击可快捷创建数据采集任务。如图 *822:* 新建采集模型工作流任务所示：

图 822: 新建采集模型工作流任务



通过快捷模板新建的工作流任务，默认包含一个数据同步节点和一个MaxCompute SQL节点，且MaxCompute SQL节点依赖于数据同步节点。

7-文件目录树：工作流任务/节点任务、脚本文件、函数、资源文件的目录列表。

27.2.3 功能索引

下表为主要面向阿里云大数据平台**开发者**在日常数据开发过程中涉及到的功能列表。如下所示：

功能模块	**功能点索引**
数据开发	开发面板功能总介绍，系统调度参数#数据同步任务配置，工作流任务新增、编辑、复制、删除，脚本文件新增、删除，资源新增、删除，函数注册、注销，导入本地数据。
数据管理	表搜索，查看表详情，数据权限申请、审批与回收，数据表新增、收藏、修改、删除。
运维中心	工作流任务测试，节点测试。
组织管理	注：需组织管理员权限。
项目管理	注：需项目管理员权限，开发者可查看**项目名称**和**数据源配置**情况。

27.3 快速开始

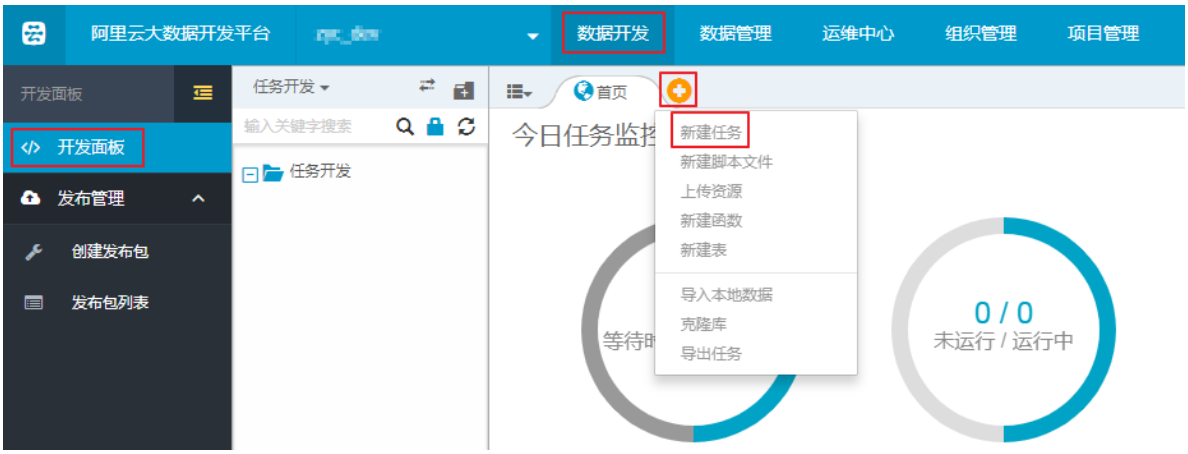
27.3.1 新建任务

场景一：新建周期性调度工作流任务

开发者新建工作流任务的具体步骤如下：

1. 以开发角色身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据开发**，导航至**开发面板** > **新建** > **新建任务**，如图 *823: 新建任务*所示：

图 823: 新建任务



3. 在**新建任务**弹出框中填写各配置项，选择任务类型为**工作流任务**，调度类型为**周期调度**，如图 *824: 新建周期调度工作流任务弹出框*所示：

图 824: 新建周期调度工作流任务弹出框



新建任务弹出框配置项具体说明：

- 任务名称：支持数字、字母、下划线组合。
- 描述：针对当前工作流任务的简单描述，支持中文、字母、数字、下划线组合。
- 任务类型：支持工作流任务和单节点任务（单节点任务包括SQL、MR、机器学习、Shell、数据同步等类型）。
- 调度类型：调度类型支持一次性调度和周期调度两种类型，且成功创建工作流任务之后无法修改。一次性调度的工作流任务属性和节点属性中不包含调度属性，同时在工作流任务开发面板中可直接运行当前工作流任务。

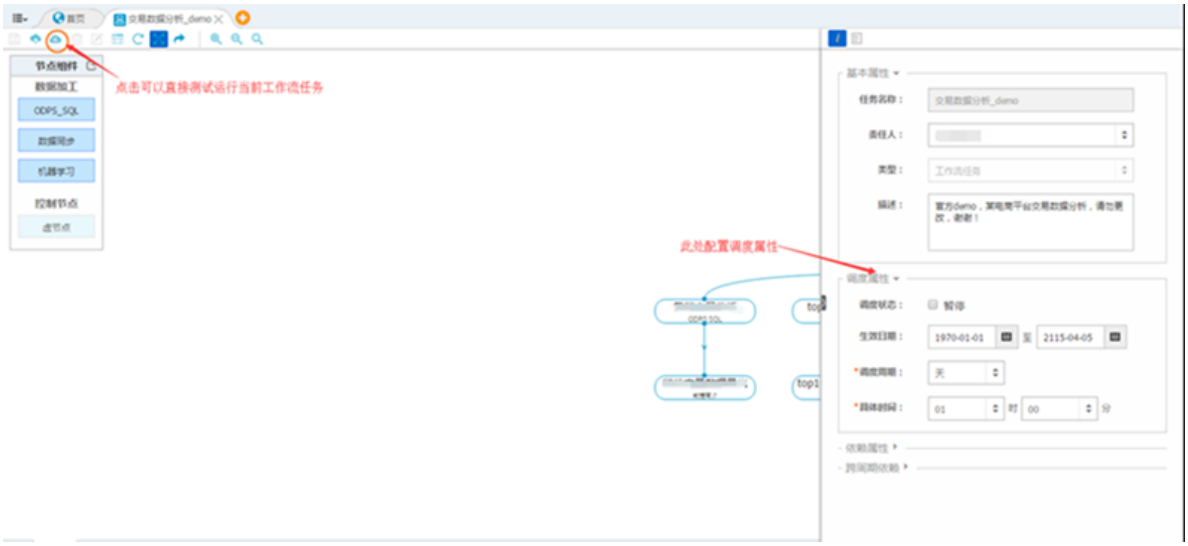
- 选择目录：选择当前任务所属的目录层级。右键单击任务开发，选择新建文件夹即可创建目录，如新建目录所示：

图 825: 新建目录



说明: 目录创建后，将同步显示在新建任务弹出框的选择目录中，您可根据自身需求选择相应的目录。

图 826: 周期性调度工作流任务开发面板



一次性调度的工作流任务适用于一次性的数据导入与加工等操作而无需周期性地执行，本示例主要阐述新建周期性调度工作流任务。

4. （**单节点任务没有此步骤**）：在**节点组件区域**中选中节点拖拽至开发面板画布。
5. （**单节点任务没有此步骤**）：在**新建节点**弹出框中填写各配置项。

图 827: 新建节点弹出框

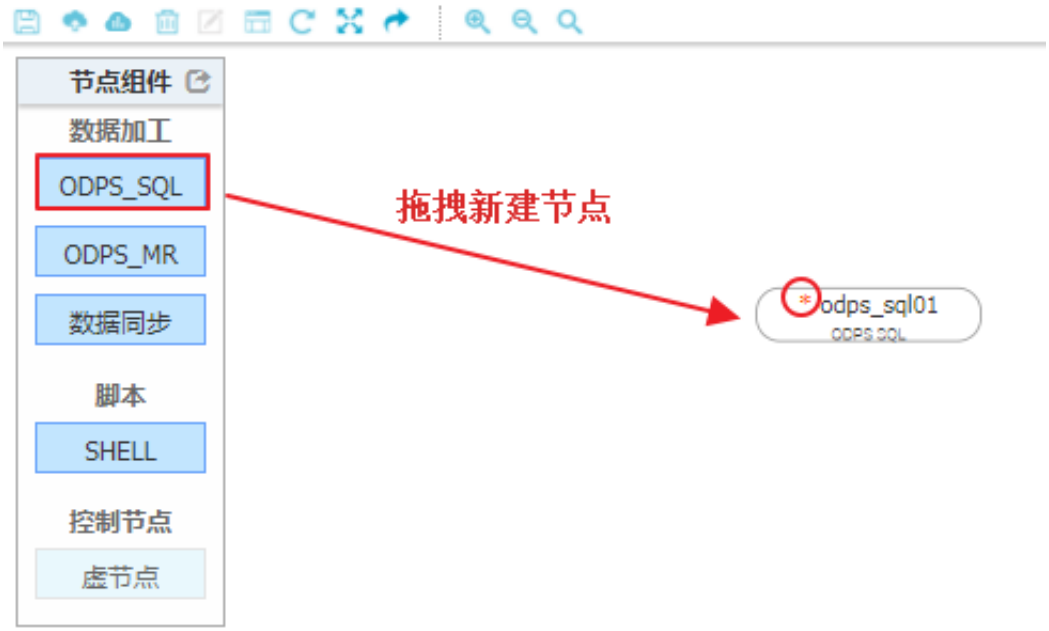


新建节点弹出框配置项具体说明：

- 节点名称：支持数字、字母、下划线组合。
- 节点类型：拖拽时选择的节点类型，不支持修改。
- 描述：针对当前节点的简单描述。

6. （**单节点任务没有此步骤**）：单击**创建**。
7. （**单节点任务没有此步骤**）：向画布中再拖拽两个ODPS SQL节点，如图 *828:* 工作流任务开发面板所示：

图 828: 工作流任务开发面板



在画布中的节点，上半部分显示的是新建节点时填写的**节点名称**，下半部分显示的是该节点的类型（包括ODPS_SQL、ODPS_MR、数据同步、SHELL和虚节点）。节点左上方显示的小红星，则表示该节点配置信息尚未保存。

8. 鼠标放置在节点底部突出的小圆点，单击拖拽至下游节点上进行连线，完成依赖关系的配置。

图 829: 配置依赖关系



9. 单击  ，或按快捷键Ctrl+S进行保存。

10. 勾选需要提交的节点，单击  进行提交。

图 830: 节点列表



11.单击弹出框中的**确定提交**。

说明:

保存和提交的区别：

- 保存是将当前工作流任务配置（包括工作流任务属性、节点代码、节点属性）保存在当前数据开发环境中，而没有到调度系统。
- 提交是将当前工作流任务配置提交至当前项目的调度系统中，以方便测试和发布。只有提交过的工作流任务才能调度执行及发布到其他项目空间。同时，提交成功的工作流任务将被锁定限制编辑。

到此，我们已成功新建了一个工作流任务，但是工作流任务属性、节点代码（双击节点进入，包括数据同步和*MaxCompute#原ODPS#*代码开发）、节点属性都尚未配置。具体可根据自身业务场景来进行配置。

说明:

工作流任务定义需遵循如下规则：

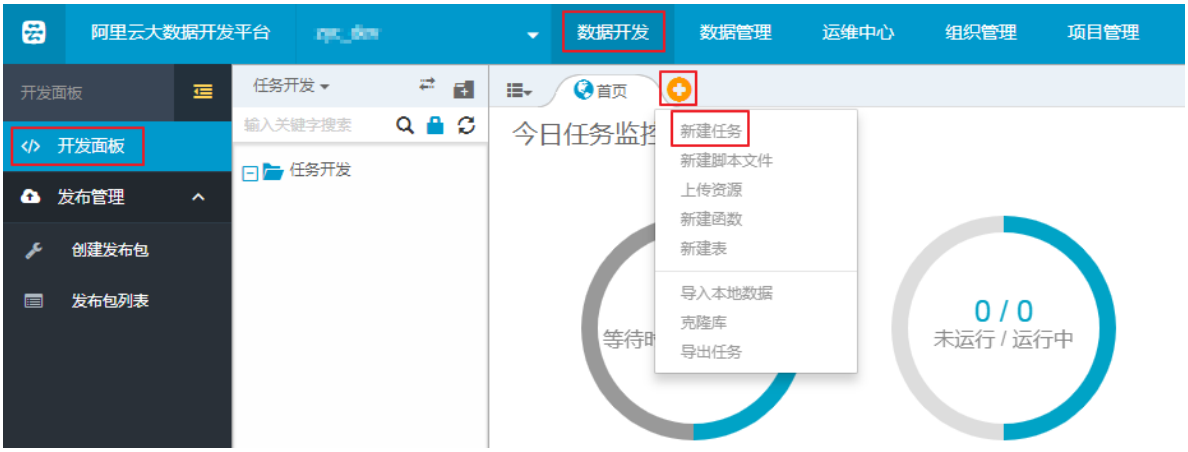
- 一个工作流任务只能有一个DAG图且只能有一个起始根节点。
- 工作流任务名称在同一个项目空间中必须唯一。
- 工作流任务作为一个整体对象被其他工作流任务依赖。
- 不同项目空间的工作流任务之间可以相互依赖。
- 工作流任务可以没有任何上游单独按照本身属性调度。

新建一次性调度工作流任务

开发者新建工作流任务的具体步骤如下：

1. 以开发角色身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据开发**，导航至**开发面板** > **新建** > **新建任务**，如图 *831: 新建一次性任务*所示：

图 831: 新建一次性任务



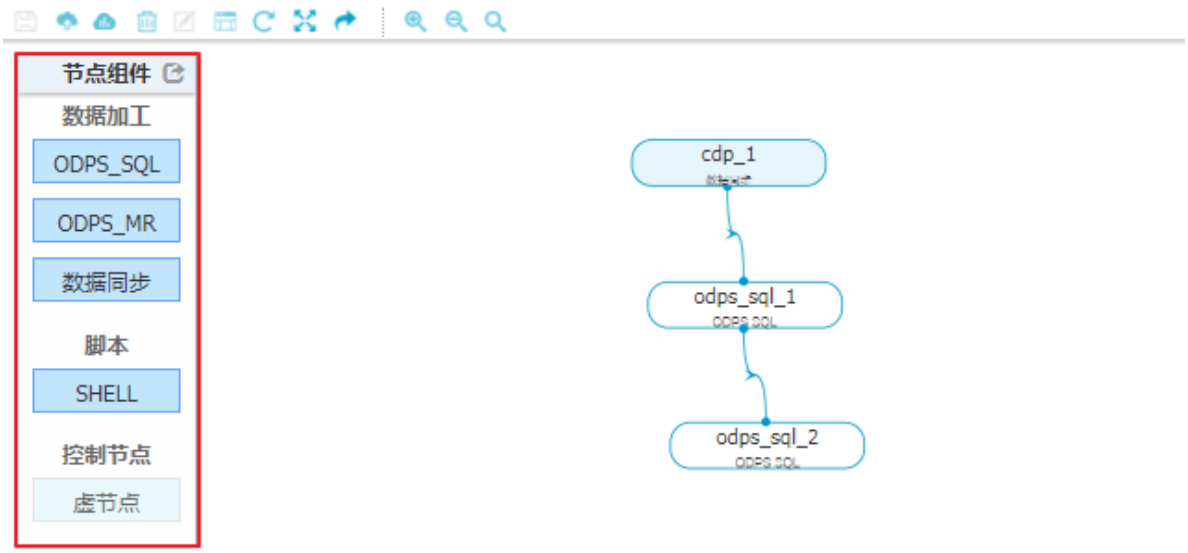
3. 在**新建任务**弹出框中填写各配置项，选择任务类型为**工作流任务**，调度类型为**一次性调度**，如图 *832: 新建一次性调度工作流任务弹出框*所示：

图 832: 新建一次性调度工作流任务弹出框



4. 在**节点组件**区域中选中节点拖拽至开发面板画布。

图 833: 工作流任务开发面板



5. 单击测试运行按钮 进行测试。

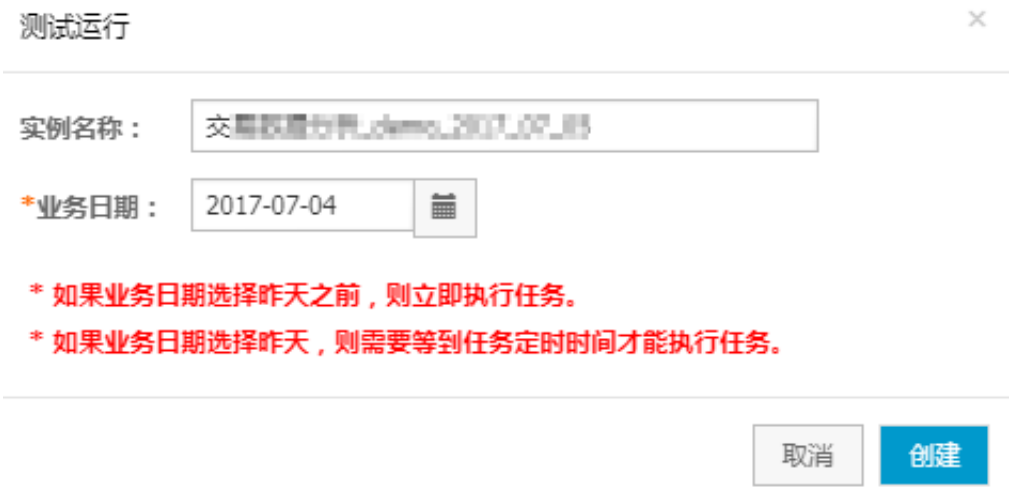
图 834: 变更节点列表



若没有提交当前工作流任务，则点击运行按钮会弹出变更节点列表对话框，单击**确定提交**即可。

6. 任务提交成功后，在**测试运行**对话框中，单击**创建**进入确认页面。

图 835: 测试运行对话框



7. 单击弹出框中的**前往运维中心**，即可查看任务执行情况。

图 836: 工作流任务测试运行



页面将跳转至**运维中心** > **任务管理** > **任务运维视图** > **测试**tab页，查看工作流任务执行情况，如图 *837:* 工作流任务执行页面所示：

图 837: 工作流任务执行页面



27.3.2 测试任务

常见场景—任务测试

开发者在完成工作流任务的开发任务之后，即可进入运维中心进行工作流任务的测试。

具体步骤如下：

1. 选择所需的项目空间，进入某具体工作流任务开发面板。
2. 单击  进行提交。

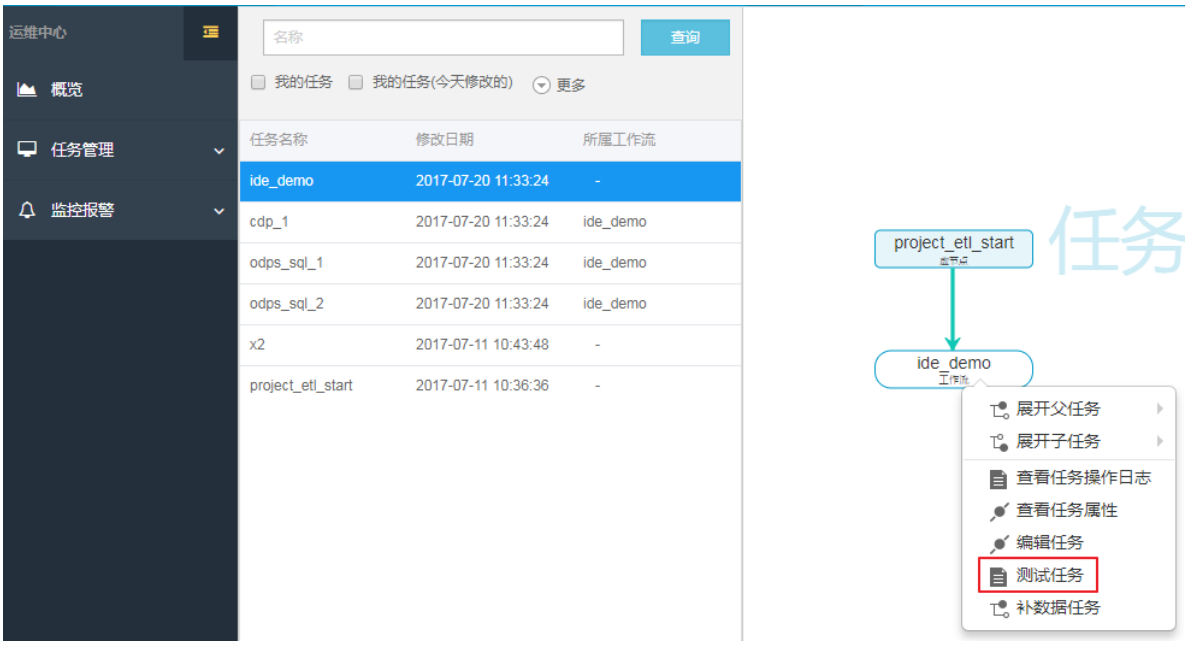
3. 在变更节点列表对话框中，单击**确定提交**。

图 838: 变更节点列表对话框



4. 导航至**运维中心** > **任务管理**，单击**任务管理视图**。
5. 选择左侧列表中已提交的工作流任务，并在右侧 DAG 图上右键选择**测试工作流任务**。

图 839: 测试工作流任务



6. 在测试运行对话框中选择**业务日期**，单击**生成并运行**。

图 840: 测试运行对话框



7. 跳转至**任务运维视图** > **测试**页面，查看其工作流任务测试运行的状态。

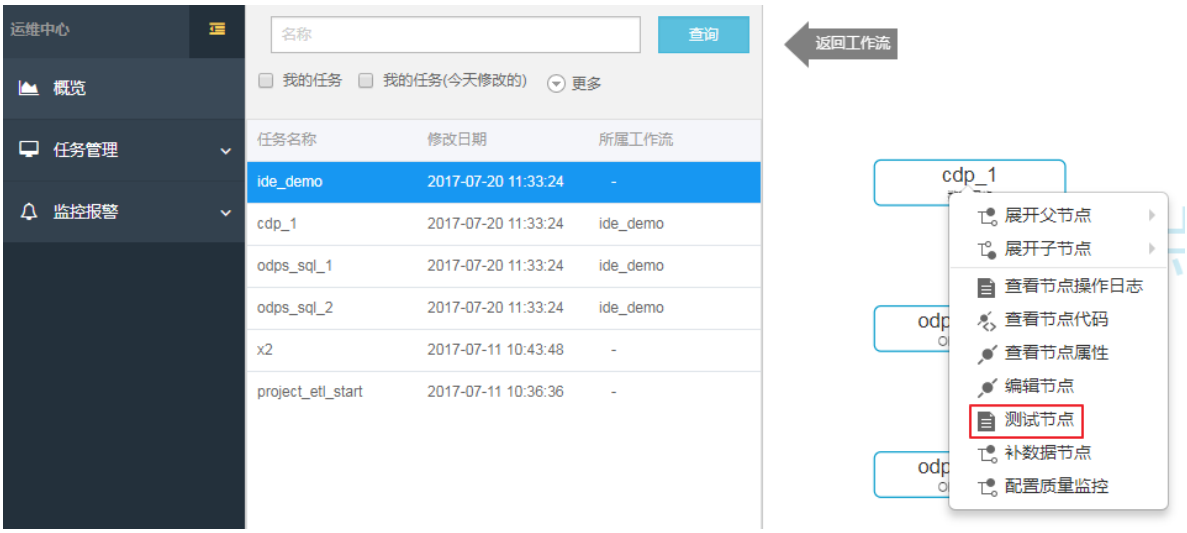
常见场景—节点测试

在阿里云大数据开发平台中即可对工作流任务进行测试，也可针对工作流任务中的某个节点进行测试。

具体步骤如下：

1. 导航至**运维中心** > **任务管理**，单击**任务管理视图**。
2. 选择左侧列表中已提交的工作流任务，在右侧DAG图中双击工作流任务进入工作流任务内，选择节点右键测试节点。

图 841: 测试节点



3. 在测试运行弹出框中选择业务日期，单击**生成并运行**。

图 842: 节点测试弹出框



4. 跳转至**任务运维视图** > **测试**页面，查看节点测试运行状态。

27.3.3 编辑任务

开发者若想编辑已提交的任务，需根据以下步骤进行操作：

操作步骤

1. 以开发角色身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据开发**，导航至**开发面板** > **文件目录树**区域。
3. 在任务开发列表搜索框中，输入任务名称来模糊匹配查找需编辑的任务。

图 843: 模糊查询



27.3.4 复制任务

大数据开发套件提供工作流任务克隆的功能，开发者可通过复制工作流任务的方式来完成克隆。

操作步骤

1. 以开发角色身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据开发**，导航至**开发面板** > **文件目录树**区域；
3. 在工作流任务列表搜索框中输入任务名称来模糊匹配查找需复制的任务。

图 844: 任务目录树



4. 右击选择**克隆**。
5. 在**复制任务**弹出框中输入新工作流任务名称，并修改任务的调度属性和所属目录。

图 845: 复制工作流任务对话框



6. 单击**执行复制**。

说明: 复制工作流任务只复制当前工作流任务的工作流任务属性、节点及节点依赖关系、节点属性、节点代码，但是工作流任务所引用的资源文件以及上下游工作流任务关系将不被复制。

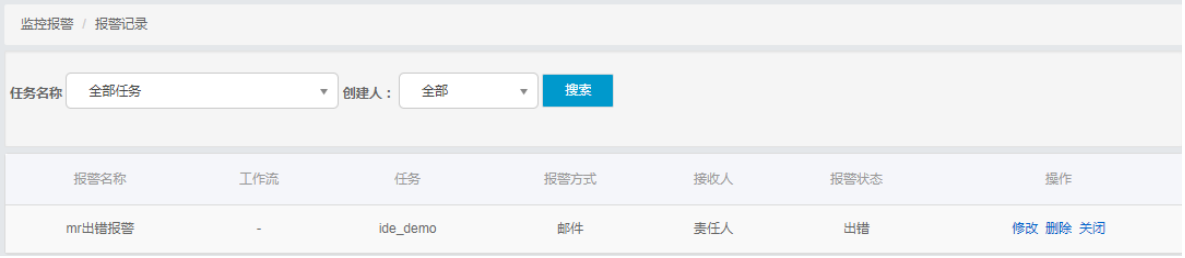
27.3.5 任务监控告警配置

常见场景—修改监控报警配置项

说明: 任务的监控报警设置，请参见添加告警。

1. 进入**运维中心** > **监控报警** > **自定义提醒**界面，之前在节点定义页面配置好的报警信息都可以在此处进行修改；

图 846: 自定义报警列表



2. 单击对应的报警节点对象，在右边的操作栏中单击**修改**，即可完成修改：

图 847: 修改报警配置



报警对象也可以进行删除（即将报警配置删除）、关闭（不删除，相当于暂时关闭，可重新开启）。

常见场景—查看已发出的报警记录

通过此页面可以查看历史的报警详细信息，以便于定位和分析问题。

1. 进入**运维中心** > **监控报警** > **报警记录**页面；

图 848: 报警记录



2. 通过任务名称、责任人、时间段来进行搜索查找；
3. 单击**查看**，可查看具体的报警信息。

27.3.6 删除任务

背景信息

开发者也可删除已创建的任务。

> **注意：** 删除工作流任务操作及下线工作流任务，其将删除该工作流任务的所有配置信息，包括工作流任务属性、节点及节点依赖关系、节点代码、节点属性，并不可恢复。若存在上下游依赖，则只删除该工作流任务及与上下游工作流任务之间的依赖关系，须谨慎操作。

操作步骤

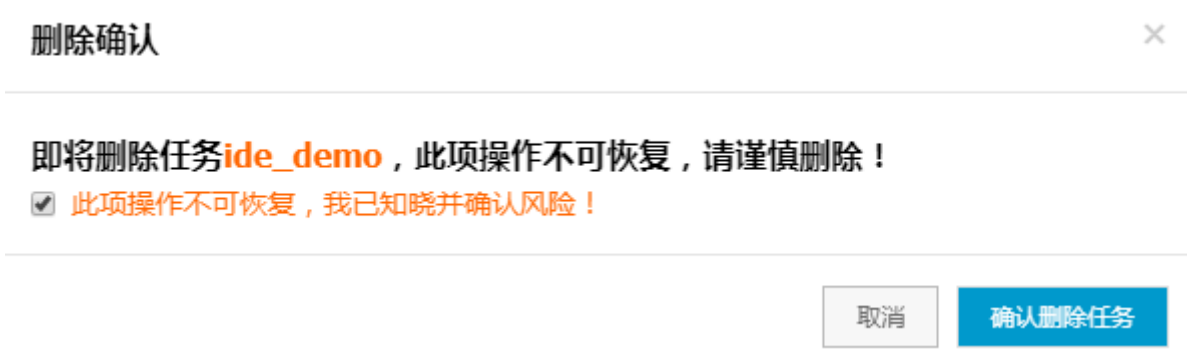
1. 以开发角色身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据开发**，导航至**数据开发** > **文件目录树**区域；
3. 在工作流任务列表搜索框中输入工作流任务名称来模糊匹配查找需删除的工作流任务。

图 849: 任务目录树



4. 右击选择**删除**。
5. 在**删除确认**对话框中，勾选**此项操作不可恢复**，**我已知晓并确认风险**！。

图 850: 删除确认对话框



6. 单击**确认删除任务**。

27.3.7 新建脚本文件

数据开发提供新建MaxCompute SQL、SHELL脚本文件的功能，开发者可在脚本文件中完成临时查询、建表等操作。

操作步骤

1. 以开发角色身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据开发**，导航至**开发面板** > **新建**，单击**新建脚本文件**。
3. 在**新建脚本文件**弹出框中，填写各配置项。

图 851: 新建脚本文件弹出框



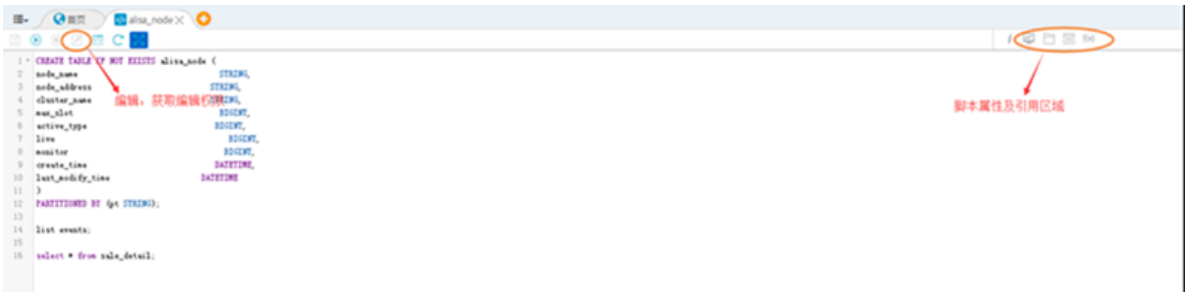
新建脚本文件对话框中配置项具体说明如下：

- 文件名称：支持数字、字母、下划线组合；
- 类型：支持MaxCompute SQL和SHELL两种类型，可选择；
- 描述：针对当前工作流任务的简单描述，支持中文、字母、数字、下划线组合。

4. 单击**提交**。

通过上述步骤可完成新建脚本文件的操作，默认进入MaxCompute代码编辑器，此界面与*MaxCompute代码编辑器*章节（双击MaxCompute SQL/SHELL节点进入）相同，但有细微区别，具体说明如下：

图 852: 脚本文件-代码编辑器界面



上图中标注出与双击节点进入的代码编辑器的差异，说明如下：

- 编辑 

  打开责任人不是当前用户的脚本文件，编辑按钮会高亮显示。即，针对项目其他成员的脚本文件默认只读，单击**编辑**来获取修改权限。

- 任务属性 

  当前脚本文件的基本属性信息，包括文件名称、责任人、脚本类型、描述。暂只支持描述配置项的修改。

  图 853: 脚本文件属性



27.3.8 删除脚本文件

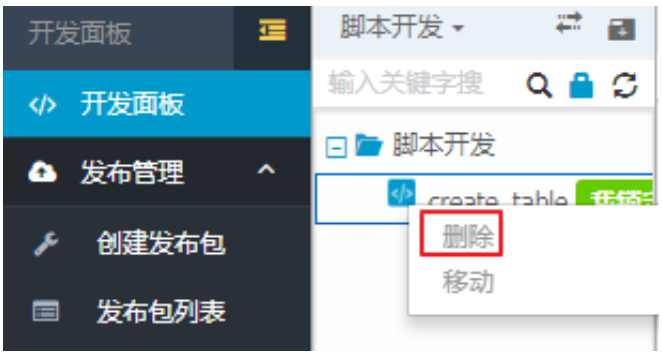
开发者也可删除已创建的脚本文件。

操作步骤

1. 以开发角色身份登录阿里云大数据开发平台；

2. 单击顶部菜单栏中的**数据开发**，导航至**开发面板** > **文件目录树**；
3. 右击选中的文件，单击删除即可。

图 854: 脚本文件管理列表



27.3.9 添加资源

开发者可上传本地自定义的jar、python或文件作为资源，在节点运行时调用。

操作步骤

1. 以开发角色身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据开发**，导航至**开发面板** > **新建**或导航至**管理面板** > **资源管理**；
3. 单击**上传资源**。
4. 在**资源上传**弹出框中，填写各配置项；

图 855: 资源上传弹出框



针对资源上传弹出框中的配置项说明如下：

- 名称：资源名称将自动获取，内容为资源文件名称+后缀，需在项目空间中唯一。
- 类型：支持jar、python、file、archive四种类型。
  - jar：jar资源为本地编译好的jar包，通常情况下，通过UDF和MaxCompute MR来使用；
  - python：python资源通常供shell调用或UDF使用；
  - file：包括自定义参数的shell脚本、xml配置文件、txt配置文件等；
  - archive：包括.zip/.tag/.tar.gz/.tar 四种格式压缩包。

- 上传：单击选择本地需要上传的资源文件，不支持多个资源文件上传。若选择上传的资源文件与选择的类型不符，提交时将提示“文件类型与选择类型不符”。
- 描述：针对当前资源文件的简单描述。
- 是否上传为ODPS资源：默认勾选，表示将该资源文件注册到MaxCompute（ODPS）计算引擎中。通常情况下，该资源文件供MaxCompute UDF和MaxCompute MR使用时，需勾选该选项。

5. 单击**提交**。

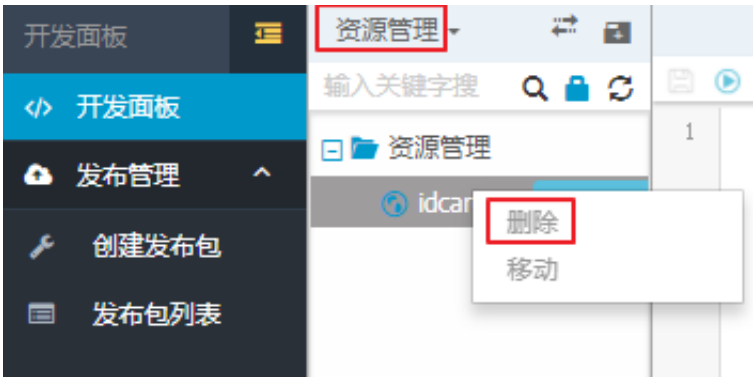
27.3.10 删除资源

开发者可删除已有的资源文件。

操作步骤

1. 以开发角色身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据开发**，导航至**开发面板** > **文件目录树** > **资源管理**。
3. 右击选中的文件，单击**删除**即可。

图 856: 资源文件管理列表



27.3.11 注册函数

阿里云大数据开发平台为开发者提供了SQL计算功能，用户可以在MaxCompute SQL中使用系统自带的SQL内建函数完成一定的计算和计数功能。

背景信息

系统内建函数，请在*MaxCompute#原ODPS#代码编辑器*中单击**函数**，查看每个函数的详情介绍并对函数进行引用。

如果系统内建函数无法满足要求时，用户可以开发自定义函数（User Defined Function，简称UDF），支持Java和Python两种语言实现。UDF 的使用方式与MaxCompute SQL中普通的内建函数相同。

本节以创建Java UDF为例，介绍注册函数的操作步骤。

操作步骤

1. 以开发角色身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据开发**，导航至**开发面板** > **新建**或**管理面板** > **资源管理**。
3. 单击**上传资源**。
4. 在**资源上传**弹出框中，填写各配置项。

图 857: 资源上传



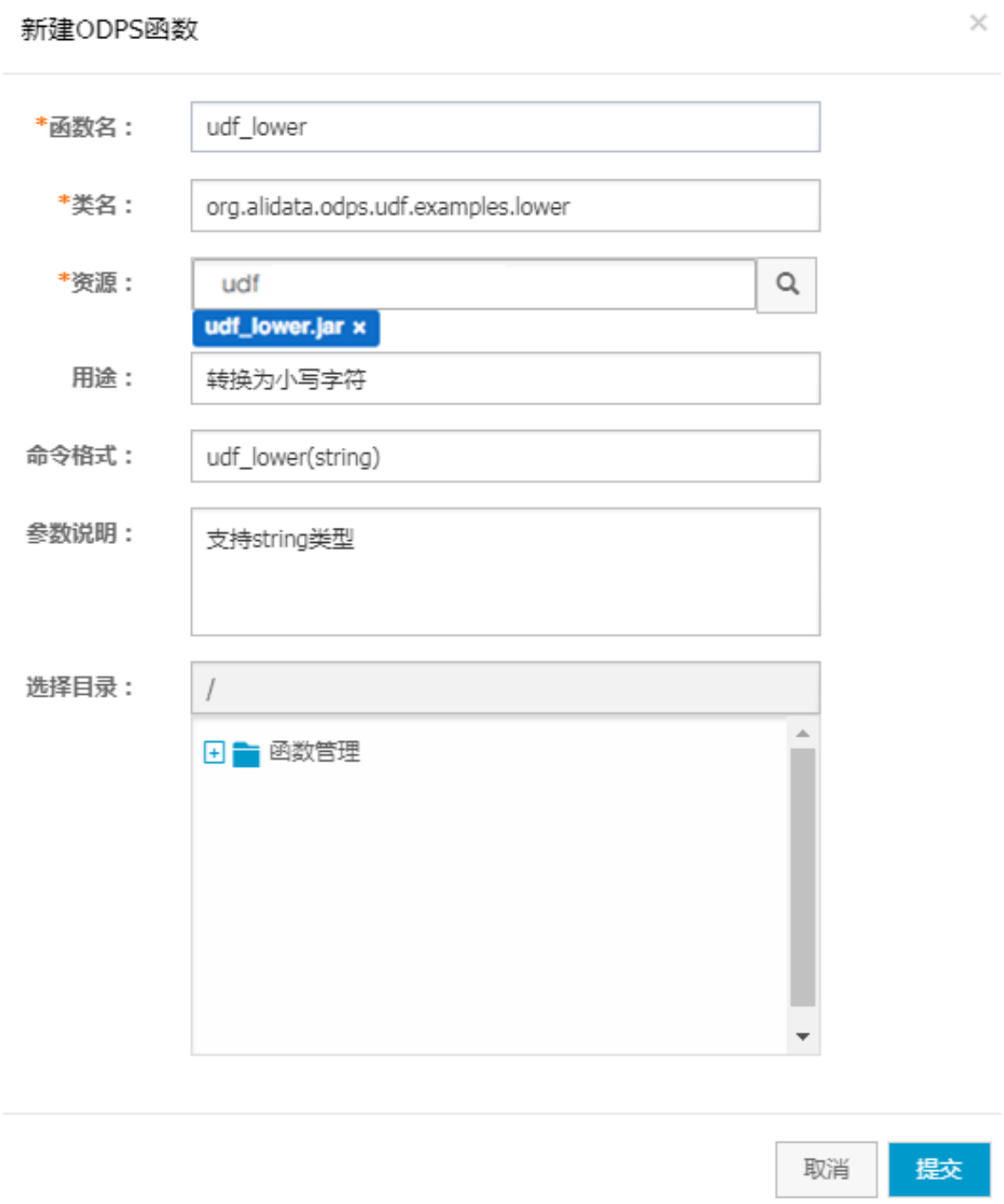

```
 -----udf_lower.jar源码-----
    package org.alidata.odps.udf.examples;
import com.aliyun.odps.udf.UDF;

publicfinalclassLowerextendsUDF {
public String evaluate(String s) {
if (s == null) { returnnull; }
return s.toLowerCase();
      }
    }
```

5. 单击**提交**。
6. 导航至**开发面板** > **新建**。

7. 单击**新建函数**。
8. 在**新建ODPS函数**弹出框中，填写各配置项，如图 *858: 新建函数弹出框*所示：

图 858: 新建函数弹出框



新建函数对话框中配置项具体说明如下：

- 函数名：UDF函数名，这个名字就是SQL中引用该函数所使用的名字，需全局唯一，且注册函数后不支持修改。

- 类名：如果是java UDF，这个名字就是从顶层包名一直到实现UDF类名的fully qualified class name。如果是python UDF，这个名字就是python脚本名.类名。
- 资源：支持模糊匹配查找本项目空间中已添加的资源。
- 用途：针对当前UDF作用的简单描述。
- 命令格式：该UDF的具体使用方法示意。
- 参数说明：支持输入的参数类型以及返回参数类型的具体说明。

9. 单击**提交**。

说明:

- 开发UDF需按照MaxCompute UDF框架的规定，实现函数功能，并进行编译；
- MaxCompute UDF默认Java版本为1.7，Python版本为2.7；
- Java UDF必须继承类com.aliyun.odps.udf.UDF；
- Python UDF必须通过annotate指定函数签名。

```
/***-----annotate指定函数签名示例------***/
from odps.udf import annotate
@annotate("bigint,bigint->bigint")
/***-----------------------------------------------***/
```

27.3.12 注销函数

开发者可注销已注册的函数。

操作步骤

1. 以开发角色身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据开发**，导航至**管理面板** > **函数管理**。

图 859: 函数管理列表



3. 在搜索框中输入关键字来模糊匹配查找需删除的函数。
4. 右击选中的函数，单击**删除**。

   通过上述步骤，已注册的UDF便可从数据开发工作台和底层MaxCompute（ODPS）计算引擎中注销。

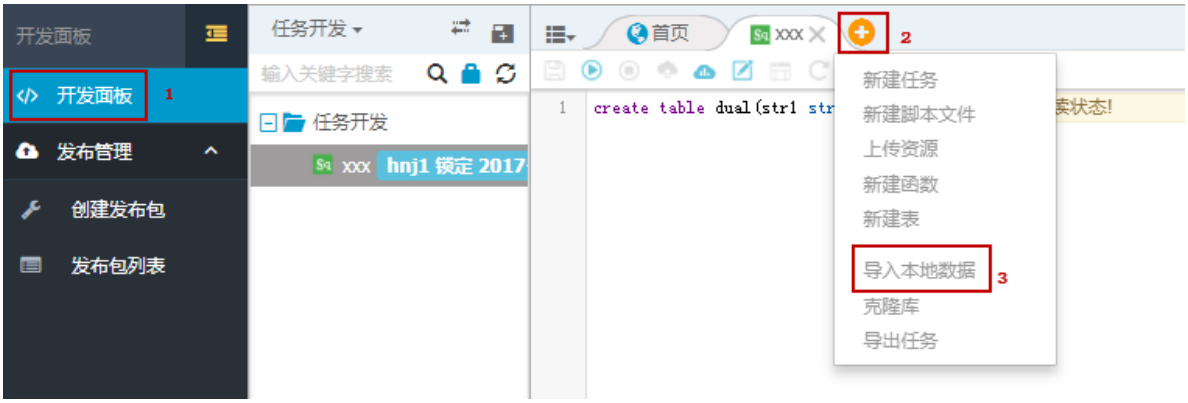
27.3.13 导入本地数据

阿里云大数据平台提供导入本地数据，只支持txt和csv文件类型。

操作步骤

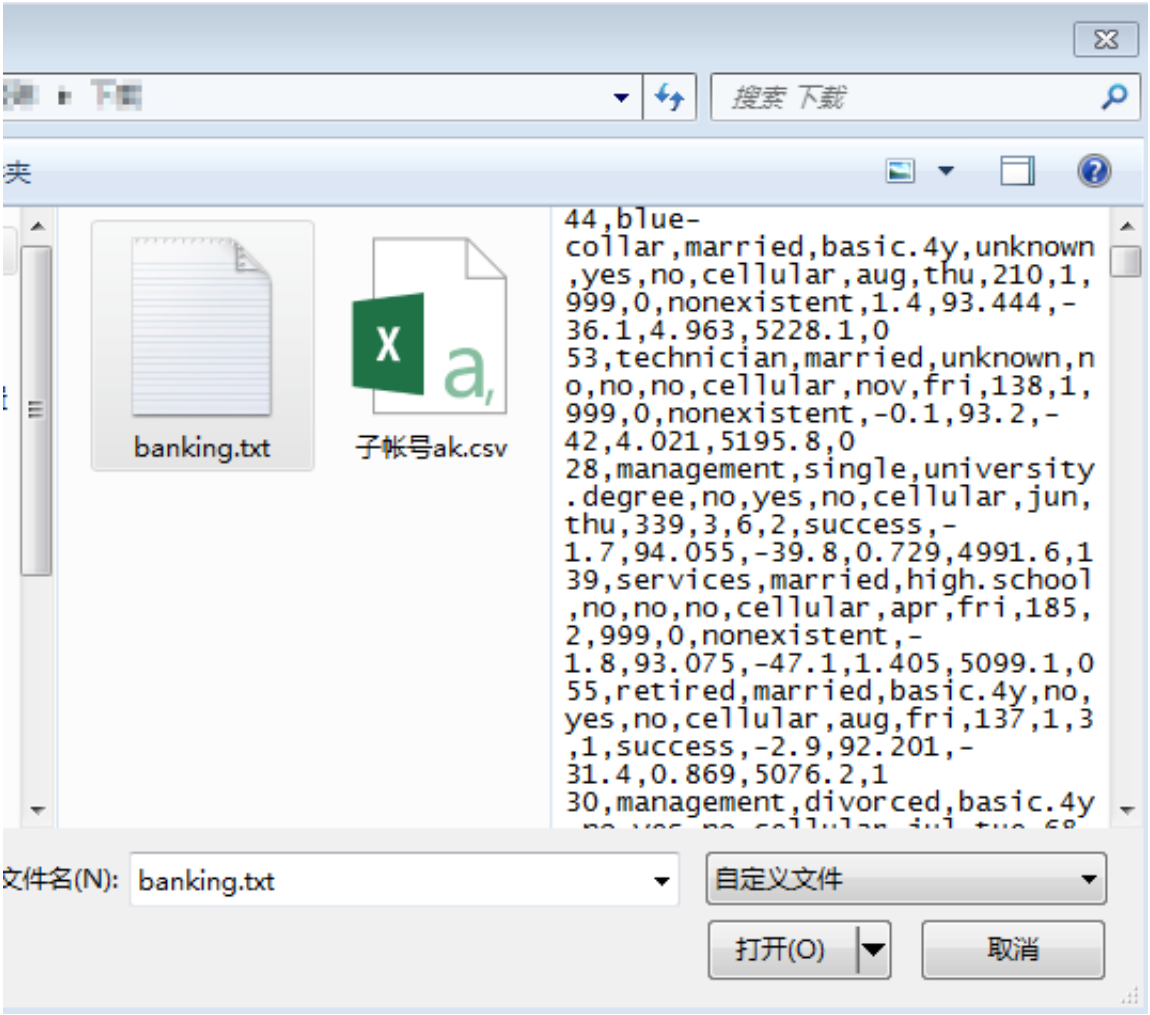
1. 以开发角色身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据开发**，导航至**开发面板** > **新建**。
3. 单击**导入本地数据**。

   图 860: 选择导入本地数据



4. 选择本地目录下的数据文件，单击**打开**。

   图 861: 选择本地文件



5. 在**本地数据导入**弹出框中，去掉勾选**首行为标题**并完成相应配置，单击**下一步**。

图 862: 本地数据导入配置



本地数据导入对话框中配置项说明：

- 已选文件：已选的导入本地文件名称；
- 分隔符号：支持逗号、Tab、分号、空格、|、# 和 &；
- 自定义分隔符：除系统内置的分隔符号外，用户可自定义分隔符；
- 原始字符集：支持GBK、UTF-8、CP936、ISO-8859；
- 导入起始行：支持1-11行可选，表示从该行开始导入本地数据。

6. 选择**导入表**并配置**字段匹配**。

图 863: 字段配置



说明: 导入表名支持模糊匹配搜索本项目空间中已有的表。

7. 单击**导入**。

27.3.14 发布任务

通常情况下，开发者在开发项目中完成工作流任务、资源、函数的开发并通过测试后，需将开发项目中已通过测试的工作流任务发布至生产项目进行调度生产。

操作步骤

1. 以项目管理员/开发/运维角色登录阿里云大数据开发平台。
2. 单击顶部切换至相应的项目空间下，导航至**数据开发 > 发布管理**。
3. 单击**创建发布包**。

4. 单击需要发布的对象类型，然后单击**加入待发布列表**中的**添加**。

图 864: 创建发布包页面



说明: 您需确保当前项目已绑定发布目标项目，方可执行发布操作。若未绑定，请联系管理员前往**项目管理**，绑定发布目标项目。

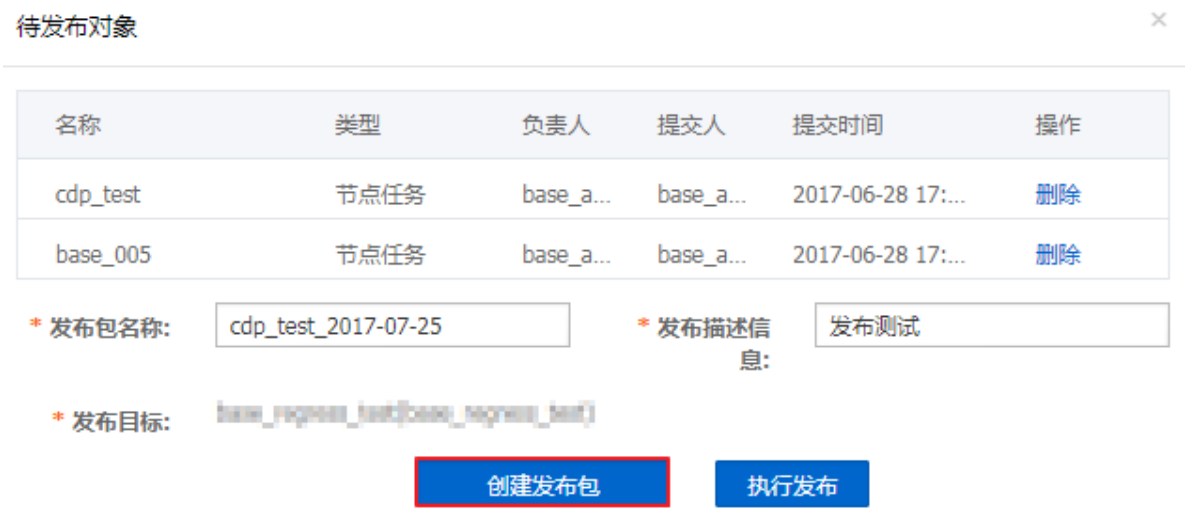
5. 单击右上角的发布篓，弹出待发布对象页面。

图 865: 已添加发布篓



6. 在**待发布对象**页面中，填写发布描述信息，单击**创建发布包**。

图 866: 待发布对象



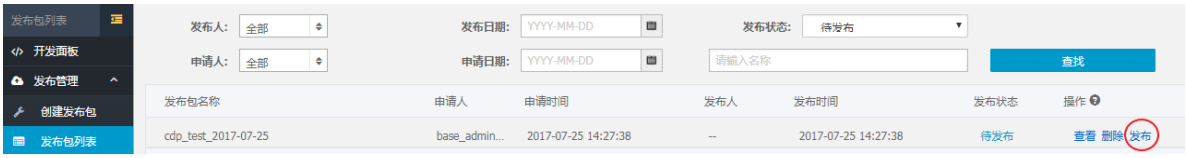
说明:

- **创建发布包**的操作只针对项目管理员、开发、运维角色用户开放；
- **执行发布**的操作只针对项目管理员和运维角色用户开放。

具体权限点划分请参见：权限点划分。

7. 跳转至发布包列表页面，单击发布包列表操作栏中的**发布**。

图 867: 发布包页面



8. 若有权限，则弹出**执行发布**对话框，单击**确定**。

发布包状态从**待发布**变为**发布成功**，表示发布包已经成功发布至目标项目。

说明: 工作流任务从开发项目发布中生产项目时，表、数据源配置等连接信息不随其一起发布至生产项目，需在生产项目中进行初始化。

27.3.15 查看MaxCompute项目名称

开发者通常在开发的过程中，需要用到MaxCompute（原ODPS）项目名称。

操作步骤

1. 登录阿里云大数据开发平台。
2. 单击顶部菜单中项目管理，导航至**项目属性 > 计算引擎配置**。
3. 查看**ODPS项目名称**中对应的名称。

图 868: 项目管理-计算引擎配置



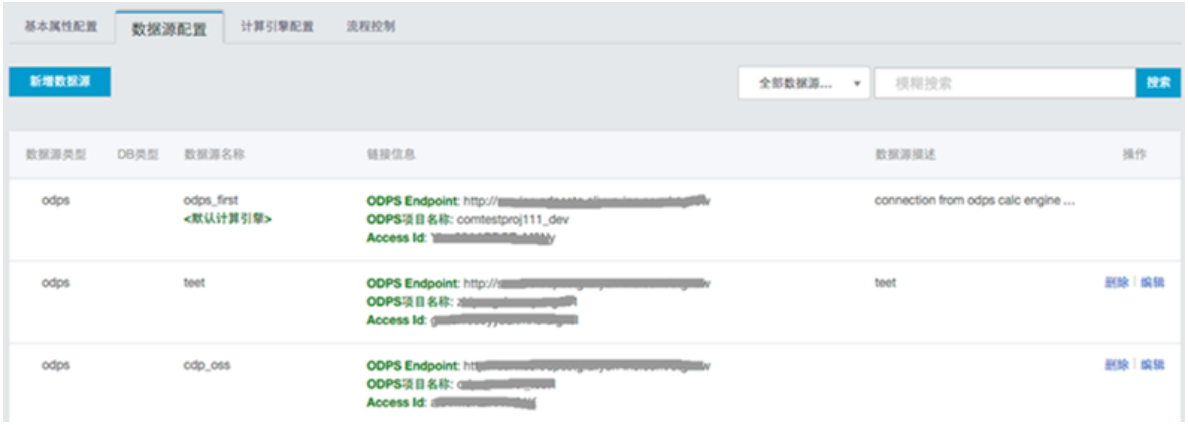
27.3.16 查看数据源配置

在数据同步任务开发过程中，开发者会遇到查看数据源配置的情况。

操作步骤

1. 登录阿里云大数据开发平台；
2. 单击顶部菜单中项目管理，导航至**项目属性 > 数据源配置**；
3. 查看数据源配置信息。

图 869: 数据源配置



27.4 数据开发

开发者在阿里云大数据平台**数据开发**套件中可以进行工作流任务设计、MaxCompute 代码开发、发布工作流任务和代码等操作。

27.4.1 概述

根据开发主要面向开发者提供可视化工作流任务设计器、MaxCompute代码编辑器，您可通过拖拽节点（SQL（MaxCompute SQL）、MR（MaxCompute MR）、数据同步、机器学习、SHELL和虚拟节点）和连线的方式来构建上下游依赖关系。

27.4.2 权限点划分

在数据开发模块中除组织管理员外，其余角色用户，包括：项目管理员、开发、运维、部署和访客的权限点如下表所示：

表 98: 数据开发模块权限点划分

父权限点名称	权限点名称	项目管理员	开发	运维	部署	访客
发布管理	创建发布包	√	√	√	/	/
	查看发布包列表	√	√	√	√	√
	删除发布包	√	√	√	/	/
	执行发布	√	/	√	√	/
	查看发布包内容	√	√	√	√	√
按钮控制	停止	√	√	/	/	/
	格式化	√	√	/	/	/
	编辑	√	√	/	/	/
	运行	√	√	/	/	/
	jar	√	√	/	/	/
	放大	√	√	/	/	/
	保存	√	√	/	/	/
	展开/收起	√	√	/	/	/
	删除	√	√	/	/	/
代码开发	保存提交代码	√	√	/	/	/
	查看代码内容	√	√	√	√	√
	创建代码	√	√	/	/	/
	删除代码	√	√	/	/	/
	查看代码列表	√	√	√	√	√
	运行代码	√	√	/	/	/
	修改代码	√	√	/	/	/
函数开发	查看函数详情	√	√	√	√	√
	创建函数	√	√	/	/	/
	查询函数	√	√	√	√	√
	删除函数	√	√	/	/	/
节点类型控制	机器学习	√	√	√	√	√

父权限点名称	权限点名称	项目管理员	开发	运维	部署	访客
	ODPS MR	√	√	√	√	√
	数据同步	√	√	√	√	√
	ODPS SQL	√	√	√	√	√
	Shell	√	√	√	√	√
	虚拟节点	√	√	√	√	√
资源管理	查看资源列表	√	√	√	√	√
	删除资源	√	√	/	/	/
	创建资源	√	√	/	/	/
工作流任务开发	保存工作流任务	√	√	/	/	/
	查看工作流任务内容	√	√	√	√	√
	提交节点代码	√	√	/	/	/
	修改工作流任务	√	√	/	/	/
	查看工作流任务列表	√	√	√	√	√
	修改owner属性	√	/	/	/	/
	打开节点代码	√	√	/	/	/
	删除工作流任务	√	√	/	/	/
	创建工作流任务	√	√	/	/	/

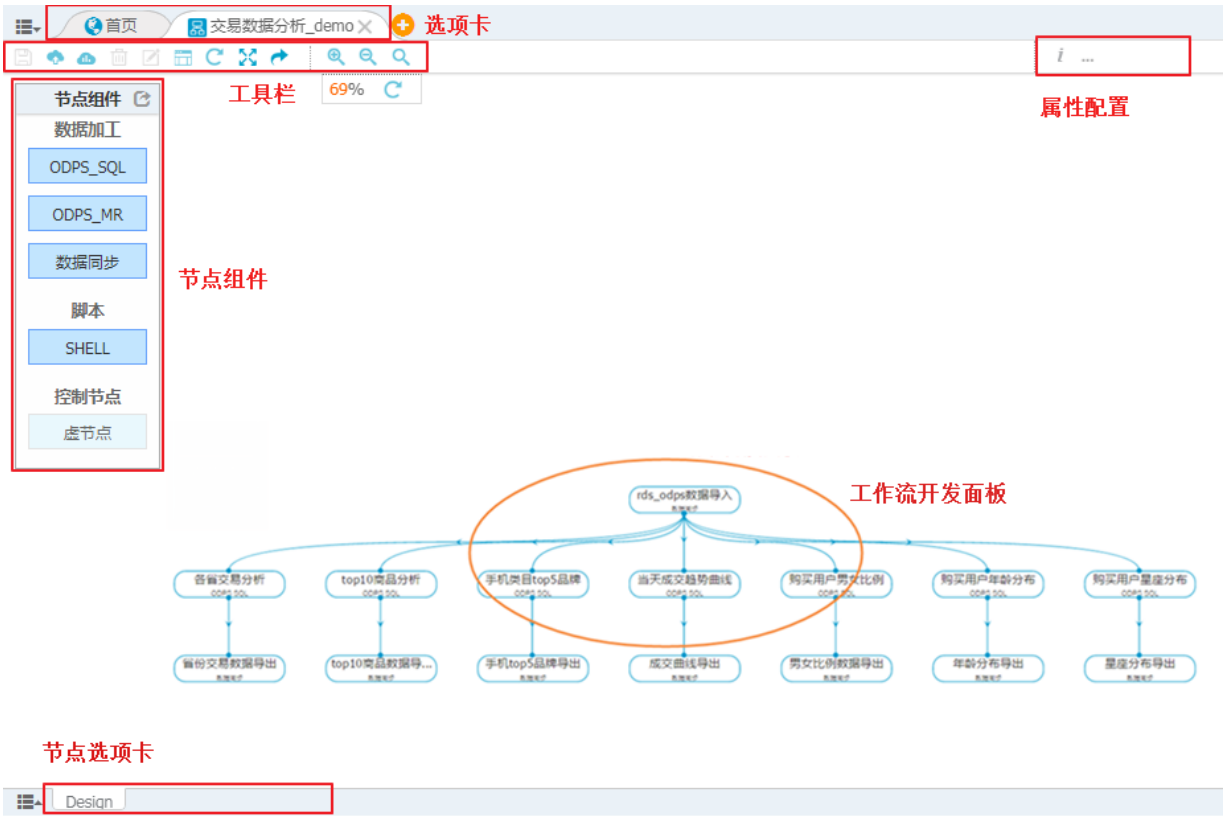
27.4.3 工作流任务设计器

27.4.3.1 设计器介绍

工作流任务是由多个节点和节点间的依赖关系所组成的一组逻辑和规则，形成一张有向无环图（DAG图）。

开发者可通过阿里云大数据平台**数据开发** > **开发面板** > **新建**来新建工作流任务进入工作流任务设计器，在工作流任务设计器中通过拖拽不同类型节点并连线的方式来开发一个工作流任务，如下图图 *870:* 工作流任务设计器所示：

图 870: 工作流任务设计器



工作流任务设计器各区域具体介绍如下：

- 工作流任务：显示当前打开的工作流任务，可切换和关闭。
  - 鼠标悬浮会显示完整工作流任务名称。
  - 鼠标点击可以左右拖拽变换选项卡位置。
  - 最多能同时打开10个工作流任务或任务，可通过**更多操作**来关闭所有、关闭其他、关闭额外（隐藏的选项卡）的工作流任务。
  - 当工作流任务选项卡有 时表示该工作流任务未保存。
- 工具栏：工作流任务设计器中常用的工具。

序号	标识	标识说明
1		**保存**，保存当前工作流任务变更，支持快捷键：ctrl+s。
2		**提交**，提交当前工作流任务到调度系统，提交成功后会释放锁定。同时，提交成功工作流任务里有变更的节点就会新增一个版本。
3		**测试运行**，提交当前工作流到调度系统，并触发生成测试实例，通过输入的业务日期进行任务测试。
4		**删除**，点击删除工作流任务选中的节点。

序号	标识	标识说明
5		**编辑**，工作流任务处在释放锁或者被他人锁定状态时，无法编辑工作流任务，可以点击“编辑”操作锁定该工作流任务进行编辑操作。
6		**放大/缩小**，放大/缩小工作流任务DAG图。
7		**全屏**，工作区全屏。
8		**格式化**，工作流任务DAG图自动布局显示。
9		**重新加载**，当工作流任务有变动但是没有保存时，可以重新加载到最初的状态。
10		**搜索**，搜索工作流任务中的节点，达到快速定位。

- 节点组件：工作流任务设计器支持的节点组件类型，您可单击具体节点组件进行拖拽至开发面板。
  - SQL（ODPS_SQL）类型

    MaxCompute SQL采用的是类似于SQL的语法，可以看作是标准SQL的子集，但不能因此简单的把MaxCompute SQL等价成一个数据库，它在很多方面并不具备数据库的特征，如事务、主键约束、索引等。
  - MR（ODPS_MR）类型

    MaxCompute提供了MapReduce 编程接口。用户可以使用MapReduce提供的接口 （Java API）编写MapReduce程序处理MaxCompute中的数据。在阿里云大数据平台中，MaxCompute MR节点（或任务）需要与资源一同使用。
  - 数据同步

    数据同步节点任务是阿里云大数据平台对外提供的稳定高效、弹性伸缩的数据同步云服务。用户利用数据同步节点可以轻松地将业务系统数据同步到MaxCompute上来。
  - Shell脚本

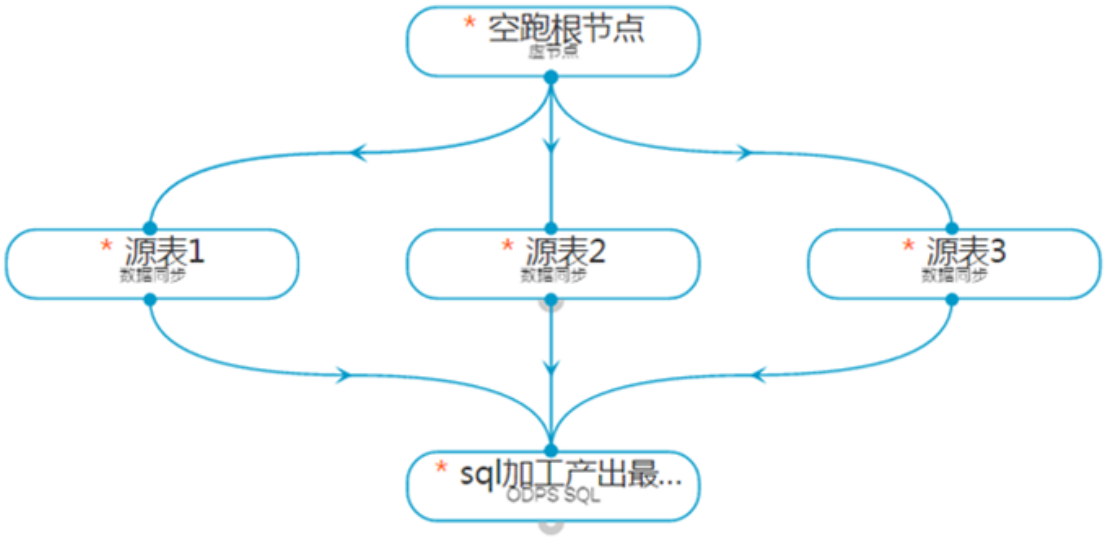
    Shell节点支持标准的Shell语法，不支持交互式语法。
  - 虚拟节点（virtual）

    虚拟节点属于控制类型节点，即：它不产生任何数据的空跑节点，是常用来项目统筹节点的根节点。

说明： 工作流任务里最终输出表有多个分支输入表，且这些输入表依赖关系时就经常用到虚拟节点。

入门示例：输出表由 3 个数据同步任务导入的源表经过 MaxCompute SQL 任务加工产出，这 3 个数据同步任务没有依赖关系，MaxCompute SQL 任务需要依赖 3 个同步任务，则工作流任务可以为如图所示：

图 871: 入门示例



用一个虚拟节点作为工作流任务的起始根节点，3个数据同步任务依赖虚拟节点，MaxCompute SQL加工任务依赖3个同步任务。

- 开发面板：工作流任务设计器的开发面板，展示当前工作流任务包含的节点及上下游依赖关系。
  - 选择的节点可通过拖拽变更其坐标位置。
  - 鼠标单击画布空白区可拖拽整个DAG图的坐标位置。
  - 可通过鼠标连线的方式来创建上下游依赖关系。当鼠标放至父节点底部突出的小圆点时，鼠标将呈十字型，此时可以单击引出连线并拖拽线到子节点图上松开，即可完成节点节点依赖关系配置。
  - 右击选中的节点可以查看节点代码（进入代码编辑区）、保存节点、删除节点、查看节点属性。当工作流任务处于锁定状态时，右键菜单只有查看节点代码选项。



- 鼠标对节点依赖线右键可以删除依赖线，即删除两个节点的依赖关系。
- 双击节点可以进入节点代码编辑区。

- 属性配置：包括工作流任务属性和节点属性的配置。
  - 工作流任务属性：主要是配置工作流任务各种调度属性，详情请参见工作流任务属性配置。
  - 节点属性：鼠标单击选中某个节点后会出现节点属性组件，主要是配置节点基本属性，详情请参见配置节点属性。
- 节点选项卡：当前打开的节点代码或配置，可单击切换进入节点代码编辑页面。

27.4.3.2 工作流任务属性配置

工作流任务是一个DAG 图（有向无环图），其描述了作业中多个节点之间的逻辑（依赖关系）和规则（运行约束）。

您可通过工作流任务开发面板来展开**工作流任务属性配置**，如图 *872:* 工作流任务属性配置所示：

图 872: 工作流任务属性配置



工作流任务属性配置包括**基本属性**、**调度属性**和**依赖属性**三大部分：

- **基本属性**
  - 工作流任务名称：新建工作流任务时填写的工作流任务名称，不支持修改。
  - 责任人：默认为当前登录人，也可修改为本项目的其他成员。只有当前负责人和管理员可以修改。
  - 描述：新建工作流任务时填写的描述，支持修改。
- **调度属性**
  - 是否启用调度：默认不启动（工作流任务不自动调度但可以手动调度）。若勾选则代表当前工作流任务将由系统自动调度，同时需配置调度生效日期、调度周期、开始时间、结束时间等信息。

- 调度生效日期：调度将在有效日期内生效并自动调度，反之，在有效期外的工作流任务将不会自动调度，也不能手动调度。
- 调度周期：设置工作流任务调度频次，支持**天/周/月/分钟/小时**。

  说明: 当创建一次性调度工作流任务时，工作流任务不具备调度属性配置选项；只有创建周期性调度工作流任务时，工作流任务才具有调度属性配置选项。

  - 天调度：每天调度一次，可设置具体调度时间。

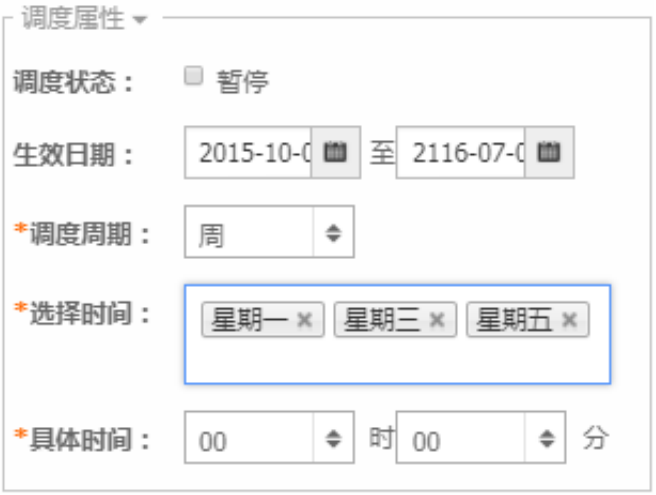
    图 873: 天调度

    上图表示在调度生效日期内，该工作流任务在每天00：00时进行调度执行，若有父工作流任务则要等父工作流任务运行成功后才能调度执行。

  - 周调度：每周可调度多次，支持设置具体调度时间。

    图 874: 周调度

上图表示在调度生效日期内，该工作流任务在每周的星期一、星期三和星期五的00：00时执行调度，而在星期二、星期四、星期六和星期日将不调度。

- 月调度：每月调度多次，支持设置具体调度时间。

图 875: 月调度



上图表示在调度生效日期内，该工作流任务在每月1日、2日的00：00时进行调度执行，而在每月3日到月末都将不调度。不支持每月月末调度，如选择每月31日，则没有31日的那个月整个月都将不调度执行。

- 分钟调度：每天一个时间段里每隔 5*n 分钟调度一次，目前分钟调度只支持5分钟的倍数间隔。

图 876: 分钟调度



上图表示在调度生效日期内，该工作流任务在每天的开始时间00：00至结束时间23：59内，每15分钟进行调度执行。

- 小时调度：每天一段时间内间隔 1*n 小时调度一次。

图 877: 小时调度



上图表示在调度生效日期内，该工作流任务在每天的开始时间08:00至结束时间20:59内，每隔1个小时整点调度一次。

说明: 当前节点/工作流任务调度执行时，若当前节点/工作流任务存在上游，则需上游执行并返回成功，且自身的定时时间已到或已过时，才会触发当前节点/工作流任务调度执行；若当前节点/工作流任务不存在上游，只需满足自身定时时间这一个条件即可调度执行。

- **依赖属性**
  - 所属项目：当前组织内的所有项目空间，可在下拉列表中进行选择。
  - 上游任务：对应所属项目空间中的工作流任务，用来设置当前工作流任务的上游工作流任务，非必填项。支持工作流任务名称模糊匹配查询。
  - 自动推荐：根据任务中的数据表流向，系统将自动判断并筛选出当前工作流任务可能依赖的其他任务，并推荐告知用户。

说明: 一个工作流任务可以依赖多个上游工作流任务，同样，一个工作流任务可被多个工作流任务依赖。依赖属性为非必填项，当下游工作流任务需依赖上游工作流任务产出的数据，则可配置依赖关系。可以使用推荐依赖的功能，让系统帮助您选择依赖的任务。

- **跨周期依赖属性**

  默认为没有跨周期依赖，可以选择**等待当前任务前一周期执行完成后再执行**、**等待当前任务下游任务执行完成后再执行**或**等待自定义任务的上一周期执行完成后再执行**。

27.4.3.3 节点属性配置

节点属于工作流任务的子对象，也称之为任务，是阿里云大数据平台数据处理和分析过程的最基本单元，每个任务对应DAG图中的一个节点，其可以是SQL Query、命令和MapReduce程序等。

在工作流任务设计器中，节点可以是节点组件（MaxCompute SQL、MaxCompute MR、数据同步、SHELL、虚节点）中的一种，您在工作流任务开发面板中选中节点并展开**节点属性配置**，如图 *878:* 节点属性所示：

图 878: 节点属性



节点属性配置包括**基本属性**、**调度属性**和**节点版本**三大部分。

- **基本属性**

1. 节点名称：新建节点时填写的节点名称，不支持修改。
2. 责任人：默认为当前用户，也可以修改成本项目的其他成员。只有当前负责人可以修改责任人。
3. 节点类型：当前节点的类型，不支持修改。
4. 描述：新建节点时填写的描述，支持修改。

- **调度属性**

  节点执行的起调时间（时分）。该属性只有当创建周期性调度工作流任务时，节点才具备起调时间配置项。

- **节点版本**

  节点每修改提交一次都将产生一个版本。节点版本则显示节点的历史版本列表，包括版本号、提交人、提交时间、描述和操作。目前只针对MaxCompute SQL、SHELL、MaxCompute MR三种类型节点的版本。也可以勾选两个版本进行代码对比。

27.4.4 单节点任务编辑器

单节点任务与工作流任务类似，是阿里云大数据平台数据处理和分析过程的最基本单元之一，包括SQL（MaxCompute SQL）、MR（MaxCompute MR）、Shell、数据同步等任务类型。其中所有的工具栏、属性布局请参见：工作流任务设计器。

27.4.5 MaxCompute代码编辑器

MaxCompute（原ODPS）代码开发编辑器主要由工具栏、代码编辑区、参数配置组件、数据表组件、资源组件、函数组件等组成，支持关键字智能联想和高亮显示。

您可通过双击工作流任务中的节点（仅限MaxCompute SQL、MaxCompute MR、SHELL）或新建脚本文件进入代码编辑器两种方式进行查看，进入的代码编辑器存在细微差异，但大体相同。以脚本文件为例，说明如下：

图 879: MaxCompute代码编辑器



MaxCompute代码编辑器各个部分的介绍如下：

- 工具栏：MaxCompute 代码编辑常用的工具。

序号	标识	标识说明
1		**保存**，保存当前工作流任务变更，支持快捷键：ctrl+s。
2		**运行**，选中部分代码运行，不选择代码则运行当前所有代码，多条语句会按顺序串行运行。
3		**停止**，正在运行代码时，不需要运行可以停止。
4		**编辑**，工作流任务处在释放锁或者被他人锁定状态时，无法编辑工作流任务，可以点击“编辑”操作锁定该工作流任务进行编辑操作。
5		**全屏**，工作区全屏。
6		**格式化**，代码格式化。
7		**重新加载**，当代码有变动但没有保存时，可以重新加载到最初的状态。



说明: 单击运行，代码将以该项目空间对应的计算引擎所配置的身份来运行MaxCompute任务。

- 代码编辑区：编写MaxCompute SQL、MaxCompute MR、Shell脚本区域。
- 参数配置  ：包括系统参数配置和自定义参数配置。

- 系统参数配置：当代码里用到系统变量（${bdp.system.bizdate}和${bdp.system.cyctime}），那么这里赋的值只作用于代码调试运行，节点调度时将使用系统参数默认值。
- 自定义参数配置：当代码里用到自定义变量，那么代码调试时需要赋固定值，需要提交到调度的话要改成调度参数再保存提交，节点调度将采用该参数进行赋值。

系统参数和自定义参数的详细说明请参见：系统调度参数。

- 找表  ：可模糊匹配查找本组织的表元数据信息。单击表名，可查看该表的列信息、分区信息以及数据预览。

图 880: 找表组件



也可对该表进行如下操作：

- 引用表：单击**引用**会在代码编辑区光标处输入表名。
- 查看表：单击**查看**可跳转至数据表详情页。
- 生成语句：勾选数据表的具体列，单击**生成语句**可在代码编辑区光标处生成select查询语句。

- 函数列表：可模糊匹配查找本项目空间内的系统内建函数和用户自定义函数。单击函数名，可查看函数类型、基本信息以及说明。单击**引用**会在代码编辑区光标处输入函数名。

图 881: 函数列表



- 资源列表：可模糊匹配查找本项目空间内的资源。单击**引用**会在代码编辑区光标处引用该资源文件。

图 882: 资源列表



- 代码选项卡：显示当前打开的代码选项卡，可切换或关闭。

27.4.6 系统调度参数

系统调度参数包括**系统参数**和**自定义参数**。

系统参数

目前系统参数只有两个，这两个系统参数只要代码里用到，不需要做额外的赋值动作，节点自动调度的时候就会自动替换。

系统参数分别为：

- ${bdp.system.bizdate}

  格式为yyyymmdd，日常调度默认为当前日期的前一天（年月日）。节点每天自动调度实例定时时间年月日减 1 天。
- ${bdp.system.cyctime}

格式　为yyyymmddhh24miss，日常调度实例定时时间（年月日时分秒）。　yyyy表示4位数年份，mm表示2位数月份，dd表示2位数天，hh24表示24小时制的时，mi表示2位数分钟，ss表示2位数秒。

入门示例：工作流任务设置为小时调度，每天00:00-23:59时间段里每隔1小时执行一次。

节点代码如下：

```
insert overwrite table tb1 partition(ds ='20150304')select
    c1,c2,c3
from (
    select * from tb2
    where ds ='${bdp.system.cyctime}') t
full outer join(
    select * from tb3
    where ds = '${bdp.system.bizdate}') y
on t.c1 = y.c1;
```

相应的工作流任务和节点属性配置如下：

图 883: 工作流任务属性



图 884: 节点参数配置



2015年10月27日当天，工作流任务生成24个实例：

- 第一个实例定时时间为2015-10-27 00:00:00，那么bdp.system.cyctime替换的结果为20151027000000，bdp.system.bizdate替换的结果为20151026。
- 第二个实例定时时间为2015-10-27 01:00:00，那么bdp.system.cyctime替换的结果为20151027010000，bdp.system.bizdate替换的结果为20151026。
- 以此类推，第24个实例定时时间为2015-10-27 23:00:00，那么bdp.system.cyctime替换的结果为20151027230000，bdp.system.bizdate替换的结果为20151026。

自定义参数

当系统参数不足以满足需求，您可以通过函数来对系统参数结果进行计算，但是函数执行过程毕竟会占用一定的计算资源，而调度也提供一个自定义参数，可以在调度时直接替换一些时间变量。参数具体为：

- $[...]基于系统参数${bdp.system.cyctime}进行自定义格式

  ${bdp.system.cyctime}=$[yyyymmddhh24miss] 可以任意分解组合，以及指定格式，如定时时间年月日$[yyyymmdd]、$[yyyy-mm-dd] 等格式可自定义，定时时间时分秒$[hh24miss]、$[hh24:mi:ss] 等格式可自定义。
- 获取+/-周期的方法：
  - 后 N 年：$[add_months(yyyymmdd,12*N)]
  - 前 N 年：$[add_months(yyyymmdd, -12*N)]
  - 后 N 月：$[add_months(yyyymmdd, N)]
  - 前 N 月：$[add_months(yyyymmdd, -N)]
  - 后 N 周：$[yyyymmdd + 7*N]
  - 前 N 周：$[yyyymmdd - 7*N]

- 后 N 天：$[yyyymmdd + N]
- 前 N 天：$[yyyymmdd - N]
- 后 N 小时：$[hh24miss + N/24]
- 前 N 小时：$[hh24miss - N/24]
- 后 N 分钟：$[hh24miss + N/24/60]
- 前 N 分钟：$[hh24miss - N/24/60]

入门示例：工作流任务设置为小时调度，每天00:00-23:59时间段里每隔1小时执行一次。

节点代码如下：

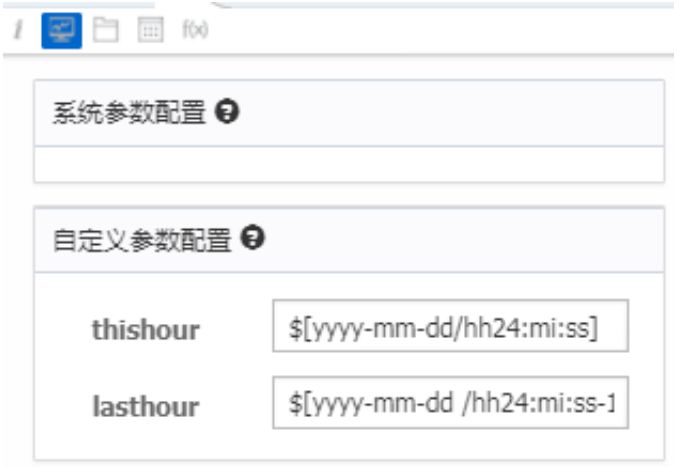
```
insert overwrite table tb1 partition(ds ='20150304')select
    c1,c2,c3
from (
    select * from tb2
    where ds ='${thishour}') t
full outer join(
    select * from tb3
    where ds = '${lasthour}') y
on t.c1 = y.c1;
```

工作流任务、节点属性如下：

图 885: 工作流任务属性



图 886: 节点参数配置



自定义参数配置如下：

thishour=$[yyyy-mm-dd/hh24:mi:ss]

lasthour =$[yyyy-mm-dd /hh24:mi:ss-1/24]

2015年10月27日当天，工作流任务生成24个实例：

- 第一个实例定时时间为2015-10-27 00:00:00，那么thishour替换的结果为2015-10-27/00:00:00，lasthour替换的结果为2015-10-26/23:00:00。
- 第二个实例定时时间为2015-10-27 01:00:00，那么thishour替换的结果为2015-10-27/01:00:00，lasthour替换的结果为2015-10-27/00:00:00。
- 以此类推第24个实例定时时间为2015-10-27 23:00:00，那么thishour替换的结果为2015-10-27/23:00:00，lasthour替换的结果为2015-10-27/22:00:00。

27.4.7 配置数据同步节点

您可向工作流任务设计器中拖入数据同步节点并双击进入数据同步节点配置界面，包括**选择数据来源和目标**、**选择要抽取的列**，**并映射到目标表字段**、**数据抽取和加载控制**和**流量与出错控制**四大配置项。

数据同步任务的四大配置项，因数据源的不同配置策略也有所不同，本文将其分为关系型数据库和无结构化数据同步两种类型来分别阐述。

27.4.7.1 选择数据来源和目标

关系型数据库同步

在数据同步任务配置过程中，首先需**选择数据源和目标**（新增数据源请联系项目管理员），并支持模糊匹配查找数据源和目标以及表名。当选择了源头和目标，其选项框末尾将显示对应数据源或目标类型。

图 887: 选择数据来源和目标



如下示意图，表示将MySQL数据源（cdp_mysql_jdbc）中的表enterprise_legal_person数据同步到MaxCompute目标中的表enterprise_legal_person。

图 888: 数据来源和目标配置



- **显示建表 DDL**：单击后，可将源头表的建表语句转化为符合MaxCompute SQL语法规范的DDL语句。如下图所示：

  图 889: 显示建表 DDL 弹出框



- **添加数据源**：单击添加数据源跳转到**项目管理>数据源配置**页面。

  说明: 目前数据同步任务支持的数据源类型包括：MaxCompute、RDS（MySQL、SQL Server、PostgreSQL）、Oracle、FTP、AnalyticDB、OSS、OCS、DRDS。**新增数据源**请联系项目管理员进行添加。

若源头为Mysql数据源，则数据同步任务还支持分库分表模式的数据导入（前提是无论数据存储在同一数据库还是不同数据库，表结构必须是一致的）。

图 890: 分库分表-数据来源和目标配置



分库分表可支持如下场景：

- **同库多表**：单击**搜索表**，添加需要同步的多张表即可。
- **分库多表**：首先单击**添加选择源库**，再单击**搜索表**来添加表。

如下图所示：

图 891: 添加选择源库



图 892: 多表搜索



无结构化数据同步

- FTP→MaxCompute

  若数据源为FTP，则选择数据来源和目标配置界面如下所示：

  图 893: 数据来源和目标配置



其中可通过**增加路径**按钮来添加多个文件同步到目标表中。同时文件路径支持正则表达式，如文件路径为 /tmp/data/*.data.log，则表示将读取 /tmp/data 目录下以 .data.log 结尾的文件。

- OSS→MaxCompute

  OSS数据同步至MaxCompute的配置页面如下所示：

  图 894: OSS 数据导入配置界面



单击增加Object可以添加多个Object数据的导入，如图中是将osstest/bool.gz Object导入至MaxCompute中。

27.4.7.2 字段配置

关系型数据库同步

其次需对字段映射关系进行配置，左侧**源表字段**和右侧**目标表字段**为一一对应的关系。

图 895: 字段映射关系



- 批量编辑

可批量编辑源表或宿表字段，通过此方式添加的表字段类型默认为空。

图 896: 批量编辑弹出框



- 增加/删除

  单击可单个增加/删除字段。

图 897: 增加/删除字段



说明: 自定义变量和常量的写入方法：如果需要把常量或者变量导入MaxCompute中表的某个字段，只需要单击插入按钮，然后输入常量或者变量的值，并且用英文单引号包起来即可，如变量‘${yesterday}’，然后再参数配置组件配置给变量赋值如yesterday=$[yyyymmdd]。

无结构化数据同步

- FTP→MaxCompute

FTP文件数据同步在字段配置界面如下所示：

图 898: FTP数据同步字段配置界面



其中FTP字段系统会默认生产5个字段，且字段类型都为string，需要用户根据指定文件的具体字段内容来进行增加/删除字段并修改字段类型。

- OSS→MaxCompute

  OSS Object数据导入至MaxCompute的字段配置同FTP数据导入至MaxCompute。

27.4.7.3 数据抽取和加载控制

数据抽取控制即数据抽取的过滤条件，而数据加载控制即数据写入时的规则。不同场景的数据同步任务配置界面不同。常见的数据同步任务类型，如下所示：

RDS/Oracle→MaxCompute

RDS/Oralce数据同步至MaxCompute中，界面如下：

图 899: 数据抽取和加载控制（RDS/Oracle>MaxCompute）



- 抽取控制，可参考相应的SQL语法填写where过滤语句（不需要填写where关键字），该过滤条件将作为增量同步的条件。

  说明:

  where条件即针对源头数据筛选条件，根据指定的column、table、where条件拼接SQL进行数据抽取。利用where条件可进行全量同步和增量同步，具体说明如下：

- 全量同步

  第一次做数据导入时通常为全量导入，可不用设置where条件；如只是在测试时，避免数据量过大，可将where条件指定为limit 10。

- 增量同步

  增量导入在实际业务场景中，往往会选择当天的数据进行同步，通常需要编写where条件语句，请先确认表中描述增量字段（时间戳）为哪一个。如tableA描述增量的字段为creat_time，那么在where条件中编写creat_time>${yesterday}，在参数配置中为其参数赋值即可。其中更多内置参数使用方法，请参见系统调度参数。

- 分区信息：分区是为了便于查询部分数据引入的特殊列，指定分区便于快速定位到需要的数据。支持常量和变量。
- 清理规则：
  - 写入前清理已有数据：导数据之前，清空表或者分区的所有数据，相当于insert overwrite。
  - 写入前保留已有数据：导数据前不清理任何数据，每次运行数据都是追加的，相当于insert into。

MaxCompute→RDS/Oracle

MaxCompute数据同步至RDS/Oracle中，界面如下：

图 900: 数据抽取和加载控制（MaxCompute>RDS/Oracle）

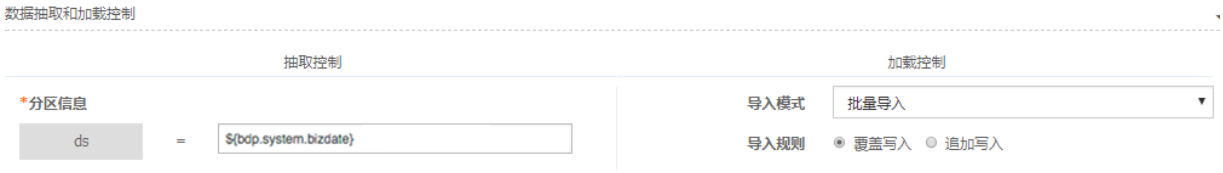


- 抽取控制：即配置源头表的分区信息，支持常量和变量。
- 加载控制：包含**准备语句**、**完成语句**和**主键冲突**三种类型配置项。

MaxCompute→AnalyticDB

MaxCompute数据同步至AnalyticDB中，界面如下：

图 901: 数据抽取和加载控制（MaxCompute→ AnalyticDB）



- 抽取控制：即配置源头表的分区信息，支持常量和变量。
- 加载控制：包含**导入模式**和**导入规则**两种类型配置项。

FTP→MaxCompute

FTP文件数据同步至MaxCompute，在数据抽取和加载控制配置界面如下：

图 902: FTP 数据导入抽取和加载控制配置项



- 抽取控制：包含**列分隔符**、**编码格式**、**null值**、**压缩格式**和**跳过表头**。
  - 列分隔符：默认为逗号，您可根据文件数据具体情况来配置。
  - 编码格式：默认为UTF-8，您可根据文件编码格式来具体填写。
  - null值：在文件中哪些字符表示null值，默认为null。
  - 压缩格式：支持none、gzip、bzip2三种格式。
  - 跳过表头：是否跳过表头选项。

OSS→MaxCompute

OSS数据同步至MaxCompute的数据抽取和加载控制配置项同FTP至MaxCompute。

27.4.7.4 流量与出错控制

流量与出错控制用来配置作业速率上限和脏数据检查规则，如图 *903:* 流量与出错控制所示：

图 903: 流量与出错控制



- 作业速率上限：即配置当前数据同步任务速率，支持最大为10MB/s（通道流量度量值是数据同步任务本身的度量值，不代表实际网卡流量）。

  说明: 若数据同步任务是RDS/Oracle→MaxCompute，在该页面中会有切分键配置。

- 切分键：读取数据时，根据配置的字段进行数据分片，实现并发读取，可提升数据同步效率。只有同步任务是RDS/Oracle数据导入至MaxCompute时，才显示切分键配置项。

以下为脏数据检查规则，可配置一个或两个，两个规则之间的关系：

- **当出错记录数超过**：当脏数据数量（即错误记录数）超过所配置的个数时，该数据同步任务结束。
- **错误百分比达到**：当脏数据数量（即错误记录数）超过所配置的百分比时，该数据同步任务结束。

27.4.8 本地数据导入

系统支持本地小于10MB的数据导入至MaxCompute（原ODPS）中，包括.txt 和.csv 的文件。

27.4.9 跨项目发布

通常情况下，开发者在开发项目中完成**工作流任务、资源、函数**的开发并通过测试后，需将开发项目中已通过测试的工作流任务发布至生产项目进行调度生产。

说明: 详细操作请参见发布任务。

27.5 数据管理

开发者在阿里云大数据开发平台**数据管理**模块中可以看到组织内全局数据视图、分权管理、元数据信息详情、数据生命周期管理、数据表权限管理审批等操作。

全局概览

查找数据

数据权限申请

新建表

收藏表

修改生命周期

修改表结构

隐藏表

修改表负责人

删除表

查看表详情

类目导航配置

27.5.1 概述

数据管理模块为开发者提供本组织范围内数据表搜索、数据表详情查看、数据表权限申请、收藏数据表、表所属类目配置（组织管理员可以配置类目权限）等功能，为项目管理员提供数据表权限审批与收回等功能。

27.5.2 权限点划分

在数据管理模块中运维、部署和访客角色用户都无权限进行相关操作，组织管理员、项目管理员和开发角色用户的权限划分如下表所示：

表 99: 数据管理模块角色与其权限划分

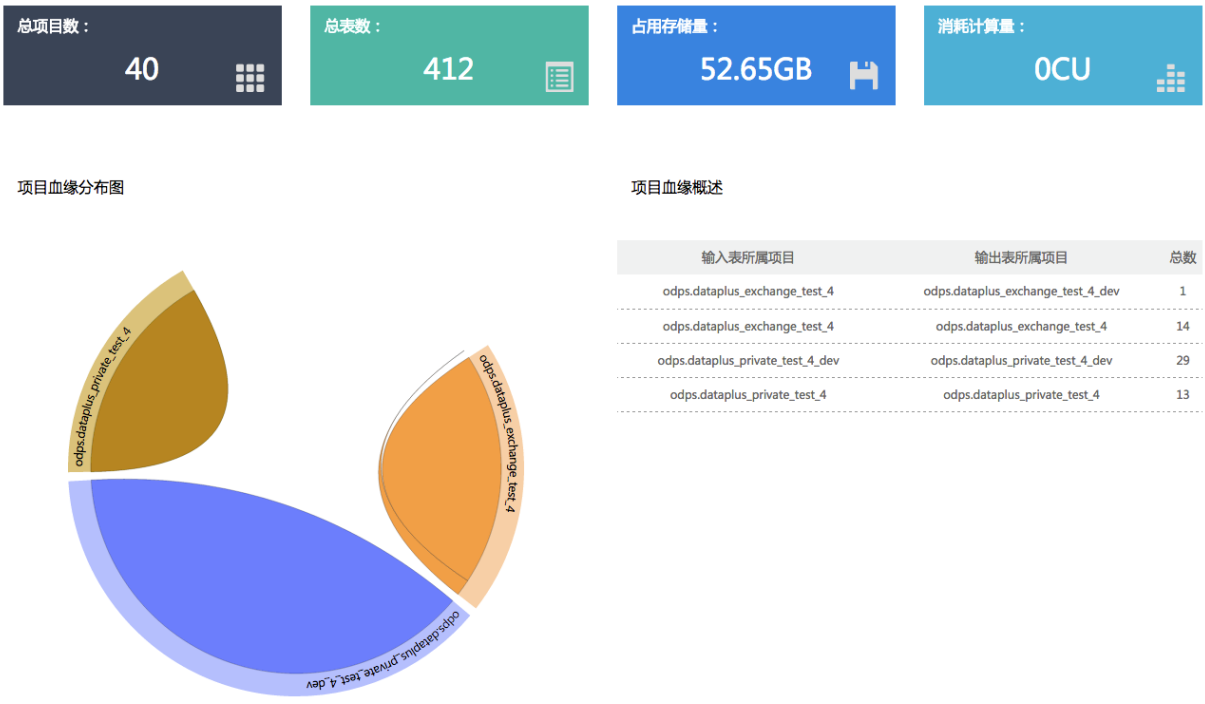
功能模块	权限点	组织管理员	项目管理员	开发
权限管理	权限审批与收回	/	√	/
管理配置	类目导航配置	√	/	/
数据管理	自己创建的表删除	√	√	√
	自己创建的表类目设置	√	√	√
	自己收藏的表查看	√	√	√

功能模块	权限点	组织管理员	项目管理员	开发
	组织下类目导航配置	√	/	/
	新建表	√	√	√
	自己创建的表取消隐藏	√	√	√
	自己创建的表结构变更	√	√	√
	自己创建的表查看	√	√	√
	自己申请的权限内容查看	√	√	√
	自己创建的表隐藏	√	√	√
	自己创建的表生命周期设置	√	√	√
	非自己创建的表数据权限申请	√	√	√

27.5.3 全局概览

您可通过**数据管理** > **全局概览**进入全局概览页面，全局概览页面中的统计都是在整个组织的前提下进行统计的，同时整个页面的数据都为离线产生，即页面中的数据为昨天的统计数据。

图 904: 全局视图





全局概览页面中相关内容说明如下：

- 总项目数、总表数、占用存储量、消耗计算量：都是在组织视角下统计项目空间个数、数据表个数、数据表占用的存储以及任务运行时所消耗的计算量（单位为1CU，1CU为一个核运行一天）。
- 项目血缘分布图：在组织视角下，以网络来刻画项目之间的血缘关系。
- 项目血缘概述：以表格的形式展示项目之间的血缘情况。
- 热门表：在组织视角下，被操作次数最多的数据表排行，显示前十名。
- 项目占用存储排行：在组织视角下，各个项目空间所占用的存储排行榜，取前十名。
- 表占用存储排行：在组织视角下，展示数据表占用存储量前十名。

27.5.4 数据搜索—找表

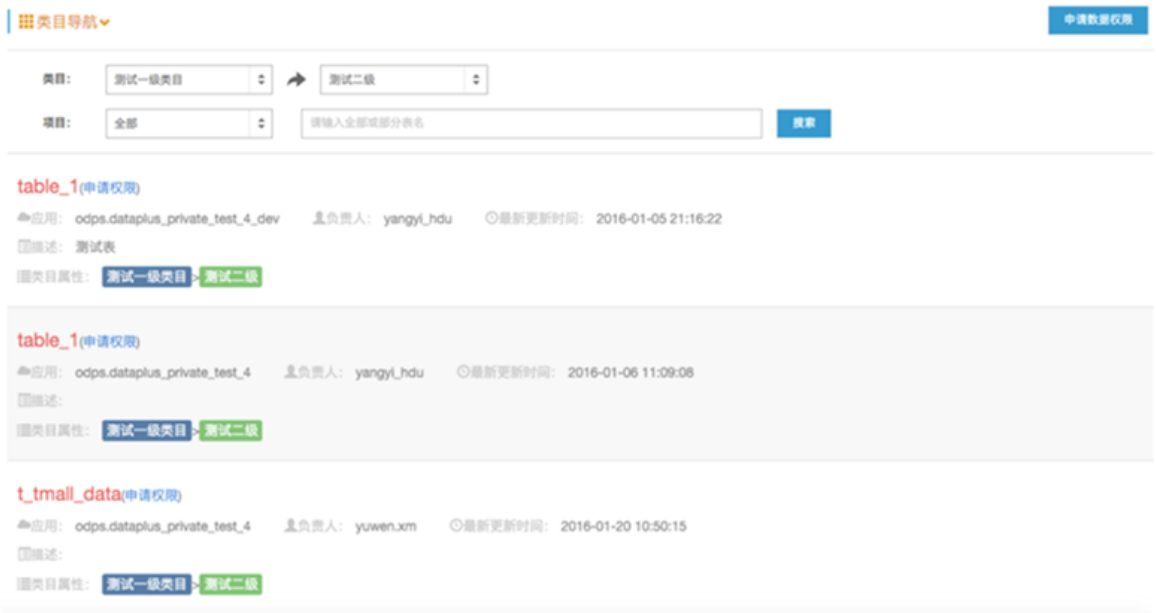
背景信息

您可通过如下方式进行本组织范围内（多项目空间）数据表的搜索：

操作步骤

1. 以开发者身份登录阿里云大数据平台。
2. 单击顶部菜单栏中的**数据管理**，导航至**查找数据**。
3. 在类目导航页面中，通过选择数据类目和在搜索框中输入表名进行模糊匹配的方式快速查找需要的表。

图 905: 数据搜索



说明:

数据搜索提供以下三种方式：

- 类目导航选择：查看当前类目下所有表。
- 具体表搜索：在搜索框里输入表名（支持表名模糊搜索）。
- 类目导航+搜索框中输入表名 + 项目：可以快速查找表。

27.5.5 申请数据权限

在阿里云大数据开发平台中，数据权限申请类型包括表、自定义函数和资源文件三种类型。

背景信息

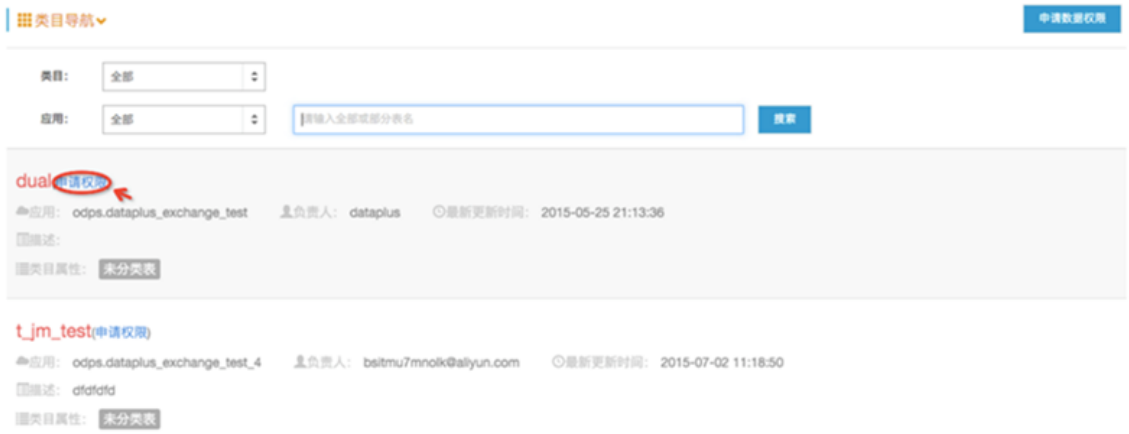
申请数据表权限有以下两种方式：

- 方式一

  a) 通过数据搜索找到需申请授权的数据表。

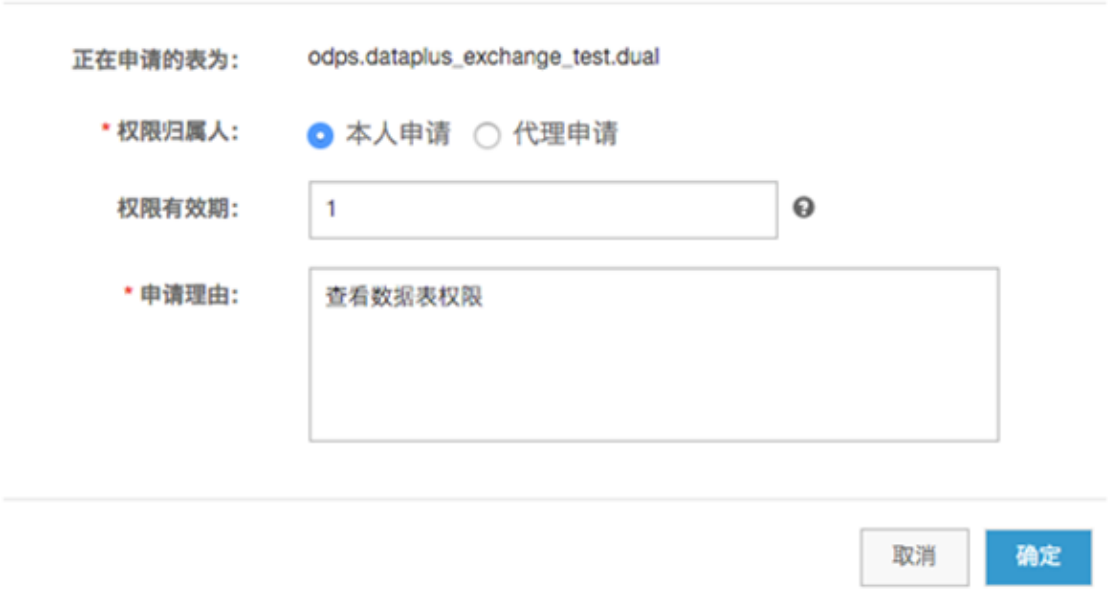
  b) 单击数据表操作栏中的**申请权限**。

图 906: 申请权限



c) 在**申请权限**弹出框中，填写相关配置项。

图 907: 本人申请



申请授权弹出框中的配置项说明如下：

- 权限归属人：支持**本人申请**和**代理申请**。
  - 本人申请：选择该项，审批通过后读表权限归属于当前用户。
  - 代理申请：选择代理申请，需填写代申请账号，相应的云账号则自动读取（只读，不支持修改）。审批通过后读表权限归属于被代理人。

图 908: 代理申请



- 申请理由：申请数据表权限理由的简单描述。

d) 单击**确定**。

- 方式二

此方式可完成自定义函数或资源文件的权限申请。

a) 登录阿里云大数据开发平台。

b) 单击顶部菜单栏中的**数据管理** > **查找数据**。

c) 单击**申请数据权限**。



d) 在**申请授权**对话框中，填写相关配置项。

- 选择申请**函数**权限

图 909: 表权限申请



配置项具体说明：

- 申请类型：选择**函数**。
- 权限归属人：支持**本人申请**和**代理申请**。
  - 本人申请：选择该项，审批通过后读表权限归属于当前用户；
  - 代理申请：选择代理申请，需填写代申请账号。审批通过后读表权限归属于被代理人。
- 项目名：选择所需申请表所在的项目名称（对应的MaxCompute项目名称），支持本组织范围内模糊匹配查找。
- 表名称：选择对应项目名称下的表，支持表名称模糊匹配查找。
- 申请理由：申请理由的简要说明，1000字以内。

- 申请Resource资源权限

图 910: Resource资源文件



配置项具体说明：

- 申请类型：选择**Resource资源**。
- 权限归属人：支持**本人申请**和**代理申请**。
  - 本人申请：选择该项，审批通过后读表权限归属于当前用户。
  - 代理申请：选择代理申请，需填写代申请账号。审批通过后读表权限归属于被代理人。
- 项目名：选择所需申请表所在的项目名称（对应的MaxCompute项目名称），支持本组织范围内模糊匹配查找。
- 资源名称：输入项目中的资源名称+后缀，如udf_lower.jar。
- 申请理由：申请理由的简要说明。

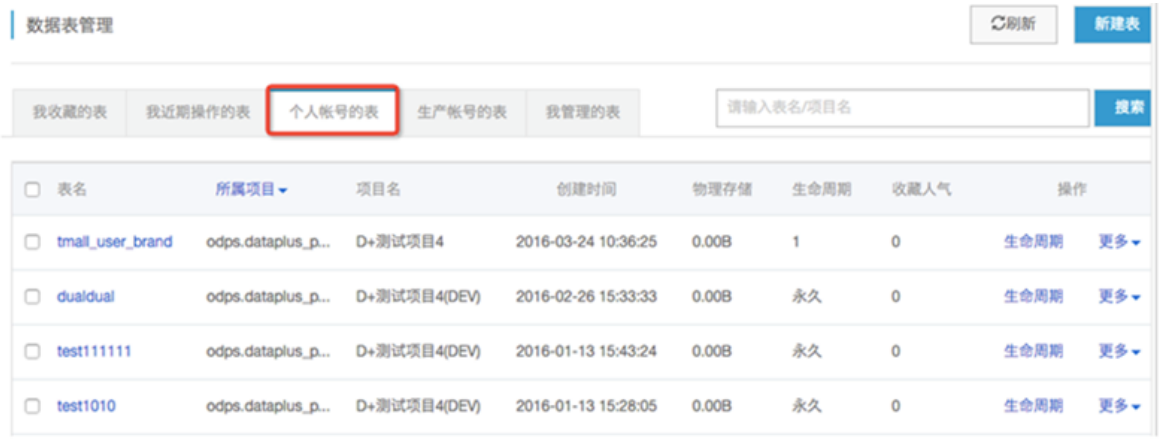
> **说明：** 系统默认当前用户具有所在项目空间中所有表/自定义函数/资源文件的读权限，无权访问的项目空间（跨项目空间）中的表/自定义函数/资源文件，需要通过**申请数据权限**来获取读权限。

27.5.6 管理数据表

数据表管理提供**我收藏的表**、**我近期操作的表**、**个人账号的表**、**生产账号的表**、**我管理的表**五大部分，可满足大部分开发者的数据管理需求，主要功能阐述如下：

- **我收藏的表**：即当前用户收藏的所有数据表。
- **我近期操作的表**：即当前用户针对表的相关操作，包括新建、收藏、取消收藏、修改owner、表结构修改、设置生命周期、设置表是否可见。
- **个人账号的表**：即当前用户在组织内所创建的数据表列表，即owner是当前用户的表。

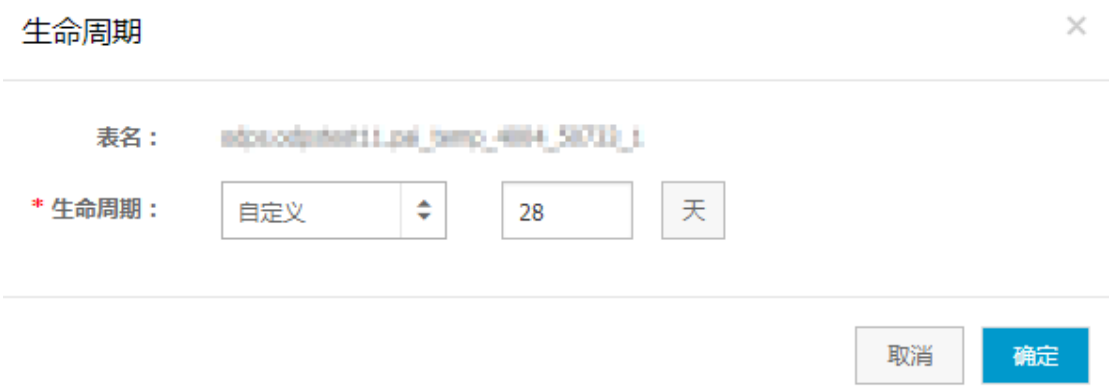
图 911: 我拥有的表



显示表名、所属项目（MaxCompute Project）、Base项目、创建时间、物理存储量、生命周期、收藏人气（当前表被收藏的人数）、最近30天浏览人数等信息。支持通过表名称模糊匹配查找，同时也可通过所属项目筛选查找表。

- **操作**：**生命周期**和**更多**。
  - **生命周期**，单击可修改表的生命周期。

图 912: 生命周期修改



- **更多**，包括**表管理**、**隐藏**和**删除**。
  - **表管理**即对表结构修改，除表名、中文名、MaxCompute项目不能修改外，其他信息都可修改。
  - **隐藏**即隐藏该表不被其他项目成员可见。
  - **删除**即删除该表结构信息。删除操作将删除该表结果信息及表中存储的数据。**需谨慎操作**。

- **生产账号的表**：即表的owner为MaxCompute访问身份配置为计算引擎指定账号（即生产账号）。
- **我管理的表**：若当前用户为项目管理员，将在此页面显示其所管理的项目空间中的所有数据表，管理员可在此修改表负责人。

27.5.6.1 新建表

通常情况下，开发者会通过新建表来存储数据同步、数据加工的结果数据，阿里云大数据平台提供多种方式来新建数据表，包括**可视化建表**、**DDL 建表**和**新建脚本文件建表**三种途径。

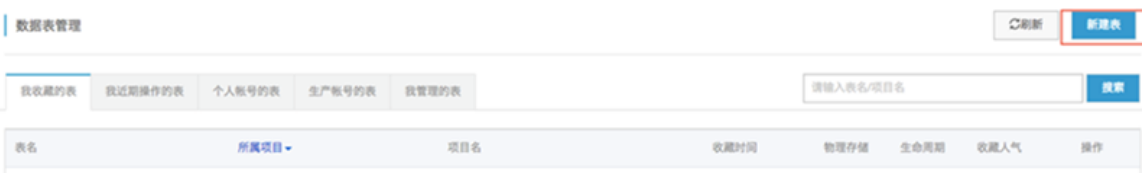
- 可视化建表

  a) 以开发者身份登录阿里云大数据开发平台。

  b) 单击顶部菜单栏中的**数据管理**，导航至**数据表管理**。

  c) 单击右上角的**新建表**。

图 913: 数据表管理界面



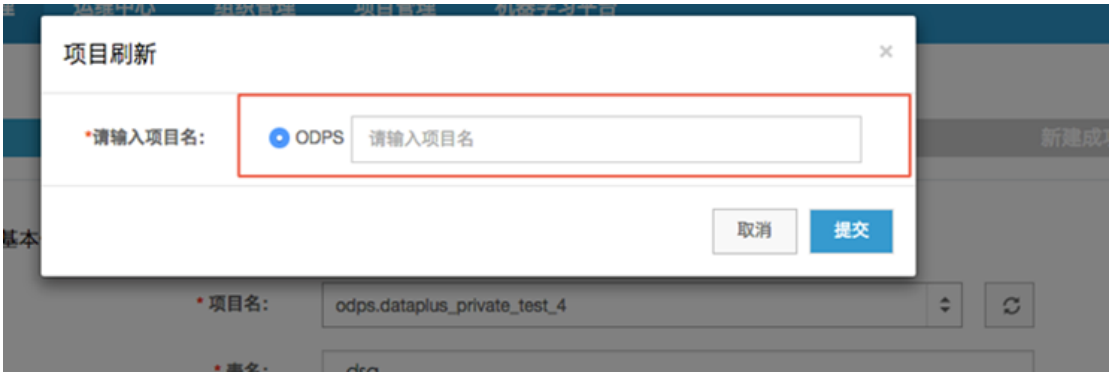
d) 在**新建表**页面中，填写基础信息。

图 914: 新建表-基础信息



可视化建表**基础信息**界面中的相关配置项，具体说明如下：

- 表名：以字母、数字、下划线组成。
- 别名：表的中文名称。
- 项目名：列表中显示当前用户已加入的MaxCompute（原 ODPS）项目空间。如果发现项目不在这个列表中，单击旁边的刷新按钮，输入需要刷行的项目名字，之后单击提交。



- 所属类目：当前表所处的类目，最多支持四级，已有类目导航配置请参见类目导航配置。
- 描述：当前表的简要说明。
- 生命周期：即MaxCompute（原 ODPS）的生命周期功能。填写一个数字表示天数，那么该表（或分区）超过一定天数未更新的数据会被清除。支持1天、7天、32天、永久、自定义四种选项。

e) 单击**下一步**。

f) 在**新建表**页面中填写**字段和分区信息**的各配置项。

- 添加字段信息设置。
- 设置分区。

图 915: 新建表-字段和分区信息



可视化建表字段和分区信息界面中的相关配置项，具体说明如下：

- 字段英文名：字段英文名，由字母、数字、下划线组成。
- 中文名：字段中文简称。
- 字段类型：MaxCompute（原ODPS）数据类型（string、bigint、double、datetime、boolean）。
- 描述：字段的详细描述。
- 操作：上移、下移、保存、编辑、删除。
- 是否设置分区：若选择设置分区，需配置分区键的具体信息，支持string和bigint类型。

g) 单击**提交**。

新建表提交成功后，系统将自动跳转返回**我拥有的表**界面。

- DDL建表

a) 以开发者身份登录阿里云大数据开发平台。

b) 单击顶部菜单栏中的**数据管理**，导航至**数据表管理**。

c) 单击**新建表**。

d) 单击新建表页面右上角的**DDL建表** 。

e) 填写MaxCompute SQL建表语句。

图 916: DDL 建表




```
/*****示例代码*****/ :
create table if not exists table2
(
 id string comment '用户ID',
 name string comment '用户名称'
) partitioned by(dt string)
LIFECYCLE 7;
```

f) 单击**提交**。

图 917: 自动填充基础信息相关配置项



基础信息页面的表名和生命周期配置项将自动填充。

图 918: 自动填充字段和分区信息相关配置项



g) 补充新建表基础信息页面中的配置项，单击**下一步**。

图 919: 补充基础信息相关配置项



h) 单击**提交**。

说明: 通过可视化建表与DDL建表，需用户在**用户信息**中绑定AK信息，否则建表时将报错。

- 新建脚本文件建表

  a) 以开发角色身份登录阿里云大数据开发平台。

  b) 单击顶部菜单栏中的**数据开发**，导航至**开发面板** > **新建**。

  c) 单击**新建脚本文件**。

  d) 在**新建脚本文件**弹出框中，填写各配置项。

图 920: 新建脚本文件弹出框



e) 单击**提交**。

f) 在 MaxCompute（原 ODPS）代码编辑器中编写建表 DDL语句。

图 921: 在脚本文件中建表



g）单击**运行**。

图 922: 执行 DDL 脚本



说明: 通过方式一、方式二新建表默认责任人为当前登录用户，而通过方式三（新建脚本文件建表）新建表的责任人将为MaxCompute（**原**ODPS）**访问身份**所对应的用户。查看当前项目空间的MaxCompute（原ODPS）访问身份信息，可导航**项目管理** > **项目属性** > **计算引擎配置**进行查看。

27.5.6.2 收藏表

数据表管理模块提供收藏表功能。

操作步骤

1. 进入表详情页面。
2. 单击**收藏**。

图 923: 收藏表



收藏后，可在**我收藏的表**页面中取消收藏。

27.5.6.3 修改生命周期

您可在**个人账号的表、我近期操作的表、生产账号的表**和**我管理的表**（项目管理员权限）中修改表的生命周期。

场景一：**修改我创建的表的生命周期**

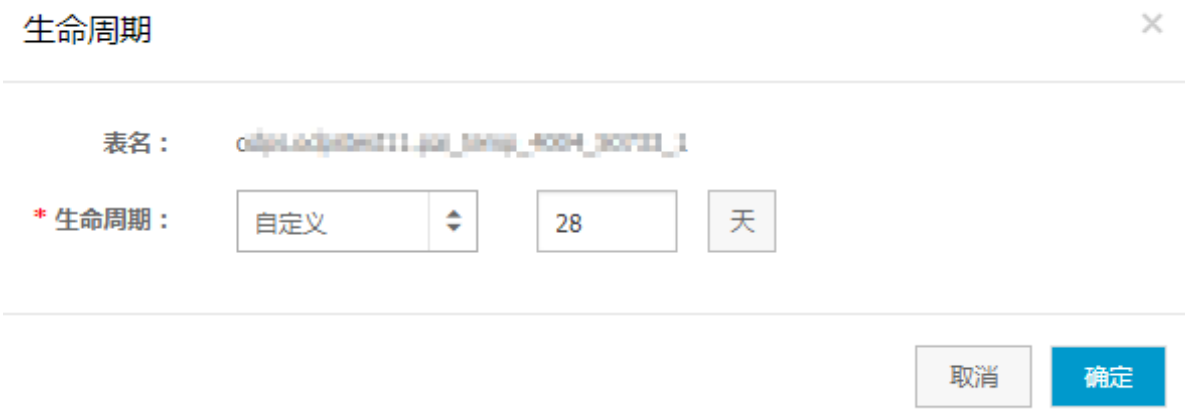
1. 以开发者身份登录阿里云大数据平台。
2. 单击顶部菜单栏中的**数据管理**，导航至**数据表管理 > 个人账号的表**。
3. 单击对应表后的**生命周期**。

图 924: 我拥有的表-修改生命周期



4. 在**修改生命周期**弹出框中，修改生命周期。

图 925: 修改生命周期弹出框



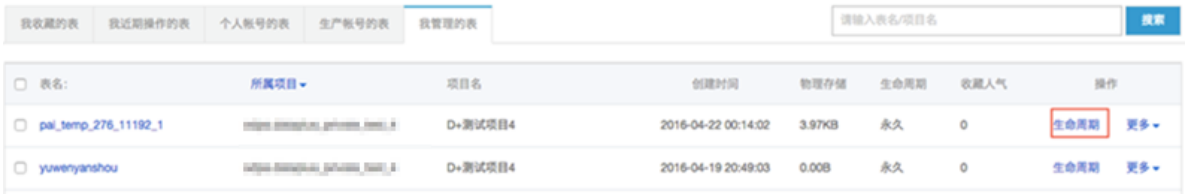
5. 单击**确定**。

> **说明:** 若想修改我近期操作的表、生产账号的表的生命周期，操作同上。

场景二：**修改我管理的表的生命周期**

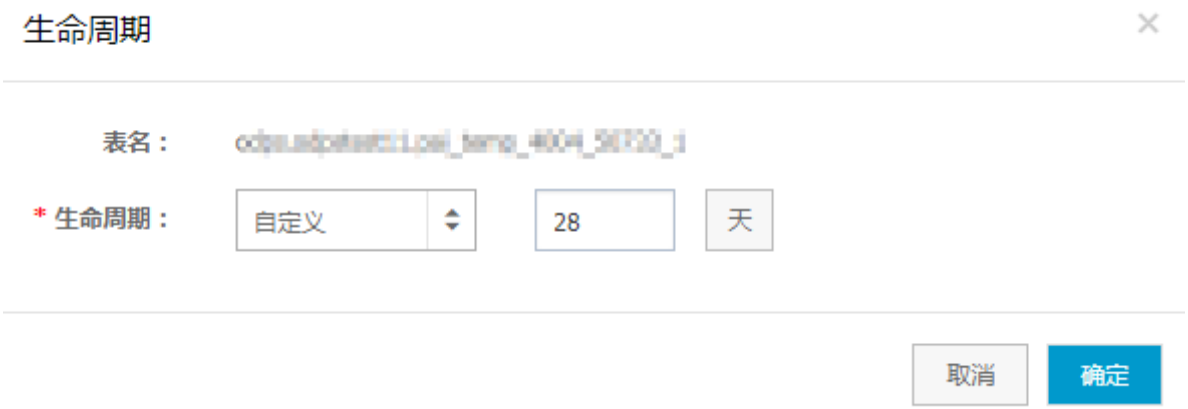
1. 以管理员身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据管理**，导航至**数据表管理** > **我管理的表**。
3. 单击对应表后的**生命周期**。

图 926: 我管理的表-修改生命周期



4. 在**修改生命周期**弹出框中，修改生命周期。

图 927: 修改生命周期弹出框



5. 单击**确定**。

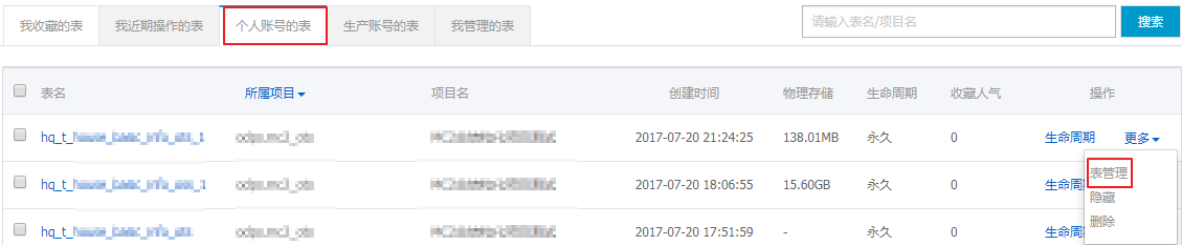
27.5.6.4 修改表结构

您可在**个人账号的表**和**我管理的表**（项目管理员权限）中修改表结构。

场景一：**修改个人账号的表**

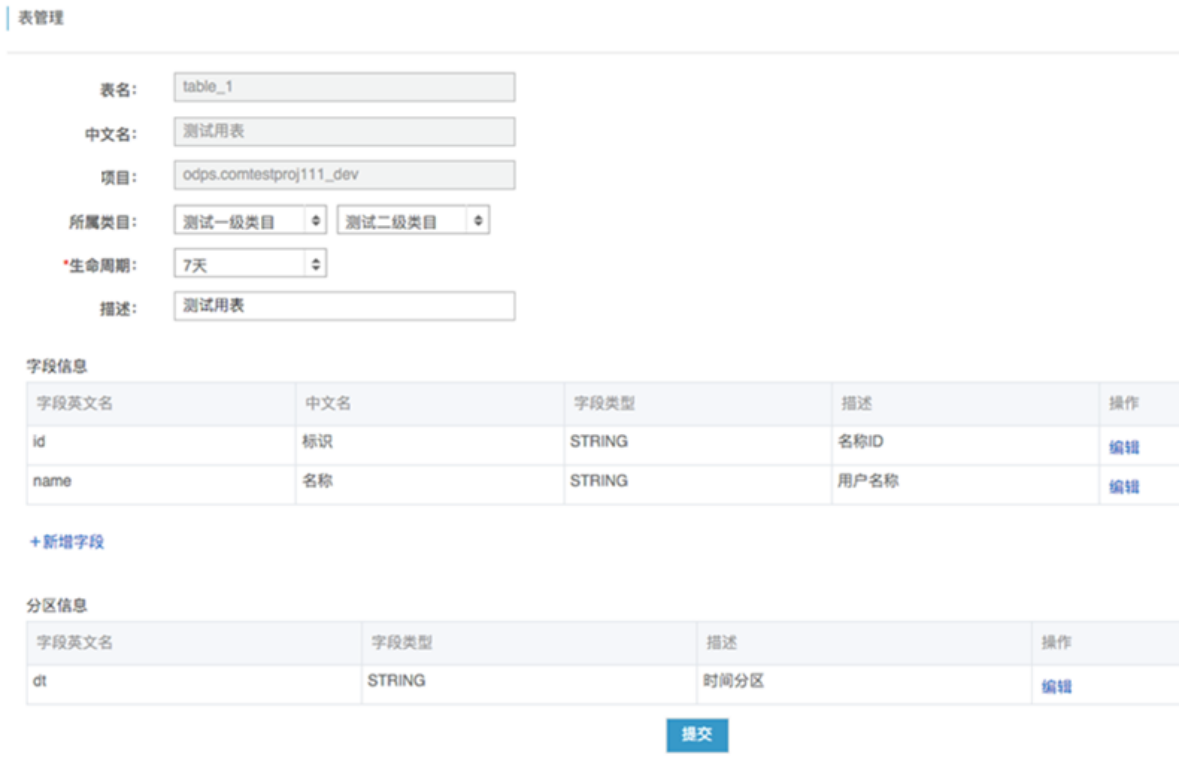
1. 以开发者身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据管理**，导航至**数据表管理** > **个人账号的表**。
3. 单击**更多** > **表管理**。

图 928: 个人账号的表



4. 在**表管理**页面中，修改相关信息。

图 929: 修改个人账号表结构

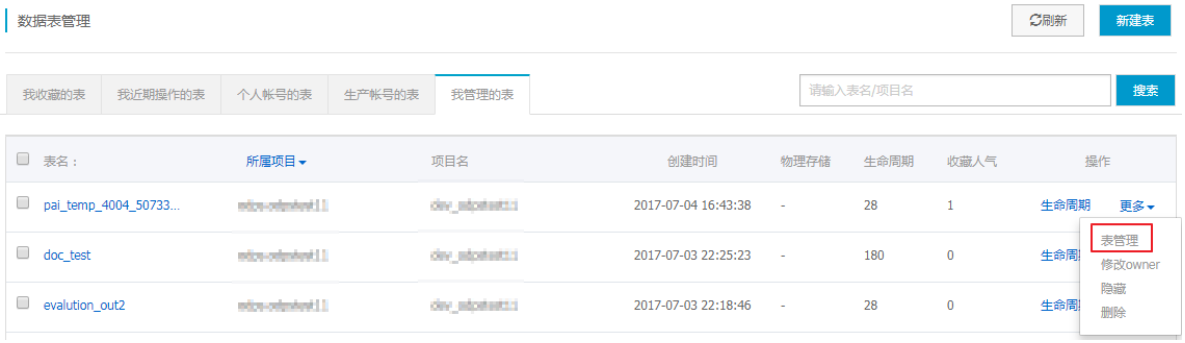


5. 单击**提交**。

场景二：修改我管理的表

1. 以管理员身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据管理**，导航至**数据表管理 > 我管理的表**。
3. 单击**更多 > 表管理**。

图 930: 个人账号的表



4. 在**表管理**页面中，修改相关信息。
5. 单击**提交**。

27.5.6.5 隐藏表

表负责人或项目管理员可对表进行隐藏，让其他成员不可见。

场景一：隐藏个人账号的表

1. 以开发者身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据管理**，导航至**数据表管理** > **个人账号的表**。
3. 单击表操作栏中的**更多** > **隐藏**。

图 931: 隐藏个人账号的表



被隐藏的表名后将有隐藏标识：

图 932: 表隐藏标识



说明: 若想隐藏我近期操作的表、生产账号的表，操作同上。

场景二：隐藏我管理的表

1. 以管理员身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据管理**，导航至**数据表管理** > **我管理的表**。
3. 单击**更多** > **隐藏**。

图 933: 隐藏我管理的表



被隐藏的表名后将有隐藏标识：

图 934: 表隐藏标识



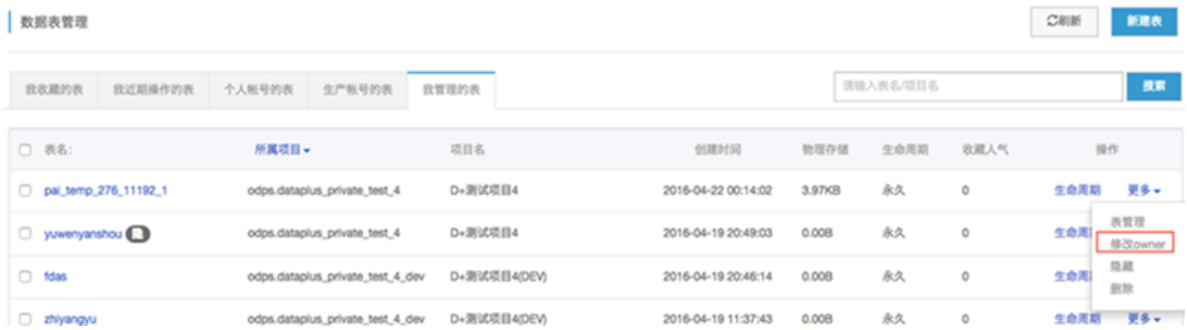
27.5.6.6 修改表负责人

项目管理员可修改表负责人。

操作步骤

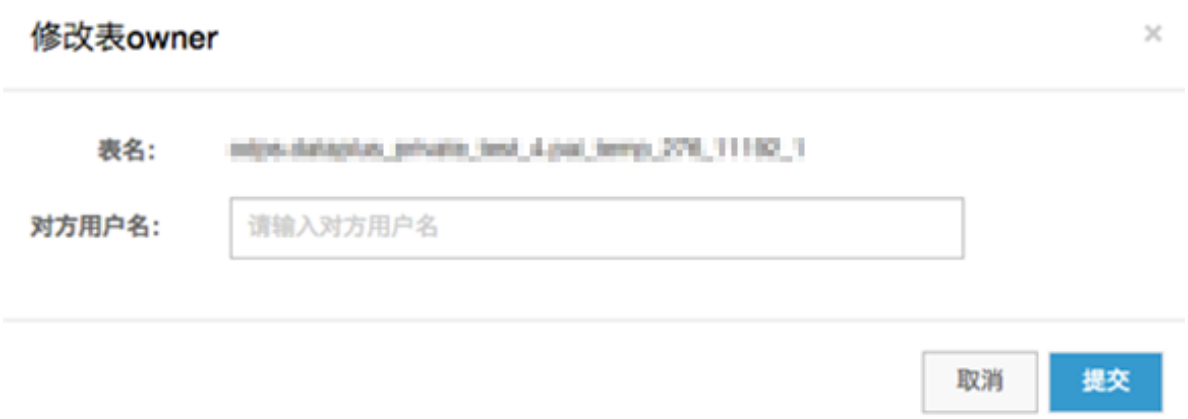
1. 以管理员身份登录阿里云大数据平台。
2. 单击顶部菜单栏中的**数据管理**，导航至**数据表管理 > 我管理的表**。
3. 单击**更多 > 修改表owner**。

图 935: 我管理的表



4. 在**修改表owner**弹出框中，输入阿里云大数据开发平台账号。

图 936: 修改表 owner



修改表owner弹出中配置项的具体说明：

- 表名：正在操作的表的名字。
- 对方用户名：阿里云大数据平台登录账号。

5. 单击**提交**。

27.5.6.7 删除表

您可以在**个人账号的表**页面中删除自己所创建的表，若为管理员权限，则可在**我管理的表**中删除所管理的表。

场景一：**删除个人账号的表**

1. 以开发者身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据管理**，导航至**数据表管理** > **个人账号的表**。
3. 单击表操作栏中的**更多** > **删除**。

图 937: 删除个人账号的表



4. 单击**确定**。

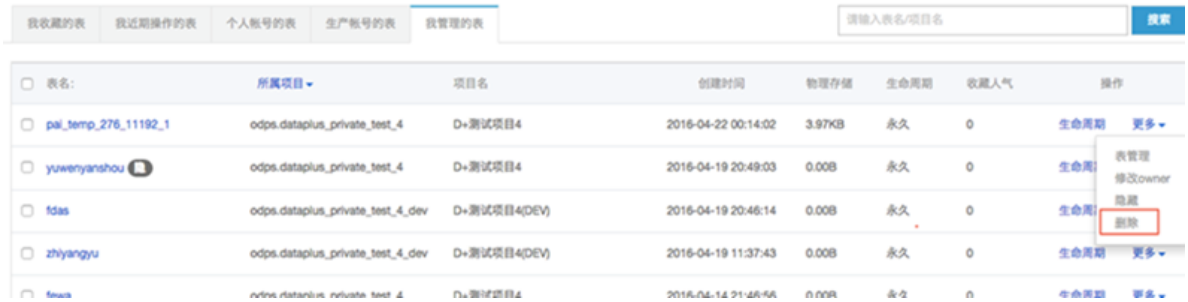
图 938: 确认删除



场景二：**删除我管理的表**

1. 以管理员身份登录阿里云大数据开发平台。
2. 单击顶部菜单栏中的**数据管理**，导航至**数据表管理** > **我管理的表**。
3. 单击**更多** > **删除**。

图 939: 删除我管理的表



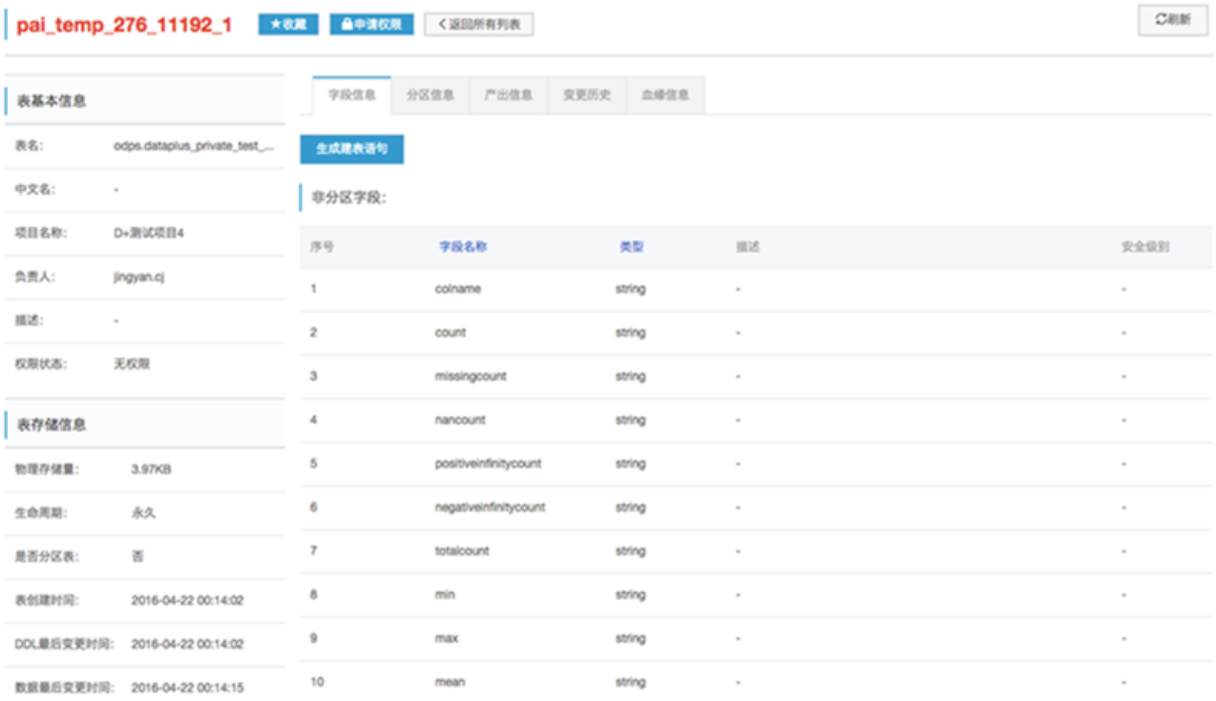
4. 单击**确定**。

说明: 若想删除生产账号的表、我近期操作的表，操作同上。

27.5.7 查看表详情

在数据表管理模块中任意页面，点击数据表名称可跳转至**表详情**页面，包括表的基本信息、存储信息、分区信息、产出信息、变更历史和血缘信息等。

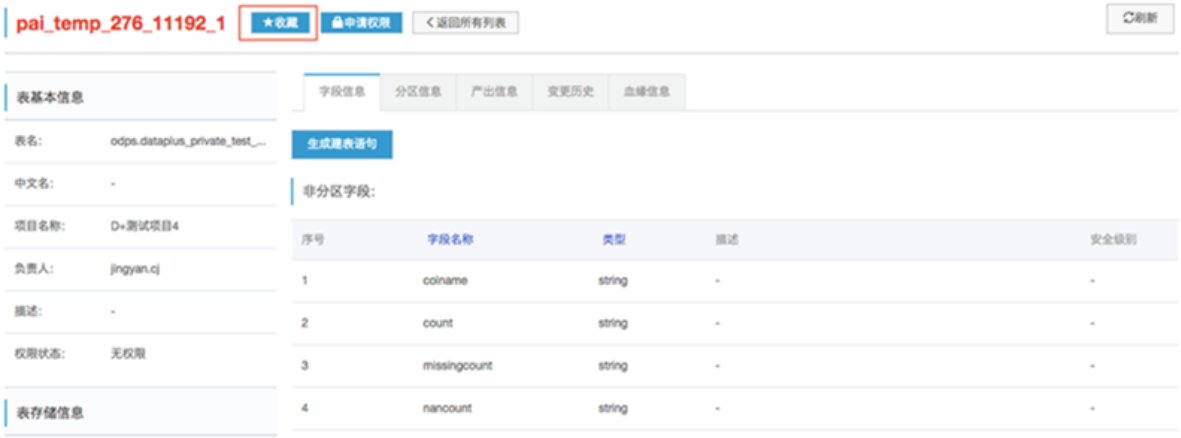
图 940: 表详情



表详情页面中的功能点具体说明如下：

- 收藏表：单击顶部收藏按钮。

图 941: 收藏该表



- 返回所有列表：单击**返回所有列表**，可返回所有数据表列表。
- 查看表基本信息：包括表名、中文名、阿里云大数据开发平台项目名称、负责人名称、描述、权限状态（为离线加工数据，滞后一天），如下所示：

图 942: 表基本信息



- 查看表存储信息：包括物理存储量、今日新增量、今日净增量、生命周期、是否分区表、表创建时间、DDL最后变更时间、最后数据变更时间。

图 943: 存储信息



- 查看字段信息：包括字段名称、类型、是否分区键、描述，也可单击**生成建表语句**按钮生成该表的DDL语句。

图 944: 字段信息



- 查看分区信息：包括分区名称、创建时间、存储量、记录数。

图 945: 分区信息



- 查看产出信息：主要包括表分区数据产出执行时长（秒）、结束时间等。

- 查看变更历史：包括表、分区粒度的历史变更信息。

图 946: 变更历史



- 查看血缘信息：该表数据流经MaxCompute（原ODPS）的血缘信息，支持字段级血缘。

图 947: 血缘信息



图 948: 字段级血缘



27.5.8 权限审批与收回

数据权限申请中的表、资源和自定义函数都将在**数据管理** > **权限管理** > **待我审批**页面中展现，项目管理员可在此进行申请记录审批**通过**或**驳回**操作。

审批通过的表、资源和自定义函数项目管理员都可在**数据管理** > **权限管理** > **权限收回**页面中进行权限收回操作。

说明: 详情请参见《阿里云大数据开发平台管理员手册》。

27.5.9 配置类目导航

您（需组织管理员权限）可以在**类目导航配置**页面中进行新建表所属类目的配置。

操作步骤

1. 登录阿里云大数据开发平台。

2. 单击顶部菜单栏中的**数据管理**，导航至**管理配置**。
3. 单击表所属类目设置后添加一级类目。

图 949: 新建一级类目



4. 单击一级类目后添加所属二级类目。

图 950: 新建二级类目



以此类推，最高可支持四级类目的设置。其中表示编辑该类目名称，表示删除该类目。

类目配置完成后，即可在新建表页面中选择已配置类目，如图 *951:* 类目导航配置所示：

图 951: 类目导航配置

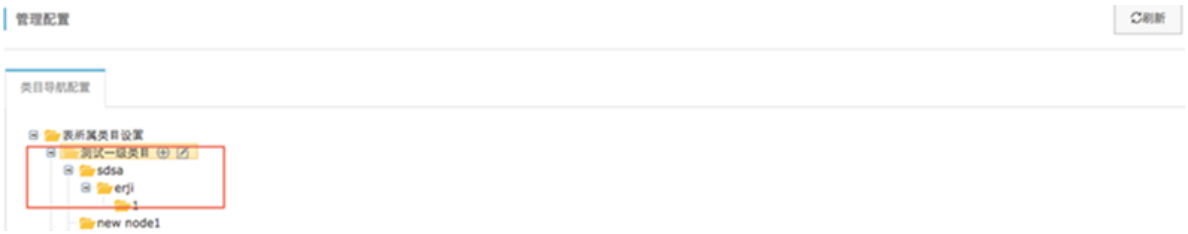


图 952: 新建表-所属类目



27.6 运维中心

开发者在阿里云大数据开发平台**运维中心**模块中通常情况下需要进行工作流任务和节点测试。

工作流任务测试

节点测试

27.6.1 概述

运维中心是日常运维的主要工具，可对已提交的工作流及其节点任务进行管理与维护。分为运维概览、任务管理、任务监控三个模块。

- **概览**：主要对平台的全局任务进行查看与管理，包括任务完成情况、任务运行情况、任务执行时长排行、调度任务数量趋势、近一月出错排行以及当前项目空间中任务类型分布。
- **任务管理**：任务管理包括：工作流管理、工作流定义、工作流实例、节点定义和节点实例五大部分。可通过工作流可视化视图及工作流列表视图两种方式来进行管理。工作流可视化视图可对节点的运行状态及上下游依赖关系等进行维护与管理，可对单个任务进行补数据、重跑等操作；列表视图则以图表形式列出任务的运行状态，可进行批量结束任务及批量重跑、批量修改任务属性、配置监控报警等操作。
- **监控报警**：主要包括报警记录和自定义提醒两部分。运维人员可在监控报警模块中查看历史报警记录，并对已自定义的报警提醒进行修改等操作。

27.6.2 使用权限说明

运维中心仅对**开发、运维、项目管理员**3种角色开放。

27.6.2.1 角色的权限点划分

组织管理员、部署和访客三种角色的用户对运维中心来说是不具备任何权限的。**项目管理员、开发**和**运维**的具体权限点划分如下表所示：

表 100: 运维中心模块权限点划分

父权限点名称	权限点名称	项目管理员	开发	运维
任务运维	定义-查看流程图	√	√	√
	定义-编辑-跳转到IDE	√	√	/
	实例-查看流程图	√	√	√
	查看flow定义-dag的上下游	√	√	√
	查看flow列表	√	√	√
	定义-查看工作流任务操作日志	√	√	√
	定义-冒烟测试	√	√	√
	定义-补数据	√	√	√
	定义-修改责任人	√	√	√
	查看flow详情	√	√	√
	查看flow实例-dag的上下游	√	√	√
	实例-暂停	√	√	√
	实例-恢复	√	√	√
	实例-杀任务	√	√	√
	实例-批量杀任务	√	/	√
	实例-列表	√	√	√
	查看执行日志	√	√	√
	实例-重跑	√	√	√
	实例-批量重跑	√	/	√
	实例-查询	√	√	√
	实例-置成功	√	√	√
节点运维	定义-修改基线	√	√	√
	定义-批量修改基线	√	/	√

父权限点名称	权限点名称	项目管理员	开发	运维
	定义-查看节点代码	√	√	√
	定义-修改责任人	√	√	√
	定义-批量修改责任人	√	/	√
	定义-修改资源组	√	√	√
	定义-批量修改资源组	√	/	√
	定义-冒烟测试	√	√	√
	定义-补数据	√	√	√
	实例-删除关系	√	/	√
	实例-暂停	√	√	√
	实例-恢复	√	√	√
	实例-终止节点实例	√	√	√
	实例-批量杀任务	√	/	√
	实例-修改优先级	√	√	√
	实例-刷新关系	√	√	√
	实例-重跑节点实例	√	√	√
	实例-批量重跑	√	/	√
	实例-置节点实例为成功	√	√	√
大屏	查看基线破线次数	√	√	√
	大屏排除某个记录	√	√	√
	查看近一月出错排行	√	√	√
	查看任务调度数量趋势	√	√	√
	查看任务完成情况	√	√	√
	查看任务运行情况	√	√	√
	查看任务执行时长排行	√	√	√
	查看任务类型分布	√	√	√
报警监控	查看通知列表	√	√	√
	关闭报警	√	√	√
	批量关闭报警	√	/	√

父权限点名称	权限点名称	项目管理员	开发	运维
	电话开关	√	√	√
	添加自定义提醒	√	√	√
	删除自定义提醒	√	√	√
	编辑自定义提醒	√	√	√
	查看自定义提醒	√	√	√
	查看全部事件列表	√	√	√
	查看事件详情	√	√	√
	查看个人事件列表	√	√	√

27.6.2.2 如果您是一个开发人员

常见应用场景

试运行及管理您在数据开发面板中编辑好并提交的工作流。

仅限于对开发环境下的项目空间的工作流/节点任务进行相关操作，且不可通过授权方式进入生产环境。

开发环境下的操作权限

- 可进行单个工作流/节点测试、补数据、暂停、重跑任务等操作。
- 还可进行批量修改工作流/节点属性、批量结束任务及批量重跑、配置监控报警等操作。

27.6.2.3 如果您是一个部署人员

很抱歉，运维中心仅对开发、运维、项目管理员3种角色开放。

您的常见应用场景是**打包发布文件包**，相关内容请参见：跨项目发布。

27.6.2.4 如果您是一个运维人员

常见应用场景

- 发布管理—打包发布文件包，相关内容请参见：跨项目发布；
- 处理任务异常

  可以对开发环境、生产环境进行相关操作，但需要通过管理员对开发/生产环境的项目授权后方可进行。

运维任务包括：单个工作流/节点测试、补数据、暂停、重跑任务等操作。同时，还可批量修改工作流/节点属性、批量结束任务及批量重跑、配置监控报警等干预性的操作。

27.6.2.5 如果您是一个项目管理员

您在运维中心模块中拥有与项目管理员同等的操作权限。

27.6.3 任务管理

任务管理模块分为视图和列表两种展现方式，任务管理视图展现的是被提交至调度系统中的任务依赖关系的DAG 图（有向无环图），任务管理列表展现的是被提交至调度系统中的任务的详细信息，包括：调度属性、调度资源组、任务属性等，可以在任务管理页面中对当前任务进行测试、操作日志查看、监控告警配置等操作。

测试

测试是指将选中的任务按您配置的起调时间进行一次模拟运行的动作。运行所需要的数据按照您选择的业务日期进行判断，使用**T-1**的数据作为模拟运行数据。

任务测试：将当前选中的任务（工作流任务/单节点任务）全部模拟测试一遍。

节点测试：只针对当前选中的某一个工作流任务中的节点，进行模拟测试。

说明: 在调度系统中的冒烟测试会直接影响到该任务中涉及到的相关数据表、文件等，在使用测试功能时请多加注意。

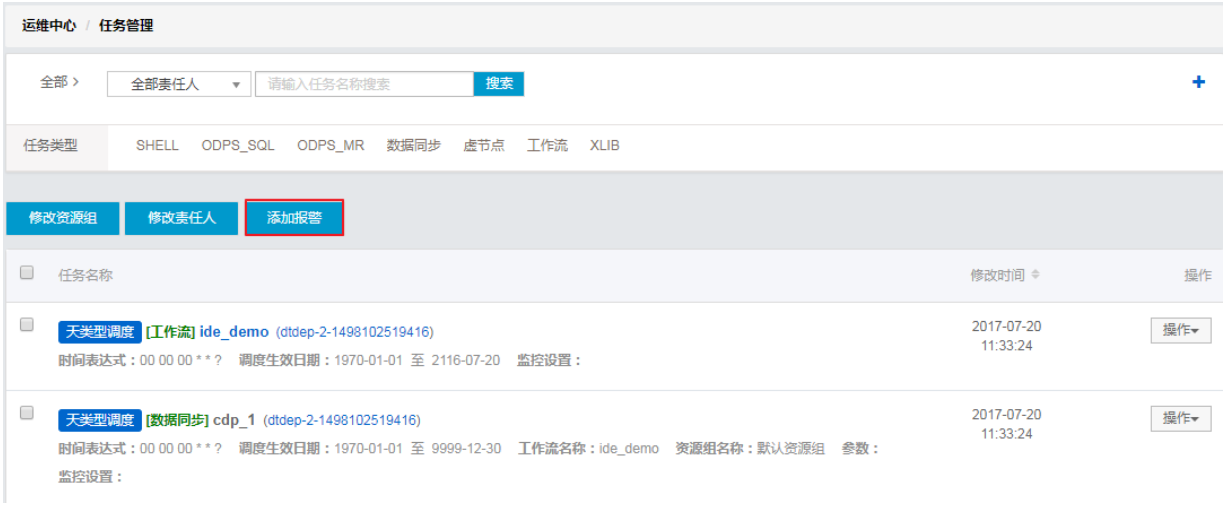
补数据

补数据通常用于处理任务/节点异常情况，即重新运行以前某个时段数据的任务/节点。将该时间段内的数据补进来重新通过任务运行一遍，以保证对历史数据的最佳利用。补数据时可以选择只对当前选中任务/节点进行补数据（不会作用在下游任务/节点上），也可以选择对当前选中任务/节点以及下游的任务/节点进行补数据。

添加告警

监控报警配置在**任务管理** > **任务管理列表**页面，进入后界面如下，

图 953: 添加报警



可以对单个或批量节点任务添加监控报警，选中需要配置报警的节点，单击**添加报警**，如图 *953: 添加报警*所示：

图 954: 添加报警



可对**出错**、**未完成**和**完成**3 种状态配置报警提醒方式，还可以指定人作为报警接收人。

修改调度资源组

调度资源组指大数据开发平台执行任务时所需要涉及到的机器资源，系统支持将自购ECS部署成为任务执行的调度资源组。

> **说明：** 新建调度资源并为其配置服务器作为专用的调度资源来使用需要组织管理员的角色操作，相关信息请参见：《阿里云大数据开发平台管理员手册》。

选中的任务可以选择修改调度资源组，设置好后，该任务将不会再系统默认的资源组中执行，会被分发至自购的ECS中执行。

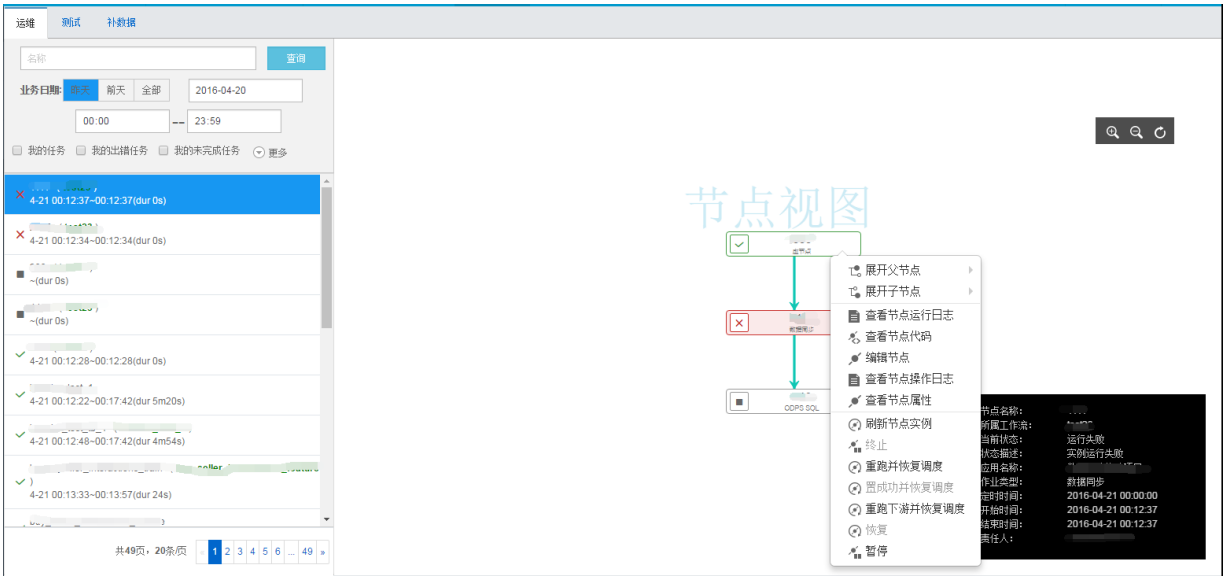
27.6.4 任务运维

任务运维模块分为视图和列表两种展现方式，任务管理视图展现的是执行中（或执行完）的任务依赖关系的DAG图（有向无环图），任务运维列表展现的是执行中（或执行完）的任务详细信息，包括调度状态、执行日志、调度属性等。可以在任务管理页面中对当前任务进行重跑、执行日志查看、结束进程等操作。

27.6.4.1 周期任务运维

实例是指当工作流任务任务达到启用调度所配置的周期性运行时间时，被调度起来的一个实例运行状态的快照。每被调度一次，则生成一个实例（分为周期性实例、测试实例、补数据实例），可对已调度起的实例任务进行日常的运维管理，如对任务进行终止、重跑、修复等操作。如图 955: 节点视图运维所示：

图 955: 节点视图运维



系统每天22：30会根据周期节点任务定义好的时间自动生成一个快照，在22：30之后新加的节点和修改的依赖关系，要在下一天的生成工作流上才能体现。

表 101: 节点状态

序号	状态类型	状态标识
1	运行成功状态	
2	未运行状态	
3	运行失败状态	
4	正在运行状态	
5	终止运行手动	
6	等待状态	

右键操作特别说明：

- **终止**：可以kill等待时间、等待资源、运行状态的任务，并将任务置为失败状态。

  前置条件：只能终止等待时间、等待资源、运行中状态的任务。
- **重跑并恢复调度**：可以重跑某任务，任务执行成功后可以触发下游未运行状态任务的调度。常用于处理出错节点和漏跑节点。

  前置条件：只能重跑未运行、成功、失败状态的任务。
- **置成功并恢复调度**：将当前节点状态改为成功，并运行下游未运行状态的任务。常用于处理出错节点。

  前置条件：只能失败状态的任务能被置成功。
- **重跑下游并恢复调度**：可以重跑某任务及其下游任务，需要用户自定义勾选，勾选的任务将被重跑，任务执行成功后可以触发下游未运行状态任务的调度。常用于处理数据修复。

  前置条件：只能勾选未运行、完成、失败状态的任务，如果勾选了其他状态的任务，页面会提示“已选节点中包含不符合运行条件的节点”，并禁止提交运行。
- **暂停**：暂停节点当前周期的调度。

说明: 此操作只暂停当前周期，下一周期仍会正常调度，如果要永久暂停调度，需要到流程设计器中，找到**节点** > **调度** > **时间属性** > **节点状态**，勾选**暂停调度**，再保存、发布方可生效。

前置条件：无。

- **恢复**：相对于上述的“暂停“操作。

27.6.4.2 查看执行日志

单节点任务实例、工作流任务节点右击展开的下拉菜单中可以选择查看当前选中对象的执行日志。可以查看当前选中对象在执行过程中的相关日志信息。如图 *956:* 节点视图运维所示：

图 956: 节点视图运维

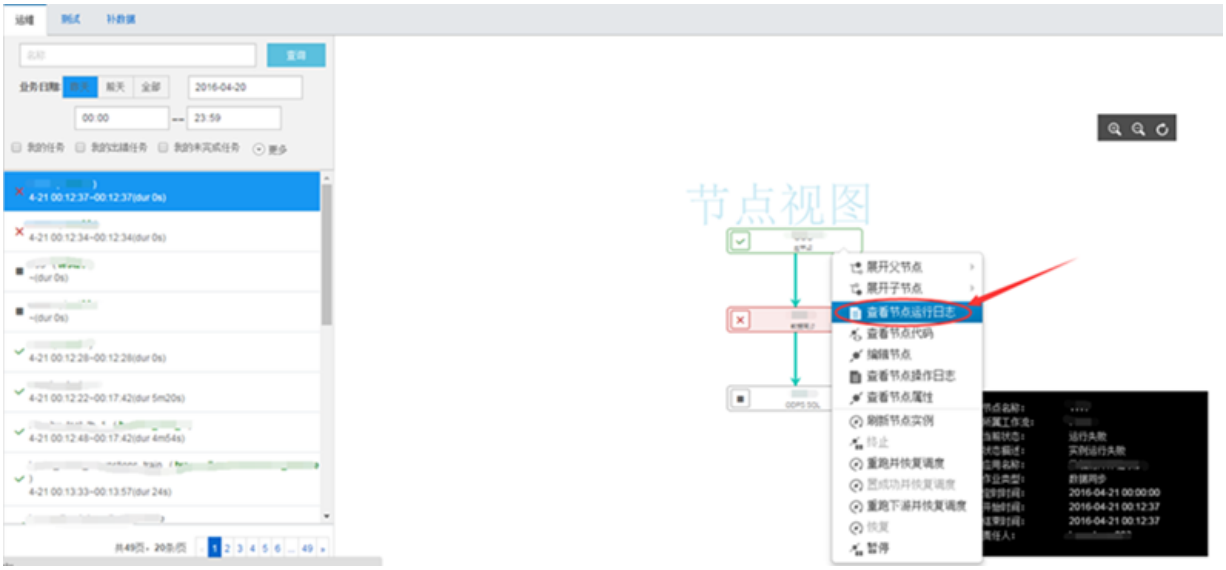
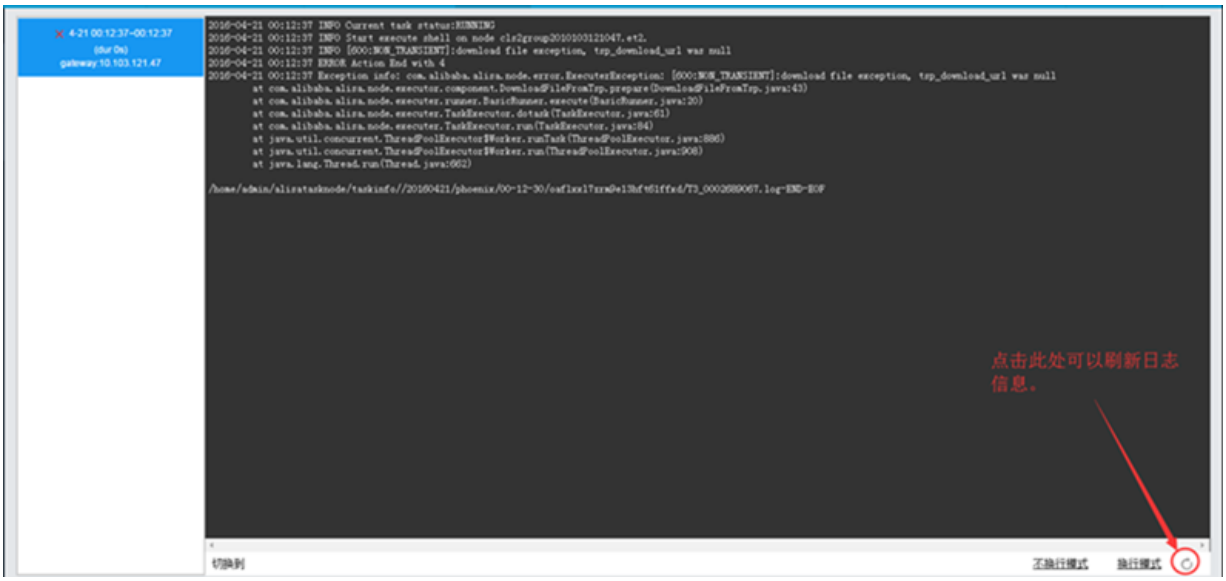


图 957: 节点视图运维



27.6.5 监控报警

监控报警模块用于查看已经设置的报警任务列表，以及对相应的报警规则进行修改和报警记录的查阅。

27.7 组织管理

组织管理模块仅针对组织管理员角色用户开放，组织管理员常见的操作包括项目管理、成员管理、调度资源和计算引擎。

项目管理

组织管理员可以进行新建项目空间、激活/禁用项目空间、编辑项目空间等操作。在新建项目空间操作中，系统提供6种项目模块来快速创建项目，包括单个开发项目、单个生产项目、开发+生产项目、开发+测试+生产项目、开发+测试+预发+生产项目和其他类型项目。

成员管理

新成员进入某项目空间中进行数据开发，DTcenter会将该成员加入对应的组织中，具体操作请参见DTcenter使用文档，然后项目管理员将其拉入某个具体的项目空间并赋予相关角色。

调度资源

组织管理员可新建调度资源并为其配置服务器作为专用的调度资源来使用。

计算引擎

组织管理员可添加已有MaxCompute（原 ODPS）项目作为计算引擎，也可新建MaxCompute（原ODPS）项目作为计算引擎。

说明: 组织管理模块的详细操作请参见《大数据开发套件管理员手册》。

27.8 项目管理

项目管理模块主要是针对项目管理员开放的功能模块，包括新增数据源、安全策略配置、成员管理等操作。开发、运维和部署角色的用户只具备查看项目基本属性和查看项目数据源配置的信息。

说明: 项目管理模块的详细操作请参见《大数据开发套件管理员手册》。

28 分析型数据库

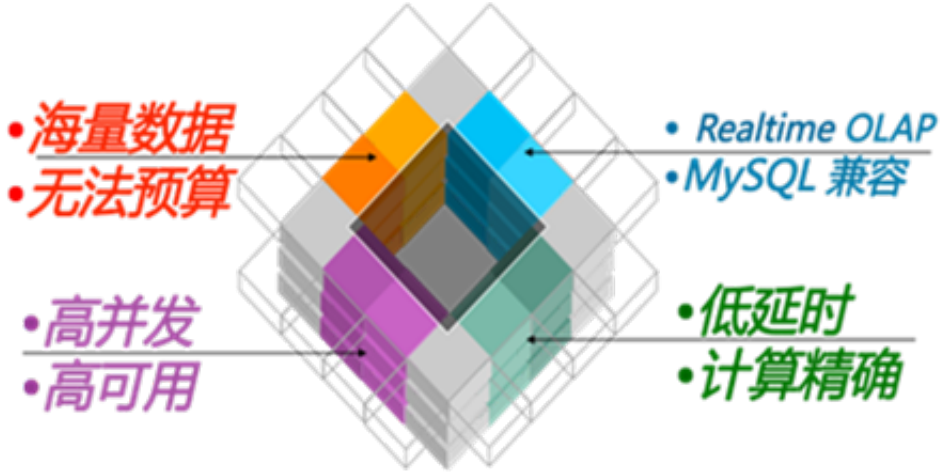
28.1 产品概述

根据IDC 2013年发布的数字宇宙研究报告（Digital Universe）显示，在接下来的8年中，我们所产生的数据量将超过40ZB（泽字节）。作为大数据特征中最重要的 Volume（容量）、Velocity（数据生产速度）的两个原始特征都在发生急剧变化，使得数据处理从业务系统的一部分演变得愈发独立，企业待加速数据分析和挖掘过程，并由报表展现为主到强调数据洞察转型，让数据直接快速产生价值（Value）。在业务系统中，我们通常使用的是OLTP （OnLine Transaction Processing，联机事务处理）系统，如MySQL、Microsoft SQL Server 等关系数据库系统。这些关系数据库系统擅长事务处理，在数据操作中保持着严格的一致性和原子性，能够很好支持频繁的数据插入和修改，但是，一旦待进行查询或计算的数据量过大，达到数千万甚至数十亿条，或待进行的计算非常复杂的情况下，OLTP类数据库系统便力不从心了。

图 958: 数据库对比



图 959: 优势对比



分析型数据库（AnalyticDB，原ADS）和主流数据系统进行对比时，我们等待OLAP（On-Line Analytical Processing，联机分析处理）系统来进行处理。从广义上，OLAP系统是针对OLTP系统而言，并不特别关心对数据进行输入、修改等事务性处理，而是关心对已有大量数据进行多维度的、复杂的分析的一类数据处理系统。而在具体的产品中，我们通常将OLAP系统分为MOLAP、ROLAP和HOLAP三类。

多维OLAP（Multi-Dimensional OLAP，简称MOLAP），是预先根据数据待分析的维度进行建模，在数据的物理存储层面以“立方体”（Cube）的结构进行存储，具有查询速度快等优点，但是数据必须预先建模，无法依据使用者的意愿进行即时灵活的修改。而关系型OLAP（Relational OLAP，简称 ROLAP），则使用类似关系数据库的模型进行数据存储，通过类似SQL等语言进行查询和计算，优点是数据查询计算自由，可以灵活地根据使用者的要求进行分析，但是缺点是在海量数据的情况下分析计算缓慢。至于HOLAP，则是MOLAP和ROLAP的混合模式。

而分析型数据库，则是一套RT-OLAP（Realtime OLAP，实时OLAP）系统。在数据存储模型上，采用自由灵活的关系模型存储，可以使用SQL进行自由灵活的计算分析，无需预先建模，而利用分布式计算技术，分析型数据库可以在处理百亿条甚至更多量级的数据上达到甚至超越MOLAP类系统的处理性能，真正实现百亿数据毫秒级计算。

分析型数据库让海量数据和实时与自由的计算可以兼得，实现了速度驱动的大数据商业变革。一方面，分析型数据库拥有快速处理迁移级别海量数据的能力，使得数据分析中使用的数据可以不再是抽样的，而是业务系统中产生的全量数据，使得数据分析的结果具有最大的代表性。

另外，AnalyticDB采用分布式计算技术，拥有强大的实时计算能力，通常可以在数百毫秒内完成百亿级的数据计算，使得使用者可以根据自己的想法在海量数据中自由的进行探索，而不是根据预先设定好的逻辑查看已有的数据报表。

更重要的是，由于分析型数据库能够支撑较高并发查询量，并且通过动态的多副本数据存储计算技术来保证较高的系统可用性，所以能够直接作为面向最终用户（End User）的产品（包括互联网产品和企业内部的分析产品）的后端系统。如淘宝数据魔方、淘宝指数、快的打车、阿里妈妈达摩盘（DMP）、淘宝美食频道等拥有数十万至上千万最终用户的互联网业务系统中，都使用了分析型数据库。

大数据领域，愈发向“4M”趋势发展：

图 960: 4M趋势



分析型数据库作为海量数据下的实时计算系统，给使用者带来极速自由的大数据在线分析计算体验，最终期望为大数据行业带来巨大的变革。

28.2 数据库、表组和表介绍

在传统的关系数据库系统中，表直接隶属于某个数据库。而在分析型数据库中，为方便对数据的关联的管理，以及进行资源分配，引入了表组的概念。在分析型数据库中，数据库、表组、表的概念各有其独特的地方。

数据库

在分析型数据库中，数据库是用户所关心的最大单元，也是用户和分析型数据库系统管理员的管理职权的分界点。分析型数据库系统管理员最小可管理数据库粒度的参数，而无法未经用户授权来查看和管理数据库内部的 结构和信息。而数据库级别的参数对于用户来讲大多只能查看而不能修改。在分析型数据库中，一个数据库对应着一个用于访问的域名和端口号，并且有且只有一个onwer为

数据库的创建者。另外对用户的宏观的资源配置是以数据库为粒度进行的，因此创建数据库时用户待输入业务预估的QPS、数据量、Query类型等信息用于智能的判断初始的资源分配。

表组

表组是一系列可发生关联的数据表的集合。分析型数据库中表组分为两类：普通表组和维度表组。其中维度表组用于存放维度表（特征上是一种数据量较小但是待和任何表进行关联的表），目前有且仅有一个，并且在分析型数据库数据库建立时会自动创建，用户不可修改和删除。而普通表组有如下特征：

- 表组是数据物理分配的最小单元，数据的物理分布情况通常无需用户关心，但是因此数据的副本数（指数据在分析型数据库中同时存在的份数）必须在表组上进行设定，一个表组所有表副本数一致。
- 一个表组内的表才能够进行快速的hash join。0.9 版本之前分析型数据库对非维度表仅支持同表组内的表进行 Join 操作（维度表除外），0.9.5版本后支持非相同表组内的表进行Join关联，但是性能较差。
- 一个表组的表可以共享一些配置，例如查询超时时间，如果表组中的单表也进行了这些配置的个性化，那么进行表关联时会通表组级别的配置进行覆盖。
- 一个表组中的所有表的一级Hash分区数建议一致（非强制）。

表

分析型数据库中的表分为普通表和维度表。普通表创建时待指定至少一级分区列和相关分区信息，并且需要指定存放在一个表组中。维度表可以和任意表组的任意表进行关联，并且创建时不需配置分区信息，但是对单表数据量大小有所限制，并且需要消耗更多的存储资源。

在分析型数据库中，普通表的分区目前最多可以有两级。分区种类有 Hash 分区和 List 分区两种，其中 Hash 分区是根据导入数据时已有的一列的内容进行散列后进行分区的，因此目前多张事实表进行快速的 Hash Join 时 JoinKey 必须有分区列参与，并且这些表的Hash分区数必须一致；0.9版本之前，不支持事实表在没有Hash分区键参与关联条件的情况下进行关联，0.9.5版本后，通过Full MPP Mode或小表广播进行计算时无此限制，关于join的详细内容请参见 多计算引擎和 *Hint* 。

另外仅采用Hash分区的数据表，在数据装载时，是进行全量覆盖历史数据的。而List分区则是根据进行导入操作时所填写的分区列值来进行分区的，一次导入的数据会进入同一个List分区中，因此List分区是支持增量的数据导入的。目前分析型数据库支持将表的一级分区设置为Hash分区，二级分区设置为List分区，既可支持Hash join又可支持增量数据导入。另外无论采用何种分区形

态，分析型数据库均不待在用户查询时指定分区列，但是指定分区列或分区列的范围进行查询可能会提高查询性能。

另外根据表的更新方式不同，分析型数据库的表分为离线批量更新的表（适合从离线系统如MaxCompute产出的数据 结果导入到分析型数据库供在线系统使用），以及实时更新的表（可以直接insert/delete单条数据，适合业务系统直接写入数据），实时更新的表不提供二级分区功能（因为天生支持增量，无需二级分区）。



注意： 分析型数据库不支持读写事务，并且数据实时更新时一分钟左右才可查询，另外在一致性方面分析型数据库遵循最终一致性。

28.3 数据类型介绍

28.3.1 支持的数据类型

- boolean：布尔类型，值只能是0或1；取值0的逻辑意义为假，取值1的逻辑意义为真；存储字节数1比特位；
- tinyint：微整数类型，取值范围-128到127；存储字节数1字节；
- smallint：整数类型，取值范围-32768到32767；存储字节数2字节；
- int：整数类型，取值范围-2147483648到2147483647；存储字节数4字节；
- bigint：大整数类型，取值范围-9223372036854775808到9223372036854775807；存储字节数8字节；
- float：单精度浮点数，取值范围-3.402823466E +38到-1.175494351E-38，0，1.175494351E-38到3.402823466E+38， IEEE标准；存储字节数4字节；
- double：双精度浮点数，取值范围-1.7976931348623157E +308到-2.2250738585072014E-308，0，2.2250738585072014E-308到1.7976931348623157E+308，IEEE标准；存储字节数4字节；
- varchar变长字符串类型；
- date：日期类型，取值范围： '1000-01-01' 到 '9999-12-31'，支持的数据格式为'YYYY-MM-DD' 存储字节数为4字节；
- timestamp：时间戳类型，取值范围：1970-01-01 00:00:01.000' UTC到'2038-01-19 03:14:07.999' UTC.， 支持的的数据格式为：'YYYY-MM-DD HH:MM:SS' 存储字节数为4字节；
- **multivalue 多值列类型**，即一个cell中包含多个值，多个值直接的分隔符默认是，当然也可以通过delimeter 关键字指定。

28.3.2 与mysql数据类型对比



说明: 分析型数据库所有的数据类型都不支持unsigned，下表不包含该差异。

分析型数据库数据类型	MySQL	差异
boolean	bool,boolean	一致
tinyint	tinyint	一致
smallint	smallint	一致
int	int,integer	一致
bigint	bigint	一致
float	float[(m,d)]	分析型数据库不支持自定义m和d，而mysql可以
double	double[(m,d)]	分析型数据库不支持自定义m和d，而mysql可以
varchar	varchar	一致
date	date	一致
timestamp	timestamp	分析型数据库只支持到精确到毫秒，而mysql是可以定经度的
multivalue	-	分析型数据库特有的，MYSQL无此类型

28.4 特色功能

作为创新型的实时OLAP服务，分析型数据库提供很多独有的特色功能，以下是对常用部分的简要介绍：

聚集列

在创建表时，您可以指定一列或者若干列作为聚集列。在物理上，一个分区内聚集列内容相同的数据会尽可能的分布在同样的区块内存放，因此如果您的查询Query的条件中会指定聚集列的内容或范围，那么这样的查询性能便会有较大的提升。



说明: 如果您指定多列为聚集列，那么指定的聚集列的顺序便是比较数据是否相同的顺序。

聚集列可以在建表后进行修改，但是目前修改后下次装载数据完毕后生效。

多值列

分析型数据库拥有一个特殊的数据类型，叫做多值列。多值列中可以存入String类型的多个值。在原始数据中，多值列为一列用分隔符（默认为半角逗号，可以建表时进行配置）分隔的String类型。多值列数据存入分析型数据库后，可使用in、contains条件对该列的单个值进行查询，枚举查询后该列的每个值可像一个普通列一样进行各类操作。但不允许在没有进行枚举查询时，对该列直接select或在group by中使用该列。

多值列的通常使用场景是：已有一个实体属性表均为普通列并以实体编号为主键的情况下，需要新增一个用于进行实体筛选的属性，而这个属性和实体编号为多对多的对应关系，按照通常的做法，则新建一张该实体编号和属性两列的数据表，和原有的实体属性表进行Join操作查询。而使用多值列后，性能会被进行Join操作计算高数倍。一个具体的例子：一张用户表中已有用户id和性别、年龄、职业、籍贯的数据，现在需要增加用户购买的商品品类的数据用于筛选，便可将这个商品的品类id存放在用户表的一个新增的多值列中使用。

智能自动索引

分析型数据库拥有智能的自动索引创建机制，通常情况下，分析型数据库会根据导入数据中每一列的分布情况，如该列的枚举值数量的多少，该列的数据是连续的或离散的，自动为导入数据的每一列创建符合该列情况的索引类型，无需您显式指定创建索引或索引类型。而在有些情况下，如果您认为某一列不需进行筛选查询，便可指定该列不需创建索引来节省数据存储空间。另外，0.9版本之前如果一个列需要作为Hash Join的Join Key，那么您需人工指定该列创建Hash索引，0.9版本已废弃Hash索引，可动态根据Join SQL进行优化，完全无需您来干预。

查询CBO优化

分析型数据库拥有高度智能的CBO（Cost-Based Optimization）优化策略。在您发起的一个Query达到分析型数据库后，分析型数据库会智能的判断该Query涉及的数据的分布情况、索引使用情况、缓存使用情况、Query 条件分布等，进行逻辑和物理执行计划优化和重组，尽可能以最优的路径执行Query查询。因此您在大部分情况下，无需关心Query的具体写法，只要保证语义正确即可。另外分析型数据库提供一些Hint参数，可以在Query时对执行计划进行部分调整，详细信息请参见多计算引擎和*Hint*。

28.5 ECU详解

ECU（弹性计算单元），是分析型数据库中存储和计算资源的分配单位。分析型数据库对您的每一个DB会分配若干个计算节点（COMPUTENODE），以及若干个接入节点（BUFFERNODE），接入节点用于接收您的应用前端连接等工作，计算节点用于存储您的数据和进行计算，另外还有若干

个用于放置实时化数据写入缓冲的缓冲节点（BUFFERNODE）。目前分析型数据库仅计算节点是您可按ECU模式进行配置的，分析型数据库会自动根据您的计算节点的量来配置接入节点等其他角色的数量。

计算节点的ECU具有如下属性：

- **内存容量**：该ECU的内存大小。
- **磁盘容量**：该ECU的磁盘容量，您在一个DB中存储的物理数据总量不能超过该DB的全部的磁盘容量，并且由于分析型数据库将您的数据分布到每一个ECU中，若您的数据倾斜导致单个ECU的磁盘空间占满，也会导致数据无法再进入分析型数据库。各个ECU的磁盘使用情况可以在云监控中查看。

目前分析型数据库提供的ECU规格为：

型号	内存	磁盘类型	磁盘容量	磁盘容量（SATA）
c8	45GB	SSD	480GB	--
c1	7.5GB	SSD	60GB	--

0.9版本的分析型数据库，提供基于SATA存储的大容量实例，采用SATA和SSD混合存储，能够大幅度降低存储成本，但是同时查询性能也以数量级而下降。大容量实例的ECU型号通常以字母s开头。专有云中原则上仅万兆网物理机能够运行大容量实例。

ECU数量，可以通过DMS for分析型数据库界面的扩容/缩容功能，或相应DDL进行动态修改。详情请参见*ECU*管理。

28.6 快速开始

28.6.1 开通服务

在专有云中，开通分析型数据库服务的方式请咨询您的系统管理员或运维人员。

28.6.2 创建数据库

操作步骤

1. 登录分析型数据库控制台，具体操作请参见登录云控制台。

2. 控制台数据库管理页面右上角单击**新建数据库**，填写数据库名称、区域（Region所在地）、ECU型号、ECU初始数量。如图 *961: 创建数据库*所示：

图 961: 创建数据库



参数说明：

- 数据库名称：ads_demo（该名字已经被使用，实际操作时请您使用别的名字），数据库名称在分析型数据库全部集群上必须是全局唯一。
- 区域（Region所在地）：分析型数据库的Region所在地，如杭州、北京等。
- ECU 型号：选择c8。分析型数据库以ECU（弹性计算单元）作为资源计量的最小单位。ECU拥有多种型号，每种型号的ECU标识着不同的vCPU核数、内存大小、磁盘空间大小。您在创建数据库时需要根据自己的需求选择这个数据库的ECU型号。
- ECU初始数量：填2。必须是偶数个且至少两个。

注意： ECU型号DB创建后不可修改，ECU数量可以在使用中随时调整（扩容/缩容），关于ECU的详细信息，请参见*ECU详解*。

3. 填好所有选项后，单击**确定**，若返回错误，则根据错误提示进行修正（通常是数据库名称重复或不符合规范，或提交的ECU资源量超过了分析型数据库允许的最大限制），否则创建成功。

   创建成功后，十分钟以内便可查看新的数据库的连接地址。如图 *962:* 数据库地址所示：

图 962: 数据库地址

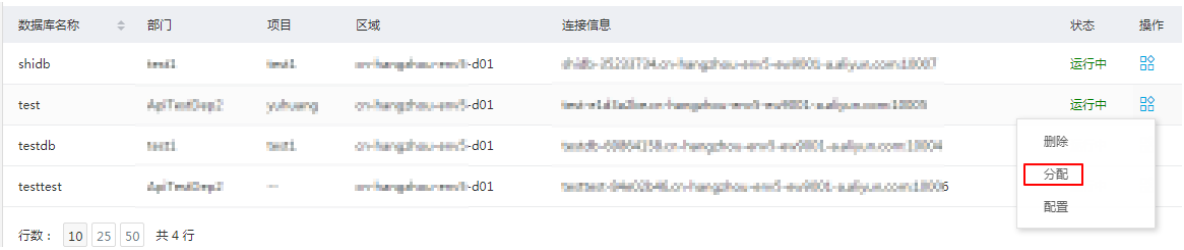


28.6.3 分配和配置数据库

分配数据库

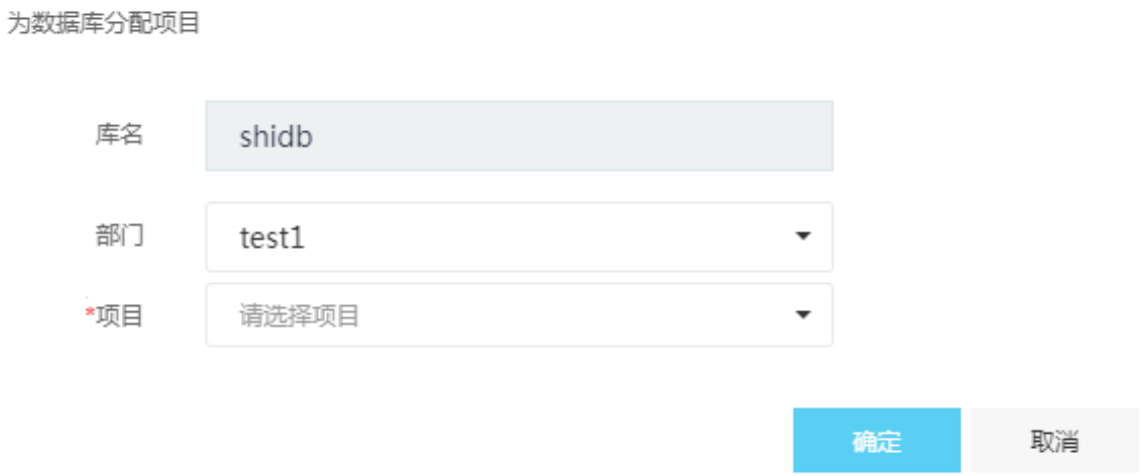
1. 新建数据库后，进入数据库列表。
2. 右击对应数据库后的操作，选择**分配**。

图 963: 数据库分配



3. 填写弹出框中的内容，为数据库分配项目。

图 964: 分配项目



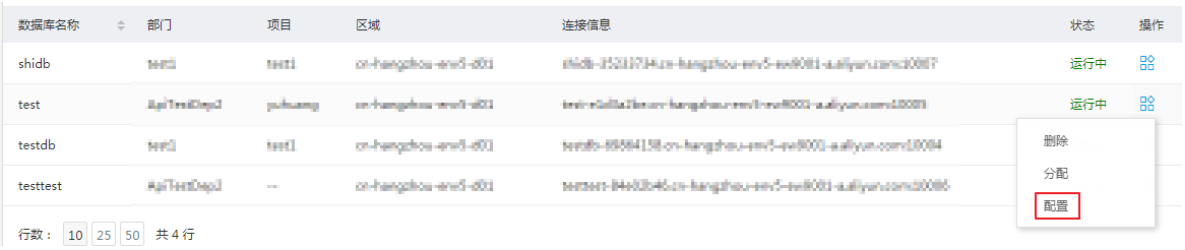
4. 确认选择后，单击**确定**。

> **说明:** 完成上述操作，即成功将数据库分配到相应项目中。

配置数据库

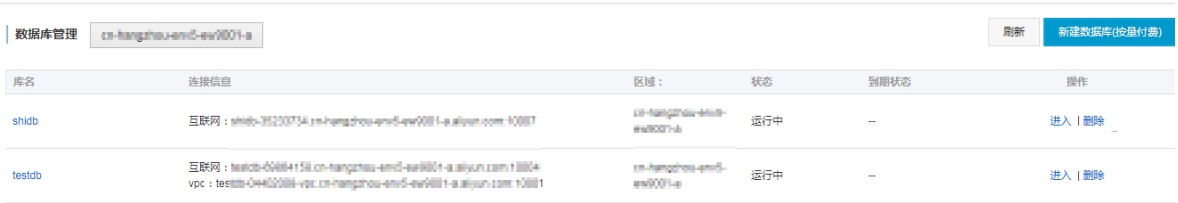
1. 新建数据库后，进入数据库列表。
2. 右击对应数据库后的操作，选择**配置**。

图 965: 数据库配置



3. 进入数据库配置页面，进行刷新、删除等操作。

图 966: 分配项目



在此页面，您可以查看数据库的连接信息、区域、状态等信息，也可进行进入或删除数据库等操作。

说明: 进入数据库后的操作请参见使用*DMS*创建和管理表。

28.6.4 使用DMS创建和管理表

背景信息

分析型数据库采用关系模型存储数据，也就是使用二维表来进行数据的组织和存储。像MySQL一样，将数据放入分析型数据库前需建立对应的数据表。而分析型数据库为了管理相关联的数据表，又引入了表组的概念，详情请参见创建和修改表组。

操作步骤

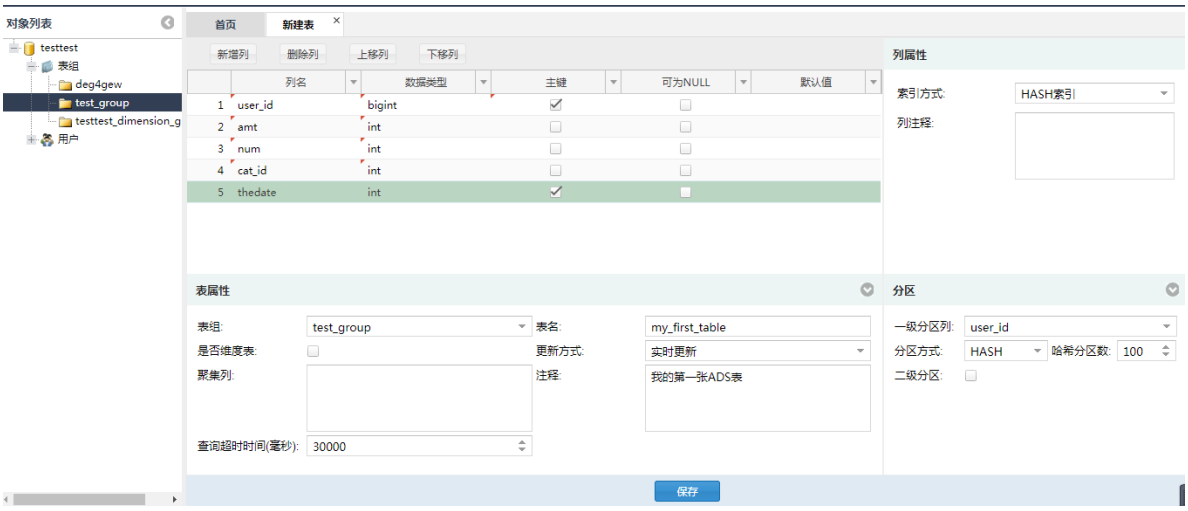
1. 进入DMS for AnalyticDB页面，右击左侧表组对象，选择**新建表组**。
2. 填写新建表组弹出框中的相关内容。

图 967: 新建表组



3. 单击**提交**，页面弹出新建表组成功的提示。
4. 右击新建的表组，选择**新建表**。在这里，我们根据测试数据的情况，建立一张有五列和一级分区的数据表，如图 *968:* 新建表所示。

图 968: 新建表



参数说明如下：

- 表组：填写表组名，此处选择新建的test_group。
- 表名：表名和列名根实际情况填写，目前需要和源头数据表的字段名称一致。
- 是否维度表：默认为否。
- 更新方式：如果这个表的数据来源是批量的从其他系统（例如MaxCompute）导入，则选择批量更新，详情请参见 导入数据 。如果这个表的数据来源来自于直接的insert插入，则选择实时更新。
- 注释：新建表的备注。
- 分区：在分析型数据库中，数据根据分区列进行分布式的存储和计算。

  举例来讲，我们在这里的原始数据是按照user_id 较均匀的进行分布的，所以我们指定按照user_id进行hash分区，分区数调整为40个（一般来讲，每个分区的数据不超过800万条为宜，当然也不绝对，分区数不能超过256个）。

  说明: 分析型数据库拥有强大的自动索引功能，您在创建表时通常无需关心一个列的索引情况，分析型数据库会根据实际数据的分布情况来自动进行索引，所以这里先不调整列的索引设置。

5. 单击**保存**按钮，弹出实际的建表DDL供校验。

图 969: 提交变更



6. 确认无误后，单击**确定**按钮。

   有关表和列的详细说明，请参见 创建和修改表 和 列的类型和属性 。创建表完毕后，右击已有的表即可进行编辑。

28.6.5 导入数据

分析型数据库支持多种接入数据的方式，您可以直接将数据通过 insert/delete SQL写入实时表（详情请参见*INSERT/DELETE*命令），或通过Kettle等ETL工具将本地文件写入分析型数据库，或是通过阿里云数据传输从阿里云RDS中实时同步数据变更（请参见《使用阿里云数据传输实时同步 *RDS* 的数据》章节），或者建立批量导入表从阿里云 MaxCompute（原名 ODPS）大批量的导入数据。

如果在建立表时选择数据来源是批量导入，则分析型数据库提供多种数据导入的方式，如通过data pipeline系列命令（详细信息请参见 导入数据）等方式。在这里，作为测试使用，我们通过DMS界面进行数据导入。

1. 进入DMS页面，选择菜单栏上的导入按钮，弹出**导入数据**对话框。
2. 在弹出框中填写数据源类型、路径等信息，如导入数据所示。

图 970: 导入数据



说明: 这里的数据源表在阿里云MaxCompute 上。因此数据导入路径按照 *odps://project_name/table_name/partition_spec* 的格式来填写。

3. 填写完毕后，单击**确定**。

之后页面会展示导入状态一览，分析型数据库会对导入任务进行调度，根据当前系统繁忙情况和需要导入数据的大小和结构不同，二十分钟至数个小时内数据导入会结束。

如果建表时数据来源选择的是实时写入，那么则可以在建表后直接在SQL窗口中编写SQL。

```
insert into table my_first_table (user_id,amt,num,cat_id,thedate) values (12345, 80, 900, 1555, 20140101);
```

实时写入的表创建后，会有一分钟左右（0.9版本数据，0.8版本一般为十五分钟左右）的准备时间，这时候写入的数据需要在准备完成后才能查询，否则查询会报错。在准备完成后，实时进行insert/delete的数据变更一般延迟一分钟内可查。

说明: 分析型数据库进行实时插入和删除时，不支持事务，并且仅遵循最终一致性的设计，所以分析型数据库并不能作为OLTP系统使用。

若表的数据是离线批量导入，但是数据源是RDS等其他的云上系统，那么我们可以通过阿里云的数据集成（Data Integration）产品进行数据同步，开通CDP后，按照如下示例配置同步Job。

```
{
"type": "job",
"traceId": "rds to ads job test", "version": "1.0", "configuration": {
"setting": {
},
```

```
"reader": {
"plugin": "mysql", "parameter": {
"instanceName": "您的RDS的实例名", "database": "RDS数据库名",
"table": "RDS表名",
"splitPk": "任意一个列的名字", "username": "RDS用户名", "password": "RDS密码",
"column": ["*"],
}
},
"writer": { "plugin": "ads", "parameter": {
"url": "在分析型数据库的控制台中选择数据库时提供的连接信息", "schema": "分析型数据库数据库名",
"table": "分析型数据库表名", "username": "您的access key id", "password": "您的access key secret",
 "partition": "",
"lifeCycle": 2, "overWrite": true
}
}
}
}
```

说明:

- 在运行这个任务之前，需要在分析型数据库中至少授予表的Load Data权限。
- Analytic DB 进行实时插入和删除时，不支持事务，并且仅遵循最终一致性的设计，所以Analytic DB并不能作为OLTP系统使用。

  若表是实时写入的，但是仍需通过数据集成从其他数据源导入到分析型数据库中，则上述reader配置不变，修改writer配置为（以原表列和目标表列相同为例，若不同需在column选项中配置Mapping）：

  ```
  "writer": {
  " plugin": "ads", "parameter": { "writeMode": "insert",
  "url":"在分析型数据库的控制台中选择数据库时提供的连接信息", "schema": "分析型数据库数据库名",
  "table":"分析型数据库表名",
  "column": ["*"],
  "username": "您的access key id", "password": "您的access key secret", "partition":
    "id,ds=2015"
  }
  ```

28.6.6 在应用中连接并使用

背景信息

首次成功导入数据到分析型数据库后，希望应用系统能够连接到分析型数据库来进行数据查询。分析型数据库可以通过任何支持5.1.x 5.4.x 5.6.x协议的客户端进行连接。

说明: 连接所使用的域名和端口号可以在DMS页面的连接信息栏进行查看。连接使用的用户名和密码为DTCenter上相应账号的Access Key。

在PHP中连接分析型数据库

在PHP环境下，假设已经安装好了php-mysql 5.1.x模块（Windows下为php_MySQL.dll），那么新建ads_conn.php，内容如下：

```
$ads_server_name="mydbname-xxxx.ads-cn-hangzhou-1.aliyuncs.com "; //数据库的连接url，请在控制台中的连接信息中获取
$ads_username="my_access_key_id"; // 连接数据库用户名
$ads_password="my_access_key_secret"; // 连接数据库密码
$ads_database="my_ads_db"; // 数据库的名字
$ads_port=3003; //数据库的端口号，请在控制台中的连接信息中获取
//连接到数据库
$ads_conn=mysqli_connect($ads_server_name, $ads_username, $ads_password, $ads_database, $ads_port);
```

执行查询时，可以使用：

```
$strsql="SELECT user_id FROM my_ads_db.my_first_table limit 20;";
$result=mysqli_query($ads_conn, $strsql);

while($row = mysqli_fetch_array($result)) { echo $row["user_id"] ; //user_id为列名
}
```

上述代码即可取出任意十条记录的user_id并打印出。

注意: 分析型数据库在数据查询中不支持SELECT *方式查询所有列。

在JAVA中连接分析型数据库

通常，在JAVA中，通过连接池来使用分析型数据库。在这里以国产的高性能连接池Druid为例，来说明连接分析型数据库的方式：

```
import com.alibaba.druid.pool.*;
DruidDataSource dataSource = new DruidDataSource();
dataSource.setDriverClassName("com.mysql.jdbc.Driver");
dataSource.setUsername("my_access_key_id");
dataSource.setPassword("my_access_key_secret");
dataSource.setUrl("jdbc:mysql://mydbname-xxxx.ads-hz.aliyuncs.com:5544/my_ads_db");
//连接数配置
dataSource.setInitialSize(5);
dataSource.setMinIdle(1);
dataSource.setMaxActive(10);
//启用监控统计功能
dataSource.setFilters("stat");
// for mysql
dataSource.setPoolPreparedStatements(false);
// 使用心跳语句检测空闲连接
dataSource.setValidationQuery('show status like "%Service_Status%";');
dataSource.setTestWhileIdle(true);
```

如上，若是在任何语言中需要使用心跳SQL来进行分析型数据库服务状态检测，请使用show status like "%Service_Status%" 语句，若返回一行两列且第二列为1，则分析型数据库服务正常。

28.6.7 新增用户和授权

在分析型数据库中，创建一个数据库的账号为这个数据库的owner账号，拥有最大化的权限。在很多情况下，我们不希望这个账号被更多的人掌握，所以需要授权其他人仅使用较小的权限连接分析型数据库，这时便需进行权限管理，更多详情请参见用户与权限。

操作步骤

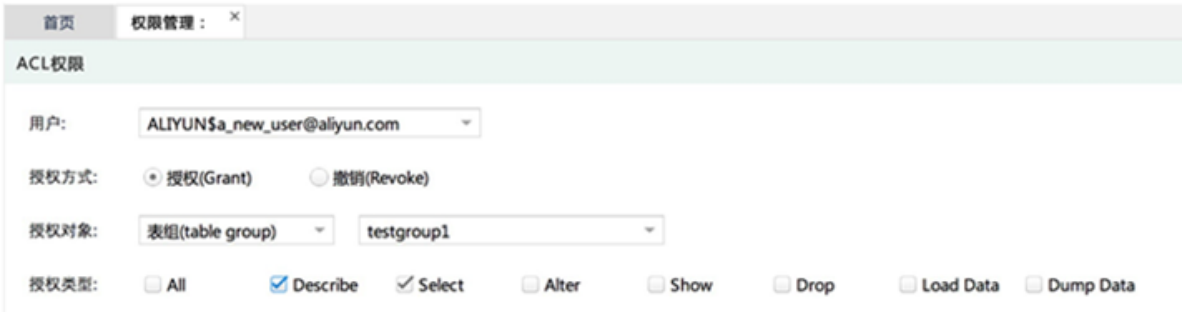
1. 进入DMS页面，单击左侧对象列表中的用户，即可看到可以访问该数据库的全部用户列表。
2. 右击用户名，选择**新建授权**或单击顶部菜单栏中的**新建授权**。

图 971: 新建授权



3. 填写弹出框中的相关内容。

图 972: 权限管理



在打开的窗口中，输入需要授权的用户账号，由于分析型数据库支持多种账号来源，故在输入账号时声明账号来源，例如 ALIYUN$ 前缀代表该账号是阿里云账号，那么输入一个需要授权的云账号如ALIYUN$a_new_user@aliyun.com，然后假设希望这个账号对testgroup1表组下的所有表有查看元信息和只读权限，那么进行如图 *972: 权限管理*中的操作。

4. 单击**保存**，完成授权操作。随后a_new_user@aliyun.com用户便可使用自己的AccessKey登录分析型数据库执行有权限的操作。

28.7 DDL

28.7.1 创建数据库与配置

创建数据库的相关操作请参见创建数据库 ，若要创建数据库，请联系您的运维管理员或DBA。

配置数据库的相关操作请参见分配和配置数据库。

28.7.2 创建和修改表组

本节将详细描述如何通过DDL来创建和修改表组以及表组各属性的含义。

创建表组时，提交如下SQL：

```
create tablegroup db_name.tablegroup_name options(minRedundancy=2 executeTimeout=30000);
```

其中db_name为数据库名称，tablegroup_name为表组名称（不能和现有表组重叠）。 options部分为可选项：minRedundancy表示该表组的副本数，默认为2，可配置为 1、2、4、8。如果将一个表组配置为1副本，那么这个表组中的表在数据导入时会有不可用的时间。而将表组副本数配置为4或更高，可以一定程度的增加分析型数据库的最大承受的QPS，但是数据存储费用也会相应增加。在绝大部分情况下，不推荐修改任何表组默认配置。

executeTimeout表示该表组的全局Query超时时间，默认为30000，单位毫秒。当待删除一个表组时，可以提交如下SQL：

```
drop tablegroup db_name.tablegroup_name;
```



说明: 允许删除没有任何表的空表组，维度表组不允许删除。

仅当待修改表组的两个属性中的任何一个时，可以提交如下SQL：

```
alter tablegroup db_name.tablegroup_name key=value;
```

其中， key为属性名，value为新的属性值。minRedundancy修改后待下次装载数据时才会生效。

28.7.3 创建和修改表

本节将详细描述如何通过DDL创建和修改表。

建表DDL由于分析型数据库有一些传统关系型数据库没有的特性，所以分析型数据库的DDL在遵循类MySQL规范的基础上，有不少的独有的属性和语法。分析型数据库中创建事实表DDL语法结构如下：

```
CREATE TABLE db_name.table_name ( col1 bigint COMMENT 'col1',
col2 varchar COMMENT 'col2', col3 int COMMENT 'col3',
col4 bigint COMMENT 'col4',
col5 multivalue COMMENT 'col5 多值列', [primary key (col1, col3)]
)
PARTITION BY HASH KEY(col1) PARTITION NUM 50
[SUBPARTITION BY LIST (part_col2 long)]
[SUBPARTITION OPTIONS (available_partition_num = 30)] [CLUSTERED BY (col3,col4)]
TABLEGROUP ads_test ;
[options (updateType='{realtime | batch}')]
```

其中'[]'中的内容为视情况填写的可选项。创建事实表（也就是非维度表）时，必须指定的是一个数据库名（db_name）和表名（table_name），以及至少一列的列信息，至少一个分区的信息以及一个表组。

首先，根据表的数据更新方式不同，分析型数据库的表根据updateType分为批量更新表（仅能够离线批量更新数据）和实时更新表（能够通过insert/delete实时更新数据），用updateType以区分，如果updateType选项不填则默认为批量更新表。

说明： updateType=realtime 即实时更新表时，必须指定合法的主键并且不能有二级分区。

列信息的基本格式为"列名类型COMMENT '注释'"，关于列的更多属性，请参见 列的类型和属性。"[primary key (col1, col3)]"一段指定了表的主键，如果表为批量更新的，则主键没有意义，而表为实时更新的，则必须指定 主键并且主键中必须含有一级Hash分区列（可以是联合主键）。

表分区关于分区方面，目前分析型数据库支持最多两级分区，并且一级分区仅支持HASH分区，二级分区仅支持LIST分区。HASH分区是一种动态分区值类型，即根据实际数据中的某一列的内容进行分区。所以在语法上，一级HASH分区的用法是：PARTITION BY HASH KEY(col1) PARTITION NUM 50。

其中col1为待进行分区的列名，必须是表中实际存在的列。'PARTITION NUM' 为分区数量，一般根据该表的数据量确定，每个分区一般不超过1500万条记录为宜（亦可通过划分二级分区实现无限扩展）。另外HASH分区列的数据分布要尽可能均匀，不能有非常明显的倾斜（即分区列值），否则会较严重的影响查询性能。

说明：

- 目前分析型数据库要求一个表组下所有表的一级分区数目一致，所以建立第一张表时指定一级分区数量时请谨慎。一级分区数量默认不能超过256个。
- 若一张表仅有一级HASH分区并且是批量导入的表，则每次导入数据时会对已有进行全量覆盖。

若需要每次导入数据时增量导入，则需再指定二级List分区信息：

```
SUBPARTITION BY LIST (part_col2 bigint)
SUBPARTITION OPTIONS (available_partition_num = 30)
```

二级List分区为非动态分区，即分区值不是由数据本身决定的，而是由每次导入/写入数据时用户指定的。所以在进行分区信息定义时需指定一个和现有数据中的列不同的新列名，以及这个列的类型（目前仅支持long）。二级分区有一个可选属性，available_partition_num，即为最大保留的二级分区数，当新的数据装载 进来后，若线上存在的二级分区数大于这个值，那么会根据二级分区的值进行排序，下线最小的若干分区的数据。

0.8版本的实时更新表暂不支持二级分区，0.9 版本支持。但是实时更新表的二级分区仅用于极大的扩展单表容量，以及进行生命周期管理，实时更新表的增量更新不依赖于二级分区。

常见的用法是，将经常需要进行Join的列（例如买家ID）作为一级Hash分区列，而将日期列作为二级分区列。这样的表既可以进行大表Join的加速，又可以每天进行增量数据导入，并且指定保留若干天的数据在线上来进行生命周期管理。

但是，一级分区数量和二级分区数量的设置，并不是越多越好的。而是要看表的数据量，以及数据库拥有的资源数。若一级二级分区过多而数据库的资源数过少，则很容易让分区的数据Meta将内存耗尽。

聚集列

CLUSTERED BY子句用于指定聚集列，您可以把一列或者多列指定为聚集列，如果指定多列，那么该表的数据聚集顺序按照DDL中这个子句中指定的列组合顺序进行排序。通常，如果您将一个表的查询中肯定会涉及到的并且数据区分度很大的列设置为聚集列，有时候能较显著的提升查询性能。

事实表与维度表

上述DDL默认创建的是事实表，另外事实表的创建上，默认有如下限制：

- 一张事实表至少有一级Hash分区并且分区数不能小于8个，默认不能大于256个。
- 一个事实表组最多可以创建256个事实表。
- 一个事实表最多不能超过1024个列。
- 一张事实表二级分区默认不能大于 365*3 个。

与创建事实表相比，创建维度表要简单的多：

```
CREATE DIMENSION TABLE db_name.dim_table_name ( col1 int comment 'col1',
col2 varchar comment 'col2' [primary key (col1)]
```

```
)
[options (updateType='{realtime | batch}')]
;
```

创建维度表时只需指定数据库名和表名即可，维度表会创建到统一的维度表组，并且无需指定分区信息。修改表在表已经创建好之后，目前支持有限的修改：主要是支持增加列。

```
ALTER TABLE db_name.table_name ADD column col_new varchar;
```

注意：批量导入表新增加的列在数据重新装载后才会生效，实时写入表新增列后，一般需要几分钟后才可以读写。

最大二级分区数目前可以在建表后进行修改，但是修改后的下一次数据导入发起后才会生效：

```
ALTER TABLE db_name.table_name subpartition_available_partition_num = N
```

N为新的二级分区数。Create-table-as-select0.9版本后的分析型数据库，支持通过create table as select语句基于查询创建临时表，语法如下：

```
CREATE [DIMENSION] TABLE [IF NOT EXISTS] new_tbl [(PRIMARY KEY (n))] [AS] SELECT expr.. FROM orig_tbl ..;
```

select子句的语法请参见*Select*语句基本语法。

28.7.4 列的类型和属性

在DDL中，建表时一列的定义是：

```
col_name type [NOT NULL | NULL] [DEFAULT default_value] [PRIMARY KEY] [COMMENT 'string'] [column_options(pr ecision, scale, disableIndex...)]
```

其中col_name为列名，type为列的数据类型，详细信息请参见特色功能。[NOT NULL | NULL]是否可为空，以及 [DEFAULT default_value] 定义的列的默认值和标准的MySQL DDL中相同。

关于[PRIMARY KEY]主键部分，对于批量更新表，分析型数据库中主键的概念是弱化的，分析型数据库不要求 一个表有主键，有主键的表的性能和用法上和没有主键的表之间没有任何区别。若一个表进行数据导入时该次导入的数据中存在主键冲突，则该次导入会失败并且报错。对于实时更新表，请使用在所有列尾部的primary key (col1, col3)中指定主键。

列属性上，一个列可以设置列属性disableIndex = true，用于屏蔽分析型数据库的默认列索引，不过需要注意的是，如果这样做，这个列应该不在实际查询中所筛选和计算的。precision和scale属性则是针对decimal数据类（目前暂未上线，未来会上线该功能）特有的属性，precision为数字整体有效数字个数， scale为小数点后的数字个数。

28.7.5 索引

同主键一样，分析型数据库中索引的概念也是弱化的。在前文中介绍分析型数据库拥有高度智能的自动化索引机制，所以通常您无需亲自为自己的数据表配置索引。但是有一种情况例外：0.8版本下您若对某列进行Hash Join时，无论是事实表之间的Join还是事实表和维度表的Join，都需为事实表的该列建立索引。

建立索引的语句如下所示：

```
ALTER TABLE tbl_name ADD INDEX [index_name] [index_type] (index_col_name) [comment=’’];
```

其中，[index_type]为索引类型，需要指定为HashMap，index_col_name为被索引列的列名。示例如下：

```
ALTER TABLE db_name.table_name
ADD INDEX user_id_index HashMap (user_id)
```

批量导入表索引修改后，需要重新导入数据后生效。实时写入表一般在24小时内自动生效，或如果要加速这个过程，可以执行optimize table命令，如下所示：

```
optimize table <tablename>;
```

执行成功一段时间后，新的索引会生效。

0.9及以后版本中分析型数据库会自动处理Hash Join时的索引构建，故无需用户自行创建HashMap索引也可以进行Hash Join，因此HashMap索引被废弃。

28.7.6 ECU管理

变更数据库容量，有以下两种方式。

- 通过DMS for分析型数据库界面的扩容/缩容功能进行修改。
- 通过DDL语句进行动态修改。

DMS界面更改

1. 进入DMS for分析型数据库页面，导航至**实例管理** > **DB容量管理**页面。
2. 单击右上角的**容量变更**，填写需要变更的ECU数量，如图 *973:* 容量变更所示。

图 973: 容量变更



3. 确认数量后，单击**确定**。

> **说明:** 在DMS for分析型数据库中，您可以在图形界面上进行扩容/缩容，以及查看扩容/缩容执行状态等操作。

DDL动态修改

通过DDL来进行ECU变更（扩容/缩容）时，需要Alter Database权限或DB owner角色。

```
ALTER DATABASE SET ecu_count = N;
```

N为要设置的目标ECU数量，若目标ECU数量大于当前的ECU数量，则为扩容行为；若目标ECU数量小于当前的ECU数量，则为缩容行为。若用户的数据量大于目标ECU的总存储，则更改会失败。

缩容和扩容都不是瞬时的同步操作，可以使用元数据查询状态。

```
select * from information_schema.resource_request;
```

查看目前已有的ECU状态。

```
select * from information_schema.current_instances;
```

28.8 DML

28.8.1 Select语句基本语法

通过 DML 进行计算是用户在分析型数据库中最常用的操作。目前分析型数据库对 DML 支持使用 SELECT 进行查询，以及支持 INSERT、DELETE 等数据修改操作（INSERT/DELETE 请参

见 *INSERT/DELETE*命令 ）。另外本节的内容需要配合附录一：函数的使用与逻辑表达式和特殊语法的特殊语法，以及多计算引擎和*Hint*中的双计算引擎和 Hint 的说明来阅读。

分析型数据库中 SELECT 语句的基本结构如下：

```
SELECT
[DISTINCT]
select_expr [, select_expr ...] [FROM  table_references [WHERE  filter_condition]
[GROUP BY {col_name | expr | position} [HAVING  having_condition]
[ORDER BY {col_name | expr | position} [ASC | DESC], ...]
[UNION ALL (SELECT select_expr..)] [{UNION|INTERSECT|MINUS}  (SELECT  select_expr..)]
[LIMIT {row_count}]
```

下面，将一个完整的 SELECT 分为几个部分加以介绍：列投射部分（列和列表达式）、FROM/JOIN 子句（表扫描、表连接与子查询）、WHERE 子句（过滤表达式）、GROUP BY/HAVING 子句（分组和分组后过滤）、ORDER BY 子句（排序）、两个查询间的 UNION [ALL]/INTERSECT/MINUS、LIMIT 子句（返回行数限制）。

列投射部分（列和列表达式）

- 列投射（列和列表达式）分析型数据库中，SELECT 语句中的列投射的基本结构为：

  ```
  SELECT [DISTINCT] select_expr [, select_expr ...]
  ```

  其中 select_expr 可以是基本列（直接从表或子查询中选出的），也可以是列表达式（包括算数计算、聚合函数、系统 UDF 等）。

  当 select_expr 为基本列时，基本列是表中的列，这里的表可以是物理表也可以是虚表（物理表的别名引用或子查询产生的表）。

  例如：

  - SELECT A.a FROM A：指定表名和列名，表来自 FROM 子句，列属于FROM子句指定的表。
  - SELECT a FROM A：仅指定列名，列属于 FROM 子句指定的表。
  - SELECT A.a AS x FROM A：指定表名和列名（带别名），表来自 FROM 子句，列属于 FROM 子句。
  - SELECT a AS x FROM A：仅指定列名（带别名），表来自 FROM 子句，列属于 FROM 子句。
  - SELECT b, a FROM A JOIN B ...，仅指定列名，列属于 FROM 子句指定的第一个包含该列的表。
  - SELECT A.a FROM (SELECT A.a, b FROM A JOIN B ...)，指定表名和列名，列属于子查询返回的列（<table>.<column>完全匹配）。

- SELECT a FROM (SELECT a, b FROM A JOIN B ...)，仅指定列名，列属于子查询返回的列（<column>完全匹配）。
- SELECT X.a FROM (SELECT a FROM A) AS X：指定表名和列名，表来自子查询且是别名引用，列属于子查 询返回的列（<column>完全匹配）。
- SELECT X.x FROM (SELECT A.a AS x, b FROM A JOIN B ...) AS X：指定表名和列名，表来自子查询且是别名引用，列属于子查询返回的列且带别名。
- SELECT count(distinct a) from A where b = 123， a 不是分区列，且经过 where 条件筛选后数据量不超过 100万条。
- SELECT NULL ...：空值列。
- SELECT 1, 2 ...：常量值列。
- SELECT true ...：常量值（boolean 类型的取值使用 1 或 0）列。
- SELECT * ...：通配符代表所有列暂不支持。
- SELECT A.* ...：通配符代表特定表的所有列。
- SELECT a, b, a ...：重复列。

0.8版本中不支持但是 0.9 版本的 Full MPP Mode 支持（详细信息请参见逻辑表达式和特殊语法）：

- SELECT distinct a | count(distinct a) from A group by b：其中 a 不是分区列。
- SELECT distinct a | count(distinct a) from A where b = 123, a 不是分区列，且经过 where 条件筛选后数据量 超过100万条。

当 select_expr 中使用列表达式时：

- SELECT a + b ...：常用算术操作符（+，-，*，/, %），其中表达式（a，b）是合法的值、基本列或列表达式，但必须是数值类型。
- SELECT min(a), UDF_EXAMPLE(a, s) ...：系统内置或用户自定义函数，可以是值、基本列或列表达式，但数据类型要符合函数参数要求。
- SELECT CASE WHEN a > 0 THEN a + b ELSE 0 END ...：CASE-WHEN 表达式，其中表达式（a > 0）是合法的行过滤表达式（参看后面），THEN 和 ELSE 取值是合法的列表达式，而且数据类型相同（int）。
- SELECT a IS NULL, a IS NOT TRUE, a >= 0, s LIKE '%foo' ...：合法的关系表达式（参看后面）。
- SELECT ! a, NOT s ...：合法的逻辑表达式（参看后面）暂不支持。

- SELECT a IS [NOT] TRUE, a BETWEEN 1 AND 2, a IN (1, 2), a CONTAINS (1, 2, 3)：不支持部分关系表达。
- SELECT a XOR b：不支持逻辑表达式（请参见*INSERT/DELETE*命令 ）。
- SELECT a & b FROM A, B：不支持位操作。

当 select_expr 为列聚集函数表达式时：包含任何聚集函数的列表达式，即聚集表达式。例如：

- SELECT COUNT(*)：常用聚合函数，详细信息请参见附件中函数表。
- SELECT UDF_SYS_COUNT_COLUMN(a) FROM A：系统内置聚合函数（UDF_SYS_...）。
- SELECT SUM(CASE WHEN ... THEN ... ELSE ... END)：聚合表达式（SUM(...)）套列普通表达式（CASE WHEN...）做为其子表达式。
- SELECT SUM(a) / COUNT(a) FROM A：列普通表达式（/）套聚合表达式（做为其子表达式） 0.8 版本中不支持但是 0.9 版本的 Full MPP Mode支持。
- SELECT a + COUNT(*) FROM A：普通表达式（+）不能同时套普通表达式（a）和聚合表达式（COUNT(...)）。
- SELECT SUM(COUNT(*))：聚合表达式不能套聚合表达式做为其子表达式。

FROM/JOIN子句（表扫描、连接与子查询）

用法：

```
FROM table_references
```

或

```
FROM table_referencesA as table_aliasA JOIN table_referencesB as table_aliasB on join_conditions
```

或

```
FROM (SELECT ...) as aliasA JOIN table_referencesB as table_aliasB on join_conditions
```

或

```
FROM (SELECT ...) as aliasA JOIN (SELECT ...)as aliasB on join_conditions
```

例如：

- FROM A：指定表。
- FROM (SELECT ...)：子查询。

- FROM A JOIN B ON A.a = B.b JOIN C ON A.a = C.c：单表或多表连接，默认是符合条件记录的笛卡尔集。

  说明: 目前分析型数据库中使用 JOIN 时若一方是物理表，那么物理表参与 JOIN 的列必须是分区列并且已建立 HASH 索引。
- FROM (SELECT ...) AS X JOIN A on X.x = A.a：表、子查询和 JOIN 结合使用。
- SELECT ... FROM A JOIN (SELECT ...)：在 V0.6.x 及后续版本支持。
- SELECT ... FROM (SELECT ...) JOIN (SELECT ...)：两个子查询之间的 Join，在 V0.6.x 及后续版本支持 不支持或语义错误。
- SELECT ... FROM A FULL JOIN B：不支持全连接。
- SELECT ... FROM A RIGHT JOIN B：不支持右连接，需要转换为左连接。
- SELECT ... FROM A SEMI JON B：不支持半连接。
- SELECT ... FROM A, B：单表或多表连接，但没有 ON 条件。
- SELECT ... FROM A, B WHERE A.a = B.b：单表或多表连接在 WHERE 子句中有隐含 ON 条件，但是没有 ON 子句的，暂不支持。

0.8 版本中两个表关联可运行的充要条件（四原则）：

- 两个表均为事实表且在同一个表组，或两个表中有一个是维度表。
- 两个表均为事实表且拥有相同的一级分区列，或两个表中有一个是维度表。
- 两个表均为事实表且关联条件(on)中至少含有一个条件是两个表各自的分区列的等值关联条件，或两个表中有一个是维度表。
- 关联条件（on）中的条件两端有有效的 HashMap 索引。

0.9 版本中两个表关联可加速运行的条件：

- 两个表均为事实表且在同一个表组，或两个表中有一个是维度表。
- 两个表均为事实表且拥有相同的一级分区列，或两个表中有一个是维度表。
- 两个表均为事实表且关联条件（on）中至少含有一个条件是两个表各自的分区列的等值关联条件，或两个表中有一个是维度表。
- 0.9 版本若使用 Full MPP Mode 支持任意形态的表关联。

WHERE 子句（过滤表达式）

WHERE 子句（过滤表达式）在过滤表达式中，单个条件之间可以用 AND 和 OR 进行连接，支持括号决定条件表达式的优先级，亦支持表达式嵌套。

像普通的 SQL 一样，单个条件的组成是一个布尔判断。即无论如何组合，单个条件最终是一个返回 True/False 的表达式。在最简单的情况下，单个条件由左侧的主体、中间的比较操作符和右侧的断言值组成。

例如：column_name IS NOT NULL 中， column_name 为布尔主体，IS NOT 为比较操作符，NULL 为断言值。

分析型数据库中比较操作符支持 = 、 >= 、 > 、 <= 、 < 、 <> 、 != 等基本操作符，也支持 IS、IN、 CONTAINS、BETWEEN...AND... 、LIKE 这样的关系操作符，并且关系操作符均支持其反义的版本（ IS NOT 、 NOT IN 、 NOT LIKE 等）。

布尔主体可以是一个列，也可以是表达式的嵌套。例如：

- WHERE a > 0：关系表达式（ >，>=，<，<=，=，<>，!= ）；
- WHERE a IS [NOT] NULL：关系表达式，判断是否为空值；
- WHERE a BETWEEN 1 AND 10：对于表达式（ a ）是数字类型（ long，int，short，double，boolean ）和 时间类型（ date，time，timestamp ）支持；
- WHERE s LIKE '%sample'：对于表达式（ s ）是字符串（ string ）类型支持；
- WHERE a IN (1,2,3)：对于表达式（ a ）是多值列类型支持；
- WHERE a IN (select a from dim_table where c=1)：对于对于表 dim_table 是维度表，支持 IN 后带子查询（ V0.6.2之后的版本支持 ）；
- WHERE a CONTAINS (1,2,3)：对于表达式（ a ）是多值列类型支持；
- WHERE a > 0 AND b > 0：逻辑表达式；
- WHERE (a >0 OR b > 0) AND c > 0：支持括号决定条件表达式的优先级；
- WHERE (CASE WHEN (a + b) > 0 THEN 0 ELSE 1 END) != 0 AND UDF_EXAMPLE(a, b) > 0：表达式嵌套。

不支持或语义错误：

- WHERE EXISTS (SELECT...)：不支持；
- WHERE a = ANY (SELECT ...)：不支持；
- WHERE a = ALL (SELECT ...)：不支持。

0.8版本中不支持但是 0.9 版本的 Full MPP Mode 支持：
WHERE a IN (select a from fact_table where c=1)：fact_table 不是维度表的情况下 0.8 版本不支持
GROUP BY/HAVING 子句（ 分组和分组后过滤 ）。

GROUP BY 子句的用法

GROUP BY {col_expr}

col_expr 为列（表中的列或子查询、别名产生的列）。

例如：

- `... FROM A GROUP BY a：仅指定列；
- `... FROM A GROUP BY A.a：同时指定表名和列；
- GROUP BY s， a IS NULL，b + c， UDF_EXAMPLE(d)：一个或多个列表达式；
- SELECT a, SUM(a) FROM A GROUP BY a, s：可以包含不属于列透视的列表达式（s`）；
- `... FROM (SELECT A.a AS x, b FROM A JOIN B ...) AS X JOIN C ... GROUP BY b, c, X.x：包含在子查询或多表连接的一个或多个列表达式；
- `SELECT CASE WHEN a > 1 THEN 'Y' ELSE 'N' END AS x ... GROUP BY x ：别名引用产生的列表达式；
- SELECT a, b, c ... GROUP BY 3, 1：按位置引用列投射的列表达式 0.8 版本中不支持但是 0.9 版本的 Full MPP Mode 支持；
- ... FROM A GROUP BY s ORDER BY COUNT(a) LIMIT 100：分区表达式（s）不是分区列（a）同时又有TOP-N（ORDER BY COUNT(a) LIMIT 100）；
- 0.8版本下计算的结果可能是不精确的。

HAVING子句的用法

HAVING 子句用于在 GROUP BY 子句执行后对分组后的结果进行过滤。支持使用列投射和聚合表达式后的结果进行过滤。

HAVING having_condition

例如：

- SELECT a, sum(length(s)) AS x FROM A JOIN B ON A.a = B.b GROUP BY a HAVING (CASE WHEN (x / a > 1)THEN a ELSE 0 END) != 0：分组表达式（a）是分区列，HAVING 表达式是基于列表达式（a，sum(length(s))，s）的行过滤表达式 0.8 版本中不支持但是 0.9 版本的 Full MPP Mode 支持；
- SELECT s, count(a) AS x FROM A GROUP BY s HAVING x > 1：分组表达式（s）不是分区列（a）。

ORDER BY 子句（排序）

例如：

- SELECT a FROM A ORDER BY a DESC LIMIT 100：按照分区列（a）排序并返回TOP-N（100）；
- SELECT a FROM A ORDER BY (a + b) DESC LIMIT 100：按照非分区列（a）排序并返回TOP-N（100）；
- SELECT a, COUNT(a) FROM A GROUP BY a ORDER BY x：按照分区列（a）分组聚合再排序。

0.8版本中不支持但是 0.9 版本的 Full MPP Mode 支持：

SELECT a + b, COUNT(1) AS x FROM A GROUP BY a + b ORDER BY x：0.8 版本下按照非分区列表达式（a+b）分组聚合再取 TOP-N 是非精确的。

UNION [ALL]/INTERSECT/MINUS、LIMIT 子句

UNION [ALL]/INTERSECT/MINUS子句UNION/INTERSECT/MINUS 在 0.8 版本下仅支持操作字段为分区列，但是在 0.9 版本的 Full MPP Mode 无任何限制。

LIMIT 子句（返回行数限制）示例：

- SELECT ... LIMIT 100：返回行限制为100 行不支持；
- SELECT ... LIMIT 100, 50：不支持带偏移量的返回行限制。

> **说明:** 分析型数据库会对通过 SELECT 语句进行查询的返回行数做全局的最大限制，公共云上目前为 10000 行，若不加 LIMIT 或 LIMIT 行数超过 10000，则只会返回 10000 行。

28.8.2 逻辑表达式和特殊语法

表Join的条件

为了更高效的进行表关联，分析型数据库对表关联操作在分析型数据库中，两个事实表进行Join的条件是：

- 这两个表在一个表组。
- 这两个表的Join Key是Hash分区列。
- 两张表的Hash分区数必须一致，否则Join结果不准确。
- 两张表的Join Key至少有一列建立了HashMap索引，推荐建立在数据量较小的一侧。

维度表参与的关联，只符合上述条件之一即可。逻辑表达式在分析型数据库中对逻辑表达式的支持如下：

- NOT a：如果表达式（a）是0则返回1，非0则返回0，NULL则返回NULL。
- !a：如果表达式（a）是0则返回1，非0则返回0，NULL则返回NULL。
- a && b：如果表达式（a）和（b）都不是NULL且都非0则返回1，如果其中有一个是0则返回0，否则返回NULL。
- a || b：如果表达式（a）和（b）都是0，则返回0；如果都不是NULL则返回1；如果其中有一个为NULL，另外一个是0，则返回0；如果是非0则返回1；如果都是NULL，则返回NULL。

示例如下：

- NOT 0：1
- NOT 5：0
- NOT NULL：NULL
- NOT a：根据表达式（a）的取值决定
- ! 0：1
- ! 5：0
- ! NULL：NULL
- ! a：根据表达式（a）的取值决定
- 1 && 5：1
- 1 && 0：0
- 5 && 0：0
- NULL && 0：0
- NULL && NULL：NULL
- NULL && 5：NULL
- a && b：根据表达式（a）和表达式（b）的取值决定
- a AND b：根据表达式（a）和表达式（b）的取值决定
- 0 || 0：0
- 1 || 5：1
- NULL || 1：1
- NULL || 5：1
- NULL || 0：0

- NULL || NULL：NULL
- a || b：根据表达式（a）和表达式（b）的取值决定
- a OR b：根据表达式（a）和表达式（b）的取值决定

多值列的用法和限制

在分析型数据库中，多值列这个数据类型有其特殊的用法和限制。对于多值列，不允许在不经由WHERE子句中用IN/CONTAINS进行枚举筛选的情况信下直接在SELECT/GROUP BY中使用。示例如下：

- SELECT m, COUNT(a) FROM A WHERE m IN('foo', 'bar') GROUP BY m， m为多值列，这种情况下可支持。
- SELECT m FROM A limit 10， m为多值列时不支持。
- SELECT m, COUNT(a) FROM A GROUP BY m，m为多值列时不支持。

  另外，由于多值列的数据结构的特殊性，在对含有多值列的行按照多值列进行分组时，多值列的每一个单值会等价于产生一个新的行，计算count/sum等聚合函数的结果可能会被多次计算。

例如原始数据如下：

a	m
0	'bar'
1	'foo', 'bar'
2	'foo', 'bar', 'oh'

在此之上执行SELECT m, COUNT(a) FROM A WHERE m IN('foo', 'bar') GROUP BY m的结果，如下所示：

m	COUNT(a)
'foo'	2
'bar'	3
'oh'	1

非分区列的使用限制分析型数据库作为分布式实时OLAP系统，其数据根据分区列分布在不同的物理服务器中，所以在一些特殊的情况下，分区列和非分区列的使用上有所不同。

- count(distinct cor_expr)中cor_expr必须是分区列或分区列衍生的列，否则不能和group by连用。
- SELECT distinct cor_expr...中若cor_expr不是分区列或分区列衍生的列，则结果可能不精确。

- TOP-N 计算（order by expr limit 100），若是同时和非分区列的GROUP BY子句连用，结果可能不精确。

0.9版本的Full MPP Mode支持上述SQL的正确运行，详细信息请参见 多计算引擎和 *Hint* 中的子查询使用限制。分析型数据库中，0.8版本及先前版本，子查询使用有所限制，示例如下：

```
select count(*) from (
select a from tbl group by a
) tmp;
-- a不是一级分区列时结果不准确，当group by a 返回结果不超过10000条时，可以临时使用：
select udf_sys_rowcount(*) from ( select a from tbl group by a
) tmp;
select a, cnt from (
select a,count(b) as cnt from tbl group by a
) tmp;
--  a,b均不是一级分区列时结果不准确，可以临时使用：
select a, sum(cnt) from (
select a,count(b) as cnt from tbl group by a
) tmp group by a;
```

0.9 版本的FullMPP Mode支持上述SQL的正确运行，详细信息请参见 多计算引擎和 *Hint* 。

28.8.3 INSERT/DELETE命令

如果一个表是实时写入的表，则分析型数据库会支持对该表的单条数据进行Insert/Delete操作，不过由于分析型数据库不支持事务，对于Insert/Delete命令，有一些限制。

在分析型数据库中Insert语句的语法如下：

```
INSERT [IGNORE]
INTO tbl_name (col1,col2...)
VALUES (value1, value2...), (value1, value2...)....
```

在分析型数据库中，能够实时插入的表一定要定义主键或组合主键，和MySQL有一个非常大的不同是，分析型数据库在进行数据插入时，若发现同主键的数据时，默认执行覆盖行为。若使用insert ignore语法，则在发现有同主键的数据时，丢弃新插入的数据，保留原有数据。但是无论如何，在主键冲突时分析型数据库无法直接返回错误给insert的执行方。

在分析型数据库中的Delete语句的语法如下：

```
DELETE FROM tbl_name WHERE where_definition
```

where_definition中暂不支持函数定义。

对分析型数据库是数据插入和删除操作，原则上在分析型数据库操作返回成功后一分钟内可查（但不保证一定是一分钟的数据可见延迟），不过在表刚刚创建时的十分钟内，数据可能无法查询。

Delete语句执行后，数据会查询不到，但不会在物理上立刻删除，原则上24小时内会自动清除掉，如果删除了大量数据想立刻生效，可执行如下语句：

```
optimize table <tablename>;
```

执行成功一段时间后，物理上数据会被删除。

最终一致性：对于主键相同的数据的多次变更，分析型数据库会遵循分析型数据库返回语句执行成功的顺序进行；对于主键不同的数据的两次变更，分析型数据库不保证先执行的变更会比后执行的变更更优先的查询到。但是当业务端暂停数据写入的若干时间后，分析型数据库会保证数据的最终一致。

若希望达成数据写入速率的最大化，有以下建议：

- 每条insert插入多条数据，具体条数视数据列数决定。若待极限性能，每条insert插入的数据需拥有相同的一级分区号（后续分析型数据库提供SDK帮助客户达成此要求）。
- 每购买一个c1/c8 ECU，客户端可增加20-30个并发连接插入数据。

```
Insert Into...Select From
```

分析型数据库0.9版本后，支持insert into ... select from语句。

```
INSERT [IGNORE]
[INTO]  db_name.tbl_name [(col_name,...)]
SELECT ...
;
```

其中tbl_name代表的表必须是实时写入表。select语句语法和 创建和修改表组 中描述一致。插入数据扔遵循无法立即可见和最终一致性原则。

28.8.4 多计算引擎和Hint

本节内容适用于分析型数据库的0.9.5或以上版本。

文中提及的Full MPP Mode，若您的集群或数据库没有开通该功能，请联系运维管理员或DBA开通。 多计算引擎在0.9.5或更新的分析型数据库版本中，拥有以下两套计算引擎：

- COMPUTENODE Local/Merge（简称LM）：先前版本的旧计算引擎，优点是计算性能很好，并发能力强，缺点是对部分跨一级分区列的计算支持较差。
- Full MPP Mode（简称MPP）：0.9.5版本新增的计算引起，优点是计算功能全面，对跨一级分区列的计算有良好的支持，可以通过全部TPC-H查询测试用例（22个），和60%以上的TPC-DS查询测试用例。缺点是计算性能相对LM引擎较差，并且计算并发能力相对很差。

在开启Full MPP Mode引擎功能的数据库中，分析型数据库会自动对查询Query进行路由，将LM引擎不支持的查询路由给MPP引擎，尽可能兼顾性能和通用性，目前以下几种情况会自动路由到Full MPP Mode：

- 特定函数：如row_number over等，LM模式不识别，捕获异常。
- Join类：
  - 事实表join事实表，join key全部在非分区列上。
  - 不同表组的事实表join事实表。
  - 维度表在前，left join事实表。
  - 事实表right join维度表（同上）。
  - 事实表join事实表，一级分区数不同。
- Group By、Order By、Having类：
  - Group By仅含非分区列，外层套子查询。
  - Group By仅含非分区列，包括Order By。
  - Group By仅含非分区列，包括Having。
- UNION/INTERSECT/MINUS不含分区列。
- SELECT复杂表达式，如SUM/SUM，任何带聚合函数的计算表达式等。
- COUNT DISTINCT或DISTINCT非分区列。

但是，在一些情况下，自动路由并非最合适于用户使用，用户可以自行判断自己的Query合适哪一种计算引擎，使用Hint强制将查询路由到某一个计算引擎中：

```
--强制使用COMPUTENODE LM计算引擎
/*+engine=COMPUTENODE*/ select * from ....
--强制使用Full MPP Mode计算引擎
/*+engine=MPP*/ select * from ....
```

Join的小表广播

在0.8.35版本后，分析型数据库支持小表广播模式进行一种特殊的高性能关联：虽然不符合关联四原则中分区、列等值或分区数一致的原则，但是关联的左表（或子查询） / 右表（或子查询）中有

一个表（或子查询）很小（一般不超过3 万行，5 列，512 KB），则可以通过小表广播模式快速关联，示例如下：

```
/*+engine=COMPUTENODE*/ select A.a, B.b from (select/*+broadcast=true*/ a from B where c=100) A join B on A. a = B.a;
```

其中A子查询数据量较小，是被广播的表，B表数据量无限制。这种Join性能也是较好的。

28.9 Data Pipeline

28.9.1 导入数据

分析型数据库目前支持从MaxCompute（原ODPS）中导入数据，OSS、RDS等数据源的导入可以通过阿里云数据集成产品进行。

背景信息

将数据导入分析型数据库有两种方法：直接使用SQL命令进行导入，或通过*DMS界面*进行导入。下文将主要详细介绍如何通过SQL命令导入数据。

操作步骤

1. 准备MaxCompute表或分区。创建好数据源头的MaxCompute表（可以是分区表或非分区表，目前要求Analytic DB中的字段名称和MaxCompute表中的字段名称一致），并在表中准备好要导入的数据。

   说明: 首次导入一个新的MaxCompute表时，需在各自账号的MaxCompute中给该表授予Describe和Select权限。

2. 通过SQL命令导入数据。

   SQL语法如下所示：

   ```
   LOAD DATA FROM 'sourcepath' [IN VERSION dataVersion] [OVERWRITE] INTO TABLE tablename [PARTITION (partition_name,...)]
   ```

   相关参数说明如下：

   - 如果使用MaxCompute数据源，则sourcepath为：

     ```
     odps://<project>/<table>/[<partition-1=v1>/.../<partition-n=vn>]
     ```

     - MaxCompute项目名称project；
     - MaxCompute表名table；
     - MaxCompute分区partition-n=vn，可以是多级任意类型分区。

- dataVersion：数据版本，是分析型数据库管理一次导入数据用的批次号，可用于未来可能开放的数据回滚等，通常情况下可以不填写。
  - 数据版本必须是long型，且取值在表上单调递增。
  - 可选，如果不指定则取当前时间（秒），格式是yyyyMMddHHmss ，如20140812080000。
- 分析型数据库表（分区）：
  - 格式 <table_schema>.<table_name> ，其中table_name是分析型数据库表名，table_schema是表所属DB 名。
  - 不区分大小写。
  - 分区格式为：partition_column=partition_value ，其中partition_column是分区列名，partition_value是分区值。
  - 分区值必须是long型。
  - 分区值不存在时表示动态分区，例如第一级hash分区。
  - 目前最多支持二级分区。
- OVERWRITE ：覆盖导入选项，导入时，如果指定数据日期的表在分析型数据库线上已存在，则返回异常，除非显示指定覆盖。
- 返回值：

  ```
  '\<jobId\>'
  ```

  任务ID（ jobId ），用于后续查询导入状态。
  - 任务ID，唯一标识该导入任务；
  - 字符串，最长256字节。

28.9.2 数据导入状态查询

数据导入命令发送后，数据并不会立刻导入到分析型数据库中，而是会在后台进行数据的导入工作。您可以通过使用SQL命令或在DMS中进行查询数据导入状态。高级用户也可以直接在information_schema中查询全部数据导入的信息。

查询数据的导入状态有多种方法：

- 通过 SQL 语句查询。

  通过 SQL 命令查询适合提交导入没有完成或完成不久的情况。

SQL 语法：

```
select state from information_schema.current_job where job_id = '\<jobid\>'
```

返回值：

```
'\<jobState>\'
```

返回值表示该任务状态（ jobState ）：

- NEW：初始状态。
- INITED：排队中。
- RUNNING：正在导入。
- SUCCEEDED：导入成功。
- FAILED：导入失败。
- ERROR：导入出错（系统内部错误）查询导入状态。
- 任务不存在，则返回空。
- 已完成的任务会被会被定时清理转为历史任务。

- 通过DMS查询数据导入状态。

  在DMS中，单击菜单上的导入导出->导入状态，即可查询每天的导入任务情况，并且通过多个维度进行筛选浏览。

- 通过Meta DB进行查询。

28.9.3 海量数据导出

背景信息

分析型数据库的查询模式适合在海量数据中进行分析计算后输出适量数据，若需要输出的数据量达到一定规模，分析型数据库提供数据导出（DUMP）的方式。

通过类DML语句导出到MaxCompute

- **前提须知**
  - 分析型数据库通过一个固定云账号进行数据导出到MaxCompute（与数据从MaxCompute导入到AnalyticDB情况类似）。

    说明: 各个专有云的导出账号名参照专有云的相关配置文档，一般为test1000000009@aliyun.com（与导入账号一致）。

- 需要给导入账号授予目标MaxCompute项目的createInstance权限，以及目标表的describe、select、alter、update、drop权限。授权命令如下：

```
--注意正确输入需要授权的表命名、project和正确的云账号
USE prj_name；--表所属MaxCompute project
ADD USER ALIYUN$xxxx@aliyun.com;
GRANT createInstance ON project prj_name TO USER ALIYUN$xxxx@aliyun.com;
GRANT Describe,Select,alter,update,drop ON TABLE table_name TO USER ALIYUN$xxxx@aliyun.com;
```

- **导出命令**

  类似于普通的SQL查询语句，您也可通过类似于DML语句进行数据导出。语法格式如下：

```
DUMP DATA
[OVERWRITE] INTO 'odps://project_name/table_name'
SELECT C1, C2 FROM DB1.TABLE1 WHERE C1 = 'xxxx' LIMIT N
```

通过类DML语句导出到OSS

导出到OSS时，需要将目标OSS设置为公共读写，或持有对该oss bucket有写权限的AK（主账号或子账号的AK均可）。语法格式如下：

```
DUMP DATA [options (accesskey_id='xxxxx' accesskey_secret='xxxxx')]
 [OVERWRITE] INTO 'oss://[<endpoint_domain>/]bulket_name/filename'
 SELECT C1, C2 FROM DB1.TABLE1 WHERE C1 = 'xxxx' LIMIT N
```

说明如下：

- 若endpoint_domain不传，则使用和分析型数据库同在一个region的OSS ENDPOINT，否则以用户填写的为准。
- 若oss bucket为公共读写，则options不填。
- 若为私有读写，则options中填入有权限的AK（主账号或子账号）。

关于返回数据行数

导出方式对海量数据的计算输出具有良好的性能（百万行数据导出在数百毫秒数据级），但是，对于数据精确度有一定牺牲，即实际返回的数据行数，可能是不完全精确。以限制导出行数为 1000为例（LIMIT 1000）：

- 实际数据行数可能稍大于1000，例如此时有120个数据分片，则等同于每个分区明确指定LIMIT 9，最多可以返回1080。
- 实际数据行数可能稍小于1000，如果符合条件的行数的总数小于1000。

- 实际数据行数可能稍小于1000，如果数据分片很均匀，例如此时有120个数据分片，如果某些分片返回数据行小于9的话，则等同于每个分区明确指定LIMIT 9。

28.10 用户与权限

28.10.1 用户类型与用户管理

分析型数据库用户建立的数据库属于该用户，用户也可以授权给其他用户访问其数据库下的表。

- **用户类型**
  - 数据库拥有者：必须开通了分析型数据库服务，同时是数据库的创建者。
  - 用户：被授权的数据库用户，由数据库拥有者授权时添加，无需开通分析型数据库服务。
- **添加用户的SQL语法如下所示**：

```
ADD USER user ON db_name.*
```

返回值：

空

添加用户：

- 创建数据库时，数据库的创建者即做为拥有者添加。
- 用户在第一次被授权一个数据库中的任何权限时，分析型数据库会自动添加为该数据库的用户。
- 主动添加到数据库的用户初始时没有任何权限。

- **查看用户列表的SQL语法如下所示**：

```
LIST USERS ON db_name.*
```

返回值：

```
\<user_id\> \<user_id\> ...
```

查看用户列表：只有数据库的拥有者可以查看该数据库的所有用户。

28.10.2 分析型数据库权限模型

分析型数据库支持基于数据库表的层级权限管理模型，提供类似MySQL的ACL授权模式。一个ACL授权由被授权的用户、授权对象和授予的对象权限组成。和MySQL略有不同的是，分析型数据库目前不支持针对用户在host上授权。

分析型数据库的权限对象和各对象权限

- Database：即db_name.* 或 *（默认数据库），指定数据库或数据库上所有表/表组。
- Table：即db_name.table_name或table_name ，指特定表。
- TableGroup：即db_name.table_group_name或table_group_name ，指特定表组。
- Column：语法上由column_list和Table组成，指定表的特定列。
- 聚合： Database -> Table[Group] -> Column，即每个权限级别能聚合其下面级别的所有权限。
- 在Database级别上，某个权限实际可能包含多个权限，例如GRANT CREATE ON *.* 同时包含创建数据库和创建表的权限。

表 102: 分析型数据库的权限对象说明

参数	Databas	Table 或 TableGrou	Column	说明
SELECT	+	+	+	查询数据
LOAD DATA	+	+	-	导入表（分区）数据
DUMP DATA	+	+	-	导出表（分区）数据
DESCRIBE	+	+	-	查看数据库、表/表组信息（Global、Database）、查看表/表组信息（Table[Group]）
SHOW	+	+	-	列出数据库、表/表组内部对象（Global、Database）、列出表内部对象（Table[Group]）
ALTER	+	+	-	修改表/表组定义
DROP	+	+	-	删除数据库、表/表组或分区（Global、Database）、删除表/表组或分区（Table[Group]）
CREATE	+	-	-	创建数据库、表/表组或分区（Global）、创建表/表组（Database）
INSERT	+	+	-	执行Insert的权限
DELETE	+	+	-	执行Delete的权限
ALL [PRIVILEGES]	+	+	+	以上所有权限

查询数据的权限

- 查询表数据待SELECT权限，最小级别是列。
- 并非所有查询都需要该权限，例如SELECT now()。

导出数据的权限

- 导出数据同时需要DUMP DATA和SELECT权限。
- 同时需要数据导出目的地的数据写入相关权限。

28.10.3 使用ACL进行权限管理

前文提到，类似MySQL，分析型数据库中的数据库创建者可以使用GRANT/REVOKE语句进行授权和权限回收。

进行授权的SQL语法，如下所示：

```
GRANT privilege_type [(column_list)] [, privilege_type [(column_list)]] ... ON [object_type] privilege_level TO user [, user] ...
```

返回值：

空

被授权用户：

```
user: 'user_name'[@'host_name']
```

- user_name和host_name必须使用单引号或双引号，如 'ALIYUN$test-user@aliyun.com'@'%'。
- 目前，host_name只支持 % ，即不支持指定Host。
- 可以写成user_name ，等价于'user_name'@'%'。

权限：

privilege_type为具体的权限类型：

[(column_list)] 为可选，当对象级别为表时，这里可以填写列的列表，进行针对具体列的授权。

[object_type] 为可选，标明权限对象的类型，如 Database、Table、TableGroup 等，建议填写 privilege_level 为被授权对象的表达式，填写方法见请参见分析型数据库权限模型 。

示例如下：

```
GRANT describe, select ON tablegroup db_name.table_group_name TO user 'ALIYUN$test_user@aliyun.com'@'%';
```

```
GRANT describe, select (col1, col2) ON table db_name.table name TO user 'ALIYUN$test_user@aliyun.com';
```

28.10.3.1 权限回收

SQL语法如下所示：

```
REVOKE privilege_type [(column_list)] [, privilege_type [(column_list)]] ... ON [object_type] privilege_level FROM user [, user] ...
```

返回值

空

说明: 在语法上与授权语句基本一致。

28.10.3.2 查看用户权限

SQL语法如下所示：

```
SHOW GRANTS [FOR user] ON [object_type] privilege_level
```

返回值

```
'GRANT ALL ON db_name.* TO user' 'GRANT SELECT(column_name) ON db_name.table_name TO user'
```

查看用户权限

不指定用户或是指定用户即当前用户，则列出当前用户在指定数据库上被授予的权限。否则，列出当前用户在指定数据库上授予指定用户的权限。

28.10.3.3 其他相关内容

- **权限的默认约定如下**：
  - 数据库创建者拥有在该数据库上的所有权限。
  - 表（组）创建者拥有在该表（组）上的所有权限。
  - 其他数据库用户拥有被各数据库拥有者授予的权限。

- SHOW DATABASES 语句不进行权限检查，可在不指定数据库的前提下，会返回用户所有拥有权限的数据库列表。
- 创建数据库时不检查权限，但用户需确保已开通分析型数据库服务。

- **授权/回收权限的权限**
  - 数据库拥有者（即创建者）在该数据库上执行Database及其级别以下的授权/回收权限，如SELECT ON db_name.*。
  - 其他数据库用户目前无法执行授权/回收权限操作。

- **设置ip白名单**

  目前分析型数据库支持设置DB粒度的ip访问白名单。

28.11 性能优化和诊断

28.11.1 使用执行计划调优

在进行一个业务SQL的性能调优，或是查看一个业务SQL的计算成本消耗时，可以使用explain命令来进行查看。DMS for分析型数据库中提供图形化的执行计划命令，同时，您也可以手动使用explain指令进行查看。

28.11.1.1 explain语句

分析型数据库支持通过explain语句来查看逻辑计划和物理执行计划，当用户发起一个explain查询到分析型数据库系统后，分析型数据库会抽样一个数据分区来分析执行计划，并以图形方式展现给用户。

explain语句的格式为：explain + select语句，示例如下：

- **简单sql的explain语法如下所示**：

```
explain select count(*) from test4dmp.test where id > 0
```

- **复杂sql的explain语法如下所示**：

```
explain
select student.id, count(*) from  test4dmp.student
inner join test4dmp.grade on student.id =grade.sid inner join test4dmp.elective on elective.id=student.id group by student.id
having count(*) > 2 order by student.id limit 10
```

28.11.1.2 返回文本格式

当用户通过查询的方式，想要获取文本格式的explain语句后，将会得到如下的json串，返回json格式：

| EXPLAIN |

| logical json |

| physical json

返回格式说明：一共会返回2行1列并且列名为EXPLAIN的ResultSet记录。其中第一行为逻辑计划，第二行为物理计划。

JSON格式说明如下：

- Node代表唯一的子节点。
- LeftNode代表左子树。
- RightNode代表右子树。
- MiddleNode代表所有的中间子树（多叉执行树），可以有多个MiddleNode。
- 其余key-value对统一在同层级的Node和*Node节点显示。

28.11.1.3 Explain逻辑计划详细语义

- **sql示例如下所示**：

```
explain
select student.id, count(*) from  test4dmp.student
inner join test4dmp.grade on student.id =grade.sid inner join test4dmp.elective on
 elective.id=student.id group by student.id
having count(*) > 2 order by student.id
limit 10
```

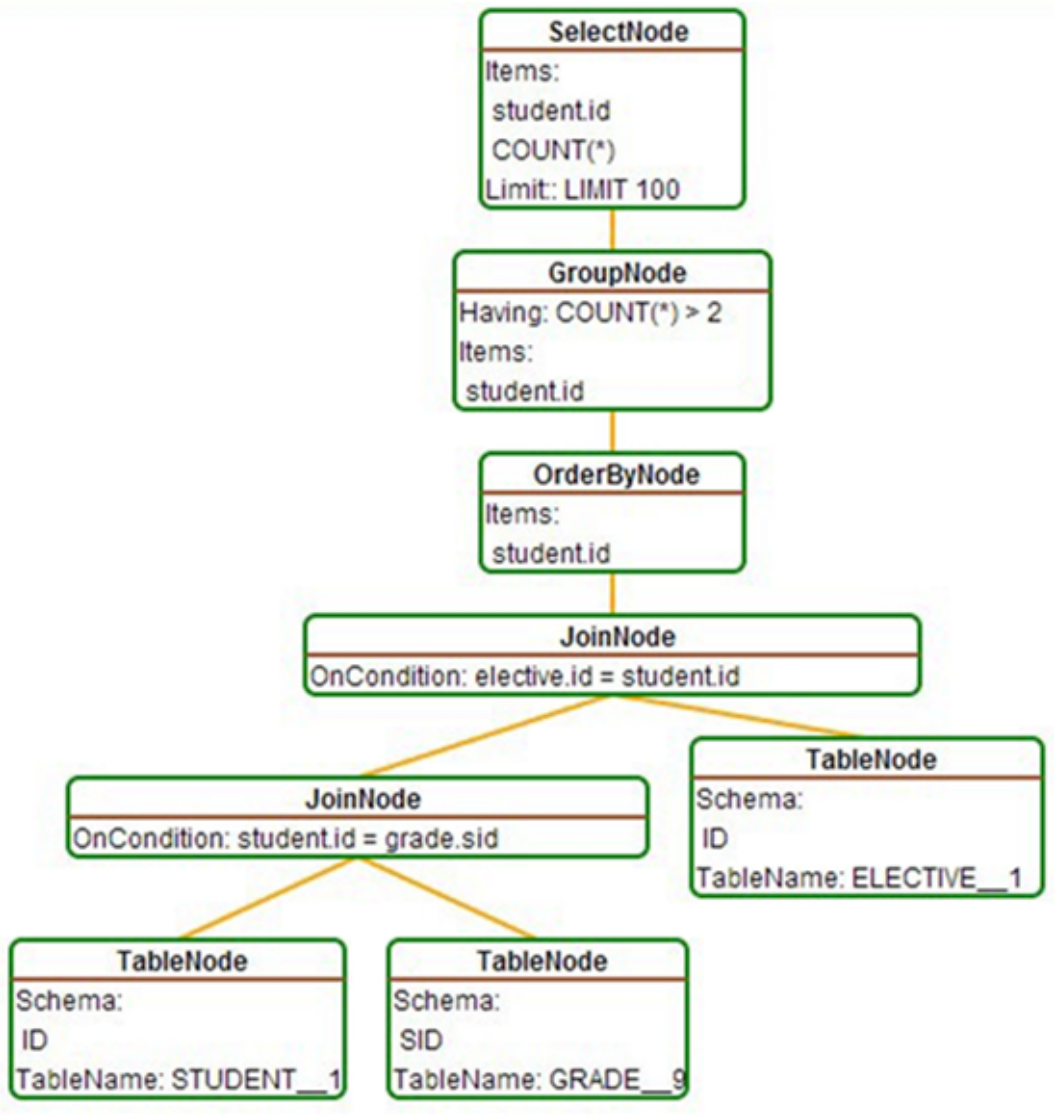
- **逻辑计划的explain** string：

```
{
"Items": ["student.id", "COUNT(*)"], "Name":  "SelectNode",
"Node": {
"Having": "COUNT(*) > 2",
"Items":  ["student.id"], "Name":  "GroupNode", "Node": {
"Items":  ["student.id"], "Name":  "OrderByNode", "Node": {
"Name": "JoinNode", "LeftNode": {
"Name": "JoinNode",

"LeftNode": {
"Name":  "TableNode",
"Schema": ["ID"], "TableName": "STUDENT  1"
},
"OnCondition": "student.id = grade.sid", "RightNode": {
"Name":  "TableNode",
"Schema": ["SID"],
```

```
"TableName": "GRADE  9"
}
},
"OnCondition": "elective.id = student.id", "RightNode": {
"Name":  "TableNode",
"Schema": ["ID"], "TableName": "ELECTIVE  1"
}
}
}
},
"Limit:": "LIMIT 100"
}
```

逻辑计划explain string的图形化展示效果：



逻辑计划各个节点说明：

- SelectNode表示这select中最终输出表达式的相关信息，例如select要输出的表达式集合。
- GroupNode表示GroupBy语句的相关信息，例如groupby的列，having的表达式等。
- OrderByNode表示OrderBy的列信息，例如列名，顺序等。

- JoinNode表示逻辑Join树的信息，例如join的on条件。
- TableNode表示分区表的信息，例如参与计算的列，表名等。

28.11.1.4 Explain物理执行计划详细语义

- **sql示例如下**：

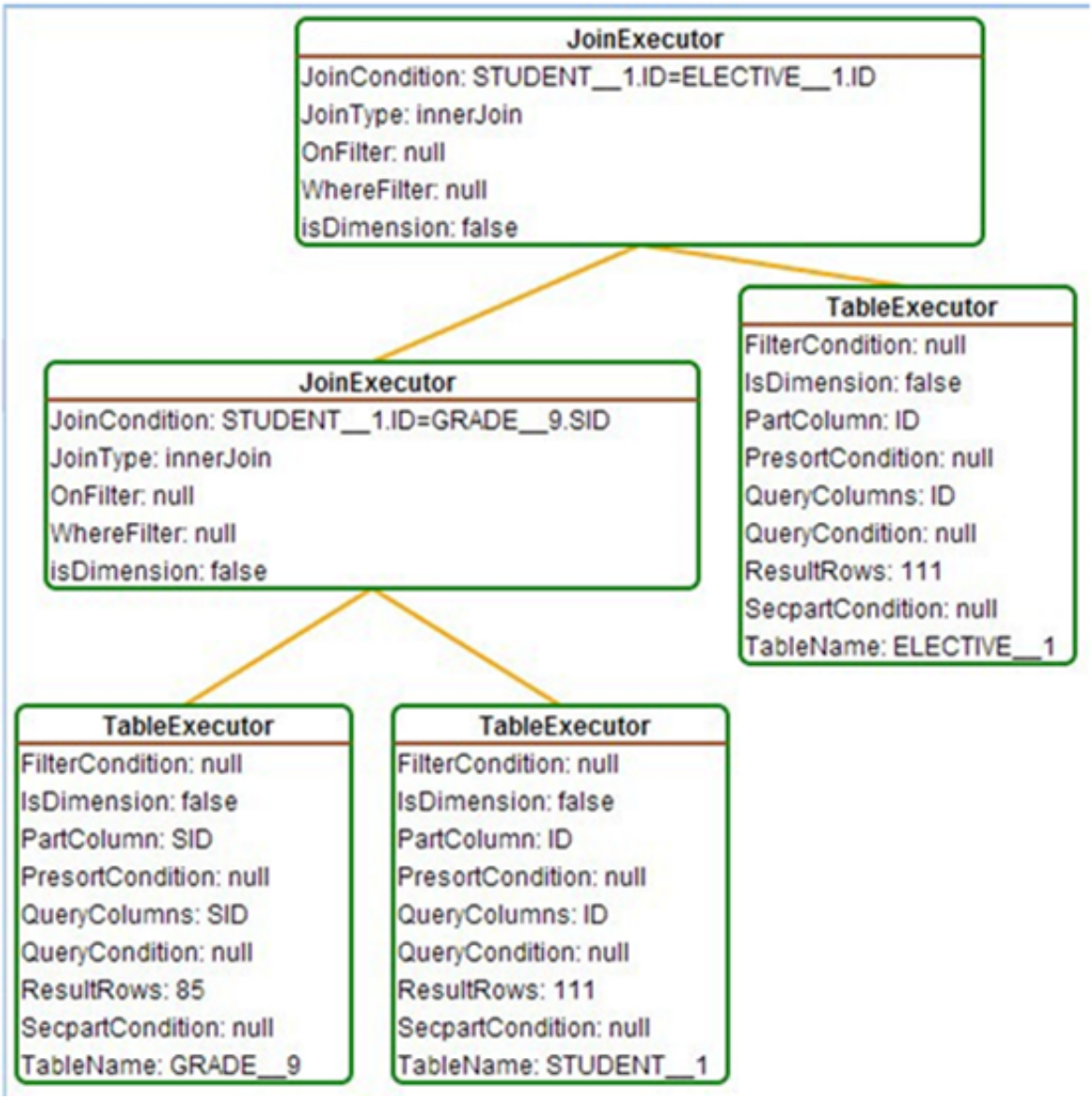
```
explain
select student.id, count(*) from  test4dmp.student
inner join test4dmp.grade on student.id =grade.sid inner join test4dmp.elective on
 elective.id=student.id group by student.id
having count(*) > 2 order by student.id
limit 10
```

- **物理计划的explain** string：

```
{
"Name":  "JoinExecutor", "isDimension":  "false", "LeftNode": {
"Name": "TableExecutor", "PresortCondition": "null", "SecpartCondition":  "null",
"QueryColumns": "ID, BOOLEAN_TEST, LONG_TEST",
"FilterCondition":  "null",  "ResultRows":  0,  "PartColumn":  "ID",  "IsDimension":  "false",
 "TableName": "TEST  2", "QueryCondition": "(ID = 0)"
},
"JoinCondition": "TEST  2.ID=TEST  2.ID", "RightNode": {
"Name":  "TableExecutor",  "PresortCondition":  "null",  "SecpartCondition":  "null",
 "QueryColumns": "ID, BOOLEAN_TEST", "FilterCondition":  "null",
"ResultRows": 0, "PartColumn": "ID", "IsDimension":  "false", "TableName": "TEST  2",
"QueryCondition": "((INT_TEST < 0) AND (ID = 0))"
},
"JoinType":  "innerJoin",
"WhereFilter": "(BOOLEAN_TEST = BOOLEAN_TEST)",
"OnFilter": "null"
}
```

- **物理计划explain string的图形化展示效果**：

图 974: 图形化效果



物理计划各个节点说明：

- JoinExecutor：表示Join的节点。
- JoinCondition：Join的条件。
- JoinType：Join的类型，innerJoin，leftJoin， rightJoin， simiJoin等。
- OnFilter：on上面的过滤条件。
- WhereFilter：where里面的过滤条件（只有leftJoin中on和where的条件才有差异）。
- IsDimension：该Join子树中是否含有维度表。
- TableExecutor：表示参与计算的表信息。
- FilterCondition：该表参与过滤计算的表达上（不能下沉索引的表达式）。
- IsDimension：该表是否是维度表。

- PartColumn：一级分区列。
- SecPartColumn：二级分区列。
- QueryColumns：参与计算的列数（下沉索引计算的列不包含在此之内）。
- QueryCondition：索引条件。
- ResultRows：该节点预估cost。
- PresortCondition：预排序条件，可以用做优先索引条件下沉。
- SecpartCondition：二级分区筛选条件，可以用做二级分区筛选。
- TableName：分区表名。

28.11.2 查看DB性能指标和报告

在日常的管理中，了解数据库的各项性能指标是非常重要的一个环节。分析型数据库提供实时的性能指标以及定时产生的性能报告，可以通过查询Meta DB的方式获取，也可以在DMS界面中查看图形化的报告。

在Meta DB中，DB的性能指标目前主要是存储在performance_schema中。实时性能指标存储在 *performance_schema.minute_db_profile* 表中。

由于分析型数据库是一个分布式系统，所以minute_db_profile中存储了每一个前端服务节点记录的信息。要查询准确的DB层级汇总信息，需要先在performance_schema.servers中查询一个数据库当前在线的前端服务节点ID。可使用如下SQL查询：

```
select distinct online_server_id from performance_schema.servers where table_schema = 'db_name';
```

其中db_name为数据库名，之后再用上述SQL返回的online_server_id列表查询minute_db_profile表获取数据库30s粒度的实时信息：

```
select update_time, qps, pv, avg_rt, data_size from performance_schema.minute_db_profile where server_id in ('4d8b2019a04df41acaf83b5101d64e5e', 'a1408c344ebc134587f684d586703ede') and table_schema = 'db_name';
```

其中db_name为数据库名，server_id的内容为servers表中返回的所有online_server_id。这条SQL会返回多条记录，原则上每个server_id一条，每条记录的update_time是一个四个数字组成的字符串，标识着这条记录的更新时间，格式为'MMSS'，原则上30秒产生一条记录。

而查询定时产生的（目前是每个小时一次，后续可能更改）性能报告则要简单的多。直接查询*performance_schema.hour_db_profile* 表即可。查询这个表时亦必须传入table_schema参数（即

数据库名）。该表的thedate用'YYYYMMDD'的格式标明报告产生的日期，hour字段用'hh'的格式标明报告产生的时间。目前最多保留最近七天的报告。

另外，performance_schema.hour_slow_query表中提供了和性能报告同样周期内的最常见的最慢查询，查询方式和hour_db_profile一致。

若不想通过SQL查询数据库的性能信息，分析型数据库在DMS中提供了图形化的性能报告。30s内更新的实时性能指标可在DMS的数据库首页中直接看到。而定时的性能报告可在DMS的数据库首页中单击性能诊断报告按钮，进入一个列表页，然后选择一个报告产生的时间即可查看图形化的性能报告和最慢查询列表。

28.12 在生产中使用分析型数据库

28.12.1 业务系统连接并进行查询

使用jdbc odbc php python R-Mysql等连接分析型数据库的例子和注意事项介绍，以及流控、retry链接、异常处理等方法。

分析型数据库集群系统部署在阿里云环境中，您在部署业务应用系统时，尽可能保证业务系统与阿里云环境的网络联通性，如创建阿里云ECS主机作为业务系统的服务器。

JDBC直接连接分析型数据库最简单直接的连接并访问分析型数据库的方式是通过JDBC，分析型数据库支持MySQL自带的客户端以及大部分版本的mysql-jdbc驱动。

支持的mysql jdbc驱动版本号如下所示：

- **5.0系列**：5.0.2，5.0.3，5.0.4，5.0.5，5.0.7，5.0.8。
- **5.1系列**：5.1.1，5.1.2，5.1.3，5.1.4，5.1.5，5.1.6，5.1.7，5.1.8，5.1.11，5.1.12，5.1.13，5.1.14，5.1.15，5.1.16，5.1.17，5.1.18，5.1.19，5.1.20，5.1.21，5.1.22，5.1.23，5.1.24，5.1.25，5.1.21.27，5.1.28，5.1.29，5.1.31。
- **5.4系列**。
- **5.5系列**。

Java程序中，将合适的mysql-jdbc驱动包（mysql-connector-java-x.x.x.jar）加入CLASSPATH中，通过以下示例程序就能连接并访问分析型数据库。通过该JDBC方式直连分析型数据库时，和直连MySQL类似，需注意在使用完连接并不准备进行复用的情况下，释放连接资源。

您根据具体情况，设置${url},${my_access_key_id},${my_access_key_secret}和${query}值：

```
Connection connection = null; Statement statement = null; ResultSet rs = null;
try {
```

```
Class.forName("com.mysql.jdbc.Driver");
String url = "jdbc:mysql://mydbname-xxxx.ads- hz.aliyuncs.com:5544/my_ads_db?
useUnicode=true&characterEncoding=UTF-8";
Properties connectionProps = new Properties(); connectionProps.put("user",
 "my_access_key_id"); connectionProps.put("password", "my_access_key_secret");

connection = DriverManager.getConnection(url, connectionProps); statement =
 connection.createStatement();
String query = "select count(*) from information_schema.tables"; rs =
 statement.executeQuery(query);
while (rs.next()) { System.out.println(rs.getObject(1));
}
} catch (ClassNotFoundException e) { e.printStackTrace();
} catch (SQLException e) { e.printStackTrace();
} catch (Exception e) {
e.printStackTrace();
} finally {
if (rs != null) { try {
rs.close();
} catch (SQLException e) { e.printStackTrace();
}
}
if (statement != null) { try {
statement.close();
} catch (SQLException e) { e.printStackTrace();
}
}
if (connection != null) { try {
connection.close();
} catch (SQLException e) { e.printStackTrace();
}
}
}
```

通过JDBC连接池连接分析型数据库

通过连接池Druid连接分析型数据库的具体操作，请参见在应用中连接和使用分析型数据库 。

通过PHP连接分析型数据库的具体操作，请参见在应用中连接和使用分析型数据库。

关于负载均衡

您在 创建数据库时，分析型数据库会根据您指定的服务参数，为您在分析型数据库集群中分配特定数量的前端服务机，这些机器对您透明，您得到的是访问该数据库的URL（例如：mydbname-xxxx.ads-hz.aliyuncs.com:5544，在控制台中的连接信息中获取），分析型数据库集群自己使用阿里云负载均衡产品SLB（http://www.aliyun.com/product/slb ）来对访问请求进行负载均衡，您无需关心访问分析型数据库时的负载均衡策略。

若您的业务应用自己想采用一定的负载均衡方案，请参见 *http://www.aliyun.com/product/slb* 。

关于重试以及异常处理

网络环境下连接和使用分析型数据库，从业务应用的角度来看，使用适当的重试和异常处理机制是保证业务应用高可用性的手段之一。无论是通过JDBC直连、JDBC连接池、PHP，还是Python等方

式连接分析型数据库，出现连接连接异常时，应用均可采用适当的重试机制来保证连接可用性。在海量数据并发处理任务量大的情 况下，偶尔在查询过程中出现异常时，也可采用适当的重试机制来重发查询。

28.12.2 数据导入任务生产指南

在使用分析型数据库时，稳定的数据导入是非常重要的生产要素。一般新用户经常在进行首次的数据导入时因为操作不当无法成功，或成功后无法稳定运行。本文将介绍建立一个生产化的数据导入任务的注意事项。

数据的准备

若想稳定的导入数据，首先要在数据的源头稳定的产出数据。稳定的数据需要注意以下事项：

- 数据所在的项目名（对应源头为MaxCompute）/文件访问路径（对应源头为OSS，暂不支持）/服务器连接串（对应源头为RDS，暂不支持）和表名与LOAD DATA命令的中的源头一致并保持稳定。
- 数据表的字段名，在源头上与在分析型数据库上的配置一致，源头表可以比分析型数据库有更多的字段，但是不能比分析型数据库表缺少字段。
- 源头表进行导入的分区的数据不能为空，进行导入的数据主键不能有NULL值，HASH分区键不能存在大量NULL值或同样的HASH分区键的数据条数过多，例如超过了每个分区的平均数据量的三倍。否则不仅会对查询性能造成影响，在极端情况下也会导致数据导入时间过长或者失败。

数据的导入

在数据产出后，可以通过MySQL连接的方式或HTTP Rest-API的方式调用数据导入命令，注意事项如下：

- 调用命令时，命令所引用的源头表/分区的数据已经完整的产出完毕，若源头是MaxCompute/OSS，不再有任何在源头的写入操作。所以通常需要一个良好的离线任务调度系统（例如阿里云DPC中的数据开发平台）来进行相关的任务运行和调度。
- 调用命令时，要确保命令所引用的源头表/分区已经对相应账号授予足够的权限并未开启保护模式等阻止数据流出的安全策略。
- 调用命令时分析型数据库中该表没有正在运行的导入任务，否则会返回失败。

查询数据导入状态和解决导入中的问题在生产系统中查询数据导入状态，通常更多的是通过HTTP API进行的查询的，如果有一个较好的离线任务调度系统，那么实现难度并不大。

在数据导入的过程中，经常会因为出现各种错误而导致导入中断，具体的错误处理请参见附录三#常见错误码表。

28.12.3 数据模型设计

BI/Adhoc业务的数据模型设计在一个BI/Adhoc类OLAP业务中，数据的模型设计是非常重要的一点。下面，我们就基于一个企业商品库存分析的业务，来看一看在分析型数据库中如何进行数据模型的设计。

以下内容仅适用于 0.8 版本。

28.12.3.1 事实表

事实表规则

- 事实表与大表分区：事实表一般都比较大（行数>1亿或数据量>20G），按行水平切分为多个分区，即实现为分区表。
- 目前，分区数一般是100。
- 选定一个字段（后续将支持多个）做为一个分区列。
- 每行数据的分区列的hash值对分区数（100）取模，决定该行被切分到哪个分区。
- 分区列推荐是long型，也支持string类型，但不支持JOIN。
- 分区列首先决定于如何查询数据，因为与SQL查询能力紧密相关。
- 分区列的取值应该是离散的分区表与SQL查询。

分区表的SQL查询能力

- 分区表与任意维度条件过滤（WHERE）：
  - 可以支持任意字段条件过滤。
  - 包含OR操作的WHERE过滤表达式的性能会随条件的个数增加而较快降低。
- 分区表与JOIN：
  - 分区表之间的JOIN只能基于分区列（事实表与维度表之间，或维度表相互之间JOIN不受此限制）。
  - 分区表之间JOIN需要分区数完全相同。
  - 分区列如果是string类型，则JOIN性能会变差。
  - 综合上述概念，Garuda引入表组（即多个事实表按照相同切分规则的组合）。
- 分区表与Group-By、Order-By查询：

- Group-By条件包含分区列，则应该放到第一列，此时结果是精确的且查询性能很好。
- Group-By表达式如果不包含分区列（或第一列不是分区列），则分组数量在范围内（<5000），结果是精确的，但性能随分组数量增加而降低。
- Group-By表达式如果不包含分区列（或第一列不是分区列），且分组数量不在范围内（>=5000），此时结果是不精确的且性能随分组数量增加而降低。
- 如果Group-By表达式第一列不是按分区列，则Having语句不支持。
- 全局分组TOP（N）：Group-By表达式包含分区列，则应该放到第一列，同时包含Order-By表达式，如果分组数量在范围内（<1000），结果是精确的，但性能随分组数量增加而较大降低。
- 全局模糊分组TOP(N)：Group-By表达式第一列如果不是分区列且包含Order-By表达式，而且分组数量不在范围内（>=1000），此时结果是不精确的且性能随分组数量增加而较大降低。
- 全数据排序： Order-By表达式第一列如果不是分区列，性能会随排序列或表达式取值增加而较大降低。

- 分区表与子查询：
  - 子查询实现为UNION （ALL）各个分区内的查询结果，因此对于嵌套子查询或子查询加JOIN的最终结果与RDBMS是不一致的。
  - 子查询之间不支持JOIN 事实表与小表。

  事实表如果较小（行数<1亿或数据量<10G），可以不建成分区表。

28.12.3.2 维度表

维度表与小表大部分情况下，维度表是小表（<1000w且<1G），分析型数据库针对维度表的支持：

- 标识为维度表，且分区数自动设置为1，即不进行任何切分。
- 维度表单表的查询能力（如子查询、Group-By、Order-By等）不受限制。
- 多个维度表之间可以按任意字段JOIN。
- 维度表与事实表（分区表）也可以按任意字段JOIN维度表与分区表。
- 维度表如果是大表，则必须建成分区表，将会有一些限制（符合上述分区表相关规则）。

28.12.3.3 宽表

- 维度表存在多个层次时，通常建议打平为单张宽表，通过减少JOIN次数会提高查询性能。
- 维度退化：维度度表列数很少时可以直接打平到事实表，这样能避免事实表与维度表之间的JOIN，提高性能。

- 如果维度表字段很多，而且字段都会参与多维查询的条件过滤或分组排序，则不建议打平，因为可能会造成事实表极度膨胀，反而影响性能。
- 如果维度表因为自身很大必须建成分区表，则不仅不能打平到事实表，更会有一些限制（符合上述分区表相关规则）。

28.12.3.4 聚合表

- 包含事实数据表的汇总信息的表。
- 按照某个维度通过聚合操作（max、min、sum、count、avg）把事实列聚合，比如按照时间（月度等）。
- 聚合表的数据量会大量减少，可以不建成分区表。

28.12.3.5 实例分析

事实表

企业商品库存表（ent_items_stock_detail）

- 数据量20亿。
- 包含企业的商品库存值。
- 事实列仅一列即库存数（code_stock_count）。

维度表（直接）

以下维度字段直接关联 -> 企业信息表（ent_info）：

- ent_type_id string COMMENT '企业类型ID'
- ent_type_name string COMMENT '企业类型名称'
- ent_info_id string COMMENT '企业ID'
- ent_info_name string COMMENT '企业名称'
- region_id string COMMENT '地域ID'
- region_name string COMMENT '地域名称'

以下维度字段直接关联 -> 商品信息表（ent_net_detail）：

- item_info_id string COMMENT '商品信息ID'
- sale_batch_id string COMMENT '商品批次ID'
- produce_batch_no string COMMENT '生产批次号'
- produce_date string COMMENT '生产日期'

- exprie_date string COMMENT '有效期至'
- item_base_info_id string COMMENT '商品通用信息ID'
- item_base_info_name string COMMENT '商品基本信息名称'
- produce_ent_info_id string COMMENT '生产企业ID'
- produce_ent_info_name string COMMENT '生产企业名称'

以下维度字段直接关联 -> 地域信息表（region_info）：

- region_id string COMMENT '地域ID'
- region_name string COMMENT '地域名称'
- prov_id string COMMENT '省ID'
- prov_name string COMMENT '省名称'
- city_id string COMMENT '市ID'
- city_name string COMMENT '市名称'
- area_id string COMMENT '县ID'
- area_name string COMMENT '县名称'

维度表（层级）

以下维度字段通过商品信息表（ent_net_detail）间接关联 -> 企业信息表（ent_info）：

- produce_ent_info_id string COMMENT '生产企业ID'
- produce_ent_info_name string COMMENT '生产企业名称'
- region_detail string COMMENT '生产企业属性：region_detail'
- produce_region_id string COMMENT '生产企业所在地域ID'
- produce_region_name string COMMENT '生产企业所在地域名称'

以下维度字段通过商品信息表（无）间接关联>商品基本信息表（ent_base）：

- item_base_info_id string COMMENT '商品通用信息ID'
- item_base_info_name string COMMENT '商品基本信息名称'
- item_spec string COMMENT '商品通用信息属性：item_spec'
- approval_no_id string COMMENT '批准文号ID'

推荐方案

企业商品库存表（ent_items_stock_detail）

- 事实表，建成分区表。
- 分区数100。
- 分区列选企业ID（ent_info_id），需要先单独生成一个对应的long型唯一ID。
- 查询或子查询按照企业ID进行Group-By，Order-By时（做为第一列），查询能力没有限制且性能很好。
- 查询或子查询按照商品ID进行Group-By，Order-By时，先在WHERE过滤条件里指定商品ID或取值范围，否则不精确且性能较差。
- 查询或子查询按照商品其他层级维度（如省市区）时，维度取值较少，或先在WHERE过滤条件里该维度条件，否则不精确且性能较差。

商品信息表（**新建表** items_info）

- 维度表，数据量小，不需建分区表。
- 存在多个层级维度（商品基本信息），打平为单张宽维度表。
- 因为打平后存在列数很多，不建议进一步打平到企业商品库存表，因为会导致数据量膨胀很大。
- 商品信息表本身SQL查询能力不受限制。
- 商品信息表与企业商品库存表按商品信息ID（item_info_id）JOIN。
- 商品信息表支持任意字段条件过滤。

企业信息表（ent_info ）

- 维度表，数据量不大（300w），不需建分区表。
- 如果以后扩展太大（列数），可转换为分区表，分区列是企业ID（ent_info_id），可以与企业商品库存表JOIN。
- 目前列数较多，不建议打平到企业商品库存表。
- 企业信息表本身SQL查询能力不受限制（若转换为分区表后有相应限制）。
- 企业信息表与企业商品库存表按企业ID（ent_info_id）JOIN。
- 企业信息表支持任意字段条件过滤 。

地域信息表（ **region_info** ）

- 维度表，数据量不大（300w），不需要建分区表。
- 仅单个层级维度。
- 目前列数较多，不建议打平到企业商品库存表。
- 地域信息表与企业商品库存表按地域ID（region_id）JOIN。

- 地域信息表与商品信息表按生产企业ID（produce_ent_info_id）JOIN。
- 地域信息表支持任意字段条件过滤。

查询示例

-- 某种商品库存TOP（100）的企业：

```
select ent_info_id, count(*) as stock from  ent.ent_items_stock_detail
join  ent.items_info  on  ent_items_stock_detail.item_info_id  =  items_info.item_info_id  where
 items_info.item_base_info_name='xxx商品'
group by ent_info_id
order by stock desc limit 100
```

28.12.4 在BI工具中进行连接和使用

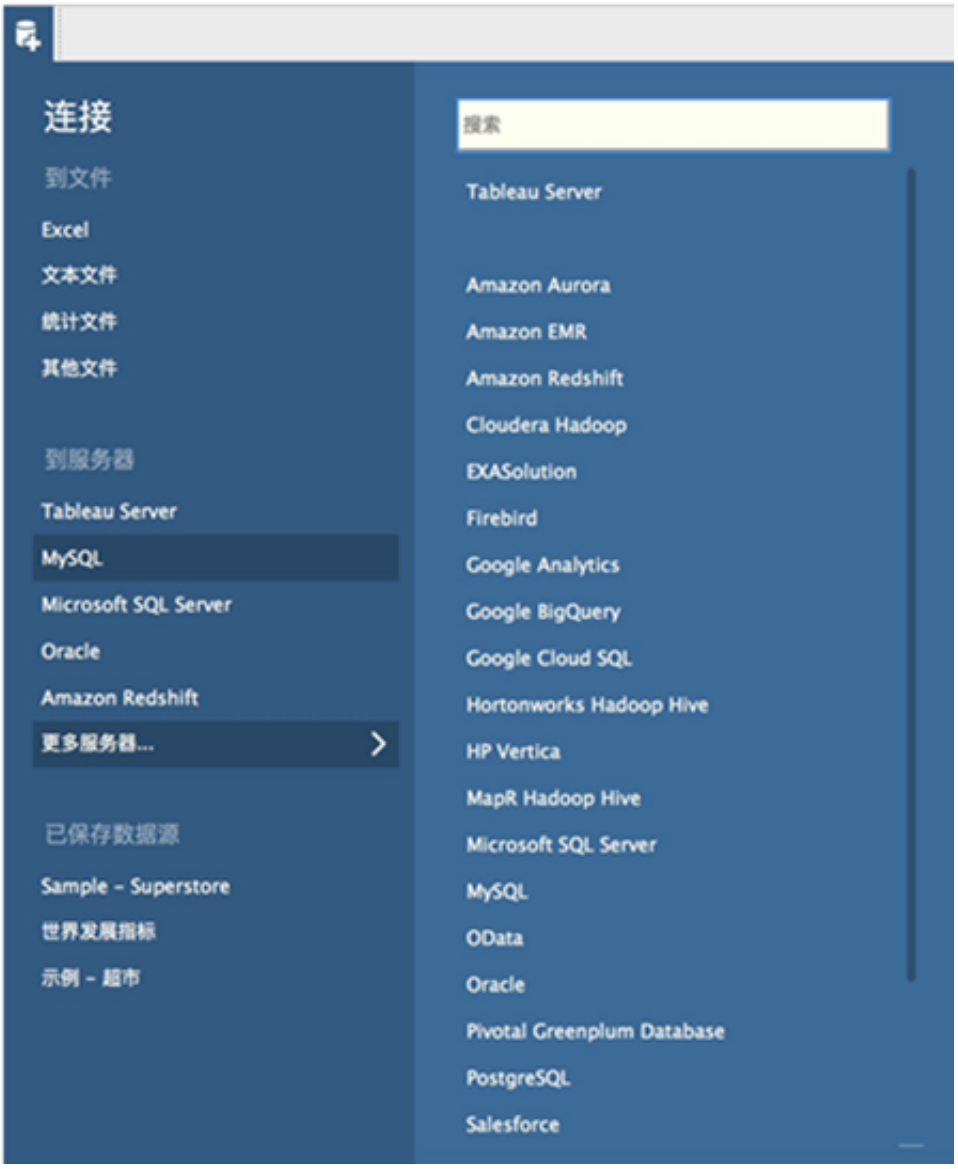
目前，阿里云分析型数据库能够支持在部分BI工具中以MySQL连接的方式进行连接和使用。

在使用这些BI工具连接分析型数据库时，通常需先安装好OBDC MySQL Driver，这里推荐使用MySQL Connector/ODBC 3.51版本（下载地址 *http://dev.mysql.com/downloads/connector/odbc/3.51.html*）。

在Tableau Desktop中使用分析型数据库

现在可以在Tableau Desktop 9.0及以上版本，通过MySQL连接的方式连接到阿里云分析型数据库读取和分析数据，产生可视化报表，大部分Tableau Desktop 9.x的功能可以在阿里云分析型数据库的连接中使用。

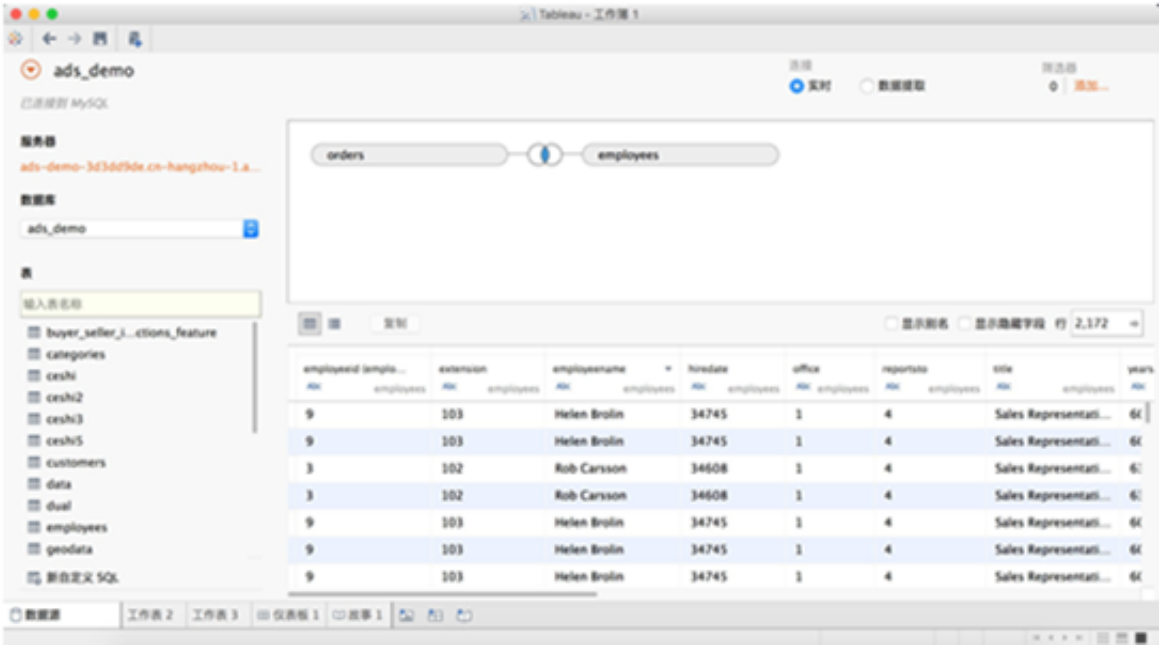
1. 打开Tableau Desktop，单击“添加新的数据源”按钮并选择数据源类型为MySQL。



