阿里云
专有云大数据版

用户指南

产品版本：V2.1.0

文档版本：20180730

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. 阿里云文档中所有内容，包括但不限于图片、架构设计、页面布局、文字描述，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、发行或公开发表本文档中的内容。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. 阿里云文档中所有内容，包括但不限于图片、页面设计、文字描述，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、发行或公开发表本文档中的内容。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）。

7. 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

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

1 对象存储OSS

1.1 什么是对象存储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。OSS可以被理解成一个即开即用，无限大空间的存储集群。

OSS 将数据文件以对象/文件（Object）的形式上传到存储空间（Bucket）中。OSS 提供的是一个 Key-Value 键值对形式的对象存储服务。用户可以根据Object的名称（Key）唯一地获取该Object的内容。

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。

1.2 基本概念

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。

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。

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DTCenter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

1.3 快速开始

本操作指引您快速完成基本任务，包括进入OSS管理界面、创建存储空间和上传文件。

1.3.1 进入OSS管理界面

对OSS的操作在大数据管家中进行。要对OSS进行操作，需要登录大数据管家，并进入其中的OSS管理界面。

1.3.1.1 获取大数据管家域名

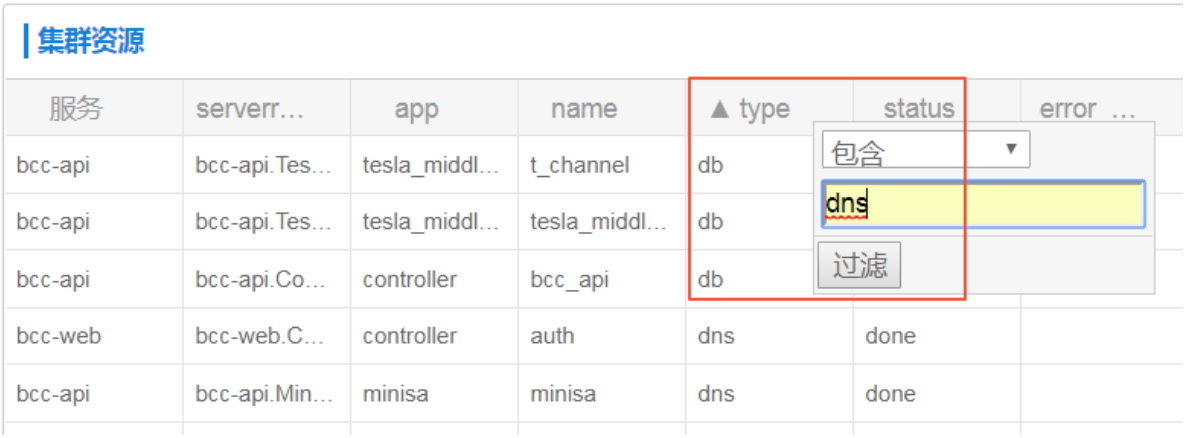
操作步骤

1. 登录天基，如图 *1-1: 天基*所示。

图 1-1: 天基

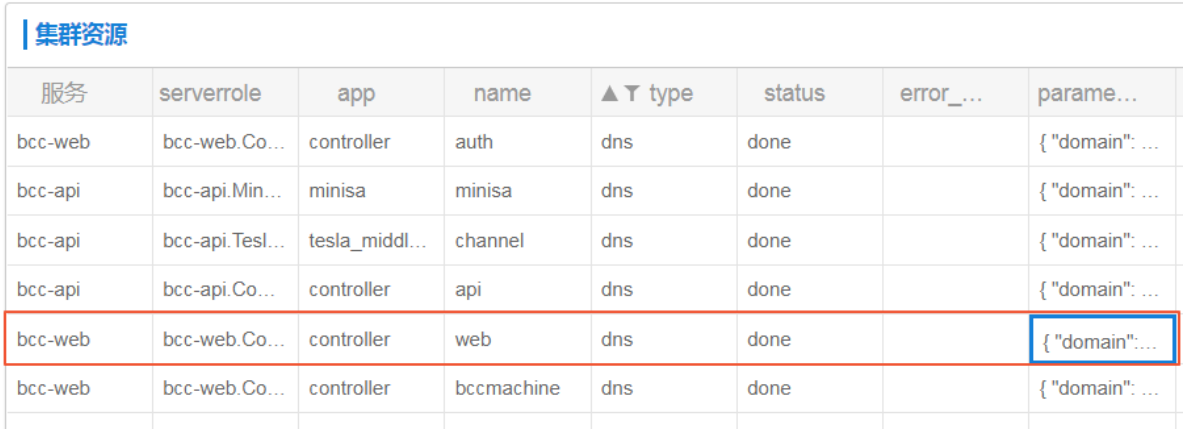
2. 在顶部菜单选择**运维** > **集群运维**，左侧选择**集群**页签，选择**所有集群**，然后在**Project**下拉列表中选择bcc，筛选出大数据管家产品。
3. 单击筛选出的大数据管家产品名称，进入大数据管家的**集群Dashboard**界面。
4. 在**集群资源**列表中，单击列表的**type**表头，以dns进行筛选，如图 *1-2: 筛选条件*所示。

图 1-2: 筛选条件



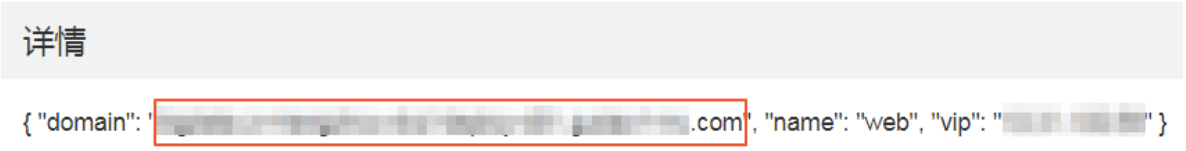
5. 在筛选结果中查找**name**为web的行，如图 *1-3:* 筛选结果所示。

图 1-3: 筛选结果



6. 右键单击该行的**parameters**单元格，并选择**显示更多**，在弹出的**详情**信息框中查看大数据管家的域名，如图 *1-4:* 详情所示。

图 1-4: 详情

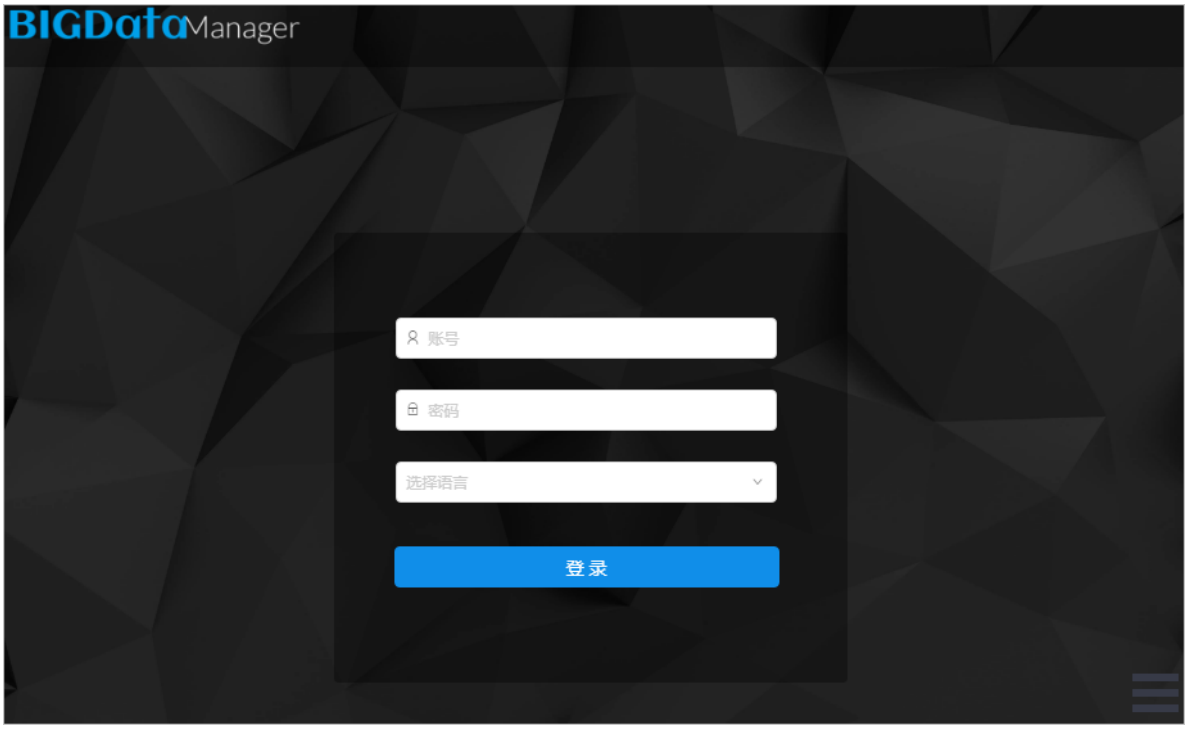


1.3.1.2 进入大数据管家OSS管理界面

操作步骤

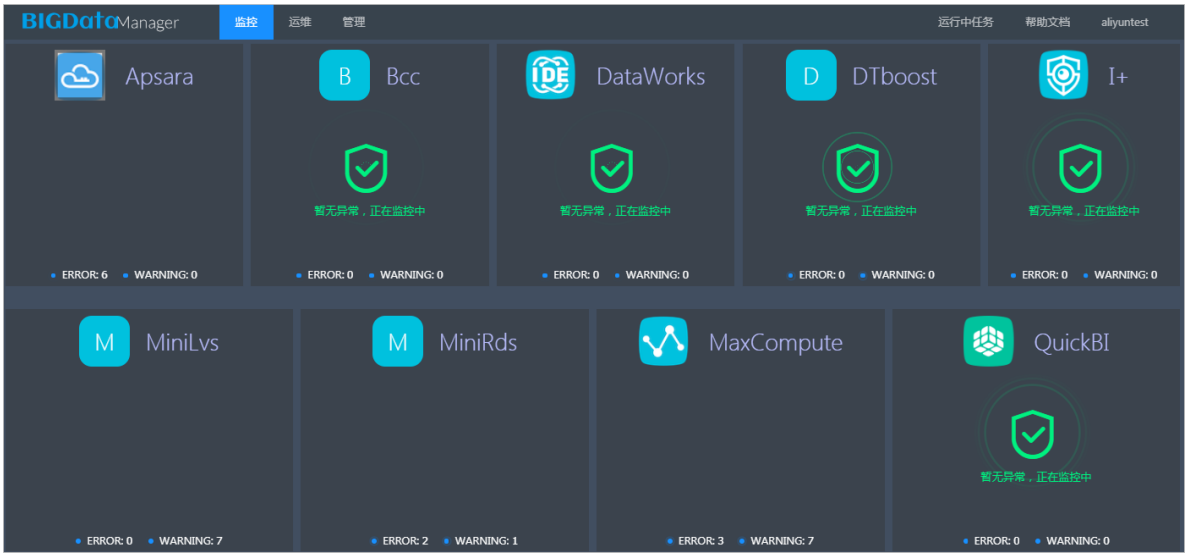
1. 根据查找大数据管家域名中查找的域名，登录大数据管家（初始账号和密码为aliyuntest/aaa111）。

图 1-5: 登录大数据管家

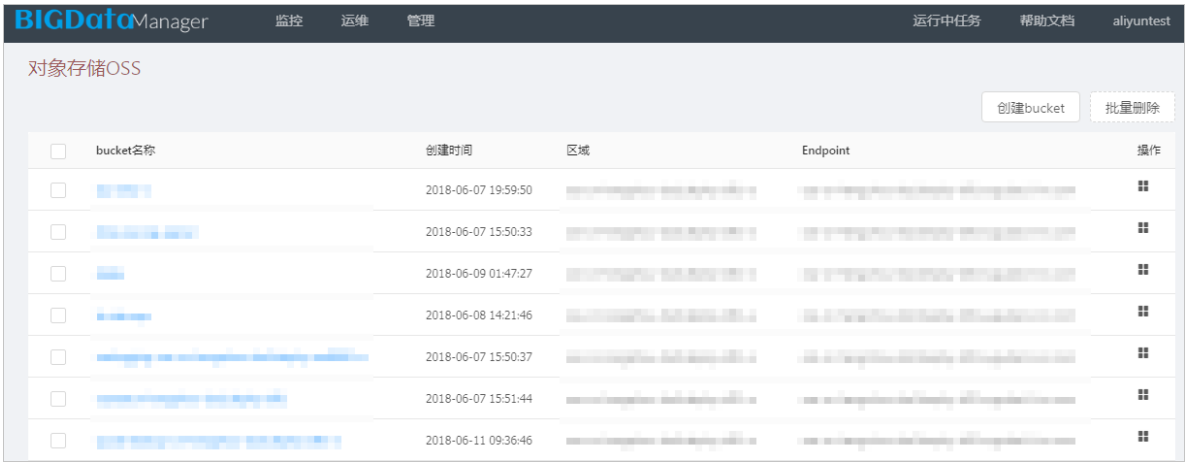


2. 输入账号、密码，并选择语言后，单击**登录**，即可进入大数据管家，如图 *1-6:* 大数据管家页面所示。

图 1-6: 大数据管家页面



3. 在大数据管家页面，单击顶部的**管理**页签，在下拉列表中选择**oss管理**，即可进入OSS管理界面，如下图所示：



1.3.2 创建存储空间

背景信息

在上传任何文件到OSS之前，您需要首先创建存储空间以用来存储文件。存储空间具有各种配置属性，包括其地理区域、访问权限以及其他元数据。

操作步骤

1. 进入大数据管家OSS管理界面。
2. 单击 **创建bucket**，弹出**创建bucket**对话框。

   bucket名称：输入存储空间名称。

   说明：

   - 存储空间的命名必须符合命名规范。
   - 所选定的存储空间名称在阿里云OSS的所有现有存储空间名称中必须具有唯一性。
   - 创建后不支持更改存储空间名称。

3. 单击**OK**，存储空间创建完成。

1.3.3 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. 进入大数据管家OSS管理界面。
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，单击**选择上传文件**。
5. 选择要上传的文件，单击**打开**，返回文件选择对话框，单击**OK**。
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。

1.4 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。

1.4.1 删除存储空间

前提条件

删除存储空间之前请确保其中存储的文件，包括尚未完成的分片上传文件产生的碎片文件全部清空，否则无法删除存储空间。

操作步骤

1. 进入大数据管家OSS管理界面。
2. 单击目标存储空间对应的图标 ，选择 **删除** 。弹出**删除** 对话框。
3. 单击 **OK**，删除该存储空间。
4. 要批量删除存储空间，勾选多个存储空间，单击**批量删除**。

1.5 管理对象（文件）

1.5.1 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` 并以正斜线（/）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正斜线（/）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：

对于任何一个以正斜线（/）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. 进入大数据管家OSS管理界面。
2. 单击目标存储空间名称，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。
5. 在**文件夹名**框中输入该文件夹名称。
6. 单击**OK**保存已创建的文件夹。

1.5.2 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。如果想删除特定的文件，或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. 进入大数据管家OSS管理界面。
2. 单击目标存储空间名称，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. 单击目标文件对应的图标 ，选择**删除** 。弹出**删除**对话框。

注意：

如您的文件夹下文件数量过多，有可能导致删除文件夹失败。

5. 单击 **OK** 删除该文件夹。

2 云盾高级版

2.1 概述

云盾是适用于核心业务应用对外防护的互联网化防护体系，能够为用户提供Web漏洞发现/修复、主机漏洞发现/修复、主机防入侵的实时防护能力。通过这些核心安全信息的分析展现，安全管理员不仅能够了解安全状况，还可以借助安全数据分析引擎开放的自定义分析界面对已有安全数据进行场景化分析，实现安全分析能力的灵活定制。

2.2 配置要求

本地PC需要满足如表 *2-1:* 配置要求表中要求才可以正常登录云盾安全中心。

表 2-1: 配置要求表

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系统

2.3 登录和注销

2.3.1 获取云盾安全中心地址

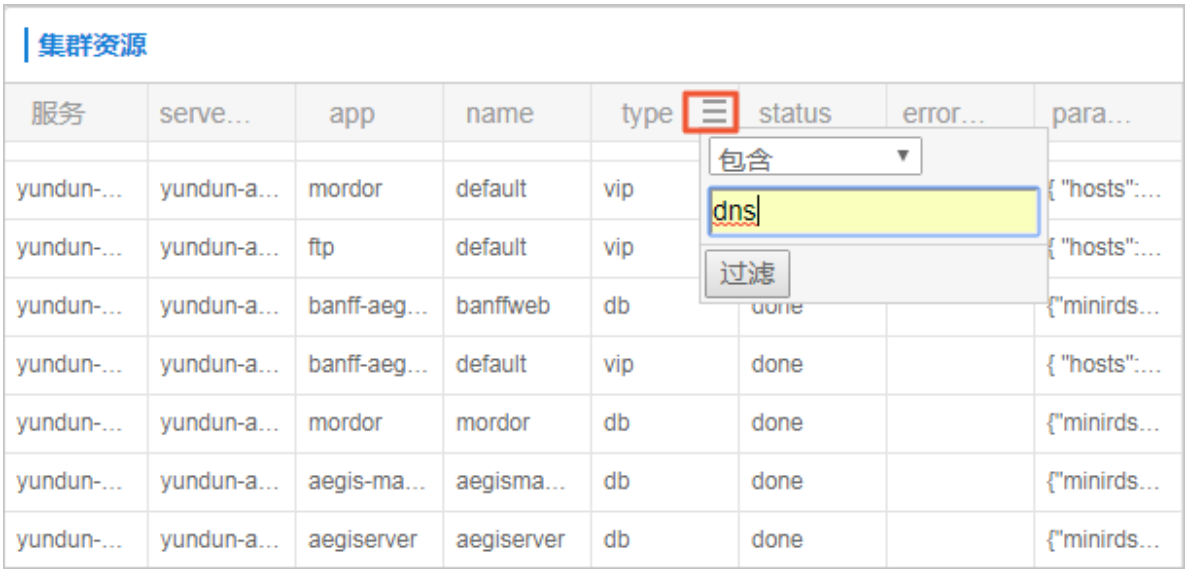
操作步骤

1. 登录天基，选择**运维** > **集群运维**，如图 *2-1:* 天基所示。

图 2-1: 天基

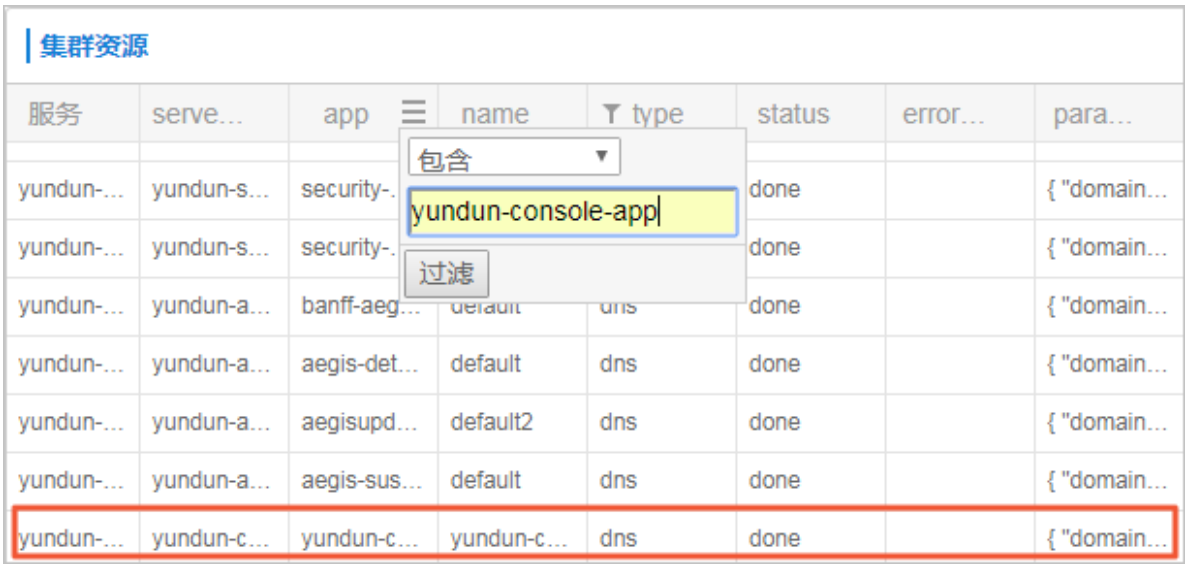
2. 在**Project**下拉列表中，选择yundun-advance，筛选出云盾。
3. 单击云盾集群名称，进入云盾的**集群Dashboard**界面。
4. 定位到**集群资源**区域，单击**type**表头的☰，以dns进行过滤，如图 *2-2: type*筛选所示。

图 2-2: type筛选



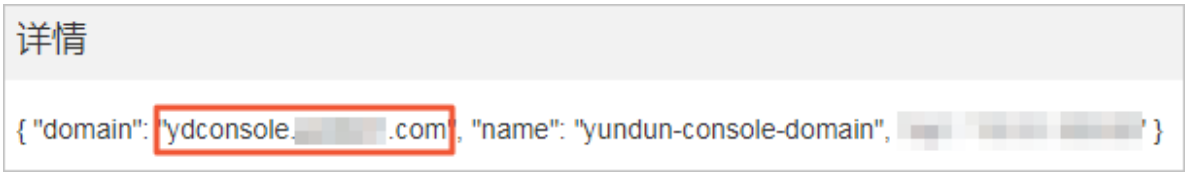
5. 单击**app**表头的☰，以yundun-console进行过滤，如图 *2-3: app*筛选所示。

图 2-3: app筛选



6. 右键单击该行的**parameters**单元格，并选择**显示更多**，在弹出的**详情**信息框中查看云盾的登录地址，如图 *2-4:* 云盾的登录地址所示。

图 2-4: 云盾的登录地址



2.3.2 登录云盾安全中心

前提条件

云盾安全中心的网站地址参见获取云盾安全中心地址，云盾安全中心的登录名和密码请联系部署人员获取。

操作步骤

1. 打开Chrome浏览器。
2. 在地址栏中，输入云盾安全中心的网站地址（例如：http://云盾安全中心网站地址），按**Enter**。
3. 输入已创建的云盾安全中心用户的登录账号、密码及验证码。
4. 单击**登录**。

2.3.3 退出云盾安全中心

- 在**云盾安全中心**页面，单击页面右上角的**退出**，即可从云盾安全中心注销。

2.4 云主机安全

2.4.1 主机安全总览

主机安全总览对主机整体安全情况进行概要性展示，以便安全管理员快速了解和掌握当前安全情况。

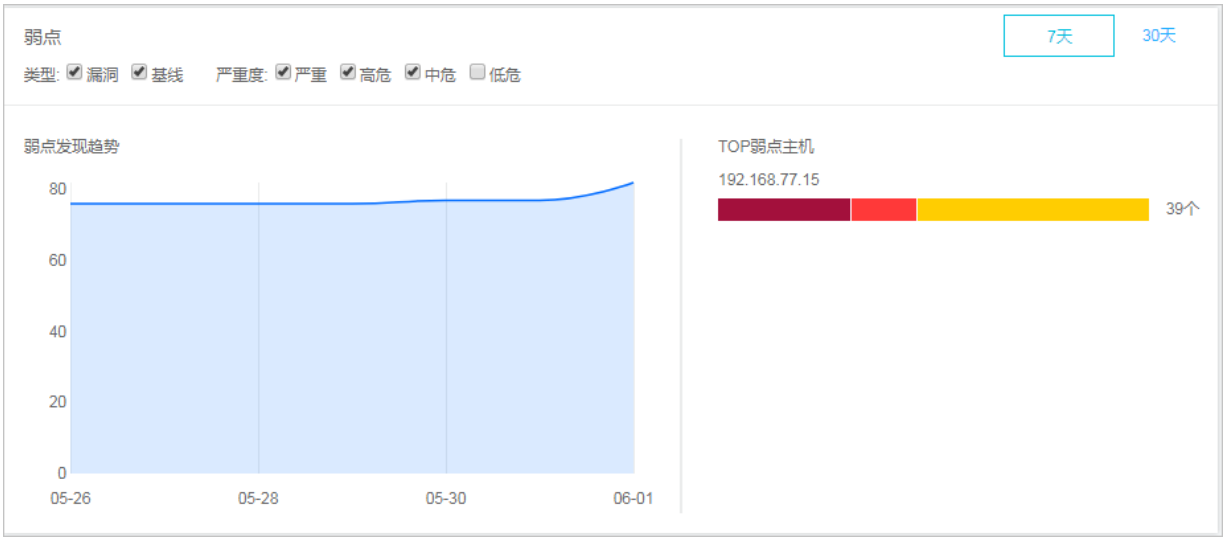
主机安全总览包括总览、弱点、事件、服务器保护状态、最近重要弱点和事件。

总览

整体展现主机环境各类型的安全弱点（待处理漏洞、基线配置不当）和安全事件（异常登录、网站后门、主机异常）的数量。

弱点

展示主机存在的安全弱点趋势。安全弱点如果不修复，存在安全隐患，具体修复建议和操作可以参考安全预防。

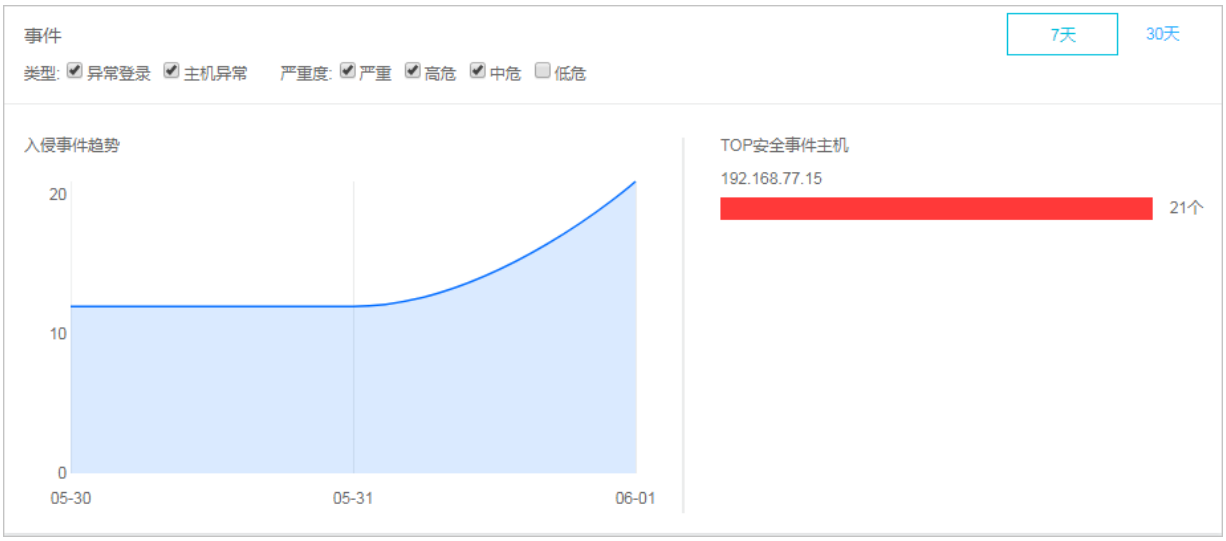


- 上面为过滤条件，通过类型、严重程度和展示时间，生成展示结果。
- 左侧为弱点发现趋势，展示一段时间内主机的弱点趋势。

- 右侧为TOP弱点主机，展示存在弱点数量最多的主机。

事件

展示主机存在的安全事件趋势。安全事件表明目前主机已经发现的安全入侵行为，具体操作可以参考入侵检测。



- 上面为过滤条件，通过类型、严重程度和展示时间，生成展示结果。
- 左侧为入侵事件趋势，展示一段时间内主机受到入侵的趋势。
- 右侧为TOP安全事件主机，展示受到入侵最多的主机。

服务器保护

查看目前主机中正在受到保护的主机数量和离线数量。

最近重要弱点和事件

展示最近重要的主机安全弱点和安全事件，单击连接可以查看具体详细情况。

2.4.2 主机列表

以主机维度，展示各主机的安全状况信息。

2.4.2.1 管理主机列表

在主机列表页面，可以查看安骑士已防护的服务器的状态。

背景信息

服务器保护状态分为在线、离线、暂停保护三种。

- **在线**：安骑士为该服务器提供全面的安全防护。
- **离线**：安骑士服务端无法与该服务器的客户端正常连通，无法提供安全防护功能。
- **暂停保护**：暂时关闭安骑士对该服务器的防护，具体参见暂停保护操作。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全 > 主机列表**。
3. （可选）搜索服务器。

   如果想查看某台服务器的安全状态，也可以在搜索框中输入该服务器的IP，并单击**搜索**，即可快速查看该服务器的详细信息和安全信息。
4. 查看服务器的保护状态和具体安全信息。

   单击右上角的 ⚙，设置服务器具体显示哪些信息列。信息主要分为以下几类。

类别	信息
服务器基本信息	• 服务器IP/名称

类别	信息
	• 标签 • 操作系统 • 地域
保护状态	保护状态
安全预防	• 漏洞 • 基线
入侵检测	• 异常登录 • 网站后门 • 主机异常
主机指纹	• 进程数 • 端口数 • Root账号/所有账号

5. 管理服务器。

功能	操作说明
更改分组	勾选服务器，单击**更改分组**，更改服务器所在的分组。分组具体说明参见管理分组。
设置标签	勾选服务器，单击**编辑标签**，设置服务器标签信息。
一键安全检查	勾选服务器，单击**一键安全检查**，可以选择从多维度对服务器进行安全检查。
删除非阿里云机器	勾选**非阿里云**的服务器，单击**更多操作** > **删除非阿里云机器**。
暂停保护	勾选**在线**状态的服务器，单击**更多操作** > **暂停保护**，暂时关闭安骑士对该服务器的防护，降低该服务器的资源消耗。
开启保护	勾选**暂停保护**状态的服务器，单击**更多操作** > **开启保护**，开启安骑士对该服务器的防护。

2.4.2.2 管理分组

为了方便对特定服务器进行安全管控，可以对服务器进行分组，通过分组的维度查看安全事件。

背景信息

未进行分组时，所有的服务器都在**未分组**中。当您删除某个分组时，该分组中的服务器也将默认移入**未分组**中。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **主机列表**。
3. 管理子分组。

- 新建子分组。

  单击**所有资源**或者子分组右侧的添加按钮，输入子分组名称并单击**确认**。

  > **说明**：
  > 最多支持三级子分组。

- 修改子分组。

  单击子分组右侧的修改按钮，输入子分组名称并单击**确认**。

- 删除子分组。

  单击子分组右侧的删除按钮，在弹出的确认框中单击**确认**。

  > **说明**：
  > 删除后该分组中的服务器默认移入**未分组**。

4. 给服务器进行分组。
   a) 在右侧的服务器列表中，勾选服务器。
   b) 单击**更换分组**。
   c) 在弹出窗口的下拉菜单中选择分组。
   d) 单击**确认**。
5. 切换分组排序。

单击**分组排序**，可以把优先级高的分组移动到上面。

2.4.3 安全预防

安全预防提供漏洞管理和基线检查功能，用于发现服务器中存在的漏洞和风险点，并提供修复建议。

2.4.3.1 漏洞管理

漏洞管理扫描出服务器中存在的漏洞，并提供漏洞修复方法。

2.4.3.1.1 管理Linux软件漏洞

比对CVE官方漏洞库，自动检测您服务器上安装的软件版本是否存在漏洞，并向您推送漏洞消息。针对检测到的漏洞，提供漏洞修复指令和验证功能。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **安全预防** > **漏洞管理**，选择**Linux软件漏洞**页签。
3. 查看所有漏洞。

   通过漏洞搜索和筛选功能，快速定位到具体漏洞，如图 *2-5: 过滤漏洞*所示。

   图 2-5: 过滤漏洞

4. 单击漏洞名称，进入漏洞详情页，查看漏洞详细信息和影响资产。

   通过搜索和筛选功能，快速展示影响资产，如图 *2-6: 筛选影响资产*所示。

图 2-6: 筛选影响资产

5. 根据漏洞影响情况，选择相应的处理方式，如表 2-2: 漏洞操作方式所示。

表 2-2: 漏洞操作方式

操作	说明
生成修复命令	自动生成修复漏洞的指令，然后登录服务器运行该指令来修复漏洞。
一键修复	直接修复漏洞。
已重启并验证	如果修复漏洞需要重启服务器才能生效，必须等待漏洞修复状态变为**修复成功待重启**后，重启该服务器，然后单击**已重启并验证**完成修复。
忽略	可忽略该漏洞，系统将不再上报并告警此服务器上被忽略的漏洞。
验证	修复漏洞后，单击**验证**，一键验证该漏洞是否已修复成功。 如果未进行手动验证，漏洞修复成功后 48 小时内系统会自动去验证。

对于影响资产，可以进行单个操作或多个批量操作。

- 单个操作：在**操作**列，对单个受影响的服务器进行处理。
- 批量操作：勾选一个或多个需要处理的服务器，使用列表下方的批量操作按钮进行批量处理。

2.4.3.1.2 管理Windows系统漏洞

比对微软官方补丁更新，自动检测您服务器上的补丁是否已更新，并向您推送漏洞消息。针对重大漏洞更新，提供自动检测和修复功能。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **安全预防** > **漏洞管理**，选择**Windows系统漏洞**页签。
3. 查看所有漏洞。

通过漏洞搜索和筛选功能，快速定位到具体漏洞，如图 2-7: 过滤漏洞所示。

图 2-7: 过滤漏洞

4. 单击漏洞名称，进入漏洞详情页，查看漏洞详细信息和影响资产。

   通过搜索和筛选功能，快速展示影响资产，如图 2-8: 筛选影响资产所示。

图 2-8: 筛选影响资产

5. 根据漏洞影响情况，选择相应的处理方式，如表 2-3: 漏洞操作方式所示。

表 2-3: 漏洞操作方式

操作	说明
立即修复	直接修复漏洞。系统会在云端缓存一份Windows官方补丁文件，您的Windows系统服务器会直接下载云端的补丁并完成自动更新。
忽略	可忽略该漏洞，系统将不再上报并告警此服务器上被忽略的漏洞。
验证	修复漏洞后，单击**验证**，一键验证该漏洞是否已修复成功。
已重启并验证	如果修复漏洞需要重启服务器才能生效，必须等待漏洞修复状态变为**修复成功待重启**后，重启该服务器，然后单击**已重启并验证**完成修复。

对于影响资产，可以进行单个操作或多个批量操作。

- 单个操作：在**操作**列，对单个受影响的服务器进行处理。
- 批量操作：勾选一个或多个需要处理的服务器，使用列表下方的批量操作按钮进行批量处理。

2.4.3.1.3 管理Web-CMS漏洞

Web-CMS 漏洞功能通过及时获取最新的漏洞预警和相关补丁，并通过云端下发补丁更新，实现漏洞快速发现、快速修复的功能。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **安全预防** > **漏洞管理**，选择**Web-CMS漏洞**页签。
3. 查看所有漏洞。

   通过漏洞搜索和筛选功能，快速定位到具体漏洞，如图 *2-9:* 过滤漏洞所示。

   图 2-9: 过滤漏洞

4. 单击漏洞名称，进入漏洞详情页，查看漏洞详细信息和影响资产。

   通过搜索和筛选功能，快速展示影响资产，如图 *2-10:* 筛选影响资产所示。

   图 2-10: 筛选影响资产

5. 根据漏洞影响情况，选择相应的处理方式，如表 *2-4:* 漏洞操作方式所示。

表 2-4: 漏洞操作方式

操作	说明
立即修复	系统将替换服务器上存在漏洞的Web文件以修复Web-CMS漏洞。 **说明**： 修复Web-CMS漏洞前，建议备份该漏洞相关的Web文件，Web文件的具体路径可参考漏洞处理说明栏中的路径。
忽略	可忽略该漏洞，系统将不再上报并告警此服务器上被忽略的漏洞。
验证	修复漏洞后，单击**验证**，一键验证该漏洞是否已修复成功。 如果未进行手动验证，漏洞修复成功后 48 小时内系统会自动去验证。
回滚	对于已修复完成的漏洞，单击**回滚**可进行漏洞回滚，还原修复前的Web文件。

对于影响资产，可以进行单个操作或多个批量操作。

- 单个操作：在**操作**列，对单个受影响的服务器进行处理。
- 批量操作：勾选一个或多个需要处理的服务器，使用列表下方的批量操作按钮进行批量处理。

2.4.3.1.4 管理其他漏洞

自动检测服务器上的Redis未授权访问漏洞、STRUTS-052命令执行漏洞等漏洞，并推送漏洞消息。同时，支持漏洞的修复验证操作。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全 > 安全预防 > 漏洞管理**，选择**其他漏洞**页签。
3. 查看所有漏洞。

   通过漏洞搜索和筛选功能，快速定位到具体漏洞，如图 *2-11:* 过滤漏洞所示。

图 2-11: 过滤漏洞

4. 单击漏洞名称，进入漏洞详情页，查看漏洞详细信息和影响资产。

   通过搜索和筛选功能，快速展示影响资产，如图 *2-12: 筛选影响资产*所示。

图 2-12: 筛选影响资产

5. 根据漏洞影响情况，选择相应的处理方式，如表 *2-5: 漏洞操作方式*所示。

   其他漏洞需要按照修复说明手动修复。

表 2-5: 漏洞操作方式

操作	说明
忽略	可忽略该漏洞，系统将不再上报并告警此服务器上被忽略的漏洞。
验证	手动修复漏洞后，单击**验证**，一键验证该漏洞是否已修复成功。 如果未进行手动验证，漏洞修复成功后 48 小时内系统会自动去验证。

对于影响资产，可以进行单个操作或多个批量操作。

- 单个操作：在**操作**列，对单个受影响的服务器进行处理。
- 批量操作：勾选一个或多个需要处理的服务器，使用列表下方的批量操作按钮进行批量处理。

2.4.3.1.5 设置漏洞管理策略

漏洞管理设置允许您开启/关闭不同类型漏洞的自动检测，有选择性地对指定服务器应用漏洞检测，对已失效漏洞设置自动删除周期，和配置漏洞白名单。

背景信息

漏洞白名单用于彻底忽略某些漏洞，您可以在漏洞列表下批量添加漏洞至白名单。添加成功后，系统将不再去检测漏洞白名单中的漏洞。使用漏洞管理设置可以维护漏洞白名单。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全 > 安全预防 > 漏洞管理**。
3. 单击右上角的**漏洞管理设置**，配置相关策略，如图 *2-13:* 漏洞管理设置所示。

图 2-13: 漏洞管理设置

- 选择需要操作的漏洞类型，单击切换开关，开启/关闭该漏洞检测。

- 选择需要操作的漏洞类型，单击**管理**，配置应用该漏洞检测的服务器。
- 勾选需要扫描的漏洞等级：严重、高危、中危、低危。
- 选择失效漏洞自动删除周期：7天、30天、90天。

> **说明**：
> 对于检测出来的漏洞不做任何处理的话，默认该告警失效，并在指定周期后自动删除。

- 在漏洞白名单下勾选相应漏洞，单击**移除**，重新启用对该漏洞的检测和告警。

2.4.3.2 基线检查

检测服务器上的系统、数据库、账号配置存在的风险点，并针对所发现的问题项提供修复建议。

2.4.3.2.1 基线检查介绍

安骑士基线检查功能自动检测服务器上的系统、数据库、账号配置存在的风险点，并针对所发现的问题项提供修复建议。

检测原理

基线检查功能自动检测服务器上的系统、权限、账号、数据库等配置存在的风险点，并提供修复建议。

检测周期

默认每三天进行一次全面自动检测，自动检测在凌晨0到6点间完成。您可以在在安全设置页面设置检测周期和检测发生时间。

注意事项

某些检测项，例如：MySQL弱密码检测、SQLSERVER弱密码检测，会采用尝试登录方式进行检查，会占用一定的服务器资源以及生产较多的登录失败记录，这些项目是默认不开启的。如果需要这些功能，请确认上述风险后，在基线检查设置中勾选这些项目。

检测内容

表 2-6: 检测内容

分类	检测项
CIS基线检测	Centos7基线检测
	Tomcat7基线检测
弱密码检测	PostgreSQL弱密码检测

分类	检测项
	SSH弱密码检测
	FTP匿名登录检测
	SQLServer登录密码检测
	MySQL弱密码检测
	RDP弱密码检测
	FTP弱密码检测
系统	组策略
	文件监控
	基线策略
	注册表
账号	系统账号安全
数据库	Redis合规检查

2.4.3.2.2 管理基线检查

通过基线检查功能，查看和修复服务器上的配置风险项。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **安全预防** > **基线检查**，进入基线检查页面。
3. 查看所有风险项。

   通过风险搜索和筛选功能，快速定位到具体风险。

4. 单击风险名称，可查看该风险详情及相关修复建议。

加入白名单：如果发现风险对服务器没有影响，后续也不需要检查该风险，可以单击右上角的**加入白名单**。

5. 参考修复建议，在您的对应服务器上进行修复。

   - 修复风险后，您可以单击**验证**，一键验证该风险是否已修复成功。如果您未进行手动验证，风险修复成功后 72 小时内安骑士会进行自动验证。
   - 如果不需要修复，您可以单击**忽略**，忽略该风险，安骑士将不再上报并告警此服务器上的这个风险项。

2.4.3.2.3 设置基线检查策略

根据实际业务情况设置基线检测项，检测周期、检测风险等级。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **安全预防** > **基线检查**，进入基线检查页面。
3. 单击**基线检查设置**，可以新建或者修改策略。
4. 单击**添加**，配置策略。

   选择某一策略，单击**编辑**修改策略；单击**删除**删除策略。
5. 管理白名单。

2.4.4 入侵检测

入侵检测提供异常登录、网站后门和主机异常等功能，用于检测发现服务器受到的入侵行为。

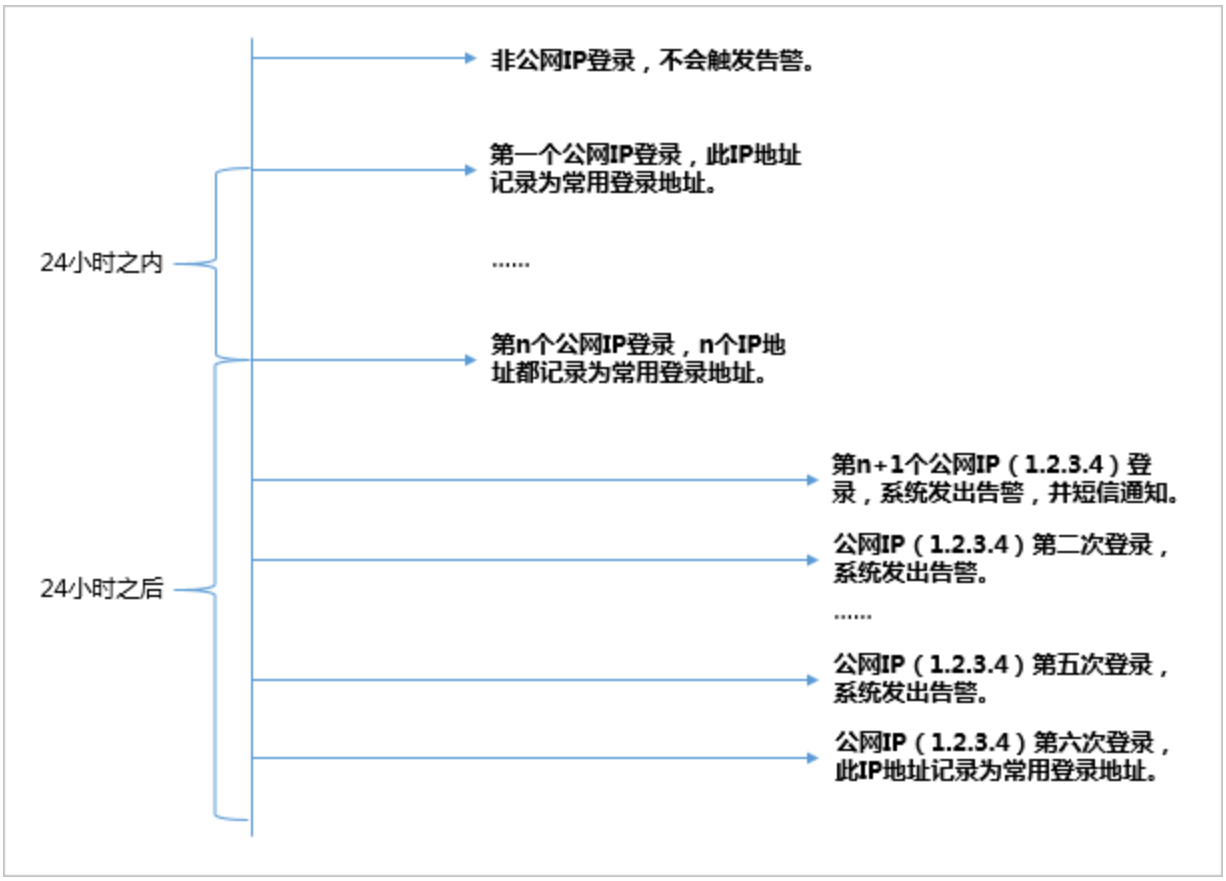
2.4.4.1 异常登录

在安骑士管理控制台中的**异常登录**页面，您可以查看服务器上每次登录行为有异常的登录IP、账号、时间，包括异地登录告警及非法登录IP、非法登录时间、非法登录账号的登录行为告警。

安骑士Agent通过定时收集您服务器上的登录日志并上传到安骑士服务器端，在安骑士服务器端进行分析和匹配。如果发现在非常用登录地或非法登录IP、非法登录时间、非法登录账号的登录成功事件，将会触发事件告警。

异地登录告警策略如图 *2-14:* 异地登录策略所示。

图 2-14: 异地登录策略



说明：

短信告警方式：可以在**设置** > **告警配置**中，选择**登录安全** > **异常登录**通知项目的告警方式（可配置为短信、邮件、及站内信方式，默认通过全部方式进行告警）。

针对机器设置合法登录IP、合法登录时间、合法登录账号，在上述合法登录IP、合法登录事件、合法登录账号之外的登录行为均提供告警，判断优先级高于异地登录判断。

2.4.4.1.1 查看异常登录

查看异常登录告警，包括异地登录、爆破登录、非法IP登录、非法账号登录和非法时间登录等告警。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **入侵检测** > **异常登录**。
3. 查看所有异常登录信息。

   通过搜索和筛选功能，快速定位到具体异常登录信息，如图 *2-15: 查找异常登录信息*所示。

图 2-15: 查找异常登录信息

4. 处理异常登录信息。

   选择异常登录信息，判断是否存在误报。

   - 如果是误报，直接单击**标记为已处理**。
   - 如果是非法入侵，对服务器进行安全加固（例如设置复杂密码，修复服务器漏洞，修复基线检查的风险点，设置黑/白名单等），完成后单击**标记为已处理**。

2.4.4.1.2 设置登录安全策略

设置登录安全策略，包括常用登录地、合法登录IP、合法登录时间和合法账号。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **入侵检测** > **异常登录**。
3. 在**异常登录**页面右上角，单击**登录安全设置**。
4. 设置常用登录地。
   a) 单击**添加**。
   b) 在下拉菜单中选择常用登录地。
   c) 设置常用登录地对哪些服务器生效。
      - **全部资源**中可以选择具体服务器。
      - **分组资产**中可以根据分组信息选择服务器。
   d) 单击**确定**，完成新增规则操作。
   e) 选择具体规则，单击**编辑**修改规则。
   f) 选择具体规则，单击**删除**删除规则。
5. 设置合法登录IP。
6. 设置合法登录时间。
7. 设置合法账号。

2.4.4.2 网站后门

安骑士采用本地查杀 + 云查杀体系，拥有定时查杀和实时防护扫描策略，支持检测常见的PHP、JSP等后门文件类型，并提供一键隔离功能。

安骑士通过检测您服务器上的Web目录中的文件，判断是否为Webshell木马文件。如果发现您的服务器存在网站后门文件，将会触发告警信息。

安骑士网站后门检测采用动态检测及静态检测两种方式：

- **动态检测**： 一旦 Web 目录中的文件发生变动，安骑士将会针对变动的内容进行动态检测。
- **静态检测**： 每天凌晨，安骑士将会扫描整个 Web 目录进行静态检测。

> **说明**：
> 默认情况下，安骑士防护的所有服务器均开启静态检测。如果需要设置特定服务器开启静态检测，可以在**设置** > **安全设置**中的**木马查杀**区域，单击**周期检查Web目录**右侧的**管理**设置需要进行静态检测的服务器。

2.4.4.2.1 管理网站后门

查看和隔离网站后门文件。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **入侵检测** > **网站后门**。
3. 选择资产，查看已发现的网站后门文件记录，如图 *2-16: 选择资产*所示。

图 2-16: 选择资产

4. 处理网站后门文件。
   - **隔离**：对发现的木马文件进行隔离操作，支持批量处理。
   - **恢复**： 如果错误隔离了某些文件，您可以单击**恢复**，将此文件恢复。
   - **忽略**： 忽略该木马文件后，安骑士将不再对此文件提示风险告警。

> **说明**：

安骑士不会将您服务器上的木马文件直接删除，只会将该文件转移到隔离区，在您确认该文件为信任文件后可通过恢复功能将该文件恢复，并且安骑士将不再对此文件进行告警。

2.4.4.3 主机异常

查看在服务器上检测到的异常进程行为和恶意进程等。

2.4.4.3.1 管理主机异常

查看服务器上的主机异常告警，并修复相应问题。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **入侵检测** > **主机异常**。
3. 选择资产，查看已发现的主机异常事件。
4. 根据主机异常事件影响情况，选择相应的处理方式，如表 *2-7:* 处理主机异常事件所示。

表 2-7: 处理主机异常事件

操作	说明
一键修复	直接修复漏洞。
忽略本次	如果该事件不影响服务器安全，可以选择忽略本次告警。
确认事件	确认该事件。
标记为误报	如果本次告警为误报，可以标记为误报。
查看	查看本次告警详细信息。

2.4.5 主机指纹

主机指纹功能定期收集并记录服务器上的运行进程、系统账号、开放端口和软件版本，帮助您全面了解资产的运行状态和进行回溯分析。

2.4.5.1 管理监听端口

定期收集服务器的对外端口监听信息，用于清点端口。

背景信息

监听端口应用场景包括：

- 清点一个端口被哪些服务器监听。
- 清点一台服务器开通了哪些端口。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **主机指纹**，选择**监听端口**页签。
3. 查看所有已开放端口、端口对应的网络协议、和开放这些端口的主机数。

   您可以通过端口号或进程名搜索，快速查找端口。
4. 单击端口号，查看对应的资产、协议等详细信息。

2.4.5.2 管理运行进程

定期收集服务器的进程信息，用于清点进程。

背景信息

运行进程应用场景包括：

- 清点一个进程被哪些服务器运行。
- 清点一台服务器运行了哪些进程。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **主机指纹**，选择**运行进程**页签。
3. 查看所有运行中进程和运行这些进程的主机数。

   您可以使用进程名或运行用户进行搜索。
4. 单击进程名，查看进程对应资产、路径、启动参数等详细信息。

2.4.5.3 管理账号信息

定期收集服务器的账号信息，用于清点账号。

背景信息

账号信息应用场景包括：

- 清点一个账号被哪些服务器创建。
- 清点一台服务器创建了哪些账号。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **主机指纹**，选择**账号信息**页签。
3. 查看所有已登录的系统账号和使用这些账号的主机数。

您可以使用账号名进行搜索。

4. 单击账号名，查看对应资产、ROOT权限、用户组等详细信息。

2.4.5.4 管理软件版本

定期收集服务器的软件版本信息，用于清点软件资产。

背景信息

软件版本应用场景包括：

- 清点非法软件资产（非法安装）。
- 清点版本过低软件资产。
- 漏洞爆发时快速定位受影响资产范围。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **主机指纹**，选择**软件版本**页签。
3. 查看所有使用中的软件和使用这些软件的主机数。

   您可以使用软件名、版本名或软件安装目录进行搜索。
4. 单击软件名，查看其对应资产、软件版本等信息。

2.4.5.5 设置主机指纹刷新频率

通过主机指纹设置功能，选择运行进程、系统账号、开放端口、软件版本数据的收集刷新频率。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **主机指纹**，单击**主机指纹设置**。
3. 在相应项目的下拉菜单中，选择刷新频率。
4. 单击**确认**，完成配置。

2.4.6 日志检索

全面支持主机的进程启动、网络连接、系统登录流水等日志查询，便于您进行安全事件分析、操作日志审计。

2.4.6.1 日志检索介绍

云主机安全提供主机检索功能，将散落在阿里云各系统中的日志集中管理，便于您在出现主机问题时一站式搜索定位问题根源。

支持180天以内的日志存储，并提供30天以内的日志查询。

功能特性

日志检索功能具备以下特性：

- **一站式日志检索平台**，集中查询专有云各产品日志，单一接口，便于问题溯源。
- **日志功能的SaaS化**，无需进行额外安装部署，即可查询云环境中所有服务器日志。
- 支持TB级数据检索，以及50个维度的数据逻辑（布尔表达式）组合，可以秒级展示日志全文检索结果。
- 支持日志投递，允许您将安全日志导入到日志服务做进一步分析。

应用场景

日志检索帮助您实现以下需求：

- **安全事件分析**：主机发生安全事件后，通过日志功能进行调查，评估资产受损范围和影响。
- **操作审计**：对主机的操作日志进行审计，对高危操作和严重问题进行细粒度排查。

可检索日志类型

表 2-8: 日志源

日志源	说明
登录流水	系统登录成功的日志记录。
暴力破解	系统登录失败的日志记录。
进程快照	某一时刻主机上的进程运行信息。
端口监听快照	某一时刻主机上的监听端口信息。
账号快照	某一时刻主机上的账号登录信息。
进程启动日志	主机上进程启动的相关信息。

日志源	说明
网络连接日志	主机对外主动连接的日志。

2.4.6.2 查询日志

搜索和查看主机日志。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **日志检索**。
3. 设置搜索条件。

   表 2-9: 搜索条件说明

搜索项	说明
请选择日志源	支持的日志源，具体内容参考表 *2-10:* 日志源。
请选择字段	各日志源支持的字段，具体内容参考表 *2-10:* 日志源。
关键字	需要搜索的字段具体关键字。
语法逻辑	语法逻辑包括and、or、not，具体说明参考表 *2-18:* 语法逻辑说明。
+	在一个搜索条件（一个日志源）下增加多个逻辑判断。
增加一组	增加多个搜索条件（不同的日志源）。

4. 单击**搜索**，查看搜索结果。

   - **重置**：如果不需要原来设置的搜索条件，单击**重置**，搜索条件回到初始状态。
   - **保存搜索逻辑**：如果以后需要重用这个搜索条件，单击**保存搜索逻辑**进行保存。
   - **已保存的搜索**：如果需要执行以前保存的搜索条件，单击**已保存的搜索**，选择已有的搜索条件执行。

2.4.6.3 各日志源字段说明

本文介绍了态势感知日志检索功能可以采集并供检索的原始日志类型和字段说明。

日志检索功能支持您查询下表列述的日志源。您可以单击一个日志源查看其支持的字段信息。

表 2-10: 日志源

日志源	说明
登录流水	系统登录成功的日志记录。
暴力破解	系统登录失败的日志记录。
进程快照	某一时刻主机上的进程运行信息。
端口监听快照	某一时刻主机上的监听端口信息。
账号快照	某一时刻主机上的账号登录信息。
进程启动日志	主机上进程启动的相关信息。
网络连接日志	主机对外主动连接的日志。

登录流水

登录流水查询支持以下字段：

表 2-11: 登录流水支持字段

字段	数据类型	说明
uuid	string	客户端编号
IP	string	IP地址
warn_ip	string	登录来源IP
warn_port	string	登录端口
warn_user	string	登录用户名
warn_type	string	登录类型
warn_count	string	登录次数
time	datetime	登录时间

暴力破解

暴力破解查询支持以下字段：

表 2-12: 暴力破解支持字段

字段	数据类型	说明
uuid	string	客户端编号

字段	数据类型	说明
IP	string	IP地址
warn_ip	string	攻击来源IP
warn_port	string	破解端口
warn_user	string	破解用户名
warn_type	string	类型
warn_count	string	破解次数
time	datetime	破解时间

进程启动日志

进程启动日志查询支持以下字段：

表 2-13: 进程启动日志支持字段

字段	数据类型	说明
uuid	string	客户端编号
IP	string	IP地址
pid	string	进程ID
groupname	string	用户组
ppid	string	父进程ID
uid	string	用户ID
username	string	用户名
filename	string	文件名
pfilename	string	父进程文件名
cmdline	string	命令行
filepath	string	进程路径
pfilepath	string	父进程路径
time	datetime	启动时间

端口监听快照

端口监听快照查询支持以下字段：

表 2-14: 端口监听快照支持字段

字段	数据类型	说明
uuid	string	客户端编号
IP	string	IP地址
src_port	string	监听端口
src_ip	string	监听IP
proc_path	string	进程路径
PID	string	进程ID
proc_name	string	进程名
proto	string	协议
time	datetime	数据获取时间

账号快照

账号快照查询支持以下字段：

表 2-15: 账号快照支持字段

字段	数据类型	说明
uuid	string	客户端编号
IP	string	IP地址
perm	string	是否拥有root权限
home_dir	string	home目录
warn_time	string	密码到期提醒时间
groups	string	用户属于的组
login_ip	string	最后一次登录的IP地址
last_chg	string	密码最后修改时间
shell	string	Linux的shell命令
domain	string	Windows域
tty	string	登录的终端
account_expire	string	账号超期时间

字段	数据类型	说明
passwd_expire	string	密码超期时间
last_logon	string	最后登录时间
user	string	用户
status	string	用户状态： • 0表示禁用 • 1表示正常
time	datetime	数据获取时间

进程快照

进程快照查询支持以下字段：

表 2-16: 进程快照支持字段

字段	数据类型	说明
uuid	string	客户端编号
IP	string	IP地址
path	string	进程路径
start_time	string	进程启动时间
uid	string	用户ID
cmdline	string	命令行
pname	string	父进程名
name	string	进程名
pid	string	进程ID
user	string	用户名
md5	string	进程文件MD5值，超过1MB不计算
time	datetime	数据获取时间

网络连接日志

网络连接日志查询支持以下字段：

表 2-17: 网络连接日志支持字段

字段	数据类型	说明
uuid	string	客户端编号
IP	string	IP地址
src_ip	string	源IP
src_port	string	源端口
proc_path	string	进程路径
dst_port	string	目标端口
proc_name	string	进程名
dst_ip	string	目标IP
status	string	状态
proto	string	协议
time	datetime	连接时间

2.4.6.4 语法逻辑说明

日志检索支持多条件逻辑检索，您可以在一个搜索条件（一个日志源）下增加多个判断逻辑，也可以对多个搜索条件（不同的日志源）进行逻辑组合。本文介绍了日志检索支持的语法逻辑，也列举了部分用例，帮助您理解和使用。

日志检索支持下表中列举的语法逻辑。

表 2-18: 语法逻辑说明

逻辑名称	描述
and	双目运算符。 形式为`query1 and query2`，搜索结果展示`query1`和`query2`查询结果的交集。 **说明**： 如果多个单词间没有语法关键词，默认是and的关系。
or	双目运算符。 形式为`query1 or query2`，搜索结果展示`query1`和`query2`查询结果的并集。

逻辑名称	描述
not	双目运算符。 形式为query1 not query2，搜索结果展示符合query1并且不符合query2的结果，相当于query1 - query2。 **说明**： 如果只有not query1，那么表示从全部日志中选取不包含query1的结果。

2.4.7 设置

2.4.7.1 管理安全配置

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **设置**。
3. 配置对服务器进行周期性木马查杀。
   a) 单击**管理**。
   b) 选择哪些服务器需要进行周期性的木马查杀。
   c) 单击**确认**，完成配置。
4. 配置安骑士Agent资源占用模式。
   - **业务优先模式**：CPU占用峰值小于10%，内存占用峰值小于50 MB。
   - **防护优先模式**：CPU占用峰值小于20%，内存占用峰值小于80 MB。

   a) 单击**管理**。
   b) 设置服务器的安骑士Agent工作模式。
   c) 单击**确认**，完成配置。

2.4.7.2 安装安骑士Agent插件

在Windows服务器或Linux服务器上手动安装安骑士Agent插件。

前提条件

如果您已在服务器上安装了安全软件（如安全狗、云锁等），可能会导致安骑士Agent插件无法正常安装，建议您暂时关闭或卸载该安全软件，然后再安装安骑士Agent插件。

背景信息

服务器必须通过安装程序（Windows）或脚本命令（Linux）方式安装安骑士Agent 插件。

如果您通过以下方式安装安骑士Agent 插件，需要删除安骑士Agent插件目录后，按照手动安装步骤重新安装安骑士Agent 插件。

- 通过已安装安骑士Agent插件的镜像批量安装服务器。
- 从已安装安骑士Agent插件的服务器上直接复制安骑士Agent插件文件。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全 > 设置 > 安装 / 卸载**。

   进入安骑士Agent插件安装页面，如图 *2-17: Agent*安装所示。

   图 2-17: Agent安装

3. 根据您的服务器操作系统，获取并安装安骑士Agent插件。

   - **Windows系统**

     1. 在左侧区域，单击**点击下载**，下载安装文件到本地计算机。
     2. 将安装文件上传至您的Windows服务器。例如，您可以通过FTP工具，将安装文件上传到服务器。
     3. 在Windows服务器上，以管理员权限运行安装文件，完成安装。

        说明：

安装Agent插件的过程中，您会收到提示，要求您输入安装验证Key。您可在安骑士Agent插件安装页面找到您的安装验证Key。

- **Linux系统**

  1. 在右侧区域，选择**非阿里云服务器**。
  2. 根据您的操作系统类型，选择32位或64位的安装命令，单击**复制**。
  3. 以管理员身份登录您的Linux服务器。
  4. 在Linux服务器上执行安装命令，下载和安装安骑士Agent插件。

4. 查看服务器在线情况。

   安骑士Agent 插件安装完成约五分钟后，可以在**云主机安全** > **主机列表**中查看您服务器的在线情况。

2.4.7.3 卸载安骑士Agent插件

如果主机不再使用安骑士服务的所有功能，可以选择以下方式进行卸载安骑士Agent插件。

背景信息

通过控制台卸载指定主机安骑士Agent，请务必确保当前机器安骑士Agent处于在线状态，否则无法接收到卸载指令。

如果卸载后重新安装安骑士Agent，请手工进行安装，忽略期间的报错，重复操作3次以上（安骑士Agent卸载会有一段保护期24小时或重复执行3次以上安装命令）。

操作步骤

1. 登录云盾安全中心。
2. 定位到**云主机安全** > **设置** > **安装 / 卸载**。
3. 单击右上角的**卸载安骑士**。
4. 在**卸载提示**对话框中，选择要卸载安骑士Agent插件的服务器。
5. 单击**确认卸载**，系统将自动卸载安骑士Agent插件。

2.5 安全审计

安全审计是指由专业审计人员根据有关法律法规、财产所有者的委托和管理当局的授权，对计算机网络环境下的有关活动或行为进行系统的、独立的检查验证，并作出相应评价。在管理员需要对系统过往的操作做回溯时，可以进行安全审计。

安全审计是一项长期的安全管理活动，贯穿云服务使用的生命周期。云盾的安全审计能够收集系统安全相关的数据，分析系统运行情况中的薄弱环节，上报审计事件，并将审计事件分为高、中、低三种风险等级，安全管理员关注和分析审计事件，从而持续改进系统，保证云服务的安全可靠。

2.5.1 查看审计一览

审计一览页面提供原始日志趋势、审计事件趋势、审计风险分布、危险事件分布四种报表。报表以趋势图或饼图的方式直观地呈现给安全管理员，便于分析云服务面临的风险趋势。

背景信息

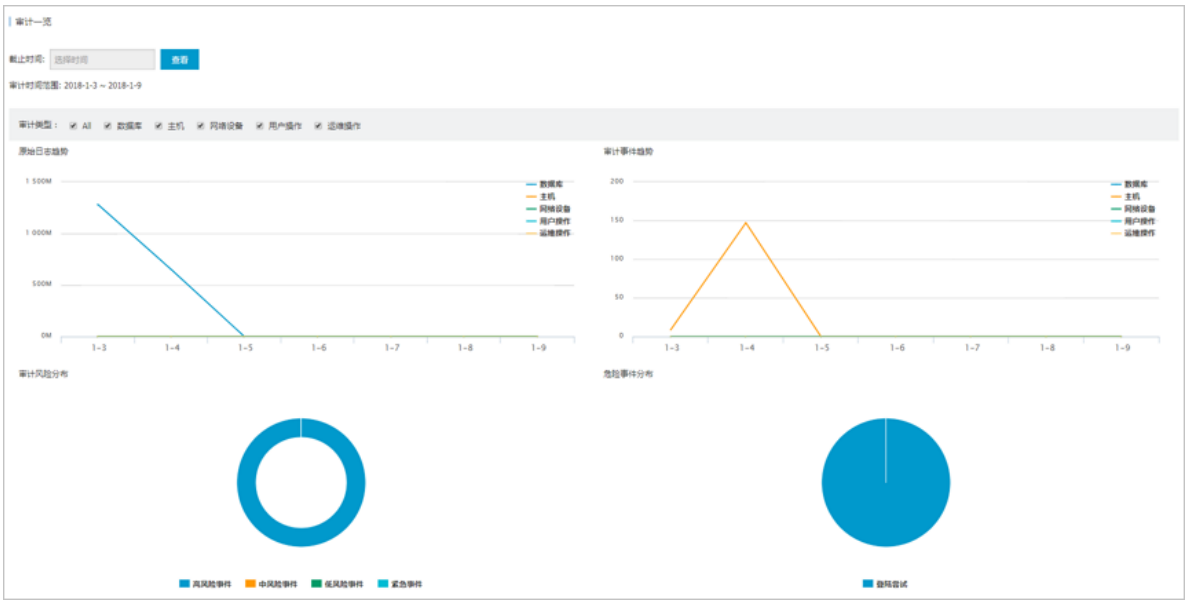
- **原始日志趋势**的数据是物理服务器、网络设备、RDS、ECS、OpenAPI一周内产生的日志个数。通过云平台日志趋势，安全管理员可以了解系统产生的日志数量是否正常。
- **审计事件趋势**的数据是物理服务器、网络设备、RDS、ECS、OpenAPI一周内产生的审计事件个数。通过审计事件趋势，安全管理员可以了解系统产生的审计事件数量是否正常。
- **审计风险分布**的数据是一周内高风险、中风险、低风险事件的个数。通过审计风险分布，安全管理员可以了解系统产生的审计事件级别是否正常。
- **危险事件分布**的数据是一周内不同事件类型占总事件的比例。通过危险事件分布，安全管理员可以了解什么类型的审计事件占比较多，识别高风险问题，做好预防措施。

同时，在**审计一览**页面，安全管理员还可以了解指定时间范围内所有审计类型的日志量信息及存储用量情况。

操作步骤

1. 登录云盾安全中心。
2. 定位到**安全审计** > **审计一览**，进入**审计一览**页面，如图 *2-18:* 审计一览页面所示。

图 2-18: 审计一览页面



3. 选择**截止时间**，单击**查看**，即可查看截止至该时间一周内的审计一览信息。

> **说明**：
> 在**审计时间范围**可以查看当前显示的审计日志信息的具体时间范围。

4. 勾选**审计类型**，可以选择是否显示该类型的审计日志信息。

2.5.2 查询审计事件

审计查询页面可查看日志创建时间、审计类型、审计对象、操作类型、风险级别、日志内容等审计事件的详细信息。

背景信息

审计事件生成的过程：将安全审计模块收集到的日志匹配审计规则，如果日志内容能匹配任意一条审计规则的正则表达式，就会上报审计事件。关于审计策略规则，参见管理审计策略。

操作步骤

1. 登录云盾安全中心。
2. 定位到**安全审计** > **审计查询**，进入**审计查询**页面。
3. 选择**审计类型**、**审计对象**、**操作类型**、**操作风险级别**等查询条件，设置查询时间，单击**查询**，查看该时间段内发现的审计事件。

说明：

单击**高级查询**，可设置更详细的审计事件过滤条件。

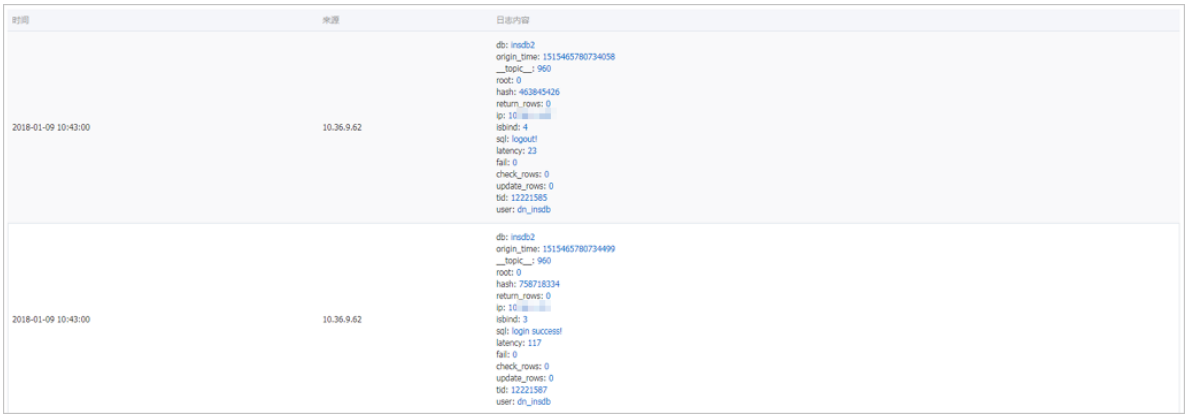
2.5.3 查看原始日志

在**原始日志**页面中，可查看审计对象在运行时产生的原始日志。原始日志作为必要的调试信息，安全管理员可以根据这些日志信息定位系统出现的故障。

操作步骤

1. 登录云盾安全中心。
2. 定位到**安全审计** > **原始日志**，进入**原始日志**页面。
3. 选择**审计类型**、**审计对象**，设置查询时间，单击**查询**，查看该时间段内指定审计对象的原始日志信息，如图 *2-19: 原始日志信息*所示。

图 2-19: 原始日志信息



2.5.4 策略设置

2.5.4.1 管理审计策略

审计策略支持正则表达式规则，当日志记录中的某个字符串匹配审计规则的正则表达式，就会上报审计事件。

背景信息

正则表达式描述了一种字符串匹配的模式，可以用来检查一个串是否含有某种子串。例如：

正则表达式	说明
`^\d{5,12}$`	表示匹配第5到第12位的连续数字

正则表达式	说明
load_file\(	表示匹配“load_file(”字符串

安全审计模块根据发生审计事件时日志中输出的字符串，定义了默认的审计策略。安全管理员也可以根据受到攻击时日志输出的字符串自定义审计策略。

操作步骤

1. 登录云盾安全中心。
2. 定位到**安全审计** > **策略设置**，选择**审计策略**页签，如图 *2-20: 审计策略页面*所示。

   图 2-20: 审计策略页面

3. 选择**审计类型**和**审计对象**，单击**查询**，查看当前已设置的审计策略。

   说明：

   在**审计对象**中选择全局，即显示对该审计类型的所有审计对象均适用的审计策略。

4. 管理审计策略。

   - 单击**新增**，在**新增规则**对话框中输入相关信息并单击**添加**，可添加审计策略，如图 *2-21: 新增规则对话框*所示。

图 2-21: 新增规则对话框

说明：

添加审计策略后，在指定的审计类型、审计对象、风险级别的审计日志中，如果出现匹配正则表达式的内容，会发送一封告警邮件给已设置的报警接收人。

- 单击**删除**，可删除该审计策略。

说明：

系统默认的审计策略无法删除。

- 单击**启用**或**禁用**，可设置该审计策略是否生效。

说明：

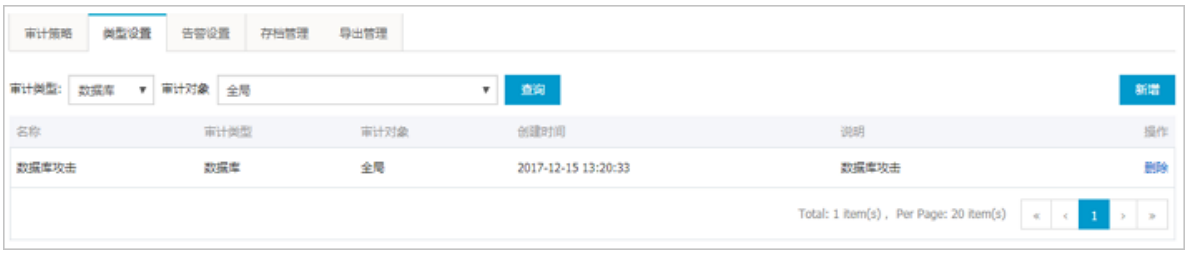
新增的审计策略默认为启用状态。

2.5.4.2 管理操作类型

操作步骤

1. 登录云盾安全中心。
2. 定位到**安全审计** > **策略设置**，选择**类型设置**页签，如图 2-22: 操作类型设置页面所示。

图 2-22: 操作类型设置页面



3. 选择**审计类型**、**审计对象**，单击**查询**，查看当前已设置的操作类型。

说明：

在**审计对象**中选择**全局**，即显示对该审计类型的所有审计对象均适用的操作类型。

4. 管理操作类型。
   - 单击**新增**，在**新增事件类型**对话框中输入相关信息即可添加操作类型，如图 2-23: 新增事件类型所示。

图 2-23: 新增事件类型

- 单击**删除**，可删除该操作类型。

> **说明**：
> 系统默认的操作类型无法删除。

2.5.4.3 设置告警接收人

设置报警接收人的邮箱，在发生审计事件后，会将事件上报到所设置的告警人的邮箱。

操作步骤

1. 登录云盾安全中心。
2. 定位到**安全审计** > **策略设置**，选择 **告警设置**页签，如图 *2-24:* 告警设置页面所示。

图 2-24: 告警设置页面

3. 选择**审计类型**、**审计对象**、风险等级，单击**查询**，查看当前已设置的告警接收人。

4. 设置告警接收人。

   - 单击**新增**，在**新增报警接收人**对话框输入相关信息即可添加告警接收人，如图 *2-25:* 新增报警接收人对话框所示。

     图 2-25: 新增报警接收人对话框

   - 单击**删除**，可删除该告警接收人。

2.5.4.4 管理事件日志存档

操作步骤

1. 登录云盾安全中心。
2. 定位到**安全审计** > **策略设置**，选择**存档管理**页签。
3. 选择**审计类型**、**归档类型**，设置**发现时间**，单击**查询**，查看相应的归档信息。

2.5.4.5 修改安全审计系统配置

通过设置安全审计的系统参数，可以配置系统单日最大报警次数及各类型原始日志的单日最大审计量。

操作步骤

1. 登录云盾安全中心。

2. 定位到**安全审计** > **策略设置**，选择**系统设置**页签。
3. 定位到想要修改的系统参数，单击**编辑**如图 *2-26: 系统设置*所示。

图 2-26: 系统设置

序号	说明	更新时间	值	操作
1	每天发送报警的最大次数	2018-07-02 09:14:12	1000	编辑
2	每天审计原始日志量(总量：GB/天)	2018-07-02 09:14:12	500	编辑
3	每天审计原始日志量(主机：GB/天)	2018-07-02 09:14:12	-1	编辑
4	每天审计原始日志量(网络设备：GB/天)	2018-07-02 09:14:12	-1	编辑
5	每天审计原始日志量(用户操作：GB/天)	2018-07-02 09:14:12	-1	编辑
6	每天审计原始日志量(运维操作：GB/天)	2018-07-02 09:14:12	-1	编辑

4. 填写对应的参数值，单击**确认**。

3 MaxCompute

3.1 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。

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。

3.2 什么是大数据计算服务

3.2.1 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。

3.2.2 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。
  - 访问并处理非结构化数据：MaxCompute团队依托MaxCompute系统架构，引入非结构化数据处理框架，解决MaxComputeSQL面对MaxCompute表外的各种用户数据时（例如：OSS上的非结构化数据或者来自TableStore的非结构化数据），需要首先通过各种工具导入MaxCompute表，才能在其上面进行计算，无法直接处理的问题。详情请参见：非结构化数据处理使用指南。
- Spark on MaxCompute：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。详情请参见：*Spark*使用指南。
- SDK：提供给开发者的工具包，SDK的相关介绍请参见：*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，详情请参见：*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，请参见：快速开始。

3.3 快速开始

3.3.1 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 Key ID**和**Access Key Secret**。

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见：*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语句的介绍请参见：*SQL 使用指南*。

3.3.2 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见：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见：项目空间的用户管理。

3.3.3 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。

3.3.3.1 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，详细介绍请参见：安全指南中的*ACL* 授权操作。

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
```

3.3.3.2 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见：*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。

3.3.3.3 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见：权限查看。

3.3.4 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见：角色管理。

3.3.5 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。

3.3.5.1 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_comm
ent]
[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见：创建表。修改分区请参见：添加及删除分区。生命周期的修改请参见：修改表的生命周期属性。

3.3.5.2 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。
```

3.3.5.3 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。

3.3.6 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。示例程序请参见：*Tunnel SDK示例*。

3.3.6.1 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。

3.3.6.1.1 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语句详情请参见：*Select*语句。

3.3.6.1.2 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='
2017', 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='2017
', 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语句详情请参见：*Insert*语句。

3.3.6.1.3 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见：*Join*语句。

3.3.6.1.4 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。

3.3.7 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。

3.3.7.1 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;
```

3.3.7.2 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 UDFExcepti
on { 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;
```

```
}
}
```

3.3.7.3 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);
}
}
}
```

3.3.8 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，具体操作请参见：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：

   创建表的语句请参见：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。

3.3.9 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：
   创建表的语句请参见：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见：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 = 20170730160742915gl205u3 2017-07-31 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_input_[bsp.sssp_in]_records=5 graph_output_[bsp.sssp_out]
   _bytes=161 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
   ```

3.4 基本介绍

3.4.1 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见：*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见：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：

表 3-1: 基本数据类型

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

类型	常量定义	描述	取值范围
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',"alibaba" ,'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：

表 3-2: 复杂数据类型

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，详情请参见：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见：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的操作介绍请参见：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Cancelled（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Cancelled
```

- **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。

3.4.2 常用命令

3.4.2.1 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、配置客户端请参见：配置客户端。

- 对于SDK的更多介绍请参见：*SDK介绍*。

3.4.2.2 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。

3.4.2.3 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见：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: 2017-01-01 17:32:13 |
  |LastDDLTime: 2017-01-01 17:57:38 |
  |LastModifiedTime: 2017-01-01 18:00:00 |
  +------------------------------------------------------------------------------------
  +
  |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见：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。

3.4.2.4 实例操作

- **Show Instances**

  命令格式如下：

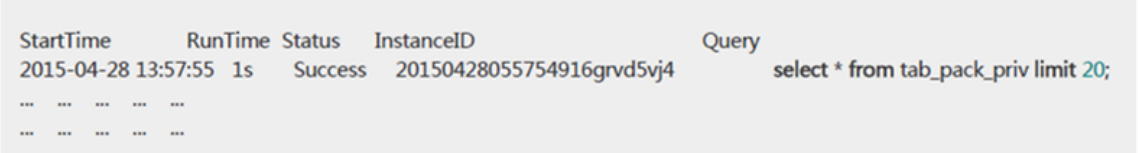
  ```
  SHOW INSTANCES [FROM startdate TO enddate] [number]
  ```

  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 3-1: 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 20171225123302267gk3u6k4y2; Success
  -- 查看ID为20171225123302267gk3u6k4y2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 20171225123302267gk3u6k4y2;
  ```

```
-- 停止ID为20171225123302267gk3u6k4y2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 20170715103441522gond1qa2;
  ID 20170715103441522gond1qa2
  Owner ALIYUN$maojing.mj@alibaba-inc.com
  StartTime 2017-07-15 18:34:41
  EndTime 2017-07-15 18:34:42
  Status Terminated
  console_select_query_task_1436956481295 Success Query select * from
  mj_test;
  -- 查询ID为20170715103441522gond1qa2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。

3.4.2.5 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见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='20170602
') as sale.res comment 'sale detail on 201706 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
```

3.4.2.6 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;
```

3.4.2.7 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='201705', 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
=201705,region=hangzhou -s false Upload session:  2017061016
39224880870a002ec60c
Start upload:d:\data.txt
Total bytes:41 Split input to 1 blocks
2017-06-10 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
='201705'; ID = 20170610084135370gyvc61z5
+-----------+-------------+-------------+-----------+--------+
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 20170610163922488 0870a002ec60c --
force;
start resume
20170610163922488 0870a002ec60c
Upload session: 20170610163922488 0870a002ec60c
Start upload:d:\data.txt
Resume 1 blocks
2017-06-10 16:46:42 upload block: '1'
2017-06-10 16:46:42 upload block complete, blockid=1
upload complete, average > speed is 0 KB/s
OK
```

其中，20170610163922488 0870a002ec60c为上传失败的session id。数据验证如下：

```
odps@ project_name>select * from sale_detail where sale_date='201705
';
ID = 20170610084801405g0a741z5
+-----------+-------------+-------------+-----------+--------+
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=201705,region=hangzhou
result.txt; Download session: 20170610165824528387 0a002ed0b9
Total records: 2
2017-06-10 16:58:24 download records: 2
2017-06-10 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
```

3.4.2.8 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='20171208') AS res_20171208; ADD
  TABLE src_part PARTITION (ds='20171209') AS res_20171209;
  ALIAS resName=res_20171208;
  jar -resources resName -libjars work.jar -classpath ./work.jar com.
  company.MainClass args ...;
  -- 作业一
  ALIAS resName=res_20171209;
  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支持的系统变量如下所示。

MaxCompute SQL及新版本Mapreduce支持的Set命令如下：

```
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命令如下：

```
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：

表 3-3: Project属性说明

属性名称	设置权限	属性描述	取值范围
odps.table.drop.ignorenonexistent	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.autoadjust	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.ppd	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，返回NULL(false)，或者报错(true)	true/false
odps.task.sql.write.str2null	ProjectOwner	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。

3.5 SQL使用指南

3.5.1 概要

3.5.1.1 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。

3.5.1.2 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
```

3.5.1.3 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，详情请参见： *DDL* 语句及 *DML* 语句部分的说明。

建表语句请参见：创建表。

3.5.1.4 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。

3.5.1.4.1 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：

表 3-4: 显式类型转换

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`**，详细说明信息请参见： *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，详细介绍请参见： *TO_CHAR* ，而to_date函数同样支持从string类型到datetime类型的转换，详细介绍请参见： *TO_DATE* 。
- 关于cast的介绍请参见： *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。

3.5.1.4.2 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：

表 3-5: 隐式类型转换

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	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：等于（=）、不等于（<>）、小于（<）、小于等于（<=）、大于（>）、大于等于（>=）、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。

表 3-6: 关系运算符作用下的隐式转换

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，请参见：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：加（+）、减（-）、乘（*）、除（/）、百分比（%），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见：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见：逻辑操作符。

3.5.1.4.3 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，详细信息请参见：内建函数部分的说明。

3.5.1.4.4 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。

3.5.1.4.5 分区列

MaxCompute SQL支持分区表，对于分区表的定义，请参见： *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。

3.5.1.4.6 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。

3.5.1.4.7 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`**。

表 3-7: 各个单位有效值域

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：2017-1-9 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2017-01-09 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(“2017-12-31 02:34:34” as datetime) 将会把string类型“2017-12-31 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("2017/12/31 02/34/34" as datetime)
cast("20171231023434" as datetime)
```

```
cast("2017-12-31 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。详细信息请参见：*TO_DATE* 。

3.5.2 运算符

3.5.2.1 关系操作符

表 3-8: 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

操作符	说明
	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，详细介绍可参见： *ABS* 。
- 通常情况下，MaxCompute的double类型能够保障16位有效数字。

3.5.2.2 算术操作符

表 3-9: 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。

3.5.2.3 位运算操作符

表 3-10: 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。

3.5.2.4 逻辑操作符

表 3-11: 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

操作符	说明
	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。

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。

3.5.3 DDL语句

3.5.3.1 表操作

3.5.3.1.1 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_comm
ent]
[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,
```

```
'2017' 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, '2017',
'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。

3.5.3.1.2 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_detai
l_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。
```

3.5.3.1.3 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;
```

3.5.3.1.4 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，具体请参见：获取表信息。

3.5.3.1.5 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。
```

3.5.3.1.6 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;
```

3.5.3.1.7 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，请参见：修改分区的修改时间。

3.5.3.1.8 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。

3.5.3.1.9 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='20170101') 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='20170101');
+------------------------------------------------------------------------------------
+
PartitionSize: 512 |
+------------------------------------------------------------------------------------
+
CreateTime: 2017-01-28 07:05:20 |
LastDDLTime: 2017-01-28 07:05:20 |
LastModifiedTime: 2017-01-28 07:05:21 |
```

```
+------------------------------------------------------------------------------
+
```

3.5.3.1.10 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='20170618') purge;
```

3.5.3.2 视图操作

3.5.3.2.1 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;
```

3.5.3.2.2 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;
```

3.5.3.2.3 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;
```

3.5.3.3 列及分区操作

3.5.3.3.1 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='201712
', region='hangzhou');
-- 成功添加分区，用来存储2017年12月杭州地区的销售记录。
alter table sale_detail add if not exists partition (sale_date='201712
', region='shanghai');
-- 成功添加分区，用来存储2017年12月上海地区的销售记录。
alter table sale_detail add if not exists partition(sale_date='
20171011');
-- 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。
```

3.5.3.3.2 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='201712',region='
hangzhou');
-- 成功删除2017年12月杭州分区的销售。
```

3.5.3.3.3 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、binary、timestamp、array、map以及struct类型
- 目前MaxCompute单表的列数上限为1200。

3.5.3.3.4 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。

3.5.3.3.5 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。

3.5.3.3.6 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，请参见：修改表的修改时间。

3.5.3.3.7 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。

3.5.4 DML语句

3.5.4.1 Insert语句

3.5.4.1.1 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='2017', region
='china');
insert overwrite table sale_detail_insert partition (sale_date='2017
', 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='2017
', 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='
2017', 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。
```

3.5.4.1.2 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_sta
tement2]
```

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='2016',
region='china' ) select shop_name, customer_id, total_price
insert overwrite table sale_detail_multi partition (sale_date='2017',
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='2017',
region='china' ) select shop_name, customer_id, total_price
insert overwrite table sale_detail_multi partition (sale_date='2017',
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='2016',
region='china' )
select shop_name, customer_id, total_price
insert into table sale_detail_multi partition (sale_date='2017',
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。
```

3.5.4.1.3 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='
2017', 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='2017
', 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。
```

3.5.4.2 Select语句

3.5.4.2.1 SELECT操作

命令格式如下：

```
select [all | distinct] select_expr, select_expr, ... from table_refe
rence
[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，详细信息请参见： *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：

  表 3-12: 过滤条件

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 >= '2015' and sale_detail.sale_date <= '
  2017';
  ```

  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，请参见：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。

3.5.4.2.2 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。

3.5.4.3 Union语句

3.5.4.3.1 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。

3.5.4.4 Join语句

3.5.4.4.1 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。
  ```

3.5.4.4.2 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。

  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;
```

3.5.4.5 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。

3.5.5 内建函数

3.5.5.1 数学运算函数

3.5.5.1.1 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(12232083745629837659238745692374 8) = 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
```

3.5.5.1.2 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
```

3.5.5.1.3 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
```

3.5.5.1.4 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
```

3.5.5.1.5 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
```

3.5.5.1.6 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'
```

3.5.5.1.7 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
```

3.5.5.1.8 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。

3.5.5.1.9 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。

3.5.5.1.10 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。

3.5.5.1.11 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
```

3.5.5.1.12 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。

3.5.5.1.13 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。

3.5.5.1.14 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。

3.5.5.1.15 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;
```

3.5.5.1.16 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
```

3.5.5.1.17 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。

3.5.5.1.18 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。

3.5.5.1.19 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。

3.5.5.1.20 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。

3.5.5.1.21 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。

3.5.5.1.22 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
```

3.5.5.1.23 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。

3.5.5.1.24 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
```

3.5.5.1.25 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
```

3.5.5.1.26 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'
```

3.5.5.1.27 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'
```

3.5.5.1.28 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'
```

3.5.5.1.29 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
```

3.5.5.1.30 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
```

3.5.5.1.31 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
```

3.5.5.1.32 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
```

3.5.5.1.33 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
```

3.5.5.1.34 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
```

3.5.5.1.35 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
```

3.5.5.1.36 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）
```

3.5.5.1.37 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）
```

3.5.5.1.38 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)
```

3.5.5.2 字符串处理函数

3.5.5.2.1 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
```

3.5.5.2.2 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。

3.5.5.2.3 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
```

3.5.5.2.4 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
```

3.5.5.2.5 IS_ENCODING

函数声明如下：

```
boolean is_encoding(string str, strin from_encoding, string to_encodin
g)
```

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
```

3.5.5.2.6 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。
```

3.5.5.2.7 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
```

3.5.5.2.8 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
```

3.5.5.2.9 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。

3.5.5.2.10 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"
```

3.5.5.2.11 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
```

3.5.5.2.12 REGEXP_INSTR

函数声明如下：

```
bigint regexp_instr(string source, string pattern[,bigint start_posi
tion[, 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
```

3.5.5.2.13 REGEXP_SUBSTR

函数声明如下：

```
string regexp_substr(string source, string pattern[, bigint start_posi
tion[, 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"
```

3.5.5.2.14 REGEXP_COUNT

函数声明如下：

```
bigint regexp_count(string source, string pattern[, bigint start_posi
tion])
```

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
```

3.5.5.2.15 SPLIT_PART

函数声明如下：

```
string split_part(string str, string delimiter, bigint start[, bigint
end])
```

用途：依照分隔符delimiter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。
- delimiter：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delimiter为空串，返回原字符串str。

说明：

- 如果delimiter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) = ''
```

3.5.5.2.16 REGEXP_REPLACE

函数声明如下：

```
string regexp_replace(string source, string pattern, string replace_st
ring[, 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。
```

3.5.5.2.17 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"
```

3.5.5.2.18 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"
```

3.5.5.2.19 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"
```

3.5.5.2.20 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'
```

3.5.5.2.21 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。

3.5.5.2.22 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  |
+-----+
```

3.5.5.2.23 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 |
+-----+
```

3.5.5.2.24 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 |
+-----+
```

3.5.5.2.25 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。
```

3.5.5.2.26 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
```

3.5.5.2.27 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
```

3.5.5.2.28 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"
```

3.5.5.2.29 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_encode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)" ```
```

3.5.5.2.30 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。

3.5.5.2.31 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','bob') = 'name:bob'
concat_ws(':','avg',null,'34')= 'null'
```

3.5.5.2.32 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
```

3.5.5.2.33 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
```

3.5.5.2.34 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('ababab','abab','12') = '12ab'
replace('ababab','cdf','123') = 'ababab'
```

```
replace('123abab456ab',null,'abab') = 'null'
```

3.5.5.2.35 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'
```

3.5.5.2.36 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'
```

3.5.5.3 日期处理函数

3.5.5.3.1 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见：*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 = 2017-02-28 00:00:00：
dateadd(trans_date, 1, 'dd') = 2017-03-01 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') = 2017-02-27 00:00:00
-- 减一天
dateadd(trans_date, 20, 'mm') = 2018-10-28 00:00:00
-- 加20个月，月份溢出，年份加1
若trans_date = 2017-02-28 00:00:00, dateadd(transdate, 1, 'mm') = 2017
-03-28 00:00:00
若trans_date = 2017-01-29 00:00:00, dateadd(transdate, 1, 'mm') = 2017
-02-28 00:00:00
-- 2017年2月没有29日，日期截取至当月最后一天，
若trans_date = 2017-03-30 00:00:00, dateadd(transdate, -1, 'mm') =
2017-02-28 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(2017-03-30 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("2017-03-30 00:00:00" as datetime), -1, 'mm') from
 tbl1;
-- 将String类型常量显式转换为Datetime类型
```

3.5.5.3.2 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 = 2017-12-31 23:59:59，end = 2018-01-01 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('2017-05-31 13:00:00', '2017-05-31 12:30:00', 'ss') = 1800
datediff('2017-05-31 13:00:00', '2017-05-31 12:30:00', 'mi') = 30
```

3.5.5.3.3 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('2017-06-08 01:10:00', 'yyyy') = 2017
datepart('2017-06-08 01:10:00', 'mm') = 6
```

3.5.5.3.4 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('2017-12-07 16:28:46', 'yyyy') = 2017-01-01 00:00:00
datetrunc('2017-12-07 16:28:46', 'month') = 2017-12-01 00:00:00
datetrunc('2017-12-07 16:28:46', 'DD') = 2017-12-07 00:00:00
```

3.5.5.3.5 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;
```

3.5.5.3.6 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）。

3.5.5.3.7 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。

3.5.5.3.8 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。

3.5.5.3.9 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2017-12*03', '阿里巴巴yyyy-mm*dd') = 2017-12-03 00:00:00
to_date('20170718', 'yyyymmdd') = 2017-07-18 00:00:00
to_date('201707182030','yyyymmddhhmi')=2017-07-18 20:30:00
to_date('2017718', 'yyyymmdd')
-- 格式不符合，返回NULL
to_date('阿里巴巴2017-12*3', '阿里巴巴yyyy-mm*dd')
-- 格式不符合，返回NULL
to_date('2017-24-01', 'yyyy')
-- 格式不符合，返回NULL
```

3.5.5.3.10 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('2017-12-03 00:00:00', '阿里金融yyyy-mm*dd') = '阿里金融2017-12*
03'
to_char('2017-07-18 00:00:00', 'yyyymmdd') = '20170718'
to_char('阿里巴巴2017-12*3', '阿里巴巴yyyy-mm*dd')
-- 返回NULL
to_char('2017-24-01', 'yyyy')
-- 返回NULL
to_char('2017718', 'yyyymmdd')
```

```
-- 返回NULL
```

说明：

关于其他类型向string类型转换请参见：字符串函数*TO_CHAR* 。

3.5.5.3.11 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。

3.5.5.3.12 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。

3.5.5.3.13 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("20171229", "yyyymmdd")) from dual;
返回结果：
+------------+
| _c0        |
+------------+
| 1          |
+------------+
-- 虽然20171229属于2017年，但是这一周的大多数日期是在2018年，因此返回结果为1，表示是2018年的第一周。
select weekofyear(to_date("20171231", "yyyymmdd")) from dual;
-- 返回结果为1。
select weekofyear(to_date("20181229", "yyyymmdd")) from dual;
-- 返回结果为53。
```

3.5.5.3.14 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('2017-01-01 12:30:00') = 2017 from dual;
```

本章节后介绍的日期函数即为新扩展的日期函数。

3.5.5.3.15 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('2017-01-01 12:30:00') = 2017
year('2017-01-01') = 2017
year('17-01-01') = 17
year(2017-01-01) = null
year('2017/03/09') = null
year(null) = null
```

3.5.5.3.16 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('2017-11-12 10:00:00') = 4
quarter('2017-11-12') = 4
```

3.5.5.3.17 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('2017-09-01') = 9
month('20170901') = null
```

3.5.5.3.18 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('2017-09-01') = 1
day('20170901') = null
```

3.5.5.3.19 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('2017-09-01') = 1
```

```
dayofmonth('20170901') = null
```

3.5.5.3.20 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('2017-09-01 12:00:00') = 12
hour('12:00:00') = 12
hour('20170901120000') = null
```

3.5.5.3.21 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('2017-09-01 12:30:00') = 30
minute('12:30:00') = 30
minute('20170901120000') = null
```

3.5.5.3.22 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('2017-09-01 12:30:45') = 45
second('12:30:45') = 45
second('20170901123045') = null
```

3.5.5.3.23 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/bigint。
- 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'
```

3.5.5.3.24 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'
```

3.5.5.3.25 ADD_MONTHS

函数声明如下：

```
string add_months(string startdate, int nummonths)
```

用途：返回开始日期startdate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
```

3.5.5.3.26 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
```

3.5.5.3.27 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
```

3.5.5.3.28 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') = 3.9495967741
935485
```

```
months_between('1996-10-30','1997-02-28 10:30:00' ) = -3.9495967741
935485
months_between('1996-09-30','1996-12-31') = -3.0
```

3.5.5.4 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。

3.5.5.4.1 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。
```

3.5.5.4.2 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。

3.5.5.4.3 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。

3.5.5.4.4 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。

3.5.5.4.5 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。

3.5.5.4.6 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。

3.5.5.4.7 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。

3.5.5.4.8 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。

3.5.5.4.9 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          |
+------------+-------+------------+------------+
```

3.5.5.4.10 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          |
+------------+-------+------------+------------+
```

3.5.5.4.11 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。

3.5.5.4.12 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;
```

3.5.5.4.13 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。

3.5.5.4.14 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           |
+------------+-------+------------+------------+
```

3.5.5.4.15 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：Bigint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 |
+-----+----------------------
```

3.5.5.5 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。

3.5.5.5.1 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   |
+-----+-------+
select key, count(value) as count from test_src group by key;
-- 此时执行上述语句，结果为：
+-----+-------+
| 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。

3.5.5.5.2 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  |
+-------+
select avg(value) as avg from tbla;
+------+
| avg  |
+------+
| 1.5  |
+------+
-- 则对该列计算avg结果为(1+2)/2=1.5
```

3.5.5.5.3 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
```

3.5.5.5.4 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
```

3.5.5.5.5 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。

3.5.5.5.6 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。

3.5.5.5.7 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。

3.5.5.5.8 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
```

3.5.5.5.9 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。

3.5.5.5.10 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。

3.5.5.5.11 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。

3.5.5.5.12 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。

3.5.5.6 其他函数

3.5.5.6.1 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]
```

3.5.5.6.2 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
```

3.5.5.6.3 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**。

3.5.5.6.4 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。

3.5.5.6.5 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。

3.5.5.6.6 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
```

3.5.5.6.7 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。

3.5.5.6.8 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。

3.5.5.6.9 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。

3.5.5.6.10 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。

3.5.5.6.11 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。

3.5.5.6.12 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='20170901'，pt='20170902' ，则以下语句中max_pt返回值为'20170902'，MaxCompute SQL语句读出pt='20170902'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。

3.5.5.6.13 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
```

3.5.5.6.14 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。

3.5.5.6.15 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
```

3.5.5.6.16 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] |
+------+
```

3.5.5.6.17 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} |
```

3.5.5.6.18 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。

3.5.5.6.19 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” 。

3.5.5.6.20 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
```

3.5.5.6.21 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。

3.5.5.6.22 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        |
+------------+
```

3.5.5.6.23 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。

3.5.5.6.24 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}
```

3.5.5.6.25 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]
```

3.5.5.6.26 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]
```

3.5.5.6.27 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]
```

3.5.5.6.28 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   |
+------------+-----+
```

3.5.5.6.29 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}
```

3.5.5.6.30 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','bob','married','F','
weight',63.50) from dual;
-- 返回结果：
{user_id:10001, user_name:bob, married:F, weight:63.5}
```

3.5.5.6.31 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','bob','
married','F','weight',63.50))) from dual;
-- 返回结果：
+------------+-----------+---------+------------+
```

```
| user_id    | user_name | married | weight     |
+------------+-----------+---------+------------+
| 10001      | bob       | F       | 63.5       |
+------------+-----------+---------+------------+
```

3.5.5.6.32 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  |
+-------+-------+-----+-----------+-----------+-----------
+------------+------------+
```

3.5.6 UDF

3.5.6.1 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：

表 3-13: UDF分类

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。

3.5.6.1.1 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：

表 3-14: 对应关系

MaxCompute Type	Java Type
Tinyint	java.lang.Byte
Smallint	java.lang.Short

MaxCompute Type	Java Type
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
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：

表 3-15: 对比情况

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

3.5.6.2 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，详情请参见：内建函数

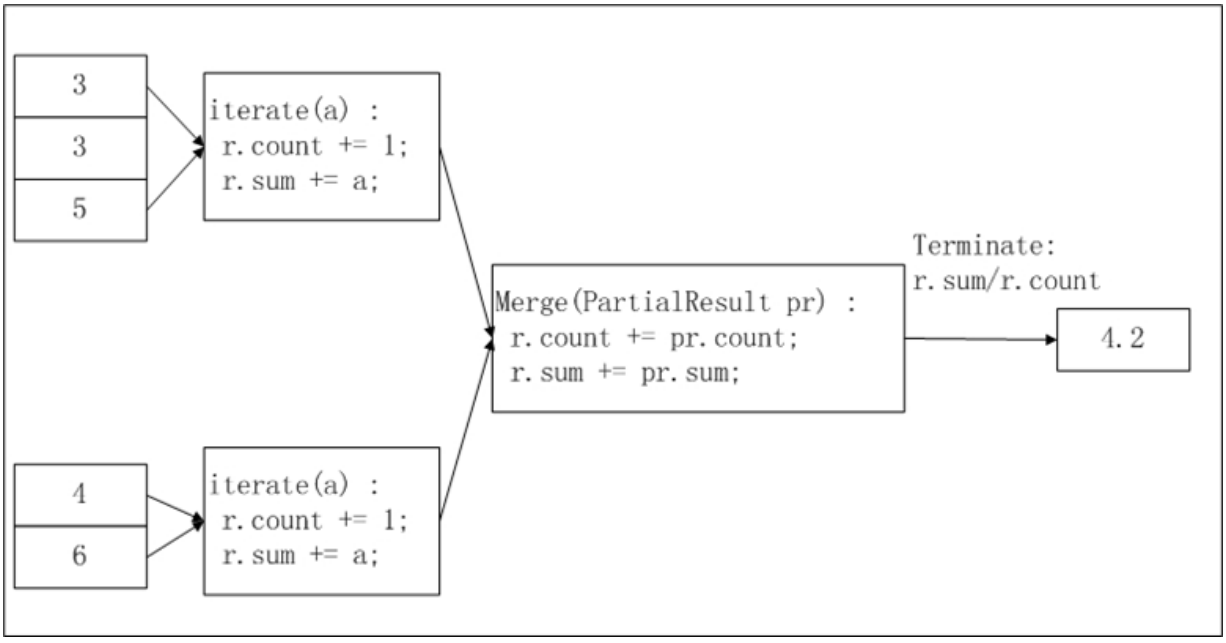
3.5.6.3 UDAF

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 UDFExcepti
on;
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见：*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 UDFExcepti
on { 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，详情请参见： 聚合函数。
- 关于如何运行UDTF的方法与UDF 类似，详情请参见： 运行*UDF*。

3.5.6.4 UDTF

3.5.6.4.1 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：

表 3-16: 接口定义

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。
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  |
+----+----+
```

3.5.6.4.2 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 +
 "|tableResource1RecordCount=" + 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.UDTFResour
ce 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|tableResource1Record
Count=0|tableResource2RecordCount=0 |
| 10    | 2          | fileResourceLineCount=3|tableResource1Record
Count=0|tableResource2RecordCount=0 |
+-------+------------+-------+
```

说明：

用户同样可以使用相同的方式获取资源，详情请参见：*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;
```

3.5.6.5 Python UDF

3.5.6.5.1 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。

3.5.6.5.2 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。

3.5.6.5.3 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 3-17: 对应关系

MaxCompute SQL Type	Python Type
Bigint	int
String	str

MaxCompute SQL Type	Python Type
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 。

3.5.6.5.4 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。

3.5.6.5.5 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]
```

3.5.6.5.6 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。

3.5.6.5.7 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)
```

3.5.7 附录

3.5.7.1 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。

表 3-18: 转义字符

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”。
```

3.5.7.2 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。

3.5.7.3 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 |
+---------+------+
```

3.5.7.4 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 CONCATENAT
E 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 MATERIALIZ
ED MINUS MSCK NOT NO_DROP NULL OF OFFLINE ON OPTION OR ORDER OUT
OUTER OUTPUTDRIVER OUTPUTFORMAT OVER OVERWRITE PARTITION PARTITIONE
D PARTITIONPROPERTIES PARTITIONS PERCENT PLUS PRECEDING PRESERVE
PROCEDURE PURGE RANGE RCFILE READ READONLY READS REBUILD RECORDREAD
ER 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
```

3.5.7.5 限制项汇总

以下为汇总的MaxCompute的限制项说明：

表 3-19: 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个	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。

边界名	最大值/限制条件	分类	说明
MAPJOIN	8个小表	数量限制	MAPJOIN最多允许8张小表。
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个	数量限制	in的最大参数限制，如 in (1,2,3....,1024)。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个	数量限制	最多查询分区个数不能超过10000个。

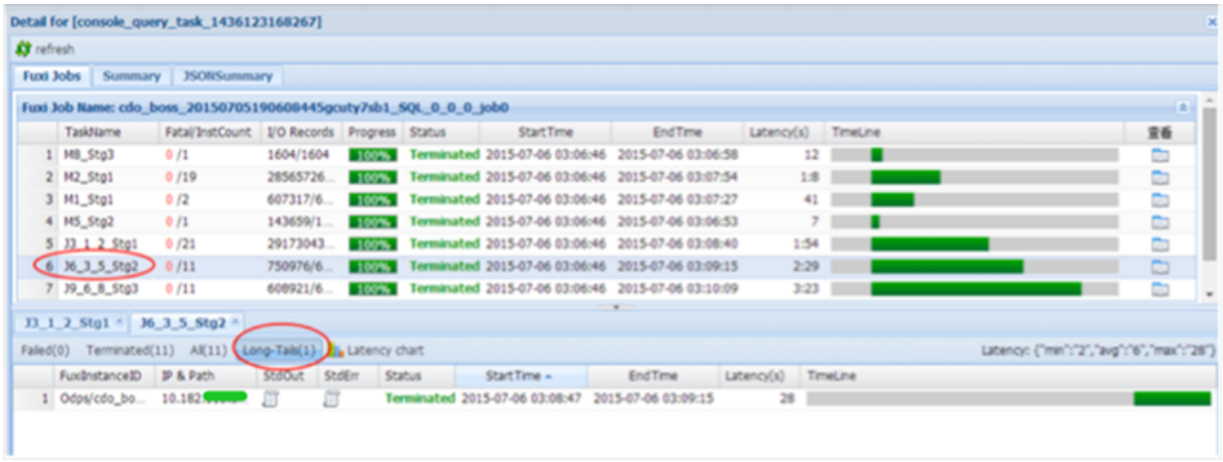
注意：

上述的MaxCompute SQL限制项均不可以被人为修改配置。

3.5.7.6 常见问题及使用建议

3.5.7.6.1 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。

3.5.7.6.1.1 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。

3.5.7.6.1.2 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。

3.5.7.6.1.3 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;
```

3.5.7.6.1.4 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。

3.5.7.6.1.5 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。

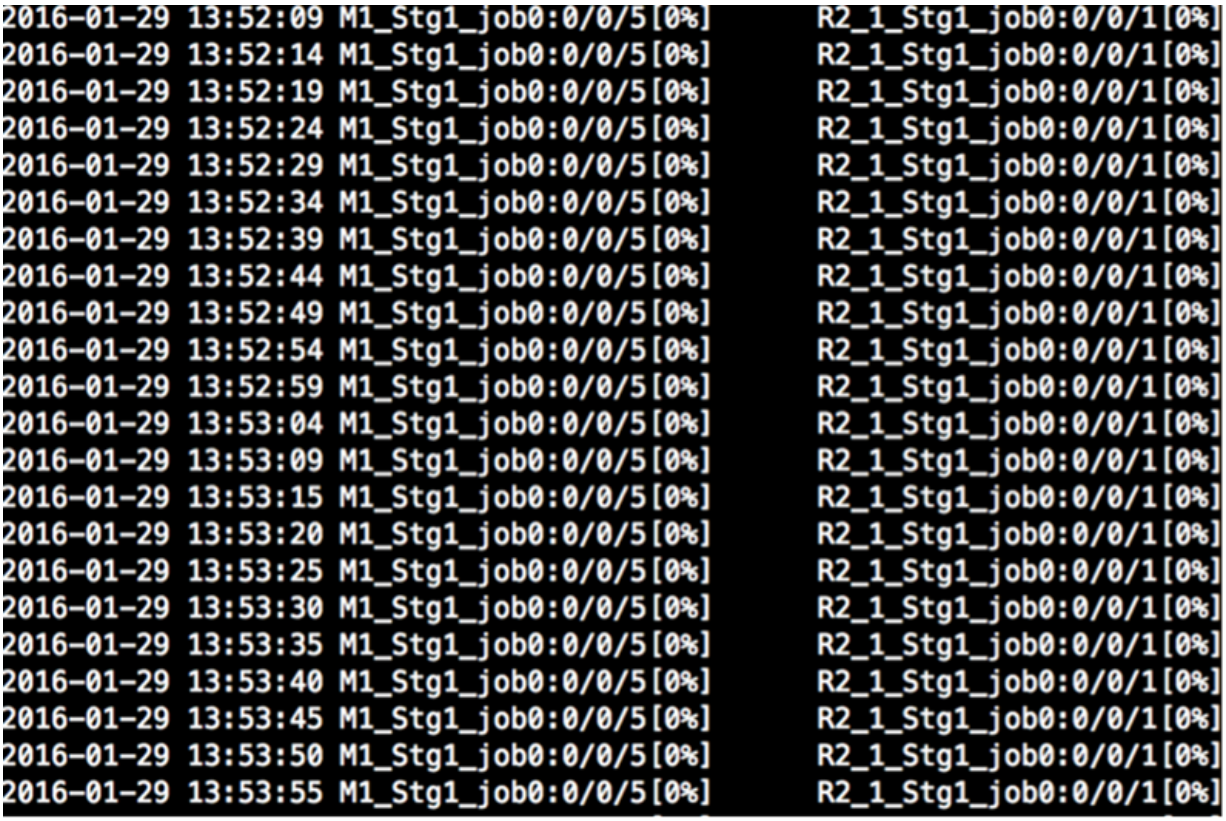
解决方法：如果目标分区数比较少，根本就不会造成小文件过多的困扰，可以通过如下命令关闭上述功能；又或者不使用动态分区。

```
set odps.sql.reshuffle.dynamicpt=false;
```

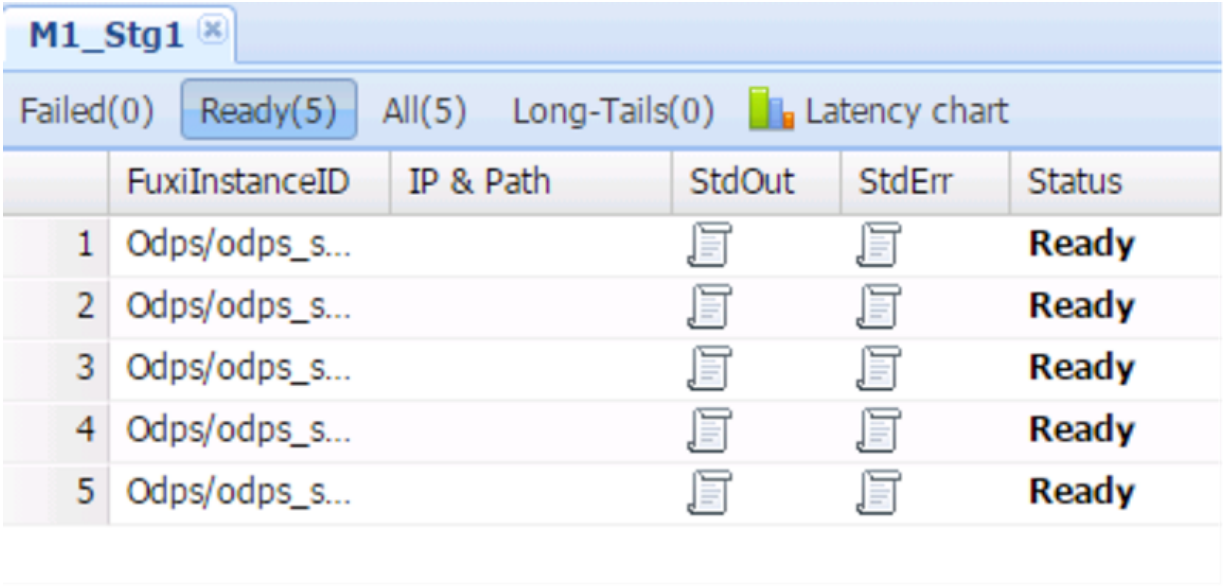
3.5.7.6.2 Quota及资源使用

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%）。



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）。

3.5.7.6.3 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‘20160901’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='20170101') 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。

3.5.7.6.4 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;
```

3.5.7.7 MaxCompute常用SQL参数设置

3.5.7.7.1 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。

3.5.7.7.2 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。

3.5.7.7.3 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。

3.5.7.7.4 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。

3.5.7.7.5 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。

3.5.7.7.6 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;
```

3.6 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。

3.6.1 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。

表 3-20: 主要接口

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。

3.6.1.1 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。

3.6.1.2 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。

3.6.1.3 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 TunnelExce
ption
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，一直到下载结束。
- 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。

3.6.2 Tunnel SDK示例

3.6.2.1 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();
    }
  }
```

```
}
```

3.6.2.2 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();
}
}
```

3.6.2.3 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();
    }
  }
}
```

3.6.2.4 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); DownloadSe
ssion 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();
}
}
}
```

3.6.3 附录

3.6.3.1 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。

3.6.3.2 Tunnel常见错误码

Tunnel常见的错误码如下所示：

表 3-21: 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

ErrorCode	错误原因	处理建议
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='2017'
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

3.7 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。

3.7.1 概要

3.7.1.1 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。

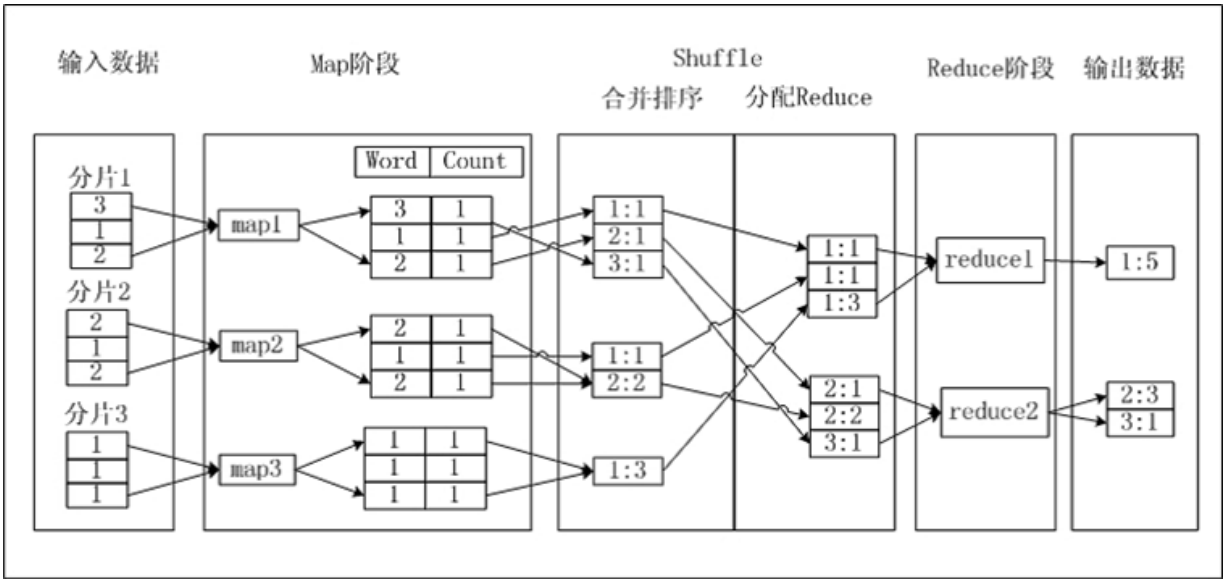
   说明：
   Combiner的逻辑可以由用户自定义实现。与经典的MapReduce框架协议不同，在MaxCompute中，Combiner的输入、输出参数必须与Reduce保持一致。这部分的处理通常也叫做**洗牌**（`Shuffle`）。

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。

3.7.1.2 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。

3.7.2 功能介绍

3.7.2.1 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
```

3.7.2.2 基本概念

3.7.2.2.1 Map/Reduce

Map及Reduce分别支持对应的map/reduce方法、setup方法及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：

详细使用示例请参见：示例程序。

3.7.2.2.2 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：

详细使用示例请参见：二次排序源代码。

3.7.2.2.3 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。

3.7.2.2.4 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。

3.7.2.2.5 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：
详细使用示例请参见： 示例程序。

3.7.2.2.6 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：
详细使用示例请参见：资源使用示例。

3.7.2.2.7 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示例的介绍请参见： *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：

表 3-22: 功能比对

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。

功能描述	Hadoop MapReduce	MaxCompute MapReduce
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。

3.7.3 SDK介绍

在本章节，会对较为常用的MapReduce核心接口做简短介绍。

3.7.3.1 核心接口

表 3-23: 主要接口

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。

3.7.3.1.1 MapperBase

主要函数接口如表所示：

表 3-24: 主要接口

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。

3.7.3.1.2 ReducerBase

主要函数接口如表所示：

表 3-25: 主要接口

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。

3.7.3.1.3 TaskContext

主要函数接口如表所示：

表 3-26: 主要接口

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。

主要接口	描述
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。

3.7.3.1.4 JobConf

主要函数接口如表所示：

表 3-27: 主要接口

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。

主要接口	描述
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。
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。

3.7.3.1.5 JobClient

主要函数接口如表所示：

表 3-28: 主要接口

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。

3.7.3.1.6 RunningJob

主要函数接口如表所示：

表 3-29: 主要接口

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。
void waitForCom pletion()	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 getJobStat us()	查询作业实例运行状态。
void killJob()	结束此作业。
Counters getCounter s()	获取Conter信息。

3.7.3.1.7 InputUtils

主要函数接口如表所示：

表 3-30: 主要接口

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。

3.7.3.1.8 OutputUtils

主要函数接口如表所示：

表 3-31: 主要接口

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。

3.7.3.1.9 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。

3.7.3.1.10 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：

表 3-32: 对应关系

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
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

3.7.4 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。

3.7.5 示例程序

3.7.5.1 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);
}
}
```

3.7.5.2 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);
}
}
```

3.7.5.3 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);
}
}
```

3.7.5.4 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.");
}
}
}
```

3.7.5.5 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);
}
```

```
}
```

3.7.5.6 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。

3.7.5.7 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);
}
}
```

3.7.5.8 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);
}
```

```
}
```

3.7.5.9 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_sr
c22 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);
}
}
```

3.7.5.10 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 reduceSlee
pCount) {
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, reduceSlee
pTime, reduceSleepCount);
}
}
```

3.7.5.11 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);
}
}
}
```

3.7.5.12 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.");
}
```

```
}
```

3.7.5.13 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。

3.7.5.14 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，PartitionC
olumns，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);
}
```

```
}
```

3.8 Graph使用指南

3.8.1 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。

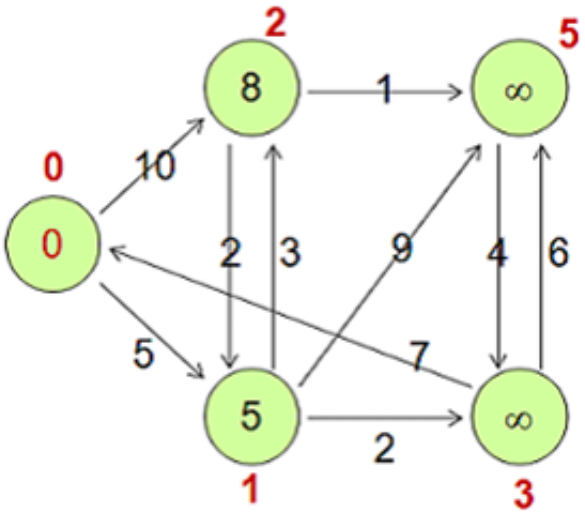
通过迭代对图进行编辑、演化，最终求解出结果，典型应用有*PageRank*、单源最短距离算法、*K*-均值聚类算法等。用户可以使用MaxCompute Graph提供的接口Java SDK编写图计算程序。

3.8.1.1 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 3-2: Graph数据结构



以上图为例，上图由下面的点组成：

表 3-33: Graph数据结构

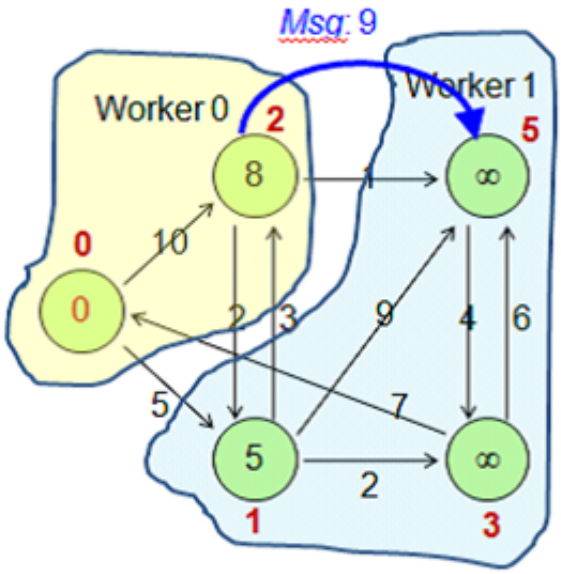
Vertex	<ID, Value, Halted, Edges>
v0	<0, 0, false, [<1, 5 >, <2, 10 >] >
v1	<1, 5, false, [<2, 3>, <3, 2>, <5, 9>]>
v2	<2, 8, false, [<1, 2>, <5, 1 >]>
v3	<3, Long.MAX_VALUE, false, [<0, 7>, <5, 6>]>
v5	<5, Long.MAX_VALUE, false, [<3, 4 >]>

3.8.1.2 Graph程序逻辑

3.8.1.2.1 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 3-3: 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。

3.8.1.2.2 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。

3.8.1.2.3 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();
```

3.8.2 Graph功能介绍

3.8.2.1 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");
```

```
}
```

3.8.2.2 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。
job.addInput(TableInfo.builder().tableName( “tblname” ).partSpec("pt1=a
/pt2=b").build(), new String[]{"col2", "col0 "});
//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
```

说明：

- 支持多路输入。
- 暂时不支持分区过滤条件。更多应用限制请参见：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();
job.addOutput(TableInfo.builder().tableName("table_name").partSpec("
pt1=a/pt2=b").build());
//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").lable("output1").build(), true);
//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
```

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。

3.8.2.3 读取资源

3.8.2.3.1 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)
```

3.8.2.3.2 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。
- 更多应用限制请参见：应用限制。

3.8.3 Graph SDK介绍

表 3-34: 主要接口

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。

主要接口	说明
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。
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。

3.8.4 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。

3.8.4.1 开发示例

背景信息

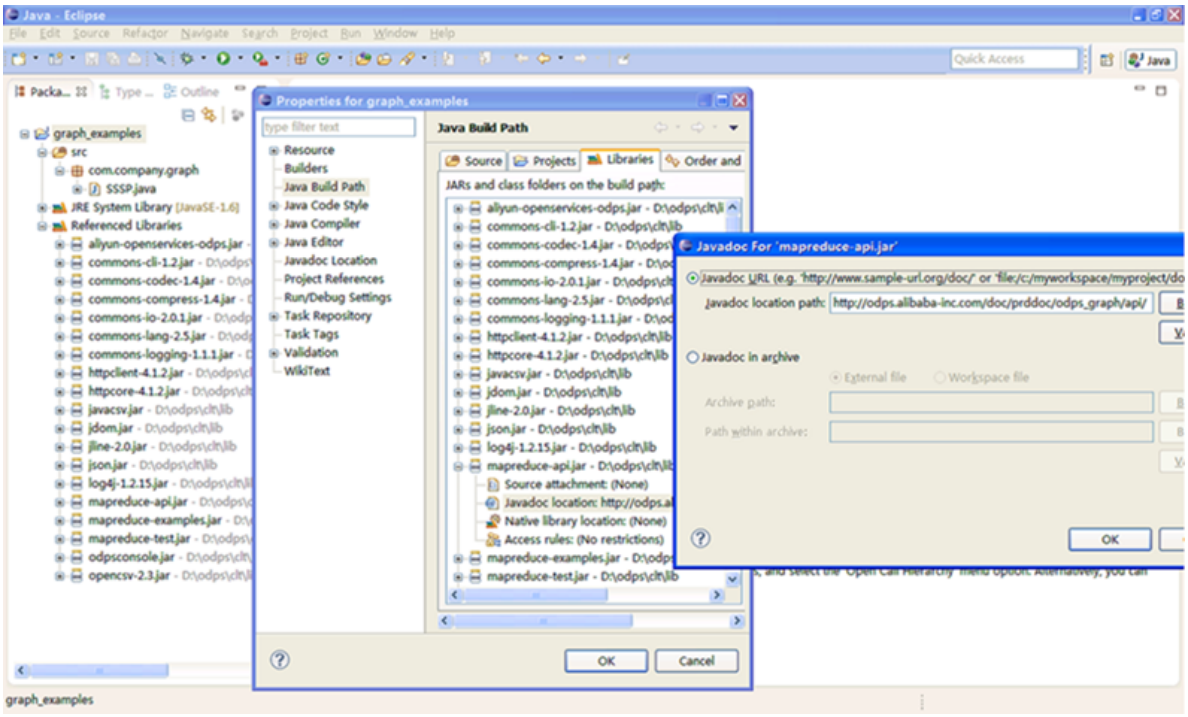
本章节以SSSP算法为例介绍如何使用Eclipse开发和调试Graph程序。具体的开发步骤如下。

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 3-4: 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，相关操作请参见：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。

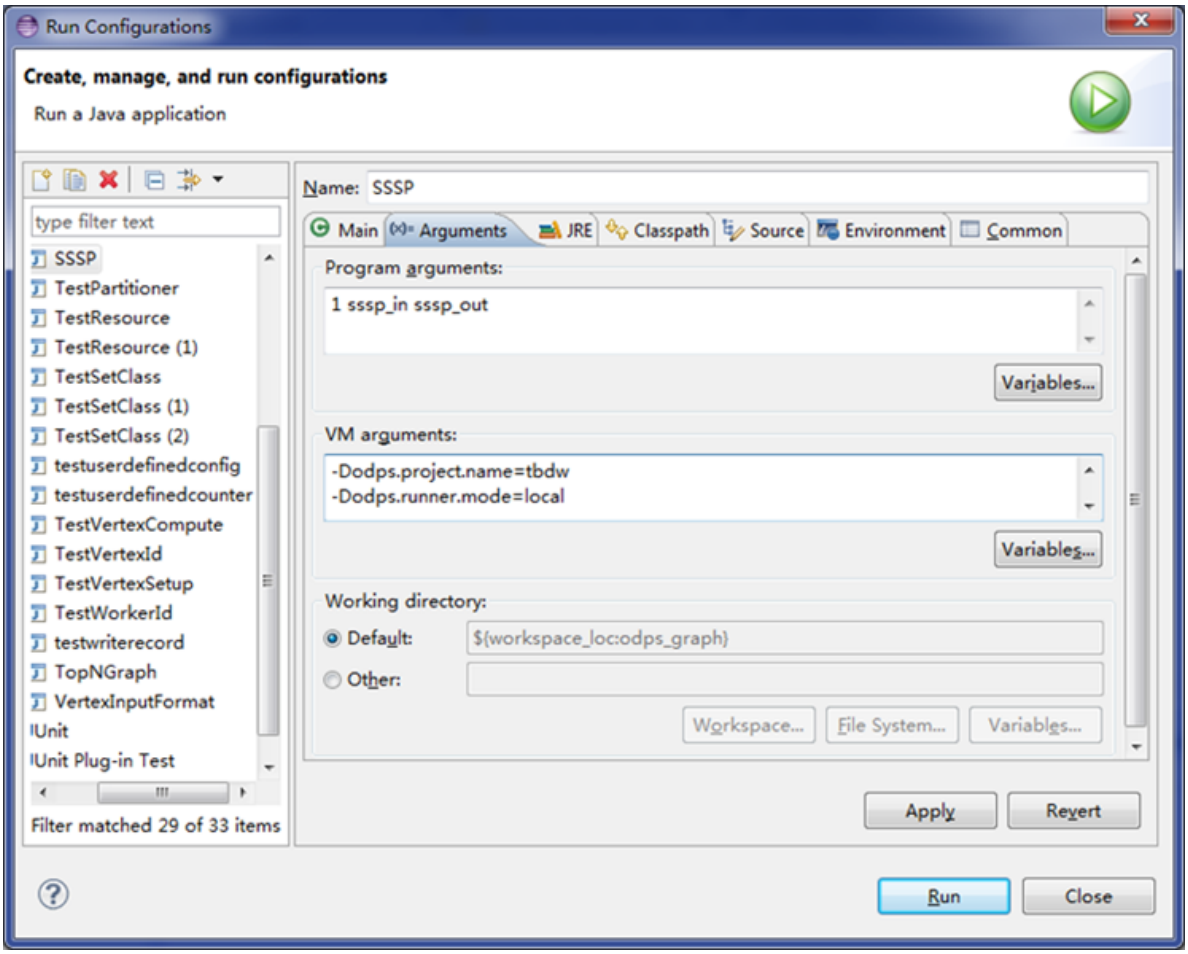
3.8.4.2 本地调试

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 3-5: 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的介绍请参见：*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：

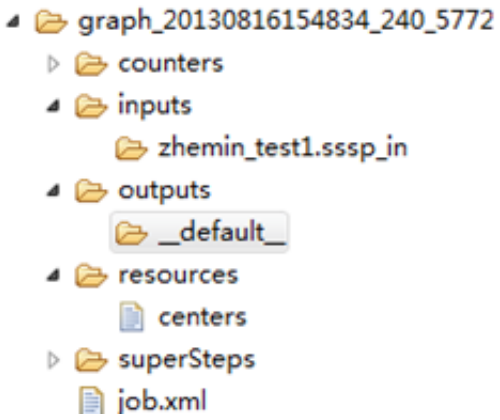
```
Counters: 3
com.aliyun.odps.graph.local.COUNTER
TASK_INPUT_BYTE=211
TASK_INPUT_RECORD=5
TASK_OUTPUT_BYTE=161
TASK_OUTPUT_RECORD=5
graph task finish
```

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表的详细信息请参见：编写并运行*Graph*。

3.8.4.3 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`**。

3.8.4.4 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的详细介绍请参见：编写并运行*Graph*。

3.8.4.5 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。

3.8.4.5.1 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。

3.8.4.5.2 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。详细介绍请参见：*Graph SDK介绍*中的Combiner部分。

3.8.4.5.3 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。

3.8.4.5.4 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。

3.8.5 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），具体限制请参见：*Java* 沙箱限制介绍。

3.8.6 示例程序

3.8.6.1 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 combinedMe
ssage, 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 LongWritab
le(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");
}
}
```

3.8.6.2 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");
}
}
```

3.8.6.3 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<KmeansAggr
Value> {
@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() - ((DoubleWrit
able) 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");
}
}
```

3.8.6.4 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();
}
}
```

3.8.6.5 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 transposeN
eighbors.
*
* 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_R
EST);
} 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");
}
}
```

3.8.6.6 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");
}
}
```

3.8.6.7 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])), NullWritab
le.get());
}
}
LOG.info(record.toString());
context.addVertexRequest(vertex);
}
}
public static class LongSumCombiner extends
Combiner<LongWritable, LongWritable> {
@Override
public void combine(LongWritable vertexId, LongWritable combinedMe
ssage, 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");
}
}
```

3.8.6.8 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.parseDoubl
e(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.
setRuntimePartitioning(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");
}
}
```

3.8.6.9 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");
}
}
```