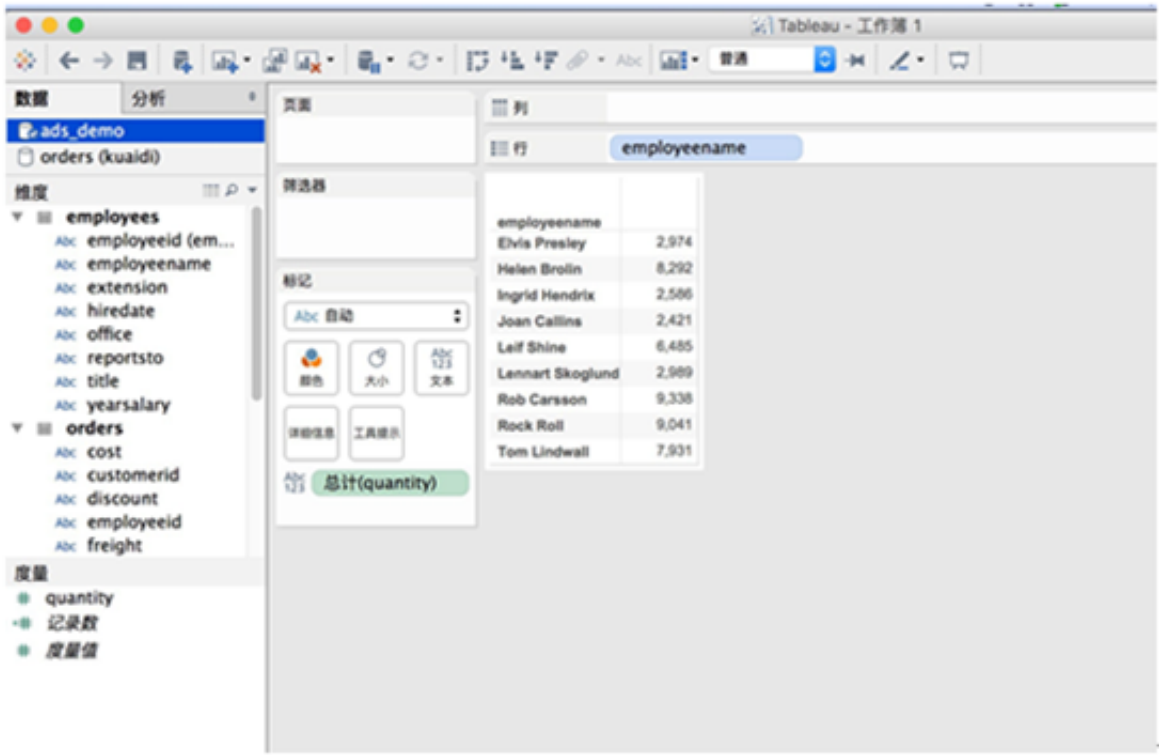
2. 在弹出的对话框中，输入您要访问的数据库的域名和端口号（在DMS的数据库选择页面获取），以及将您的账号在阿里云的Access Key ID / Access Key Secret输入到用户名/密码框中，单击**确定**。



3. 此时在Tableau的数据源页面上，可以看到您有权限的数据库，注意：这里只能选择您连接到的域名端口所对应的数据库。选择后可以进行数据表选取，以及数据预览。

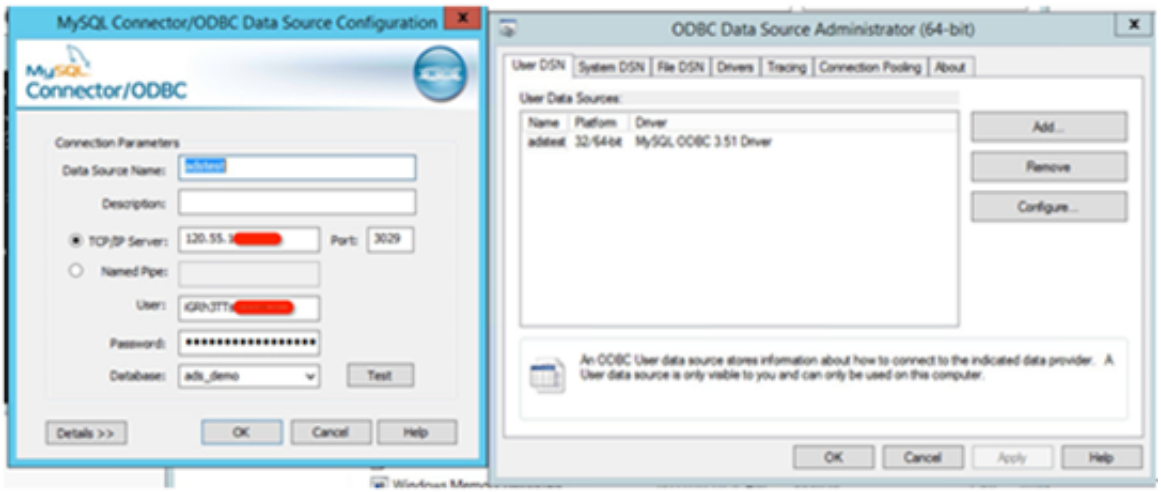


4. 完成上述操作即可进行工作表的建立以及更多其他功能。



在QlikView中使用分析型数据库

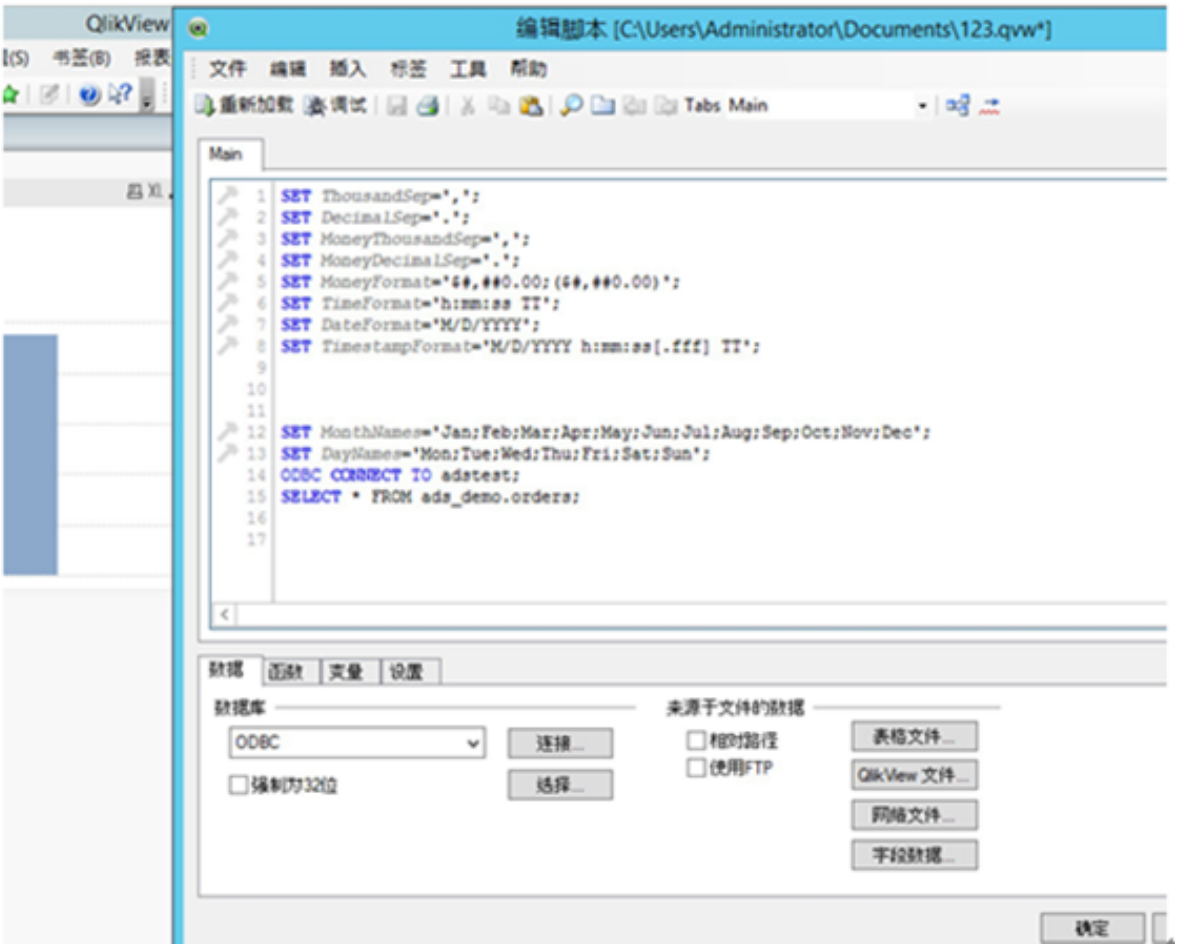
1. 要使用QlikView连接分析型数据库，首先需配置操作系统的ODBC数据源。安装OBDC MySQL Driver后，进入控制面板-管理工具-ODBC数据源（不同操作系统此步骤可能不同），新建一个用户DSN，选择“MySQL ODBC 5.xx Driver”。



2. 在ODBC　TCP/IP服务器处填入连接分析型数据库的域名和端口号（在DMS的数据库选择页面获取），部分情况下，可能需要您将域名解析为 IP 地址填入，以及将您的账号在阿里云的Access Key ID / Access Key Secret输入到用户名/密码框中，在数据库一项中选择您要连接的数据库，单击测试按钮测试连接通过后，选择 **确定**。
3. 打开QlikView（建议版本：11.20.x），打开**编辑脚本**，在末端键入：

```
ODBC CONNECT TO adstest;
```

```
select * from ads_demo.example_table limit 100;
```



其中adstest为之前在ODBC中建立的数据源名称。之后运行脚本，即可在QlikView中读取和使用分析型数据库中的数据。

在Microsoft Excel中读取分析型数据库中的数据 若是要在Excel中读取分析型数据库中的数据，建议安装微软的免费插件：Microsoft Power Query for Excel。

按照Qlikview一节中相同的方式配置好ODBC数据源之后，参照MySQL相关教程进行后续操作。

28.13 附录

28.13.1 附录一：元数据库数据字典

分析型数据库的元数据库分为记载性能相关信息的performance_schema和记载元数据的information_schema，并和MySQL的元数据库有一定的兼容性，但是不是100%一致。

查询元数据库可以直接在JDBC连接中使用SQL语句进行查询，如下所示：

```
SELECT state
FROM
information_schema.job_instances
```

```
JOIN
information_schema.table_data_loads
ON
job_instances.id=table_data_loads.job_instance_id
WHERE
table_data_loads.table_schema='db_name'
AND   table_data_loads.table_name='table_name'
AND  table_partition='partition'
ORDER BY table_data_loads.data_version DESC LIMIT 10
```

即可查询分析型数据库库的数据导入状态。

28.13.1.1 performance_schema库

- **HOUR_DB_PROFILE**

  小时粒度的db性能指标：

FIELD	dataType	comment
table_schema	String	db名称，主键
thedate	int	日期(yyyymmdd)，主键2
hour	int	小时（hh），主键3
total_pv	bigint	总pv
avg_rt	int	平均rtint
max_rt	int	最大rtint
zero_result	bigint	0结果数量bigint
time_out	bigint	超时数量bigint
cache_hit_rate	float	缓存命中率float
rt_dist	string	rt分布，k=v,k=v格式，string

- **HOUR_SLOW_QUERY**

  小时粒度的sql排行榜：

FIELD	dataType	comment
table_schema	string	主键1，数据库名
thedate	int	主键2，日期
hour	int	主键3小时
order_idx	int	int，排序顺序，主键4
sql	string	sql内容，string

FIELD	dataType	comment
rt	int	rt，int
pv	bigint	pv, bigint
last_query_time	string	最后查询时间，stringor date

- **MINUTE_DB_PROFILE**

  小时内的以分钟或30s刷新的db性能数据，仅保留最近一个小时的：

FIELD	dataType	comment
table_schema	string	数据库名
update_time	string	更新分钟和秒，用MMSS表示
server_id	string	服务节点的ip
qps	int	时间间隔内的qps
pv	int	时间间隔内的总pv
avg_rt	int	时间间隔内的平均rt
data_size	bigint	采样的db占用空间大小（单位：字节）

- **SERVERS**

  当前db在线的服务节点列表，即时更新：

FIELD	dataType	comment
table_schema	string	主键，数据库名
update_time	string	yyy-mm-ddhh:mm:ss 该条数据写入时间
online_server_id	string	服务节点的ip

28.13.1.2 information_schema库

- **SCHEMATA**

  SCHEMATA表提供了关于数据库的信息：

FIELD	TYPE	是否为扩展字段	COMMENT
CATALOG_NAME	varchar(512)	N	分析型数据库中返回NULL
SCHEMA_NAME	varchar(64)	N	数据库名称
DEFAULT_CHARACTER_SET_NAME	varchar(32)	N	字符集名称，分析型数据库中返回utf8
DEFAULT_COLLATION_NAME	varchar(32)	N	字符校验规则名称，分析型数据库中返回utf8_general_ci
SQL_PATH	varchar(512)	N	分析型数据库中返回null

- **TABLES**

  TABLES表提供数据库表信息。该部分数据包括表的元数据与部分表对应数据的元数据：

FIELD	TYPE	是否为扩展字段	COMMENT
TABLE_CATALOG	varchar(512)	N	NULL
TABLE_SCHEMA	varchar(64)	N	数据库名称
TABLE_NAME	varchar(64)	N	表名称
TABLE_GROUP_Name	varchar(64)	Y	表组名称
TABLE_TYPE	varchar(64)	N	分析型数据库中返回：meta表[, 跨mergenode]、分区表、维度表
ENGINE	varchar(64)	N	数据库引擎名称，分析型数据库中目前返回 ANALYSISTABLE
VERSION	bigint(21) unsigned	N	表的版本号，目前返回1
ROW_FORMAT	varchar(10)	N	分析型数据库中返回''
TABLE_ROWS	bigint(21) unsigned	N	表记录数
AVG_ROW_LENGTH	bigint(21) unsigned	N	存储在该表中的行的平均长度，in bytes
DATA_LENGTH	bigint(21) unsigned	N	表的总长度，以字节为单位

FIELD	TYPE	是否为扩展字段	COMMENT
MAX_DATA_LENGTH	bigint(21) unsigned	N	最大可以存储的字节数
INDEX_LENGTH	bigint(21) unsigned	N	index文件的长度，单位为byte
DATA_FREE	bigint(21) unsigned	N	总是返回0
AUTO_INCREMENT	bigint(21) unsigned	N	是否有自增列：1为是、0为否
CREATE_TIME	datetime	N	创建时间
UPDATE_TIME	datetime	N	更新时间
CHECK_TIME	datetime	N	总是返回NULL
TABLE_COLLATION	varchar(32)	N	表的校验规则名称。分析型数据库中返回utf8_general_ci
CHECKSUM	bigint(21) unsigned	N	表的checksum
CREATE_OPTIONS	varchar(255)	N	创建表时其他选项。以 key=value在，目前包含有引擎名称、表组名称、最小副本数PRESORT_ORDER等信息[key=value ...]形式存
TABLE_COAMMENT	varchar(80)	N	表注释

- **COLUMNS**

  提供所有表列信息：

FIELD	TYPE	是否为分析型数据库扩展字段	COMMENT
TABLE_CATALOG	varchar(512)	N	NULL
TABLE_SCHEMA	varchar(64)	N	数据库名称
TABLE_NAME	varchar(64)	N	表名称
COLUMN_NAME	varchar(64)	N	列名称
ORDINAL_POSITION	bigint(21)unsigned	N	列编号，从1开始

FIELD	TYPE	是否为分析型数据库扩展字段	COMMENT
COLUMN_DEFAULT	varchar(1024)	N	默认值
IS_NULLABLE	varchar(3)	N	是否可以为空，默认为''
IS_PRIMARYKEY	varchar(3)	N	是否为主键，默认为''
IS_AUTOINCREMENT	varchar(3)	N	是否为自增，默认为''
DATA_TYPE	varchar(64)	N	数据类型，比如varchar。默认为''
CHARACTER_MAXIMUM_LENGTH	bigint(21)unsigned	N	字符最大长度，以字符为单位。默认为NULL。如果是大对象类型，返回-1
CHARACTER_OCTET_LENGTH	bigint(21)unsigned	N	最大长度，以字节为单位，默认为NULL，对象列类型，返回-1
NUMERIC_PRECISION	bigint(21)unsigned	N	数据最大精度。分析型数据库中返回NULL
NUMERIC_SCALE	bigint(21)unsigned	N	小数位数分析型数据库中返回NULL
CHARACTER_SET_NAME	varchar(32)	N	字符集名称，分析型数据库中返回NULL
COLLATION_NAME	varchar(32)	N	校验规则名称，默认为NULL
COLUMN_TYPE	varchar(1024)	N	列类型，目前支持类型：boolean/byte/ short/ int/long/float/ double/date/time/ timesta mp/ string(varchar)/ multiple string
COLUMN_KEY	varchar(3)	N	字段的键类型。分析型数据库中返回''

FIELD	TYPE	是否为分析型数据库扩展字段	COMMENT
EXTRA	varchar(27)	N	附加信息，目前返回为''
PRIVILEGES	varchar(80)	N	该列对应权限，分析型数据库中返回'SELECT'(大写)
COLUMN_COMMENT	varchar(255)	N	列注释，默认为''

- **PARTITIONS**

  提供表分区信息：

FIELD	TYPE	是否为分析型数据库扩展字段	COMMENT
TABLE_CATALOG	varchar(512)	N	NULL
TABLE_SCHEMA	varchar(64)	N	数据库名称
TABLE_NAME	varchar(64)	N	表名
PARTITION_NAME	varchar(64)	N	分区名称
SUBPARTITION_NAME	varchar(64)	N	二级分区名称
PARTITION_ORDINAL_POSITION	bigint(21)unsigned	N	分区编号，从1开始
SUBPARTITION_ORDINAL_POSITION	bigint(21)unsigned	N	子分区编号，从1开始
PARTITION_METHOD	varchar(12)	N	RANGE,LIST, HASH, LINEAR HASH, KEY, or LINEAR KEY
SUBPARTITION_METHOD	varchar(12)	N	子分区分区方法
PARTITION_EXPRESSION	varchar(1024)	N	分区表达式
SUBPARTITION_EXPRESSION	varchar(1024)	N	子分区表达式
PARTITION_DESCRIPTION	varchar(1024)	N	分区描述

FIELD	TYPE	是否为分析型数据库扩展字段	COMMENT
TABLE_ROWS	bigint(21)unsigned	N	该分区表中的记录数。innodb中是预估的
AVG_ROW_LENGTH	bigint(21)unsigned	N	平均行记录长度，inbytes
DATA_LENGTH	bigint(21)unsigned	N	数据长度，inbytes
MAX_DATA_LENGTH	bigint(21)unsigned	N	最大数据长度
INDEX_LENGTH	bigint(21)unsigned	N	索引长度
DATA_FREE	bigint(21)unsigned	N	空闲的长度，inbytes
CREATE_TIME	datetime	N	
UPDATE_TIME	datetime	N	
CHECK_TIME	datetime	N	分区最后一次检查时间
CHECKSUM	bigint(21)unsigned	N	分区的checksum
PARTITION_COMMENT	varchar(80)	N	注释
NODEGROUP	varchar(12)	N	所属的节点组。分析型数据库中置为""
TABLESPACE_NAME	varchar(64)	N	现在都是Default.分析型数据库中置为""
PARTITION_NO	bigint(21)unsigned	Y	分区号，从0开始
DATA_DISK_SIZE	bigint(21)unsigned	Y	该分区占用的磁盘空间
COMPRESS_RATIO	float	Y	数据压缩率
MEM_SIZE	bigint(21)unsigned	Y	装载该分区待的内存空间
ZIP_SIZE	bigint(21)unsigned	Y	该分区压缩包的大小

- **TABLE_DATA_LO分析型数据库**

  提供数据导入记录信息：

FIELD	TYPE	是否为分析型数据库扩展字段	COMMENT
FIELD	TYPE	DEFAULT	COMMENT
TABLE_SCHEMA	varchar(64)	-	数据库名
TABLE_NAME	varchar(64)	-	表名
TABLE_PARTITION	varchar(1024)		分区（/user_id/pt=20140801）
DATA_VERSION	bigint	-	数据版本
DATA_SOURCE_TYPE	int	-	数据源类型
DATA_SOURCE_PATH	varchar(1024)	-	数据源路径
DATA_SOURCE_SIZE	bigint	-	数据源大小
JOB_INSTANCE_ID	varchar(256)	-	Job实例ID
USER	varchar(64)	-	导入用户账号
USER_HOST	varchar(32)	-	导入用户主机地址
CREATE_TIME	datetime	-	导入开始时间

- **INDEXES**

  提供关于表索引的信息：

FIELD	TYPE	是否为分析型数据库扩展字段	COMMENT
TABLE_SCHEMA	varchar(64)	N	数据库名称。不分区大小写
TABLE_NAME	varchar(64)	N	表名称。不区分带瞎写
INDEX_NAME	varchar(64)	N	key的名称
COLUMN_NAME	varchar(64)	N	列名称
Asc_Or_Desc	varchar(1)	N	是否为倒序
INDEX_TYPE	varchar(16)	N	用过的索引方法。分析型数据库中返回BITMAP，HASH
COMMENT	varchar(16)	N	注释

28.13.2 附录二：SQL函数表

28.13.2.1 特色函数

UDF_SYS_COUNT_COLUMN

- **作用**：用于做多group by的聚合，可以将多个group by的语句合并成多个UDF，写到一条sql中。
- **格式**：UDF_SYS_COUNT_COLUMN(columnName,columnName2...)，参数必须是列名。
- **返回**：一个json的字符串列，类似：{"0":3331656,"2":3338142,"1":3330202}。
- **实例**：

```
a. select UDF_SYS_COUNT_COLUMN(c1) from table 等价于 select count(*) from table group by c1
b. select UDF_SYS_COUNT_COLUMN(c1,c2) from table 等价于select count(*) from table group by c1,c2
c. select UDF_SYS_COUNT_COLUMN(c1),UDF_SYS_COUNT_COLUMN(c2), UDF_SYS_COUNT_COLUMN(c3), UDF_SYS_COUNT_COLUMN(c4)等价于如下四条sql语句：
select count() from table group by c1 select count() from table group by c2 select count() from table group by c3 select count() from table group by c4d.
d.一个真实的实例：select UDF_SYS_COUNT_COLUMN(user_gender), UDF_SYS_COUNT_COLUMN(user_level) from db_name.userbase 返回1行，2列：{"0":3331656,"2":3338142,"1":3330202}，{"0":4668150,"2":1891176,"1":1984606，" 6":5818}
```

UDF_SYS_COUNT_DISTINCT_COLUMN

UDF_SYS_COUNT_COLUMN函数的去重计数版本。

UDF_SYS_RANGECOUNT_SAMPLING_COLUMN

- **作用**：用于做动态样本分段（旧版UDF_SYS_RANGECOUNT_SAMPLING_COLUMN函数已经deprecated，请使用该函数）。
- **格式**：UDF_SYS_RANGECOUNT_SAMPLING_COLUMN(columnName, count, min, max)：
  - 第一个参数是列名。
  - 第二个参数分段数目。
  - 第三个参数是参与分段的该列在全表的最小值。
  - 第四个参数是参与分段的该列在全表的最大值。
- **返回**：一个json的字符串列，类似：{"ranges":[{"start":0,"end":599},{"start":600,"end":1899},{"start":1900,"end":65326003}]}；
- **使用说明**：使用该函数进行动态分段统计，需要以下三个步骤：
  1. 通过 min, max 求出符合条件的最小值和最大值。
  2. 通过UDF_SYS_RANGECOUNT_SAMPLING_COLUMN获取各个分段。

3. 通过case when+group by获取每个分段中真实聚合数据。

- **实例**：
  - 得到分段的min和max值：

```
select min(freq), max(freq) from db_name.userbase where shoplist in (123456)
```

  - 采样取得分段值：

```
select  udf_sys_rangecount_sampling_column(db_name.userbase.freq,10,0,24000)  from
 db_name.userbase where db_name.userbase in (123456)
```

  - 根据采样的分段**case when**然后**group by**然后运行以下代码：

```
count select case
when  ((db_name.userbase.amt_avg>=0  and  db_name.userbase.amt_avg<=299))  then
 '0-299' when ((db_name.userbase.amt_avg>=300 and db_name.userbase.amt_avg<=599))
          then          '300-599'          when          ((db_name.userbase.amt_avg>=600          and
               db_name.userbase.amt_avg<=999))               then               '600-999'               when
       ((db_name.userbase.amt_avg>=1000        and        db_name.userbase.amt_avg<=1699))
      then       '1000-       1699'       when       ((db_name.userbase.amt_avg>=1700       and
            db_name.userbase.amt_avg<=3999))             then             '1700-3999'             when
 ((db_name.userbase.amt_avg>=4000 and db_name.userbase.amt_avg<=33902685)) then
  '4000-33902685'  else  'others'  end  as  x,  count(*)  from  db_name.userbase  where
db_name.userbase.shoplist in (123456) group by x
```

UDF_SYS_RANGECOUNT_COLUMN

- **作用**：用于做静态样本分段（旧版函数UDF_SYS_SEGCOUNT_COLUMN已经弃用，请使用该函数）；
- **格式**：UDF_SYS_RANGECOUNT_COLUMN(columnName, count, min, max)；
  - 第一个参数是列名。
  - 第二个参数分段数目。
  - 第三个参数是参与分段的该列在全表的最小值。
  - 第四个参数是参与分段的该列在全表的最大值。
- **返回**：一个json的字符串列，类似：{"ranges":[{"start":0,"end":599},{"start":600,"end":1899},{"start":1900,"end":65326003}]}；
- **使用说明**：使用该函数进行动态分段统计，需要以下三个步骤：

1. 通过min，max求出符合条件的最小值和最大值。
2. 通过UDF_SYS_RANGECOUNT_COLUMN获取各个分段。
3. 通过case when+group by获取每个分段中真实聚合数据。

- **特别说明**：

UDF_SYS_RAGNECOUNT_COLUMN和通过UDF_SYS_RANGECOUNT_SAMPLING_COLUMN的区别在于：前者是静态分段，也就是根据 (max-min+1)/segcount进行分段；后者是动态分段，可以保证每个分段区间内的数目是大致均衡的。

UDF_SYS_PERCENTILE_COLUMN

- **作用**：用于做均衡的分位数分段；
- **格式**：UDF_SYS_PERCENTILE_COLUMN(columnName, count, min, max)；
  - 第一个参数是列名。
  - 第二个参数分段数目。
  - 第三个参数是参与分段的该列在全表的最小值。
  - 第四个参数是参与分段的该列在全表的最大值。
- **返回**：一个json的字符串列，类似：

  {"ranges":[{"percent":1,"value":0},{"percent":2,"value":0},{"percent":3,"value":0}]}；
- **使用说明**：使用该函数进行动态分段统计，需要以下三个步骤：
  1. 通过 min, max 求出符合条件的最小值和最大值。
  2. UDF_SYS_PERCENTILE_COLUMN。
  3. 通过 casewhen+groupby 获取每个分段中真实聚合数据。

UDF_SYS_GEO_IN_CYCLE

- **作用**：用于做基于地理位置的经纬度画圈。
- **格式**：UDF_SYS_GEO_IN_CYCLE(longitude, latitude, point, radius)。
  - 第一个参数为经度列名称，类型 float。
  - 第二个参数为纬度列名称，类型 float。
  - 第三个参数为圆圈中心点的位置，格式=>"经度，维度", =>"120.85979,30.011984"。
  - 第四个参数为圆圈的半径，单位米。
- **返回**：返回一个boolean值。

- **使用说明**：

```
select count(*) db_name.usertag where udf_sys_geo_in_cycle(longitude,latitude, "120.85979,30.011984", 5000)=true
```

求以"120.85979,30.011984"为中心点，半径为5km的圆圈内的人数：

```
select longitude, latitude from db_name.usertag where udf_sys_geo_in_cycle (longitude,latitude, "120.85979,30.011984", 5000)=true order by longitude
```

UDF_SYS_GEO_IN_RECTANGLE

- **作用**：用于做基于地理位置的经纬度画矩形。
- **格式**：UDF_SYS_GEO_IN_RECTANGLE(longitude, latitude, pointA, pointB)：
  - 第一个参数为经度列名称，类型float。
  - 第二个参数为纬度列名称，类型float。
  - 第三个参数为矩形的左下角坐标，格式=>"经度，维度", =>"120.85979,30.011984"。
  - 第四个参数为矩形的右上角坐标，格式=>"经度，维度", =>"120.88450,31.21011"。
- **返回**：返回一个boolean值。
- **使用说明**：

```
select count(*) db_name.usertag where udf_sys_geo_in_rectangle(longitude,latitude, "120.85979,30.011984", "120.88450,31.21011")=true
```

求以"120.85979,30.011984"和""120.88450,31.21011""为2个斜角构成的矩形圈内的人数。

UDF_SYS_GEO_DISTANCE

- **作用**：用作一个经纬度列和一个固定的坐标点的距离计算。
- **格式**：UDF_SYS_GEO_DISTANCE(longitude, latitude, pointA)。
  - 第一个参数为经度列名称，类型float；
  - 第二个参数为纬度列名称，类型float；
  - 第三个参数为固定坐标点的经纬度，格式=>"经度，维度", =>"120.85979,30.011984" 。
- **返回**：返回一个int值，单位为米（M）。
- **使用说明**：

```
select count(*) db_name.usertag where udf_sys_geo_in_rectangle(longitude,latitude, "120.85979,30.011984", "120.88450,31.21011")=true
```

求以"120.85979,30.011984"和""120.88450,31.21011""为2个斜角构成的矩形圈内的人数。

28.13.2.2 字符串函数

- Concat：连接2个字符串，格式类似 cancat(str1, str2,...)。
- Lcase：返回字符串的全小写Ucase返回字符串的全大写。
- **Length:**返回字符串的长度Substring返回字符串的字串。
- **SUBSTRING (str,** pos， **[len]):**这里的pos为下标从1开始，比如substring('abc'，2,2)返回的是'bc'。
- **Trim:**类似java string的trim，用于除去字符串前后的space。
- **MID**函数用于从文本字段中提取字符，MID(column_name,start[,length])：
  - column_name 必需。要提取字符的字段。
  - start 必需。规定开始位置（起始值是 1）。
  - length 可选，要返回的字符数。如果省略，则MID函数返回剩余文本。
- **Left/Right(str,len)** 返回从字符串str开始的len最左/右字符。

28.13.2.3 日期函数

- **Year**

  查询指定列的年份，例如：year(date_test)

- **Month**

  查询指定列的月份，例如：mon th(date_test)

- **Day**

  查询指定列的日，例如：day(date_test)

- **Week**

  查询指定列的是所属年的第几周，支持mode参数（与MySQL定义相同），例如：week(date_test,1)

- **WeekDay**

  返回日期d是星期几的索引（位置），0 表示星期一，1 表示星期二，...，6 表示星期日

- **WeekofYear**

  相当于WEEK(d,3)

- **Hour/Minute/Second**

  返回一个Timestamp的小时/分钟/秒，例如：hour(time_stamp_test)

- **Datediff**

用于判断在两个日期之间存在的指定时间间隔的数目，用法Datediff(date1,date2)

- **to_days:**

  给定一个日期date,返回一个天数（从年份0开始的天数），用法to_days(date)

- **Yearmonth**

  查询指定列的日和月，例如：YEARMONTH('20140602')=201406

- **Curdate**

  查询当前日期，例如：curdate()

28.13.2.4 其他函数

- **CAST**

  转换当前列的数据类型，例如：cast(string_test as bigint)

- **Coalesce**

  返回第一个非null的表达式，使用方式：coalesce(expression [,...n]) ，例如：colesce(string_test, "")

- **Bit_Count**

  Bit_Count(value)：返回value的二进制转换“1”字节的个数

28.13.3 附录三：常见错误码表

类别	错误码	SQL状态码	错误信息	处理方法
DML异常	-1(65535)	HY000	Reject executionof sql with#key#,statement#sql#	SQL包含被过滤的关键字，联系阿里云团队确认
DML异常	1047	08S01	Nodatabase selected	修改语句，指定databaseschema，修改后重新执行
DDL异常	1064	HY000	ShowSyntaxException: You have an error in your SQL syntax	检查SHOW语句语法
权限异常	1142	42000	DenyAccessException: User[#user#@#host#] does not have[#action#] access to	确认用户的执行权限

类别	错误码	SQL状态码	错误信息	处理方法
			resource[#resource#]	
权限异常	1142	42000	NotAllowAcessException:User[#user#] does not have[#action#] access to resource[#resource#]	确认用户的执行权限
权限异常	1142	42000	AclException:User[#user#] does not have[#action#] access to resource[#resource#]	确认用户的执行权限
DML异常	1149	42000	SQLparsing failed(SQL syntax error):#Parse error message#	检查SQL语法，修改后重新执行
DML异常	1149	42000	SQLparsing unknown error:#Parse error message#	检查SQL语法，修改后重新执行
DML异常	1235	42000	SQLdump data selecting columns count exceed limit:#dump_column_count_limit#	DUMP列超过系统允许的最大值
DML异常	1235	42000	SQLselecting columns count exceed limit:#select_column_count_limit#	确认待select的列数超过系统允许的最大值
DML异常	1235	42000	SQLfeature NOT supported yet: SELECT orSELECT #tb#.	修改SQL，重新执行

类别	错误码	SQL状态码	错误信息	处理方法
DML异常	1236	42000	SQLfeature NOT supported yet: SELECT agg_func() ... UNION/UNION ALL/INTERSECT SELECT agg_func()...	修改SQL，采用子查询方式，重新执行
DML异常	1236	42000	SQLfeature NOT supported yet: SELECT ... ORDER BY ... UNION/UNION ALL/INTERSECT SELECT ... ORDER BY ...	修改SQL，目前分布式查询模式下，UNION分支中ORDER BY无意义，考虑修改查询逻辑
DML异常	1236	42000	SQLfeature NOT supported yet: SELECT ... LIMIT X UNION/UNION ALL/INTERSECT SELECT ... LIMIT X	修改SQL，目前分布式查询模式下，UNION分支中LIMIT无意义，考虑修改查询逻辑
DDL异常	3080	HY000	AcquireLockException :lockName =#schema_table#	提交重新执行
DDL异常	3081	HY000	IllegalParameterException: paramaterName=#parameter name#	检查DDL语句参数
DDL异常	3081	HY000	IllegalParameterException: paramaterName=Column type is invalid:#data_type#	检查DDL列的数据类型
DDL异常	3082	HY000	IllegalParameterValueExceptio n: [DDL-	检查DDL语句参数，为维度表指定groupName

类别	错误码	SQL状态码	错误信息	处理方法
			N3000]:groupName should be provided for DIMENSION tb.	
DDL异常	3082	HY000	IllegalParameterValueException: [DDL-N3001]: PARTITION NUM should = 1 for DIMENSION tb.	检查DDL语句参数，维度表分区数必须为1
DDL异常	3082	HY000	IllegalParameterValueException: [DDL-N3002]:hashPartitionNum should be provided and set value to 1 for DIMENSION tb.	检查DDL语句参数，为维度表指定hashPartitionNum，值必须为1
DDL异常	3082	HY000	IllegalParameterValueException: [DDL-N3003]: target tb group is not for DIMENSION tb:#tb group#	检查DDL语句参数，目标表组为分区表表组，不能创建维度表
DDL异常	3082	HY000	IllegalParameterValueException: [DDL-N3004]: target tb group is only for DIMENSION tb:#tb group#	检查DDL语句参数，目标表组为维度表组，不能创建分区表
DDL异常	3082	HY000	IllegalParameterValueException: [DDL-N3005]: cannot use DIMENSION_GROUP as target tb group, which is reserved for	检查DDL语句参数，不能使用DIMENSION_GROUP作为目标表组，为内部虚拟维度表组预留

类别	错误码	SQL状态码	错误信息	处理方法
			virtual dimension tb group.	
DDL异常	3082	HY000	IllegalParameterValueException: [DDL-N4000]: PARTITION NUMhould#### 0 for HASHpartition.	检查DDL语句参数，分区表的分区数必须大于0
DDL异常	3082	HY000	IllegalParameterValueExceptio n: [DDL-N5000]:originalTableName should not be null.	检查DDL语句参数，指定originalTableName
DDL异常	3082	HY000	IllegalParameterValueException:[DDL-NFFFF] Unexpectedexception message#	检查DDL语句参数on:#exception
DDL异常	3082	HY000	IllegalParameterValueExceptio n: parameterName=#parameterName#, parameterValue=#parameterValue#, correct type is #type#	检查DDL语句参数
DDL异常	3082	HY000	IllegalParameterValueExceptio n: #name#### is exceed the valid range, reasonable range: [#min#,#max#]	检查并修改DDL语句参数值范围
DDL异常	3082	HY000	IllegalParameterValueExceptio n:loader.origina	检查DDL语句参数，指定originalTableName

类别	错误码	SQL状态码	错误信息	处理方法
			lTableName is not null	
DDL异常	3082	HY000	IllegalParameterValueExceptio n:loader.originalTableName conflict	检查DDL语句参数，目标originalTableName已存在，修改originalTableName
DDL异常	3082	HY000	IllegalParameterValueException:valueType must is lllegal	检查DDL语句参数valueType
DDL异常	3082	HY000	IllegalParameterValueException:valueType is not null	检查DDL语句参数valueType
DDL异常	3084	HY000	DropTableOperationFailedException:#error message#	联系阿里云技术支持
DDL异常	3086	HY000	CreateTableOperation Failed:Table'#schema#.#tableName#' already exists	目标表已经存在，无需执行
DDL异常	3086	HY000	CreateTableOperation Failed:puttableInfo failed, sql:#DDL statement#	联系阿里云技术支持
DDL异常	3086	HY000	CreateTableOperation Failed:putonlineGroupInfo failed, sql:#DDL statement#	联系阿里云技术支持
DDL异常	3087	HY000	CreateTableRollBackException:#error message#	联系阿里云技术支持
DDL异常	3088	HY000	AlterTableIsNotExist:alter tb is not exist! tablePath:#tb path#	目标表不存在，确认表的正确性
DDL异常	3000	HY000	AlterRepeatException	目标index已存在

类别	错误码	SQL状态码	错误信息	处理方法
			:indexName:#indexName#### repeat	
DDL异常	3092	HY000	DescTableSqlFormatException:check sql	检查并修正SHOW语法，重新执行
DDL异常	3094	HY000	DropSyntaxException:#DDL statement#	检查并修正DROP#DDL语法，重新执行
DDL异常	3096	HY000	AlterSyntaxException:#DDL statement#	检查并修正ALTER DDL语法，重新执行
DDL异常	3099	HY000	ExceedMaxTableNumException:maxTableNum=#maximum tb number allowedin this database#	目标数据库已到达允许的表数上限，请删除无用表
DML异常	31000	HY000	SQLplanning partition routing error:#error message#	系统错误，联系阿里云技术支持，若消息中含有数据，无法查询DATA_NOT_FOUND，亦有可能是离线导入的表还没第一次导入
DML异常	32000	HY000	SQLplanning partition routing error: ZERO_ROUTE	系统错误，联系阿里云技术支持
DML异常	51000	HY000	Partitionsquery error: all failed#error message#	系统错误，联系阿里云技术支持
DML异常	55000	HY000	Partitionmerging error:#error message#	系统错误，联系阿里云技术支持

类别	错误码	SQL状态码	错误信息	处理方法
DML异常	55001	HY000	Partitionsquery error: partitions miss: #missed part count#	系统错误，联系阿里云技术支持
导入异常	1066	HY000	unableto extract param#sourcePath#	语法检查失败，检查语句中参数，重新load data
导入异常	1066	HY000	jobdoes not exist:	确认job是否成功提交
导入异常	1066	HY000	checksignature exception:	系统错误，联系阿里云技术支持
导入异常	1066	HY000	targettable not exist:	确认loaddata中的分析型数据库表存在后再发送 load data
导入异常	1066	HY000	targetdatabase not found:	确认loaddata中的分析型数据库存在后再发送 load data
导入异常	1066	HY000	tableDB.TABLE hash partition num:XXX more than limit XXX	分析型数据库离线导入表表 HASH分区数限制为不超过256，需调整分区数
导入异常	1066	HY000	tablesize exceed limit	暂未使用
导入异常	1066	HY000	sourcetable has no records	确认odps源表中有数据后再发送 load data
导入异常	1066	HY000	sourcetable column mapping with dest table error	确定分析型数据库表中的列全部在源表中找得到对应的列
导入异常	1066	HY000	getodps table error	确认是否授权garuda_build@aliyun.com对

类别	错误码	SQL状态码	错误信息	处理方法
				源表/视图的DESC权限
导入异常	1066	HY000	nopermission to select source table	确认是否授权garuda_build@aliyun.com对原表/视图的SELECT权限

29 流计算

29.1 产品概述

29.1.1 什么是流计算

目前对信息高时效性、可操作性的需求不断增长，这要求软件系统在更少的时间内能处理更多的数据。传统的大数据处理模型将在线事务处理和离线分析从时序上将两者完全分割开来，但显然该架构目前已经越来越落后于人们对于大数据实时处理的需求。

流计算的产生即来源于对于上述数据加工时效性的严苛需求： 数据的业务价值随着时间的流失而迅速降低，因此在数据发生后必须尽快对其进行计算和处理。而传统的大数据处理模式对于数据加工均遵循传统日清日毕模式，即以小时甚至以天为计算周期对当前数据进行累计并处理，显然这类处理方式无法满足数据实时计算的需求。在诸如实时大数据分析、风控预警、实时预测、金融交易等诸多业务场景领域，批量（或者说离线）处理对于上述对于数据处理时延要求苛刻的应用领域而言是完全无法胜任其业务需求的。而流计算作为一类针对流数据的实时计算模型，可有效地缩短全链路数据流时延、实时化计算逻辑、平摊计算成本，最终有效满足实时处理大数据的业务需求。

什么是流数据？

从广义上说，所有大数据的生成均可以看作是一连串发生的离散事件。这些离散的事件以时间轴为维度进行观看就形成了一条条事件流/数据流。不同于传统的离线数据，流数据是指由数千个数据源持续生成的数据，流数据通常也以数据记录的形式发送，但相较于离线数据，流数据普遍的规模较小。流数据产生源头来自于源源不断的事件流，例如客户使用您的移动或 Web 应用程序生成的日志文件、网购数据、游戏内玩家活动、社交网站信息、金融交易大厅或地理空间服务，以及来自数据中心内所连接设备或仪器的遥测数据。

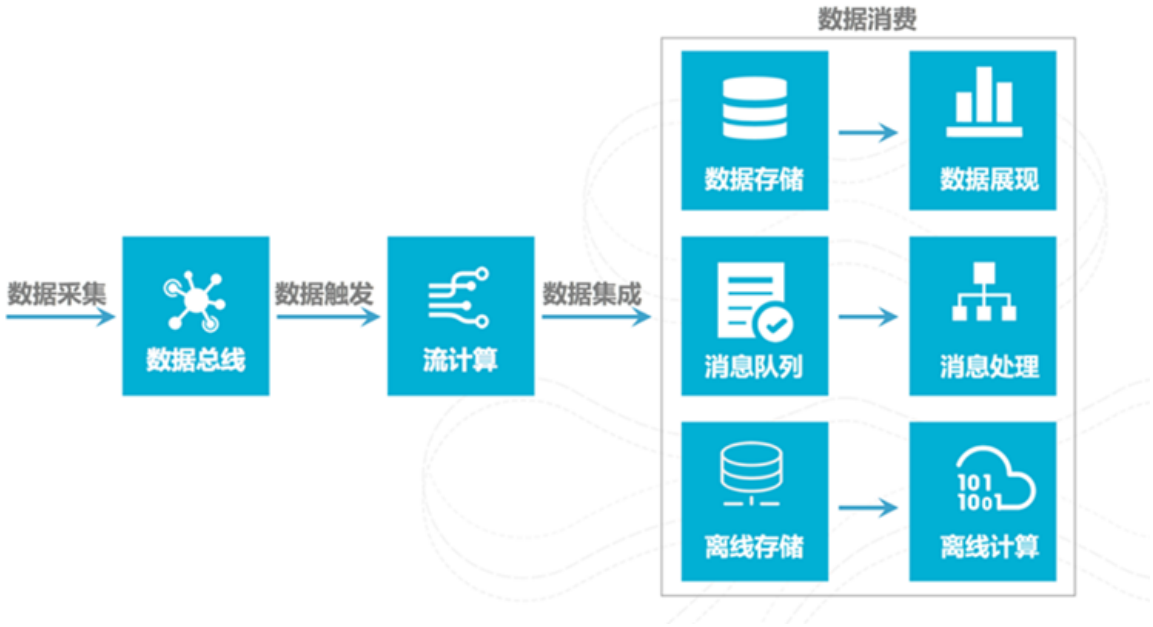
通常而言，流计算具备三大类特点：

- 实时（realtime）且无界（unbounded）的数据流。流计算面对计算的数据源是实时且流式的，流数据是按照时间发生顺序地被流计算订阅和消费。且由于数据发生的持续性，数据流将长久且持续地集成进入流计算系统。例如，对于网站的访问点击日志流，只要网站不关闭其点击日志流将一直不停产生并进入流计算系统。因此，对于流系统而言，数据是实时且不终止（无界）的。

- 持续（continuos）且高效的计算。流计算是一种”事件触发”的计算模式，触发源就是上述的无界流式数据。一旦有新的流数据进入流计算，流计算立刻发起并进行一次计算任务，因此整个流计算是持续进行的计算。
- 流式（streaming）且实时的数据集成。流数据触发一次流计算的计算结果，可以被直接写入目的数据存储，例如将计算后的报表数据直接写入RDS进行报表展示。因此流数据的计算结果可以类似流式数据源一样持续写入目的数据存储。

29.1.2 全链路流计算

不同于现有的离线/批量计算模型，流计算全链路整体上更加强调数据的实时性，具备数据实时采集、数据实时计算和数据实时集成的功能。三大类数据的实时处理逻辑在全链路上保证了流式计算的低时延。全链路流计算示意图如下：



- 数据采集

  用户使用流式数据采集工具将数据**流式且实时**地采集并传输到大数据消息 Pub/Sub 系统，该系统将为下游流计算提供源源不断的事件源去触发流式计算任务的运行。

- 流式计算

  流数据作为流计算的触发源驱动流计算运行。因此，**一个流计算任务必须至少使用一个流数据作为数据源**。一批进入的数据流将直接触发下游流计算的一次流式计算处理。

- 数据集成

流计算将计算的结果数据直接写入目的数据源，这其中包括多种数据源，包括数据存储系统、消息投递系统，甚至直接对接业务规则告警系统发出告警信息。不同于批量计算（例如阿里云 MaxCompute 或者开源 Hadoop），**流计算天生自带数据集成模块，可以将结果数据直接写入到目的数据源。**

- 数据消费

  流计算一旦将结果数据投递到目的数据源后，后续的数据消费从系统划分来说，和流计算已经完全解耦。用户可以使用数据存储系统访问数据，使用消息投递系统进行信息接收，或者直接使用告警系统进行告警。

29.1.3 流计算和批量计算对比

相比于批量大数据计算，流计算整体上还属于较为新颖的计算概念，下面我们从用户/产品层面来理解两类计算方式的区别。

29.1.3.1 批量计算

目前绝大部分传统数据计算和数据分析服务均是基于批量数据处理模型：使用 ETL 系统或者 OLTP 系统进行构造数据源，在线的数据服务（包括 Ad-Hoc 查询、DashBoard 等服务）通过构造 SQL 语言访问上述数据源并取得分析结果。这套数据处理的方法论伴随着关系型数据库在工业界的演进而被广泛采用。但在大数据时代下，伴随着越来越多的人类活动被信息化、进而数据化，越来越多的数据处理要求实时化、流式化，当前这类处理模型开始面临实时化的巨大挑战。传统的批量数据处理模型传统的批量数据处理通常基于如下处理模型：

批量计算



使用 ETL 系统或者 OLTP 系统构造原始的数据源，以提供给后续的数据服务进行数据分析和数据计算。如上图，用户装载数据，系统将根据自己的存储和计算情况，对于装载的数据进行索引构建等

一系列查询优化工作。因此，对于批量计算，**数据一定需要预先加载到计算系统**，**后续计算系统才在数据加载完成后方能进行计算。**

用户/系统**主动**发起一个计算任务（例如 MaxCompute 的 SQL 任务，或者 Hive 的 SQL 任务）并向上述数据系统进行请求。此时计算系统开始调度（启动）计算节点进行大量数据计算，该过程的计算量可能巨大，耗时长达数分钟乃至于数小时。同时，由于数据累计的不可及时性，上述计算过程的数据一定是历史数据，无法保证数据的”新鲜”。用户可以根据自己需要随时调整计算 SQL，甚至于使用 AdHoc 查询，可以做到即时修改即时查询。

计算结果返回，计算任务完成后将数据以结果集形式返回用户，或者可能由于计算结果数据量巨大保存着数据计算系统中，用户进行再次数据集成到其他系统。一旦数据结果巨大，整体的数据集成过程漫长，耗时可能长达数分钟乃至于数小时。

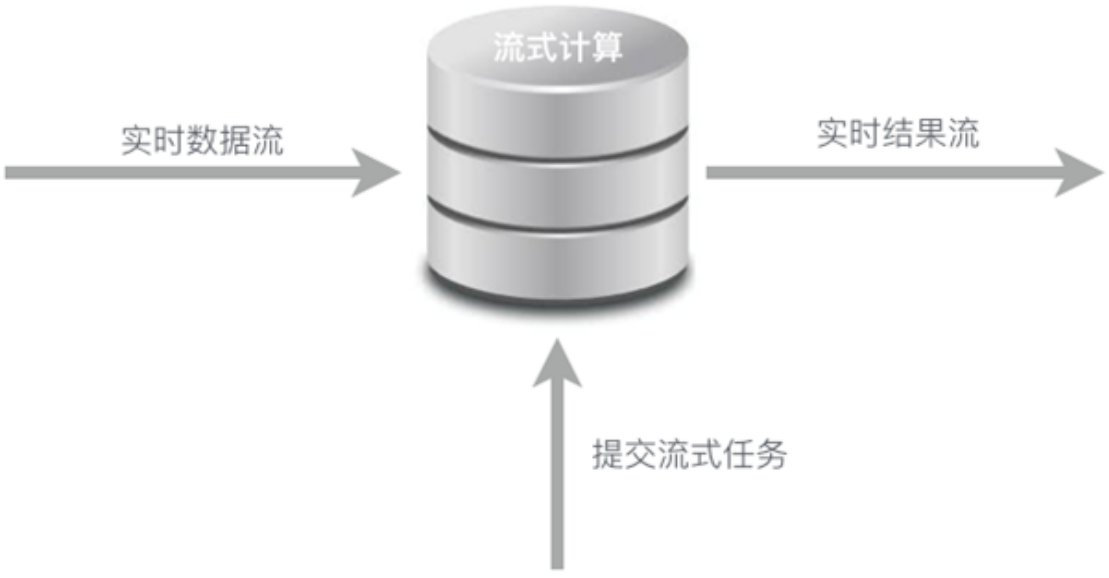
批量计算是一种批量、高时延、主动发起的计算任务。 用户使用的批量计算的顺序是：

1. 预先加载数据。
2. 提交计算任务，并且可以根据业务需要修改计算任务，再次提交任务。
3. 计算结果返回。

29.1.3.2 流式计算

不同于批量计算模型，流式计算更加强调计算数据流和低时延，流式计算数据处理模型如下：

流式计算



使用实时数据集成工具，将数据实时变化传输到流式数据存储（即消息队列，如 DataHub）；此时数据的传输变成实时化，将长时间**累积**大量的数据平摊到每个时间点不停地小批量实时传输，因此数据集成的时延得以保证。

此时数据将源源不断写入流数据存储，不需要预先加载的过程。同时流计算对于流式数据不提供存储服务，数据是持续流动，在计算完成后就立刻丢弃。

数据计算环节在流式和批量处理模型差距更大，由于数据集成从**累积**变为**实时**，不同于批量计算**等待数据集成全部就绪后才启动计算任务**，流式计算任务是一种常驻计算**服务**，一旦启动将一直处于**等待事件触发**的状态，一旦有小批量数据进入流式数据存储，流计算立刻计算并迅速得到结果。同时，阿里云流计算还使用了增量计算模型，将大批量数据分批进行增量计算，进一步减少单次运算规模并有效降低整体运算时延。

从用户角度，对于流式任务，必须预先定义计算逻辑，并提交到流式计算系统中。在整个运行期间，流计算作业逻辑不可更改！用户通过停止当前作业运行后再次提交作业，此时之前已经计算完成的数据是无法重新再次计算。

不同于批量计算结果数据需等待数据计算结果完成后，批量将数据传输到在线系统；流式计算任务在每次小批量数据计算后可以立刻将数据写入在线/批量系统，无需等待整体数据的计算结果，可以立刻将数据结果投递到在线系统，进一步做到实时计算结果的实时化展现。

流计算是一种持续、低时延、事件触发的计算任务，用户使用流计算的顺序是：

1. 提交流计算任务。
2. 等待流式数据触发流计算任务。
3. 计算结果持续不断对外写出。

29.1.3.3 模型对比

下表给出了流计算与批量计算两类计算模型的差别：

对比指标	批量计算	流式计算
数据集成方式	预先加载数据。	实时加载数据实时计算。
使用方式	业务逻辑可以修改，数据可重新计算。	业务逻辑一旦修改，之前的数据不可重新计算（流数据易逝性）。
数据范围	对数据集中的所有或大部分数据进行查询或处理。	对滚动时间窗口内的数据或仅对最近的数据记录进行查询或处理。

对比指标	批量计算	流式计算
数据大小	大批量数据。	单条记录或包含几条记录的微批量数据。
性能	几分钟至几小时的延迟。	只需大约几秒或几毫秒的延迟。
分析	复杂分析。	简单的响应函数、聚合和滚动指标。

在大部分大数据处理场景下，受限于当前流计算的整个计算模型较为简单，流计算是批量计算的**有效增强**，特别在于对于事件流处理时效性上，**流计算对于大数据计算是一个不可或缺的增值服务**。

29.2 产品说明

29.2.1 产品历程

阿里云流计算脱胎于阿里集团内部双十一实时大屏业务，在阿里集团内部从最开始支持双十一大屏展现和部分实时报表业务的实时数据业务团队，历经4、5年的长期摸索和发展，到最终成长一个独立稳定的云计算产品团队。阿里云流计算期望将阿里集团本身沉淀多年的流计算产品、架构、业务能够以云产品的方式对外提供服务，助力更多中小企业实时化自身大数据业务。

最初阿里集团支撑双十一大屏等业务同样采用的是开源的Storm作为基础系统支持，并在上面开发相关 Storm 代码。这个时期的实时业务处于萌芽阶段，规模尚小。数据开发人员使用 Storm 原生 API 开发流式作业，开发门槛高，系统调试难，存在大量重复的人肉工作。

阿里集团的工程师针对这类大量重复工作，开始考虑进行业务封装和抽象。工程师们基于 Storm 的 API 开发出大量可复用的数据统计组件，例如实现了简单过滤、聚合、窗口等等作为基础的编程组件，并基于这类组件提供了一套 XML 语义的业务描述语言。基于这套设计，流式计算用户可以使用 XML 语言将不同的组件进行拼装描述，最终完成一整套完整的流计算处理流程。基于 XML +Storm 组件的编程方式，从底层上避免了用户大量的重复开发工作，同时亦降低了部分使用门槛。但我们的数据分析人员仍然需要熟悉整套编程组件和 XML 描述语法，这套编程方式离分析人员最熟悉的 SQL 方式仍然差距甚远。

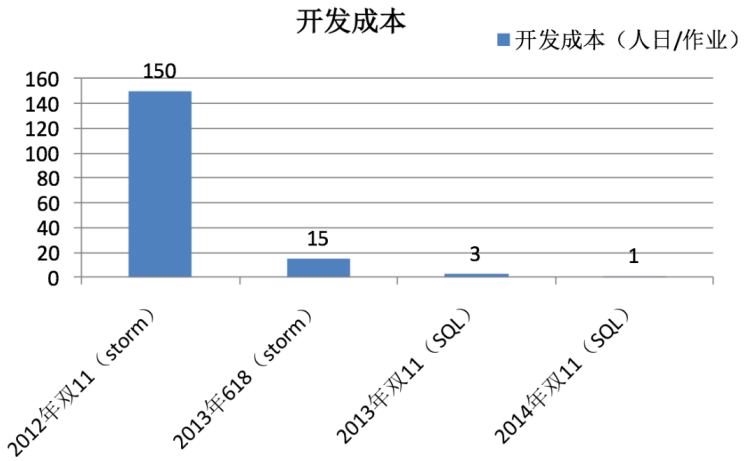
任何技术的发展一定遵循 **小众/创新** 到 **大众/普及** 的成长轨迹，而从小众到大众，从创新到普及的转折点一定在于技术的功能成熟和成本降低。阿里工程师开始思考如何更大程度降低数据分析产品门槛从而普及到更多的用户。得益于关系型数据库几十年沉淀的用户群体，使用经典的 SQL 模式去计算和处理数据一则可以对标 SQL 功能从而提炼我们的技术成熟度，二则可以利用用户熟悉的 SQL模型可极大降低用户上手使用流计算的门槛。因此，阿里工程师最终开发一套 StreamSQL 替换了

原有的 XML+ 组件的编程方式，这套系统成为今天阿里云流计算的核心计算引擎（Galaxy）。当前这套系统以单机群数千台机器规模，在阿里集团内部服务20+BU，日均消息处理数千亿，流量近 PB 级别，成为阿里集团最核心的流式计算集群。

当前阿里云流计算在原有 Galaxy 系统基础上，更加丰富和提升了用户的使用体验，包括提供一整套的开发平台，完整的流式数据处理业务流程。使用阿里云流计算，受益于阿里大数据多年的技术和业务沉淀，用户可以完全享受到阿里集团最新最前沿的计算引擎能力，业务上可规避阿里集团多年在流式大数据的试错和教训，让用户自身可以更快、更轻松地实时化大数据处理流程，助力业务发展。

以下是阿里集团工程师使用阿里云流计算开发工期对比，如图 *975: 开发成本*所示。

图 975: 开发成本



29.2.2 产品定位

阿里云流计算提供类标准的 StreamSQL 语义协助用户简单轻松完成流式计算逻辑的处理。同时，受限于 SQL 代码功能有限无法满足某些特定场景的业务需求，阿里云流计算同时为部分授信用户提供全功能的 UDF 函数，帮助用户完成业务定制化的数据处理逻辑。在流数据分析领域用户直接使用 StreamSQL+UDF 即可完成大部分流式数据分析处理逻辑，目前的流计算**更擅长于做流式数据分析、统计、处理**，对于非 SQL 能够解决的领域，例如复杂的迭代数据处理、复杂的规则引擎告警则不适合现有的流计算产品去解决。

目前流计算擅长解决的几个领域的应用场景：

- 实时的网络点击 PV、UV 统计。

- 统计交通卡口的平均五分钟通过车流量。
- 水利大坝的压力数据统计和展现。
- 网络支付涉及金融盗窃固定行为规则的告警。

曾经阿里云流计算对接，但发现无法满足的情况:

- Oracle 存储过程使用阿里云流计算替换: 流计算无法从功能上完全替换掉 Oracle 存储过程，两者面向问题领域不一致。
- 现有的 Spark 作业无缝迁移到流计算：Spark部分涉及流计算可以考虑改造并迁移到流计算，用户可以完全省去运维 Spark 和开发 Spark 的各类成本，但无法做到 Spark 作业无缝迁移到流计算。
- 多种复杂规则引擎告警：针对单一一条数据存在多条复杂规则告警，且该规则在系统运行时变化。这类应该有专门的规则引擎系统解决，当前流计算面对不是该问题域。

当前流计算对外接口定义为 StreamSQL/UDF，提供服务于流式数据分析、统计、处理的一站式开发工具，面向客户更多是数仓开发人员、数据分析师，这类客户不希望更多参与底层代码开发，而希望简单编写流计算 SQL 即可完成自身流式数据分析业务。

29.2.3 产品术语

- 项目空间（Project）

  项目空间是阿里云流计算最基本的业务组织单元，是用户管理集群、作业、资源、人员的基本单元。用户可以选择创建项目，同时亦可以子账号身份加入其他 Project 中。流计算的项目空间通过阿里云 RAM 主子账号支持多人协作。

- 作业（Job）

  类似一个 MaxCompute/Hadoop Job，一个流计算的作业描述了一个完整的流式数据处理业务逻辑，是流式计算的基础业务单元。

- 流计算单元（CU）

  在阿里云流计算中，作业的流计算单元为 CU。一个 CU 描述了一个流计算作业最小运行能力，即在限定的 CPU、内存、IO 情况下对于事件流处理的能力。一个流计算作业可以指定在 1 个或者多个 CU 上运行。

  当前流计算定义 `1 CU = 1 Core CPU + 4G MEM`。

- StreamSQL

不同于诸多开源的流式数据处理系统提供非常底层的编程 API，阿里云流计算提供更加高层更加面向业务化的 StreamSQL（标准 SQL 语法上提供了关于流式处理的语法扩展），方便数据开发人员使用标准化的 SQL 即可完成流式数据计算加工的业务流程。因此，阿里云流计算适合面向更大众的数据分析人员快速、方便地完成一个流式数据处理业务。

- UDF

  类似于 Hive UDF 函数，StreamSQL 提供了标准化的流式数据处理能力同时，对于部分业务特殊自定义处理逻辑，建议用户使用 UDF 函数表达。目前阿里云流计算仅支持 Java 的 UDF 函数扩展。

- 资源（Resource）

  阿里云流计算支持 UDF（User Define Function，即用户自定义函数)函数），当前 UDF 函数仅支持使用 Java 语言表达，对于每个用户上传的 Jar，流计算定义为一个 Resource。

- 数据采集（Data Collection）

  广义的数据采集指将数据从数据源产生方收集并传输进入到大数据处理引擎的过程，在阿里云流计算，数据采集原则上遵循上述定义，但更加聚焦为将流式数据从数据源产生方收集并传输进入数据总线的过程。

- 数据存储（Data Store）

  阿里云流计算定义为一种轻量级计算引擎，**本身不带有任何业务数据存储系统**。阿里云流计算均是使用外部数据存储作为数据来源和数据目的端进行使用。阿里云流计算将数据存储均定义为外部的数据存储，例如将用户 RDS 作为结果表，那么RDS即是流计算的一类DataStore。

- 数据加工（Data Develop）

  流式计算的开发过程（即编写 StreamSQL 的过程），将其定义为数据加工。阿里云流计算提供一整套包括开发、调试的在线 IDE，服务流式数据加工过程。

- 数据运维（Data Operation）

  流计算作业的在线运维定义为数据运维。阿里云流计算提供一整套管控平台，方便用户进行流式数据的运维管控。

29.2.4 产品特点

相较于其他流计算产品，阿里云流计算提供一些极具竞争力的产品优势，用户可以充分利用阿里云流计算提供的产品优势，方便快捷的解决自身业务实时化大数据分析的问题。

强大的实时处理能力

不同于其他开源流计算中间件只提供粗陋的计算框架，大量的流计算细节需要业务人员造轮子重新实现。阿里云流计算集成诸多全链路功能，方便用户进行全链路流计算开发，包括：

- 强大的流计算引擎，阿里云流计算提供提供标准的 StreamSQL，支持各类Fail场景的自动恢复，保证故障情况下数据处理的准确性；支持多种内建的字符串处理、时间、统计等类型函数；精确的计算资源控制，彻底保证公共云用户作业的隔离性。
- 关键性能指标超越 Storm 的六到八倍，数据计算延迟优化到秒级乃至毫秒级，单个作业吞吐量可做到百万级别，单集群规模在数千台。
- 深度整合各类云数据存储，包括 DataHub、日志服务(SLS)、RDS、OTS、ADS、IOTHub 等各类数据存储系统，无需额外的数据集成工作，阿里云流计算可以直接读写上述产品数据。

托管的实时计算服务

不同于开源或者自建的流式处理服务，阿里云流计算是完全托管的流式计算引擎，阿里云可针对流数据运行查询，无需预置或管理任何基础设施。在阿里云流计算，您可以享受一键启用的流式数据服务能力。阿里云流计算天然集成数据开发、数据运维、监控预警等服务，方便您最小成本试用和迁移流式计算。

提供完全租户隔离的托管运行服务，从最上层工作空间到最底层执行机器，提供最有效的隔离和全面防护，让用户放心使用流计算。

良好的流式开发体验

支持标准 SQL（产品名称为：StreamSQL），提供内建的字符串处理、时间、统计等各类计算函数，替换业界低效且复杂的 Storm 开发，让更多的 BI 人员、运营人员通过简单的 StreamSQL 可以完成实时化大数据分析和处理，让实时大数据处理普适化、平民化。

提供全流程的流式数据处理方案，针对全链路流计算提供包括数据开发、数据运维、监控预警等不同阶段辅助套件，让数据开发仅需三步，即可完成流式计算任务上线。

低廉的人力和集群成本

大量优化的 SQL 执行引擎，会产生比手写原生 Storm 任务更高效且更廉价的计算任务，无论开发成本和运行成本，阿里云流计算均要远低于开源流式框架。考虑下，编写一个复杂业务逻辑下 Storm 任务 Java 代码行数，考虑下针对这个任务的调试、测试、调优、上线工作成本，考虑下后续长期对于 Storm、Zookeeper 等开源软件的运维成本。如果使用阿里云流计算，上述问题完全交由阿里云平台承担，用户完全聚焦业务，快速实现市场目标。

另外，在公共云环境下，阿里云流计算在内测和公测阶段均免费提供一定限额的计算资源。

29.2.5 产品安全

安全事关产品和业务生死，阿里云流计算从产品设计第一天就重点考量了整体全链路流计算的安全性。

29.2.5.1 账号安全

流计算账号安全

流计算账号当前支持且仅支持阿里云账号体系（包括登录用户名+密码、签名密钥），这部分全部遵守阿里云现有安全体系，同时传输链路全部使用 HTTPs 协议，保证全链路的用户账户安全。

更多系统账号管理，请参见《账号管理》。

29.2.5.2 业务安全

项目隔离安全

流计算对不同的项目进行了严格的项目权限区分，不同用户/项目之间是无法进行访问、操作，包括项目下属的所有子产品实体均无法操作。

项目级别的资源隔离能够保证不同用户的资源使用情况相互之间不相互干扰影响，例如一个用户任务在运行期间随着数据量的突增提升了其作业CPU使用。阿里云流计算在底层使用虚拟化技术进行资源隔离，保证该用户的作业CPU使用率增加不会影响到其他用户作业的CPU使用情况。

业务流程安全

流计算对于流式计算开发进行了严格的流程定义，区分了数据开发和数据运维，在尽可能不影响用户使用体验基础上，保证了整体业务流程的完整和安全性：

- 提供代码版本：支持代码版本回滚和对比，方便用户进行代码追溯、比对、排错。
- 提供IDE单机调试容器：避免代码线下运行影响线上真实数据。用户可以对输入表、维表、输出表自行构造数据，以避免线下任务调试对于线上生产任务影响。
- 提供发布流程：避免线下代码改动直接影响生产运行。用户调试完成后，通过上线任务将作业提交到数据运维系统。此时正在运行的流计算任务并不直接使用新代码运行，而需要用户经过人工确认后将运行任务停止并使用新代码启动，从流程上保证发布的严谨性。

29.2.5.3 数据安全

系统数据安全

- 流计算系统数据安全交由系统本身安全保证，流计算为系统安全做了诸多工作。
- 访问链路全部HTTPs化，保证传输链路的安全。
- 数据存储连接信息使用AES高强度加密方式，保证敏感信息不泄露。
- 全面且深入的攻击测试，阿里云安全团队为流计算保驾护航。

业务数据安全

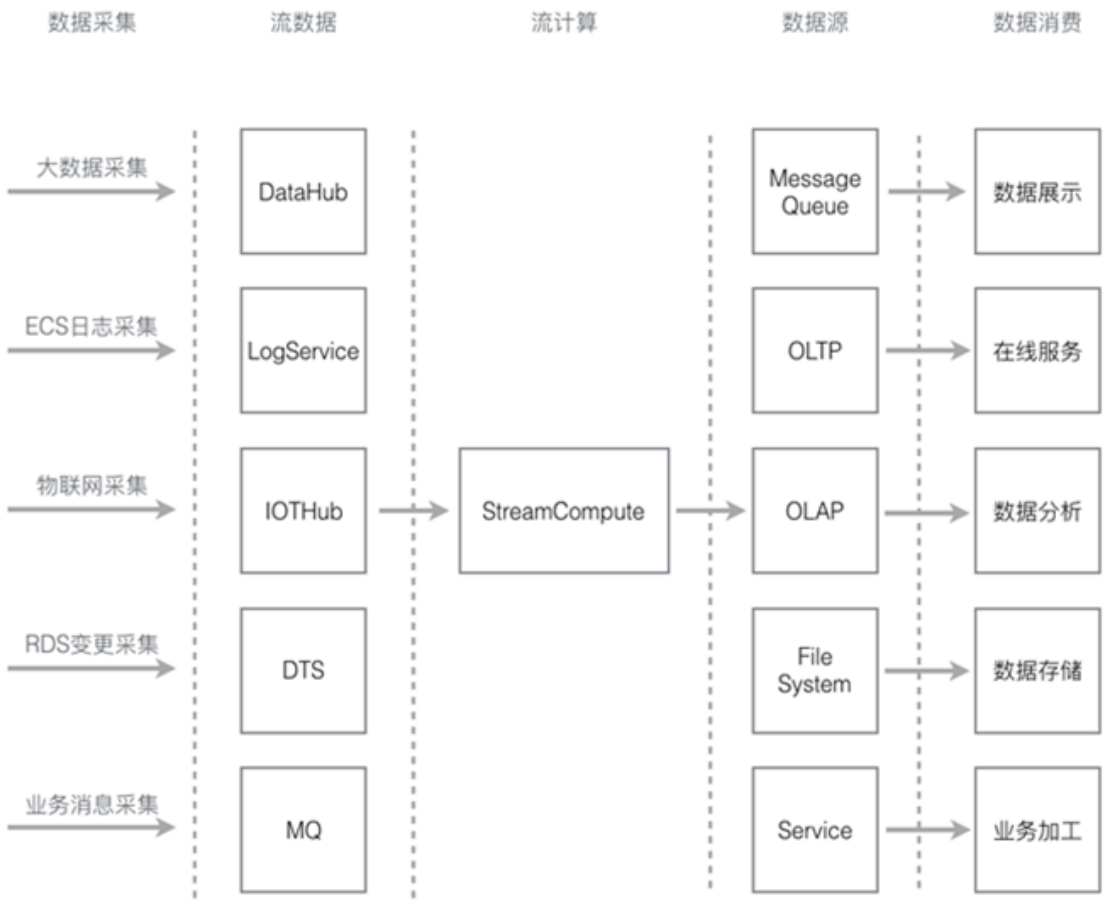
流计算本身不负责存储用户的业务数据，具体业务数据安全交由不同的阿里云存储系统保证，具体请参考不同的数据存储的安全模型以及最佳安全实践。

29.2.6 业务流程

在阿里云流计算使用前，对流式数据处理整体全链路有个简单认识可以极大方便用户梳理业务流程，制定相应的系统设计方案。

下面将简单介绍下阿里云流计算全流程系统架构情况。

阿里云流计算全流程系统架构



- **数据采集**

  广义的实时数据采集指：用户使用流式数据采集工具将数据**流式且实时地采集并传输到大数据Pub/Sub系统**，该系统将为下游流计算提供源源不断的事件源去触发流式计算任务的运行。阿里云大数据生态中提供了诸多针对不同场景领域的流式数据Pub/Sub系统，阿里云流计算天然集成上图中诸多Pub/Sub系统，以方便用户可以轻松集成各类流式数据存储系统。例如用户可以直接使用流计算对接SLS的LogHub系统，以做到快速集成并使用ECS日志。

- **流式计算**

  流数据作为流计算的触发源驱动流计算运行。因此，**一个流计算任务必须至少使用一个流数据作为数据源**。同时，对于一些业务较为复杂的场景，流计算还支持和静态数据存储进行关联查询。例如针对每条DataHub流式数据，流计算将根据流式数据的主键和RDS中数据进行关联查询（即join查询）；同时，阿里云流计算还支持针对多条数据流进行关联操作，StreamSQL支持阿里集团量级的复杂业务也不在话下。

- **实时数据集成**

为尽可能减少数据处理时延，同时减少数据链路复杂度，阿里云流计算将计算的结果数据可不经其他过程直接写入目的数据源，从而最大程度降低全链路数据时延，保证数据加工的新鲜度。为了打通阿里云生态，阿里云流计算天然集成了OLTP（RDS产品线等）、NoSQL（Table Store等）、OLAP（Analytic DB等）、MessageQueue（DataHub、ONS等）、MassiveStorage（OSS、MaxCompute等）。

- **数据消费**

  流式计算的结果数据进入各类数据源后，用户可以使用各类个性化的应用消费结果数据： 用户可以使用数据存储系统访问数据，使用消息投递系统进行信息接收，或者直接使用告警系统进行告警。

29.2.6.1 数据链路情况

对于上图的数据链路，部分数据源由于和流计算模型不能一一匹配，需要使用其他类型的流数据做中转，说明如下:

DataHub

DataHub提供了多类数据(包括日志、数据库BinLog、Iot数据流等等)从数据源上传到DataHub的工具、界面，以及和一些开源、商业软件的集成，请参见《DataHub相关介绍文档》，即可获取丰富多样的数据采集工具。

日志服务（LogService）

LogService是针对日志类数据一站式服务，在阿里巴巴集团经历大量大数据场景锤炼而成。Log服务提供了诸多的针对日志的采集、消费、投递、查询分析等功能。

您可以查看《*LogService*有关数据采集》一章，了解如何使用日志进行流式数据消费。

物联网套件（IOTHub）

物联网套件是阿里云专门为物联网领域的开发人员推出的，其目的是帮助开发者搭建安全性能强大的数据通道，方便终端（如传感器、执行器、嵌入式设备或智能家电等等）和云端的双向通信。

使用IotHub 规则引擎可以将IOT数据方便投递到DataHub，并利用流计算和MaxCompute进行数据加工计算。查看《IOT规则引擎使用》以查看如何将IOT数据推送到DataHub。

数据传输（DTS）

DTS支持以数据库为核心的结构化存储产品之间的数据传输。 它是一种集数据迁移、数据订阅及数据实时同步于一体的数据传输服务。使用DTS的数据传输功能，可以方便的将RDS等BinLog解析并投递到DataHub，并利用流计算和MaxCompute进行数据加工计算。

当前DTS传输到DataHub功能已经上线，欢迎使用。具体文档请查看《*RDS*到*DataHub*数据实时同步》。

Message Service

阿里云消息服务阿里云商用的消息中间件服务，具有大规模，高可靠、高并发访问和超强消息堆积能力的特点。流计算可以直接从消息服务读取流式数据。

阿里云流计算对接消息服务当前仍在开发中，敬请期待。

MQ

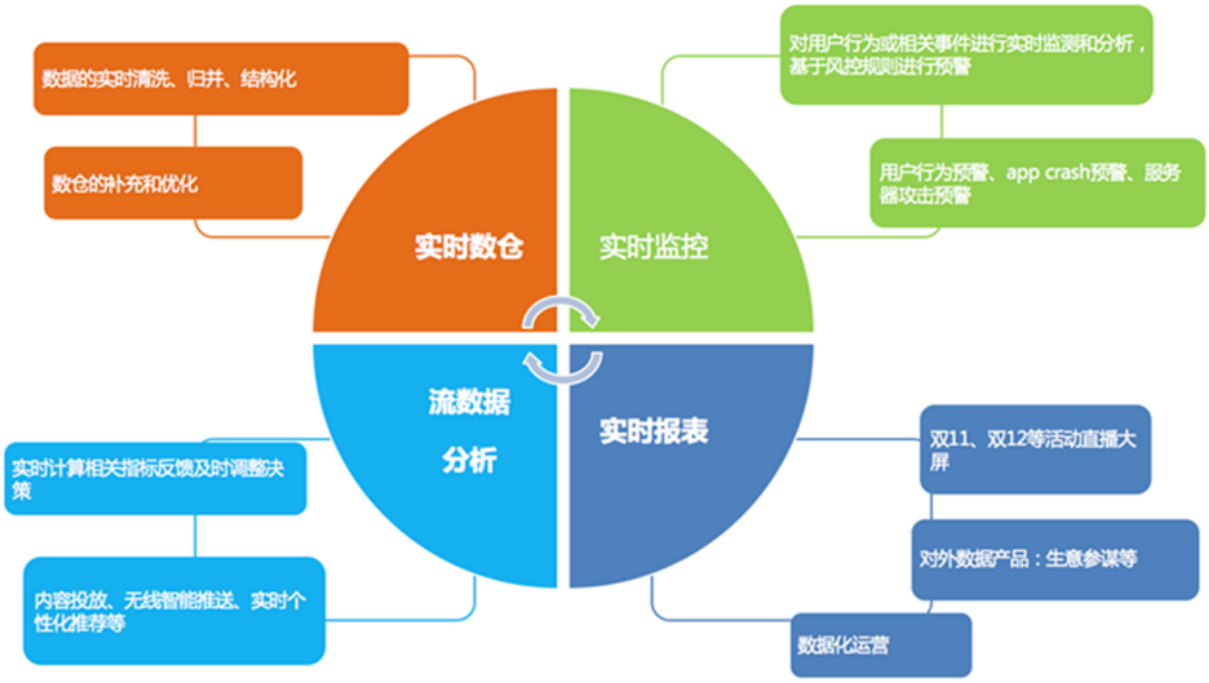
阿里云MQ服务是企业级互联网架构的核心产品，基于高可用分布式集群技术，搭建了包括发布订阅、消息轨迹、资源统计、定时（延时）、监控报警等一套完整的消息云服务。

阿里云流计算对接ONS服务当前仍在开发中，敬请期待。

29.2.7 应用场景

流计算使用 StreamSQL 主打流式数据分析场景，目前在如下领域有使用场景：

应用场景



- 实时 ETL：集成流计算现有的诸多数据通道和 SQL 灵活的加工能力，对流式数据进行实时清洗、归并、结构化。作为离线数仓有效的补充和优化，作为数据实时传输的可计算通道。
- 实时报表：实时化采集、加工流式数据源，实时监控和展现业务、客户各类指标，让数据化运营实时化。
- 监控预警：对系统和用户行为进行实时检测和分析，实时监测和发现危险行为。
- 在线系统：实时计算各类数据指标，并利用实时结果及时调整在线系统相关策略。在各类内容投放、无线智能推送领域有大量场景。

29.2.8 应用案例

29.2.8.1 天猫双十一大屏

每年的天猫双十一购物狂欢节已逐渐成为全球互联网最大规模的商业促销狂欢活动。而每年双十一节除了”买买买”外，最引人注目的就是天猫双十一大屏不停闪变跳跃的总体成交总额。这份实时化的大数据展示链路凝结阿里集团诸多顶尖级工程师长达数月辛勤劳作，其中关键指标也颇具亮点，包括从天猫交易下单购买到数据采集、数据计算、数据校验，最终落到双十一大屏上展现的全链路时间压缩在 5 秒以内，0 点顶峰计算性能高达数十万订单/秒，多条链路流计算备份确保万无一失。



天猫双十一大屏背后最强大最核心的支持力量来自于阿里云流计算。曾经天猫双十一大屏后台流式计算使用开源的 Storm 来进行开发，整个开发时间长达一个月。而后阿里数据事业部选择使用阿里云流计算 StreamSQL，将整个双十一大屏的开发周期缩短到一周，并且由于阿里云流计算底层完全屏蔽了故障处理、执行优化，最终上线的阿里云流计算作业比 Storm 更快、更高效。

下面是天猫双十一整体数据流图，天猫使用了阿里云流计算平台，让整体处理链路变得更加稳定和高效：

双十一整体数据流图



- **用户在线抢购**

海量的天猫用户使用天猫的购买系统进行双十一抢购活动。在天猫交易的高峰期，如11日凌晨0点的诸多秒杀活动，整个流计算的面临的顶峰性能可能高达数十万订单/秒。

- **数据实时采集**

  数据采集系统将数据库变更日志采集并进入 DataHub 系统。使用阿里集团集团内部的数据库变更日志复制计算（阿里云对应的产品名称为*DTS*），在线交易数据库即使在零点最高峰交易时段，也可以保证在数秒内将数据写入 DataHub。

- **数据实时计算**

  阿里云流计算订阅上述 DataHub 的流式数据，持续读取并计算截止到当前时间点的天猫成交总和。阿里云流计算单机群在数千台规模，支持吞吐量可达百万级/秒，应对天猫的数十万笔交易/秒自然不在话下。流计算实时订阅实时计算，并将结果数据即时写入在线的 RDS 系统中。

- **前端可视化展现**

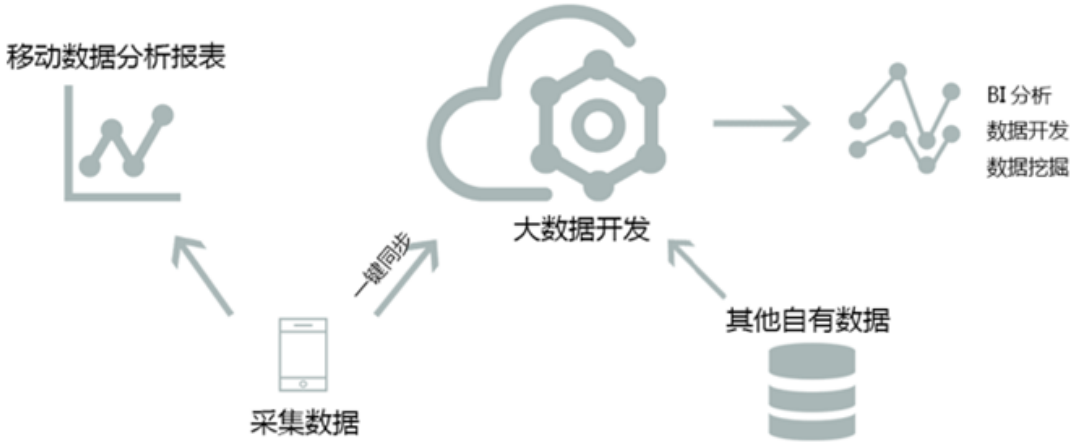
  前端数据可视化组件团队为天猫双十一大屏专门定制了翻牌器、全球交易热点等诸多酷炫内容。前端服务器将定期轮询RDS系统，并利用Web前端技术实现双十一大屏的震撼效果。

阿里云流计算团队正是是阿里大数据平台内部产品衍生而来，并支持天猫双十一活动长达数年之久，为天猫双十一活动立下汗马功劳。

29.2.8.2 无线数据分析

阿里云流计算同样可以为无线 APP 的数据分析场景助力实时化，帮助我们的客户做到实时化分析手机 APP 的各项指标，包括 APP 版本分布情况、Crash 检测和分布等等。阿里云移动数据分析（Mobile Analytics，简称 MAN）是阿里云推出的一款移动 APP 数据统计分析产品，提供通用的多维度用户行为分析，支持日志自主分析，助力移动开发者实现基于大数据技术的精细化运营、提升产品质量和体验、增强用户黏性。阿里云 MAN 其底层大数据计算全部基于阿里云的大数据产品，包括使用流计算、MaxCompute 等。在流式处理部分，MAN 采用阿里云流计算作为底层的流式大数据分析引擎，为移动数据分析的客户提供一整套实时化的移动 APP 分析报表服务。如下图：

移动 APP 分析报表服务



MAN 目前在阿里云上用户数已经超过数百家，结合阿里云大数据平台（数加），未来 MAN 将会开放更多实时化、个性化，甚至自定义分析逻辑的无线分析功能给用户，可以极大拓展整个产品功能覆盖面。MAN 现有全流程的数据流如下：

全流程的数据流



- **数据采集**

  开发者选择使用”阿里云移动数据分析”提供的SDK，将其内置到其 APP的应用安装包中。该 SDK 会针对不同手机操作系统提供数据采集组件，将负责将用户手机数据、行为数据采集进入 MAN 的后台系统进行数据分析。

- **数据上报**

  MAN 的后台提供了一整套 SDK 数据上报服务，将负责收集使用 SDK 的手机上报的数据信息。上报服务系统将进行简单去噪处理后直接投送到 DataHub。

  说明: 未来 DataHub 将直接提供手机端数据上报 SDK，实际上 MAN 后台的上报服务环节可以完全省略（将去噪处理也可以移到流计算处理），进一步减少 MAN 的机器成本。

- **流式计算**

阿里云流计算将持续订阅上述 DataHub 的流式数据，持续读取并计算各类 APP 指标，并立即将各时间段结果数据写入在线的 RDS/Table Store 系统中。

- **数据展现**

  MAN　提供一整套的运营指标体系，让开发者快速了解用户来自哪里，访问了哪些页面，停留了多长时间，用户终端及网络环境如何，应用程序卡顿或崩溃的实时反馈，其中 Crash 分析能精确到设备粒度，查看具体设备的详细 Crash 信息。其中的实时数据来源即来自于上述流式计算的结果。

  数据展现



  以上数据采集自线下 DEMO 数据，不代表真实数据情况。

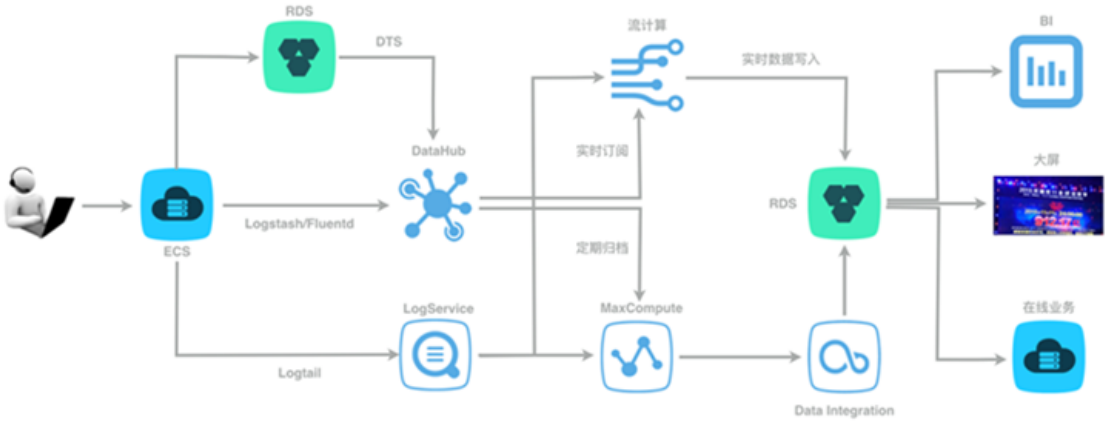
29.2.8.3 电商运营活动

阿里云流计算脱胎于阿里集团电商行业大数据架构，可以说天生适合电商行业各类流式数据分析和报表支持。从电商行业来看，对于流式数据实时处理需求主要集中在：

- **对于用户行为的实时分析**，例如交易大屏、用户大屏等一系列用户行为大屏展示。使用传统的离线分析不仅整体速度慢时延高，而且可能对于在线系统存在一定压力，从而系统稳定性得不到完全保证。
- **对于用户、业务、系统的实时监控**，例如全站交易时段曲线图可以协助运营或者技术人员了解当前时间点全站交易情况，如果交易出现不正常波动（例如突然下跌），应该立刻触发报警机制以方便相关人员介入排查问题，以减少交易波动对于公司业务的巨大影响。
- **对于一些大促活动的实时监控**，例如双十一、618等电商大促，运营人员需要实时获取到各类指标信息，用以迅速决定是否需要更换大促运营方案。

使用阿里云流计算，结合阿里云各类计算、存储系统，可以方便支持上述各类个性化的流式数据分析需求。不同于其他数据分析系统，阿里云流计算既满足业务灵活性，同时又采用 SQL 保证业务开发的低门槛。

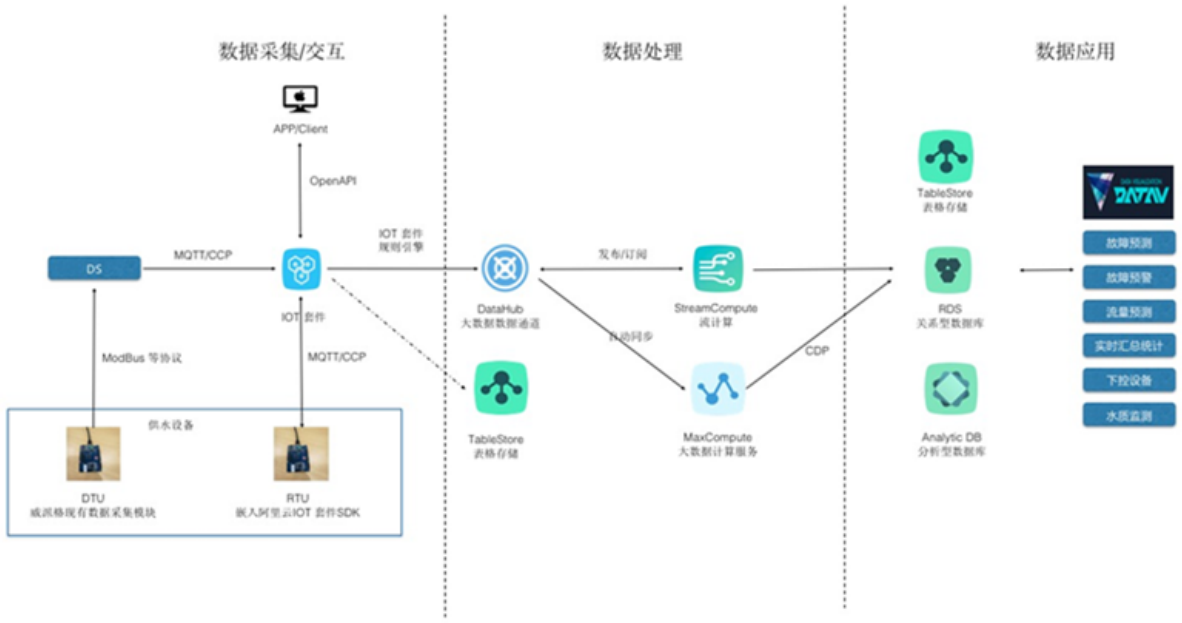
图 976: 流式数据实时处理



29.2.8.4 物联网设备监控

在 IOT 领域，由于存在大量的实时数据流（物联网场景下传感器产生的数据流更多），同时核心关键业务对于数据监控的时效性要求非常高，在这类领域流计算、实时计算的应用场景将非常广泛。阿里云流计算提供全链路的流式数据加工、生产服务，极大方便 IOT 开发者能够迅速搭建一整套完整的实时流数据监控、分析系统，助力 IOT 的实时大数据业务发展。

图 977: 物联网设备监控



在 IOT 领域，由于存在大量的实时数据流（物联网场景下传感器产生的数据流更多），同时核心关键业务对于数据监控的时效性要求非常高，在这类领域流计算、实时计算的应用场景将非常广泛。阿里云流计算提供全链路的流式数据加工、生产服务，极大方便 IOT 开发者能够迅速搭建一整套完整的实时流数据监控、分析系统，助力 IOT 的实时大数据业务发展。

29.3 使用准备

29.3.1 开通流计算服务

专有云用户请联系您的系统管理员予以开通。

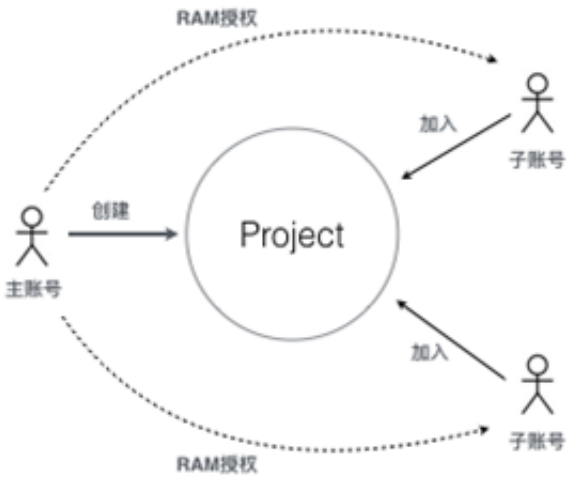
29.3.2 账号管理

29.3.2.1 账号模型介绍

阿里云流计算账号系统目前仅支持阿里云账号体系，且支持RAM 主子账号模型。用户可以使用阿里云主账号购买和创建流计算项目，同时亦可授权创建和授权子账号使用上述创建的流计算项目。

特别注意，不同于 MaxCompute （包括 MaxCompute 开放平台 Base）自建复杂的账号权限体系，流计算按照标准的阿里云产品规格提供账号信息支持，即一个项目隶属于一个 Owner，多人协作模型必须使用主子账号完成。如下图：

图 978: 多人协作模型



29.3.2.2 主子账号使用流程

阿里云流计算支持 RAM 主子账号模型，可以有效做到账号管理和权限分权，有关 RAM 主子账号介绍的请参考 RAM 相关文档。下面讲解流计算使用主子账号流程。

初始化 RAM ，第一次使用 RAM 需要进行初始化工作，点此查看 *RAM* 初始化说明。

创建子账号，该账号后续就分配给流计算的开发人员进行使用，点此查看 *RAM* 创建子账号。

添加自定义权限策略，用户需要在 RAM 中添加自定义的权限策略，点此查看如何添加。

自定义权限策略。对于流计算策略授权，如下图所示：

图 979: 创建授权策略

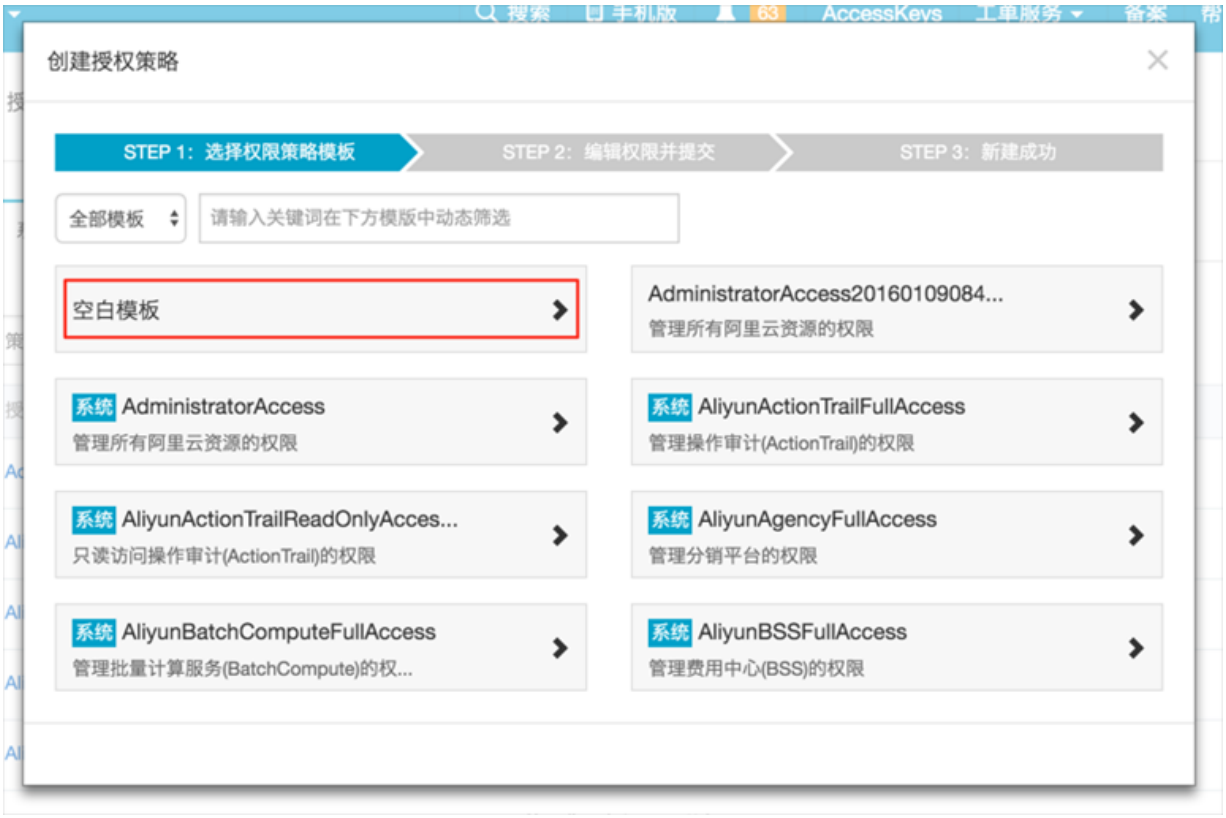

创建授权策略
STEP 1：选择权限策略模板
STEP 2：编辑权限并提交
STEP 3：新建成功
全部模板
请输入关键词在下方模版中动态筛选
空白模板
AdministratorAccess20160109084...
管理所有阿里云资源的权限
系统 AdministratorAccess
管理所有阿里云资源的权限
系统 AliyunActionTrailFullAccess
管理操作审计(ActionTrail)的权限
系统 AliyunActionTrailReadOnlyAcces...
只读访问操作审计(ActionTrail)的权限
系统 AliyunAgencyFullAccess
管理分销平台的权限
系统 AliyunBatchComputeFullAccess
管理批量计算服务(BatchCompute)的权...
系统 AliyunBSSFullAccess
管理费用中心(BSS)的权限

图 980: 策略授权

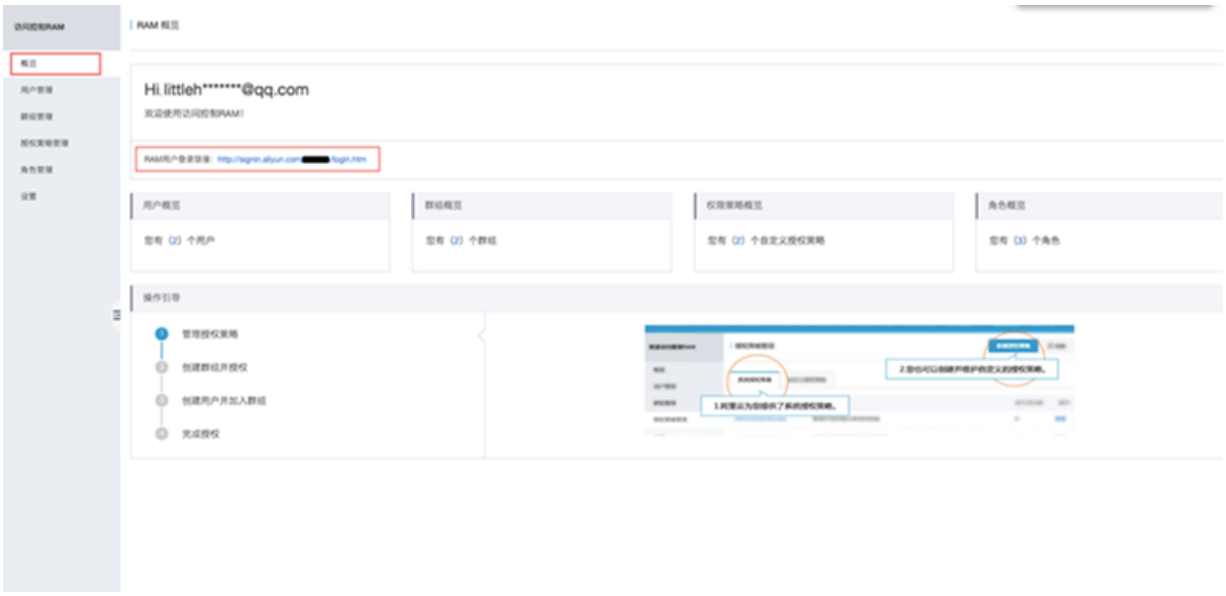


对于流计算授权策略代码为：

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": "stream:*",
      "Resource": "acs:stream:*:*:*",
      "Effect": "Allow"
    },
    {
      "Action": "ram:PassRole",
      "Resource": "acs:ram:*:*:*",
      "Effect": "Allow"
    }
  ]
}
```

将上述权限策略授权给指定用户或者用户组，点此查看如何进行授权。

将 RAM 登录地址发给需要登录的子账号用户，进行登录。在 RAM 管理控制台首页的概览信息可以查看登录地址，如下图：



29.3.2.3 对子账号分权

如果需要对子账号分 Project 管理给予不同的授权策略，请在上述针对子账号的授权加入 Project 的限定即可：

```
{
    "Version": "1",
    "Statement": [
      {
        "Action": "stream:*",
        "Resource": "acs:stream:*:*:这里填写流计算Project的名称",
        "Effect": "Allow"
      },
      {
        "Action": "ram:PassRole",
        "Resource": "acs:ram:*:*:*",
        "Effect": "Allow"
      }
    ]
  }
```

29.3.3 自动化角色授权流程

在用户开通流计算服务时，需要授予一个名称为**AliyunStreamComputeDefaultRole**的系统默认角色给流计算的服务账号，当且仅当该角色被正确授予后，流计算才能正常地调用相关服务，例如 RDS 等。

注意: 流计算必须提前进行角色授权，否则后续不能正常运行！

当新用户创建项目或第一次进入项目时，如果没有正确的授予默认角色给流计算服务账号，会看到如下视图，此时请单击**前往RAM进行创建**链接，进行角色授权。

图 981: 流计算角色授权



当点击 **前往RAM进行创建** 链接后跳转到的视图如下，点击**同意授权**按钮后，即完成了将默认角色**AliyunStreamComputeDefaultRole**授予给流计算 服务账号。此时即完成了授权的全部内容。

图 982: 云资源访问授权



当完成以上授权步骤后，重新刷新流计算的控制台，就可以进行操作了。如果想要查看**AliyunStreamComputeDefaultRole** 相关的详细信息，可以登陆Ram的控制台查看，也可以点击查看链接。

流计算角色当前授权的权限如下：

```
{
 "Version": "1",
 "Statement": [
   {
     "Action": [
       "ots:List*",
       "ots:DescribeTable",
       "ots:Get*",
       "ots:*Row"
```

```
    ],
    "Resource": "*",
    "Effect": "Allow"
  },
  {
    "Action": [
      "dhs:Create*",
      "dhs:List*",
      "dhs:Get*",
      "dhs:PutRecords",
      "dhs:DeleteTopic"
    ],
    "Resource": "*",
    "Effect": "Allow"
  },
  {
    "Action": [
      "rds:Describe*",
      "rds:ModifySecurityIps*"
    ],
    "Resource": "*",
    "Effect": "Allow"
  },
  {
    "Action": [
      "log:List*",
      "log:Get*",
      "log:Post*"
    ],
    "Resource": "*",
    "Effect": "Allow"
  },
  {
    "Action": [
      "mns:List*",
      "mns:Get*",
      "mns:Send*",
      "mns:Publish*",
      "mns:Subscribe"
    ],
    "Resource": "*",
    "Effect": "Allow"
  },
  {
    "Action": [
      "drds:Describe*",
      "drds:ModifySecurityIps*"
    ],
    "Resource": "*",
    "Effect": "Allow"
  }
 ]
}
```

29.4 热词统计

热词统计分析在搜索热词、论坛热词、标签热词等场景有非常广泛的应用，例如微博搜索的实时热词统计，可以方便引导用户了解微博上最新最火的热词。热词统计分析实际上就是一个简单的 WordCount 任务，而流式实时热词统计分析将 WordCount 处理逻辑整体转换为流式实时处理，可

以做到实时对热词进行统计分析，并可以实时展现。下面我们以 WordCount 流式任务为例，讲解下如何编写第一个流式计算任务。

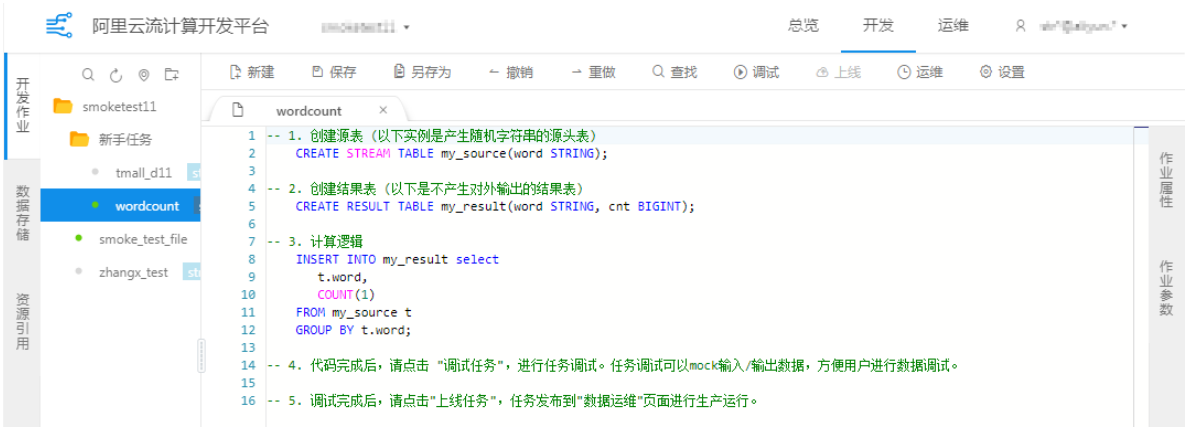
Q：什么是 Wordcount？

A：大数据的 wordcount 任务好比编程教学中的 "Hello World"，通常均作为新手用户必不可少的入门任务。下面就以阿里云流计算的 wordcount 为例，讲解如何开发一个流式版本的 wordcount。新手用户通过 wordcount 任务，可以学习基本的 StreamSQL 语法格式，以及任务编写/发布基本操作。

29.4.1 代码开发

1. 进入阿里云流计算产品首页，点击头部导航栏**开发**页签，进入数据开发的 IDE页面。
2. 打开阿里云流计算内置的名称为**新手任务**文件夹，双击打开**wordcount**任务，如图 *983: wordcount任务*所示。

图 983: wordcount任务



流计算的**wordcount**任务和批量任务原理基本一致，仅仅是在于流式的**wordcount**数据源是持续且无界的，因此流计算**wordcount**理论上除非用户显示终止（Kill），否则不应该停止运行。

这段 StreamSQL 代码讲解如下：

```
1  create stream table stream_source(word string);
2  create result table stream_result(word string, cnt bigint);
3
4  insert into stream_result select
5     t.word
6     ,count(1)
7   from stream_source t
8   group by t.word;
```

代码第 1 行，我们声明引用一张流式数据表，该数据表名称为 stream_source，内部仅包含一个类型为 string，名称为 word 的列。

说明: 如前所述，流计算的数据驱动源来自于流式数据。因此这里的 stream_source 就是数据驱动源，stream_source 每条（批）数据均会触发下游流计算的一次计算。

代码第 2 行，我们声明引用一张结果表，用来存放我们的 wordcount 计算结果。该数据表名称为 stream_result，内部包含一个类型为 string，名称为 word 的列，另加一个类型为 bigint，名称为 cnt 的列。

说明: 如前所述，流计算本身不带有任何数据存储，所有的结果数据存储理论上均为普通的 RDS、Table Store 等存储系统。我们这里声明引用一张结果表，为计算的结果数据存储所用。

代码从第 4 行开始，进入正式的 wordcount 计算逻辑，熟悉 SQL 的用户应该很清楚这段 SQL 的含义：从 stream_source 表读取数据，针对每条进入的数据，统计各个 word 出现的次数/频度。

说明: 为了尽量减少用户学习流计算的成本，阿里云流计算提供的 StreamSQL 和 SQL 标准格式基本一致，最大限度降低用户学习门槛。

29.4.2 代码调试

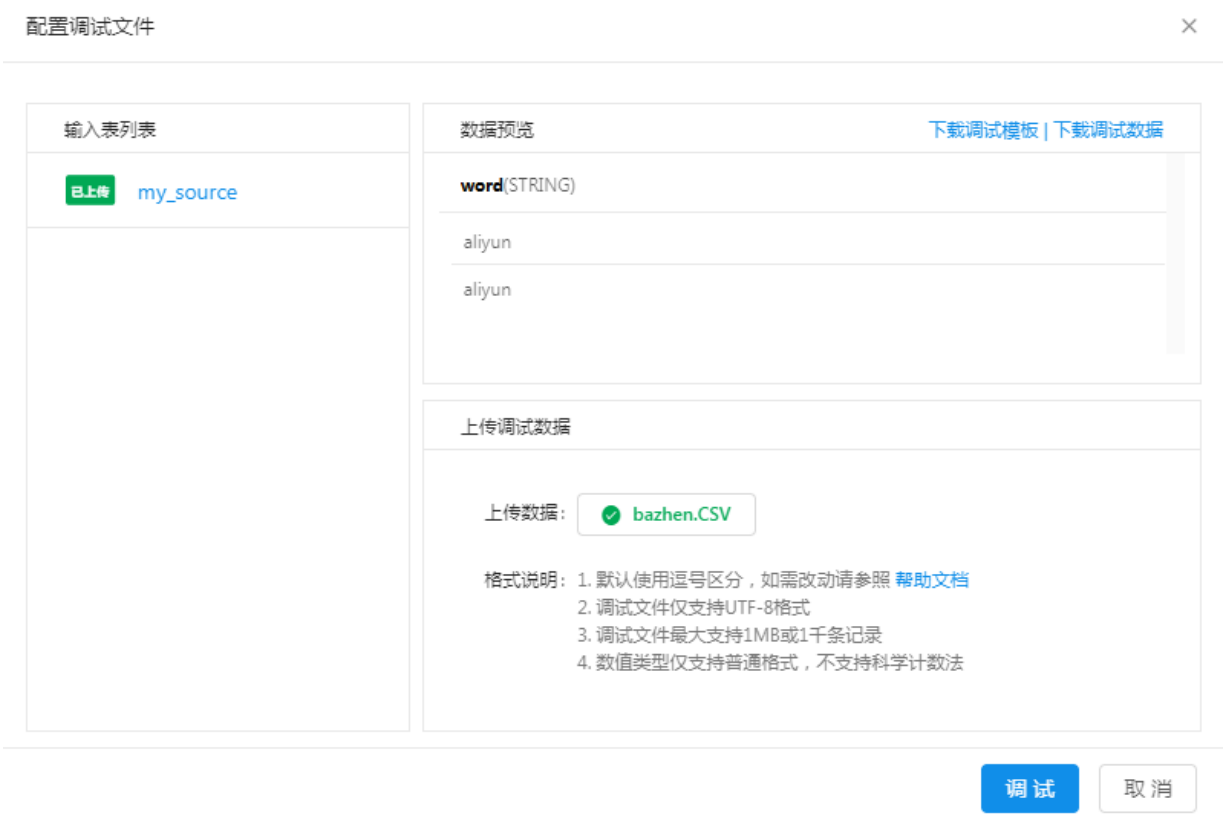
为方便用户调试 StreamSQL，阿里云流计算提供了在线调试 /Debug 功能，方便用户构造调试数据并方便的进行回归测试。阿里云流计算提供的调试功能非常强大，可以将流式存储、静态存储、结果存储全部进行模拟调试，方便用户构造各类数据进行 SQL 正确性验证。

说明:

- 为防止对线上存储系统读写影响，流计算调试过程要求所有的输入表必须全部提供测试数据，不允许读取线上存储系统。
- 所有写入（insert）操作亦是在屏幕打印，不会对线上系统造成影响。

用户点击**数据开发**IDE页面上的**调试**，启动**调试任务**。在该任务初次启动调试时，系统会弹窗提示上传测试数据文件，如下：

图 984: 上传调试数据

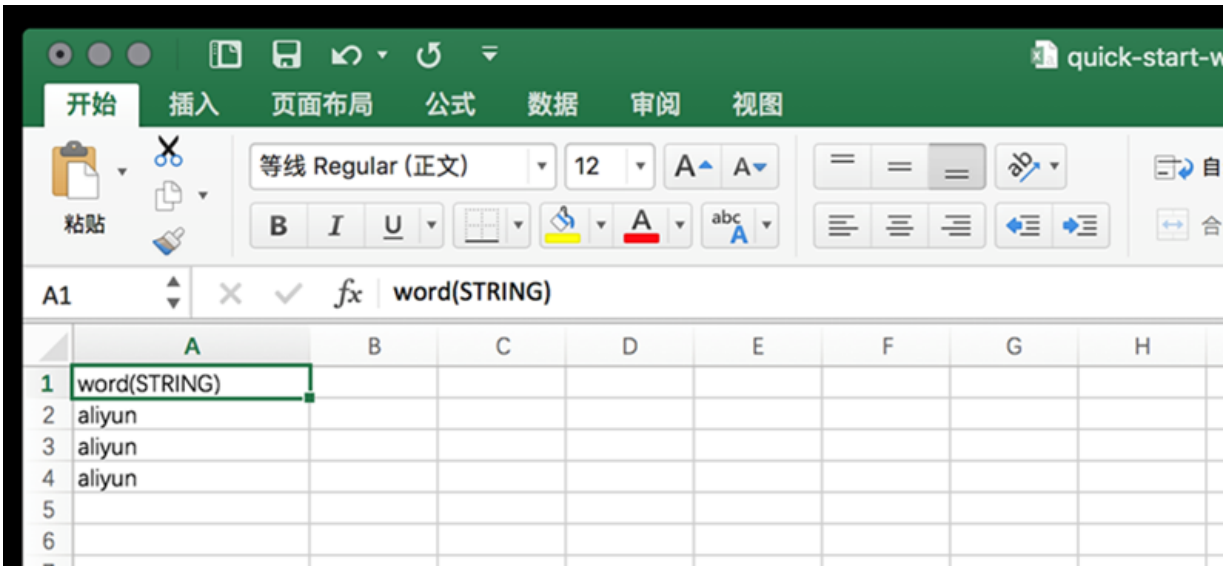


上图左侧是当前 StreamSQL 用户声明的数据表引用清单，包括流式输入表、静态维表的清单，用户需要为每个输入表提供调试数据方能够进行调试运行。为了方便用户进行数据调试，阿里云流计算针对每个表自动构建了数据模板，并在自动构建的模板文件第一行附带提供了字段名、字段类型的Title，格式为**字段名**（字段类型）。用户可以根据Title的提示信息进行修改自己需要的测试数据。在调试页面用户点击**下载调试模板**并根据自己的测试策略填写调试数据。流计算对于上传的调试数据有较严格的定义：

- 调试数据文件最多支持 1MB 或者1K 条记录。
- 调试文件仅支持 UTF-8 格式。
- 采用逗号分隔的 CSV 格式，避免内容出现逗号。
- 数值类型仅支持普通格式，不支持科学计数法。

因为流计算使用的是 CSV 作为调试文件类型，对于下载的文件，我们推荐 Windows 平台下用户使用 Excel 软件，Mac 平台下使用 VIM/Sublime 软件打开模板数据进行修改（**Mac下 Number 工具修改 CSV 会增加很多字段信息，不推荐使用 Number!**），wordcount 的测试模板样例如下：

图 985: wordcount 测试模板样例



新手任务为了方便用户尽快上手，我们已经提供了一份测试数据，用户点击可以直接下载该样本数据并通过测试界面上传。

说明: PDF 版本文档中《热词统计测试数据》无法通过链接下载，流计算作为附件文件打包提供了测试数据，请咨询您的系统管理员索要测试数据。

点击**调试**按钮，流计算会立刻启动一个测试流计算任务进行调试。测试任务将直接使用用户提供的测试数据进行运行，并且最终的测试结果会直接在屏幕输出，如下：



对于流计算而言，计算驱动源来自于流式数据触发；在测试状态下，数据源stream_source每条数据将会直接触发一次流式计算并产出计算结果，因此我们看到测试文件有三条数据，每条导致一次计算，所以结果展示页面亦是有三条数据，其运算轨迹分别如下。

第一行源头数据（数据为aliyun）到达流计算，此时流计算发现之前不存在aliyun单词，因此计算结果为<aliyun, 1>，输出屏幕。

第二行源头数据（仍然为aliyun）到达流计算，此时流计算检测发现已经存在<aliyun, 1>的记录，因此将该值+1，得出结果为<aliyun, 2>，输出屏幕。

第三行源头数据（仍然为aliyun）到达流计算，此时流计算检测发现已经存在<aliyun, 2>的记录，因此将该值+1，得出结果为<aliyun, 3>，输出屏幕。

最终我们观察结果是以最后一条产出结果为准，即<aliyun, 3>代表我们本次调试数据最终产生的结果。另外我们提供另一份[测试数据]，方便大家在不同 word 测试数据下，观察调试界面输出情况。

29.4.3 数据运维

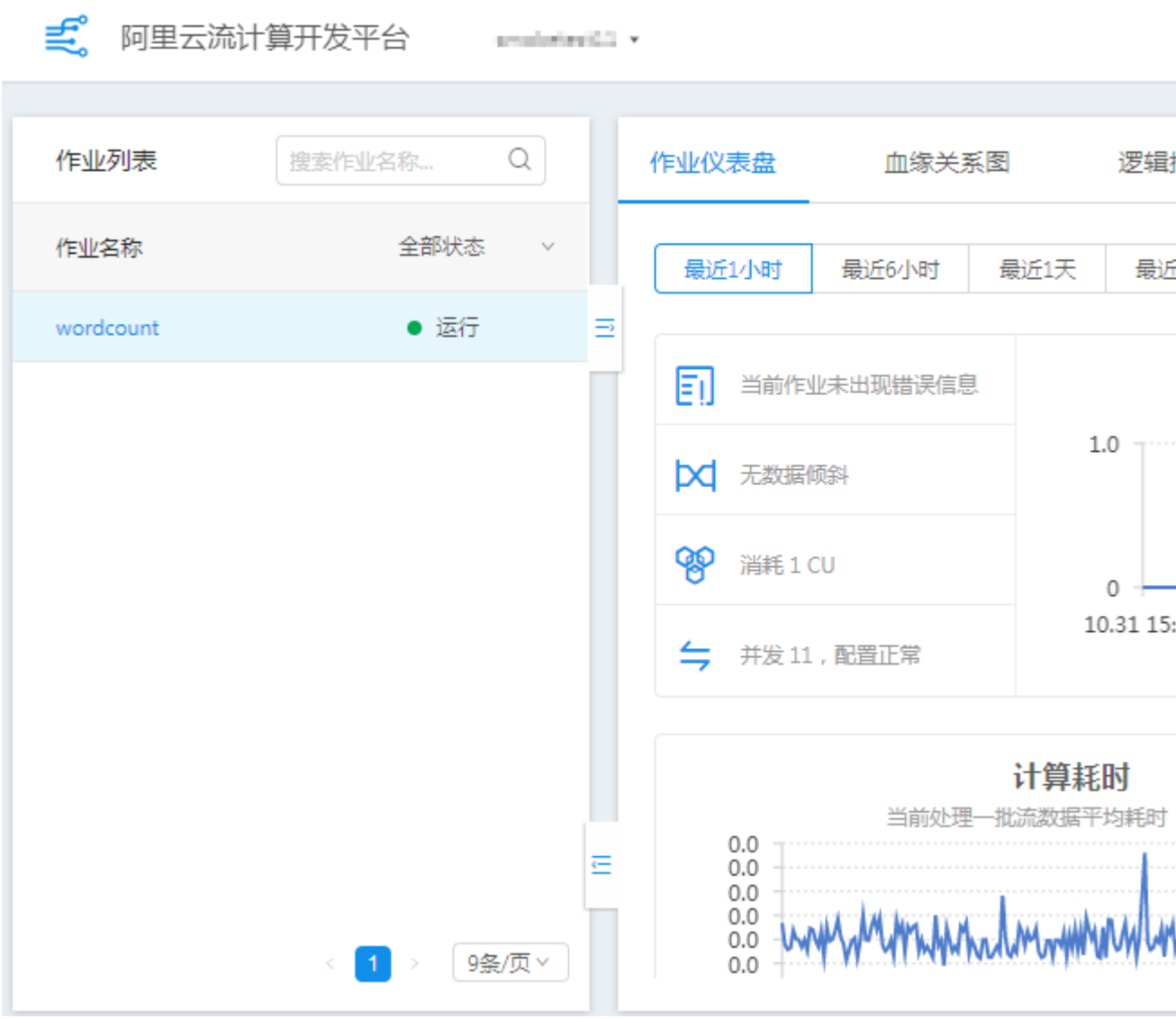
代码测试完毕经验证准确无误后，即可将其发布到**数据运维**模块，提交任务进入流计算集群进行生产运行。

操作步骤

1. 在**开发**页面单击**上线**按钮，弹出**上线新版本**窗口。
2. 输入上线注释后，单击**上线**按钮，上线新版本成功。
3. 点击**运维**页签，在作业列表中可查看新上线的 wordcount 任务。
4. 单击 wordcount 任务对应操作列的**启动**按钮，弹出**启动作业**窗口。
5. 选择**指定数据读取时间**后，单击**按以上配置启动**按钮，流计算任务即可被生产集群调度起来。

   一旦任务启动成功，任务的界面将变为绿色，同时会有流动线条提示计算正在运行，如图 *986: 任务启动*所示。

   图 986: 任务启动



可能您会关心：既然已经在分布式流计算集群运行起来了，为何这个计算任务即没有流式数据输入，也没有数据输出，那整体还能愉快地运行起来吗？原因就是我们在定义上述my_source、my_result表并未指定因外部引用数据源类型。因此，在这类未指定具体数据源类型的情况下，流计算将输入的 Stream 表视作内部随机产生字符串/数字的随机表，同时将输出的结果表视作直接丢弃数据。

29.5 天猫双十一大屏

双十一大屏是每年天猫双十一购物狂欢节的亮点应用，整个阿里集团对外实时交易总额的披露均是通过这块大屏完成。曾经天猫双十一大屏后台流式计算使用开源的 Storm 来进行开发，须考虑各种异常情况、故障情况，使得整个大屏开发时间长达一个月左右。而后阿里数据事业部选择使用阿里

云流计算 StreamSQL，将整个双十一大屏的开发周期缩短到三天，并且由于阿里云流计算底层完全屏蔽了故障处理、执行优化，最终上线的双十一大屏全链路实际上比 Storm 作业更快、更高效。

图 987: 天猫双十一大屏



29.5.1 问题描述

双十一的大屏流式数据来源是天猫的交易订单表。为简化问题，我们将源源不断的天猫交易数据抽象简化为如下二维表（tmall_trade_detail）：

字段名	类型	注释
tid	bigint	交易订单ID
buyer_uid	bigint	买家ID
seller_uid	bigint	卖家ID
gmtdate	timestamp	交易时间
payment	double	订单金额

针对上述流式数据表，我们需要计算两个指标，包括截止到当前时间点的交易总笔数以及交易总金额。这两个数据指标经过流计算写入在线的 RDS 系统，并通过大屏页面展示出来。RDS 的表设计如下（tmall_trade_state）：

字段名	类型	注释
gmtdate	varchar(16)	交易日期
trade_count	bigint	交易总数
trade_sum	double	交易总量

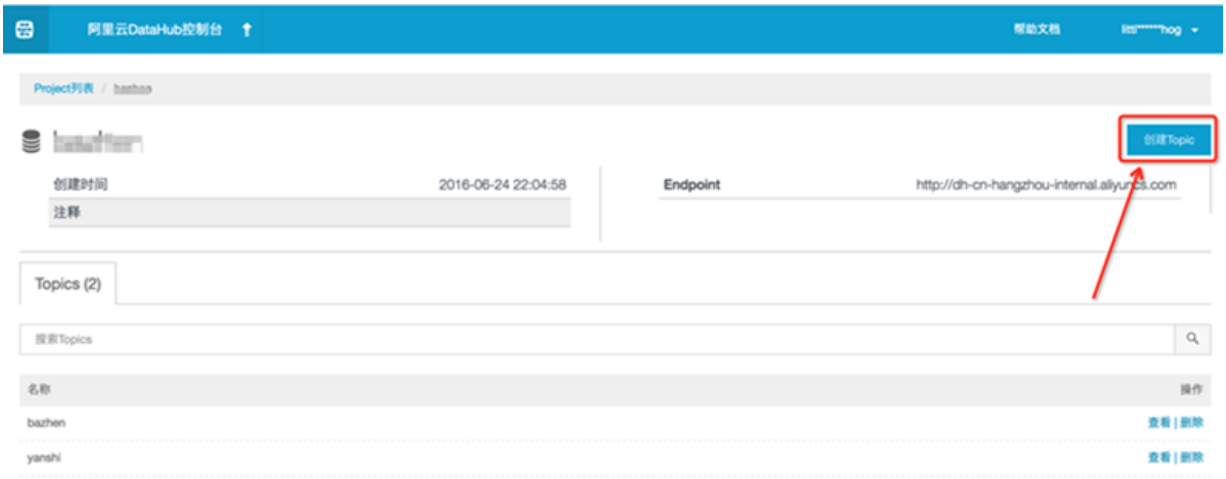
接下来，我们要做的是运用阿里云流计算，在10分钟之内搭一套全链路双十一大屏。

29.5.2 前置准备

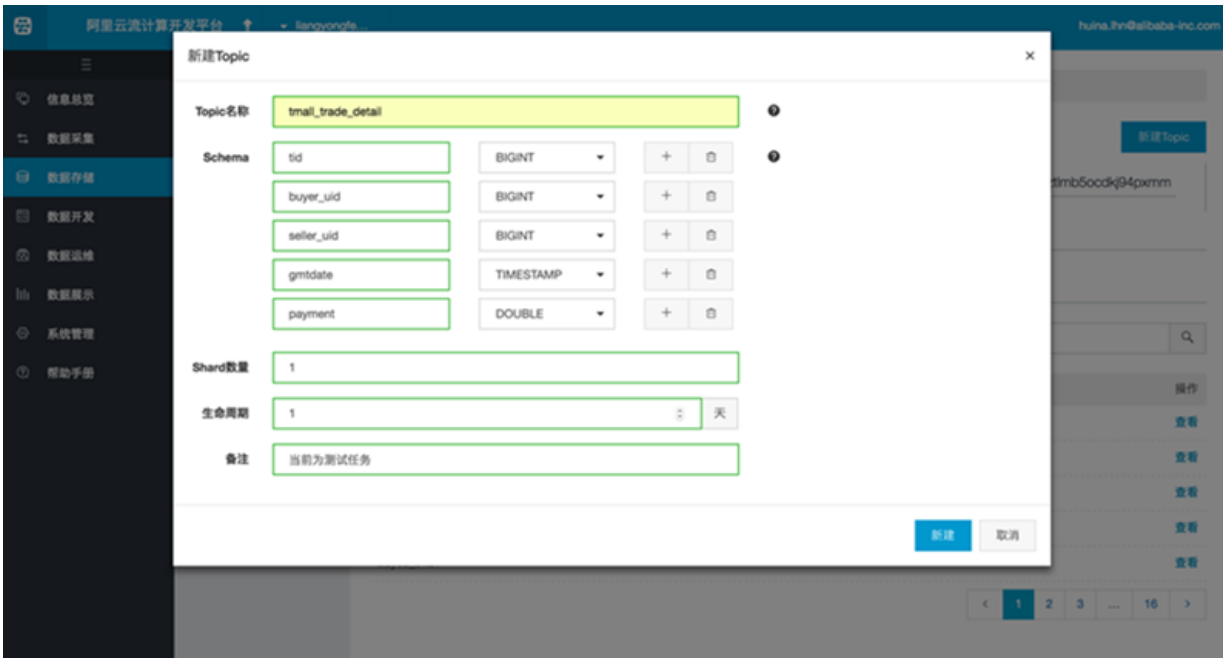
在流计算任务编写开始前，我们需要创建上下游相关存储系统，这里主要是 DataHub 和 RDS 两类存储系统，下面分别叙述。

创建 DataHub Topic

进入 DataHub WebConsole 界面，选择进入具体的 Project，点击**创建Topic**，进入 Topic 创建页面，如下图：



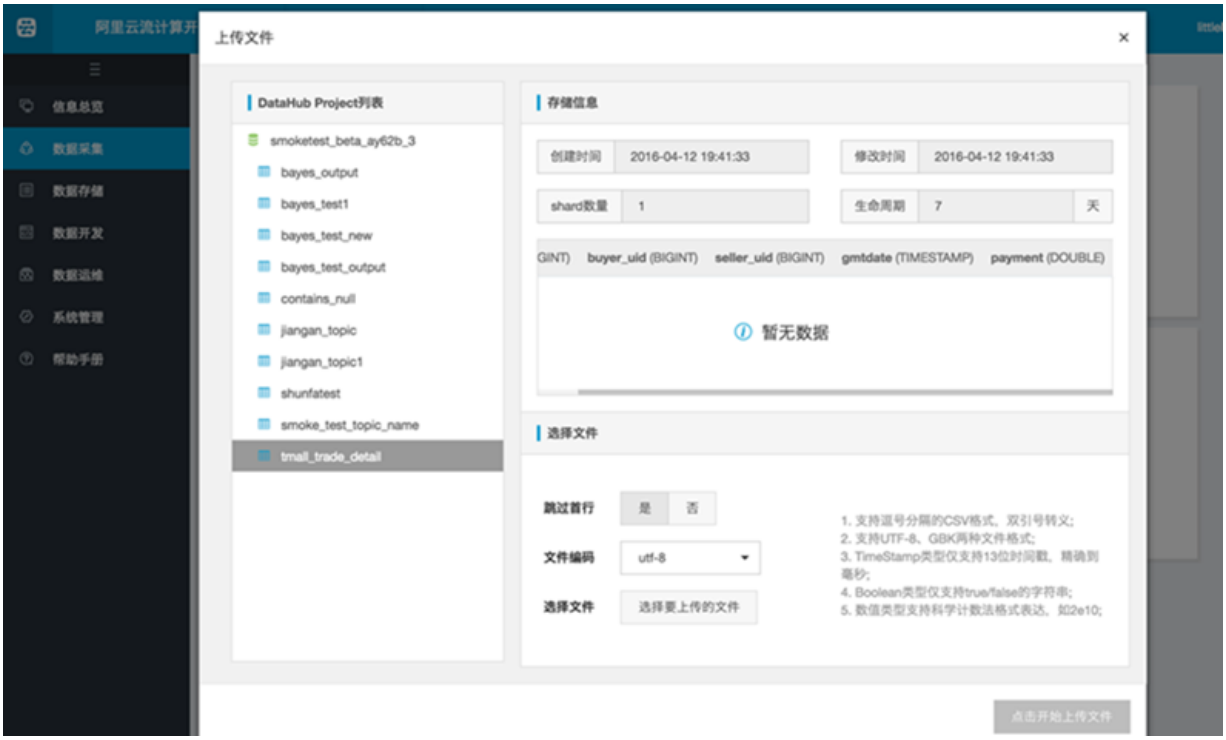
根据上述 RDS 表结构设计表，如下图：



准备就绪，开始进入实际的 SQL 编写了。

上传 DataHub 数据

在实际的流计算业务开发中，开发者第一个面临的问题就在于如何将数据上传进入 DataHub 系统。利用 DataHub 的数据上传功能用户可以轻松上传数据。进入[DataHub 界面]，选择上述创建完成的 DataHub Topic 进行数据上传。



用户点击侧边导航栏的**数据采集** ->**文件上传** 功能。这里为了简化用户的操作，我们提供双十一大屏案例数据，点击下载后利用 DataHub 的文件上传工具即可完成数据采集。



说明： 实际生产应用中，我们很少有机会利用文件上传工具进行数据采集工作。更多数据采集工具的使用请参考**使用手册**中有关**数据采集**的章节。

29.5.3 注册存储

使用阿里云流计算内置的数据存储功能可以方便的添加 DataHub 的Topic 信息、表的创建或者数据源的引用。

操作步骤

1. 在**开发**页面点击左侧菜单栏的**数据存储**页签。
2. 选择 **DataHub数据存储**后右击，在弹出的菜单中点击**注册数据存储**，将DataHub 的 Project 注册到流计算，具体参数配置可参考如下章节：*DataHub 的注册*。

   其中，DataHub 的连接信息为：Endpoint， 请联系管理员了解 DataHub 的 Endpoint，Project 填写管理员分配给您的 Project 名称。

   同样，我们选择使用 RDS（Mysql）为数据做可视化展现的存储，并在流计算注册 RDS 的连接信息。具体参考如 RDS 的注册。

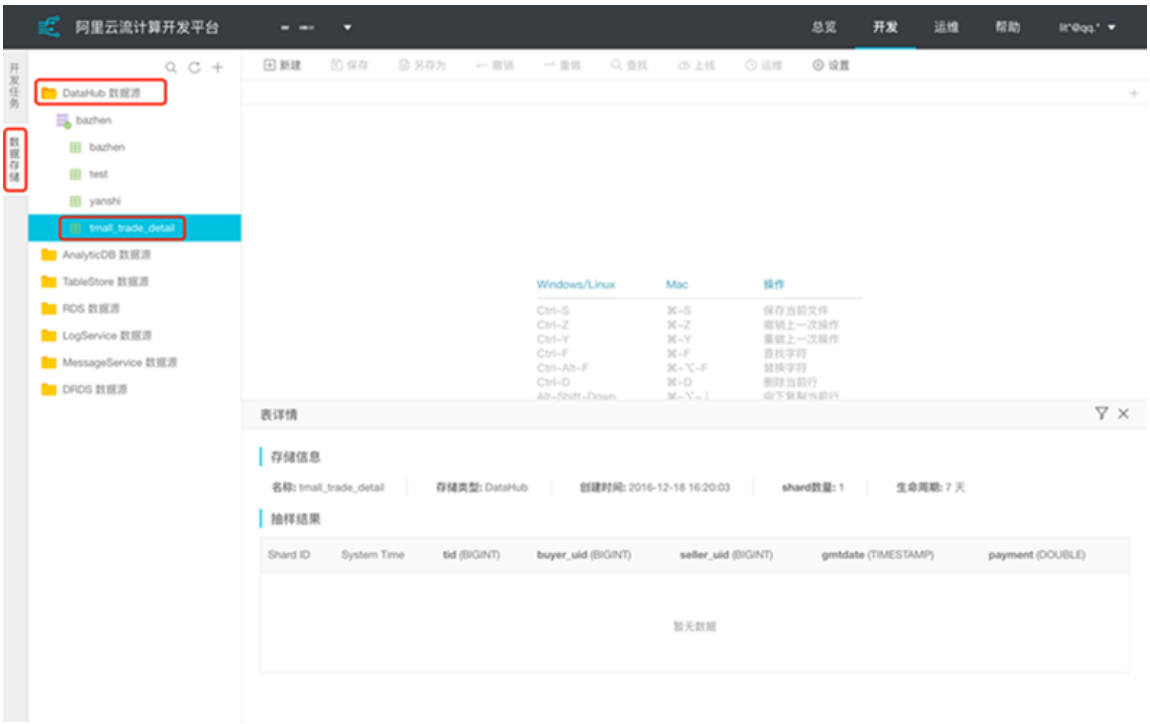
29.5.4 开发

一旦完成数据采集工作，我们就可以专心来研究 StreamSQL 的编写工作。同样，为方便新手用户上手流计算，我们在 **数据开发** 新手任务中提供了双十一大屏的教学任务：tmall_d11。点击这个任务，可以查看 StreamSQL 情况。

- **引用数据源**

  编写 SQL 第一步，我们应该为流计算声明数据的输入表（这里是 DataHub）以及数据的输出表（ RDS）。在**开发**页面选择**数据存储**，分别找到对应的 DataHub 的 Topic 和 RDS的表信息，选择性引用：

  - **DataHub 的 Topic 作为我们数据输入源**，点击选择**作为输入表引用**，流计算将自动解析 Topic 的 Schema，并自动添加对应的 SQL 到 IDE：



- **RDS 的表作为我们的数据结果输出**，点击选择**作为结果表引用**，流计算将自动解析 RDS 的表 Schema，并自动添加对应的 SQL 到 IDE。

- **编写 StreamSQL**

  如果我们严格按照上述教程说明的 Topic /表名称进行创建工作，那么 tmall_d11 任务中包含的 StreamSQL 已经能够直接运行，否则请根据您的实际建表情况调整中 tmall_d11 任务中有关 DataHub Topic、RDS Table 名称。代码如下：

```
replace into tmall_trade_state
   select
      from_unixtime(FLOOR(tmall_trade_detail.gmtdate/1000), 'yyyy-MM-dd') as gmt_date,
      count(tid) as trade_count,
      sum(payment) as trade_sum
   from
      tmall_trade_detail
   group by
      from_unixtime(FLOOR(tmall_trade_detail.gmtdate/1000), 'yyyy-MM-dd');
```

  说明: 以上表、字段信息请根据实际情况进行修改。

- **调试 StreamSQL**

  我们提供双十一大屏案例数据，下载后使用调试功能上传该测试数据进行数据调试。

- **任务上线**

调试完成后，经验证逻辑无误后，在数据开发中点击**上线任务**，用户即可完成任务上线工作。上线任务操作将当前用户的改动提交到数据运维中，用户即可在生产环境下进行任务启动等生产运维工作。

29.5.5 运维

在数据运维页面，选中上述**tmall_d11**这样的任务，点击启动后流式任务即可启动。

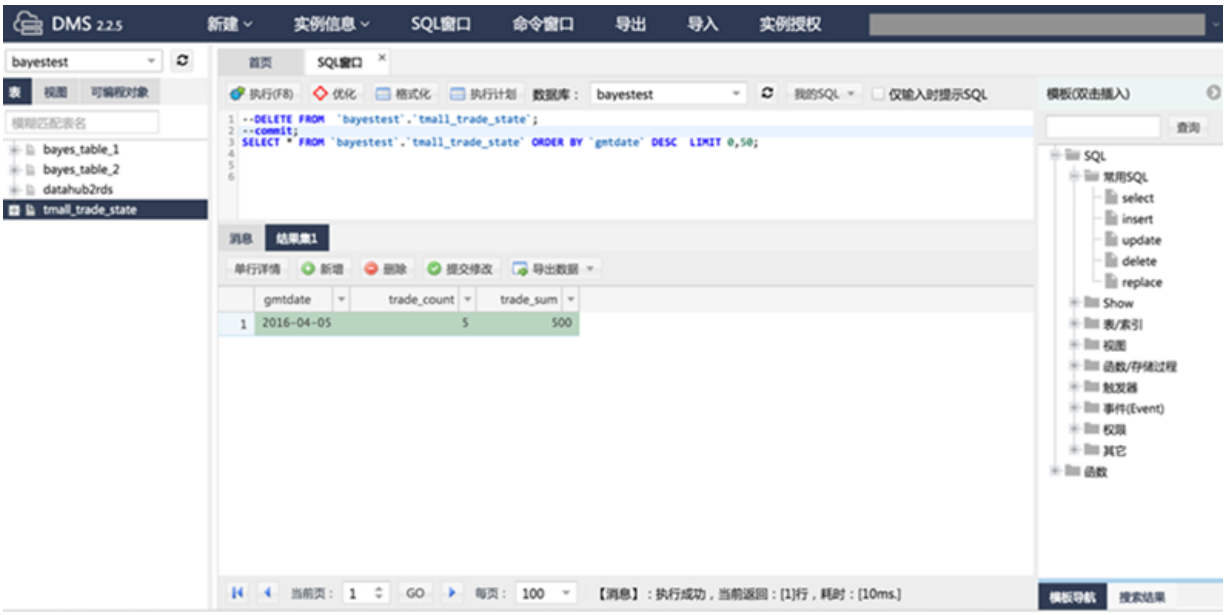
注意： 流计算任务启动时，需要您指定启动时间，实际上就是从源头数据存储的指定时间点开始读取数据。

这里我们需要指定的时间需要在上述上传之间，例如设置启动时间为一个小时之前。当前时间点是 13 点 50 分，前面数据上传时间是在 10 分钟前，因此我们这里设置启动时间为 13 点正。



流计算任务即可被生产集群调度起来。一旦任务启动成功，任务的界面将变为绿色，同时会有流动线条提示计算正在运行。

运行起来后，到 RDS 数据库去查看最终数据输出，我们的构造数据是 5笔交易，总交易额为500RMB，和最终流式计算结果一致。这样从端到端验证了我们业务代码的正确性。

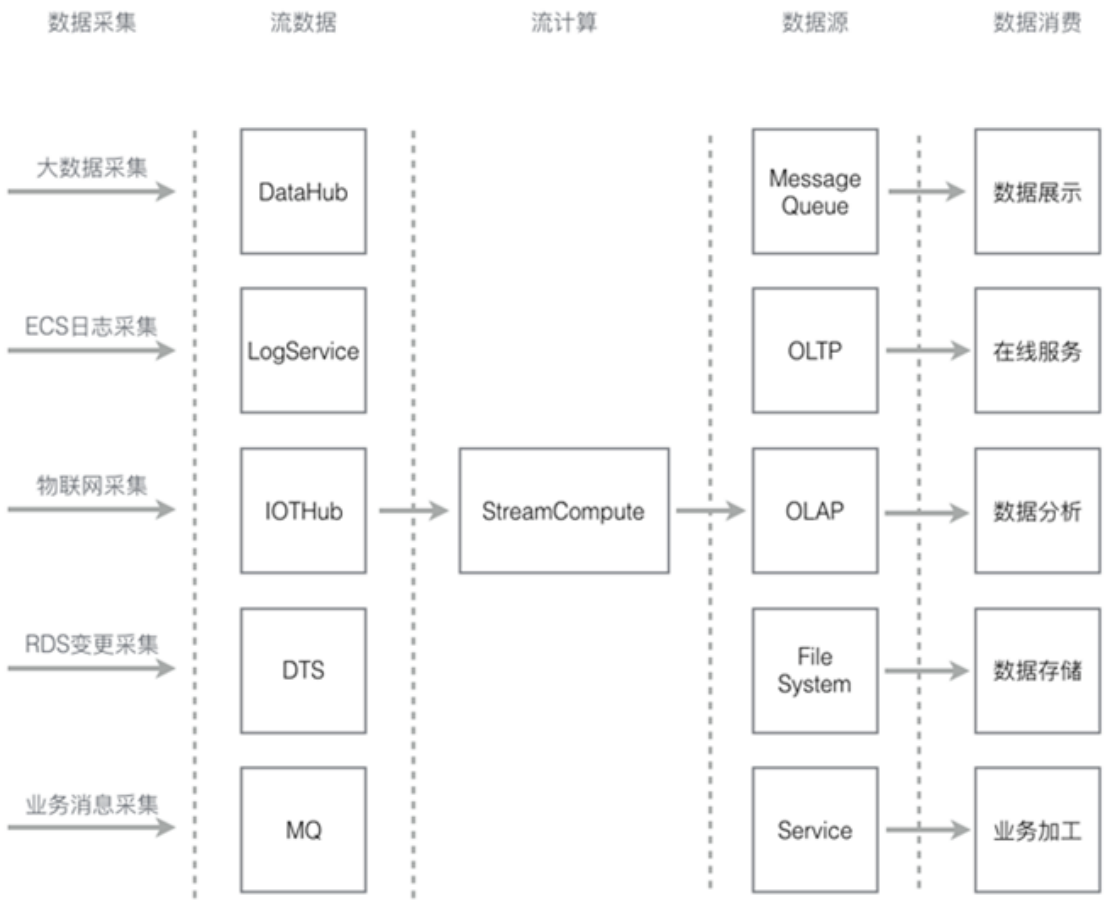


29.6 WebConsole手册

本章主要介绍阿里云流计算控制台的使用，从数据采集、数据存储、数据开发和数据运维等方面指导用户如何在云上进行流式数据实时化分析。

29.6.1 数据采集

所有的大数据分析系统都基于一个前提： 数据需要采集进入大数据系统。为最大化利用用户现有的流式存储系统，阿里云流计算对接了多种上游的流式存储，让用户可以不用进行数据采集、数据集成，即可享受现有的数据流式存储。



阿里云流计算支持上述若干个数据存储系统，针对上述不同的流式存储，需要用户使用不同的数据集成工具进行数据采集，流计算对接的有：

- DataHub

  DataHub 提供了多类数据（包括日志、数据库 BinLog、Iot 数据流等）从数据源上传到 DataHub 的工具、界面，以及和一些开源、商业软件的集成。

- 日志服务（LogHub）

  LogHub 是针对日志类数据一站式服务，在阿里巴巴集团经历大量大数据场景锤炼而成。Log 服务提供了诸多的针对日志的采集、消费、投递、查询分析等功能。

- 物联网套件（IOTHub ）

  物联网套件是阿里云专门为物联网领域的开发人员推出的，其目的是帮助开发者搭建安全性能强大的数据通道，方便终端（如传感器、执行器、嵌入式设备或智能家电等等）和云端的双向通信。

- 使用 IotHub 规则引擎可以将 IOT 数据方便投递到 DataHub，并利用流计算和 MaxCompute 进行数据加工计算。
- 数据传输（DTS）

  DTS 支持以数据库为核心的结构化存储产品之间的数据传输。它是一种集数据迁移、数据订阅及数据实时同步于一体的数据传输服务。使用 DTS 的数据传输功能，可以方便的将 RDS 等 BinLog 解析并投递到 DataHub，并利用流计算和 MaxCompute 进行数据加工计算。
- Message Service

  阿里云消息服务阿里云商用的消息中间件服务，具有大规模，高可靠、高并发访问和超强消息堆积能力的特点。流计算可以直接从消息服务读取流式数据。
- MQ

  阿里云 MQ 服务是企业级互联网架构的核心产品，基于高可用分布式集群技术，搭建了包括发布订阅、消息轨迹、资源统计、定时（延时）、监控报警等一套完整的消息云服务。

29.6.2 数据存储

29.6.2.1 存储概览

为方便用户管理数据源，阿里云流计算提供包括 RDS、Analytic DB、Table Store 等各类数据存储系统的管理界面。让用户无需跨越多种产品的管理页面，使用阿里云流计算平台，即可让您一站式管理您的云上数据存储。

注意:

- 流计算数据存储功能当前仅支持同账号属主下的存储资源，即当前使用流计算的 A 用户（包括 A 下所属的子账户），所注册的数据源必须是 A 购买的数据存储。当前不支持跨账号授权。
- 对于不属于当前用户的资源使用，用户可以直接使用 AccessId/AccessKey 在 DDL 定义语句中直接写出。

数据源注册需要提前授权，请参见角色授权。

29.6.2.2 存储类别

流式存储

流式存储为下游流计算提供数据驱动，同时也可以为流计算作业提供数据输出。

支持情况	输入	输出
DataHub	支持	支持
日志服务（LogHub）	支持	支持
数据传输（DTS）	[1]	无法支持
物联网套件（IOT）	[2]	暂不支持
消息服务（Message Service）	暂不支持	支持，[3]
消息队列（Message Queue）	暂不支持	暂不支持



说明:

- DTS 本身无法和流计算直接打通，需要将DTS数据导入Dat aHub后，下游流计算和 MaxCompute 均可以无缝集成 DataHub。DTS 本身作为 RDS 的 BinLog 采集工具，是不支持数据写入。
- 物联网套件本身无法和流计算直接打通，需要将 IOT 数据导入 DataHub 后，下游流计算和 MaxCompute 均可以无缝集成 DataHub。IOT不支持数据写入。
- 目前仅支持以 JSON 格式写出到消息服务。
- 数据写入 MaxCompute 的路径可以先写入 DataHub，DataHub 使用 Connector 功能实现分钟级别归档到 MaxCompute。

静态存储

静态存储为流计算提供了数据关联查询，同时也可以作为流计算作业数据输出。

支持情况	维表	输出
分析型数据库（AnalyticDB）	不支持	支持
表格存储（TableStore）	支持	支持
RDS（Mysql/SqlServer/ PostgreSQL/Oracle）	支持	支持
DRDS	暂不支持	支持
OSS	暂不支持	暂不支持
MaxCompute	暂不支持	[1]



说明: 数据写入 MaxCompute 的路径可以先写入 DataHub，DataHub 使用 Connector 功能实现分钟级别归档到 MaxCompute。

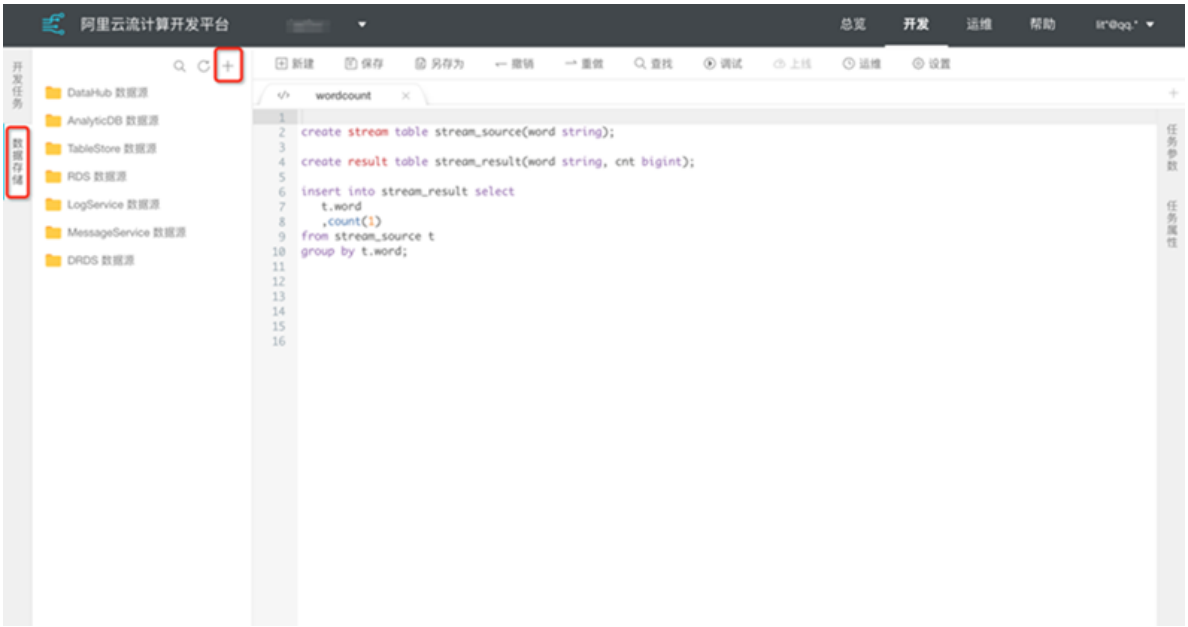
29.6.2.3 存储使用

注册的数据源将为以下使用场景提供服务，可以看到通过提前注册数据存储，用户能够享受到更多一站式流计算开发平台提供的便利性。

> **注意:** 对于不属于当前用户的资源使用，用户可以直接用 AccessId/AccessKey 在 DDL 定义语句中直接写出，此时用户无法界面化操作数据源，但是作业是可以直接运行的。

- **数据注册**

  用户第一次在未注册任何数据存储前无法使用数据存储提供的各项功能，用户必须**先注册流计算需要的的相关数据存储信息才能够使用**，用户进入**开发**界面，在左侧边工具栏点击**数据存储**，单击**+**，进入数据存储注册页面，如下图所示：



  再次提醒，**流计算数据存储功能当前仅支持同账号属主下的存储资源，即当前使用流计算的A用户（包括A下所属的子账户），所注册的数据源必须是A购买的数据存储。当前不支持跨账号授权。**

- **数据预览**

  流计算为每个已经注册的数据存储提供了数据预览功能，点击”数据存储”，选择某个数据存储类型，即可预览数据。以DataHub为例讲解下数据预览功能：

  1. 选择**数据存储 ->DataHub存储** -> 选择具体的 Project -> 选择具体需要预览的 Topic，双击即可进入查看数据源。
  2. 选择高级查看，可以针对数据源进行指定的抽样预览，例如针对 DataHub 可以设置时间查询预览具体的数据。

说明: 部分二进制存储系统不提供数据源预览功能，例如消息服务底层实际上是无结构化存储，当前流计算不提供数据预览功能。

- 自动生成 DDL

  流计算在引用外部存储时候，需要提前对于外部存储进行声明工作，例如对于声明一个流式输入引用：

```
CREATE STREAM TABLE tmall_trade_detail (
    tid                 BIGINT,
    buyer_uid           BIGINT,
    seller_uid          BIGINT,
    gmtdate             BIGINT,
    payment             DOUBLE
) WITH (
    type='datahub',
    endpoint='http://dh-cn-hangzhou-internal.aliyuncs.com',
    accessId='',
    accessKey='',
    projectName='smoketest_beta_ay62b_3',
    topic='tmall_trade_detail'
);
```

  流计算要求声明的表字段名称与源 DataHub 表保持一致，类型需要根据两边的类型进行一定的转换。流计算提供了辅助生成 DDL 功能，帮助用户可以一键生成建表 DDL 语句。

  在数据开发页面，进入需要编辑的任务，点开侧边栏的**数据存储**工具箱。

  这里选择作为输入表进行引用，点击**作为输入表引用**。此时流计算系统会在当前光标界面生成上述 DDL 信息。

附：跨账号的资源引用

当前流计算界面不支持跨账号数据存储注册和使用，流计算数据存储功能当前仅支持同账号属主下的存储资源，即当前使用流计算的 A 用户（包括 A 下所属的子账户），所注册的数据源必须是 A 购买的数据存储。当前不支持跨账号授权。此时用户需要使用，可以考虑直接在 DDL 语句中手写外部数据引用。例如，如果用户 A 需要使用用户 B 的资源，那么他可以在 DDL 定义：

```
create stream table stream_source (
    name            STRING,
    value           STRING
) WITH (
    type='datahub',
    endpoint='http://dh-cn-hangzhou.aliyuncs.com',
    accessId='B用户授权的AccessId',
    accessKey='B用户授权的AccessKey',
    projectName='project',
    topic='topic'
```

29.6.3 大数据总线（DataHub）

DataHub 作为一个流式数据总线，为阿里云数加平台提供了大数据的入口服务。结合阿里云众多云产品，可以构建一站式的数据处理平台。流计算通常使用 DataHub 作为流式数据源头和输出目的端。同时，上游众多流式数据，包括 DTS、IOT 等均选择 DataHub 作为大数据平台的数据入口。

- 注册

图 988: 注册



- **Endpoint**

  填写 DataHub 的 Endpoint，不同的地域下 DataHub 有不同的 Project。如需了解更多 Endpoint 情况，联系您的管理员。

  > **说明:** 有关专有云的 Endpoint 填写，请联系您的专有云系统管理员咨询有关 DataHub Endpoint 的地址。

- **Project**

  填写DataHub的Project名称。

- **AccessId**

  填写具有读写该DataHub的Project权限的AccessId。

- **AccessKey**

  填写具有读写该DataHub的Project权限的AccessKey。

- 使用

  由于 DataHub 本身是流数据存储，流计算只能将其作为**流式数据输入和输出**，**无法作为维表引用**。有关 DataHub DDL 定义，请参见具体章节：

  DataHub 的源表 DDL 定义。

  DataHub 的输出 DDL 定义。

- 常见问题

  Q：为什么数据抽样仅仅针对时间抽样，不支持其他字段抽样吗？

  A：DataHub 定位是流数据存储，对外提供的接口也仅仅只有时间参数，因此流计算也只能提供基于时间的抽样。

29.6.4 日志服务

日志服务（Log Service，Log）是针对日志场景的一站式解决方案，解决海量日志数据采集/订阅、转储与查询功能。日志服务是阿里云提供的一整套非常优秀的日志管理平台，在用户直接使用日志服务进行了 ECS 日志管理情况下，流计算可以直接对接日志服务的 LogHub 存储，避免用户进行数据搬运。

- 注册
  - Endpoint

    填写日志服务的 Endpoint，不同的地域下日志服务有不同的 Project。

    说明: 有关专有云的 Endpoint 填写，请联系您的专有云系统管理员咨询有关日志服务 Endpoint 的地址。

  - Project

    填写日志服务的 Project。

- 使用场景

  由于日志服务本身是流数据存储，流计算只能将其作为**流式数据输入和输出**，**无法作为维表引用**。有关日志服务 DDL定义，请参见具体章节，包括：

  日志服务的源表 DDL 定义

  日志服务的输出 DDL 定义

- 常见问题

Q：**为什么我注册失败，失败原因提示**XXX？

A：流计算的数据存储页面仅仅提供协助您完成数据管理，其本身就是使用相关存储 SDK 代为访问各类存储。因此很多情况下可能是您注册过程出现疏忽导致，请排查如下原因：

- 请确认是否已经开通并拥有日志服务的 Project。请登录日志服务控制台，客户可以访问日志服务控制台看您是否有权限访问您的 Project。
- 请确认您填写的日志服务的 Endpoint 和 Project 完全正确。

  注意: 日志服务 Endpoint 必须以 http 开头，且不能以/结尾，例如http://cn-hangzhou.log.aliyuncs.com是正确的，但http://cn-hangzhou.log.aliyuncs.com/是错误的。
- 不要重复注册，流计算提供注册检测机制，避免您重复注册。

Q：为什么数据抽样仅仅针对时间抽样，不支持其他字段抽样吗？

A：日志服务定位是流数据存储，对外提供的接口也仅仅只有时间参数，因此流计算也只能提供基于时间的抽样。

说明: 如果希望使用日志服务的检索功能，请登录日志服务控制台使用检索。

29.6.5 分析型数据库

分析型数据库（AnalyticDB），是阿里巴巴自主研发的海量数据实时高并发在线分析（Realtime OLAP）云计算服务，使得您可以在毫秒级针对千亿级数据进行即时的多维分析透视和业务探索。分析型数据库对海量数据的自由计算和极速响应能力，能让用户在瞬息之间进行灵活的数据探索，快速发现数据价值，并可直接嵌入业务系统为终端客户提供分析服务。

- 注册

  因为当前 AnalyticDB 不支持 RAM 的服务间授权模型，因此用户只能填写 AccessId 和 AccessKey 信息作为登录信息。流计算平台会将登录信息加密存储。根据 AnalyticDB 的数据库实际情况填写如下信息：

  - URL

    填写 AnalyticDB URL 连接，格式为 my.ads-cn-hangzhou-1.aliyuncs.com:3003。不要在后续添加/符号，同时不能填写 JDBC 格式的连接串。



说明: 目前 AnalyticDB 暂不支持 VPC，且 AnalyticDB 的 URL 均为公网地址，流计算使用该地址配置即可。

- 数据库名

  填写 AnalyticDB 的数据库名称。

- 登录名称

  数据库登录名称，即阿里云访问的 AccessId 信息。



注意: 建议用户在 AnalyticDB 注册时，单独为该 AnalyticDB 创建一个 RAM 子账号，且对于该子账号权限仅限定在当前访问 AnalyticDB 的权限域内，不要将主账号的 AccessId 和 AccessKey 作为注册信息填入。

- 登录密码

  数据库登录密码，即阿里云访问的 AccessKey 信息。

- 使用

  分析型数据库不适合小批量高并发的业务场景，因此维表场景使用分析型数据库并不适合，流计算通常将其用作数据写入目的端。

  有关分析型数据库的DDL定义，请参见具体章节，包括：**分析型数据库的目标表DDL定义**。

29.6.6 表格存储

- 注册

  表格存储（Table Store）是构建在阿里云飞天分布式系统之上的 NoSQL 数据存储服务，提供海量结构化数据的存储和实时访问。流计算对于请求访问时延要求较高，同时对于关系型代数需求又较低，因此流计算非常适合使用Table Store作为数据维表和结果表。

  - **Endpoint**

    填写 Table Store 的 Endpoint，请在 Table Store 控制台查看 Table Store 的 Endpoint 信息，请注意需要填写 Table Store 内网地址。同时，由于流计算本身不支持 VPC，用户需要打开 Table Store 访问网络类型为**允许任意网络访问**。

  - **实例名称**

    填写 Table Store的Instance。

  - **AccessId**

    填写具有读写该实例权限的AccessId。

- **AccessKey**

  填写具有读写该实例权限的AccessKey。

- 使用场景

  Table Store 具备海量存储和低延迟响应的特点，适合流计算作为维表和结果表，

  具体请参见**Table Store 维表定义**和**Table Store 输出表定义**章节内容。

  。

29.6.7 云数据库

云数据库 RDS（ApsaraDB for RDS，简称 RDS）是一种稳定可靠、可弹性伸缩的在线数据库服务。基于飞天分布式系统和全 SSD 盘高性能存储，支持 MySQL、SQL Server、PostgreSQL 和 PPAS（高度兼容 Oracle)引擎。目前流计算支持的 RDS 引擎包括：MySQL、SQL Server、PostgreSQL 和 PPAS。

通常受限于关系型模型，RDS 对于大容量高并发请求的支持不如 TableStore，因此流计算和 RDS 搭配的情况更多在将 RDS 作为结果表，但对于小批量低并发的数据情况下，流计算仍然可以使用 RDS 作为维表。

- 注册

  信息填写如下：

  - 地域

    选择 RDS 的地域。

  - URL连接

    填写 RDSDB URL 连接，格式为 ip:port。

  - DBName

    填写连接的数据库名称，即 RDS 的数据库名称。

    说明: 目前 RDS 使用白名单进行安全保证，流计算需要为 RDS 自动添加白名单，请参见如何添加 *RDS* 的白名单。否则流计算可能连接不了 RDS。

  - Username

    数据库登录名称。



注意: 为了和数据库通信协议兼容，RDS在登录用户名/密码不支持阿里云的AccessId、AccessKey方式，流计算推荐用户为流计算读写RDS提供单独的用户名密码，分账号管理更容易保证RDS系统安全！

- Password

  数据库登录密码。



注意: 为了和数据库通信协议兼容，RDS在登录用户名/密码不支持阿里云的AccessId、AccessKey方式，流计算推荐用户为流计算读写RDS提供单独的用户名密码，分账号管理更容易保证RDS系统安全！

- 引擎选择

  选择引擎类型为mysql、postgresql或sqlserver。

- 使用

  受限于关系型模型，RDS 对于大容量高并发请求的支持不如 TableStore，因此流计算和 RDS 搭配的情况更多在将 RDS 作为结果表，但对于小批量低并发的数据情况下，流计算仍然可以使用 RDS 作为维表：

  RDS 维表 DDL 定义。

  RDS 输出表 DDL 定义。

- 附录

  - **流计算使用 RDS 的 VPC 模式**

    RDS 支持 VPC 网络访问，但当前流计算暂不支持 VPC 网络，因此上述 JDBC 信息中有关 RDS 的连接信息，请填写 RDS 内网地址/外网地址。RDS 内网地址/外网地址查看在 RDS 的控制台里”数据库连接”，查看”内网地址”。

    使用该地址作为 JDBC 连接串的 IP 地址，提供给流计算使用：如果您使用的 RDS 内网地址，在流计算注册中填写内网地址接口；如果您开启的 RDS 外网访问，请使用**外网地址作为流计算使用地址**。

  - **如何添加 RDS 的白名单**

    部分数据存储为了自我的安全需求，提供白名单机制，即仅允许用户指定的白名单 IP 地址中的访问链接，其他不在白名单均予以拒绝，但同样机制会拦住其他阿里云产品写入数据存储的需求。以 RDS 为例，新建的 RDS 数据库是完全拒绝任何外部连接的，此时必须要添加访问白名单，即加入访问者的 IP 才能够访问 RDS。同样，对于流计算需要将

RDS 作为维表、结果表使用时候，需要多次读写数据库，因此对于存在白名单机制的数据存储而言，必须将流计算的WebConsole以及执行节点（Worker）加入白名单才能够连接RDS。说明如下：

RDS 支持两种访问地址，包括内网地址、外网地址（VPC 地址流计算暂不支持）。针对RDS，用户只需要补充流计算白名单网段地址即可，具体的 IP 地址范围请咨询相关的管理员。

29.6.8 分布式数据库

DRDS（ Distributed Relational Database Service）是阿里巴巴自主研发致力于解决单机数据库服务瓶颈问题而推出的分布式数据库产品。 DRDS 高度兼容 MySQL 协议和语法、支持自动化水平拆分、平滑扩容、弹性扩展、透明读写分离、分布式事务、具备分布式数据库全生命周期的运维管控能力。DRDS 前身为淘宝 TDDL，是近千核心应用首选组件，已稳定服务 8 年以上。

DRDS 在流计算可以被用作外部维表关联，也可以作为结果表进行下游数据消费。

说明: DRDS 在专有云使用需要用户授予流计算代为用户访问 DRDS 权限，具体请参见流计算角色授权。否则可能出现报错**No Permission**的情况。

- 注册

  信息填写如下：

  - Region

    选择 RDS 的地域。

    Instance

    填写 DRDS 实例 ID，注意是实例 ID，非实例名称。

    说明: 目前流计算不支持 DRDS 的 VPC 网络。

  - DBName

    填写连接的数据库名称。注意，是 RDS 的数据库名称，非 Instance 的名称。

    说明: 目前 DRDS 使用白名单进行安全保证，流计算需要为 DRDS 自动添加白名单，否则流计算可能连接不了 DRDS。

  - Username

数据库登录名称。**为了和数据库通信协议兼容**，DRDS **在登录用户名/密码不支持阿里云的AccessId、AccessKey 方式**，**流计算推荐用户为流计算读写 DRDS 提供单独的用户名密码**，**分账号管理更容易保证 DRDS 系统安全**！

- Password

  数据库登录密码。**为了和数据库通信协议兼容**，DRDS **在登录用户名/密码不支持阿里云的AccessId、AccessKey 方式**，**流计算推荐用户为流计算读写 DRDS 提供单独的用户名密码**，**分账号管理更容易保证 DRDS 系统安全**！

- 网络类型

  区分内外网，优先建议用户使用内网环境。

- 使用

  DRDS 在流计算可以被用作外部维表关联，也可以作为结果表进行下游数据消费：

  DRDS 维表 DDL 定义。

  DRDS 输出表DDL 定义。

29.6.9 消息服务

阿里云消息服务（Message Service）是一种高效、可靠、安全、便捷、可弹性扩展的分布式消息服务。MNS能够帮助应用开发者在他们应用的分布式组件上自由的传递数据、通知消息，构建松耦合系统。流计算通常使用消息服务作为数据下游投递的目的端，用于告警等业务场景。

说明:

- 当前流计算对于消息服务的支持仅是作为数据输出，还不支持数据输入。
- 消息服务在专有云使用需要用户授予流计算代为用户访问消息服务权限，具体请参见《流计算角色授权》。

注册

消息服务没有所谓的 Project 级别的产品模型，所有的 Topic 和 Queue 均是在当前用户下。流计算对此作了部分抽象，将 Topic 和 Queue 组织为两类 **Project**。例如用户在流计算注册时指定**模式**为 Queue、别名为 queue_bazhen、Endpoint 填写http://1111111111.mns.cn-hangzhou-internal.aliyuncs.com，那么该用户在该Endpoint下的所有 Queue 就展示在流计算数据存储页面下名称为 queue_bazhen 的 Project 中。

图 989: 注册消息服务数据存储



- **别名**

  别名是用户在流计算里面进行管理消息服务的标识符，仅仅作为名称标识易于管理，实际上在 MNS 并没有这个概念，流计算抽象了该层。消息服务没有所谓的 Project 级别的产品模型，所有的 Topic 和 Queue 均是在当前用户下。流计算对此作了部分抽象，将 Topic 和 Queue 组织为两类”Project”，此时的别名就是流计算记录的模拟 Project 的名称。

- **Endpoint**

  消息服务的Endpoint，例如http://1111111111.mns.cn-hangzhou-internal.aliyuncs.com，特别注意的是由于消息服务 SDK 的限制，上述 Endpoint 的域名结尾不能是/符号。例如http://1111.mns.cn-hangzhou-internal.aliyuncs.com/是错误的，必须是http://111111.mns.cn-hangzhou-internal.aliyuncs.com，URL 不能以/结尾。

  > **说明:** 当前流计算不支持 VPC 模式，因此流计算当前使用消息服务的经典网络地址（Endpoint）即可。

- **AccessId**

填写具有读写该MessageService的Project权限的AccessId。

- **AccessKey**

  填写具有读写该MessageService的Project权限的AccessKey。

- **模式**

  目前消息服务支持两类消息机制，即 Queue 和 Topic，请根据业务情况选择不同的数据源。例如这里用户选择注册的是 Queue 类型，那么流计算将创建一个 Project，名称为用户定义的“别名”。所有的 Queue 均组织在这个别名下的 Project 中。

使用场景

当前流计算对于消息服务的支持仅是作为数据输出，还不支持数据输入。

有关消息服务 DDL 定义，请参见具体章节，包括：消息服务结果表 DDL 定义。

29.7 数据开发

29.7.1 开发阶段

29.7.1.1 SQL辅助

数据开发提供了一套完成的在线 SQL IDE 工具，支持如下功能辅助用户进行StreamSQL开发：

- StreamSQL 语法检查

  用户在修改 IDE 文本后即可进行自动保存，保存操作可以触发 SQL 语法检查功能。语法校验出错误后，将在 IDE 界面提示出错行数、列数以及错误原因。

- StreamSQL 智能提示

  用户在输入 StreamSQL 过程中，IDE提供包括关键字、内置函数、表/字段智能记忆等提示功能。

- StreamSQL 语法高亮

  针对 StreamSQL 关键字，提供不同颜色的语法高亮功能，以区分 StreamSQL 不同结构。

29.7.1.2 SQL版本管理

数据开发涵盖了日常开发工作的关键领域，包括代码辅助、代码版本，数据开发提供了一个代码版本管理功能。用户每次提交一个作业，发布线上，流计算均会为其生成一份代码快照，该快照可用于代码追踪修改以及版本回滚。

版本管理

点击**数据开发** > **任务属性** ，在**任务属性**下罗列了该作业的所有版本信息。

用户可以使用对比功能查看最新代码和指定版本的 Diff，可以使用回滚功能回滚到指定版本。

版本清理

流计算为每个用户设定了版本最大上限值，专有云默认是 20 个版本（其他环境请咨询流计算系统管理员）。如果生成的版本超过最大值，则系统将不允许提交，报错提示用户需要删除部分旧版本作业。

用户在**开发**右侧边栏的**任务属性** 版本列表的**详情**中，选择删除部分过期且业务不需要的版本后，即可再次进行上线作业操作。

29.7.1.3 数据存储管理

“开发”提供了一整套数据存储管理的便捷工具，用户通过在“开发”注册数据源，即可享受到多种遍历的数据存储服务，包括：

- 数据预览

  数据开发页面中，为各类数据存储类型提供数据预览功能。使用数据预览可以有效辅助用户洞察上下游数据特征，识别关键业务逻辑，快速完成业务开发工作。

- DDL 辅助生成

  流计算 DDL 生成工作大部分均属于比较机械的翻译工作，即将需要映射的数据存储 DDL 语句人工翻译为流计算的 DDL 语句。流计算提供辅助生成 DDL 工作，进一步减少用户手工编写流式任务的复杂度，有效降低人工编写 SQL 的错误率，并最终提供流计算业务产出效率。

29.7.1.4 资源（Jar）管理

资源管理是“开发”提供了用户管理 UDF的便捷工具，使用资源管理，用户可以任意上传、下载 Jar 资源，并在后续的 SQL 作业中进行引用和 UDF 处理。

上传资源

1. 点击资源管理工具栏的**上传资源**图标，弹出**上传资源**窗口。
2. 进行如下操作：
   a. 资源选择：单击**选择资源**按钮，选择待上传的资源文件。
   b. 资源名称：输入资源名称，资源名称必须唯一。
   c. 资源备注：输入资源的备注信息。

3. 单击**上传资源**按钮。

修改资源

资源上传后，支持对资源的修改操作，但资源名称不可修改。

1. 点击待修改资源操作列的**修改**，弹出**修改资源**窗口。
2. 进行如下操作：
   a. 修改注释：输入修改资源的原因。
   b. 资源选择：单击**选择资源**按钮，选择待上传的资源文件。
   c. 资源备注：输入资源的备注信息。
3. 单击**修改资源**按钮。

引用资源

支持将**开发作业**页签下的作业引用已上传的资源。资源引用后，用户便可以使用该资源的 UDF 函数。

1. 打开**开发作业**页签下待引用资源的作业。
2. 在**资源引用**页签下，点击资源操作列的**引用**。

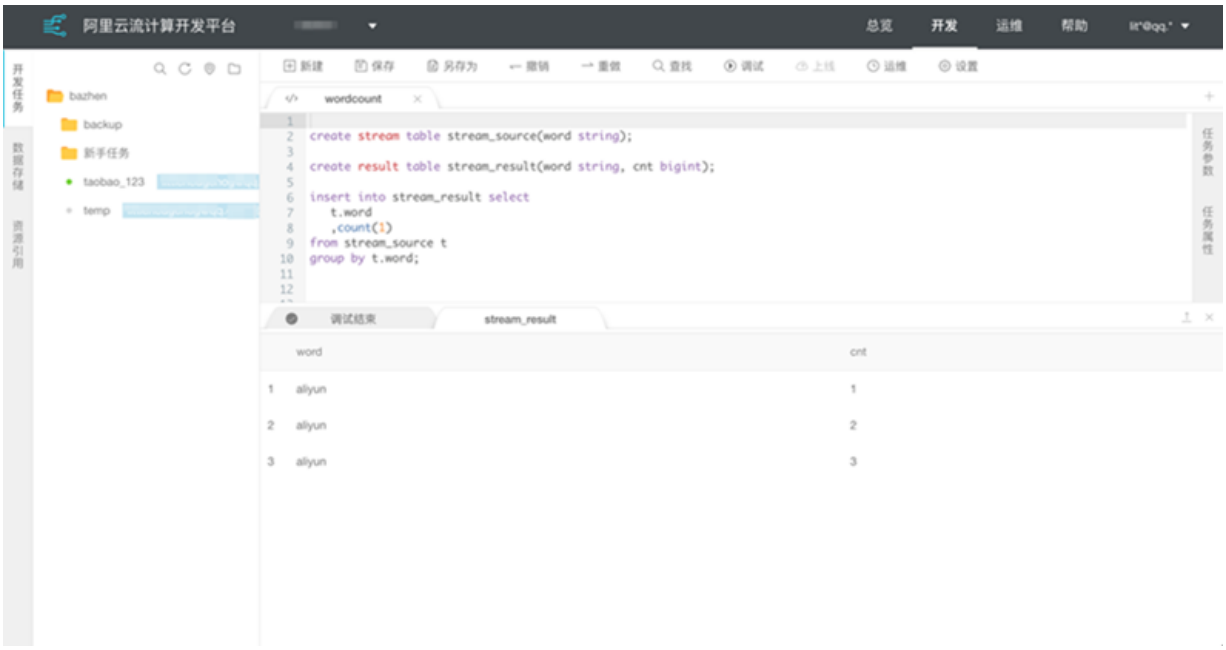
作业详情

支持查看作业的详情，包括资源版本和关联作业等内容。

1. 点击资源操作列的**详情**，弹出**资源详情**窗口。
2. 在**资源版本**页签下，可进行如下操作：
   - 回滚版本：会生成一个新的文件，文件内容为该资源版本修改前的内容。
   - 删除版本：将该资源版本删除。
   - 下载版本：下载该资源版本。
   - 删除整个资源：删除该作业下的所有版本。
   - 下载当前版本资源：下载该作业最新版本的资源文件。
3. 点击**关联作业**页签，可查看引用了该资源的作业。

29.7.2 调试阶段

开发模块为用户提供了一套模拟的运行环境，用户可以在调试环境中自定义上传数据，模拟运行，检查输出结果。

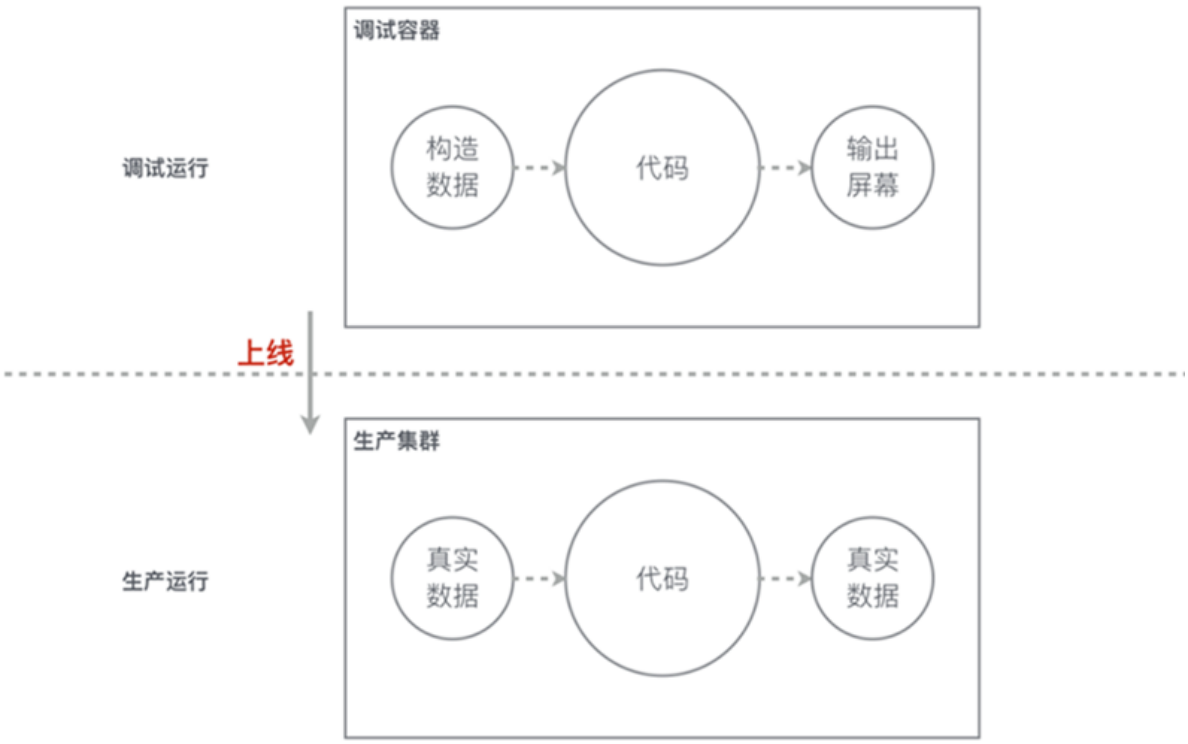


在该环境运行 StreamSQL 可以实现：

- 生产完全隔离

  调试环境下，所有的 StreamSQL 运行将在独立的调试容器运行，且**所有的输出将被直接改写到调试结果屏幕**，**不会对线上生产流计算作业、线上生产的数据存储系统造成任何影响**，让用户可以放心大胆运行任务。

  数据调试实际上不会真正写入到外部数据源，**而是被流计算拦截输出到屏幕**，**因此在流计算调试完成的代码是在调试容器中完成**，**真正线上运行过程中可能由于对目标数据源写入格式导致运行失败。**这类错误调试阶段无法完全规避，只能到线上运行才能发现。例如，用户结果数据输出到 RDS 系统，其中某些字段输出字符串数据长度大于 RDS 建表最大值，在 Debug 环境下我们无法测试出该类问题，但实际生产运行过程中会有引发异常。后续，流计算将提供针对本地调试运行也支持写出到真实数据源的功能，届时可以有效辅助用户缩短调试和生产的差距，尽可能在调试阶段解决问题。



- 支持构造测试数据

  调试环境下，所有的 StreamSQL 运行均不会从源头数据存储系统读取数据，包括 DataHub 的流式输入、RDS 等维表输入，调试作业均不会读取。调试环境要求用户必须进行自己构建测试数据集，并将测试数据上传到”数据开发”。

  您担心测试数据构造困难？没关系，流计算针对不同任务提供测试数据模板，您完全可以下载数据模板开始直接填写构造数据。**我们强烈建议用户使用下载的数据模板构造数据，以避免报错。**

调试分隔符

默认情况下，调试文件使用逗号作为分隔符，例如用户构造了如下的测试文件：

```
id,name,age
1,alicloud,13
2,stream,1
```

在不指定调试分隔符情况下，默认使用了逗号进行分割。但一旦用户需要使用JSON作为字段内容，字段内容即包含了逗号，此时用户需要指定分割符为其他字符。

注意: 流计算仅支持指定单个英文字符为分隔符，不允许字符串，例如不允许’aaa’作为分隔符。

如下：

```
id|name|content|
|1|alicloud|a,b,c|
|2|stream|{"attr": "b"}|
```

此时用户需要针对该数据源在 WITH 中指定相对应的分隔符，使用 galaxy.semantic.source.input.field.delimiter 指定分隔符即可，如下：

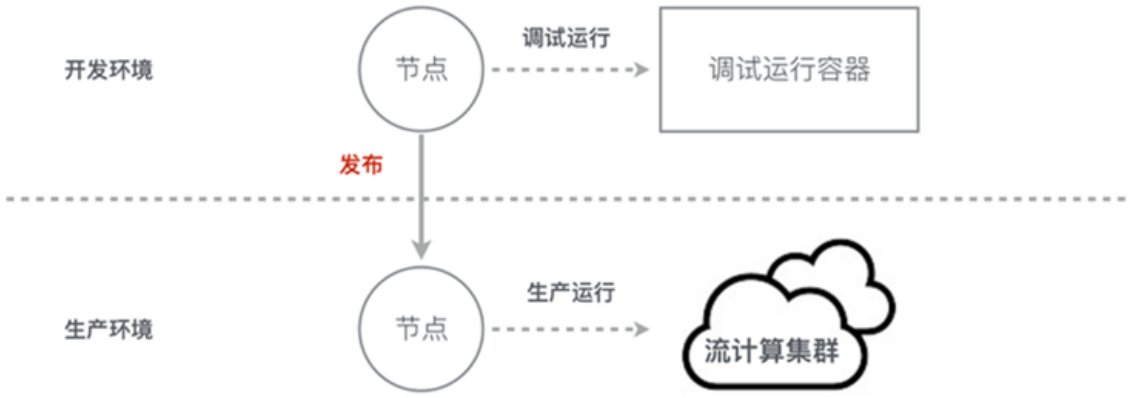
```
CREATE STREAM TABLE source_table (
    id              BIGINT,
    name            STRING,
    content                 STRING
) WITH (
    type='datahub',
    galaxy.semantic.source.input.field.delimiter='|'
);
```

此时流计算调试即可完美适应|分隔符。

29.7.3 上线阶段

当用户完成开发、调试，经验证 StreamSQL 正确无误之后，用户点击**上线**任务即可将数据发布到生产系统。任务一旦发布到生产环境，用户就需要到**运维页面**进行任务管理。

在**数据开发**中，用户所有的修改、调试操作均和**数据运维**下的真正运行的生产任务完全隔离，从而保证了开发、生产互不干扰。



29.7.4 参数变量

流计算支持任务级别参数定义，通过对流计算任务配置参数，用户可以：

- 使用系统参数调整任务运行时行为，例如用户可以指定任务运行的并发度，从而增加或者减缩吞吐量。

- 使用逻辑参数隔离环境变化，例如用户可以使用同一份代码，填充不同的环境参数，已做到不同环境之间任务切换。

必须说明的是，目前流计算仅支持任务级别的参数配置，未来还会拓展到 Project 级别的参数配置。

29.7.4.1 流计算任务静态构成

一个静态的流计算任务目前分为几大类静态信息，包括：

- 代码

  SQL 代码，用户编写的业务逻辑，流计算任务核心逻辑。

- 参数

  参数用来描述任务运行时指标，例如并发量、批处理数据量等信息。

- 属性

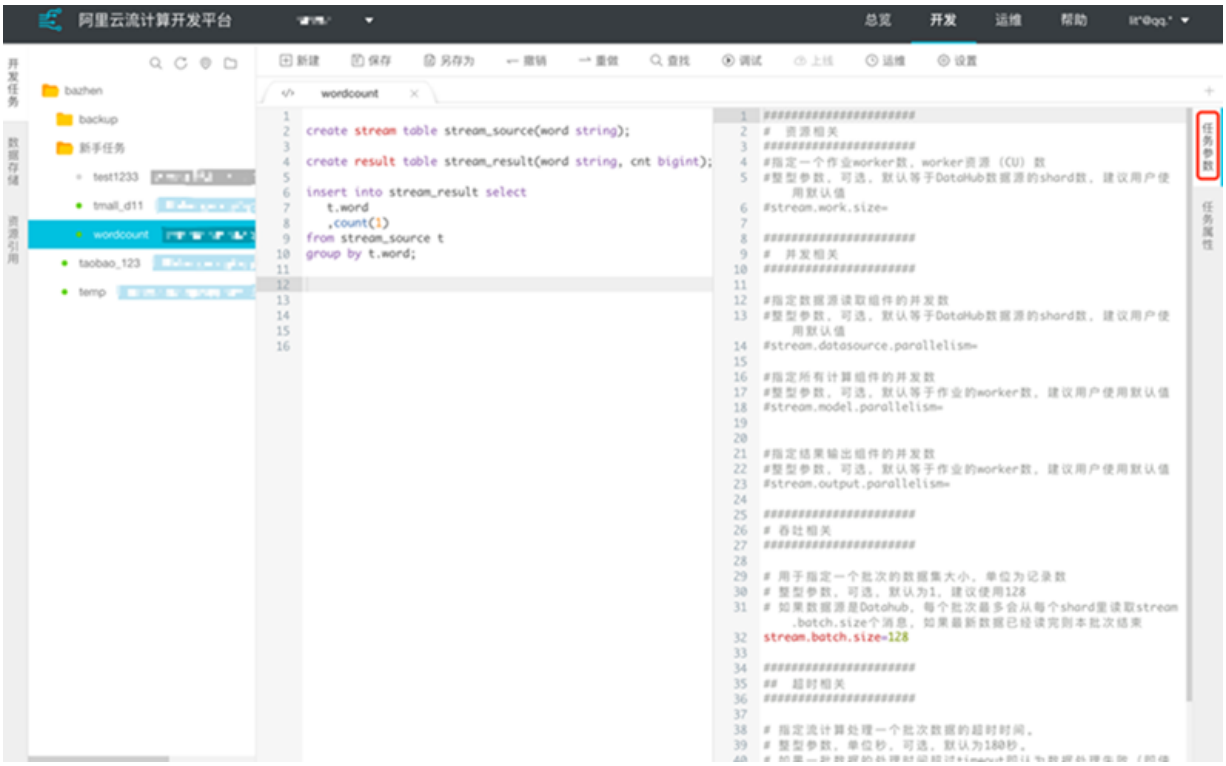
  任务的业务信息，例如创建人、创建时间等相关记录。

上述三类信息是一个流计算任务的静态信息，它们共同构成了一个完整的流计算任务静态信息。流计算任务一旦启动后，任务会附带出现各类**运行时动态信息**。这些信息非用户生成，而是我们流计算系统运行时状态反馈。

流计算任务参数分为两种类型，解决两类场景问题。我们在后面分别讲解。

29.7.4.2 系统参数

流计算预留了部分系统参数名称。这类参数是系统运行时配置，用户使用这些参数可以修改任务运行时特征，例如任务并发度等。这类系统参数均是使用**stream**或者**galaxy**全小写字符串开头。有关系统参数具体的参考手册请查阅*SQL系统参数说明*。对于系统参数，用户使用方式如下：



用户在任务参数栏填写系统参数即可。特别注意的是参数属于任务组成部分之一，因此参数的改动和 SQL 改动、任务属性改动一样，同样也会被流计算视作任务有改动，会触发流计算的自动保存功能。

说明： 系统参数改动后，如果需要对生产任务进行修改，必须进行提交并重启任务（可以是作业停止重启，也可以是暂停并按照最新配置恢复），否则修改的参数不会在流计算集群生效。

29.7.5 用户参数

29.7.5.1 用户参数说明

用户参数指用户自定义的参数信息，用户通过在 SQL 文件中设置参数名称，同时在任务参数中填写相应参数值，流计算即在提交中将根据用户填写参数进行渲染，并将最终渲染后的代码提交流计算生产集群。使用步骤如下：

1. 用户在 SQL 的 DDL with 语句中预设参数变量，并使用 ${参数变量名}进行占位。

用户在上述**任务参数**工具栏中，参看系统参数的规范，使用如下方式即可进行变量值定义，如下所示：

```
参数变量名 = 参数变量值
```

用户在 SQL 中定义参数中如果出现没有定义的参数值，流计算将报错，并提示存在没有渲染的变量值。**用户必须在提交上线任务前，将所有填写的变量名提供变量值。**

用户在提交过程中，流计算将渲染完成所有用户定义的参数，并将替换完成的最终代码文件提交到生产集群。

29.7.5.2 用户参数约束

针对用户参数灵活性，流计算对此进行一系列约束，用户必须按照如下约束处理参数：

- 用户参数使用范围仅限于 with 子句中，对于不在 with 语句中的用户自定义参数，流计算不推荐使用，且产品不定义这类行为。
- 所有的用户参数名称仅支持大小写英文字符、数字、下划线、中划线、英文点号，不允许其他字符集，不支持中文。
- 所有的用户参数名称不允许使用 stream 或者 galaxy 开头，这类参数名是系统预留参数，不允许重新定义。重新定义产品行为不确定。
- 任务的用户参数变量个数不做限制，但整体参数文件不超过1MB。
- 变量的定义不支持嵌套，例如：

```
--sql定义
    with (
        topic='${bazhen}'
    );

    --变量定义
    bazhen=${bazhen}
```

上述属于嵌套定义，产品不支持，也不做行为定义，不推荐用户使用。

29.7.6 数据运维

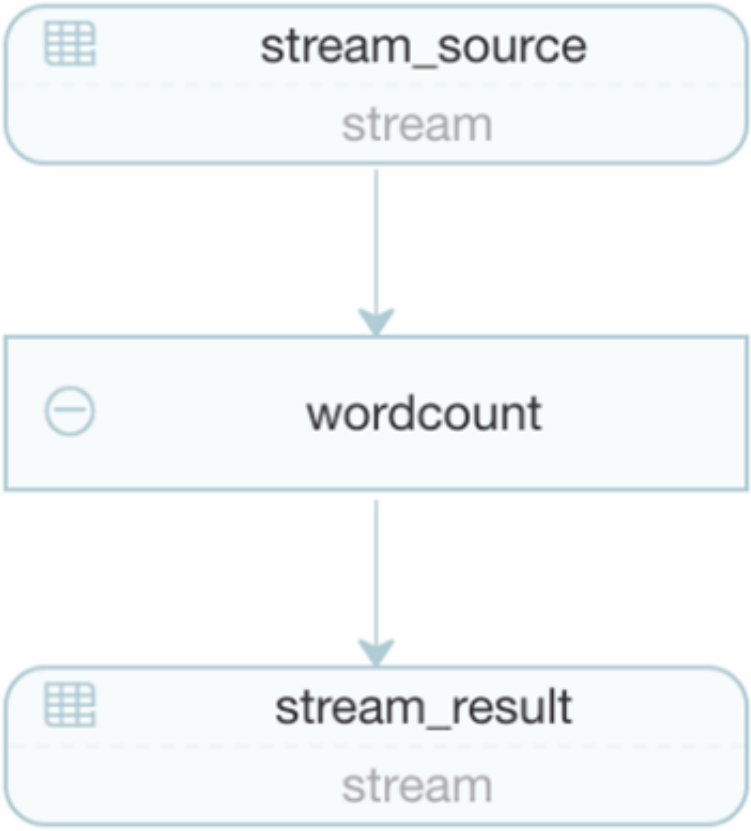
29.7.6.1 作业管理

运维面板提供了流计算作业的管理控制台，包括对任务启停操作进行了简易封装。

流计算任务状态

- 停止（未启动）

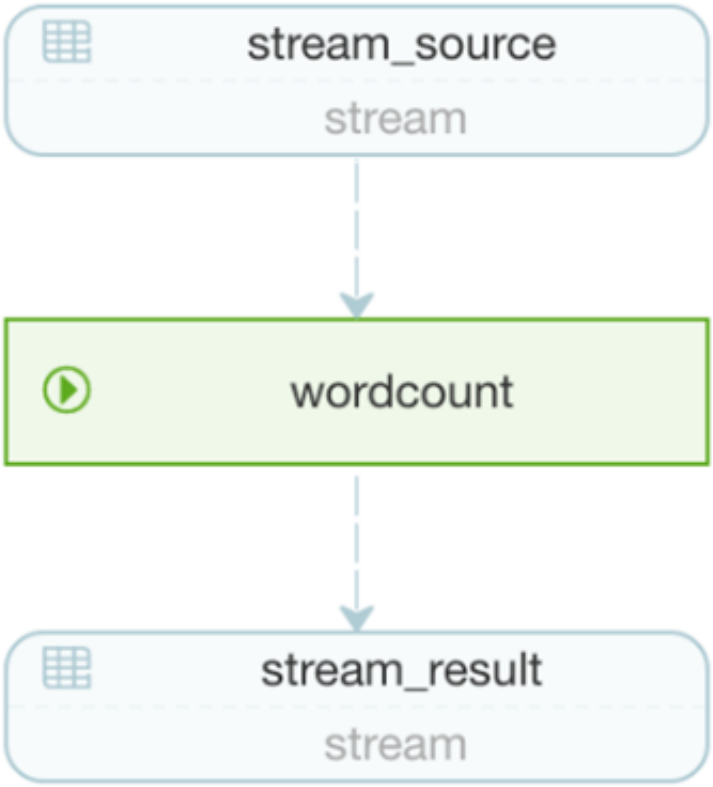
当前任务未运行，可能是已经被停止，或者新任务从来未启动过，均视作广义的“停止”状态。停止任务使用灰色表示，如下图：



- 运行

  当前任务正处于运行状态，运行任务使用绿色表示，如下图所示。

  注意: 流计算作业一旦启动，除非用户显示终止该任务，否则流计算任务一直处于运行状态。即使底层系统报错，流计算调度系统同样会尝试进行 Failover 工作。

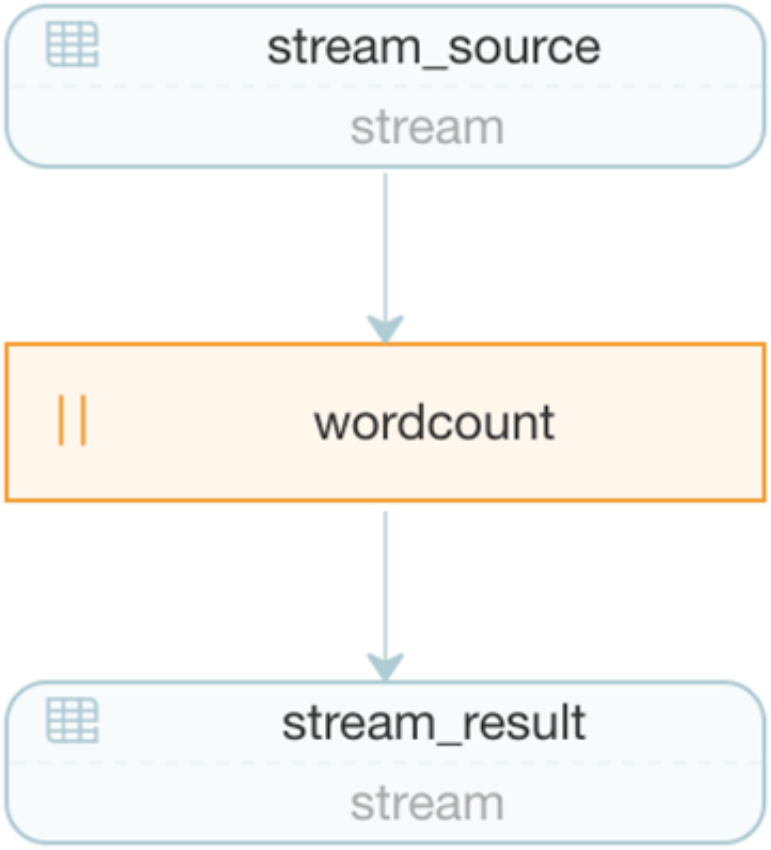


说明: 流计算是长期运行的作业，不存在所谓的**Fail**状态。即使底层出现 Fail 状况（例如连接外部 RDS 出现异常），分布式系统将尝试不停重启流计算 Fail 的节点，从任务运行状态层面，此时任务仍然是运行态。

对于如何发现底层错误，请参见错误排查，如何诊断流计算任务异常情况下对于业务的影响，请参见告警管理。

- 暂停

  当前任务被暂停，暂停意味着流计算任务仅仅是暂停运行，如果恢复后仍然可以从暂停前的状态继续计算。例如，双十一大屏任务，在读取 DataHub 时间到凌晨 2 两点整被用户暂停。随后进行恢复操作，流计算仍然从两点开始读取 DataHub 数据，且之前计算的截止到两点交易成交总额仍然可以继续使用，两点后新成交数据可以直接累加在之前的交易总和上。暂停任务使用橙色表示，如下图：



和“停止”不太一样的是，任务一旦停止，所有计算状态丢失。仍然以上述双十一大屏为例，任务停止后再启动，之前计算的成交总额全部丢失，需要重新计算。

- 下线

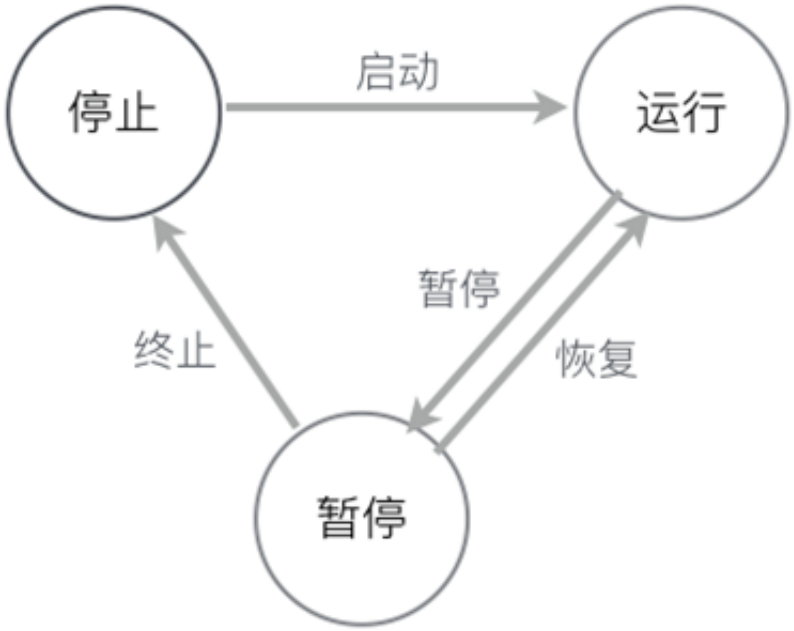
  当前任务被下线，作业下线后将在运维界面不可见。若要重新上线该作业，则需要在**开发** > **开发作业**页面，打开该作业后点击**上线**按钮。

  注意: 任务从**运行**到**停止**状态，需要先执行暂停操作，随后再执行停止操作。不支持从**运行**直接到**停止**的状态转换。

流计算任务状态图

流计算定义的任务状态转换图如图 *990: 任务状态转换图*所示。

图 990: 任务状态转换图



注意: 任务从**运行**到**停止**状态，需要先执行**暂停**操作，随后再执行**停止**操作。不支持从**运行**直接到**停止**的状态转换。

29.7.6.2 作业仪表盘

阿里云流计算提供了当前任务的核心指标概览页面，用户可以通过这个工具进行当前运行任务运行情况的一键诊断，未来流计算还会提供更多的基于任务现状深度智能分析算法，以辅助用户进行智能化、自动化诊断。目前的任务诊断如下图：

图 991: 作业仪表盘


作业仪表盘
血缘关系图
逻辑执行图
物理节点图
作业详情
日志 | 暂停 | 停止
最近1小时
最近6小时
最近1天
最近1周
2017-10-30 10:05:02
2017-10-30 11:05:02
查询
打开自动刷新
当前作业未出现错误信息
无数据倾斜
消耗 1 CU
并发 7，配置正常
业务延时
当前最大 (数据时间 - 数据产生时间) = 0.00 秒
计算耗时
当前处理一批流数据平均耗时 0.11 秒
数据输入 (RPS)
存在没有输入数据源
数据输出 (RPS)
存在没有输出的组件
CPU使用 (Core)
使用核数/申请核数 5.00 %
内存使用 (MB)
单个Worker内存使用量 255.00 MB
维表RT (ms)
每次读取维表数据的RT
暂无数据
源表RT (ms)
每次读取源头数据的RT
维表关联率
关联查询成功的数据百分比
暂无数据
维表命中率
JOIN关联命中在Cache的比率
暂无数据
源表脏数据
源表读取数据出现脏数据比例
10.30 10:05
10.30 10:13
10.30 10:21
10.30 10:29
10.30 10:37
10.30 10:45
10.30 10:53
10.30 11:01
10.30 10:25

> **说明：** 上述所有指标在流计算作业运行状态下才提供显示，**暂停以及停止状态均不提供显示。**另外，作业指标是流计算系统异步后台采集，有一定延迟。因此，**在作业启动一分钟内，指标无法显示，等待一分钟后各项指标才能逐步采集到位。**

- 异常检测

  提供该流计算任务运行的**最近的**异常信息，用户可以通过这个异常信息推断业务在底层分布式集群运行下出现的问题。例如，在流计算写入RDS过程中，出现了网络异常，此时流计算作业会抛出异常并登记在任务诊断中以方便用户事后查阅。该检测项在没有异常情况下，会显示**任务运行正常。**

  在出现异常情况下，用户可以点击**查看详情**进一步查看当前的异常栈信息。请参看错误排查一章节。

- 数据倾斜

  所谓数据倾斜，通俗而言指的是由于数据分布不均匀，个别值集中占据大部分数据量，导致计算资源不均匀引起性能下降。任务诊断中自带数据倾斜检测模块。对于可能存在数据倾斜的任务提供信息提示。

  > **说明：** 数据倾斜的判别依赖于业务上的取舍，有些数据倾斜在系统级别认为有倾斜，但是业务上可能认为合理。

  在数据倾斜情况下，流计算将展示整个作业运行链路每个阶段的不同线程处理最大值和最小值。关于如何对于数据倾斜进行调优，请参看性能调优一章节。

- 业务延时

  链路延时指**当前流计算处理时刻** 减去 **流式数据业务时间戳**（如果业务时间不存在，使用上游的DataHub/LogHub等均会对进入的数据加入系统时间戳），该数据集中反映出当前流计算全链路的一个时效情况。例如，当前流计算时间点为凌晨5点，但流计算读取处理的数据源时间戳为凌晨1点，意味着当前流计算处理的数据还是1点的数据，即处理的数据相比于当前处理时间落后近4个小时，那么业务延时就是4小时。业务延时用来监控全链路的数据进度，**如果源头采集数据由于故障没有进入**DataHub，**链路延时也会随之主键增大。**业务延时的图示如下：

  图 992: 业务延时



业务延时页面会提供当前该流计算任务的业务延时，包括当前的业务延时以及历史的延时曲线，单位是秒。

- 计算耗时

计算耗时指一批数据从进入流计算到最终输出的过程时间，该数据集中反映出当前流计算处理本身的时延，是流计算**本身处理能力**数据指标。一般而言，阿里云流计算保证计算耗时在秒级别以内。如果大于秒级，很有可能是内部处理逻辑过于复杂，此时应需要人工介入调优。计算耗时的图示如下：

图 993: 计算耗时



计算耗时页面会提供当前该流计算任务的计算计算耗时，包括当前的计算耗时以及历史的延时曲线，单位是秒。

- 数据输入

数据输入对于该流计算任务所有的流式数据输入进行统计，给用户一个直观的检测数据源 RPS（Record Per Secodng） 情况。通常，我们在系统运维过程中，如果出现没有数据输出的情况，就需要查看该值是否从源头数据输入就已经没有数据了。

数据输入一栏提供了该流计算任务数据输入是否处于正确状态的判断，如果出现任意一个流式数据源没有输入，即判断为不正常，反之为正常。



说明: 输入还针对不同流式数据源提供历史曲线。

- 数据输出

  数据输出对于该流计算任务所有的数据输（注意，并非是流式数据源，而是全部数据源）出进行统计，给用户一个直观的检测数据源 RPS（Record Per Secodng）情况。通常，我们在系统运维过程中，如果出现没有数据输出的情况，除了检查上游是否存在数据输入，同样要检查下游是否真的存在数据输出。

  数据输出一栏提供了该流计算任务数据输出是否处于正确状态的判断，如果出现任意一个数据源没有输出，即判断为不正常，反之为正常。

  说明: 输入还针对不同流式数据源提供历史曲线。

- CPU 使用

  CPU 使用反映的是流计算任务对于 CPU 资源消耗情况，流计算提供了两个指标反映 CPU 占用情况。

  当前 CPU 平均使用率： **每个流计算进程的 CPU 平均使用率**，需要注意的是平均使用率一定在[0%, 100%]波动（有时统计误差可能瞬时大于 100%）。如果当前 CPU 平均使用率持续在80%以上，可能需要考虑任务需要扩容。即调大 CU。

  CPU使用核数：**当前流计算作业的使用总 CPU Core 数**，需要注意的是 CPU 使用核数一定在[0, 申请 CU 数量]之间波动，可能是 0.3 核类似的小数。CPU 使用核数是一个曲线，展示历史的 CPU 使用核数。对于用户申请核数（CU 数量），流计算提供了一条绿色基准线，如果使用核数持续接近申请核数，需要考虑扩容，即调大CU。

- 内存使用

  内存使用率反映的是流计算任务对于内存资源消耗情况，流计算提供两个指标反映内存占用情况。

  当前内存平均使用量：**每个流计算进程的内存平均使用量**，需要注意的是平均使用率一定在[0MB, 4GB]之间波动，注意单个CU消耗的内存值在4GB以内。如果内存平均使用率大于3G，建议用户扩大该任务的流计算CU。

  内存使用量：**当前流计算作业的使用总内存量**，非平均值。内存使用量是一个曲线，展示历史的内存使用量。对于用户申请内存量（单位内存量 * CU 数量），流计算提供了一条绿色基准线，如果使用量持续接近申请量，需要考虑扩容，即调大 CU。

- 维表 RT

  维表 RT 反映的是流计算读取维表数据一次的平均 RT 时间，通常在毫秒级别。如果持续大于毫秒级，应该向相关的数据存储方咨询性能问题。例如，对于表格存储，读取一次的平均 RT 应该在毫秒级别，如果大于毫秒，请联系表格存储相关技术支持寻求建议。

  如果作业没有维表，该曲线持续为0。

- 源表 RT

  源表 RT 反映的是流计算读取源头数据一次的平均 RT 时间，通常在毫秒级别。如果持续大于毫秒级，应该向相关的流式数据存储方咨询性能问题。例如，对于日志服务，读取一次的平均 RT 应该在毫秒级别，如果大于毫秒，请联系日志服务相关技术支持寻求建议。

- 维表关联率

  维表命中率反映的是流计算在读取维表过程中，数据关联成功/总数据请求次数。如果在作业存在维表关联情况下，该值为 0 代表数据全部没有命中，进而导致用户在数据结果表就看不到数据产出的情况。

- 维表命中率

  维表命中率反映的是流计算在读取维表过程中，命中流计算内存 Cache 次数/总数据请求次数，该值越低表明数据约离散。流计算为了加快用户的查询，提供了将外部维表数据进行内存缓存，因此数据尽可能在流计算内存命中性能最优。大量的不命中导致数据需要向外部物理请求数据，可能导致性能急剧降低。

  如果作业没有维表，该曲线持续为0。

- 源表脏数据

  源表脏数据反映的是在读取源表时候，实际数据格式和用户 DDL 定义数据格式不匹配，导致数据被当做脏数据处理，进而导致用户在数据结果表就看不到数据产出的情况。

29.7.6.3 血缘关系图

29.7.6.3.1 数据拓扑

流计算任务的血缘关系图集中反映了一个流计算任务上下游数据的依赖关系，对于任务较为复杂的上下游业务依赖，数据拓扑更加能够清晰地反映出上下游依赖信息。如下图：

图 994: 数据拓扑



- 图示案例

  使用矩形表示计算作业，使用圆矩形表示上下游数据源。

  颜色图示参考任务管理栏中的颜色，灰色代表停止（或未启动），绿色代表运行，橙色代表暂停。

- 追溯按钮

  当鼠标滑动到追溯按钮，可以向上向下追溯最大三层的业务数据依赖。

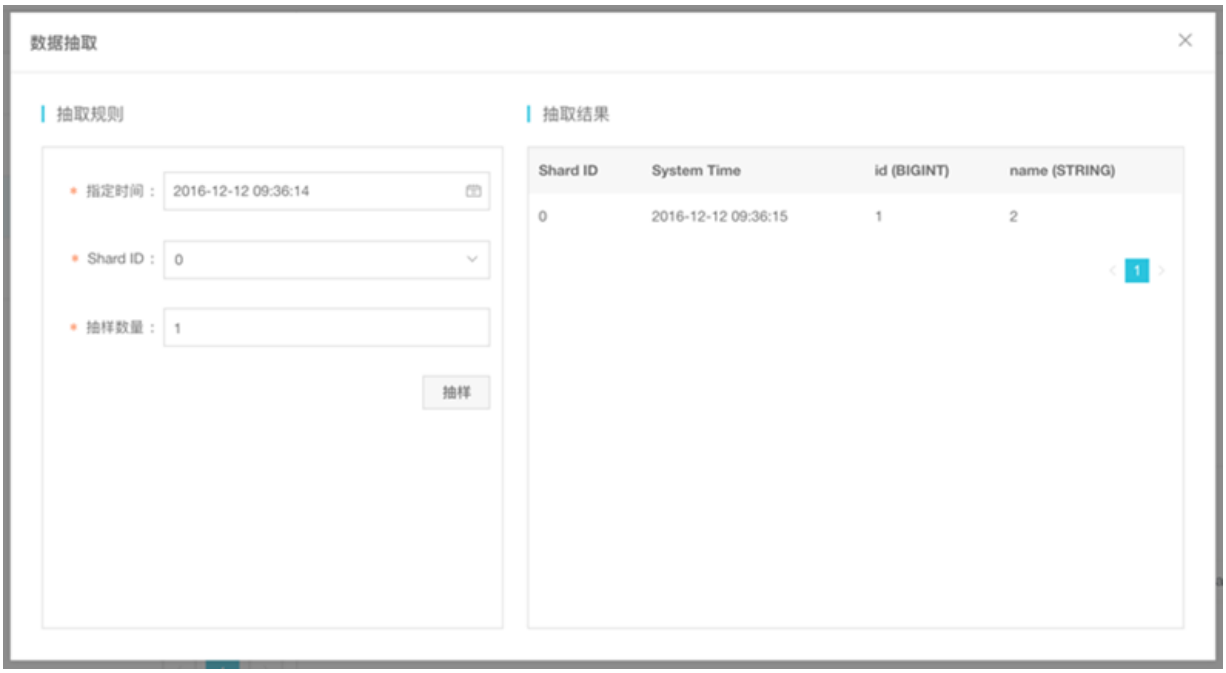
- 工具栏

  用户可以调大、调小、恢复、全屏数据拓扑图。

29.7.6.3.2 数据抽样

血缘关系图提供了数据抽样功能，该功能和数据开发页面一致，方便用户在数据运维页面进行随时数据探测，方便用户定位问题。数据抽样在血缘关系图中的拓扑上右击，在弹出的菜单中选择**数据抽样**点击即可打开数据抽样功能。

图 995: 数据抽样



29.7.6.4 逻辑执行图

不同于业务数据拓扑表现的业务上下游拓扑信息，系统拓扑反映的是底层计算节点拓扑关系图，表现经过流计算 SQL 引擎处理后的底层流式任务处理关系图。

29.7.6.4.1 组件分类

目前流计算对于底层运行节点（Task）进行分类，不同的 Task 属于不同的任务类型（负责完成同一个计算逻辑的多个并发 Task 称之为组件），目前主要分为四类：

流计算作业



- Source

  数据源节点，类似 Storm 的 Spout，主要负责向源头请求数据，并提交给下游计算处理节点。

- Map

  类似 Hadoop/MaxCompute 的 Map 节点，主要负责处理数据 Map 阶段操作，并根据数据分区（Shuffle）提交给下游 Reduce 节点。

- Reduce

类似 Hadoop/MaxCompute 的 Reduce 节点，主要负责数据 Reduce 阶段操作，从上游 Map 收集数据进行计算，并提交给下游 Merge 节点。

- Merge

  阿里云流计算 StreamSQL 为增量计算专门设计的增量计算节点，主要负责将状态进行增量计算并进行持久化操作的节点。

组件拓扑

组件拓扑关系反映的是底层不同组件上下游数据流依赖关系，类似 Hadoop/MaxCompute 的 MapReduce 模型，组件拓扑同样反映出 StreamSQL 经过引擎处理后底层运行的物理节点实际分布图。



逻辑执行图支持两种展示方式：图模式和列表模式。其中，图模式下可对节点进行右键操作。

组件图上，选中组件后右击，选择**查看节点详情**，可查看组件的详情信息，包括：

- 指标详情，提供该组件的运行指标详情，包括名称、类型等（详情参看下文的表格信息）。
- Task 流量分布，展示组件下不同 Task 流量分布情况，以**辅助用户判断数据倾斜问题**。
- 点击组件与组件之间的连线，流计算将显示**最近十分钟组件之间的流量**，这个信息非常有助于我们系统定位数据缺失问题，即数据在某个组件的处理后数据全被过滤或者 JOIN 全部失败情况。这个功能非常有用。

针对不同的组件类型，流计算提供不同的指标详情，分别为：

表 103: Source类型

名称	解释
名称	组件名称
类型	组件类型，Source

名称	解释
并发量	组件下属Task并发个数
启动时延	一批数据到来后，Source节点开始处理的时间
读取耗时	从源头读取一批数据平均耗时
处理耗时	一批源头数据解析、处理的时间
读取速率	单位为RPS，反映Source节点读取速度
读取流量	单位为Bytes/s，反映Source节点读取流速

表 104: Map 类型

名称	解释
名称	组件名称
类型	组件类型，Mapper
并发量	组件下属Task并发个数
启动时延	一批数据到来后，Mapper节点开始处理的时间
处理耗时	一批数据进行Map处理的时间

表 105: Reduce 类型

名称	解释
名称	组件名称
类型	组件类型，Reducer
并发量	组件下属Task并发个数
启动时延	一批数据到来后，Mapper节点开始处理的时间
收集时延	启动后，从上游收集齐数据，直到收集齐数据的时延
处理耗时	一批数据进行Reduce处理的时间



说明: 通常流计算会对 Reduce 阶段和 Merge 阶段进行优化，合并为一个节点操作。

表 106: Merge 类型

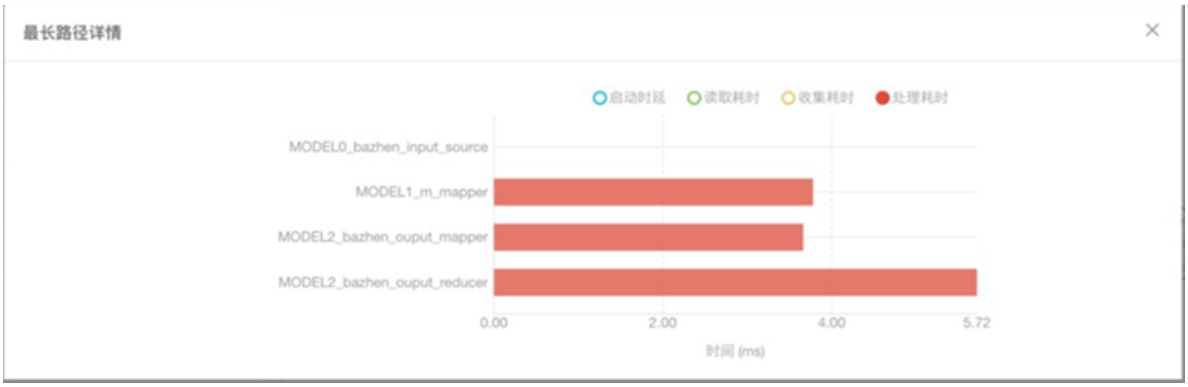
名称	解释
名称	组件名称
类型	组件类型，Merger
并发量	组件下属Task并发个数
启动时延	一批数据到来后，Mapper节点开始处理的时间
收集时延	启动后，从上游收集齐数据，直到收集齐数据的时延
处理耗时	一批数据进行Merge的时间
状态持久化耗时	一批数据进行Reduce阶段后，需要进行增量计算和状态持久化操作耗时

说明: 通常流计算会对 Reduce 阶段和 Merge 阶段进行优化，合并为一个节点操作。

29.7.6.4.2 关键路径

为方便用户定位任务运行瓶颈，流计算专门提供了关键路径分析，协助用户追踪任务时延最大的关键路径。用法如下：

1. 选择一个需要进行关键路径定位的任务，单击**逻辑执行图**，选中一个叶子节点（所谓叶子节点就是没有下游的节点，通常都是输出节点）后右击。
2. 在弹出的下拉菜单中选择**追溯关键路径**，流计算系统产出追踪出的关键路径信息，并以时间线维度展示。通过观察时间线图，您就可以查看造成该计算最大时延的关键路径。



- 关键路径追踪算法是根据用户指定的输出叶子节点，**追踪系统拓扑中每层最大耗时节点，并以时间线维度展现。**

- 时间线图从顶至下，是这批数据从最开始的 Source 节点到最末端的 Output 节点（通常是 Mapper 或者 Merger 节点）处理 Timeline 结构图。横坐标单位为 ms，表示各个节点的各个阶段耗时分布；纵坐标是组件名称。
- 横坐标起点是一批数据开始进入流计算系统后，流计算任务**各个组件开始计时**，到这个组件收集上游数据，直至最终这个组件处理完毕这批数据，该矩形框结束。因此，对于不同类型的组件存在不同处理步骤，基本上都是：
  1. 等待上游数据处理完并开始发送到该节点（启动时延），使用蓝色表示。
  2. 读取数据/收集数据的时间，使用绿色或者黄色表示。
  3. 实际计算处理的耗时，使用紫色表示。一般用户实际业务需要关心的是处理耗时，这部分执行的是用户实际业务 SQL 代码。
- 占比最大的耗时，通常是一些 Join、GroupBy 操作，定位出具体**计算慢**的原因，可以进行下一步调优工作。

29.7.6.4.3 SQL和组件对应关系

SQL 经过流计算系统编译为底层执行的组件进行物理执行，但通常流计算会对 SQL 进行深度的优化编译，导致最终的组件和业务 SQL 难以存在一对一的映射关系。用户时常需要根据组件寻找如何修改 SQL，用以解决 SQL 业务问题。此时，流计算提供了强大的映射关系图，方便您寻找组件和 SQL 对应关系。

选中任意一个组件后右击，在弹出的下拉菜单中选择**查看底层算子**，如下图：



弹出的文本框是 JSON 内容，里面记录的是当前组件下运行了哪几类 Operator，每个 Operator 从语义上可以对应出具体的SQL含义，目前有几类常见的 Operator 如下：

- ProducerOperator：源头数据读取的 Operator。
- OutputOperator：目的端数据写出的 Operator。

- MapperOperator：执行 Map 操作的 Operator。
- GroupingOperator：执行 GroupBy 操作的 Operator。
- MergerOperator：执行 Merger 操作的 Operator。
- ReducerOperator：执行 Reducer 操作的 Operator。

以一个具体的实例讲解如何查看 SQL 和组件对应关系，SQL 如下：

```
CREATE STREAM TABLE stream_sls (
    `create`    STRING
);

CREATE RESULT TABLE stream_result (
    id    STRING,
    counter BIGINT
);

REPLACE INTO stream_result
SELECT
    date_format(`create`, 'yyyy-MM-dd HH:mm:ss.S', 'yyyy-MM-dd HH:mm'),
    count(1)
FROM
    stream_sls
GROUP BY
    '1';
```

此时对应的 JSON 内容依次为：

- MODEL0_stream_sls_source

  代表从内存产生数据的源头 Operator

  ```
  {
   "RandomProducerOperator-207": [
     "stream_sls"
   ]
  }
  ```

- MODEL1_mrm_mapper

  代表执行了一系列 MAP 操作，包括 GroupingOperator 操作（以下第一个）、获取常量表达式（以下第二个）、date_format 格式化操作（以下第三个）、count 操作（以下第四个）。

  ```
  {
   "GroupingOperator-211": [
            "InAdapterProcessor{Schema[GroupingOperator-211;  in:g_127,bayes_1020_test_stre
  am_result_0,bayes_1020_test_stream_result_1;out:g_127,bayes_1020_test_stream_result_0,
  bayes_1020_test_stream_result_1}"
   ],
   "MapperOperator-208": [
      "InAdapterProcessor{Schema[MapperOperator-208; in:bayes_1020_test_stream_sls_0;out:
  bayes_1020_test_stream_sls_0,g_127}",
     "EvaluatorProcessor{ConstantValueEvaluator['1']}"
   ],
   "MapperOperator-209": [
  ```

```
                    "InAdapterProcessor{Schema[MapperOperator-209;      in:bayes_1020_test_stre
am_sls_0,g_127;out:bayes_1020_test_stream_sls_0,g_127,bayes_1020_test_stre
am_result_0}",
        "EvaluatorProcessor{GenericUDFEvaluator[date_format(bayes_1020_test_stream_sls_0,
 'yyyy-MM-dd HH:mm:ss.S', 'yyyy-MM-dd HH:mm')]}"
 ],
 "MapperOperator-210": [
            "InAdapterProcessor{Schema[MapperOperator-210;    in:g_127,bayes_1020_test_stre
am_result_0;out:g_127,bayes_1020_test_stream_result_0,bayes_1020_test_stre
am_result_1}",
   "EvaluatorProcessor{GenericUDAFMapEvaluator[count(1)]}"
 ]
}
```

- MODEL1_mrm_reducer_merger

  代表一系列 reducer 和 merge 操作，最为重要的是第二个 Count 操作。

```
{
 "MergerOperator-213": [
    "InAdapterProcessor{Schema[MapperOperator-208; in:bayes_1020_test_stream_sls_0;out:
bayes_1020_test_stream_sls_0,g_127}",
   "EvaluatorProcessor{ConstantValueEvaluator['1']}"
 ],
 "ReducerOperator-212": [
    "InAdapterProcessor{Schema[ReducerOperator-212; in:bayes_1020_test_stream_result_0,
bayes_1020_test_stream_result_1;out:bayes_1020_test_stream_result_0,
bayes_1020_test_stream_result_1}",
                "EvaluatorProcessor{GenericUDAFReduceEvaluator[count(bayes_1020_test_stre
am_result_1)]}"
 ]
}
```

- MODEL2_stream_result_mapper

  代表输出的 Map 节点，注意该节点实际上并没有实际上输出，为下游实际输出的 Reduce 节点发送数据。

```
{
 "OutputOperator": [
   "stream_result"
 ]
}
```

- MODEL2_stream_result_reducer

  代表输出的 Reduce 节点，实际负责对外写数据，即将流计算数据写出到外部存储。

```
{
 "OutputOperator": [
   "stream_result"
 ]
}
```

29.7.6.5 作业详情

任务详情提供了当前作业的详情信息，包括当前运行信息以及历史运行记录。



29.7.6.5.1 当前运行代码

当前运行代码包括当前运行作业的所有信息，用户可以通过该页面查询运行代码层面的细节。

- 运行代码

  记录当前运行的 SQL 代码。

- 运行属性

  记录当前作业的属性，包括任务 ID、任务标签、启停时间等等。

- 运行参数

  记录当前运行的参数信息。

29.7.6.5.2 历史运行记录

记录该作业的历史运行记录，即作业历史上多次运行的周期。

29.7.6.6 作业告警

和批处理任务差别在于，流计算任务启动后除非显示被用户终止，该任务一直不会停止。因此，对于流计算任务的监控**不能使用简单的任务正确结束和异常结束来进行区分**。通常，我们对于流计算任务监控使用**延迟监控**的手段进行预警。

29.7.6.6.1 延迟监控的原理

当流计算任务出现错误或者故障，例如用户提供的 RDS 用户名/密码存在错误，流计算作业无法进行写入操作。流计算系统将尝试进行修复、重试操作，尽管对用户而言，这类修复/重试操作是完全透明的。但从整体计算结果而言，由于数据无法写入会导致流计算任务一直”卡住”在写出部分，进而导致源头新的数据流计算无法消费。从任务全链路上看，该流计算任务呈现一种”卡住”或者”变慢”的状态，表现出来的情况是新的数据源源不断有进入，但是流计算仍然在消费的几分钟、甚至几个小时前的源头数据，一直没有读取新数据。我们用公式：

业务延时=当前真实时间-当前读取数据源业务时间

来表示任务整体延迟情况。关于业务延迟，也可以参看《任务仪表盘》中有关业务延迟一章节内容。

> **说明:** 现有流计算监控收集的是**一分钟平均业务延迟**，**有可能在某个瞬间大于设定阈值**，**但一分钟平均下来可能小于预警值**，**不会引发异常。**

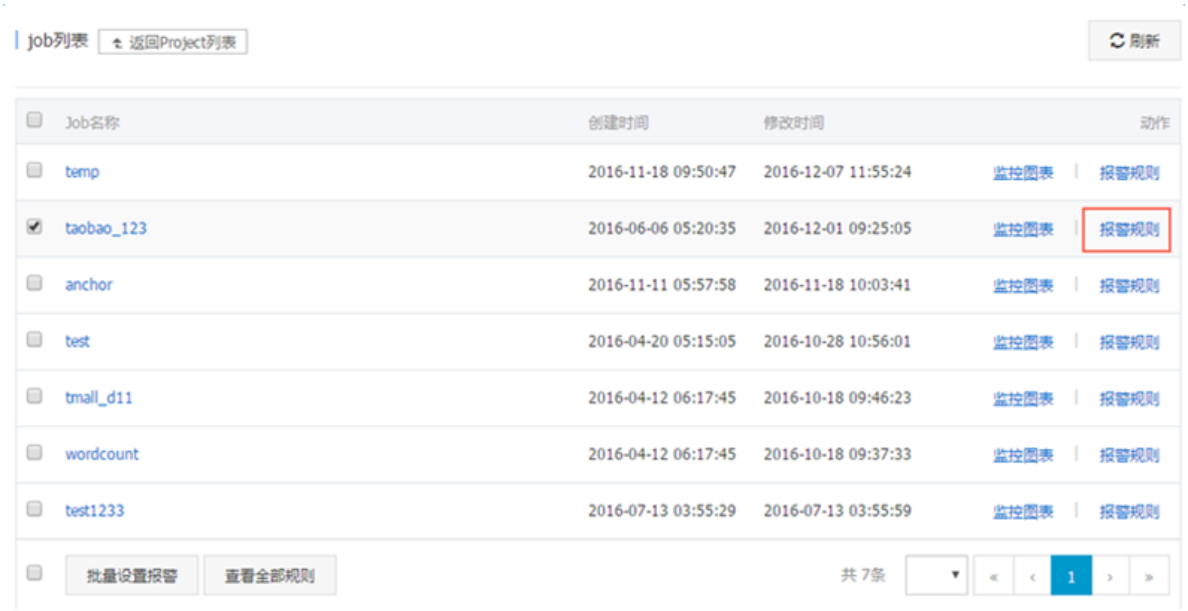
29.7.6.6.2 云监控告警

用户要使用云监控，必须先开通云监控相关项目，请参看《云监控产品》开通和使用流程。

流计算对接云监控原理如下：



1. 在云监控页面，单击**流计算**，进入流计算项目列表。
2. 单击**查看Job**，进入作业页面，点击**报警规则 - 创建报警规则。**



3. 按照云监控引导，配置告警规则。



之后，云监控可以根据用户告警规则派发告警。此外，云监控同样提供了告警指标的曲线，您可以查看过去触发告警的业务延迟变化情况。



29.7.6.7 进阶说明

29.7.6.7.1 异常排查

在开发流计算逻辑过程中，阿里云流计算通过一系列的优化措施（例如提供 StreamSQL、提供调试环境等）努力降低用户使用门槛，同时极力帮助用户避免错误。但由于流计算领域本身固有的复杂度，仍然不可避免用户会有错误发生。

阿里云流计算推荐的开发流程是：

1. 在**数据开发**中编写 SQL，数据开发提供语法校验工具保证 SQL 语法正确。
2. 在**数据开发**中打开**数据调试**，保证逻辑正确
3. 发布到**数据运维**，使用**任务诊断**排查生产问题。

阿里云流计算建议用户遵循上述开发流程，可尽量避免流式任务的各类错误。

在**数据运维**页面，搜索需要运维的作业，单击查看作业详情。任务列表栏中，任务状态为红色反映出当前任务运行有错误，点击**任务诊断**，观察任务诊断中**运行异常**一栏是否有错误提示，如果标红表示有错误提示。

单击**详细信息**，在弹出的对话框中可以看到具体的报错信息，您可以根据该报错信息修正自己的作业错误。



这个作业运行错误的原因是 Table Store（原OTS）写入失败，由于数据格式错误（String 类型无法写入 Integer）导致格式错误。

29.7.6.7.1.1 常见作业错误

- 数据源鉴权失败

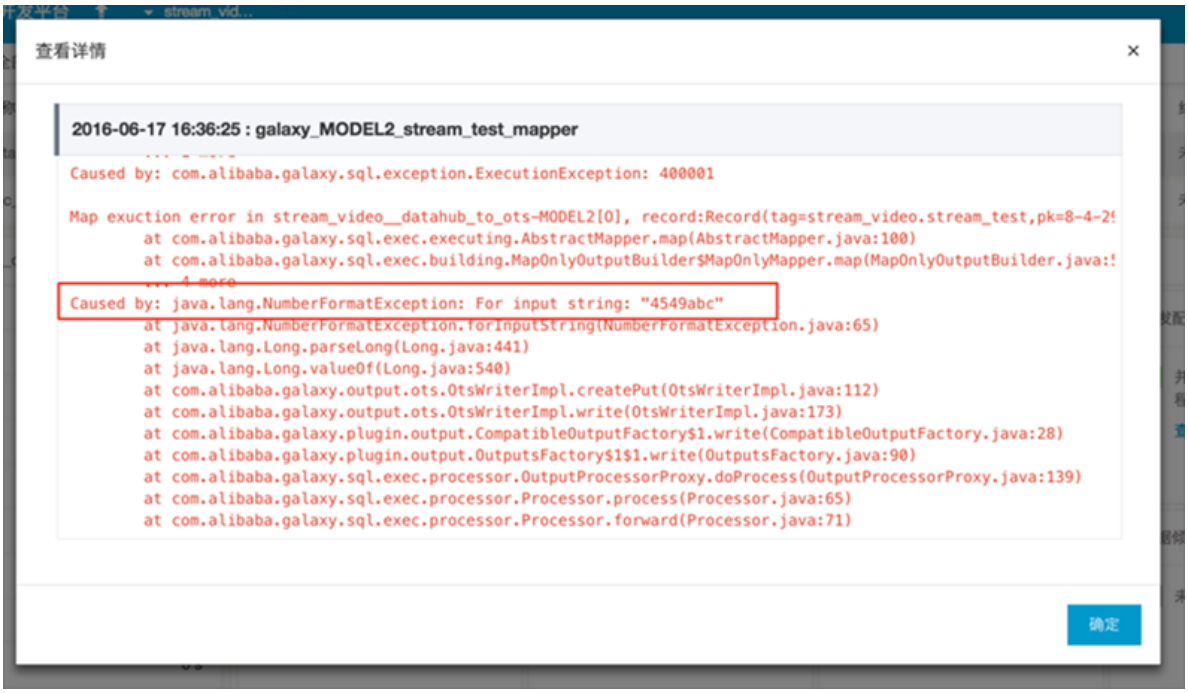
  由于填写的数据源连接信息有误导致，例如 RDS 用户名/密码有误。此时需要您仔细检查涉及到报错数据源连接信息。



  解法是修改 SQL 文件中有关数据源连接的信息，并重新提交上线。

- 数据格式不匹配

源头数据的格式为 String，但是写入目标端数据存储要求为整数类型，两边数据格式不匹配必然导致数据写入失败。如上述的报错信息：



解法是调整两端数据类型为一致，或者在 StreamSQL 中进行类型转换。

- 业务脏数据

  由于部分业务存在脏数据情况，目前流计算对于脏数据暂不支持丢弃处理，您需要在 StreamSQL 中过滤这类脏数据。例如目标端 RDS 字段类型为 varchar(16)，但是当前写入字段为100字符长度，必然写入失败。您可以调整 RDS 字段类型长度，或者使用 StreamSQL 过滤掉长度大于16字符的字段。

  后期流计算会上线针对脏数据容错策略，可以根据用户的配置策略处理脏数据情况。

29.7.6.7.2 性能调优

29.7.6.7.2.1 调优指标

相比于离线作业而言，除了作业功能运行正常，流计算作业更加强调整体计算的吞吐以及时延。尽管阿里云流计算对于底层系统已经做了大量优化，但人工调优的过程在部分性能苛刻的情况下仍然需要。

学习性能调优前，须了解目前流计算提供有哪些指标可以反映出底层流计算性能能力，否则我们无法清楚优化的方向以及优化的效果评估。目前对于性能指标，流计算提供包括：

- **业务延时**

流计算是一直持续处理数据的系统，不存在一个数据处理结束的时间点，无法直接计算耗时来度量流计算的当前业务处理进度。通常，对于流式处理系统，我们使用流计算当前**处理的数据时间点来表征流计算当前业务处理进度**，例如当前时间点为18:15，但流计算处理的是17:15的数据，这之间的时间差记为业务延时。

业务延时使用的算法是 当前流计算系统时间 - 上游流式存储数据业务时间（上游的 DataHub/LogHub 等均会对进入的数据加入系统时间戳），该数据集中反映出当前流计算全链路的一个时效情况。例如，当前时间点为凌晨5点，但流计算读取的数据还是凌晨1点的数据，意味着当前流计算处理的数据还是1点的数据，给出的结果数据应比现在时间落后近4个小时。此时业务延时为4小时。

业务延时集中反映出当前流计算业务处理的性能情况，理论上业务延迟肯定是越小越好。实际上取决于业务对于流计算的时延要求（例如天猫双十一业务整体业务延时要求不超过5秒），但通常在秒级、分钟级。一旦出现过大的业务延时，通常情况有：

- 流计算性能无法跟上外部数据写入 DataHub/LogHub 等上游数据总线的性能，这样会造成上游数据生产能力>流计算消费，造成数据一直处于”堆积”状态。可能是上游数据流量很大导致流计算处理来不及，或者流计算**计算耗时**过大导致整体流计算吞吐不够，但更经常的情况是写入目的端出现脏数据（参见问题排查）或者写入性能并发过低导致。
- 流计算读取的历史数据。例如流计算启动就读取的是几个小时前的 DataHub 数据，那么当前的业务延时必定在小时级别。对于这类情况，随着后面流计算作业消费数据，会逐步追赶上历史数据的。

  业务延时



- 业务延时反映出当前流计算处理的业务进度，这个是业务核心中的核心指标。但业务延迟没有一个统一的标准，例如天猫大屏整体链路的业务延迟不超过 3 秒，但一般的报表产出业务延迟不超过半个小时，具体取决于业务对于链路延迟情况。我们通常对于报表建议不超过小时级别都理解为合理的。

- **计算耗时**

  计算耗时指一批数据从进入流计算到最终输出的过程时间，该数据集中反映出当前流计算处理本身的时延，是流计算**本身处理能力**数据指标。一般而言，阿里云流计算保证计算耗时在秒级。如果大于秒级，延迟到分钟级别，很有可能是内部处理逻辑过于复杂，此时应需要人工介入调优。计算耗时就是流计算处理一批数据的耗时，类似网络请求的 RT 时间，仅反映一个请求处理时间，不能反映吞吐量。有可能上游数据进入量过大导致流计算处理不够及时导致数据延迟。

  计算耗时



  计算耗时反映流计算当前处理的速度。

- **输入 RPS**

  流计算上游流式数据源（包括 DataHub、LogService 等）数据每秒记录读取数，该指标直接反映出上游数据消费能力。如果流计算读取 RPS 小于数据进入上游流式存储的 RPS，必然导致数据在上游越来越堆积。输入 RPS 反映流计算当前吞吐量。

- **输出 RPS**

  流计算将将结果写出外部数据源的每秒记录数，该指标直接反映出业务最终交付性能。如果流计算每秒写出过大，甚至大于下游存储例如 RDS 的 TPS 能力，肯定导致流计算处理能力变慢；如果流计算每秒写出比业务预期少，我们也需要关注是否业务代码出现问题、上游数据输入是否有问题，导致整体上数据输出变少了。

总结来说，时间相关的指标反映的是流计算处理能力的及时性（强调速度），输入输出的吞吐量集中反映流计算处理流式数据能力的大小（强调吞吐）。整体上，流计算优化亦是参考上述两类指标进行。

流计算吞吐能力不够，例如处理速度达不到上游数据进入 DataHub 的速度，立刻会导致数据在 DataHub 堆积，并进而导致业务延时加大。因此，实际上流计算最终都可以通过业务延时来反映整体作业性能问题。

29.7.6.7.2.2 调优参数

在了解到阿里云流计算的四个性能指标后，我们就可以学习下流计算提供的几个调优参数：

- CU 数量
- Batch 大小
- Model 并发
- 数据倾斜处理
- 优化开关
- 其他优化

下面分别讲述上述参数意义和调节方法：

- CU 数量

  用户在任务参数配置里面，使用 stream.worker.size 配置 CU 数量。

  CU 数量反映的流计算处理能力，在公共云上流计算 1 个 CU 平均处理能力在 1000 条/S（具体处理能力需要根据业务场景，简单的数据处理会较大超出1000 条/S，而复杂的计算例如多维度聚合查询导致性能会下降），用户可以根据业务情况适当调整 CU 数量。

- Batch 大小

  您在任务参数配置里面，使用 stream.batch.size 配置 Batch 大小。

  一个批次的实际数据量可以这样计算：batch.size 通道数，一个消息的记录数，一般控制在 10w 以内，不宜过大，太大容易导致处理超时或 OOM。例如对于 DataHub： 100个通道的 Topic，100(topic) 100(batch.size)1（一个DataHub消息就是一个流计算的记录） = 1w。而对于 LogHub 而言，batch 大小主要影响的是数据产出的实时性及数据处理的吞吐能力。batch 小则相对而言批处理耗时会更低，但是吞吐必然受影响；反之 batch 设置的大，批处理耗时会有所增加，同时所需要内存也会增加，但是处理的吞吐理论上会随之增长。当然如果由于增加 batch 大小之后导致内存都超了，可能会因为 GC 频繁，即资源受限导致处理耗时过大，吞吐反而会有所下降，一般活动期间为了扛过峰值流量，都会更强调处理的吞吐，当然具体问题需要具体处理：

  - 吞吐不够，业务延迟增大，但是计算处理耗时比较低（一般在几百毫秒 ~ 10秒之内），这种情况下的处理方式一般是直接增加 galaxy.batch.size 大小，从而提升源头处理的 qps，具体增加大小可以根据当前配置下所能达到的 qps1（当前的批大小为 b1），根据相应比例得出待调批大小b2=b1*qps2/qps1。

- 吞吐不够，消费延迟增大，但实际批处理耗时还很高，这个时候就需要考虑先优化作业本身的性能，从资源、model 并发、倾斜处理等维度进行调优了，这些将在后续进行介绍。

  说明: Batch 调节的其他方式

  通过以上的方法进行 batch 大小配置调优，基本上可以满足活动的要求，但由于单个批次真实可以解析出的消息数可能超出预估，这种极端情况下可能会出现调优后的内存不足导致 OOM 的问题，如果业务上允许超出预估流量峰值后有可控的延迟，但任务必须正常运行，有以下两种方式进行控制，可以结合一起同时使用或单独使用：

  - galaxy.fetch.list.size：该参数主要是用来控制每个通道的预取线程可以从各消息中间件（tt/metaq等）读取的最大 block 数，默认不设置的情况下是 galaxy.batch.size 的10倍，如果所订阅日志/ topic 的数据量很大，可以酌情缩减该参数大小，此时一般建议其大小在1.5 ~ 3倍。
  - galaxy.semantic.source.finish.type=fixedcount 及 galaxy.semantic.source.finish.value=m：这两个参数用以控制每个 queue 每批读取的最大处理条数，默认情况下不设置，则一批是以 galaxy.batch.size=b2 达到条件而结束，而加上上述配置之后，则是以 b=min{m条消息所对应的实际 block 数 b1, b2} 结束一批数据的处理。

  通过以上两组配置，则可以保证在流量超出预期之后，业务仍然可以正常运行，同时可以满足或在可接受范围内满足业务的吞吐需求。

- Model并发

  并发是针对 Model 级的参数，实际上一个并发对应一个线程，太大浪费资源，太小会导致作业的吞吐瓶颈，设置并发的关键在于找到作业的瓶颈，通过调高存在瓶颈的 Model 并发来提高作业的整体吞吐。可以通过一些参数指定并发：

```
stream.datasource.parallelism=1     ##指定datasource并发数
stream.model.parallelism=1          ##指定model并发数
stream.output.parallelism=1         ##指定output并发数
${MODEL}.parallelism                ##指定某个Model的并发
${MODEL}.[mapper|reducer|merger].parallelism   ##指定Model的指定阶段的并发
```

  为了不浪费资源、同时不会使 worker 负载过高，建议所有model并发总数/worker 数之比最好在5 ~ 25倍之内，如果不满足则可以相对调整并发大小或 worker 数。

  - 源头并发

源头并发的大小建议设置为所订阅上游数据总线的 Shard/分区数，例如 app_taobao_ctrl_sys 通道数为256，则该源头的并发则可以设置为256，stream.datasource.parallelism=256，如果待调优作业中存在多个源头，如还有一个源头订阅了图6的日志，则建议通过 ${MODEL}.parallelism 来设置，即可分别设置 MODEL0.Parallelism=256，MODEL1.Parallelism=128。

- 写出并发

  考虑最后输出的数据量大小，如果最终结果的聚合度高（通过聚合运算后数据量很小），可以设置小点，如果聚合度不高，需要设置得大一些，通过并发提高吞吐。所有 output 并发大小可通过 stream.output.parallelism 统一设置，或通过 ${MODEL}.parallelism 单独设置每个 output 并发，在调优过程中，建议使用后者，因为不同的输出所对应的外部存储类型、集群吞吐等都可能不一样。

- 中间并发

  中间并发指非源头输入，目的端写出的节点并发数量，目前有如下调优经验指导：

  作业吞吐量比较大的，可以考虑并发数设置大写。但请注意所有 model 的并发总数/worker 数在5 ~ 25之内较佳。

  有 JOIN 维表，且使用 ALL 方式加载的，并发不要太大，在加载维表数据时容易导致 Table Store 压力过大。

  Merger 阶段相对消耗的资源较多，并发可以考虑大一些，特别是有使用布隆过滤器或基数估计计算 UV 的地方。

  并发数一般设置为 work 数的倍数，每个 worker 的使用率会平均一些。

  参考物理执行图的每个物理执行节点处理耗时指标或者是追踪关键路径中发现最耗时的节点，该节点就是系统运行瓶颈，找到对应的Model使用参数${MODEL}.parallelism适当调大并发数量。

  Max-Queue-Size 过大的 model，需要加大并发以提升处理能力。

- 数据倾斜处理

  数据倾斜情况一般发生在聚合计算逻辑时，如下图业务代码：

```
Insert into my_result
select
  t.thedate, cardinality(0.99, t.user_id) as uv
from my_source t
group by t.thedate;
```

统计天级别的访问 uv，因为实时业务正常运行时，只有一个 key（thedate）值，这样所有的聚合运算都会根据 key shuffle 到一个节点上进行计算，不可避免的会出现瓶颈。处理倾斜的情况，一般建议先进行分桶打散进行一次聚合计算，然后再在此基础上进行最后的汇总，处理后的业务逻辑代码如下：

```
Insert into t_uv
select
  t.thedate, cardinality(0.99, t.user_id) as uv
from my_source t
group by t.thedate, hash(t.user_id) % 17;

insert into my_result
select
  t.thedate, sum(t.uv)
from t_uv t
group by t.thedate;
```

上述业务代码中的 17 为分桶数，该值建议设置为质数，具体大小视具体业务逻辑和对应的倾斜度而定，在数据量很大的情况下，虽然增加了拓扑的深度，但是其在倾斜程度较大的时候，性能增益依然会相当明显。

- Model合并优化

  很容易理解的是，一个业务的性能与其计算复杂度有关系，计算复杂度对性能的影响可以简单从作业拓扑/ DAG 的深度和广度来评估。

  深度上，一般深度多一级，性能下降 20%-30%。

  广度上，多一个分支取决于最慢的那个分支，不过由于消息需要多复制一份，一般有10%-20%的损耗。

  Model 合并则是为了从深度上对业务的性能进行优化，其可以通过配置项 galaxy.exec.plan.optimize.enable=true 来打开，默认为 false 关闭，其可以将源头节点与下游 mapper 节点进行合并，一般下游的 mapper 会对应一些过滤或数据解析等业务逻辑，减少一层的数据 shuffle，尤其在源头消息量特别的情况下，可以极大的提升作业性能。

  但该配置默认设置为false，是因为存在一些限制：

  - 某些场景不支持：例如一个源头数据被下游多个节点直接订阅，则该开关即便打开也不会生效。
  - 副作用：下游 mapper 是维表 join 或其他一些特殊情况下，打开性能反而可能会有所下降。

- 业务调优

  在一些情况下，还可以通过一些业务层面的调整达到性能上的优化，其中包括：

- 过滤逻辑前置：减少往下游分发的数据量，例如尽可能在前面将数据过滤，使用 Where 条件处理掉多余的数据，避免到下游再过滤。
- 分拆业务：如果分支过多（广度上的性能下降），则可以考虑分拆业务，将作业分成2个或多个作业，减少因广度上的性能消耗，同时更少的业务逻辑的作业可以极大减少调优难度。
- 增加消息中间件或外部存储的吞吐能力：对于上游数据存储而言，例如 DataHub，如果流量很大，可以考虑通过扩容 DataHub 通道，例如从 10 个 Shard 扩展到 100 个 Shard，增加源头的并发，从而提升性能。

29.7.6.7.2.3 调优思路

在对流计算任务性能调优之前，一定需要清楚是当前性能问题主要集中在哪个指标上，找到问题根节点，不要为调优而调优。

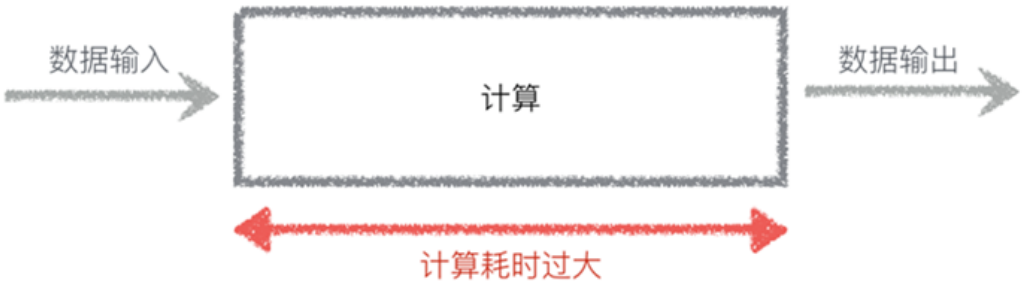
以下四个指标，在**数据运维 > 作业仪表盘**中可以查阅到：

图 996: 作业仪表盘



- **计算耗时过大**

  图 997: 计算耗时过大



通常而言，流计算计算耗时在秒级到分钟级别之间，大于分钟级别的计算耗时意味着一批数据经过流计算需要处理的时间长达分钟。计算耗时可以集中在”数据运维” - “任务诊断”，其中**计算耗时**即可反映当前采集的计算耗时瞬时值。如果该值在分钟级别，或者大于业务预期，可以尝试进行调优。

相比于其他性能指标，计算耗时集中反映的是流计算内部处理逻辑耗时，因而问题的排查可以更加集中：

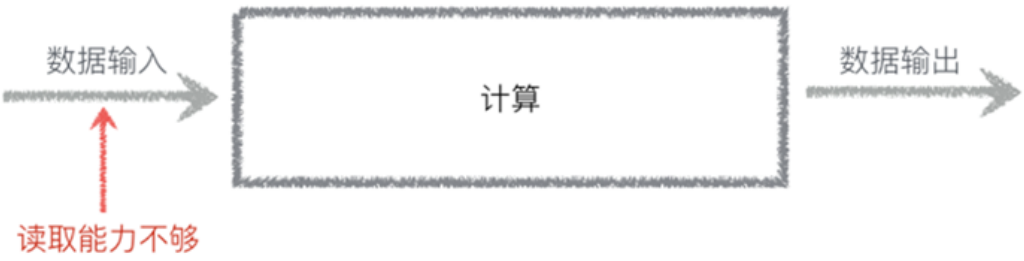
1. 点击**数据运维- 逻辑执行图**，找到叶子节点，并点击**追溯关键路径**，找到最长的执行路径。
2. 从最长执行路径中找到，最大处理耗时的节点（关键路径时间线图使用红色标注），通常这些耗时严重的节点执行占到整体路径执行耗时的30%以上。
3. 尝试进行调优工作：
   - 这个节点可能由于：
     - 叶子节点在对外写出数据，这说明是 Output 慢，解决办法如下：
       - 参看前一章节的 Output 并发调节，优化并发。
       - 优化外部维表存储性能，例如购买 Table Store 的更高规格。
     - Join 节点在执行对外维表关联操作，这说明是维表关联比较慢，解决方法如下：
       - 参看前一章节的中间并发调节，提升维表关联并发度。
       - 优化维表指标，提升维表 Cache 内存，参看《维表参数》。
     - 计算逻辑复杂（例如 UDF 复杂）：
       - 参看前一章节的中间并发调节，优化并发。

- 针对这些情况需要调大这个节点的并发（线程并发数量），使用 ${MODEL}.parallelism 可以针对这个 Model 调整并发数量，请参见 StreamSQL 中的参数说明。

- 当前处理逻辑存在数据倾斜，导致一个节点下并发 task 处理能力不均匀，进而导致节点计算耗时加大。针对数据倾斜情况，用户可以通过修改 SQL 进行解决。参看前一章节有关数据倾斜的优化。

- **输入RPS低**

图 998: 输入 RPS 低



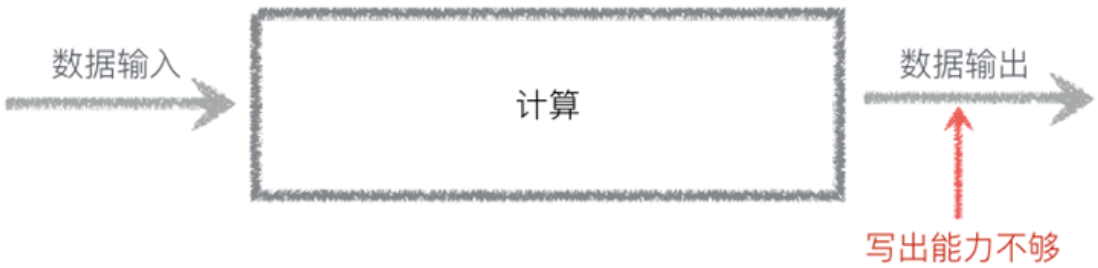
输入 RPS 低指流计算本身处理数据吞吐性能就不足，导致每秒读取数据量达不到业务预期。例如业务估算每秒 ECS 采集的日志量是 1 万条/s，但流计算 RPS 在 7 千条/s，此时数据一定在源头大量堆积，并导致业务延时迅速增大（**业务延时同样作为一个非常重要的参考指标**！）。

1. 观察业务延时是否增大，如果没有增大说明**数据进来的就比较少**，应该到数据采集端看看是否数据采集出现问题。常见数据采集问题有
   - 采集任务 Fail 导致部分数据少采集。
   - 业务数据量下降，例如非用户活跃期（凌晨）日志量就是比较少。
2. 如果业务延时逐步增大，说明流计算吞吐能力已经达不到源端数据进入流量，此时应该进行调优：
   - 观察计算耗时，如果比较大说明流计算任务计算处理能力存在问题，请参见上述**计算耗时过大**一章节。
   - 如果计算耗时正常，此时说明流计算本身处理的吞吐能力不够，应该调大 CU。目前一个 CU 平均处理能力在 1000 条/s左右（具体处理能力视代码复杂度），用户可以根据这个规格进行逐步调优。

说明: CU 的参数为 stream.work.size，请参见《参数说明》一章节。

- **输出 RPS 低**

  图 999: 输出 RPS 低

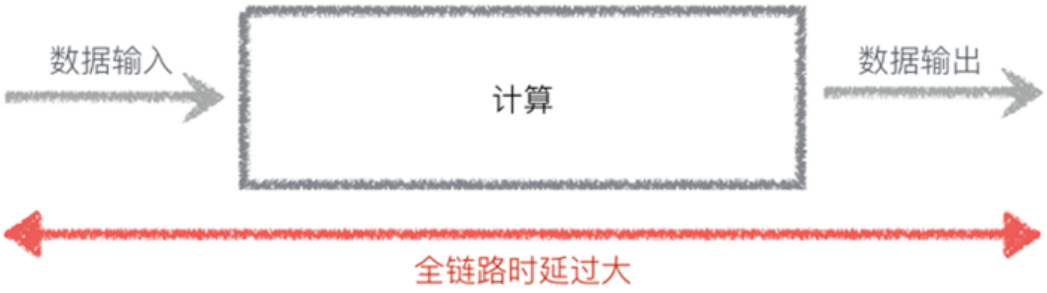


  输出 RPS 低指流计算对外产出数据吞吐性能不足，导致结果数据流量达不到业务预期。

  1. 观察输入 RPS，是否输入 RPS 就比较低呢？如果输入 RPS 就比较低，那么请参考上述**输入 RPS 低**问题排查。
  2. 输出数据本身就不多，是否经过 SQL 过滤、关联查询等一系列处理后数据本身就不多，请排查 SQL 逻辑是否存在问题。
  3. 流计算处理能力不足，导致输出 RPS 就会比较少，此时参考**计算耗时过大**一章进行优化。
  4. 输出的数据存储 TPS 不够，例如输出到 Table Store（原 OTS），但 Table Store 的 TPS 不够，此时**计算耗时**同样会增大。需要尝试调大输出叶子节点的并发度，或者调大目标端数据存储的 TPS 能力，例如调大 Table Store 的规格，调整 RDS 的 CPU/MEM 规格等。

- **业务延时大**

  图 1000: 业务延时大



业务延时反映的是整体业务业务延时，告诉用户是**当前流计算处理的数据业务时间具体是哪一个时刻，和当前时刻相比落后多长时间。**上述三个指标**计算耗时、输入 RPS、输出 RPS**不正常均会导致**业务延时**的增大。因此，如果出现**业务延时**过大，首先排查上述三个指标是否存在异常情况，如果上述指标均没有问题，那可以基本排除流计算本身问题，考虑是外围因素导致数据延迟：

1. 当前任务数据采集工作出现故障？请查看任务仪表盘中，数据输入是否正常，是否新数据没有进入，流计算处理的时间一直停留在数据终止上传的时间点。这类情况请排查上游数据存储（DataHub等），查看是否有数据进入。
2. 当前任务是否就在处理历史数据？例如当前在处理昨天的数据，流计算汇报业务延时会以处理数据的业务时间为业务延时。

29.7.7 StreamSQL

StreamSQL 是阿里云流计算为简化流计算模型，降低用户使用流计算门槛而设计的一套符合标准 SQL 语义的流计算 SQL。使用 StreamSQL，用户可以享受到如下好处：

- 开发成本急剧降低。使用阿里云 StreamSQL，用户可以不再编写长达数千行的 Storm 代码，通过一两句简单的 SQL 即可描述复杂且多变的业务逻辑。让人人均可以使用流 计算，让人人均可以享受到流计算。
- 维护成本急剧降低。使用阿里云 StreamSQL，用户可以不用再维护数千行的 Storm 代码，使用人人皆易懂的 SQL 可以让维护成本急剧降低。

使用最原始的 Storm 编程模式，用户必须考虑各类系统异常情况，包括系统异常、系统宕机、错误情况下数据准确性等各类细节问题。

阿里云流计算团队将上述各类问题均在 SQL 语义中进行抽象和解决，使用阿里云流计算，您可以：

- 不用操心数据准确性，阿里云流计算独特的 Exactly-Once Processing 语义保证，能够保证各种 Failover 场景的数据准确性。
- 不要担心性能优化，阿里云流计算集成 SQL 优化逻辑，将 SQL 翻译为最底层的执行代码的普通人员写的 Storm 代码效率更高，完全不用操心 SQL 代码的效率。
- 不用考虑编程可扩展性，阿里云流计算支持 UDF 的扩展功能，对于 SQL无法表达的、或者业务定制化逻辑，阿里云流计算能够通过自定义 UDF 进行功能扩展，完全不用考虑 SQL 编程的可扩展性。
- 不用关心系统的扩展，阿里云流计算在保证任务和系统稳定性同时，在数据突增时（比如双11）可以做到弹性扩容。

29.7.7.1 关键字

阿里云流计算常用的关键字如下：

表 107: 关键字

类型	关键字
数据类型	INT, BIGINT, DOUBLE, STRING
TABLE 类型	STREAM, RESULT, TMP, DIM
DDL	CREATE TABLE
DML	INSERT INTO、REPALCE INTO
SELECT 子句	SELECT FROM、WHERE、GROUP BY、DISTRIBUTE BY、 UNION ALL、JOIN/ LEFT OUTER JOIN、TOP ORDER BY ASC/DESC PARTITION BY

29.7.7.2 数据类型

在 StreamSQL 中，支持 INT/BIGINT、DOUBLE、STRING 四种类型。

- INT 整型。在 SQL 文本中定义的数字常量默认解析为 INT 类型，取值范围对应 Java 的 Integer 类型，例如：`select 1 from T t where t.a > 10;`
- BIGINT 整型。取值范围对应 Java 的 Long 类型，强制转换使用如下函数：`select cast(1 as bigint), cast('100' as bigint), from T t;`

- DOUBLE 浮点类型。统一使用 DOUBLE 类型，带小数点的字符常量解析为 DOUBLE 类型：select 1.0 from T t where t.a > 10.1;

强制转换使用如下函数：select cast(1 as double), cast('100' as double), from T t;

- STRING 字符串类型。字符串常量使用单引号(')或者双引号(")括起来。

29.7.7.3 运算符

29.7.7.3.1 关系操作符

操作符	说明
A=B	如果A或B为NULL，返回NULL；如果A等于B，返回TRUE，否则返回FALSE。
A<>B	如果A或B为NULL，返回NULL；如果A不等于B，返回TRUE，否则返回FALSE。
A<B	如果A或B为NULL，返回NULL；如果A小于B，返回TRUE，否则返回FALSE。
A<=B	如果A或B为NULL，返回NULL；如果A小于等于B，返回TRUE，否则返回FALSE。
A>B	如果A或B为NULL，返回NULL；如果A大于B，返回TRUE，否则返回FALSE。
A>=B	如果A或B为NULL，返回NULL；如果A大于等于B，返回TRUE，否则返回FALSE。
A IS NULL	如果A为NULL，返回TRUE，否则返回FALSE。
A IS NOT NULL	如果A不为NULL，返回TRUE，否则返回FALSE。
A LIKE B	如果A或B为NULL，返回NULL，A为字符串，B为要匹配的模式，如果匹配，返回TRUE，否则返回FALSE。'%'匹配任意多个字符，'_'匹配单 个字符。要匹配'%'或‘ _’ 需要用转义符表示‘ \%’ ，‘ _’ 。 ‘aaa’ like ‘a’ = TRUE ‘aaa’ like ‘a%’ = TRUE ‘aaa’ like ‘aab’ = FALSE ‘a%b’ like ‘a\%b’ = TRUE ‘axb’ like ‘a\%b’ = FALSE
A RLIKE B	A是字符串，B是字符串常量正则表达式；如果匹配成功，返回TRUE，否则返回FALSE；如果B为空串会报错退出；如果A或B为NULL，返回 NULL；
A IN B	B是一个集合，如果A为NULL，返回NULL，如A在B中则返回TRUE，否则返回FALSE 若B仅有一个元素NULL，即A IN (NULL)，则返回NULL。 若B含有NULL元素，将NULL视为B集合中其他元素的类型。 B必须是常数并且至少有一项，所有类型要一致。

注意:

由于double值存在一定的精度差，因此，不建议使用等号（=）直接对两个 double 类型数据进行比较。用户可以先将两个 double 类型相减，而后取绝对值的方法比较两个 double 类型的数据。当绝对值足够小时，认为两个 double 数值相等，例如：

```
abs(0.9999999999 - 1.0000000000) < 0.000000001
-- 0.9999999999和1.0000000000为10位精度，#0.000000001为9位精度。
-- 此时可以认为0.9999999999和1.0000000000相等。
```

29.7.7.3.2 算术操作符

操作符	说明
A + B	如果A或B为NULL，返回NULL；否则返回A + B的结果。
A – B	如果A或B为NULL，返回NULL；否则返回A – B的结果。
A * B	如果A或B为NULL，返回NULL；否则返回A * B的结果。
A / B	如果A或B为NULL，返回NULL；否则返回A / B的结果。注：如果A和B为 bigint类型，返回结果为double类型。
A % B	如果A或B为NULL，返回NULL；否则返回A模B的结果。
A DIV B	如果A或B为NULL，返回NULL；否则返回A整除B的结果。注：如果A和B为INT/BIGINT类型，返回结果为BIGINT
+A	仍然返回A。
-A	如果A为NULL，返回NULL，否则返回-A。

注意:

- 只有 STRING，bigint，double 才能参与算术运算，日期型不允许参与运算。
- String 类型在参与运算前会进行隐式类型转换到 double 类型。
- bigint 和 double 共同参与计算时，会将 bigint 隐式转换为 double 再进行计算，返回结果为 double 类型。
- A 和 B 都是 bigint 类型，进行 A/B 运算，返回结果为 double 类型；进行上述其他运算仍然返回 bigint 类型。

29.7.7.3.3 位操作符

表 108: 位操作符

操作符	说明
A & B	回A与B进行按位与的结果。例如：1&2返回0，1&3返回1，NULL与任何值按位与都为NULL。 A和B必须为Bigint类型。
A | B	返回A与B进行按位或的结果。例如：1 |2返回3，1 |3返回3，NULL与任何值按位或都为NULL。 A和B 必须为Bigint类型。



说明: 位运算符不支持隐式转换，只允许 bigint 类型。

29.7.7.3.4 逻辑操作符

表 109: 逻辑操作符

操作符	说明
A and B	TRUE and TRUE => TRUE TRUE and FALSE => FALSE FALSE and TRUE => FALSE FALSE and NULL => FALSE NULL and FALSE => FALSE TRUE and NULL => NULL NULL and TRUE => NULL NULL and NULL => NULL
A or B	TRUE or TRUE => TRUE TRUE or FALSE => TRUE FALSE or TRUE => TRUE FALSE or NULL => NULL NULL or FALSE => NULL TRUE or NULL => TRUE NULL or TRUE => TRUE NULL or NULL => NULL
NOT A	如果A是NULL，返回NULL 如果A是TRUE，返回FALSE 如果A是FALSE，返回TRUE。

逻辑操作符只允许 Boolean 类型参与运算，不支持隐式类型转换。当前流计算只有在函数或者表达式返回 Boolean 类型结果。

29.7.7.4 DDL语法

阿里云流计算对于表 DDL 仅支持创建操作，格式如下：

```
CREATE (STREAM|TMP|RESULT|DIM) TABLE table_name
[(col_name data_type, ...[, PRIMARY KEY(col_name, ...)])] [WITH (property_name=
other_property_value,...)];

data_type
: primitive_type

primitive_type
: INT
```

```
| BIGINT
| DOUBLE
| STRING
```

说明：

- 阿里云流计算本身不带有数据存储功能，因此所有的涉及表创建 DDL 操作实际上均是对于外部数据表/存储的引用声明，例如：

  ```
  create stream table stream_source ( name STRING,
  value STRING
  ) WITH (
  type='datahub',
  endpoint='http://dh-cn-hangzhou.aliyuncs.com', accessId='accessId',
  accessKey='accessKey', projectName='project', topic='topic'
  );
  ```

  这里是在流计算 SQL 中并非创建一个 DataHub 的 Topic，仅仅声明了一个名称为 stream_source 的表引用，下游所有的对这张 DataHub 的 Topic 相关 DML 操作均可以使用 stream_source 别名来进行操作。
- 流计算对于声明表的作用域是在当前任务（一个 SQL 文件提交后生成一个流计算任务），即上述有关 **stream_source** 的声明仅在当前 SQL有效。在同一个 Project 下的其他 SQL 文件同样可以声明名称为 stream_source 的表。
- DDL 语法中关键字、表名、列名等不区分大小写。
- 表名、列名、WITH 子句中的属性参数值都只能使用 ascii 字符，不能使用 unicode。表名、列名必须以字母或者数字开头，表名、列名中只能包含字母、数字、下划线和原点符号。
- 在 DDL 声明表语法中，声明字段名和目标表的字段名需要名称完全一致。DDL 声明中，根据名称进行映射。
- PRIMARY KEY 用于定义流数据的主键。在流计算中，有些流数据的处理计算要求流数据在业务上拥有主键属性，这些操作包括：双流 JOIN/LEFT OUTER JOIN和 RESULT TABLE 的数据输出等。
- WITH 定义了数据源存储位置，参看不同的存储类型下的 WITH 描述。

29.7.7.4.1 创建流式源表

流计算的源表是指流式数据源，流式数据源驱动流计算的运行。因此，每个流计算子任务必须提供至少一个流式数据源！目前流计算仅支持 DataHub 做流式数据输入，声明一个 DataHub 流式源表的语法如下：

```
CREATE STREAM TABLE table_name
[(col_name data_type, ...[, PRIMARY KEY(col_name, ...)])] [WITH (property_name=
other_property_value,...)];
```

```
data_type
: primitive_type

primitive_type
: INT
| BIGINT
| DOUBLE
| STRING
```

对于 WITH 配置说明如下：

表 110: WITH 配置说明

字段名称	说明
type	数据源类型，当前仅支持DataHub，必须填写**datahub**（全小写）。
accessId	能够读写该Topic的accessId。
accessKey	能够读写该Topic的accessKey。
endpoint	DataHub的Endpoint。
projectName	DataHub的Project名称。
topic	DataHub的Topic名称。

一个简单声明 DataHub 引用的语法如下：

```
create stream table stream_source ( name STRING,
value STRING
) WITH (
type='datahub',
endpoint='http://dh-cn-hangzhou.aliyuncs.com', accessId='accessId',
accessKey='accessKey', projectName='project', topic='topic'
);
```

29.7.7.4.2 创建维表

流计算使用 DIM TABLE 定义维表数据。维表数据是一类特殊的外部数据，它相对流数据来说比较稳定且变化缓慢，是静态或准静态数据，作为 JOIN/LEFT OUTER JOIN 的右表使用。

```
CREATE DIM TABLE table_name
[(col_name data_type, ...[, PRIMARY KEY(col_name, ...)])] [WITH (property_name=
other_property_value,...)];

data_type
: primitive_type

primitive_type
: INT
| BIGINT
| DOUBLE
| STRING
```

29.7.7.4.2.1 维表通用配置

在实际生产业务中，维表数据可能存在大量的关联查询操作，因此流计算对于维表设置了部分通用配置，以方便用户针对维表进行细节调优操作。

Cache 类

为优化查询性能，避免每次流数据 Join 操作均从实际维表数据源查询数据，所有的维表操作在流计算内部实际上均支持使用内存 Cache 进行缓存，配置如下：

- cacheType cache 的类型，目前支持三种类型，包括：
  - LRU，使用 LRU 算法进行过期数据替换，流计算默认使用该选项。
  - ALL，所有维表数据将直接加载到流计算每个计算节点内。注意，如果数据太大，可能造成 OOM 情况。
  - NONE，关闭 cache，每次 Join 操作流计算将直接查询维表数据源。

  默认值为 LRU 类型。
- cacheTTL

  指定 cache 的 Time-To-Live 时间，即数据过期时间，单位为秒。一旦数据在 cache 存在时间大于 cacheTTL，数据将从维表数据源重新加载。

  默认值为600，单位为秒。
- cacheSize

  指定 cache 的大小，单位为记录数。注意，该值设置过大，或者每条记录过大，可能造成计算进程 OOM。

  默认值为10000，单位为记录数。

29.7.7.4.2.2 TableStore（源TABLESTORE）维表

流计算支持使用 TABLESTORE 作为维表，示例如下：

```
CREATE DIM TABLE user ( id STRING,
name STRING,
age STRING, PRIMARY KEY (id)
) WITH (
type='ots',
endpoint='http://ots-cn-hangzhou.aliyuncs.com',
accessId='accessId',
accessKey='accessKey',
instance='instance',
tableName='table'
);
```

其中 WITH 配置说明如下：

表 111: TableStore 维表

字段名称	说明
type	数据源类型，TABLESTORE类型必须填写**TableStore**
accessId	能够读写该TABLESTORE的accessId
accessKey	能够读写该TABLESTORE的accessKey
endpoint	TABLESTORE的Endpoint
instance	TABLESTORE的Instance名称
tableName	TABLESTORE的Table名称

说明:

- 对于 TABLESTORE 维表，流计算要求声明维表时必须指定 TABLESTORE 的 PrimaryKey，同时该 PrimaryKey 必须是 TABLESTORE 的 PrimaryKey。
- 使用 TABLESTORE 维表进行关联查询时，用户必须使用上述 TABLESTORE 的 PrimaryKey 进行 JOIN 查询，即在 JOIN ON 后续的关联查询的字段必须是 TABLESTORE 的 PrimaryKey。
- 当前阶段流计算在创建 TABLESTORE 维表仅支持 STRING 类型，即维表情况下，TABLESTORE 其他类型均需转换为流计算的 STRING 类型。

29.7.7.4.3 创建结果表

流计算使用 RESULT TABLE 作为输出结果数据的格式定义，同时定义数据如何写出到目的数据源。

流计算将结果表简单分为两类：Append 类型和 Update 类型。

- Append 类型。使用 INSERT INTO 语法即可支持 Append 类型写入，包括 DataHub、未定义主键的 RDS 等均为 Append 类型的数据存储，即流计算的结果将直接追加到目的数据存储，而不会修改目的存储中现有的数据。

  注意: 使用 INSERT INTO 语法，RESULT 表不能定义 PRIMARY KEY，否则流计算提交运行阶段将报错。
- Update 类型。使用 REPALCE INTO 语法即可支持 Update 类型写入，包括定义主键的 RDS、TABLESTORE 等均为 Replace 类型的数据存储，即流计算的结果将 Insert（如果相同 Key 数据不存在）或者 Update （如果相同 Key 数据已存在）。

注意: 使用 REPLACE INTO 语法，RESULT 表必须定义 PRIMARY KEY，否则流计算提交运行阶段将报错。

当前流计算各类存储对于写入支持如下：

类型	追加写入	更新写入
DataHub	支持	不支持
TABLESTORE	不支持	支持
RDS(Mysql)	支持	支持
Analytic DB	不支持	支持

结果表代码如下：

```
CREATE RESULT TABLE table_name
[(col_name data_type, ...[, PRIMARY KEY(col_name, ...)])] [WITH (property_name=
other_property_value,...)];

data_type
: primitive_type

primitive_type
: INT
| BIGINT
| DOUBLE
| STRING
```

29.7.7.4.3.1 DataHub结果表

流计算用持使用 DataHub 作为结果输出，数据将以 append 的方式追加到 DataHub 中。示例代码如下：

```
create result table stream_result ( name STRING,
value STRING
) WITH (
type='datahub',
endpoint='http://dh-cn-hangzhou.aliyuncs.com', accessId='accessId',
accessKey='accessKey', projectName='project', topic='topic'
);
```

对于 WITH 配置说明如下：

字段名称	说明
type	数据源类型，DataHub类型请填写'datahub' (全小写)
accessId	能够读写该Topic的accessId
accessKey	能够读写该Topic的accessKey

字段名称	说明
endpoint	DataHub的Endpoint
projectName	DataHub的Project名称
topic	DataHub的Topic名称



说明: 对于 DataHub 存储仅支持追加写方式，因此对于 DataHub只能用 INSERT INTO 方式写入。

29.7.7.4.3.2 RDS（Mysql）结果表

流计算支持使用 RDS 作为结果输出。示例代码如下：

```
CREATE RESULT TABLE rds_result ( name STRING,
value STRING
) WITH (
type='mysql', url='jdbc:mysql://localhost:3306/javademo?
useUnicode=true&characterEncoding=UTF8', username='username',
password='password', tableName='table'
);
```

其中 WITH 配置说明如下：

字段名称	说明
type	数据源类型，这里填写 mysql。
username	能够读写该 RDS 的用户名。
password	能够读写该 RDS 的密码。
url	该数据库 jdbc 连接串信息，采用 jdbc 格式填写。
tableName	该 RDS 的数据表名称。

注意

- 如果声明 RDS 结果表没有定义 PrimaryKey，那么输出数据将以 Append 方式追加到现有 RDS 表，即搭配使用 INSERT INTO 进行写入。
- 如果声明 RDS 结果表定义有PrimaryKey，那么输出数据将以Update#式追加到现有 RDS 表，即搭配使用 REPLACE INTO 进行写入。

29.7.7.4.3.3 TABLESTORE结果表

流计算支持使用 TABLESTORE 作为结果输出。示例代码如下：

```
CREATE RESULT TABLE user (
id bigint,
```

```
name string,
age bigint PRIMARY KEY (id)
) WITH (
type='ots',
endpoint='http://ots-cn-hangzhou.aliyuncs.com', accessId='accessId',
accessKey='accessKey', instance='instance', tableName='table'
);
```

其中 WITH 配置说明如下：

字段名称	说明
type	数据源类型
accessId	能够读写该TABLESTORE的accessId
accessKey	能够读写该TABLESTORE的accessKey
endpoint	TABLESTORE的Endpoint
instance	TABLESTORE的Instance名称
tableName	TABLESTORE的Table名称

说明:

- TABLESTORE 结果表须定义有 PrimaryKey，输出数据以 Update 方式追加到现有 TABLESTORE 表，即必须搭配使用 REPLACE INTO 进行写入。
- TABLESTORE 作为结果表仅支持 BIGINT、STRING 两种类型，即所有的 TABLESTORE 类型均在声明表创建时必须转换为 BIGINT、STRING。

29.7.7.4.3.4 Analytic DB 结果表

流计算支持使用 Analytic DB 作为结果输出。示例代码如下：

```
CREATE RESULT TABLE user (
id bigint,
name string,
age bigint PRIMARY KEY (id)
) WITH (
type='ads', url='jdbc:mysql://localhost:3306/javademo',
username='username',
password='password',
tableName='table'
);
```

其中 WITH 配置说明如下：

字段名称	说明
type	类型，这里必须是 Analytic DB。

字段名称	说明
username	能够读写该 Analytic DB 的用户名。
password	能够读写该 Analytic DB 的密码。
url	该 Analytic DB 库的jdbc连接串信息，采用jdbc格式填写。
tableName	该 Analytic DB 的数据表名称。



说明: Analytic DB 结果表须定义有 PrimaryKey，输出数据以 Update 方式追加到现有 Analytic DB 表，即必须搭配使用 **REPLACE INTO**进行写入。

29.7.7.4.4 创建临时表

如果计算的逻辑比较复杂用一个 StreamSQL 难以描述，阿里云流计算支持通过定义临时表的方式来简化开发的过程。需要明确的是，临时表仅仅用于辅助计算逻辑的描述，不会产生数据的物理存储。

```
CREATE TMP TABLE table_name
[(col_name data_type, ...[, PRIMARY KEY(col_name, ...)])]
);
```

代码示例如下：

```
CREATE TMP TABLE user (
id bigint,
name string,
age bigint, visit_count bigint
);

insert into user select
s_user.id,
s_user.name,
s_user.age,
s_visit.visit_count
from s_user join s_visit on s_user.id = s_visit.id;
```

29.7.7.5 DML语法

INSERT INTO

语法格式：

```
[INSERT|REPALCE] INTO TABLE tablename
SELECT [DISTINCT ]sel_expr1, sel_expr2...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
;
```

INSERT/REPALCE 语法主要用途在于更新数据表操作。注意事项如下：

- 流计算支持两类数据表更新：
  - 包括 INSERT，即 Append 写入目的数据存储，例如写入 DataHub 需要使用Insert写入，因为 DataHub 是一个队列只能提供 Append 写入；
  - 以及 REPLACE，即向目的数据更新写入，例如天猫双十一大屏，对于当前成交总额是不断更新上一次的写入结果因此是 REPALCE 写入。
  - 对于 INSERT 写入，RESULT 表不允许定义 PRIMARY KEY；
  - 对于 REPLACE 写入，RESULT 必须定义 PRIMARY KEY，且该 KEY 需要和实际数据存储保持一致。例如定义 TABLESTORE 的 PRIMARY KEY，须和 TABLESTORE 定义的 PRIMARY KEY 一致，否则可能出现将数据写错乱情况！
- 流计算不支持单独 SELECT 操作，当前在执行 SELECT 查询之前必须执行 INSERT 操作将结果保存起来。同时，须注意的是，一个 SQL 文件支持多个源表输入和多个结果表输出。
- 只有 RESULT 表和 TMP 表可以执行 INSERT 操作，且每张表只能执行一次 INSERT 操作，DIM 表和 STREAM 表不能执行 INSERT 操作。操作约束如下表：

表类型	操作约束
源表	只能引用，不可更新
维表	仅支持更新（INSERT）
结果表	仅支持更新（INSERT）
临时表	只能引用，不可更新

- 指定 DISTINCT 选项时，将符合条件的记录进行去重输出。
- sel_expr 可以为表中的列名，或计算表达式，支持”*”表示表中所有的列。
- table_reference 支持表的Join及子查询。
- 如果使用了聚合函数，则要指定 GROUP BY 分类的列。

子查询

普通的 SELECT 是从几张表中读数据，如 SELECT column_1, column_2 ... FROM table_name，但查询的对象也可以是另外一个 SELECT 操作，需要注意的是子查询必须加 别名。代码如下：

```
INSERT INTO result_table
 SELECT * from
(
SELECT t.a,
sum(t.b) AS sum_b
FROM t1 t GROUP BY t.a) t1
WHERE  t1.sum_b > 100;
```

UNION ALL

语法格式：

```
select_statement
UNION ALL
select_statement
```

UNION ALL 将两个流给合并起来，要求两个流的字段完全一模一样（字段个数，字段顺序）。示例代码如下：

```
SELECT a.id,
a.col1,
a.col2
FROM stream_tablea a
WHERE  a.col1 = 1 UNION
ALL
SELECT b.id,
b.col1,
b.col2
FROM stream_tableb b
WHERE  b.col2 = 1 GROUP
BY b.col1,b.col2
```

JOIN

语法格式：

```
join_table:
table_reference JOIN table_factor [join_condition]
| table_reference {LEFT OUTER |INNER} JOIN table_reference join_condition

table_reference: table_factor
| join_table

table_factor: tbl_name [alias]
| table_subquery alias
| ( table_references )

join_condition:
ON equality_expression ( AND equality_expression )*
```

说明：

- 只支持等值连接。
- 对维表的连接，只支持一个连接键。
- LEFT JOIN 会从左表(shop) 那里返回所有的记录，即使在右表（sale_detail）中没有匹配的行。
- RIGHT OUTER JOIN 右连接，返回右表中的所有记录，即使在左表中没有记录与它匹配。
- FULL OUTER JOIN 全连接，返回左右表中的所有记录。在表中存在至少一个匹配时，INNER JOIN 返回行。关键字 INNER 可省略。

CASE WHEN

CASE WHEN 表达式可以根据表达式 value 的计算结果灵活返回不同的值，阿里云流计算支持两种

CASE WHEN 表达式：

```
CASE value
WHEN (_condition1) THEN result1 WHEN (_condition2) THEN result2
...
ELSE resultn END
```

或者

```
CASE
WHEN (_condition1) THEN result1 WHEN (_condition2) THEN result2 WHEN (_condition3) THEN
 result3
...
ELSE resultn END
```

- 实例一：

  ```
  SELECT CASE
  WHEN shop_name IS NULL THEN 'default_region' WHEN shop_name LIKE 'hang%' THEN
   'zj_region'
  END AS region FROM sale_detail;
  ```

- 实例二：

  ```
  SELECT
  CASE shop_name
  WHEN IS NULL THEN 'default_region' WHEN LIKE 'hang%' THEN 'zj_region'
  END AS region FROM sale_detail;
  ```

DISTRIBUTE BY

当前 select 子句、where 子句、group by 子句、distribute by 子句等可以包含以下表达式：

- Column 字段：直接 Column 名，用户可以使用 * 代表所有列。
- CASE WHEN：子句系统内置函数。

29.7.7.6 内建函数

29.7.7.6.1 数学函数

注意，数学运算函数的参数支持以下隐式转换：

int -> bigint -> double

string -> double

ABS

- 函数定义：

```
double abs(double number)
bigint abs(bigint number)
```

- 功能说明：返回绝对值。
- 参数说明：number 为 double 或 bigint 类型，输入为 bigint 时返回 bigint；输入为 double 时返回 double 类型。参数的隐式转换支持 int、string，其他类型参数不支持。
- 返回值：double 或者 bigint 类型，取决于输入参数的类型。若输入为 NULL，返回NULL。
- 示例：

```
abs(NULL) = NULL
abs(-1) = 1
abs(-1.2) = 1.2
abs("-2") = 2.0
abs(122320837456298376592387456923748) = 1.2232083745629837e32
```

完整 SQL 例子如下：

```
SELECT
abs(t.id) FROM Table t;
```

CEIL

- 函数定义：

```
bigint ceil(double value)
```

- 功能说明：返回不小于输入值 value 的最小整数。
- 参数说明：value 为 double 类型。
- 返回值：bigint 类型。若输入为 NULL，返回 NULL。
- 示例：

```
ceil(1.1) = 2
ceil(-1.1) = -1
```

E

- 函数定义：double E()
- 功能说明：获取常量 E。
- 返回值：double 类型，返回常量 E。

EXP

- 函数定义：double exp(double number)
- 功能说明：指数函数。返回 number 的指数值。

- 参数说明：number 为 double 类型。参数的隐式转换支持 int、bigint、string，其他类型参数不支持。
- 返回值：double 类型。若 number 为 NULL，返回 NULL。

FLOOR

- 函数定义：bigint floor(double number)
- 功能说明：向下取整，返回 number 的整数值。
- 参数说明：number 为 double 类型。
- 返回值：返回 bigint 类型。若 number 为 NULL，返回 NULL。
- 示例：

```
floor(1.2)=1
floor(1.9)=1
floor(0.1)=0
floor(-1.2)=-2
floor(-0.1)=-1
floor(0.0)=0
floor(-0.0)=0
```

LN

- 函数定义：double ln(double number)。
- 功能说明：返回 number 的自然对数。
- 参数说明：number 为 double 类型，若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常。若 number 为 NULL 返回 NULL，若 number 为负数或零，则抛异常。
- 返回值：double 类型。

LOG

- 函数定义：double log(double base, double x)。
- 功能说明：返回以 base 为底的 x 的对数。
- 参数说明：base 为 double 类型，若输入为 string 类型或 bigint 类型会隐式转换到 double 类型后参与运算，其他类型抛异常； x 为 double 类型，若输入为 string 类型或 bigint 类型会隐式转换到double 类型后参与运算，其他类型抛异常。
- 返回值：double 类型的对数值，若 base 和 x 中存在 NULL，则返回 NULL；若 base 和 x 中某一个值为负数或 0，会引发异常；若 base 为 1（会引发一个除零行为）也会引发异常。

LOG10

- 函数定义：double log10(double x)。
- 功能说明：返回以 10 为底的 x 的对数。
- 参数说明：x 为 double 类型。
- 返回值：double 类型的对数值，若 x 为 NULL，则返回 NULL；若 x 为负数，会引发异常。

LOG2

- 函数定义：double log2(double x)。
- 功能说明：返回以 2 为底的 x 的对数。
- 参数说明：x 为 double 类型。
- 返回值：double 类型的对数值，若 x 为 NULL，则返回 NULL；若 x 为负数，会引发异常。

PI

- 函数定义：double PI()。
- 功能说明：获取圆周率常量 PI。
- 返回值：double 类型，返回圆周率常量 PI。

POW

- 函数定义：double pow(double x, double y)。
- 功能说明：返回 x 的 y 次方，即 x^y。
- 参数说明：X 为 double 类型。 Y 为 double 类型。
- 返回值：double 类型。若 x 或 y 为 NULL，则返回 NULL。

RAND

- 函数定义：double rand([bigint seed])。
- 功能说明：返回 0~1 之间的 double 类型随机数如果使用了参数 seed，则以 seed 为种子。
- 参数说明：seed 为 bigint 类型，随机数种子，决定随机数序列的起始值。
- 返回值：double 类型，返回值区间为0~1。

ROUND

- 函数定义：double round(double number, [bigint decimal_places])。
- 功能说明：四舍五入到指定小数点位置。
- 参数说明：number 为 double 类型，decimal_place 为 bigint 类型常量，四舍五入计算到小数点后的位置。如果省略表示四舍五入到个位数。 默认值为0。

- 返回值：返回 double 类型。若 number 或 decimal_places 为 NULL，返回 NULL。

  说明: decimal_places 可以是负数。负数会从小数点向左开始计数，并且不保留小数部分；如果 decimal_places 超过了整数部分长度，返回0。

- 示例：

```
round(125.315) = 125
round(125.315, 0) = 125
round(125.315, 1) = 125.3
round(125.315, 2) = 125.32
round(125.315, 3) = 125.315
round(-125.315, 2) = -125.32
round(123.345, -2) = 100.0
round(NULL) = NULL round(123.345, 4) = 123.345
round(123.345, -4) = 0.0
```

SQRT

- 函数定义：double sqrt(double number)。
- 功能说明：计算平方根。
- 参数说明：number 为 double 类型，必须大于 0，小于0时抛出异常。参数的隐式转换支持 int、bigint、string，其他类型参数不支持。
- 返回值：返回 double 类型。若 number 为 NULL，返回 NULL。

29.7.7.6.2 日期函数

DATE_ADD

- 函数定义：string date_add(string startdate, int days)。
- 功能说明：从起始日期加上指定天数，返回新的日期。
- 参数说明：startdate 为string 类型日期，日期格式：yyyy-MM-dd 或 yyyy-MM-dd HH:mm:ss
- 参数说明：startdate 为 string 类型日期，日期格式：yyyy-MM-dd 或 yyyy-MM-dd HH:mm:ss。days 为 int 类型，指定偏移的天数，可为负数。
- 返回值：string 类型日期。任一输入参数是 NULL，返回 NULL。
- 示例：

```
date_add('2008-12-31', 1) = '2009-01-01'
date_add('2008-12-31', -1) = '2008-12-30'
```

DATE_SUB

- 函数定义：string date_sub(string startdate, int days)。
- 功能说明：从起始日期减去指定天数，返回新的日期。

- 参数说明：startdate 为 string 类型日期，日期格式：yyyy-MM-dd 或 yyyy-MM-dd HH:mm:ss。days 为 int 类型，指定偏移的天数，可为负数。
- 返回值：string 类型日期。任一输入参数是 NULL，返回 NULL。
- 示例：

```
date_sub('2008-12-31', 1) = '2008-12-30'
date_sub('2008-12-31', -1) = '2009-01-01'
```

DATEDIFF

- 函数定义：int datediff(string enddate, string startdate)。
- 功能说明：计算从 enddate 到 startdate 两个时间的天数差值。

  参数说明：startdate，enddate 为 string 类型日期，日期格式：yyyy-MM-dd 或 yyyy-MM-dd HH:mm:ss。
- 返回值：int 类型。任一输入参数是 NULL，返回 NULL。
- 示例：

```
datediff('2009-03-01', '2009-02-27') = 2
```

FROM_UNIXTIME

- 函数定义：string from_unixtime(bigint unixtime[, string format])
- 功能说明：将数字型的 unix 时间日期值 unixtime 转为日期值。
- 参数说明：unixtime 为 bigint 类型，秒数，unix 格式的#期时间值，其他类型抛出异常。 format 为指定输出日期格式。
- 返回值：string 类型的日期值，默认日期格式：yyyy-MM-dd HH:mm:ss，若指定日期格式按指定格式输出，任一输入参数是 NULL，返回 NULL。
- 示例：

```
from_unixtime(1394502696) = '2014-03-11 09:51:36'
from_unixtime(1394502696, 'yyyyMMddHH') = '2014031109'
```

UNIX_TIMESTAMP

- 函数定义：

```
bigint unix_timestamp()
bigint unix_timestamp(string date)
bigint unix_timestamp(string date, string format)
```

- 功能说明：获取指定日期对应的秒数（从1970-01-01 00:00:00起至今），未指定参数时为当前系统时间的秒数。

- 参数说明：date 为 string 类型日期，默认日期格式：yyyy-MM-dd HH:mm:ss。 format 为 string 类型，指定输入参数日期格式。
- 返回值：bigint 类型，输入参数是 NULL，返回 NULL。
- 示例：

```
unix_timestamp() = 1394502696
unix_timestamp('2009-03-20 11:30:01') = 1237573801
unix_timestamp('2009-03-20', 'yyyy-MM-dd') = 1237532400
```

TO_DATE

- 函数定义：`string to_date(string timestamp)`。
- 功能说明：返回时间戳字符串中的日期字符串。
- 参数说明：timestamp 为 string 类型日期，日期格式：yyyy-MM-dd HH:mm:ss。
- 返回值：string 类型。若任一输入为 NULL，返回 NULL 值。
- 示例：

```
to_date('1970-01-01 00:00:00') = '1970-01-01'
```

DATE_FORMAT

- 函数定义：

```
string date_format(string date, string, to_format)
string date_format(string date, string from_format, string, to_format)
```

- 功能说明：对日期字符串进行格式转换。
- 参数说明：date 为 string 类型日期，默认日期格式：yyyy-MM-dd HH:mm:ss。 from_format 为指定输入日期格式， to_format 为指定输出日期格式。
- 返回值：string 类型日期。若任一输入为 NULL，返回 NULL 值。
- 示例：

```
date_format('1970-01-01 00:00:00', 'yyyyMMdd') = '19700101' date_format('1970-01-01', 'yyyy-MM-dd', 'yyyyMMdd') = '19700101'
```

YEAR

- 函数定义：`int year(string date)`。
- 功能说明：获取日期中的年份。
- 参数说明：date 为 string 类型日期，日期格式：yyyy-MM-dd 或 yyyy-MM-dd HH:mm:ss。
- 返回值：int 类型，若输入为 NULL，返回 NULL 值。
- 示例：

```
year('1970-01-01 00:00:00') = 1970
```

MONTH

- 函数定义：int month(string date)。
- 功能说明：获取日期中的月份。
- 参数说明：date 为 string 类型日期，日期格式：yyyy-MM-dd 或 yyyy-MM-dd HH:mm:ss。

  返回值：int 类型，若输入为 NULL，返回 NULL 值。
- 示例：

```
month('1970-11-01 00:00:00') = 11
```

DAY

- 函数定义：int day(string date)。
- 功能说明：获取日期中在一个月中的天数。
- 参数说明：date 为 string 类型日期，日期格式：yyyy-MM-dd 或 yyyy-MM-dd HH:mm:ss。
- 返回值：int 类型，若输入为 NULL，返回 NULL 值。
- 示例：

```
day('1970-11-01 00:00:00') = 1
```

WEEKOFYEAR

- 函数定义：int weekofyear(string date)。
- 功能说明：计算指定日期在一年中的第几周，周数取值区间1~53，周一为一周起始。
- 参数说明：date为string类型日期，日期格式：yyyy-MM-dd或yyyy-MM-dd HH:mm:ss。
- 返回值：int类型，若输入为NULL，返回NULL值，日期格式不对返回NULL。
- 示例：

```
weekofyear("1970-11-01 00:00:00") = 44
weekofyear("1970-11-01") = 44
```

HOUR

- 函数定义：int hour(string date)。
- 功能说明：获取时间戳中小时数。
- 参数说明：date 为 string 类型时间戳，时间戳格式： yyyy-MM-dd HH:mm:ss 或 HH:mm:ss。
- 返回值：int 类型，若输入为 NULL，返回 NULL 值。
- 示例：

```
hour('2009-07-30 12:58:59') = 12
hour('12:58:59') = 12
```

MINUTE

- 函数定义：int minute(string date)。
- 功能说明：获取时间戳中分钟数。
- 参数说明：date 为 string 类型时间戳，时间戳格式： yyyy-MM-dd HH:mm:ss 或 HH:mm:ss。
- 返回值：int 类型，若输入为 NULL，返回 NULL 值。
- 示例：

```
minute('2009-07-30 12:58:59') = 58
minute('12:58:59') = 58
```

SECOND

- 函数定义：int second(string date)。
- 功能说明：获取时间戳中秒数。
- 参数说明：date 为 string 类型时间戳，时间戳格式： yyyy-MM-dd HH:mm:ss 或 HH:mm:ss。
- 返回值：int 类型，若输入为 NULL，返回 NULL 值。
- 示例：

```
second('2009-07-30 12:58:59') = 59
second('12:58:59') = 59
```

29.7.7.6.3 条件函数

IF

- 函数定义：T if(boolean testCondition, T valueTrue, T valueFalseOrNull)
- 功能说明：如果 testCondition = true ，返回 valueTrue ，如果 testCondition = false 或者为NULL，返回 valueFalseOrNull 。
- 参数说明：testCondition 为布尔类型； valueTrue 和 valueFalseOrNull 的参数类型必须一致，可以为任意原生类型。
- 示例：if('abc' > 'ab' and 100 >= 999, 1, 0)

COALESCE

- 函数定义：T COALESCE(T v1, T v2, ...)
- 功能说明：返回从左到右第一个不为 NULL 的参数的值。
- 参数说明：v1， v2 ...: 参数类型必须一致。

CASE WHEN

- 函数定义：

```
T CASE
WHEN a THEN b [WHEN c THEN d]* [ELSE e]
END
```

- 功能说明：当 a = true ，则返回 b；当 c = true , 则返回 d；若都不满足返回 e；若没有 ELSE 子句，则返回 NULL。
- 参数说明：a，c 为布尔类型； b， d， e 的返回值类型必须一致。
- 示例：

```
case
when score < 60 then '不及格' when score < 90 then '及格' else '优秀'
```

29.7.7.6.4 字符串函数

ASCII2STR

- 函数定义：`string ascii2str(int ascii)`
- 功能说明：将数字转换成对应的 ascii 字符；任一参数为 NULL，返回NULL。
- 参数说明：ascii 取值在 1-127 之间，其他值抛出异常。
- 示例：

```
ascii2str(9) = '\t'
ascii2str(65) = 'A'
```

CONCAT

- 函数定义：`string concat(string a, string b, ...)`
- 功能说明：字符串连接函数，将所有参数拼接成新的字符串返回；任一参数为 NULL，返回 NULL。
- 示例：

```
concat('abc', 'def', 'gh') = 'abcdefgh'
concat('abc', NULL) = NULL
```

CONCAT_WS

- 函数定义：`string concat_ws(string separator, string a, string b, ...)`
- 功能说明：字符串连接函数，与 concat() 函数类型，用户可以指定拼接字符串之间的分隔符 separator ；任一参数为 NULL，返回 NULL。
- 示例：

```
concat_ws(':', 'abc', 'def', 'gh') = 'abc:def:gh'
concat(':', 'abc', NULL) = NULL
```

SUBSTR

- 函数定义：

```
string substr(string a, int start)
string substr(string a, int start, int len)
```

- 功能说明：获取字符串子串，截取从位置 start 开始长度为 len 的子串，若未指定 len 则截取到字符串结尾 start 从 1 开始，为负数时表示从字符串末尾计算位置； 任一参数为 NULL，返回 NULL。
- 示例：

```
substr('galaxy_sql', 5) = 'xy_sql'
substr('galaxy_sql', -5) = 'y_sql'
substr('galaxy_sql', 5, 2) = 'xy'
```

LOWER

- 函数定义：

```
string lower(string a)
string lcase(string a)
```

- 功能说明：将字符串中的大写字母转成小写字母，若参数为 NULL，返回 NULL 。
- 示例：lower('fOoBaR') = 'foobar'

UPPER

- 函数定义：

```
string upper(string a)
string ucase(string a)
```

- 功能说明：将字符串中的小写字母转成小写字母，若参数为 NULL，返回 NULL 。
- 示例：upper('fOoBaR') = 'FOOBAR'

TRIM

- 函数定义：string trim(string a)
- 功能说明：删除字符串两端的空格，若参数为 NULL，返回 NULL 。
- 示例：trim(' foobar\t ') = 'foobar\t'

LTRIM

- 函数定义：string ltrim(string a)

- 功能说明：删除字符串左端的空白字符，若参数为 NULL，返回 NULL 。
- 示例：ltrim(' foobar ') = 'foobar '

RTRIM

- 函数定义：string rtrim(string a)
- 功能说明：删除字符串右端的空格，若参数为 NULL，返回 NULL 。
- 示例：rtrim(' foobar ') = ' foobar'

LPAD

- 函数定义：string lpad(string str, int len, string pad)
- 功能说明：字符串 str 左端填充若干个字符串 pad，直到新的字符串达到指定长度 len 为止，任一参数为 NULL，返回 NULL。
- 示例：

```
lpad('hi', 5, '??') = '???hi'
lpad('hi', 1, '??') = 'h'
lpad('---', 10, 'abc') = 'abcabca---'
```

RPAD

- 函数定义：string rpad(string str, int len, string pad)
- 功能说明：字符串 str 右端填充若干个字符串 pad，直到新的字符串达到指定长度 len 为止，任一参数为NULL，返回NULL 。
- 示例：

```
rpad('hi', 5, '??') = 'hi???'
rpad('hi', 1, '??') = 'h'
rpad('---', 10, 'abc') = '---abcabca'
```

LENGTH

- 函数定义：int length(string str)
- 功能说明：计算字符串长度，任一参数为NULL，返回NULL 。
- 示例：

```
length('hi') = 2
length('我') = 1
length(NULL) = NULL
```

REPEAT

- 函数定义：string repeat(string str, int n)
- 功能说明：以字符串 str 重复 n 产生新的字符串，任一参数为 NULL，返回 NULL 。

- 示例：repeat('hi', 2) = 'hihi'

REVERSE

- 函数定义：string reverse(string str)
- 功能说明：反转字符串，任一参数为 NULL，返回 NULL 。
- 示例：reverse('abc') = 'cba'

SPLIT_EX

- 函数定义：string split_ex(string str, string sep, int index)
- 功能说明：以 sep 作为分隔符，将字符串 str 分隔成若干段，取其中的第 index 段，取不到返回NULL， index 从0开始任一参数为 NULL，返回 NULL。
- 示例：

```
split_ex('1.2.3.4', '.', 1) = '2'
split_ex('1.2.3.4', '.', -1) = NULL
split_ex('1.2.3.4', '.', 4) = NULL
```

REGEXP_REPLACE

- 函数定义：string regexp_replace(string str, string pattern, string replacement)
- 功能说明：用字符串 replacement 替换字符串 str 中正则模式为 pattern 的子串，返回新的字符串。
- 注意：正则表达式中的斜杠（\)需要转义，任一参数为 NULL，返回 NULL。
- 示例：

```
regexp_replace('100-200', '(\\d+)', 'num') = 'num-num'
regexp_replace('2014-03-13', '-', '') = '20140313'
regexp_replace('foobar', 'oo|ar', '') = 'fb'
```

REGEXP_EXTRACT

- 函数定义：string regexp_extract(string str, string pattern, int index)
- 功能说明：使用正则模式 pattern 匹配抽取字符串 str 中的第 index 个子串， index 从1开始。
- 注意：正则表达式中的斜杠（\)需要转义，任一参数为 NULL，返回 NULL 。
- 示例：

```
regexp_extract('foothebar', 'foo(.*?)(bar)', 2) = 'bar'
regexp_extract('100-200', '(\\d+)-(\\d+)', 1) = '100'
```

INSTR

- 函数定义：int instr(string str, string substr)

- 功能说明：返回字符串 str 中第一次出现字符串 substr 的位置，从1开始；若不存在该子串则返回0。任一参数为 NULL，返回 NULL。
- 示例：

```
instr('foobar', 'foo') = 1
instr('foobar', 'abc') = 0
```

KEYVALUE

- 函数定义：`string keyvalue(string str, string split1, string split2, string key_name)`
- 功能说明：解析字符串中的 key-value 对，获取指定 key_name 对应的值；若不存在则返回 NULL，任一参数为 NULL，返回 NULL 。
- 示例：

```
keyvalue('k1=v1;k2=v2', ';', '=', 'k2') = 'v2'
keyvalue('k1:v1,k2:v2', ',', ':', 'k3') = NULL
```

HASH

- 函数定义：`int hash(string str)`
- 功能说明：返回字符串的 hashCode() 的绝对值，任一参数为 NULL，返回 NULL。

MD5

- 函数定义：`string md5(string str)`
- 功能说明：返回字符串的 MD5 值，如果参数为 NULL 或空串，返回空串。

29.7.7.6.5 聚合函数

COUNT

- 函数定义：`BIGINT COUNT([DISTINCT] value)`
- 功能说明：计算记录数。
- 参数说明：value 可以是任意的变量。
- 返回值：bigint 类型。
- 注意事项：
  - 目前 StreamCompute 不支持 count(*) 操作。
  - 指定 distinct 操作符可以使返回的结果中不包含重复的行。
  - 示例：`SELECT COUNT(a.col1) FROM tbla a;`

SUM

- 函数定义：

```
BIGINT SUM([distinct] value)
DOUBLE SUM([distinct] value)
```

- 功能说明：计算汇总值。
- 参数说明：alue 为字符串或数字。
- 返回值：输入参数为整型时返回 bigint，否则返回 double。
- 示例：SELECT SUM(distinct a.col1) FROM tbla a;

BLOOM_FILTER

- 函数定义：BIGINT bloom_filter(accuracy,max_count,[fresh_group],expr)
- 功能说明：count(distinct) 的布隆过滤器实现。
- 参数说明：accuracy 为 double 类型，在 0 到 1 之间，表示布隆过滤器的精度；max_count 为 bigint 类型，可能出现的 key 的最大数量；fresh_group可以为任意类型，设置该字段后档过滤器满后会重新启用一个新的过滤器 expr 为需要统计的字段。
- 返回值：BIGINT 类型。
- 示例：SELECT bloom_filter(0.99, 10000, thedate, a.col1) FROM tbla a group by thedate;

CARDINALITY

- 函数定义：BIGINT cardinality(accuracy, expr)
- 功能说明：count(distinct) 的基数估计实现。
- 参数说明：accuracy 为 double类型，表示基数估计的精度。 expr 为需要统计的列。
- 返回值：bigint 类型。
- 示例：SELECT cardinality(0.99, a.col1) FROM tbla a;

MAX

- 函数定义：MAX(value)
- 功能说明：计算最大值。
- 参数说明：除 object 以外的所有类型。
- 返回值：与 value 同类型。
- 示例：SELECT MAX(a.col1) FROM tbla a;

MIN

- 函数定义：MIN(value)
- 功能说明：计算最小值。
- 参数说明：除 object 以外的所有类型。
- 返回值：与 value 同类型。
- 示例：SELECT MIN(a.col1) FROM tbla a;

CONCAT_AGG

- 函数定义：concat_agg([linedelimiter,] value)
- 功能说明：连接本批对应字段，默认连接符"\n"。
- 参数说明：连接符，目前只支持字符串常量对应字段。
- 返回值：连接完成后新生成的字符串。
- 示例：SELECT concat_agg(',', a.col1) FROM tbla a;

CONTEXT

- 函数定义：

  ```
  T context(value: T)
  T context(distinct value: T)
  ```

- 功能说明：

  context(value: T)：获取本批的第一条数据。

  context(distinct value: T)：如果加了关键字 distinct ，表示基于该字段进行全局去重，只输出第一条。

- 参数说明：可输入任意产生类型，只接受一个参数。
- 返回值：返回值参数类型与输入参数类型一致。
- 应用场景：在传统 DB 中，即使表定义了primary key，但是当表中的一条记录被多次修改时，也会产生多条 binlog 日志，通过实时采集和同步过来的数据里就包含了这条记录的 多次更新。如何通过流计算此类数据呢？

  比如交易订单表，假设订单 id：order_id 是 primary key，交易状态有 gmv、alipay 等状态，如何按小时计算状态为 alipay 的订单数和成交金额：

  ```
  select date_format(pay_time, 'yyyyMMddHH'), count(1) as order_cnt, sum(ali_fee) as ali_fee fr
   ( select context(distinct order_id) as order_id, context(gmt_modify) as pay_time, context(ali
  from pay_order where status = 'alipay' group by order_id
  ) t group by date_format(pay_time, 'yyyyMMddHH')
  ```

29.7.7.6.6 散分函数

SPLIT

- 函数定义：split(string source, [string splitchar])
- 功能说明：将字符串按照 splitchar 分割成多个部分，如果不指定 splitchar，则默认采用逗号分割。
- 参数说明： source 为 string 输入的待分割字符串。 splitchar 为 string分割符。
- 返回值：返回字符串分割后每列。
- 示例： select split('abc,bcd,cde', ',') from dual;

  结果输出3条记录：abc， bcd， cde。

29.7.7.6.7 窗口函数

流计算后续计划在商业化前支持窗口函数。

当前流计算暂不支持窗口函数，用户需要通过 Group By 进行间接表达。例如：按照每小时统计用户的 PV 数量，实际上这个是一个滚动窗口需求，用户可以根据 Group By 完成。

```
select
date_format(click_time, 'yyyy-MM-dd HH') as pv_time, count(1) as pv_count
from
click_stream group by
date_format(click_time, 'yyyy-MM-dd HH');
```

这样按小时聚合用户点击 PV，即可完成滚窗的间接表达效果。

29.7.8 SQL最佳实践

为最大化降低用户使用 StreamSQL 成本，我们提供了最佳实践和技巧的 StreamSQL 写法。

29.7.8.1 DDL声明技巧

流计算中，对于表的创建实际上只是声明对于外部表的引用，因此有部分特殊技巧可以帮助大家不易犯错。

29.7.8.1.1 使用"数据开发"进行创建

使用 DDL 语句创建外部表，流计算推荐使用自带的 DDL 生成工具进行生成，而不推荐用户手工编写 DDL。详情参看：*DDL辅助生成* 。

29.7.8.1.2 DDL大小写忽略

SQL 语句中对于字段大小写的定义是忽略的，因此在 StreamSQL 创建 table 中，务必不要使用大小写区分字段。如果外部表正好存在使用大小写区分字段名，例如两个字段，一个是 Name，另外一个是 name。目前 StreamSQL 暂时无法支持该类情况。

-- 以下SQL为错误案例！

```
create result table ots_reference (
id bigint,
Name string,
name string
);
```

29.7.8.1.3 支持多次声明

前面提到，表的创建实际上只是声明对于外部表的引用，那么实际上我们可以对于同一张表进行多次声明引用。例如，我们可以对一个 DataHub 的 Topic 声明为结果表，同样我们亦可在同一个流计算 Job 里面将其声明为一个源头表，这样我们在后续 SQL 中可以将该 Topic 用作数据源使用。

```
create stream table source_table (
id string,
name string
) with (
同一个外部DataHub的Topic
);

create stream table dest_table (
id string,
name string
) with (
同一个外部DataHub的Topic
);
```

特别注意，必须保证在同一个 Job 里面，声明的表名（实际上是别名）必须不重名，否则 SQL 解析器无法判断 SQL 的意图。

29.7.8.1.4 支持部分字段声明

对于一个外部表声明，例如源表有 a，b，c 三列，流计算声明该表引用时候，可以只用填写 a，b 两列，例如：

```
create stream table source_table (
a string, b string
)
```

这样，后续的 SQL 仅仅只能引用 source_table 的 a、b 两个字段。对于 STREAM 和 DIM 的外部表声明，由于不涉及到外部表修改，较为简单。如果对于目标表声明，如果声明部分字

段，如果未声明字段恰好不允许为空，那么插入过程底层的数据源是报错的。请务必注意这类情况的发生。

29.7.8.2 广告效果分析

29.7.8.2.1 问题描述

某女性电商公司为了吸引更多女性向新客户，决定在多家门户网站女性频道投放公司广告。为进一步追踪外投广告的效果并修改广告投放策略，公司决定实时对外投广告点击来源进行效果分析。业务需求是对于不同的活动页面，分类统计来源信息。

29.7.8.2.2 解决方案

由于涉及到广告点击日志产生源在 ECS 的 Ngnix 上，公司的数据开发人员使用 Fluentd 将上述日志数据采集进行 DataHub，DataHub Topic 定义如下：

字段名	类型	注释
click_time	String	访问时间，格式为yyyy-MM-dd HH:mm:ss。
from_url	String	访问来源，格式为http://xxxx.xxxx.xxx/xxxx/xxxx。

为最终实时展示来源效果，公司决定使用 RDS 进行计算结果存储，同时使用 BI 报表的饼状图来实时展现 RDS 中存储的结果。

其中 RDS 表定义如下：

字段名	类型	注释
ad_date	varchar(16)	主键，点击日期，格式为yyyy-MM-dd。
from_site	varchar(128)	访问来源网站，xxxx.xxxx.xxx。
from_count	bigint	来源数量。

```
replace into result_rds select
to_date(click_time) as ad_date, --以天为维度分析点击来源
regexp_extract(from_url, 'http(s?)(://)([^/]+)(.*?)', 3) as from_site, --获取URL域名
 count(1) as from_count
from
click_stream group by
to_date(click_time),
regexp_extract(from_url, 'http(s?)(://)([^/]+)(.*?)', 3);
```

29.7.8.3 PV/UV统计

29.7.8.3.1 问题描述

当前公司正处于业务快速爆发期，CEO 希望能够实时观察到当前公司的 PV/UV 情况。而当前使用的离线计算模型，仅能够每天拿到一份汇总 PV/UV 信息，CEO 对于时延过大表示不满。

29.7.8.3.2 解决方案

由于涉及到广告点击日志产生源在 ECS 的 Ngnix 上，公司的数据开发人员使用 Fluentd 将上述日志数据采集进行 DataHub，DataHub Topic 定义如下：

字段名	类型	注释
uid	String	用户ID
cookie	String	用户Cookie
click_time	String	访问时间，格式为yyyy-MM-dd HH:mm:ss
click_url	String	访问URL

其中 RDS 表定义如下：

字段名	类型	注释
count_date	varchar(16)	主键，点击日期，格式为yyyy-MM-dd
pv	bigint	指定日期的PV
uv	bigint	指定日期的UV

涉及到的 PV/UV 操作比较简单，仅需要以日期维度统计 click_url 的总个数即可：

```
replace into result_rds select
to_date(click_time) as count_date, count(1) as pv, count(distinct(cookie)) as uv
from
click_stream group by
to_date(click_time);
```

29.7.8.4 实时用户路径追踪

29.7.8.4.1 问题背景

用户的访问日志保留了很多用户的偏好和行为，通过实时追踪用户访问路径，跟踪用户行为，判断当前产品设计缺陷。例如，客户在电话咨询客服部之前，客服可以通过实施路径追踪知道用户在最近一天在当前网站访问路径，以方便客服部有针对性的解答客户问。

说明: 用户访问的漏洞分析应该在离线计算解决，但实时路径追踪一般通过流计算解决。

29.7.8.4.2 解决方案

用户访问日志集中在 ECS 上，选择使用 Logstash+DataHub 可以完成日志数据采集任务。

DataHub 的 Topic 定义如下：

字段名	类型	注释
uid	String	#户ID
click_time	String	访问时间，格式为yyyy-MM-dd HH:mm:ss
click_url	String	访问URL

用户的行为数据需要按照用户粒度，以时间范围进行高并发低时延，我们推荐使用 TABLESTORE 保存用户数据，TABLESTORE 表格式设计如下：

字段名	类型	注释
uid	String	主键PK，#户标示符
click_time	String	主键PK，访问时间，格式为yyyy-MM-dd HH:mm:ss
click_url	String	访问URL

对于每条日志，使用流计算进行简单清洗即可写入 TABLESTORE。

29.7.9 参数说明

29.7.9.1 资源相关

stream.work.size

指定一个作业 worker 数，worker 资源（CU）数。

整型参数，可选，默认等于 DataHub 数据源的 shard 数。

29.7.9.2 并发相关

- stream.datasource.parallelism

  指定数据源读取组件的并发数。

  整型参数，可选，默认等于 DataHub 数据源的 shard 数。

- stream.model.parallelism

  指定所有计算组件的并发数。

  整型参数，可选，默认等于作业的 worker 数。

- stream.output.parallelism

  指定结果输出组件的并发数。

  整型参数，可选，默认等于作业的 worker 数。

- ${MODEL}.parallelism

  指定某个 Model 的并发。

  整型参数，可选。

- ${MODEL}.[mapper|reducer|merger].parallelism

  指定 Model 的指定阶段的并发。

  整型参数，可选。

29.7.9.3 吞吐相关

- stream.batch.size

  用于指定一个批次的数据集大小

  整型参数，可选，不同流式存储指定不一样的值，DataHub 推荐 128，Log 推荐 32。

  如果数据源是 Datahub，每个批次最多会从每个 shard 里读取 stream.batch.size 个消息，如果最新数据已经读完则本批次结束。

29.7.9.4 超时相关

- stream.message.timeout

  指定流计算处理一个批次数据的超时时间。

  整型参数，单位秒，可选，默认为180 秒。

  如果一批数据的处理时间超过 timeout 即认为数据处理失败（即使最终处理成功了），会触发数据的重新计算。Timeout 时间设置太小导致误判处理失败的概率增大，设置太大会导致发现系统故障进行 failover 的时间变长。

29.7.10 错误码

以下是流计算 WebConsole 错误码汇总，格式为：

```
错误码 = 错误信息
```

其中错误信息包含信息占位符，信息占位符在实际运行过程中将由具体错误信息填充。

29.7.10.1 910 system

- 910100=系统错误：内部错误，请联系系统管理员修复!
- 910111=系统错误：SQL 错误，该错误通常由于环境部署导致，请联系管理员修复。
- 910112=系统错误：数据库更新失败，结果不为1。请联系管理员。
- 910113=系统错误：数据库删除失败，结果不为1。请联系管理员。
- 910121=系统错误：OSS 操作异常，该错误通常由于环境部署导致，请联系管理员修复。\n 错误详情： {0}。
- 910122=系统错误：缺少参数 {0}，该错误通常由于代码错误导致，请联系管理员。
- 910123=系统错误：参数校验错误，请联系管理员。
- 910124=系统错误：文件读取失败，请联系管理员。
- 910125=系统错误：文件写入失败，请联系管理员。
- 910126=系统错误：HTTP响应格式错误，请联系管理员。
- 910127=操作错误：cache重复错误，请联系管理员。
- 910128=系统错误：加密密钥获取失败，请联系管理员。\n错误详情： {0}
- 910129=不支持该操作错误。
- 910131=冒烟失败：{0}。
- 910132=操作错误：名称[{0}]格式不合法，请重新命名。
- 910133=模板文件目录不存在。
- 910134=模板文件读取错误。
- 910135=模板文件名称格式错误。
- 910136=TT 开关参数错误：{0}。
- 910137=TT 服务异常：{0}。

29.7.10.2 911 project

- 911001=操作错误： 项目名称格式错误，项目名称仅支持大小写字符及数字，请修改您的项目名。

- 911002=操作错误：项目名称已经存在，项目名称不允许重名，请修改您的项目名。
- 911003=操作错误：项目管理员已经超过最大值，不允许再添加管理员。
- 911004=操作错误：项目修改错误，项目必须保证至少一个管理员。
- 911005=操作错误：项目参数错误，项目创建人仅能为一个。
- 911006=系统错误：项目参数错误，项目id不能为空。
- 911007=操作错误：项目参数错误，项目不存在， projectId:{0} 。
- 911008=系统错误：项目状态错误，请联系管理员。 \n 错误详情： ProjectId:{0} 。
- 911009=系统错误：入参用户 id 为空，请联系管理员。
- 911010=系统错误：该用户不在项目，请联系管理员。\n 错误详情： User:{0}，Project:{1}。
- 911011=操作错误：该用户已经在项目，请检查是否误操作。
- 911012=操作错误：删除用户不在项目，请检查是否误操作。
- 911013=系统错误：用户类型错误，请联系管理员。
- 911014=系统错误：项目地域参数错误，请检查是否误操作。\n 错误详情： Region{0}。
- 911015=操作错误：创建项目已经操作最大值({0})，请检查是否误操作。
- 911016=操作错误：当前项目[{0}]被锁住。

29.7.10.3 912 execution

- 912001=执行环境 name 为空。
- 912002=从 SDS 获取 execution 错误。
- 912003=操作错误： 执行引擎参数错误，执行引擎不能为空或者引擎不存在。
- 912004=操作错误： 引擎 [{0}] 已存在，请检查是否误操作。
- 912005=操作错误： 项目 [{0}] 中执行引擎 [{1}] 不存在，请联系您的项目管理员添加相关执行引擎。
- 912006=系统错误： 引擎类型 [{0}] 错误，请联系系统管理员。
- 912007=操作错误： 引擎 [{0}] 连接失败，请检查您的配置信息并尝试重新提交。
- 912008=操作错误： 引擎 [{0}] 已经被作业使用，不能删除！

29.7.10.4 913 folder

- 913001=操作错误： 您提交的文件夹名称不能为空，请检查您的文件夹名称。
- 913002=操作错误： 您操作的文件夹不存在，请尝试重新操作。
- 913003=操作错误： 当前项目下，文件夹数目超过最大数{0}，当前值{1}。
- 913004=操作错误： 文件夹名称非法，名称仅允许[A-Za-z0-9_中文]，请修改您的文件夹名称。

- 913005=系统错误： 当前出现子目录类型父目录不一致情况，请联系管理员。
- 913006=操作错误： 文件夹深度超过最大数{0}，当前值{1}。
- 913007=参数错误： 创建文件夹失败，参数校验异常。请联系管理员。
- 913008=操作错误： 创建文件夹失败，父文件夹不存在。请尝试重新操作。
- 913009=操作错误： 创建文件夹失败，存在同名文件夹 {0}。请尝试换名称重新操作。
- 913010=操作错误： 修改文件夹失败，文件夹不存在。请尝试重新操作。
- 913011=操作错误： 修改文件夹失败，父文件夹不存在。请尝试重新操作。
- 913012=操作错误： 修改文件夹失败，存在同名文件夹 {0}。请尝试换名称重新操作。
- 913013=操作错误： 删除文件夹失败，文件夹非空。请先清空文件夹。
- 913014=系统错误： 查询根文件夹失败，文件类型不是根目录。请联系管理员。
- 913015=操作错误： 删除文件夹失败，根文件夹无法删除。请检查是否误操作。
- 913016=操作错误： 根文件夹 {0} 无法修改，请检查是否误操作。
- 913017=操作错误： 移动文件夹失败，不能将文件夹移动到当前文件夹。
- 913018=操作错误： 移动文件夹失败，目的文件夹不存在。请尝试重新操作。
- 913019=参数错误： 移动文件夹失败，不能出现环路。请联系管理员。
- 913020=参数错误： 查找祖先节点异常，不能出现环路。请联系管理员。

29.7.10.5 914 file

- 914001=操作错误：文件不存在，请尝试重新操作。
- 914002=操作错误：文件被他人锁定，请先点击编辑后后执行该操作。
- 914003=参数错误：文件 id 不能为空，请联系管理员。
- 914004=操作错误：文件版本不存在，请尝试重新操作。
- 914005=系统错误：文件版本内容格式错误，请联系系统管理员。
- 914006=系统错误：文件类型与文件接口类型不匹配，请确保调用了正确的文件接口，并且文件类型与之一致。
- 914007=操作错误：文件名不能为空，请修改文件名并尝试重新操作。
- 914008=文件类型不能为空。
- 914009=操作错误：同名文件{0}已经存在，请尝试修改名称再次提交。
- 914011=操作错误：上线任务操作必须填写版本注释信息，请尝试重新填写注释。
- 914012=操作错误：指定版本的文件不存在，请尝试重新操作。
- 914013=操作错误：文件名格式不正确，当前支持格式为[A-Za-z0-9_]。

- 914014=系统错误：文件类型与目录类型不匹配，请联系系统管理员。
- 914015=系统错误：文件锁定状态错误，请联系管理员。
- 914016=文件不存在。
- 914017=文件被他人锁定，请先抢锁后执行该操作。

29.7.10.6 915 node

- 915001=操作错误：作业版本超过最大数{0}，当前值{1}。请在数据开发，删除该作业部分版本再尝试重新提交。
- 915002=操作错误：作业引用 Jar 包的个数不能超过5个。请考虑线下合并部分Jar包。
- 915003=操作错误：当前项目的作业数量超过最大数{0}，当前值{1}。请尝试先删除部分作业。
- 915004=参数错误：作业类型错误，该错误通常属于在编程错误。
- 915005=操作错误：storm和chitu任务只能引用一个Jar包，请考虑线下合并部分Jar包。
- 915006=操作错误：同名作业[{0}]已存在，请尝试修改名称。
- 915007=操作错误：节点[{0}]不存在，可能由于并发操作导致，请尝试重新操作。
- 915008=节点[{0}]已下线。
- 915009=节点[{0}]已删除。
- 915010=系统错误：节点[{0}]状态错误，请联系管理员。
- 915011=系统错误：节点[{0}]类型错误，请联系管理员。
- 915012=操作错误：节点存在下游引用节点：{0}。
- 915013=系统错误：节点[{0}]找不到解析器，请联系管理员。
- 915014=操作错误：节点 TAG[{0}]重复，请删除部分重复 TAG。
- 915015=操作错误：节点 TAG 数目不能大于3，请删除部分 TAG
- 915016=操作错误：节点TAG[{0}]校验错误，应该为 [A-Za-z0-9_.\\-中文]'{'1,8'}'。请修改 TAG。

29.7.10.7 916 resource

- 916001=操作错误：创建资源失败，错误信息为： {0}。请尝试重新操作。
- 916002=操作错误：修改资源失败，错误信息为： {0}。请尝试重新操作。
- 916003=操作错误：删除资源失败，错误信息为： {0}。请尝试重新操作。
- 916004=操作错误：同名资源已存在，请尝试换名称重新操作。
- 916005=操作错误：资源名非法，资源只能包含[A-Za-z_.]，且以字母开头。
- 916006=操作错误：资源不存在，请尝试重新操作。
- 916007=操作错误：下载资源失败，请尝试重新操作。

- 916008=操作错误：上传资源失败，请尝试重新操作。
- 916009=操作错误：删除资源失败，存在引用节点。请先解除引用后再执行删除操作。
- 916010=参数错误：资源id不能为空，请联系管理员。
- 916011=操作错误：资源文件超过最大数{0}，当前值{1}。请尝试删除部分资源文件。
- 916012=操作错误：资源版本超过最大数{0}，当前值{1}。请尝试删除部分版本。
- 916013=操作错误：资源[{0}]已存在，请尝试换名称重新操作。
- 916014=操作错误：资源[{0}]不存在，请尝试重新操作。
- 916015=参数错误：资源类型[{0}]错误，请联系管理员。
- 916016=参数错误：资源[{0}]内容为空，请检查您的操作。
- 916017=操作错误：资源不存在，请尝试重新上线作业。

29.7.10.8 917 instance

- 917001=操作错误：该作业存在运行的实例[{0}]未结束，请先停止该运行的作业。
- 917002=操作错误：该作业运行的实例[{0}]不存在，请尝试重新操作。
- 917003=操作错误：该作业运行的实例[{0}]未结束。
- 917004=操作错误：实例[{0}]状态错误：{1}。
- 917005=操作错误：该作业正在提交，不允许终止。

29.7.10.9 918 storage

- 918001=外部错误：连接外部数据存储连接失败，请联系相关存储技术支持团队。\n错误详情为： {0}。
- 918002=参数错误：指定的数据存储不存在，请尝试重新操作。
- 918003=外部错误：创建DataHub的Topic失败，请联系相关存储技术支持团队。\n错误详情为：{0}。
- 918004=外部错误：预览DataHub数据失败，请联系相关存储技术支持团队。\n错误详情为：{0}。
- 918005=外部错误：预览外部存储数据失败，请联系相关存储技术支持团队。\n错误详情为：{0}。
- 918006=外部错误：获取外部存储表详情失败，请联系相关存储技术支持团队。\n错误详情为：{0}。
- 918007=外部错误：获取外部存储表Schema失败，请联系相关存储技术支持团队。\n错误详情为： {0}。

- 918008=外部错误：获取外部存储表列表失败，请联系相关存储技术支持团队。\n错误详情为：{0}。
- 918009=操作错误：DataHub的名称格式错误，目前格式为[a-zA-Z][a-zA-Z0-9_]{0,127}。
- 918010=操作错误：流计算已经存在相同的数据存储，请避免重复添加。
- 918011=外部错误：连接外部数据存储连接失败，请联系相关存储技术支持团队。\n错误详情为： {0}。
- 918012=操作错误：数据源数目过最大数{0}，当前值{1}。请尝试删除部分数据源。
- 918013=操作错误：MaxCompute Hub 分区表不能作为结果表。
- 918014=外部错误：创建MaxCompute Hubtable错误。\n错误详情为： {0}。
- 918015=外部错误：预览TABLESTORE数据发生错误，请尝试重新操作。\n错误详情为： {0}。
- 918016=文件超过最大上传限制。
- 918017=上传文件的数据列数与topic中的列个数不一致。
- 918018=上传文件的数据列类型错误。
- 918019=上传文件的数据列解析格式错误， line = [{0}]。
- 918020=上传文件数据上传全部失败。
- 918021=上传文件数据部分上传失败。
- 918022=没有可用的激活的 shard。

29.7.10.10 919 user

- 919001=参数错误：用户信息不存在，请检查您的操作。\n错误信息为： {0}。
- 919002=操作错误：用户不能删除自身账号，请检查是否误操作。 \nUserId： {0}。
- 919003=操作错误：用户已经存在，请不要重复添加。
- 919004=操作错误：用户不能修改自身角色， \nUserId： {0}。
- 919005=外部错误：用户登出发生系统错误，请尝试重新操作。\n错误信息为： {0}。
- 919006=系统错误：管理员id不能为空，请尝试重新操作。
- 919007=操作错误：管理员{0}不存在，请尝试重新操作。
- 919008=参数错误：登录用户ID不正确或者不存在于集团内部账号中， \nUserId： {0}。

29.7.10.11 920 region

- 920001=操作错误：同名区域已添加，请勿重复操作。
- 920002=操作错误：区域名称不能为空，请添加名称后尝试重新提交。
- 920003=操作错误：区域 url 不能为空，请添加 url 后尝试重新提交。

- 920004=操作错误：ops 不能为空，请添加 ops 后尝试重新提交。
- 920005=操作错误：区域名称不能存在，请修改区域名称后尝试重新提交。
- 920006=操作错误：ops：{0} 不可用，请添加 ops 后尝试重新提交。

29.7.10.12 921 debug

- 921001=操作错误：调试服务器忙，请稍后再试。
- 921002=操作错误：当前操作用户为非法用户，请检查您是否具备调试该作业的权限。
- 921003=操作错误：调试过程中调试作业被他人编辑，请联系该作业当前编辑人进行协商。
- 921004=参数错误：调试参数异常，请尝试重新登录并打开作业进行调试。
- 921005=参数错误：操作的调试进程不存在，请重新打开作业进行调试工作。
- 921006=参数错误：调试文件不存在，请尝试重新上传一份调试文件。
- 921007=操作错误：解析 SQL 出现错误，请检查您的 SQL 文件。\n错误详情为： {0}
- 921008=操作错误：调试的数据文件不能大于{0}，请选择较小的文件重新上传。
- 921009=操作错误：调试需要对每张表提供测试数据，表{0}数据未提供。请上传测试文件。
- 921010=操作错误：上传的调试文件检测出非法 Schema 格式，请下载模板文件查看 Schema 格式。
- 921011=操作错误：调试的数据总行数必须小于 {0}，请选择较小的文件重新上传。
- 921012=操作错误：上传调试文件失败，请修复错误并尝试重新上传文件。\n错误详情为： {0}。

29.7.10.13 922 AUTH

- 922001=操作错误：权限认证失败，请尝试重新操作或者联系您的管理员。
- 922002=操作错误：无法完成本次操作，您不是系统管理员。
- 922003=操作错误：无法完成本次操作，您不是本项目管理员。
- 922004=操作错误：无法完成本次操作，您不是本项目成员。
- 922005=系统错误：内部认证体系错误，该错误通常由于代码错误导致，请联系管理员。\n 错误详情为： {0}。
- 922006=系统错误：未知的内部权限角色，该错误通常由于代码错误导致，请联系管理员。\n 错误详情为： {0}。
- 922007=系统错误：内部认证角色不能为 null，该错误通常由于代码错误导致，请联系管理员。
- 922008=系统错误：无法找到认证角色，该错误通常由于部署环境/编程错误导致，请联系管理员。
- 922009=操作错误：token 获取错误，请尝试重新操作。

- 922010=操作错误：session 超时或未登录，请尝试重新登录。
- 922011=操作错误：您未能授权进行本操作，请检查您的 RAM 授权。\n 错误详情为： {0}。
- 922012=操作错误：流计算 RAM Role 不存在，请参考相关文档进行手工添加。 \n错误详情为：{0}。
- 922013=操作错误：数加未能开通流计算服务，请先申请开通。
- 922014=操作错误：阿里云 RAM 子账号父账号不能为空。
- 922015=外部错误：阿里云 RAM 获取用户失败，请尝试重新操作。
- 922016=操作错误：阿里云账号类型不匹配，请尝试再次操作或者联系系统管理员。
- 922017=系统错误：该模块不显示，请联系系统管理员添加模块。
- 922018=操作错误：系统管理操作必须使用管理员的域名。
- 922019=操作错误：子账号没有 passRole 权限，请到 RAM 对该子账号赋权。\n 错误详情为：{0}。

29.7.10.14 923 alarm

- 923001=系统错误：报警类型错误，请尝试再次操作。
- 923002=系统错误：报警状态错误，请尝试再次操作。
- 923003=系统错误：报警发送失败，请联系管理员。

29.7.10.15 924 galaxy

- 924001=系统错误：调用底层流计算集群发生错误。\n 错误原因： {0}。
- 924002=操作错误：解析 SQL 出现错误。\n 错误原因： {0}。
- 924003=操作错误：赤兔解析异常。\n 错误原因： {0}。
- 924004=操作错误：STORM 名称 [{0}] 错误。
- 924005=操作错误：系统处理 STORM 作业异常异常。\n 错误原因： {0}。
- 924006=操作错误：解析参数文件异常，请检查您的参数文件是否符合参数格式。\n错误详情：PROPERTIES 文件错误：{0}。
- 924007=系统错误：流计算集群调用接口异常，错误详情： {0}。
- 924008=操作错误：流计算获取统计信息失败，请尝试重新操作。
- 924009=系统错误：调用底层流计算集群发生错误。\n 错误原因： {0}。

29.7.10.16 990 sdk

- 990001=编程错误： SDK 缺少鉴权参数： {0}。

- 990002=编程错误： SDK 鉴权失败： {0}。

30 大数据应用加速器

30.1 产品概述

阿里云 DTBoost 数据加速器产品从大数据应用落地点出发，提供了一套大数据应用开发套件，能够帮助开发者从业务需求的角度有效的整合阿里云各个大数据产品，大大降低搭建大数据应用系统当中绝大部分的系统工程工作，在相应行业应用解决方案的结合下，能够让不是很熟悉大数据应用系统开发的程序员也能够快速为企业搭建大数据应用，从而实现大数据价值的快速落地。

概括来讲，DTBoost 是以标签中心为基础，建立跨多个云计算资源之上的统一逻辑模型，开发者可以在“标签”这种逻辑模型视图上结合画像分析、规则预警、文本挖掘、个性化推荐、关系网络等多个业务场景的数据服务模块，通过接口的方式进行快速的应用搭建。这种方式的好处一在于屏蔽掉应用开发人员对于下层多个计算存储资源的深入理解与复杂的系统对接工作，二在于通过数据服务的形式透出也有助于 IT 部门对数据使用的管理，避免资源的重复和冗余。

30.2 标签中心

DTBoost 标签中心是作为 DTBoost 产品系列的最基础部分，您需要在标签中心配置好。

可以分为如下三个步骤：

1. 云计算资源注册
2. 实体关系建模
3. 数据聚合同步

如图 *1001:* 主界面所示，我们可以点击主界面 DTBoost 导航栏上的标签中心选项卡进入。

图 1001: 主界面



左侧有如下几个子导航：

- 概览：概览显示了标签中心主要三个功能步骤的状况，已经注册的云计算资源情况、所建立的实体关系模型情况以及数据聚合同步的情况。
- 云计算资源管理：云计算资源管理对应云计算资源注册，把所需要用的的云计算资源授权给 DTBoost 进行管理使用。
- 实体关系标签：管理实体关系标签等内容，新增需要的实体关系资源，并在实体关系上绑定标签信息。
- 模型探索：模型探索对应数据模型的查看与数据聚合同步，能够以可视化的方式查看实体关系模型。
- 标签同步：智能数据同步。

30.2.1 添加云计算资源

背景信息

通过 DTBoost 建立大数据应用前，需要先把相关的云计算资源授权给 DTBoost，以便获取数据的元信息来进行数据模型视图的构建。

目前 DTBoost 支持以下几种云计算资源：

- 阿里云关系型数据库（RDS）
- 阿里云大数据计算（MaxCompute）
- 阿里云分析型数据库（AnalyticDB）
- 阿里云表格存储（Table Store）
- 阿里云数据中枢（DataHub）

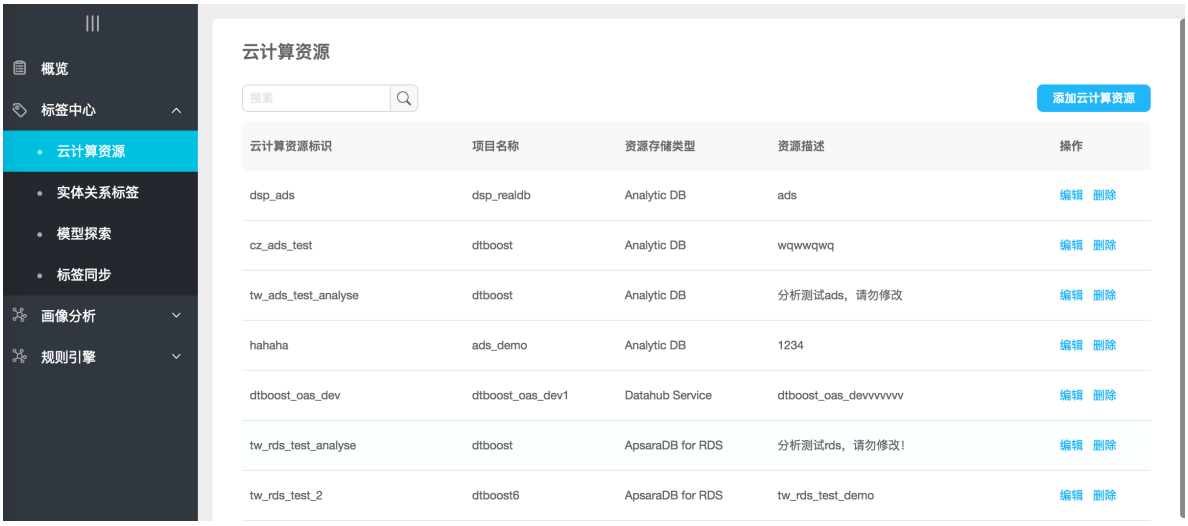
- 阿里云流式计算（StreamCompute）

操作步骤

1. 在**云计算资源管理**界面右上角，单击**添加云计算资源**，添加新的云计算资源。

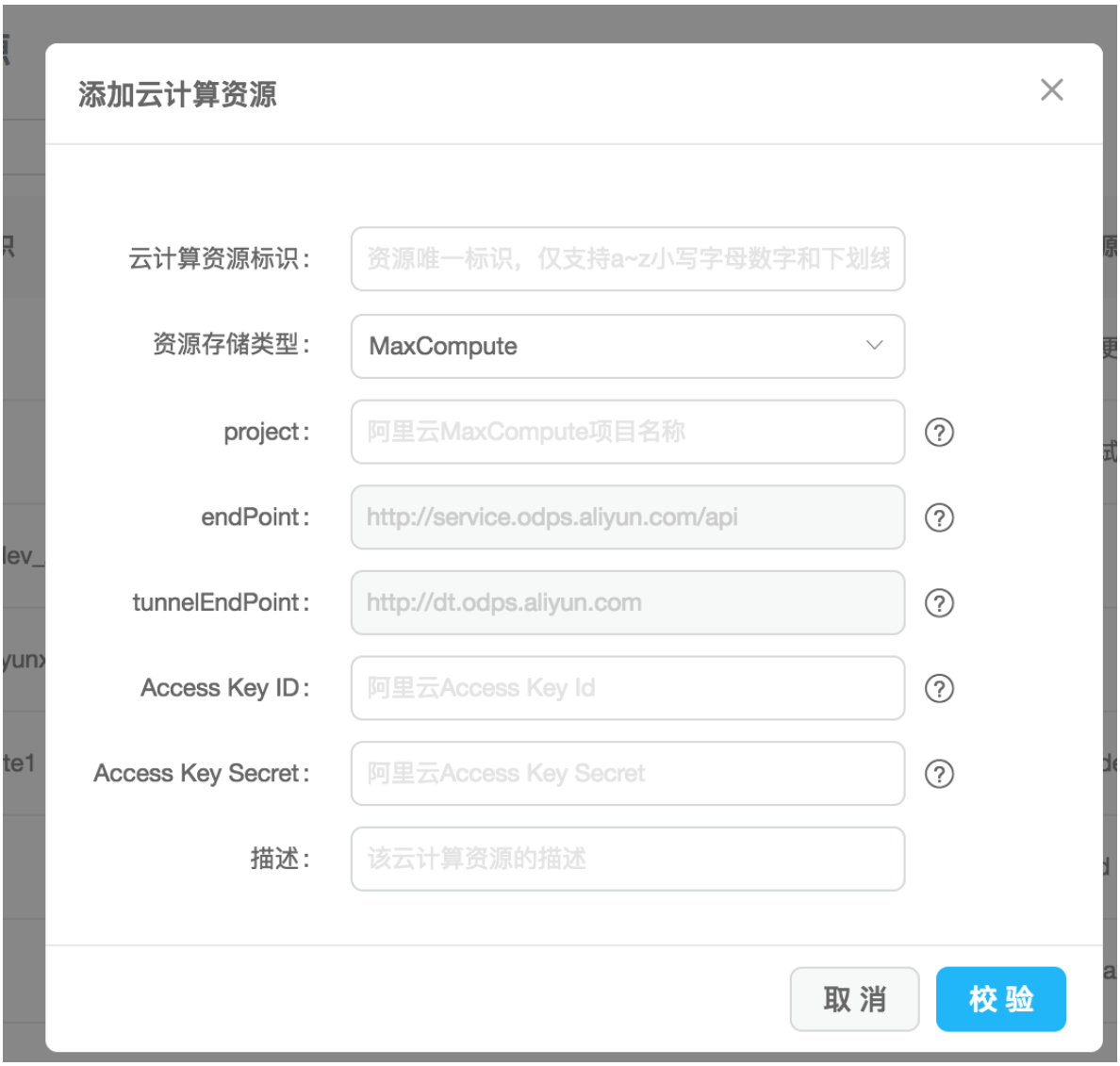
   您也可以在该界面查看您已经注册的云计算资源，如图 *1002:* 添加云计算资源所示。

   图 1002: 添加云计算资源



2. 在**添加云计算资源**面板中，如图 *1003:* 填写相关参数所示，填写以下内容：

   图 1003: 填写相关参数



- 云计算资源标识：在 DTBoost 当中对您添加的云计算资源的唯一命名标识；
- 资源存储类型：选择 DTBoost 所支持的计算存储资源；
- 项目名称/数据库名：选择您所要添加的计算资源的数据库或项目名称；
- endpoint（如有）：仅在选择 **MaxCompute** 的时候填写；
- tunnelEndPoint（如有）：仅在选择 **MaxCompute** 的时候填写；
- Access Key ID：填写对应的登陆密钥 ID；
- Access Key Secret：填写对应的登陆密钥 Secret；
- 描述：可以对该计算资源的用途进行相应描述。

3. 单击**校验**。

   校验成功后即添加云计算资源成功。

30.2.2 实体关系建模

背景信息

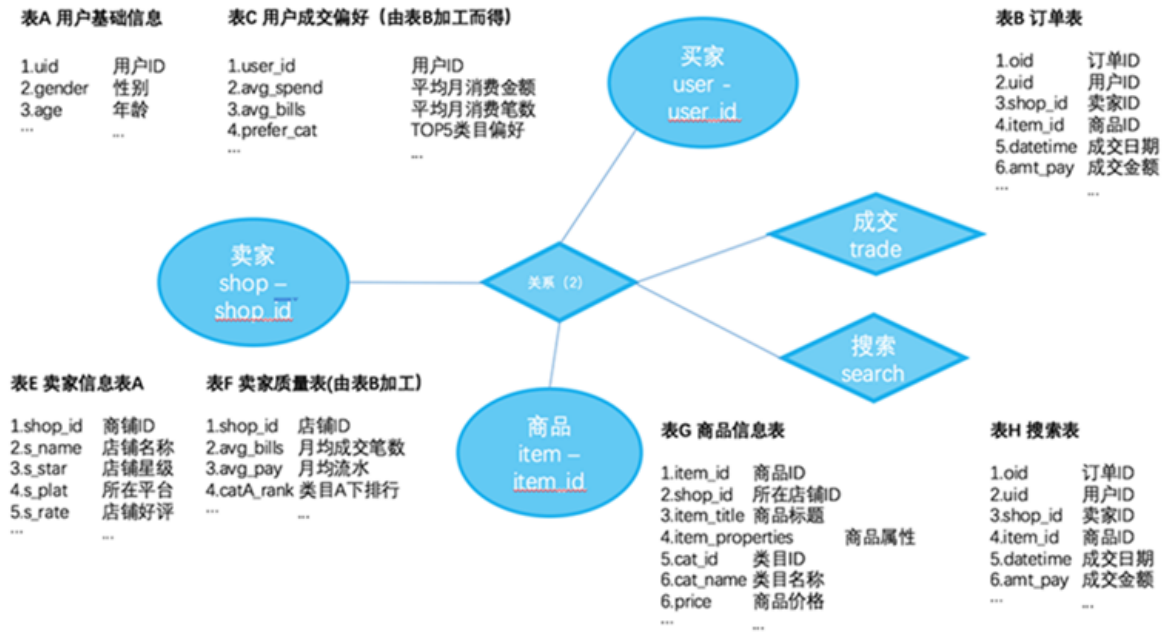
实体关系建模是把分散的多张表，以业务形态进行组织的一种方式，即围绕着一个或多个实体（即描述的对象）的各种属性和行为的描述。整个建模围绕**实体** - **关系** - **标签/属性**三个主要概念展开。

实体关系建模是期望能够把散落在不同存储中的多张数据源表，及其之间的关系能够进行统一管理、建模、数据同步，在这个统一的模型层上进行兴趣提取、整合分析、规则引擎、移动营销各个产品或模块的应用计算中。

实体，即客观世界的一个对象，如人员、车辆、买家、卖家、商品都可以看作是一个实体。从数据表的角度来说，通常带有唯一主键的表则代表一个实体，表中的每一个字段则相当于这个实体的属性，如性别、年龄、收入、月均消费等。

如图 *1004: 实体关系建模示意图*所示，可以把多张表以买家、卖家、商品、成交、搜索这几个实体和关系有机地组织起来，对跨存储的数据表形成一个单一的视图，进行有效的组织管理。在后续的分析与算法处理上，也都是基于这个实体关系模型进行相应的表达，免除对下层各个计算的复杂对接。

图 1004: 实体关系建模示意图



在标签建模当中，实体可以将多张具有同一主键的产生于不同数据库同类表在逻辑层上聚合在一起，每一个字段都是这个实体的一个属性标签，形成一张**大宽表**。

30.2.2.1 配置实体

前提条件

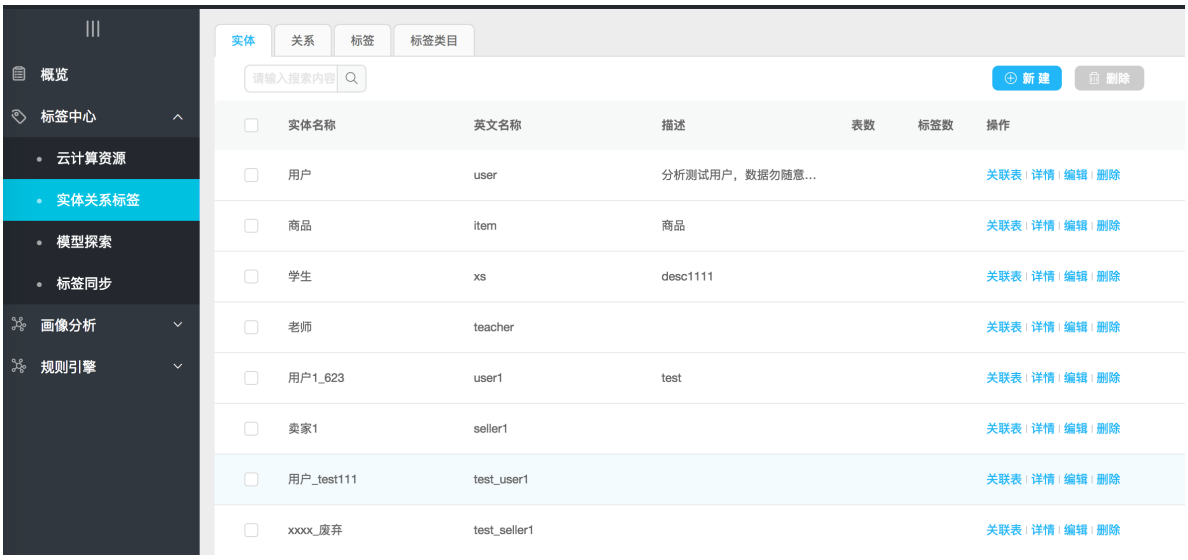
添加实体前，请确认已添加云计算资源，添加云计算资源相关操作请参见添加云计算资源。

操作步骤

1. 单击**标签中心 > 实体关系标签**。

   在**实体关系标签**界面，可以看到已创建的实体关系模型列表。您也可以单击左侧的实体/关系列表，查看实体详情，包括实体的定义和实体下所关联的字段与标签，如实体关系标签界面所示。

   图 1005: 实体关系标签界面



2. 在界面右上角，单击**新建**。
3. 填写实例相关信息后，单击**确定**。

   例如，买家，您可以将其命名（要求唯一），中文名称为**用户**，英文识别名称为**user**，同时给这个逻辑实体命名一个抽象的**主键**，例如：**user_id**，如图 *1006:* 填写实体信息所示。

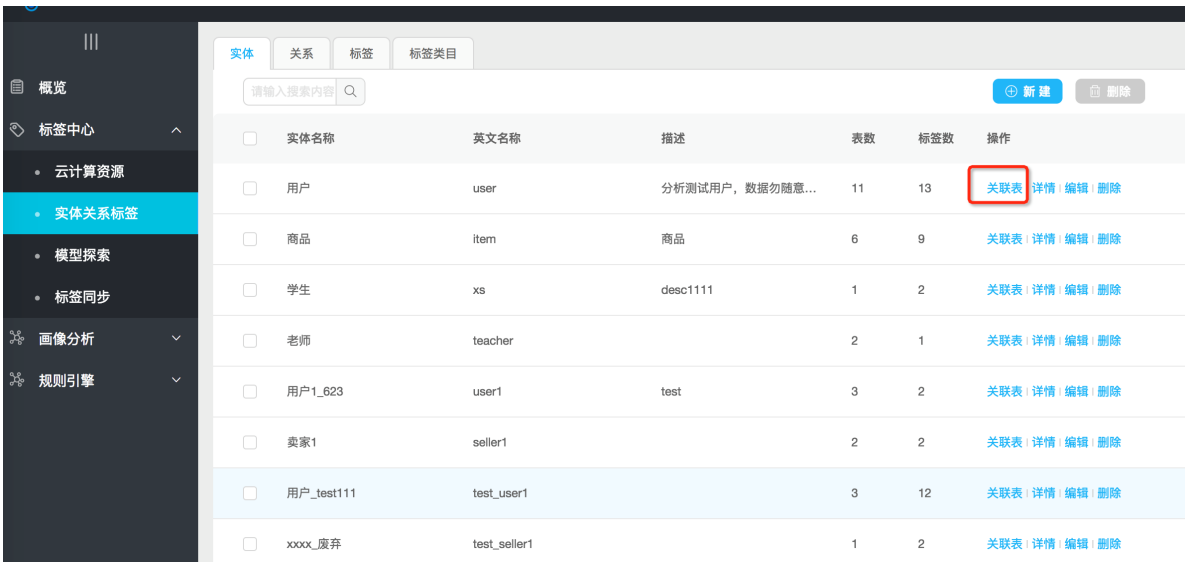
   图 1006: 填写实体信息



添加完成后，您可以将多个描述这个实体的表进行关联。

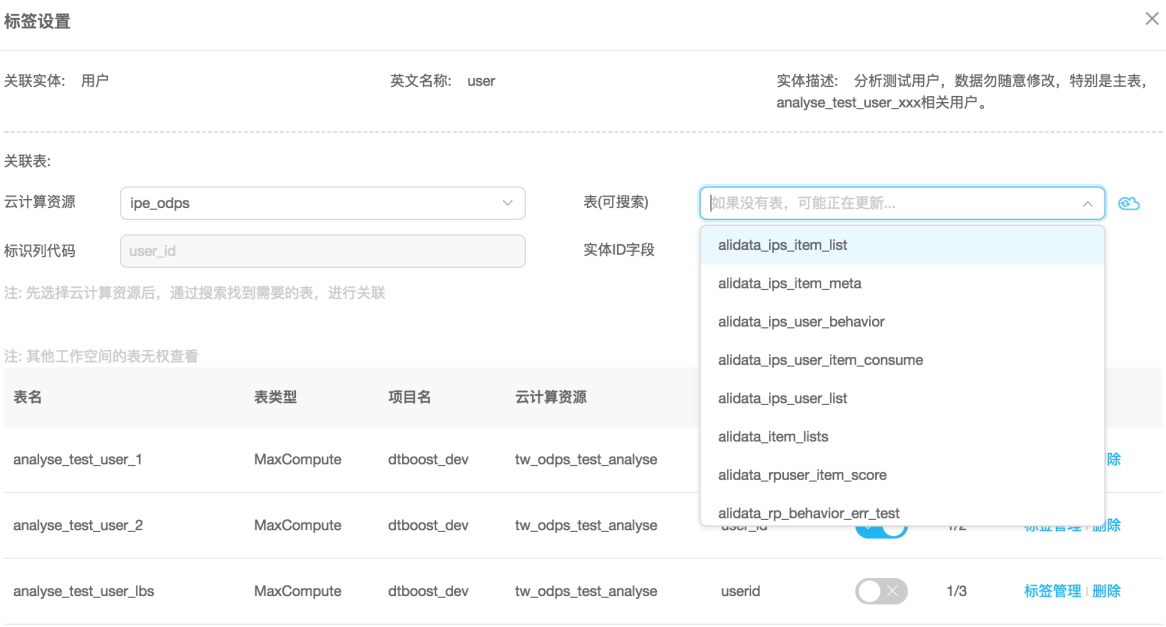
4. 回到**实体**页签页面，单击右侧的**关联表**，如图 *1007:* 新增关联表所示。

图 1007: 新增关联表



5. 在**标签设置**窗口，选择相应数据库下的表的主键，关联到这个实体上，再把相应表中所会用于分析的字段进行一一设置，如图 *1008:* 添加实体关联字段所示。

图 1008: 添加实体关联字段



30.2.2.2 配置标签

背景信息

标签是描述实体/关系的某一个属性，一个字段作为同一个实体/关系下具有唯一性属性描述。

操作步骤

1. 在实体/关系设置页面下方已经关联的表当中，单击**标签数**列下的具体数字链接，如图 *1009:* 标签数所示。

   单击**标签管理**，会对该表设置新的标签。

   图 1009: 标签数



2. 根据需要，设置标签，如图 *1010:* 设置标签所示。

   图 1010: 设置标签



- 英文名称（必填）：标签的标识，在一个租户下不可重复，API 调用时，需要使用英文名；
- 中文名称（必填）：标签中文名，便于查找；
- 标签描述（选填）：标签的详细描述；
- 所在类目（选填）：当实体下的标签太多时，可通过建立类目，对标签进行分类浏览查看；
- 为空条件（选填）：指定标签数据为空时的字符，便于使用时，排除为空时的数据；
- 值类型（选填）：标签的数据分为枚举、多值和数值。

  枚举和多值需要通过指定标签的具体取值，如性别的标签需要明确取值，如{0:女；1:男}。标签的值域类型有以下几项。该标签的为空条件、标签类型和值字典。为空条件输入一个或多个作为空值、未知值的数值；标签类型有枚举、数值、多值。值字典则是指定该属性的字典，您可以通过人工录入的方式；

  说明: 多值是分析型数据库的一种特殊类型，是以多个个数不定的 Key 组成，通常一些人工标注的标签属性、选择个数不定的多选题都可以用多值列的方式来存储。
- 关联实体（选填）：或者是建立一个实体，把该标签关联到相应的实体上，并标注值字典所需要匹配返回的属性字段。 标签的取值可以通过某个实体下的标签数据进行获取，也可以跳过此项，在值来源中手动输入；

- 值来源（选填）：当标签的取值不多时，可通过手动的方式录入；如果标签取值量非常大，比如省份、城市、品牌这样的标签可以通过关联其他标签的方式来获取；
- 值类型（必填）：常用的数据值类型，根据标签具体的数据情况选择即可。系统会根据相应表的字段来匹配一个字段类型，如 BIGINT/DATETIME/STRING。

3. 单击**绑定**。

30.2.2.3 配置实体关系

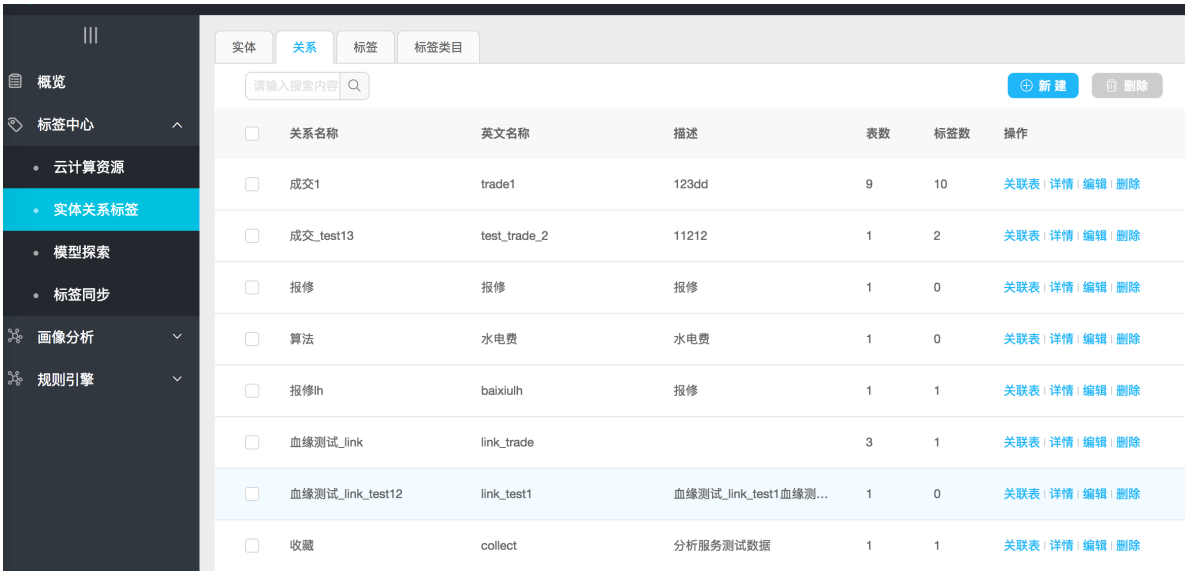
背景信息

关系，是实体与实体之间所发生的连接，通常表示某一种行为/一个事实，如成交、搜索、出行。从数据表的角度来看，这样的表通常被称为**事实表**，往往是有多个联合主键（或是说都是外键），如成交表示的是**买家** - **卖家** - **商品**之间的关系，往往没有唯一的主键（或是说唯一主键，如订单 ID 在分析场景中不发挥主要作用）。

操作步骤

1. 在**实体关系标签**界面，单击**关系**选项卡，如图 *1011: 关系选项卡*所示。

图 1011: 关系选项卡



2. 在**关系**选项卡右上角，单击**新建**。

显示**添加关系**对话框。

3. 为新建关系指定名称，以及关系是连接到哪几个实体，单击**确定**，如图 *1012: 添加关系*所示。

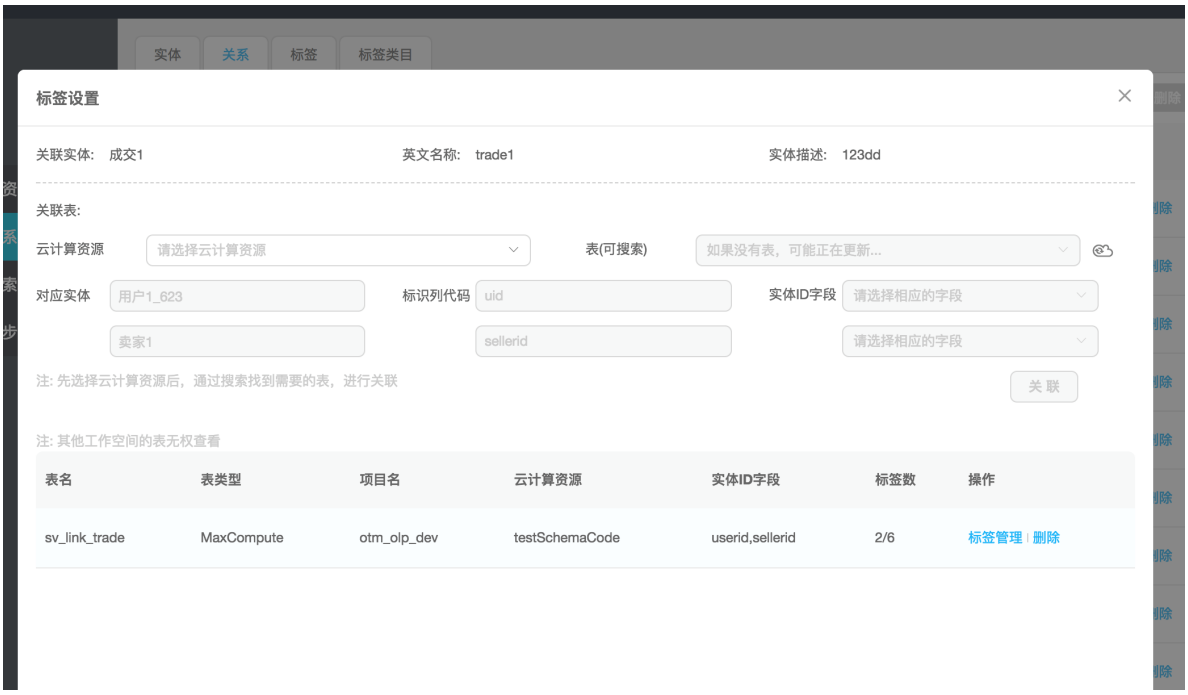
图 1012: 添加关系



4. 在**关系的实体**区域框中，单击**更换表**。
5. 选择该关系所关联的数据表，并把几个实体所对应的外键进行指定，如图 *1013: 关系表关联设置*所示。

   关系上也可以挂接属性，如成交的时间、金额、次数等，下一步您可以将指定为关系的表中的其他字段配置位标签。标签的相关配置与实体的相关配置当中一致。

图 1013: 关系表关联设置



在实体和关系设置完成之后，即可以在模型探索当中查看所构建的实体关系模型，并进行数据同步。

30.2.3 数据整合与同步

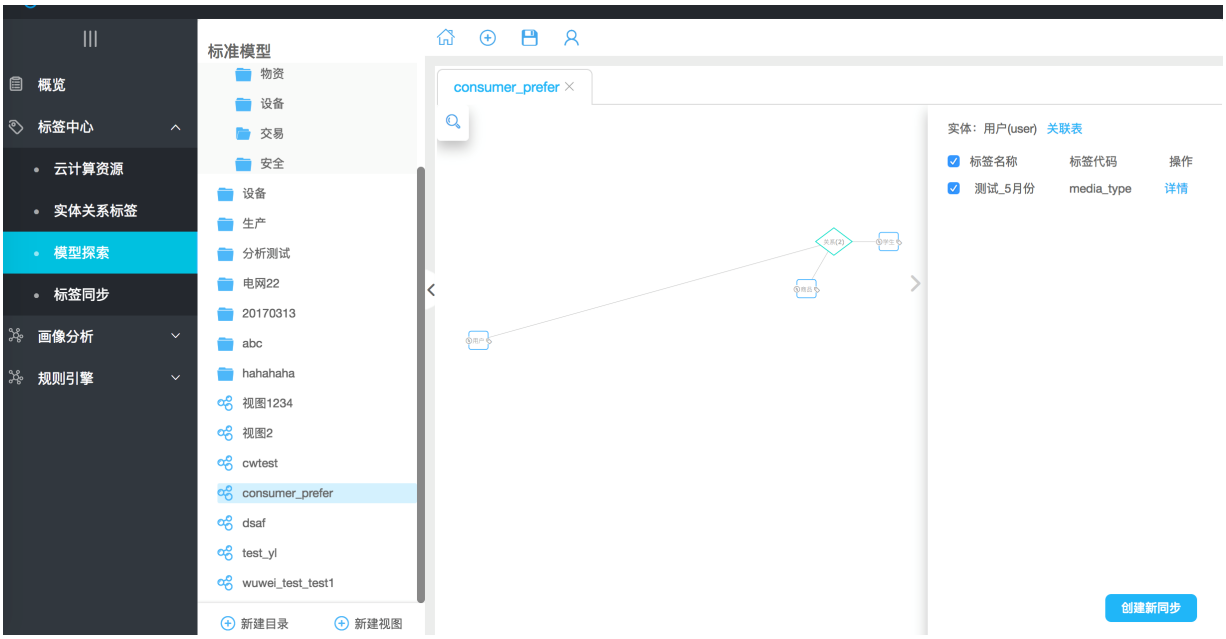
背景信息

配置好相应的模型后，您便可以进入模型探索，搜索相应的实体后拖入到画布当中，就可以看到这个实体所连接的所有一度关系，还有哪些实体通过什么关系与这个实体相连接。此时您就可以开始进行数据的整合，将多个来源的数据根据相应的场景，进行数据的同步聚合，为分析做好数据准备。

为了能够把分散在多个数据库当中的表进行关联分析，那就需要把这些表能够聚合到一个数据库当中。

整合分析内置了几个常规的数据同步对于您需要进行整合分析的实体。如图 *1014: 模型探索界面*所示，方块表示实体，菱形和圆代表关系。

图 1014: 模型探索界面



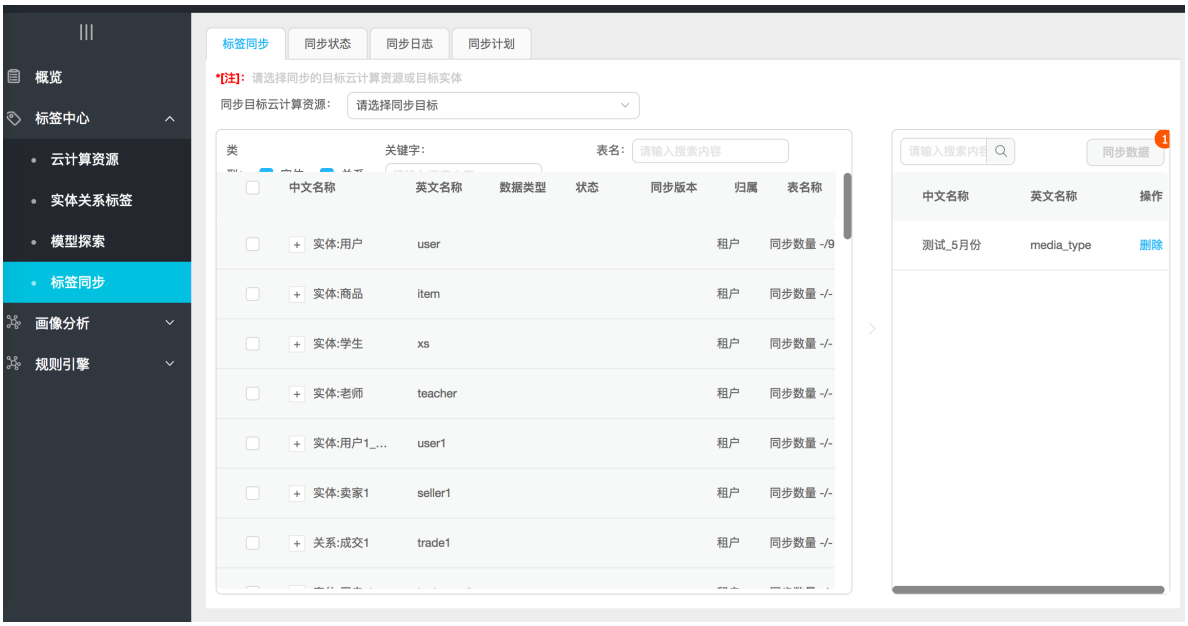
操作步骤

1. 在**模型探索**界面，单击某一个实体/关系，勾选相应的属性，单击**创建新同步**。

   显示**创建同步**对话框。

2. 确认待同步的标签，如图 *1015: 确认待同步的标签*所示。

   图 1015: 确认待同步的标签



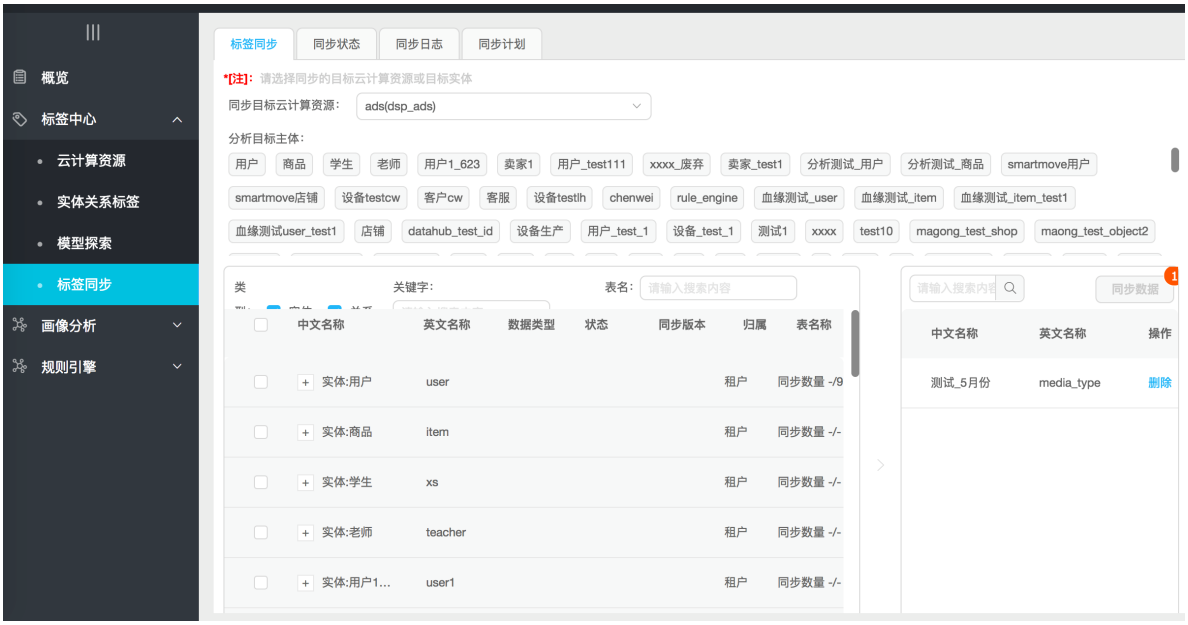
3. 选择同步目标，即选择您需要把数据进行集中同步的存储计算资源，如图 *1016: 选择同步目标*所示。

如果您选择的是阿里云分析型数据库，还需要您指定您所需要分析的分析主体是哪一个，便于我们帮助您对分析型数据库的表组策略进行优化。

目前整合分析支持的数据目标库仅支持阿里云分析型数据库、关系型数据库 -MYSQL 以及阿里云大数据计算服务（MaxCompute）。

通常来说，此处指定的目标是作为数据聚合之后的在线分析库，故通常选择分析型数据库或关系型数据库作为数据同步目标。

图 1016: 选择同步目标



4. 单击**下一步**。
5. 选择同步方式，如图 *1017:* 选择同步方式所示。

   需要设置如下内容：

   - 数据同步的生命周期

     如果您设置了生命周期，对于有多分区的表只会进行增量同步，对于有分区的表会保留相应的分区数量。
   - 数据同步的模式

     在此您还可以选择同步的方式，对于同步多张小数据量的表您可以打开实时插入同步开关以获得更快的写入速度，对于大数据量的表则还是推荐关闭此开关使用批量建表的模式进行以保证更稳定。
   - 数据同步的主键

此处填写/修改数据同步的主键是为了确保数据在写入时候避免因为主键设置不完整，而导致记录被覆盖的问题。

图 1017: 选择同步方式



6. 完成后，单击**发起同步**。

30.3 整合分析

您可以通过整合分析服务为标签中心所管理的数据灵活搭建交互式数据分析应用，能够对筛选出来的特定的分析对象进行多维透视，并进一步钻取分析，并可以将分析筛选出来的对象导出到其他系统当中，如结合广告投放系统进行精准营销。

整个界面的代码是完全开放的，可以无缝与您的现有系统进行整合。或者您可以在产品当中自助封装 API 接口，供您更便捷的搭建您的多个分析应用。

整合分析在功能上主要有两部分：

- 第一部分提供了接口调试与自助封装的工具，能够帮助您快速封装支持动态传参动态计算的分析接口；
- 第二部分则提供了界面化配置的交互式分析应用搭建工具，能够生成独立部署的代码包。

30.3.1 接口调试

30.3.1.1 接口架构

从系统上来看是建立在实体关系模型之上的分析查询系统，Web 应用开发者直接通过与 API 的交互，结合可视化引擎或是其他图表组件，即可以快速搭建自己的分析型数据产品。

整合分析是架设在应用与 DB 之间的逻辑路由层。上层应用通过 TQL（Tag Query Language）或者 Expr（Analysis Expression）向整合分析查询数据。整合分析解析 TQL（或Expr）并基于不同的数据源构造查询。

整合分析的总体架构如图 *1018: 总体架构*所示。

图 1018: 总体架构

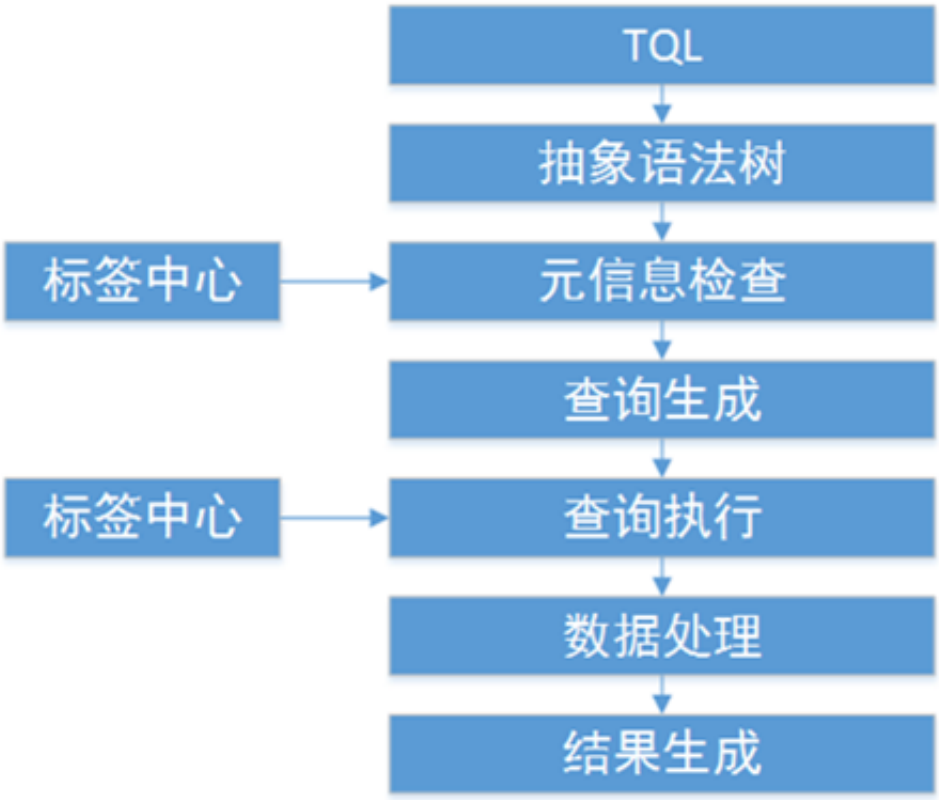


整合分析各模块的主要功能如下：

- TQL/Expr 解析器：解析 TQL/Expr，生成抽象语法树。
- 查询生成器：根据标签元信息和抽象语法树，生成查询。
- 查询执行器：执行查询。
- 数据预处理：对查询返回的结果数据，进一步处理（比如，计算百分比等）。
- 结果生成器：将结果数据的列替换为对应的标签或者维度。

整合分析的查询执行流程如图 *1019: 查询执行流程*所示。

图 1019: 查询执行流程



30.3.1.2 调试界面

分析服务接口模块可以让您在此进行分析语句的调试和自助化封装数据分析接口。整合分析的查询表达都是建构在实体关系模型之上的。

图 1020: 调试界面



如图 *1020: 调试界面*所示，在界面的左侧，您可以在分析接口的列表和数据标签元信息查询列表之间通过选项卡进行切换。

右侧上方您可以指定您当前需要分析的数据在哪个数据库上，以及以哪个实体作为主要的分析实体。如果您没有特殊的要求的话，您可以无需指定，分析服务会帮您做相应的路由判断。

下方的输入可以选择分析查询的类型和例子，可以通过选择简单的例子来查看使用方法。

再下方的输入框就是分析语句的编辑器。

您可以单击**执行**，查看查询的结果/执行错误，以及语法每一步解析所耗费的时间，所解析的真实SQL语句，来帮助您调试分析接口，如图 *1021: 调试结果*所示。

图 1021: 调试结果



30.3.1.3 分析语法表达

TQL（Tag **Query** Language）

类似于 SQL 的语法。简单来说，就是把每一个实体和关系当作一个视图表。

举例如下：

1. 查询某个用户的性别：

```
//sql
select user.gender from user where user.uid = xxx;
```

2. 计算所有类目下的成交笔数：

```
//sql
select trade.cateid,count(*) as cnt from trade group by trade.cateid;
```

Expr

表达式的语法更贴合实体关系模型的语义逻辑。

举例如下：