3.8.6.10 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, MutationCo
ntext)，可以使用 > MutationContext#addVertexRequest(Vertex)向图中请求添
加点，使用link MutationContext#addEdgeRequest(WritableComparable,Edge
)向图中请求添加边，然后，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, WritableCo
mparable)
* 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.getAddedEd
geList()) { 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();
}
```

```
}
```

3.9 Java SDK简介

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。具体的接口介绍，请参见《大数据计算服务*(MaxCompute)*开发指南》。

3.10 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，即在用户代码中直接使用上面的这些方法会报**`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()
```

3.11 Spark使用指南

3.11.1 概要

Spark on Maxcompute是阿里云开发的基于Maxcompute平台无缝运行使用spark的方案，其很大程度上丰富了Maxcompute产品的功能体系。**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。

3.11.2 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*。

3.11.3 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。

3.11.3.1 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。

```
.
|-- R
|-- RELEASE
|-- __spark_libs__.zip
|-- bin
|-- conf
|-- cupid
|-- derby.log
|-- examples
|-- jars
|-- logs
|-- metastore_db
|-- python
|-- sbin
|-- yarn
```

3.11.3.2 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
```

对于SparkR用户，需要安装R至*/home/admin/R*目录，并设置path：

```
export PATH=/home/admin/R/bin/:$PATH
```

对于pyspark用户，选装python2.7，并设置path：

```
export PATH=/path/to/python/bin/:$PATH
```

3.11.3.3 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。

```
# Example:
# spark.master                     spark://master:7077
# spark.eventLog.enabled           true
# spark.eventLog.dir               hdfs://namenode:8021/directory
# spark.executor.instances=5
# spark.executor.cores=4
# spark.executor.memory=6g
# spark.driver.cores=4
# spark.driver.memory=5g
# spark.executor.extraJavaOptions  -XX:+PrintGCDetails -Dkey=value -
Dnumbers="one two three"

# OdpsAccount Info Setting
spark.hadoop.odps.project.name =
spark.hadoop.odps.access.id =
spark.hadoop.odps.access.key =
spark.hadoop.odps.end.point =
# spark.hadoop.odps.moye.trackurl.host =
# spark.hadoop.odps.cupid.webproxy.endpoint =
spark.sql.catalogImplementation=odps
spark.hadoop.odps.cupid.bearer.token.enable=false
spark.hadoop.odps.exec.dynamic.partition.mode=nonstrict

# SparkR Setting
# odps.cupid.spark.r.archive=/path/to/R-PreCompile-Package.zip
# spark.r.command=/home/admin/R/bin/Rscript

# 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
# spark.hadoop.fs.oss.accessKeyId=
# spark.hadoop.fs.oss.accessKeySecret=
# spark.hadoop.fs.oss.endpoint=
```

3.11.4 快速开始

下面将演示如何快速使用**Spark on MaxCompute**进行工作。

1. 获取并解压spark包，spark包的下载地址为：*Spark on MaxCompute*。
2. 设置环境变量。

```
export SPARK_HOME=/path/to/spark-2.1.0-private-cloud-v3.1.0
export JAVA_HOME=/path/to/java/
```

3. 设置**spark-defaults.conf**。

```
cp $SPARK_HOME/conf/spark-defaults.conf.template $SPARK_HOME/conf/
spark-defaults.conf
```

编辑**$SPARK_HOME/conf/spark-defaults.conf**，将空白部分填上即可。

```
# OdpsAccount Info Setting
spark.hadoop.odps.project.name=
spark.hadoop.odps.access.id=
spark.hadoop.odps.access.key=
spark.hadoop.odps.end.point=
#spark.hadoop.odps.moye.trackurl.host=
#spark.hadoop.odps.cupid.webproxy.endpoint=
spark.sql.catalogImplementation=odps

# spark-shell Setting
spark.driver.extraJavaOptions -Dscala.repl.reader=com.aliyun.odps.
spark_repl.OdpsInteractiveReader -Dscala.usejavacp=true

# SparkR Setting
# odps.cupid.spark.r.archive=/path/to/R-PreCompile-Package.zip

# 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

# spark.r.command=/home/admin/R/bin/Rscript
# spark.hadoop.odps.cupid.disk.driver.enable=false
spark.hadoop.odps.cupid.bearer.token.enable=false
spark.hadoop.odps.exec.dynamic.partition.mode=nonstrict
```

4. 准备**spark-example**。

```
git clone https://github.com/aliyun/aliyun-cupid-sdk.git
cd aliyun-cupid-sdk
mvn -T 1C clean install -DskipTests
```

5. 运行SparkPi。

```
cd $SPARK_HOME
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
```

成功标准：看到以下输出，可能会有其他日志一并输出。

```
18/02/09 15:52:28 INFO Client: Application report for applicatio
n_1518162700322_1635034099 (state: FINISHED)
```

```
18/02/09 15:52:28 INFO Client:
  client token: N/A
  diagnostics: N/A
  ApplicationMaster host: ***
  ApplicationMaster RPC port: ***
  queue: queue
  start time: 1518162732343
  final status: SUCCEEDED
  tracking URL: http://***:80/proxyview/jobview/?h=http://***:80/api
&p=odps_smoke_test&i=20180209075148695gbkp8e01&t=spark&id=applicatio
n_1518162700322_1635034099&metaname=20180209075148695gbkp8e01&token=
YkhjNXJWZ0dvdzVScXVFQWpCQWMra1RSZHVFPSxPRFBTX09CTzoxMzY1OTM3MTUwNzcyMj
EzLDE1MTg0MjE5MzUseyJTdGF0ZW1lbnQiOlt7IkFjdGlvbiI6WyJvZHBzOlJlYWQiXSwi
RWZmZWN0IjoiQWxsb3ciLCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnByb2plY3RzL29kcH
Nfc21va2VfdGVzdC9pbnN0YW5jZXMvMjAxODAyMDkwNzUxNDg2OTVnYmtwOGUwMSJdfV0s
IlZlcnNpb24iOiIxIn0=
  user: user
18/02/09 15:52:28 INFO ShutdownHookManager: Shutdown hook called
18/02/09 15:52:28 INFO ShutdownHookManager: Deleting directory /tmp/
spark-d77416ad-79a8-49f7-931d-0533663b5d85
```

3.11.5 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。

```
git clone https://github.com/aliyun/aliyun-cupid-sdk.git
-- 下载GitHub项目。
cd aliyun-cupid-sdk
mvn -T 1C clean install -DskipTests
-- 编译GitHub项目。
```

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。

3.11.5.1 WordCount案例

Spark运行简单的WordCount。

案例代码如下。

```
package com.aliyun.odps.spark.examples

import org.apache.spark.sql.SparkSession

object WordCount {
  def main(args: Array[String]) {
    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
```

3.11.5.2 访问OSS案例

Spark可以访问OSS数据，示例如下。

案例代码如下。

```
package com.aliyun.odps.spark.examples.oss

import org.apache.spark.sql.SparkSession

object SparkUnstructuredDataCompute {
  def main(args: Array[String]) {
    val spark = SparkSession
      .builder()
      .appName("SparkUnstructuredDataCompute")
      .config("spark.hadoop.fs.oss.accessKeyId", "***")
      .config("spark.hadoop.fs.oss.accessKeySecret", "***")
      .config("spark.hadoop.fs.oss.endpoint", "oss-cn-hangzhou-zmf.
aliyuncs.com")
      .getOrCreate()

    val sc = spark.sparkContext
    try {
      val pathIn = "oss://bucket/inputdata/"
      val inputData = sc.textFile(pathIn, 5)
      val cnt = inputData.count
      println(s"count: $cnt")
    } finally {
      sc.stop()
    }
  }
}
```

提交作业的运行命令如下。

```
./bin/spark-submit
--jars cupid/hadoop-aliyun-package-3.0.0-alpha2-odps-jar-with-
dependencies.jar
--class com.aliyun.odps.spark.examples.oss.SparkUnstructuredDat
aCompute
```

```
/path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-
examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

3.11.5.3 MaxCompute Table ReadWrite案例

读写MaxCompute Table，转化为Spark RDD。

注意：

案例Demo中的Project/Table必须存在，或者更改为对应的Project/Table。

案例代码如下。

```
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

object OdpsTableReadWrite {
  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
      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
      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
      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
      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
      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
      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
      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_part
ition_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。

3.11.5.4 MaxCompute Table Spark-SQL案例

使用sqlContext读写MaxCompute Table。

注意：

- 案例Demo中的Project/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。

```
package com.aliyun.odps.spark.examples

import org.apache.spark.sql.SparkSession

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
```

3.11.5.5 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。

```
package com.aliyun.odps.spark.examples

import com.aliyun.odps.cupid.client.spark.client.CupidSparkClientRunn
er

object SparkClientNormalFT {
  def main(args: Array[String]) {
    val cupidSparkClient = CupidSparkClientRunner.getReadyCupidSparkCl
ient()
    val jarPath = args(0) //client-jobexamples jar path
    val sparkClientNormalApp = new SparkClientNormalApp(cupidSpark
Client)
    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
```

3.11.5.6 MaxCompute Table PySpark案例

PySpark读写MaxCompute Table。

案例代码如下。

```
from odps.odps_sdk import OdpsOps
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext, DataFrame

if __name__ == '__main__':
    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"

    normal_df = OdpsOps.read_odps_table(sql_context, project_name,
in_table_name)

    for i in normal_df.sample(False, 0.01).collect():
        print i
    print "Read normal odps table finished"

    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
```

3.11.5.7 Mllib案例

Mllib的model建议使用OSS进行读写。

案例代码如下。

```
package com.aliyun.odps.spark.examples.mllib

import org.apache.spark.mllib.clustering.KMeans._
import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.sql.SparkSession
```

```
object KmeansModelSaveToOss {
  val modelOssDir = "oss://bucket/kmeans-model"

  def main(args: Array[String]) {

    //1. train and save the model
    val spark = SparkSession
      .builder()
      .appName("KmeansModelSaveToOss")
      .config("spark.hadoop.fs.oss.accessKeyId", "***")
      .config("spark.hadoop.fs.oss.accessKeySecret", "***")
      .config("spark.hadoop.fs.oss.endpoint", "***")
      .getOrCreate()

    val sc = spark.sparkContext
    val points = Seq(
      Vectors.dense(0.0, 0.0),
      Vectors.dense(0.0, 0.1),
      Vectors.dense(0.1, 0.0),
      Vectors.dense(9.0, 0.0),
      Vectors.dense(9.0, 0.2),
      Vectors.dense(9.2, 0.0)
    )
    val rdd = sc.parallelize(points, 3)
    val initMode = K_MEANS_PARALLEL
    val model = KMeans.train(rdd, k = 2, maxIterations = 2, runs = 1,
initMode)
    val predictResult1 = rdd.map(feature => "cluster id: " + model
.predict(feature) + " feature:" + feature.toArray.mkString(",")).
collect
    println("modelOssDir=" + modelOssDir)
    model.save(sc, modelOssDir)

    //2. predict from the oss model
    val modelLoadOss = KMeansModel.load(sc, modelOssDir)
    val predictResult2 = rdd.map(feature => "cluster id: " +
modelLoadOss.predict(feature) + " feature:" + feature.toArray.mkString
(",")).collect
    assert(predictResult1.size == predictResult2.size)
    predictResult2.foreach(result2 => assert(predictResult1.contains(
result2)))
  }
}
```

提交作业的运行命令如下。

```
./bin/spark-submit
--jars cupid/hadoop-aliyun-package-3.0.0-alpha2-odps-jar-with-
dependencies.jar
--class com.aliyun.odps.spark.examples.mllib.KmeansModelSaveToOss
/path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-
examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

3.11.5.8 pyspark交互式案例

只有能直连计算集群的机器上可执行pyspark。

选装python2.7，并设置path：

```
export PATH=/path/to/python/bin/:$PATH
```

启动命令：

```
bin/pyspark --master yarn-client
```

交互执行：

```
df=spark.sql("select * from spark_user_data")
df.show()
```

3.11.5.9 spark-shell交互式案例（读表）

只有能直连计算集群的机器上可执行spark-shell。

启动命令：

```
bin/spark-shell --master yarn
```

交互执行：

```
sc.parallelize(0 to 100, 2).collect
sql("show tables").show
sql("select * from spark_user_data").show(200,100)
```

3.11.5.10 spark-shell交互式案例（mlib+OSS读写）

只有能直连计算集群的机器上可执行spark-shell。

在**conf/spark-defaults.conf**中加入以下配置：

```
spark.hadoop.fs.oss.accessKeyId=***
spark.hadoop.fs.oss.accessKeySecret=***
spark.hadoop.fs.oss.endpoint=***
```

启动命令：

```
bin/spark-shell --master yarn --jars cupid/hadoop-aliyun-package-3.0.0
-alpha2-odps-jar-with-dependencies.jar
```

交互执行：

```
import org.apache.spark.mllib.clustering.KMeans._
import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
import org.apache.spark.mllib.linalg.Vectors
val modelOssDir = "oss://your_bucket/kmeans-model"
val points = Seq(
      Vectors.dense(0.0, 0.0),
      Vectors.dense(0.0, 0.1),
```

```
      Vectors.dense(0.1, 0.0),
      Vectors.dense(9.0, 0.0),
      Vectors.dense(9.0, 0.2),
      Vectors.dense(9.2, 0.0)
    )
val rdd = sc.parallelize(points, 3)
val initMode = K_MEANS_PARALLEL
val model = KMeans.train(rdd, k = 2, maxIterations = 2, runs = 1,
initMode)
val predictResult1 = rdd.map(feature => "cluster id: " + model.predict
(feature) + " feature:" + feature.toArray.mkString(",")).collect
println("modelOssDir=" + modelOssDir)
model.save(sc, modelOssDir)
val modelLoadOss = KMeansModel.load(sc, modelOssDir)
val predictResult2 = rdd.map(feature => "cluster id: " + modelLoadO
ss.predict(feature) + " feature:" + feature.toArray.mkString(",")).
collect
assert(predictResult1.size == predictResult2.size)
predictResult2.foreach(result2 => assert(predictResult1.contains(
result2)))
```

3.11.5.11 sparkR交互式案例

只有能直连计算集群的机器上可执行sparkR，并且需要安装R至*/home/admin/R*目录，同时设置path：

```
export PATH=/home/admin/R/bin/:$PATH
```

启动命令：

```
bin/sparkR --master yarn -—archives ./R/R.zip
```

交互执行：

```
df <- as.DataFrame(faithful)
df
head(select(df, df$eruptions))
head(select(df, "eruptions"))
head(filter(df, df$waiting < 50))

results <- sql("FROM spark_user_data SELECT *")
head(results)
```

3.11.5.12 GraphX--PageRank案例

支持原生的graphx。

案例代码如下。

```
package com.aliyun.odps.spark.examples.graphx

import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD
```

```
object PageRank {
  def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setAppName("pagerank")
    val sc = new SparkContext(conf)

    // construct vertices
    val users: RDD[(VertexId, Array[String])] = sc.parallelize(List(
      "1,BarackObama,Barack Obama",
      "2,ladygaga,Goddess of Love",
      "3,jeresig,John Resig",
      "4,justinbieber,Justin Bieber",
      "6,matei_zaharia,Matei Zaharia",
      "7,odersky,Martin Odersky",
      "8,anonsys"
    ).map(line => line.split(",")).map(parts => (parts.head.toLong,
parts.tail)))

    // construct edges
    val followers: RDD[Edge[Double]] = sc.parallelize(Array(
      Edge(2L,1L,1.0),
      Edge(4L,1L,1.0),
      Edge(1L,2L,1.0),
      Edge(6L,3L,1.0),
      Edge(7L,3L,1.0),
      Edge(7L,6L,1.0),
      Edge(6L,7L,1.0),
      Edge(3L,7L,1.0)
    ))

    // construct graph
    val followerGraph: Graph[Array[String], Double] = Graph(users,
followers)

    // restrict the graph to users with usernames and names
    val subgraph = followerGraph.subgraph(vpred = (vid, attr) => attr.
size == 2)

    // compute PageRank
    val pageRankGraph = subgraph.pageRank(0.001)

    // get attributes of the top pagerank users
    val userInfoWithPageRank = subgraph.outerJoinVertices(pageRankGr
aph.vertices) {
      case (uid, attrList, Some(pr)) => (pr, attrList.toList)
      case (uid, attrList, None) => (0.0, attrList.toList)
    }

    println(userInfoWithPageRank.vertices.top(5)(Ordering.by(_._2._1
)).mkString("\n"))
  }
}
```

提交作业的运行命令如下。

```
bin/spark-submit \
--master yarn-cluster \
--class com.aliyun.odps.spark.examples.graphx.PageRank \
```

```
/path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-
examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

3.11.5.13 Spark Streaming--NetworkWordCount案例

支持原生Spark Streaming，以NetworkWordCount为例，首先本地需安装Netcat，并执行以下命令：

```
$ nc -lk 9999
```

此时，控制台的输入将成为SparkStreaming的输入。

案例代码如下。

```
package com.aliyun.odps.spark.examples.streaming

import org.apache.spark.SparkConf
import org.apache.spark.examples.streaming.StreamingExamples
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.{Seconds, StreamingContext}

object NetworkWordCount {
  def main(args: Array[String]) {
    if (args.length < 2) {
      System.err.println("Usage: NetworkWordCount <hostname> <port>")
      System.exit(1)
    }

    StreamingExamples.setStreamingLogLevels()

    // Create the context with a 1 second batch size
    val sparkConf = new SparkConf().setAppName("NetworkWordCount")
    val ssc = new StreamingContext(sparkConf, Seconds(1))

    // Create a socket stream on target ip:port and count the
    // words in input stream of \n delimited text (eg. generated by '
nc')
    // Note that no duplication in storage level only for running
locally.
    // Replication necessary in distributed scenario for fault
tolerance.
    val lines = ssc.socketTextStream(args(0), args(1).toInt,
StorageLevel.MEMORY_AND_DISK_SER)
    val words = lines.flatMap(_.split(" "))
    val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
    wordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```

提交作业的运行命令如下。

```
bin/spark-submit \
--master local[4] \
--class com.aliyun.odps.spark.examples.streaming.NetworkWordCount \
```

```
/path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-
examples_2.11-1.0.0-SNAPSHOT-shaded.jar localhost 9999
```

3.11.6 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>
```

3.11.7 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。

3.11.7.1 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

```
spark.hadoop.odps.cupid.engine.running.type=longtime
-- 配置成longtime才不会被回收。
spark.hadoop.odps.cupid.job.capability.duration.hours=25920
-- 配置时间长度。
spark.yarn.maxAppAttempts=10
-- 配置failOver最大重试次数。
spark.streaming.receiver.writeAheadLog.enable=true
-- 是否启用writeAheadLog模式，能够保证数据不丢失，但是效率会降低。
```

3.11.7.2 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=20170828065141675g5h
4t6u1&t=spark&id= application_1503903039442_1185611255&metaname
=20170828065141675g5h4t6u1&token= L0dSMHRkSlNXS2ZkeFE1UkVsckthTT
ZQWHV3PSxPRFBTX09CTzoxMDU5NTgyNzI0MzIyOT k5LDE1MDQxNjIzODMsey
JTdGF0ZW1lbnQiOlt7IkFjdGlvbiI6WyJvZHBzOlJlYWQiXSwi RWZmZWN0Ij
oiQWxsb3ciLCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnByb2plY3RzL29kcH
NfcHVibGljX2Rldi9pbnN0YW5jZXMvMjAxNzA4MjgwNjUxNDE2NzVnNWg0dDZ1MSJdfV0s
 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。

3.11.8 Spark接口支持

Spark目前支持spark shell、spark r、spark sql及spark jdbc四种接口。

3.11.8.1 Spark Shell

执行如下命令，启动应用。

```
$cd $SPARK_HOME
-- 进入spark目录。
$bin/spark-shell --master yarn
-- 选择运行模式并启动应用。
```

应用示例如下：

```
sc.parallelize(0 to 100, 2).collect
sql("show tables").show
sql("select * from spark_user_data").show(200,100)
```

3.11.8.2 Spark R

执行如下命令，启动应用。

```
$mkdir -p /home/admin/R && unzip ./R/R.zip -d /home/admin/R/
-- 创建一个目录R并解压该目录下的R.zip。
$export PATH=/home/admin/R/bin/:$PATH
-- 设置环境变量。
$bin/sparkR --master yarn --archives ./R/R.zip
-- 选择运行模式并启动应用。
```

应用示例如下：

```
df <- as.DataFrame(faithful)
df
head(select(df, df$eruptions))
head(select(df, "eruptions"))
head(filter(df, df$waiting < 50))
results <- sql("FROM spark_user_data SELECT *")
```

```
head(results)
```

3.11.8.3 Spark SQL

执行如下命令，启动应用。

```
$cd $SPARK_HOME
-- 进入spark目录。
$bin/spark-sql --master yarn
-- 选择运行模式并启动应用。
```

应用示例如下：

```
show tables;
select * from spark_user_data limit 3;;
quit;
```

3.11.8.4 Spark JDBC

执行如下命令，启动应用。

```
$sbin/stop-thriftserver.sh
-- 停止线程。
$sbin/start-thriftserver.sh
-- 重启线程。
$bin/beeline
-- 启动应用。
```

应用示例如下：

```
!connect jdbc:hive2://localhost:10000/odps_smoke_test
show tables;
select * from mr_input limit 3;
!quit
```

3.12 非结构化数据处理使用指南

MaxCompute团队依托MaxCompute系统架构，引入非结构化数据处理框架，解决MaxComputeSQL面对MaxCompute表外的各种用户数据时（例如：OSS上的非结构化数据或者来自TableStore的非结构化数据），需要首先通过各种工具导入MaxCompute表，才能在其上面进行计算，无法直接处理的问题。

用户只需要通过一条简单的DDL语句，在MaxCompute上创建一张外部表，建立MaxCompute表与外部数据源的关联，即可以为各种数据在MaxCompute上的计算处理提供入口。并且创建好的外部表可以像普通的MaxCompute表一样使用，充分利用MaxCompute SQL的强大计算功能。

本章节将以OSS数据和TableStore数据为例，介绍如何在MaxCompute上访问并处理非结构化数据。

3.12.1 如何在MaxCompute上访问并处理OSS非结构化数据

3.12.1.1 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。

3.12.1.2 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。

3.12.1.2.1 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。

3.12.1.2.2 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,-92.08174,9/14/2017 0:00,S
1,2,13,1,46.81006,-92.08174,9/14/2017 0:00,NE
1,3,48,1,46.81006,-92.08174,9/14/2017 0:00,NE
1,4,30,1,46.81006,-92.08174,9/14/2017 0:00,W
1,5,47,1,46.81006,-92.08174,9/14/2017 0:00,S
1,6,9,1,46.81006,-92.08174,9/14/2017 0:00,S
1,7,53,1,46.81006,-92.08174,9/14/2017 0:00,N
1,8,63,1,46.81006,-92.08174,9/14/2017 0:00,SW
1,9,4,1,46.81006,-92.08174,9/14/2017 0:00,NE
1,10,31,1,46.81006,-92.08174,9/14/2017 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。

3.12.1.3 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。

3.12.1.3.1 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。

3.12.1.3.2 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。完整实现可以参见：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
}
}
```

3.12.1.3.3 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;
```

3.12.1.3.4 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。
```

3.12.1.3.5 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|9/14/2017 0:00|S
1|2|13|1|46.81006|-92.08174|9/14/2017 0:00|NE
1|3|48|1|46.81006|-92.08174|9/14/2017 0:00|NE
1|4|30|1|46.81006|-92.08174|9/14/2017 0:00|W
1|5|47|1|46.81006|-92.08174|9/14/2017 0:00|S
1|6|9|1|46.81006|-92.08174|9/14/2017 0:00|S
1|7|53|1|46.81006|-92.08174|9/14/2017 0:00|N
1|8|63|1|46.81006|-92.08174|9/14/2017 0:00|SW
1|9|4|1|46.81006|-92.08174|9/14/2017 0:00|NE
1|10|31|1|46.81006|-92.08174|9/14/2017 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 |
+------------+------------+-----------+
```

3.12.1.4 进阶使用

相关的进阶使用示例如下。

3.12.1.4.1 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。

3.12.1.5 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。

3.12.1.5.1 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/
2017-09-14 08:03:35 128MB Standard oss://oss-odps-
test/log_data/year=2017/month=06/day=01/logfile
2017-09-14 08:04:12 127MB Standard oss://oss-odps-
test/log_data/year=2017/month=06/day=01/logfile.1
2017-09-14 08:05:02 118MB Standard oss://oss-odps-
test/log_data/year=2017/month=06/day=02/logfile
2017-09-14 08:06:45 123MB Standard oss://oss-odps-
test/log_data/year=2017/month=07/day=10/logfile
2017-09-14 08:07:11 115MB Standard oss://oss-odps-
test/log_data/year=2017/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 = '2017', month =
 '06', day = '01')
ALTER TABLE log_table_external ADD PARTITION (year = '2017', month =
 '06', day = '02')
ALTER TABLE log_table_external ADD PARTITION (year = '2017', month =
 '07', day = '10')
ALTER TABLE log_table_external ADD PARTITION (year = '2017', 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2017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 = '2017' 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2017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 = '2017' AND month > '06';
```

说明：

此时，只访问OSS上面存储的下半年的LOG。

3.12.1.5.2 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/
2017-09-14 08:03:35 128MB Standard oss://oss-odps-
test/log_data_customized/2017/06/01/logfile
2017-09-14 08:04:12 127MB Standard oss://oss-odps-
test/log_data_customized/2017/06/01/logfile.1
2017-09-14 08:05:02 118MB Standard oss://oss-odps-
test/log_data_customized/2017/06/02/logfile
2017-09-14 08:06:45 123MB Standard oss://oss-odps-
test/log_data_customized/2017/07/10/logfile
2017-09-14 08:07:11 115MB Standard oss://oss-odps-
test/log_data_customized/2017/08/08/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 = '2017', month =
 '06', day = '01')
LOCATION 'oss://<ak_id>:<ak_key>@oss-cn-shanghai-internal.aliyuncs.
com/oss-odps-test/log_data_customized/2017/06/01/';
```

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。

3.12.1.5.3 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。

3.12.1.6 非结构化数据的输出

相关的操作示例如下。

3.12.1.6.1 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。

3.12.1.6.2 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。

3.12.1.6.3 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。

3.12.1.6.4 以MaxCompute为计算介质，实现不同存储介质之间的数据转移

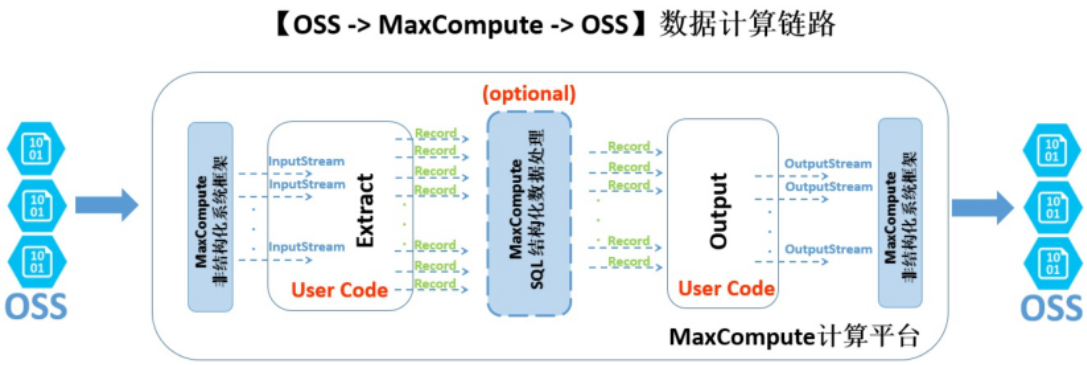
External Table作为一个MaxCompute与外部存储介质的切入点，提供了对多种外部数据（包括OSS，TableStore等）的读取和写出的功能。依赖于接入点，我们可以实现各种各样的数据计算/存储链路，例如：

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。

3.12.2 如何在MaxCompute上访问并处理TableStore数据

3.12.2.1 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是在飞天操作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数据。

3.12.2.2 MaxCompute对TableStore数据进行读取和计算

相关的操作示例如下。

3.12.2.2.1 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，可以先自行了解一些TableStore的基本概念（例如主键、分区键、属性列等）。

3.12.2.2.2 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。
```

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。

3.12.2.2.3 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_order
date <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。

3.12.2.3 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_custk
ey,
'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。

3.13 安全指南

3.13.1 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

3.13.2 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见：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。详情请参见：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。详情请参见：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。

3.13.3 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。

3.13.4 项目空间的用户及授权管理

3.13.4.1 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，详情请参见：跨项目空间的资源分享。

3.13.4.2 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见：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。

3.13.4.3 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。

3.13.4.4 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：

表 3-35: 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。

客体（Object）	操作（Action）	说明
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。
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

客体（Object）	操作（Action）	说明
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。
```

3.13.4.5 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，详细信息请参见：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
```

3.13.5 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，详情请参见：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。

3.13.5.1 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。

3.13.5.1.1 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>
  ```

3.13.5.1.2 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"
}]
```

```
}
```

3.13.6 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。

3.13.6.1 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。

3.13.6.2 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。
```

3.13.6.3 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。

3.13.7 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。
```

3.13.8 Policy权限策略

3.13.8.1 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“2017-11-11 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$alice@aliyun.com",
  "Action":["odps:CreateTable","odps:CreateInstance","odps:List"],
  "Resource":"acs:odps:*:projects/test_project",
  "Condition":{ "DateLessThan": {
  "acs:CurrentTime":"2017-11-11T23:59:59Z"
  },
  "IpAddress": { "acs:SourceIp":"10.32.180.0/23"
  }
  }
  },
  {
  "Effect":"Deny", "Principal":"ALIYUN$alice@aliyun.com", "Action":"
  odps:Drop", "Resource":"acs:odps:*:projects/test_project/tables/*"
  }]
  }
  ```json
  ```
- ```json
  {
  "Version": "1", "Statement": [{
  "Effect":"Allow", "Principal":"ALIYUN$alice@aliyun.com",
  "Action":["odps:CreateTable","odps:CreateInstance","odps:List"],
  "Resource":"acs:odps:*:projects/test_project",
  "Condition":{ "DateLessThan": {
  "acs:CurrentTime":"2017-11-11T23:59:59Z"
  },
  "IpAddress": { "acs:SourceIp":"10.32.180.0/23"
  }
  }
  },
  {
  "Effect":"Deny", "Principal":"ALIYUN$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":"2017-11-11T23:59:59Z"
  },
  "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。

3.13.8.2 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2017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2017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2017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2017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。

3.13.8.3 访问策略语言结构

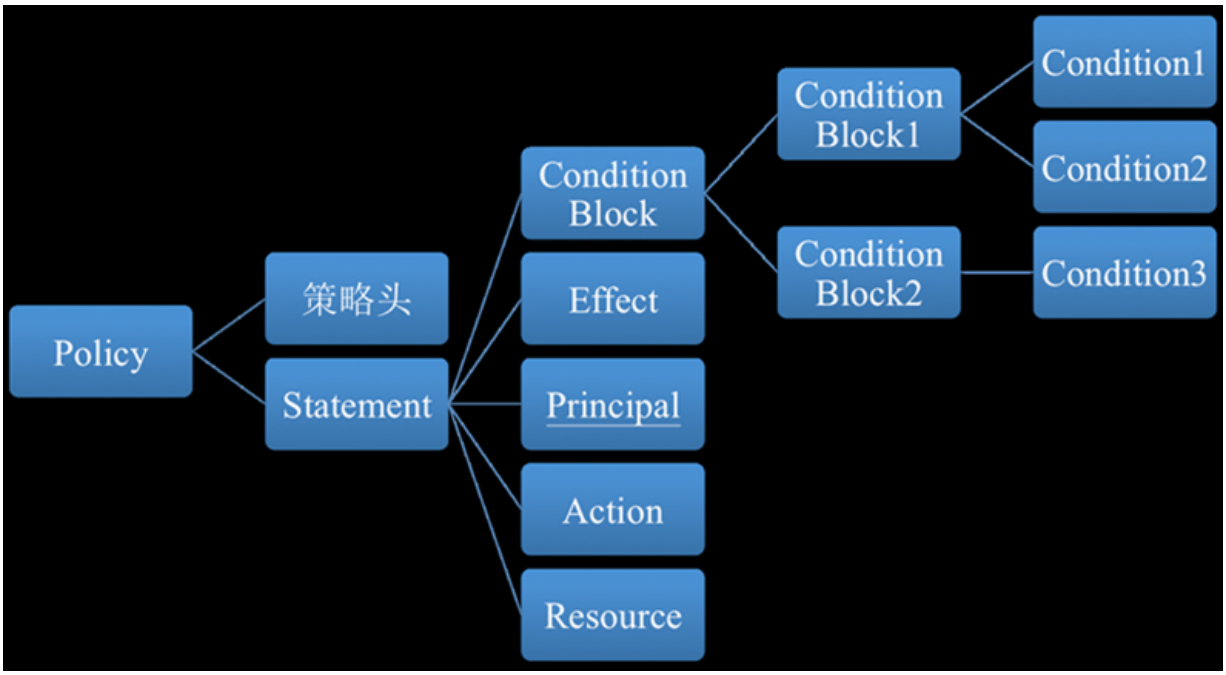
下图描述了访问策略的基本结构。一个策略（Policy）由如下部分组成：

- 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 3-6: Policy 结构图



3.13.8.3.1 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。

3.13.8.3.2 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。

3.13.8.3.3 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
```

3.13.8.3.4 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：

表 3-36: 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2017-11-11T23:59:59Z。
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。

3.13.8.4 访问策略语言规范

3.13.8.4.1 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$bob@aliyun.com"
"Principal": ["ALIYUN$bob@aliyun.com","ALIYUN$jack@aliyun.com", "
TAOBAO$alice"]
```

3.13.8.4.2 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：

表 3-37: 命名实例

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。

3.13.8.4.3 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：

表 3-38: 命名实例

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。

3.13.8.4.4 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。

3.13.8.4.5 访问策略样例

Policy Sample：

```
{
"Version": "1",
"Statement": [{
"Effect":"Allow",
"Principal":"ALIYUN$alice@aliyun.com",
"Action":["odps:CreateTable","odps:CreateInstance","odps:List"],
"Resource":"acs:odps:*:projects/prj1",
"Condition":{ "DateLessThan": {
"acs:CurrentTime":"2017-11-11T23:59:59Z"
},
"IpAddress": { "acs:SourceIp":"10.32.180.0/23"
}
}
},
{
"Effect":"Deny", "Principal":"ALIYUN$alice@aliyun.com",
"Action":"odps:Drop",
"Resource":"acs:odps:*:projects/prj1/tables/*"
}]
}
```

说明：

授权用户alice@aliyun.com只能在2017-11-11T23:59:59Z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。

3.13.8.5 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2017-11-11 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。

3.13.8.6 应用限制

表 3-39: 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。

名称	限制数	说明
ALLOW_PROJECT_NUMBER_L IMIT_IN_ON E_PACKAGE	1024	一个Package允许被多少Project安装。
RESOURCE_NUMBER_LIMIT_I N_ONE_PACK AGE	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_I N_ONE_PROJECT	64	一个Project允许安装的Package数量。

3.13.9 安全相关语句汇总

3.13.9.1 项目空间的安全配置

鉴权配置

表 3-40: 命令

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

表 3-41: 命令

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。

语句	说明
add TrustedProject <projectName>	添加可信项目空间。
remove TrustedProject <projectName>	移除可信项目空间。

3.13.9.2 项目空间的权限管理

用户管理

表 3-42: 命令

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

表 3-43: 命令

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

表 3-44: 命令

语句	说明
grant <privList> on <objType> <objName> to user <username>	对用户授权。

语句	说明
grant <privList> on <objType> <objName> to role <rolename>	对角色授权。
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

表 3-45: 命令

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

表 3-46: 命令

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。

3.13.9.3 基于Package的资源分享

分享资源

表 3-47: 命令

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。
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

表 3-48: 命令

语句	说明
install package <pkgName>	安装Package。
uninstall package <pkgName >	卸载Package。

查看Package

表 3-49: 命令

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。

3.14 工具

3.14.1 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。

3.14.1.1 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：

- Access ID和Access Key是用户的云账号信息。
- 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;
```

3.14.1.2 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 = 20170528122432906gux77io3
     Log view:
     http://webconsole.odps.aliyun-inc.com:8080/logview/?h=http://
     service-corp.odps.aliyun- inc.com/api&podps_public_dev&i2017052
      8122432906gux77io3&tokenRnlrSzJoL242YW43dFFIc1dmb1ZWZzFxQ1R
     FPSxPRFBTX09CTzoxMDcwMDI1NjI3ODA1 NjI5LDE0MzM0MjA2NzMseyJTdGF0ZW
     1lbnQiOlt7IkFjdGlvbiI6WyJvZHBzOlJlYWQiXSwiRWZmZWN0IjoiQWxsb3
     ciLCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnB yb2plY3RzL29kcHNfcHV
     ibGljX2Rld i9pbnN0YW5jZXMvMjAxNTA1MjgxMjI0MzI5MDZndXg3N2lvMyJ
     dfV0sIlZlcnNpb24iOiIxIn0=
     OK
     ID = 20170528122439318gcmkk6u1
     Log view:
     ```

```
http://webconsole.odps.aliyun-inc.com:8080/logview/?h=http://
service-corp.odps.aliyun- inc.com/api&podps_public_dev&i2017052
 8122439318gcmkk6u1&tokendSt0RXdlV0M5YjZET2I1MnJuUFkzWDN1aWp
zPSxPRFBTX09CTzoxMDcwMDI1NjI3ODA1NjI5LDE0MzM0MjA2ODAseyJTdGF
0ZW1lbnQiOlt7IkFjdGlvbiI6WyJvZHBzOlJlYWQiXSwiRWZmZWN0IjoiQWx
sb3ciLCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnB yb2plY3RzL29kcHNfcHV
ibGljX2Rld i9pbnN0YW5jZXMvMjAxNTA1MjgxMjI0MzkzMThnY21razZ1MSJ
dfV0sIlZlcnNpb24iOiIxIn0=
OK
ID = 20170528122440389g98cmlmf
Log view:
http://webconsole.odps.aliyun-inc.com:8080/logview/?h=http://
service-corp.odps.aliyun- inc.com/api&podps_public_dev&i2017052
 8122440389g98cmlmf&tokenNWlwL0EvQThxUXhzcTRERDc5NFg0b2IxZ3Q
wPSxPRFBTX09CTzoxMDcwMDI1NjI3OD A1NjI5LDE0MzM0MjA2ODAseyJTdGF0
ZW1lbnQiOlt7IkFjdGlvbiI6WyJvZHBzOlJlYWQiXSwiRWZmZWN0IjoiQWxs
b3ciLCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnB yb2plY3RzL29kcHNfcHV
ibGljX2Rld i9pbnN0YW5jZXMvMjAxNTA1MjgxMjI0NDAzODlnOThjbWxtZiJ
dfV0sIlZlcnNpb24iOiIxIn0=
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
```

3.14.2 Eclipse开发插件

3.14.2.1 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

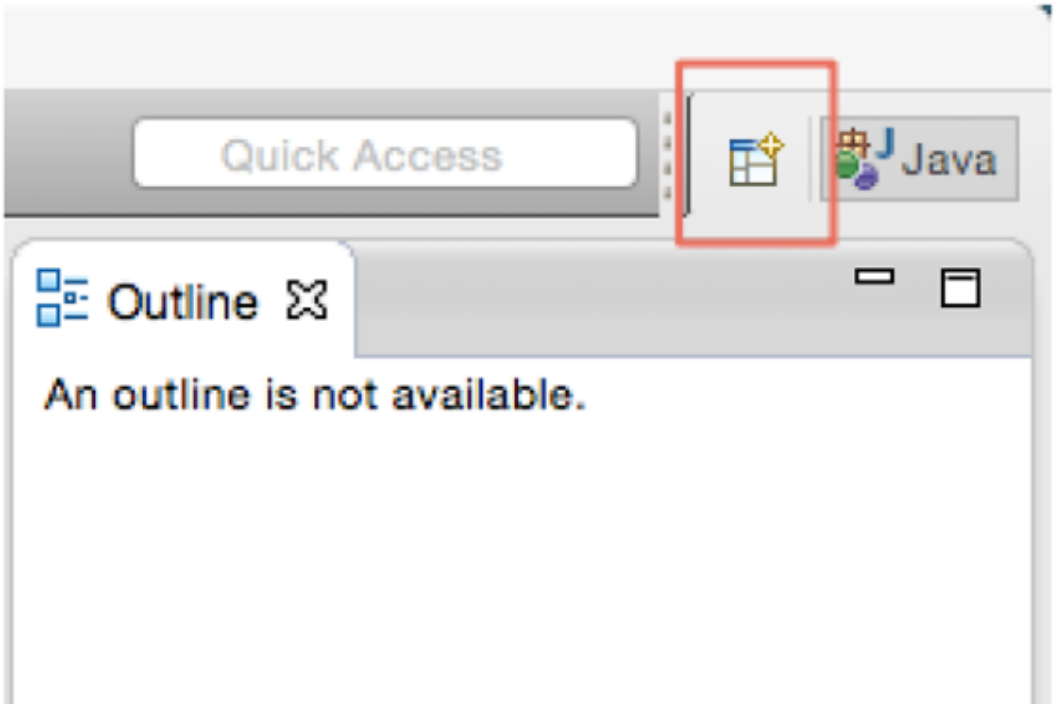
1. 将Eclipse软件包解压。解压后将看到如下的jar内容。

   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。

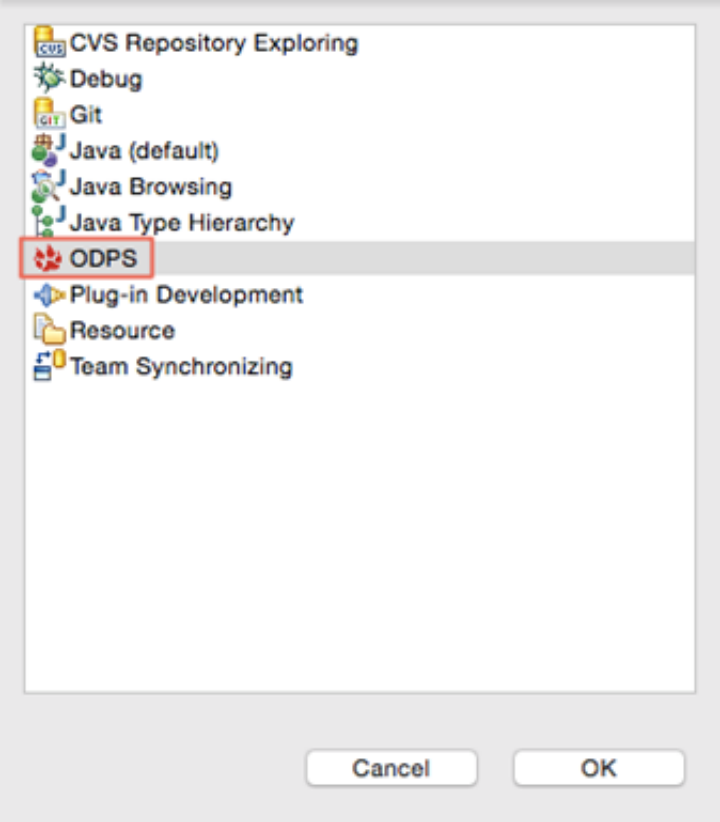
3. 打开Eclipse，单击右上角的打开透视图（Open Perspective），页面如下图所示。

图 3-7: Eclipse安装图1



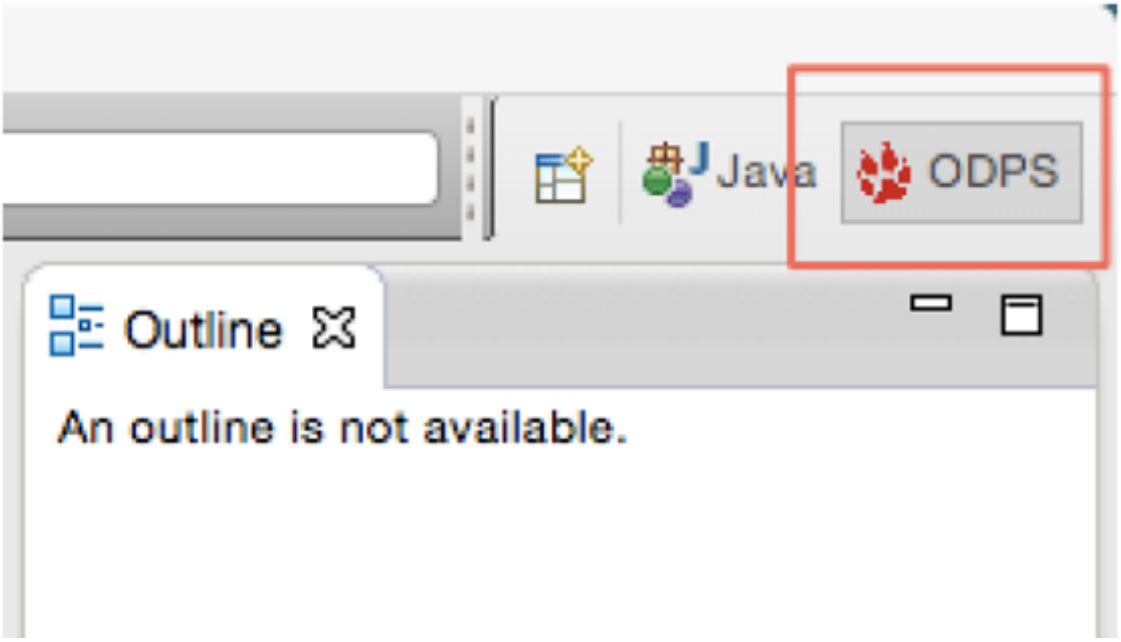
4. 在弹出的如下对话框中单击**ODPS**。

图 3-8: Eclipse安装图2



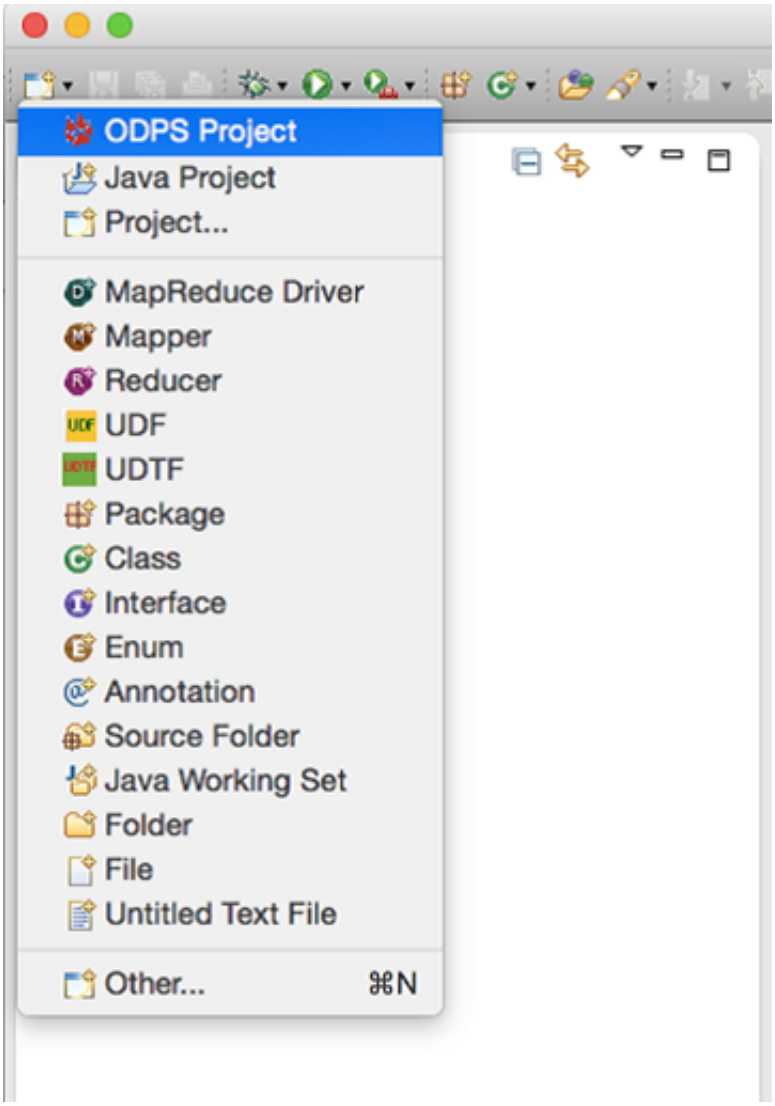
5. 单击**ODPS**后出现如下图的对话框。

图 3-9: 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 3-10: Eclipse安装图4



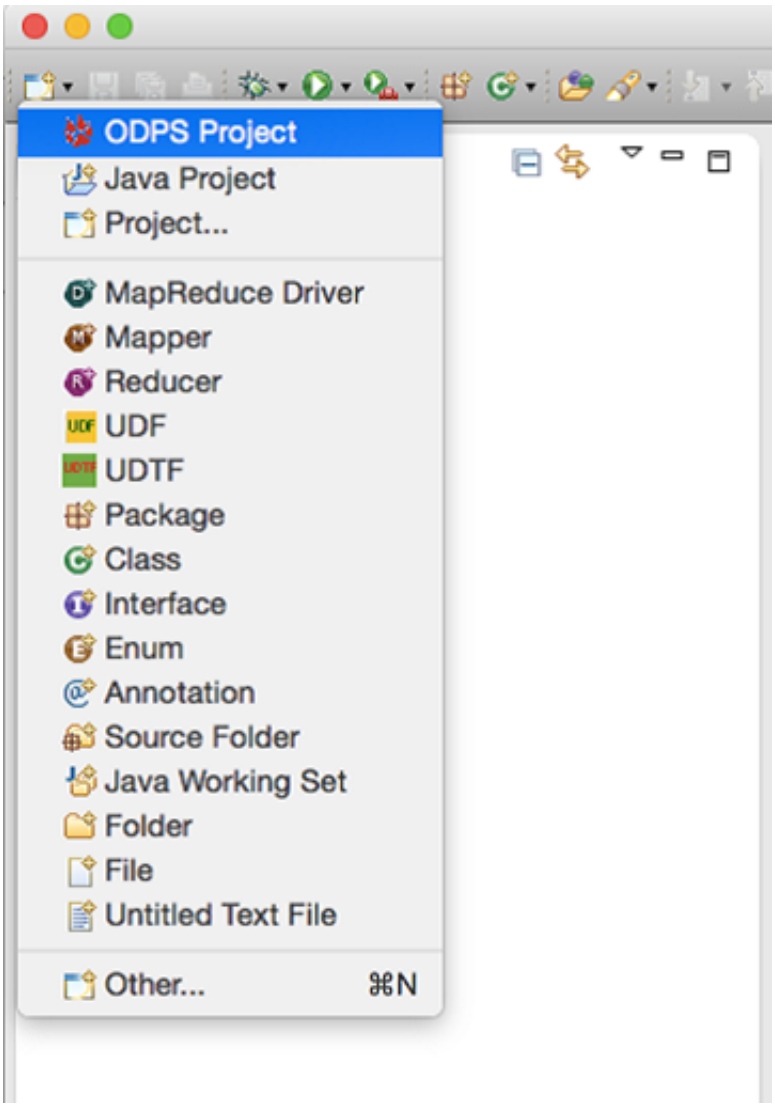
3.14.2.2 创建工程

3.14.2.2.1 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 3-11: Eclipse创建数据库步骤1

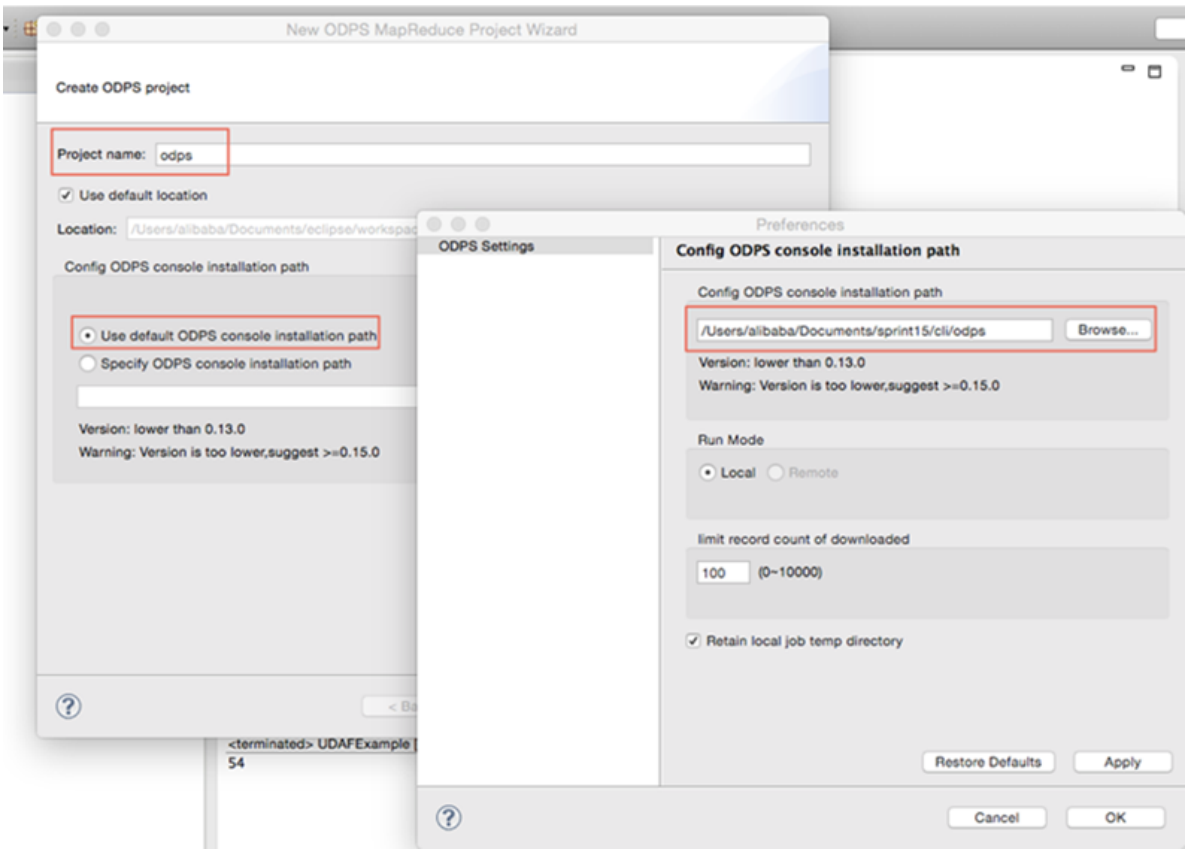


说明：

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 3-12: Eclipse创建数据库步骤2

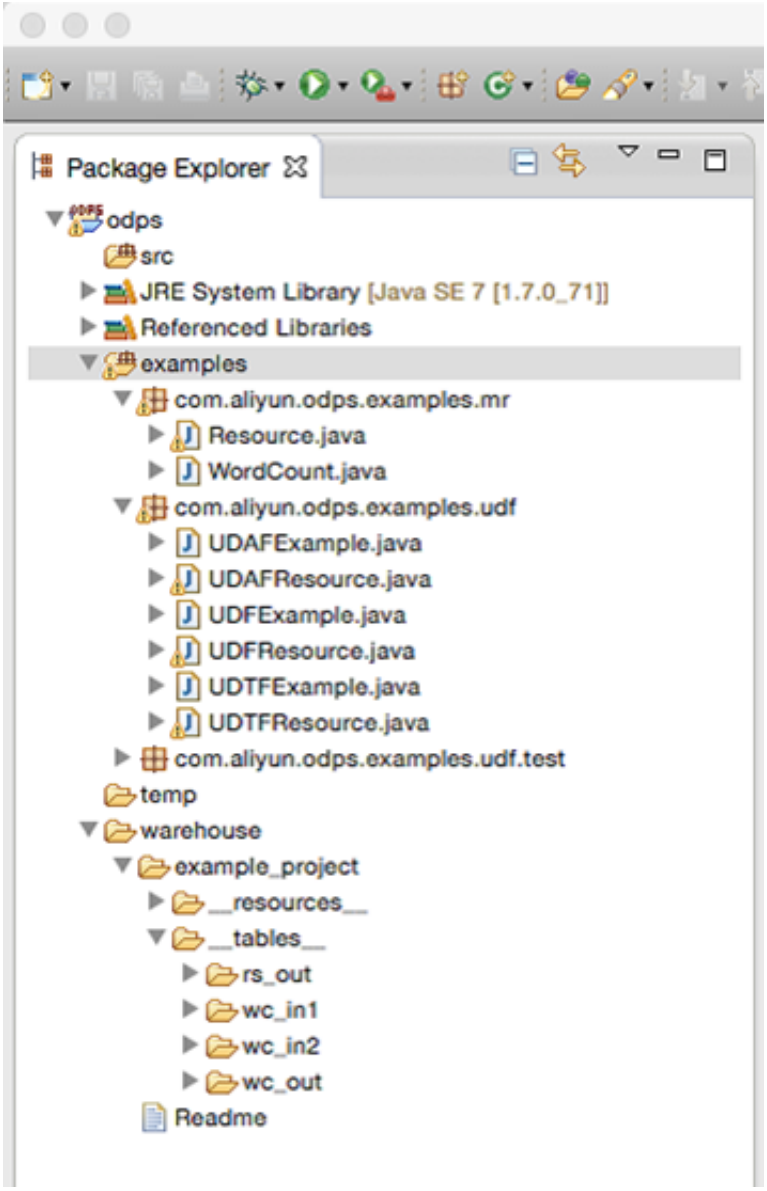


说明：

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 3-13: Eclipse创建数据库步骤3

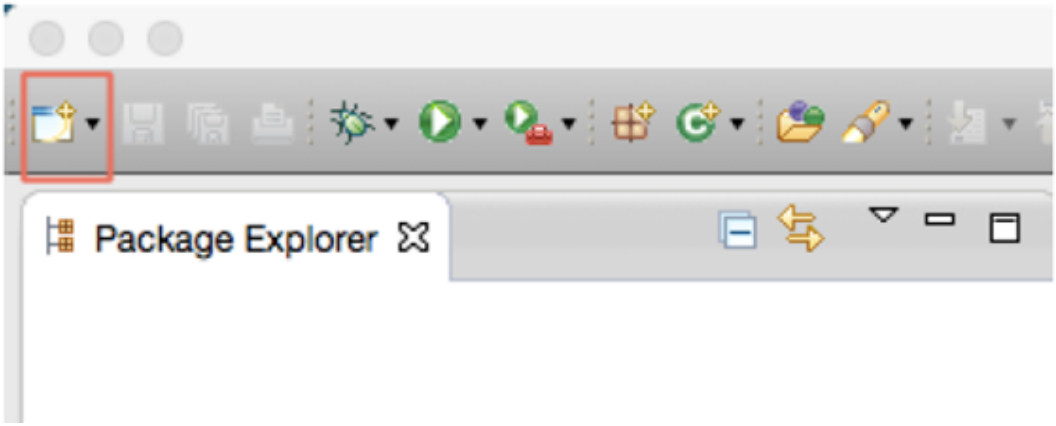


3.14.2.2.2 方式二

操作步骤

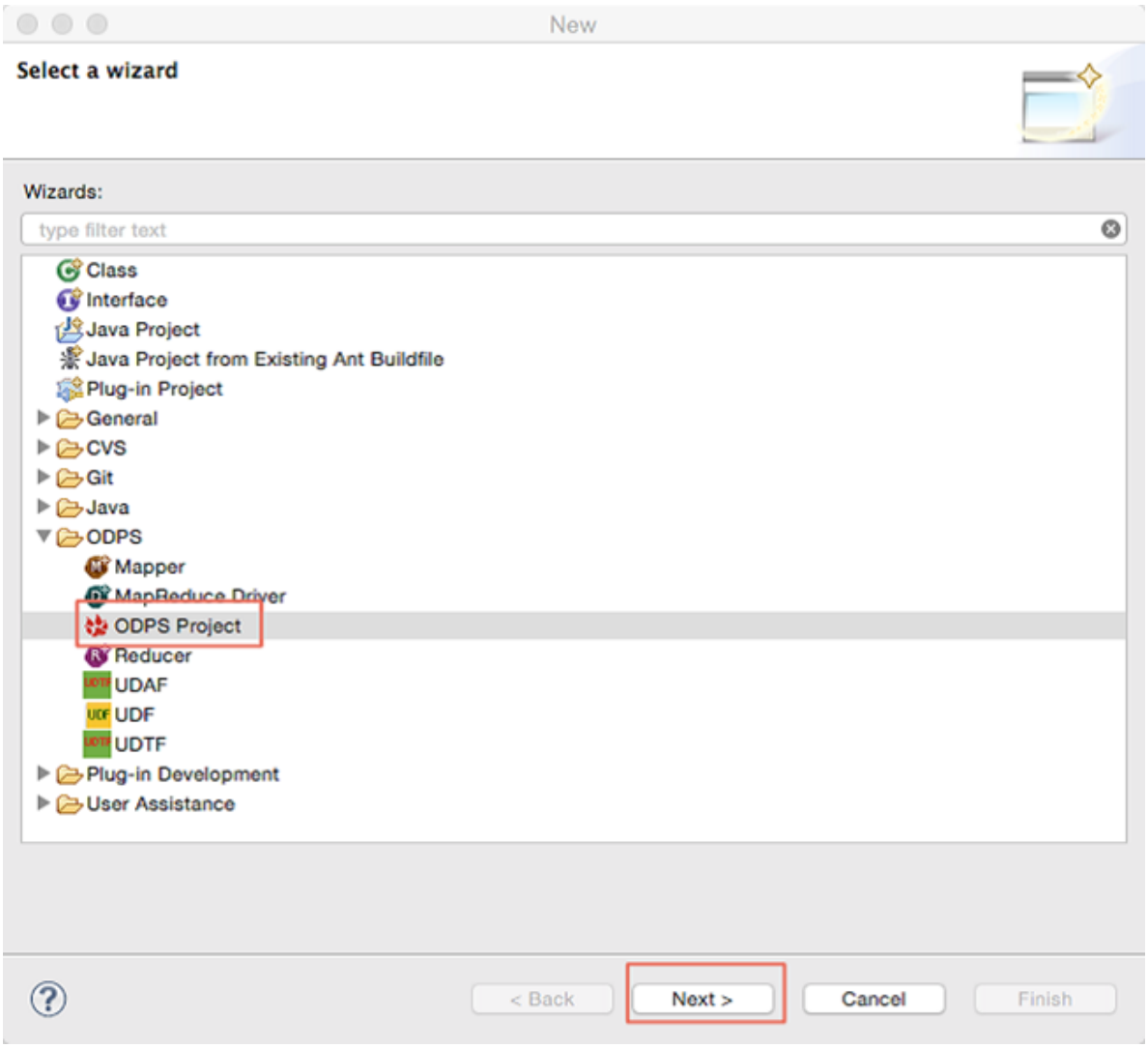
1. 在Eclipse直接单击左上角的**新建**，如下图所示。

图 3-14: 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 3-15: 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的功能介绍请参见：*MapReduce开发插件介绍*，UDF的程序编写请参见：*UDF开发插件介绍*。

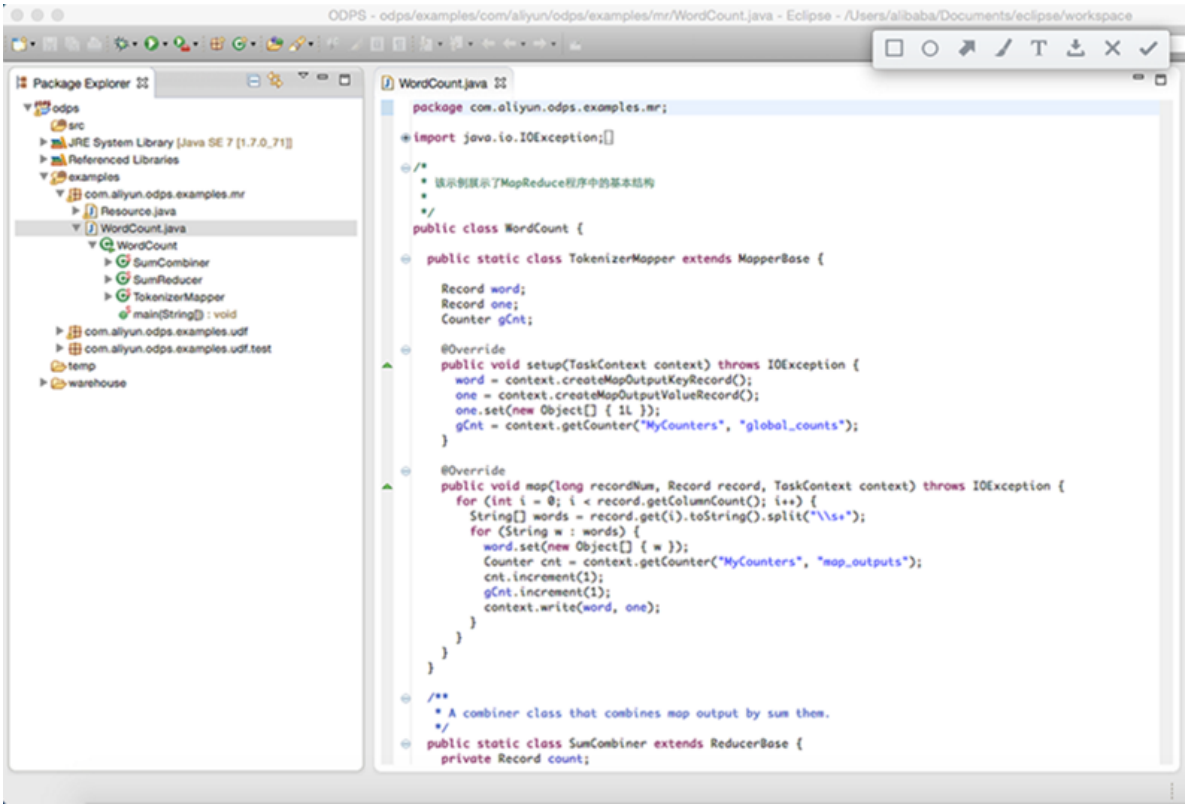
3.14.2.3 MapReduce开发插件介绍

3.14.2.3.1 快速运行WordCount示例

操作步骤

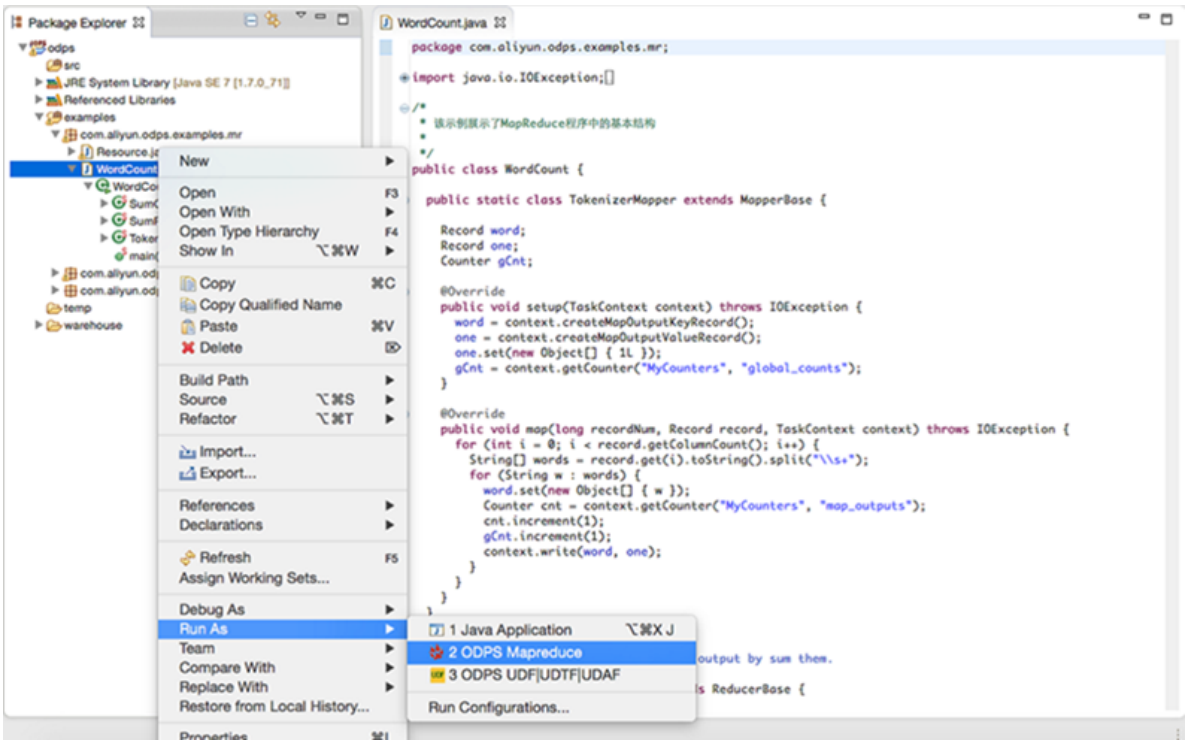
1. 选择ODPS项目中的WordCount示例，如下图所示。

图 3-16: WordCount 示例



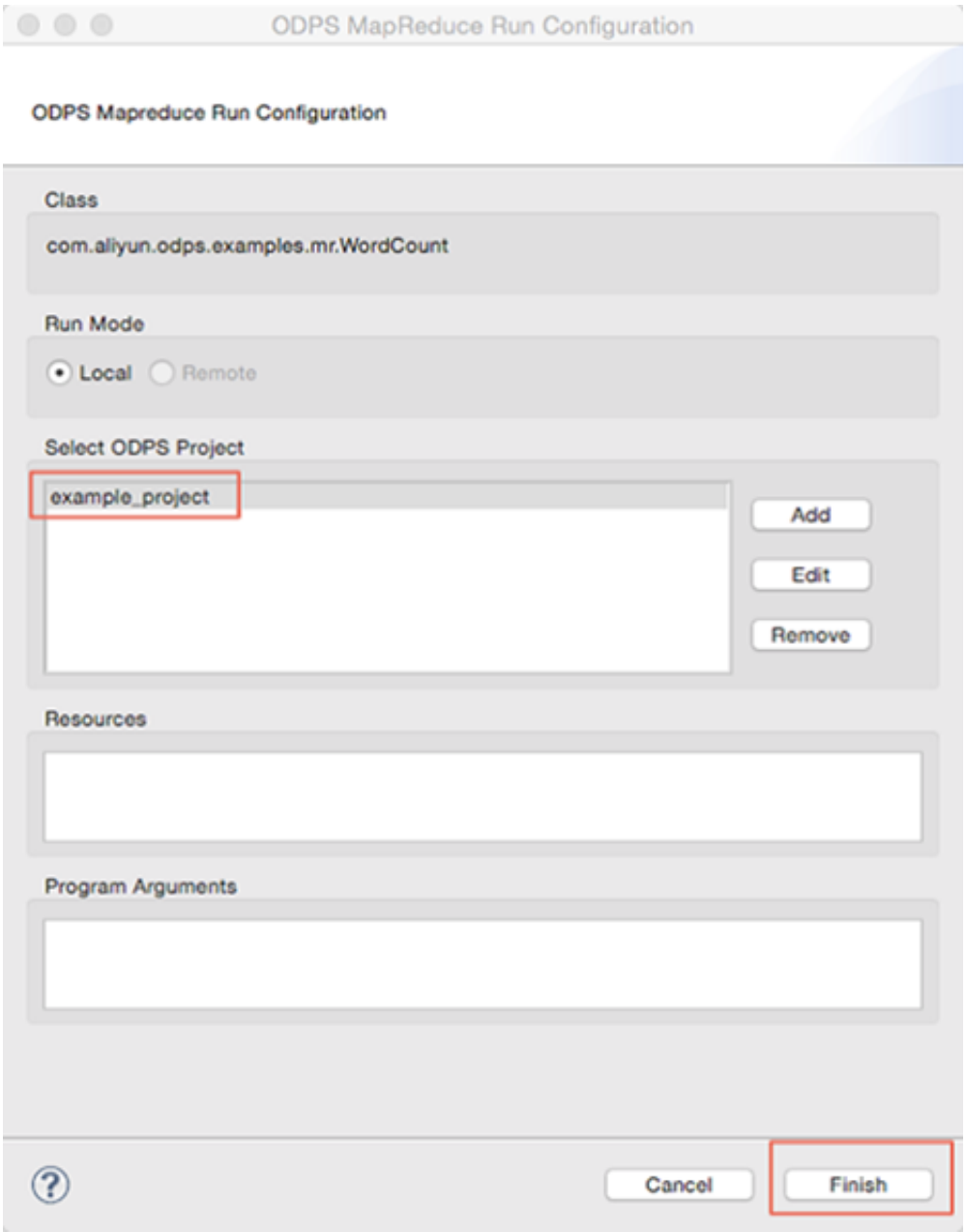
2. 右键单击**WordCount.java**，依次选择**Run AsODPS MapReduce**，如下图所示。

图 3-17: 运行WordCount示例



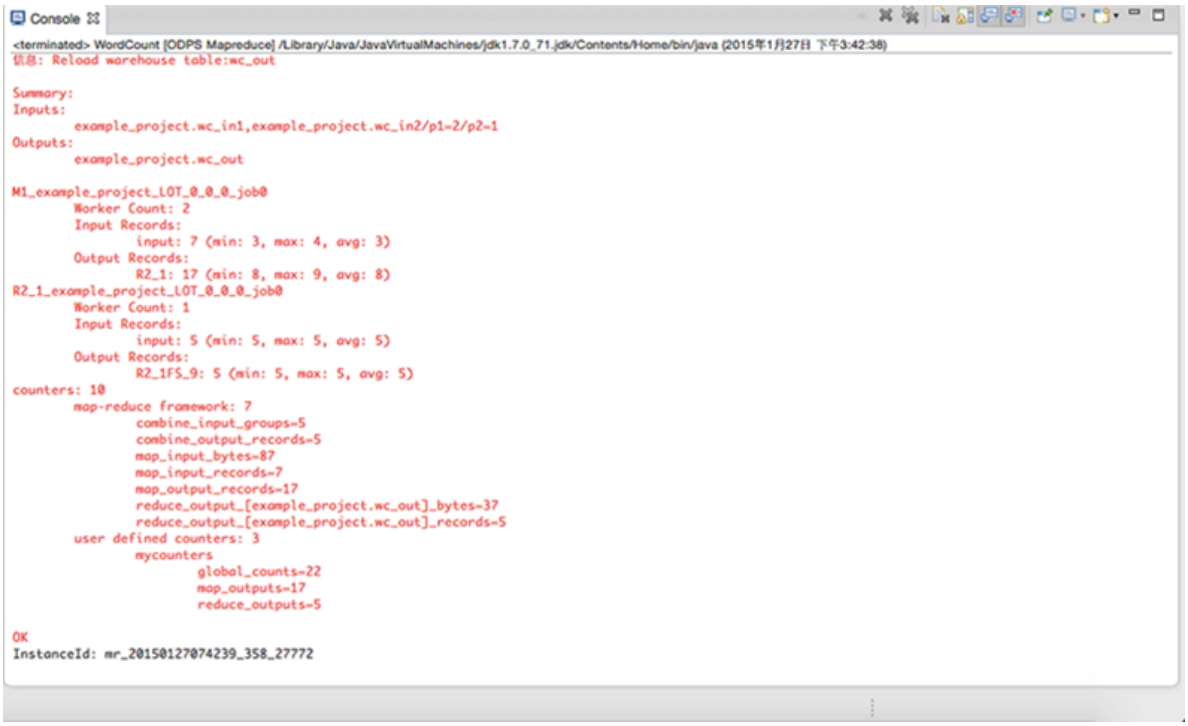
3. 在弹出的对话框中，选择**example_project**，单击**确认**，如下图所示。

图 3-18: 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 3-19: WordCount示例运行结果

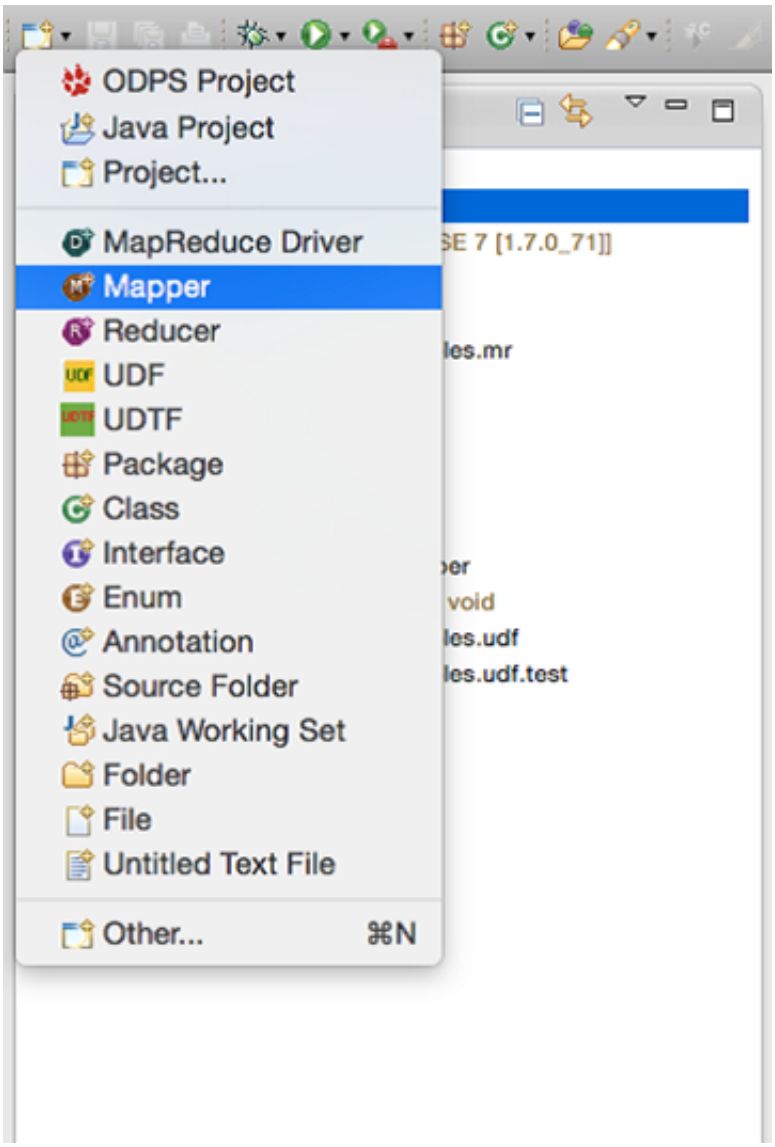


3.14.2.3.2 运行自定义MapReduce程序

操作步骤

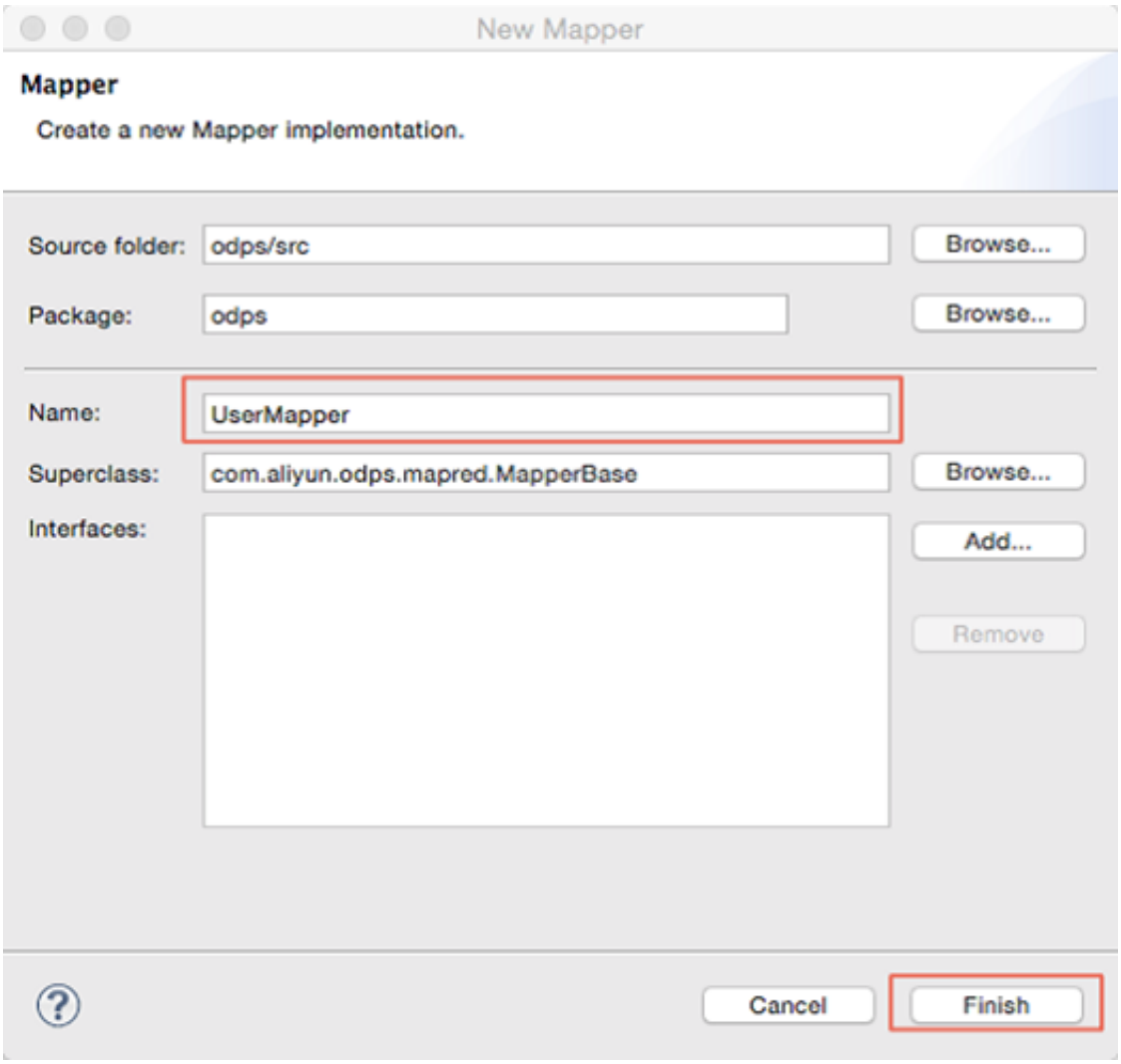
1. 在Eclipse中右键选择**src**目录，选择**New** > **Mapper**，如下图所示。

图 3-20: 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 3-21: 运行自定义MapReduce程序步骤2



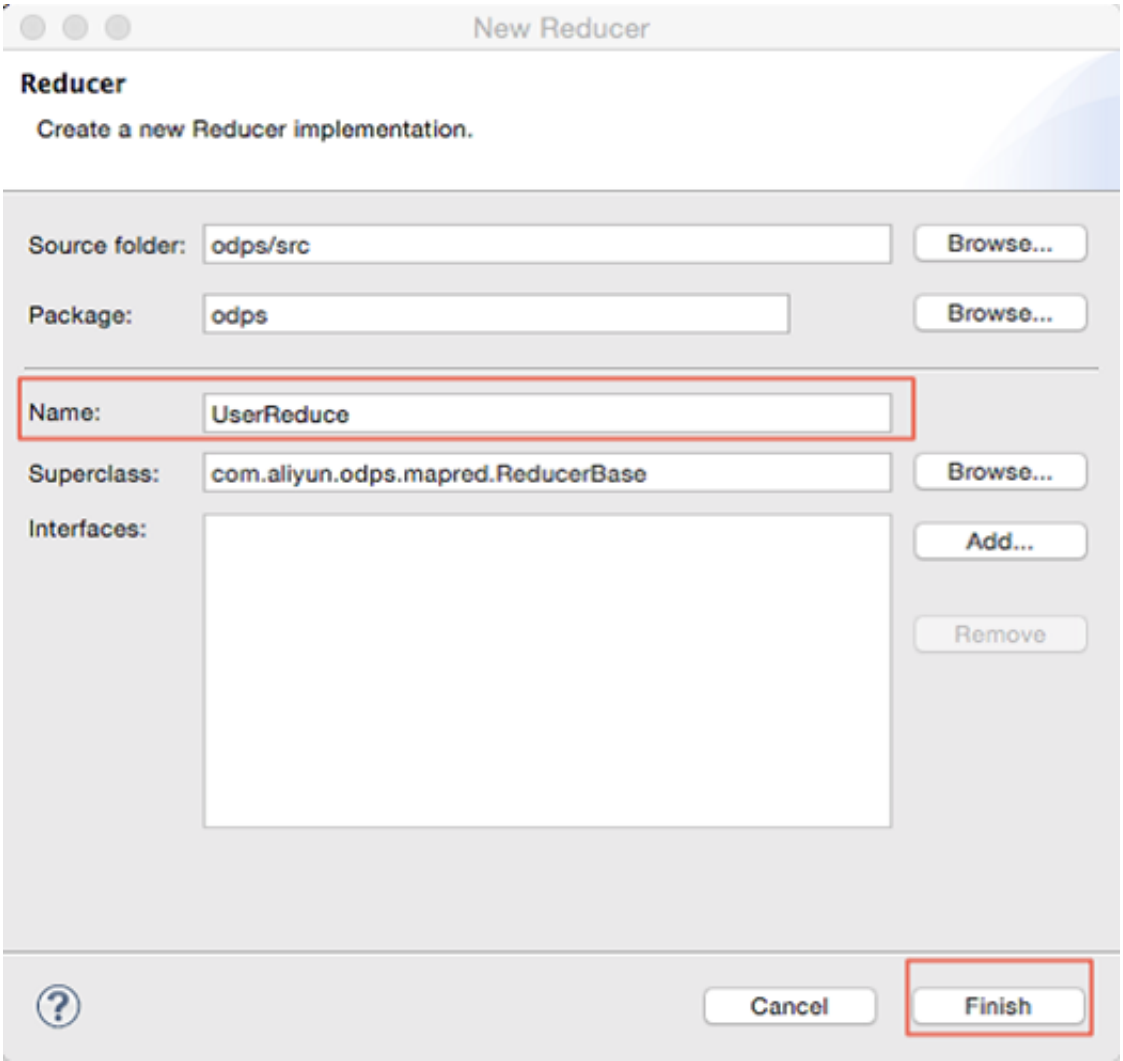
3. 在左侧包资源管理器（Package Explorer）中，src目录下生成文件UserMapper.java。该文件的内容即是一个Mapper类的模板。将模板中package名称默认配置为odps，编写模板内容如下。

```
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.createMapO
utputValueRecord(); 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 3-22: Reducer



5. 在弹出的对话框中输入Reduce类的名字，并确认。

说明：

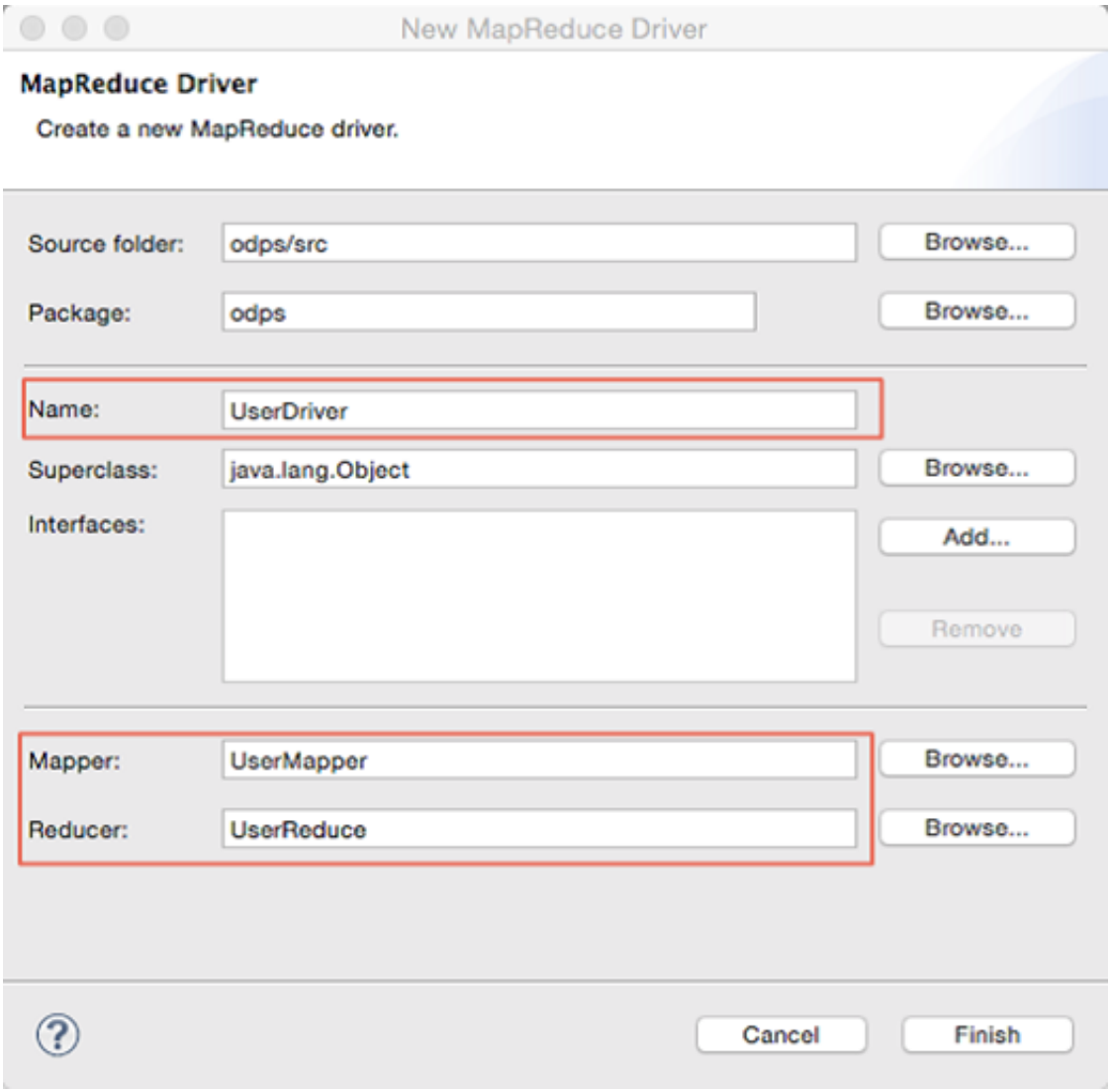
本示例使用UserReducer。

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 3-23: MapReduce Driver



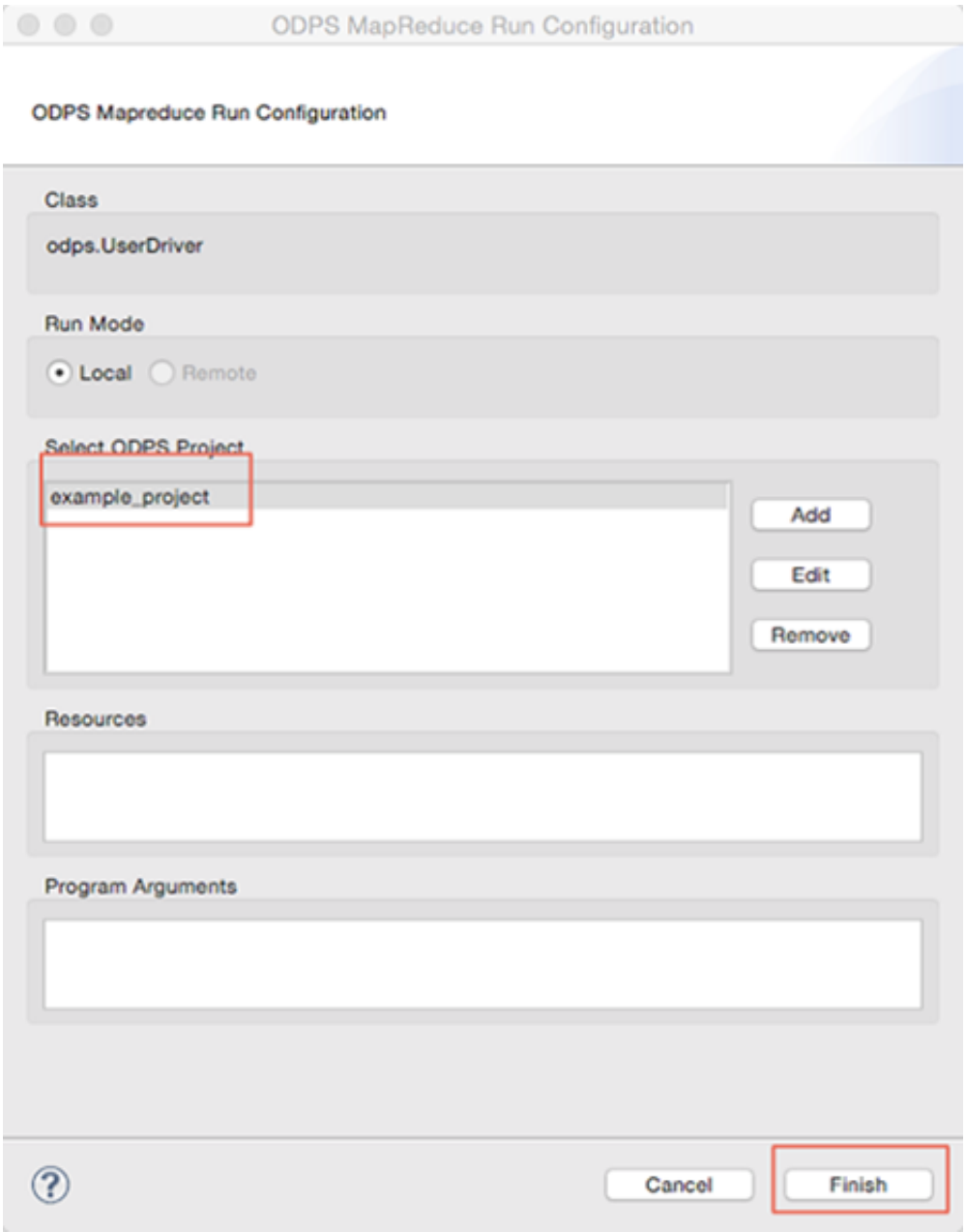
9. 在左侧包资源管理器（Package Explorer）中，src目录下生成文件MyDriver.java。该文件的内容即是一个MapReduce Driver的模板。将模板中package名称默认配置为odps，编写模板内容如下。

```
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();
}
}
```

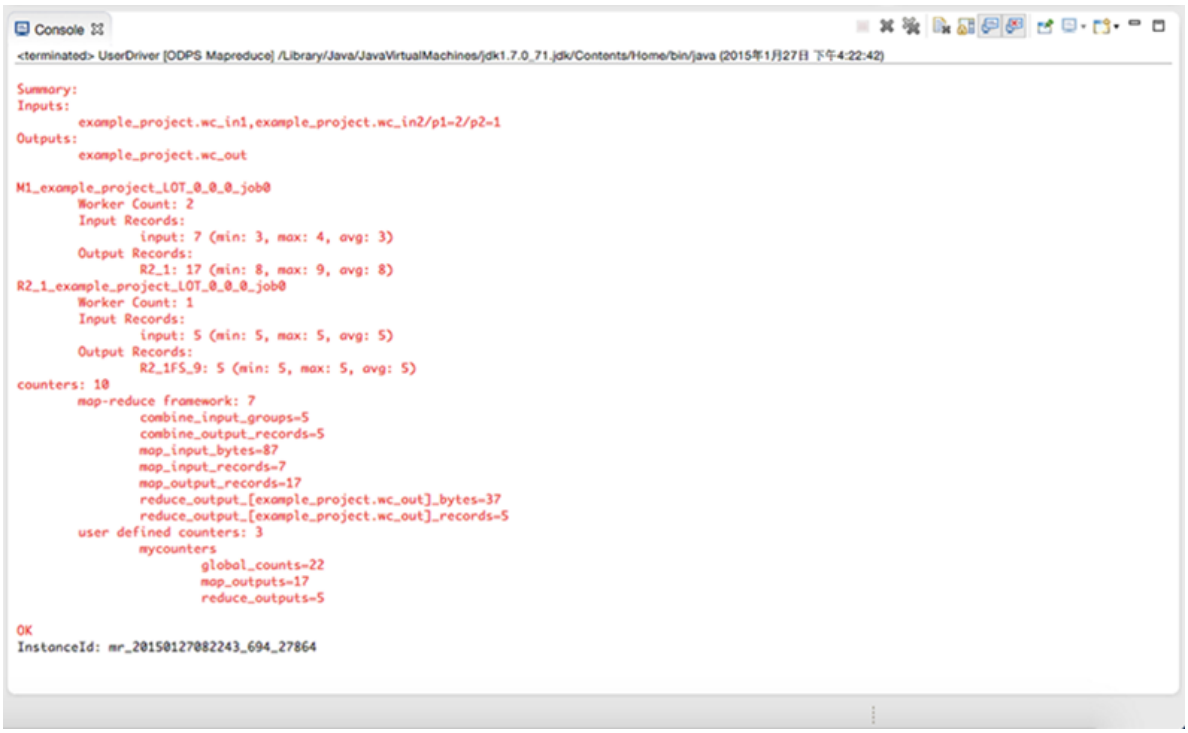
10. 运行MapReduce程序，选中**`UserDriver.java`**，右键选择**Run As** > **ODPS MapReduce** ，单击**确认**。出现如下图所示对话框。

图 3-24: ODPS MapReduce Run Configuration



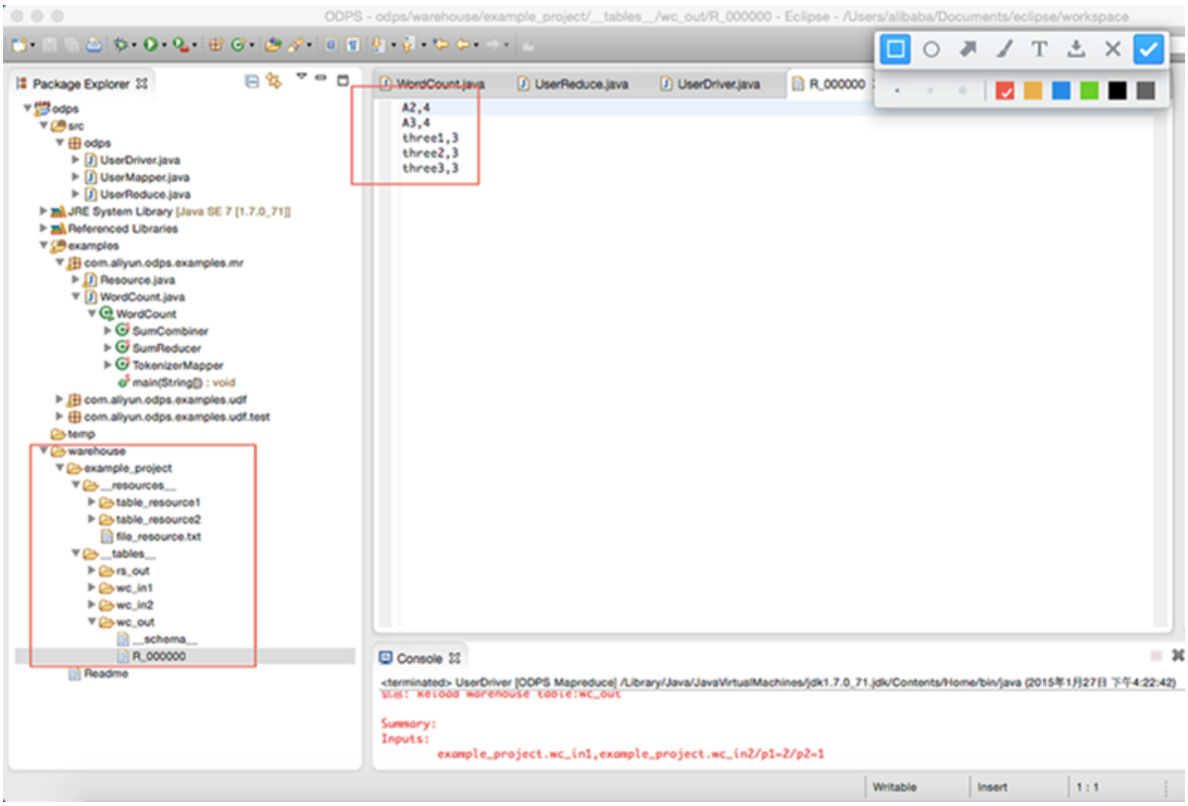
11. 选择ODPS Project为`example_project`，单击**Finish**开始本地运行MapReduce程序，有如下图所示的输出信息，说明本地运行成功。

图 3-25: 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 3-26: 输出结果



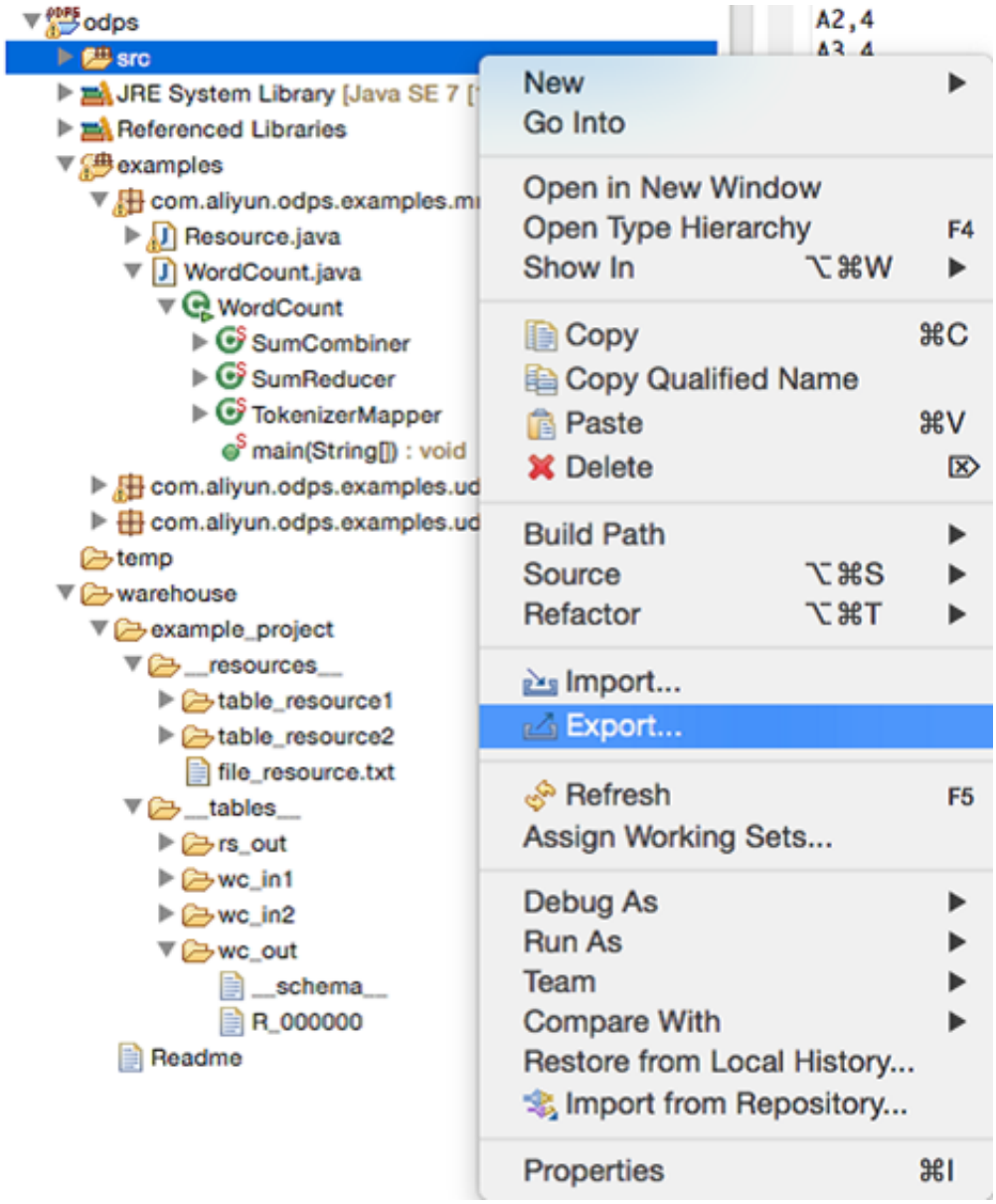
说明：

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。

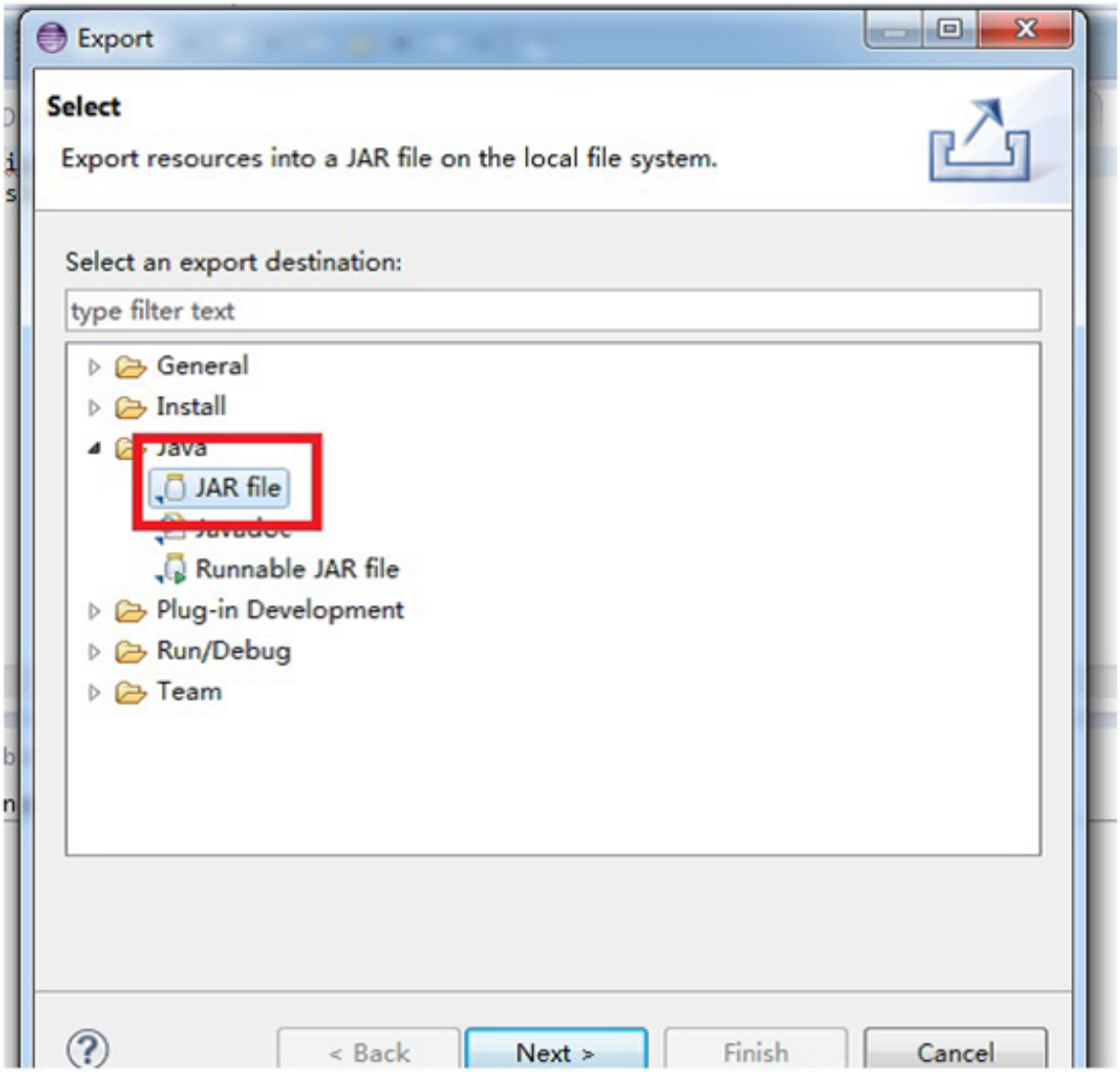
1. 选择**src**目录，单击**Export**，如下图所示。

图 3-27: 导出文件步骤1



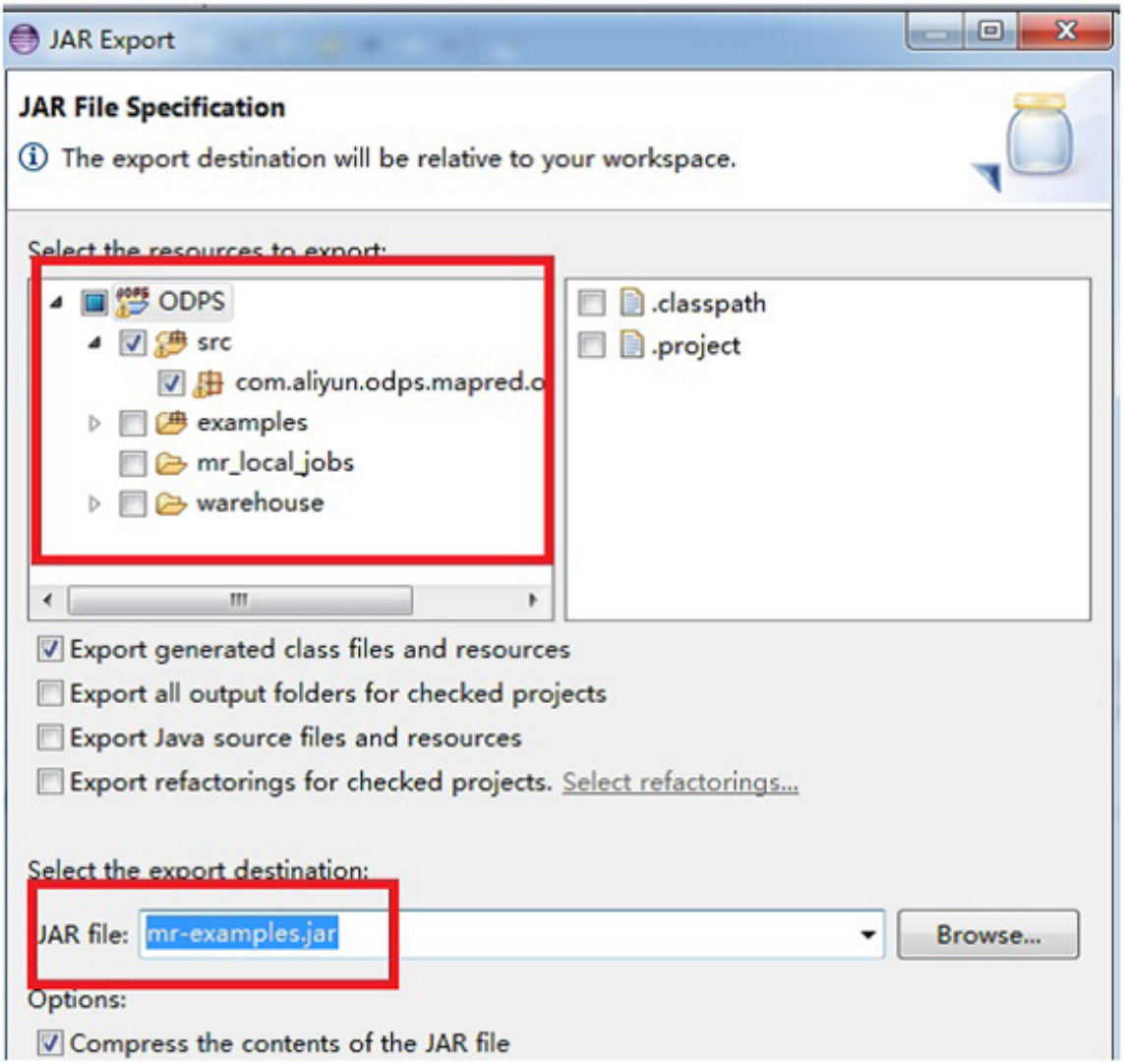
2. 选择导出模式为`Jar File`，如下图所示。

图 3-28: 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 3-29: 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。

3.14.2.4 UDF开发插件介绍

3.14.2.4.1 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和UDTF的编写执行过程与UDF类似，均可参考UDF的示例完成。

3.14.2.4.2 Local Debug UDF程序

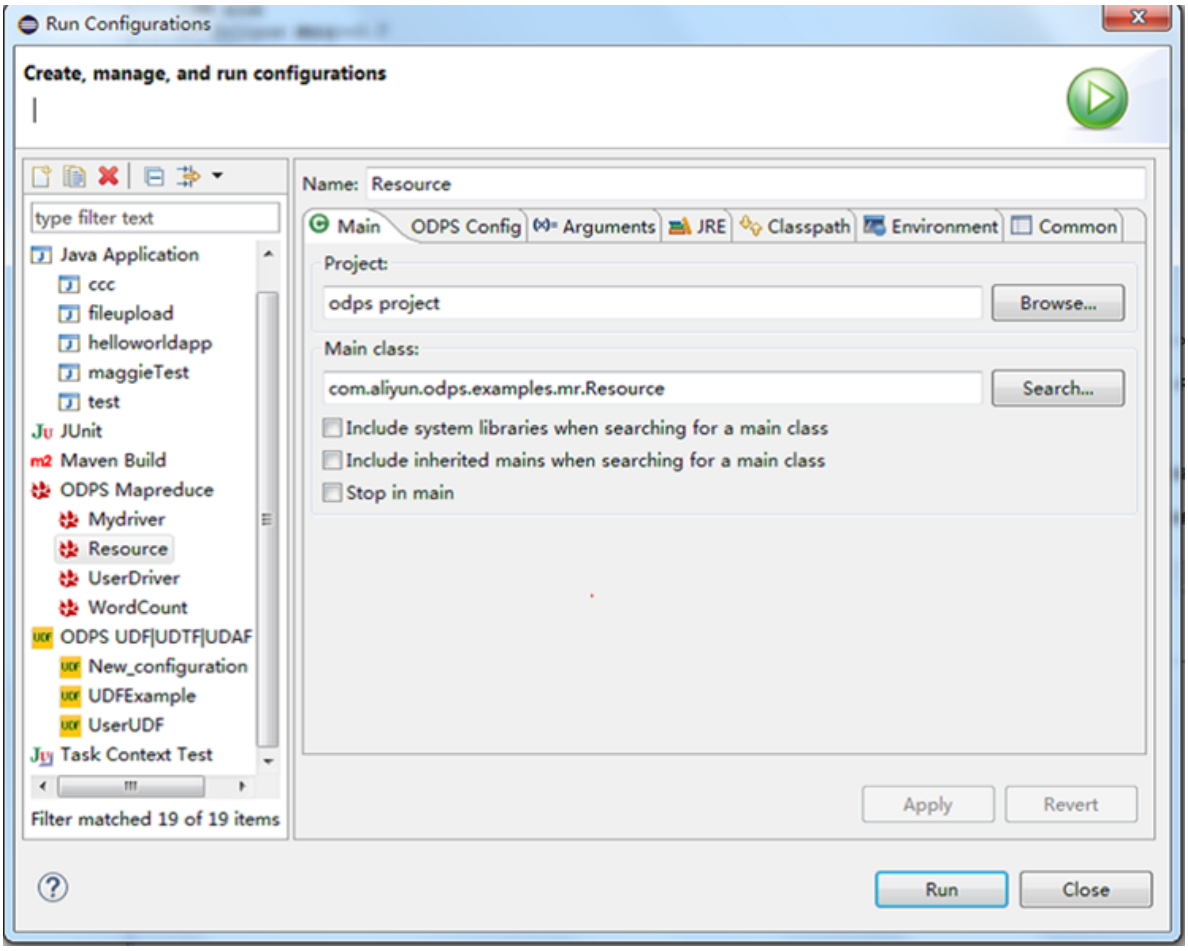
MaxCompute Eclipse插件提供两种运行UDF的方式：菜单栏和右键单击快速运行方式。

3.14.2.4.2.1 菜单栏运行

操作步骤

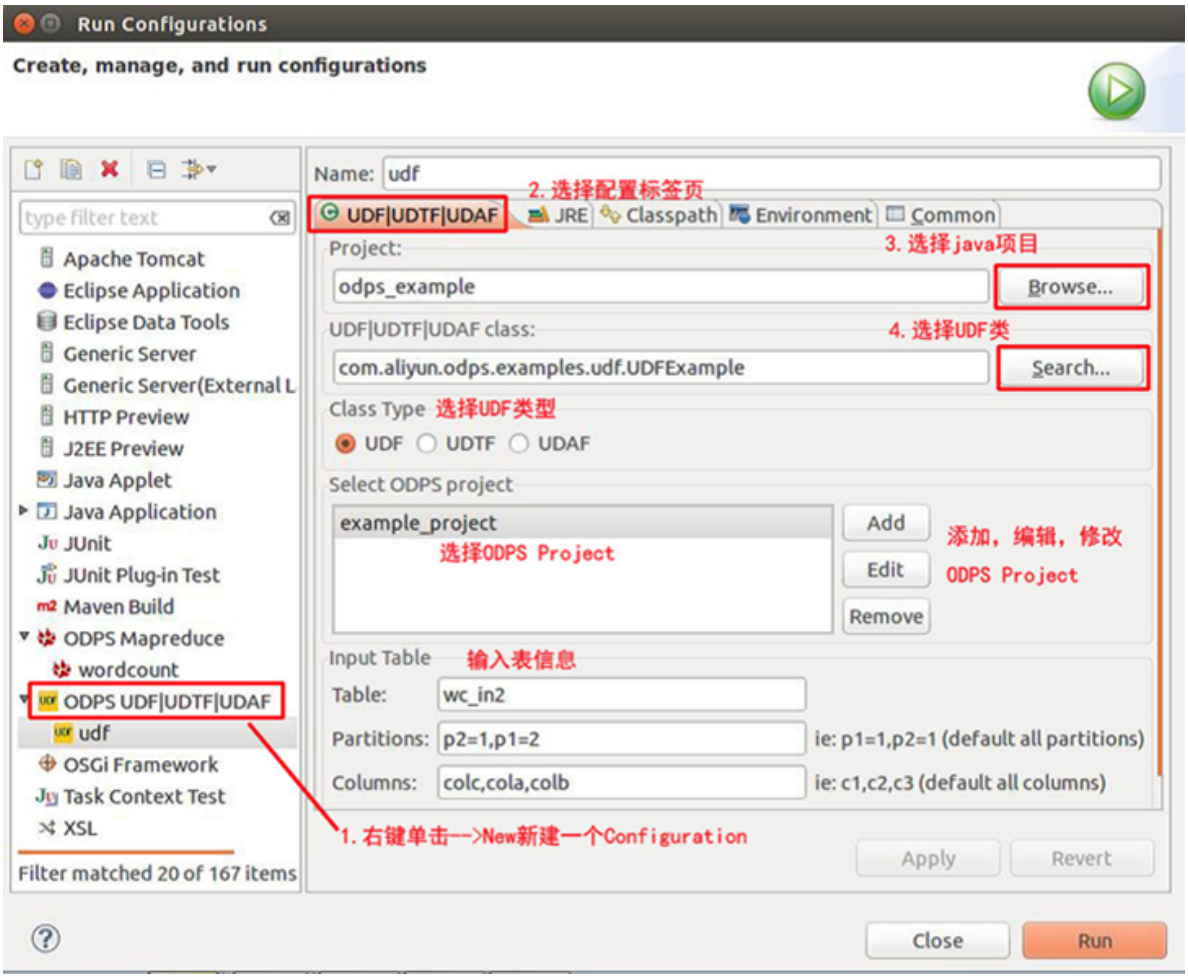
1. 从菜单栏选择**Run** > **Run Configurations**，弹出如下图所示的对话框。

图 3-30: 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 3-31: 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。

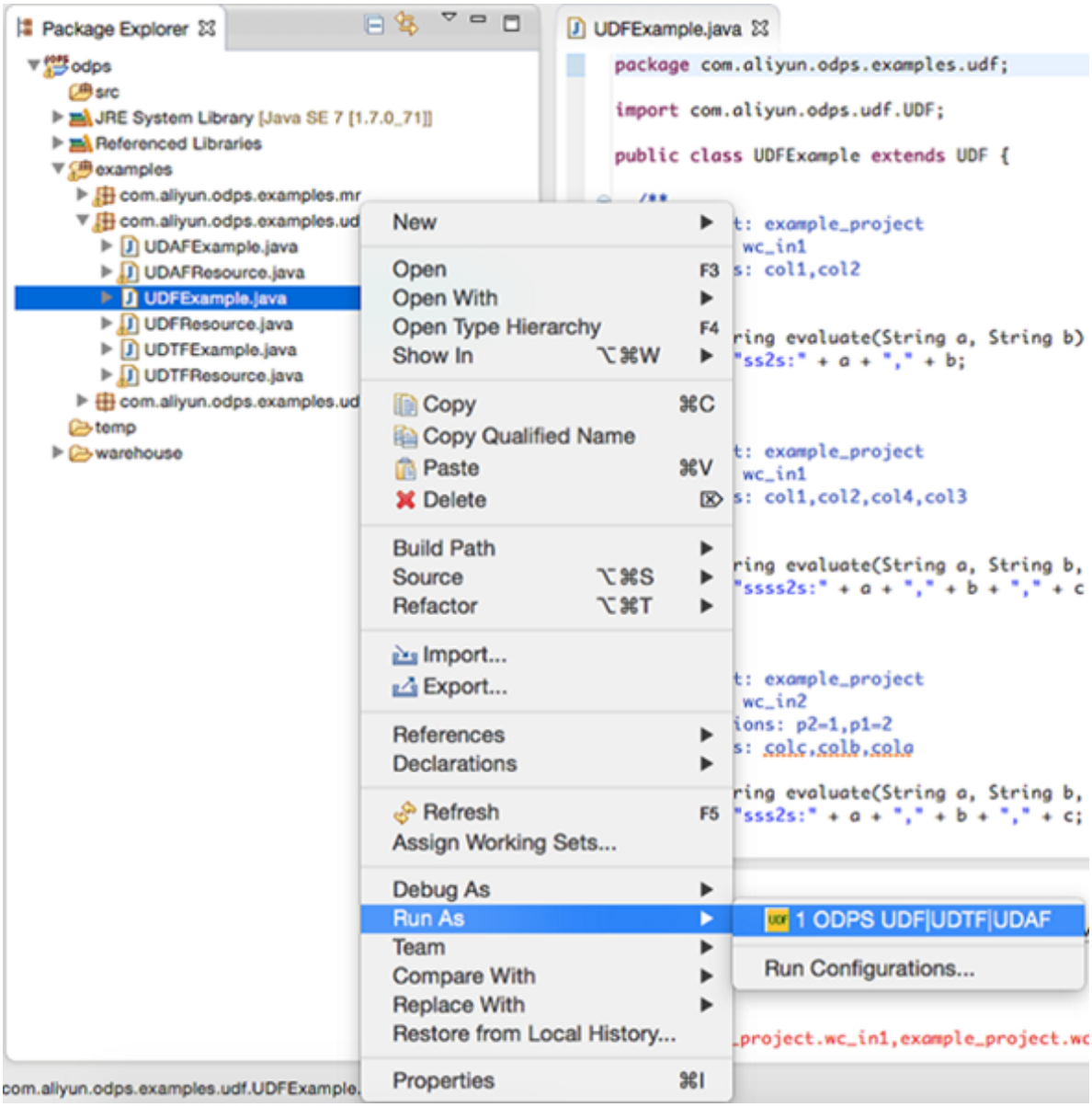
图 3-32: console

3.14.2.4.2.2 右键单击快速运行

操作步骤

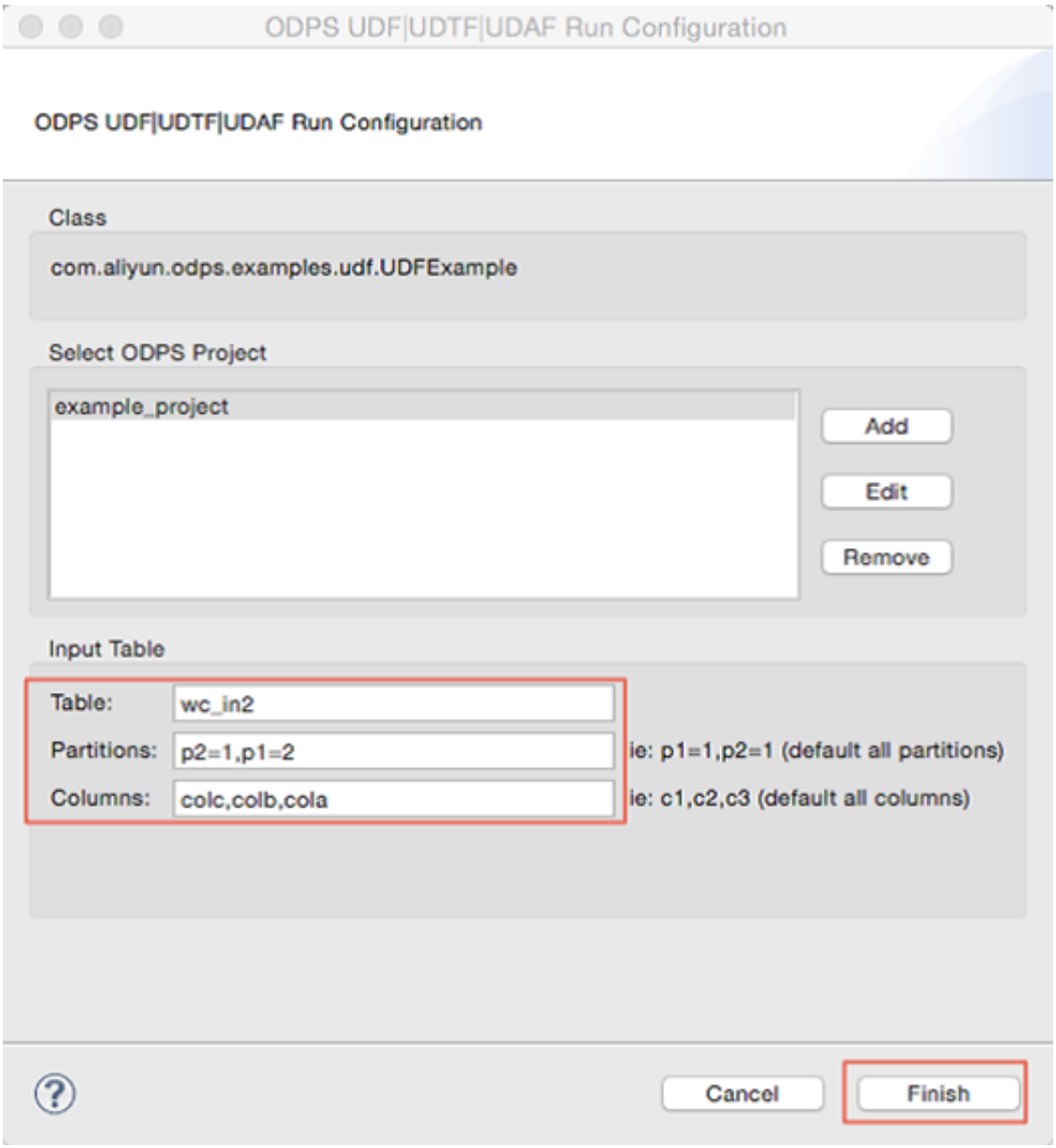
1. 选中一个udf.java文件（例如：UDFExample.java）并单击鼠标右键，选择**Run As** > **Run UDF|UDAF|UDTF**，如下图所示。

图 3-33: 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 3-34: 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。

3.14.2.4.3 运行用户自定义UDF程序

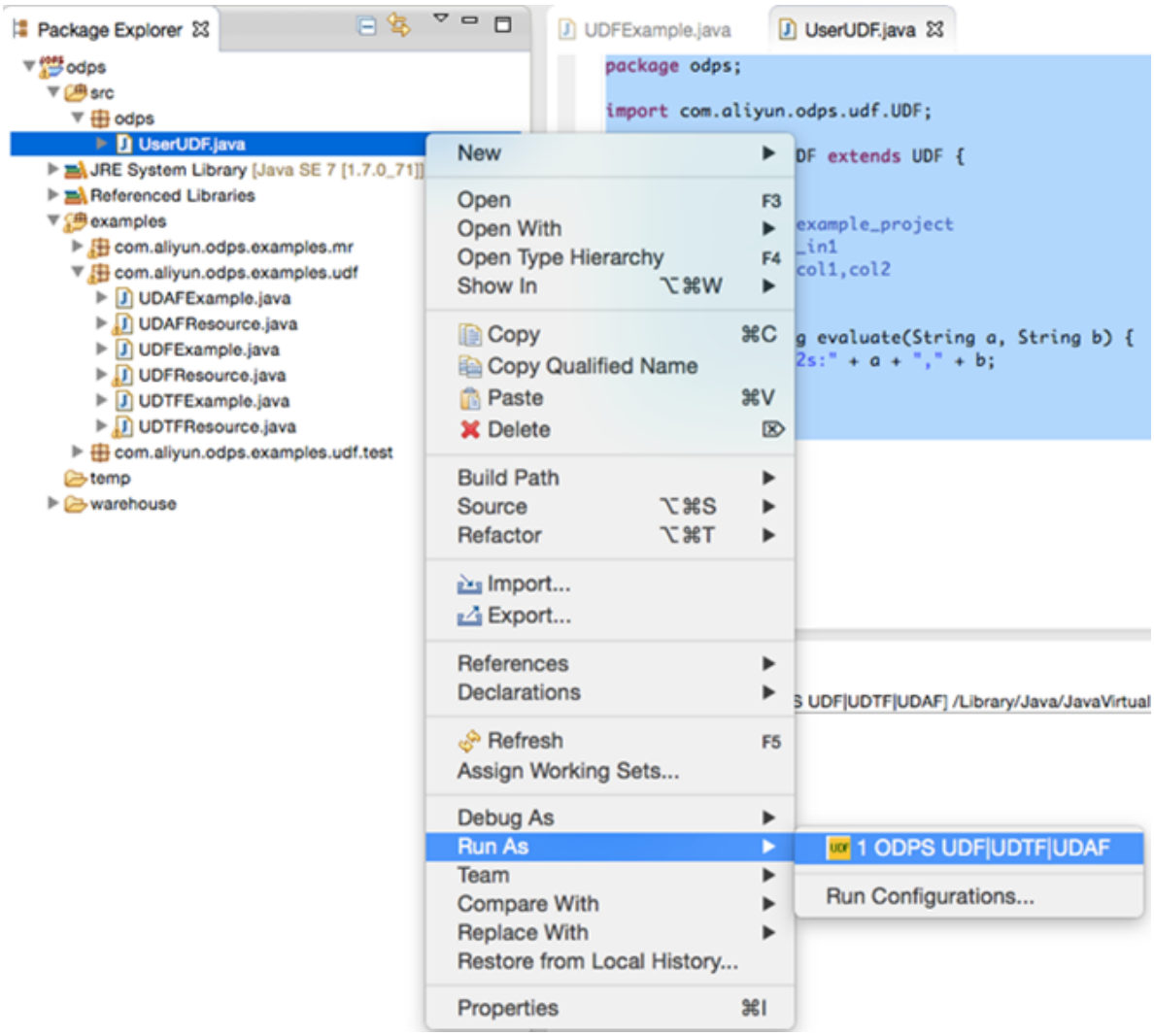
操作步骤

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;
}
}
```