```
Target: user(sex="male")->trade(date>20160105)->shop(star>5)
Return: sum(trade.pay) as pay, user.sex as sex
```

其中，Target 部分，从语义上来看是声明分析的对象，如 `user(sex="male")->trade(date>20160105)->shop(star>5)`，是在表达对在20160105日之后，成交过店铺星级大于**5**的男性。从数据的角度来说，则是把 user、trade、shop join 关联起来。

Return 部分，即是声明返回的字段/属性，如 sum(trade.pay) as pay, user.sex as sex，是在表达，按照性别作为维度汇总成交的总金额，其中非带有聚合函数部分的字段都会作为 group by 的字段列。这基本就组成了 Expr 最基本的表达元素。

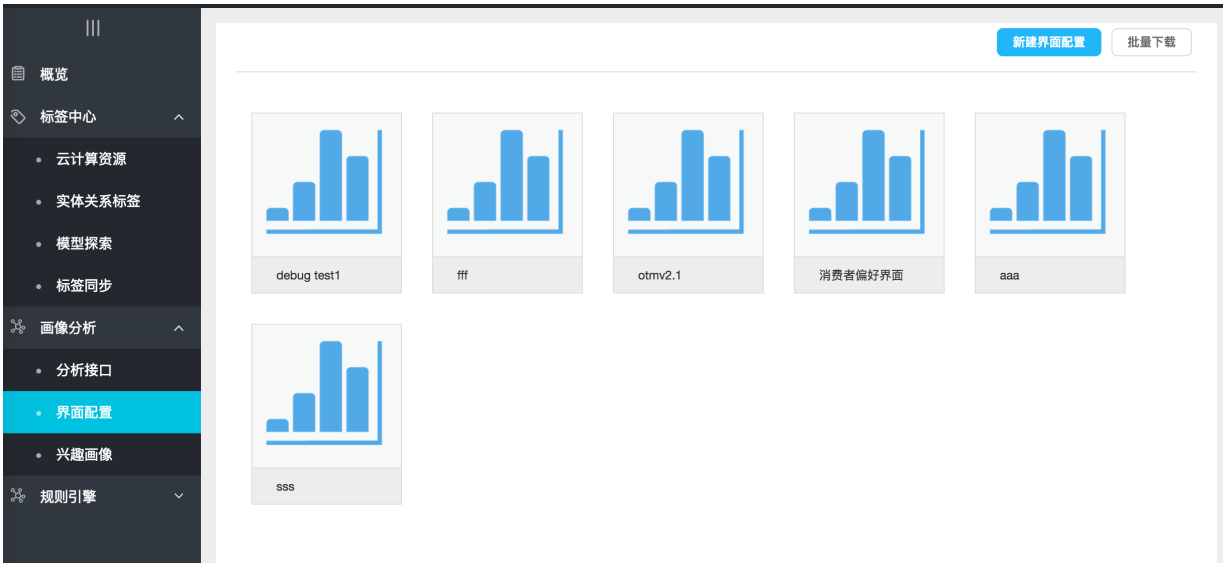
详细的语法明细请参见《阿里云_大数据应用加速器_*DTBoost_API*开发指南》。

30.3.2 界面配置

背景信息

整合分析的界面配置可以通过可视化的方式配置出如图 *1022: 界面配置*所示的整合分析应用，并且能够下载界面的源代码在您的服务器部署、修改样式布局等。

图 1022: 界面配置



- 查看应用列表

  选择**画像分析** > **界面配置**，进入**界面配置**，可以查看已经创建的应用列表。

- 批量下载应用

  在**界面配置**界面，单击**批量下载**后，选中多个应用。

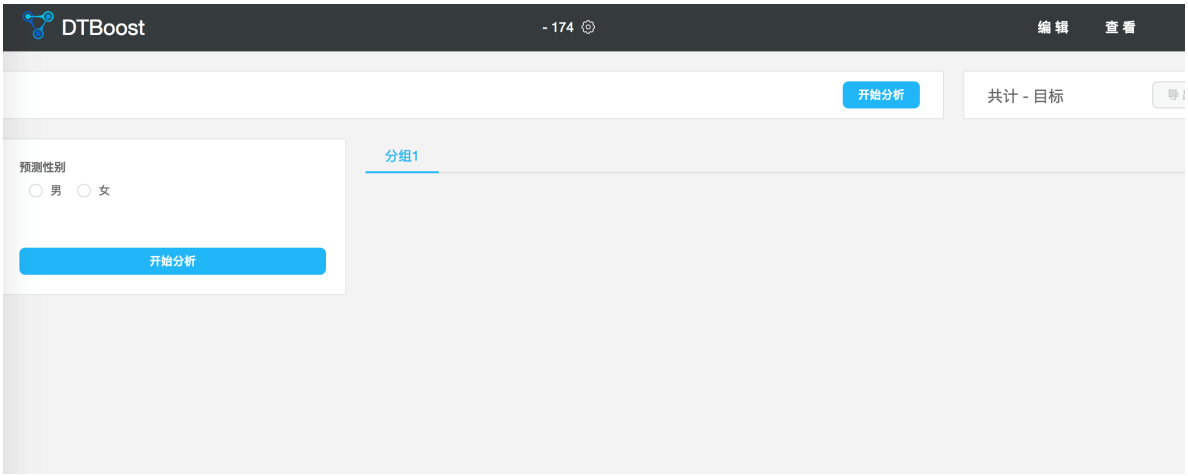
  在**界面配置**界面，单击左上角出现的**下载**，可以将多个应用打包下载。

  具体的部署方法请参见下载包中的说明手册。

- 分析应用

在**界面配置**界面，单击**新建界面配置**，为应用命名后即进入如图 *1023: 准备分析应用*所示界面，可以开始准备配置分析应用。

图 1023: 准备分析应用



30.3.2.1 功能介绍

如图 *1024: 分析应用*所示，为通过配置所得到的最终分析应用。

图 1024: 分析应用



具有如下功能：

- 左侧是筛选器区域，支持日期、单选、多选、数值、范围、文本框等多种形式。您可以通过脚本的编写配置一些比较复杂的筛选条件。
- 右侧是图表展示区域，目前支持基础图表、地图、表格三大类，您可以点击图表上的元素以增加这一项作为总体分析的筛选条件，进行钻取分析。
- 左上方是目前已经选中的筛选条件，您可以点击某个筛选条件上的 **x** 以去掉某个条件。
- 右上方是对分析对象的个数去重统计，并可以根据预设的导出设置，把筛选出的对象连同指定的属性字段导出到 OSS 或其他存储当中。

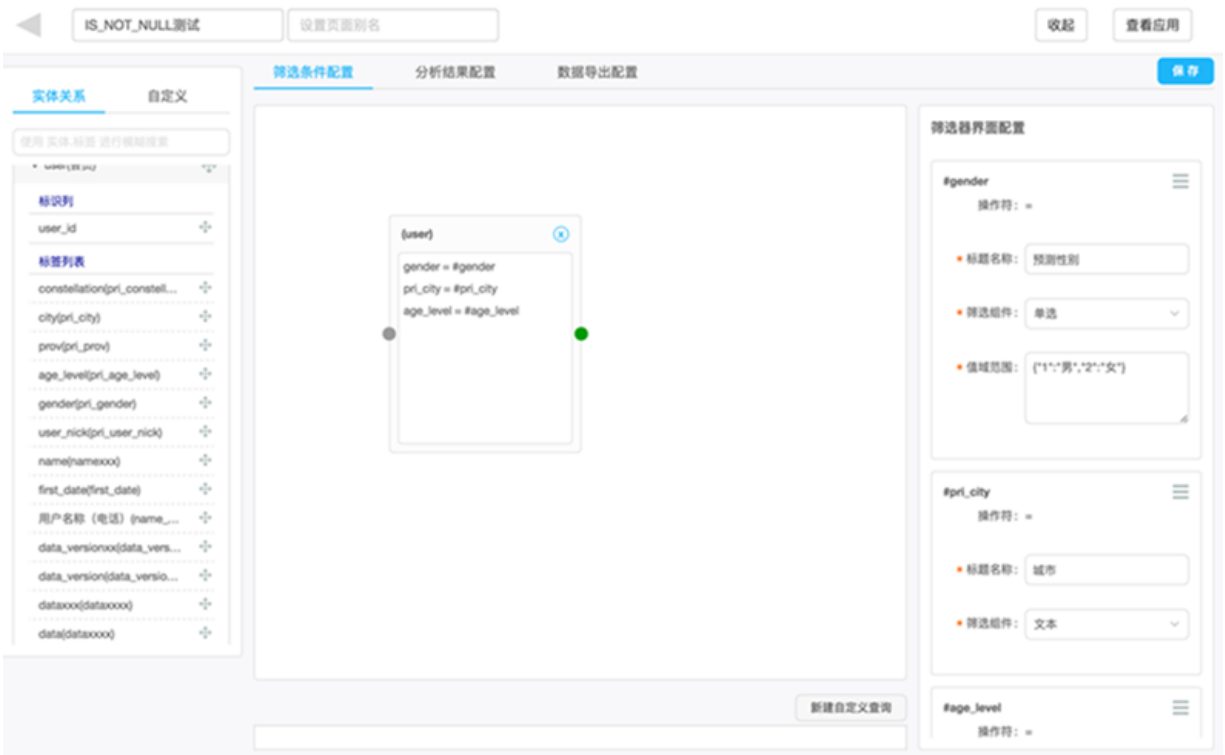
30.3.2.2 配置说明

30.3.2.2.1 筛选条件配置

1. 在**画像分析** > **界面配置**页面，单击某个界面配置。
2. 在显示的界面配置右上角，单击**编辑**按钮，默认进入筛选条件配置页签页面。

如图 *1025: 筛选条件配置*所示，筛选条件配置对应着最终整合分析应用界面的筛选器区域，从 SQL 的解析来讲，是对应 where，from， join 部分的主要查询逻辑。

图 1025: 筛选条件配置



筛选条件配置界面主要包含以下几部分：

- 筛选逻辑

  中间部分是主查询逻辑的配置区域，方框代表一个实体/关系或者子查询，方框和方框之间的连线代表 join 关联，方框中的每一行代表一个筛选条件，各个筛选条件之间是 and 关系。您可以设置每一个属性筛选的操作符（大于、等于、小于等），您可以设置固定的筛选值，或者是通过#来表示变量。相应的变量会在右侧的筛选器配置区域出现。您也可以对每个属性作一些函数变换。

- 实体关系列表

  左侧是实体、关系属性的列表，可以通过拖拽的方式把相关的实体/关系/属性拖动到中间查询逻辑的画布区域。

- 筛选器配置

  右侧是筛选器配置区域，画布当中的变量会在此处显示，您可以对筛选器的名称，筛选器类型以及筛选器的值域范围进行相应的配置。

- 自定义查询

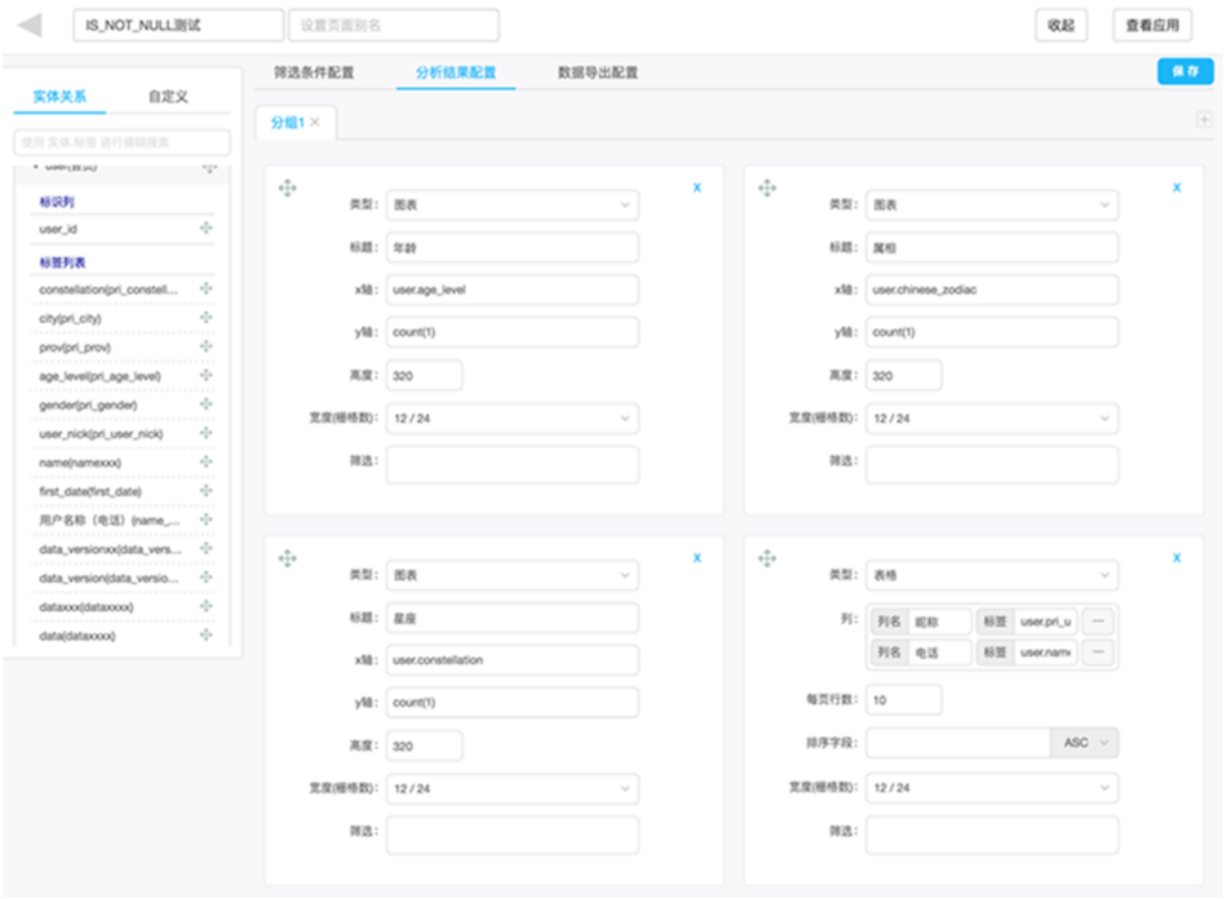
  下方您可以进行自定义子查询，自定义的查询也可以作为一个结点拖入到主查询逻辑当中作为子查询存在，方便您配置一些比较复杂的分析逻辑。

30.3.2.2.2 分析结果配置

1. 在**画像分析** > **界面配置**页面，单击某个界面配置。
2. 在显示的界面配置右上角，单击**编辑**按钮，默认进入**筛选条件配置**页签页面。
3. 点击**分析结果配置**页签，进入**分析结果配置**页面。

如图 *1026: 分析结果配置*所示，分析结果配置对应着整合分析界面的图表展示区域，从 SQL 解析来讲，是对应 select 后面的字段。

图 1026: 分析结果配置



在右侧区域，您可以设置所显示分析结果图表的相关配置：

- 图表分组

  右侧上方的选项卡区域是对应到分析界面图表区的分析结果分组，您可以通过双击选项卡，更改分组名称。

  可以点击选项卡右侧的+来增加新的图表分组。

  在选项卡下可以单击**新增图表**，在该分组下增加新的分析图表。

- 图表类型

  目前图表有四种类型常规图表、多系列图表、地图、表格。

- 输入数据

  对于两个常规图表类型的组件，分别需要输入 x 轴（维度）、y 轴（指标）和系列（复合维度），您可以通过对左侧的列表中的标签拖拽至相应的输入框内，您也可以手工输入。指标计算的方法您可以自由定义，语法同 SQL。

  对于地图，您需要分别指定代表经纬度的标签，以及被用于数值统计的标签。对于表格，您需要单击+来添加表格列，为每一列命名并指定每一列的标签字段。

其中的筛选和排序同 SQ L当中的 where 和 order by。其中，筛选与筛选器不同的是，筛选器的筛选条件是全局过滤条件；而该筛选条件只对本图表起作用。

- 图表宽高

  您可以自由配置图表的高度（按像素计算）和宽度（以按每行的比例计算）。

30.3.2.2.3 数据导出配置

1. 在**画像分析** > **界面配置**页面，单击某个界面配置。
2. 在显示的界面配置右上角，单击**编辑**按钮，默认进入**筛选条件配置**页签页面。
3. 点击**数据导出配置**页签，进入**数据导出配置**页面。

如图 *1027: 数据导出配置*所示，数据导出配置对应的是分析面板右上角的个案数统计和数据导出的配置。

图 1027: 数据导出配置



您可以对筛选后的数据条数进行计数统计，并把这些筛选出的个案可以进行导出，便于您把筛选出来的数据可以和下游系统进行对接。

相关配置说明如下：

- 计数对象

计数对象在分析界面对应着您所统计的目标对象的个案条数（count distinct），您需要在此选择配置您所计数的对象是哪个实体，以及在界面上显示的计数单位是什么。

- 导出源/目标

  您可以将筛选出来的对象数据进行导出，在此需要选择您默认导出的数据源和导出目标所在的云计算资源的位置。

- 导出字段

  除了导出计数对象的标识列/主键以外，您还可以选择其他的返回字段一并导出。所选择返回的标签字段需要从属于计数对象实体之下。

30.3.3 分析应用部署

具体请参见所下载分析应用压缩包中的帮助文件。

30.4 规则引擎

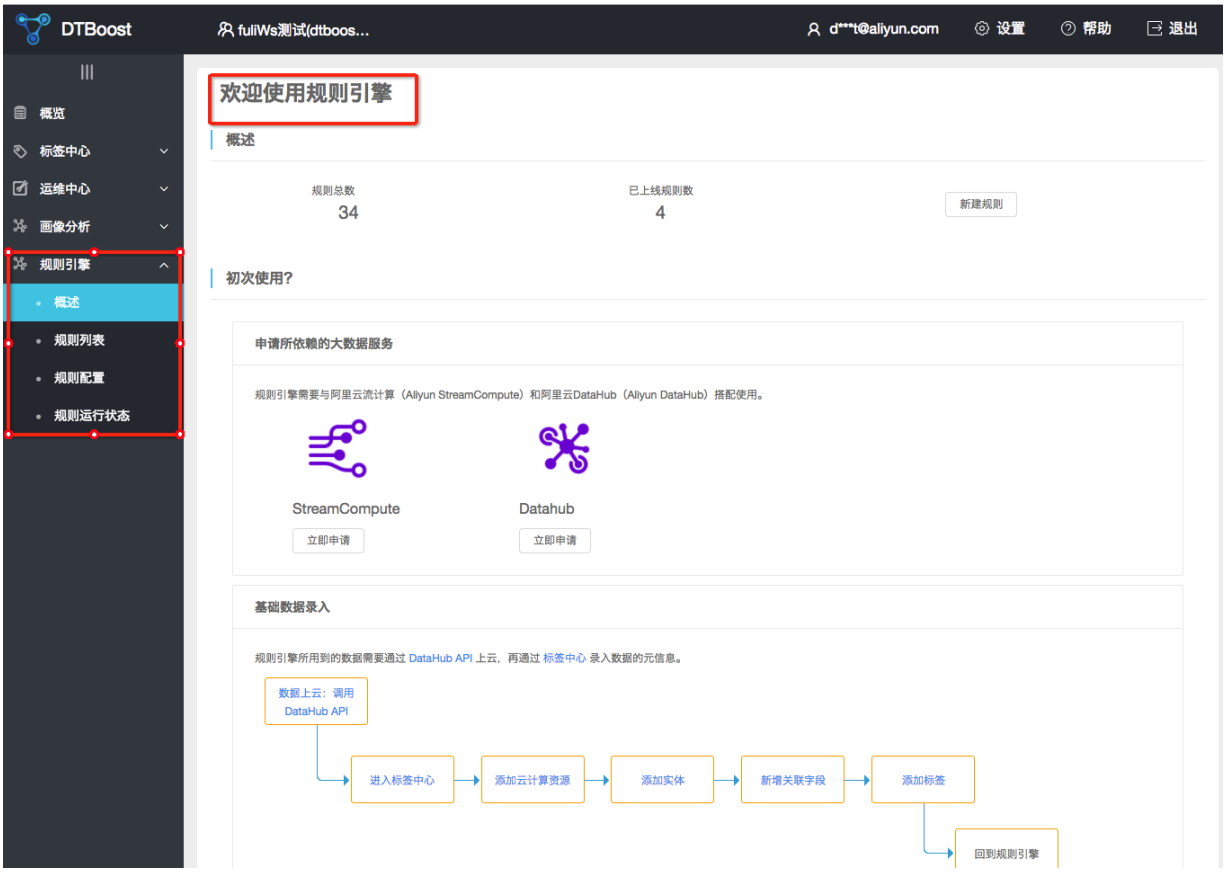
基于流计算的规则引擎适用于业务决策的实时评估，包括风险拦截和预警、资源分配、流程改进等。特别适用于状态或行为数据频繁变化的场景下，针对特定的业务目标进行实时决策。

使用者以图形化操作形式，结合专家业务经验配置自定义规则，规则引擎实时处理数据，当自定义规则命中时，实时返回结果。

规则引擎欢迎界面如图 *1028:* 规则引擎欢迎界面所示，左侧有如下几个导航:

- 概览：规则引擎的总体概览。
- 规则列表：主要是用户创建的规则的列表信息。
- 规则配置：主要是用户用户创建和配置规则信息。
- 规则运行状态：主要是用于查看规则的实时运行状态和结果。

图 1028: 规则引擎欢迎界面



30.4.1 概览

申请所依赖的大数据服务

初次使用时，如果没有阿里云流计算（Aliyun StreamCompute）和阿里云 DataHub（Aliyun DataHub）计算资源，需要申请所依赖的大数据服务，如图 *1029: 依赖的大数据服务*所示。

规则引擎需要与阿里云流计算（Aliyun StreamCompute）和阿里云 DataHub（Aliyun DataHub）搭配使用。

规则引擎运行的流式任务会提交到阿里云流计算（Aliyun StreamCompute）上面运行，用户的基础流式数据使用阿里云 DataHub（Aliyun DataHub）存储。

图 1029: 依赖的大数据服务



基础的数据录入

规则引擎所用到的数据需要通过 **DataHub API** 上云，再通过**标签中心**录入数据的元信息，如图 *1030:* 基础数据录入所示。

图 1030: 基础数据录入



创建规则流程图

使用规则引擎创建规则的流程图如图 *1031:* 使用规则引擎所示。

图 1031: 使用规则引擎



30.4.2 规则基本操作

30.4.2.1 创建规则

背景信息

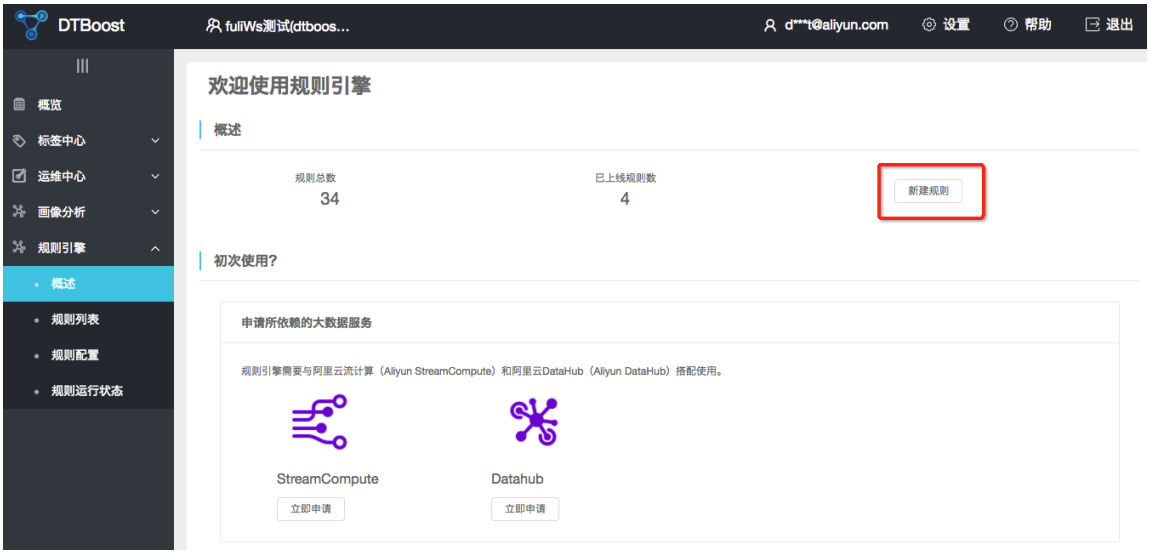
您可以通过以下三种方式创建规则。

操作步骤

1. 进入 DTBoost 欢迎界面。

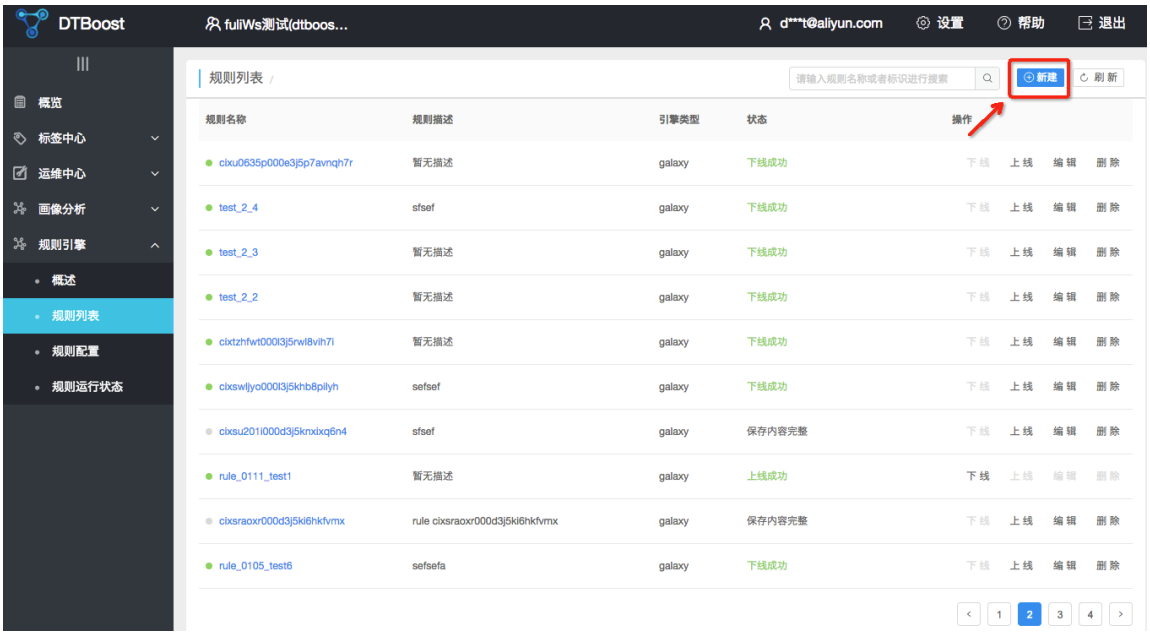
   a) 在欢迎界面，找到**概述**，单击**新建规则**创建规则，如图 *1032: 新建规则1*所示。

   图 1032: 新建规则1



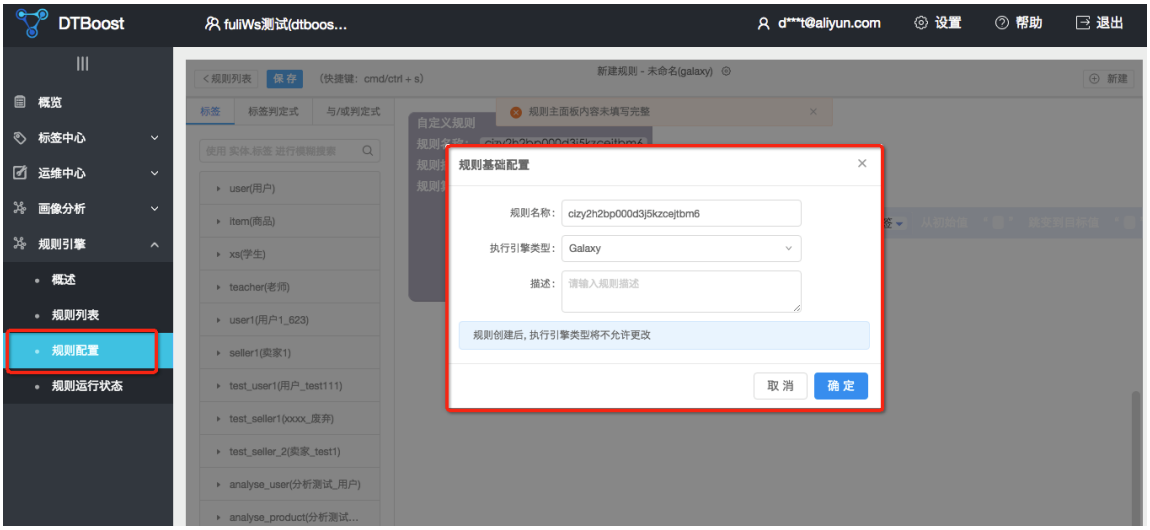
   b) 找到**规则列表**，单击右上角的**新建**进行规则的创建，如图 *1033: 新建规则2*所示。

   图 1033: 新建规则2



c）找到**规则配置**进行规则的创建，如图 *1034: 新建规则3*所示。

图 1034: 新建规则3



2. 填入规则的基础配置信息，如图 *1035: 规则基础配置*所示。

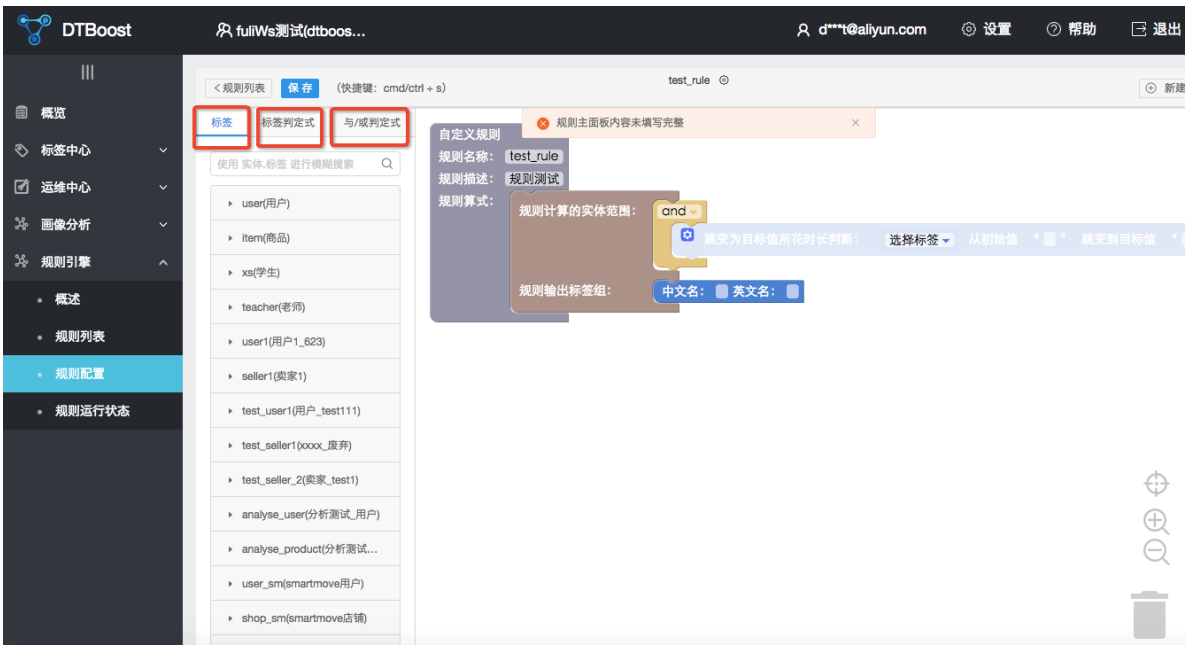
图 1035: 规则基础配置



说明: 规则创建后，执行引擎类型将不允许更改。

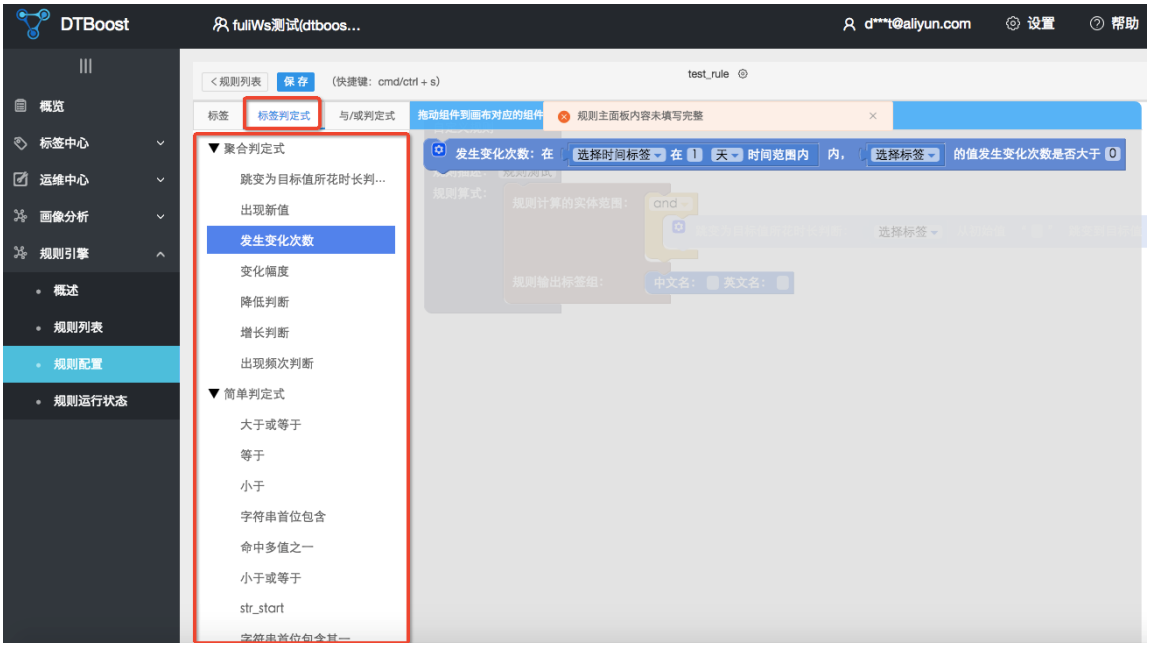
3. 创建完成之后，进行规则的配置。自定义规则的左侧栏目分别是标签、标签判定式、与/或判定式，如图 *1036:* 规则标签和判定式配置所示。

图 1036: 规则标签和判定式配置



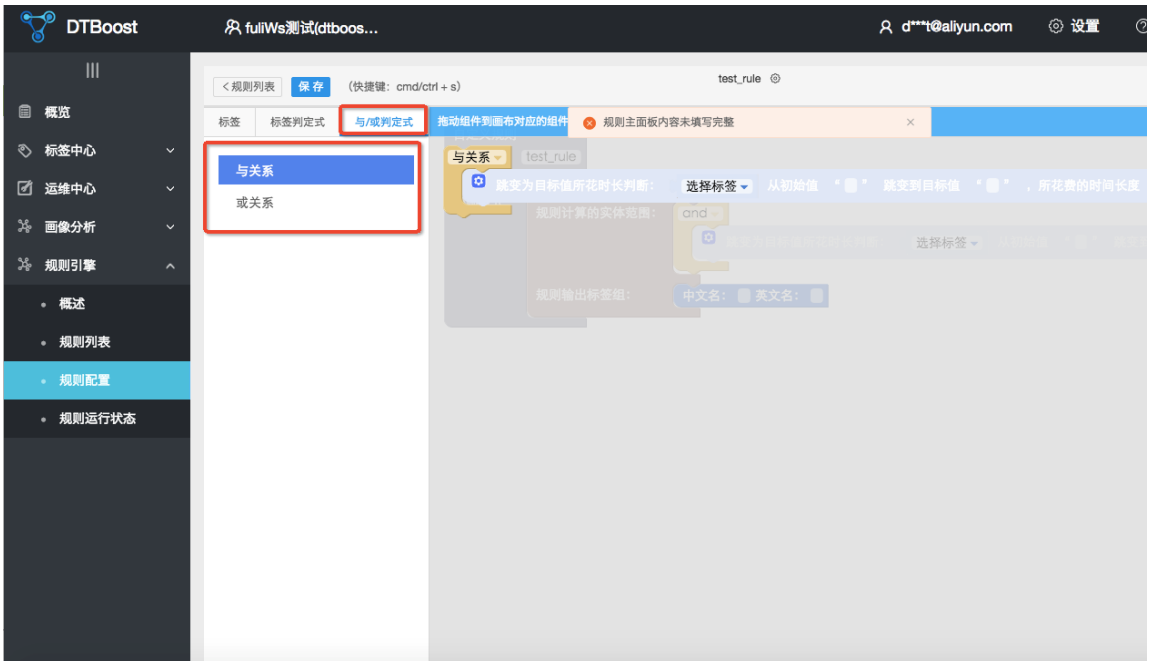
- 目前支持的标签判断式如图 *1037: 标签判断式*所示，支持**跳变为目标值所花时长判断**、**发生变化次数**等规则。

图 1037: 标签判断式



- 目前支持的与/或判断式如图 *1038: 与/或判断式*所示，支持**与关系**、**或关系**判定式。

图 1038: 与/或判断式

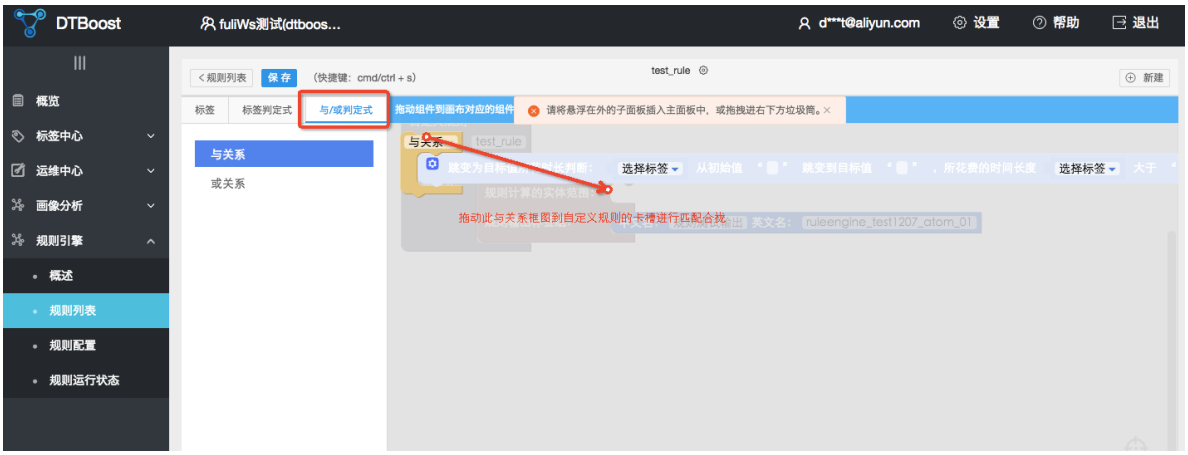


30.4.2.2 配置规则

操作步骤

1. 单击**规则列表**，选择**与或判定式**标签页，然后选择一个 and 和 or 判定式到页面上，如图 *1039: 配置与/或判定式*所示。

图 1039: 配置与/或判定式



2. 将其拖动到自定义界面的卡槽里，拖动完会默认弹出弹窗界面，如图 *1040: 标签判定式配置*所示。我们可以先不选择具体配置，单击**取消**，稍后通过在左侧栏的标签栏进行选择实体和标签。

图 1040: 标签判定式配置



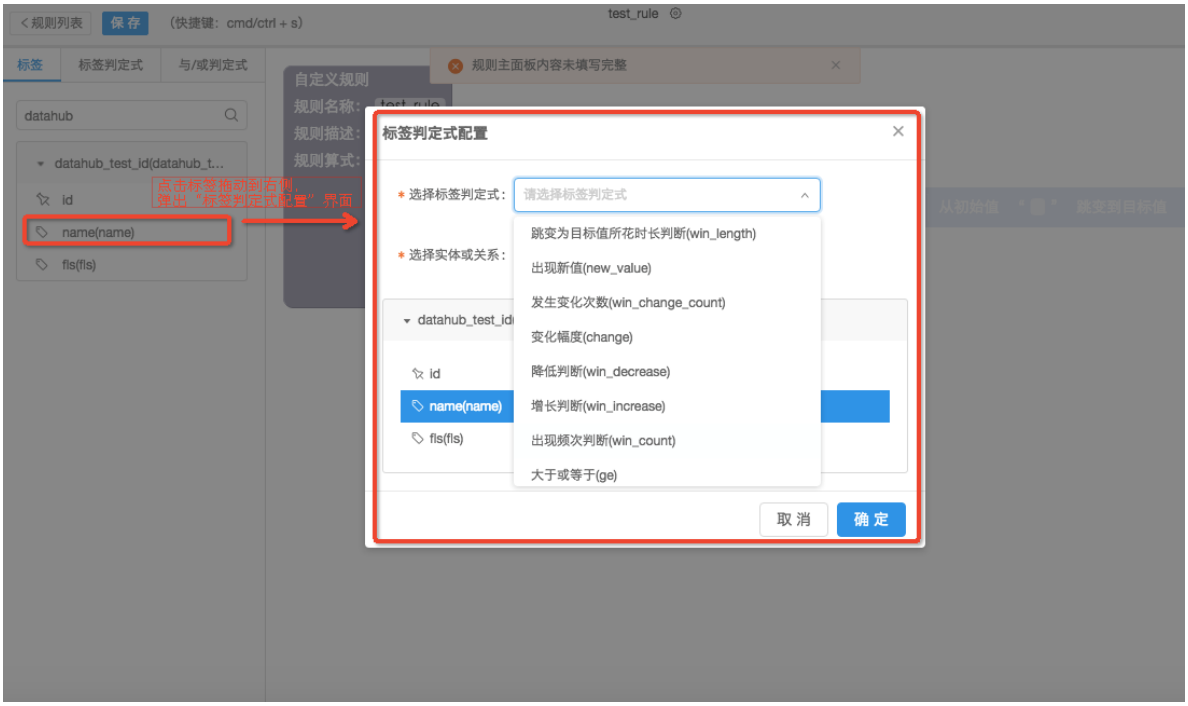
3. 如果判定式选择了 and，拖动后的效果如图 *1041: 判定式效果*所示。

图 1041: 判定式效果



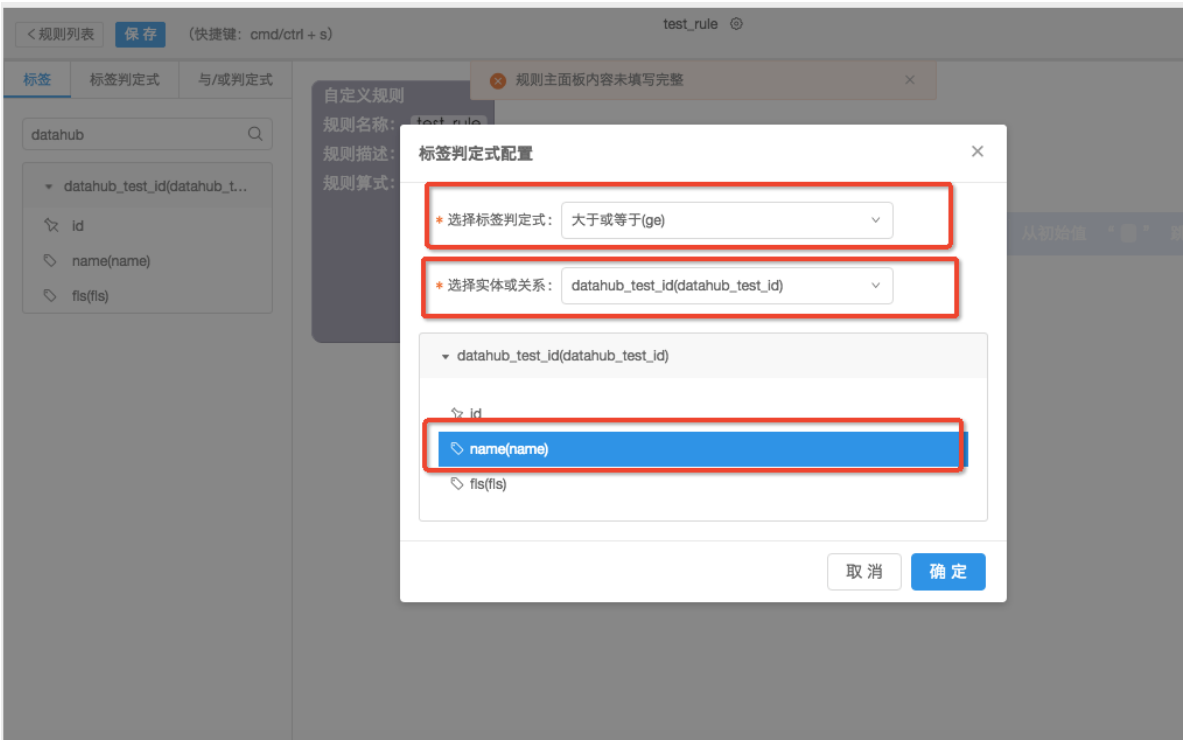
4. 从**标签**页面中拉取一个标签到右侧的自定义规则面板，如图 *1042:* 配置标签判定式所示。

图 1042: 配置标签判定式



5. 可以选择标签判定式是大于等于（ge）判定式，如图 *1043:* 配置标签判定式所示，然后单击**确定**。

图 1043: 配置标签判定式



6. 拖动判定框到 and 判定式下面匹配合拢，如图 *1044:* 拖动判定式所示。

图 1044: 拖动判定式



7. 匹配合拢之后，您可以填写对应规则的大于等于的值，如图 *1045: 判定式合拢*所示。

图 1045: 判定式合拢



这里按照先前的步骤再拉取一个标签并选择判定式，并填写对应的判定值，如图 *1046: 配置规则详细内容*所示，图中对应的规则输出标签组的中文名是输出表的中文注释，英文名是输出表的表名（规则引擎会默认生成输出表表名会在次基础上添加 _result 后缀，这个后缀可以根据用户需求进行相应更改的），当如果有一个规则判定式不需要的时候，可以通过拖动对应的判定式按照如图所示拖动到右下角的垃圾桶进行删除。配置完成之后单击左上角的**保存**按钮，更新时也可以通过此保存进行更新。

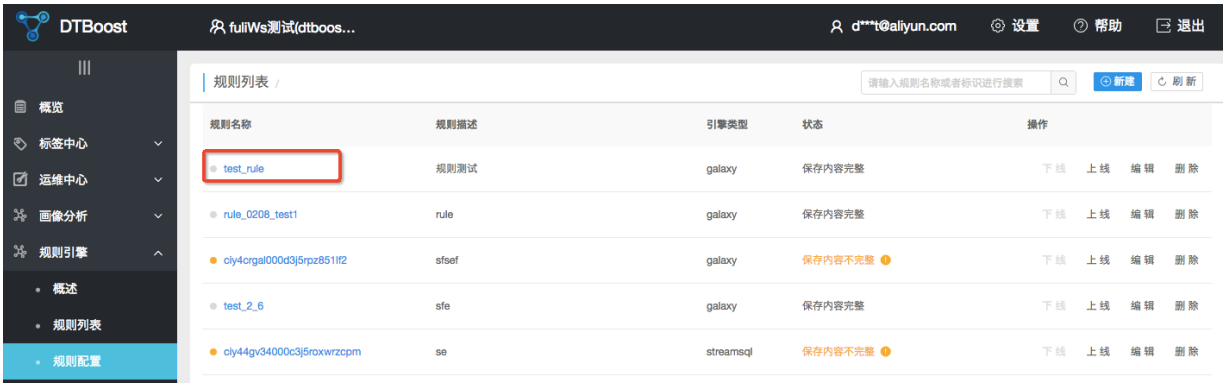
图 1046: 配置规则详细内容



30.4.2.3 规则列表

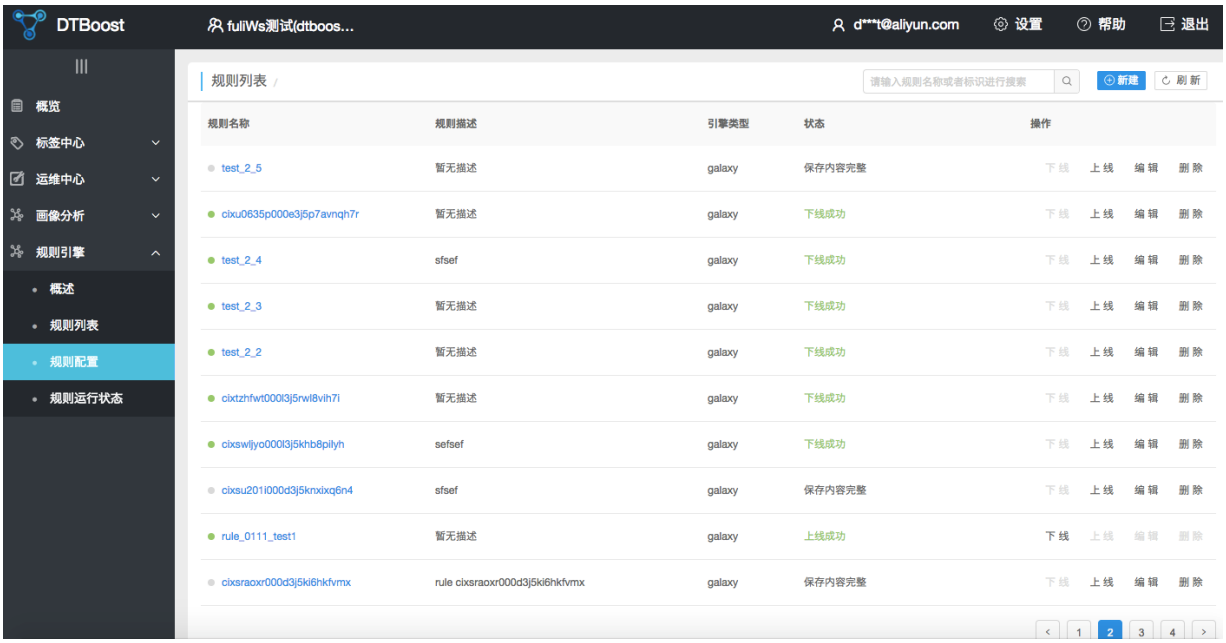
保存之后可以通过点击左侧的规则列表进行查看，如图 *1047: 规则列表*所示。图中可以看到我们刚刚 创建的 test_rule 的规则，我们可以进行**上线**、**编辑**、**删除**等操作。

图 1047: 规则列表



如图 *1048: 规则列表2*所示，对于已经上线的规则可以进行**下线**操作。

图 1048: 规则列表2



30.4.2.4 上线规则

我们对刚刚创建的规则 test_rule 进行上线操作。在规则上线的时候选择执行的云计算资源和结果输出的云计算资源，如图 *1049: 上线规则1*所示，单击**确定**。

图 1049: 上线规则1

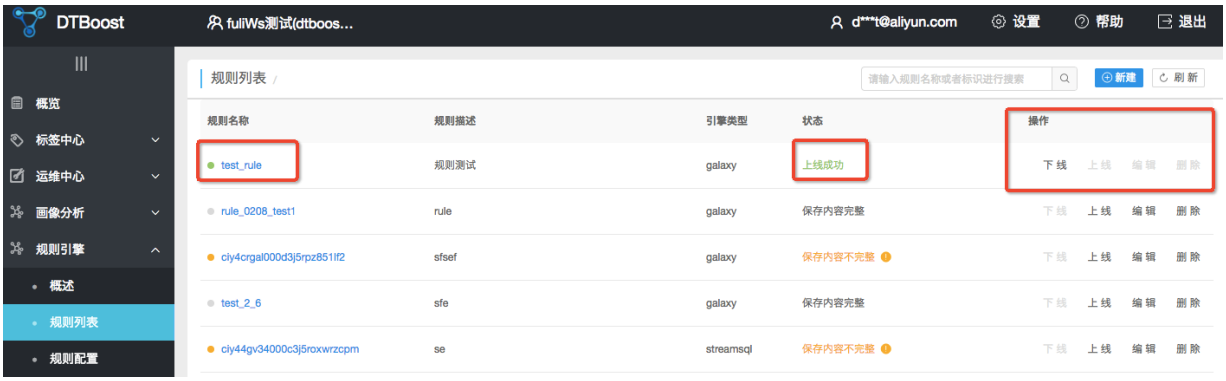


确定后，系统会显示提交中，如图 *1050: 上线规则2*所示。提交成功后，规则会显示为上线，如图 *1051: 上线规则3*所示。

图 1050: 上线规则2



图 1051: 上线规则3



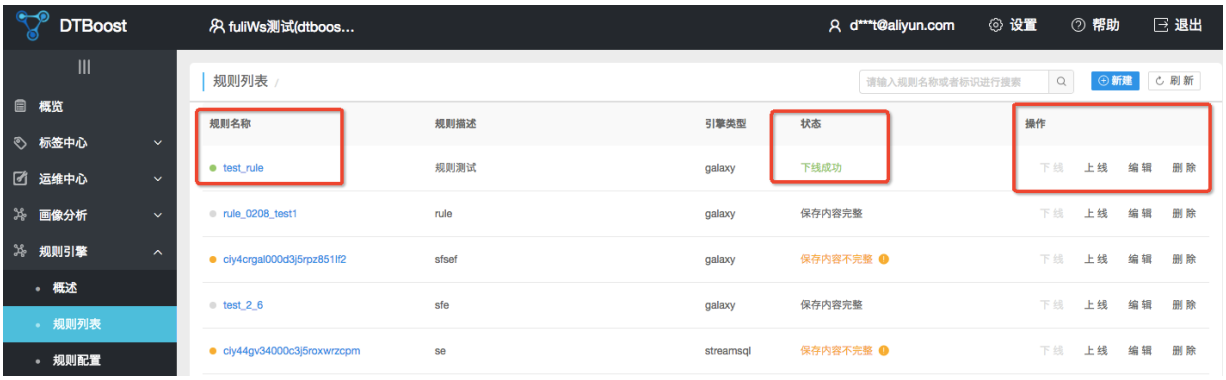
说明: 对于已经上线成功的规则只能够进行下线操作。

30.4.2.5 下线规则

我们对刚刚上线的 test_rule 规则进行下线操作，如图 *1052:* 下线规则所示。

单击**下线**之后，规则 test_rule 的状态是下线成功，操作可以有**上线、编辑、删除。**

图 1052: 下线规则



30.4.3 规则运行状态查看

对于已经上线的规则，可以查看其运行状态如图 *1053: 规则运行状态*所示。

单击左侧**规则引擎**下的**规则运行状态**，可以查看规则实时运行的状态和相关的历史触发次数。单击**查看明细**还可以查看明细，如图 *1054: 规则运行状态明细*所示。

图 1053: 规则运行状态

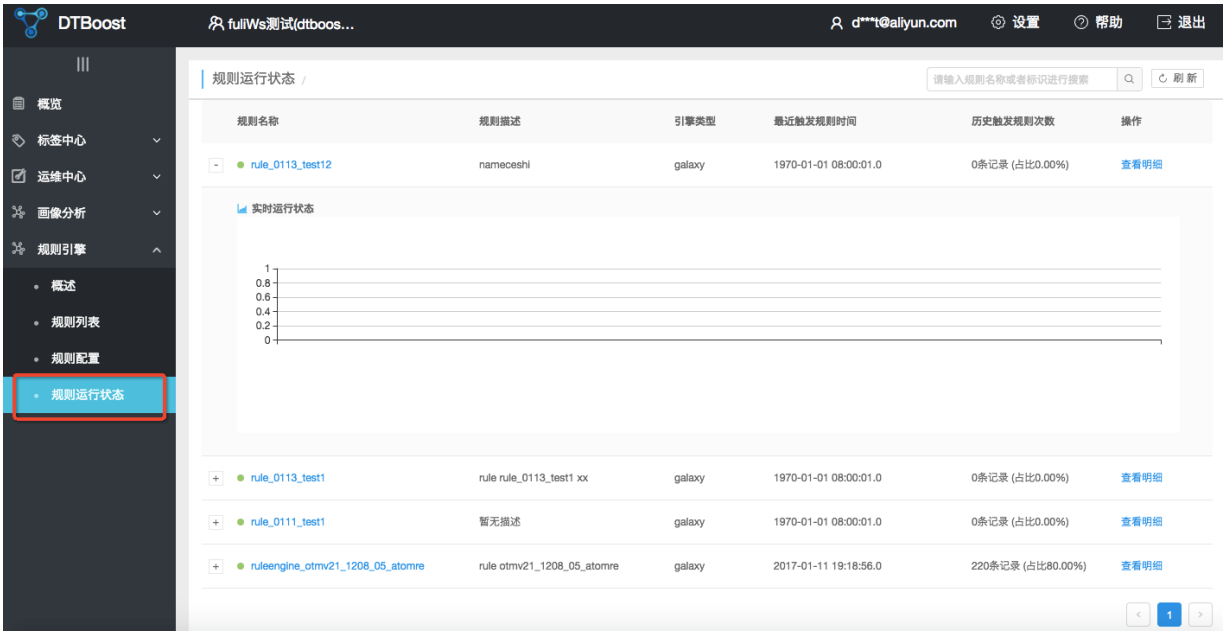
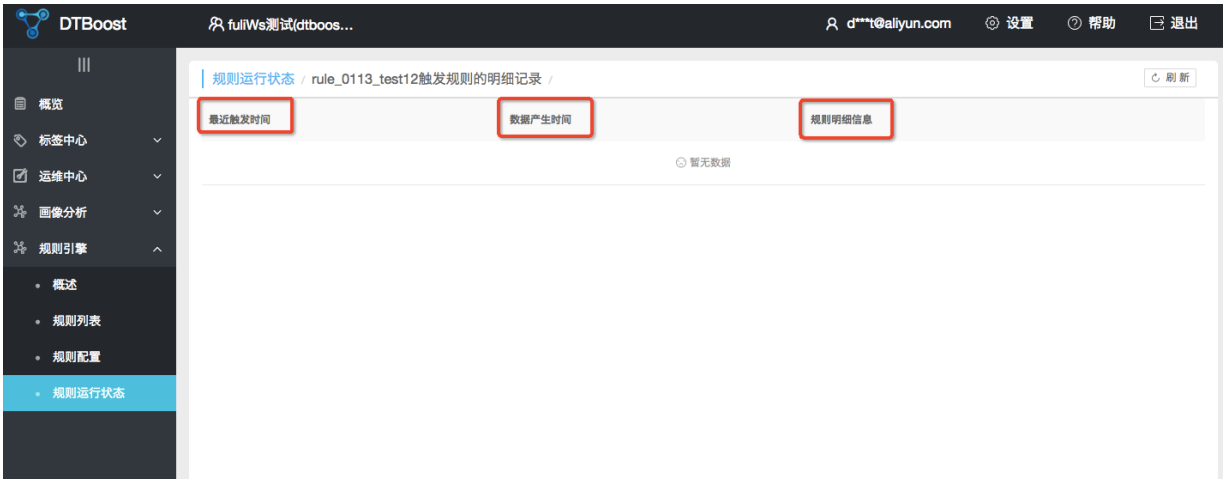


图 1054: 规则运行状态明细



30.5 常见问题

1. 问：为什么select * from object不支持？

回答：分析服务目前不支持 * 查询，需要指定本次查询的具体标签。

2. 问：为什么select object.id from object报错？

   回答： 因为一个实体可以被绑定到多个物理表，上述 TQL 只是查询实体所关联的物理列，但因为一个实体可以关联多张表的原因，上述 TQL 提供的信息不能唯一定位到一张物理表。

   解决办法：查询语句带上至少一个标签，例如，select entity.id, entity.tag1 from entity。

3. 问：为什么我的 TQL 语句报实体或者标签未找到或者同步？

   回答：

   1. 确认您在运行 TQL 之前已经选中了目标 ads 云计算资源和目标分析主体。
   2. 先确认您引用的标签 code 没有拼写错误。
   3. 到**运维中心** > **同步结果**中，选中目标云计算资源，再选中分析主体，确认实体下的维度表标签和实体表标签都已经同步成功。
   4. 确认您引用的某一个实体是否只引用了实体列，没有带上标签，造成问题第2个问题的错误。

      如果是，解决办法是在 where 条件部分加上这个实体的某个标签不为 null 的子句， 例如，where object.tag is not nul。

4. 问：为什么有的地方要两个.引用数据，比如：link.object.tag，而有的地方只要一个.就可以引用数据？

   回答：用两个.号引用数据的地方只出现在您需要引用**关系上的某个实体的绑定标识列**，比如上述例子引用是公**民犯罪关系上的公民实体所在的身份证号**这个实体列，两个.号可以理解为：**关系的.实体的.标识列**。

   一个.出现的地方有两个：一个是引用标签的地方，比如**buyer.xb** 或者 **buyer.nl**，不管是实体还是关系，引用标签都只需要一个.；另外在引用某个单一实体的绑定列，也是用一个.。

5. 问：分析服务支持那些函数及其用法说明？

   回答：分析服务支持 Analytic DB 内置的所有函数，具体用法请参见*https://help.aliyun.com/document_detail/26430.html?spm=5176.doc26430.6.146.l3qspC*。

6. 问：TQL 支持哪些 SQL 预发，为什么一个 TQL 用 join 和 in 一个子查询结果不一致？

   回答：TBoost 的 TQL 语法支持范围为 Analytic DB SQL 的语法范畴， Analytic DB SQL 的预发范畴请参见*https://docs.aliyun.com/?spm=5176.7937365.209467.4.hMrhLb#/tun/ads/user-manual/4&4.1*， join 和子查询方式结果不同的原因。

7. 问：为什么我的标签或者实体删除不掉？

回答：请严格遵循如下的删除顺序：取消同步 - 删除标签 - 删除关系 - 删除实体。

如果您的标签曾经同步成功过，请到**运维中心 > 标签同步结果**，选择**ADS云计算资源和分析主体**，在维度表或实体表下找到同步成功过的标签，选中**取消同步**，然后再去做：删除标签 - 删除关系 - 删除实体。

8. 问：为什么在 TQL 中用 in 和 Join 操作得到的最终结果不一致？

   回答：DTBoost TQL 后台运行在 Analytic DB 上，Analytic DB 对子查询有限制，`WHERE a IN (select a from fact_table where c=1): fact_table`不是维度表的情况下不支持，所以 TQL 支持 in 子查询并且要求子查询的数据只能来自维度表。

31 Quick BI

31.1 产品概述

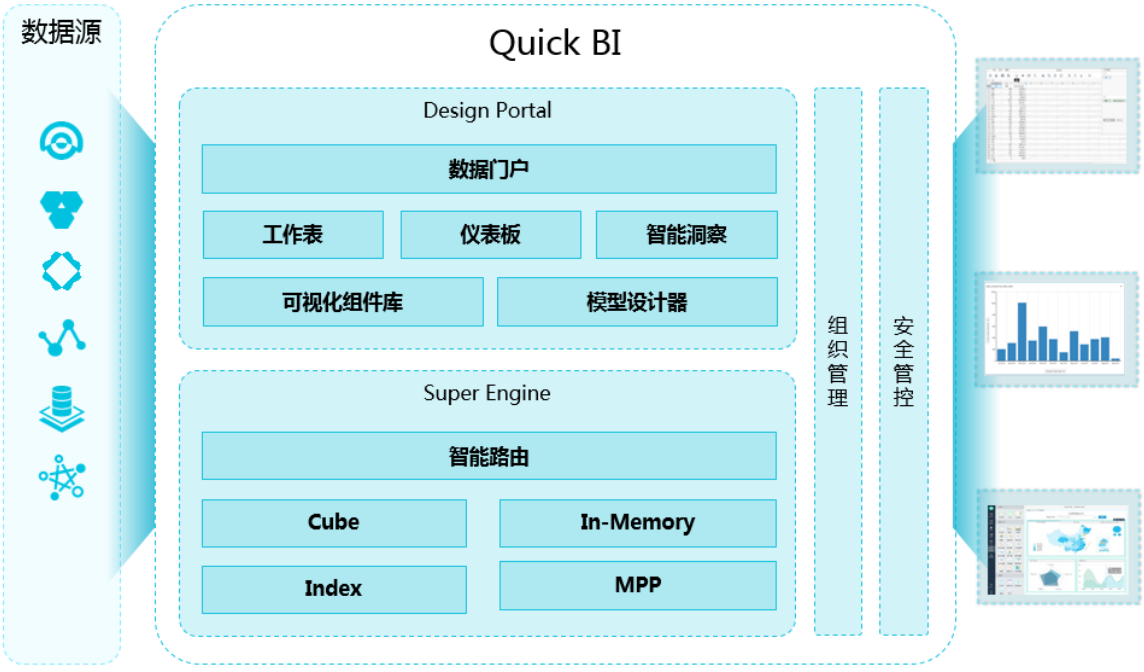
本章节将为您介绍 Quick BI 的产品特点。

Quick BI是一个基于云计算的灵活的轻量级自助 BI 工具服务平台，它支持众多种类的数据源，既可以连接 MaxCompute (ODPS)、RDS、Analytic DB、HybirdDB (Greenplum) 等云数据源，也支持连接 ECS 上您自建的 MySQL 数据库和 SQL Server 数据库。

Quick BI 可以对海量数据进行实时在线分析，您无需提前进行大量的数据预处理就可以得到秒级回应。Quick BI 还支持每天 TB 级的增量数据。

通过提供智能化的数据建模工具以及丰富的可视化图表工具，Quick BI 极大地降低了数据的获取成本和使用门槛，帮助您轻松地完成数据透视分析、自助取数、业务数据查询、报表制作等工作。

图 1055: Quick BI 产品架构



31.2 登录 Quick BI

本章节将为您介绍如何登录 Quick BI 产品。

背景信息

如果您已经登录了阿里云控制台，您可以通过导航菜单，直接登录 Quick BI 控制台，无需您再次登录。

操作步骤

1. 输入登录名和密码，登录阿里云控制台。
2. 单击**云资源中心** > **云产品管理**，如图 *1056: 登录控制台*所示。

图 1056: 登录控制台



3. 单击 **Quick BI**。
4. 选择一个部门，单击产品图标，进入 Quick BI 控制台，如图 *1057: 登录 Quick BI*所示。

图 1057: 登录 Quick BI



- 单击 Quick BI 图标，您可以直接登录产品首页并开始使用。
- 单击 Quick BI 管理控制台图标，您可以直接登录产品的管理控制台，对该产品进行续费、添加新用户等操作。

31.3 数据建模

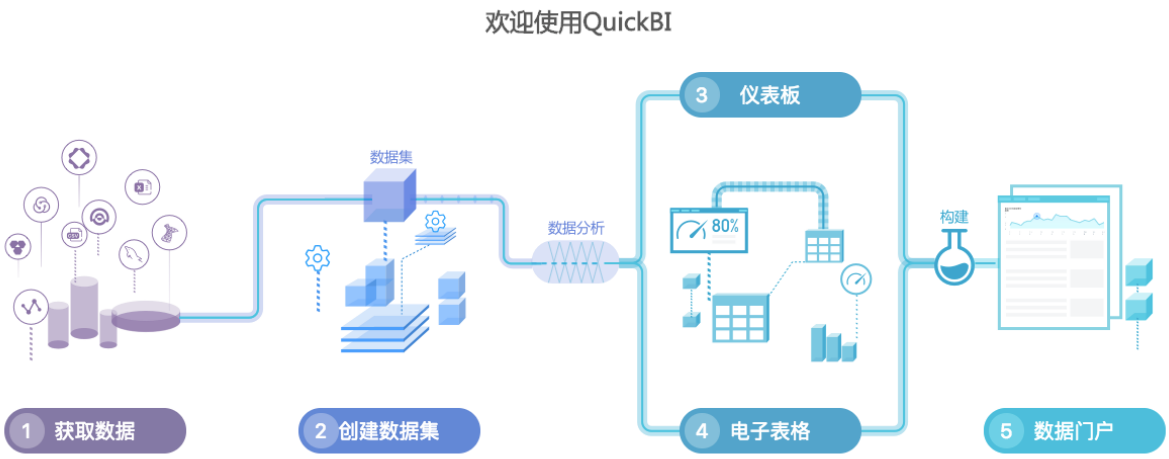
本章节将为您介绍如何使用 Quick BI 实现数据建模。

数据建模的基本流程可分为以下几个步骤：

- 新建数据源
- 选择数据源中的表来创建数据集
- 通过自定义 SQL 来创建数据集（可选）

建模的基本流程如图 *1058:* 数据建模所示。

图 1058: 数据建模



31.3.1 管理数据源

本章节将为您介绍 Quick BI 所支持的数据源类型。

Quick BI 支持以下几种类型的数据源：

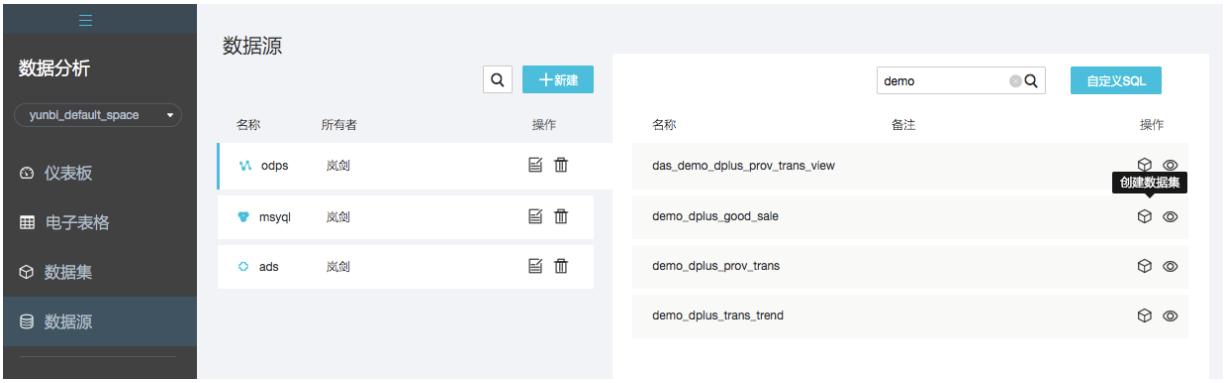
- 云数据源（MaxCompute（ODPS），RDS（MySQL），RDS（SQL Server），HybirdDB（Greenplum），AnalyticDB（ADS））
- ECS下的数据源（MySQL，SQL Server）

说明: SQL Server 暂不支持使用视图。

31.3.1.1 数据源列表

数据源列表可以集中管理所有的数据源，并在列表页上提供**新建数据源**，**编辑数据源**，**删除数据源**等命令的操作入口，如图 *1059:* 数据源列表所示。

图 1059: 数据源列表



31.3.1.2 新建数据源

背景信息

数据集、工作表、仪表板和数据门户等的操作都需要数据源作为基础。本章节将为您介绍如何新建数据源。

操作步骤

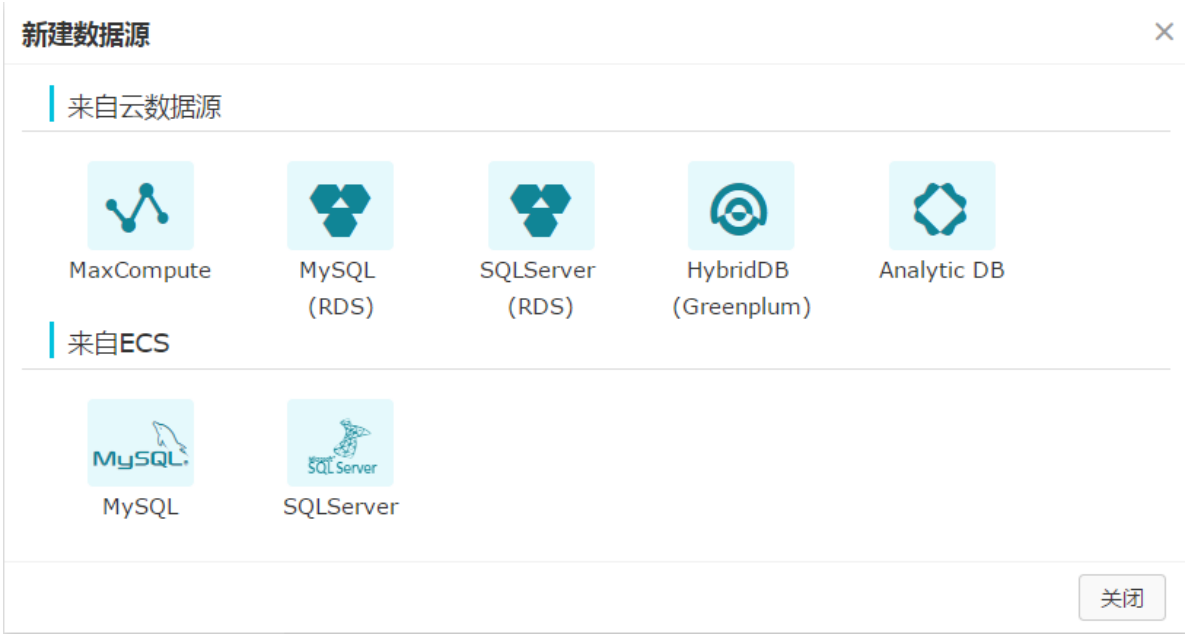
1. 在 Quick BI 控制台单击**数据**图标进入数据管理页面。
2. 单击**数据源**标签页进入数据源管理页面。
3. 单击**新建数据源**，如图 *1060:* 新建数据源所示。

图 1060: 新建数据源



4. 在新弹出的对话框中选择一个数据源的来源，如图 *1061:* 选择数据源来源所示。

图 1061: 选择数据源来源



5. 根据系统提示，填写数据源所需的各种信息，然后单击**添加**。

31.3.1.2.1 来自云数据源

本章节将为您介绍云数据源的新建流程。

31.3.1.2.1.1 MaxCompute (ODPS) 数据源

操作步骤

1. 单击 MaxCompute 图标，系统会自动弹出添加数据源的对话框，如图 *1062:* 添加数据源所示。

图 1062: 添加数据源



2. 按照系统提示，在对话框中输入所需要的数据源连接信息，包括：
   - 显示名称：数据源配置列表的显示名称
   - 数据库地址：此处有默认地址，无需修改
   - 项目名称：项目名称
   - Access Id：阿里云官网管控台Access Key ID
   - Access Key：阿里云官网管控台Access Key Secret
3. 单击**连接测试**按钮，先进行数据源连通性测试。如果测试通过，系统会弹出以下提示：

   图 1063: 连接测试



4. 连接测试通过后，单击**添加**。

添加成功后，页面会自动跳转到数据源列表页，并在页面右侧展示数据源所包含的数据表列表。

31.3.1.2.1.2 RDS （MySQL）

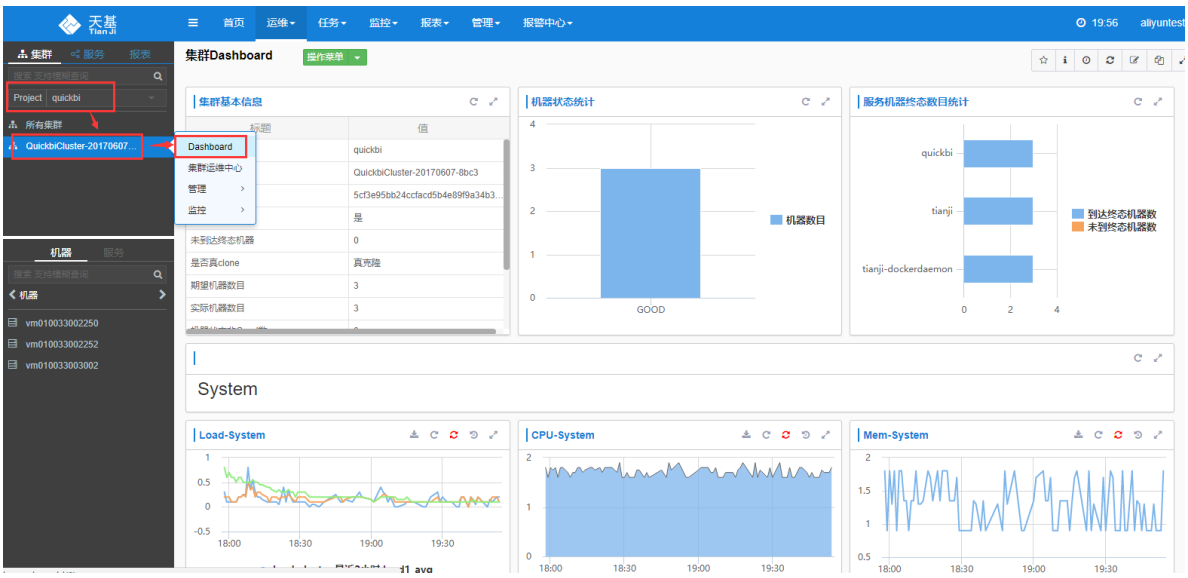
背景信息

受限于 RDS 的白名单策略，在添加 RDS 数据源时，您需要先登录天基，查询到可用的 IP 地址后，手动添加到 RDS 控制台上。

操作步骤

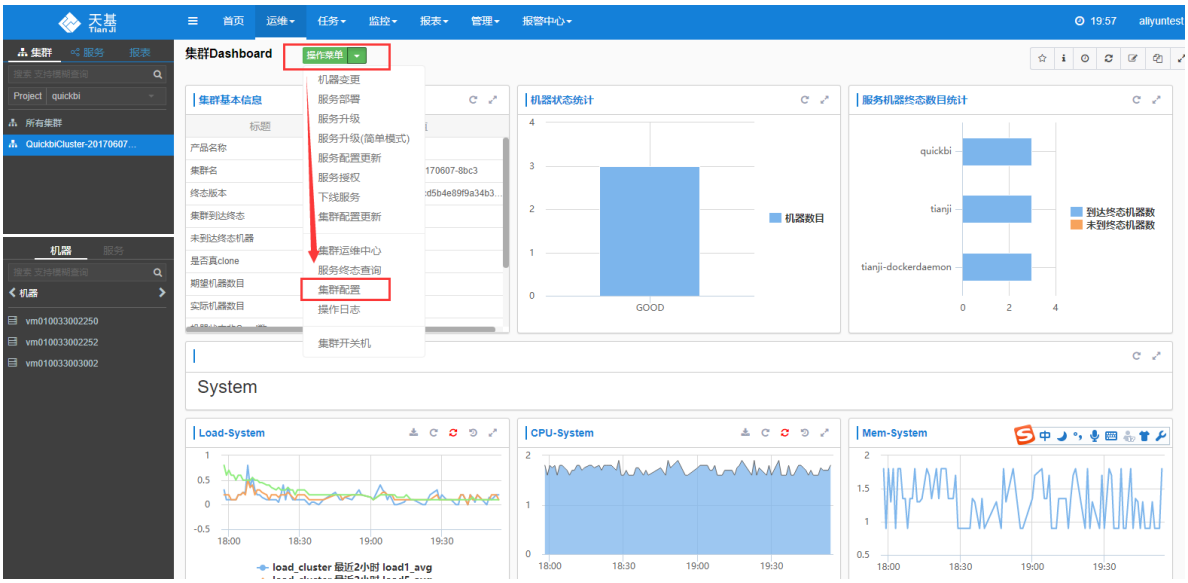
1. 登录天基，在页面上方，单击**运维**，进入到运维目录。
2. 在左侧导航栏中，选择**集群**标签页。
3. 在 Project 搜索框中，输入 Quick BI，找到 Quick BI 集群。
4. 单击 Quick BI 集群，在弹出的菜单中，选择 Dashboard，如图 *1064: 集群 Dashboard*所示。

图 1064: 集群 Dashboard



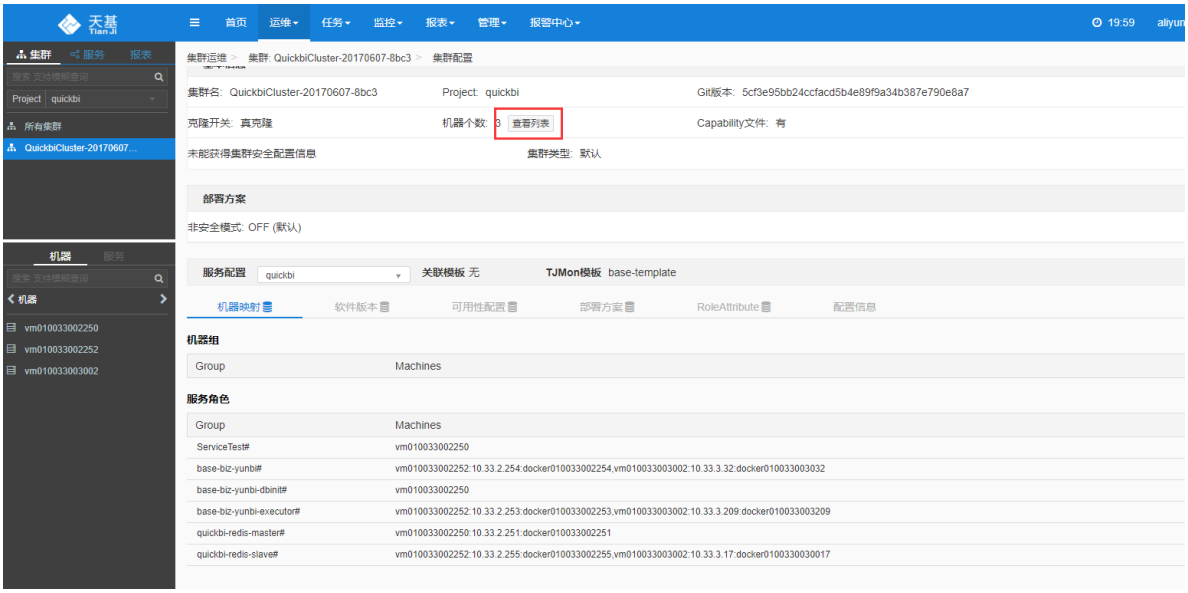
5. 在集群 Dashboard 页面，单击**操作菜单**。
6. 在弹出的下拉列表中，选择**集群配置**，如图 *1065: 集群配置*所示。

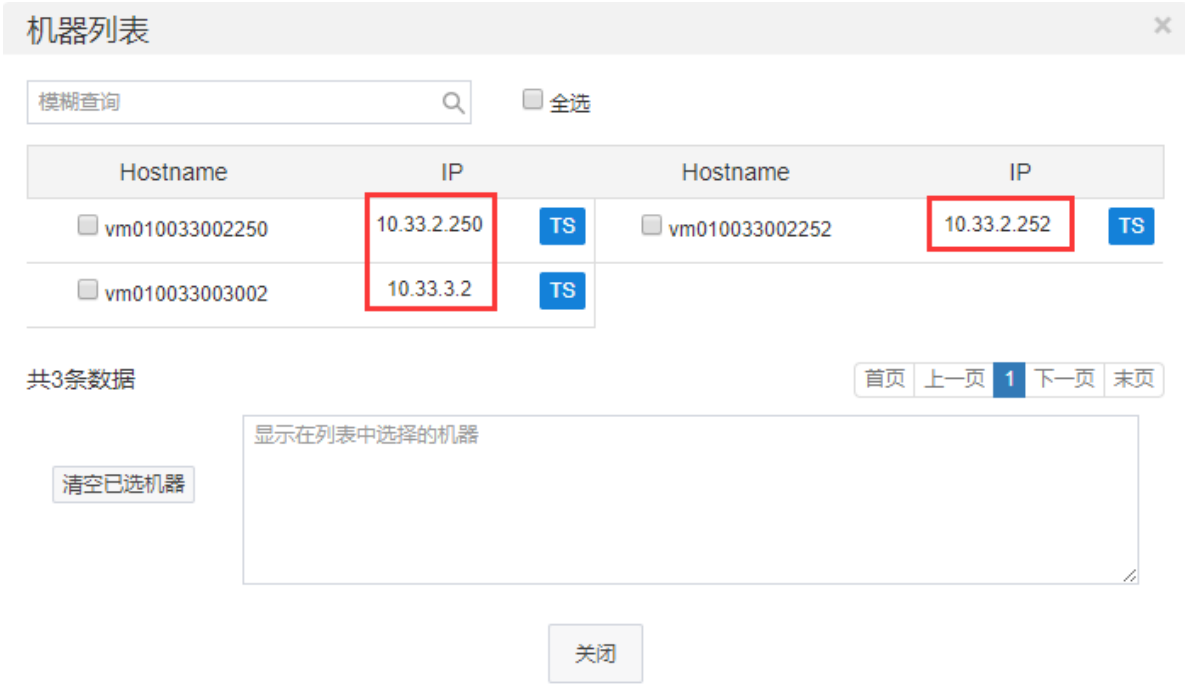
图 1065: 集群配置



7. 在集群配置页面，系统会自动给出可用的机器个数，单击后面的**查看列表**，查看到机器的 IP 地址，如图 *1066:* 查看机器列表所示。

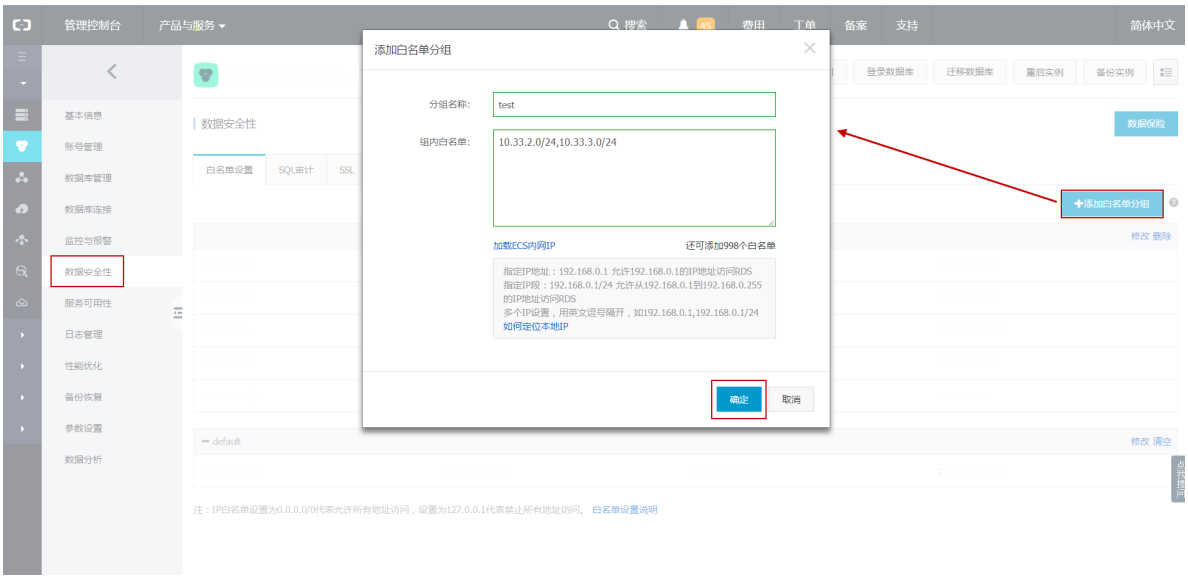
图 1066: 查看机器列表





8. 在机器列表中，选择 IP 地址前三位不重复的地址，并在后面添加.0/24，例如10.33.2.0/24和10.33.3.0/24。
9. 将选中的 IP 地址用英文逗号相连，例如10.33.2.0/24,10.33.3.0/24，并将它们添加到 RDS 白名单中，如图 *1067: 添加 IP地址*所示。

图 1067: 添加 IP地址



10. IP 地址添加完成后，退出天基，登录 Quick BI 控制台。
11. 单击**作品**图标，在**数据源**标签页，单击**新建数据源**。

12.选择 RDS (MySQL) 图标，在新弹出的对话框中输入所需要的数据源连接信息。

图 1068: 连接 RDS



- 显示名称：数据源列表显示名称
- 数据库地址：写主机名或IP地址即可
- 端口号：填写正确的端口号即可
- 数据库
- 用户名
- 密码

说明: 如果您不确定您的用户名和密码，请联系您的数据仓库管理员获取。

13.单击**连接测试**按钮，进行数据源连通性测试。

如测试成功，系统会给出连通测试成功提示，否则，系统会给出失败提示。

14.连通测试成功后，单击**添加**。

如果数据源已经存在，系统会给出错误提示。

31.3.1.2.1.3 RDS（SQL Server）数据源

背景信息

受限于 RDS 的白名单策略，在添加 RDS 数据源时，您需要先登录天基，查询到可用的 IP 地址后，手动添加到 RDS 控制台上。如果您想了解如何添加 IP 地址到 RDS 控制台，请参阅*RDS #MySQL#*。

RDS（SQL Server）数据源的配置方法与 RDS（MySQL）数据源的配置方法类似。多了一个RDS（SQL Server）数据源特有的配置项：schema。

操作步骤

1. 单击 RDS（SQL Server）图标，在新弹出的对话框中输入所需要的数据源连接信息，如图 *1069: 添加 RDS #SQL Server#* 数据源所示。

   图 1069: 添加 RDS（SQL Server）数据源



- 显示名称：数据源列表显示名称
- 数据库地址：填写主机名或IP地址即可
- 端口号：填写正确的端口号即可
- 数据库
- Schema：dbo
- 用户名
- 密码

2. 单击**连接测试**按钮，进行数据源连通性测试。

   如测试成功，系统会给出连接测试成功提示，否则，系统会给出连接测试失败的提示。

3. 连接测试成功后，单击**添加**。

31.3.1.2.1.4 Hybird DB（Greenplum）数据源

背景信息

受限于 RDS 的白名单策略，在添加 RDS 数据源时，您需要先登录天基，查询到可用的 IP 地址后，手动添加到 RDS 控制台上。如果您想了解如何添加 IP 地址到 RDS 控制台，请参阅，*RDS #MySQL#*。

Hybrid DB（Greenplum）数据源的配置方法与RDS（SQL Server）数据源的配置方法类似。默认端口会变成Greenplum特有的端口。

操作步骤

1. 单击 HybirdDB（Greenplum）图标，在新弹出的对话框中输入所需要的数据源连接信息，如图*1070: 添加 Hybird DB#Greenplum#数据源*所示。

   图 1070: 添加 Hybird DB（Greenplum）数据源



- 显示名称：数据源列表显示名称
- 数据库地址：填写主机名或IP地址即可
- 端口号：填写正确的端口号即可
- 数据库
- Schema：public
- 用户名
- 密码

2. 单击**连接测试**按钮，进行数据源连通性测试。

   如测试成功，系统会给出连接测试成功提示，否则，系统会给出连接测试失败的提示。

3. 连接测试成功后，单击**添加**。

31.3.1.2.1.5 Analytic DB 数据源

背景信息

Analytic DB 也叫 ADS。

操作步骤

1. 单击 Analytic DB 图标，在新弹出的对话框中输入所需要的数据源连接信息，如图 *1071: 添加 Analytic DB 数据源*所示。

   图 1071: 添加 Analytic DB 数据源



   - 显示名称：数据源列表显示名称
   - 数据库地址：填写主机名或IP地址即可
   - 端口号：填写正确的端口号即可
   - 数据库
   - Access Id
   - Access Key

2. 单击**连接测试**按钮，进行数据源连通性测试。

   如测试成功，系统会给出连接测试成功提示，否则，系统会给出连接测试失败的提示。

3. 连接测试成功后，单击**添加**。

31.3.1.2.2 来自ECS数据源

本章节将为您介绍如何用 Quick BI 产品添加来自 ESC 的数据源。

31.3.1.2.2.1 MySQL

背景信息

受限于 RDS 的白名单策略，在添加 RDS 数据源时，您需要先登录天基，查询到可用的 IP 地址后，手动添加到 RDS 控制台上。如果您想了解如何添加 IP 地址到 RDS 控制台，请参阅，*RDS #MySQL#*。

ECS 下的 MySQL 数据源的配置方法与 RDS（MySQL）数据源的配置方法类似。

操作步骤

1. 单击 MySQL 图标，在新弹出的对话框中输入所需要的数据源连接信息，如图 *1072: 添加 MySQL 数据源*所示。

   图 1072: 添加 MySQL 数据源



- 显示名称：数据源列表显示名称
- 数据库地址：填写主机名或IP地址即可
- 默认是3306
- 数据库
- 用户名
- 密码

2. 单击**连接测试**按钮，进行数据源连通性测试。
3. 连接测试成功后，单击**添加**。
4. 您需要打开 ECS 下的防火墙，外部才能访问 MySQL。请输入以下命令进入防火墙配置文件：

   vi /etc/sysconfig/iptables
5. 在防火墙配置文件中增加以下命令：

   -A RH-Firewall-1-INPUT -m state –state NEW -m tcp -p tcp –dport 3306 -j ACCEPT

6. 配置成功后，可通过以下命令重新启动 iptable。

   service iptables restart

31.3.1.2.2.2 SQL Server 数据源

背景信息

受限于 RDS 的白名单策略，在添加 RDS 数据源时，您需要先登录天基，查询到可用的 IP 地址后，手动添加到 RDS 控制台上。如果您想了解如何添加 IP 地址到 RDS 控制台，请参阅，*RDS #MySQL#*。

ECS 下的 SQL Server 数据源的配置方法与 RDS（SQL Server）数据源的配置方法类似。

操作步骤

1. 单击 SQL Server 图标，在新弹出的对话框中输入所需要的数据源连接信息。

   图 1073: 添加 SQL Server 数据源



- 显示名称：数据源列表显示名称
- 数据库地址：填写主机名或IP地址即可
- 端口号：默认是1433
- 数据库
- Schema：dbo
- 用户名
- 密码

2. 单击**连接测试**按钮，进行数据源连通性测试。
3. 连接测试成功后，单击**添加**。

> **说明:** 数据源不可重复添加，否则系统会给出错误提示。

31.3.1.3 编辑数据源

操作步骤

1. 在数据源列表中，选中一个数据源。
2. 单击**编辑**，可以对该数据源进行编辑。

31.3.1.4 删除数据源

背景信息

如果基于该数据源创建了数据集，则此数据源将无法删除，如图 *1074:* 删除数据源所示。

图 1074: 删除数据源



操作步骤

1. 在数据源列表中，选中一个数据源。
2. 单击**删除**，可以删除该数据源。

31.3.1.5 查询数据源

操作步骤

1. 在 Quick BI 控制台单击**数据**图标，进入数据管理页面。
2. 选择**数据源**标签页。
3. 在搜索框中输入关键字查询需要的数据源，如图 *1075:* 查询数据源所示。

图 1075: 查询数据源



31.3.1.6 查询数据源中包含的表

操作步骤

1. 在 Quick BI 控制台，单击**数据**图标，进入数据管理页面。
2. 选择**数据源**标签页。
3. 在列表中选择一个数据源，该数据源下所包含的表会自动列在页面右侧。
4. 在右侧页面的搜索框中输入关键字段搜索需要的表，如图 *1076:* 查询数据源中包含的表所示。

图 1076: 查询数据源中包含的表



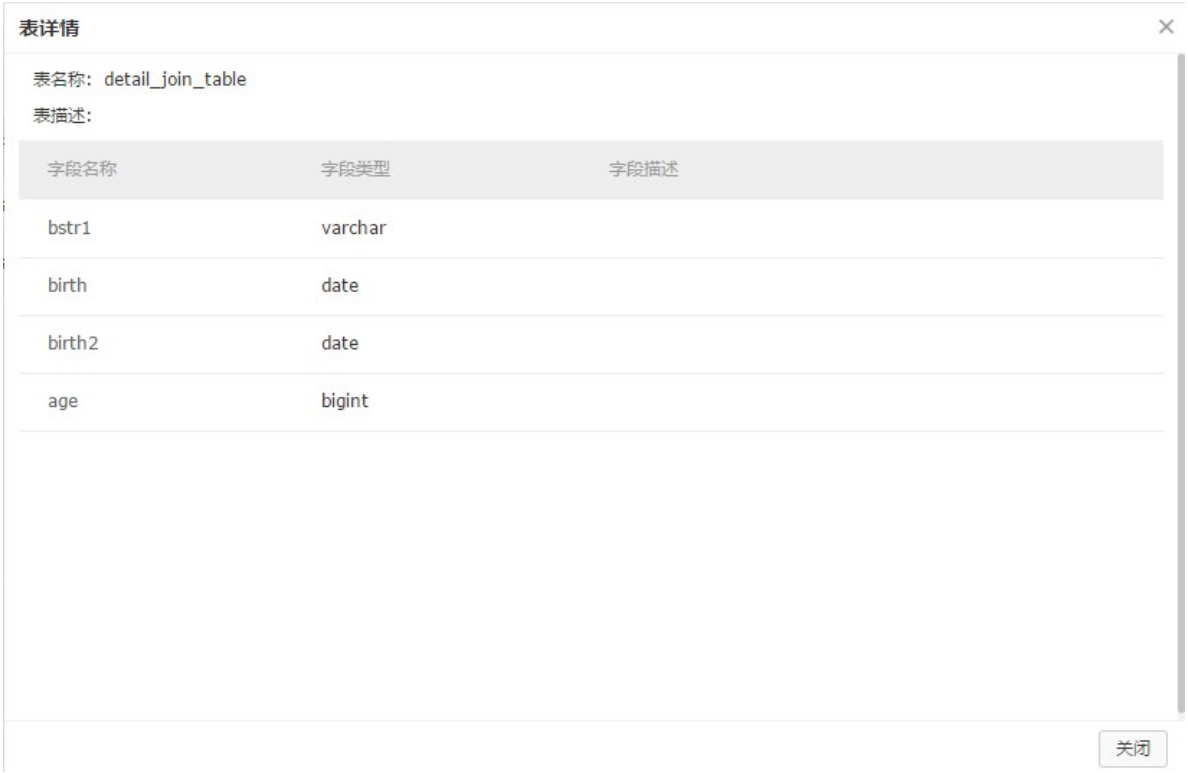
31.3.1.7 查询数据源中表的详情

操作步骤

1. 在 Quick BI 控制台，单击**数据**图标，进入数据管理页面。
2. 选择**数据源**标签页。
3. 在列表中选择一个数据源，该数据源下所包含的表会自动列在页面右侧。

4. 选择一张表，单击后面的**详情**，可以查看表以及字段详情，如图 *1077:* 查询数据源中表的详情所示。

图 1077: 查询数据源中表的详情



31.3.2 管理数据集

本章节将为您介绍如何用 Quick BI 管理数据集。

数据源中，各种不同的表可以创建为数据集。以下章节将为您介绍数据集的创建、编辑、查询等操作。

31.3.2.1 创建数据集

数据集可通过以下两种方式创建：

- 从数据源创建
- 通过自定义 SQL 创建

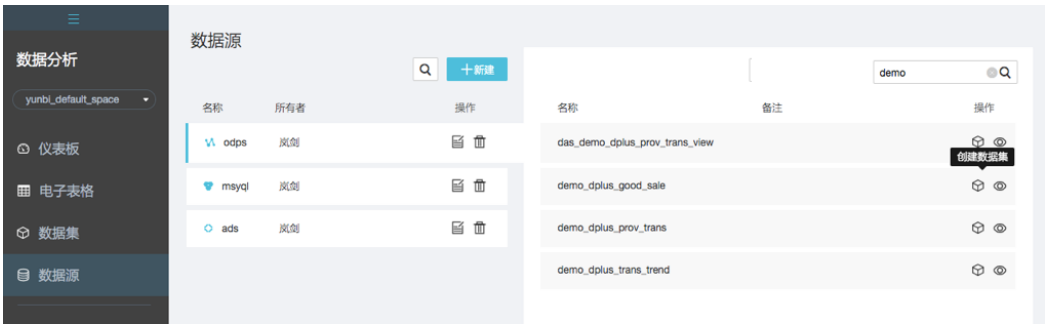
31.3.2.1.1 从数据源创建

操作步骤

1. 在 Quick BI 控制台，单击**数据**图标，进入数据管理页面。

2. 选择**数据源**标签页。
3. 在列表中选择一个数据源，该数据源下所有的表会自动列在页面右侧。
4. 选中其中一个表，单击后面的**创建数据集**，如图 *1078: 从数据源创建*所示。

图 1078: 从数据源创建



创建成功后，页面会自动跳转到数据集标签页，新创建的数据集会显示在数据集列表中，并且会带有 New 的图标，方便您快速定位新的数据集，如图 *1079: 新添加数据集*所示。

图 1079: 新添加数据集



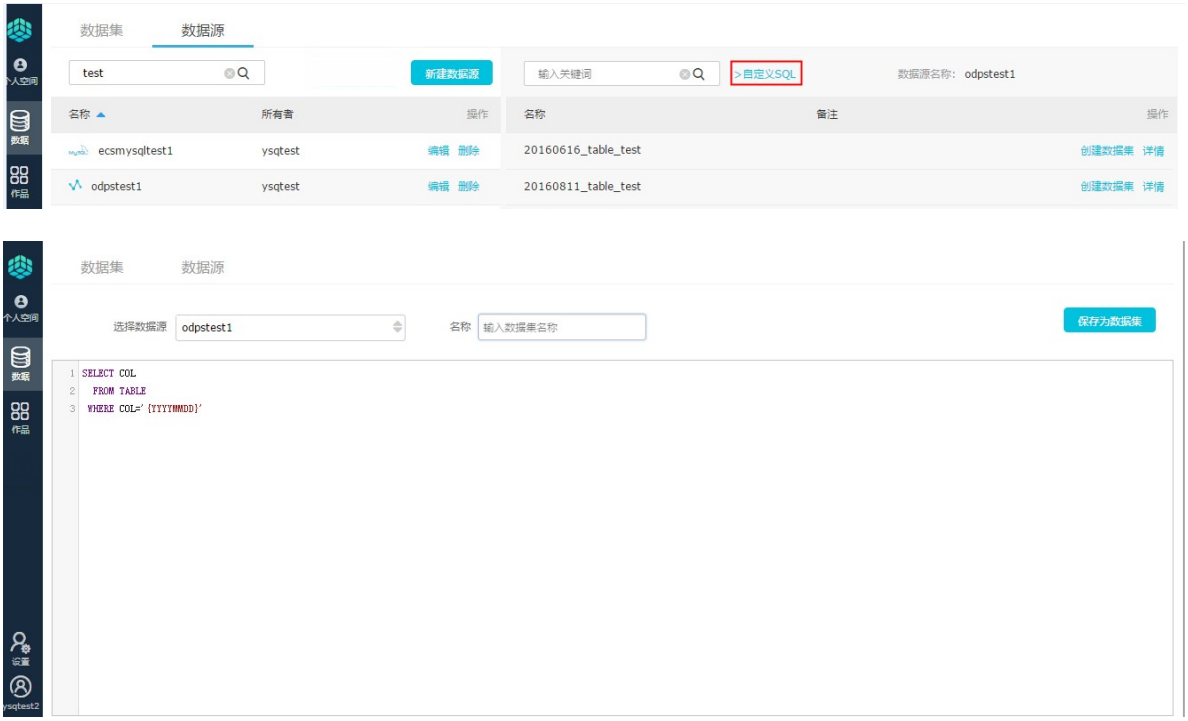
31.3.2.1.2 MaxCompute 数据源下自定义 SQL

目前仅在 MaxCompute（ODPS）数据源下支持自定义 SQL。

操作步骤

1. 在 Quick BI 控制台，单击**数据**图标，进入数据管理页面。
2. 选择**数据源**标签页。
3. 在列表中选择一个来自 MaxCompute 的数据源。
4. 在页面右侧，单击**自定义 SQL**创建数据集，如图 *1080: 自定义 SQL*所示。

图 1080: 自定义 SQL



创建完成后，可直接保存为数据集。

31.3.2.2 设置数据集默认名称

背景信息

Quick BI 会自动根据物理表的元数据信息创建数据集，将物理表中的字段转换为数据集中的维度或度量。维度和度量的名字是自动生成的，名字可以取自物理字段名，也可以取自字段名的注释。在管理面板的**偏好设置**中，您可以根据自己的喜好来选择，偏好更改以后，后续新创建的数据集中的维度和度量的命名就按照偏好设置的要求进行自动命名。

操作步骤

1. 在 Quick BI 控制台，单击**设置**图标，进入设置页面。
2. 单击**个人设置** > **偏好设置**，如图 *1081:* 设置数据集默认名称所示。

图 1081: 设置数据集默认名称



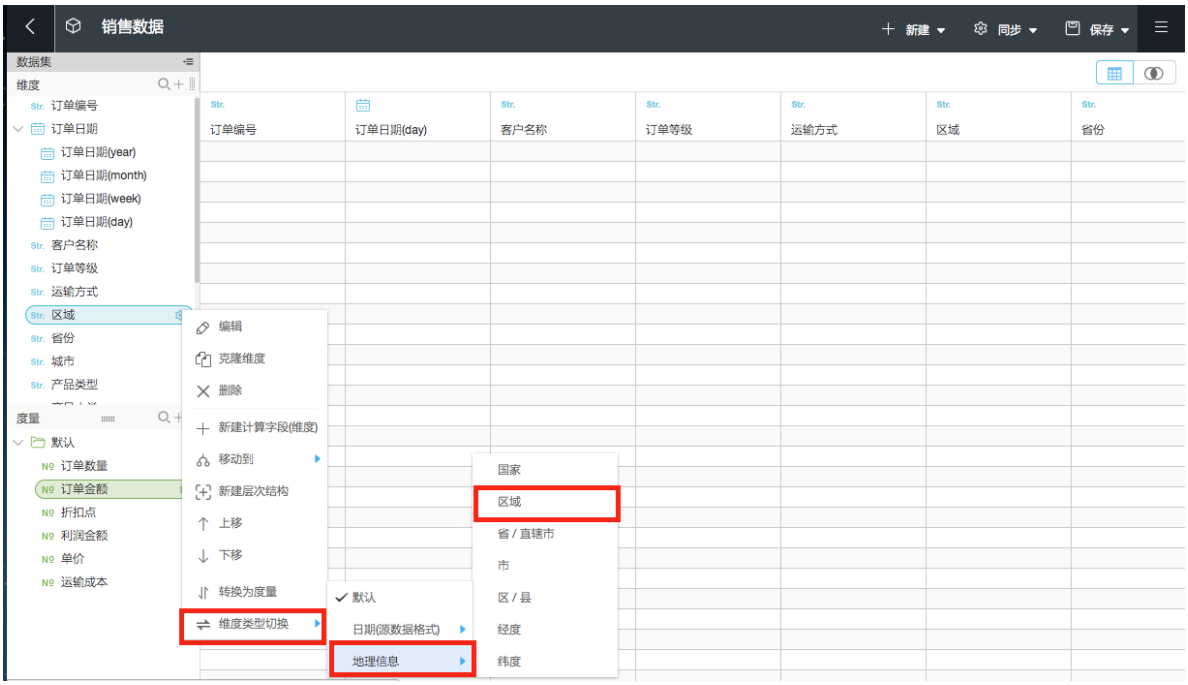
31.3.2.3 编辑数据集

操作步骤

1. 在 Quick BI 控制台，单击**数据**图标，进入数据管理页面。
2. 选择**数据集**标签页。
3. 选择一个数据集，单击后面的**编辑**进入数据集编辑页面，如图 *1082:* 数据集编辑页面所示。

图 1082: 数据集编辑页面





31.3.2.3.1 编辑维度

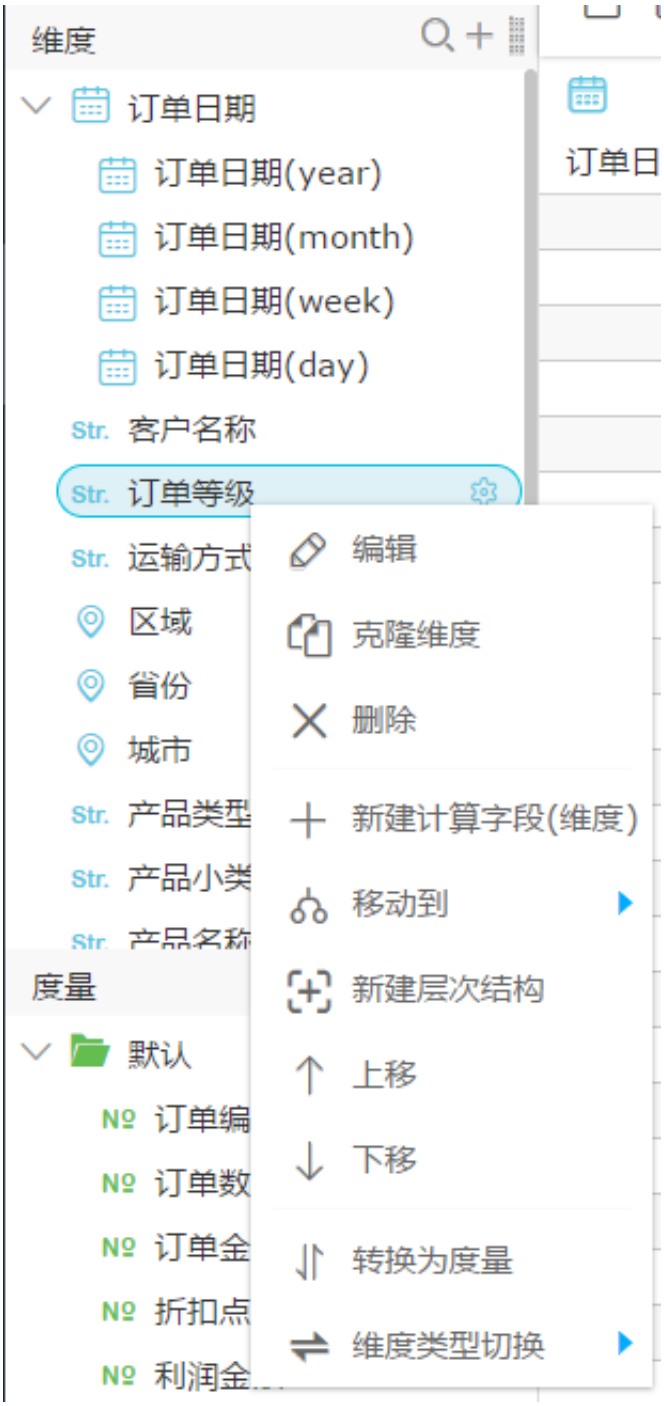
背景信息

表字段中，如果是字符或其它类型，系统会默认为维度。您可以在维度和度量列表中，对字段进行编辑。

操作步骤

1. 在维度列表中，选择一个字段。
2. 单击鼠标右键，编辑菜单如所示。

图 1083: 维度编辑菜单



- 编辑：修改维度显示名以及备注信息。
- 克隆维度：快速复制一个维度，生成的维度将会自动带上副本以做提示。
- 删除：删除该字段。
- 新建计算字段（维度）：可新创建一个维度字段，并且可自定义其计算方式。
- 移动到层次结构：快速将维度纳入到已有层次结构中，可用来实现钻取。
- 新建层次结构：快速将维度纳入到新建的层次结构中。

- 移出层次结构：移除层次结构，维度不会删除。
- 上移/下移：移动字段位置，支持鼠标拖拽和右键。
- 转换为度量：可将当前维度字段转换为度量字段。
- 维度类型切换：默认、日期以及地理维度的切换，您可以通过单击维度的操作按钮或者单击鼠标右键打开编辑菜单。

  例如，创建气泡地图和色彩地图时，您需要切换维度类型为地理信息，否则将无法正常制作地图图表。

31.3.2.3.2 编辑度量

背景信息

表字段中，如果是数值类型，系统会默认为度量。您可以在维度和度量列表中，对字段进行编辑。

操作步骤

1. 在度量列表中，选择一个字段。
2. 单击鼠标右键，编辑菜单如图 *1084:* 度量编辑菜单所示。

图 1084: 度量编辑菜单



- 编辑：修改维度显示名以及备注信息。
- 删除: 删除该字段。
- 新建计算字段：可新建一个度量字段，并且可自定义其计算方式。
- 移动到（文件夹）：快速将度量纳入到已有文件夹中。
- 上移/下移：移动字段位置，支持鼠标拖拽和右键。
- 转换为维度：可将当前度量字段转换为维度字段。
- 数字格式化：可决定数字的显示格式。
- 默认聚合方式：可在菜单中选择聚合方式，如求和，最大值，最小值等。

31.3.2.3.3 工具按钮

Quick BI 提供以下工具按钮，如图 *1085:* 工具按钮区所示，供您来编辑数据集。

图 1085: 工具按钮区



新建

您可以通过**新建**功能，新建仪表板和电子表格，如图 *1086: 新建*所示。

图 1086: 新建



- 新建仪表板：将当前数据集自动带到新建仪表板中去使用。
- 新建电子表格：将当前数据集自动带到新建电子表格中去使用。

同步

您可以通过**同步**功能同步表结构和刷新数据集数，如图 *1087: 同步*所示。

图 1087: 同步



- 同步表结构：此操作可以用于线上物理表发现改变，如新增了某字段，这样可以很方便的从线上把字段同步下来。

  如果线上表字段被删除、更改了字段名、comment 修改或者维表结构变化，系统将不会对数据集做相应删除处理，如图 *1088: 同步表结构*所示。

  图 1088: 同步表结构



- 刷新数据集数据：刷新数据集的预览数据。如果您需要实时查看最新的数据，请保存数据集后刷新数据。

保存

您可以通过**保存**功能，保存或另存数据集，如图 *1089: 保存*所示。

图 1089: 保存



- 保存：保存数据集。

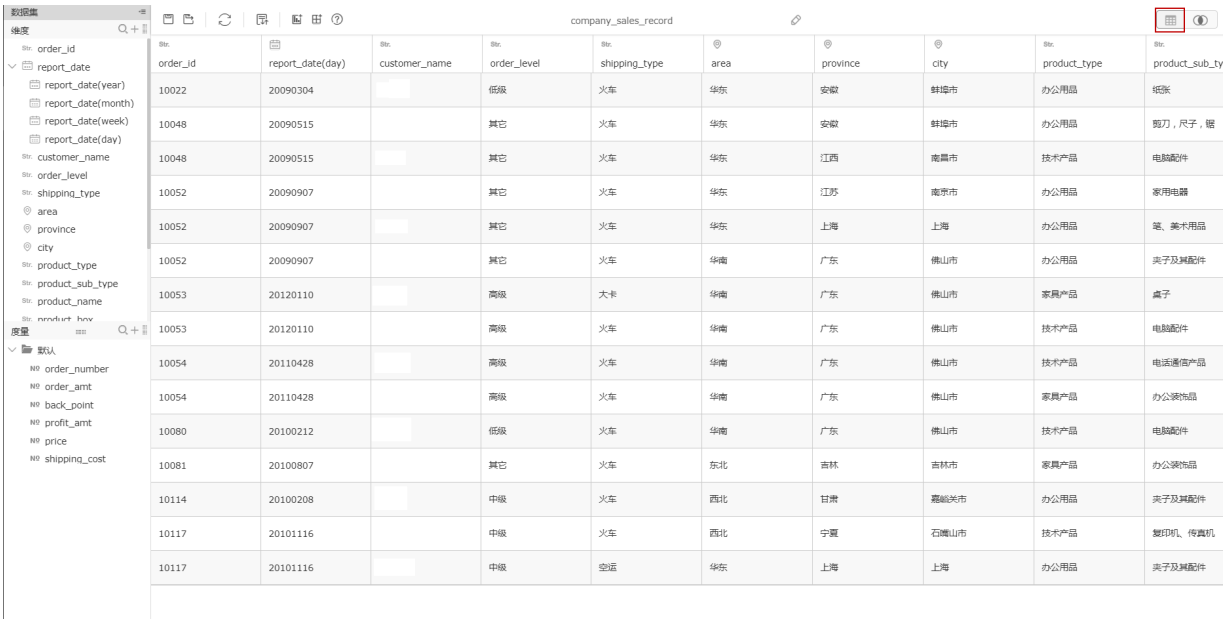
  当您在数据集编辑界面进行新建字段（度量）、删除字段、维度度量转化等操作后，需要保存数据集，进行刷新数据查看效果。
- 另存为：把当前的数据集另存为新的数据集，此操作可以用于快速复制一个新的数据集或备份数据集等场景。

31.3.2.3.4 预览数据

背景信息

单击**预览**图标，可切换到数据预览模式，如图 *1090: 数据预览*所示。

图 1090: 数据预览



31.3.2.3.5 关联工作表

操作步骤

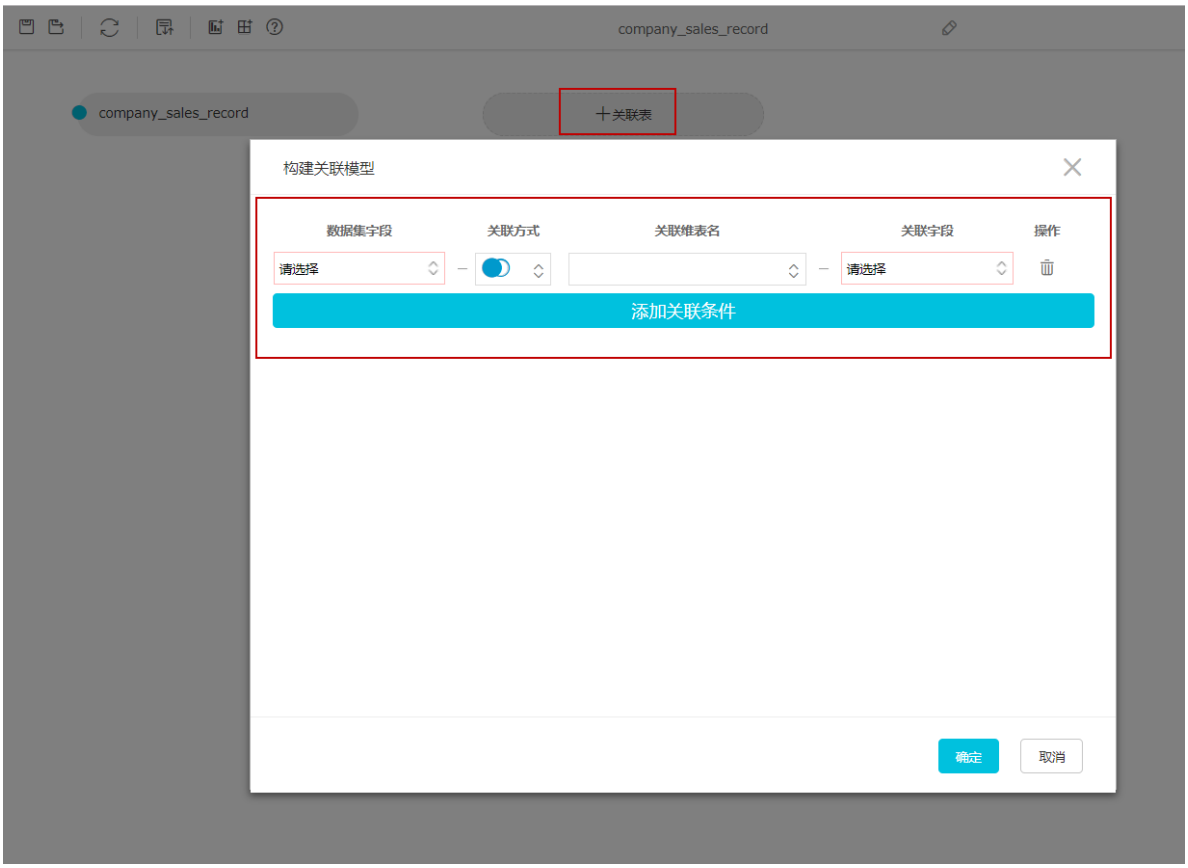
1. 单击**关联**图标，可切换到关联模式，如图 *1091:* 数据关联所示。

   图 1091: 数据关联



2. 单击**+ 关联表**按钮，打开**构建关联模型**对话框。
3. 选择需要关联的字段和关联方式，如图 *1092:* 构建关联模型所示。

   图 1092: 构建关联模型



关联方式有以下两种：

- 内关联（inner join）
- 左外关联（left outer join）

说明: 类似表 A 关联表 B，表 B 关联表 C 的模式暂不支持，同时暂不支持非同源表关联。

4. 单击**添加关联条件**按钮可关联多个表字段，如图 *1093:* 添加关联条件所示。

图 1093: 添加关联条件



5. 单击**确定**按钮，完成关联模型构建。
6. 单击**预览**图标，进入到数据预览模式。
7. 在预览模式下，单击**保存**。
8. 保存成功后，单击**刷新**，即可查看数据表关联效果。

31.3.2.3.6 关联工作表示例

背景信息

把表 mysql_date_table 和表 mysql_date_table_join 以 id 进行**左外关联**。

操作步骤

1. 在数据集列表中，找到 mysql_date_table。
2. 单击**编辑**，进入数据集编辑页面。
3. 单击**关联**图标，进入数据表关联页面。
4. 单击**+ 关联表**按钮，打开**构建关联模型**对话框。

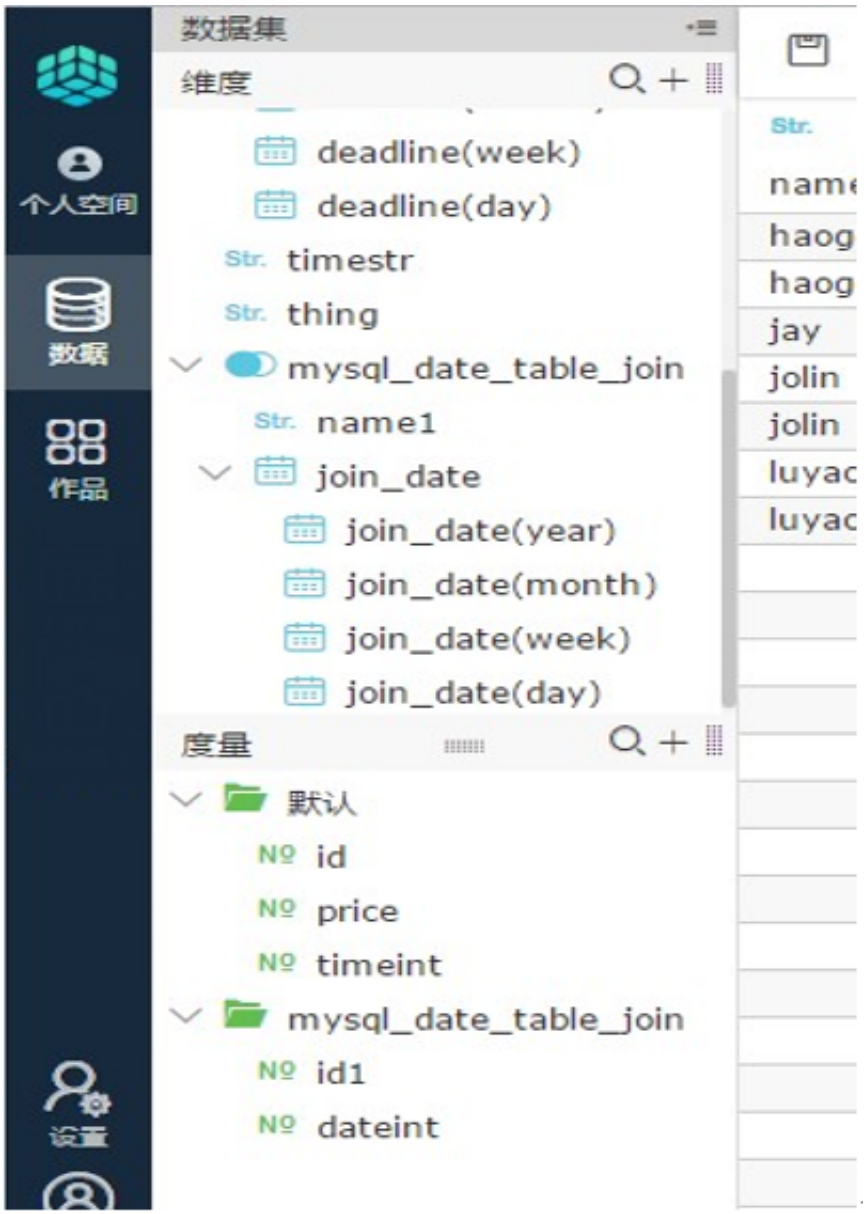
5. 单击下拉箭头，选择需要关联的字段和关联方式，如图 *1094:* 数据表左外关联所示。

图 1094: 数据表左外关联



6. 单击**确定**按钮，完成关联模型构建。
7. 单击**预览**图标，进入到数据预览页面。
8. 单击**保存**图标，保存新关联的模型。
9. 单击**刷新**图标，查看关联后的数据，如图 *1095:* 数据表左外关联预览所示。

图 1095: 数据表左外关联预览



31.3.2.3.7 钻取

背景信息

以 company_sales_record 数据集为例，查询公司在全国各大区域的销售订单金额和利润金额，单击区域可以下钻到省份，单击省份可以下钻到城市。

操作步骤

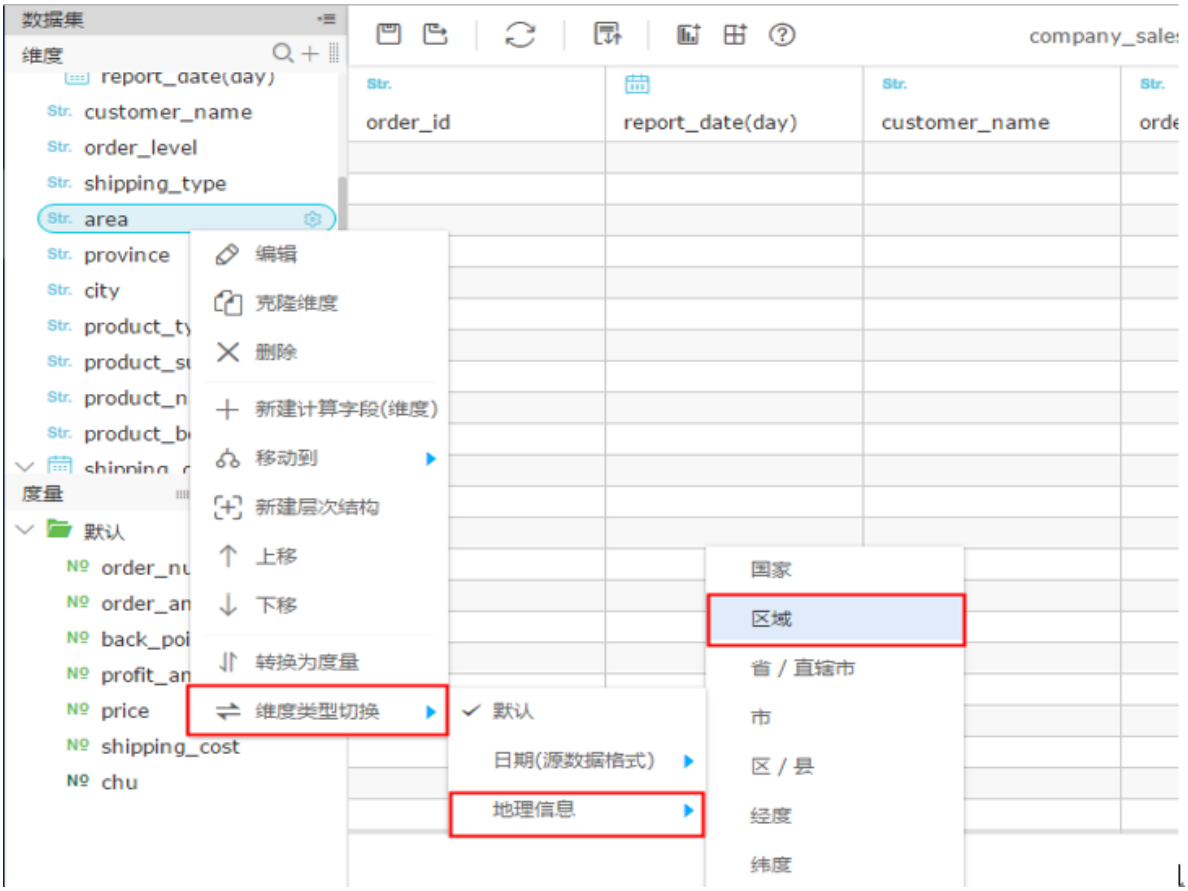
1. 在**数据源**列表中找到 company_sales_record。
2. 单击后面的**创建数据集**。

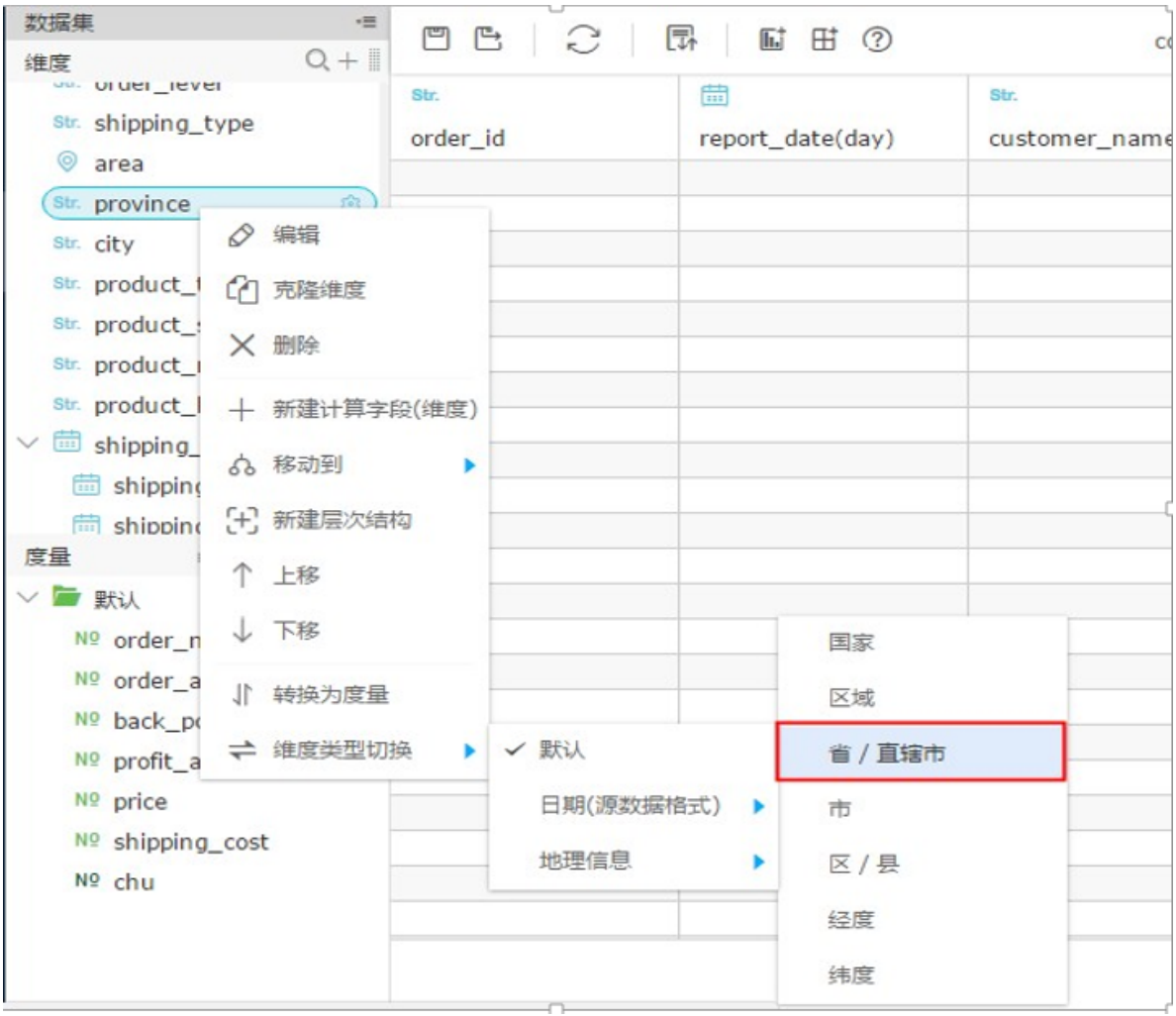
切换字段类型

3. 将普通文本字段转为地理信息。

销售记录中的 城市、省份、区域 三个字段包含了地理信息，需要将这三个字段的类型指定为地理信息类型，且必须真实的一一对应，如图 *1096: 切换字段类型*所示。

图 1096: 切换字段类型

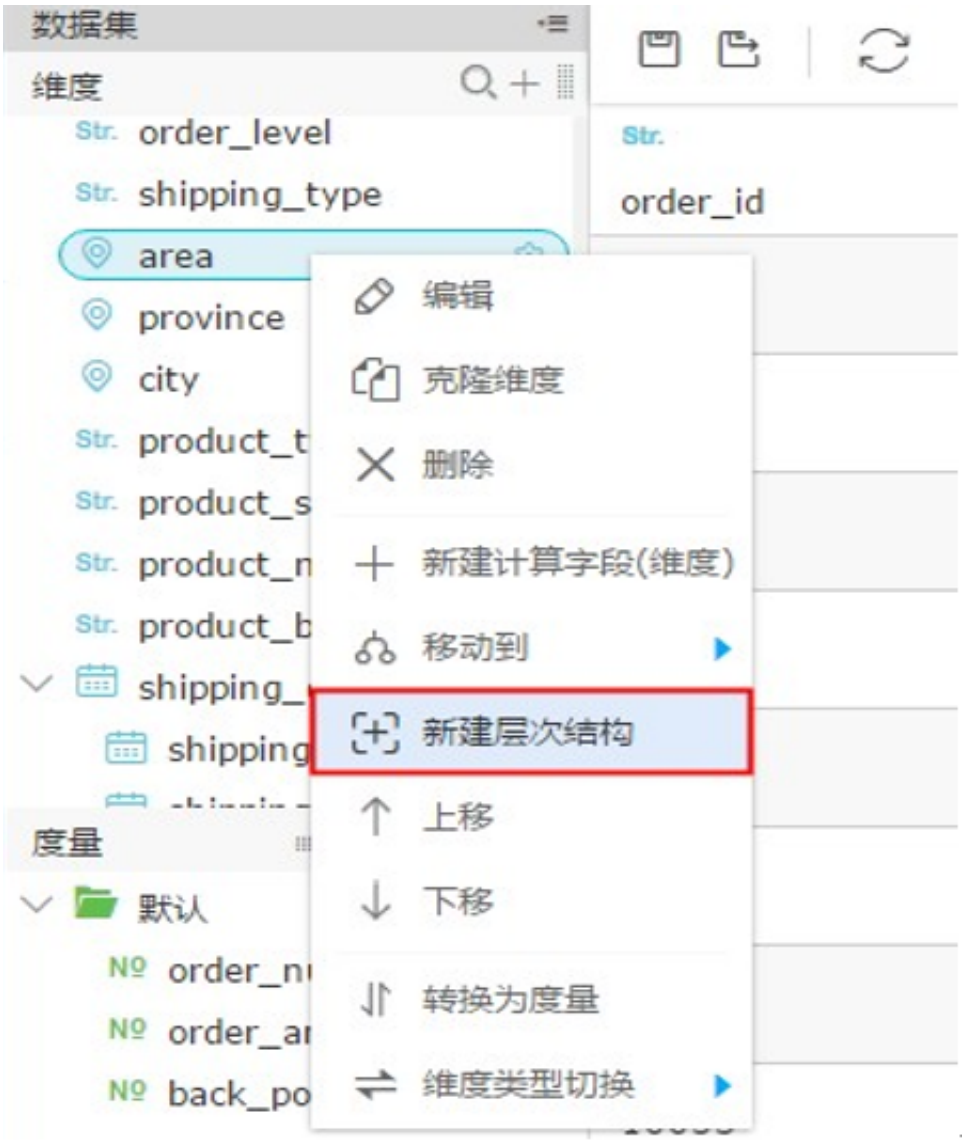




或创建具有地理信息的层次结构。

4. 在维度列表中，选择一个字段。
5. 单击鼠标右键，打开编辑菜单。
6. 选择**新建层次结构**，如图 *1097: 新建层次结构*所示。

图 1097: 新建层次结构



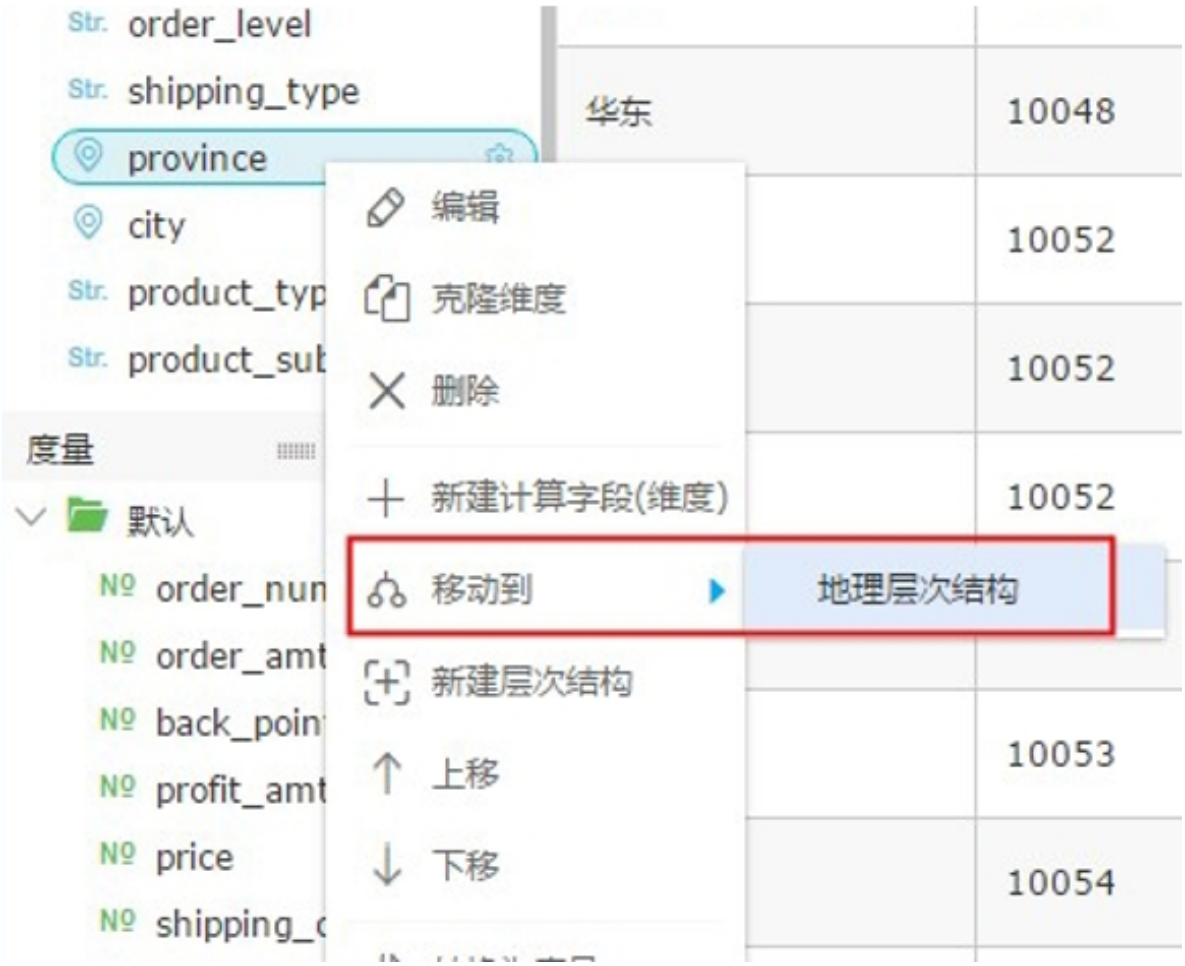
7. 输入层次结构名称，单击**确定**，如图 *1098:* 层次结构名称所示。

图 1098: 层次结构名称



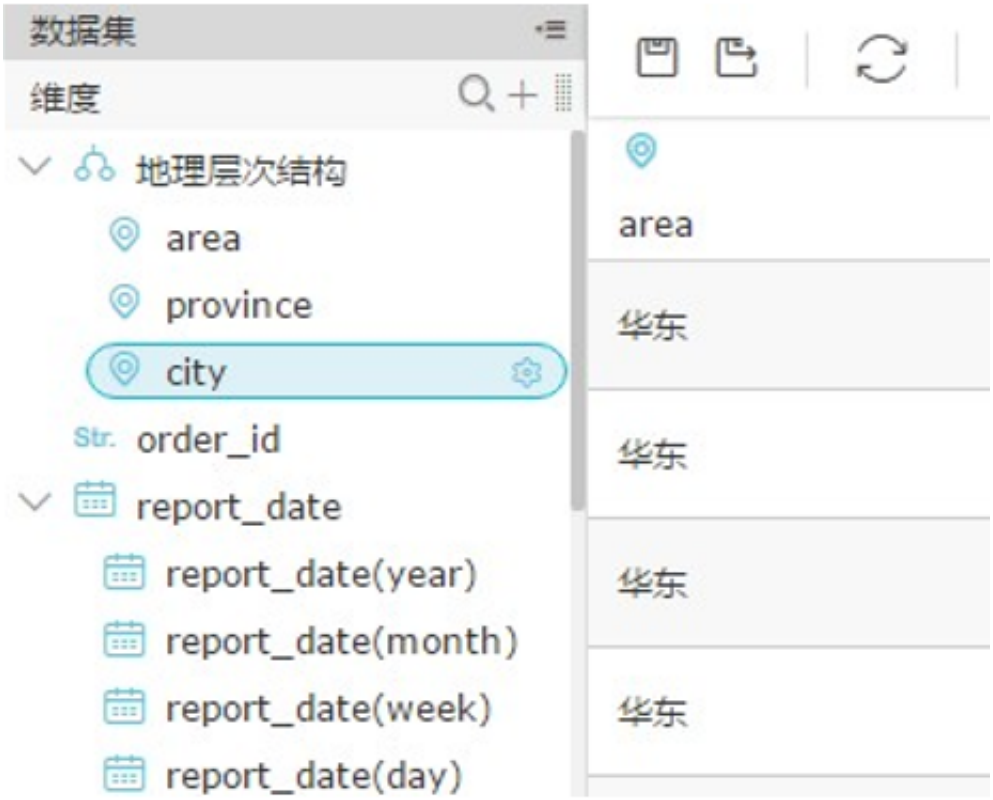
8. 将具有地理含义的维度字段移动到新建的层次结构中，如图 *1099:* 添加维度字段所示。

图 1099: 添加维度字段



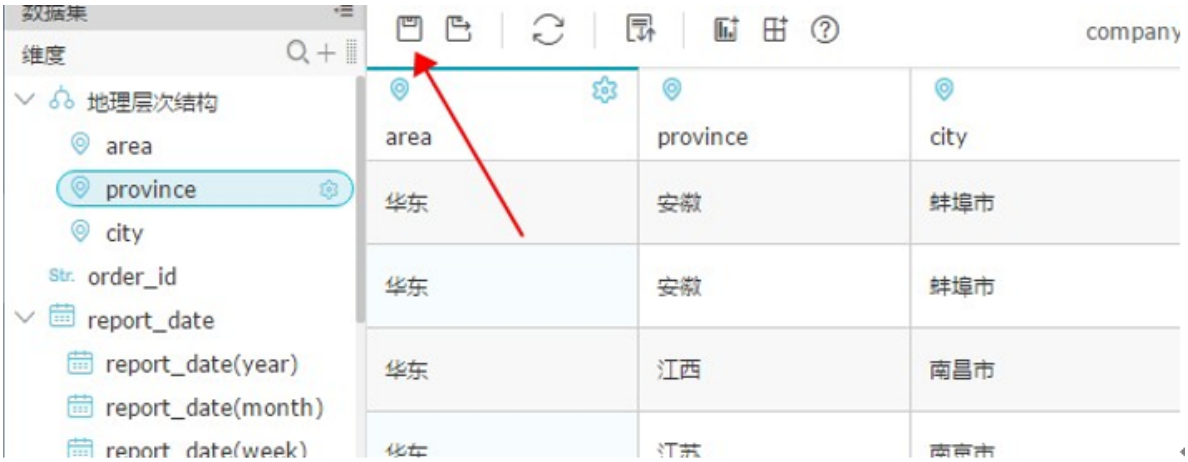
转换完成以后，维度列表字段如图 *1100:* 切换后的维度字段所示。

图 1100: 切换后的维度字段



9. 单击**保存**，保存数据集。

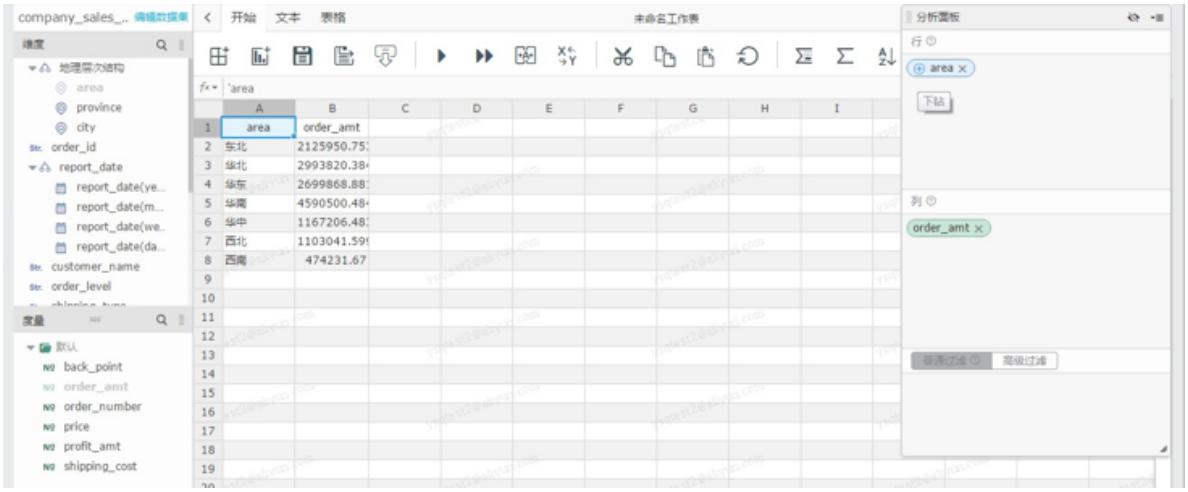
图 1101: 保存数据集



说明: 如果没有单击**保存**，数据集的编辑结果会丢失

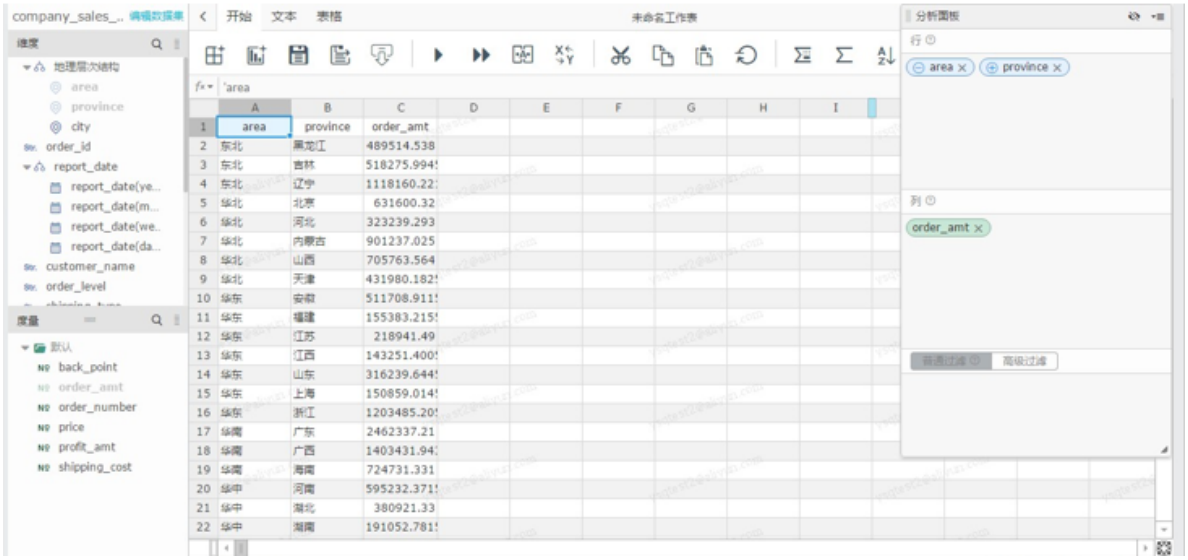
10.将 area，order_amt 字段依次添加到分析面板中，如图 *1102:* 在分析面板中添加字段所示。

图 1102: 在分析面板中添加字段



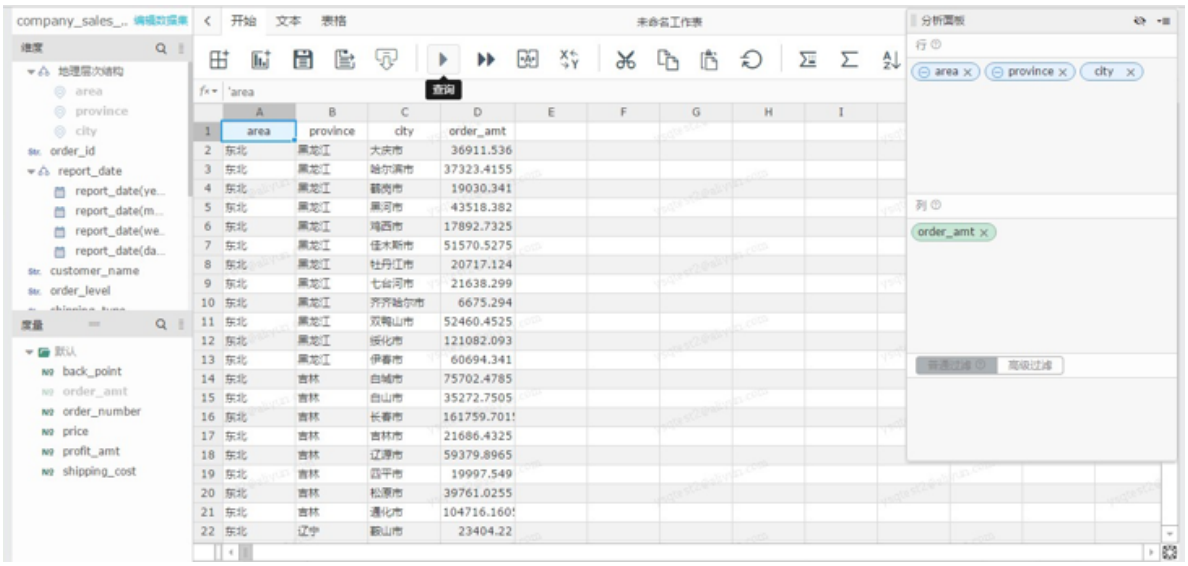
鼠标指向 area 时，左侧会出现加号，单击即可下钻各省份。

图 1103: 区域字段下钻



下钻到华北以下各省份，province 左侧同样有加号，单击后系统会自动下钻到各省份下的城市，如图 *1104:* 省份字段下钻所示。

图 1104: 省份字段下钻

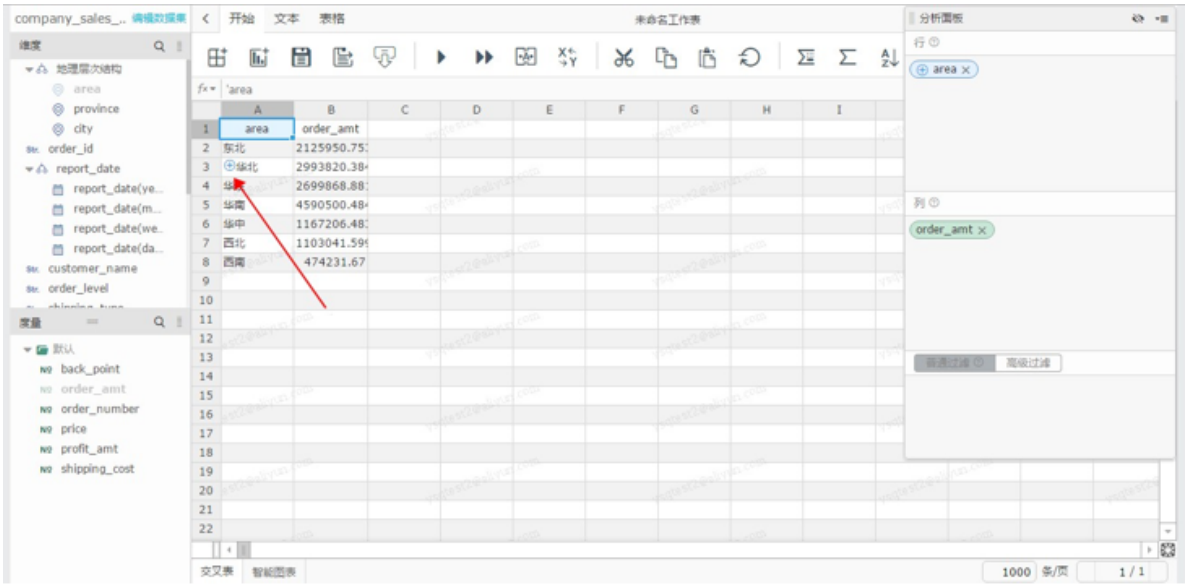


province 和 area 左侧都由加号变为了减号，指向会出现提示上钻信息，即从城市上钻到省/直辖市，或者由省/直辖市上钻到区域。

实现局部概念下的钻取。

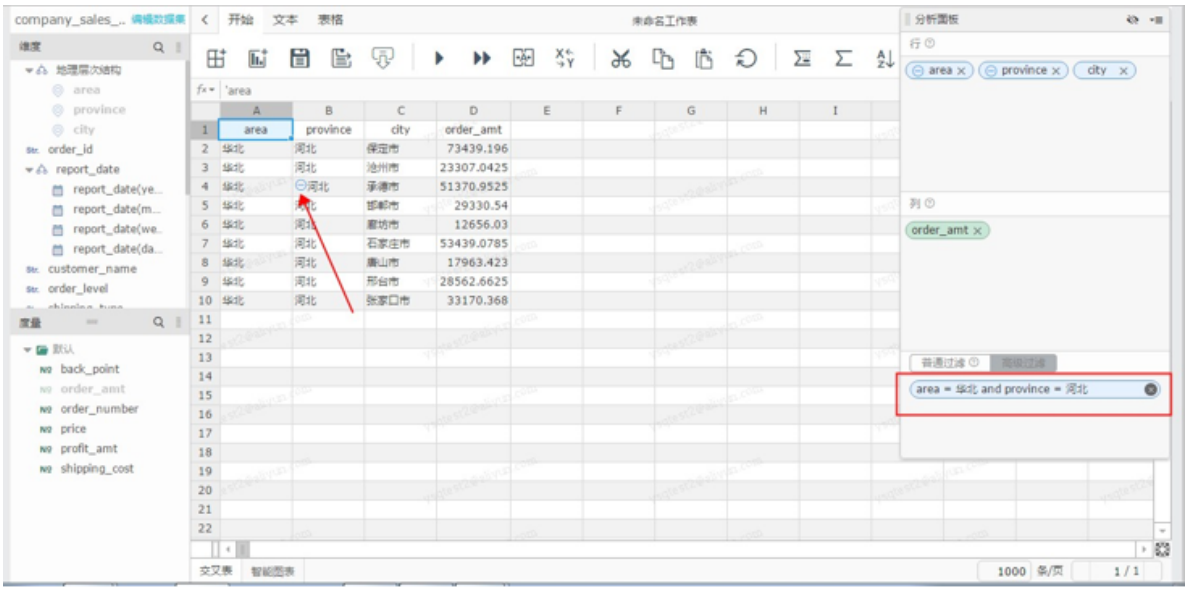
如下图红箭头指向华北区域时，左侧会出现加号，说明可以下钻到华北并下钻到各省，效果相当于添加了过滤条件：area = 华北。

图 1105: 华北区域下钻



鼠标指向河北，左侧同样会出现加号，说明可以钻取到河北省份下的各城市情况。

图 1106: 河北省份下钻



此时河北左侧的加号变为了减号，单击减号即可进行上钻。

11.单击**保存**，保存编辑好的工作表。

31.3.2.3.8 计算字段

本章节将为您介绍什么是计算字段，并且为您举例说明如何使用计算字段。

计算字段就是符合当前数据源 SQL 列定义语法规则的，用已有字段和 SQL 支持函数构造出的新的列。

如果您需要在数据源中已有的数据基础上进行计算以得到新的数值，您可以选择添加计算字段。构建计算字段的时候，支持使用业务人员也容易理解的语义化的维度或度量名称作为表达式参数，计算字段语义形式的逻辑表达式最后在 Quick BI 引擎生成真实执行 SQL 的时候，由 Quick BI 执行引擎翻译成底层的物理字段名构成的列表达式。

您可以在数据集编辑界面下，在维度栏和度量栏中单击加号，并在弹出的计算字段编辑器对话框中使用支持的函数和已有字段的组合即可。

从维度列表中新建的计算字段将自动设置为计算维度，从度量列表中新建的计算字段将自动设置为计算度量。

在计算字段的表达式编辑框中，当前所在数据源支持的函数和列表达式语法都可以使用。

函数的名称需要您手动输入。字段名也可以手动输入，格式是 [字段名]，也可以通过在英文键盘模式下输入“[”，选择列表中的字段或双击左侧维度或度量树中的节点来插入维度或度量字段名到表达式的编辑框中。正确输入的 SQL 表达式在编辑框中会自动有语法着色。

您在编写计算字段表达式的时候最容易出错的地方是：中英文引号，中英文逗号，中英文小括号等中英文标点符号混用，导致语法解析出错，事实上只允许英文的标点符号作为词法符号出现在 SQL 表达式中。如果计算字段报错，您需要首先检查是否引入了中文的符号。

在数据集编辑页面进行操作后，需要先保存数据集，再进行数据刷新。

已添加的计算字段目前不可以作为表达式再被使用在其他计算字段中。但若计算字段中所使用的原始基础字段物理层被删除，则该计算字段也将失效。

31.3.2.3.8.1 计算字段的使用规则

未聚合的计算字段可以用作维度，也可以在设置聚合方式后用作度量。已聚合的计算度量只能用作度量，不能再转为维度。

计算字段可以设置数据类型，目前支持三种数据类型：数值，文本和日期时间。

如果设置计算字段的数据类型为文本，实际内容也为文本，然后又设置其聚合方式为 sum，avg 等聚合方法之一，最后实际执行查询的时候会报告类型转换错误而无法得到查询结果。

与数据源中的原生字段生成的维度和度量相同，计算维度或计算度量也可以被使用在行列，属性面板以及过滤器中。您也可以将计算字段进行维度和度量的转换。

31.3.2.3.8.2 计算字段示例

- 求和聚合：sum([订单金额])
- 平均值聚合：avg([订单金额])
- 最大值聚合：max([订单金额])
- 最小值聚合：min([订单金额])
- 计数聚合：count([客户名称])
- 计数（去重）聚合：count(distinct[客户名称])

四则运算

- 订单成本 ([订单金额]-[利润金额])/100

字符串截取

- Substring([客户名称],1,1)

Case 度量区间分组

- 订单金额区间

Case when[订单金额] < 500 then ‘小订单’ when [订单金额] >= 500 and [订单金额] < 2000 then ‘中订单’ when [订单金额] >= 2000 and [订单金额] < 5000 then ‘大订单’ else ‘超大订单’ end

- Case 维度成员组合分组（对特定省份组合后的省份区域）

  case when [省份] in (‘黑龙江’ ,’ 辽宁’ , ‘吉林’) then ‘东北地区’ else [省份] end
- 复合聚合度量 —— 人均订单

  sum([订单金额]) / count(distinct [客户名称])
- Unix时间戳制作

  from_unixtime([订单编号]+1234567890)
- 提取每月中不同的天

  day([订单日期])

  返回 1-31区间的数字
- 提取一天中不同的小时段

  hour([订单日期])

  返回0-23区间内的数字
- 广告效果转化率

  case when sum([来访次数]) >0 then sum([转化次数])/sum([来访次数])else 0 end

  下面是转化率错误的写法：sum(case when [来访次数] >0 then [转化次数]/[来访次数] else 0 end)。因为比率型指标不能先除法运算再合计，要先合计再施加除法运算。

计算字段的表达式中可以直接使用当前数据库支持的各种函数：

- MySQL 的函数列表，适用Analytic DB

  http://doc.mysql.cn/mysql5/refman-5.1-zh.html-chapter/functions.html?spm=5176.7730345.2.2.eDKwqM
- Greenplum的函数列表

  http://www.postgres.cn/docs/9.5/functions.html?spm=5176.7730345.2.4.K8Pak6
- SQL Server的函数列表

  https://msdn.microsoft.com/zh-cn/library/ms174318.aspx?spm=5176.7730345.2.5.b6orHl

31.3.2.3.8.3 计算度量的类型

计算度量的类型有两种：普通度量和聚合度量。

没有使用聚合函数的表达式构成的度量为普通度量。使用了聚合函数的表达式构成的度量为聚合度量。

您可以使用 count ( ) 或 count (distinct) 函数将维度字段作为函数参数来构成去重聚合度量。

聚合度量的例子：人均购买金额　sum(购买金额)/countd(用户ID)；订单成本占比　sum(订单成本)/sum(订单金额)，如果用 avg(订单成本/订单金额) 计算，计算结果会出错。

普通度量和聚合度量不能混合使用，类似这样的写法是错误的：sum(订单成本)/订单金额。

普通度量，也就是不包含聚合函数的度量的聚合方式可以更改其聚合函数，聚合度量没有更改聚合函数的菜单选项了，聚合度量也不能再转为维度。

聚合度量支持的聚合函数如下：SUM，AVG，MIN，MAX，COUNT，COUNT distinct。

31.3.2.3.8.4 新建计算字段

背景信息

计算字段的使用规则和示例，请参见计算字段的使用规则和计算字段示例。

计算字段分为两种：计算维度与计算度量。其中计算度量的类型，请参见计算度量的类型。

以 company_sales_record 为例，求每份订单的平均利润。

操作步骤

1. 在 Quick BI 控制台，单击**数据**图标，进入数据管理页面。
2. 选择**数据集**标签页，找到 company_sales_record 数据集。
3. 单击后面的**编辑**，进入数据集编辑页面。
4. 在度量列表，单击加号，进入新建计算字段的对话框，如图 *1107:* 新建计算字段所示。

图 1107: 新建计算字段



5. 填写度量名称及表达式，如图 *1108:* 新建计算字段表达式所示。

   例如，求订单平均利润，那么表达式就是订单总利润除以订单数量。

   图 1108: 新建计算字段表达式



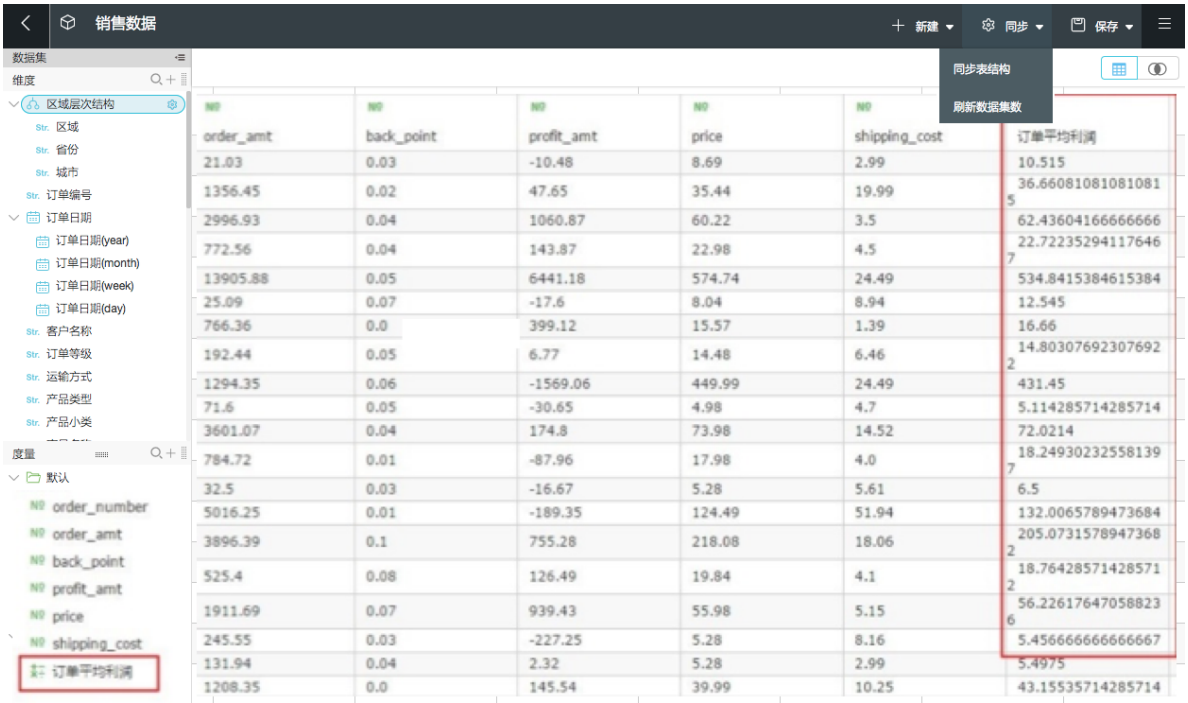
   说明: 填写表达式时，请务必使用英文输入法。

6. 选择一个数据类型。

例如，平均值为一个数值，那么在数据类型中，请选择**数值**选项。

7. 单击**确定**，完成字段新建。
8. 单击**保存**，保存数据集。
9. 单击**刷新数据集**，查看新建的计算字段，如图 *1109:* 刷新数据集所示。

图 1109: 刷新数据集



31.3.2.4 删除数据集

背景信息

如果有基于该数据集的工作表，删除该数据集的时候系统会给出错误提示，如图 *1110:* 数据集删除失败提示一所示。

删除数据集后，如果有基于该数据集所创建的仪表板，打开仪表板的时候，系统也会给出错误提示，如图 *1111:* 数据集删除失败提示二所示。

图 1110: 数据集删除失败提示一



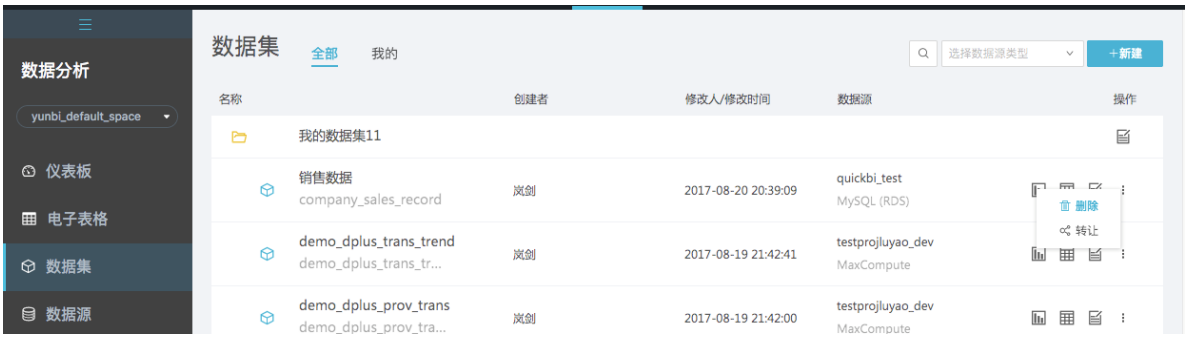
图 1111: 数据集删除失败提示二



操作步骤

1. 在**数据集**列表页面，选择一个数据集。
2. 单击鼠标右键，选择**删除**，可删除当前数据集，如图 *1112:* 删除数据集所示。

图 1112: 删除数据集



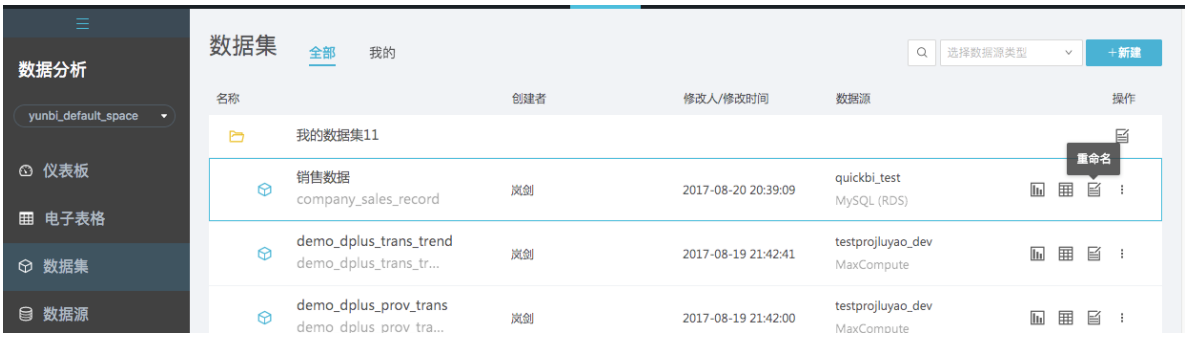
31.3.2.5 重命名数据集

背景信息

操作步骤

1. 在**数据集**列表页面，选择一个数据集。
2. 单击鼠标右键，选择**重命名**，可重命名当前数据集，如图 *1113:* 重命名数据集所示。

图 1113: 重命名数据集



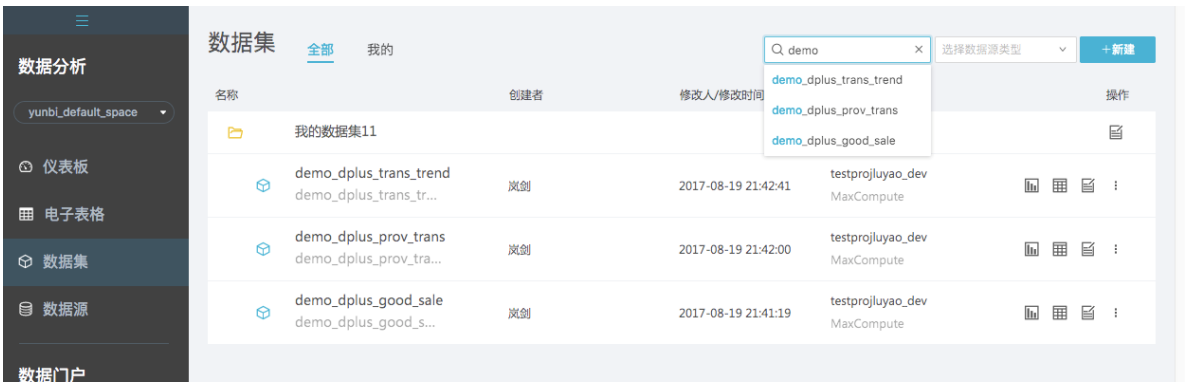
31.3.2.6 查询数据集

背景信息

操作步骤

1. 在**数据集**列表页面，找到搜索框。
2. 输入关键词，单击**搜索**图标查询数据集，如图 *1114:* 查询数据集所示。

图 1114: 查询数据集



31.3.2.7 转让数据集

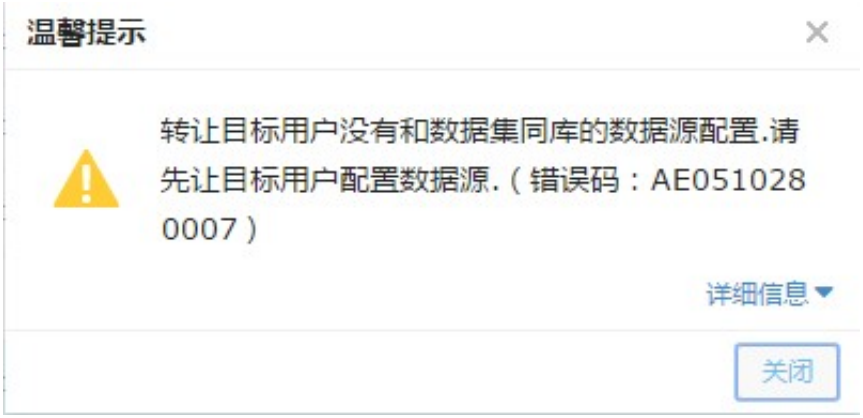
背景信息

数据集可以转让给其他阿里云账户，转让后，转让方将无法再看到该数据集。

目前 Analytic DB 数据集不支持转让。

如果被转让方没有该数据集所对应的数据源，系统会给出错误提示，如图 *1115:* 转让错误提示所示。

图 1115: 转让错误提示



操作步骤

1. 在**数据集**列表页面，选择一个数据集。
2. 单击鼠标右键，选择**转让**。
3. 输入被转让方账号，单击**转让**，如图 *1116:* 转让数据集所示。

图 1116: 转让数据集



31.3.2.8 新建数据集文件夹

背景信息

您可以在**数据集**列表页创建多个文件夹以便更好的管理数据集。

操作步骤

1. 在**数据集**列表页面，选择**我的数据集**标签页。

2. 单击鼠标右键，在下拉菜单中，选择**新建文件夹**，如图 *1117: 新建数据集文件夹*所示。

图 1117: 新建数据集文件夹



31.3.2.9 重命名数据集文件夹

操作步骤

1. 在**数据集**列表页面，选择一个文件夹。
2. 单击鼠标右键，在下拉菜单中，选择**重命名**，如图 *1118: 数据集文件夹重命名*所示。

图 1118: 数据集文件夹重命名



3. 输入新名字后，单击页面任意地方即可。

31.3.2.10 设置数据集行级权限

背景信息

通过设置数据集的行级权限，对于某些特殊的有比较大权限的组织成员，极大降低了权限管理员对其权限的维护工作量。

行级权限的控制可在数据集上进行。

操作步骤

1. 在**数据集**列表页中，选择一个数据集。
2. 单击该数据集后面的省略号，打开编辑菜单。

3. 单击 **行级权限**进入数据集行级权限控制对话框，如图 *1119:* 行级权限控制所示。

图 1119: 行级权限控制



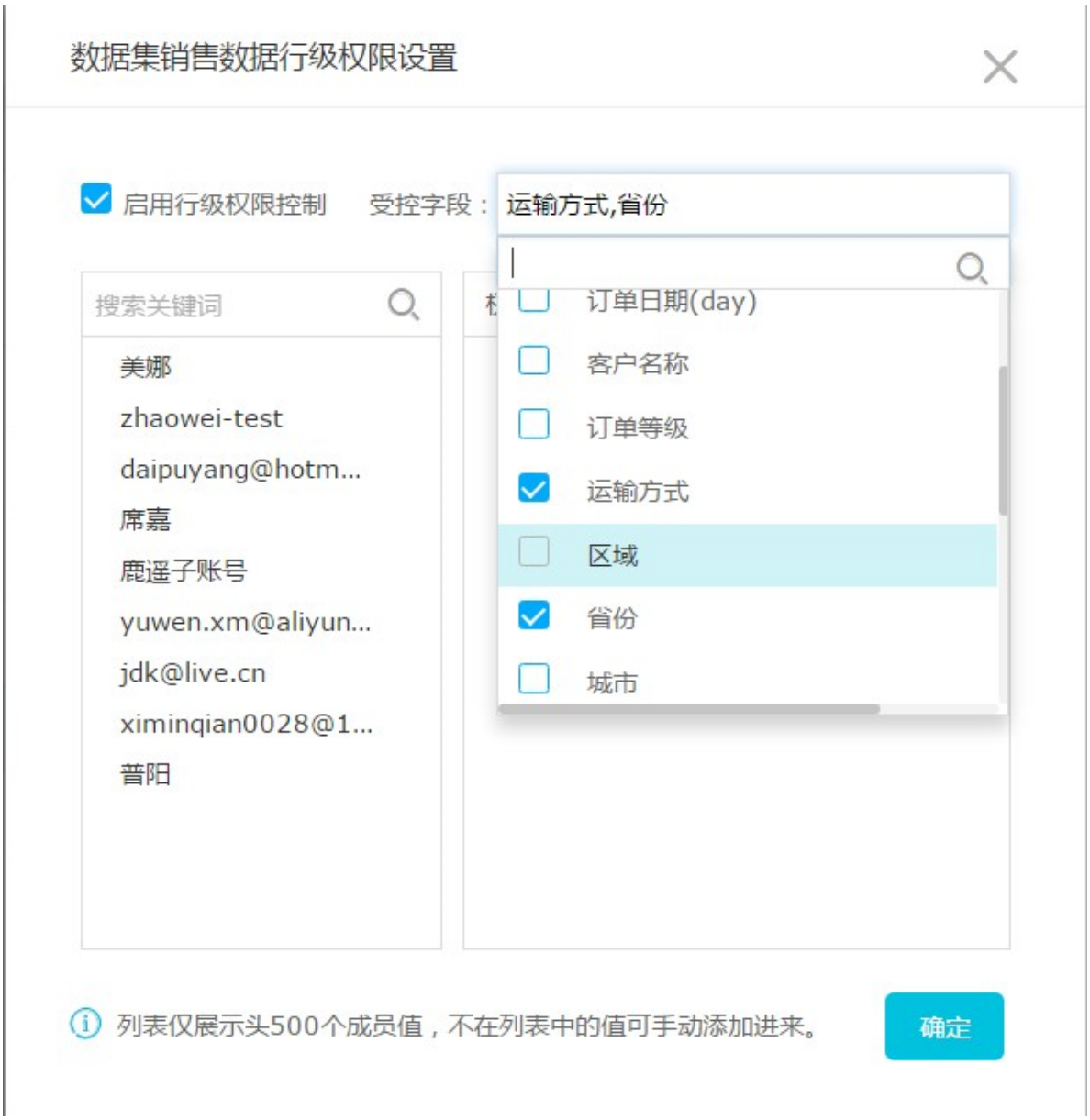
4. 选择数据集上需要受控的字段，如图 *1120:* 选择受控字段所示。

不是数据集中的所有字段都需要进行行级权限控制。根据业务需求，选择需要进行行级别权限控制的字段。例如在维度列表中，选择运输方式和省份字段进行控制。

数据集是由维度和度量构成的，所有度量的列表构成一个特殊字段：度量值。

度量值的成员就是数据集中的所有度量指标。通过对度量值字段施加控制，能够实现不同的用户能看到的度量指标也都不一样。

图 1120: 选择受控字段



5. 在受控的字段下，为不同的用户指定其有权限访问的字段。

   某个数据集上哪怕只要有一个字段要进行行级权限控制，就需要为组织中所有的成员在该数据集的受控字段上指定其有权限访问的字段成员列表，如果不指定，则默认该成员访问该数据集生成的任何数据报表都将返回没有权限无数据可阅览。

   下面展示下为两个不同用户进行数据行级权限设置的过程，但是实际中需要为所有用户都进行行级权限设置。

   - 为用户1设置行级权限。

     图 1121: 为用户1设置行级权限



- 为用户2设置行级权限。

  图 1122: 为用户2设置行级权限：



列表选择中仅展示该字段下头500个的成员值，如果该字段下的成员值个数超过500，且搜索不到某个成员但该成员又是真实存在的，那么这些成员值可通过手动操作添加进来。

在成员列表选择框中，有一个特殊的成员：**所有**。如果将这个成员赋予组织中的某位成员，则该成员就在该字段上不再受行级权限的限制，哪怕该字段未来的成员有增加或减少都不受影响；选择该字段的**所有**成员项后，该字段的其他成员项的选择就不再有约束效果了。

6. 验证行级权限控制效果，如图 *1123:* 控制效果所示。

图 1123: 控制效果



如果某组织成员在受控字段上没有被分配到可查看的权限，则报表将无法执行，并且系统会提醒该成员在某受控字段上没有权限，如图 *1124:* 无权限提示所示。

图 1124: 无权限提示



31.4 管理仪表板

本节将为您介绍如何使用仪表板，对具体的数据集进行数据筛选和查询，从而利用不同的数据图表对数据做可视化展示以及动态数据查询。

31.4.1 仪表板简介

以下章节将为您介绍仪表板的基本概念、仪表板中的数据图表类型和使用场景以及图表的数据要素。

31.4.1.1 仪表板特点

Quick BI 提供了功能强大的控件，通过拖拽控件的方式完成各类产品的页面制作。

同时，Quick BI 还提供了丰富的仪表板组件，您可以通过设置各类图表的构成要素就可以轻松完成各类报表的制作。

仪表板采用了更加灵活的磁贴式布局来显示报表数据的交互。它不仅可以将数据以可视化的方式呈现，还支持通过各种数据筛选和查询，使用各种数据展现方式，突出数据中的关键字段。

从数据的展示层面看，新版仪表板通过引导，拖拽，单击字段，让数据展示得更直观明了；从数据分析层面看，通过友好的提示，提升您的交互体验。

数据在展示性能上也有了更大的提升，您可以在仪表板的编辑页面实现动态数据查询。

31.4.1.2 仪表板的优化和新增

较之上一个版本，目前的仪表板优化和新增了以下功能。

优化

- 改进了页面排版，拖拽交互，磁铁式布局等功能
- 升级优化了图表的功能，如散点图可支持1000个维度值
- 日期控件合并到了查询控件中使用
- 改善了图表的配色方案
- 提供了17种图表样式和4个控件

图 1125: 图表样式和控件



新增

- 新增了对字段的过滤筛选

  图 1126: 过滤器



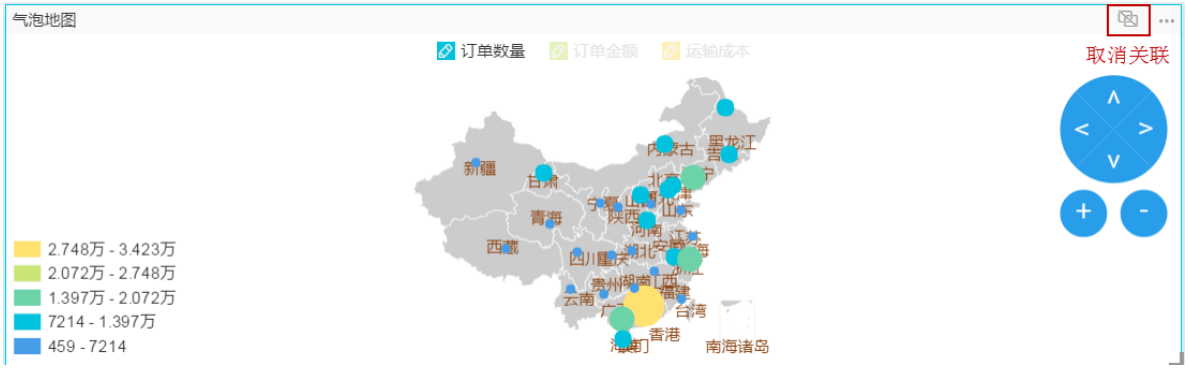
- 查询控件可支持非同源关联

  图 1127: 非同源关联



- 图表的高级联动功能新增了取消关联

图 1128: 取消关联



31.4.1.3 数据图表类型和使用场景

不同的数据需要不同的图表类型来展示，目前 Quick BI 支持17种数据图表，包括线图，柱图，气泡地图，漏斗图等。

各种图表的制作流程，请参见创建仪表板。

下表为您提供了各个图表的分析类型及其常见的使用场景。

表 112: 数据图表类型和使用场景

分析类型	说明	场景举例	可用图表
比较	对比各个值之间的差别，或者需要显示对度量按类别区分的简单比较。	比较不同国家或地区的销售/收入差别。	柱图，雷达图，漏斗图，交叉表，极坐标图，旋风漏斗，词云图
百分比	显示某个部分占整体的百分比，或某个值相对于整体的比例。	显示哪位销售人员的销售额，在总销售额中所占的比率最大。	饼图，漏斗图，仪表盘，矩阵树图
相关	显示各个值之间的关系，或比较多个度量值。	可以查看两个度量之间的相关性，了解第一个度量对第二个度量的影响。	散点图，矩阵树图，指标看板，树图，来源去向图
趋势	显示数据值的趋势（尤其是基于时间变化的趋（例如年/月/日）；或数据指标进展以及可能具有的模式。	可以查看某一个产品在一定时间范围内的销售或收入趋势。	线图

分析类型	说明	场景举例	可用图表
地理图	在地图上直观显示国家或地区的相关数据指标大小和分布范围。使用的数据集必需包含地理数据。	可以查看某一个国家，各个地区的收入情况。	气泡地图，色彩地图

31.4.1.4 数据图表的数据要素

每一个图表都配有**数据**，**样式**和**高级**三个标签页，如图 *1129:* 图表标签页所示。

图 1129: 图表标签页



- 数据标签页决定了图表展示的数据内容。
- 样式标签页决定了图表的外观和所要显示的细节。
- 高级标签页决定了数据和多个图表之间是否能形成联动，并根据您的需求动态展示数据之间的互动和对比。

每一个图表之所以能区分其它的图表而单独存在，其原因就在于核心数据要素与其它图表不同，比如地图图表，核心要素是必须有一个地理纬度字段，否则将无法在地图上展示数据。

下表展示了各个图表所需要的核心数据要素。

表 113: 图表的数据要素

图表名称	数据图表要素	数据要素构成
线图	类别轴，值轴	类别轴上最少1个维度；值轴上最少1个度量

图表名称	数据图表要素	数据要素构成
柱图	类别轴，值轴	类别轴上最少1个维度；值轴上最少1个度量
饼图	扇区标签，扇区角度	扇区标签上有且仅有1个维度，并且维度值小于等于12；扇区角度上有且仅有1个度量
气泡地图	地理区域，气泡大小	地理区域有且仅有1个维度，并且为地理纬度；气泡大小有最少1个最多5个度量
色彩地图	地理区域，色彩饱和度	地理区域有且仅有1个维度，并且为地理纬度；色彩饱和度最少1个最多5个度量
交叉表	行，列	行对维度无限制；列对度量无限制
仪表盘	指针角度，工具提示	有且仅有1个度量
雷达图	分支标签，分支长度	分支标签最少1个最多2个维度，分支长度最少1个度量
散点图	颜色图例，X 轴，Y 轴	颜色图例有且仅有1个维度，并且维度成员的数值可达1000；X 轴：最少1个最多3个度量；Y 轴： 有且仅有1个度量
漏斗图	漏斗层标签，漏斗层宽	漏斗层标签有且仅有1个维度；漏斗层宽有且仅有1个度量
指标看板	看板标签，看板指标	看板标签最多1个维度；看板指标最少1个，最多10个度量
矩阵树图	色块标签，色块大小	色块标签有且仅有1个维度，并且维度值小于等于12；色块大小有且仅有1个度量
极坐标图	扇区标签，扇区长度	扇区标签上有且仅有1个维度，并且维度值大于等于3且小于等于12；扇区长度上有且仅有1个度量
词云图	词大小，词标签	词大小有且仅有1个维度；词标签有且仅有1个度量

图表名称	数据图表要素	数据要素构成
旋风漏斗	对比主题，对比指标	对比主题有且仅有1个维度；对比指标最少1个度量
树图	树父子节点标签，树父子节点指标	树父子节点标签最少2个维度，树父子节点指标最少1个度量
来源去向图	前一页面，当前页面，后一页面；前一页面PV，前一页面UV，当前页面PV，当前页面UV，下一页面PV，下一页面UV，路径转化率，页面跳出率	所有数据要素，分别只取1个维度或者1个度量

31.4.2 进入仪表板

操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中，单击**仪表板**图标，进入仪表板列表页面。
3. 单击 **新建** > **仪表板**，进入仪表板，如图 *1130: 新建仪表板*所示。

图 1130: 新建仪表板

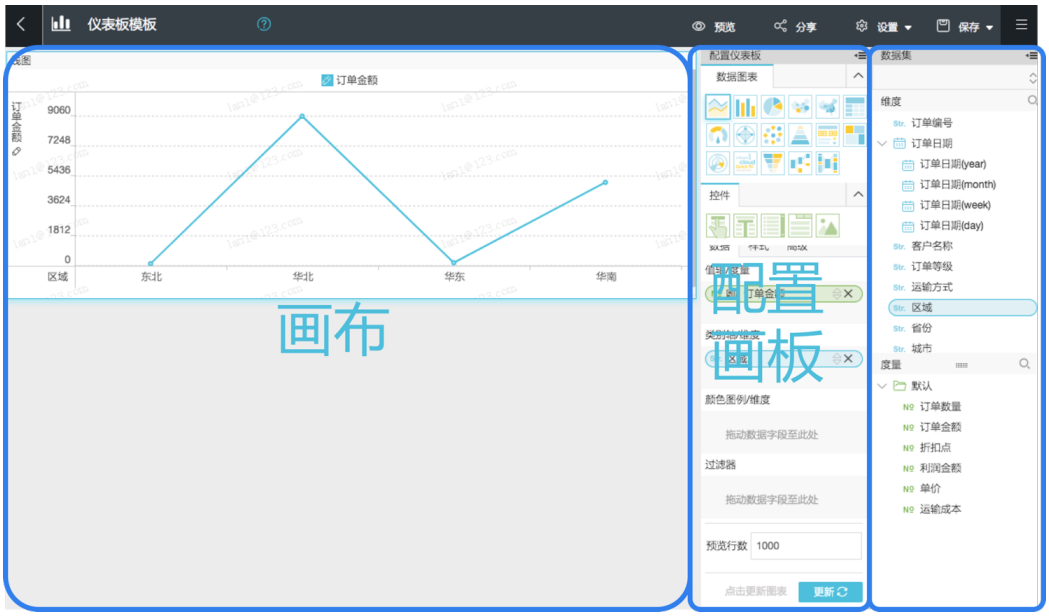


31.4.3 仪表板分区

仪表板编辑页面可分为以下三个区域，如图 *1131: 仪表板*所示。

- 数据集选择区
- 仪表板配置区（配置画板）
- 仪表板展示区（画布）

图 1131: 仪表板



- 数据集选择区：您可以在数据集选择区内切换已有的数据集，并且每一个数据集的数据类型都会按照系统的预设，分别列在维度和度量的列表中。您可根据数据图表所提供的数据要素，在列表中选择需要的维度和度量字段。
- 仪表板配置区（配置画板）：您可以在仪表板配置区选择要制作的数据图表，并且根据展示需要，编辑图表的显示标题，布局和显示图例。通过高级功能，您可以关联多张图表，多视角展示数据分析结果。您可以通过筛选功能过滤数据内容，也可以插入查询控件，查询图表中的关键数据。
- 仪表板展示区（画布）：您可以在仪表板展示区，通过拖拽的方式，随意调换图表的位置，并且还可以随意切换图表的样式，如将柱图切换成气泡地图，系统会根据不同图表的构成要素，将缺失或错误的要素信息展示给您。您还可以通过仪表板展示区提供的功能键保存，预览和新建仪表板。仪表板还提供了引导功能，让您可以自学如何制作仪表板。

31.4.3.1 数据集选择区

在数据集选择区，您可以选择或切换数据集，并对数据集的维度和度量进行搜索。

31.4.3.1.1 切换数据集

背景信息

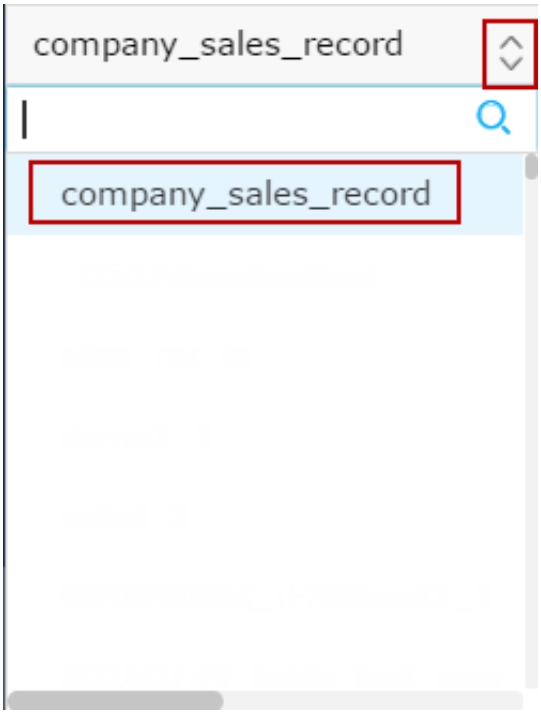
如果您在下拉菜单中找不到需要的数据集，可以单击左侧导航栏中的**数据**图标，返回到数据集列表页面，检查数据集是否创建成功。

创建数据集，请参见创建数据集。

操作步骤

1. 在数据集选择区，单击**切换**图标。
2. 在下拉菜单中，选择需要的数据集，如图 *1132:* 切换数据集所示。

图 1132: 切换数据集

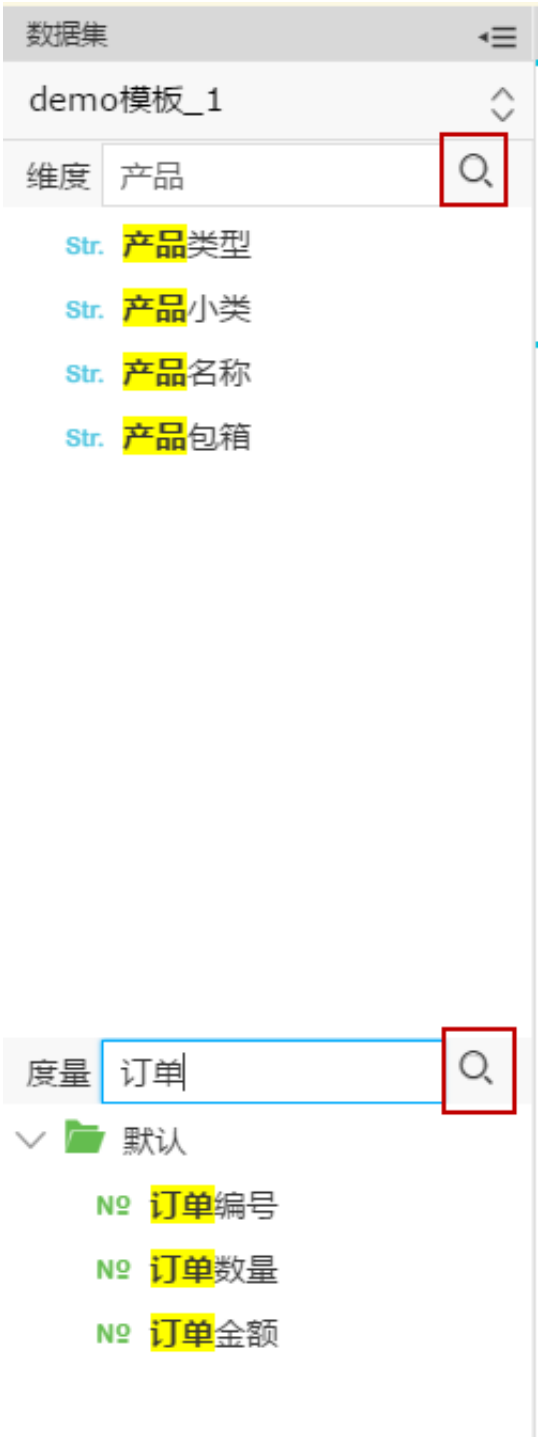


31.4.3.1.2 搜索维度字段和度量字段

操作步骤

1. 在字段搜索框中，输入字段关键字，例如产品。
2. 单击**搜索**图标，如图 *1133:* 搜索维度字段和度量字段所示。

图 1133: 搜索维度字段和度量字段



编辑维度字段和度量字段，请参见编辑维度和编辑度量。

31.4.3.2 仪表板配置区（配置画板）

您可以在仪表板配置区选择数据图表，并对数据图表进行编辑。

31.4.3.2.1 选择字段

背景信息

制作数据图表之前，请确保您已在数据集选择区选择好了数据集，并对该数据集做了需要的编辑。

编辑数据集，请参见编辑数据集。

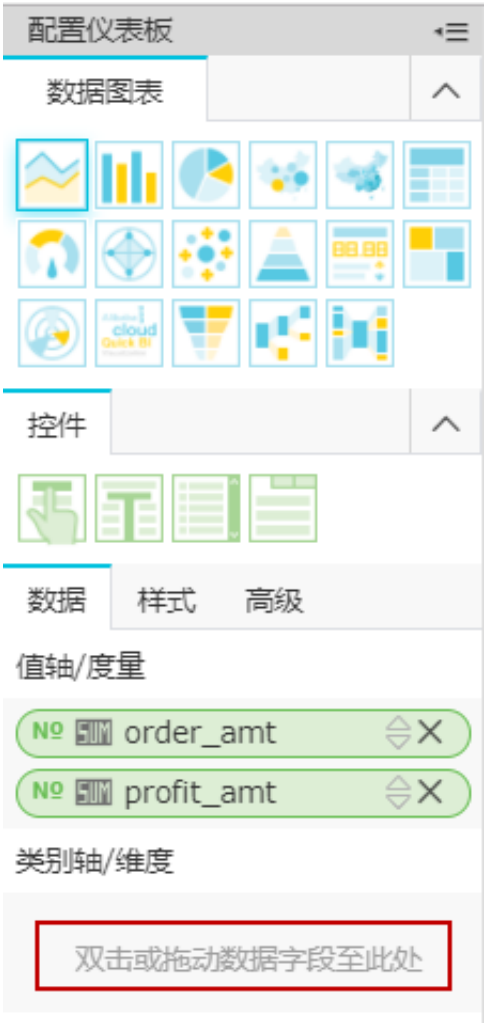
操作步骤

1. 在仪表板配置区，选择一个图表。
2. 双击该图表图标，图表会自动显示在仪表板展示区内。

   如果想切换成其它样式的图表，单击其图标即可。
3. 在**数据**标签页，选择需要的字段，如图 *1134:* 选择字段所示。

   双击选中的字段，字段会自动显示在维度和度量的显示区域，或者将选中的字段拖拽至显示区域。

   图 1134: 选择字段



- 单击字段后面的删除图标，可将字段从区域中删除。

- 单击字段后面的小三角图标，可决定字段在图表中的顺序是升序还是降序。

4. 单击**更新**，系统会自动绘制图表。

31.4.3.2.2 启用颜色图例

背景信息

颜色图例功能可以将选中的字段以不同的颜色展示在图表中。

颜色图例区域只能添加维度字段。

操作步骤

1. 选择一个维度字段，拖拽至**颜色图例**区域，例如产品类型，如图 *1135: 颜色图例*所示。

图 1135: 颜色图例



2. 单击**更新**，字段会以不同颜色分列在图表中，如图 *1136: 颜色图例效果*所示。

图 1136: 颜色图例效果



3. 单击各产品前的色块图标，可更改该产品的配色方案，如图 *1137: 图例配色方案*所示。

图 1137: 图例配色方案



31.4.3.2.3 排序

背景信息

在**数据**标签页，您可以对选定的维度和度量字段进行排序。

操作步骤

1. 选中一个字段，例如订单数量。
2. 单击字段后面向上的小三角图标，如图 *1138:* 设置排序所示。

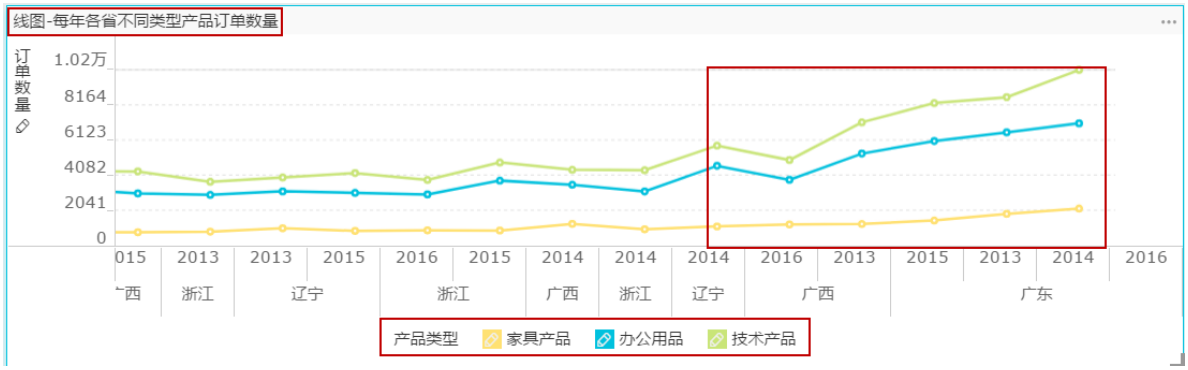
   向上的三角图标为升序，向下的三角图标为降序。

   图 1138: 设置排序



3. 选择完成后，单击**更新**，更新后的图表如图 *1139:* 排序效果所示。

图 1139: 排序效果



31.4.3.2.4 过滤字段

背景信息

任意拖动一个维度或度量字段到**过滤器**区域，可过滤字段的内容。

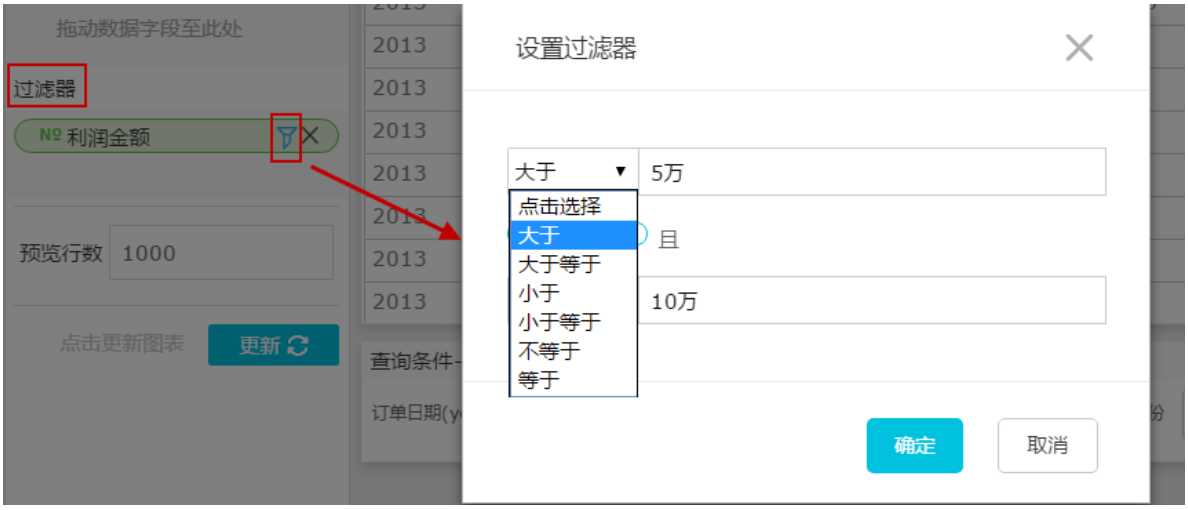
以**利润金额**为例。

操作步骤

1. 将**利润金额**字段拖拽至过滤器区域。

2. 单击**过滤**图标，在新弹出的对话框中设置过滤器，如图 *1140:* 设置过滤器所示。

图 1140: 设置过滤器



3. 选择需要的过滤条件，例如大于，小于或等于，如图 *1141:* 设置过滤范围所示。

图 1141: 设置过滤范围



4. 设置完成后，单击**确定**。
5. 单击**更新**，系统会按照过滤器的设置，将图表重新绘制。

31.4.3.2.5 多图联动

背景信息

在仪表板配置区中的**高级**标签页，可实现多图表间的联动。

启用多图联动功能时，请确保仪表板展示区内至少有两张图表可用。

操作步骤

1. 选中一个图表，例如漏斗图。
2. 在仪表板配置区，单击**高级**。
3. 在**高级**标签页的列表中，系统会自动显示出可供关联的图表，如图 *1142:* 高级标签页所示。

图 1142: 高级标签页



4. 根据源字段，在可选的图表中选择同样的字段来关联图表，如图 *1143:* 关联设置所示。

图 1143: 关联设置



如果选择错误，系统会自动给出错误提示。

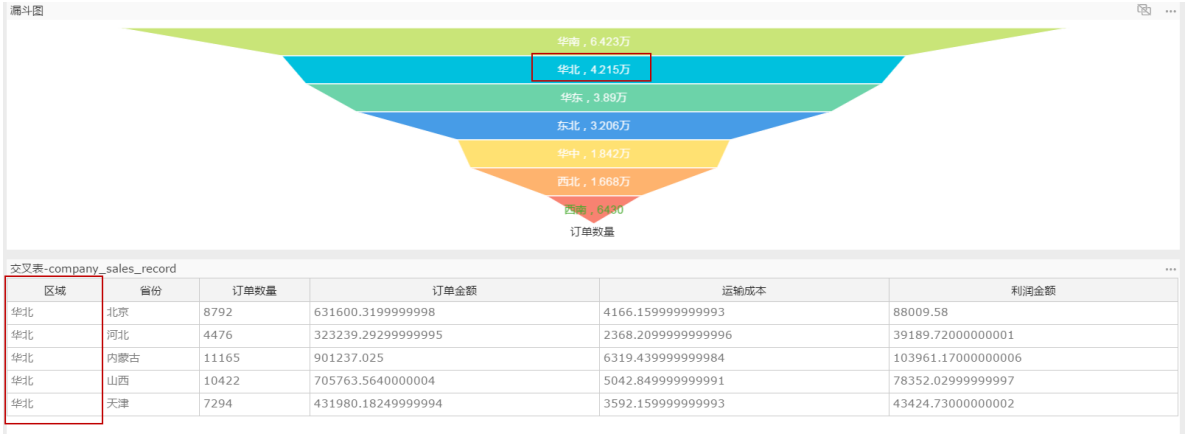
5. 在仪表板展示区上方，单击**预览**图标，进入预览页面。

图 1144: 仪表板预览图标



6. 在漏斗图中单击**华北区域**，与之关联的交叉表会自动显示出华北区域的其它数据，如图 *1145: 关联效果*所示。

图 1145: 关联效果



交叉表-company_sales_record

区域	省份	订单数量	订单金额	运输成本	利润金额
华北	北京	8792	631600.3199999998	4166.159999999993	88009.58
华北	河北	4476	323239.29299999995	2368.2099999999996	39189.72000000001
华北	内蒙古	11165	901237.025	6319.439999999984	103961.17000000006
华北	山西	10422	705763.5640000004	5042.849999999991	78352.02999999997
华北	天津	7294	431980.18249999994	3592.159999999993	43424.73000000002

7. 在图表的右上角，单击**取消关联**图标，可取消图表之间的联动。

31.4.3.3 仪表板展示区（画布）

仪表板展示区可对单个或多个图表进行以下操作：

- 管理仪表板
- 调整各个图表位置
- 查看图表数据
- 删除图表

31.4.3.3.1 工具栏

仪表板展示区提供了仪表板的保存，预览，新建等功能，如图 *1146:* 仪表板工具栏所示。

图 1146: 仪表板工具栏



31.4.3.3.2 调整图表位置

背景信息

仪表板的展示区（画布），可以展示单个或多个图表，并通过拖拽的方式，调整各个图表的位置。

操作步骤

1. 选择一个图表或一个控件。
2. 用鼠标点住该图表，并将其拖拽到指定区域。

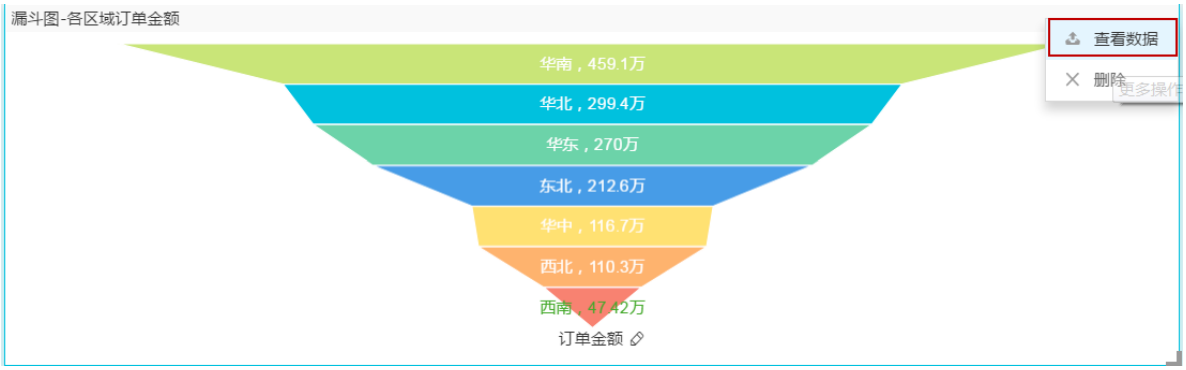
说明: 图表可拖拽的区域仅限于仪表板展示区（画布）内。

31.4.3.3.3 查看图表数据

操作步骤

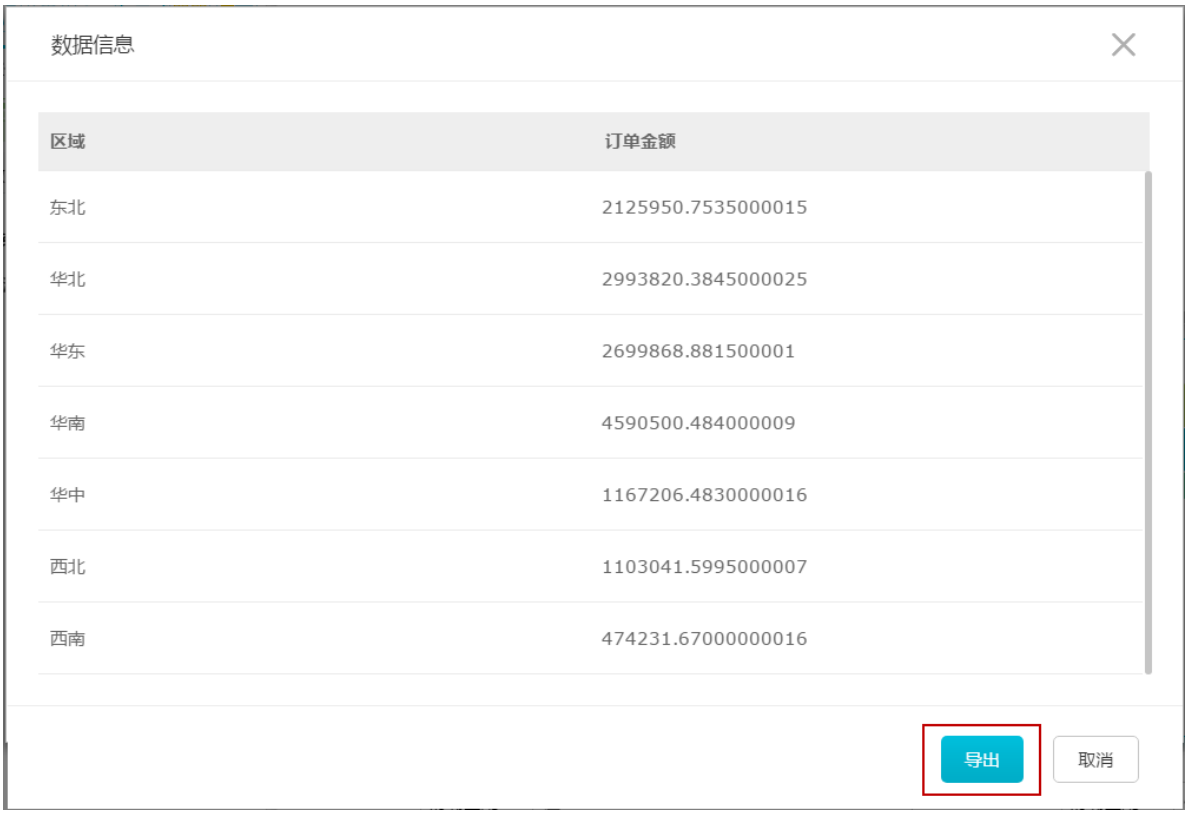
1. 选中一个图表，例如漏斗图。
2. 鼠标指向图表右上方。
3. 单击**查看数据**，如图 *1147:* 查看图表数据所示。

图 1147: 查看图表数据



4. 单击**导出**，下载数据到本地，如图 *1148:* 导出图表数据所示。

图 1148: 导出图表数据

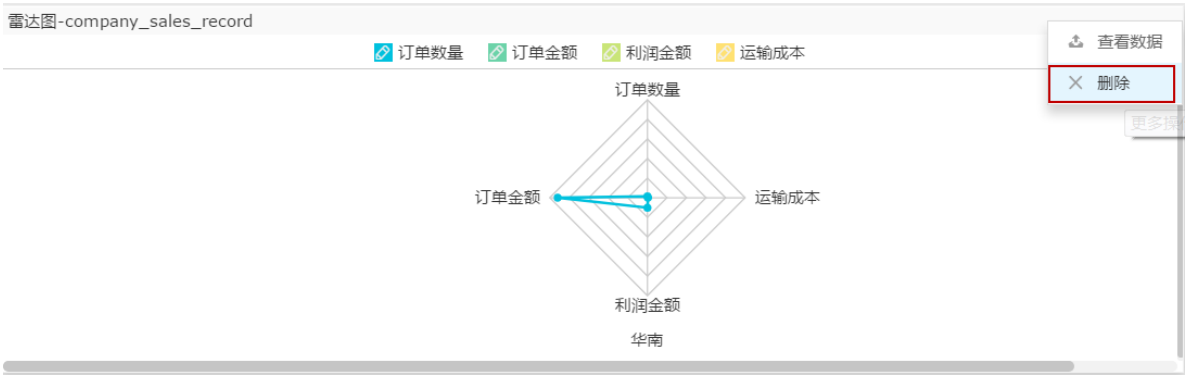


31.4.3.3.4 删除图表

操作步骤

1. 选中一个图表，例如雷达图。
2. 鼠标指向图表右上方。
3. 单击**删除**，如图 *1149:* 删除图表所示。

图 1149: 删除图表



31.4.3.3.5 切换图表

背景信息

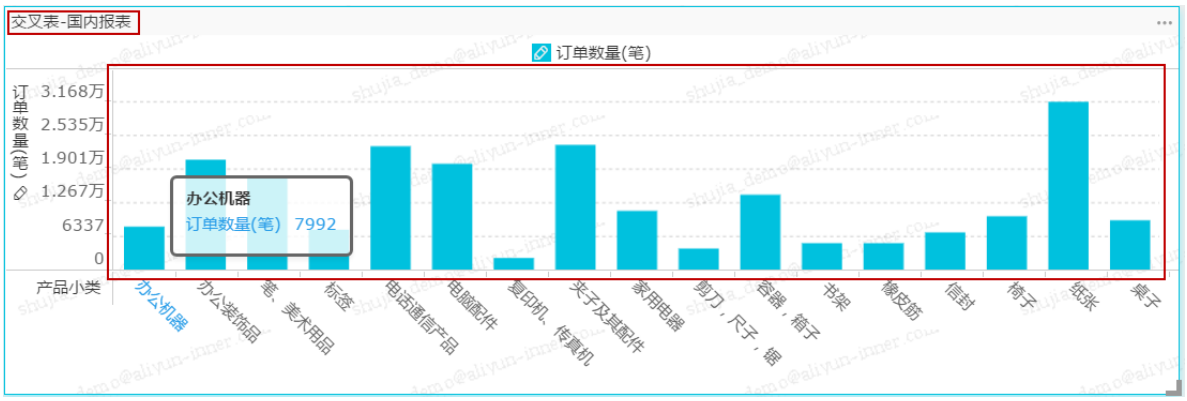
在仪表板展示区（画布），您可随意切换图表样式。

操作步骤

1. 选择一个图表，例如交叉表。
2. 在仪表板配置区（画布），选择另一个图表，例如**柱图**。
3. 单击**柱图**图标，切换图表。

   图表会根据您所选择的图表样式，自动完成切换，如图 *1150:* 切换图表所示。

图 1150: 切换图表



如果不能正常切换，说明所选择的图表要素与当前的图表要素不匹配，您需要手动调整图表要素来完成图表切换。

系统会根据图表的切换样式，来提示您需要调整的要素名称，如图 *1151: 系统提示*所示。

图 1151: 系统提示



您可以根据系统提示，手动调整维度和度量的字段，从而完成图表切换。

31.4.3.3.6 引导功能

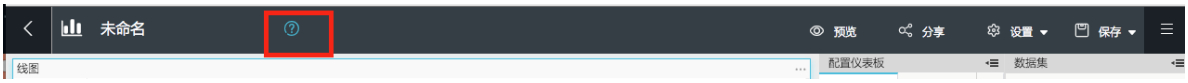
背景信息

在仪表板展示区（画布），可开启引导功能。系统会自动展示图表的制作流程，您可以根据系统的提示来学习如何制作仪表板。

操作步骤

1. 在仪表板展示区（画布），单击**问号**图标，如图 *1152: 引导图标*所示。

图 1152: 引导图标



2. 单击**上一步**，**下一步**或**跳过**自行学习如何制作仪表板，如图 *1153: 引导功能*所示。

图 1153: 引导功能



31.4.3.3.7 控件

仪表板展示区域支持以下四种控件，如图 *1154: 控件*所示。

- 查询条件
- 文本框
- IFRAME
- TAB

图 1154: 控件



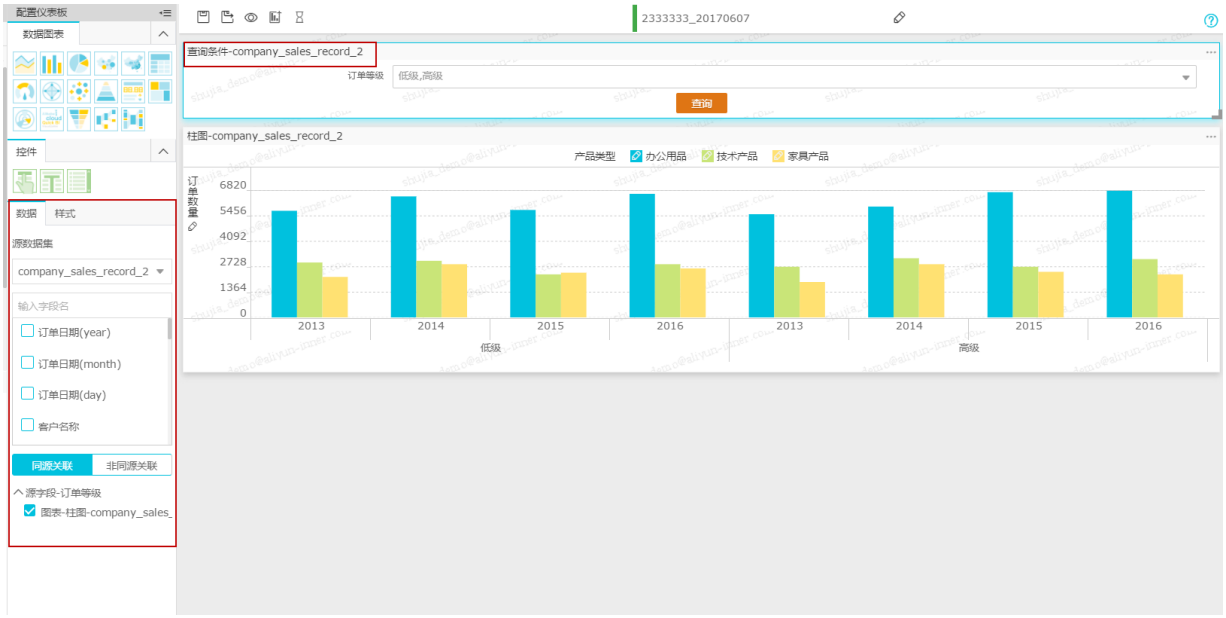
31.4.3.3.7.1 查询条件

在仪表板配置区域，您可以选择查询条件控件来查询单个或多个图表中的数据。

背景信息

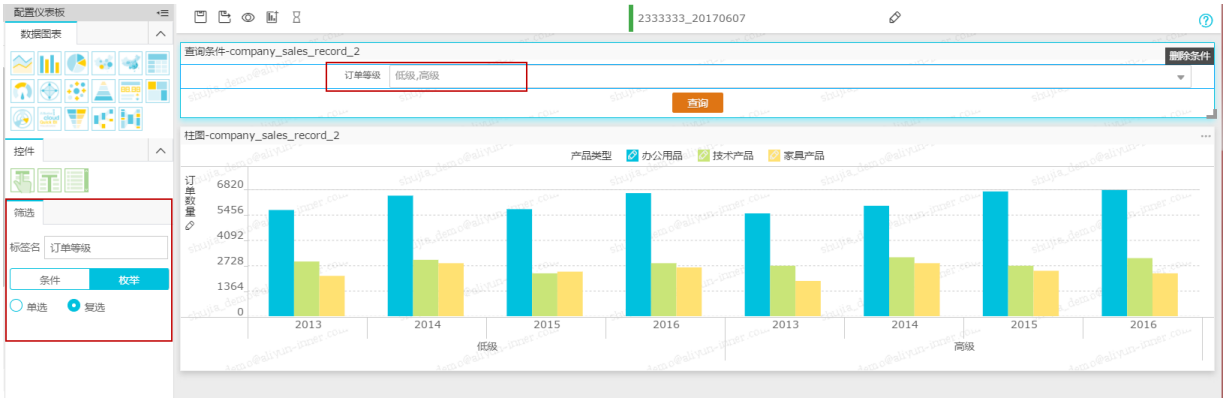
单击查询条件控件，可打开查询条件控件编辑菜单，如图 *1155: 查询条件*所示。

图 1155: 查询条件



单击查询条件中的过滤条件，可打开过滤条件的编辑菜单，如图 *1156: 过滤条件编辑菜单*所示。

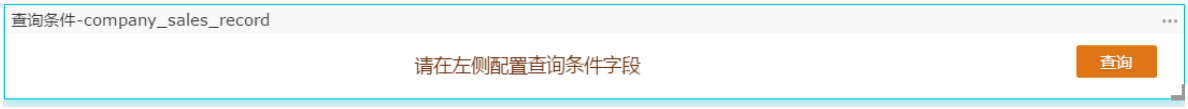
图 1156: 过滤条件编辑菜单



操作步骤

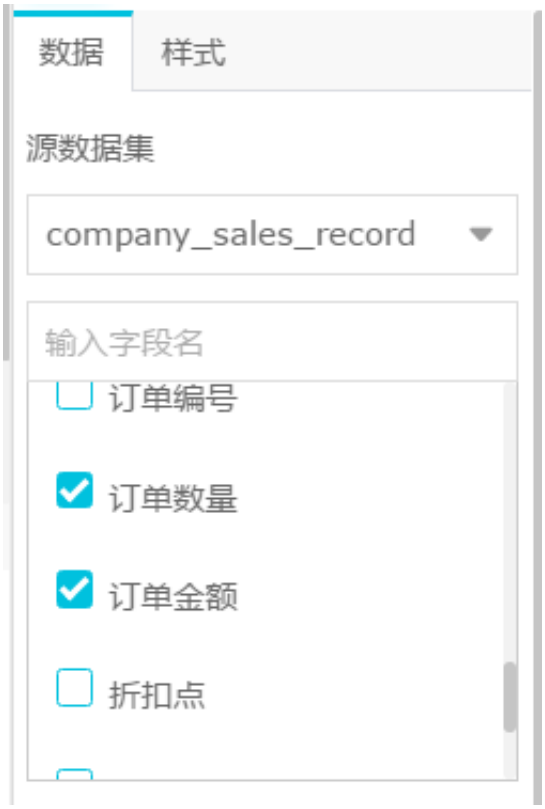
1. 双击**查询条件**图标，如图 *1157: 查询条件图例*所示。

图 1157: 查询条件图例



2. 在**数据**标签页，选择数据集和查询条件字段，如图 *1158:* 查询条件设置所示。

图 1158: 查询条件设置



目前，控件支持同源关联和非同源关联。

同源关联示例

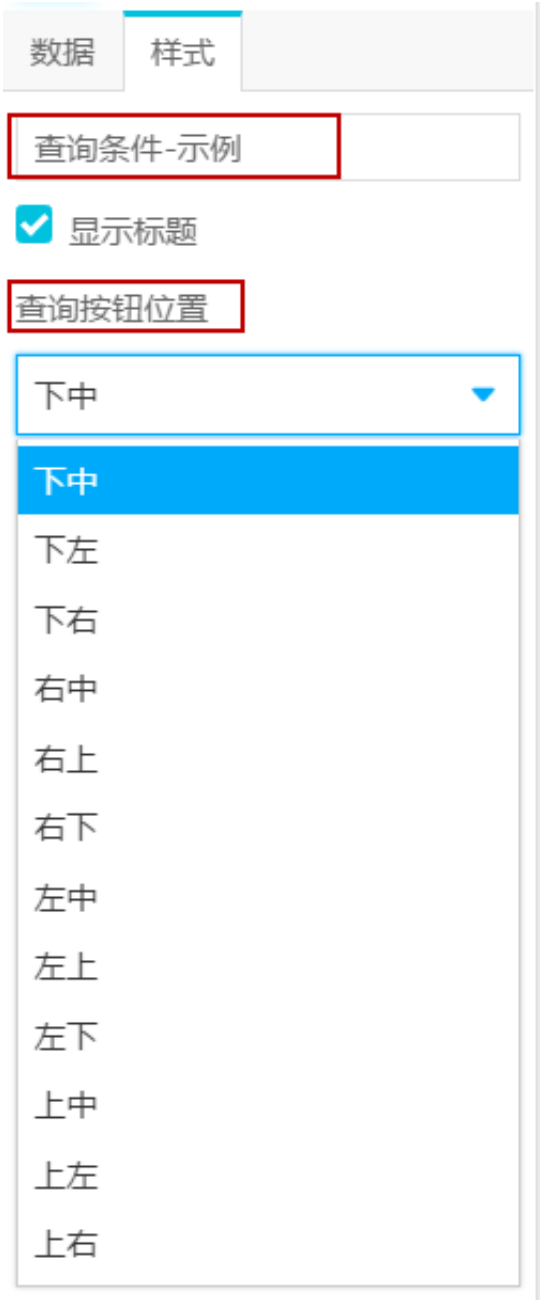
3. 选择同源关联，按字段类型选择图表，如图 *1159:* 同源关联所示。

图 1159: 同源关联



4. 单击**样式**，编辑控件的显示标题和查询按钮的位置，如图 *1160:* 查询条件编辑所示。

图 1160: 查询条件编辑



编辑完成后，控件如图 *1161:* 查询条件展示所示。

图 1161: 查询条件展示



5. 选中其中一个字段，例如**订单数量**。

6. 将订单数量的值设置为50000，如图 *1162:* 查询条件设置所示。

图 1162: 查询条件设置



7. 单击**查询**按钮，字段所作用的图表会自动更新，如图 *1163:* 查询结果所示。

矩阵树图中，订单数量不满5万的产品被自动过滤掉了。

图 1163: 查询结果



非同源关联示例

控件条件可将来自不同数据集的数据做关联，但是要确保所关联的选项中，数据成员的值是一致的，否则关联无效。

以关联不同数据集中的订单等级为例。

8. 选择一个数据集。
9. 编辑数据的维度和度量类型。
10. 选择一个数据图表，例如交叉表。
11. 选择需要的字段，将它们依次放到图表对应的行和列区域。
12. 单击**更新**，绘制图表。
13. 单击**样式**，更改交叉表的显示标题和布局。

例如，将交叉表的显示标题设置为**海外报表**，如图 *1164:* 海外报表所示。

图 1164: 海外报表

交叉表-海外报表

order_level	province	product_sub_type	product_box	order_amt	profit_amt	shipping_cost
低级	安徽	办公机器	巨型木箱	1255.48	-292.65	56.14
低级	安徽	办公装饰品	大型箱子	38.3	-42.49	6.27
低级	安徽	办公装饰品	小型包裹	1005.74	199.3	8.99
低级	安徽	办公装饰品	小型箱子	1332.48	-402.63	26.14
低级	安徽	办公装饰品	中型箱子	10627.08	958.6499999999999	176.57999999999998
低级	安徽	笔、美术用品	打包纸袋	405.76	52.28	6.3999999999999995
低级	安徽	笔、美术用品	小型箱子	679.95	-93.3	8.65
低级	安徽	电话通信产品	小型包裹	1705.6525	-177.69	9.29
低级	安徽	电话通信产品	小型箱子	12659.755500000001	2701.65	26.54
低级	安徽	电脑配件	小型包裹	370.15	-108.28999999999999	4.73

14.单击数据集切换图标，切换一个数据集。

15.编辑数据集的维度和度量类型。

16.选择一个数据图表，例如交叉表。

17.选择需要的字段，将它们依次放到图表对应的行和列区域。

18.单击**更新**，绘制图表。

19.单击**样式**，更改交叉表的显示标题和布局。

例如，将交叉表的显示标题设置为**国内报表**，如图 *1165:* 国内报表所示。

图 1165: 国内报表

交叉表-国内报表

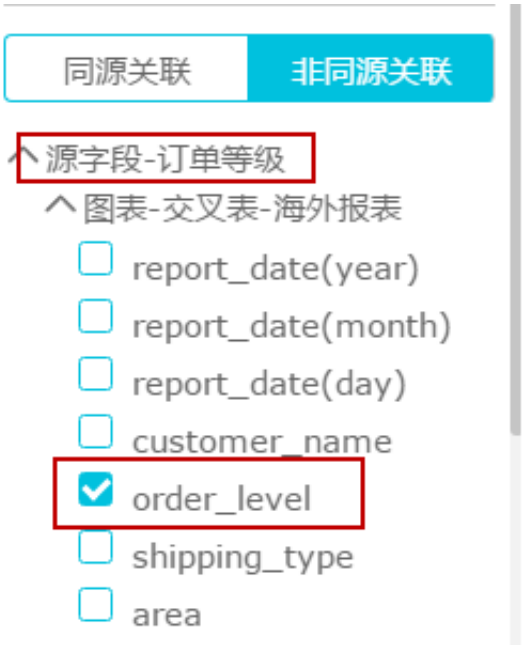
订单等级	省份	产品小类	产品包箱	订单数量	利润金额	运输成本
低级	安徽	办公机器	巨型木箱	11	-292.65	56.14
低级	安徽	办公装饰品	大型箱子	9	-42.49	6.27
低级	安徽	办公装饰品	小型包裹	47	199.3	8.99
低级	安徽	办公装饰品	小型箱子	121	-402.63	26.14
低级	安徽	办公装饰品	中型箱子	90	958.6499999999999	176.58000000000004
低级	安徽	笔、美术用品	打包纸袋	107	52.28	6.3999999999999995
低级	安徽	笔、美术用品	小型箱子	39	-93.3	8.65
低级	安徽	电话通信产品	小型包裹	47	-177.68999999999997	9.290000000000001
低级	安徽	电话通信产品	小型箱子	177	2701.65	26.54
低级	安徽	电脑配件	小型包裹	73	-108.28999999999999	4.73

20.双击**控件条件**图标，选择数据集和查询条件字段。

21.选择**非同源关联**，按字段类型选择关联项。

例如选择订单等级，如图 *1166:* 非同源关联所示。

图 1166: 非同源关联



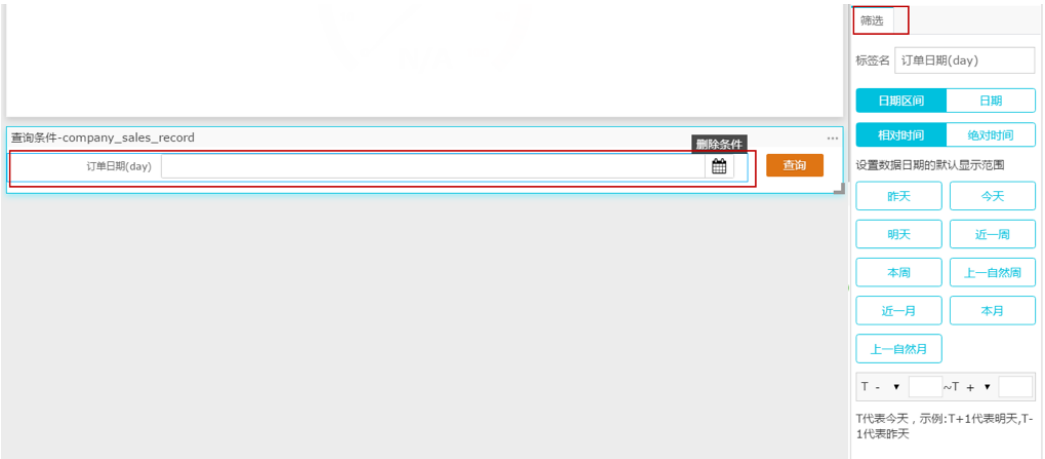
选择完成后，该控件就可以对两张来自不同数据集的图表进行查询了。

31.4.3.3.7.1.1 查询日期

操作步骤

1. 在数据页面，选择数据集和查询条件字段，例如订单日期。
2. 选择需要作用的图表。
3. 单击**订单日期**条目，打开日期筛选页面，如图 *1167:* 日期查询所示。

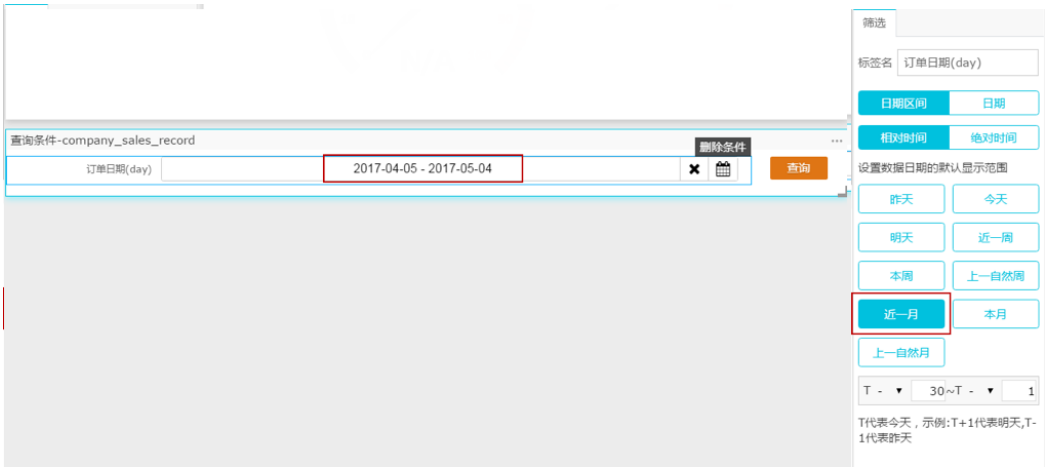
图 1167: 日期查询



4. 选择一个日期范围，例如，近一月。

   近一个月的时间区间会自动显示在控件中，如图 *1168:* 编辑日期范围所示。

   图 1168: 编辑日期范围



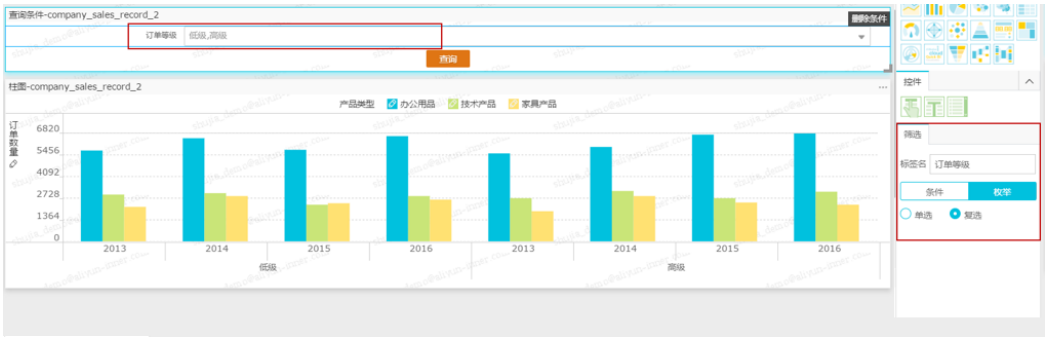
5. 单击**查询**。

31.4.3.3.7.1.2 查询文本

操作步骤

1. 在数据页面，选择数据集和查询条件字段，例如订单等级。
2. 选择需要作用的图表。
3. 单击**订单等级**条目，打开文本筛选页面，如图 *1169:* 文本查询所示。

   图 1169: 文本查询

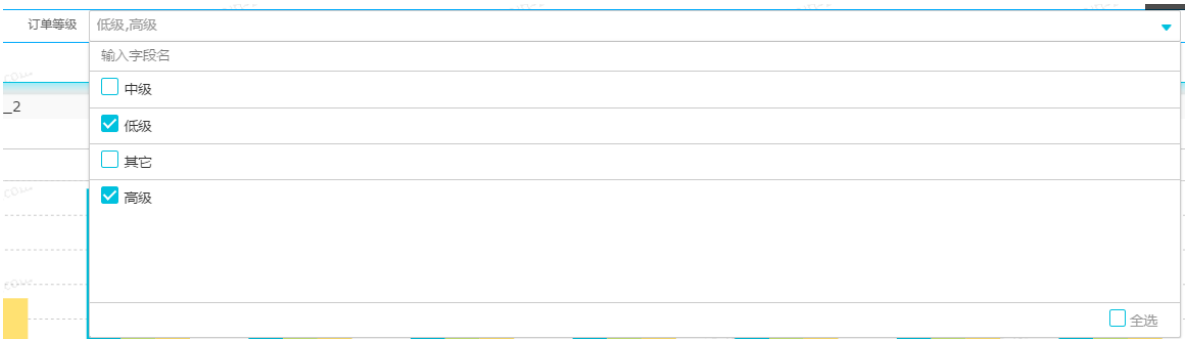


4. 选择一个文本查询条件，如枚举。

   系统会自动将订单等级中所有可选择的选项加载到查询控件上。您可以决定选项是单选还是复选。

5. 单击下拉箭头，选择需要查询的选项，如图 *1170:* 枚举所示。

图 1170: 枚举



6. 单击页面空白处。

查询条件上会自动显示出需要查询的内容。

7. 单击**查询**，如图 *1171:* 枚举查询所示。

图 1171: 枚举查询



31.4.3.3.7.2 文本框

背景信息

文本框可用于一段固定文本的输入，可用于制作报表的标题等内容。

操作步骤

1. 双击文本框图标。
2. 输入文本框内容，如图 *1172:* 文本框所示。

图 1172: 文本框



31.4.3.3.7.3 IFRAME

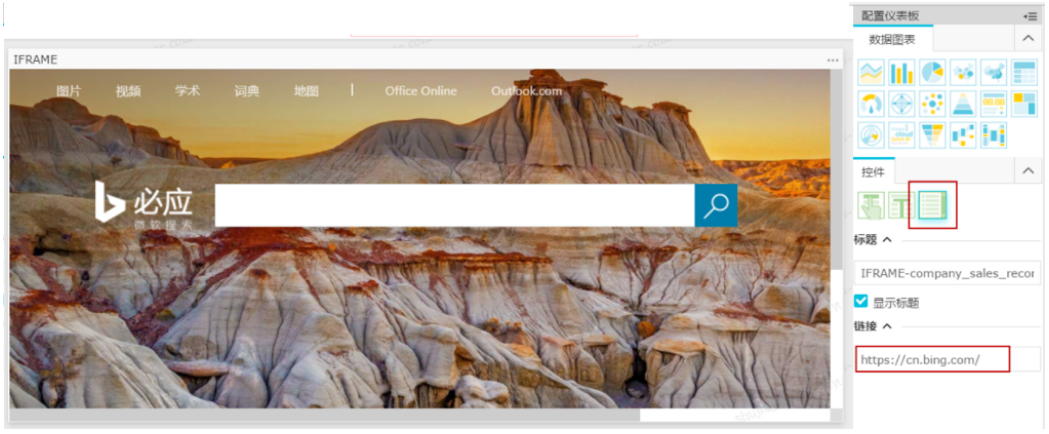
背景信息

通过 IFRAME，您可以在仪表板中插入您需要的网页，用来实时查询网络数据或浏览有关当前数据的网页或者网站。

操作步骤

1. 双击 IFRAME 图标。
2. 在链接地址区域，输入网页地址，如图 *1173: IFRAME*所示。

图 1173: IFRAME



说明: 网页地址必须以 https 的方式输入。

在标题区域，可更改当前 IFRAME 控件的显示标题。

鼠标指向 IFRAME 控件右上角，在自动弹出的菜单中选择**删除**，可删除当前控件。

31.4.3.3.7.4 TAB

背景信息

通过 TAB 功能，您可以将多张图表以标签页的形式展示。

操作步骤

1. 双击 TAB 图标。
2. 单击**新增 TAB 标签**新增 TAB 标签页数，如图 *1174: TAB* 编辑菜单所示。

图 1174: TAB 编辑菜单



3. 选择一个 TAB 页来插入图表，如图 *1175: TAB* 标签页所示。

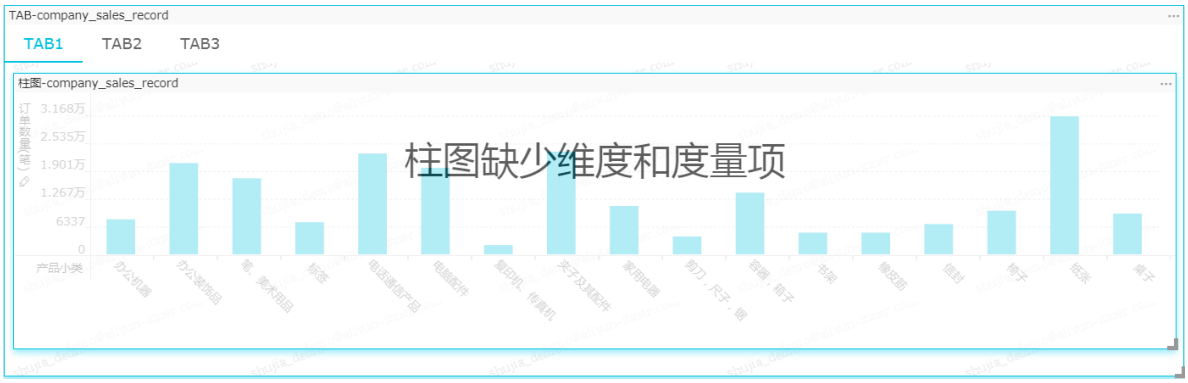
   单击 TAB1，TAB1 的颜色会被标蓝。

   图 1175: TAB 标签页



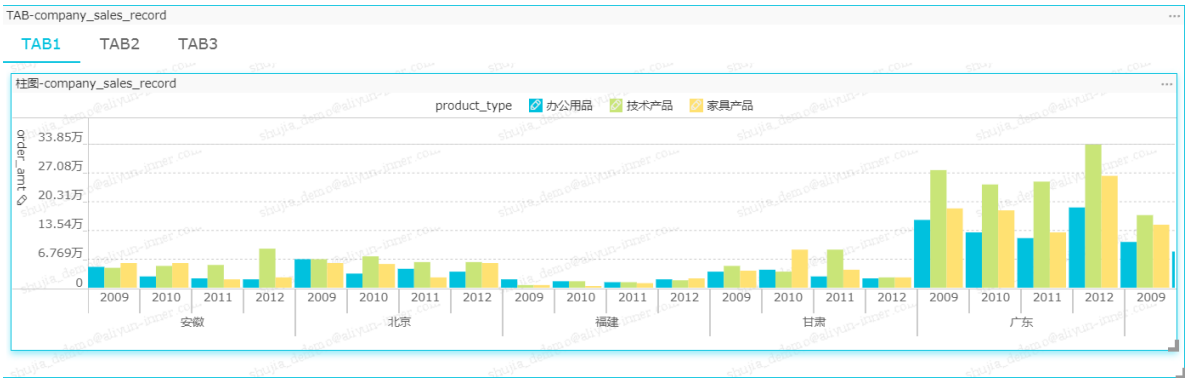
4. 双击图表图标，将图表添加到 TAB1 上，如图 *1176:* 添加图表所示。

   图 1176: 添加图表



按照图表的制作流程来制作图表，制作好的 TAB 控件如图 *1177: TAB* 标签页展示所示。

图 1177: TAB 标签页展示



如果您想删除当前控件，您可用鼠标指向图表的右上方，在自动弹出的菜单中选择**删除**，当前控件即可被删除。

31.4.4 创建仪表板

本章节将为您逐一介绍各个数据图表的制作流程。

31.4.4.1 线图

线图可以通过折线的方式显示数据的变化趋势，并且可以显示随时间而变化的连续数据，非常适合用来分析和显示在相等的时间间隔下数据的趋势走向。线图也可用来分析多组数据随时间变化的相互作用和相互影响，比如，用来分析某类商品或是某几类相关的商品随时间变化的销售数量，从而进一步预测未来的销售情况。

背景信息

线图是由类别轴和值轴构成的。类别轴沿水平分布，并且只能放置维度字段，如日期，省份，产品类型等；值轴沿垂直分布，并且只能放置度量字段，如分析对象所对应的业务指标，订单数量等。

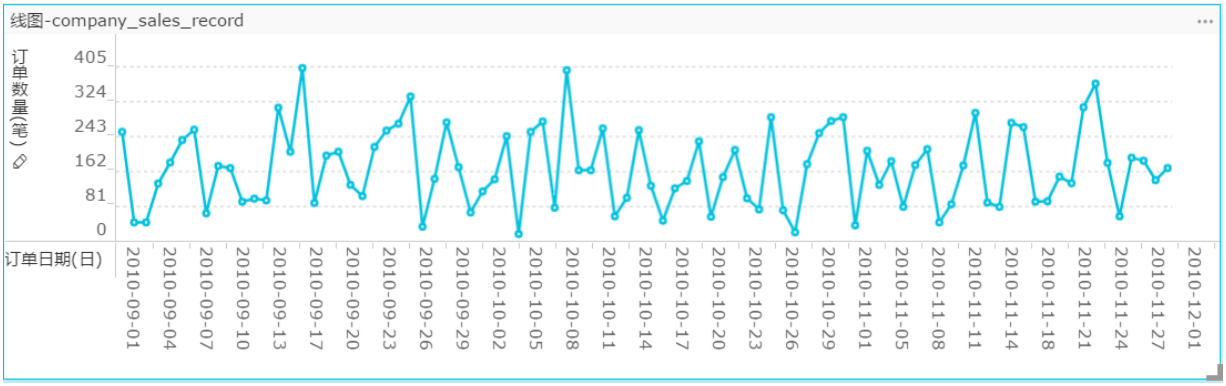
在仪表板中，系统已自动将线图的类别轴、值轴与维度字段、度量字段做好了匹配，如图 *1178: 线图的类别轴、值轴*所示，您只需要按照系统提示，在维度和度量列表中选择您需要的字段即可。

图 1178: 线图的类别轴、值轴



一张典型的线图如图 *1179: 线图示例*所示。

图 1179: 线图示例



线图的类别轴上至少选择1个维度；值轴上至少选择1个度量；如果需要用颜色图例，那么颜色图例上最多只能选择1个维度。

说明: 只有当值轴上仅有1个度量字段的时候，才可以启用颜色图例，否则，该功能将不可用。

以 company_sales_record 数据集为例，用线图展示，每年各省份各类产品的订单数量。

操作步骤

1. 登录 Quick BI 控制台。

2. 在左侧导航栏中单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**线图**图标。
5. 在**数据**标签页，选择需要的维度字段和度量字段。

   在维度列表中，找到订单日期（年），省份和产品类型，并将它们依次添加到类别轴区域中；在度量列表中，找到订单数量，并将其添加到值轴区域中，如图 *1180:* 选择线图字段所示。

   说明: 请确保省份字段的维度类型已经从字符串切换为了地理信息。

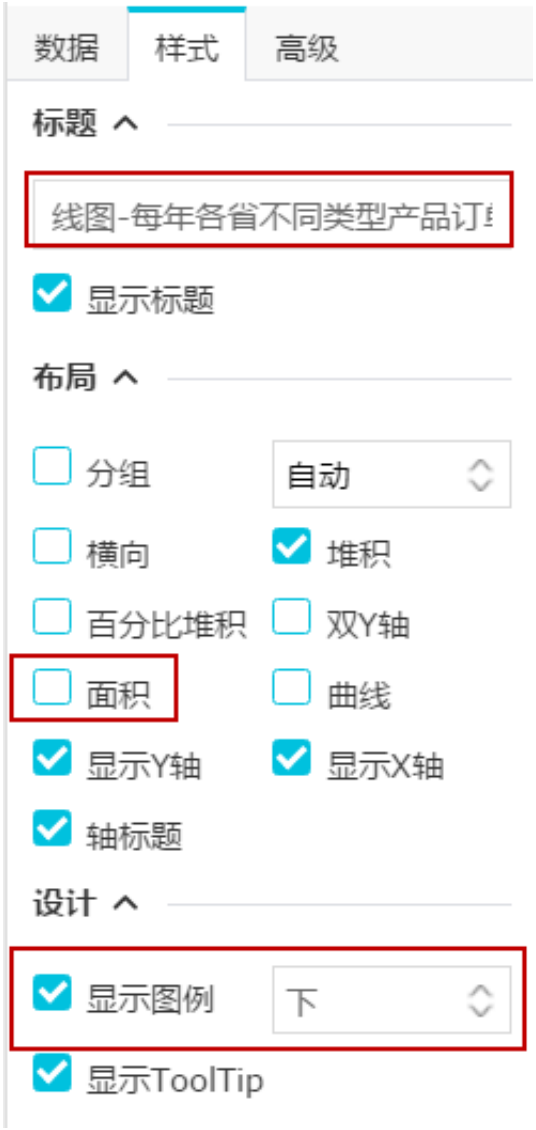
   切换维度字段类型，请参见编辑维度。

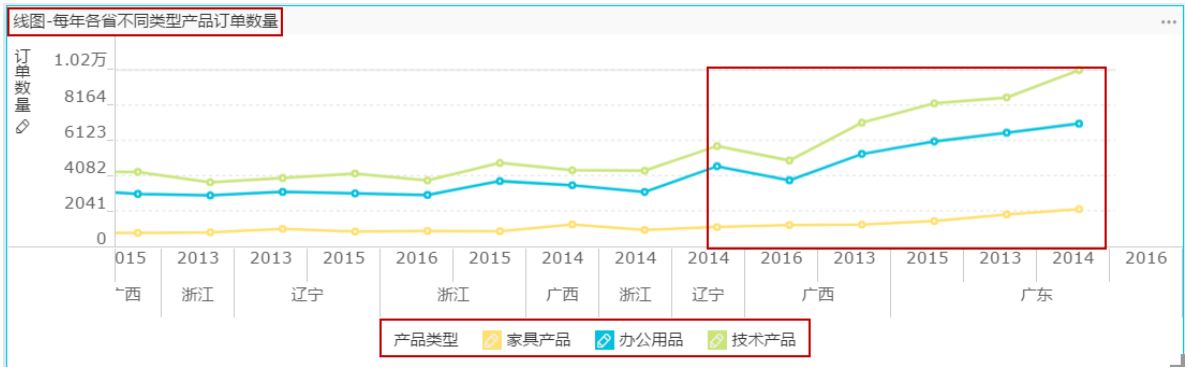
   图 1180: 选择线图字段



6. 将**产品类型**字段拖拽至**颜色图例**区域。
7. 单击**更新**，绘制图表。
8. 单击**样式**标签页，更改图表的标题，布局和显示图例，如图 *1181:* 编辑后的线图所示。

   图 1181: 编辑后的线图





9. 单击**保存**，输入一个仪表板名称。

10. 单击**确定**，保存仪表板。

    系统会默认将仪表板保存到**我的仪表板**中。

31.4.4.2 柱图

由于易于比较各组数据之间的差别，柱图可以用来显示一段时间内的数据变化或显示各项之间的比较情况，比如展示某个路口不同时间段的车流量对比。

背景信息

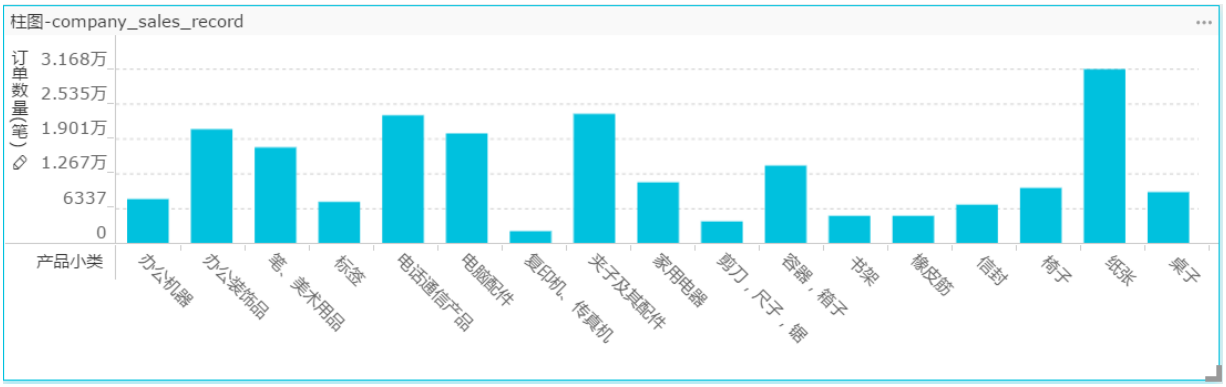
柱图的构成要素与线图类似，都是由类别轴和值轴构成。用来显示一段时间内的数据变化或显示各项之间的比较情况。

本章节将会为您介绍以下两个场景，包括如何使用过滤器和如何实现柱图的双 Y 轴显示：

- 场景一：华东各省不同产品的运输成本比较
- 场景二：各省各类产品的订单数量和平均利润金额比较

一张典型的柱图如图 *1182:* 柱图示例所示。

图 1182: 柱图示例



柱图的类别轴上至少取1个维度，如省份，产品类型等；值轴上至少取1个度量，如订单数量，利润金额等；颜色图例上只能取维度字段，并且最多取1个维度。

> **说明:** 只有当值轴上仅有1个度量字段的时候，才可以启用颜色图例，否则，该功能将不可用。

场景一：以 company_sales_record 数据集为例，用柱图展示，华东各省不同产品的运输成本比较。

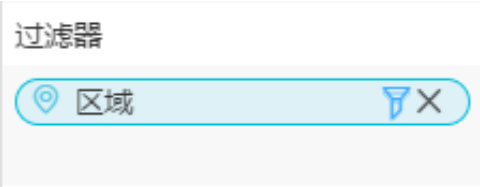
操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。

4. 在仪表板配置区，双击**柱图**图标。
5. 找到区域字段，并将其添加到过滤器中，如图 *1183:* 过滤器所示。

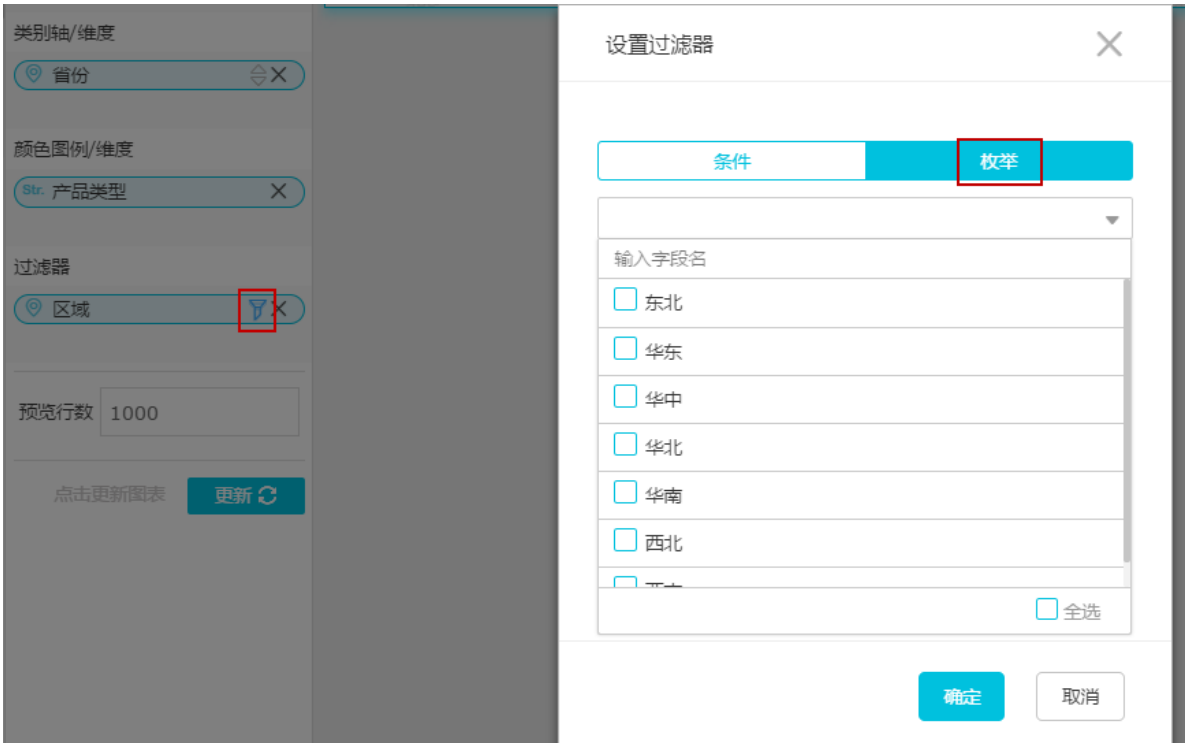
   我们需要通过过滤器，将**华东**从区域中筛选出来。

   图 1183: 过滤器



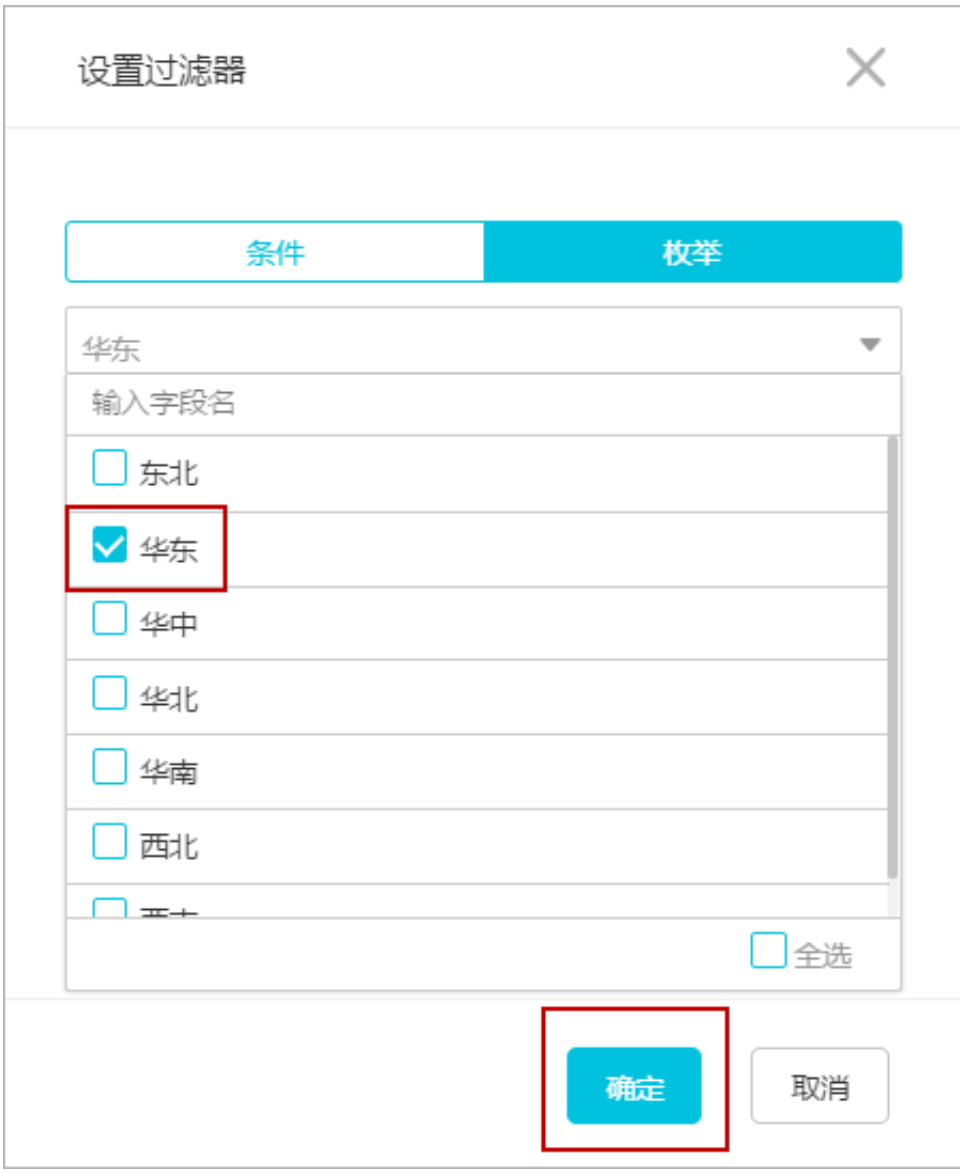
6. 单击**过滤**图标，设置过滤条件。
7. 选择**枚举**，系统会自动列出区域下所有可选择的选项，如图 *1184:* 枚举所示。

   图 1184: 枚举



8. 选择**华东**，并单击**确定**，如图 *1185:* 选择枚举选项所示。

   图 1185: 选择枚举选项



9. 找到省份和产品类型字段，并将它们依次添加到类别轴区域中。
10. 找到运输成本字段，并将其添加到值轴区域中。

> **说明:** 请确保省份字段的维度类型已经从字符串切换为了地理信息。

切换维度字段类型，请参见编辑维度。

11. 将**产品类型**字段拖拽到**颜色图例**中，如图 *1186:* 启用颜色图例所示。

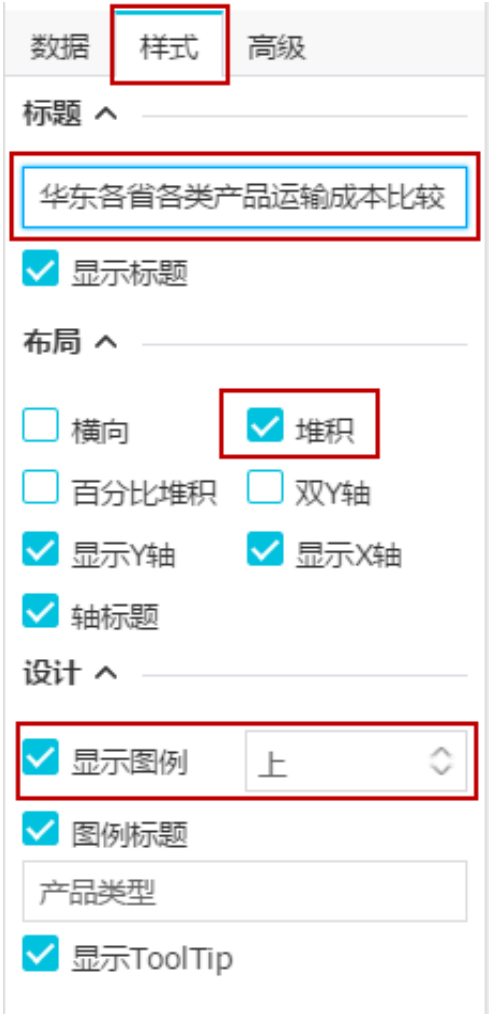
图 1186: 启用颜色图例

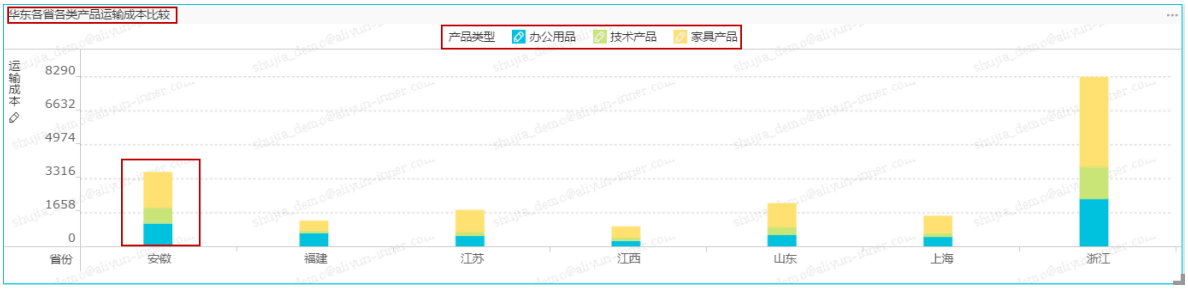


12.单击**更新**，更新图表。

13.单击**样式**标签页，选择**堆积**，如图 *1187:* 编辑后的柱图所示。

图 1187: 编辑后的柱图





场景二：以 company_sales_record 数据集为例，用柱图展示，各省各类产品的订单数量和平均利润金额比较。

此场景中有可能会涉及到数据建模过程。实现数据建模，请参见新建计算字段。

14.在**数据**标签页，选择需要的维度字段和度量字段。

在维度列表中，找到省份和产品类型，并将它们依次添加到类别轴区域中；在度量列表中，找到订单数量和平均利润金额，并将它们依次添加到值轴区域中。

图 1188: 选择柱图字段

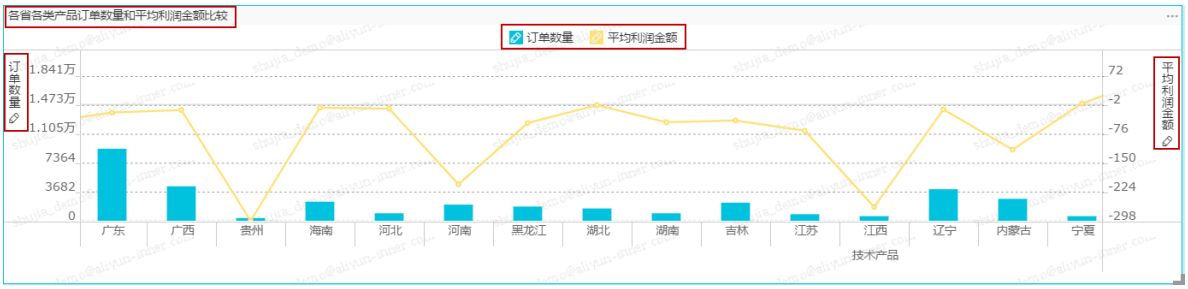


15.单击**更新**，更新图表。

16.单击**样式**标签页，选择**双Y轴**，如图 *1189:* 编辑后的柱图所示。

图 1189: 编辑后的柱图





17.单击**保存**图标，输入一个仪表板名称。

18.单击**确定**，保存仪表板。

系统会默认将仪表板保存到**我的仪表板**中。

31.4.4.3 饼图

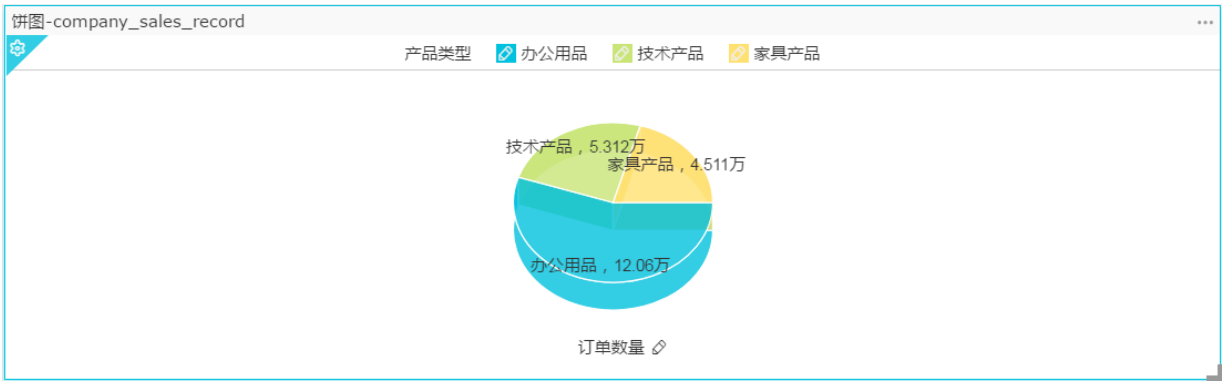
饼图显示的是一个数据系列，每个数据系列具有唯一的颜色或图案。饼图可用来展示数据中各项的大小与各项总和的比例，比如展示五险一金在个人收入中的支出比例，或者展示某一种汽车品牌在汽车总销量中占有的销售比。

背景信息

饼图是由一个个扇区构成的。每个扇区的标签由数据的维度决定，如区域，产品类型等；每个扇区角度的大小由数据的度量决定，如订单数量，订单金额等。

一个典型的饼图如图 *1190:* 饼图示例所示。

图 1190: 饼图示例



饼图的扇区标签区域最多取1个维度，并且维度的值必须小于或等于12，如区域、产品类型等；扇区角度区域最多取1个度量，如订单数量、利润金额等。

以 company_sales_record 数据集为例，用饼图展示，不同区域的运输成本比较。

操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**饼图**图标。
5. 在**数据**标签页，选择需要的维度字段和度量字段。

   在维度列表中，找到区域，并将其添加到扇区标签区域中；在度量列表中，找到运输成本，并将其放到扇区大小区域中，如图 *1191:* 选择饼图字段所示。

   说明: 请确保区域字段的维度类型已经从字符串切换为了地理信息，切换维度字段类型，请参见编辑维度。

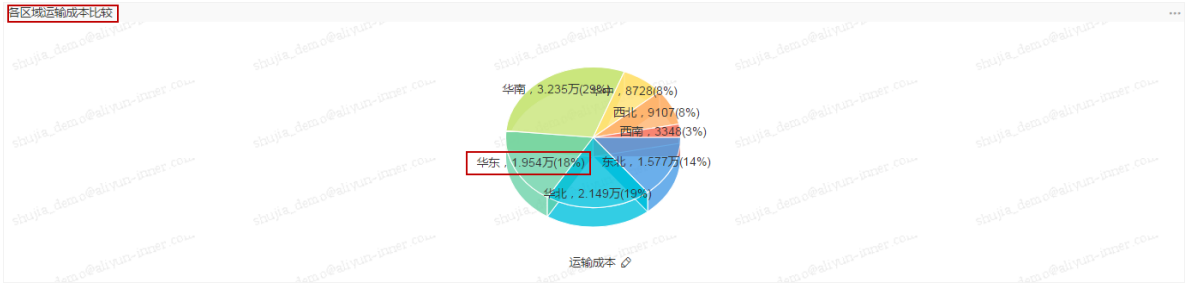
   图 1191: 选择饼图字段



6. 单击**更新**，更新图表。
7. 单击**样式**标签页，选择**3D**和**标题**，**值**（**百分比**），如图 *1192:* 编辑后的饼图所示。

   图 1192: 编辑后的饼图





8. 单击**保存**图标，输入一个仪表板名称。
9. 单击**确定**，保存仪表板。

   系统会默认将仪表板保存到**我的仪表板**中。

31.4.4.4 气泡地图

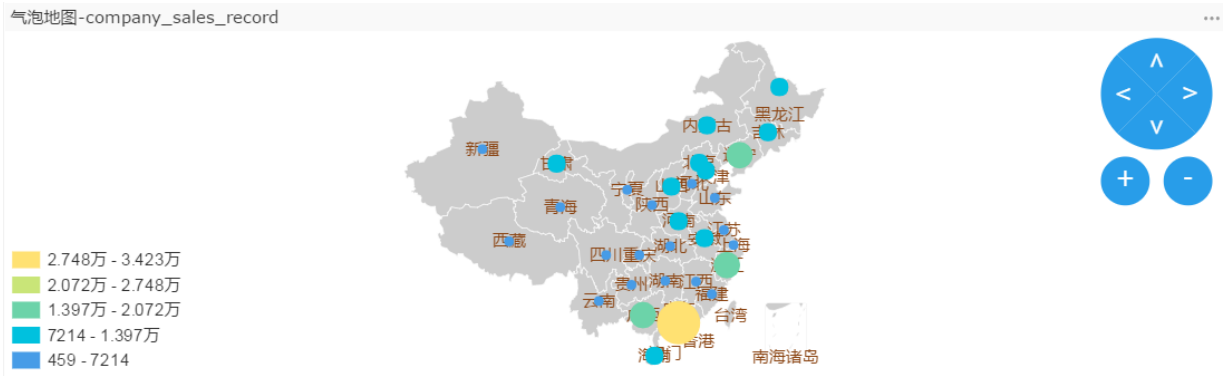
气泡地图以一个地图轮廓为背景，用附着在地图上面的气泡来反映数据的大小，而且，还可以直观地显示国家或地区的相关数据指标大小和分布范围。如展示各地旅游景点的客流量，或者展示各地区的人均收入等。

背景信息

气泡地图是由地理区域和气泡大小构成的。地理区域由数据的维度决定，如省份；气泡的大小由数据的度量决定，如运输成本，订单数量等。

一个典型的气泡地图如图 *1193:* 气泡地图示例所示。

图 1193: 气泡地图示例



气泡地图的地理区域最多只能取1个维度，并且维度类型必须为地理信息，如区域，省，城市等；气泡大小区域最少取1个，并且最多取5个度量。

以 company_sales_record 数据集为例，用气泡地图展示各省份订单数量和平均利润金额比较。

此场景中有可能会涉及到数据建模过程。

实现数据建模，请参见新建计算字段。

操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**气泡地图**图标。
5. 在**数据**标签页，选择需要的维度字段和度量字段。

   在维度列表中，找到省份，并将其添加到地理区域中；在度量列表中，找到订单数量和平均利润金额，并将依次添加到气泡大小区域中，如图 *1194:* 选择气泡地图字段所示。

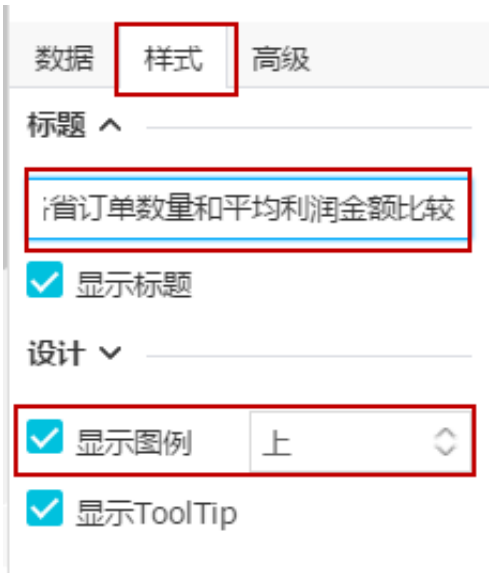
   说明: 请确保省份字段的维度类型已经从字符串切换为了地理信息，切换维度字段类型，请参见编辑维度。

   图 1194: 选择气泡地图字段



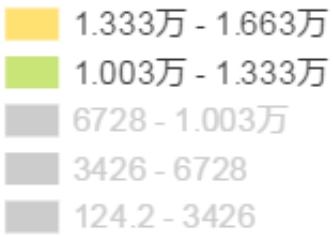
6. 单击**更新**，更新图表。
7. 单击**样式**标签页，更改图表的显示标题和显示图例，如图 *1195:* 编辑后的气泡地图所示。

图 1195: 编辑后的气泡地图





- （1）单击显示图例名称，可以在地图上切换显示的内容。
- （2）单击色块图标，可隐藏不需要的数据。



- （3）单击方向箭头，可调整地图的位置；单击加号/减号，可放大或缩小地图。

8. 单击**保存**图标，输入一个仪表板名称。
9. 单击**确定**，保存仪表板。

   系统会默认将仪表板保存到**我的仪表板**中。

31.4.4.5 色彩地图

与气泡地图类似，色彩地图用色彩的深浅来展示数据的大小和分布范围。

背景信息

色彩地图是由地理区域和色彩的饱和度构成的。地理区域由数据的维度决定，如省份；色彩的饱和度由数据的度量决定，如订单金额，利润金额等。

一个典型的色彩地图如图 *1196:* 色彩地图示例所示。

图 1196: 色彩地图示例



色彩地图的地理区域只能取1个维度，并且维度类型必须为地理信息；色彩饱和度最少取1个，最多取5个度量。

以 company_sales_record 数据集为例，用色彩地图展示各区域运输成本，订单金额和利润金额的比较。

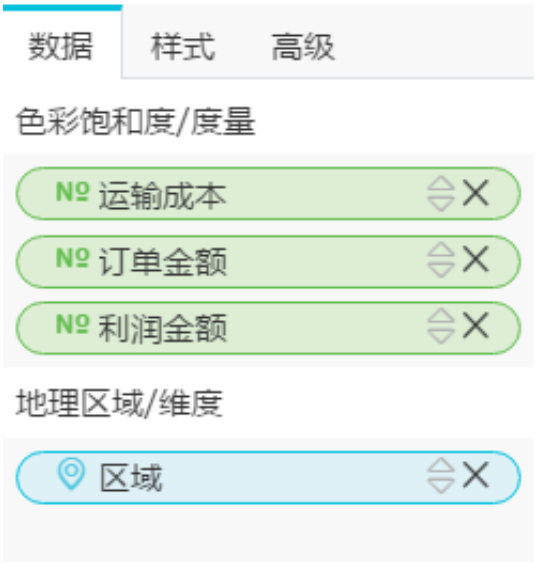
操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**色彩地图**图标。
5. 在**数据**标签页，选择需要的维度字段和度量字段。

   在维度列表中，找到区域，并将其添加到地理区域中；在度量列表中，找到订单金额、利润金额和运输成本，并将它们依次添加到色彩饱和度区域中，如图 *1197:* 选择色彩地图字段所示。

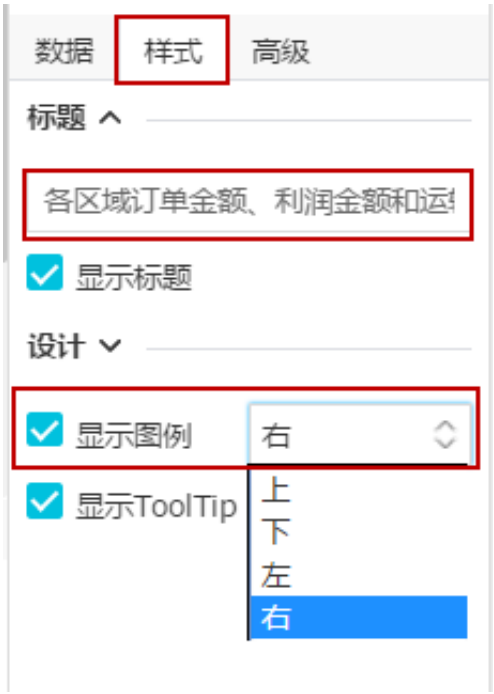
   说明: 请确保区域字段的维度类型已经从字符串切换为了地理信息，切换维度字段类型，请参见编辑维度。

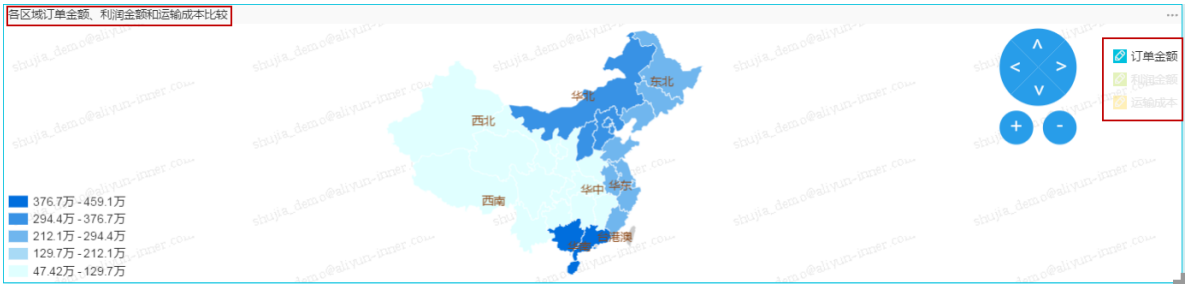
   图 1197: 选择色彩地图字段



6. 单击**更新**，更新图表。
7. 单击**样式**标签页，选择**右**，如图 *1198:* 编辑后的色彩地图所示。

   图 1198: 编辑后的色彩地图





在图表中，您可以任意切换显示图例的名称；调整地图的位置和大小；还可以隐藏不需要的数据，具体操作您可以参阅气泡地图。

8. 单击**保存**图标，输入一个仪表板名称。
9. 单击**确定**，保存仪表板。

   系统会默认将仪表板保存到**我的仪表板**中。

31.4.4.6 交叉表

交叉表可以用来显示表中某个字段的汇总值，并将它们分组；其中一组列在数据表的左侧，另一组列在数据表的上部。行和列的交叉处可以对数据进行多种汇总计算，比如求和、平均值、记数、最大值、最小值等。

背景信息

交叉表是由行和列构成的。行沿水平分布，由数据的维度决定，如省份，产品类型等；列沿垂直分布，由数据的度量决定，如订单数量，利润金额等。

一个典型的交叉表如图 *1199: 交叉表示例*所示。

图 1199: 交叉表示例

交叉表-company_sales_record

省份	区域	城市	订单数量	订单金额	利润金额	运输成本	单价
安徽	华东	安庆市	422	26047.266500000005	1517.7799999999997	116.92	1024.06
安徽	华东	蚌埠市	1934	122906.6695	-5572.280000000001	1101.24	8667.669999999995
安徽	华东	池州市	332	24645.126	7888.74	110.19999999999999	917.0300000000001
安徽	华东	滁州市	174	19308.8815	6055.28	92.69	892.75
安徽	华东	阜阳市	341	13551.349999999999	1672.81	127.88000000000001	560.8000000000001
安徽	华东	合肥市	296	12532.914	1710.9500000000003	121.53	662.2900000000001
安徽	华东	淮北市	397	32265.649999999994	6457.2	119.32999999999998	1510.3600000000001
安徽	华东	淮南市	399	38188.42049999999	-2238.6499999999996	225.70000000000005	2004.2000000000003
安徽	华东	黄山市	312	35147.8735	2394.3900000000003	280.98	1566.73

交叉表的行和列对维度和度量的取值无限制。

以 company_sales_record 数据集为例，用交叉表展示各省份不同类型产品的包装，以及它们的运输成本，订单数量和平均利润金额。

操作步骤

1. 登录 Quick BI 控制台.
2. 在左侧导航栏中单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**交叉表**图标。
5. 在**数据**标签页，为交叉表选择需要的维度字段和度量字段。

   在维度列表中，找到省份、产品类型和产品包箱，并将它们依次添加到行的区域中；在度量列表中，找到订单数量、运输成本和平均利润金额，并将它们依次添加到列的区域中，如图 *1200: 选择交叉表字段*所示。

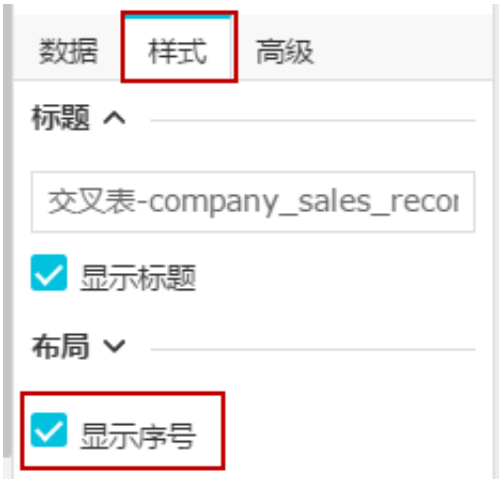
   说明: 请确保省份字段的维度类型已经从字符串切换为了地理信息，切换维度字段类型，请参见编辑维度。

   图 1200: 选择交叉表字段



6. 单击**更新**，更新图表。
7. 单击**样式**标签页，选择**显示序号**，如图 *1201:* 编辑后的交叉表所示。

图 1201: 编辑后的交叉表



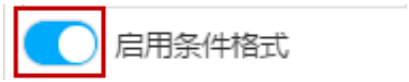
交叉表-company_sales_record_2

	省份	产品类型	产品包箱
1	安徽	办公用品	打包纸袋
2	安徽	办公用品	大型箱子
3	安徽	办公用品	巨型木箱
4	安徽	办公用品	小型包裹
5	安徽	办公用品	小型箱子
6	安徽	办公用品	中型箱子
7	安徽	技术产品	打包纸袋
8	安徽	技术产品	大型箱子
9	安徽	技术产品	巨型木箱
10			

在规则菜单中，您还可以编辑交叉表中数据的展示样式，比如更改数据的字体颜色，在数据旁边加标识，或者提亮数字所在的区域，方便看表的人快速定位重要的数据。

1. 单击**启用条件格式**开关，开启规则功能，如图 *1202:* 启用条件格式所示。

图 1202: 启用条件格式



2. 单击**切换**图标，选择图表中的数据系列进行规则设置，如图 *1203:* 规则设置所示。

图 1203: 规则设置



3. 单击下拉箭头，编辑数值和数值的显示样式，如图 *1204:* 数值显示样式所示。

图 1204: 数值显示样式



以订单数量数据为例，将数值大于1000的、数值在1000与500之间的和数值低于500的数据进行规则处理，如图 *1205:* 样式编辑示例所示。

图 1205: 样式编辑示例



- 当数值大于1000时，该数值的字体会标为红色，所在的区域也会被提亮，并且旁边会有红色向上的箭头做标识。
- 当数值的范围在1000与500之间时，该数值的字体会标为橙色，所在的区域不会提亮，并且旁边会有橙色的横线做标识。
- 当数值小于500时，该数值的字体颜色为系统默认颜色，所在的区域不会提亮，并且旁边会有绿色向下的箭头做标识。

图 1206: 编辑后的交叉表


交叉表-company_sales_record_2

	省份	产品类型	产品包箱	订单数量	运输成本	平均利润金额
23	北京	办公用品	巨型木箱	↓75	152.37	-7.961876313025209
24	北京	办公用品	小型包裹	↓425	93.16	-0.6084067437863455
25	北京	办公用品	小型箱子	↑3467	982.9400000000002	9.653549115817661
26	北京	办公用品	中型箱子	↓41	26.759999999999998	-4.761617647058824
27	北京	技术产品	打包纸袋	↓150	9.95	9.279342494714587
28	北京	技术产品	大型箱子	↓17	48.98	-20.525624999999998
29	北京	技术产品	巨型木箱	↓181	194.4	52.17100480769231
30	北京	技术产品	巨型纸箱	↓52	113.13	-538.7094814814815
31	北京	技术产品	小型包裹	↓426	42.26	0.8521707416420036
32	北京	技术产品	小型箱子	↑1146	326.9400000000001	2.1132574319938886
33	北京	技术产品	中型箱子	↓21	23.83	4.8979166666666645
34	北京	家具产品	打包纸袋	↓47	34.2	-21.35276595744681

8. 单击**保存**图标，输入一个仪表板名称。
9. 单击**确定**，保存仪表板。

系统会默认将仪表板保存到**我的仪表板**中。

31.4.4.7 仪表盘

类似于汽车使用的仪表盘，仪表盘可清晰地展示出某个指标值所在的范围。您可以直观地看出当前任务的完成程度，或者某个数据是在可控制范围内，还是即将超出预期。比如用仪表盘展示某一类商品的库存状态，是库存充足还是需要补货。

背景信息

仪表盘由指针角度和工具提示组成。指针角度和工具提示都由数据的度量决定，如折扣点，利润金额等。

一个典型的仪表盘如图 *1207: 仪表盘示例*所示。

图 1207: 仪表盘示例



仪表盘的指针角度和工具提示都只能取1个度量。

以 company_sales_record 数据集为例，用仪表盘展示展示订单金额。

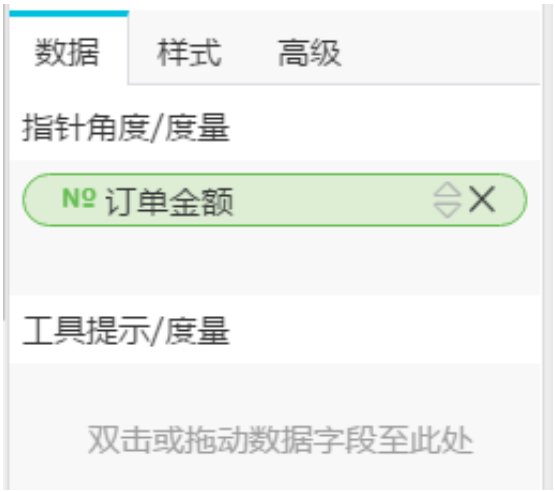
操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**仪表盘**图标。
5. 在**数据**标签页，选择需要的度量字段。

> **说明:** 系统会按照字段选择的先后顺序自动调整指针角度和工具提示的字段数量。

在度量列表中，找到订单金额，并将其添加到指针角度区域或工具提示区域，如图 *1208:* 选择仪表盘字段所示。

图 1208: 选择仪表盘字段



6. 单击**更新**，更新图表。
7. 单击**样式**标签页，编辑仪表盘的显示标题，布局，显示图例以及是否显示刻度等，如图 *1209:* 编辑仪表盘样式所示。

图 1209: 编辑仪表盘样式



8. 在区间设置区域，单击**添加**，输入区间的起始值和结束值。

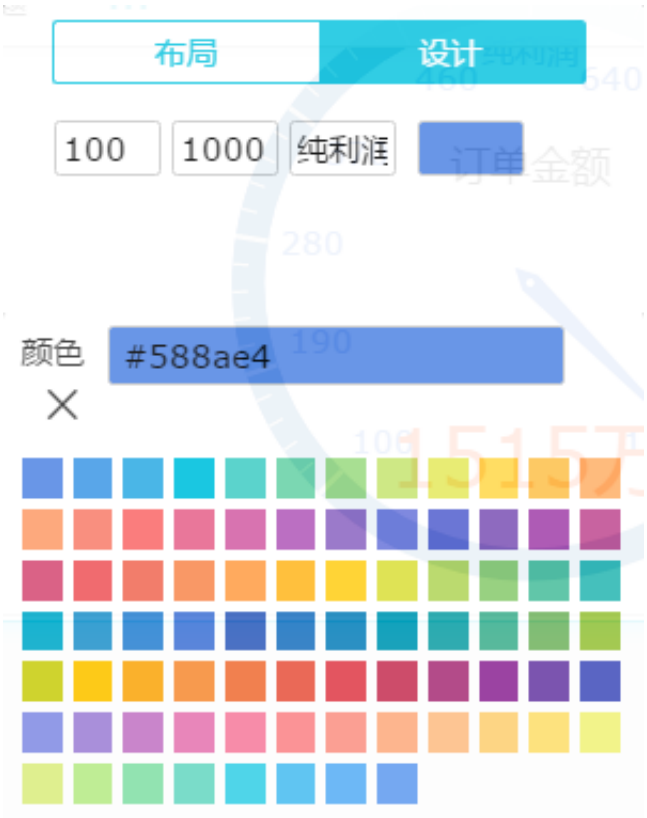
   如起始值为100，结束值为1000，区间的标题可设置为纯利润，如图 *1210:* 设置展示区间所示。

   图 1210: 设置展示区间



9. 单击色块，更改区间显示的配色方案，如图 *1211:* 更改区间的配色方案所示。

   图 1211: 更改区间的配色方案



10.单击**更新**，更新图表，如图 *1212:* 编辑后的仪表盘所示。

图 1212: 编辑后的仪表盘



11.单击**保存**图标，输入一个仪表板名称。

12.单击**确定**，保存仪表板。

系统会默认将仪表板保存到**我的仪表板**中。

31.4.4.8 雷达图

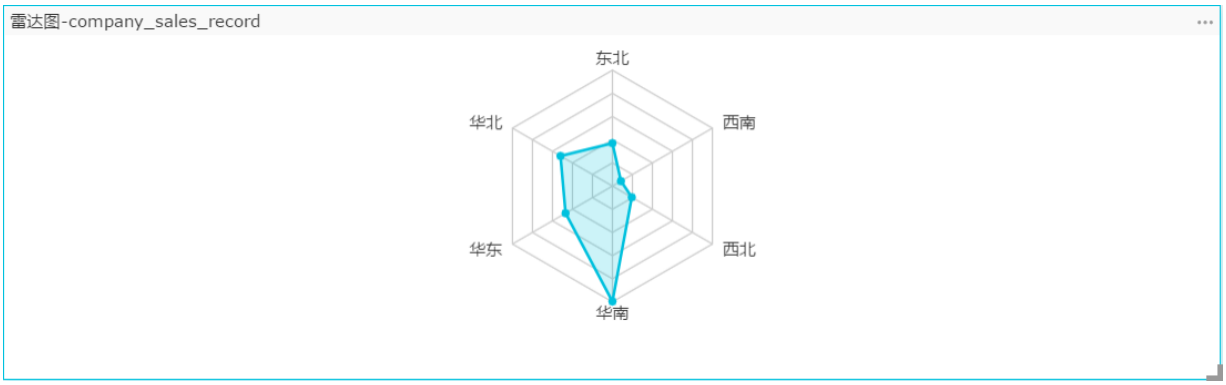
雷达图可用来展示分析所得的数字或比率，使用者能一目了然的了解各数据指标的变动情形及其好坏趋向，比如，用雷达图展示各区域的销售额统计。

背景信息

雷达图是由分支标签和分支长度构成的。分支标签由数据的维度决定，如产品类别；分支长度由数据的度量决定，如运输成本。

一个典型的雷达图如图 *1213:* 雷达图示例所示。

图 1213: 雷达图示例



雷达图的分支标签最少取1个，最多取2个维度，并且该维度下的维度值必须大于等于3，且小于等于12；分支长度最少取1个度量。

以 company_sales_record 数据集为例，用雷达图展示各区域的订单数量和订单金额。

操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**雷达图**图标。
5. 在**数据**标签页，选择需要的维度字段和度量字段。

   在维度列表中，找到区域，并将其添加到分支标签区域中；在度量列表中，找到订单数量和订单金额，并将它们依次添加到分支长度区域中，如图 *1214:* 选择雷达图字段所示。

说明: 请确保区域字段的维度类型已经从字符串切换为了地理信息。

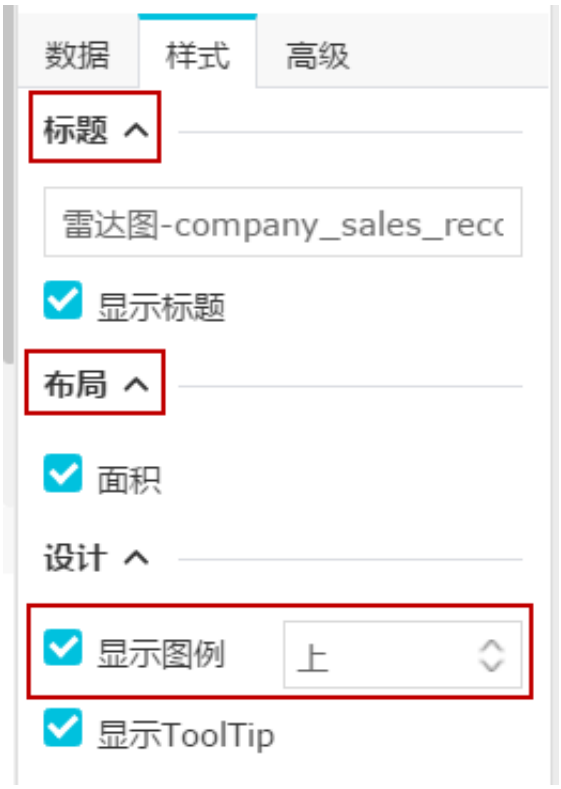
切换维度字段类型，请参见编辑维度。

图 1214: 选择雷达图字段



6. 单击**更新**，更新图表。
7. 单击**样式**标签页，编辑雷达图的显示标题，布局和显示图例，如图 *1215:* 编辑后的雷达图所示。

图 1215: 编辑后的雷达图





8. 单击**保存**图标，输入一个仪表板名称。
9. 单击**确定**，保存仪表板。

   系统会默认将仪表板保存到**我的仪表板**中。

31.4.4.9 散点图

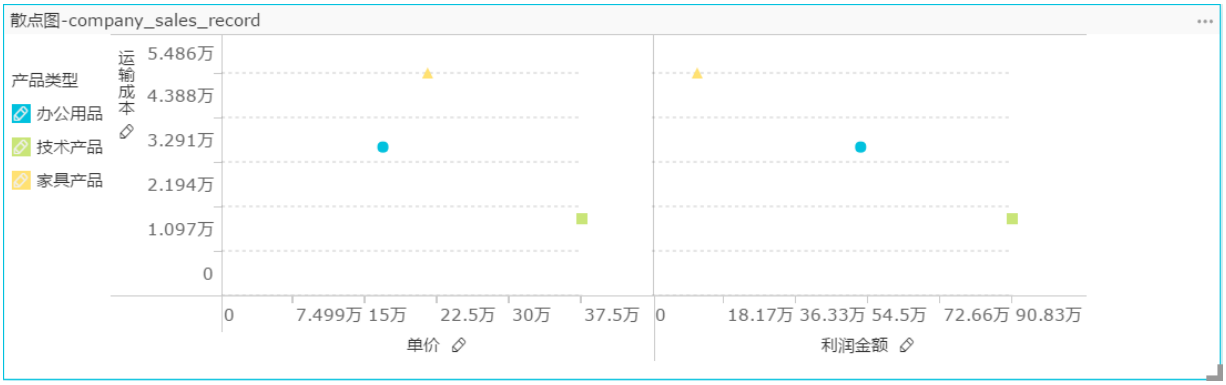
散点图可以用来展示数据的分布和聚合情况。

背景信息

散点图是由 X 轴和 Y 轴组成的。散点图的颜色图例由数据的维度决定，如产品类型；X 轴和 Y 轴的分布由数据的度量决定。

一个典型的散点图如图 *1216:* 散点图示例所示。

图 1216: 散点图示例



散点图的颜色图例取且只能取1个维度，并且维度成员的数值最大可达1000。

X 轴：最少取1个，最多取3个度量。

Y 轴：取且只能取1个度量。

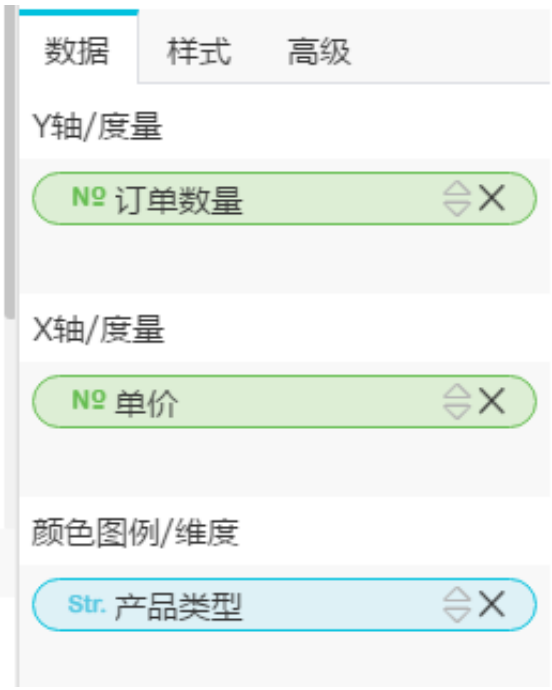
以 company_sales_record 数据集为例，用散点图展示不同类型产品的单价和订单数量。

操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**散点图**图标。
5. 在**数据**标签页，为散点图选择需要的维度字段和度量字段。

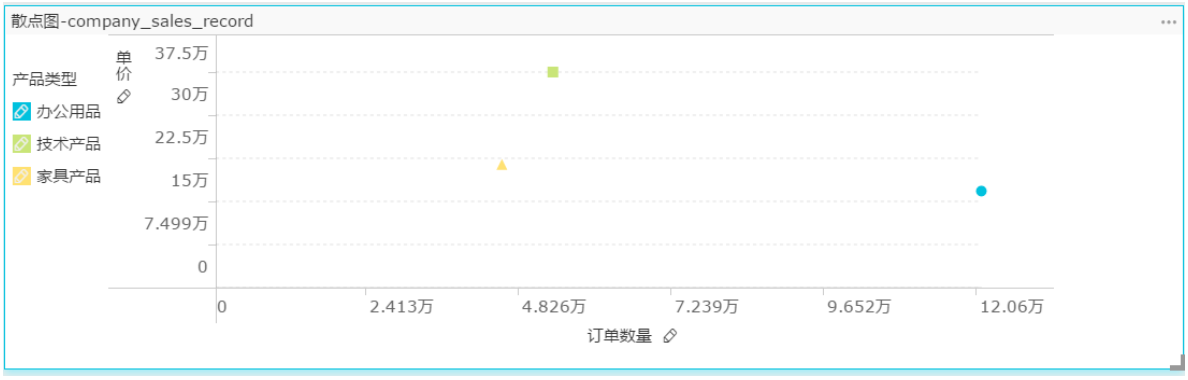
   在维度列表中，找到产品类型，并将其添加到颜色图例区域；在度量列表中，找到单价和订单数量，并将它们依次添加到Y轴和X轴区域，如图 *1217:* 选择散点图字段所示。

   图 1217: 选择散点图字段



6. 单击**更新**，更新图表。
7. 单击**样式**标签页，编辑散点图的显示标题，布局和显示图例，如图 *1218:* 编辑后的散点图所示。

图 1218: 编辑后的散点图



8. 单击**保存**图标，输入一个仪表板名称。
9. 单击**确定**，保存仪表板。

   系统会默认将仪表板保存到**我的仪表板**中。

31.4.4.10 漏斗图

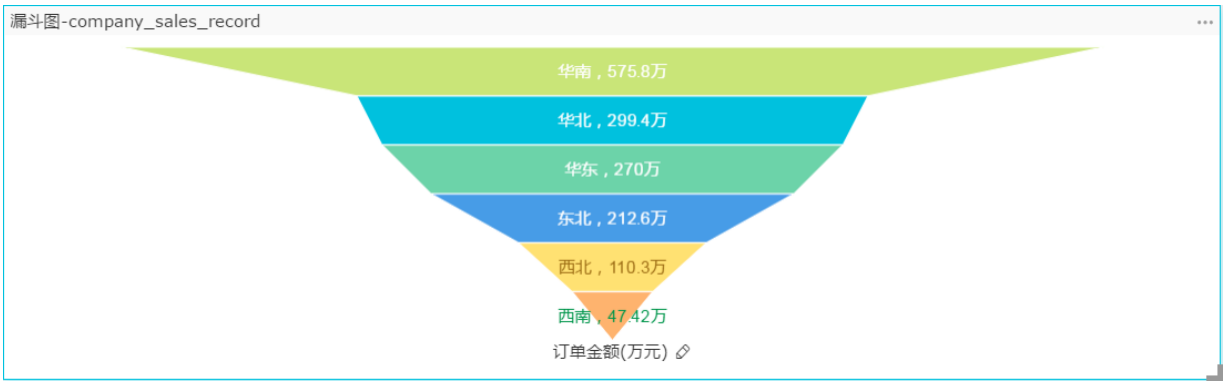
漏斗图适用于业务流程比较规范、周期长、环节多的流程分析，通过漏斗各环节业务数据的比较，能够直观地发现和说明问题所在。漏斗图还可以用来展示各步骤的转化率，适用于业务流程多的流程分析，比如通过漏斗图，可以清楚地展示用户从进入网站到实现购买的最终转化率。

背景信息

漏斗图是由漏斗层标签和漏斗层宽决定的，漏斗层标签由数据的维度决定，如区域；漏斗层宽度由数据的度量决定，如订单金额。

一个典型的漏斗图如图 *1219:* 漏斗图示例所示。

图 1219: 漏斗图示例



漏斗图的漏斗层标签只能取1个维度；漏斗层宽度只能取1个度量。

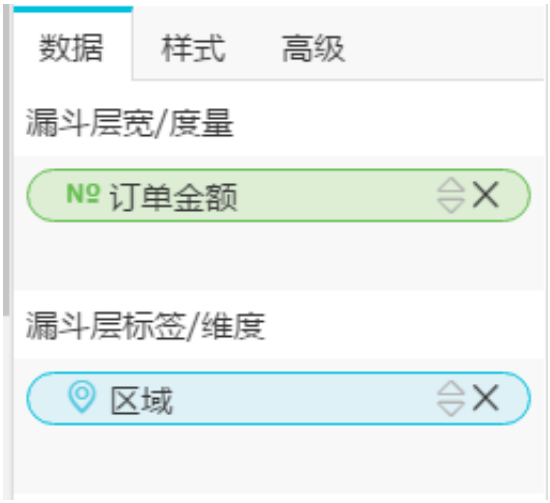
以 company_sales_record 数据集为例，用漏斗图展示各区域的订单金额比较。

操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中，单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**漏斗图**图标。
5. 在**数据**标签页，选择需要的维度字段和度量字段。

   在维度列表中，找到区域，并将其添加到漏斗层标签区域；在度量列表中，找到订单金额，并将其添加到漏斗层宽度区域，如图 *1220:* 选择漏斗图字段所示。

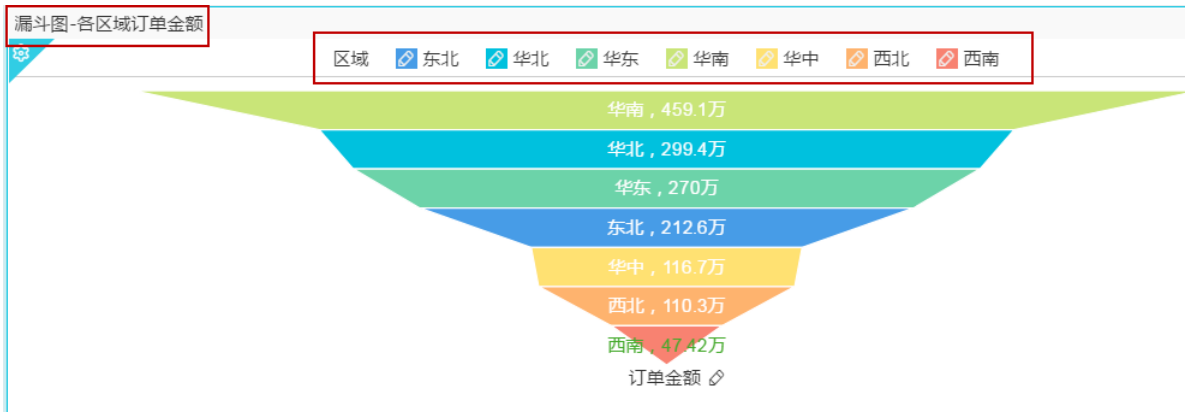
   图 1220: 选择漏斗图字段



6. 单击**更新**，更新图表。
7. 在**样式**标签页，编辑漏斗图的显示标题，布局和显示图例，如图 *1221:* 编辑后的漏斗图所示。

图 1221: 编辑后的漏斗图





8. 单击**保存**图标，输入一个仪表板名称。
9. 单击**确定**，保存仪表板。

   系统会默认将仪表板保存到**我的仪表板**中。

31.4.4.11 指标看板

指标看板可以对数据、或销售业绩等状况一目了然地展示，以便项目的参与者们可以及时掌握销售状况或者管理现状，从而能够快速制定并实施应对措施。因此，指标看板是发现问题、解决问题的很有效且直观的手段之一。

背景信息

指标看板是由看板标签和看板指标组成。看板标签由数据的维度决定，如区域；看板指标由数据的度量决定，如订单数量，订单金额等。

一个典型的指标看板如图 *1222:* 指标看板示例所示。

图 1222: 指标看板示例



指标看板的看板标签最多取1个维度；看板指标最少取1个，最多取10个度量。

以 company_sales_record 数据集为例，用指标看板展示各省份订单数量，订单金额，运输成本和利润金额的比较。

操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中，单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**指标看板**图标。
5. 在**数据**标签页，选择需要的维度字段和度量字段。

   在维度列表中，找到省份，并将其添加到看板标签区域；在度量列表中，找到订单数量，订单金额，运输成本和利润金额，并将它们依次添加到看板指标区域，如图 *1223:* 选择指标看板字段所示。

   说明: 请确保省份字段的维度类型已经从字符串切换为了地理信息。

   切换维度字段类型，请参见编辑维度。

   图 1223: 选择指标看板字段



6. 单击**更新**，更新图表。
7. 单击**样式**标签页，将每行显示的个数手动调整为3，如图 *1224:* 编辑后的指标看板所示。

   图 1224: 编辑后的指标看板





8. 单击**保存**图标，输入一个仪表板名称。
9. 单击**确定**，保存仪表板。

   系统会默认将仪表板保存到**我的仪表板**中。

31.4.4.12 矩阵树图

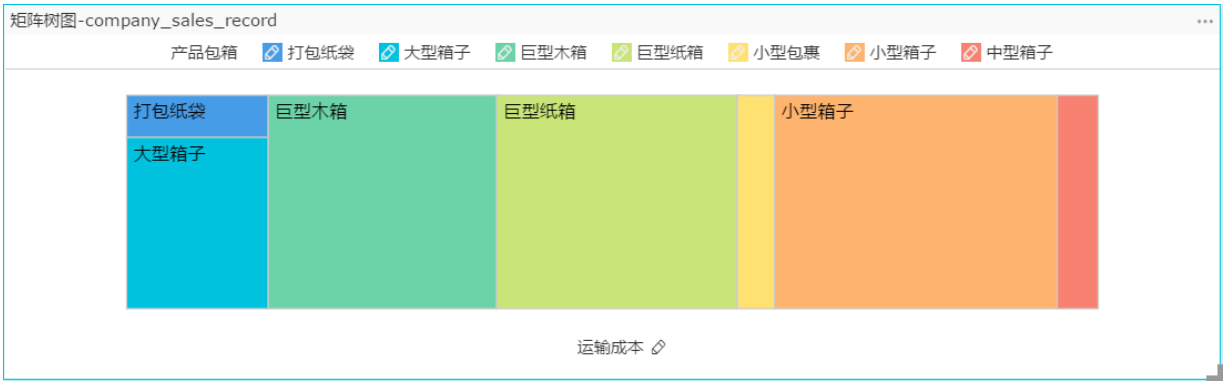
矩阵树图用来描述考察对象之间数据指标的相对占比关系。

背景信息

矩阵树图是由色块标签和色块大小组成的。色块标签由数据的维度决定，如产品包箱；色块大小由数据的度量决定，如运输成本。

一个典型的矩阵树图如图 *1225:* 矩阵树图示例所示。

图 1225: 矩阵树图示例



矩阵树图的色块标签取且只能取1个维度，并且维度值小于等于12；色块大小取且只能取1个度量。

以company_sales_record数据集为例，用矩阵树图展示各类产品订单数量的比较。

操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中，单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**矩阵树图**图标。
5. 在**数据**标签页，选择需要的维度字段和度量字段。

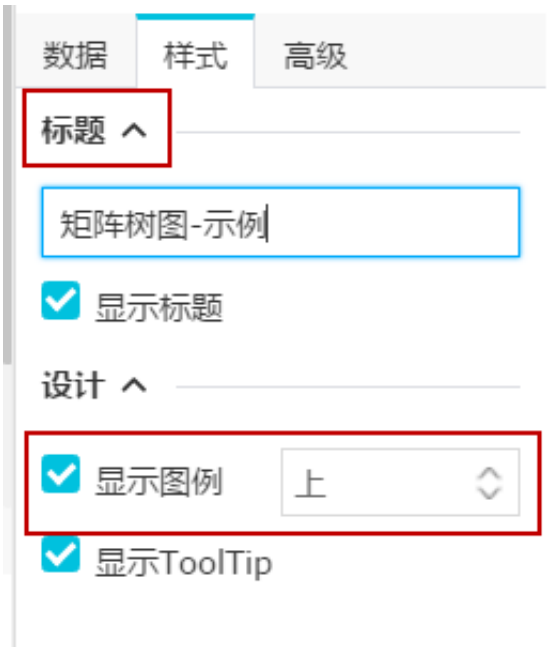
   在维度列表中，找到产品类型，并将其添加到色块标签区域；在度量列表中，找到订单数量，并将其添加到色块大小区域，如图 *1226:* 选择矩阵树图字段所示。

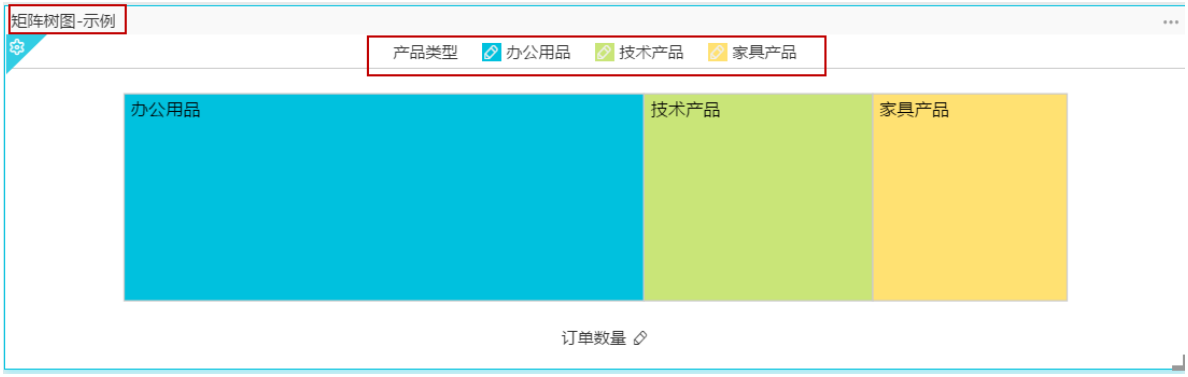
   图 1226: 选择矩阵树图字段



6. 单击**更新**，更新图表。
7. 单击**样式**标签页，编辑矩阵树图的显示标题和显示图例，如图 *1227:* 编辑后的矩阵树图所示。

图 1227: 编辑后的矩阵树图





8. 单击**保存**图标，输入一个仪表板名称。
9. 单击**确定**，保存仪表板。

   系统会默认将仪表板保存到**我的仪表板**中。

31.4.4.13 极坐标图

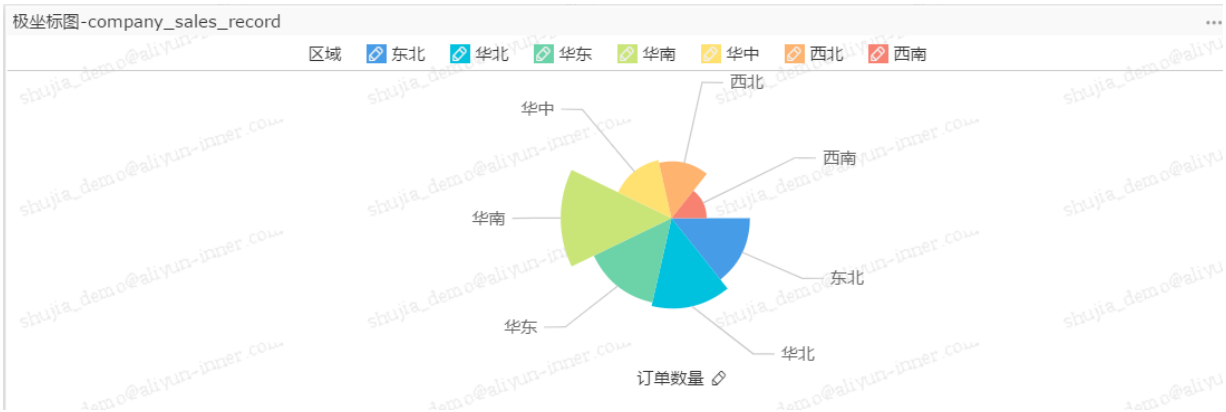
极坐标图可用来显示一段时间内的数据变化，或显示各项之间的比较情况。适用于枚举的数据，比如不同地域之间的数据比较。

背景信息

与饼图类似，极坐标图也是由一个个扇区构成的。每个扇区的标签由数据的维度决定，如区域，产品类型等；每个扇区长度由数据的度量决定，如订单数量，订单金额等。

一个典型的极坐标图如图 *1228:* 极坐标图示例所示。

图 1228: 极坐标图示例



极坐标图的扇区标签只能取1个维度，并且该维度下的成员数大于等于3且小于等于12；扇区长度只能取1个度量。

以 company_sales_record 数据集为例，用极坐标图展示各区域订单数量的比较（划分的区域数量必须大于等于3且小于等于12）。

操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中，单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**极坐标图**图标。
5. 在**数据**标签页，为极坐标图选择需要的维度字段和度量字段。

   在维度列表中，找到区域，并将其添加到扇区标签区域中；在度量列表中，找到订单数量，并将其添加到扇区长度区域中，如图 *1229:* 选择极坐标图字段所示。

   说明: 请确保，区域字段的维度类型已经从字符串切换为了地理信息。

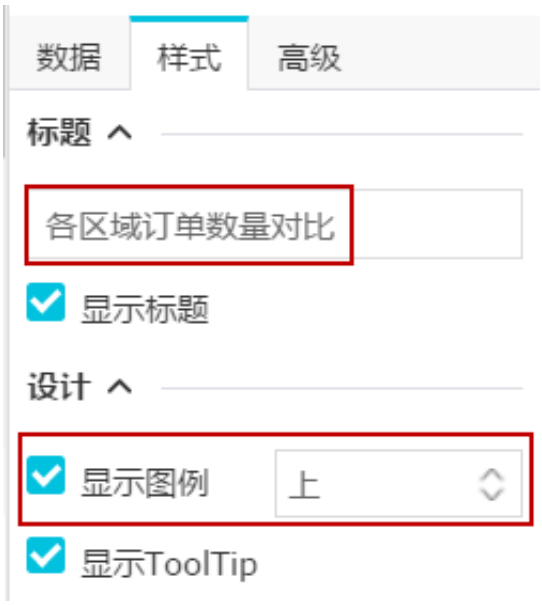
   切换维度字段类型，请参见编辑维度。

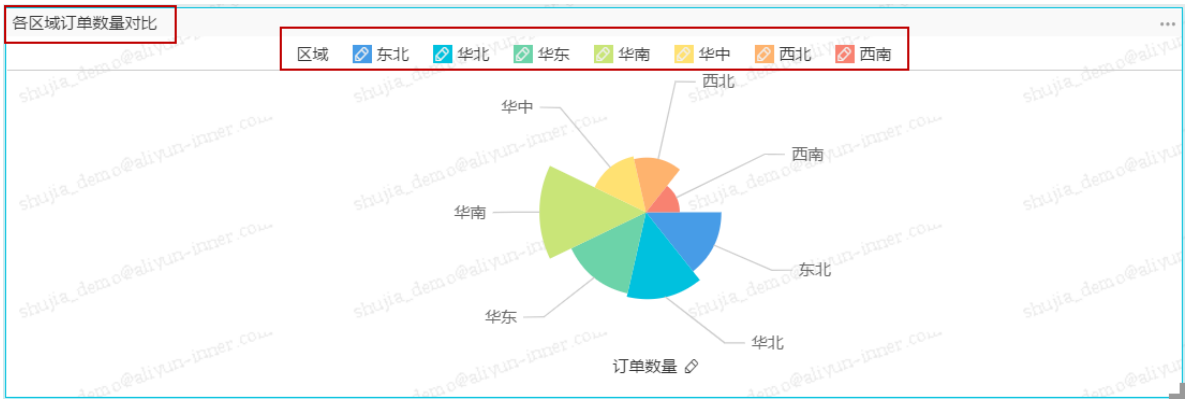
   图 1229: 选择极坐标图字段



6. 单击**更新**，更新图表。
7. 单击**样式**标签页，更改图表的显示标题和显示图例，如图 *1230:* 编辑后的极坐标图所示。

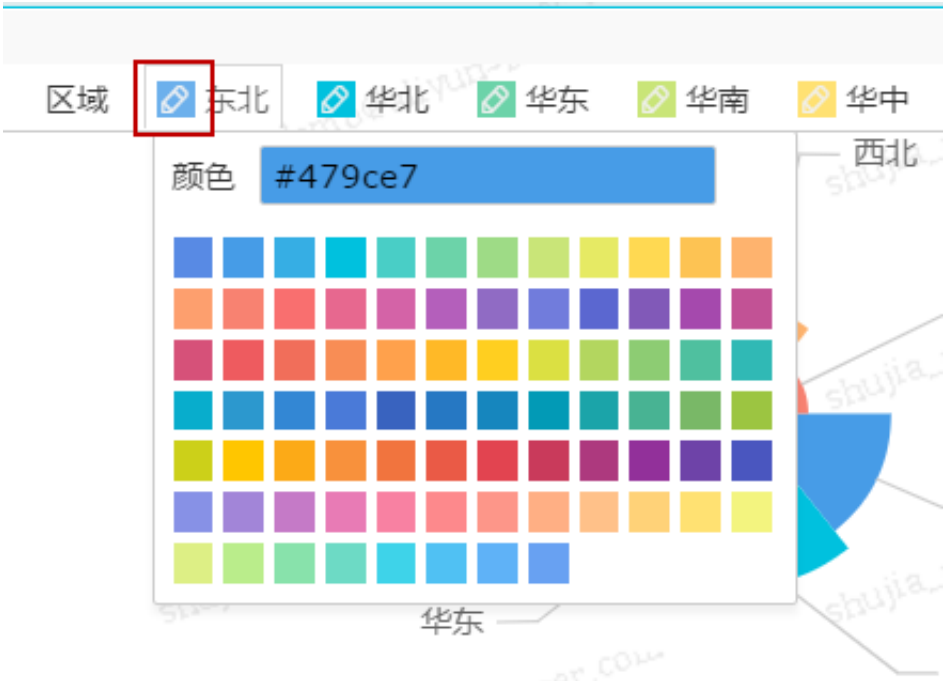
   图 1230: 编辑后的极坐标图





8. 单击图例前面的色块，更改显示图例的配色方案。

图 1231: 更改配色方案



9. 单击**保存**图标，输入一个仪表板名称。

10. 单击**确定**，保存仪表板。

系统会默认将仪表板保存到**我的仪表板**中。

31.4.4.14 词云图

词云图可以很直观的显示词频。适用于做一些用户的画像和用户的标签。

背景信息

词云图是由词标签和词大小构成的。每个词标签由数据的维度决定，如客户名称，产品名称等；每个词大小由数据的度量决定，如利润金额，单价等。

一个典型的词云图如图 *1232:* 词云图示例所示。

图 1232: 词云图示例



词云图的词标签只能取1个维度；词大小只能取1个度量。

以 company_sales_record 数据集为例，用词云图展示各省份订单数量的比较。

操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中，单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**词云图**图标。
5. 在**数据**标签页，选择需要的维度字段和度量字段。

   在维度列表中，找到省份，并将其添加到词标签区域中；在度量列表中，找到订单数量，并将其添加到词大小区域中，如图 *1233:* 选择词云图字段所示。

   说明: 请确保省份字段的维度类型已经从字符串切换为了地理信息。

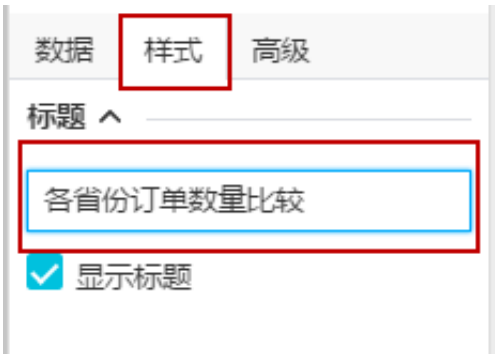
   切换维度字段类型，请参见编辑维度。

   图 1233: 选择词云图字段



6. 单击**更新**，更新图表。
7. 单击**样式**标签页，更改图表的显示标题，如图 *1234:* 编辑后的词云图所示。

   图 1234: 编辑后的词云图





8. 单击**保存**图标，输入一个仪表板名称。
9. 单击**确定**，保存仪表板。

   系统会默认将仪表板保存到**我的仪表板**中。

31.4.4.15 旋风漏斗图

旋风漏斗图可被看做是旋风图和漏斗图的结合体。旋风图可以用来比对两类事物在不同指标下的数据情况，比如，对比不同城市的工资收入和受教育程度。漏斗图可以用来展示各步骤的转化率，适用于业务流程多的流程分析，比如通过漏斗图，可以清楚地展示用户从进入网站到实现购买的最终转化率。

背景信息

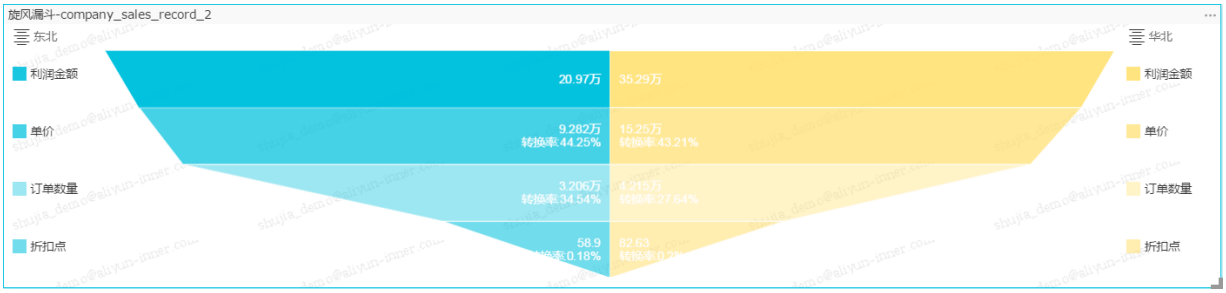
旋风漏斗图集合了这两个图表的特点，比如，对比北京和上海两个城市的流动人口比例、就业率、以及商品房交易量，如果对比项之间含有漏斗效应，那么旋风漏斗图既可以展示两个城市在不同指标下的数据对比，又可以将对比项之间的漏斗层级展示出来。

如果对比项之间没有漏斗效应，那么使用该图表展示数据时，图表会以普通旋风图展示；如果对比项之间形成了漏斗效应，但比对的对象只有一类事物，那么该图表会以普通漏斗图来展示数据。

旋风漏斗图是由对比主题和对比指标构成的。每个对比主题由数据的维度决定，如区域，产品类型等；每个对比指标由数据的度量决定，如订单数量，订单金额等。

一个典型的旋风漏斗图如图 *1235:* 旋风漏斗示例所示。

图 1235: 旋风漏斗示例



旋风漏斗图的对比主题只能取1个维度；对比指标至少取1个度量。

以 company_sales_record 数据集为例，用旋风漏斗图展示不同类型产品的订单数量、利润金额以及平均利润金额比较。

操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中，单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**旋风漏斗图**图标。
5. 在**数据**标签页，选择需要的维度字段和度量字段。

   在维度列表中，找到产品类型，并将其添加到对比主题区域中；在度量列表中，找到订单数量、利润金额和平均利润金额，并将它们依次添加到对比指标区域中，如图 *1236:* 选择旋风漏斗字段所示。

   图 1236: 选择旋风漏斗字段



6. 单击**更新**，更新图表。
7. 在**样式**标签页，更改图表的显示标题、图形布局、图例位置、配色方案和转化率显示。
   1. 旋风漏斗图提供了两种图表布局，您可以根据您看图的习惯自行选择。

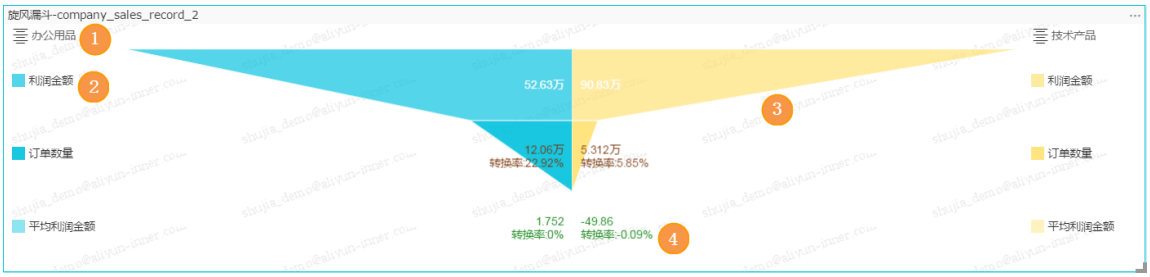
      图 1237: 图表布局

   2. 在**设计**菜单中，您可以更改图例的位置、调整图表的配色方案，以及选择是否在图表上显示转化率，如图 *1238:* 编辑后的旋风漏斗图所示。

      图 1238: 编辑后的旋风漏斗图



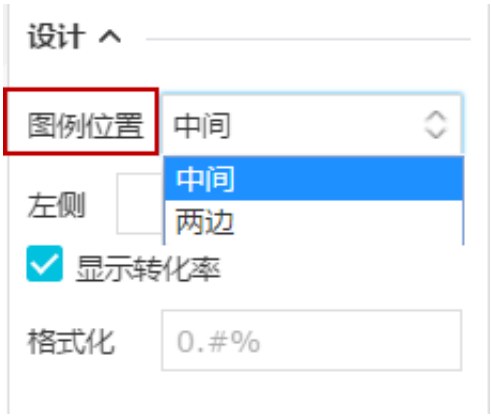
- （1）鼠标指向产品类型字段，可以任意切换产品类型，如图 *1239:* 切换产品类型所示。

图 1239: 切换产品类型



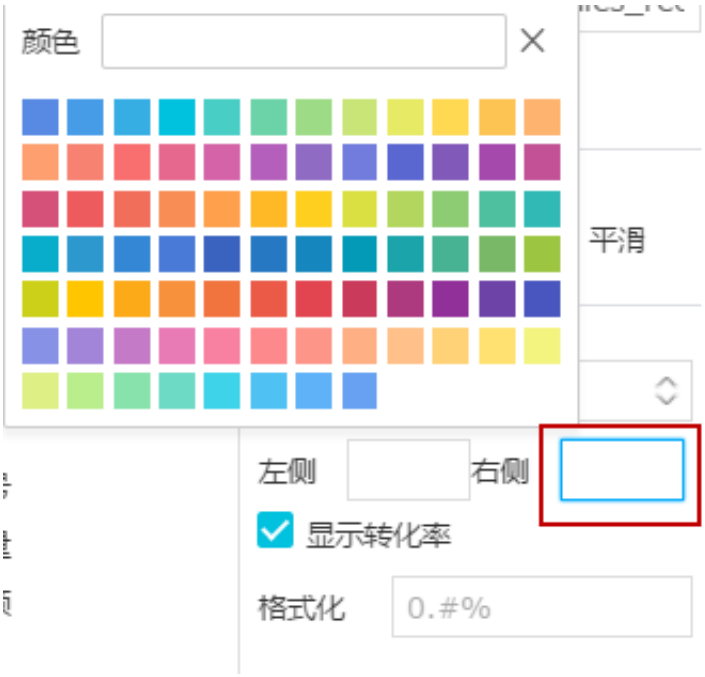
- （2）调整显示图例的展示位置，如图 *1240:* 图例位置所示。

图 1240: 图例位置



- （3）单击左侧方框和右侧方框，在颜色菜单中选择您喜欢的颜色，如图 *1241:* 更改配色方案所示。

图 1241: 更改配色方案



- （4）手动调整转化率的格式，如图 *1242: 转化率格式*所示。

图 1242: 转化率格式



8. 单击**保存**图标，输入一个仪表板名称。
9. 单击**确定**，保存仪表板。

   系统会默认将仪表板保存到**我的仪表板**中。

31.4.4.16 树图

树图是通过树形结构来展现层级数据的组织关系，以父子层次结构来组织对象，是枚举法的一种表达方式，比如查看某个省份下各地级市的收入状况，那么省份与地级市之间的关系就可以看做是父子层次结构。树图适用于与组织结构有关的分析，如公司的人员组织结构，或者医院的科室组织结构。

背景信息

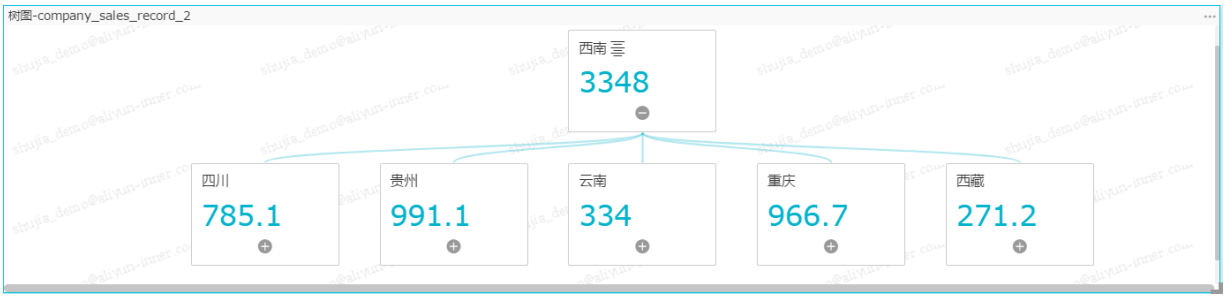
树图是由树父子节点标签和树父子节点指标构成的。每个树父子节点标签由数据的维度决定，如区域，产品类型等；每个树父子节点指标由数据的度量决定，如订单数量，订单金额等。

本章节为您提供了以下两个场景，包括如何使用过滤器。

- 场景一：各区域下各省份不同产品的订单数量比较
- 场景二：查看各直辖市不同产品的平均利润金额

一个典型的树图如图 *1243:* 树图示例所示。

图 1243: 树图示例



树图的树父子节点标签最少取2个维度，并且维度字段之间最好能构成父子关系；树父子节点指标最少取1个度量。

场景一：以 company_sales_record数据集为例，用树图展示各区域下各省份不同产品的订单数量比较。

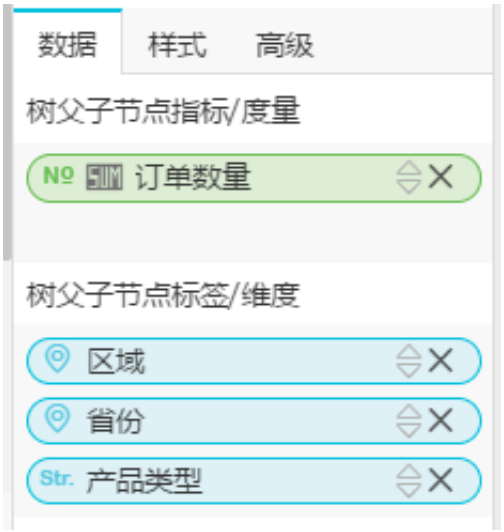
操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中，单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**树图**图标。
5. 在**数据**标签页，选择需要的维度字段和度量字段。

   在维度列表中，找到区域、省份和产品类型，并将它们依次添加到树父子节点标签区域中，它们当前的排列顺序就是图表中将要展示出的父子关系；在度量列表中，找到订单数量，并将其添加到树父子节点指标区域中，如图 *1244:* 选择树图字段所示。

   说明: 请确保区域字段和省份字段的维度类型已经从字符串切换为了地理信息，切换维度字段类型，请参见编辑维度。

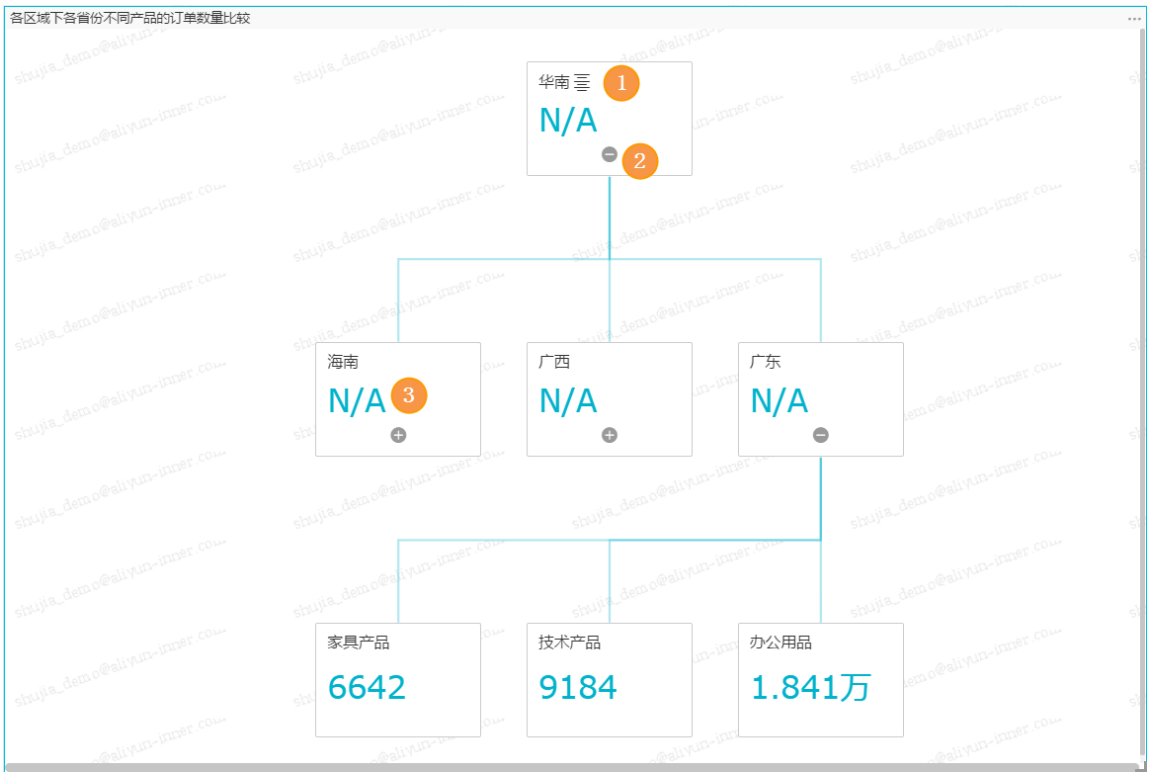
   图 1244: 选择树图字段



6. 单击**更新**，更新图表。
7. 在**样式**标签页，更改图表的显示标题，布局和设计。
   1. 树图提供了三种图表布局可供您选择，父子节点的延展方式（系统默认为合并根节点）和显示模式也可以根据您看图的习惯自行选择，以下示例在布局上选择了**直线**的显示模式，如图 *1245: 图表布局*所示。

      图 1245: 图表布局



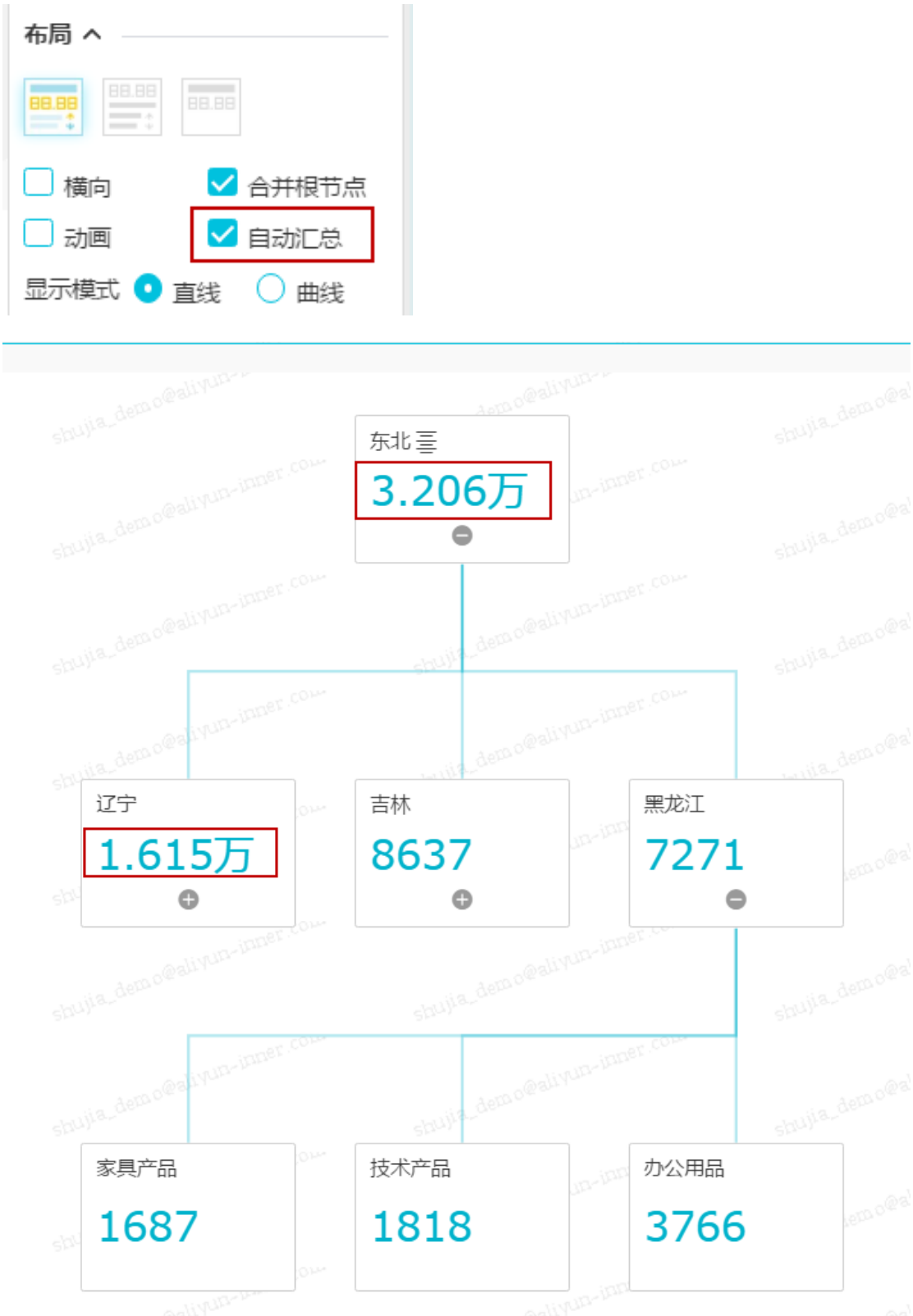
- （1）鼠标指向区域字段，可以任意切换区域名称，如图 *1246:* 切换区域所示。

图 1246: 切换区域



- （2）单击减号或者加号来收起或者释放子节点信息。
- （3） 在布局中，如果选择了**自动汇总**，图表会自动将汇总好的值展示在方框中，如图 *1247:* 自动汇总所示。

图 1247: 自动汇总

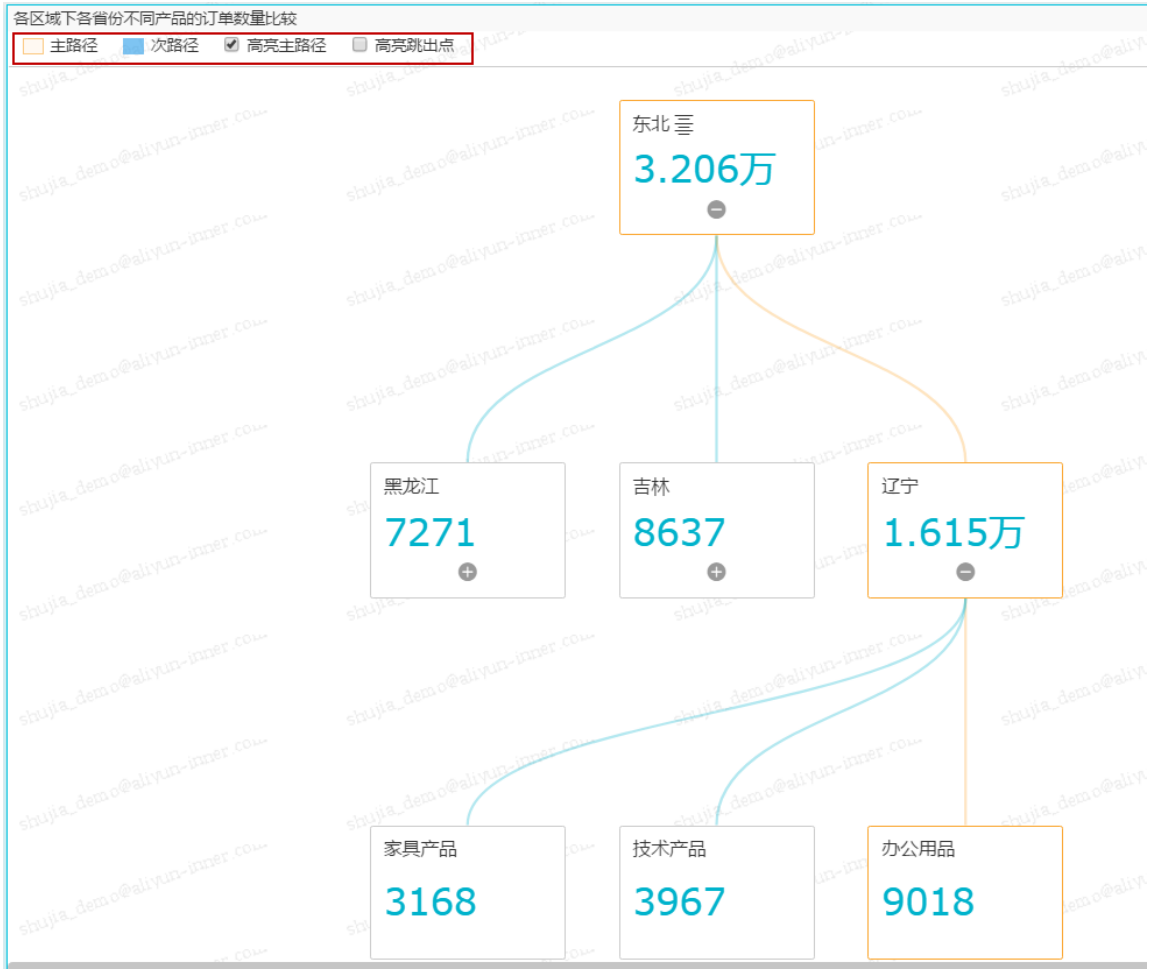


2. 您可以在设计菜单中编辑图表的展示层级，展示的层级数目可以手动输入；您还可以通过字段信息选择一条主路径，主路径将会以不同的颜色与其它路径区分开；您还可以将工具条加载到图表中，方便您在预览模式下或者在仪表板中继续修改。

以下示例在设计中选择了以订单数量为主路径，排列的方式为升序，并且将工具条嵌入了图表中，在布局中选择了**曲线**的显示模式，如图 *1248:* 编辑后的树图所示。

图 1248: 编辑后的树图





8. 单击**保存**图标，输入一个仪表板名称。
9. 单击**确定**，保存仪表板。

   系统会默认将仪表板保存到**我的仪表板**中。

场景二：**查看各直辖市不同产品的平均利润金额。**

此场景中有可能会涉及到数据建模过程。实现数据建模，请参见新建计算字段。

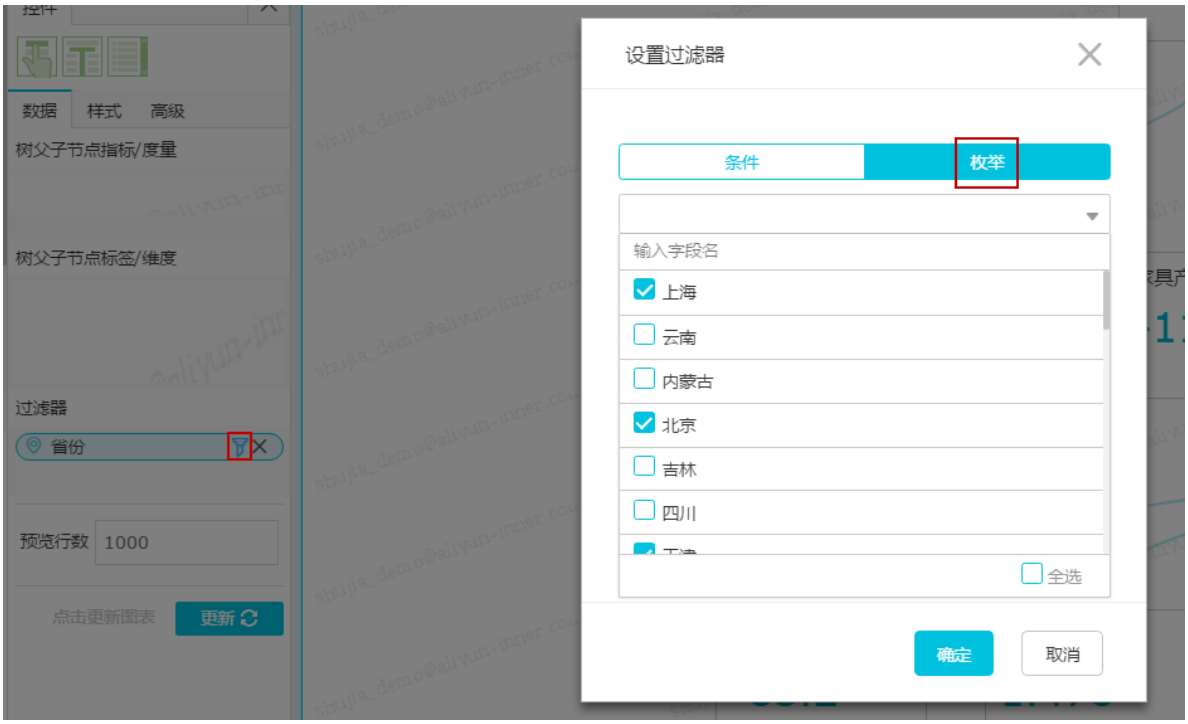
10. 在维度列表中，找到省份字段，并将其添加到过滤器中。

    我们需要通过过滤器将直辖市从省份中筛选出来。

11. 单击**筛选**图标，选择**枚举**，如所示。

    系统会自动将省份字段中所有可选的信息列图 *1249:* 枚举在下拉菜单中。

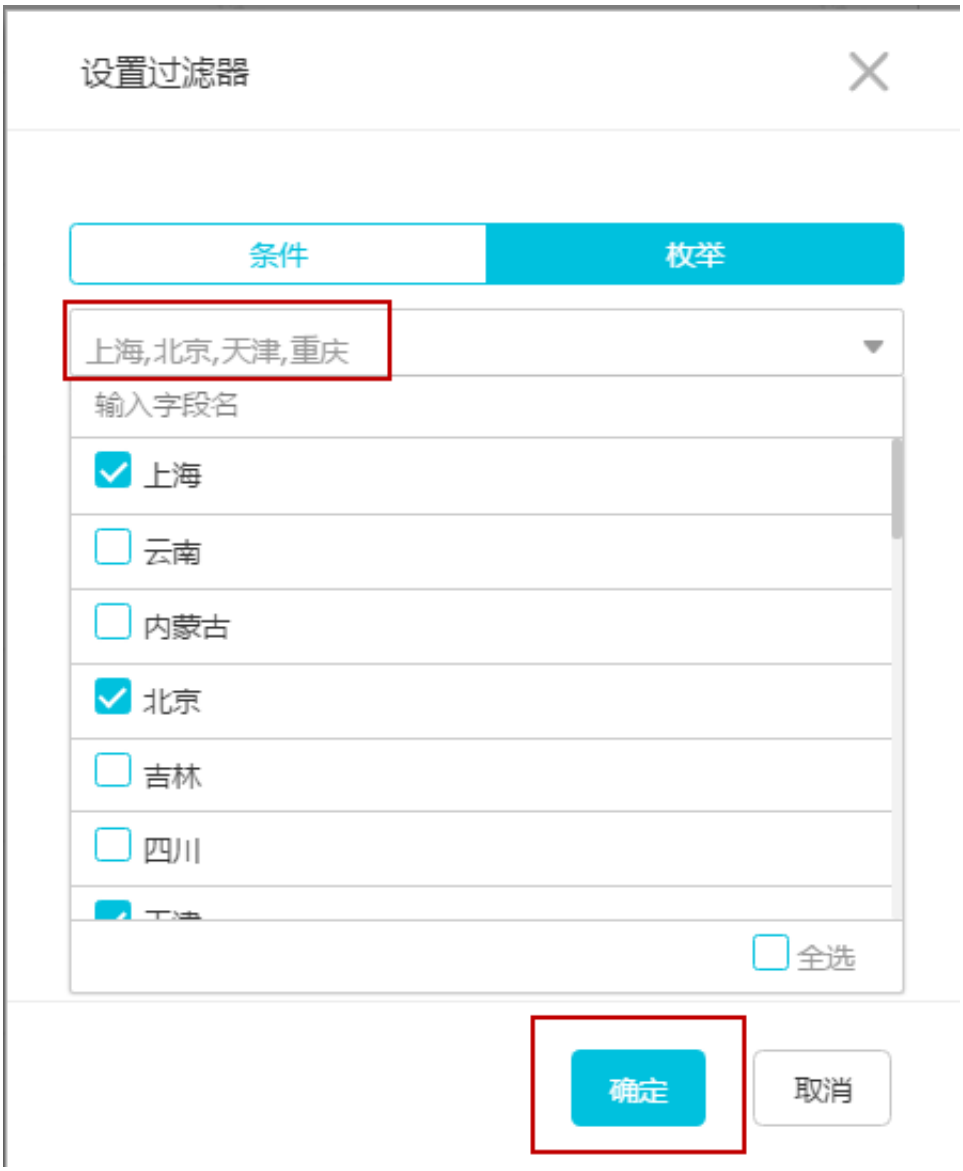
    图 1249: 枚举



12.选择需要的城市，或者手动输入字段名。

13.单击**确定**，完成过滤条件设置，如图 *1250:* 选择枚举条件所示。

图 1250: 选择枚举条件



14.继续从维度列表中，找到城市，产品类型和产品小类，并将它们依次添加到树父子节点标签区域，如图 *1251:* 选择树图字段所示。

它们当前的排列顺序就是图表中将要展示出的父子关系；在度量列表中，找到平均利润金额，并将其添加到树父子节点指标区域。

图 1251: 选择树图字段

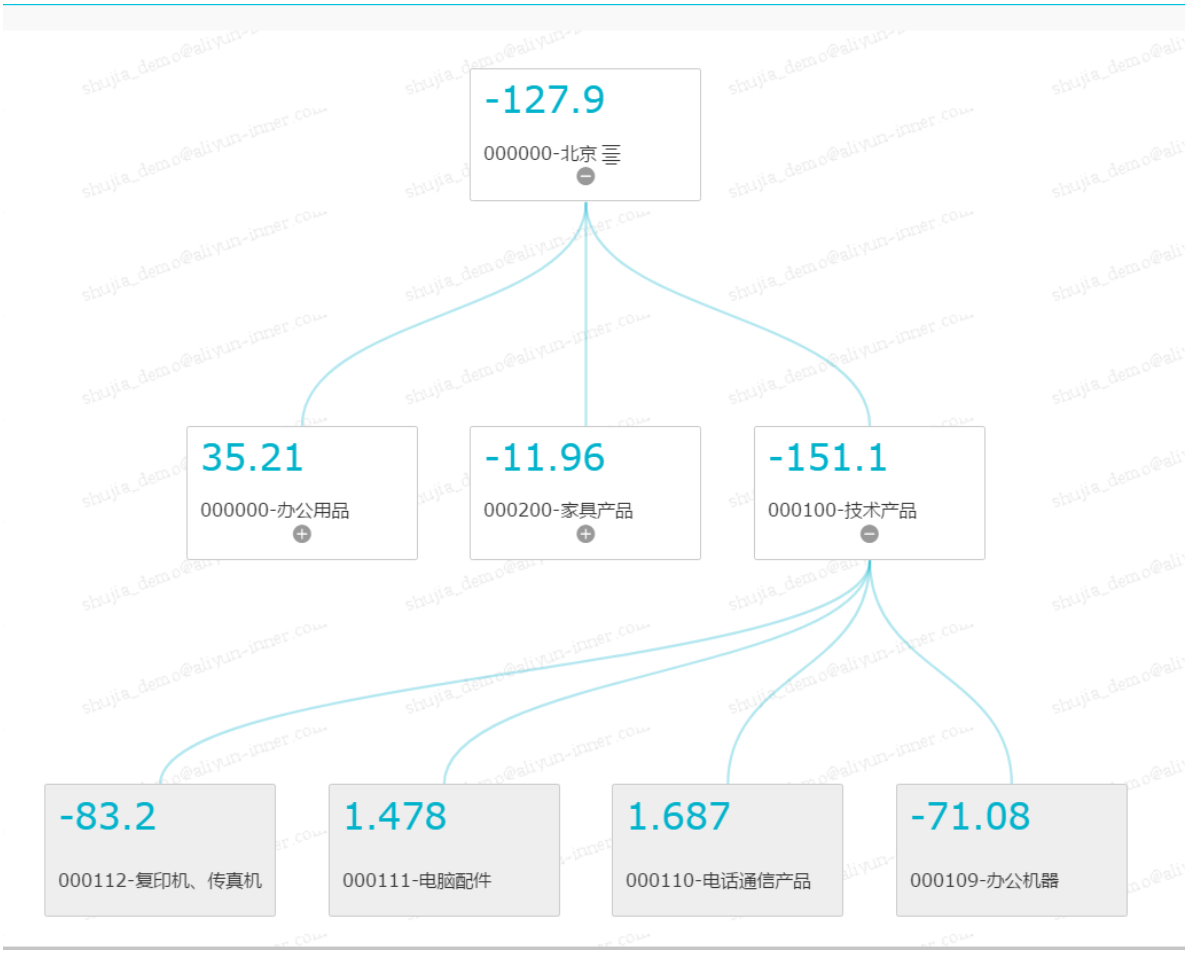


15.单击**更新**，更新图表。

16.单击**样式**标签页，更改图标的显示标题，布局和设计，如图 *1252:* 编辑后的树图所示。

图 1252: 编辑后的树图


数据
样式
高级
标题
树图-company_sales_record_
显示标题
布局
横向
合并根节点
动画
自动汇总
显示模式
直线
曲线
设计
层级
全部
0
主路径
无
排序
高亮主路径
高亮跳出路径
工具条



17.单击**保存**图标，输入一个仪表板名称。

18.单击**确定**，保存仪表板。

系统会默认将仪表板保存到**我的仪表板**中。

31.4.4.17 来源去向图

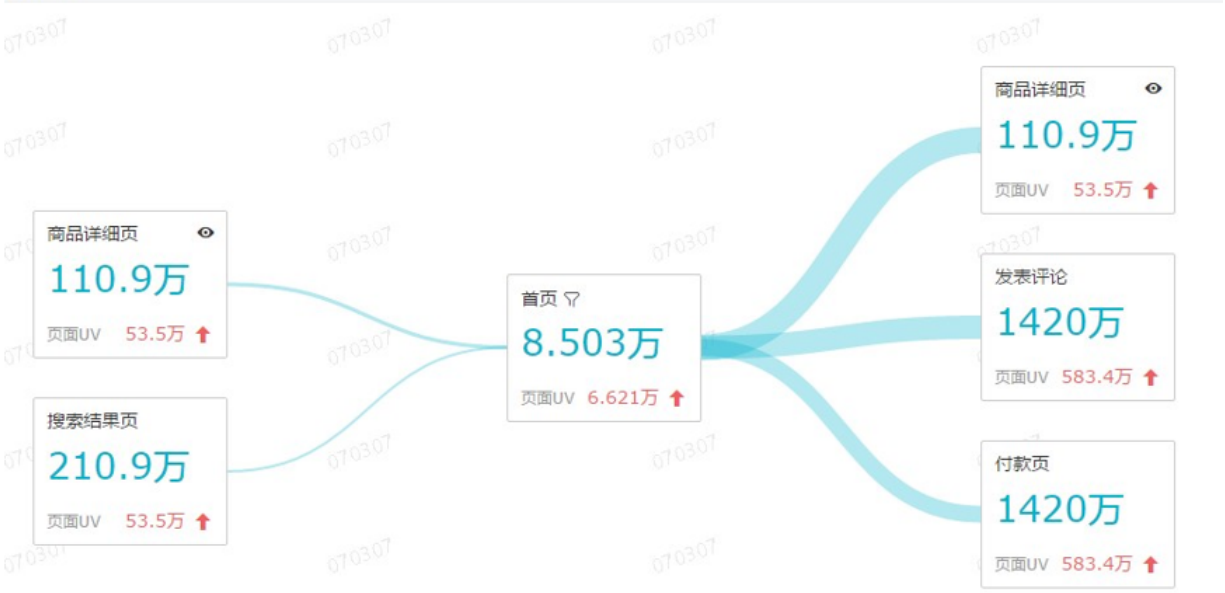
来源去向图是通过页面访问量或点击量 PV（Page View）和访客的数量 UV（Unique Vistor）推算出网页的转化率，进而可以知晓网站的整体运营效果和某一类商品的最终成交量。来源去向图适用于电商或与营销有关系的分析，比如分析购物网站中，哪些商品最畅销或者哪一个时间段是访问高峰。

背景信息

来源去向图目前仅支持三级的维度，这三级维度是由前一页面，当前页面和后一页面构成的；图表的度量是由PV、UV、路径转化率和页面跳出率构成的，其中 PV 和 UV 的来源均来自前一页面，当前页面和下一页面。

一个典型的来源去向图如图 *1253:* 来源去向图示例所示。

图 1253: 来源去向图示例



来源去向图的前一页面，当前页面和后一页面，每一个页面最多取1个维度，并且维度字段之间是有层级关系的，它们排列的先后顺序就是将要展示在图表中的层级关系；PV（前一页面、当前页面和后一页面）、UV（前一页面、当前页面和后一页面）、路径转化率和页面跳出率，每一项最多取1个度量。

制作图表时，除了三个维度项和两个转化率是必选项以外，三个PV项和三个UV项，可以任选其一，即选择三个 PV 值，或者选择三个 UV 值。在添加维度字段和度量字段时，如果填入的字段有误，系统也会给出相应的提示。

以 page_source_target_day_stat 数据集为例，通过页面的点击量，用来源去向图展示展示各页面之间的转化率和跳出率。

操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中，单击**作品**图标，进入作品列表页面。
3. 单击**新建** > **新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**来源去向图**图标。
5. 在**数据**标签页，选择需要的维度字段和度量字段。

在维度列表中，找到前一页面，当前页面和后一页面的名称，并将它们依次填入对应的区域。它们的排列顺序就是图表将要展示的层级关系；在度量列表，找到页面路径转化率和页面跳出率，并将它们依次填入到对应的区域，继续在度量列表中找到三个 PV 数值和 UV 数值，也将它们依次填入到对应的区域（也可以只选择三个 PV 数值或者三个 UV 数值填入）。

图 1254: 选择来源去向图字段

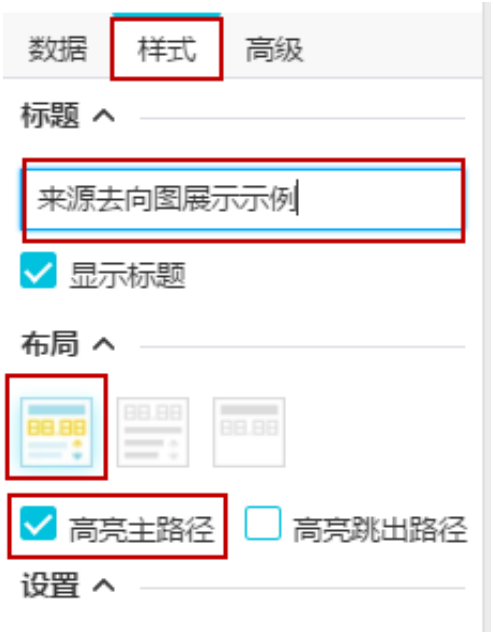


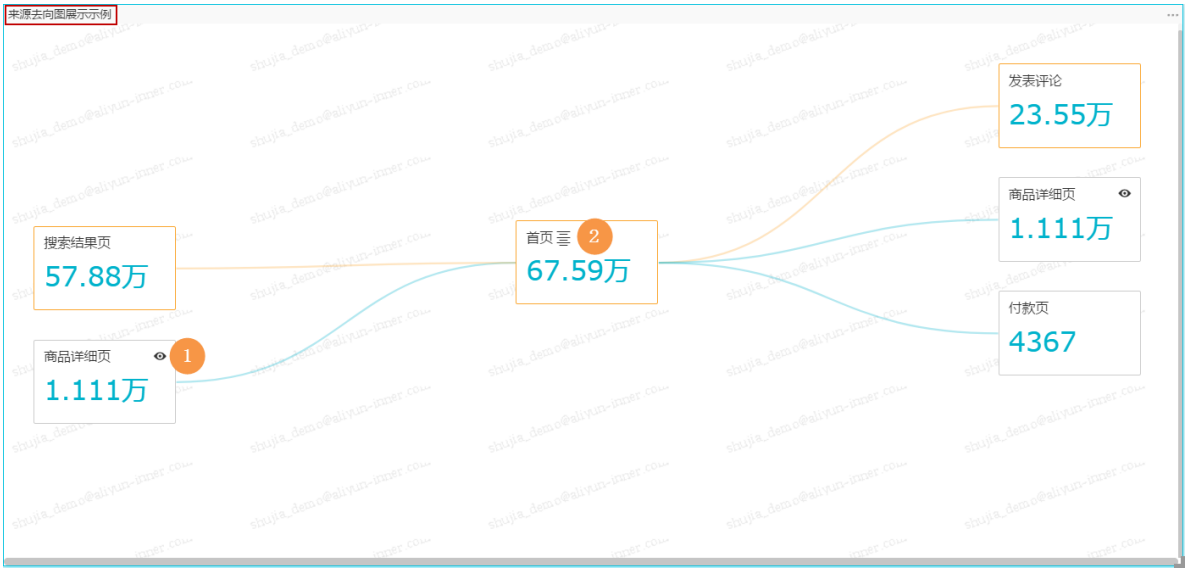


6. 单击**更新**，更新图表。
7. 单击**样式**标签页，更改图表的显示标题和布局。

   来源去向图提供了三种布局可供您选择，包括将主路径提亮或者将弹出的窗口提亮。以下示例在布局上选择了**高亮主路径**，主路径将会以不同的颜色展示在图表中，如图 *1255:* 编辑后的来源去向图所示。

   图 1255: 编辑后的来源去向图





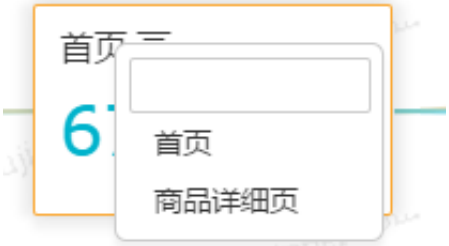
- （1）单击**查看**图标，可查看该页面的来源去向；如果方框中没有查看图标，则代表该页面没有来源去向可查，如图 *1256:* 查看页面来源去向所示。

图 1256: 查看页面来源去向



- （2）鼠标指向菜单展示图标，可选择当前需要分析的页面，如图 *1257: 切换分析页面*所示。

  图 1257: 切换分析页面



8. 单击**保存**图标，输入一个仪表板名称。
9. 单击**确定**，保存仪表板。

   系统会默认将仪表板保存到**我的仪表板**中。

31.4.5 查询仪表板

操作步骤

1. 在 Quick BI 控制台，单击**作品**图标。
2. 选择**仪表板**。
3. 在搜索框中输入仪表板关键字，查询仪表板。

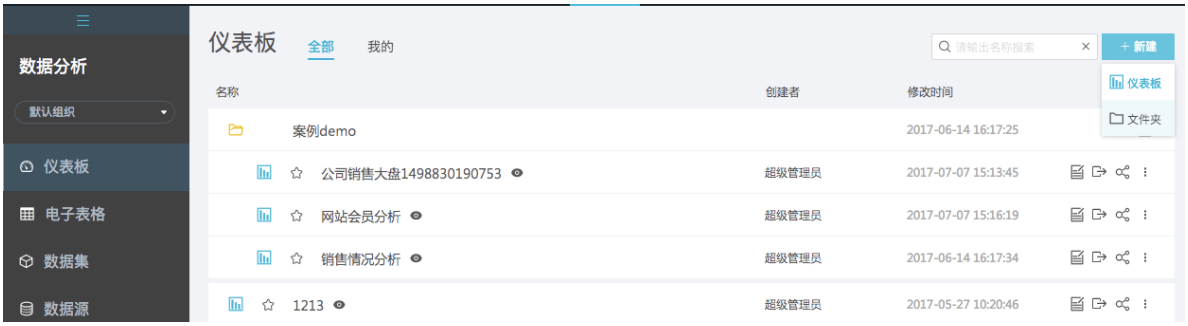
31.4.6 新建仪表板文件夹

操作步骤

1. 在仪表板列表页面，单击**新建**。

2. 选择**文件夹**，如图 *1258:* 新建文件夹所示。

图 1258: 新建文件夹



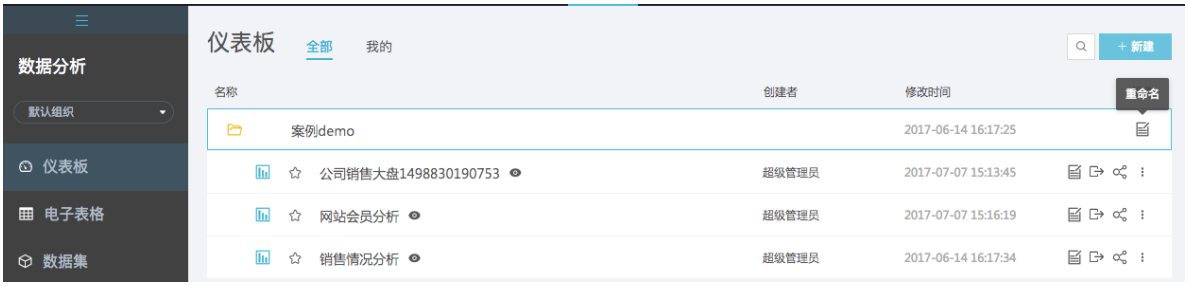
3. 输入一个文件夹名称，单击页面空白处即可。

31.4.7 重命名仪表板文件夹

操作步骤

1. 在仪表板列表中，选择一个仪表板文件夹。
2. 单击**重命名**，如图 *1259:* 重命名仪表板文件夹所示。

图 1259: 重命名仪表板文件夹



3. 修改文件夹名称，单击页面空白处即可。

31.4.8 转让仪表板

操作步骤

1. 在 Quick BI 控制台，单击**仪表板**图标。
2. 选择一个要转让的仪表板，单击后面的省略号图标。
3. 选择**转让**选项。
4. 输入被转让人的账号，单击**转让**，如图 *1260:* 仪表板转让所示。

图 1260: 仪表板转让



31.4.9 分享仪表板

操作步骤

1. 在 Quick BI 控制台，单击**仪表板**图标。
2. 选择一个要分享的仪表板，单击后面的省略号图标。
3. 选择**分享**选项。
4. 输入被分享人的账号，同时选定一个分享的截止日期，如图 *1261: 仪表板分享*所示。

图 1261: 仪表板分享



5. 单击**保存**。

31.4.10 公开仪表板

操作步骤

1. 在 Quick BI 控制台，单击**仪表板**图标。
2. 选择一个要公开的仪表板，单击后面的省略号图标。
3. 选择**公开**选项。
4. 选择一个公开的截止日期，单击**公开**，如图 *1262:* 仪表板公开所示。

图 1262: 仪表板公开



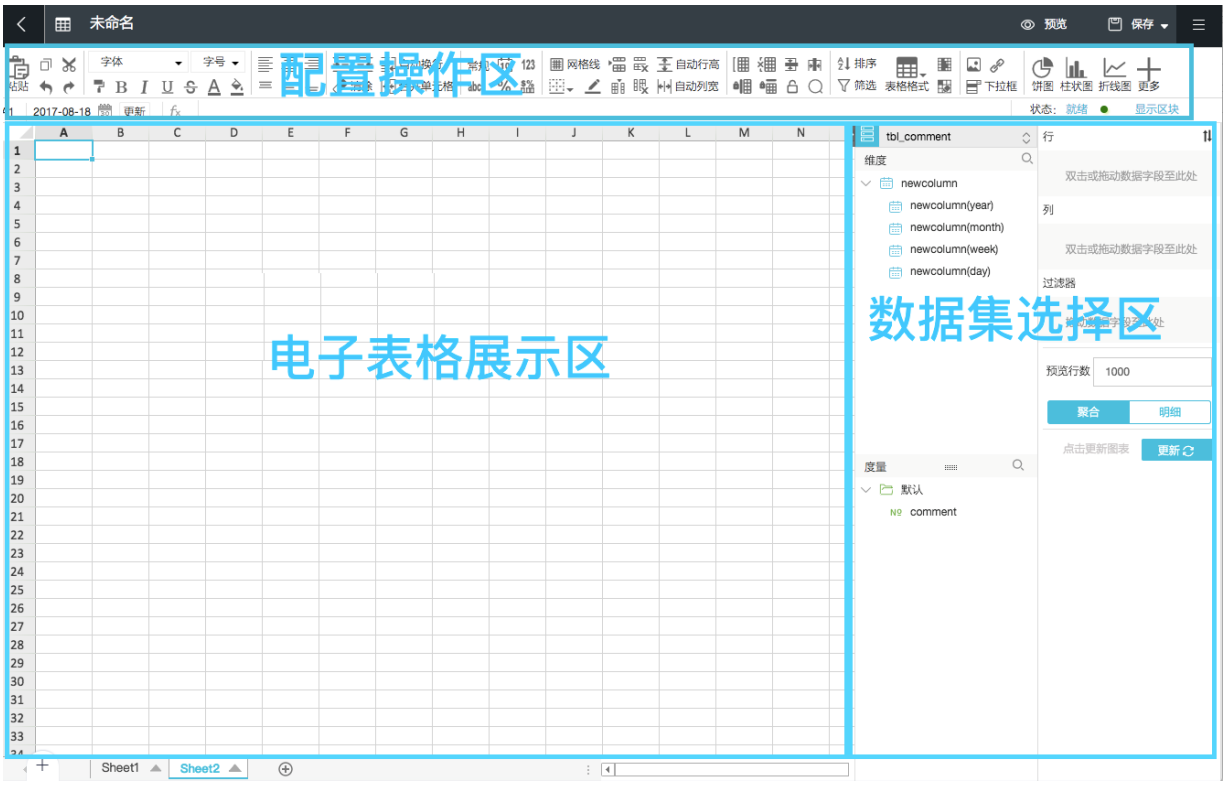
31.5 电子表格操作

本章节将为您介绍如何进入电子表格编辑页面，对具体的数据集进行数据筛选和查询，从而利用不同的数据图表对数据做可视化展示。

电子表格编辑页面可分为以下三个区域，如电子表格标记页所示。

- 数据集选择区
- 电子表格配置操作区
- 电子表格展示区

图 1263: 电子表格编辑页



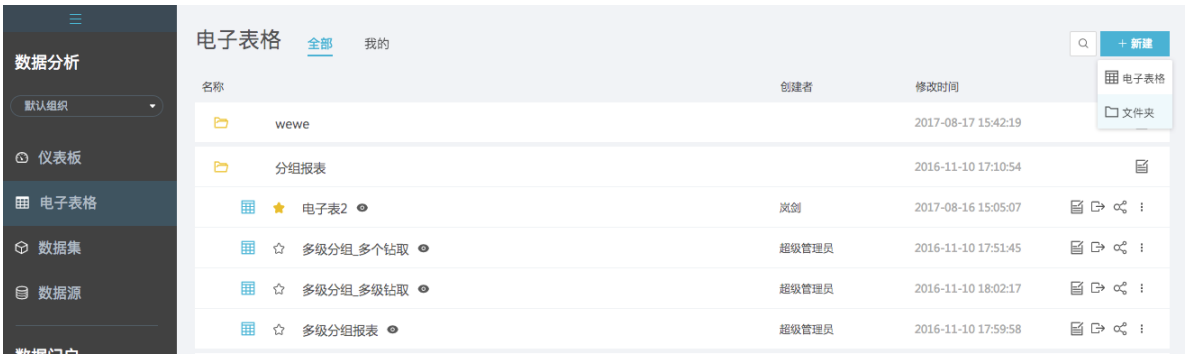
- 数据集选择区：您可以在数据集选择区内切换已有的数据集，并且每一个数据集的数据类型都会按照系统的预设，分别列在维度和度量的列表中。您可根据数据图表所提供的数据要素，在列表中选择需要的维度和度量字段。
- 电子表格配置区：您可以在电子表格配置区选择要制作的数据图表，并且根据展示需要，设置单元格的颜色、字体、数据格式等多种操作。
- 电子表格展示区：您可以在电子表格展示区，按照单元格展示，和引用数据，完成数据的再加工。

31.5.1 新建电子表格

操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中，单击**电子表格**。
3. 单击**新建**，打开新建菜单。
4. 单击**电子表格**，进入电子表格编辑页面，如图 *1264:* 新建电子表格所示。

图 1264: 新建电子表格



31.5.2 切换数据集

在数据集选择区，您可以选择或切换数据集。

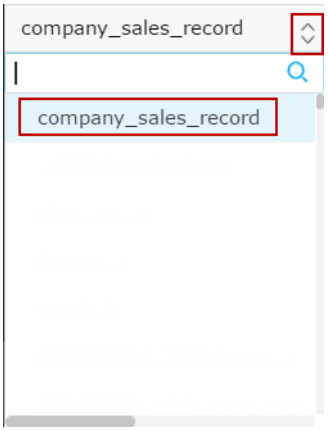
背景信息

如果您找不到需要的数据集，可以点击左侧导航栏中的**数据集**，返回到数据集列表，检查数据集是否创建成功。创建数据集，请参阅，创建数据集。

操作步骤

1. 单击数据集切换图标。
2. 在下拉列表中，选择或搜索需要分析的数据集，如图 *1265:* 切换数据集所示。

图 1265: 切换数据集



31.5.3 搜索维度字段和度量字段

您可以在维度和度量区域搜索需要的字段。

背景信息

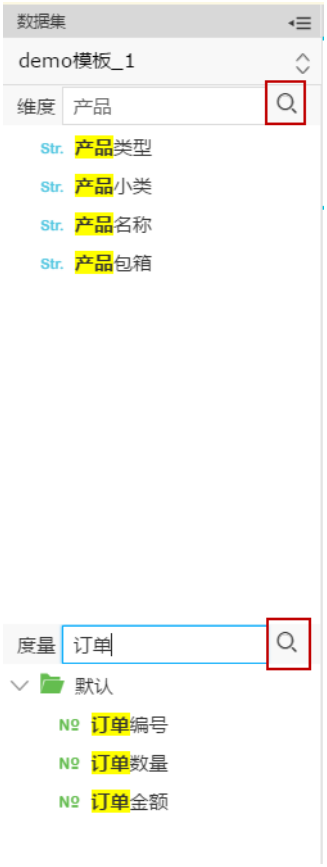
选择好数据集后，系统会自动将不同类型的列分别显示在维度和度量区域。

编辑维度字段和度量字段，请参阅，编辑维度和编辑度量。

操作步骤

1. 手动输入字段的关键字。
2. 单击放大镜图标，，搜索相关字段。

图 1266: 搜索字段



31.5.4 设置字体

您可以通过设置字体功能调整选中文本的字体，包括字体样式、字体大小，字体颜色、背景颜色等。

操作步骤

设置字体样式

1. 登录 Quick BI 管控台。
2. 在左侧导航栏中，单击**电子表格**。
3. 在列表中，单击电子表格名称，进入电子表格编辑页面。

新建电子表格，请参阅，新建电子表格。

4. 在字体区域，单击下拉箭头。
5. 在下拉菜单中，选择合适的字体样式，如图 *1267:* 字体样式所示。

图 1267: 字体样式



设置字体大小

6. 在字号区域，单击下拉箭头。
7. 在下拉菜单中，选择合适的字体大小，如图 *1268:* 字体大小所示。

图 1268: 字体大小



电子表格有设置反显功能。如果字体的样式设置为粗体，那么在字体设置框里，粗体的标识 B 为反写；如果字体的颜色设置为红色，那么字体设置框里，字体颜色标识 A 下面的横线显示为红色。

31.5.5 设置对齐方式

您可以通过对齐方式功能调整文本的格式布局。

操作步骤

1. 登录 Quick BI 管控台。
2. 在左侧导航栏中，单击**电子表格**。
3. 在列表中，单击电子表格名称，进入电子表格编辑页面。

   新建电子表格，请参阅，新建电子表格。
4. 单击对齐方式图标，调整表格的文本布局，如图 *1269:* 对齐方式所示。

图 1269: 对齐方式



31.5.6 设置文本格式和数字格式

您可以通过设置格式功能设置表格文本或数字的显示格式。

操作步骤

1. 登录 Quick BI 管控台。
2. 在左侧导航栏中，单击**电子表格**。
3. 在列表中，单击电子表格名称，进入电子表格编辑页面。

   新建电子表格，请参阅，新建电子表格。
4. 单击格式图标，调整文本或者数字的显示格式，如图 *1270:* 显示格式所示。

   图 1270: 显示格式

   - 常规：常规单元格格式不包含任何特定的数字格式。
   - 日期：推荐日期格式为 YYYY-MM-DD。
   - 数字：默认靠右显示，保留两位小数，无千分位分隔符，超出列宽的数字用 ### 填充。双击列宽，可以完整展示数字，您调整列宽到所需宽度也可以完整展示数字。
   - 文本：默认靠左；若字符数超过列宽，则显示 abc...，鼠标悬浮上去的时候显示完整字符串。双击列宽可以完整展示字符串，您手动调整列宽到所需宽度也可以完整展示字符串。
   - 百分比：默认靠右显示，保留两位小数，无千分位分隔符，超出列宽的数字用 ### 填充。双击列宽，可以完整展示数字，您调整列宽到所需宽度也可以完整展示数字。

31.5.7 设置样式、单元格和窗口

您可以通过样式、单元格和窗口的设置功能调整表格的网格线展示、边框样式、单元格高度等。

操作步骤

1. 登录 Quick BI 管控台。

2. 在左侧导航栏中，单击**电子表格**。
3. 在列表中，单击电子表格名称，进入电子表格编辑页面。

   新建电子表格，请参阅，新建电子表格。
4. 单击样式、单元格和窗口图标，调整表格的整体样式，如图 *1271:* 样式、单元格和窗口所示。

   图 1271: 样式、单元格和窗口



- 网格线：默认显示。单击**网格线**可以去除网格底纹。
- 边框：支持上边框、下边框、左边框、右边框、无边框、全边框、外边框和粗边框；支持边框的颜色设置。
- 线条颜色：支持任意线条单独划颜色及线型单独选择。
- 支持插入行、删除行、插入列、删除列，电子表格全局操作的插入和删除。
- 自动列宽：双击列宽图标，系统自动调整列宽。
- 自动行高：双击行高图标，系统自动调整行高。

31.5.8 设置图片、链接和下拉框

您可以通过图片、链接和下拉框功能为表格插入图片、相关链接和下拉框。

操作步骤

1. 登录 Quick BI 管控台。
2. 在左侧导航栏中，单击**电子表格**。
3. 在列表中，单击电子表格名称，进入电子表格编辑页面。

   新建电子表格，请参阅，新建电子表格。

插入图片

4. 单击图片图标。
5. 单击**选择文件**，选择图片。
6. 选择完成后，单击**确定**，插入图片。

插入超链接

7. 单击链接图标。

8. 输入链接地址和链接显示名称，如图 *1272:* 插入超链接所示。

图 1272: 插入超链接



9. 单击**确定**，插入超链接。

插入下拉框

10.单击下拉框图标。

11.输入条目标签和条目值，如图 *1273:* 下拉框设置所示。

图 1273: 下拉框设置



12.单击**确定**，插入下拉框。

31.5.9 设置表格格式

您可以通过表格格式功能调整电子表格的显示格式。

操作步骤

1. 登录 Quick BI 管控台。
2. 在左侧导航栏中，单击**电子表格**。
3. 在列表中，单击电子表格名称，进入电子表格编辑页面。

   新建电子表格，请参阅，新建电子表格。
4. 单击**表格格式**图标，打开格式列表。
5. 单击格式图标，选择适合的表格格式，如图 *1274:* 表格格式所示。

   图 1274: 表格格式



31.5.10 设置条件规则

您可以通过条件规则功能设置表格的内容，例如高亮某些数值，或者为数值添加代表上涨或下跌的箭头图标。

操作步骤

1. 登录 Quick BI 管控台。
2. 在左侧导航栏中，单击**电子表格**。
3. 在列表中，单击电子表格名称，进入电子表格编辑页面。

   新建电子表格，请参阅，新建电子表格。

高亮

4. 打开条件规则菜单。
5. 单击**高亮**标签页，如图 *1275:* 高亮所示。

   图 1275: 高亮



- 规则：单击下拉箭头，在下拉菜单中选择您需要的高亮规则，并在后面的方框中手动输入数字。
- 前景色：单击色块，选择您需要的前景色。
- 背景色：单击色块，选择您需要的背景色。
- 预览：手动输入需要预览的内容来预览高亮的效果。

6. 设置完成后，单击**确定**。

数据栏

7. 单击**数据栏**标签页，如图 *1276: 数据栏*所示。

图 1276: 数据栏



- 最小值：手动输入最小数值。
- 最大值：手动输入最大数值。
- 颜色：单击色块选择您需要的颜色。

8. 设置完成后，单击**确定**。

图标

9. 单击**图标**标签页，如图 *1277:* 图标所示。

图 1277: 图标



单击下拉箭头，在下拉菜单中选择您需要的数字符号，并在后面的方框内手动输入数值。在符合设定范围内的数字旁边，会自动添加绿色、黄色和红色三种箭头。

10. 设置完成后，单击**确定**。

31.5.11 查询电子表格

操作步骤

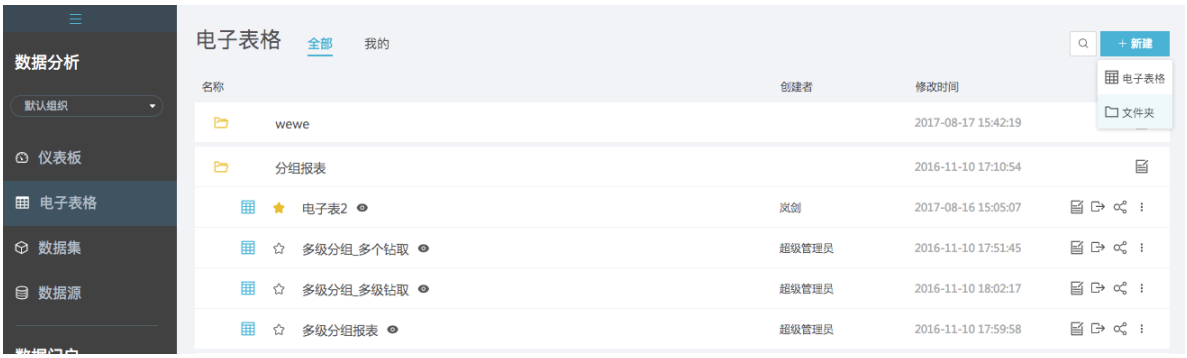
1. 登录 Quick BI 管控台。
2. 在左侧导航栏中，单击**电子表格**。
3. 在搜索框中输入电子表格关键字。
4. 单击搜索图标，查询电子表格。

31.5.12 新建电子表格文件夹

操作步骤

1. 登录 Quick BI 管控台。
2. 在左侧导航栏中，单击**电子表格**。
3. 单击**新建** > **文件夹**如图 *1278:* 新建文件夹所示。

图 1278: 新建文件夹



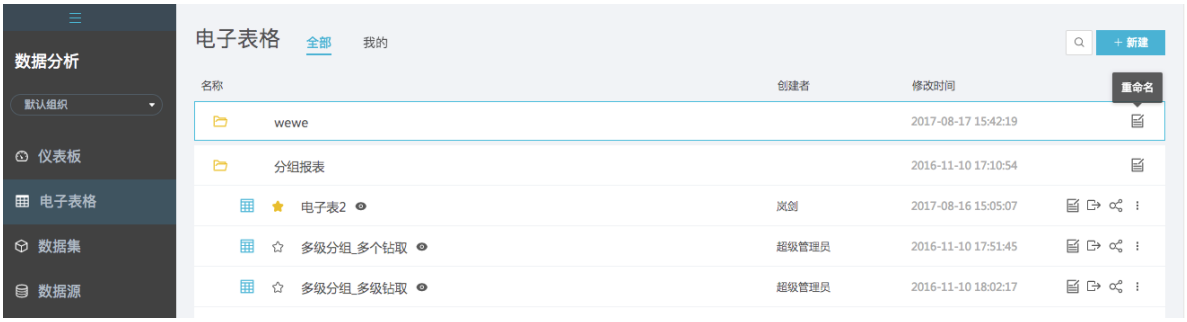
4. 输入一个文件夹名称，单击页面空白处即可。

31.5.13 重命名电子表格文件夹

操作步骤

1. 登录 Quick BI 管控台。
2. 在左侧导航栏中，单击**电子表格**。
3. 在列表中，选择一个电子表格。
4. 单击**重命名**图标，如图 *1279:* 重命名文件夹所示。

图 1279: 重命名文件夹



31.5.14 转让电子表格

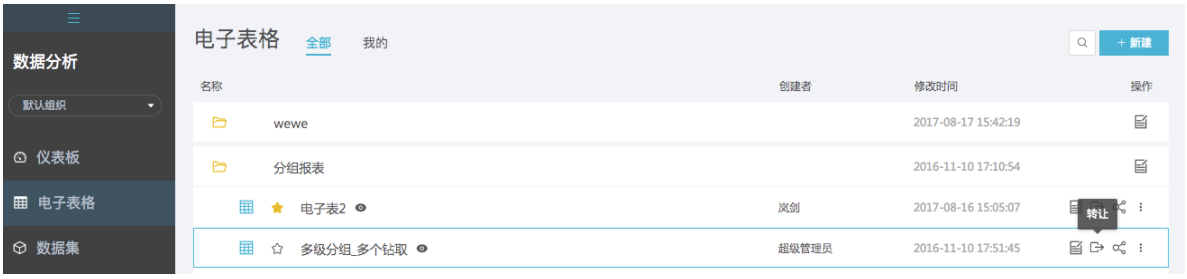
制作好的电子表格可以转让给他人。

操作步骤

1. 登录 Quick BI 管控台。
2. 在左侧导航栏中，单击**电子表格**。
3. 在列表中，选择一个电子表格。

4. 单击**转让**图标，如图 *1280:* 表格转让所示。

图 1280: 表格转让



5. 手动输入被转让人的账号，单击**转让**。

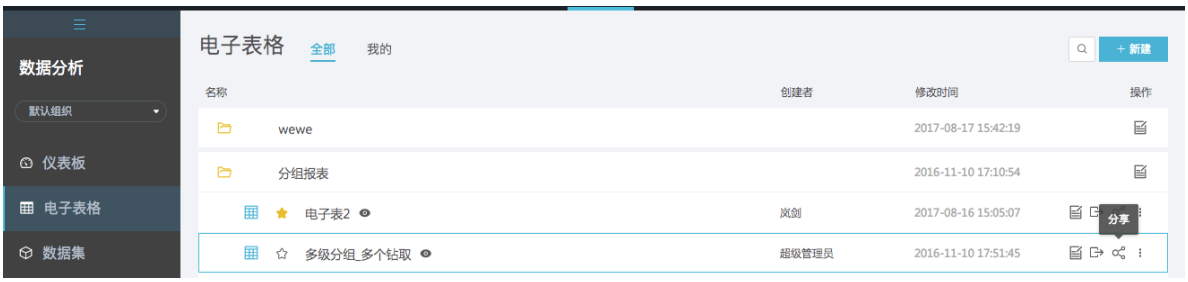
31.5.15 分享电子表格

制作好的电子表格可以分享给他人。

操作步骤

1. 登录 Quick BI 管控台。
2. 在左侧导航栏中，单击**电子表格**。
3. 在列表中，选择一个电子表格。
4. 单击**分享**图标，如图 *1281:* 分享电子表格所示。

图 1281: 分享电子表格



5. 手动输入被分享人的账号。
6. 选择一个分享的截止日期。
7. 单击**保存**，完成表格分享。

31.5.16 公开电子表格

制作好的电子表格可以公开给他人。

操作步骤

1. 登录 Quick BI 管控台。
2. 在左侧导航栏中，单击**电子表格**。
3. 在列表中，选择一个电子表格。
4. 单击**公开**图标。
5. 选择一个公开的截止日期。
6. 勾选**生成新链接**，单击**公开**。

31.6 数据门户搭建

数据门户也叫数据产品，是通过菜单形式组织的仪表板的集合。通过数据门户您可以制作复杂的带导航菜单的专题类分析。

31.6.1 新建数据门户

操作步骤

1. 登录 Quick BI 管控台。
2. 在左侧导航栏中，单击**数据门户** > **门户**。
3. 在数据门户列表页，单击**新建** > **数据门户**，如图 *1282:* 新建数据门户所示。

图 1282: 新建数据门户



31.6.2 设置模板

在模板设置的标签页中，可以设置模板的标题、上传 logo 以及编辑页脚。

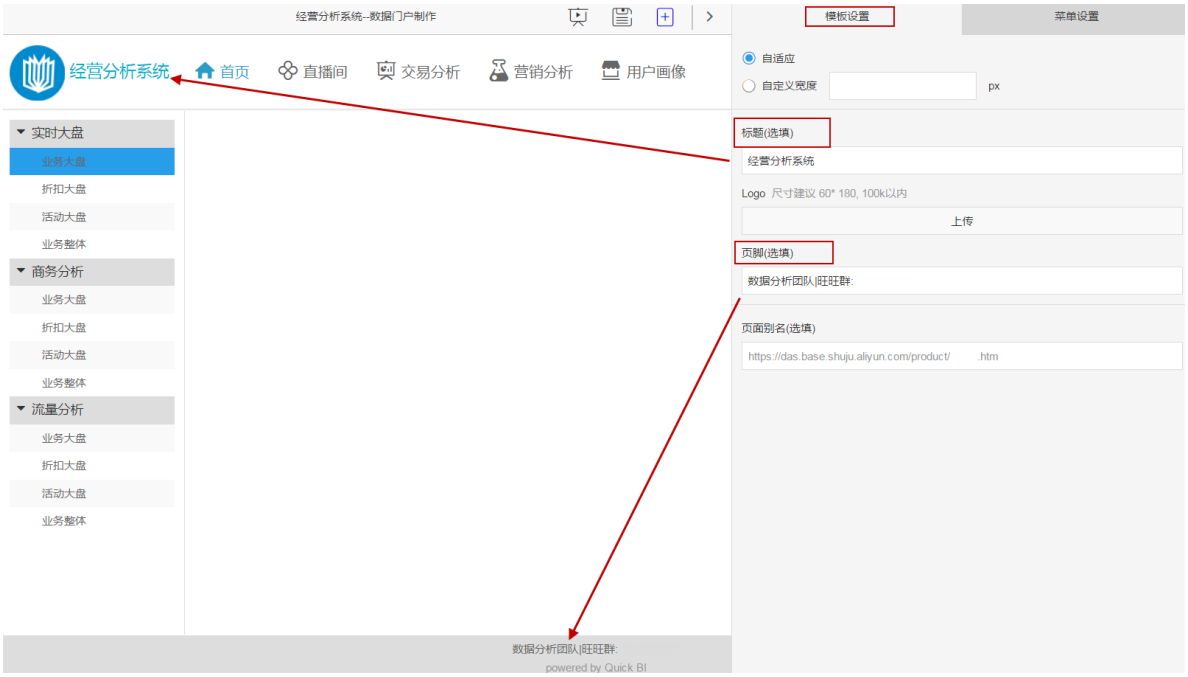
操作步骤

1. 登录 Quick BI 管控台。

2. 在左侧导航栏中，单击**数据门户** > **门户**。
3. 在数据门户列表页，单击一个数据门户名称。

   新建数据门户，请参阅，新建数据门户。
4. 单击**模板设置**标签页，编辑门户模板，如图 *1283:* 模板设置所示。

图 1283: 模板设置



31.6.3 设置菜单

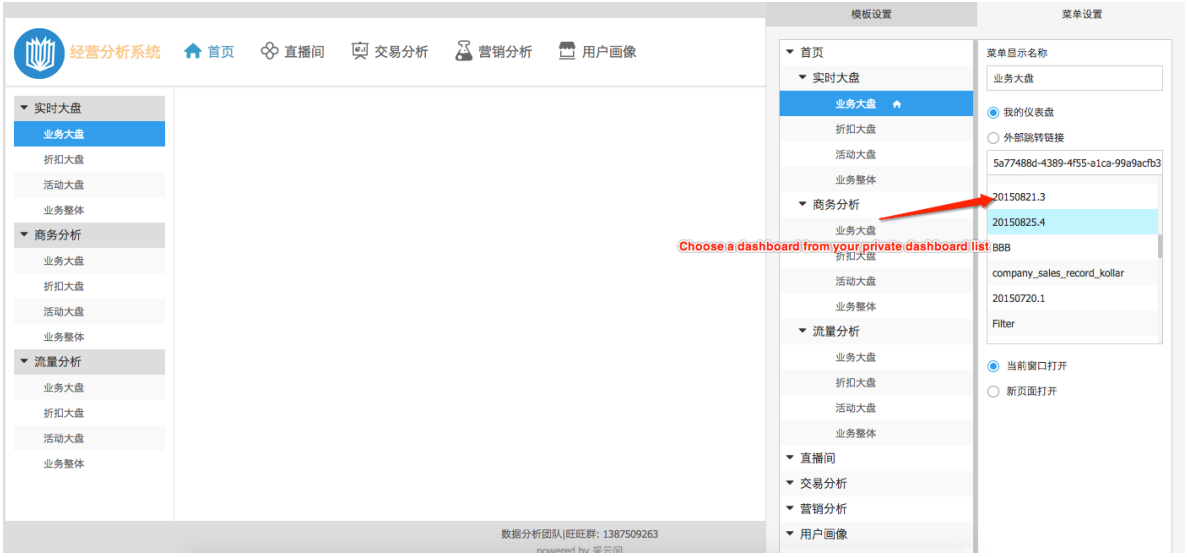
在菜单设置的标签页中，可以设置菜单名称、单击菜单要打开的 URL 等信息。

操作步骤

1. 登录 Quick BI 管控台。
2. 在左侧导航栏中，单击**数据门户** > **门户**。
3. 在数据门户列表页，单击一个数据门户名称。

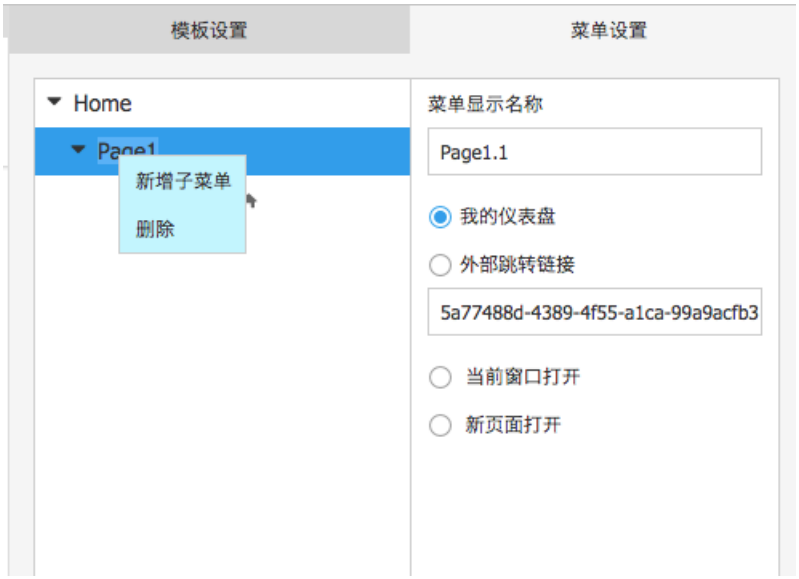
   新建数据门户，请参阅，新建数据门户。
4. 单击**菜单设置**标签页，编辑门户模板，如图 *1284:* 菜单设置所示。

图 1284: 菜单设置



- 鼠标右键单击菜单名称可以打开菜单编辑列表，如图 *1285:* 编辑菜单结构所示。

图 1285: 编辑菜单结构



- 支持对仪表板和工作表的引用。

31.7 组织管理

本章节将为您介绍如何创建、管理组织，以及如何创建、管理群空间。

组织一般指中小型企业，事业单位，学校院系或大型公司的部门。

如果您的组织人数较多，并且需要多人协同配合完成数据分析，并且对数据安全有比较高的要求，那么 Quick BI 可以协助您满足以下几个需求：

- 不同的部门看不同的报表同一张报表
- 不同角色的员工看到的内容不一样

在组织层面，只有两种组织用户：管理员和普通成员。

31.7.1 创建组织

操作步骤

1. 在 Quick BI 控制台，单击**管理**图标，进入管理页面。
2. 在组织权限设置中，选择**组织管理**，进入组织管理页面。
3. 勾选**同意**，并单击**开通组织**。
4. 在新建组织页面，输入一个组织名称，如图 *1286:* 创建组织所示。

图 1286: 创建组织



说明:

- 只有当前没有开通加入过任何组织的阿里云主账号才允许创建组织。
- 一个阿里云账号最多只允许开通或加入一个组织。

31.7.1.1 修改组织信息

组织管理员可以对组织的属性进行再次编辑：

图 1287: 修改组织信息



组织成员管理工作负责把需要在该组织中进行协同工作的阿里云用户加到组织中。

群空间管理工作负责把组织中的成员按照其工作范围和职责放置到不同的群空间中。群空间可以和组织下真实的业务部门一一对应起来，组织管理员可以先按照真实业务情况，如果该组织有销售部门和人力资源部门，就对应的创建销售群和人力资源群，然后把销售部的员工加入到销售群，人力资源部的员工加入到人力资源群中。

组织成员管理功能模块，只有组织管理员才有权限访问。组织创建者默认就是组织管理员。

在组织层面，只有两种组织用户：管理员和普通成员。

31.7.1.2 退出组织

在组织配置页面，单击**退出组织**可退出当前组织，如图 *1288:* 退出组织所示。

图 1288: 退出组织

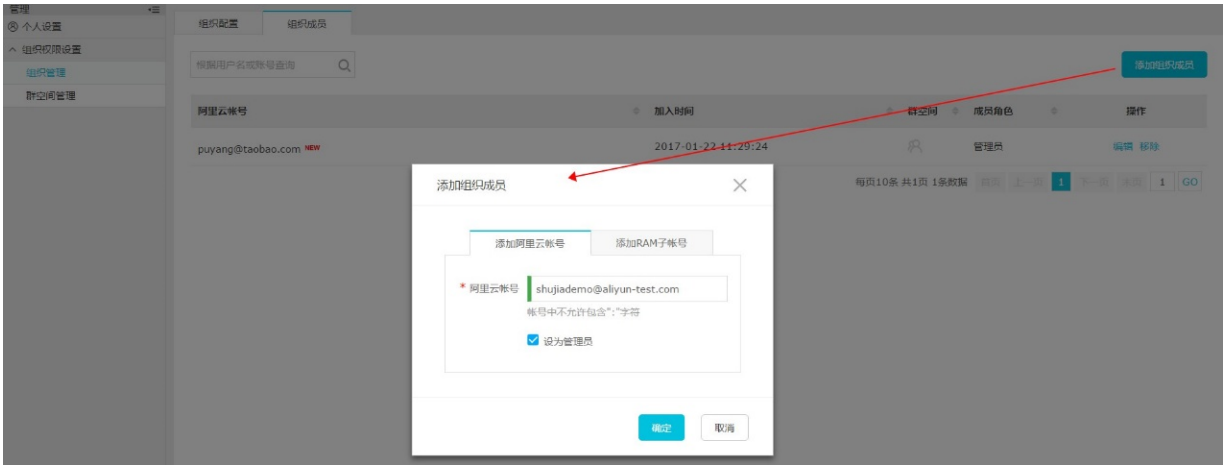


31.7.1.3 添加组织成员

通过阿里云主账号添加组织成员。

说明: 必须填写真实存在并且真实登录过阿里云网站的阿里云账号。

图 1289: 添加组织成员



通过RAM子账号添加组织成员。

图 1290: 添加组织成员



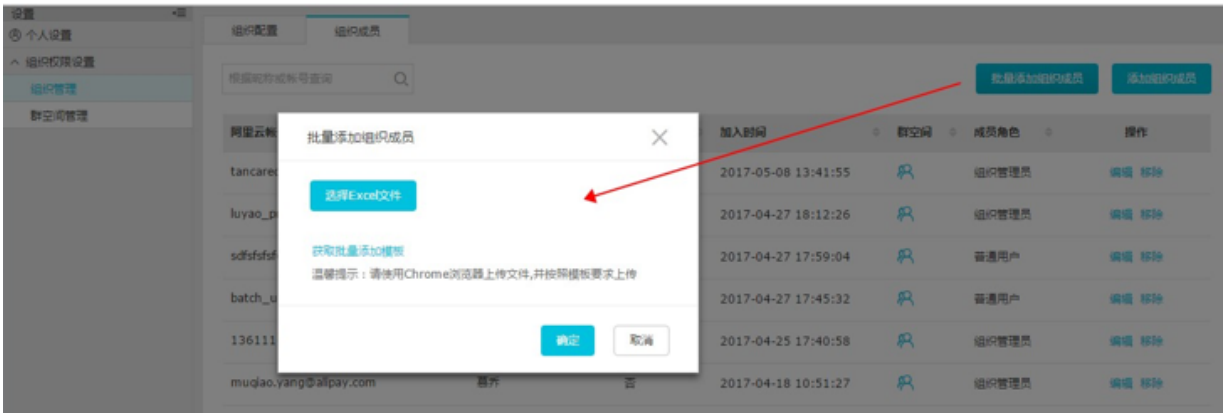
说明: 如果该用户已经被其他组织吸纳，系统会给出如下提示：

图 1291: 添加成员失败提示



批量添加用户

图 1292: 批量添加用户



31.7.1.4 编辑组织成员

设置成员在组织中的角色：组织管理员还是普通成员；还可以设置成员的昵称便于查找成员。

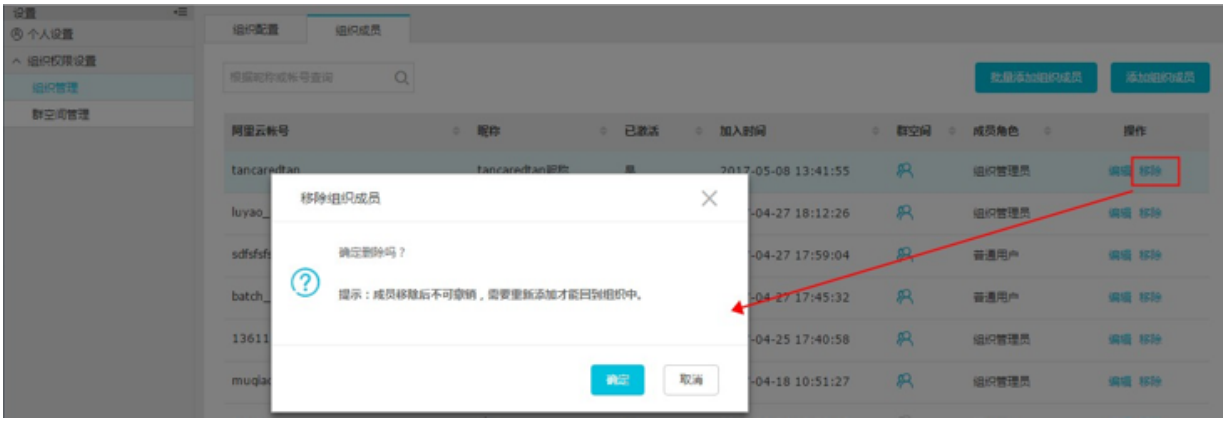
图 1293: 编辑组织成员



31.7.1.5 移除组织成员

管理员有权移除组织成员。

图 1294: 移除组织成员



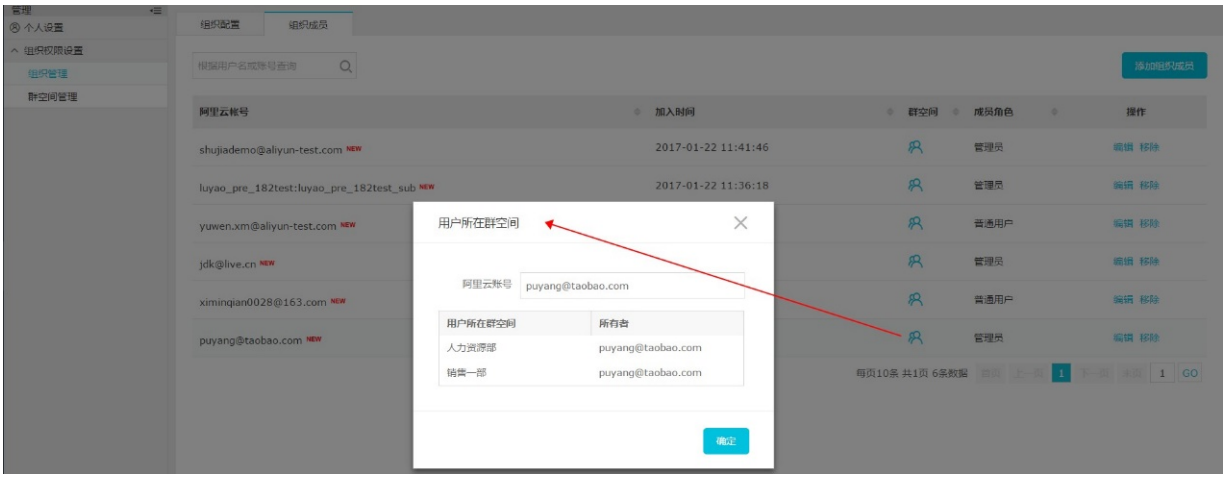
说明: 移除组织成员时，若组织成员仍在某群空间中，需要先将该成员从群空间中移除，否则系统会给出如下提示：

图 1295: 移除成员失败提示



31.7.1.6 查看用户所在的群空间

图 1296: 查看用户所在的群空间



31.7.1.7 查询组织成员

可以通过昵称或者账号查询组织成员，如图 *1297:* 查询组织成员所示。

图 1297: 查询组织成员



31.7.2 群空间管理

群空间管理工作由群空间管理员来完成，群空间管理员是由创建群的组织管理员来任命的。群空间管理员也可以把群中的其他成员升级为群管理员。

群空间管理包括以下工作：

- 新建群空间
- 编辑群空间
- 设置默认群空间

31.7.2.1 什么是群空间

群空间是组织成员进行协同开发的工作空间，在群空间中，群成员按不同的角色一起相互配合完成数据源和数据集的创建和修改，工作表、仪表板和数据门户的创建和修改。 这些数据对象生存在其所在的群空间中，不同的群空间中有不同的数据对象。

31.7.2.2 个人空间和群空间的差异

标准版中，用户工作的空间叫个人空间。 个人空间和群空间的主要差异表现在：

- 个人空间在用户首次登录时自动创建，而群空间需要组织管理员手动创建。
- 个人空间无法新建/删除。
- 个人空间不允许添加其他用户，也自然没协作共享的功能。
- 个人工作空间中数据对象的转让和分享范围为任何一个阿里云Quick BI用户，而群空间中数据对象的转让范围是群空间内部，分享范围也是组织内部。

31.7.3 新建群空间

操作步骤

1. 在组织权限管理页面，选择**群空间管理** > **新建群空间**。
2. 在新弹出的页面中，输入一个群空间名称，然后单击**确定**。

图 1298: 新建群空间



31.7.4 修改群空间

群空间有以下主要属性：

- 群空间名称
- 群空间说明
- 功能权限：工作表可导出，默认允许，改为不允许后该群空间中数据无法导出下载。仪表板可公开，默认允许，改为不允许后该群空间中的仪表板不能公开给所有人使用。作品可分享，默认允许，改为不允许后该群空间中的作品不能被分享给群外的人。
- 偏好设置：使用物理字段名作为维度、度量名称 还是使用字段注释作为维度、度量名称。修改后影响该群空间中新数据集创建时候的名称默认设置，但原有的不受影响。

图 1299: 修改群空间

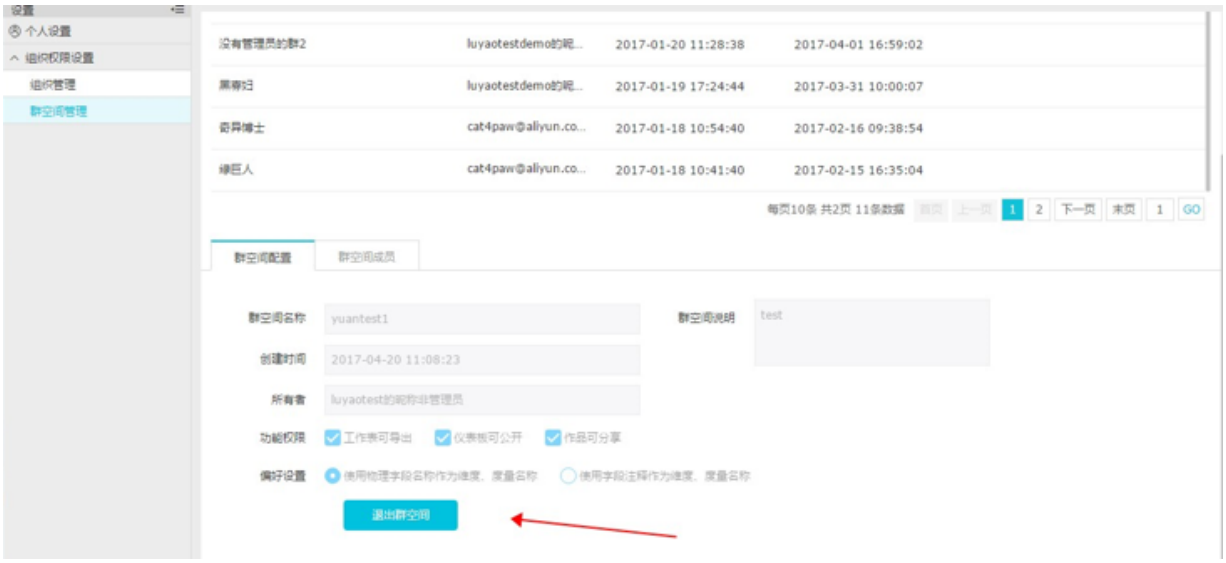


说明: 个人空间只允许空间 owner 修改参数配置，而群空间只有群空间管理员才可以修改群空间的参数配置。

31.7.5 退出群空间

在群空间管理页面，单击**退出群空间**可退出当前群空间。

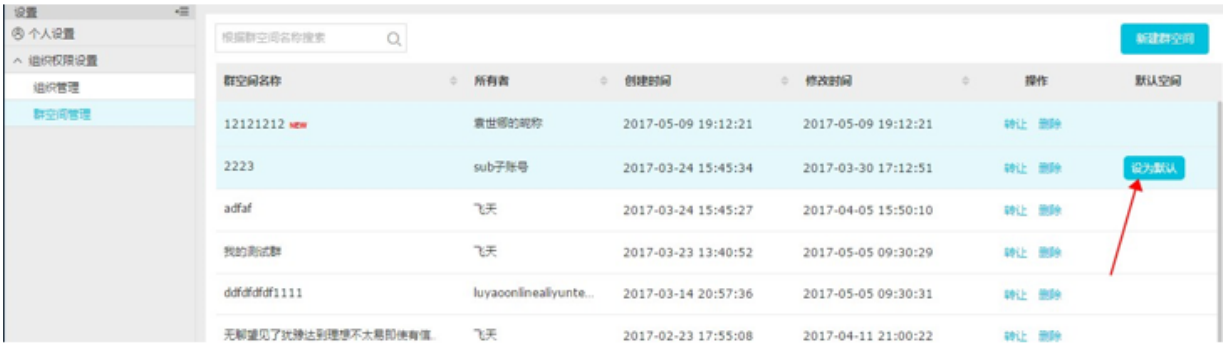
图 1300: 退出群空间



31.7.6 设为默认群空间

鼠标指向群空间列表时，会出现**设为默认**按钮，单击可以修改默认空间。

图 1301: 设为默认群空间



31.7.7 转让群空间

如果一个群空间的创建者想从组织中被移除，可以通过转让群空间操作将拥有的群空间转给组织中的其他成员。被转让人不需要为组织管理员，任何普通成员都可以担任被转让人。

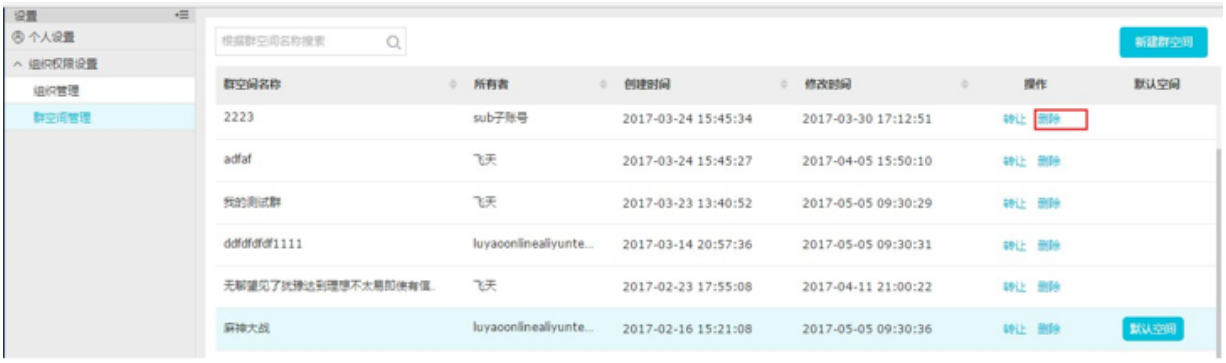
图 1302: 转让群空间



31.7.8 删除群空间

在群空间列表中，选择一个群空间，单击**删除**，可删除该群空间。

图 1303: 删除群空间



31.7.9 添加群空间成员

单击**添加群空间成员**，在对话框中输入成员账号，然后单击**确定**。

图 1304: 添加群空间成员



用户输入账号进行组织内模糊匹配，并分配相应角色。不同角色拥有不同的视角和权限点，而同一用户可以拥有不同角色，但至少要拥有一个角色。

角色列表包括：群管理员、开发者、分析师、阅览者。

角色和权限的映射列表

角色和权限的映射是固化的，不可修改，在给用户授权时，只需要指定该用户所属的角色即可。角色和权限的映射列表如下：

- 功能导航入口

  表 114: 功能导航入口

权限点	开发者	分析师	阅览者
数据	有	无	无
工作表	有	有	有
仪表板	有	有	有
数据门户	有	有	有

- 数据

表 115: 数据

权限点	开发者	分析师	阅览者
创建数据源	有	无	无
修改数据源	只能修改自己的数据源	无	无
删除数据源	只能删除自己的数据源	无	无
转让数据源	只能转让自己的数据源	无	无
使用数据源	有	无	无
创建数据集	有	无	无
修改数据集	只能修改自己的数据集	无	无
删除数据集	只能删除自己的数据集	无	无
转让数据集	只能转让自己的数据集	无	无
使用数据集	有	有	无

- 电子表格

表 116: 电子表格

权限点	开发者	分析师	阅览者
创建电子表格	有	有	无
修改电子表格	只能修改自己的电子表格	只能修改自己的电子表格	无
删除电子表格	只能删除自己的电子表格	只能删除自己的电子表格	无
预览电子表格	有	有	有
转让电子表格	只能转让自己的电子表格	只能转让自己的电子表格	无

权限点	开发者	分析师	阅览者
分享电子表格	只能分享自己的电子表格	只能分享自己的电子表格	无
引用电子表格	有	有	无

- 仪表板

表 117: 仪表板

权限点	开发者	分析师	阅览者
创建仪表板	有	有	无
修改仪表板	只能修改自己的仪表板	只能修改自己的仪表板	无
删除仪表板	只能删除自己的仪表板	只能删除自己的仪表板	无
预览仪表板	有	有	有
转让仪表板	只能转让自己的仪表板	只能转让自己的仪表板	无
分享仪表板	只能分享自己的仪表板	只能分享自己的仪表板	无
引用仪表板	有	有	无
发布仪表板	只能发布自己的仪表板	只能发布自己的仪表板	无

- 数据门户

表 118: 数据门户

权限点	开发者	分析师	阅览者
创建数据门户	有	有	无
修改数据门户	只能修改自己的数据门户	只能修改自己的数据门户	无
删除数据门户	只能删除自己的数据门户	只能删除自己的数据门户	无
预览数据门户	有	有	有

权限点	开发者	分析师	阅览者
转让数据门户	只能转让自己的数据门户	只能转让自己的数据门户	无
分享数据门户	只能分享自己的数据门户	只能分享自己的数据门户	无

31.7.10 编辑群空间成员

选中群空间成员列表中的某一位成员，单击**编辑**，可以编辑该成员信息。

图 1305: 编辑群空间成员



31.7.11 删除群空间成员

选中群空间成员列表中的某一位成员，单击**删除**，可删除该成员。

图 1306: 删除群空间成员

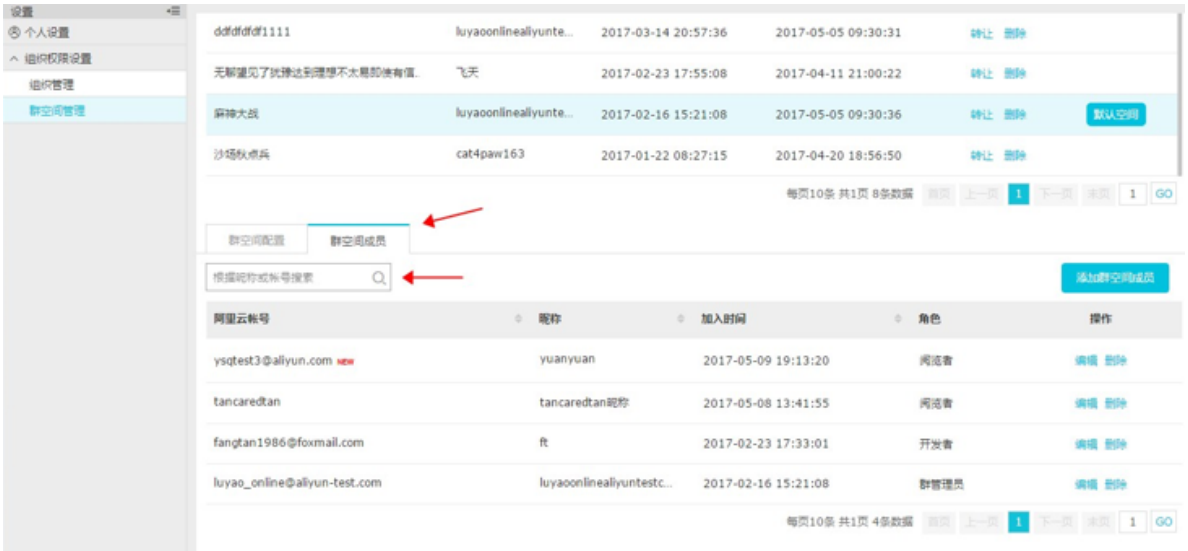


31.7.12 查询群空间成员

操作步骤

1. 在群空间管理页面，选择群空间。
2. 单击**群空间成员**，如图 *1307: 查询群空间成员*所示。

图 1307: 查询群空间成员



您也可以在搜索框中，输入成员关键字来搜索成员。

31.8 权限管理

权限管理包括数据对象管理和行级权限管理。

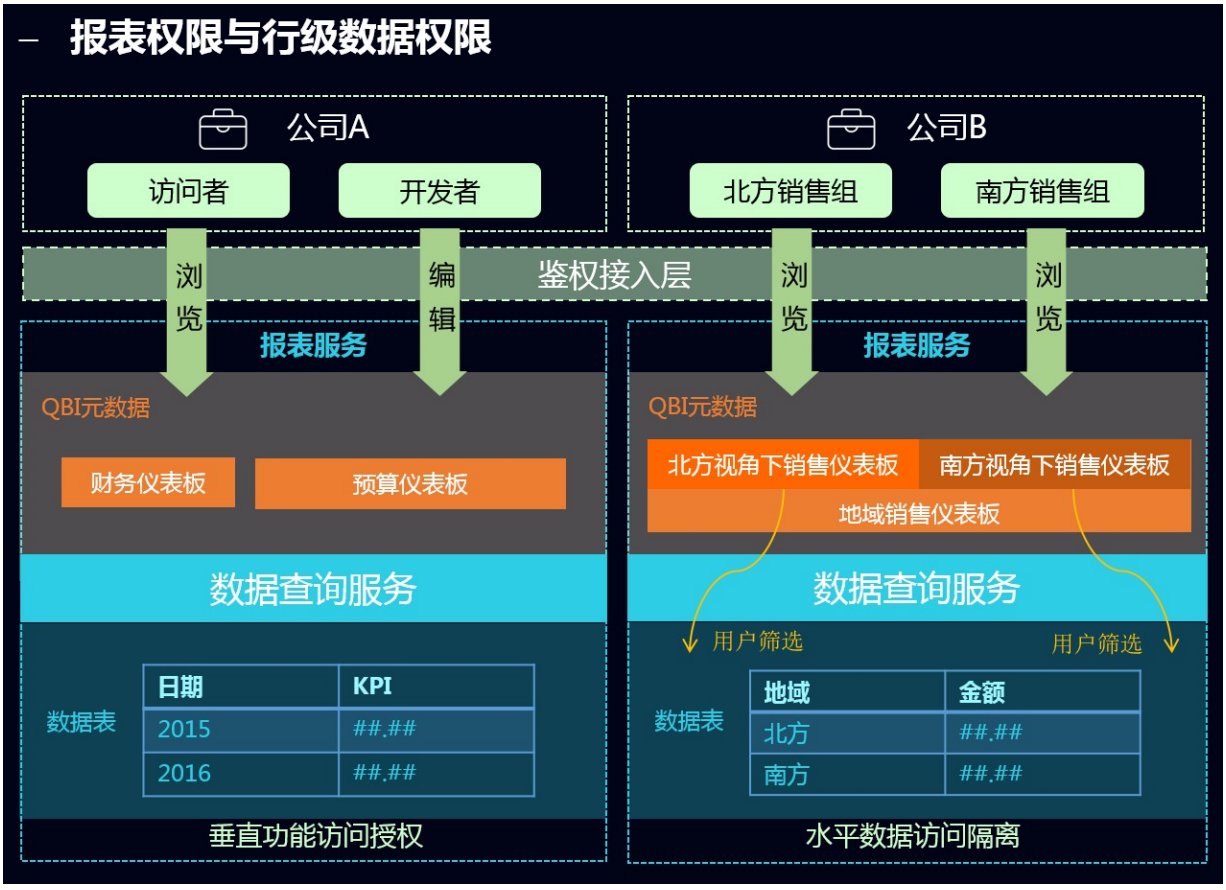
数据对象包括数据源、数据集、工作表、仪表板、数据门户等，而数据对象的管理又分个人空间下数据对象的管理和群空间下数据对象的管理。

31.8.1 数据对象管理

数据对象包括 数据源、数据集、工作表、仪表板、数据门户等。

数据对象的管理又分个人空间下数据对象的管理 和群空间中数据对象的管理。

图 1308: 数据对象管理



群空间中数据对象的转让

支持数据源、数据集、工作表、仪表板、数据门户的转让功能：

- 只有作品的作者和群管理员才有权限转让。
- 只允许在群空间内部转让。

群空间中数据对象的分享

支持工作表、仪表板、数据门户的分享功能，被分享的作品可以被其他用户以只读的方式访问，但不允许修改、删除和另存。

- 只有作品的作者和群管理员才有权限分享。
- 如果群空间属性中设置了不允许分享，则该群空间中的所有作品都无法分享。
- 只支持组织内分享，暂不支持分享到组织外的阿里云用户。

群空间内的作品默认群成员进入所在群空间就可以看到并进行阅览，如图 *1309:* 阅览群空间内作品所示。

图 1309: 阅览群空间内作品



群内的作品也可以分享给组织内的个人，该人可以不在当前作品所在的群。

被分享的人可以在**我的**页面下可以看到分享给我的作品，如图 *1310:* 阅览被分享的作品所示。

图 1310: 阅览被分享的作品



群空间中的仪表板对象也支持公开，被公开的仪表板所有人都可以访问，所以不建议您将包含业务数据的仪表板公开。

31.8.2 行级权限

详细内容，请参阅，数据集行级权限控制。

31.8.3 个人空间下数据对象管理

个人空间下的数据对象只有作者本人才能进行编辑和删除操作，其他任何人没有权限。

个人空间中数据对象的转让

支持数据集、工作表、仪表板、数据门户的转让功能。

- 只有数据对象的作者才有权限转让
- 转让范围：阿里云 Quick BI 用户

目前， Analytic DB 数据源下的数据集暂时不能转让。

个人空间中数据对象的分享

支持工作表、 仪表板、数据门户的分享功能。被分享的作品可以被其他用户以只读的方式访问，但不允许修改、删除和另存。

- 只有数据对象的作者才有权限分享
- 分享范围：阿里云 Quick BI 用户

被分享的人可以在**我的**页面下看到并阅览该作品。

个人空间中的仪表板对象也支持公开，被公开的仪表板所有人都可以访问，所以不建议您将包含业务数据的仪表板公开。

32 关系网络分析

32.1 产品概述

I+产品是基于时空关系网络的大数据可视化分析平台，其起源于支付宝，在内部广泛应用于反欺诈、反作弊、反洗钱等风控业务，目前正面向公安、税务、海关、银行、保险、互联网等行业提供整体解决方案。

I+产品围绕**大数据多源融合、计算应用、可视分析、业务智能**设计实现，结合关系网络、时空数据、地理制图学建立可视化表征，揭示对象关联及与时间、空间密切相关的模式及规律。目前产品已具备关系分析、路径分析、群集分析、共同邻居、骨干分析、伴随分析、时序分析、行为分析、统计分析、地图分析等功能，以可视化的方式有效融合计算机的计算能力和人的认知能力，获得对于大规模复杂网络的洞察力，帮助用户更为直观、高效地获取知识和信息。

I+产品是依托于关系网络，从时间、空间维度来提供交互式分析的大数据处理平台。主体业务模块有：搜索、关系网络，时空分析三大部分。同时产品主要着力于安全相关领域，权限控制也是产品重要模块。

32.1.1 搜索

搜索提供对象信息检索的功能，用户在分析过程中逐步递进拓展信息，从线索关键词开始细化分析，如“姓名”，“住址”等。搜索提供检索工具帮助用户快速定位信息，同时作为关系网络和时空分析的入口，可以将检索对象信息引入到两个模块中进行时空关系拓展分析。搜索产品模块入口如下图所示：

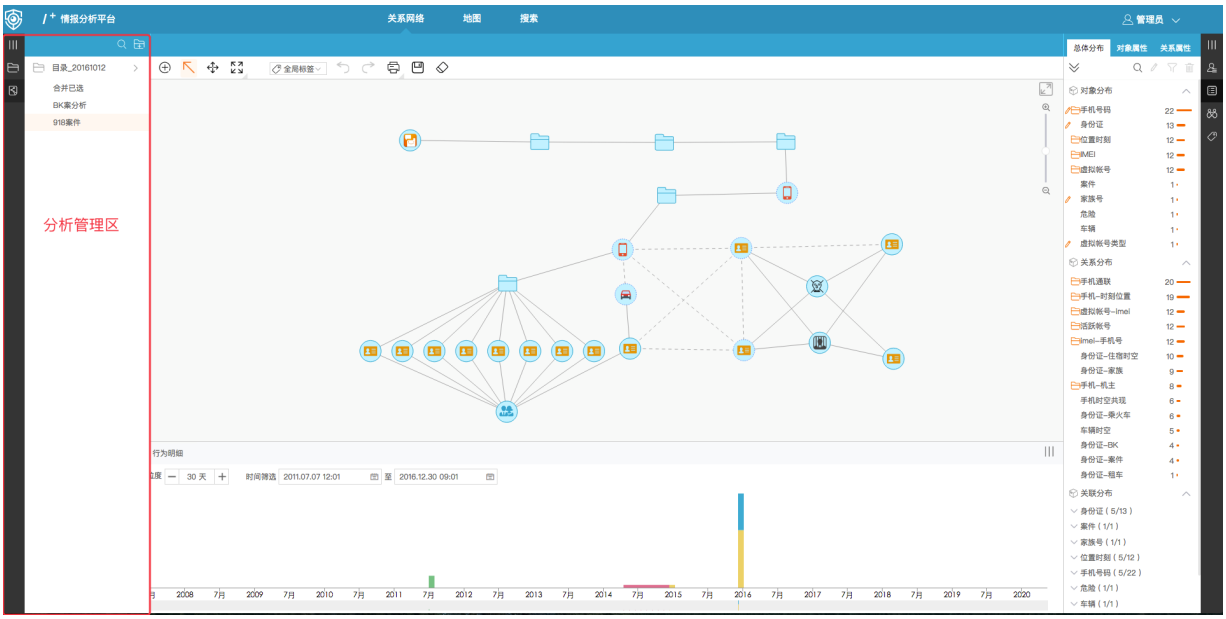
图 1311: 搜索产品

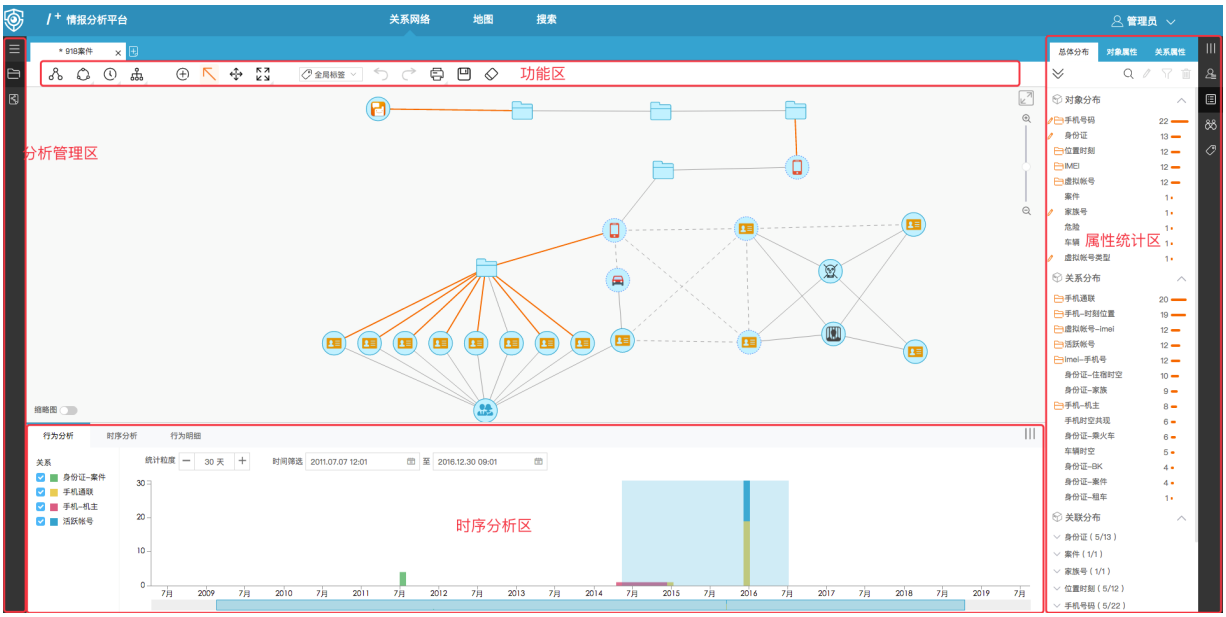


32.1.2 关系网络

关系网络是 I+ 研判分析平台的核心模块，所有实体之间的关系拓扑，业务计算，可视化布局，以及图交互操作其中。同时有三大辅助分析组件用户空间、时序分析和信息立方补充关系网络研判，使之可以涵盖大部分研判业务场景。如下是关系网络界面，包含“功能区”，“统计区”，“时序分析区”，“分析管理区”和“主分析图区”（中间部分）。其中“分析管理区”包括：“分析管理目录”、“协同分析目录”，采用收缩弹层展示，具体如下图所示：

图 1312: 关系网络

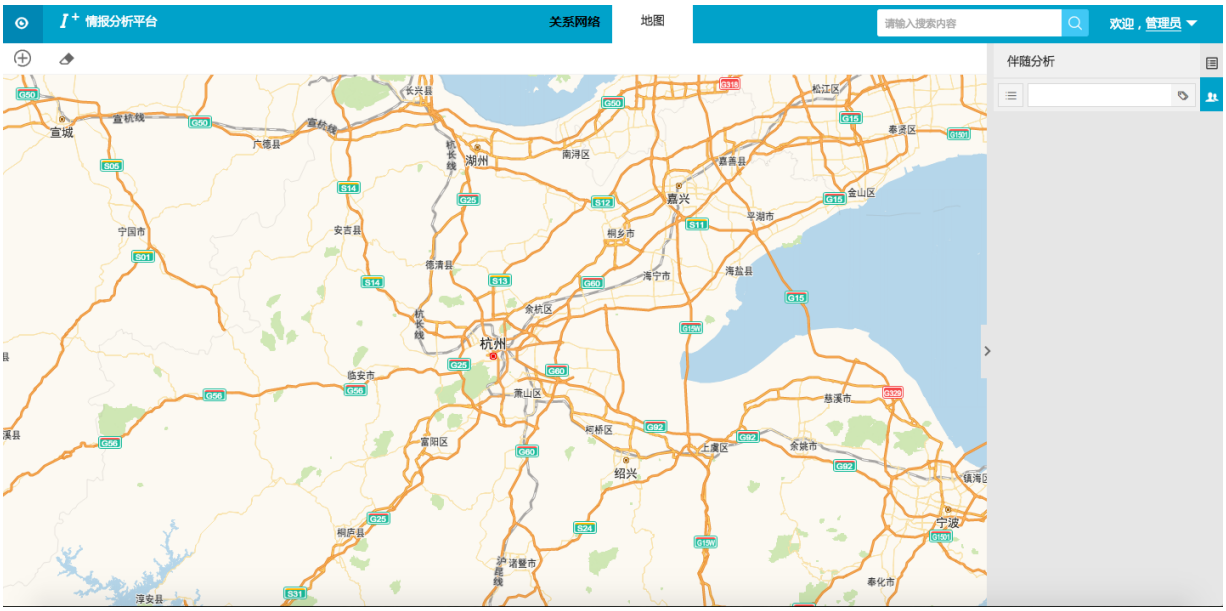




32.1.3 时空分析

地图分析引入空间维度，将实体、关联等数据和 GIS 地图结合，发掘出空间关联关系，并利用机器学习，智能计算同轨伴随、空间活动等信息。

图 1313: 时空分析



32.2 产品优势

海量数据实时挖掘

I+ 能够在百亿节点、千亿边、万亿记录的 PB 量级数据中，按照用户的业务指令进行关系挖掘和时空计算，并且实时交互响应。

模型认知万物相连

I+ 用 OLP 模型认知世界万物相连，以实体（Object）和关联（Link）显像表征，以属性（Property）实现异构数据的理解和整合。

业务场景灵活赋能

I+ 自带以 OLP 为核心的中枢控制，通过业务配置和感知实现人机交互学习，赋能公安、欺诈、金融、税务等不同场景业务研判。

可视分析高效体验

I+ 全面分析潜在用户体验要素和业务痛点，沉淀出数据、交互、结果的分阶可视化体验和协同共享，使得有证可查，有据可说。

智能深入以人为本

I+ 精准智能化帮助业务员思考学习，解决常见的业务难题，目前已有涉恐指数、伴随分析、亲密度、涉毒指数等深度训练模型。

重大项目深度考验

I+ 已经作为亮点应用参与多个国家级重点项目，在安保、反恐、关税等领域让客户耳目一新，业务价值得到深度考验和认可。

32.3 相关概念

搜索

通过对象属性进行数据检索分析，同时作为关系网络和地图分析视图的入口。

关联反查

以任意单个或一批对象为起点进行关系的无限拓展分析。

群体分析

分析一批相同或不同类型的对象内部之间的关联关系。

共同邻居

分析一批相同或不同类型的对象共同联系对象。

路径分析

分析两个对象之间的关系路径。

骨干分析

针对团伙网络，通过智能业务算法，探索关系网络中核心骨干节点。

血缘分析

以家族户号为血缘脉络，展示所有人之间的血缘关联。

行为分析

展示事件在时间维度上发生的分布频率情况。

时序分析

在时间维度上详细展示每个事件发生细节。

行为明细

将事件的明细信息（原数据记录按规则筛选）展示出来。

对象信息

汇总关系网络中的实体，并按实体类型分类。

统计信息

统计关系网络中的关系和实体，包含总体分布、对象熟悉和关系属性。

总体分布

含三类统计：对象分布、关系分布和关联分布。

对象属性

每一类对象按照属性字段进行分类统计。

关系属性

每一类关系按照关系属性字段进行分类统计。

伴随分析

计算在时间和空间维度计算和预设对象伴随的一批对象。

OLP数据

以一种高度抽象的方式，将各类数据析解成对象（Object）、属性（Property）以及对象间的关联关系（Link），以 OLP 模型来组织关系数据。

32.4 登录系统

背景信息

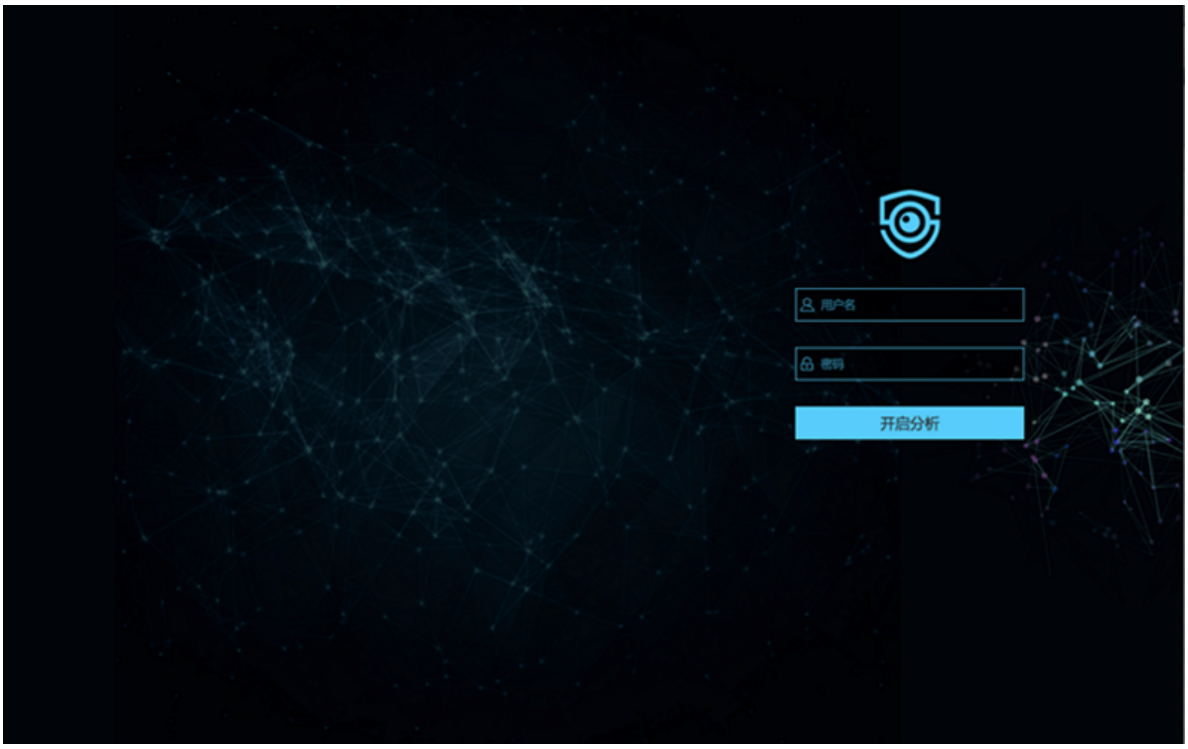
登录系统首先是产品安全的第一道防护线，所有用户都首先取得相应的账号，然后进入到登录页面。账号密码首先是由系统管理员（注：系统管理员是系统部署应用过程中指定的业务负责人员。）分配账号（分配账号首次登录使用默认密码，具体由系统管理员明确告知）。

首次登录提示用户修改密码。同时提供修改密码入口方便用户安全管理账号。

操作步骤

1. 在浏览器中输入控制台的访问 URL，默认地址为分析工作台地址+端口号（3005），例如：iplus.graphanalytics.net:3005/login/index.html。
2. 进入登录页面，如图 *1314:* 登录页面所示。

图 1314: 登录页面



3. 输入账号和密码，单击**开启分析**。

   在账号输入过程中会检测输入字符是否合法，如果输入非法字符，如%%，则系统提示**用户名包含不合法字符，请重新输入!**，合法输入情况下，系统会对账号和密码做校验。

4. 用户首次登录时，系统会提示修改密码，根据提示，修改密码。

   默认进入系统启动页，如图 *1315:* 系统启动页所示。

   图 1315: 系统启动页



在系统启动页中，将会分别展示三类常用分析入口：**最近打开的文件**、**分享给我的文件**以及**开始使用**。

这三个入口中分别对应最近保存的分析、别人分享给您的文件，以及您要新建分析的一个入口。单击每一个 list 下面的每一个 item，用户即可直接进入该分析。

32.5 搜索对象

搜索作为 I+ 三大独立模块之一，可以作为研判人员查找对象的工具，该对象包含手机，身份证等不同对象实体，方便研判人员快速查看对象信息。同时搜索出的对象也可以作为单独对象节点添加到关系分析和GIS分析中去，可以作为研判起点。

搜索功能的入口，如图 *1316:* 搜索功能入口所示。

图 1316: 搜索功能入口



注意: 当搜索条件为空时，不能搜索出结果，提示**请至少输入一项条件值。**

32.5.1 简单搜索

操作步骤

1. 选择搜索的关键字类型，可以选择的类型枚举在搜索框下面，搜索框可以收起或展开，如图 *1317: 搜索框*所示。

   图 1317: 搜索框



2. 在搜索框中输入关键字。支持模糊搜索，例如要搜索身份证中含有123的对象，则在搜索框中输入**123**即可。
3. 单击**搜索**按钮或通过点按键盘的**Enter**键即可进行搜索。

   搜索结果如图 *1318: 搜索结果*所示。

   图 1318: 搜索结果



32.5.2 高级搜索

高级搜索可以帮助用户指定更具体的业务范围，支持相关属性的模糊搜索和数据源指定搜索两种方式。

高级搜索区域在搜索框正下方，可以收起，也可以默认不填写。如图 *1319:* 高级搜索所示。

图 1319: 高级搜索



多关键字高级搜索

用户可以指定多个关键字进行精确查找。

例如，用户要查找身份证证件号码中含有“234”，且姓名为“liqing3440”的对象，操作步骤如下：

1. 选择关键字为**身份证**，并在搜索框中输入“234”。
2. 选择关键字为**姓名**，并在搜索框中输入“liqing3440”。
3. 单击**搜索**按钮或通过点按键盘的**Enter**键即可进行搜索。

   搜索结果如图 *1320: 多关键字高级搜索*所示。

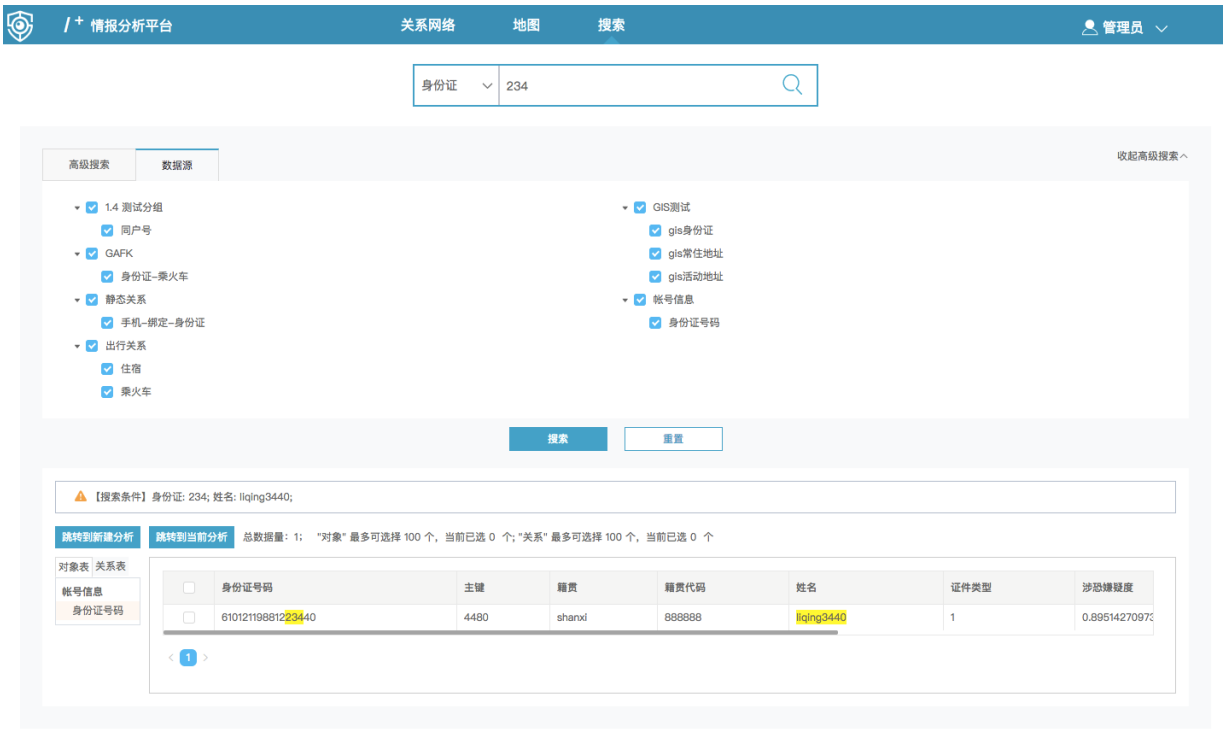
图 1320: 多关键字高级搜索



数据源高级搜索

数据源是后台业务定义的逻辑实体和逻辑关系，如图 *1321: 数据源高级搜索*所示。

图 1321: 数据源高级搜索



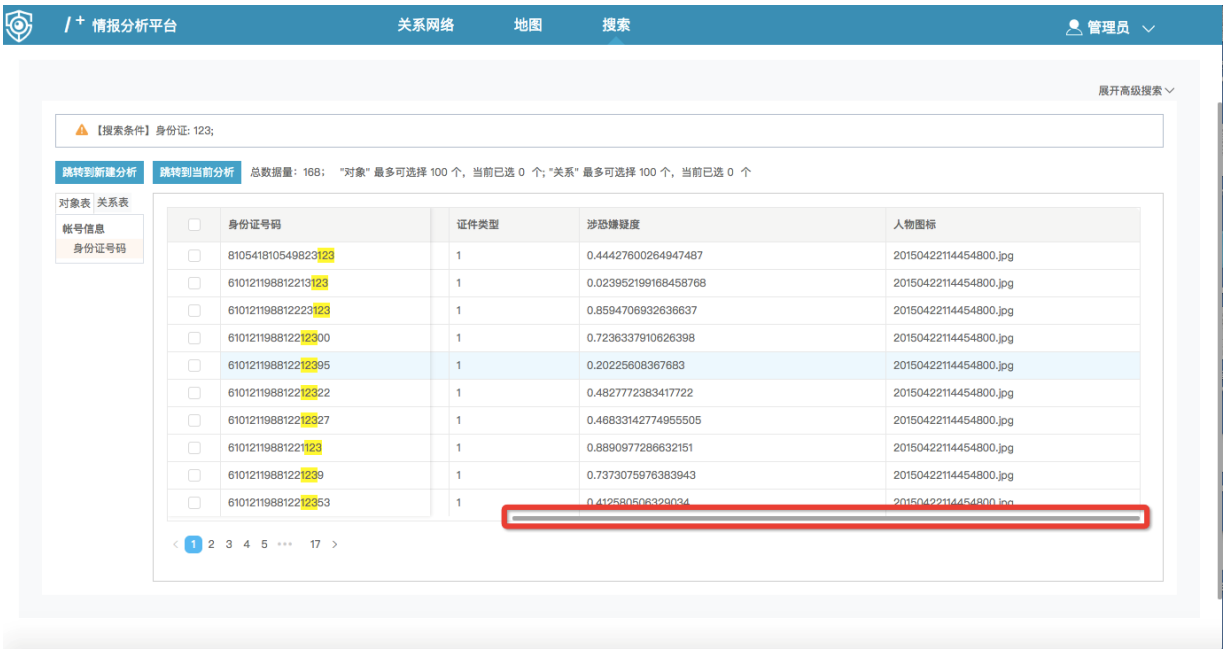
高级搜索提供了条件重置功能，您可以单击**重置**按钮清空搜索条件并重新输入。

32.5.3 搜索结果

搜索结果显示

搜索结果中，命中关键字的部分会黄色高亮显示。同时搜索结果会附带显示关键字以外的信息，用户可以通过滚动条来左右移动查看，但关键字那一列固定不动，如图 *1322:* 搜索结果展示所示。

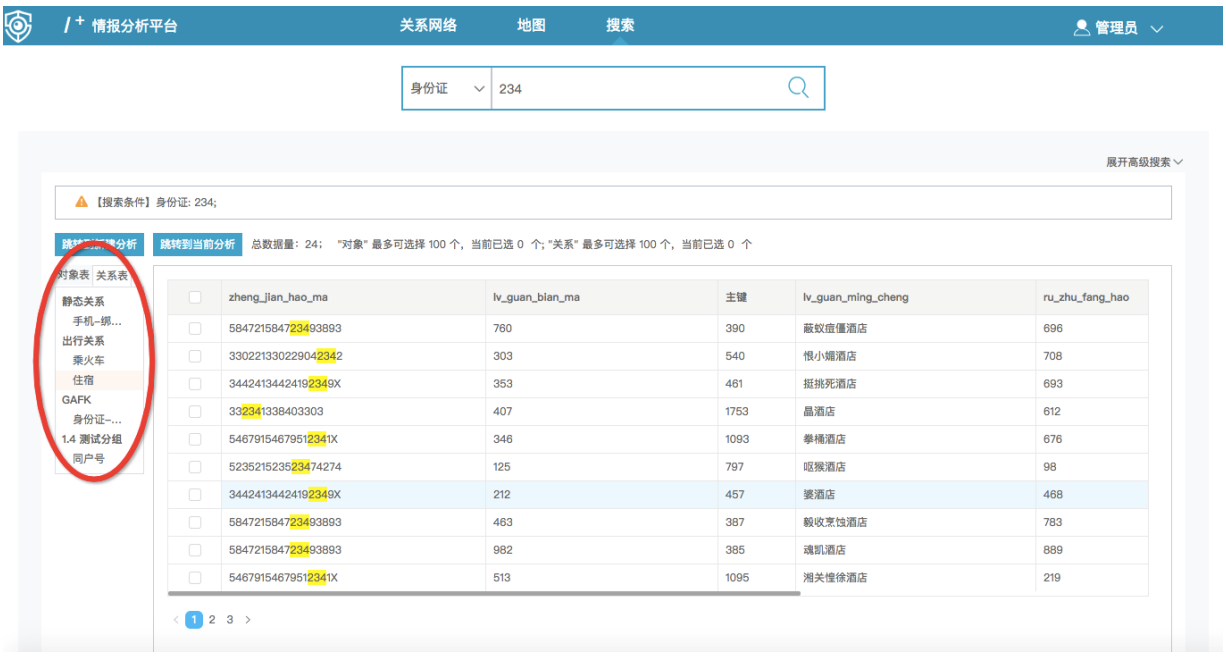
图 1322: 搜索结果展示



说明: 当搜索出的结果较多时，I + 提供分页展示，每一页展示10条记录。

搜索结果提供对象表和关系表两种展示方式，您可以单击页面左侧的**对象表**或**关系表**页签进行切换。如图 *1323: 相关信息*所示，搜索身份证号，不仅能搜索出身份证信息表，还能搜索出与身份证相关的事件信息，如乘火车、住宿、同户号等。

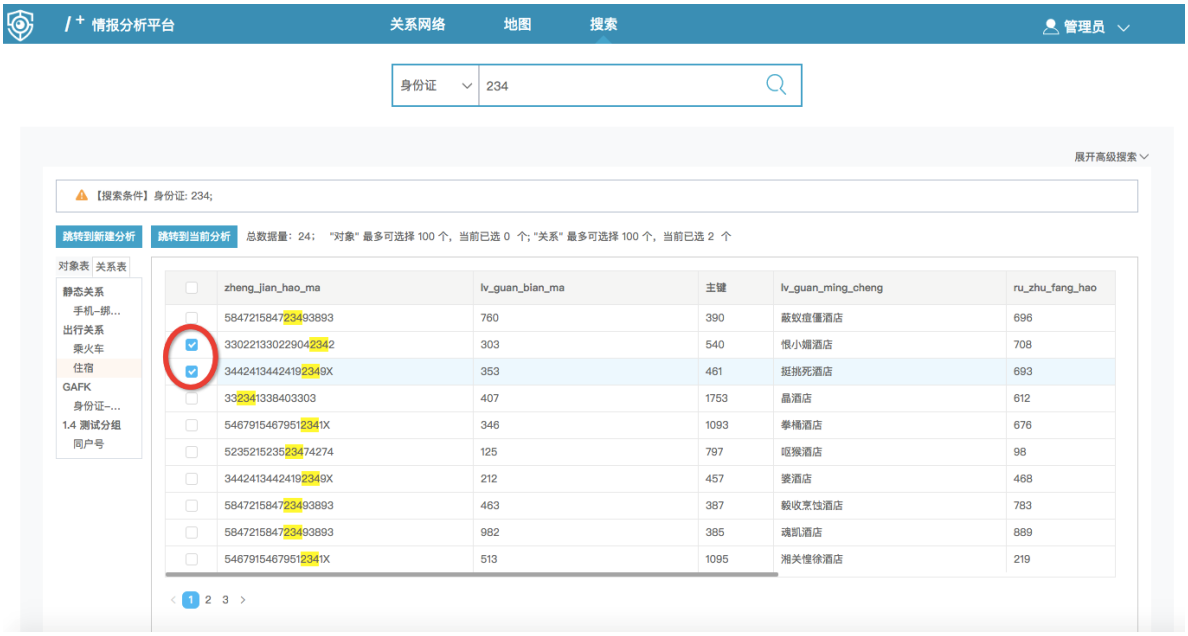
图 1323: 相关信息



搜索结果分析

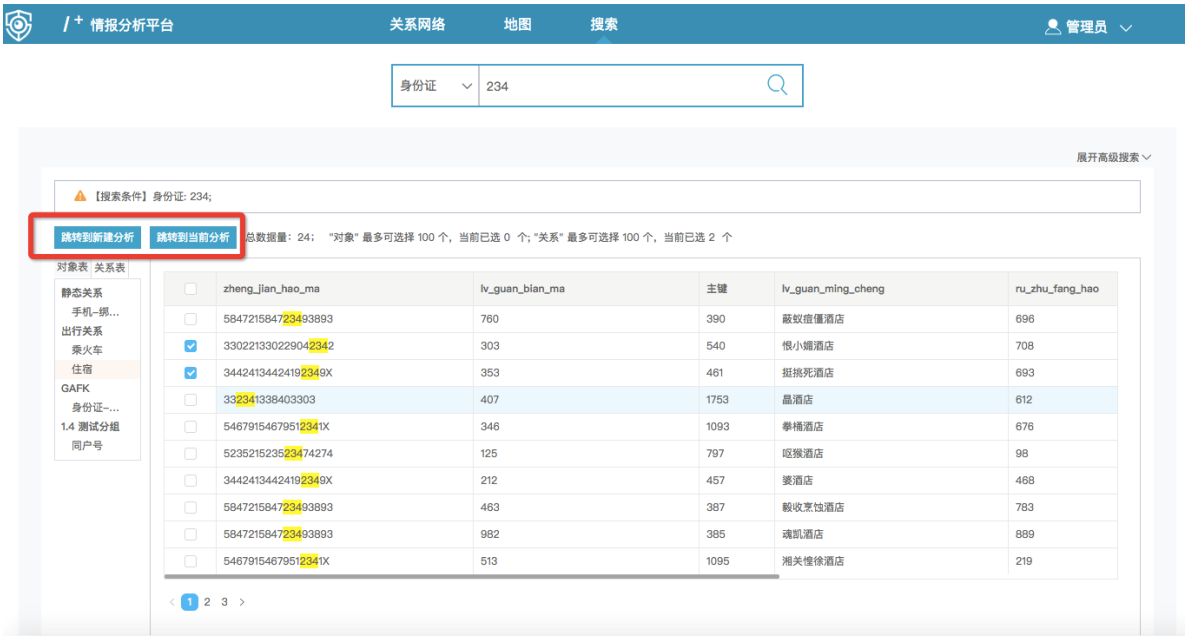
1. 在搜索结果中，选中部分或者全选（全选指当前页的10条记录），如图 *1324:* 选择搜索结果所示。

图 1324: 选择搜索结果



2. 单击**跳转到新建分析**按钮或**跳转到当前分析**按钮，如图 *1325:* 搜索结果分析所示。

图 1325: 搜索结果分析



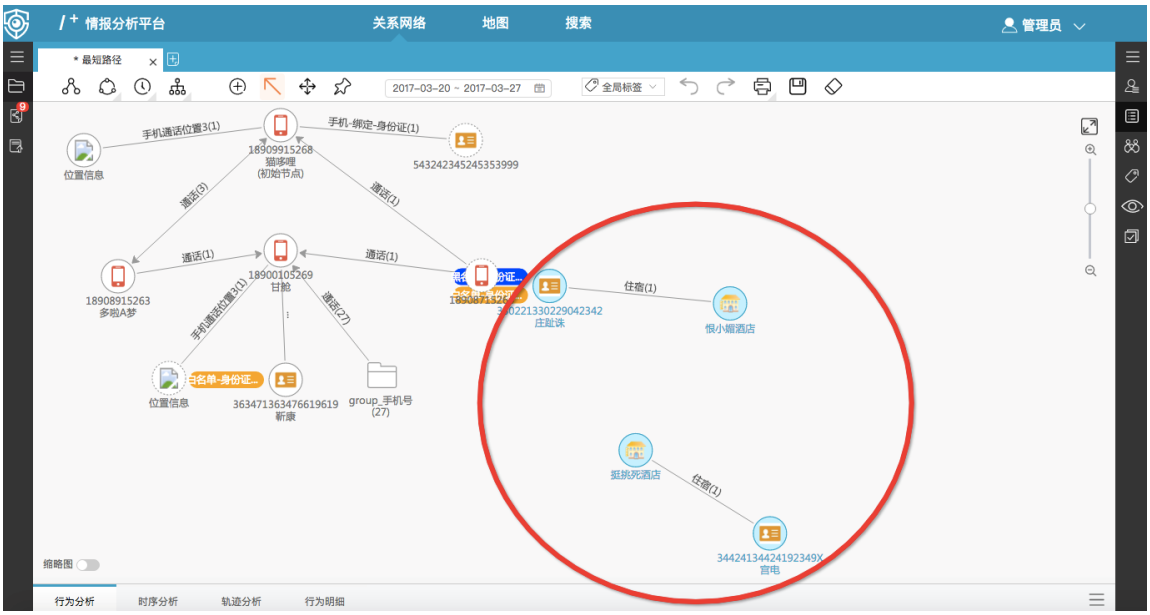
- 跳转到新建分析，如图 *1326: 新建分析*所示。

图 1326: 新建分析



- 跳转到当前分析，如图 *1327: 当前分析*所示。

图 1327: 当前分析



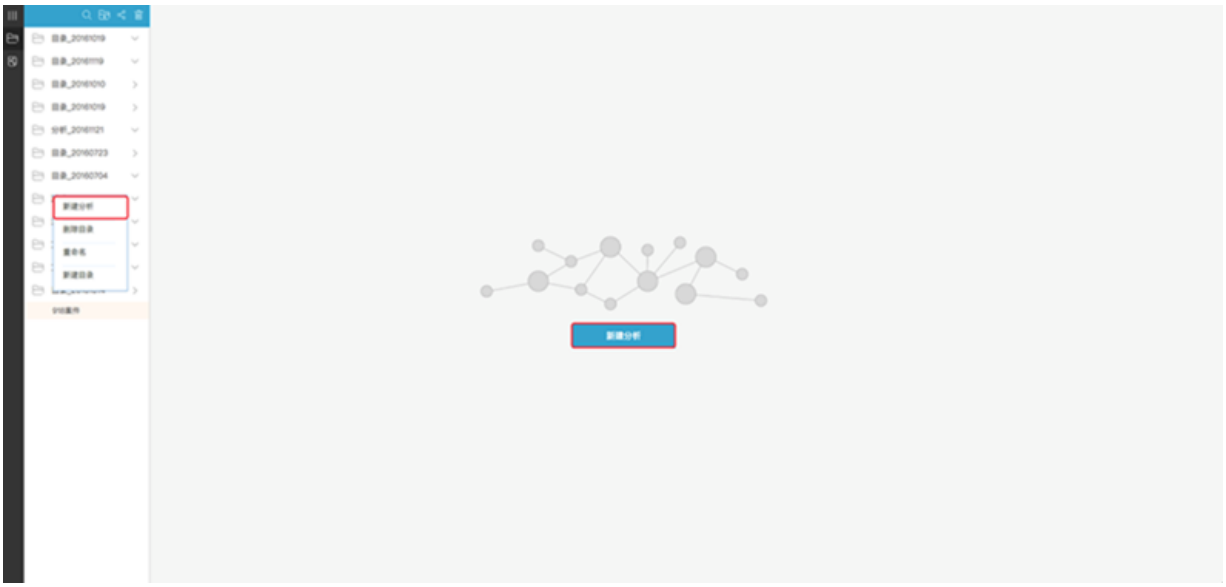
32.6 关系网络

32.6.1 新建分析

新建分析

在分析主界面上，单击**新建分析**或直接在左侧目录树上新建分析，如图 *1328: 新建分析*所示。

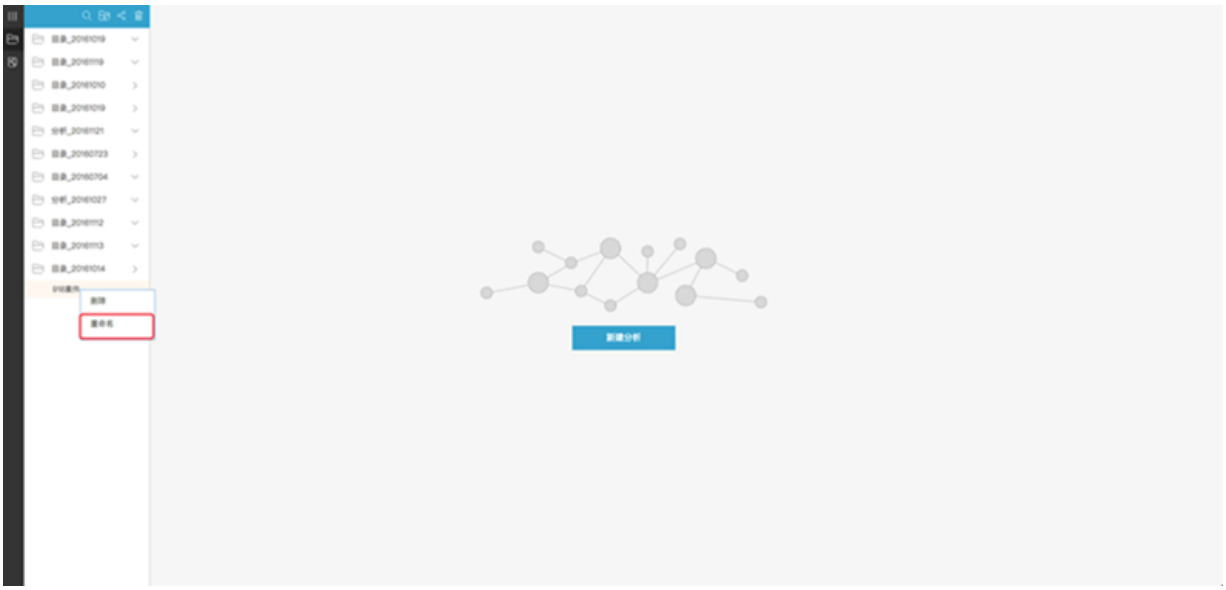
图 1328: 新建分析



修改分析名称

单击已经存在的分析，选择**重命名**，可以修改分析的名称，如图 *1329: 修改分析名称*所示。

图 1329: 修改分析名称

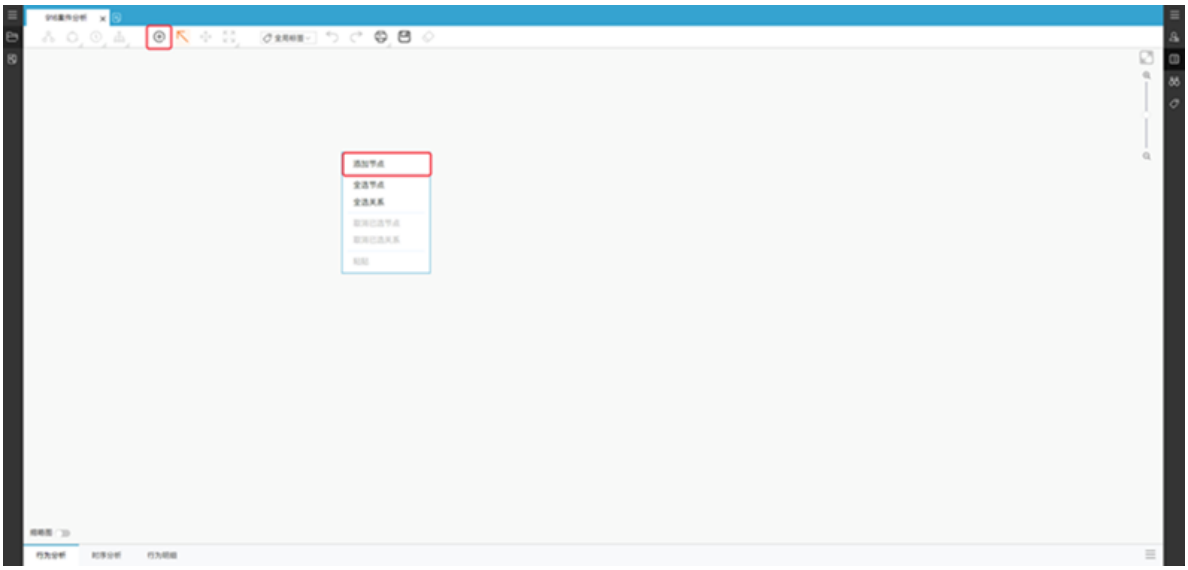


32.6.2 新增节点

操作步骤

1. 单击分析主界面的图标，或右键单击图区空白区域，选择**添加节点**，如图 *1330:* 添加节点所示。

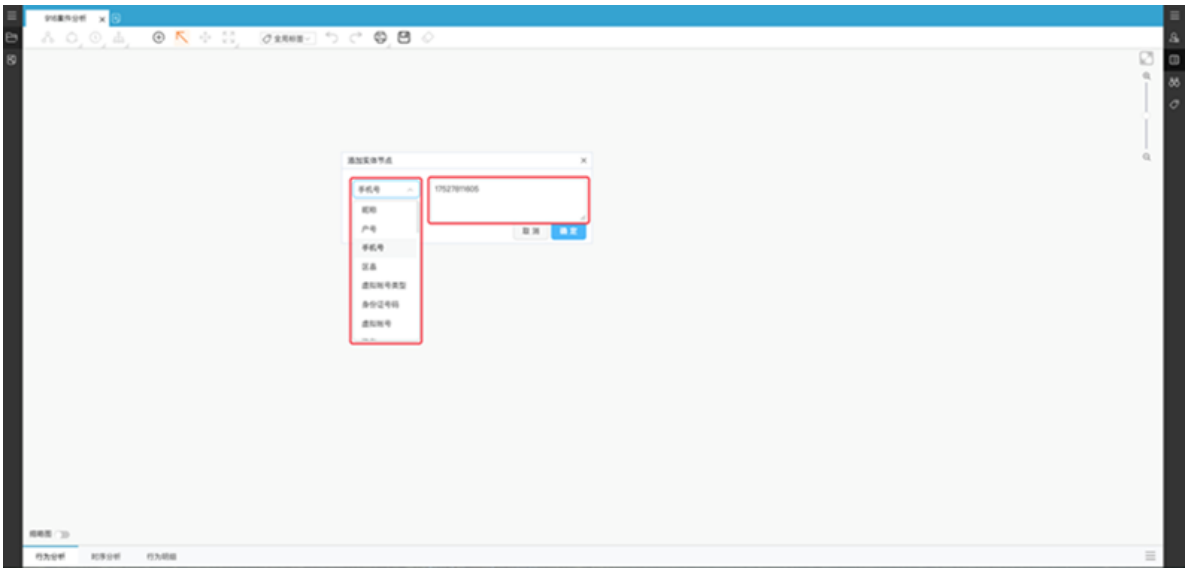
   图 1330: 添加节点

2. 选择节点类型，可输入单个或多个节点信息。

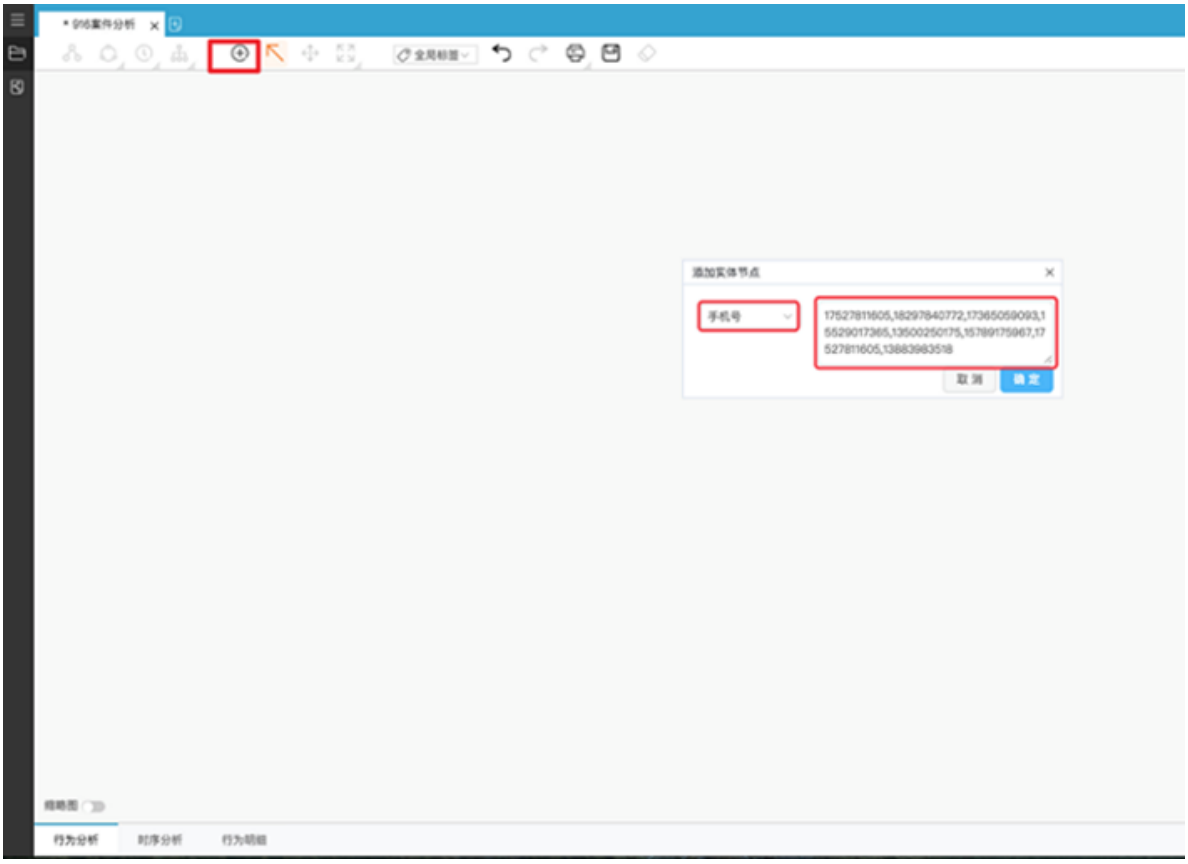
   输入单个节点如图 *1331:* 添加单个节点信息所示。

   图 1331: 添加单个节点信息



输入多个节点信息，各个节点信息之间以英文逗号相隔，如图 *1332: 添加多个节点信息*所示。

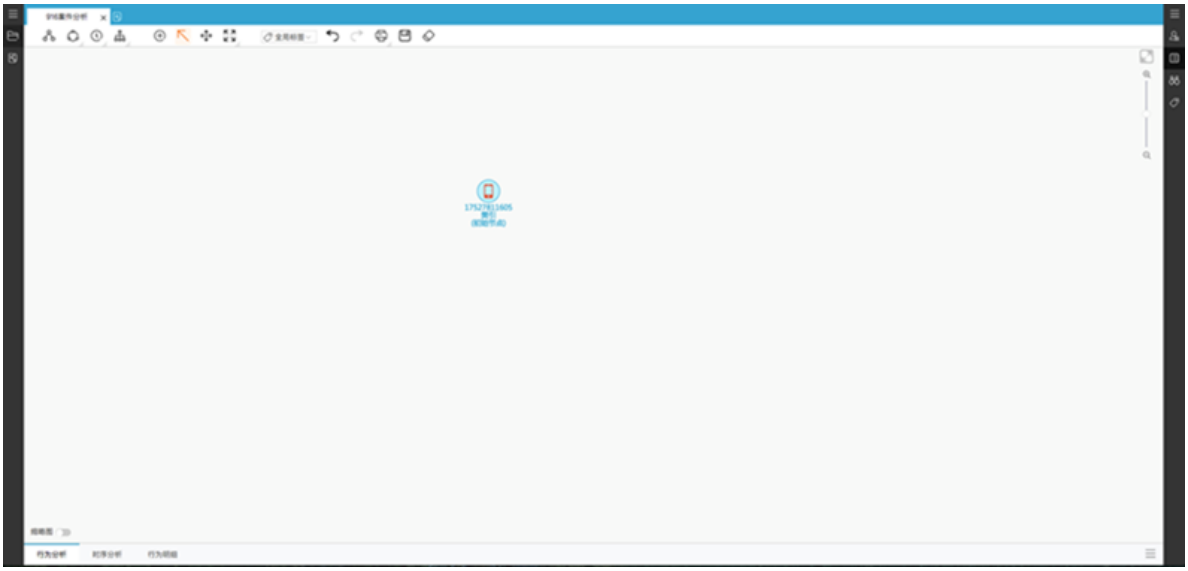
图 1332: 添加多个节点信息



3. 单击**确定**按钮，新添加的节点会显示在图区中。

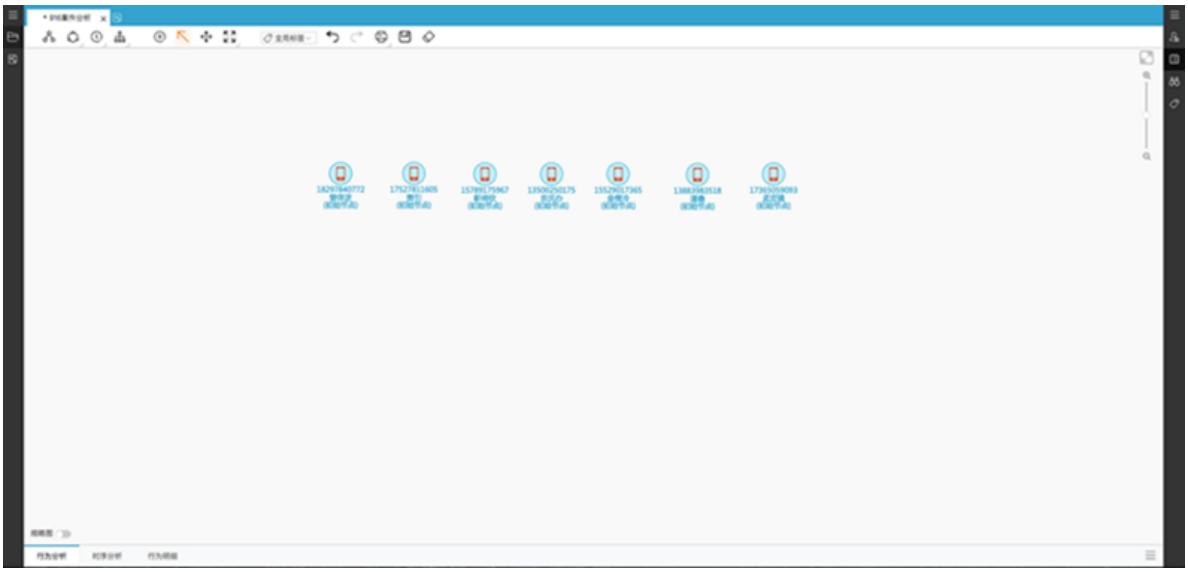
添加单个节点成功，如图 *1333:* 添加单个节点成功所示。

图 1333: 添加单个节点成功



添加多个节点成功，如图 *1334:* 添加多个节点成功所示。

图 1334: 添加多个节点成功



32.6.3 关联反查

背景信息

关联反查可帮助用户实现信息的无限关联。情报分析工作的核心是从大量的、没有关联的信息中发现少量的关联性线索和情报，即将信息转换为可操作情报的过程。

关联反查提供如下两种方式：

- 简单方式：用户直接双击需要关联反查的节点，就可以拓展其关联信息。
- 高级方式：用户通过业务经验筛选特定的条件来拓展关联信息。

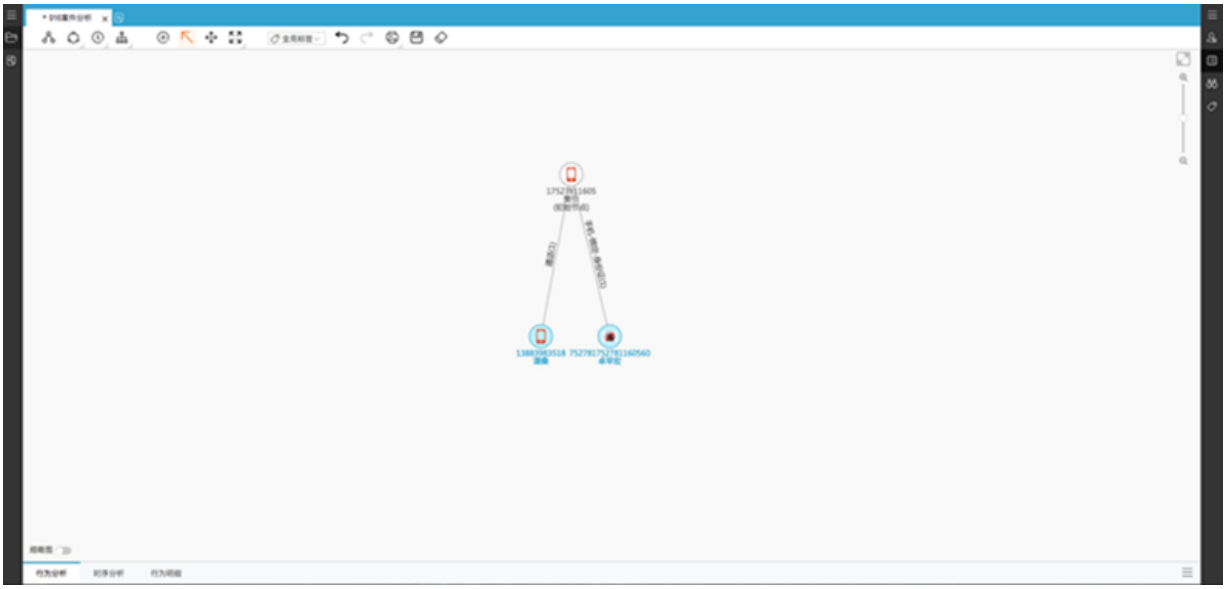
前提条件

- 已存在分析文件，新建分析操作请参见新建分析。
- 已存在节点对象，新增节点操作请参见新增节点。

通过简单方式进行关联反查

直接双击节点，默认分析，结果如图 *1335:* 简单方式所示。

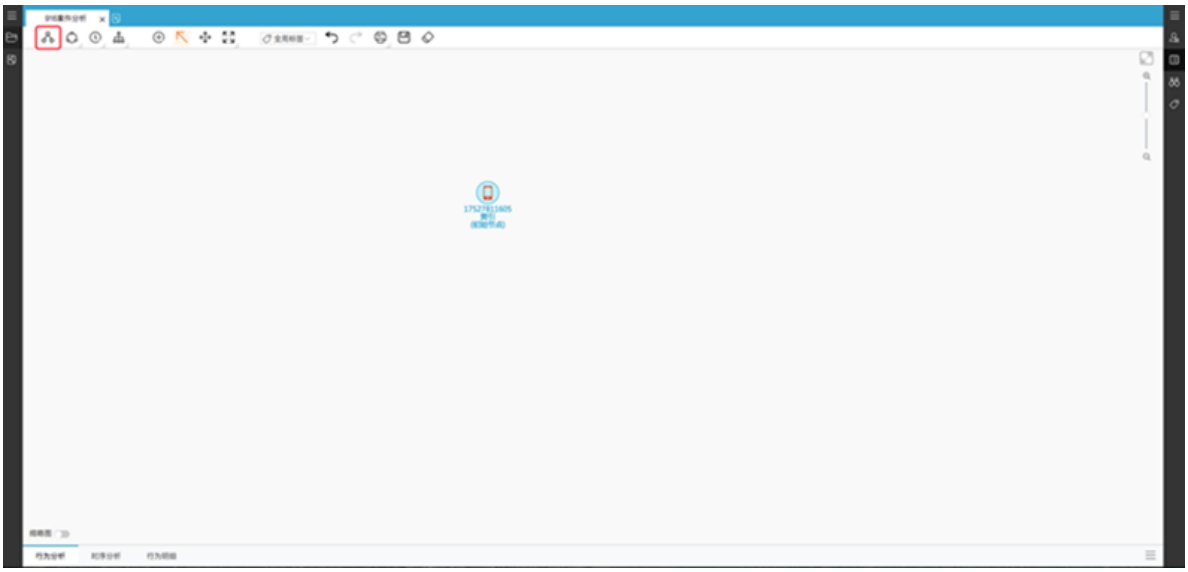
图 1335: 简单方式



通过高级方式进行关联反查

1. 选中节点，在工具栏上选择关联反查的图标，如图 *1336:* 高级方式所示。

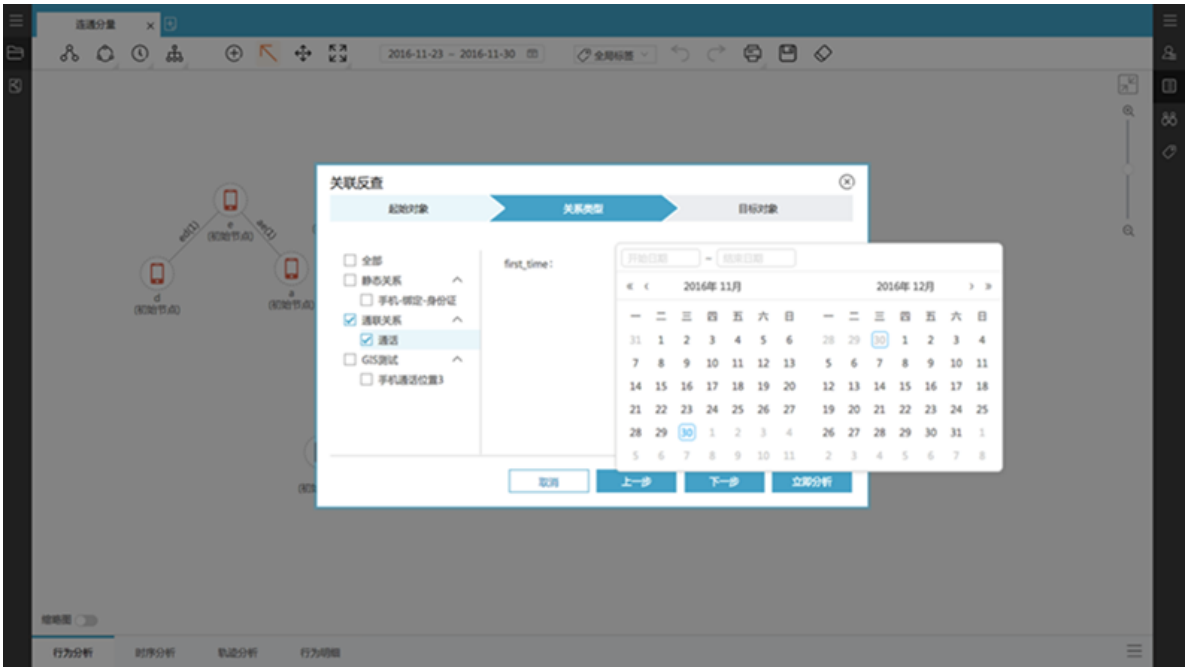
图 1336: 高级方式



2. 在出现的对话框中，进行对象选择、关系选择及关系条件的设置。

   如图 *1337:* 设置关系类型所示，分析用户可以设定时间范围，系统支持时间类型、枚举类型、等值类型的设定。

   图 1337: 设置关系类型



32.6.4 群体分析

前提条件

- 已存在分析文件，新建分析操作请参见新建分析。

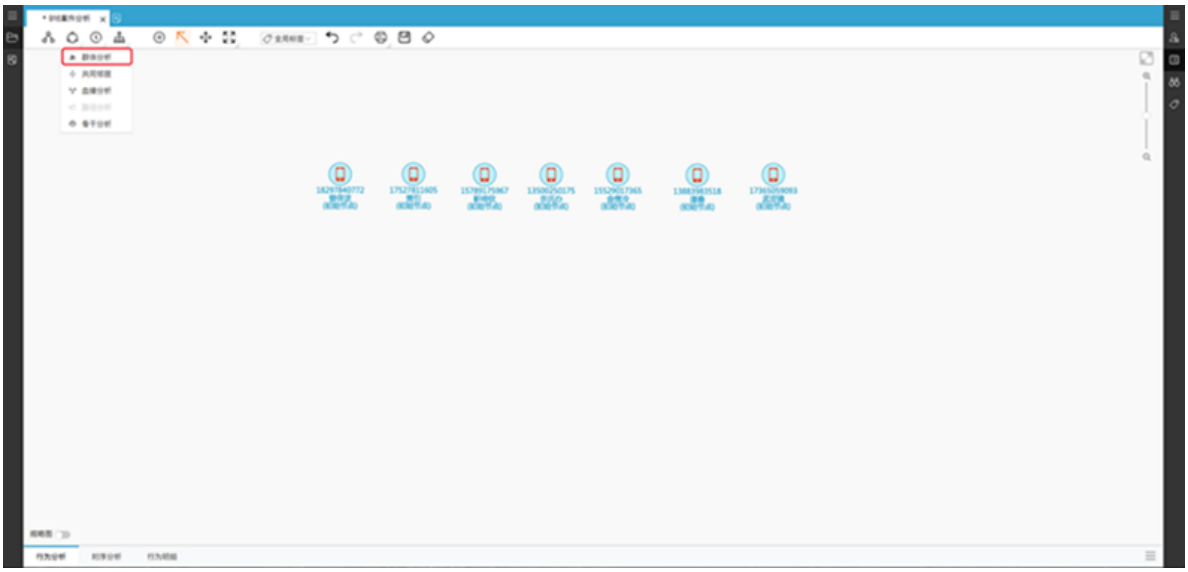
- 已存在节点对象，新增节点操作请参见新增节点。

操作步骤

1. 使用鼠标拖拉框选中需要做群体分析的实体图标，单击如图 *1338: 群体分析*所示的按钮，选择**群体分析**，弹出**群体分析**设置框。

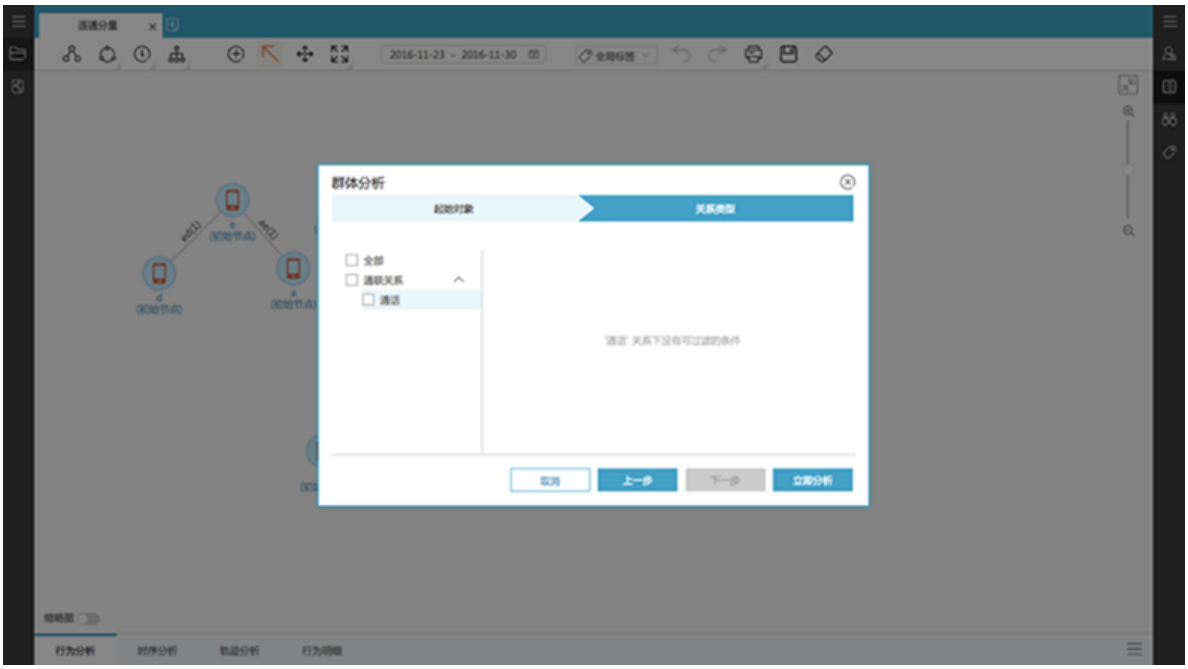
   说明: 复选多个节点可以通过**control**键来控制。

   图 1338: 群体分析

2. 依次进行实体选择、关系选择，同时用户可以针对关系类型选择控制，也可以结合到特定业务需求，对每种关系类型上的属性内容细化，系统支持时间类型、枚举类型、等值类型的条件设定，如图 *1339: 设定群体分析条件*所示。

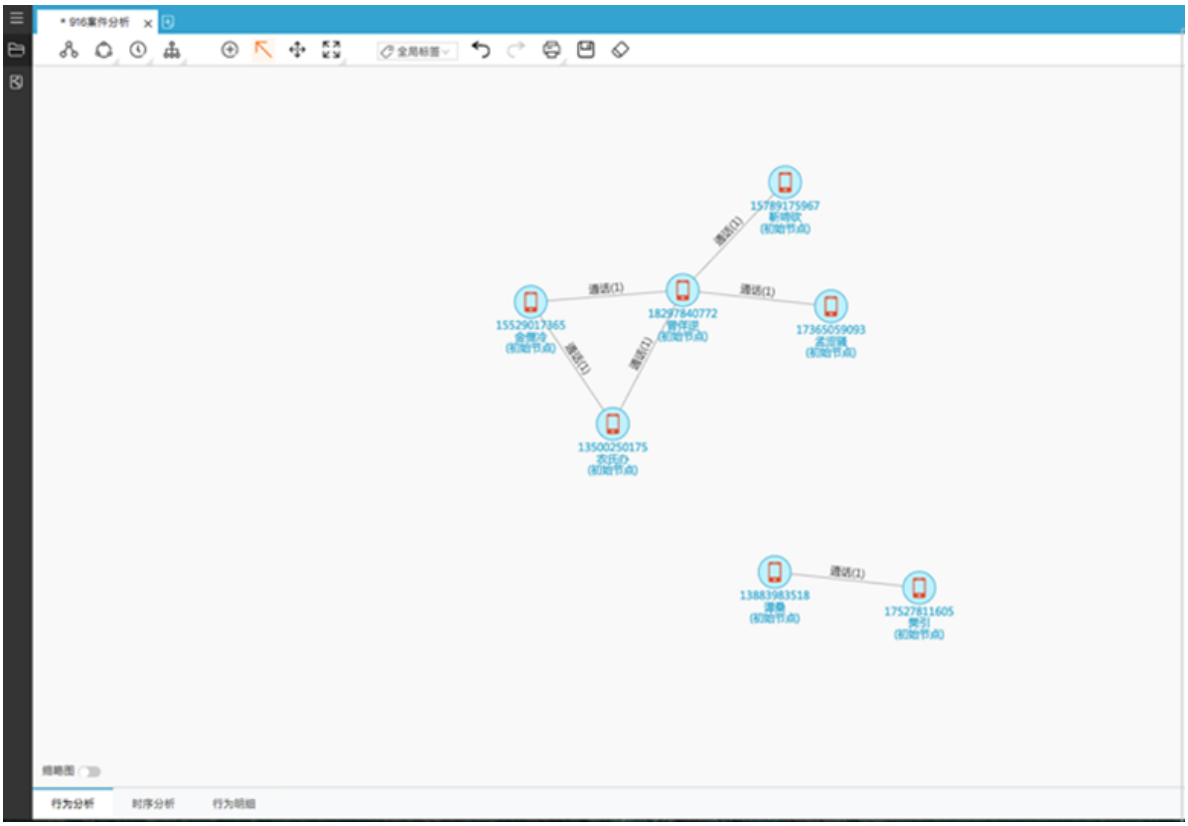
   图 1339: 设定群体分析条件



3. 用户完成条件设定后，单击**立即分析**。

   进入分析，如图 *1340: 开始分析*所示。

图 1340: 开始分析



32.6.5 共同邻居分析

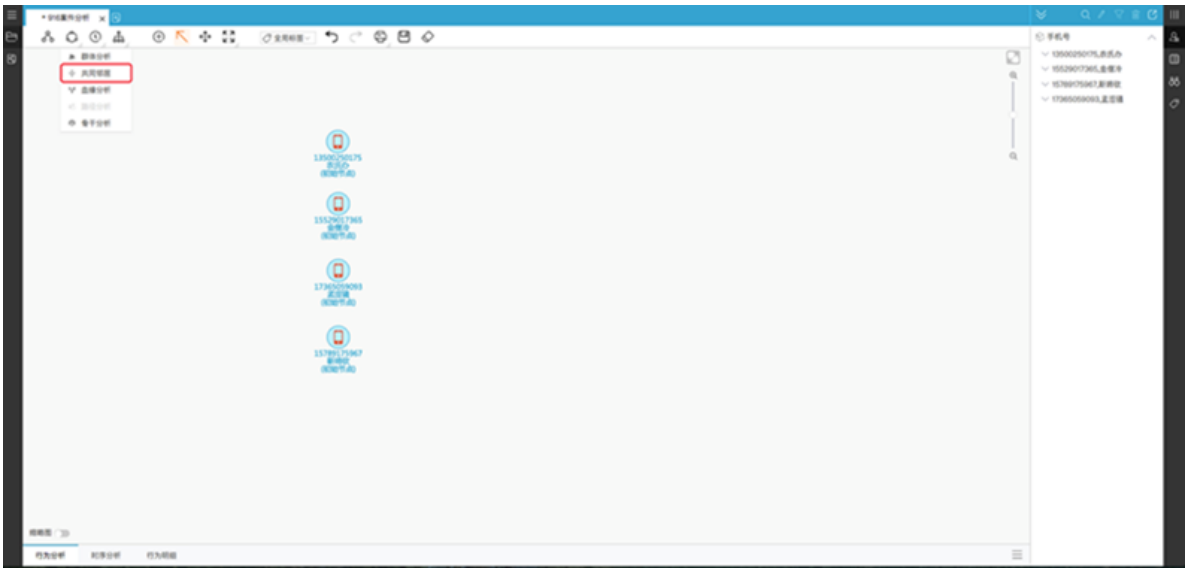
前提条件

- 已存在分析文件，新建分析操作请参见新建分析。
- 已存在节点对象，新增节点操作请参见新增节点。

操作步骤

1. 选中两个或多个节点，并在工具栏中选择**共同邻居**，如图 *1341:* 共同邻居所示。

   图 1341: 共同邻居

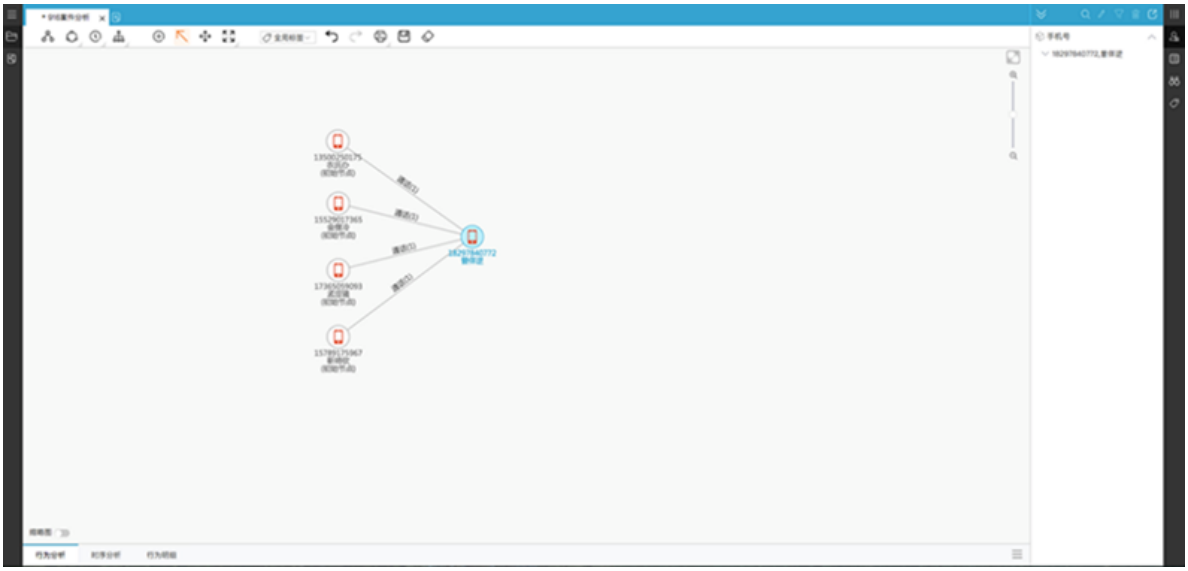
2. 共同邻居关系选择及条件设置，如图 *1342:* 共同邻居分析条件设定所示。

   图 1342: 共同邻居分析条件设定



3. 完成后，单击**立即分析**，结果如图 *1343:* 开始分析所示。

图 1343: 开始分析



32.6.6 路径分析

前提条件

- 已存在分析文件，新建分析操作请参见新建分析。
- 已存在节点对象，新增节点操作请参见新增节点。

背景信息

用户可以选中两个目标对象分析其关联路径。

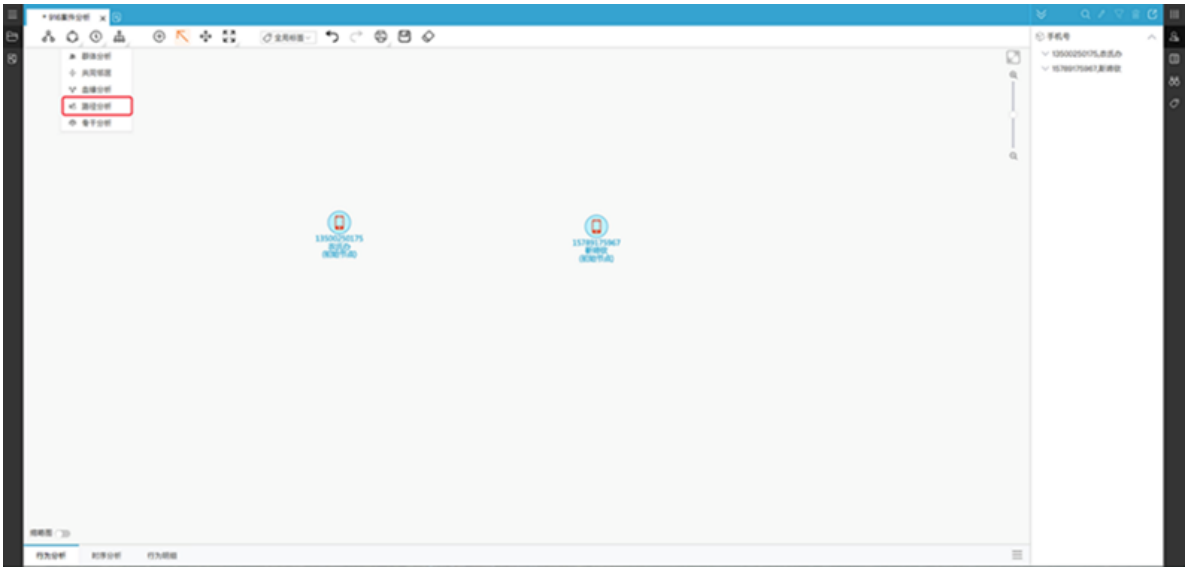
操作步骤

1. 选中两个目标对象，单击**分析** > **路径分析**，如图 *1344:* 路径分析入口所示。

   说明:

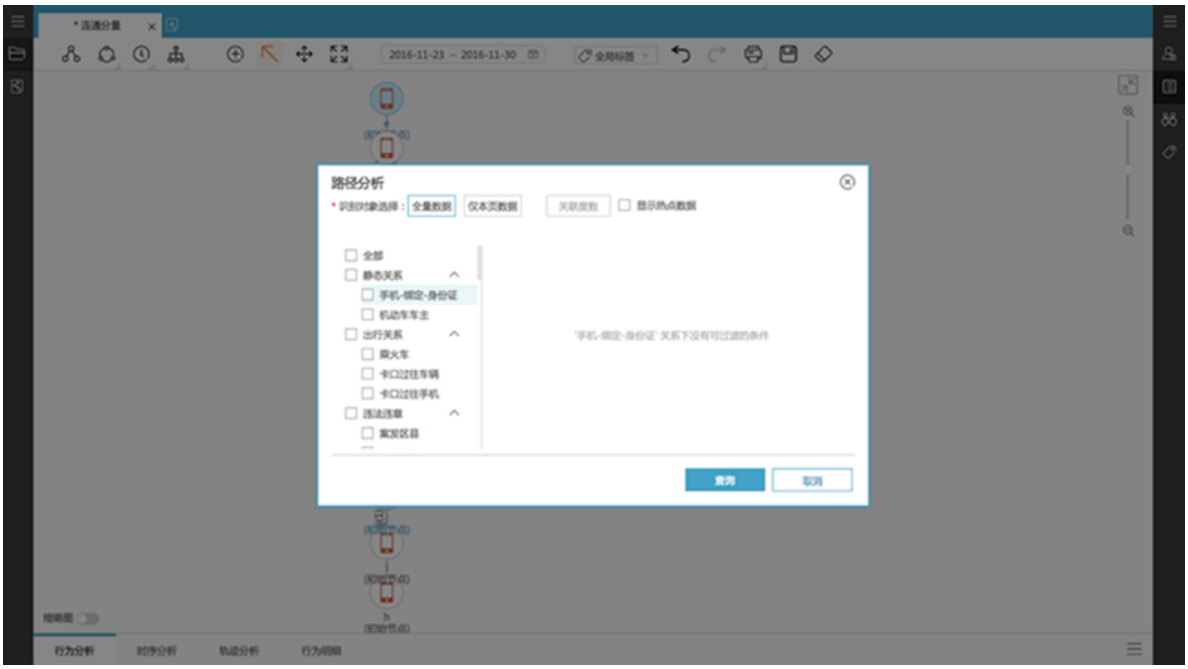
   - 选中目标对象，可以通过拉框选择和键盘快捷键**control**辅助完成。
   - 路径分析功能只支持两个对象节点的路径分析。

   图 1344: 路径分析入口



2. 选择**全量数据**或**仅本页数据**进行分析，如图 *1345:* 路径分析所示。

   图 1345: 路径分析



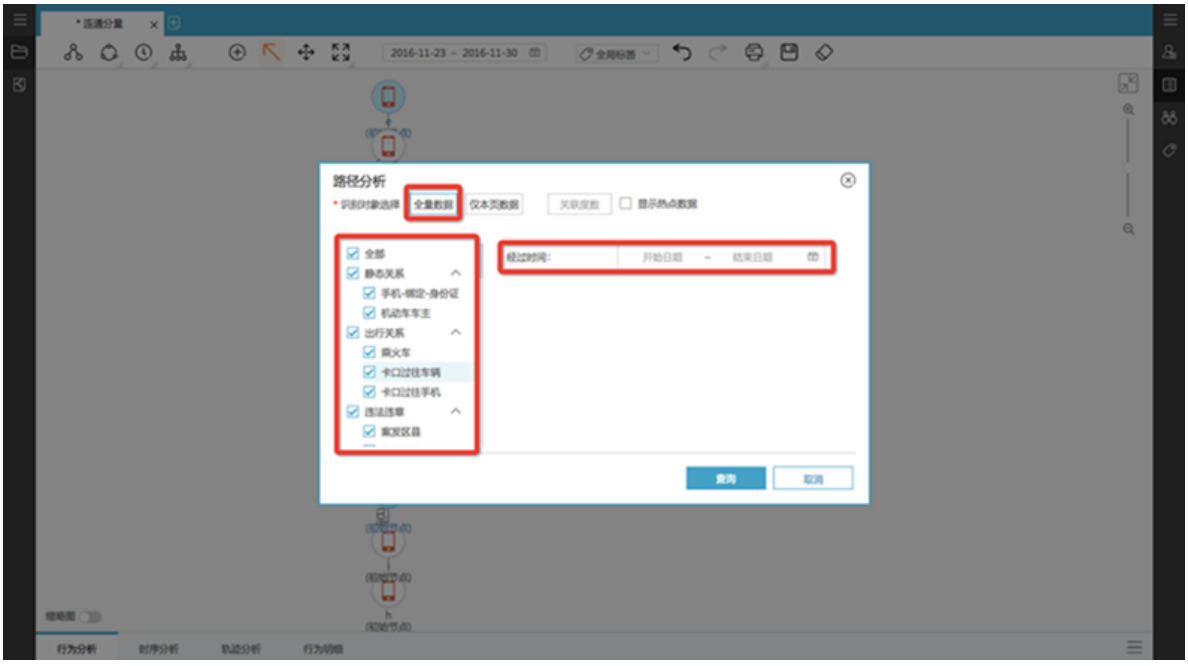
如果勾选**显示热点数据**，会将热点数据显示出来，但不会扩展。

说明: 热点数据由系统管理员控制，用于在关系网络呈现上给热点对象节点增加标红标签，警示用户。

全量数据分析

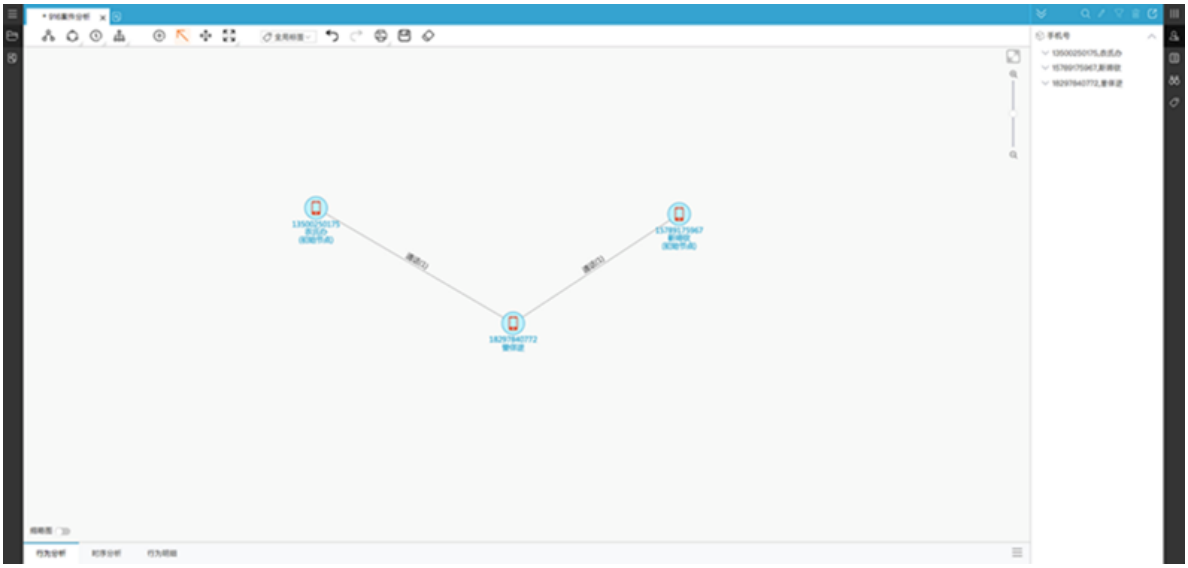
3. 设定全量数据的分析条件，进行路径分析，如图 *1346:* 设定分析条件所示。

图 1346: 设定分析条件



分析结果如图 *1347: 分析结果*所示。

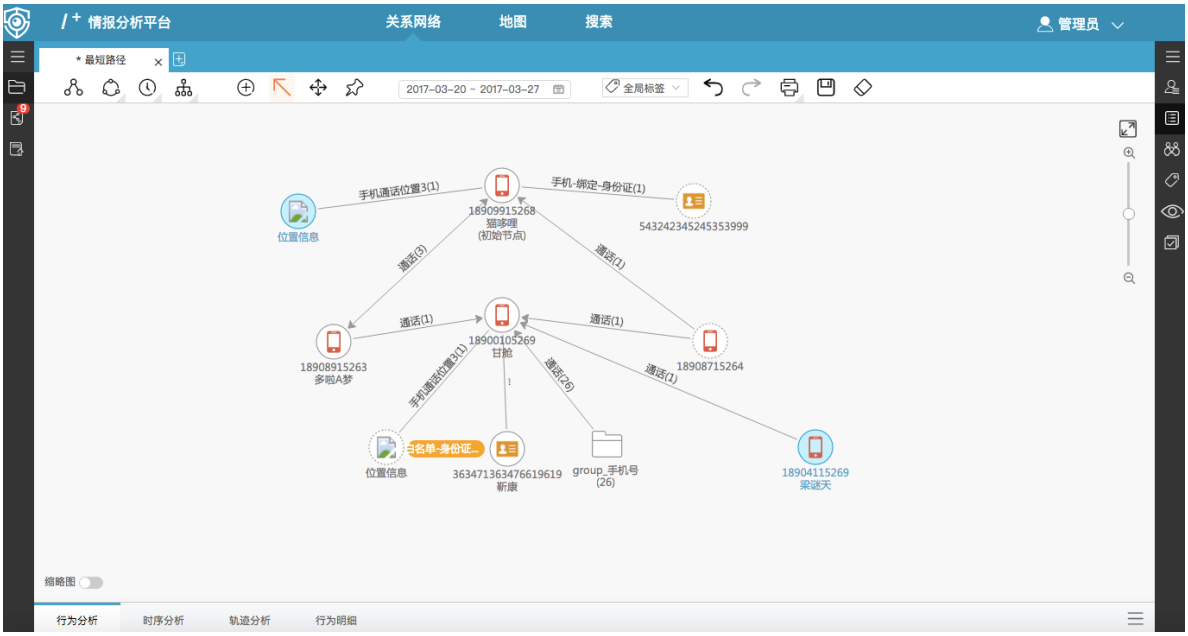
图 1347: 分析结果



仅本页数据分析

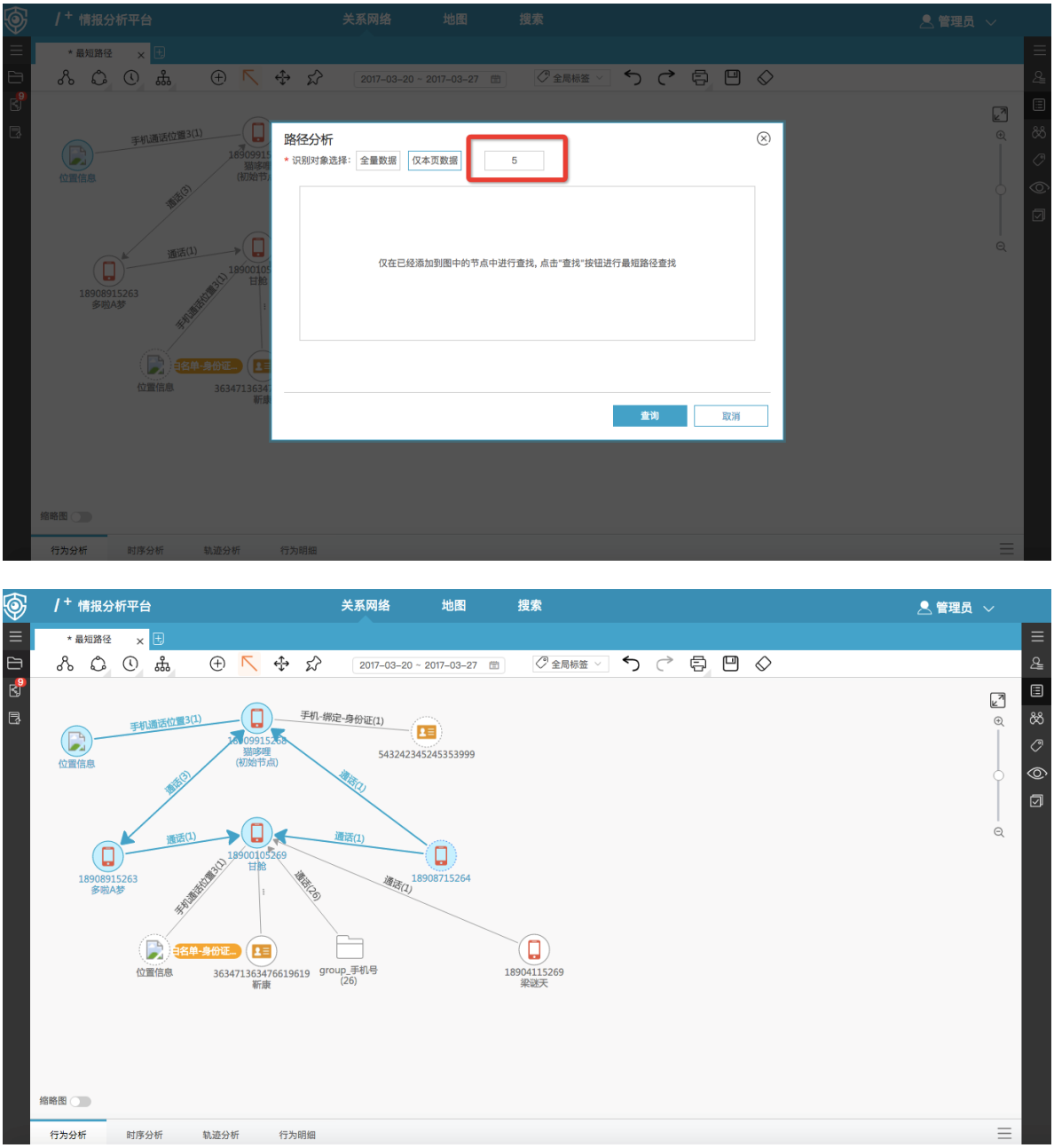
4. 仅本页数据的路径分析指当前页面中的两个点的路径，支持最短路径和指定度数的路径分析，如图 *1348: 分析结果2*所示。

图 1348: 分析结果2



如果是指定度数的路径分析，会找出所有符合条件度数（如果输入 n 度，会找出所有度数 <=n 的路径，n 由后台指定，最大不超过6）的路径，如图 *1349: 分析结果3*所示。

图 1349: 分析结果3



32.6.7 骨干分析

前提条件

- 已存在分析文件，新建分析操作请参见新建分析。
- 已存在节点对象，新增节点操作请参见新增节点。

操作步骤

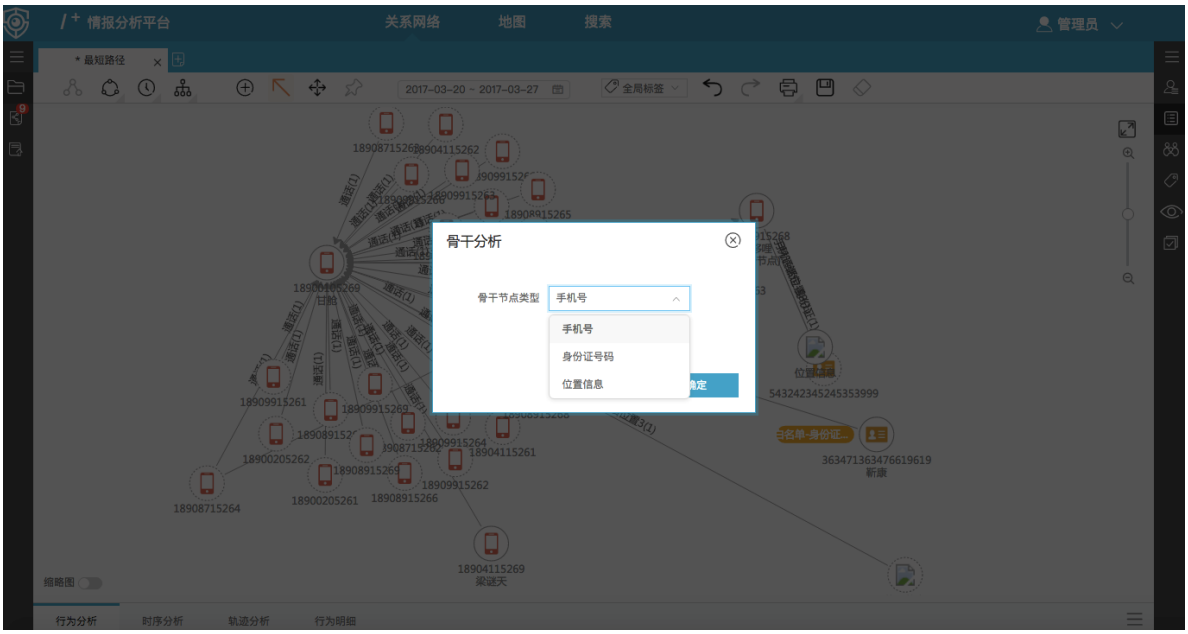
1. 点击**查询分析**图标，在弹出的下拉菜单中选择**骨干分析**项，如图 *1350: 骨干分析*所示。

图 1350: 骨干分析



2. 选择指定的骨干节点类型，如图 *1351: 骨干节点类型*所示。

图 1351: 骨干节点类型



3. 选择完成后，单击**确定**，图区的骨干节点会高亮显示，如图 *1352: 分析结果*所示。

图 1352: 分析结果



32.6.8 血缘分析

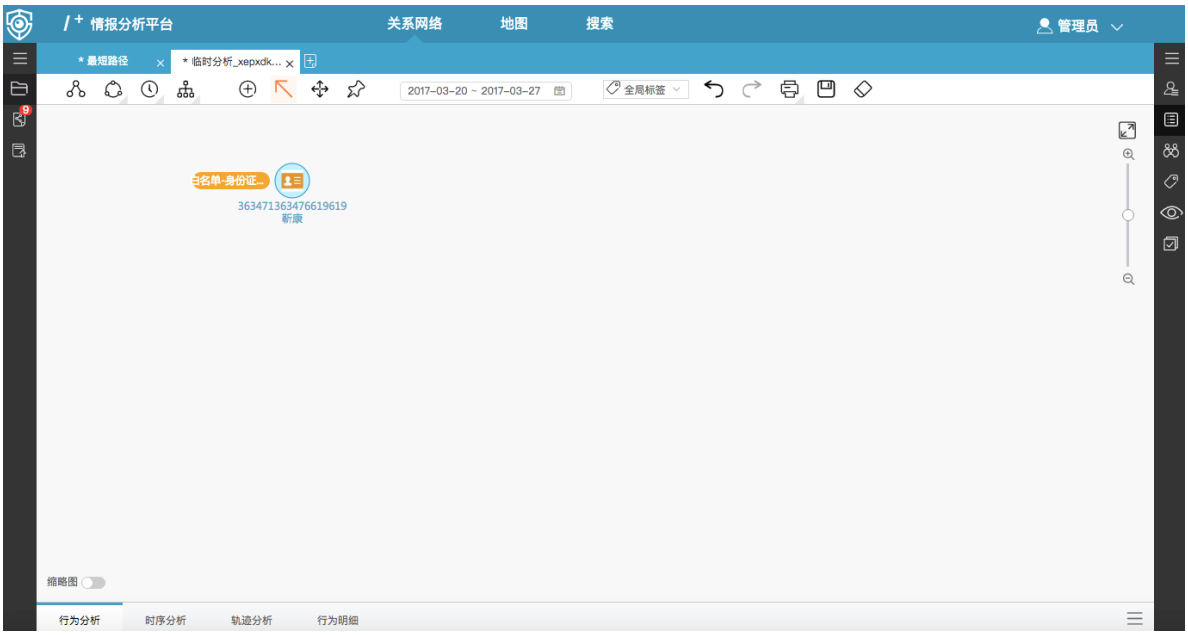
前提条件

- 已存在分析文件，新建分析操作请参见新建分析。
- 已存在节点对象，新增节点操作请参见新增节点。

操作步骤

1. 选中一个或多个节点，单击**血缘分析**，在图区显示该节点的血缘关系，如图 *1353: 血缘分析*所示。

图 1353: 血缘分析



2. 可以交互选择血缘关系，以及相应的条件。如图 *1354:* 选择血缘关系所示。

图 1354: 选择血缘关系





3. 分析结果如图 *1355: 分析结果*所示。

图 1355: 分析结果



32.6.9 时序分析

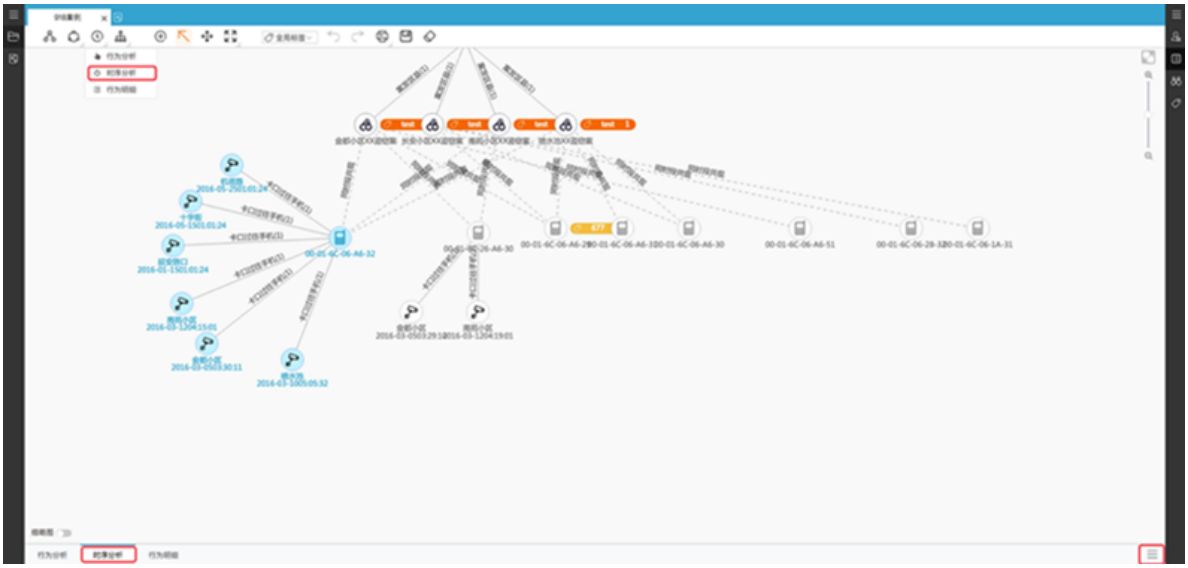
前提条件

- 已存在分析文件，新建分析操作请参见新建分析。
- 已存在节点对象，新增节点操作请参见新增节点。

操作步骤

在图区选中一批节点，单击**时序分析**，如图 *1356:* 时序分析所示。

图 1356: 时序分析



下方弹出时序分析视图，如图 *1357:* 分析结果所示。

图 1357: 分析结果



默认时序图中只允许五个对象进行展示分析。

图 *1357:* 分析结果中区域1，为时序缩略图，选中两侧任意一边的按钮可放大缩小缩略图，选中中间可移动缩略图。

图 *1357:* 分析结果中区域2，进行等比例放大缩小缩略图。

图 *1357: 分析结果*中区域3，当鼠标移动到对应时序线上时，悬浮弹出时序明细。

图 *1357: 分析结果*中区域4，可进行关系的选择、对象的选择，最多只允许同时对五个对象进行分析。

32.6.10 信息立方

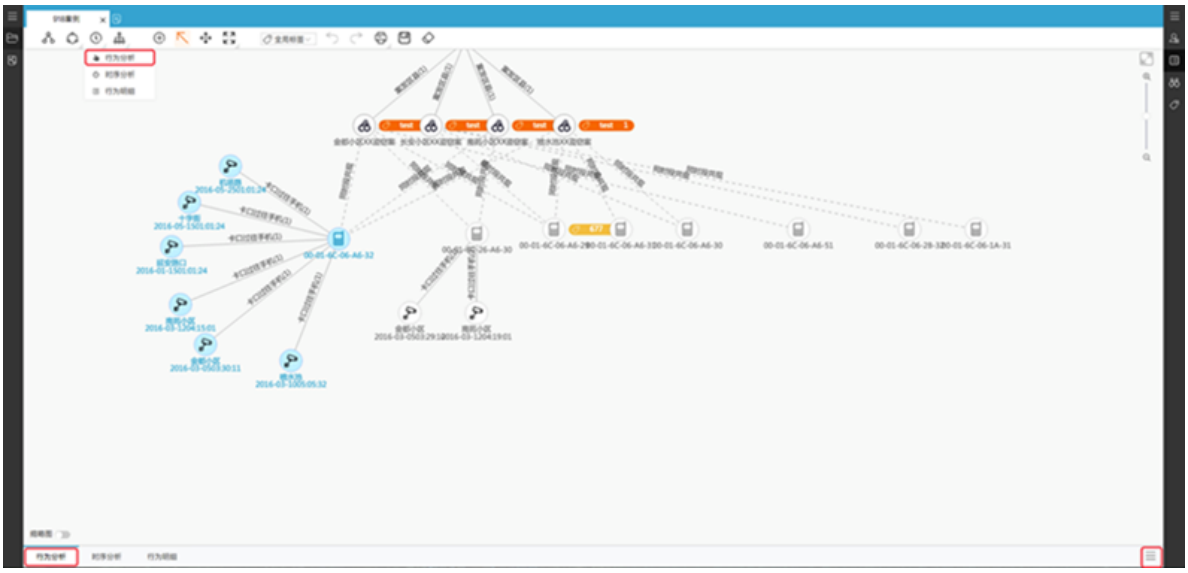
32.6.10.1 使用行为分析

行为分析可以对分析页面上带时间属性的关系数据进行可视化展示及统计分析。

操作步骤

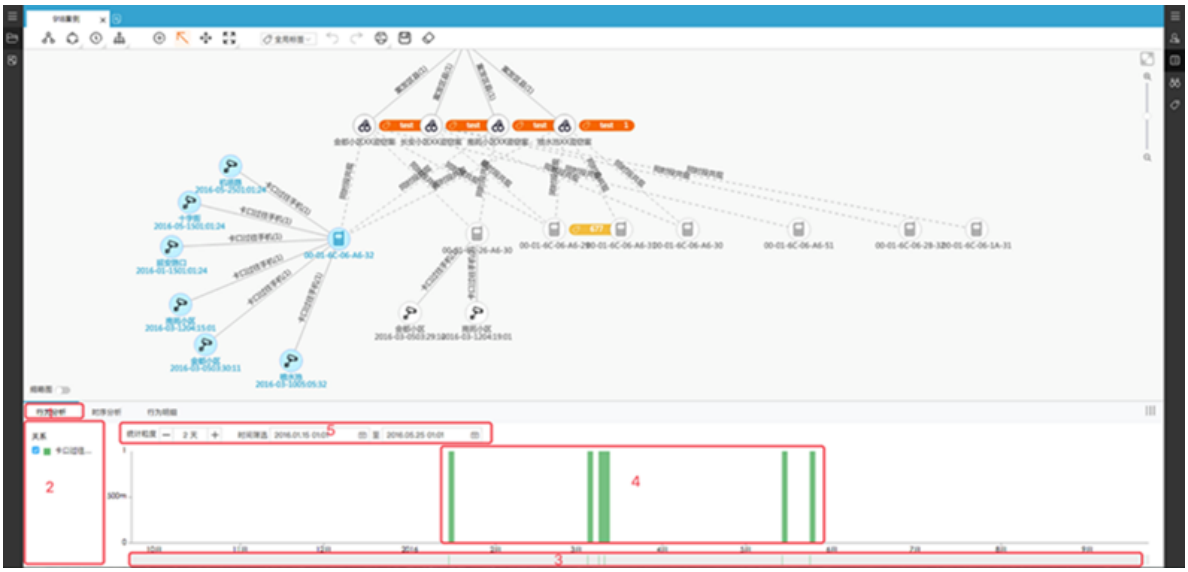
1. 选择页面分析内容，点击**时间分析**图标，在弹出的下拉菜单中选择**行为分析**项，如图 *1358: 行为分析*所示。

图 1358: 行为分析



2. 在显示的如图 *1359: 分析结果*所示的分析页面进行如下操作：

图 1359: 分析结果



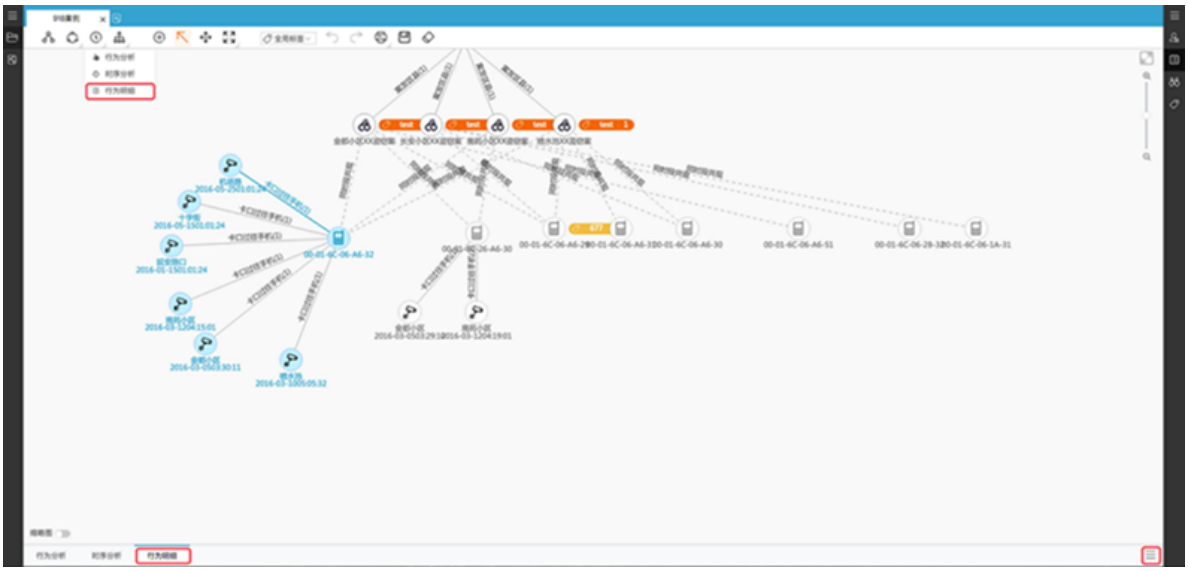
- 区域1：tab页选择，选择**行为分析**tab页。
- 区域2：关系筛选，筛选需要分析的关系。
- 区域3：缩略图筛选，通过下方缩略图进行范围定位，选中中间可移动缩略图。
- 区域4：选中行为，鼠标悬浮在柱状图上，可显示行为的明细信息。
- 区域5：点击+、-，可进行统计图的放大和缩小。

32.6.10.2 使用行为明细

操作步骤

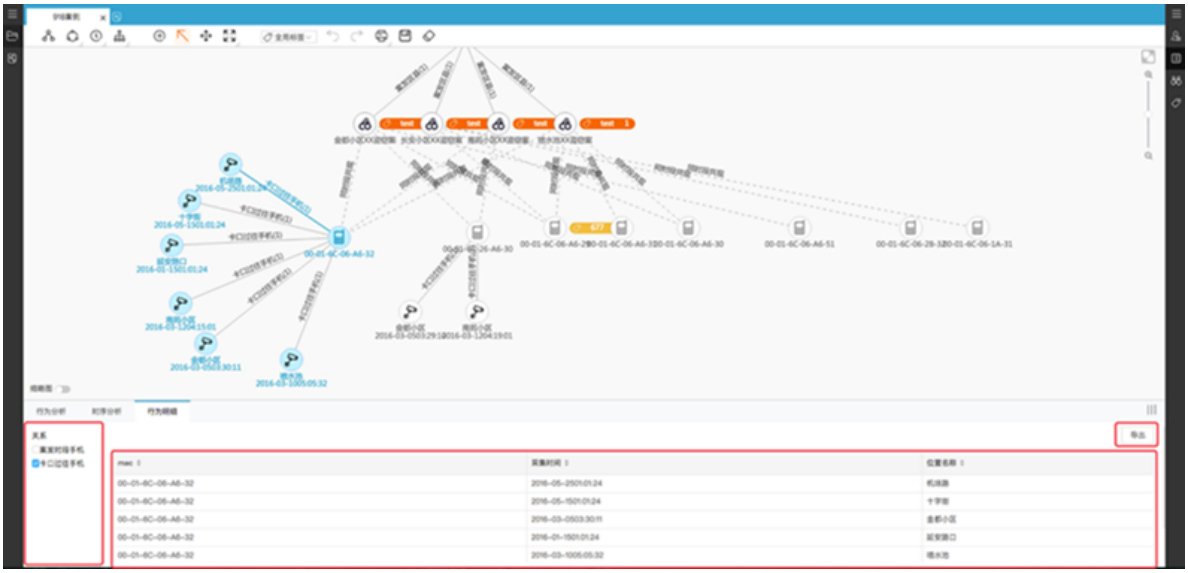
1. 在分析图区点击边（蓝色）或者选定点（蓝色），触发**行为明细**，如图 *1360:* 行为明细所示。

图 1360: 行为明细



2. 查看详情信息，如图 *1361:* 查看详情所示。

图 1361: 查看详情



多种关系可以通过Tab 页面进行切换选择。

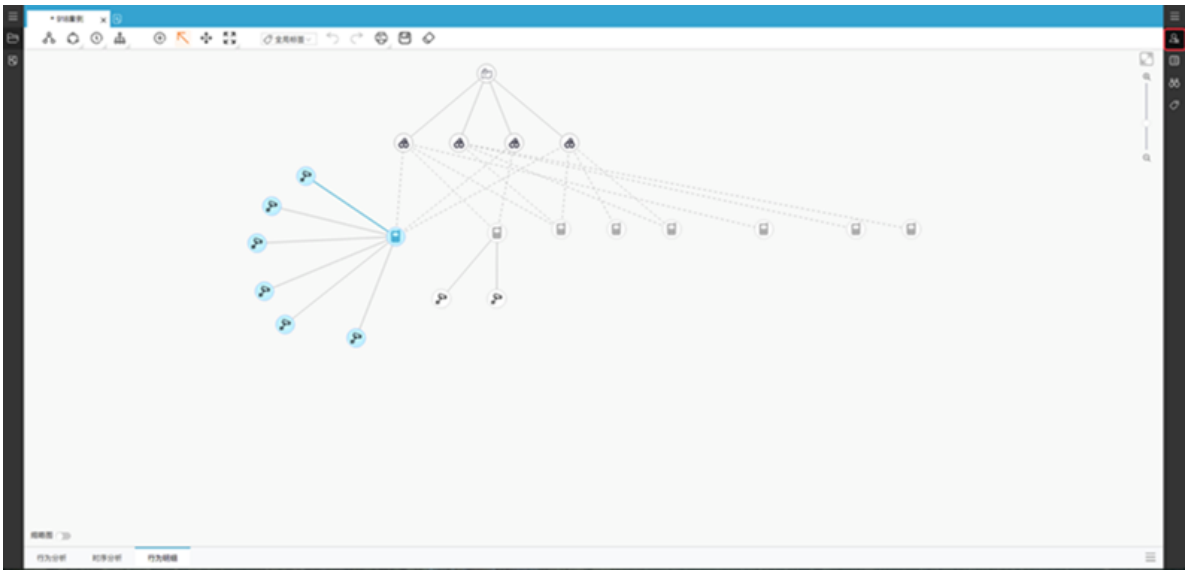
3. （可选）在**行为明细**Tab页面，单击**导出**，可以将明细导出。

32.6.10.3 管理对象信息

操作步骤

1. 单击右边模块如图 *1362:* 对象信息按钮所示的按钮，展示选中对象的基本信息，包含图标、对象ID值以及对象存在在系统中的属性值。

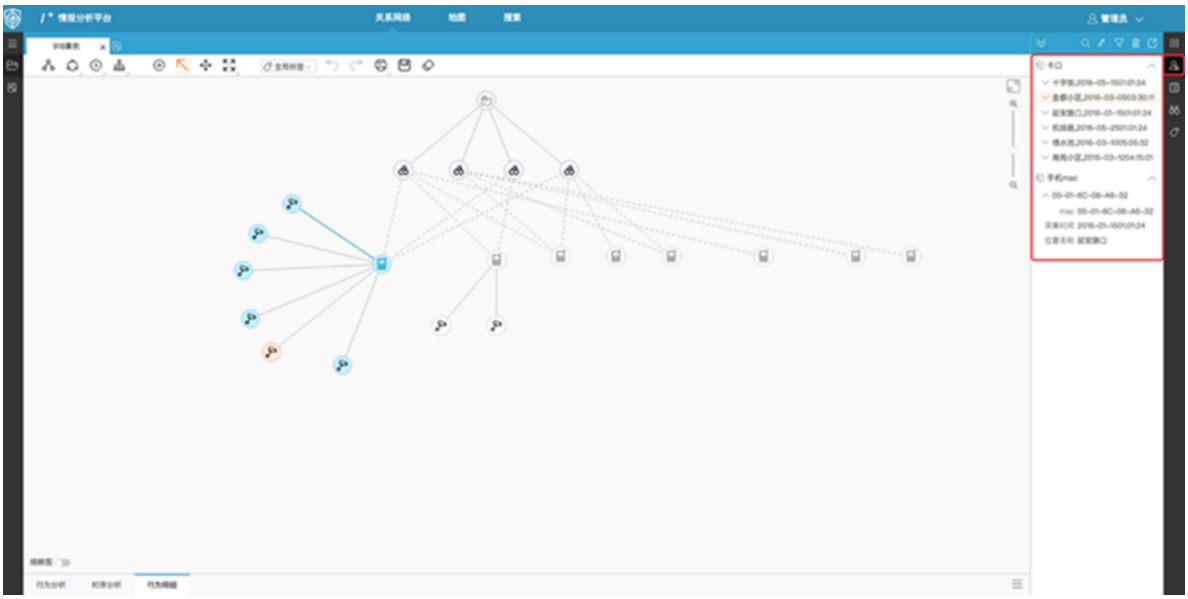
图 1362: 对象信息按钮



2. 单击收放按钮，可以展开或收起该点的信息，如图 *1363:* 展示对象信息所示。

   当鼠标单击左边主图上的的节点，右边节点信息会对应的展示选定的节点的信息，可以多选，如果多选，按住**Ctrl**键左键多选或框选，右边节点信息栏中，会同步展示指定节点的全部属性信息。

   图 1363: 展示对象信息



   面板右键可以针对选中内容进行如下操作：

   - 删除所选：在图区中将选中的对象删除。

- 在图区合并：将选中的两个或多个对象合并为一个节点，在图区中以一个文件夹的方式显示。
- 从合并节点中拆出：将选中对象从已合并的文件夹中拆分，单独显示在图区中。
- 在图区选中：在图区中高亮显示该对象。
- 在图区反选：在图区中高亮显示除该对象之外的对象。
- 选中关联点：在图区中高亮显示与该对象存在关联关系的对象。

32.6.10.4 管理统计信息

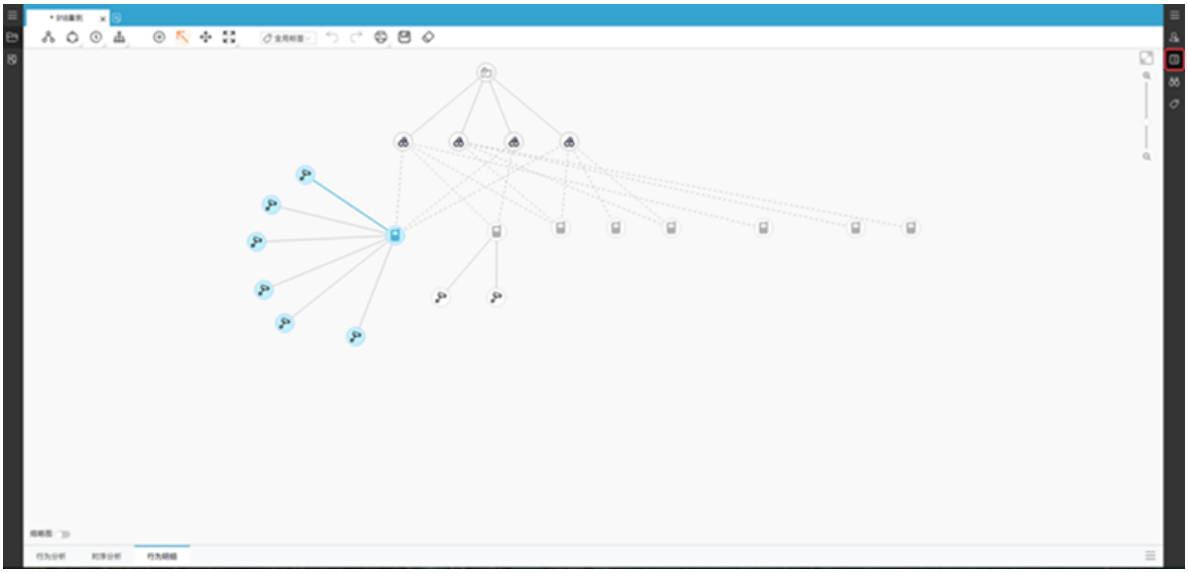
背景信息

右侧的统计面板展示对选中部分的对象或关系统计。

操作步骤

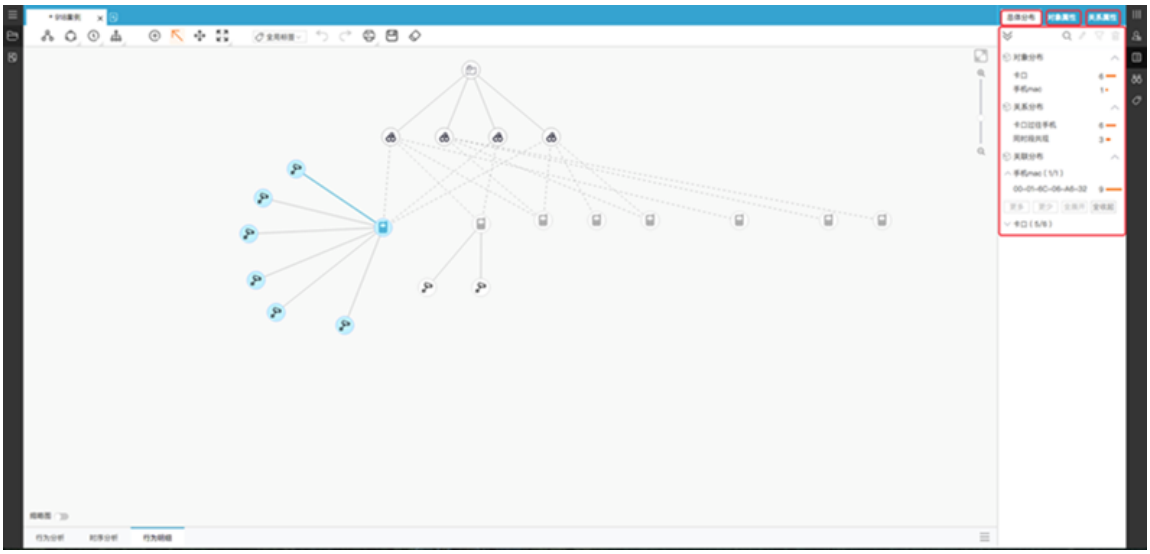
1. 单击如图 *1364: 统计面板按钮*所示按钮，显示统计面板。

图 1364: 统计面板按钮



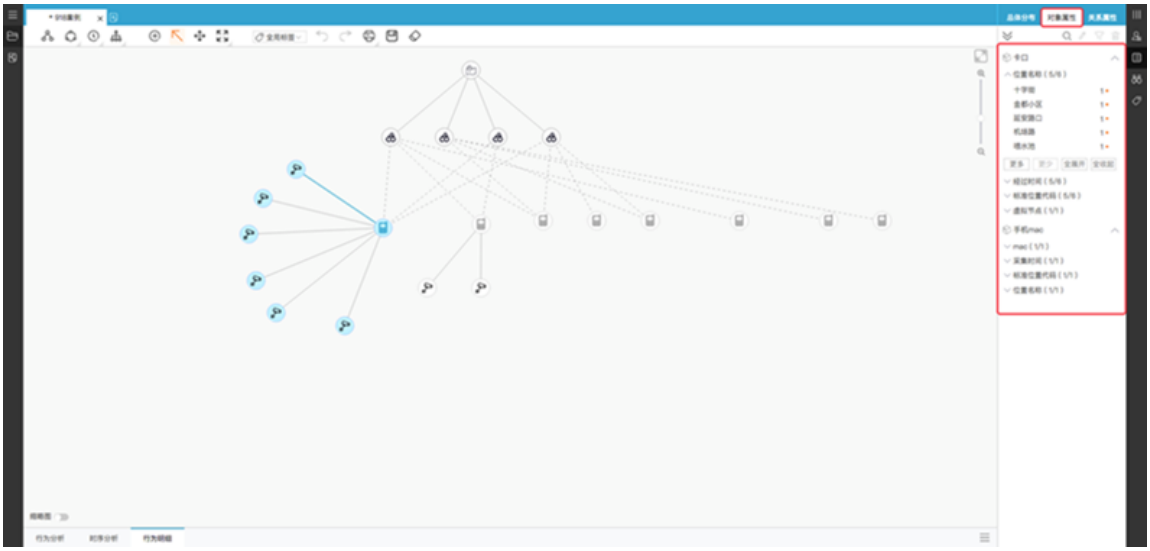
2. 单击收放按钮，可以展开或收起选中对象或关系的属性统计信息。
   - 查看总体分布，如图 *1365: 总体分布*所示。

图 1365: 总体分布



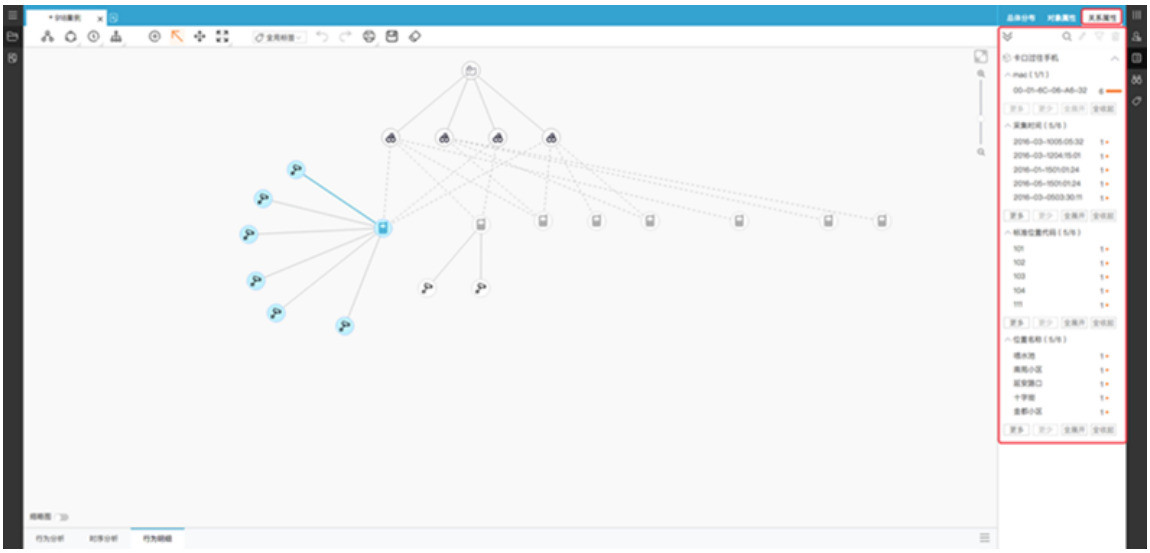
- 查看对象属性，如图 *1366: 对象属性*所示。

图 1366: 对象属性



- 查看关系属性，如图 *1367: 关系属性*所示。

图 1367: 关系属性



面板右键可以针对选中内容进行如下操作：

- 删除所选：在图区中将选中的对象删除。
- 在图区合并：将选中的两个或多个对象合并为一个节点，在图区中以一个文件夹的方式显示。
- 从合并节点中拆出：将选中对象从已合并的文件夹中拆分，单独显示在图区中。
- 在图区选中：在图区中高亮显示该对象。
- 在图区反选：在图区中高亮显示除该对象之外的对象。
- 选中关联点：在图区中高亮显示与该对象存在关联关系的对象。

32.6.10.5 使用群集统计

背景信息

群集统计是统计关系网络分析中的群集分布。一般常用于群集分析中，输入多个节点后，通过I+分析这些分析之间相互关联关系，结果呈现，通过群集统计，能查看结果的群集分布。其中，群集是指一群对象节点，任意两两对象节点在拓扑上拥有联通路径，其中合并节点作为一个联通桥，在拓扑上认为其内部全部联通。

操作步骤

单击右侧如图 *1368:* 群集统计所示按钮，显示群集统计面板。

图 1368: 群集统计



该关系网络分析中存在三个群集。

在群集统计中，我们将所有孤立的节点统称为一个群集，称之为**孤立节点**。

如图 *1368: 群集统计*所示，群集统计的 list 中，分别展示每个群集中度数最多节点 label（孤立节点群体除外），以及该群体中对象节点的个数。其中 list 的展示顺序为，估计节点群体最上面，后面按照群集个数中依次降序排列。

群集统计的右键，以及交互操作，同对象统计一样。

说明: 群集统计的范围是关系网络分析的全部节点，与用户在分析中是否选中节点无关。

32.6.10.6 使用标签统计

背景信息

标签统计是为了统计关系网络中的对象节点的标签信息。

在 I+ 系统中，标签分为两类：

- 系统标签

  系统标签是指业务系统对某些节点定义的标签，如红黑名单等，会展示在对象 icon 左侧。

- 用户标签

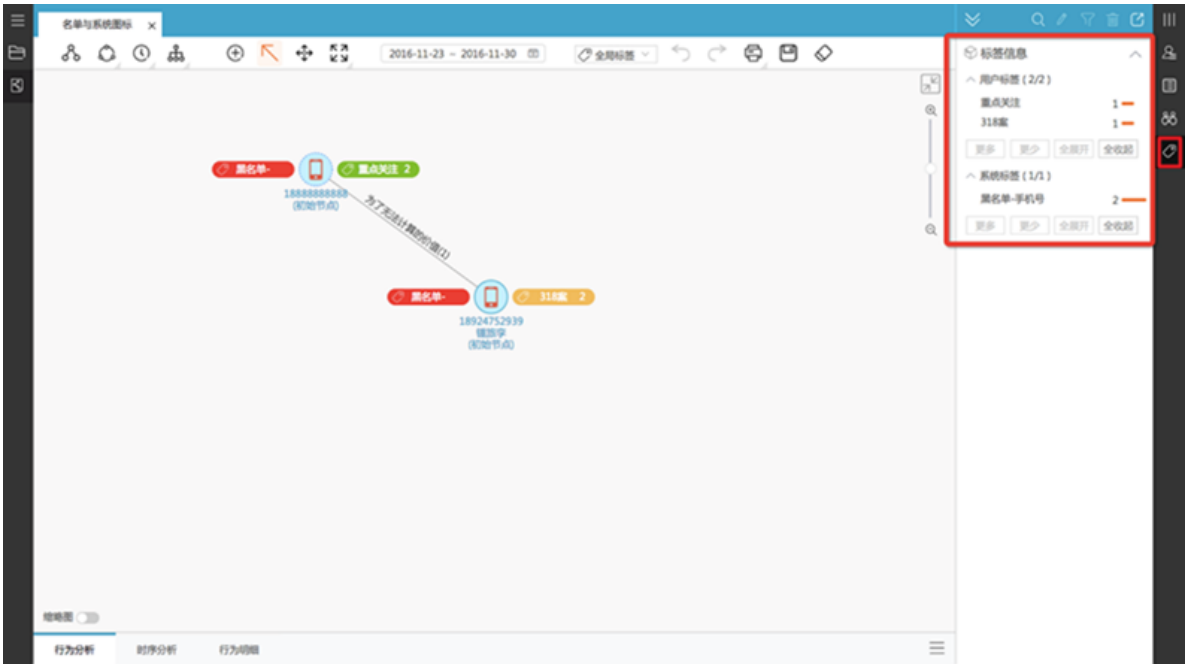
  用户标签是指每个 I+ 用户对某些节点通过 I+ 添加的标签，会展示在对象 icon 右侧。

操作步骤

选中对象节点，单击如图 *1369: 标签统计*所示按钮，展示标签统计面板。

说明: 标签统计是根据用户在关系网络分析中选中的对象节点来统计的，如果用户没有选中对象节点，那标签统计中将没有结果呈现。

图 1369: 标签统计



如图 *1369: 标签统计*所示，标签统计中，分别统计用户标签和系统标签。右侧面板中统计的数字为拥有该标签的对象个数。而标签后面的数字则表示，认同该对象该标签的用户数。

标签统计中的右键交互与统计信息中一样，但是有一点需要注意，右键中的删除是指删除该标签对应的实体，而不是删除标签本身。

32.6.11 图区布局

关系网络分析图区支持矩阵布局、圆环布局、横直线布局、竖直线布局、力导向布局和层次布局。

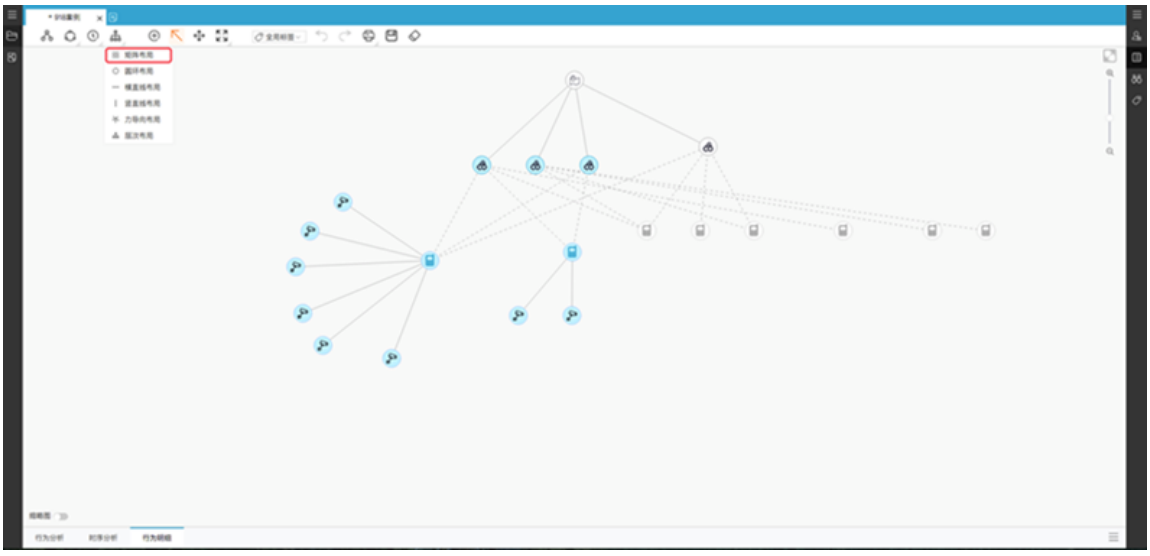
- 矩阵布局

  矩阵布局按照矩阵形状，将对象整齐顺序排列，在分析过程中帮助用户将信息归整，方便其对信息梳理。

  a) 选中对象节点之后，再选择**矩阵布局**，如图 *1370: 矩阵布局*所示。

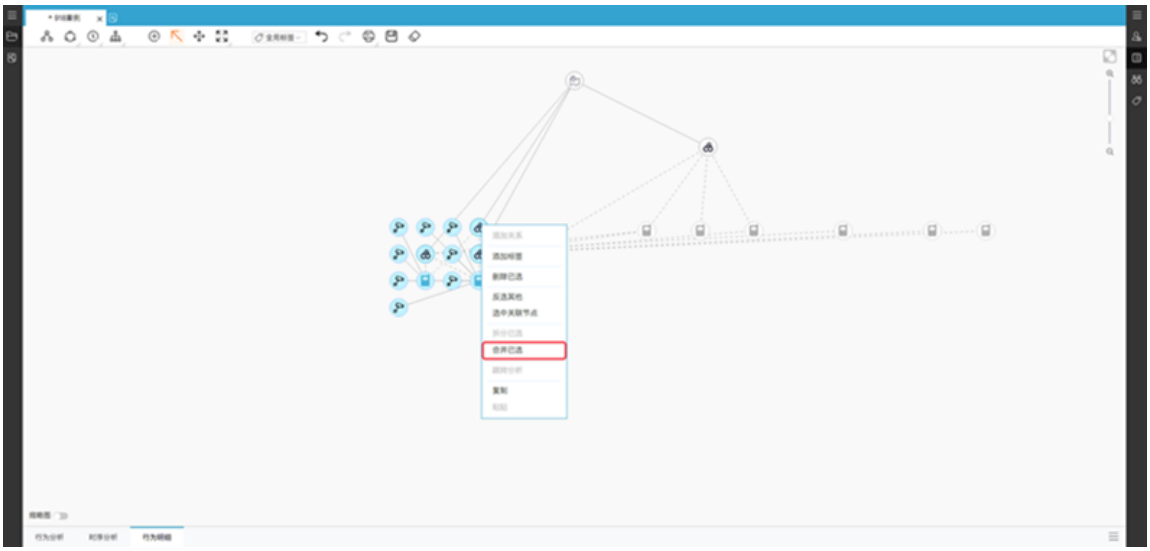
  选择对象节点的范围可以是全部节点或者部分节点。

  图 1370: 矩阵布局



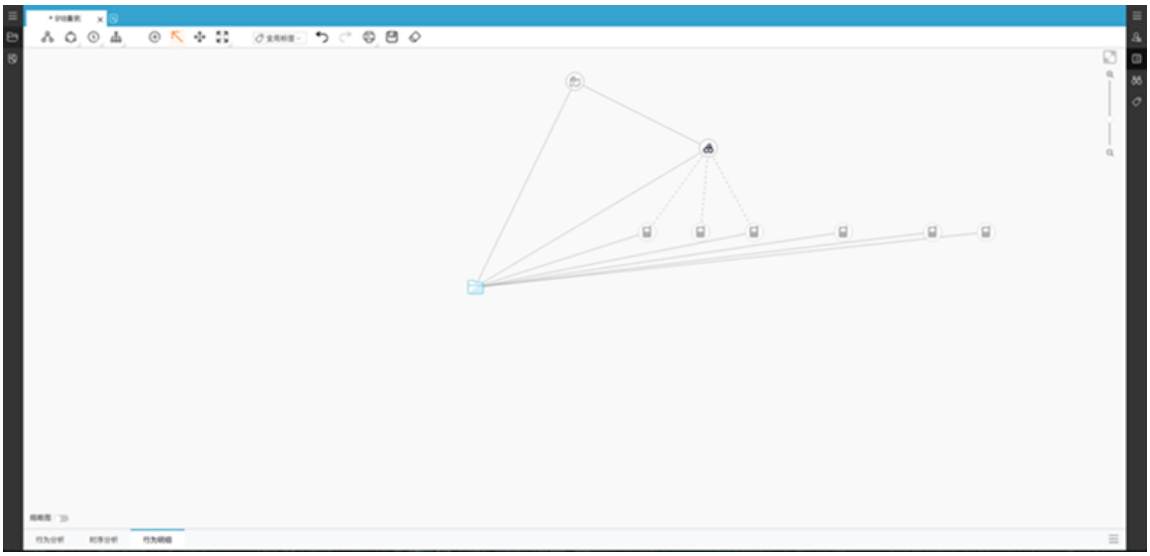
b) 矩阵布局可以配合**合并节点**功能一起使用，如图 *1371: 合并节点*所示，能够在信息量大的情况下，将分析后的信息归整合并。

图 1371: 合并节点



合并后的效果如图 *1372: 合并结果*所示。

图 1372: 合并结果



- 圆环布局

  圆环布局按照圆环形状，在分析过程中帮助用户将信息归整，采用环形形状将对象均匀分布于环形圈上，将其网络关联在圈内呈现，方便用户分析关键节点。

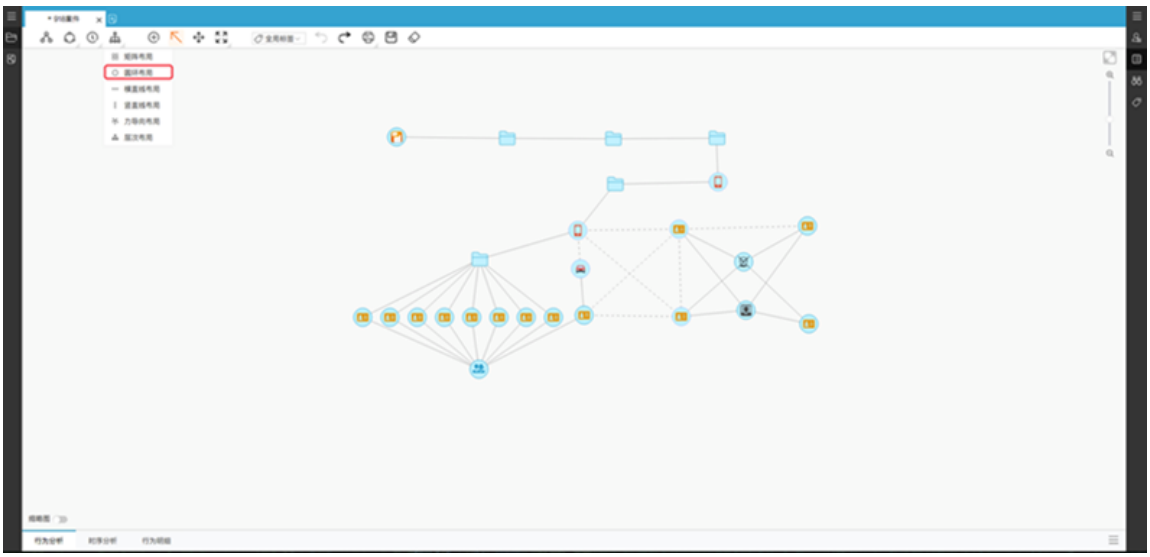
  操作方法：选中对象节点后，再选择**圆环布局**。

  选择对象节点的范围可以是全部节点或者部分节点：

  - 全部选中

    所有节点选中状态，都为蓝色外圈，如图 *1373:* 圆环布局全部选中所示。

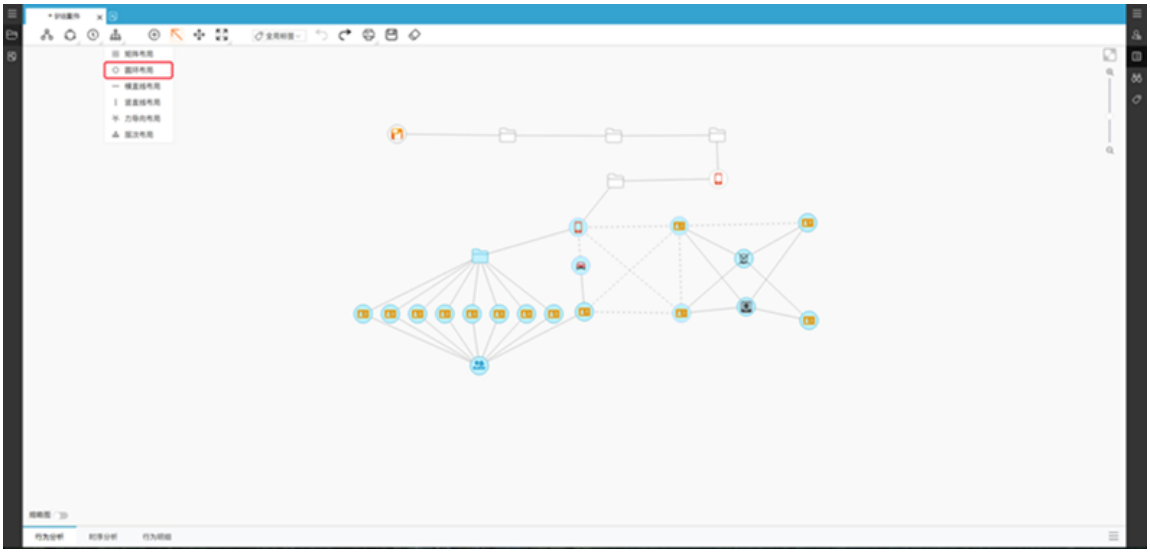
    图 1373: 圆环布局全部选中



  - 部分选中

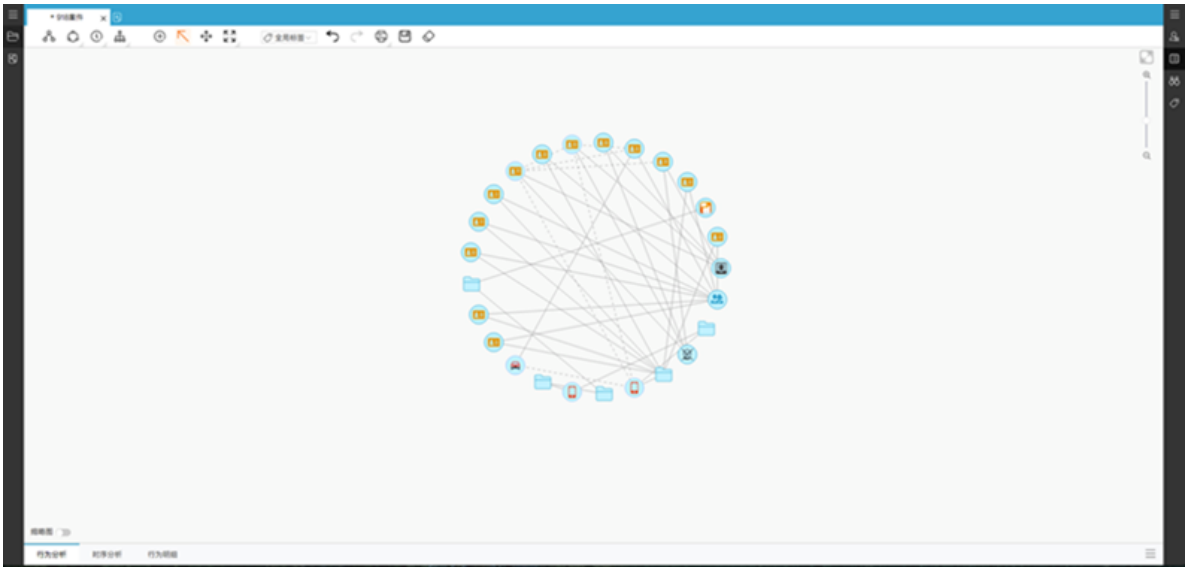
蓝色外圈节点为选中节点，如图 *1374:* 圆环布局部分选中所示。

图 1374: 圆环布局部分选中



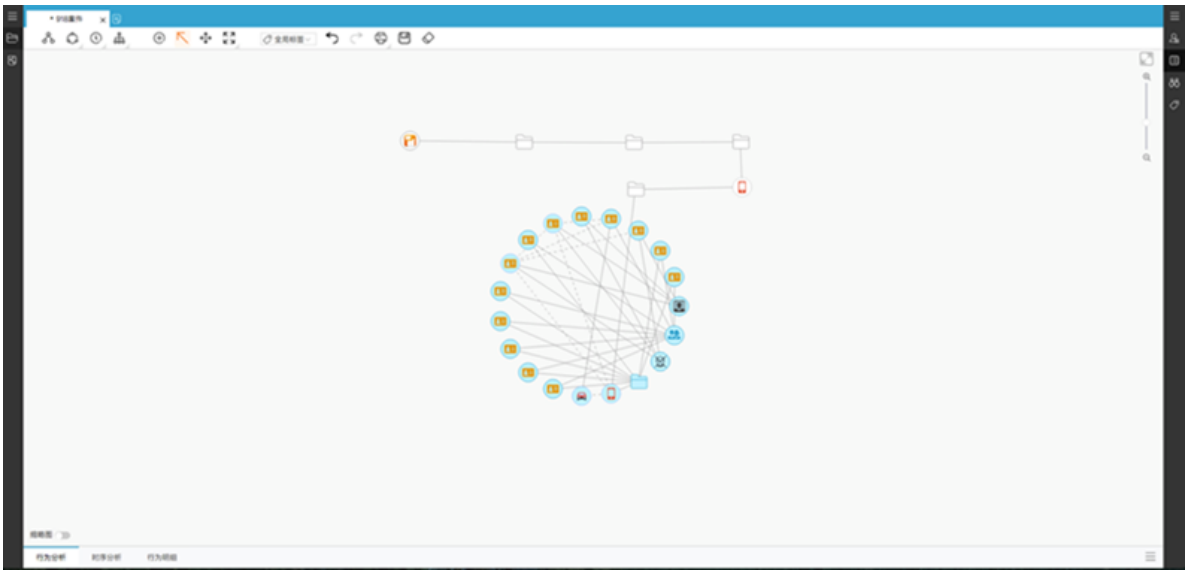
全部选中后的圆环布局效果如图 *1375:* 圆环布局全部选中效果所示。

图 1375: 圆环布局全部选中效果



部分选中后的圆环布局效果如图 *1376:* 圆环布局部分选中效果所示。

图 1376: 圆环布局部分选中效果



- 横直线布局

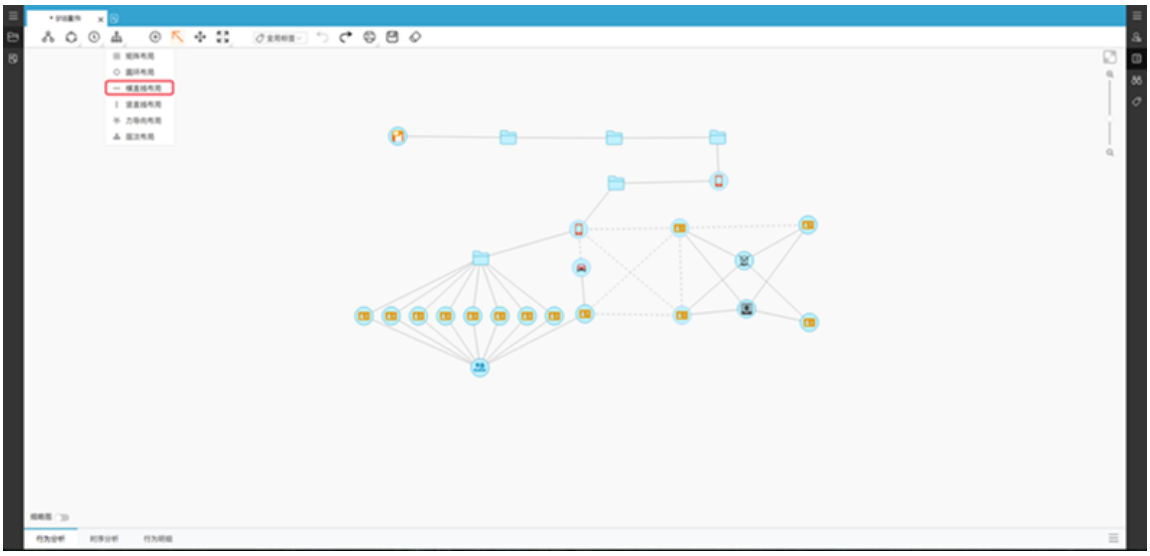
  横直线布局按照横向直线将对象整齐排列，将信息格式化归整，方便用户从能从横向流程分析。

  操作方法：选中对象节点后，再选择**横直线布局**。

  选择对象节点的范围可以是全部节点或者部分节点：

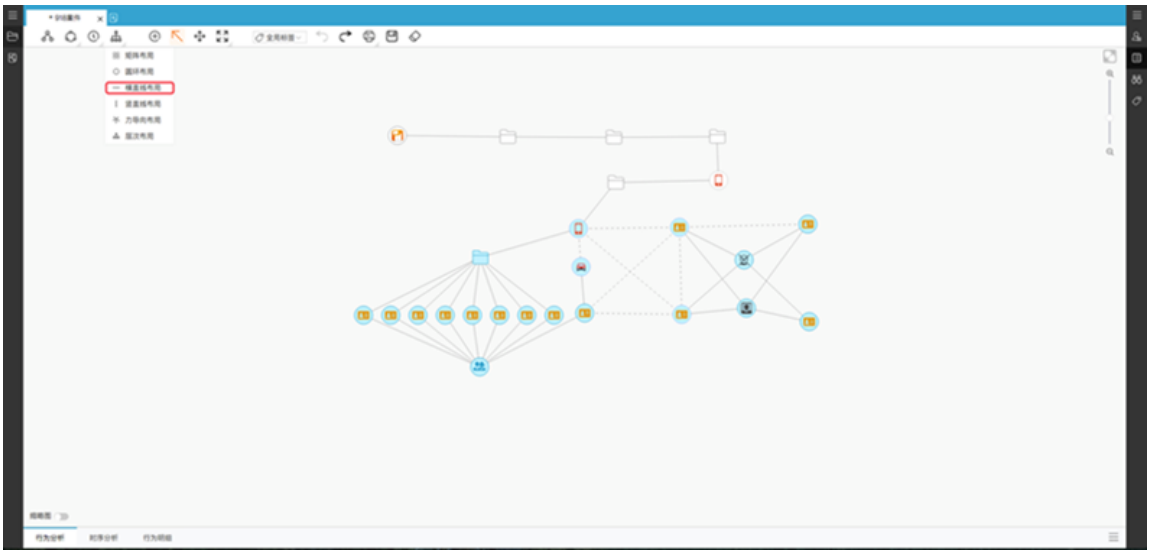
  - 全部选中，如图 *1377:* 横直线布局全部选中所示。

    图 1377: 横直线布局全部选中



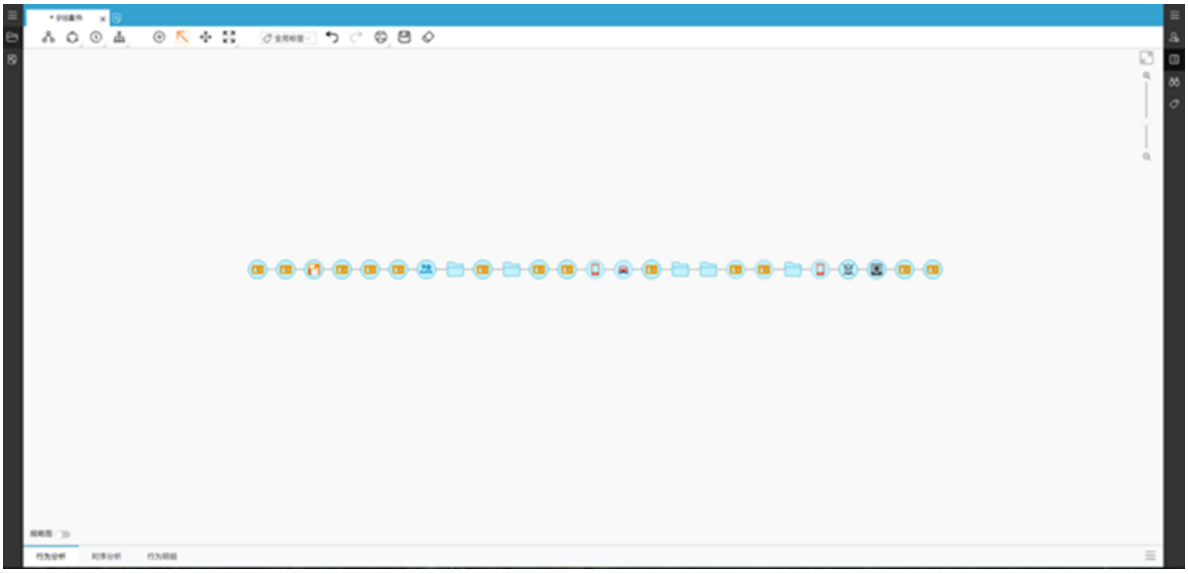
  - 部分选中，如图 *1378:* 横直线布局部分选中所示。

    图 1378: 横直线布局部分选中



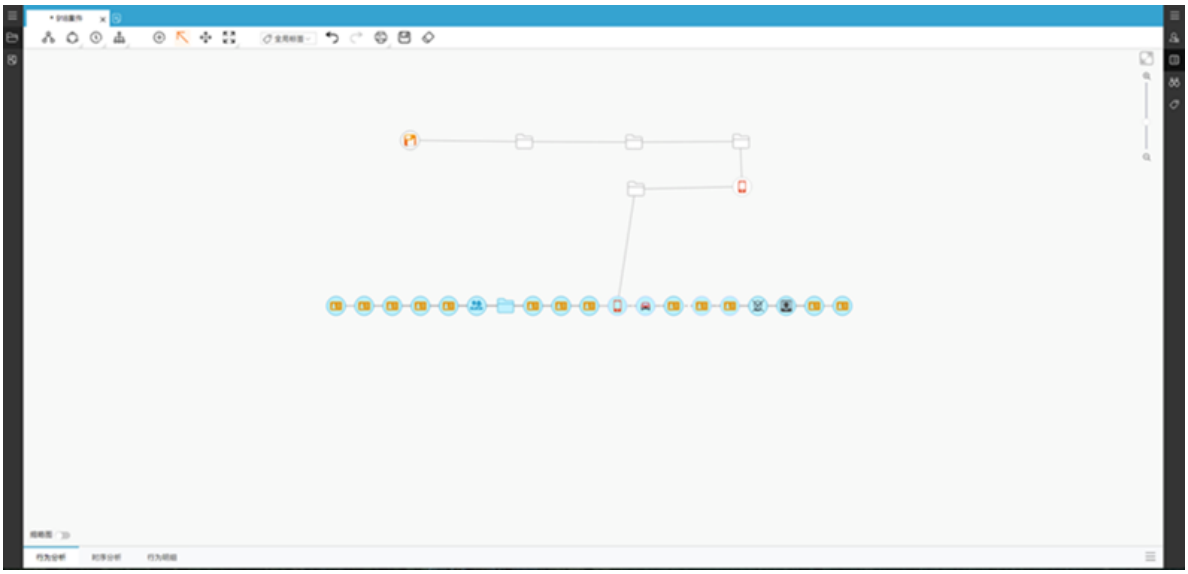
全部选中后的横直线布局效果如图 *1379:* 横直线布局全部选中效果所示。

图 1379: 横直线布局全部选中效果



部分选中后的横直线布局效果如图 *1380:* 横直线布局部分选中效果所示。

图 1380: 横直线布局部分选中效果



- 竖直线布局

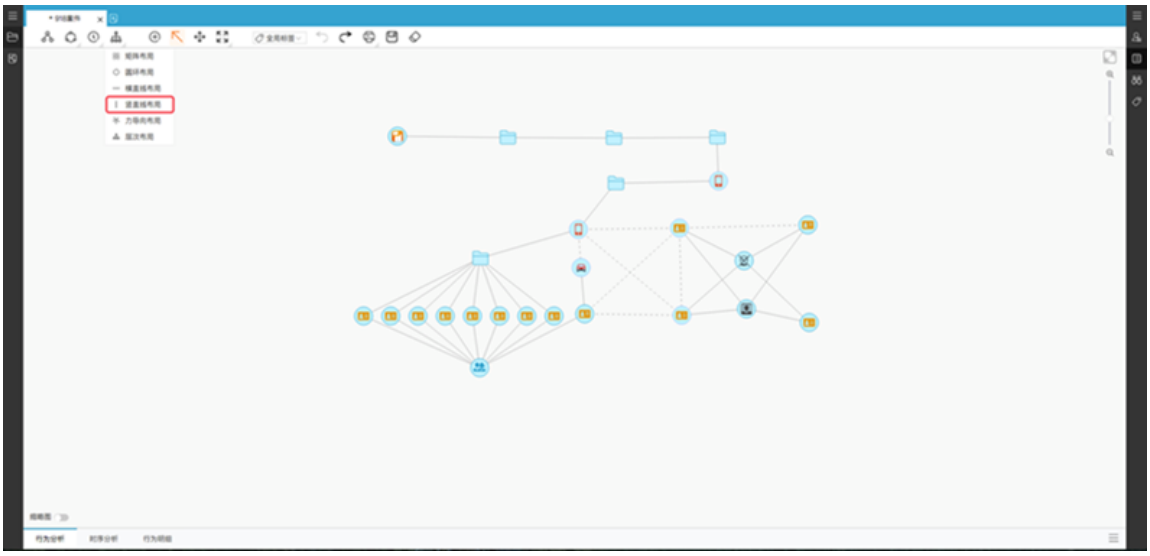
  竖直线布局按照横向直线将对象整齐排列，将信息格式化归整，方便用户从能从横向流程分析。

  操作方法：选中对象节点后，再选择**竖直线布局**。

  选择对象节点的范围可以是全部节点或者部分节点：

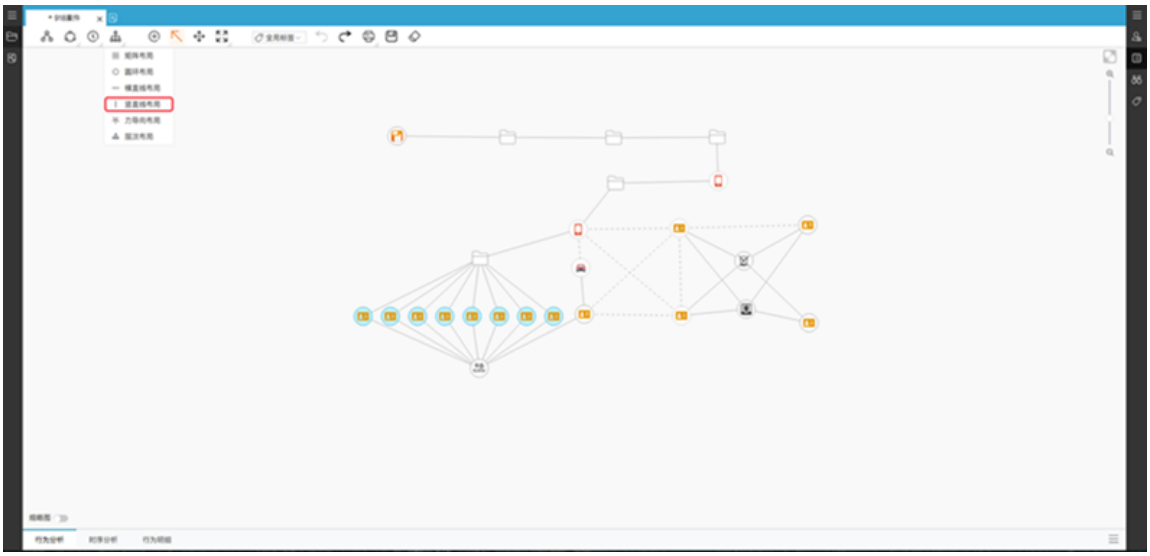
  - 全部选中，如图 *1381:* 竖直线布局全部选中所示。

    图 1381: 竖直线布局全部选中



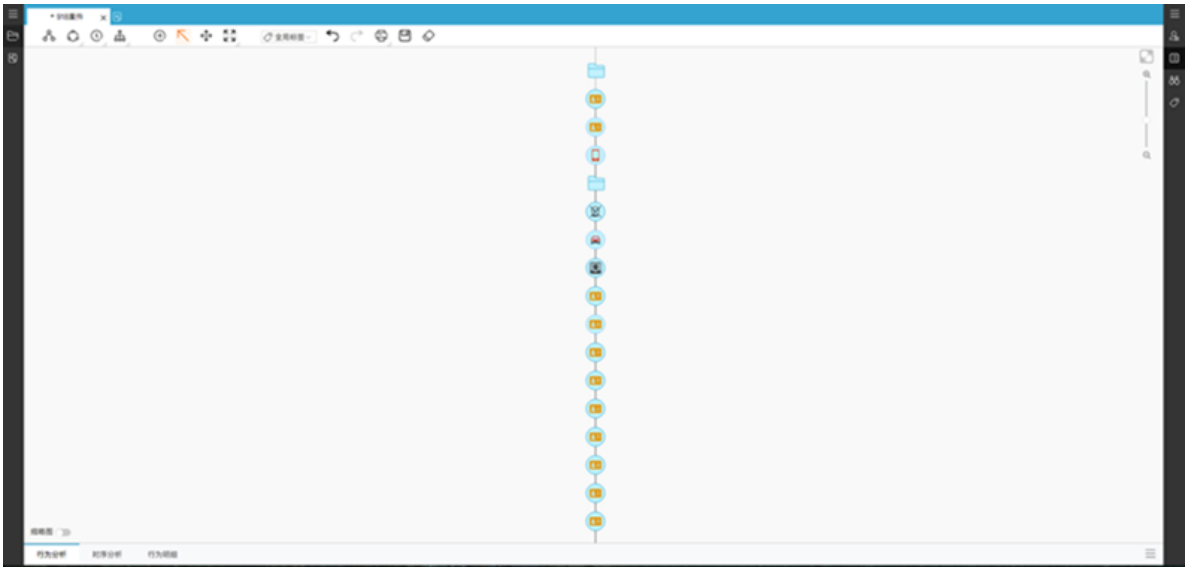
  - 部分选中，如图 *1382:* 竖直线布局部分选中所示。

    图 1382: 竖直线布局部分选中



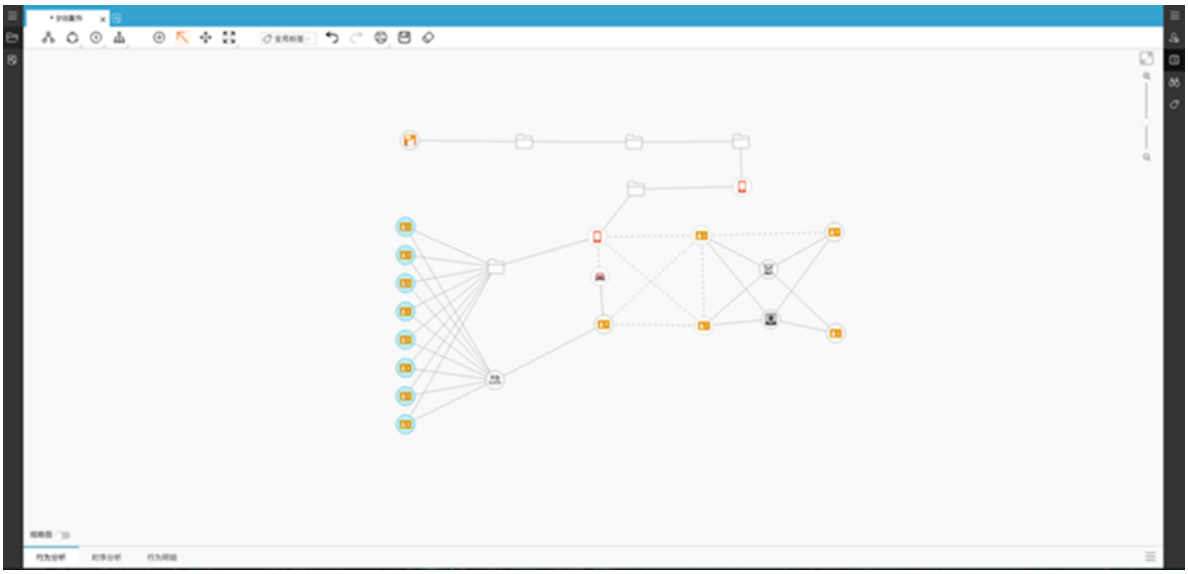
全部选中后的竖直线布局效果如图 *1383:* 竖直线布局全部选中效果所示。

图 1383: 竖直线布局全部选中效果



部分选中后的竖直线布局效果如图 *1384:* 竖直线布局部分选中效果所示。

图 1384: 竖直线布局部分选中效果



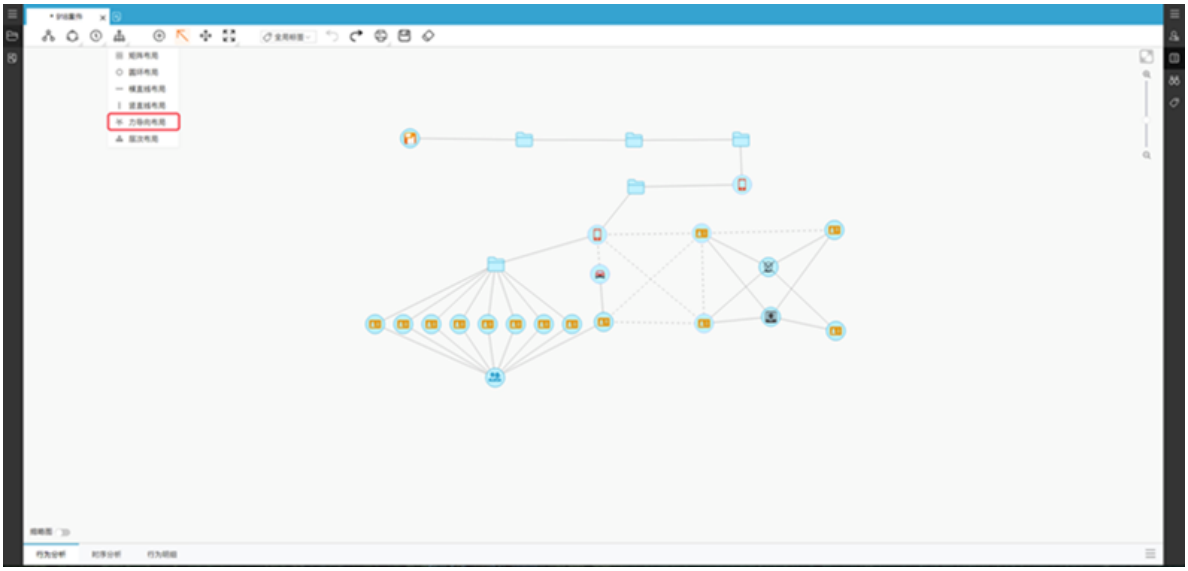
- 力导向布局

  力导向布局算法主要应用于复杂网络可视化。力引导布局最早由 Peter Dades 在1984年的**启发式画图算法**文章中提出。目的是减少布局中边的交叉，尽量保持边长一致。主要引入库伦斥力和胡克弹力，考虑阻尼衰减（这就是为什么我们拉动力引导图，它能很快稳定回来的原因）。事先定义好图里的点，边的权重等信息，力引导图可以根据实时状态自动完成很好的聚类，方便地看出点之间的亲疏关系。

  力导向布局针对全图布局，关系网络分析图区所有对象节点和关联关系都参与布局计算。

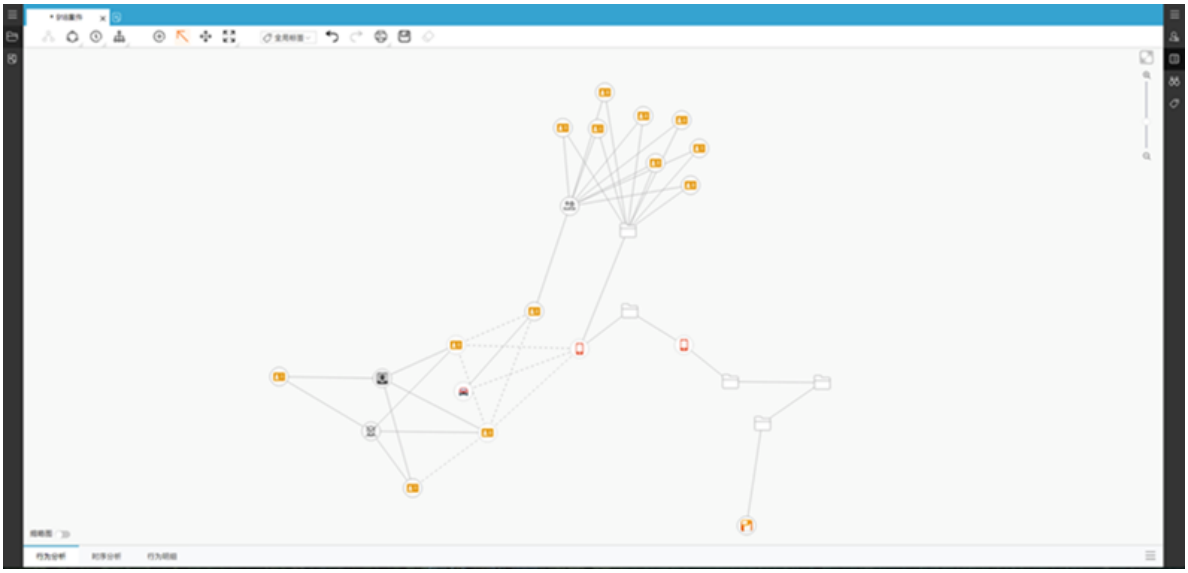
  操作方法：选中对象节点后，再选择**力导向布局**，如图 *1385:* 力导向布局所示。

  图 1385: 力导向布局



选中后，力导向布局的效果如图 *1386: 力导向布局效果*所示。

图 1386: 力导向布局效果



- 层次布局

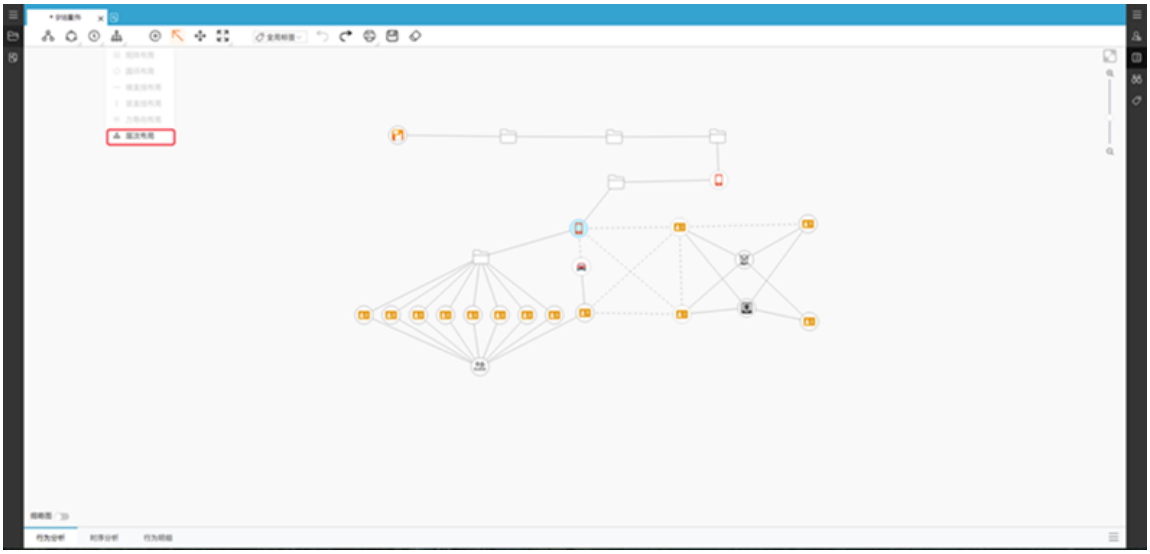
  层次布局按照树状结构将对象排列布局，主要应用在家族图谱，企业组织结构上。

  操作方法：选中对象节点，选择**层次布局**。

  对象节点可以选中单个节点或多个节点，选中节点为布局树的根节点。

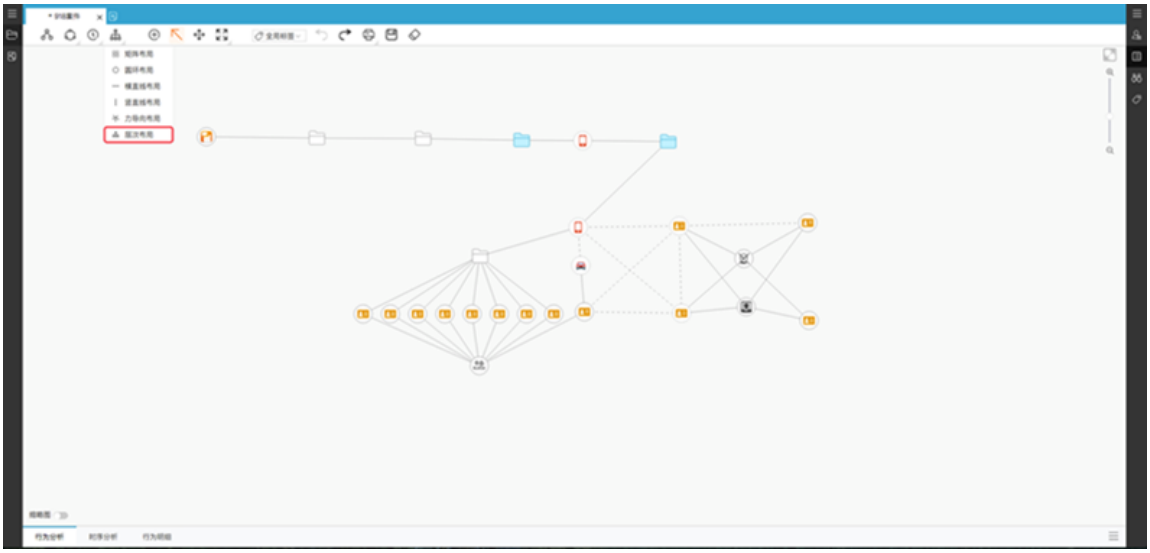
  - 选中单个节点，如图 *1387: 层次布局选中单个节点*所示。

    图 1387: 层次布局选中单个节点



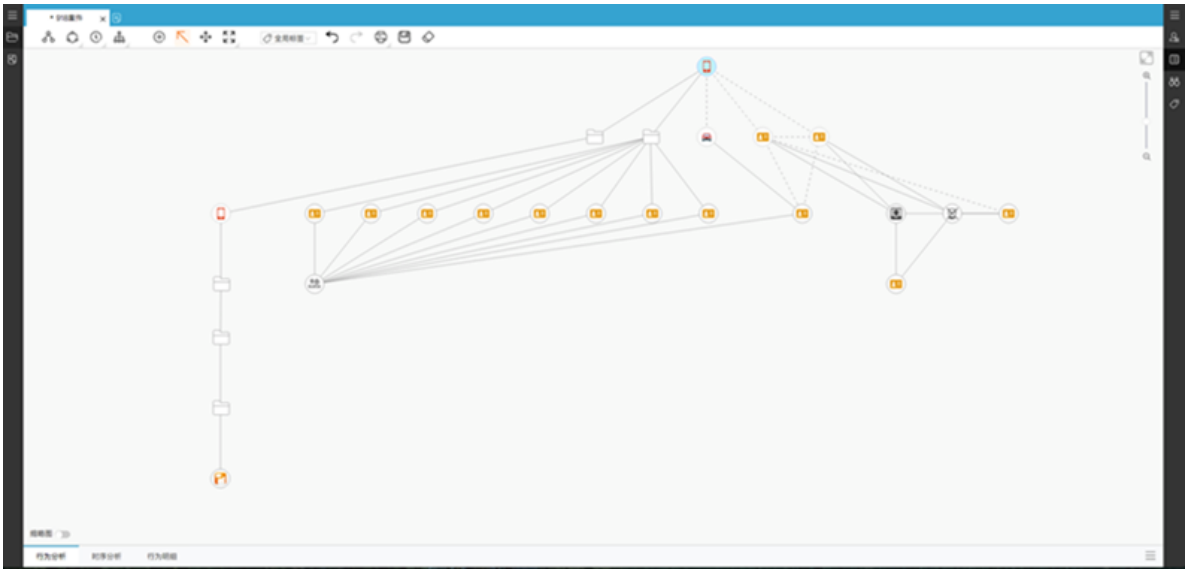
- 选中多个节点，如图 *1388: 层次布局选中多个节点*所示。

图 1388: 层次布局选中多个节点



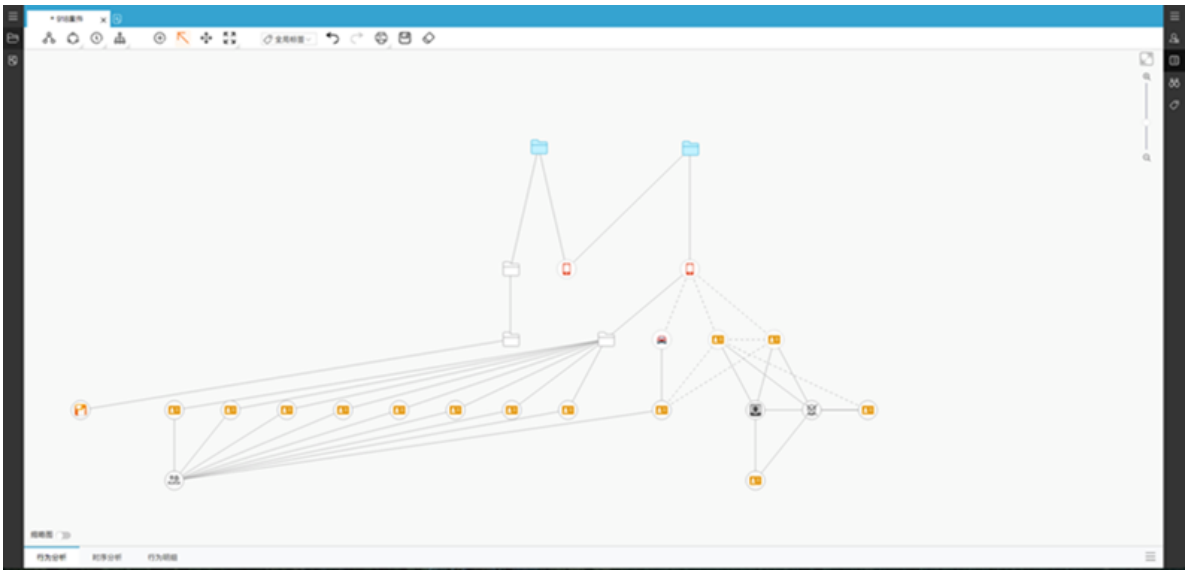
选中单个节点的层次布局效果如图 *1389: 层次布局选中单个节点效果*所示。

图 1389: 层次布局选中单个节点效果



选中多个节点的层次布局效果如图 *1390: 层次布局选中多个节点效果*所示。

图 1390: 层次布局选中多个节点效果



32.6.12 功能操作

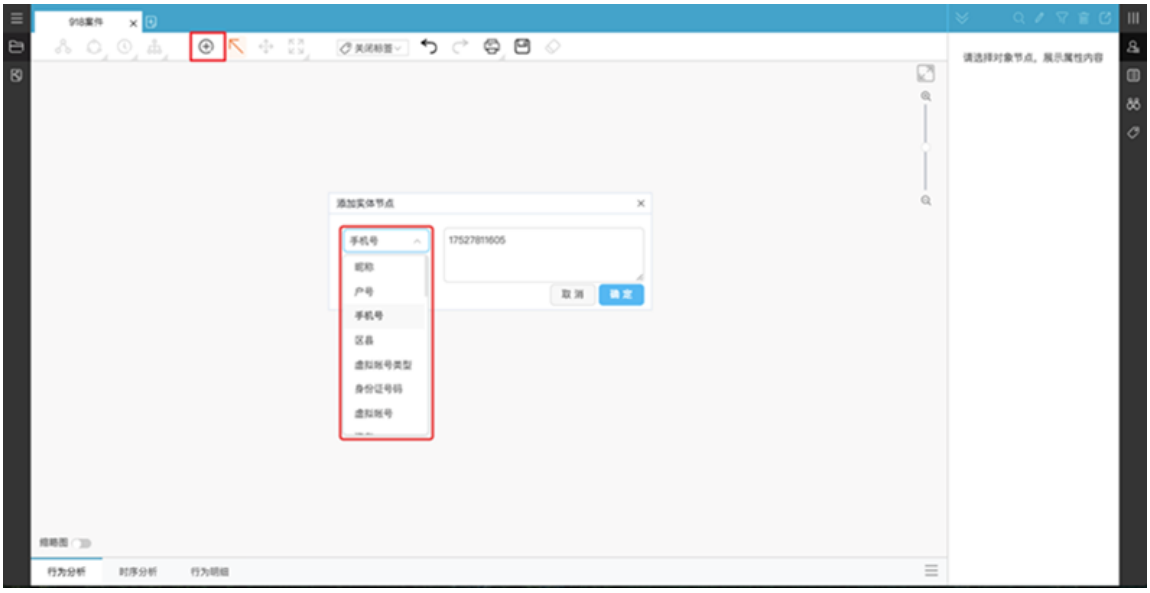
功能操作主要围绕关系网络分析图区展开，针对分析内容做功能操作。

32.6.12.1 新增节点

- 从工具栏新增节点

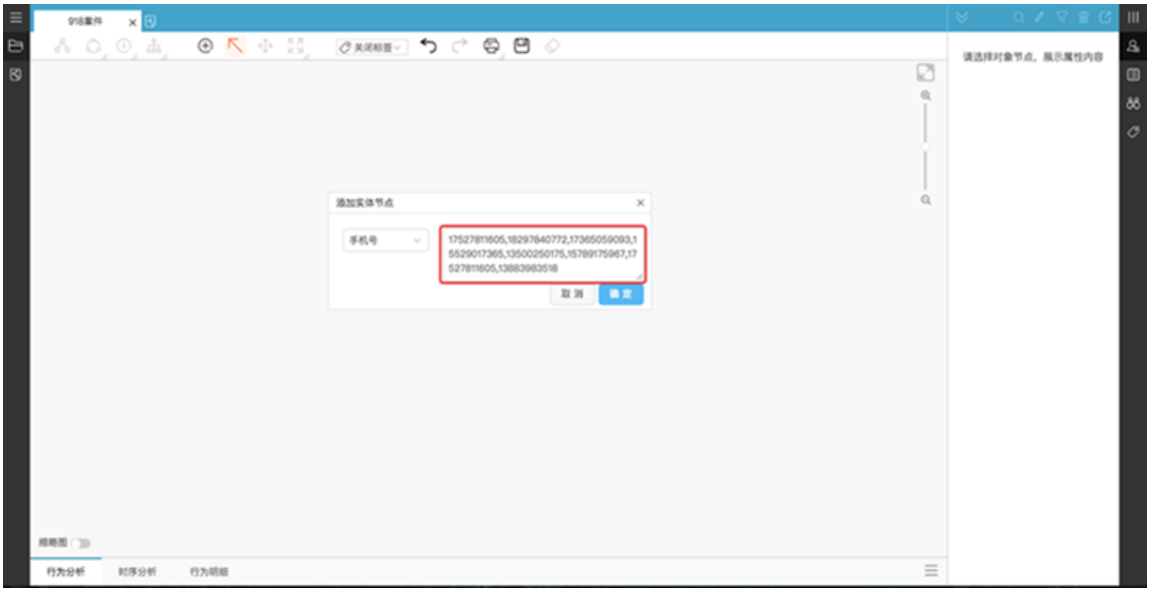
  a) 单击  后，在图区空白处单击并选择实体类型，如图 *1391:* 新增节点按钮所示。

图 1391: 新增节点按钮



b) 输入具体内容，多个节点之间用英文逗号隔开，如图 *1392:* 输入节点信息所示。

图 1392: 输入节点信息



c) 单击**确定**。

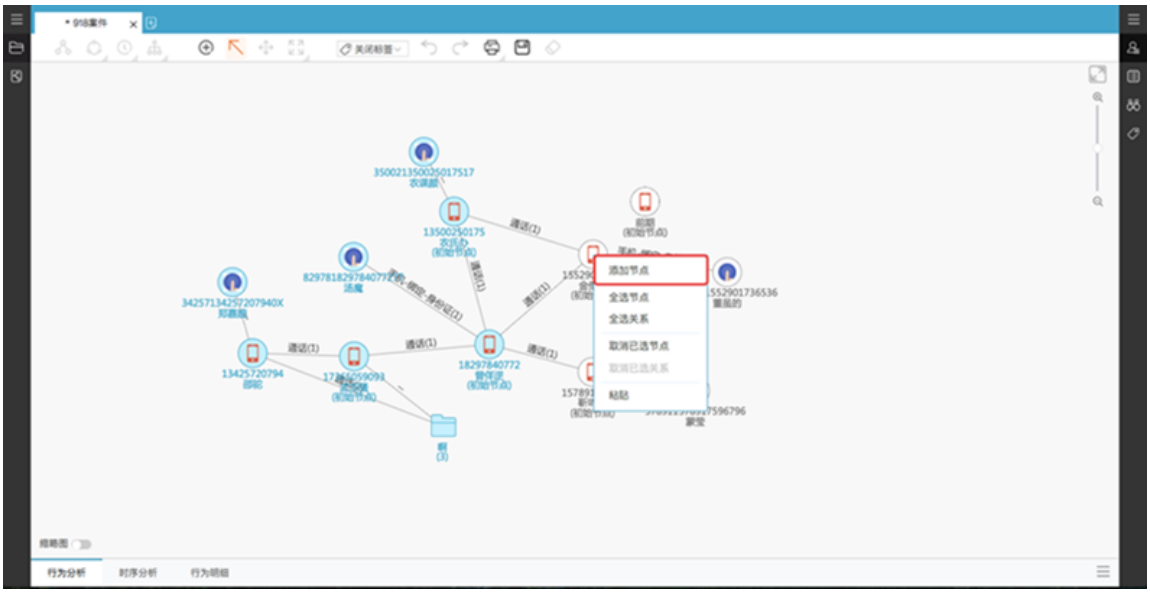
- 在页面中新增节点

  适合针对特定位置增加节点。

  a) 在页面空白处单击鼠标右键，选择**添加节点**。

  或者您也可以在未选中节点或未选中的边上右键单击，选择**添加节点**，如图 *1393:* 页面中新增节点所示。

  图 1393: 页面中新增节点



b) 输入节点具体内容，多个节点之间用英文逗号隔开。

c) 单击**确定**。

32.6.12.2 画布拖动

背景信息

关系网络分析图区支持如下两种模式操作：

- 选择模式

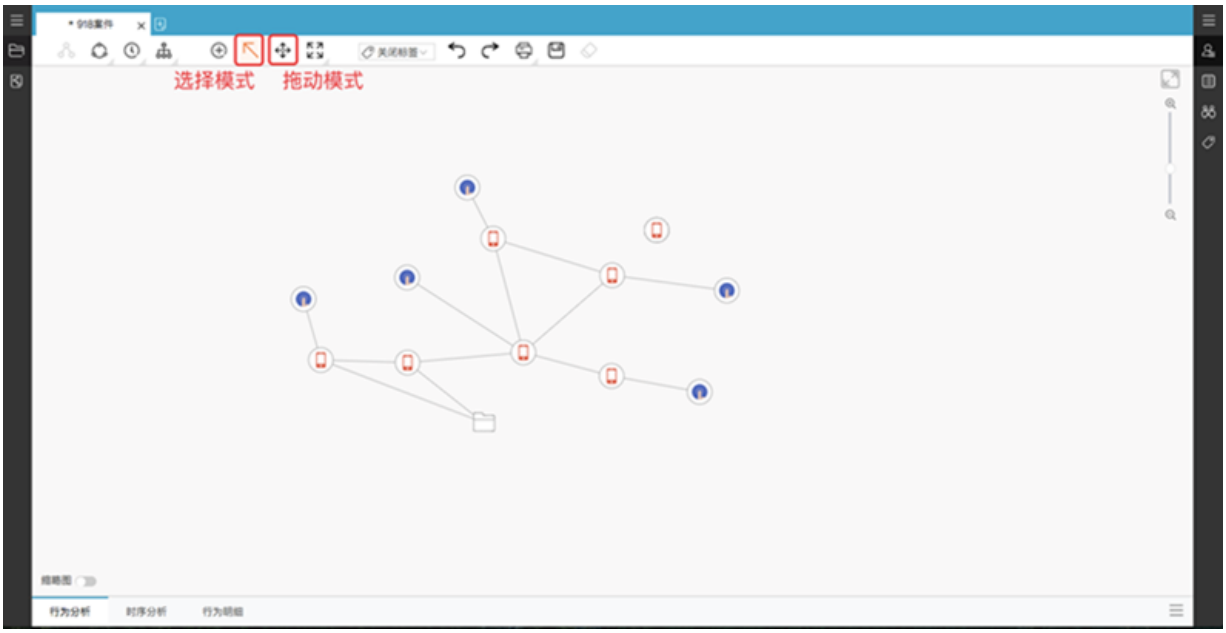
  为默认状态，用户可以对图区做细粒度操作，对节点拖动布局、选中及右键触发相应操作等。

- 拖动模式

  对分析图区整体移动，方便用户比对分析。

画布拖动模式切换按钮如图 *1394:* 画布拖动模式所示。

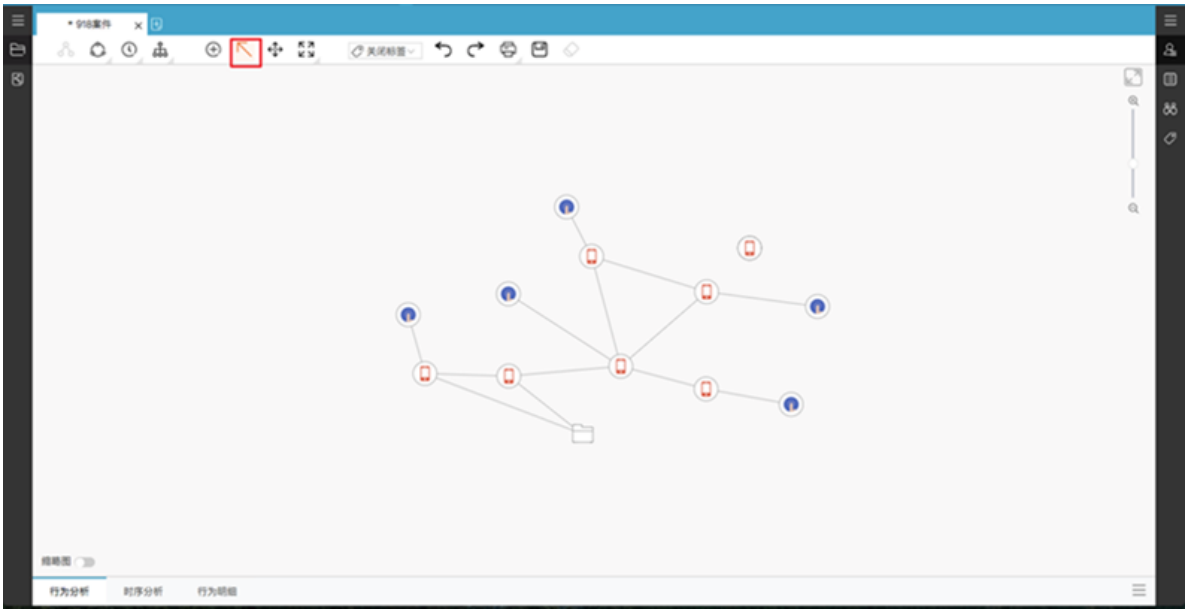
图 1394: 画布拖动模式



操作步骤

1. 在工具栏中选中拖动画布的操作按钮，图区鼠标切换为**手型**。
2. 使用鼠标全图分析拖动。
3. 在工具栏中单击操作按钮，切换到选择模式，如图 *1395:* 切换至选择模式所示。

图 1395: 切换至选择模式



32.6.12.3 图区缩放

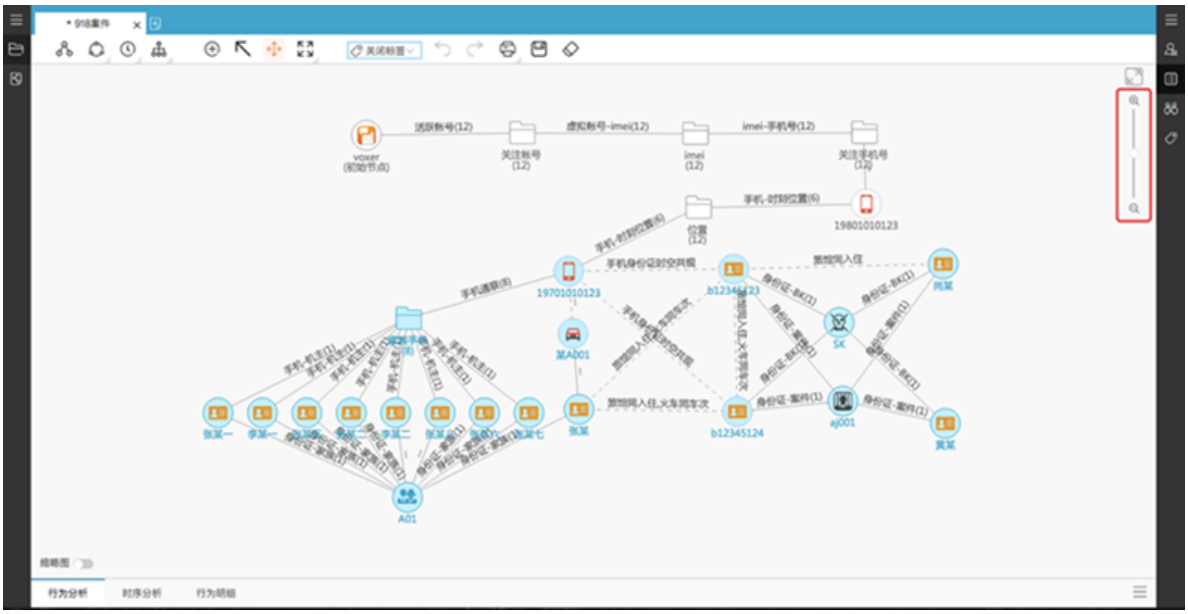
背景信息

针对分析图区内容进行缩放操作，方便用户从整体或局部来对比分析。

操作步骤

1. 拖动图区右侧的放大缩小按钮，如图 *1396: 放大缩小按钮*所示。

   图 1396: 放大缩小按钮



2. 进行缩放操作。
   - 向上拖动，放大图区。
   - 向下拖动，缩小图区。
   - 双击中间的小圆点，恢复原始比例。

32.6.12.4 节点锁定

背景信息

节点锁定是为了方便用户对于某些特殊节点锁定，即位置在空间画布中不变。其功能键为钉子形状。

操作步骤

1. 在画布中，选中一个节点。
2. 单击节点锁定图标，如图 *1397: 节点锁定*所示。

   当没有节点选中的时候，锁定节点的功能键为灰色；当有节点选中的时候，功能键正常显示。

   图 1397: 节点锁定



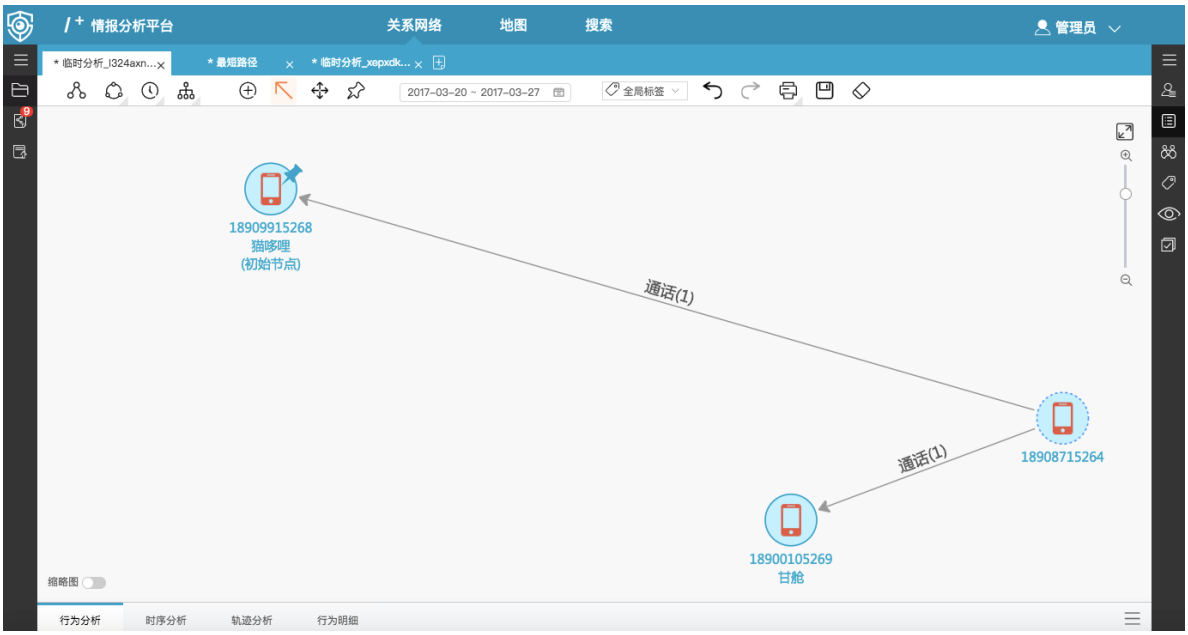
3. 节点锁定后，状态如图 *1398:* 节点锁定效果所示。

图 1398: 节点锁定效果



此时该节点的位置不会被改变（除了画布整体移动外），如图 *1399:* 节点锁定效果2所示。

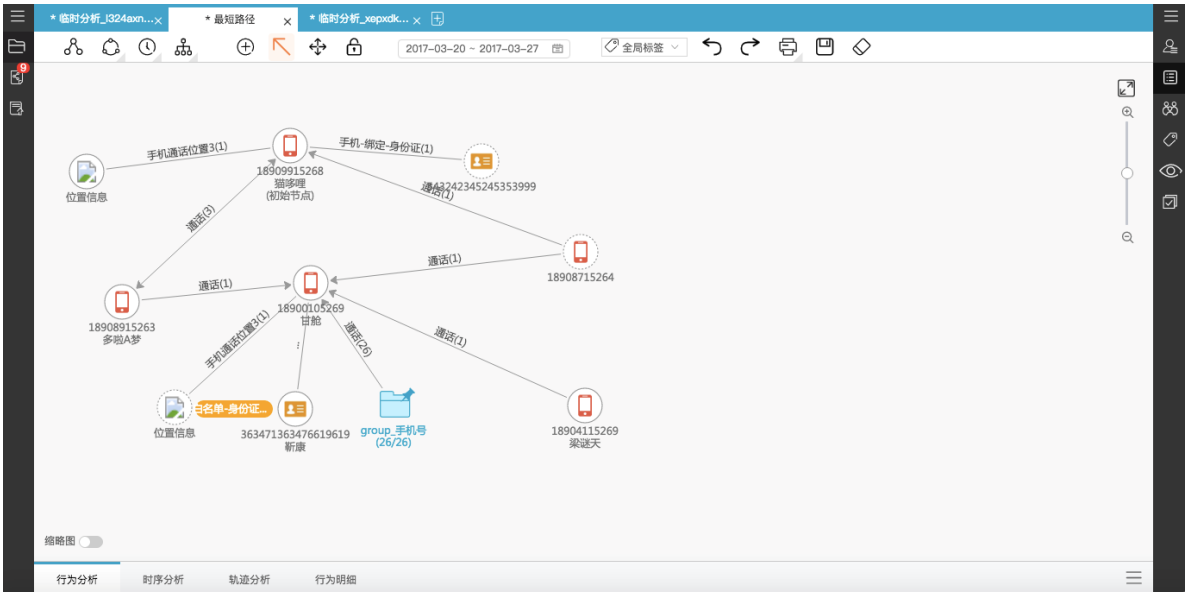
图 1399: 节点锁定效果2



4. 选中一个被锁定的节点，单击节点锁定功能键，可将该节点解锁。
5. 锁定合并节点。

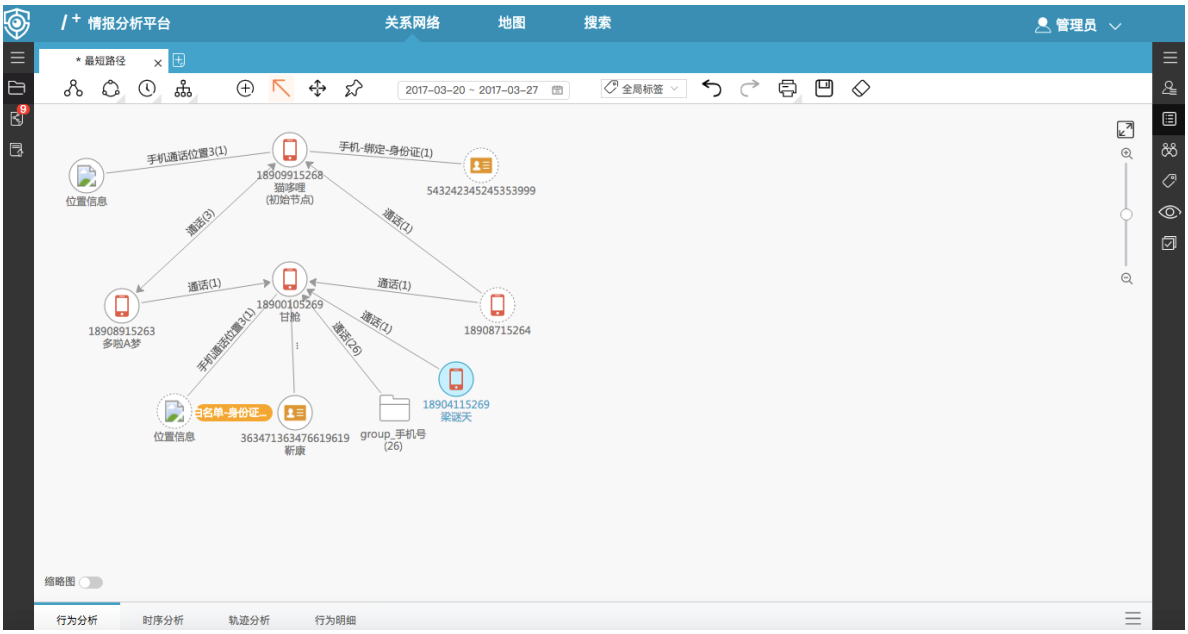
   如图 *1400:* 合并节点锁定所示，合并节点可以被锁定。

图 1400: 合并节点锁定



如果合并节点被拆分，即自动解锁，如图 *1401:* 拆分合并节点所示。

图 1401: 拆分合并节点



32.6.12.5 标签开关

关系网络分析图区支持对节点对象增加标签内容，同时也增加标签开关来控制标签内容的展示。

全局标签

显示所有标签内容。

在图区工具栏上，选择**全局标签**，如图 *1402: 标签开关*所示。

我的标签

仅显示该用户自己增加的标签。

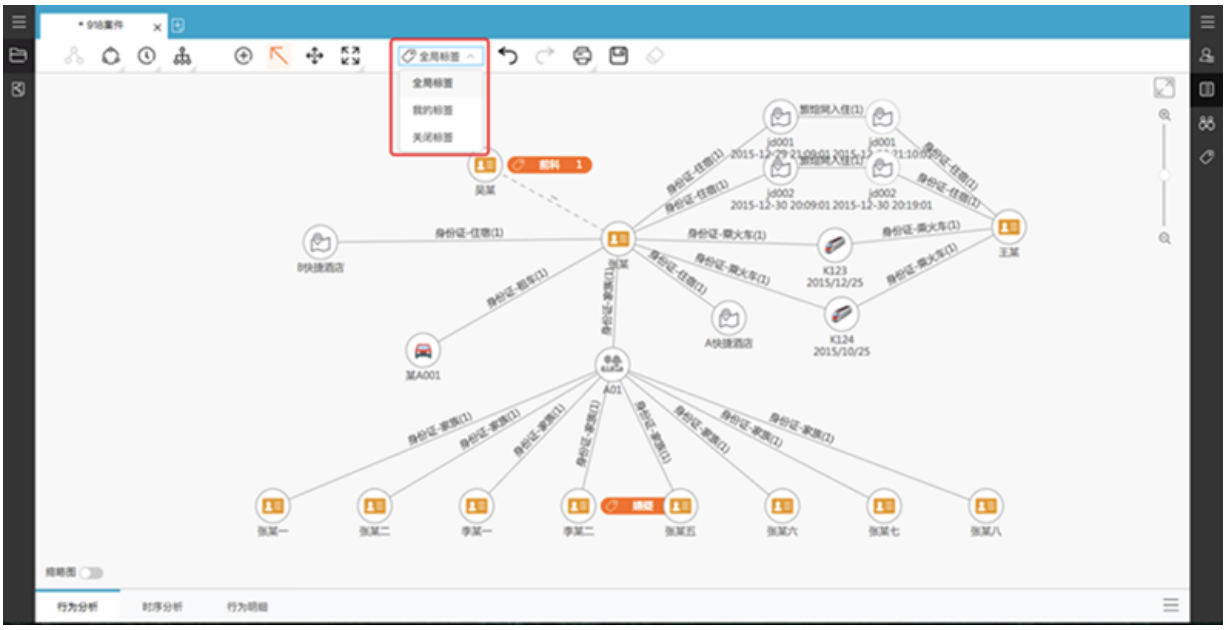
在图区工具栏上，选择**我的标签**，如图 *1402: 标签开关*所示。

关闭标签

将所有标签信息都不展示。

在图区工具栏上，选择**关闭标签**，如图 *1402: 标签开关*所示。

图 1402: 标签开关



32.6.12.6 撤销回退

撤销回退按钮，可对图区的对象进行撤销回退操作，支持回退20步。

撤销

在工具栏上单击撤销按钮，对图区节点进行撤销操作，如图 *1403: 撤销*所示。

图 1403: 撤销

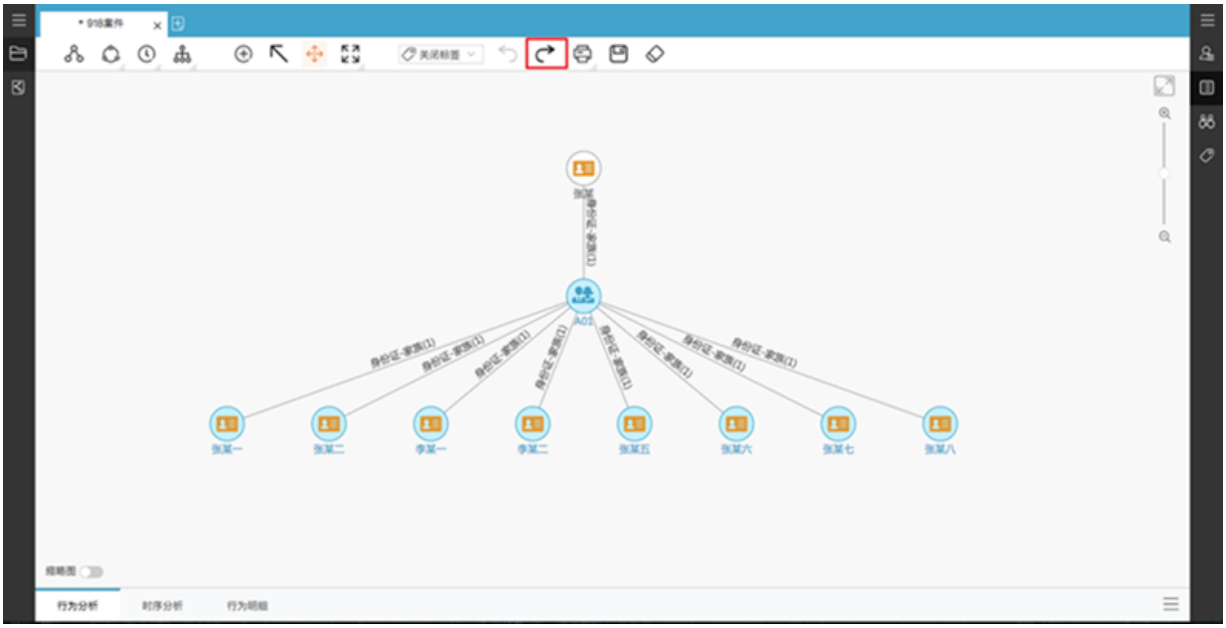


回退

撤销后，用户可通过回退功能，回退到撤销前的状态。

在工具栏上单击回退按钮，可回退到撤销前的状态，如图 *1404:* 回退所示。

图 1404: 回退



32.6.12.7 图区打印

关系网络分析图区支持打印，需要系统对接打印设备。

图区打印功能支持全图打印和可视区域打印。

全图打印

将分析图区所有内容打印出来。

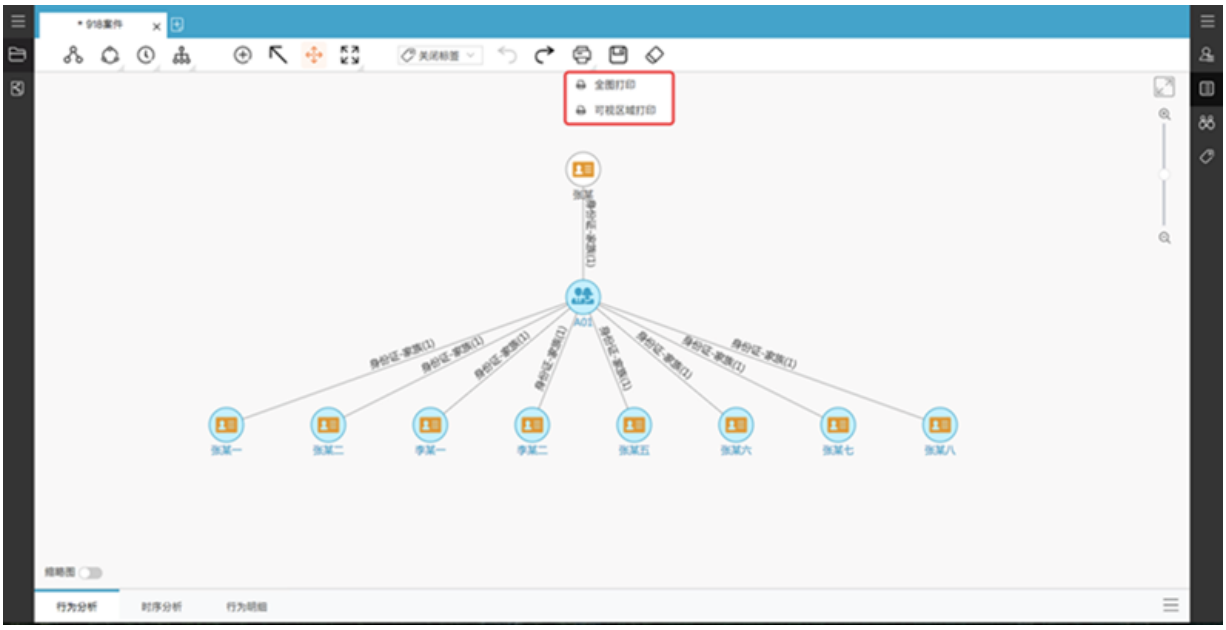
在工具栏上，选择**全图打印**，如图 *1405:* 图区打印所示。

可视区域打印

将关系网络可视部分打印。

在工具栏上，选择**可视区域打印**，如图 *1405:* 图区打印所示。

图 1405: 图区打印

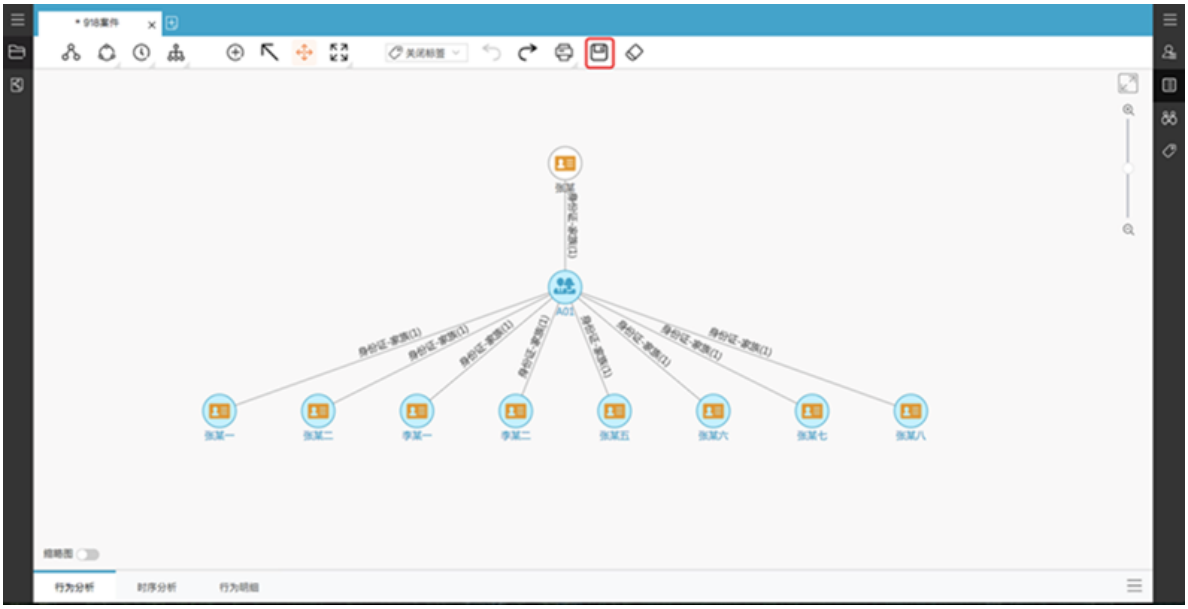


32.6.12.8 结果保存

操作步骤

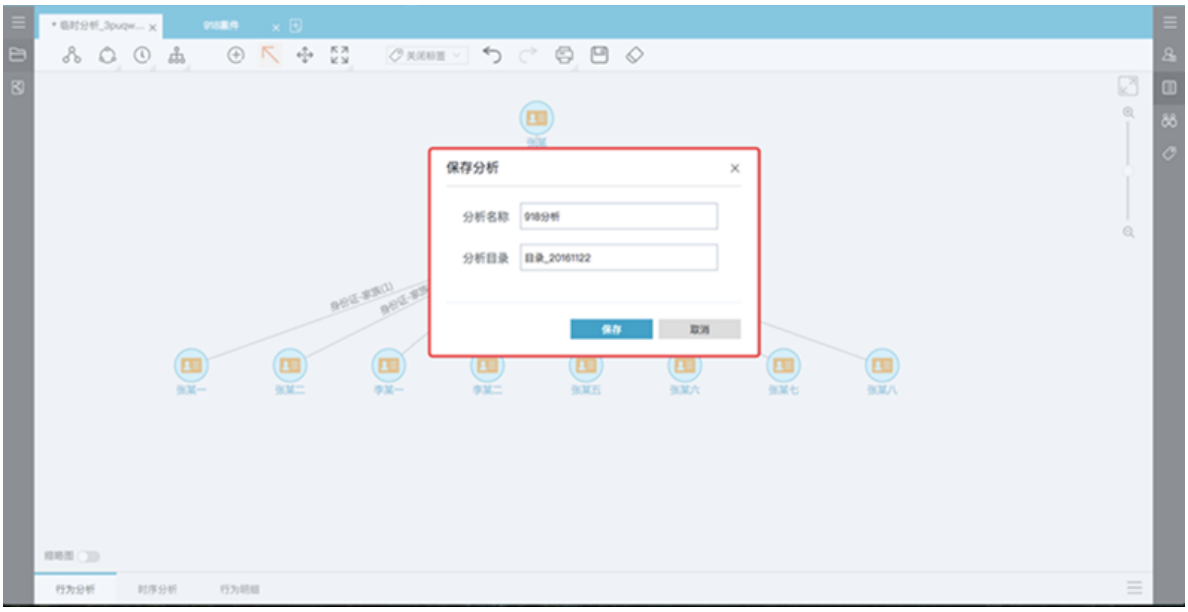
1. 点击工具栏的保存按钮，如图 *1406: 保存按钮*所示，保存图区分析内容。

图 1406: 保存按钮



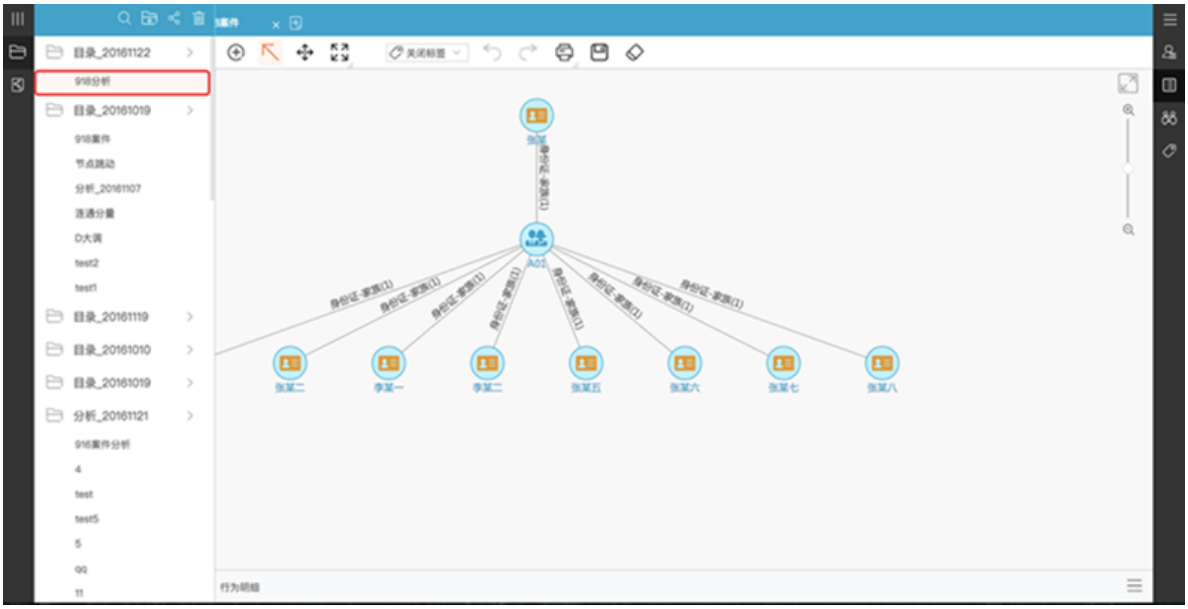
2. 如果分析文件为临时文件，则会弹出**保存分析**对话框，需输入需要保存的名字及目录，如图 *1407: 保存分析*所示；如果是打开已经保存的分析，则会将新的编辑内容同步保存。

图 1407: 保存分析



3. 保存后，单击左侧目录树，可查看保存的结构，如图 *1408:* 查看保存的结构所示。

图 1408: 查看保存的结构



32.6.12.9 删除所选

背景信息

删除所选可以将分析图区的选中内容清除，同时给予用户自主提示。

操作步骤

1. 选中对象后，单击清空画布的按钮。

选中对象可以选择全部选中或部分选中，如图 *1409: 选中全部对象*和图 *1410: 选中部分对象*所示。

图 1409: 选中全部对象

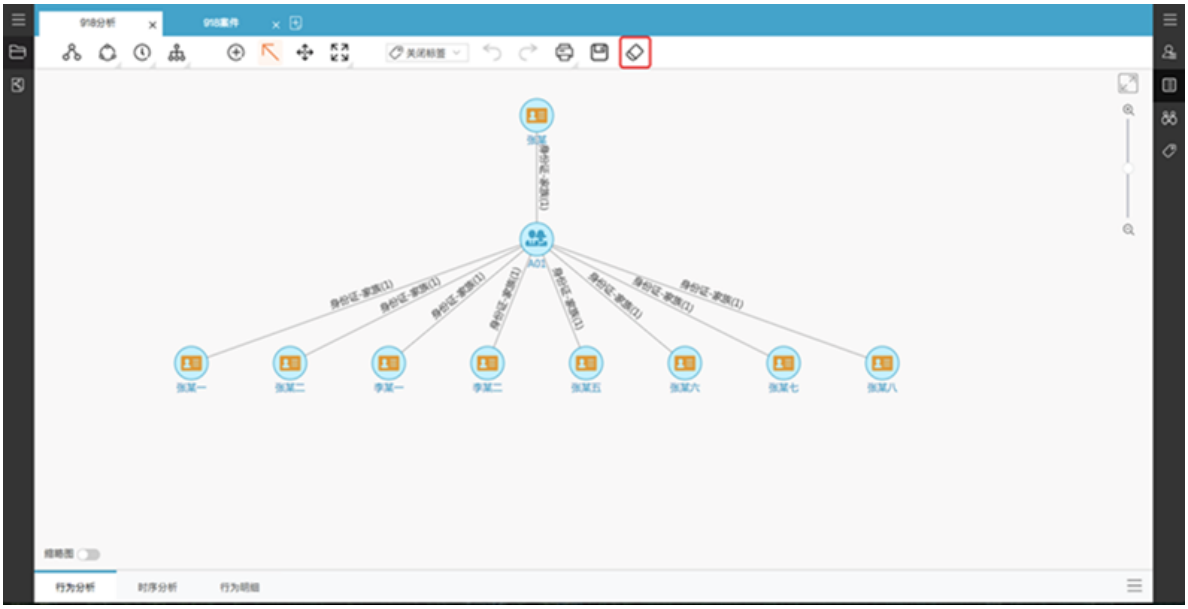
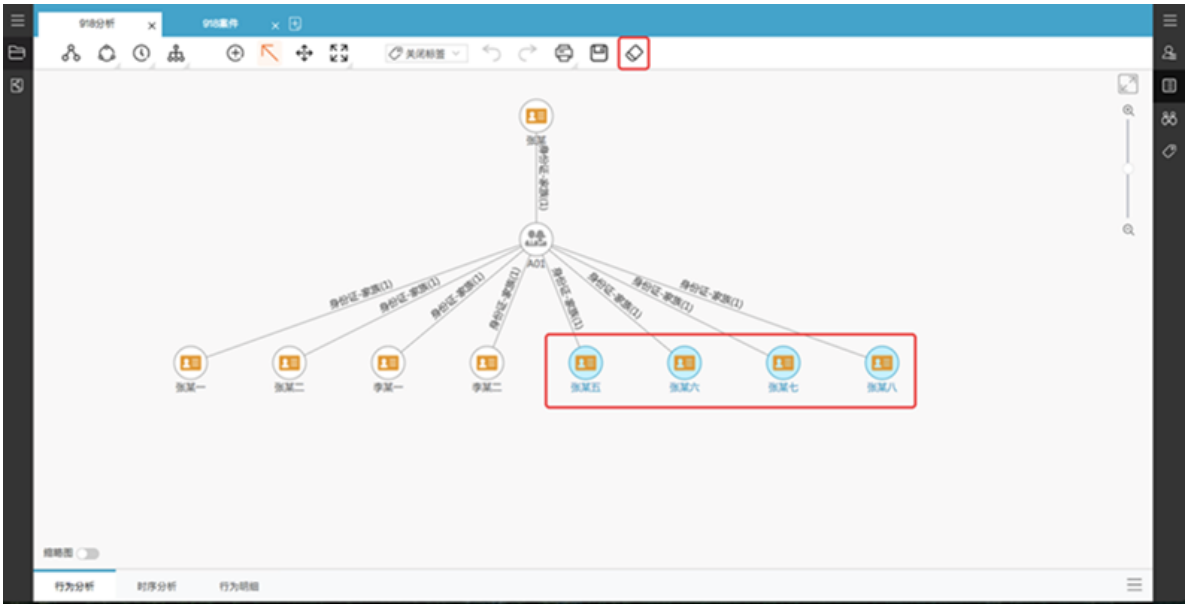
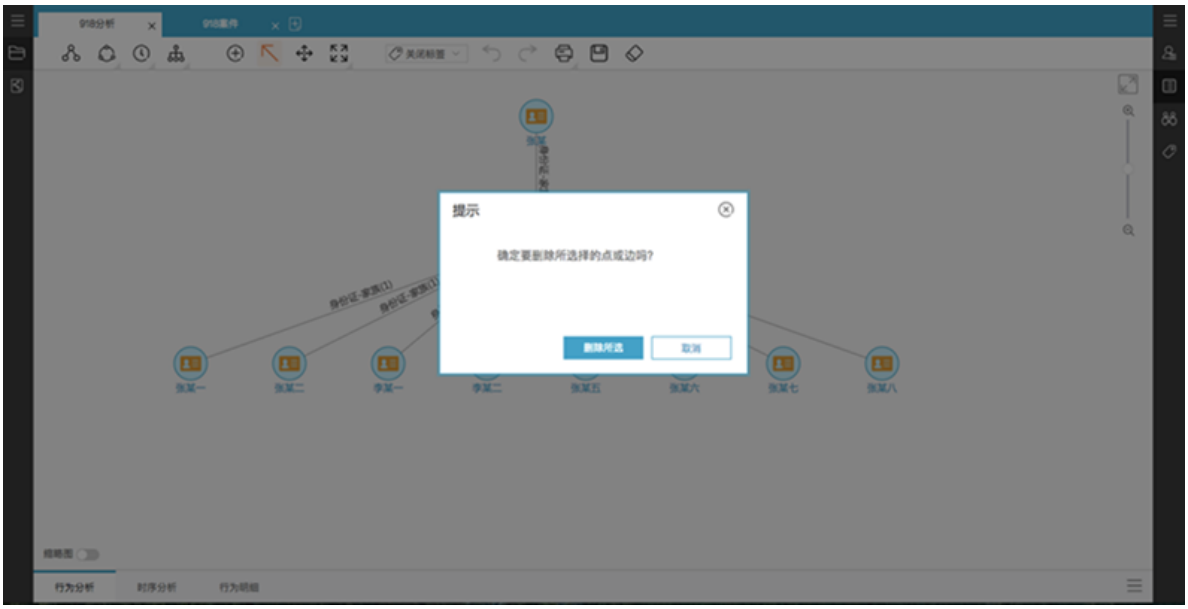


图 1410: 选中部分对象



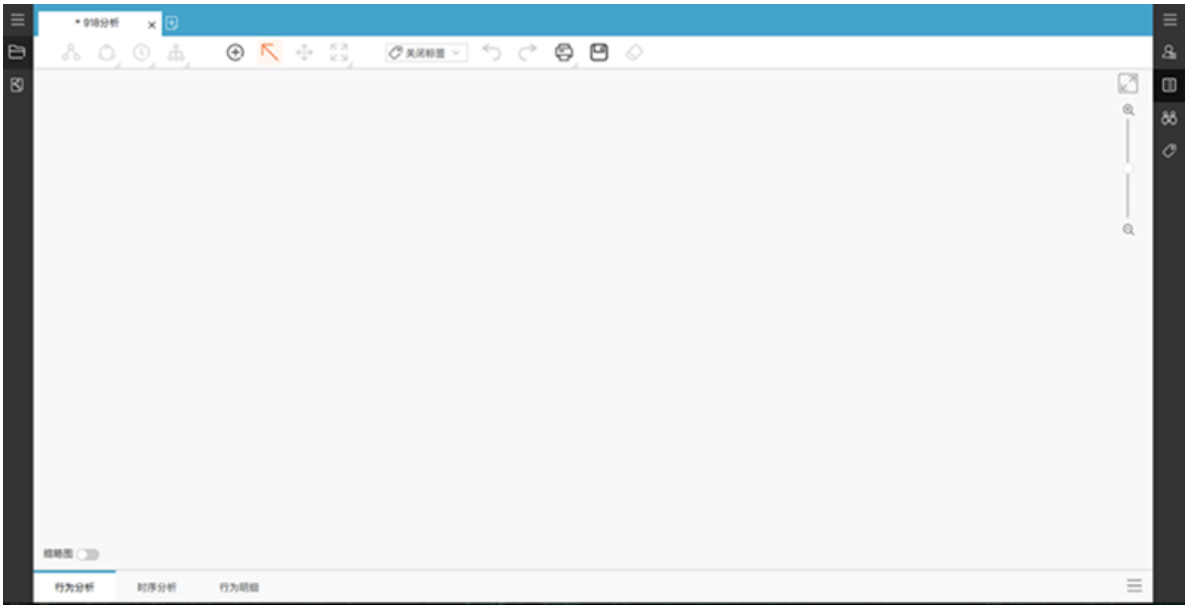
2. 在弹出的对话框中，单击**删除所选**，如图 *1411: 删除所选*所示。

图 1411: 删除所选



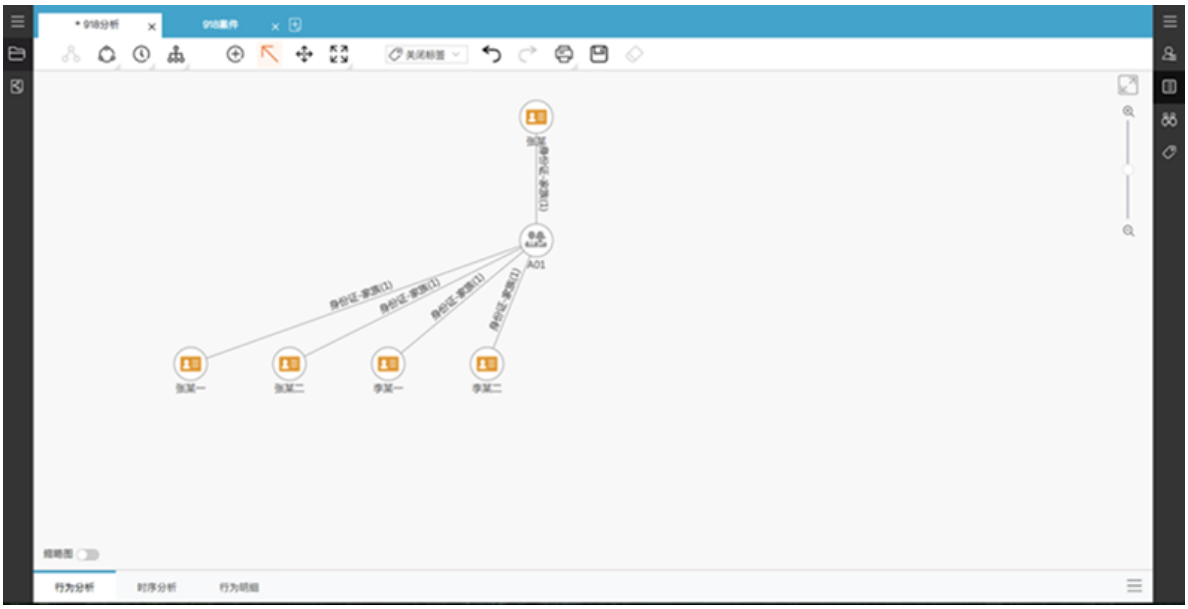
全部选中后删除的结果如图 *1412:* 全部选中后的结果所示。

图 1412: 全部选中后的结果



部分选中后删除的结果如图 *1413:* 部分选中后的结果所示。

图 1413: 部分选中后的结果

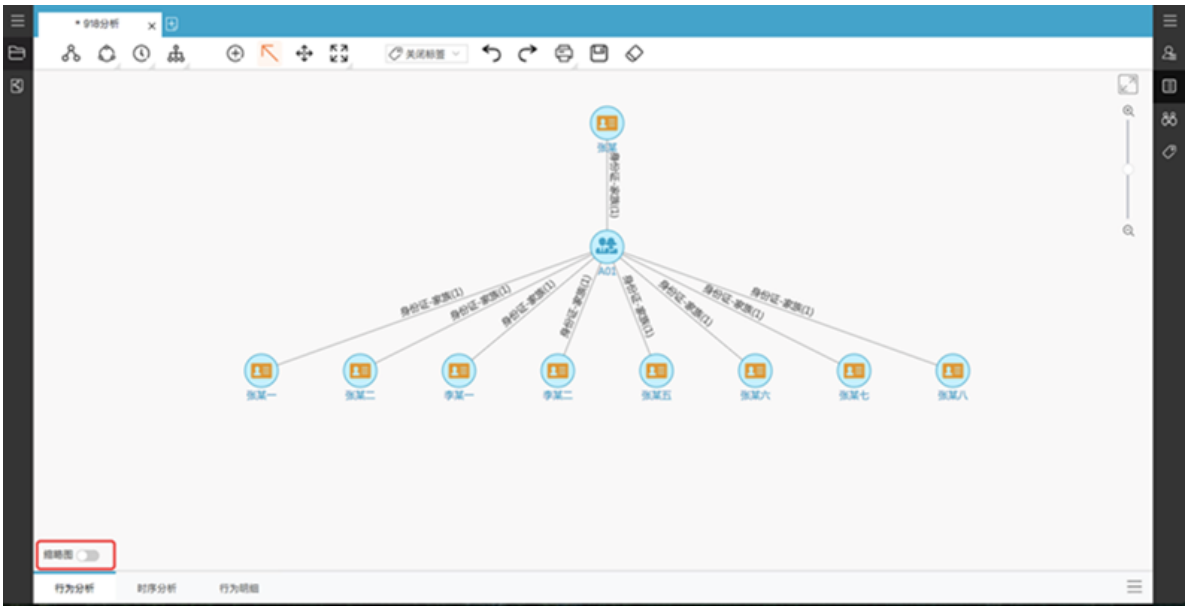


32.6.12.10 缩略图

操作步骤

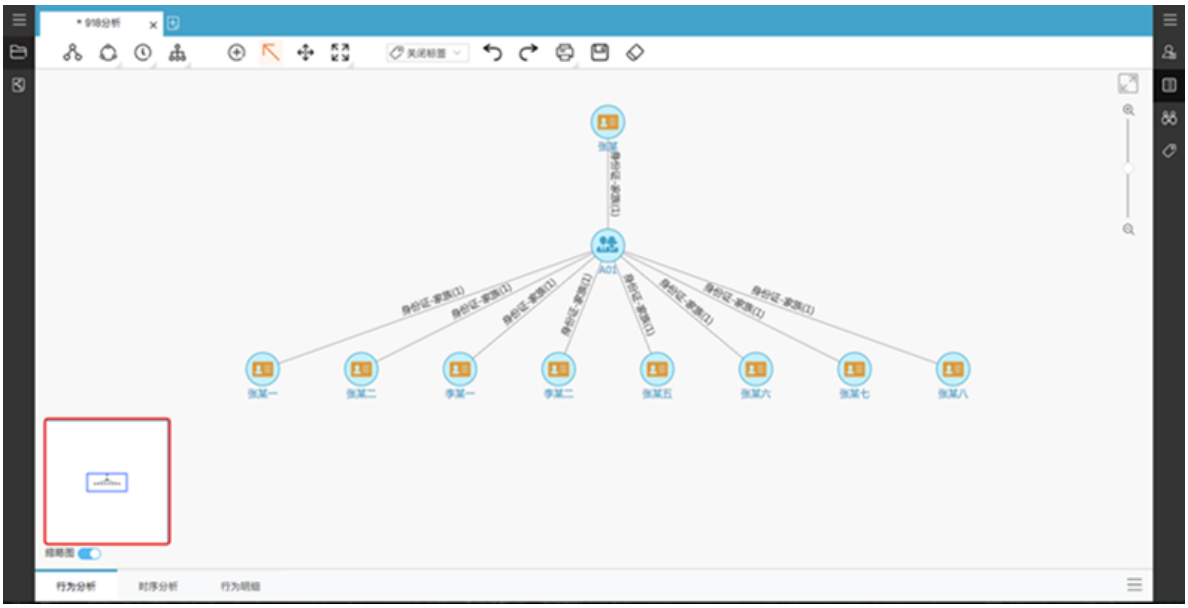
1. 单击左下角的缩略图开关，如图 *1414:* 缩略图开关所示。

图 1414: 缩略图开关



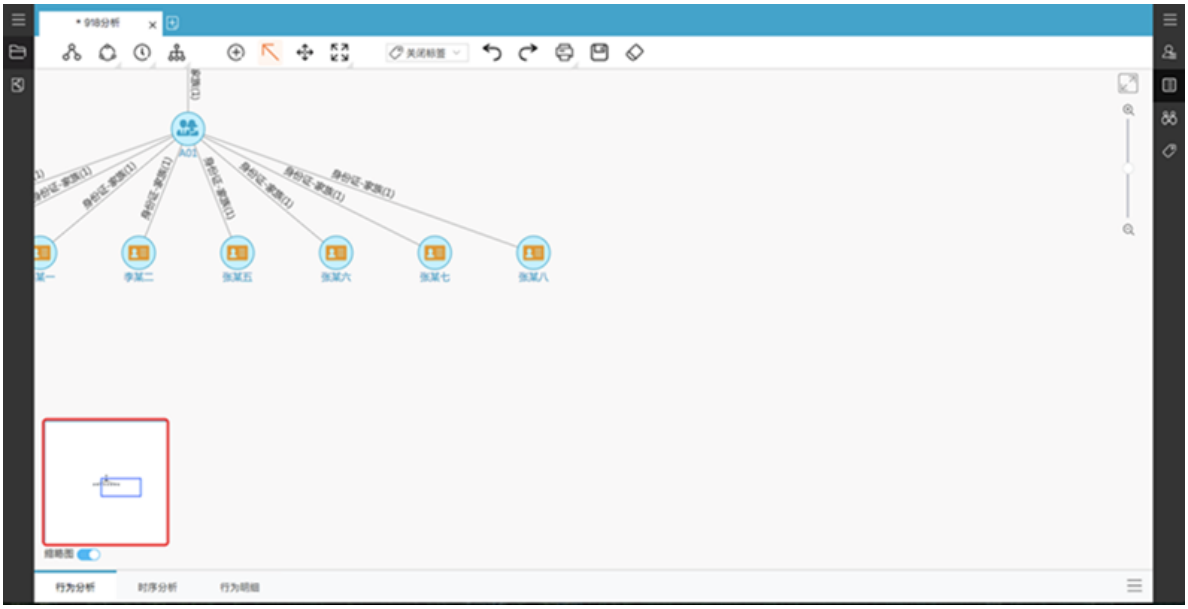
显示的缩略图如图 *1415:* 显示缩略图所示。

图 1415: 显示缩略图



2. 拖动缩略图中间的矩形框，可以动态拖动画布中矩形框对应的节点，如图 *1416:* 拖动缩略图所示。

图 1416: 拖动缩略图



32.6.13 右键操作

背景信息

关系网络分析图区中的信息包括对象、关系内容，映射的网络结构中的图。其中点和边是核心元素，所有的分析都是基于图对其中的点和边做操作。

右键操作是围绕着对关系网络编辑分析设计的功能点。

关系网络分析图区中的右键操作包含如图 *1417: 右键操作形态一*和图 *1418: 右键操作形态二*所示的操作。

图 1417: 右键操作形态一

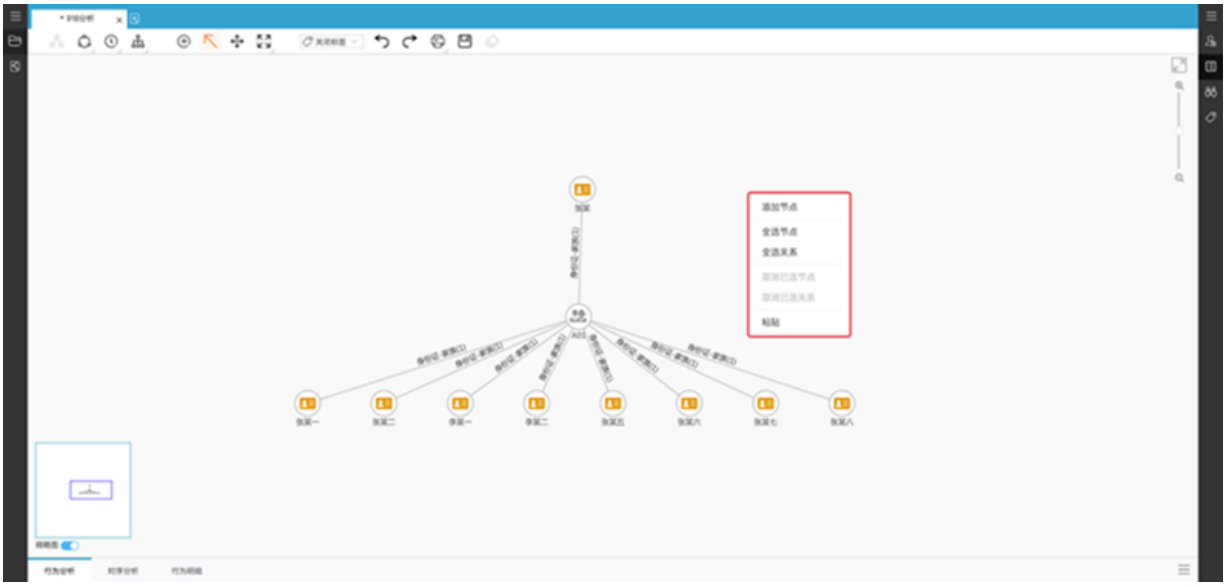
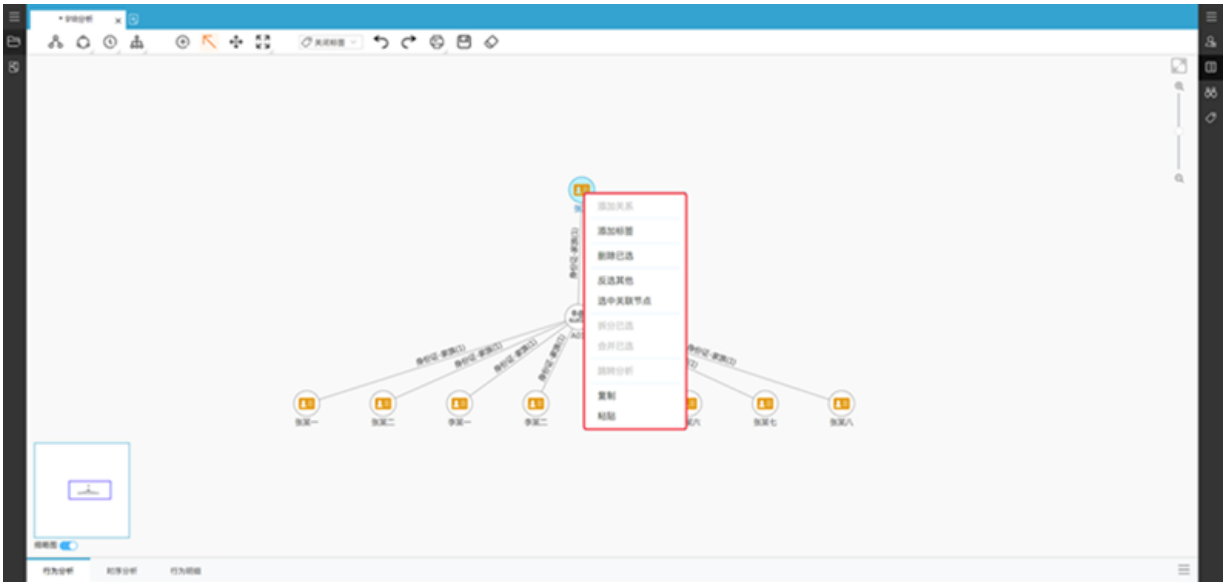


图 1418: 右键操作形态二

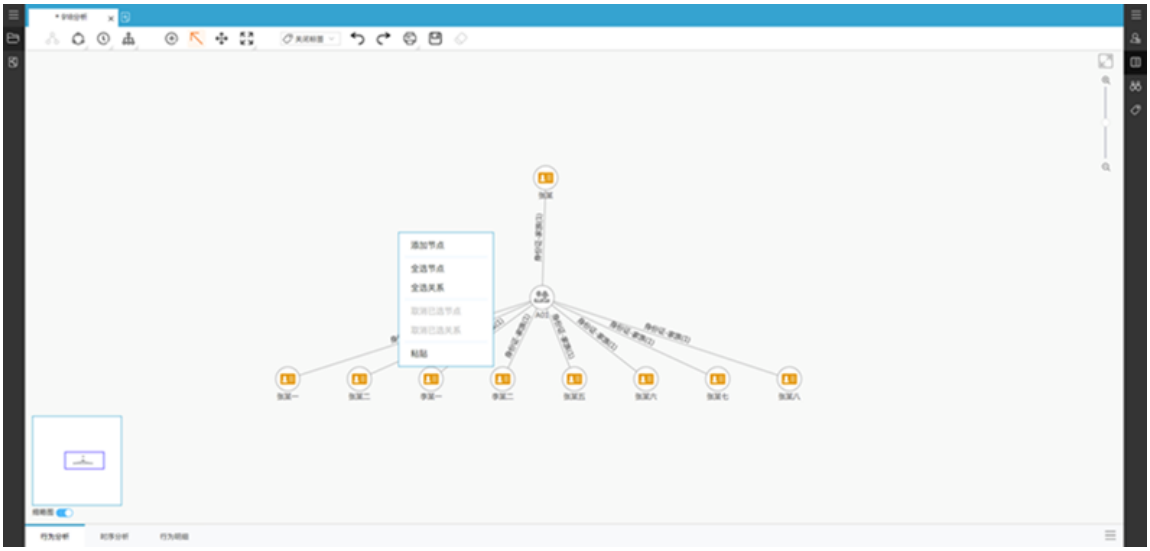


- 全选节点

  在分析图区将所有节点全部转化成选中状态。

- 当页面没有任何点、边处于选中状态时，在图上任何位置右键触发，如图 *1419: 右键操作1*所示。

图 1419: 右键操作1



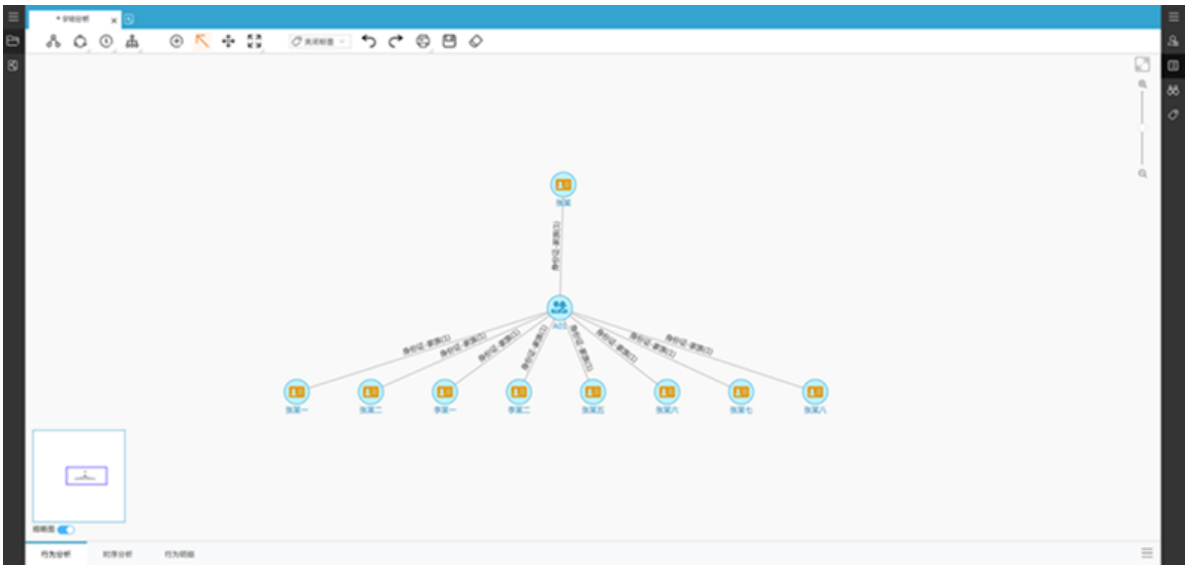
- 当图区点或边存在部分内容选中状态，在非选中态的点或边上右键触发，如图 *1420: 右键操作2*所示。

图 1420: 右键操作2



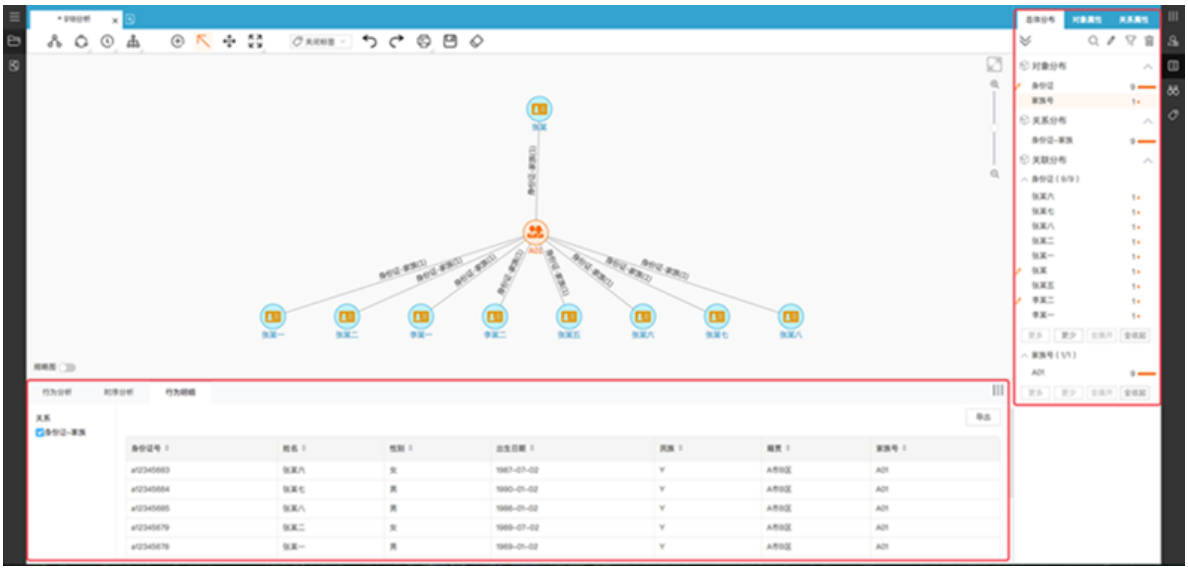
操作完成后的效果如图 *1421: 全选节点效果*所示。

图 1421: 全选节点效果



同时，如果将右侧面板和下侧时序面板展开，可看到数据同步变化，如图 *1422:* 展开相关面板所示。

图 1422: 展开相关面板



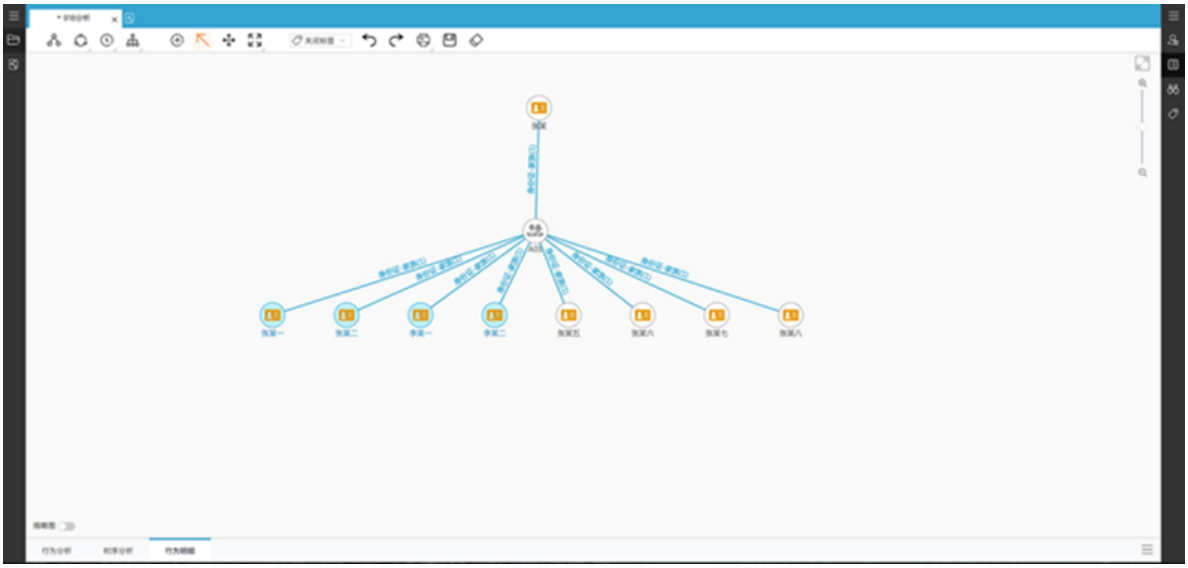
- 全选关系

  在分析图区将所有关系（边）全部转化成选中状态。

  - 当页面没有任何点、边处于选中状态时，在图上任何位置右键触发。
  - 当图区点或边存在部分内容选中状态，在非选中态的点或边上右键触发。

操作完成后的效果如图 *1423:* 全选关系效果所示。

图 1423: 全选关系效果



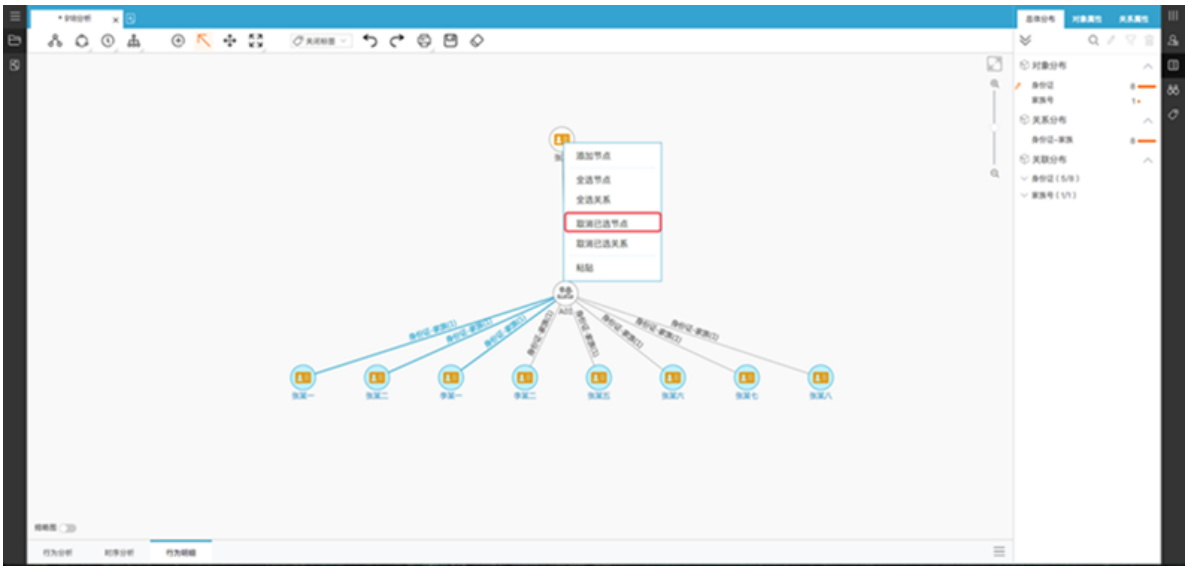
同时，如果将右侧面板和下侧时序面板展开，也可以看到数据同步变化。

- 取消已选节点

  在分析图区将选中状态的节点转变成为未选中状态。

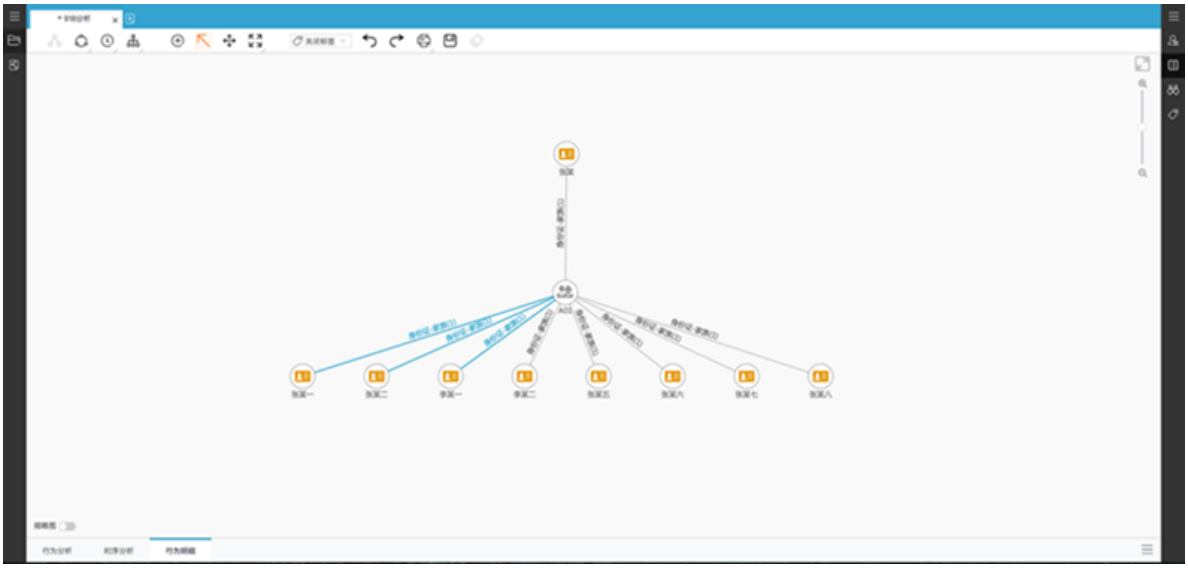
  当图区点或边存在部分内容选中状态，在非选中态的点或边上右键触发，如图 *1424:* 取消已选节点所示。

图 1424: 取消已选节点



操作完成后的效果如图 *1425:* 取消已选节点效果所示。

图 1425: 取消已选节点效果



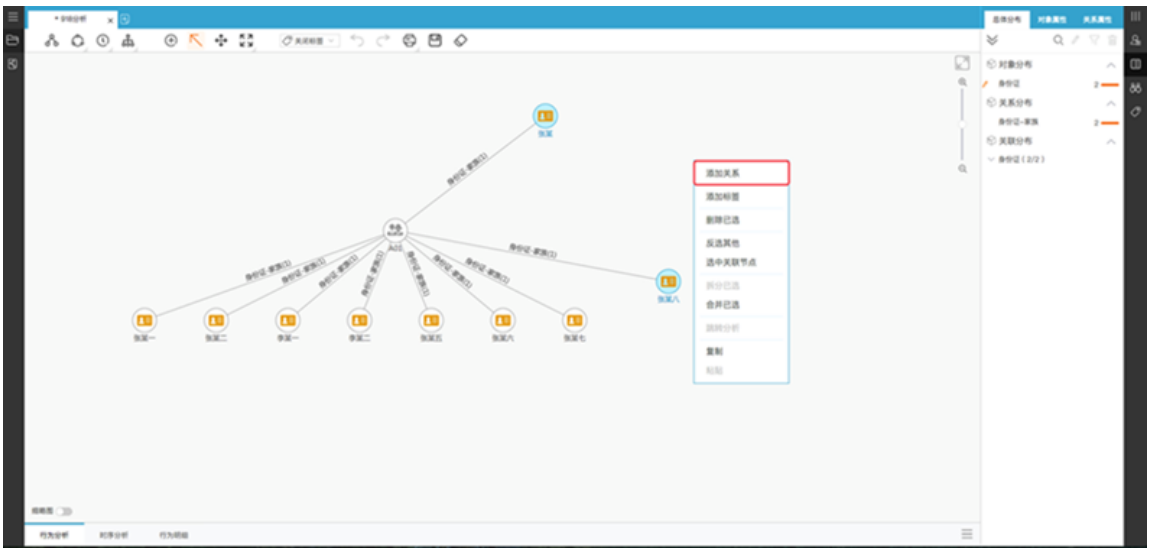
同时，如果将右侧面板和下侧时序面板展开，可看到数据同步变化。

- 新增关系

a) 选中两个节点，在图区空白处右键，选择**添加关系**，如图 *1426:* 新增关系所示。

说明: 选中节点数量必须等于2。

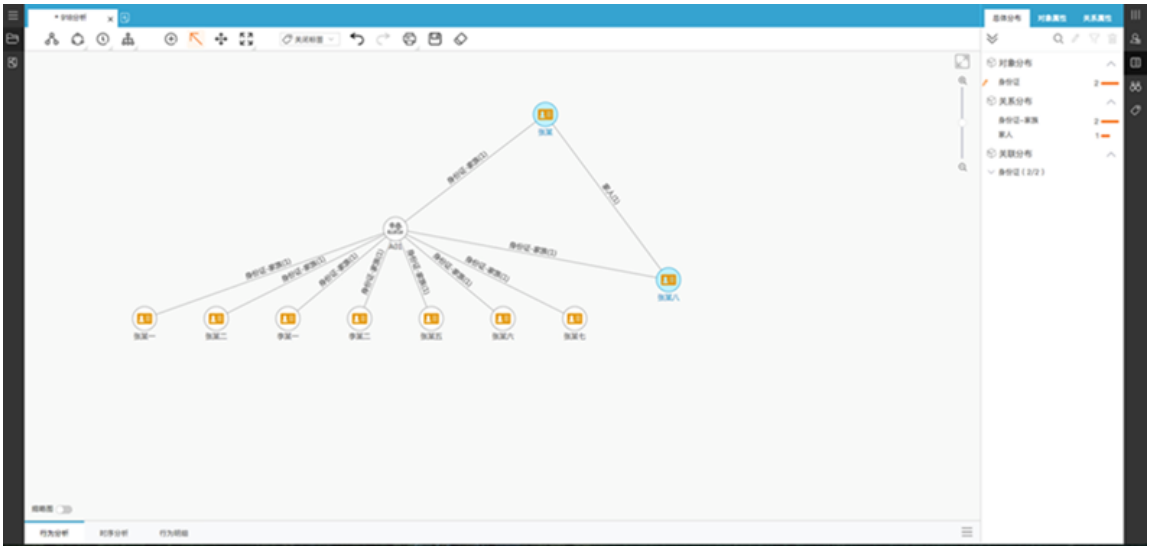
图 1426: 新增关系



b) 在弹出的对话框中，输入关系名称。

新增关系后的结果如图 *1427:* 新增关系结果所示。

图 1427: 新增关系结果



- 新增标签

用户在分析过程中可以对关注的对象节点增加特殊信息，通过**新增标签**为其添加标识信息，便于分析判别。同时也支持对标签信息管理。

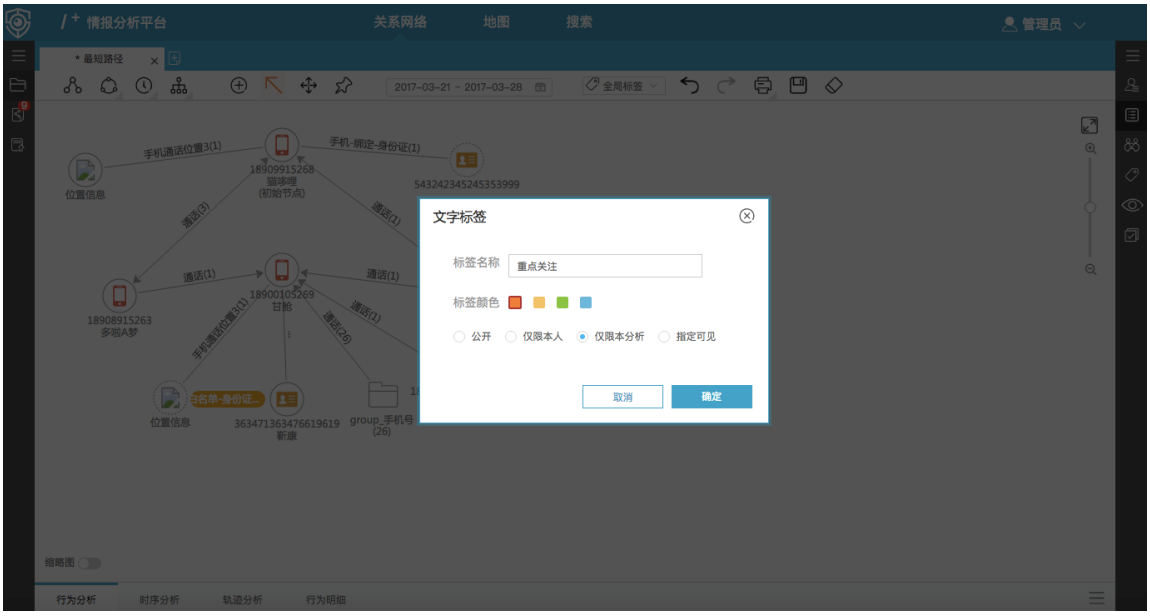
a) 选中节点，右键单击，选择**添加标签**，如图 *1428:* 添加标签所示。

图 1428: 添加标签



b) 输入标记值，标签值不超过20个字符，可选择不同颜色以表示不同的等级，如图 *1429: 输入标记值*所示。

图 1429: 输入标记值



关于权限控制，标签的权限分为以下四类：

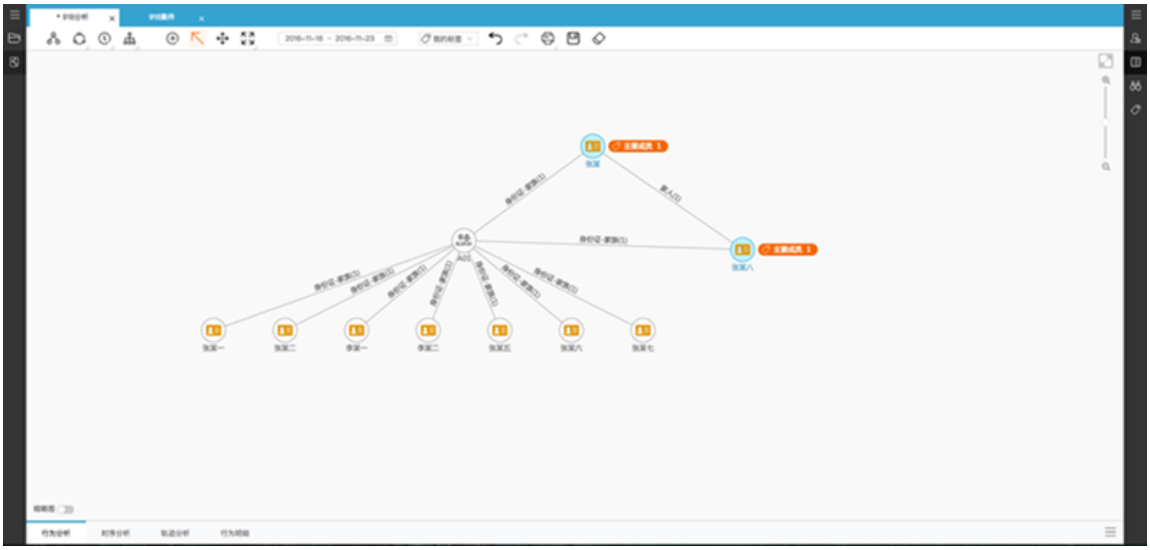
- 公开：即所有人可见。
- 仅本人可见：只有当前用户可以见。
- 仅本分析可见：该标签只有该分析中生效，其他分析任何人不可见，为默认选项。
- 指定可见：指定标签的共享范围，如图 *1430: 权限控制*所示。

图 1430: 权限控制



标记结果如图 *1431: 标记结果*所示。

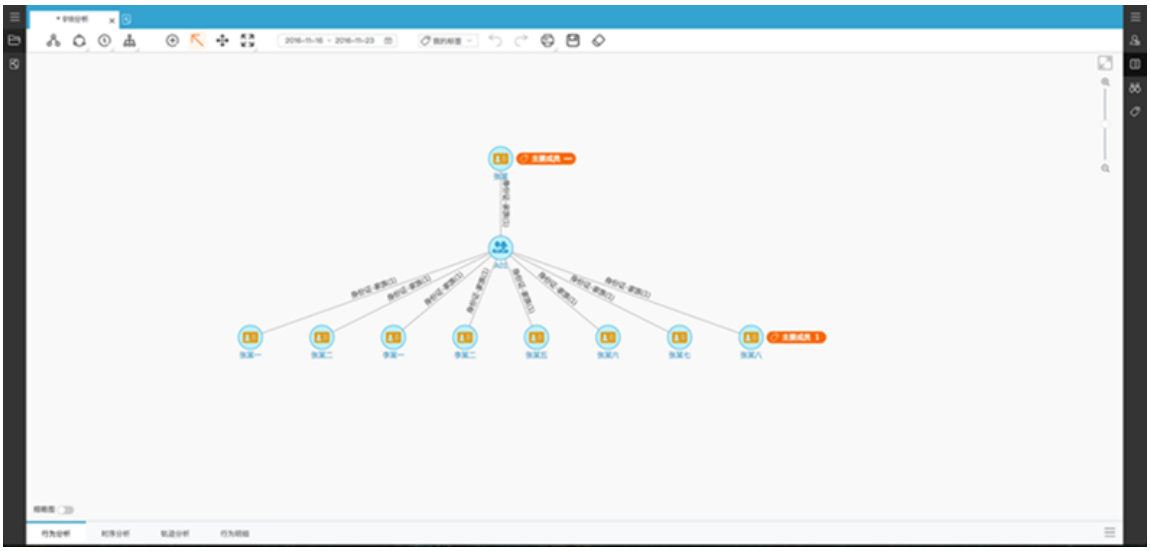
图 1431: 标记结果



c) 在图区每个标签上有数字标识，该标签被认同的人数，用户可以点 + 或者 - 来表示认可与不认可，如图 *1432:* 标签认可所示。

用户自己添加的标签则会默认先显示为**1**，鼠标浮动到数字上会变为 - ，单击 - 表示不认可，当数量减为0，则该标签被删除；对于系统中其他用户分享的标签，初始数字为**1**，其他用户鼠标浮动到数字上，会变为 + 或 - ：认可则 + ，同时数字 + 1，不认可 - 数字减1。当然 + 、 - 操作只能操作一次。

图 1432: 标签认可



说明: 合并节点不支持打标签。

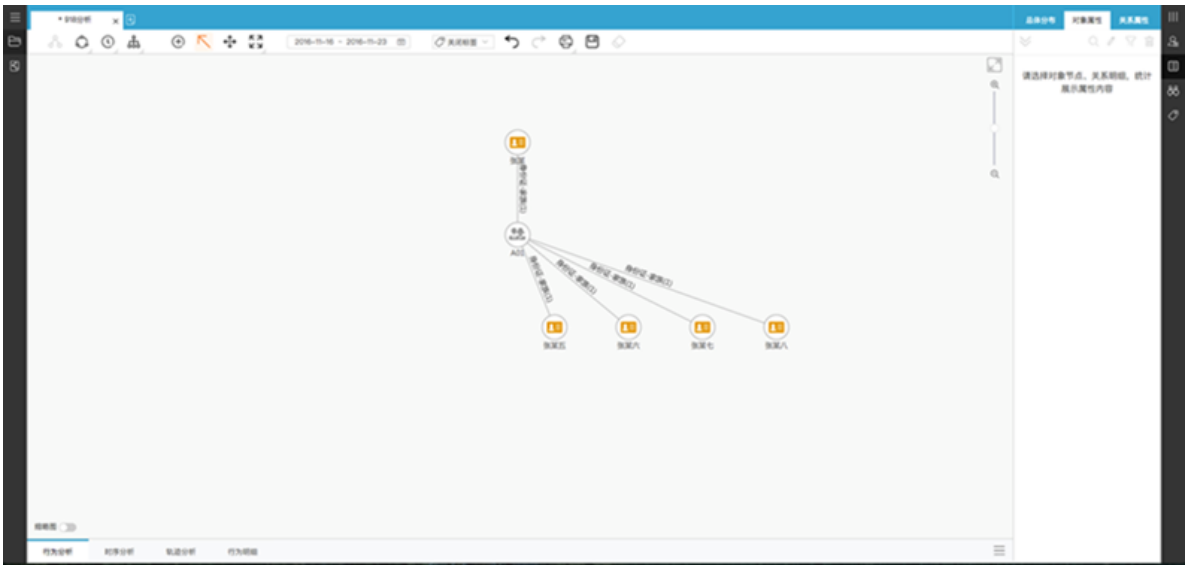
- 删除已选

主要帮助用户完成对于关系网络分析的删减，对于用户认定的可以删除信息做快捷清除操作。其中对于对象节点的删除同时会将其关联的关系也删除。

- 当图区点或边存在部分内容选中状态，在选中态的点或边上右键触发。
- 当图区点或边存在部分内容选中状态，在图区空白区域右键触发。

删除已选后，如果右侧面板和下侧时序面板展开，因为选中内容清楚，所有右侧面板和时序面板内容被清空，如图 *1433:* 删除已选结果所示。

图 1433: 删除已选结果



- 反选其他

  主要帮助用户完成对于关系网络分析的信息定位，帮助用户快速将页面上的选中状态反转，该功能主要围绕对象节点操作。

  - 当图区点或边存在部分内容选中状态，在选中态的点或边上右键触发。
  - 当图区点或边存在部分内容选中状态，在图区空白区域右键触发。

  操作完成后，如果右侧面板和下侧时序面板展开，可看到数据也同步变化。

- 选中关联节点

  主要帮助用户完成对于关系网络分析的信息定位，帮助用户快速对选中点的关联点选中，该功能主要围绕对象节点操作。

  - 当图区点或边存在部分内容选中状态，在选中态的点或边上右键触发。
  - 当图区点或边存在部分内容选中状态，在图区空白区域右键触发。

  操作完成后，如果右侧面板和下侧时序面板展开，可看到数据也同步变化。

- 拆分已选

  方便用户将置于合并节点内部的信息分离出来单独分析，该功能只对对象节点操作，且必须确保选中节点中有节点置于合并节点内部，同时，**拆分已选**不改变分析图区对象节点和关系（边）的选中状态。

  - 当图区点或边存在部分内容选中状态，在选中态的点或边上右键触发。
  - 当图区点或边存在部分内容选中状态，在图区空白区域右键触发。

拆分会提供条件过滤，如图 *1434:* 条件过滤所示。

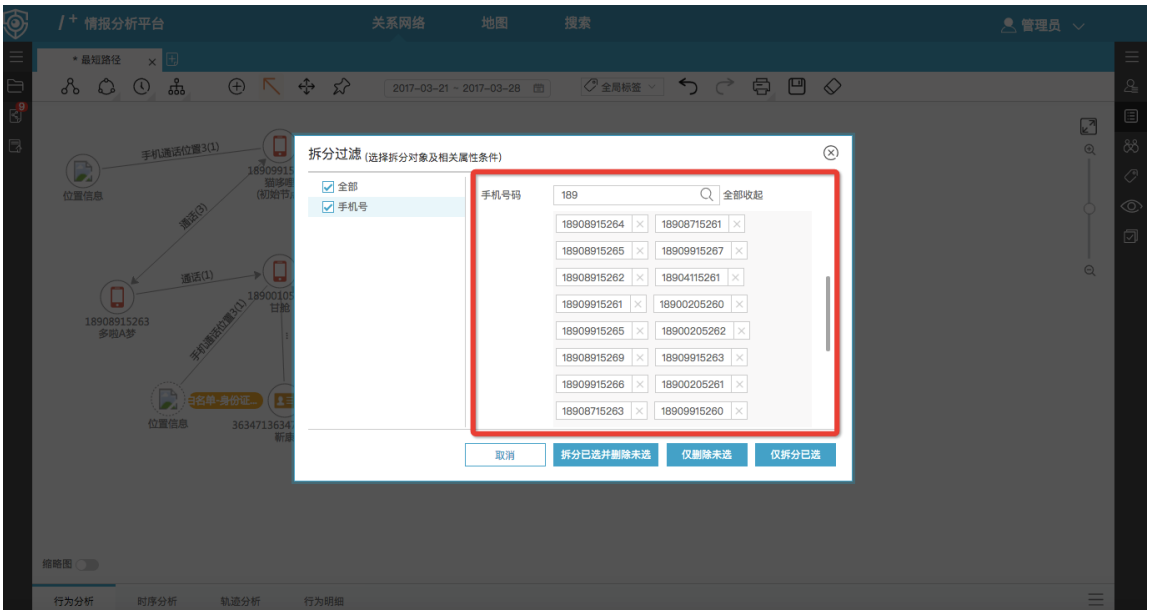
图 1434: 条件过滤



会将合并节点中的标签信息，属性信息规整，按照类型分类：

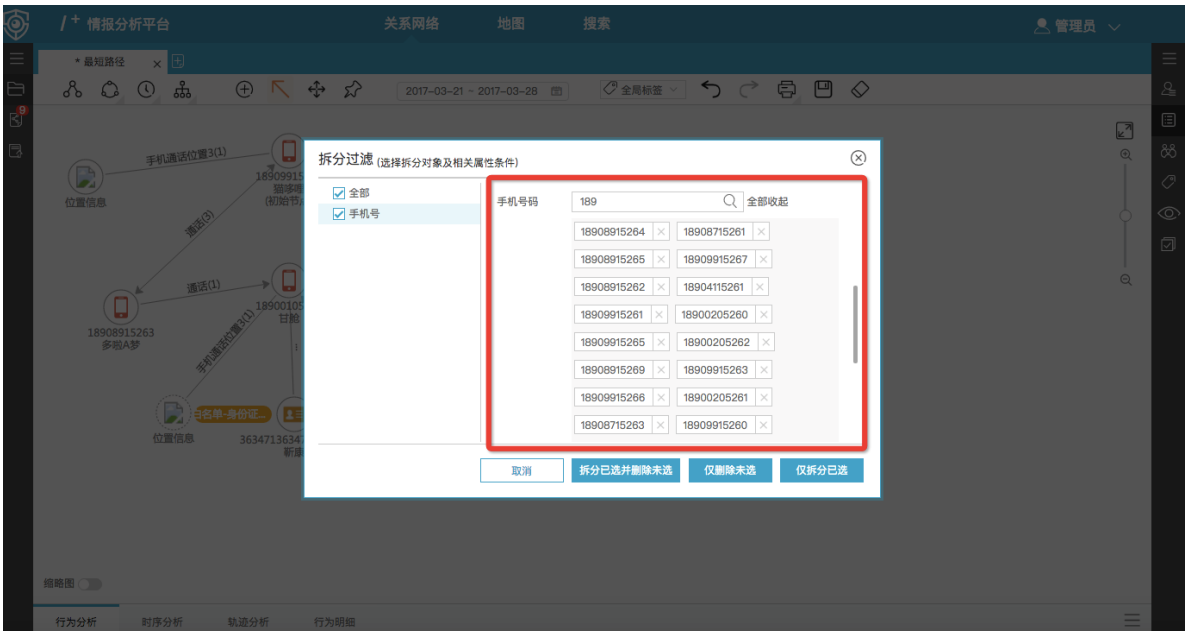
- 标签：全部枚举，用户可以删除部分。
- 时间类型：列出时间属性的最大值和最小值，用户可以调整范围，如发车时间。
- 数值型：列出数值的最大值和最小值，用户可以调整范围，如年龄。