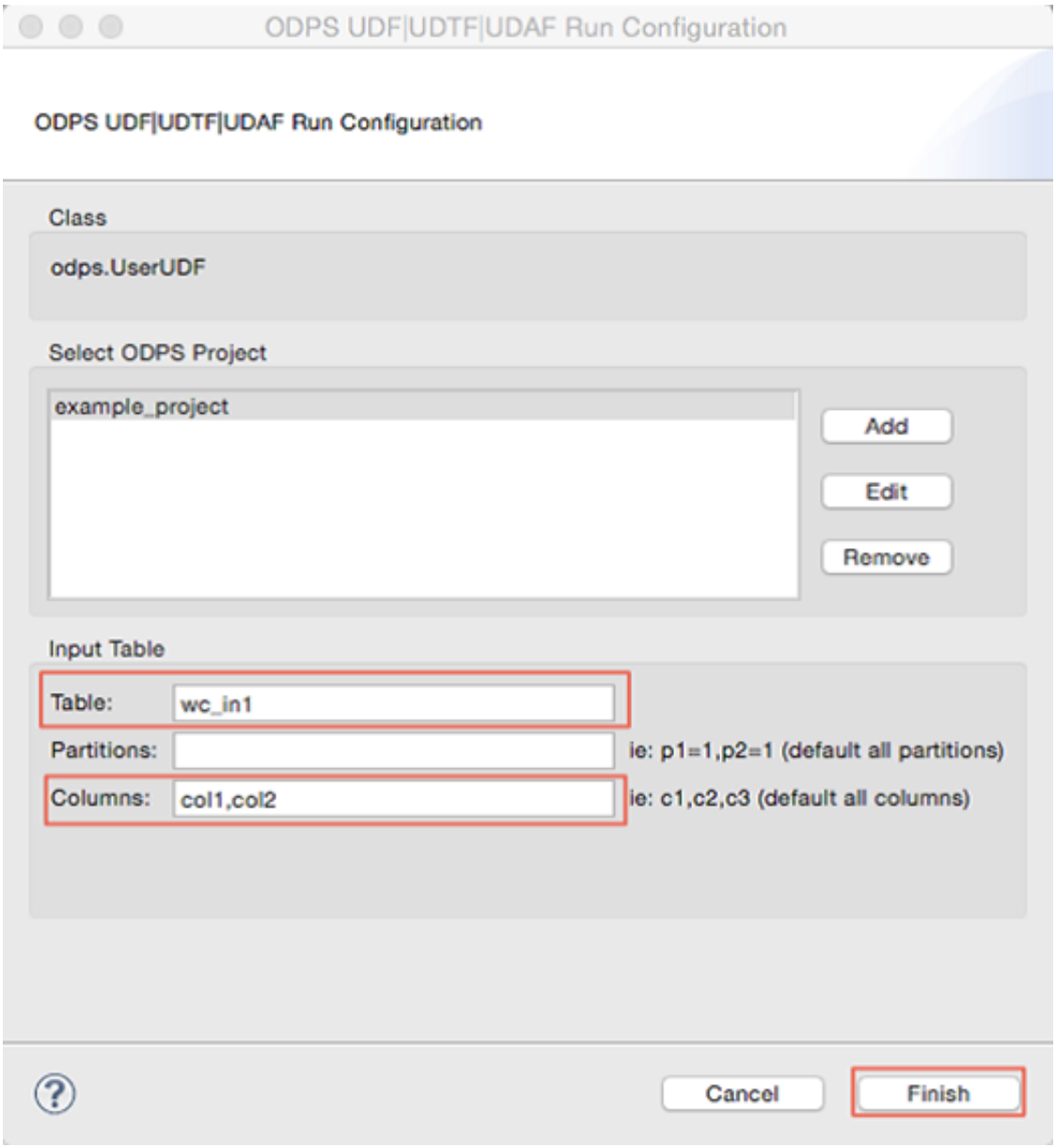
2. 右键选择该java文件（如UserUDF.java），选择**Run As** > **ODPS UDF|UDTF|UDAF**，如下图所示。

图 3-35: 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 3-36: 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。

3.14.2.5 Graph开发插件介绍

背景信息

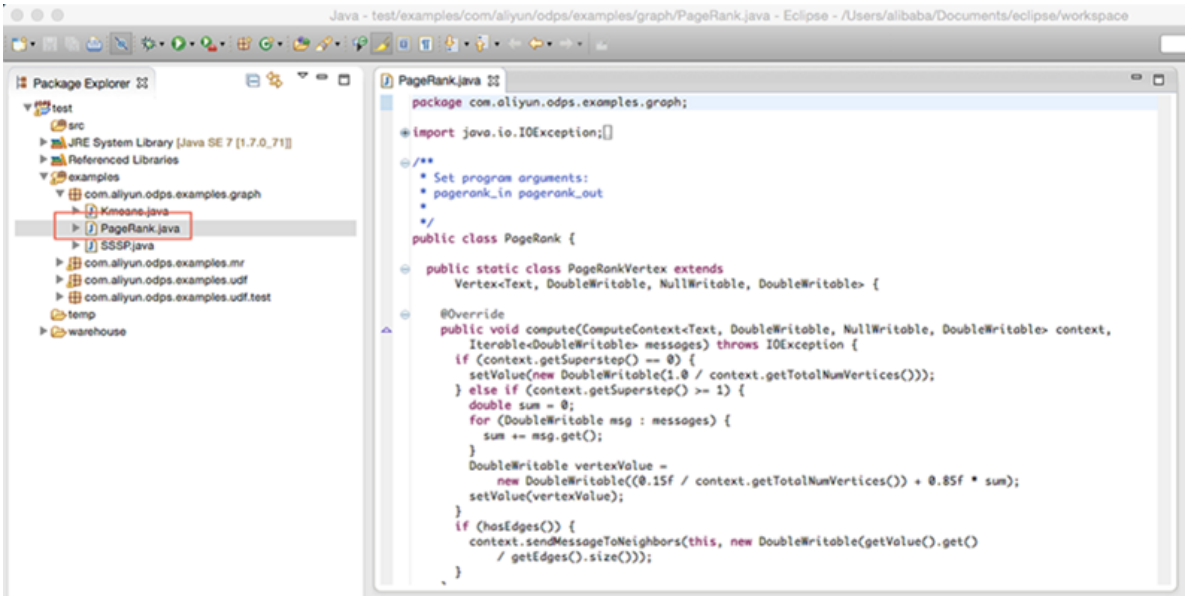
创建MaxCompute项目后，用户可以编写自己的Graph程序，并依照如下操作步骤完成本地调试。

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

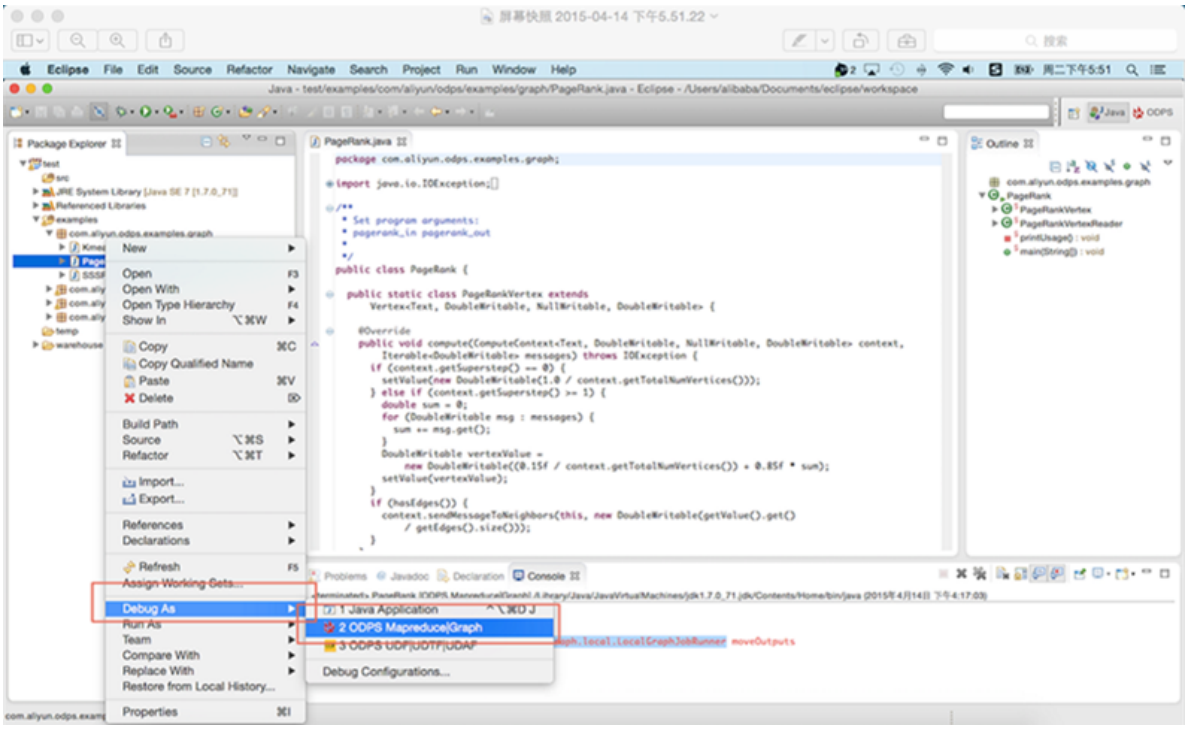
1. 选择**examples** > **PageRank.java**，如下图所示。

图 3-37: PageRank.java 代码图1



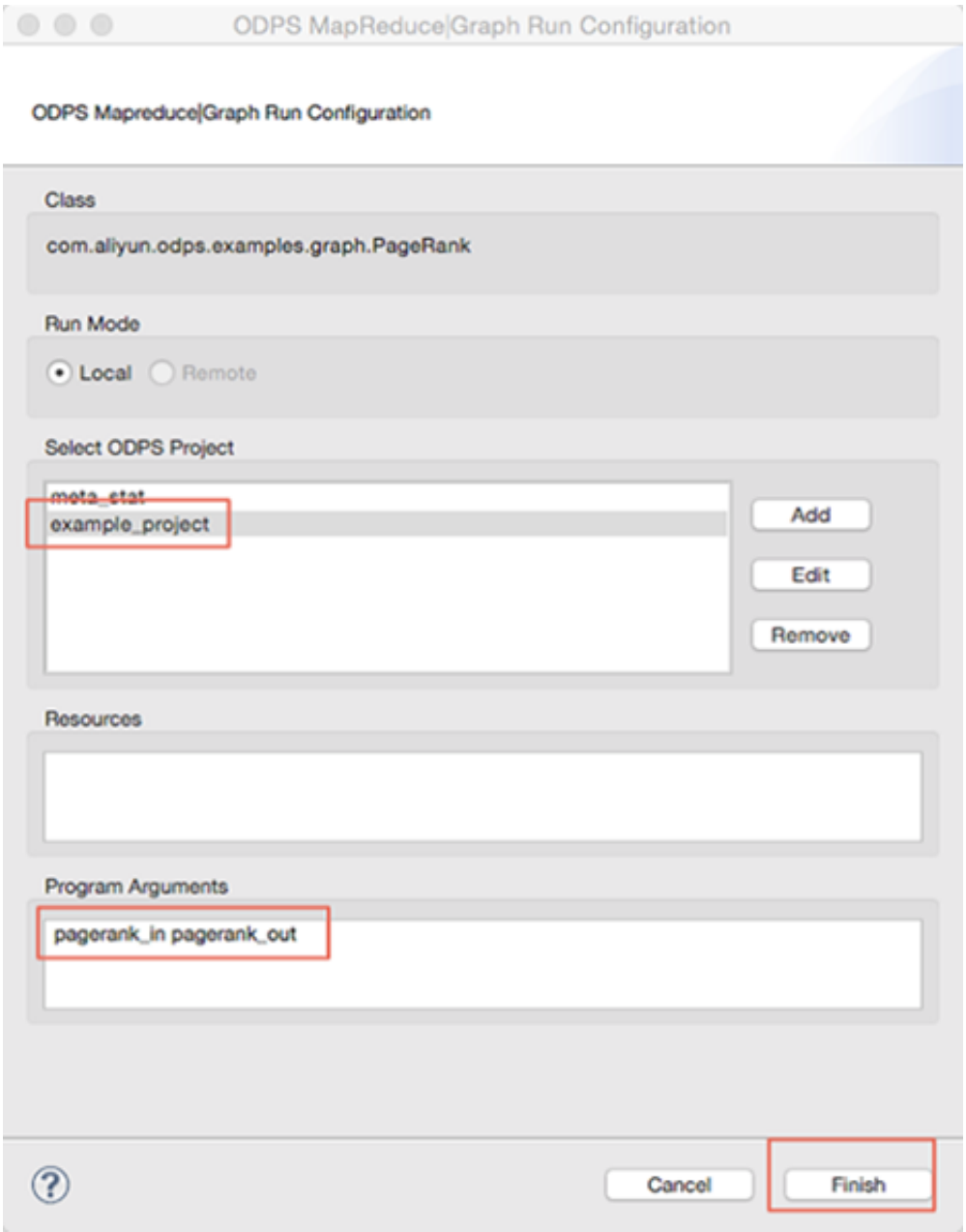
2. 右键单击，选择**Debug As** > **ODPS MapReduce|Graph**，如下图所示。

图 3-38: PageRank.java 代码图2



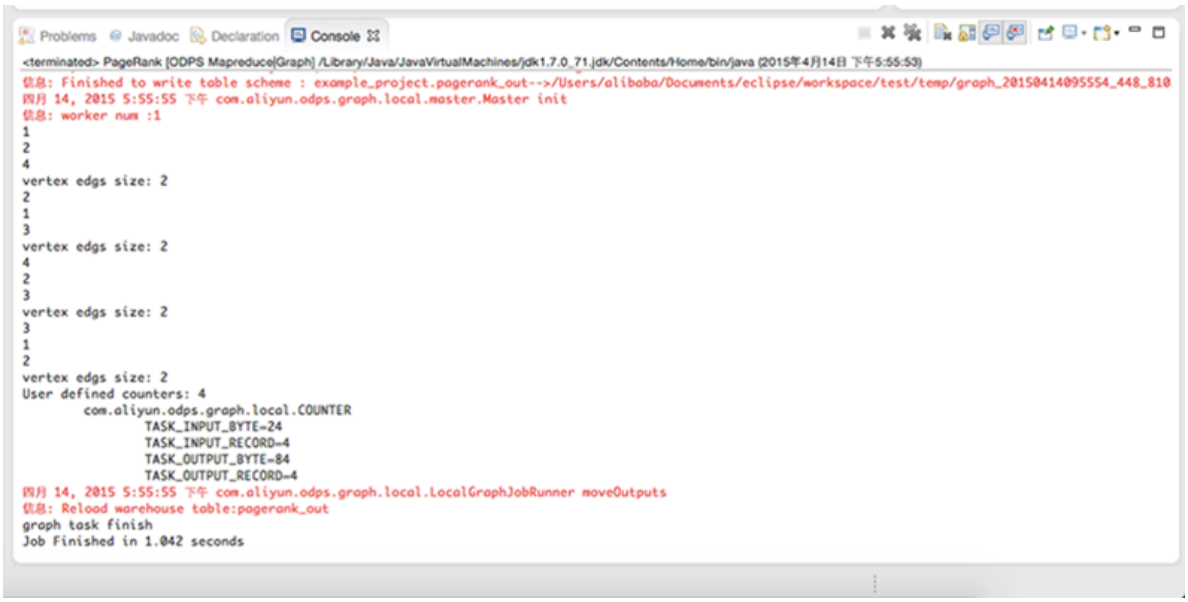
3. 在弹出的对话框中，按照如下图所示的信息进行配置。

图 3-39: 配置图



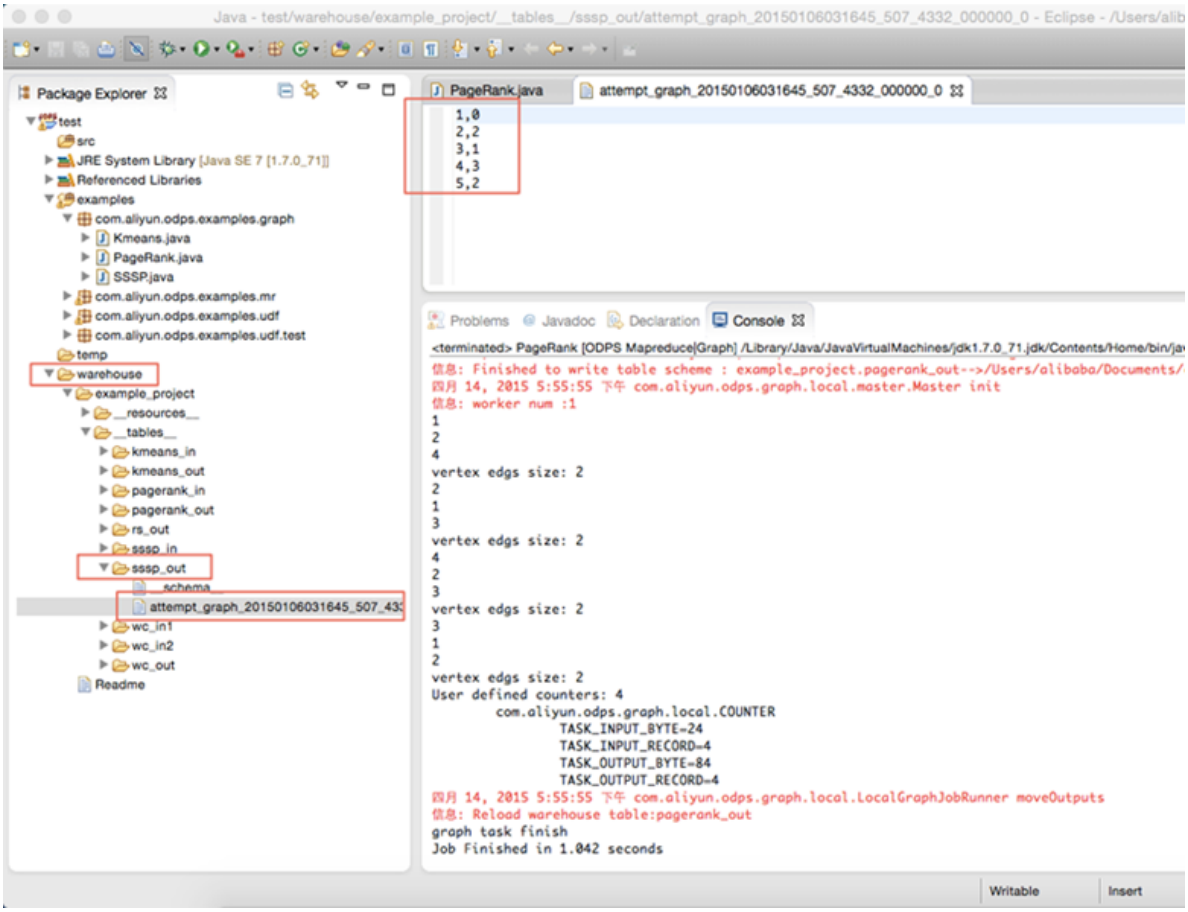
4. 单击**Finish**，查看作业运行结果，如下图所示。

图 3-40: 运行结果



可以查看在本地的计算结果，如下图所示。

图 3-41: 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：

- 打包过程可参见：*MapReduce开发插件介绍*。
- 有关本地结果的目录结构介绍，也可以参见：*MapReduce开发插件介绍*。
- 关于上传jar资源的详细介绍，可参见：编写并运行*Graph*中的相关部分。

4 DataWorks

4.1 什么是DataWorks

产品架构

DataWorks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，利用MaxCompute（原ODPS）可处理海量数据，无需关心集群的搭建和运维。

产品架构如图 *4-1:* 阿里云大数据开发平台架构图所示：

图 4-1: 阿里云大数据开发平台架构图

由上图可以看出，DataWorks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。

主要特性

DataWorks引入全新的工作流任务设计理念，特性如下：

- **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。

- **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 4-2: 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。
- **数据分析与处理**：随之可对MaxCompute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。

4.2 准备工作

背景信息

登录DataWorks集群的控制台前，您需要为DataWorks集群创建云账号、租户和计算引擎，您可以选择一个云账号作为租户管理员，其他云账号可以添加到租户中。

操作步骤

登录大数据管家，创建一个云账号、租户和计算引擎。

1. 登录大数据管家。
2. 创建云账号。
   a) 在菜单栏中单击**管理** > **云账号管理**，进入**云账号管理**界面。
   b) 单击**新增云账号**，弹出**创建云账号**对话框，如图 *4-3:* 创建云账号所示。

图 4-3: 创建云账号

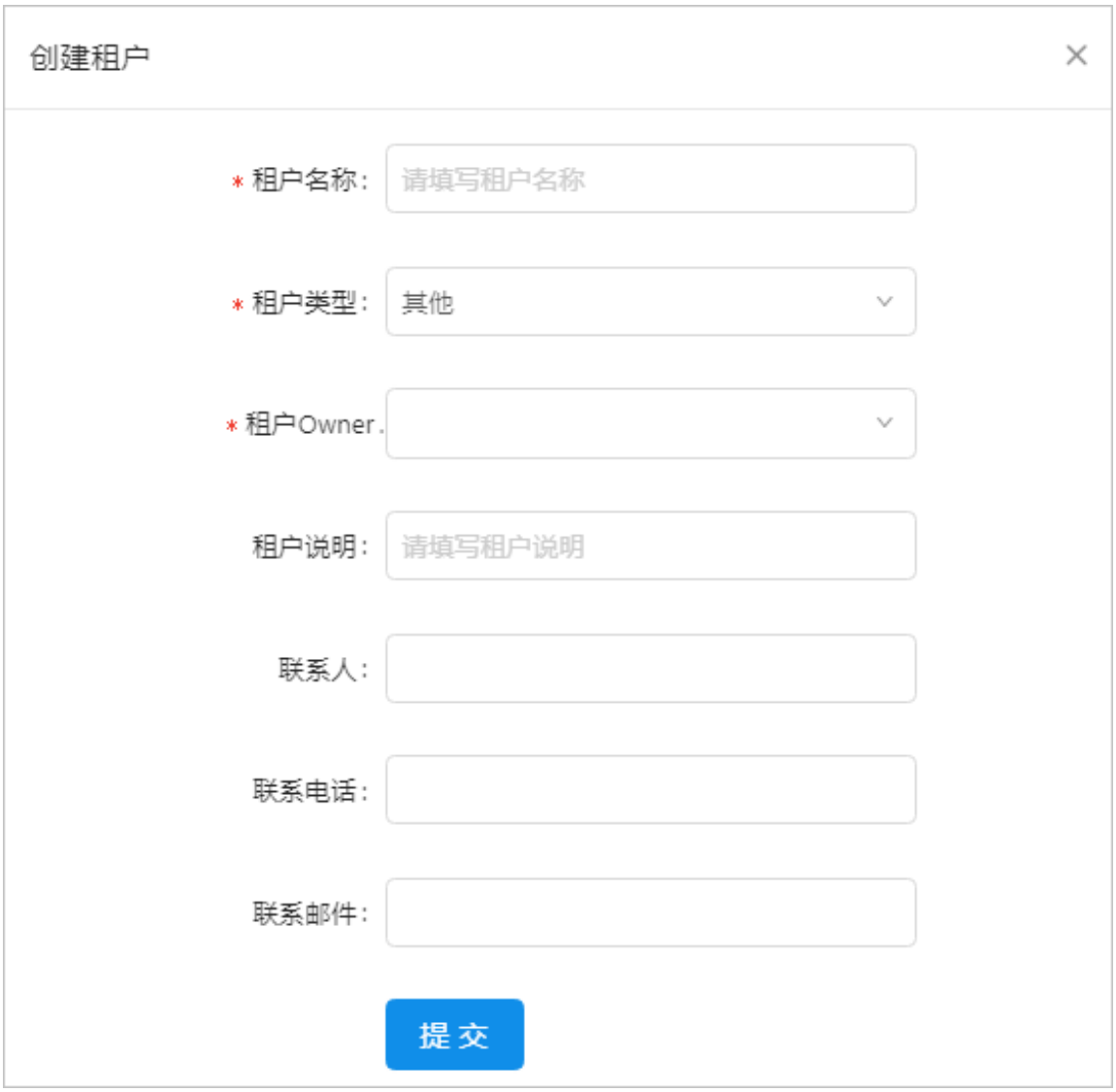
c) 设置各参数后，单击**创建**，创建一个云账号。

3. 创建租户。

a) 在菜单栏中单击**管理** > **租户管理**，进入**租户管理**界面。

b) 单击**新增租户**，弹出**创建租户**对话框，如图 *4-4:* 创建租户所示。

图 4-4: 创建租户



租户Owner说明：一个云账号只能作为一个租户Owner，不能跨租户。创建租户时系统会自动匹配无租户的Owner（不匹配系统用户）。

c) 单击**提交**，创建一个租户。创建的租户显示在**租户管理**界面。

d) 在**租户管理**界面中，单击新创建的租户后面的**用户详情**，弹出**可添加用户列表**对话框。

e) 单击用户后面的**增加**，添加用户到租户中。

4. 创建计算引擎。

a) 在菜单栏中单击**管理** > **计算引擎**，进入**计算引擎管理**界面。

b) 单击**新增计算引擎**，弹出**新增计算引擎**对话框，如图 *4-5:* 新增计算引擎所示。

图 4-5: 新增计算引擎

c) 单击**确定**，创建一个计算引擎。

4.3 登录DataWorks控制台

背景信息

登录DataWorks控制台前，需登录天基获取DataWorks集群的域名。

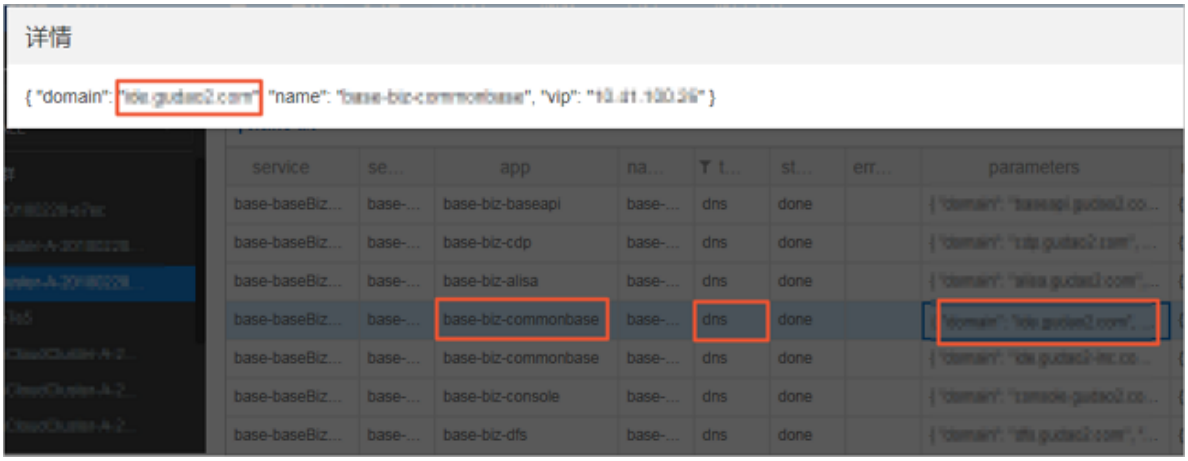
操作步骤

1. 登录天基。
2. 在菜单栏中单击**运维** > **集群运维**，进入**集群运维**界面，查询DataWorks集群，如图 *4-6:* 集群运维界面所示。

图 4-6: 集群运维界面

3. 单击查询到的DataWorks集群，进入**集群Dashboard**页面。
4. 在**集群资源**区域框中选中**app**为**base-biz-commonbase**、**type**为**dns**的行，右键单击该行的**parameters**单元格，选择**显示更多**，弹出**详情**提示框。在提示框中查看该DataWorks集群的域名，如图 *4-7: 查询DataWorks集群的域名*所示。

图 4-7: 查询DataWorks集群的域名



5. 在浏览器地址栏中，输入已获取的DataWorks域名，按下**Enter**键，进入**云资源管控平台**登录界面，如图 *4-8: 登录云资源管控平台*所示。

图 4-8: 登录云资源管控平台

6. 选择**我是主账号**页签，输入DataWorks集群的云账号信息，单击**登录**，登录到DataWorks集群的控制台。
7. 单击页面中的**这里**，进入**组织管理** > **项目管理**页面。

说明：

仅没有项目的账号会出现此页面，如果当前已有项目存在，可跳过创建的步骤，直接进入**数据开发**页面进行操作。

8. 单击**新建项目**，根据自身需求创建项目空间，详情请参见新建项目空间。
9. 项目创建成功后，单击顶部导航栏中的**数据开发**，进入数据开发页面。

图 4-9: 进入数据开发页面

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，您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。

图 4-9: 用户详情页

- 4-标签页：可单击新建按钮进行新建任务、脚本文件、上传资源、新建函数等操作。

图 4-10: 新建标签

- 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 *4-11: 新建采集模型工作流任务*所示。

图 4-11: 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。

4.4 快速入门

4.4.1 新建任务

背景信息

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. 以开发者角色创建项目。
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 *4-13:* 新建任务所示。

图 4-13: 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 *4-14:* 新建周期调度工作流任务弹出框所示。

图 4-14: 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：

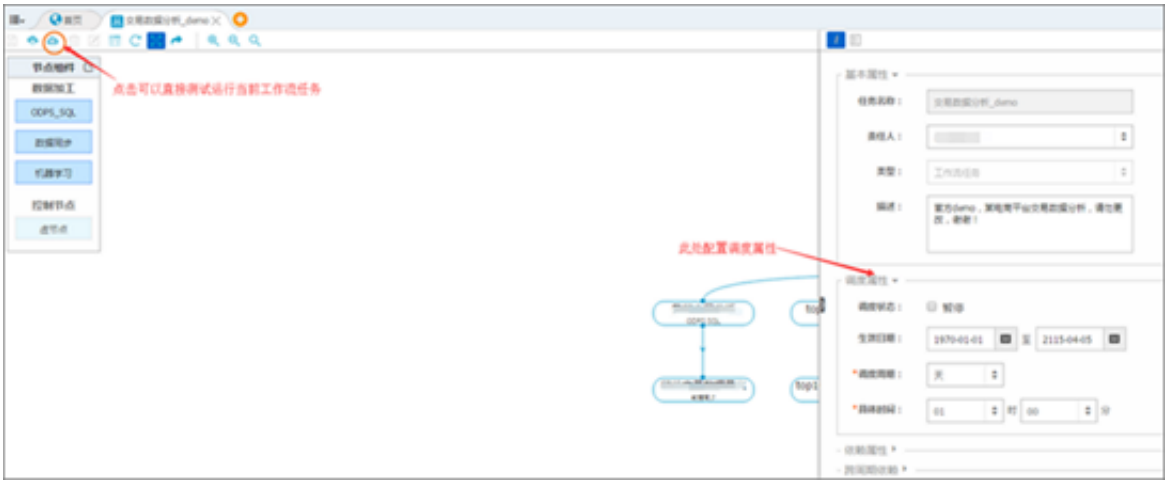
- 名称：支持数字、字母、下划线组合。
- 描述：针对当前工作流任务的简单描述，支持中文、字母、数字、下划线组合。
- 任务类型：支持工作流任务和单节点任务（单节点任务包括SQL、MR、机器学习、Shell、数据同步等类型）。
- 调度类型：调度类型支持一次性调度和周期调度两种类型，且成功创建工作流任务之后无法修改。一次性调度的工作流任务属性和节点属性中不包含调度属性，同时在工作流任务开发面板中可直接运行当前工作流任务。
- 选择目录：选择当前任务所属的目录层级。右键单击任务开发，选择新建文件夹即可创建目录，如新建目录所示。

图 4-15: 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 4-16: 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。

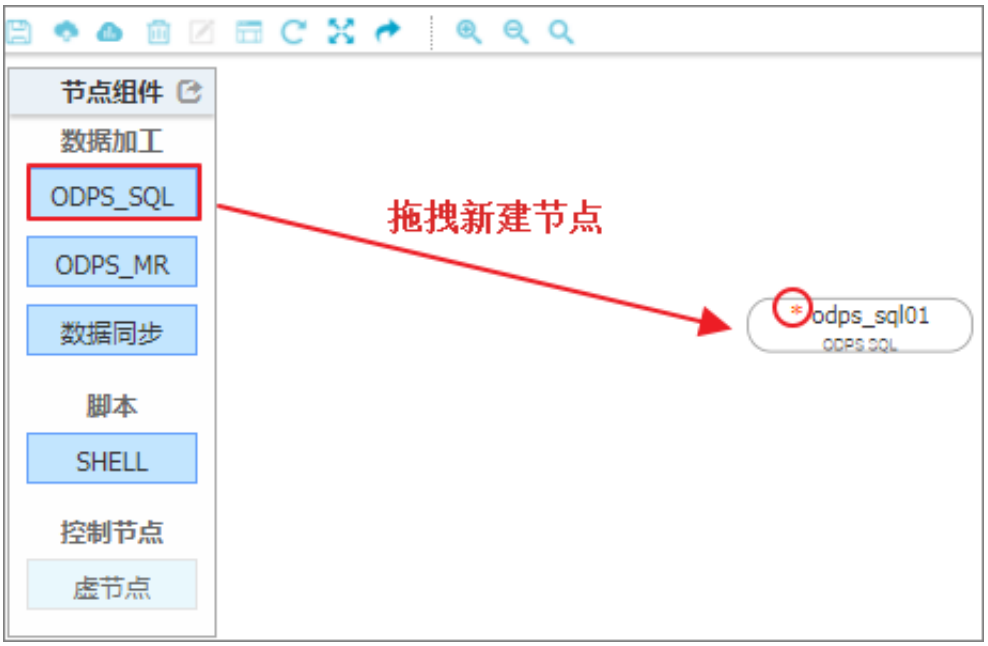
图 4-17: 新建节点弹出框

新建节点弹出框配置项具体说明：

- 名称：支持数字、字母、下划线组合。
- 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 *4-18:* 工作流任务开发面板所示。

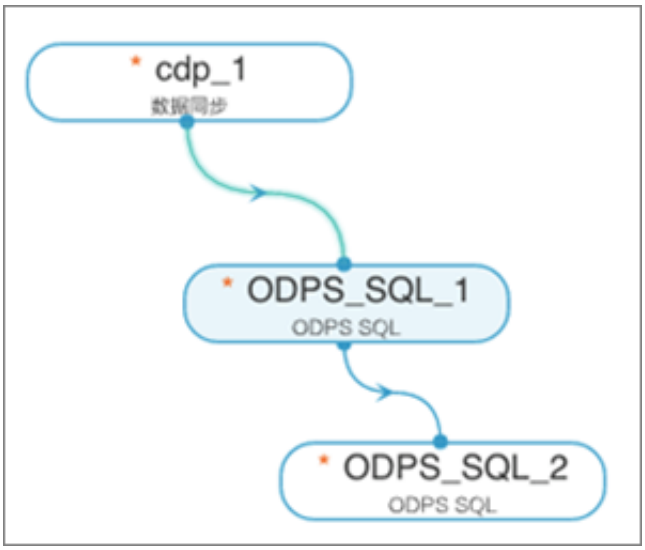
图 4-18: 工作流任务开发面板



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。

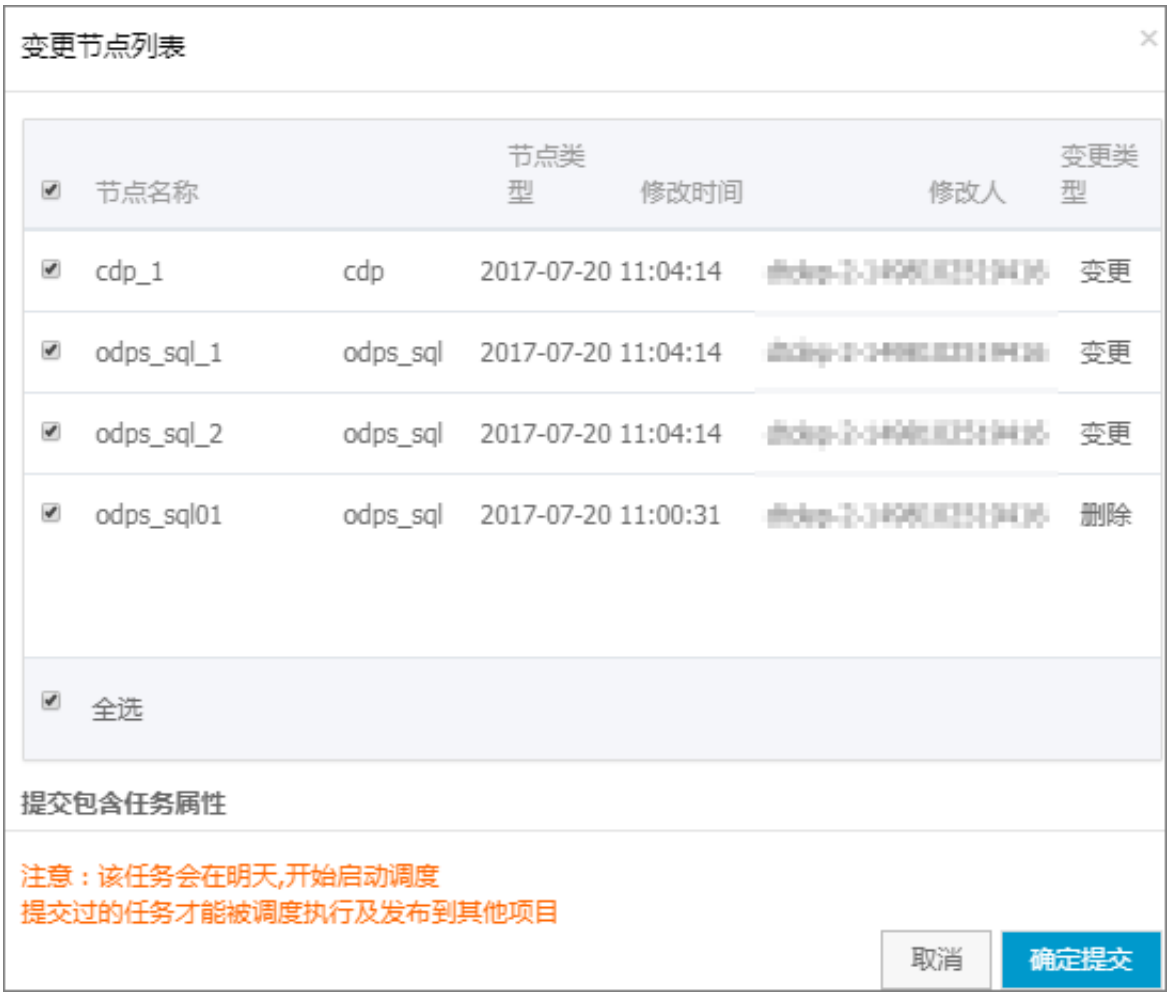
图 4-19: 配置依赖关系



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 4-20: 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。

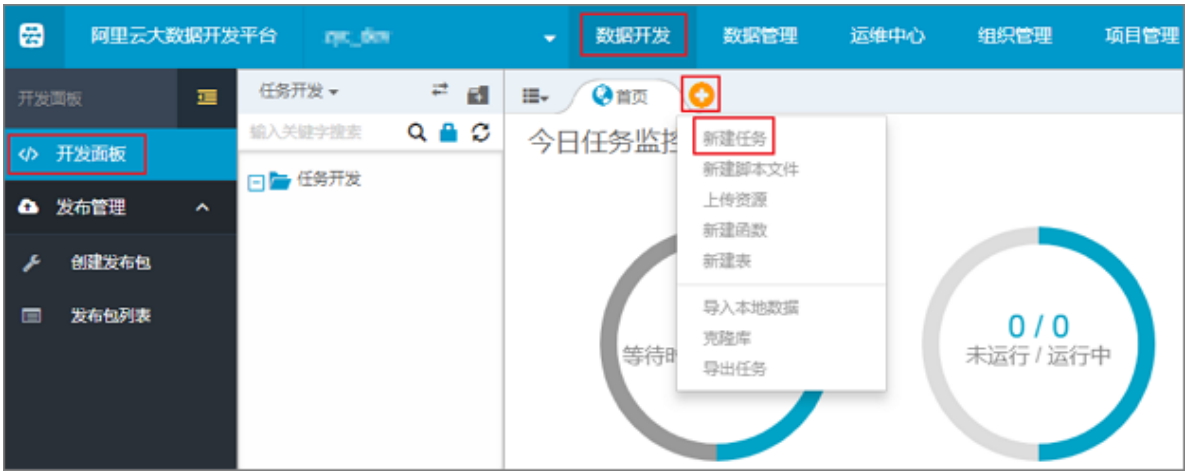
到此，我们已成功新建了一个工作流任务，但是工作流任务属性、节点代码（双击节点进入，包括数据同步和*MaxCompute#原ODPS#代码开发*）、节点属性都尚未配置。具体可根据自身业务场景来进行配置。

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. 以开发者角色创建项目。
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 *4-21: 新建一次性任务*所示。

图 4-21: 新建一次性任务

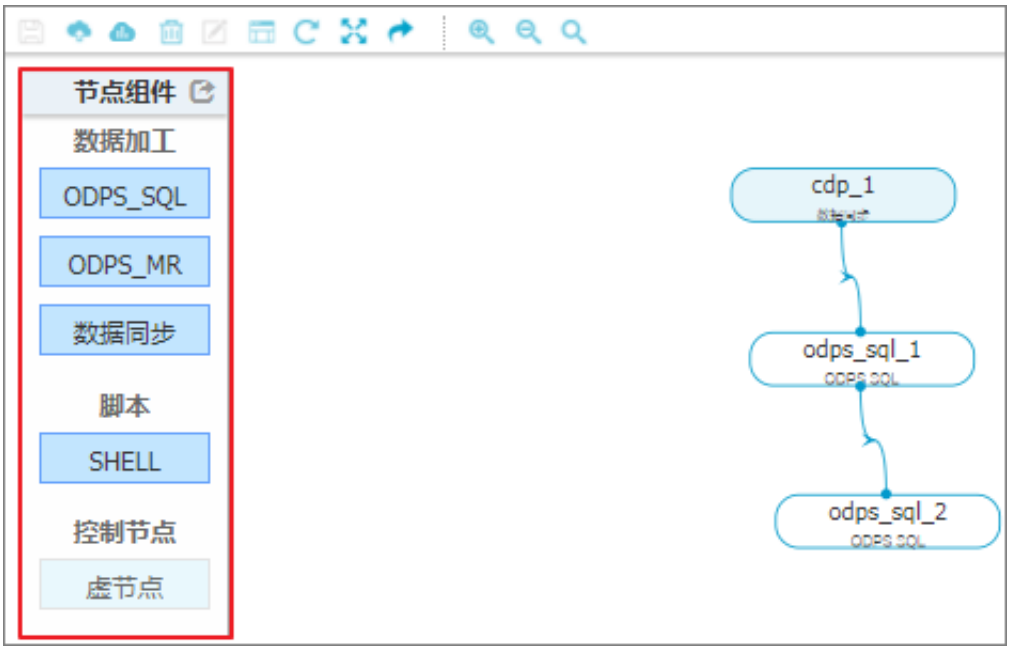


3. 在**新建任务**弹出框中填写各配置项，选择任务类型为**工作流任务**，调度类型为**一次性调度**，如图 *4-22: 新建一次性调度工作流任务弹出框*所示。

图 4-22: 新建一次性调度工作流任务弹出框

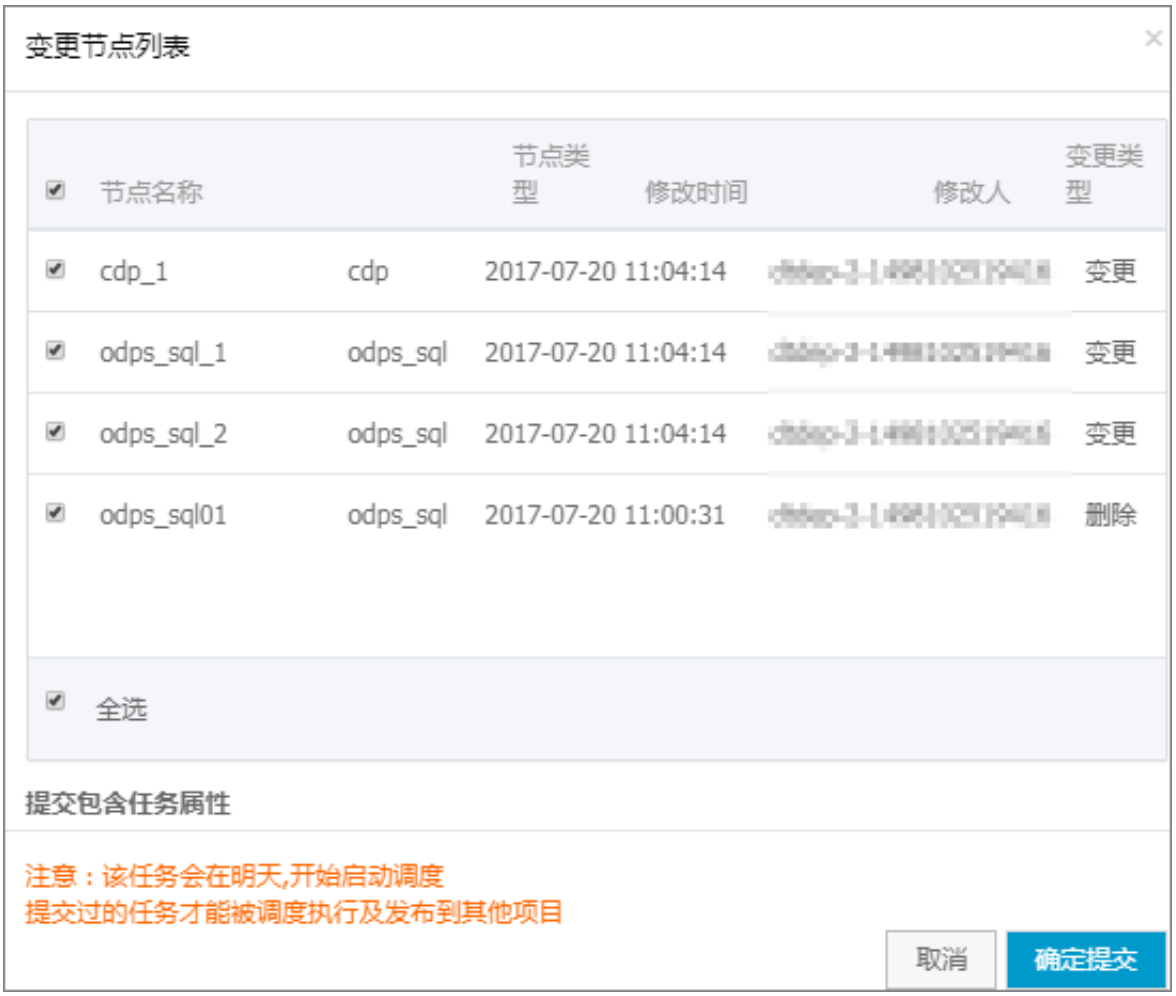
4. 在**节点组件**区域中选中节点拖拽至开发面板画布。

图 4-23: 工作流任务开发面板



5. 单击测试运行按钮 进行测试。

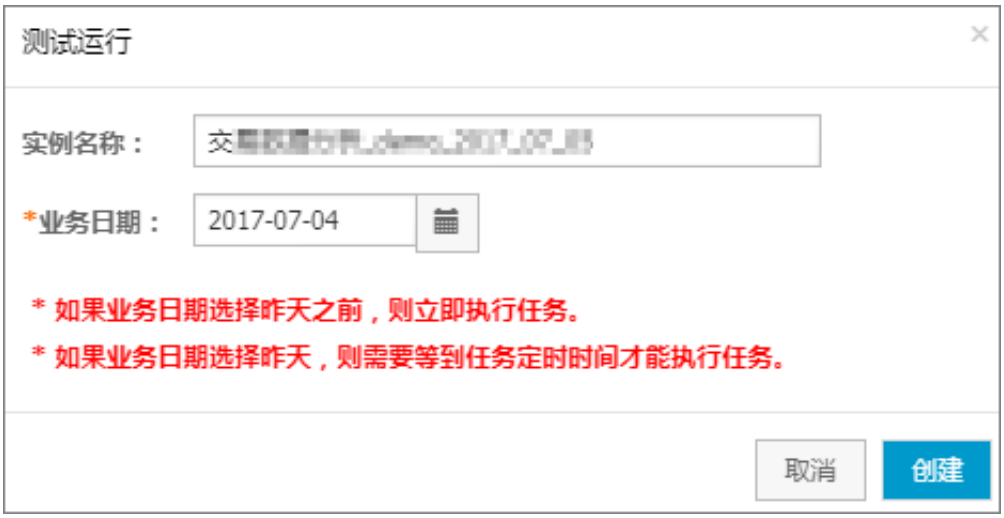
图 4-24: 变更节点列表



若没有提交当前工作流任务，则单击**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 4-25: 测试运行对话框



7. 单击弹出框中的**前往运维中心**，选择需要查看的任务名称，即可进入任务执行页面进行查看。

图 4-26: 工作流任务测试运行

图 4-26: 工作流任务执行页面

4.4.2 测试任务

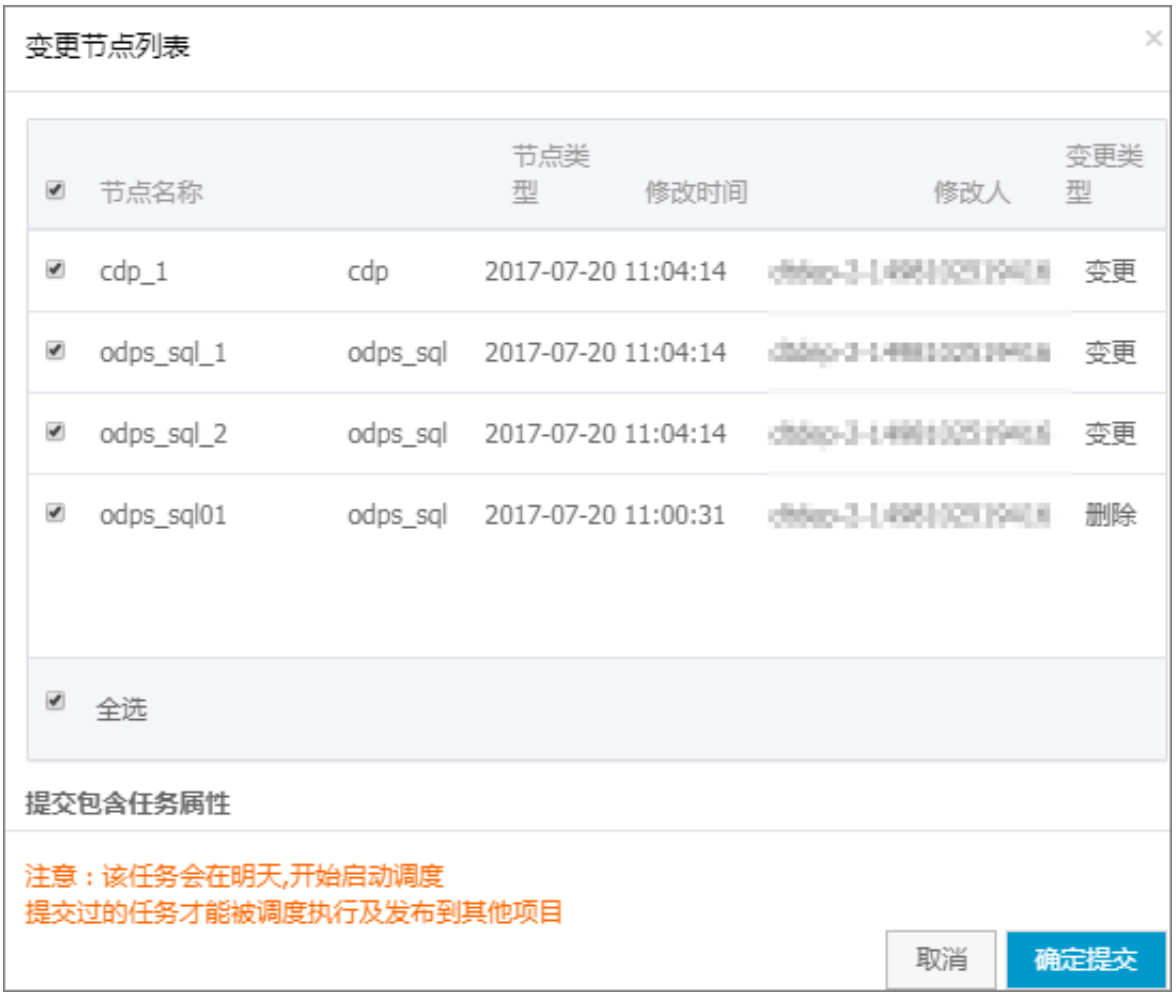
常见场景—任务测试

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：

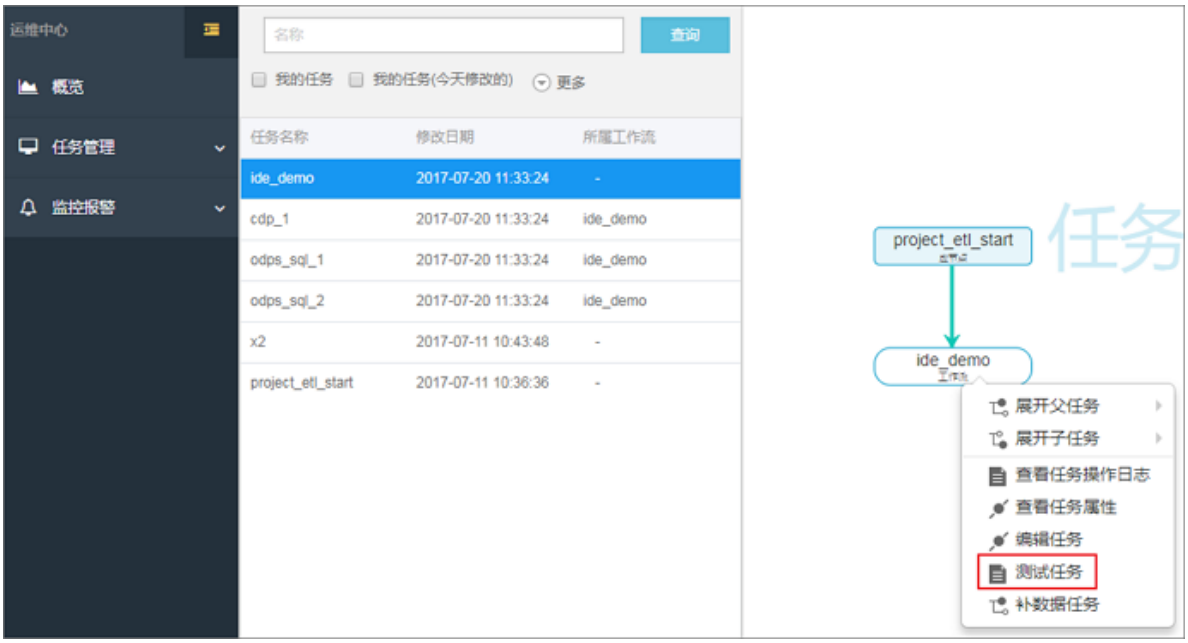
1. 选择所需的项目空间，进入某具体工作流任务开发面板。
2. 单击  进行提交。
3. 在变更节点列表对话框中，单击**确定提交**。

图 4-28: 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，在右侧DAG图中右键单击工作流任务，选择**测试任务。**

图 4-29: 测试任务



6. 在测试运行对话框中选择**业务日期**，单击**生成并运行**。

图 4-30: 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。

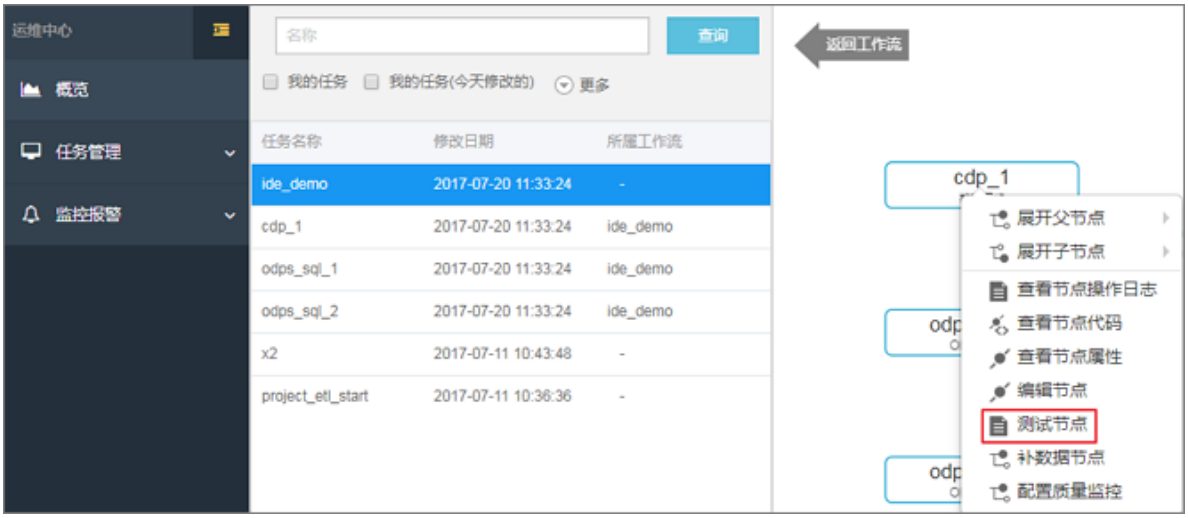
常见场景—节点测试

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，选择节点右键单击**测试节点**。

图 4-31: 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 4-32: 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。

4.4.3 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 4-33: 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 4-34: 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 4-35: 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。

4.4.4 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. 以开发者角色创建项目。
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 4-36: 新建脚本文件弹出框

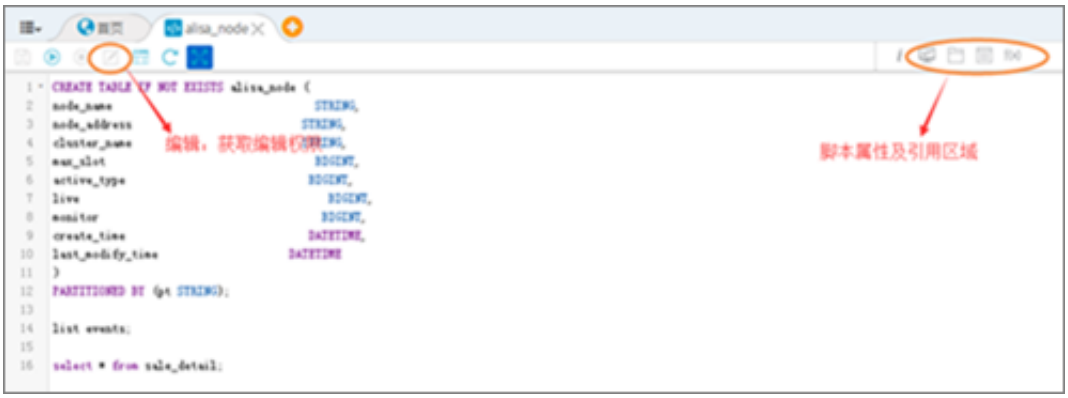
新建脚本文件对话框中配置项具体说明如下：

- 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 4-37: 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：

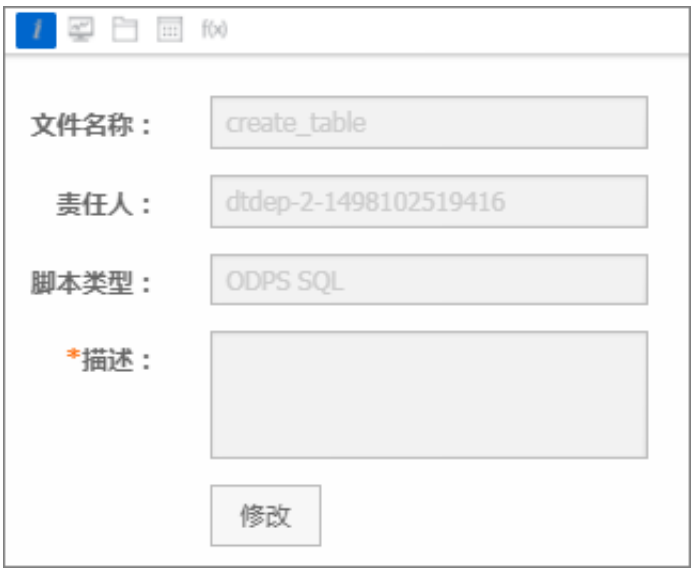
- 编辑 

  如果打开的不是当前用户创建的脚本文件，则默认只读状态，您需单击**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 4-38: 脚本文件属性



4.4.5 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. 以开发者角色创建项目。
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**页面。
3. 单击**上传资源**。
4. 填写**资源上传**弹出框中的各配置项。

图 4-39: 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和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MaxCompute 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。
- 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**。

4.4.6 创建函数

阿里云大数据开发平台为开发者提供了SQL计算功能，您可以在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，您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

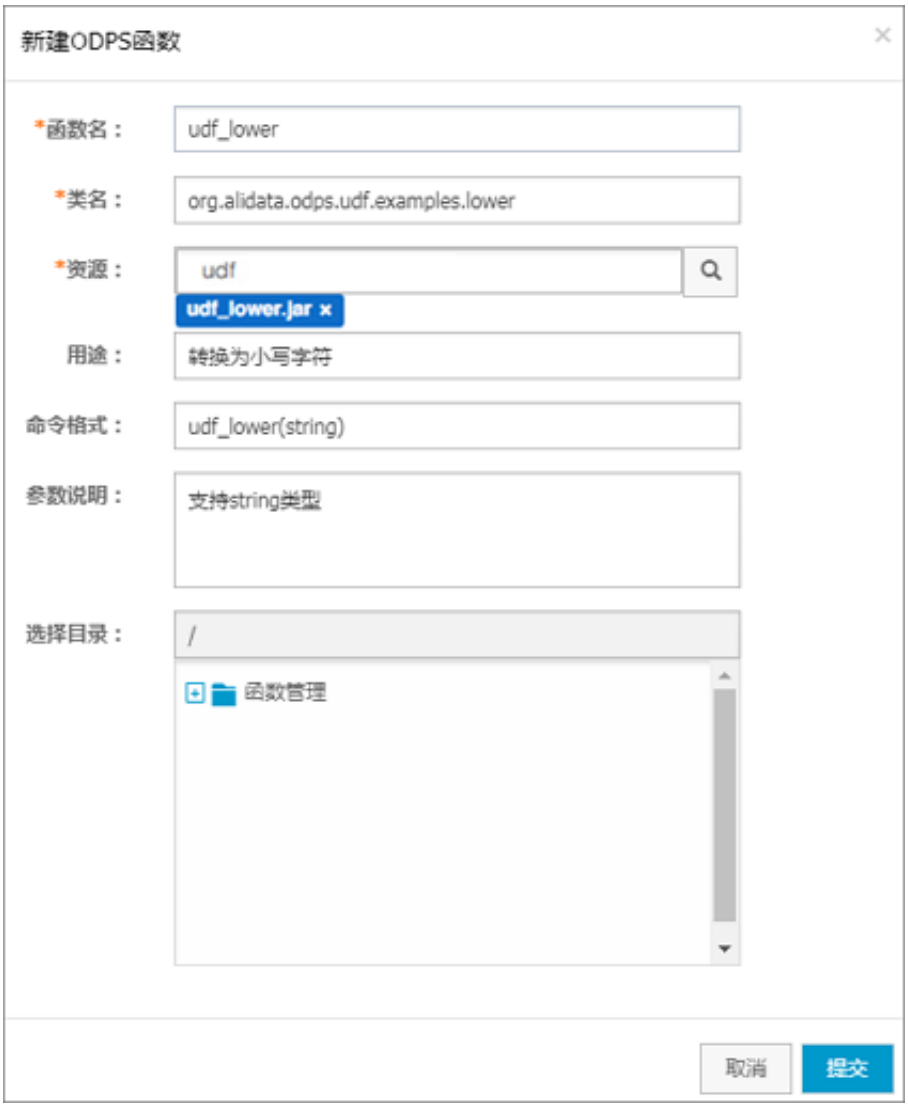
1. 以开发者角色创建项目。
2. 单击顶部菜单栏中的**数据开发**，导航至**开发面板** > **新建**或**管理面板** > **资源管理**。
3. 单击**上传资源**。
4. 填写**资源上传**弹出框中的各配置项。

图 4-40: 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 *4-41:* 新建函数弹出框所示：

图 4-41: 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")
/***------------------------------------------------***/
```

4.4.7 使用Spark

Spark在DataWorks中使用。

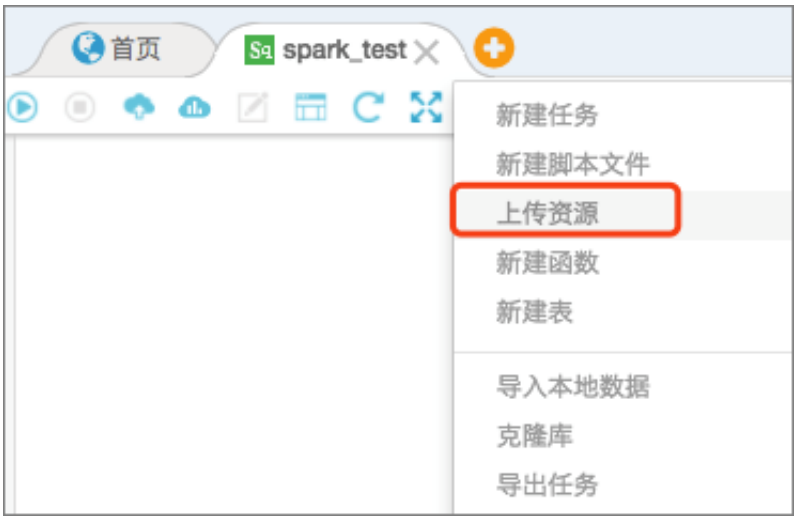
操作步骤

1. 新建SQL节点。

图 4-42: 新建SQL节点

2. 上传jar包。

图 4-43: 上传jar包



3. 上传配置文件。

   由于是ODPS_SQL节点任务，配置文件中可以不设置MaxCompute（原ODPS）的相关配置，这些信息可以从base环境中自动获取。

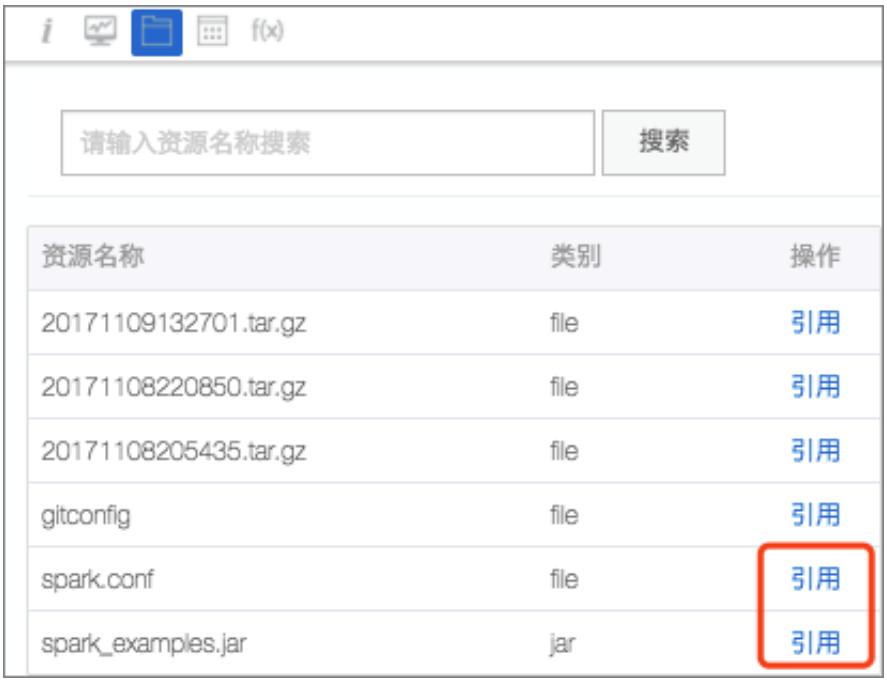
```
spark.hadoop.odps.project.name
spark.hadoop.odps.access.id
spark.hadoop.odps.access.key
```

```
spark.hadoop.odps.end.point
```

图 4-44: 上传配置文件

4. 编写spark jar命令，找到需要引用的资源，单击操作栏的**引用**。

图 4-45: 引用资源



示例代码：

- java Spark示例：

```
--@resource_reference{"spark.conf"}
--@resource_reference{"spark_examples.jar"}
jar -classpath
/opt/taobao/tbdpapp/spark-on-odps/2.1.0/__spark_libs__.zip,
spark_examples.jar
com.aliyun.odps.cupid.tools.OdpsCltWrapper
"--class" com.aliyun.odps.spark.examples.SparkPi
"--properties-file" spark.conf
"--conf" spark.yarn.archive=/opt/taobao/tbdpapp/spark-on-odps/2.1.
0/__spark_libs__.zip
"--master" yarn-cluster
spark_examples.jar
```

- py spark示例：

```
##@resource_reference{"odps_table_rw.py"}
##@resource_reference{"spark.conf"}
##@resource_reference{"odps.zip"}
export PYSPARK_ARCHIVES_PATH=/opt/taobao/tbdpapp/spark-on-odps/2
.1.0/python/lib/py4j-0.10.4-src.zip,/opt/taobao/tbdpapp/spark-on-
odps/2.1.0/python/lib/pyspark.zip,odps.zip
java -cp /opt/taobao/tbdpapp/spark-on-odps/2.1.0/:/opt/taobao
/tbdpapp/spark-on-odps/2.1.0/jars/* org.apache.spark.deploy.
SparkSubmit \
"--properties-file" spark.conf \
"--jars" /opt/taobao/tbdpapp/spark-on-odps/2.1.0/cupid/odps-spark-
datasource-2.0.4.jar \
```

```
"--conf" spark.yarn.archive=/opt/taobao/tbdpapp/spark-on-odps/2.1.
0/__spark_libs__.zip \
"--master" yarn-cluster  \
odps_table_rw.py
```

说明：

- *odps.zip*。
- 用户的python入口类（以odps_table_rw.py为例：https://github.com/aliyun/aliyun-cupid-sdk/blob/master/examples/spark-examples/src/main/python/odps_table_rw.py）。
- 由于采用Shell节点，MaxCompute相关相关信息不能缺省。

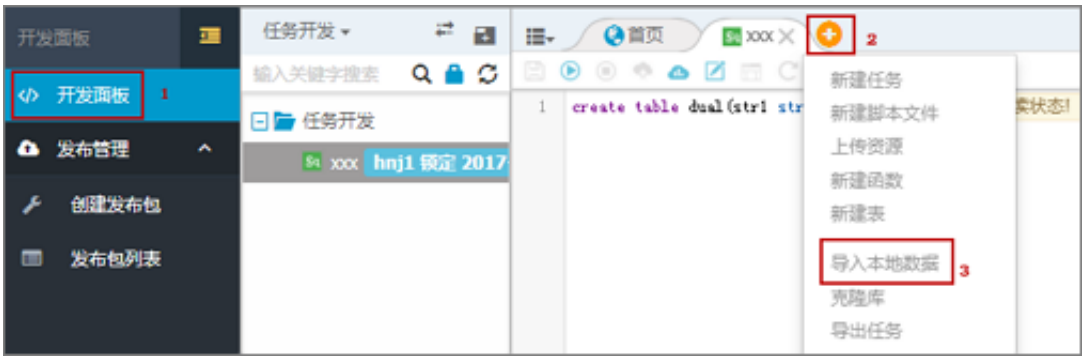
4.4.8 导入本地数据

阿里云大数据平台提供导入本地数据的功能，仅支持txt和csv文件类型。

操作步骤

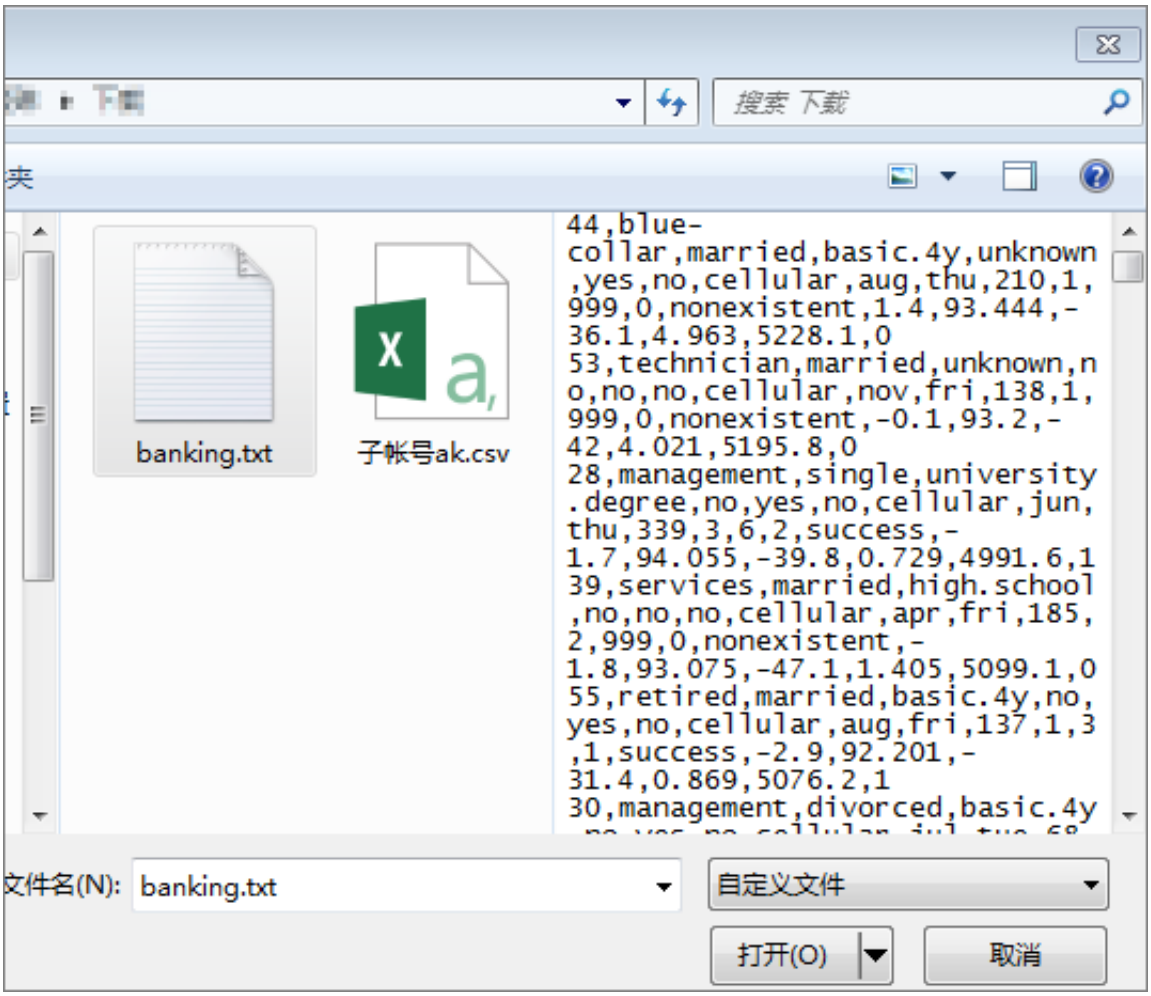
1. 以开发者角色创建项目。
2. 单击顶部菜单栏中的**数据开发**，导航至**开发面板** > **新建**。
3. 单击**导入本地数据**。

图 4-46: 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 4-47: 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**。

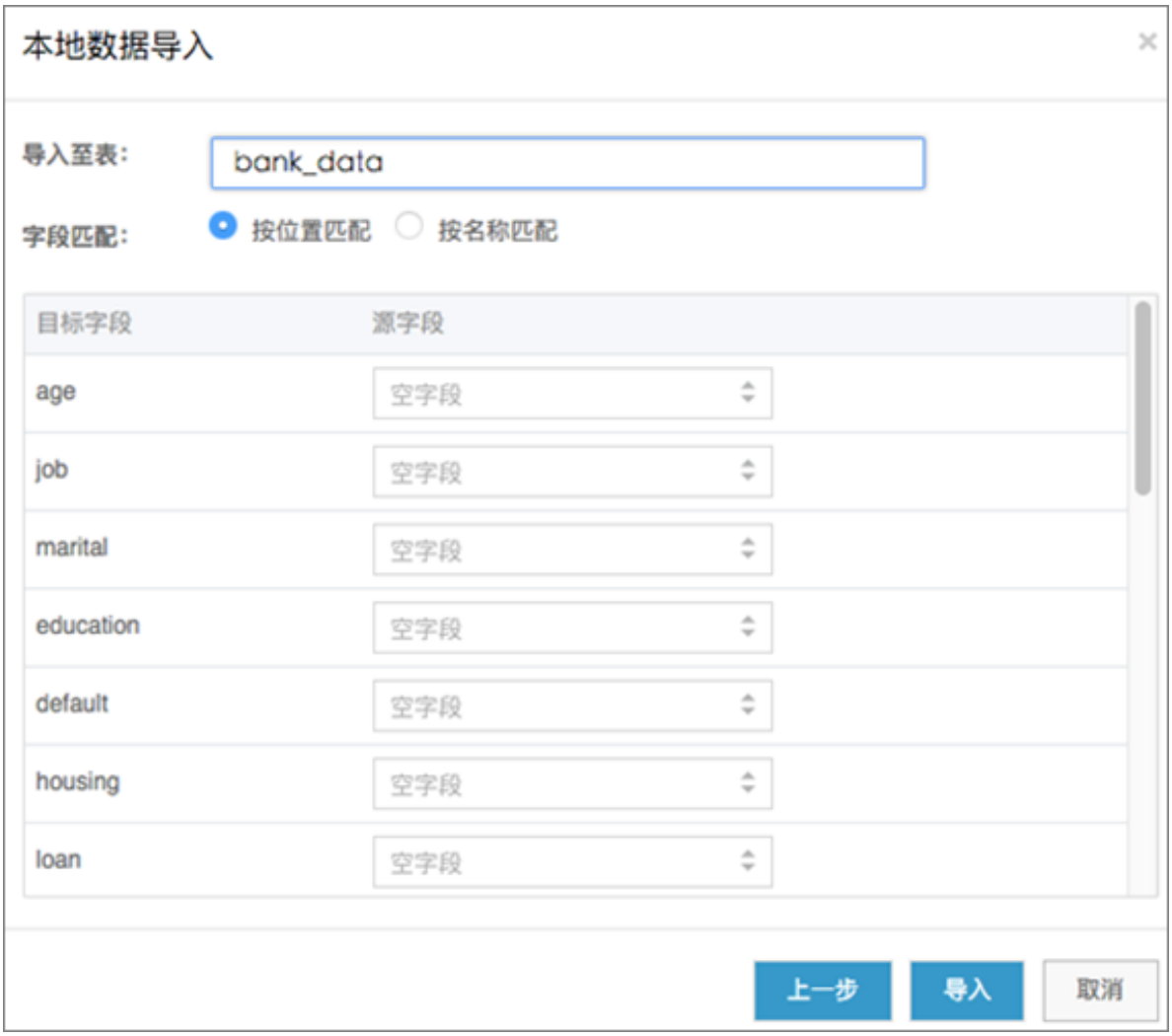
图 4-48: 本地数据导入配置

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. 选择选择需要导入到的MaxCompute表并配置**字段匹配**。

图 4-49: 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**。

4.4.9 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/运维角色创建项目。
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 4-50: 创建发布包页面

说明：

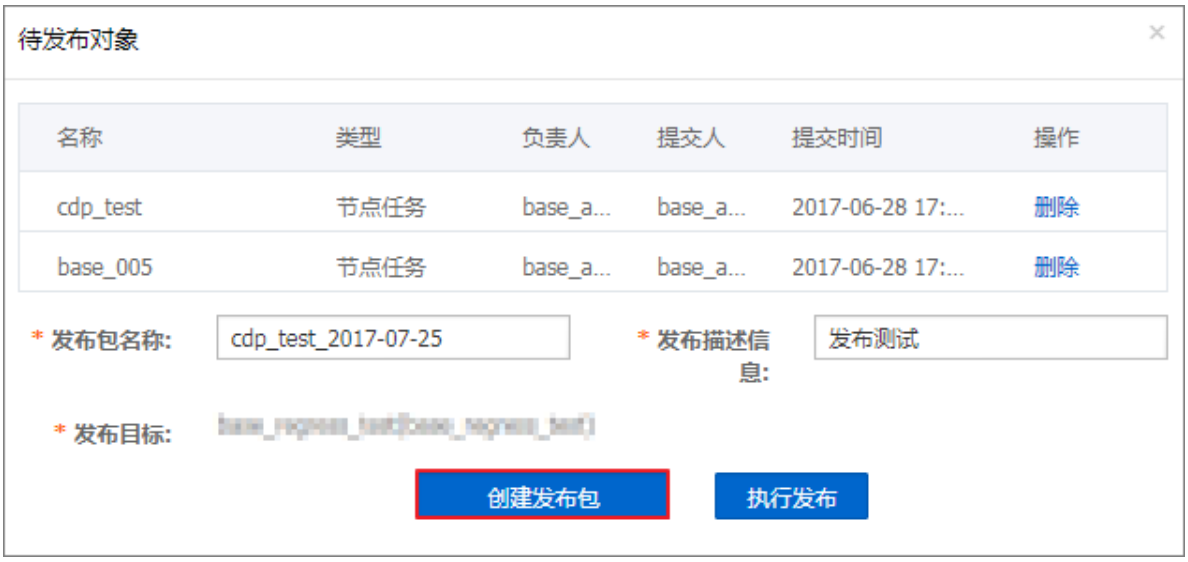
您需确保当前项目已绑定发布目标项目，方可执行发布操作。若未绑定，请联系管理员前往**项目管理**，绑定发布目标项目。

5. 单击右上角的发布篓，弹出待发布对象页面。

图 4-51: 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 4-52: 待发布对象



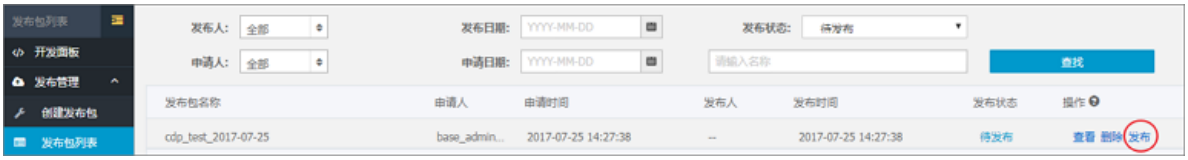
说明：

- **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。

具体权限划分请参见权限划分。

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 4-53: 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。

4.4.10 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. 创建项目。
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 4-54: 项目管理-计算引擎配置

4.4.11 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*DataWorks控制台*。
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 4-55: 数据源配置

4.5 数据开发

开发者在阿里云DataWorks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。

4.5.1 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。

4.5.2 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. 以开发者角色创建项目。
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 4-56: 模糊查询

4. 双击需要编辑的任务，即可进入任务页面进行编辑操作。

> **说明**：
> 任务在没有其他人编辑的状态下，您即可根据任务类型进行相关编辑。

4.5.3 复制任务

DataWorks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. 以开发者角色创建项目。
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 4-57: 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 4-58: 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。

4.5.4 删除任务

背景信息

开发者也可删除已创建的任务。

> **说明**：
> 删除工作流任务操作及下线工作流任务，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，需谨慎操作。

操作步骤

1. 以开发者角色创建项目。
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工作流任务名称来模糊匹配查找需删除的工作流任务。

图 4-59: 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 4-60: 删除确认对话框

6. 单击**确认删除任务**。

4.5.5 删除脚本文件

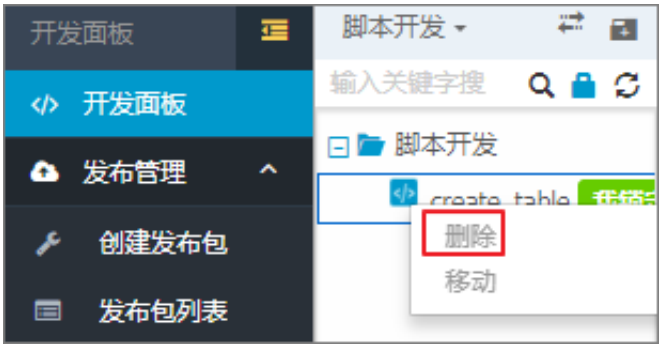
开发者也可删除已创建的脚本文件。

操作步骤

1. 以开发者角色创建项目。
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 4-61: 脚本文件管理列表



4.5.6 删除资源

开发者可删除已有的资源文件。

操作步骤

1. 以开发者角色创建项目。
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 4-62: 资源文件管理列表

4.5.7 删除函数

开发者可删除已创建的函数。

操作步骤

1. 以开发者角色创建项目。
2. 单击顶部菜单栏中的**数据开发**，导航至**开发面板** > **函数管理**。

图 4-63: 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，即可从数据开发工作台和底层MaxCompute（ODPS）计算引擎中删除已创建的UDF。

4.5.8 权限划分

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、部署和访客的权限如下表所示。

表 4-1: 数据开发模块权限划分

父权限名称	权限名称	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	√	√	—	—	—

父权限名称	权限名称	项目管理员	开发	运维	部署	访客
	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	√	√	√	√	√
	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	√	√	—	—	—

父权限名称	权限名称	项目管理员	开发	运维	部署	访客
	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	√	√	—	—	—

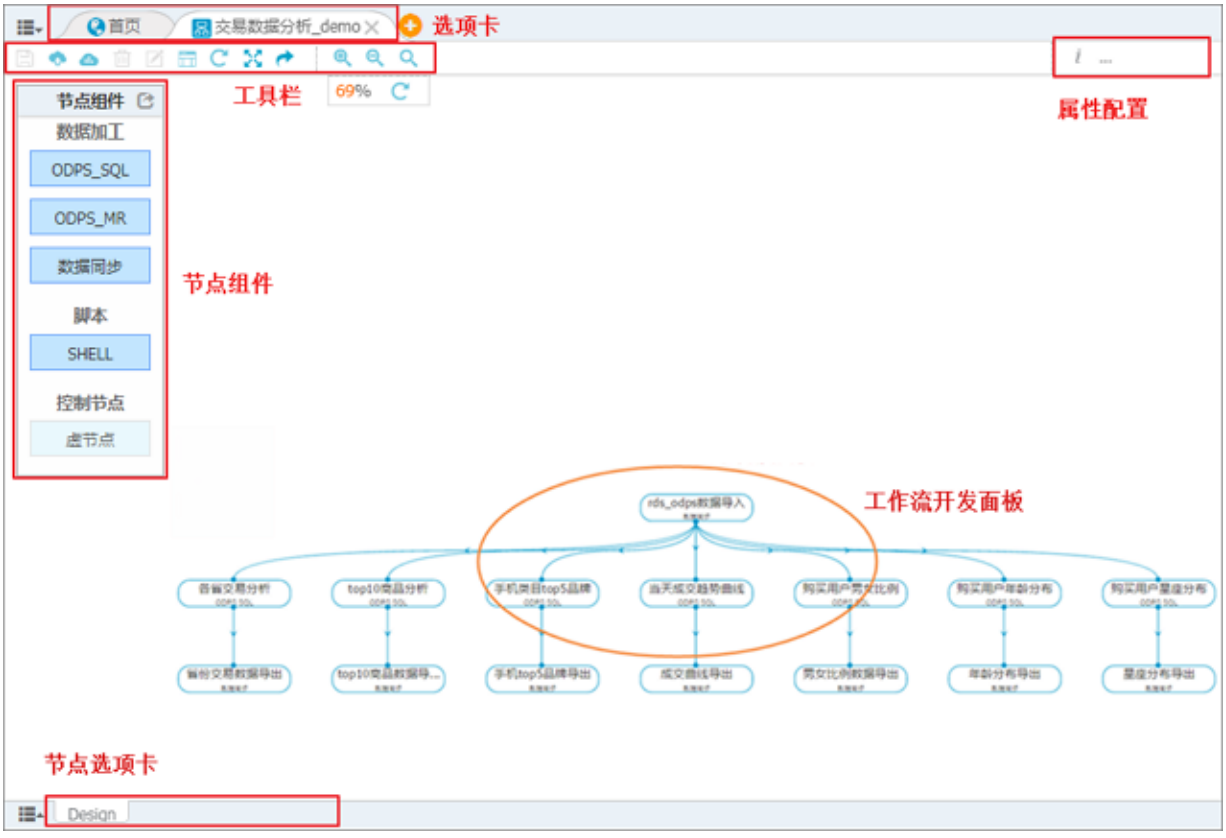
4.5.9 工作流任务设计器

4.5.9.1 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 *4-64: 工作流任务设计器*所示。

图 4-64: 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。
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，但不能因此简单地把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。您可以使用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DataWorks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。

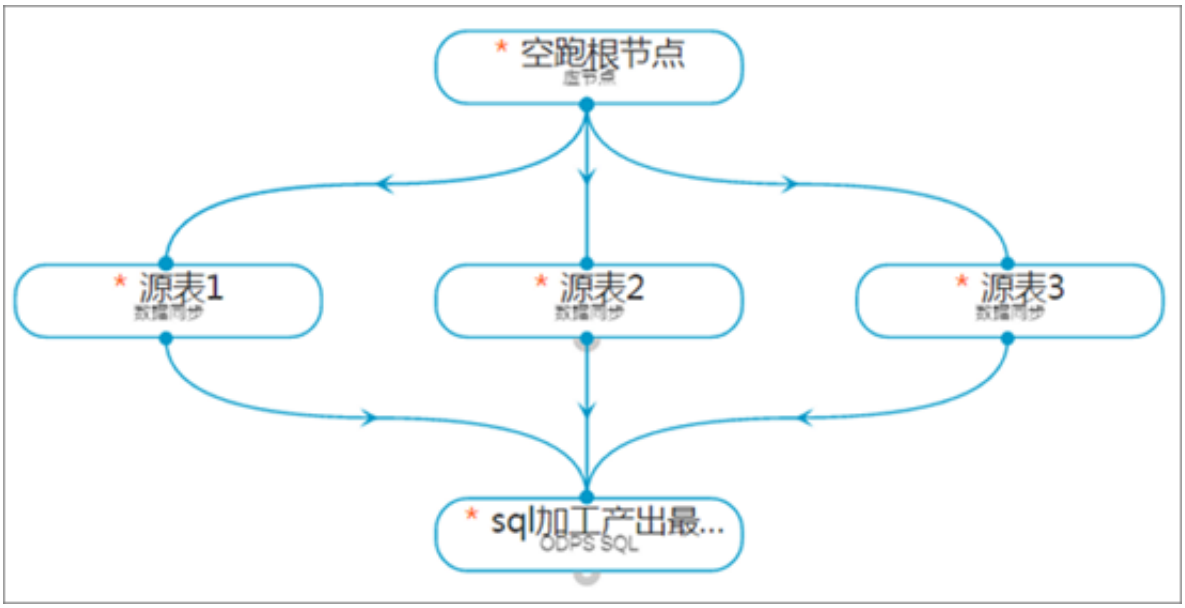
- 虚拟节点（virtual）

  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。

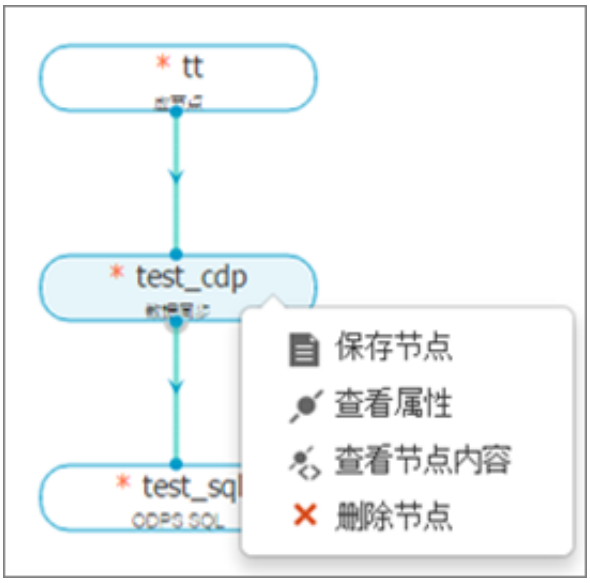
入门示例：输出表由3个数据同步任务导入的源表经过MaxCompute SQL任务加工产出，这3个数据同步任务没有依赖关系，MaxCompute SQL任务需要依赖3个同步任务，则工作流任务如图 4-65: 入门示例所示。

图 4-65: 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**保存节点**、**查看属性**、**查看节点内容**和**删除节点**。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内容页面。
- 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。

4.5.9.2 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 *4-66:* 工作流任务属性配置所示。

图 4-66: 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**

- 调度状态：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。
- 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。

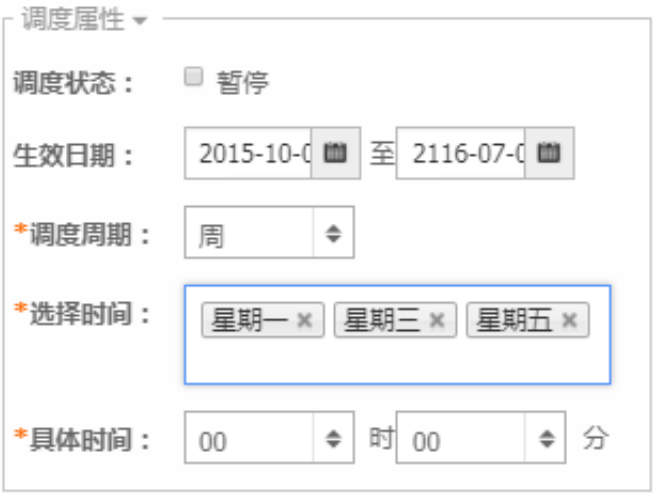
  - 天调度：每天调度一次，可设置具体调度时间。

    图 4-66: 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 4-67: 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 4-68: 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 4-69: 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 4-70: 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**。

4.5.9.3 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 *4-72: 节点属性*所示。

图 4-72: 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**
  - 节点名称：新建节点时填写的节点名称，不支持修改。
  - 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。
  - 节点类型：当前节点的类型，不支持修改。
  - 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。

4.5.10 节点任务编辑器

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。

4.5.11 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 4-73: 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		**返回工作流任务**
2		**保存节点**，保存当前工作流任务变更，支持快捷键ctrl+s。

序号	标识	标识说明
3		**运行节点**，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。
4		**停止运行**，正在运行代码时，不需要运行可以停止。
5		**格式化**，代码格式化。
6		**全屏**，工作区全屏。
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。

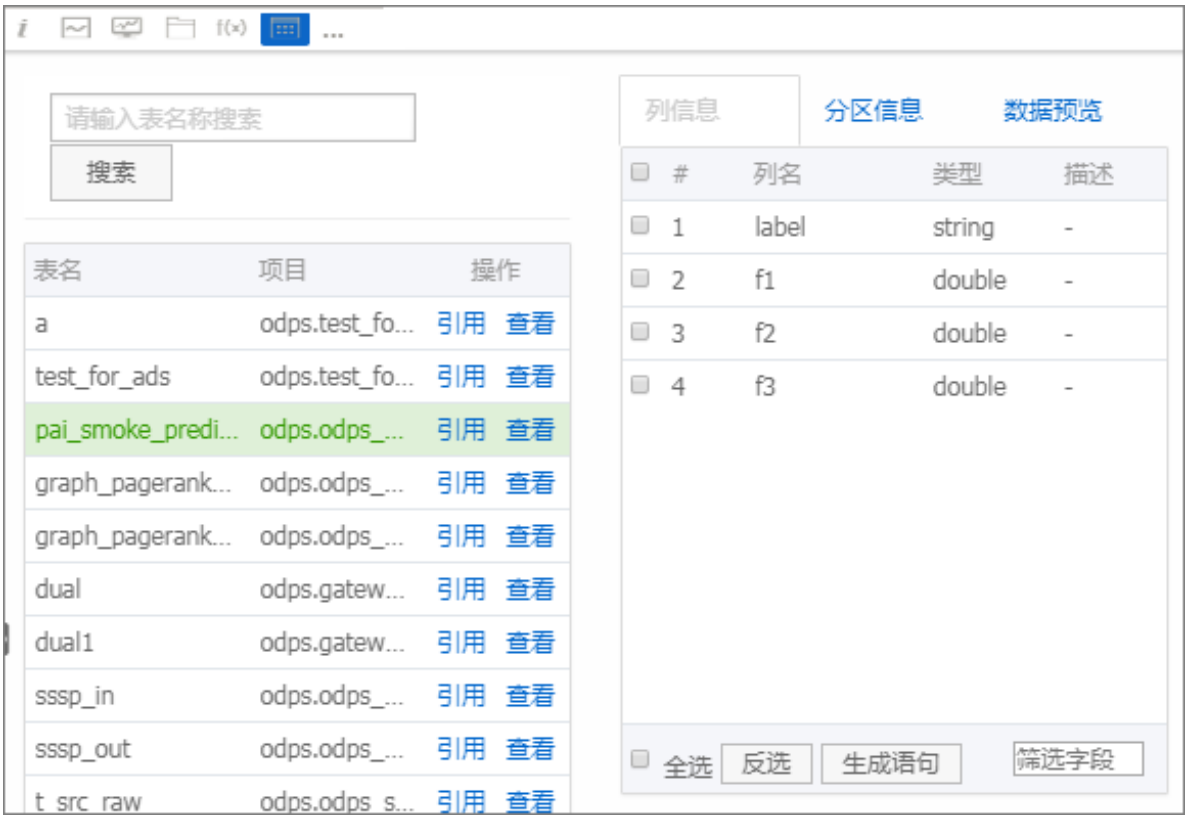
 **说明**：

单击**运行**，MaxCompute任务将以该项目空间对应的计算引擎所配置的身份来运行。

- 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。如果需要提交为调度任务，则需要改为调度参数再进行提交，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 4-74: 找表组件

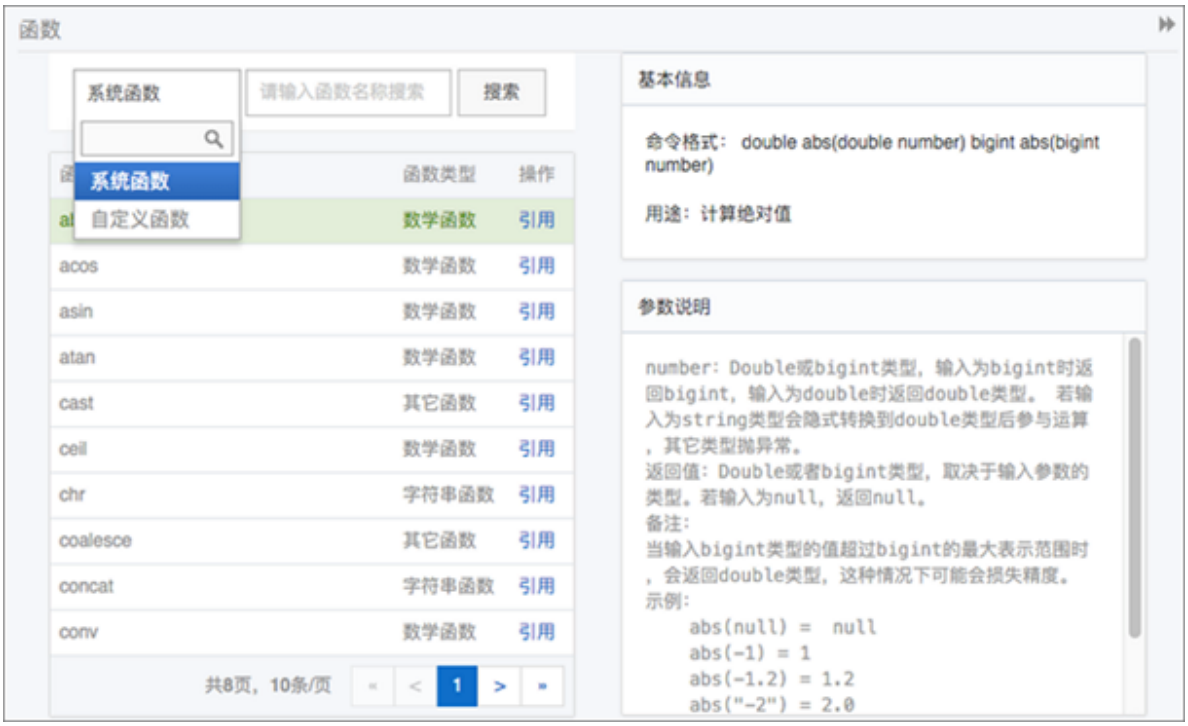


也可对该表进行如下操作。

- 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 4-75: 函数列表

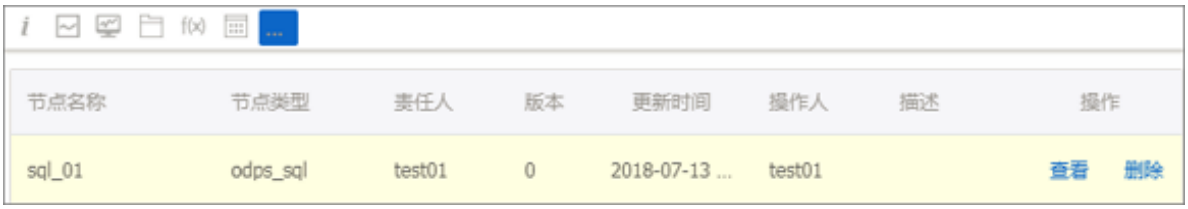


- 资源列表：可模糊匹配查找本项目空间内的资源。单击**引用**会在代码编辑区光标处引用该资源文件。

图 4-76: 资源列表

- 节点列表：显示当前打开节点的名称、类型、责任人等信息，可进行**查看**和**删除**操作。

图 4-77: 节点列表



4.5.12 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，调度日期默认为当前日期的前一天（年月日）。

- ${bdp.system.cyctime}

  格式为yyyymmddhh24miss，调度实例的定时时间（年月日时分秒）。 yyyy表示4位数的年份，mm表示2位数的月份，dd表示2位数的天，hh24表示24小时制的时，mi表示2位数的分钟，ss表示2位数的秒。

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。

节点代码如下所示。

```
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所示。

图 4-78: 工作流任务属性

图 4-79: 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。参数详情如下所示。

- $[...]基于系统参数${bdp.system.cyctime}进行自定义配置。

  ${bdp.system.cyctime}=$[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。

节点代码如下所示。

```
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、节点属性如下所示。

图 4-80: 工作流任务属性

图 4-81: 节点参数配置

自定义参数配置如下所示。

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。

4.5.13 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页面，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。

4.5.13.1 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 4-82: 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 4-83: 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。

图 4-84: 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 4-85: 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。

图 4-85: 添加选择源库

图 4-86: 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 4-88: 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 4-89: 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。

4.5.13.2 字段配置

关系型数据库同步

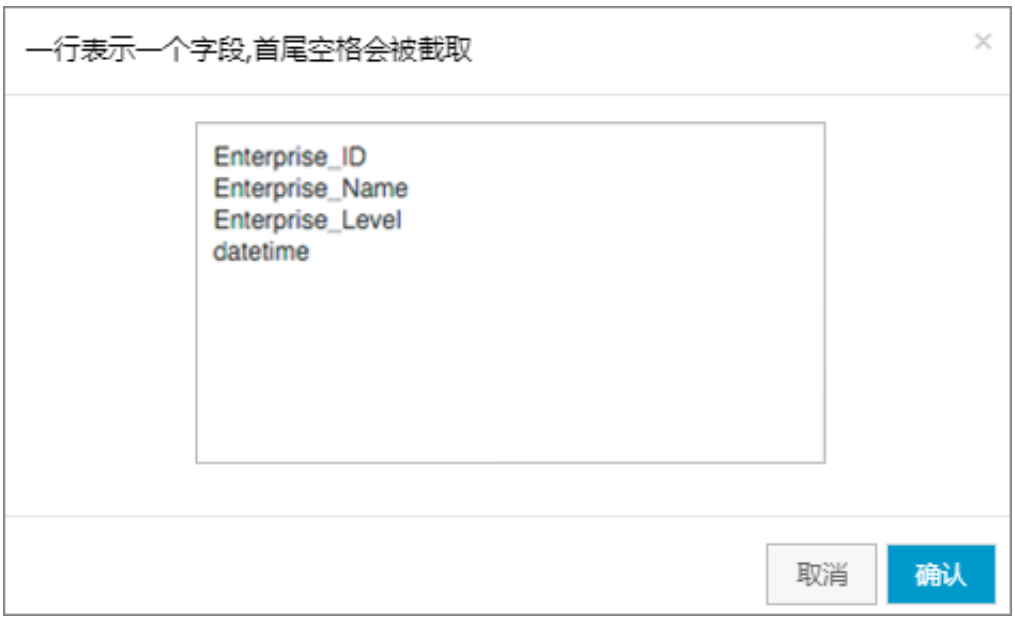
其次需对字段映射关系进行配置，左侧**源表字段**和右侧**目标表字段**为一一对应的关系。

图 4-90: 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 4-91: 批量编辑弹出框



- 增加/删除

  单击可单个增加/删除字段。

  图 4-92: 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 ‘${yesterday}’ ，然后在参数配置模块给变量赋值，如yesterday=$[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 4-93: 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。

4.5.13.3 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 4-94: 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 4-95: 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 4-96: 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 4-97: 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。

4.5.13.4 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 *4-98:* 流量与出错控制所示。

图 4-98: 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。

- **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。

4.5.14 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。

4.5.15 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。

4.6 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

4.6.1 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。

4.6.2 权限划分

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 4-2: 数据管理模块角色与其权限划分

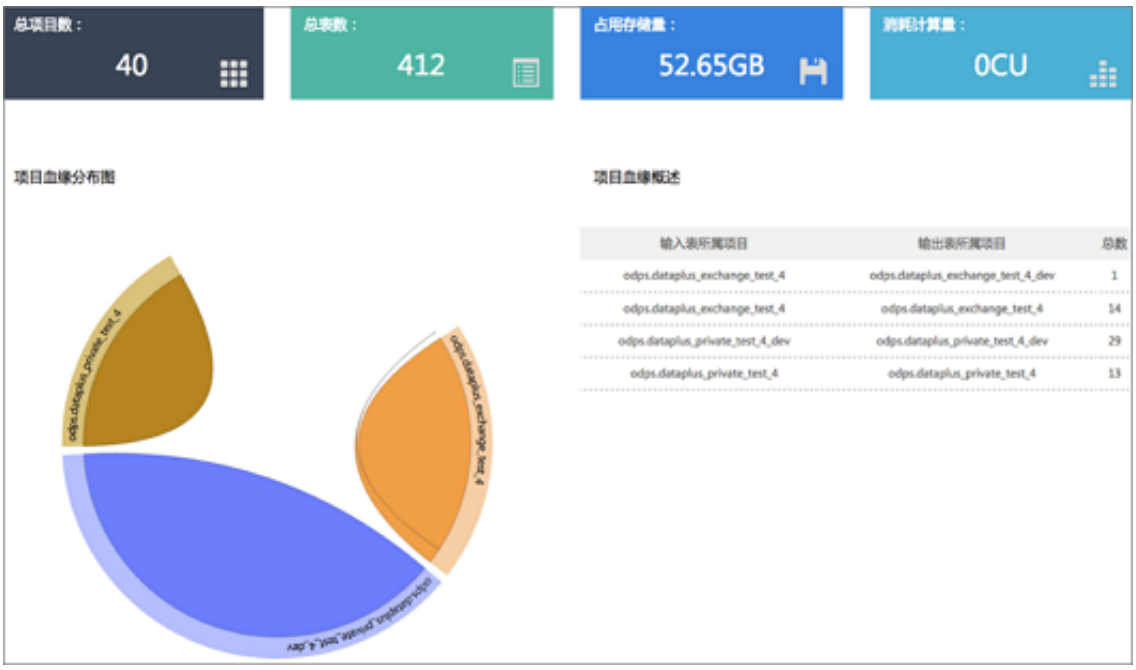
功能模块	权限	组织管理员	项目管理员	开发
权限管理	权限审批与收回	—	√	—
管理配置	类目导航配置	√	—	—

功能模块	权限	组织管理员	项目管理员	开发
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	√	√	√
	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	√	√	√

4.6.3 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 4-99: 全局视图



项目占用存储Top		表占用存储Top		热门表	
odps.dataplus_private_test_6	51.79GB	logtest	51.79GB	tianchi_mobile_recommend_train_u...	
odps.dataplus_private_test_4_dev	420.11MB	tianchi_mobile_recommend_train_u...	124.23MB	sm_test_long	
odps.dataplus_private_test_4	332.83MB	tianchi_mobile_recommend_train_u...	118.69MB	tmall_train_ub_action_view	
odps.dplus_test_1	66.59MB	ods_alipath_xlib	25.86MB	sm_test_ex	
odps.dataplus_private_test_3_dev	45.62MB	ods_log_tracker	23.05MB	mb_test_0013	
odps.dataplus_exchange_test_4_dev	13.78MB	tianchi_mobile_recommendation_p...	20.07MB	mb_00002	
odps.dataplus_exchange_test_4	2.23MB	tianchi_mobile_recommendation_p...	20.07MB	mb_00004	
odps.dataplus_private_test_5_dev	23.31KB	ods_alipath_xlib_predict	13.16MB	mbna0001	
odps.dataplus_private_test_5	21.09KB	pai_temp_29_120_1	12.91MB	mb00000001	
odps.dataplus_exchange_test	0.00B	pai_temp_29_116_1	12.91MB	dual	

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CU）。
- 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。

4.6.4 数据搜索—找表

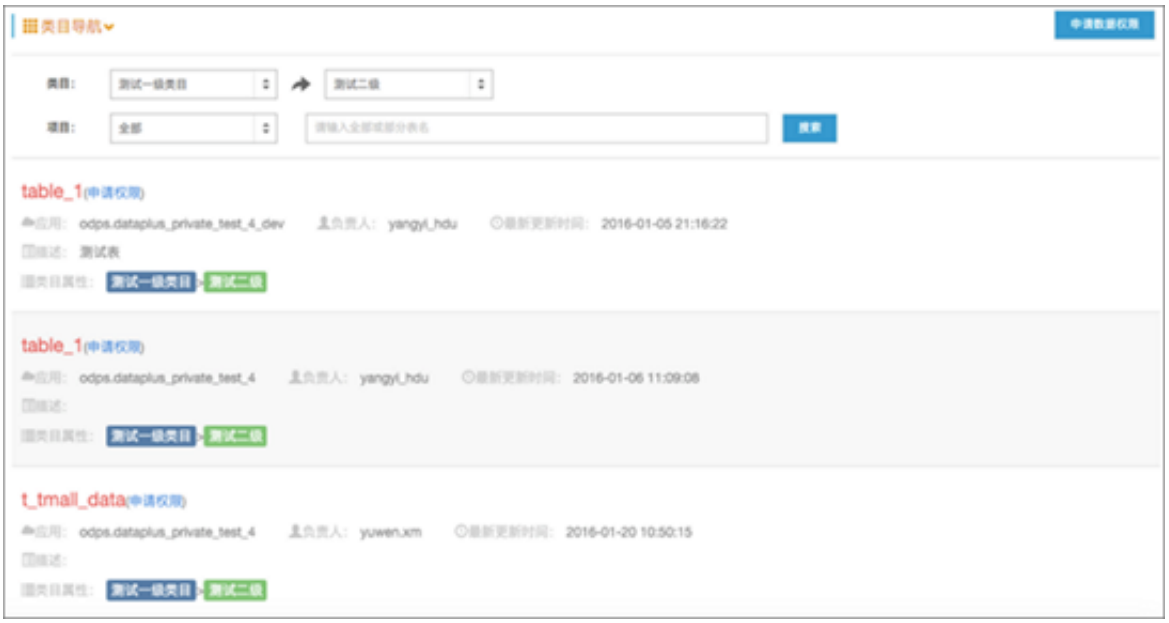
背景信息

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. 以开发者身份创建项目。
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 4-100: 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。

4.6.5 申请数据权限

在DataWorks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。

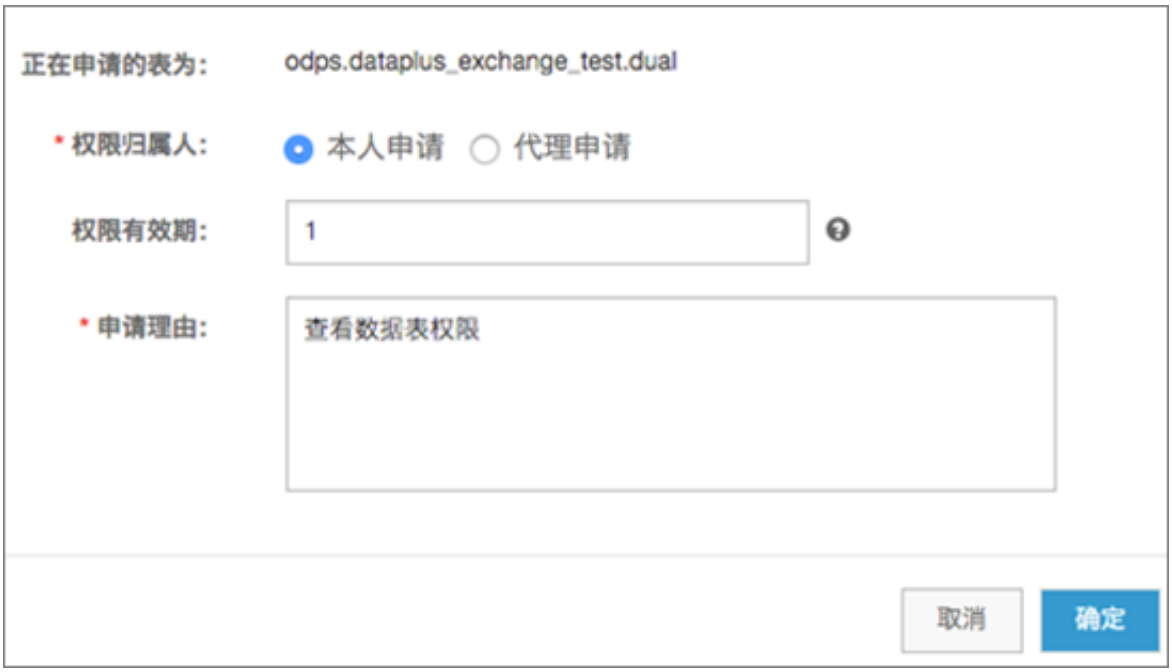
数据表权限的申请

1. 通过数据搜索找到需申请授权的数据表。
2. 单击数据表操作栏中的**申请权限**。

图 4-101: 申请权限

3. 在**申请权限**弹出框中，填写相关配置项。

图 4-102: 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 4-102: 代理申请

- 申请理由：申请数据表权限理由的简单描述。

4. 单击**确定**。

自定义函数或资源文件的权限申请

1. 登录DataWorks。
2. 单击顶部菜单栏中的**数据管理** > **查找数据**。
3. 单击**申请数据权限**。

4. 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 4-104: 函数权限申请

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，函数的读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，函数的读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。
- 函数名称：选择对应项目名称下的函数。
- 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 4-105: 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，资源的读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，资源的读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：

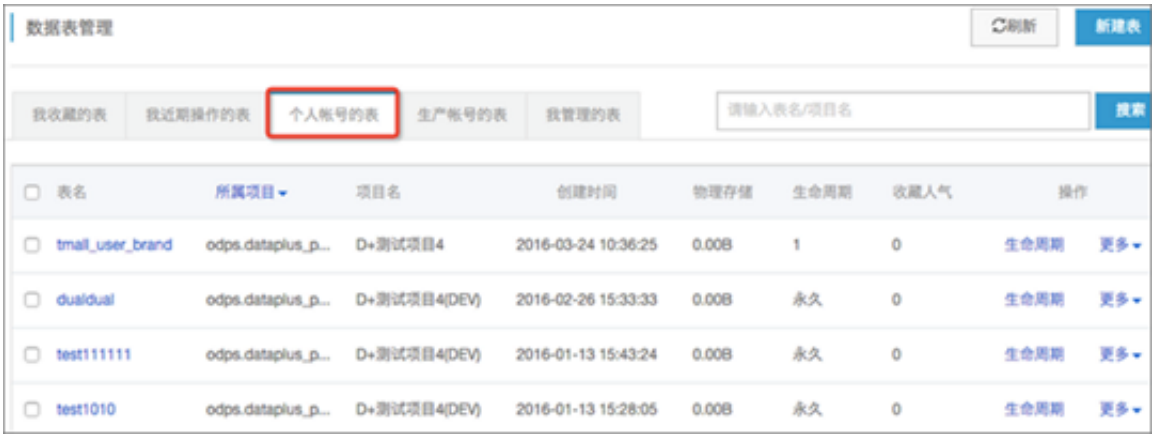
系统默认当前用户具有所在项目空间中所有表/自定义函数/资源文件的读权限，无权访问的项目空间（跨项目空间）中的表/自定义函数/资源文件，需要通过**申请数据权限**来获取读权限。

4.6.6 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 4-106: 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。

- 单击**生命周期**可修改表的生命周期。

图 4-106: 生命周期修改

- 单击**更多**，可进行表管理、隐藏和删除操作。
  - **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。

4.6.6.1 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

1. 以开发者身份创建项目。
2. 单击顶部菜单栏中的**数据管理**，导航至**数据表管理**。
3. 单击右上角的**新建表**。

图 4-108: 数据表管理界面

4. 在**新建表**页面中，填写基础信息。

图 4-109: 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，输入需要刷新的项目名字，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、自定义五种选项。

5. 单击**下一步**。
6. 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 4-110: 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。
   - 安全等级：设置各字段的安全等级（Label Security）。
   - 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。

7. 单击**提交**。

说明：
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

1. 以开发者身份创建项目。
2. 单击顶部菜单栏中的**数据管理**，导航至**数据表管理**。
3. 单击**新建表**。
4. 单击新建表页面右上角的**DDL建表** 。
5. 填写MaxCompute SQL建表语句。

图 4-111: 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;
```

6. 单击**提交**。

图 4-112: 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 4-113: 自动填充字段和分区信息相关配置项

7. 补充新建表基础信息页面中的配置项，单击**下一步**。

图 4-114: 补充基础信息相关配置项

8. 单击**提交**。

> **说明**：
> 通过可视化建表与DDL建表，需您在**用户信息**中绑定AK信息，否则建表时将报错。

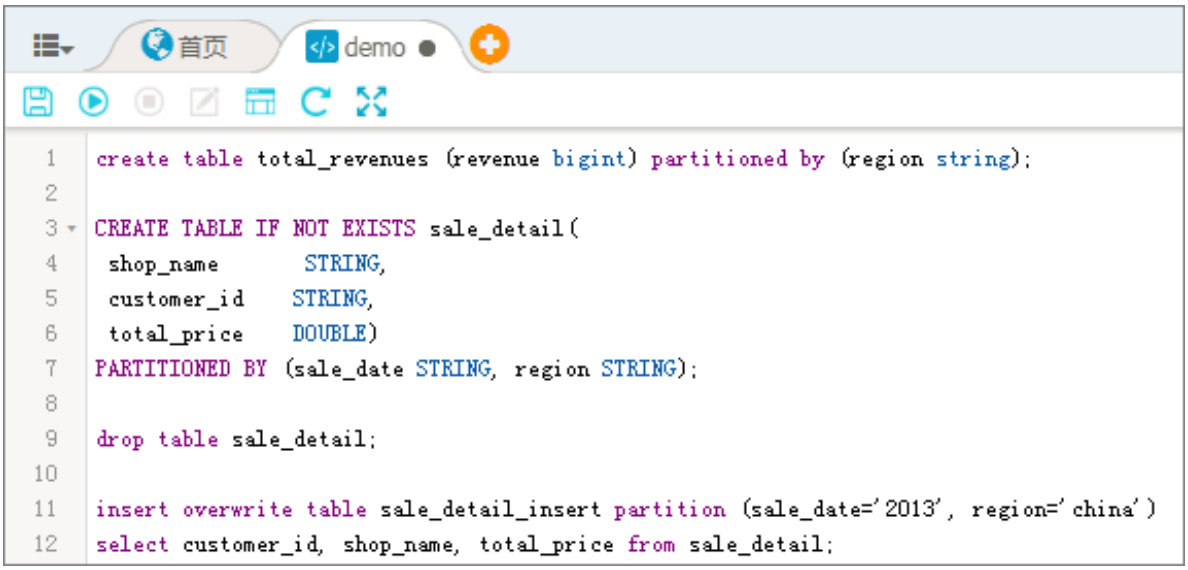
新建脚本文件建表

1. 以开发角色身份创建项目。
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**新建脚本文件**。
4. 在**新建脚本文件**弹出框中，填写各配置项。

图 4-115: 新建脚本文件弹出框

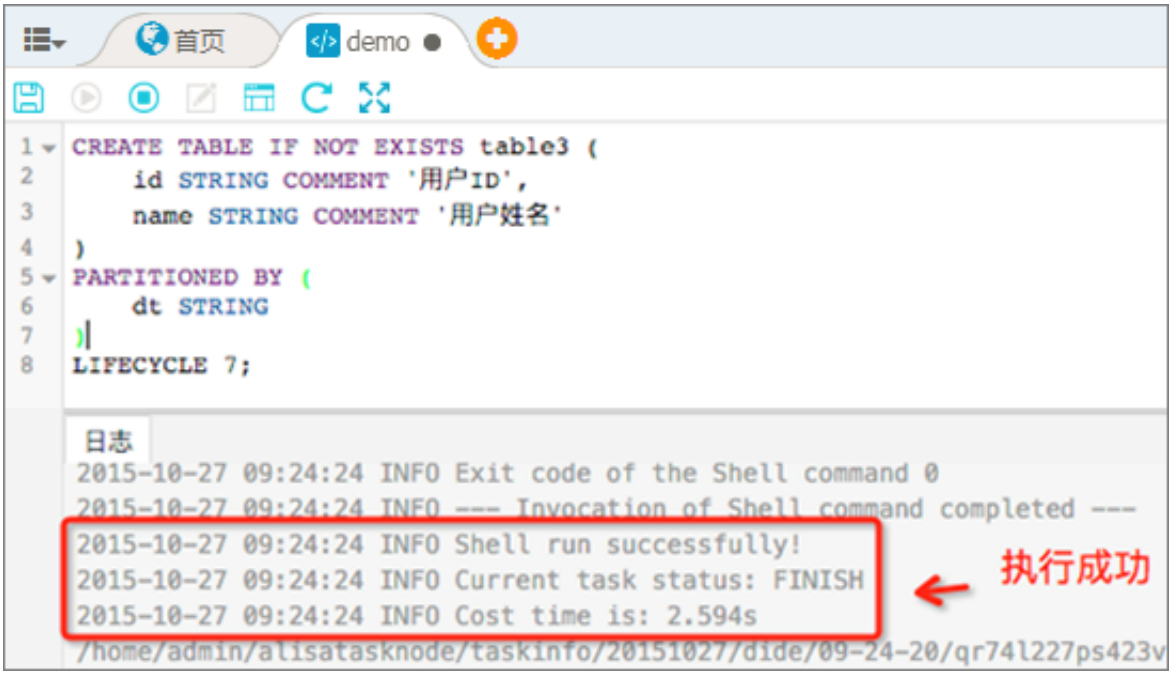
5. 单击**提交**。
6. 在MaxCompute（原ODPS）代码编辑器中编写建表DDL语句。

图 4-116: 在脚本文件中建表



7. 单击**运行**。

图 4-117: 执行DDL脚本



说明：

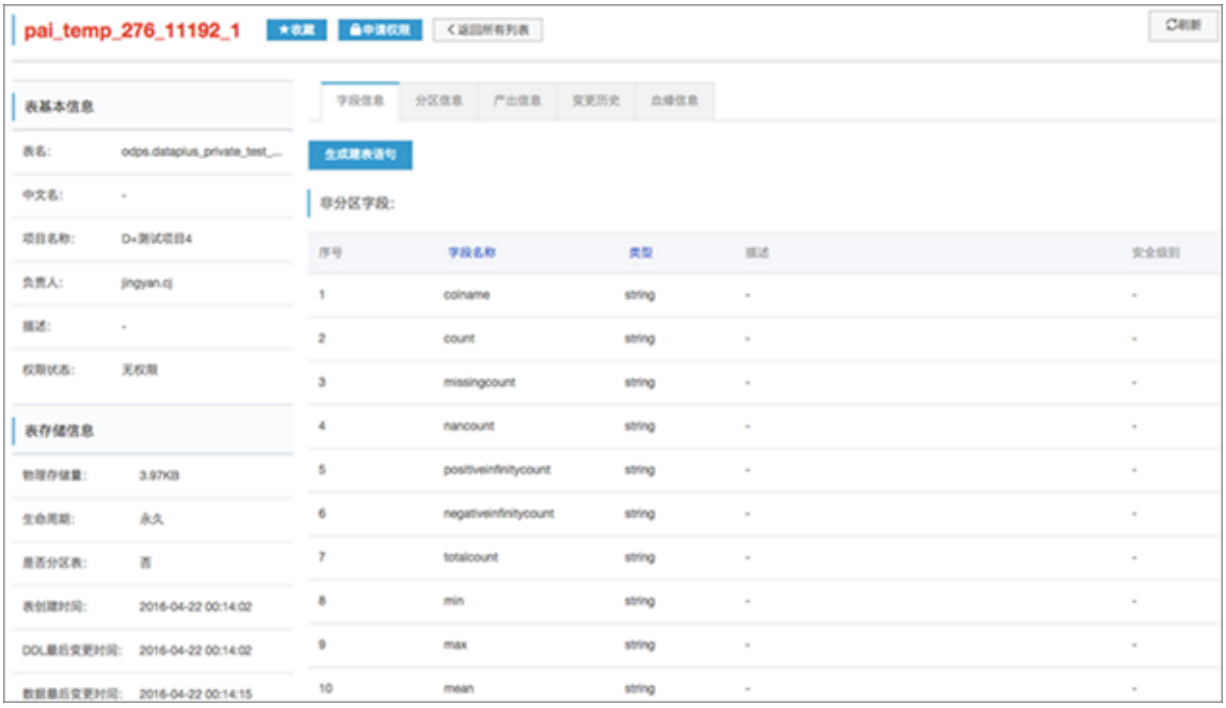
通过方式一、方式二新建表默认责任人为当前登录用户，而通过方式三（新建脚本文件建表）新建表的责任人将为 **MaxCompute（原ODPS）访问身份**所对应的用户。查看当前项目空

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。

4.6.6.2 查看表详情

在数据表管理模块中任意页面，单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 4-118: 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 4-118: 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 4-119: 表基本信息

- 查看表存储信息：包括物理存储量、生命周期、是否分区表、表创建时间、DDL最后变更时间、数据最后变更时间。

图 4-120: 存储信息

- 查看字段信息：包括字段名称、类型、描述，也可单击**生成建表语句**按钮生成该表的DDL语句。

图 4-121: 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 4-122: 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 4-123: 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 4-124: 血缘信息

图 4-125: 字段级血缘

4.6.6.3 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 4-127: 收藏表

收藏后，可在**我收藏的表**页面中取消收藏。

4.6.6.4 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：**修改个人账号的表的生命周期**

1. 以开发者身份创建项目。
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 4-128: 个人账号的表-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 4-129: 修改生命周期弹出框

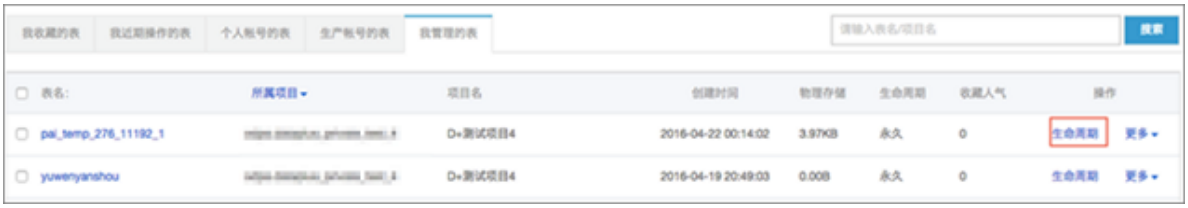
5. 单击**确定**。

> **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**

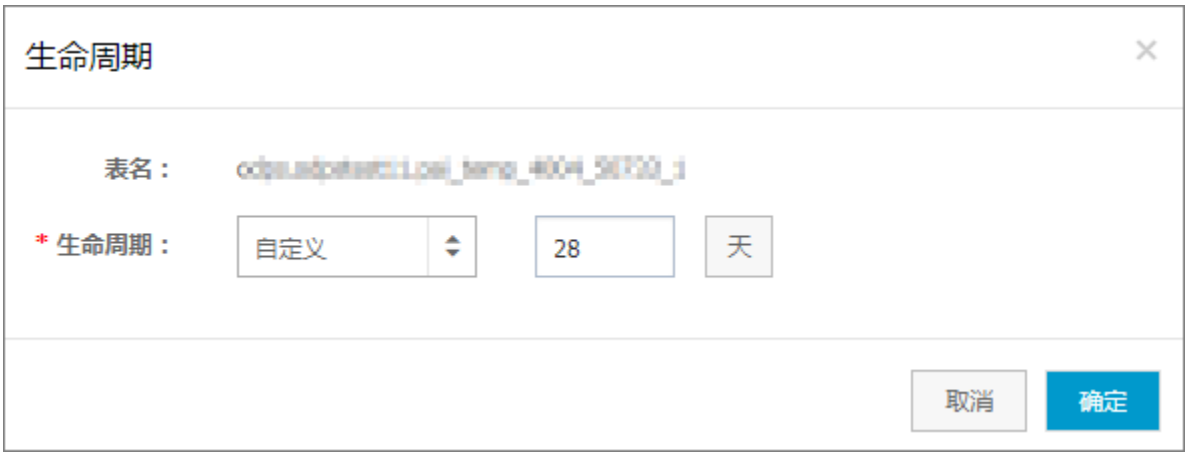
1. 以管理员身份创建项目。
2. 单击顶部菜单栏中的**数据管理**，导航至**数据表管理** > **我管理的表**。
3. 单击对应表后的**生命周期**。

图 4-130: 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 4-131: 修改生命周期弹出框



5. 单击**确定**。

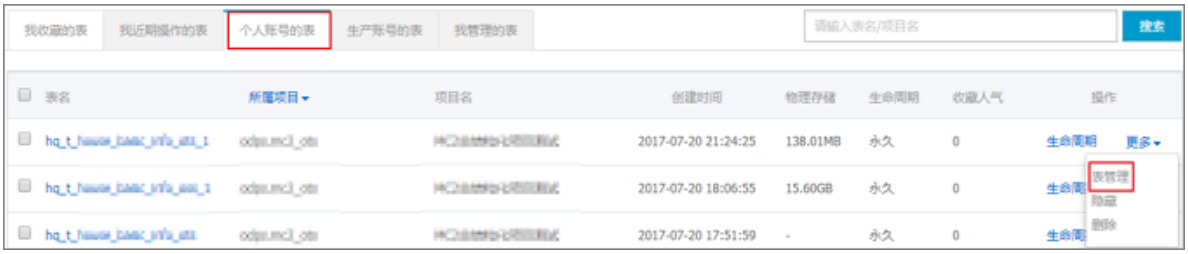
4.6.6.5 修改表结构

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**

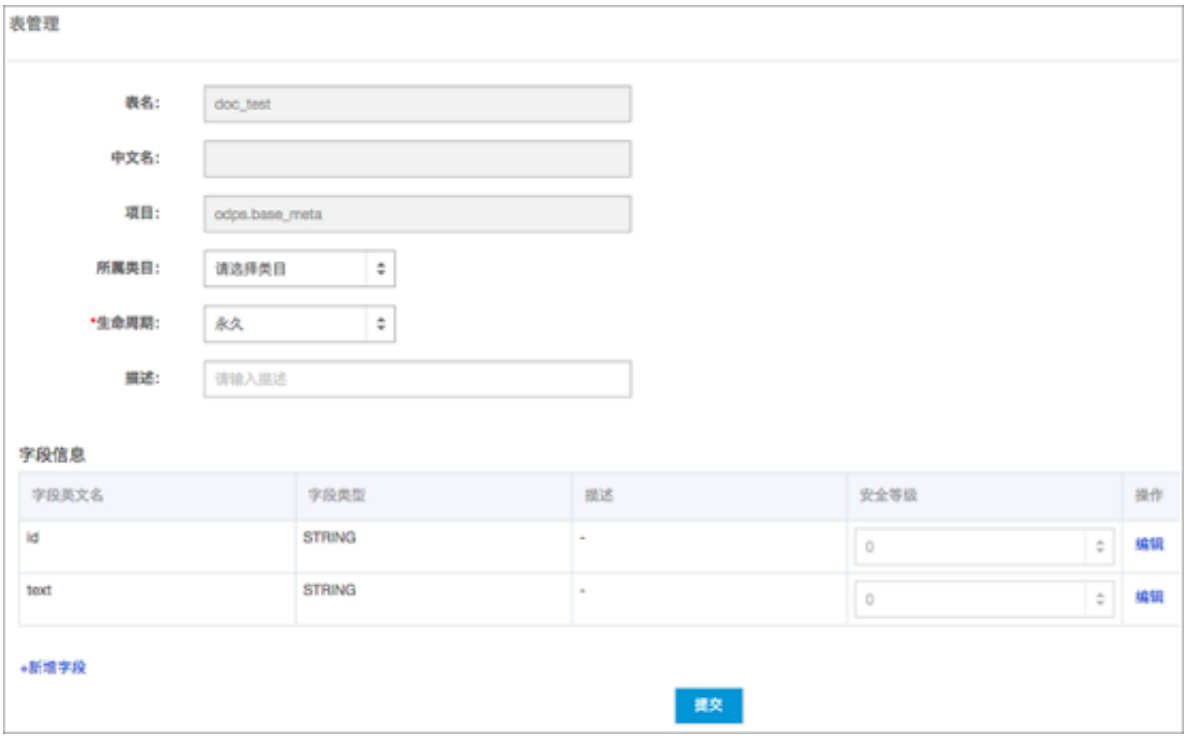
1. 以开发者身份创建项目。
2. 单击顶部菜单栏中的**数据管理**，导航至**数据表管理** > **个人账号的表**。
3. 单击**更多** > **表管理**。

图 4-132: 个人账号的表



4. 在**表管理**页面中，修改相关信息。

图 4-133: 修改个人账号表结构

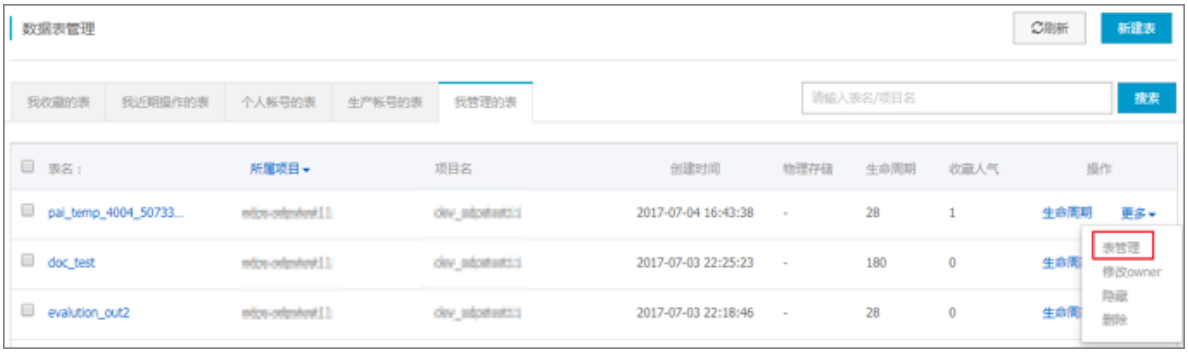


5. 单击**提交**。

场景二：修改我管理的表

1. 以管理员身份创建项目。
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 4-134: 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**。

4.6.6.6 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. 以开发者身份创建项目。
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 4-135: 隐藏个人账号的表

被隐藏的表名后将有隐藏标识：

图 4-135: 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. 以管理员身份创建项目。
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 4-137: 隐藏我管理的表

被隐藏的表名后将有隐藏标识。

图 4-138: 表隐藏标识

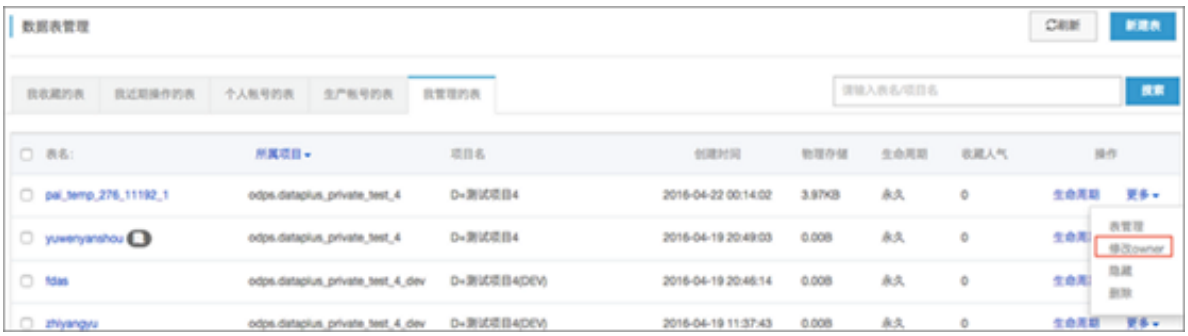
4.6.6.7 修改表负责人

项目管理员可修改表负责人。

操作步骤

1. 以管理员身份创建项目。
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 4-139: 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 4-140: 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**。

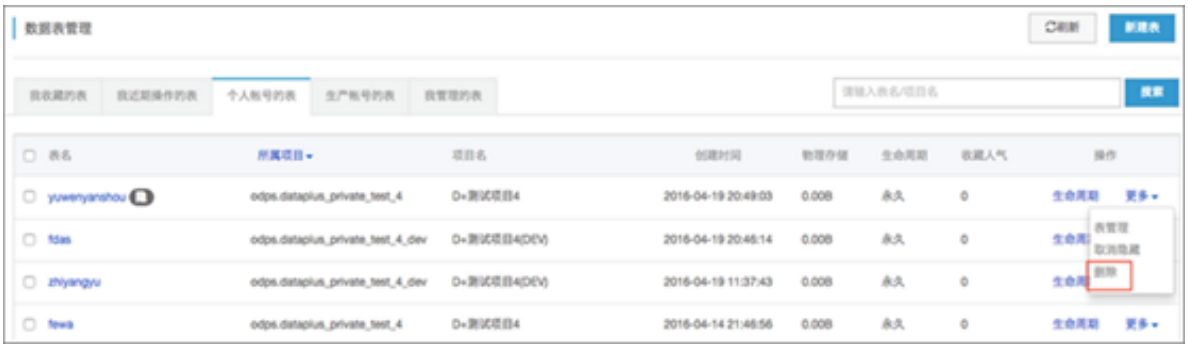
4.6.6.8 删除表

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. 以开发者身份创建项目。
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 4-141: 删除个人账号的表



4. 单击**确定**。

图 4-142: 确认删除

场景二：**删除我管理的表**

1. 以管理员身份创建项目。
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 4-143: 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。

4.6.7 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。

4.6.7.1 待我审批

概述

待我审批子模块以列表的形式展现开发角色用户的数据表、资源和自定义函数权限申请信息，项目管理员可在此页面中针对数据表权限申请进行单个或批量审批操作，包括审批通过和驳回操作。

应用示例

假设开发角色的用户A一次性申请了五张数据表权限，用于在开发过程中进行测试，如表 *4-3:* 待我审批列表所示，项目管理员B需对其所属项目空间的数据表权限申请信息进行审批通过和驳回操作。

说明：

同一个项目空间可具备多名项目管理员，都可对**待我审批**列表中的申请记录进行授权。

表 4-3: 待我审批列表

数据表名称	所属项目	申请人	是否代申请/对方账号
table1	通用版测试111_开发	cloud1	是 / cloud2
table2	通用版测试111_开发	cloud1	否 / -
table3	通用版测试111_开发	cloud1	否 / -
table4	通用版测试111_开发	cloud1	是 / cloud2
table5	通用版测试111_开发	cloud1	是 / cloud3

操作步骤

项目管理员B处理**待我审批**列表中的事件，具体操作如下：

1. 以项目管理员身份创建项目。
2. 单击顶部切换到数据表所属的项目空间下。

图 4-144: 进入数据表所属项目空间

3. 单击顶部菜单**数据管理**，导航至**权限管理** > **待我审批**。
4. 通过搜索框模糊匹配搜索table1。

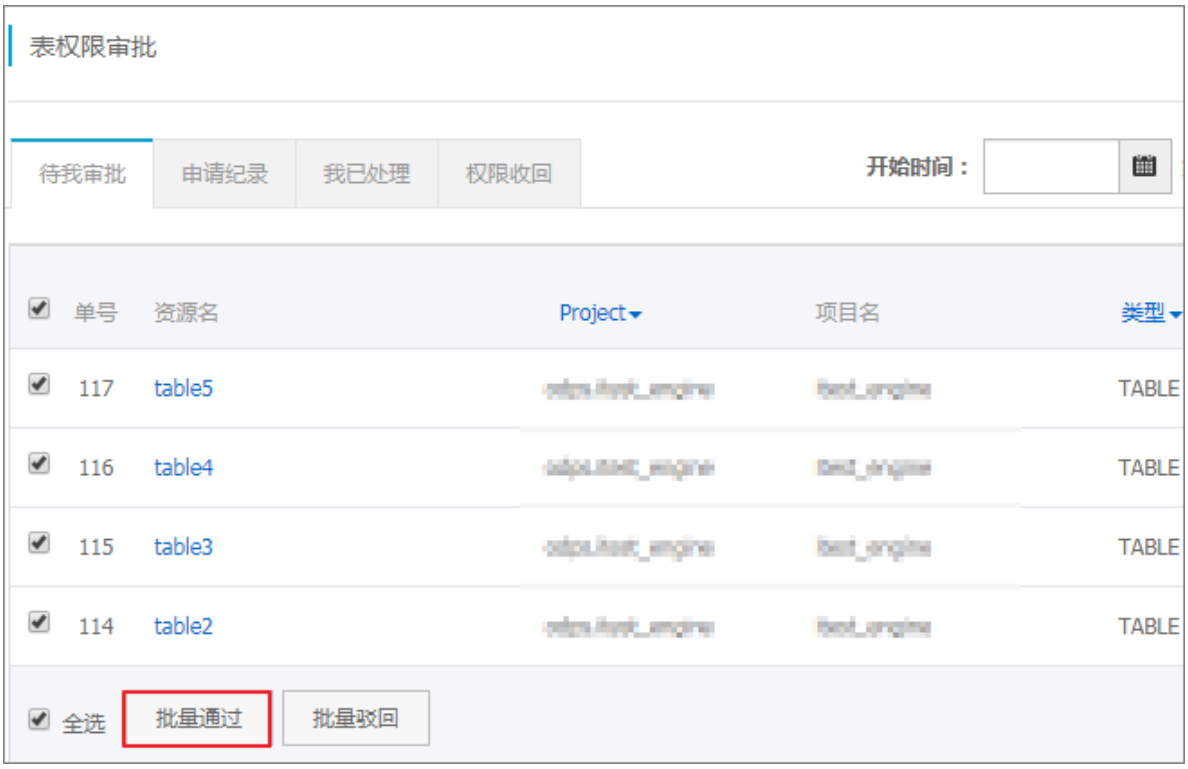
图 4-145: 待我审批列表

5. 单击操作栏中的**驳回**。
6. 在**驳回申请**弹出框中，填写驳回理由。

图 4-146: 驳回申请弹出框

7. 单击**提交**。
8. 勾选其余四张数据表权限申请信息，并单击**批量通过**。

图 4-147: 批量审批操作



9. 在**批量权限审批**对话框中，填写审批理由。

图 4-148: 批量权限审批

10.单击**提交**。

> **说明**：
> 针对代申请的权限审批通过操作，数据表权限最终归属人为被代理人。如表 *4-3:* 待我审批列表中，table4的数据表权限最终归属人为cloud2，table5的数据表权限最终归属人为cloud3。

4.6.7.2 权限收回

概述

权限收回模块以列表的形式展现已审批通过的数据表权限，项目管理员可进行单个或批量勾选记录并进行批量权限收回操作。权限收回后，数据表权限归属人将会立即失去该数据表权限。

应用示例

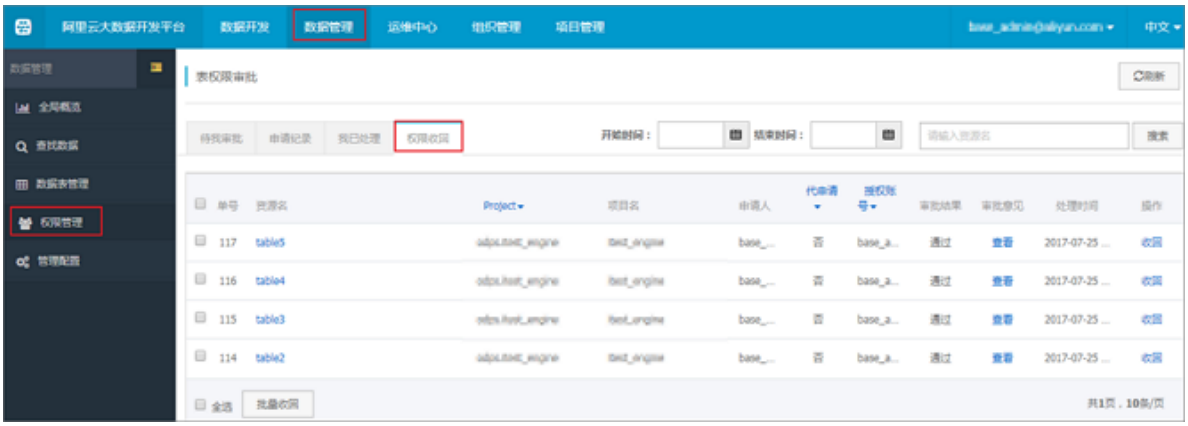
假设由于项目涉密等级提高等因素，需项目管理员B对**待我审批**中已通过权限审批的table2和table3的数据权限进行收回操作，这也就意味着申请人cloud1将立即失去对该数据表的权限。

操作步骤

项目管理员B收回已授权的数据表权限，具体操作如下：

1. 以项目管理员身份创建项目。
2. 单击顶部切换到数据表所属的项目空间下。
3. 单击顶部菜单**数据管理**，导航至**权限管理** > **权限收回**。

图 4-149: 权限收回页面



4. 勾选table2和table3两条记录，并单击**批量收回**。

图 4-150: 勾选需披批量收回的记录

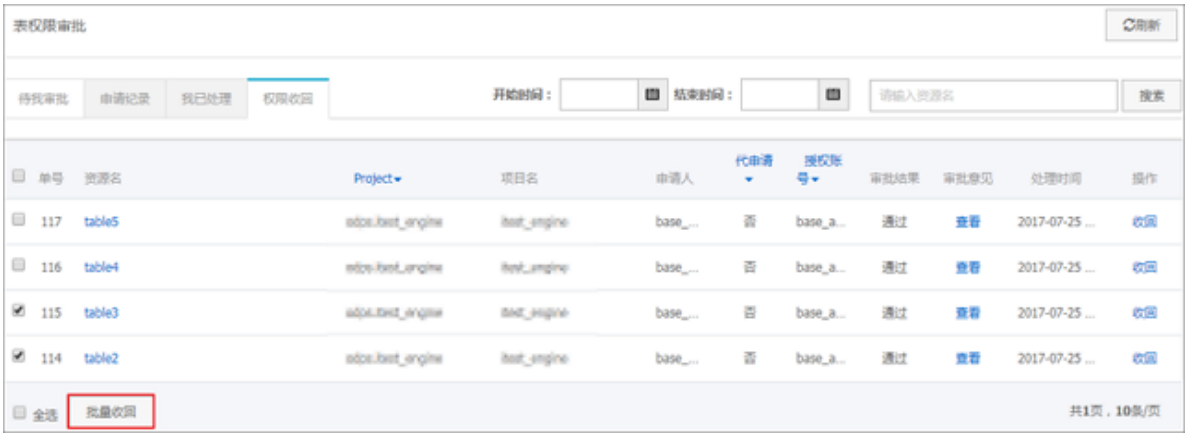


图 4-151: 收回权限对话框

5. 单击**提交**。

4.6.7.3 我已处理

概述

我已处理模块以列表形式展现项目管理员已处理过的事件，包括审批通过、审批驳回、权限回收的记录。同时也查看具体某条记录的审批意见。

应用示例

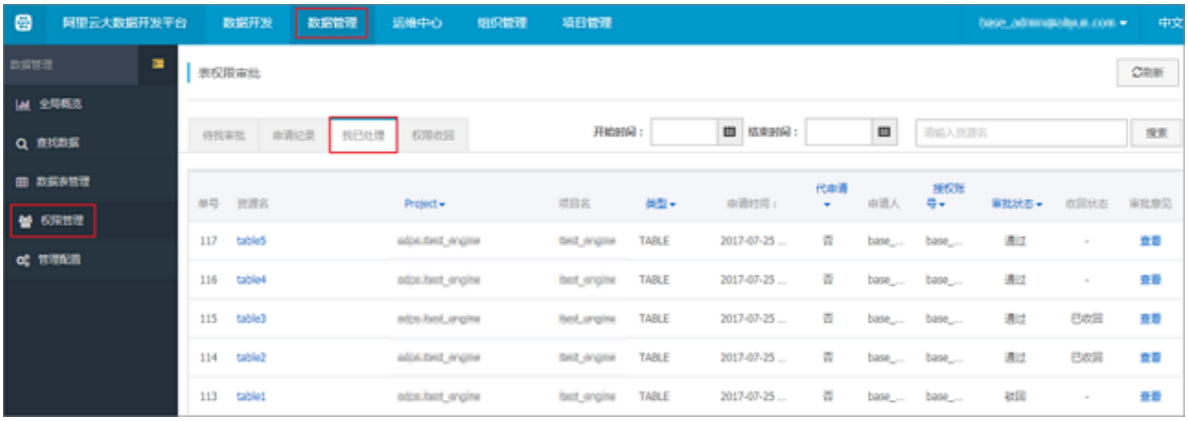
假设项目管理员B需要查看自己已处理的权限审批历史记录，并对待我审批中已通过权限审批的数据表table4进行审批意见的查看。

操作步骤

项目管理员B完成历史记录查询与批量权限收回操作，具体操作如下：

1. 以项目管理员的身份创建项目。
2. 单击顶部切换到数据表所属的项目空间下。
3. 单击顶部菜单**数据管理**，导航至**权限管理** > **我已处理**。

图 4-152: 我已处理页面



4. 单击table4记录操作栏中的**查看**。

图 4-153: 审批意见查看

5. 单击**确定**。

4.6.8 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。

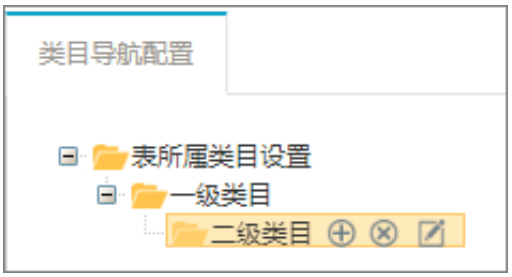
操作步骤

1. 登录*DataWorks管理控制台*。
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的 ，添加一级类目。

图 4-154: 新建一级类目

4. 单击一级类目后的 ，添加所属二级类目。

图 4-155: 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 *4-156: 类目导航配置*所示。

图 4-156: 类目导航配置

图 4-157: 新建表-所属类目

4.7 运维中心

开发者在DataWorks控制台的**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

4.7.1 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。分为概览、任务管理和监控报警三个模块。

- **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。

4.7.2 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。

4.7.2.1 角色的权限划分

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 4-4: 运维中心模块权限划分

父权限名称	权限名称	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	√	√	√
	查看节点定义-在DAG图中可查看节点的上下游	√	√	√
	查看节点列表	√	√	√
	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	√	√	√
	查看节点详情	√	√	√
	查看节点实例-在DAG图中可查看节点的上下游	√	√	√
	实例-暂停	√	√	√
	实例-恢复	√	√	√
	实例-杀任务	√	√	√
	实例-批量杀任务	√	—	√
	实例-列表	√	√	√

父权限名称	权限名称	项目管理员	开发	运维
	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	√	—	√
	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	√	√	√

父权限名称	权限名称	项目管理员	开发	运维
	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	√	—	√
	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	√	√	√

4.7.2.2 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。

4.7.2.3 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。

4.7.2.4 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。

4.7.2.5 如果您是一个项目管理员

您在运维中心模块中拥有该项目所有操作的权限。

4.7.3 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、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。

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。

修改调度资源

调度资源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。

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。

选中的任务可以选择修改调度资源，设置好后，该任务将不会再系统默认的资源中执行，会被分发至自购的ECS中执行。

4.7.3.1 任务管理视图

登录*DataWorks*控制台，导航至**运维中心 > 任务管理 > 任务管理视图。**

图 4-158: 任务管理视图

您在该页面可查看任务运行的详情。右键单击任务名称，可进行编辑任务、测试任务和补数据任务等操作。

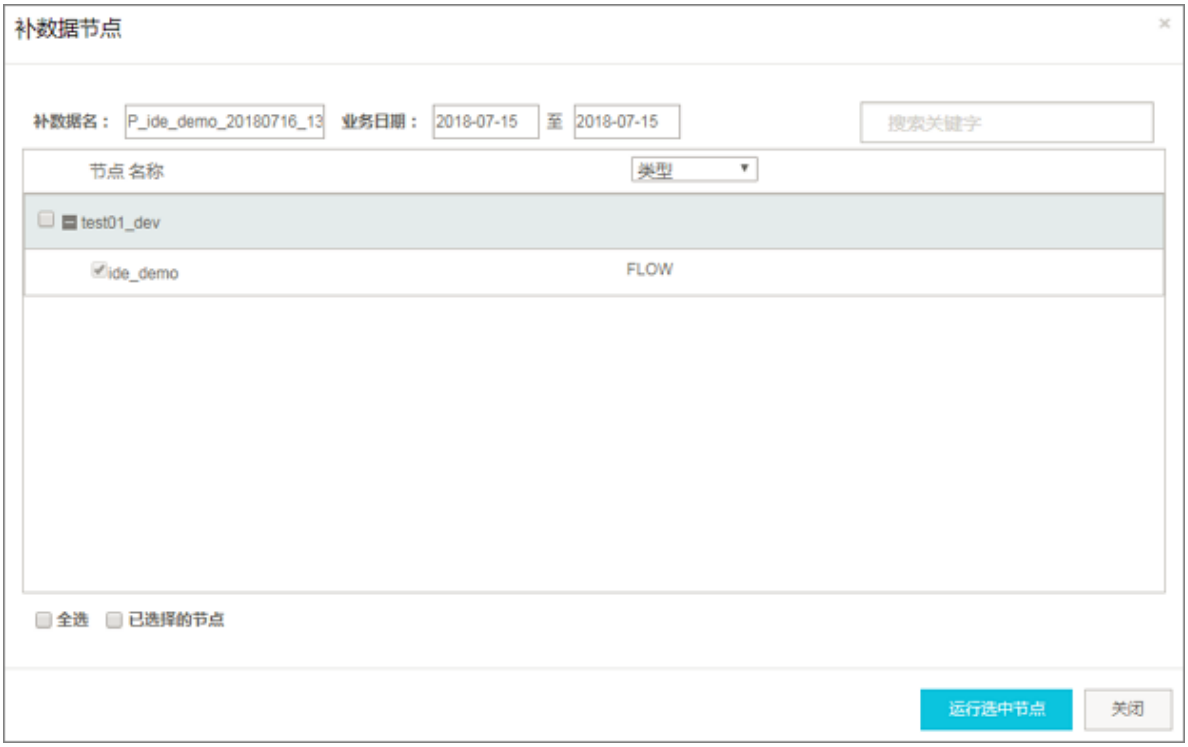
图 4-159: 任务DAG图

- **展开父任务/子任务**：当一个工作流有3个节点及以上时，运维中心展示任务时会自动隐藏节点，您可通过展开父子层级，来看到全部节点的内容。
- **查看任务操作日志**：记录您对该任务进行的操作，例如终止运行，重跑等。
- **查看任务属性**：查看任务属性，包括任务运行的各种时间信息、运行状态等。
- **编辑任务**：单击**编辑任务**，即可进入任务开发页面进行编辑操作。
- **测试任务**：单击测试任务，可对该任务进行测试运行。

图 4-160: 测试运行

- **补数据任务**：补数据可以对工作流/节点操作执行发生在过去的一段时间的调度。

图 4-161: 补数据任务



单击**运行选择节点**后会生产补数据节点。

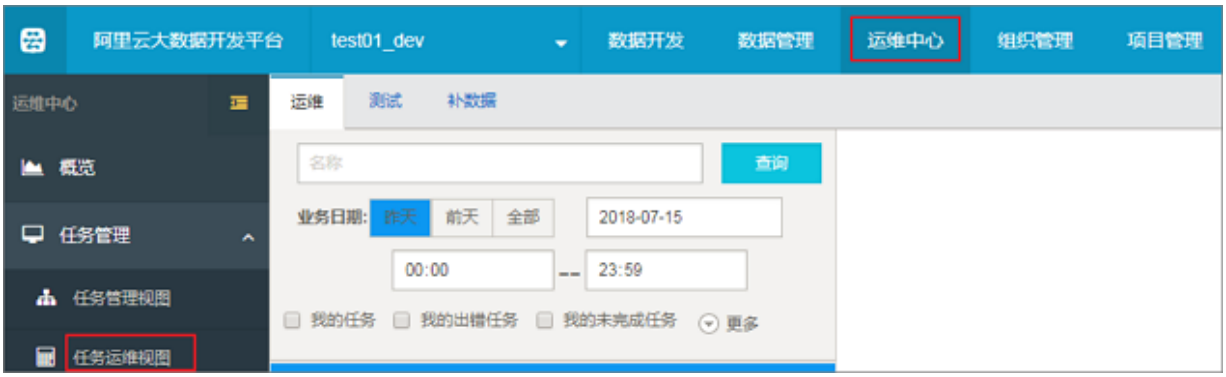
图 4-162: 补数据节点

单击前往查看补数据结果，即可跳转到**补数据**模块，并定位到具体的任务。

4.7.3.2 任务运维视图

登录*DataWorks控制台*，导航至**运维中心** > **任务管理** > **任务运维视图**。

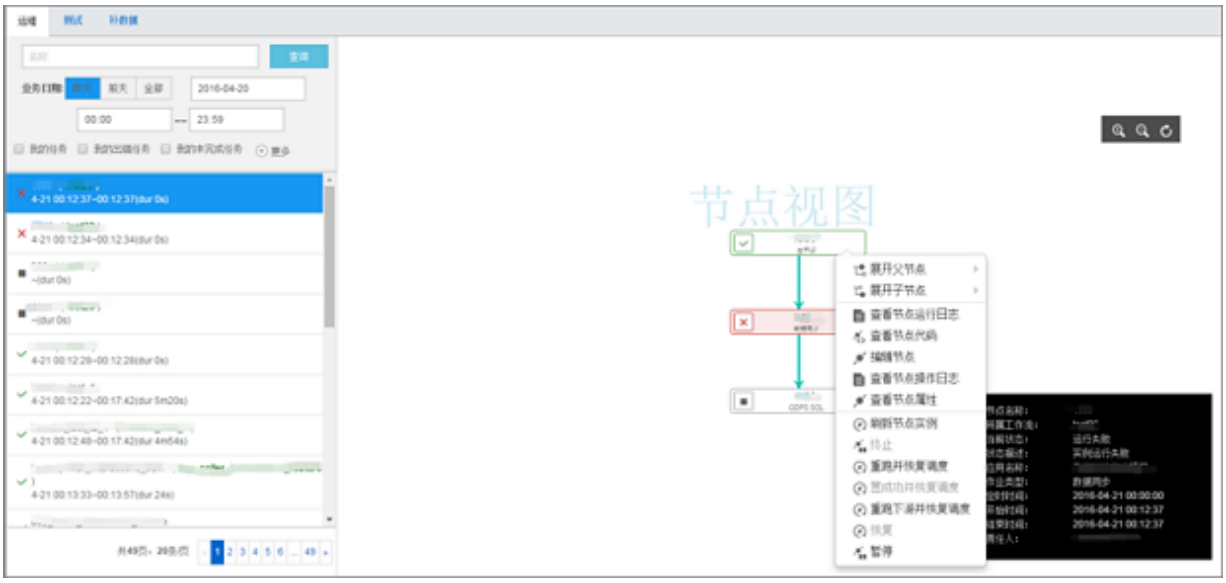
图 4-163: 任务运维视图



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 *4-164: 节点视图运维*所示。

图 4-164: 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 4-5: 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。

测试

测试是指将选中的任务按您配置的开始调度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。

4.7.3.3 任务管理列表

登录*DataWorks控制台*，导航至**运维中心 > 任务管理 > 任务管理列表**。

图 4-165: 任务管理列表

您可在该页面进行修改资源组、修改责任人和添加报警等操作。

- **修改资源组**

  图 4-166: 修改资源组

  勾选需要修改资源组的任务，单击**修改资源组**，选择需要的资源组，单击**确定**。

  > **说明**：
  > 工作流类型的任务不支持修改资源组。

- **修改责任人**

图 4-167: 修改责任人

勾选需要修改责任人的任务，单击**修改责任人**，选择需要的责任人，单击**确定**。

- **添加报警**

图 4-168: 添加报警

您可对**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。

添加报警时需要设置报警名称、状态、内容、报警方式和接收人等信息。

4.7.3.4 任务运维列表

登录*DataWorks控制台*，导航至**运维中心 > 任务管理 > 任务运维列表**。

图 4-169: 任务运维列表

您可在该页面进行批量杀任务、批量修复重跑任务等操作。

4.7.4 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。

4.7.4.1 报警记录

登录*DataWorks控制台*，导航至**运维中心 > 监控报警 > 报警记录**。

图 4-170: 报警记录

您可在该页面查看报警任务的时间、工作流、任务、报警方式、接收人和状态等信息。

4.7.4.2 自定义报警

登录*DataWorks控制台*，导航至**运维中心 > 监控报警 > 自定义提醒**。

图 4-171: 自定义提醒

您可在该页面对报警任务进行修改、删除和关闭等自定义操作。

4.8 组织管理

4.8.1 项目管理

登录*DataWorks控制台*，导航至**组织管理** > **项目管理**页面，在该页面您可以对项目空间进行新建、编辑、禁用/激活等操作，还可以进行更多配置，更多详情请参见项目管理。

4.8.1.1 新建项目空间

前提条件

新建项目空间需先新建计算引擎来完成MaxCompute project的初始化，其次在新建项目空间页面中来绑定计算引擎（一个计算引擎只能被一个项目空间所引用）。

概述

新建项目空间隶属于组织管理模块下，仅限于组织管理员才有权限创建项目空间。所以在新建项目空间前，要确保已在专有云控制台中创建了组织，并指定组织管理员。

新建项目空间时，系统提供预设模板可供组织管理员进行选择创建开发、测试、预发、生产等项目环境中的一种或多种，并可自动生成项目之间的关联关系。以此可见，组织与项目空间之间是**一对多**的关系，即同一个组织下可包含多个项目空间。

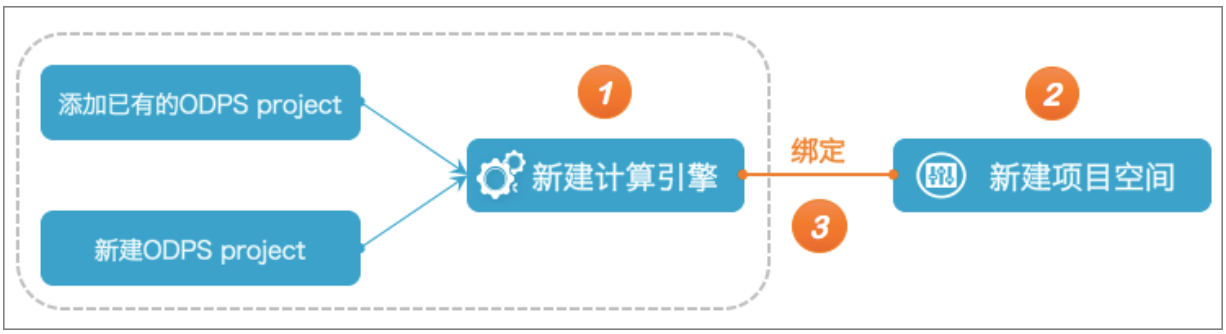
系统预设模板主要分为单个和联合两种类型。

- 开发项目（单个）
- 生产项目（单个）
- 开发+生产（联合）
- 开发+测试+生产（联合）
- 开发+测试+预发+生产（联合）

- 其他类型项目（单个）

新建项目空间的总体流程如图 *4-172: 新建项目空间总体流程图*所示：

图 4-172: 新建项目空间总体流程图



应用示例

- 假设由于**网络日志分析项目**的需求及其它因素，组织管理员需创建**开发环境**的项目空间，用以项目成员进行数据开发工作；
- 假设由于**网络日志分析项目**的需求及其它因素，组织管理员需创建**生产环境**的项目空间，用以数据任务的日常自动调度与生产；
- 假设由于**网络日志分析项目**的需求及其它因素，组织管理员需一次性创建可供数据开发的**开发环境**和用于日常调度的**生产环境**的项目空间；
- 假设由于**网络日志分析项目**的需求及其它因素，组织管理员需创建**开发环境**、**测试环境**和**生产环境**的项目空间，分别用于项目成员进行数据开发、测试以及日常调度生产使用；
- 假设由于**网络日志分析项目**的需求及其它因素，组织管理员需创建**开发环境**、**测试环境**、**预发环境**和**生产环境**的项目空间，分别用于项目成员进行数据开发、测试、生产前预发及日常调度生产使用；
- 假设由于**网络日志分析项目**的需求及其特殊因素，组织管理员需**自定义创建项目空间**可供项目成员使用的环境。

操作步骤

- **创建单个开发环境的项目空间**

  组织管理员创建开发环境的项目空间，具体操作如下。

  1. 以组织管理员身份登录DataWorks。
  2. 单击顶部菜单中的**组织管理** > **项目管理**。
  3. 单击**新建项目空间**。

4. 在**新建项目空间**对话框中选择**开发项目（单个）**模板；

图 4-173: 新建项目-选择开发项目（单个）模板

5. 在**新建项目空间**页面中配置开发项目的各信息项；

图 4-174: 配置开发项目（单个）

开发项目（单个）模板配置项说明如下。

- **项目名称**：项目名称需在组织内全局唯一，且创建后不支持修改。同时建议项目名称中带有开发或dev等关键字的后缀（不区分大小写）。
- **项目标识**：由字母、数字、下划线组成的项目空间的唯一标识符，需在组织内全局唯一，且创建后不支持修改。建议项目标识符中带有dev关键字的后缀。
- **项目描述**：针对当前项目空间的文字描述。
- **项目管理员**：可指定一个或多个项目管理员，默认为当前用户。
- **计算引擎**：从下拉列表中选择一个已创建的MaxCompute计算引擎，若没有，请进入BCC进行新建。

  说明：
  同一个MaxCompute计算引擎不能被多个项目空间所绑定。
- **发布目标项目**：需指定一个测试项目或生产项目作为发布目标项目，支持关键字模糊匹配本组织下的项目名称。创建成功后支持修改，详细操作请参见基础属性配置。

- **是否允许在本项目中直接编辑工作流和代码**：默认允许在本项目中直接编辑工作流和代码。该配置成功后**不支持**修改。

  说明：
  开发项目默认为允许在本项目中直接编辑工作流和代码，且该配置项成功后不支持修改，否则您无法编辑本项目空间中的工作流和代码。

- **是否启用周期性调度与自动执行工作流**：开发项目默认不勾选。勾选该选项则表示本项目空间中的工作流将自动调度执行。**开发环境的项目空间建议无需启用周期性调度。**支持创建后修改，详细操作请参见流程控制。

6. 勾选**我已查看每个项目的默认配置信息，确认符合预期**前的复选框，并单击**创建项目**。

   说明：
   若新建项目空间时，未指定**发布目标项目**，单击**创建项目**后会弹出提示框，单击**继续创建项目**即可。

图 4-175: 未指定发布目标项目提示框

- **创建单个生产环境的项目空间**

1. 以组织管理员身份登录DataWorks。
2. 单击顶部菜单中的**组织管理** > **项目管理**。
3. 单击**新建项目空间**。
4. 在**新建项目空间**对话框中选择**生产项目**（**单个**）模板。

图 4-176: 新建项目-选择生产项目（单个）模板

5. 在**新建项目空间**页面中配置生产项目的各信息项。

图 4-177: 配置生产项目（单个）

生产项目（单个）模板配置项说明如下：

- **项目名称**：项目名称需在组织内全局唯一，且创建后不支持修改。同时建议项目名称中带有生产或prd/prod/product/production等关键字的后缀（不区分大小写）。
- **项目标识**：由字母、数字、下划线组成的项目空间的唯一标识符，需在组织内全局唯一，且创建后不支持修改，建议项目标识符中带有prod关键字的后缀。
- **项目描述**：针对当前项目空间的文字描述。
- **项目管理员**：可指定一个或多个项目管理员，默认为当前用户。
- **计算引擎**：从下拉列表中选择一个已创建的MaxCompute计算引擎，若没有，请进入BCC进行新建。
- **发布目标项目**：默认为空，且不支持指定目标项目。
- **是否允许在本项目中直接编辑工作流和代码**：建议不选中，选中该选项且保存后不可修改。

说明：

通常情况下，生产环境的工作流和代码是从开发、测试或预发环境发布而来的，因此生产环境的项目空间建议不允许使用编辑器直接修改生产环境下的工作流和代码。

- **是否启用周期性调度**：默认选中状态。创建生产项目建议启用周期性调度，这样任务将按照配置进行自动调度执行。支持创建后修改，详情请参见流程控制。

6. 勾选**我已查看每个项目的默认配置信息，确认符合预期**前的复选框，并单击**创建项目**。

- **创建开发环境和生产环境的项目空间**

组织管理员一次性创建开发环境和生产环境的项目空间，具体操作如下：

1. 以组织管理员身份登录DataWorks。
2. 单击顶部菜单中的**组织管理** > **项目管理**。
3. 单击**新建项目空间**。
4. 在**新建项目空间**对话框中选择**开发+生产**模板。

图 4-178: 新建项目-选择开发+生产模板

5. 在**新建项目空间**页面中配置生产项目的各信息项。

图 4-179: 配置开发+生产项目

> **说明**：
> 选择开发和生产项目模板，其开发项目的**发布目标项目**选项将会自动绑定生产项目，无需组织管理员进行手动配置。

6. 勾选**我已查看每个项目的默认配置信息**，**确认符合预期**前的复选框，并单击**创建项目**。

- **创建开发、测试和生产环境的项目空间**

  组织管理员一次性创建开发、测试和生产环境的项目空间，具体操作如下：

  1. 以组织管理员身份登录DataWorks。
  2. 单击顶部菜单中的**组织管理** > **项目管理**。
  3. 单击**新建项目空间**。
  4. 在**新建项目空间**对话框中选择**开发+测试+生产**模板。

  图 4-180: 新建项目-选择开发+测试+生产模板

  5. 在**新建项目空间**页面中配置开发、测试和生产项目的各信息项。

  图 4-181: 配置开发+测试+生产项目

说明：

当填写开发项目名称和标识时，测试项目、生产项目的名称和标识将自动产生，各环境发布目标项目也自动绑定，无需管理员进行手动配置。

关于**测试项目**中的有关配置项说明如下：

- **项目名称**：项目名称需在组织内全局唯一，且创建后不支持修改。同时建议项目名称中带有**测试**或`test`等关键字的后缀（不区分大小写）。
- **项目标识**：由字母、数字、下划线组成的项目空间的唯一标识符，需在组织内全局唯一，且创建后不支持修改，建议项目标识符中带有`test`关键字的后缀。

其他配置项的要求与开发、生产项目相同，此处不再叙述。

6. 勾选**我已查看每个项目的默认配置信息**，**确认符合预期**前的复选框，并单击**创建项目**。

- **创建开发、测试、预发和生产环境的项目空间**

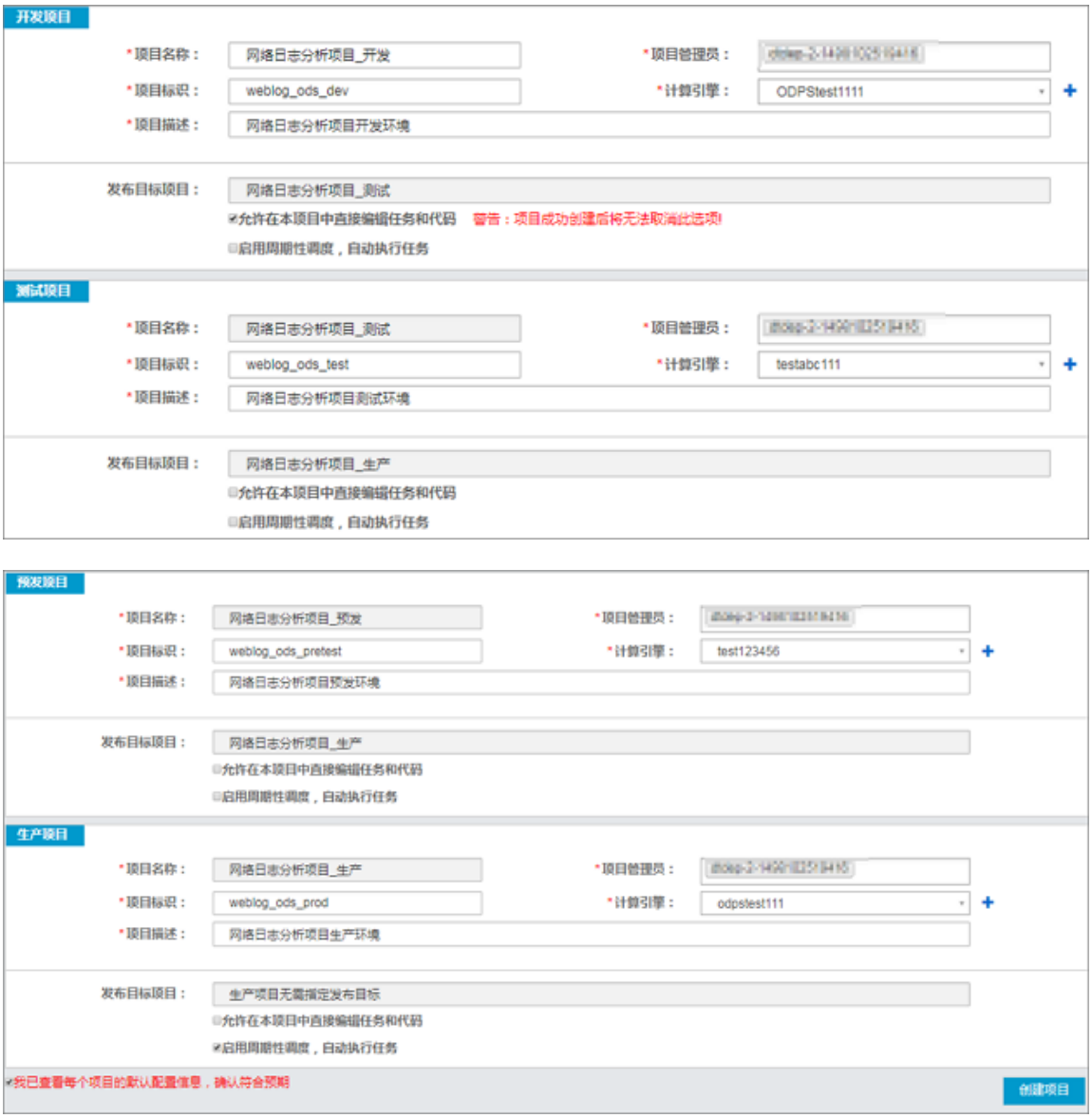
组织管理员一次性创建开发、测试、预发和生产环境的项目空间，具体操作如下：

1. 以组织管理员身份登录DataWorks。
2. 单击顶部菜单中的**组织管理** > **项目管理**。
3. 单击**新建项目空间**。
4. 在**新建项目空间**对话框中选择**开发+测试+预发+生产**模板。

图 4-182: 新建项目-选择开发+测试+预发+生产模板

5. 在**新建项目空间**页面中配置开发、测试、预发和生产项目的各信息项。

图 4-183: 配置开发+测试+预发+生产项目



说明：

当填写开发项目名称和标识时，测试项目、生产项目的名称和标识将自动产生，各环境发布目标项目也自动绑定，无需管理员进行手动配置。

关于**预发项目**中的有关配置项说明如下：

- **项目名称**：项目名称需在组织内全局唯一，且创建后不支持修改。同时建议项目名称中带有预发或pretest等关键字的后缀（不区分大小写）。
- **项目标识**：由字母、数字、下划线组成的项目空间的唯一标识符，需在组织内全局唯一，且创建后不支持修改，建议项目标识符中带有pretest关键字的后缀。

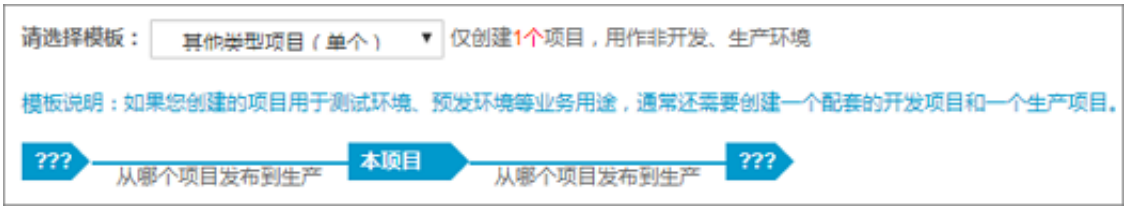
其他配置项的要求与开发、测试、生产项目相同，此处不再叙述。

6. 勾选**我已查看每个项目的默认配置信息，确认符合预期**前的复选框，并单击**创建项目**。

- **创建其他类型环境的项目空间**

1. 以组织管理员身份登录DataWorks。
2. 单击顶部菜单中的**组织管理** > **项目管理**。
3. 单击**新建项目空间**。
4. 在**新建项目空间**对话框中选择**其他类型项目（单个）**模板。

图 4-184: 新建项目-选择其他类型项目（单个）模板



5. 在**新建项目空间**页面中配置生产项目的各信息项。

图 4-185: 配置其他类型项目（单个）

关于其他类型项目中的相关配置项说明如下：

- **项目名称**：项目名称需在组织内全局唯一，且创建后不支持修改。同时根据项目具体用户，建议项目名称中带有开发环境、测试环境、预发环境、生产环境等关键字的后缀，以便项目成员进行识别与管理。
- **项目标识**：由字母、数字、下划线组成的项目空间的唯一标识符，需在组织内全局唯一，且创建后不支持修改，建议项目标识符中带有具体用途的关键字后缀。

其他配置项的要求与开发、测试、生产项目相同，此处不再叙述。

6. 勾选**我已查看每个项目的默认配置信息，确认符合预期**前的复选框，并单击**创建项目**。

4.8.1.2 编辑项目空间

概述

组织管理员成功创建项目空间后，可在**组织管理** > **项目管理**列表中对其基础信息进行修改配置，可编辑的配置项包括项目描述、项目管理员、调度资源组和ODPS访问身份。

应用示例

组织管理需编辑已成功创建的“网络日志分析项目_测试”项目空间的基本信息。

操作步骤

组织管理员编辑已成功创建的，具体操作如下：

1. 以组织管理员身份登录*DataWorks*控制台。
2. 单击顶部菜单中的**组织管理** > **项目管理**。
3. 在项目空间列表搜索空中模糊查询**网络日志分析项目_测试**。
4. 单击对应操作栏中的**编辑**。

图 4-186: 编辑项目

5. 在**编辑项目**弹出框中配置相关基础信息。

图 4-187: 编辑项目弹出框

编辑项目空间弹出框中配置项的说明如下：

- **项目名称**：不支持修改。
- **唯一标识**：不支持修改。
- **项目描述**：可修改，支持中文、英文字母、数字、下划线组合。
- **项目管理员**：可新增多个管理员。
- **调度资源**：可新建调度资源，详情请参见调度资源管理章节。
- **计算引擎**：不支持修改。
- **ODPS访问身份**：支持修改，包括**个人身份**和**计算引擎指定账号**。

> **说明**：
>
> ODPS访问身份选项说明如下：

个人身份：选择个人身份选项，意味着本项目空间中的每一位成员将使用**个人账号**来访问MaxCompute，运行任务时涉及到为每个项目成员独立授权。

计算引擎指定账号：选择默认的计算引擎指定账号，则意味着本项目空间中的所有成员将通过同一个**系统账号**来访问MaxCompute及运行任务。

4.8.1.3 禁用项目空间

概述

组织管理员新建项目空间成功后，该项目空间状态将由**初始化**转变为**正常**。组织管理员也可禁用某项目空间，禁用项目空间后，项目成员立即不能在该项目空间中进行数据开发（如创建工作流、资源，发布包等操作），且该项目空间中的所有工作流将不能被系统自动调度。

同时，针对已禁用的项目空间，组织管理员和项目管理员不能修改该项目的其他属性配置。

应用示例

假设由于项目机器资源紧张等因素，现需组织管理员暂时禁用一部分已激活状态的项目空间，为优先级更高的项目空间让出可用的计算资源和调度资源。

操作步骤

以禁用**网络日志分析项目_测试**为例，具体操作如下。

1. 以组织管理员身份登录*DataWorks*控制台。
2. 单击顶部菜单中的**组织管理** > **项目管理**。
3. 通过搜索框模糊查询需禁用的项目空间，并单击项目空间操作栏中的**禁用**。

图 4-188: 项目列表

4. 单击禁用提醒对话框中的**确定**。

图 4-189: 禁用提醒对话框

4.8.1.4 激活项目空间

概述

只有组织管理员才可进行将**禁用**置为**正常**状态的操作。只有在已激活状态的项目空间中，项目成员才可进行数据开发且工作流才可被系统自动调度，否则系统提示“该项目空间已被禁用，请联系组织管理”。“激活项目空间”只针对项目空间的状态进行修改，不附带其他操作。

应用示例

假设组织管理员已成功创建了开发和生产环境的项目空间，现需将其状态进行激活并提供给项目成员进行数据开发。

操作步骤

以激活**网络日志分析项目_测试**为例，具体操作如下。

1. 以组织管理员身份登录DataWorks。
2. 单击顶部菜单中的**组织管理** > **项目管理**。
3. 通过搜索框模糊查询需激活的项目空间，并单击项目空间操作栏中的**激活**。

图 4-190: 项目列表

4. 单击激活提醒对话框中的**确定**。

图 4-191: 激活提醒对话框

4.8.2 成员管理

登录*DataWorks*控制台，导航至**组织管理** > **成员管理**页面，在该页面您可以查看项目空间的成员信息。

说明：
仅组织管理员可新增或删除组织成员。

4.8.2.1 新增组织成员

您可根据准备工作中的操作，新增组织成员。

4.8.2.2 删除组织成员

概述

组织管理员可将已有成员移出本组织，被移出组织的成员将无法继续使用本组织内的所有资源，也将不能正常使用DataWorks的相关服务。

应用示例

假设由于项目成员离职，现需组织管理员将其从组织成员列表中删除以收回该成员在该组织下所有项目空间中的权限。

操作步骤

组织管理员移出成员，具体操作如下：

1. 以组织管理员身份创建项目。
2. 单击顶部菜单中的**组织管理** > **成员管理**。

3. 在**组织管理** > **成员管理**列表的搜索框中，输入成员名称或登录名称来模糊匹配查找需要移出组织的成员.
4. 单击操作栏中的**移出本组织**。

图 4-192: 成员列表

5. 单击提醒对话框中的**确定**。

图 4-193: 移出提醒对话框

4.8.3 调度资源

登录*DataWorks*控制台，导航至**组织管理** > **调度资源**中，您可新建、配置、编辑调度资源。

关于调度资源

新建调度资源

配置调度资源

编辑调度资源

4.8.3.1 关于调度资源

调度资源属于组织范畴内的对象，一个专用的调度资源可以拥有若干台物理机或ECS虚机，用于运行指定的任务。组织管理员可在**调度资源**页面中新建、配置调度资源，以完成专用调度资源的创建，也可对已有的调度资源进行编辑。

图 4-194: 调度资源

4.8.3.2 新建调度资源

概述

组织管理员可以新建专用的调度资源，以便后续可将专用的调度资源分配给某些项目空间来执行特定的任务。指定的任务可以是数据同步任务、SHELL脚本、MaxCompute SQL脚本，MaxCompute MR等任务。而在未指定专用调度资源的情况下，项目空间内的所有任务都将默认使用系统托管集群中的资源。

应用示例

新建调度资源可解决如下问题：

- 大量任务处于**等待资源状态**：当前项目的任务量达到一定的量值，已有gateway的资源已经不满足业务需求，需要扩大资源，申请添加gateway。
- 项目下有一些特殊的任务（比如某些SHELL脚本）需要在特定的机器上执行，需申请自己的gateway作为任务执行机器。

现假设由于具体的数据开发项目等因素，组织管理员需创建专用的调度资源以分配给特殊需求的项目空间来运行指定的数据同步任务。

操作步骤

组织管理员新建调度资源，具体操作如下：

1. 以组织管理员身份创建项目。
2. 单击顶部菜单中的**组织管理** > **调度资源**。
3. 单击左上角的**增加调度资源**。
4. 在**新建资源**弹出框中配置**资源名称**和**资源唯一标识**。

图 4-195: 新建资源弹出框

新建调度资源配置项，具体说明如下：

- **资源名称**：由中文、英文字母、下划线、数字组成，不超过60个字符。
- **资源唯一标识**：由字母、数字、下划线组成的资源名称的唯一标识符，需在组织内全局唯一，且创建后不支持修改。

5. 单击**提交**。

4.8.3.3 编辑调度资源

概述

已新建、配置的专用调度资源也可修改其调度资源名称。

应用示例

为项目成员快速分辨与使用专用调度资源，组织管理需对已配置的调度资源进行编辑修改。

操作步骤

组织管理员修改已配置的专用调度资源，具体操作如下：

1. 以组织管理员身份创建项目。
2. 单击顶部菜单中的**组织管理** > **调度资源**。
3. 在**组织管理** > **调度资源**列表搜索框中，输入调度资源名称来进行模糊匹配查找需要编辑的调度资源。
4. 单击**编辑**。

图 4-196: 调度资源列表

5. 在**编辑资源**弹出框中编辑**资源名称**。

图 4-197: 编辑调度资源弹出框

说明：

调度资源名称由中文、英文字母、下划线、数字组成，不能超过60个字符。

6. 单击**提交**。

4.8.3.4 配置服务器

概述

组织管理员成功添加调度资源后，还需通过绑定服务器来完成专用调度资源的配置。专用调度资源绑定服务器不超过1000台，且只能从属于该组织范畴内的ECS机器列表中选取，同时一台ECS虚机只能属于一个自定义调度资源。

应用示例

- 组织管理员已成功添加数据同步服务的专用调度资源但未绑定服务器（新建调度资源），现需ECS机器列表中进行选取添加。

- 假设已配置的调度资源不能够满足日常调度的正常生产，导致大量数据同步任务处于等待资源状态。现需组织管理员对已配置的数据同步服务专用调度资源进行新增配置服务器以满足需求。

操作步骤

1. 单击**配置服务器**。

   图 4-198: 配置服务器

   说明：
   从自己的机器上查找需要的服务器名称。

2. 单击**添加服务器**，填写弹出框中的各配置项。

   图 4-199: 添加服务器

   说明：
   从自己的机器上查找需要的服务器名称。

3. 单击**执行初始化**按钮，按照提示内容进行操作，如图 *4-200: 执行初始化*所示。

图 4-200: 执行初始化

4.8.4 计算引擎

目前，阿里云大数据平台仅支持MaxCompute计算引擎，项目成员创建的工作流、代码都将运行在项目空间所绑定的MaxCompute上。

4.8.4.1 配置计算引擎

概述

组织管理员可对已成功配置的计算引擎进行编辑、修改。支持修改的配置项包括：计算引擎描述、是否用MaxCompute owner账号运行MaxCompute任务、运行MaxCompute任务账号及AK信息。

应用示例

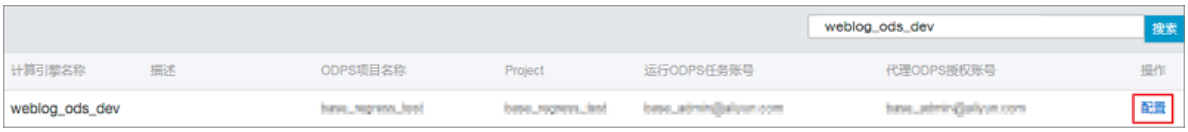
假设由于运行MaxCompute任务账号对应的项目成员离职而导致引用该计算引擎的项目空间中所有任务执行（计算引擎配置中未勾选了**用ODPS owner账号运行ODPS任务**），现需组织管理员及时修改运行MaxCompute任务账号及AK信息。

操作步骤

组织管理员修改运行MaxCompute任务账号及AK信息，具体操作如下。

1. 以组织管理员身份创建项目。
2. 单击顶部菜单中的**组织管理** > **计算引擎**。
3. 在**组织管理** > **计算引擎**列表搜索框中输入计算引擎名称模糊匹配查找受影响的计算引擎。
4. 单击需更改的计算引擎操作栏中的**配置**。

图 4-201: 计算引擎列表



5. 在**配置计算引擎**弹出框中编辑输入正确的**运行ODPS任务账号**及相应的AK信息。

图 4-202: 配置计算引擎弹出框

说明：

组织管理员针对此场景，也可通过勾选**用ODPS owner账号运行ODPS任务**方式，或通过创建新**运行ODPS任务账号**的方式来快捷完成计算引擎的修改配置。

6. 单击**提交**。

4.9 项目管理

项目管理员可以通过阿里云大数据平台的管理控制台套件来完成项目管理操作，主要包括项目属性管理和成员管理。本章节将以**项目管理**视图来详细阐述各部分的应用实例与具体操作，关于组织管理视图下的实例与操作，详见组织管理”。

4.9.1 项目属性

在阿里云大数据平台的**项目管理** > **项目属性**中，您可对当前项目空间的基本属性信息进行配置与管理。

基本属性配置

数据源配置

计算引擎配置

流程控制

4.9.1.1 配置基本属性

概述

项目基本属性信息包括项目名称、项目标识、默认调度资源、发布目标、项目描述和项目管理，其中默认调度资源、项目描述和发布目标可修改，其他配置项在此页面只能查看不可编辑。

应用示例

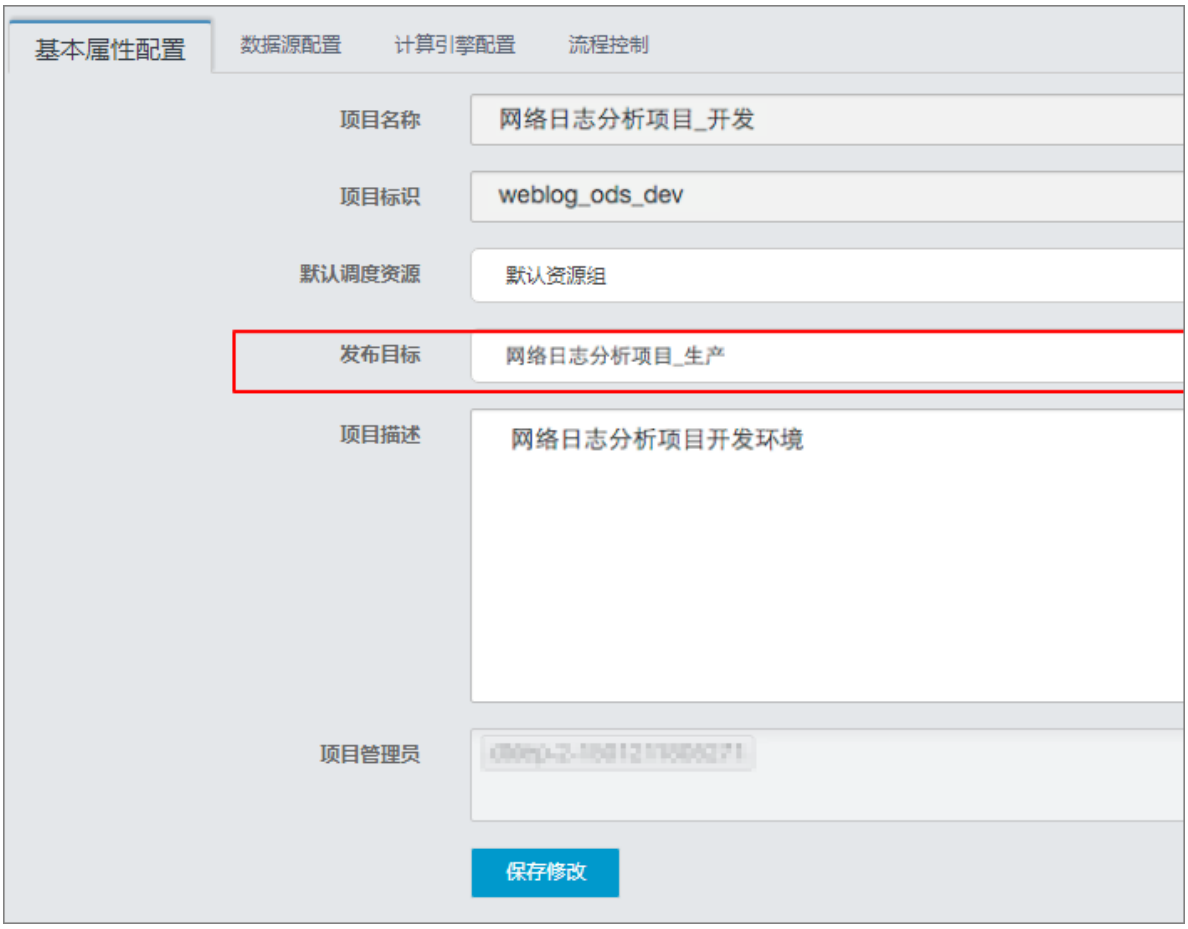
假设由于项目因素，项目管理员需查看项目基本属性信息并修改默认调度资源、项目描述和发布目标。

操作步骤

项目管理员修改默认调度资源、项目描述以及发布目标等，具体操作如下：

1. 以项目管理员身份创建项目。
2. 单击顶部菜单中的**项目管理** > **基本属性配置**。
3. 在**基本属性配置**页面中，修改默认调度资源、项目描述和发布目标。

图 4-203: 基本属性配置



项目基本属性配置页面中可修改的配置项说明如下：

- **默认调度资源**：默认为**默认调度资源**。可从默认调度资源和已分配给本项目空间的调度资源（详情请参见调度资源管理）中选择一个作为本项目的默认值。
- **项目描述**：对项目的简单描述，不超过1024个字符。
- **发布目标**：默认为空。可从下拉表中选择一个作为本项目的发布目标。

4. 单击**保存修改**。

4.9.1.2 配置数据源

4.9.1.2.1 关于数据源

数据源配置是数据集成的首要任务，在进行数据同步（数据导入或数据导出）任务开发时，项目管理员需配置可连通的数据源来支撑整个数据开发项目。

项目管理员可在当前项目空间下进行新建、编辑和删除数据源。目前支持多种数据源类型，具体包括：新建*MaxCompute*数据源、RDS（新建*MySQL*数据源、新建*SQL Server*数据源、新建*PostgreSQL*数据源）、新建*Oracle*数据源、新建*ADS*数据源、新建*OSS*数据源、新建*OCS*数据源、新建*DRDS*数据源、新建*FTP*数据源。

4.9.1.2.2 新增数据源

新建MaxCompute数据源

创建MaxCompute（原名ODPS）数据源，具体操作如下：

1. 登录*DataWorks*控制台。
2. 单击顶部菜单栏中**项目管理**，导航至**项目属性** > **数据源配置**。
3. 单击**新增数据源**。
4. 在**新增数据源**对话框中，选择**数据源类型**为MaxCompute。
5. 配置MaxCompute数据源的各个信息项。

图 4-204: 配置MaxCompute数据源

针对MaxCompute数据源配置项的具体说明如下：

- **数据源名称**：由英文字母、数字、下划线组成且需以字符或下划线开头，长度不超过30个字符。
- **数据源描述**：对数据源的简单描述，不超过1024个字符。
- **数据源类型**：当前选择的数据源类型。
- **ODPS Endpoint**：默认只读。从系统配置中自动读取。
- **ODPS项目名称**：对应的MaxCompute Project标识。
- **以登录用户身份连接**：以当前用户的身份连接MaxCompute Project。若选择该选项，对应的AccessID和AccessKey选项将隐藏，无需配置。
- **以系统身份连接**：以MaxCompute Project Owner身份连接该数据源。
  - **AccessID**：与MaxCompute Project Owner云账号对应的AccessID。
  - **AccessKey**：与MaxCompute Project Owner云账号对应的AccessKey，与AccessID成对使用。

6. 完成上述信息项的配置后，单击**测试连通性**。
7. 测试连通性通过后，单击**确定**。

> **说明**：
> 每个项目空间系统都将生成一个默认的数据源（odps_first），对应的MaxCompute项目名称为当前项目空间对应的计算引擎MaxCompute项目名称。

新建MySQL数据源

创建MySQL数据源，具体操作如下：

1. 登录*DataWorks控制台*。
2. 单击顶部菜单栏中**项目管理**，导航至**项目属性** > **数据源配置**。
3. 单击**新增数据源**。
4. 在**新增数据源**弹出框中，选择**数据源类型**为**RDS>MySQL**。
5. 选择以**RDS实例形式**或**JDBC形式**配置该MySQL数据源。
   - 选择以**RDS实例形式**配置该MySQL数据源。

图 4-205: 以实例形式

上图中的配置项具体说明如下：

- **数据源名称**：由英文字母、数字、下划线组成且需以字符或下划线开头，长度不超过30个字符。
- **数据源描述**：对数据源的简单描述，不超过1024个字符。
- **数据源类型**：当前选择的数据源类型（RDS->MySQL->RDS实例形式）。
- **RDS实例名称**：该MySQL数据源的instanceName名称。
- **数据库名**：该数据源对应的数据库名。
- **用户名/密码**：数据库对应的用户名和密码。

- 选择以**JDBC形式**配置该MySQL数据源。

图 4-206: 以JDBC形式

上图中的配置项具体说明如下：

- **数据源名称**：由英文字母、数字、下划线组成且需以字符或下划线开头，长度不超过30个字符。
- **数据源描述**：对数据源的简单描述，不超过1024个字符。
- **数据源类型**：当前选择的数据源类型（RDS->MySQL->JDBC形式）。
- **JDBCUrl**：JDBC连接信息，格式为：jdbc:mysql://IP:Port/database。
- **用户名/密码**：对应连接信息的用户名和密码。

6. 完成上述信息项的配置后，单击**测试连通性**。
7. 测试连通性通过后，单击**确定**。

新建SQL Server数据源

创建SQL Server数据源，具体操作如下：

1. 登录*DataWorks控制台*。
2. 单击顶部菜单栏中**项目管理**，导航至**项目属性** > **数据源配置**。
3. 单击**新增数据源**。
4. 在**新增数据源**弹出框中，选择**数据源类型**为RDS->SQL Server；

5. 选择以**RDS实例形式**或**JDBC形式**配置该SQL Server数据源。

   - 选择以**RDS实例形式**配置该SQL Server数据源。

     图 4-207: 配置SQL Server数据源

     上图中的配置项具体说明如下：

     - **数据源名称**：由英文字母、数字、下划线组成且需以字符或下划线开头，长度不超过30个字符。
     - **数据源描述**：对数据源的简单描述，不超过1024个字符。
     - **数据源类型**：当前选择的数据源类型（RDS->SQL Server->RDS实例形式）。
     - **RDS实例名称**：该SQL Server数据源的instanceName名称。
     - **数据库名**：该数据源对应的数据库名。
     - **用户名/密码**：数据库对应的用户名和密码。

   - 选择以**JDBC形式**配置该SQL Server数据源。

图 4-208: 以JDBC形式

上图中的配置项具体说明如下：

- **数据源名称**：由英文字母、数字、下划线组成且需以字符或下划线开头，长度不超过30个字符。
- **数据源描述**：对数据源的简单描述，不超过1024个字符。
- **数据源类型**：当前选择的数据源类型（RDSr->SQL Serverr->JDBC形式）。
- **JDBCUrl**：JDBC连接信息，格式为：jdbc:sqlserver://IP:Port;database。
- **用户名/密码**：对应连接信息的用户名和密码。

6. 完成上述信息项的配置后，单击**测试连通性**。
7. 测试连通性通过后，单击**确定**。

新建PostgreSQL数据源

创建PostgreSQL数据源，具体操作如下：

1. 登录*DataWorks控制台*。
2. 单击顶部菜单栏中**项目管理**，导航至**项目属性** > **数据源配置**。
3. 单击**新增数据源**。
4. 在**新增数据源**弹出框中，选择**数据源类型**为RDS->PostgreSQL。

5. 选择以**RDS实例形式**或**JDBC形式**配置该PostgreSQL数据源。

   - 选择以**RDS实例形式**配置该PostgreSQL数据源。

     图 4-209: 以RDS实例形式

     上图中的配置项具体说明如下：

     - **数据源名称**：由英文字母、数字、下划线组成且需以字符或下划线开头，长度不超过30个字符。
     - **数据源描述**：对数据源的简单描述，不超过1024个字符。
     - **数据源类型**：当前选择的数据源类型（RDS->PostgreSQL->RDS实例形式）。
     - **RDS实例名称**：该PostgreSQL数据源的instanceName名称。
     - **数据库名**：该数据源对应的数据库名。
     - **用户名/密码**：数据库对应的用户名和密码。

   - 选择以**JDBC形式**配置该PostgreSQL数据源。

图 4-210: 以JDBC形式

上图中的配置项具体说明如下：

- **数据源名称**：由英文字母、数字、下划线组成且需以字符或下划线开头，长度不超过30个字符。
- **数据源描述**：对数据源的简单描述，不超过1024个字符。
- **数据源类型**：当前选择的数据源类型（RDS->PostgreSQL->JDBC形式）。
- **JDBCUrl**：JDBC连接信息，格式为：jdbc:postgresql//IP:Port/database。
- **用户名/密码**：对应连接信息的用户名和密码。

6. 完成上述信息项的配置后，单击**测试连通性**。
7. 测试连通性通过后，单击**确定**。

新建Oracle数据源

创建Oracle数据源，具体操作如下：

1. 登录*DataWorks控制台*。
2. 单击顶部菜单栏中**项目管理**，导航至**项目属性 > 数据源配置**。
3. 单击**新增数据源**。
4. 在**新增数据源**弹出框中，选择**数据源类型**为Oracle。

5. 配置Oracle数据源的各个信息项。

图 4-211: 配置Oracle数据源

针对Oracle数据源配置项的具体说明如下：

- **数据源名称**：由英文字母、数字、下划线组成且需以字符或下划线开头，长度不超过30个字符。
- **数据源描述**：对数据源的简单描述，不超过1024个字符。
- **数据源类型**：当前选择的数据源类型。
- **JDBCUrl**：JDBC连接信息，格式为：jdbc:oracle:thin:@serverIP:Port:Database。
- **用户名/密码**：对应的用户名和密码。

6. 完成上述信息项的配置后，单击**测试连通性**。
7. 测试连通性通过后，单击**确定**。

> **说明**：
> 每个项目空间系统都将生成一个默认的数据源（odps_first），对应的ODPS项目名称为当前项目空间对应的计算引擎ODPS项目名称。

新建ADS数据源

创建ADS数据源，具体操作如下：

1. 登录*DataWorks控制台*。
2. 单击顶部菜单栏中**项目管理**，导航至**项目属性** > **数据源配置**。
3. 单击**新增数据源**。
4. 在**新增数据源**弹出框中，选择**数据源类型**为ADS。
5. 配置ADS数据源的各个信息项。

图 4-212: 配置ADS数据源

针对ADS数据源配置项的具体说明如下：

- **数据源名称**：由英文字母、数字、下划线组成且需以字符或下划线开头，长度不超过30个字符。
- **数据源描述**：对数据源的简单描述，不超过1024个字符。
- **数据源类型**：当前选择的数据源类型。
- **连接Url**：ADS连接信息，格式为：serverIP:Port。
- **Schema**：相应的ADS Schema信息。
- **AccessID/AceessKey**：对应的用户名和密码，即AK对。

6. 完成上述信息项的配置后，单击**测试连通性**。
7. 测试连通性通过后，单击**确定**。

新建OSS数据源

创建OSS数据源，具体操作如下：

1. 登录*DataWorks控制台*。
2. 单击顶部菜单栏中**项目管理**，导航至**项目属性** > **数据源配置**。
3. 单击**新增数据源**。
4. 在**新增数据源**弹出框中，选择**数据源类型**为OSS。
5. 配置OSS数据源的各个信息项。

图 4-213: 配置OSS数据源

针对OSS数据源配置项的具体说明如下：

- **数据源名称**：由英文字母、数字、下划线组成且需以字符或下划线开头，长度不超过30个字符。
- **数据源描述**：对数据源的简单描述，不超过1024个字符。
- **数据源类型**：当前选择的数据源类型。
- **Endpoint**：OSS Endpoint信息，格式为：http://oss.aliyuncs.com。
- **Bucket**：相应的OSS Bucket信息。
- **AccessID/AceessKey**：对应的用户名和密码，即AK对。

6. 完成上述信息项的配置后，单击**测试连通性**。
7. 测试连通性通过后，单击**确定**。

新建OCS数据源

创建OCS数据源，具体操作如下：

1. 登录*DataWorks控制台*。
2. 单击顶部菜单栏中**项目管理**，导航至**项目属性** > **数据源配置**。
3. 单击**新增数据源**。
4. 在**新增数据源**弹出框中，选择**数据源类型**为OCS。
5. 配置OCS数据源的各个信息项。

图 4-214: 配置OCS数据源

针对OCS数据源配置项的具体说明如下：

- **数据源名称**：由英文字母、数字、下划线组成且需以字符或下划线开头，长度不超过30个字符。
- **数据源描述**：对数据源的简单描述，不超过1024个字符。
- **数据源类型**：当前选择的数据源类型。
- **PROXY**：相应的OCS Proxy。

- **端口**：相应的OCS 端口。
- **用户名/密码**：对应的用户名和密码。

6. 完成上述信息项的配置后，单击**测试连通性**。
7. 测试连通性通过后，单击**确定**。

新建DRDS数据源

创建DRDS数据源，具体操作如下：

1. 登录*DataWorks控制台*。
2. 单击顶部菜单栏中**项目管理**，导航至**项目属性 > 数据源配置**。
3. 单击**新增数据源**。
4. 在**新增数据源**弹出框中，选择**数据源类型**为DRDS。
5. 配置DRDS数据源的各个信息项。

图 4-215: 配置DRDS数据源

针对DRDS数据源配置项的具体说明如下：

- **数据源名称**：由英文字母、数字、下划线组成且需以字符或下划线开头，长度不超过30个字符。
- **数据源描述**：对数据源的简单描述，不超过1024个字符。
- **数据源类型**：当前选择的数据源类型。

- **JDBCUrl**：JDBC连接信息，格式为：jdbc://mysql://serverIP:Port/database。
- **用户名/密码**：对应的用户名和密码。

6. 完成上述信息项的配置后，单击**测试连通性**。
7. 测试连通性通过后，单击**确定**。

新建FTP数据源

创建FTP数据源，具体操作如下：

1. 登录*DataWorks控制台*。
2. 单击顶部菜单栏中**项目管理**，导航至**项目属性 > 数据源配置**。
3. 单击**新增数据源**。
4. 在**新增数据源**话弹出框中，选择**数据源类型**为FTP。
5. 配置FTP数据源的各个信息项。

图 4-216: 配置FTP数据源

针对FTP数据源配置项的具体说明如下：

- **数据源名称**：由英文字母、数字、下划线组成且需以字符或下划线开头，长度不超过30个字符。

- **数据源描述**：对数据源的简单描述，不超过1024个字符。
- **数据源类型**：当前选择的数据源类型。
- **Portocol**：目前仅支持FTP和SFTP协议。
- **Host**：对应FTP主机的IP地址。
- **Port**：若选择的是FTP协议则端口默认为21，若选择的是SFTP协议则端口默认为22。
- **用户名/密码**：访问该FTP服务的账号密码。

6. 完成上述信息项的配置后，单击**测试连通性**。
7. 测试连通性通过后，单击**确定**。

4.9.1.2.3 编辑数据源

项目管理员需要更改已有数据源的配置信息。

操作步骤

1. 以项目管理员身份创建项目
2. 单击顶部菜单栏中**项目管理**，导航至**项目属性 > 数据源配置**。
3. 在数据源列表搜索框中输入数据源名称模糊匹配查找需要编辑的数据源。
4. 单击操作栏中的**编辑**。

图 4-217: 编辑数据源

5. 配置数据源的信息项，详情请参见新增数据源。
6. 完成上述信息项的配置后，单击**测试连通性**。
7. 测试连通性通过后，单击**确定**。

4.9.1.2.4 删除数据源

项目管理员可以删除已有数据源的配置信息。

操作步骤

1. 以项目管理员身份创建项目

2. 单击顶部菜单栏中**项目管理**，导航至**项目属性** > **数据源配置**。
3. 在数据源列表搜索框中输入数据源名称模糊匹配查找需要删除的数据源。
4. 单击操作栏中的**删除**。

图 4-218: 删除数据源

5. 单击弹出框中的**确定**。

图 4-219: 确定删除

说明：

项目管理员在编辑、删除已有数据源配置时需谨慎操作，以免影响引用该数据源配置的工作流、代码等正常执行，造成生产故障。

4.9.1.3 配置计算引擎

概述

组织管理员成功创建计算引擎后，系统将在引擎初始化的过程中，在相应MaxCompute Project中生成六种角色且自动分配各角色的默认权限。随后，组织管理员在创建项目空间、绑定计算引擎时，开发平台中的角色将与MaxCompute中的角色进行一一关联（具体详见表），且具备MaxCompute Project相应角色的默认权限（可查看**项目管理** > **项目属性** > **计算引擎配置**中的**安全策略**）。

表 4-6: 开发平台角色与MaxCompute角色对应关系

开发平台角色	MaxCompute角色	MaxCompute权限
管理员	role_project_admin	project/table/fuction/resource/instance/job/volume/offlinemodel/package的所有权限
开发	role_project_dev	project/fuction/resource/instance/job/volume/offlinemodel/package的所有权限，拥有table的read和describe权限
运维	role_project_pe	project/fuction/resource/instance/job/offlinemodel的所有权限，拥有volume/package的read权限，拥有table的read和describe权限
部署	role_project_deploy	默认无权限
访客	role_project_guest	默认无权限
系统任务	role_project_scheduler	project/table/fuction/resource/instance/job/volume所有权限，offlinemodel/package的read权限

之后，项目管理员可在**项目管理** > **项目属性** > **计算引擎配置**中查看与调整项目成员角色在计算引擎中的安全策略（默认权限），也可修改在创建项目空间时所勾选的MaxCompute访问身份的配置。

应用示例

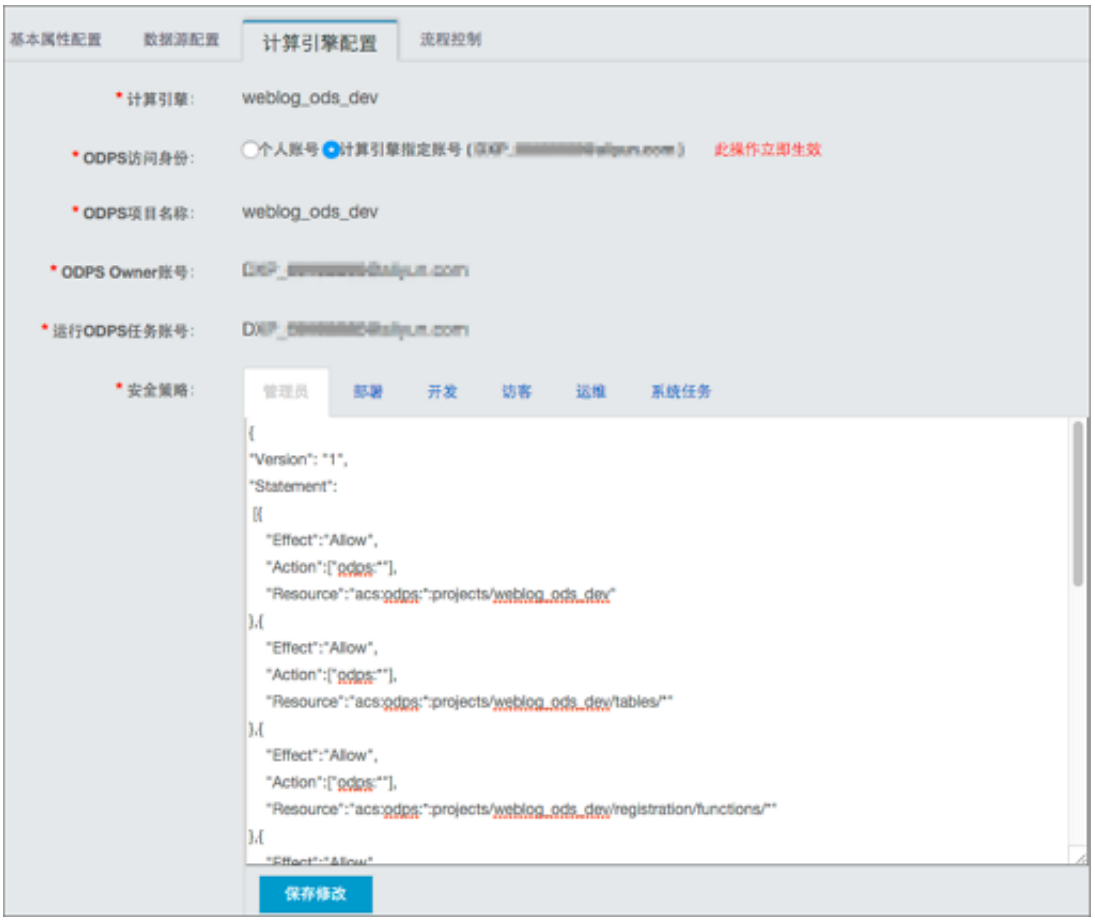
假设由于项目权限分配等因素，项目管理员需对当前项目空间中的角色配置默认的权限。

操作步骤

项目管理员修改角色在计算引擎中的安全策略，具体操作如下。

1. 以组织管理员身份创建项目。
2. 单击顶部菜单中的**项目管理**，导航至**项目属性** > **计算引擎配置**。
3. 在**计算引擎配置**页面中勾选需要配置角色的MaxCompute（原名ODPS）授权信息。

图 4-220: 计算引擎配置页面



针对计算引擎配置页面信息项的具体说明如下：

- **计算引擎**：默认只读，不允许修改项目空间与计算引擎的映射关系。
- **ODPS访问身份**：默认为组织管理员创建项目空间时的配置，即计算引擎指定账号，支持项目管理员切换MaxCompute访问身份。MaxCompute访问身份修改成功后，页面会给出提示。

说明：

Policy文件的简单描述：

- Effect（效力）：表示该Statement的权限类型，取值必须是Allow或Deny。
- Action（操作）：表示主体的访问方式，即授权操作。

  如授权describe和select权限："Action":["odps:Describe","odps:Select"]。
- Resource（资源）：表示主体的访问对象，即授权对象。

4. 完成配置后，单击**保存修改**。

4.9.1.4 流程控制

概述

项目管理员可在**项目属性 > 流程控制**中查看当前项目空间是否**允许在本项目中直接编辑任务和代码**、是否**启用周期性调度**，**自动执行任务**、是否**在本项目中能下载临时结果**的配置信息。

应用示例

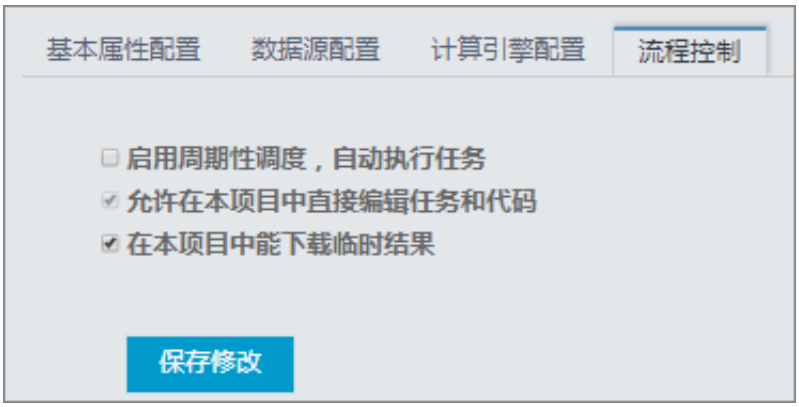
假设由于项目内部因素，现需项目管理员编辑修改当前项目空间是否**启用周期性调度**，**自动执行任务**、是否**允许在本项目中直接编辑任务和代码**和是否**在本项目中能下载临时结果**的配置信息。

操作步骤

项目管理员实现应用实例中描述的场景，具体操作如下：

1. 以组织管理员身份创建项目。
2. 单击顶部菜单中的**项目管理**，导航至**项目属性 > 流程控制**。
3. 在**流程控制**页面中编辑需要修改的配置项。

图 4-221: 流程控制页面



流程控制页面中配置项的具体说明如下：

- **启用周期性调度**，**自动执行任务**：默认为组织管理员创建项目空间时的配置，支持修改；
- **允许在本项目中直接编辑任务和代码**：默认为组织管理员创建项目空间时的配置。只有在创建项目空间时，此选项为未勾选状态，才支持编辑，否则为只读状态。
- **在本项目中下载临时结果**：默认为组织管理员创建项目空间时的配置，不允许下载。

4. 完成上述配置项的编辑后，单击**保存修改**。

4.9.2 成员管理

在阿里云大数据平台的**项目管理** > **成员管理**中，您可进行新增、赋权与删除项目成员等操作。

成员权限点

新增成员

4.9.2.1 成员权限点

阿里云大数据平台角色包括：管理员、开发、运维、部署和访客。具体角色的权限说明如表 *4-7:* 角色权限所示。

表 4-7: 角色权限

DataWorks角色	平台权限特征	MaxCompute角色	MaxCompute权限
项目管理员	指项目空间的管理者，可对该项目空间的基本属性、数据源、当前项目空间计算引擎配置和项目成员等进行管理，并为项目成员赋予项目管理员、开发、运维、部署、访客角色 。	role_project_admin	project/table/fuction/resource/instance/job/volume/offlinemodel/package 的所有权限。
开发	开发角色的用户能够创建工作流、脚本文件、资源和 UDF，新建或删除表，同时可以创建发布包，但不能执行发布操作 。	role_project_dev	project/fuction/resource/instance/job/volume/offlinemodel/package/table 的所有权限。
运维	运维角色的用户由项目管理员分配运维权限；拥有发布及线上运维的操作权限，没有数据开发的操作权限 。	role_project_pe	project/fuction/resource/instance/job/offlinemodel的所有权限，拥有volume/package的read权限和table的read/describe权限 。
部署	部署角色与运维角色相似，但是它没有线上运维的操作权限 。	role_project_deploy	默认无权限
访客	访客角色的用户只有查看权限，没有权限进行编辑工作流和代码等操作 。	role_project_guest	默认无权限

4.9.2.2 新增项目成员

概述

项目管理员可以在当前项目空间下添加项目成员并给赋予角色，包括管理员、开发、运维、部署和访客，成员的用户账号在专有云控制台中创建。具体角色的权限说明如下。

- **管理员**：同一个项目空间可拥有多个项目管理员，具有修改项目空间的配置信息的权限，包括基本属性配置、数据源配置、计算引擎配置、流程控制以及成员管理。
- **开发**：具有数据开发权限，可新建工作流、代码文件、资源等对象。
- **运维**：能够对线上的工作流、节点进行干预操作如修改工作流的调度属性、修改工作流是否采用专用调度资源等，但不具有修改代码的权限，同时也可发布工作流和代码。
- **部署**：能够发布工作流和代码。如从开发环境发布至生产环境。
- **访客**：具有浏览项目空间所有对象的权限，如查看工作流运行情况、查看工作流日志等。

应用示例

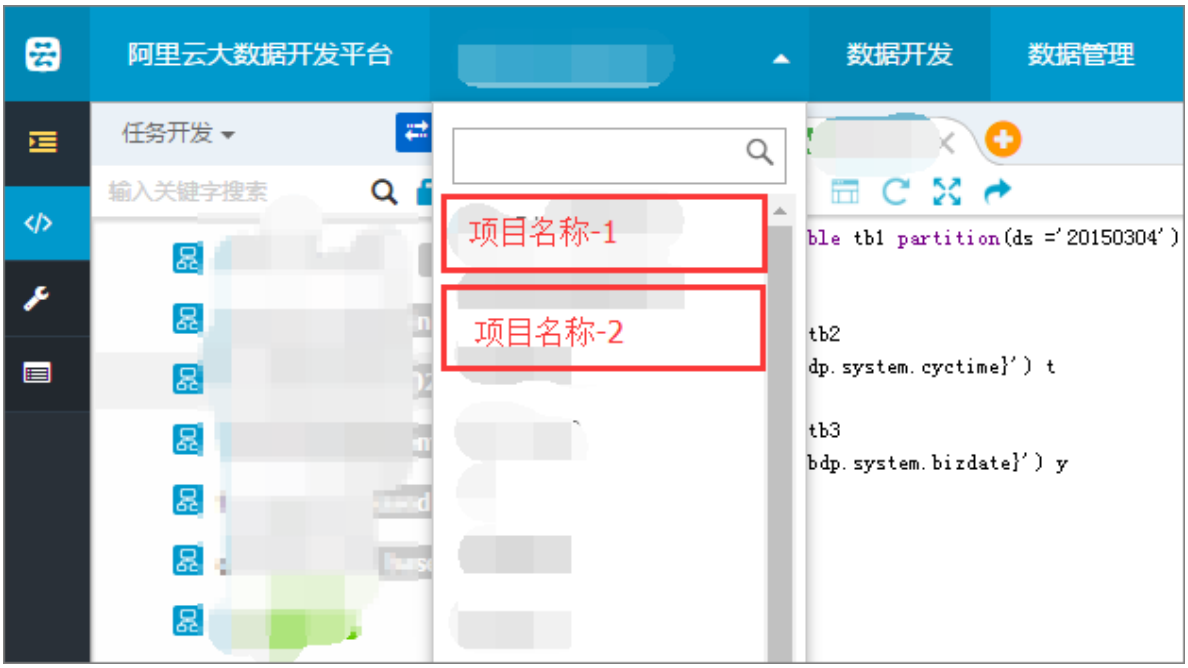
假设您已加入组织，可访问DataWorks，但不能正常使用其服务。现需进入某项目空间进行数据开发工作（赋予开发角色）。

操作步骤

项目管理员需将您加入某项目空间，并赋予开发角色，具体操作如下。

1. 以组织管理员身份创建项目。
2. 在顶部项目空间处切换到所属项目空间视角下。

图 4-222: 切换项目空间



3. 单击顶部菜单栏中的**项目管理** > **成员管理**。
4. 单击**添加成员**。
5. 在**添加项目成员**弹出框中，通过成员名称模糊匹配查找，并选中匹配列表中需添加的成员（支持选择多个用户）。

图 4-223: 添加成员

6. 单击**提交**。

4.9.2.3 删除项目成员

概述

项目管理员可主动移出本项目空间中的成员，被移出项目空间的成员将不再能访问该项目空间中的所有资源，但该用户之前开发的工作流、代码等任务均不会受影响。

应用示例

假设某项目成员已离职，现需项目管理员收回其相关权限并将其移出该项目空间。

操作步骤

项目管理员移出成员，具体操作如下：

1. 以项目管理员身份创建项目。
2. 在顶部菜单栏项目空间处切换之至所属项目空间视角下。
3. 单击顶部菜单中的**项目管理** > **成员管理**。
4. 在**项目管理** > **成员管理**列表的搜索框中，输入成员名称或登录名称来模糊匹配查找需要移出项目空间的成员。
5. 单击操作栏中的**移出本项目**。

图 4-224: 项目成员列表

6. 在**移出项目**对话框中，单击**确定**。

5 分析型数据库AnalyticDB

5.1 什么是分析型数据库

分析型数据库AnalyticDB（原名 ADS）是阿里巴巴自主研发的海量数据实时高并发在线分析RT-OLAP（Realtime OLAP）云计算服务，支持毫秒级对万亿级数据进行即时的多维分析透视和业务探索。

分析型数据库可以让您使用现有的商业智能 (BI) 工具和ETL工具经济高效地轻松分析与集成您的所有数据。

图 5-1: 数据库对比

在业务系统中， 我们通常使用OLTP （OnLine Transaction Processing）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。当需要进行查询或计算的数据量过大（达到数千万甚至数十亿条），或需要进行的计算非常复杂时，OLTP类数据库系统便力不从心了。

与主流数据系统相比，AnalyticDB使用OLAP（On-Line Analytical Processing）系统。从广义上，OLAP系统是相对OLTP系统而言的，它不特别关心数据的输入、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。OLAP系统通常分为MOLAP（Multi-Dimensional OLAP）、ROLAP（Relational OLAP）和HOLAP三类。

- MOLAP：多维OLAP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。它的优点是查询速度快等；缺点是数据必须预先建模，无法根据使用者的意愿进行即时灵活的修改。
- ROLAP：关系型OLAP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。它的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；缺点是在海量数据的情况下分析计算缓慢。
- HOLAP是MOLAP和ROLAP的混合模式。

AnalyticDB是一套RT-OLAP系统。在数据存储模型上，AnalyticDB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。AnalyticDB使用分布式计算技术，可以处理百亿条甚至更多量级的数据，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。

AnalyticDB兼具实时和自由计算海量数据的能力，实现了速度驱动的大数据商业变革。AnalyticDB拥有快速处理千亿级别海量数据的能力，数据分析中使用的数据是业务系统中产生的全量数据而不再是抽样的，这使得数据分析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，AnalyticDB能够支撑较高并发查询量，同时通过动态的多副本数据存储计算技术也保证了较高的系统可用性，所以它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AnalyticDB。

AnalyticDB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。

5.2 基本概念

用户

用户是AnalyticDB数据库的使用者，可通过云账号进行登录数据库。在AnayticDB数据库中，不同用户可以被授予不同的权限，用户的操作也均可以被细粒度审计。

数据库

数据库是组织、存储和管理数据的仓库。在AnalyticDB中，数据库是租户隔离的基本单位，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，未经用户授权，无法查看和管理数据库的内部结构和信息。用户只能查看大多数数据库级别的参数，但不能修改。

在AnalyticDB中，一个数据库对应一个用于访问的域名和端口号，并且有且只有一个Owner（即创建者）。

在AnalyticDB中，用户的宏观资源配置是以数据库为粒度进行的，所以在创建数据库时分析型数据库通过业务预估的QPS、数据量、Query类型等信息智能的分配数据库初始资源。

表组

表组是一系列可发生关联的表的集合，是AnalyticDB为方便管理关联数据和资源配置而引入的概念。在AnalyticDB中，表组可分为普通表组和维度表组，说明如下：

- 普通表组：
  - 普通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。
  - 一个普通表组最大支持创建256个普通表。
  - 数据库中数据的副本数（指数据在AnalyticDB中同时存在的份数）必须在表组上进行设定，同一个表组的所有表的副本数一致。
  - 只有同一个表组的表才支持快速HASH JOIN。
  - 同一个表组内的表可以共享一些配置项（例如：查询超时时间）。如果表组中的单表对这些配置项进行了个性化配置，那么在进行表关联时会用表组级别的配置进行覆盖单表的个性化配置。
  - 同一个表组的所有表的一级分区（即HASH分区）的分区数建议一致。
- 维度表组：
  - 用于存放维度表（一种数据量较小，但能和任何表进行关联的表）。
  - 维度表组是创建数据库时自动创建的，一个数据库有且只有一个，不可修改和删除。

表

AnalyticDB支持标准的关系表模型。在AnalyticDB中，表通常分为普通表（也称事实表）和维度表，说明如下：

- 普通表：创建时必须至少指定一级分区（即HASH分区）列、分区相关信息，同时还需要指定该普通表所属的表组。普通表支持根据若干列进行数据聚集（聚集列），以实现高性能查询优化。单表最大支持1024个列，可支持数千亿行甚至更多的数据。
- 维度表：创建时不需配置分区信息，但会消耗更多的存储资源，单表数据量大小受限。维度表最大可支持千万级的数据条数，可以和任意表组的任意表进行关联。

普通表最多支持两级分区，一级分区是 HASH分区，二级分区是LIST 分区。HASH 分区和LIST分区说明如下：

- HASH 分区：
  - 根据导入操作时已有的一列内容进行散列后进行分区。
  - 一个普通表至少有一级HASH分区，分区数最小支持8个，最大支持256个。
  - 多张普通表进行快速 HASH JOIN ，JOIN KEY必须包含分区列，并且这些表的HASH分区数必须一致。
  - 数据装载时，仅包含HASH分区的数据表会全量覆盖历史数据。
- LIST分区：
  - 根据导入操作时所填写的分区列值来进行分区。同一次导入的数据会进入同一个LIST分区，因此LIST分区支持增量的数据导入。
  - 一个普通表默认最大支持365*3个二级分区。

在AnalyticDB中，两级分区表同时支持HASH JOIN和增量数据导入。对于任一分区形态的表，查询操作均不强制要求指定分区列，但查询操作指定分区列或分区列范围时可能提高查询性能。

在AnalyticDB中，根据表的数据更新方式，表分为批量更新表和实时更新表，说明如下：

- 批量更新表：适合将离线系统（如MaxCompute）产生的数据批量导入到分析型数据库，供在线系统使用。
- 实时更新表：支持通过INSERT、DELETE语句增加和删除单条数据，适合从业务系统直接写入数据。

注意：

- 分析型数据库不支持读写事务。
- 数据实时更新后，一分钟左右才可查询。
- 分析型数据库遵循最终一致性。

列

- 支持boolean、tinyint、smallint、int、bigint、float、double、varchar、date、timestamp等多种MySQL标准数据类型。
- 支持多值列multivalue（AnalyticDB特有的数据类型列），高性能存储和查询一个列中的多种属性值信息。
- 支持删除列的自动化索引，无需手动追加建立HashMap索引。

ECU

ECU是弹性计算单元，是Analytic DB的资源调度和计量的基本单位。

ECU可配置不同的型号，每种型号的ECU可配置CPU核数（最大、最小）、内存空间、磁盘空间、网络带宽等多种资源隔离指标。Front Node、Compute Node、Buffer Node均由ECU进行资源隔离。Compute Node的ECU数量和型号需由用户配置（弹性扩容/缩容），Front Node、Buffer Node的数量和型号系统自动根据ComputeNode的情况换算和配置。Analytic DB出厂时已根据机型配置预设了多种最佳的ECU型号。

5.3 数据类型介绍

5.3.1 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。

5.3.2 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

5.4 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，如果一个列需要作为Hash Join的Join Key，那么您可动态根据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*。

5.5 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规格如下表所示。

型号	内存	SSD磁盘容量	SATA磁盘容量
c1	7.5GB	60GB	N/A
c8	45GB	480GB	N/A
s1	25GB	250GB	1536GB
s8	60GB	600GB	6000GB

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*管理语句。

5.6 快速开始

5.6.1 登录AnalyticDB控制台

AnalyticDB For DMS控制台是AnalyticDB数据库的管控平台，通过AnalyticDB For DMS控制台您可以创建、删除和进入数据库等。

前提条件

1. 已获取AnalyticDB For DMS控制台的云账号（通过大数据管家开户）。
2. 已获取天基的URL地址。

> **说明**：
> 如果还未获取以上信息，请联系系统管理员获取。

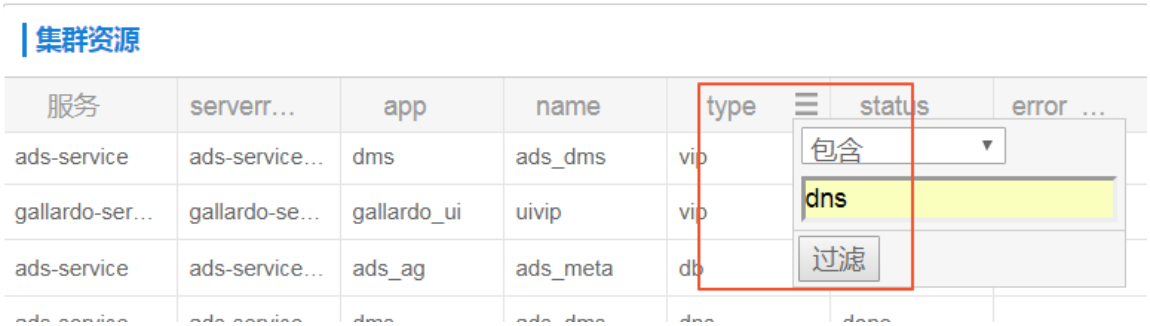
操作步骤

1. 登录天基，从天基查看AnalyticDB For DMS控制台的域名。
   a) 登录天基，如图 *5-2: 天基*所示。

   图 5-2: 天基

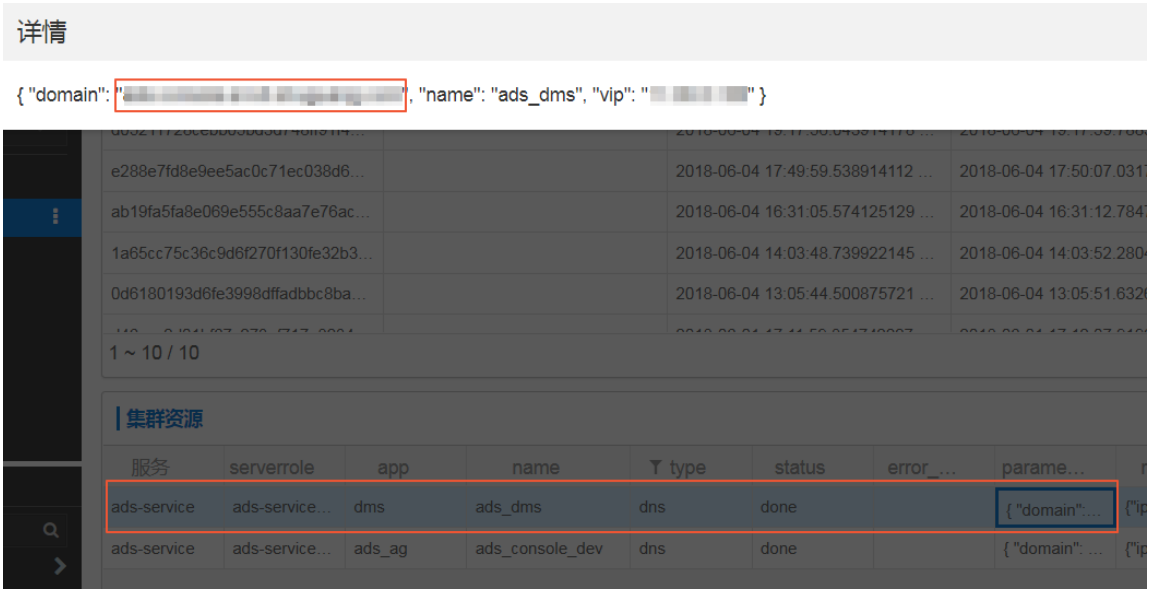
   b) 在左侧选择**集群**，然后在`Project`下拉列表中选择ads，筛选出AnalyticDB产品。
   c) 选择筛选出的AnalyticDB产品，将鼠标指向后面的菜单项图标并选择**Dashboard**，进入AnalyticDB的**集群Dashboard**界面。
   d) 在**集群资源**列表中，单击列表的**type**表头，以dns进行筛选，如图 *5-3: 筛选条件*所示。

图 5-3: 筛选条件



e) 在筛选结果中查找**name**为ads_dms的行。右键单击该行的**parameters**单元格，并选择**显示更多**，在弹出的**详情**信息框中查看AnalyticDB的域名，如图 *5-4:* 查看域名所示。

图 5-4: 查看域名



2. 拷贝AnalyticDB域名到浏览器地址栏，进入AnalyticDB For DMS控制台登录界面，如图 *5-5:* 登录界面所示。

图 5-5: 登录界面

3. 输入用户名和密码，登录到AnalyticDB For DMS控制台的数据库管理界面，如图 *5-6:* 数据库管理界面所示。

图 5-6: 数据库管理界面

5.6.2 创建数据库

操作步骤

1. 登录*AnalyticDB*控制台

2. 控制台数据库页面右上角单击**新建数据库**，根据实际需要设置各参数，如图 5-7: 创建数据库所示。

图 5-7: 创建数据库

表 5-1: 数据库参数描述

参数名称	描述
数据库名称	数据库名称在AnalyticDB全部集群上必须是全局唯一。 **说明**： ads_demo名称已经被使用，实际操作时请您不要设置为此名称。
区域	Region所在地，从下拉列表中选择AnalyticDB的Region所在地。
ECU型号	根据实际需求选择ECU型号。 **说明**： 数据库创建完成后，ECU型号不可修改。 ECU是AnalyticDB的资源计量的最小单位，具有多种型号，每种型号的vCPU核数、内存、磁盘空间不同，详细信息请参见*ECU*详解。

参数名称	描述
ECU初始数量	根据实际需求填写所需ECU数量，必须是偶数且至少2个。 **说明**： 数据库创建完成后，AnalyticDB支持通过调整ECU数量进行扩容/缩容，您可根据实际需要随时调整ECU数量。

3. 单击**确定**，弹出**执行结果**提示框，提示**操作成功**，**十五分钟后生效**。

   如果返回错误，请根据错误提示进行修正。错误一般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等。

   创建成功后，当数据库状态变为运行中时（约需要15分钟），则表示数据库已创建好，如图 *5-8: 数据库创建完成*所示。

   图 5-8: 数据库创建完成

4. 单击数据库后面的**进入**，进入该数据库。

5.6.3 创建和管理表组

数据库分配和配置完成后，您需要根据实际需要创建表组来容纳您的数据表。本节将指导您创建和管理数据表组。

背景信息

与MySQL不同的是，在AnalyticDB中创建数据表前，您必须确保该表所归属的表组是存在的，如果不存在，则首先需要创建一个表组。

创建表组前，您可能需要了解表组信息。

AnalyticDB支持GUI界面和SQL语句两种方式创建表组，本节介绍通过GUI界面方式来创建表组，SQL语句方式请参见表组管理语句。

操作步骤

1. 在DMS For AnalyticDB的数据库管理界面中，单击某个数据库的**库名**，进入该数据库的DMS For AnalyticDB界面，如图 *5-9: 数据库的DMS For AnalyticDB界面*所示。

图 5-9: 数据库的DMS For AnalyticDB界面

2. 在左上角选择**新建** > **表组**，打开**新建表组**对话框，如图 *5-10: 新建表组*所示。

图 5-10: 新建表组

3. 根据实际情况填写各参数，参数说明如表 *5-2: 表组参数描述*所示。

表 5-2: 表组参数描述

参数名称	描述
表组名	表组名称，用户根据实际需要自定义。
最小副本数	数据在数据库中同时存储的份数，可选值为2、4、8，默认为2。
超时时间（毫秒）	表组全局查询超时时间，默认为30000。

4. 参数设置完成后，单击**提交**，弹出**提示**提示框，提示执行成功。

5.6.4 创建和管理表

表组创建完成后，您需要根据实际需要创建一张或多张数据表来存储您的项目数据。本节将指导您创建和管理数据表。

背景信息

AnalyticDB采用关系模型存储数据（即用二维表组织和存储数据）。与MySQL一样，在数据写入AnalyticDB前，您需要先创建对应的数据表。

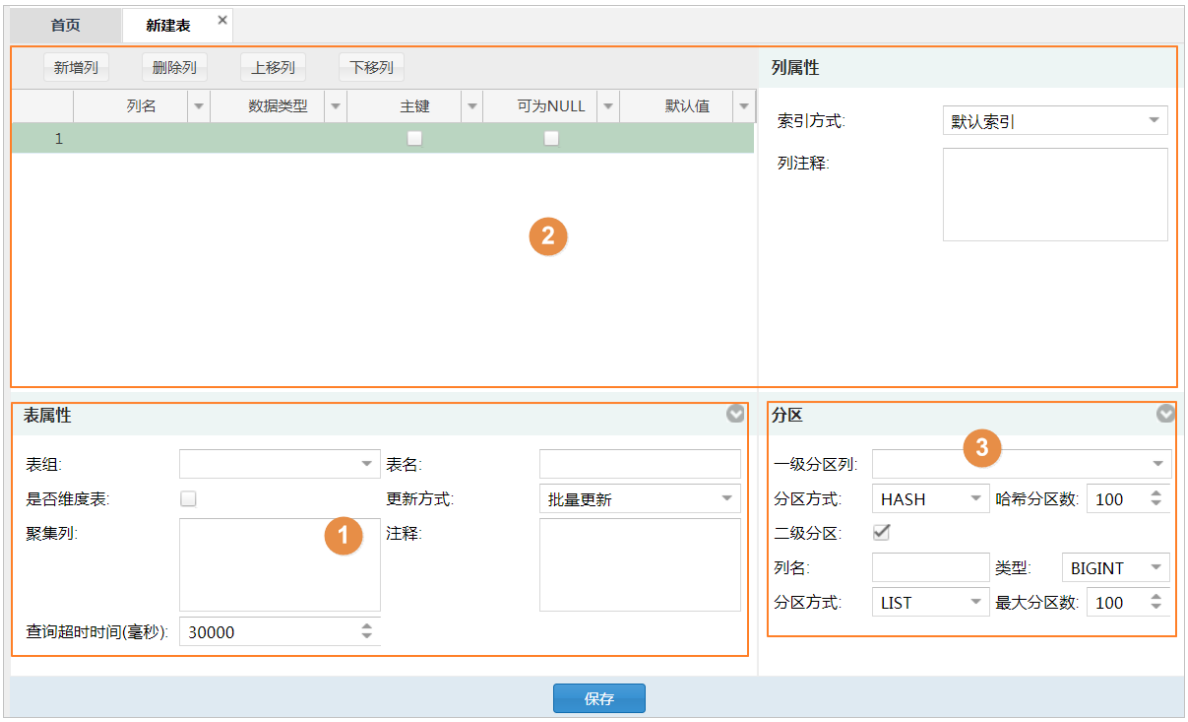
创建数据表前，您可能需要了解表组、表、列和列的类型和属性信息。

AnalyticDB数据库支持通过GUI界面或SQL语句两种方法来创建表。本节介绍通过GUI界面来创建表，SQL语句方法请参见表管理语句。

操作步骤

1. 在数据库的DMS For AnalyticDB界面，选择左上角的**新建** > **表**，进入**新建表**对话框，如图 *5-11: 新建表*所示。

图 5-11: 新建表



2. 根据实际情况设置各参数，参数说明如表 *5-3:* 表属性、表 *5-4:* 列属性和表 *5-5:* 分区所示。

说明：

表属性、**列属性**、**分区**区域中的部分参数是有关联关系的，设置表参数时，请您按照**表属性** > **列属性** > **分区**的顺序进行设置。

分区区域中的参数只有普通表才需要填写，维度表不需要填写。

表 5-3: 表属性

参数名称	描述	备注
表组	表组名称，从下拉列表中选择。	必填参数，但维度表自动归属维度表组，不需要填写此参数。
表名	数据表的名称。	必填参数，表名必须与源数据表的表名一致。
是否维度表	创建的是否为维度表，勾选则创建维度表。	N/A

参数名称	描述	备注
	维度表自动归属维度表组，勾选此参数项时，**表组**参数以及**分区**区域内的所有参数均不需要配置。	
更新方式	表数据更新方式： • 批量更新：仅支持离线批量更新数据，适用于从其他系统批量导入数据源的场景。例如：从MaxCompute批量导入数据。 • 实时更新：仅支持通过INSERT实时插入数据。	必填参数。
聚集列	设置表的聚集列，可设置多个。 在物理上，同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进行存储。查询数据时，指定聚集列内容或范围，可提高查询性能。	选填参数。
查询超时时间（毫秒）	表查询超时时间，默认为30000。	必填参数，表关联时表组的**查询超时**会覆盖此参数。
注释	表的注释说明。	N/A

表 5-4: 列属性

参数名称	描述	备注
列名	数据表中每一列的列名称。	必填参数，数据表为批量更新表时，列名必须与数据源对应列的名称一致。
数据类型	列的数据类型。	必填参数，必须与数据源对应列的数据类型一致。
主键	是否将此列设置为表数据的主键，主键用于唯一地标识表中的某一条记录；表关联时用来在一个表中引用另一个表的指定记录。 主键设置方式如下： • 勾选：设置列为主键列。 • 去勾选：设置列为非主键列。 默认为去勾选。	• 实时更新表：至少设置一列为主键。 • 批量更新表：不支持设置主键。

参数名称	描述	备注
可为NULL	列数据是否可以为空值，设置方式如下： • 勾选：支持列值为空值。 • 去勾选：不支持列值 为空值。 默认为去勾选。	主键列和分区列不支持空值，所以不支持勾选此项。
默认值	列数据的默认值。	选填参数。
索引方式	列数据的索引方式： • 默认索引：使用AnalyticDB的自动索引功能，AnalyticDB会根据实际数据的分布情况来自动进行索引。 • 无索引：如果您确定某一列不需要进行筛选查询，则可以指定该列不创建索引来节省数据存储空间。 默认值为**默认索引**。	AnalyticDB自动索引功能强大，您在创建表时通常无需关心一个列的索引情况，保持此参数的默认值即可。
列注释	列的注释说明。	N/A
新增、删除、移动列	通过**新增列**、**删除列**、**上移列**和**下移列**进行相应操作。	一个普通表最大支持1024列。

表 5-5: 分区

参数名称	描述	备注
一级分区列	一级分区列，必须是一个主键列。	必填参数。
分区方式	HASH分区。	必填参数。
哈希分区数	一级分区个数，当前最小支持8个，最大支持256个，默认为100。	必填参数。一般来讲，每个分区的数据不超过1500万条为宜，但您可通过划分二级分区来扩大表的数据存储量。
二级分区	设置表格是否进行二级分区： • 勾选：进行二级分区。 • 去勾选：不进行二级分区。去勾选时，以下参数自动隐藏，不需要填写。 默认为去勾选。	选填参数。

参数名称	描述	备注
列名	二级分区列的列名称。不可与**列属性**中的列重复。	选填参数。
类型	二级分区列的数据类型，当前只支持BIGINT类型。	选填参数。
分区方式	LIST分区。	选填参数。
最大分区数	一个一级分区包含的二级分区的最大分区数，最大支持365*3个，默认为100。	选填参数。

3. 参数设置完成后，单击**保存**，弹出**提交变更**对话框，显示实际的建表DDL，示例如图 *5-12:* 提交变更所示。

图 5-12: 提交变更

4. 检查无误后，单击**确定**。

说明：

创建表后，如果需要修改表，您可以右键单击表，选择**编辑表**来修改表的相关信息。当前只支持修改表的**聚集列**、**查询超时时长**、**注释**和二级分区的**最大分区数**，以及新增列。

5.6.5 导入/写入数据

数据表创建完成后，您需要将实际业务的数据导入或写入到AnalyticDB中。本节将指导您将数据导入或写入AnalyticDB的数据表中。

背景信息

AnalyticDB支持多种数据导入和写入方式，常见的有：

- 从MaxCompute批量导入数据。如果AnalyticDB中的数据表是批量导入表，则您可通过多种方式将数据导入到此表中，例如：通过DATA PIPELINE系列命令（详细信息请参见 导入数据）导入数据、通过DMS For AnalyticDB界面导入数据等。
- 通过INSERT/DELETE实时写入数据。如果AnalyticDB中的数据表是实时更新表，则您可以直接通过DMS For AnalyticDB界面的SQL功能进行实时写入数据，详细信息请参见*INSERT/DELETE*命令。
- 通过Kettle等ETL工具实时写入数据。
- 通过阿里云数据传输将RDS数据表变更实时同步到AnalyticDB中，详细信息请参见《使用阿里云数据传输实时同步 *RDS* 的数据》章节。

操作步骤

- 从MaxCompute批量导入数据，以DMS For AnalyticDB界面操作为例。

  1. 登录并进入数据库的DMS For AnalyticDB界面，选择**导入导出** > **导入**，弹出**导入数据**对话框，如图 *5-13:* 导入数据所示。

     说明：

     以下示例是从阿里云MaxCompute 批量导入数据到AnalyticDB中，数据导入路径按照 *odps://project_name/table_name/partition_spec*的格式来填写。

图 5-13: 导入数据

2. 根据实际情况填写各参数，其中**数据源路径**是MaxCompute上的表路径，**目标表**是AnalyticDB中的实时更新表，**一级分区列**是AnalyticDB中的实时更新表的一级分区列。
3. 参数设置完成后，单击确定，进入导入状态一览界面。AnalyticDB会对导入任务进行调度，根据当前系统繁忙情况、需要导入的数据大小和结构不同，数据导入可能需要二十分钟到数小时不等。

- 通过INSERT/DELETE实时写入数据。

```
insert into table my_first_table (user_id,amt,num,cat_id,thedate)
values (12345, 80, 900, 1555, 20140101);
```

实时更新表创建后，需要约一分钟的准备时间，此时写入的数据需要在准备完成后才可查询，否则查询会报错。实时更新表准备完成后，通过INSERT/DELETE实时更新的数据一般会延迟一分钟才可查询。

说明：

AnalyticDB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AnalyticDB并不能作为OLTP系统使用。

- 从RDS等其他云系统离线批量导入数据。您可通过阿里云的数据集成（Data Integration）产品将RDS等其他云系统数据同步到AnalyticDB中。开通CDP后，按照如下示例配置同步Job：

说明：

- 运行以下任务之前，至少需要在AnalyticDB中将表的Load Data权限授权给登录账号。
- 以下示例中的AnalyticDB数据表是批量更新表。
- AnalyticDB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AnalyticDB并不能作为OLTP系统使用。

```
{
"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
}
}
}
}
```

如果AnalyticDB的数据表是实时更新表，而您仍需通过数据集成从其他数据源导入数据，则只需要修改writer配置即可，示例如下（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"
}
```

5.6.6 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 和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，内容如下所示。

```
$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，可以使用以下命令。

```
$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。

5.6.7 用户授权

数据库创建全部完成后，您可以对每个使用该数据库的用户进行授权，以使每个用户都能以最小的权限来完成日常工作。本节将向您展示如将数据库的操作权限授权给用户。

背景信息

在AnalyticDB中，创建数据库的账号是这个数据库的Owner账号，拥有最高的权限。为安全起见，Owner账号通常只有数据库管理员才可使用。为使其他用户也可以使用数据库，您可以根据每个用户的实际需要对其进行授权，授权一般遵循最小权限原则。数据库的权限管理请参见用户与权限。

操作步骤

1. 登录DMS For AnalyticDB控制台，在数据库管理界面单击某个数据库的**库名**，进入该数据库。
2. 在左侧**对象列表**中，单击数据库中的**用户**，查看具有访问该数据库权限的全部用户账号，如所示。
3. 右键单击某个账号，选择**新建授权**或单击顶部菜单栏中的**新建授权**，如图 *5-14:* 新建授权按钮所示。

图 5-14: 新建授权按钮

4. 根据实际需要填写图 *5-15:* 权限管理对话框中参数，参数说明如所示。

图 5-15: 权限管理

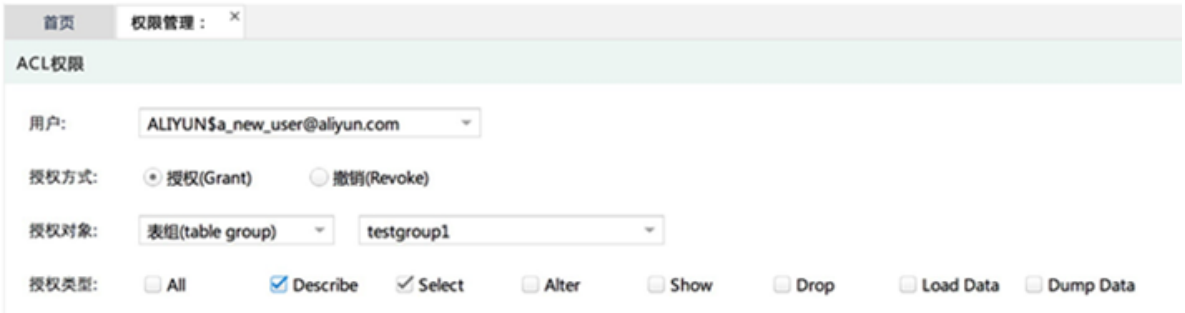


表 5-6: 权限管理参数描述

参数名称	描述	备注
用户	填写或从下拉列表中选择所要授权的用户。	AnalyticDB支持多种账号来源，所以在输入账号时需要声明账号来源。例如：ALIYUN$ 前缀代表该账号是阿里云账号，则输入阿里云账号的格式为ALIYUN$a_new_user@aliyun.com。
授权方式	可选值如下： • 授权（Grant）：授权权限。 • 撤销（Revoke）：撤销已有权限。	N/A
授权对象	授权对象可以是数据库、表组、表、列。	N/A

参数名称	描述	备注
授权类型	选择授与用户何种操作权限。根据用户实际需要选择。	为安全起见，授权一般遵循最小权限原则，以防用户权限过高损坏数据库。例如：误操作导致数据丢失等

5. 参数设置完成后，单击**保存**，完成授权操作。

   操作完成后，a_new_user@aliyun.com用户即可使用AccessKey登录AnalyticDB并进行相应的操作。

5.7 DDL

5.7.1 数据库创建与配置

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。

5.7.2 表组管理语句

表组是一系列可发生关联的表的集合。在创建表时，您需要填写表所归属的表组，所以在创建表之前，您必须确保已创建好表所归属的表组。本节将向您展示如何通过DDL来创建、修改和删除表组。

在数据库DMS For AnalyticDB界面中的SQL窗口中，您可以通过DDL创建、修改和删除表组。

创建表组

DDL语句格式如下：

```
create tablegroup db_name.tablegroup_name options(minRedundancy=2
executeTimeout=30000);
```

DDL语句中各参数说明如所示。

参数名称		描述	备注
db_name		数据库名称。	N/A
tablegroup_name		表组名称。	不可与当前数据库中现有表组重复。
options	minRedundancy	表组的副本数，默认为2，可配置为2、4、8。 表组副本数配置为4或更高时，可在一定程度上增加数据库的最大承受	可选项，不带此两个参数时，按默认值创建表组。

参数名称		描述	备注
		QPS，但是数据存储费用也会相应增加。	**说明**： 一般情况 下，不推荐修改表组的默认配置。
	executeTimeout	表组的全局Query超时时间，默认为30000，单位毫秒。	

修改表组

AnalyticDB允许您修改表组的*minRedundancy*和*executeTimeout*参数。

DDL语句格式如下：

```
alter tablegroup db_name.tablegroup_name key=value;
```

*key*为属性名，可以设置为*minRedundancy*或*executeTimeout*；*value*为新的属性值。*minRedundancy*修改后，新值在下次装载数据时生效。

删除表组

DDL语句格式如下：

```
drop tablegroup db_name.tablegroup_name;
```

说明：
AnalyticDB仅允许您删除空的普通表组，不允许删除维度表组。

5.7.3 表管理语句

表是数据库用来存储数据的对象。在进行数据存储前，您需要先创建相应的数据表。本节将向您展示如何通过DDL创建、修改和删除表。

创建普通表

相比于传统关系型数据库，AnalyticDB具有一些独有特性，所以AnalyticDB的DDL在遵循类MySQL规范的基础上，还有一些独有的属性和语法。

DDL语句格式：

```
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 bigint)]
[SUBPARTITION OPTIONS (available_partition_num = 30)]
[CLUSTERED BY (col3,col4)]
TABLEGROUP ads_test ;
[options (updateType='{realtime | batch}')]
```

参数说明如表 5-7: 必填参数描述和表 5-8: 选填参数描述所示。

说明：

[]内的参数项均为可选项。

如果表仅有一级HASH分区并且是批量更新表，则每次导入数据时会对已有数据进行全量覆盖。

表 5-7: 必填参数描述

参数名称	描述	备注
db_name	数据库名称。	N/A
table_name	表名称。	不可与当前数据库中现有表重复。
col*	语句中的*col1 bigint COMMENT 'col1'*等是表的列信息，基本格式为列名 类型 *COMMENT '注释'*，详细信息请参见 列的类型和属性。	同一张数据表中，列名称不可重复。一个普通表最大支持1024个列。
PARTITION BY HASH KEY()	一级分区列，必须是一个主键列，只支持HASH分区。HASH分区动态分区值类型，即根据实际数据中的某一列的内容进行分区。一个HASH分区一般不宜超过1500万条记录。您可通过设置二级分区来扩大数据存储容量。	普通表必须指定一级分区列。HASH分区列的数据需要尽可能保持均匀分布，如果有明显的数据倾斜，则会严重的影响查询性能。
PARTITION NUM	HASH分区数，最小支持8个，最大支持256个。	**说明**：同一个表组中的所有表，HASH分区数建议一致。
TABLEGROUP	表所归属的表组。普通表必须指定其所归属的表组。	一个普通表组最多可以创建256个普通表。

表 5-8: 选填参数描述

参数名称	描述	备注
primary key ()	表的主键，用于唯一地标识表中的某一条记录；表关联时用来在一个表中引用另一个表的指定记录。	实时更新表至少需要设置一个主键；批量更新表不需要设置主键。
SUBPARTITION BY LIST ()	二级分区列，只支持LIST分区。二级分区为非动态分区，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。 二级分区列必须是新的列，不可与已有的列重复，列数据类型只支持bigint型。	如果每次导入数据是增量导入，则需要指定二级LIST分区信息。 实时更新表的二级分区仅用于极大的扩展单表容量、生命周期管理，其增量更新不依赖于二级分区。
SUBPARTITION OPTIONS ()	最大保留的二级分区数，最大支持365*3个。 新数据装载后，如果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。	N/A
CLUSTERED BY ()	聚集列，可指定多列。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。 在物理上，同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进行存储。	如果您将表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，则可能较显著的提升查询性能。
updateType	表数据更新方式： • realtime：实时更新，只支持实时写入数据。 • batch：批量更新，只支持批量离线导入数据。 不带此参数时，默认为批量更新。	N/A

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。

一级分区数量和二级分区数量需要根据表的数据量、数据库拥有的资源数来设置，并非越多越好。如果一级分区和二级分区过多而数据库的资源数过少，则分区的数据Meta很容易将内存耗尽。

创建维度表

DDL语句格式：

```
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。

说明：

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。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。

修改表

表创建完成后，您可以对表进行部分修改，目前主要支持新增列。

DDL语句格式：

```
ALTER TABLE db_name.table_name ADD column col_new varchar;
```

说明：

批量更新表新增加的列后，在数据重新装载后才会生效；实时更新表新增列后，一般需要几分钟后才可以读写。

删除表

DDL语句格式如下：

```
drop table db_name.table_name;
```

5.7.4 列的类型和属性

通过DDL建表时，列的定义如下：

```
col_name type [NOT NULL | NULL] [DEFAULT default_value] [COMMENT '
string'] [disableIndex true]
```

其中[]中的参数为可选参数，关键参数说明如下：

- col_name：列名。
- type：列的数据类型，详细信息请参见支持的数据类型。
- [NOT NULL | NULL]：列的值是否可为空值。
- [DEFAULT default_value]：列的默认值，与标准MySQL DDL相同。
- disableIndex：列属性。如果某列不在实际查询中筛选和计算，则可通过设置列属性disableIndex = true来屏蔽AnalyticDB的默认列索引。

5.7.5 索引语句

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。

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。

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。

5.7.6 ECU管理语句

ECU是AnalyticDB中存储资源和计算资源的分配单元，您可以通过修改ECU的数量来对数据库进行扩容或缩容。本节将向您展示如何修改ECU的数量。

您可以通过以下方式来修改ECU数量：

- 通过DMS For AnalyticDB界面中的**DB容量管理**功能进行修改。
- 通过DDL语句进行动态修改。

通过DMS界面修改ECU数量

1. 登录数据库的DMS For AnalyticDB，在菜单栏选择**实例管理** > **DB容量管理**，打开**DB容量管理**界面。
2. 单击右上角的**容量变更**，填写需要变更的ECU数量，如图 *5-16:* 容量变更所示。

   图 5-16: 容量变更

3. 确认数量后，单击**确定**，扩容/缩容任务显示在界面中，您可以查看扩容/缩容任务的当前状态等信息。

通过DDL动态修改ECU数量

说明：

通过DDL修改ECU数量时，需要Alter Database权限或数据库Owner角色。

通过DDL修改ECU数量的语句如下：

```
ALTER DATABASE SET ecu_count = N;
```

其中N为要设置的目标ECU数量，如果N大于当前的ECU数量，则为扩容行为；如果N小于当前的ECU数量，则为缩容行为。

说明：
如果用户的数据量大于目标ECU的总存储，则更改会失败。

扩容/缩容都不是瞬时的同步操作，您可以使用元数据查询扩容/缩容状态，语句如下：

```
select * from information_schema.resource_request;
```

查看已有的ECU状态的语句如下：

```
select * from information_schema.current_instances;
```

5.8 DML

5.8.1 SELECT语句

SELECT语句是AnalyticDB的查询语句，您可以通过该语句查询数据库中存在的数据。本节将向您展示如何通过SELECT语句查询数据库中的数据。

SELECT语句中涉及了附录一#元数据库数据字典、逻辑表达式和特殊语法和多计算引擎和*Hint*中的概念或内容，为了更好的理解SELECT语句，建议您先了解一下这些内容。

SELECT 语句如下：

```
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}]
```

SELECT通常可分为以下几个部分：

- 列投射（列和列表达式）
- FROM/JOIN 子句（表扫描、连接和子查询）

- 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（列和列表达式）

列投射（列和列表达式）的基本结构如下：

```
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列），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或虚表（物理表的别名引用或子查询产生的表），支持的基本列类型如表 *5-9:* 支持的基本列结构类型所示。

表 5-9: 支持的基本列结构类型

支持的基本列结构类型	描述
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 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（*table.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。

支持的基本列结构类型	描述
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。
SELECT A.* ...	通配符代表指定表的所有列。
SELECT COUNT(DISTINCT a) FROM A GROUP BY b	a 不是分区列，详细信息请参见逻辑表达式和特殊语法。
SELECT DISTINCT a FROM A	a 不是分区列，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 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，详细信息请参见逻辑表达式和特殊语法。

当 select_expr为列表达式时，支持和不支持的列表达式类型如表 *5-10:* 支持的列表达式结构类型和表 *5-11:* 不支持的列表达式结构类型所示。

表 5-10: 支持的列表达式结构类型

支持的列表达式结构类型	描述
SELECT a + b ...	常用算术操作符（+，-，*，/, %），其中表达式a和b可以是合法的值、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，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。
SELECT COUNT(*)	常用聚合函数，详细信息请参见附录二#SQL函数表。
SELECT UDF_SYS_CO UNT_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(...)）嵌套列普通表达式（CASE WHEN...）做为其子表达式。

支持的列表达式结构类型	描述
SELECT SUM(a) / COUNT(a) FROM A	列普通表达式（/）嵌套聚合表达式（COUNT(...)）做为其子表达式。

表 5-11: 不支持的列表达式结构类型

不支持的列表达式结构类型	描述
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 。
SELECT a & b FROM A, B	不支持位操作。
SELECT a + COUNT(*) FROM A	不支持普通表达式（+）同时嵌套普通表达式（a）和聚合表达式（COUNT(...)）。
SELECT SUM(COUNT(*))	不支持聚合表达式嵌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）

FROM/JOIN子句的基本结构如下：