- 字典型：同标签一样，全部枚举，用户可以删除部分。
- 字符串：用户搜索，支持模糊搜索，如图 *1435:* 模糊查询所示。

图 1435: 模糊查询



搜索含有189的手机号码，如图 *1436: 搜索手机号码*所示，搜索结果每一项后面可以单击叉号删除。当搜索不到结果时，系统会提醒**搜索结果为空**。

图 1436: 搜索手机号码



- 取消：取消拆分。
- 拆分已选并删除未选：将满足的条件的节点拆分出来，其它的删除。
- 仅删除未选：将满足条件的节点保留在合并节点中，其它的删除。
- 仅拆分已选：将满足条件的节点拆分出来，其它的保留在合并节点中。

- 合并已选

  方便用户将置于整个分析图区信息梳理，将部分信息合并，简化图区视觉内容，该功能只对对象节点操作，且必须确保选中节点数量等于或超过，同时，**合并已选**不改变分析图区对象节点和关系（边）的选中状态。

  - 当图区点或边存在部分内容选中状态，在选中态的点或边上右键触发。
  - 当图区点或边存在部分内容选中状态，在图区空白区域右键触发。

- 跳转分析

  提供给I + 一个跟其他产品对接的渠道。比如在I+分析过程中对于某个特殊的对象节点，需要在其他系统中查看相关信息，则**跳转分析**就提供该渠道。该模块需要跟具体场景对接。

- 复制粘贴

  关系网络分析图区同时提供**复制**和**粘贴**操作功能，方便用户快速便捷的将选定的信息同步复制粘贴到新分析中，方便细化分析。**复制粘贴**功能支持两种形态操作：对象节点复制和关系图复制。

  - 对象节点复制

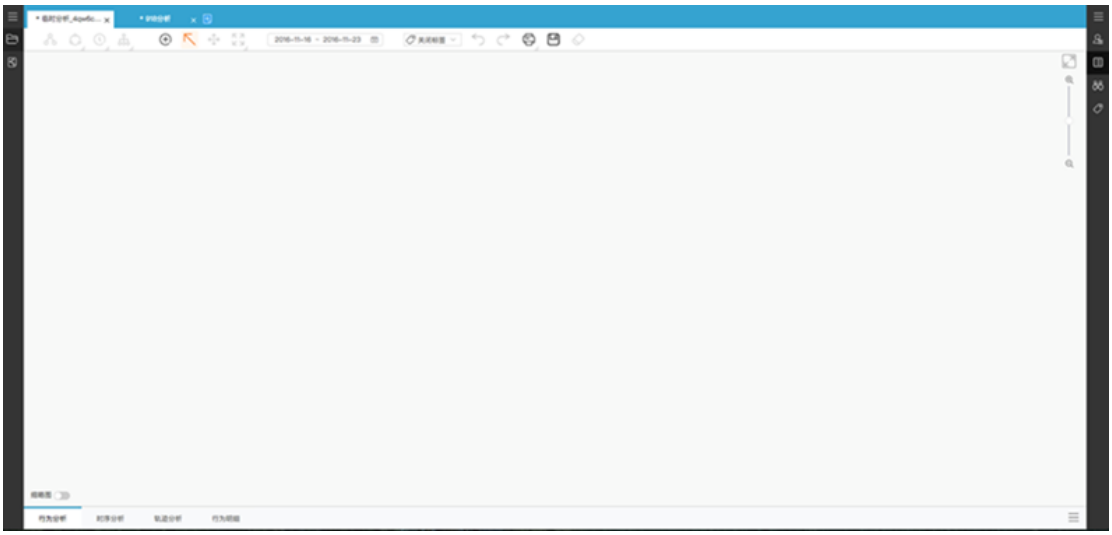
    1. 选中需要复制的点，选择**复制**如图 *1437: 复制节点*所示。

       图 1437: 复制节点

    2. 打开新的分析tab，如图 *1438: 新的tab*所示。

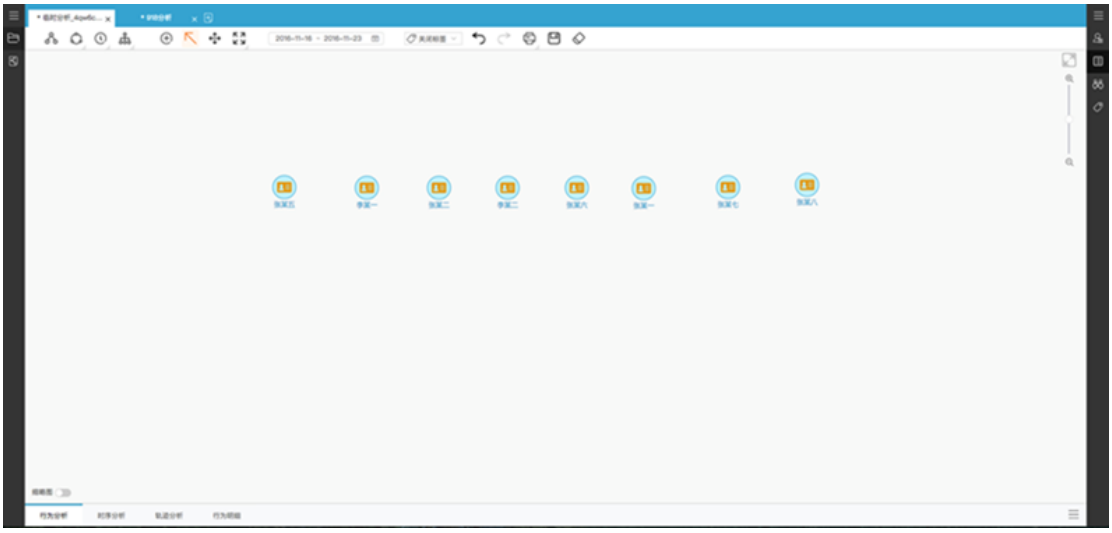
       图 1438: 新的tab



3. 在空白处单击右键，选择**粘贴**。

   粘贴效果如图 *1439: 粘贴节点*所示。

   图 1439: 粘贴节点



- 关系图复制

  1. 选中需要复制的一个子图，可单选或框选。
  2. 在图区空白处（非选中节点上）单击右键，在弹出的菜单中选择**复制**。
  3. 打开新的分析tab。
  4. 在空白处单击右键，选择**粘贴**。

32.6.14 目录管理

目录管理可以对目录和分析管理操作。

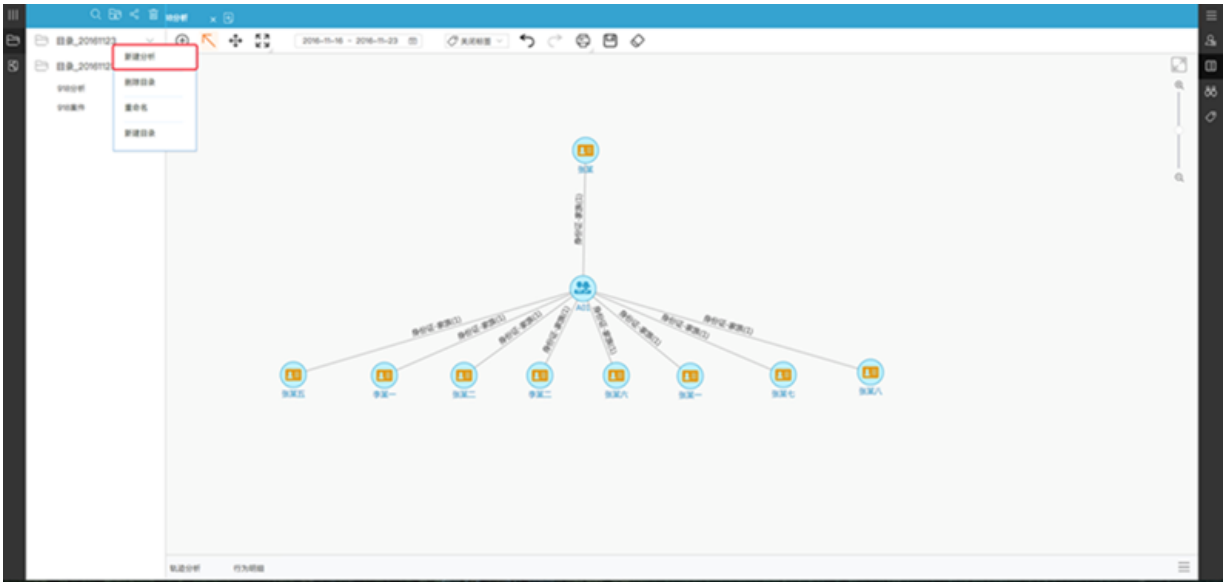
对于目录提供**新建分析**、**删除目录**、**重命名**、**新建目录**等操作；对于分析提供**打开**、**删除**、**重命名**的功能操作。

同时目录管理提供**搜索**对于内容检索定位和**分享**功能将个人的分析内容分享给他人延续分析。

新建分析

在目录上单击右键，选择**新建分析**，如图 *1440: 新建分析*所示。

图 1440: 新建分析



新建后默认分析命名为**分析 + 时间日期**，用户可以更改分析名称。

删除目录

1. 在目录上单击右键，选择**删除目录**。
2. 在弹出的提示框中，单击**确定**。

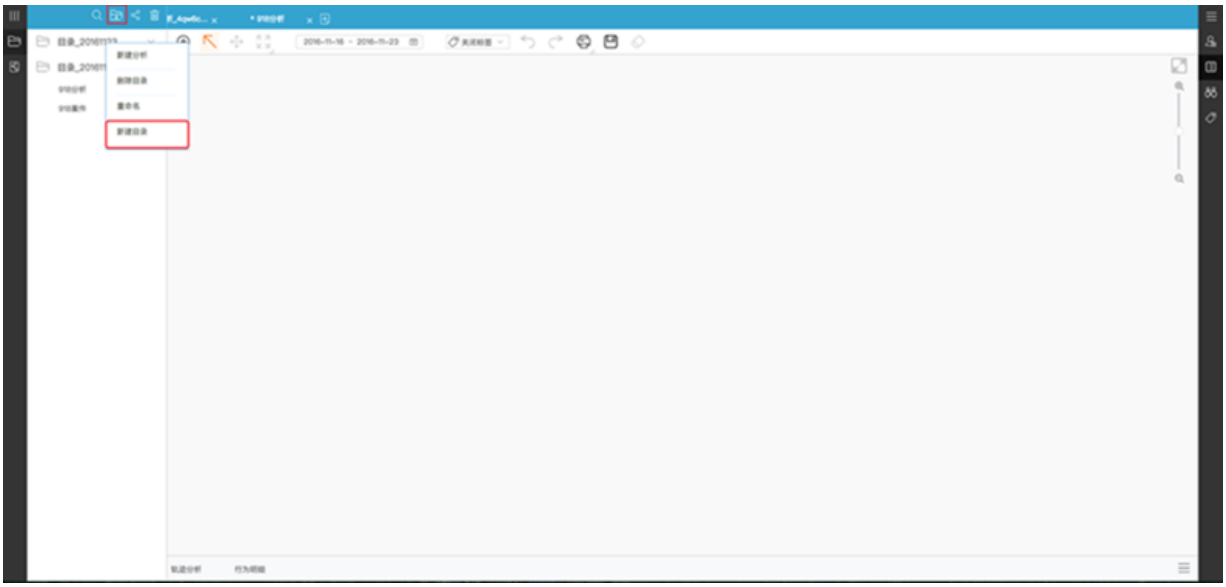
重命名目录

在目录上单击右键，选择**重命名**，修改目录名称。

新建目录

在目录上单击右键，选择**新建目录**或单击新建目录图标，如图 *1441: 新建目录*所示。

图 1441: 新建目录

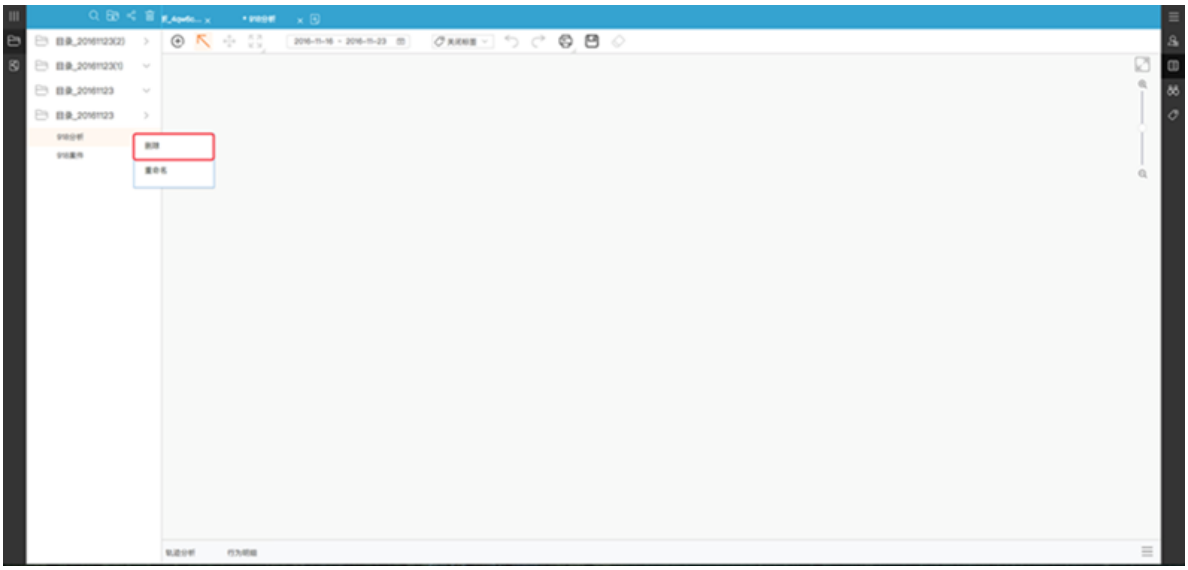


系统采用**目录 + 日期时间**作为默认名字，用户可以重新编辑名字。

分析删除

1. 在分析文件单击右键，选择**删除**，如图 *1442: 分析删除*所示。

图 1442: 分析删除



2. 弹出的提示框中，单击**确定**。

分析重命名

在分析文件单击右键，选择**重命名**，用户可以根据需要，修改分析名称。

32.6.15 协作共享

背景信息

协作共享是产品提供的一种新分析模式，可以将个人的分析共享给其他人，将个人的思路和经验传递给其他分析用户，同时可以将其他分析用户的智慧和经验统一起来，形成多人协作，团体共同推进的局面，能够将团队融合为一个整体。

协作共享的角色分为发起者、协同者。整体流程为：发起者对个人的分析内容分享，分享过程中会设定分享范围，指定分享对象。协同者在接收到分析内容后，可以延续分析，同时将结果保存，形成新的分析版本。

同时产品支持对协同分析的管理，包括删除、重命名、历史版本等管理。

说明: 删除和重命名操作都限定为发起者操作。

32.6.15.1 分享分发

操作步骤

1. 在**分析空间**，选择需要分析的文件。
2. 单击左侧面板上的分享按钮，如图 *1443: 分享按钮*所示。

图 1443: 分享按钮



3. 在新弹出的选择交互框中，选择分享的成员，如图 *1444:* 共享成员设置所示。

图 1444: 共享成员设置



共享成员的选择可以按照成员个人来选，支持搜索定位；也可以按照部门来选择。

右侧的**共享成员**列表中会显示被选中分享的成员，鼠标移到名字后面会显示删除图标，可以将当前成员删除，如图 *1445:* 删除共享成员所示。

图 1445: 删除共享成员



4. 成员选择完成后，单击**共享**按钮。如果共享成功，系统会提示**分析共享成功**，如图 *1446: 共享成功*所示。

图 1446: 共享成功



当被分享成员登录时，系统会提示该成员有分享任务。

针对每个分享者的分享会建立一个和分享分析同名的分享目录，类似文件夹。

该文件夹下会默认有两个文件：

- 分享的初始文件
- 自动合并文件

32.6.15.2 初始文件

对于分享给用户的初始文件，用户可以对其进行编辑操作。

打开分享的初始文件，如图 *1447: 初始文件*所示。

图 1447: 初始文件



初始文件中的对象节点正上方有一个红点，以特殊标示。被拓展出的新的对象节点则没有这样的标示。

32.6.15.3 保存草稿

对于分享给用户的文件，用户对其进行编辑后，可以保存为草稿。草稿文件仅对当前用户可见。

操作步骤

1. 打开待修改的文件，对其进行编辑。
2. 单击文件页面上方的功能标签页上的纸飞机按钮，弹出菜单项如图 *1448:* 发布版本和保存草稿所示。

图 1448: 发布版本和保存草稿



3. 单击**保存草稿**按钮，弹出保存草稿对话框，如图 *1449: 保存草稿*所示。

图 1449: 保存草稿



4. 输入草稿备注，字数不超过15个字。
5. 单击**保存**按钮。

   草稿保存后，会在分享分析目录下显示该草稿文件，草稿文件仅对当前用户可见，如图 *1450: 草稿文件*所示。

图 1450: 草稿文件



- 草稿文件显示位置：位于**初始文件**上方。
- 草稿文件命名格式：当前用户的用户名+草稿+(编号)，如test123草稿(1)。
- 草稿文件备注：草稿文件名下方显示文件备注。
- 草稿文件生成时间：当鼠标移至分享分析目录下该草稿文件位置时，可以显示出文件生成的时间。

32.6.15.4 发布版本

对于分享给用户的文件，用户对其进行编辑后，可以发布为新版本的文件。发布的文件，对当前共享文件范围内的人都可见，但是只有分享人和新版本文件发布人，可以对其进行删除操作。

操作步骤

1. 打开待修改的文件，对其进行编辑。
2. 单击文件页面上方的功能标签页上的纸飞机按钮，弹出菜单项如图 *1451:* 发布版本和保存草稿所示。

图 1451: 发布版本和保存草稿



3. 单击**发布版本**按钮，弹出发布共享文件对话框，如图 *1452:* 发布版本所示。

图 1452: 发布版本



4. 输入发布备注，字数不超过15个字。
5. 单击**确定**按钮。

文件发布成功后，会在分享分析目录下显示该发布文件，如图 *1453:* 发布成功所示。

图 1453: 发布成功



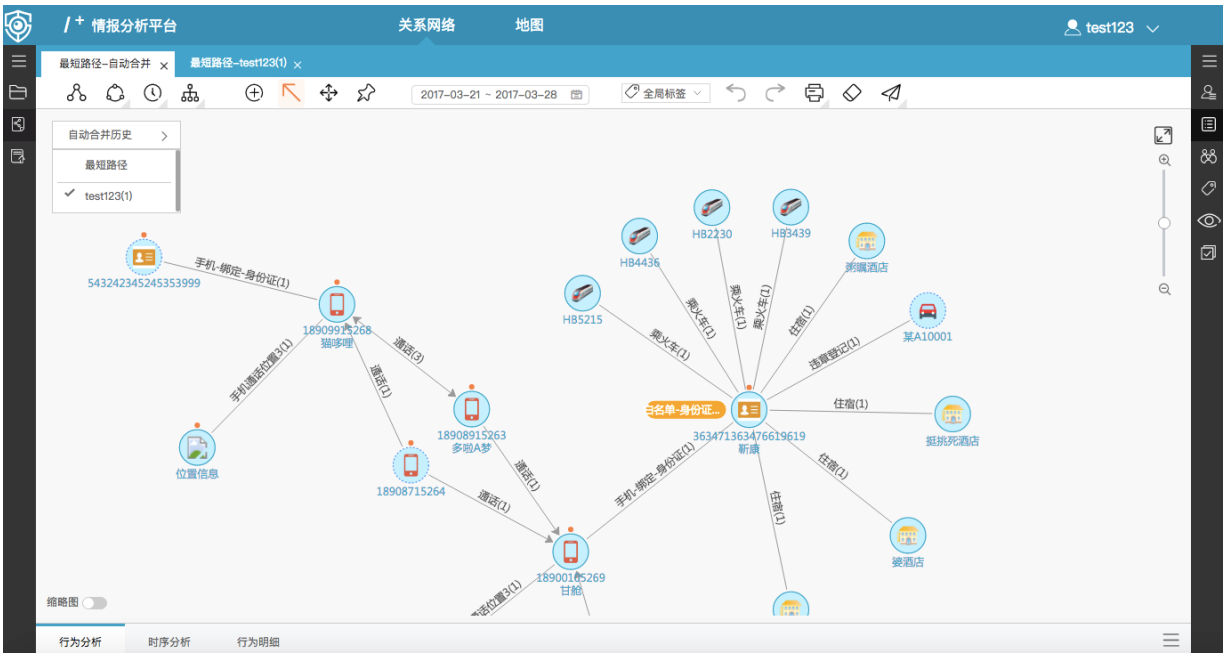
- 发布文件显示位置：位于**自动合并**下方。
- 发布文件命名格式：当前用户的用户名+（版本号），如test123草稿(1)。
- 发布文件备注：发布文件名下方显示文件备注。
- 发布文件生成时间：当鼠标移至分享分析目录下该发布文件位置时，可以显示出文件生成的时间。

32.6.15.5 自动合并

当有新的版本发布的时候，不论是哪个分享成员，包括分享者自己，自动合并会将发布的版本和原来的自动合并文件（最开始即为初始文件）自动合并。

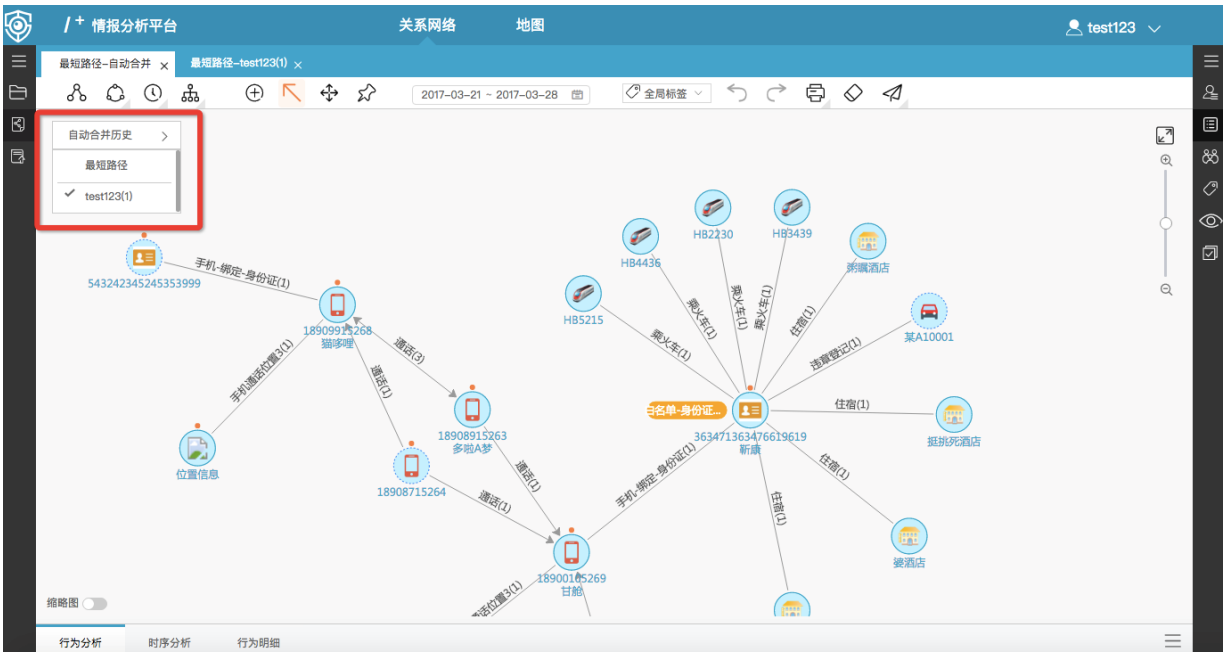
发布了一个新版本后，自动合并文件会有更新，如图 *1454: 文件更新*所示。

图 1454: 文件更新



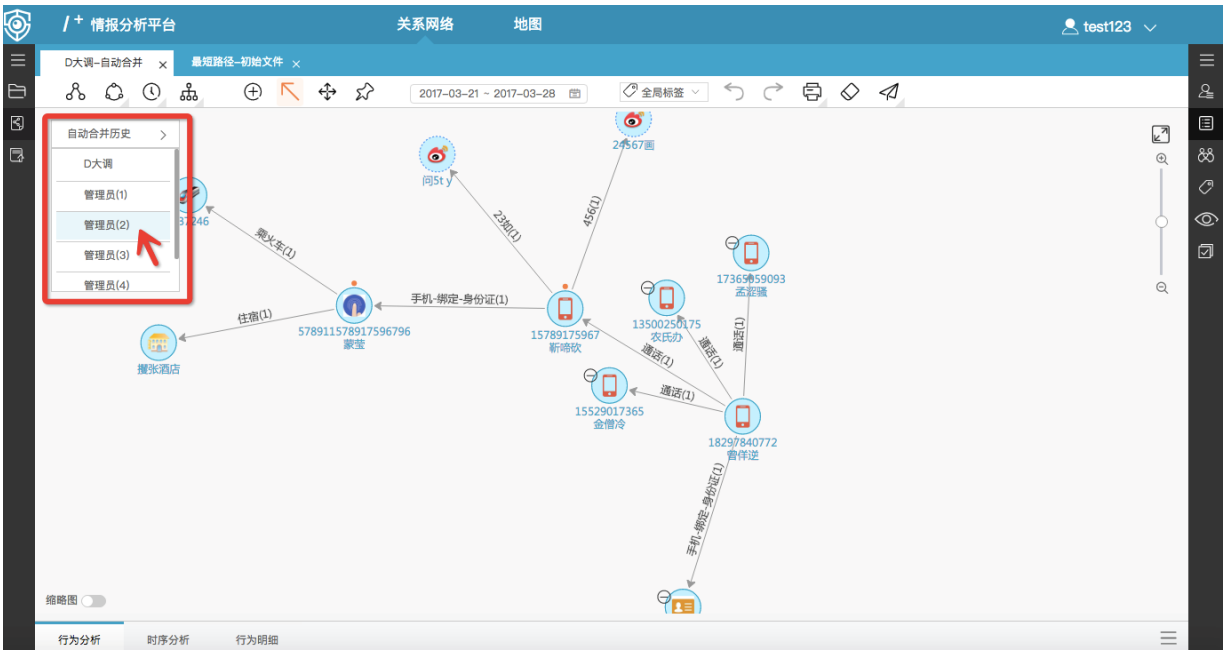
同时，在左上角，可以看到不同版本的列表，默认展示最新的版本，如图 *1455: 版本列表*所示。

图 1455: 版本列表



您可以选择回溯。通过选择**自动合并历史**中的不同版本，查看以前的版本，如图 *1456: 版本回溯*所示。

图 1456: 版本回溯



32.6.15.6 手动合并

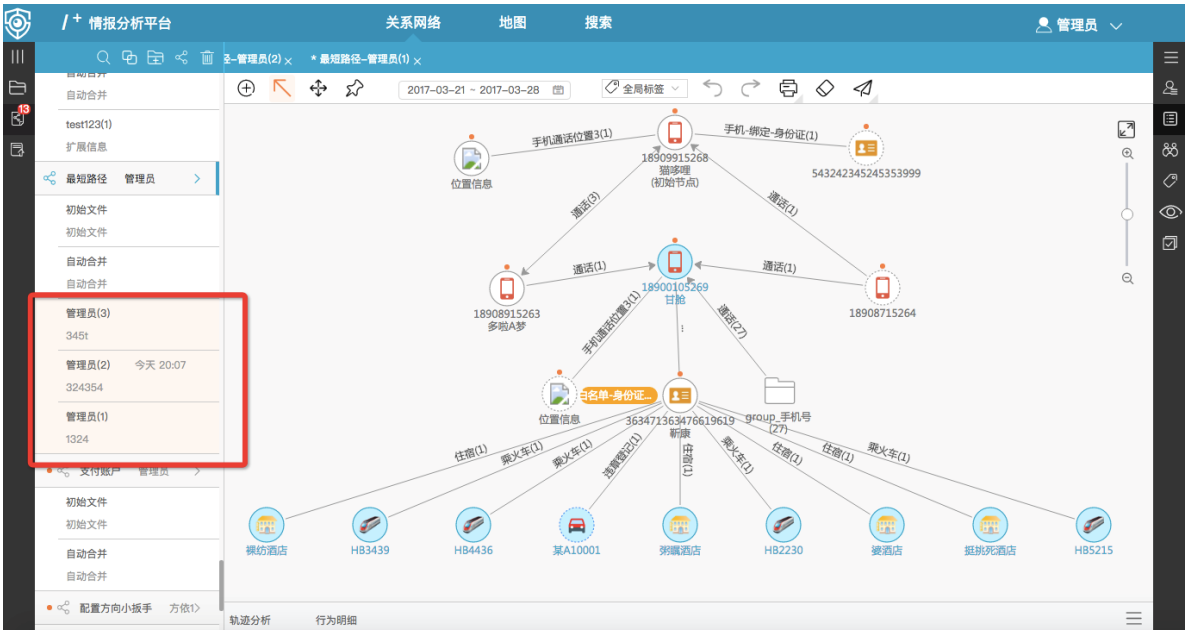
背景信息

您可以选择多个同一个分享分析的多个版本进行合并。

操作步骤

1. 选择需要合并的分析，通过键盘上的**Control**键来选择，如图 *1457: 选择分析*所示：

图 1457: 选择分析



2. 单击合并按钮，如图 *1458: 合并文件*所示。

图 1458: 合并文件

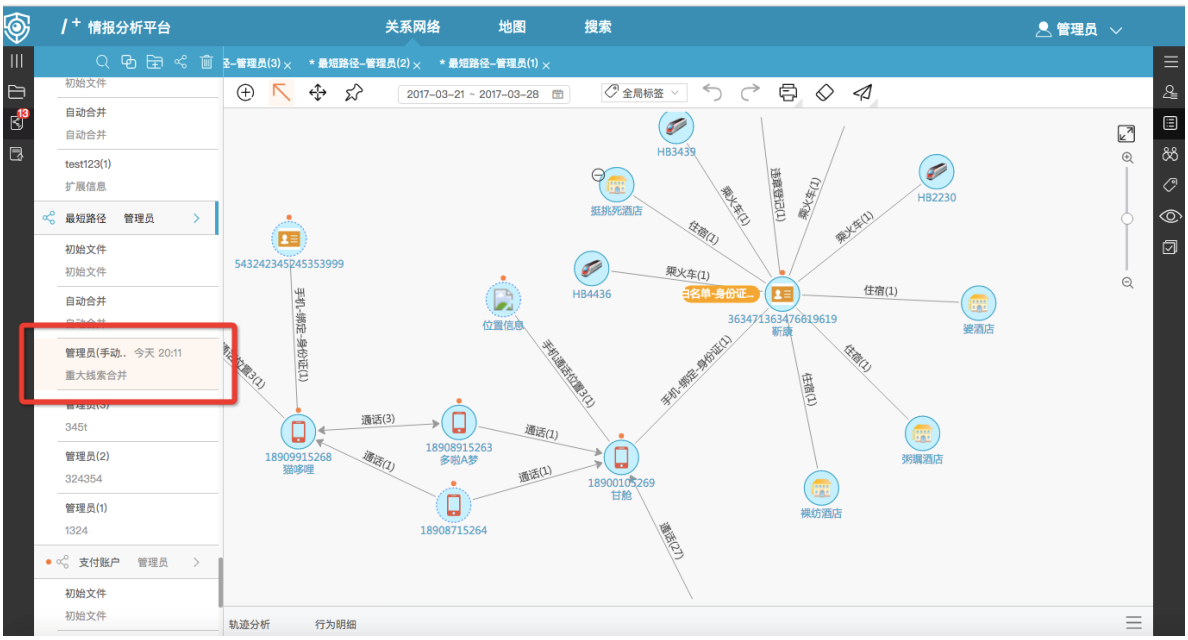


3. 填写备注。

   文件合并成功后，系统会提示**合并成功**。

4. 您可以查看合并后的结果，如图 *1459: 查看结果*所示。

图 1459: 查看结果



当版本中有不一致的地方，取并集。但是某个节点的删除个数大于留存的个数时，合并文件中该对象图表左上方有特殊标示，如图 *1460:* 特殊标示所示。

图 1460: 特殊标示



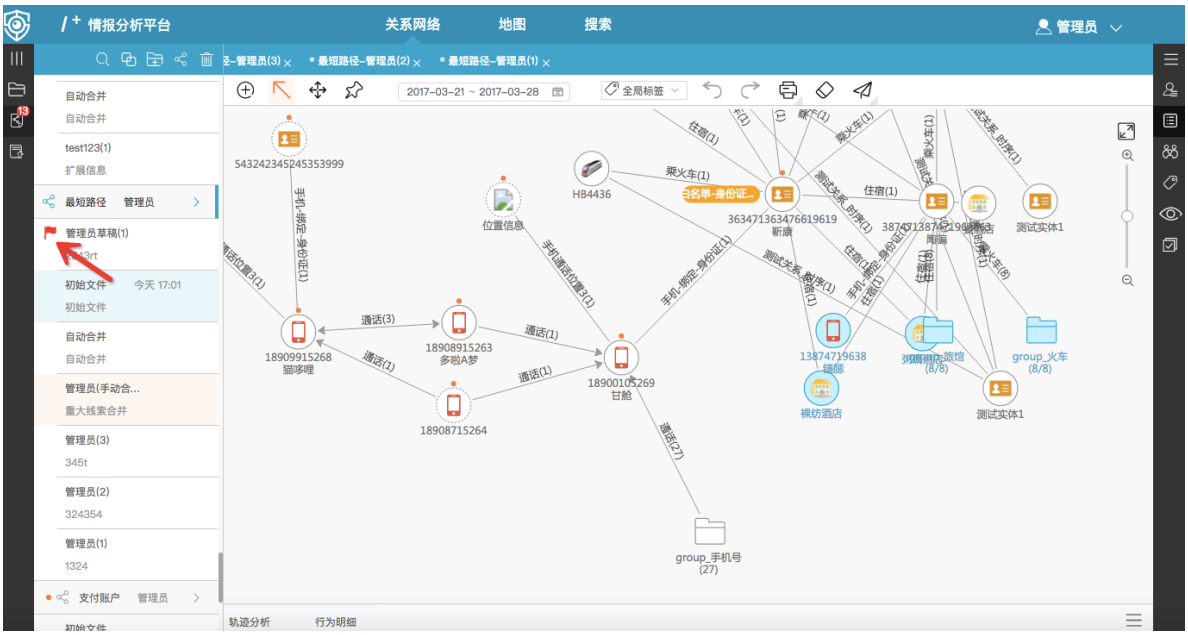
32.6.15.7 右键操作

分享目录下的文件顺序按照类型排序，从上到下依次为草稿文件、初始文件、自动合并、手动合并、已发布版本文件，同类型的文件按照时间最近排在最上面。

文件右键菜单项说明

- 重点文件标注：将重点文件进行标注，以作区分，如图 *1461:* 重点文件标注所示。该操作仅限文件分享者使用。

图 1461: 重点文件标注



- 删除文件：将文件删除后将不能再查看该共享分析。
- 修改备注：修改文件的备注信息。
- 查看文件信息：可以查看文件的详细信息，例如查看文件的作者、发布时间和备注信息（针对草稿、初始文件和已发布版本文件）或者查看文件的合并者、合并文件和合并时间（针对自动合并和手动合并文件）。

各种类型文件的右键菜单项不一样，并且不同用户（分享者和被分享者）可对文件进行的操作也不同。

从分享者和被分享者的角度，对各类型文件的右键操作权限说明如下：

分享者右键操作权限

- 草稿文件：
  - 重点文件标注
  - 删除文件
  - 修改备注
  - 查看文件信息
- 初始文件：
  - 重点文件标注
  - 修改备注
  - 查看文件信息

- 自动合并：
  - 重点文件标注
  - 查看文件信息
- 手动合并：
  - 重点文件标注
  - 删除文件
  - 修改备注
  - 查看文件信息
- 已发布版本文件：
  - 重点文件标注
  - 删除文件
  - 修改备注
  - 查看文件信息

被分享者右键操作权限

- 草稿文件：
  - 删除文件
  - 修改备注
  - 查看文件信息
- 初始文件：
  - 查看文件信息
- 自动合并：
  - 查看文件信息
- 手动合并：
  - 修改备注
  - 查看文件信息
- 已发布版本文件：
  - 重点文件标注
  - 删除文件

- 修改备注
- 查看文件信息

版本发布者和非版本发布者右键操作权限区分

版本发布者和非版本发布者的权限不一样，如图 *1462: 版本发布者右键操作权限*和图 *1463: 非版本发布者右键操作权限*所示。

图 1462: 版本发布者右键操作权限

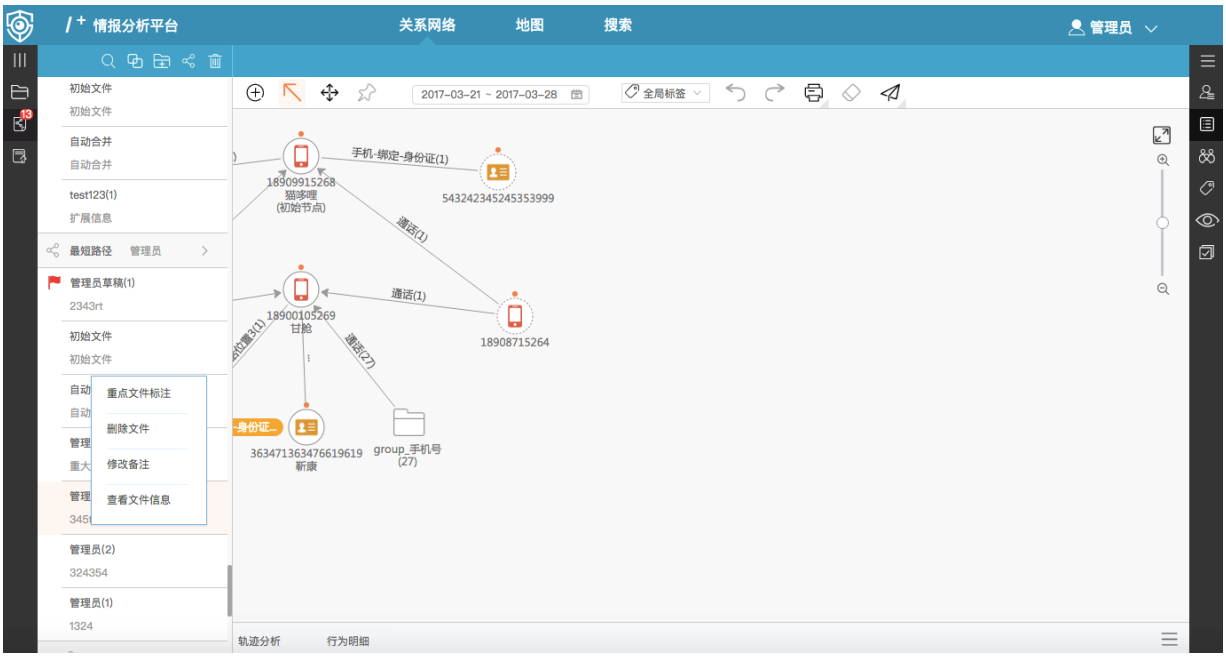
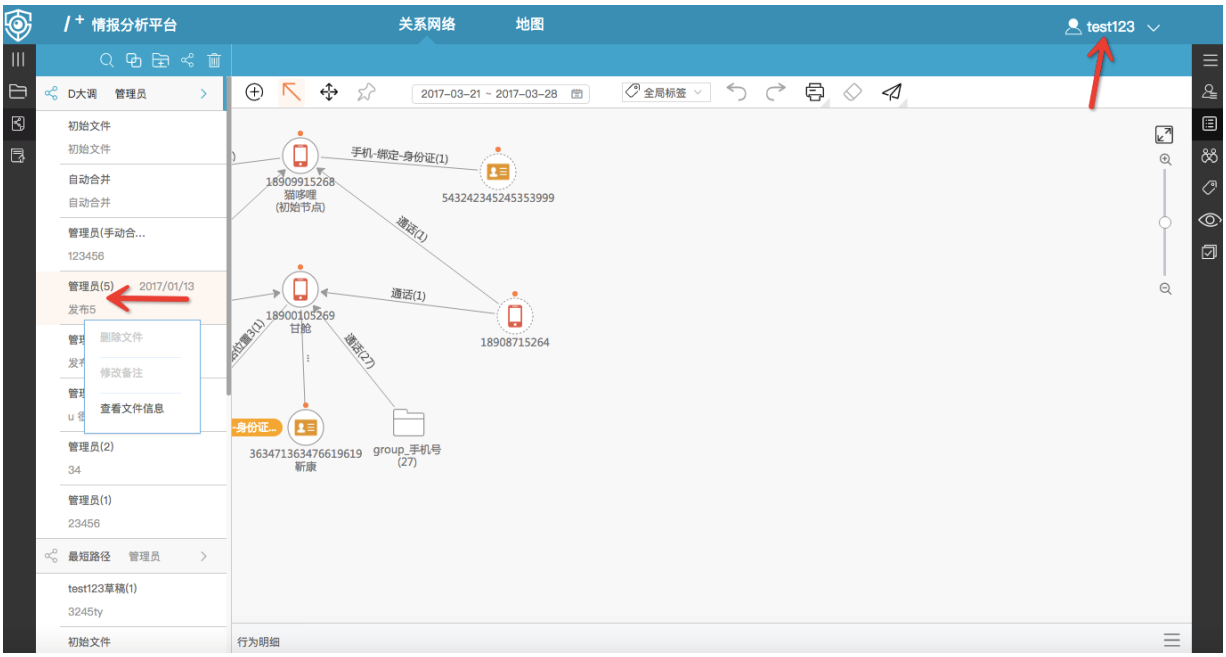


图 1463: 非版本发布者右键操作权限



32.6.16 关系方向

根据业务的需求，后台定义关系模型的时候，可以定义有向关系或者无向关系。

选择有向关系或无向关系，前端页面的显示效果不同。

有向关系

例如：通话关系，有明确指向性，方向有特定含义，关系线有箭头，如图 *1464:* 有向关系所示。

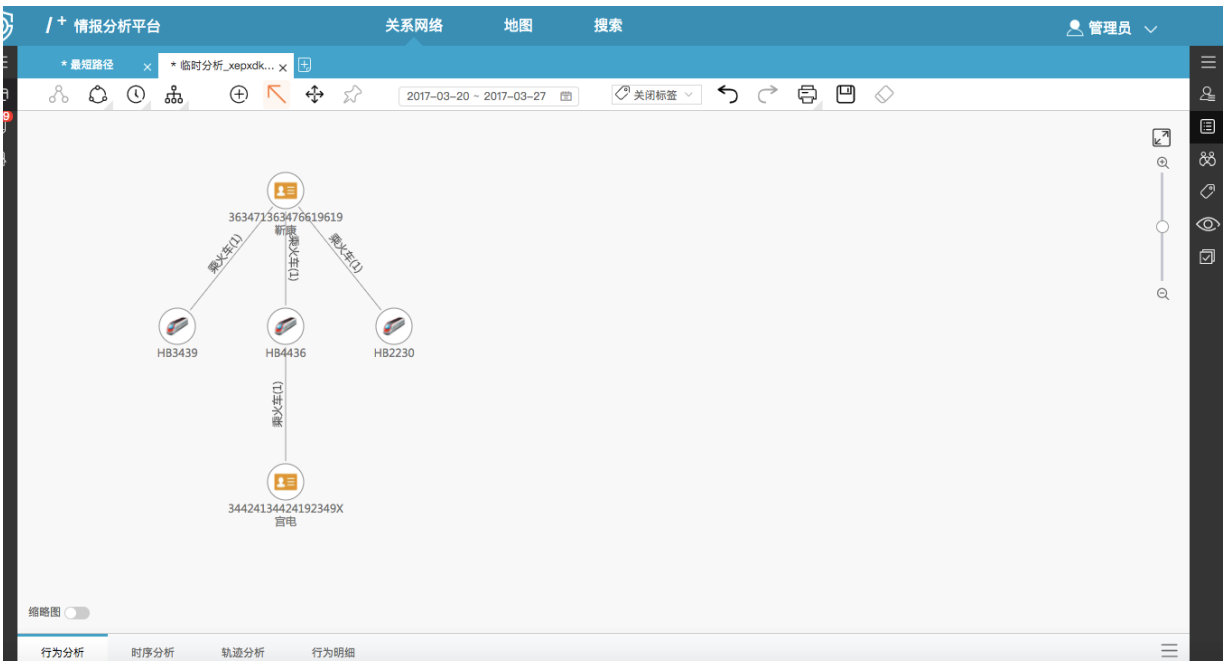
图 1464: 有向关系



无向关系

例如：人乘坐火车没有明确的指向，为无向关系，如图 *1465: 无向关系*所示。

图 1465: 无向关系



32.6.17 可视化分层

画布分析中经常会有非常多的实体对象，尤其当节点比较多的时候，业务用户需要将重点关注的某类对象突出显示。I+ 提供的可视化分层展示功能，可以将用户重点关注的对象突出显示，以作区分。

可视化分层展示功能的入口在页面右侧，如图 *1466: 可视化分层入口*所示。

图 1466: 可视化分层入口



在右侧会将当前画布中的所有实体类型和关系类型枚举出来，您可以选择需要重点关注的类型。系统支持多选，如图 *1467: 选择关注类型*所示，在众多类型中，手机号和身份证号码是需要重点关注的对象。您可以通过键盘上的 **Control** 键来进行多选。

图 1467: 选择关注类型



画布中除了手机号和身份证号码以外的实体和关系都会暗淡下来。如果继续做扩展，新出的对象或关系同样遵循定义的可视化分层。

关系分层指对应点和关系线会特别显示，支持实体和关系同时参与可视化分层，如图 *1468: 关系分层*所示。

图 1468: 关系分层



取消可视化分层效果，有以下两种方式：

- 双击画布中空白处
- 单击右侧面板的**删除**按钮

32.6.18 二次过滤

二次过滤只要为用户提供针对画布过滤的交互操作。

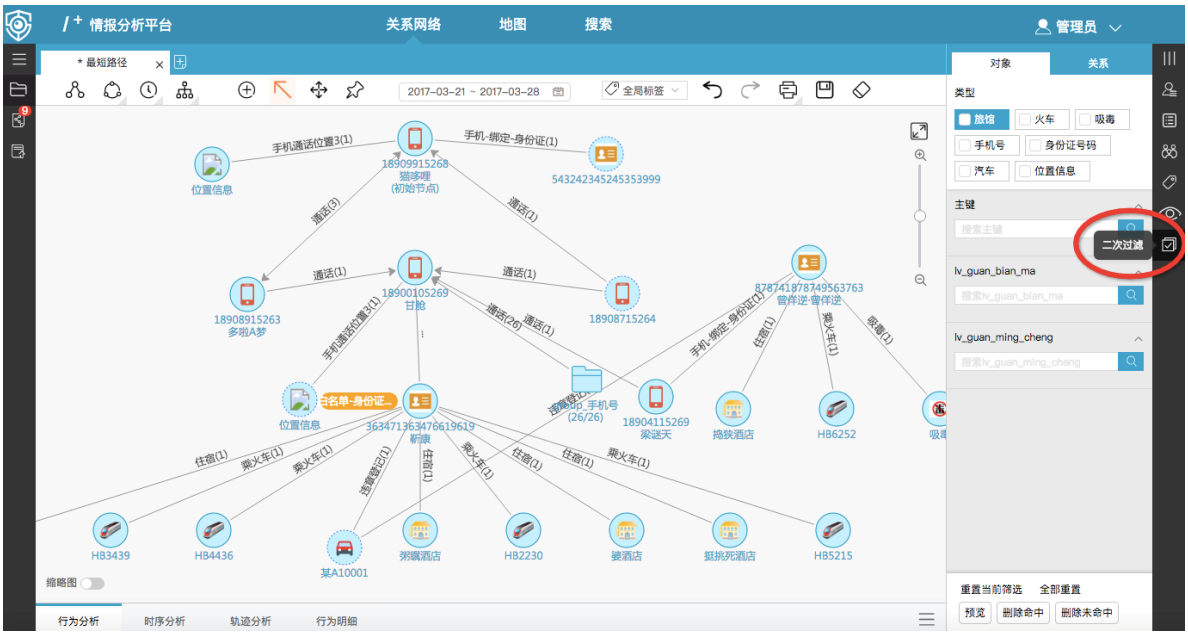
背景信息

在很多业务场景中，由于数据场景复杂，常常分析拓展几步之后，画布分析会非常复杂，继续研判的难度会加大。因此二次过滤是十分有必要的。

操作步骤

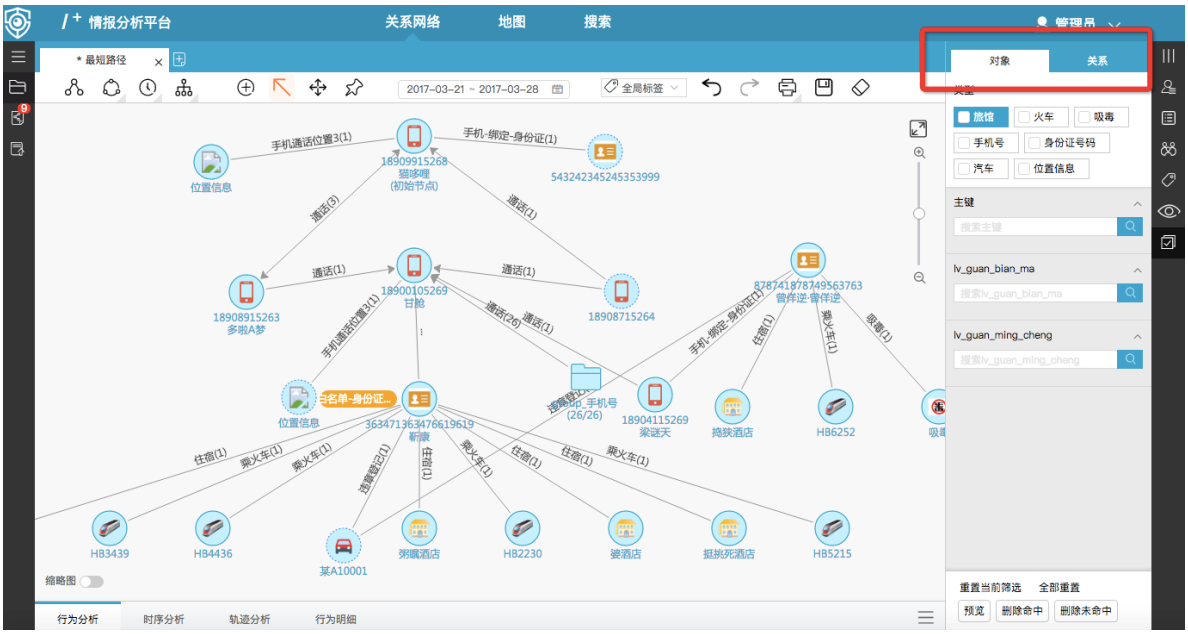
1. 在分析页面右侧，单击**二次过滤**，如图 *1469: 二次过滤入口*所示。

图 1469: 二次过滤入口



展开右侧面板的二次过滤界面。二次过滤提供基于关系和对象的协同过滤策略，单击可以来回切换，并且会保留之前的编辑状态，如图 *1470: 二次过滤*所示。

图 1470: 二次过滤



2. 选中画布后，在**对象**页签或**关系**页签中设置属性进行筛选。

   相关参数说明如下：

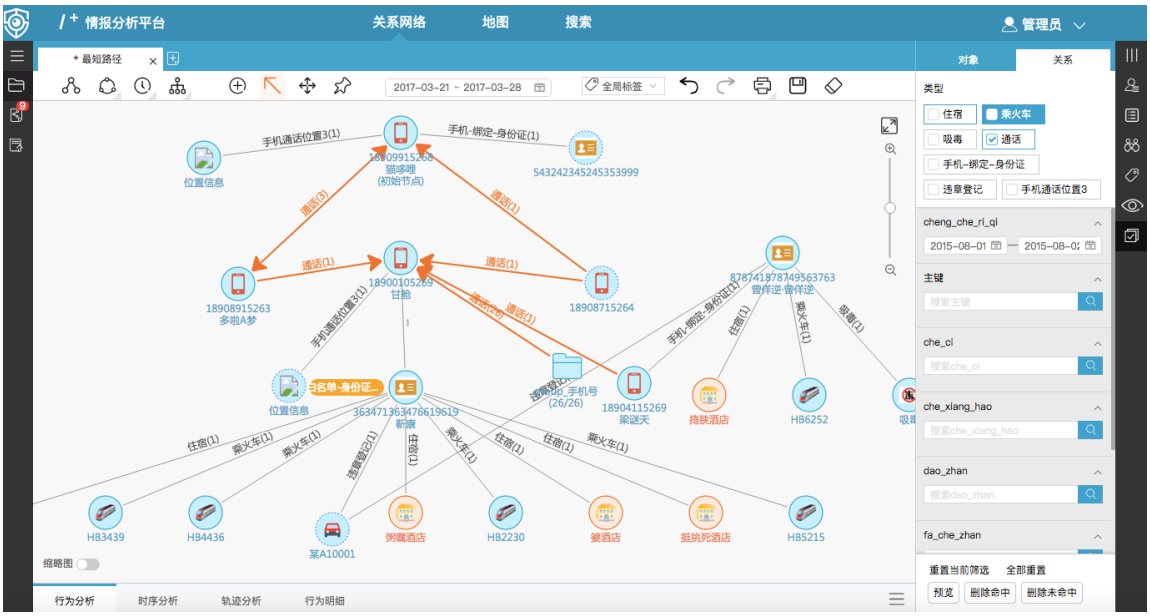
   - 时间类型：列出时间属性的最大值和最小值，用户可以调整范围，如发车时间。
   - 数值型：列出数值的最大值和最小值，用户可以调整范围，如年龄。

- 字典型：全部枚举，用户可以删除部分。
- 字符串：用户搜索，支持模糊搜索。

3. 可进行如下操作：

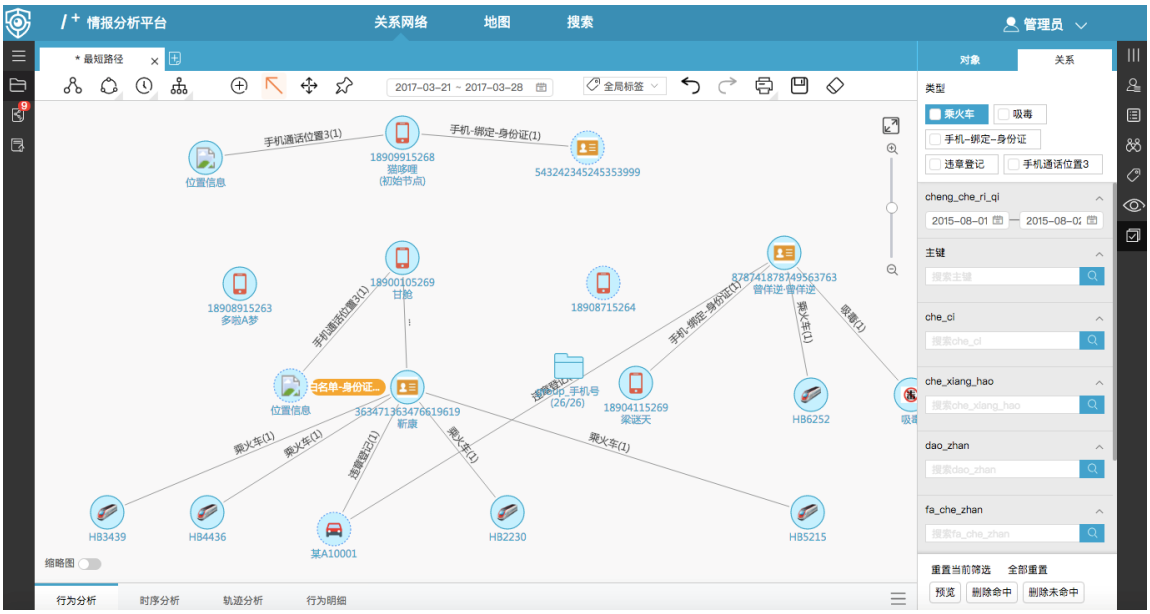
- 重置当前筛选：当前被编辑的类型属性条件重置。
- 全部重置：所有对象和关系的筛选条件全部重置。
- 预览：针对当前所有条件，预览哪些关系和对象命中条件，如图 *1471: 预览*所示。

图 1471: 预览



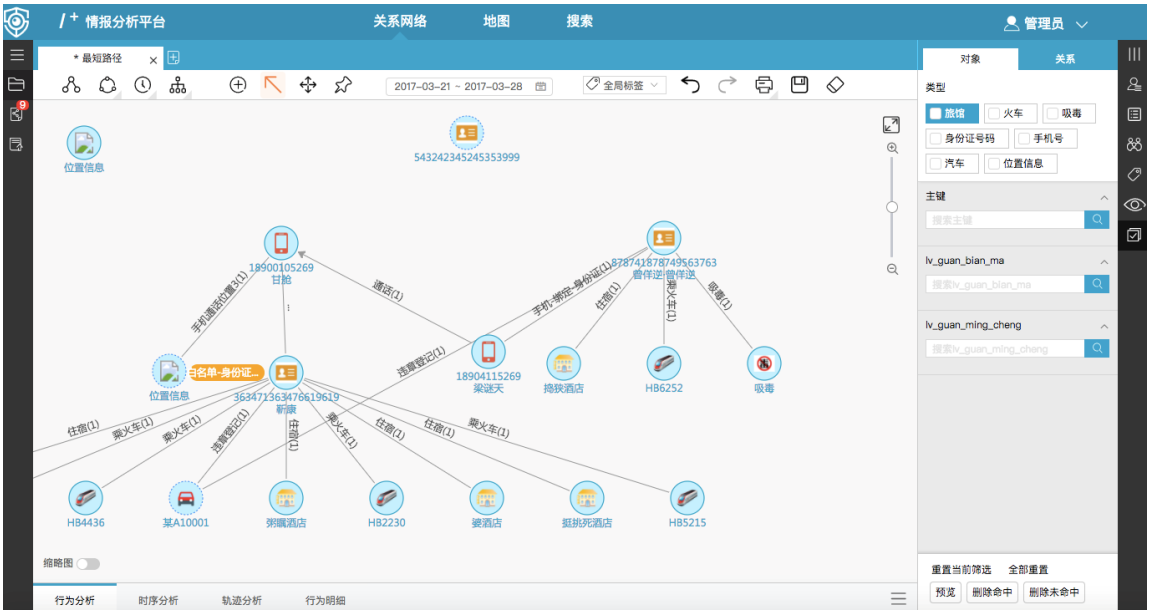
- 删除命中：将命中的对象节点和关系删除，如图 *1472: 删除命中*所示。

图 1472: 删除命中



- 删除未命中：针对选择的对象和关系类型，保留条件命中的具体对象和具体关系，其它的删除，如图 *1473:* 删除未命中所示。

图 1473: 删除未命中



32.7 地图分析

地图分析引入空间维度，将实体、关联等数据和 GIS 地图结合，发掘出空间关联关系，并利用机器学习，智能计算同轨伴随、空间活动等信息。

GIS 模块作为 I+ 情报分析平台三大主要模块之一。其主要功能有三个：显示对象地点、显示对象轨迹和伴随分析。

如图 *1474: GIS*所示，GIS 前端左上侧的有放大缩小按钮，来控制GIS地图显示的级别；右上角有两个按钮：临时分析和分析结果列表，该列表多为伴随分析结果列表。

图 1474: GIS



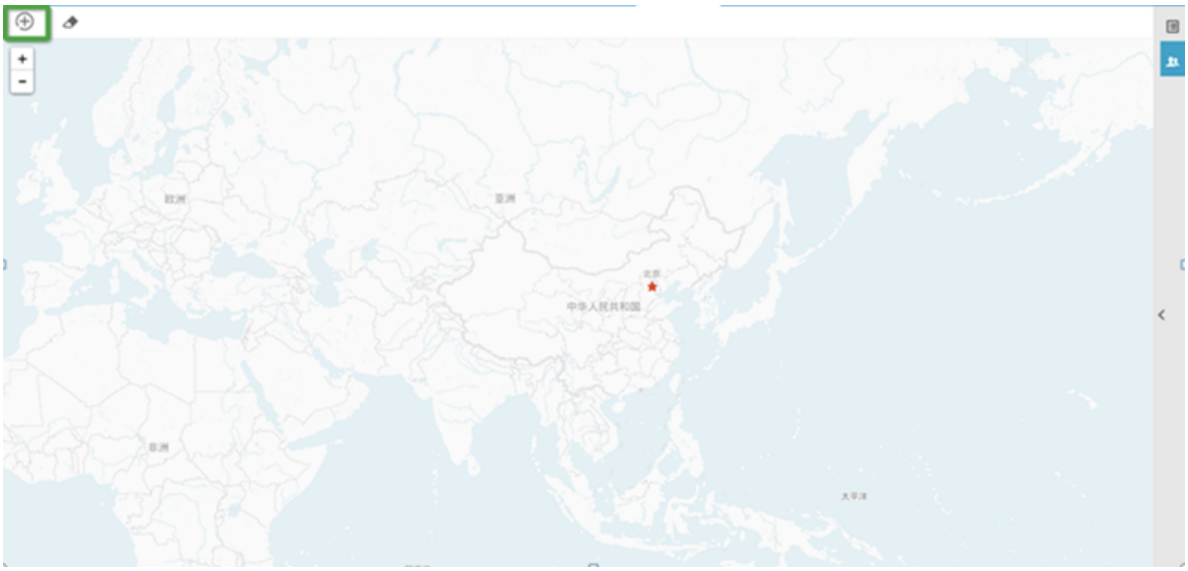
显示对象地点

该地点是该电话号码最近一次出现地点。

操作步骤

1. 单击，添加节点，如图 *1475:* 添加节点所示。

图 1475: 添加节点



2. 在**添加分析对象**对话框中，输入电话号码，单击**确定**，如图 *1476:* 输入电话号码所示。

图 1476: 输入电话号码



显示节点GIS位置，如图 *1477:* 显示*GIS*位置所示。

图 1477: 显示GIS位置



3. 调动放大（左侧 + 、 - ）按钮，来观测精确位置。
4. 在右侧工具条上，单击  ，展示的临时分析区中，有该对象的详细属性信息，如图 *1478: 展开临时分析区*所示。

图 1478: 展开临时分析区



显示对象轨迹

操作步骤

1. 在已经显示位置的对象点右侧，单击属性面板中的**轨迹展现**，如图 *1479: 轨迹展现*所示。

图 1479: 轨迹展现



2. 在出现的对话框中，设置时间范围（起始时间～结束时间），如图 *1480: 设置时间范围*所示。

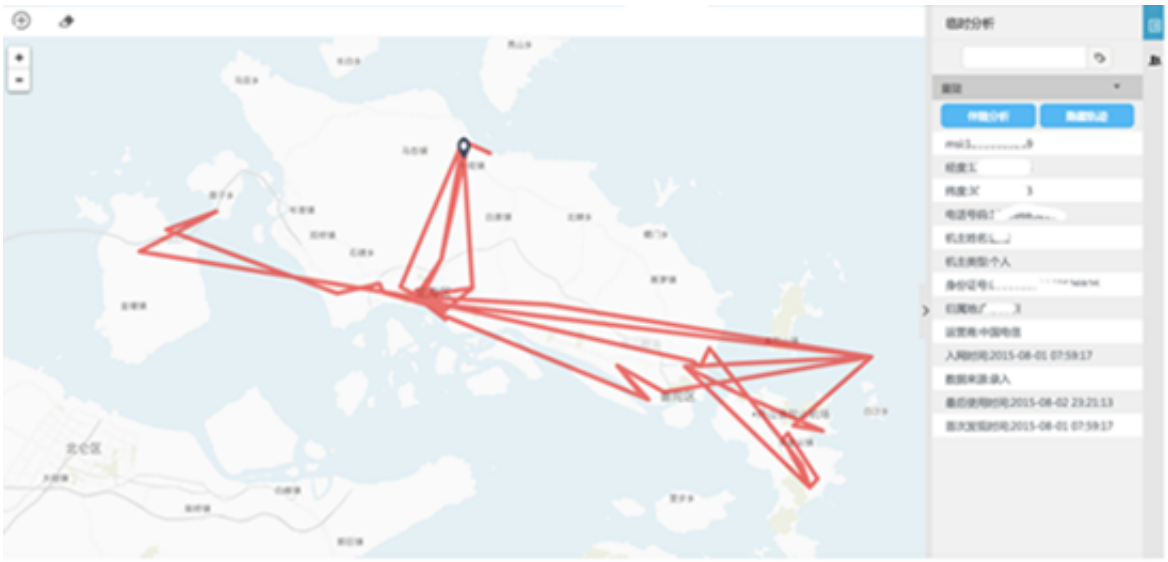
图 1480: 设置时间范围



3. 单击**确定**。

出现如图 *1481: 轨迹图*所示轨迹图。

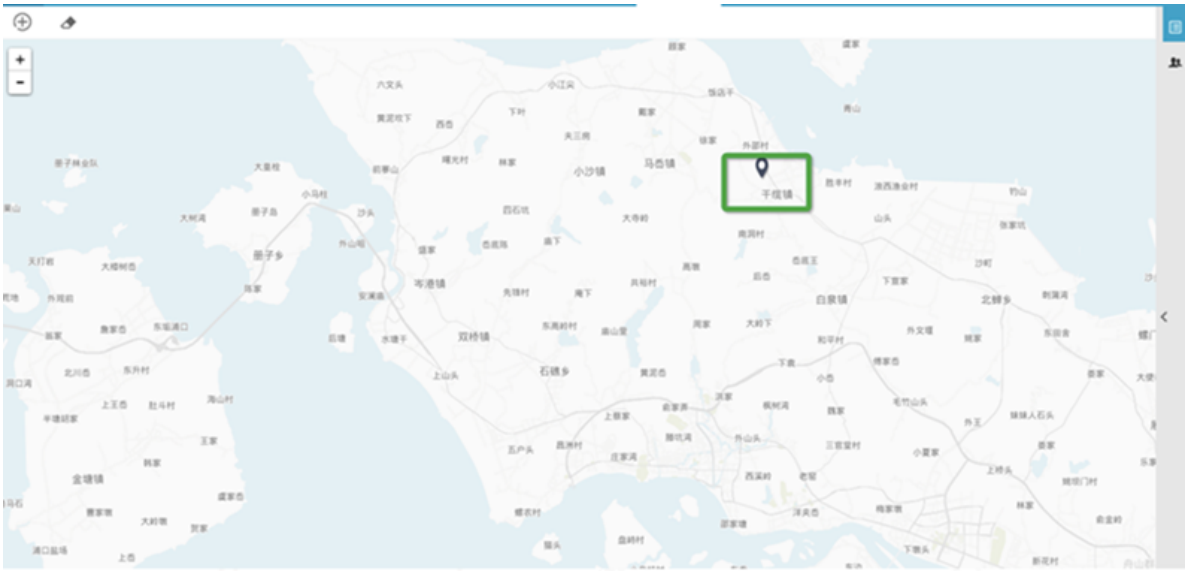
图 1481: 轨迹图



4. 单击右侧面板中的**隐藏轨迹**，可以隐藏显示的轨迹。

   会将显示的轨迹结果隐藏，如图 *1482:* 隐藏轨迹所示。

   图 1482: 隐藏轨迹



伴随分析

操作步骤

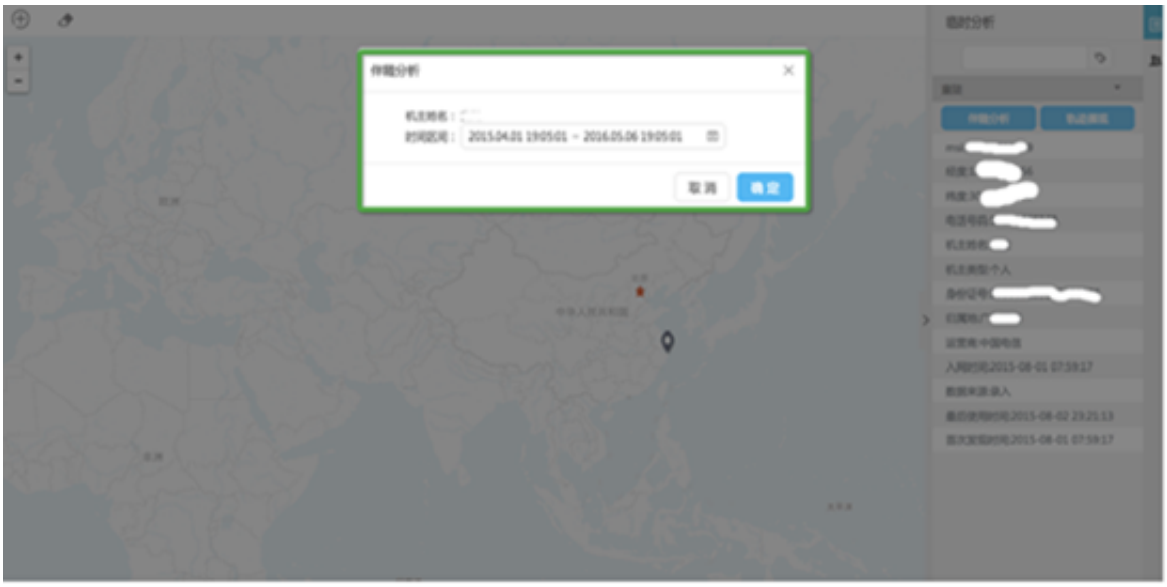
1. 在临时分析中的对象节点属性面板中，单击**伴随分析**，如伴随分析所示。

   图 1483: 伴随分析



2. 在弹出的对话框中，选择伴随的时间区间（开始时间～结束时间），如图 *1484: 选择伴随时间*所示。

图 1484: 选择伴随时间



3. 单击**确定**。

在**结果列表**中会出显示伴随结果。用户可以展开查看伴随相似度分数和结果对象的基本信息。

说明: 伴随结果按伴随相似度排序，且会计算前 TopN 的结果（TopN 可以配置调整，默认为50）。

32.8 用户意见反馈

I + 作为一个研判平台，解决复杂的业务问题，为了提升产品质量，I + 提供用户意见反馈功能。

32.8.1 提交用户反馈意见

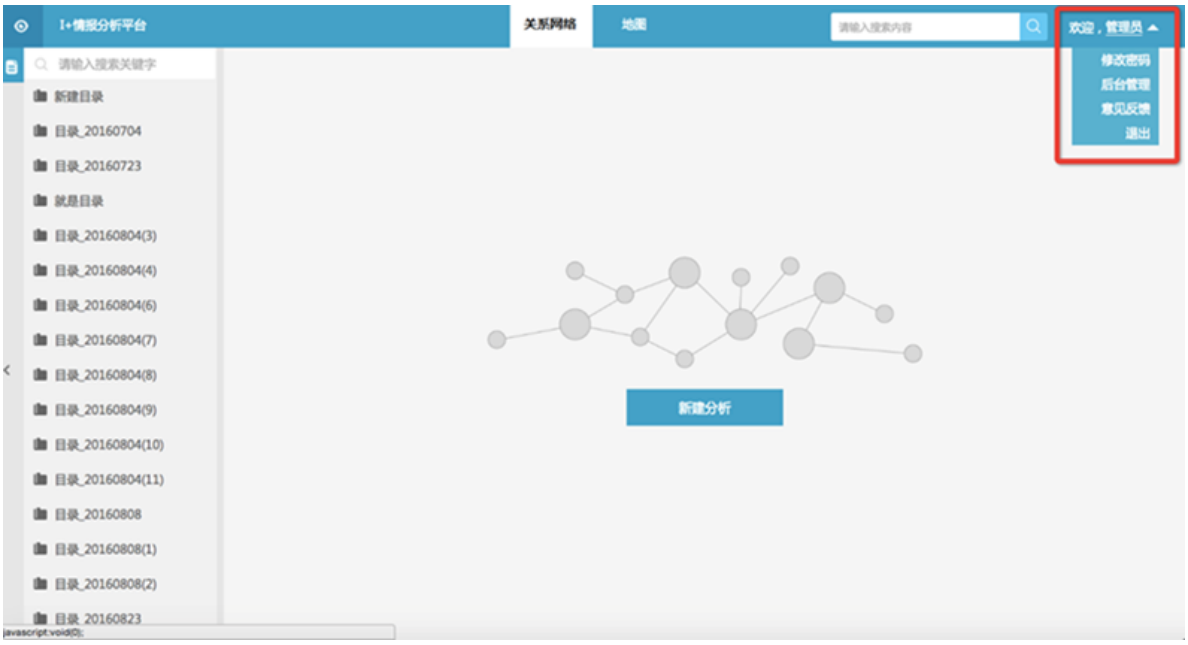
用户可以通过意见反馈功能，提交对产品的意见或建议。

操作步骤

1. 用户登录I + 系统后，单击右上角的**意见反馈**，如图 *1485:* 意见反馈所示。

   显示**意见反馈**对话框。

   图 1485: 意见反馈



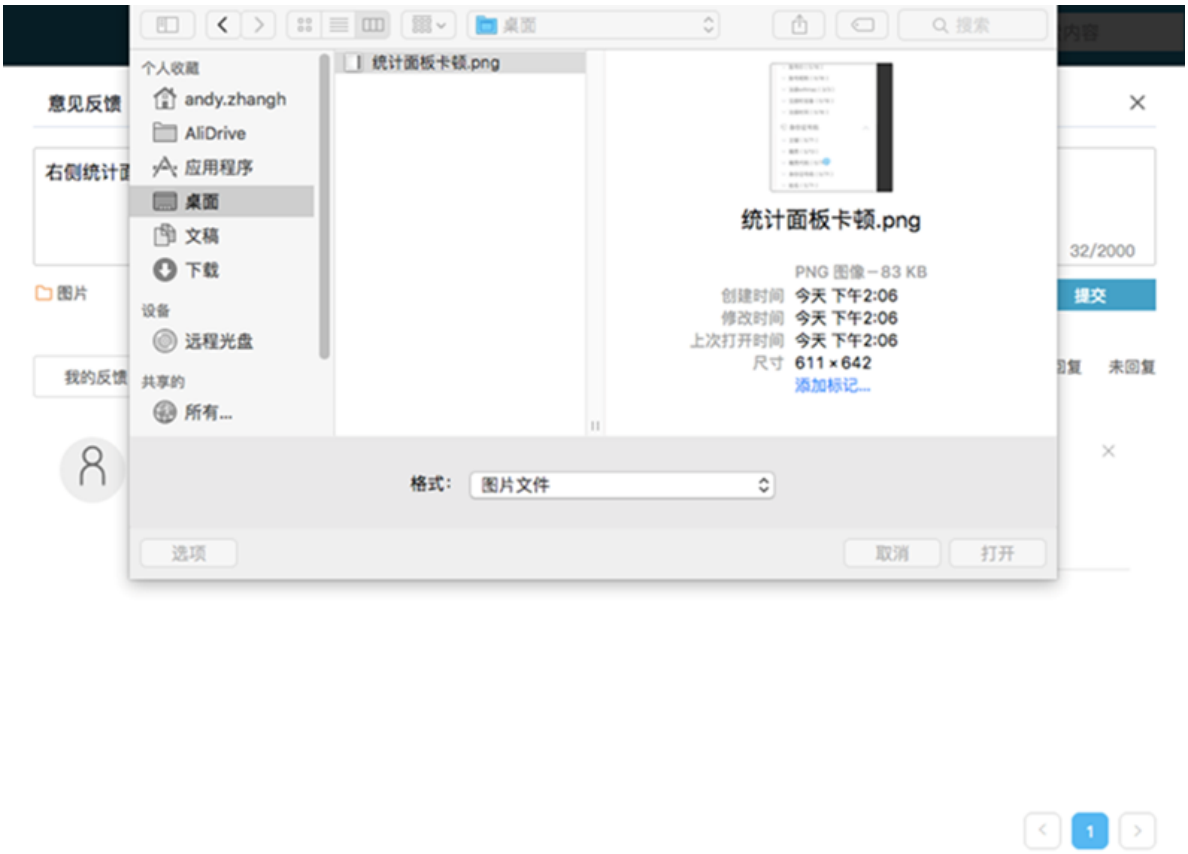
2. 在反馈栏中填写自己的意见，如图 *1486:* 填写反馈意见所示。

   图 1486: 填写反馈意见



3. 如果需要添加附件，可以单击**图片**，在弹出的文件目录框中，选择需要上传的附件，如图 *1487: 上传附件*所示。

图 1487: 上传附件



选中之后，图片会加载到意见反馈内容框中。

4. 用户可根据需要选择意见类型，默认为**产品建议**。

   反馈意见提供**产品建议**、**产品问题**、**帮助咨询**和**其他**四种类型，如图 *1488:* 意见提交成功所示。

5. 单击**提交**按钮，反馈意见提交成功。

   提交成功后，在**我的反馈**中就会出现刚刚提交的反馈意见。

图 1488: 意见提交成功



32.8.2 查看用户反馈意见回复

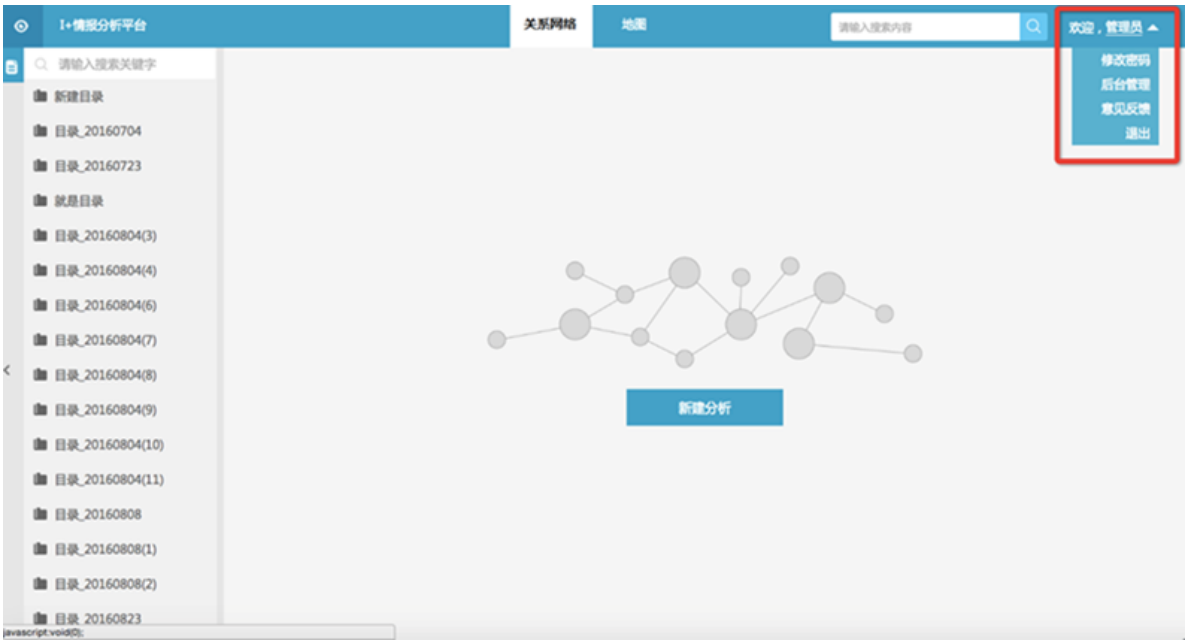
用户反馈产品意见或建议后，I + 技术支持会给予回复，用户可通过查看回复解疑。

操作步骤

1. 用户登录I + 系统后，单击右上角的**意见反馈**，如图 *1489: 意见反馈*所示。

   显示**意见反馈**对话框。

   图 1489: 意见反馈



2. 点击**我的反馈**或**所有反馈**查看已反馈的意见。

   意见反馈中可以按照回复状态进行分类，如图 *1490:* 回复状态分类所示。

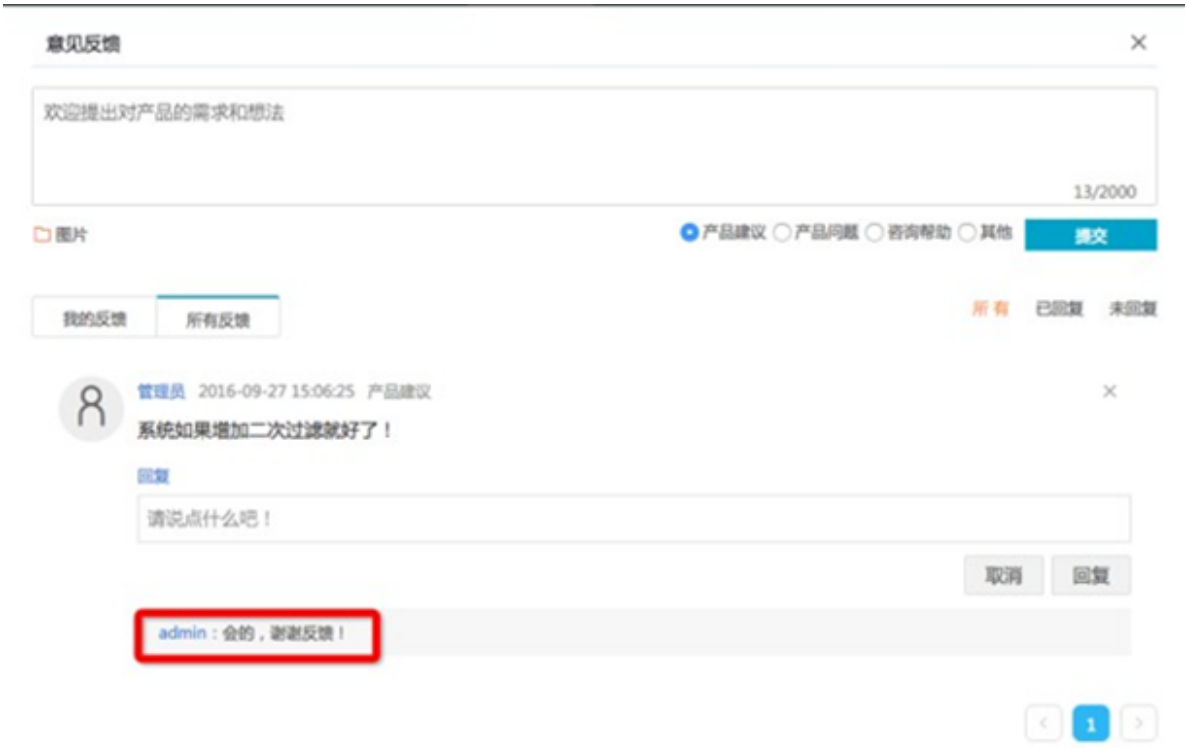
   图 1490: 回复状态分类



3. 点击**已回复**可查看已经得到回复的意见。

4. 针对其他用户提交的问题，用户也可进行回复，如图 *1491:* 回复他人所示。

图 1491: 回复他人



32.8.3 系统管理员删除评论

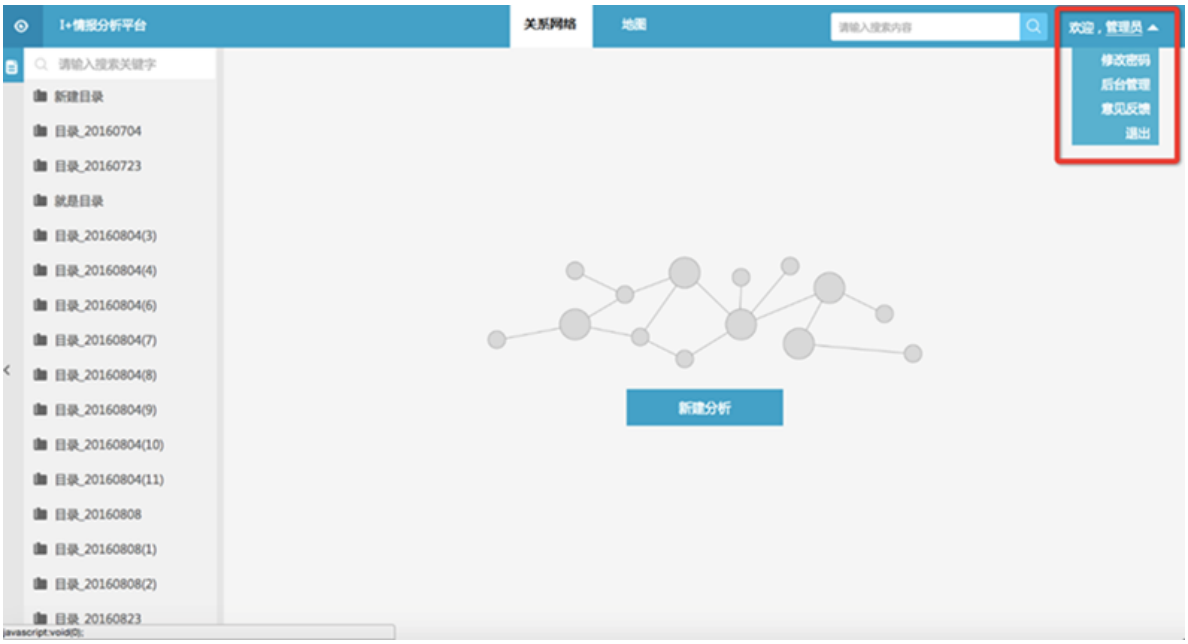
作为系统管理员，可以删除用户的评论。

操作步骤

1. 系统管理员登录I + 系统后，单击右上角的**意见反馈**，如图 *1492:* 意见反馈所示。

   显示**意见反馈**对话框。

图 1492: 意见反馈



2. 单击每条反馈的右上角叉号，如图 *1493:* 管理员删除评论所示。

图 1493: 管理员删除评论



32.9 应用场景（具体案件模拟）

本案例讲述如何通过 I+ 平台侦破复杂 BK 案。

前提条件

本案例所需数据资源如下：

- 通讯工具、即时通讯通联
- 人员基本信息
- 手机轨迹、车轨迹
- 租车信息、住宿信息、火车信息
- 护照信息、出入境记录

背景信息

2015年底，某部门监测到部分小众通讯工具异常活跃，使用该工具的账号与境外某国家频繁联系，形迹可疑，按照工作要求，分析人员通过I+快速核实情况，甄别风险。

注意: 特别申明：本故事不代表真实案件，仅为 I+ 平台功能侧杜撰！

操作步骤

分析过程

1. 分析人员筛选2015年12月份活跃用户，如图 *1494: 筛选活跃用户*所示。

图 1494: 筛选活跃用户



结果显示，有一批账号在该时段频繁使用小众工具，如图 *1495:* 筛选结果所示。

图 1495: 筛选结果



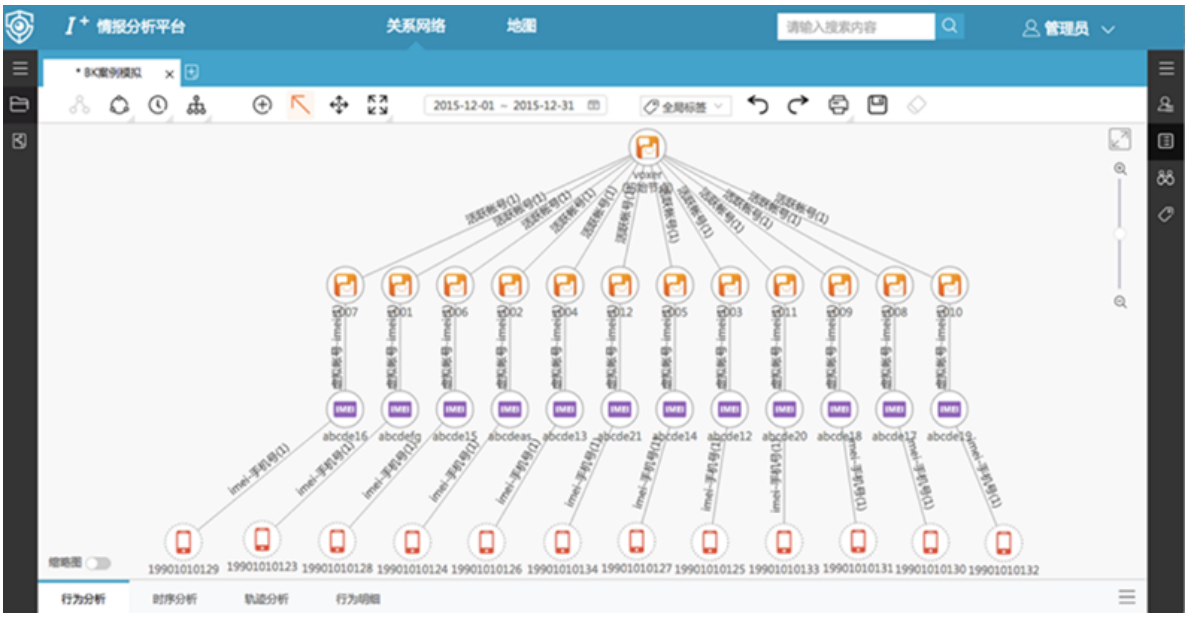
2. 进一步分析这批账号在这段时间的通联关系，结果显示，这批账号与境外某国的另外一批账号存在单线联系，分析人员尝试获取境外账号的情况，没有查到相关信息，如图 *1496:* 账号间的通联关系分析所示。

图 1496: 账号间的通联关系分析



3. 分析人员尝试获取到这批账号的实际使用人，进一步拓展分析发现，这批账号分别在不同的手机上使用，关联反查得到对应的手机号，结果显示，这批手机号只在单独的手机上使用过，如图 *1497: 账号使用人分析*所示。

图 1497: 账号使用人分析



4. 对这批手机进行分析，未获取到其开户信息，但发现与同一个手机号码存在联系，如图 *1498: 关联到同一手机号*所示。

图 1498: 关联到同一手机号



5. 进一步分析发现，此号码在多个地点出现，如图 *1499:* 对手机号进一步分析所示。

图 1499: 对手机号进一步分析



6. 接下来，分析人员对该号码展开分析，尝试获取与其同行的其他号码，如图 *1500:* 对可疑号码展开分析所示。

图 1500: 对可疑号码展开分析



如图 *1501: 获取同行的其他号码*和图 *1502: 获取到伴行关系*显示，19701010123与该号码存在6次伴行关系。

图 1501: 获取同行的其他号码

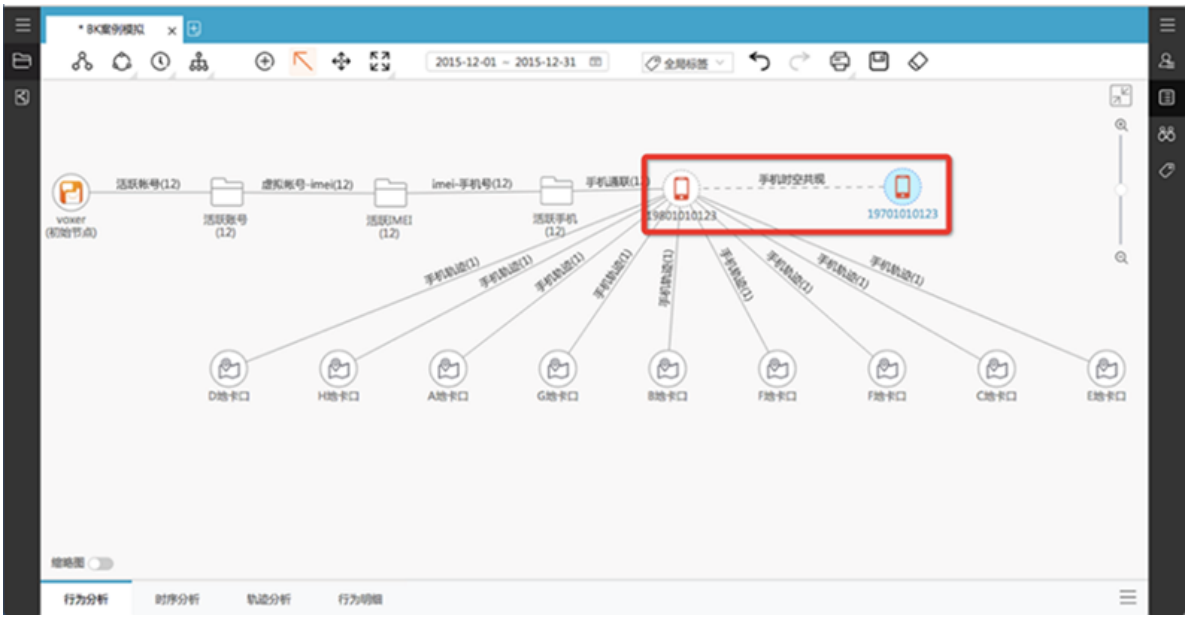
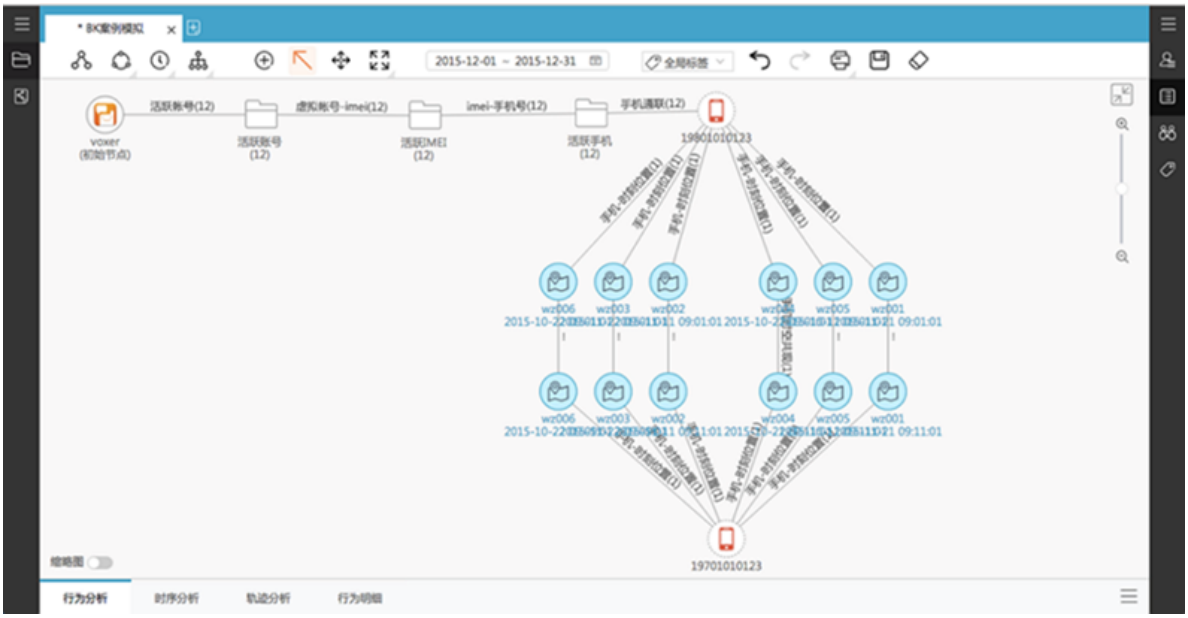
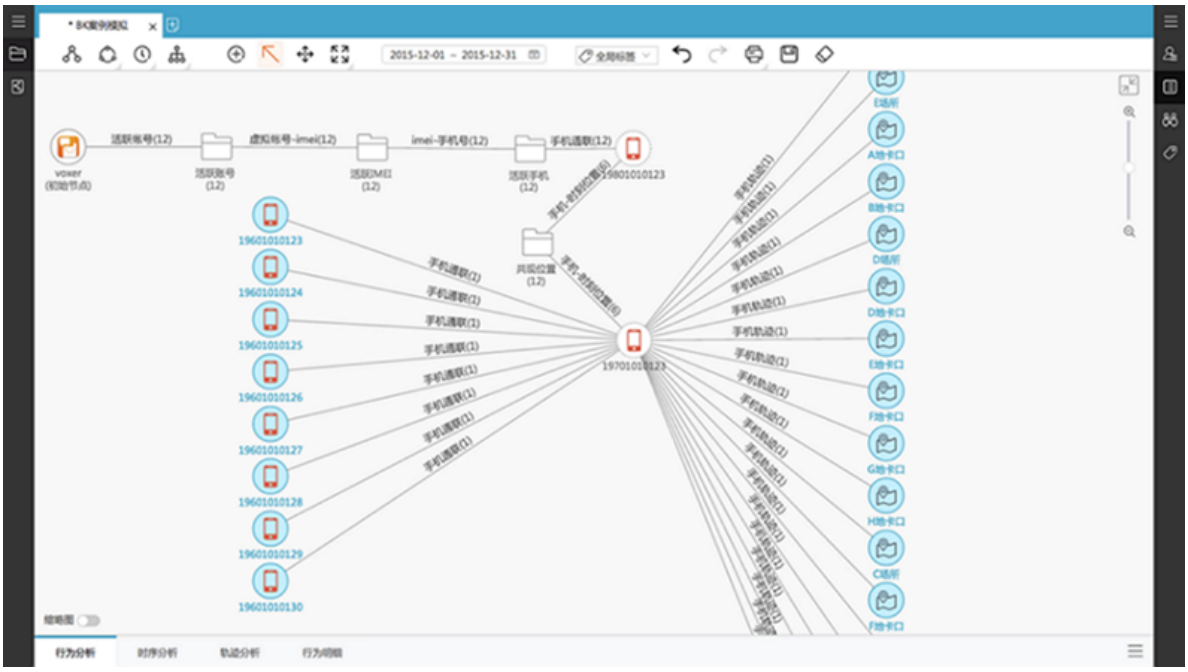


图 1502: 获取到伴行关系



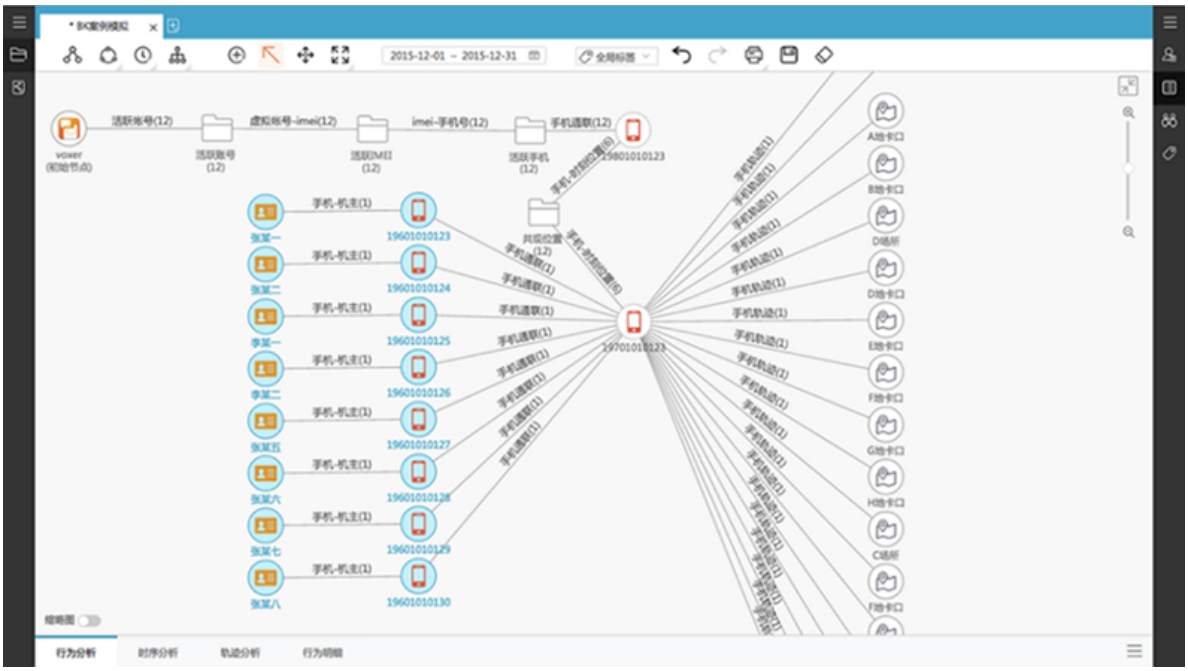
7. 根据经验判断，19701010123很可能为19801010123持有的另一部手机。分析人员对19701010123开展分析，发现该部手机存在大量的通联信息、位置信息，如图 *1503: 19701010123号码分析结果*所示。

图 1503: 19701010123号码分析结果



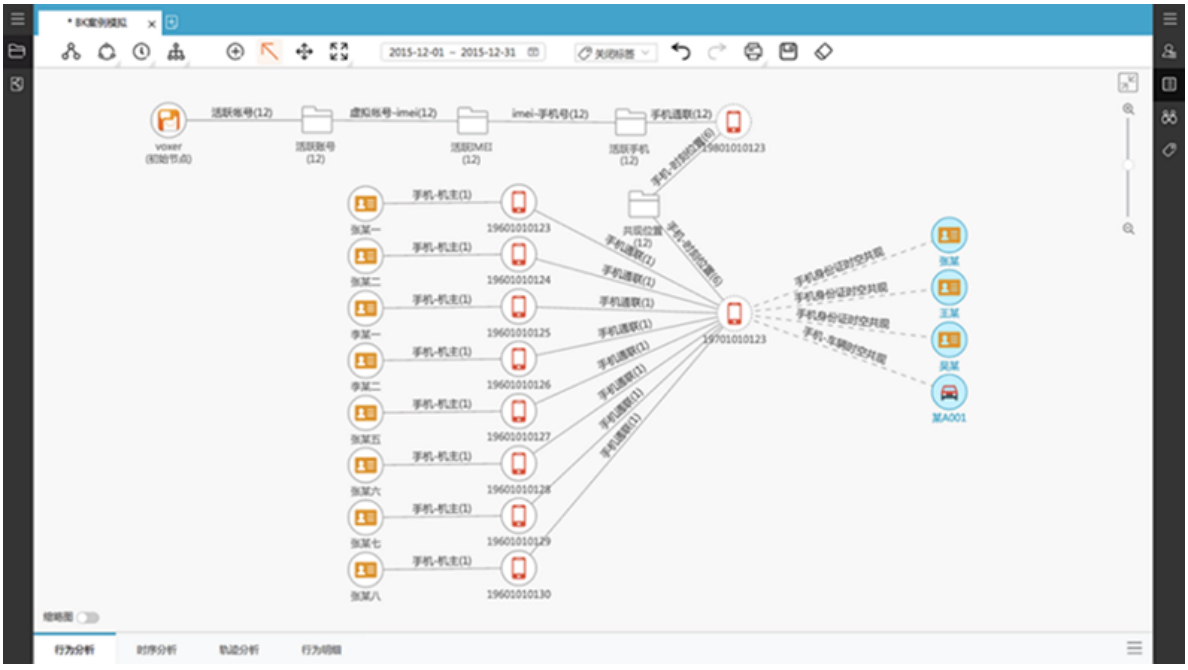
8. 对该手机的通联对象展开分析，发现这批通联号码存在机主登记信息，如图 *1504:* 通联对象分析结果所示。

图 1504: 通联对象分析结果



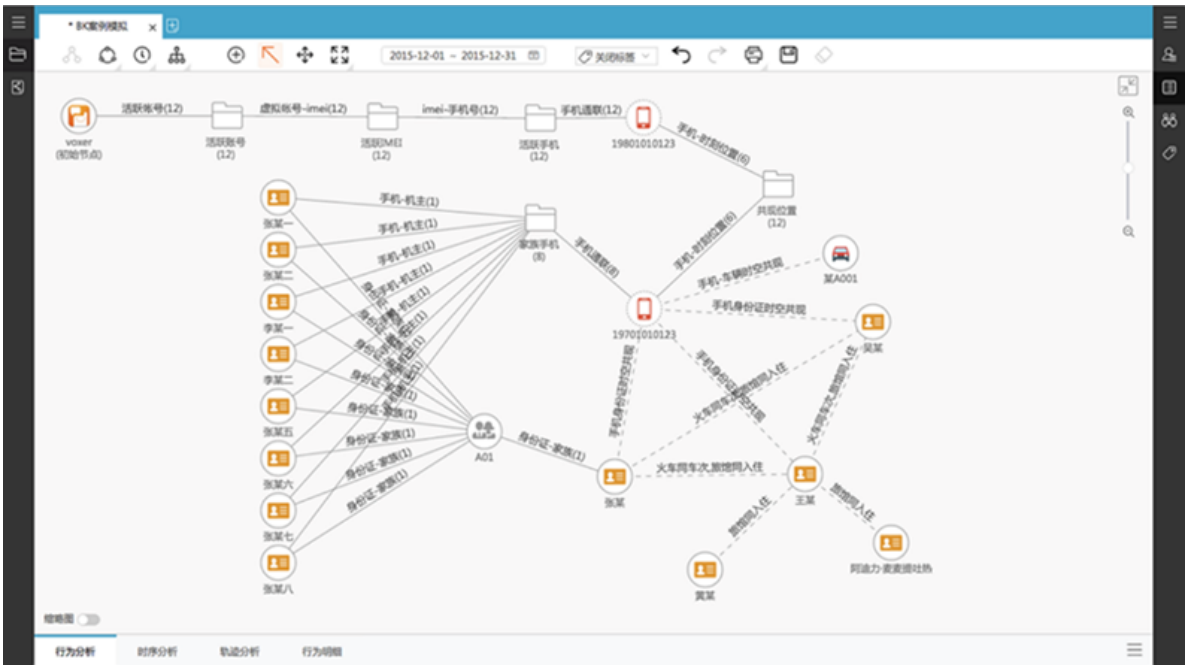
9. 通过对19701010123的轨迹进行分析，发现该号码与张某等三人及某A001车存在伴行关系，此手机号很有可能为此三人所有，如图 *1505: 19701010123轨迹分析结果*所示。

图 1505: 19701010123轨迹分析结果



10.进一步对上述发现的身份证进行分析，发现其中9人为同一家族成员，且张某与另外两人存在多次同行同住关系，同时，还挖掘到黄某、尚某与b12345123存在密切往来，三人曾经多次入住相同酒店，如图 *1506: 身份证分析结果*所示。

图 1506: 身份证分析结果



11.对黄某、尚某等四人进行重点分析，如图 *1507: 重点分析结果1*和图 *1508: 重点分析结果2*所示，发现此四人为 SK 前科人员，2011年由于涉嫌某 BK 案件被逮捕，于2015年释放出狱，结合上述异常行为，分析人员判断此四人很可能预谋作案。

图 1507: 重点分析结果1

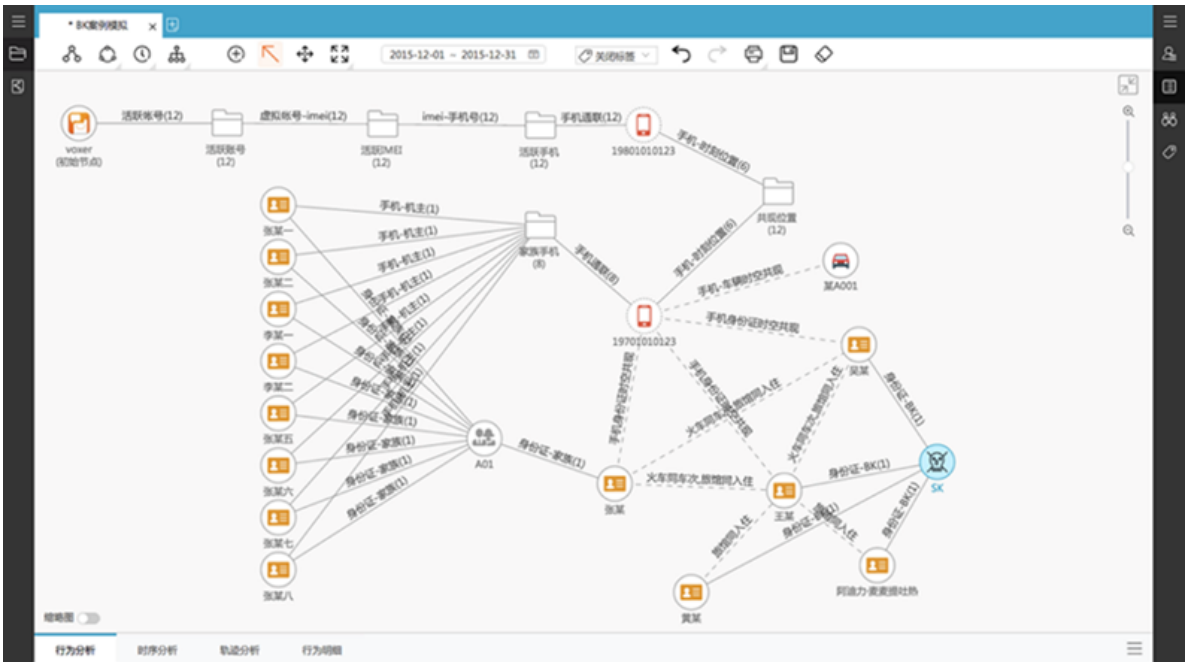
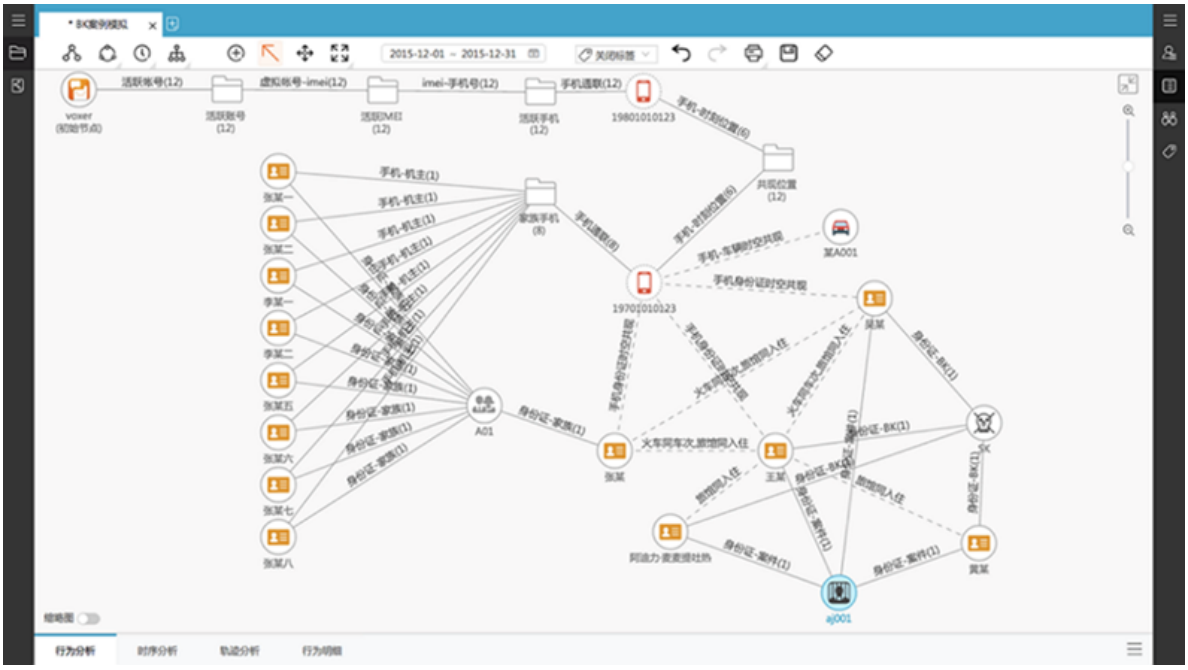
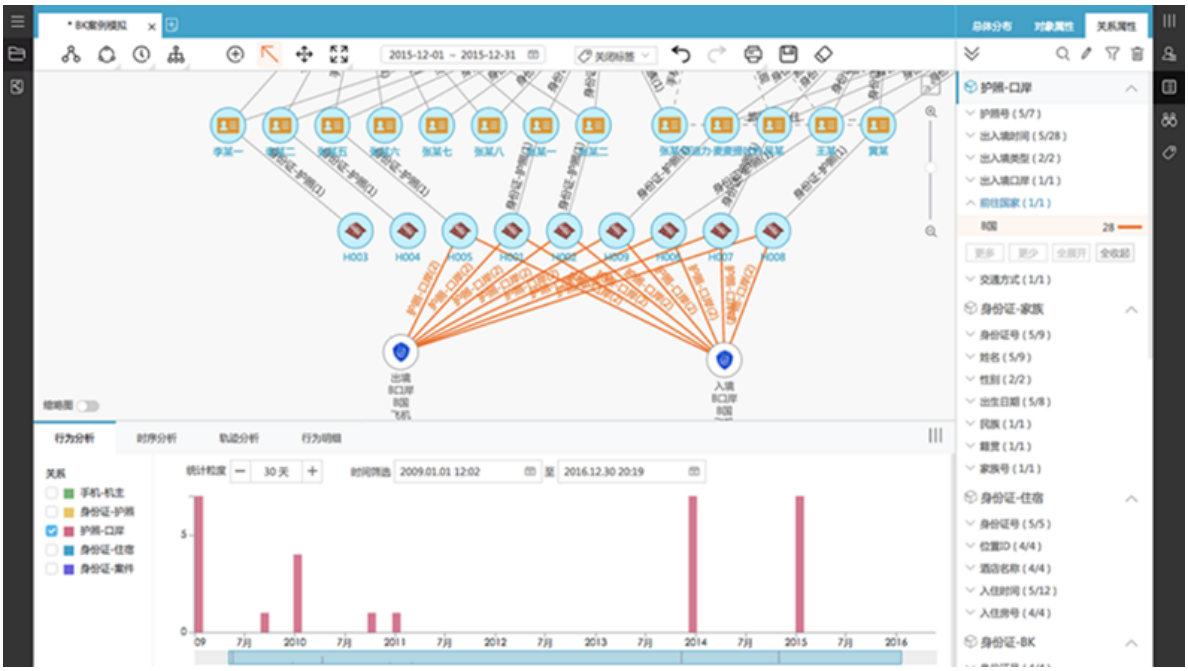


图 1508: 重点分析结果2



12.再次拓展分析发现，黄某、尚某以及张某家族共有9人持有护照，并于2009年、2014年、2015年相继往返 B 国，如图 *1509:* 再次拓展分析所示。

图 1509: 再次拓展分析



B 国为境外 SK 敏感国家，上述异常监测中所发现的小众通讯工具通联对象所在国家恰好也在 B 国，分析人员初步认为，张某、黄某、尚某等人很有可能预谋作案。

以上情报立即上报相关部分，经过民警实地排查，在某出租屋内发现张某、黄某、尚某等人正预谋在2016年元旦期间实施BK袭击，相关部门当即调派精干警力，摧毁了该团伙，避免了恶性案件的发生。

32.10 常见问题

1. 问：首次怎么登录？

   回答：请与对应管理员联系获取登录账号，首次登录初始密码默认为“123456”，为保证账户安全，请按照系统提示修改密码。

2. 问：登录时提示输入的用户名和密码不正确？

   回答：如果无法正常登录，请您检查输入的账户名和密码是否正确，密码输入时是否在半角状态、大小写是否正确、有没有留空格等。如再无效则联系管理员。

3. 问：对节点双击关联反查默认规则是什么？

   回答：为方便用户快速进行分析，管理员会在后台统一设定常用关系，并配置到双击操作中，常用关系以管理员发布的为准，如需增加或删除双击快捷关系，请与管理员联系。

4. 问：全选图区节点进行关联反查，提示当前最多可选5个节点？

回答：产品默认支持5个节点的批量关联反查，如需要支持更多节点的关联反查，请与管理员联系，由管理员根据系统规模评估后进行调整。

5. 问：为什么按键盘的**delete**键无法删除节点？

   回答：目前暂不支持按**delete**键删除节点，您可以在选中节点处右键单击，在弹出菜单中选择**删除**操作。

6. 问：撤销回退操作可以支持多少步？

   回答：可支持20步的回退、撤销操作。

7. 问：为什么选择节点后**路径分析**按钮为灰色？

   回答：只有选择两个节点后，路径分析才可用。

8. 问：为什么选择一个节点，**群体分析**、**共同邻居**为灰色？

   回答：只有选择两个及以上节点，**群体分析**、**共同邻居**才可用。

9. 问：系统使用过程中，问题如何反馈？

   回答：如果在系统使用过程有产品问题或需求反馈，请在右上角意见反馈中填写对应的信息，管理员会进行处理。

10. 问：在使用过程中，发掘新的关系战法，可否进行配置？

    回答：目前关系配置由管理员统一操作，如有新的关系战法需求，请与管理员联系。

11. 问：为什么点击行为分析、时序分析、行为明细没有数据？

    回答：需要在图区选择节点后才能进行行为分析、时序分析、查看行为明细。

12. 问：为什么在没有选中图区节点的情况下，右侧显示统计信息？

    回答：在没有选中任何节点的情况下，默认对所有的节点进行统计。

13. 问：部分节点超出分析画布，如何找到？

    回答：可以打开缩略图进行导航，将画布外的信息移动到可视画布区域。

14. 问：为什么布局方式按钮都为灰色不可用？

    回答：当您选中图区节点后，布局方式才可点击使用。

15. 问：我全选了所有节点，为什么点击层次布局没反应？

    回答：层次布局为从某个节点出发查看层次关系，请选择一个节点后再点击层次布局。

16. 问：除了工具栏画布拖动按钮，有没有什么快捷操作能够拖动画布？

    回答：按住键盘空格键，点击鼠标左键，即可根据需要移动画布。

33 采云间（DPC）

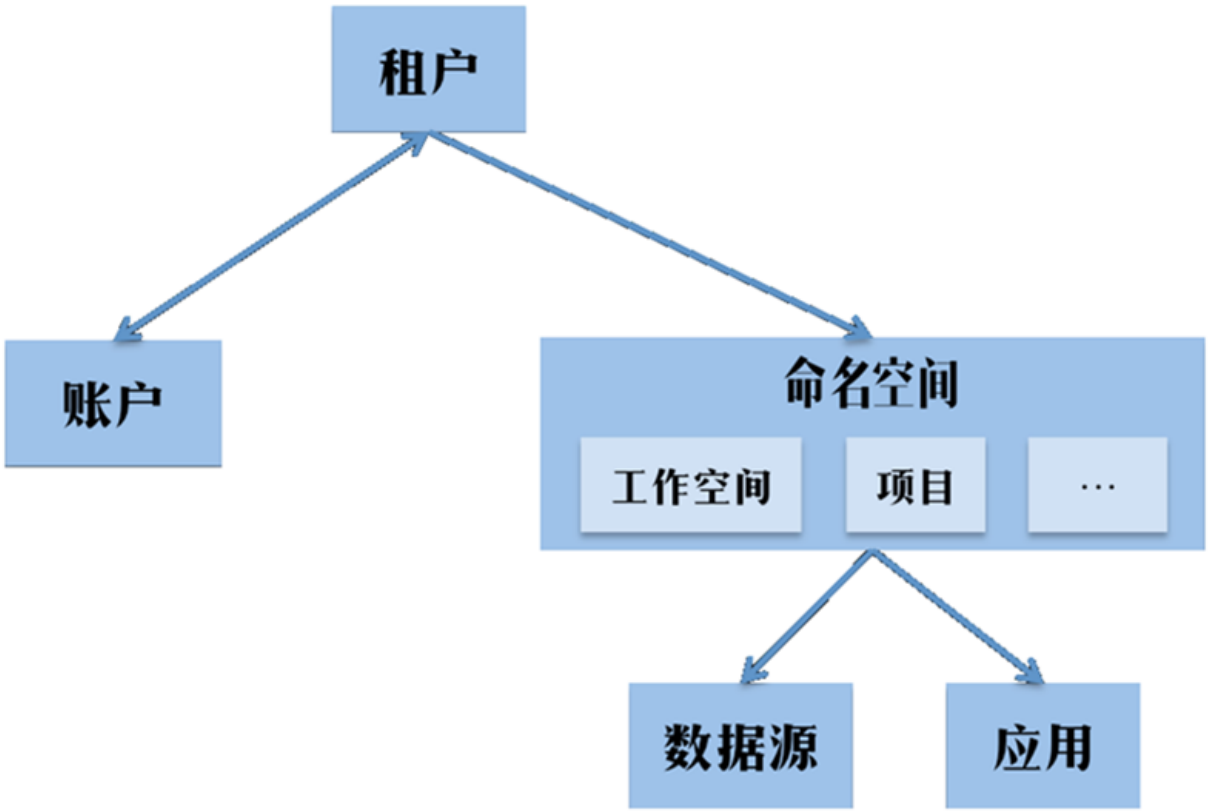
33.1 系统管理

33.1.1 系统管理概述

系统管理用于对整个大数据应用进行配置。

如图 *1510: 大数据应用*所示，大数据的应用有如下几个核心的概念。

图 1510: 大数据应用



其中：

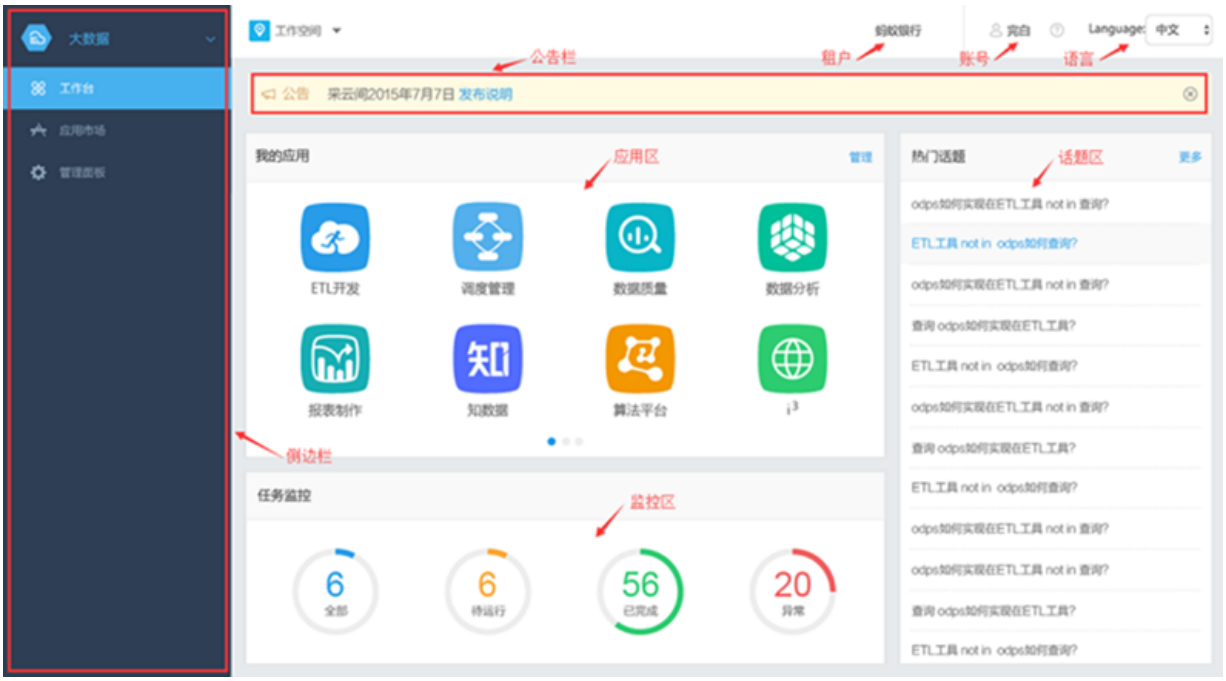
- 租户：和金融云中枢的**工作空间**对应；
- 工作空间：和金融云中枢**工作空间**下的**环境**一一对应
- 数据源：用户使用大数据要分析的数据来源；
- 应用：对应大数据这边的一个个工具，例如，ETL开发服务和任务调度服务。

因此系统管理模块会有对应的**租户管理员**的配置，以及每个**工作空间**的管理员和数据源配置。

33.1.2 大数据首页

在大数据首页的顶端，从左至右依次展示了当前用户的工作空间、租户、账号、语言信息，如图 *1511: 大数据首页*所示。

图 1511: 大数据首页



侧边栏

侧边栏为导航栏，包括工作台、应用市场和管理面板三个主菜单，单击某个主菜单可显示其子菜单。

公告栏

公告栏展示系统最新公告。

可以通过单击右侧的关闭按钮关闭公告栏。

应用区

应用区内展示当前用户的应用。

单击某个应用的图标，即可进入该应用的详情页面。

单击应用区右上角的**管理**，可对应用进行添加、删除等操作。

当应用数量过多，不能排列在一页时，应用图标会自动分页显示，可单击应用区下方页码图标进行翻页。

监控区

监控区展示任务监控的仪表盘，包括全部、待运行、已完成和异常四种任务的数据统计，且不同状态以不同颜色展示。

话题区

话题区展示论坛中的热门话题。

单击某个话题链接，可进入该话题的页面。

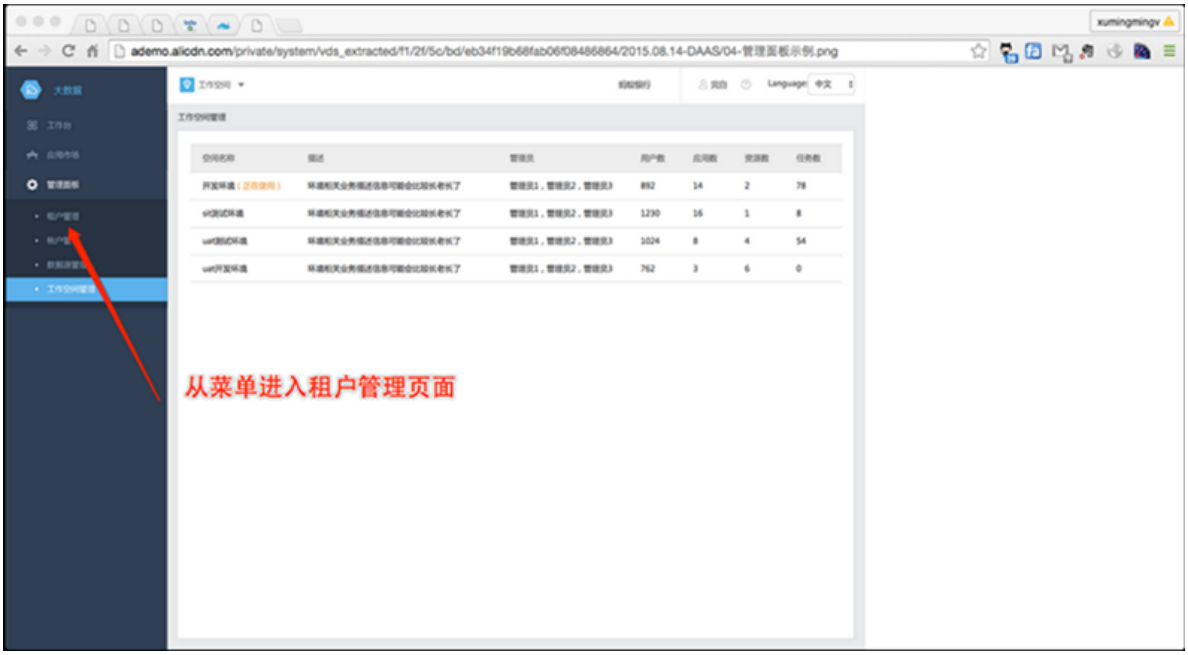
单击话题区右上角的**更多**，可进入论坛首页。

33.1.3 管理租户

操作步骤

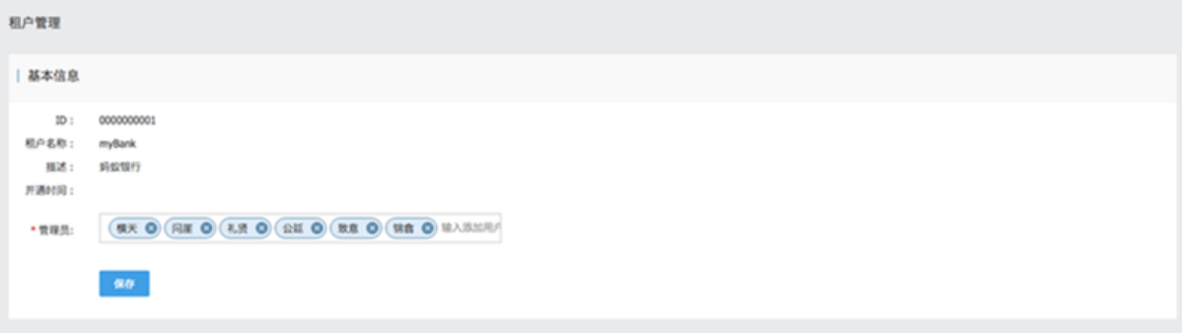
1. 在大数据左侧导航栏中，选择**管理面板** > **租户管理**，如图 *1512:* 租户管理所示。

图 1512: 租户管理



2. 在**租户管理**窗口，单击**编辑**，如图 *1513:* 设置管理员所示。

图 1513: 设置管理员



您可以进行如下操作：

- 设置管理员

  添加用户，设置整个租户在大数据产品下的租户管理员。

- 删除管理员

  单击某个用户名上的删除按钮，删除该管理员。当前操作的用户可以将自己从管理员列表中删除。

  注意: 不能删除全部管理员，应至少保留一个。

3. 确认修改完成后，单击**保存**。

33.1.4 管理工作空间

33.1.4.1 设置管理员

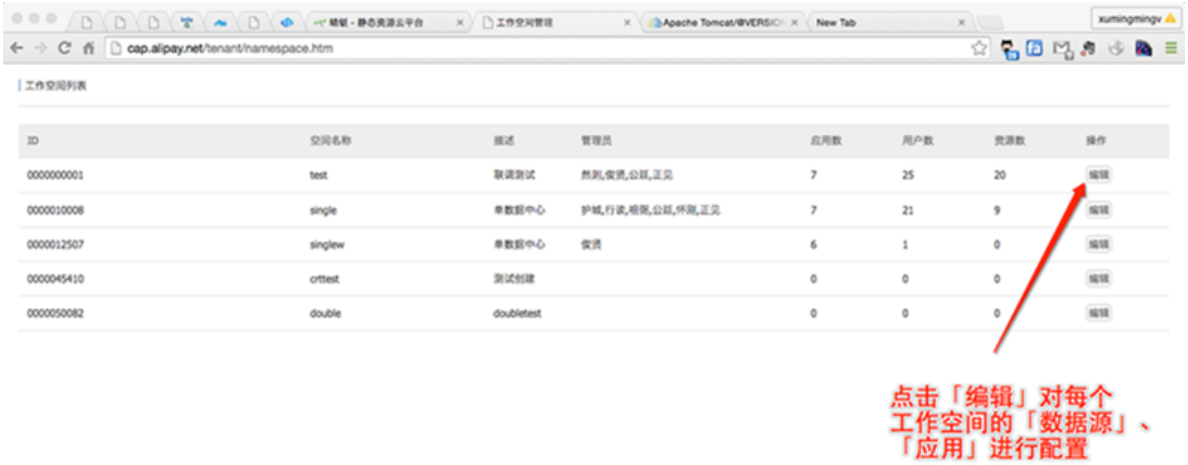
背景信息

大数据的租户管理员可以对工作空间的管理员名单进行设置。

操作步骤

1. 在大数据左侧导航栏中，选择**管理面板** > **工作空间管理**。
2. 在**工作空间列表**中，单击某个工作空间右边的**编辑**，如图 *1514:* 工作空间列表所示。

图 1514: 工作空间列表



进入编辑页面。

默认进入基础信息标签页，可以对管理员进行添加和删除。

当前操作的用户可以将自己从管理员列表中删除。

注意: 不能删除全部管理员，应至少保留一个。

3. 修改完成后，单击**保存**。

33.1.4.2 管理用户

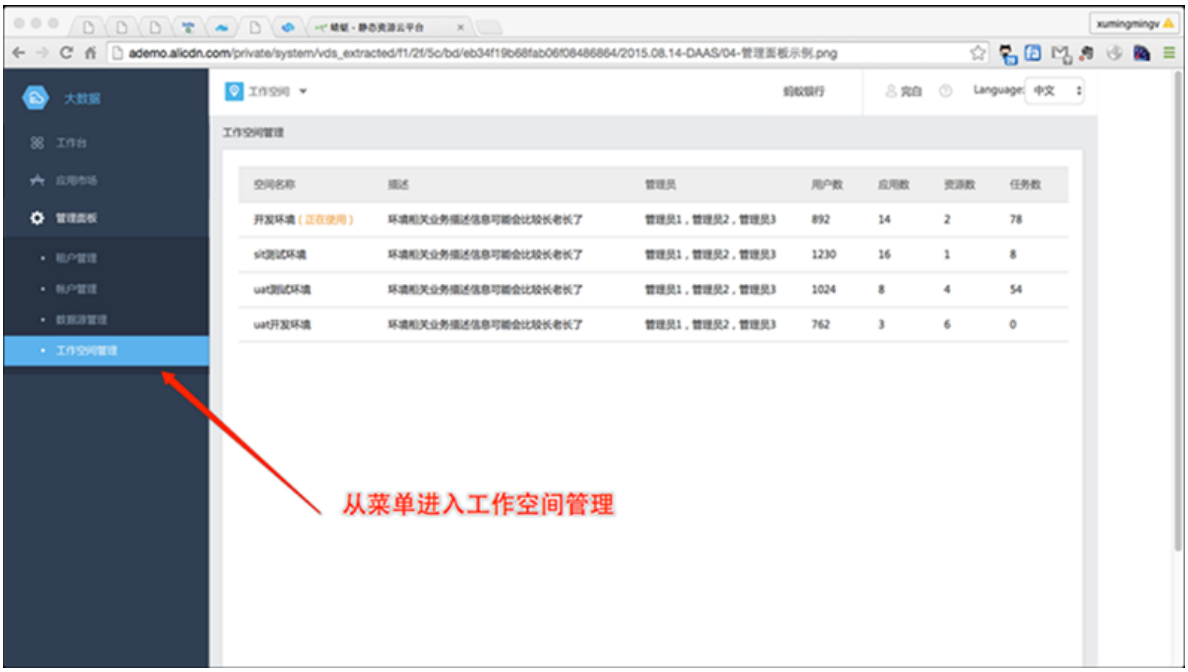
背景信息

在大数据应用里，每个用户可以访问的工作空间列表是不一样的，管理员可以对当前租户下的用户进行管理。

操作步骤

1. 在大数据左侧导航栏中，选择**管理面板** > **工作空间管理**，如图 *1515:* 工作空间管理所示。

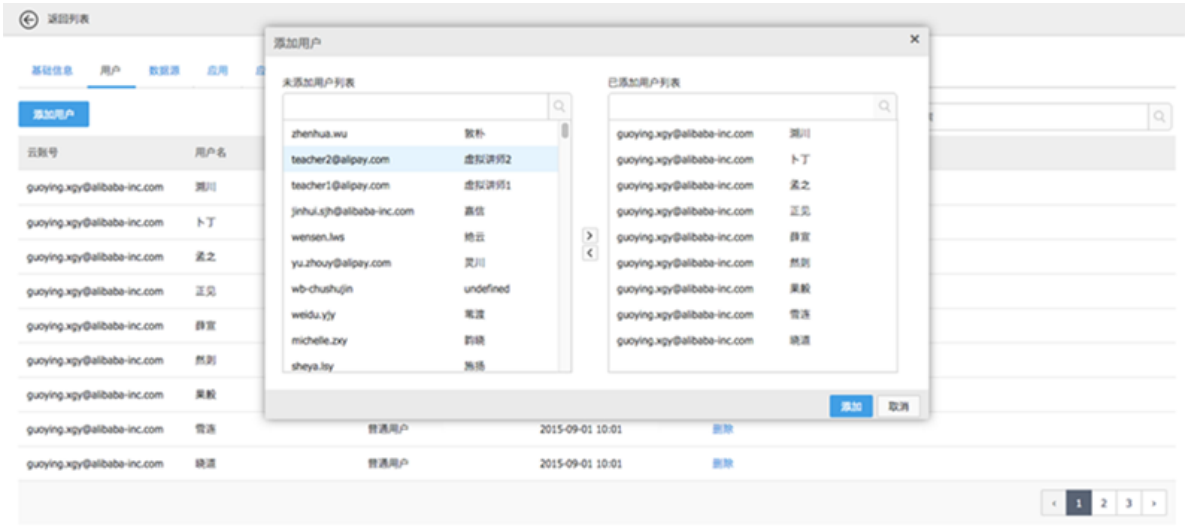
图 1515: 工作空间管理



2. 在对应工作空间的右侧，单击**编辑**，进入工作空间编辑页面。
3. 选择**用户**页签后，进入用户管理页面。
4. 单击**添加用户**。

   系统会弹出一个窗口，弹出窗口的左侧列表中列出了当前租户下的所有用户，可以通过名字对用户进行搜索，如图 *1516:* 用户页签所示。

   图 1516: 用户页签



5. 通过名字搜索用户后，单击**>**。

   用户即出现在右侧已添加用户列表中。

如果要删除某个用户，在右侧已添加用户列表中选中该用户，单击**<**，即可取消对该用户的添加。

6. 确认添加列表无误后，单击**添加**，即可把用户添加到当前的工作空间。

   添加用户成功后，可以单击用户右侧的**删除**，收回对该用户的授权。

33.1.4.3 管理数据源

背景信息

在每个工作空间里可以使用的数据源是相互独立的，一个工作空间的数据源可供当前工作空间内的所有应用使用。

管理员可以对每个工作空间的数据源列表进行配置和管理。

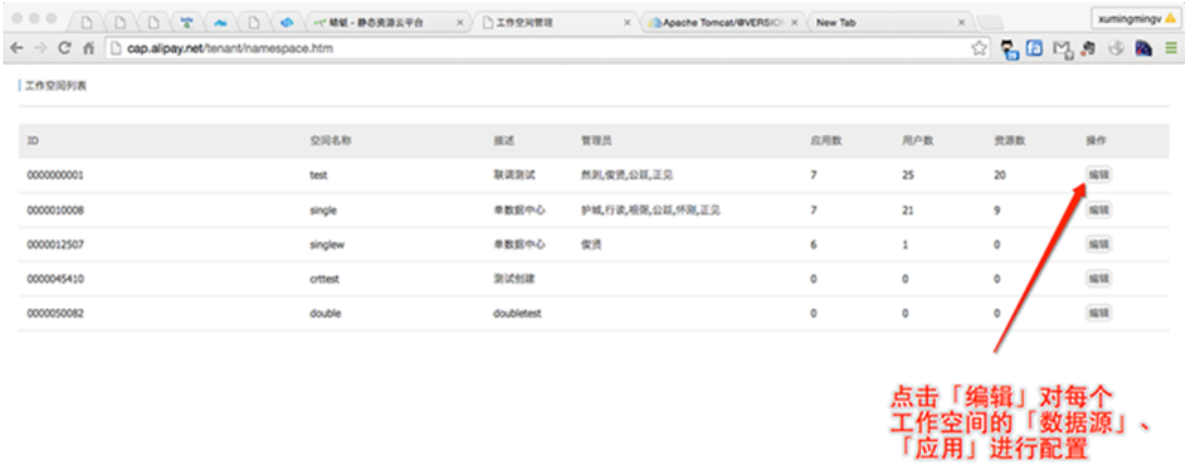
目前我们支持如下数据源：

- MaxCompute
- RDS
- OSS
- Table Store
- OceanBase

操作步骤

1. 在大数据左侧导航栏中，选择**管理面板** > **工作空间管理**。
2. 在**工作空间列表**中，单击某个工作空间右边的**编辑**，如图 *1517:* 工作空间列表所示。

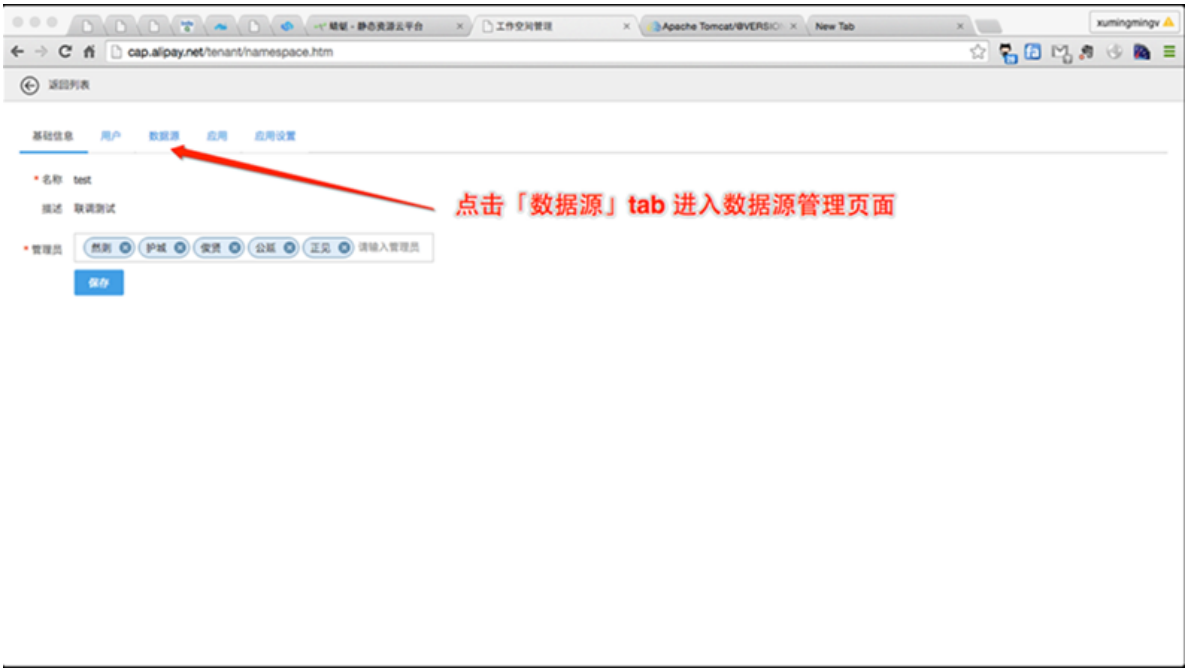
   图 1517: 工作空间列表



3. 选择**数据源**页签，进入数据源管理页面，如图 *1518: 数据源页签*所示。

   数据源列表中展示了数据源的连接名称、类型、实例ID 、数据库等信息。

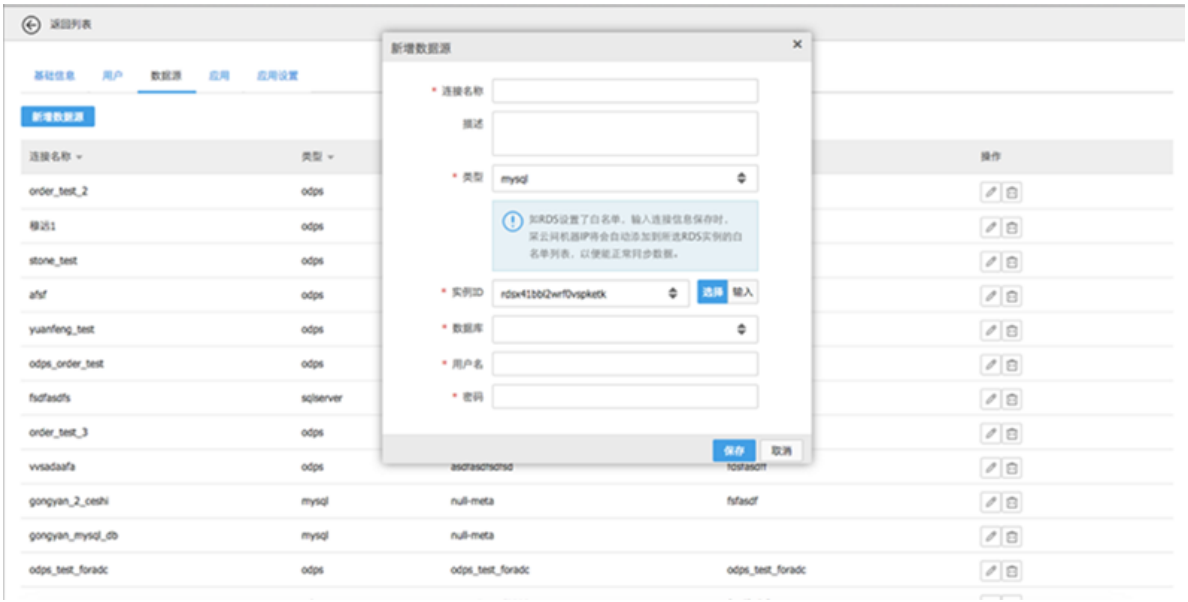
   图 1518: 数据源页签



4. 单击**新增数据源**。

弹出如图 *1519:* 新增数据源所示对话框。

图 1519: 新增数据源



5. 依次输入连接名称、描述、类型、实例 ID 、数据库、用户名、密码等信息，单击**保存**，完成配置。

   其中，**实例 ID** 可以由选择模式提供已经存在的实例 ID ，也可以由输入模式手动输入实例 IP 、用户名、密码等信息。

   对于在项目中不再需要的数据源，可以单击数据源列表右侧的**删除**，删除数据源。

33.1.4.4 管理应用

背景信息

一个应用是指用户在大数据平台里面可以使用的一个完整功能，目前我们提供如下应用：

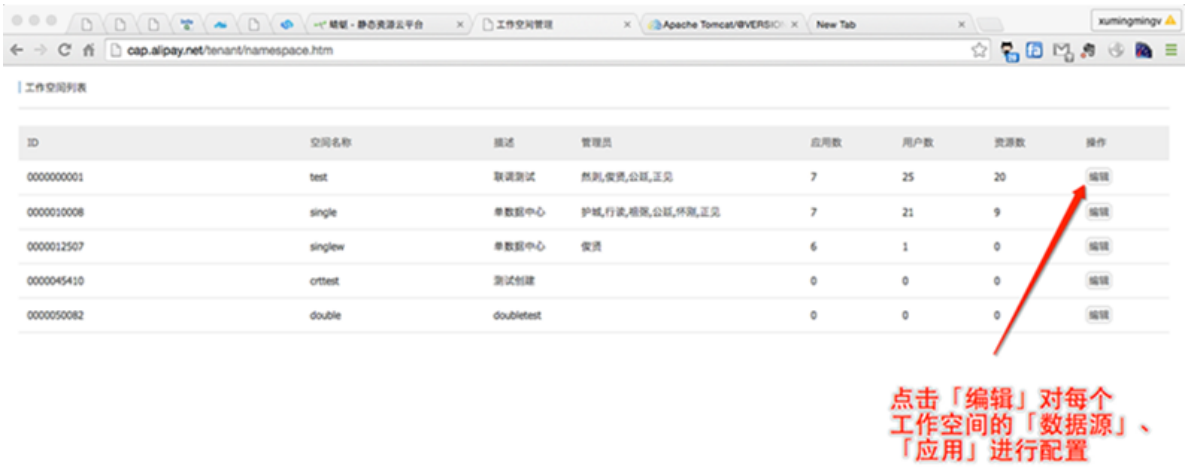
- ETL 开发
- 任务管理
- 数据分析
- 报表制作
- 数据挖掘

操作步骤

1. 在大数据左侧导航栏中，选择**管理面板** > **工作空间管理**。

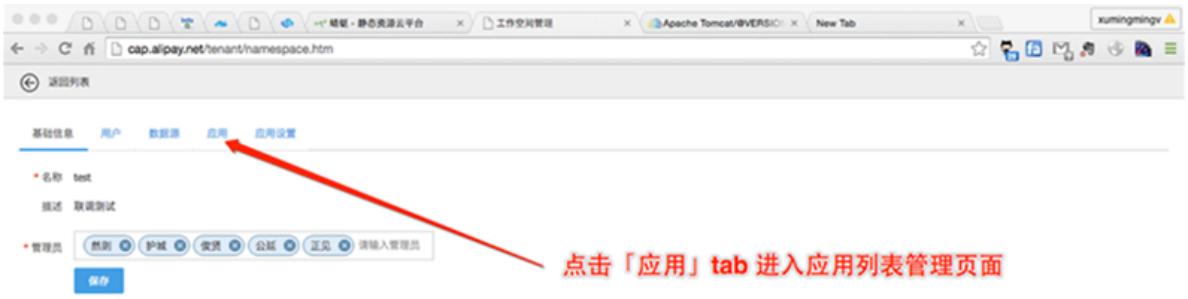
2. 在**工作空间列表**中，单击某个工作空间右边的**编辑**，如图 *1520:* 工作空间列表所示。

图 1520: 工作空间列表



3. 选择**应用**页签，进入应用列表管理页面，如图 *1521:* 应用页签所示。

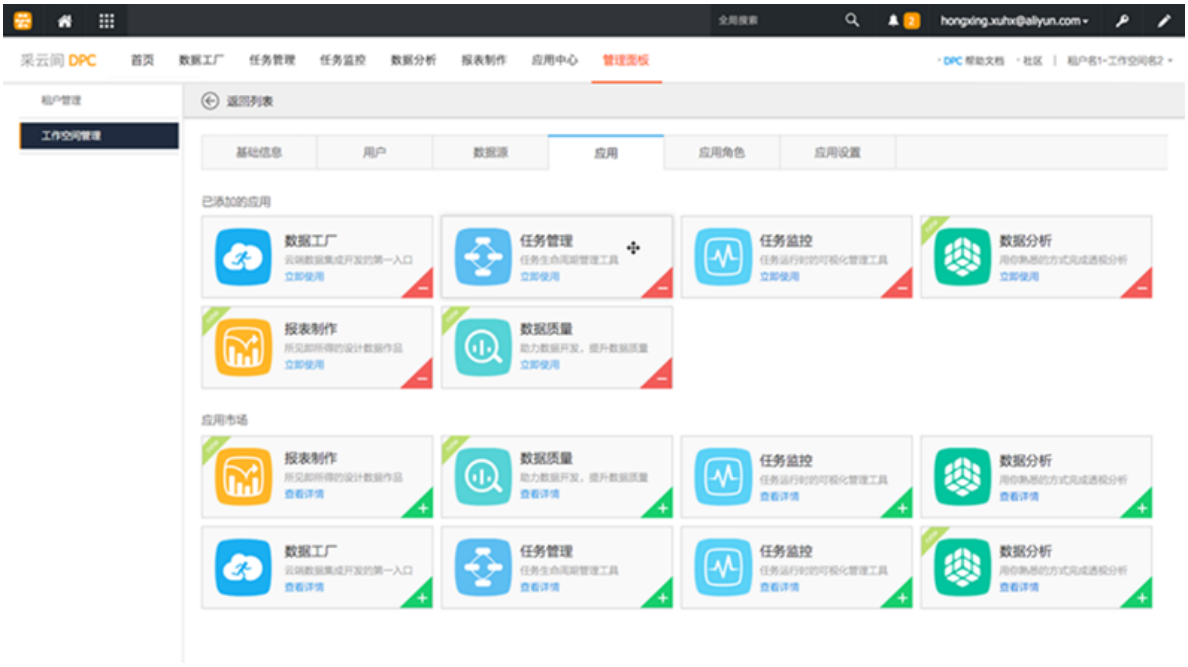
图 1521: 应用页签



管理员在应用管理页面对每个工作空间的应用进行管理。

如图 *1522:* 应用管理所示，应用管理页面分为**已添加的应用**和**应用市场**两个区域。

图 1522: 应用管理



您可以进行如下操作：

- 删除应用

  在**已添加的应用**区域中，单击应用卡片右下角的红色减号，可从当前工作空间中删除该应用。

  删除后的应用卡片会移动到**应用市场**区域。

- 添加应用

  在**应用市场**区域中，单击应用卡片右下角的绿色加号，可以将该应用添加到当前工作空间。

  添加后的应用卡片会移动到**已添加的应用**区域。

33.2 数据工厂

33.2.1 ETL 开发服务简介

ETL 开发服务 ETLDS（ETL Development Service），提供基于 MaxCompute 平台的 ETL 开发的编程环境。

ETL 设计与开发是大数据的 DW 和 BI 建设的关键环节之一，负责系统数据的生成，将数据在 EDW 数据架构的层次之间进行加工传输。

功能特性

具备如下功能特性：

- 提供个人和团队以 SQL 等方式进行 ETL 脚本开发。
- 方便的 SQL 执行和调试功能，可以对执行结果集进行各类操作，满足灵活查询的需要。
- 以文件树的方式记录、编辑和管理代码脚本，支持文件加锁和版本管理。
- 支持 UDF 函数上传，支持数据分析师上传自己的数据进行关联查询。

ETL 的组成

ETL 通常在包括以下几项内容：

- 数据抽取
- 数据加载
- 数据清洗和转换
- 数据任务调度

其中，数据抽取、数据加载和数据任务调度在任务调度服务简介章节中详细介绍。

数据清洗和转换在大数据中，是通过脚本来实现的，脚本的类型包括：SQL。

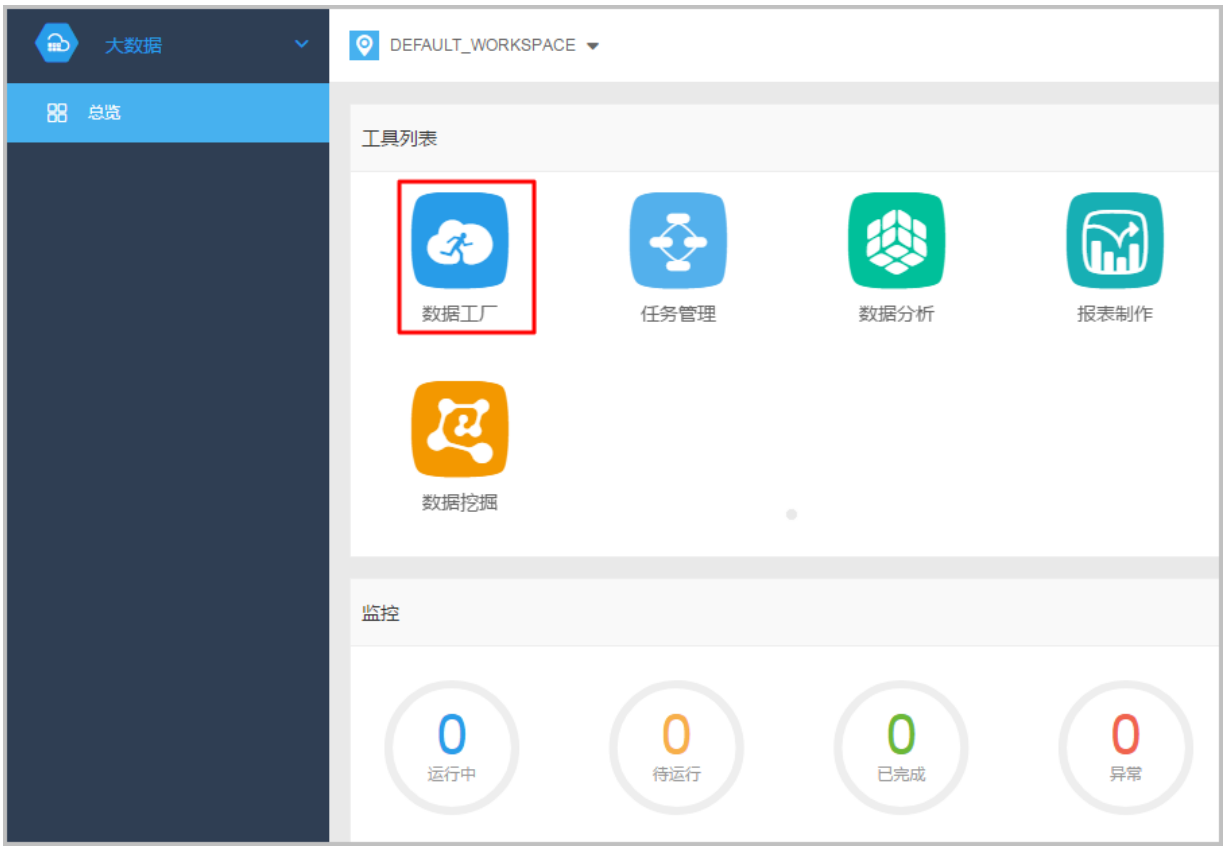
开发者通过开发脚本程序实现复杂的 ETL 过程。

ETL 开发模块（又叫数据工厂）是用于开发 ETL 脚本的工具。

33.2.2 进入 ETL 开发

进入采云间控制台，单击右边**工具列表**中的**数据工厂**，如图 *1523:* 数据工厂所示。

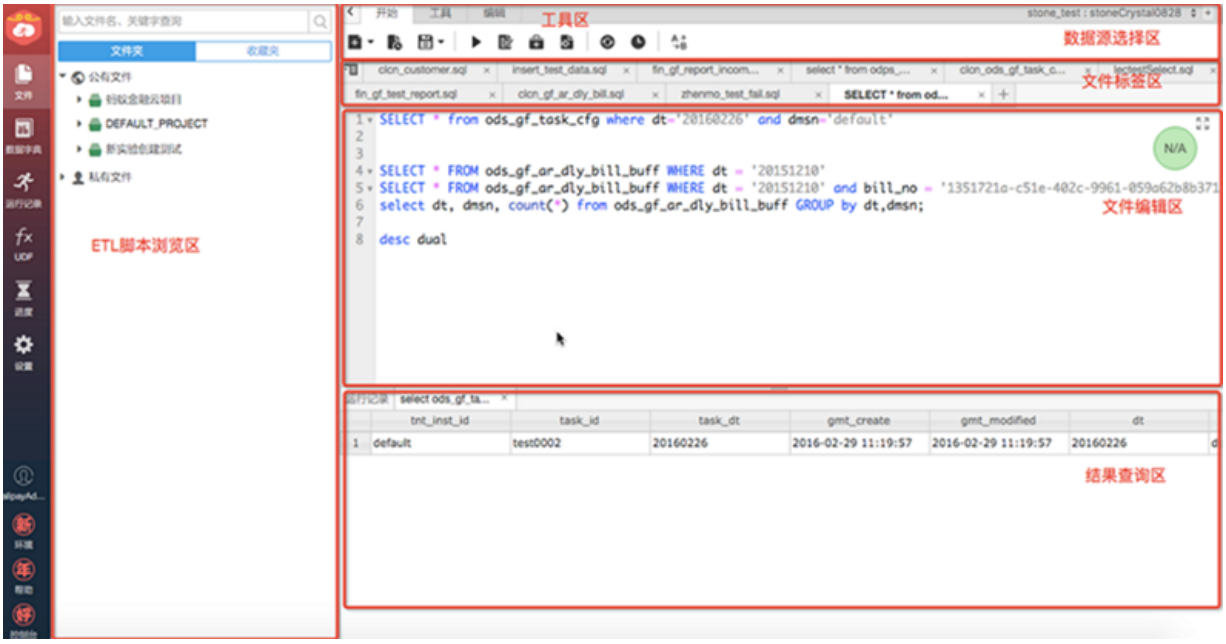
图 1523: 数据工厂



33.2.3 ETL 开发界面介绍

ETL 开发界面如图 *1524: ETL 开发界面*所示。

图 1524: ETL 开发界面



侧边栏

侧边栏提供导航功能，可以点击相应按钮进入文件浏览、数据字典、运行记录、函数和设置等功能。

浏览区

浏览区展示的内容随侧边栏中选择的功能进行变化，对于文件管理、数据字典、运行记录和自定义函数，展示内容分别如下：

- 文件管理：对文件和文件目录进行创建、编辑、查看、删除等管理；
- 数据字典：对表做查询、分组等管理；
- 运行记录：执行 SQL 语句的历史记录；
- 自定义函数：对函数进行查询、删除等管理。
-

在浏览窗口，可以通过两种方式进行交互操作：鼠标右键和拖拽，通过鼠标右键可以进行创建、详情查看等操作，通过拖拽可以进行移动。

通过搜索框，输入关键词可以进行查找操作。

工具栏

数据工厂的工具栏包含新建、保存、执行等命令。另外，该工具栏还包含**美化**、**左缩进**、**右缩进**等常见SQL功能。

工具栏有助于快速访问常用工具和操作。

编辑区

数据工厂的编辑区与常见文字编辑软件的功能类似。

在编辑区，工具栏中的功能命令都可以使用。

通过执行以下操作之一，可以新建文档：

- 选择**文件** > **新建 SQL 文件**。
- 单击工具栏上的**新建 SQL**图标。
- 单击工具栏的**+**，创建新的工作文档。

通过执行以下操作之一，可以打开现有工作文档：

- 选择**文件** > **最近打开的文件**，择文件并打开。

- 双击或右键打开浏览窗口**我的文件**里的文件。

日志区

日志区展现当前执行语句的 Log，包含以下三块内容：

- 当前运行的 SQL 语句；
- 执行的日志；
- 执行的状态。

状态栏

状态栏位于界面的底部，显示执行情况的信息。例如，图 *1524: ETL 开发界面*的状态栏显示该 SQL 总共执行了 4 秒，状态是失败。

33.2.4 管理文件

33.2.4.1 公有文件和私有文件的区别

浏览窗口中的两个根目录分别为公有文件和私有文件，如图 *1525: 浏览器窗口*所示。

图 1525: 浏览器窗口



公有文件是租户内部所有人都可以看到和编辑的文件；私有文件是只有自己才能看到和编辑的文件。

说明: 私有文件可以移动到公有文件夹中，但是公有文件不能移动到私有文件夹中。

33.2.4.2 新建文件

数据工厂中支持 SQL、Shell、Perl 三种脚本的编写和执行，可以从工具栏和浏览窗口新建各类文件。默认支持SQL文件，具体支持的其他文件类型，视现场安装版本而定。

33.2.4.2.1 SQL 文件

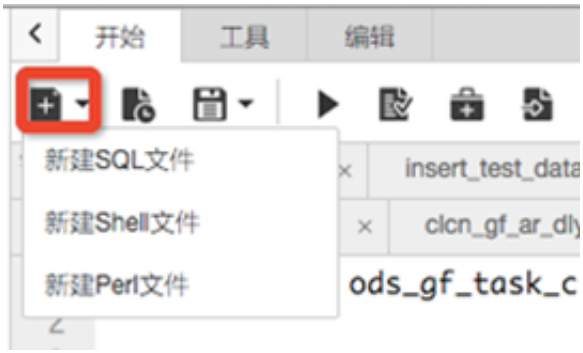
背景信息

支持三种方式新建SQL 文件。

- 通过工具栏新建

  在工具栏中，选择**新建** > **新建 SQL 文件**，并在弹出的窗口中填写文件名，如图 *1526: 在工具栏上新建*所示。

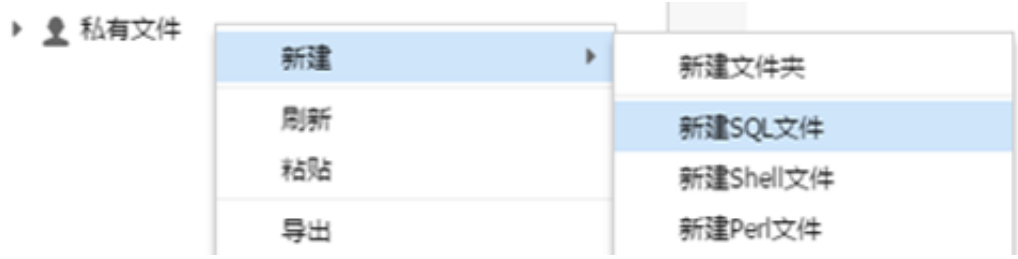
  图 1526: 在工具栏上新建



- 在浏览窗口新建

  在浏览窗口选择目标位置，单击鼠标右键，选择**新建** > **新建 SQL 文件**，并在弹出的窗口中填写文件名，如图 *1527: 在浏览窗口新建*所示。

  图 1527: 在浏览窗口新建



- 在标签页上新建

  在标签页上，单击**+**，自动新建 SQL 文件，此时 SQL 文件并未指定位置，需在保存时选择目标位置并填写文件名，如图 *1528: 在标签页上新建*所示。

  图 1528: 在标签页上新建



33.2.4.2.2 其他脚本文件

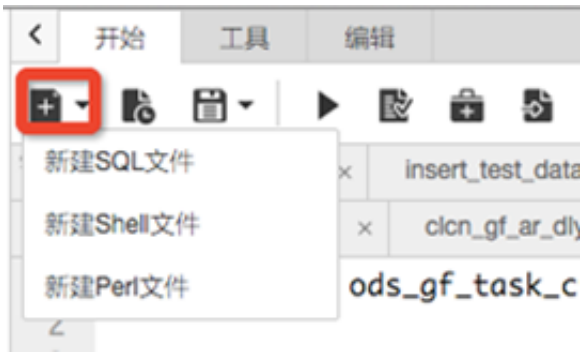
背景信息

支持两种方式新建其他脚本文件。

- 通过工具栏新建

  在工具栏中，选择**新建 > 新建 Shell 文件**或**新建 > 新建 Perl 文件**，并在弹出的窗口中填写文件名，如图 *1529:* 在工具栏上新建所示。

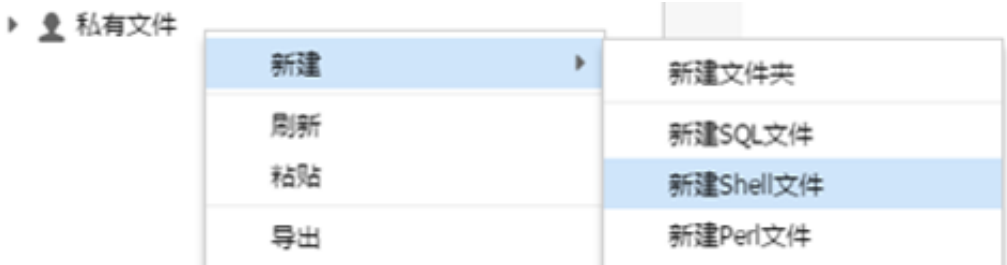
  图 1529: 在工具栏上新建



- 在浏览窗口新建

  在浏览窗口选择目标位置，单击鼠标右键，选择**新建 > 新建 Shell 文件**或**新建 > 新建 Perl 文件**，并在弹出的窗口中填写文件名，如图 *1530:* 在浏览窗口新建所示。

  图 1530: 在浏览窗口新建



33.2.4.3 编辑公有文件

打开公有文件夹中的文件，在编辑区中进行编辑。

在编辑公有文件前，操作者需对文件加锁，否则无法编辑文件。

当前用户未加锁或已有他人加锁的文件，编辑区会弹出**没有文件编辑权限**，**请先解锁文件**的提示。

完成编辑后，需及时解锁文件，以便他人使用。

33.2.4.4 编辑私有文件

打开私有文件夹中的文件，在编辑区中进行编辑，如图 *1531:* 编辑私有文件所示。

图 1531: 编辑私有文件



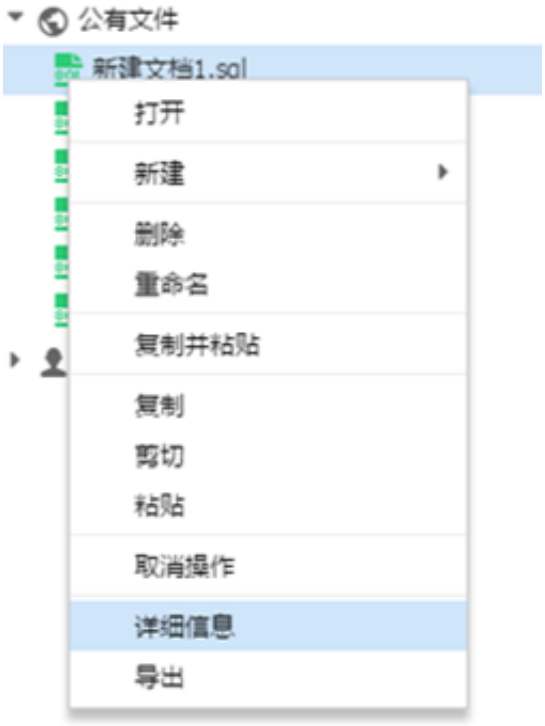
私有文件无加锁解锁概念，当前用户可以随时编辑。

33.2.4.5 查看文件基本信息

操作步骤

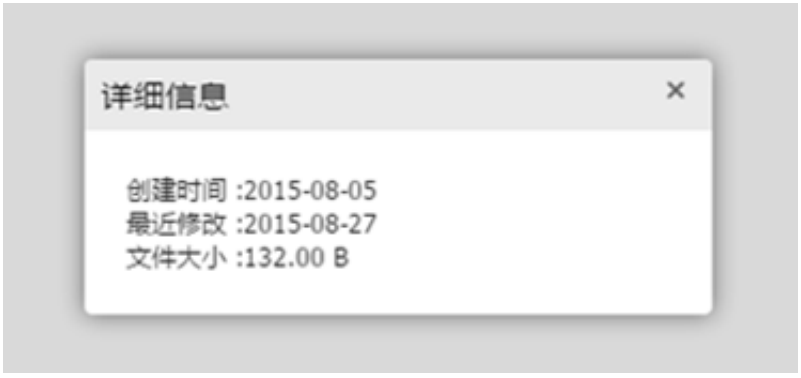
1. 在浏览器窗口中，选中目标文件，单击鼠标右键，选择**详细信息**，如图 *1532:* 右键菜单所示。

图 1532: 右键菜单



2. 在弹出的窗口，查看文件的详细信息，如图 *1533:* 详细信息所示。

图 1533: 详细信息



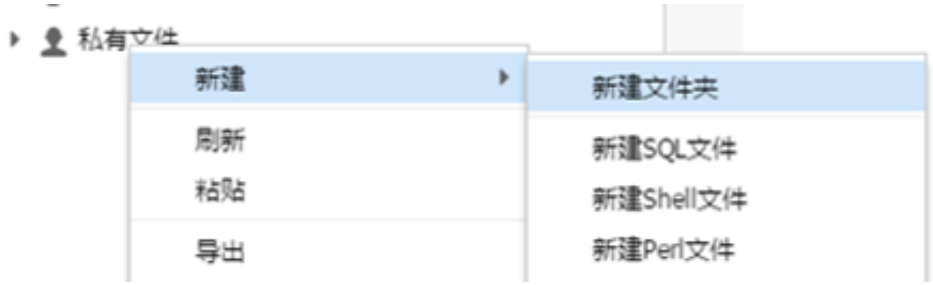
33.2.4.6 管理文件夹

33.2.4.6.1 新建文件夹

操作步骤

1. 在浏览窗口中，选择目标位置，单击鼠标右键，选择**新建** > **新建文件夹**，如图 *1534:* 新建文件夹所示。

图 1534: 新建文件夹



2. 在弹出的窗口中，填写文件夹名。

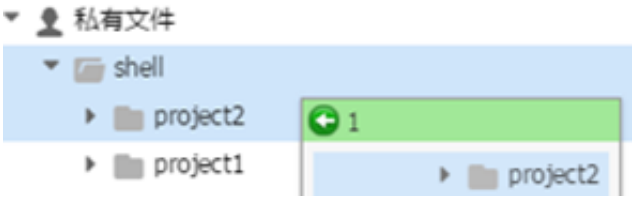
33.2.4.6.2 移动文件夹

选中要移动的文件夹，拖动文件夹至目标位置。

拖动时，鼠标右侧会显示文件夹的可移动状态：

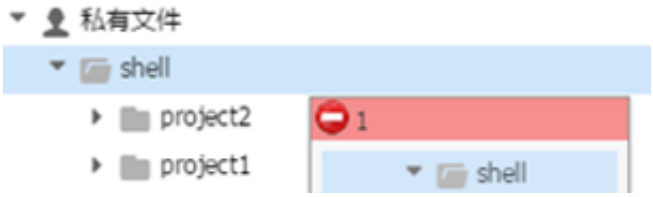
- 绿色表示可以移动到当前位置，如图 *1535: 可移动*所示。

图 1535: 可移动



- 红色表示不可以移动到当前位置，如图 *1536: 不可移动*所示。

图 1536: 不可移动

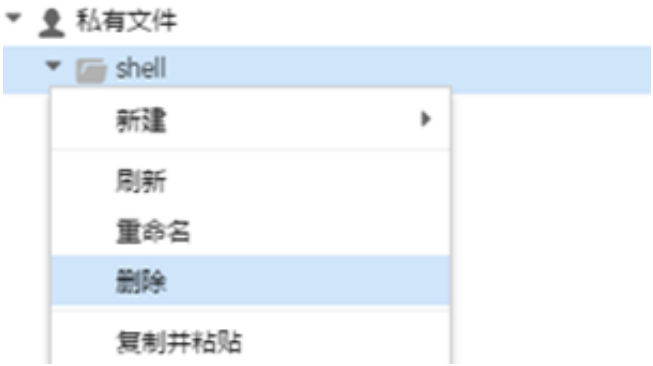


33.2.4.6.3 删除文件夹

操作步骤

1. 右键单击准备删除的文件夹，选择**删除**，如图 *1537: 删除文件夹*所示。

图 1537: 删除文件夹



2. 在弹出的对话框中，单击**确定**。

33.2.4.7 复制文件

操作步骤

1. 右键单击要复制的文件，选择**复制**。
2. 在目标位置，单击右键选择**粘贴**。

   或者您也可以右键单击要复制的文件，选择**复制并粘贴**。

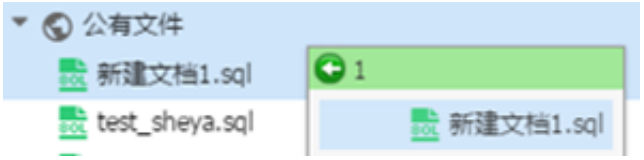
33.2.4.8 移动文件

选中要移动的文件，拖动文件至目标位置。

拖动时，鼠标右侧会显示文件的可移动状态：

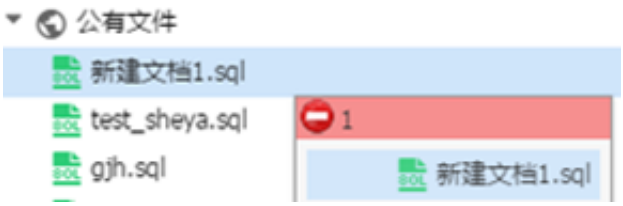
- 绿色表示可以移动到当前位置，如图 *1538:* 可移动所示。

  图 1538: 可移动



- 红色表示不可以移动到当前位置，如图 *1539:* 不可移动所示。

  图 1539: 不可移动



33.2.4.9 删除文件

操作步骤

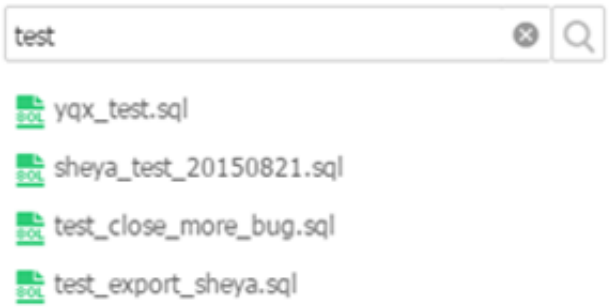
1. 右键单击准备删除的文件，选择**删除**。
2. 在弹出的对话框中，单击**确定**。

33.2.4.10 搜索文件

操作步骤

1. 在浏览窗口的搜索框中输入文件名等关键字，如图 *1540:* 搜索文件所示。

   图 1540: 搜索文件

2. 搜索。

   如果云中提供 OpenSearch 服务，则是按照全文搜索，如果未提供 OpenSearch 则只搜索文件名称。

33.2.4.11 加锁文件

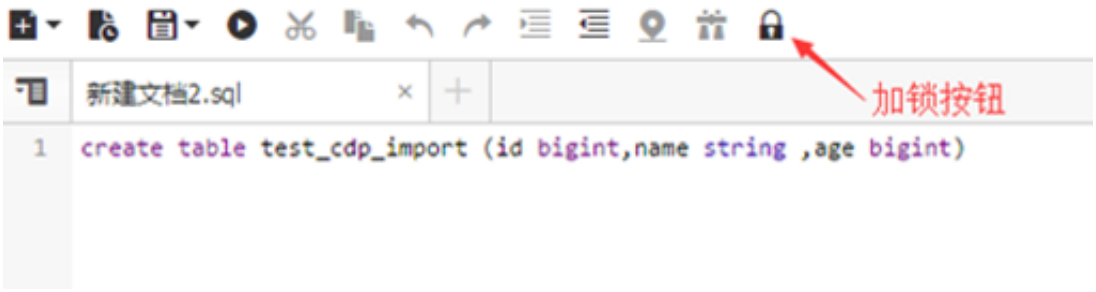
背景信息

对于公有文件夹中的文件，只有经操作者加锁后才允许编辑，因此编辑公有文件前需要进行加锁操作。

操作步骤

1. 打开公有文件夹中未加锁的文件。
2. 在工具栏中，单击**加锁**图标，如图 *1541:* 加锁文件所示。

   图 1541: 加锁文件



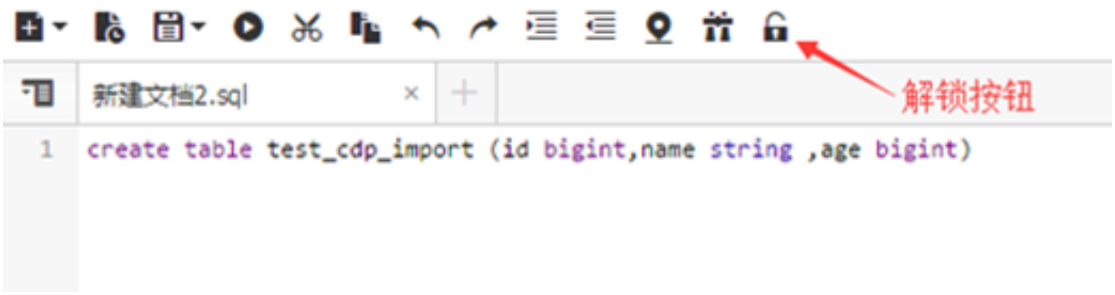
33.2.4.12 解锁文件

对于公有文件夹中由当前用户加锁的文件，经当前用户编辑完成后，为避免资源占用，应当及时进行解锁操作。

操作步骤

1. 打开公有文件夹中已加锁的文件。
2. 在工具栏中，单击**解锁**图标，如图 *1542:* 解锁文件所示。

图 1542: 解锁文件



33.2.4.13 查看文件版本

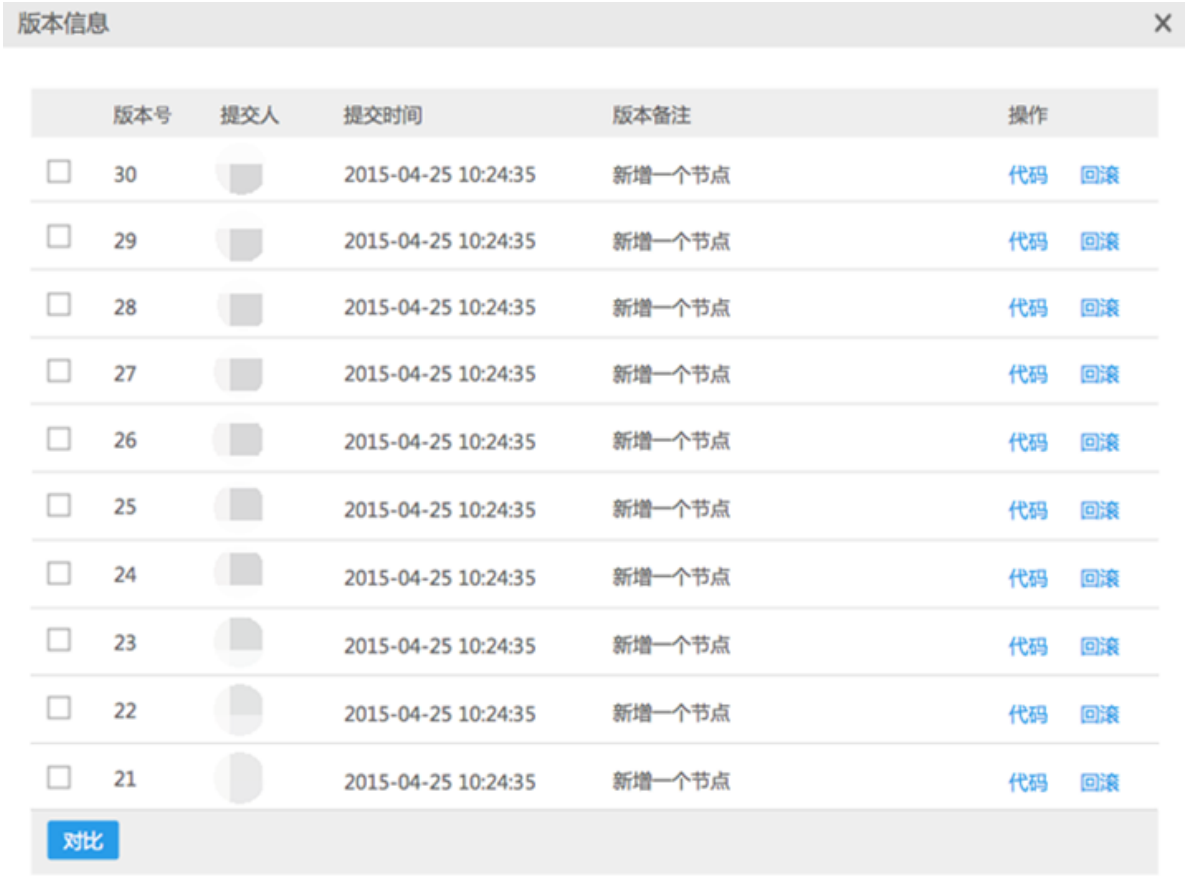
背景信息

公有文件可以多人协同工作，同一个文件可能由于被不同成员或不同时间的多次修改，从而存在多个版本。

操作步骤

1. 选中目标文件，单击右键，选择**代码对比**。
2. 在弹出的窗口中，可以查看文件的版本信息，包括版本号、提交人、提交时间和版本备注等，如图 *1543:* 查看版本信息所示。

图 1543: 查看版本信息



33.2.4.14 版本比对

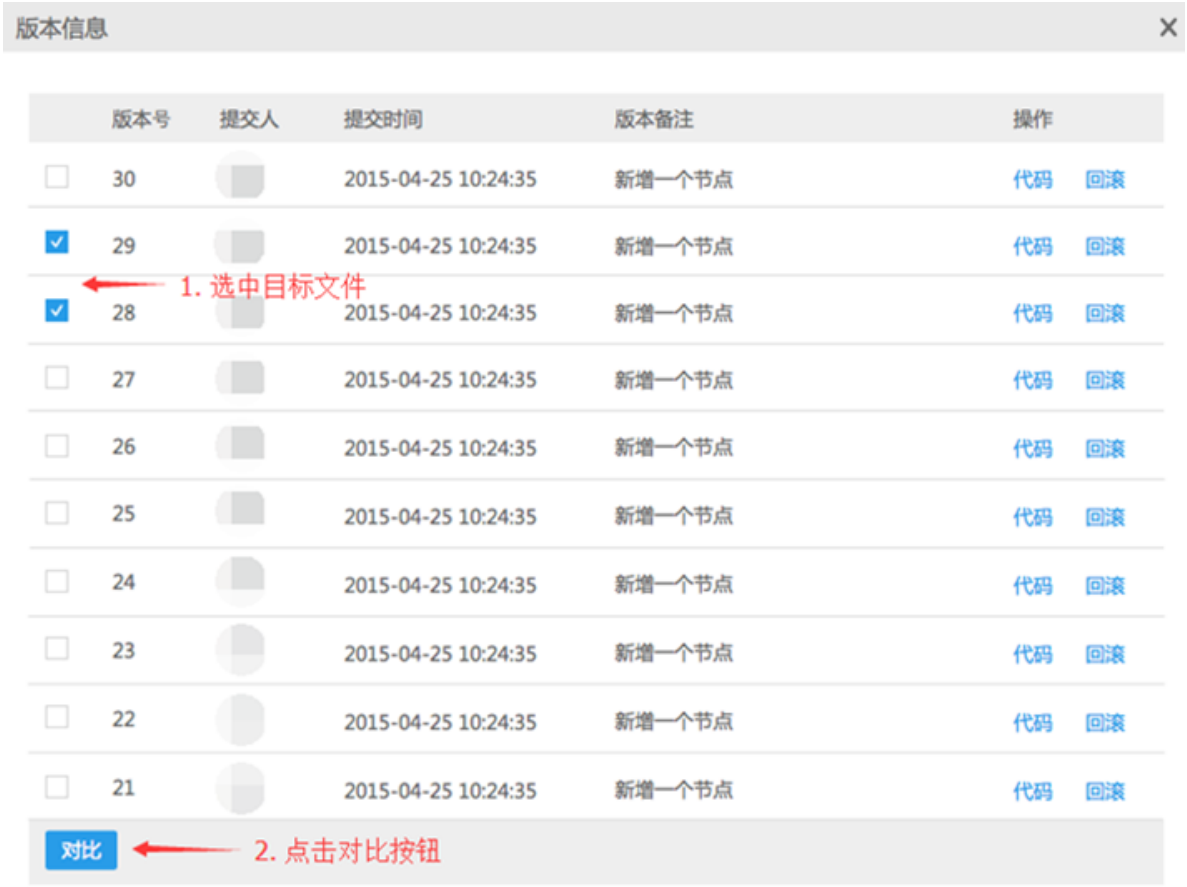
背景信息

目前仅支持两个版本的文件对比。

操作步骤

1. 在版本信息窗口中选择需要对比的文件，如图 *1544:* 选择文件所示。

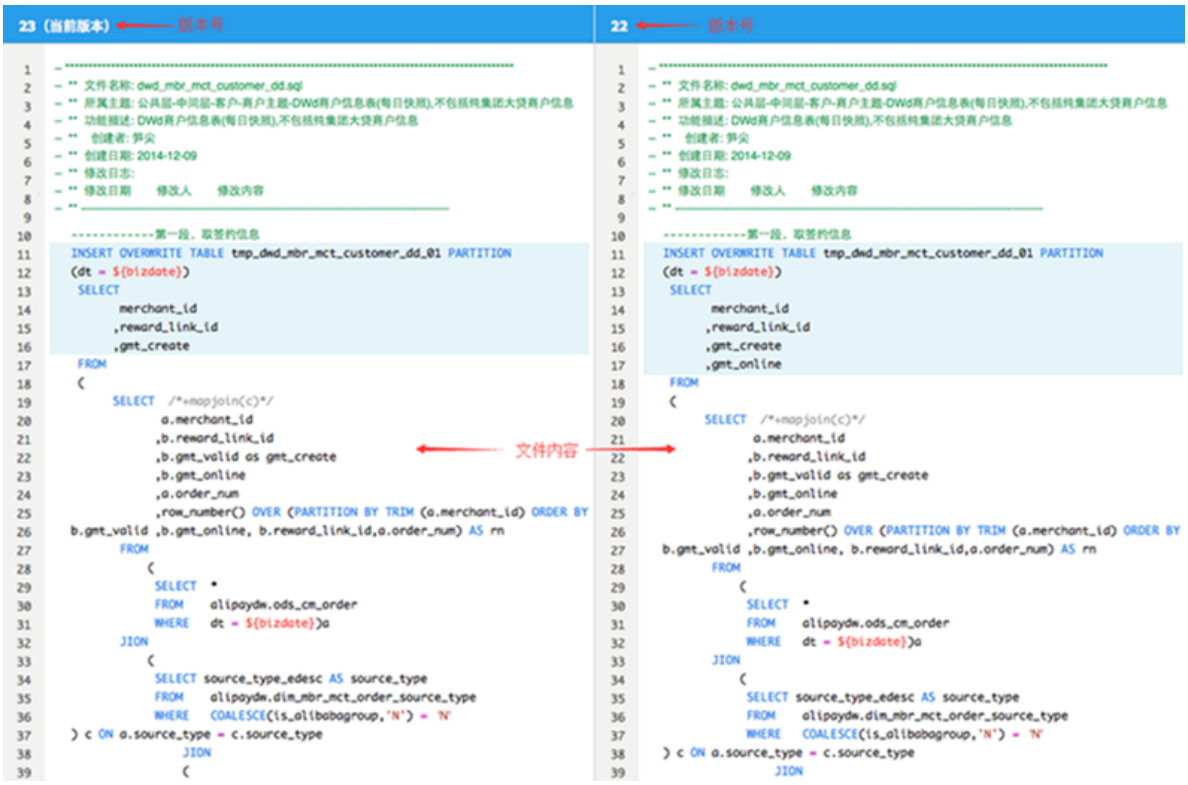
图 1544: 选择文件



2. 单击**对比**。

   进入文件版本对比窗口，如图 *1545:* 文件版本对比窗口所示。

   图 1545: 文件版本对比窗口



33.2.4.15 版本回滚

操作步骤

1. 在版本信息窗口中，找到目标版本。
2. 单击**回滚**，执行版本回滚操作，如图 *1546:* 版本回滚所示。

图 1546: 版本回滚

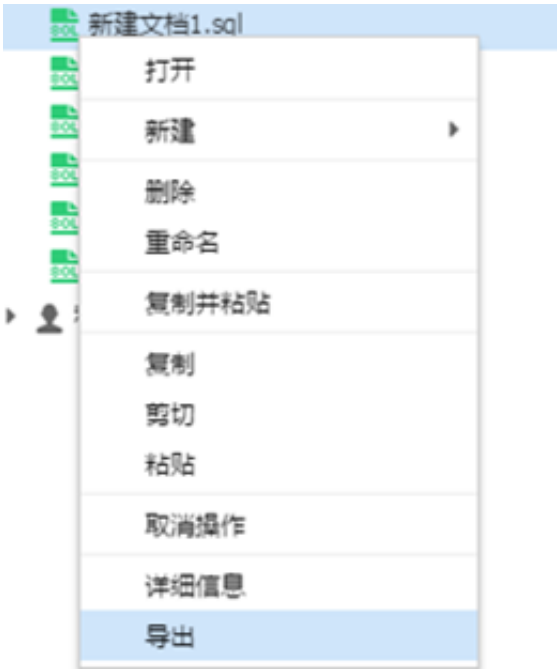


33.2.4.16 导出文件

操作步骤

1. 在浏览窗口中，右键单击要导出的文件，选择**导出**，如图 *1547:* 导出文件所示。

   图 1547: 导出文件



2. 将文件保存至本地。

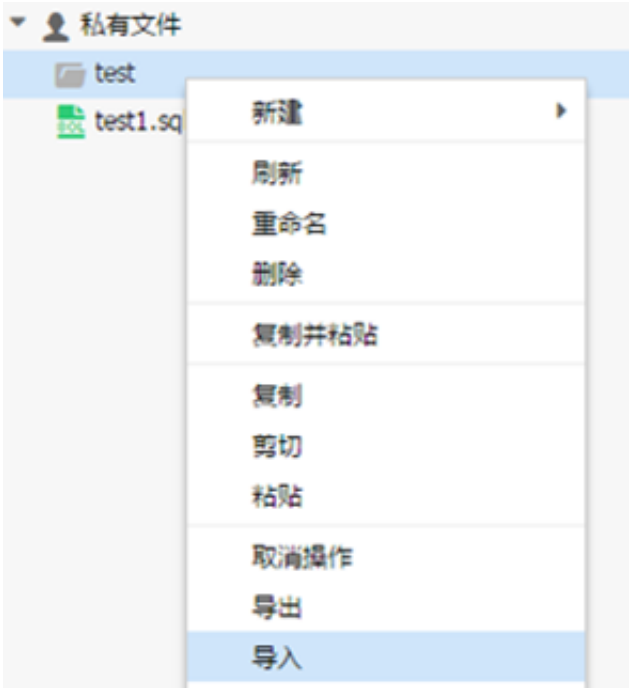
   文件将以 zip 压缩包的形式导出到本地机器。

33.2.4.17 导入文件

操作步骤

1. 右键单击目标文件夹，选择**导入**，如图 *1548:* 导入文件所示。

   图 1548: 导入文件



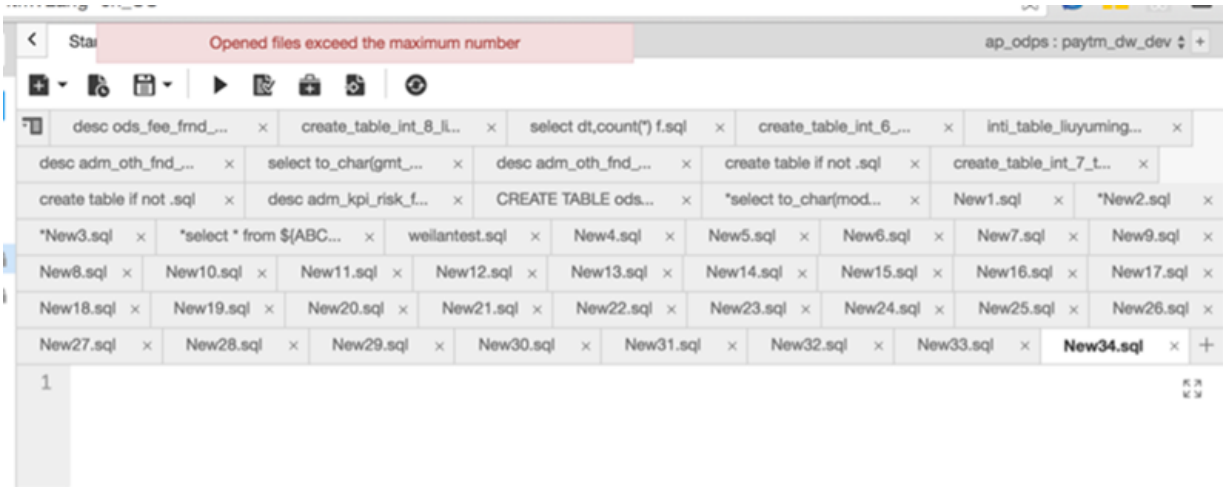
2. 在弹出的窗口中，选择由文件导出功能导出的压缩文件（不支持其他类型的文件），单击**打开**。

   文件即被导入到当前文件夹。

33.2.4.18 打开文件

编辑区最多可以打开 15 个文件，超过该数目，编辑区会弹出**打开的文件超过上限**的提醒，如图 *1549: 最大打开文件数提醒*所示。

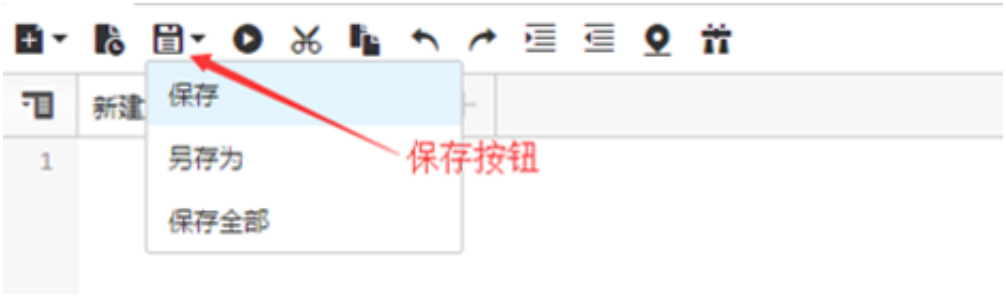
图 1549: 最大打开文件数提醒



33.2.4.19 保存文件

在工具栏中，单击**保存**图标，选择**保存**、**另存为**或**保存全部**，如图 *1550: 保存文件*所示。

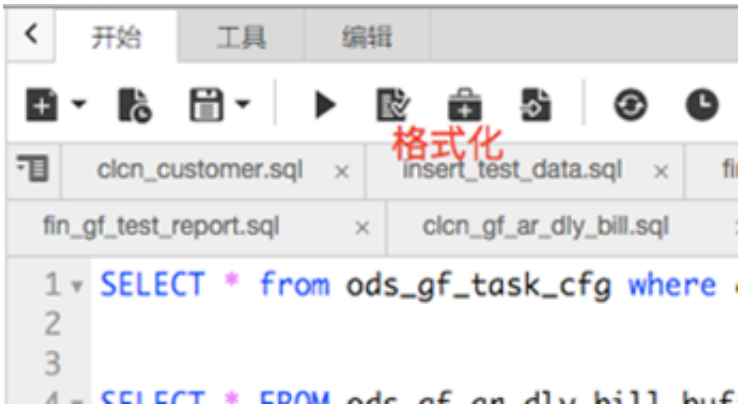
图 1550: 保存文件



33.2.4.20 格式化文件

在工具栏中，单击**格式化**图标，可以格式化当前的SQL语句，关键字会高亮显示，会自动换行，如图 *1551: 格式化文件*所示。

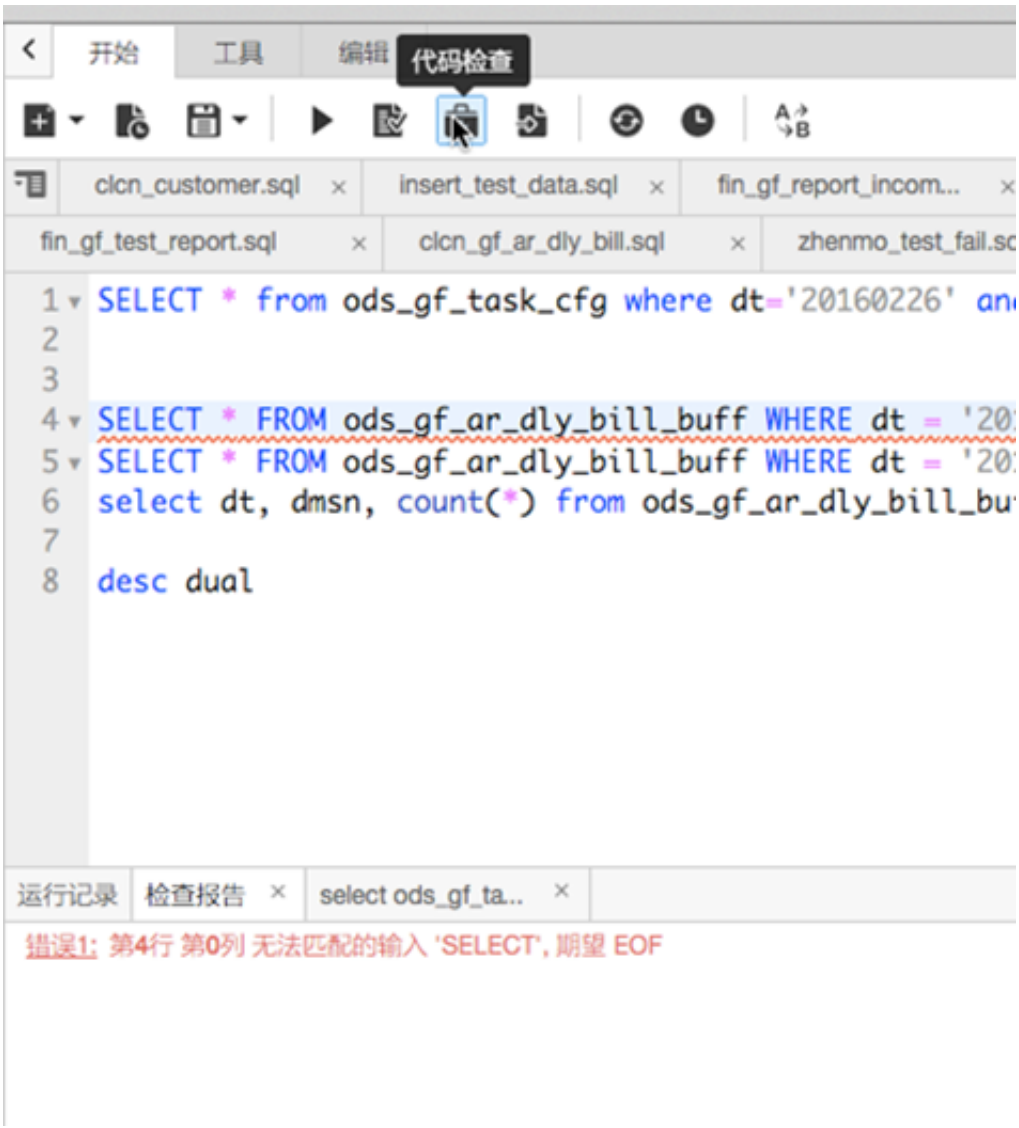
图 1551: 格式化文件



33.2.4.21 语法检查

在工具栏中，单击**语法检查**图标，可以检查当前的SQL语句的语法是否正确，如图 *1552: 语法检查*所示。

图 1552: 语法检查



33.2.4.22 编辑文件

33.2.4.22.1 剪切

选中需要剪切的内容，单击工具栏上的**剪切**按钮或由键盘键入**CTRL+X** ，如图 *1553: 剪切*所示。

图 1553: 剪切



33.2.4.22.2 复制与粘贴

选中需要复制的内容，单击工具栏上的**复制**按钮或由键盘键入**CTRL+C**，如图 *1554: 复制与粘贴*所示。

图 1554: 复制与粘贴



再将光标移动目标位置，由键盘输入**CTRL+V**，完成粘贴。

33.2.4.22.3 撤销与重做

对已执行的操作，可以进行撤销和重做。

在工具栏上，单击**撤销**或**重做**按钮即可，如图 *1555: 撤销与重做*所示。

图 1555: 撤销与重做



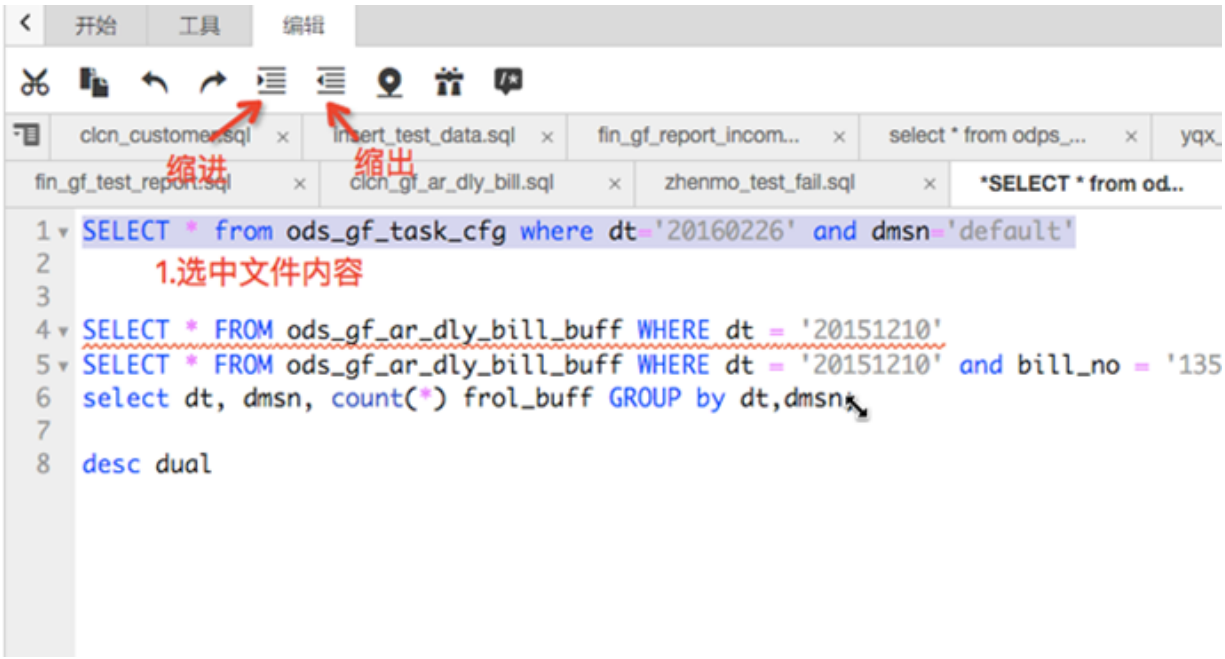
33.2.4.22.4 缩进和取消缩进

为了代码排版美观、便于阅读，可以在数据工厂中对代码进行缩进和取消缩进。

选中需要操作的内容，单击工具栏上的**缩进**按钮或由键盘键入**TAB**进行缩进。

单击工具栏上的**取消缩进**按钮或由键盘键入**SHIFT + TAB**键进行取消缩进，如图 *1556: 缩进与取消缩进*所示。

图 1556: 缩进与取消缩进



33.2.4.22.5 全选

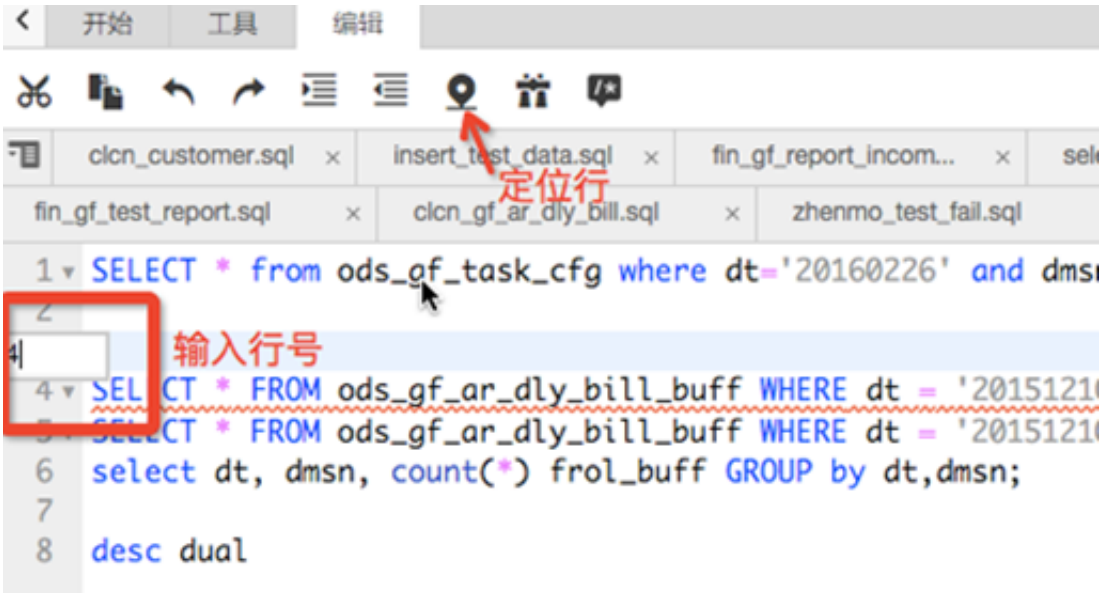
将光标移至编辑区，由键盘键入**CTRL + A**，编辑区内所有文字即被选中。

33.2.4.22.6 转到行

为了快速定位到某一行代码，可以单击工具栏的**转到行**按钮。

在弹出的行号输入框中输入目标行号，单击回车，光标将跳转到输入的行数处，如图 *1557: 转到行*所示。

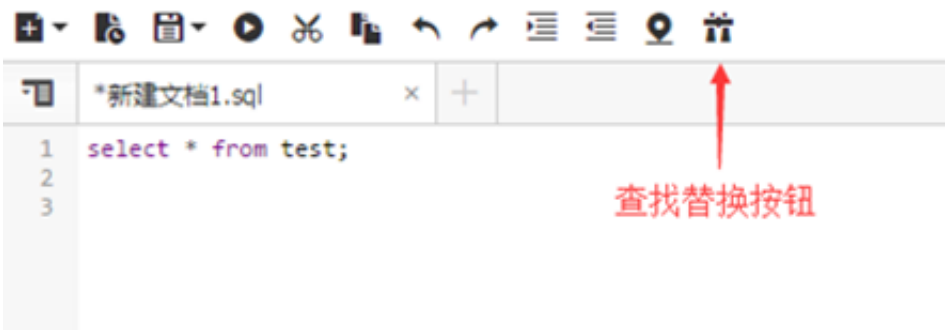
图 1557: 转到行



33.2.4.22.7 查找

单击工具栏的**查找替换**按钮，或由键盘键入**CTRL + F**，调出查找框，如图 *1558: 查找替换按钮*所示。

图 1558: 查找替换按钮



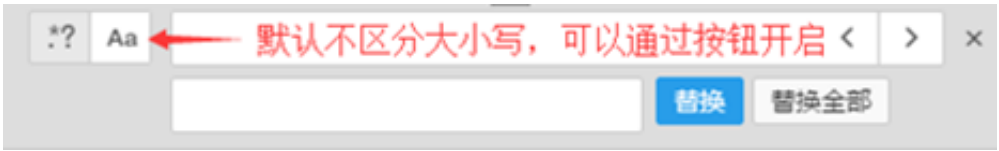
在查找框中输入查找条件，如图 *1559: 输入查找条件*所示。匹配的内容会在编辑区内背景高亮显示。

图 1559: 输入查找条件



默认不区分大小写，可以点击左侧**大小写**按钮进行区分，如图 *1560: 区分大小写*所示。

图 1560: 区分大小写



查找功能支持正则表达式（仅支持 JavaScript 中正则表达式语法），可以单击左侧的**正则**按钮打开和关闭，如图 *1561: 正则表达式开关*所示。

图 1561: 正则表达式开关



33.2.4.22.8 替换

单击工具栏的**查找替换**按钮，在查找框中输入查找条件，在替换框中输入替换内容，如图 *1562: 输入查找条件*所示。

图 1562: 输入查找条件



查找条件同样支持大小写区分和正则表达式。

待完成查找条件和替换内容的填写，单击**替换**或**替换全部**。

33.2.4.22.9 代码注释

数据工厂目前支持一种简易模式的注释，即 --。虽然支持类似 Oracle 的标准注释，形如/* ..... */，但并不推荐，因为/* ..... */ 也为 MaxCompute 的关键字，如图 *1563: 代码注释*所示。

图 1563: 代码注释

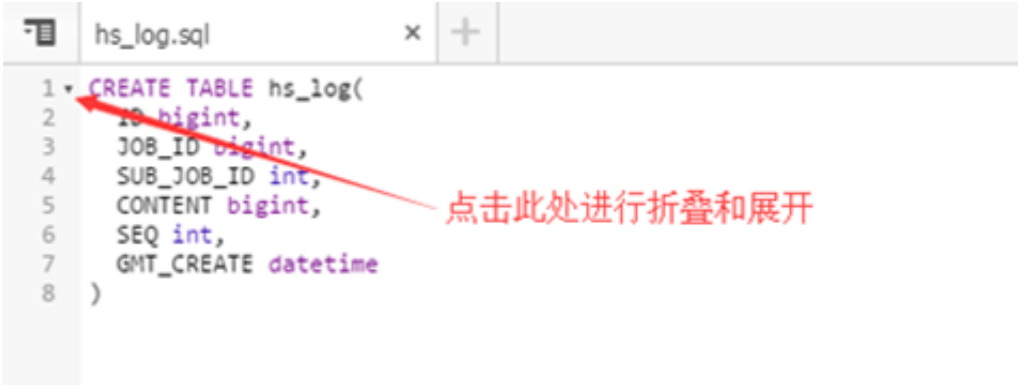


33.2.4.22.10 代码折叠

为方便代码阅读，可以将代码块进行折叠。

单击编辑区左侧的**折叠**或**展开**按钮即可，如图 *1564: 代码折叠*所示。

图 1564: 代码折叠



33.2.4.22.11 全屏展示

单击**全屏切换**按钮，可以全屏化编辑区，方便脚本编写和查阅，如图 *1565:* 全屏展示所示。

图 1565: 全屏展示



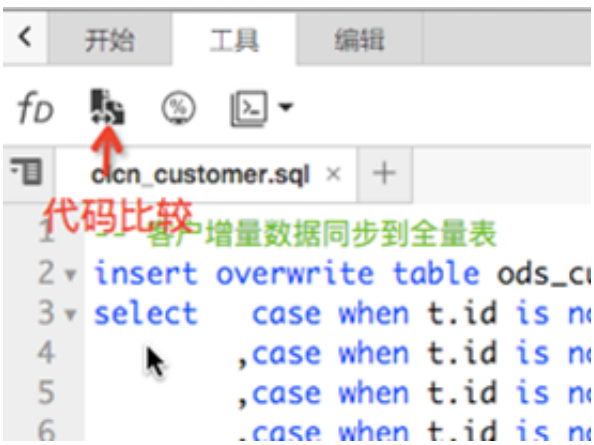
33.2.4.23 文件工具

33.2.4.23.1 文件比较

- 在工具栏上中操作

  在工具栏中，单击**文件比较**图标，可以选择任意两个文件，进行内容比较，如图 *1566:* 工具栏上所示。

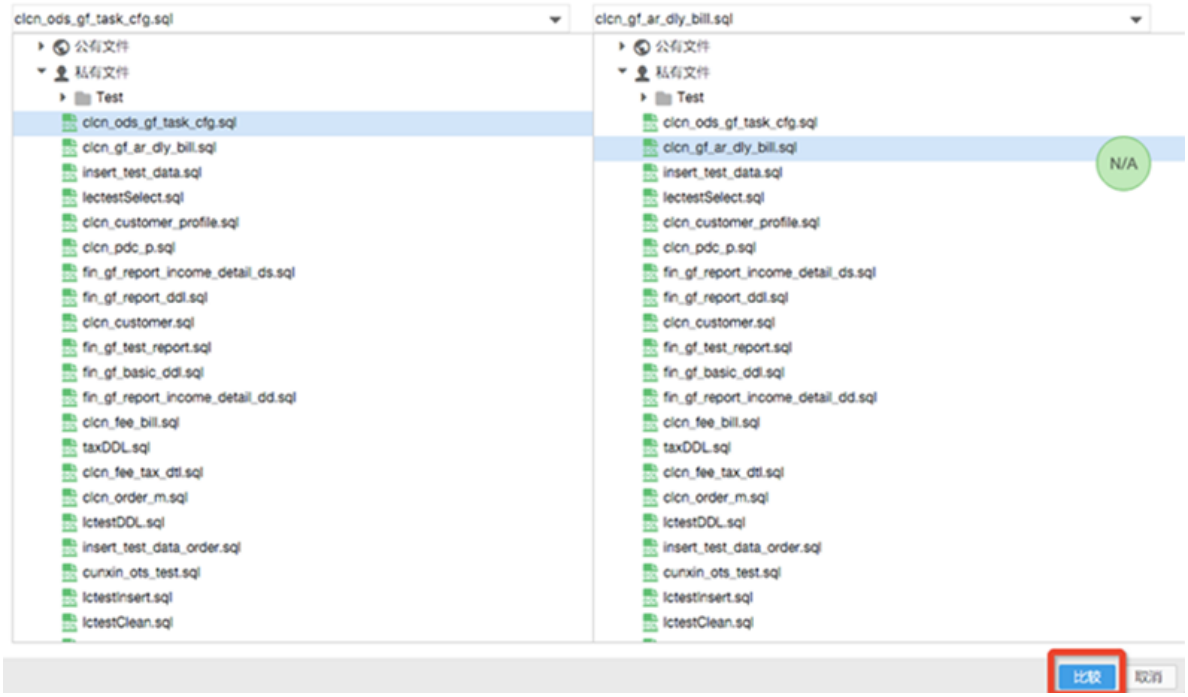
  图 1566: 工具栏上



- 在文件浏览器中操作

  在文件浏览器中，选择要比较的文件，单击**比较**，如图 *1567: 浏览器中*所示。

  图 1567: 浏览器中



33.2.4.23.2 资源球

系统可以计算当前project 的资源占用情况，并将结果以球的形式，漂浮在编辑区。

” 在工具栏中，单击**资源球**图标，可以关闭和展开资源球，如图 *1568: 资源球*所示。

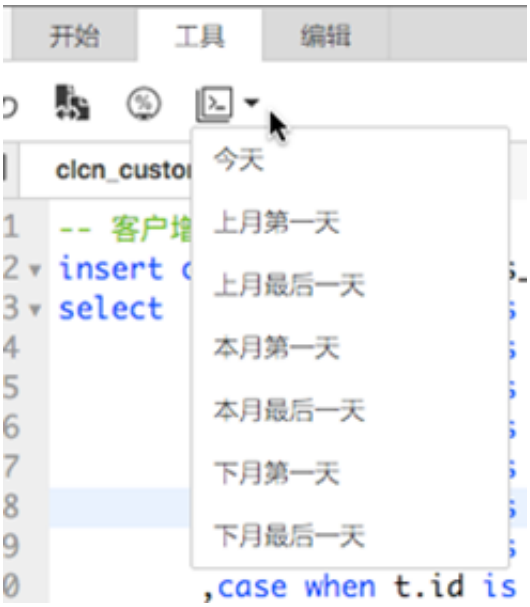
图 1568: 资源球



33.2.4.23.3 时间函数

系统支持常见的时间变量的替换，如图 *1569: 时间函数*所示。

图 1569: 时间函数



33.2.5 提交 SQL 运行

选中需要运行的 SQL 代码行，单击工具栏的**运行**按钮，如图 *1570: 运行 SQL 代码行*所示。

没有选中编辑区中的任何代码时，单击**运行**后，将执行当前文件中的所有代码。

注意: 当前数据工厂虽然支持 Shell 和 Perl 文件的编辑，但不支持这两类文件的运行。

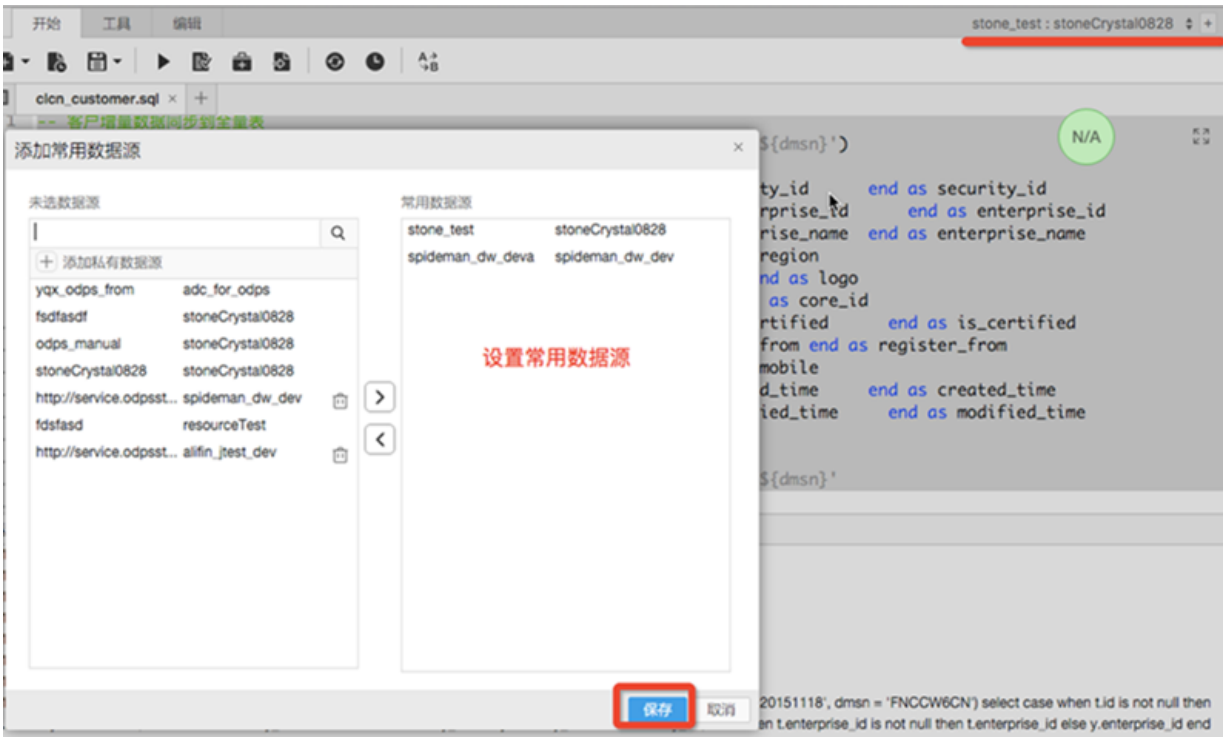
图 1570: 运行 SQL 代码行



33.2.6 设置常用数据源

若当前没有选择 Project ，或选择了错误的 Project ，代码执行将会出错。可以通过工具栏栏选择需要使用的Project，如图 *1571: 设置常用数据源*所示。

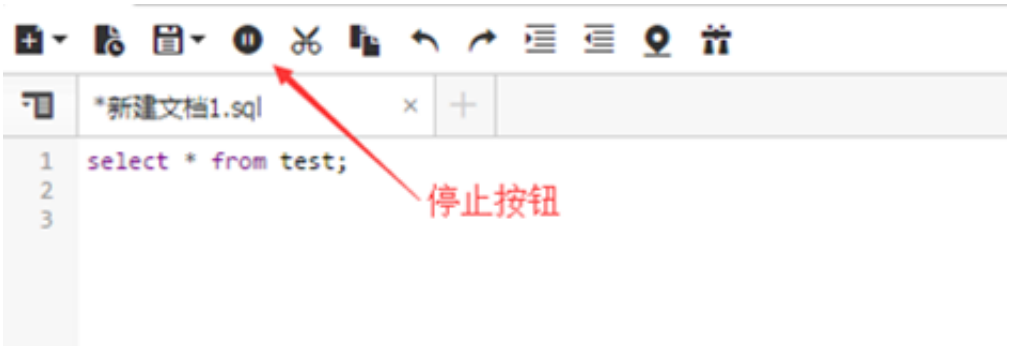
图 1571: 设置常用数据源



33.2.7 终止 SQL 运行

对于已经运行但尚未完成的任务，可以在工具栏单击**停止**按钮，此时处于等待中的任务将不会被执行，如图 *1572: 终止 SQL 运行*所示。

图 1572: 终止 SQL 运行



33.2.8 SQL 变量替换

为减少 SQL 编码，方便执行，可以设置变量替换。

需要替换的变量以 **$** 符号开始，并紧跟 **{}** ，变量名置于花括号中，如图 *1573: 待替换变量*所示。

图 1573: 待替换变量



单击**执行**时，若语句中有变量替换，则需在弹出窗口中填写变量替换的值，如图 *1574:* 替换变量所示。

图 1574: 替换变量



33.2.9 查看运行日志

运行脚本后，日志区将呈现当前执行语句的 Log，包括当前运行的语句、执行的日志以及执行的状态，如图 *1575:* 运行日志所示。

图 1575: 运行日志



33.2.10 查看运行结果

查询语句执行成功后，将会在日志区开启一个新的标签页展示查询结果，如图 *1576:* 运行结果所示。

图 1576: 运行结果



每个文件都会产生与之对应的运行结果标签页，当切换文件标签页时，结果集也将随之切换。

此外，结果集展示的默认数据量可以由设置里的**数据工厂最大显示行数**进行设置（最多不超过 5000 行）。

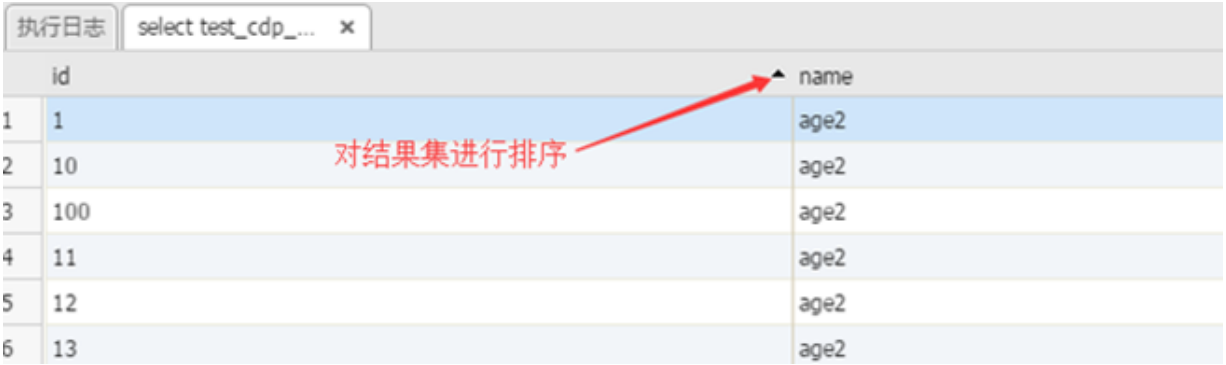
执行的 select 语句将会自动加上limit <数据工厂最大显示行数>，不需自行编写 limit 命令。

33.2.11 进行结果集操作

排序

在查询结果的标签页中，可以对结果集进行排序等操作，如图 *1577:* 排序所示。

图 1577: 排序



导出结果集

可以将结果集导出到本地机器。

在结果集任意位置单击右键，选择**导出**，文件将以 txt 类型保存到本地，如图 *1578: 导出结果集*所示。

图 1578: 导出结果集



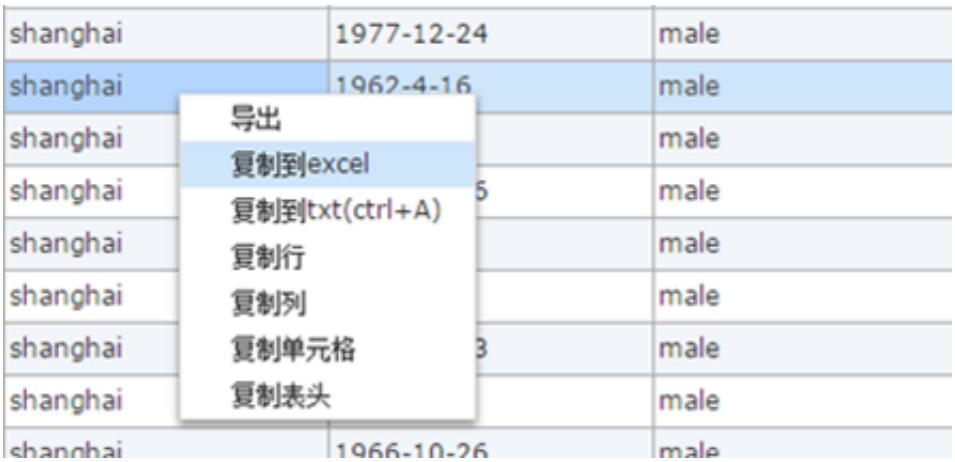
结果集中的数据能否导出可以由**设置**里的**数据允许下载到本地**进行设置。

复制

数据工厂还支持复制到 excel 、复制到 txt 、复制行、复制列、复制单元格、复制表头等多种复制操作，以便使用者将结果集应用到多种目标程序中。

首先在结果集的目标位置单击右键，选择复制类型（以复制到 excel 为例），如图 *1579: 复制到 excel*所示。

图 1579: 复制到 excel



所需内容被高亮以表示选中，按照提示由键盘键入 **CTRL** + **C**，完成所需内容和格式的复制，如图 *1580:* 完成复制所示。

图 1580: 完成复制



33.2.12 管理数据字典

数据工厂提供数据字典功能以便查看Project下表结构或对表进行收藏。

33.2.12.1 选择 Project

背景信息

您可以对不同 Project 中的表进行操作。

操作步骤

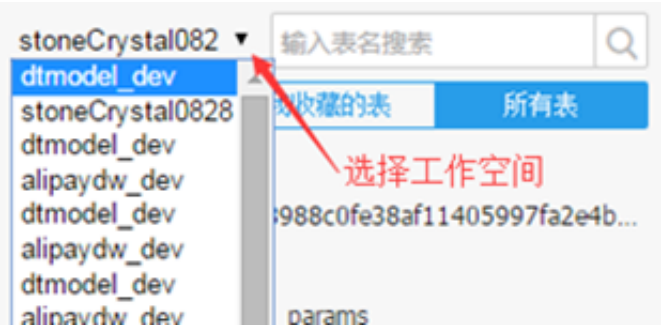
1. 单击左侧导航栏中**数据字典**，显示如图 *1581:* 数据字典界面所示界面。

图 1581: 数据字典界面



2. 在浏览窗口左上角，单击其后的三角形按钮展开Project 列表，并选择一个 Project ，如图 *1582: 选择 Project*所示。

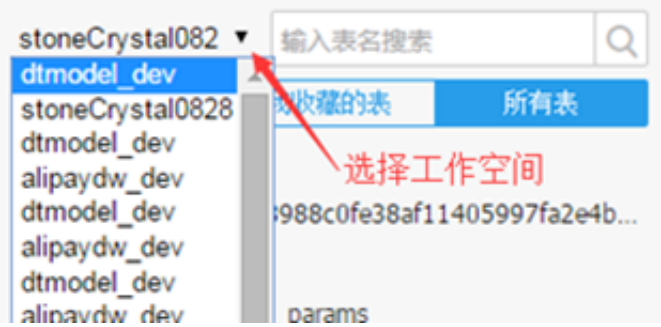
图 1582: 选择 Project



33.2.12.2 浏览表

选择 Project 后，浏览窗口的列表将按照**我创建的表**、**我收藏的表**和**所有表**对当前 Project 中的表进行展示，如图 *1583: 浏览表*所示。

图 1583: 浏览表



您可以按照不同的需要选择不同的标签页进行查看。

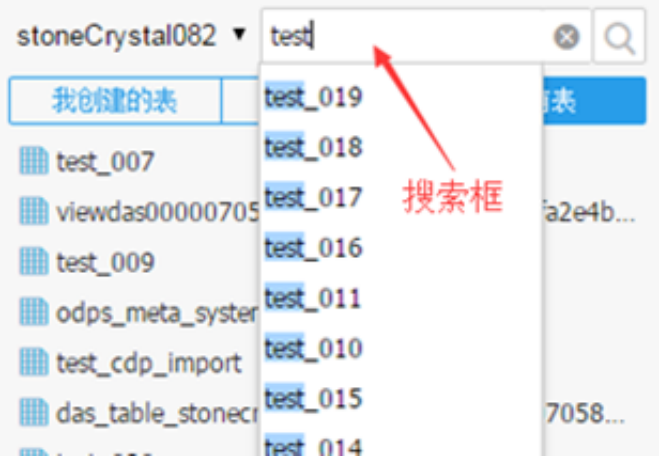
当列表中的表数量过大时，将自动分页。列表下方展示了当前页码和所有页数，您可以通过**上一页**和**下一页**按钮进行翻页。

33.2.12.3 搜索表

当表的数量过多时，难以从浏览窗口中快速找到目标表，为提高工作效率，数据字典提供表的搜索功能。

在浏览窗口右上角的搜索框中输入搜索关键字，将会列出匹配到的表名候选项，以供选择，如图 *1584:* 搜索表所示。

图 1584: 搜索表



33.2.12.4 查看表详细信息

可通过如下三种方式进入表的详细信息页面：

- 在浏览窗口中双击目标表，如图 *1585:* 双击方式所示。

图 1585: 双击方式



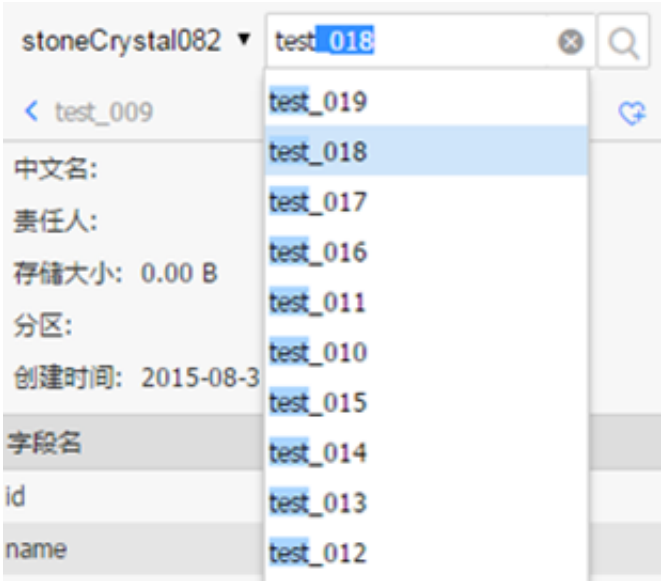
- 在浏览窗口中，右键单击目标表，选择**详细信息**，如图 *1586: 右键方式*所示。

图 1586: 右键方式



- 在搜索给出的候选项中通过上下移动光标，选中目标表，然后在键盘上敲回车键，如图 *1587: 敲回车方式*所示。

图 1587: 敲回车方式



在表的详细信息页面展示了表的基本信息（包括中文名、责任人、存储大小、分区、创建时间等）和表结构信息（包括字段名、字段类型和描述），如图 *1588:* 查看表详细信息所示。

图 1588: 查看表详细信息



33.2.12.5 收藏表

为方便表的查找，可以将常用表进行收藏。

在表的详细信息页面右上角，单击**收藏**按钮，进行收藏或取消收藏，如图 *1589:* 收藏表所示。

图 1589: 收藏表



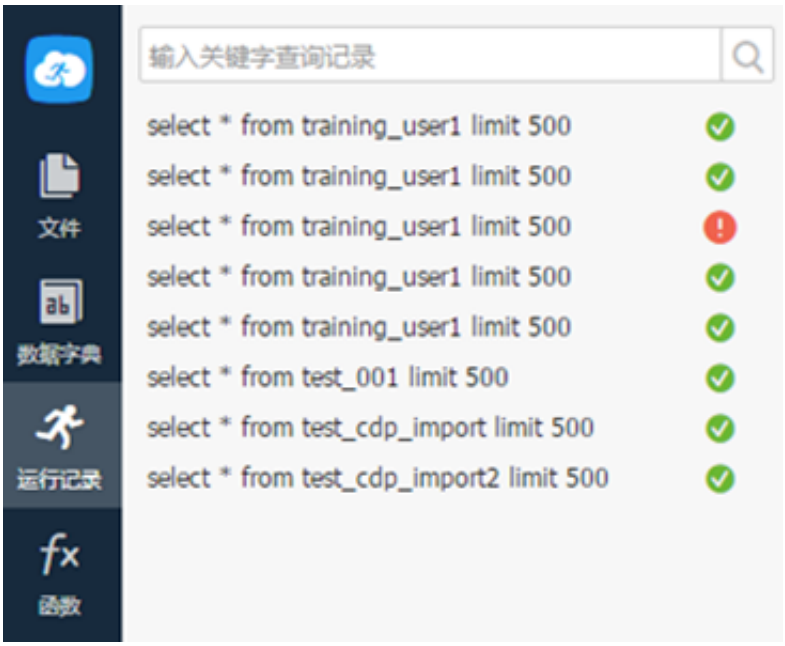
收藏完成后，表将会出现在**我收藏的表**标签页中。

33.2.13 管理运行记录

33.2.13.1 查看运行记录

单击左侧导航栏中的**运行记录**，您可以查看当前 Project 中已执行的所有 SQL 语句的记录和执行状态，如图 *1590:* 查看运行状态所示。

图 1590: 查看运行状态



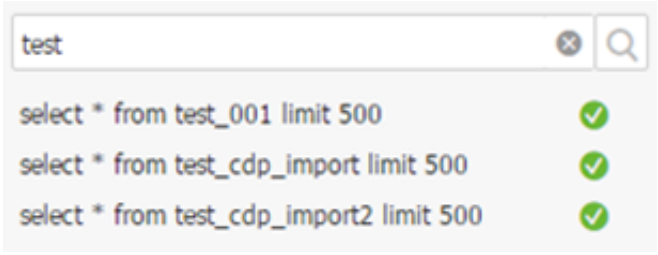
切换不同的 Project ，运行记录会相应切换。

浏览窗中语句右侧的绿色或红色图标分别表示该语句执行成功或失败。

33.2.13.2 搜索运行记录

在浏览窗口上方的搜索框中输入搜索关键字，敲击回车键，浏览窗口将会展示符合查询条件的所有执行记录，如图 *1591: 搜索运行记录*所示。

图 1591: 搜索运行记录



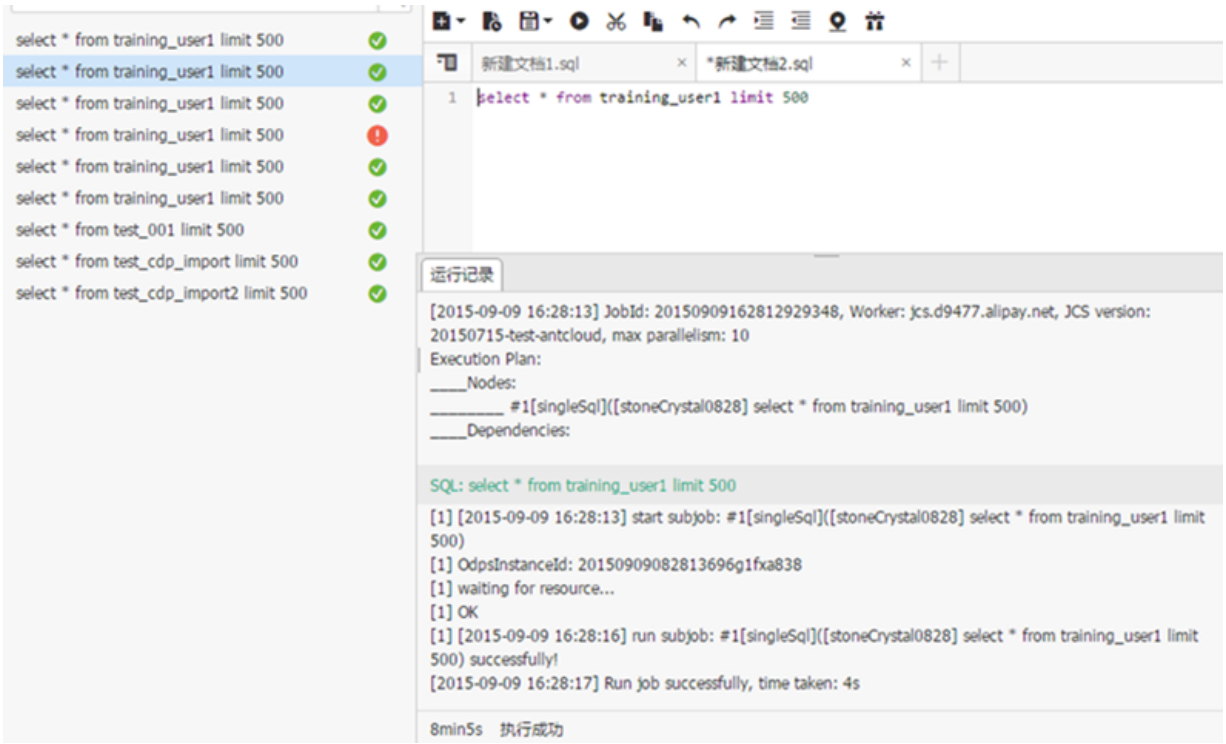
33.2.13.3 打开运行记录

在浏览窗口中，双击目标运行记录或单击右键，选择**打开**。

运行记录中的 SQL 语句将会在新的文件标签页中打开。

语句运行时的详细信息将会展示在运行记录标签页中，如图 *1592: 打开运行记录*所示。

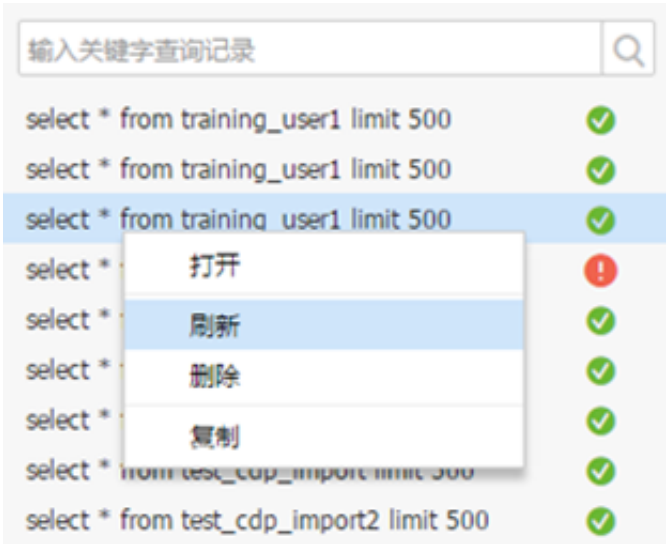
图 1592: 打开运行记录



33.2.13.4 刷新运行记录

在浏览窗口中，右键单击目标运行记录，选择**刷新**，将当前已执行完成但尚未显示的运行记录展示在浏览窗口中，如图 *1593: 刷新运行记录*所示。

图 1593: 刷新运行记录



33.2.13.5 删除运行记录

操作步骤

1. 在浏览窗口中，右键单击目标运行记录，选择**删除**。
2. 在弹出的对话框中，单击**确认**，被选中的运行记录将会被删除。

33.2.13.6 复制运行记录

在浏览窗口中，右键单击目标运行记录，选择**复制**。

被选中的 SQL 语句将会被复制到粘贴板，可以粘贴到其他应用程序中。

33.2.14 管理 MaxCompute 的 UDF

您可以根据一定规则，注册自定义函数 UDF ，经过注册的 UDF 可以在脚本文件中使用，减少重复编码，提高开发效率。

33.2.14.1 创建 UDF

操作步骤

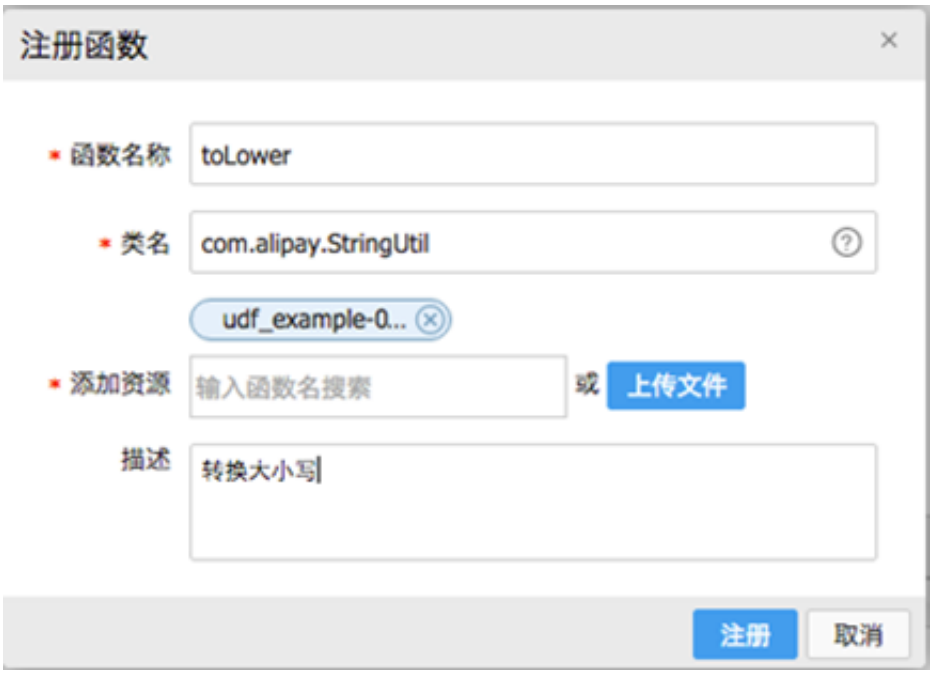
1. 在工具栏选择工具标签页，单击**注册函数**按钮，如图 *1594:* 注册函数所示。

图 1594: 注册函数



2. 在弹出窗口中填写相应内容后，单击**注册**，如图 *1595:* 填写注册信息所示。

图 1595: 填写注册信息



33.2.14.2 修改 UDF

如需修改 UDF ，创建一个同名的 UDF 。

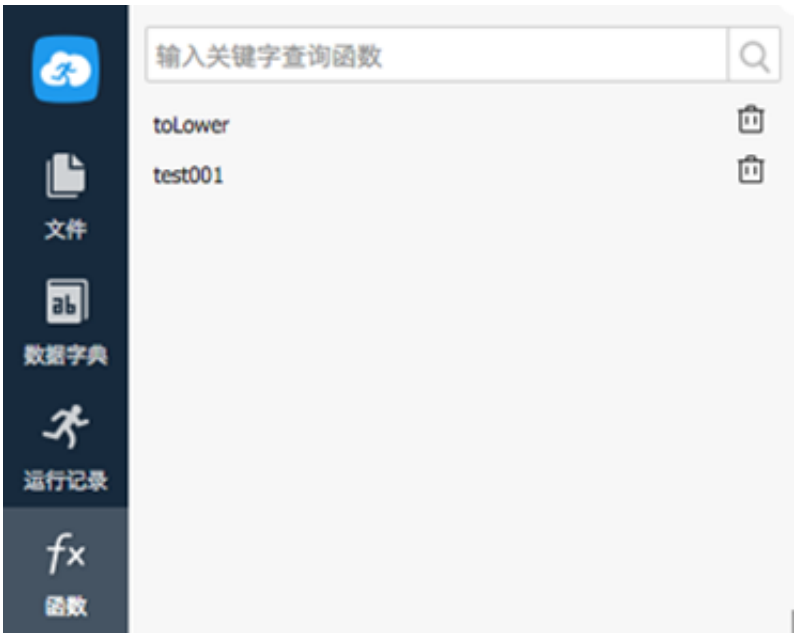
同名 UDF 会覆盖旧版本。

33.2.14.3 查看 UDF

在**数据工厂**侧边栏，单击**函数**按钮，进入函数浏览窗口。

在函数列表中查看已注册的 UDF，如图 *1596: 查看 UDF*所示。

图 1596: 查看 UDF



33.2.14.4 删除 UDF

操作步骤

1. 在**数据工厂**侧边栏，单击**函数**按钮，进入函数浏览窗口。
2. 在函数列表中，单击要删除的 UDF 右侧的**删除**按钮，如图 *1597: 删除 UDF*所示。

   图 1597: 删除 UDF

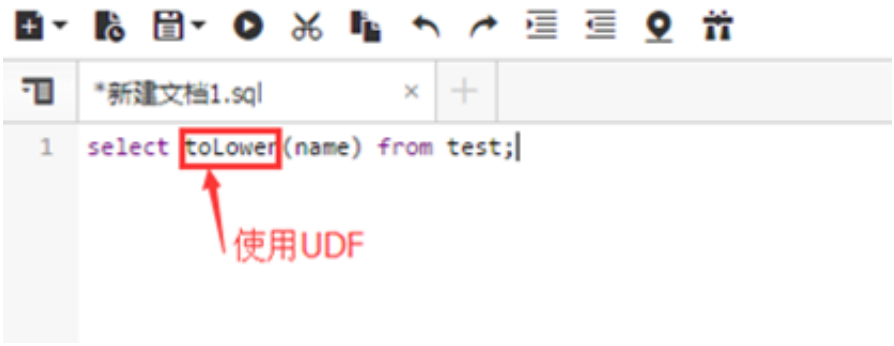


3. 在弹出的窗口中，单击**确认**。

33.2.14.5 使用 UDF

在脚本中直接使用已注册的 UDF 即可，如图 *1598: 使用 UDF*所示。

图 1598: 使用 UDF



33.2.15 上传本地数据文件

操作步骤

1. 在工具栏的**工具**标签页，单击**本地数据导入**按钮，如图 *1599: 导入按钮*所示。

   图 1599: 导入按钮

2. 在弹出的对话框中，选择需要上传的数据文件。
3. 在**本地数据导入**对话框中，选择文件的解析格式、**分隔符号**、**起始行号**、**是否包含表头**和**原始字符集**等内容，如图 *1600: 导入文件设置*所示。

   图 1600: 导入文件设置



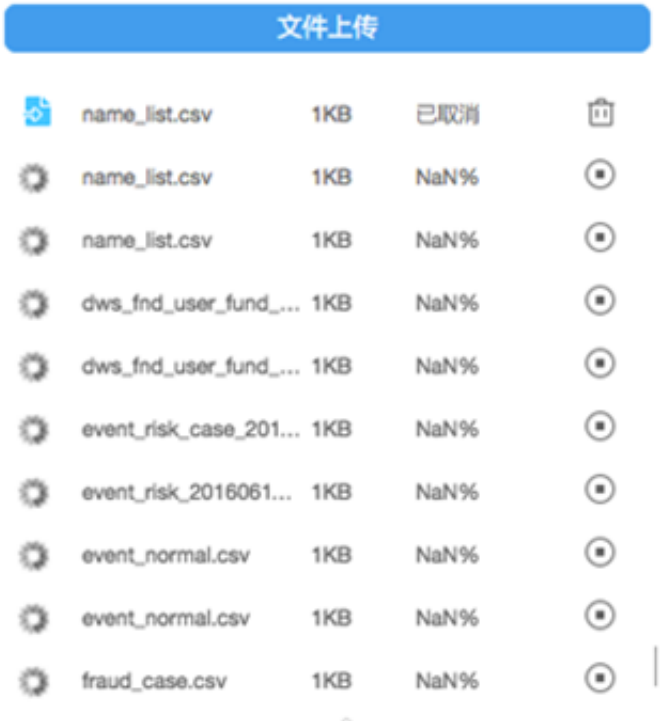
4. 单击**下一步**。
5. 填写目标表名，查看字段映射是否正确，必要时可以对字段映射进行修改。
6. 单击**导入**。

   导入状态将展示在浏览窗口的标签页中。

33.2.16 上传状态和日志

上传本地数据文件后，在屏幕左边侧边栏出现云图标，单击图标，浏览窗口中会展示已上传文件的名称、大小和状态，如图 *1601: 上传状态及日志*所示。

图 1601: 上传状态及日志



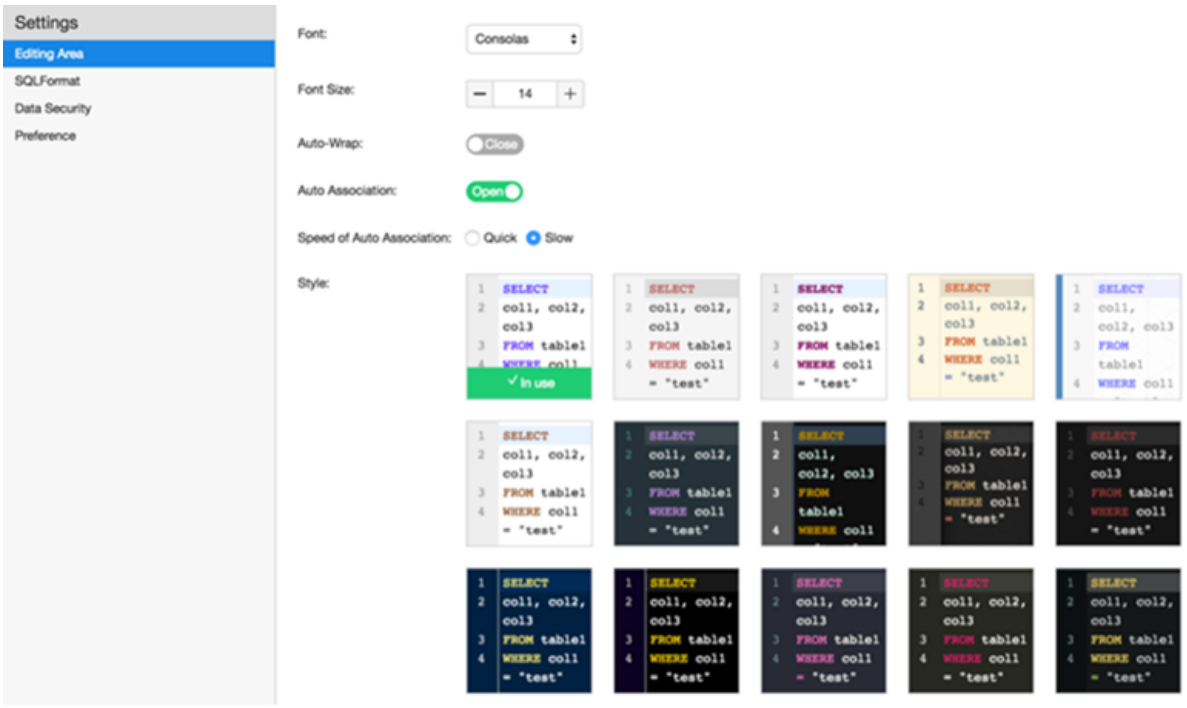
33.2.17 设置编辑区

进入采云间的管理面板，单击**编辑区**。

如图 *1602: 编辑区设置*所示，可以设置脚本的如下属性：

- 字体：设置显示的字体；
- 字号：设置默认显示的字号；
- 换行：是否自动换行；
- 自动提示：是否启用自动提示，启用后，可以做脚本提示；
- 自动提示速度：做脚本提示时，显示的速度；
- 编辑风格：系统预置了18种编辑风格，您可以配置自己喜欢的风格。

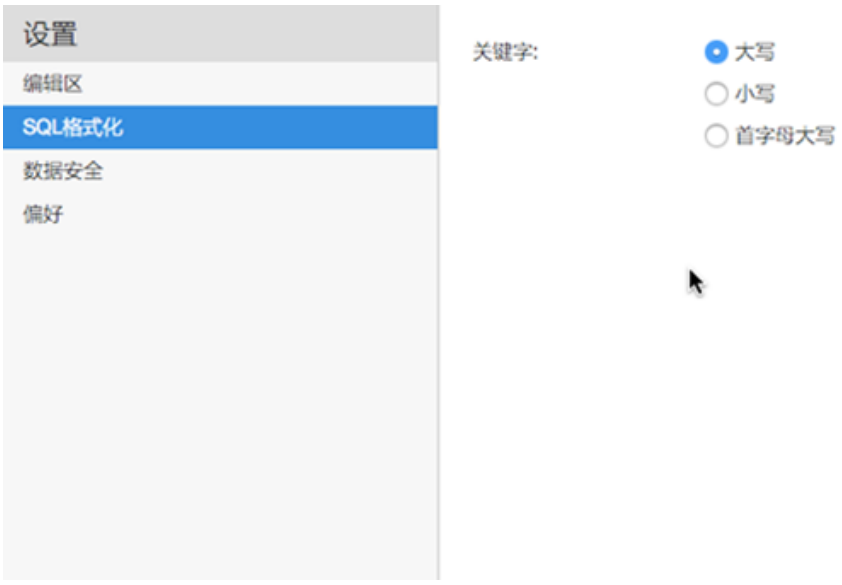
图 1602: 编辑区设置



33.2.18 设置脚本格式化

设置格式化时，关键字的格式规则如图 *1603:* 格式规则所示。

图 1603: 格式规则



33.2.19 数据安全

33.2.19.1 设置最大显示行数

操作步骤

1. 进入采云间的管理面板，单击**数据安全**。

   进入数据安全防护页面。

2. 在**数据工厂最大显示行数**中进行设置，如图 *1604:* 最大显示行数设置所示。

   默认最大行数为 500 ，您可以选择自定义，并填写相应行数。

   图 1604: 最大显示行数设置



33.2.19.2 设置隐私数据展示策略

操作步骤

1. 进入采云间的管理面板，单击**数据安全**。

   进入数据安全防护页面。

2. 在**用户隐私数据展示屏蔽**中进行设置，如图 *1605:* 隐私数据展示的设置所示。

   默认设置为不屏蔽（即展示隐私数据），您可以选择屏蔽或白名单可见。

   选择白名单可见时，需添加白名单用户账号，以逗号分隔。

   图 1605: 隐私数据展示的设置



用户隐私数据展示屏蔽后，用户的身份证号、邮箱、手机号码等隐私信息将以隐藏部分信息的形式展现，而其他数据例如生日、性别等可以正常显示，如图 *1606:* 隐私数据展示屏蔽后所示。

如果关闭隐私数据展示屏蔽，则用户的隐私信息也将正常显示。

图 1606: 隐私数据展示屏蔽后

user_birthday	user_gender	email	mobile_phone
1966-10-26	男	abc@****le.com	130****9314
1977-12-24	女	abc@****le.com	130****9314
1962-4-16	男	abc@****le.com	130****9314
1987-5-5	男	abc@****le.com	130****9314
1968-12-16	男	abc@****le.com	130****9314
1965-2-24	女	abc@****le.com	130****9314
1991-7-6	男	abc@****le.com	130****9314

33.2.19.3 设置结果集下载策略

操作步骤

1. 进入采云间的管理面板，单击**数据安全**。

   进入数据安全防护页面。

2. 在**数据允许下载到本地**中进行设置，如图 *1607: 数据允许下载到本地的设置*所示。

   默认设置为允许，您可以选择不允许或按权限。

   选择**按权限**时，需添加白名单用户账号，账号间以英文（半角）逗号分隔。

图 1607: 数据允许下载到本地的设置



33.2.20 设置个人偏好

进入采云间的管理面板，单击**偏好**，如图 *1608: 个人偏好设置*所示。

图 1608: 个人偏好设置



33.2.21 FAQ

1. 问：MaxCompute 的 SQL 语法规则在哪里能查到？

   回答：在这里。

2. 问：为什么在数据工厂中刚刚创建的表，在任务管理中看不到？

   回答：MaxCompute 元数据的同步不是实时的，一般 10 分钟同步一次。

3. 问：是否可以创建 UDF ？

   回答：可以，在工具菜单中。

4. 问：怎么将本地的数据导入到 MaxCompute 表中？

   回答：可以使用数据上传的功能。

5. 问：我查询一张表，提示我没有权限怎么办？

   回答：请首先向金融云的 MaxCompute 管理员申请 Project 和表的使用权限。

33.3 任务管理

33.3.1 任务调度服务简介

任务调度服务JSS（Job Scheduling Service），提供数据任务的全生命周期管理功能。JSS 是 ETL 中的一项重要内容，主要包括数据抽取、数据加载和数据任务调度。

功能特性

具备如下功能特性：

- 云端数据同步和加载，通过简单易用的配置，支持 RDS、Table Store、MaxCompute 等异构数据库之间的双向同步。
- 图形化任务调度的设置，支持复杂的任务调度规则，包括任务定义、任务上下线、任务挂起、补数据、任务监控和异常报警等。
- 实时监控调度任务，发现问题可快速分析和定位出错原因。

产品主要功能

产品主要有如下功能：

- 数据抽取

  数据抽取是指从数据源中将数据读取出来。数据源在这里是指在系统管理中配置的数据源。

- 数据加载

  数据加载是指将数据加载到数据源中。

  目前数据抽取和加载的数据源支持的类型有：OSS、Table Store、RDS、MaxCompute、Oceanbase。

  在任务调度里面，数据抽取与加载是用同一个任务完成的，任务类型叫做同步任务，也就是说数据通常不需要在第三方落地，直接从目标抽取后加载到目标表中。这能大大的简化数据传输转移的过程。

- 数据任务调度

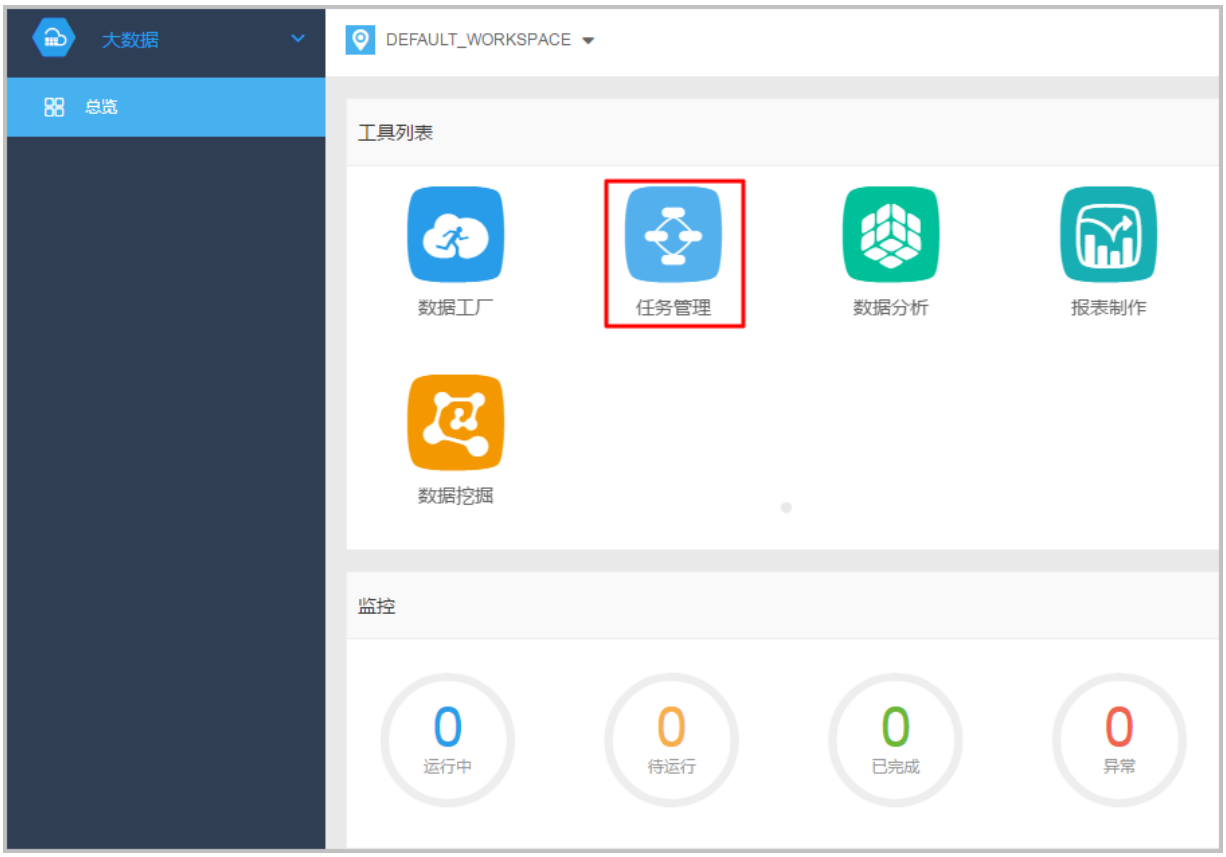
  在 DW 环境中，通常绝大多数的 ETL 任务都不是孤立运行的，都需要满足一定的依赖条件，依赖条件可能是时间、上游任务、外部文件或者其他触发条件等，当大量的任务都需要进行条件判断来决定是否运行的时候，就需要任务调度来指挥。

  任务调度需要设置任务类型、任务输入参数和任务调度参数等，任务执行后通常会返回结果和日志。

33.3.2 进入任务管理

进入采云间控制台，单击右边**工具列表**中的**任务管理**，如图 *1609:* 进入任务管理所示。

图 1609: 进入任务管理

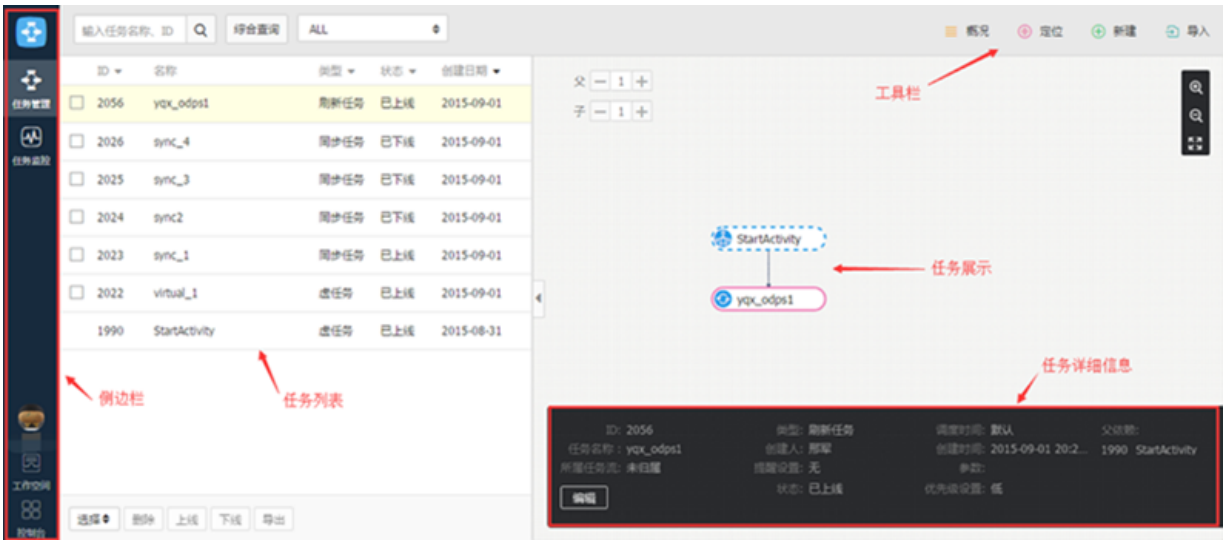


33.3.3 管理任务

33.3.3.1 任务管理界面介绍

单击**任务调度**后，进入任务管理界面，如图 *1610:* 任务管理界面所示。

图 1610: 任务管理界面



侧边栏

任务调度主要分为任务管理和任务监控两大模块，可以在侧边栏点击按钮进入相应模块。

通过侧边栏下方的工作空间按钮，可以进行工作空间的切换。

工具栏

工具栏主要提供如下四个功能：

- 概况：展示当前工作空间下各类任务的统计信息
- 定位：根据任务名称或 ID 对任务进行搜索和定位；
- 新建：新建同步、刷新、检测和虚任务等；
- 导入：导入本地文件。

任务列表

任务列表中展示了任务的 ID 、名称、类型、状态和创建时间等，可以根据这些属性对任务进行排序。

任务列表上方的搜索框可以根据不同条件对任务进行筛选。

通过勾选任务列表左侧的复选框，可以选中目标任务，单击任务列表底部的**删除**、**上线**、**下线**、**导出**等按钮进行相应操作。

任务展示

在任务列表右侧的面板中，以可视化的方式展示了任务的名称、类型和依赖关系等。

您可以选中目标任务，单击右键，在环境菜单中选择相应功能对任务进行操作。通过面板左上角的任务父子层级控件来调整当前面板中所展示的任务数量。

任务详细信息

单击任务展示面板中的任务，面板下方的详细信息会相应改变。

该区域展示了任务基本信息、调度设置等。

33.3.3.2 新建同步任务

同步任务的目的是在不同类型的数据库间进行数据的迁移。

背景信息

目前支持MySQL、OSS、Table Store、Oceanbase、SQL Server 与 MaxCompute 之间的数据同步。

操作步骤

1. 在任务管理工具栏中选择**新建** > **同步任务**，如图 *1611:* 创建同步任务所示。

   图 1611: 创建同步任务



2. 在弹出的对话框中填写相关内容。

   对于不同数据源之间的同步任务，所需填写的内容不尽相同。

   以下章节中将分别描述不同类型的同步任务。

33.3.3.2.1 MySQL 与 Maxcompute 之间的数据同步（单表模式）

操作步骤

1. 数据源类型选择 **MySQL**和**单表**，选择所需同步数据所在的数据库并填写表名；目标类型选择**ODPS**，选择目标表所在的 Project，并填写目标表名，如图 *1612:* 选择数据源及目标类型所示。

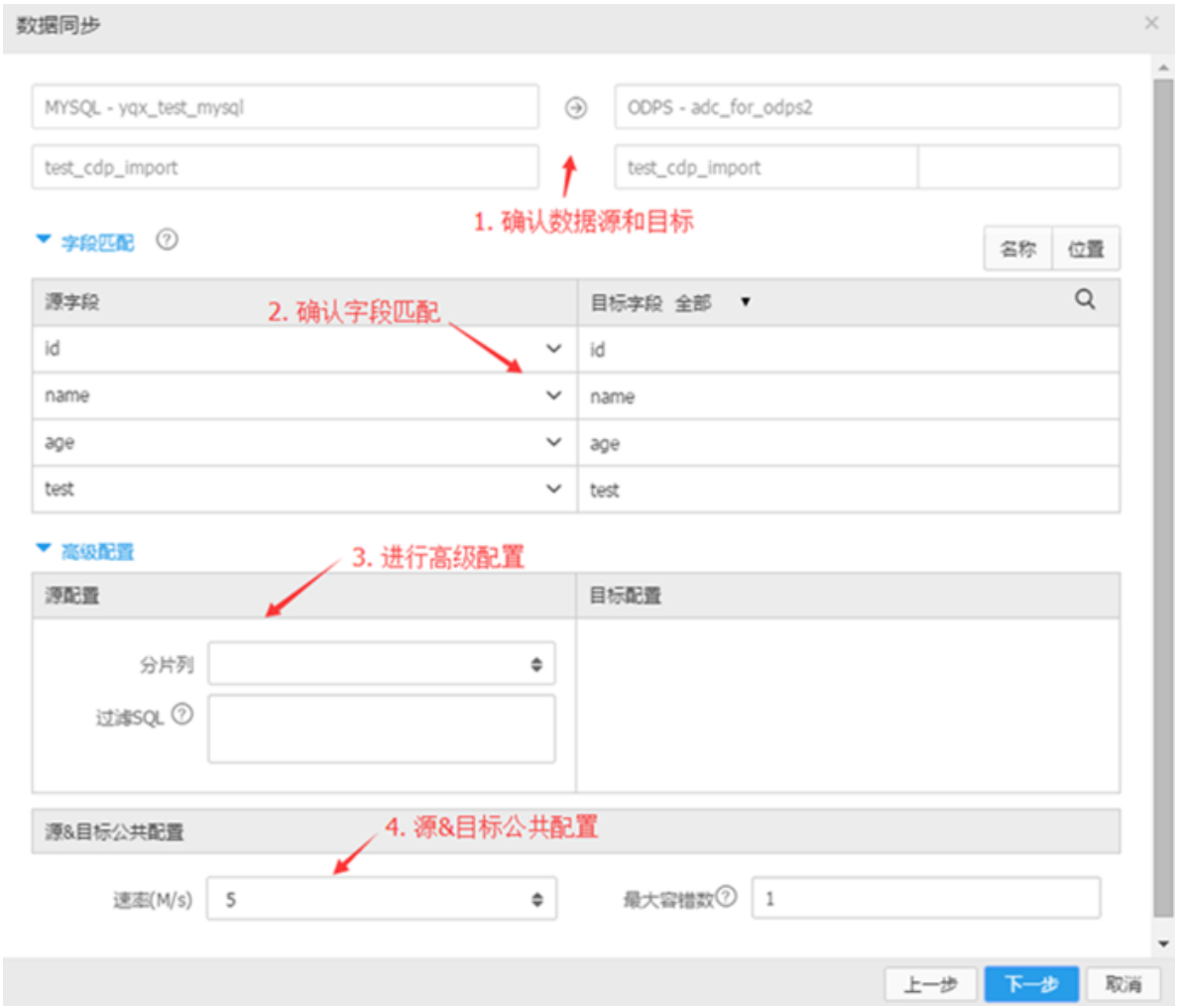
   图 1612: 选择数据源及目标类型



2. 单击**下一步**。

3. 确认或按照需求调整**字段匹配**，如图 *1613:* 数据同步设置所示。

图 1613: 数据同步设置



4. （可选）在**高级配置**中填写相应字段，相关说明如表 *119:* 高级配置项说明所示。

表 119: 高级配置项说明

配置项	说明
分片列	用户可以选择表中的一列，作为将表划分为不同片的依据，以便进行多线程同步。 系统将使用智能算法，根据当前表的大小、网络状况、计算资源等将表分片，以达到数据同步的最高效率。 建议选择主键或重复取值较少的数值类型的字段作为分片列。

配置项	说明
过滤 SQL	在进行数据同步时，可以使用过滤 SQL 选择符合条件的数据，而忽略其他数据，相当于 SQL 语句中的WHERE 条件。在填写过滤 SQL 时，不需再填写“WHERE”关键字，系统会自动添加。

5. 在**源&目标公共配置**中填写**速率**和**最大容错数**，相关配置项说明如表 *120: 源&目标公共配置项说明*所示。

表 120: 源&目标公共配置项说明

配置项	说明
最大速率	通过设置最大速率限制当前同步任务的数据传输速度，以免单个任务占用过多网络资源。 默认的最大速率为 5M/s。
最大容错	最大容错表示该任务结束后，能够接受的源表与目标表中数据不一致的最大行数。 若错误行数大于等于最大容错数，则任务失败，默认最大若错数为 1 ，即不接受任何数据不一致的情况。

6. 单击**下一步**。

33.3.3.2.2 MySQL 与 Maxcompute 之间的数据同步（多表模式）

背景信息

其中多表模式表示从一个或多个数据源的多个表同步数据到单一的目标表中，不能同步到多个目标表中。

多表模式中必须保证多个数据源表具有相同的结构或拥有相同的字段。

操作步骤

1. 数据源类型选择 **MySQL**和**多表**。
2. 在显示的页面，单击**选择表**，进入新窗口，如图 *1614: 选择表*所示。

图 1614: 选择表



3. 在**选择表**窗口中，输入查询条件，选中正确数据源和表，单击**选择**，在右侧列表中设置模板表，如图 *1615:* 选择表窗口所示。

   用来做数据映射（mapping）的时候，选择模板表的字段会作为字段清单。

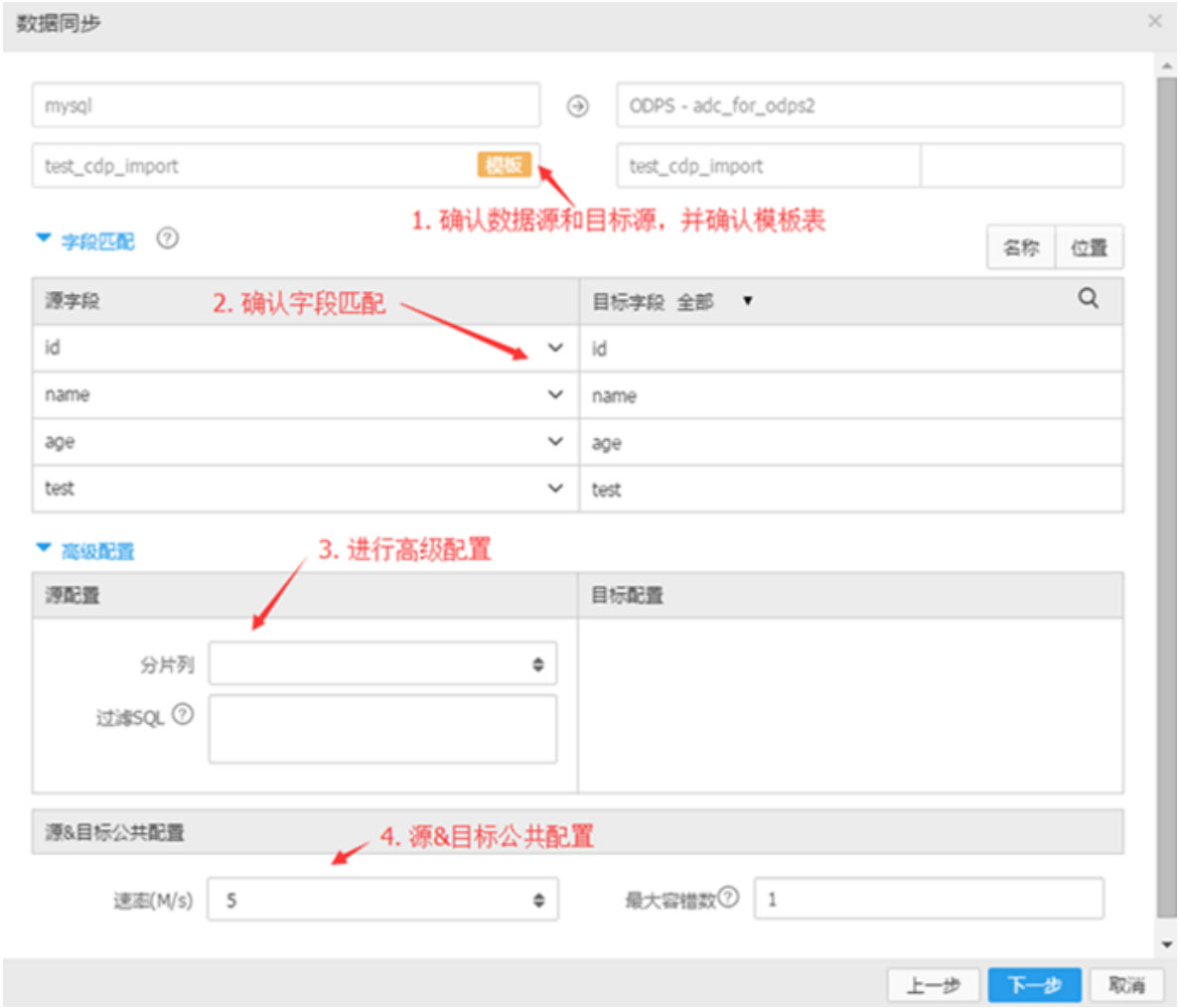
   说明: 只有多表同步才选择模板表。

图 1615: 选择表窗口



4. 单击**确定**，确认模板表的选择。
5. 确认或按照需求调整**字段匹配**，如图 *1616:* 同步设置所示。

图 1616: 同步设置



6. （可选）在**高级配置**中填写相应字段，相关说明如表 *121:* 高级配置项说明所示。

表 121: 高级配置项说明

配置项	说明
分片列	用户可以选择表中的一列，作为将表划分为不同片的依据，以便进行多线程同步。 系统将使用智能算法，根据当前表的大小、网络状况、计算资源等将表分片，以达到数据同步的最高效率。 建议选择主键或重复取值较少的数值类型的字段作为分片列。
过滤 SQL	在进行数据同步时，可以使用过滤 SQL 选择符合条件的数据，而忽略其他数据，相当于 SQL 语句中的WHERE 条件。在填写过滤 SQL

配置项	说明
	时，不需再填写“WHERE”关键字，系统会自动添加。

7. 在**源&目标公共配置**中填写**速率**和**最大容错数**，相关配置项说明如表 *122: 源&目标公共配置项说明*所示。

表 122: 源&目标公共配置项说明

配置项	说明
最大速率	通过设置最大速率限制当前同步任务的数据传输速度，以免单个任务占用过多网络资源。 默认的最大速率为 5M/s。
最大容错	最大容错表示该任务结束后，能够接受的源表与目标表中数据不一致的最大行数。 若错误行数大于等于最大容错数，则任务失败，默认最大若错数为 1 ，即不接受任何数据不一致的情况。

8. 单击**下一步**。
9. 填写任务名称，任务名称不能重复，根据需求为该任务设置父依赖、任务分组、参数及优先级，如图 *1617: 任务设置*所示。

   为方便管理，建议在任务描述中对该任务进行简要描述。

图 1617: 任务设置



10. （可选）设置调度计划。

    目前支持分钟、小时、每日、每周、每月执行的调度计划。

- 以分钟为单位执行调度计划

  选择调度计划的**分钟**标签页，并填写起调时间，如图 *1618: 分钟标签页*所示。

  图 1618: 分钟标签页



- 以小时为单位执行调度计划

  选择调度计划的**小时**标签页，并填写起调时间，如图 *1619: 小时标签页*所示。

  图 1619: 小时标签页



- 每日执行调度计划

  选择调度计划的**每日**标签页，并填写起调时间，如图 *1620: 每日标签页*所示。

  图 1620: 每日标签页



- 每周执行调度计划

选择调度计划的**每周**标签页，选择需要执行的日期，如周一至周五，并填写起调时间，如图*1621: 每周标签页*所示。

图 1621: 每周标签页



- 每月执行调度计划

  选择调度计划的**每月**标签页，选择需要执行的日期，如 1 日、 16 日，并填写起调时间，如图*1622: 每月标签页*所示。

  图 1622: 每月标签页



  注意: 如果设置了每周或者每月的调度，任务还是会每天启动，但是如果没到执行日期条件，任务内容是不执行，缺省显示是成功的，这是为了能触发下游的任务。

11.设置正常提醒和异常提醒方式。

12.单击**确定**。

33.3.3.2.3 OSS 与 Maxcompute 之间的数据同步

操作步骤

1. 数据源类型选择 **OSS**， 选择所需同步数据所在的文件系统和相对地址，根据数据源选择文件类型、分隔符等选项，并填写列模板；目标类型选择 **ODPS**，选择目标表所在的 Project，并填写目标表名，如图 *1623:* 选择数据源及目标类型所示。

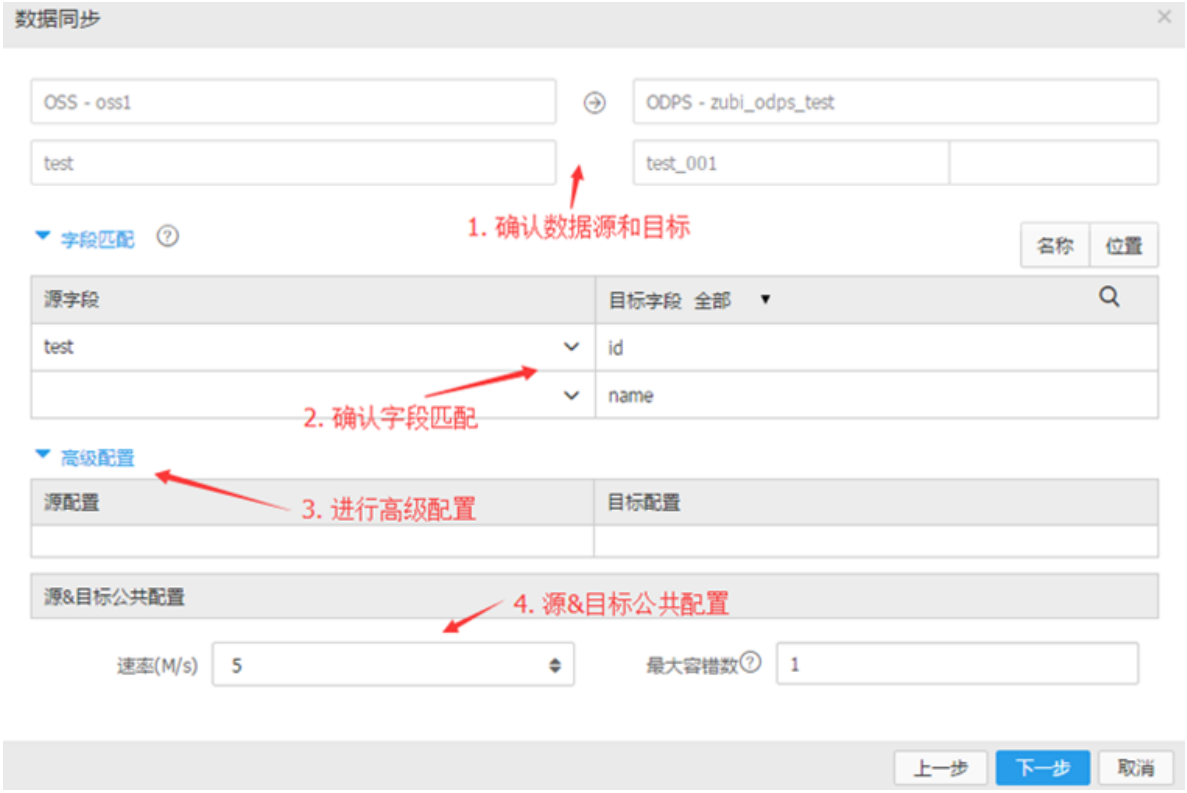
   其中，列模板是一个字符串，用半角逗号分隔；文件匹配模式支持？和*的模糊匹配。

   图 1623: 选择数据源及目标类型



2. 单击**下一步**。
3. 确认或按照需求调整**字段匹配**，如图 *1624:* 数据同步设置所示。

   图 1624: 数据同步设置



4. （可选）在**高级配置**中填写相应字段，相关说明如表 *123:* 高级配置项说明所示。

表 123: 高级配置项说明

配置项	说明
分片列	用户可以选择表中的一列，作为将表划分为不同片的依据，以便进行多线程同步。 系统将使用智能算法，根据当前表的大小、网络状况、计算资源等将表分片，以达到数据同步的最高效率。 建议选择主键或重复取值较少的数值类型的字段作为分片列。
过滤 SQL	在进行数据同步时，可以使用过滤 SQL 选择符合条件的数据，而忽略其他数据，相当于 SQL 语句中的WHERE 条件。在填写过滤 SQL 时，不需再填写“WHERE”关键字，系统会自动添加。

5. 在**源&目标公共配置**中填写**速率**和**最大容错数**，相关配置项说明如表 *124:* 源&目标公共配置项说明所示。

表 124: 源&目标公共配置项说明

配置项	说明
最大速率	通过设置最大速率限制当前同步任务的数据传输速度，以免单个任务占用过多网络资源。 默认的最大速率为 5M/s。
最大容错	最大容错表示该任务结束后，能够接受的源表与目标表中数据不一致的最大行数。 若错误行数大于等于最大容错数，则任务失败，默认最大若错数为 1 ，即不接受任何数据不一致的情况。

6. 单击**下一步**。
7. 填写任务名称，任务名称不能重复，根据需求为该任务设置父依赖、任务分组、参数及优先级，如图 *1625:* 任务设置所示。

   为方便管理，建议在任务描述中对该任务进行简要描述。

   图 1625: 任务设置



8. （可选）设置调度计划。

   目前支持分钟、小时、每日、每周、每月执行的调度计划。

   - 以分钟为单位执行调度计划

     选择调度计划的**分钟**标签页，并填写起调时间，如图 *1626:* 分钟标签页所示。

     图 1626: 分钟标签页



- 以小时为单位执行调度计划

  选择调度计划的**小时**标签页，并填写起调时间，如图 *1627: 小时标签页*所示。

  图 1627: 小时标签页



- 每日执行调度计划

  选择调度计划的**每日**标签页，并填写起调时间，如图 *1628: 每日标签页*所示。

  图 1628: 每日标签页



- 每周执行调度计划

  选择调度计划的**每周**标签页，选择需要执行的日期，如周一至周五，并填写起调时间，如图 *1629: 每周标签页*所示。

  图 1629: 每周标签页



- 每月执行调度计划

  选择调度计划的**每月**标签页，选择需要执行的日期，如 1 日、 16 日，并填写起调时间，如图 *1630: 每月标签页*所示。

  图 1630: 每月标签页



注意: 如果设置了每周或者每月的调度，任务还是会每天启动，但是如果没到执行日期条件，任务内容是不执行，缺省显示是成功的，这是为了能触发下游的任务。

9. 设置正常提醒和异常提醒方式。
10. 单击**确定**。

33.3.3.2.4 Table Store 与 Maxcompute 之间的数据同步

操作步骤

1. 数据源类型选择 **OTS**，选择所需同步数据所在的数据库并填写表名；目标类型选择 **ODPS**，选择目标表所在的 Project，并填写目标表名，如图 *1631: 选择数据源及目录类型*所示。

   由于 Table Store（原OTS）中存在数据的多个版本，这里提供数据的普通和多版本同步。

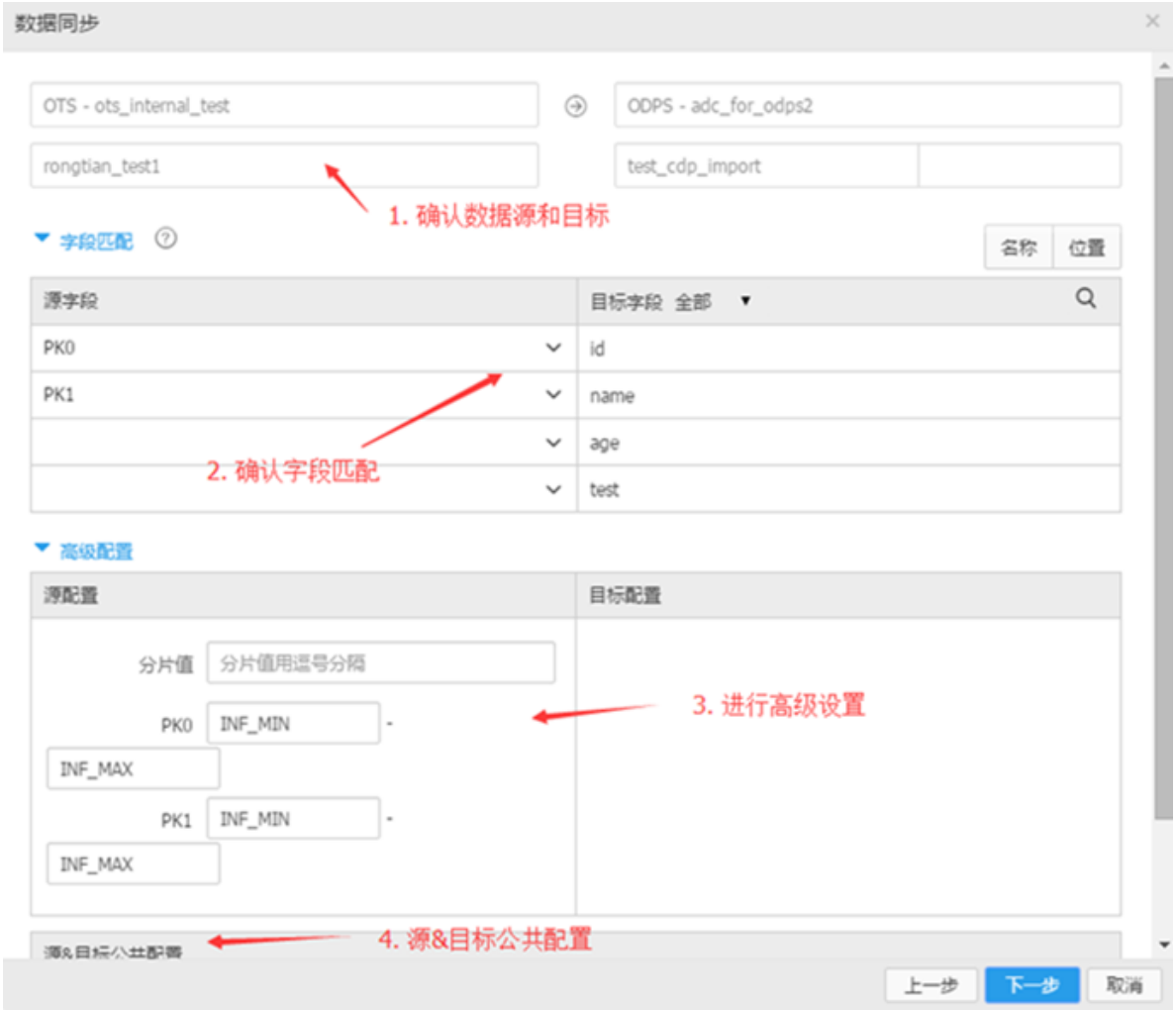
普通模式仅同步最新版本的数据，需要根据数据源填写列模板；多版本模式将会同步所有数据，不需填写列模板。

图 1631: 选择数据源及目录类型



2. 单击**下一步**。
3. 确认或按照需求调整**字段匹配**，如图 *1632:* 数据同步设置所示。

图 1632: 数据同步设置



4. （可选）在**高级配置**中填写相应字段，相关说明如表 *125:* 高级配置项说明所示。

表 125: 高级配置项说明

配置项	说明
分片列	用户可以选择表中的一列，作为将表划分为不同片的依据，以便进行多线程同步。 系统将使用智能算法，根据当前表的大小、网络状况、计算资源等将表分片，以达到数据同步的最高效率。 建议选择主键或重复取值较少的数值类型的字段作为分片列。
过滤 SQL	在进行数据同步时，可以使用过滤 SQL 选择符合条件的数据，而忽略其他数据，相当于 SQL 语句中的WHERE 条件。在填写过滤 SQL

配置项	说明
	时，不需再填写“WHERE”关键字，系统会自动添加。

5. 在**源&目标公共配置**中填写**速率**和**最大容错数**，相关配置项说明如表 *126: 源&目标公共配置项说明*所示。

表 126: 源&目标公共配置项说明

配置项	说明
最大速率	通过设置最大速率限制当前同步任务的数据传输速度，以免单个任务占用过多网络资源。 默认的最大速率为 5M/s。
最大容错	最大容错表示该任务结束后，能够接受的源表与目标表中数据不一致的最大行数。 若错误行数大于等于最大容错数，则任务失败，默认最大若错数为 1 ，即不接受任何数据不一致的情况。

6. 单击**下一步**。
7. 填写任务名称，任务名称不能重复，根据需求为该任务设置父依赖、任务分组、参数及优先级，如图 *1633: 任务设置*所示。

   为方便管理，建议在任务描述中对该任务进行简要描述。

图 1633: 任务设置



8. （可选）设置调度计划。

   目前支持分钟、小时、每日、每周、每月执行的调度计划。

- 以分钟为单位执行调度计划

  选择调度计划的**分钟**标签页，并填写起调时间，如图 *1634: 分钟标签页*所示。

  图 1634: 分钟标签页



- 以小时为单位执行调度计划

  选择调度计划的**小时**标签页，并填写起调时间，如图 *1635: 小时标签页*所示。

  图 1635: 小时标签页



- 每日执行调度计划

  选择调度计划的**每日**标签页，并填写起调时间，如图 *1636: 每日标签页*所示。

  图 1636: 每日标签页



- 每周执行调度计划

选择调度计划的**每周**标签页，选择需要执行的日期，如周一至周五，并填写起调时间，如图*1637: 每周标签页*所示。

图 1637: 每周标签页



- 每月执行调度计划

  选择调度计划的**每月**标签页，选择需要执行的日期，如 1 日、 16 日，并填写起调时间，如图*1638: 每月标签页*所示。

  图 1638: 每月标签页



  注意: 如果设置了每周或者每月的调度，任务还是会每天启动，但是如果没到执行日期条件，任务内容是不执行，缺省显示是成功的，这是为了能触发下游的任务。

9. 设置正常提醒和异常提醒方式。
10. 单击**确定**。

33.3.3.2.5 Oceanbase 与 Maxcompute 之间的数据同步（单表模式）

操作步骤

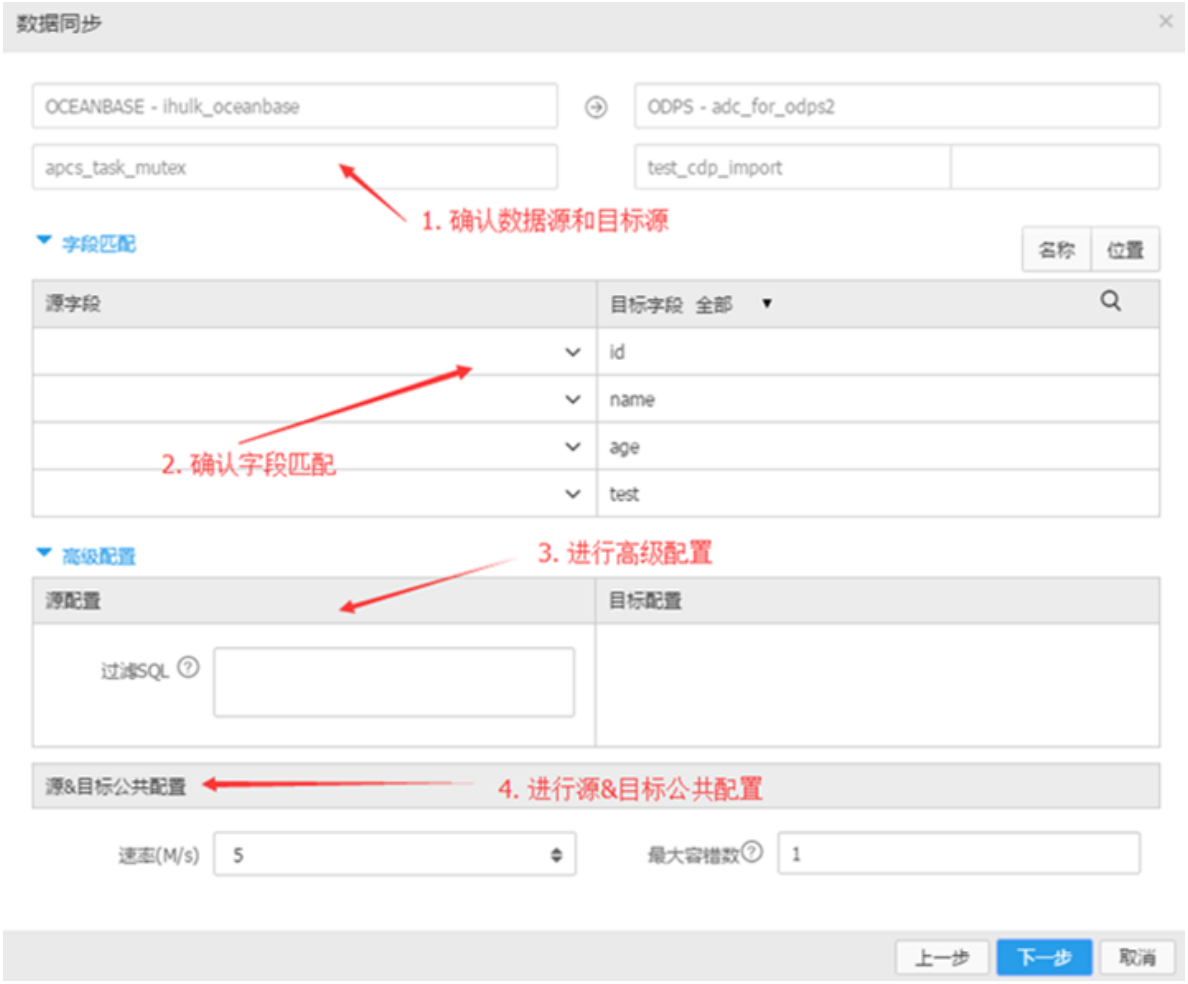
1. 数据源类型选择 **OCEANBASE**和**单表**；目标类型选择 **ODPS**，选择目标表所在的 Project，并填写目标表名，如所图 *1639:* 选择数据源及目标类型示。

   图 1639: 选择数据源及目标类型



2. 单击**下一步**。
3. 确认或按照需求调整**字段匹配**，如图 *1640:* 数据同步设置所示。

   图 1640: 数据同步设置



4. （可选）在**高级配置**中填写相应字段，相关说明如表 *127: 高级配置项说明*所示。

表 127: 高级配置项说明

配置项	说明
分片列	用户可以选择表中的一列，作为将表划分为不同片的依据，以便进行多线程同步。 系统将使用智能算法，根据当前表的大小、网络状况、计算资源等将表分片，以达到数据同步的最高效率。 建议选择主键或重复取值较少的数值类型的字段作为分片列。
过滤 SQL	在进行数据同步时，可以使用过滤 SQL 选择符合条件的数据，而忽略其他数据，相当于 SQL 语句中的WHERE 条件。在填写过滤 SQL 时，不需再填写“WHERE”关键字，系统会自动添加。

5. 在**源&目标公共配置**中填写**速率**和**最大容错数**，相关配置项说明如表 *128: 源&目标公共配置项说明*所示。

   表 128: 源&目标公共配置项说明

配置项	说明
最大速率	通过设置最大速率限制当前同步任务的数据传输速度，以免单个任务占用过多网络资源。 默认的最大速率为 5M/s。
最大容错	最大容错表示该任务结束后，能够接受的源表与目标表中数据不一致的最大行数。 若错误行数大于等于最大容错数，则任务失败，默认最大若错数为 1 ，即不接受任何数据不一致的情况。

6. 单击**下一步**。

33.3.3.2.6 Oceanbase 与 Maxcompute 之间的数据同步（多表模式）

背景信息

Oceanbase 中的多表模式与 MySQL中的多表模式类似，相关内容请参见 MySQL 的*MySQL 与 Maxcompute 之间的数据同步#多表模式#*章节内容。

操作步骤

1. 参考 MySQL 中的多表模式设置后，在窗口中填写任务名称，任务名称不能重复，根据需求为该任务设置父依赖、任务分组、参数、优先级、调度计划和提醒，如图 *1641: 任务设置*所示。

   为方便管理，建议在任务描述中对该任务进行简要描述。

   图 1641: 任务设置



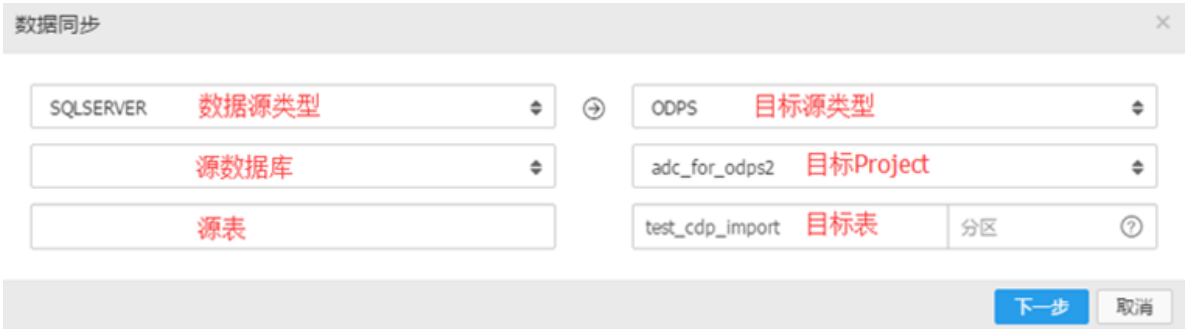
2. 单击**确定**。

33.3.3.2.7 SQL Server 与 Maxcompute 之间的数据同步

操作步骤

1. 数据源类型选择 **SQLSERVER**，选择所需同步数据所在数据库并填写表名；目标类型选择 **ODPS**，选择目标表所在的 Project，并填写目标表名，如图 *1642:* 选择数据源及目标类型所示。

   图 1642: 选择数据源及目标类型



2. 单击**下一步**。
3. 其余部分与 MySQL 单表的同步任务相同，相关内容请参考*MySQL 与 Maxcompute 之间的数据同步#单表模式#*章节。

33.3.3.3 新建刷新任务

刷新任务是将数据工厂里撰写的代码配置成任务。

背景信息

刷新任务目前支持的数据源有 ODPS ；支持的脚本语言有 SQL、Shell、Perl。

操作步骤

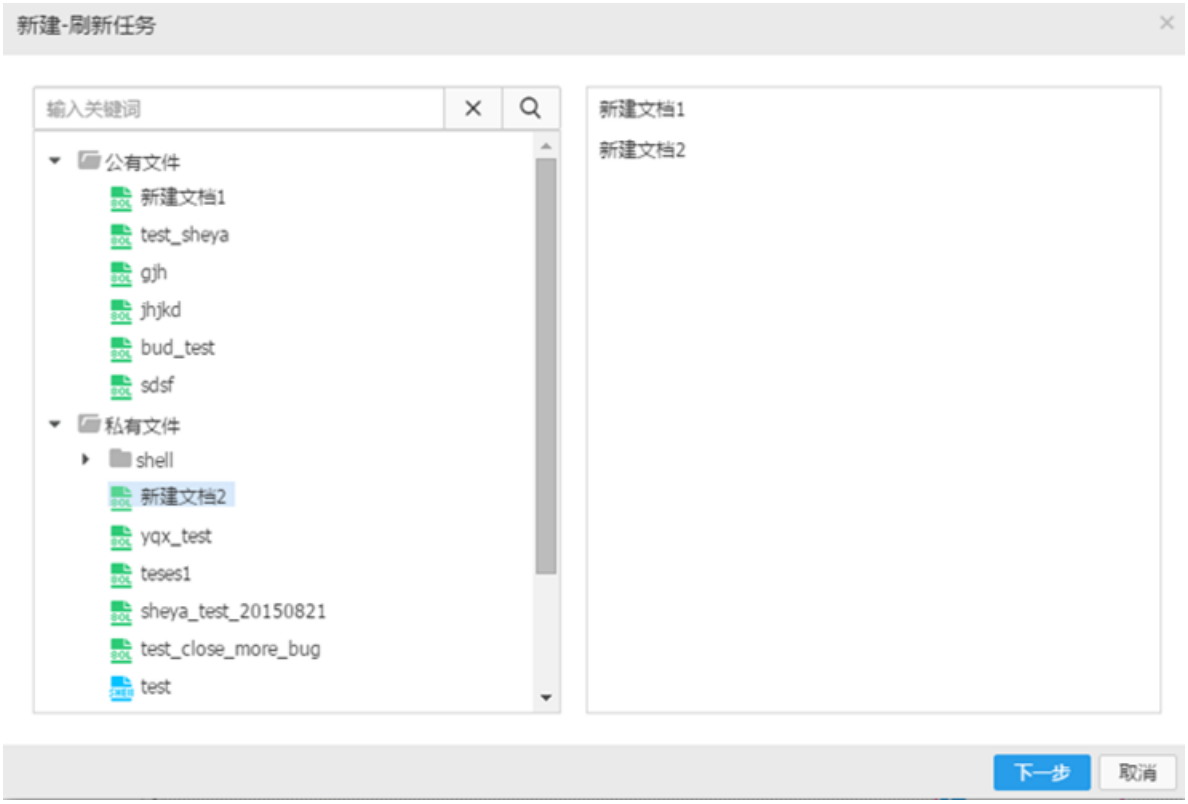
1. 在数据工厂里，保存好数据清洗处理的代码。
2. 在任务管理工具栏中，选择**新建** > **刷新任务**，进行刷新任务的配置和创建，如图 *1643:* 新建刷新任务所示。

图 1643: 新建刷新任务



3. 在弹出的窗口左侧的列表中，选择需要使用的文件，选中的文件会展示在右侧列表中，如图 *1644:* 选择文件所示。

图 1644: 选择文件



4. 单击**下一步**。

5. 刷新任务配置。

   支持单个配置和批量配置模式，分别解决不同场景下的任务创建模式：

   - 进行单个任务新建时，配置窗口如图 *1645: 配置单个任务*所示。

     图 1645: 配置单个任务

   - 进行批量任务新建时，配置窗口如图 *1646: 配置批量任务*所示。

     图 1646: 配置批量任务

   调度计划的设置可参考*MySQL 与 Maxcompute 之间的数据同步#多表模式#*中**设置调度计划**的内容。

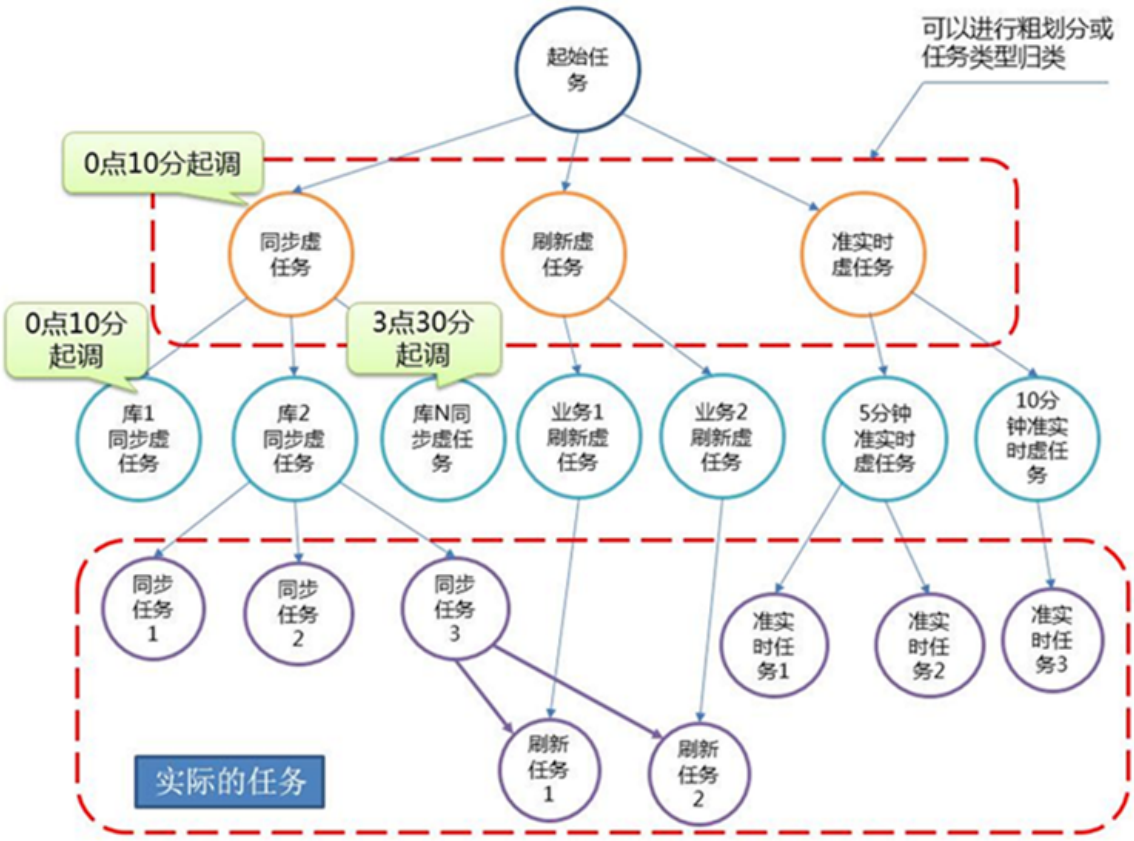
33.3.3.4 新建虚任务

虚任务正如名称所表述的一样，基本不占时间、不消耗资源，主要用于体现任务之间的某种衔接关系或业务属性关系，在任务节点上，引入虚任务主要是为了方便管理。

背景信息

如图 *1647:* 典型虚任务所示，是一个典型的虚任务应用场景。

图 1647: 典型虚任务



虚任务用于如下几种情况：

- 虚任务用于不同任务类型的分类

  比如：所有同步任务都挂在同步虚任务节点下；所有准实时任务都挂在准实时虚任务下。

- 虚任务用于管理

  比如：同步库 1 的数据，所有同步库 1 的任务都可以挂在库 1 的同步虚任务节点上；数仓领域经常区分不同的主题或业务 BU ，如 A 部门创建的任务挂在业务 1 刷新虚任务下（这里建议仅挂几个初始的刷新任务到某个刷新虚任务下，原因在于刷新任务更多的是依赖同步过来的数据或加工过的数据）

- 虚任务用于依赖衔接

比如：库 N 在 3 点 30 分才能开始同步数据，可以设置一个虚节点，将起调时间设置为 3 点 30 分起调，同步库 N 的任务都挂到该节点下，调度计划设置为默认。某一天，库N 需要提前或延后同步数据，更改这个虚拟节点的起调时间即可。

操作步骤

1. 在任务管理工具栏中，选择**新建** > **虚任务**，如图 *1648: 新建虚任务*所示。

   图 1648: 新建虚任务

2. 在弹出的对话框中填写相应内容，如图 *1649: 虚任务设置*所示。

   任务名称需保持唯一，为方便管理，建议为任务添加简单的任务描述。

   根据需求对该任务设置父任务、产出表、任务分组、优先级、调度计划和提醒。

   调度计划的设置可参考*MySQL 与 Maxcompute 之间的数据同步#多表模式#*中**设置调度计划**的内容。

   图 1649: 虚任务设置

3. 单击**确定**。

33.3.3.5 新建检测任务

对于来自外部系统的触发条件，可以通过调用 API 新建检测任务。

背景信息

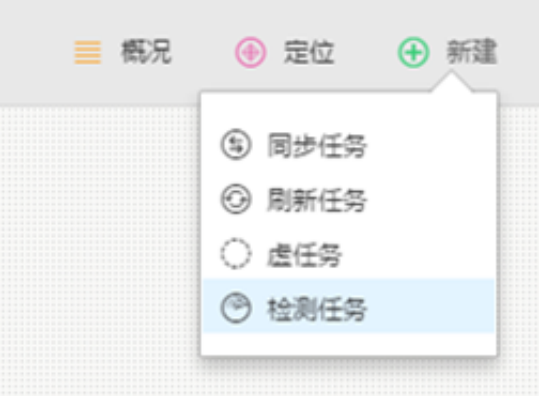
当检测任务检测到外部触发条件得到满足时，则启动任务。

检测任务的等待时间是有限制的，如果超过一天没有触发，将会报告错误信息。

操作步骤

1. 在任务管理工具栏中，选择**新建** > **检测任务**，如图 *1650:* 新建检测任务所示。

图 1650: 新建检测任务



2. 在弹出的窗口中填写任务名称，任务名称不能重复，根据需求对该任务设置父任务、任务分组、优先级和调度计划，如图 *1651:* 检测任务设置所示。

   为方便任务管理，建议填写简要的任务描述。

图 1651: 检测任务设置



3. 单击**确定**。

33.3.3.6 触发接口

接口路径

/popapi/v1.0/task.json? dqsPopMethod=triggerJobRun

调用方式

直接调用采云间 http 接口的 get 请求。

描述

触发检测任务（即条件节点）状态为可以执行通过，但检测节点运行时会运行结束并运行下级任务节点。

传入参数

传入参数相关说明如表 *129: 传入参数说明*所示。

表 129: 传入参数说明

名称	类型	必填	描述
renterId	String	Y	租户 ID
workspaceId	String	Y	工作空间 ID
userNumber	Long	Y	云账号对应的 UID
taskId	Long	Y	任务 ID
bizDate	String	Y	业务日期，格式：YYYY-MM-DD

返回结果

返回结果如表 *130: 返回结果*所示。

表 130: 返回结果

名称	名称	名称	名称
resultFlag	Boolean	Y	触发检测节点成功
errCode	String	N	触发失败返回错误码
errMessage	String	N	触发失败返回错误描述信息

错误代码

错误代码如表 *131:* 错误代码说明所示。

表 131: 错误代码说明

代码	中文描述	英文描述
DQS06010	无效的参数	Invalid Parameters
DQS06066	触发检测节点失败，因为任务无效或不存在	Invalid Task Or Task Not Exists
DQS06065	触发检测节点失败	Trigger TaskException: {ExceptionMessage}

33.3.3.7 搜索任务

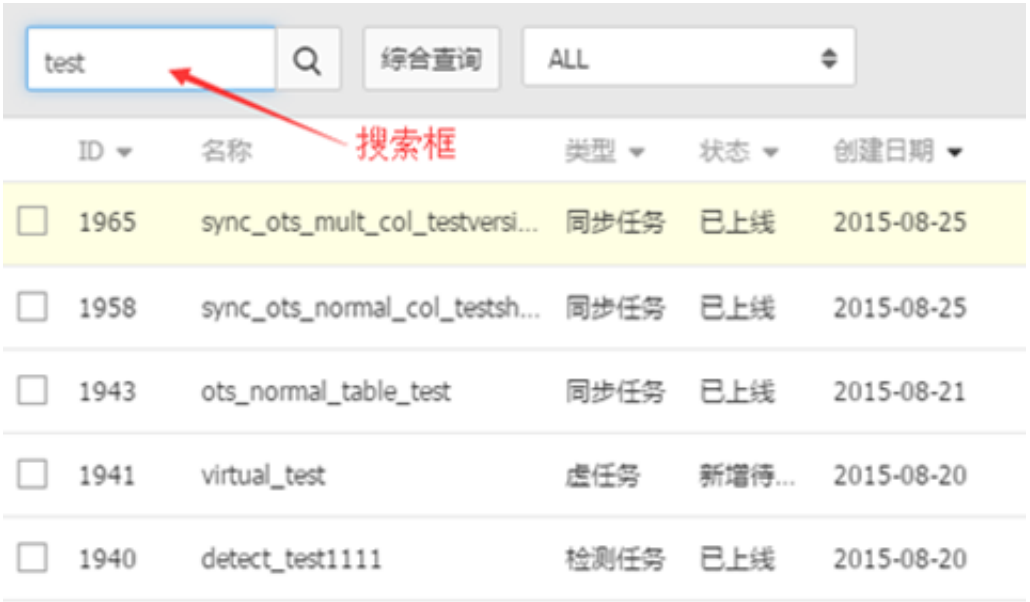
通常项目中会创建较多数量的任务，为快速定位到某一任务，可以进行任务搜索。

操作步骤

1. 在任务管理的搜索框中输入搜索条件（包括任务名称和 ID ），并回车，如图 *1652:* 搜索任务所示。

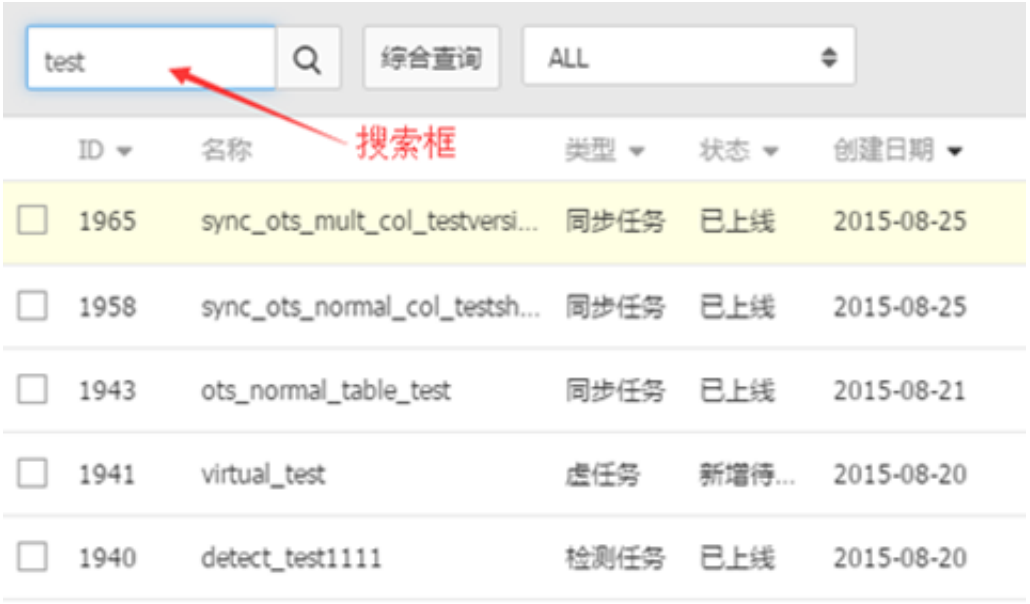
   搜索任务支持模糊匹配。

   图 1652: 搜索任务



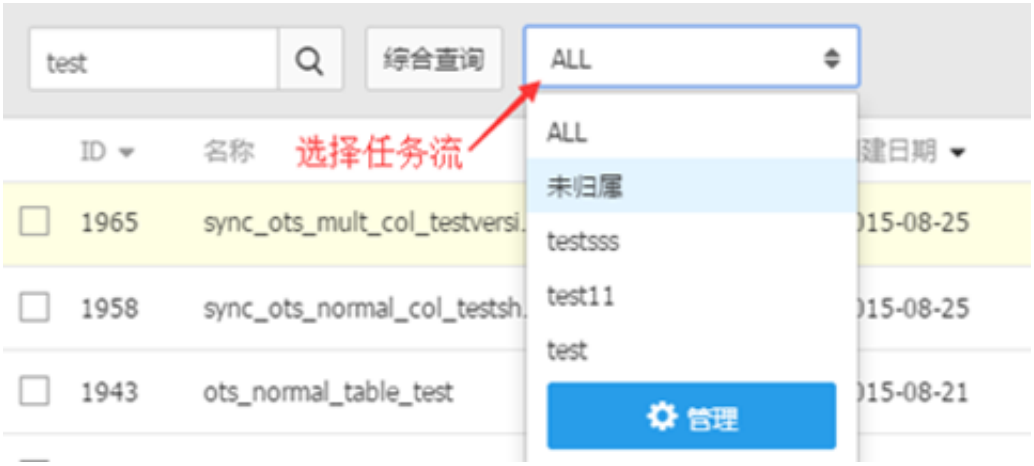
2. 在综合查询中，可以对任务状态、类型、创建者进行筛选，如图 *1653:* 筛选任务所示。

图 1653: 筛选任务



3. 在综合查询右侧的下拉框中，可以对任务分组进行筛选，如图 *1654:* 任务分组筛选所示。

图 1654: 任务分组筛选



33.3.3.8 上线任务

新建的任务，需要上线后，才能作用到生产环境。

背景信息

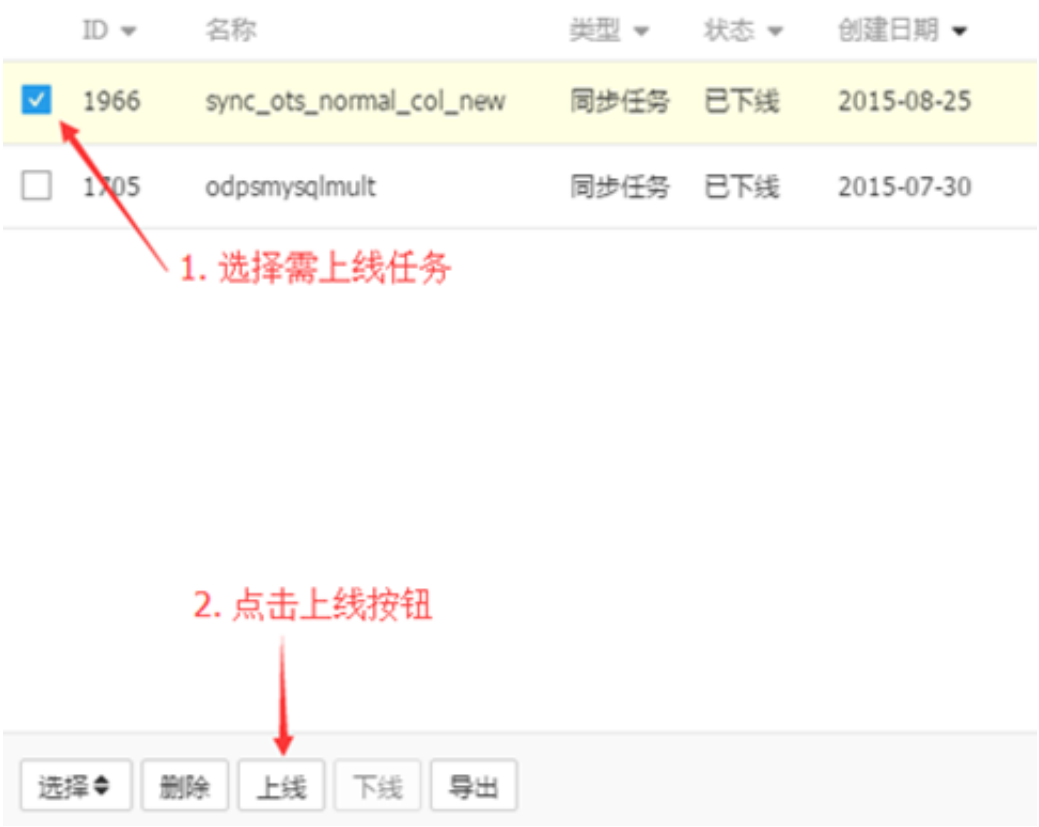
建的任务，在未发布之前，在画布上显示的时候，是灰色的。

您可以通过左侧列表或右键进行任务上线。

- 通过左侧列表上线

  在任务列表中，勾选需要上线的任务，单击**上线**，如图 *1655:* 选择待上线的任务所示。

  图 1655: 选择待上线的任务



- 通过右键方式上线

  在面板中，右键单击需要上线的任务，选择**上线**，如图 *1656:* 右键方式上线所示。

  图 1656: 右键方式上线



33.3.3.9 下线任务

与任务上线类似，可以通过左侧列表或右键进行任务下线。

背景信息

任务下线后，在画布上显示为灰色。

- 通过左侧列表下线

  在任务列表中，勾选需要下线的任务，单击**下线**，如图 *1657:* 选择待下线的任务所示。

  图 1657: 选择待下线的任务



- 通过右键方式下线

  在面板中，右键单击需要下线的任务，选择**下线**，如图 *1658:* 右键方式下线所示。

  图 1658: 右键方式下线



33.3.3.10 删除任务

对于不再需要的任务，可以进行删除操作。

背景信息

已上线的任务（包括已上线和修改待发布状态）不能删除。

已下线和新增待发布状态的任务可以进行删除操作。

- 在左侧列表中选中需要删除的任务，单击下方**删除**。
- 在面板中，右键单击需要删除的任务，选择**删除**。

33.3.3.11 任务执行补数据

任务创建后需要同步历史数据，需要通过补数据来完成。

背景信息

任务失败导致数据缺失，需要重跑一个或多个工作日的某任务，也可以通过补数据来进行。

操作步骤

1. 在任务管理里，右键单击需要补数据的任务，选择**补数据** > **本任务**或**本任务与下游子任务**，如图 *1659: 右键单击任务*所示。

图 1659: 右键单击任务



2. 在弹出的对话框中，选择任务日期的起止时间，单击**确定**，如图 *1660:* 选择任务日期所示。

图 1660: 选择任务日期



33.3.3.12 查看任务操作日志

操作步骤

1. 在任务管理面板中，右键单击目标任务，选择**查看任务操作日志**，如图 *1661:* 右键单击任务所示。

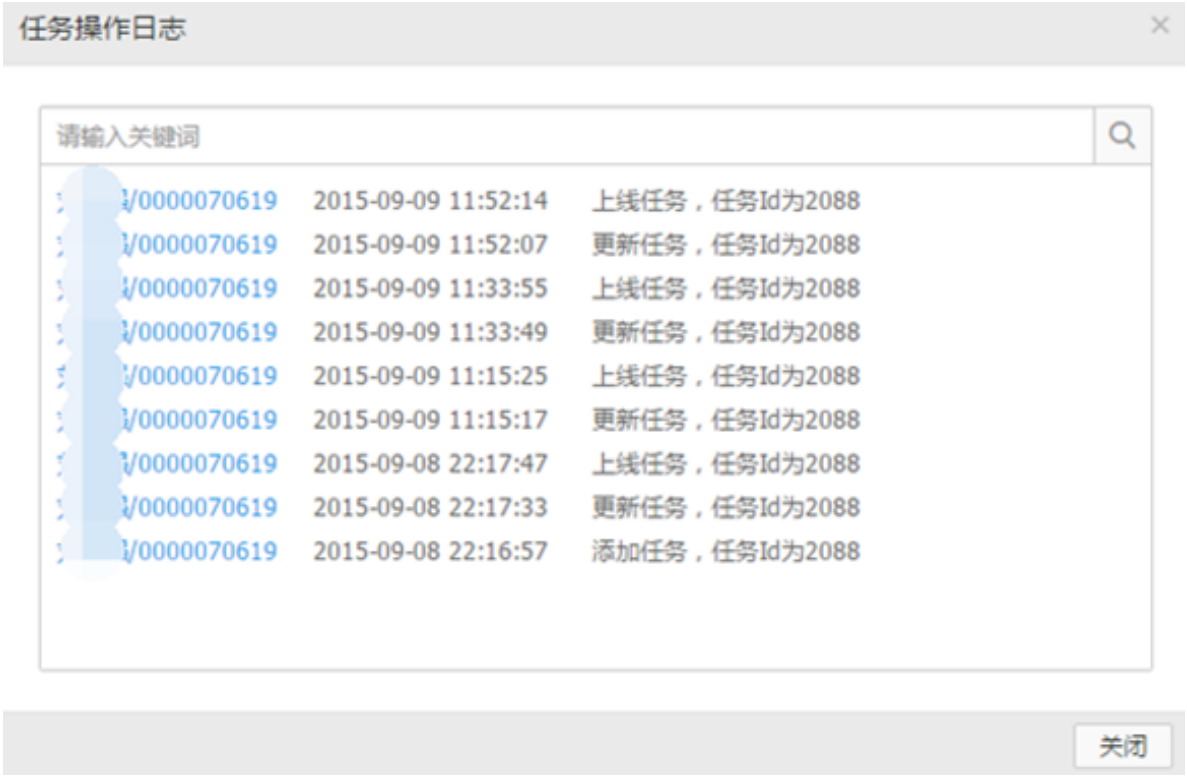
   日志将展示在弹出的对话框中。

图 1661: 右键单击任务



2. 在对话框中可以输入关键字对日志进行过滤，快速定位需要分析的日志内容，如图 *1662:* 查看操作日志所示。

图 1662: 查看操作日志



33.3.3.13 修改根节点启动时间

操作步骤

1. 在面板中，右键单击根节点，选择**编辑启动时间**，如图 *1663:* 右键单击根节点所示。

图 1663: 右键单击根节点



2. 在弹出的窗口中设置**起调时间**，如图 *1664:* 设置起调时间所示。

调度计划的设置可参考*MySQL 与 Maxcompute 之间的数据同步#多表模式#*中**设置调度计划**的内容。

图 1664: 设置起调时间



33.3.3.14 导出任务

操作步骤

1. 在浏览窗口中，选中一个或多个需要导出的任务，如图 *1665:* 导出任务所示。

图 1665: 导出任务



2. 单击**导出**。

   文件将以 zip 压缩文件的形式导出到本地机器。

33.3.3.15 导入任务

背景信息

用户可以从本地机器导入由任务导出功能导出的 ZIP 压缩文件。

操作步骤

1. 在页面右上角单击**导入**，如图 *1666:* 导入任务所示。

   图 1666: 导入任务



2. 在弹出的窗口中选择欲导入文件，完成任务导入。

33.3.3.16 跨周期依赖

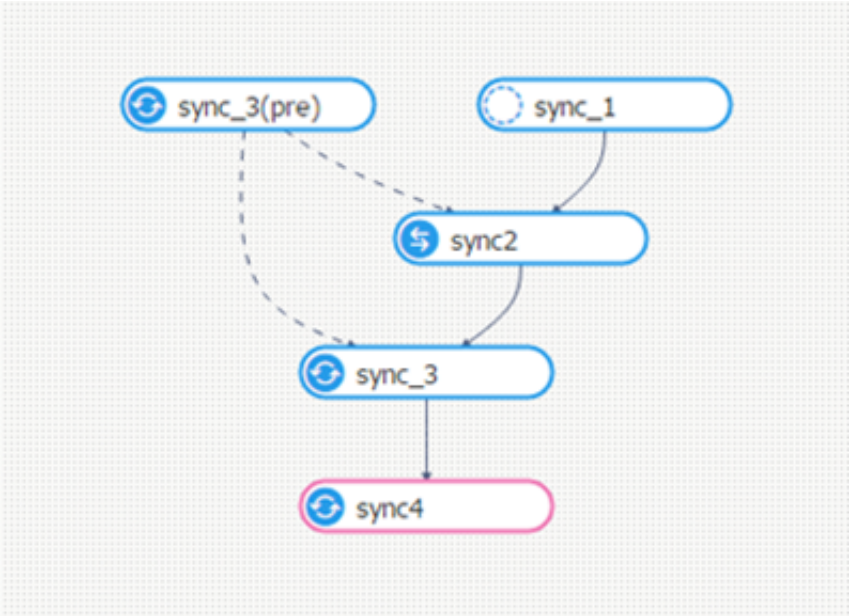
背景信息

若任务依赖于其他任务（包括自己）的历史执行状态时，成为跨周期依赖。

跨周期依赖在右侧面板的图形中以虚线展示。

如图 *1667: 跨周期依赖示例*所示中的 sync2 、 sync_3 任务都依赖于 sync_3(pre) 任务。

图 1667: 跨周期依赖示例

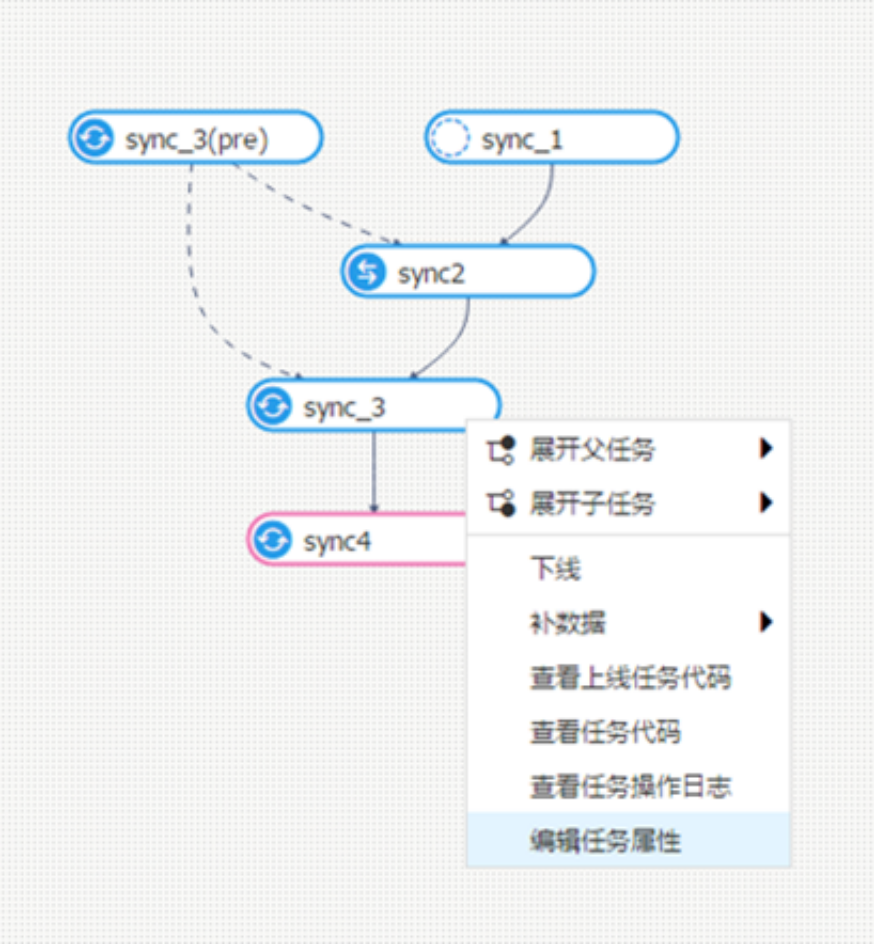


其中， sync_3（pre）其实代表了sync_3 任务的上一次执行情况。也就是说只有当 sync_3 任务本次执行成功，在下一个任务周期里，sync2 和 sync_3 任务才会执行。

操作步骤

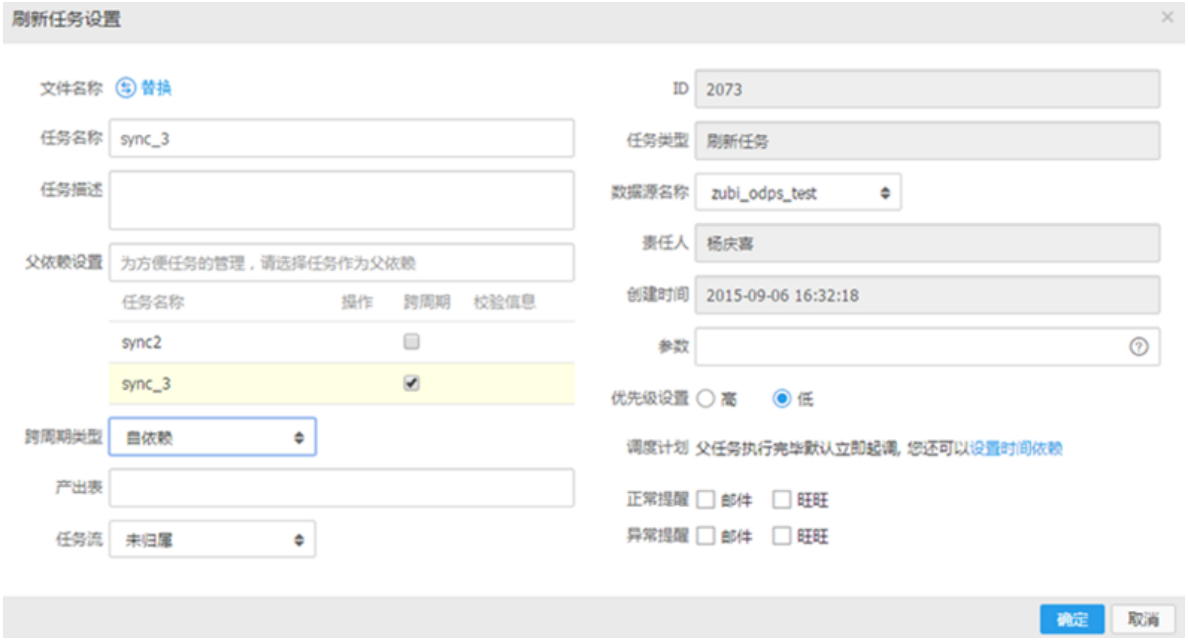
1. 右键单击任务，选择**编辑任务属性**，如图 *1668:* 右键单击任务所示。

图 1668: 右键单击任务



2. 在弹出的窗口中，对跨周期依赖进行编辑，如图 *1669:* 编辑跨周期依赖所示。

图 1669: 编辑跨周期依赖



在**父依赖设置**中，勾选**任务名称**后的**跨周期**选项，并选择跨周期类型。

其中，**自依赖**是任务依赖于任务本身的历史执行状态；一级子节点依赖是任务依赖于一级子任务的历史执行状态；自定义可以自行设置依赖情况。

33.3.3.17 业务类参数替换

33.3.3.17.1 通用参数替换

配置举例：

某个 Activity 为 MaxCompute 类型，其代码如下：

```
select * from ${myTable} where ${theNum} > 3 and ${theString} is not null
```

其中，该 Activity 的 parameter = “myTable=’ mytableName’ theNum=10 theString=’ haha’ ”。

那么最终执行的代码为：

```
select * from mytableName where 10 > 3 and haha is not null
```

33.3.3.17.2 Shell 的特殊处理

配置举例：

某个 Activity 为 Shell 类型，其代码如下：

```
echo $1 $2
```

其中，该 Activity 的 parameter=" mytableName 10" 。

那么最终执行的代码为：

```
echo mytableName 10
```

Shell 的 Activity 级别的参数无视 flow 级别的参数。 flow 级别的参数，必须是 a=b 这类规则。

Shell 中如果有非常复杂的多Shell 文件，请注意如下规则：

用cd dirname $0 这个以后，就会转到目前正在执行的Shell 的路径下，然后用相对路径就可以访问 zip 里面的路径；如果是访问固定的 rmp 包程序，用绝对路径每次执行，都可能是在不同的 gateway 上的。

33.3.3.18 系统上下文

33.3.3.18.1 上下文的写入与读取

Shell 写入上下文 Shell 代码中，可以按如下方式写入：

```
pro=xxxxx tab=yyyyy @@{writeContext} project=$pro table=$tab
```

从上下文读取任何类型的 Activity 的代码中，可以如下读取 context ：

```
@@{project}
```

33.3.3.18.2 根据 dataDate 进行加减计算

业务代码中可以根据 dataDate 进行加减计算，精确到天，一般形式如：

```
@@{yyyyMMdd +/- Ny}
```

说明如下：

- @@{}：表示以数据日期为基准进行计算
- yyyyMMdd：表示日期的格式，还可以为 yyy；yyyyMM；MM； dd 等
- +/-：表示进行加或减计算
- N：N 表示需要加减的数值；y 表示单位：年，其中包括如下的单位：
  - y：年
  - m：月

- d：天

例子：

假设当前的数据日期为 20140603 ，则

- @@{yyyy-MM-dd + 2y}：2016-06-03
- @@{yyyyMMdd - 2m}：20140403
- @@{yyyyMMdd - 2d}：20140601
- @@{yyyy - 2y}：2012
- @@{yyyyMM + 3m}：201409

在代码中使用例如：

```
SELECT * FROM table_a WHERE t_date = @@{yyyyMMdd - 1d}
```

上例中， @@{yyyyMMdd - 1d} 即为数据日期 -1 天，如果数据日期为 20140603 ，则 @@{yyyyMMdd - 1d}表示 20140602 ，那么最终执行的代码为：

```
SELECT * FROM table_a WHERE t_date = 20140603
```

33.3.3.18.3 根据定时的表达式进行加减计算

业务代码中可以根据 Activity 中定义的 scheduleExpression ，即定时的表达式时间进行加减计算，精确到秒。

一般形式如：

```
@@[yyyyMMdd HH:mm:ss +/- Ny]
```

说明如下：

- @@[]：表示以定时的表达式时间为基准进行计算
- yyyyMMdd HH:mm:ss：表示日期的格式，还可以为 yyyy, yyyyMM、 MM、 dd、 HH:mm:ss 等
- +/-：表示进行加或减计算
- Ny：N 表示需要加减的数值；y 表示单位：年，其中包括如下的单位：
  - y：年
  - m：月
  - d：天
  - h：小时

- mi：分

例子

以 Activity 的定时时间 20140603132000 为例，则：

- @@[yyyy-MM-dd HH:mm:ss + 2y]：2016-06-03 13:20:00
- @@[yyyyMMdd - 2m]：20140403
- @@[yyyyMMdd - 2d]：20140601
- @@[yyyy - 2y]：2012
- @@[yyyyMM + 3m]：201409
- @@[HH:mm:ss + 3mi]：13:23:00

在代码中实际使用例如：

```
SELECT * FROM table_a WHERE t_date = '@@[yyyy-MM-dd HH:mm:ss - 1d]'
```

上例中，如果数据日期为 20140603132000 ，则 @@[yyyy-MM-dd HH:mm:ss - 1d] 表示 2014-06-02 13:20:00 ，那么最终执行的代码为：

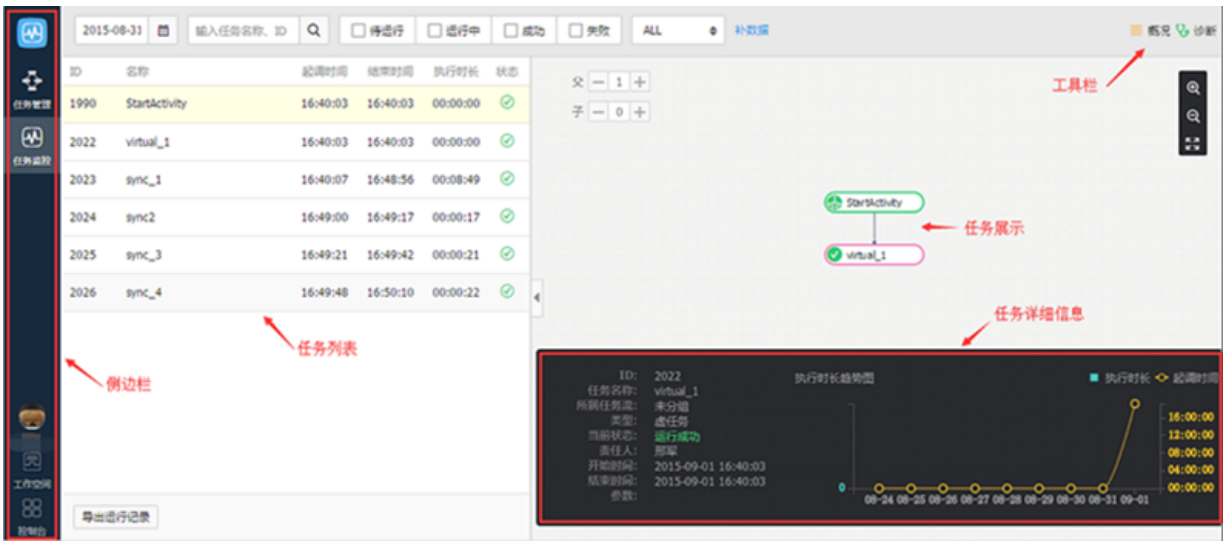
```
SELECT * FROM table_a WHERE t_date = '2014-06-02 13:20:00'
```

33.3.4 监控任务

33.3.4.1 任务监控界面介绍

单击侧边栏中的**任务监控**按钮，进入任务监控界面，如图 *1670:* 任务监控界面所示。

图 1670: 任务监控界面



侧边栏

通过单击侧边栏的**任务监控**按钮，进入任务监控界面。

工具栏

工具栏主要提供两个功能：

- 概况：对当前工作空间下的成功、运行中、待运行、失败各类状态任务进行统计展现；
- 诊断：对诊断进行创建和管理。

任务列表

任务列表中展示了任务的 ID、名称、起调时间、结束时间、执行时长和状态等。

任务列表上方的搜索框可以根据日期、任务名称、ID、执行状态和任务流等不同条件对任务进行筛选。

任务展示

在任务列表右侧的面板中，以可视化的方式展示了任务的名称、执行状态和依赖关系等。

可以选中目标任务，单击右键，在右键菜单中选择相应功能对任务进行操作。

通过面板左上角的任务父子层级控件来调整当前面板中所展示的任务数量。

任务详细信息

单击任务展示面板中的任务，面板下方的详细信息会相应改变。

该区域展示了任务基本信息、执行时长历史和起调时间历史等。

33.3.4.2 查看任务执行历史

在任务监控面板中选中目标任务，即会在面板下方展示该任务的执行历史，包括任务基本信息、执行时长趋势和起调时间趋势，如图 *1671:* 查看任务执行历史所示。

图 1671: 查看任务执行历史



33.3.4.3 查看同步任务日志

操作步骤

1. 右键单击目标同步任务，选择**查看任务运行日志**，如图 *1672:* 查看运行日志所示。

   图 1672: 查看运行日志



2. 在弹出窗口中查看运行日志和同步日志，如图 *1673:* 运行日志和图 *1674:* 同步日志所示。

   运行日志记录的是任务调度引擎的明细信息；同步日志是同步程序在执行数据抽取和加载过程的明细信息。

   图 1673: 运行日志

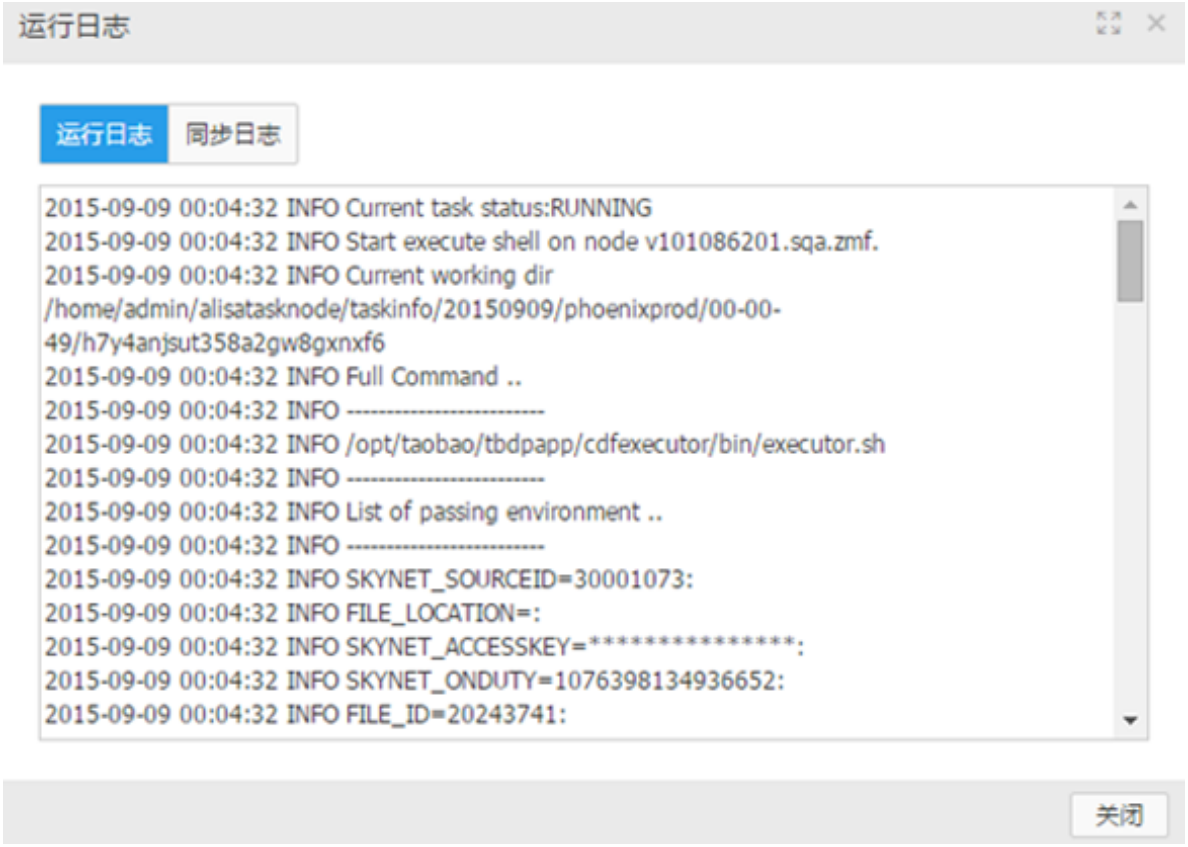
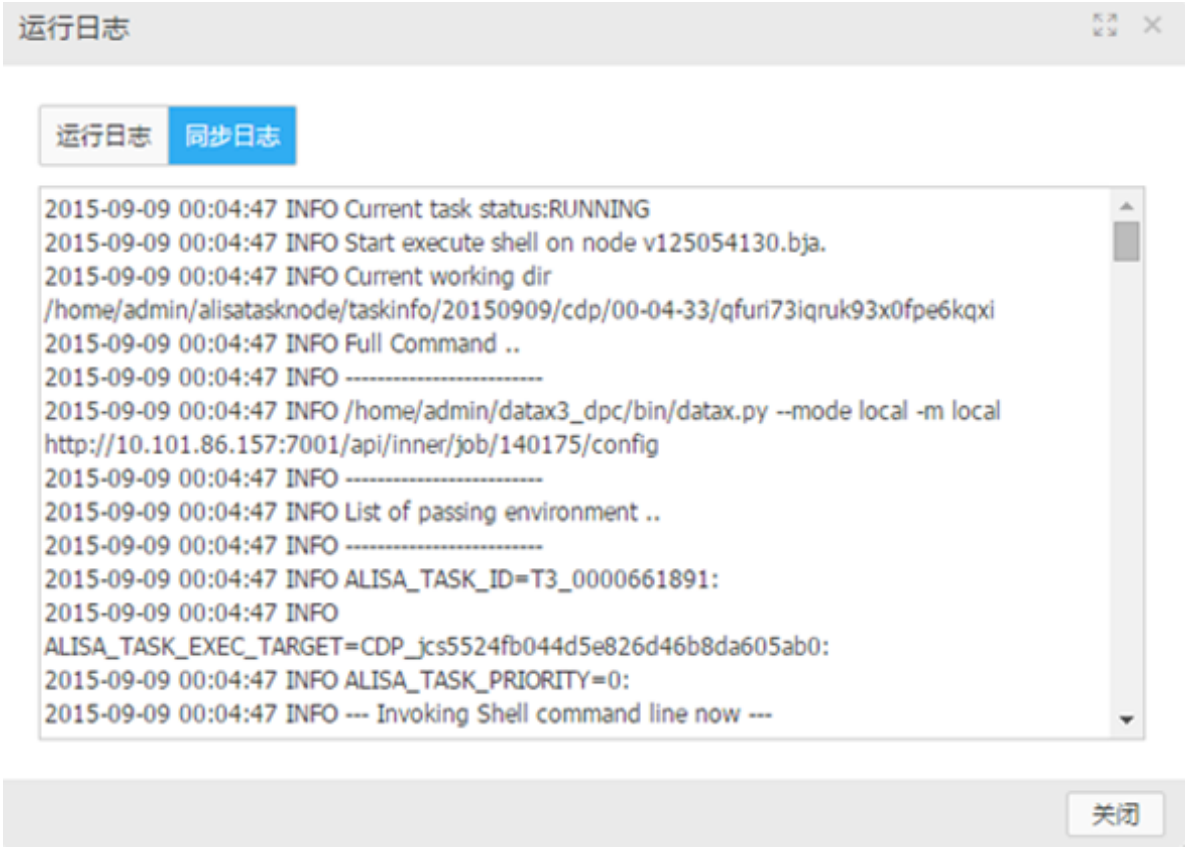

运行日志
运行日志
同步日志
2015-09-09 00:04:32 INFO Current task status:RUNNING
2015-09-09 00:04:32 INFO Start execute shell on node v101086201.sqa.zmf.
2015-09-09 00:04:32 INFO Current working dir
/home/admin/alisatasknode/taskinfo/20150909/phoenixprod/00-00-
49/h7y4anjsut358a2gw8gxnxf6
2015-09-09 00:04:32 INFO Full Command ..
2015-09-09 00:04:32 INFO ----------
2015-09-09 00:04:32 INFO /opt/taobao/tbdpapp/cdfexecutor/bin/executor.sh
2015-09-09 00:04:32 INFO ----------
2015-09-09 00:04:32 INFO List of passing environment ..
2015-09-09 00:04:32 INFO ----------
2015-09-09 00:04:32 INFO SKYNET_SOURCEID=30001073:
2015-09-09 00:04:32 INFO FILE_LOCATION=:
2015-09-09 00:04:32 INFO SKYNET_ACCESSKEY=**********:
2015-09-09 00:04:32 INFO SKYNET_ONDUTY=1076398134936652:
2015-09-09 00:04:32 INFO FILE_ID=20243741:
关闭

图 1674: 同步日志



33.3.4.4 查看刷新任务日志

操作步骤

1. 右键单击目标刷新任务，选择**查看任务运行日志。**
2. 在弹出的窗口中查看运行日志。

33.3.4.5 查看检测任务日志

操作步骤

1. 右键单击目标检测任务，选择**查看任务运行日志。**
2. 在弹出的窗口中查看运行日志。

33.3.4.6 重跑任务

操作步骤

右键单击目标任务，选择**重跑**，可以重跑任务，如图 *1675:* 重跑任务所示。

图 1675: 重跑任务



33.3.4.7 查看补数据任务状态

操作步骤

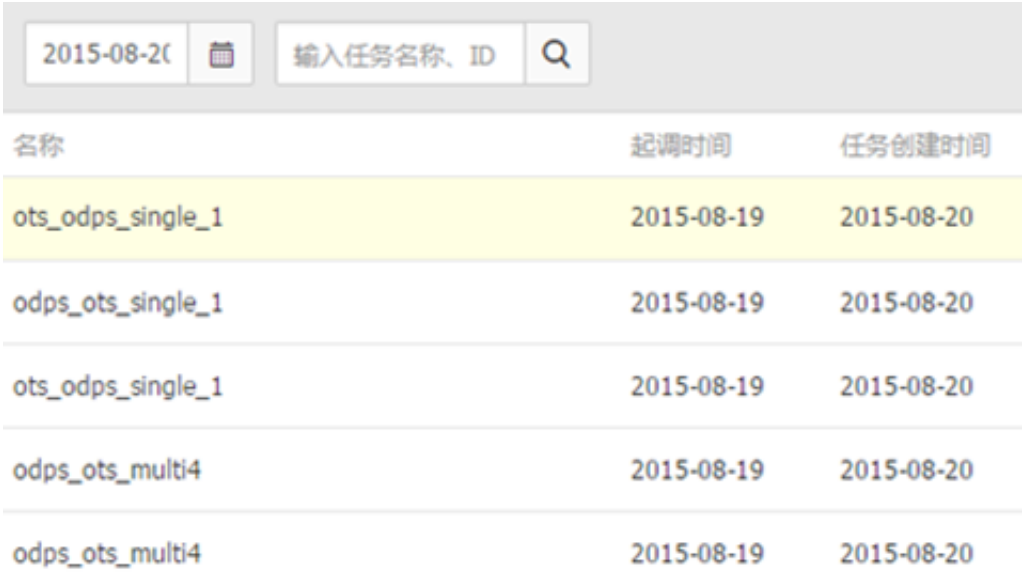
1. 在任务监控面板中，单击顶部的**补数据**链接，如图 *1676:* 补数据链接所示。

   图 1676: 补数据链接



2. 在新窗口中按运行时间进行和任务名称、ID 进行查询，查看任务状态，如图 *1677:* 查看任务状态所示。

   图 1677: 查看任务状态



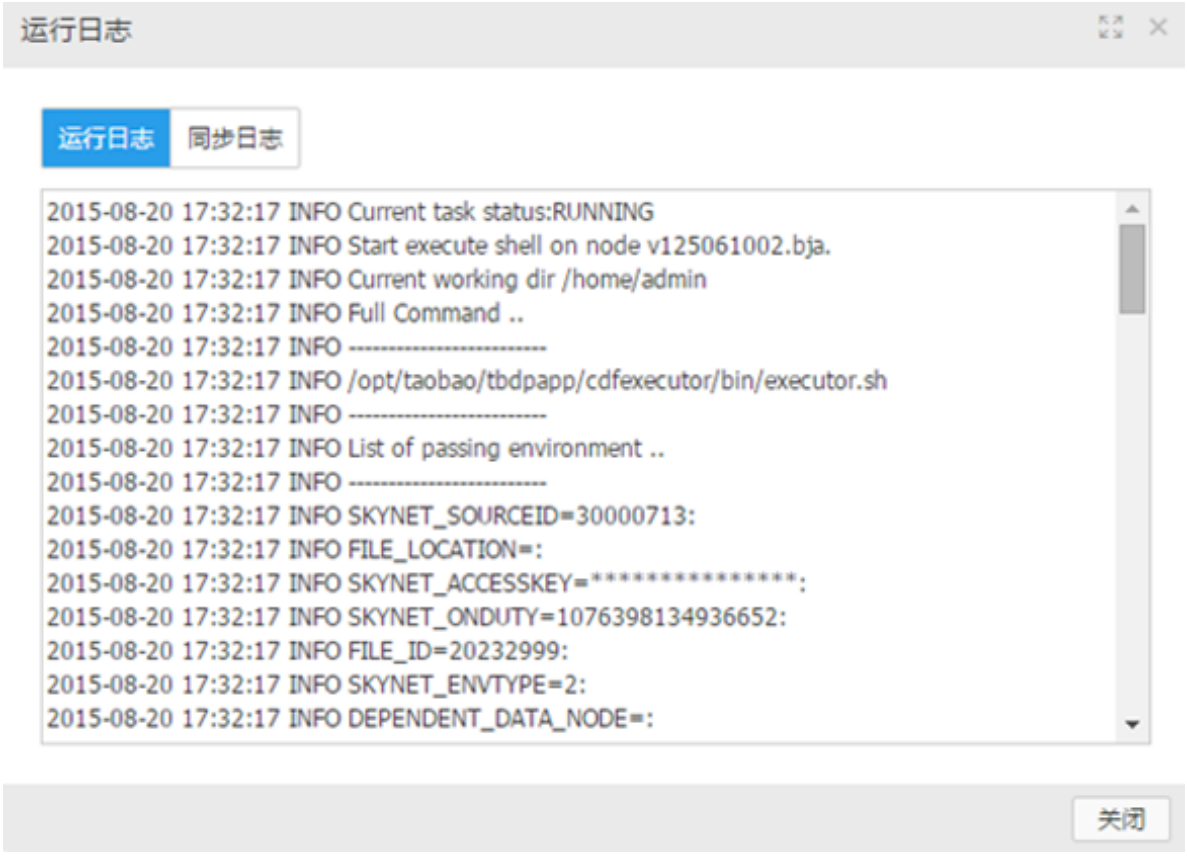
3. 右键单击目标任务，选择**查看任务运行日志**，如图 *1678:* 右键单击任务所示。

图 1678: 右键单击任务



4. 在弹出窗口中查看运行日志，如图 *1679:* 查看运行日志所示。

图 1679: 查看运行日志



33.3.4.8 导出运行日志

背景信息

任务监控中可以将某一日期的运行记录导出成 CSV 格式的文件至本地机器。

操作步骤

1. 在浏览窗口中，选择目标日期，如图 *1680:* 选择日期所示。

图 1680: 选择日期



2. 单击**导出运行记录**。

   导出结果如图 *1681:* 导出结果所示。

   图 1681: 导出结果

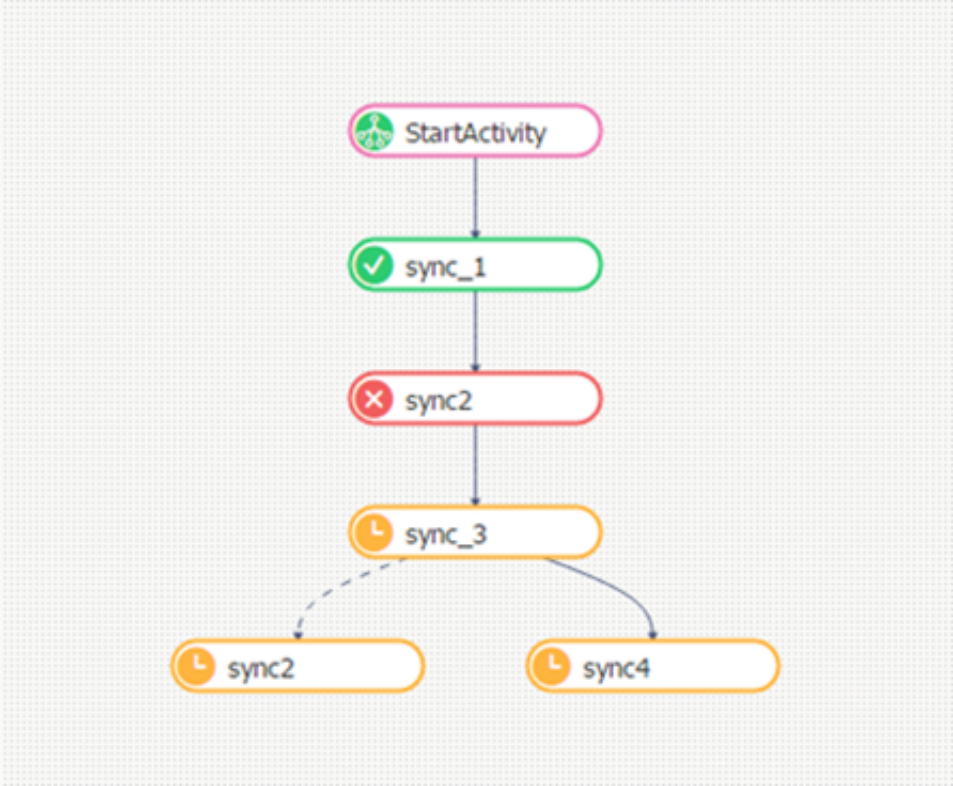
	A	B	C	D	E	F
1	id	name	start_time	end_time	executior	status
2	2062	StartActivity	2015/9/9 0:00	2015/9/9 0:00		SUCCESS
3	2071	sync_1	2015/9/9 15:00	2015/9/9 15:00		SUCCESS
4	2072	sync2	2015/9/9 15:00	2015/9/9 15:00	0:00:01	FAILURE
5	2073	sync_3				NOT_RUN
6	2074	sync4				NOT_RUN
7	2075	sync_consume_recor	2015/9/9 0:03	2015/9/9 0:03	0:00:01	FAILURE
8	2076	rt_mysql_odps	2015/9/9 0:00	2015/9/9 0:01	0:00:32	SUCCESS
9	2078	ocean_odps	2015/9/9 0:00	2015/9/9 0:01	0:00:23	SUCCESS
10	2079	sync_rongtian_test_	2015/9/9 0:04	2015/9/9 0:05	0:00:31	SUCCESS
11	2080	ots_odps_success1	2015/9/9 0:01	2015/9/9 0:04	0:02:58	SUCCESS
12	2081	sync_consume_recor	2015/9/9 0:00	2015/9/9 0:01	0:00:17	FAILURE
13	2082	odps_ots_single_su	2015/9/9 0:04	2015/9/9 0:05	0:00:37	SUCCESS
14	2083	sync_consume_recor	2015/9/9 0:01	2015/9/9 0:06	0:05:26	FAILURE
15	2084	ots_multi_odps_suc	2015/9/9 0:01	2015/9/9 0:02	0:00:45	SUCCESS
16	2085	odps_ots_multi_suc	2015/9/9 0:03	2015/9/9 0:04	0:00:57	SUCCESS
17	2086	sync_2_odps	2015/9/9 0:02	2015/9/9 0:04	0:02:39	SUCCESS
18	2087	sync_2_oss	2015/9/9 15:37	2015/9/9 15:37		SUCCESS
19	2088	yangqx_detect	2015/9/9 14:53	2015/9/9 14:54	0:00:31	FAILURE

33.3.4.9 跳过任务

背景信息

在任务监控中的面板中，我们可以看到由于任务的依赖关系，如果中间节点任务执行失败，会导致子任务没有运行，如图 *1682:* 任务失败所示。

图 1682: 任务失败

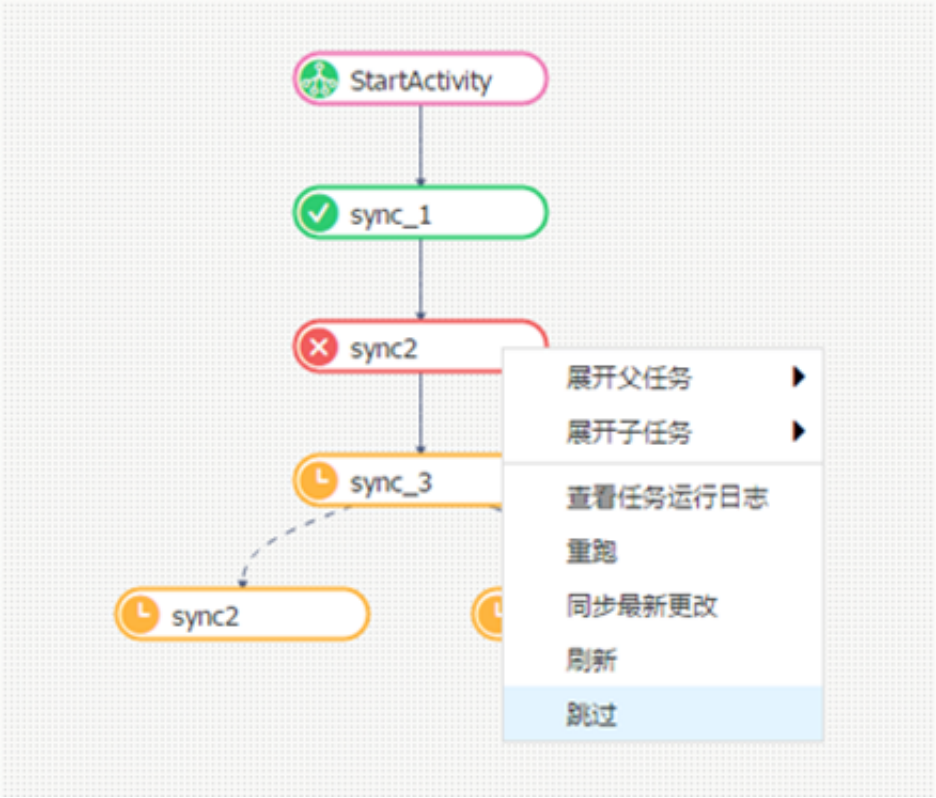


若希望可以执行子任务，您可以设置跳过执行失败的中间节点。

操作步骤

右键单击失败的任务，选择**跳过**，如图 *1683:* 跳过任务所示。

图 1683: 跳过任务



33.3.4.10 同步最新更改

背景信息

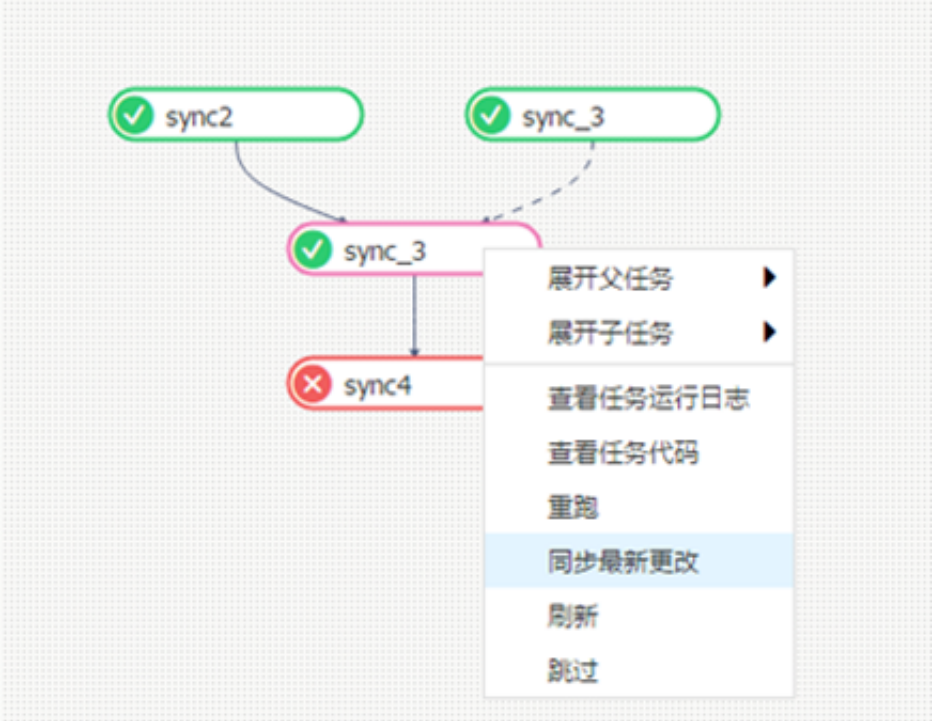
若用户在任务管理中对任务进行了更改，比如重新设置了起调时间，任务不会立刻生成可运行的实例。此时，在任务监控中执行任务重跑不会使用最新更改的任务来执行。

为了使任务的更新立刻生效，可以通过同步最新更改来执行最新定义的任务。

操作步骤

1. 右键单击目标任务，选择**同步最近更新**，如图 *1684:* 同步最新更改所示。

图 1684: 同步最新更改



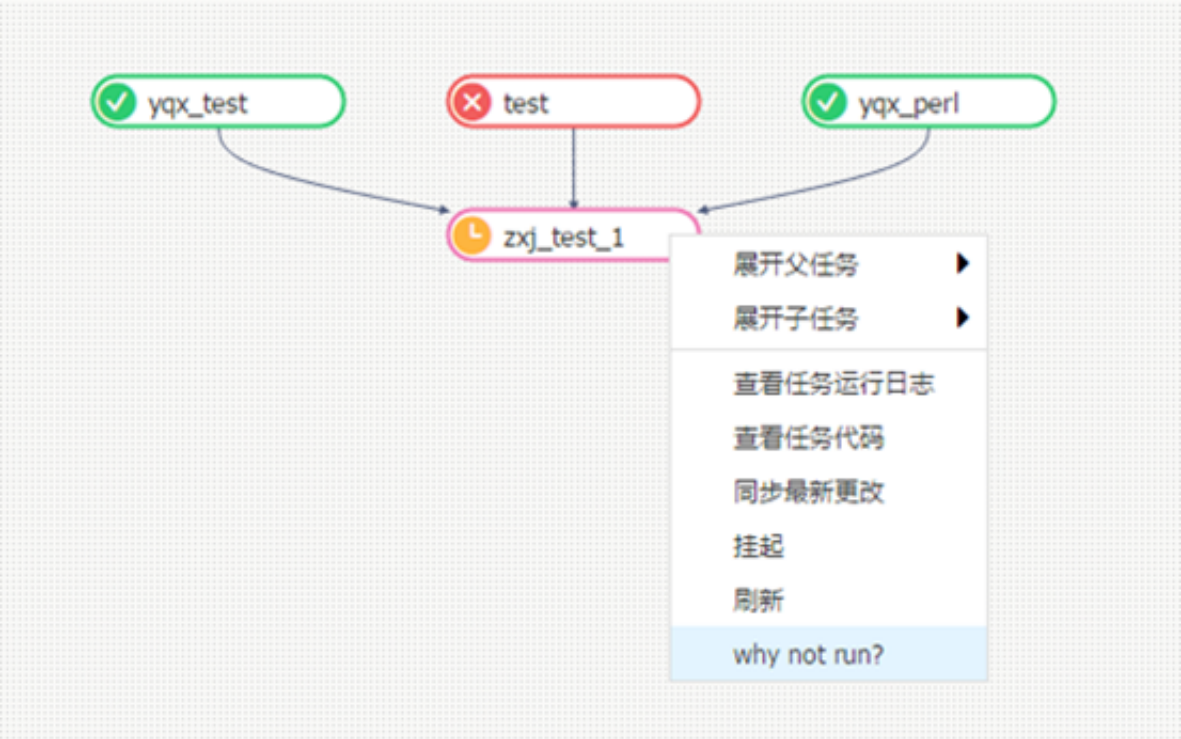
2. 此时更新后的实例生成，重跑任务可以执行最新定义的任务。

33.3.4.11 查看未执行的诊断

操作步骤

1. 对于未执行的任务，右键单击该任务，选择**why not run?**，查看任务未执行的诊断，如图 *1685: 查看未执行的诊断*所示。

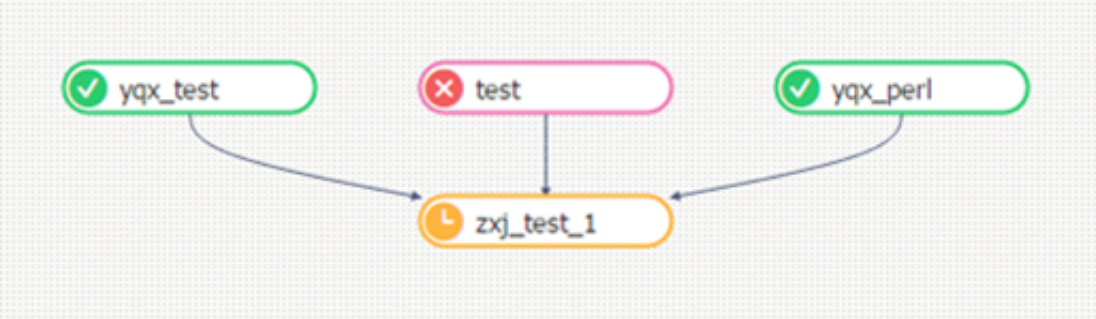
   图 1685: 查看未执行的诊断



2. 完成诊断后，面板将定位到导致任务没有未执行的根源，即在依赖关系链路上最早失败的任务。

   如图 *1686:* 定位失败任务所示，面板中的任务关系图聚焦到 test 任务。

   图 1686: 定位失败任务



33.3.4.12 新建诊断

背景信息

用户可以通过诊断，找出执行时长或起调时间异常的任务进行跟踪和分析，以便优化任务调度。

操作步骤

1. 在页面右上角的**诊断**中，单击**管理**，如图 *1687:* 诊断页面所示。

   图 1687: 诊断页面



2. 在弹出的窗口，单击**新建诊断**，如图 *1688:* 新建诊断所示。

图 1688: 新建诊断



3. 填写诊断名称，选择诊断项目为执行时长或起调时间，最后选择对比基准，共有对比前一天、对比上周同一天和对比最近七天平均三个选项，如图 *1689:* 设置诊断信息所示。

筛选条件可以选择增量超过、减量超过、大于、小于、降幅百分比超过、增幅百分比超过等。根据需求填写筛选条件的数值，并选择合适的单位（分钟、小时、百分比等）。

图 1689: 设置诊断信息



4. 单击**保存**。

33.3.4.13 使用诊断

操作步骤

1. 在页面右上角的**诊断**窗口，选择需要使用的诊断，如图 *1690:* 选择需要使用的诊断所示。

图 1690: 选择需要使用的诊断



2. 单击诊断后，左侧浏览窗口将展示所有符合该诊断的所有任务，如图 *1691:* 查看任务所示。

图 1691: 查看任务

ID	名称	起调时间	结束时间	执行时长	状态
2853	yqx_test_2015032111w	00:10:03	00:10:12	00:00:09	
4929	20150310	00:15:40	00:16:39	00:00:59	
7050	sync_m_security_users	01:10:03	01:10:20	00:00:17	
7048	sync_m_security_object_acl	01:10:22	01:11:15	00:00:53	
7231	sync_odps_projects_mysqlTo...	00:07:01	00:07:28	00:00:27	
6465	sync_yqx_0806	00:06:55	00:07:28	00:00:33	
6453	20150806sync_test_cdp_imp...	00:06:58	00:07:31	00:00:33	
6454	20150806sqlserver	00:06:03	00:06:26	00:00:23	

33.3.4.14 修改诊断

操作步骤

1. 在页面右上角的**诊断**中，单击**管理**，如图 *1692:* 诊断页面所示。

图 1692: 诊断页面



2. 在诊断列表中选择需要编辑的诊断，单击右侧的**编辑**按钮，如图 *1693:* 编辑诊断所示。

图 1693: 编辑诊断



进入编辑页面后，按照需求填写相应字段。

33.3.4.15 删除诊断

操作步骤

1. 在页面右上角的**诊断**中，单击**管理**，如图 *1694:* 诊断页面所示。

图 1694: 诊断页面



2. 在诊断列表中，选择需要删除的诊断，单击右侧的**删除**按钮，如图 *1695:* 删除诊断所示。

完成删除。

图 1695: 删除诊断



33.3.5 任务报警

33.3.5.1 配置模板

任务报警支持创建模板，选择每个渠道，进入模板编辑页面。

33.3.5.1.1 查看模板

操作步骤

1. 在**任务报警**窗口，选择**模板管理**页签。
2. 单击**邮件模板**，如图 *1696: 查看模板*所示。

   可以查看邮件的所有模板。

   图 1696: 查看模板



33.3.5.1.2 新增模板

背景信息

支持创建四种渠道的模板，选择每个渠道，进入模板编辑页面，本节以邮件渠道的模板编辑为例，其他渠道操作方法一样。

操作步骤

1. 在**邮件模板**页面，单击**新增模板**，创建新的模板。
2. 在**编辑模板**窗口，如图 *1697: 编辑模板信息*所示，输入以下信息：
   - 模板名称：建议创建一个唯一的名字

- 报警类型：当前支持以下五种规则类型：
  - 运行成功
  - 运行失败
  - 超时未开始
  - 超时未结束
  - 运行时长超时
- 报警标题：当渠道是邮件和旺旺时，要填写标题
- 邮件内容：编写邮件模板内容，支持参数替换

图 1697: 编辑模板信息



3. 单击**保存**。

33.3.5.1.3 修改模板

背景信息

模板内容支持参数替换，可以显示更个性化的信息。

参数用 **${参数名}** 表示，当前支持的参数如下：

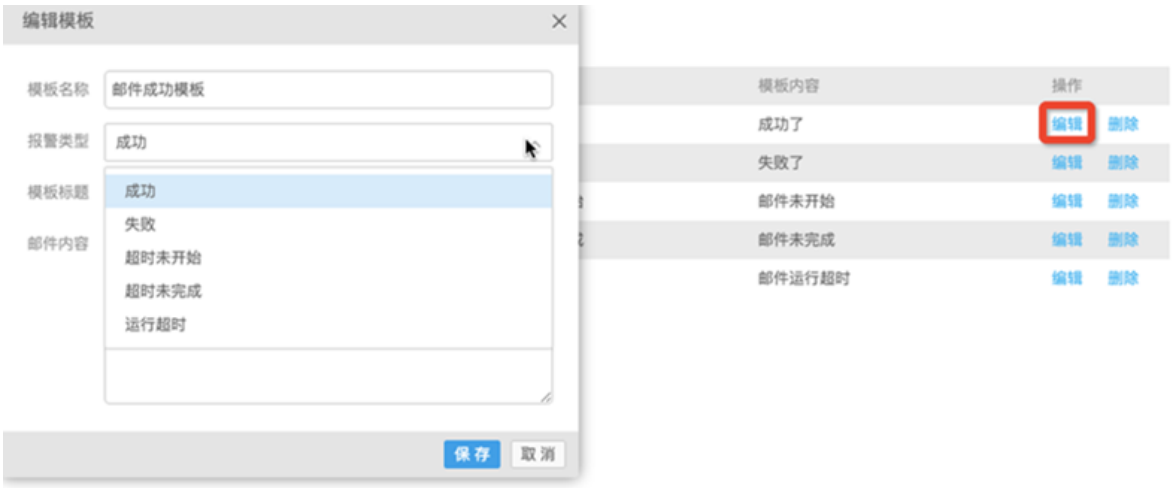
- userRealName：用户的名称
- taskId：任务编号
- taskName：任务名称
- taskInstId：任务实例编号
- taskInstBeginRunTime：任务开始运行时间
- taskInstFinishTime：任务结束运行时间
- alarmLevel 有三种级别：
  - 严重
  - 重要
  - 提醒
- alarmType 有六种类型：
  - 成功
  - 失败
  - 启动延迟
  - 完成延迟
  - 运行超时
  - 等待资源超时

操作步骤

1. 在模板列表中，选择待编辑的模板，单击**编辑**，如图 *1698:* 编辑模板所示。

   进入模板编辑界面。

   图 1698: 编辑模板



2. 在**编辑模板**窗口，根据需要编辑相关信息。
3. 单击**保存**。

33.3.5.1.4 删除模板

操作步骤

1. 在模板列表中，选择不需要的模板。
2. 单击**删除**，删除模板。

33.3.5.2 配置报警规则

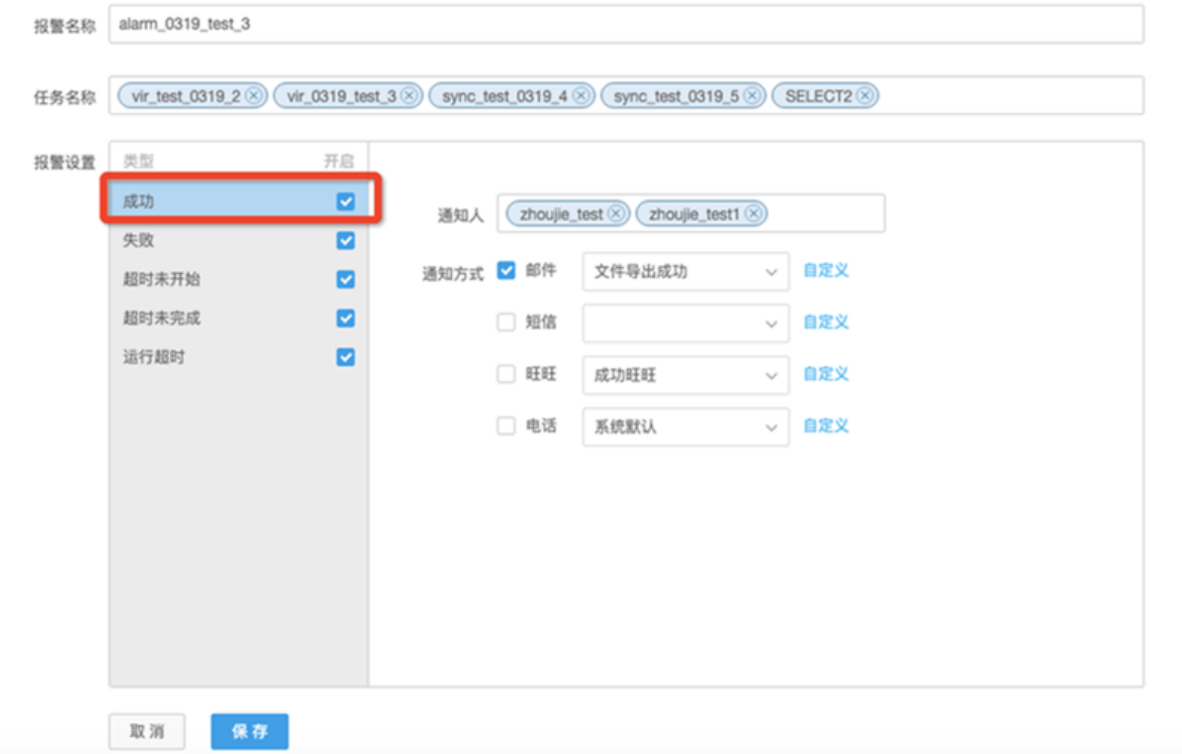
任务报警功能支持创建报警规则，监控任务的执行情况，针对各种运行异常，多渠道及时发出警告。

33.3.5.2.1 配置成功报警规则

操作步骤

1. 在**任务报警**中，选择**监控列表**页签，进入报警管理列表。
2. 单击**新增报警**。
3. 在编辑页面，配置报警接收人和相应报警渠道模板，如图 *1699:* 配置报警成功规则所示。

图 1699: 配置报警成功规则



其中：

- 报警名称：建议创建唯一的名称。
- 报警任务：本次报警的任务的集合。
- 报警规则：本次报警的规则。

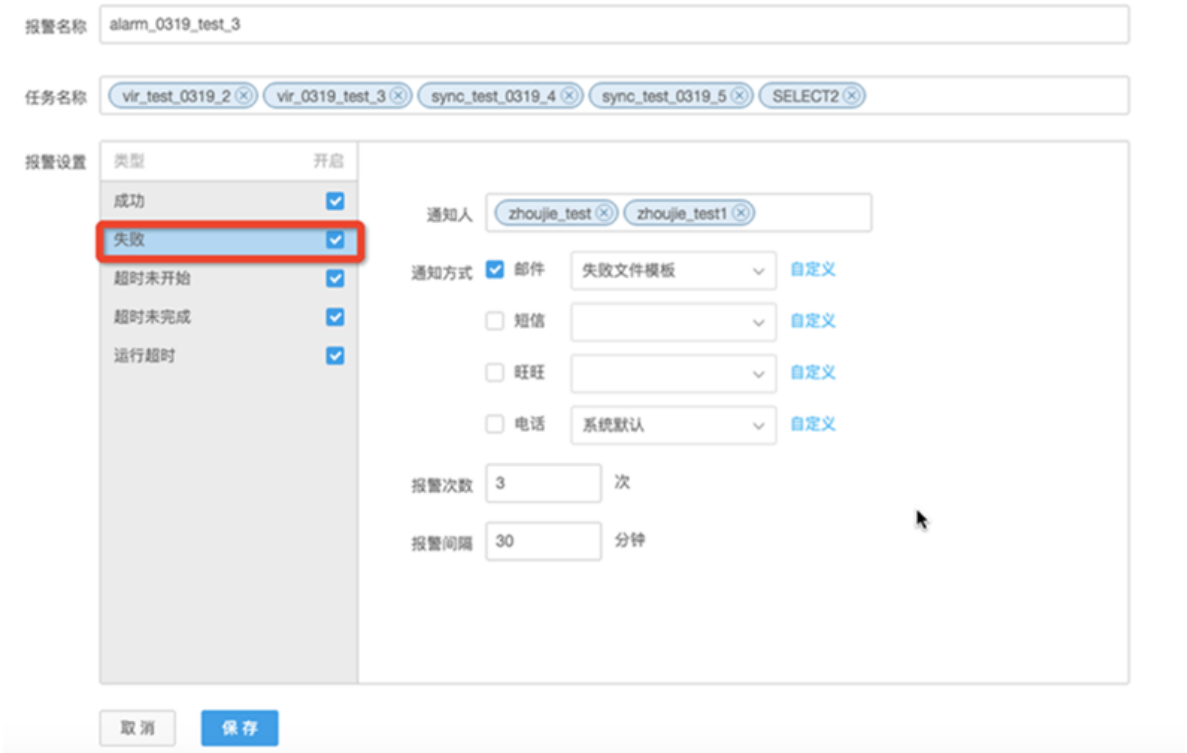
4. 设置完成后，单击**保存**。

33.3.5.2.2 配置失败报警规则

操作步骤

1. 在编辑页面，配置报警接收人、报警渠道的模板，最大报警次数和报警间隔时长，如图 *1700:* 配置失败报警规则所示。

图 1700: 配置失败报警规则



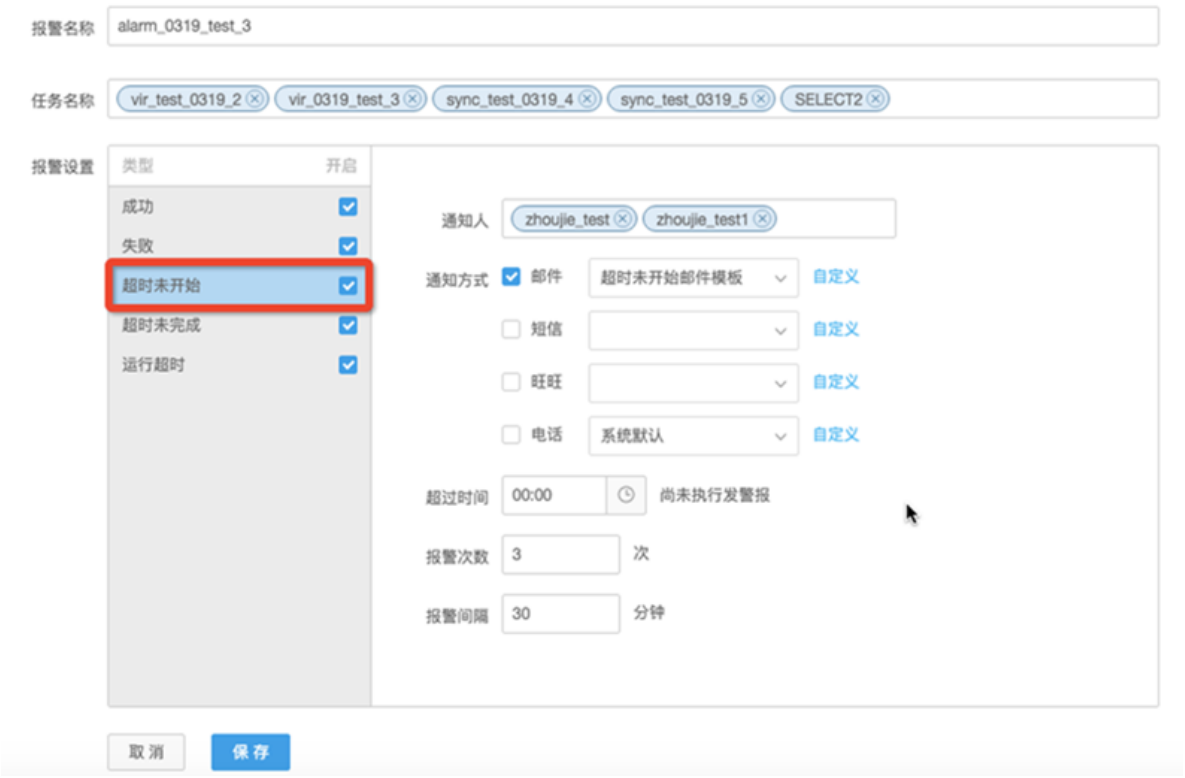
2. 配置完成后，单击**保存**。

33.3.5.2.3 配置超时未开始规则

操作步骤

1. 在编辑页面，配置报警接收人、报警渠道的模板、检查时间、最大报警次数和报警间隔时长，如图 *1701:* 配置超时未开始规则所示。

图 1701: 配置超时未开始规则



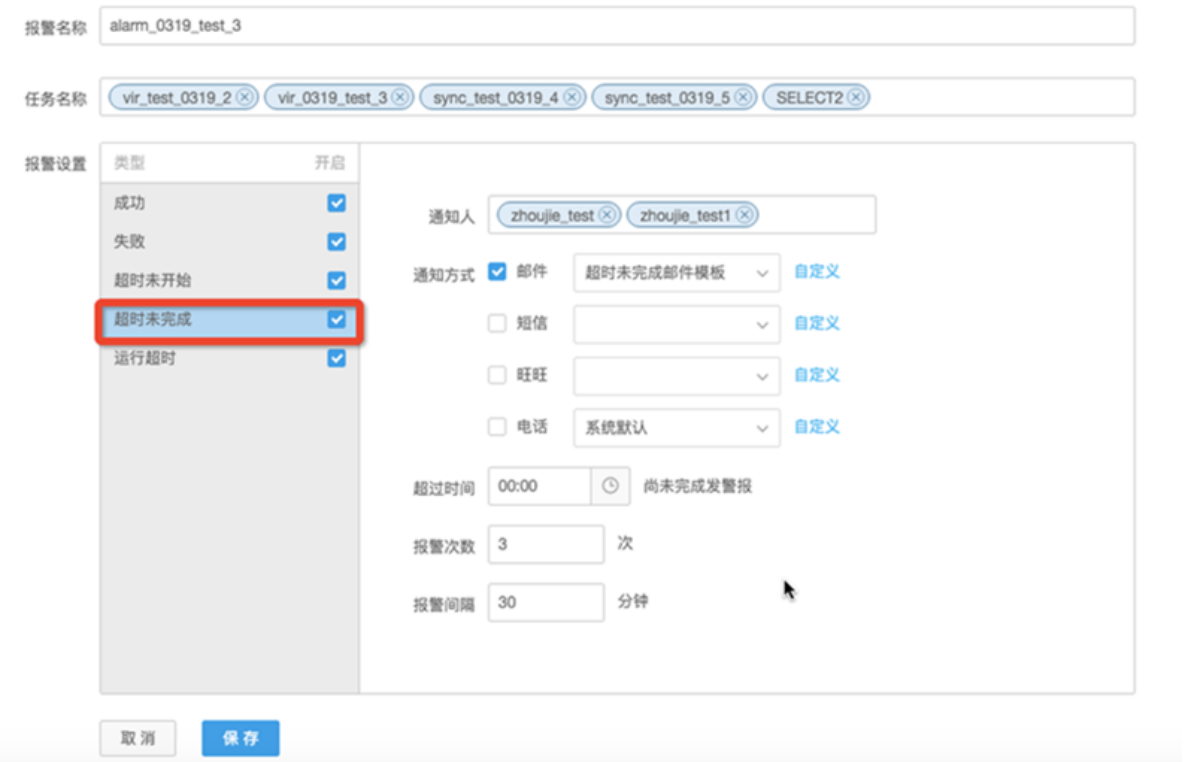
2. 配置完成后，单击**保存**。

33.3.5.2.4 配置超时未结束规则

操作步骤

1. 在编辑页面，配置报警接收人、报警渠道的模板、检查时间、最大报警次数和报警间隔时长，如图 *1702:* 配置超时未结束规则所示。

图 1702: 配置超时未结束规则



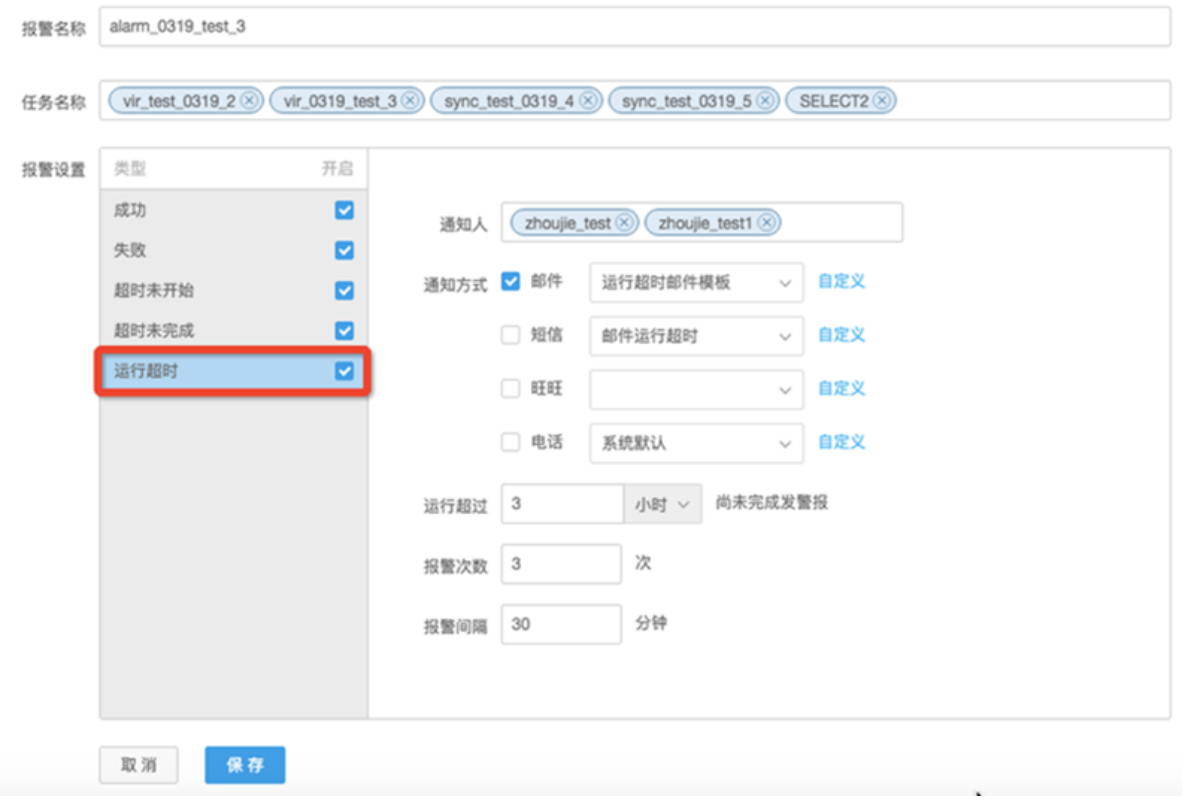
2. 配置完成后，单击**保存**。

33.3.5.2.5 配置运行超时规则

操作步骤

1. 在编辑页面，配置报警接收人、报警渠道的模板、运行超时时间、最大报警次数和报警间隔时长，如图 *1703:* 配置运行超时规则所示。

图 1703: 配置运行超时规则



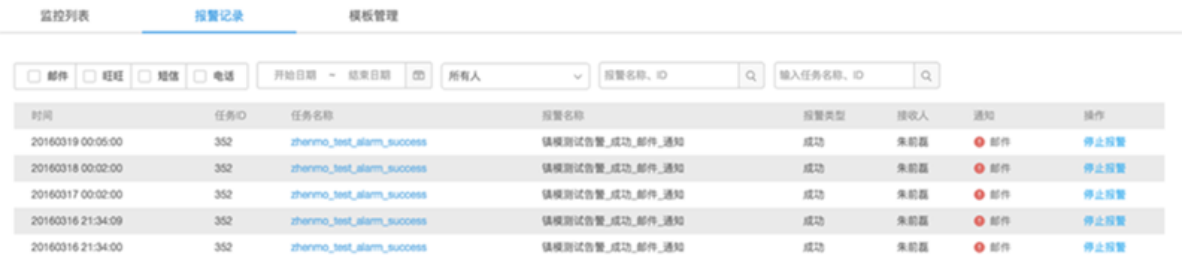
2. 配置完成后，单击**保存**。

33.3.5.3 查看报警记录

操作步骤

1. 在**任务报警**窗口，选择**报警记录**页签，如图 *1704:* 查看报警记录所示。

图 1704: 查看报警记录



2. 查看所有报警记录。

33.3.6 FAQ

1. 问：任务的参数如何配置？

回答：请参见**任务调度服务 > 任务管理。**

2. 问：是否支持实时调度？例如到小时级或者分钟级？

   回答：支持，最小5分钟。

3. 问：是否支持外部接口调度采云间的任务？

   回答：目前支持外部调用 HTTP 的 Web Service 接口触发采云间任务，具体配置方式请参见**任务调度服务 > 检测任务。**

4. 问：是否支持手工任务和一次性任务？

   回答：当前支持手工任务，一次性任务可以通过手工任务实现。

5. 问：任务权限的控制，不能修改别人的任务？

   回答：本系统支持权限管理，有修改权限的才能修改任务。同一个项目下的任务，项目组成员可以修改，私有任务只能自己修改。

33.4 数据分析

33.4.1 数据分析服务简介

数据分析服务DAS （Data Analysis Service），是一个云数据敏捷分析平台，可以连接云计算MaxCompute、RDS、 Analytic DB数据源，用您熟悉的方式进行数据透视分析、制作报表、设计自助取数工具、设计数据产品，简单易用，使人人都有使用数据的能力。

数据分析服务主要由如下几部分组成：

- 数据源的连接和管理
- 数据集配置与管理
- 数据分析报表配置

33.4.2 进入数据分析

操作步骤

1. 单击大数据左侧导航栏中的**工作台**，如图 *1705:* 工作台所示。

   图 1705: 工作台



2. 在屏幕中间的**我的应用**中，单击**数据分析**。

33.4.3 连接数据源

支持三种数据源，即 MaxCompute、Analytic DB（Garuda）和 RDS（MySQL），其中，MaxCompute 和 RDS 可以直接连接； AnalyticDB 和 RDS 通过加速功能，可以间接实现连接。

33.4.3.1 连接MaxCompute数据源

做数据分析之前，首先需要连接数据源。

背景信息

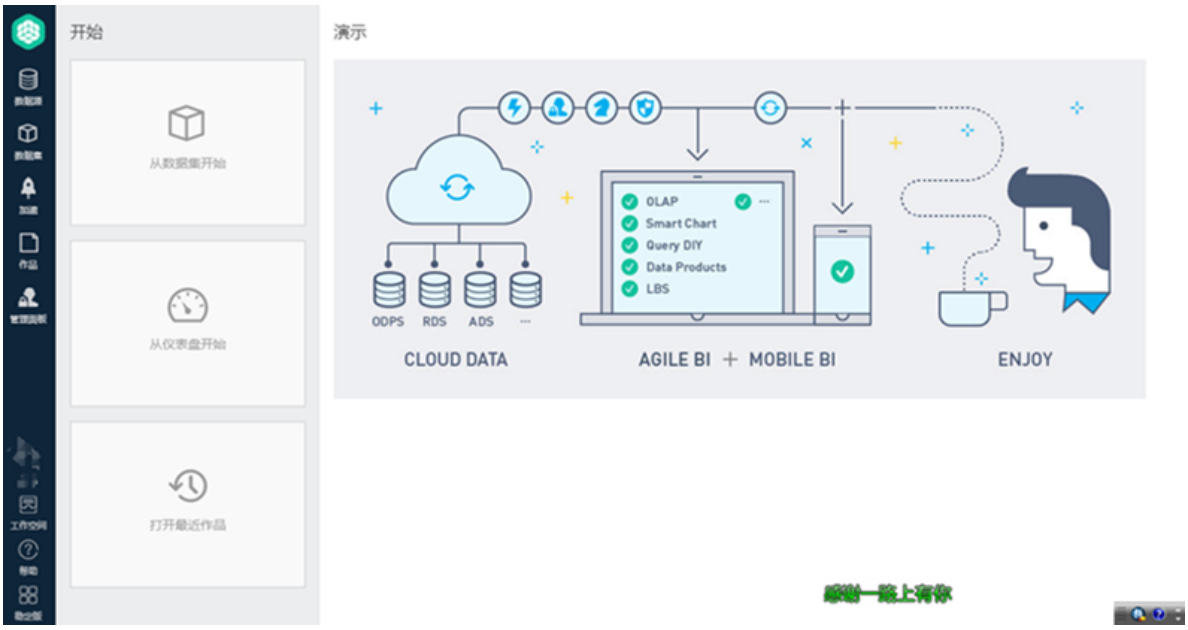
目前支持三种数据源，即MaxCompute（原ODPS）、AnalyticDB（Garuda）和RDS（MySQL），其中，MaxCompute（原ODPS）和RDS可以直接连接；AnalyticDB和RDS通过加速功能，也可以间接实现连接。

下面以MaxCompute（原ODPS）类型的数据源为例，为大家简单介绍一下，如何创建一个数据源连接。

操作步骤

1. 打开数据分析应用首页，如图 *1706:* 数据分析应用首页所示。

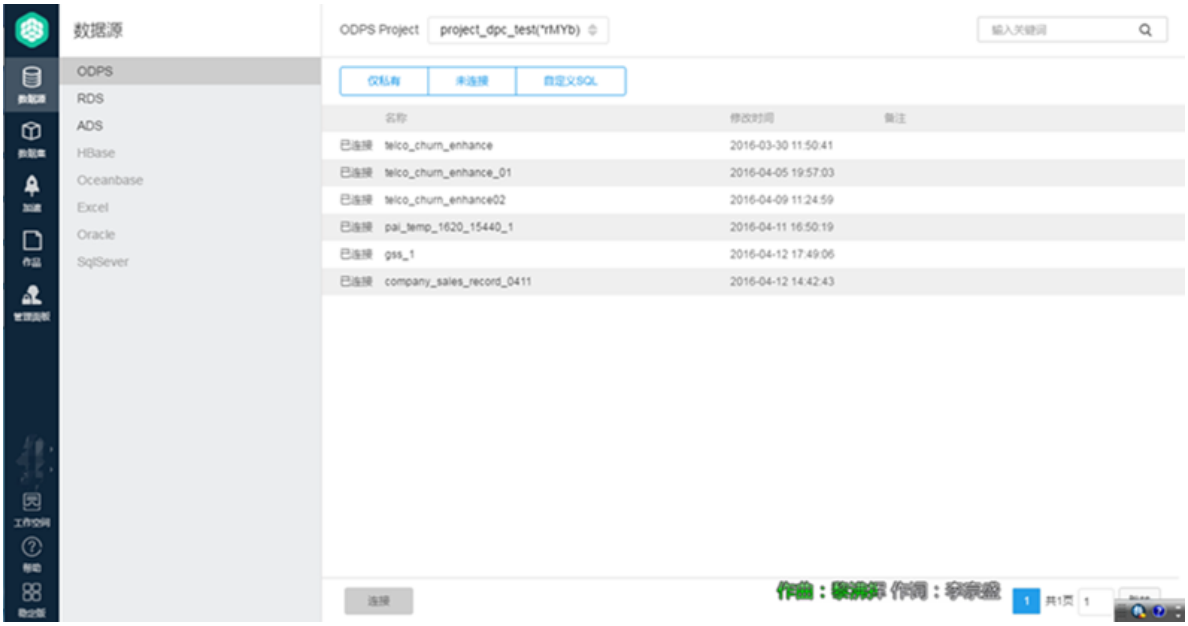
图 1706: 数据分析应用首页



2. 单击侧边栏中的**数据源**，并选择对应的数据源类型。

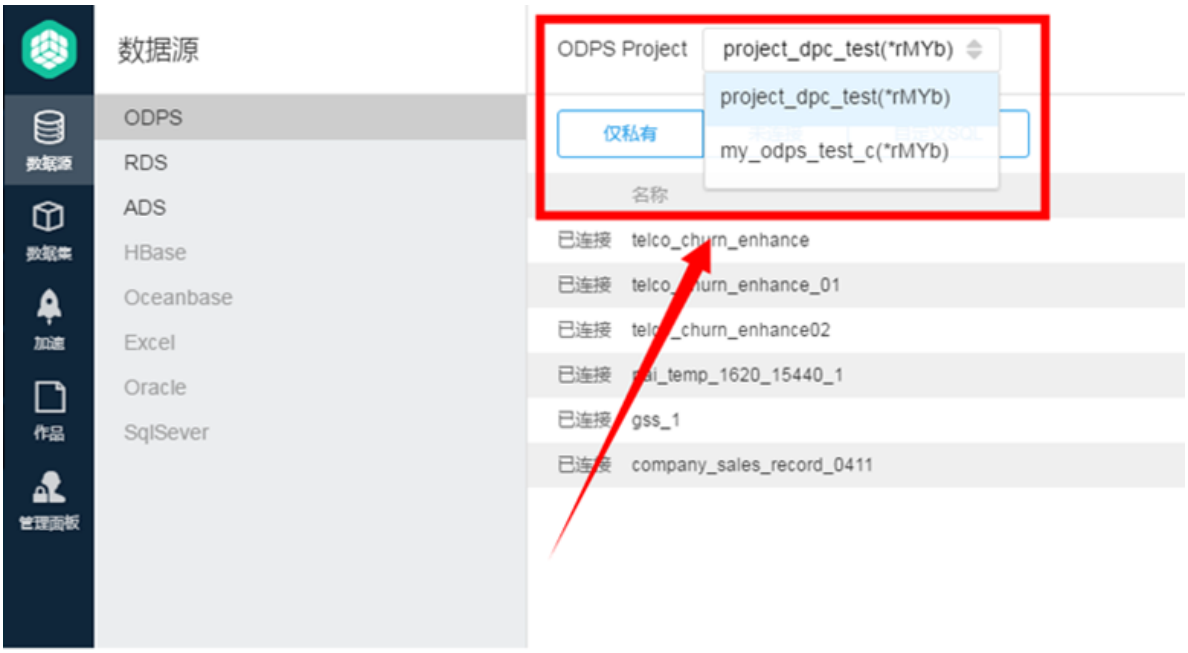
   目前数据分析支持MaxCompute（原ODPS）、RDS、AnalyticDB等几种数据源，可以通过选择类型，查看对应的已连接数据源，此处选择**ODPS**，如图 *1707: 选择数据源*所示。

图 1707: 选择数据源



3. 单击展开 **ODPS Project** 下拉框，可以看到已有的project，如图 *1708: 选择project*所示。

图 1708: 选择project



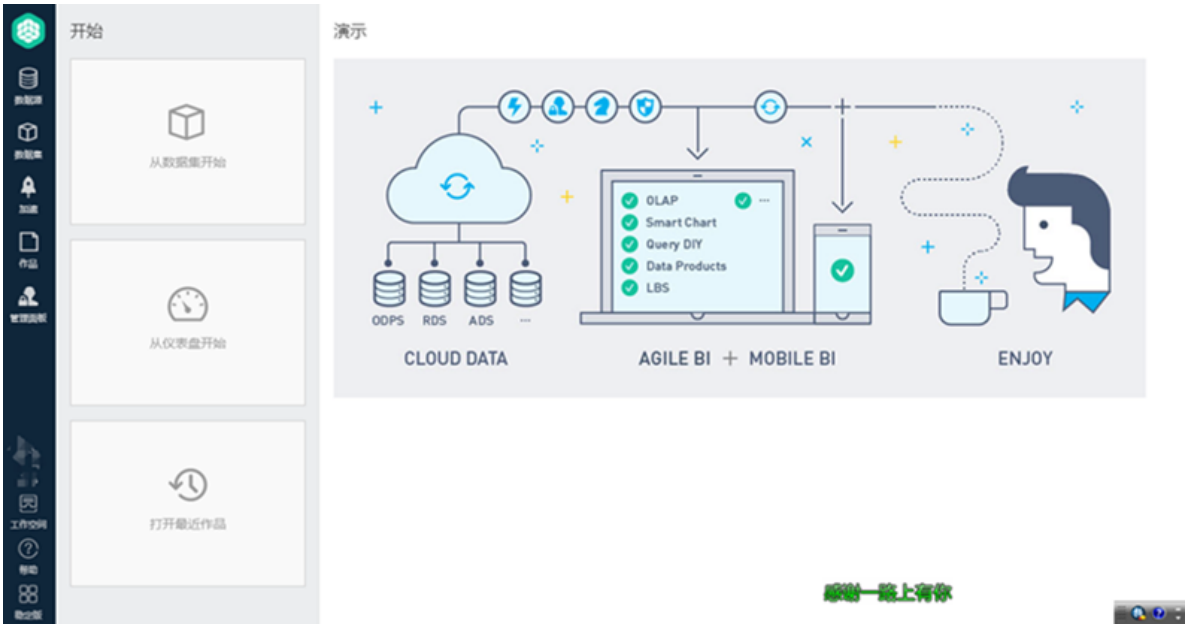
4. 选择相应的 project ，下方表格中关联显示该 project 下的表信息。
5. 选择需要分析的表，单击**连接**。

33.4.3.2 连接 RDS 数据源

操作步骤

1. 打开数据分析应用首页，如图 *1709:* 数据分析应用首页所示。

图 1709: 数据分析应用首页



2. 单击侧边栏中的**数据源**，并选择对应的数据源类型。

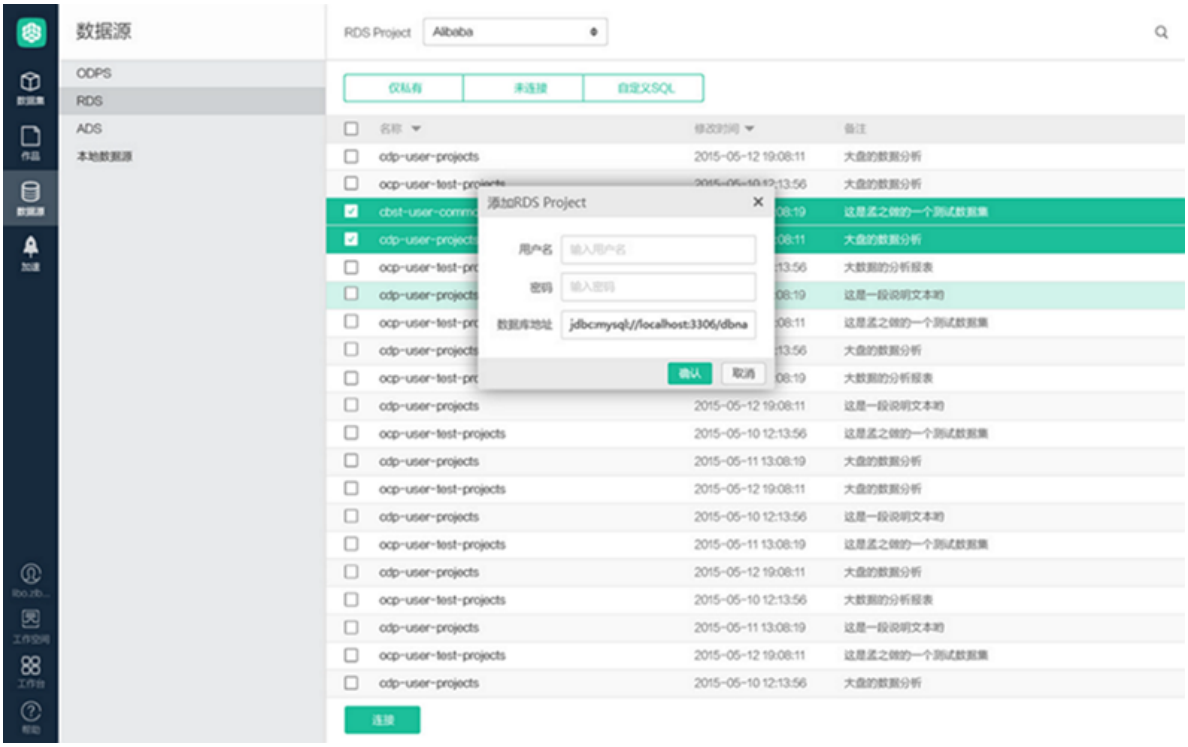
   目前数据分析支持 MaxCompute（原ODPS）、RDS、Analytic DB（原ADS） 等几种数据源，可以通过选择类型，查看对应的已连接数据源，此处选择 **RDS**。

   在左侧主页面显示已经连接的 RDS Project 及表信息。

3. 展开 **RDS Project** 下拉框，单击末尾的**添加 Project** 按钮，并在弹出的**添加 RDS Project** 连接配置窗口，填写需要连接的数据源信息，如图 *1710: 添加 RDS Project*所示。

   其中，用户名和密码等，如果不知道相关信息，请联系您的数据仓库管理员获取。

   图 1710: 添加 RDS Project



4. 单击**确认**。

   数据源信息校验成功后，会提示成功并自动加载对应的 RDS 下的数据表信息。

33.4.3.3 通过自定义 SQL 创建数据集

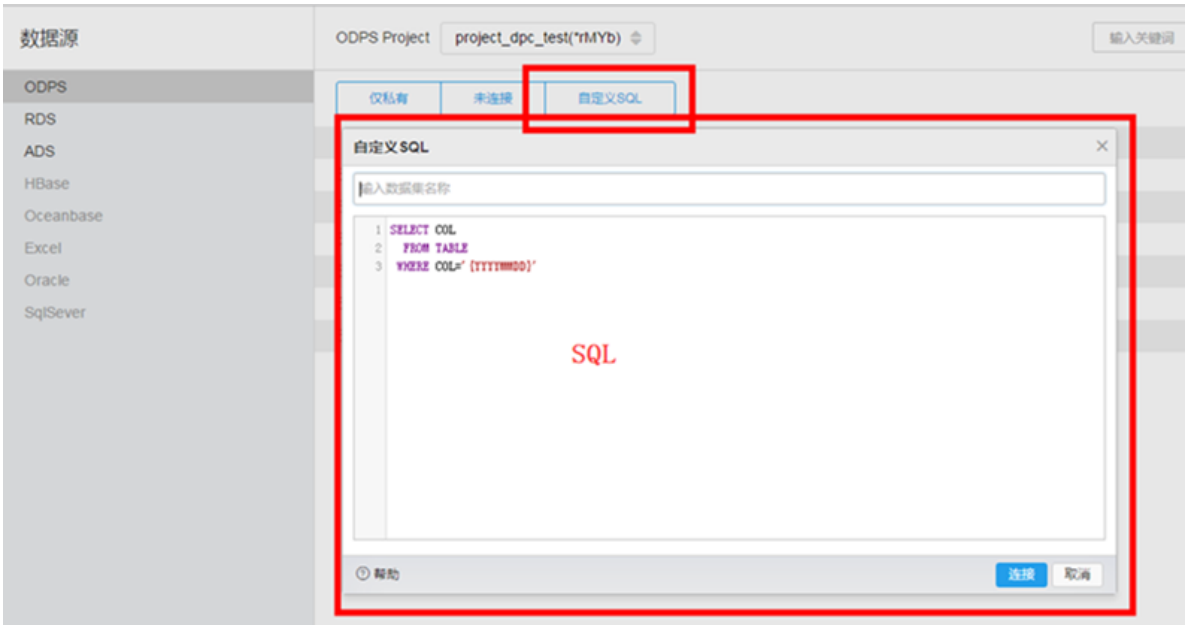
背景信息

除了上述的两种方式外，数据分析还支持您通过自定义 SQL 的方式来创建一个数据集。

操作步骤

1. 在对应的数据源窗口，单击**自定义 SQL**，如图 *1711: 自定义SQL*所示。

图 1711: 自定义SQL



2. 在**自定义 SQL**区域框下方，创建您需要的数据集。

33.4.4 创作一张工作表

连接好数据源之后，下面就可以做数据分析了。

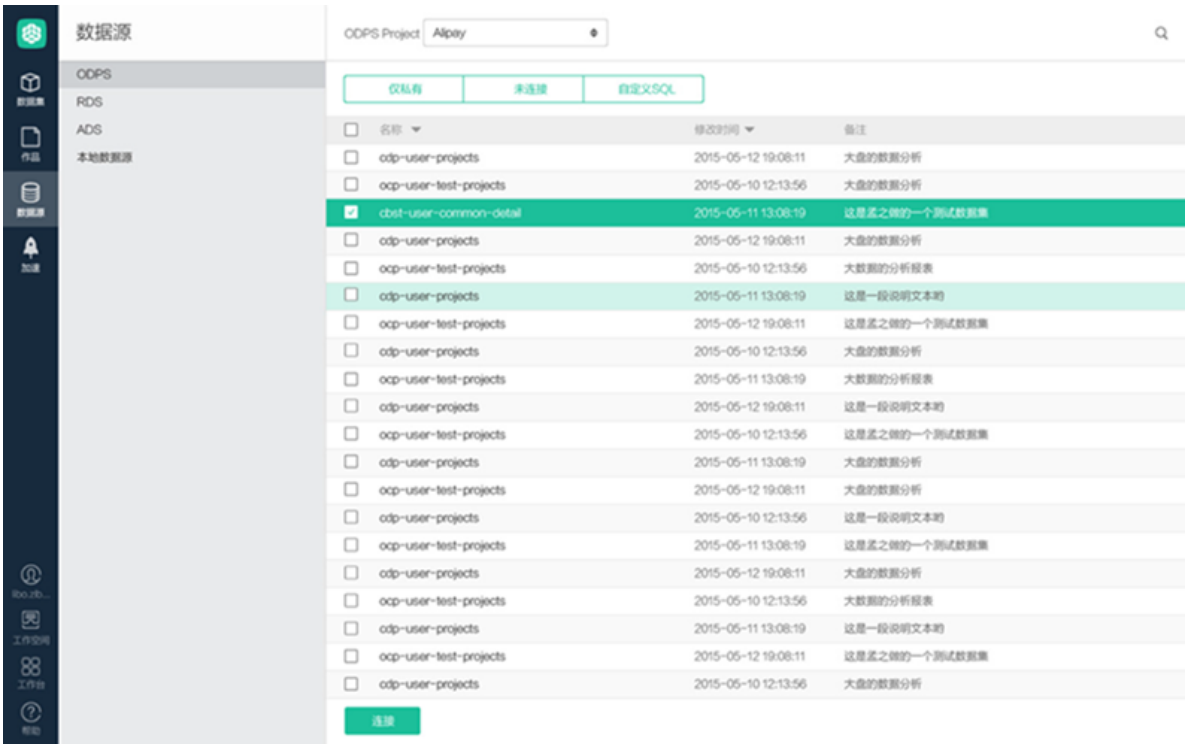
背景信息

本节通过一个小例子，简单介绍一下如何创作一张工作表。

操作步骤

1. 在左侧导航栏中，单击**数据源**，找到并选中需要做分析的数据表。
2. 单击页面下方的**连接**，如图 *1712: 连接数据源*所示。

图 1712: 连接数据源



系统会基于数据表创建对应的数据集。

3. 导入的数据集，默认显示在**数据集** > **我的数据集**目录下，您可以单击右侧的 **...**，对数据集做预览、重命名等操作，此处选择**立即使用**，如图 *1713:* 立即使用数据集所示。

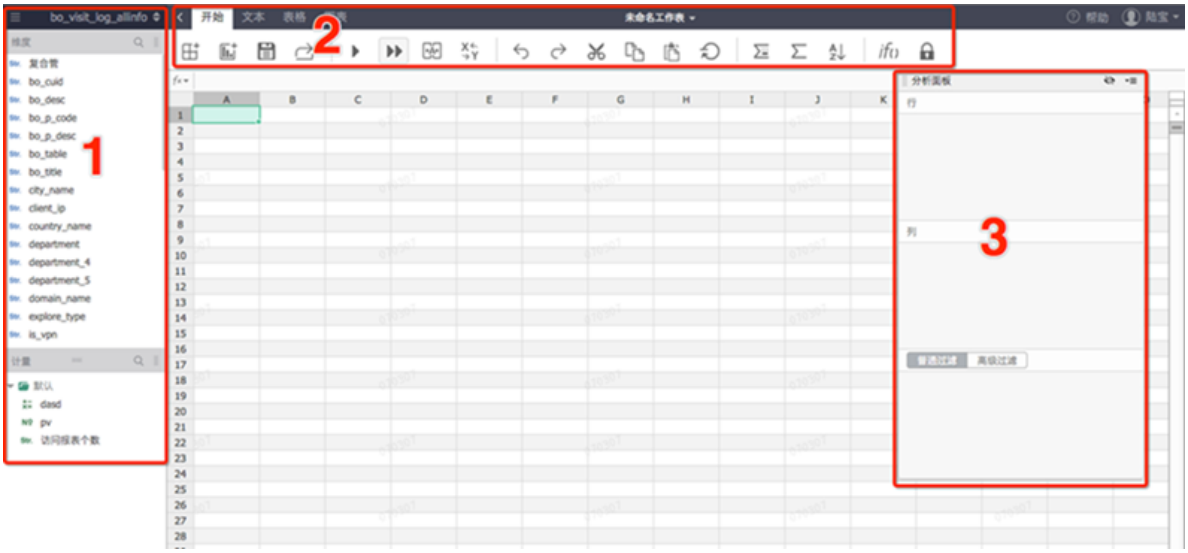
图 1713: 立即使用数据集



进入数据分析的工作界面。

如图 *1714:* 数据分析界面所示，区域 1 是该数据集的字段属性，默认按照类型的不同分为维度、计量；区域 2 是工具栏；区域 3 是分析面板。

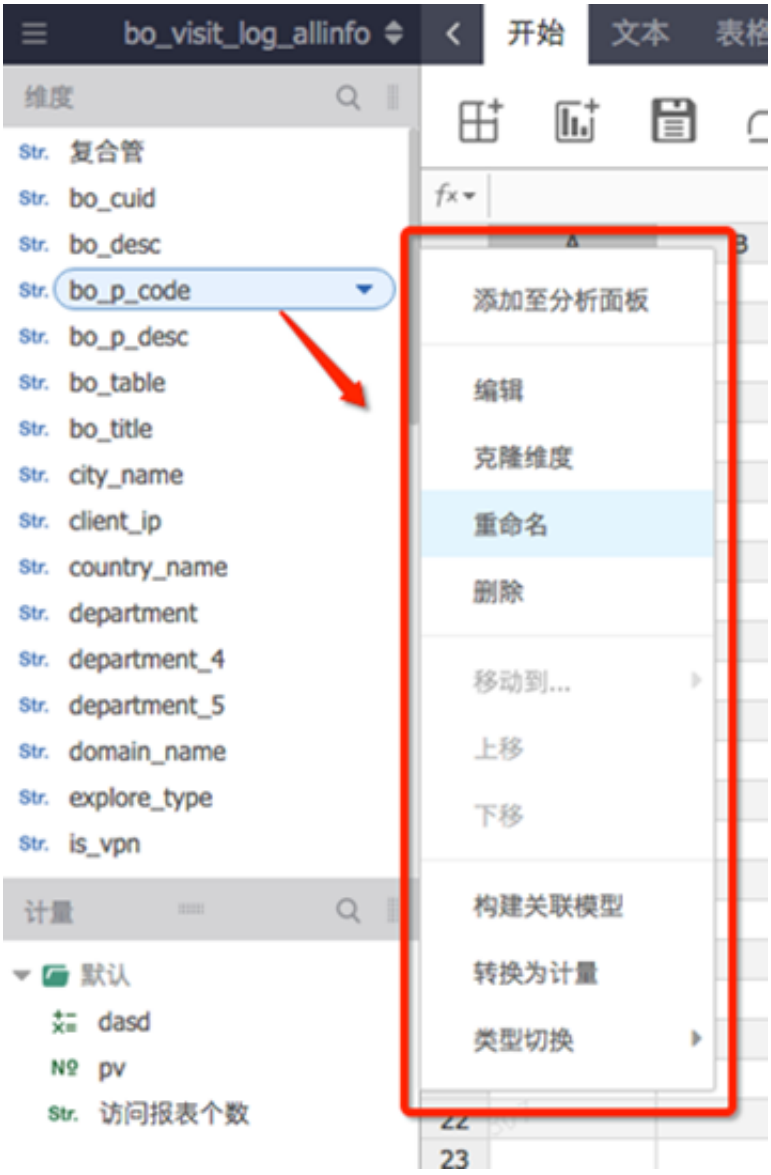
图 1714: 数据分析界面



说明: 在区域 1 中，对于数据集的字段属性，基本的规则是源表字段如果是数值型则默认为计量；如果是字符等其他类型则默认为维度。

4. 单击区域 1 中某个维度右边的箭头图标或者单击右键可以显示字段属性的编辑菜单，如图 *1715: 右键菜单*所示。

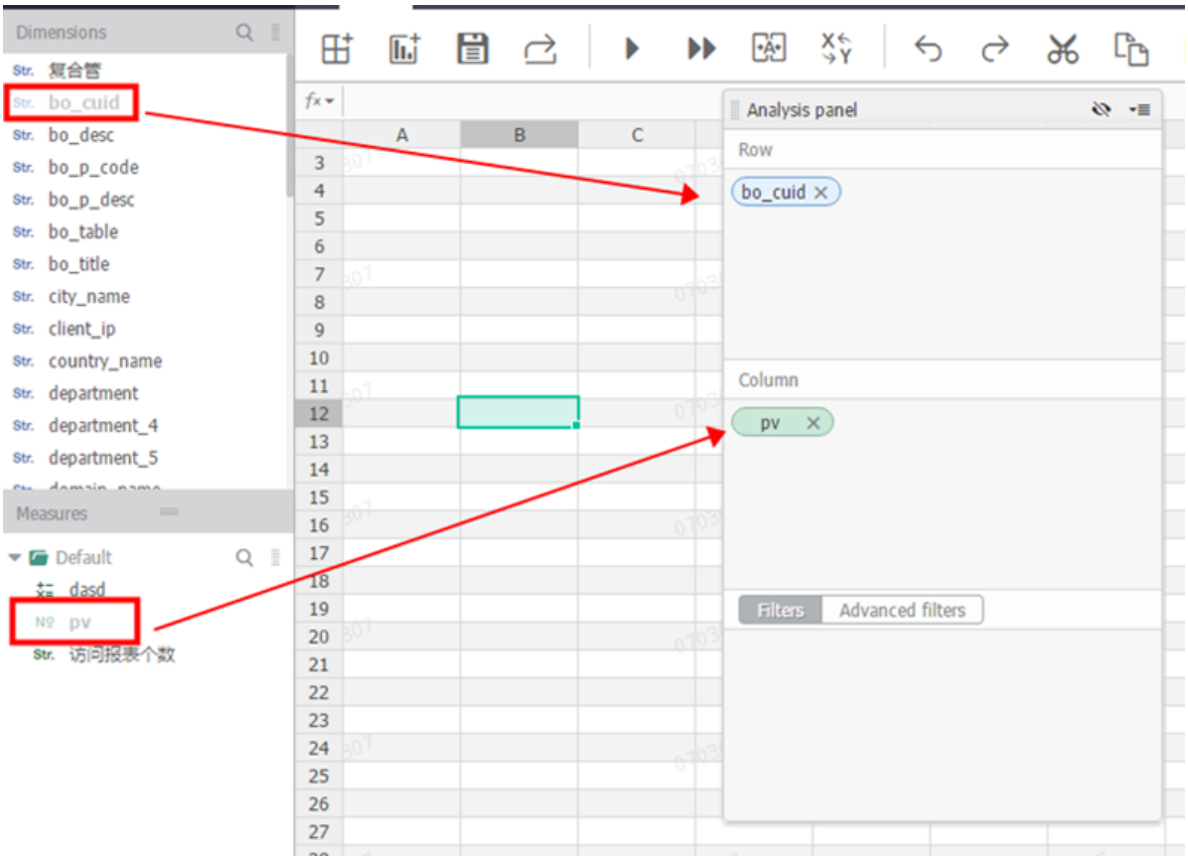
图 1715: 右键菜单



此处可以对字段作：添加至分析面板、编辑、克隆、重命名、删除、移动等操作，同时也支持维度、计量的类型转换。

5. 通过双击或者拖拽的方式添加字段至分析面板，如图 *1716: 添加字段至分析面板*所示。

图 1716: 添加字段至分析面板



已经添加的字段会被置灰，为不可编辑状态，如果您需要编辑字段，需要先把该字段从分析面板移除。

6. 通过拖拽的方式，将分析面板中的字段移动到您希望的位置，如图 *1717: 移动字段*所示。

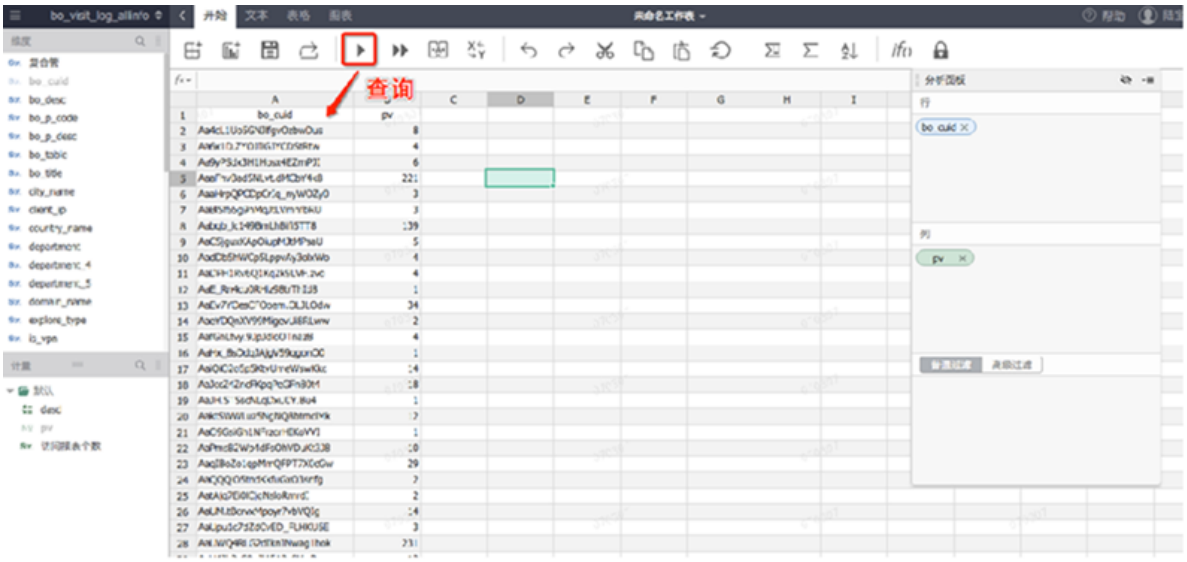
图 1717: 移动字段



7. 按照行、列、查询条件设置好您的数据分析后，单击工具栏上的查询图标。

   系统会按照您的设置去数据库查询并以表格的方式展现数据，如图 *1718:* 设置查询条件所示。

   图 1718: 设置查询条件



此处，如果您希望您的每一步操作，都会被自动执行查询操作，可以点选**自动执行**按钮，如图 *1719: 自动执行*所示。

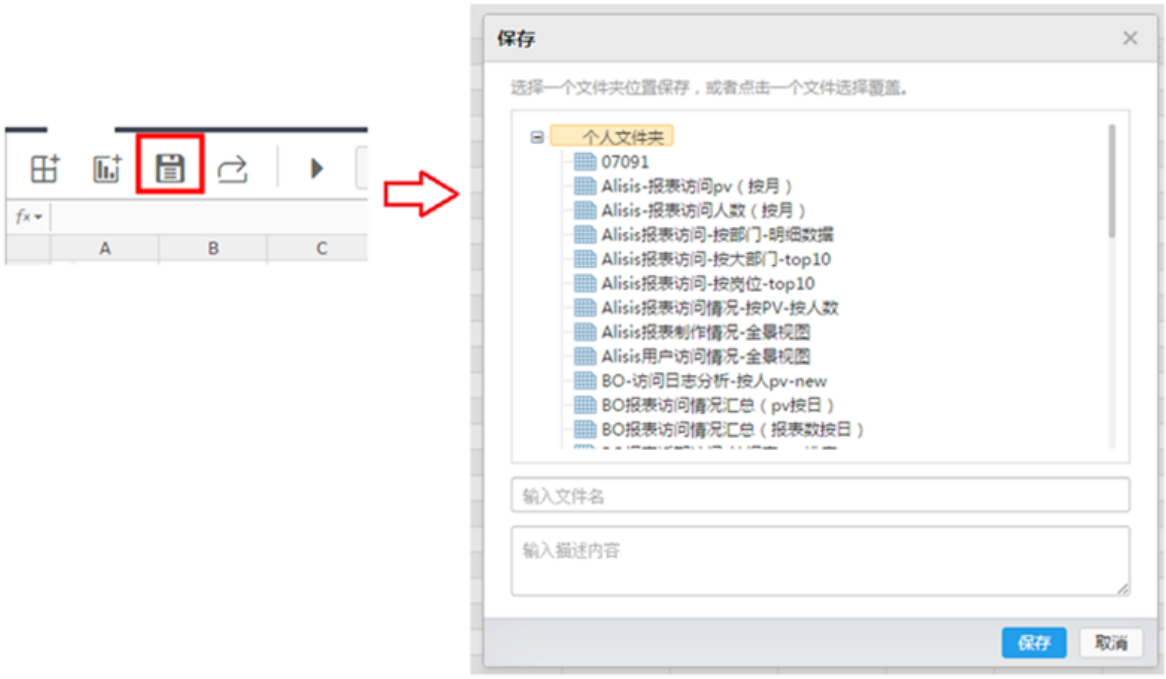
图 1719: 自动执行



说明: 如果连接的是 MaxCompute 的表，因为查询性能的原因，自动执行功能默认是关闭的；如果连接的Analytic DB、RDS 的表，自动执行功能默认打开。

8. 按照需要，您可以单击工具栏上的保存图标，并在弹出窗口中填写名称、描述等信息，把当前的分析保存为一张工作表，如图 *1720: 保存工作表*所示。

图 1720: 保存工作表



33.4.5 管理数据集

与数据源建立连接后，导入表结构信息，与物理表建立连接的一张逻辑表，称为数据集。

背景信息

数据集分为 3 类：我的个人数据集、共享给我的数据集、数据集市场，如图 *1721: 数据集*所示。

图 1721: 数据集



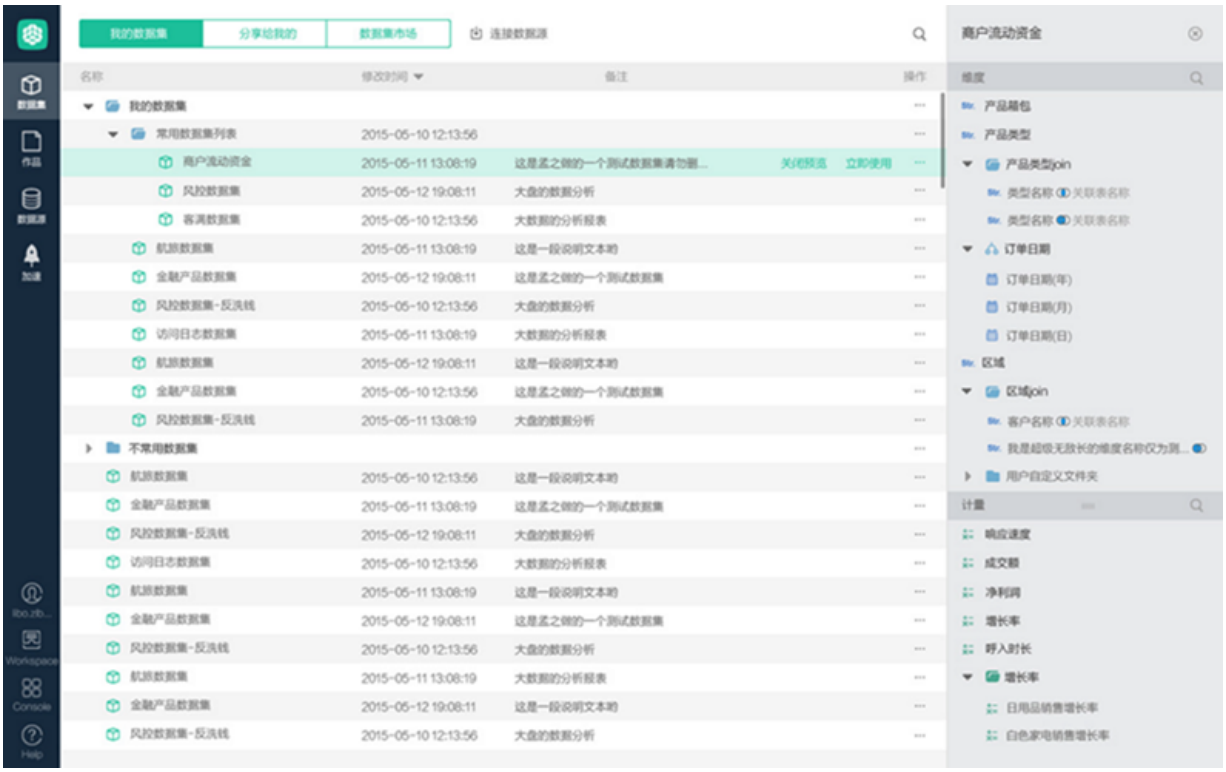
其中：

- 我的个人数据集：我创建的数据集，我可以对数据集定义进行编辑。
- 共享给我的数据集：别人分享给我，或者我申请的数据集。
- 数据集市场：别人发布出来的数据集，可以在此寻找我想要的数据集，然后申请使用。

33.4.5.1 预览数据集

预览功能主要用来查看数据集的字段信息，如图 *1722:* 预览所示。

图 1722: 预览



33.4.5.2 立即使用数据集

右键单击选择**立即使用**后，会进入数据分析的工作界面，可以对该数据集进行分析。

33.4.5.3 重命名/删除/移动数据集

- 重命名：对该数据集进行重命名操作；
- 删除：从个人数据集中删除所选择的数据集；
- 移动到...：移动数据集到个人数据集的其他目录下；

33.4.5.4 查看数据集属性

查看属性提供了查看数据集更详细信息的功能，主要包括数据源、物理表名称、创建时间、最后修改时间、所有者、查询次数等信息。

右键单击选择**查看属性**后，显示如图 *1723: 查看属性*所示界面。

图 1723: 查看属性



33.4.5.5 更新数据集结构

如果数据源表结构发生变化，可以通过这个功能，获取最新数据集结构。

33.4.5.6 构建关联模型

可以将多表进行 join 设置。

右键单击选择**构建关联模型**后，显示如图 *1724: 创建关联模型*所示界面。

图 1724: 创建关联模型



其中：

- 图中1，表示当前选择的数据集用来做关联的字段；
- 图中2，表示与当前数据集做关联的维表；
- 图中3，表示用来做关联的维表字段；
- 图中4，可以通过**新建关联**来新增该数据集的关联关系；
- 图中5，单击**删除**可以删除当前行的关联关系。

33.4.6 使用分析面板

将数据集的维度和计量字段拖放到分析面板的行、列或者过滤条件中，进行多维分析。

分析面板如图 *1725: 分析面板*所示。

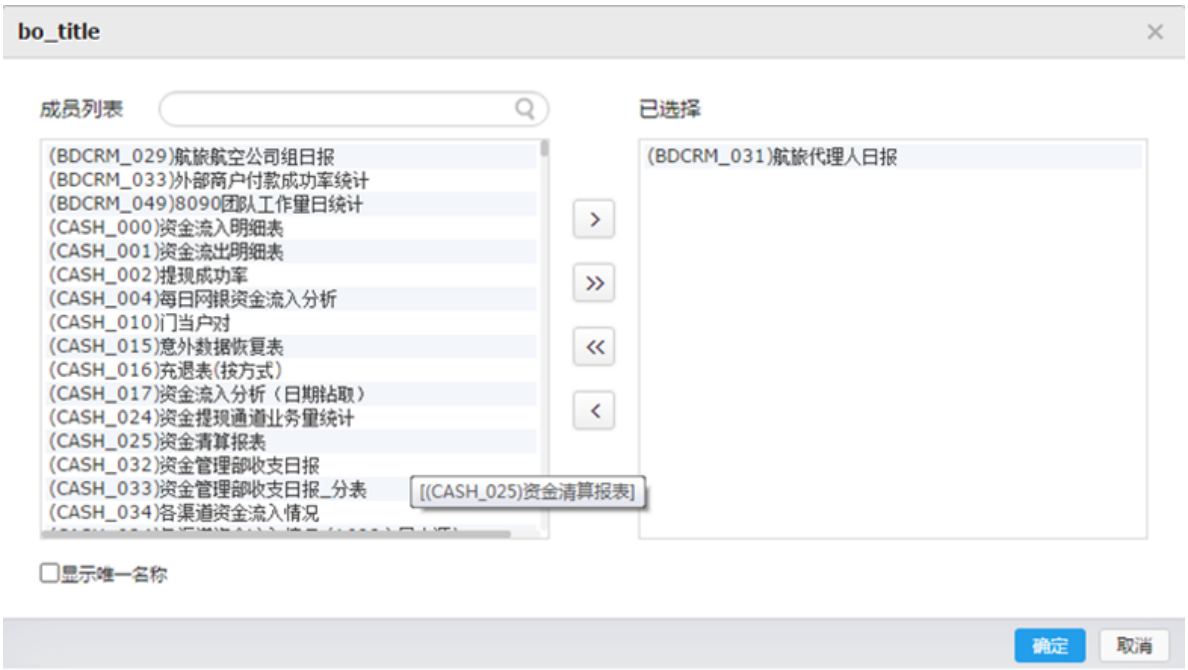
图 1725: 分析面板



在分析面板上，您可以：

- 单击**维度**图标，在弹出的窗口中设置维值作为筛选条件，如图 *1726:* 设置维值所示；

  图 1726: 设置维值



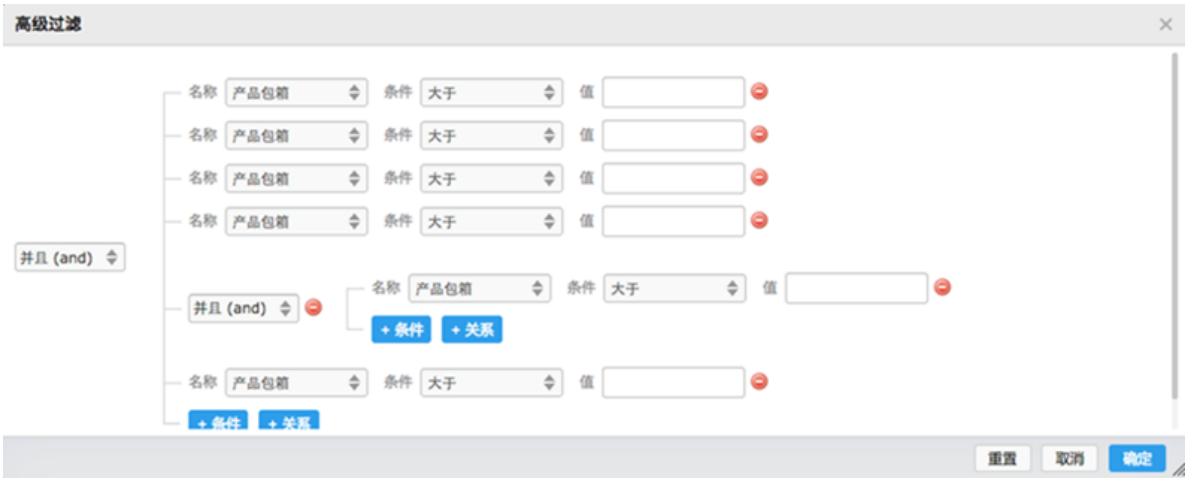
- 单击分析面板右上角的**维度查看**图标，查看当前维度的设置情况，如图 *1727:* 查看维度设置情况所示；

图 1727: 查看维度设置情况



- 单击**高级过滤**标签页，设置更复杂的查询条件，如图 *1728:* 高级过滤所示。

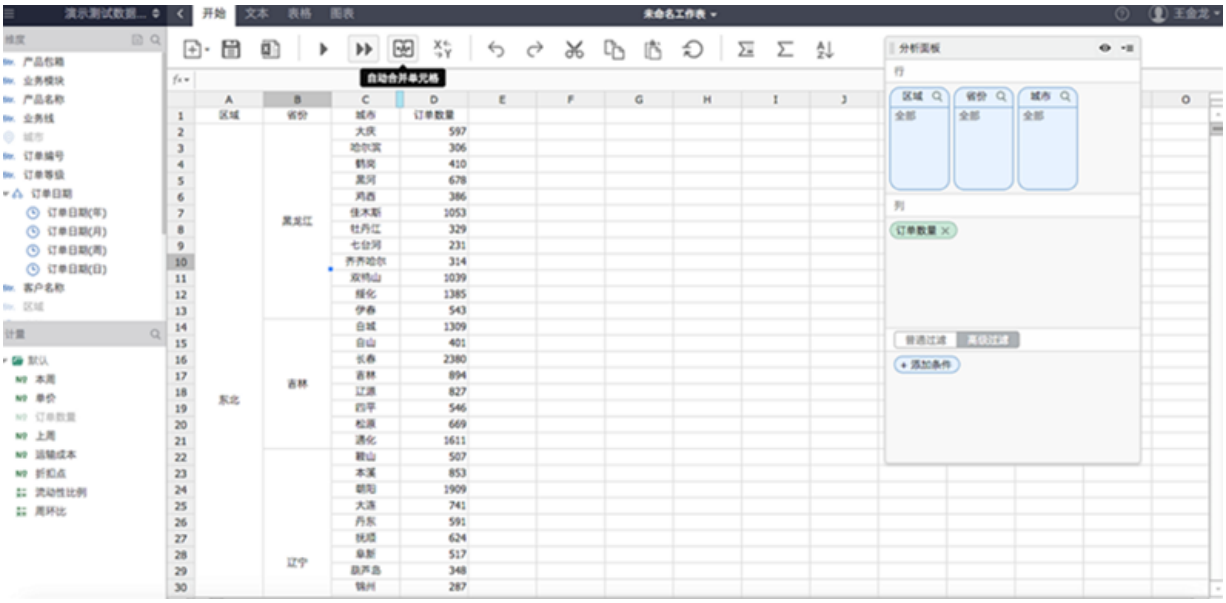
图 1728: 高级过滤



33.4.7 合并单元格

单击工具栏上的**合并单元格**图标，对分析结果可以格式化，让结果更美观，如图 *1729: 合并单元格*所示。

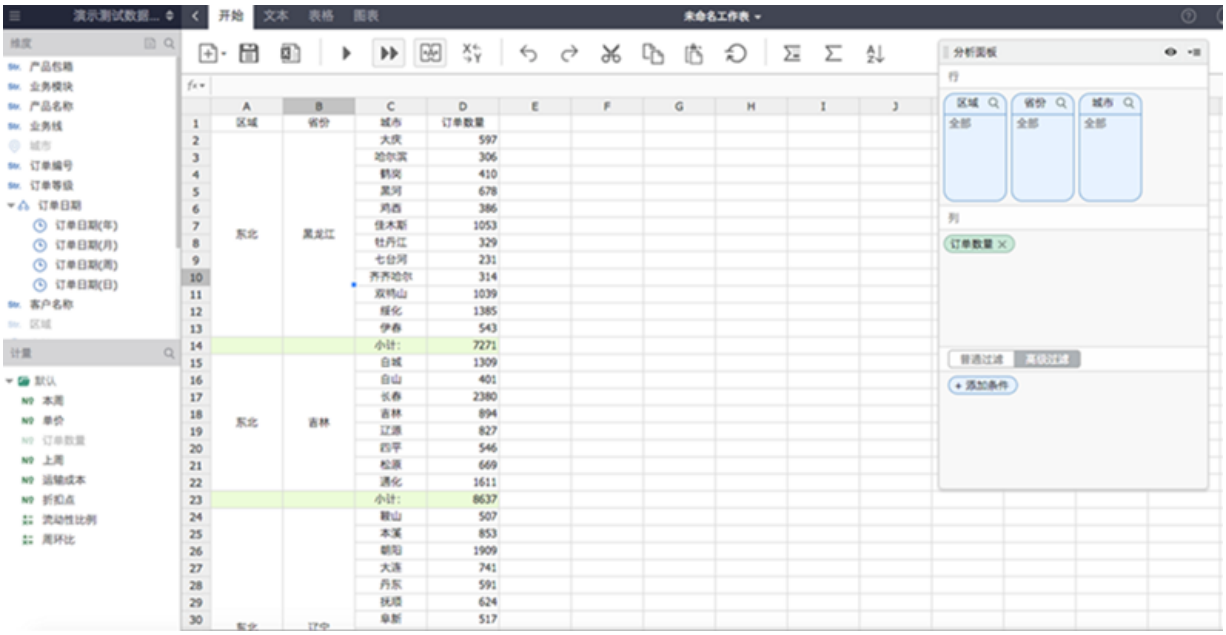
图 1729: 合并单元格



33.4.8 分类汇总和自动求和

单击工具栏上的**分类汇总**图标，对分析结果可以分类汇总，实现小计功能，如图 *1730: 分类汇总*所示。

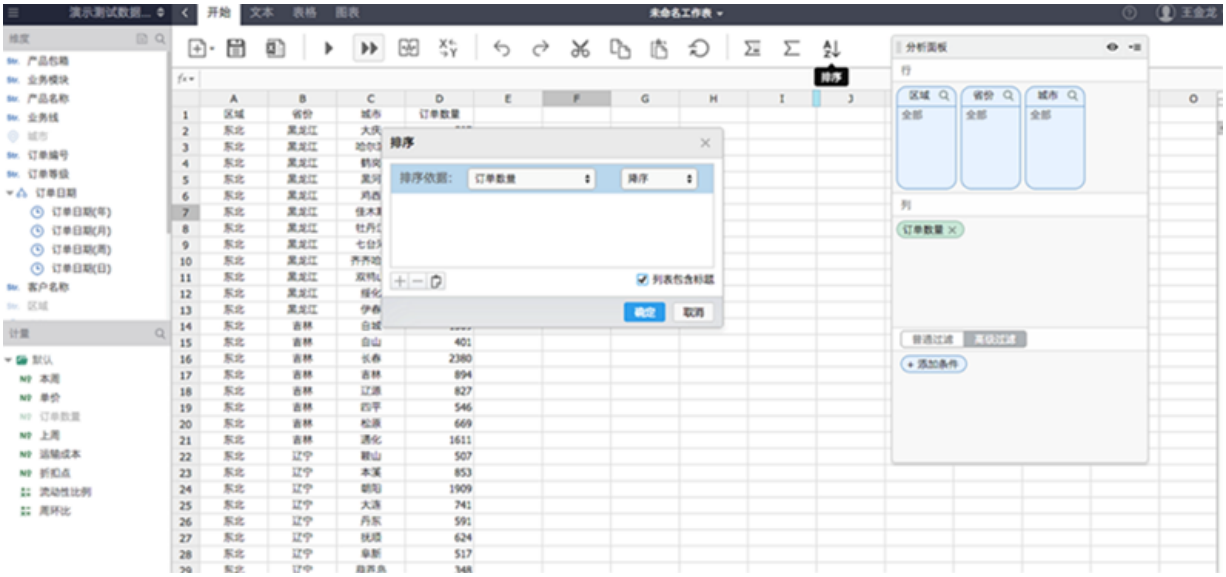
图 1730: 分类汇总



33.4.9 排序

单击工具栏上的**排序**图标，在弹出窗口中设置排序规则，实现对数据的后台排序，如图 *1731: 排序*所示。

图 1731: 排序



33.4.10 交换 XY 轴

单击工具栏上的**交换 XY 轴**图标，可以让数据横向展示，如图 *1732: 交换 XY 轴*所示。

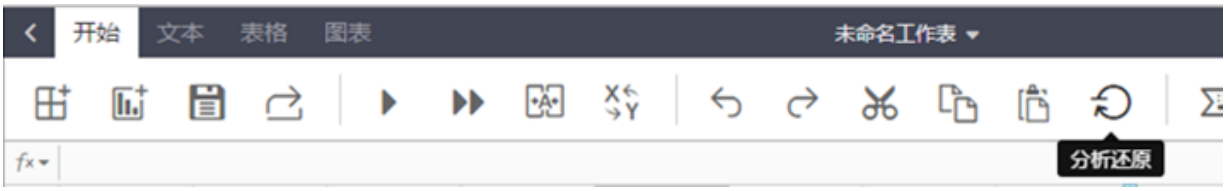
图 1732: 交换 XY 轴



33.4.11 分析还原

单击工具栏上的**分析还原**图标，可以将分析结果还原到初始状态，如图 *1733:* 分析还原所示。

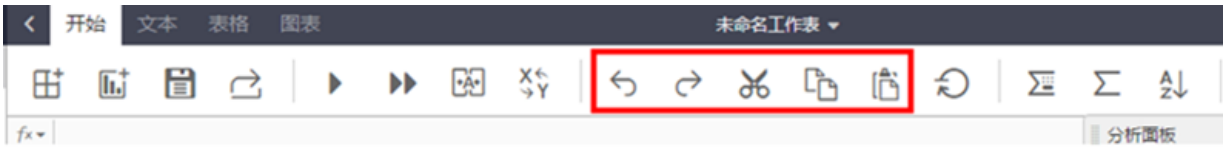
图 1733: 分析还原



33.4.12 撤销/重做/剪切/复制/粘贴

可以对表格中编辑的内容做**撤销**、**重做**、**剪切**、**复制**和**粘贴**操作，如图 *1734:* 撤销等操作所示。

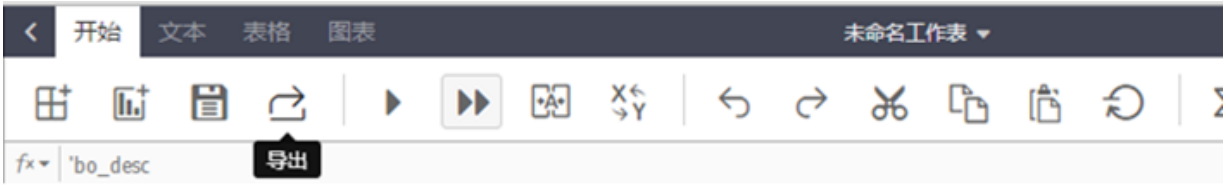
图 1734: 撤销等操作



33.4.13 导出

单击工具栏上的**导出**图标，可以将分析结果导出为 EXCEL 文件，根据管理员的配置，此处文件有可能会上传至预设的文件平台，如图 *1735:* 导出分析结果所示。

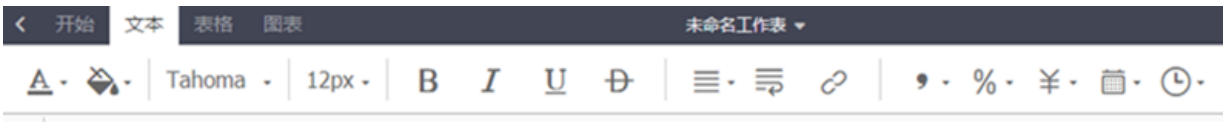
图 1735: 导出分析结果



33.4.14 文本格式设置

在工具栏的**文本**标签页中，选择相应的工具进行文本格式设置，如图 *1736:* 文本格式设置所示。

图 1736: 文本格式设置



支持如下操作：

- 设置文字的字体、颜色、背景；
- 设置文字的大小、样式（粗体、斜体、下划线、删除线等等）；
- 设置单元格的样式（文字对齐方式、换行、超链接等）；
- 设置文字的格式化方式（小数点、百分号、日期等）。

33.4.15 表格属性设置

在工具栏的**表格**标签页中，选择相应的工具进行表格属性设置，如图 *1737:* 表格属性设置所示。

图 1737: 表格属性设置



支持如下操作：

- 插入行、插入列、删除行、删除列；
- 合并单元格、拆分单元格、隐藏行、隐藏列；
- 框线设置，如图 *1738:* 框线设置所示。

图 1738: 框线设置



33.4.16 斜线表头

在工具栏的**表格**标签页中，单击**斜线**图标，可以设置斜线表头，如图 *1739: 设置斜线表头*所示。

图 1739: 设置斜线表头



33.4.17 锁定首行首列

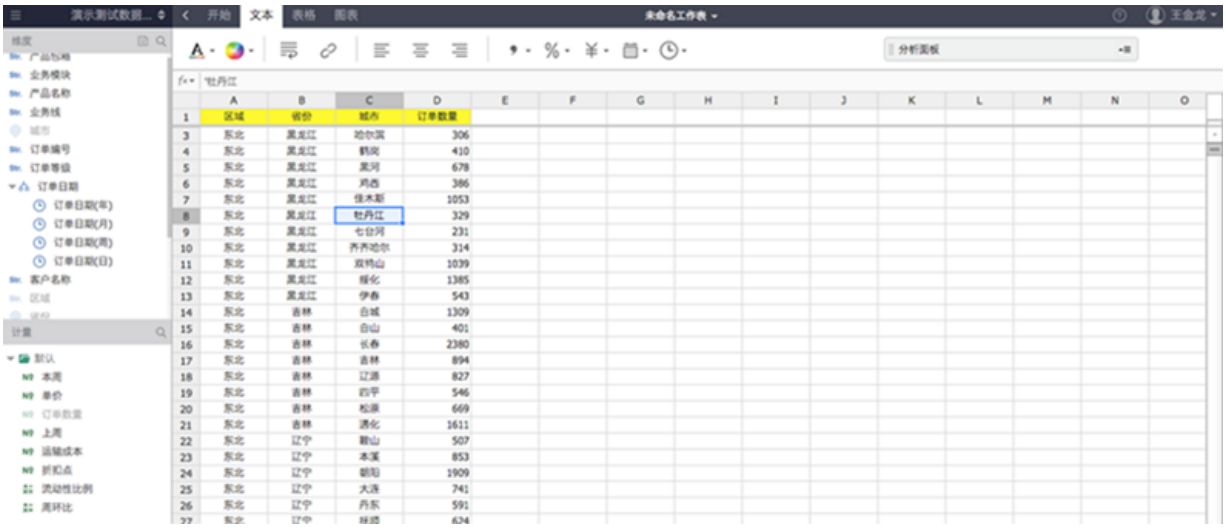
在工具栏的**表格**标签页中，单击**冻结**图标，可以设置表格行列固定，如图 *1740: 冻结*所示。

图 1740: 冻结



最终实现的效果如图 *1741: 冻结后的结果*所示。

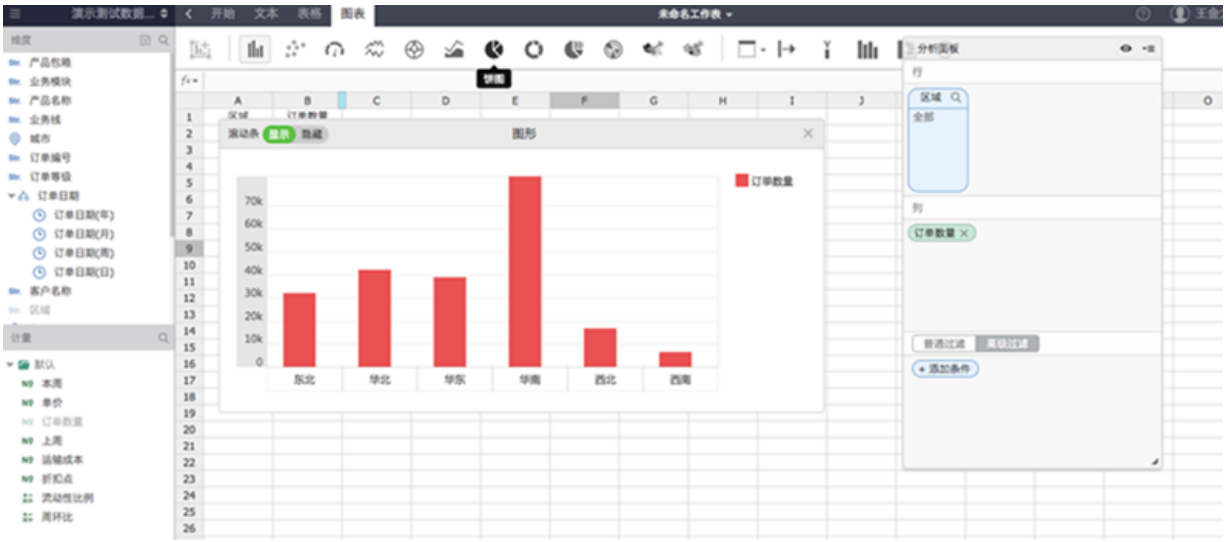
图 1741: 冻结后的结果



33.4.18 图表

在工具栏的**图表**标签页中，选择相应的图标，可以设置分析结果的图表展现方式，如图 *1742: 图表显示*所示。

图 1742: 图表显示

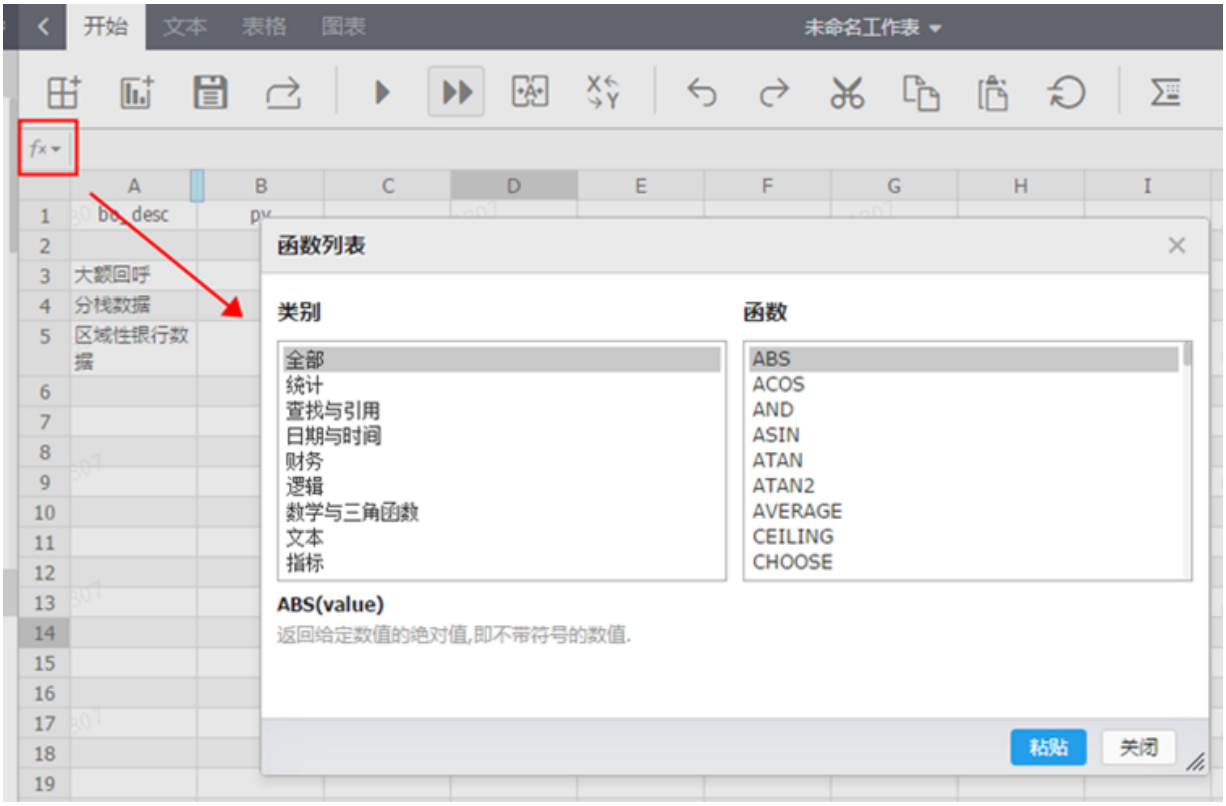


目前支持柱状图、气泡图、饼图、折线图、环状图、地图等 12 种常用图表。

33.4.19 函数

通过单击编辑栏中的**函数**图标，可以调用函数窗口，目前支持常用的约 100 种函数，可以按需选用对分析结果进行处理，如图 *1743:* 调用函数所示。

图 1743: 调用函数



33.4.20 条件格式

在工具栏的**开始**页签中，单击**条件格式**图标，可以设置条件格式。

工作表条件格式的使用方法，如图 *1744:* 设置数值、图 *1745:* 设置图表和图 *1746:* 设置空值所示。

图 1744: 设置数值

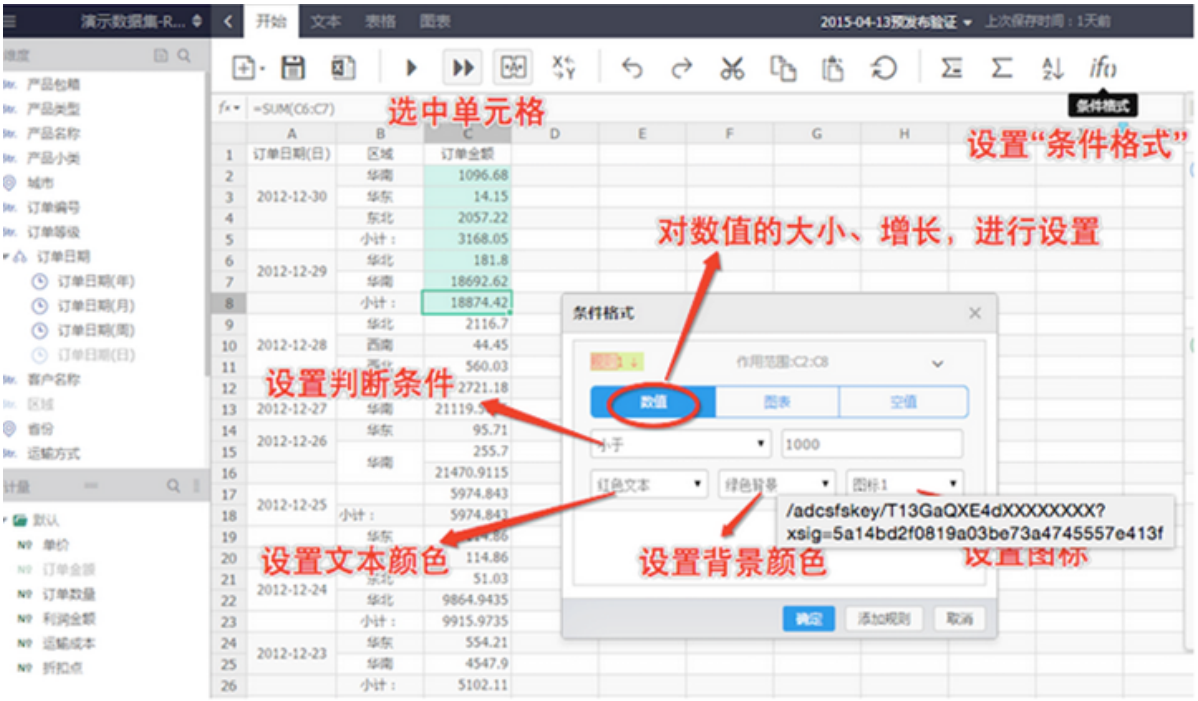


图 1745: 设置图表

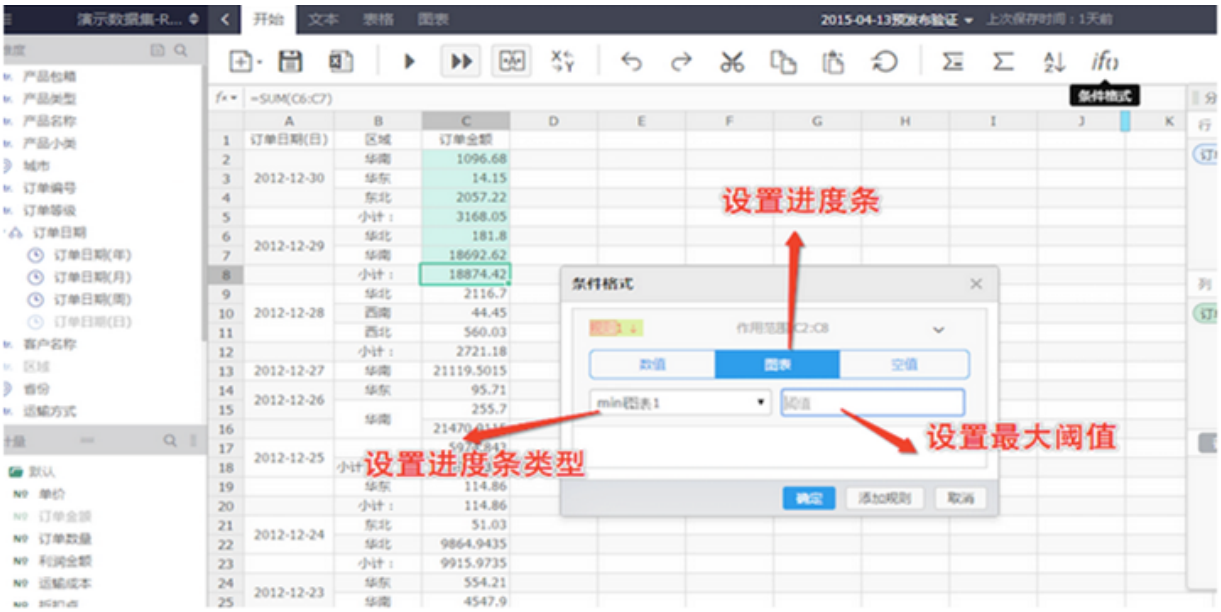
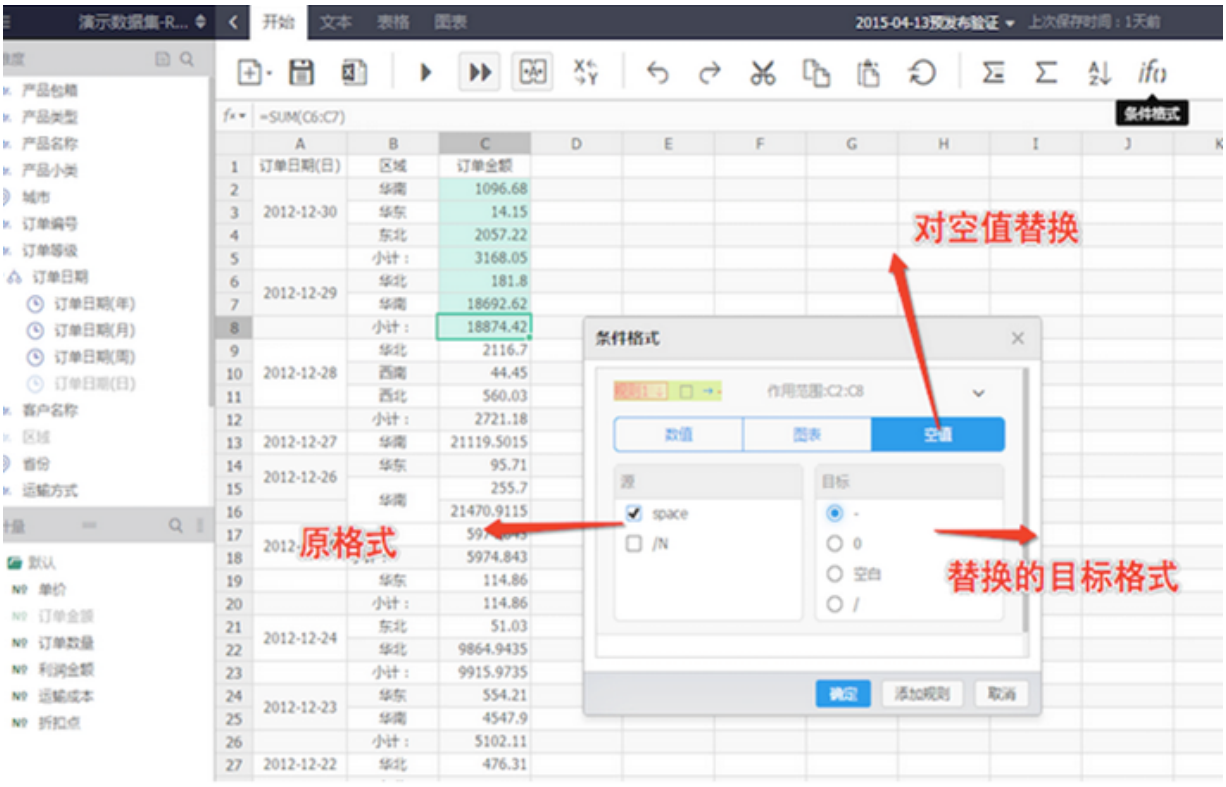


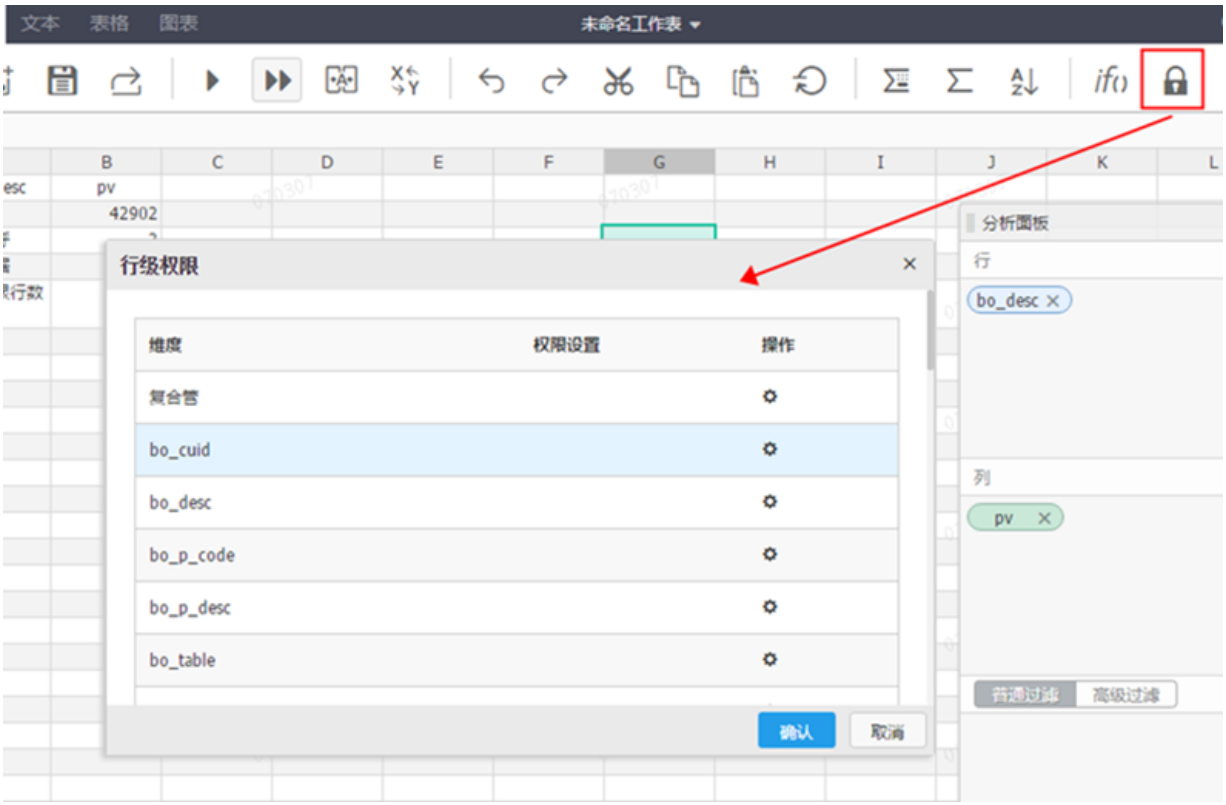
图 1746: 设置空值



33.4.21 行级权限

通过单击编辑栏中的**行级权限**图标，可以调用行级权限配置窗口，如图 *1747: 行级权限*所示。

图 1747: 行级权限



其中：

- 维度是指当前数据集的维度字段；
- 权限设置是指用来过滤权限的规则；
- 通过操作图标，可以设置维度与预设的权限规则的对应关系。

33.5 报表制作

33.5.1 报表制作服务简介

报表制作服务RBS （Report Building Service），提供所见即所得的设计数据产品功能。简单易用，使人人都 有使用数据的能力。大大缩短从产生决策欲望到满足欲望的时间周期。

报表制作服务主要由如下几部分组成：

- 仪表盘制作与管理
- 数据产品制作与管理
- 分享发布与权限管理

仪表盘制作与管理

制作仪表盘样式的交互式报表，包含如下功能：

- 支持以行列布局形式拖放制作仪表盘报表；
- 支持布局组件，布局用于存放**控件**。

  包括：行列布局、TAB 页、iFrame 引用。
- 支持控件组件，控件用于支持报表交互操作；

  包括：文本、过滤器、时间、查询条件、按钮、评论
- 支持图表组件，以图表的形式展示数据；

  包括：表格、柱形图、折线图、饼图、油量表、雷达图、极地坐标图、地图、漏斗图、指标看板
- 支持组件间的参数传递与联动，并根据依赖顺序进行数据显示刷新；
- 支持仪表盘在编辑界面、预览界面和阅读界面的不同展示和处理；
- 支持通过 URL 参数的形式传递参数至仪表盘，仪表盘根据输入参数进行数据展示；
- 图表组件支持来自工作表的数据，并根据数据特征进行图表样式的智能推荐；
- 图表组件支持来自指标集的数据，并支持对指标集的同步创建；
- 图表组件支持来自数据集的数据，支持按照数据集的维度与计量进行 OLAP 查询；
- 支持画布内容的 PDF 导出与图表组件的后台数据导出。

数据产品制作与管理

数据产品是预定义的仪表盘集合，提供类似菜单形式的用户交互式选择方式，包含如下功能：

- 支持按照模板进行数据产品的编辑制作；
- 支持数据产品的列表、预览和删除操作。

分享发布与权限管理

系统支持以分享与发布的形式，将制作好的报表分享给报表阅读者，包含如下功能：

- 支持仪表盘、数据产品的转让、分享和发布；
- 支持仪表盘、数据产品的页面访问权限控制；
- 支持仪表盘、数据产品的访问权限穿透传递；
- 支持在以下场景中对数据行级权限的设置，包括：仪表盘中来自数据集的数据；
- 支持在仪表盘阅读者界面中，根据阅读者权限，进行数据行级权限控制。

33.5.2 制作报表

以下介绍一个简单报表的制作过程。

背景信息

- 已经完成《数据分析》章节的学习。
- 已经完成一张测试工作表的制作，相关内容请参见创作一张工作表。

操作步骤

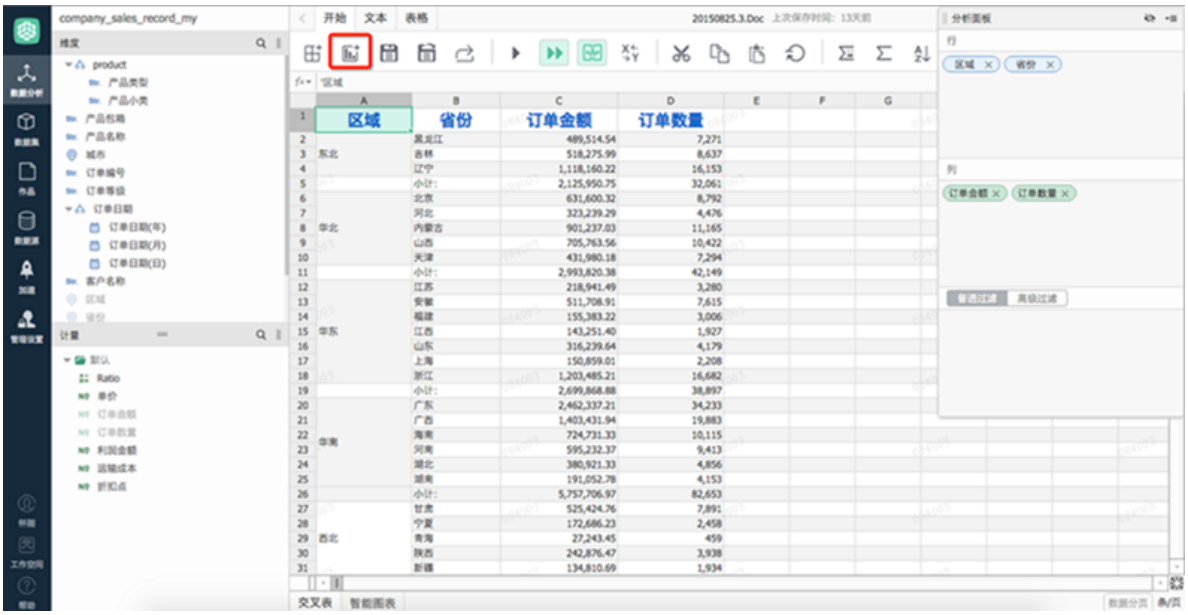
1. 单击大数据左侧导航栏中的**工作台**。
2. 在屏幕中间**我的应用**中，单击**报表制作**，如图 *1748:* 我的应用所示。

图 1748: 我的应用



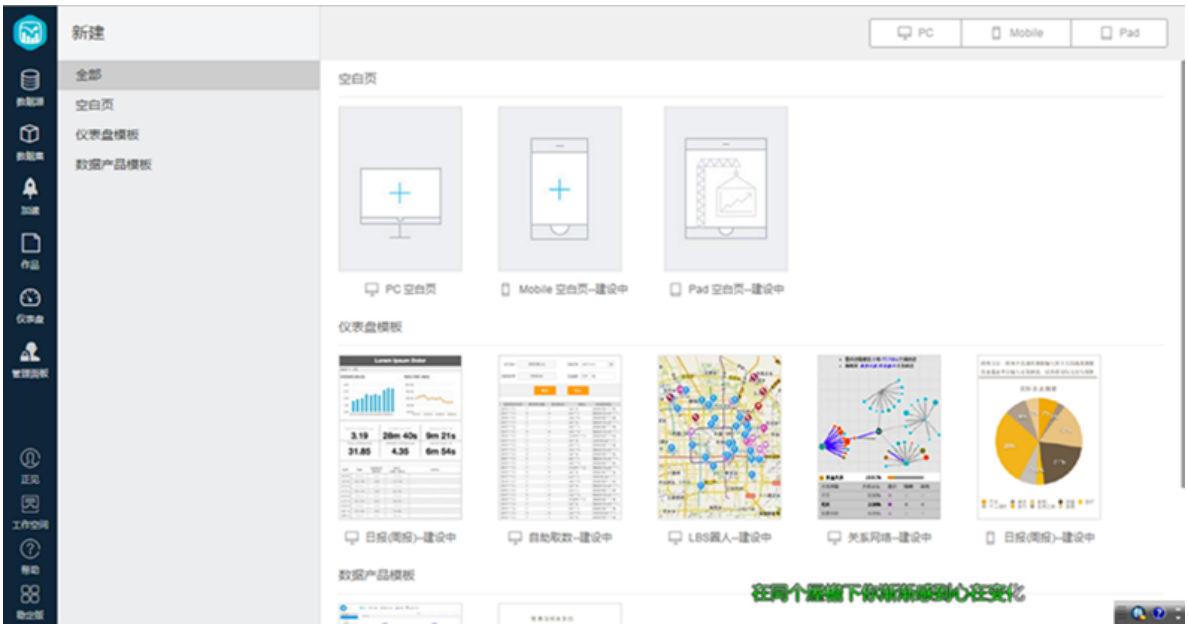
3. 单击如图 *1749:* 图标按钮所示中的按钮，进入报表制作界面。

图 1749: 图标按钮



4. 选择空白仪表盘模板，如图 *1750: 选择仪表盘模板*所示。

图 1750: 选择仪表盘模板



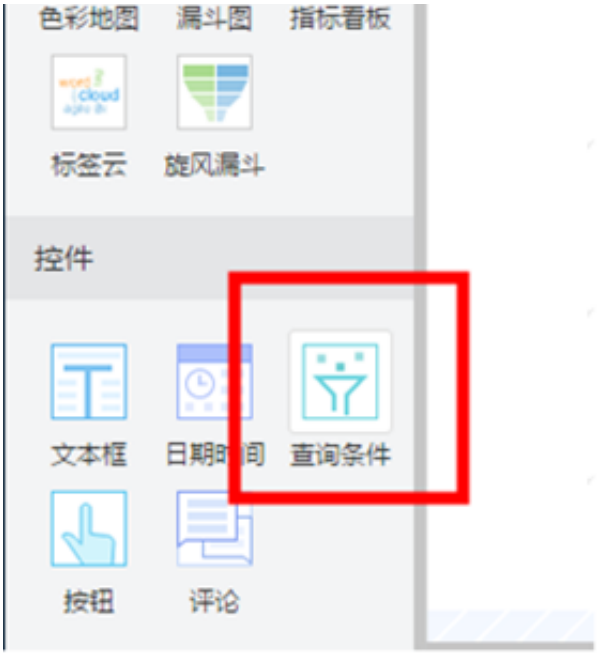
进入仪表盘编辑页面，如图 *1751: 编辑仪表盘*所示。

图 1751: 编辑仪表盘



5. 选择**查询条件**控件，单击**查询条件**，如图 *1752: 查询条件控件*所示。

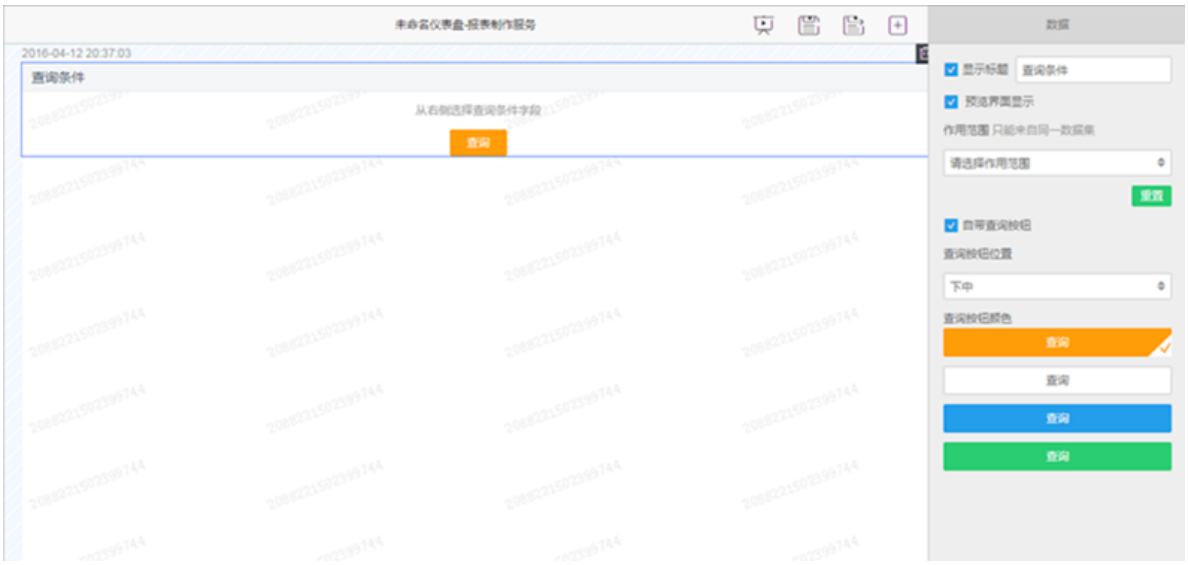
图 1752: 查询条件控件



条件查询控件被添加到报表中。

6. 从报表右侧的面板中选择需要添加的查询条件以添加到控件中，如图 *1753: 添加查询条件*所示。

图 1753: 添加查询条件



7. 选择**图表**控件，如图 *1754:* 图表控件所示。

图 1754: 图表控件



8. 编辑**图表**控件，选择图表关联到测试工作表。

a) 选择条目，如图 *1755:* 选择条目所示。

图 1755: 选择条目



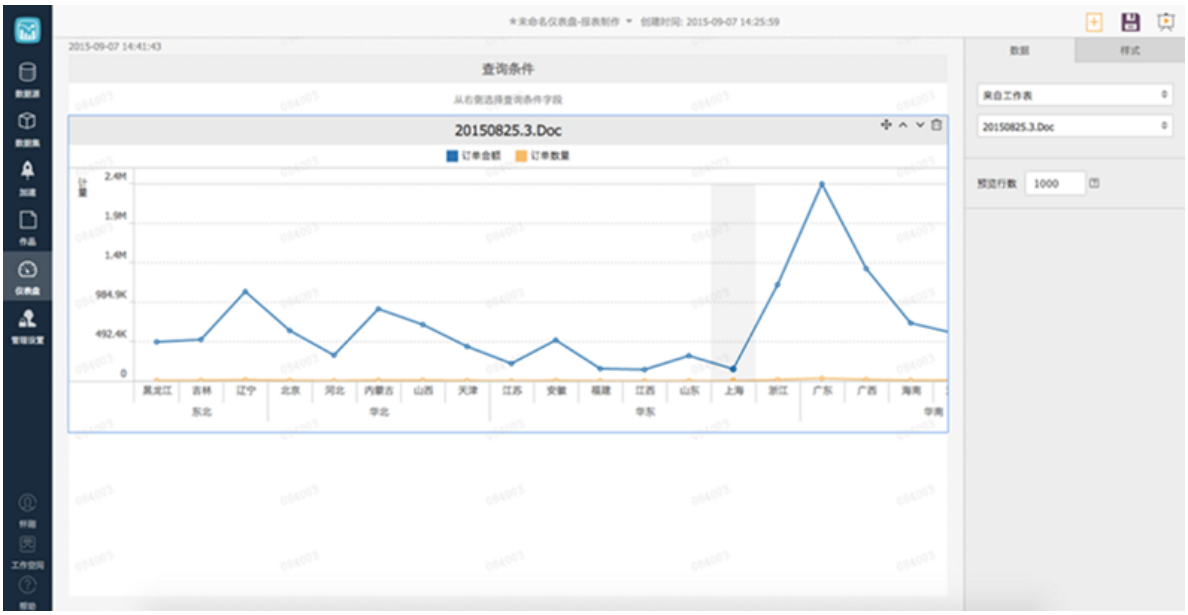
b) 选择工作表，如图 *1756:* 选择工作表所示。

图 1756: 选择工作表



完成图表与工作表的关联，工作表中的数据由绑定的图表展示，如图 *1757:* 图表与工作表关联后所示。

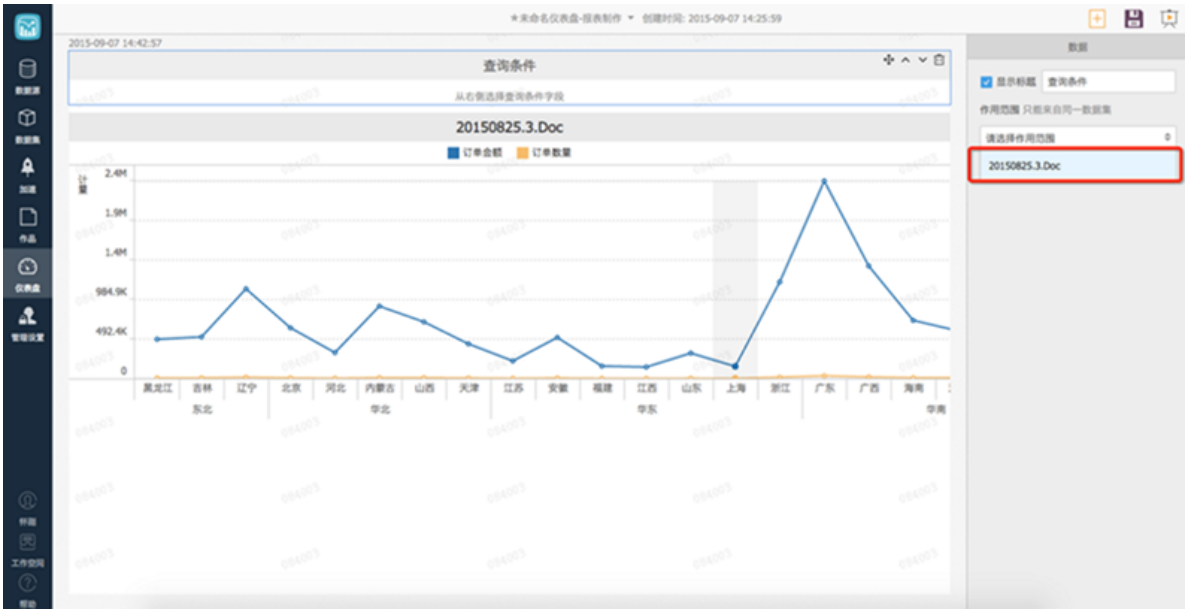
图 1757: 图表与工作表关联后



9. 编辑**查询条件**控件，选择关联到上一步选择的报表。

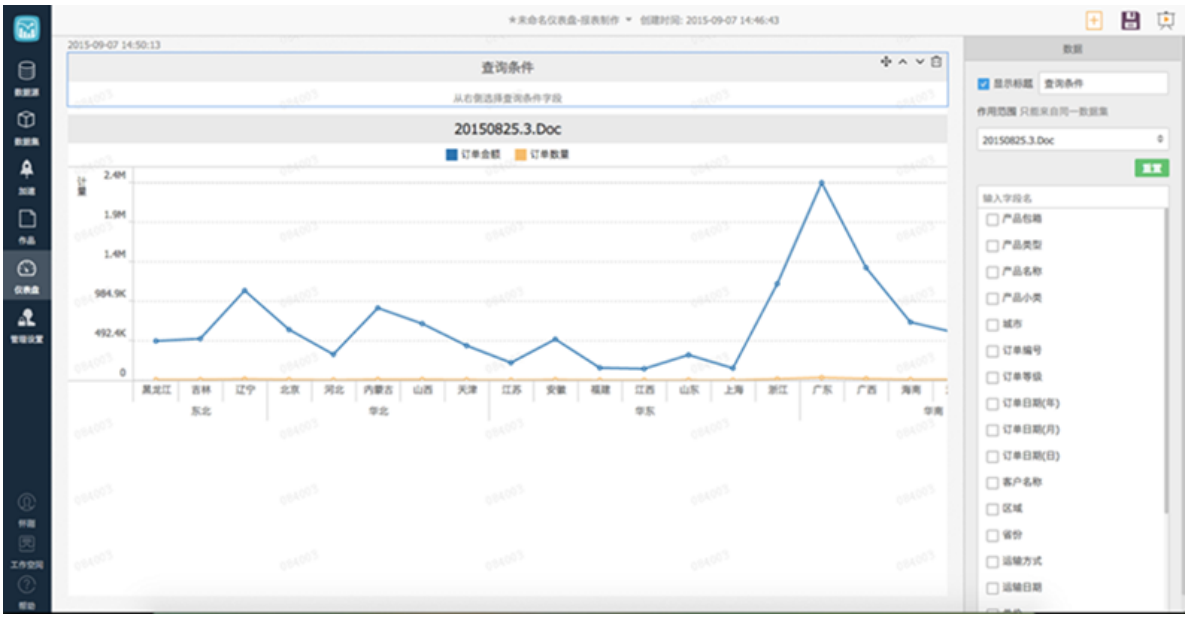
   选中已添加的**查询条件**控件，在右侧面板中将其与报表关联，如图 *1758:* 查询条件与报表关联所示。

图 1758: 查询条件与报表关联



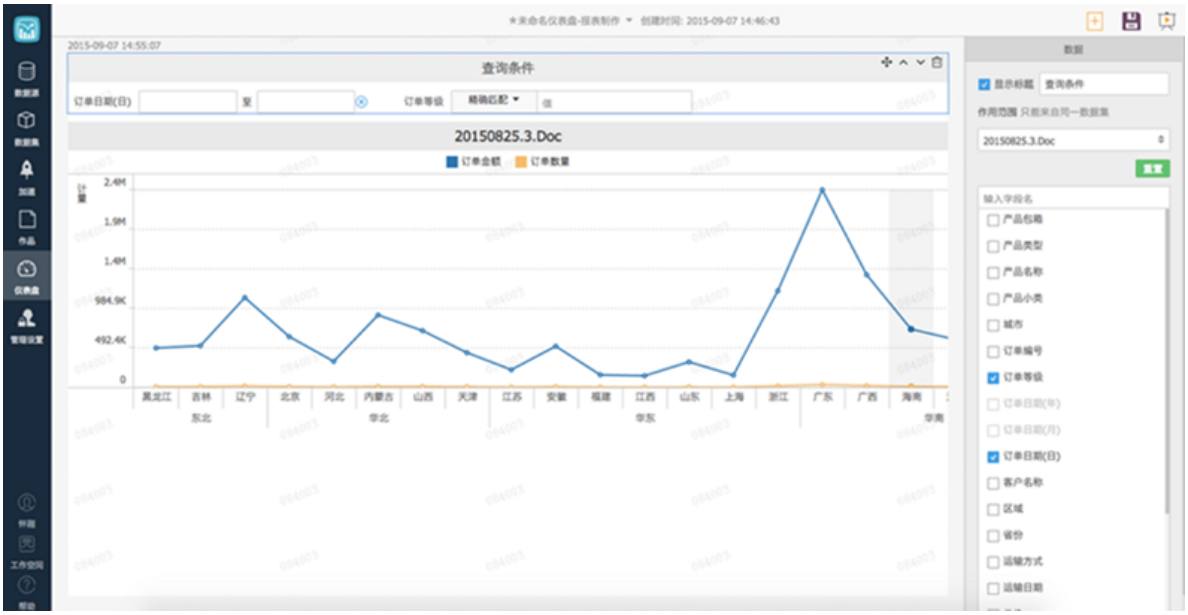
10. 关联后，右侧面板中将展示可用的筛选条件，勾选需要的条件，将其添加到**查询条件**控件中，如图 *1759:* 显示可用的条件所示。

图 1759: 显示可用的条件



选择**订单日期**和**订单等级**作为查询条件，如图 *1760:* 选择查询条件所示。

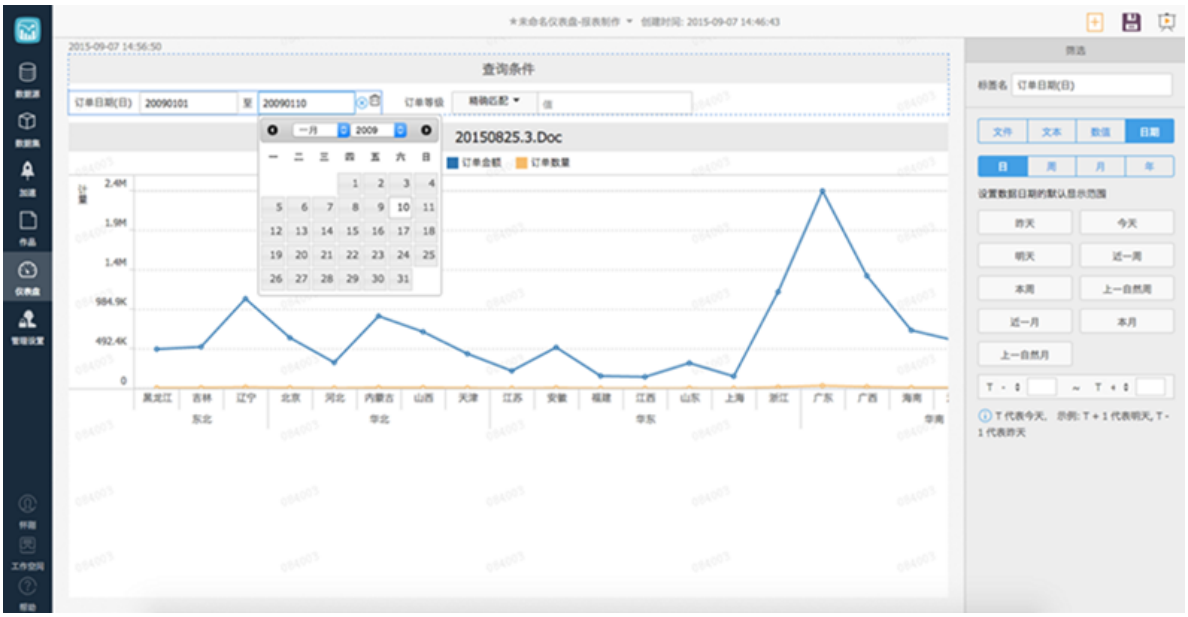
图 1760: 选择查询条件



11. 编辑订单日期的取值。

在**查询条件**控件中，按照所需条件，编辑订单日期的取值，修改取值后，图表中所展示的数据会相应变动，如图 *1761:* 修改订单日期所示。

图 1761: 修改订单日期



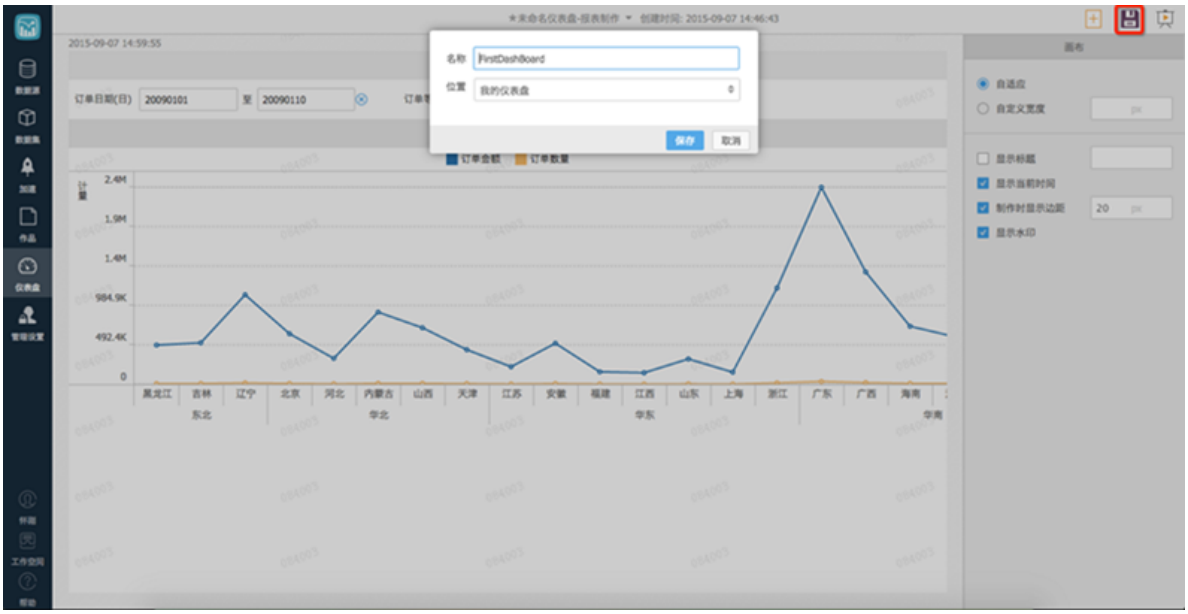
12.编辑订单等级的查询方式，如图 *1762: 修改订单等级*所示。

图 1762: 修改订单等级



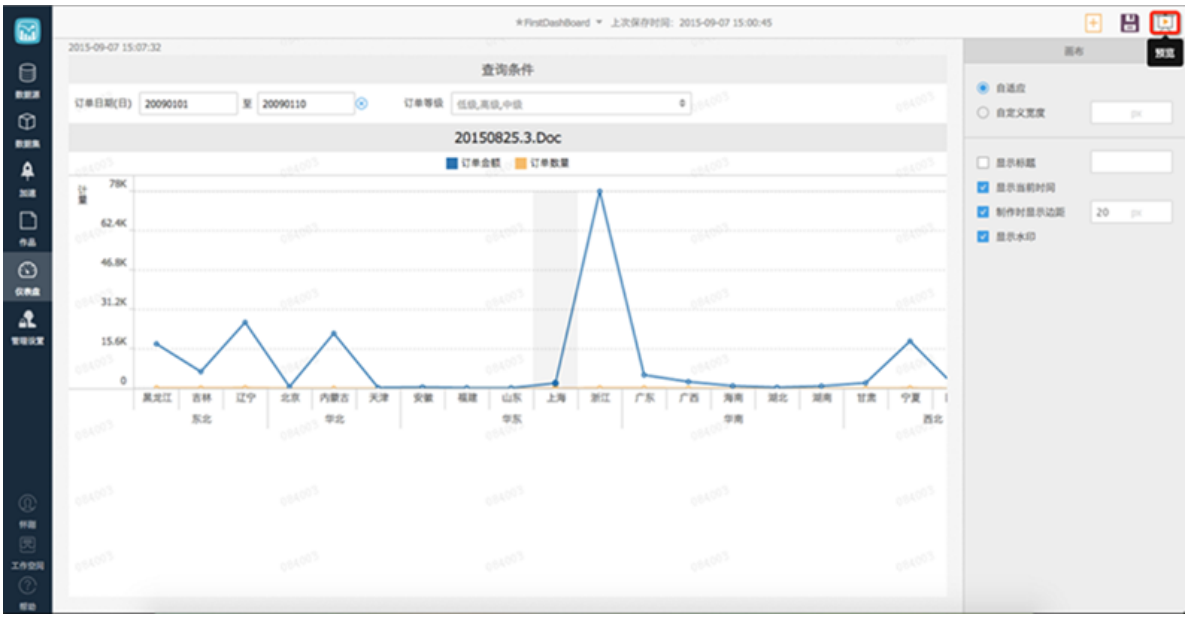
13.保存仪表盘，如图 *1763: 保存仪表盘*所示。

图 1763: 保存仪表盘



14.单击页面左上角的**预览**，预览仪表盘，如图 *1764:* 预览仪表盘所示。

图 1764: 预览仪表盘

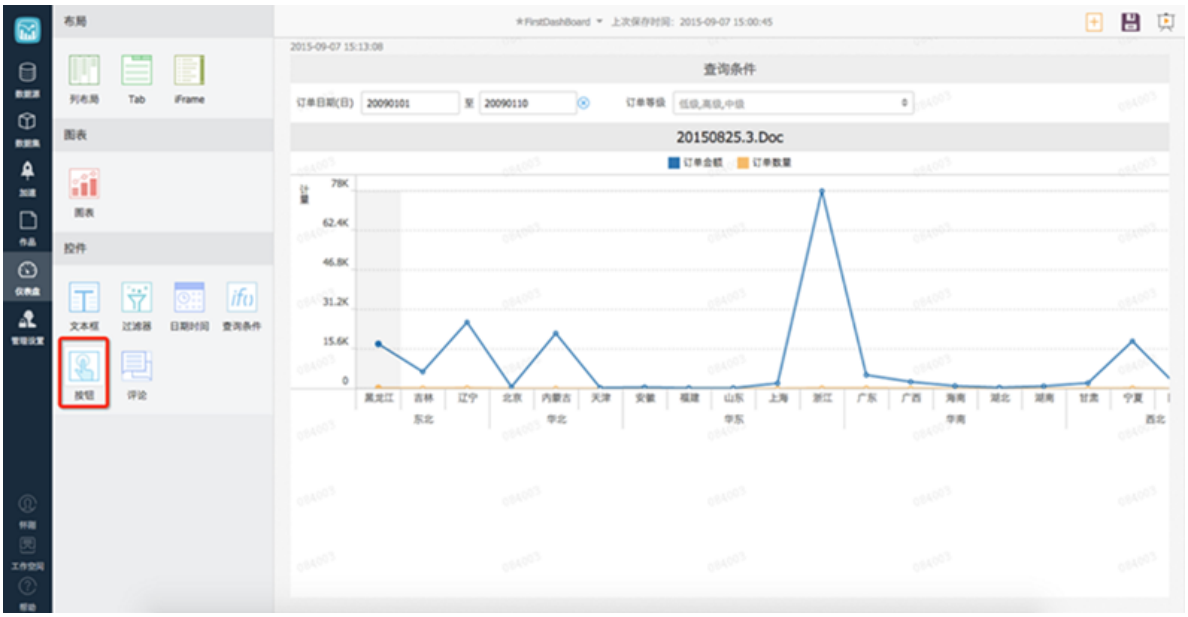


仪表盘退出编辑模式，进入预览模式。

15.单击右上角**关闭**按钮，关闭预览页面。

16.返回增加**按钮**控件，如图 *1765:* 按钮控件所示。

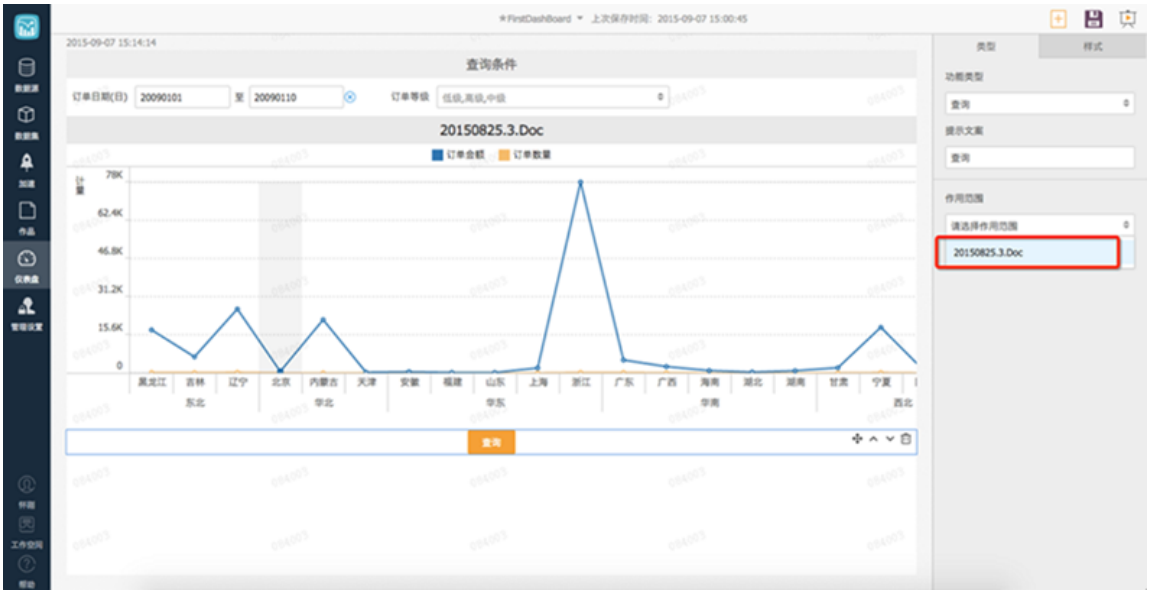
图 1765: 按钮控件



17.编辑**按钮**控件，关联到报表，并调整页面位置。

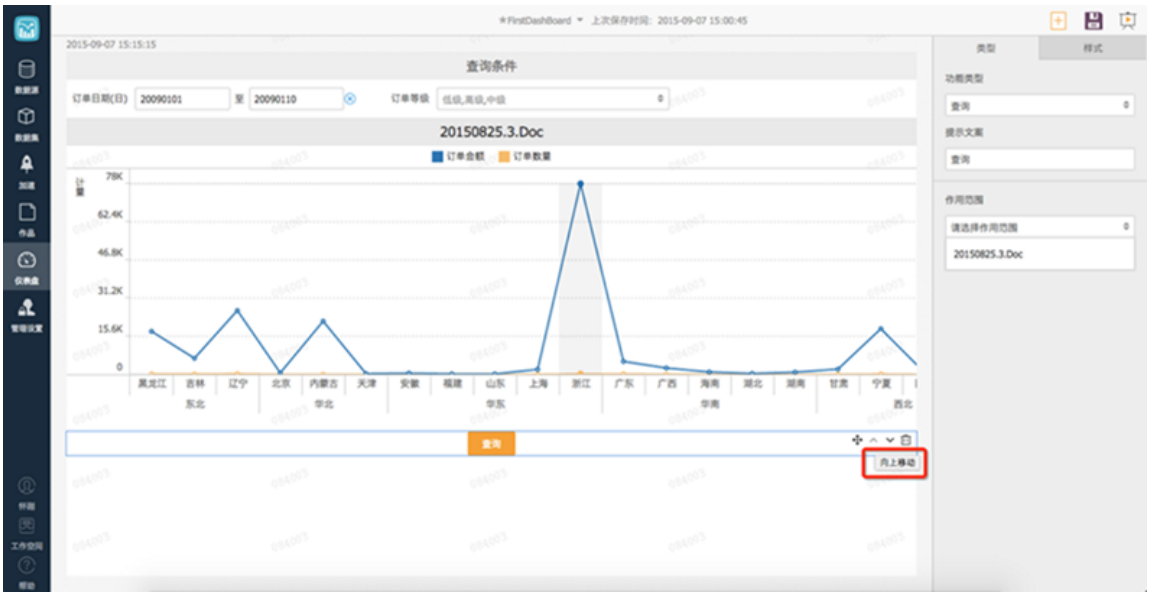
a) 选中添加的**按钮**控件，在右侧面板中将**按钮**控件关联到报表，如图 *1766: 关联报表*所示。

图 1766: 关联报表



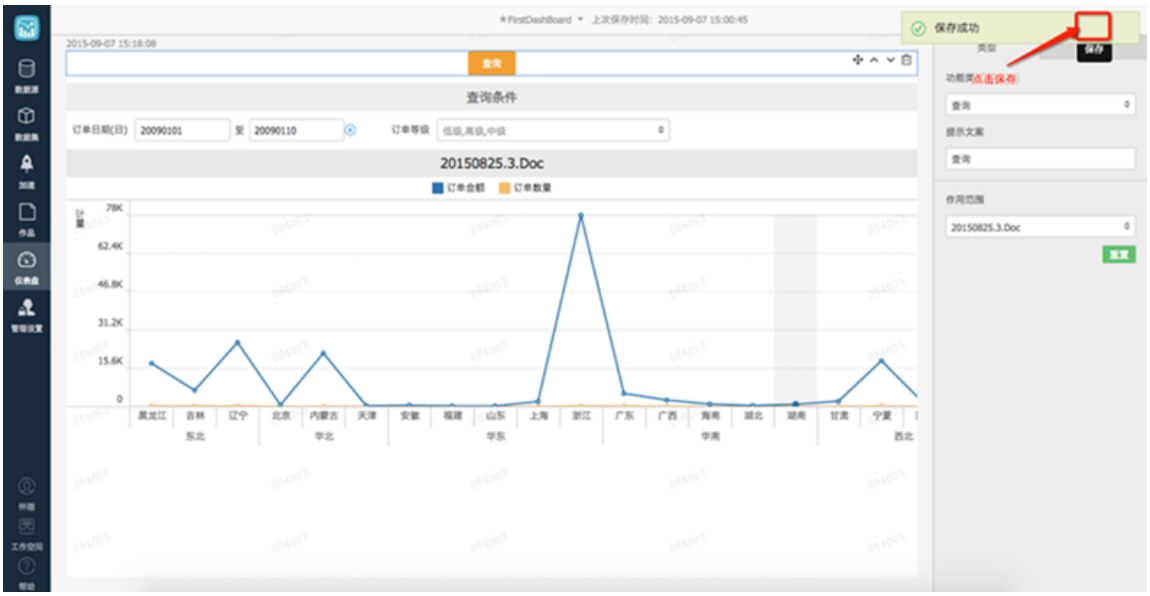
b) 调整控件所在的位置，如图 *1767: 调整位置*所示。

图 1767: 调整位置



c) 保存报表，如图 *1768: 保存报表*所示。

图 1768: 保存报表

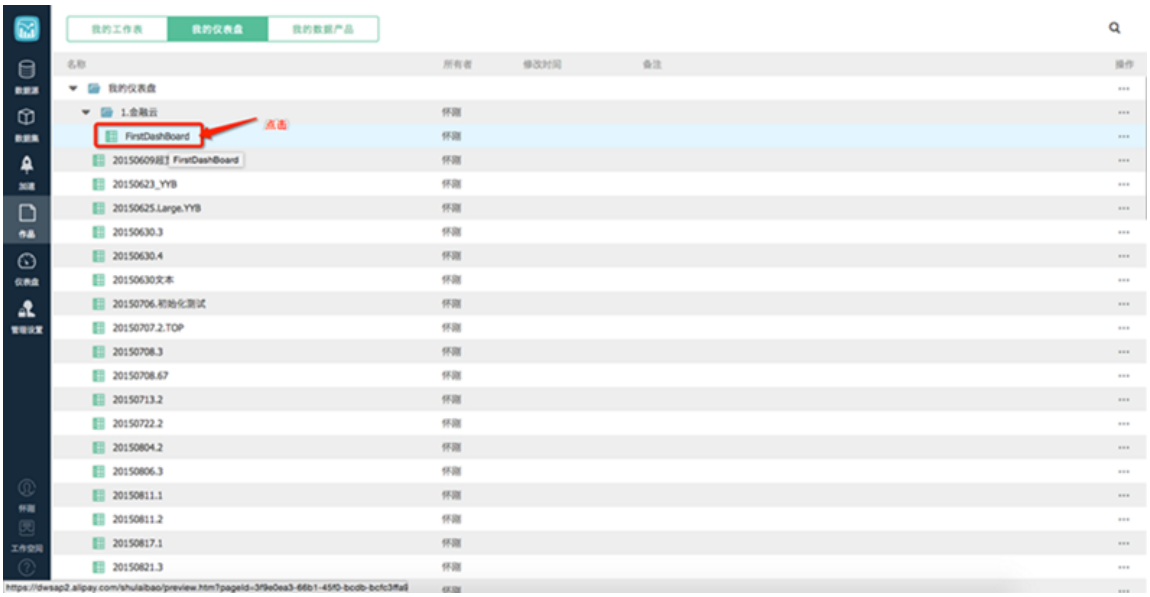


18.从报表树中打开阅读报表，并选择查询条件。

a) 在左侧导航树中，单击**作品**。

b) 在**我的仪表盘**中，选择某个报表并单击，如图 *1769: 打开报表*所示。

图 1769: 打开报表



c) 进行条件查询，在订单日查询条件中选择起止日期，单击**查询**按钮，报表内容相应变动，如图 *1770:* 查询订单日所示。

图 1770: 查询订单日



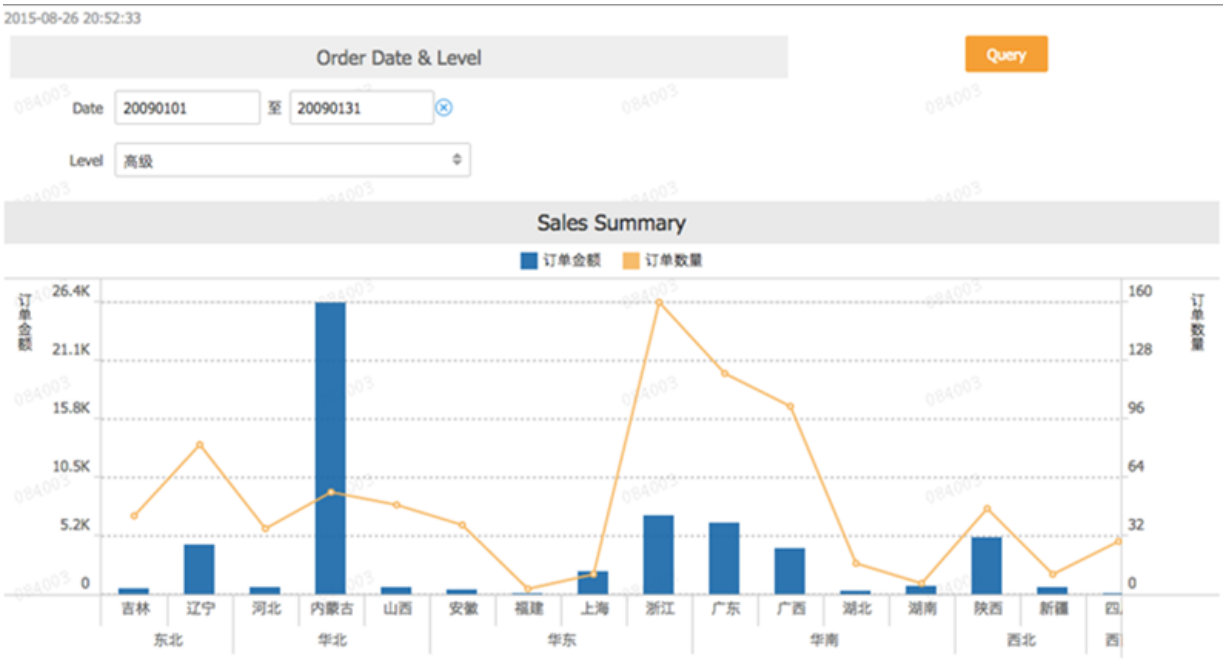
以上通过非常简单的操作，完成一张交互式仪表盘的制作。

33.5.3 仪表盘

仪表盘是一个显示报表数据的交互式页面，报表阅读者能够基于仪表盘设置数据查询条件，仪表盘能够根据设置的条件进行数据查询。

一个典型的仪表盘类似如图 *1771: 仪表盘*所示的样式。

图 1771: 仪表盘



33.5.4 数据产品

数据产品是通过菜单形式组织的仪表盘，通过数据产品可以制作更复杂的、带导航菜单的专题类分析。

33.5.4.1 新建数据产品

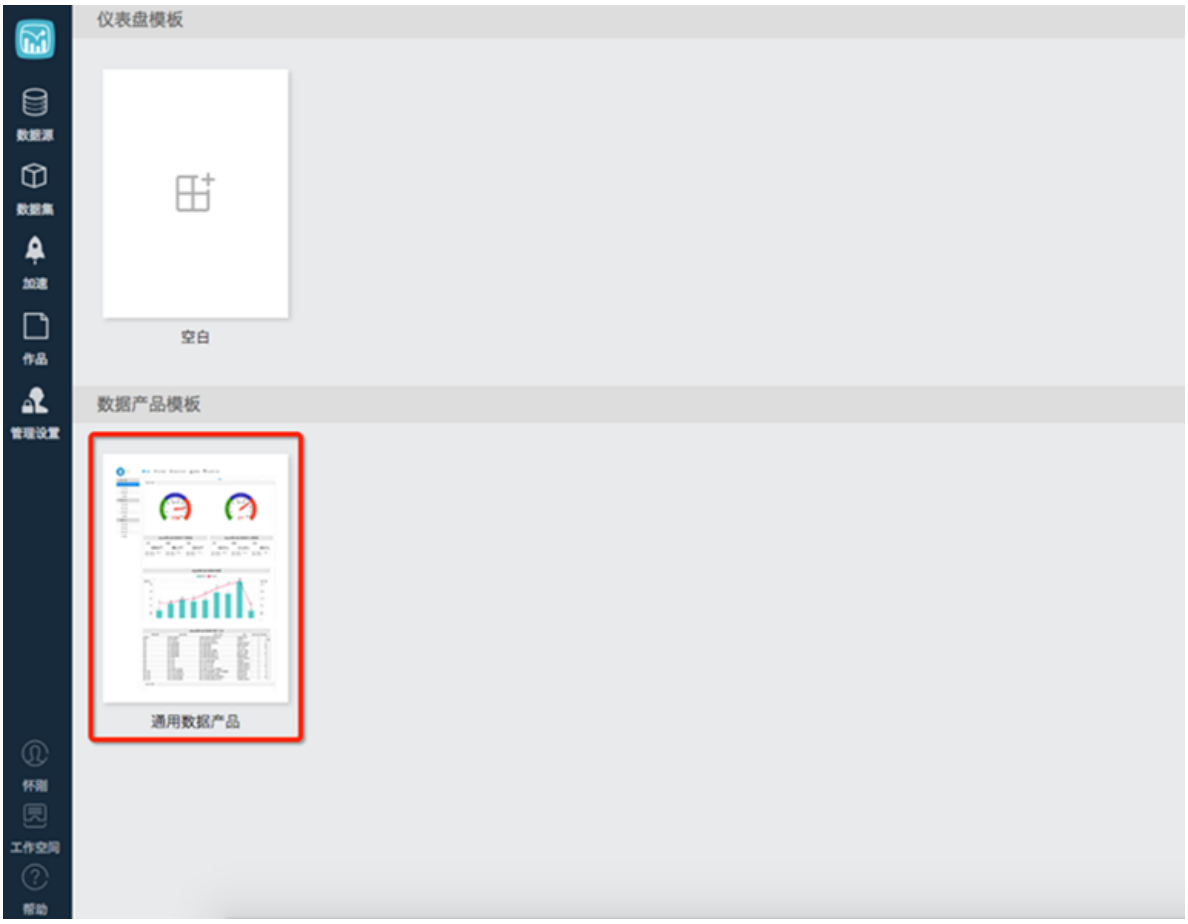
背景信息

可以按照如下步骤完成数据产品的新建过程。

操作步骤

1. 单击一个菜单模板，进入编辑页面，如图 *1772: 选择仪表盘模板*所示。

图 1772: 选择仪表盘模板



2. 针对菜单模板中的每个菜单项，选择报表，设计菜单，如图 *1773:* 设计菜单项所示。

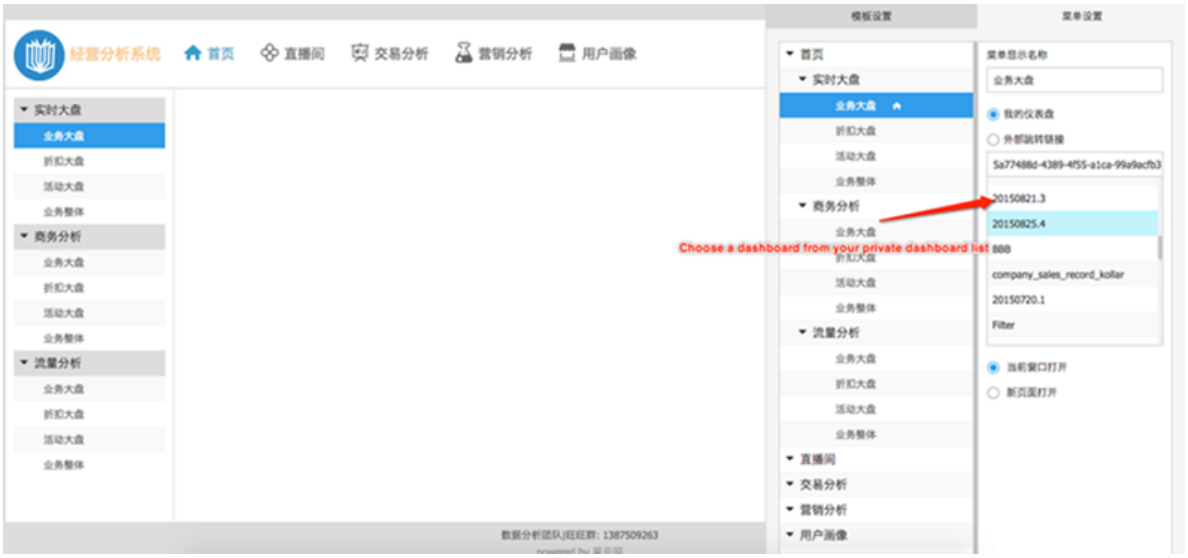
图 1773: 设计菜单项



3. 在**菜单设置**标签页中，设置菜单名称、单击菜单要打开的 URL 等信息。

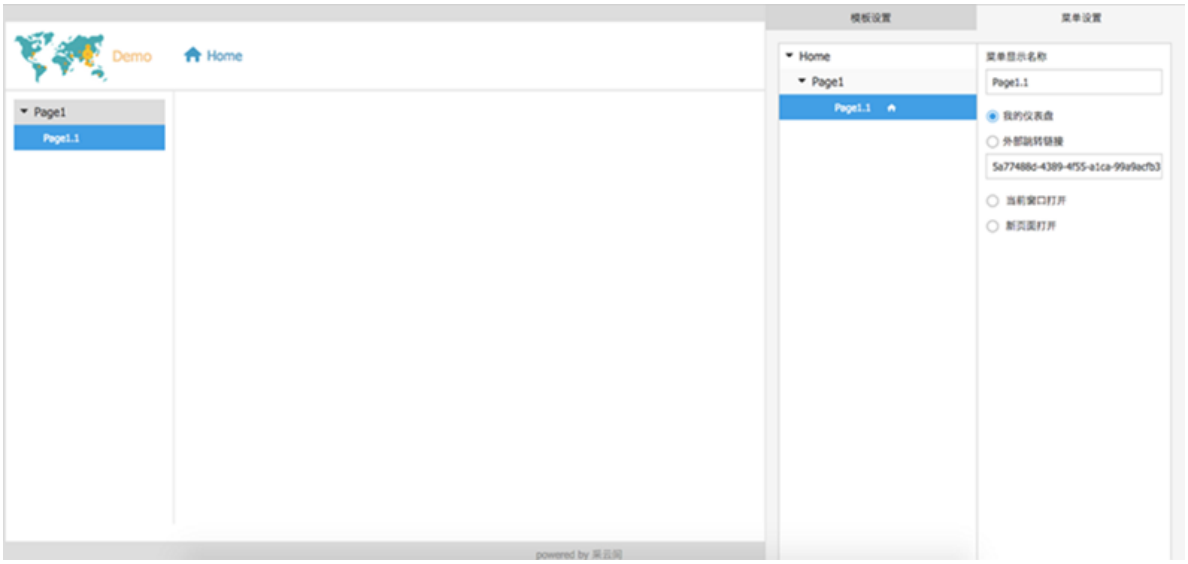
此处我们从私有仪表盘列表中选择一个已有的页面，如图 *1774:* 菜单设置所示。

图 1774: 菜单设置



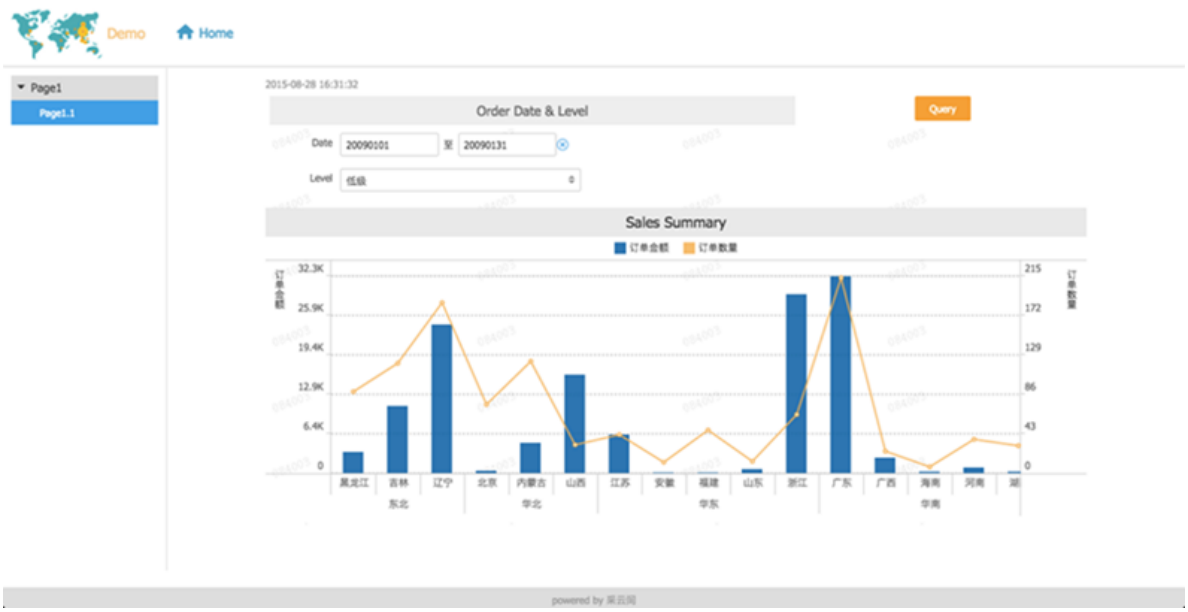
设置对应的菜单名称、URL 等信息之后，页面的变化如图 *1775:* 菜单设置完成所示。

图 1775: 菜单设置完成



4. 单击**预览**，在独立页面中预览，如图 *1776:* 预览所示。

图 1776: 预览

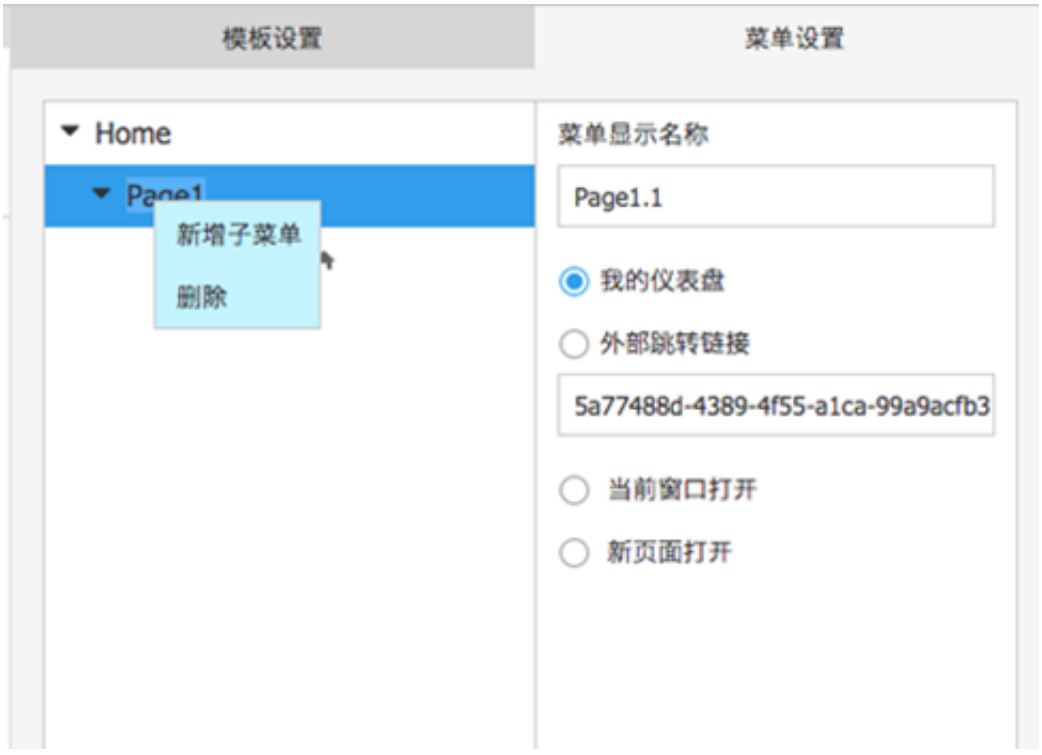


33.5.4.2 可编辑功能

包含如下可编辑功能：

- 菜单结构的自定义，如图 *1777:* 自定义菜单结构所示；

图 1777: 自定义菜单结构



- 支持对仪表盘的页面引用，以及第三方页面的引用；

- 支持对数据产品模板属性的编辑，如图 *1778:* 编辑模板属性所示；

图 1778: 编辑模板属性



- 支持数据产品的预览。

33.5.5 画布

画布是容纳其他页面元素的容器，控件等页面元素都在画布上。

在画布上，您可以选择：

- 自适应画布宽度；
- 设定画布宽度。

在报表制作首页，选择**新建仪表盘**，进入仪表盘页面后，在右侧面板中可以进行画布设置，如图 *1779:* 画布设置所示。

图 1779: 画布设置



33.5.6 布局

33.5.6.1 列布局

列布局是一个容器，用于将多个控件（图表）放置在画布的一行内。

选中一个列，可以添加控件。

在列布局中：

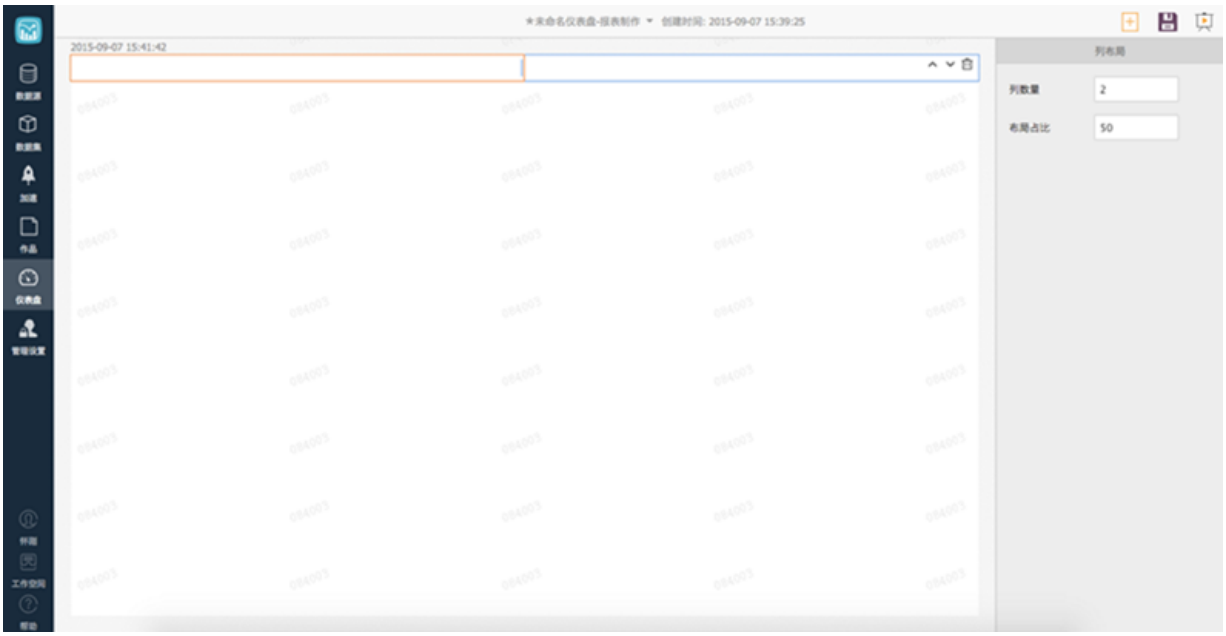
- 支持设置多列的布局；
- 支持设置列的宽度比例；

在**布局**中选择**列布局**后，展示两列的布局，如图 *1780: 选择列布局*和图 *1781: 展示两列的布局*所示。

图 1780: 选择列布局

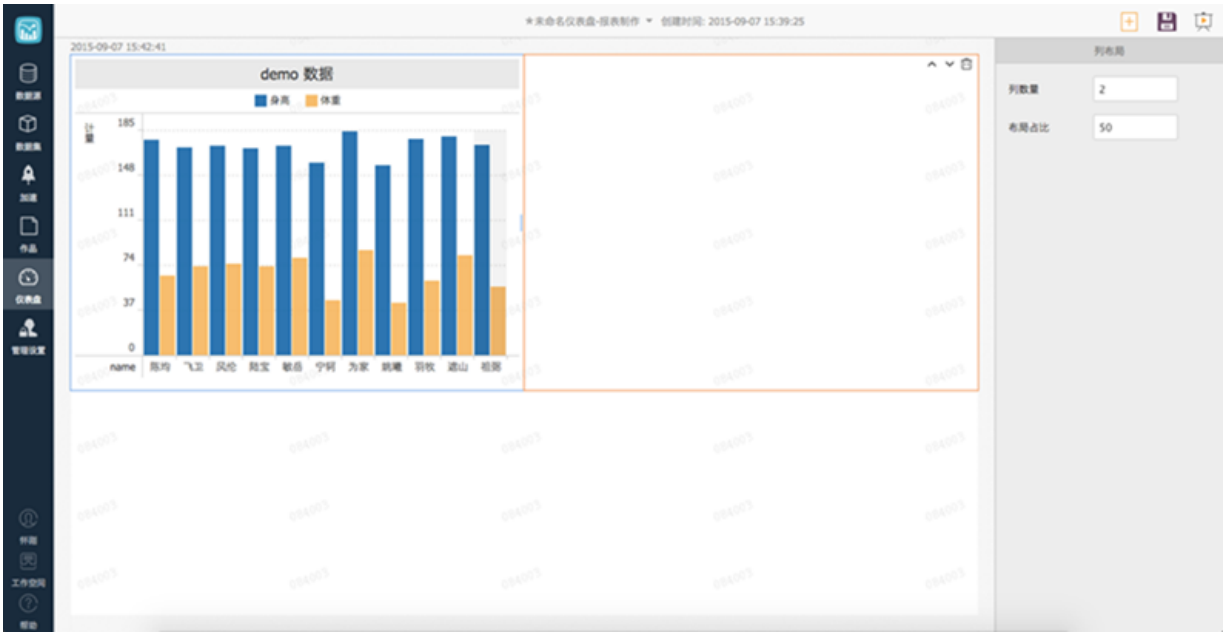


图 1781: 展示两列的布局



将图表控件置于第一列中，如图 *1782: 放置图表控件*所示。

图 1782: 放置图表控件



33.5.6.2 TAB 页

TAB 页布局是以 TAB 标签页方式进行的画布布局。

每个 TAB 页内的内容之间互相独立。

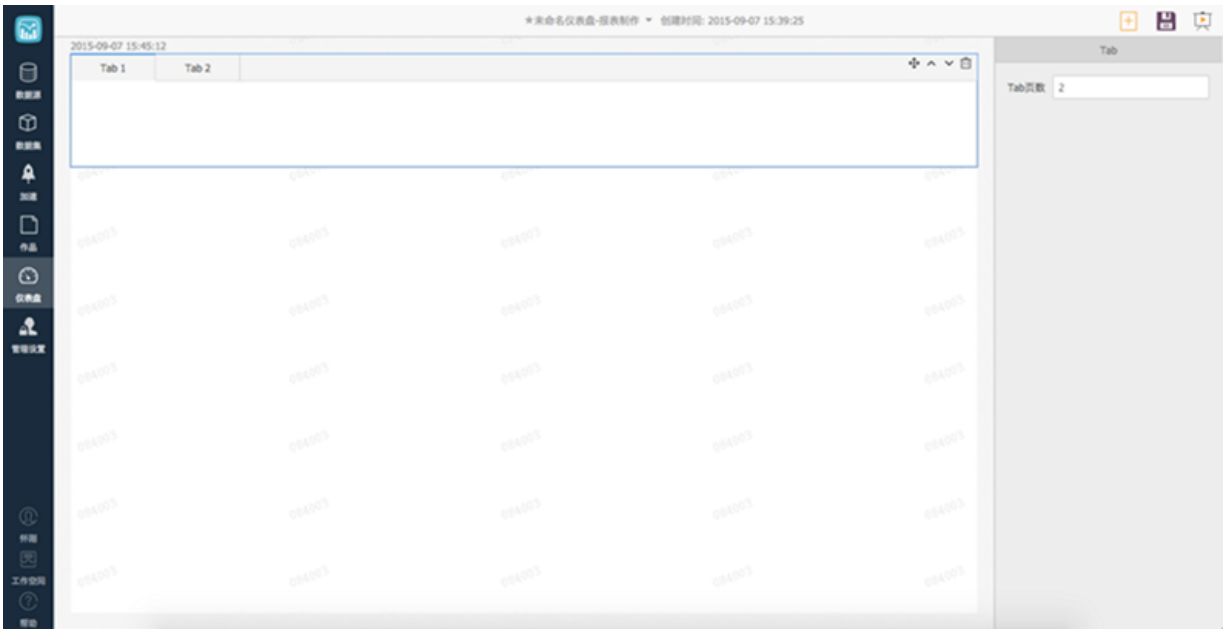
报表在单击 TAB 页的时候，完成页面加载。

在**布局**中选择**Tab**后，进入TAB 标签页，如图 *1783: 选择 Tab*和图 *1784: 进入Tab 标签页*所示。

图 1783: 选择 Tab



图 1784: 进入Tab 标签页



33.5.6.3 WEB 页面

Web 页面以 iFrame 嵌入的方式，嵌入一个独立的 URL 网页。

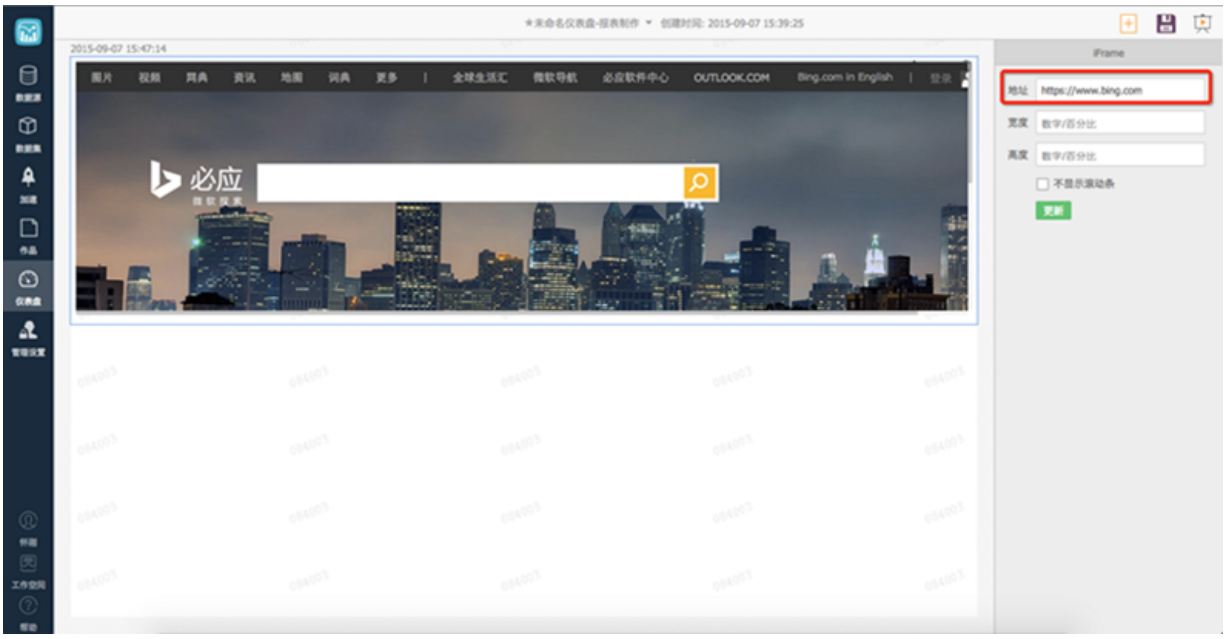
嵌入的 URL必须以 HTTPS 方式提供。

在**布局**中选择**iFrame**，进入 WEB 页面，如图 *1785: 选择 iFrame*和图 *1786: WEB 页面*所示。

图 1785: 选择 iFrame


布局
列布局
Tab
iFrame
图表
图表
控件
文本框
过滤器
日期时间
查询条件
按钮
评论
数据源
数据集
加速
作品
仪表盘
管理设置
收藏
工作空间
帮助

图 1786: WEB 页面

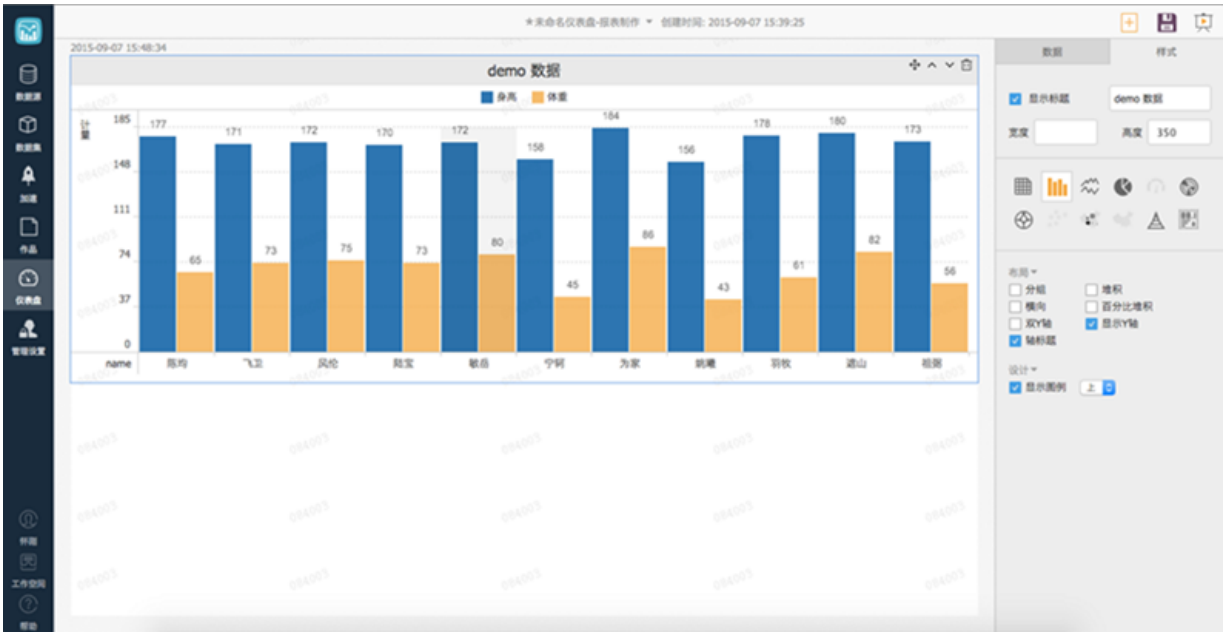


33.5.7 图表

33.5.7.1 图表简介

如图 *1787: 图表组件*所示，系统通过图表组件显示数据。

图 1787: 图表组件



图表组件提供多种类型的数据显示，系统能够根据要查询的数据结构的维度和计量特征，智能推荐数据的显示形式。

包括以下数据显示类型：

- 交叉表
- 柱状图
- 折线图
- 饼图
- 油量表
- 雷达图
- 极地坐标图
- 散点图
- 地图气泡图
- 地图色彩图
- 漏斗图
- 指标看板

33.5.7.2 数据源

图表组件的数据源支持以下数据源：

- 来自工作表：关联显示某个工作表中的数据内容

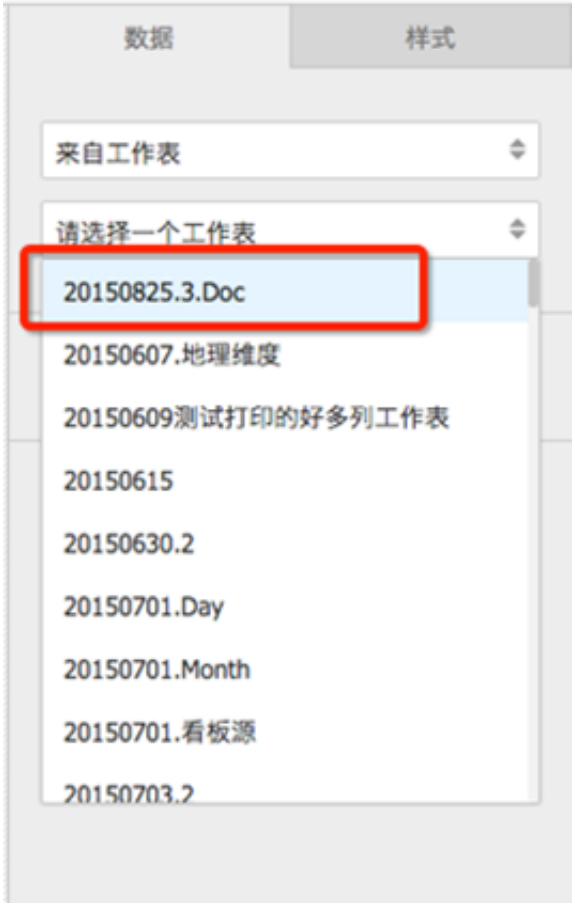
  1. 选择**来自工作表**为数据源类型，如图 *1788:* 选择来自工作表所示。

     图 1788: 选择来自工作表



2. 从工作表列表中选择一个工作表，如图 *1789:* 选择工作表所示。

图 1789: 选择工作表



3. 图表控件关联到工作表，如图 *1790:* 关联到工作表所示。

图 1790: 关联到工作表



- 来自数据集：关联显示某个数据集中的数据内容

1. 选择**来自数据集**为数据源类型，如图 *1791:* 选择来自数据集所示。

图 1791: 选择来自数据集



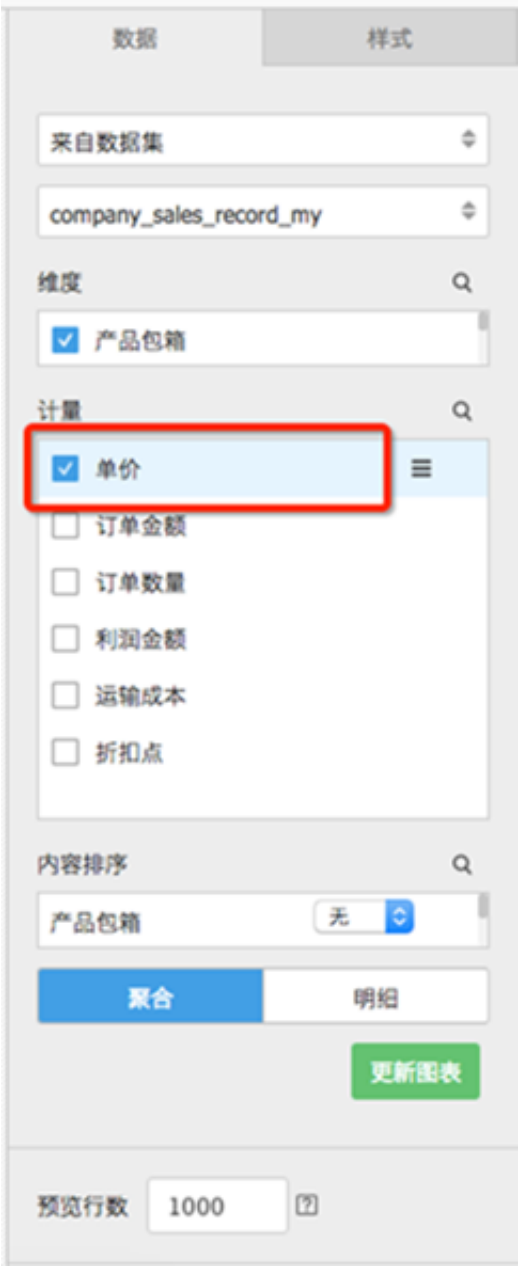
2. 选择数据集中需要显示的维度，如图 *1792:* 选择维度所示。

图 1792: 选择维度



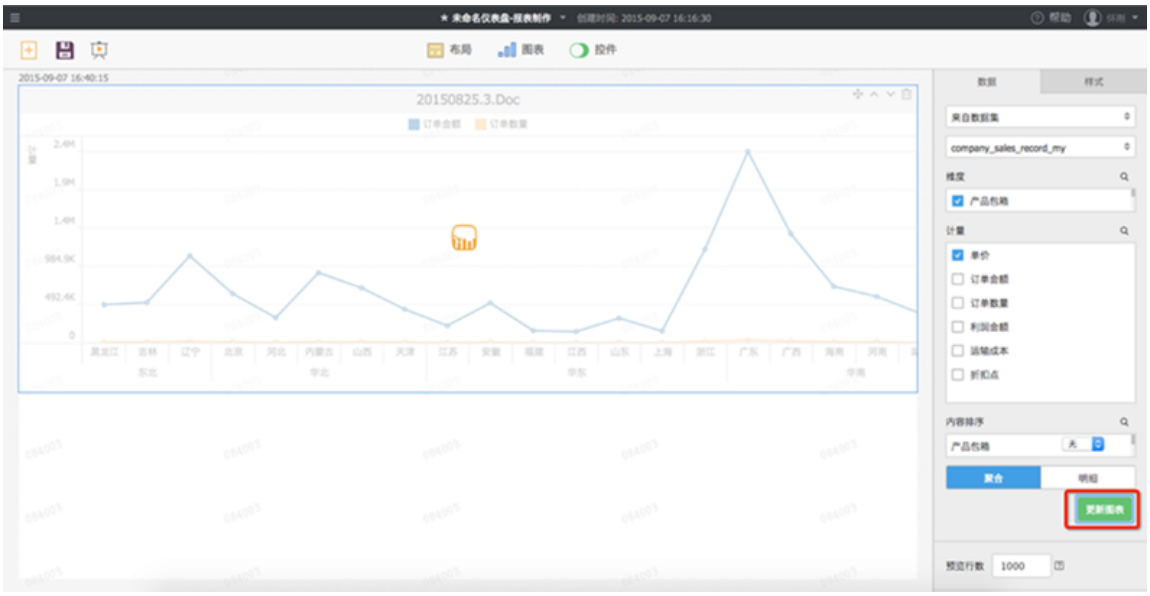
3. 选择数据集中需要显示的计量，如图 *1793:* 选择计量所示。

图 1793: 选择计量



4. 更新图表显示，如图 *1794:* 更新图表显示所示。

图 1794: 更新图表显示



图表组件的其他特性：图表组件的数据源能够被其他页面控件中的条件所过滤，能够设置单次页面的最大显示条数，避免因为数据量过多导致页面占用系统资源过大。

33.5.7.3 图表样式

在样式标签页中，进行对图表样式的编辑。

33.5.7.3.1 交叉表

交叉表形式展示的数据格式与工作表展示格式相同，即可以直接以工作表的样式，在仪表盘中显示数据，如图 *1795:* 交叉表所示。

图 1795: 交叉表



支持单击列头进行排序，如图 *1796:* 排序所示。

图 1796: 排序



33.5.7.3.2 柱状图

以 Demo 数据为例，柱状图可以通过选择右侧面板中布局方式的不同组合，如分组、堆叠、填充等，显示不同的效果，例如图 *1797: 柱状图*所示。

图 1797: 柱状图



报表阅读者界面同样可在在报表阅读过程中，进行部分属性的设置，如图 *1798: 设置属性*所示。

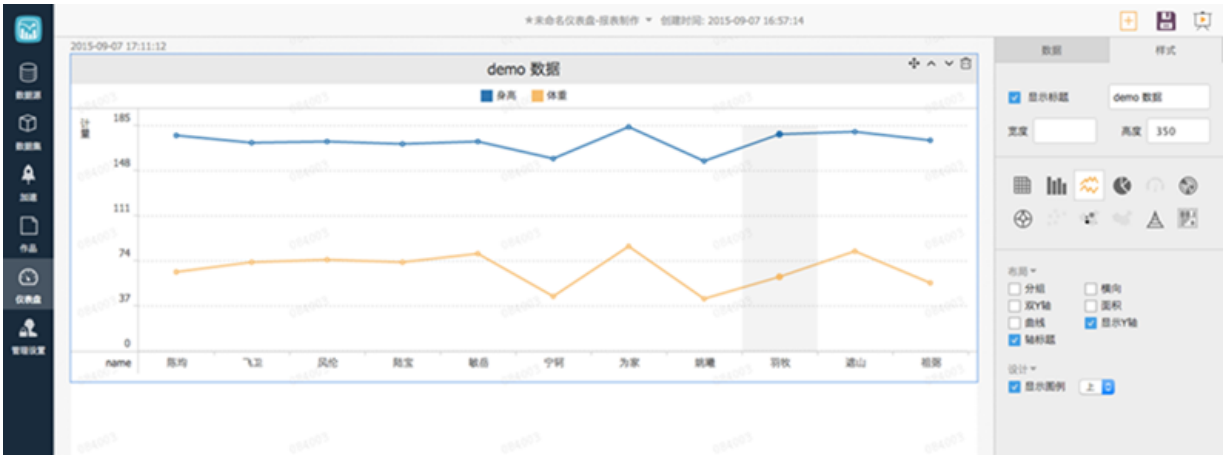
图 1798: 设置属性



33.5.7.3.3 曲线图

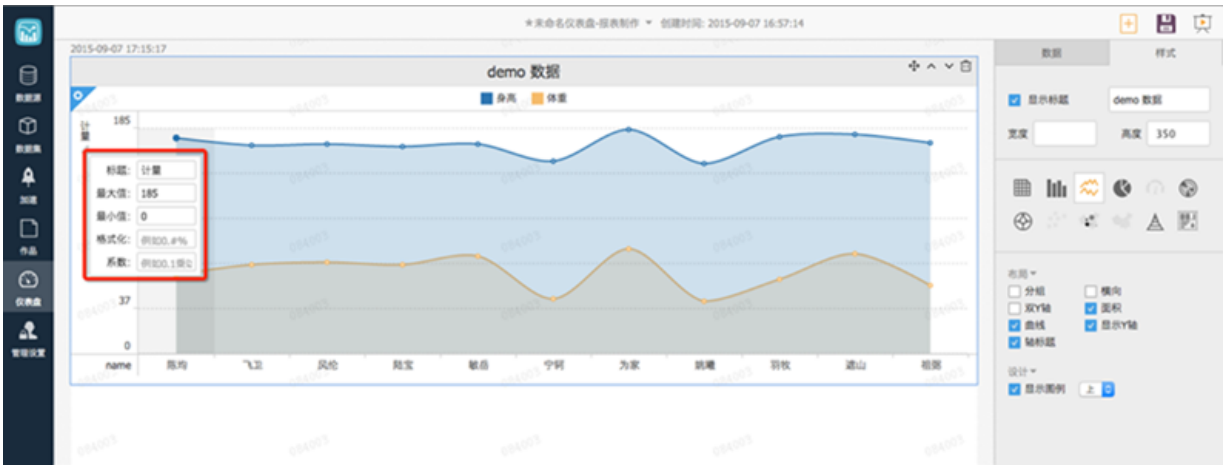
以 Demo 数据为例，曲线图可以通过选择右侧面板中布局方式的不同组合，如分组、横向、曲线等，显示为不同效果，例如图 *1799:* 曲线图所示。

图 1799: 曲线图



报表阅读者界面同样可在报表阅读过程中，进行部分属性的设置，如图 *1800:* 属性设置所示。

图 1800: 属性设置



33.5.7.3.4 饼图

以 Demo 数据为例，饼图可以通过在右侧面板中选择布局方式的不同选项，如默认、环状和3D ，显示为不同效果，如图 *1801: 默认图*、图 *1802: 环状图*和图 *1803: 3D 图*所示。

图 1801: 默认图



图 1802: 环状图



图 1803: 3D 图



33.5.7.3.5 油量表

油量表用于显示没有维度的数据结构，如图 *1804:* 油量表所示。

图 1804: 油量表



33.5.7.3.6 雷达图

以 Demo 数据为例，曲线可以显示为如图 *1805: 雷达图*所示效果。

图 1805: 雷达图



33.5.7.3.7 极地坐标图

以 Demo 数据为例，曲线可以显示为如图 *1806: 极地坐标图*所示效果。

图 1806: 极地坐标图



33.5.7.3.8 散点图

如果其中某个维度的取值有 5 种，则推荐散点图的显示如图 *1807:* 散点图所示。

图 1807: 散点图



33.5.7.3.9 地图气泡图

在数据集中存在地理维度的时候，数据集能够以地图方式进行数据显示。

支持区域、省、市三级显示，以省维度举例显示如图 *1808:* 地图气泡图所示。

图 1808: 地图气泡图



33.5.7.3.10 地图色彩图

在数据集中存在地理维度的时候，数据集能够以地图方式进行数据显示。

支持区域、省、市三级显示，以省维度举例显示如图 *1809:* 地图色彩图所示。

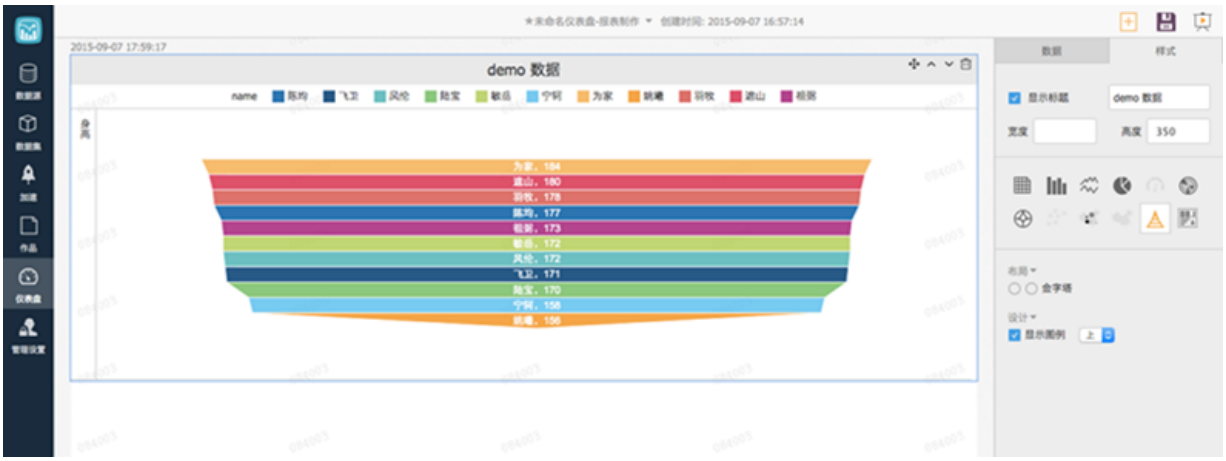
图 1809: 地图色彩图



33.5.7.3.11 漏斗图

漏斗图如图 *1810:* 漏斗图所示。

图 1810: 漏斗图



33.5.7.3.12 指标看板

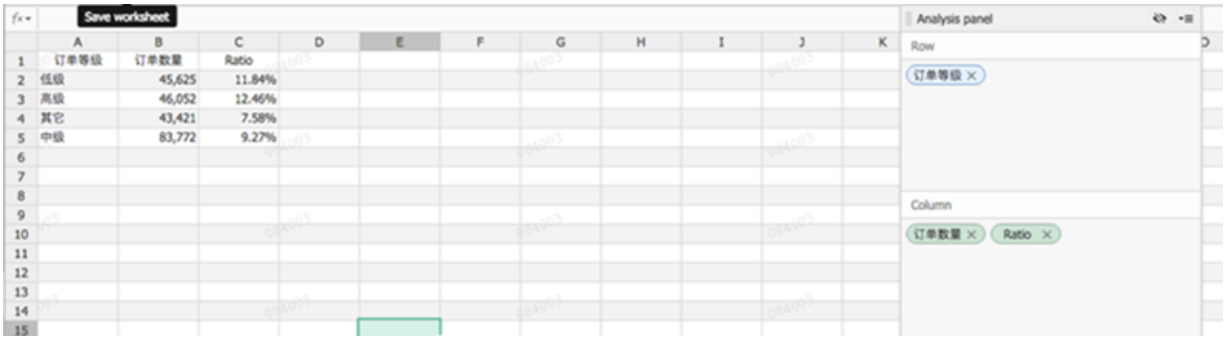
如果工作表存在如下数据结构，使用指标看板样式能够更好地进行数据的展示。

工作表的数据结构如下：

1. 维度
2. 指标
3. 指标变化率

例如，工作表的结构如图 *1811:* 工作表结构所示。

图 1811: 工作表结构



指标看板可以通过在右侧面板中选择不同模板，展现如图 *1812:* 模板*1*、图 *1813:* 模板*2*和图 *1814:* 模板*3*三种样式。

图 1812: 模板1



图 1813: 模板2



图 1814: 模板3



33.5.8 控件

33.5.8.1 控件简介

控件用于过滤页面图表组件的显示数据，并提供其他的辅助功能，比如，导出等。

如图 *1815: 控件*所示，包含如下控件：

- 文本框：用于各种固定文本的输入；
- 过滤器：用于对数据集（包括工作表关联到的数据集）的字段过滤；
- 日期时间：是针对日期类型的特殊的一种过滤器；
- 查询条件：针对数据集中的维度，进行多个条件的设置，相当于过滤器的组合；
- 按钮：触发查询请求和触发导出操作；
- 评论：提供报表阅读者填写信息的组件，相关信息被报表阅读者共享。

图 1815: 控件



33.5.8.2 文本框

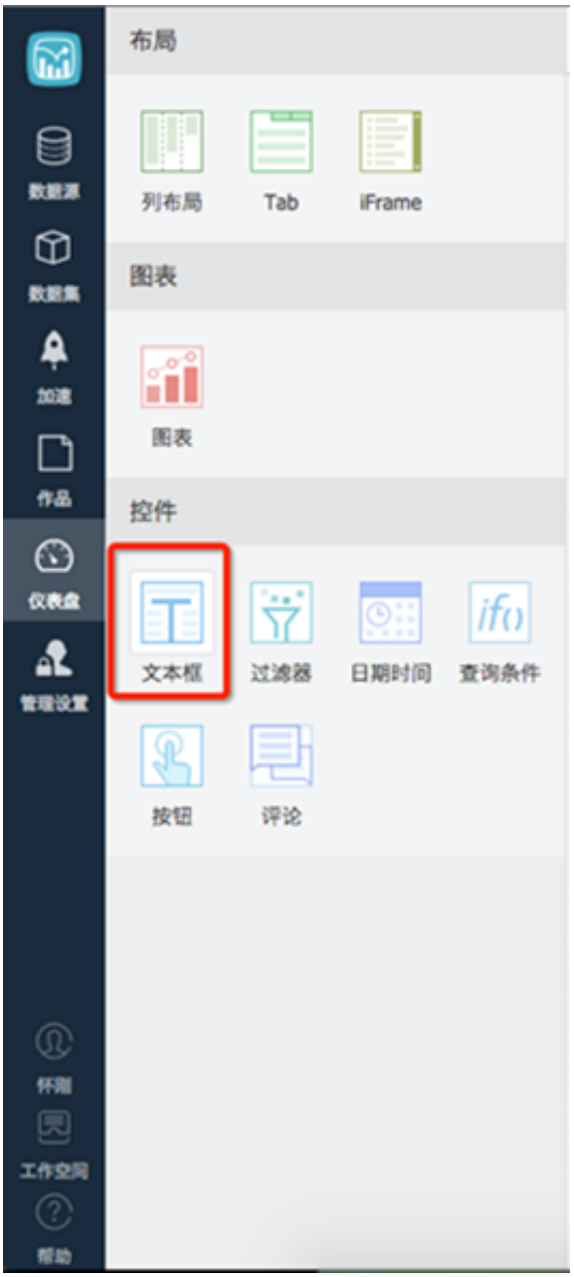
文本框用于一段固定文本的输入。

文本框支持富文本的编辑模式。

文本框可用于制作报表的标题等。

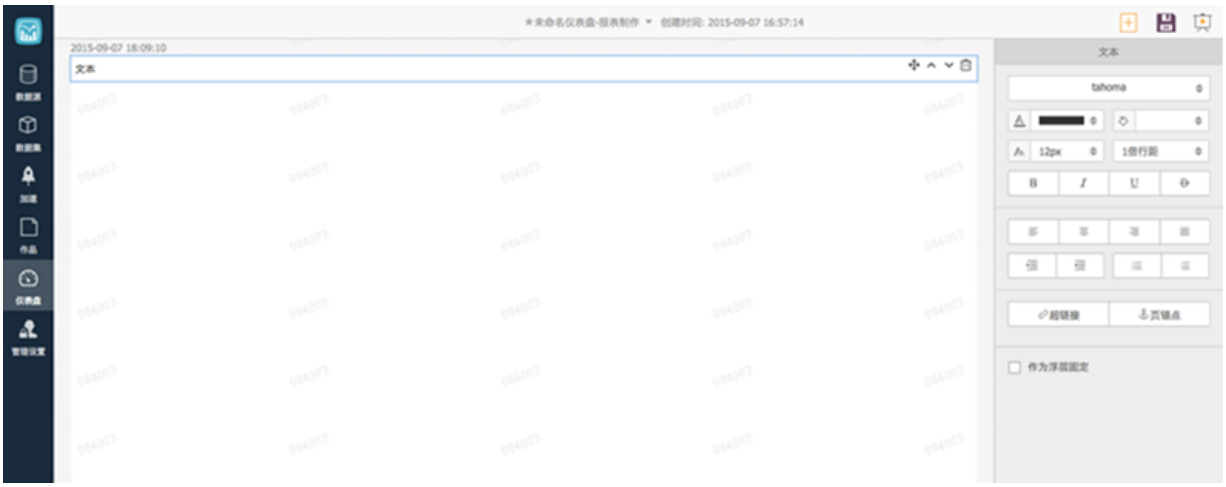
文本框控件入口如图 *1816: 文本框入口*所示。

图 1816: 文本框入口



文本框界面如图 *1817: 文本框界面*所示。

图 1817: 文本框界面



33.5.8.3 过滤器

过滤器用于对数据集字段的条件过滤，其中的数据集是指图表组件关联到的数据集。

属性设置

1. 关联到图表的数据集，如图 *1818:* 关联到图表的数据集所示。

图 1818: 关联到图表的数据集



2. 选择过滤字段，如图 *1819:* 选择过滤字段所示。

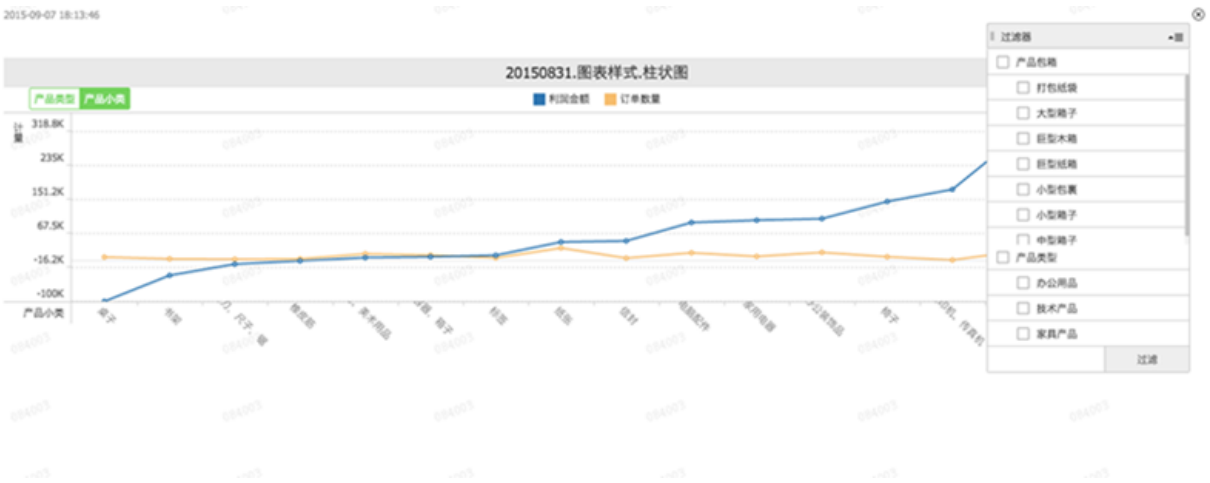
图 1819: 选择过滤字段



阅读者界面

报表阅读者能设置字段的过滤条件，然后单击**过滤**，即可完成对数据的过滤操作，如图 *1820: 阅读者界面*所示。

图 1820: 阅读者界面



约束条件

使用过滤器需要遵循如下几个约束条件：

- 一个过滤器只能过滤一个数据集；
- 对于时间类型字段需要通过日期时间控件完成过滤；
- 各个条件以枚举值形式显示，查询中以 AND 关系共同作用到数据集。

33.5.8.4 日期时间

时间控件能够完成对日期类型维度的过滤，能够设置单个时间，也可以设置时间区间。

属性设置

1. 关联到图表的数据集，如图 *1821:* 关联到图表的数据集所示。

图 1821: 关联到图表的数据集



2. 设置初始化时间。

   通过时间变量 T 设置，这样设置的时间是一个动态时间，例如 T-1 到 T ，表示每次打开报表显示的时间为前一天时间。

   也可以通过页面设置初始化时间，如图 *1822:* 设置初始化时间所示。

图 1822: 设置初始化时间



阅读者界面

报表阅读者界面可以设置时间控件中的时间，系统在设置完成后，自动刷新数据。

约束条件

使用日期时间控件需要遵循如下两个约束条件：

- 时间控件中的时间范围是范围两头闭区间；
- 时间控件中的年、周、月、日，是选择时间的快捷方式，不代表选择数据粒度。

33.5.8.5 查询条件

查询条件是数据集中各个字段设置查询的过滤条件，如图 *1823: 查询条件示例*所示，查询条件需要配合按钮控件一同使用，通过按钮触发查询条件的查询动作。

图 1823: 查询条件示例



属性设置

- 文件形式：

  对于数据量比较大的场景，支持上传文件作为字段匹配条件，如图 *1824: 文件形式*所示。

  图 1824: 文件形式



- 条件匹配：如图 *1825: 条件匹配*所示。

  图 1825: 条件匹配



- 枚举值：如图 *1826: 枚举值*所示。

图 1826: 枚举值



- 数字形式：

  数字型字段选择如图 *1827: 数字形式*所示方式。

图 1827: 数字形式



- 日期形式：

  类似日期控件，同样支持快捷键的选择，如图 *1828:* 日期形式所示。

  图 1828: 日期形式



约束条件

使用查询条件控件需要遵循如下几个约束条件：

- 查询条件需要配合查询按钮一并使用；
- 各个查询条件之间关系为 AND；
- 一个查询条件关联到一个数据集，影响到使用该数据集的所有图表控件。

33.5.8.6 按钮

按钮有以下两个作用：

- 触发页面查询，刷新页面数据；
- 触发页面数据的导出功能依赖租户的配置参数。

一个按钮需要关联到一个工作表或者数据集。

属性设置

- 查询：如图 *1829:* 查询所示。

  图 1829: 查询



- 导出：

  导出功能根据租户的不同而不同，一个复杂的配置如图 *1830:* 导出所示。

  图 1830: 导出



- 按钮样式：如图 *1831:* 按钮样式所示。

图 1831: 按钮样式



约束条件

使用按钮需要遵循如下两个约束条件：

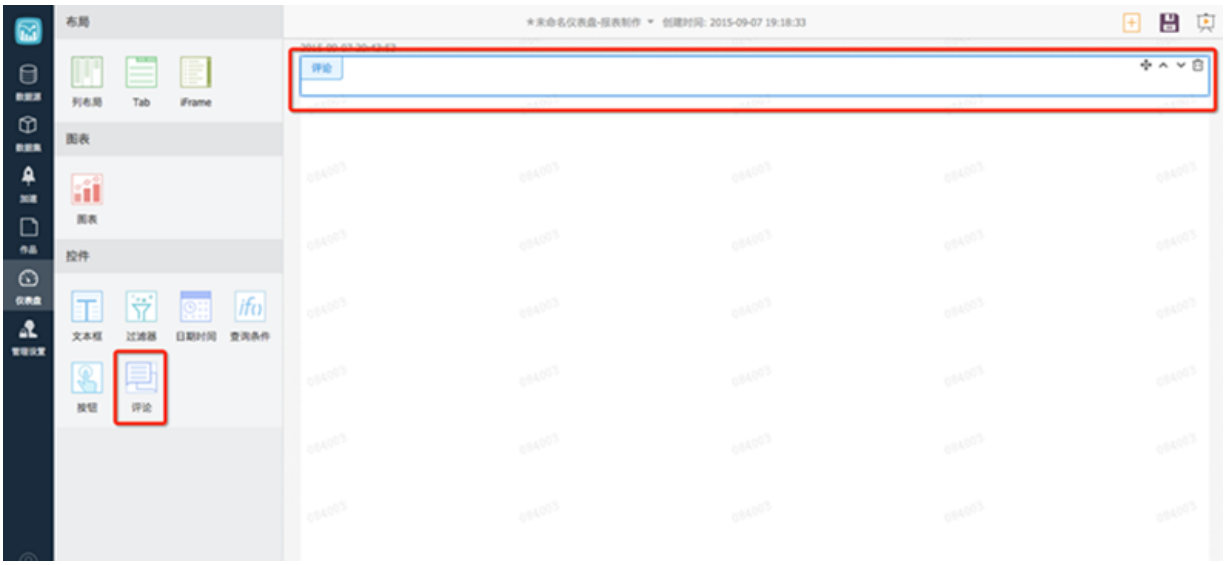
- 导出的目的地根据租户设置不同而不同；
- 导出为异步的后台任务。

33.5.8.7 评论

评论控件是报表阅读者填写并被报表阅读者分享的一段文本。

评论控件如图 *1832:* 评论所示。

图 1832: 评论



转让、分享、发布功能可作用于数据集、工作表、仪表盘和数据产品。

33.5.9 转让

通过转让功能，修改报表的作者。

在**我的工作表**中，右键单击某个报表，选择**转让**后，在弹出的对话框中选择转让对象即可将报表转让给其他人，如图 *1833: 转让*和图 *1834: 选择转让对象*所示。

图 1833: 转让

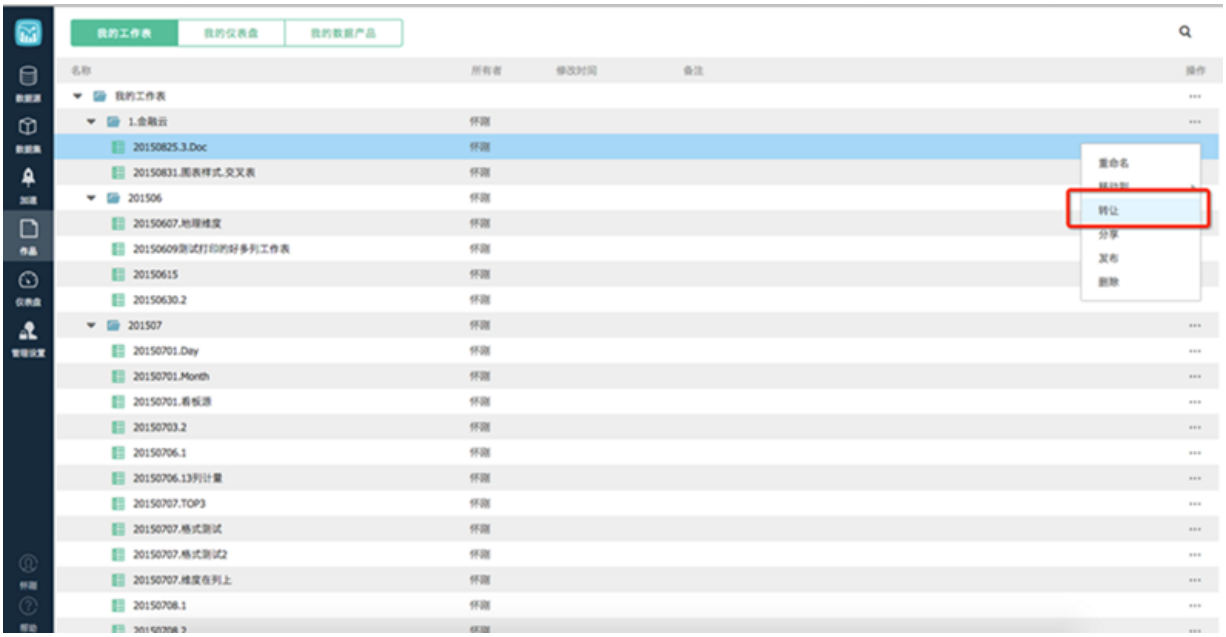
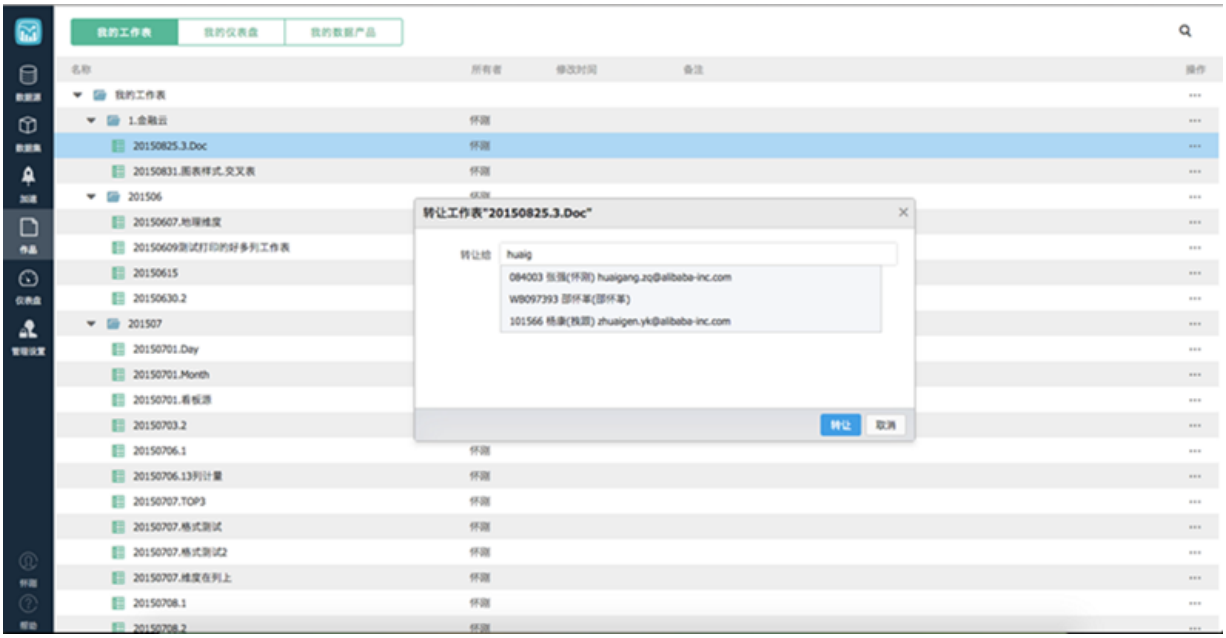


图 1834: 选择转让对象



33.5.10 分享

通过分享功能，报表制作者可以授予报表阅读者阅读报表的权限。

分享的有效期可以通过租户配置功能进行设置。

在**我的工作表**中，右键单击某个报表，选择**分享**后，在弹出的对话框中选择分享对象和有效期，即可将报表分享给其他人，如图 *1835: 分享*和图 *1836: 选择分享对象及有效期*所示。

图 1835: 分享

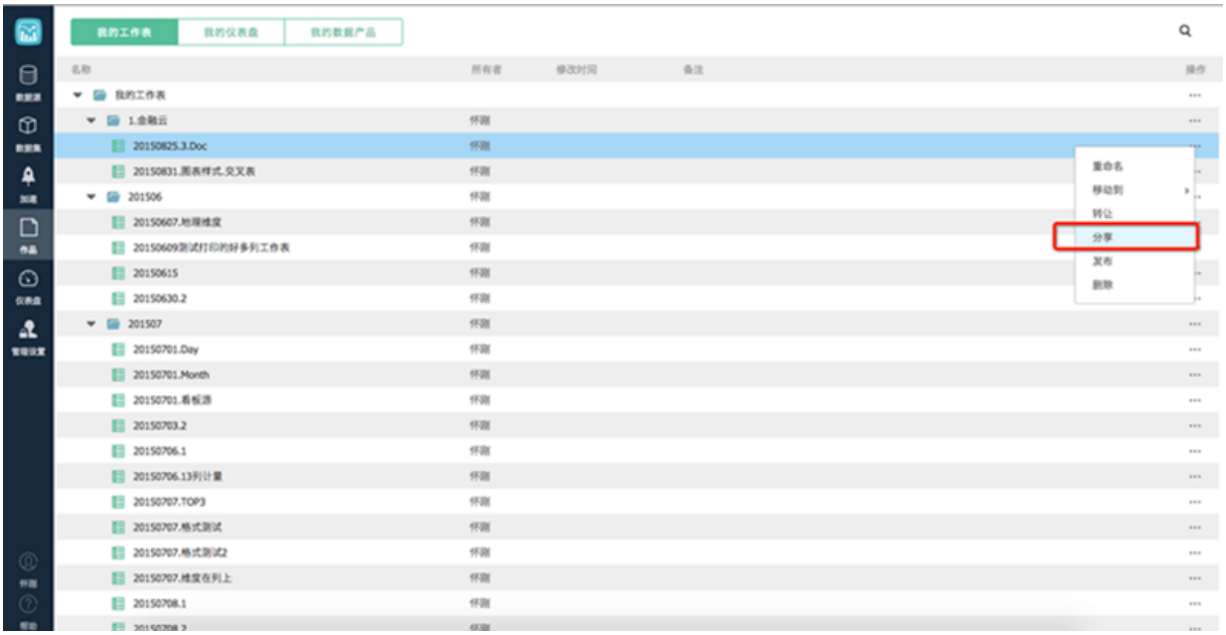
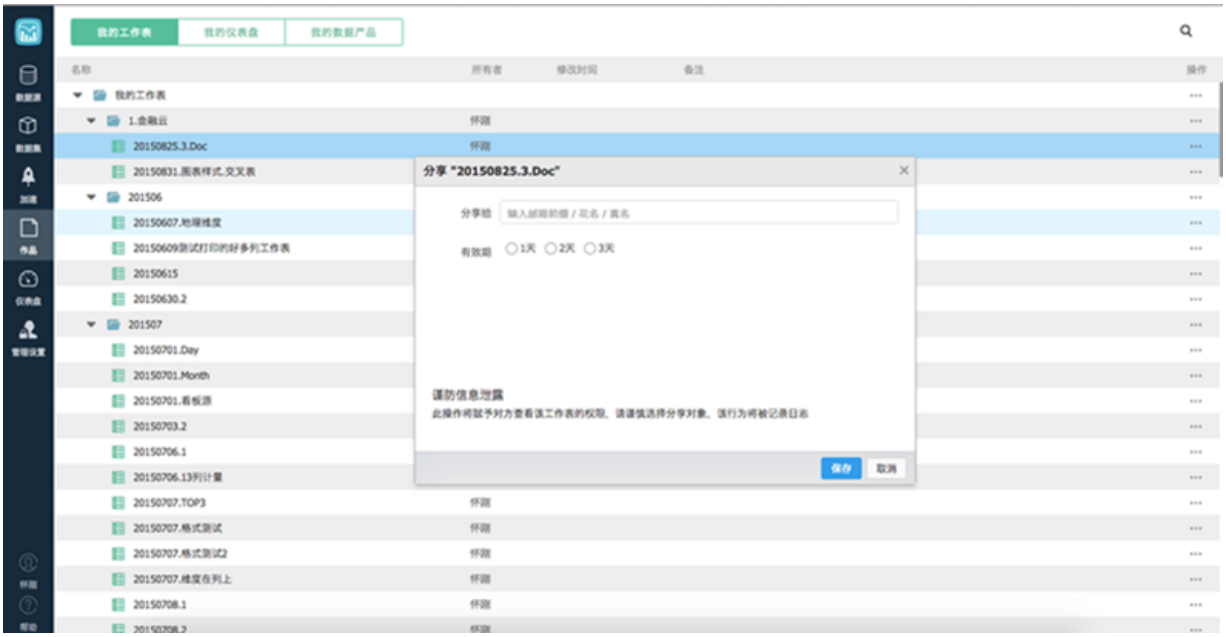


图 1836: 选择分享对象及有效期



33.5.11 发布

通过发布功能，报表制作者将报表发布至门户页面，报表阅读者可以申请报表的阅读权限。

说明: 发布功能需要结合特定的权限系统存在，故在某些配置环境中，不提供该功能。

在**我的工作表**中，右键单击某个报表，选择**发布**后，在弹出的对话框中填写发布信息，如图 *1837: 发布*和图 *1838: 填写发布信息*所示。

图 1837: 发布

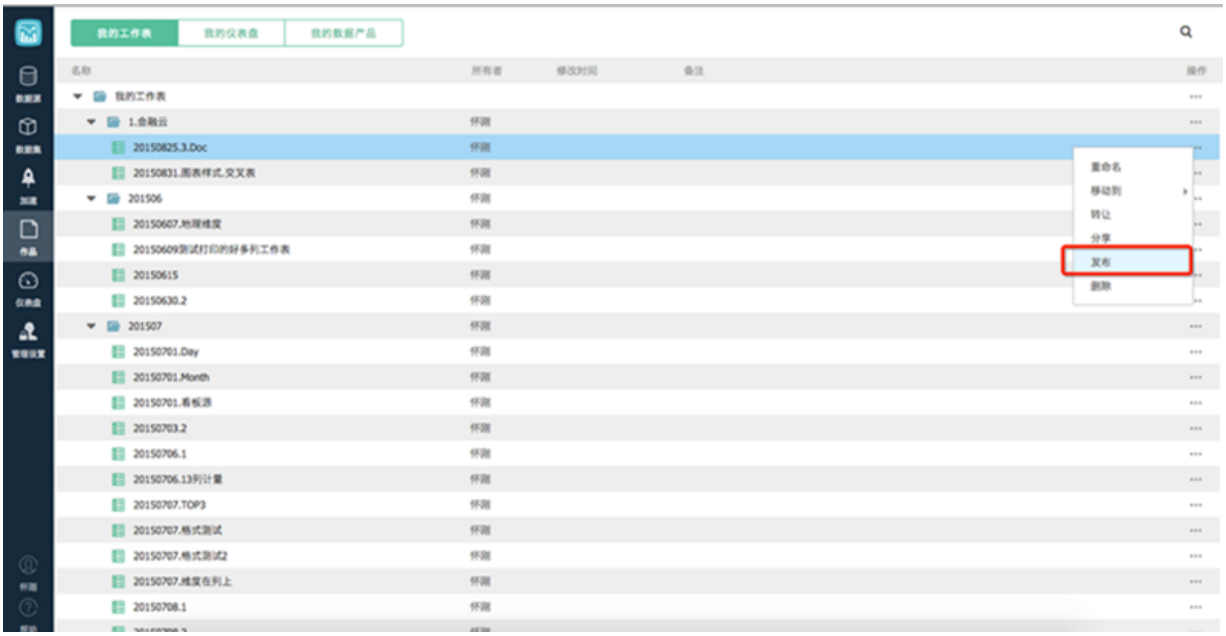


图 1838: 填写发布信息



33.5.12 文档管理

系统中的已经制作好的数据集、工作表、仪表盘和数据产品都是用户的文档。

用户可以对自己的文档进行查询和编辑。

在左侧导航栏中单击**作品**，即可进入文档管理页面：

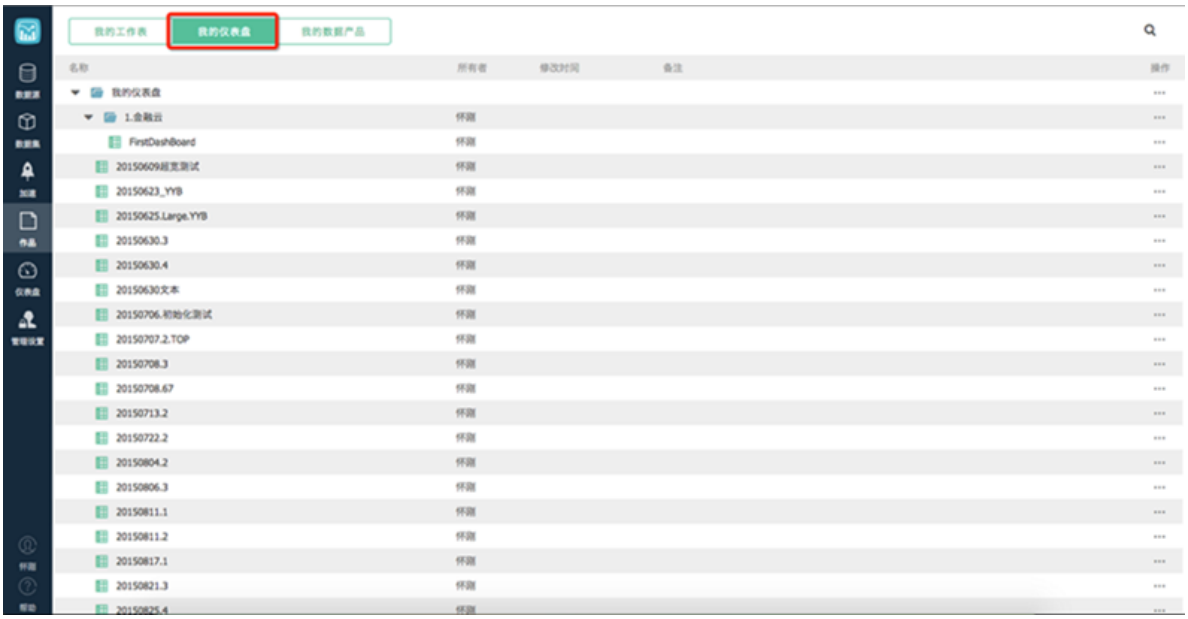
- 我的工作表 TAB 页，如图 *1839: 工作表 TAB 页*所示。

图 1839: 工作表 TAB 页



- 我的仪表盘 TAB 页，如图 *1840: 仪表盘 TAB 页*所示。

图 1840: 仪表盘 TAB 页



- 我的数据作品 TAB 页，如图 *1841: 数据作品 TAB 页*所示。

图 1841: 数据作品 TAB 页



针对自己的文档支持如下操作：

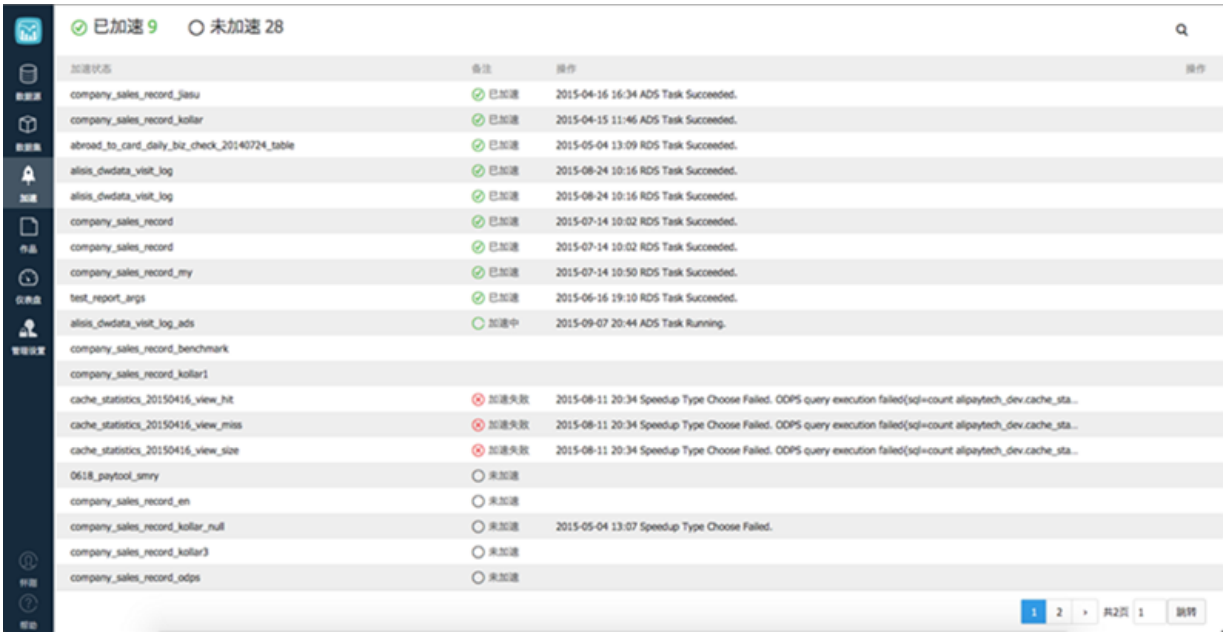
- 重命名：修改工作表的名称。
- 删除：删除改工作表。
- 移动：移动工作表至目标文档结构。
- 转让、分享、发布。

33.5.13 查询加速

在租户配置了加速资源的情况下，系统提供加速报表功能。

通过加速，报表编辑者和报表阅读者能够更快速地编辑和阅读报表，如图 *1842: 加速界面*所示。

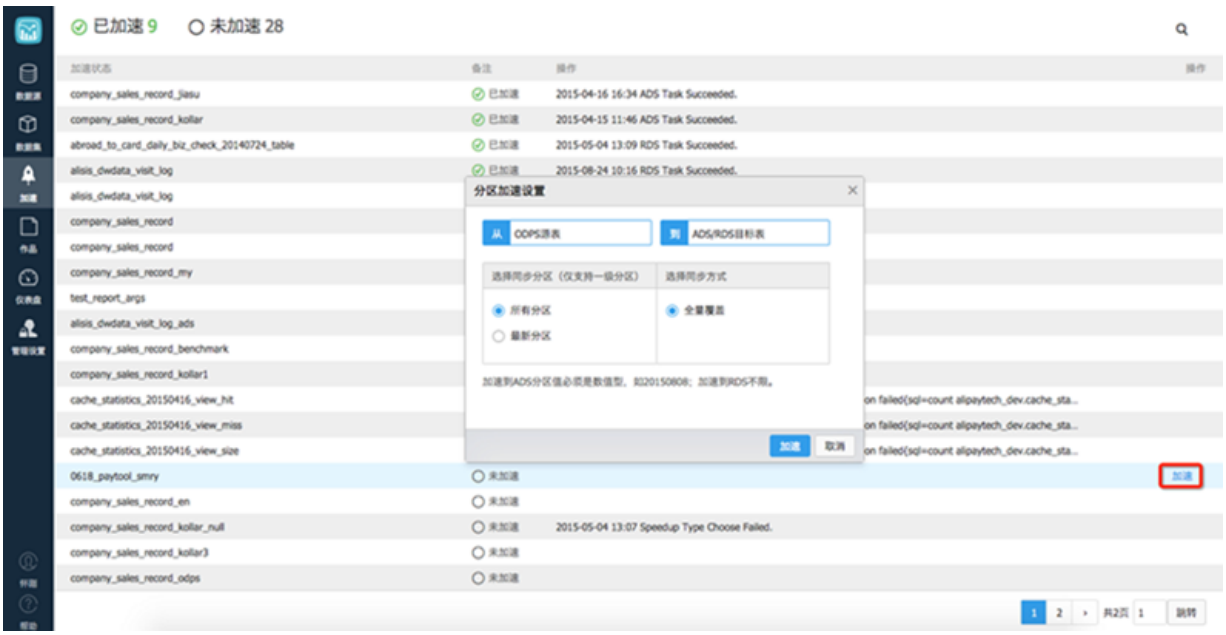
图 1842: 加速界面



加速操作是将数据从源同步到更快的数据存储上，以实现更快的查询速度。

如图 *1843: 设置加速信息*所示，选中工作表，单击**加速**，在弹出的对话框中设置加速信息。

图 1843: 设置加速信息



目前支持 RDS 和 AnalyticDB 两种加速环境的设置。如果配置了两种加速环境，系统根据数据量，自动选择目标环境。

目前加速支持以下 3 种模式：

- 源表取全表数据，覆盖全部目标表；
- 源表取最新分区数据，覆盖全部目标表；
- 源表取最新分区数据，增量添加到目标表数据。

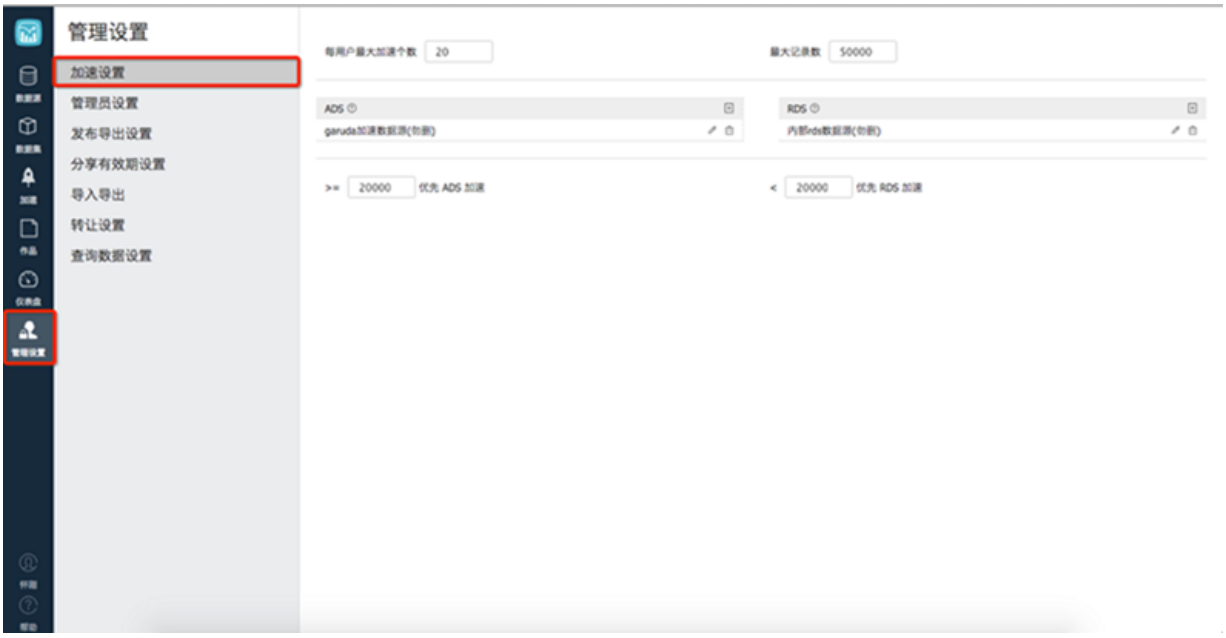
33.5.14 租户管理

租户管理的配置项是根据当前用户所在的租户和工作空间，进行对应设置，仅仅是租户管理员的用户才能使用该功能。

33.5.14.1 加速设置

如图 *1844: 加速设置入口*所示，进入加速设置。

图 1844: 加速设置入口



加速设置包含如下设置项：

- 每用户最大加速个数
- 加速支持的最大记录数
- Analytic DB 加速地址
- RDS 加速地址
- Analytic DB/RDS 加速数据量大小分界线

33.5.14.2 管理员设置

通过管理员设置功能，可以将某些用户设置为管理员，使其具备相应的操作权限，如图 *1845: 设置管理员*所示。

图 1845: 设置管理员



33.5.14.3 发布导出设置

在环境具备的情况下，可以对发布和导出功能进行配置，如图 *1846: 发布导出设置入口*所示。

图 1846: 发布导出设置入口



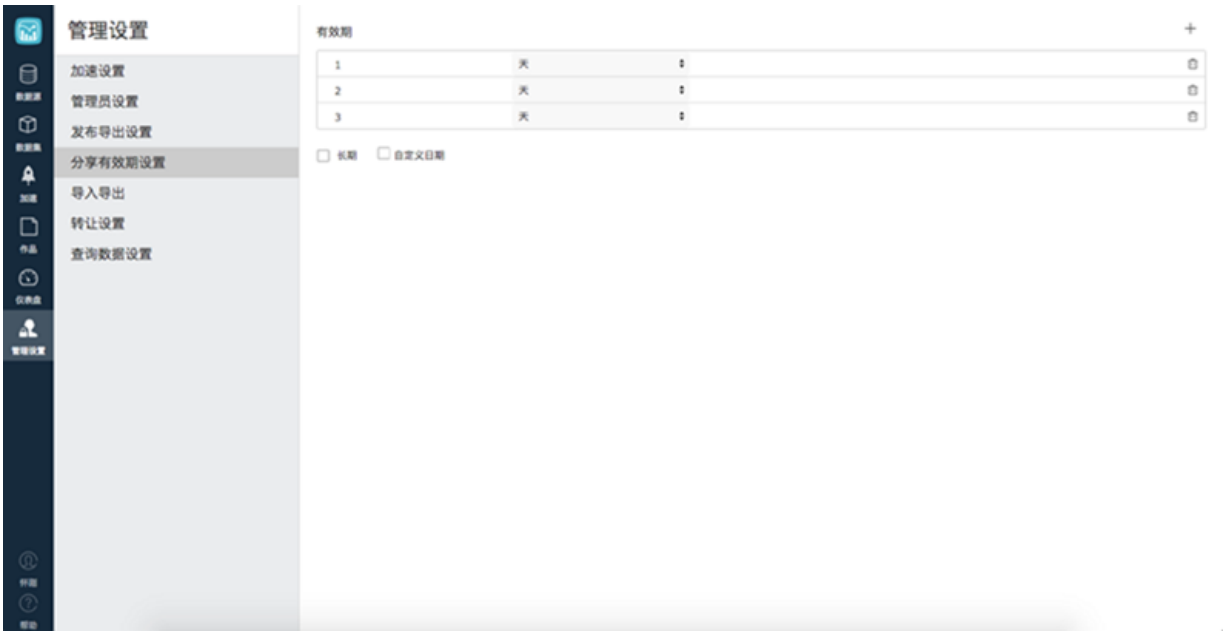
发布导出设置包含如下设置项：

- 工作表发布渠道
- 工作表权限申请渠道
- 仪表盘发布渠道
- 仪表盘权限申请渠道
- 数据集发布渠道
- 数据集权限申请渠道
- 数据产品发布渠道
- 数据产品权限申请渠道

33.5.14.4 分享有效期设置

在环境具备的情况下，可以对分享功能进行配置，如图 *1847:* 分享有效期设置入口所示。

图 1847: 分享有效期设置入口

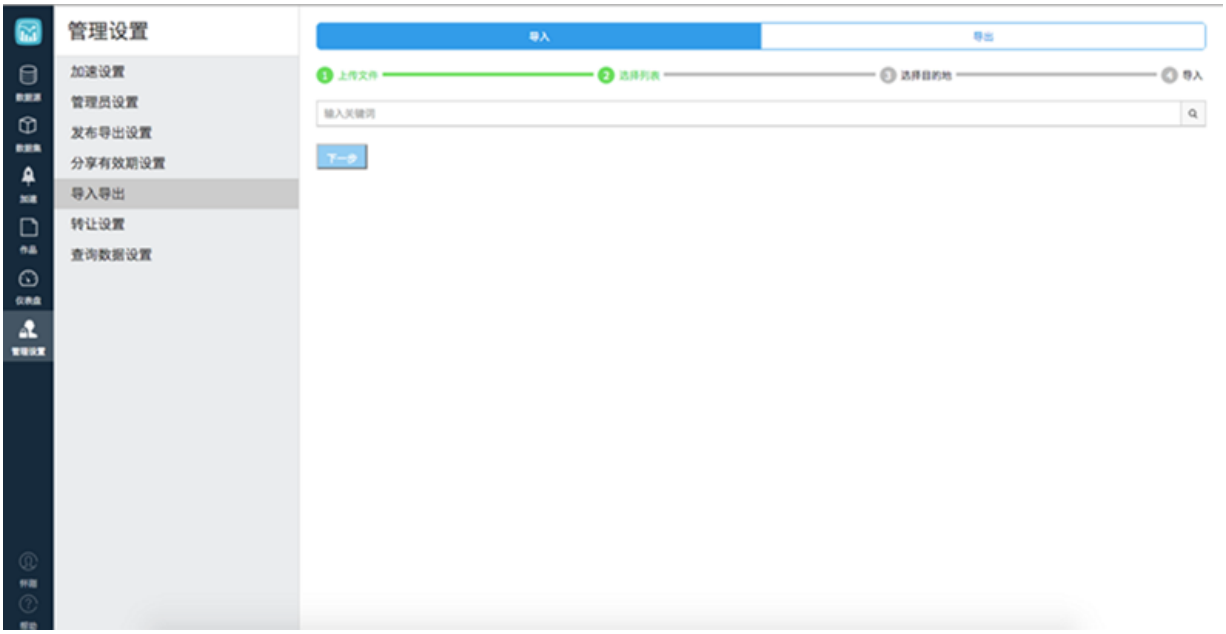


33.5.14.5 导入导出

数据集、工作表等模板可以被导出成模板文件，并且支持导入到目标系统中去（目标系统与源系统的物理表和字段保持一致）。

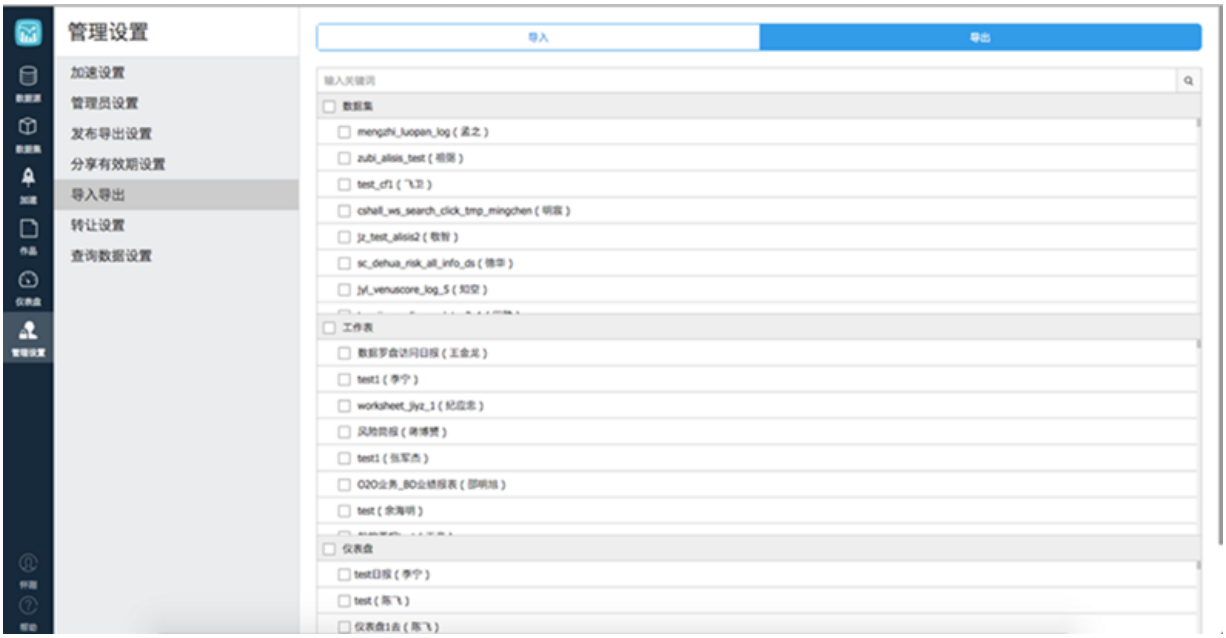
导入模板，如图 *1848:* 导入模板所示。

图 1848: 导入模板



导出模板，如图 *1849:* 导出模板所示。

图 1849: 导出模板



33.5.15 FAQ

33.5.15.1 基于金融云的登录认证体系

实际使用中，绝大多数场景使用金融云的账号进行登录。

在此情况下：

- 系统按照金融云账号体系进行登录认证，WEB 客户端只要按照金融云的账户登录，就能够使用报表服务的页面进行报表制作、能够进行报表查看。
- 系统报表（包括：数据集、工作表、仪表盘、数据产品）按照金融云账户进行赋权，只有被赋予权限的金融云账户才能访问相应的报表。

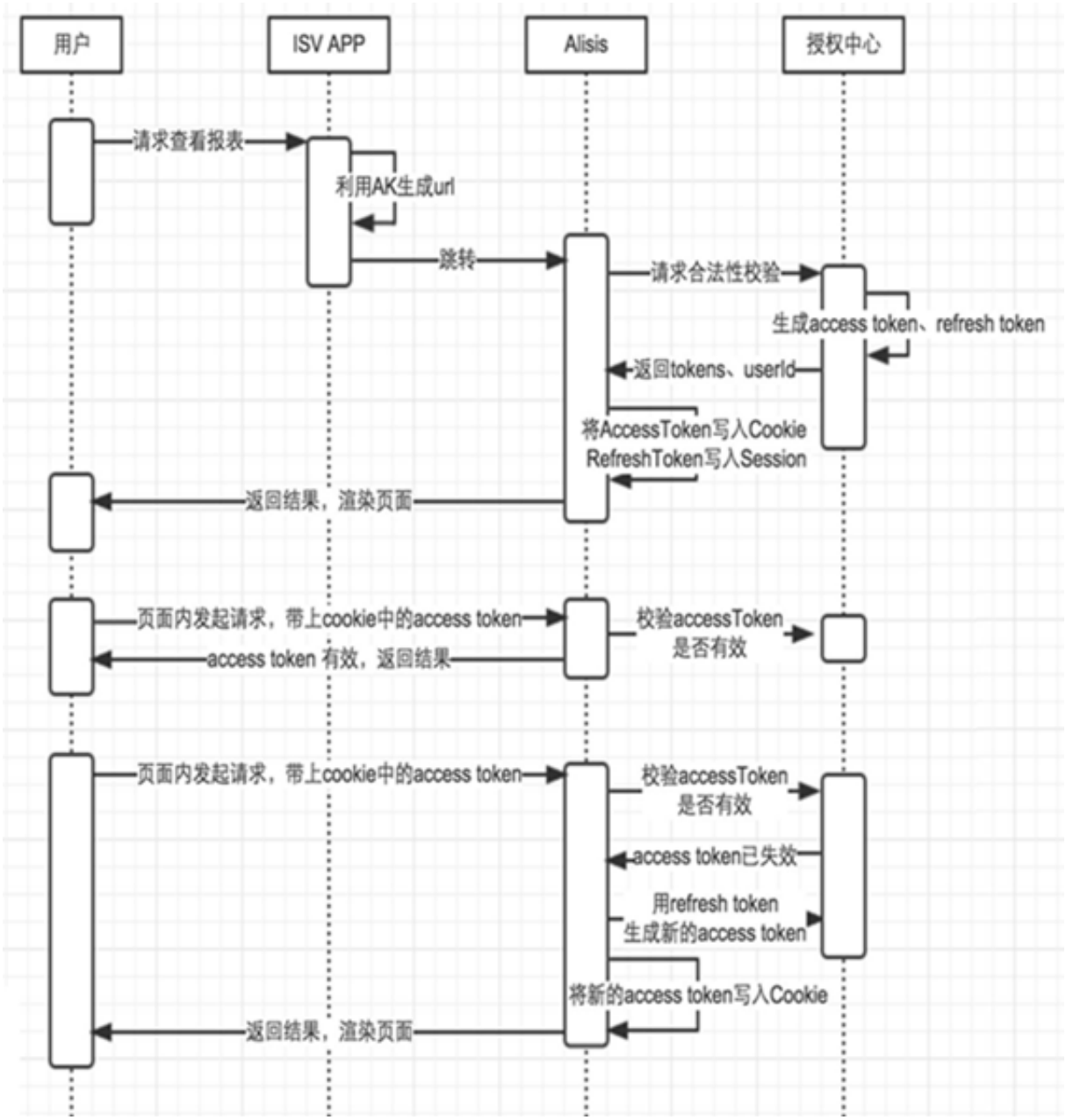
33.5.15.2 使用第三方账户体系登录

1. 业务方系统使用金融云账号体系生成用于免登的 AccessToken，业务方用户在浏览器中使用带有AccessToken 的 URL 访问系统。
2. 系统报表（包括：数据集、工作表、仪表盘、数据产品）按照金融云账户进行赋权，只有被赋予权限的金融云账户才能访问响应的报表。
3. 第三方应用：调用授权中心服务，根据 AccessKey 生成 AccessToken， AccessToken 的有效期可在后台管理进行调整。
4. 报表服务：
   - 验证 AccessToken 的有效性，返回详细的状态信息：过期、非法、正常（返回对应的用户信息、RefreshToken）；

- 如果 AccessToken 过期，根据 RefreshToken 获取新的 AccessToken。

系统架构如图 *1850: 系统架构图*所示。

图 1850: 系统架构图



开发者按照如下步骤集成：

- 加入Maven依赖，代码示例如下：