```
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_condi
tions
```

FROM/JOIN子句支持和不支持的结构分别如表 *5-12: FROM/JOIN*支持的结构类型和表 *5-13: FROM/JOIN*不支持的结构类型所示。

表 5-12: FROM/JOIN支持的结构类型

FROM/JOIN支持的结构类型	描述
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**： 在JOIN 子查询中，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 ...)	表和子查询之间的JOIN。
SELECT ... FROM (SELECT ...) JOIN (SELECT ...)	两个子查询之间的 JOIN。

表 5-13: FROM/JOIN不支持的结构类型

FROM/JOIN不支持的结构	描述
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	不支持没有ON条件的单表或多表连接。
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，不支持。

两个表关联可运行的充要条件（四原则）如下：

- 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）中的条件两端均有效的 HashMap 索引。

两个表关联可加速运行的条件如下：

- 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。
- 如果使用Full MPP Mode引擎，则支持任意形态的表关联。

WHERE 子句（过滤表达式）

在WHERE 子句（过滤表达式）中，单个条件之间可以用 AND 和 OR 进行连接、支持通过括号来决定条件表达式的优先级、支持表达式嵌套。与SQL一样，WHERE子句中的单个条件即是一个boolean型的判断，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：在 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。

AnalyticDB支持 = 、 >= 、 > 、 <= 、 < 、 <> 、 != 等基本比较操作符，也支持 IS、IN、CONTAINS、BETWEEN...AND... 、LIKE 等关系操作符，并且关系操作符均支持其反义的版本（IS NOT 、 NOT IN 、 NOT LIKE 等）。

在单个条件中，boolean主体可以是一个列，也可以是表达式的嵌套，其支持和不支持的结构分别如表 *5-14: boolean*主体支持的结构类型和表 *5-15: boolean*主体不支持的结构类型所示。

表 5-14: boolean主体支持的结构类型

boolean主体支持的结构类型	描述
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)	对于表 dim_table 是维度表，支持 IN 后带子查询。
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。

boolean主体支持的结构类型	描述
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。

表 5-15: boolean主体不支持的结构类型

boolean主体不支持的结构类型	描述
WHERE EXISTS (SELECT...)	不支持。
WHERE a = ANY (SELECT ...)	不支持。
WHERE a = ALL (SELECT ...)	不支持。
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 GROUP BY/HAVING 子句（分组和分组后过滤）。

GROUP BY 子句的用法

GROUP BY用于对查询结果进行分组，其基本结构如下：

```
GROUP BY {col_expr}
```

col_expr 为列，可以是表中的列、子查询或别名产生的列。GROUP BY其支持的结构如表 *5-16: GROUP BY*支持的结构类型所示。

表 5-16: GROUP BY支持的结构类型

GROUP BY支持的结构类型	描述
... 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。

GROUP BY支持的结构类型	描述
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 。
... 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）。

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，它支持使用列投射和聚合表达式后的结果进行过滤。

HAVING 子句的基本结构如下：

```
HAVING having_condition
```

HAVING子句支持的结构如表 *5-17: HAVING子句支持的结构类型*所示。

表 5-17: HAVING子句支持的结构类型

HAVING子句支持的结构类型	描述
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。
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）

ORDER BY用于对查询结果进行排序，其支持的结构如表 *5-18: ORDER BY支持的结构类型*所示。

表 5-18: ORDER BY支持的结构类型

ORDER BY支持的结构	描述
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+b）排序并返回TOP-N（100）。

ORDER BY支持的结构	描述
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。
SELECT a + b, COUNT(1) AS x FROM A GROUP BY a + b ORDER BY x LIMIT 100	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 无任何限制。

LIMIT 子句（返回行数限制）说明如下：

- SELECT ... LIMIT 100：返回行限制为100 行。
- SELECT ... LIMIT 100, 50：带偏移量的返回行限制，表示从第100行开始，显示后面的50行。

> **说明**：
> AnalyticDB对SELECT查询返回的行数做了全局的最大限制，目前最大支持10000 行。如果未带LIMIT或LIMIT行数超过10000，则只会返回 10000 行。

5.8.2 逻辑表达式和特殊语法

表JOIN的条件

在AnalyticDB中，两个事实表进行JOIN时需要满足以下全部条件：

- 两个事实表在同一个表组中。
- 两个事实表的JOIN KEY是HASH分区列。
- 两个事实表的HASH分区数必须一致，否则JOIN结果不准确。
- 两个事实表的JOIN KEY至少有一列建立了HashMap索引，建议在数据量小的表中建立。

维度表参与的关联，只需要符合上述条件之一即可。

AnalyticDB支持的逻辑表达式如表 *5-19:* 逻辑表达式所示。

表 5-19: 逻辑表达式

逻辑表达式	描述	示例
NOT a	• 如果表达式（a）是0，则返回1。	• NOT 0：1 • NOT 5：0

逻辑表达式	描述	示例
	• 如果表达式（a）是非0，则返回0。 • 如果表达式（a）是NULL，则返回NULL。	• NOT NULL：NULL • NOT a：根据表达式（a）的取值决定
!a	• 如果表达式（a）是0，则返回1。 • 如果表达式（a）是非0，则返回0。 • 如果表达式（a）是NULL，则返回NULL。	• ! 0：1 • ! 5：0 • ! NULL：NULL • ! a：根据表达式（a）的取值决定
a && b	• 如果表达式（a）和（b）都不是NULL且都非0，则返回1。 • 如果表达式（a）和（b）其中有一个是0，则返回0，否则返回NULL。	• 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
a || b	• 如果表达式（a）和（b）都是0，则返回0。 • 如果表达式（a）和（b）都不是NULL则返回1。 • 如果表达式（a）和（b）其中有一个为NULL，另外一个是0，则返回0。 • 如果表达式（a）和（b）是非0则返回1。 • 如果表达式（a）和（b）都是NULL，则返回NULL。	• 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AnalyticDB中，多值列有其特殊的用法和限制。在SELECT/GROUP BY中使用多值列时，您必须通过WHERE子句的IN/CONTAINS对数据进行枚举筛选，否则您无法使用多值列。举例说明如下：

举例	描述
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, COUNT(a) FROM A GROUP BY m	m为多值列时不支持。
SELECT m FROM A limit 10	m为多值列时不支持。

多值列的数据结构特殊，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，COUNT/SUM等聚合函数可能会多次计算相同的单值。例如：对表 *5-20:* 原始数据执行`SELECT m, COUNT(a) FROM A WHERE m IN('foo', 'bar') GROUP BY m`后，结果如表 *5-21:* 查询结果所示。

表 5-20: 原始数据

a	m
0	'bar'
1	'foo', 'bar'
2	'foo', 'bar', 'oh'

表 5-21: 查询结果

m	COUNT(a)
'foo'	2
'bar'	3
'oh'	1

AnalyticDB是分布式实时OLAP系统，其数据根据分区列分布在不同的物理服务器中，在一些特殊的情况下，分区列和非分区列在使用上有所不同。

- COUNT(DISTINCT cor_expr)：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...：如果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）：如果同时和非分区列的GROUP BY子句连用，结果可能不精确。

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;
```

详细信息请参见 多计算引擎和 *Hint* 。

5.8.3 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
```

```
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。

5.8.4 多计算引擎和Hint

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）：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）：新增的计算引擎，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

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。

5.9 Data Pipeline

5.9.1 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。

5.9.2 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 语法如下所示。

```
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。

5.9.3 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。授权命令如下所示。

```
--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。语法格式如下所示。

```
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）。语法格式如下所示。

```
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
```

- 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。

5.10 用户与权限

5.10.1 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：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。
- **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。

5.10.2 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 5-22: 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。

5.10.3 ACL权限管理

与MySQL相似，AnalyticDB的数据库创建者可以通过GRANT/REVOKE语句对其他用户进行授权和权限回收。

5.10.3.1 授权

数据库创建全部完成后，您可以对每个使用该数据库的用户进行授权，以使每个用户都能以最小的权限来完成日常工作。本节将向您展示如将数据库的操作权限授权给用户。

授权语句

授权语句如下：

```
GRANT privilege_type [(column_list)] [, privilege_type [(column_list
)]] ... ON [object_type] privilege_level TO user [, user] ...
```

返回值为空。

关键参数说明如下表所示。

表 5-23: 参数描述

参数名称	
privilege_type[(column_list)]	具体的权限类型，[(column_list)]为可选。当对象级别为表时，[(column_list)]可以填写列将具体列的相关权限授予用户。
[object_type]	权限对象的类型，如 Database、Table、TableGroup 等。建议填写object_type为授权对象的表达式，详细信息请参见分析型数据库权限模型 。

参数名称	
user	被授权用户，格式为： `'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'@'%'。 • host_name目前只支持 % ，即不支持指定HOST。host_name为可选项，可以省略。即'user_name'[@'host_name']可以写成'user_name'，等价于'user_name'@'%'。

授权示例

将表组的describe、select权限授权给test_user@aliyun.com用户，示例如下：

```
GRANT describe, select ON tablegroup db_name.table_group_name TO '
ALIYUN$test_user@aliyun.com'@'%';
```

将列的describe、select权限授权给test_user@aliyun.com用户，示例如下：

```
GRANT describe, select (col1, col2) ON table db_name.table name TO '
ALIYUN$test_user@aliyun.com';
```

5.10.3.2 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。

5.10.3.3 查看用户权限

AnalyticDB提供查看用户权限的SQL语句，以便您进行权限管理。

查询用户权限语句如下：

```
SHOW GRANTS [FOR user] ON [object_type] privilege_level
```

如果[FOR user]未填写用户或是填写了当前用户，则列出当前用户在指定数据库上被授予的权限。
如果[FOR user]填写其他用户，则列出当前用户在指定数据库上授予[FOR user]用户的权限。

返回值如下：

```
'GRANT ALL ON db_name.* TO 'user'
```

```
GRANT SELECT(column_name) ON db_name.table_name TO 'user'
```

5.10.3.4 其他相关内容

- **权限的默认约定如下所示。**
  - 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。

5.11 在生产中使用分析型数据库

5.11.1 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**：5.1.1，5.1.2，5.1.3，5.1.4，5.1.5，5.1.6，5.1.7，5.1.8，5.1.11，5.1.12，5.1.13，5.1.14，5.1.15，5.1.16，5.1.17，5.1.18，5.1.19，5.1.20，5.1.21，5.1.22，5.1.23，5.1.24，5.1.25，5.1.26，5. 1.27，5.1.28，5.1.29，5.1.31。
- **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。

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。

5.11.2 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/服务器连接串和表名与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。

5.11.3 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。

5.11.3.1 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。

5.11.3.2 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）。

5.11.3.3 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）。

5.11.3.4 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。

5.11.3.5 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
```

5.11.4 在BI工具中进行连接和使用

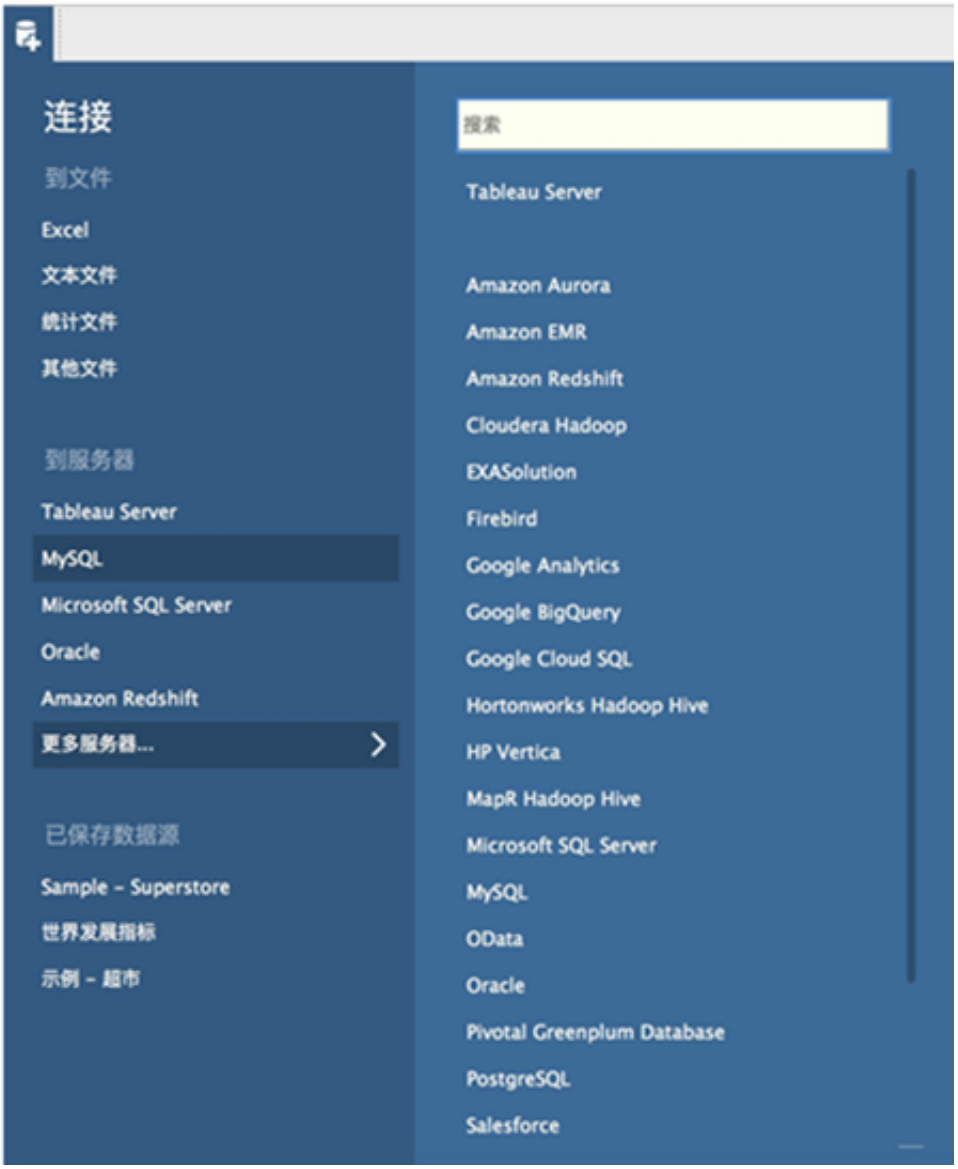
目前，阿里云分析型数据库能够支持在部分BI工具中以MySQL连接的方式进行连接和使用。

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。

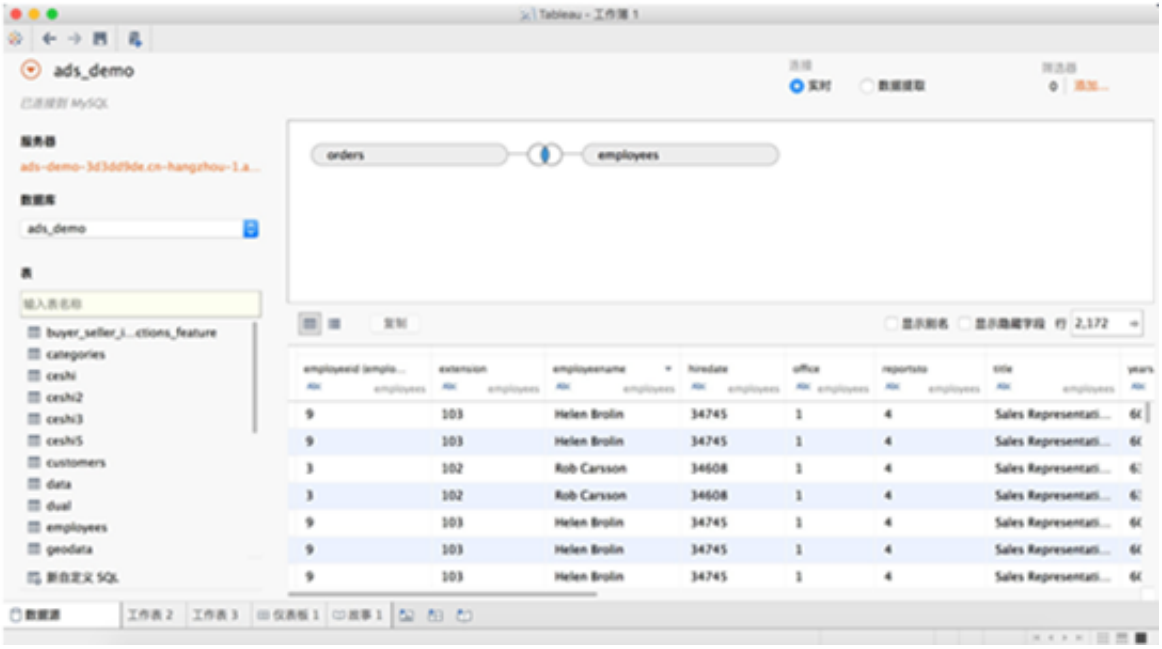
1. 打开Tableau Desktop，单击**添加新的数据源**并选择数据源类型为MySQL。



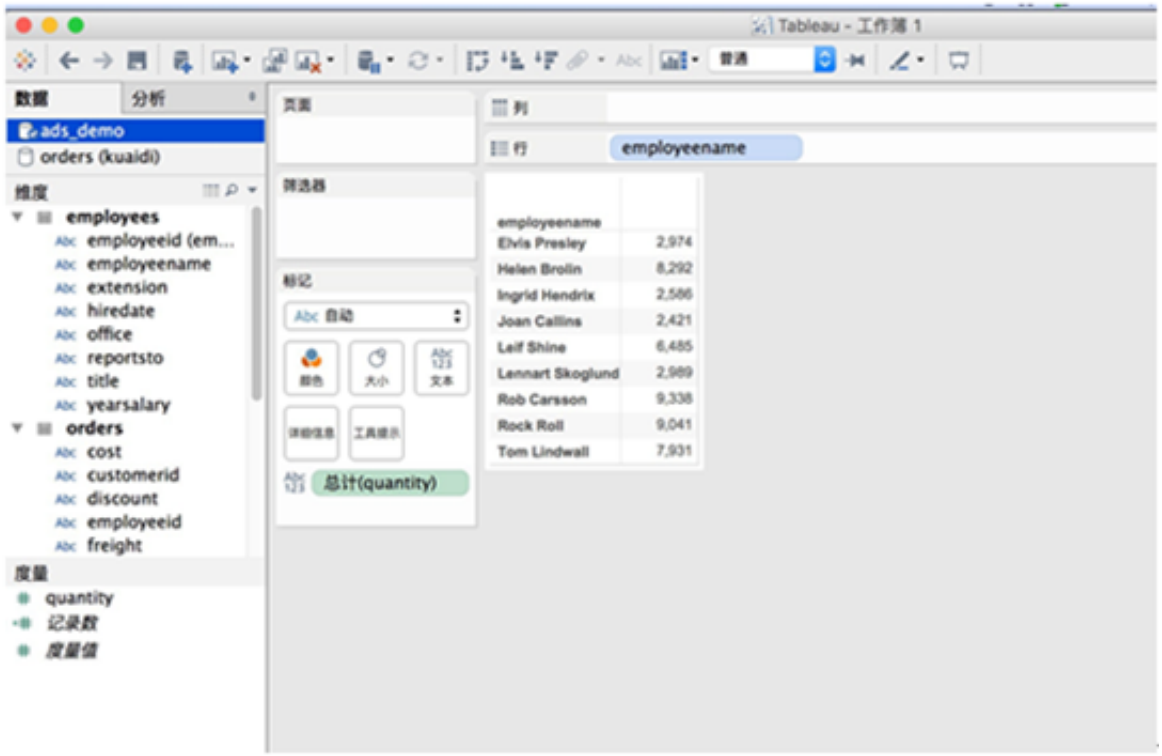
2. 在弹出的对话框中，输入您要访问的数据库的域名和端口号（在DMS的数据库选择页面获取），以及将您的账号在阿里云的Access Key ID / Access Key Secret输入到用户名/密码框中，然后单击**确定**。

3. 此时在Tableau的数据源页面上，可以看到您有权限的数据库，选择后可以进行数据表选取，以及数据预览。

> **说明**：
> 这里只能选择您连接到的域名端口所对应的数据库。

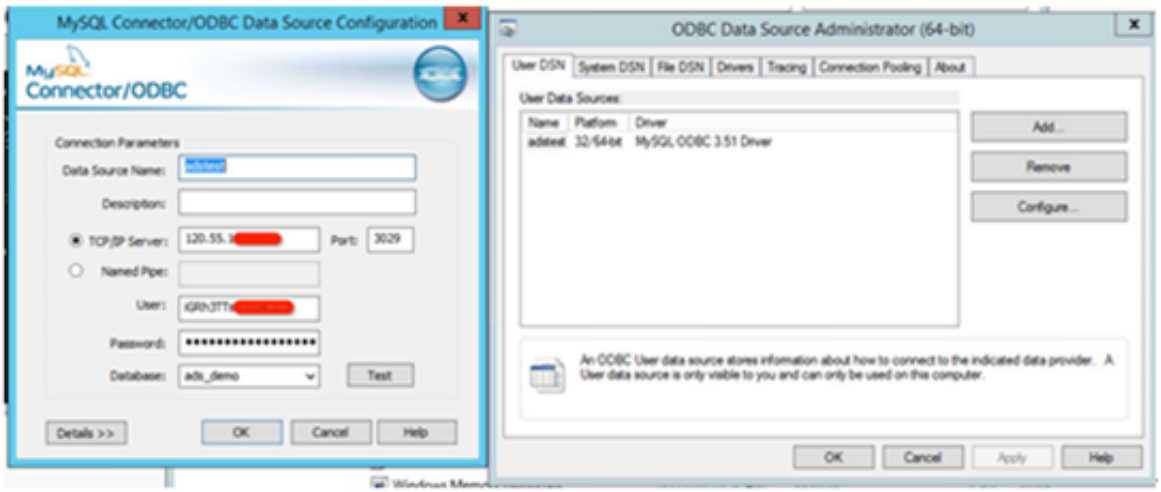


4. 完成上述操作即可进行工作表的建立以及更多其他功能。



在QlikView中使用分析型数据库

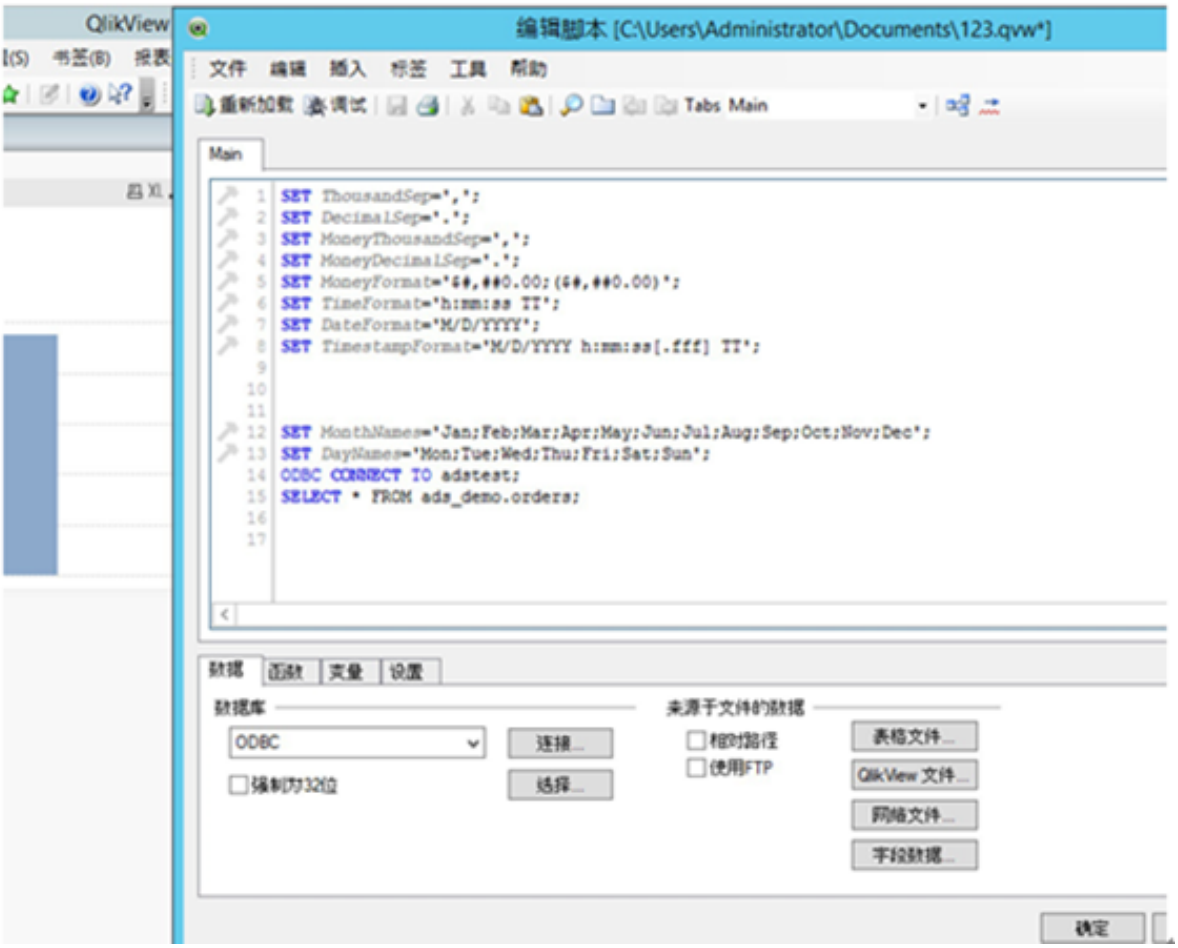
1. 要使用QlikView连接分析型数据库，首先需配置操作系统的ODBC数据源。安装OBDC MySQL Driver后，单击**控制面板** > **管理工具** > **ODBC数据源**（不同操作系统此步骤可能不同），新建一个用户DSN，选择**`MySQL ODBC 5.xx Driver`**。



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**，在末端键入以下信息。

```
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。

5.12 附录

5.12.1 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。

5.12.1.1 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

5.12.1.2 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''

FIELD	TYPE	是否为扩展字段	COMMENT
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
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
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

FIELD	TYPE	是否为分析型数据库扩展字段	COMMENT
			/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''
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

FIELD	TYPE	是否为分析型数据库扩展字段	COMMENT
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
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

FIELD	TYPE	是否为分析型数据库扩展字段	COMMENT
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

5.12.2 附录二：SQL函数表

5.12.2.1 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_CO
UNT_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_CO
UNT_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_ge
o_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。

5.12.2.2 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。

5.12.2.3 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()

5.12.2.4 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

5.12.3 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 executiono f 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	ShowSyntax Exception: You have an error in your SQL syntax	检查SHOW语句语法
权限异常	1142	42000	DenyAccess Exception:User [#us er#] does not have[#action #] access to resource[#reso urce#]	确认用户的执行权限
权限异常	1142	42000	NotAllowAc essException: User[#user#] does not have[# action#] access to resource[#reso urce#]	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[#reso urce#]	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

类别	错误码	SQL状态码	错误信息	处理方法
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_colum n_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
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	修改SQL，目前分布式查询模式下，UNION分支中LIMIT无意义

类别	错误码	SQL状态码	错误信息	处理方法
			SELECT ... LIMIT X	，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=#parametername#	检查DDL语句参数
DDL异常	3081	HY000	IllegalParameterException: paramaterNam e=Column type is invalid:#data_type#	检查DDL列的数据类型
DDL异常	3082	HY000	IllegalParameterValueExceptio n: [DDL- N3000]:groupName should be provided for DIMENSION tb.	检查DDL语句参数，为维度表指定groupName
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	IllegalParameterValueExceptio n: [DDL- N3002]: hashPartitionNum should be provided and set	检查DDL语句参数，为维度表指定 hashPartitionNum，值必须为1

类别	错误码	SQL状态码	错误信息	处理方法
			value to 1 for DIMENSION tb.	
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 virtual dimension tb group.	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
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	IllegalParameterValueException: [DDL- N5000]:	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
			originalTableName should not be null.	
DDL异常	3082	HY000	IllegalParameterValueException:[DDL-NFFFF] Unexpected exception:#exception message#	检查DDL语句参数
DDL异常	3082	HY000	IllegalParameterValueException: parameterName=#parameterName#, parameterValue=#parameterValue#, correct type is #type#	检查DDL语句参数
DDL异常	3082	HY000	IllegalParameterValueException: #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	IllegalParameterValueException: loader.originalTableName is not null	检查DDL语句参数，指定originalTableName
DDL异常	3082	HY000	IllegalParameterValueException: loader.origina	检查DDL语句参数，目标originalTableName已存在，修改

类别	错误码	SQL状态码	错误信息	处理方法
			lTableName conflict	originalTableName
DDL异常	3082	HY000	IllegalParameterValueException: valueTypemust is lllegal	检查DDL语句参数valueType
DDL异常	3082	HY000	IllegalParameterValueException: valueTypeis 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	CreateTableOperationFailed:Table'#schema#.#tableName#' already exists	目标表已经存在，无需执行
DDL异常	3086	HY000	CreateTableOperationFailed:puttableInfo failed, sql:#DDL statement#	联系阿里云技术支持
DDL异常	3086	HY000	CreateTableOperationFailed: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!	目标表不存在，确认表的正确性

类别	错误码	SQL状态码	错误信息	处理方法
			tablePath:#tb path#	
DDL异常	3000	HY000	AlterRepea tException: indexName:# indexNam e#### repeat	目标index已存在
DDL异常	3092	HY000	DescTableS qlFormatEx cpetion:check sql	检查并修正SHOW语法，重新执行
DDL异常	3094	HY000	DropSyntax Exception:# DDLstatement#	检查并修正DROP DDL语法，重新执行
DDL异常	3096	HY000	AlterSynta xException:#DDL statement#	检查并修正ALTER DDL语法，重新执行
DDL异常	3099	HY000	ExceedMaxT ableNumExc eption: maxTableNu mInDB=# maximun 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	SQLplannin g partition routing error: ZERO_ROUTE	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	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
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

类别	错误码	SQL状态码	错误信息	处理方法
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

6 大数据应用加速器DTBoost

6.1 什么是DTBoost

阿里云DTBoost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，让不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。这种方式有以下好处：

- 可屏蔽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。
- 通过数据服务的形式，有助于IT部门对数据使用进行管理，避免资源的重复和冗余。

6.2 登录DTBoost

本章将为您介绍如何登录DTBoost。

背景信息

- 您需要先从天基上获取DTBoost的域名，然后用获取后的域名登录DTBoost控制台。
- 如果您是首次登录DTBoost，需创建工作空间。如果您不是首次登录，可直接进入DTBoost**概览**页面。

操作步骤

1. 登录天基。
2. 在菜单栏中单击**运维** > **集群运维**，进入**集群运维**界面，查询DTBoost集群。

图 6-1: 集群运维界面

3. 单击查询到的DTBoost集群，进入**集群Dashboard**页面。
4. 在**集群资源**区域框中选中**app**为**dtb-dtboost-server**、**type**为**dns**的行，右键单击该行的**parameters**单元格，选择**显示更多**，弹出**详情**提示框，在提示框中查看该DTBoost集群的域名。

图 6-2: 查看域名

5. 复制DTBoost域名到浏览器地址栏，按照系统提示登录DTBoost控制台。
6. 单击**我是主账号**，进入**工作空间配置管理** > **工作组列表**页面。
7. 单击**新建**。
8. 根据页面提示创建工作空间，工作空间创建完成后自动跳转至DTBoost**概览**页面。
9. 单击左侧导航栏中的相应功能，即可进行后续操作。

图 6-3: 功能页面

6.3 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 *6-4:* 主界面所示，您可以单击主界面 DTBoost 导航栏上的**标签中心**选项卡进入。

图 6-4: 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。

6.3.1 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。

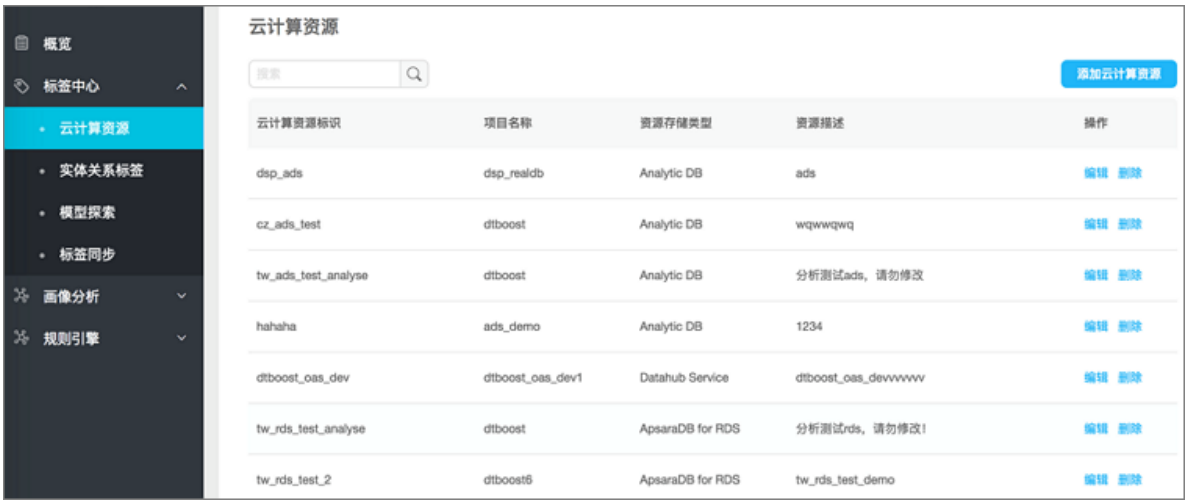
- 阿里云大数据计算服务（MaxCompute）
- 阿里云分析型数据库（AnalyticDB）

操作步骤

1. 单击**云计算资源管理**页面右上角的**添加云计算资源**，添加新的云计算资源。

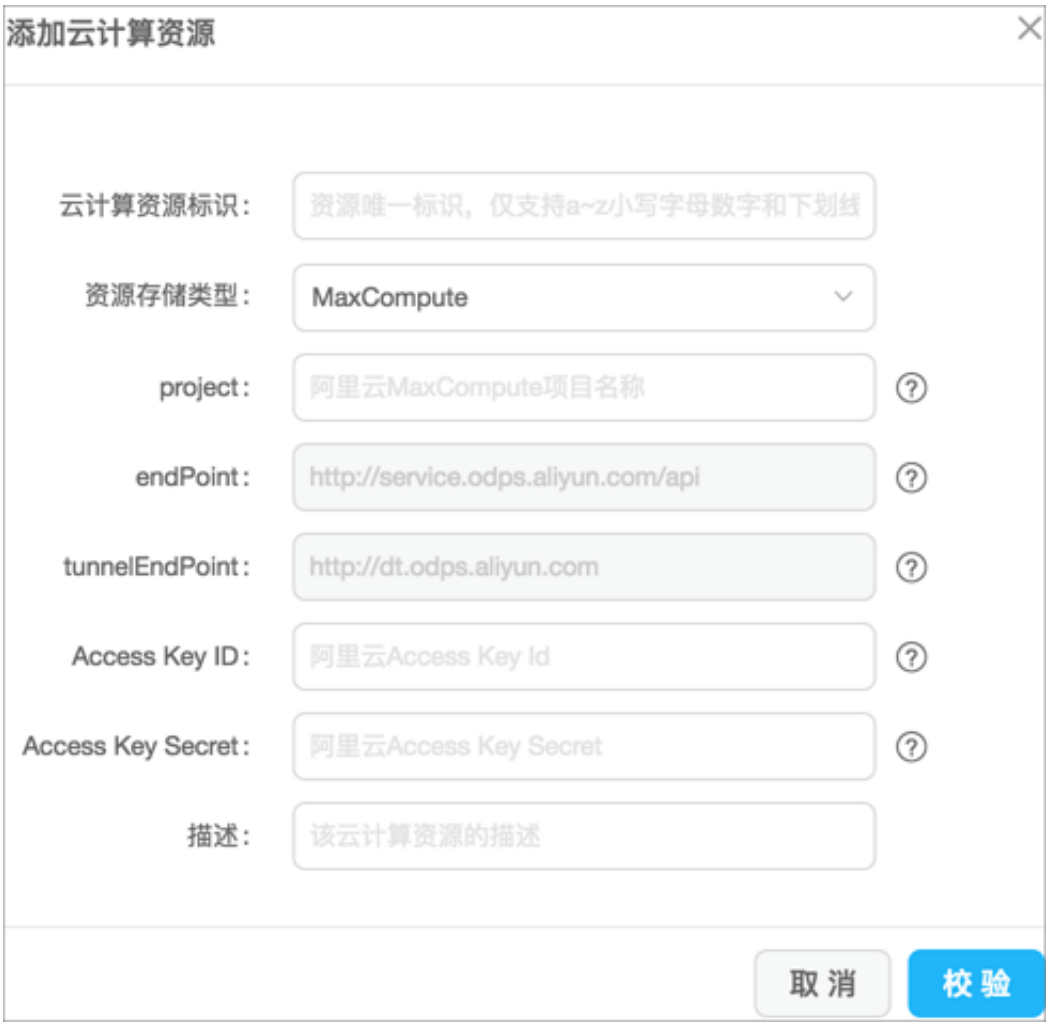
   说明：
   您也可以在该界面查看您已经注册的云计算资源，如图 *6-5:* 添加云计算资源所示。

图 6-5: 添加云计算资源



2. 在**添加云计算资源**面板中，如图 *6-6:* 填写相关参数所示，填写以下内容。

图 6-6: 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：填写对应的登录密钥ID（可向管理员索要）。
- Access Key Secret：填写对应的登录密钥Secret（可向管理员索要）。
- 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。

6.3.2 实体关系建模

背景信息

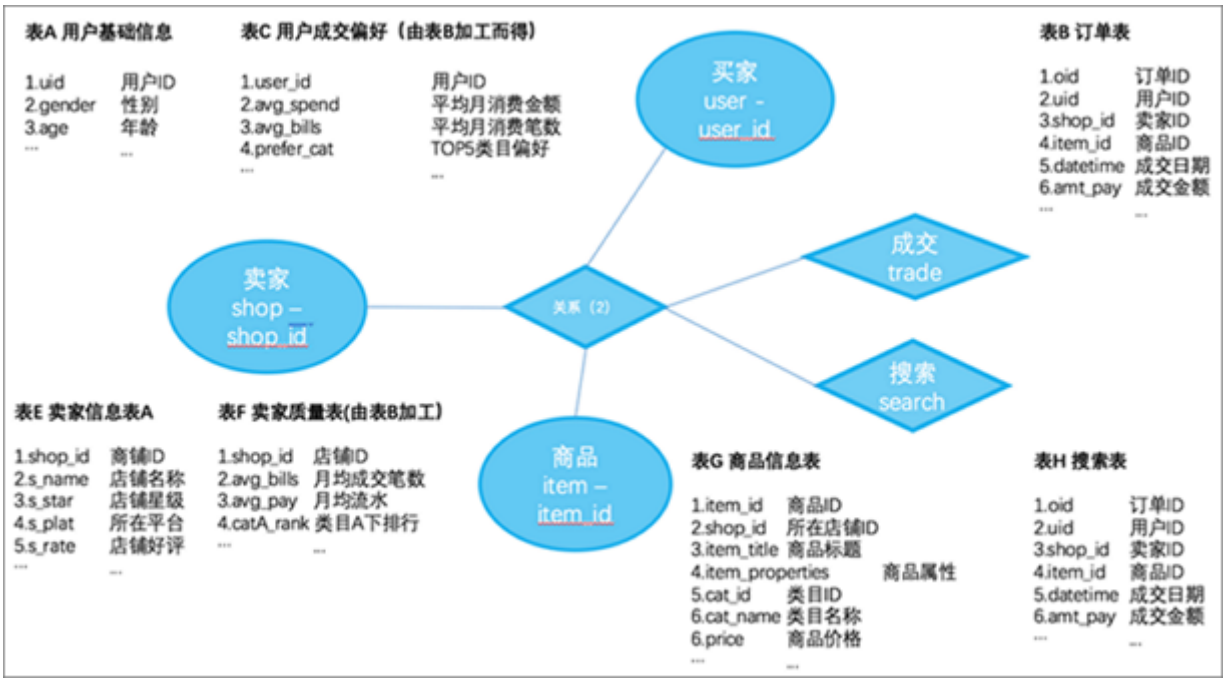
实体关系建模是把分散的多张表，以业务形态进行组织的一种方式，即围绕着一个或多个实体（即描述的对象）的各种属性和行为的描述。整个建模围绕**实体**、**关系**和**标签/属性**三个主要概念展开。

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 *6-7:* 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 6-7: 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**（即原始表，通过各实体和关系裁剪成小表）。

6.3.2.1 配置实体

前提条件

添加实体前，请确认已添加云计算资源，详情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**页面，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 6-8: 实体关系标签界面

2. 单击页面右上角的**新建**。
3. 填写实例相关信息后，单击**确定**。

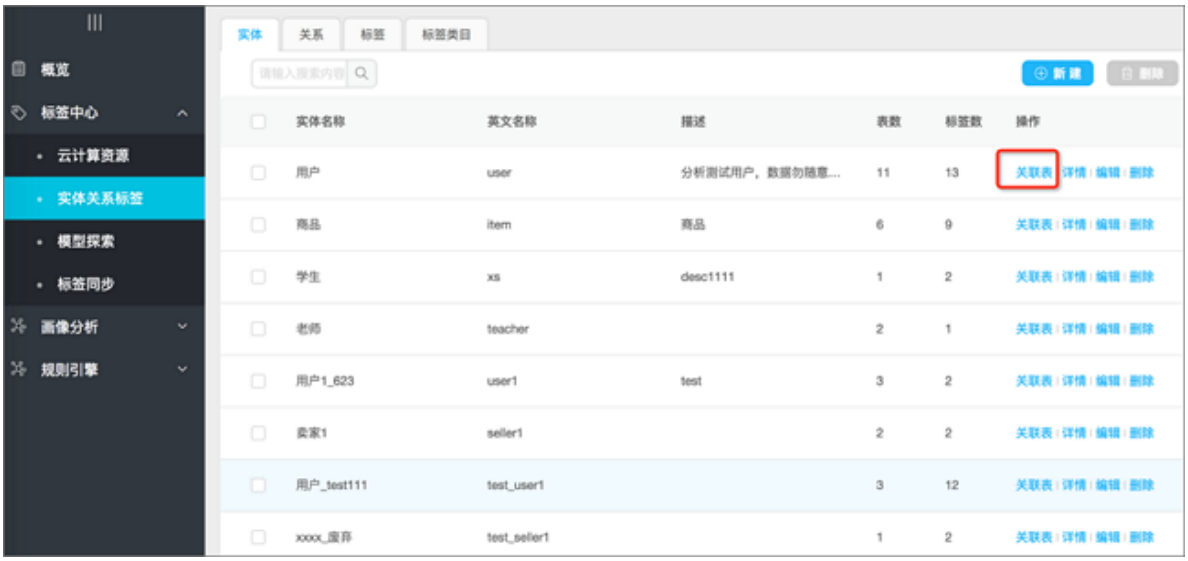
   例如：买家，您可以将其命名（要求唯一），中文名称为**用户**，英文识别名称为**user**，同时给这个逻辑实体命名一个抽象的**主键**，例如**user_id**，如图 *6-9:* 填写实体信息所示。

图 6-9: 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**页面，单击右侧的**关联表**，如图 *6-10:* 新增关联表所示。

图 6-10: 新增关联表



5. 选择**标签设置**窗口中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 *6-11: 添加实体关联字段*所示。

图 6-11: 添加实体关联字段

6.3.2.2 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 *6-12: 标签数*所示。

   单击**标签管理**，会对该表设置新的标签。

   图 6-12: 标签数

2. 根据需要，设置标签，如图 *6-13: 设置标签*所示。

   图 6-13: 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**。

6.3.2.3 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. 单击**实体关系标签**页面中的**关系**选项卡，如图 *6-14: 关系选项卡*所示。

图 6-14: 关系选项卡

2. 单击**关系**选项卡右上角的**新建**。

   显示**新建关系**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 *6-15: 新建关系*所示。

图 6-15: 新建关系

4. 单击**关系的实体区域框**中的**关联表**。
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 *6-16: 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 6-16: 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。

6.3.3 数据整合与同步

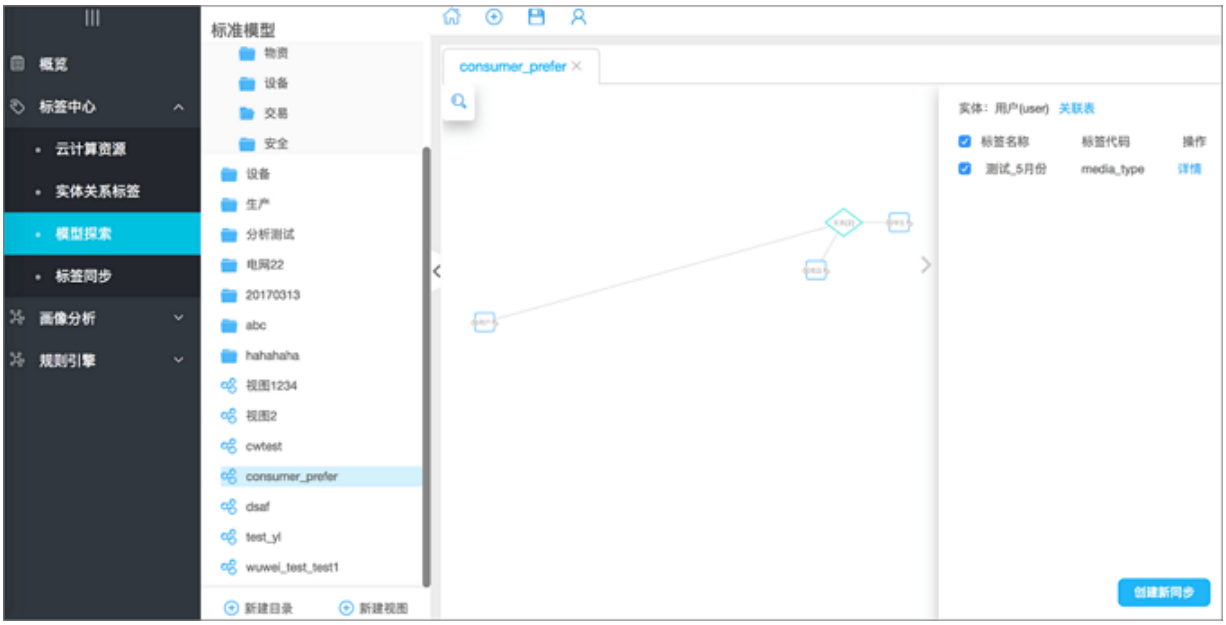
背景信息

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，即可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 *6-17: 模型探索界面*所示，方块表示实体，菱形和圆代表关系。

图 6-17: 模型探索界面



操作步骤

1. 单击**模型探索**页面的某一个实体/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 *6-18: 确认待同步的标签*所示。

   图 6-18: 确认待同步的标签

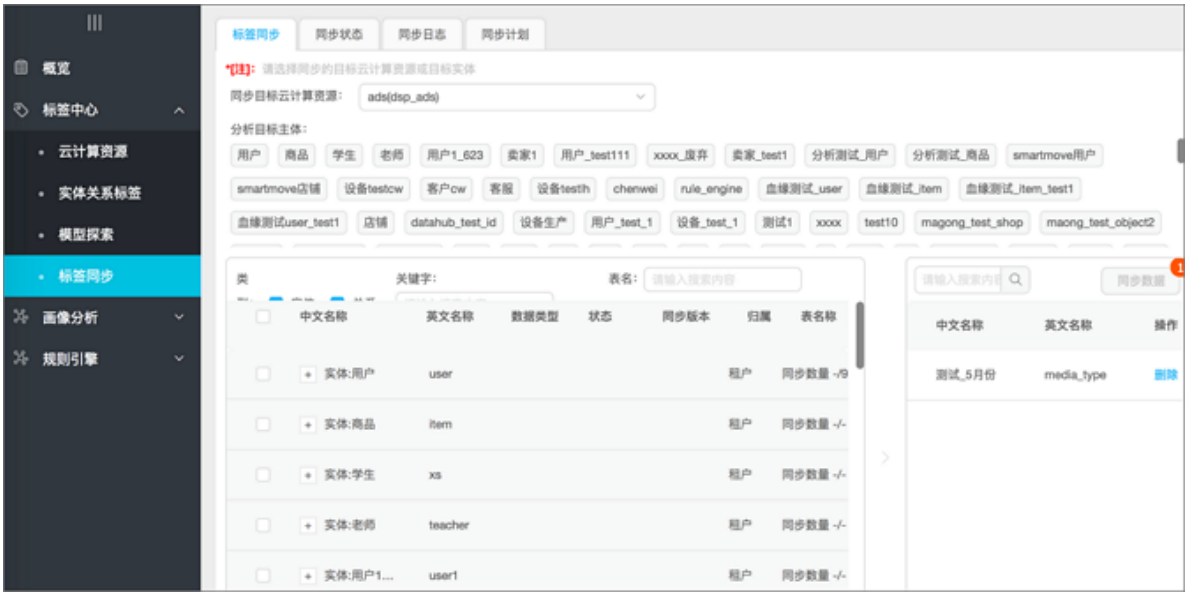
3. 选择同步目标，即选择您需要把数据进行集中同步的存储计算资源，如图 *6-19: 选择同步目标*所示。

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 6-19: 选择同步目标



4. 单击**下一步**。
5. 选择同步方式，如图 *6-20: 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 6-20: 选择同步方式

6. 完成后，单击**发起同步**。

6.4 画像分析

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。您也可以在产品当中自助封装API接口，以便您更方便地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。

6.4.1 接口调试

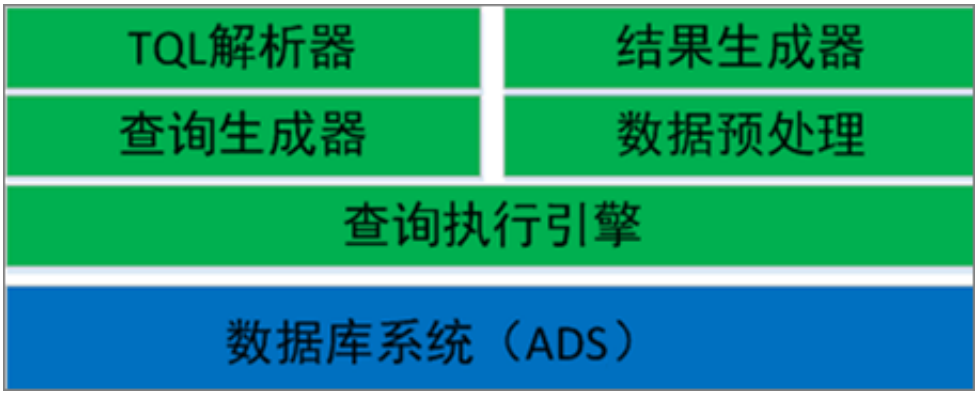
6.4.1.1 接口架构

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 *6-21:* 总体架构所示。

图 6-21: 总体架构

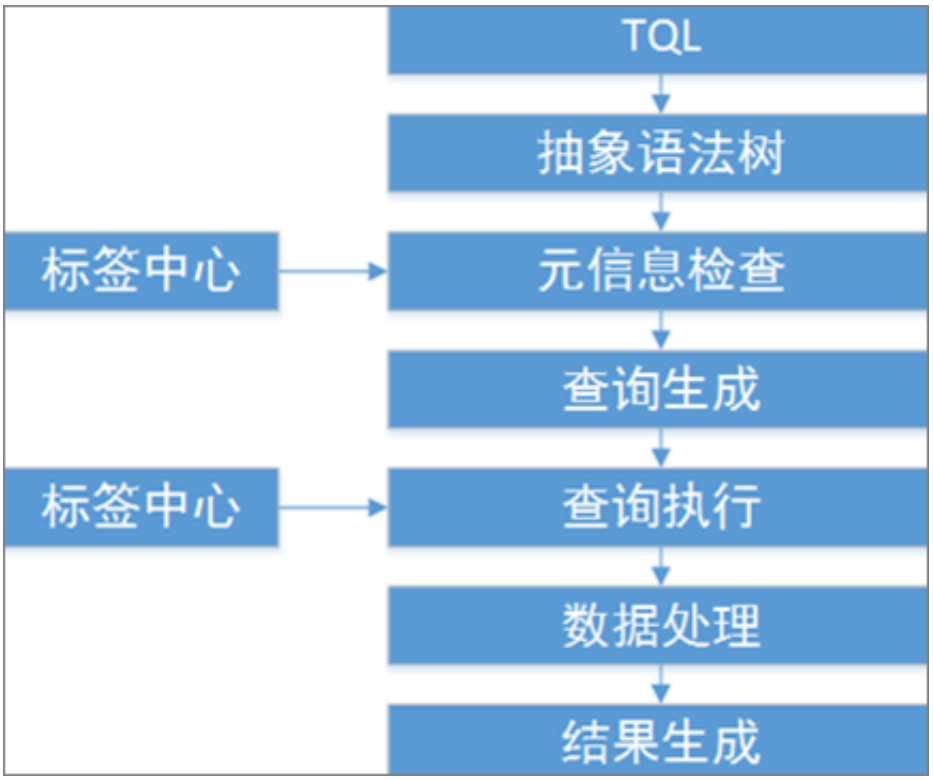


整合分析各模块的主要功能如下：

- 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 *6-22:* 查询执行流程所示。

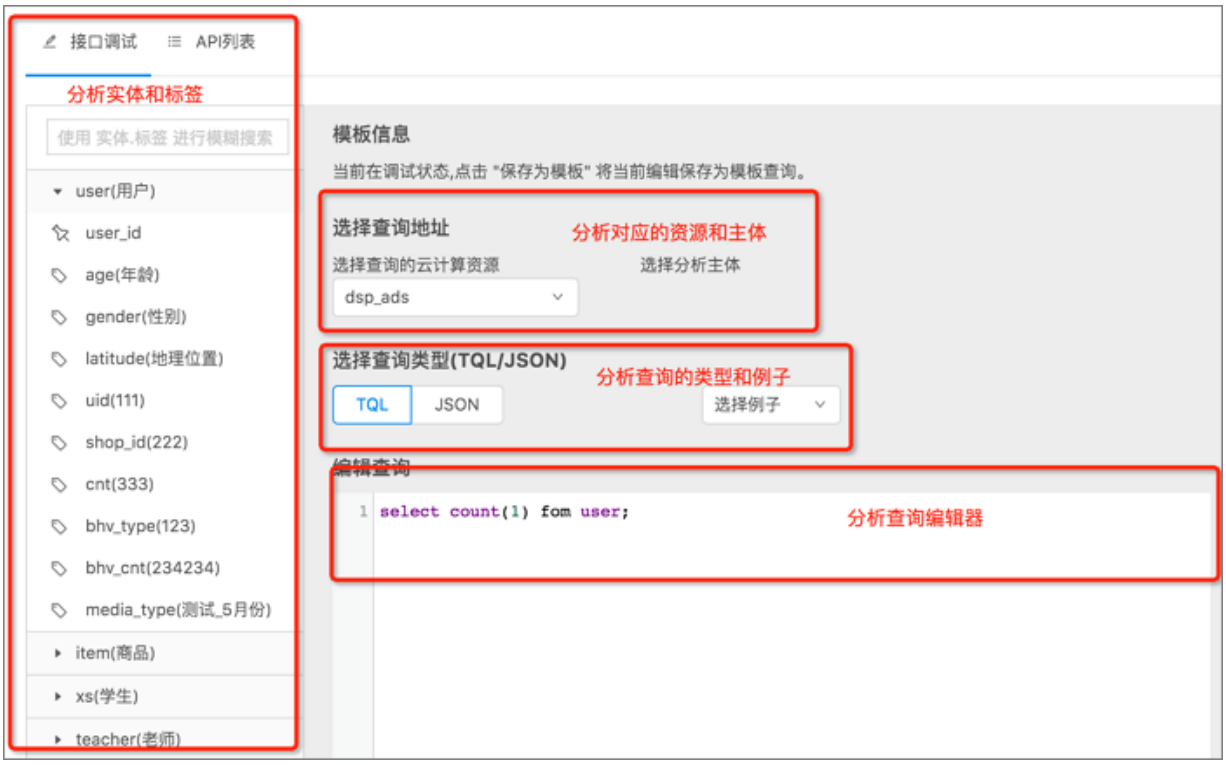
图 6-22: 查询执行流程



6.4.1.2 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 6-23: 调试界面



如图 *6-23: 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 *6-24: 调试结果*所示。

图 6-24: 调试结果

6.4.1.3 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。

6.4.2 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 *6-25: 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 6-25: 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**，即可查看已经创建的应用列表。

- 批量下载应用

  单击**界面配置**页面的**批量下载**后，选中多个应用。

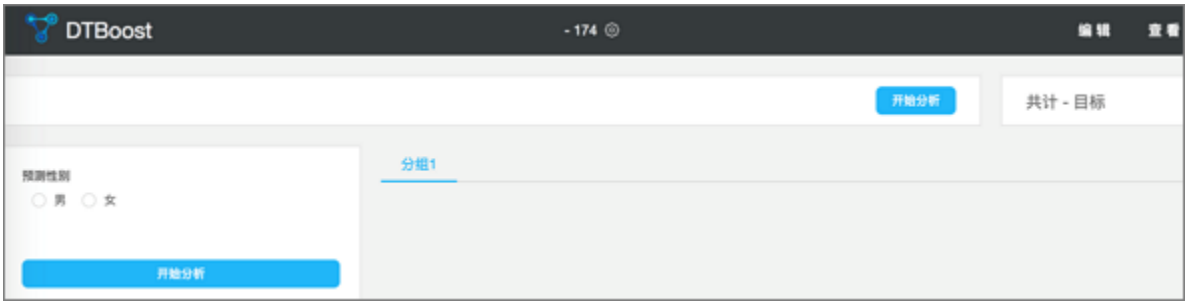
  单击**界面配置**页面左上角的**下载**，可以将多个应用打包下载。

  具体的部署方法请参见下载包中的说明手册。

- 分析应用

单击**界面配置**页面的**新建界面配置**，为应用命名后即进入如图 *6-26: 准备分析应用*所示界面，可以开始准备配置分析应用。

图 6-26: 准备分析应用



6.4.2.1 功能介绍

如图 *6-27: 分析应用*所示，为通过配置所得到的最终分析应用。

图 6-27: 分析应用

- 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，您可以单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其他存储当中。

6.4.2.2 配置说明

6.4.2.2.1 筛选条件配置

1. 单击**画像分析** > **界面配置**中的某个界面配置。
2. 单击显示的界面配置右上角的**编辑**按钮，默认进入筛选条件配置页面。

   如图 *6-28: 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 6-28: 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

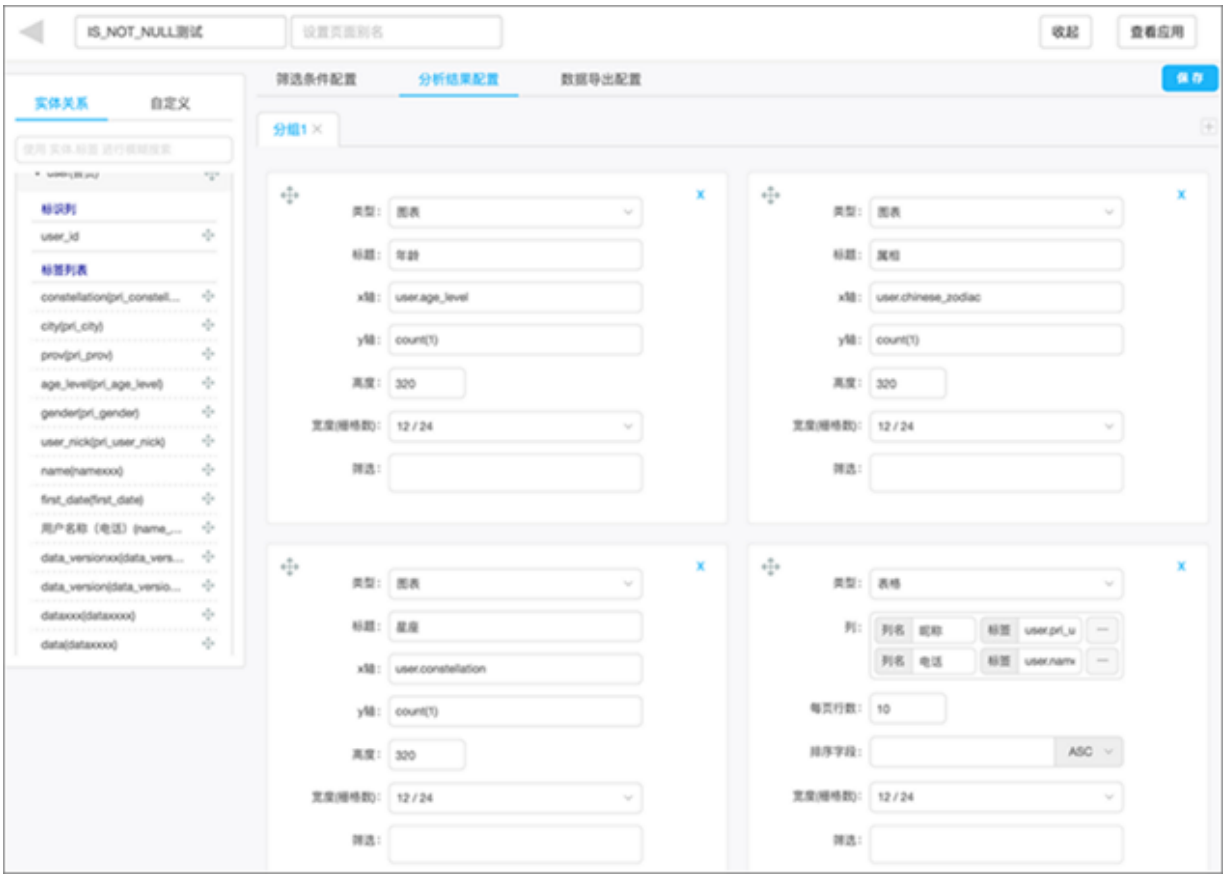
  下方您可以进行自定义子查询，自定义的查询也可以作为一个结点拖入到主查询逻辑当中作为子查询存在，方便您配置一些比较复杂的分析逻辑。

6.4.2.2.2 分析结果配置

1. 单击**画像分析** > **界面配置**页面中的某个界面配置。
2. 单击显示的界面配置右上角的**编辑**按钮，默认进入**筛选条件配置**页面。
3. 单击**分析结果配置**页签，进入**分析结果配置**页面。

如图 *6-29:* 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 6-29: 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。

  可以单击选项卡右侧的+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SQL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）。

6.4.2.2.3 数据导出配置

1. 单击**画像分析** > **界面配置**页面中的某个界面配置。
2. 单击显示的界面配置右上角的**编辑**按钮，默认进入**筛选条件配置**页面。
3. 单击**数据导出配置**，进入**数据导出配置**页面。

如图 *6-30:* 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 6-30: 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。

6.4.3 分析应用部署

详情请参见分析应用压缩包中的帮助文件。

6.5 常见问题

1. Q：为什么`select * from object`不支持？

   A：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. Q：为什么`select object.id from object`报错？

   A： 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. Q：为什么我的 TQL 语句报实体或者标签未找到或者同步？

   A：

   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. 进入**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，造成第2个问题的报错情况。

      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. Q：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？

   A：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. Q：为什么我的标签或者实体删除不掉？

   A：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。

6. Q：为什么在TQL中用in和Join操作得到的最终结果不一致？

   A：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。

7 大数据管家BCC

7.1 什么是大数据管家

大数据管家Big Data Manager（原名BCC）是为大数据产品量身定做的运维管理平台。当前运维的大数据产品包括MaxCompute（原名ODPS）、DataWorks（原名大数据开发套件）、AnalyticDB（原名ADS）、Quick BI、IPlus（关系网络分析）、OSS（对象存储）、Apsara（飞天操作系统），并为这些产品提供服务监控、日常自检、日志搜索、运维、配置以及服务特性运维功能，维护产品稳定性。

大数据管家以服务组件的形式给与产品提供运维功能，每个服务组件包含产品树结构、配置、自动化服务自检、工作流、包管理、日志搜索、指标信息和Metrics信息，还包括各服务组件自定义的一些功能。

通过大数据管家的赋能，专有云驻场人员可以轻松地管理大数据产品，比如查看大数据产品的运行指标，修改大数据产品运行配置，对大数据产品进行自检，搜索日志等功能。

7.1.1 服务组件功能介绍

7.1.1.1 产品树结构

产品服务树是产品的组织结构体现形式，每个服务组件（产品Product）都是一个独立存在的个体，都拥有任意多个服务（Service），所有服务的组织形成产品服务树。

当进入产品页面时，单击右上角的**服务树**，会左移出来一个新的页面，显示当前产品的服务树并定位到当前界面展示的服务或节点；如果当前服务或节点包含子服务，则还会显示当前服务或节点的所有子服务。服务树中含有服务列表以及对应的节点，通过单击服务或者节点同步切换左侧的产品的服务或者节点。

7.1.1.2 配置

产品服务配置是根据服务的实际需求针对已部署的产品服务进行的相关配置、操作，从而实现您的实际运维需求，包括工作流配置、服务配置，自检配置、指标配置、表配置、图配置、日志搜索配置、节点信息配置、采集插件配置、健康配置、基础配置、私有配置、配置项配置。

7.1.1.3 工作流

工作流是事先根据一系列过程规则定义处理流程和步骤的过程，是封装好的一种框架，能够自动执行的过程，它可以解决一些繁琐或者重复性工作。

7.1.1.4 监控

监控是对整个大数据管家平台所承载的产品的健康监控。监控按照服务树层次进行监控项管理和报警汇聚，并将监控结果显示在界面中，让您更清楚地知道服务的当前运行状态。服务与监控项是多对多的关系，一个服务上可以有多个监控项，一个监控项也可以部署到多个服务上。一个服务上的一个监控项我们称之为监控项实例。

7.1.1.5 包管理

包管理是专有云为了提供更方便地服务升级方案的产物，做到快速应用patch方案、自助升级，是轻量容器运维入口，方便运维，减少人为错误，提高系统稳定性。目前包管理功能支持包的推送和管理功能。

7.1.1.6 日志搜索

日志搜索是提供服务器日志关键字搜索功能。您在不同服务下根据关键字信息搜索对应的相应时间段内日志信息。

7.1.1.7 指标信息

指标信息是服务下各个指标数据的展示，您可以通过指标数据查看当前服务的资源使用情况等有效的运维数据。

7.1.1.8 Metrics信息

Metrics信息也是图信息，主要是服务下各个指标历史数据的展示，您可以通过Metrics数据查看服务一段时间内的资源使用情况等有效的运维数据。

7.1.1.9 产品特定功能

大数据管家中，每个组件提供的功能基本都包含在上述工能中，然而有部分功能出于产品服务的特殊性，专门定制运维功能页面，以表格或其他形式展现。您可以通过这些功能更好的运维产品，例如大数据管家服务组件下拓扑结构图，通过该拓扑可以大概了解大数据管家下依赖及管理运维组件及其当前运行状态。

7.1.2 大数据管家登录

7.1.2.1 获取大数据管家域名

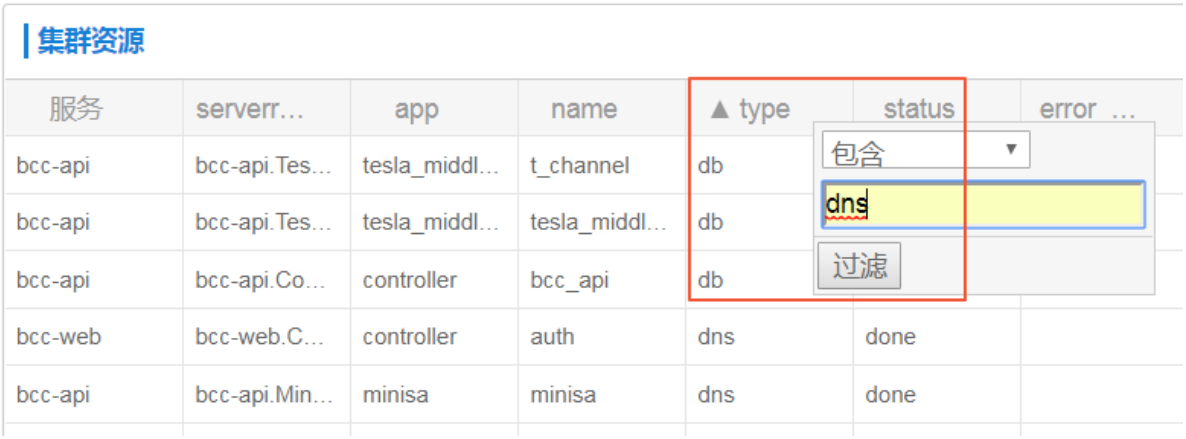
操作步骤

1. 登录天基，如图 *7-1:* 天基所示。

图 7-1: 天基

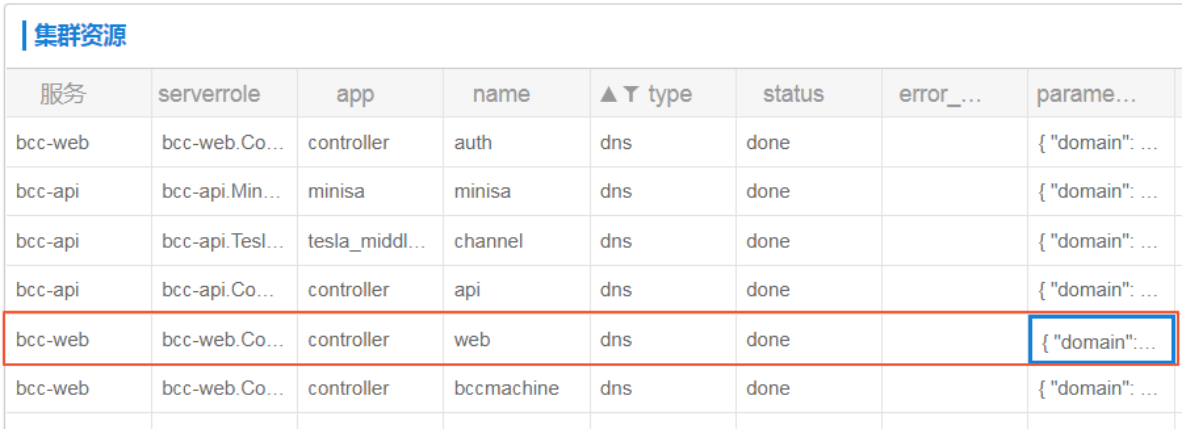
2. 在左侧选择**集群**，然后在**Project**下拉列表中选择bcc，筛选出大数据管家产品。
3. 选择筛选出的大数据管家产品，将鼠标指向后面的菜单项图标并选择**Dashboard**，进入大数据管家的**集群Dashboard**界面。
4. 在**集群资源**列表中，单击列表的**type**表头，以dns进行筛选，如图 *7-2: 筛选条件*所示。

图 7-2: 筛选条件



5. 在筛选结果中查找**name**为web的行，如图 *7-3: 筛选结果*所示。

图 7-3: 筛选结果



6. 右键单击该行的**parameters**单元格，并选择**显示更多**，在弹出的**详情**信息框中查看大数据管家的域名，如图 *7-4:* 详情所示。

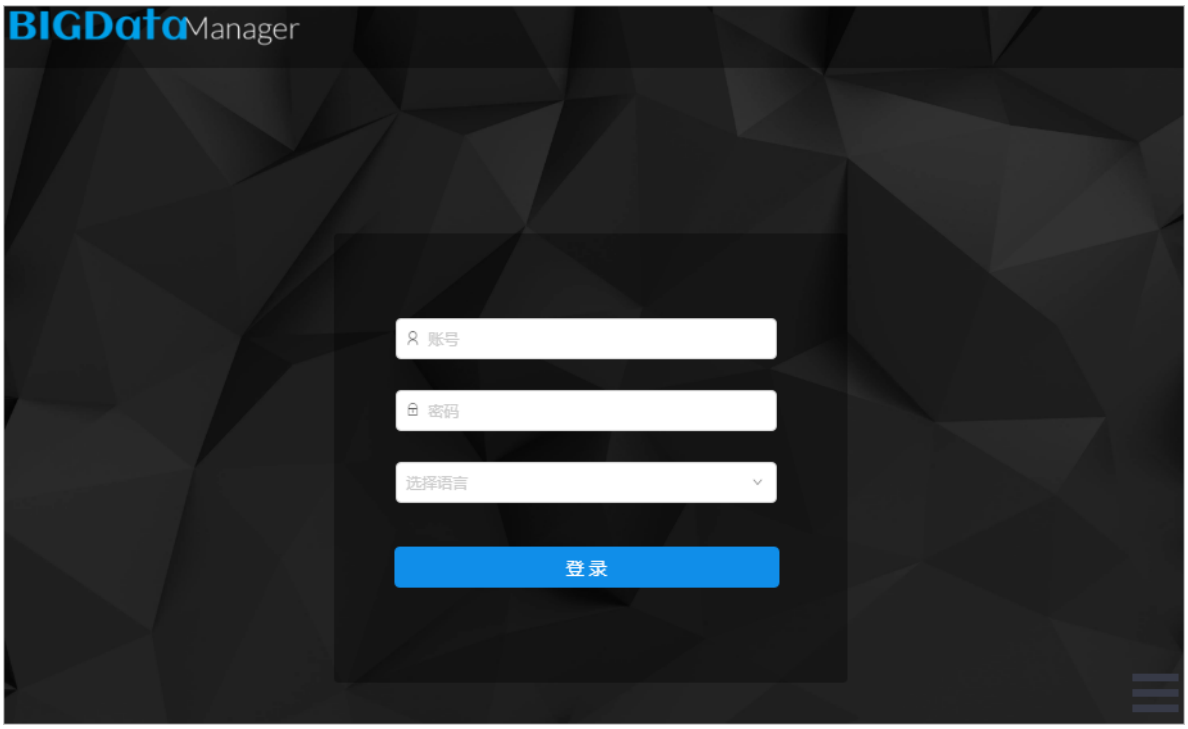
图 7-4: 详情

7.1.2.2 登录大数据管家

操作步骤

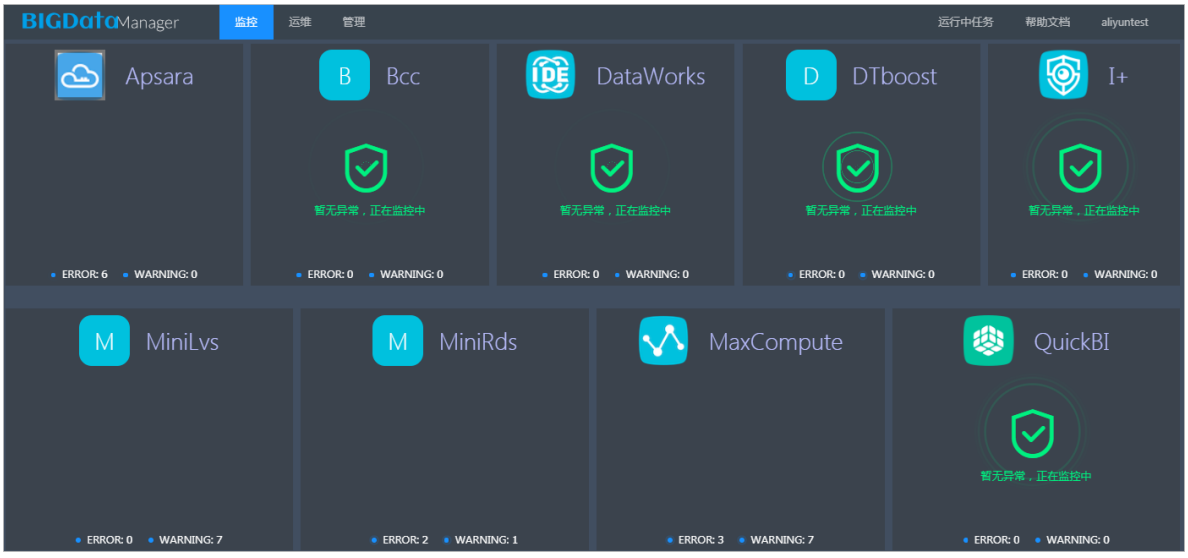
1. 根据查找大数据管家域名中查找的域名，登录大数据管家（初始账号和密码为aliyuntest/aaa111）。

图 7-5: 登录大数据管家



2. 输入账号、密码，并选择语言后，单击**登录**，即可进入大数据管家，如图 *7-6:* 大数据管家页面所示。

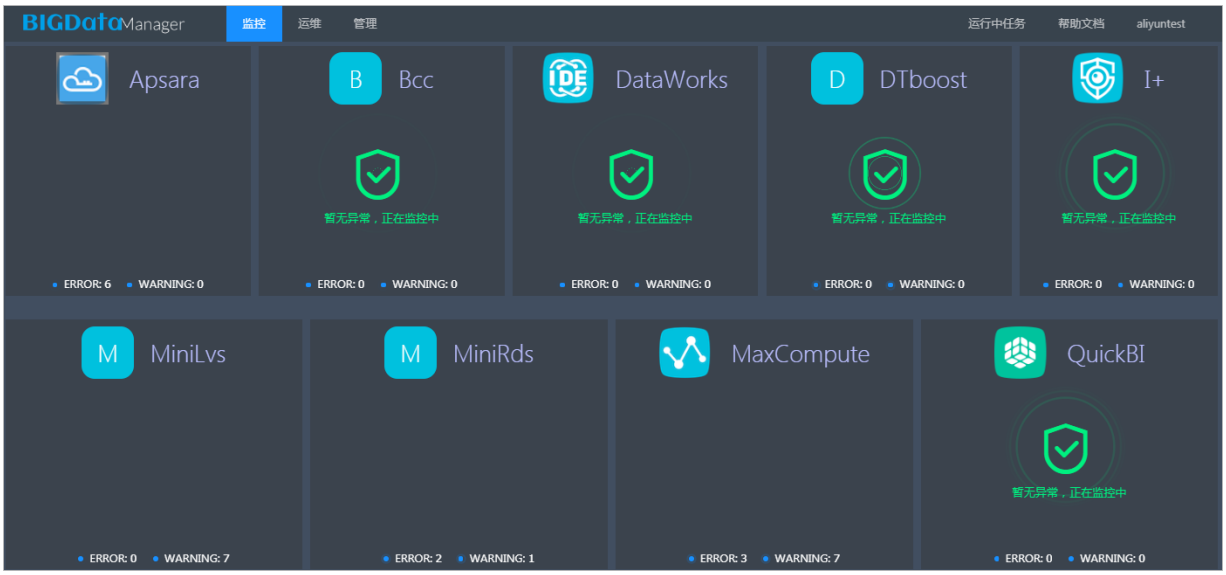
图 7-6: 大数据管家页面



7.1.3 大数据管家首页功能

登录完成后，即可看到大数据管家首页。它主要分为 监控、运维、管理和运行中任务。

图 7-7: 大数据管家首页



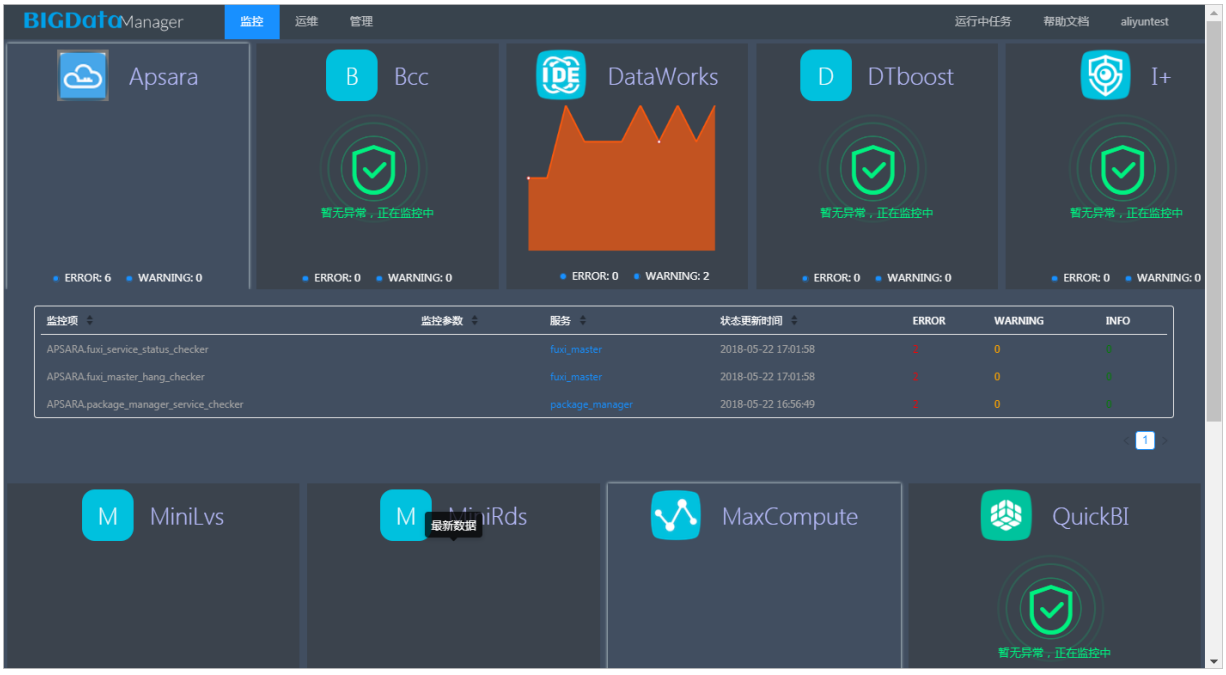
7.1.3.1 监控

监控功能用于展示大数据管家监控的所有产品的自检信息，包括产品当前的自检状态及自检告警和系统错误。同时，大数据数据管家监控还是各产品运维界面的入口。

状态概览

在**监控**页面中可以查看各产品的图 *7-8:* 自检结果。

图 7-8: 自检结果



产品自检结果说明：

- 有绿色图标说明产品自检均正常，例如：Bcc、DTboost等。
- 无图标说明产品有告警或系统错误，但一周内告警无新增或减少变化，例如：Apsara、MaxCompute等。
- 有橙色图标说明产品有告警并且一周内告警有新增或减少变化，橙色图是告警每分钟上报的告警次数的折线图，例如：DataWorks。

产品组件入口

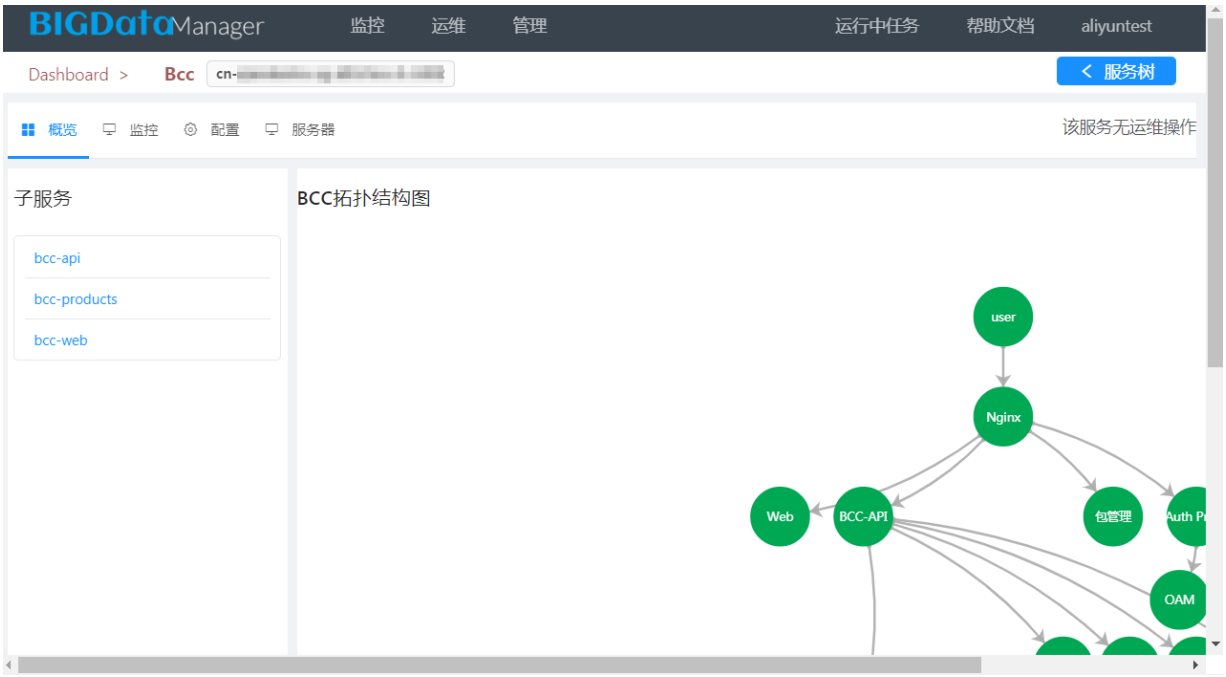
在**监控**界面，单击某个产品的产品名称进入该产品组件界面。

以大数据管家为例：单击**Bcc**，进入大数据管家组件界面，如图 *7-9: 产品组件入口*所示。

图 7-9: 产品组件入口

大数据管家组件界面示例如图 *7-10: 大数据管家组件界面-示例*所示。

图 7-10: 大数据管家组件界面-示例



查看告警和系统错误详情

1. 在**监控**页面中，选中存在告警或系统错误的产品，则在产品下方列出告警或系统错误涉及的监控项列表，如图 *7-11: 告警或系统错误涉及监控项列表*所示。

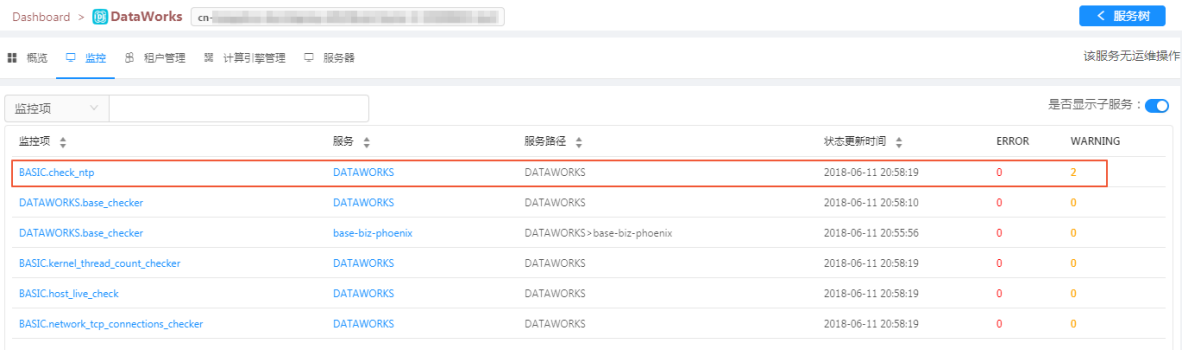
   图 7-11: 告警或系统错误涉及监控项列表

2. 在图 *7-11: 告警或系统错误涉及监控项列表*中，您可通过单击相应产品监控项的**服务**进入相应产品的**监控**页面。

   产品**监控**页面显示该产品的所有监控项及自检状态，如图 *7-12: 产品监控页面*所示。

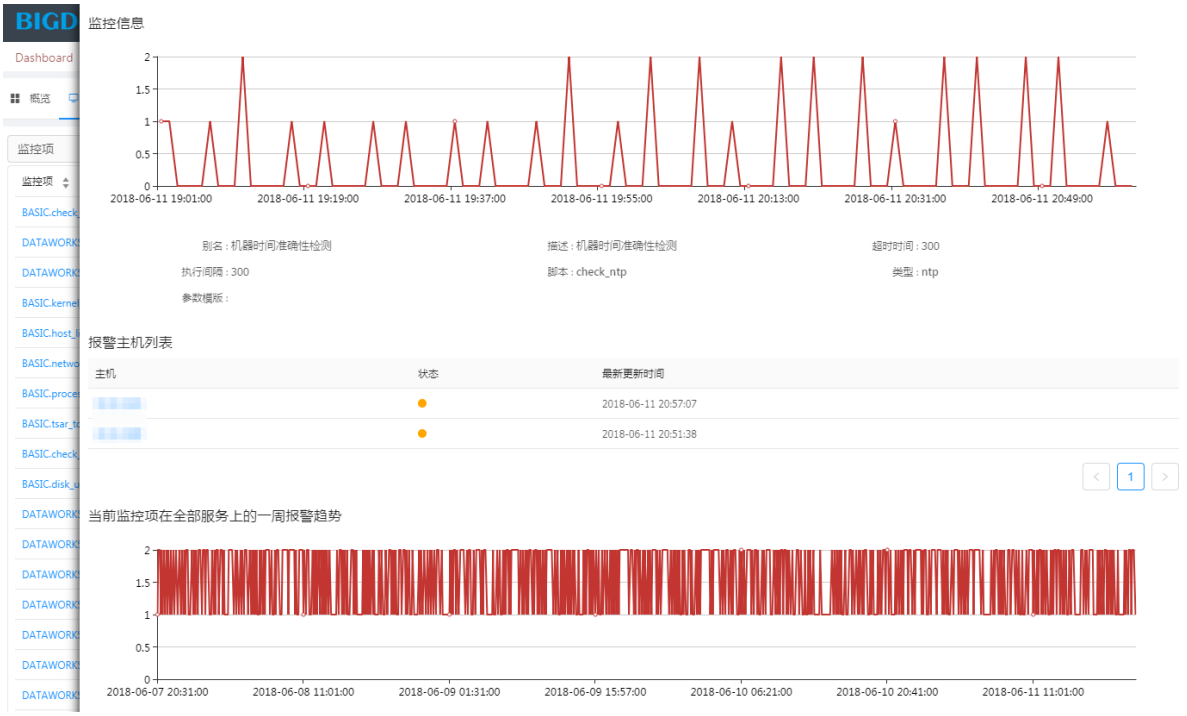
图 7-12: 产品监控页面



3. 对于有告警或系统错误的监控项，您可单击**监控项**名称进入**监控信息**页面

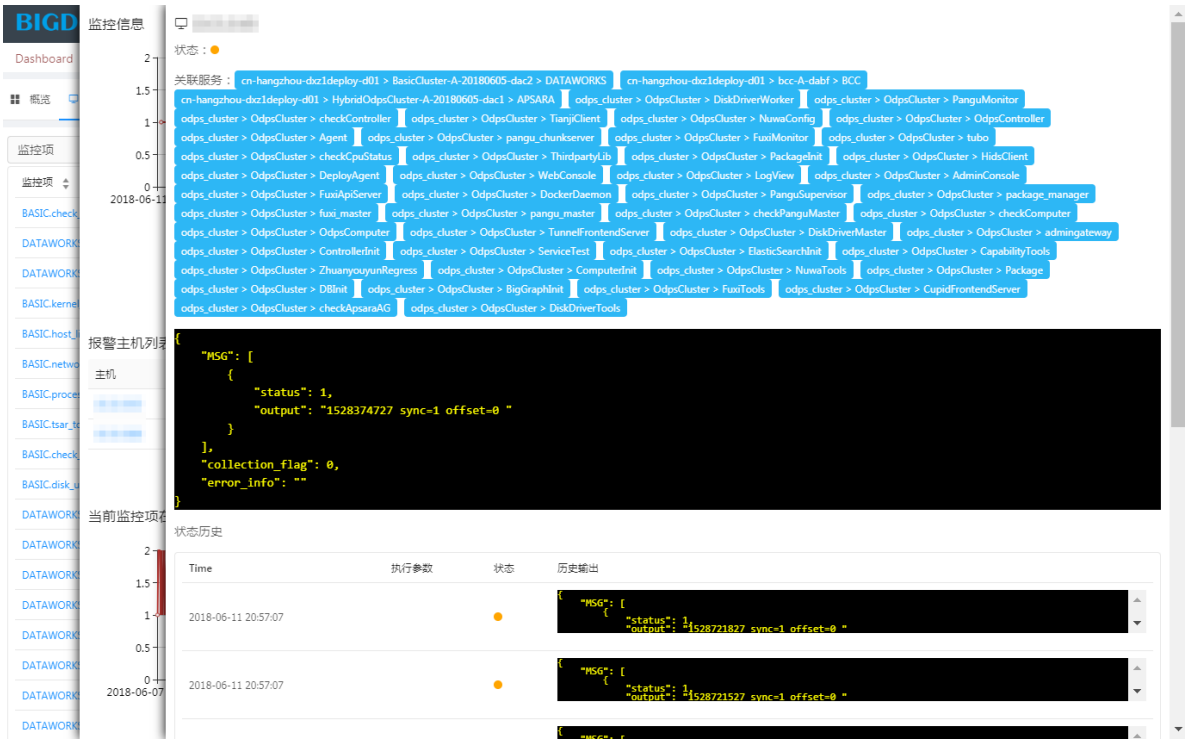
   在**监控信息**页面，您可查看监控项实例的历史趋势、监控项信息、主机列表以及该监控项在大数据管家平台上的历史趋势，如图 *7-13:* 监控信息所示。

图 7-13: 监控信息



4. 单击**主机**下的某个IP地址，进入**执行状态**界面，如图 *7-14:* 执行状态所示。

图 7-14: 执行状态



在**执行状态**界面，您可以查看以下信息：

- 该主机在该监控项实例上的当前输出内容。
- 关联服务：鼠标指向**关联服务**，则显示关联的服务在该监控项实例上的当前报错数量。
- 该主机在该监控项实例上的历史执行情况和输出。

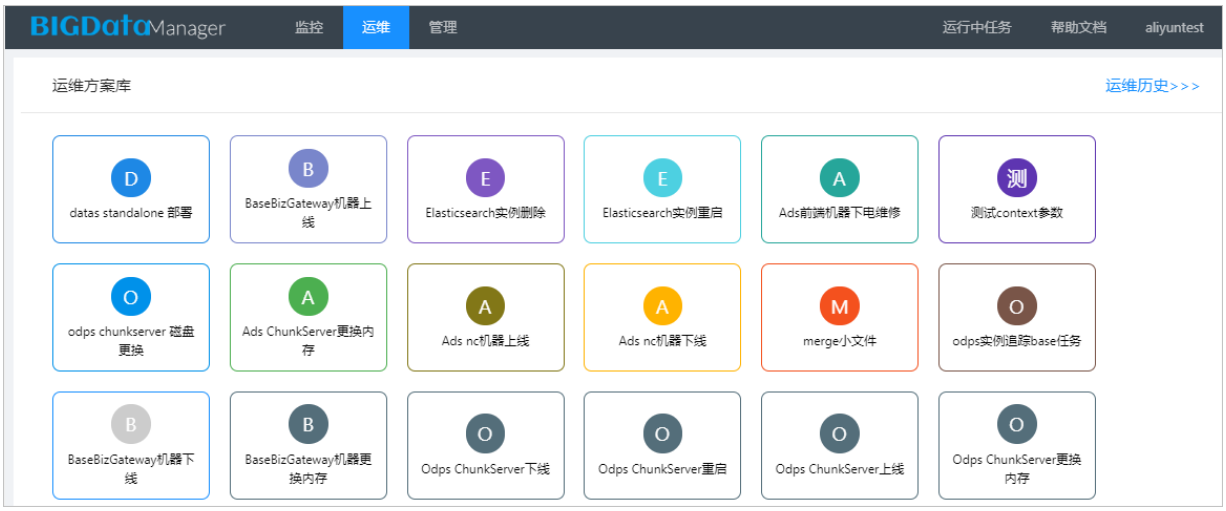
7.1.3.2 运维

运维是大数据管家提供的常用运维案例库。通过运维案例库模板，您只需要填写少量参数即可为您的实例环境创建一个产品运维任务。

运维案例库

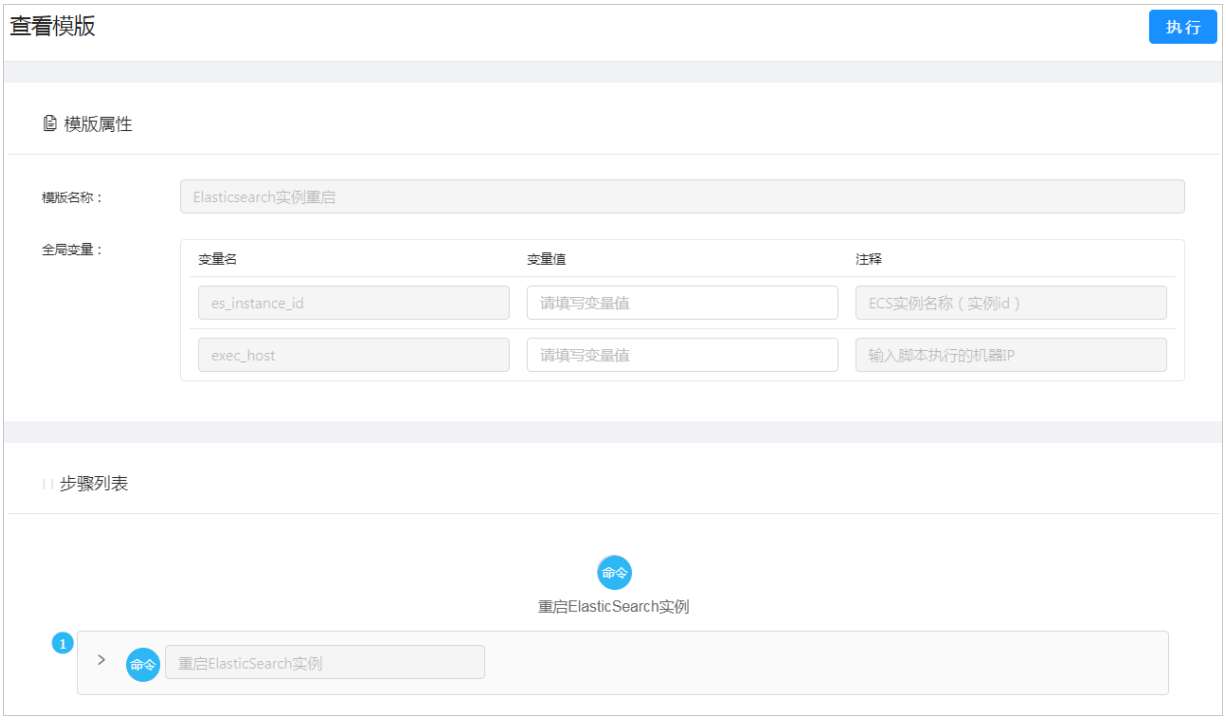
运维案例库界面如图 *7-15:* 运维案例库所示。

图 7-15: 运维案例库



每个运维案例均包括**模板属性**和**步骤列表**。**模板属性**和**步骤列表**中大部分参数均已固定，不可修改，用户只需要根据实际情况填写实例相关的参数即可执行。运维案例界面示例如图 *7-16:* 运维案例界面-示例所示。

图 7-16: 运维案例界面-示例



模板属性是实例环境的参数，您需要根据实际环境填写相关的设备和实例参数。

步骤列表的操作步骤包括脚本步骤和人工步骤。脚本步骤会自动执行，执行过程中自动映射**模板属性**中填写的参数。人工步骤需要用户根据步骤描述对实例进行相应操作。人工步骤正确执行完成后，系统会继续执行脚本步骤，直到执行完所有步骤。

说明：

步骤列表中的脚本步骤运行需要运行实例参数，这部分参数值均自动从**模板属性**中映射，不需要用户填写。

示例操作

下面以**odps chunkserver磁盘更换**为例进行介绍。

1. 单击**odps chunkserver磁盘更换**，进入**odps chunkserver磁盘更换**的查看模板界面，**模板属性**和**步骤列表**分别如图 *7-17: 模板属性*和图 *7-18: 步骤列表*所示。

图 7-17: 模板属性

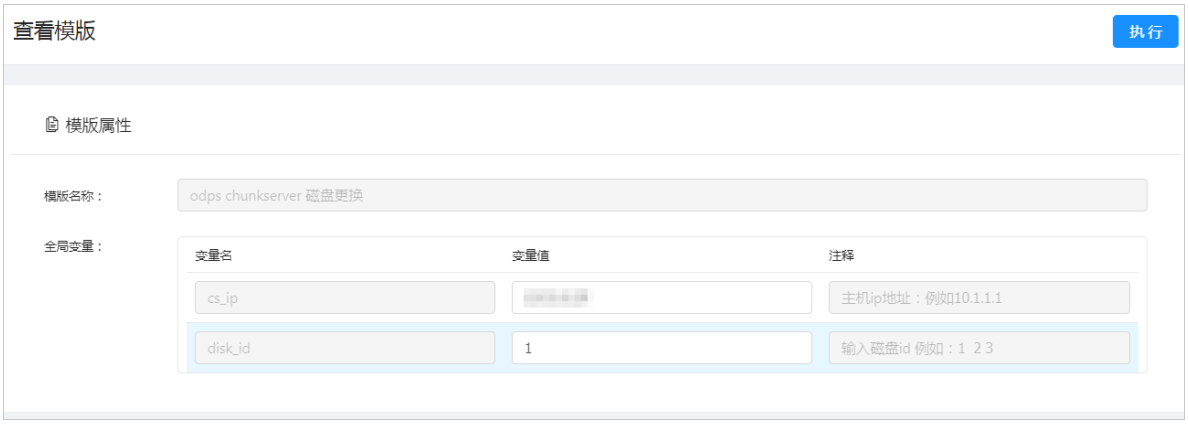
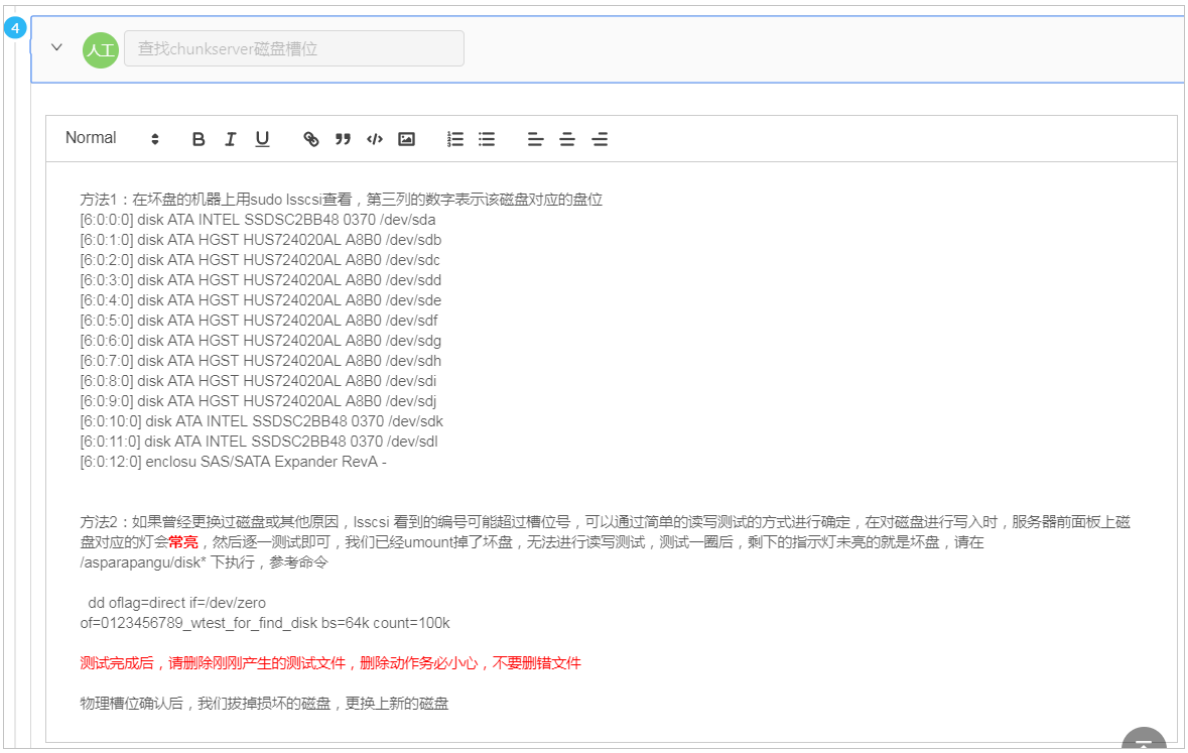


图 7-18: 步骤列表

2. 在**模板属性**中，根据**变量名**的**注释**提示，填写实际的运行实例的**变量值**。
3. 填写好各参数后，单击**执行**，系统自动执行脚本步骤。
4. 当执行到步骤4**人工**：**查找chunkserver磁盘槽位**时，说明需要人工手动执行。您可根据步骤4的提示进行手动操作，示例如下：

图 7-19: 人工操作描述



5. 人工操作完成后，系统会自动执行接下来的脚本步骤。

> **说明**：
> 后续有人工步骤时，均需要人工手动执行，直到执行完所有步骤。

运维历史

单击图 *7-15:* 运维案例库界面右上角的**运维历史**，您可查看产品运维的任务列表，详见运行中任务。

7.1.3.3 管理

大数据管家提供docker管理、云账号管理、oss管理、租户管理、计算引擎和热升级管理的功能，方便您对各产品进行运维管理。

7.1.3.3.1 docker管理

Docker是一个应用容器引擎，大数据管家为开发人员提供docker管理功能，方便开发人员快速的更新产品服务包。与在天基更新产品服务包相比，通过docker管理功能更新产品服务包时，您不需要更新整个docker的镜像，只需要替换docker中有更新的文件即可。

docker管理下左侧显示产品树，在该产品树下的子节点服务中，可以显示docker的机器IP地址、部署详情、运维详情、运维操作以及部署，如图 *7-20:* 管理功能所示。

图 7-20: 管理功能

右侧还有搜索、新增包、切换版本和删除功能，如图 *7-21:* 管理操作所示。

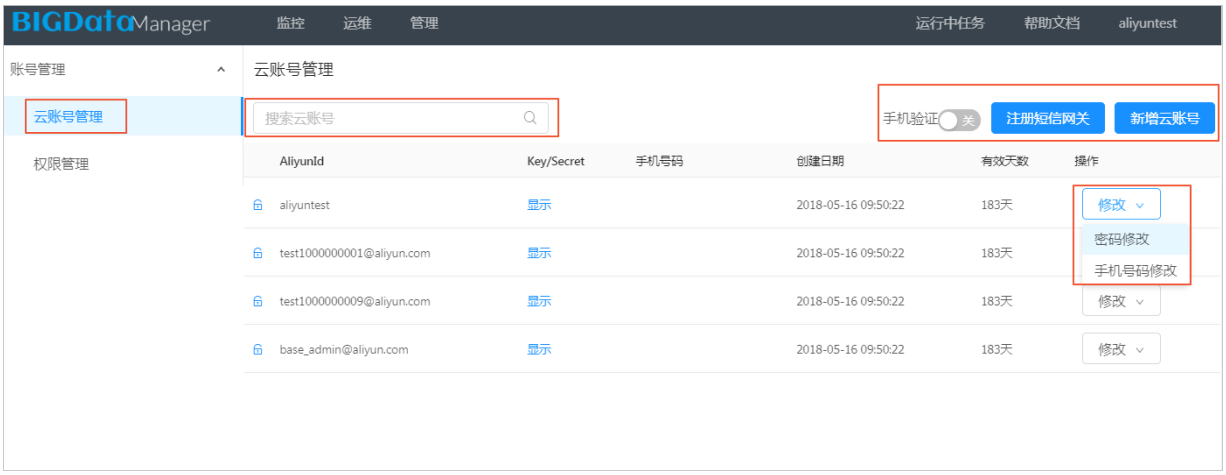
图 7-21: 管理操作

7.1.3.3.2 云账号管理

云账号是大数据各产品的运维、管理、配置账号。在使用各产品前，您首先需要在大数据管家开通一个云账号。

通过**新增云账号**按钮来添加账号，如图 *7-22:* 新增云账号所示；修改密码可以通过密码或者手机进行修改，并且在显示按钮中可以查看accessKeyId和accessKeySecret。

图 7-22: 新增云账号



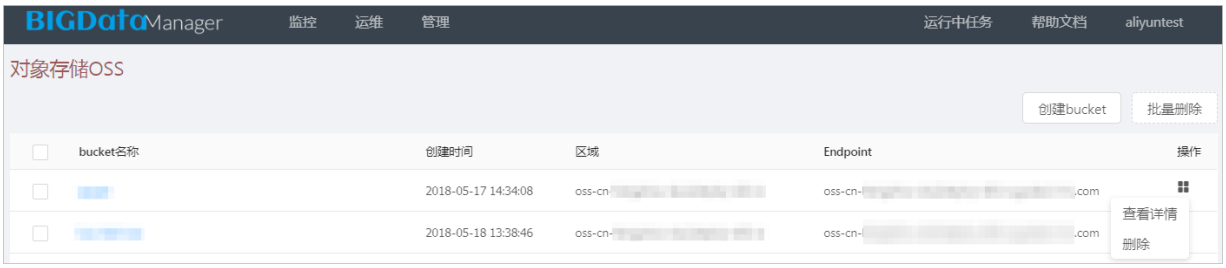
云账号具有账号管理和授权的功能，可以给某个角色授权、解除权限和查看详情，如图 *7-23:* 云账号管理所示。

图 7-23: 云账号管理

7.1.3.3.3 oss管理

大数据管家支持管理对象存储服务OSS（Object Storage Service），支持bucket创建、删除、查看。

图 7-24: oss管理



创建和删除bucket

1. 单击**创建bucket**，在弹出的**创建bucket**对话框中填写**bucket名称**。
2. 单击**OK**，新建bucket显示在bucket列表中。

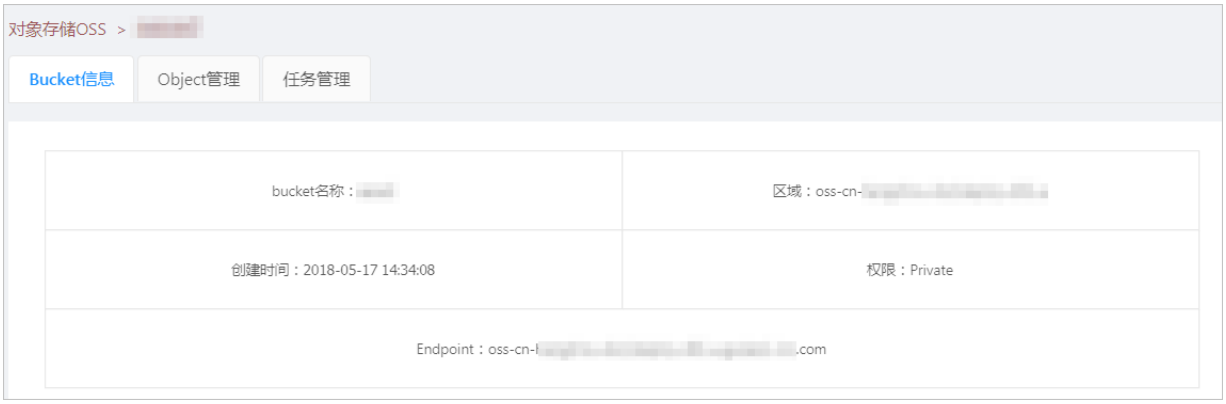
当bucket不再使用时，您可通过选中bucket，然后单击**批量删除**来删除bucket。

查看bucket信息

大数据管家支持查看已创建的bucket的信息，包括bucket名称、区域、创建时间、权限和Endpoint。

单击某个bucket的**bucket名称**，查看该bucket信息，如图 *7-25: Bucket信息*所示。

图 7-25: Bucket信息



Object管理

Object用于管理bucket内的数据文件，支持新建文件夹、上传文件、下载文件、删除文件或文件夹等操作，同时还支持批量删除操作。

您可通过单击**Object管理**进入**Object管理**界面，如图 *7-26: Object管理*所示。

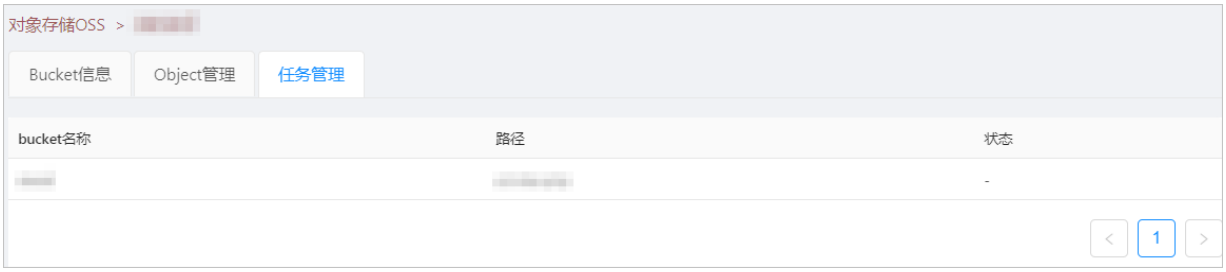
图 7-26: Object管理

您可通过图 *7-26: Object*管理中各功能按钮进行相关操作。

任务管理

任务管理用于查看文件上传任务的状态，如图 *7-27:* 任务管理所示。

图 7-27: 任务管理



7.1.3.3.4 租户管理

租户管理是DataWorks特有功能，请参见租户管理。

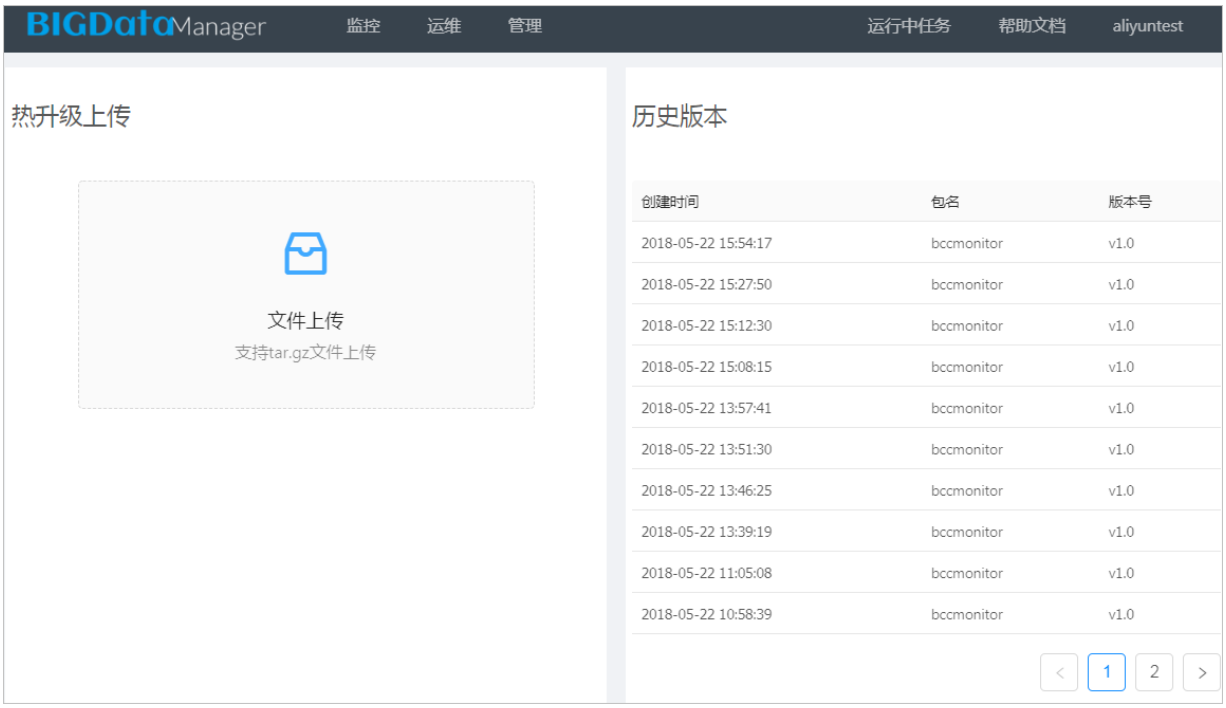
7.1.3.3.5 计算引擎

计算引擎是DataWorks特有功能，请参见计算引擎。

7.1.3.3.6 热升级管理

热升级管理用于对大数据管家的监控项进行在线热升级，不会中断业务。热升级管理界面如图 *7-28:* 热升级管理所示。

图 7-28: 热升级管理



7.1.3.4 运行中任务

大数据管家中的任务分为业务运维任务和产品运维任务。运行中任务是大数据管家系统中当前没有完成的工作流任务列表，包括执行中和执行失败的工作流。

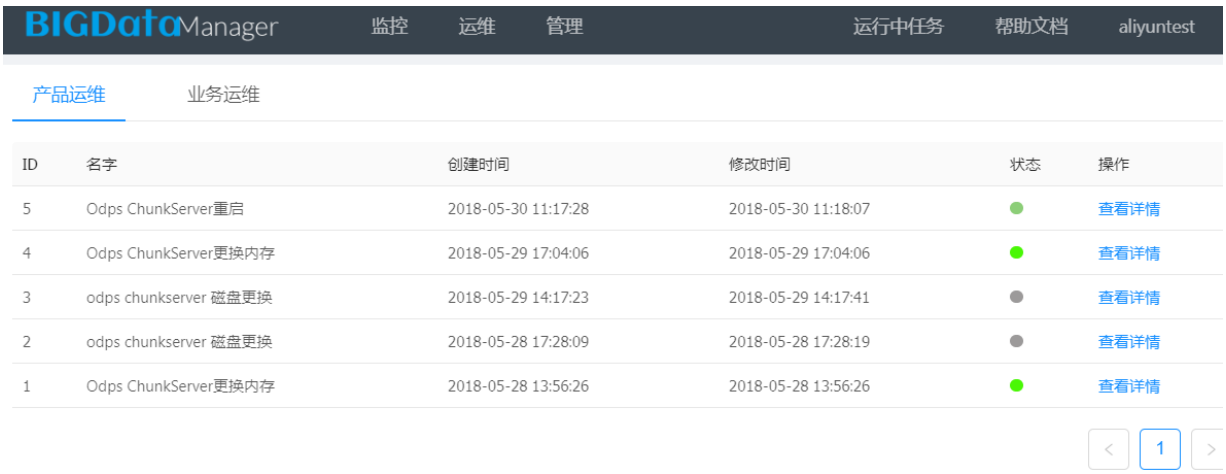
图 7-29: 运行中任务

单击**更多**您可以查看任务列表（包括所有任务）；单击某个运行中任务，您可以查看该任务的详情。

查看任务列表

单击**更多**，进入任务列表界面，如图 *7-30: 任务列表*所示。

图 7-30: 任务列表



任务列表中的任务包括产品运维和业务运维两种，产品运维任务是通过运维创建的产品相关的任务实例，业务运维任务是各具体产品的运维功能创建的业务相关的任务。

产品运维任务的状态包括未开始、等待、异常、失败、完成、取消，状态图标及说明如下：

- 未开始（ ● ）：表示任务未开始执行。
- 等待（ ● ）：表示任务执行到人工步骤，需要人工干预才可以继续执行后续步骤。
- 异常（ ● ）：表示任务执行过程中某步骤出现异常，无法继续执行后续步骤。
- 失败（ ● ）：表示任务执行失败。
- 完成（ ● ）：表示任务正常执行完成。
- 取消（ ● ）：表示任务被人为取消。

业务运维任务的状态包括完成、失败、取消、执行中，状态图标及说明如下：

- 失败（ ● ）：表示任务执行失败。
- 完成（ ● ）：表示任务正常执行完成。
- 取消（ ● ）：表示任务被人为取消。
- 执行中（ ● ）：表示任务正在执行中。

查看产品运维任务详情

单击**运行中任务**，选择某个产品运维任务；或单击**任务列表**中某个产品运维任务的**查看详情**，查看该产品运维任务的详情。

产品运维任务状态为未开始、等待、异常、失败时，您可根据实际情况进行相关操作：

- 未开始（ • ）：
  - 单击**开始**，开始执行该任务。
  - 单击**参数配置**，查看该任务的参数配置。
  - 单击**取消**，取消该任务。
- 等待（ • ）：
  - 根据提示进行相关操作，然后单击**确认**，继续执行后续步骤。
  - 单击**回滚**，回滚到上一步骤执行完的状态。
  - 单击**参数配置**，查看该任务的参数配置。
  - 单击**取消**，取消该任务。
- 异常（ • ）：
  - 单击**重试**，重新执行该步骤。
  - 单击**跳过**，跳过此步骤继续执行后续步骤。
  - 单击**回滚**，回滚到上一步骤执行完的状态。
  - 单击**参数配置**，查看该任务的参数配置。
  - 单击**取消**，取消该任务。
- 失败（ • ）：
  - 单击**重试**，重新执行该步骤。
  - 单击**跳过**，跳过此步骤继续执行后续步骤。
  - 单击**回滚**，回滚到上一步骤执行完的状态。
  - 单击**参数配置**，查看该任务的参数配置。
  - 单击**取消**，取消该任务。

以**BaseBizGateway机器更换内存**任务为例，任务失败的界面示例如图 *7-31:* 产品运维任务-失败示例所示。

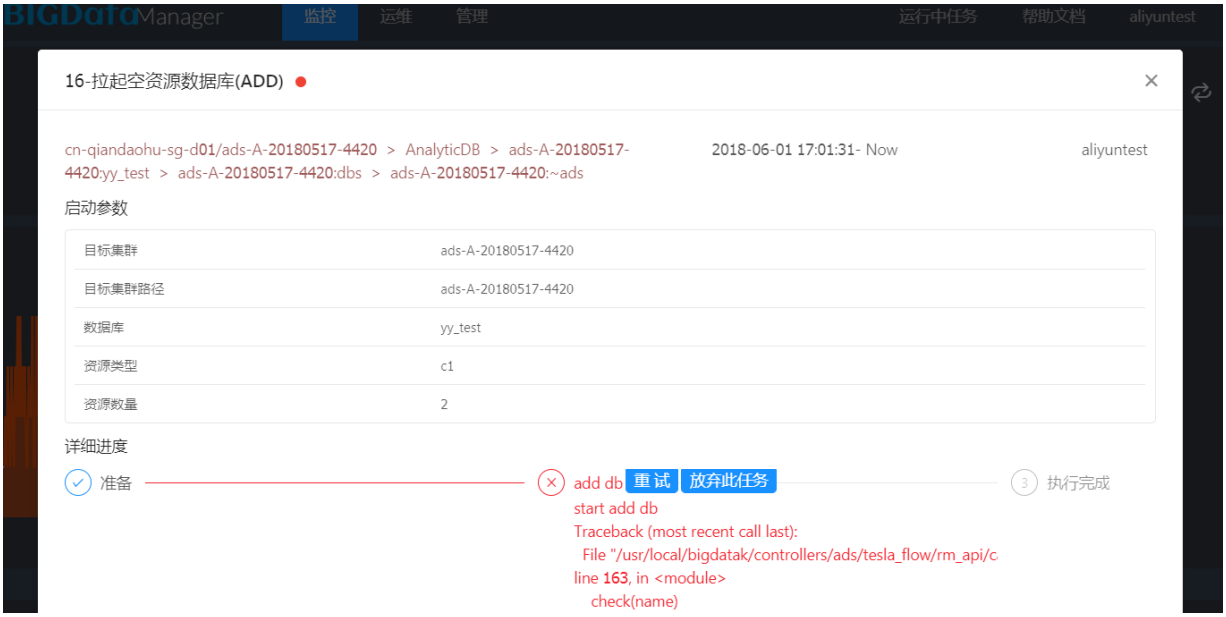
图 7-31: 产品运维任务-失败示例

查看业务运维任务详情

单击**运行中任务**，选择某个业务运维任务；或单击**任务列表**中某个业务运维任务的**查看详情**，查看该业务运维任务的详情。

业务运维任务状态为失败时，您可以进行单击**重试**来重新执行该任务，也可单击**放弃此任务**来取消该任务，如图 *7-32: 任务详情示例-执行失败*。

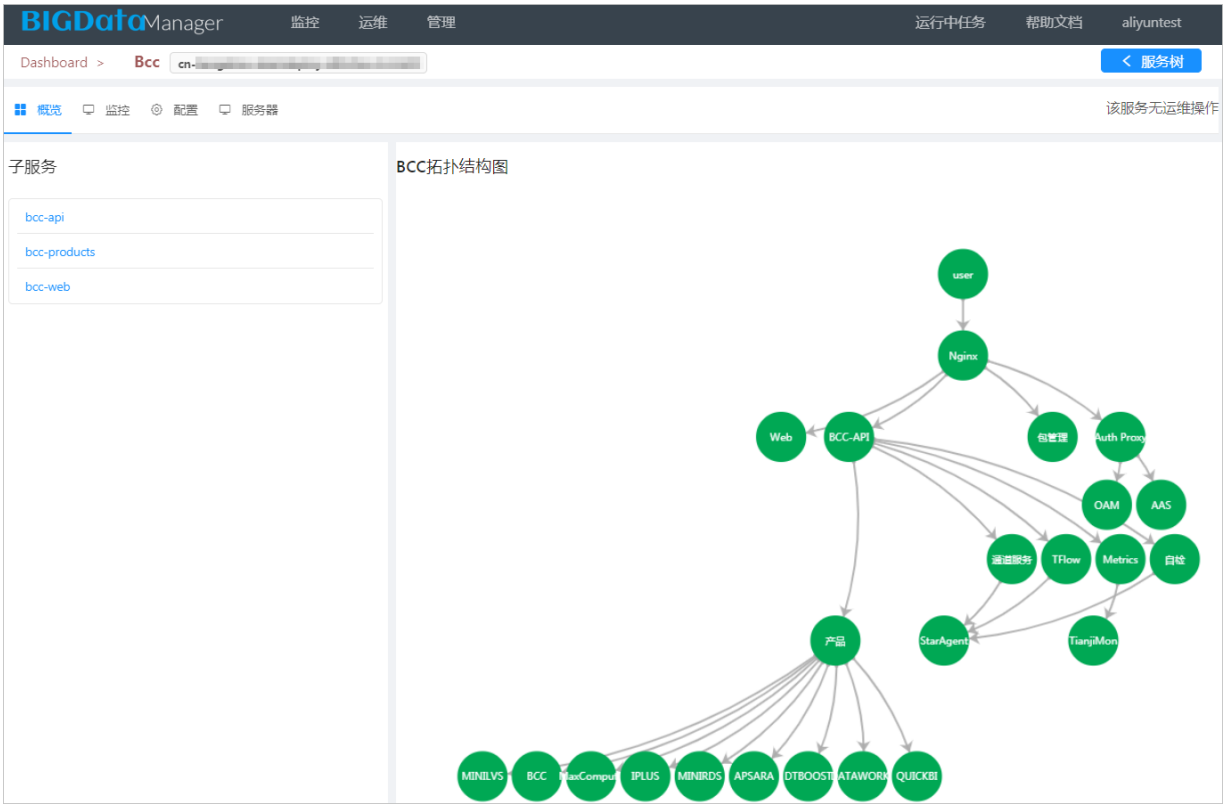
图 7-32: 任务详情示例-执行失败



7.1.4 大数据管家拓扑结构图

大数据管家产品组件界面显示系统各组件、运维产品的拓扑和状态图，如图大数据管家概览所示。

图 7-33: 大数据管家概览



在大数据管家概览的大数据管家拓扑结构图中，绿色表示服务正常，红色表示服务异常。通过大数据管家拓扑结构图，您可以轻松掌握当前系统各服务的运行状态。

7.2 大数据管家公共功能

7.2.1 产品树结构

产品服务树是产品的组织结构体现形式，每个服务组件（产品Product）都是一个独立存在的个体，都拥有任意多个服务（Service），所有服务的组织形成产品服务树。产品树分为两种，一种是静态产品树，一种是动态产品树。

产品树简介

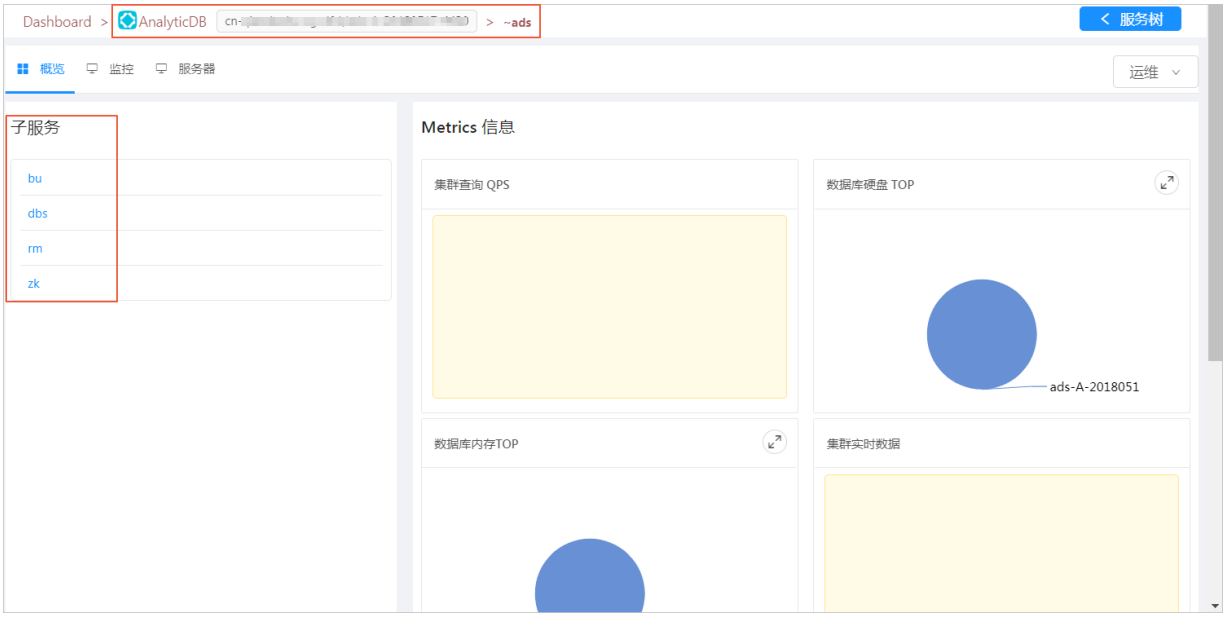
以AnalyticDB为例，如图*AnalyticDB*概览所示，左上角为当前页面所在的服务位置，路径包括产品（AnalyticDB）、REGION（cn-hz-dxz008-d01）、集群（ads-A-20171025-9cff）、ServiceId。

左侧子服务列表中，包含了当前服务下其子服务，此次表示AnalyticDB产品下包含APSARA、admingateway、~ads、~gallardo四个子服务，此处显示为服务名称。

图 7-34: AnalyticDB概览

选择~ads，查看~ads服务对应的服务路径和子服务信息，如图*service信息*，服务路径为AnalyticDB::~ads（ads-service），而子服务为bu、dbs、rm和zk，以此类推，其他的产品、服务也有类似的层级结构。这样一级一级的组织就形成了该产品特有的树状结构，称其为产品树结构。

图 7-35: service信息



静态产品树

静态产品树顾名思义就是配置好之后就不会再改变了，比如DataWorks这类服务比较固定的产品，只要在开发过程中按照一定的规则组织好就行了，部署完之后页面的展示根据配置值渲染出来。在后面的配置章节中看到的，如图产品树配置1、产品树配置2所示。

图 7-36: 产品树配置1

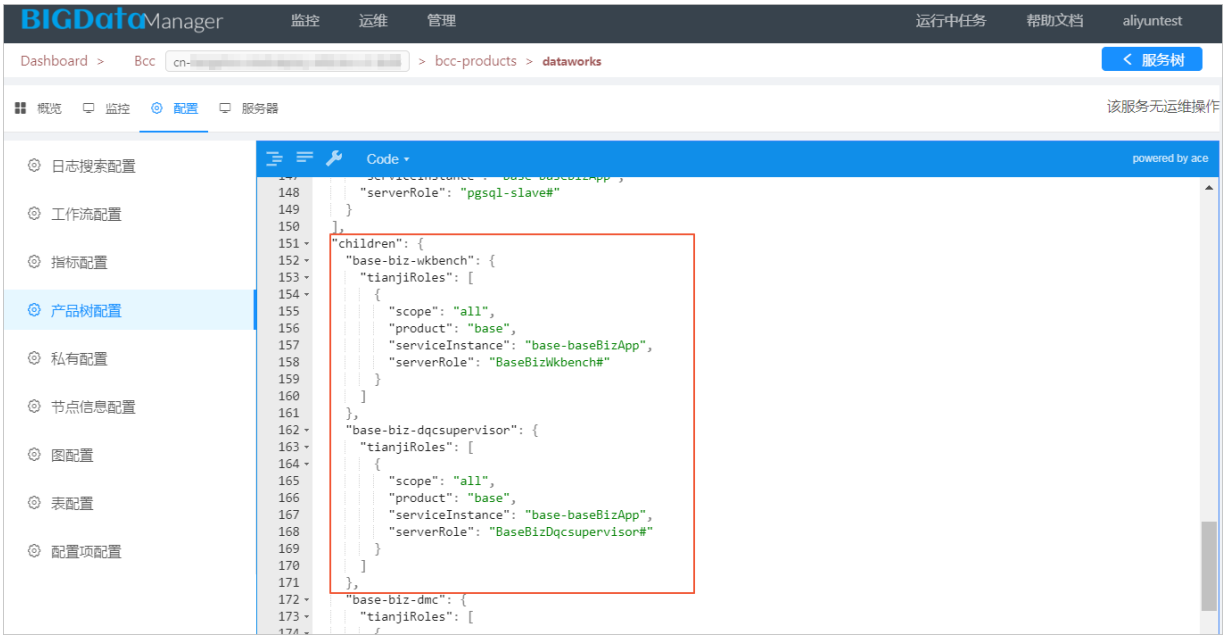
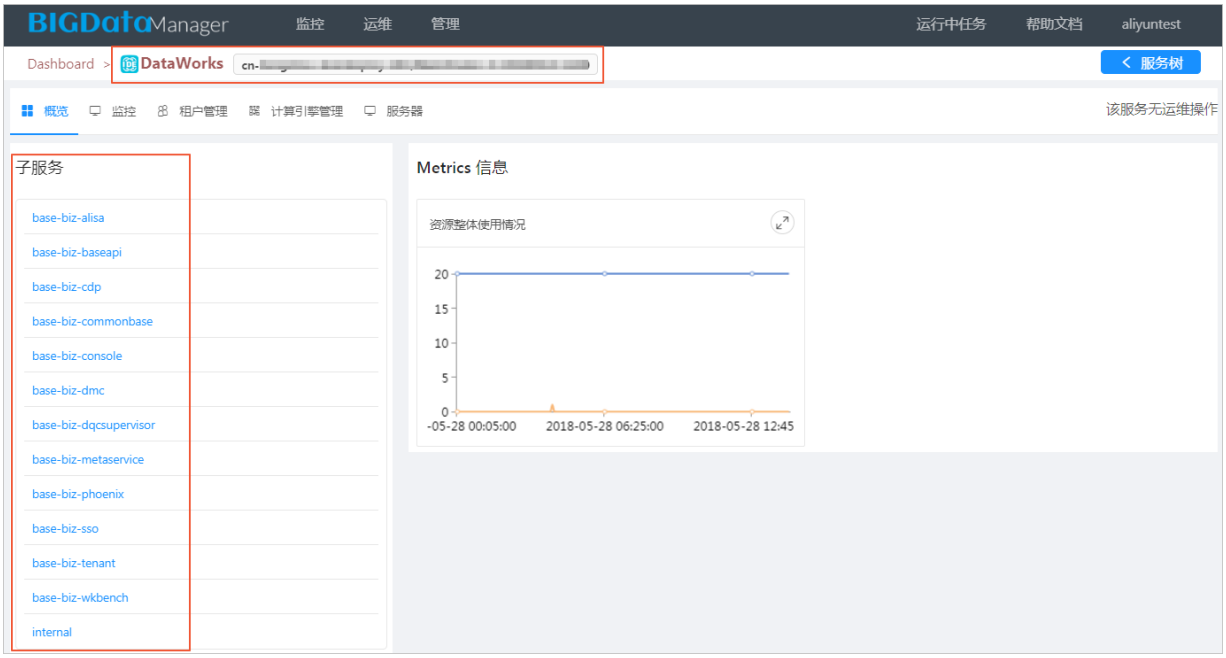


图 7-37: 产品树配置2



动态产品树

动态产品树则为动态生成的，某个服务下的子服务是不固定的，会根据当前系统的情况组织成不一样的产品树，体现当前产品组织结构。比如AnalyticDB，图产品树配置3、产品树配置4中配置侧产品树是一定的，只是子服务为服务类型**db**，那么类型是DB的服务都会在挂在ads::dbs服务下。

图 7-38: 产品树配置3

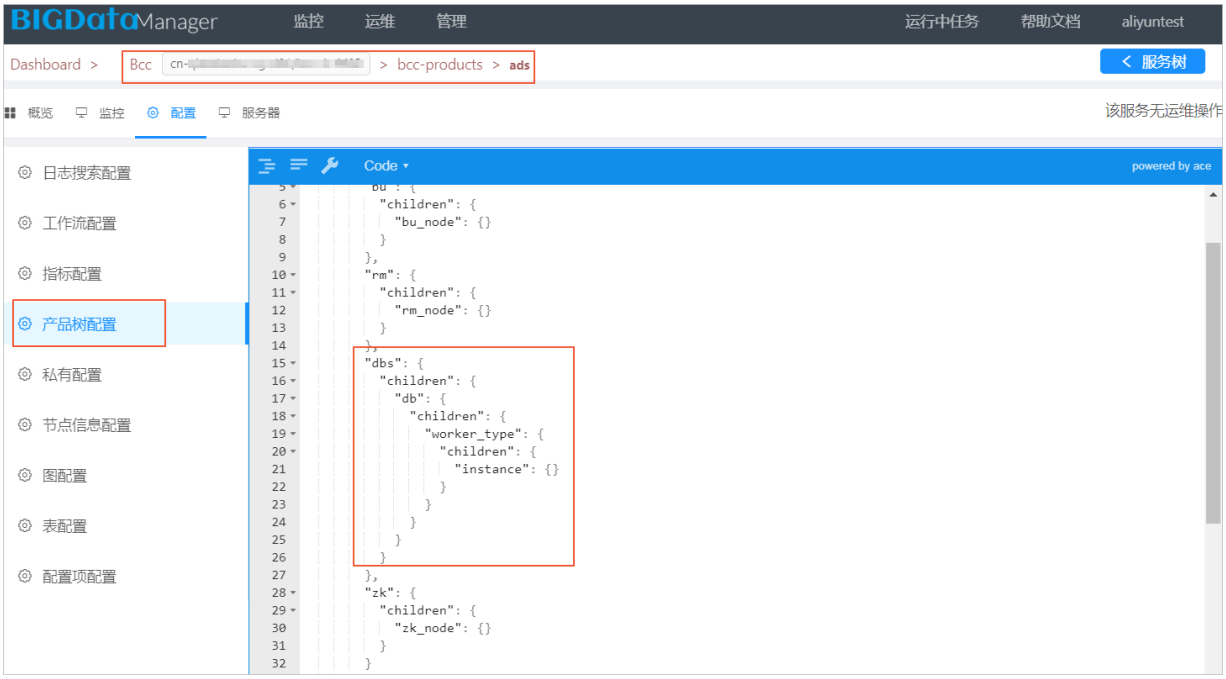
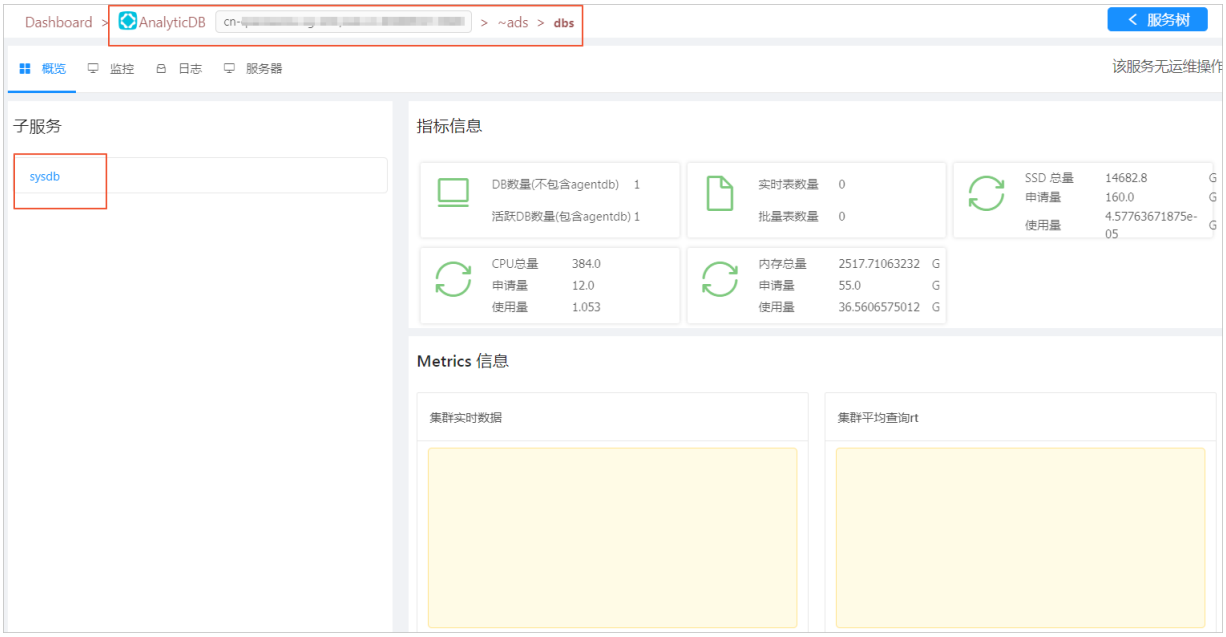


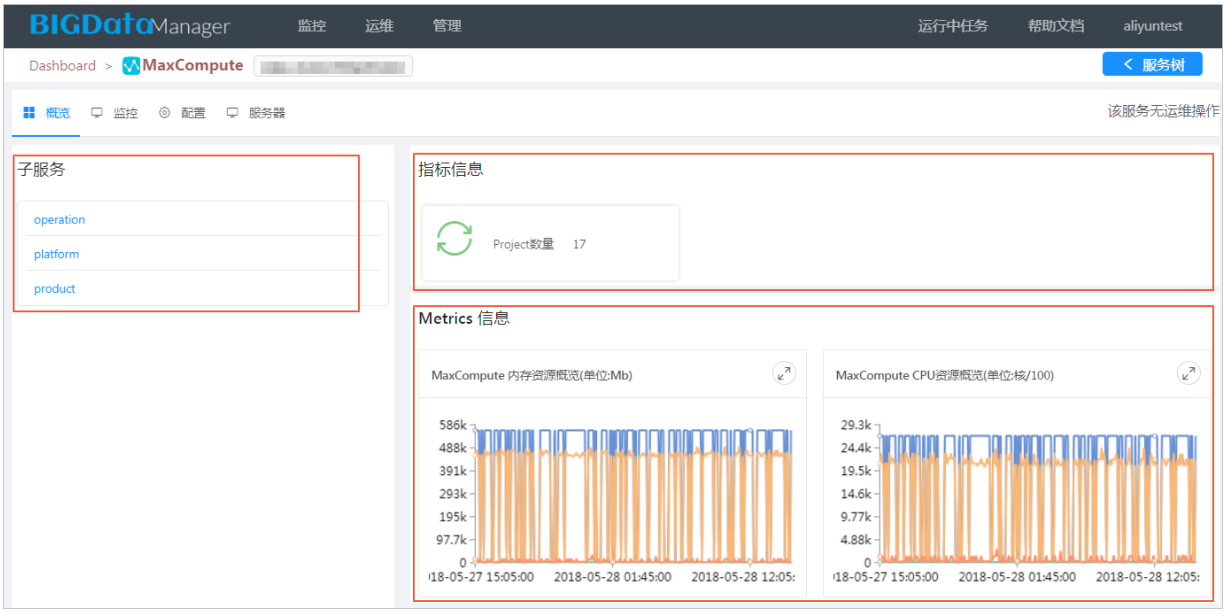
图 7-39: 产品树配置4



7.2.2 概览

概览页面用于展示大数据库管家所监控的产品的概要信息，左侧子服务列表用于展示产品的子服务，右侧用于展示产品的指示信息和Metrics信息，如图 *7-40: 概览页面*所示。

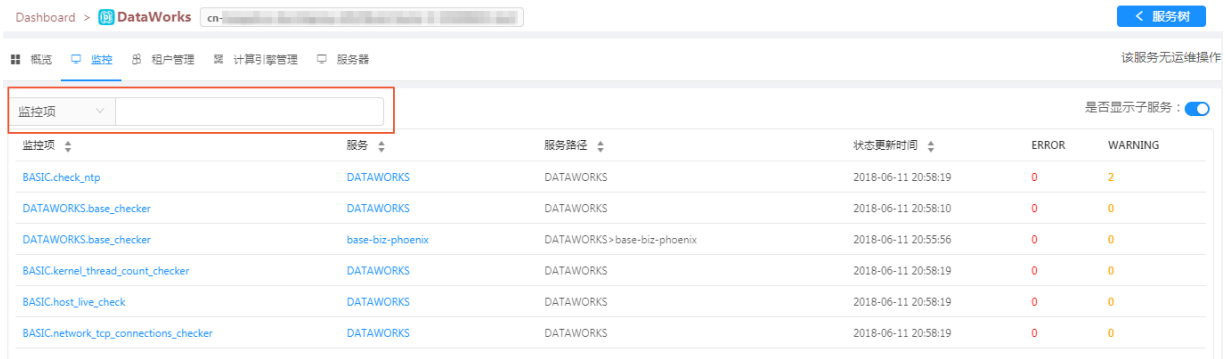
图 7-40: 概览页面



7.2.3 监控

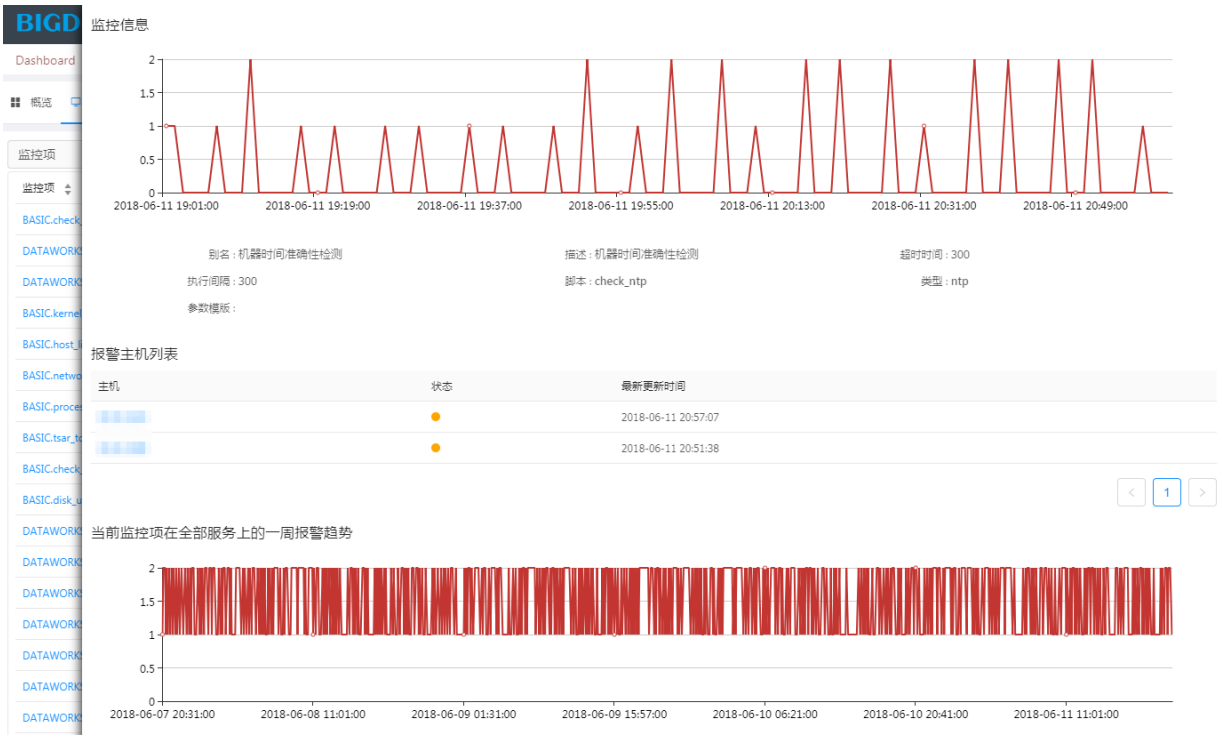
监控页面是大数据库管家所监控的产品均有的页面，用于展示所监控产品的监控项及监控状态，如图 *7-41: 监控页面*所示。

图 7-41: 监控页面



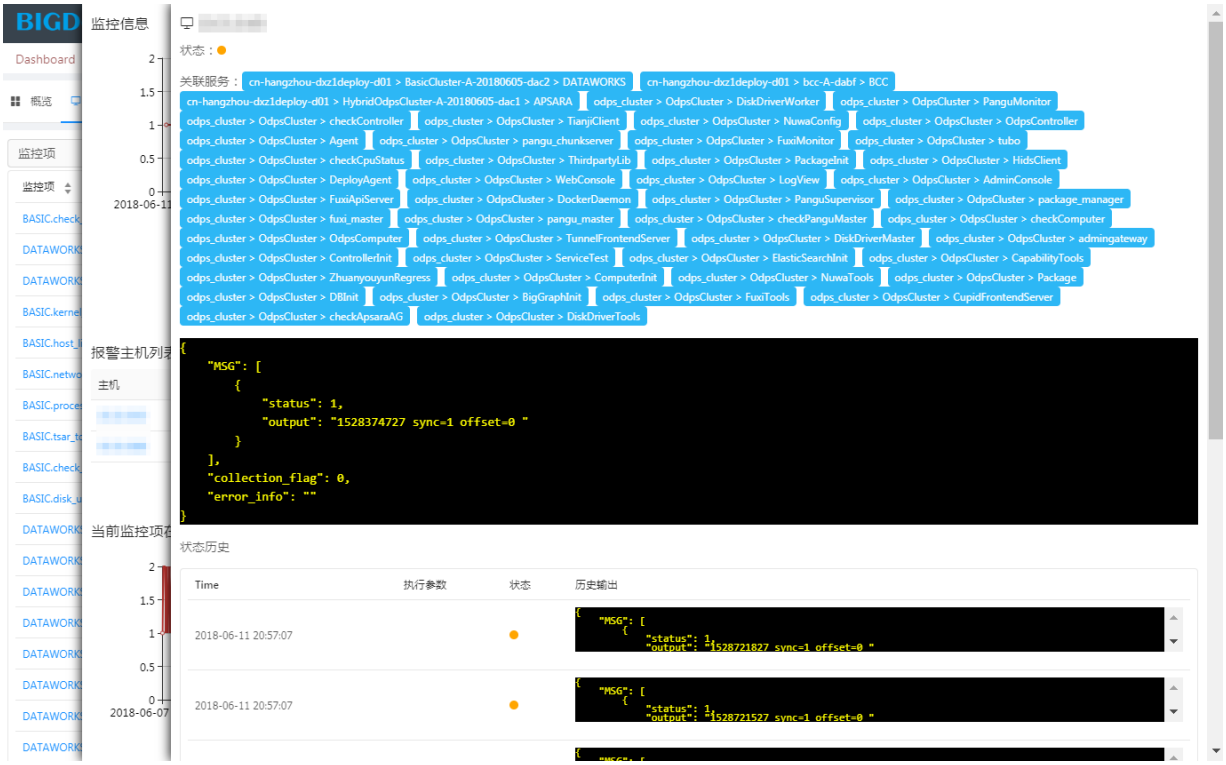
在**监控**页面，您可通过**监控项**、**服务**、**运行实例**等进行过滤搜索。您也可通过单击某个**监控项**，查看该监控项实例的历史趋势、监控项信息、主机列表以及该监控项在大数据管家平台上的历史趋势，如图 *7-42: 监控信息*所示。

图 7-42: 监控信息



在**监控信息**界面，通过单击**主机**下的IP地址，进入**执行状态**界面，如图 *7-43: 执行状态*所示。

图 7-43: 执行状态



在**执行状态**界面，您可以查看以下信息：

- 该主机在该监控项实例上的当前输出内容。
- 关联服务：鼠标指向**关联服务**，则显示关联的服务在该监控项实例上的当前报错数量。
- 该主机在该监控项实例上的历史执行情况和输出。

7.2.4 配置

产品服务配置是根据服务的实际需求针对已部署的产品服务进行的相关配置、操作，从而实现您实际的运维需求，包括工作流配置、服务配置，自检配置、指标配置、表配置、图配置、日志搜索配置、节点信息配置、采集插件配置、健康配置、基础配置、私有配置、配置项配置。

现在的配置有两种体现，一种是配置类配置，基本上都在大数据管家产品的子服务（bcc-products）下，另一种是服务应用类配置，一般在每个产品的服务配置菜单下。

7.2.4.1 配置类配置

配置类配置展示了大数据管家下所有产品静态配置文件，以便用户快速查看服务下面的信息。

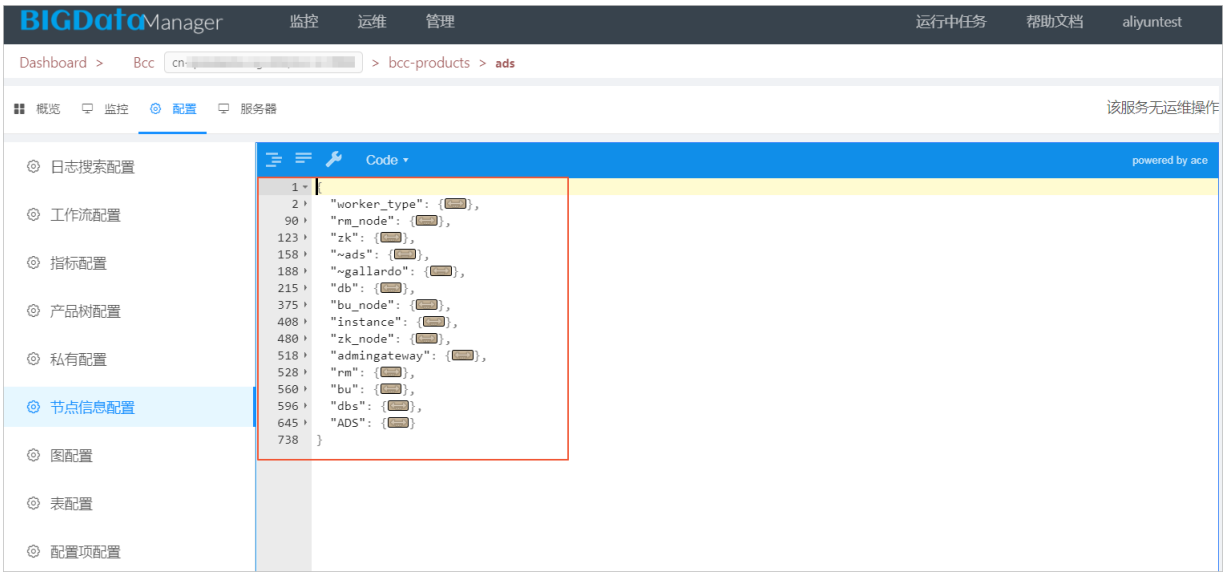
各配置项配置的数据都是key-value形式，key为配置的ID，value为配置项具体内容。配置项的配置ID最后整合到节点信息配置中，页面的布局、服务菜单选择都在节点信息配置中定义。

下面以AnalyticDB的节点配置、工作流配置、指标配置、图配置为例进行介绍。

节点信息配置

节点信息配置主要是把其他配置项整合起来然后在前端进行布局的配置信息，key为服务类型或者服务名称，如图 *7-44:* 节点信息配置所示。

图 7-44: 节点信息配置



在节点信息配置中，AnalyticDB概览对应图 *7-45:* 首页说明中的**workflow**和**summary**，具体说明如下：

- workflow：即工作流（如概览中的4）。
- summary：即概览，包含child-service-info即子服务（如概览中的1）、quota即指标信息（如概览中的2）、metrix即图信息（如概览中的3）。

图 7-45: 首页说明

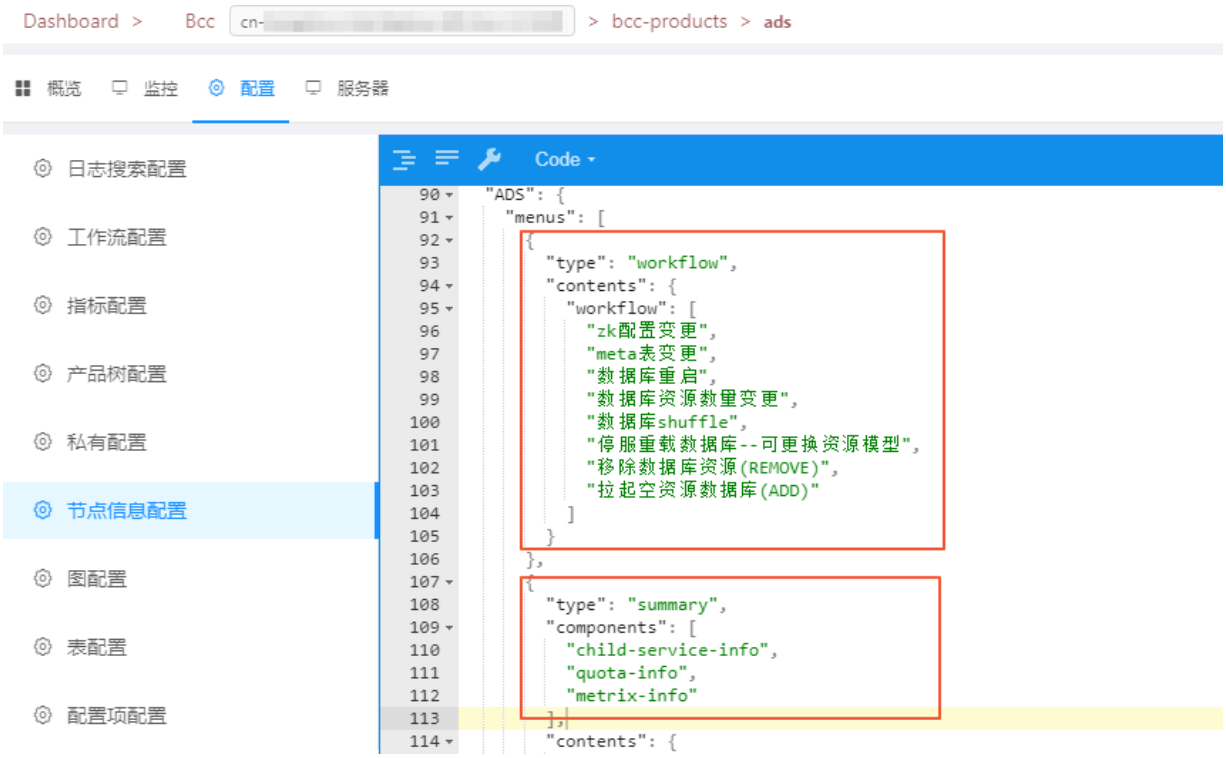
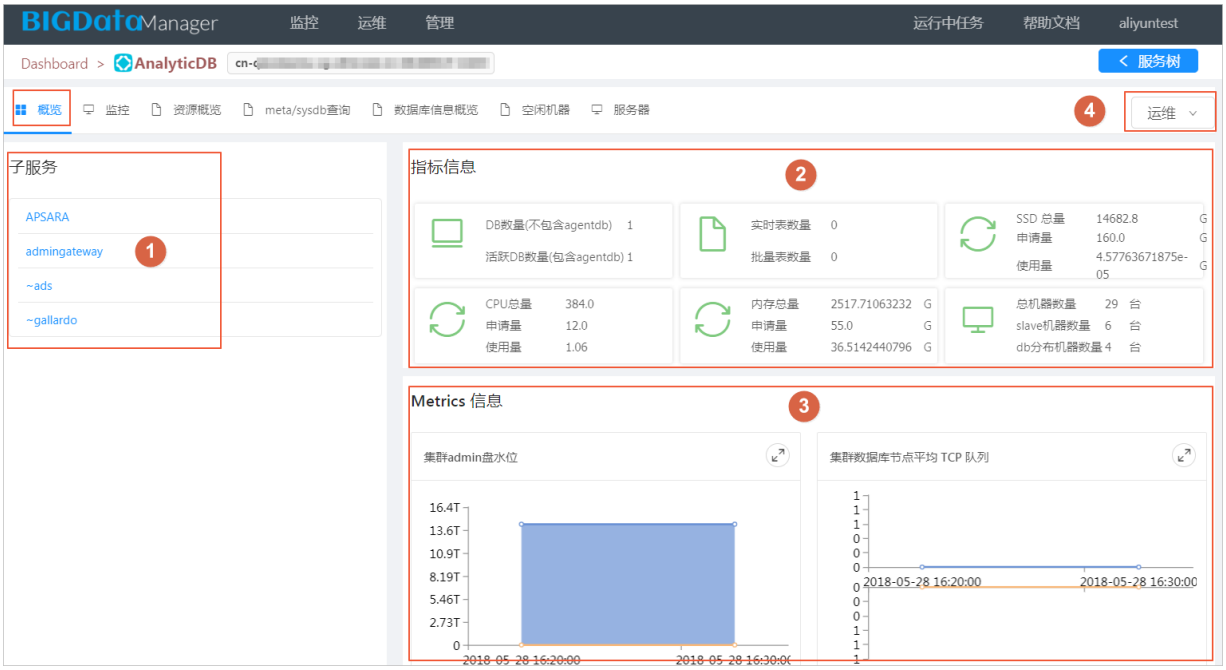


图 7-46: 概览



工作流配置

工作流配置、节点信息配置和前端菜单信息是一致的，具体如下：

- 工作流配置如工作流配置所示。
- 节点信息配置如节点信息配置所示。
- 前端菜单信息如图 *7-49: 前端菜单信息*所示。

图 7-47: 工作流配置

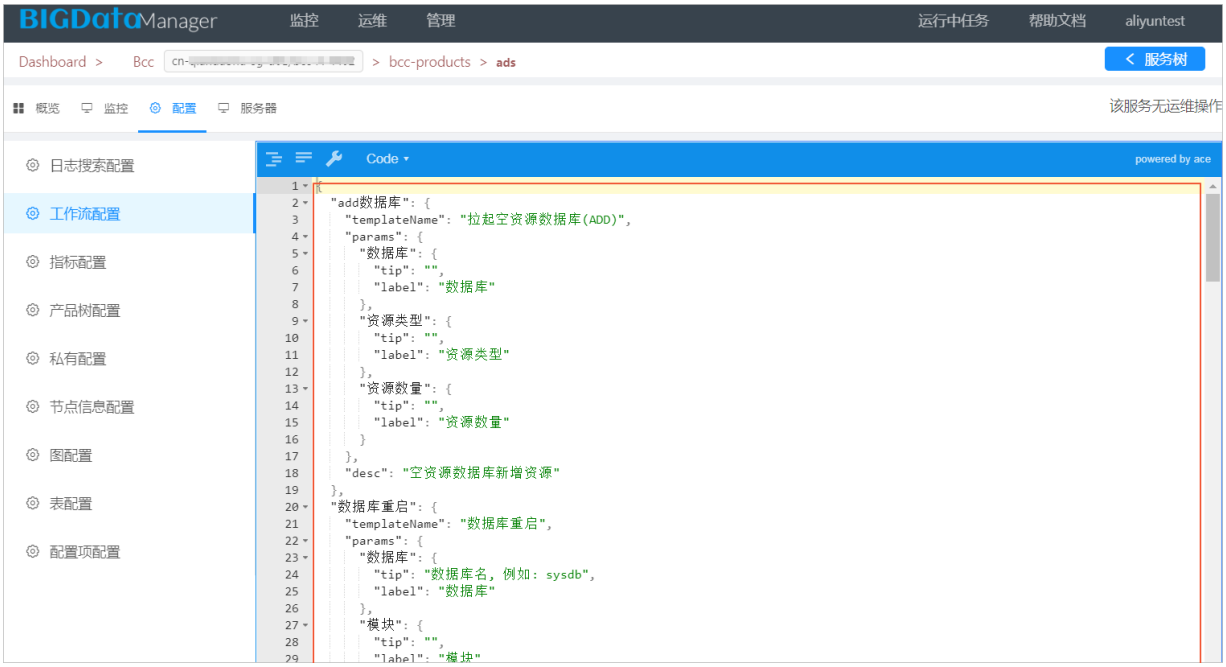


图 7-48: 节点信息配置-workflow

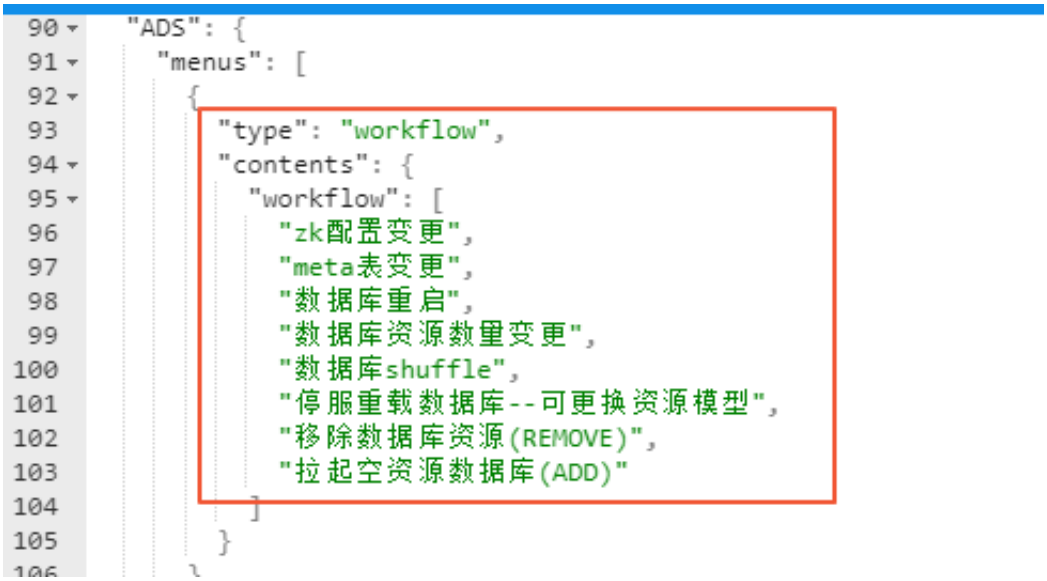
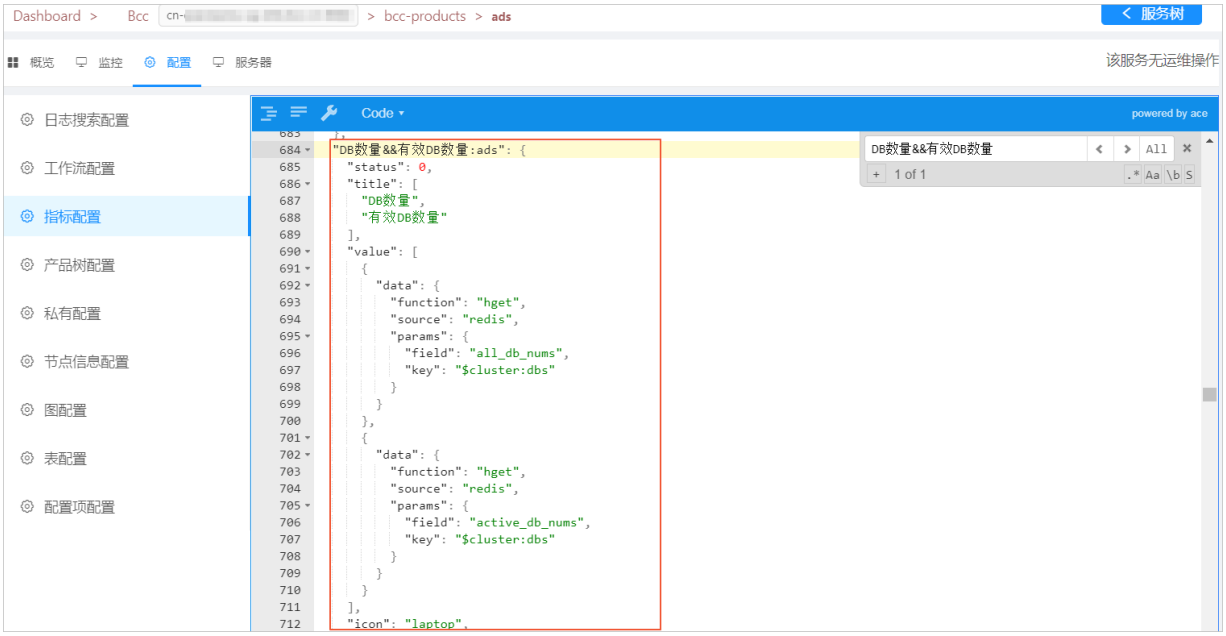


图 7-49: 前端菜单信息

指标配置

在节点信息配置中，配置的是各类型配置的keyID。例如：指标信息（quota-info）中配置了**DB数量&&有效DB数量:ads**，则其显示信息是从指标配置中key为**DB数量&&有效DB数量:ads**的指标信息来获取的，如图指标配置。

图 7-50: 指标配置

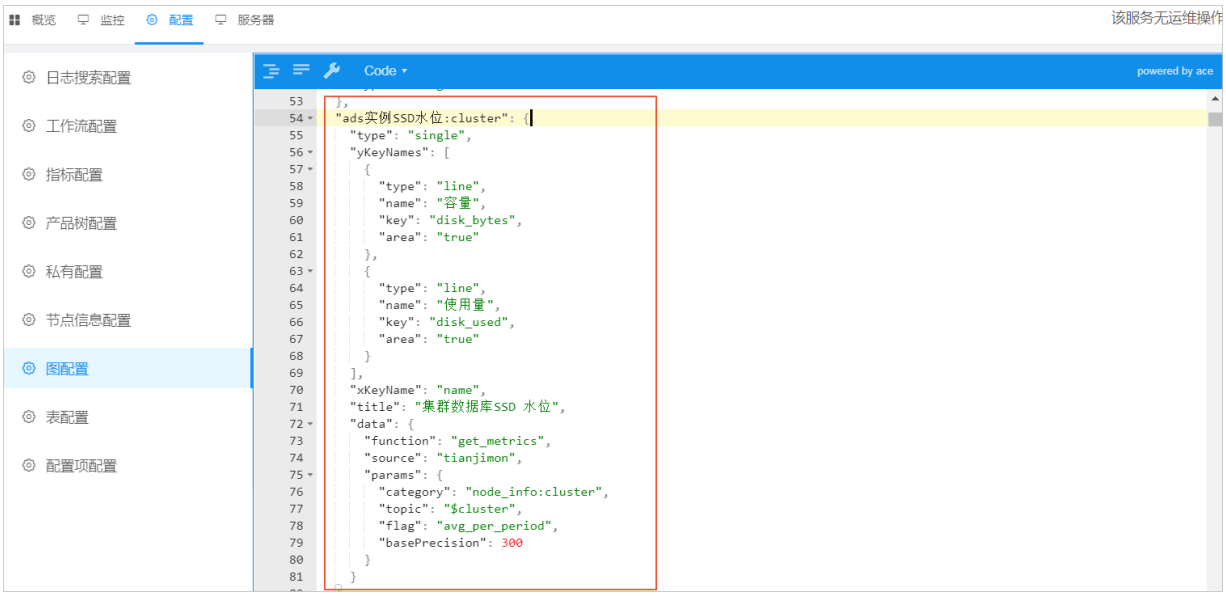


页面的指标信息也是根据该配置获取的，显示的数据个数、数字代表内容等均有描述。

图配置

在节点信息配置中，配置的是各类型配置的keyID。例如：节点信息配置中配置的**SSD 水位:cluster**，在图配置中都有详细的配置说明，如图配置所示。

图 7-51: 图配置

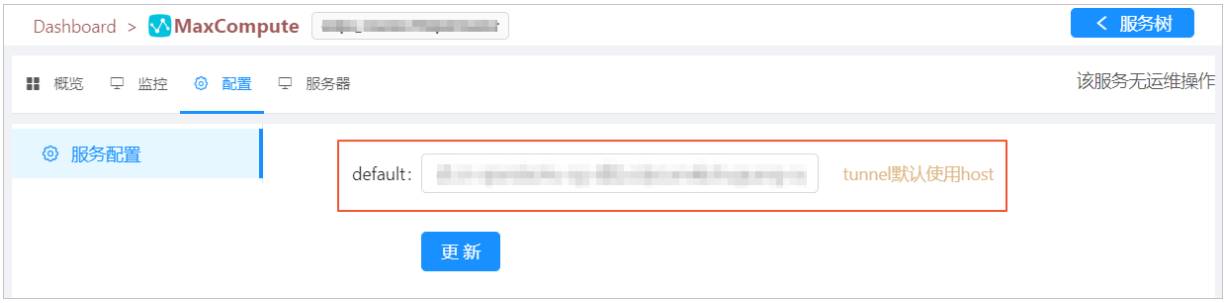


您在使用时可以通过节点信息配置和其他配置文件配置确认服务页面菜单和数据显示。

7.2.4.2 服务应用类配置

服务应用类配置是在产品服务下的配置，此类配置主要是应用级别的配置，只有部分产品或产品服务支持。例如，Maxcompute的服务配置如下：

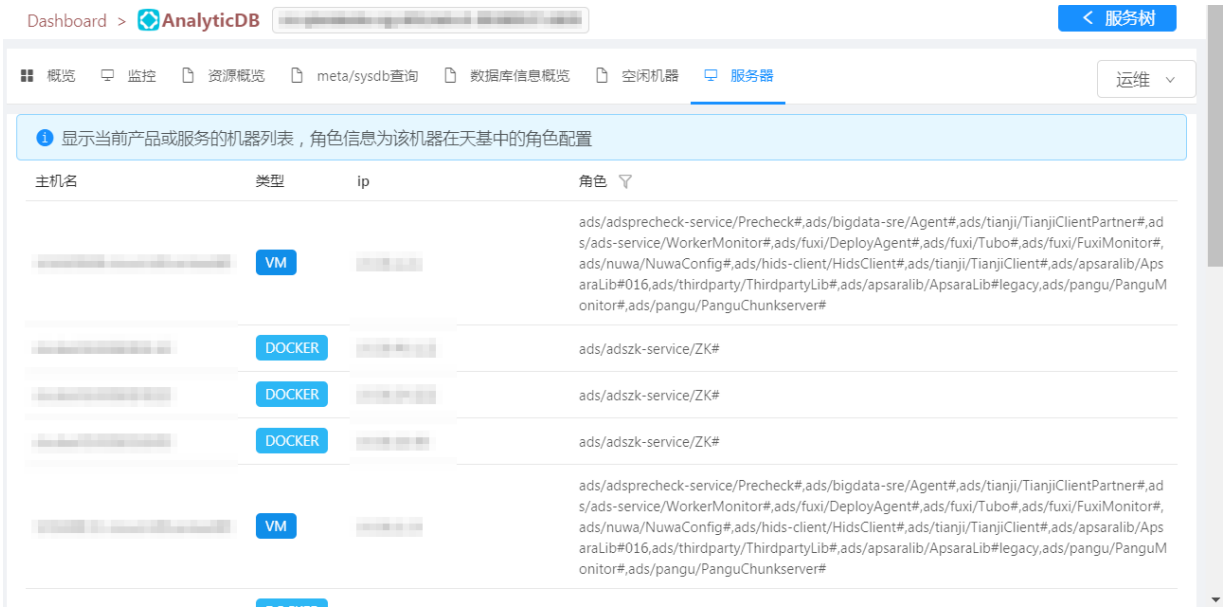
图 7-52: MaxCompute服务配置



7.2.5 服务器

服务器页面是大数据管家展示产品或服务机器列表的页面，如图 *7-53:* 服务器页面所示。

图 7-53: 服务器页面



服务器页面展示当前产品或服务的机器列表，机器信息包括主机名、类型、IP地址和角色。其中角色信息是该机器在天基中的角色配置。

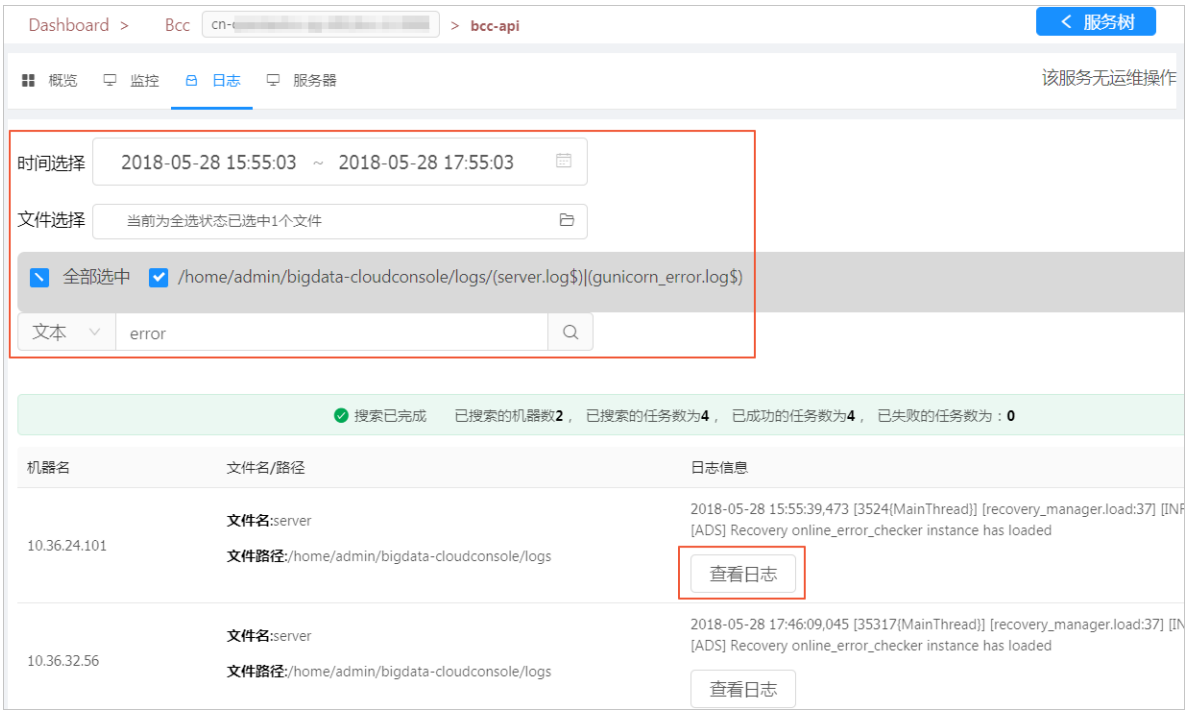
7.2.6 日志搜索

日志搜索用于查询产品中各服务产生的工作日志。本节以查询bcc-api服务的日志为例进行介绍。

操作步骤

1. 进入bcc-api服务页面，选择**日志**页签，进入**日志**页面。
2. 设置搜索条件，查询日志，如图 *7-54:* 搜索条件设置所示，参数说明所示所示。

图 7-54: 搜索条件设置



参数名称	描述
时间选择	设置查询什么时间段的日志。支持最近1小时、最近12小时、最近1天、最近1周、最近1个月、最近3个月、六个快捷时间段快速查询。
文件选择	支持一键全部选中文件进行查询。
文本	支持文本、正则两种文本关键字过滤方式。

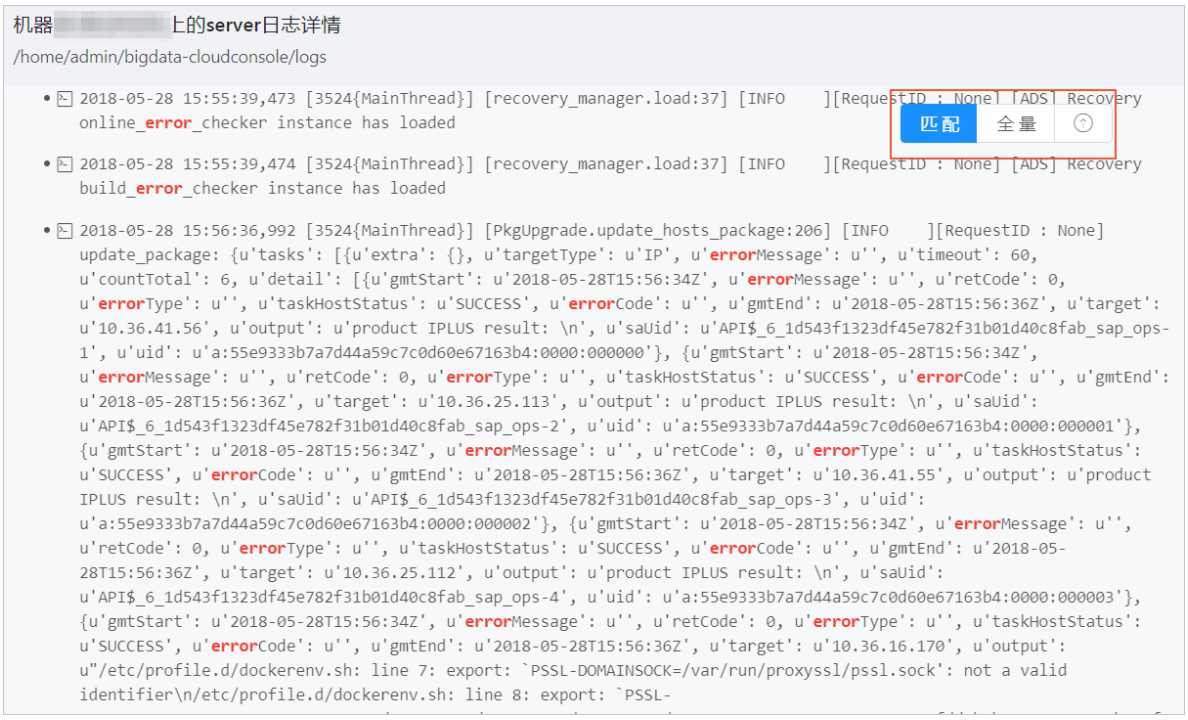
3. 设置好各参数后，单击**搜索**即可。

   搜索过程中，系统会显示搜索状态、已搜索的机器数、已搜索的任务数、成功的任务数为、失败的任务数。完成搜索后，会显示匹配搜索结果包括：机器名、文件名/路径、日志信息。

4. 在搜索结果中单击**查看日志**，查看日志详情，如查看日志详情所示。

> **说明**：
>
> 日志中关键字有标红处理。

图 7-55: 查看日志详情



5. 左侧可单击**点击展开上下文**进行上下文阅览。
6. 可选：日志页面支持匹配、全量两周阅览模式，全量即为全部日志。

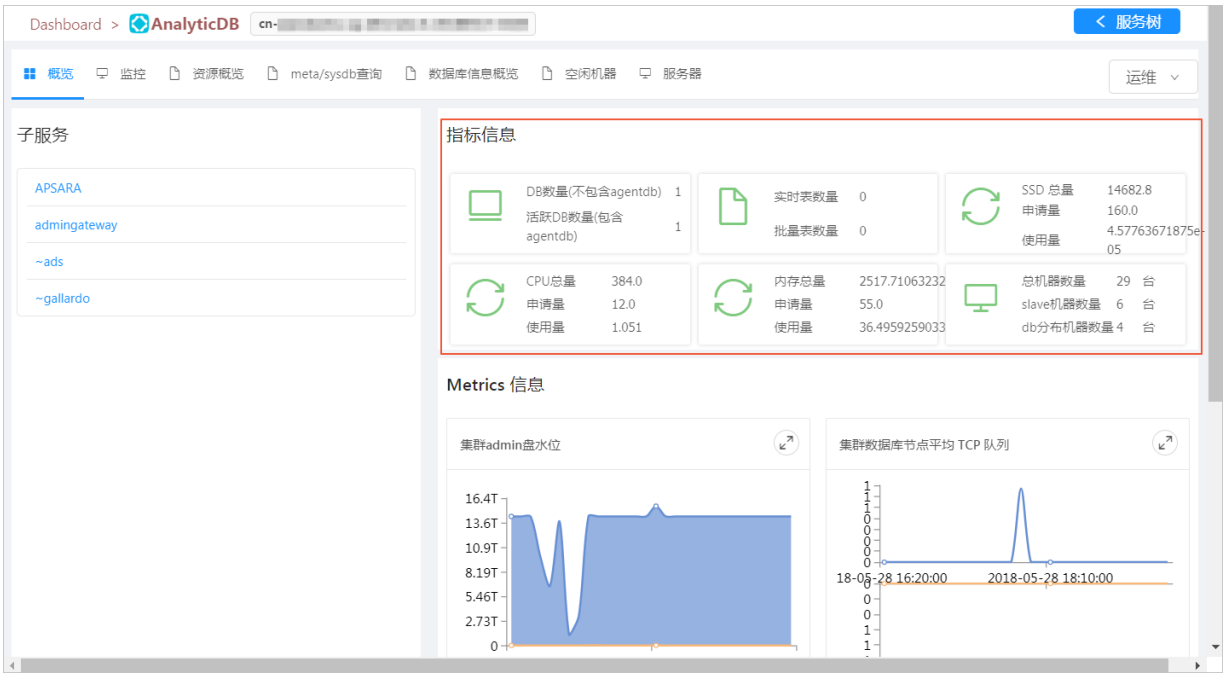
7.2.7 指标信息

指标信息是服务下各个指标数据的展示，您可以通过指标数据查看当前服务的资源使用情况等有效的运维数据。

以AnalyticDB产品为例，AnalyticDB指标信息包括总机器数量、内存总量、DB数量、实时表数量、SSD总量和CPU总量，如指标信息所示。

注意：
不同产品指标信息种类和个数有差异。

图 7-56: 指标信息



7.2.8 Metrics信息

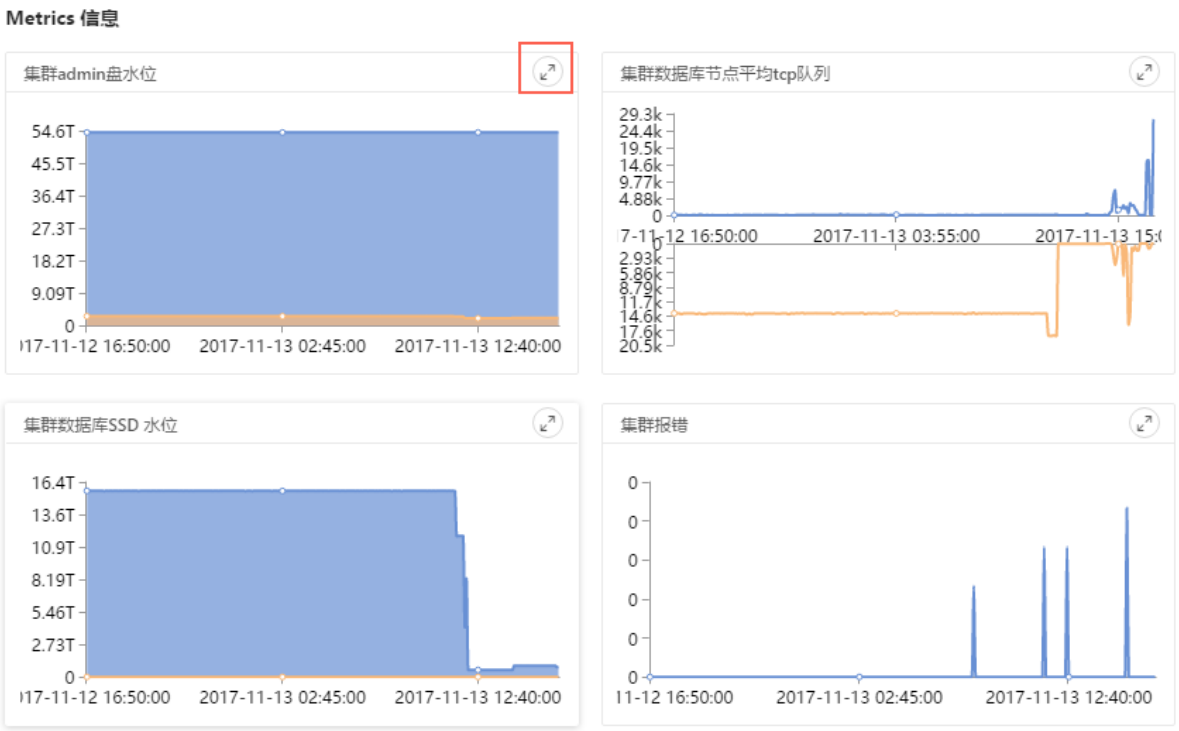
Metrics信息也是图信息，主要是服务下各个指标历史数据的展示，您可以通过Metrics数据查看服务一段时间内的资源使用情况等有效的运维数据。

注意：

不同产品的Metrics信息种类和个数有差异。

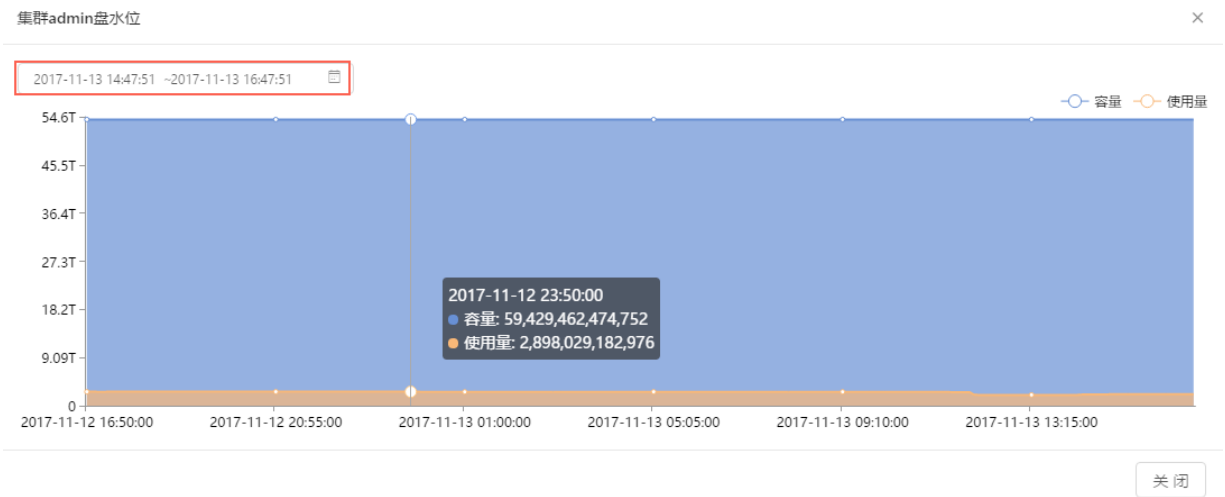
以AnalyticDB产品为例，某AnalyticDB集群2017-11-12 16:50:00-2017-11-13 12:40:00时间段的图信息包括集群admin盘水位、集群数据库节点平均tcp队列、集群数据库SSD水位和集群报错四个指标信息，如图*Metrics信息*所示。

图 7-57: Metrics信息



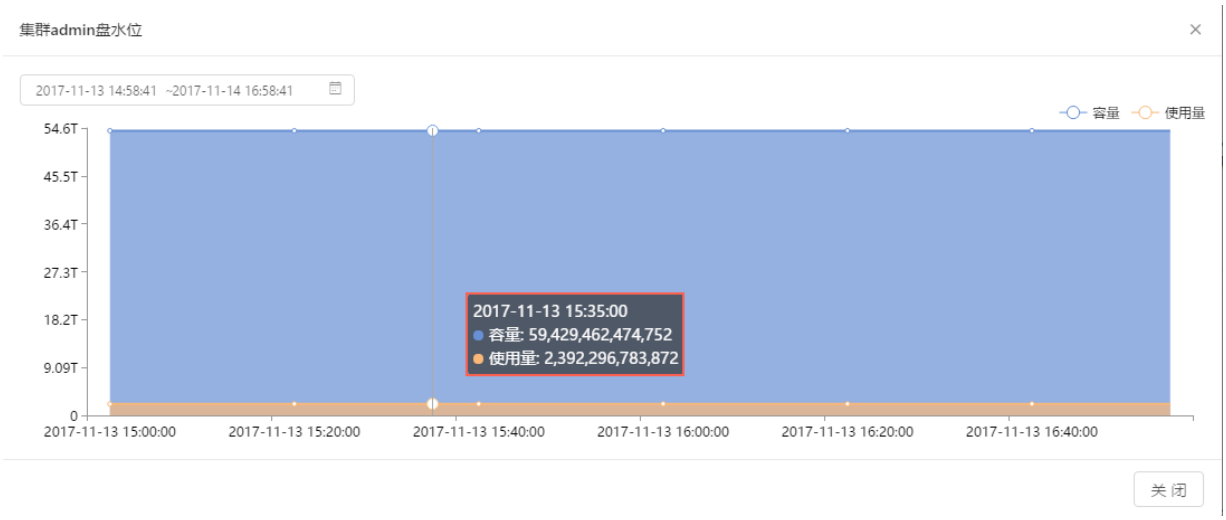
单击放大按钮进入图信息放大图，以当前时间点为基准，支持日期修改，即时间点为当前时间，日期为选择的日期的组合时间段。

图 7-58: 图信息放大图



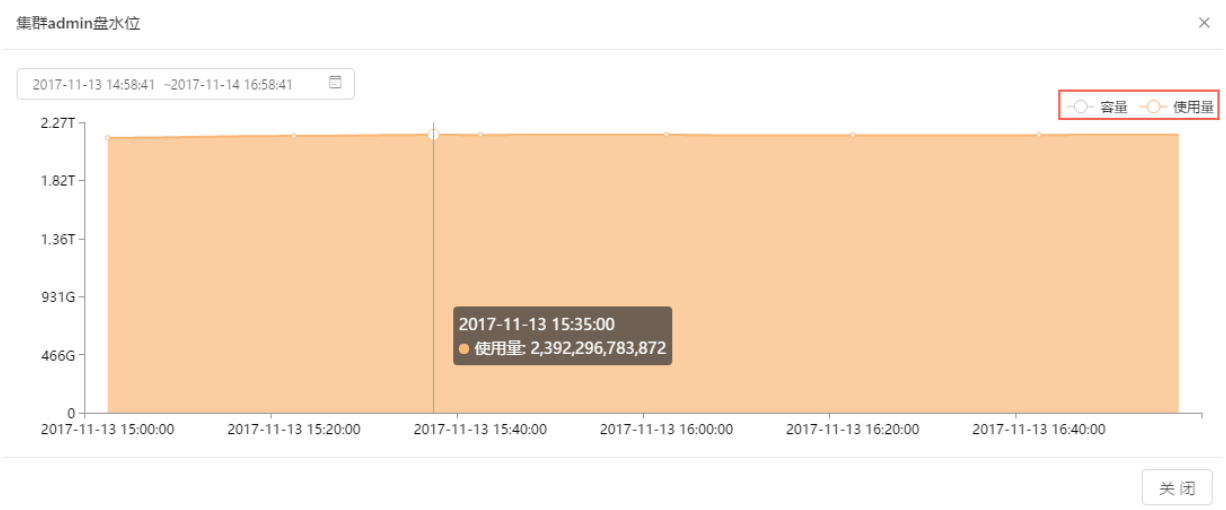
在图信息放大图中，移动鼠标在图的任意位置，会显示当前时间的具体数据包括时间、容量、使用量，如图信息详情所示。

图 7-59: 图信息详情



在图信息放大图右上角的图例展示有颜色表示显示当前曲线，灰色表示隐藏曲线。您可通过单击图例在来切换显示和隐藏，以隐藏容量、显示使用量为例，示例如使用情况所示。

图 7-60: 使用情况



7.2.9 服务树

服务树用于展示出产品的服务树结构，在服务树结构中，单击服务下子服务可以切换到对应的页面。

在大数据管家产品下，单击页面右上角的**服务树**（如图 *7-61:* 进入服务树所示），弹出产品服务树下拉框，如图 *7-62:* 服务树结构所示。

图 7-61: 进入服务树

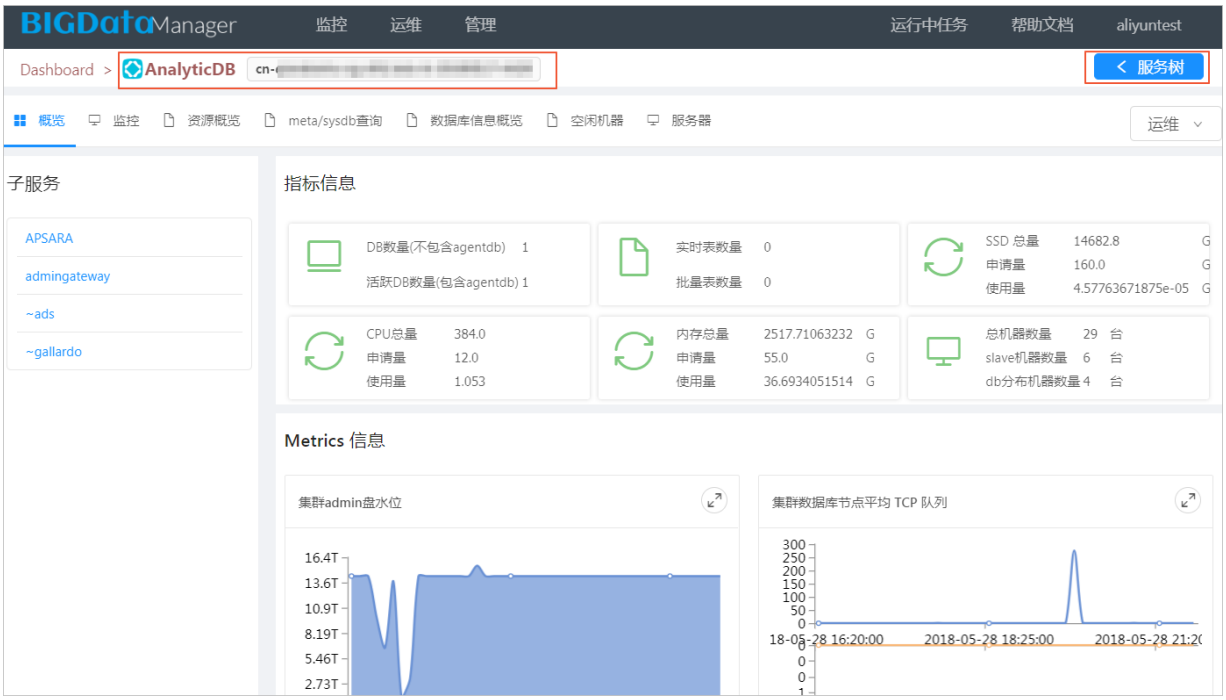
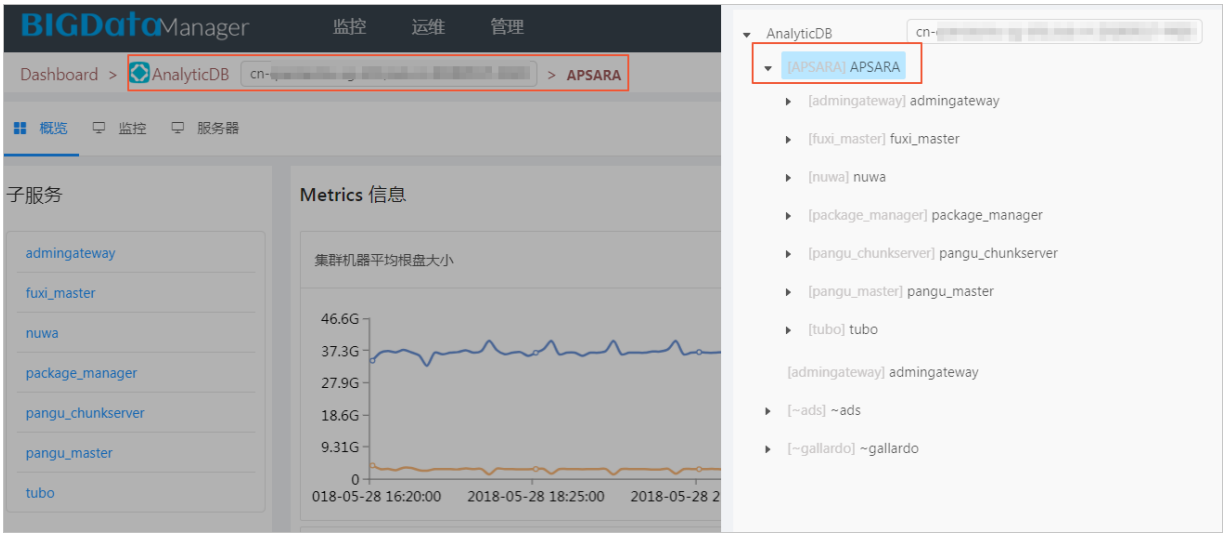


图 7-62: 服务树结构



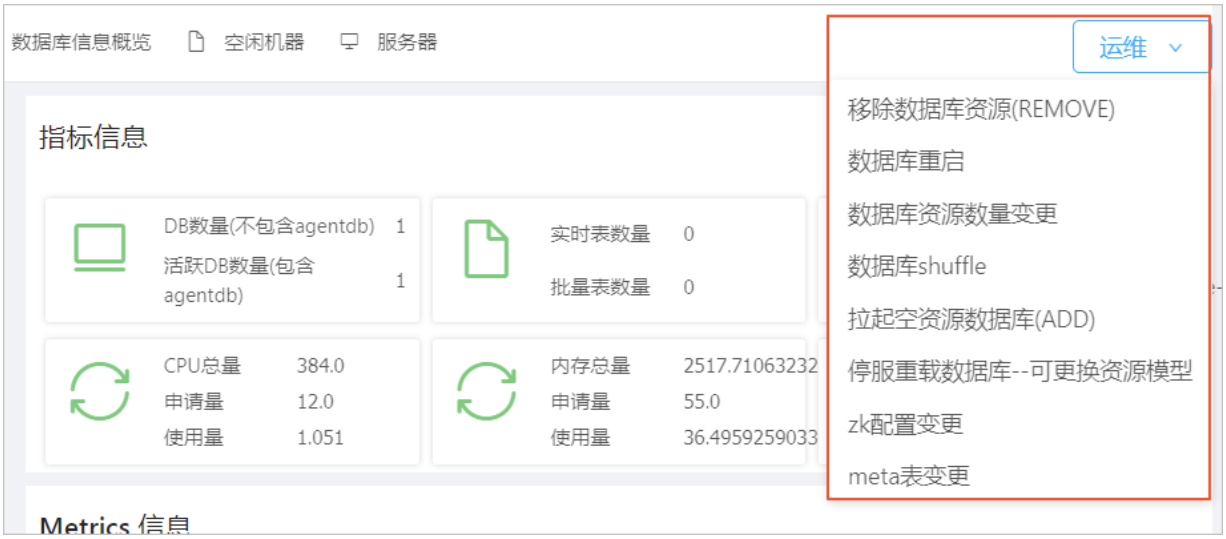
7.2.10 AnalyticDB工作流

工作流是事先根据一系列过程规则定义处理流程和步骤的过程，是封装好的一种框架，能够自动执行的过程。它能解决一些繁琐或者重复性工作。

工作流中所有进行的任务有3种状态：任务进行中、任务成功和任务失败。

AnalyticDB产品中工作流包括：移除数据库资源(REMOVE)、数据库重启、数据库资源数据量变更、数据库shuffle、停服重载数据库--可更换资源模型、zk配置变更、meta表变更、拉起空资源数据库(ADD)。

图 7-63: AnalyticDB工作流



7.2.10.1 移除数据库资源(REMOVE)

删除数据库资源，即释放数据库资源。

图 7-64: remove数据库

7.2.10.2 数据库重启

AnalyticDB产品中数据库重启主要支持COMPUTENODE、BUFFERNODE、FRONTNODE 3个节点的数据库以safe、unsafe两种模式重启。

以BUFFERNODE节点sysdb数据库safe模式重启为例，示例如图数据库重启。

图 7-65: 数据库重启

单击**提交**后会弹出运行对话框，显示重启服务的启动参数、详细进度。您可查看任务失败时的反馈日志。当任务失败时可选择重试、放弃此任务两种方式继续或者结束任务。

图 7-66: 运行对话框

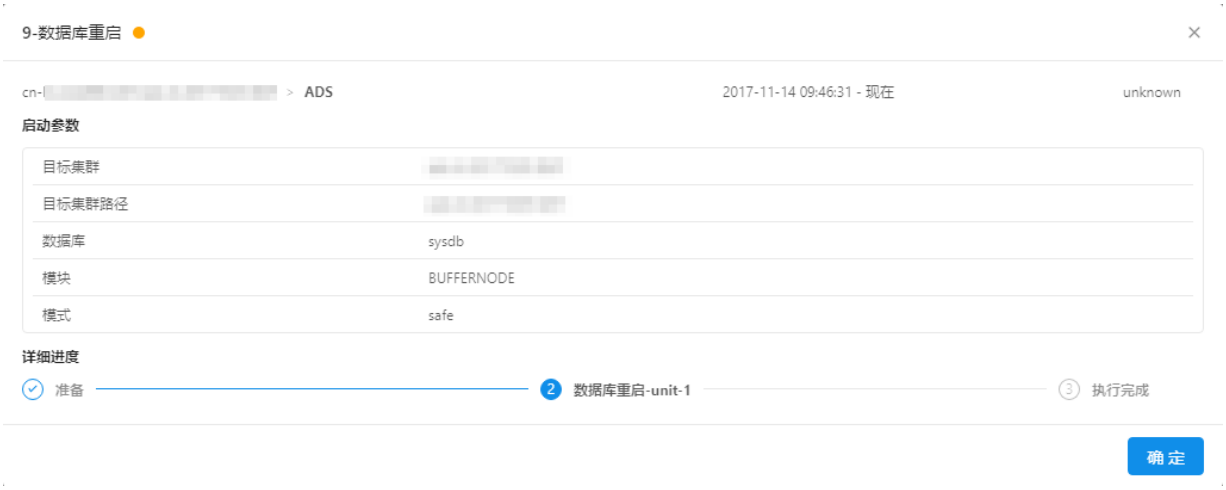
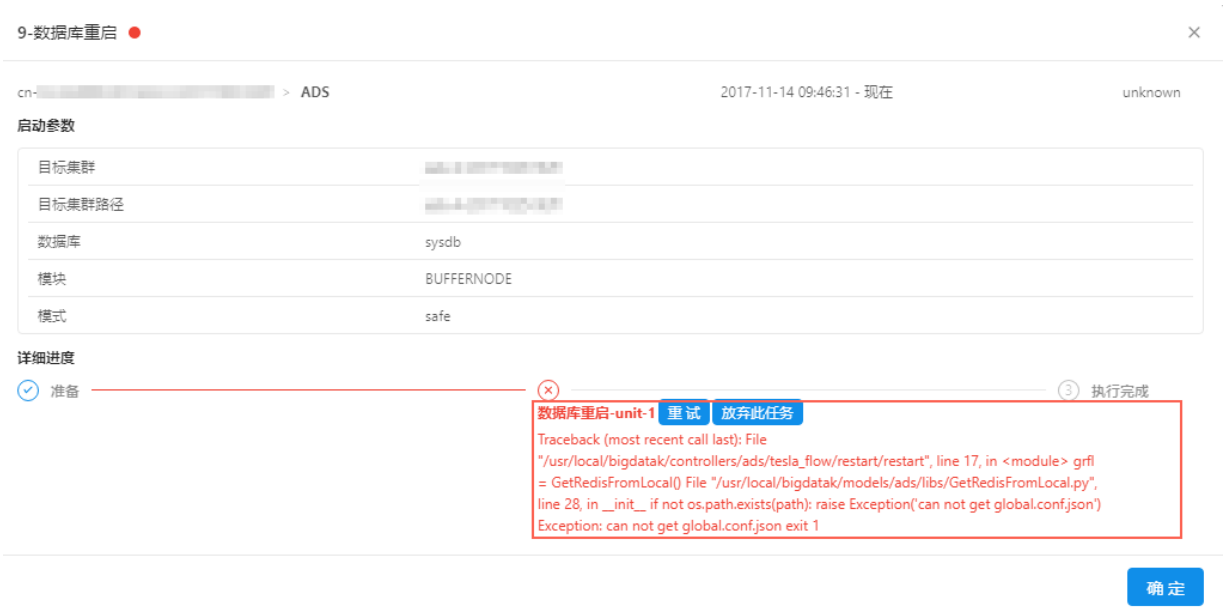
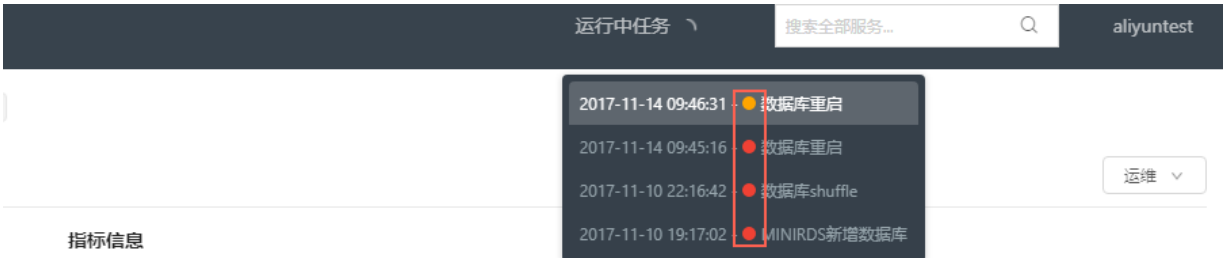


图 7-67: 反馈日志



工作流下所有的执行中、执行失败的任务都可以在运行中任务中看见，如图查看任务所示，橙色表示运行中的任务，红色表示运行失败的任务。

图 7-68: 查看任务



7.2.10.3 数据库资源数据量变更

数据库资源数据量变更用于对数据进行扩容或缩容。

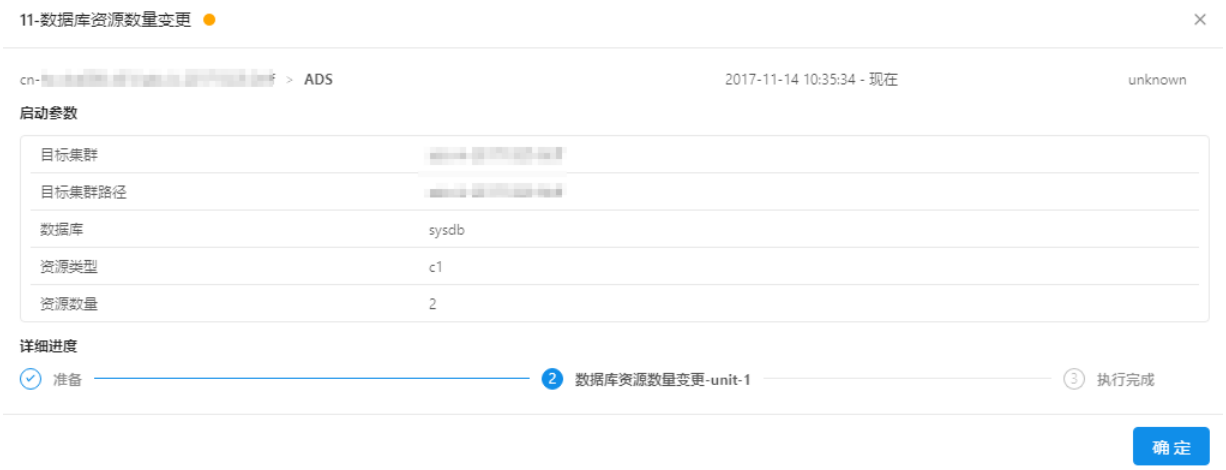
数据库资源数据量变更的参数包括：数据库、资源类型（主要有：c1、c8、s1、s8，测试一般用c1，c1最小。每个类型代表使用的资源不一样）、资源数量（必须为偶数）。

如图数据库资源数量变更所示，更改SYSDB数据库，资源类型为c1，资源数量为2。

图 7-69: 数据库资源数量变更

同数据库重启一样，工作流下的所有任务提交后都有任务执行对话框，任务成功，任务失败后的日志信息，以及任务失败后可重启、放弃任务后续操作。以下不再重复，详情请参见数据库重启。

图 7-70: 启动参数

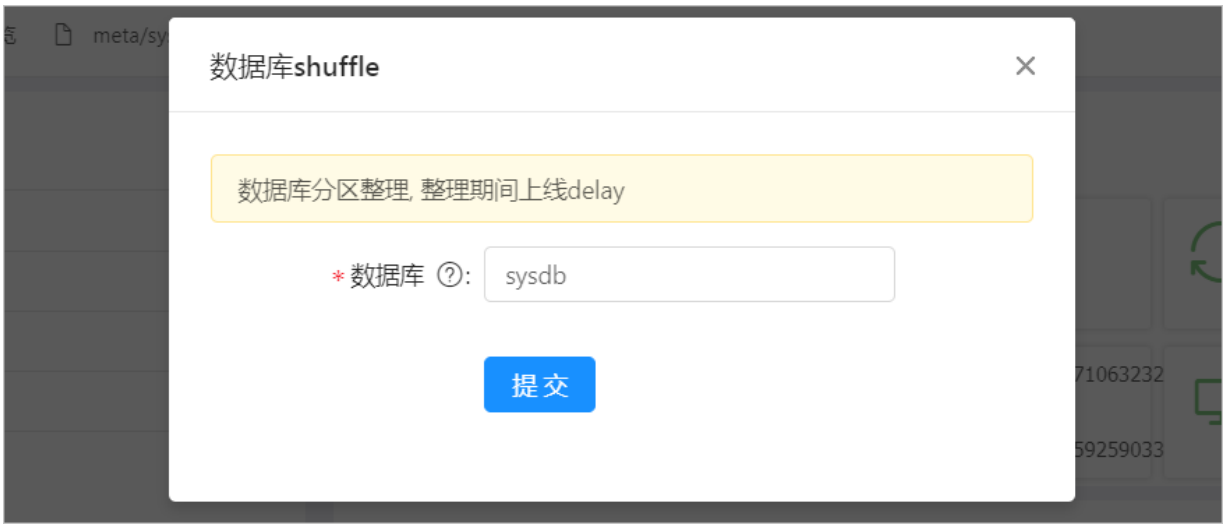


7.2.10.4 数据库shuffle

数据库shuffle用于重新规划分区。如果数据库表分区不均匀，则可能导致各别节点数据量特别大（即所谓的分区倾斜），此时需要重新规划表分区。

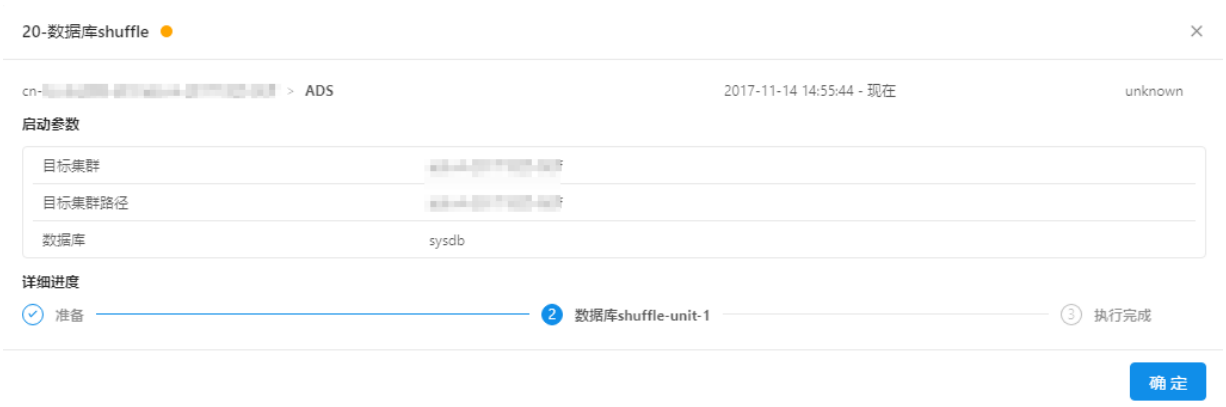
以sysdb数据库为例，执行数据库shuffle的示例如重新规划*sysdb*数据库表分区所示。

图 7-71: 重新规划sysdb数据库表分区



提交任务后进入任务执行对话框，等待几秒显示任务执行结果，任务失败有对应日志信息，以及任务失败后可重启、放弃任务后续操作。以下不再重复，详情请参见数据库重启。

图 7-72: 任务执行对话框



7.2.10.5 拉起空资源数据库(ADD)

ADD数据库就是AnalyticDB中的扩容ECU，需要提供参数：数据库、资源类型、资源数量。图中是扩容sysdb，资源类型为c1，资源数量为2。

图 7-73: 拉起空资源数据库(ADD)

7.2.10.6 停服重载数据库--可更换资源模型

停服更改数据库用于更改数据库资源模型、数据数量，执行期间数据库不提供服务。

以将修改数据库sysdb数据库的资源类型修改为c1、资源数量修改为2为例，示例如停服重载数据库所示。

图 7-74: 停服重载数据库

7.2.10.7 zk配置变更

zk配置变更，包括的参数有：path、key、value、type。

Type默认可选类型有：int、float、string、double、bool，下拉框中可见。

如图*zk*配置变更所示，修改路径/dbmg/service/10_1_40_36_9999下key为**port**，value为9999，数据类型为string。

当变更的key值为原配置中不存在的key时，添加该key配置。

图 7-75: zk配置变更

7.2.10.8 meta表变更

meta表变更用于更改AnalyticDB数据库的元数据信息。

meta表变更的参数为：执行语句、数据库（不填时默认为meta库）。执行语句列支持多个sql变更语句。

以修改meta库里的某用户对指定数据库的权限为例，示例如*meta*表变更中的SQL如下：