```
<dependency>
<groupId>com.alipay.auth_center</groupId>
<artifactId>auth_center-common-service-facade</artifactId>
<version>1.0.20150602</version>
```

```
</dependency>
```

- 实现 AuthServiceClient 接口，示例代码如下：

```
public class AuthServiceClientImpl implements AuthServiceClient {
private static final Logger LOGGER = LoggerFactory.getLogger(AuthServiceClientImpl.class);
/** token服务*/
private CertificateService certificateService;
@Override
public String getAccessToken(String aliyunId, String authKeyId) {
LOGGER.warn("开始获取采云间 AccessToken 信息，aliyunId=" + aliyunId);
String authToken = "";
try {
ServiceResponse<String> resp = certificateService.issueToken(aliyunId, authKeyId);
if (resp.isSuccess()) {
authToken = resp.getContent();
} else {
LOGGER.warn("获取采云间 AccessToken 信息失败，message=" + resp.getMessage());
}
} catch (Exception e) {
LOGGER.error("获取 AccessToken 异常", e);
}
LOGGER.warn("获取采云间 AccessToken 信息成功，authToken=" + authToken);
return authToken;
}
/**
* Setter method for property <tt>certificateService</tt>.
*
* @param certificateService value to be assigned to property certificateService
*/
public void setCertificateService(CertificateService certificateService) {
this.certificateService = certificateService;
}
}
```

说明:

- AccessToken 有时间和 session 限制：用户每次刷新页面，业务系统必须要更新 URL 中的 AccessToken。
- AccessToken 如果在用户A的浏览器中访问过，就会和当前浏览器的 session 绑定，那么在其他用户电脑上、其他浏览器上就不能继续使用，系统会提示 Session Timeout。

通常情况下，在仪表盘中的图表控件接受过滤器控件、日期控件、查询条件控件的控制。但是如果需要传入参数联动图表控件，则需要使用如下的集成方式。

在报表阅读者界面打开报表，选择过滤条件，获取浏览器地址栏中的地址。

URL 地址为：

```
https://das4int2.alipay.com/shulaibao/preview.htm?pageId=92d0098c-4204-4519-
a943-939be08250ed#f3d57cd5- e39f-4132-9c49- 4eb2e235f648*=*{"string":
{"isCondition":false,"condition":{"type":"or","item":[{"type":"=","left":{"pathId":"schema072
774.de43228f-dead-4ada-9f2dee73abedf7aa_ company_sales_record
```

company_sales_record_my_order_level].[company_sales_record_my_order _level].[company_sales_record_my_order_level]"}}]},"enumerate":{"type":"in","right":["低级 "],"left":{"pathId":"schema072774.de43228f-dead-4ada-9f2dee73abedf7aa_ company_sales_record_my.[company_sales_record_my_order_level].[company_sales_record_my_order _level].[company_sales_record_my_order_level]"}}}}

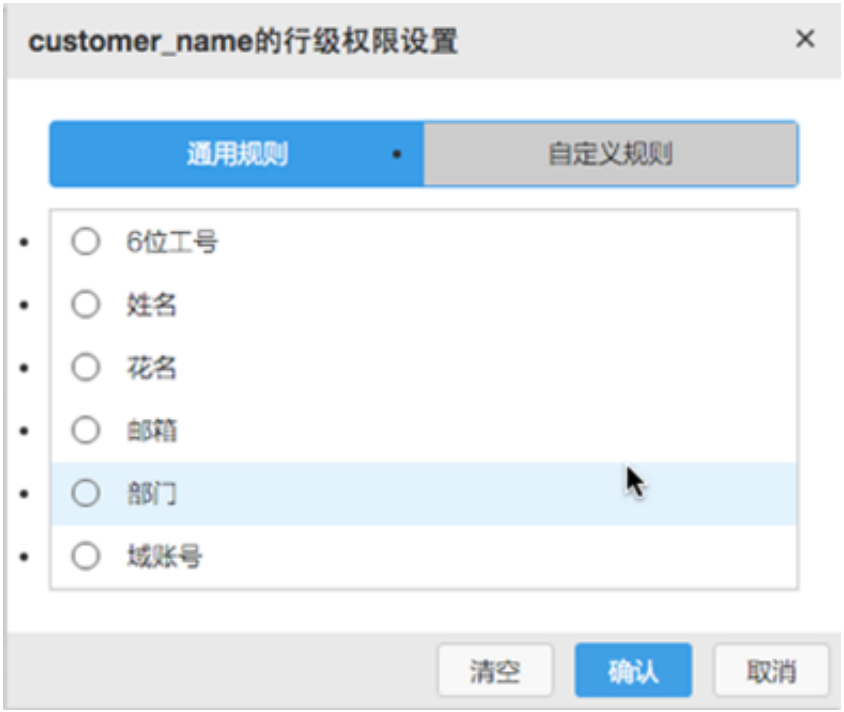
其中，**低级**就是过滤器的输入值，可以将该值改为**高级**。

URL 为：

https://das4int2.alipay.com/shulaibao/preview.htm?pageId=92d0098c-4204-4519-a943-939be08250ed#f3d57cd5-e39f-4132-9c49-4eb2e235f648*=*{"string":{"isCondition":false,"condition":{"type":"or","item":[{"type":"=","left":{"pathId":"schema072774.de43228f-dead-4ada-9f2dee73abedf7aa_company_sales_record_my.[company_sales_record_my_order_level].[company_sales_record_my_order_level].[company_sales_record_my_order_level]"}}]},"enumerate":{"type":"in","right":["高级"],"left":{"pathId":"schema072774.de43228f-dead-4ada-9f2dee73abedf7aa_company_sales_record_my.[company_sales_record_my_order_level].[company_sales_record_my_order_level].[company_sales_record_my_order_level]"}}}}

在使用金融云账户登录情况下的行级权限设置，如图 *1851: 行级设置*所示的配置。

图 1851: 行级设置



其中，以下四个参数的设置有效：

- 姓名：对应到金融云的登录账号中的**姓名**。

- 花名：对应到金融云的登录账号中的**昵称**。
- 邮箱：对应到金融云的登录账号中的**邮箱**。
- 域账号：对应到金融云的登录账号中的**账户名**，即登录系统的账号名称。

在行级权限设置的情况下，系统会根据报表阅读者的上述信息，对数据进行额外的过滤，并将满足条件的数据返回给报表页面进行呈现。

34 机器学习PAI

34.1 产品概述

概述

阿里云机器学习平台是构建在阿里云 MaxCompute（原ODPS）计算平台之上，集数据处理、建模、离线预测、在线预测为一体的机器学习平台。阿里云机器学习 PAI 封装了阿里巴巴集团内成熟的算法，向机器学习用户提供了更简易的操作体验。真正做到，让人工智能触手可及。

优势

- 算法成熟度高，全部算法脱胎于阿里集团内部业务的锤炼，经历过 PB 级别业务的考验。
- 算法种类全，提供了包括数据预处理、特征工程、算法、预测和评估这一套数据挖掘全链路组件。
- 操作简易，通过将算法封装为算法组件，让操作者通过拖拉拽的方式可以快速上手实验，降低算法的使用门槛。

基础架构



阿里云机器学习平台的基础架构分为四层，用户通过阿里云机器学习平台调用下一层的模型和算法，然后系统将相应的算法转换成对应的计算类型，比如说对于两个表的 join 操作，会自动生成 SQL 的 workflow 传到底层 MaxCompute 进行运算和操作。所有算法都是以 plugin 的形式存放在底层的计算引擎当中，用户在使用算法的时候只关心算法的调用即可，真正实现了算法和计算引擎的解耦。

34.2 基本概念

实验

用户自己搭建的一系列数据挖掘流程。

MaxCompute（**原**ODPS）

开放数据处理服务由阿里云自主研发的计算引擎，提供针对TB/PB级数据、实时性要求不高的分布式处理能力，应用于数据分析、挖掘、商业智能等领域。

项目（Project）

项目（也称项目空间）是 MaxCompute（ODPS）最基本的组织对象。其他对象，例如表（Table）和实例（Instance）等都归属于某个项目。

MaxCompute 源表与 MaxCompute 目标表（Table）

表（Table）是 MaxCompute（原ODPS）中数据存储对象。与常见的关系型数据类似，MaxCompute（原ODPS）中的表逻辑上也是二维结构。源表指一个算法节点的输入，目标表指算法节点的输出。

组件（Nodes）

组件是用户可以在 PAI 平台上调用执行的最小操作单元，例如数据导入导出、数据处理、数据分析、模型训练或者预测。

34.3 快速开始

一个完整的建模步骤包括以下6个步骤：

- step1：数据准备。
- step2：数据预处理。
- step3：数据可视化。
- step4：算法建模。
- step5：多模型评估。
- step6：数据存储。

下面章节在 PAI 上给您做一个入门级的示例。

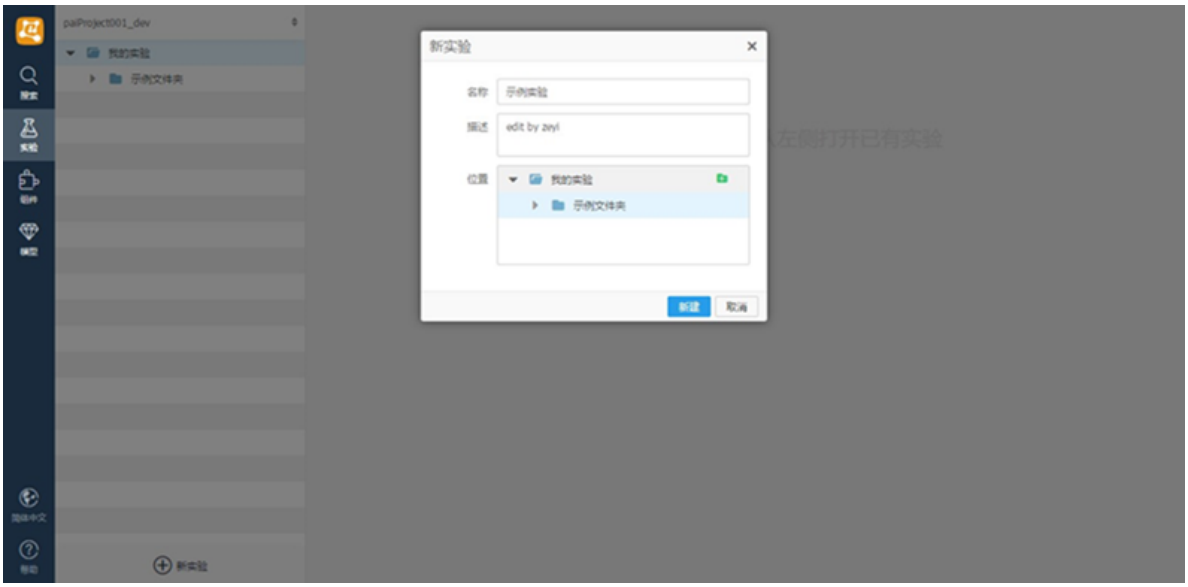
说明： 以下章节中的步骤默认用户已拥有自己的 MaxCompute projet 并将表数据传入对应的 project 中。数据下载地址：*http://archive.ics.uci.edu/ml/datasets/Heart+Disease*。

34.3.1 数据准备

操作步骤

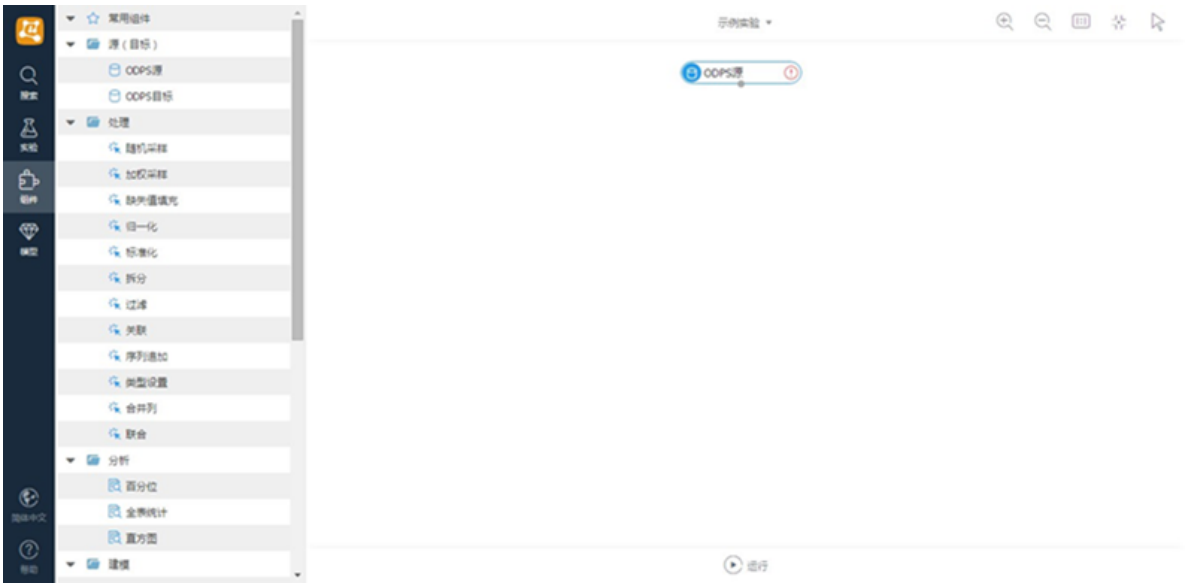
1. 在**实验**页面，右击**我的实验** > **新建空白实验**，在弹出的窗口中输入实验名和实验描述。

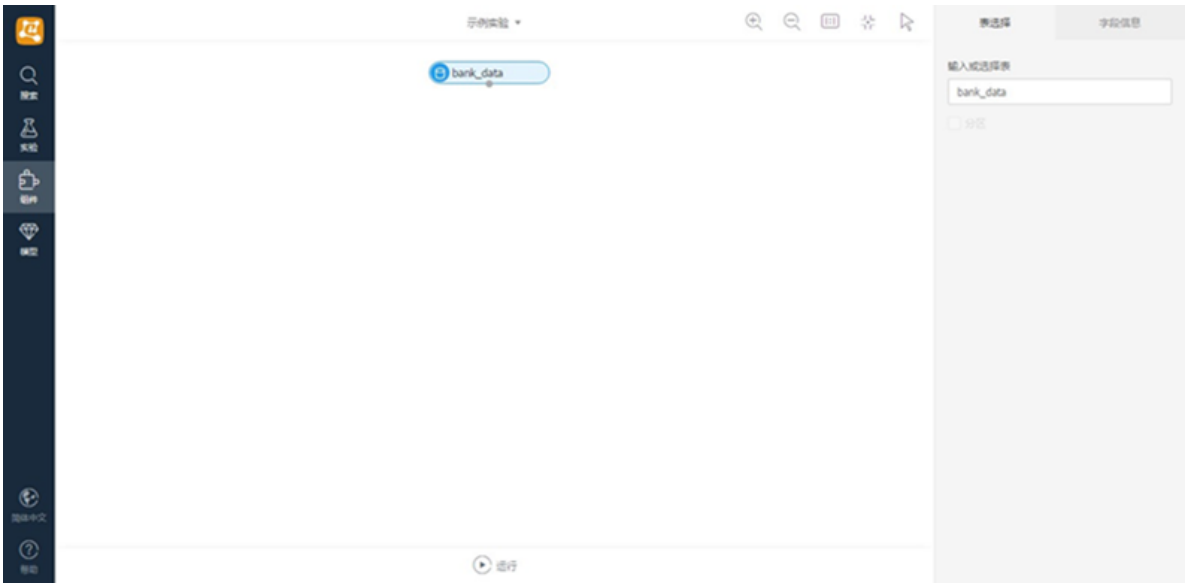
图 1852: 新实验



2. 切换到**组件**栏，向画布中拖入 ODPS 源，点击 ODPS 源，在右侧表选择栏填入 ODPS。

图 1853: 拖入ODPS源





3. 切换到**字段信息**栏，如图 *1854:* 字段信息所示，可以查看输入表的字段名、数据类型和前100行数据的数值分布。

图 1854: 字段信息


表选择 字段信息

源表字段信息

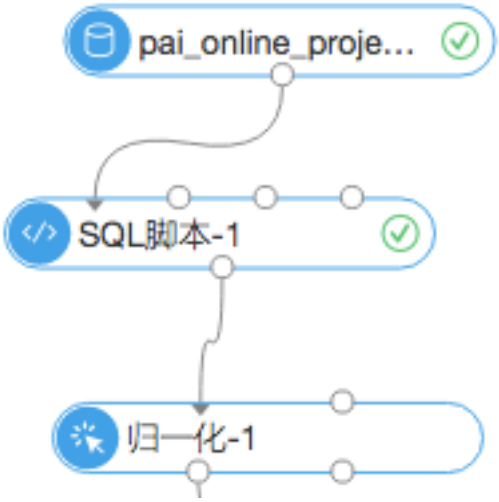
字段	类型	前 100 条记录范围
age	STRING	37.0,41.0,63.0,67.0
sex	STRING	fem,male
cp	STRING	abnang,angina,asympt,nc
trestbps	STRING	120.0,130.0,145.0,160.0
chol	STRING	204.0,229.0,233.0,250.0,2
fbs	STRING	fal,true
restecg	STRING	hyp,norm
thalach	STRING	108.0,129.0,150.0,172.0,1
exang	STRING	fal,true
oldpeak	STRING	1.4,1.5,2.3,2.6,3.5
slop	STRING	down,flat,up
ca	STRING	0.0,2.0,3.0
thal	STRING	fix,norm,rev
status	STRING	buff,sick
style	STRING	H,S1,S2

34.3.2 数据预处理

操作步骤

1. 从**处理**中拖入**SQL、归一化**组件到画布。

   图 1855: 拖入组件



2. 右击**SQL 组件**，进行特征转换，并且将字段类型转换为**double**。

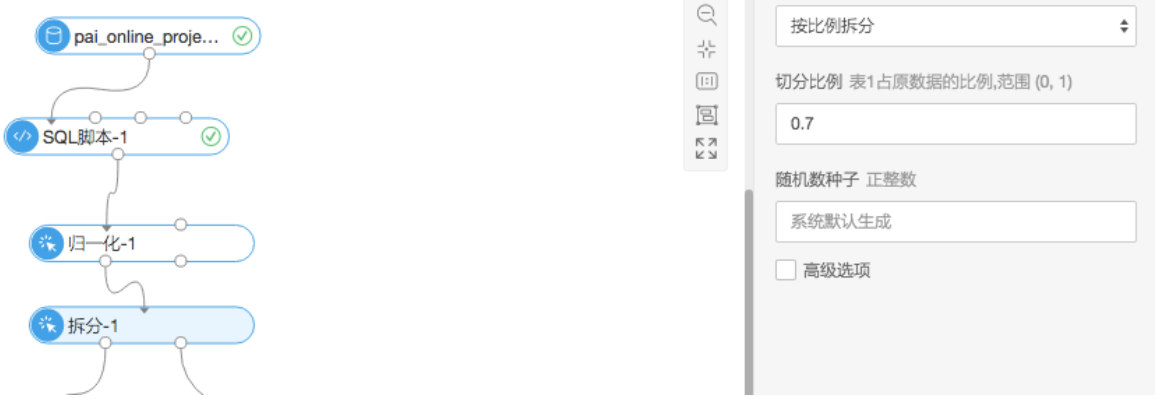
   图 1856: 特征转换

```
select age,
(case sex when 'male' then 1 else 0 end) as sex,
(case cp when 'angina' then 0  when 'notang' then 1 else 2 end) as cp,
trestbps,
chol,
(case fbs when 'true' then 1 else 0 end) as fbs,
(case restecg when 'norm' then 0  when 'abn' then 1 else 2 end) as restecg,
thalach,
(case exang when 'true' then 1 else 0 end) as exang,
oldpeak,
(case slop when 'up' then 0  when 'flat' then 1 else 2 end) as slop,
ca,
(case thal when 'norm' then 0  when 'fix' then 1 else 2 end) as thal,
(case status  when 'sick' then 1 else 0 end) as ifHealth
from  ${t1};
```

说明: 回归组件选择全部字段执行。

3. 将**拆分**组件拖到画布。

图 1857: 拖入拆分组件



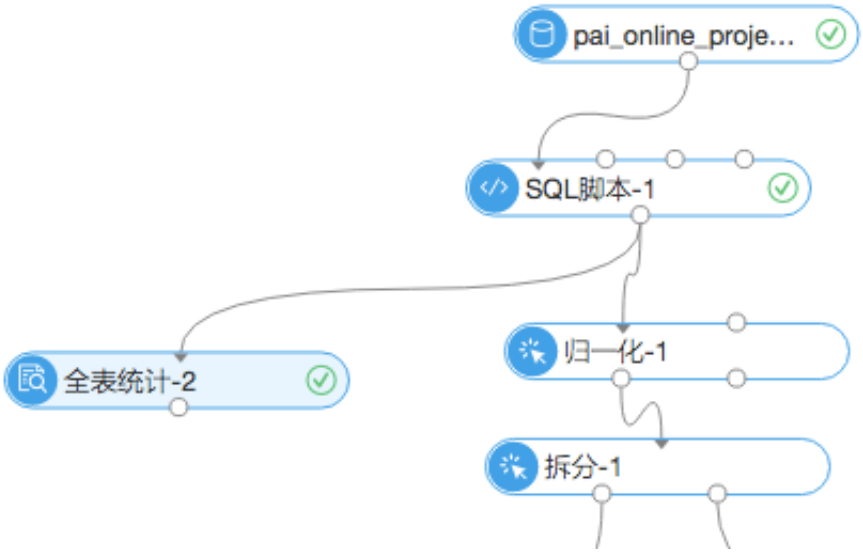
说明: 此步骤的目的是将数据拆分成两份，70%作为模型训练集，30%作为模型预测集。

34.3.3 数据可视化

操作步骤

1. 向画布拖入**全表统计**组件，连线并点击画布底部的**运行**按钮。

图 1858: 拖入全表统计组件



2. 待实验运行结束后，右击**全表统计**，点击**查看分析报告**，可看到数据的全表统计信息。

图 1859: 分析报告

数据探查 - pai_temp_39578_610010_1 - (仅显示前一百条)

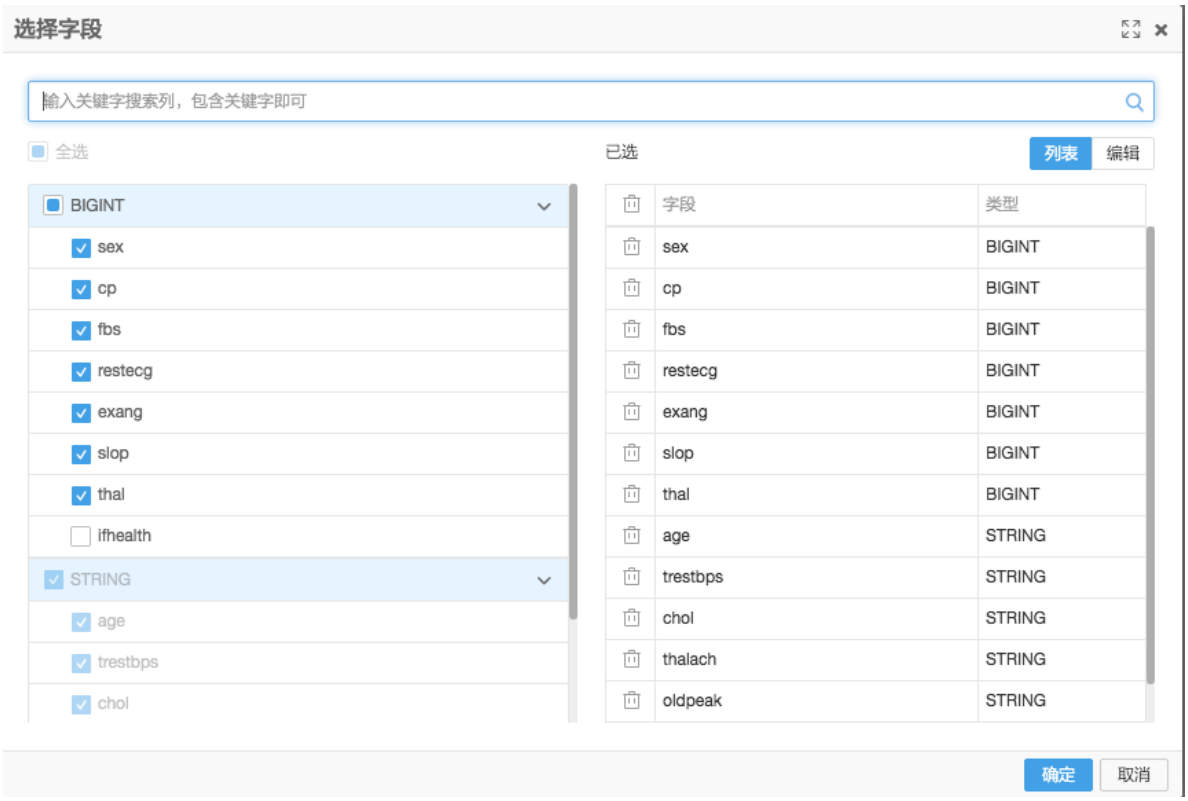
序号	colname	datatype	totalcount	count	missingcount	nancount	positiveinfinitycount	negativeinfinitycount	min	max	m
1	age	string	303	303	0	0	0	0	-	-	-
2	ca	string	303	303	0	0	0	0	-	-	-
3	chol	string	303	303	0	0	0	0	-	-	-
4	cp	bigint	303	303	0	0	0	0	0	2	1.
5	exang	bigint	303	303	0	0	0	0	0	1	0.
6	fbs	bigint	303	303	0	0	0	0	0	1	0.
7	ifhealth	bigint	303	303	0	0	0	0	0	1	0.
8	oldpeak	string	303	303	0	0	0	0	-	-	-
9	restecg	bigint	303	303	0	0	0	0	0	2	0.
10	sex	bigint	303	303	0	0	0	0	0	1	0.
11	slop	bigint	303	303	0	0	0	0	0	2	0.
12	thal	bigint	303	303	0	0	0	0	0	2	0.
13	thalach	string	303	303	0	0	0	0	-	-	-
14	trestbps	string	303	303	0	0	0	0	-	-	-

34.3.4 算法建模

操作步骤

1. 向画布拖入两个**逻辑回归**组件。
2. 组件1*训练特征列*选择13个 feature 列，参数设置采用系统默认参数。

图 1860: 选择字段

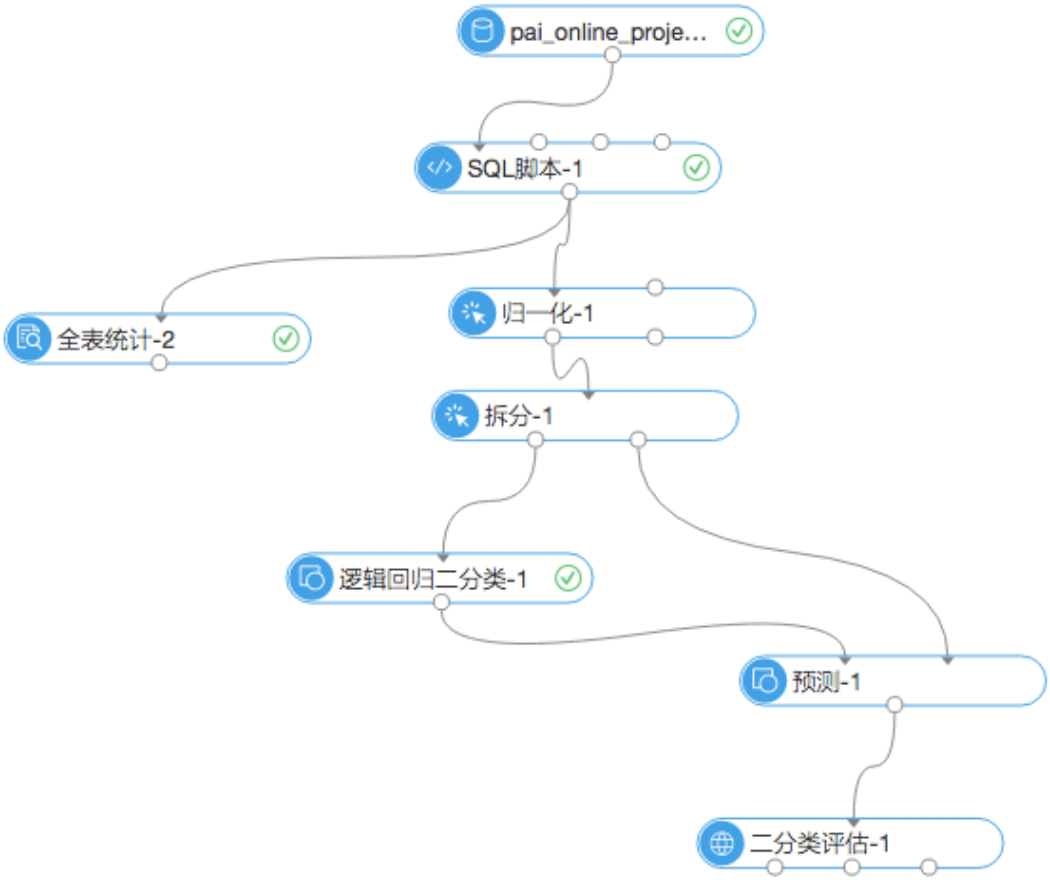


34.3.5 模型评估

操作步骤

1. 单击**机器学习** > **预测**，向画布拖入**预测**组件，并连接对应的组件流和数据流。

   图 1861: 拖入预测组件



2. 单击**机器学习** > **评估** > **二分类评估**，向画布拖入**二分类评估**组件，并连接对应的组件流和数据流。
3. 单击**运行**，得到不同参数下训练的 LR 模型的 ROC 曲线。

   图 1862: 获得ROC曲线



34.4 使用教程

34.4.1 源/目标

读数据表

读取 MaxCompute 的表数据组件，默认读取本工程下的数据；若读取其他工程的表数据且拥有该 project 的操作权限），只需在表名前添加工程名，格式：**工程名.表名**，如：tianchi_project.weibo_data 当输入表后，会自动读取表的结构数据，可点击**字段信息**查看。本组件不支持视图。

若输入表是分区表，后台会自动勾选分区框，用户可选择或输入分区参数，目前仅支持输入单个分区。不勾选分区框或勾选后不输入分区参数均默认为输入全表若输入表是非分区表，分区框不可勾选读 ODPS 表的输入框。

图 1863: 读数据表



写数据表

写入数据表的数据组件，同样支持写入其他工程的表数据。可以设置分区，如果需要写入分区表，需要先在 ODPS CONSOLE 端创建好这个表的分区。可以设置表的生命周期 lifecycle，以天为单位。

图 1864: 写 MaxCompute 表



34.4.2 数据预处理

34.4.2.1 采样与过滤

34.4.2.1.1 加权采样

以加权方式生成采样数据；权重列必须为 double 或 int 类型，按照该列的 value 大小采样；如 col 的值是 1.2 和 1.0；则 value=1.2 所属样本的被采样的概率就大一些。

参数设置



- 可以选择放回采样或者不放回采样，默认为不放回，勾选后变为放回。
- 可手动输入采样个数，默认是10000个点。

> **说明:** 不放回时采样个数不能多于数据条数。

- 下拉框选择加权列，加权列支持 double 型和 bigint 型。

PAI 命令

```
PAI –name WeightedSample –project algo_public -DprobCol="previous" -DsampleSize="500" -DoutputTableName="test2" -DinputPartitions="pt=20150501" -DinputTableName="bank_data_partition";
```

- name：组件名字；
- project：project 名字，用于指定算法所在空间。系统默认是 algo_public，用户自己更改后系统会报错；
- replace：是否放回。其中 true 为放回，false 为不放回；
- probCol：选择的要加权的列，每个值代表所在 record 出现的权重，不需要归一化；
- sampleSize：采样个数。

> **说明:** 不放回时采样个数不能多于数据条数。

- outputTableNames：输出表的名字，多张表用逗号分隔；

- inputPartitions：（可选）训练输入表分区。输入表对应的输入分区，选中全表则为 None；
- inputTableNmae：输入表的名字；
- replace：（可选）是否放回。其中 true 为放回，不放回时无此参数。

34.4.2.1.2 随机采样

- 以随机方式生成采样数据，每次采样是各自独立的。可以按照个数或者比例进行采样，也可以选择是否放回。

图 1865: 随机采样



PAI 命令

```
Pai –name sample –project algo_public -DinputTableName=wbpc -DoutputTableName=wpbc_sample -Dratio=0.3;
```

- **算法参数**

参数名称	参数描述	参数值可选项	默认值
inputTableName	必选，输入表的表名	-	-
inputTablePartitions	可选，输入表中指定哪些分区参与训练，格式为：partition_name=value。如果是多级，格式	-	n

参数名称	参数描述	参数值可选项	默认值
	为name1=value1/name2=value2；如果指定多个分区，中间用','分开		
ratio	必选，指定采样比例	(0,1)	-
outputTableName	必选，输出结果表	-	-
outputTablePartition	可选，输出结果表partition	-	输出表为非partition表
lifecycle	可选，指定输出表生命周期	正整数，[1,3650]	输出表没有生命周期

34.4.2.1.3 过滤与映射

对数据按照过滤表达式进行筛选；可以重命名字段名。

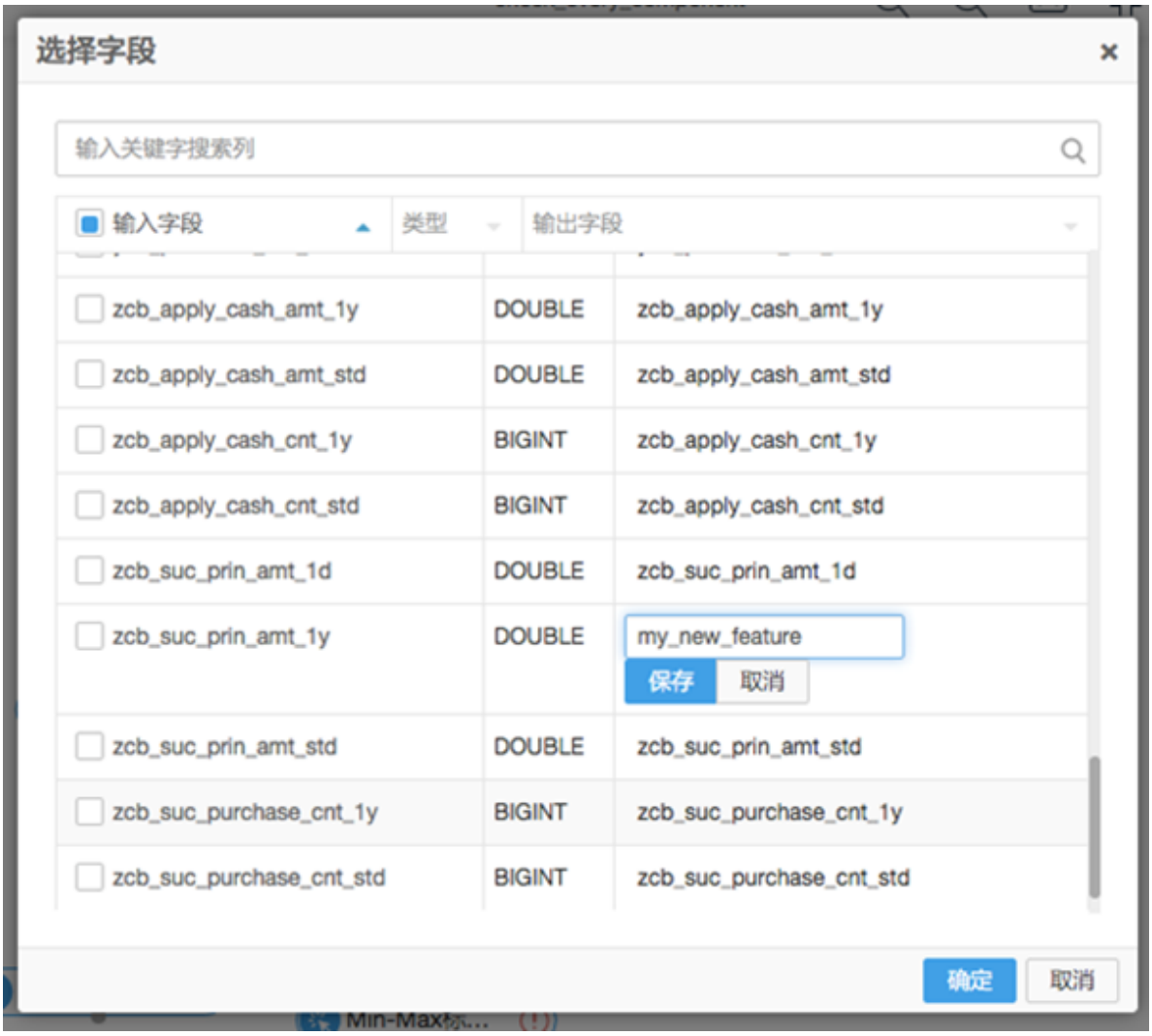
参数设置

- 通过 where 条件实现数据过滤，与 SQL 类似。



筛选条件：目前操作符支持等号（=），不等号（!=），大于号（>），小于号（<），大于等于号（>=）和小于等于号（<=），like，rlike。

- 重命名字段



PAI 命令

```
PAI –name Filter –project algo_public -DoutTableName="test_9" -DinputPartitions="pt=20150501" -DinputTableName="bank_data_partition" -Dfilter="age>=40";
```

- name:：组件名字；
- project： project 名字，用于指定算法所在空间。系统默认是 algo_public，用户自己更改后系统会报错；
- outTableName：输出表的名字；
- inputPartitions：（可选）训练输入表分区。输入表对应的输入分区，选中全表则为None；
- inputTableName：输入表的名字；
- filter：where 筛选条件，目前操作符支持“=” ，“！=” ，“ >” ，“ <” ， “>=” ， “<=”，“like”和“rlike”。

34.4.2.1.4 分层采样

先将总体的单位按某种特征分为若干次级总体（层），然后再从每一层内进行单纯随机抽样，组成一个样本的统计学计算方法。

图 1866: 分层采样



参数设置

- 字段设置

  分组列-必选项，按此列划分层次。

- 参数设置

  采样比例/样本数：必选项，小于1表示每层采样的比例，大于1表示每层采样的个数。

  特殊采样配置：可选项，提供不同层采集不同数量的功能。

  随机种子数：可选项，1234567。

PAI 命令

```
Pai –name sample –project algo_public -DinputTableName=wbpc -DoutputTableName=wpbc_sample -DstrataColName="label" -DsampleSize="A:200,B:300,C:500" -DrandomSeed=1007 -Dlifecycle=30
```

算法参数

参数名称	参数描述	参数值可选项	默认值
inputTableName	必选，输入表的表名	-	-
inputTablePartitions	可选，输入表中指定哪些分区参与训练，格式为：partition_name=value。如果是多级，格式为name1=value1/	-	输入表的所有partition

参数名称	参数描述	参数值可选项	默认值
	name2=value2；如果指定多个分区，中间用','分开		
strataColName	必选，指定分层列	-	-
outputTableName	必选，输出结果表	-	-
sampleSize	可选，整数时表示每个stratum的采样个数。字符串时，格式为strata0:n0,strata1:n1....表示每个startum分别配置采样个数	-	
sampleRatio	可选，数字时：范围[0,1]表示每个startum的采样比例。字符串时：格式为strata0：r0，strata1：r1...表示每个stratum分别配置采样比例	-	-
randomSeed	可选，随机种子数	-	0
lifecycle	可选，指定输出表生命周期	正整数，[1,3650]	输出表没有生命周期
coreNum	可选，核心数，默认自动分配	-	-
memSizePerCore	可选，每个核心的内存，默认自动分配	-	-

34.4.2.2 数据合并

34.4.2.2.1 合并列

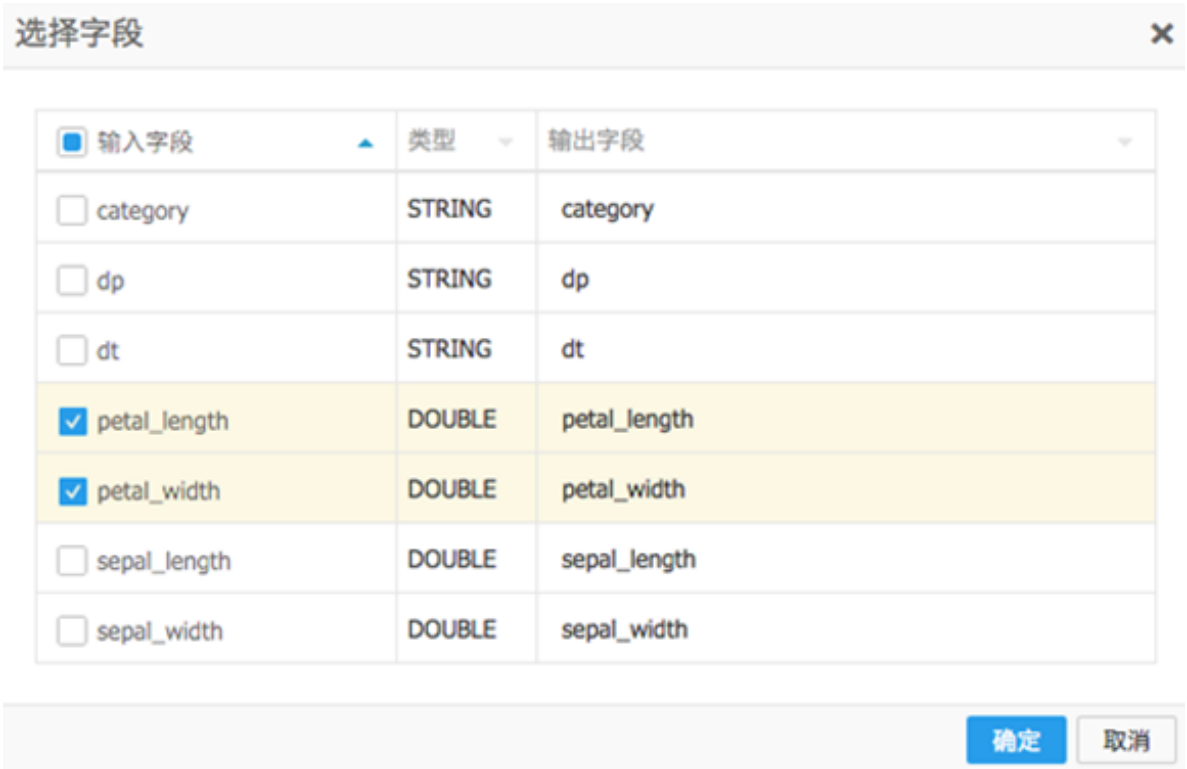
将两张表的数据按列合并，需要表的行数保持一致。

参数设置



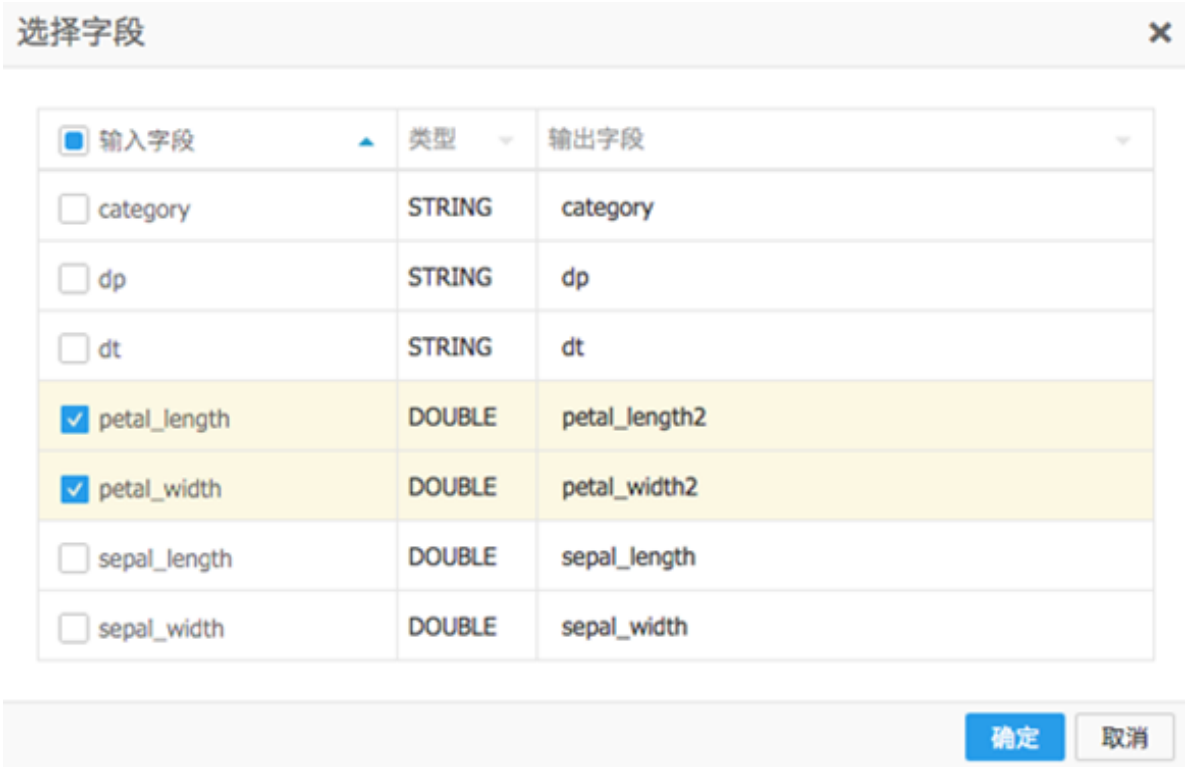
1. 左表选择输入列，如图 *1867: 左表选择输入列*所示。

图 1867: 左表选择输入列



2. 右表选择输入列，如图 *1868: 右表选择输入列*所示。

图 1868: 右表选择输入列



进行合并列时：

- 选择的两张表行数需保持一致；
- 左表与右表选择的输出列名不能重复；
- 选择输出字段列时可手动更改输出字段名；
- 若左表右表均不选择输出列，则默认输出全表。此时勾选**是否自动重命名输出列**，会将重复列重命名后输出。

PAI 命令

```
PAI –name AppendColumns –project algo_public -DoutputTableColNames="petal_length,petal_width,petal_length2,petal_width2" -DautoRenameCol="false" -DoutputTableName="pai_temp_770_6840_1" -DinputTableNames="iris_twopartition,iris_twopartition" -DinputPartitionsInfoList="dt=20150125/dp=20150124;dt=20150124/dp=20150123" -DselectedColNamesList="petal_length,petal_width;sepal_length,sepal_width";
```

- name：组件名字；
- project：project 名字，用于指定算法所在空间。系统默认是 algo_public，用户自己更改后系统会报错；
- outputTableColNames：新表中各列的列名。逗号分隔，如果 autoRenameCol 为 true，则此参数无效；

- autoRenameCol：（可选）输出表是否自动重命名列，true 为重命名，false 不进行重命名，默认 false；
- outputTableName：输出的表名；
- inputTableNames：输入的表名，如果有多个，逗号分隔；
- inputPartitionsInfoList：（可选）输入表对应选择的 partition 列表，同一个表的各 partition 按逗号分隔，不同表的 partition 分号分隔；
- selectedColNamesList
  ：选择输入的列名，同张表之间列名用逗号分隔，不同表之间用分号分隔。

34.4.2.2.2 拆分

对输入表或分区进行按比例拆分，分别从两个输出桩写出两表。

算法组件

图 1869: 算法组件



- 拆分组件，对应两个输出桩；
- 如上图所示，配置0.6，则左边的输出桩对应60%的数据，右边对应40%的数据。

PAI 命令

```
pai -name split -project algo_public \ -DinputTableName=wbpc \ -Doutput1TableName=wpbc_split1 \ -Doutput2TableName=wpbc_split2 \ -Dfraction=0.25;
```

表 132: 参数设置

参数名称	参数描述	参数值可选项	默认值
inputTableName	必选，输入表的表名	-	-
inputTablePartitions	可选，输入表中指定哪些分区参与训练，格式为: partition_name=value。如果是多级，格式为 name1=value1/ name2=value2；如果指定多个分区，中间用’,’分开	-	输入表的所有 partition
output1TableName	必选，输出结果表1	-	-
output1TablePartition	可选，输出结果表1分区名	-	输出表1为非分区表
output2TableName	必选，输出结果表2	-	-
output2TablePartition	可选，输出结果表2分区名	-	输出表2为非分区表
fraction	必选，切分至输出表1的数据比例	（0,1）	-
lifecycle	可选，指定输出表生命周期	正整数，[1,3650]	输出表没有生命周期

34.4.2.2.3 UNION

将两张表的数据按行合并，左表及右表选择输出的字段个数以及类型应保持一致，如图 *1870: 参数设置*所示，整合了 union 和 union all 的功能。

图 1870: 参数设置

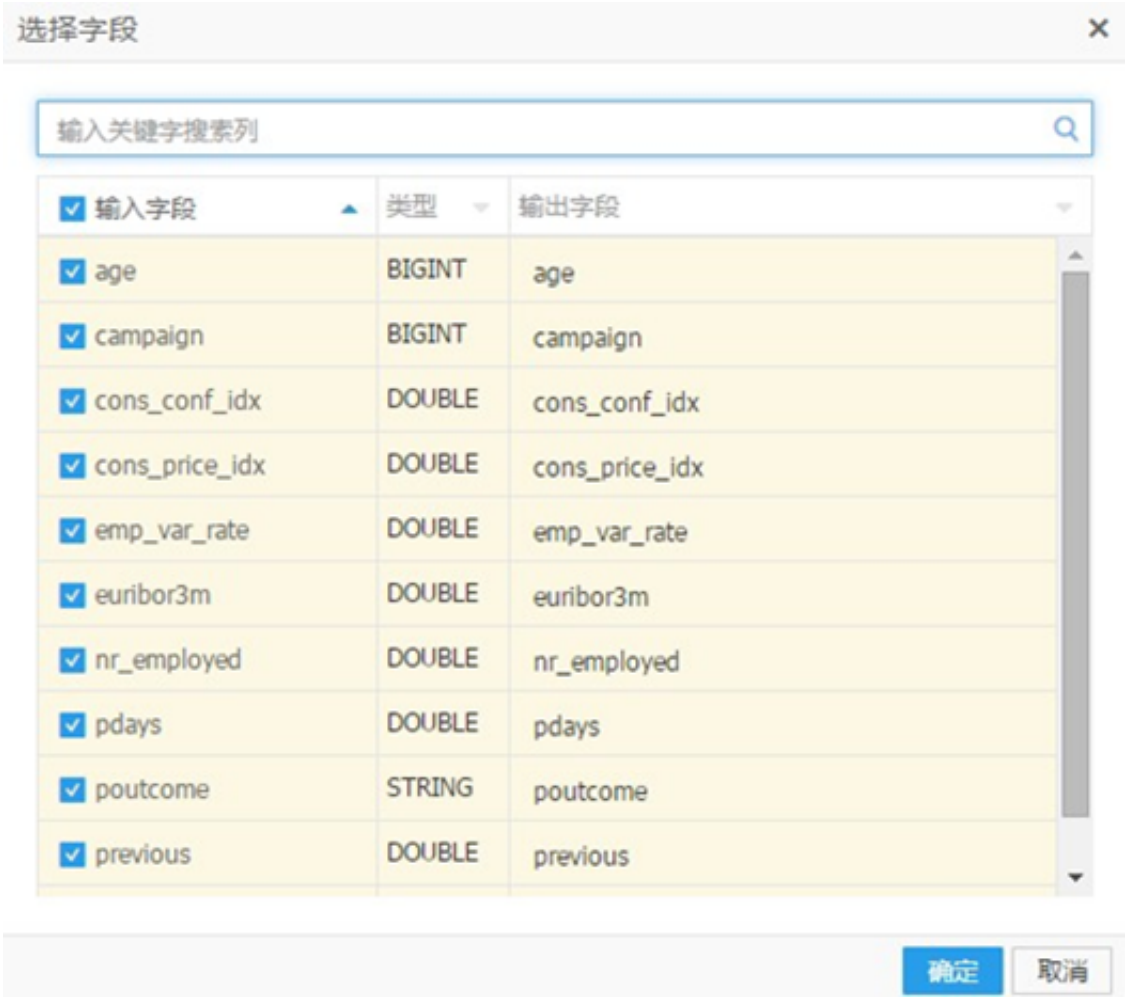


进行联合操作时，左右表选择的列数需相同，对应列的类型需保证一致；

- 可根据实际情况在条件框中手动输入已选字段的过滤条件，默认情况下全表，目前操作符支持**=**，**!=**，**>**，**<**，**>=**，**<=**，**like**和**rlike**；
- 系统默认勾选**去重**。勾选后将会对生成的数据表的重复行进行去重操作（distinct）；

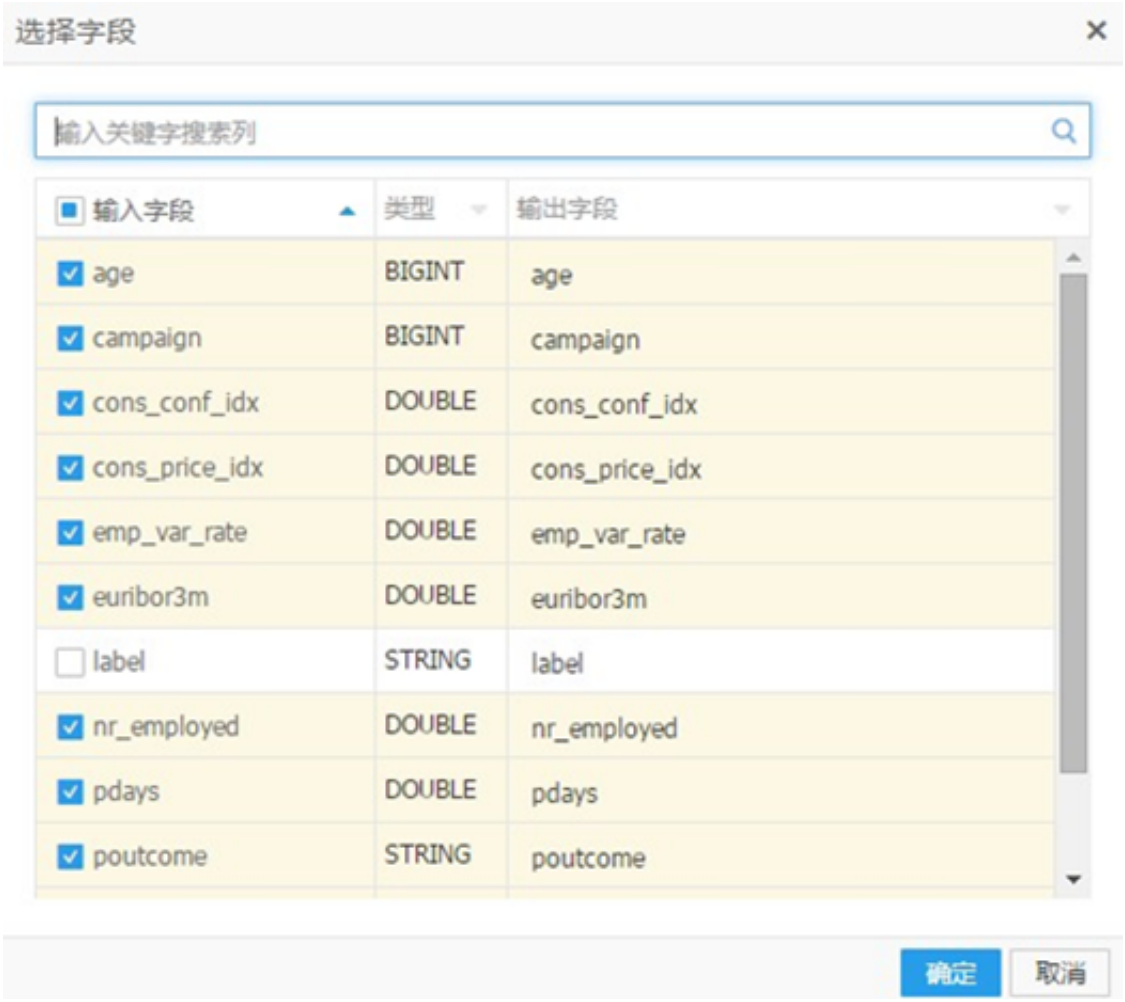
左表联合列，如图 *1871: 左表联合列*所示。

图 1871: 左表联合列



右表联合列，如图 *1872:* 右表联合列所示。

图 1872: 右表联合列



PAI 命令

不提供 PAI 命令。

34.4.2.3 其他

34.4.2.3.1 增加序号列

参数设置

在数据表第一列追加 ID 列，并将 append id 后的内容存为新表，如图 *1873:* 参数设置所示。

图 1873: 参数设置



PAI 命令

```
PAI –name AppendId –project algo_public -DIDColName="append_id" -DoutputTableName="test_11" -DinputTableName="bank_data" -DselectedColNames="age,campaign,cons_conf_idx,cons_price_idx,emp_var_rate,euribor3m,nr_employed,pda
```

- name：组件名字。
- project： project 名字，用于指定算法所在空间。系统默认是 algo_public，用户自己更改后系统会报错。
- IDColName：追加的 ID 列列名，从 0 开始编号，依次为0，1，2，3...。
- outputTableNames：输出表的名字。
- inputTableNmae：输入表的名字。
- selectedColNames：要保留的字段名，多个用逗号分隔。

34.4.2.3.2 归一化

对一个表的某一列或多列，进行归一化处理，产生的数据存入新表中。目前支持的是线性函数转换，表达式如下：y=(x-MinValue)/(MaxValue-MinValue)。

MaxValue、MinValue 分别为样本的最大值和最小值。

参数设置页面如图 *1874: 参数设置*所示。

图 1874: 参数设置



- 点击选择字段按钮可以选择想要归一化的列，目前支持 double 类型与 bigint 类型。
- 可以选择是否保留原始列，勾选后原始列会被保留，处理过的列重命名。

PAI 命令

```
PAI –name normalize_wf –project algo_public -DkeepOriginal="true" -DoutputTableName="test_4" -DinputPartitions="pt=20150501" -DinputTableName="bank_data_partition" -DselectedColNames="emp_var_rate,euribor3m";
```

算法参数

参数名称	参数描述	参数值可选项	默认值
inputTableName	必选，输入表	-	-
selectedColNames	可选，输入表选择列	-	默认选中全部列
inputTablePartitions	可选，训练输入表分区。	-	默认选中全表
outputTableName	必选，输出表	-	-
outputParaTableName	必选，配置输出表	-	-
inputParaTableName	可选，配置输入表	-	默认不输入
outputPMMLTableName	必选，PMML输出表	-	-
keepOriginal	可选，是否保留原始列（keepOriginal =true时，处理过的列重命名（normalized_前缀），原始列保留，keepOriginal=false时，全部列保留且不重命名）	-	默认为 false
lifecycle	可选，输出表生命周期	-	默认不设置

参数名称	参数描述	参数值可选项	默认值
coreNum	可选，核心个数	-	默认不设置，系统自动分配
memSizePerCore	可选，单个核心使用的内存数	-	默认不设置，系统自动分配

34.4.2.3.3 标准化

- 对一个表的某一列或多列，进行标准化处理，产生的数据存入新表中。
- 标准化所使用的公式：（X - Mean）/（standard deviation）。
- Mean：样本平均值。
- tandard deviation：样本标准偏差，针对从总体抽样，利用样本来计算总体偏差，为了使算出的值与总体水平更接近，就必须将算出的标准偏差的值适度放大，即 $\frac{1}{N-1}$
- 样本标准偏差公式：

  图 1875: 样本标准差公式

  $$S = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(X_i - \bar{X})^2}$$

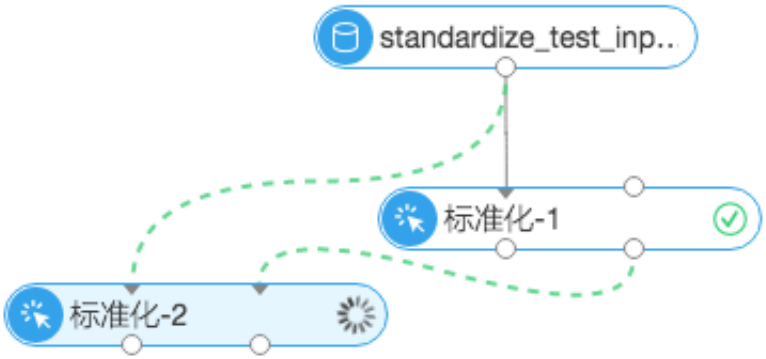
  其中，$\bar{X}$代表所采用的样本X1，X2，Xn的均值。
- 可以选择是否保留原始列，勾选后原始列会被保留，处理过的列重命名。
- 点击选择字段按钮可以选择想要标准化的列，目前支持 double 类型与 bigint 类型。

PAI 命令

```
PAI -name Standardize -project algo_public -DkeepOriginal="false"
    -DoutputTableName="test_5" -DinputTablePartitions="pt=20150501" -
DinputTableName="bank_data_partition" -DselectedColNames="euribor3m,pdays"
```

标准化界面

图 1876: 标准化



两个输入桩依次对应参数为：

- inputTableName 输入表，即要标准化的表。
- inputParaTableName 配置输入表，即标准化节点生成的参数列表，通过此参数，可以将一张表的配置参数应用到一张新的表。

两个输出桩依次对应参数为：

- outputTableName 输出表，即标准化完成的表。
- outputParaTableName 输出配置表，用于应用到其他的数据集上。

标准化参数界面

图 1877: 字段设置



- 保留原始列对应参数 keepOriginal。

参数说明

参数名称	参数描述	参数值可选项	默认值
inputTableName	必选，输入表的表名	-	-
selectedColNames	可选，输入表选择列	-	默认选中全部列
inputTablePartitions	可选，输入表中指定哪些分区参与训练，格式为：partition_name=value。如果是多级，格式为name1=value1/name2=value2；如果指定多个分区，中间用','分开	-	输入表的所有partition
outputTableName	必选，输出结果表	-	-
outputParaTableName	必选，配置输出表		
outputPartition	可选，输出表的partition	-	-
inputParaTableName	可选，配置输入表	-	默认不输入
keepOriginal	可选，保留原始列，原数列被重命名后保留，重命名规则是加“_orig”后缀	true/false	false
lifecycle	可选，输出表生命周期		默认不设置
coreNum	可选，核心个数		默认不设置，系统自动分配
memSizePerCore	可选，单个核心使用的内存数		默认不设置，系统自动分配

具体示例

```
drop table if exists standardize_test_input; create table standardize_test_input( col_string string, col_bigint bigint, col_double double, col_boolean boolean, col_datetime datetime); insert overwrite table standardize_test_input select * from ( select '01' as col_string, 10 as col_bigint, 10.1 as col_double, True as col_boolean, cast('2016-07-01 10:00:00' as datetime) as col_datetime from dual union all select cast(null as string) as col_string, 11 as col_bigint, 10.2 as col_double, False as col_boolean, cast('2016-07-02 10:00:00' as datetime) as col_datetime from dual union all select '02' as col_string, cast(null as bigint) as col_bigint, 10.3 as col_double, True as col_boolean, cast('2016-07-03 10:00:00' as datetime) as col_datetime from dual union all select '03' as col_string, 12 as col_bigint, cast(null as double) as col_double, False as col_boolean, cast('2016-07-04
```

```
10:00:00' as datetime) as col_datetime from dual union all select '04' as col_string, 13 as col_bigint, 10.4 as col_double, cast(null as boolean) as col_boolean, cast('2016-07-05 10:00:00' as datetime) as col_datetime from dual union all select '05' as col_string, 14 as col_bigint, 10.5 as col_double, True as col_boolean, cast(null as datetime) as col_datetime from dual ) tmp;
```

PAI 命令行

```
drop table if exists standardize_test_input_output; drop table if exists standardize_test_input_model_output; PAI -name Standardize -project algo_public -DoutputParaTableName="standardize_test_input_model_output" -Dlifecycle="28" -DoutputTableName="standardize_test_input_output" -DinputTableName="standardize_test_input" -DselectedColNames="col_double,col_bigint" -DkeepOriginal="true"; drop table if exists standardize_test_input_output_using_model; drop table if exists standardize_test_input_output_using_model_model_output; PAI -name Standardize -project algo_public -DoutputParaTableName="standardize_test_input_output_using_model_model_output" -DinputParaTableName="standardize_test_input_model_output" -Dlifecycle="28" -DoutputTableName="standardize_test_input_output_using_model" -DinputTableName="standardize_test_input"
```

输入说明

表 133: standardize_test_input

col_string	col_bigint	col_double	col_boolean	col_datetime
01	10	10.1	true	2016-07-01 10:00:00
NULL	11	10.2	false	2016-07-02 10:00:00
02	NULL	10.3	true	2016-07-03 10:00:00
03	12	NULL	false	2016-07-04 10:00:00
04	13	10.4	NULL	2016-07-05 10:00:00
05	14	10.5	true	NULL

输出说明

表 134: standardize_test_input_output

col_string	col_bigint	col_double	col_boolean	col_datetime	stdized_col_	stdized_col_double
01	10	10.1	true	2016-0	-1.264911064	-1.264911064

col_string	col_bigint	col_double	col_boolean	col_datetime	stdized_col_	stdized_col_double
				7-01 10:00:00	0673518	0683832
NULL	11	10.2	false	2016-07-02 10:00:00	-0.6324555320336759	1.3632455320341972
02	NULL	10.3	true	2016-07-03 10:00:00	NULL	0.0
03	12	NULL	false	2016-07-04 10:00:00	0.0	NULL
04	13	10.4	NULL	2016-07-05 10:00:00	0.6324555320336759	1.3632455320341859
05	14	10.5	true	NULL	1.2649110640673515	1.2649110640683718

表 135: standardize_test_input_model_output

Feature	json
col_bigint	{"name": "standardize", "type":"bigint", "paras": {"mean":12, "std": 1.58113883008419}}
col_double	{"name": "standardize", "type":"double", "paras": {"mean":10.3, "std": 0.1581138830082909}}

表 136: standardize_test_input_output_using_model

col_string	col_bigint	col_double	col_boolean	col_datetime
01	-1.2649110640673515	-1.2649110640683718	true	2016-07-01 10:00:00
NULL	-0.6324555320336759	-0.6324555320341971	false	2016-07-02 10:00:00
02	NULL	0.0	true	2016-07-03 10:00:00
03	0.0	NULL	false	2016-07-04 10:00:00
04	0.6324555320336759	0.6324555320341858	NULL	2016-07-05 10:00:00
05	1.2649110640673515	1.2649110640683718	true	NULL

表 137: standardize_test_input_output_using_model_model_output

feature	json
col_bigint	{"name": "standardize", "type":"bigint", "paras": {"mean":12, "std": 1.58113883008419}}
col_double	{"name": "standardize", "type":"double", "paras": {"mean":10.3, "std": 0.1581138830082909}}

34.4.3 特征工程

34.4.3.1 特征变换

34.4.3.1.1 主成分分析 PCA

PCA 利用主成分分析方法，实现降维和降维的功能。PCA算法原理见*wiki*。

目前支持稠密数据格式。

PAI 命令

```
PAI -name PrinCompAnalysis –project algo_public -DinputTableName=bank_data -DeigOutputTableName=pai_temp_2032_17900_2 -DprincompOutputTableName=pai_temp_2032_17900_1 -DselectedColNames=pdays,previous,emp_var_rate,cons_price_idx,cons_conf_idx,euribor3m,nr_employed -DtransType=Simple -DcalcuType=CORR -DcontriRate=0.9;
```

算法参数说明

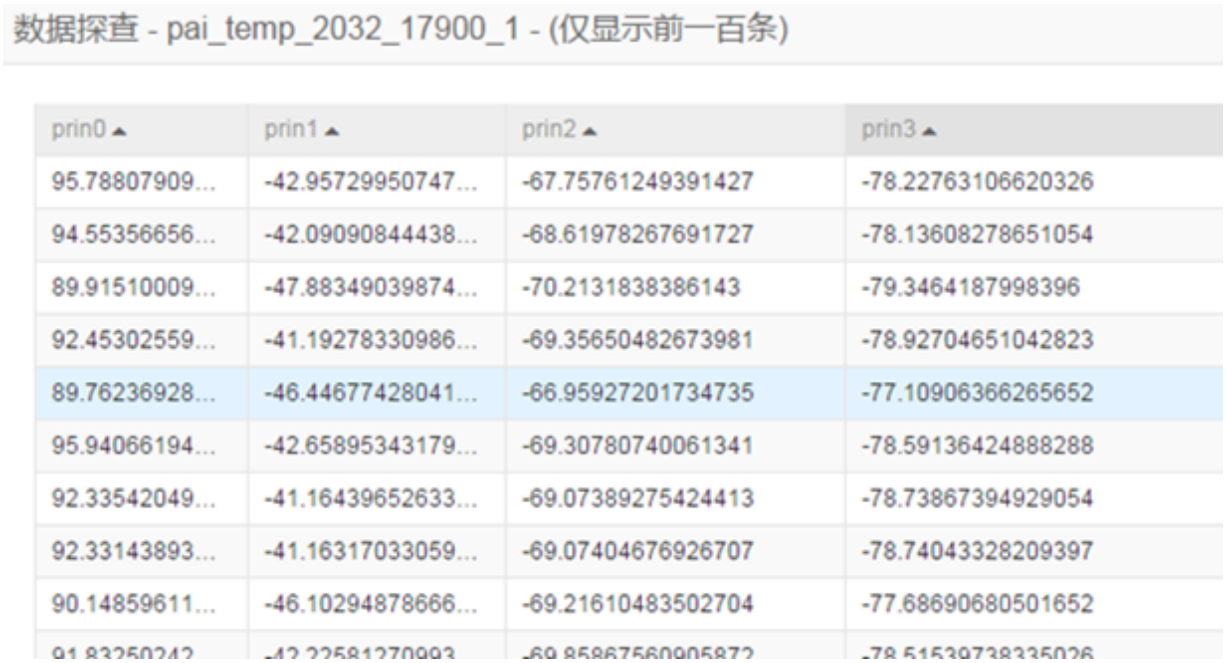
参数名称	参数描述	参数值可选项	默认值
inputTableName	必选，进行主成分分析的输入表	-	-
eigOutputTableName	必选，特征向量与特征值的输出表	-	-
princompOutputTableName	必选，进行主成分降维降噪后的结果输出表	-	-
selectedColNames	必选，参与主成分分析运算的特征列	-	-

参数名称	参数描述	参数值可选项	默认值
transType	可选，原表转换为主成分表的方式，支持Simple、 Sub-Mean和Normalization	-	Simple
calcuType	可选，对原表进行特征分解的方式，支持 CORR/COVAR_SAMP/COVAR_POP	-	CORR
contriRate	可选，降维后数据信息保留的百分比	-	0.9
remainColumns	可选，降维表保留原表的字段	-	-

PCA 输出示例

降维后的数据表如图 *1878:* 降维后的数据表所示。

图 1878: 降维后的数据表



特征值和特征向量表如图 *1879:* 特征值和特征向量表所示。

图 1879: 特征值和特征向量表

数据探查 - pai_temp_2032_17900_2 - (仅显示前一百条)

prin_name	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	eigenvalue	contributionrate	sumcontributionrate
prin0	0.2289...	-0.307801...	0.49043713976...	0.36762210238 47...	0.10370416863 3...	0.49327195...	0.4723413876...	3.864397508...	0.55205678698 04135	0.55205678698 04135
prin1	0.6651...	-0.511030...	-0.1750362148...	-0.306844720053...	-0.38529019595...	-0.1519122...	0.0083997057...	1.333469023...	0.19049557471 7606...	0.74255236169 80203
prin2	0.1344...	-0.248552...	-0.1027813425...	-0.355514661117...	0.88289843645 1...	0.01908078...	-0.057503451...	0.953306884...	0.13618669781 9829...	0.87873905951 78498
prin3	-0.216...	0.2034146...	0.07125284421...	-0.732942136387...	-0.16599040564...	0.21796215...	0.5426739826...	0.426458636...	0.06092266236 9309...	0.93966172188 71589

34.4.3.2 特征重要性评估

34.4.3.2.1 随机森林特征重要性

计算随机森林模型中特征重要性。

图 1880: 字段设置



PAI 命令

```
pai -name feature_importance -project algo_public -DinputTableName=input -DoutputTableName=output -Dlabel=label -DmodelName=model
```

算法参数

参数名称	参数描述	参数值可选项	默认值
inputTableName	必选，输入表	-	-
outputTableName	必选，输出表	-	-
labelColName	必选，label所在的列	-	-
modelName	必选，输入的模型名	-	-
featureColNames	可选，输入表选择的特征	-	默认除label外的其他列

参数名称	参数描述	参数值可选项	默认值
inputTablePartitions	可选，输入表选择的分区	-	默认选择全表
lifecycle	可选，输出表的生命周期	-	默认不设置
coreNum	可选，核心数	-	默认自动计算
memSizePerCore	可选，内存数	-	默认自动计算

34.4.3.2.2 线性模型特征重要性

基于预测结果和原始结果，评价线性算法模型的优劣，包含指标和残差直返图。其中指标包括SST，SSE， SSR，R2，R，MSE，RMSE，MAE，MAD，MAPE， count，yMean 和predictMean。

PAI 命令

```
pai -name regression_evaluation -project algo_public -DinputTableName=input_table -DyColName=y_col -DpredictionColName=prediction_col -DindexOutputTableName=index_output_table-DresidualOutputTableName=residual_output_table
```

表 138: 参数说明

参数名称	参数描述	参数值可选项	默认值
inputTableName	必选，输入表的表名	-	-
inputTablePartitions	可选，输入表中指定参与计算的分区	-	输入表的所有partitions
yColName	必选，输入表原始因变量列名，数值类型	-	-
predictionColName	必选，预测结果因变量列名，数值类型	-	-
indexOutputTableName	必选，回归指标输出表表名	-	-
residualOutputTableName	必选，残差直方图输出表表名	-	-
intervalNum	可选，直方图区间个数	-	100
lifecycle	可选，指定输出表的生命周期	正整数	没有生命周期

参数名称	参数描述	参数值可选项	默认值
coreNum	可选，指定 instance 个数	-	不设置
memSizePerCore	可选， 每个核的内存大小	-	不设置

输出结果

输出结果是一个 json ，json 个字段描述。

表 139:

字段名称	描述
SST	总平方和
SSE	误差平方和
SSR	回归平方和
R2	判定系数
R	多重相关系数
MSE	均方误差
RMSE	均方根误差
MAE	平均绝对误差
MAD	平均误差
MAPE	平均绝对百分误差
count	行数
yMean	原始因变量的均值
predictionMean	预测结果均值

34.4.3.3 统计分析

34.4.3.3.1 百分位

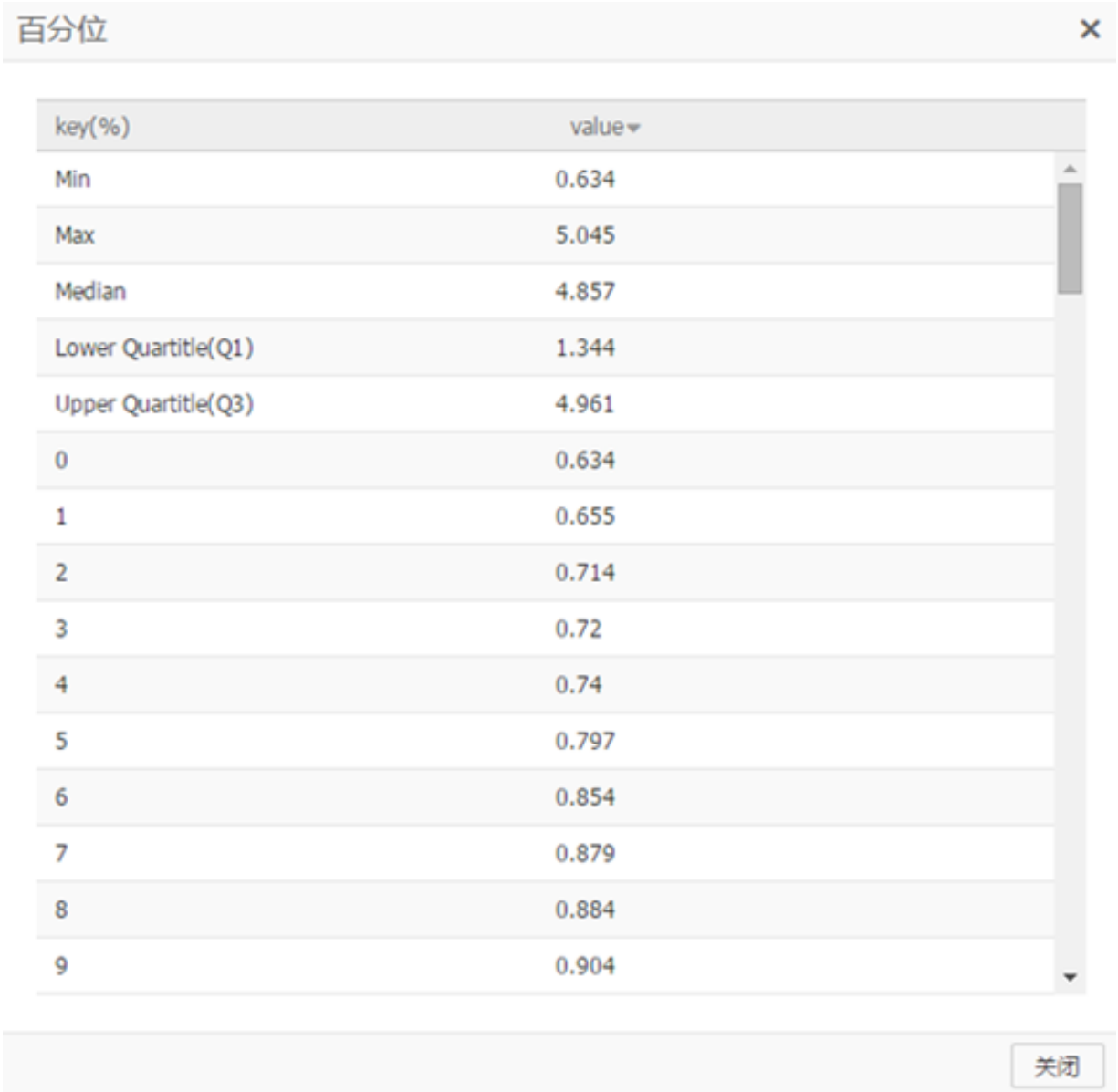
对一个存在的表，单列数据计算百分位。

参数设置

选择需要分析的字段，仅支持 double 类型和 bigint 类型。

运行结果如图 *1881:* 运行结果所示。

图 1881: 运行结果



PAI 命令

```
PAI –name Percentile –project algo_public -DoutputTableName="pai_temp_666_6014_1" -DcolName="euribor3m" -DinputTableName="bank_data";
```

- name：组件名字；
- project：project名字，用于指定算法所在空间。系统默认是 algo_public，用户自己更改后系统会报错；
- outputTableName：系统执行百分位运算后自动分配的结果表；

- colName：要计算百分位的列，仅支持数字型；
- inputTableName：输入表的名字。

34.4.3.3.2 全表统计

对一个存在的表，进行全表基本统计，或者仅对选中的列做统计。

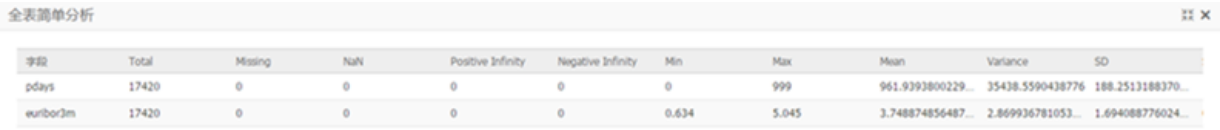
参数设置



选择输入列框中可选择需要进行统计的列，默认情况下统计全部列 - 条件框中可手动输入已选字段的过滤条件，默认情况下全表统计，目前操作符支持“=”，”！=”，”>”，”<”，“>=”，“<=”，“like”和“rlike”；

运行结果如图 *1882:* 运行结果所示。

图 1882: 运行结果



PAI 命令

```
PAI –name SimpleSummary –project algo_public -DsummaryColNames="euribor3m,pdays" -DoutputTableNames="pai_temp_667_6017_1" -DinputTableName="bank_data" -Dfilter="age>40";
```

- name：组件名字；
- project：project 名字，用于指定算法所在空间。系统默认是 algo_public，用户自己更改后系统会报错；
- summaryColNames：需要进行统计的列，多个列用逗号分隔；
- outputTableNames：系统执行全表统计运算后自动分配的结果表；

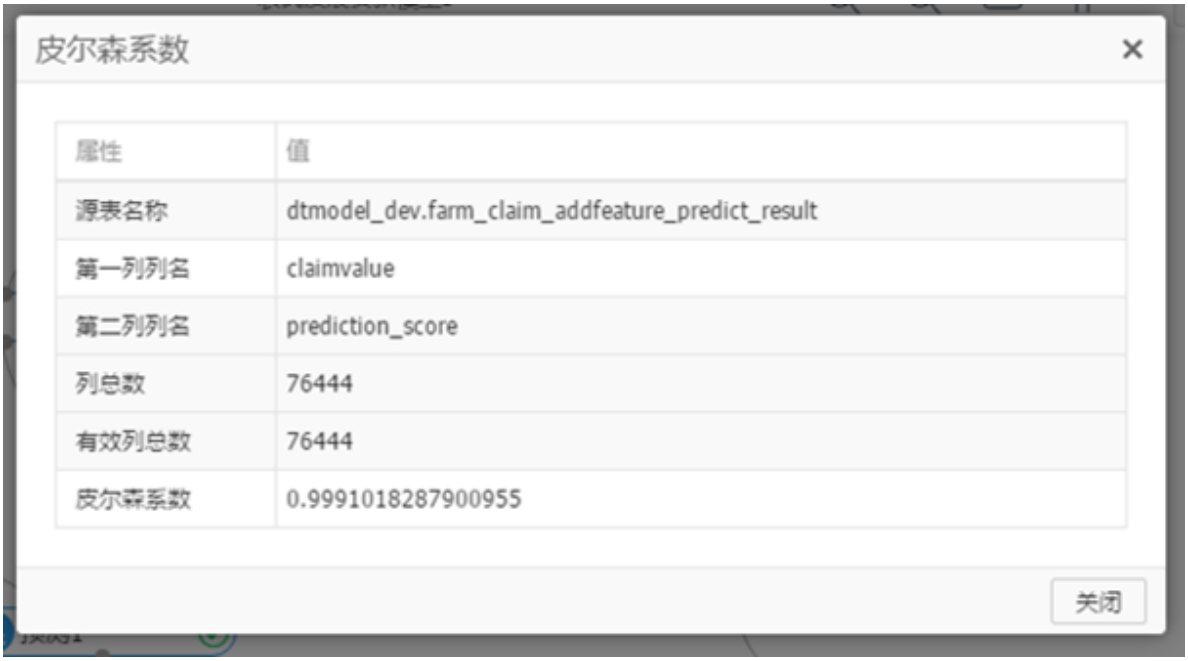
- inputTableName：输入表的名字；
- filter：过滤条件，目前操作符支持”=” ，” ！=” ，” >” ，” <” ， “>=” ， “<=” ， “like” 和 “rlike” 。

34.4.3.3.3 皮尔森系数

对输入表或分区的两列（必须为数值列），计算其 pearson 相关系数，结果存入输出表。

使用说明

- 组件的仅两个参数：输入列1、输入列2；将需要计算相关系数的两列的列名填入即可；
- 运行后，组件右击**菜单 > 查看分析报告。**



- 最后一列皮尔森系数值。

PAI 命令示例

```
pai –name pearson –project algo_test -DinputTableName=wpbc -Dcol1Name=f1 -Dcol2Name=f2
 -DoutputTableName=wpbc_pear;
```

算法参数

参数key名称	参数描述	取值范围	是否必选，默认值/行为
inputTableName	输入表的表名	表名	必选
inputTablePartitions	输入表中指定哪些分区参与计算	格式为：partition_name=value。如果是多级格式为name1=value1/name2=value2；如果是指定多个分区，中间用','分开	输入表的所有partition
col1Name	输入列1	列名	必选
col2Name	输入列2	列名	必选
outputTableName	输出	表名	必选

参数key名称	参数描述	取值范围	是否必选，默认值/行为
	结果表		

34.4.3.3.4 直方图

对一个存在的表，单列数据计算直方图。

参数设置

- 选择需要分析字段，支持 double 类型和 bigint 类型。
- 查看分析报告。



- 可调节步长大小，以及滑动查看直方图。

34.4.3.3.5 散点图

在回归分析中，数据点在直角坐标系平面上的分布图。

图 1883: 字段设置



PAI 命令

```
PAI -name scatter_diagram -project algo_public -DselectedCols=emp_var_rate,cons_price_rate,cons_conf_idx,euribor3m -DsampleSize=1000 -DlabelCol=y -DmapTable=pai_temp_2447_22859_2 -DinputTable=scatter_diagram -DoutputTable=pai_temp_2447_22859_1
```

表 140: 参数说明

参数key名称	参数描述	必填/选填	默认值
inputTable	输入数据表名	必填	-
inputTablePartitions	输入表分区	选填	-
outputTable	输出表名，存放采样样本	必填	-
mapTable	输出信息表，存放每个特征的最小最大值，枚举取值等	必填	-
selectedCols	输入表选择列名类型，用于绘制两两特征之间的散点图，最多勾选5个特征	必填	-
labelCol	把 Ins 或者 String 字段当做枚举标签列	选填	“”
sampleSize	对输入数据进行采样	选填	1000
lifecycle	输出表生命周期（单位：天）	选填	28

实例

输入数据：

```
create table scatter_diagram as select emp_var_rate,cons_price_rate,
 cons_conf_idx,euribor3m,y from pai_bank_data limit 10
```

emp_var_rate	cons_price_rate	cons_conf_idx	euribor3m	y
1.4	93.918	-42.7	4.962	0
-0.1	93.2	-42.0	4.021	0
-1.7	94.055	-39.8	0.729	1
-1.8	93.075	-47.1	1.405	0
-2.9	92.201	-31.4	0.869	1
1.4	93.918	-42.7	4.961	0
-1.8	92.893	-46.2	1.327	0
-1.8	92.893	-46.2	1.313	0
-2.9	92.963	-40.8	1.266	1
-1.8	93.075	-47.1	1.41	0
1.1	93.994	-36.4	4.864	0
1.4	93.444	-36.1	4.964	0
1.4	93.444	-36.1	4.965	1
-1.8	92.893	-46.2	1.291	0
1.4	94.465	-41.8	4.96	0
1.4	93.918	-42.7	4.962	0
-1.8	93.075	-47.1	1.365	1
-0.1	93.798	-40.4	4.86	1
1.1	93.994	-36.4	4.86	0
1.4	93.918	-42.7	4.96	0
-1.8	93.075	-47.1	1.405	0
1.4	94.465	-41.8	4.967	0
1.4	93.918	-42.7	4.963	0
1.4	93.918	-42.7	4.968	0
1.4	93.918	-42.7	4.962	0

emp_var_rate	cons_price_rate	cons_conf_idx	euribor3m	y
-1.8	92.893	-46.2	1.344	0
-3.4	92.431	-26.9	0.754	0
-1.8	93.075	-47.1	1.365	0
-1.8	92.893	-46.2	1.313	0
1.4	93.918	-42.7	4.961	0
1.4	94.465	-41.8	4.961	0
-1.8	92.893	-46.2	1.327	0
-1.8	92.893	-46.2	1.299	0
-2.9	92.963	-40.8	1.268	1
1.4	93.918	-42.7	4.963	0
-1.8	92.893	-46.2	1.334	0
1.4	93.918	-42.7	4.96	0
-1.8	93.075	-47.1	1.405	0
1.4	94.465	-41.8	4.96	0
1.4	93.444	-36.1	4.962	0
1.1	93.994	-36.4	4.86	0
1.1	93.994	-36.4	4.857	0
1.4	93.918	-42.7	4.961	0
-3.4	92.649	-30.1	0.715	1
1.4	93.444	-36.1	4.966	0
-0.1	93.2	-42.0	4.076	0
1.4	93.444	-36.1	4.965	0
-1.8	92.893	-46.2	1.354	0
1.4	93.444	-36.1	4.967	0
1.4	94.465	-41.8	4.959	0
-1.8	92.893	-46.2	1.354	0
1.4	94.465	-41.8	4.958	0
-1.8	92.893	-46.2	1.354	0
1.4	94.465	-41.8	4.864	0

emp_var_rate	cons_price_rate	cons_conf_idx	euribor3m	y
1.1	93.994	-36.4	4.859	0
1.1	93.994	-36.4	4.857	0
-1.8	92.893	-46.2	1.27	0
1.1	93.994	-36.4	4.857	0
1.1	93.994	-36.4	4.859	0
1.4	94.465	-41.8	4.959	0
1.1	93.994	-36.4	4.856	0
-1.8	93.075	-47.1	1.405	0
-1.8	92.843	-50.0	1.811	1
-0.1	93.2	-42.0	4.021	0
-2.9	92.469	-33.6	1.029	0
1.4	93.918	-42.7	4.962	0
-1.8	93.075	-47.1	1.365	0
1.1	93.994	-36.4	4.857	0
-1.8	92.893	-46.2	1.259	0
1.1	93.994	-36.4	4.857	0
1.4	94.465	-41.8	4.866	0
-2.9	92.201	-31.4	0.883	0
-0.1	93.2	-42.0	4.076	0
1.1	93.994	-36.4	4.857	0
1.4	93.918	-42.7	4.96	0
1.4	93.444	-36.1	4.962	0
1.1	93.994	-36.4	4.858	0
1.1	93.994	-36.4	4.857	0
1.1	93.994	-36.4	4.856	0
1.4	93.918	-42.7	4.968	0
1.4	93.444	-36.1	4.966	0
1.4	94.465	-41.8	4.962	0
1.4	93.444	-36.1	4.963	0

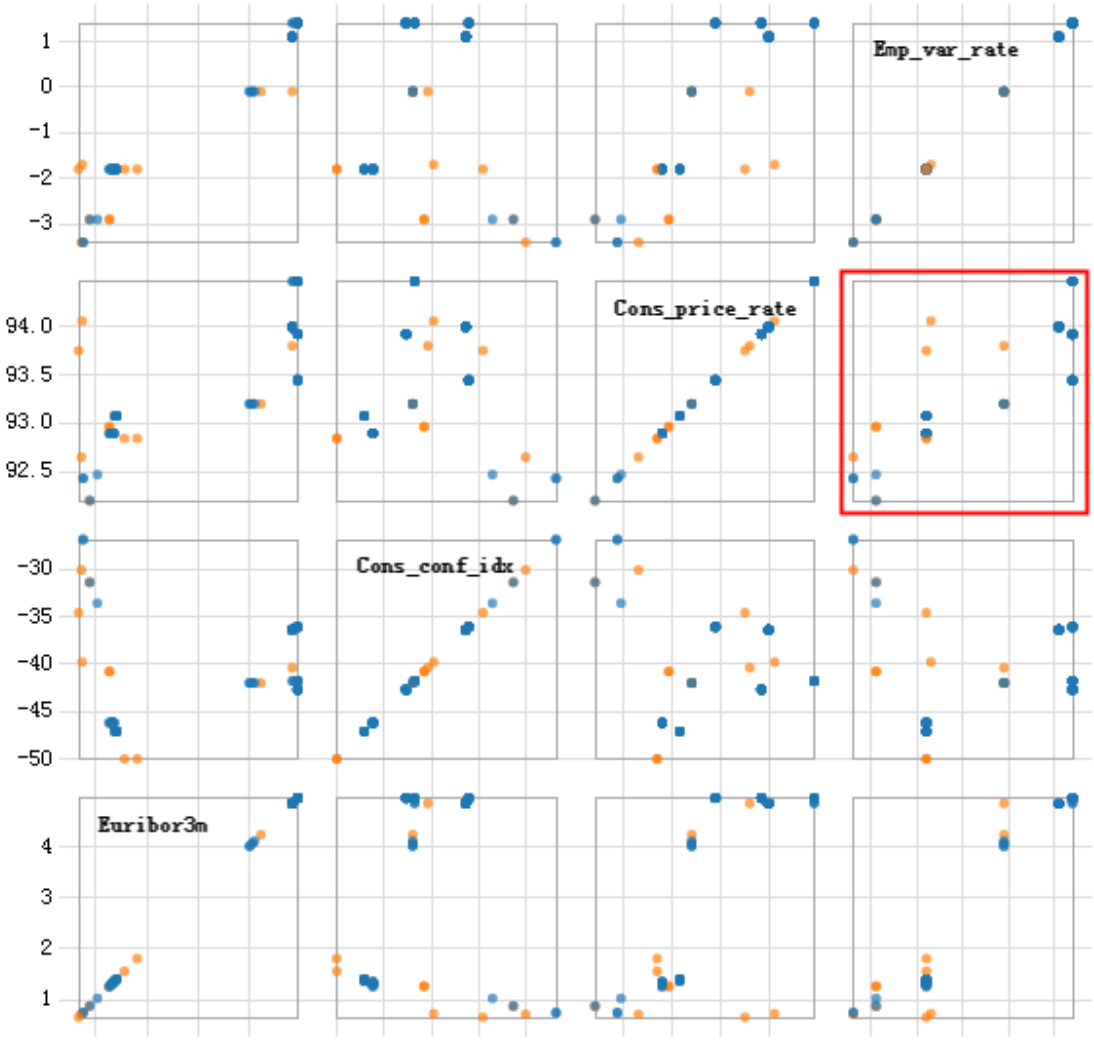
emp_var_rate	cons_price_rate	cons_conf_idx	euribor3m	y
-1.8	92.843	-50.0	1.56	1
1.4	93.918	-42.7	4.96	0
1.4	93.444	-36.1	4.963	0
-3.4	92.431	-26.9	0.74	0
1.1	93.994	-36.4	4.856	0
1.4	93.918	-42.7	4.962	0
1.1	93.994	-36.4	4.856	0
-0.1	93.2	-42.0	4.245	1
1.1	93.994	-36.4	4.857	0
-1.8	93.075	-47.1	1.405	0
-1.8	92.893	-46.2	1.327	0
-0.1	93.2	-42.0	4.12	0
1.4	94.465	-41.8	4.958	0
-1.8	93.749	-34.6	0.659	1
1.1	93.994	-36.4	4.858	0
1.1	93.994	-36.4	4.858	0
1.4	93.444	-36.1	4.963	0

参数配置

散点图参数配置：特征列选择 select

emp_var_rate，cons_price_rate，cons_conf_idx，euribor3m，可选的标签列选择 y。

运行效果：



可以直观的看出特征与特征直接，分类标签的分布情况。

34.4.3.3.6 相关系数矩阵

相关系数算法用于计算一个矩阵中每一列之间的相关系数，范围在[-1，1]之间。计算的时候，count数按两列间同时非空的元素个数计算，两两列之间可能不同。



PAI 命令

```
PAI -name corrcoef -project algo_public -DinputTableName=maple_test_corrcoef_basic12x10_input -DoutputTableName=maple_test_corrcoef_basic12x10_output -DcoreNum=1 -DmemSizePerCore=110
```

表 141: 参数说明

参数名称	参数描述	取值范围	是否必选，默认值
inputTableName	输入表	表名	必选
inputTablePartition	输入表中指定哪些分区参与训练，格式为: Partition_name=value. 如果是多级格式为 name1=value1/ name2=value2; 如果是指定多个分区， 中间用“,”分开	-	可选， 默认值选择所有分区
outputTableName	输出表名列表	表名	必选
selectedColNames	输入表选择列名类型	列名	可选默认选择全部列
lifecycle	可选，指定输出表的生命周	正整数	没有生命周期
coreNum	节点个数	与参数memSizePerCore配对使用，正整数，范围[1, 9999]	可选， 默认自动计算
memSizePerCore	单个节点内存大小，单位M	正整数，范围[1024, 64*1024]	可选， 默认自动计算

具体示例

数据生成：

col0:dou	col1:bigi	col2:dou	col3:bigi	col4:dou	col5:bigi	col6:dou	col7:bigi	col8:dou	col9:double
19	95	33	52	115	43	32	98	76	40
114	26	101	69	56	59	116	23	109	105
103	89	7	9	65	118	73	50	55	81
79	20	63	71	5	24	77	31	21	75

col0:dou	col1:bigi	col2:dou	col3:bigi	col4:dou	col5:bigi	col6:dou	col7:bigi	col8:dou	col9:double
87	16	66	47	25	14	42	99	108	57
11	104	38	37	106	51	3	91	80	97
84	30	70	46	8	6	94	22	45	48
35	17	107	64	10	78	53	34	90	96
13	61	39	1	29	117	112	2	82	28
62	4	102	88	100	36	67	54	12	85
49	27	44	93	68	110	60	72	86	58
92	119	0	113	41	15	74	83	18	111

PAI 命令

```
PAI -name corrcoef -project algo_public -DinputTableName=maple_test_corrcoef_basic12x10_input -DoutputTableName=maple_test_corrcoef_basic12x10_output -DcoreNum=1 -DmemSizePerCore=110
```

输入说明

表 142: 输出表

columnsnames	col0	col1	col2
col0	1	-0.2115657251820724	0.0598306259706561
col1	-0.2115657251820724	1	-0.8444477377898585
col2	0.0598306259706561	-0.8444477377898585	1
col3	0.2599903570684693	-0.17507636221594533	0.18518346647293102
col4	-0.3483249188225586	0.40943384150571377	-0.20934839228057014
col5	-0.28716254396809926	0.09135976026101403	-0.1896417512389659
col6	0.47880162127435116	-0.3018506374626574	0.1799377498863213
col7	-0.13646519484213326	0.40733726912808044	-0.3858885676469948
col8	-0.19500158764680092	-0.11827739124590071	0.20254569203773892
col9	0.3897390240949085	0.12433851389455183	0.13476160753756655

34.4.3.4 机器学习

34.4.3.4.1 二分类

34.4.3.4.1.1 线性支持向量机

支持向量机（SVM）是建立在 VC 维理论和结构风险最小原理基础上的一种模式识别方法。本版线性支持向量机不是采用核函数方式实现的，具体实现理论请参见：http://www.csie.ntu.edu.tw/~cjlin/papers/logistic.pdf 中的**Trust Region Method for L2-SVM**；本算法仅支持二分类。

参数设置

1. 设置组件的字段参数，如图 *1884:* 字段设置所示。

图 1884: 字段设置



输入列：选择输入列只支持 bigint 与 double 类型。

标签列：支持 bigint 类型、double 类型和 string 类型；本组件仅支持二分类问题。

2. 设置算法参数，如图 *1885:* 参数设置所示。

图 1885: 参数设置



- 惩罚因子：默认为1。
- 目标基准值：（可选）正例的值，不指定则随机选一个。建议正负例样本差异大时指定。
- 正例权重值：（可选）正例惩罚因子，默认1.0，范围（0, ~）。
- 负例权重值：（可选）负例惩罚因子，默认1.0，范围（0, ~）。
- 收敛系数：（可选）收敛误差，默认0.001，范围（0, 1）。

说明: 当不指定目标基准值时，正例权重值和负例权重值必须相同。

3. 运行结果（见随机森林组件说明）。

PAI 命令（未沿用类型设置节点）

```
PAI –name LinearSVM –project algo_public -DnegativeCost="1.0" -DmodelName="xlab_m_LinearSVM_6143" -DpositiveCost="1.0" -Depsilon="0.001" -DlabelColName="y" -DfeatureColNames="pdays,emp_var_rate,cons_conf_idx" -DinputTableName="bank_data" -DpositiveLabel="0";
```

- name：组件名字。
- project：project 名字。用于指定算法所在空间，系统默认是 algo_public，用户自己更改后系统会报错。
- negativeCost：（可选）负例权重值。即负例惩罚因子，默认1.0，范围（0, ~）。
- modelName：输出的模型名。

- positiveCost：（可选）正例权重值。即正例惩罚因子，默认1.0，范围（0, ~）。
- epsilon：（可选）收敛系数。默认值为0.001，范围（0, 1）。
- labelColName：标签列列名。
- featureColNames：输入表中选择的用于训练的特征列名。
- inputTableNmae：训练输入表的表名。
- positiveLabel：（可选）正例值。不指定则随机选一个。

34.4.3.4.1.2 逻辑回归二分类

经典逻辑回归是一个二分类算法，算法平台的逻辑回归可以支持多分类。逻辑回归组件支持稀疏、稠密两种数据格式。

参数设置



- 最大迭代次数：（可选）L-BFGS的最大迭代次数，默认是100；
- 最小收敛误差：（可选）L-BFGS 的终止条件，即两次迭代之间 log-likelihood 的差，默认为1.0e-06；
- 正则项：（可选）正则化类型，可以选择 ‘l1、 ‘l2’ 、 ‘None’ ，默认为 ‘l1’ ；
- 正则系数：（可选）正则项系数，默认为 1.0；当 regularizedType 为 None 时，该项会被忽略；

逻辑回归组件运行结果是模型，可在模型列表中查找，如图 *1886:* 查找逻辑回归组件所示。

图 1886: 查找逻辑回归组件



模型名称的命名规则：实验名 + "-" + 算法组件名称 + "model"。

PAI 命令（未沿用类型设置节点）

```
PAI –name LogisticRegression –project algo_public -DmodelName="xlab_m_logistic_regression_6096" -DregularizedLevel="1" -DmaxIter="100" -DregularizedType="l1" -Depsilon="0.000001" -DlabelColName="y" -DfeatureColNames="pdays,emp_var_rate" -DgoodValue="1" -DinputTableName="bank_data";
```

- name：组件名字。
- project：project 名字。用于指定算法所在空间，系统默认是 algo_public，用户自己更改后系统会报错。
- modelName：输出的模型名。
- regularizedLevel：（可选）正则化系数。默认为 1.0；当 regularizedType 为 None 时，该项会被忽略。
- maxIter：（可选）最大迭代数。指定L-BFGS的最大迭代次数，默认是100。
- regularizedType：（可选）正则化类型，可以选择**l1**、**12**、**None**，默认为**l1**。
- epsilon：（可选）收敛误差。L-BFGS 的终止条件，即两次迭代之间 log-likelihood 的差，默认为**1.0e-06**。
- labelColName：输入表标签列列名。
- featureColNames：输入表中选择的用于训练的特征列名。
- goodValue：（可选）目标基准值。二分类时，指定训练系数针对的 label 值，如果填空，会随机选择一个。
- inputTableName：训练输入表的表名。

34.4.3.4.1.3 GBDT二分类

在 GBDT 回归与排序的基础上，用于二分类问题，既设定阈值：大于阈值为正例，反之为负例。

组件介绍

1. 向画布拖入 GBDT 二分类组件训练，调整参数，如图 *1887:* 字段设置所示。

图 1887: 字段设置

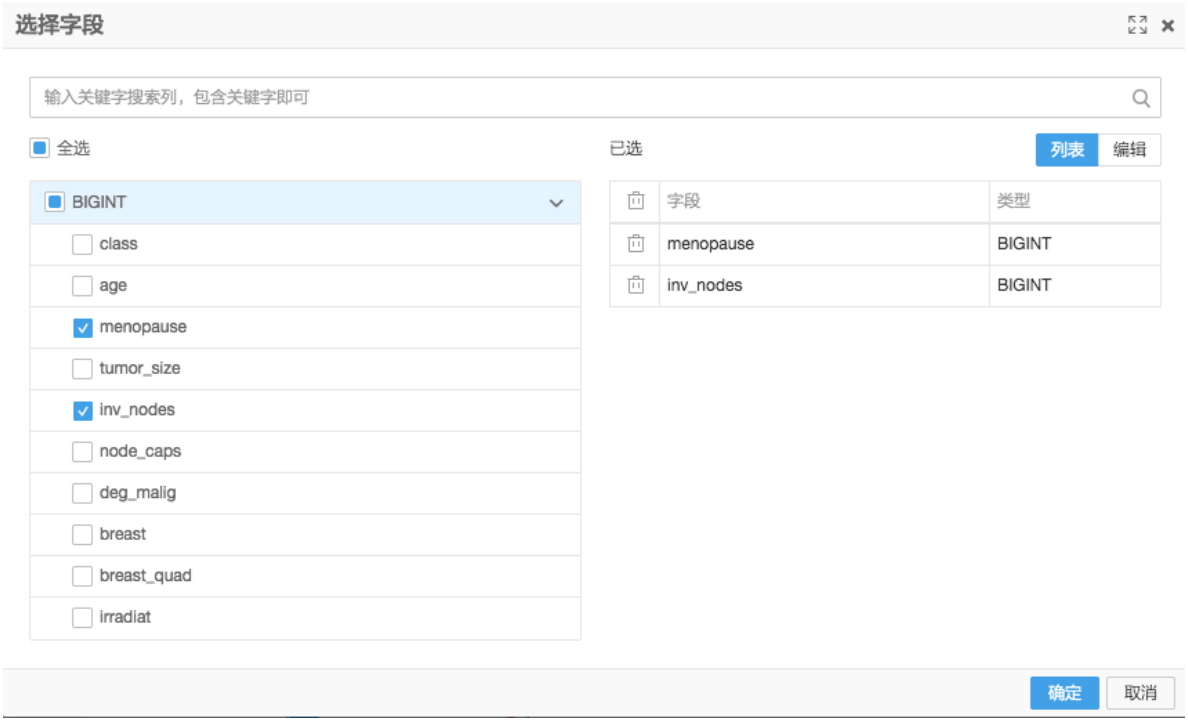


沿用类型设置节点必须在输入表进行类型设置处理后才能勾选（参照随机森林训练和生成模型部分的步骤2和3）；

- 选择特征列：支持 double 类型与 bigint 类型，最多支持 800 列输入；
- 选择标签列：只能选择非输入列的其他列，value 只能是二分类，否则会报错，支持 bigint 类型；
- 选择分组列(可选)，默认是全表，支持 double 类型与 bigint 类型。

2. 选择的输入列可调整字段类型，如图 *1888:* 选择字段所示。

图 1888: 选择字段



GBDT 二分类输入列只支持连续型，选择连续型和离散型处理方式一致。

3. 调整参数设置，如图 *1889:* 参数设置所示。

图 1889: 参数设置



- metric类型：0(NDCG)-：normalized discounted cumulative gain；1(DCG)：discounted cumulative gain；2 (AUC) 只适应0/1 label（默认值）。
- 树的数目：范围[1,10000]，默认500。

- 学习速率：范围(0,1)，默认0.05。
- 最大叶子数：必须为整数，范围[2,1000]，默认32。
- 树最大深度：必须为整数，范围[1,11]，默认为11。
- 叶节点最少样本数：必须为整数，范围[100,1000]，默认500。
- 训练采集样本比例：范围(0,1)，默认0.6。
- 训练采集特征比例：范围（0,1），默认0.6。
- 测试数据比例：范围[0,1]，默认0.0。
- 随机数种子：必须为整数，范围[0,10]，默认0。
- 特征分裂的最大数量：范围[1,1000]，默认500。

4. 运行结果（见随机森林组件说明）。

说明:

- GBDT 与 GBDT_LR 默认损失函数类型不一致：GBDT 默认为 regression loss:mean squared error loss，GBDT_LR 默认为 logistic regression loss。其中 GBDT_LR 不需要用户设置损失函数类型，系统直接写入默认损失函数；
- GBDT 二分类的标签列只能选择二分类列，且不支持 string 型；
- 连接 ROC 曲线时预测组件应该选择自定义并选择目标基准值。

PAI 命令（未沿用类型设置节点）

```
PAI -name GBDT_LR -project algo_public -DfeatureSplitValueMaxSize="500" -DrandSeed="0" -Dshrinkage="0.5" -DmaxLeafCount="32" -DlabelColName="y" -DinputTableName="bank_data_partition" -DminLeafSampleCount="500" -DgroupIDColName="nr_employed" -DsampleRatio="0.6" -DmaxDepth="11" -DmodelName="xlab_m_GBDT_LR_21208" -DmetricType="2" -DfeatureRatio="0.6" -DinputTablePartitions="pt=20150501" -DtestRatio="0.0" -DfeatureColNames="age,previous,cons_conf_idx,euribor3m" -DtreeCount="500";
```

- name：组件名字。
- project：（可选）默认 project 是 algo_public。如果选择其他的 project，需要指定 algo_public 下的算法包，否则报错。
- featureSplitValueMaxSize：（可选）一个特征分裂的最大数量，范围[1,1000]，默认500。
- randSeed:（可选）随机数种子，必须为整数，范围：[0,10]，默认0。
- shrinkage：（可选）学习速率。范围(0-1)，默认0.05。
- maxLeafCount：（可选）最大叶子数，必须为整数，范围：[2,1000]，默认32。
- labelColName：输入表中选择的标签列列名。
- inputTableName：训练输入表的表名。

- minLeafSampleCount：（可选）叶子节点容纳的最少样本数，必须为整数，范围：[100,1000]，默认500。
- groupIDColName:（可选）数据分组列，默认将整个表作为一个组。
- sampleRatio：（可选）训练采集样本比例，范围：(0,1)，默认0.6。
- maxDepth：（可选）一棵树的最大深度，必须为整数，范围：[1,11]，默认为11。
- modelName：输出的模型名。
- metricType：metric类型。（可选）0(NDCG)-：normalized discounted cumulative gain；1(DCG) : discounted cumulative gain；2 (AUC) 只适应0/1 label（默认值）。
- featureRatio：（可选）训练中采集的特征比例，范围：（0,1），默认0.6。
- inputTablePartitions：（可选）预测输入表分区。输入表对应的输入分区，选中全表则为None。
- testRatio：（可选）测试样本数比例，范围：[0-1]，默认0.0。
- featureColNames：输入表中用于训练的特征列。
- treeCount：（可选）树数量，范围 [1,10000]，默认500。

34.4.3.4.2 多分类

34.4.3.4.2.1 逻辑回归多分类

经典逻辑回归是一个二分类算法，算法平台的逻辑回归可以支持多分类。逻辑回归组件支持稀疏、稠密两种数据格式。

参数设置



- 最大迭代次数：（可选）L-BFGS 的最大迭代次数，默认是100。
- 最小收敛误差：（可选）L-BFGS 的终止条件，即两次迭代之间 log-likelihood 的差，默认为1.0e-06。
- 正则项类型：（可选）正则化类型，可以选择 ‘l1、 ‘l2’ 、 ‘None’ ，默认为 ‘l1’ 。
- 正则项系数：（可选）正则项系数，默认为 1.0；当 regularizedType 为 None 时，该项会被忽略。

逻辑回归组件运行结果是模型，可在模型列表中查找，如图 *1890:* 查找逻辑回归组件所示。

图 1890: 查找逻辑回归组件



模型名称的命名规则：实验名 + "-" + 算法组件名称 + "model"。

PAI 命令（未沿用类型设置节点）

```
PAI –name LogisticRegression –project algo_public -DmodelName="xlab_m_logistic_regression_6096" -DregularizedLevel="1" -DmaxIter="100" -DregularizedType="l1" -Depsilon="0.000001" -DlabelColName="y" -DfeatureColNames="pdays,emp_var_rate" -DgoodValue="1" -DinputTableName="bank_data";
```

- name：组件名字。
- project：project 名字。用于指定算法所在空间，系统默认是 algo_public，用户自己更改后系统会报错。
- modelName：输出的模型名。
- regularizedLevel：（可选）正则化系数。默认为 1.0；当 regularizedType 为 None 时，该项会被忽略。
- maxIter：（可选）最大迭代数。指定 L-BFGS 的最大迭代次数，默认是100。
- regularizedType：（可选）正则化类型，可以选择l1、l2、None，默认为l1。
- epsilon：（可选）收敛误差。L-BFGS的终止条件，即两次迭代之间 log-likelihood 的差，默认为1.0e-06。
- labelColName：输入表标签列列名。
- featureColNames：输入表中选择的用于训练的特征列名。
- goodValue：（可选）目标基准值。二分类时，指定训练系数针对的 label 值；如果填空，会随机选择一个。
- inputTableName：训练输入表的表名。

34.4.3.4.2.2 随机森林

随机森林是一个包含多个决策树的分类器，并且其输出的类别是由个别树输出的类别的众数而定。

参数设置

1. 向画布拖入随机森林组件训练，设置字段。



   - 输入列：支持 string 类型、double 类型与 bigint 类型；
   - 标签列：只能选择非输入列的其他列，支持 string 类型、double 类型和 bigint 类型；
   - 权重列的列名：（可选）只能选择非输入列和非标签列，支持 double 类型和 bigint 类型。
2. 设置步骤2拖入的随机森林组件的参数。



- 树的数目：森林中树的个数，范围(0, 1000)；
- 随机属性类型：（可选）单颗树在生成时，随机选择的特征个数。可供选择的类型有logN，N/3，sqrtN，N 四种类型；
- 树最大深度：（可选）单颗树的最大深度。-1表示完全生长。范围[1, ∞]；
- 叶子节点最少记录数：（可选）叶节点数据的最小个数。最小个数为2；
- 叶子节点最少记录百分比：（可选）叶节点数据个数占父节点的最小比例，-1表示无这一限制。默认-1，范围[0,100]；
- 每棵树最大记录数：（可选）森林中单颗树输入的随机数据的个数。范围为(1000, 1000000)；
- 单棵树的算法类型：（可选）设置森林中的每棵树的算法类型，有三种算法类型：id3,c4.5,cart。设置格式必须为[a,b]，a、b 为数字；比如有 n棵树，algorithmTypes=[a,b]，则[0,a] 是 id3，[a,b] 是 cart， [b,n] 是c4.5。例如：在一个拥有5棵树的森林中，[2，4]表

示0，1为id3算法，2, 3为 cart 算法，4为 c4.5 算法。如果输入为 None，则算法在森林中均分。

- **说明:**
  - 随机森林通过对 bagging 方法的改进，在大数据集上构建一个单颗树不相关的森林。在很多问题上，随机森林跟 boosting 方法类似，有类似的训练过程；
  - 本方法对于单颗树的增长，可选的有 id3，cart 或 c4.5。通过参数 treeNum 来指定森林中树的个数，树个数的范围为[1，1000]。通过编辑好的模版，可以控制单棵树的结构。可以通过其他参数控制单颗树叶结点上数据的最小个数，叶结点上数据占父节点的最小比例数，树的最大深度等；
  - 权重列的每一行和样本对应，表示这条样本在训练时占的比重。比如选择了 age 列作为权重列，那 age 列中权重高的 value 所在的条样本在训练时比重会更高；
  - 当出现 input table is empty! 错误时，可能是以下情况：抽样比率设置太小，也就是 maxRecordSize 太小；输入数据为空表。

PAI 命令（未沿用类型设置节点）

```
PAI –name RandomForests –project algo_public -DmodelName="xlab_m_random_forests_6036" -DrandomColNum="1.0" -DlabelColName="campaign" -DmaxTreeDeep="10" -DmaxRecordSize="100000" -DfeatureColNames="age,pdays,previous,emp_var_rate,cons_price_idx,cons_conf_idx,euribor3m,nr_employe
    -DisFeatureContinuous="1,1,1,1,1,1,1,1" -DminNumPer="-1" -DminNumObj="2" -DinputTableName="bank_data" -DweightColName="y" -DtreeNum="10";
```

- name：组件名字。
- project：project 名字。用于指定算法所在空间，系统默认是 algo_public，用户自己更改后系统会报错；
- modelName：输出的模型名。
- randomColNum：（可选）随机属性类型。单颗树在生成时，每次随机选择的特征个数。-1表示log2N（N为特征的总个数），有效值为[1,N]，其中 N 为 feature 个数。
- labelColName：目标列。输入表中选择的标签列列名；
- maxTreeDeep：（可选）树最大深度。-1表示完全生长。范围[1, ∞]。
- maxRecordSize：（可选）每棵树最大记录数。森林中单颗树输入的随机数据的个数。范围为（1000, 1000000）。-1表示100000。

- featureColNames：特征列。输入表中选择的用于训练的特征列名。
- isFeatureContinuous：特征列类型，表示对应特征是否为连续值。1为连续，0为离散。1,0,0表示三列特征中第一列为连续，第二和第三列为离散，值个数对应features 长度。
- minNumPer：（可选）叶子节点最少记录百分比，即叶节点数据个数占父节点的最小比例。-1表示无这一限制。范围[0.0,1.0]。
- minNumObj：（可选）叶子节点最少记录数。即叶节点数据的最小个数，最小个数为。
- inputTableNmae：训练输入表的表名。
- weightColName：（可选）权重列。输入表中权重列的列名，无权重列则为None，weightColName>0。
- treeNum：树的数目。即森林中树的个数，范围（0, 1000）。

34.4.3.4.2.3 朴素贝叶斯

朴素贝叶斯分类是一种应用基于独立假设的贝叶斯定理的简单概率分类算法，更精确的描述这种潜在的概率模型为独立特征模型。算法详细信息请参见： Naive Bayes classifier 。

组件介绍

组件设置参数（参照随机森林组件）。



PAI 命令

```
PAI –name NaiveBayes –project algo_public -DmodelName="xlab_m_NaiveBayes_23772" -DinputTablePartitions="pt=20150501" -DlabelColName="poutcome" -DfeatureColNames="age,previous,cons_conf_idx,euribor3m" -DisFeatureContinuous="1,1,1,1" -DinputTableName="bank_data_partition";
```

- name：组件名字；
- project：（可选）默认 project 是 algo_public。如果选择其他的 project，需要指定 algo_public 下的算法包，否则报错；
- modelName：训练生成的模型名；
- inputTablePartitions：（可选）预测输入表分区。输入表对应的输入分区，选中全表则为 None；
- labelColName：输入表中选择的标签列列名；
- featureColNames：输入表中用于训练的特征列；
- isFeatureContinuous：特征对应的类型，表示对应特征是否为连续值。1为连续，0为离散。1,0,0表示三列特征中第一列为连续，第二和第三列为离散，值个数对应 features 长度；
- inputTableName：输入的表名。

34.4.3.4.3 协同过滤etrec

etrec 是一个 item base 的协同过滤算法，输入为两列，输出为 item 之间相似度 topK。jaccard 可参考：*Jaccard_index*。



- 分别填写用户和商品对应的字段



- 可以配置三种相似度类型
- TopN 为提取相似度最高的前 n 个
- 计算行为：同一个商品出现多次的情况下的计算行为

PAI 命令

```
PAI -name pai_etrec -project algo_public -DsimilarityType="wbcosine" -
Dweight="1" -DminUserBehavior="2" -Dlifecycle="28" -DtopN="2000" -
Dalpha="0.5" -DoutputTableName="etrec_test_result" -DmaxUserBehavior="500" -
```

```
DinputTableName="etrec_test_input" -Doperator="add" -DuserColName="user" -
DitemColName="item"
```

参数说明

表 143: 参数说明

参数名称	参数描述	参数值可选项	默认值
inputTableName	必选，输入表名	-	-
userColName	必选，输入表选择的user列名	-	-
itemColName	必选，输入表选择的item列名	-	-
	可选，输入表选择的payload列名	-	默认无payload列
inputTablePartitions	可选，输入表选择的分区名	-	默认选择全表
outputTableName	必选，输出表名	-	-
outputTablePartition	可选，输出表的partition	-	无partition
similarityType	可选，相似度类型	wbcosine，asymcosine，jaccard	wbcosine
topN	可选，相似度最大的N个item	[1,10000]	2000
minUserBehavior	可选，最小用户行为	[2,)	2
maxUserBehavior	可选，最大用户行为	[2,100000]	500
itemDelimiter	可选，输出表中item之间的分隔符	-	" "
kvDelimiter	可选，输出表中key-value之间的分隔符	-	":"
alpha	可选，asymcosine的平滑因子的值	-	0.5
weight	可选，asymcosine的权重指数	-	1.0

参数名称	参数描述	参数值可选项	默认值
operator	可选，当同一个user对应出现相同的item时发生的行为	add,mul,min,max	add
lifecycle	可选，输出结果表的生命周期	-	1

具体示例

数据生成：

```
drop table if exists etrec_test_input; create table etrec_test_input as select * from ( select cast(0 as string) as user, cast(0 as string) as item from dual union all select cast(0 as string) as user, cast(1 as string) as item from dual union all select cast(1 as string) as user, cast(0 as string) as item from dual union all select cast(1 as string) as user, cast(1 as string) as item from dual ) a;
```

PAI 命令

```
drop table if exists etrec_test_result; PAI -name pai_etrec -project algo_public -DsimilarityType="wbcosine" -Dweight="1" -DminUserBehavior="2" -Dlifecycle="28" -DtopN="2000" -Dalpha="0.5" -DoutputTableName="etrec_test_result" -DmaxUserBehavior="500" -DinputTableName="etrec_test_input" -Doperator="add" -DuserColName="user" -DitemColName="item"
```

输入说明：

表 144: etrec_test_input

user	item
0	0
0	1
1	0
1	1

输出说明：

表 145: etrec_test_result

itemid	similarity
0	1:1
1	0:1

34.4.3.4.4 GBDT 回归

GBDT （梯度渐进回归树）是一种迭代的决策树算法，该算法由多棵决策树组成，所有树的结论累加起来做最终答案。GBDT 几乎可用于所有回归问题（线性/非线性），相对逻辑回归仅能用于线性回归，GBDT 的适用面非常广。

更多内容，请参阅*GBDT 二分类*。

PAI 命令

```
PAI -name gbdt -project algo_public -DfeatureSplitValueMaxSize="500" -DlossType="0"
-DrandSeed="0" -DnewtonStep="0" -Dshrinkage="0.05" -DmaxLeafCount="32"
-DlabelColName="campaign" -DinputTableName="bank_data_partition" -
DminLeafSampleCount="500" -DsampleRatio="0.6" -DgroupIDColName="age"
-DmaxDepth="11" -DmodelName="xlab_m_GBDT_83602" -DmetricType="2" -
DfeatureRatio="0.6" -DinputTablePartitions="pt=20150501" -Dtau="0.6" -Dp="1"
-DtestRatio="0.0" -DfeatureColNames="previous,cons_conf_idx,euribor3m" -
DtreeCount="500"
```

表 146: 参数说明

参数名称	参数描述	取值范围	是否必选，默认值
inputTableName	输入表	表名	必选
featureColNames	输入表中选择的用于训练的特征列名	列名	可选，默认值选择所有数值列
labelColName	输入表标签列列名	列名	必选
inputTablePartitions	输入表中指定哪些分区参与训练， 格式为: partition_name=value. 如果是多级格式为name1=value1/name2=value2; 如果是指定多个分区， 中间用','分开	-	可选， 默认值选择所有分区
modelName	输出模型名	-	必选
outputImportanceTableName	输出特征重要性表名	-	可选
groupIDColName	数据分组列	列名	可选，默认值整表
lossType	损失函数类型，0:GBRANK，1:LAMBDAMART_DCG，2:LAMBDAMART_NDCG，3:LEAST_SQ	0,1,2,3,4	可选，默认为 0

参数名称	参数描述	取值范围	是否必选，默认值
	metric类型，0(NDCG)-：normalized discounted cumulative gain；1(DCG)：discounted cumulative gain；2(AUC) 只适应0/1 label	0,1,2	可选，默认值2
treeCount	树数量	[1,10000]	可选，默认值500
shrinkage	学习速率	(0,1]	可选，默认0.05
maxLeafCount	最大叶子数，必须为整数	[2,1000]	可选，默认32
	一棵树的最大深度，必须为整数	[1,11]	可选，默认11
minLeafSampleCount	叶子节点容纳的最少样本数，必须为整数	[100,1000]	可选，默认500
sampleRatio	训练采集样本比例	(0,1]	可选，默认0.6
featureRatio	训练中采集的特征比例	(0,1]	可选，默认0.6
tau	gbrank loss中的Tau参数	[0,1]	可选，默认0.6
p	gbrank loss中的p参数	[1,10]	可选，默认1
randSeed	随机数种子	[0,10]	可选，默认0
newtonStep	是否使用newton法	0,1	可选，默认1
featureSplitValueMaxSize	一个特征分裂的最大数量	[1,1000]	可选，默认500
lifecycle	输出表的生命周期	-	可选，默认不设置

具体示例

数据生成：

```
drop table if exists gbdt_ls_test_input; create table gbdt_ls_test_input as select * from ( select cast(1 as double) as f0, cast(0 as double) as f1, cast(0 as double) as f2, cast(0 as double) as f3, cast(0 as bigint) as label from dual union all select cast(0 as double) as f0, cast(1 as double) as f1, cast(0 as double) as f2, cast(0 as double) as f3, cast(0 as bigint) as label from dual union all select cast(0 as double) as f0, cast(0 as double) as f1, cast(1 as double) as f2, cast(0 as double) as f3, cast(1 as bigint) as label from dual union all select
```

```
        cast(0 as double) as f0,        cast(0 as double) as f1,        cast(0 as double) as f2,
    cast(1 as double) as f3,        cast(1 as bigint) as label    from dual    union all        select
        cast(1 as double) as f0,        cast(0 as double) as f1,        cast(0 as double) as f2,
        cast(0 as double) as f3,        cast(0 as bigint) as label    from dual    union all
select        cast(0 as double) as f0,        cast(1 as double) as f1,        cast(0 as double)
as f2,        cast(0 as double) as f3,        cast(0 as bigint) as label    from dual ) a;
```

PAI 命令

训练：

```
drop offlinemodel if exists gbdt_ls_test_model; PAI -name gbdt      -project algo_public      -
DfeatureSplitValueMaxSize="500"    -DlossType="3"    -DrandSeed="0"    -DnewtonStep="1"
        -Dshrinkage="0.5"        -DmaxLeafCount="32"        -DlabelColName="label"        -
DinputTableName="gbdt_ls_test_input"      -DminLeafSampleCount="1"      -DsampleRatio="1"
      -DmaxDepth="10"      -DmetricType="0"      -DmodelName="gbdt_ls_test_model"      -
DfeatureRatio="1"    -Dp="1"    -Dtau="0.6"    -DtestRatio="0"    -DfeatureColNames="f0,f1,f2,f3"
    -DtreeCount="10"
```

预测：

```
drop table if exists gbdt_ls_test_prediction_result; PAI -name prediction      -project algo_public
        -DdetailColName="prediction_detail"        -DmodelName="gbdt_ls_test_model"        -
DitemDelimiter="," -DresultColName="prediction_result"        -Dlifecycle="28"        -
DoutputTableName="gbdt_ls_test_prediction_result"      -DscoreColName="prediction_score"
  -DkvDelimiter=":"      -DinputTableName="gbdt_ls_test_input"      -DenableSparse="false"      -
DappendColNames="label"
```

输入说明

表 147: gbdt_ls_test_input

f0	f1	f2	f3	label
1.0	0.0	0.0	0.0	0
0.0	0.0	1.0	0.0	1
0.0	0.0	0.0	1.0	1
0.0	1.0	0.0	0.0	0
1.0	0.0	0.0	0.0	0
0.0	1.0	0.0	0.0	0

输出说明

表 148: gbdt_ls_test_prediction_result

label	prediction_result	prediction_score	prediction_detail
0	NULL	0.0	{"label": 0}

label	prediction_result	prediction_score	prediction_detail
0	NULL	0.0	{"label": 0}
1	NULL	0.9990234375	{"label": 0.9990234375}
1	NULL	0.9990234375	{"label": 0.9990234375}
0	NULL	0.0	{"label": 0}
0	NULL	0.0	{"label": 0}

34.4.3.4.5 K 均值聚类

K 均值聚类是一种得到最广泛使用的聚类算法，把 n 个对象分为 k 个簇，使簇内具有较高的相似度。相似度的计算根据一个簇中对象的平均值来进行。它与处理混合正态分布的最大期望算法很相似，因为他们都试图找到数据中自然聚类的中心。算法首先随机地选择 k 个对象，每个对象初始地代表了一个簇的平均值或中心。对剩余的每个对象根据其与各个簇中心的距离，将它赋给最近的簇，然后重新计算每个簇的平均值。这个过程不断重复，直到准则函数收敛。它假设对象属性来自于空间向量，并且目标是使各个群组内部的均方误差总和最小。

参数设置



距离计算方式：支持欧式距离（euclidean，默认），绝度误差和（cityblock），夹角余弦（cosine）。

聚类数：单位整数型，默认是10。

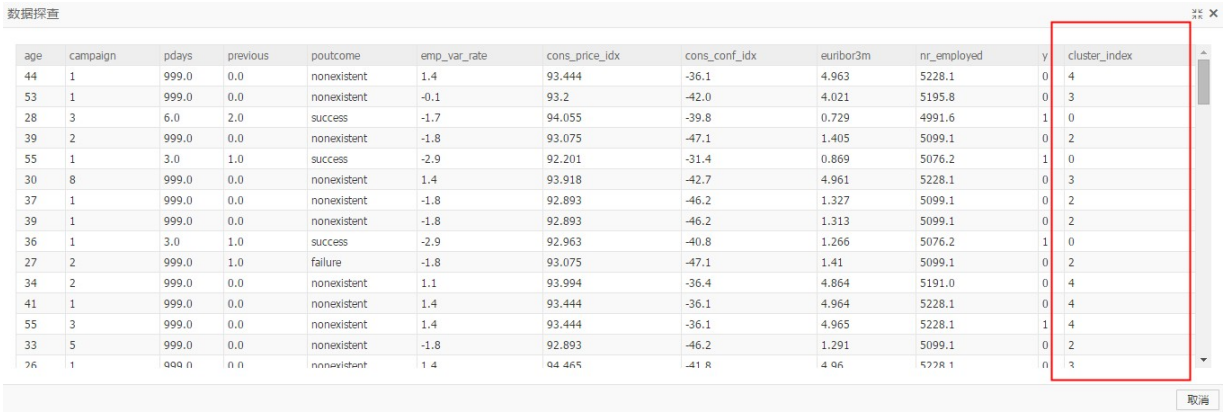
初始质心位置选择：支持 sample（随机选取，默认），topk（前K行），uniform（分布范围均匀随机生成），matrix（需指定表作为初始质心位置），kmpp（kmeans++初始化）。

最大迭代次数：计算最大迭代次数，默认为100。

最小迭代精度：计算最小迭代精度，默认为0.0。

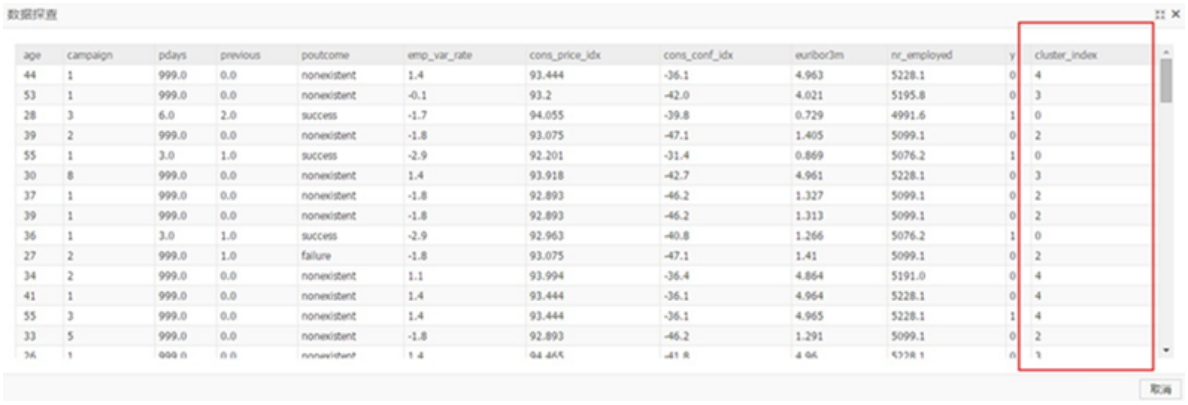
运行后，查看输出，可以查看聚类质心表（centerTable）。

聚类质心表（centerTable）



类质心表：列数等于输入表列数（有选中则等于选中列总数），行数等于聚类数，每行表示一个聚类质心位置。

1. 右击目标数据表，单击**查看数据**，可以查看结果聚类编号表（idxTablename）



- 聚类编号表：行数等于输入表总行数，每行的值表示输入表对应行表示的点的聚类编号。
- 会显示所有列名，并在最后追加一个分类标示字段。
- 其中显示的0,1,2,3为分类标识。
- 聚类质心表、聚类编号表及聚类个数表还可以通过 PAI 命令生成的表名在 IDE 里查看。

2. 注意事项

   若初始质心位置选择 matrix，初始质心表需要用户自己定义，定义的新表列和原始表相同，行数是聚类数。做表的时候要定好 k 个质心，可以用 sql、mr 采样，或者根据自己需要选其他的模式。

PAI 命令

```
PAI –name KMeans –project algo_public -DcenterCount="10" -DidxTableName="bank_data_index" -DdistanceType="euclidean" -DappendColsIndex="0,1,2,3,4,5,6,7,8,9,10" -DcenterTableName="pai_temp_3300_27298_3" -Dloop="100" -DclusterCountTableName="pai_temp_3300_27298_2" -
```

```
DinitCentersMethod="sample" -Daccuracy="0.0" -DinputTableName="bank_data" -
DselectedColNames="cons_conf_idx,emp_var_rate,euribor3m,pdays,previous";
```

- name：组件名字；
- project：（可选) 默认 project 是 algo_public。如果选择其他的 Project，需要指定 algo_public 下的算法包，否则报错。
- centerCount：聚类数，单位整数型。默认是10。
- idxTableName：输出聚类编号表。行数等于输入表总行数，每行的值表示输入表对应行表示的点的聚类编号。
- distanceType: （可选）距离测度方法。支持欧式距离（euclidean，），绝度误差和（cityblock），夹角余（cosine）。默认欧式距离（euclidean，）。
- appendColsIndex：（可选）输出表附加的 id 列名，默认输出表不附加 id 列。
- centerTableName：输出聚类质心表，列数等于输入表（有选中则等于选中列总数），行数等于聚类数，每行表示一个聚类质心位置。
- loop：（可选）计算最大迭代次数，整数型，默认为100。
- clusterCountTableNam
  e：输出聚类点总数表名，行数等于聚类数，每行聚类质心的类中所包含的聚类点的总数。
- initCentersMethod： （可选）初始质心位置选择，支持 sample（随机选取），topk（前K行），uniform（分布范围均匀随机生成），matrix（指定表作为初始质心位置），kmpp（kmeans++初始化）。默认 sample。
- accuracy：计算最小迭代精度，默认为0.0。
- inputTableName：输入表的名字。
- selectedColNames：输入表选择列，逗号分隔。只支持 double 型输入。
- initCenterTableName：（可选）输入起始 center 值的表格，当 initCentersMethod 为 matrix 时候必须输入，否则不用填。注意：若初始质心位置选择 matrix，初始质心表需要用户自己定义，定义的新表列和原始表相同，行数是聚类数。做表的时候要定好 k 个质心，可以用 sql，mr 采样，或者根据自己需要选其他的模式。

34.4.3.4.6 评估

34.4.3.4.6.1 混淆矩阵

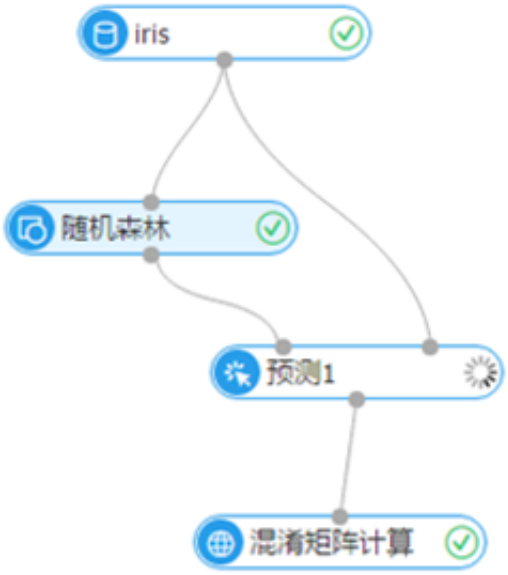
混淆矩阵（confusion matrix）是可视化工具，特别用于监督学习，在无监督学习一般叫做匹配矩阵，主要用于比较分类结果和实际测得值，可以把分类结果的精度显示在一个混淆矩阵里面。

1. 混淆矩阵参数如下图：



一般情况下采用默认沿用设置，用户也可自行选择目标列和预测概率列，预测概率列指预测组件生成预测目标列名。

2. 混淆矩阵组件上层节点只能为预测组件；只能分类模型才可进行混淆矩阵分析。



3. 右击**混淆矩阵组件 > 查看评估报告。**



PAI命令

```
PAI –name confusionmatrix –project algo_public -DoutputTableName="pai_temp_2954_24178_1" -DlabelColName="age" -DpredictionColName="prediction_result" -DinputTableName="pai_temp_2954_24176_1";
```

- name：组件名字。
- project：（可选）默认 project 是 algo_public。如果选择其他的 Project，需要指定 algo_public 下的算法包，否则报错。
- outputTableName：混淆矩阵结果表。
- labelColName：输入表的标签列列名。
- predictionColName：预测结果列。
- inputTableName：输入预测结果表名。

34.4.3.4.6.2 回归模型评估

基于预测结果和原始结果，评价回归算法模型的优劣，指标包括

SST、SSE、SSR、R2、R、MSE、RMSE、MAE、MAD、MAPE、count、yMean 和 predictMean。

PAI 命令

```
Pai –name regression_evaluation –project algo_public -DinputTableName=input_table -DyColName=y_col -DpredictionColName=prediction_col -DoutputTableName=output_table;
```

参数名称	参数描述	参数值可选项	默认值
inputTableName	必选，输入表的表名	-	-
inputTablePartitions	可选，输入表中指定参与计算的分区	-	输入表的所有partitions

参数名称	参数描述	参数值可选项	默认值
yColName	必选，输入表原始因变量列名，数值类型	-	-
predictionColName	必选，预测结果因变量列名，数值类型	-	-
outputTableName	必选，输出表表名	-	-
inputTablePartitions	可选，输入表选择的分区	-	-
lifecycle	可选，输出表的生命周期	-	默认不设置

输出结果

输出结果是一个 json ，json 各字段描述：

字段名称	描述
SST	总平方和
SSE	误差平方和
SSR	回归平方和
R2	判定系数
R	多重相关系数
MSE	均方误差
RMSE	均方根误差
MAE	平均绝对误差
MAD	平均误差
MAPE	平均绝对百分误差
count	行数
yMean	原始因变量的均值
predictionMean	预测结果的均值

34.4.3.4.6.3 多分类评估

算法参数

参数名称	参数描述	参数可选项	参数默认值
inputTableName	必选，输入表的表名	-	-

参数名称	参数描述	参数可选项	参数默认值
inputTablePartitions	可选，输入表中指定参与计算的分区	-	输入表的所有partitions
labelColName	必选，输入表原始label列名，数值类型	-	-
predictionColName	必选，预测结果label列名，数值类型	-	-
outputTableName	必选，输出表表名	-	-
predictionColName	可选，预测结果的概率列，格式如：{"A":0.2,"B":0.3}	-	-
lifecycle	可选，指定输出表的生命周期	正整数	没有生命周期

34.4.3.4.6.4 二分类评估

基于预测结果和原始结果，评价回归算法模型的优劣，指标包括 MSE、MAE、MAPE。

PAI 命令示例

```
pai –name evaluation -DinputTableName=input_table -DlabelColName=label_name -DpredictionColName=prediction_score -DoutputTableName=output_table;
```

算法参数

参数名称	参数描述	参数可选项	参数默认值
inputTableName	必选，输入表的表名	-	-
inputTablePartitions	可选，输入表中指定参与计算的分区	-	输入表的所有partitions
labelColName	必选，输入表原始label列名，数值类型	-	-
predictionColName	必选，预测结果label列名，数值类型	-	-
outputTableName	必选，输出表表名	-	-
lifecycle	可选，指定输出表的生命周期	正整数	没有生命周期

输出表

字段名称	描述
MSE	均方误差，衡量平均误差，可以评价数据的变化程度，MSE 越小，说明预测模型描述实验数据具有更好的精确度
MAE	平均绝对误差，衡量预测值与真实值之间平均相差多大
MAPE	平均绝对百分误差，衡量预测精度，值为百分比。举例：若 MAPE 值为15，则表示平均绝对百分误差为15%

34.4.3.4.6.5 聚类模型评估

基于原始数据和聚类模型，评价聚类模型的优劣，包含指标和图标。

PAI 命令

```
PAI -name cluster_evaluation -project algo_public -DinputTableName=pai_cluster_evaluation_test_input -DselectedColNames=f0,f3 -DmodelName=pai_kmeans_test_model -DoutputTableName=pai_ft_cluster_evaluation_out;
```

参数说明

表 149: 参数说明

参数说明	参数描述	取值范围	是否必选，默认值/行为
inputTableName	输入表	表名	必选
selectedColNames	输入表中用于评估的列名，以逗号分隔，必须与模型存储的特征名一致	列名	可选，默认值选择所有列
inputTablePartitions	输入表中指定哪些分区参与评估，格式为 name1=value1/name2=value2; 如果是指定多个分区， 中间用','分开		可选， 默认值选择所有分区
enableSparse	输入表数据是否为稀疏格式	true， false	可选， 默认值false

参数说明	参数描述	取值范围	是否必选，默认值/行为
itemDelimiter	当输入表数据为稀疏格式时，kv 间的分割符		可选， 默认值为空格
kvDelimiter	当输入表数据为稀疏格式时，key 和 value 的分割符		可选， 默认值冒号
modelName	输入聚类模型	模型名	必选
outputTableName	输出表	表名	必选
lifecycle	可选，指定输出表的生命周期	正整数	没有生命周期

评估公式

- Calinski-Harabasz 指标又称 VRC （ variance ratio criterion ）定义如下：

$$VRC_k = \frac{SS_B}{SS_W} \times \frac{(N-k)}{(k-1)},$$

其中 SS_B 是整个聚类间的方差； SS_W 是整个聚类内的方差；N 是记录总数，k 聚类中心点个数。

SS_B

定义如下：

$$SS_B = \sum_{i=1}^{k} n_i \|m_i - m\|^2,$$

k 聚类中心点个数，

m_i

聚类i的中心点, m 是输入数据的均值。

SS_W 定义如下：

$$SS_W = \sum_{i=1}^{k} \sum_{x \in c_i} \|x - m_i\|^2,$$

k 聚类中心点个数, x 是数据点； c_i 第i个聚类； m_i 聚类i的中心点。

具体示例

- 测试数据

```
create table if not exists pai_cluster_evaluation_test_input  as select * from ( select 1 as id, 1 as f0,2 as f3 from dual union all select 2 as id, 1 as f0,3 as f3 from dual union all select 3 as id, 1 as f0,4 as f3 from dual union all select 4 as id, 0 as f0,3 as f3 from dual union all select 5 as id, 0 as f0,4 as f3 from dual )tmp;
```

- 构造聚类模型

```
pai -name kmeans -project algo_public -DinputTableName=pai_cluster_evaluation_test_input -DselectedColNames=f0,f3 -DcenterCount=3 -Dloop=10 -Daccuracy=0.00001 -DdistanceType=euclidean -DinitCenterMethod=random -Dseed=1 -DmodelName=pai_kmeans_test_model -DidxTableName=pai_kmeans_test_idx
```

- PAI 命令示例

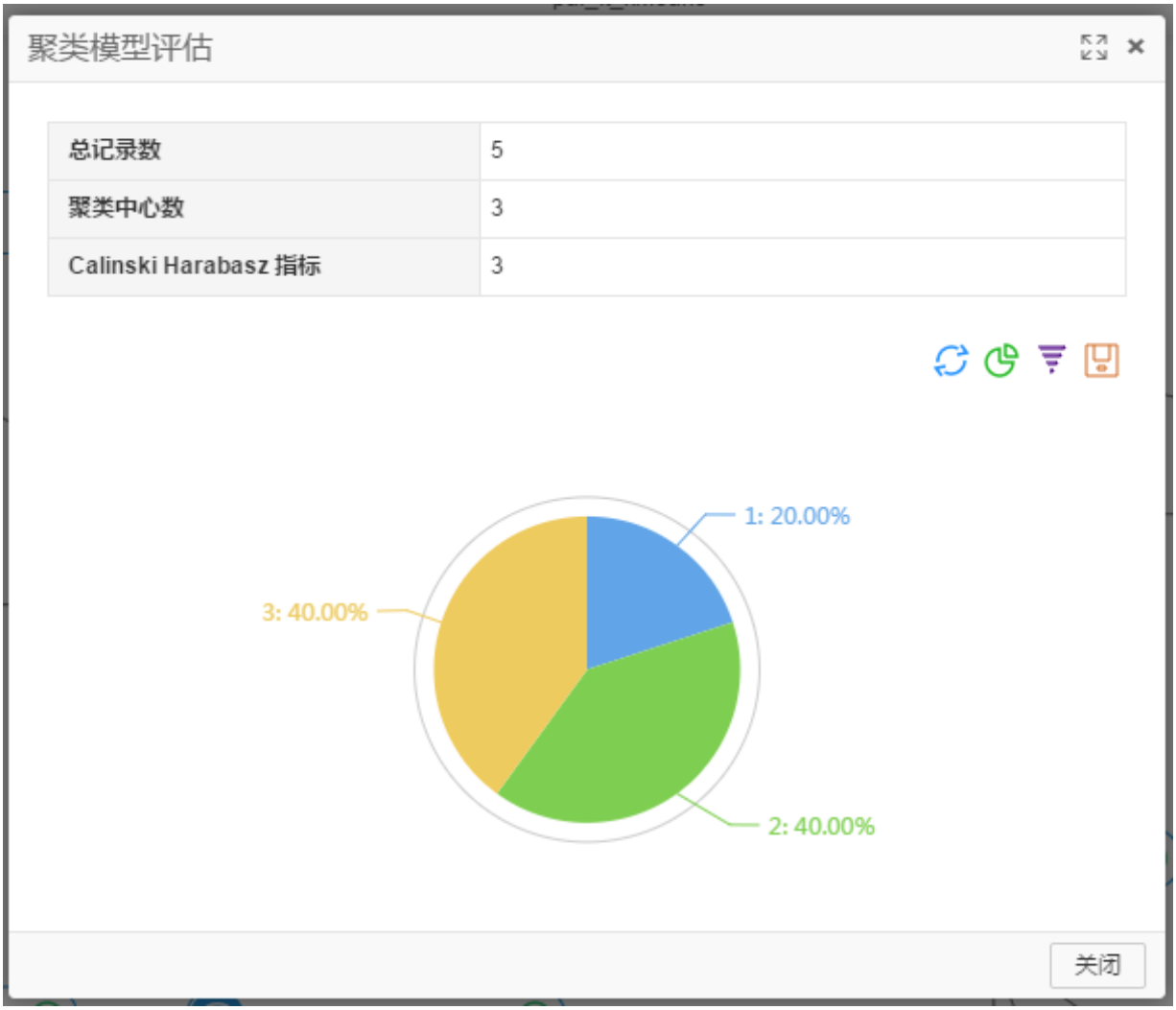
```
PAI -name cluster_evaluation -project algo_public -DinputTableName=pai_cluster_evaluation_test_input -DselectedColNames=f0,f3 -DmodelName=pai_kmeans_test_model -DoutputTableName=pai_ft_cluster_evaluation_out;
```

- 输出说明

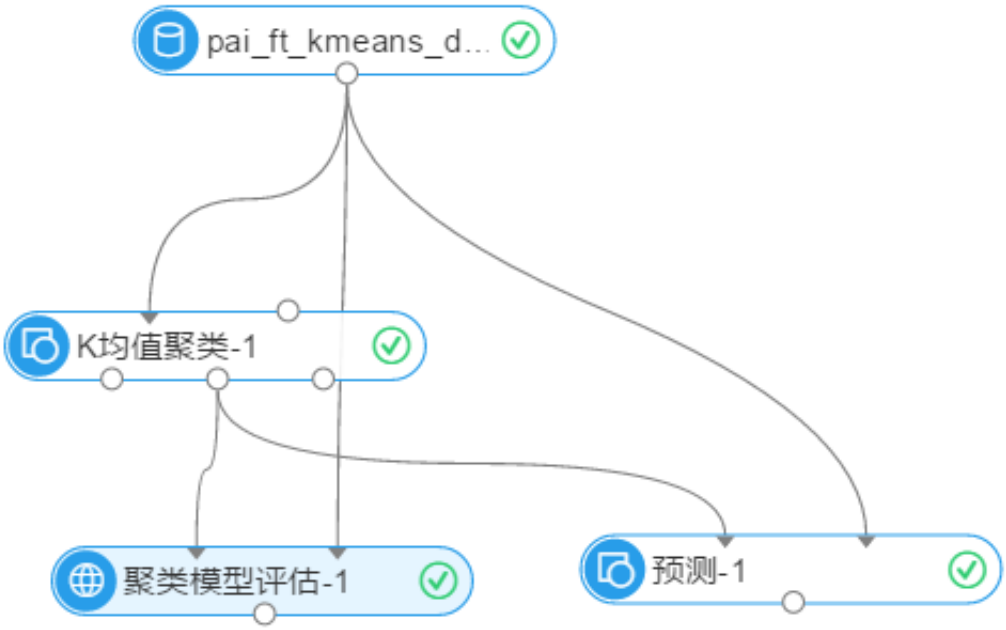
表 150: 输出表 outputTableName

column name	comment
count	总记录数
centerCount	聚类中心数
calinhara	Calinski Harabasz 指标
clusterCounts	各个聚类包含的点的数目

PaiWeb 展示


聚类模型评估
总记录数 5
聚类中心数 3
Calinski Harabasz 指标 3
1: 20.00%
2: 40.00%
3: 40.00%
关闭

PaiWeb-Pipeline

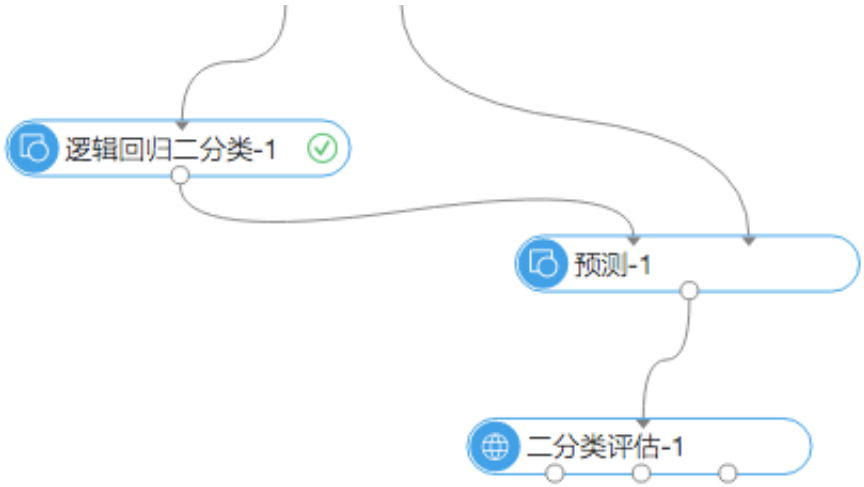


34.4.3.4.7 预测

预测组件是专门用于模型预测的组件，两个输入：训练模型和预测数据，输出为预测结果。传统的数据挖掘算法一般都采用该组件进行预测操作。

组件介绍

1. 链接所有组件。



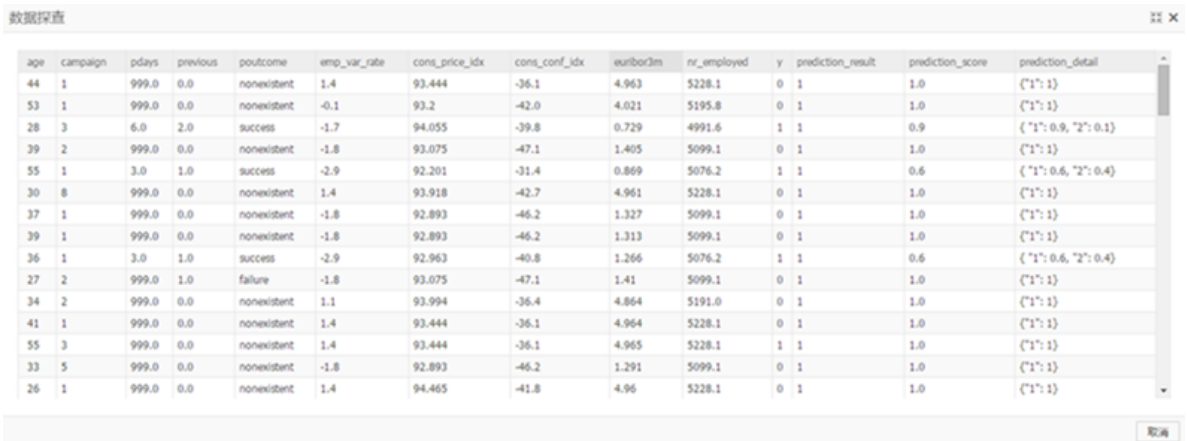
2. 设置字段。

   相关参数说明如下：

   - 特征列：用于预测的数据输入特征，默认为全选；
   - 原样输出列：在输出结果中，本参数所选择的字段将一同输出；
   - 输出结果列名、输出分数列名、输出详细列名均取默认值即可；
   - 若数据是稀疏格式，需要选择稀疏矩阵的特征列，稀疏的格式：k1:v1,k2:v2。



3. 执行结束后，右键单击**预测组件** > **查看数据** 。



预测的数据中，会新增三列：

- predict_result：预测结果列；
- predict_score：预测结果概率得分；仅模型为二分类时有效；
- predict_detail：每个类别的预测结果；仅模型为二分类时有效。

PAI 命令

```
PAI –name Prediction –project algo_public -DdetailColName="prediction_detail" -DsplitCharacteristic="2" -DappendColNames="age,campaign,pdays,previous,poutcome,emp_var_rate,cons_price_idx,cons_conf_idx,e -DmodelName="xlab_m_random_forests_6036" -DresultColName="prediction_result" -DoutputTableName="pai_temp_675_6048_1" -DscoreColName="prediction_score" -DinputTableName="bank_data";
```

- name：组件名字；
- project：project 名字。用于指定算法所在空间，系统默认是 algo_public，用户自己更改后系统会报错；
- detailColName：（可选）详细信息列。输出表中 detail 列名，默认 prediction_detail；
- splitCharacteristic：（可选）是否二分类。模型对应的分类类型，1代表二分类；2代表多分类。
- appendColNames：（可选）向输出表追加列。输出表中需要增加的预测输入表的列；
- modelName：随机森林模型；
- resultColName：（可选）结果列名。输出表中 Result 列名，默认 prediction_result；
- outputTableName：预测输出表的表名；
- scoreColName：（可选）评分列名。输出表中 Score 列名，默认 prediction_score；
- inputTableName：预测输入表的表名。

34.4.3.4.8 工具

34.4.3.4.8.1 SQL脚本

您可以通过 SQL 脚本编辑器编写 SQL 语句。

1. 向画布拖入**SQL 脚本**组件。
2. 连接输入表和 **SQL 脚本**组件后，单击**SQL 脚本**，出现信息框。



3. 在相应位置写入想要实现的功能的 SQL 脚本。



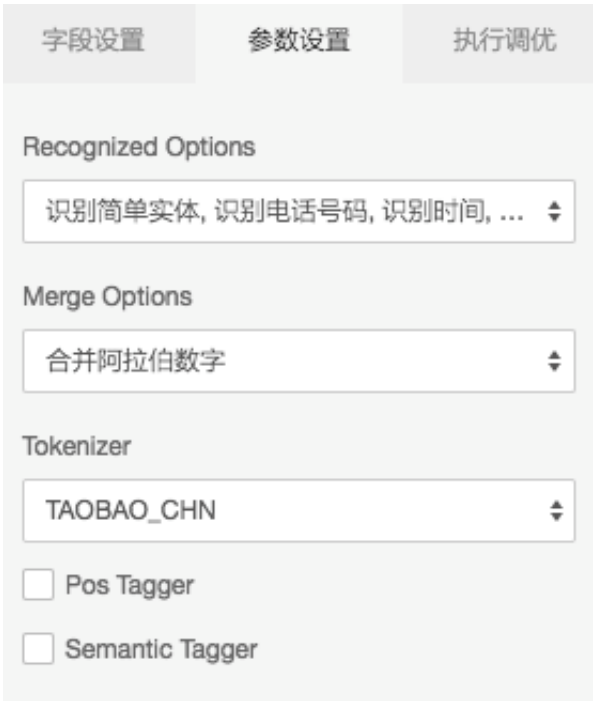
- SQL 脚本支持1-2个输入，1个输出。
- 用户只能写入单条 SQL 语句。
- 输入数据已自动映射成 t1~t2，用户可直接使用。使用时直接调用 ${t1}、${t2}，不用再写入原表名。

- 示例的 SQL 脚本用于实现对输入表进行行数统计。

34.4.3.4.9 文本分析

34.4.3.4.9.1 阿里分词

基于 AliWS（Alibaba Word Segmenter的简称）词法分析系统，对指定列对应的文章内容进行分词，分词后的各个词语间以空格作为分隔符，若用户指定了词性标注或语义标注相关参数，则会将分词结果、词性标注结果和语义标注结果一同输出，其中词性标注分隔符为"/"，语义标注分隔符为"|"。目前仅支持中文淘宝分词和互联网分词。



- 可以选择词库是淘宝分词或是互联网分词，推荐淘宝分词。
- PosTagger

参数设置

- 分词算法：CRF、UNIGRAM。
- 识别选项：分词中，是否识别特殊意义的名词。
- 合并选项：将具有特殊领域的名词作为整体，不进行切分操作。
- 字符串切分长度：>=0，数字串按指定长度进行截断作为检索单元；默认为0，不对数字串进行长度切分。
- 使用词频纠错：是否使用纠错词典。
- 标注词性：输出结果中，标注词性。

实例介绍

输入包含两列的表，第一列是文档 ID，第二列是文档内容 text。





PAI 命令

```
pai -name split_word -project algo_public -DinputTableName=doc_test -DselectedColNames=content1,content2 -DoutputTableName=doc_test_split_word -DinputTablePartitions="region=cctv_news" -DoutputTablePartition="region=news" -Dtokenizer=TAOBAO_CHN -DenableDfa=true -DenablePersonNameTagger=false -DenableOrgnizationTagger=false -DenablePosTagger=false -DenableTelephoneRetrievalUnit=true -DenableTimeRetrievalUnit=true -DenableDateRetrievalUnit=true -DenableNumberLetterRetrievalUnit=true -DenableChnNumMerge=false -DenableNumMerge=true -DenableChnTimeMerge=false -DenableChnDateMerge=false -DenableSemanticTagger=true
```

算法参数

参数 Key 名称	参数描述	参数 value 可选项	默认值
inputTableName	输入表名	-	-
selectedColNames	输入表中用于分词的列名	可指定多列，列名间用逗号(,)间隔	-
outputTableName	输出表名	-	-
inputTablePartitions	输入表中指定参与分词的分区名，格式为: partition_name=value。	-	输入表的所有partition

参数 Key 名称	参数描述	参数 value 可选项	默认值
	如果是多级格式为name1=value1/name2=value2；如果是指定多个分区，中间用','分开		
outputTablePartition	指定输出表的分区	-	输出表不进行分区
tokenizer	分类器类型	TAOBAO_CHN,INTERNET_CHN	默认为TAOBAO_CHN，淘宝中文分词；INTERNET_CHN，互联网中文分词
enableDfa	简单实体识别	true,false	true
enablePersonNameTagger	人名识别	true,false	false
enableOrgnizationTagger	机构名识别	true,false	false
enablePosTagger	是否词性标注	true,false	false
enableTelephoneRetrievalUnit	检索单元配置 - 电话号码识别	true,false	true
enableTimeRetrievalUnit	检索单元配置 - 时间号码识别	true,false	true
enableDateRetrievalUnit	检索单元配置 - 日期号码识别	true,false	true
enableNumberLetterRetrievalUnit	检索单元配置 - 数字字母识别	true,false	true
enableChnNumMerge	中文数字合并为一个检索单元	true,false	false
enableNumMerge	普通数字合并为一个检索单元	true,false	true
enableChnTimeMerge	中文时间合并为一个语意单元	true,false	false
enableChnDateMerge	中文日期合并为一个语意单元	true,false	false

参数 Key 名称	参数描述	参数 value 可选项	默认值
enableSemanticTagger	是否语义标准	true,false	false

34.4.3.4.9.2 词频统计

功能介绍

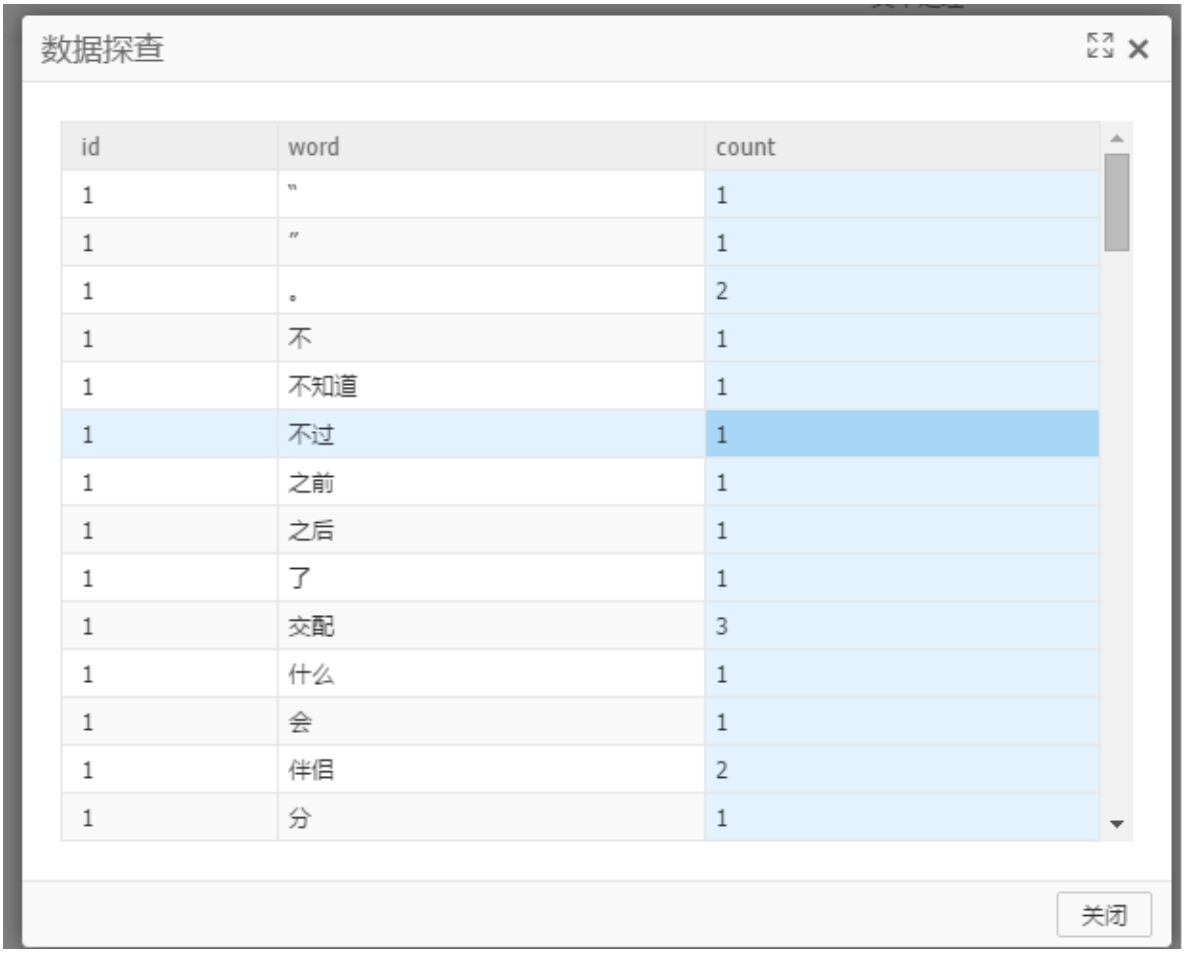
在对文章进行分词的基础上，按行保序输出对应文章 ID列（docId）对应文章的词，统计指定文章 ID 列（docId）对应文章内容（docContent）的词频。

参数设置

输入参数：经过分词组件生成两列（文档 ID 列和分词后的文档内容列）。

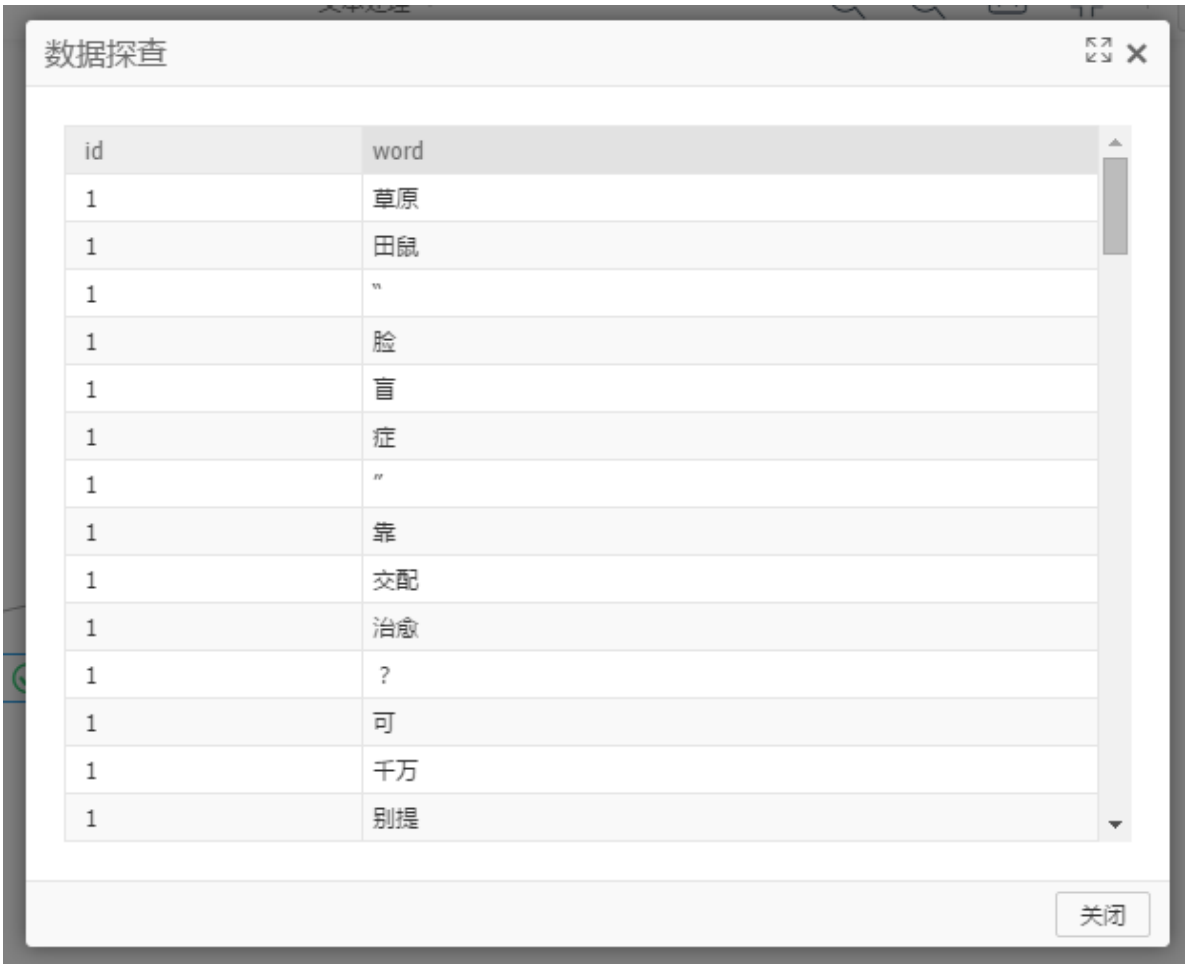
两个输出参数：

1. 第一个输出端：输出表包含三个字段（id、word、count）



count：统计每个文档中，对应 word 词汇出现的次数。

2. 第二个输出端：输出包含两个字段（id、word）。



本端口输出表按词语在文章中出现的顺序依次输出，没有统计词语的出现次数，因此同一文档中某个词汇可能出现多条记录。 包输出表格式主要用于兼容 Word2Vec 组件使用。

实例

采用阿里分词实例数据中，将分别将输出表的两个列作为词频统计的输入参数：

- 选择文档ID列：ID。
- 选择文档内容列：text 经过词频统计运算后，生成的结果 见本组件中第一个输出参数展示图。

PAI 命令

```
pai -name doc_word_stat -project algo_public -DinputTableName=doc_test_split_word -DdocId=id -DdocContent=content -DoutputTableNameMulti=doc_test_stat_multi -DoutputTableNameTriple=doc_test_stat_triple -DinputTablePartitions="region=cctv_news"
```

算法参数

参数 Key 名称	参数描述	参数 value 可选项	默认值
inputTableName	输入表名	-	-

参数 Key 名称	参数描述	参数 value 可选项	默认值
docId	标识文章id的列名	仅可指定一列	-
docContent	标识文章内容的列名	仅可指定一列	-
outputTableNameMulti	输出保序词语表名	-	-
outputTableNameTriple	输出词频统计表名	-	-
inputTablePartitions	输入表中指定参与分词的分区名，格式为:partition_name=value。如果是多级格式为name1=value1/name2=value2；如果是指定多个分区，中间用','分开	-	输入表的所有partition

- 其中参数 outputTableNameMulti 指定的表是 docId 列及 docId 列对应的文章内容（docContent）完成分词后，按各个词语在文章中出现的顺序依次输出。
- 参数 outputTableNameTriple 指定的表输出 docId 列及 docId 列对应的文章内容（docContent）完成分词后，统计得到的各个词语及其在文章中出现的次数。

34.4.3.4.9.3 TF-IDF

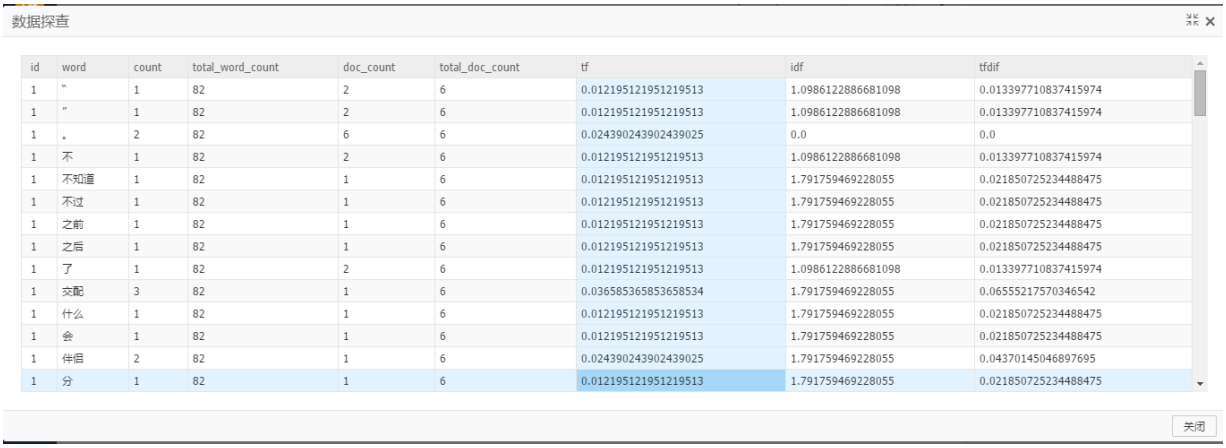
- TF-IDF（term frequency–inverse document frequency）是一种用于资讯检索与文本挖掘的常用加权技术。TF-IDF 是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。 字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。TF-IDF 加权的各种形式常被搜索引擎应用，作为文件与用户查询之间相关程度的度量或评级。
- 详细介绍，请参考维基百科*tf-idf*。
- 本组件是词频统计输出的基础上，计算各个 word 对于各个文章的 tfidf 值。

实例

以词频统计组件实例中的输出表作为 TF-IDF 组件的输入表，对应的参数设置如下：

- 选择文档 ID 列： id
- 选择单词列：word
- 选择单词计数列：count

输出表有9列：docid，word，word_count（当前word在当前doc中出现次数），total_word_count（当前doc中总word数），doc_count（包含当前word的总doc数），total_doc_count（全部doc数），tf，idf，tfidf 结果如下：


数据探查

id	word	count	total_word_count	doc_count	total_doc_count	tf	idf	tfdif
1	"	1	82	2	6	0.012195121951219513	1.0986122886681098	0.013397710837415974
1	"	1	82	2	6	0.012195121951219513	1.0986122886681098	0.013397710837415974
1	。	2	82	6	6	0.024390243902439025	0.0	0.0
1	不	1	82	2	6	0.012195121951219513	1.0986122886681098	0.013397710837415974
1	不知道	1	82	1	6	0.012195121951219513	1.791759469228055	0.021850725234488475
1	不过	1	82	1	6	0.012195121951219513	1.791759469228055	0.021850725234488475
1	之前	1	82	1	6	0.012195121951219513	1.791759469228055	0.021850725234488475
1	之后	1	82	1	6	0.012195121951219513	1.791759469228055	0.021850725234488475
1	了	1	82	2	6	0.012195121951219513	1.0986122886681098	0.013397710837415974
1	交配	3	82	1	6	0.036585365853658534	1.791759469228055	0.06555217570346542
1	什么	1	82	1	6	0.012195121951219513	1.791759469228055	0.021850725234488475
1	会	1	82	1	6	0.012195121951219513	1.791759469228055	0.021850725234488475
1	伴侣	2	82	1	6	0.024390243902439025	1.791759469228055	0.04370145046897695
1	分	1	82	1	6	0.012195121951219513	1.791759469228055	0.021850725234488475

关闭

PAI 命令

```
pai -name tfidf -project algo_public -DinputTableName=rgdoc_split_triple_out -DdocIdCol=id
    -DwordCol=word -DcountCol=count -DoutputTableName=rg_tfidf_out;
```

算法参数

参数 Key 名称	参数描述	取值范围	是否必选，默认值/行为
inputTableName	输入表名	表名	必选
inputTablePartitions	输入表中指定参与分词的分区名	格式为: partition_name=value。如果是多级格式为name1=value1/name2=value2；如果是指定多个分区，中间用','分开	非必选，默认行为：输入表的所有partition
docIdCol	标识文章id的列名	仅可指定一列	必选
wordCol	word列名	仅可指定一列	必选
countCol	count列名	仅可指定一列	必选
outputTableName	输出表名	表名	必选
outputTablePartition	输出表partition	分区名	非必选，默认行为：输出表为非partition表

34.4.3.4.9.4 PLDA

- 主题模型，返回文档对应的主题。
- LDA（Latent Dirichlet allocation），是一种主题模型，它可以将文档集中每篇文档的主题按照概率分布的形式给出。同时它是一种无监督学习算法，在训练时不需要手工标注的训练集，需要的仅仅是文档集以及指定主题的数量k即可。 LDA 首先由 David M. Blei、Andrew Y. Ng 和 Michael I. Jordan 于2003年提出，目前在文本挖掘领域包括文本主题识别、文本分类以及文本相似度计算方面都有应用。

参数设置

- 主题个数: 设置 LDA 的输出的主题个数。
- Alpha：P(z/d)的先验狄利克雷分布的参数。
- beta：P(w/z)的先验狄利克雷分布的参数。
- burn In：burn in 迭代次数，必须小于总迭代次数，默认值：100。
- 总迭代次数：正整数 | 非必选，默认值：150。



说明: z是主题，w是词，d是文档。

输入/输出设置

输入：数据必须为稀疏矩阵的格式（格式见数据格式说明章节）。可以通过三元组转 KV 组件进行转换。

输入格式如下：

```
| id          | features |
+-------------+----------+
| 2           | 38:3.0,39:1.0,40:3.0,41:1.0,42:1.0,43:2.0,44:1.0,45:1.0,46:1.0,47:1.0,48:1.0,49:2.0,50:1.0,51:1.0,52:1.0,53:1.0,54:1.0,55:1.0,5
6:1.0,57:1.0,58:1.0,59:1.0,60:1.0,61:1.0,62:1.0,63:1.0,64:1.0,65:1.0,66:1.0,67:1.0,68:1.0,69:1.0,70:1.0,71:1.0,72:1.0,73:1.0,74:1.0,75:1.0,76:
1.0,77:2.0 |
| 1           | 0:1.0,1:2.0,3:1.0,4:1.0,5:1.0,6:1.0,7:1.0,8:1.0,9:1.0,10:1.0,11:1.0,12:1.0,13:1.0,14:2.0,15:1.0,16:1.0,17:1.0,18:1.0,19:1.0,20:
1.0,21:1.0,22:1.0,23:1.0,24:1.0,25:1.0,26:1.0,27:1.0,28:1.0,29:1.0,30:1.0,31:1.0,32:1.0,33:1.0,34:1.0,35:1.0,36:2.0,39:2.0,77:3.0 |
+-------------+----------+
```

- 第一列：docid。
- 第二列：单词及词频的 kv 数据。

输出依次为： ·topic-word 频率贡献表，单词|主题输出表，主题|单词输出表，文档|主题输出表，主题|文档输出表，主题输出表。

topic-word 频率贡献表的输出格式如下：

wordid	topic_0	topic_1
0	1	0
1	2	0
2	0	0
3	1	0
4	1	0
5	1	0
6	1	0
7	1	0
8	0	1
9	1	0
10	1	0
11	1	0

PAI 命令

```
pai -name PLDA -project algo_public -DinputTableName=lda_input –DtopicNum=10
 -topicWordTableName=lda_output;
```

算法参数

参数 Key 名称	参数描述	取值范围	是否必选，默认值/行为
inputTableName	输入表名	表名	必选
inputTablePartitions	输入表中指定参与分词的分区名	格式为: partition_name=value。如果是多级格式为name1=value1/name2=value2；如果是指定多个分区，中间用','分开	非必选，默认值：输入表的所有partition
selectedColNames	输入表中用于LDA的列名	列名，逗号分隔	非必选，默认值：输入表中所有的列名
topicNum	topic的数量	[2, 500]	必选
kvDelimiter	key和value间的分分隔符	空格、逗号、冒号	非必选，默认值：冒号
itemDelimiter	key和key间的分隔符	空格、逗号、冒号	非必选，默认值：空格
alpha	P(z/d)的先验狄利克雷分布的参数	(0, ∞)	非必选，默认值：0.1
beta	P(w/z)的先验狄利克雷分布的参数	(0, ∞)	非必选，默认值：0.01

参数 Key 名称	参数描述	取值范围	是否必选，默认值/行为
topicWordTableName	topic-word频率贡献表	表名	必选
pwzTableName	P(w/z)输出表	表名	非必选，默认行为：不输出P(w/z)表
pzwTableName	P(z/w)输出表	表名	非必选，默认行为：不输出P(z/w)表
pdzTableName	P(d/z)输出表	表名	非必选，默认行为：不输出P(d/z)表
pzdTableName	P(z/d)输出表	表名	非必选，默认行为：不输出P(z/d)表
pzTableName	P(z)输出表	表名	非必选，默认行为：不输出P(z)表
burnInIterations	burn in 迭代次数	正整数	非必选，必须小于totalIterations，默认值：100
totalIterations	迭代次数	正整数	非必选，默认值：150

34.4.3.4.9.5 Word2Vec

功能介绍

- Word2Vec 是 Google 在 2013 年开源的一个将词表转为向量的算法，其利用神经网络，可以通过训练，将词映射到 K 维度空间向量，甚至对于表示词的向量进行操作还能和语义相对应，由于其简单和高效引起了很多人的关注。
- Google Word2Vec 的工具包相关链接：*https://code.google.com/p/word2vec/*。

参数设置

算法参数：

- 单词的特征维度：建议 0-1000
- 向下采样阈值：建议值为1e-3-1e-5
- 输入：单词列和词汇表
- 输出：输出词向量表和词汇表

PAI 命令

```
pai -name Word2Vec -project algo_public -DinputTableName=w2v_input
 –DwordColName=word -DoutputTableName=w2v_output;
```

算法参数

参数 Key 名称	参数描述	取值范围	是否必选，默认值/行为
inputTableName	输入表名	表名	必选
inputTablePartitions	输入表中指定参与分词的分区名	格式为: partition_name=value。如果是多级格式为name1=value1/name2=value2；如果是指定多个分区，中间用','分开	非必选，默认值：输入表的所有partition
wordColName	单词列名，单词列中每行为一个单词，语料中换行符用</s>表示	列名	必选
inVocabularyTableName	输入词表，该表为inputTableName的wordcount输出	表名	非必选，默认行为：程序内部会对输出表做wordcount
inVocabularyPartitions	输入词表分区	分区名	非必选，默认值：inVocabularyTableName对应表的所有分区
layerSize	单词的特征维度	0-1000	非必选，默认值：100
cbow	语言模型	值为1：表示cbow模型，值为0：skip-gram模型	非必选，默认值：0
window	单词窗口大小	正整数	非必选，默认值：5
minCount	截断的最小词频	正整数	非必选：默认值：5
hs	是否采用HIERARCHICAL SOFTMAX	值为1：表示采用，值为0：不采用	非必选，默认值：1
negative	NEGATIVE SAMPLING	值为0不可用，建议值5-10	非必选，默认值：0

参数 Key 名称	参数描述	取值范围	是否必选，默认值/行为
sample	向下采样阈值	值为小于等于0：不采用，建议值为1e-3-1e-5	非必选，默认值：0
alpha	开始学习速率	大于0	非必选，默认值：0.025
iterTrain	训练的迭代次数	大于等于1	非必选，默认值：1
randomWindow	window是否随机	值为1，表示大小在1~5间随机；值为0，表示不随机，其值由window参数指定	非必选，默认值：1
outVocabularyTableName	输出词表	表名	非必选，默认行为：不输出 ‘输出词表’
outVocabularyPartition	输出词表分区	分区名	非必选，默认行为：输出词表为非分区表
outputTableName	输出表	表名	必选
outputPartition	输出表分区信息	分区名	非必选，默认行为：输出表为非分区表

34.4.3.4.9.6 关键词提取

关键词抽取是自然语言处理中的重要技术之一，具体是指从文本里面把跟这篇文章意义最相关的一些词抽取出来。

本算法基于 TextRank，它受到网页之间关系 PageRank 算法启发，利用局部词汇之间关系（共现窗口）构建网络，计算词的重要性，选取权重大的做为关键词。

PAI 命令

```
PAI -name KeywordsExtraction -DinputTableName=maple_test_keywords_basic_input
    -DdocIdCol=docid -DdocContent=word -DoututTableName=maple_test_keywords_
basic_output -DtopN=19;
```

参数说明

参数名称	参数描述	取值范围	是否必选，默认值
inputTableName	输入表	表名	必选

参数名称	参数描述	取值范围	是否必选，默认值
inputTablePartitions	输入表中指定哪些分区参与训练， 格式为：Partition_name=value。如果是多级格式为name1=value1/name2=value2；如果是指定多个分区， 中间用英文逗号（,）分开。	-	可选， 默认值选择所有分区
outputTableName	输出表名	必选	-
docIdCol	标识文章id的列名，仅可指定一列	必选	-
docContent	Word列,仅可指定一列	必选	-
topN	输出前多少个关键词，当关键词个数小于全部词个数时，全部输出	可选5	-
windowSize	TextRank算法的窗口大小	可选2	-
dumpingFactor	TextRank算法的阻尼系数	可选0.85	-
maxIter	TextRank算法的最大迭代次数	可选100	-
epsilon	TextRank算法的收敛残差阈值	可选0.000001	-
lifecycle	可选，指定输出表的生命周期	正整数	没有生命周期
coreNum	节点个数	与参数memSizePerCore配对使用，正整数，范围[1, 9999]	可选， 默认自动计算
memSizePerCore	单个节点内存大小，单位M	正整数，范围[1024, 64*1024]	可选， 默认自动计算

具体示例

数据生成输入表需采用空格分词，并过滤掉停用词（类似**的**、**地**、**得**、**了**、**个**）和所有的标点符号。

docid: string	word: string
doc0	翼身融合 飞机 是 未来 航空 领域 发展 一个 新 方向 国内外 诸多 研究 机构 已经 开展 对翼身融合 飞机 研究 而 其 全自动 外形 优化 算法 已 成为 新 研究 热点 国内 外 现有 成果 基础 之上 分析 比较 常用 建模 求解 平台 使用 方式 及 特点 设计 编写 翼身融合 飞机 外形 优化 几何 建模 网格 划分 流场 求解 外形 优化 模块 比 较 不同 算法 间 优劣 实现 翼身融合 飞机 概念设计 中 外形 优化 几何 建模 及 网格 生成 模块 实现 基于 超限 插值 网格 生成 算法 基于 样条 曲线 建模 方法 流场 求解 模块 包括 有限 差分 求解器 有限元 求解器和面元法 求解器 其中 有限 差分 求解器 主要 包括 基于 有限 差分法 势流 数学 建模 基于 笛卡尔 网格 变 步长 差分 格式 推导 笛卡尔 网格 生成 索引 算法 基于 笛卡尔 网格 诺 依曼 边界条件 表达 形式 推导 实现 基于 有限 差分 求解器 二维 翼型 气动 参数 计算 算例 有限元 求解器 主要 包括 基于 变分 原理 势流 有限元 理论 建模 二维 有限元 库塔 条件 表达式 推导 基于 最小 二乘 速度 求解 算法 设计 基于 Gmsh 二维 带尾迹 翼型 空间 网格 生成器 开发 实现 基于 有限元 求解器 二维 翼型 气动 参数 计算 算例 面元法 求解器 主要 包括 基于 面元法 势流 理论 建模 自动 尾迹 生成 算法 设计 基于 面元法 三维 翼身融合 体 流场 求解器 开发 基于 布拉 修斯 平板 解 阻力 估算 算法 设计 求解器 Fortran 语言 上 移 植 Python 和 Fortran 代码 混编 基于 OpenMP 和 CUDA 并行 加速 算法 设计 与 开发 实现 基于 面元法 求解器 三维 翼身融合 体 气动 参数 计算 算例 外形 优化 模块 实 现了 基于 自由 形状 变形 网格 变形 算法 遗传算法 差分 进化 算法 飞机 表面积 计算 算法 基于 矩 积分 飞 机 体积 计算 算法 开发 基于 VTK 数据 可视化 格式 工具

PAI 命令

```
PAI -name KeywordsExtraction -DinputTableName=maple_test_keywords_basic_input
    -DdocIdCol=docid -DdocContent=word -DoututTableName=maple_test_keywords_
basic_output -DtopN=19;
```

输入/输出说明

- 输出表

docid	keywords	weight
doc0	基于	0.041306752223538405
doc0	算法	0.03089845626854151
doc0	建模	0.021782865850562882
doc0	网格	0.020669749212693957
doc0	求解器	0.020245609506360847
doc0	飞机	0.019850761705313365
doc0	研究	0.014193732541852615
doc0	有限元	0.013831122054200538
doc0	求解	0.012924593244133104
doc0	模块	0.01280216562287212
doc0	推导	0.011907588923852495
doc0	外形	0.011505456605632607
doc0	差分	0.011477831662367547
doc0	势流	0.010969269350293957
doc0	设计	0.010830986516637251
doc0	实现	0.010747536556701583
doc0	二维	0.010695570768457084
doc0	开发	0.010527342662670088
doc0	新	0.010096978306668461

34.4.3.4.9.7 文本摘要

所谓自动文摘就是利用计算机自动地从原始文献中提取文摘，文摘是全面准确地反映某一文献中心内容地简单连贯的短文。本算法基于 TextRank，通过提取文档中已存在的句子形成摘要。

PAI 命令

```
PAI -name TextSummarization -project algo_public –DinputTableName="test_input" -DoutputTableName="test_output" -DdocIdCol="doc_id" -DsentenceCol="sentence" -DtopN=2 -Dlifecycle=30;
```

参数说明

参数名称	参数描述	参数可选项	默认值
inputTableName	必选，输入表名	-	-
inputTablePartitions	可选，输入表中指定参与计算的分区	-	输入表的所有partition
outputTableName	必选，输出表名	-	-
docIdCol	必选，标识文章id的列名	-	-
sentenceCol	必选，句子列，仅可指定一列	-	-
topN	可选，输出前多个关键句	-	3
similarityType	可选，句子相似度的计算方法	lcs_sim, levenshtein_sim, cosine, ssk	lcs_sim
lambda	可选，匹配字符串的权重，ssk中可用	(0, 1)	0.5
k	可选，子串的长度，ssk和cosine中可用	(0, 100)	2
dampingFactor	可选，阻尼系数	(0, 1)	0.85
maxIter	可选，最大迭代次数	[1, +]	100
epsilon	可选，收敛系数	(0, 正无穷)	0.000001
lifecycle	可选，输入出表的生命周期	正整数	无生命周期
coreNum	可选，参与计算的核心数	正整数	系统自动计算
memSizePerCore	可选，每个核心需要的内存	正整数	系统自动计算

句子相似度的可选项如下：（符号说明：A，B表示两个字符串，len(A)表示字符串A的长度）：

- lcs_sim: 公式 1.0 - (最长公共子串(Longest Common Subsequence)的长度)/max(len(A), len(B))
- levenshtein_sim: 公式 1.0 - (编辑距离(Levenshtein Distance)) / max(len(A), len(B))
- cosine: 参考 Lodhi, Huma; Saunders, Craig; Shawe-Taylor, John; Cristianini, Nello; Watkins, Chris (2002). "Text classification using string kernels". Journal of Machine Learning Research: 419–444
- ssk: 参考 Leslie, C.; Eskin, E.; Noble, W.S. (2002), The spectrum kernel: A string kernel for SVM protein classification 7, pp. 566–575

输出格式说明

输出表为两列，分别是 doc_id, abstract。示例如下：

doc_id	abstract
1000894	早在2008年，上交所便发布了上市公司社会责任披露相关指引，强制要求三类公司披露社会责任报告，同时鼓励其他有条件的上市公司进行自愿披露。统计显示，2012年，沪市上市公司共计379家披露社会责任报告，包括强制披露公司305家和自愿披露公司74家，合计占沪市全部上市公司的40%。胡汝银表示，下一步上交所将探索扩大社会责任报告的披露范围，修订细化有关社会责任报告披露的指引，并鼓励更多的机构推进社会责任产品创新。

34.4.3.4.9.8 三元组转 kv

给定三元组（row,col,value）类型为 XXD 或 XXL，X 表示任意类型，D 表示 Double，L 表示 bigint，转成kv 格式（row,[col_id:value]），其中 row 和 value 类型和原始输入数据一致，col_id 类型是 bigint，并给出 col 的索引表映射到 col_id。

PAI 命令

```
PAI -name triple_to_kv -project algo_public -DinputTableName=test_data
-DoutputTableName=test_kv_out -DindexOutputTableName=test_index_out -
DidColName=id -DkeyColName=word -DvalueColName=count -DinputTablePartition
```

```
s=ds=test1 -DindexInputTableName=test_index_input -DindexInputKeyColName=word -DindexInputKeyIdColName=word_id -DkvDelimiter=: -DpairDelimiter=; -Dlifecycle=3
```

参数说明

参数名称	参数描述	取值范围	是否必选，默认值
inputTableName	输入表	-	必选，不能为空表
idColName	转成kv表时保持不变的列名	-	必选
keyColName	kv中的key	-	必选
valueColName	kv中的value	-	必选
outputTableName	输出kv表	-	必选
indexOutputTableName	输出key的索引表	-	必选
indexInputTableName	输入索引表	-	可选，“”，不能是空表，可以只有部分key的索引
indexInputKeyColName	输入索引表key的列名	-	可选，“”，输入indexInputTableName时必选此项
indexInputKeyIdColName	输入索引表key索引号的列名	-	可选，“”，输入indexInputTableName时必选此项
inputTablePartitions	输入表分区	-	可选，""，只能输入单个分区
kvDelimiter	key和value之间分隔符	-	可选，默认":"
pairDelimiter	kv对之间分隔符	-	可选，默认";"
lifecycle	可选，输出结果表的生命周期	-	不设生命周期
coreNum	可选，指定instance的总数	-	-1
memSizePerCore	可选，指定memory大小，范围在100~64*1024之间	-	-1

具体示例

数据生成：

```
drop table if exists triple2kv_test_input;  create table triple2kv_test_input as  select  *  from  (  select '01' as id, 'a' as word, 10 as count from dual  union all  select '01' as id, 'b' as word, 20 as count from dual  union all  select '01' as id, 'c' as word, 30 as count from dual  union all  select '02' as id, 'a' as word, 100 as count from dual  union all  select '02' as id, 'd' as word, 200 as count from dual  union all  select '02' as id, 'e' as word, 300 as count from dual  ) tmp;
```

PAI 命令：

```
PAI -name triple_to_kv -project algo_public -DinputTableName=triple2kv_test_input -DoutputTableName=triple2kv_test_input_out -DindexOutputTableName=triple2kv_test_input_index_out -DidColName=id -DkeyColName=word -DvalueColName=count -Dlifecycle=1;
```

输入说明

输入表

id	word	count
01	a	10
01	b	20
01	c	30

输出说明

- 输出 kv 表如下，kv 分隔符可以自定义：

id	key_value
01	1:10;2:20;3:30

- 输出 word 的索引表：

key	key_id
a	1
b	2
c	3

34.4.3.4.9.9 句子拆分

组件介绍

将一段文本，按标点进行句子拆分。

该组件主要用于文本摘要前的预处理，将一段文本拆分成一句一行的形式。

PAI 命令

```
PAI -name SplitSentences -project algo_public -DinputTableName="test_input" -DoutputTableName="test_output" -DdocIdCol="doc_id" -DdocContent="content" -Dlifecycle=30
```

参数说明

参数名称	参数描述	参数可选项	默认值
inputTableName	必选，输入表名	-	-
inputTablePartitions	可选，输入表中指定参与计算的分区	-	输入表的所有partition
outputTableName	必选，输出表名	-	-
docIdCol	必选，标识文章id的列名	-	-
docContent	必选，标示文章内容的列名，仅可指定一列	-	-
delimiter	可选，句子的间隔字符集合	-	。！？
lifecycle	可选，输入出表的生命周期	正整数	无生命周期
coreNum	可选，参与计算的核心数	正整数	系统自动计算
memSizePerCore	可选，每个核心需要的内存	正整数	系统自动计算

输出格式说明

输出表为两列，分别是 doc_id 和 sentence。示例如下：

doc_id	sentence
1000894	早在2008年，上交所便发布了上市公司社会责任披露相关指引，强制要求三类公司披露社会责任报告，同时鼓励其他有条件的上市公司进行自愿披露。
1000894	统计显示，2012年，沪市上市公司共计379家披露社会责任报告，包括强制披露公司305家和自

doc_id	sentence
	愿披露公司74家，合计占沪市全部上市公司的40%。

34.4.3.4.10 网络分析

34.4.3.4.10.1 k-Core

功能介绍

一个图的 KCore 是指反复去除度小于或等于 k 的节点后，所剩余的子图。若一个节点存在于KCore，而在 （K+1）CORE 中被移去，那么此节点的核数（coreness）为 k。因此所有度为1的节点的核数必然为0，节点核数的最大值被称为图的核数。

参数设置

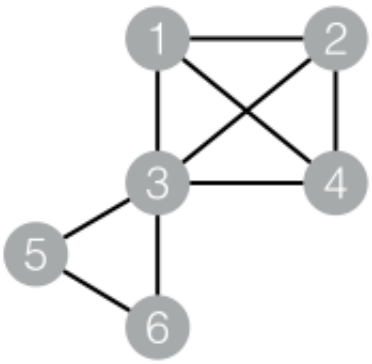
k：核数的值，必填，默认3。

实例-测试数据

新建数据 SQL：

```
drop table if exists KCore_func_test_edge; create table KCore_func_test_edge as select * from
 (   select '1' as flow_out_id,'2' as flow_in_id from dual   union all   select '1' as flow_out_id,'3' as
 flow_in_id from dual   union all   select '1' as flow_out_id,'4' as flow_in_id from dual   union all
   select '2' as flow_out_id,'3' as flow_in_id from dual   union all   select '2' as flow_out_id,'4' as
 flow_in_id from dual   union all   select '3' as flow_out_id,'4' as flow_in_id from dual   union all
   select '3' as flow_out_id,'5' as flow_in_id from dual   union all   select '3' as flow_out_id,'6' as
 flow_in_id from dual   union all   select '5' as flow_out_id,'6' as flow_in_id from dual )tmp;
```

数据对应的 graph 结构：



设定k = 2，运行结果：

```
+-------+-------+
| node1 | node2 |
+-------+-------+
| 1     | 2     |
| 1     | 3     |
| 1     | 4     |
| 2     | 1     |
| 2     | 3     |
| 2     | 4     |
| 3     | 1     |
| 3     | 2     |
| 3     | 4     |
| 4     | 1     |
| 4     | 2     |
| 4     | 3     |
+-------+-------+
```

PAI 命令

```
pai -name KCore -project algo_public -DinputEdgeTableName=KCore_func_test_edge -DfromVertexCol=flow_out_id -DtoVertexCol=flow_in_id -DoutputTableName=KCore_func_test_result -Dk=2;
```

算法参数

参数 key 名称	参数描述	必/选填	默认值
inputEdgeTableName	输入边表名	必填	-
inputEdgeTablePartitions	输入边表的分区	选填	全表读入
fromVertexCol	边表中起点所在列	必填	-
toVertexCol	边表中终点所在列	必填	-
outputTableName	输出表名	必填	-
outputTablePartitions	输出表的分区	选填	-
lifecycle	输出表申明周期	选填	-
workerNum	进程数量	选填	未设置
workerMem	进程内存	选填	4096
splitSize	数据切分大小	选填	64
k	核数	必填	3

34.4.3.4.10.2 单源最短路径

功能介绍

单源最短路径参考 Dijkstra 算法，本算法中当给定起点，则输出该点和其他所有节点的最短路径。

参数设置

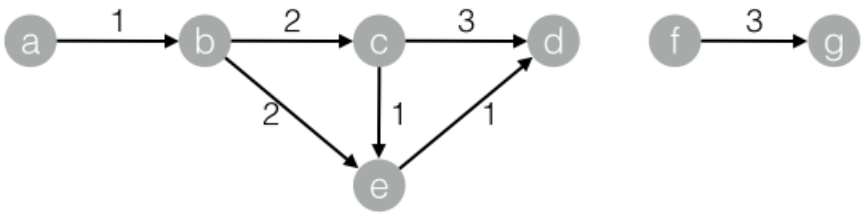
起始节点id：用于计算最短路径的起始节点，必填。

实例-测试数据

新建数据的 SQL 语句：

```
drop table if exists SSSP_func_test_edge; create table SSSP_func_test_edge as select flow_out_id,flow_in_id,edge_weight from ( select "a" as flow_out_id,"b" as flow_in_id,1.0 as edge_weight from dual union all select "b" as flow_out_id,"c" as flow_in_id,2.0 as edge_weight from dual union all select "c" as flow_out_id,"d" as flow_in_id,1.0 as edge_weight from dual union all select "b" as flow_out_id,"e" as flow_in_id,2.0 as edge_weight from dual union all select "e" as flow_out_id,"d" as flow_in_id,1.0 as edge_weight from dual union all select "c" as flow_out_id,"e" as flow_in_id,1.0 as edge_weight from dual union all select "f" as flow_out_id,"g" as flow_in_id,3.0 as edge_weight from dual union all select "a" as flow_out_id,"d" as flow_in_id,4.0 as edge_weight from dual ) tmp ;
```

数据对应的 graph 结构：



运行结果

结果如下：

start_node	dest_node	distance	distance_cnt
a	b	1.0	1
a	c	3.0	1
a	d	4.0	3
a	a	0.0	0
a	e	3.0	1

PAI 命令

```
pai -name SSSP -project algo_public -DinputEdgeTableName=SSSP_func_test_edge
-DfromVertexCol=flow_out_id -DtoVertexCol=flow_in_id -DoutputTableName=
SSSP_func_test_result -DhasEdgeWeight=true -DedgeWeightCol=edge_weight -
DstartVertex=a;
```

算法参数

参数 key 名称	参数描述	必/选填	默认值
inputEdgeTableName	输入边表名	必填	-
inputEdgeTablePartitions	输入边表的分区	选填	全表读入
fromVertexCol	输入边表的起点所在列	必填	-
toVertexCol	输入边表的终点所在列	必填	-
outputTableName	输出表名	必填	-
outputTablePartitions	输出表的分区	选填	-
lifecycle	输出表申明周期	选填	-
workerNum	进程数量	选填	未设置
workerMem	进程内存	选填	4096
splitSize	数据切分大小	选填	64
startVertex	起始节点ID	必填	-
hasEdgeWeight	输入边表的边是否有权重	选填	false
edgeWeightCol	输入边表边的权重所在列	选填	-

34.4.3.4.10.3 PageRank

功能介绍

PageRank 起于网页的搜索排序，google 利用网页的链接结构计算每个网页的等级排名，其基本思路是：如果一个网页被其他多个网页指向，这说明该网页比较重要或者质量较高。除考虑网页的链接数量，还考虑网页本身的权重级别，以及该网页有多少条出链到其它网页。 对于用户构成的人际网络，除了用户本身的影响力之外，边的权重也是重要因素之一。例如：新浪微博的某个用户，会更容易影响粉丝中关系比较亲密的家人、同学、同事等，而对陌生的弱关系粉丝影响较小。在人际网络中，边的权重等价为用户-用户的关系强弱指数。带连接权重的 PageRank 公式为：

$$W(A) = (1-d) + d * (\sum_{i} W(i) * C(Ai))$$

其中，w(i) 为节点i的权重，c(A,i) 为链接权重，d 为阻尼系数，算法迭代稳定后的节点权重 W 即为每个用户的影响力指数。

参数设置

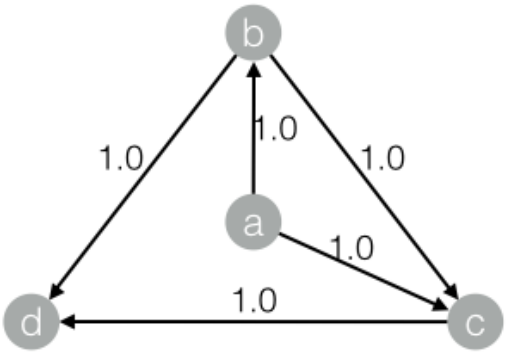
最大迭代次数：算法自身会收敛并停止迭代，选填，默认30。

实例-测试数据

新建数据的 SQL 语句：

```
drop table if exists PageRankWithWeight_func_test_edge; create table PageRankWithWeight_func_test_edge as select * from (  select 'a' as flow_out_id,'b' as flow_in_id,1.0 as weight from dual  union all  select 'a' as flow_out_id,'c' as flow_in_id,1.0 as weight from dual  union all  select 'b' as flow_out_id,'c' as flow_in_id,1.0 as weight from dual  union all  select 'b' as flow_out_id,'d' as flow_in_id,1.0 as weight from dual  union all  select 'c' as flow_out_id,'d' as flow_in_id,1.0 as weight from dual )tmp ;
```

对应的 graph 结构：



运行结果

```
结果如下:
+------+------------+
| node | weight     |
+------+------------+
| a    | 0.0375     |
| b    | 0.06938    |
| c    | 0.12834    |
| d    | 0.20556    |
+------+------------+
```

PAI 命令

```
pai -name PageRankWithWeight -project algo_public -DinputEdgeTableName=PageRankWithWeight_func_test_edge -DfromVertexCol=flow_out_id -DtoVertexCol=flow_in_id -DoutputTableName=PageRankWithWeight_func_test_result -DhasEdgeWeight=true -DedgeWeightCol=weight -DmaxIter 100;
```

算法参数

参数 key 名称	参数描述	必/选填	默认值
inputEdgeTableName	输入边表名	必填	-
inputEdgeTablePartitions	输入边表的分区	选填	全表读入
fromVertexCol	输入边表的起点所在列	必填	-
toVertexCol	输入边表的终点所在列	必填	-
outputTableName	输出表名	必填	-
outputTablePartitions	输出表的分区	选填	-
lifecycle	输出表申明周期	选填	-
workerNum	进程数量	选填	未设置
workerMem	进程内存	选填	4096
splitSize	数据切分大小	选填	64
hasEdgeWeight	输入边表的边是否有权重	选填	false
edgeWeightCol	输入边表边的权重所在列	选填	-

参数 key 名称	参数描述	必/选填	默认值
maxIter	最大迭代次数	选填	30

34.4.3.4.10.4 标签传播聚类

功能介绍

图聚类是根据图的拓扑结构，进行子图的划分，使得子图内部节点的链接较多，子图之间的连接较少。标签传播算法（Label Propagation Algorithm, LPA）是基于图的半监督学习方法，其基本思路是节点的标签（community）依赖其邻居节点的标签信息，影响程度由节点相似度决定，并通过传播迭代更新达到稳定。

参数介绍

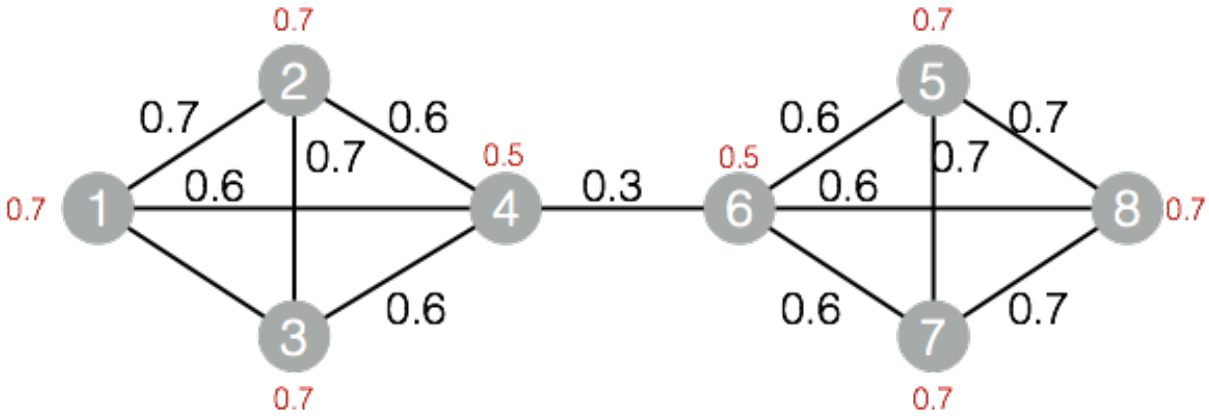
最大迭代次数：选填，默认30。

实例-测试数据

数据生成 SQL:

```
drop table if exists LabelPropagationClustering_func_test_edge; create table LabelPropagationClustering_func_test_edge as select * from ( select '1' as flow_out_id,'2' as flow_in_id,0.7 as edge_weight from dual union all select '1' as flow_out_id,'3' as flow_in_id,0.7 as edge_weight from dual union all select '1' as flow_out_id,'4' as flow_in_id,0.6 as edge_weight from dual union all select '2' as flow_out_id,'3' as flow_in_id,0.7 as edge_weight from dual union all select '2' as flow_out_id,'4' as flow_in_id,0.6 as edge_weight from dual union all select '3' as flow_out_id,'4' as flow_in_id,0.6 as edge_weight from dual union all select '4' as flow_out_id,'6' as flow_in_id,0.3 as edge_weight from dual union all select '5' as flow_out_id,'6' as flow_in_id,0.6 as edge_weight from dual union all select '5' as flow_out_id,'7' as flow_in_id,0.7 as edge_weight from dual union all select '5' as flow_out_id,'8' as flow_in_id,0.7 as edge_weight from dual union all select '6' as flow_out_id,'7' as flow_in_id,0.6 as edge_weight from dual union all select '6' as flow_out_id,'8' as flow_in_id,0.6 as edge_weight from dual union all select '7' as flow_out_id,'8' as flow_in_id,0.7 as edge_weight from dual )tmp ; drop table if exists LabelPropagationClustering_func_test_node; create table LabelPropagationClustering_func_test_node as select * from ( select '1' as node,0.7 as node_weight from dual union all select '2' as node,0.7 as node_weight from dual union all select '3' as node,0.7 as node_weight from dual union all select '4' as node,0.5 as node_weight from dual union all select '5' as node,0.7 as node_weight from dual union all select '6' as node,0.5 as node_weight from dual union all select '7' as node,0.7 as node_weight from dual union all select '8' as node,0.7 as node_weight from dual )tmp ;
```

数据对应的 group 结构：



运行结果

node	group_id
1	1
2	1
3	1
4	1
5	5
6	5
7	5
8	5

PAI 命令

```
pai -name LabelPropagationClustering -project algo_public -DinputEdgeTableName=
LabelPropagationClustering_func_test_edge -DfromVertexCol=flow_out_id
-DtoVertexCol=flow_in_id -DinputVertexTableName=LabelPropagationClus
tering_func_test_node -DvertexCol=node -DoutputTableName=LabelPropagationClus
tering_func_test_result -DhasEdgeWeight=true -DedgeWeightCol=edge_weight
-DhasVertexWeight=true -DvertexWeightCol=node_weight -DrandSelect=true -
DmaxIter=100;
```

算法参数

参数 key 名称	参数描述	必/选填	默认值
inputEdgeTableName	输入边表名	必填	-
inputEdgeTablePartitions	输入边表的分区	选填	全表读入
fromVertexCol	输入边表的起点所在列	必填	-

参数 key 名称	参数描述	必/选填	默认值
toVertexCol	输入边表的终点所在列	必填	-
inputVertexTableName	输入点表名称	必填	-
inputVertexTablePartitions	输入点表的分区	选填	全表读入
vertexCol	输入点表的点所在列	必填	-
outputTableName	输出表名	必填	-
outputTablePartitions	输出表的分区	选填	-
lifecycle	输出表申明周期	选填	-
workerNum	进程数量	选填	未设置
workerMem	进程内存	选填	4096
splitSize	数据切分大小	选填	64
hasEdgeWeight	输入边表的边是否有权重	选填	false
edgeWeightCol	输入边表边的权重所在列	选填	-
hasVertexWeight	输入点表的点是否有权重	选填	false
vertexWeightCol	输入点表的点的权重所在列	选填	-
randSelect	是否随机选择最大标签	选填	false
maxIter	最大迭代次数	选填	30

34.4.3.4.10.5 标签传播分类

功能介绍

该算法为半监督的分类算法，原理为用已标记节点的标签信息去预测未标记节点的标签信息。

在算法执行过程中，每个节点的标签按相似度传播给相邻节点，在节点传播的每一步，每个节点根据相邻节点的标签来更新自己的标签，与该节点相似度越大，其相邻节点对其标注的影响权值越大，相似节点的标签越趋于一致，其标签就越容易传播。在标签传播过程中，保持已标注数据的标签不变，使其像一个源头把标签传向未标注数据。

最终，当迭代过程结束时，相似节点的概率分布也趋于相似，可以划分到同一个类别中，从而完成标签传播过程。

参数设置

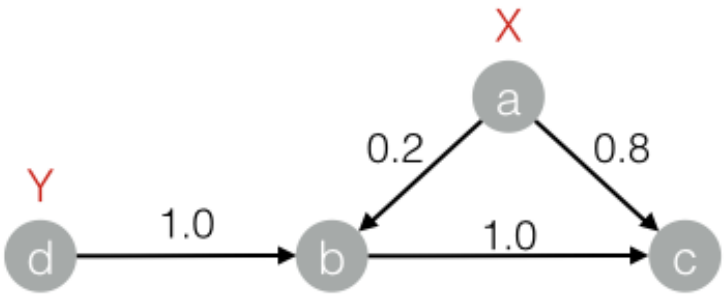
阻尼系数:默认0.8 收敛系数:默认0.000001。

实例-测试数据

生成数据的 SQL：

```
drop table if exists LabelPropagationClassification_func_test_edge; create table LabelPropagationClassification_func_test_edge as select * from ( select 'a' as flow_out_id, 'b' as flow_in_id, 0.2 as edge_weight from dual union all select 'a' as flow_out_id, 'c' as flow_in_id, 0.8 as edge_weight from dual union all select 'b' as flow_out_id, 'c' as flow_in_id, 1.0 as edge_weight from dual union all select 'd' as flow_out_id, 'b' as flow_in_id, 1.0 as edge_weight from dual )tmp ; drop table if exists LabelPropagationClassification_func_test_node; create table LabelPropagationClassification_func_test_node as select * from ( select 'a' as node,'X' as label, 1.0 as label_weight from dual union all select 'd' as node,'Y' as label, 1.0 as label_weight from dual )tmp ;
```

对应的图结构：



运行结果

结果如下:

node	tag	weight
a	X	1.0
b	X	0.16667
b	Y	0.83333
c	X	0.53704
c	Y	0.46296
d	Y	1.0

PAI 命令

```
pai -name LabelPropagationClassification -project algo_public -DinputEdgeTableName=LabelPropagationClassification_func_test_edge -DfromVertexCol=flow_out_id -DtoVertexCol=flow_in_id -DinputVertexTableName=LabelPropagationClassification_func_test_node -DvertexCol=node -DvertexLabelCol=label -DoutputTableName=LabelPropagationClassification_func_test_result -DhasEdgeWeight=true -DedgeWeightCol=edge_weight -DhasVertexWeight=true -DvertexWeightCol=label_weight -Dalpha=0.8 -Depsilon=0.000001;
```

算法参数

参数 key 名称	参数描述	必/选填	默认值
inputEdgeTableName	输入边表名	必填	-
inputEdgeTablePartitions	输入边表的分区	选填	全表读入
fromVertexCol	输入边表的起点所在列	必填	-
toVertexCol	输入边表的终点所在列	必填	-
inputVertexTableName	输入点表名称	必填	-
inputVertexTablePartitions	输入点表的分区	选填	全表读入
vertexCol	输入点表的点所在列	必填	-
vertexLabelCol	输入点表的点的标签	必填	-
outputTableName	输出表名	必填	-
outputTablePartitions	输出表的分区	选填	-
lifecycle	输出表申明周期	选填	-
workerNum	进程数量	选填	未设置
workerMem	进程内存	选填	4096
splitSize	数据切分大小	选填	64
hasEdgeWeight	输入边表的边是否有权重	选填	false
edgeWeightCol	输入边表边的权重所在列	选填	-
hasVertexWeight	输入点表的点是否有权重	选填	false

参数 key 名称	参数描述	必/选填	默认值
vertexWeightCol	输入点表的点的权重所在列	选填	-
alpha	阻尼系数	选填	0.8
epsilon	收敛系数	选填	0.000001
maxIter	最大迭代次数	选填	30

34.4.3.4.10.6 Modularity

功能介绍

Modularity 是一种评估社区网络结构的指标，来评估网络结构中划分出来社区的紧密程度，往往0.3以上是比较明显的社区结构。

实例-测试数据

略（与标签传播聚类算法的数据相同）。

运行结果

```
结果如下:
+--------------+
| val          |
+--------------+
| 0.4230769    |
+--------------+
```

PAI 命令

```
pai -name Modularity -project algo_public -DinputEdgeTableName=Modularity_func_test_edge -DfromVertexCol=flow_out_id -DfromGroupCol=group_out_id -DtoVertexCol=flow_in_id -DtoGroupCol=group_in_id -DoutputTableName=Modularity_func_test_result;
```

算法参数

参数 key 名称	参数描述	必/选填	默认值
inputEdgeTableName	输入边表名	必填	-
inputEdgeTablePartitions	输入边表的分区	选填	全表读入
fromVertexCol	输入边表的起点所在列	必填	-

参数 key 名称	参数描述	必/选填	默认值
fromGroupCol	输入边表起点的群组	必填	-
toVertexCol	输入边表的终点所在列	必填	-
toGroupCol	输入边表终点的群组	必填	-
outputTableName	输出表名	必填	-
outputTablePartitions	输出表的分区	选填	-
lifecycle	输出表申明周期	选填	-
workerNum	进程数量	选填	未设置
workerMem	进程内存	选填	4096
splitSize	数据切分大小	选填	64

34.4.3.4.10.7 最大联通子图

功能介绍

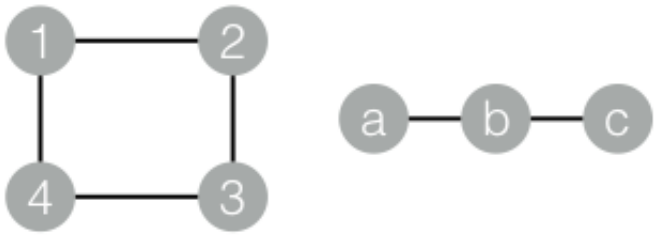
在无向图 G 中，若从顶点 A 到顶点 B 有路径相连，则称 A 和 B 是连通的；在图 G 种存在若干子图，其中每个子图中所有顶点之间都是连通的，但在不同子图间不存在顶点连通，那么称图 G 的这些子图为最大连通子图。

实例-测试数据

生成数据的 SQL：

```
drop table if exists MaximalConnectedComponent_func_test_edge; create table MaximalConnectedComponent_func_test_edge as select * from ( select '1' as flow_out_id,'2' as flow_in_id from dual union all select '2' as flow_out_id,'3' as flow_in_id from dual union all select '3' as flow_out_id,'4' as flow_in_id from dual union all select '1' as flow_out_id,'4' as flow_in_id from dual union all select 'a' as flow_out_id,'b' as flow_in_id from dual union all select 'b' as flow_out_id,'c' as flow_in_id from dual )tmp; drop table if exists MaximalConnectedComponent_func_test_result; create table MaximalConnectedComponent_func_test_result ( node string, grp_id string );
```

对应的图结构：



运行结果

```
结果如下:
+-------+-------+
| node  | grp_id|
+-------+-------+
| 1     | 4     |
| 2     | 4     |
| 3     | 4     |
| 4     | 4     |
| a     | c     |
| b     | c     |
| c     | c     |
+-------+-------+
```

PAI 命令

```
pai -name MaximalConnectedComponent -project algo_public -DinputEdgeTableName=MaximalConnectedComponent_func_test_edge -DfromVertexCol=flow_out_id -DtoVertexCol=flow_in_id -DoutputTableName=MaximalConnectedComponent_func_test_result;
```

算法参数

参数 key 名称	参数描述	必/选填	默认值
inputEdgeTableName	输入边表名	必填	-
inputEdgeTablePartitions	输入边表的分区	选填	全表读入
fromVertexCol	输入边表的起点所在列	必填	-
toVertexCol	输入边表的终点所在列	必填	-
outputTableName	输出表名	必填	-
outputTablePartitions	输出表的分区	选填	-
lifecycle	输出表申明周期	选填	-
workerNum	进程数量	选填	未设置
workerMem	进程内存	选填	4096
splitSize	数据切分大小	选填	64

34.4.3.4.10.8 点聚类系数

功能介绍

在无向图 G 中，计算每一个节点周围的稠密度，星状网络稠密度为0，全联通网络稠密度为1。

参数设置

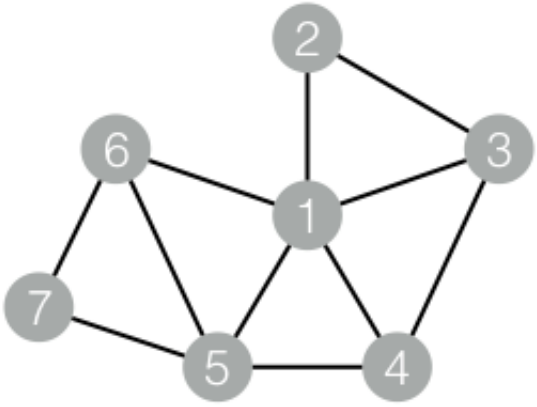
maxEdgeCnt：若节点度大于该值，则进行抽样，默认500，选填。

实例-测试数据

生成数据的 SQL：

```
drop table if exists NodeDensity_func_test_edge; create table NodeDensity_func_test_edge as
 select * from (    select '1' as flow_out_id, '2' as flow_in_id from dual    union all    select '1' as
 flow_out_id, '3' as flow_in_id from dual    union all    select '1' as flow_out_id, '4' as flow_in_id
 from dual    union all    select '1' as flow_out_id, '5' as flow_in_id from dual    union all    select '1'
 as flow_out_id, '6' as flow_in_id from dual    union all    select '2' as flow_out_id, '3' as flow_in_id
 from dual    union all    select '3' as flow_out_id, '4' as flow_in_id from dual    union all    select '4'
 as flow_out_id, '5' as flow_in_id from dual    union all    select '5' as flow_out_id, '6' as flow_in_id
 from dual    union all    select '5' as flow_out_id, '7' as flow_in_id from dual    union all    select '6' as
 flow_out_id, '7' as flow_in_id from dual )tmp;  drop table if exists NodeDensity_func_test_result;
 create table NodeDensity_func_test_result (    node string,    node_cnt bigint,    edge_cnt bigint,
  density double,    log_density double );
```

对应的图结构：



运行结果

```
结果如下:
1,5,4,0.4,1.45657
2,2,1,1.0,1.24696
3,3,2,0.66667,1.35204
4,3,2,0.66667,1.35204
5,4,3,0.5,1.41189
6,3,2,0.66667,1.35204
7,2,1,1.0,1.24696
```

PAI 命令

```
pai -name NodeDensity    -project algo_public    -DinputEdgeTableName=NodeDensity_func_test_edge      -DfromVertexCol=flow_out_id      -DtoVertexCol=flow_in_id      -DoutputTableName=NodeDensity_func_test_result     -DmaxEdgeCnt=500;
```

算法参数

参数 key 名称	参数描述	必/选填	默认值
inputEdgeTableName	输入边表名	必填	-
inputEdgeTablePartitions	输入边表的分区	选填	全表读入
fromVertexCol	输入边表的起点所在列	必填	-
toVertexCol	输入边表的终点所在列	必填	-
outputTableName	输出表名	必填	-
outputTablePartitions	输出表的分区	选填	-
lifecycle	输出表申明周期	选填	-
maxEdgeCnt	若节点度大于该值，则进行抽样。	选填	500
workerNum	进程数量	选填	未设置
workerMem	进程内存	选填	4096
splitSize	数据切分大小	选填	64

34.4.3.4.10.9 边聚类系数

功能介绍

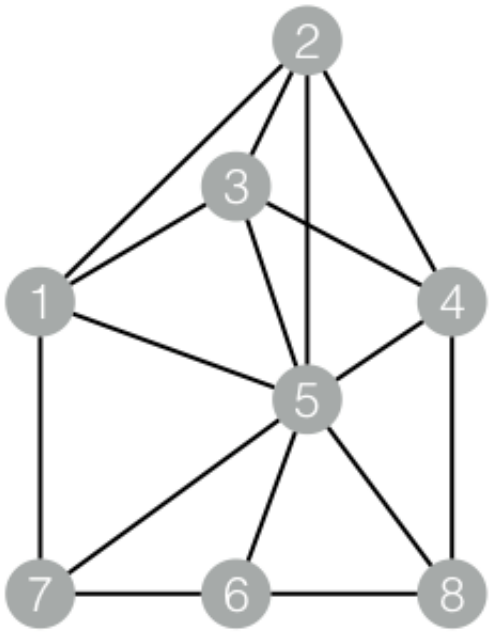
在无向图 G 中，计算每一条边周围的稠密度。

实例-测试数据

生成数据的 SQL：

```
drop table if exists EdgeDensity_func_test_edge; create table EdgeDensity_func_test_edge as select * from ( select '1' as flow_out_id,'2' as flow_in_id from dual union all select '1' as flow_out_id,'3' as flow_in_id from dual union all select '1' as flow_out_id,'5' as flow_in_id from dual union all select '1' as flow_out_id,'7' as flow_in_id from dual union all select '2' as flow_out_id,'5' as flow_in_id from dual union all select '2' as flow_out_id,'4' as flow_in_id from dual union all select '2' as flow_out_id,'3' as flow_in_id from dual union all select '3' as flow_out_id,'5' as flow_in_id from dual union all select '3' as flow_out_id,'4' as flow_in_id from dual union all select '4' as flow_out_id,'5' as flow_in_id from dual union all select '4' as flow_out_id,'8' as flow_in_id from dual union all select '5' as flow_out_id,'6' as flow_in_id from dual union all select '5' as flow_out_id,'7' as flow_in_id from dual union all select '5' as flow_out_id,'8' as flow_in_id from dual union all select '7' as flow_out_id,'6' as flow_in_id from dual union all select '6' as flow_out_id,'8' as flow_in_id from dual )tmp; drop table if exists EdgeDensity_func_test_result; create table EdgeDensity_func_test_result ( node1 string, node2 string, node1_edge_cnt bigint, node2_edge_cnt bigint, triangle_cnt bigint, density double );
```

对应的图结构：



运行结果

```
结果如下:
1,2,4,4,2,0.5
2,3,4,4,3,0.75
2,5,4,7,3,0.75
3,1,4,4,2,0.5
3,4,4,4,2,0.5
4,2,4,4,2,0.5
4,5,4,7,3,0.75
5,1,7,4,3,0.75
5,3,7,4,3,0.75
5,6,7,3,2,0.66667
5,8,7,3,2,0.66667
6,7,3,3,1,0.33333
7,1,3,4,1,0.33333
7,5,3,7,2,0.66667
8,4,3,4,1,0.33333
8,6,3,3,1,0.33333
```

PAI 命令

```
pai -name EdgeDensity -project algo_public -DinputEdgeTableName=EdgeDensity_func_test_edge -DfromVertexCol=flow_out_id -DtoVertexCol=flow_in_id -DoutputTableName=EdgeDensity_func_test_result;
```

算法参数

参数 key 名称	参数描述	必/选填	默认值
inputEdgeTableName	输入边表名	必填	-
inputEdgeTablePartitions	输入边表的分区	选填	全表读入
fromVertexCol	输入边表的起点所在列	必填	-
toVertexCol	输入边表的终点所在列	必填	-
outputTableName	输出表名	必填	-
outputTablePartitions	输出表的分区	选填	-
lifecycle	输出表申明周期	选填	-
workerNum	进程数量	选填	未设置
workerMem	进程内存	选填	4096

参数 key 名称	参数描述	必/选填	默认值
splitSize	数据切分大小	选填	64

34.4.3.4.10.10 计数三角形

功能介绍

在无向图 G 中，输出所有三角形。

参数设置

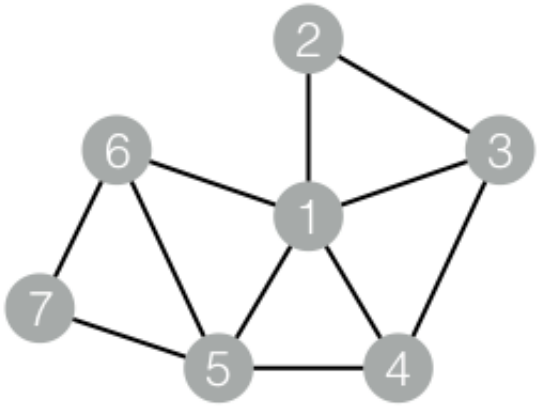
maxEdgeCnt：若节点度大于该值，则进行抽样，默认500，选填。

实例-测试数据

生成数据的 SQL：

```
drop table if exists TriangleCount_func_test_edge; create table TriangleCount_func_test_edge
 as select * from (    select '1' as flow_out_id,'2' as flow_in_id from dual    union all    select '1'
 as flow_out_id,'3' as flow_in_id from dual    union all    select '1' as flow_out_id,'4' as flow_in_id
 from dual    union all    select '1' as flow_out_id,'5' as flow_in_id from dual    union all    select '1'
 as flow_out_id,'6' as flow_in_id from dual    union all    select '2' as flow_out_id,'3' as flow_in_id
 from dual    union all    select '3' as flow_out_id,'4' as flow_in_id from dual    union all    select '4'
 as flow_out_id,'5' as flow_in_id from dual    union all    select '5' as flow_out_id,'6' as flow_in_id
 from dual   union all   select '5' as flow_out_id,'7' as flow_in_id from dual   union all   select '6' as
 flow_out_id,'7' as flow_in_id from dual )tmp;  drop table if exists TriangleCount_func_test_result;
 create table TriangleCount_func_test_result (   node1 string,   node2 string,   node3 string );
```

对应的图结构：



运行结果

```
结果如下:
1,2,3
1,3,4
1,4,5
1,5,6
5,6,7
```

PAI 命令

```
pai -name TriangleCount -project algo_public -DinputEdgeTableName=TriangleCount_func_test_edge -DfromVertexCol=flow_out_id -DtoVertexCol=flow_in_id -DoutputTableName=TriangleCount_func_test_result;
```

算法参数

参数 key 名称	参数描述	必/选填	默认值
inputEdgeTableName	输入边表名	必填	-
inputEdgeTablePartitions	输入边表的分区	选填	全表读入
fromVertexCol	输入边表的起点所在列	必填	-
toVertexCol	输入边表的终点所在列	必填	-
outputTableName	输出表名	必填	-
outputTablePartitions	输出表的分区	选填	-
lifecycle	输出表申明周期	选填	-
maxEdgeCnt	若节点度大于该值，则进行抽样。	选填	500
workerNum	进程数量	选填	未设置
workerMem	进程内存	选填	4096
splitSize	数据切分大小	选填	64

34.4.3.4.10.11 树深度

功能介绍

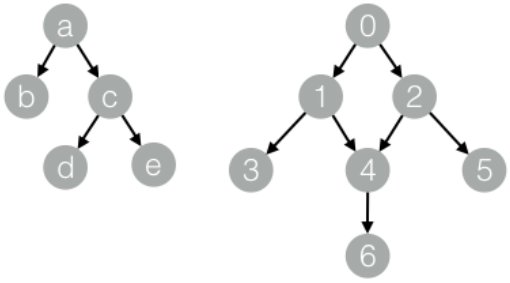
对于众多树状网络，输出每个节点的所处深度和树 ID。

实例-测试数据

生成数据的 SQL：

```
drop table if exists TreeDepth_func_test_edge; create table TreeDepth_func_test_edge as select * from ( select '0' as flow_out_id, '1' as flow_in_id from dual union all select '0' as flow_out_id, '2' as flow_in_id from dual union all select '1' as flow_out_id, '3' as flow_in_id from dual union all select '1' as flow_out_id, '4' as flow_in_id from dual union all select '2' as flow_out_id, '4' as flow_in_id from dual union all select '2' as flow_out_id, '5' as flow_in_id from dual union all select '4' as flow_out_id, '6' as flow_in_id from dual union all select 'a' as flow_out_id, 'b' as flow_in_id from dual union all select 'a' as flow_out_id, 'c' as flow_in_id from dual union all select 'c' as flow_out_id, 'd' as flow_in_id from dual union all select 'c' as flow_out_id, 'e' as flow_in_id from dual )tmp; drop table if exists TreeDepth_func_test_result; create table TreeDepth_func_test_result ( node string, root string, depth bigint );
```

对应的图结构：



运行结果

```
结果如下:
0,0,0
1,0,1
2,0,1
3,0,2
4,0,2
5,0,2
6,0,3
a,a,0
b,a,1
c,a,1
d,a,2
e,a,2
```

PAI 命令

```
pai -name TreeDepth -project algo_public -DinputEdgeTableName=TreeDepth_func_test_edge -DfromVertexCol=flow_out_id -DtoVertexCol=flow_in_id -DoutputTableName=TreeDepth_func_test_result;
```

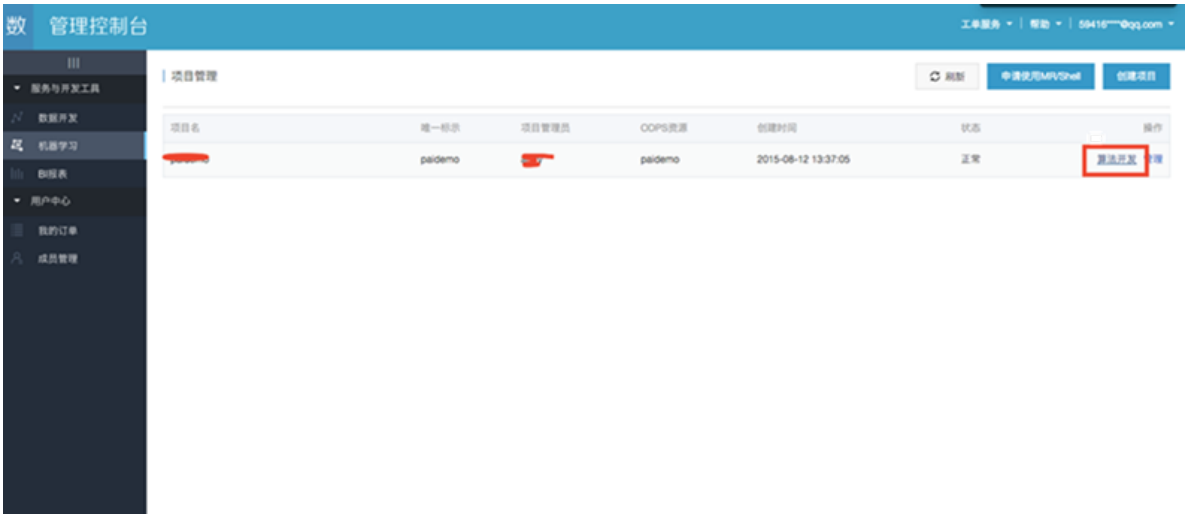
算法参数

参数 key 名称	参数描述	必/选填	默认值
inputEdgeTableName	输入边表名	必填	-
inputEdgeTablePartitions	输入边表的分区	选填	全表读入
fromVertexCol	输入边表的起点所在列	必填	-
toVertexCol	输入边表的终点所在列	必填	-
outputTableName	输出表名	必填	-
outputTablePartitions	输出表的分区	选填	-
lifecycle	输出表申明周期	选填	-
workerNum	进程数量	选填	未设置
workerMem	进程内存	选填	4096
splitSize	数据切分大小	选填	64

34.4.3.4.11 常见问题

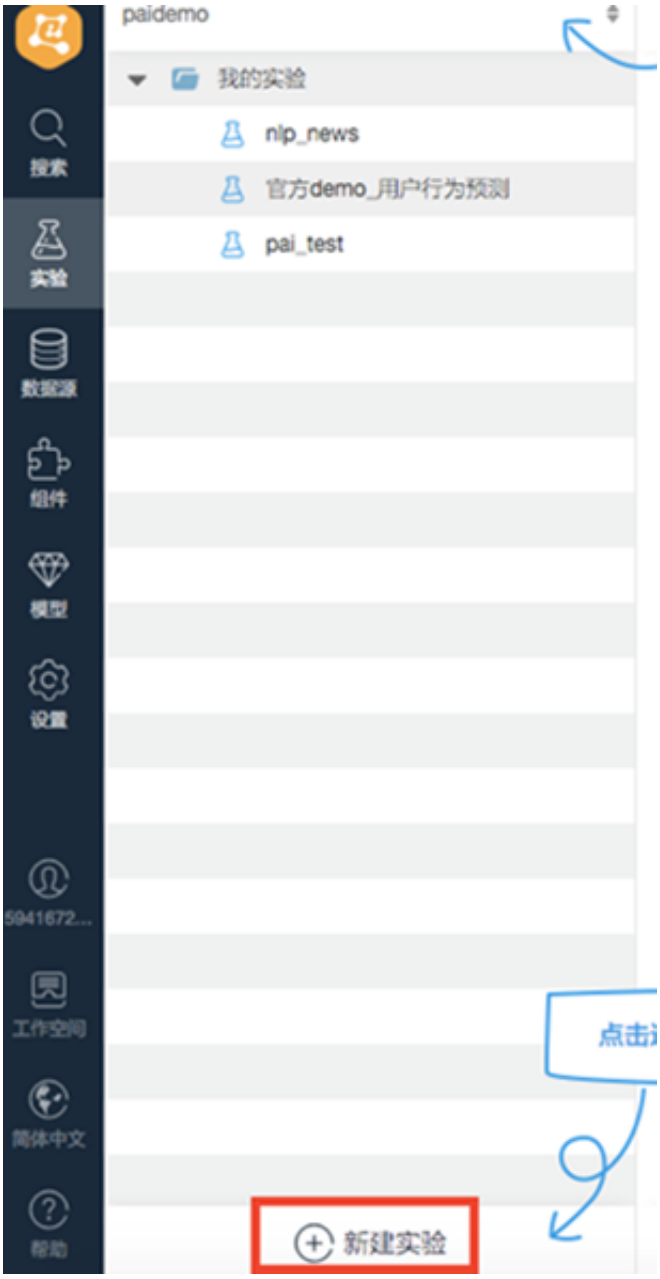
- Q：如何进入机器学习平台？

  选中机器学习板块，进入算法开发模块。



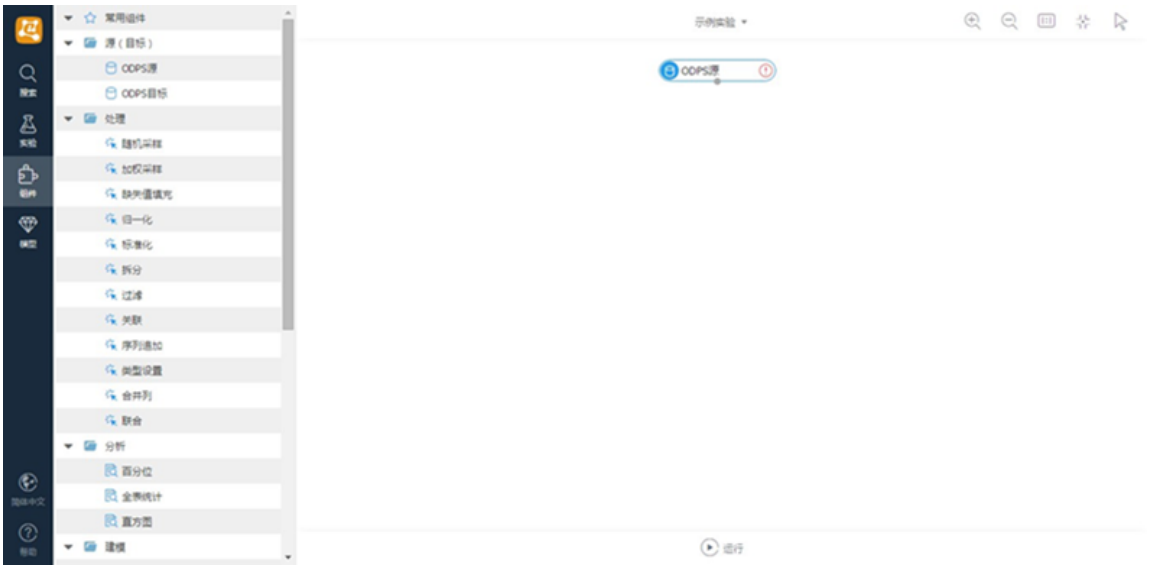
- Q：如何新建实验？

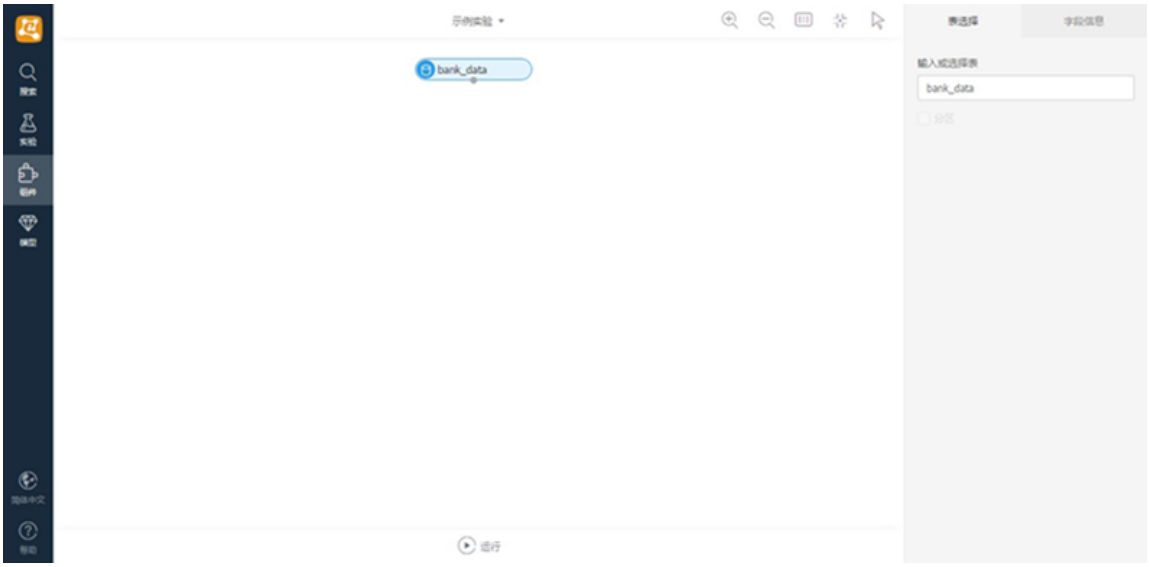
进入实验板块，点击新建实验：



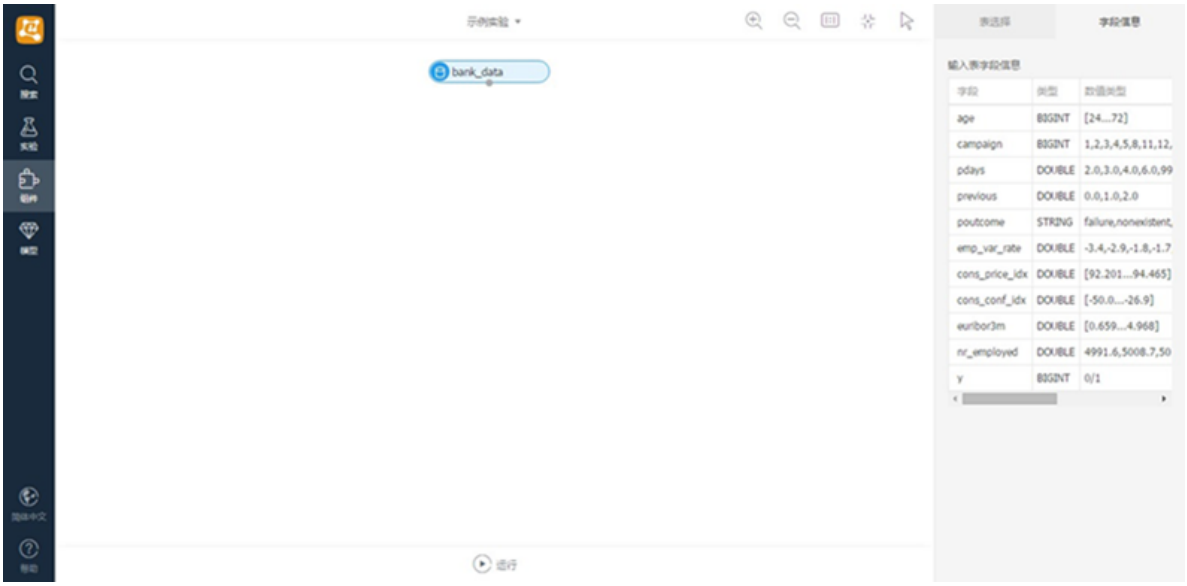
- Q：如何做数据准备？

  1. 切换到组件栏，向画布中拖入 ODPS 源，点击 ODPS 源，在右侧表选择栏填入您的 ODPS 表。



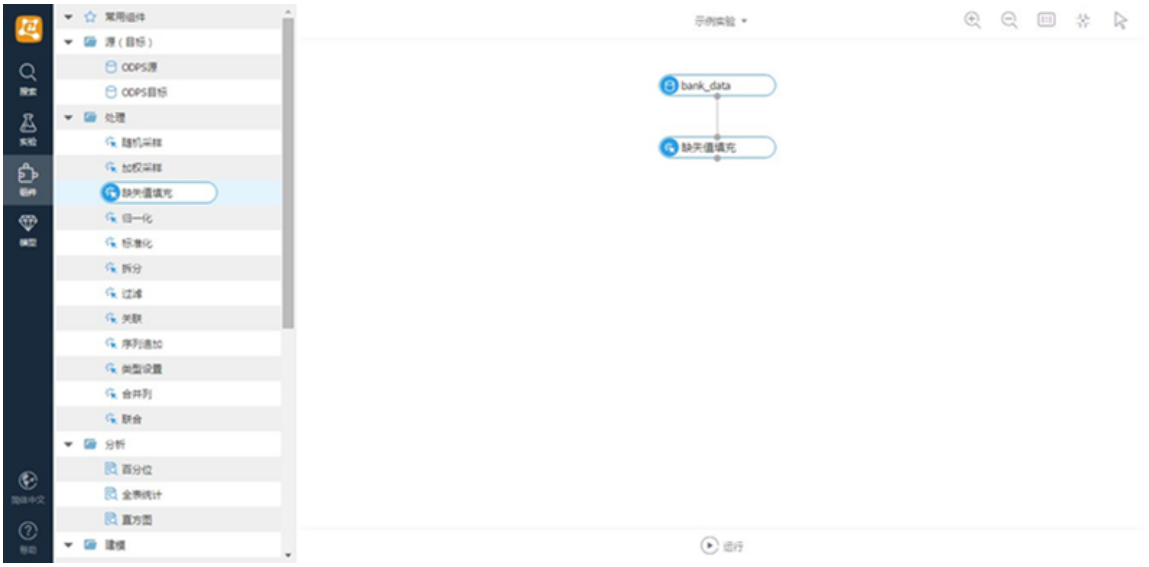


- 切换到**字段信息**栏，可以查看输入表的字段名、数据类型和前 100 行数据的数值布。

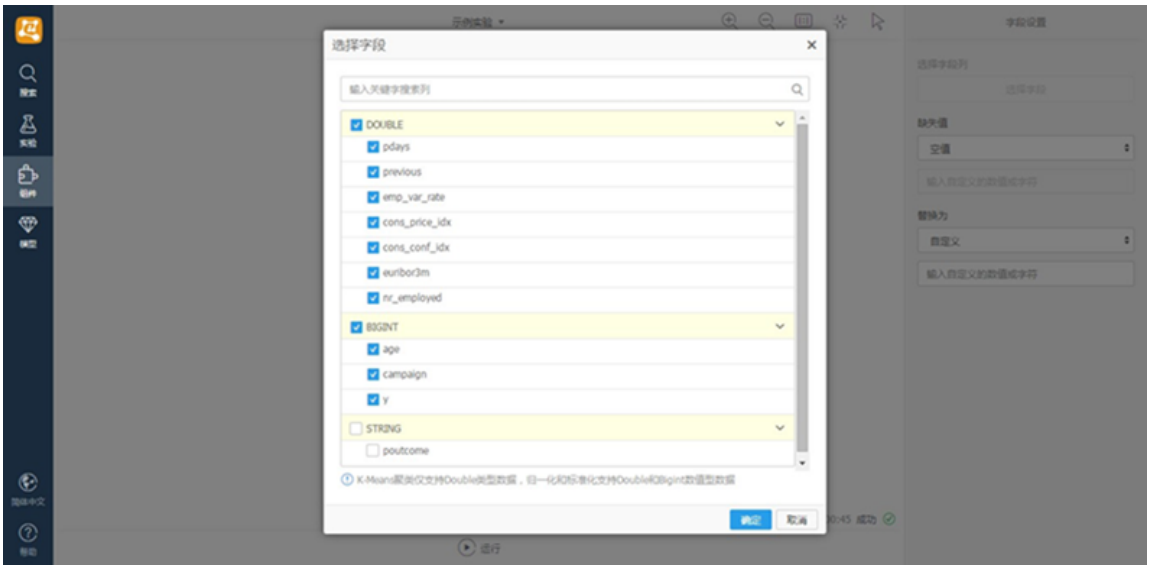


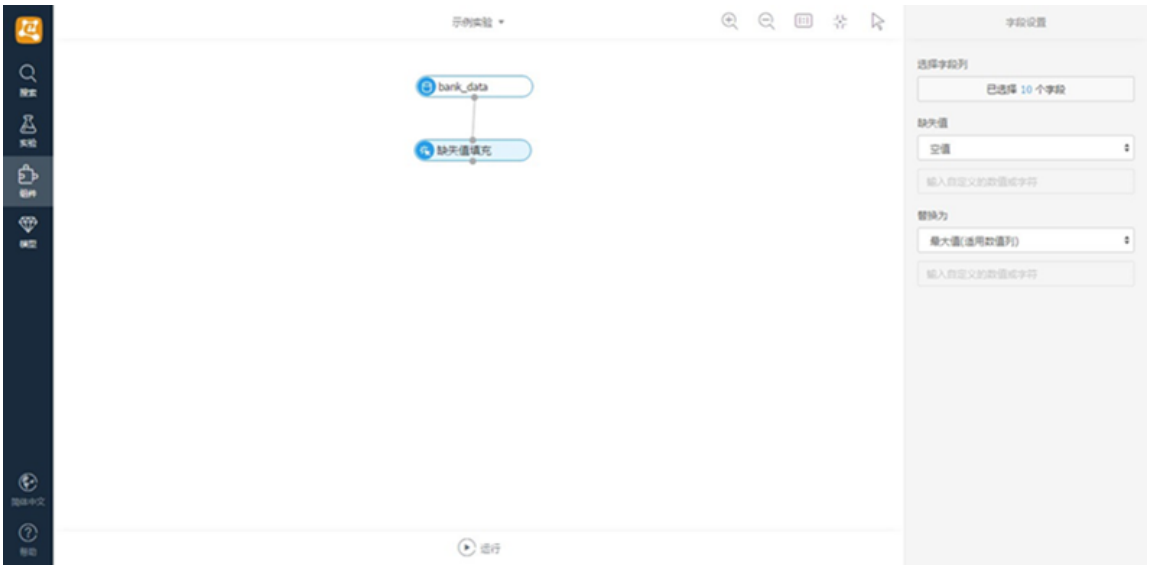
- Q：如何做数据预处理？

  1. 从处理栏拖入缺失值填充组件到画布：



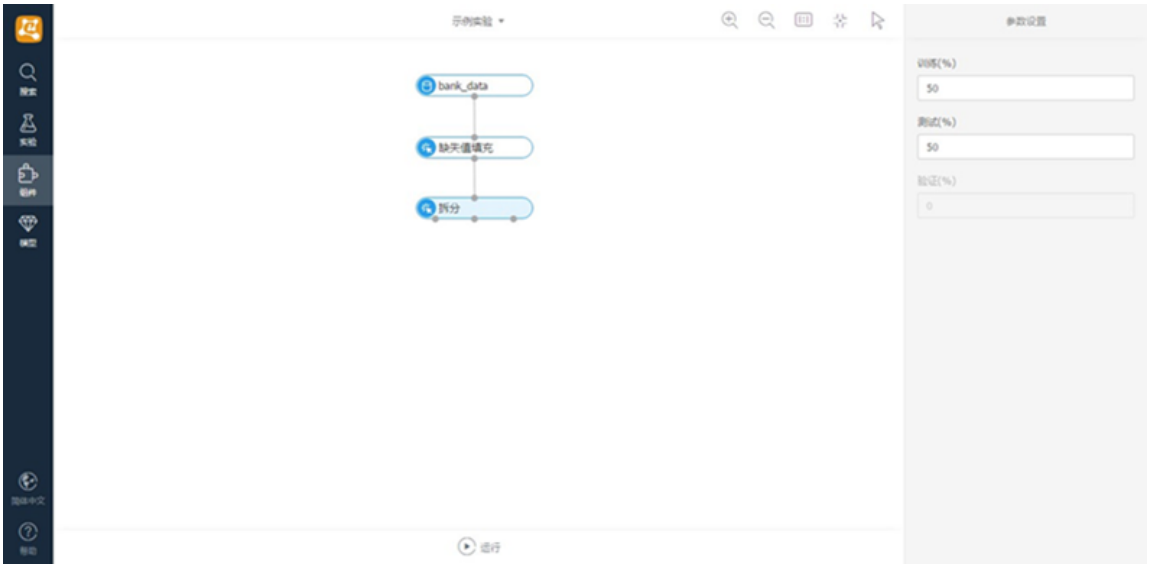
  2. 右击缺失值填充，进行字段设置。点击选择字段列，勾选所有数值型列，并用最大值进行填充：





将数字进行缺失值填充是模型训练前数据处理很重要的步骤之一。

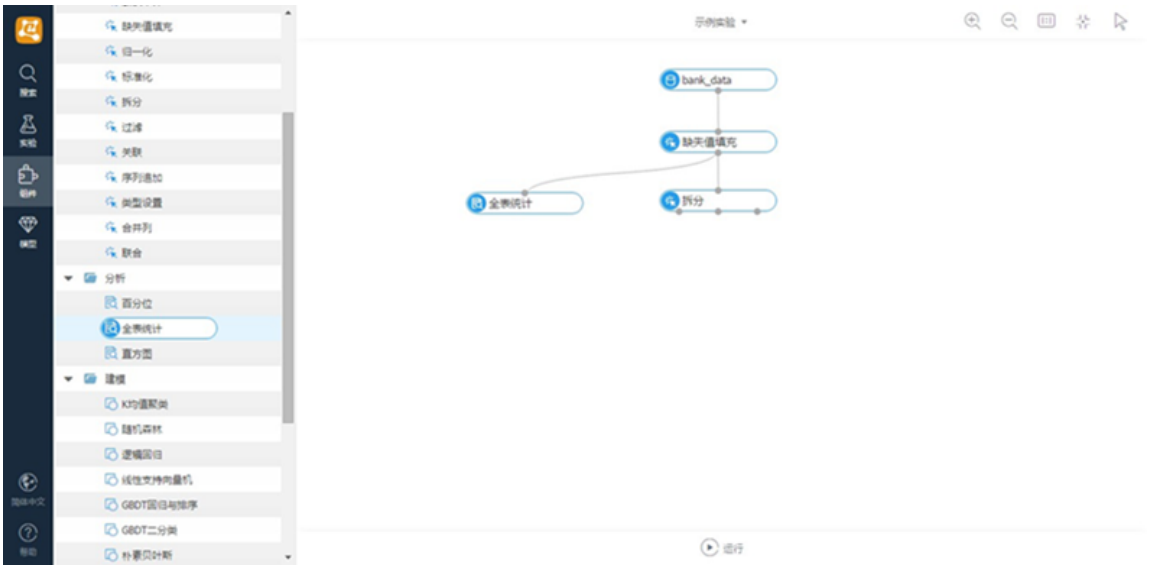
3. 将拆分组件拖到画布，步骤同上：



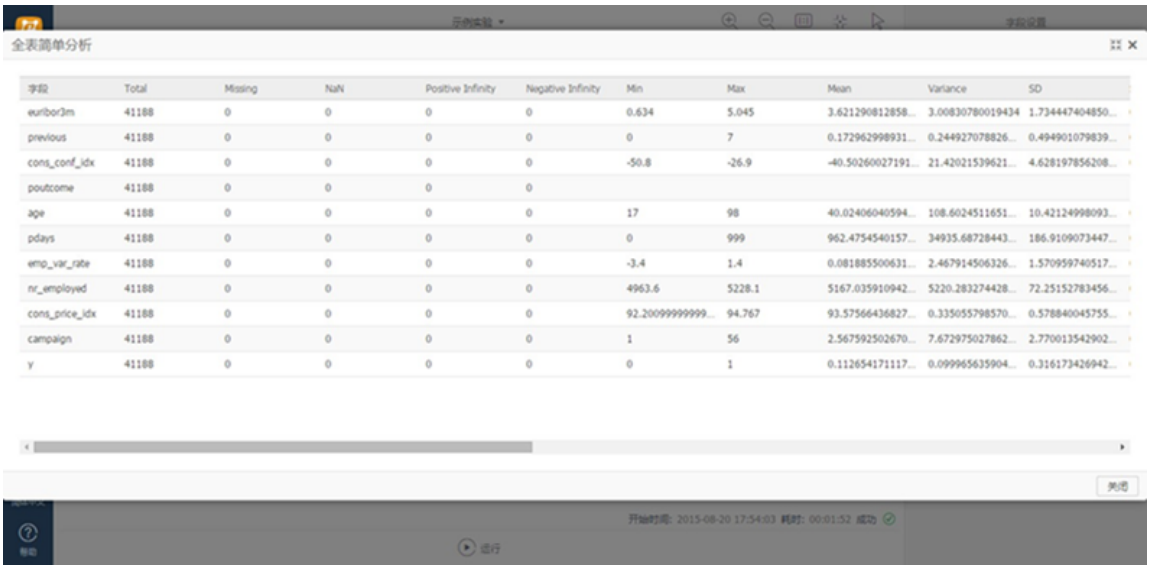
此步骤的目的是将数据拆分成两份，50%作为模型训练集，50%作为模型预测集。

- Q：如何实现数据可视化？

1. 向画布拖入全表统计组件，连线并点击画布底部的运行按钮：



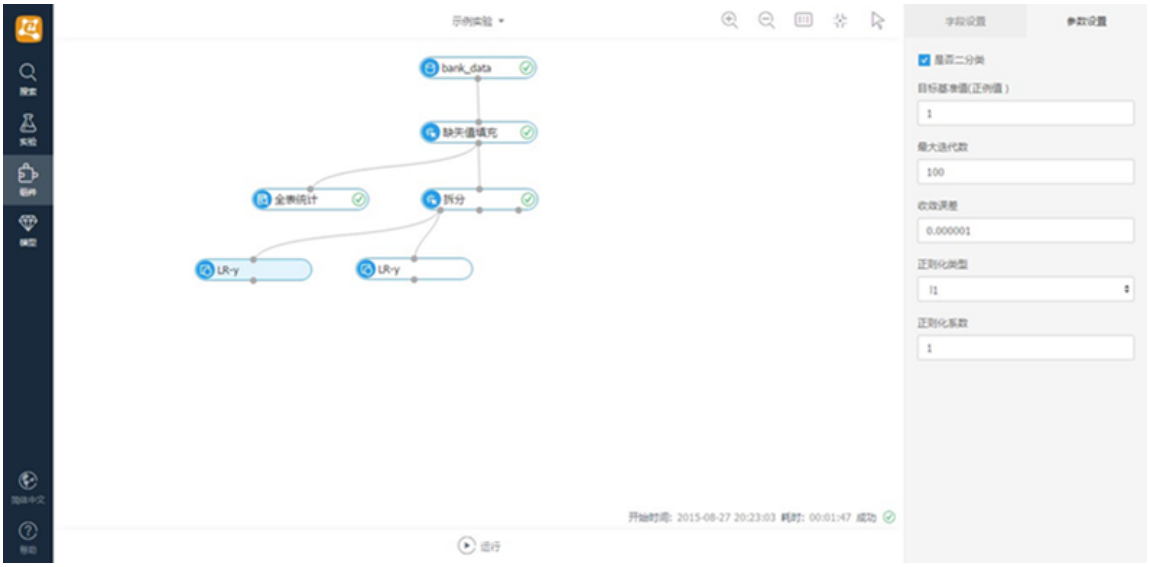
2. 待实验运行结束后，右击全表统计，点击查看分析报告，可看到数据的全表统计信息，如下：



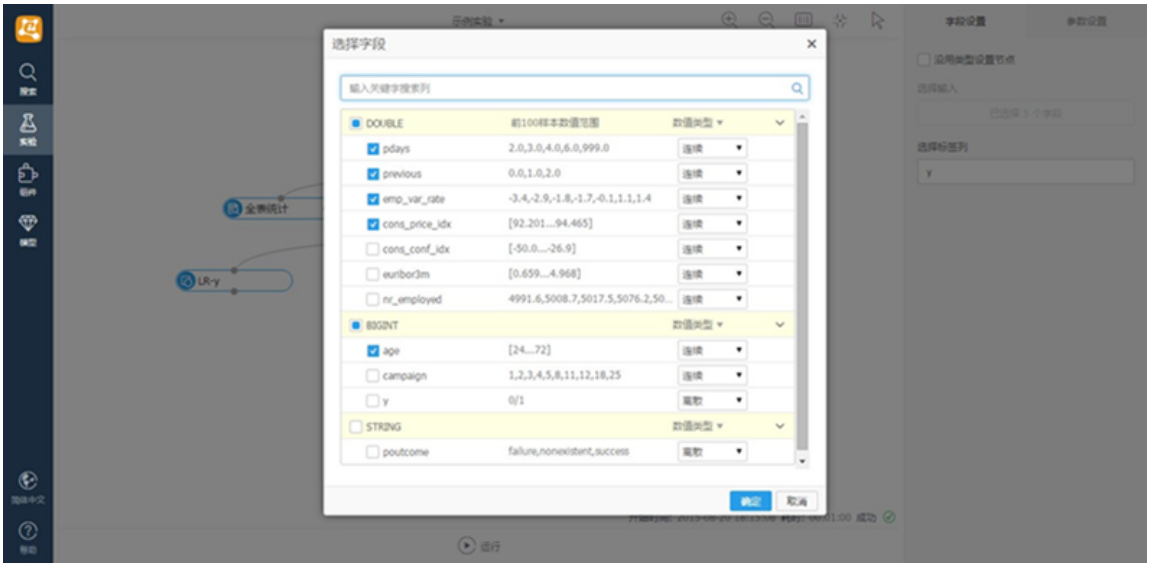
- Q：如何实现算法建模？

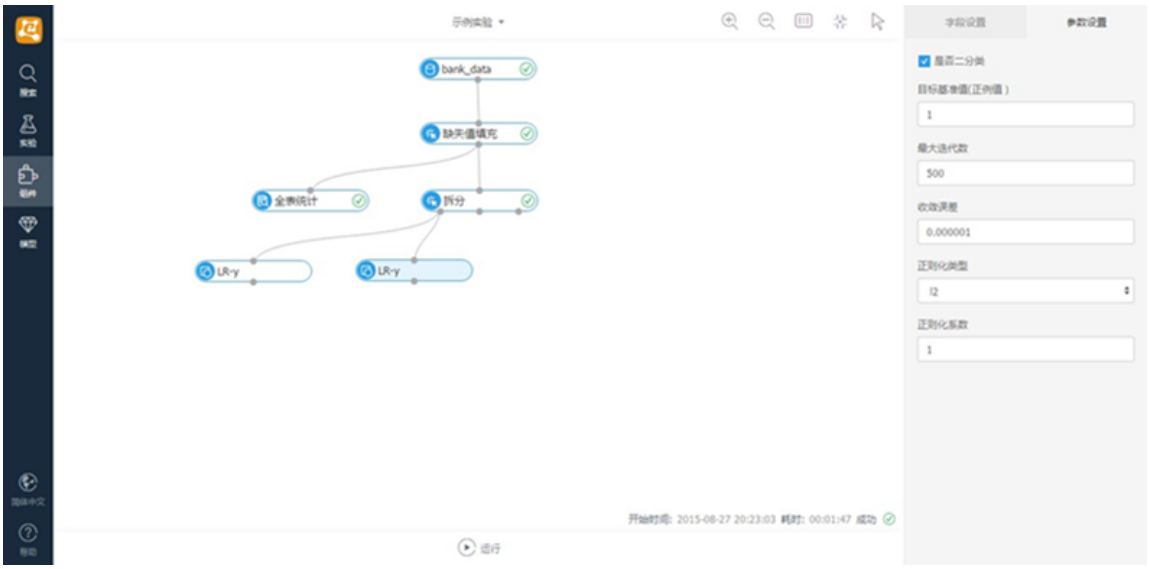
  1. 向画布拖入两个逻辑回归组件。
  2. 组件1*选择输入列* 选择 9 个 feature 列，参数设置采用系统默认参数。





3. 组件2*选择输入列* 选择 5 个 feature 列，参数设置稍作修改。





34.5 使用场景-泰坦尼克号沉船事件

场景介绍

泰坦尼克号沉船事件：通过分析泰坦尼克号沉船事件幸存者和丧生者的数据，判断拥有什么样的属性的人有更大的概率获救。

数据准备

图 1891: 数据准备

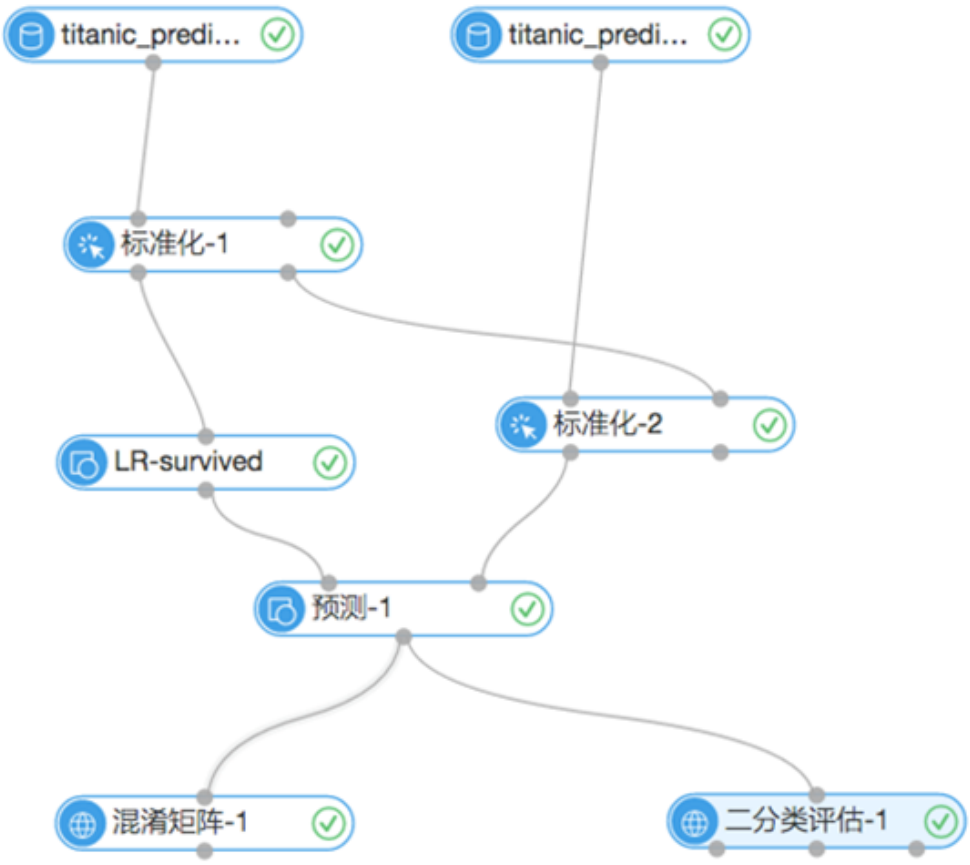
passengerid	survived	pclass	sex	age	sibsp	parch	fare	cabin	embarked
1.0	0.0	3.0	1.0	22.0	1.0	0.0	7.25	0.0	1.0
2.0	1.0	1.0	0.0	38.0	1.0	0.0	71.2833	1.0	2.0
4.0	1.0	1.0	0.0	35.0	1.0	0.0	53.1	1.0	1.0
5.0	0.0	3.0	1.0	35.0	0.0	0.0	8.05	0.0	1.0
7.0	0.0	1.0	1.0	54.0	0.0	0.0	51.8625	1.0	1.0
8.0	0.0	3.0	1.0	2.0	3.0	1.0	21.075	0.0	1.0
9.0	1.0	3.0	0.0	27.0	0.0	2.0	11.1333	0.0	1.0
10.0	1.0	2.0	0.0	14.0	1.0	0.0	30.0708	0.0	2.0

将数据导入 MaxCompute。

- PassengerId：用户的 ID 号。
- Survived：乘客是否获救，**1**表示获救，**0**表示没有获救，目标队列（target）。
- Pclass：乘客的社会阶层，**1**表示 Upper，**2**表示 Middle，**3**表示 Lower。

- Sex：乘客的性别，**1**表示男，**0**表示女。
- Age：乘客的年龄。
- Sibsp：乘客在船上的配偶数量或兄弟姐妹数量。
- Parch：乘客在船上的父母或子女数量。
- Fare：乘客的船费。
- Cabin：是否住在独立的房间，**1**表示是，**0**为否。
- Embarked：表示乘客上船的码头距离泰坦尼克出发码头的距离，数值越大表示距离越远。

搭建实验流程



1. 首先将数据集按照7：3进行拆分，一部分作为 titanic 训练集，一部分作为预测集。
2. 将数据进行标准化处理，去除量纲对于数据造成的干扰。
3. 训练数据通过逻辑回归 LR 算法生成模型。
4. 对预测集进行预测。

5. 通过 ROC 曲线和混淆矩阵来对结果进行评估。

评估结果

- **混淆矩阵结果**

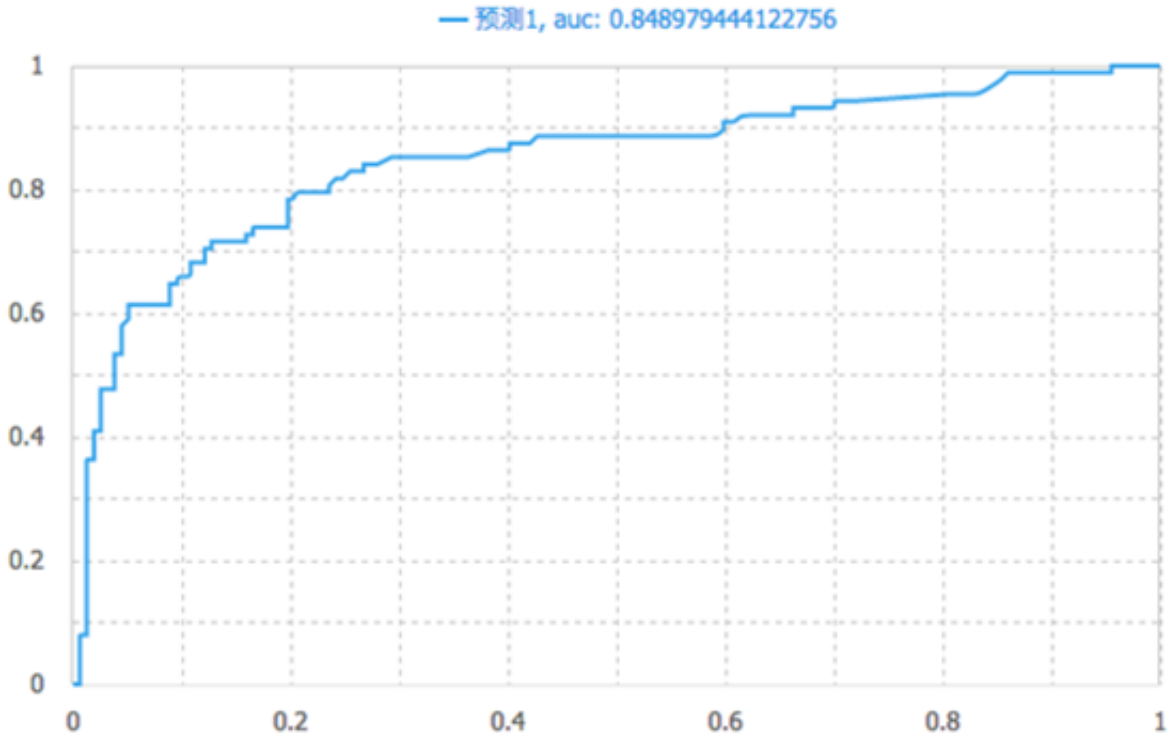
预测1

模型	正确个数	错误个数	总计	正确率	召回率	F1指标
1	65	27	92	70.652%	73.864%	72.222%
0	130	23	153	84.967%	82.803%	83.871%

说明：

1. 正确率：预测正确的正例个数占预测为正例的比例，即 P=TP/(TP+FP)。
2. 召回率：预测正确的正例个数占实际正例的比例，即 R=TP/(TP+FN)。
3. F1 指标：P 和 R 的中权调和平均，即 F1=2PR/(P+R)。当 F1 较高时说明实验方法比较理想。

- **ROC 曲线评估结果**



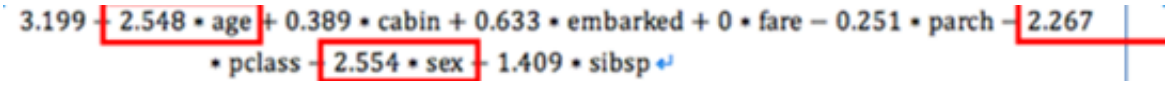
说明：

- TPR=TP/(TP+FN)。
- FPR=FP/(FP+TN)。

- ROC 曲线：
  - Y轴：TPR； X 轴：FPR。
  - (0,1)：FRP=0,TPR=1。FN=0,FP=0，将所有样本都正确分类。
  - (1,0)：即FPR=1,TPR=0,预测结果相反，正变负，负变正。
  - (0,0)：FRP=TPR=0，将所有样本划分为负样本。
  - (1,1)：所有样本划分为正样本。
  - ROC 曲线越接近左上角，该分类器越好。
- y=x，随机猜测，一半正样本，一半负样本。
- AUC：ROC 曲线下面积，应该介于0.5~1之间。AUC 越大，分类器越好。

模型分析

图 1892: 逻辑回归生成模型



根据 Logical regression 的特性，model 输出的是每个特征的线性组合。3.199为常数项，不予考虑。其他系数绝对值越大说明对结果影响越大。通过这一结论得出 age、sex 和 pclass 对于结果影响最大。

根据 sigmoid 函数得出负号系数的绝对值越大其结果的正例可能性越大。所以我们可以得出结论，age、pclass 和 sex 的值越小，目标值越大。

也就是有钱人家的女人和小孩有更大的获救概率。

通过真实数据比对也印证了我们的分析结果：

age	cabin	embarked	fare	parch	pclass	sex	sibsp	survived	prediction_result
26.0	0.0	1.0	7.925	0.0	3.0	0.0	0.0	1.0	1
30.0	0.0	3.0	8.4583	0.0	3.0	1.0	0.0	0.0	0
58.0	1.0	1.0	26.55	0.0	1.0	0.0	0.0	1.0	1
14.0	0.0	1.0	7.8542	0.0	3.0	0.0	0.0	0.0	1
35.0	0.0	1.0	26.0	0.0	2.0	1.0	0.0	0.0	0
15.0	0.0	3.0	8.0292	0.0	3.0	0.0	0.0	1.0	1
19.0	1.0	1.0	263.0	2.0	1.0	1.0	3.0	0.0	0
30.0	0.0	3.0	7.8792	0.0	3.0	0.0	0.0	1.0	1
30.0	0.0	3.0	7.75	0.0	3.0	0.0	0.0	1.0	1
30.0	0.0	2.0	7.2292	0.0	3.0	1.0	0.0	1.0	0
21.0	0.0	1.0	8.05	0.0	3.0	1.0	0.0	0.0	0
3.0	0.0	2.0	41.5792	2.0	2.0	0.0	1.0	1.0	1
30.0	0.0	3.0	15.5	0.0	3.0	1.0	1.0	0.0	0
30.0	0.0	3.0	7.75	0.0	3.0	0.0	0.0	1.0	1
30.0	0.0	2.0	21.6792	0.0	3.0	1.0	2.0	0.0	0