```
insert into privilege_schema values('$[cluster_name]','$[schema_id
]','$[schema_name]','$[creator_id]','$[creator_name]','$[creator_id
]','$[creator_name]','$[creator_id]','$[creator_name]','ALL');
```

图 7-76: meta表变更

7.2.11 大数据管家各分站特殊功能展示

大数据管家各组件的功能基本都是一致的，都包括产品树结构、配置、工作流、日志搜索、指标信息和Metrics信息功能，各组件也有自己独特的亮点服务。

7.2.11.1 采集插件配置

大数据管家产品配置项下的采集插件配置主要对应大数据管家拓扑结构图。

配置页面的视图支持：Code、View两种浏览模式。Code模式为代码按顺序依次全展示，View模式为关键参数的键值对。

图 7-77: 采集插件配置-Code模式

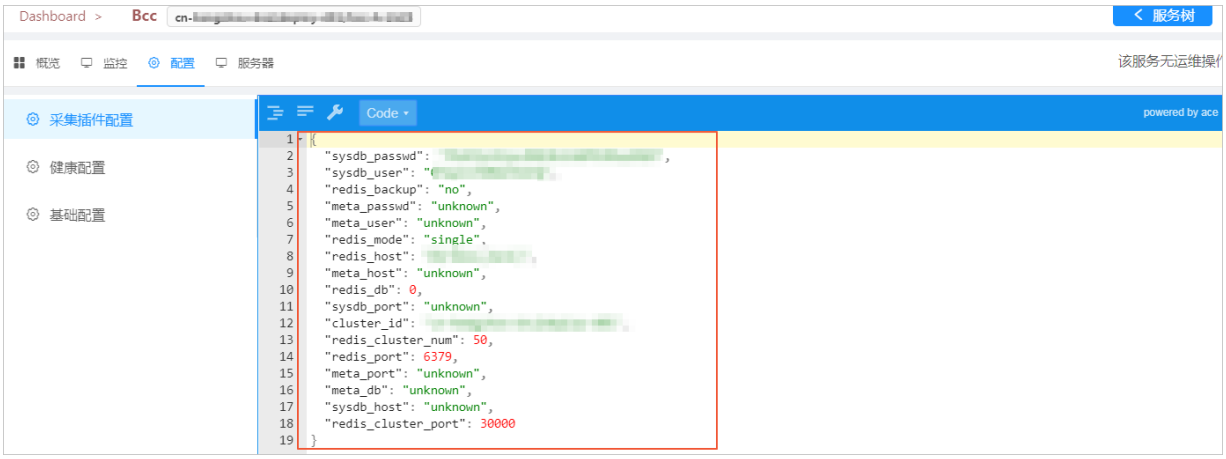
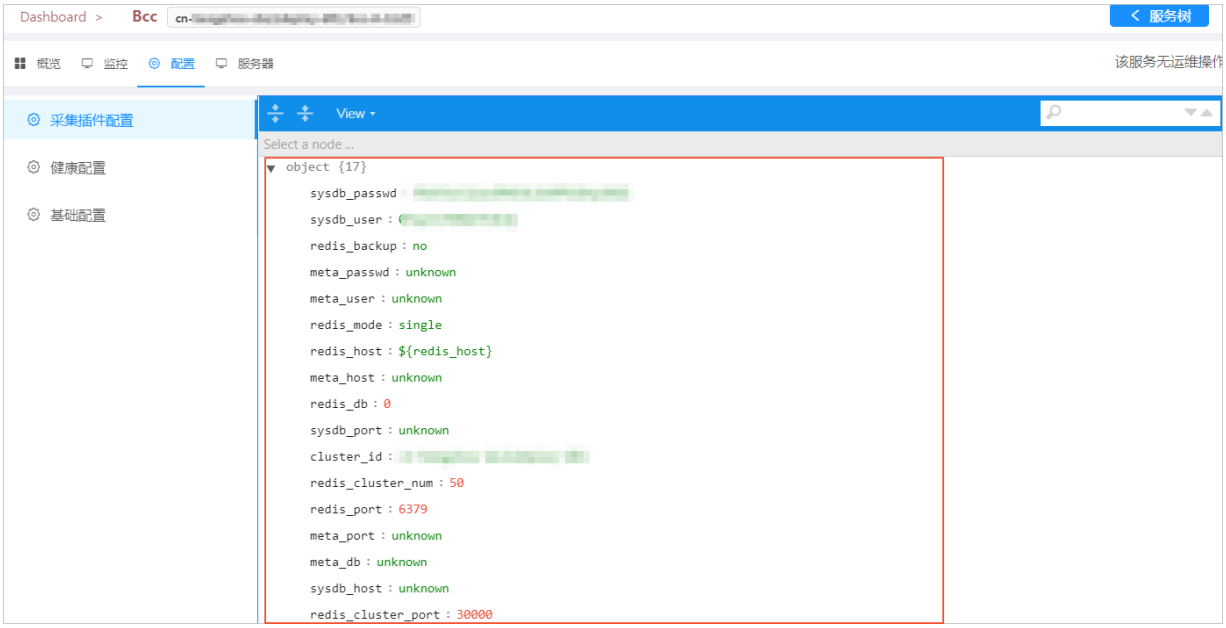


图 7-78: 采集插件配置-View模式



7.2.11.2 健康配置

大数据管家产品配置项下的健康配置主要为各组件的检查配置信息。

配置页面的视图支持：Code、View两种浏览模式。Code模式为代码按顺序依次全展示，View模式为关键参数的键值对。

图 7-79: 健康配置-Code模式

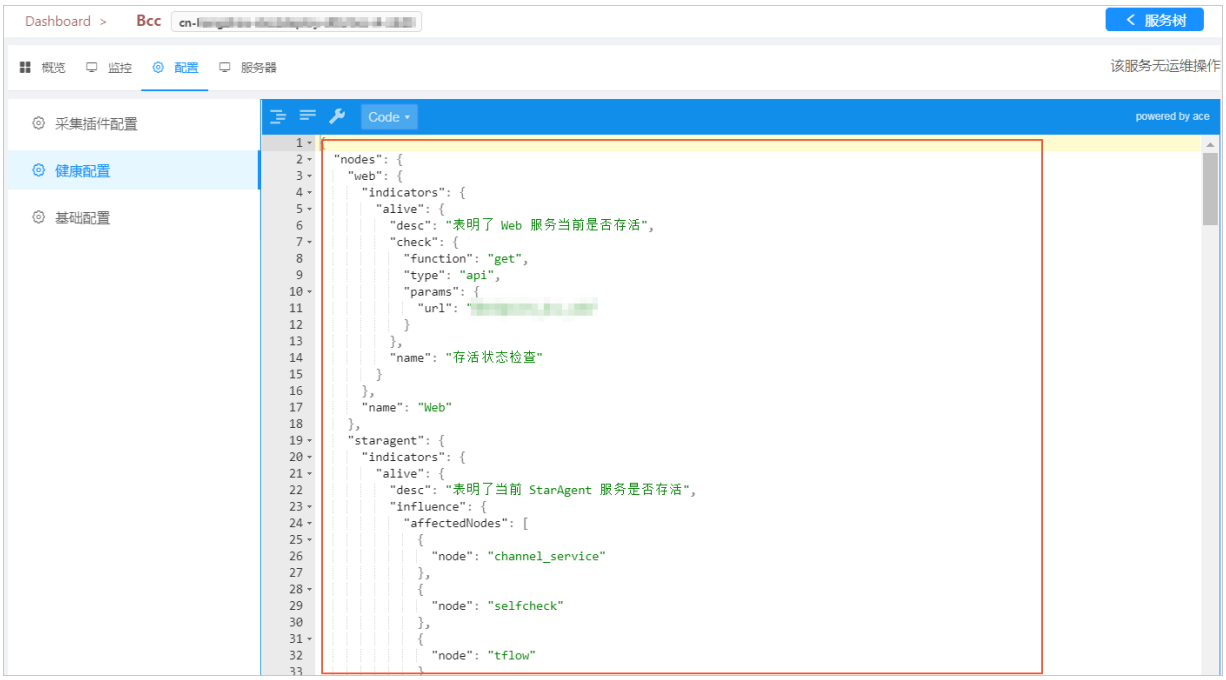
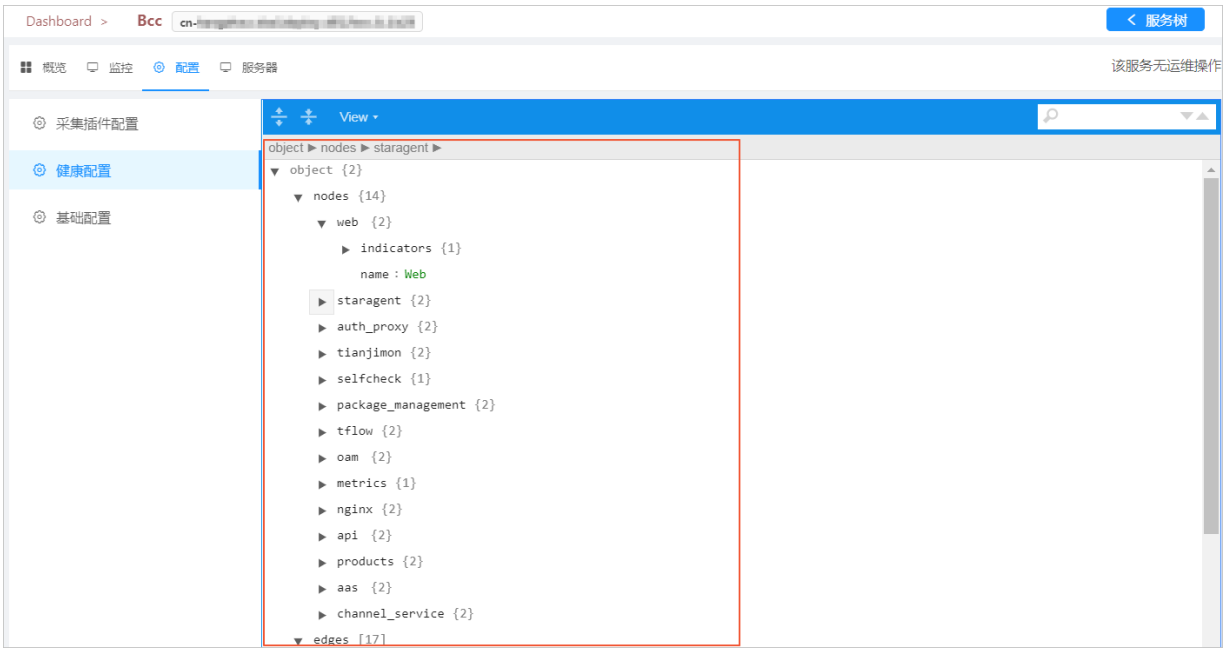


图 7-80: 健康配置-View模式



7.2.11.3 AnalyticDB

本节介绍AnalyticDB产品下的空闲机器，以及子服务~ads > dbs > sysdb的节点信息。

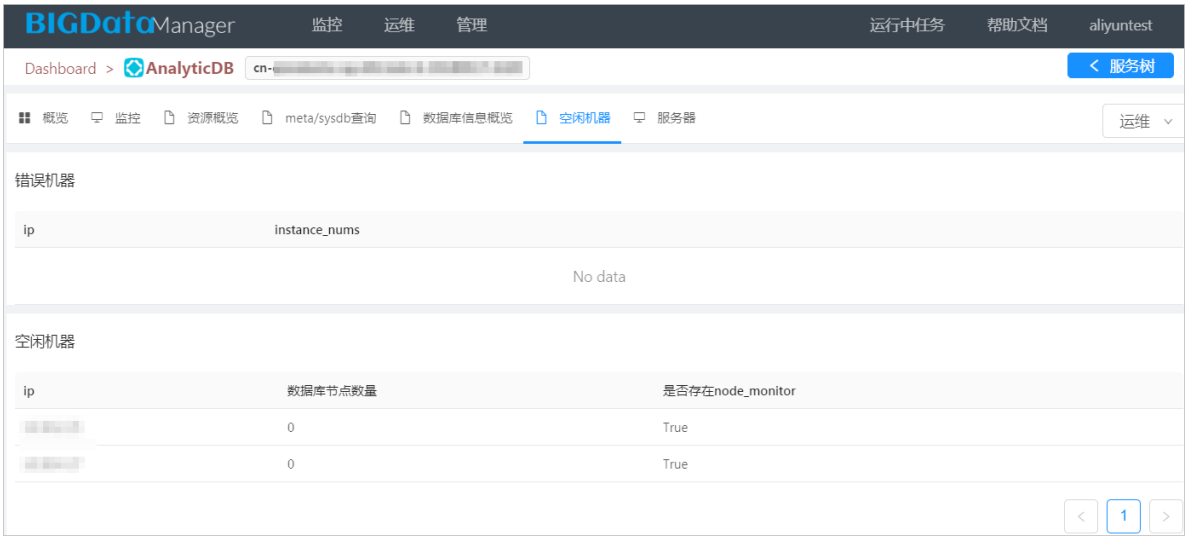
7.2.11.3.1 空闲机器

空闲机器页面用于展示错误机器、空闲机器的IP、实例数目。

您可通过以下操作查看空闲机器：

1. 在**监控**界面，单击**AnalyticDB**图标中的产品名称进入AnalyticDB产品界面。
2. 选择**空闲机器**，查看AnalyticDB的空闲机器 信息。

图 7-81: 空闲机器



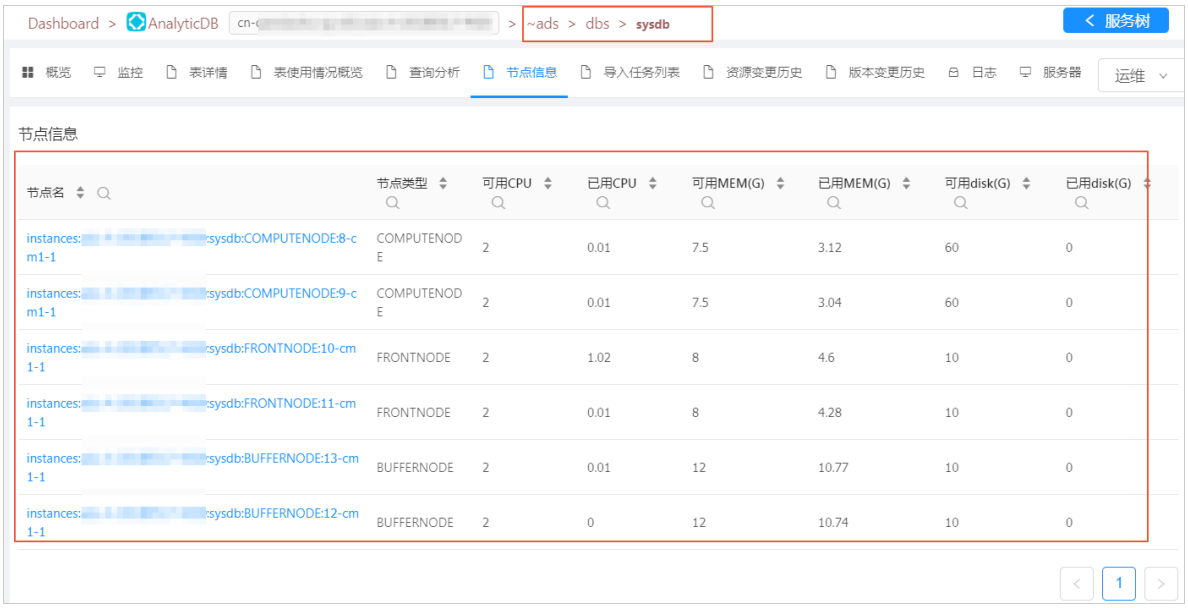
7.2.11.3.2 节点信息

sysdb下的节点信息用于展示AnalyticDB下COMPUTENODE、FRONTNODE、BUFFERNODE 3个节点的详细信息。

您可通过以下操作查看节点信息：

1. 在**监控**界面，单击**AnalyticDB**图标中的产品名称进入AnalyticDB产品界面。
2. 在**子服务**列表中单击**~ads**。
3. 在**子服务**列表中单击**dbs**。
4. 在**子服务**列表中单击**sysdb**， 然后选择**节点信息**来查看Analytic的节点信息，如下图所示。

图 7-82: 节点信息



节点信息包括节点名、节点类型、可用cpu、已用cpu、可用mem、已用mem、可用disk、已用disk 8个参数。在AnalyticDB中，节点以副本形式存在，一共6条信息。

7.2.11.4 DataWorks

DataWorks产品提供了特有服务：租户管理、计算引擎、base-biz-alisa资源管理和base-biz-alisa Gateway管理。

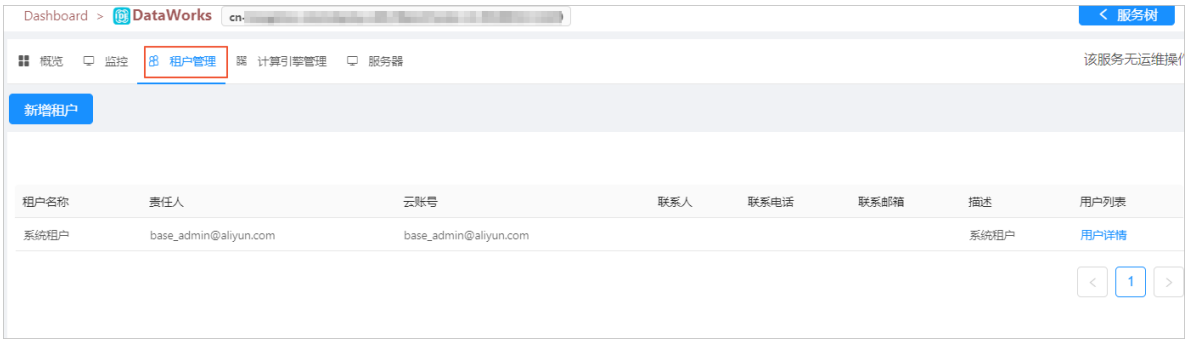
7.2.11.4.1 租户管理

DataWorks租户管理下提供了产品租户的系统管理，显示的已有租户信息包括：租户名称、责任人、云账号、联系人、联系电话、联系邮箱、描述、用户列表。租户管理下提供新增租户、租户添加用户、租户移除用户等操作。

新增租户

1. 在大数据管家任意界面选择**管理** > **租户管理**，进入**租户管理**界面，如下图所示。

图 7-83: 租户管理



2. 单击**新增租户**，填写创建租户参数信息。**创建租户**对话框和关键参数说明如图 *7-84:* 创建租户和表 *7-1:* 关键参数说明所示。

图 7-84: 创建租户

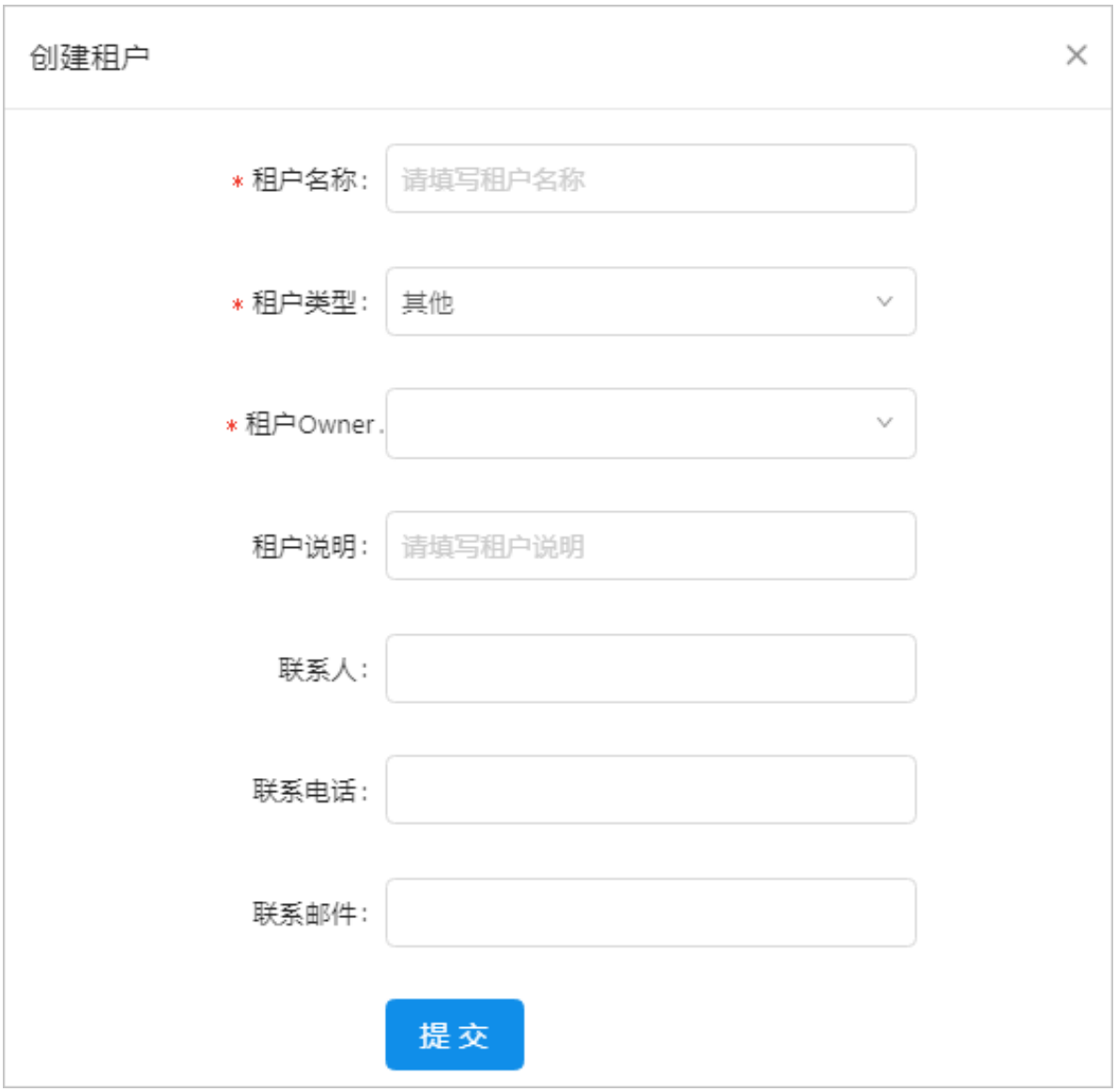


表 7-1: 关键参数说明

参数名称	描述
租户名称	不可与已有租户名称重复。
租户类型	根据实际选择，可选值包括：其他、商家、ISV、企业。 ISV：Independent Software Vendors
租户Owner	租户的拥有者，可选值为系统中的云账号。 一个云账号只能作为一个租户的租户Owner，不能跨租户。一个云账号成为一个租户的租户Owner后，再次创建租户时，将

参数名称	描述
	不再显示在**租户Owner**下拉列表中。创建租户时系统会自动匹配无租户的Owner（不匹配系统用户）。

3. 提交创建租户后，会自动回到租户管理主页，您可在租户列表中查看新增的租户。

图 7-85: 新增租户-示例结果

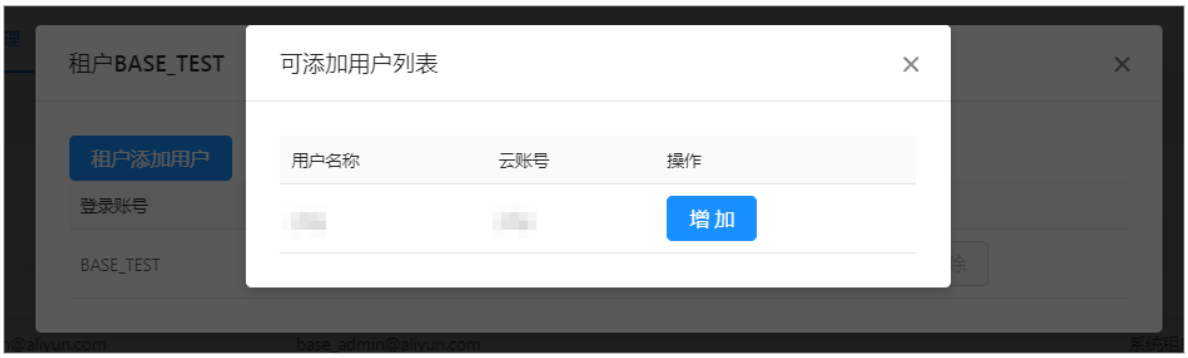
租户添加用户

1. 添加租户后，单击租户后面的**用户详情**进入租户租户用户详情。

图 7-86: 租户用户详情

2. 单击**租户添加用户**，进入图 *7-87: 可添加用户列表*，列表中列出了当前可添加为租户用户的所有云账号。

图 7-87: 可添加用户列表



3. 单击某个云账号的**增加**即可添加该云账号为租户用户。

   添加完成后，您可在图 *7-88:* 租户列表中查看新添加的租户用户。

> **说明**：
> 一个云账号只能添加到一个租户中。云账号添加到某租户后，该云账号将不会在任何租户的图 *7-87:* 可添加用户列表中显示。

图 7-88: 租户列表

租户移除用户

如果多用户租户组中某个用户不再使用，则您可进行用户移除，但不能移除租户Owner。

在图 *7-88:* 租户列表中，单击租户用户的**移除**，从租户中删除该用户。

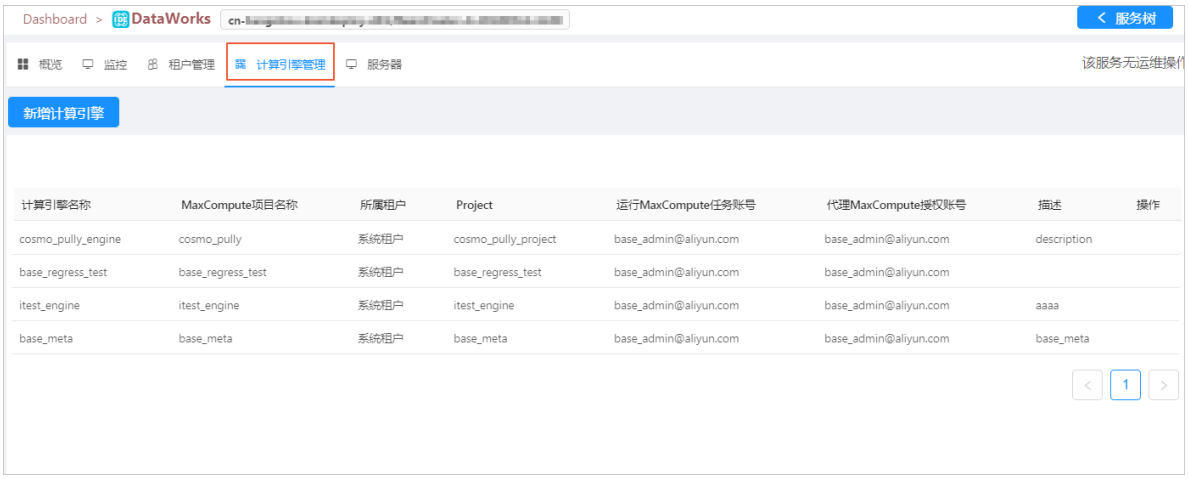
7.2.11.4.2 计算引擎

计算引擎下展示了目前的计算引擎信息，包括：计算引擎名称、MaxCompute项目名称、所属租户、Project、运行MaxCompute任务账号、代理MaxCompute授权账号、描述、操作。计算引擎页面提供新增和删除计算引擎操作。

新增计算引擎

1. 在大数据管家任意界面选择**管理** > **计算引擎**，进入**计算引擎管理**界面，如下图所示。

图 7-89: 计算引擎管理



2. 单击**新增计算引擎**，根据实际需要填写各参数。**新增计算引擎**对话框和参数说明分别如下所示。

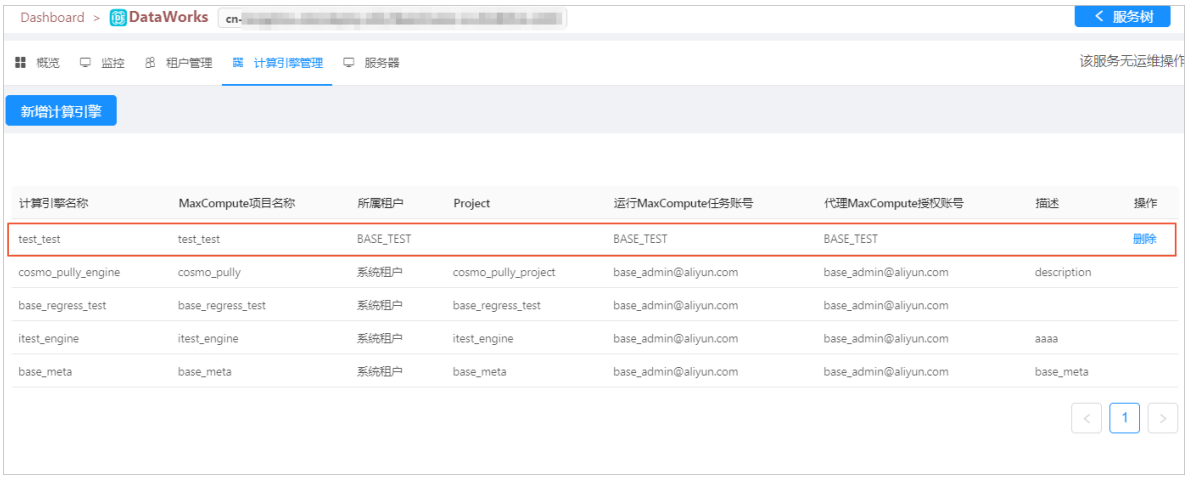
图 7-90: 新增计算引擎

表 7-2: 计算引擎参数说明

参数名称	描述
计算引擎名称	按参数提示填写一个合法的名称即可。
隶属租户	根据实际需要下拉列表中选择一个租户。
MaxCompute项目OWNER	下拉列表中选择，可选值为所选**隶属租户**下的租户用户。
运行MaxCompute任务用户	下拉列表中选择，可选值为所选**隶属租户**下的租户用户，但该值不可与**MaxCompute项目OWNER**相同。
描述	计算引擎的描述内容。

3. 单击**确认**，您可在计算引擎首页查看新建的计算引擎。

图 7-91: 新增计算引擎-示例结果



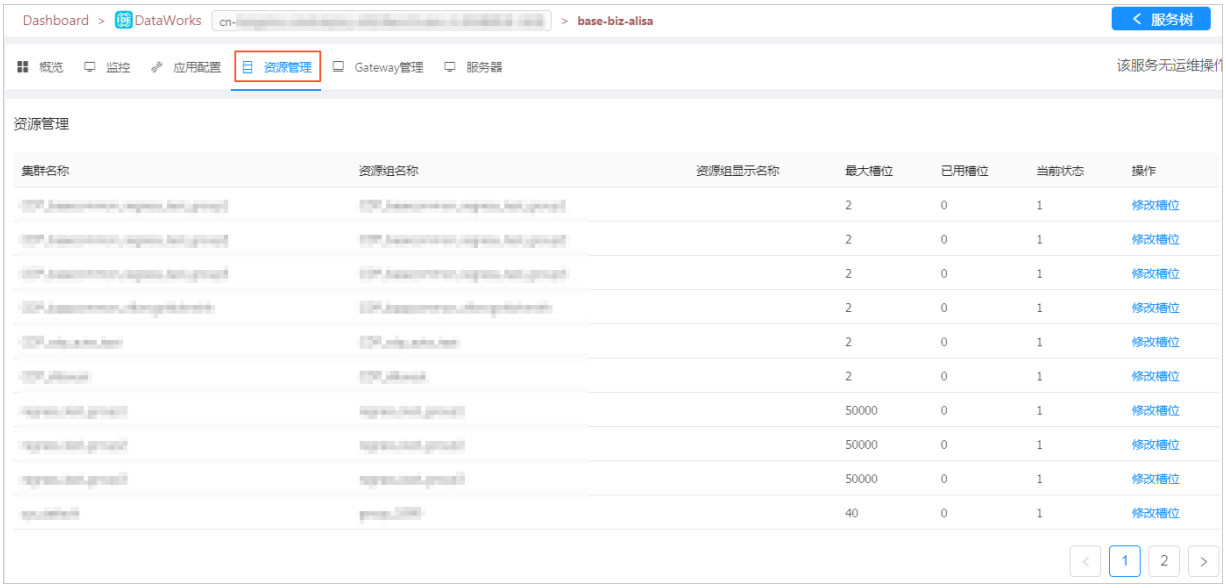
删除计算引擎

对于不再使用的计算引擎，单击该计算引擎后面的**删除**，即可删除该计算引擎。

7.2.11.4.3 base-biz-alisa资源管理

资源管理下展示了集群资源组的详细信息，包括集群名称、资源组名称、资源组显示、最大槽位数、已用槽位数、当前状态、操作。

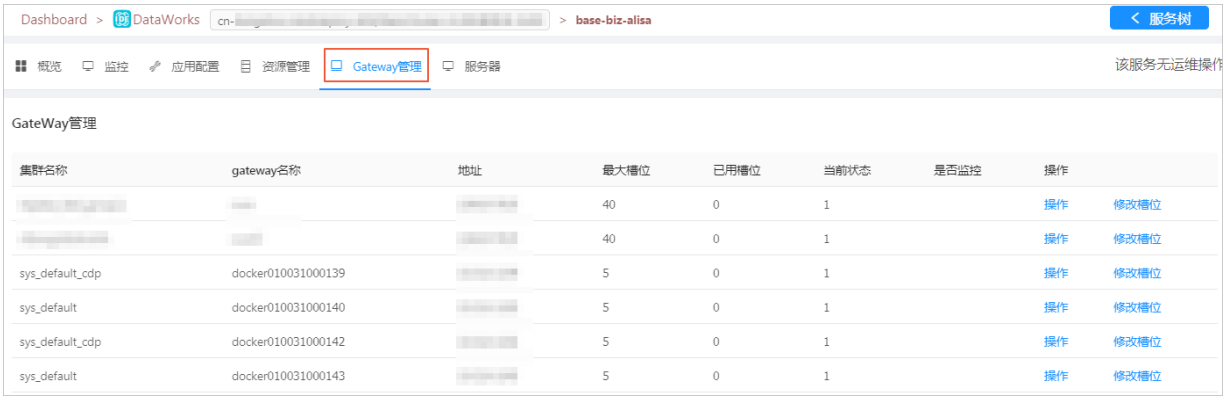
图 7-92: 资源管理



7.2.11.4.4 base-biz-alisa Gateway管理

Gateway管理下展示了当前Gateway的详细记录，包括：集群名称、gateway名称、地址、最大槽位、已用槽位、当前状态、是否监控、操作。

图 7-93: Gateway管理



7.2.11.5 Apsara

Apsara是Analytic DB、MaxCompute等依赖的小飞天的底层产品组件。使用Apsara产品可以更好的运维和展示大数据产品。

8 关系网络分析I+

8.1 管理员指南

8.1.1 什么是关系网络分析

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、关系网络和时空分析三大部分。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。

8.1.2 登录关系网络分析管理后台

本章介绍如何登录I+管理控制台进入具体的管理配置界面，以便进行I+配置操作。

前提条件

在登录I+管理控制台前，您需要向管理员申请开通账号，并授予控制台的访问权限。

操作步骤

单击右上角用户信息菜单栏中的**后台管理**项，将会在浏览器中会重新打开一个Tab页，无需输入用户名密码等信息，可直接进入管理控制台页面，如图 *8-1:* 管理控制台界面所示。

图 8-1: 管理控制台界面

8.1.3 配置源数据

8.1.3.1 配置数据源

进行源数据配置时，请先获取对应的源数据链接并确保可正常访问，数据源不能重复添加，如果数据源已经被使用，将不允许删除。

数据源作为新建对象、关系（新建的对象或关系，前端用户需要重新登录后，才可使用）的配置入口之一，也是对象、关系与物理表映射的唯一入口。另一新建对象、关系的入口是对象信息、关系信息。但是从对象信息、关系信息入口新建的是纯业务逻辑OLEP（无映射），且需要同时通过属性Tab的“新增一行”配置逻辑属性。

8.1.3.1.1 新建数据源

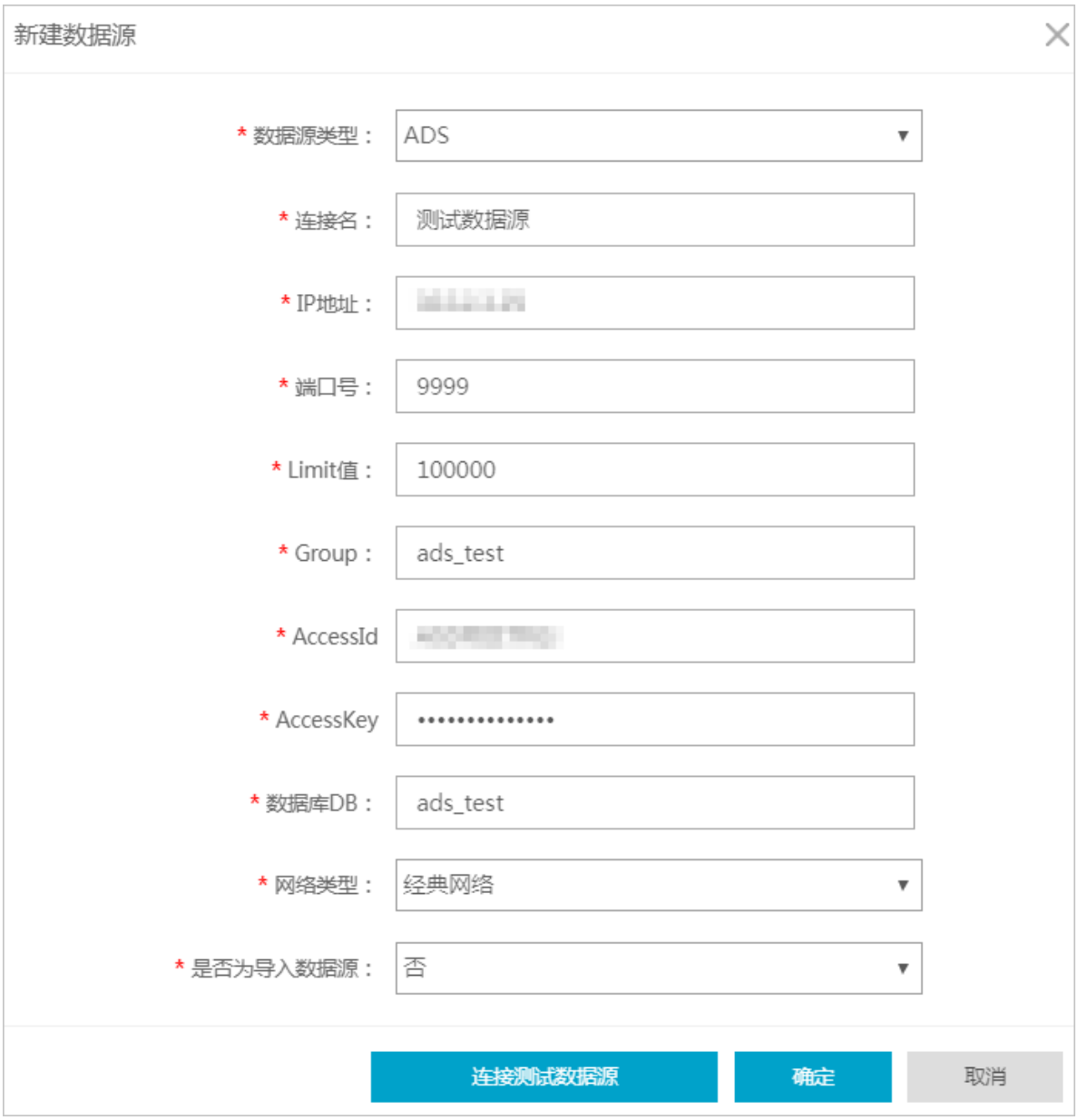
操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**源数据表**页签，进入数据源页面，如图 *8-2:* 数据源页面所示。

图 8-2: 数据源页面

3. 点击**新建数据源**图标。
4. 在弹出的新建数据源窗口中，填写正确的数据源信息，如图 *8-3:* 填写数据源信息所示。

图 8-3: 填写数据源信息



参数说明如下：

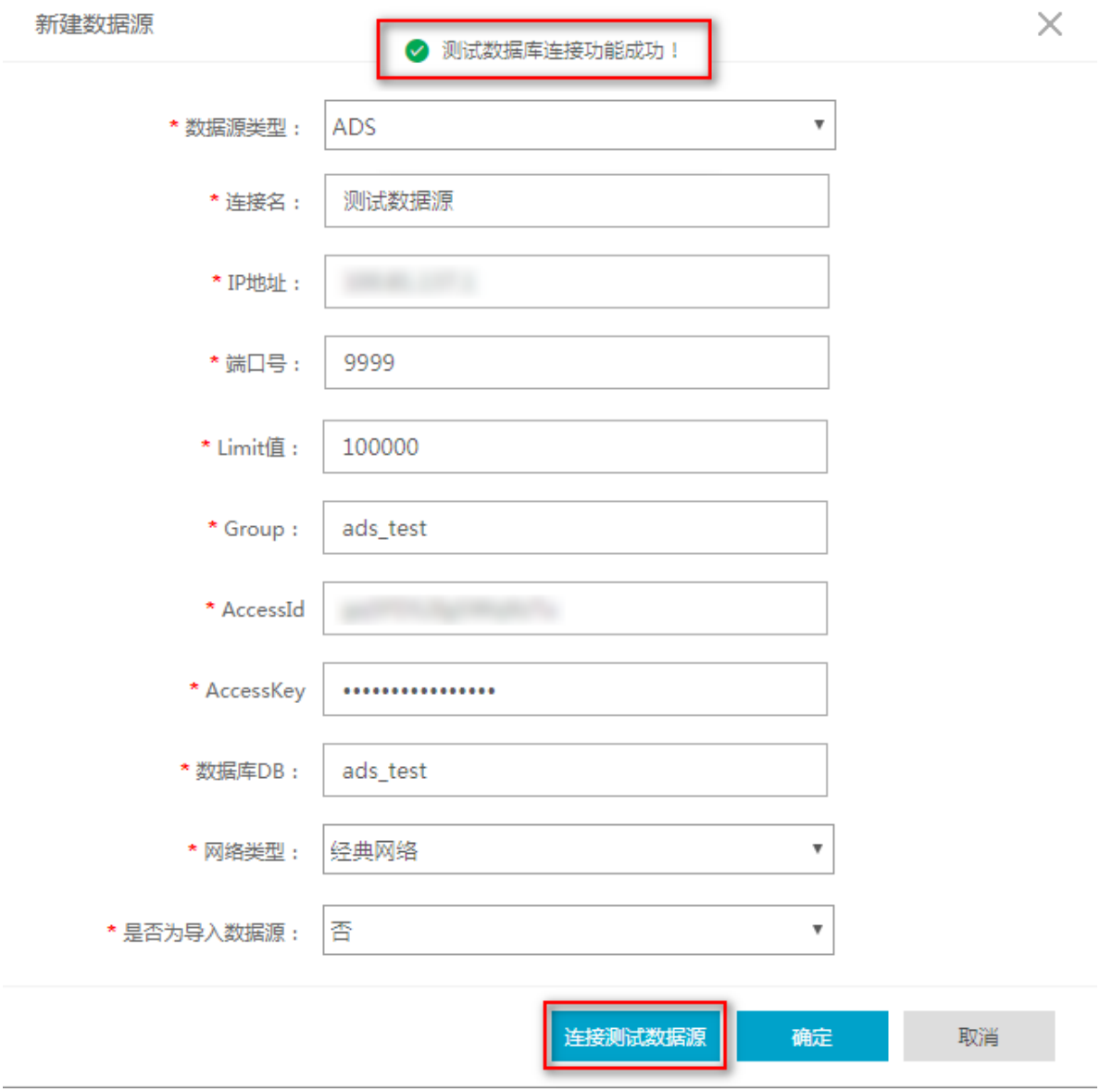
- 数据源类型：可下拉选择，选项包括ADS、MYSQL、ORACLE、RDS和GREENPLUM，根据实际数据源的类型选择。
- 连接名：用于标识数据源的名称，可自定义填写。
- IP地址：数据源对应的IP地址。
- 端口号：数据源对应的端口号。
- Limit值：当选择的数据源为ADS时，需要根据ADS实际情况填写Limit值，一般为10000。

- Group：数据库所在的分组。
- AccessId/AccessKey：当选择的数据源为ADS时，需要根据ADS实际情况填写AccessId和AccessKey。
- 用户名/密码：请根据数据源的实际情况进行填写。
- 数据库DB：请根据数据源的实际情况进行填写。
- 网络类型：选择网络类型为经典网络或VPC网络。
- 是否为导入数据源：选择某一数据源，作为系统导入数据源的配置。每个租户有且只能配置一个导入数据源。

5. 单击**连接测试数据源**按钮，测试所填写内容是否正确。

   测试成功，会出现提示，如图 *8-4:* 连接测试数据源所示。

图 8-4: 连接测试数据源



6. 单击**确定**按钮，添加数据源成功，如图 *8-5:* 数据源添加成功所示。

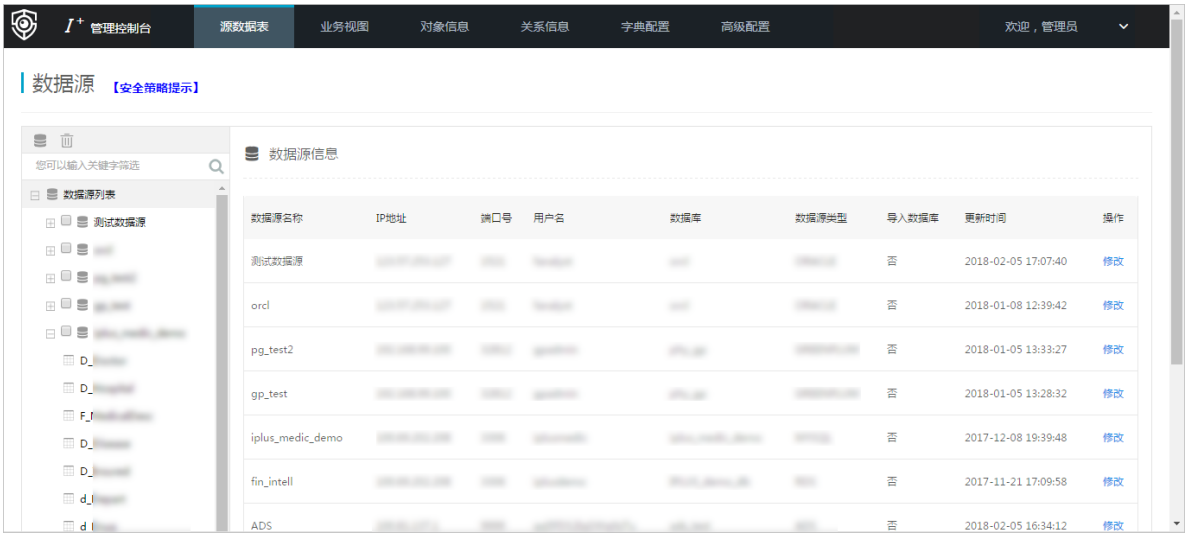
图 8-5: 数据源添加成功

8.1.3.1.2 查看数据源

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**源数据表**页签，进入**数据源**页面，如图 *8-6: 数据源*所示。

图 8-6: 数据源



3. 单击左侧导航树中的某一个数据源，可查看该数据源定义的具体信息，如图 *8-7: 查看某个数据源*所示。

图 8-7: 查看某个数据源

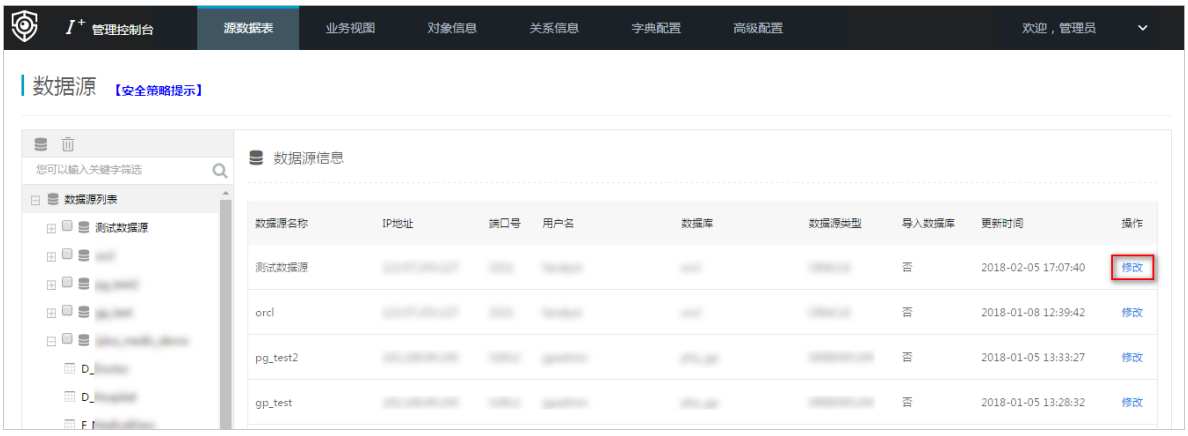
8.1.3.1.3 修改数据源

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**源数据表**页签，进入**数据源**页面。
3. 单击数据源基本信息中的**编辑信息**或数据源列表中的**修改**，如图 *8-8:* 数据源基本信息界面和图 *8-9:* 数据源列表界面所示。

图 8-8: 数据源基本信息界面

图 8-9: 数据源列表界面



4. 在弹出的窗口中，编辑需要修改的信息，编辑时，数据源类型不允许修改。
5. 单击**连接测试数据源**按钮，可验证修改内容是否正确。
6. 单击**确定**按钮，修改数据源信息成功。

8.1.3.1.4 删除数据源

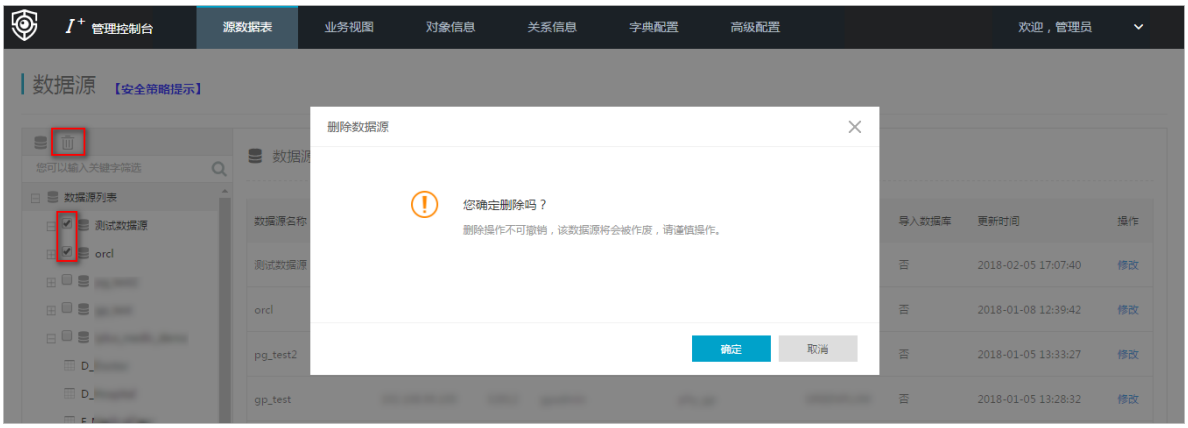
前提条件

如果待删除数据源**已添加至 OLEP** 中已经配置了数据表，则需要先将其移除，再进行删除操作，否则无法删除该数据源。

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**源数据表**页签，进入**数据源**页面。
3. 勾选一个或多个需要删除的数据源，点击**删除**图标，如图 *8-10: 删除数据源*所示。

图 8-10: 删除数据源



4. 在弹出的删除数据源窗口中，单击**确定**按钮，删除数据源。

8.1.3.2 配置OLEP数据表

配置OLEP数据表时，请先梳理好需要进行OLEP建模的数据表及每张表对应的字段，以及整理好需要配置的业务模型。如果数据表被引用，将不允许删除。

配置OLEP数据表作为对象、一度关系的配置入口。一张物理表可以映射多个对象、一度关系。

8.1.3.2.1 添加数据表

数据源添加成功后，根据业务需要，在**未添加**页签页面将表添加至OLEP中，便于后续OLEP配置，初始**已添加至OLEP** 页签页面中无任何数据。

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**源数据表**页签，进入**数据源**页面。
3. 在数据源列表中点击已建数据表，页面右侧显示数据表信息。

4. 在某个数据源信息的详细页面，点击**未添加**页签，页面会自动读取并显示数据源中未添加至OLEP的数据表，如图 *8-11: 未添加至OLEP的表*所示。

图 8-11: 未添加至OLEP的表

5. 点击待添加数据表对应操作列中的**添加至OLEP**链接。

当数据表较多时，也可通过输入**表名**、**表中文名**或**表组**信息，快速准确的定位到目标数据表，如图 *8-12: 搜索表*所示。

图 8-12: 搜索表

6. 在弹出的选择OLEP窗口中，会显示系统中已有的对象和一度关系列表，如图 *8-13: 选择OLEP窗口*所示。

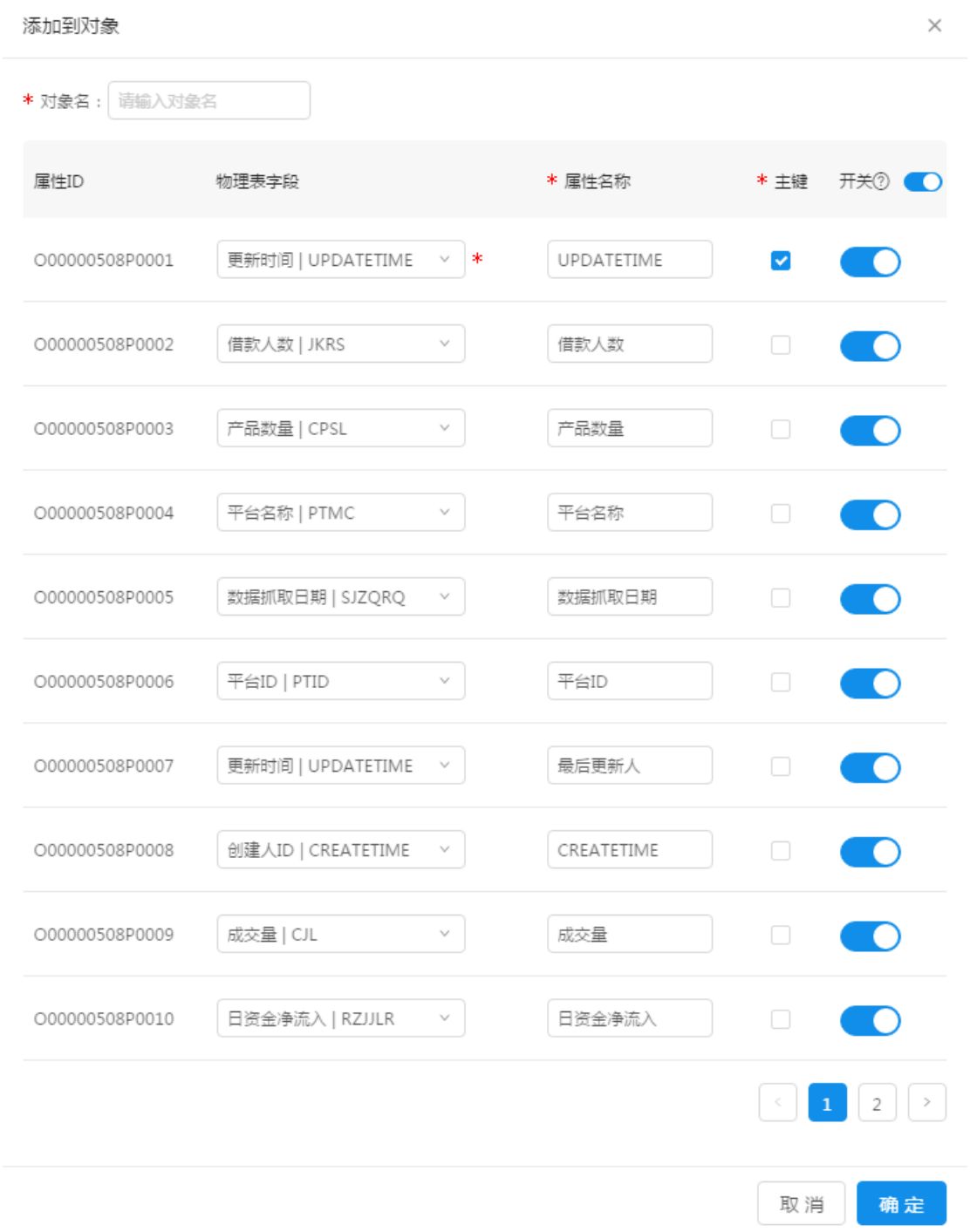
图 8-13: 选择OLEP窗口

7. 选择数据表要映射到的对象，有如下情况：

   - 映射到新增对象

     1. 选择图 *8-13: 选择OLEP窗口*中的**新增对象**后，单击**确定**按钮。
     2. 在弹出的添加到对象窗口中，如图 *8-14: 添加到对象窗口-新增对象*所示，通过开关来控制对象属性和物理表字段的映射。

图 8-14: 添加到对象窗口-新增对象



- 对象名：必填项，名称在系统中唯一。
- 主键：必须要设置主键且主键属性开关为开启状态。可以设置一个或多个属性为主键，只需要勾选属性对应的主键列的复选框即可。

3. 单击**确定**按钮，则新增对象成功，且保存数据表和该对象的映射。

若对象的主键属性只有一个，则在用户角色中配置该对象权限后，可以直接在Web端，新增该实体。

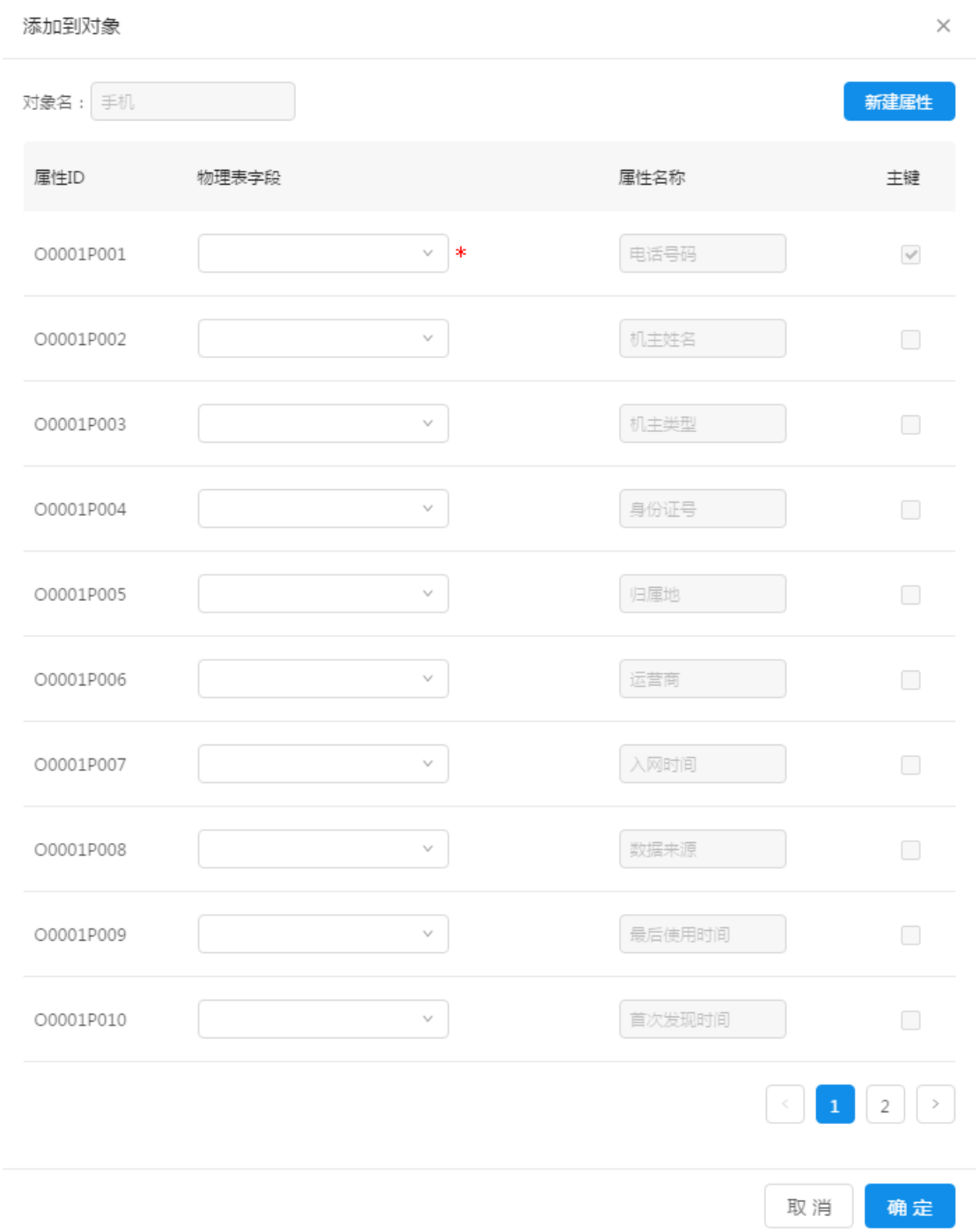
若对该对象本身或者该对象属性有业务配置需求，可在对象信息默认实体分组中，查看到该对象信息及属性，并进行业务配置。

- 映射到已有对象

  1. 选择图 *8-13:* 选择*OLEP*窗口中的某个已有对象后，单击**确定**按钮。
  2. 在弹出的添加到对象窗口中，如图 *8-15:* 添加到对象窗口-已有对象所示，通过下拉框选择物理表字段来实现对象与该物理表字段的映射。主键属性必须映射物理表字段。

     对象名和主键不可编辑。

图 8-15: 添加到对象窗口-已有对象



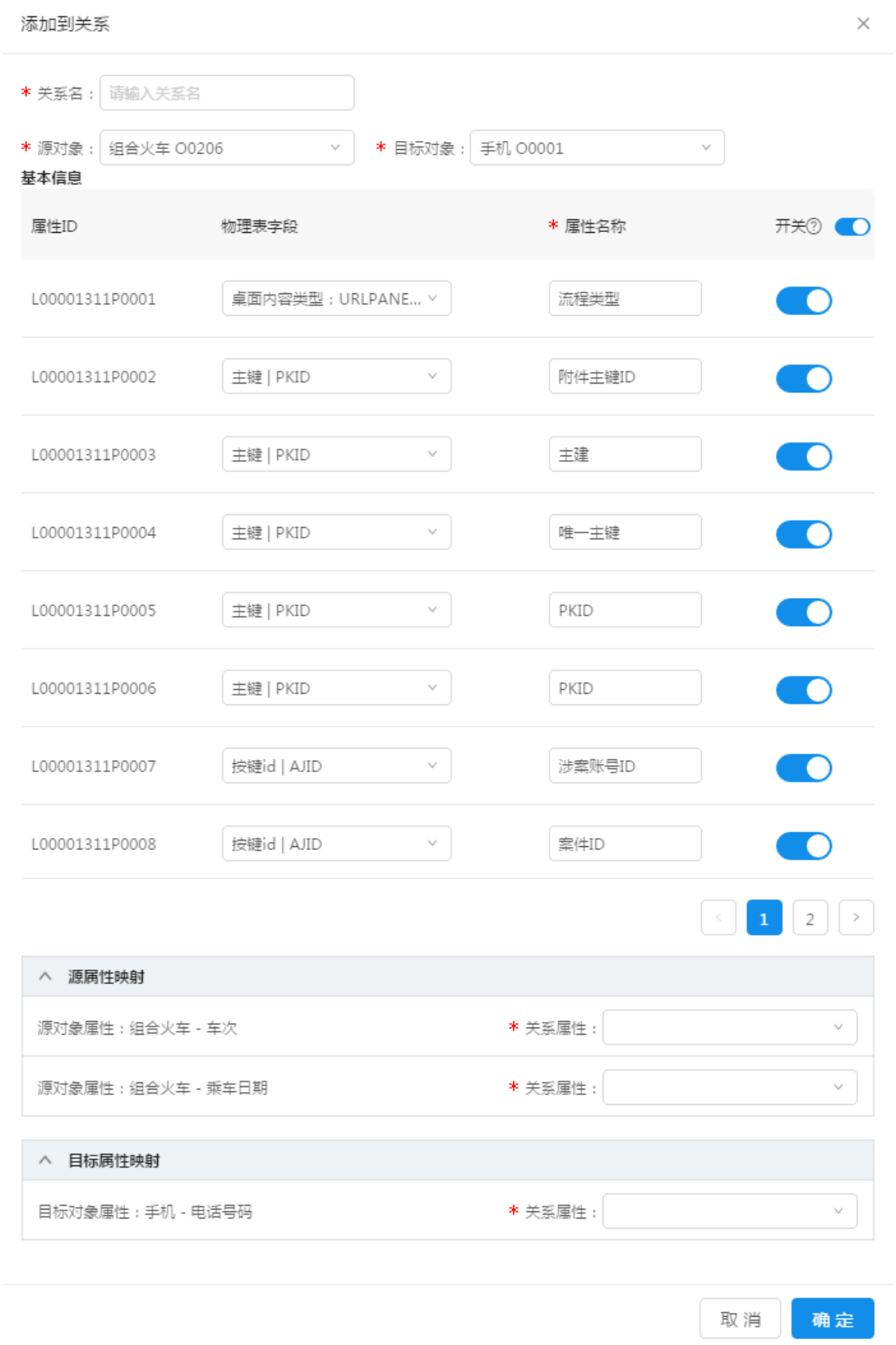
3. 若该对象的属性无法覆盖全物理字段，可通过单击**新建属性**按钮，增加映射。
4. 单击**确定**按钮，保存数据表和该对象的映射。

8. （步骤四和步骤五选其一）选择数据表要映射到的关系，有如下情况：

- 映射到新增关系

1. 选择图 *8-13:* 选择*OLEP*窗口中的**新增关系**后，单击**确定**按钮。
2. 在弹出的添加到关系窗口中，如图 *8-16:* 添加到关系窗口-新增关系所示，通过开关来控制关系属性和物理表字段的映射。

图 8-16: 添加到关系窗口-新增关系



- 关系名：必填项，名称在系统中唯一。
- 源对象/目标对象：通过下拉框选择后，页面下方会显示源属性映射或目标属性映射的配置项。

3. 单击**确定**按钮，则新增关系成功，且保存数据表和该关系的映射。

在用户角色中配置该关系权限后，可以直接在Web端源对象关联反查时可见并选择使用。

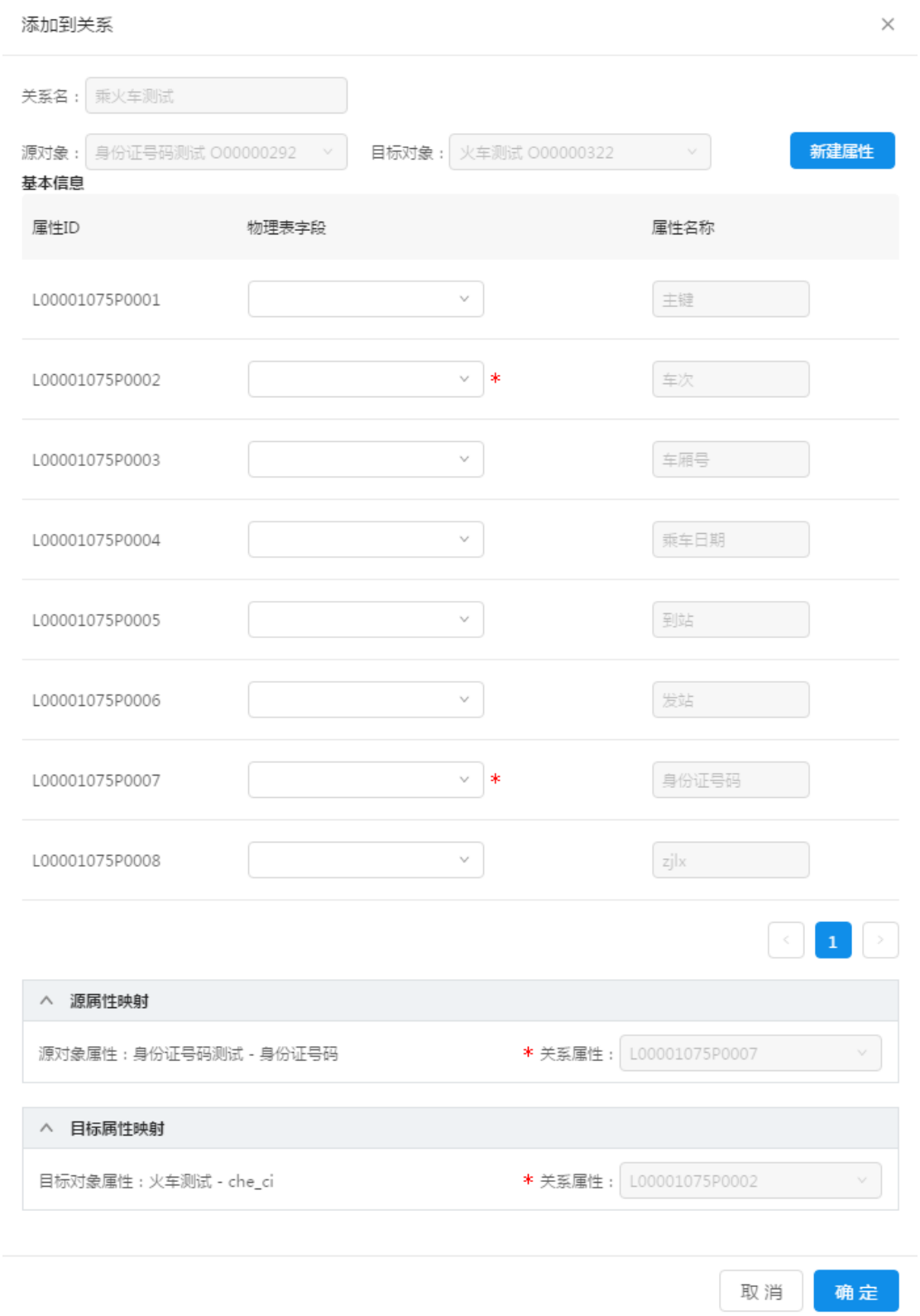
若对该关系本身或者其属性有业务配置需求，可在关系信息默认关系分组中，查看到该关系信息及属性，并进行业务配置。

- 映射到已有关系

1. 选择图 *8-13:* 选择*OLEP*窗口中的某个已有关系后，单击**确定**按钮。
2. 在弹出的添加到关系窗口中，如图 *8-17:* 添加到关系窗口-已有关系所示，通过下拉框选择物理表字段来实现关系与该物理表字段的映射。源属性和目标属性映射里的关系属性必须映射物理表字段。

   关系名、源属性和目标属性不可编辑。

图 8-17: 添加到关系窗口-已有关系



3. 若该关系的属性无法覆盖全物理字段，可通过单击**新建属性**按钮，增加映射。
4. 单击**确定**按钮，保存数据表和该关系的映射。

9. 将数据表添加至OLEP后，在**未添加** 页签页面中，已添加的表会自动从**未添加**中移除，**已添加至OLEP** 页签页面中可以查看添加的表，如图 *8-18: 查看添加结果*所示。

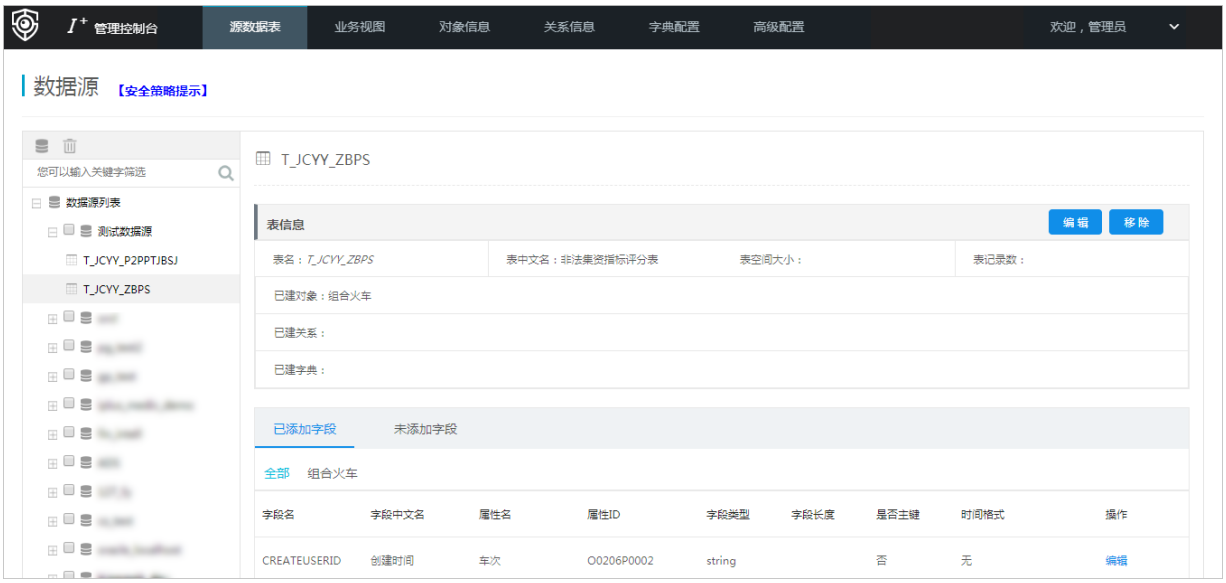
图 8-18: 查看添加结果

8.1.3.2.2 查看数据表

在左侧导航树中，点击需要查看的数据表。

右侧会显示数据表的信息，包含数据表的基本定义信息，以及该数据表后置依赖的对象、关系及字段，如图 *8-19: 查看数据表*所示。

图 8-19: 查看数据表

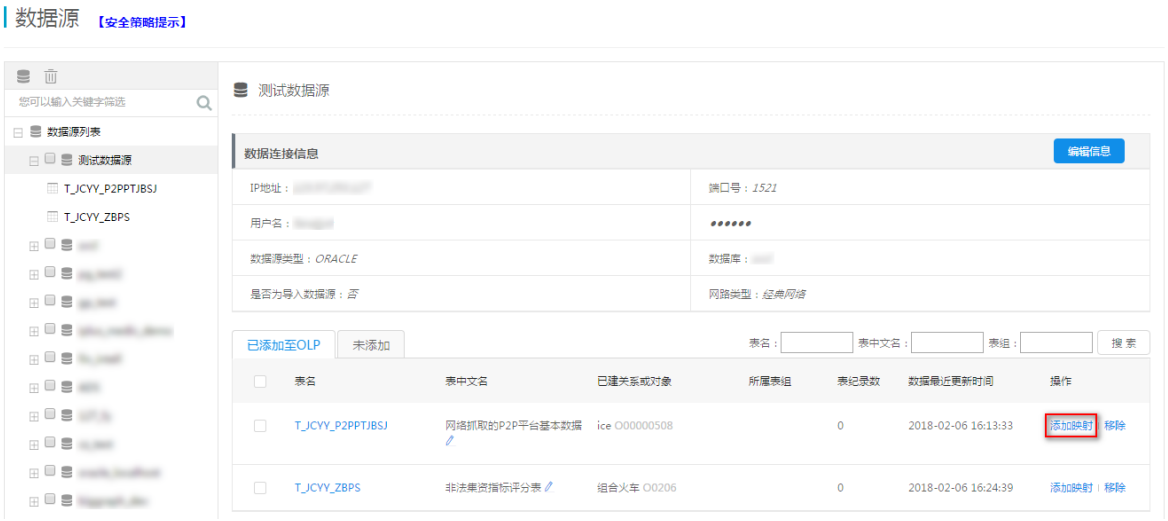


8.1.3.2.3 编辑数据表

已添加至 OLEP-添加映射

1. 登录关系网络分析管理后台。
2. 在数据源列表中点击已建数据源，页面右侧显示数据源信息。
3. 在**已添加至OLEP**页签页面，如图 *8-20: 添加映射*所示，点击数据表对应操作列的**添加映射**链接，可在弹出的选择OLEP窗口中配置该表与对象或关系的映射关系，详细请参见添加数据表。

图 8-20: 添加映射



编辑表中文名

编辑表中文名有如下方式：

- 在**已添加至 OLEP**页面编辑表中文名

  1. 在数据源列表中点击已建数据源，页面右侧显示数据源信息。
  2. 在**已添加至OLEP**页签页面，可点击数据表对应表中文名列的**编辑**图标，如图 *8-21: 已添加至OLEP页面-修改表名*所示。

     图 8-21: 已添加至OLEP页面-修改表名

  3. 修改表中文名称后，单击**确定**按钮，修改成功。

- 在数据表信息页面编辑表中文名

  1. 在数据源列表中点击已建数据表，页面右侧显示数据表信息。
  2. 单击**编辑**按钮，修改表中文名称，如图 *8-22: 数据表信息页面-修改表名*所示。

     图 8-22: 数据表信息页面-修改表名

  3. 单击**保存**按钮，修改成功。

8.1.3.2.4 移除数据表

用户可根据需要，删除未被依赖的数据表。

前提条件

如果待删除数据表已经被依赖，则需要先删除依赖的对象、关系或字典，否则系统将不允许移除该数据表。

移除数据表

1. 登录关系网络分析管理后台。
2. 单击页面上方的**源数据表**页签，进入**数据源**页面。
3. 在数据源列表中点击已建数据表，页面右侧显示数据表信息。
4. 单击**移除**按钮，如图 *8-23: 数据表信息页面*所示。

图 8-23: 数据表信息页面

5. 在弹出的确认对话框中，单击**确定**按钮，完成移除数据表操作。

如果待删除数据表已经被依赖，会提示“数据表已有逻辑映射，请先删除逻辑映射”，则需要先移除映射关系。

已添加至 OLEP-移除映射

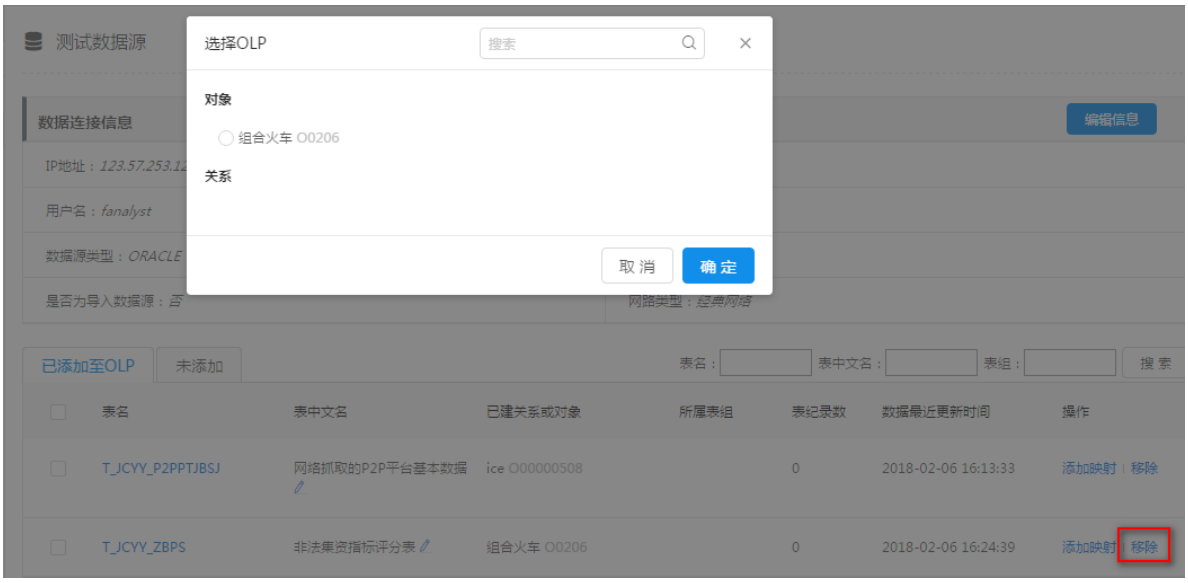
说明：

此处只移除映射，不会删除对象或关系，即对象或关系仍可通过对象信息、关系信息查询到，也可以继续在“添加至OLEP”和“添加映射”的对象关系列表中搜索到。

1. 在数据源列表中点击已建数据源，页面右侧显示数据源信息。

2. 在**已添加至OLEP**页签页面，点击数据表对应操作列的**移除**链接。
3. 在弹出的选择OLEP窗口中，如图 *8-24: 移除映射*所示，展示了当前该数据表已配置的对象或关系列表。

图 8-24: 移除映射



4. 勾选想要移除的对象或关系，单击**确定**按钮，移除该映射。

   若已添加的表，在移除最后一个对象或者关系映射后，该表也会被移除到未添加列表。

8.1.3.3 配置数据表字段

配置OLEP数据表字段时，请先梳理好需要进行OLEP建模的字段及字段的数据类型，尤其是业务含义为时间类型的字段。如果字段被引用，将不允许删除。

8.1.3.3.1 编辑数据表字段

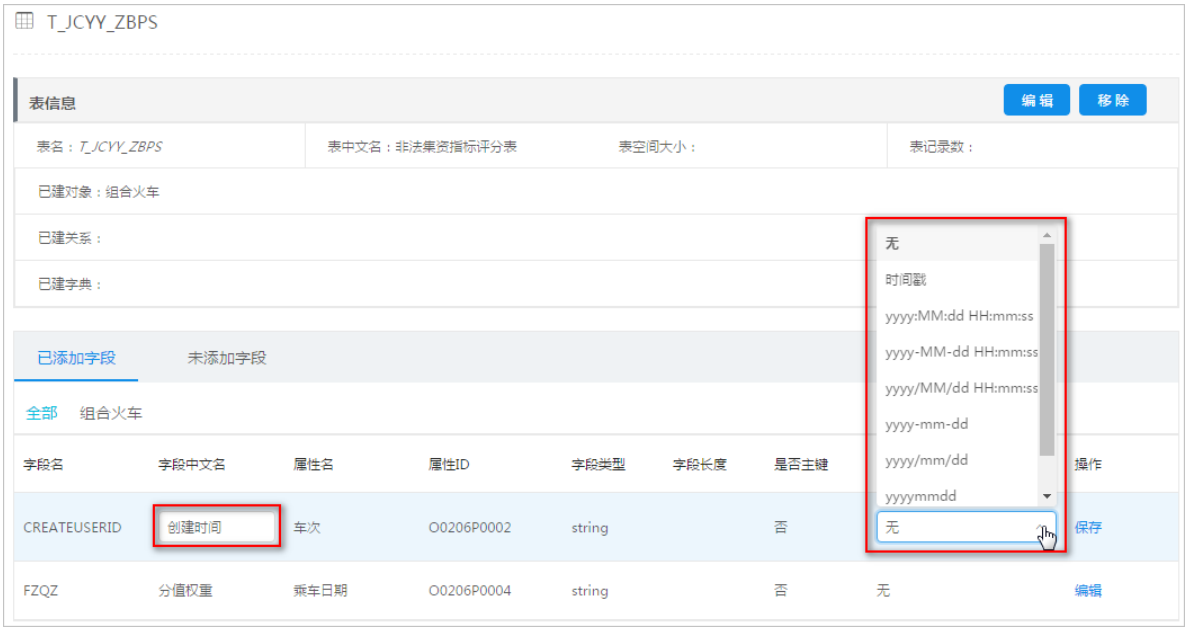
操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**源数据表**页签，进入**数据源**页面。
3. 在数据源列表中点击已建数据表，页面右侧显示数据表信息。
4. 点击**已添加字段** 页签页面，系统默认在**全部**页签下显示所有已添加字段，如图 *8-25: 已添加字段页签页面*所示。

图 8-25: 已添加字段页签页面

5. 点击待编辑字段对应操作列的**编辑**链接，可以编辑**字段中文名**和**时间格式**，如图 *8-26: 编辑字段*所示。

图 8-26: 编辑字段



如果字段为时间字段，请选择对应的时间格式，目前产品支持八种格式：timestamp10位时间戳、YYYY:MM:DD HH:MM:SS、YYYY-MM-DD HH:MM:SS、YYYY/MM/DD HH:MM:SS、YYYY-MM-DD、YYYY/MM/DD、YYYYMMDD、YYYYMMDDHHMISS。

6. 修改后，单击**保存**按钮。

8.1.3.3.2 移除数据表字段

用户可根据需要，删除未被对象引用的数据表字段。

前提条件

移除数据表字段前，请确认其未被对象引用，若被引用，请在**对象信息**页面，在引用了该字段的所有对象的属性页签下，将该数据表字段删除。

操作步骤

1. 登录关系网络分析管理后台。
2. 在数据源列表中单击已建数据表，页面右侧显示数据表信息。
3. 单击**已添加字段** 页签页面，系统默认在**全部**页签下显示所有已添加字段。
4. 单击选择该表已配置的某一对象或关系的页签页面，与该对象或关系映射的字段会高亮显示，如图 *8-27: 移除数据表字段*所示。

图 8-27: 移除数据表字段

5. 单击字段对应操作列的**移除**链接，移除该字段与对象或关系的映射。

若该字段没有跟任何已配置的对象或关系属性映射，则会显示在**未添加字段**页签页面。

8.1.3.3.3 添加数据表字段

没有属性映射的字段，如果需要添加，可以在**未添加字段** 页签页面中进行添加。

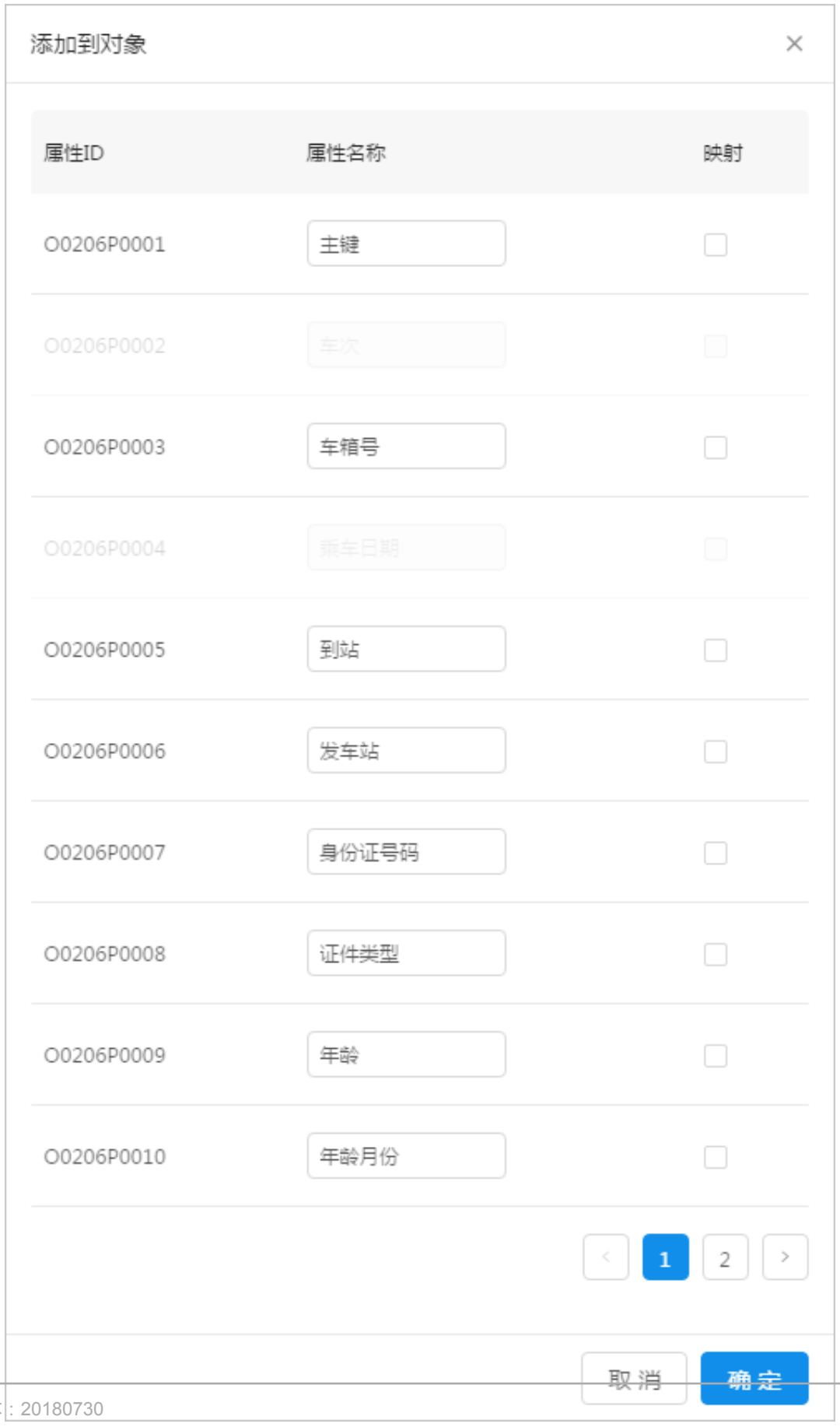
操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**源数据表**页签，进入**数据源**页面。
3. 在数据源列表中点击已建数据表，页面右侧显示数据表信息。
4. 点击**未添加字段** 页签页面，显示所有未添加字段。
5. 点击待添加字段对应操作列的**添加**链接，弹出的选择OLEP窗口中显示了该表已配置的对象和关系，如图 *8-28: 选择OLEP窗口*所示。

图 8-28: 选择OLEP窗口

6. 选择某一对象或关系，单击**确定**按钮。
7. 在弹出的窗口中，已与该表的其他字段有映射的属性置灰。勾选映射列的复选框，如图 *8-29: 添加映射*所示。

图 8-29: 添加映射


添加到对象
属性ID
属性名称
映射
O0206P0001
主键
O0206P0002
车次
O0206P0003
车箱号
O0206P0004
乘车日期
O0206P0005
到站
O0206P0006
发车站
O0206P0007
身份证号码
O0206P0008
证件类型
O0206P0009
年龄
O0206P0010
年龄月份
1
2
取 消
确 定

8. 单击**确定**按钮，完成添加该数据表字段与该对象或关系的属性映射。

8.1.4 配置字典

前提条件

在进行字典配置前，请先梳理系统数据中涉及需要进行转换的字典，如果字典被引用，将不允许删除。

8.1.4.1 新建字典

前提条件

新建字典前，请确认已进行如下操作：

- 已新建数据源，请参见新建数据源。
- 已添加数据表，请参见添加数据表。
- 数据表中已添加数据表字段，请参见添加数据表字段。

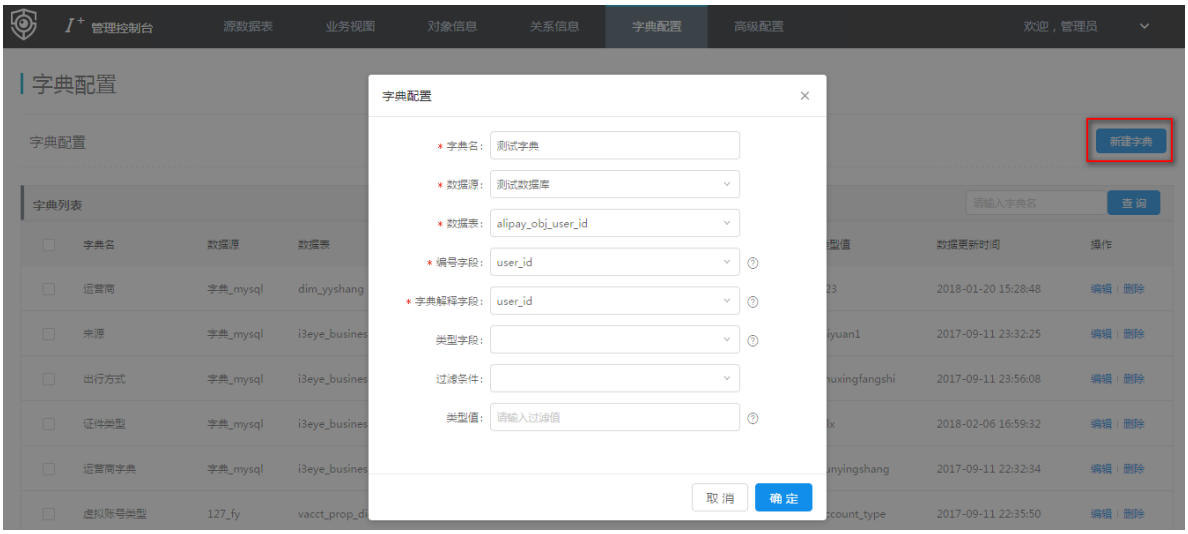
操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**字典配置**页签，进入字典配置页面，如图 *8-30: 字典配置页面*所示。

图 8-30: 字典配置页面

3. 单击页面右上角**新建字典**按钮，弹出字典配置窗口，如图 *8-31: 字典配置窗口*所示。

图 8-31: 字典配置窗口



4. 编辑字典信息，具体如下：

   - 字典名：配置的字典的命名，自定义编写。
   - 数据源：所要引用的数据源。
   - 数据表：所要引用的数据源下的数据表。
   - 编号字段：所选择的数据表中对应的字典编码字段。
   - 字典解释字段：字典转换后对应的名称字段。
   - 类型字段、过滤条件、类型值：非必填项，根据需要用于进行字典表的条件过滤。如果三个选项中填写了一项，则其余两项为必填。

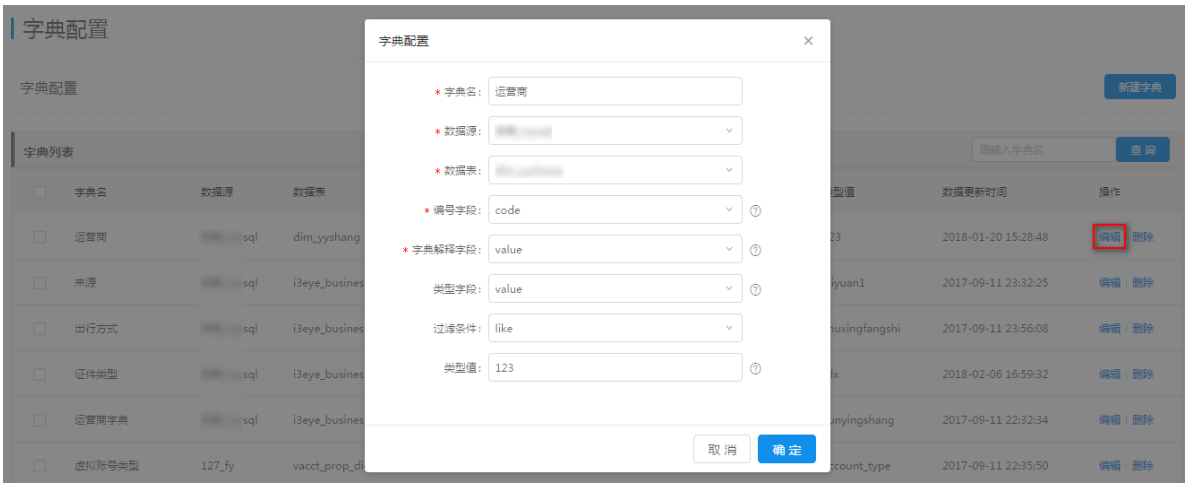
5. 单击**确定**按钮，完成新建字典操作。

8.1.4.2 修改字典

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**字典配置**页签，进入字典配置页面，字典列表显示了系统中的字典信息。
3. 点击需要编辑的字典对应操作列的**编辑**链接，如图 *8-32: 修改字典*所示。

图 8-32: 修改字典



4. 在弹出的字典配置窗口中，编辑需要修改的信息项，具体参数介绍请参见字典参数说明。
5. 单击**确定**按钮，完成修改字典信息操作。

8.1.4.3 删除字典

删除字典时，如果字典被引用，则不允许删除。

删除单个字典

1. 登录关系网络分析管理后台。
2. 单击页面上方的**字典配置**页签，进入字典配置页面，字典列表显示了系统中的字典信息。
3. 点击需要删除字典对应操作列的**删除**链接，弹出确认对话框，如图 *8-33:* 删除单个字典所示。

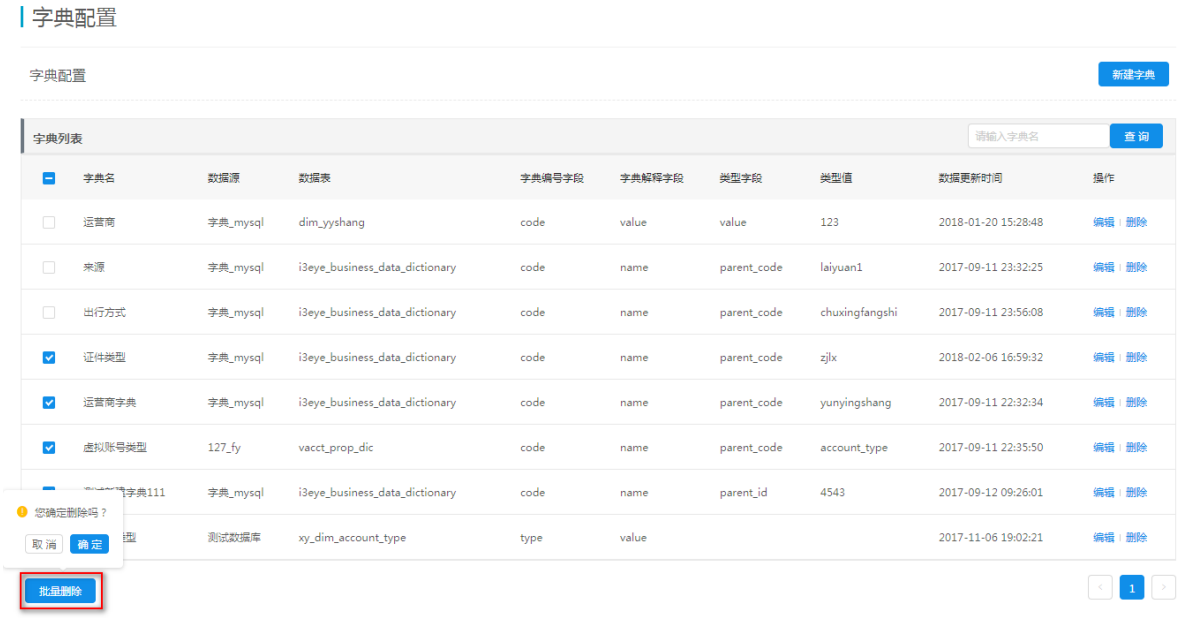
图 8-33: 删除单个字典

4. 单击**确定**按钮，完成删除单个字典操作。

批量删除字典

1. 登录关系网络分析管理后台。
2. 单击页面上方的**字典配置**页签，进入字典配置页面，字典列表显示了系统中的字典信息。
3. 勾选一个或多个字典名前方的复选框，单击页面左下角的**批量删除**按钮，弹出确认对话框，如图 *8-34: 批量删除字典*所示。

图 8-34: 批量删除字典



4. 单击**确定**按钮，完成批量删除字典操作。

8.1.5 配置对象信息

新建的对象，用户需要重新登录系统后，才可使用。

8.1.5.1 管理对象分组

8.1.5.1.1 新建分组

1. 登录关系网络分析管理后台。
2. 单击页面上方的**对象信息**页签，进入对象信息页面，如图 *8-35: 新建分组*所示。

图 8-35: 新建分组

3. 点击**新建分组**图标，弹出新建分组窗口。
4. 输入分组名称，选择该分组的上级分组，如图 *8-36: 新建分组窗口*所示。

图 8-36: 新建分组窗口

5. 单击**确定**按钮，完成新建对象分组操作。

8.1.5.1.2 查看分组

1. 登录关系网络分析管理后台。
2. 单击页面上方的**对象信息**页签，进入对象信息页面，左侧导航树显示了系统中所有的对象分组。
3. 点击某一分组名称，页面右侧会显示该分组的详细信息，如图 *8-37: 查看分组信息*所示。

图 8-37: 查看分组信息

8.1.5.1.3 修改分组

修改分组基本信息

- 方法一：

  1. 登录关系网络分析管理后台。
  2. 单击页面上方的**对象信息**页签，进入对象信息页面，左导航树显示了系统中所有的对象分组信息。
  3. 点击需要修改分组对应操作列的**修改**链接，如图 *8-38: 对象分组信息列表*所示，可修改对象分组的名称和排序号。

     图 8-38: 对象分组信息列表

  4. 单击**确定**按钮。

- 方法二：

  1. 登录关系网络分析管理后台。
  2. 单击页面上方的**对象信息**页签，进入对象信息页面，左导航树显示了系统中所有的对象分组。

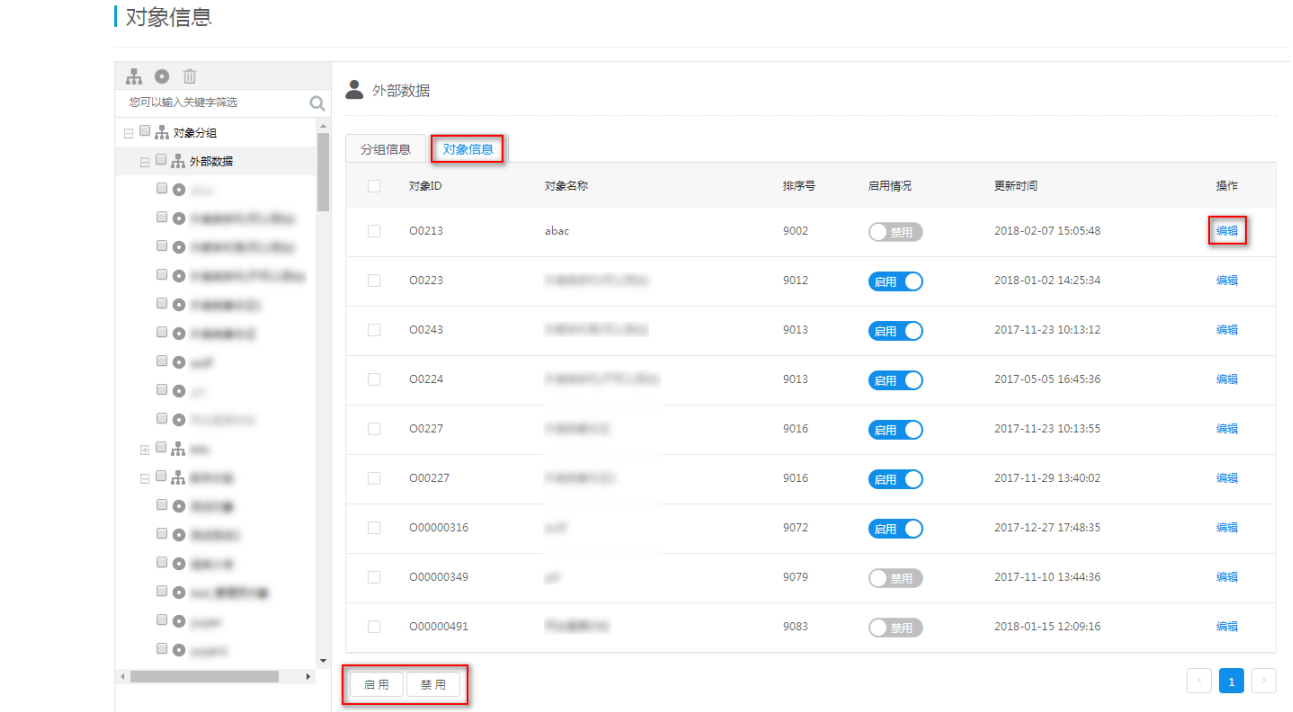
3. 点击某一分组名称，页面右侧显示该分组的详细信息。
4. 单击**编辑基本信息**，可修改对象分组的名称和排序号，如图 *8-39: 基本信息界面*所示。

图 8-39: 基本信息界面

修改分组中的对象信息

1. 登录关系网络分析管理后台。
2. 单击页面上方的**对象信息**页签，进入对象信息页面，左导航树显示了系统中所有的对象分组。
3. 点击某一分组名称，页面右侧显示该分组的详细信息。
4. 点击**对象信息**页签，显示该分组下的对象信息。
5. 点击需要修改对象对应操作列的**编辑**链接，弹出编辑信息窗口，如图 *8-40: 对象信息页面*所示。

图 8-40: 对象信息页面



支持修改信息如下：

- 可修改对象名称和排序号。
- 可批量启用或禁用对象。

8.1.5.1.4 删除分组

前提条件

您仅可删除空的分组，如果分组下存在对象，则不允许删除。

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**对象信息**页签，进入对象信息页面，左导航树显示了系统中所有的对象分组。
3. 勾选一个或多个待删除的对象分组，点击页面左上角的**删除**图标，如图 *8-41: 删除对象分组*所示。

图 8-41: 删除对象分组

4. 在弹出的删除对象信息窗口中，单击**确定**按钮，完成删除对象分组操作。

8.1.5.2 管理对象基本信息

8.1.5.2.1 新建对象

前提条件

新建对象前，请确认已新建对象分组，请参见新建分组。

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**对象信息**页签，进入对象信息页面，如图 *8-42: 对象信息页面*所示。

图 8-42: 对象信息页面

3. 点击**新建对象**图标，弹出新建对象窗口。
4. 配置对象信息，如图 *8-43: 配置对象信息*所示。

图 8-43: 配置对象信息

参数说明如下：

- 对象名称：必填项，输入对象的自定义名称，不可与已有对象重复。
- 对象描述：可选。
- 所属分组：选择该对象所属的分组。
- 是否允许添加：表示是否允许在界面上手动添加该节点。
- 对象图标展示位置：可以选择四种类型：
  - 图区使用图标，右侧面板不展示
  - 图区使用图标，右侧面板展示头像

- 图区使用头像，右侧面板不展示
- 图区使用头像，右侧面板展示头像

- 是否有对象表：默认选择**是**。
- 对象icon：为对象的图标，可以从图标库中勾选，也可以填写可访问的URL引用外部图标。

5. 单击**确定**按钮，完成新建对象操作。

8.1.5.2.2 修改对象

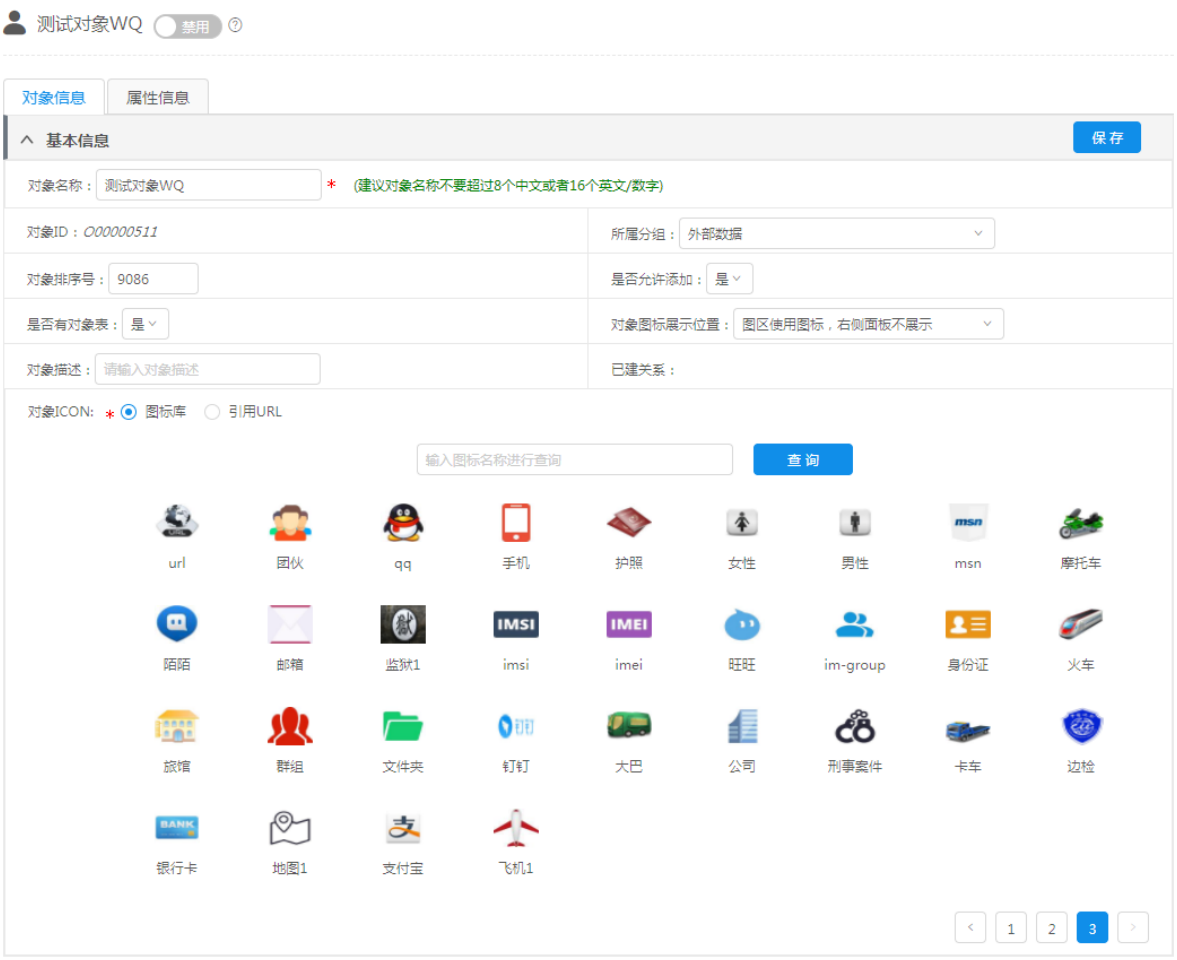
操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**对象信息**页签，进入对象信息页面，左侧导航树显示了系统中所有对象分组及其下属的对象。
3. 点击某一对象名称，页面右侧会显示该对象的详细信息。
4. 单击**编辑基本信息**按钮，如图 *8-44: 基本信息界面*所示。

图 8-44: 基本信息界面

5. 修改对象信息，如图 *8-45: 修改对象信息*所示，除**对象 ID** 不可修改外，其余参数均可修改，有关参数的详细说明请参见对象参数说明。

图 8-45: 修改对象信息



6. 单击**保存**按钮，完成修改对象信息操作。

8.1.5.2.3 删除对象

前提条件

您仅可删除未与任何关系绑定的对象，如果一个对象被某个关系所依赖，则不允许删除。

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**对象信息**页签，进入对象信息页面，左侧导航树显示了系统中所有对象分组及其下属的对象。
3. 勾选一个或多个待删除的对象，点击页面左上角的**删除**图标，如图 *8-46: 删除对象信息*所示。

图 8-46: 删除对象信息

4. 在弹出的删除对象信息窗口中，单击**确定**按钮，完成删除对象操作。

8.1.5.2.4 启用/禁用对象

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**对象信息**页签，进入对象信息页面，左侧导航树显示了系统中所有对象分组及其下属的对象。
3. 点击某一对象名称，页面右侧会显示该对象的详细信息。
4. 点击**启用/禁用**图标，可切换对象的使用状态。从启用状态切换到禁用状态，会屏蔽该对象，屏蔽后该对象不可用，如图 *8-47: 禁用对象*所示。

图 8-47: 禁用对象

5. 禁用对象后，可再次单击**启用/禁用**按钮，从禁用状态切换到启用状态，启用对象。

8.1.5.3 管理对象属性信息

8.1.5.3.1 增加一行-配置逻辑属性

前提条件

增加一行属性信息前，请确认已进行如下操作：

- 已新建数据源，请参见新建数据源。
- 已添加数据表，请参见添加数据表。

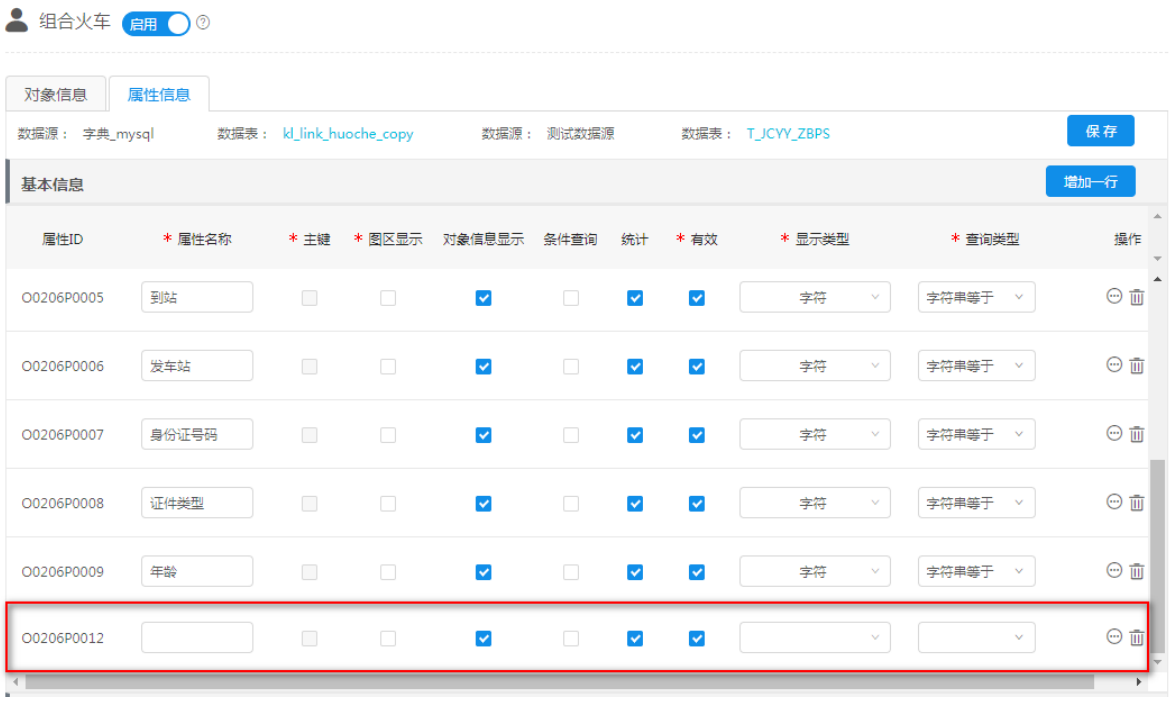
操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**对象信息**页签，进入对象信息页面，左侧导航树显示了系统中所有对象分组及其下属的对象。
3. 点击某一对象名称，页面右侧会显示该对象的详细信息。
4. 点击切换至**属性信息**页签页面，若该对象在源数据处已配置，则会显示已配置的数据源和数据表的信息，如图 *8-48: 属性信息页面*所示。点击数据表，可跳转到源数据表对应的物理表信息。

图 8-48: 属性信息页面

5. 单击**增加一行**按钮，如图 *8-49: 增加一行*所示，可为对象配置纯逻辑属性。

图 8-49: 增加一行



对象属性字段说明如下：

- 属性ID：由系统自动生成。
- 属性名称：展示在分析工作台的显示属性名，请填写可体现业务的名称。
- 主键：勾选，在添加节点时需要输入的信息。例如，身份证号码为主键，则在前端Web页面添加节点时，需要输入身份证号码。主键勾选保存后不可修改、不可删除。
- 图区显示：若勾选该项，则在前端Web页面画布中添加节点对象后，该对象显示的内容。例如，添加人的属性，身份证号码和姓名两个属性，如果只勾选身份证号码，则前端Web页面显示身份证号码，如果只勾选姓名，则前端Web页面显示姓名，如果二者同时勾选，则显示姓名+身份证号码。
- 对象信息显示：若勾选该项，则在前端Web页面右侧面板“对象信息”中显示。
- 条件查询：若勾选该项，将在前端Web页面“关系分析”时“目标对象”的查询条件中呈现。请根据实际情况输入筛选条件。
- 统计：若勾选该项，则会在前端Web页面右侧的“统计信息”中展现。
- 有效：被勾选，则表明该条记录生效，会在前端Web页面显示，若属性后续有依赖，则不允许置为无效。
- 显示类型：根据字段实际情况进行选择，设置后在右侧面板的对象信息中将按照选择的类型展示。

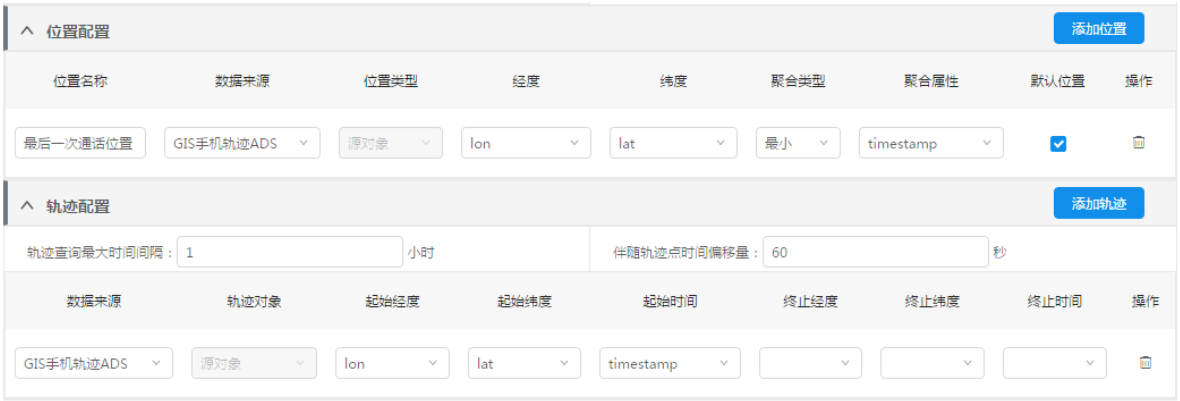
- 查询类型：在“条件查询”勾选后配置生效，类型根据实际情况选择。
- 字典：与查询类型有关，当查询类型下拉框中勾选了字典后，需要配置字典选项。
- 搜索项配置：展示的是“搜索配置”的搜索项，此处做搜索名与属性值关联。
- 默认查询条件设置：缺省值，根据需要进行默认查询条件的设置，包含查询规则和参数值。
- “高级设置”的图标展示字段，选择需要在图区展示头像的字段，并设置访问的URL，并根据实际情况设置URL访问的图片后缀名。

6. 单击**保存**按钮，完成配置逻辑属性操作。

8.1.5.3.2 配置GIS

1. 登录关系网络分析管理后台。
2. 单击页面上方的**对象信息**页签，进入对象信息页面，左侧导航树显示了系统中所有对象分组及其下属的对象。
3. 点击某一GIS对象名称，页面右侧会显示该对象的详细信息。
4. 点击切换至**属性信息**页签页面，在位置配置和轨迹配置区域配置对象信息与GIS物理表的映射（数据需要经度、纬度信息，最好有geohash）。
5. 如图 *8-50: 位置配置和轨迹配置*所示，进行位置配置和轨迹配置。

图 8-50: 位置配置和轨迹配置



8.1.6 配置关系信息

新建的对象关系，用户需要重新登录系统后，才可使用。

8.1.6.1 管理关系分组

8.1.6.1.1 新建分组

1. 登录关系网络分析管理后台。

2. 单击页面上方的**关系信息**页签，进入关系信息页面，如图 *8-51: 关系信息页面*所示。

图 8-51: 关系信息页面

3. 点击**新建分组**图标，弹出新建分组窗口。
4. 输入分组名称，选择该分组的上级分组，如图 *8-52: 新建分组窗口*所示。

图 8-52: 新建分组窗口

5. 单击**确定**按钮，完成新建关系分组操作。

8.1.6.1.2 查看分组

1. 登录关系网络分析管理后台。
2. 单击页面上方的**关系信息**页签，进入关系信息页面，左侧导航树显示了系统中所有的关系分组。
3. 点击某一分组名称，页面右侧会显示该分组的详细信息，如图 *8-53: 查看关系分组信息*所示。

图 8-53: 查看关系分组信息

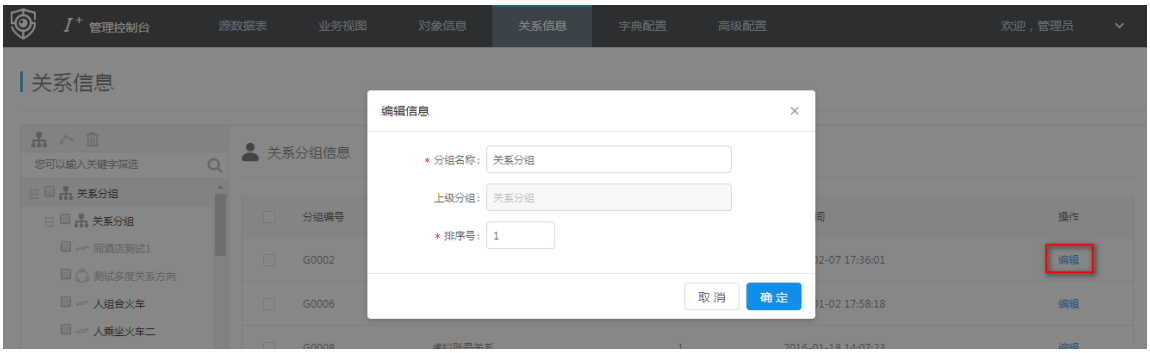
8.1.6.1.3 修改分组

修改分组基本信息

- 方法一：

  1. 登录关系网络分析管理后台。
  2. 单击页面上方的**关系信息**页签，进入关系信息页面，左侧导航树显示了系统中所有的关系分组信息。
  3. 单击需要修改分组对应**操作**列的**编辑**，如图 *8-54:* 关系分组信息列表所示，可修改关系分组的名称和排序号。

     图 8-54: 关系分组信息列表

  4. 单击**确定**按钮。

- 方法二：

  1. 登录关系网络分析管理后台。
  2. 单击页面上方的**关系信息**页签，进入关系信息页面，左导航树显示了系统中所有的关系分组。

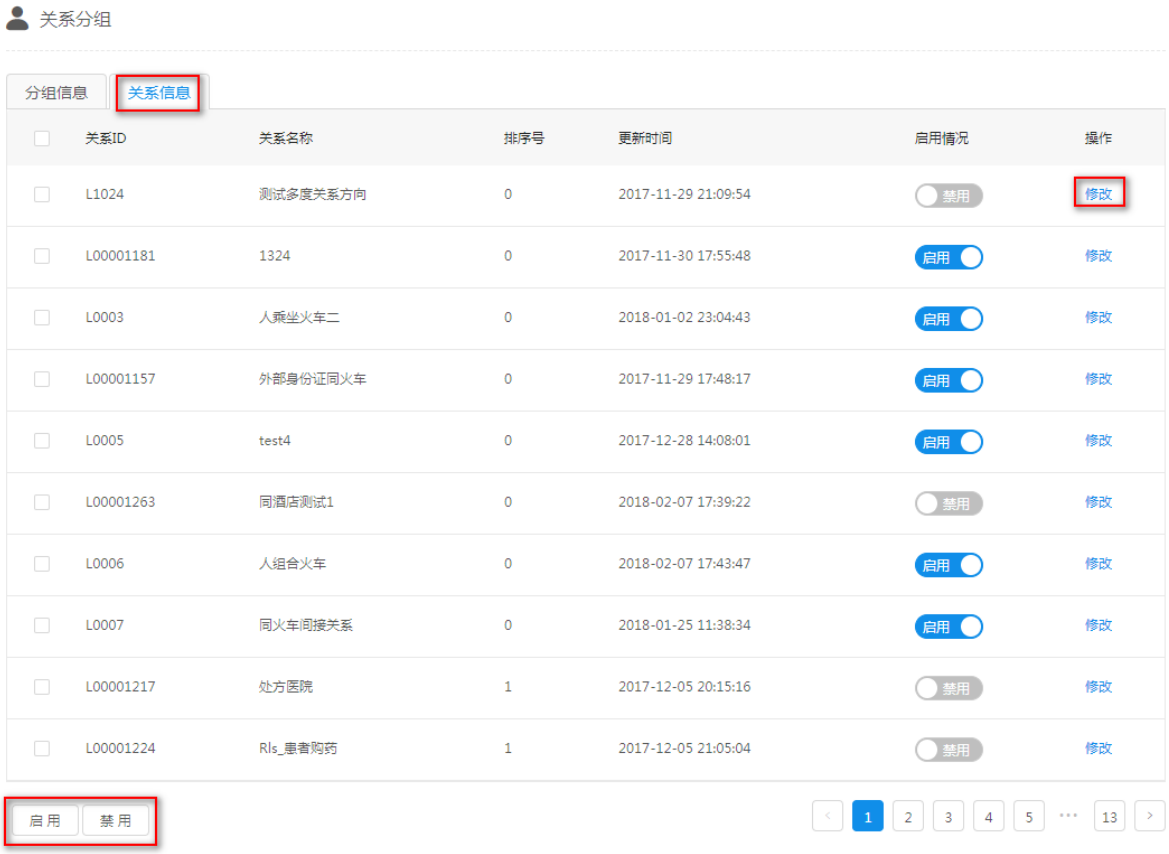
3. 单击某一分组名称，页面右侧显示该分组的详细信息。
4. 单击**编辑基本信息**按钮，可修改关系分组的名称和排序号，如图 *8-55: 关系分组页面*所示。

图 8-55: 关系分组页面

修改分组中的关系信息

1. 单击页面上方的**关系信息**页签，进入关系信息页面，左导航树显示了系统中所有的关系分组。
2. 单击某一分组名称，页面右侧显示该分组的详细信息。
3. 单击**关系信息**页签，显示该分组下的关系信息。
4. 单击需要修改关系对应操作列的**修改**链接，弹出编辑信息窗口，如图 *8-56: 关系信息页面*所示。

图 8-56: 关系信息页面



支持修改信息如下：

- 可修改关系名称和排序号。
- 可批量启用或禁用关系。

8.1.6.1.4 删除分组

前提条件

您仅可删除空的分组，如果分组下存在关系，则不允许删除。

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**关系信息**页签，进入关系信息页面，左导航树显示了系统中所有的关系分组。
3. 勾选一个或多个待删除的关系分组，点击页面左上角的**删除**图标，如图 *8-57: 删除关系分组*所示。

图 8-57: 删除关系分组

4. 在弹出的删除关系信息窗口中，单击**确定**按钮，完成删除关系分组操作。

8.1.6.2 配置一度关系

8.1.6.2.1 新建关系基本信息

前提条件

新建关系前，请确保系统中已存在新建的对象且该对象被启用。新建对象的方法请参见新建对象。

操作步骤

1. 登录关系网络分析管理后台。
2. 点击页面上方的**关系信息**页签，进入关系信息页面。
3. 点击**新建关系**图标，在弹出的下拉菜单中选择**一度关系**项，如图 *8-58: 新建关系*所示，弹出新建关系窗口。

图 8-58: 新建关系

4. 配置关系信息，如图 *8-59: 配置关系信息*所示。

图 8-59: 配置关系信息

参数说明如下：

- 关系名称：必填，输入关系的自定义名称，不可与已有关系重复。
- 所属分组：默认为左侧当前选中分组，根据需要可重新选择。
- 是否显示：用于表示该关系是否作为在关系筛选时候显示。
- 方向：表示该关系为有向关系还是无向关系。
- 源对象：在下拉列表中选择已启用的对象，作为该关系的源对象。
- 目标对象：在下拉列表中选择已启用的对象，作为该关系的目标对象。
- 关系描述：选填项，输入该关系的描述信息。

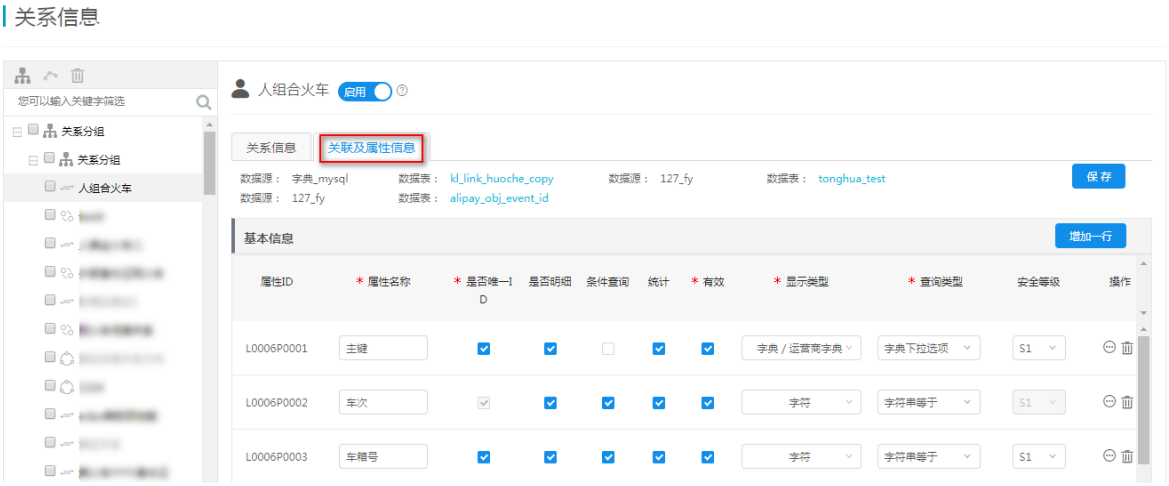
5. 单击**确定**按钮，完成新建关系操作。

8.1.6.2.2 配置关系属性

操作步骤

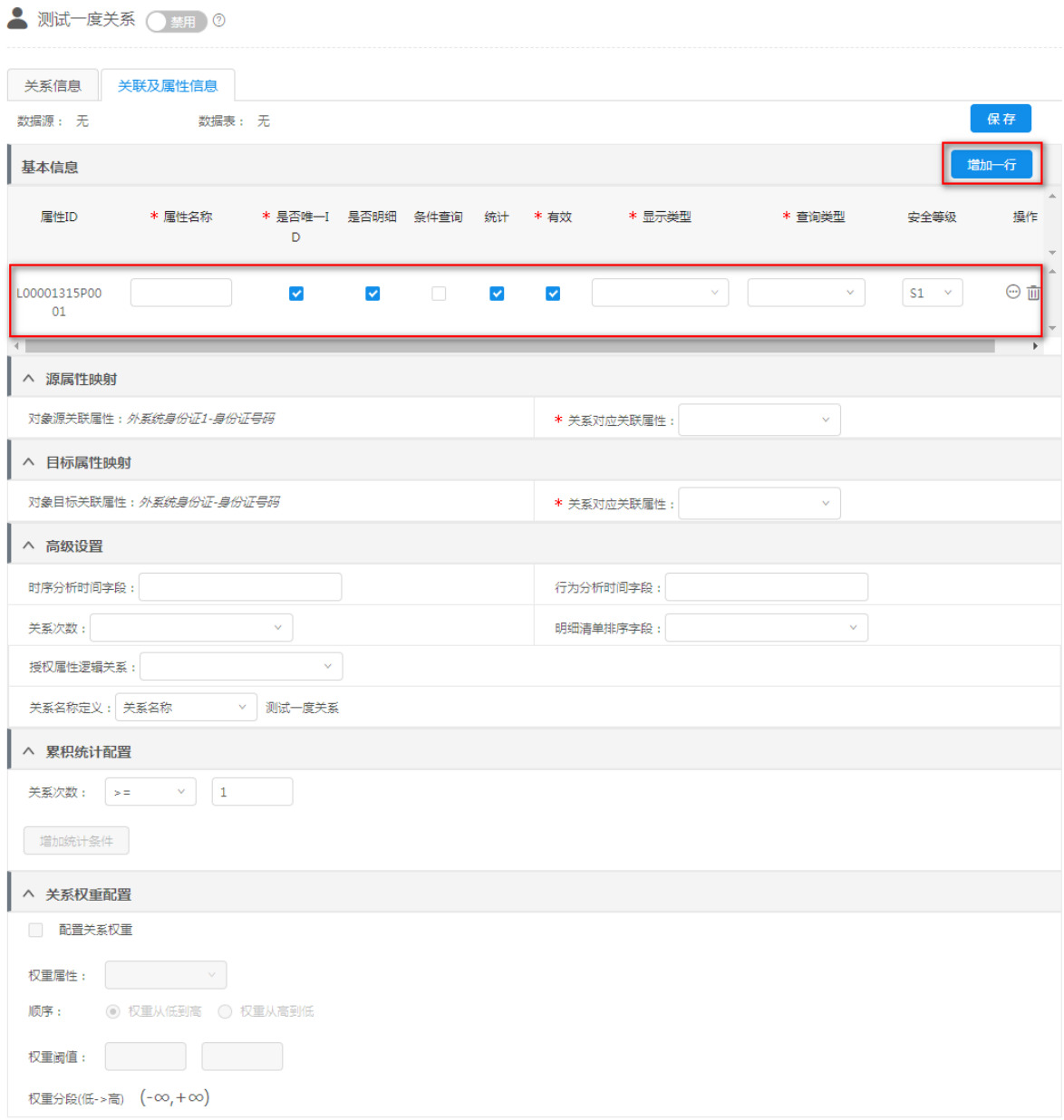
1. 登录关系网络分析管理后台。
2. 单击页面上方的**关系信息**页签，进入关系信息页面，左导航树显示了系统中的关系分组及其下属的关系。
3. 单击某个关系名称，页面右侧会显示该关系的详细信息。
4. 单击切换至**关联及属性信息**页签页面，如图 *8-60: 关联及属性信息页*所示。若已在源数据表处配置关系属性，则会展示该关系已配置的所有数据源和数据表的信息，单击数据表，可跳转到源数据表对应的物理表信息。

图 8-60: 关联及属性信息页



5. 单击**增加一行**按钮，如图 *8-61:* 配置相关信息所示，可为配置关系属性及源、目标属性映射。

图 8-61: 配置相关信息



因为源属性映射属性和目标属性映射属性不能相同，且两处配置均为必填项，所以通过**增加一行**至少要配置两个纯逻辑的关系属性。

关系属性字段说明如下：

- 属性ID：由系统自动生成。
- 属性名称：在前端Web页面**明细清单**中呈现的属性。配合**有效**属性，有效属性打勾，则该属性名称在前端Web页面呈现。
- 是否唯一ID：用于定义关系表的逻辑主键。

- 是否明细：若勾选该项，则该项内容在前端Web页面**明细清单**中呈现。
- 条件查询：若勾选该项，则在前端Web页面高级查询时，在**关系类型**页面中，该项数据可进行条件查询。
- 字典：对于查询类型或显示类型勾选了字典下拉选项的属性，需要配置字典。
- 统计：若勾选该项，则会在前端Web页面右侧的**统计信息**中展现。
- 有效：若勾选该项，则表明该条数据在前端可呈现。
- 显示类型：用于定义属性显示的格式。
- 查询类型：用于定义作为查询条件时的查询项格式。
- 默认查询：用于定义关系查询时的默认条件，该设置对全局关系筛选生效。
- 上下箭头：调整各属性在前端页面的排列顺序。
- 源属性映射、目标属性映射：与新建直接关系的**关系信息**中的**源对象**和**目标对象**有关，对象源关联属性 ID 和对象目标关联属性 ID 对应**关联及属性信息**表中的源关系对应关联属性 ID 和目标关系对应关联属性 ID 保持一致，也即对象中的属性 ID 和关系中的属性 ID 中字段值要一致，关系关联对应属性中选择与实际关系的源对象、目标对象匹配的属性。
- 时序时间字段：定义该关系中进行时序分析的字段。在下拉框中选择关系属性中查询类型为时间的属性，且下拉框可多选。
- 行为时间字段：定义该关系中需要进行行为分析的字段。在下拉框中选择关系属性中查询类型为时间的属性，且下拉框可多选。
- 关系次数：用于汇总关系中将对应的属性值直接作为关系的汇总次数进行展示定义。关系次数为关系类型过滤时的默认关系配置。
- 明细清单排序字段：返回行为明细的默认排序字段。
- 累积统计配置：是对数字区间类型的属性，进行一些Top，+，-等逻辑统计计算。该配置主要适用于需要对关系查询结果做统计过滤的业务场景。其中，**关系次数**的配置，主要是对该关系查询的明细记录条数做过滤；用户可通过增加统计条件，对关系中查询类型为时间区间的属性，进行统计过滤配置。
- 关系权重配置：把关系中某一数字区间类型的属性，按照区间范围值作为关系边权重值计算的配置。

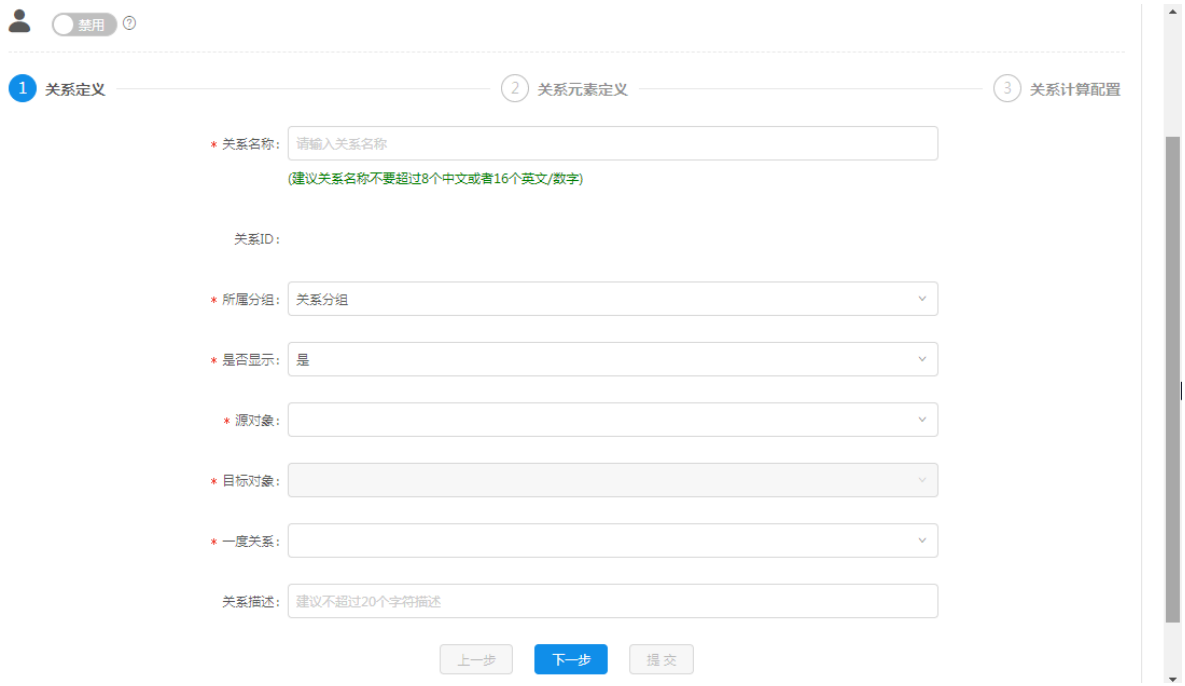
6. 根据业务需求，重复以上步骤，配置其他**直接关系**。

8.1.6.3 配置二度关系

操作步骤

1. 登录关系网络分析管理后台。
2. 点击页面上方的**关系信息**页签，进入关系信息页面。
3. 点击**新建关系**图标，在弹出的下拉菜单中选择**二度关系**项，进入关系定义页面。
4. 根据业务需求配置，进行关系定义，如图 *8-62: 关系定义*所示。

图 8-62: 关系定义



相关配置信息说明如下：

- 关系名称：需要配置的间接关系的名称，即两个不同的对象A和对象B通过第三个对象C产生间接关系，最终查询对象A和对象B有间接关系。可拆分成两个直接对像，对象A作用对象C，对象B作用对象C。例如同乘火车、同乘飞机、同住酒店。
- 所属分组：选择预先创建的二度关系分组。
- 是否显示：选择**是**即可。
- 源对象、目标对象：为前面**关系名称**中解释的对象A和对象B，例如同乘火车，乘火车的对象为人，即通过对象A作用C，关联出对象B，则源对象和目标对象都选择在**对象信息**中定义的对象，如人。
- 一度关系：即为直接关系，对象A作用对象C，和需要配置的二度关系有关，如配置同乘火车关系，则一度关系选择乘火车关系。

5. 单击**下一步**，进行关系元素定义。

基础查询条件定义中**查询属性选择**指在前端Web页面高级功能**关系分析**中，对于**关系类型**的组合查询和默认值的配置，根据业务情况，选择相关查询属性。**基础关系定义**中**关系属性选择**和**对象属性选择**直接定义物理表字段，也即需要选择物理表中哪些字段进行查询，同时通过**基础关系名称**定义该字段别名，为下一步**关系配置**做基础关系引用。其中**关联规则**配置的**默认设置值**将作为前端Web页面查询时的默认值。

6. 在**基础查询条件定义**区域单击**查询属性选择**按钮，根据业务需求，选择相关属性。
7. 在**基础关系定义**区域单击**关系属性选择**和**对象属性选择**按钮，根据业务需求，选择相关属性，如图 *8-63: 关系元素定义*所示。

图 8-63: 关系元素定义

说明：

- **基础查询条件定义**中的**查询属性选择**是指在前端 web 页面高级功能**关系分析**中，对于**关系类型**的组合查询和默认值的配置，根据业务情况，可以选择相关查询属性。
- **基础关系定义**中**关系属性选择**和**对象属性选择**直接定义物理表字段，也即需要选择物理表中哪些字段进行查询，同时通过**基础关系名称**定义该字段别名，为下一步的关系配置作基础关系引用。其中**关联规则**配置的**默认设置值**将作为前端 web 页面查询时的默认值。

8. 单击**下一步**，进行关系计算配置，如图 *8-64: 关系计算配置*所示。

图 8-64: 关系计算配置

说明：

- **关系配置**中的**新增关系配置**，作为数据库SQL组合执行语句，根据业务需求选择相应的组合关系。
- **关系组合**中的**新增关系组合**是前端Web页面呈现的组合查询选项，依赖于**关系配置**。其中：
  - 组合名称：自动生成。
  - 组合内容：根据业务需求选择。
  - 默认选中关系：在前端 web 页面显示默认选项值。
  - 是否互斥：指的组合条件中的选择是多选还是单选，选择是，指的是单选，选择否，指的是多选。
  - 默认次数：指出现该组合关系几次以上算符合查询条件，默认值2次。

9. 单击**新增关系配置**，显示**新增关系配置**对话框，如图 *8-65:* 新增关系配置所示。

图 8-65: 新增关系配置

10.根据业务需求，完成关系配置后，单击**确定**。

11.单击**新增关系组合**，选择原子关系中的内容为新增关系配置中的内容，根据上一步配置，已经勾选的关系将从该弹出框列表中移除，初始图如图 *8-66: 新增关系组合*所示。

图 8-66: 新增关系组合

12.根据业务需求，选择相关关系，单击**确定**。

13.根据业务需求，可重复上一步骤，选择多次关系组合，完成后单击**提交**即可。

8.1.6.4 配置多度关系

操作步骤

1. 登录关系网络分析管理后台。
2. 点击页面上方的**关系信息**页签，进入关系信息页面。

3. 点击**新建关系**图标，在弹出的下拉菜单中选择**多度关系**项，进入关系定义页面。
4. 编辑相关关系，如图 *8-67: 关系定义*所示，单击**下一步**。

图 8-67: 关系定义

说明：

源对象通过多度关系查询到**目标对象**。例如，由于源和目标对象手机之间没有通话未产生直接关系，所以通过手机直查手机无法关联，但由于其通过源对象手机发送邮件到目标对象手机收邮件，产生了间接关系，也即手机使用邮箱，邮箱到邮箱，邮箱到手机，涉及3个直接关系，最终目的是查手机到手机，因此源对象为手机，目标对象也为手机。要求是通过多个直接关系能够把源对象和目标对象串联起来。同样还有其他类似场景，手机用 QQ，手机用微信等。

5. 单击右上角的**关系选择**，显示**关系选择**页面，如图 *8-68: 关系选择*所示。

图 8-68: 关系选择

6. 根据多度关系业务要求，选择相关联的直接关系后，单击**确定**。

   单击**确定**后，可重复点**关系选择**按钮，重复选择直接关系。例如手机发邮件，查询方向是手机到邮箱；手机收邮件，查询方向是邮箱到手机，因此该关系可以选择两次，设置查询不同的查询方向，如图 *8-69: 图例*所示。

图 8-69: 图例

说明：

查询方向属性列对应的左右箭头通过点击可调整方向。

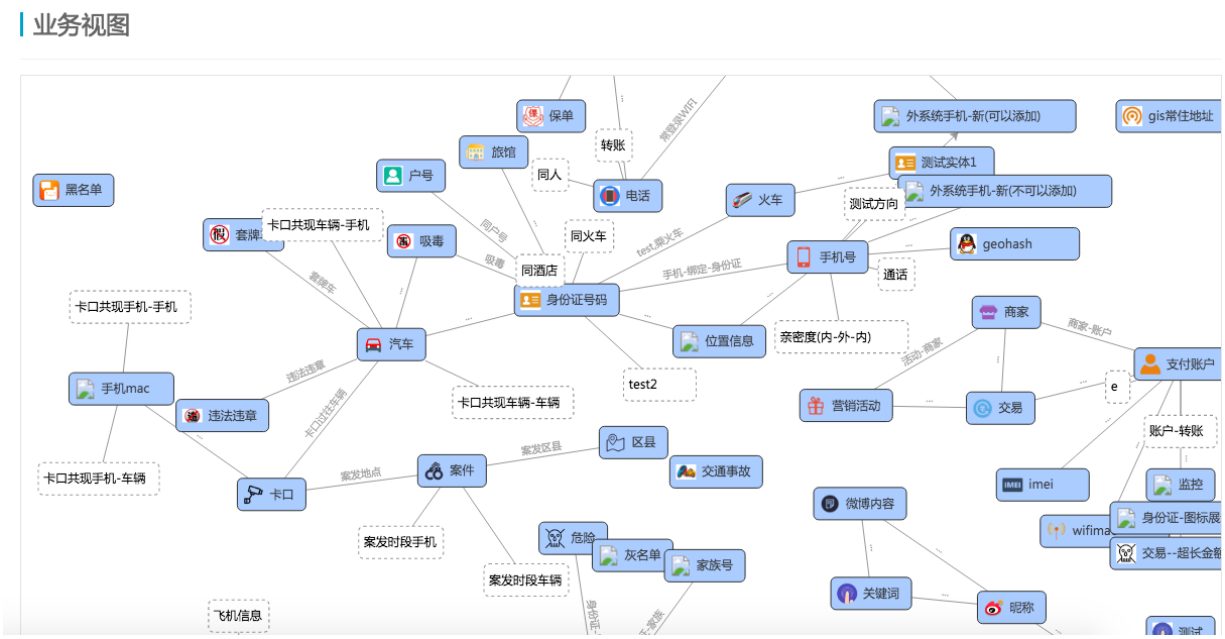
7. 最后单击**提交**，完成多度关系配置。

8.1.7 业务视图

单击页面上方的**关系信息**页签，进入业务视图页面。

业务视图页面中直观展示配置的所有关系模型，实线为直接关系，虚线为二度或多度关系，如图 *8-70:* 业务视图所示。

图 8-70: 业务视图



8.1.8 高级配置

8.1.8.1 导入模型

登录关系网络分析管理后台，单击页面上方的**高级配置** > **导入模型**菜单项，进入导入模型页面，如图 *8-71:* 导入模型页面所示。

图 8-71: 导入模型页面

新建模型

1. 单击页面右上角的**新建模型**按钮，弹出新建自定义数据模型窗口。

2. 选择模型类别，如图 *8-72: 选择模型类别*所示。

图 8-72: 选择模型类别

- 模型类型：模型类型包括对象、关系、对象和关系三种，请根据实际的导入数据表选择。
- 模型名称：输入模型的名称。

3. 单击**下一步**按钮。
4. 选择模型关系，如图 *8-73: 选择模型关系*所示。

图 8-73: 选择模型关系

- 在左侧的模型类别树中，选择具体的对象或关系。
- 右侧出现的模型字段区域，按照导入数据的具体情况，编辑勾选模型字段。

5. 单击**新建模型**按钮，完成新建模型操作。

查看模型

在导入模型列表中，可点击对应模型操作列的**查看**链接，在弹出的模型详情窗口中查看该模型的信息，如图 *8-74:* 模型详情窗口所示。

图 8-74: 模型详情窗口

删除模型

在导入模型列表中，可点击对应模型操作列的**删除**链接，删除该模型。

8.1.8.2 搜索配置

登录关系网络分析管理后台，单击页面上方的**高级配置** > **搜索配置**菜单项，进入搜索配置页面，如图 *8-75: 搜索配置页面*所示。

图 8-75: 搜索配置页面

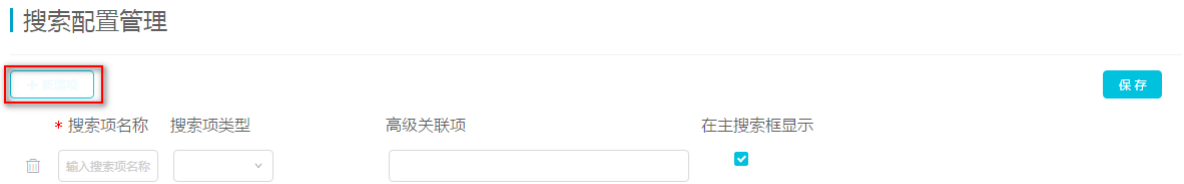
名词解释如下：

- 搜索项名称：显示在用户页面上的搜索名称，如手机号、身份证号码、姓名等。
- 搜索项类型：搜索项关系的实体，关系属性类型需要一致。
  - 日期类型
  - 时间类型
  - 数字区间类型
  - 字符串等于
  - 字符串模糊查询
  - 字典-下拉选项

新增搜索项

1. 单击页面左上角的**新增项**按钮，下方的搜索项列表中会新增一行，如图 *8-76: 新增项*所示。

   图 8-76: 新增项



2. 填写搜索项名称、搜索项类型、高级关联项，并选择是否在主搜索框显示。
3. 单击**保存**按钮。

修改搜索项

直接在搜索项列表中修改某一项的内容，单击**保存**按钮即可。

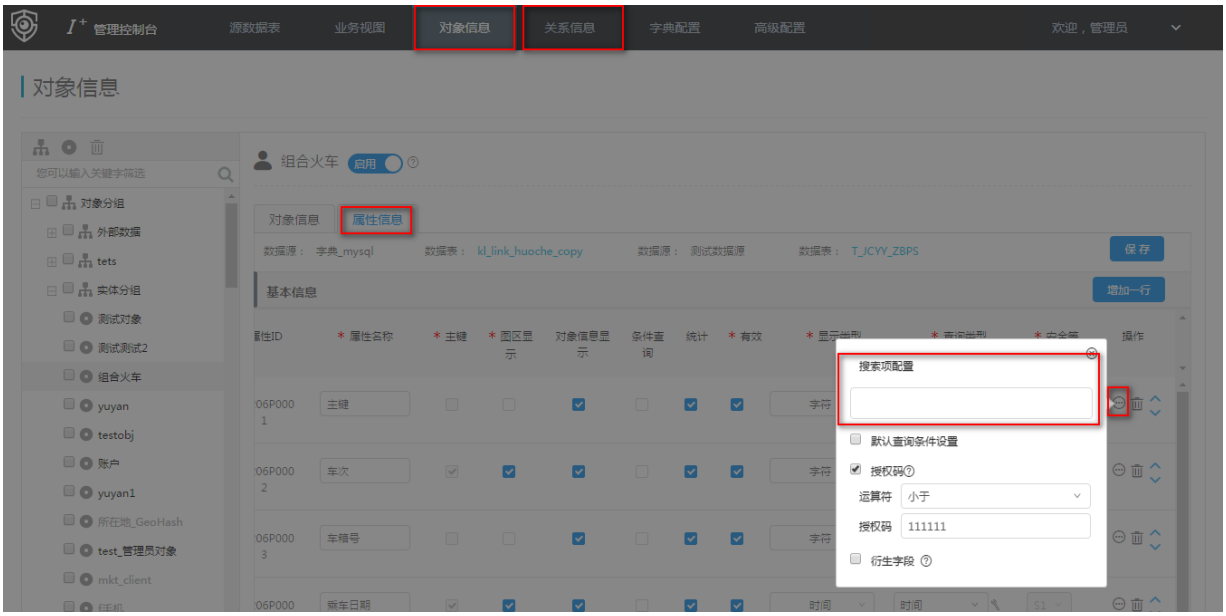
删除搜索项

点击待删除搜索项前边的**删除**图标，单击**保存**按钮即可。

属性关联搜索项

在搜索配置页面配置搜索项后，在对象信息或关系信息的属性信息页签页面，配置关系关联搜索项，如图 *8-77: 配置关系关联搜索项*所示。

图 8-77: 配置关系关联搜索项



8.1.8.3 系统配置

8.1.8.3.1 配置功能组件

前提条件

功能组件配置属于产品全局性功能启用、禁用功能，请根据实际需要谨慎配置。

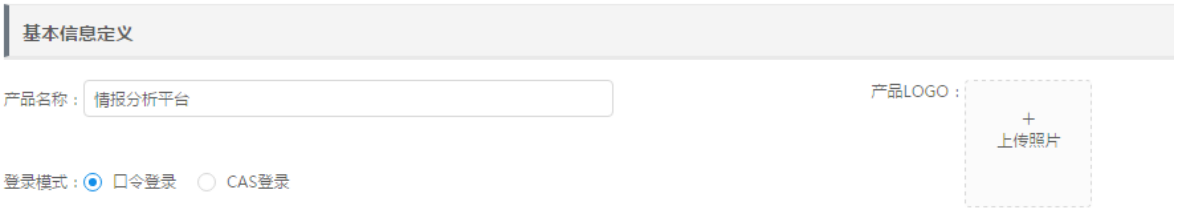
操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**高级配置** > **系统配置** > **功能组件配置**菜单项，进入功能组件配置页面，如图 *8-78: 功能组件配置*所示。

图 8-78: 功能组件配置

3. 根据需要，填写需要配置的产品名称、产品 LOGO 及登录方式，如图 *8-79: 基本信息定义*所示。

图 8-79: 基本信息定义



如果选择了 CAS 登录模式，如图 *8-80: 选择 CAS 登录模式*所示，请填写对应的信息，该配置需要与其他系统进行集成，配置时请与产品厂商联系。

图 8-80: 选择 CAS 登录模式

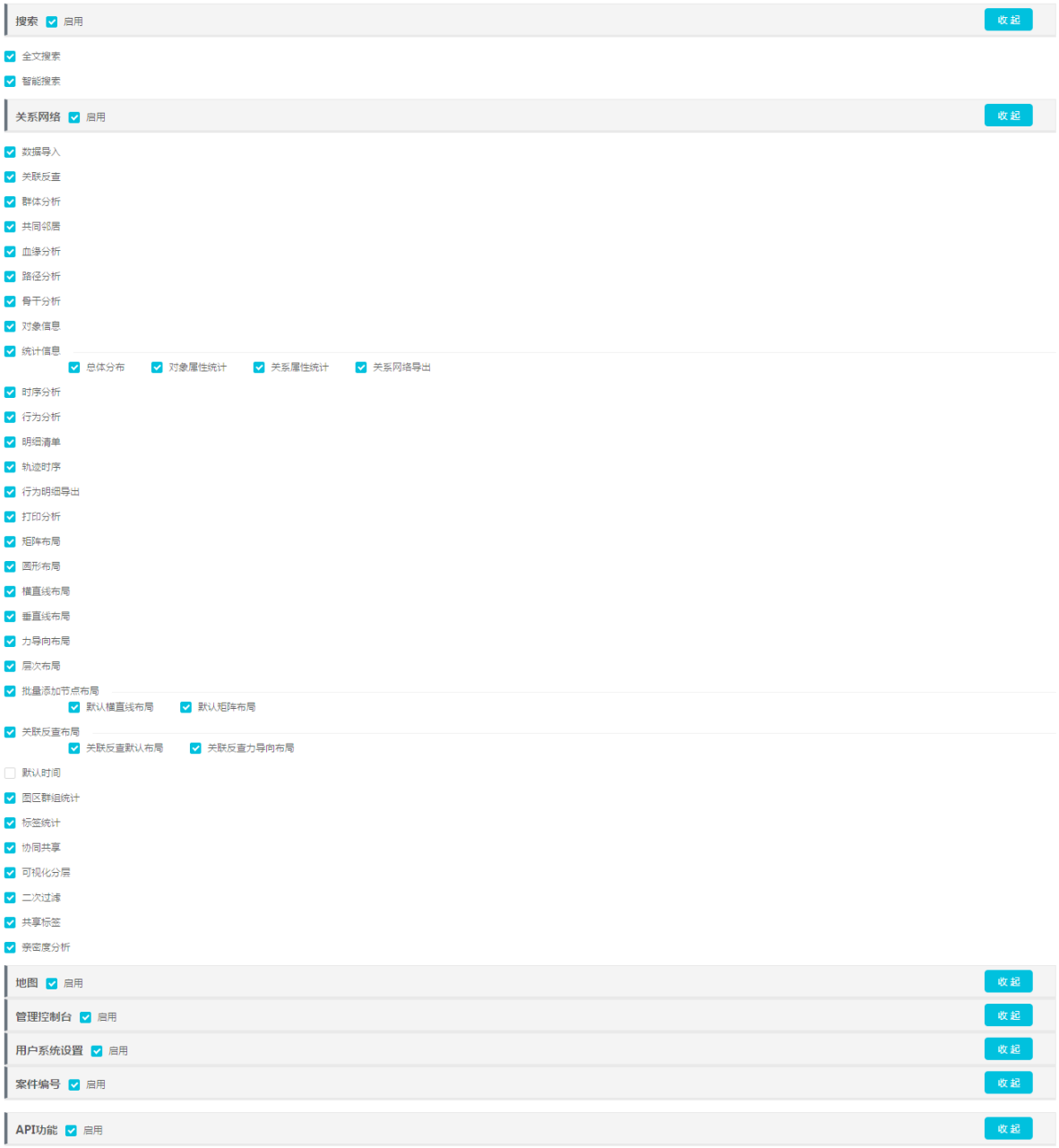
说明：

产品名称、产品 LOGO 没有填写时，将采用系统自带的名称和 LOGO。

4. 勾选需要启用或禁用的模块，如图 *8-81: 勾选启用或禁用的模块*所示。

该模块分为三级组织，勾选或取消勾选父节点，会同步勾选或取消勾选子节点。

图 8-81: 勾选启用或禁用的模块



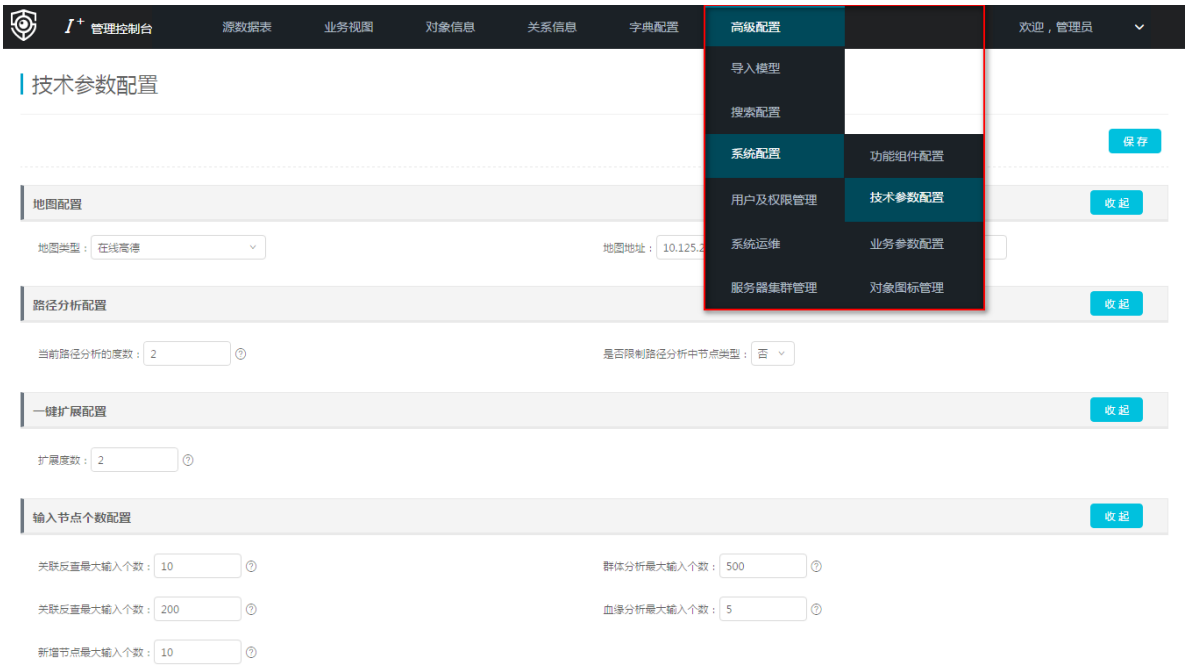
5. 完成后，单击**保存**，保存配置内容。

8.1.8.3.2 配置技术参数

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**高级配置** > **系统配置** > **技术参数配置**菜单项，进入技术参数配置页面，如图 *8-82:* 技术参数配置页面所示。

图 8-82: 技术参数配置页面



8.1.8.3.2.1 地图配置

前提条件

该选项一般采用系统默认配置，请谨慎配置。

1. 在技术参数配置页面的**地图配置**区域框中，设置如下信息，如图 *8-83: 地图配置*所示。
   - 地图类型：选择所使用地图的类型，包括在线高德、离线高德和PGIS。
   - 地图地址：输入所使用地图的访问地址。

图 8-83: 地图配置

8.1.8.3.2.2 路径分析配置

前提条件

该选项一般采用系统默认配置，请谨慎配置。

操作步骤

在技术参数配置页面的**路径分析配置**区域框中，设置如下信息，如图 *8-84: 路径分析配置*所示。

- 当前路径分析的度数：设置需要配置的度数，不超过3。
- 是否需要限制路径中的节点类型：如果选择**是**，则路径分析过程中，只计算选中节点相同类型的对象，反之支持所有类型的节点。

图 8-84: 路径分析配置

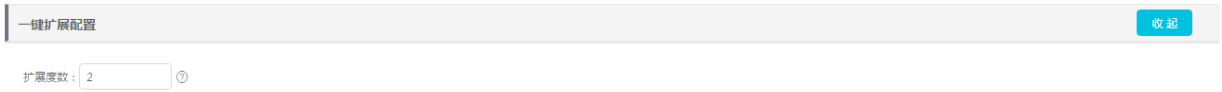
8.1.8.3.2.3 一键扩展配置

前提条件

该选项一般采用系统默认配置，请谨慎配置。

1. 在技术参数配置页面的**一键扩展配置**区域框中，设置如下信息，如图 *8-85: 一键扩展配置*所示。
   - 扩展度数：设置关联反查操作一次执行几步。

图 8-85: 一键扩展配置



8.1.8.3.2.4 输入节点个数配置

前提条件

该选项一般采用系统默认配置，请谨慎配置。

操作步骤

在技术参数配置页面的**输入节点个数配置**区域框中，设置允许输入的节点个数，如图 *8-86: 输入节点个数配置*所示，一般建议直接采用产品默认配置。

设置后，分析工作台将允许所设数值范围的节点添加、选中分析。

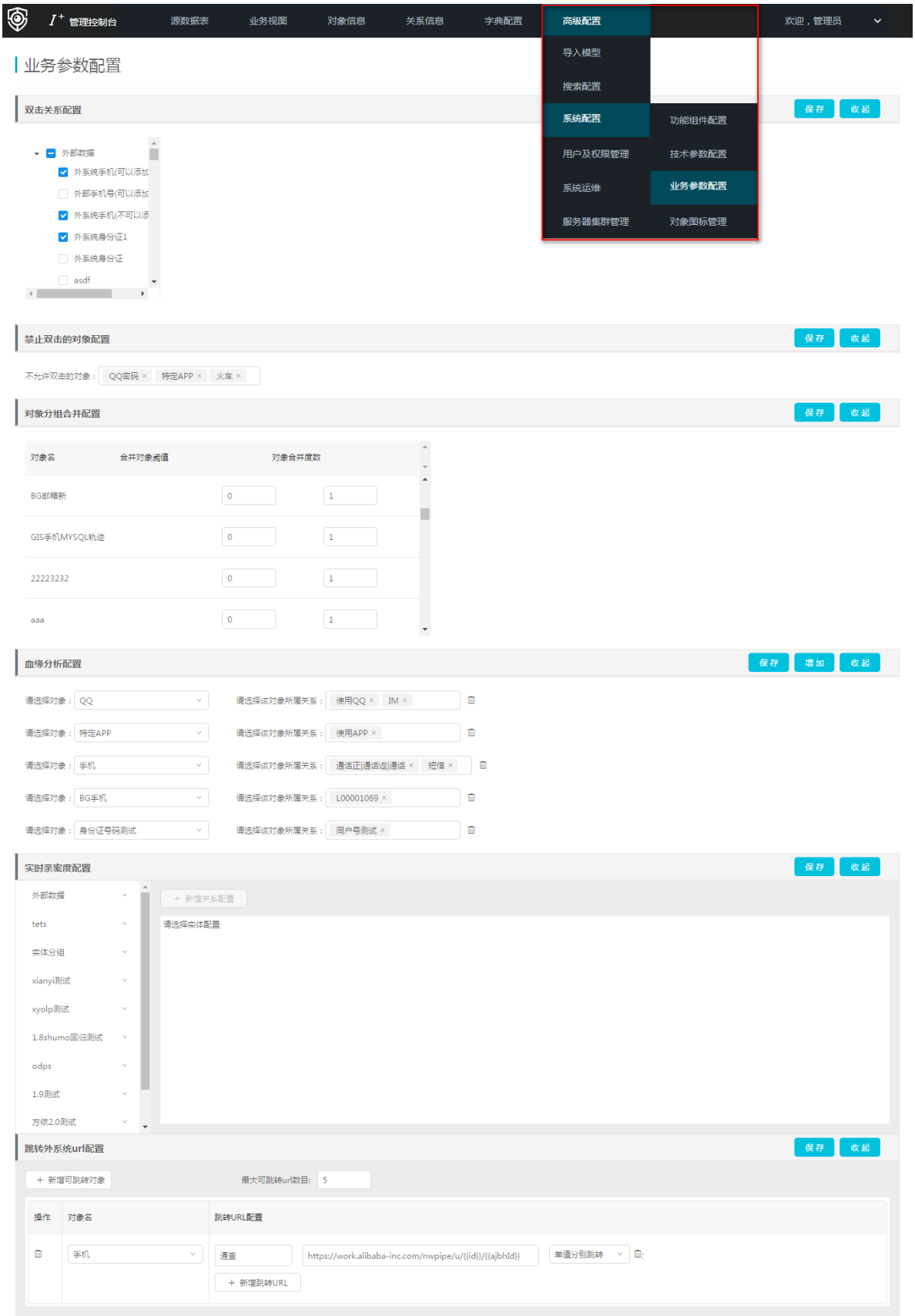
图 8-86: 输入节点个数配置

8.1.8.3.3 配置业务参数

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**高级配置** > **系统配置** > **业务参数配置**菜单项，进入业务参数配置页面，如图 *8-87:* 业务参数配置页面所示。

图 8-87: 业务参数配置页面



8.1.8.3.3.1 双击关系配置

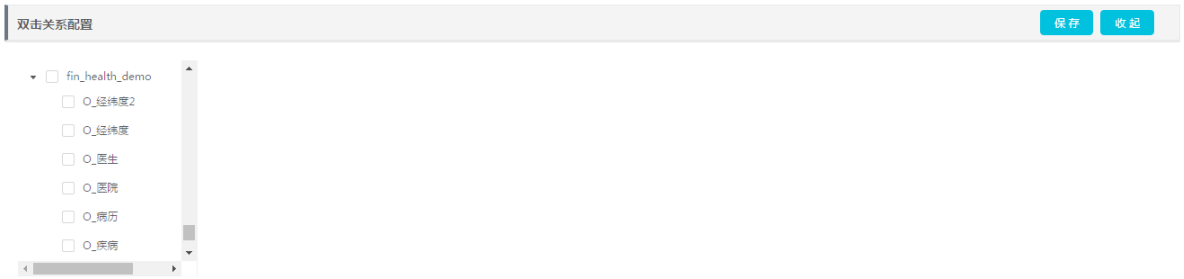
前提条件

双击关系配置对全局生效，请谨慎进行配置。

操作步骤

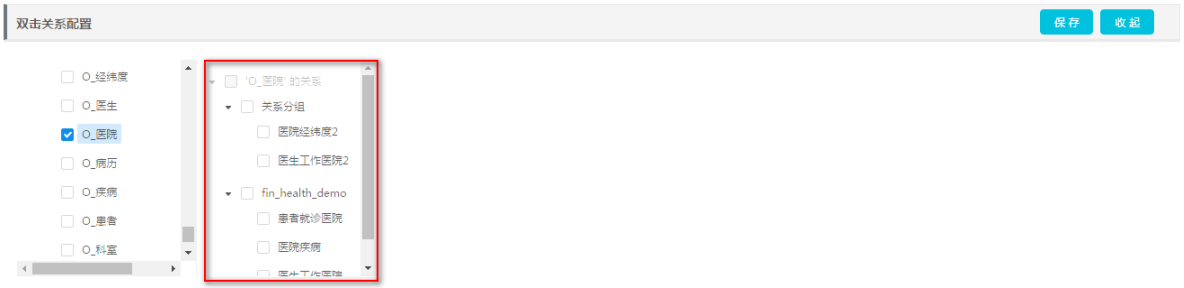
1. 在业务参数配置页面的**双击关系配置**区域框中，查看双击关系配置的情况，初始可以查看配置双击关联反查的对象，如图 *8-88: 查看双击关联反查对象*所示。

图 8-88: 查看双击关联反查对象



2. 单击左侧对象，在右侧显示配置的关系，如图 *8-89: 显示配置关系*所示，可以勾选需要配置双击的关系。

图 8-89: 显示配置关系



单击对应的关系，配置双击关联反查的关系条件，如图 *8-90: 配置关系条件*所示。

图 8-90: 配置关系条件

3. 配置后，单击**保存**。

8.1.8.3.3.2 禁止双击的对象配置

前提条件

禁止双击对象配置对全局生效，请谨慎进行配置，建议将产生数据较多的对象双击操作禁用。

操作步骤

1. 在业务参数配置页面的**禁止双击的对象配置**区域框中，单击下拉框，选择不允许双击的对象，如图 *8-91: 禁止双击的对象配置*所示。

图 8-91: 禁止双击的对象配置

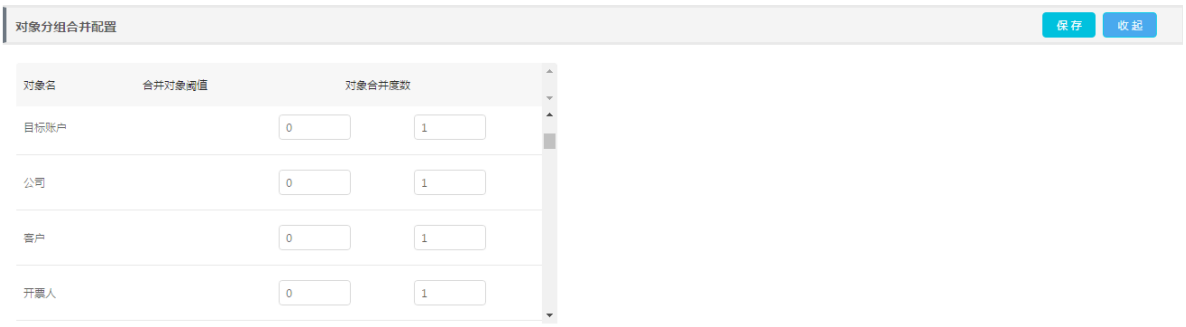
2. 配置后，单击**保存**，保存配置信息。

8.1.8.3.3.3 对象分组合并配置

操作步骤

1. 在业务参数配置页面的**对象分组合并配置**区域框中，设置全局合并阈值以及每个对象合并的阈值配置，如图 *8-92: 对象分组合并配置*所示。

图 8-92: 对象分组合并配置



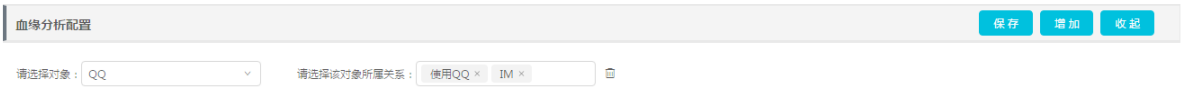
2. 配置后，单击**保存**，保存配置信息。

8.1.8.3.3.4 血缘分析配置

操作步骤

1. 在业务参数配置页面的**血缘分析配置**区域框中，单击右上角的**增加**按钮，显示配置区域。
2. 输入分析名称，并选择需要进行血缘分析的对象，及其所属关系，如图 *8-93: 血缘分析配置*所示。

图 8-93: 血缘分析配置



3. 配置后，单击**保存**按钮，保存配置信息。

8.1.8.3.3.5 实时亲密度配置

1. 在业务参数配置页面的**实时亲密度配置**区域框中，选择一个或多个关系。
2. 在右侧输入框中单击**新增关系配置**按钮，选择关系类型及其权重。
3. 单击**保存**按钮，保存关系配置。
4. 单击**实时亲密度配置**区域框右上角的**保存**按钮，完成实时亲密度的配置。

图 8-94: 实时权重配置页面

8.1.8.3.3.6 跳转外系统URL设置

该设置是需要通过I+跳转到其他系统的时用到的。

8.1.8.3.4 管理对象图标

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**高级配置** > **系统配置** > **对象图标管理**菜单项，进入管理对象图标页面页面，如图 *8-95:* 管理对象图标页面所示。

图 8-95: 管理对象图标页面

8.1.8.3.4.1 上传图标

操作步骤

1. 在管理对象图标页面，单击**上传图标**按钮。
2. 在弹出的上传图标窗口中，点击上传区域，从本地上传图标，上传后请填写图标名称，如图 *8-96:* 上传图标所示。

图 8-96: 上传图标

3. 单击**确定**按钮，完成上传图标操作。

8.1.8.3.4.2 修改图标

操作步骤

1. 在管理对象图标页面，将鼠标悬浮在待修改的图标下方，点击**修改**图标，如图 *8-97: 修改图标*所示。

图 8-97: 修改图标

2. 在弹出的编辑窗口中修改名称或重新上传图标，如图 *8-98: 修改图标*所示。

图 8-98: 修改图标

3. 修改完成后，单击**确定**按钮。

8.1.8.3.4.3 删除图标

操作步骤

1. 在管理对象图标页面，将鼠标悬浮在图标下方，点击**删除**图标，如图 *8-99: 删除图标*所示。

图 8-99: 删除图标

2. 在弹出的对话框中，单击**确定**按钮。

8.1.8.4 管理用户及权限

8.1.8.4.1 管理单位及用户

8.1.8.4.1.1 新建单位

前提条件

建议在创建单位及用户时，预先整理一份表格，明确组织架构及每个部门的编号，同时明确每个部门的人员、职责、工号等信息。

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**高级配置** > **用户及权限管理** > **单位及用户管理**菜单项，进入单位及用户管理页面，如图 *8-100:* 单位及用户管理页面所示。

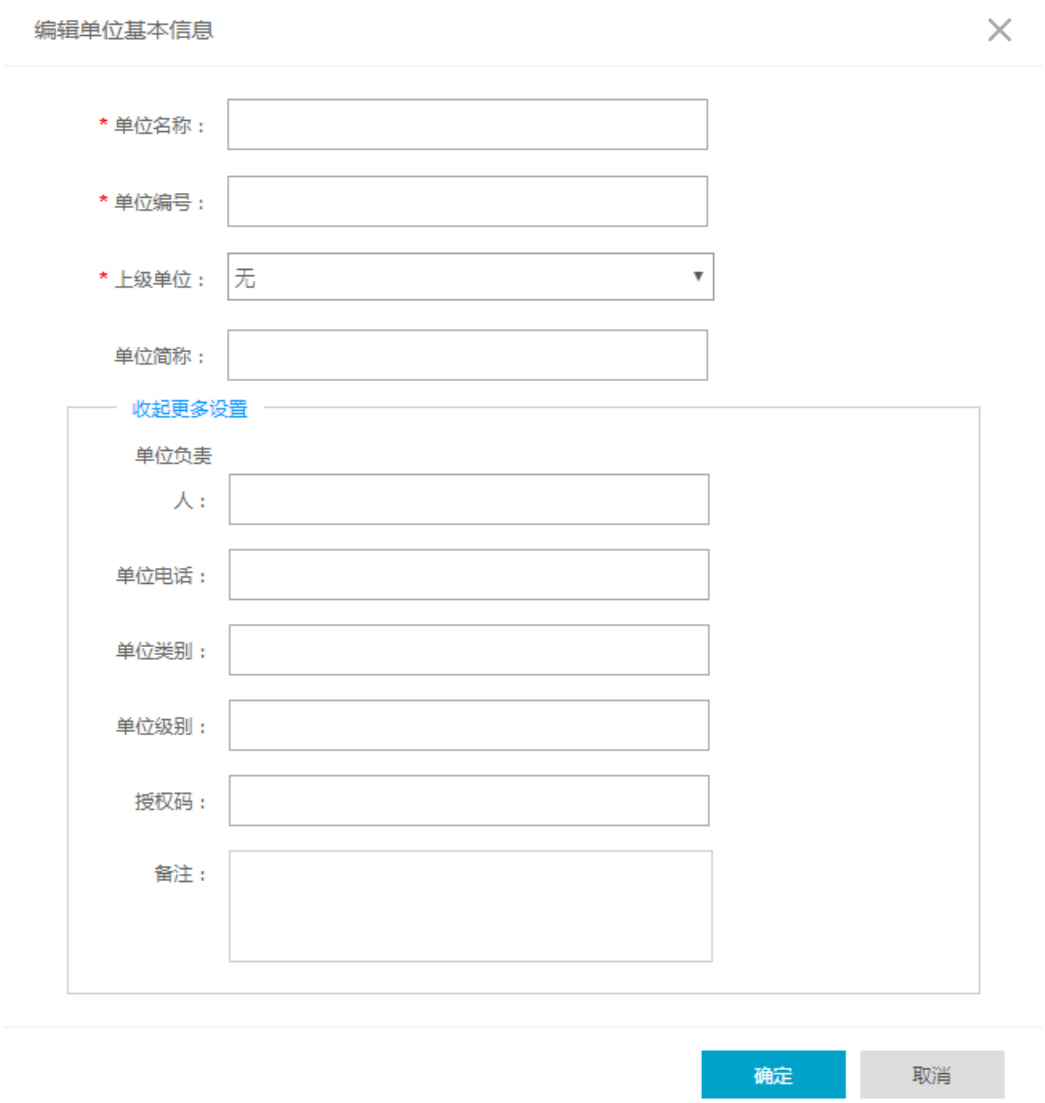
图 8-100: 单位及用户管理页面

3. 在单位及用户管理页面，点击**新建单位**图标，如图 *8-101:* 新建单位所示。

图 8-101: 新建单位

4. 输入单位信息，如图 *8-102:* 输入单位信息所示。

图 8-102: 输入单位信息



首次创建时没有上级单位，因此第一次创建的单位名称即为根节点，因此**上级单位**内容显示为**无**。

5. 完成后，单击**确定**按钮，生成单位信息，如图 *8-103:* 生成单位信息所示。

图 8-103: 生成单位信息

6. 重复创建单位的步骤，创建部门，完成后，单击**确定**按钮。

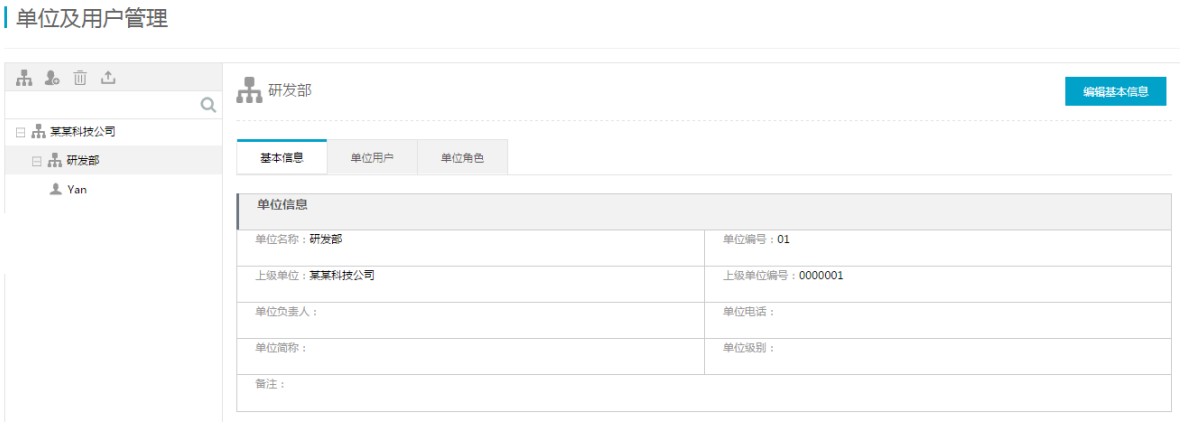
   创建部门时，根据需要，选择对应**上级单位**的单位名称。

7. 重复上一步骤，创建多个部门，生成单位或公司组织结构如图 *8-104:* 单位组织结构所示。

图 8-104: 单位组织结构

8. 在各部门的**基本信息**页签中，可以查看部门属性信息，有基本信息、单位用户、单位角色，如图 *8-105:* 基本信息页签所示。

图 8-105: 基本信息页签



9. 在各部门的**单位用户**页签中，可以查看单位用户信息，如图 *8-106:* 单位用户页签所示。

图 8-106: 单位用户页签

通过**添加用户**按钮，可在该部门下添加用户。

10. 在各部门的**单位角色**页签中，可以查看单位角色信息，如图 *8-107:* 单位角色页签所示。

图 8-107: 单位角色页签

通过**添加角色**按钮，可为单位添加角色。

说明：

- 对于单位授权，应授予最小权限，一般建议只授予只读或者查看权限。

- 对于用户授权，可根据用户角色不同，授予不同的权限。也即单位的权限要小于或等于任何一个用户的权限。

8.1.8.4.1.2 新建用户

管理员可通过新建用户操作为单位员工创建登录I+分析工作台的帐号，新建的用户初始密码为“iplus1688”，用户首次登录系统后会提示修改用户密码。

前提条件

建议在创建单位及用户时，预先整理一份表格，明确组织架构及每个部门的编号，同时明确每个部门的人员、职责、工号等信息。

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**高级配置** > **用户及权限管理** > **单位及用户管理**菜单项，进入单位及用户管理页面，如图 *8-108: 单位及用户管理页面*所示。

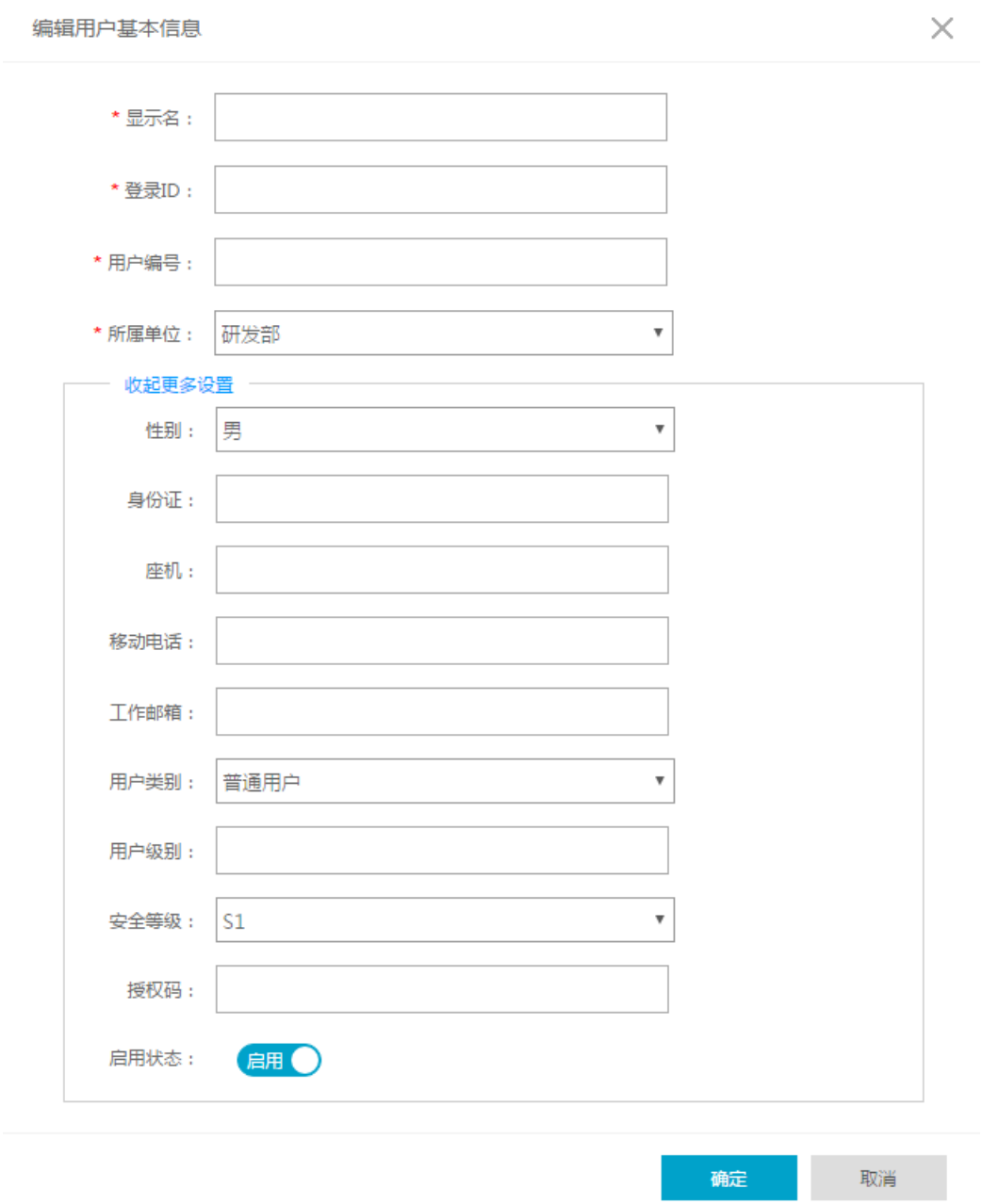
图 8-108: 单位及用户管理页面

3. 在单位及用户管理页面，点击**新建用户**图标，如图 *8-109: 新建用户*所示。

图 8-109: 新建用户

4. 在弹出的对话框中，添加用户相关信息，如图 *8-110: 添加用户相关信息*所示。

图 8-110: 添加用户相关信息

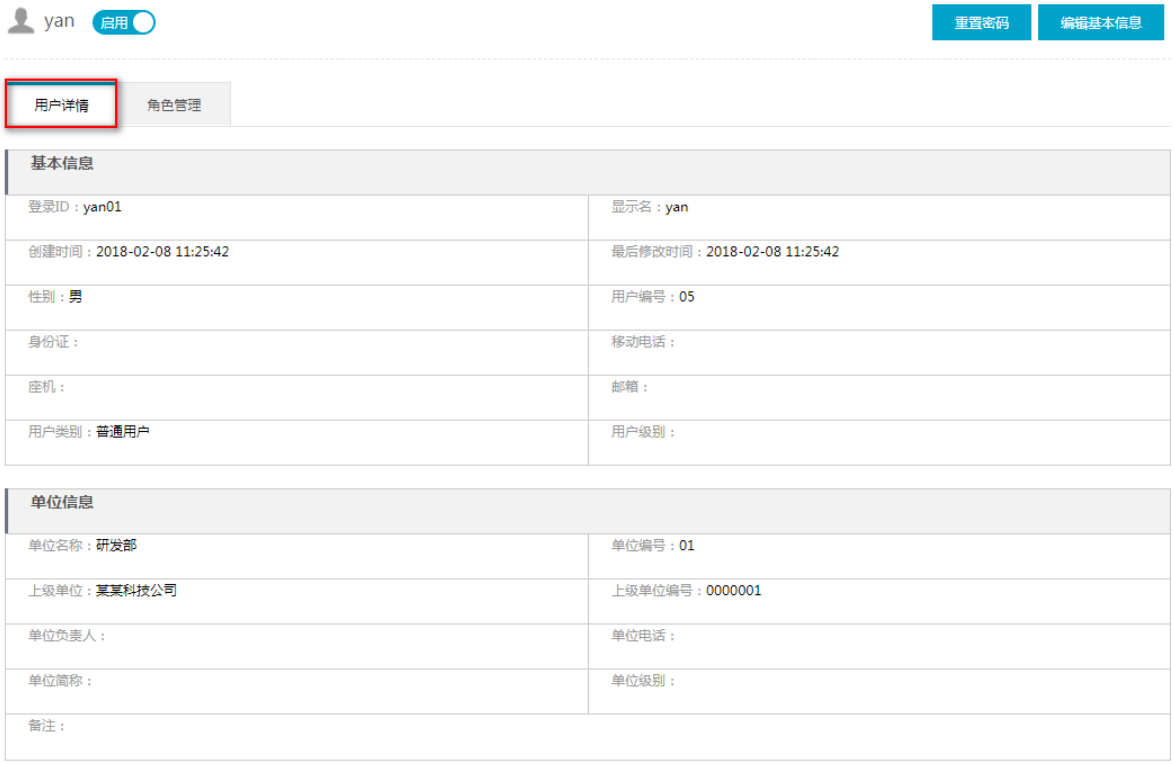


说明：

普通管理员不可选择用户类别；超级管理员可以选择管理员和普通用户。

5. 在**用户详情**页签，可以查看用户的基本信息和单位信息，如图 *8-111:* 用户详情页签所示。

图 8-111: 用户详情页签



可通过单击**重置密码**按钮，将用户密码修改为初始密码“iplus1688”。

6. 在**角色管理**页签，单击**添加角色**，为用户添加角色，如图 *8-112:* 添加角色所示。

图 8-112: 添加角色

添加完成后，结果如图 *8-113:* 角色添加成功所示。

图 8-113: 角色添加成功

8.1.8.4.1.3 批量上传用户

前提条件

建议在创建单位及用户时，预先整理一份表格，明确组织架构及每个部门的编号，同时明确每个部门的人员、职责、工号等信息。

操作步骤

1. 准备用户账户信息。

   您可以参照以下模板，如图 *8-114: 用户账户信息*所示，将用户信息保存成txt或者csv格式文件。推荐首行统一为 userId、name、password、orgName、orgId 字段。

   - userId：登录Id
   - name：显示名称
   - password：密码（可为空）
   - orgName：机构名称
   - orgId：机构编码

   图 8-114: 用户账户信息

   账号体系.csv

   ```
   登陆ID,昵称,密码,组织,组织ID
   userId,name,password,orgName,orgId
   hongtang1111,红塘1,123456,alibaba,31
   hongtang2111,红塘2,123456,alibaba,31
   hongtang3111,红塘3,123456,alibaba,31
   hongtang4111,红塘4,123456,alibaba,31
   ```

2. 登录关系网络分析管理后台。
3. 单击页面上方的**高级配置** > **用户及权限管理** > **单位及用户管理**菜单项，进入单位及用户管理页面，如图 *8-115: 单位及用户管理页面*所示。

   图 8-115: 单位及用户管理页面

4. 在单位及用户管理页面，点击**上传**图标，弹出上传客户账号窗口。
5. 单击**浏览**按钮，选择需要导入的txt或csv文件，如图 *8-116: 上传文件*所示。

图 8-116: 上传文件

6. 单击**下一步**，系统默认预览10条记录，单击**导入**即可将数据导入配置库中。
7. 导入成功后，您可以在左侧目录树中查看单位及用户信息，如图 *8-117:* 查看导入信息所示。

图 8-117: 查看导入信息

8.1.8.4.2 管理角色

8.1.8.4.2.1 新建角色

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**高级配置** > **用户及权限管理** > **角色管理**菜单项，进入角色管理页面，如图 *8-118:* 角色管理页面所示。

图 8-118: 角色管理页面

3. 在角色管理页面，点击**新建角色**按钮，如图 *8-119: 新建角色*所示。

图 8-119: 新建角色

4. 在弹出的编辑基本信息窗口中，填写相关角色信息，如图 *8-120: 输入角色相关信息*所示。

图 8-120: 输入角色相关信息

5. 单击**确定**按钮，完成新建角色操作。

8.1.8.4.2.2 查看角色

可以查看基本信息、对象关系授权、功能组件授权及被添加对象信息。

8.1.8.4.2.2.1 基本信息

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**高级配置** > **用户及权限管理** > **角色管理**菜单项，进入角色管理页面，如图 *8-121:* 角色管理页面所示。

图 8-121: 角色管理页面

3. 在角色管理页面，单击左导航树中的某一个角色，页面右侧显示该角色的基本信息，如图 *8-122:* 查看角色基本信息所示。

图 8-122: 查看角色基本信息

用户可单击**编辑基本信息**按钮，编辑角色的基本信息。

8.1.8.4.2.2.2 对象关系授权

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**高级配置** > **用户及权限管理** > **角色管理**菜单项，进入角色管理页面，如图 *8-123:* 角色管理页面所示。

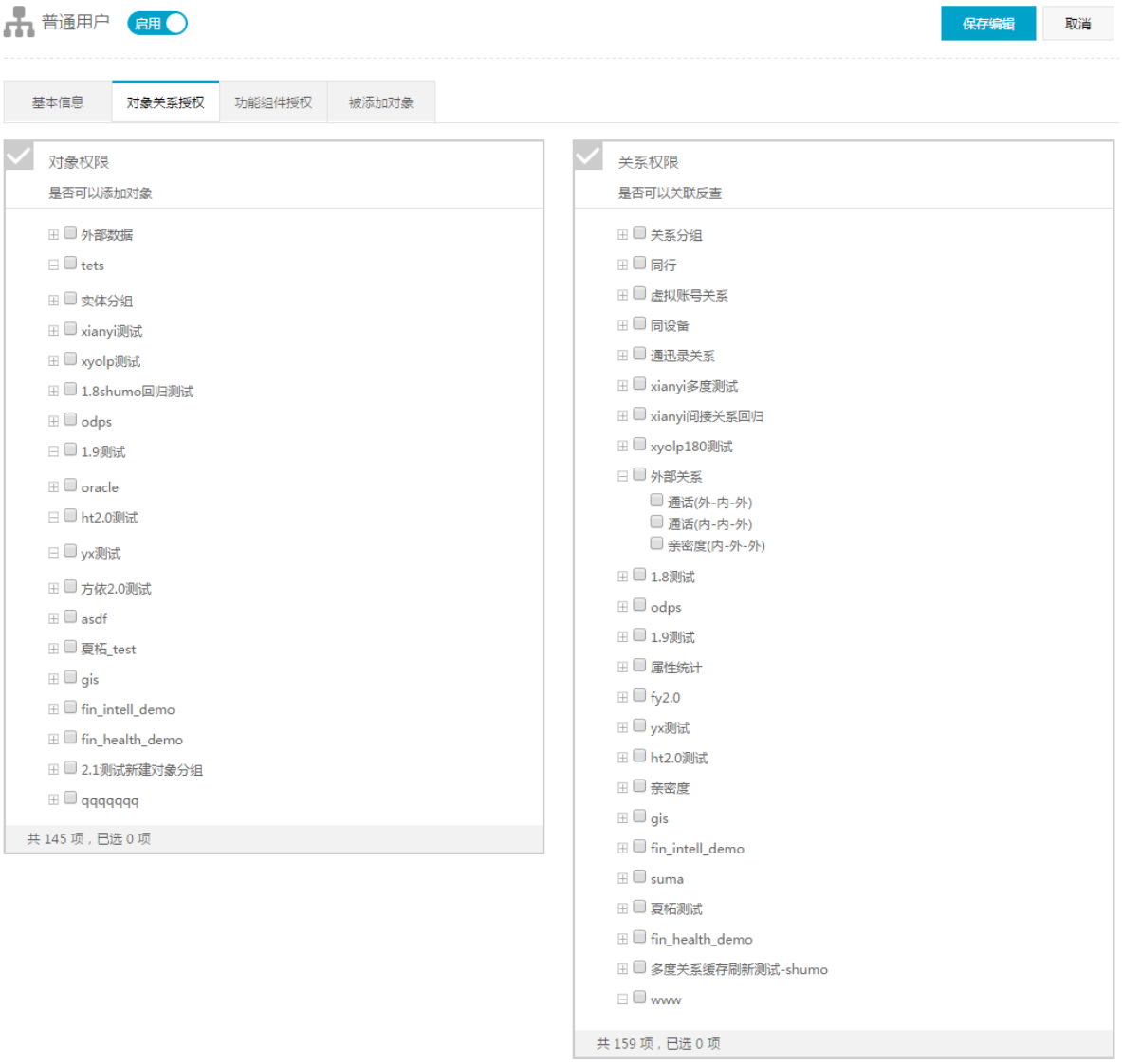
图 8-123: 角色管理页面

3. 在角色管理页面，单击左导航树中的某一个角色，页面右侧显示该角色的基本信息。
4. 单击**对象关系授权**页签，如图 *8-124: 对象关系授权*所示。

图 8-124: 对象关系授权

5. 单击**编辑角色权限**按钮，查看对象关系权限信息，如图 *8-125: 查看角色权限*所示。

图 8-125: 查看角色权限



说明：

对象权限和**关系权限**所显示的内容和OLEP配置的内容有直接关系，如果OLEP配置没有信息，则该对象关系授权属性中无可选内容。

6. 单击**保存编辑**按钮，退出。

8.1.8.4.2.2.3 功能组件授权

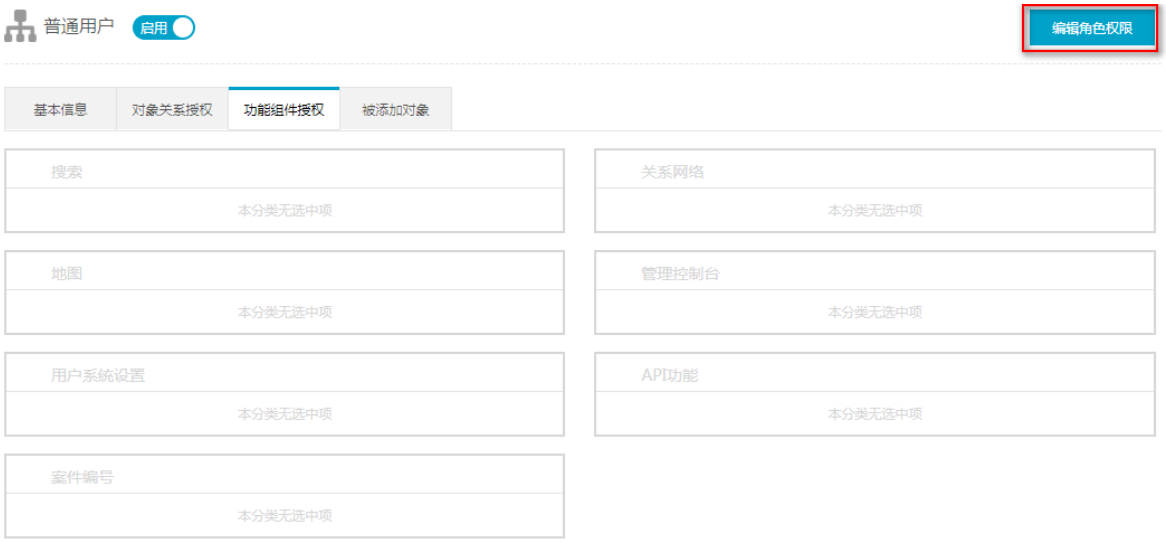
操作步骤

1. 单击页面上方的**高级配置** > **用户及权限管理** > **角色管理**菜单项，进入角色管理页面，如图 *8-126:* 角色管理页面所示。

图 8-126: 角色管理页面

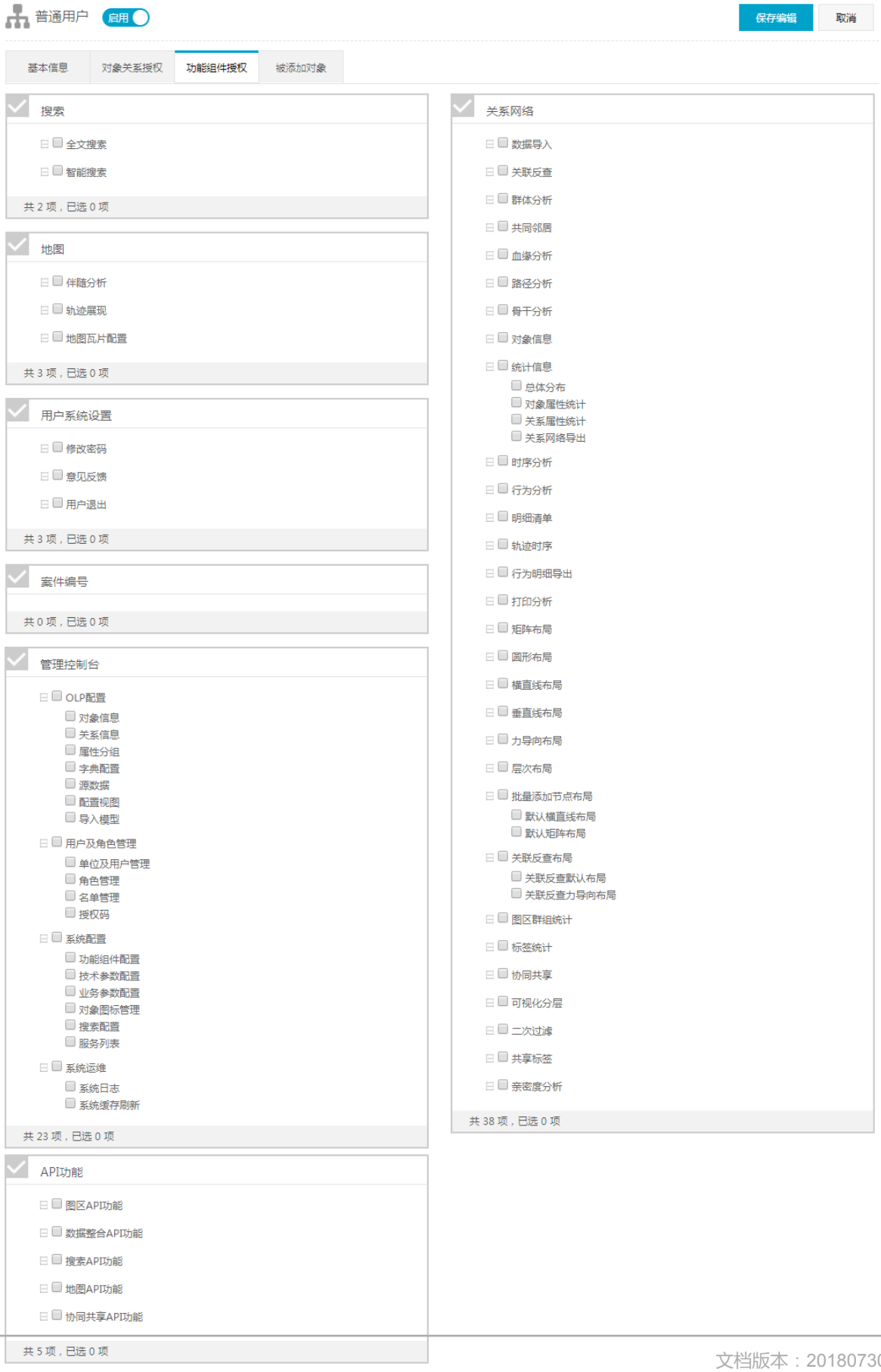
2. 在角色管理页面，单击左导航树中的某一个角色，页面右侧显示该角色的基本信息。
3. 单击**功能组件授权**页签，如图 *8-127: 功能组件授权*所示。

图 8-127: 功能组件授权



4. 单击**编辑角色权限**，出现如图 *8-128: 编辑角色权限*所示功能组件选项，根据实际角色权限要求，选择相关功能。

图 8-128: 编辑角色权限



5. 完成后，单击**保存编辑**按钮。

8.1.8.4.2.2.4 为单位和用户添加授权

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**高级配置** > **用户及权限管理** > **角色管理**菜单项，进入角色管理页面，如图 *8-129: 角色管理页面*所示。

图 8-129: 角色管理页面

3. 在角色管理页面，单击左导航树中的某一个角色，页面右侧显示该角色的基本信息。
4. 单击**被添加对象**页签，如图 *8-130: 被添加对象页签*所示。

图 8-130: 被添加对象页签

5. 单击**添加授权**按钮，如图 *8-131: 添加授权*所示，对单位及用户管理章节所创建的用户赋权限。

图 8-131: 添加授权

完成添加后，如图 *8-132: 添加个人授权成功*所示。

图 8-132: 添加个人授权成功

6. 也可添加部门，如图 *8-133: 添加部门*所示，对单位及用户管理章节所创建的部门赋权限。

图 8-133: 添加部门

8.1.8.4.3 管理系统标签

8.1.8.4.3.1 新建分组

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**高级配置** > **用户及权限管理** > **系统标签**菜单项，进入角色管理页面，如图 *8-134: 系统标签页面*所示。

图 8-134: 系统标签页面

3. 在**系统标签**页面，单击**新建分组**图标，如图 *8-135: 新建分组*所示。

图 8-135: 新建分组

4. 在弹出的新建分组窗口中，输入分组名称，如图 *8-136: 输入分组名称*所示。

图 8-136: 输入分组名称

5. 单击**确定**按钮，系统标签分组新建成功。

8.1.8.4.3.2 新建系统标签

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**高级配置** > **用户及权限管理** > **系统标签**菜单项，进入角色管理页面，如图 *8-137: 系统标签页面*所示。

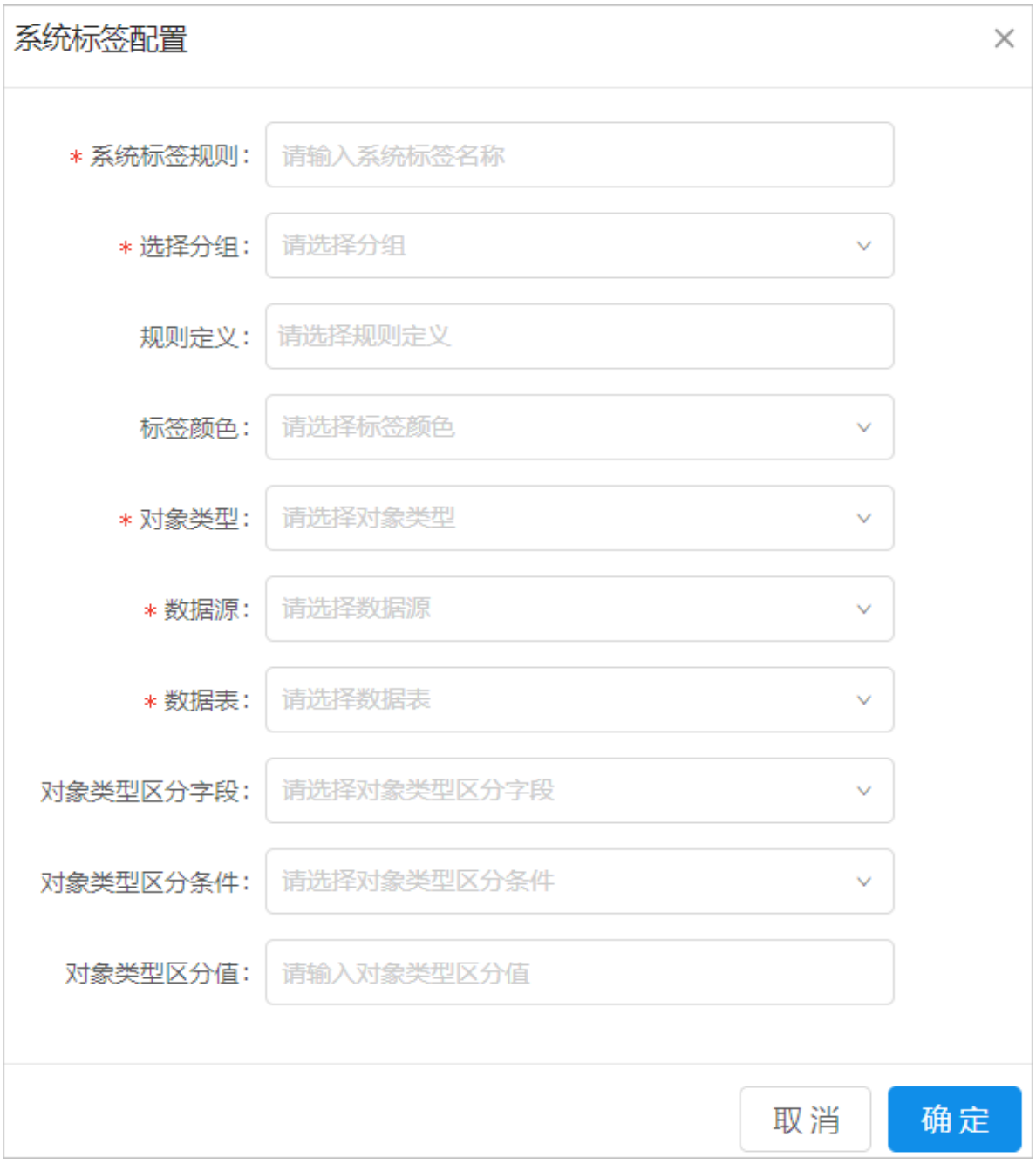
图 8-137: 系统标签页面

3. 在系统标签页面，单击**新建系统标签**按钮，如图 *8-138: 新建系统标签*所示。

图 8-138: 新建系统标签

4. 在弹出的**系统标签配置**对话框中输入相应的信息，如图 *8-139:* 系统标签配置所示。

图 8-139: 系统标签配置



相关配置信息说明如下：

- 系统标签规则：该份系统标签的别名。
- 选择分组：系统标签所在的分组。
- 规则定义：用于定义对象与系统标签匹配时的处理规则，分别为**对象不显示**、**对象不扩展**、**属性不显示**、**属性不统计**、**行为不展示**，当用户选择了**对象不显示**时，其余四项和标签颜色均变为不可选。

- 标签颜色：系统标签展示的颜色定义。
- 对象类型：对象信息中的所有可用对象，在此选择的对象将进行系统标签的过滤，当选择对象类型后，弹窗中会动态新增对象主键属性，在对应的下拉框中请选择系统标签数据表中与主键碰撞的字段。
- 数据源：系统标签所在的数据源。
- 数据表：系统标签所在数据表。
- 对象类型区分字段、对象类型区分条件、对象类型区分值：非必填项，根据实际系统标签数据的存储情况进行系统标签的过滤，三个选项中如果填写了一项，其余两项为必填。

5. 单击**确定**按钮，新建系统标签成功。

8.1.8.4.3.3 修改系统标签

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**高级配置** > **用户及权限管理** > **系统标签**菜单项，进入角色管理页面，如图 *8-140: 系统标签页面*所示。

图 8-140: 系统标签页面

3. 在系统标签页面，单击左导航树中的某个待修改系统标签名称。
4. 在页面右侧的系统标签详细信息区域，单击**编辑**链接，如图 *8-141: 修改系统标签*所示。

图 8-141: 修改系统标签

5. 编辑需要修改的信息，修改时，对象类型不可修改。
6. 单击**确定**按钮，修改成功。

8.1.8.4.3.4 删除系统标签

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**高级配置** > **用户及权限管理** > **系统标签**菜单项，进入角色管理页面，如图 *8-142: 系统标签页面*所示。

图 8-142: 系统标签页面

3. 单击**删除**或左侧勾选需要删除的系统标签，单击下方的**删除**，即可删除系统标签，如图 *8-143: 删除系统标签*所示。

图 8-143: 删除系统标签

8.1.8.5 系统运维

8.1.8.5.1 审计日志

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**高级配置** > **系统运维** > **审计日志**菜单项，进入审计日志页面。
3. 输入对应的查询条件，单击**开始查询**按钮，如图 *8-144: 查询日志*所示。

   没有设置条件的情况下单击**开始查询**按钮后，会返回所有数据。

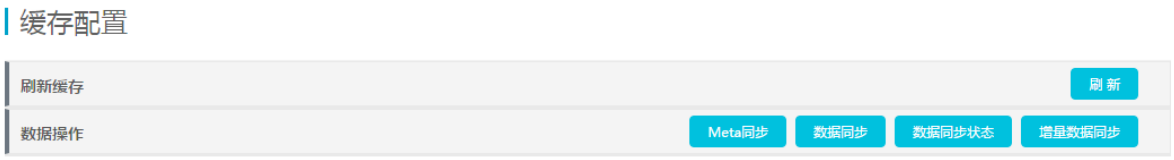
图 8-144: 查询日志

8.1.8.5.2 缓存配置

操作步骤

1. 登录关系网络分析管理后台。
2. 单击页面上方的**高级配置** > **系统运维** > **缓存配置**菜单项，进入缓存配置页面。
3. 配置信息修改后，请单击**刷新**，如图 *8-145: 缓存配置页面*所示。

图 8-145: 缓存配置页面



4. 数据操作支持Meta同步、数据同步、数据同步状态和增量同步操作。

8.2 用户指南

8.2.1 什么是关系网络分析

关系网络分析，又名Alibaba Cloud I+（以下简称I+）是基于关系网络的大数据可视化分析平台，在阿里巴巴、蚂蚁金服集团内广泛应用于反欺诈、反作弊、反洗钱等风控业务，面向公安、税务、海关、银行、保险、互联网等提供行业解决方案。

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。

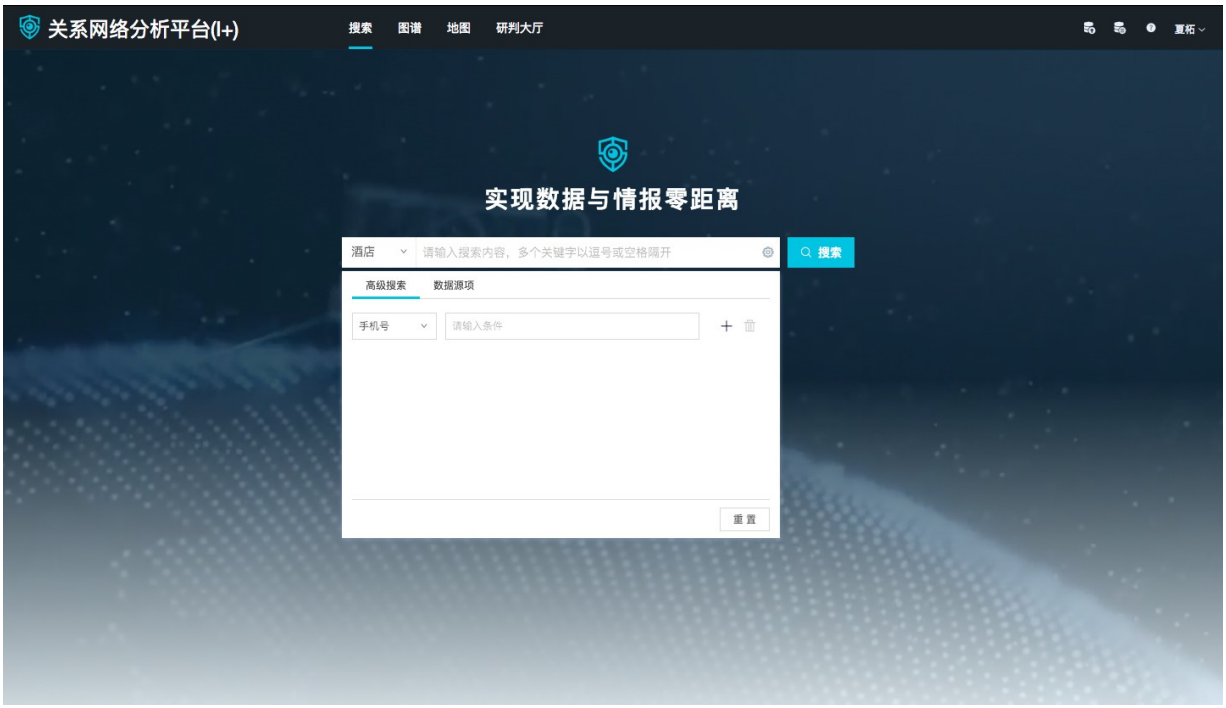
I+产品是依托于关系网络，从时间、空间维度来提供交互式分析的大数据处理平台。主体业务模块有：搜索、关系网络，时空分析三大部分。同时产品主要着力于安全相关领域，权限控制也是产品重要模块。

8.2.1.1 搜索

在分析过程中，用户会递进拓展信息，从线索关键词（例如：姓名、住址等）开始细化分析。搜索模块提供检索工具，支持对对象信息进行检索，以帮助用户快速定位信息。同时，搜索模块还可以作为图谱模块和时空分析模块的入口，将检索的对象信息引入到两个模块中进行时空关系拓展分析。

搜索模块入口如下图所示。

图 8-146: 搜索



8.2.1.2 图谱

图谱模块是I + 研判分析平台的核心模块，涵盖了大部分研判业务场景。图谱模块包含所有实体之间的关系拓扑、业务计算、可视化布局和图交互操作。同时图谱模块还通过用户空间、时序分析和信息立方三大辅助分析组件来补充关系网络研判。

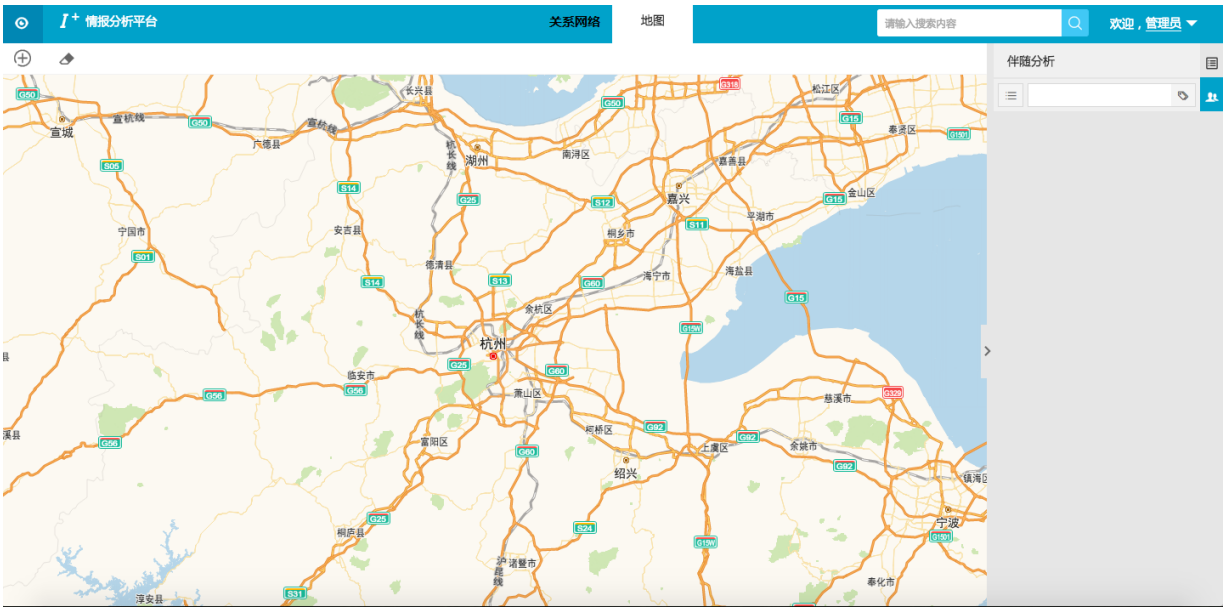
图谱界面包含**功能区**、**属性统计区**、**行为时序分析区**和**主分析图区**，具体如下图所示。

图 8-147: 图谱界面

8.2.1.3 地图分析

地图分析模块引入了空间维度，将实体和关联等数据与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 8-148: 时空分析



8.2.2 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

分析一批相同或不同类型的对象的共同联系对象。

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。

OLEP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）、事件（Event）以及对象间的关联关系（Link），以OLEP模型来组织关系数据。

8.2.3 登录关系网络分析工作台

关系网络分析平台是I+产品的数据关系分析工作台，本节介绍如何登录关系网络分析工作台。

背景信息

登录系统是产品安全的第一道防护线，所有用户都首先要取得相应的账号，然后进入到登录页面，账号密码由系统管理员分配。

> **说明**：
> 系统管理员是系统部署应用过程中指定的业务负责人员。

首次登录时，系统会提示用户修改密码，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. 登录天基，从天基查看关系网络分析的域名。
   a) 登录天基，如图 *8-149:* 天基所示。

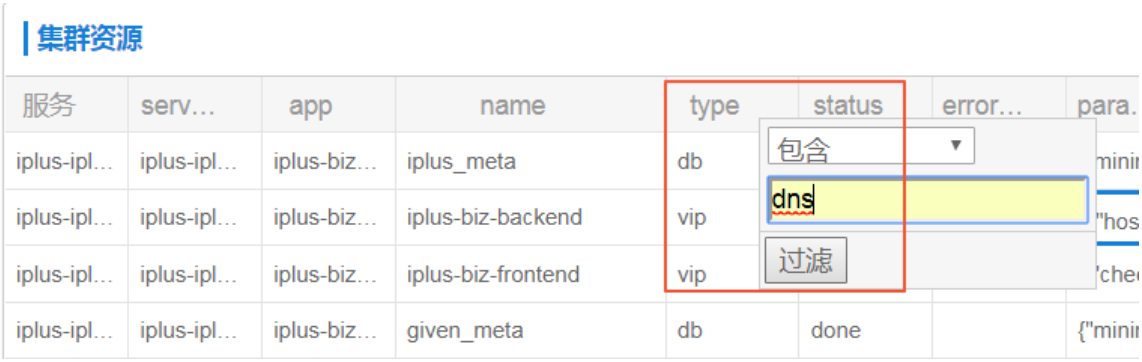
图 8-149: 天基

b) 在左侧选择**集群**，然后在**Project**下拉列表中选择iplus，筛选出I+产品。

c) 选择筛选出I+产品，将鼠标指向后面的菜单项图标并选择**Dashboard**，进入I+的**集群Dashboard**界面。

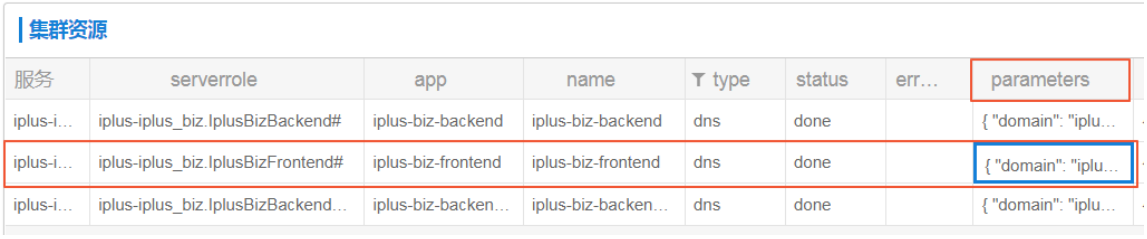
d) 在**集群资源**列表中，单击列表的**type**表头，以dns进行筛选，如图 *8-150: 筛选条件*所示。

图 8-150: 筛选条件



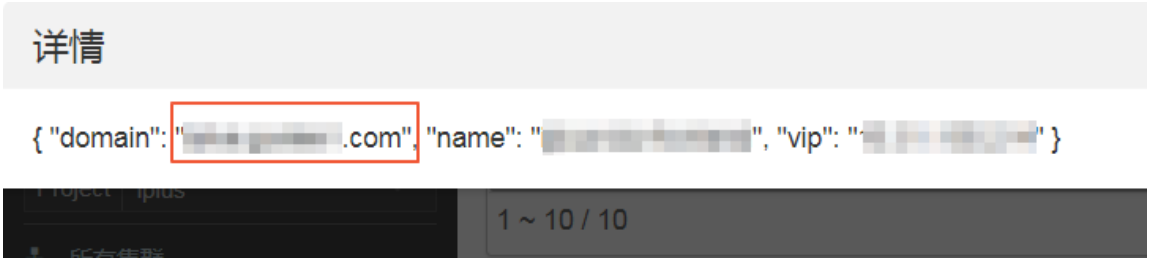
e) 在筛选结果中查找**name**为iplus-biz-frontend的行，如图 *8-151: 筛选结果*所示。

图 8-151: 筛选结果



f) 右键单击该行的**parameters**单元格，并选择**显示更多**，在弹出的**详情**信息框中查看I+关系分析的域名，如图 *8-152: 详情*所示。

图 8-152: 详情



2. 在浏览器中输入I+关系分析的域名，进入登录页面，如图 *8-153: 登录页面*所示。

图 8-153: 登录页面

3. 输入账号和密码，单击**登录**，可能出现以下情况：

在账号输入过程中，系统会自动检测输入字符的合法性。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！**；如果输入合法字符，则系统对账号和密码进行校验。

是否首次登录	操作
否	单击**登录**，可能出现以下情况：进入I+工作台页面，如图 *8-154: I+工作台*所示。 用户可通过单击**新建分析**，使用I+关系网络分析平台。
是	1. 单击**登录**，弹出**首次登录请修改密码**对话框，如图 *8-155: 首次登录修改密码*所示。 2. 输入新密码和确认密码，单击**确定**，系统自动进入图 *8-153: 登录页面*。 3. 重新以新密码登录即可。

图 8-154: I+工作台

图 8-155: 首次登录修改密码

8.2.4 导入数据

I+关系网络分析除了系统管理员在数据库中配置的数据以外，系统使用者还可以通过上传csv、txt、xls文件的方式导入数据。

8.2.4.1 数据模型

数据模型是一个用于规定上传数据的文件格式，用于表示上传文件的各列字段所对应的是对象或关系的哪个属性。

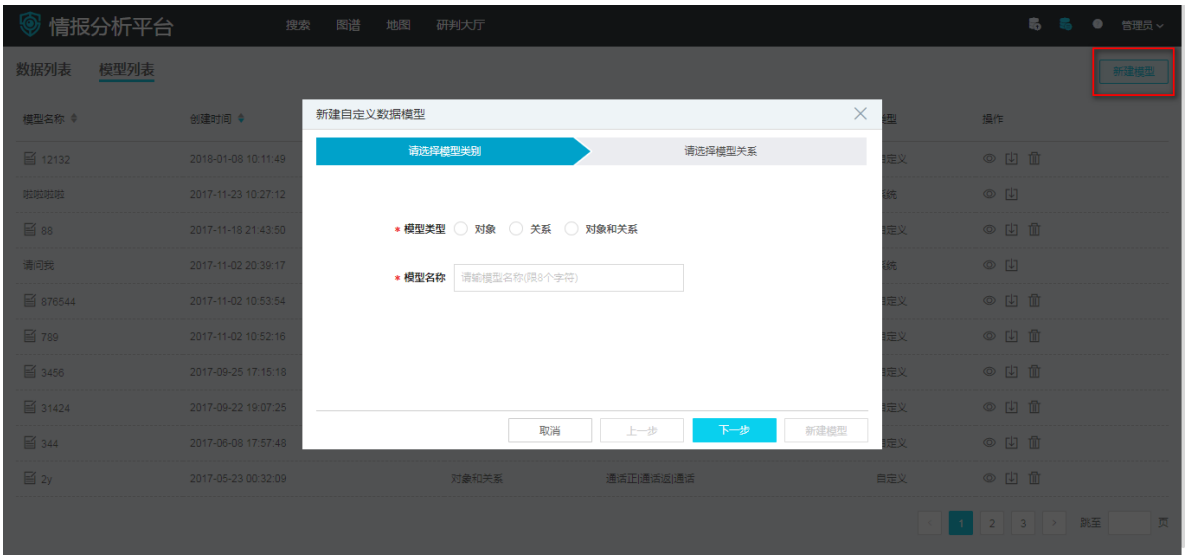
新建模型

1. 登录关系网络分析工作台。
2. 点击页面右上角的**数据导入**图标，弹出导入数据窗口，如图 *8-156:* 数据导入列表所示。

图 8-156: 数据导入列表

3. 点击切换到**模型列表**页签页面。
4. 单击页面右上角的**新建模型**按钮，弹出新建自定义数据模型窗口，如图 *8-157:* 选择模型类型所示。

图 8-157: 选择模型类型



a. 选择模型类型：对象、关系、对象和关系。
b. 输入模型名称。
c. 单击**下一步**按钮。

5. 选择新建模型要对应的一个对象或关系后，选择数据字段和字段名。

图 8-158: 选择模型关系

6. 单击**新建模型**按钮，完成新建模型。

查看数据模型

1. 点击页面右上角的**数据导入列表**图标。
2. 点击切换到**模型列表**页签页面，即可查看所有的数据模型。

图 8-159: 模型列表

- 编辑模型名称：点击模型名称前的**编辑模型名称**图标，可修改模型名称。
- 查看模型：点击操作列的**查看模型**图标，可查看模型的详细信息。

- 下载模型：点击操作列的**下载模型**图标，下载xls文件模版填入数据内容后再导入到 I+ 系统中。
- 删除模型：点击操作列的**删除模型**图标，可删除该模型。

8.2.4.2 数据导入

I+关系网络分析除了系统管理员在数据库中配置的数据以外，系统使用者还可以通过上传csv、txt、xls文件的方式导入数据。

1. 登录关系网络分析工作台。
2. 点击页面右上角的**数据导入**图标，弹出导入数据窗口，如图 *8-160: 导入数据窗口*所示。

图 8-160: 导入数据窗口

3. 单击**上传**按钮，在本地选择待导入的文件后，下方会显示文件的预览页面（仅展示文件内容的前10 行和内容总行数），如图 *8-161: 数据预览*所示。

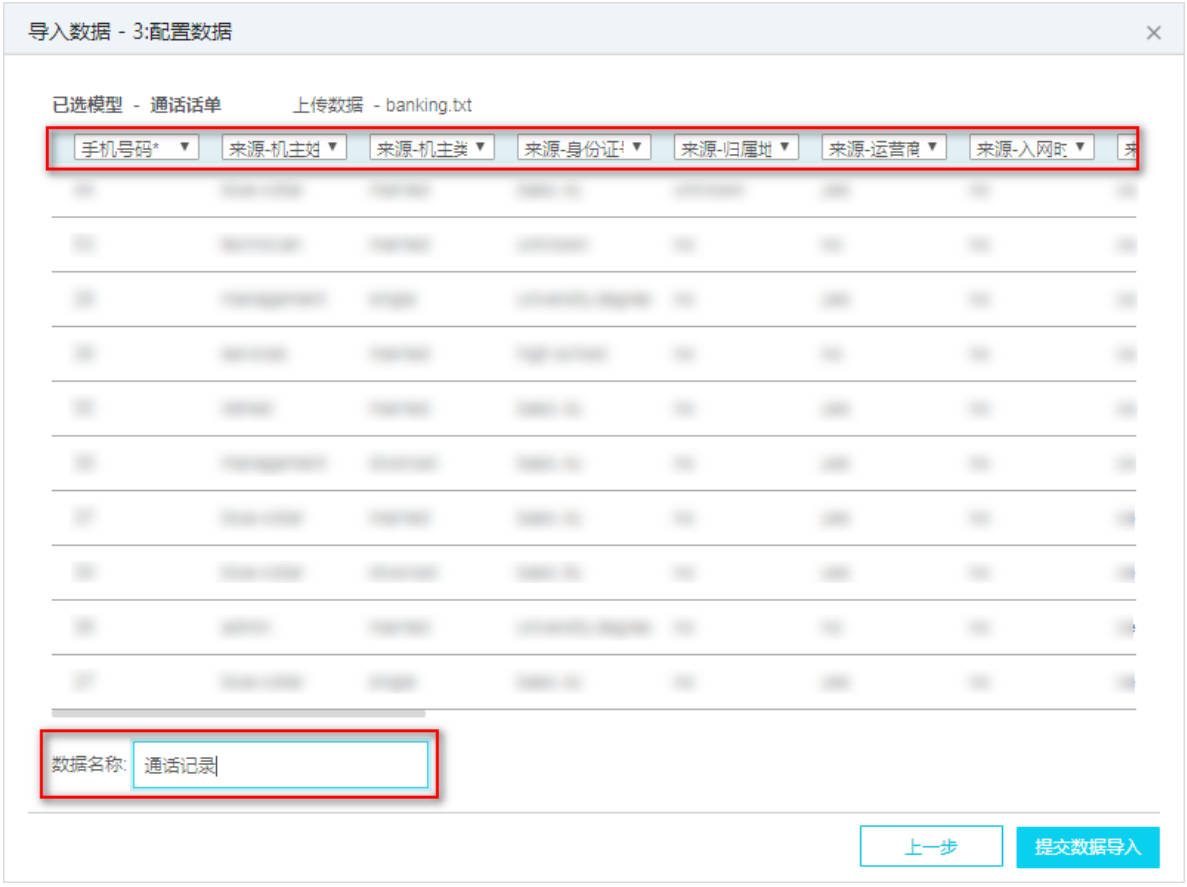
图 8-161: 数据预览

4. 配置文件内容的读取格式：

   - 字段分隔符：设置每行内容的字段间分隔符。
   - 编码方式：设置文件内容的字符编码。
   - 选择模型：设置上传数据对应的数据模型。

5. 单击**下一步**按钮，进入配置数据页面，如图 *8-162:* 配置数据页面所示。

图 8-162: 配置数据页面



a. 选择数据每列字段对应的模型属性。

b. 输入数据名称。

6. 单击**提交数据导入**按钮，将数据上传至系统中。

查看数据列表

1. 点击页面右上角的**数据导入列表**图标，默认进入数据列表页签页面。

图 8-163: 数据列表

数据列表展示了所有导入的数据表记录。

2. 可进行如下操作：

   - 编辑数据名：点击数据名称前的**编辑数据名**图标，可修改数据名称。
   - 查看数据：点击操作列的**查看数据**图标，可查看数据的详细信息。
   - 导入到图谱：点击操作列的**导入到图谱**图标，可将数据展示在图谱中。
   - 删除：点击操作列的**删除**图标，可删除该数据。

8.2.5 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。用户可以拿到一些模糊信息，在搜索模块中查询到实体与关系的记录，搜索出的结果可以作为单独对象节点添加到关系分析和GIS分析中去，作为研判起点进行分析。

搜索功能的入口，如图 *8-164:* 搜索功能入口所示。

图 8-164: 搜索功能入口

注意：

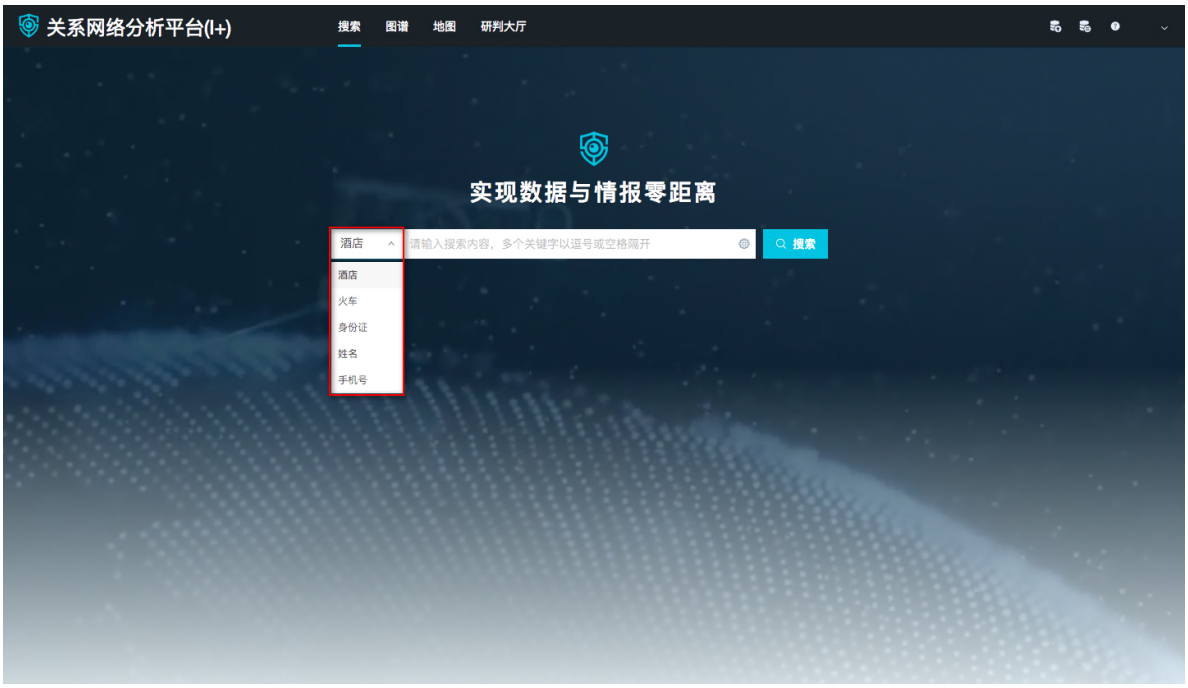
当搜索条件为空时，不能搜索出结果，提示**请至少输入一项条件值。**

8.2.5.1 简单搜索

操作步骤

1. 登录关系网络分析工作台。
2. 单击上方的**搜索**，进入**搜索**界面。
3. 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 *8-165:* 搜索框所示。

图 8-165: 搜索框



4. 在搜索框中输入关键字。

   支持模糊搜索，例如要搜索手机号中含有138的对象，则在搜索框中输入**138**即可。

5. 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 *8-166: 搜索结果*所示。

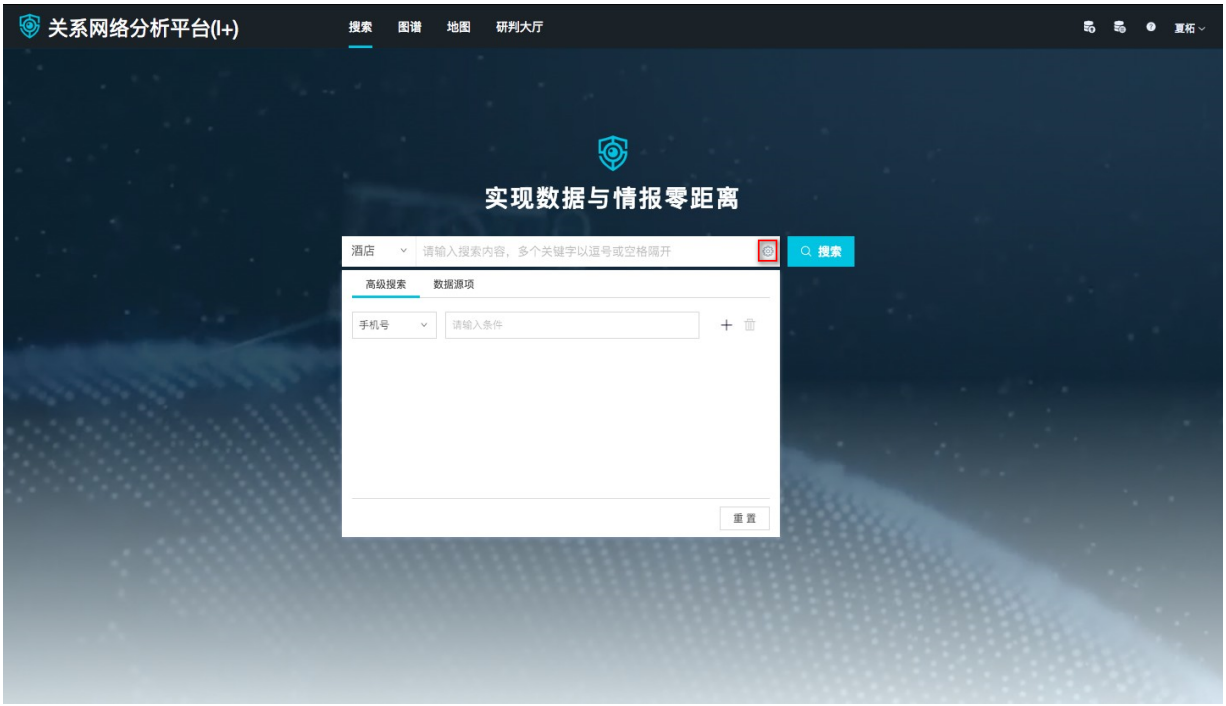
图 8-166: 搜索结果

8.2.5.2 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。

通过点击设置图标可展开高级搜索区域，高级搜索条件也可以默认不填写。如图 *8-167: 高级搜索*所示。

图 8-167: 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**liqing344**的对象，操作步骤如下：

1. 登录关系网络分析工作台。
2. 单击上方的**搜索**，进入**搜索**界面。
3. 选择关键字为**身份证**，并在搜索框中输入**234**。
4. 选择关键字为**姓名**，并在搜索框中输入**liqing344**。
5. 单击**搜索**按钮或通过点按键盘的**Enter**键即可进行搜索。

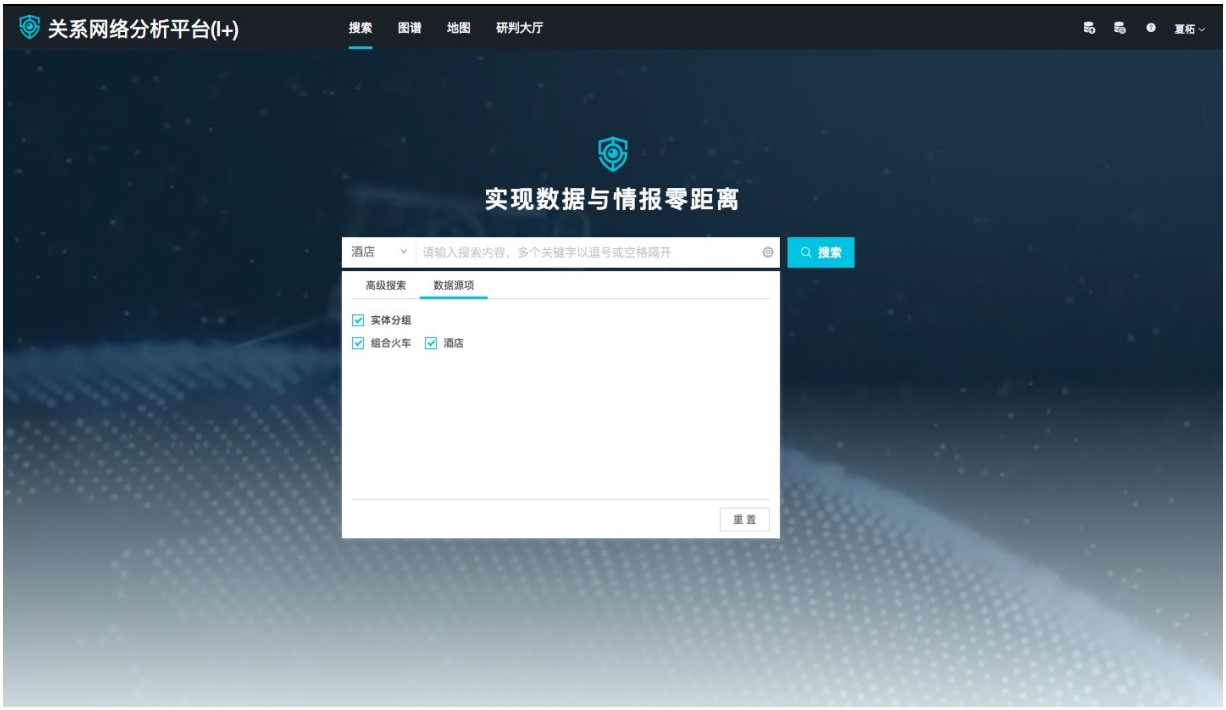
   搜索结果如图 *8-168: 多关键字高级搜索*所示。

图 8-168: 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 *8-169: 数据源高级搜索*所示。

图 8-169: 数据源高级搜索



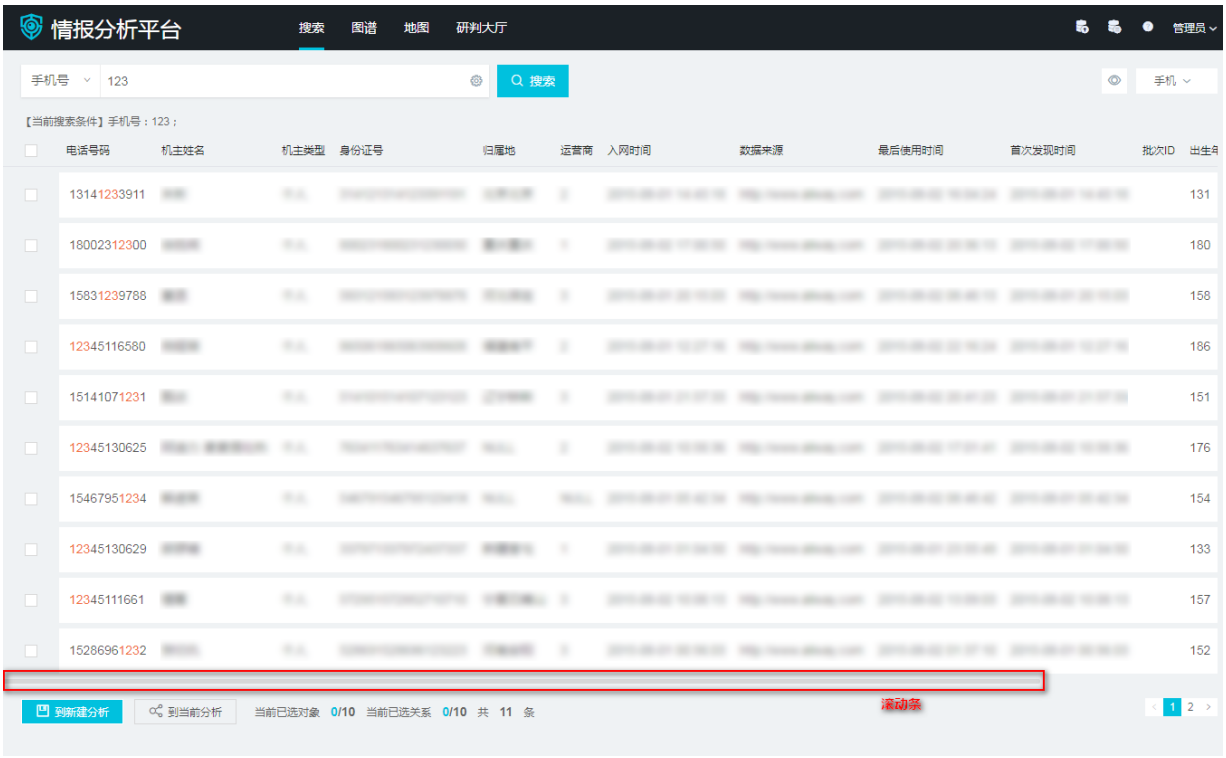
高级搜索提供了条件重置功能，您可以单击**重置**按钮清空搜索条件并重新输入。

8.2.5.3 搜索结果

搜索结果显示

搜索结果中，命中关键字的部分字体为红色。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 *8-170: 搜索结果展示*所示。

图 8-170: 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。

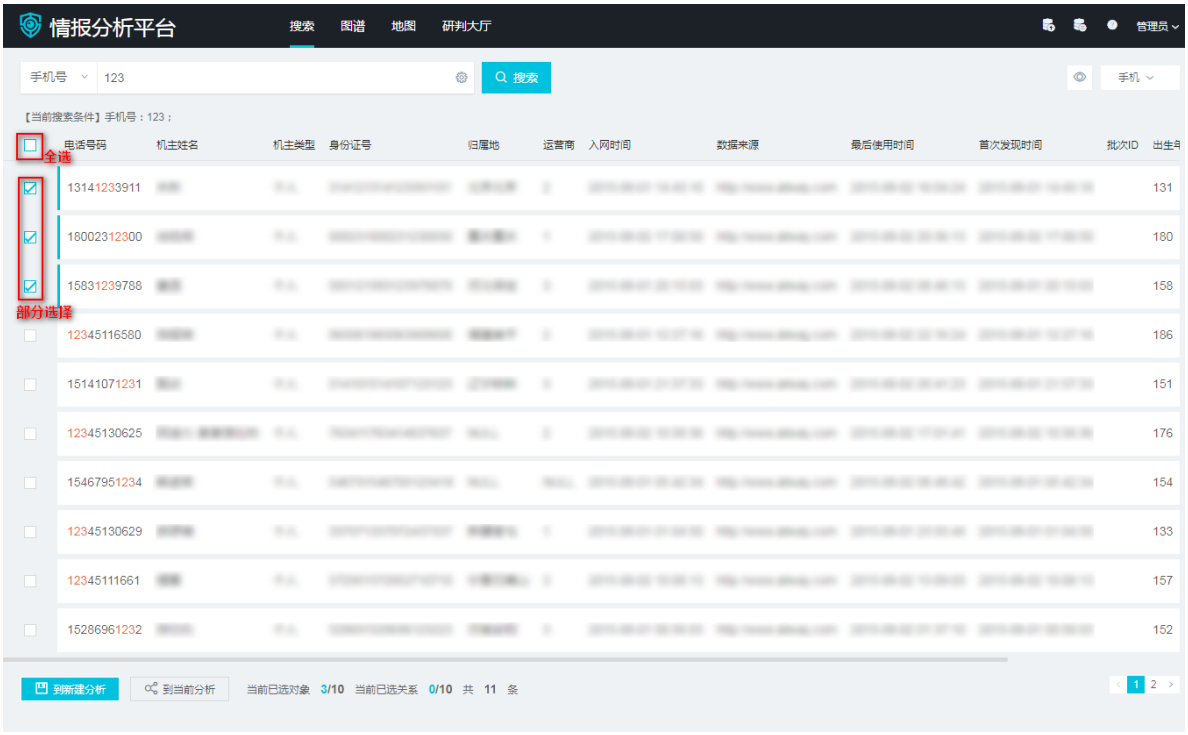
搜索结果提供对象表和关系表两种展示方式，可以点击页面右上角的筛选条件选择切换**对象数据**或**关系数据**页面。如图 *8-171: 相关信息*所示，搜索手机号，不仅能搜索出手机信息表，还能搜索出与手机号相关的事件信息，如短息、通话等。

图 8-171: 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 *8-172: 选择搜索结果*所示。

图 8-172: 选择搜索结果



2. 点击页面右上角**只看已选数据**图标，可以查看已选中的对象信息，如图 *8-173: 只看已选数据*所示。

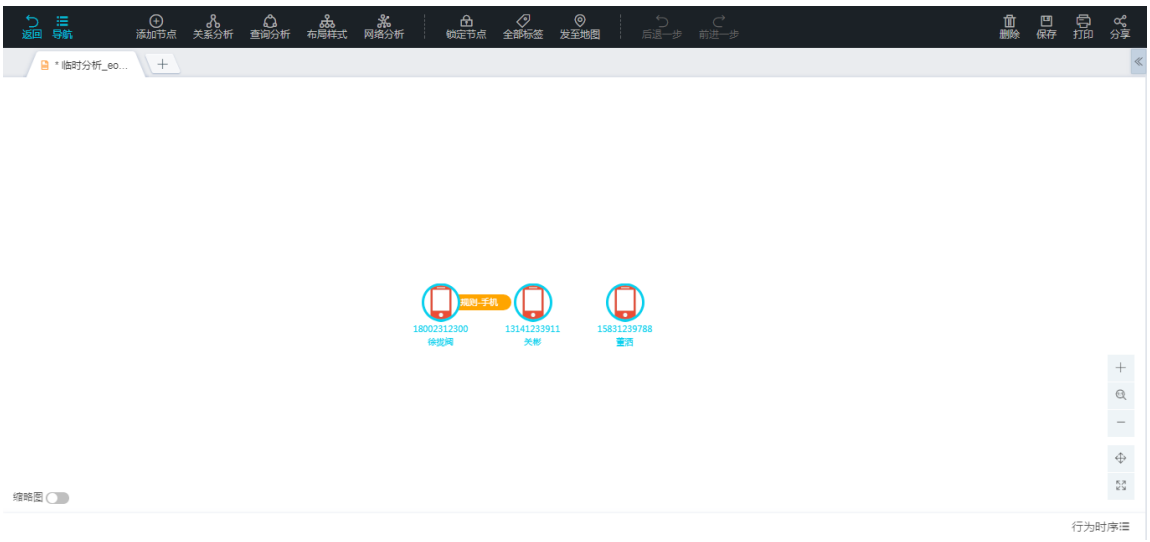
图 8-173: 只看已选数据

3. 单击**到新建分析**按钮或**到当前分析**按钮，如图 *8-174: 搜索结果分析*所示。

图 8-174: 搜索结果分析

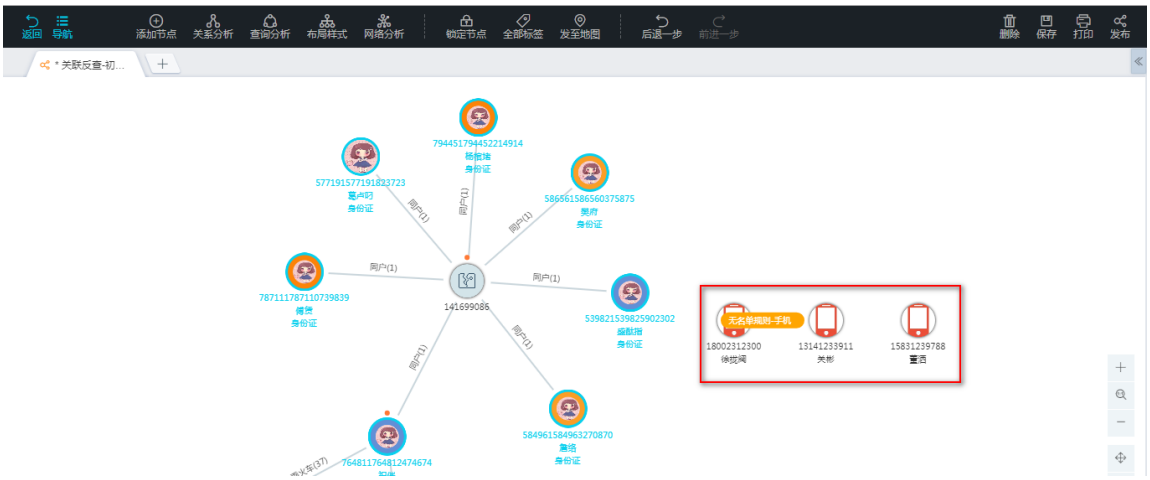
- 跳转到新建分析，如图 *8-175: 新建分析*所示。

图 8-175: 新建分析



- 跳转到当前分析，如图 *8-176: 当前分析*所示。

图 8-176: 当前分析



8.2.6 图谱

8.2.6.1 分析类型

I+支持四种分析类型：临时分析、普通分析、分享文件分析和上传导入的分析。

- **临时分析**

  用户刚刚新建的分析，即为临时分析，临时分析的操作和功能和其他类型的分析是一样的，只是在保存之前无法分享。

- **普通分析**

  普通分析是新建分析保存之后再打开的分析，是I+中最常用的分析类型。

- **分享文件分析**

  - **分享文件-初始文件**

    当普通分析分享出去之后，即成为了分享文件，每一个分享文件都会生成一个和当时分享文件内容一致的文件作为初始文件。

  - **分享文件-历史分析（草稿）**

    当被共享用户编辑并保存初始文件后，会生成该分享文件的一个历史分析（又名草稿分析）。

  - **分享文件-合并文件**

    系统会将多个历史分析自动合并成一个分析，用户也可以手动进行合并。

- **上传导入的分析**

从上传导入模块打开的分析为上传导入分析，上传导入分析无法保存。

8.2.6.2 新建分析

前提条件

已创建源数据表、对象信息和关系信息。

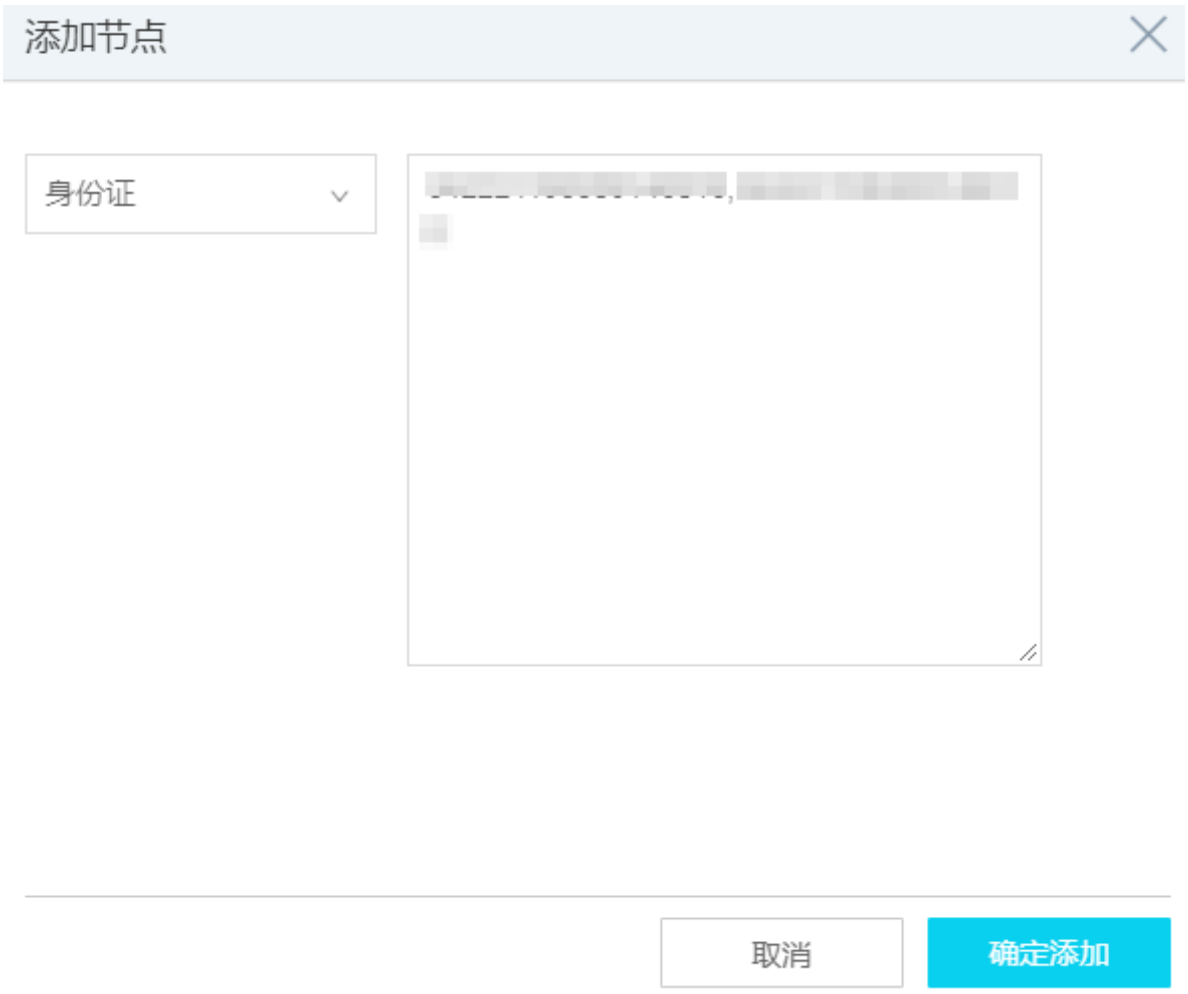
操作步骤

1. 登录关系网络分析工作台。

图 8-177: I+工作台

2. 单击**新建分析**图标，弹出**临时分析**页面。
3. 单击**添加节点**并单击空白处，或直接右键单击空白处，弹出**添加节点**对话框，如下图所示。

图 8-178: 添加节点

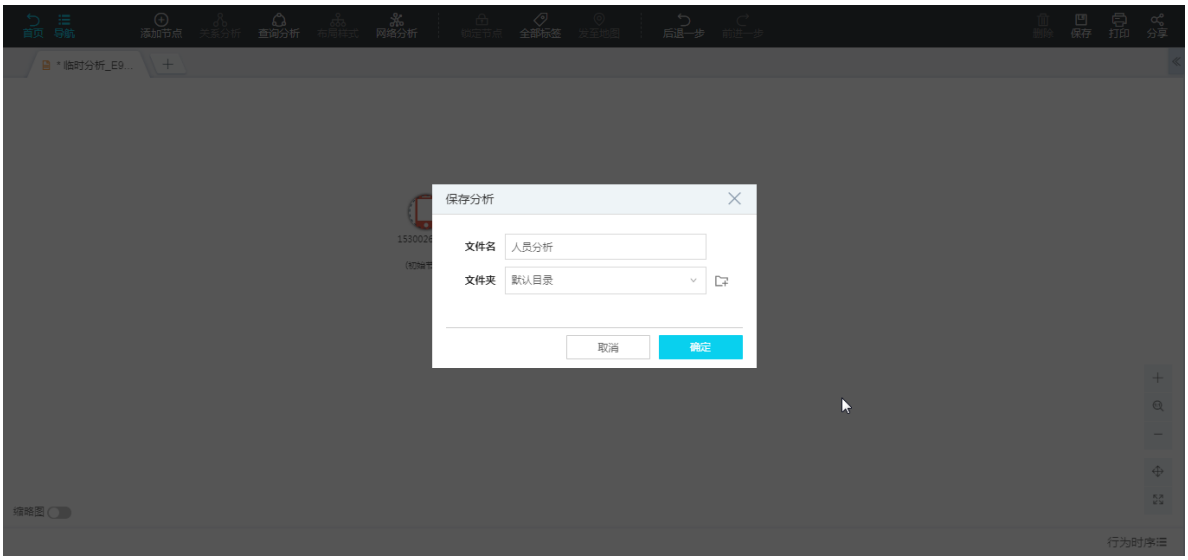


4. 根据实际需要设置左侧下拉列表参数和右侧文本框参数，参数说明如下表所示。

参数名称	描述
下拉列表	下拉列表框中显示所有已创建的**对象信息**，根据实际需要选择一个即可。
文本框	填写**对象信息**对应的数据表中的一个或多个主键值。 填写多个主键值时，每个值间需要英文半角逗号（,）隔开。

5. 单击右上角的**保存**图标，在弹出的**保存分析**窗口中输入文件名，选择文件目录，如下图所示。

图 8-179: 保存分析



6. 单击**确定**按钮，完成新建分析操作。

8.2.6.3 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。

8.2.6.3.1 图区功能操作

8.2.6.3.1.1 添加节点

I+提供两种添加节点的方式：从工具栏添加节点和从右键菜单中添加节点。

添加节点方式

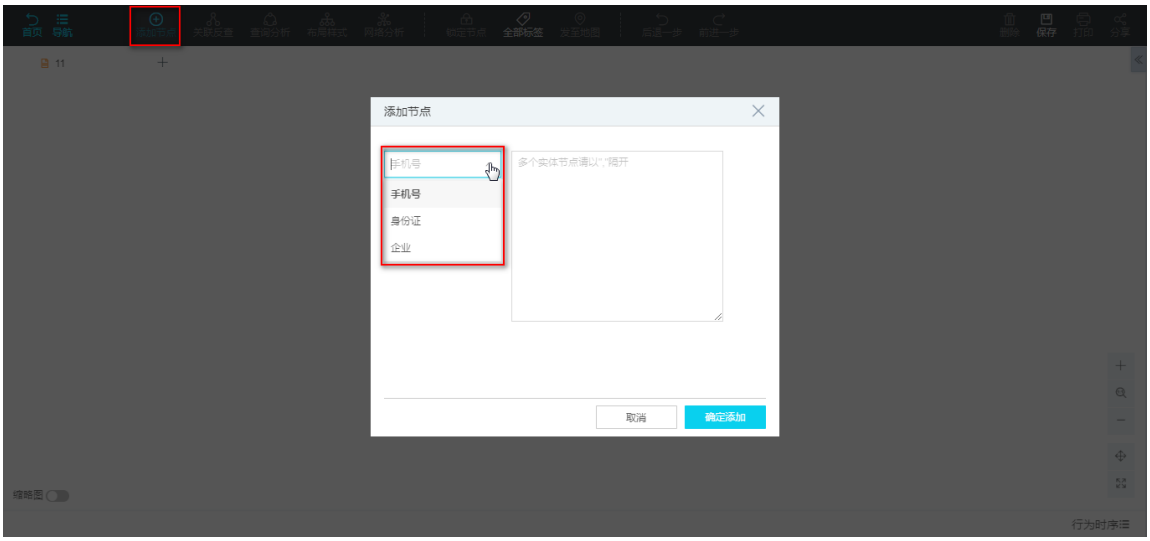
- **从工具栏上添加节点**

  适合针对特定位置添加节点的场景。

  1. 点击 

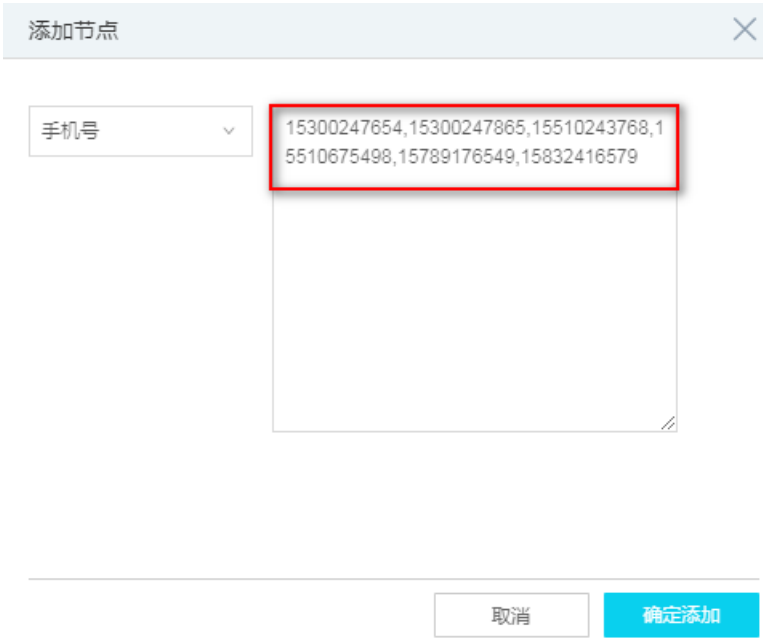
，画布进入到添加节点模式，此时鼠标的图标变成添加节点的图标，工具栏上的添加节点按钮变成按下选中状态。
  2. 在图区空白处单击并选择实体类型，如图 *8-180: 添加节点按钮*所示。

图 8-180: 添加节点按钮



3. 输入具体内容，多个实体节点之间用英文逗号（，）隔开，如图 *8-181: 输入节点信息*所示。

图 8-181: 输入节点信息



4. 单击**确定**按钮，添加节点成功，鼠标点击的位置即为新添加节点的参考位置。

- **从右键菜单中添加节点**

  适合针对特定位置添加节点的场景。

  1. 在页面空白处单击鼠标右键，在弹出的菜单列表中选择**添加节点**项，如图 *8-182: 右键菜单中添加节点*所示。

单击鼠标右键后，画布不会进入添加节点模式，而是直接触发添加节点对话框。

图 8-182: 右键菜单中添加节点

2. 输入节点具体内容，多个节点之间用英文逗号（,）隔开。
3. 单击**确定**按钮，添加节点成功，鼠标右键的位置即为新添加节点的参考位置。

添加节点表单

I+支持添加复合类型的对象节点，例如人这个对象类型可以配置身份证和护照两个子类型（即通过身份证或护照均可确定唯一的一个人），来作为人的复合类型出现。

1. 选择需要添加节点的类型，如图 *8-183:* 添加复合类型的对象节点所示。

图 8-183: 添加复合类型的对象节点

2. 在右侧的文本框中输入需要添加的节点内容，如果有多个节点可以用英文逗号（,），或者换行隔开。
3. 单击**确定**按钮后，I+立即去数据库中搜索需要添加的节点的详情，并返回结果。

添加节点布局

添加节点成功后，I+会对添加的节点进行布局：

- 如果只添加一个节点，则直接放置在鼠标单击或右键的参考位置处。
- 如果添加多个节点，则会根据I+系统后台对添加节点布局的配置，进行直线或者矩阵布局。

8.2.6.3.1.2 移动画布

用户可通过移动画布对分析图区进行整体移动，方便比对分析。

移动画布的方式有三种：

- 点击图区右下角的**进入移动画布模式**图标，如图 *8-184:* 画布拖动模式所示，鼠标拖动画布。

图 8-184: 画布拖动模式

- 长按空格键，同时鼠标拖动画布。
- 鼠标滚轮上下移动画布。

说明：

移动画布之后，画布会进行重绘，需要耗费一个短暂的重绘时间，在此期间请尽量不要操作画布，以免发生一些意外的情况。

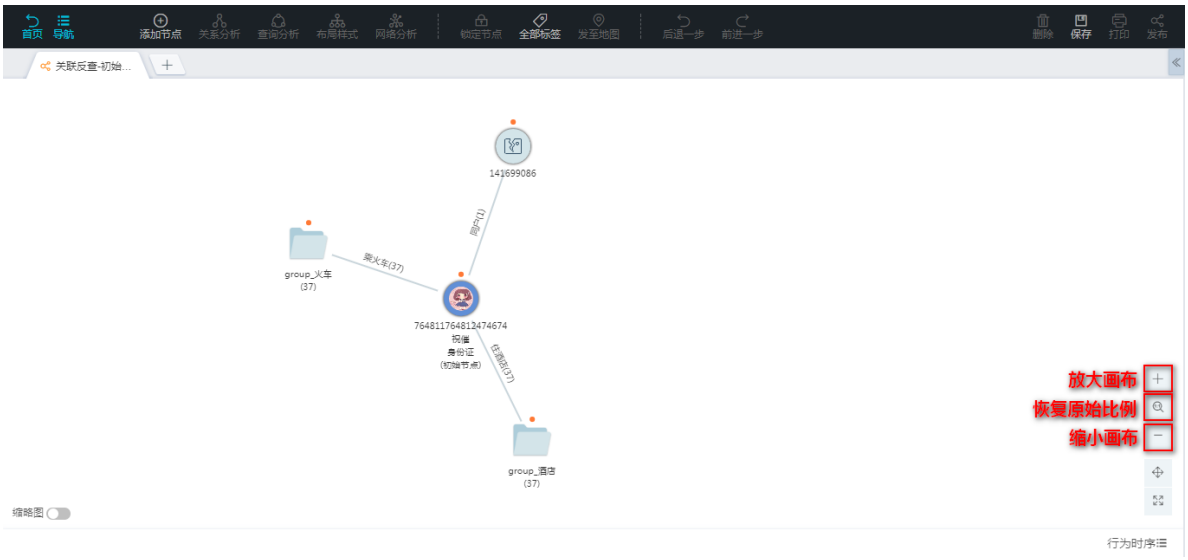
8.2.6.3.1.3 缩放画布

I+支持对分析图区的内容进行缩放操作，方便用户从整体或局部来进行对比分析。

缩放画布的方式有两种：

- 按住**Shift**键后通过转动鼠标滚轮进行画布的缩放。
- 缩放画布按钮如图 *8-185:* 缩放画布按钮所示。

图 8-185: 缩放画布按钮



- 点击**放大画布**图标，放大画布。
- 点击**恢复原始比例**图标，画布恢复正常显示比例。
- 点击**放大画布**图标，放大画布。

说明：

缩放画布之后，画布会进行重绘，需要耗费一个短暂的重绘时间，在此期间请尽量不要操作画布，以免发生一些意外的情况。

8.2.6.3.1.4 锁定/解锁节点

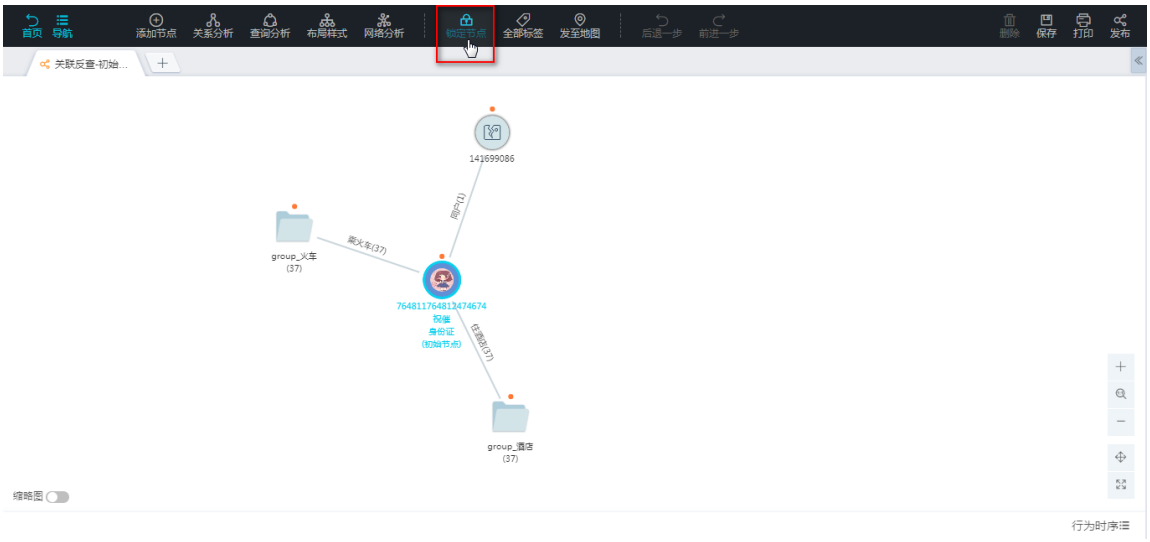
锁定节点功能可以使节点位置在空间画布中不变，方便用户操作。

锁定节点

- **锁定一个或多个节点**

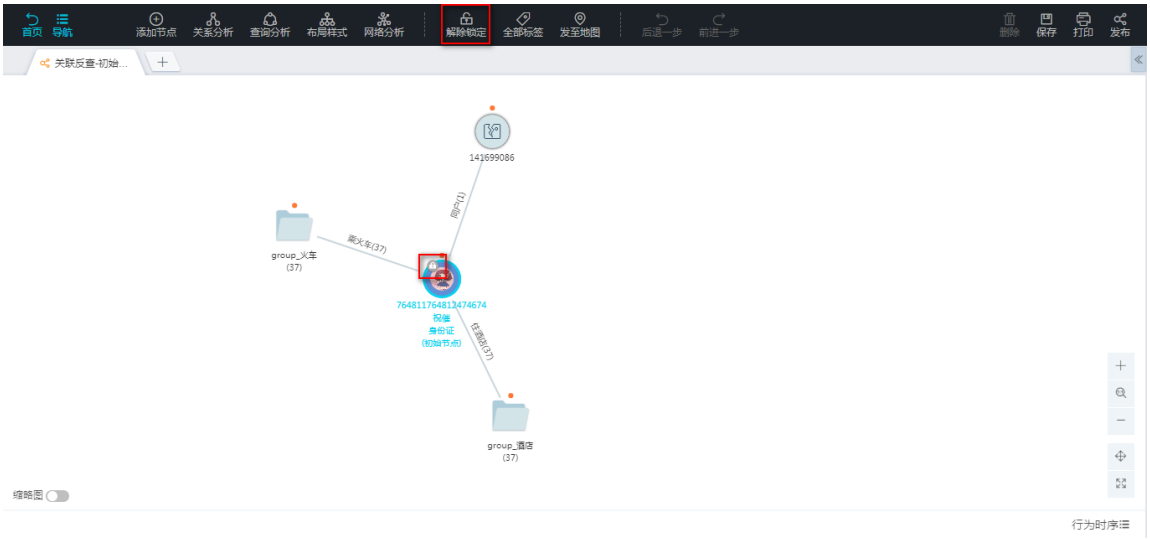
1. 在画布中，选中一个或多个节点。
2. 点击**锁定节点**图标，如图 *8-186: 节点锁定*所示。

图 8-186: 节点锁定



已锁定节点的左上方会标示有一把灰色的小锁，如图 *8-187: 节点锁定效果*所示，锁定后的节点无法被拖动，除了弹性力布局外，其他的布局不会引起锁定节点位置的变化（除了画布整体移动外）。

图 8-187: 节点锁定效果



- **锁定合并节点**

  如图 *8-188: 合并节点锁定*所示，选中合并节点后，点击**锁定节点**图标，锁定合并节点。

图 8-188: 合并节点锁定

解锁节点

节点解除锁定后左上方的灰色小锁标记会消失，节点可以随意移动位置。

- **解锁一个或多个节点**

  选中一个或多个的节点：

  - 如果选中的节点都是已经锁定的节点，则工具栏会显示**解除锁定**按钮，点击后即可解锁节点。
  - 如果选中的节点中存在未锁定节点，则工具栏上仍显示**锁定节点**，点击后会将没有锁定的节点锁定，已经锁定的节点不变，还是锁定状态。此时再点击**解除锁定**按钮，可将选中的所有已锁定节点解锁。

- **解锁合并节点**

  如图 *8-189: 拆分合并节点*所示，有两种方式解锁合并节点：

  - 选中已锁定的合并节点，点击**解除锁定**图标后，即可解锁节点。
  - 如果合并节点被拆分，则节点自动解锁。

图 8-189: 拆分合并节点

8.2.6.3.1.5 标签

I+关系网络分析图区支持用户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。

8.2.6.3.1.5.1 标签类型

I+支持为每一个节点对象打标签功能，标签分为如下类型：

- **系统标签**：系统根据算法等为节点对象添加的标签。

  系统标签显示在节点的左侧，且系统标签不支持删除、修改和赞。

- **用户标签**：用户为节点对象添加的标签。

  用户标签显示在节点的右侧，且符合权限的用户标签是可以被相应用户删除和赞。

8.2.6.3.1.5.2 用户标签

用户标签具有四种权限类型，不同类型的标签在可视范围和操作上有所区别。

用户标签类型

- **全局标签（公开）**

  在分析被分享之后，所有人都能看到的标签，且所有人都可以对这个标签进行点赞操作。

- **仅限本人**

  该标签只有标签的添加者能看到，分析被分享出去后，非添加者本人是无法看到这个标签的。

- **仅限本分析**

该标签只在当前分析中可见，可赞，但是节点被复制到其他分析后，节点的标签将不可见。

- **指定用户可见**

  该标签在分享之后只有分享时指定的用户才能看到和点赞。

用户标签颜色

为了区分不同的标签，用户标签在添加时可以选择不同的颜色，系统默认提供了红、黄、蓝、绿四种可选的颜色，如图 *8-190: 用户标签颜色*所示，方便用户标识。

图 8-190: 用户标签颜色

操作用户标签

- **删除标签**

  删除标签操作仅限于添加标签的用户执行，如图 *8-191: 删除标签*所示，分为如下情况：

  - 标签类型为仅限本分析时，在标签操作区域会显示删除图标，点击该图标可删除标签。
  - 标签类型为公开、仅限本人、指定可见时，可操作区域显示数字为1，鼠标悬浮上去出现取消赞的图标，则点击该图标即可删除标签。

图 8-191: 删除标签

- **赞/取消赞**

  用户操作被分享的分析时，可对标签进行赞/取消赞操作。当标签可操作区域显示数字，鼠标悬浮上去显示赞的图标，点击该图标即可完成赞的操作，标签的数字会加1，如图 *8-192: 赞*所示。

  图 8-192: 赞

  赞过的标签，鼠标再次放至标签可操作区域上，点击取消赞的图标后，即可完成取消赞操作。

8.2.6.3.1.5.3 添加用户标签

合并节点不支持添加标签。

1. 有如下两种方式触发添加标签窗口：
   - 选中一个或多个节点，单击右键，在弹出的下拉菜单中选择**添加标签**项，弹出**添加文字标签**窗口。
   - 从页面右侧面板中，在下方列表中选中一个或多个节点后，点击面板上方的**文字标注**图标，弹出**添加文字标签**窗口。
2. 配置如下参数：
   - 标签名称：不能超过20个字符。
   - 标签颜色：选择标签添加后显示的颜色。
   - 标签类型：选择标签的可见范围。

     当选择标签类型为**指定可见**时，如图 *8-193: 指定标签可见范围*所示，选择该标签允许查看的人员即可。

图 8-193: 指定标签可见范围

- 单击**确定**按钮，标签添加成功。

  用户自己添加的标签则会默认显示为**1**，鼠标浮动到标签上会显示**取消赞**的图标，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标签上，会显示**赞**的图标：认可则点赞，同时数字 + 1，用户对标签进行赞操作后，还可取消赞。

8.2.6.3.1.5.4 查看标签

I+支持分类查看各标签类型功能，方便用户分析对象节点。

- **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 *8-194:* 标签开关所示。

- **我的标签**

  仅显示该用户自己添加的标签。

在图区工具栏上，选择**我的标签**，如图 *8-194: 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 *8-194: 标签开关*所示。

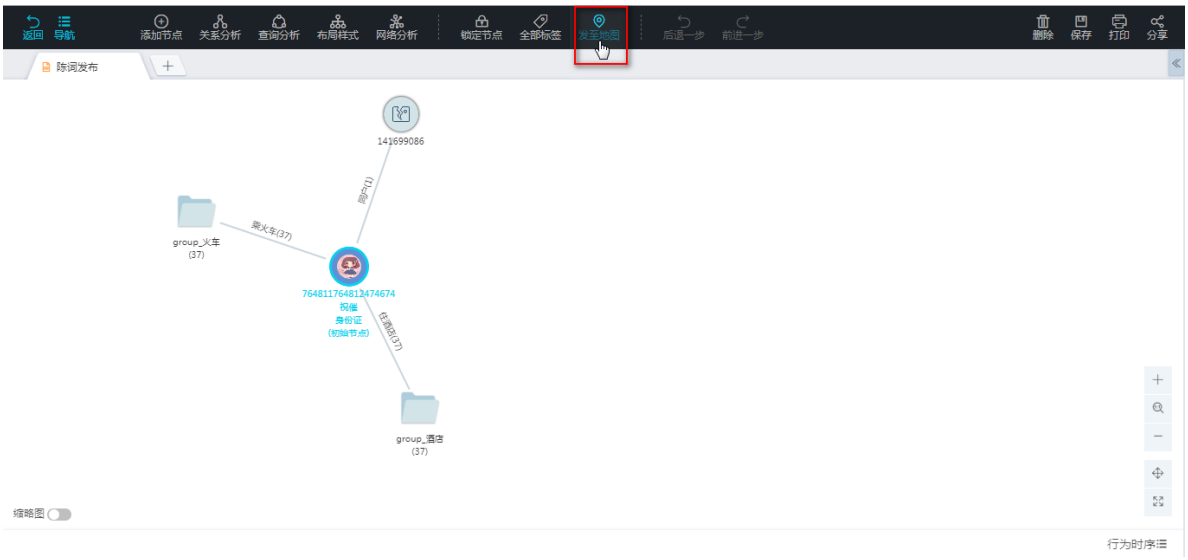
图 8-194: 标签开关

8.2.6.3.1.6 发至地图

图区和地图的节点可以来回跳转，可以将图谱的关系网络中的节点发送至地图中进行分析。

1. 在图区选中一个或多个对象节点。
2. 在工具栏选择**发至地图**按钮，如图 *8-195: 发至地图*所示，即可把图区的关系图导入到地图中去。

图 8-195: 发至地图



说明：

导入到图区的节点必须具有经纬度属性，不具备经纬度的属性会在导入到地图的时候被过滤掉。

8.2.6.3.1.7 撤销/回退操作

在图区进行操作例如添加节点、删除节点等会引起节点或者边数据增删的操作，I+会把操作之前的图区状态放入操作历史中暂存起来，当需要回退到上个历史版本的时候，可对图区进行回退操作，回退后如果想要恢复操作结果也可进行撤销操作，支持回退20步。

目前I+对以下操作支持了历史保存的操作：添加节点、添加边、保存虚拟节点、合并/拆分节点、布局相关操作、关系分析的所有相关功能、删除节点或者边、边下钻、粘贴操作。

撤销

在工具栏上单击**后退一步**按钮，对图区节点进行撤销操作，如图 *8-196:* 撤销所示。

图 8-196: 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**前进一步**按钮，可回退到撤销前的状态，如图 *8-197:* 回退所示。

图 8-197: 回退

8.2.6.3.1.8 查看缩略图

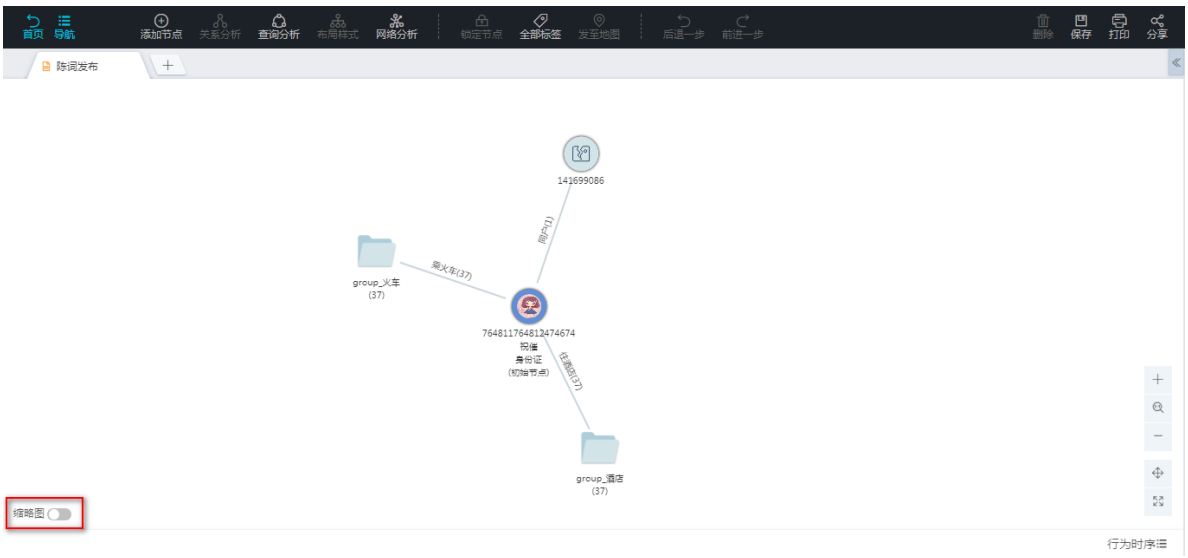
I+图区的缩略图在左下角，点开缩略图后会将当前分析全图的缩略图绘制在其上，并用矩形框标示出可视区域所在的位置，方便用户浏览分析图。

背景信息

操作步骤

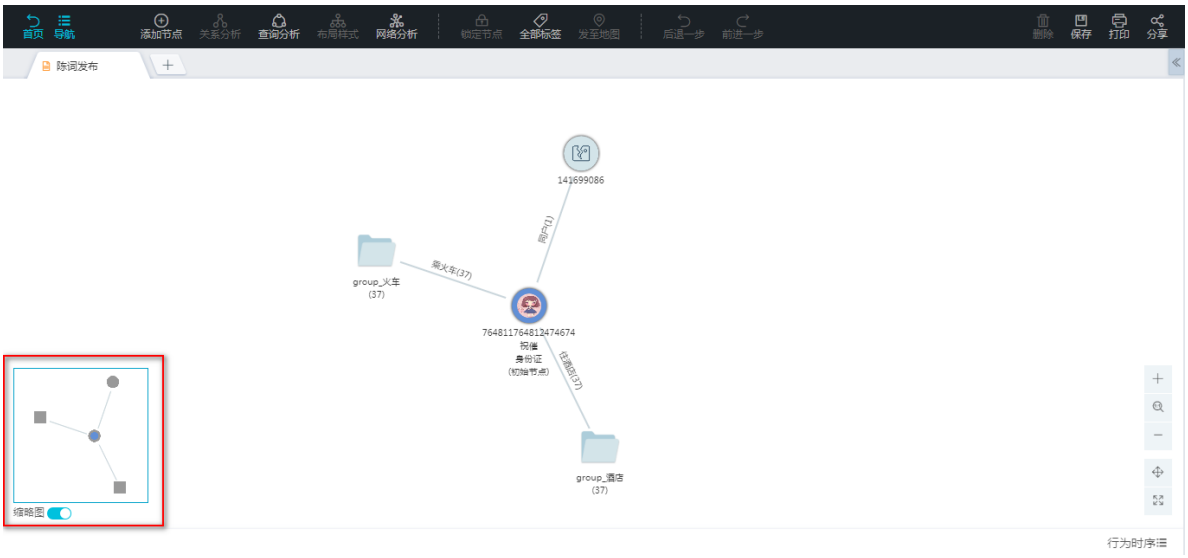
1. 单击左下角的缩略图开关，如图 *8-198: 缩略图开关*所示。

图 8-198: 缩略图开关



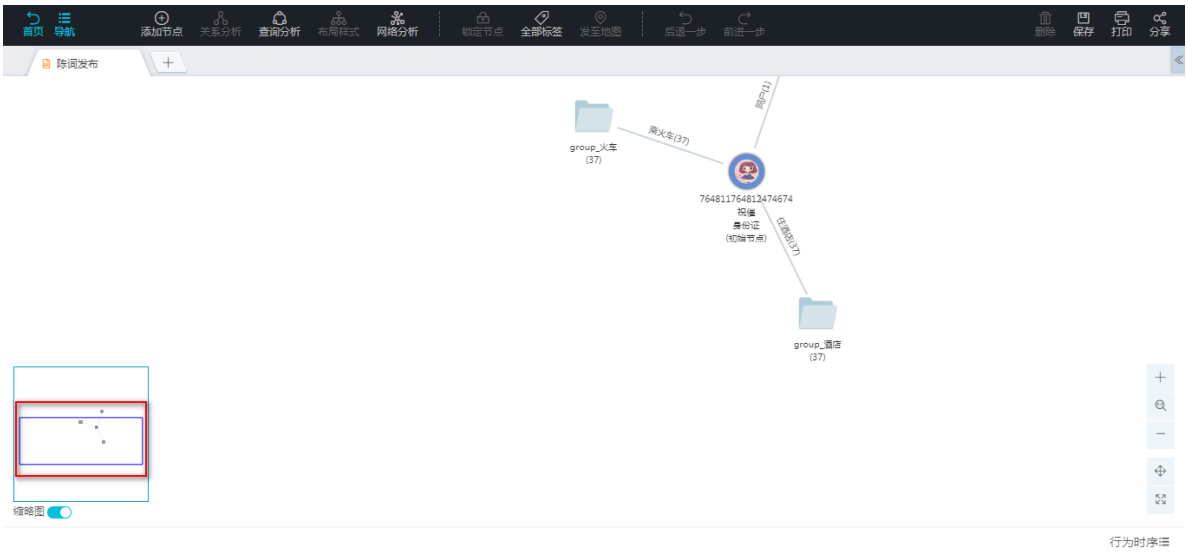
显示的缩略图如图 *8-199: 显示缩略图*所示。

图 8-199: 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 *8-200: 拖动缩略图*所示。

图 8-200: 拖动缩略图



8.2.6.3.1.9 删除已选

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。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。

1. 选中需要删除的节点或边。
2. 有如下方式进行删除操作：

   - 点击图谱工具栏上的**删除**图标。
   - 单击鼠标右键，在弹出的下拉列表中选择**删除已选**项。
   - 从页面右侧面板中，在下方列表中选中一个或多个节点后，点击面板上方的**删除**图标。

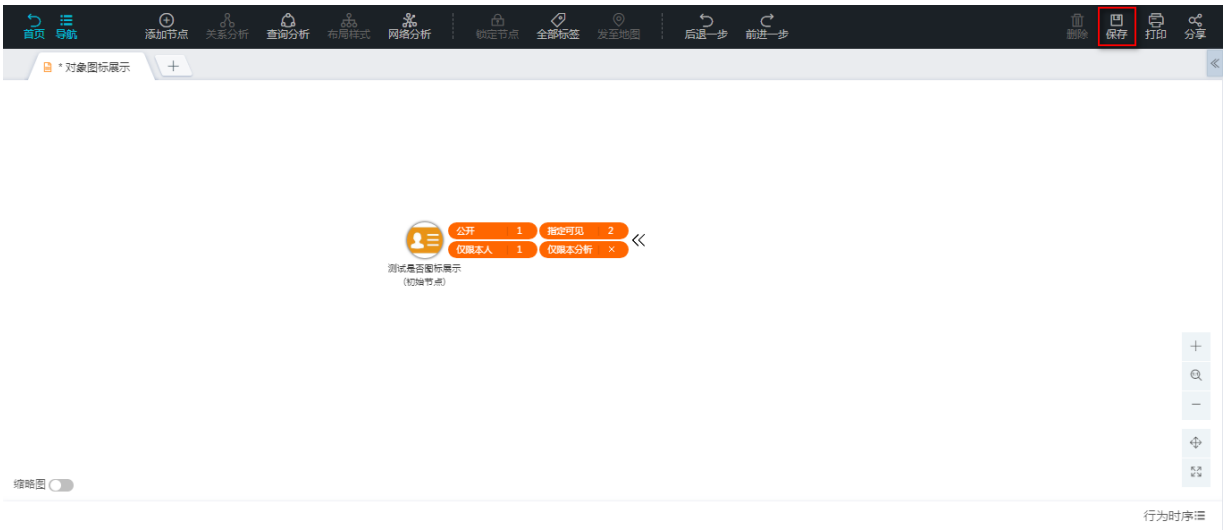
在删除节点时，和节点相连的边也会被删除。

8.2.6.3.1.10 保存分析

用户对分析进行编辑后，在关闭该分析时，需要保存所做的修改。I+提供分析截图的功能，在保存的时候I+会为每一幅图生成一个全局视角的截图存到服务器。

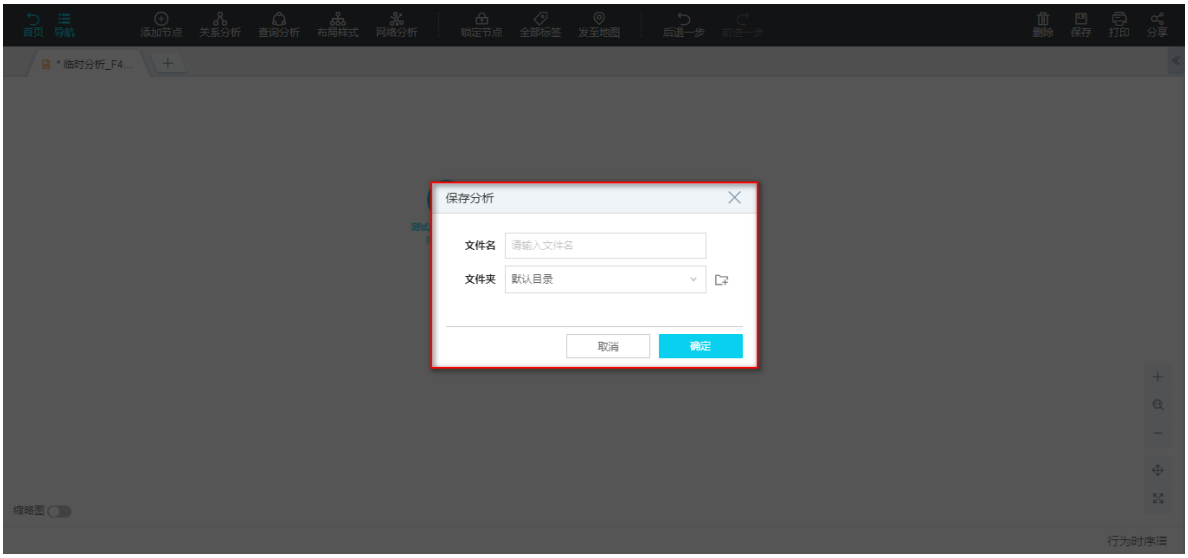
点击工具栏的保存按钮，如图 *8-201: 保存按钮*所示，保存图区分析内容。

图 8-201: 保存按钮



- 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。
- 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 *8-202: 保存临时分析*所示。

图 8-202: 保存临时分析



8.2.6.3.1.11 打印图区

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。

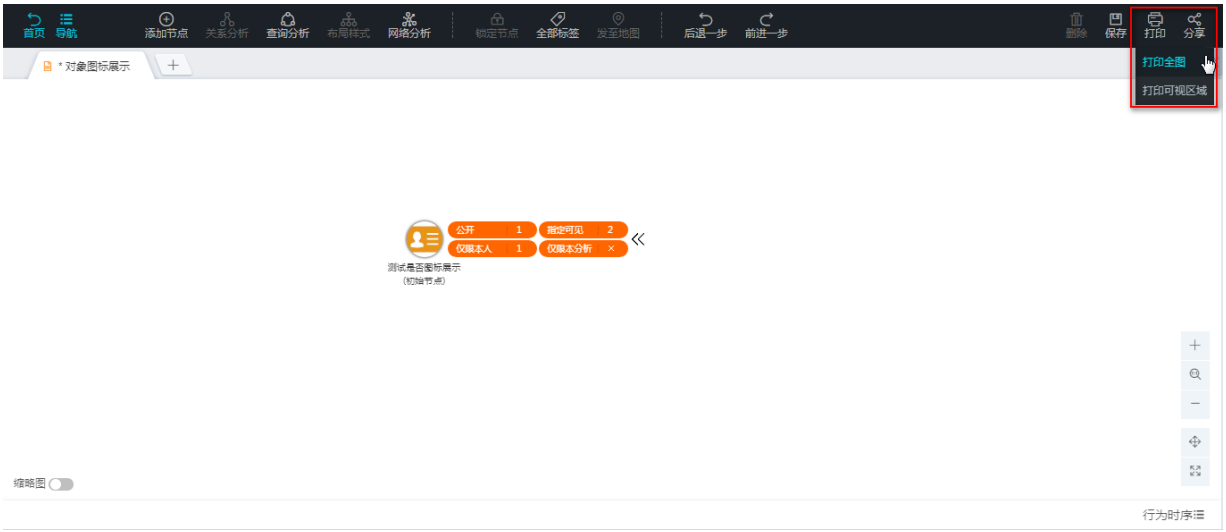
在工具栏上，选择**打印全图**，如图 *8-203: 打印图区*所示。

可视区域打印

将关系网络可视部分打印。

在工具栏上，选择**打印可视区域**，如图 *8-203: 打印图区*所示。

图 8-203: 打印图区



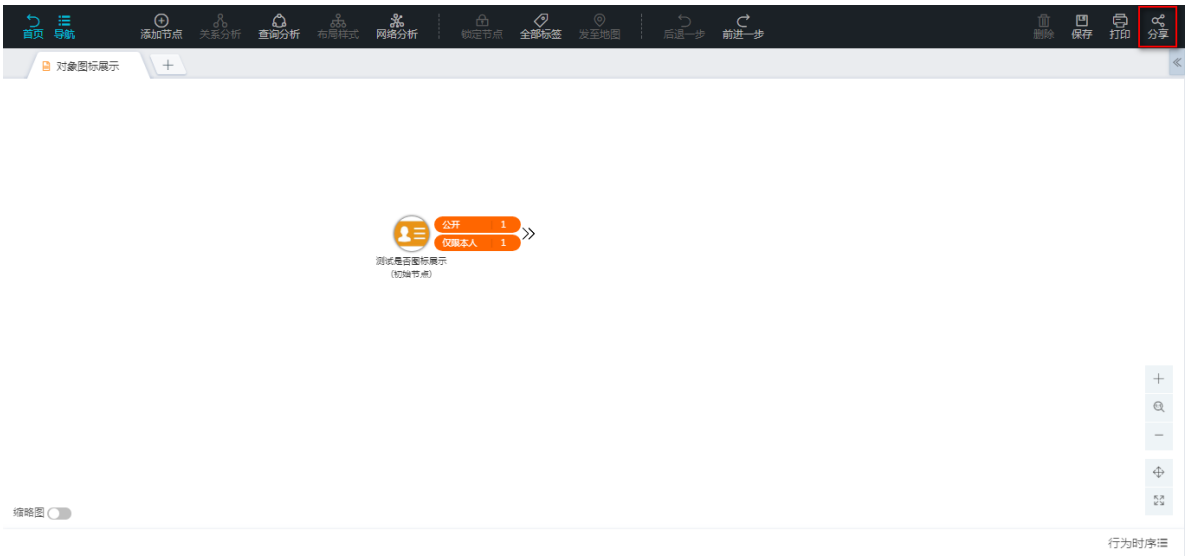
8.2.6.3.1.12 分享分析

I+支持将当前的分析分享给指定的用户或者群组，被分享的用户登录I+后即可看到共享的分析文件。

共享者分享分析文件

1. 点击图谱工具栏上的**分享**图标，如图 *8-204: 分享按钮*所示，弹出共享分析文件窗口。

图 8-204: 分享按钮



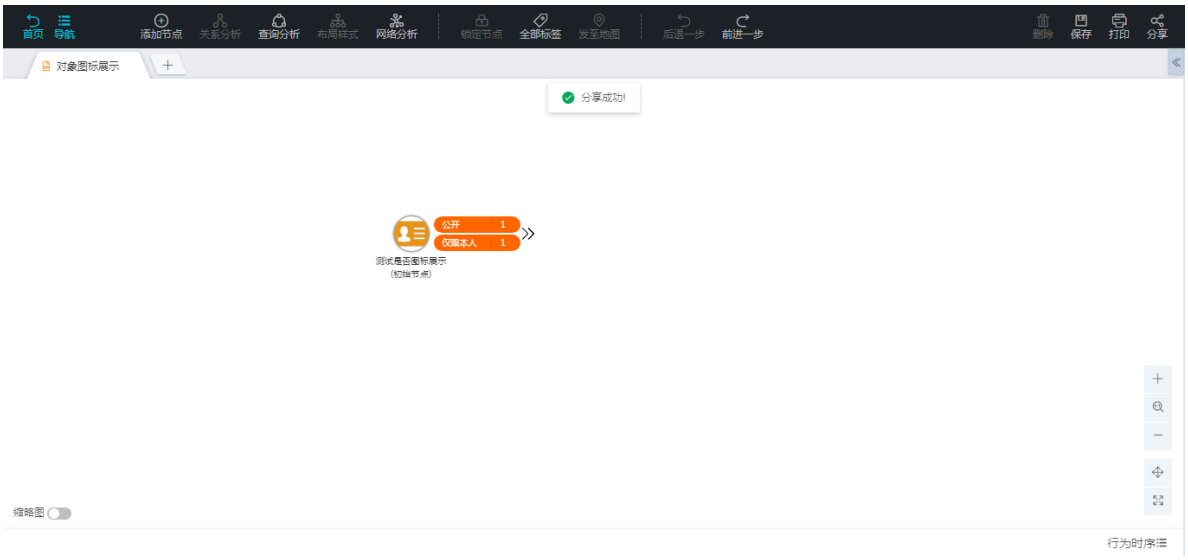
2. 选择共享的成员，共享成员的选择可以按照成员个人来选，支持搜索定位；也可以按照部门来选择。

   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 *8-205: 删除共享成员*所示。

图 8-205: 删除共享成员

3. 单击**分享**按钮，分析分享成功，如图 *8-206: 分析分享成功*所示。

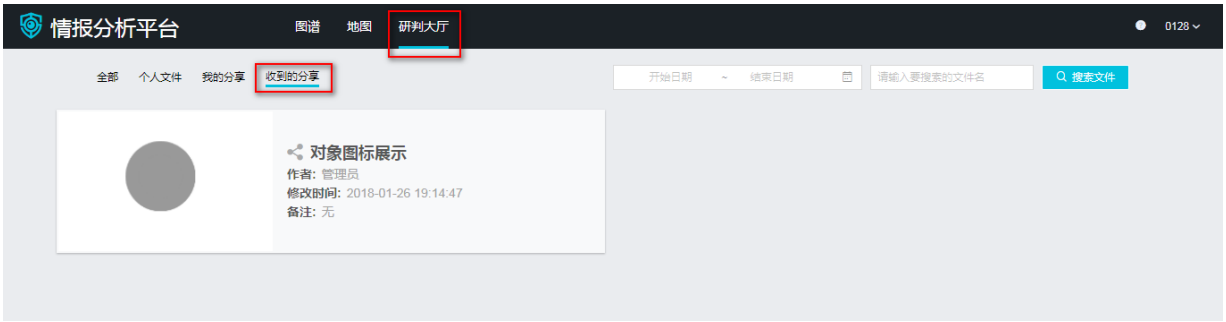
图 8-206: 分析分享成功



被分享者查看共享的分析文件

当被分享成员登录I+后，可在**研判大厅** > **收到的分享**页签页面查看共享的分析文件，如图 *8-207: 被分享成员查看共享的分析*所示。

图 8-207: 被分享成员查看共享的分析



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

8.2.6.3.2 图区右键操作

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。

图区右键操作弹出的菜单列表项根据右键点击的位置不同显示也不同：

- 在图区空白处右键，如图 *8-208: 在图区空白处右键*所示。

图 8-208: 在图区空白处右键

- 在图区的点或边上右键，如图 *8-209: 在图区的点或边上右键*所示。

图 8-209: 在图区的点或边上右键

各菜单项说明如下：

- **添加节点**：在图区中新增节点，详细请参见从右键菜单中添加节点。
- **添加关系**：为选中的两个节点添加关系，从而建立关系网络。

  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。

  - 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。
  - 无向关系：例如：人乘坐火车没有明确的指向，为无向关系。
- **添加标签**：为选中的一个或多个节点添加标识信息，便于分析判别。有关标签的详细介绍，请参见添加用户标签。
- **全选节点**：图区中所有节点都为选中状态。
- **全选关系**：图区中所有关系边都为选中状态。
- **删除已选**：将选中的节点或关系从图区中删除。详细请参见删除已选。
- **反选其他**：当前选中节点以外的所有节点都呈选中状态。

  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。
- **选中关联节点**：当前选中节点所连接的节点都呈选中状态。

  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。
- **合并已选**：将选中的两个或两个以上的节点进行合并显示。

  方便用户梳理整个分析图区的信息，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大于或等于2，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。
- **拆分已选**：将合并的节点进行拆分显示。

  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。

  拆分会提供条件过滤，如图 *8-210:* 条件过滤所示。

图 8-210: 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 *8-211:* 模糊查询所示，搜索含有189的手机号码，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 8-211: 模糊查询

- **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，其它的节点删除。
- **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。

- **一键扩展**：对选中节点进行一键扩展查询操作，获取一键扩展的图结果，并做弹性力布局。
- **节点跳转**：根据节点类型按照后台配置的URL跳转到第三方业务系统。

  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。
- **相关图跳转**：将选中的子图内容发送给后台配置的第三方业务系统。
- **复制**：将选中的对象节点或关系图进行复制，置入粘贴缓存区。支持对象节点复制和关系图复制。
- **粘贴**：如果粘贴缓存区没有可粘贴的对象或者边，该选项将置灰，否则将粘贴缓存区中的内容粘贴到当前分析。
- **保存虚节点**：将虚拟节点持久化。
- **查看选中节点详情**：可在页面右侧面板中查看选中节点的详细信息。

8.2.6.4 关系分析

关系分析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。

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添加节点。

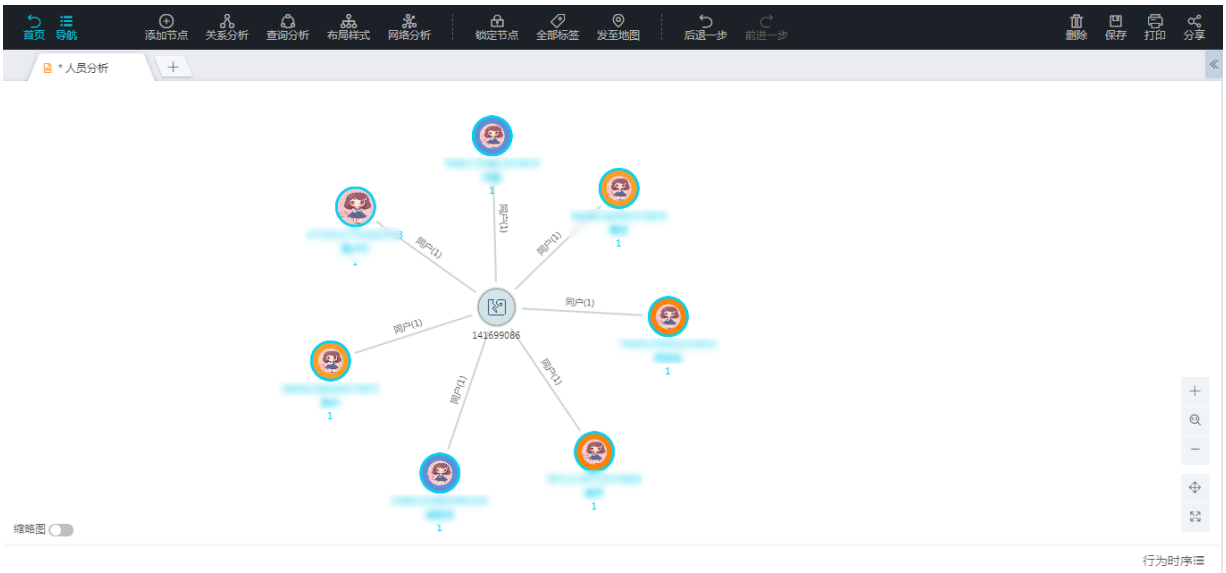
关系分析提供如下两种方式：

- 简单方式：用户直接双击需要进行关系分析的节点，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。

关联分析-简单方式

直接双击节点，默认分析，结果如图 *8-212:* 简单方式所示。

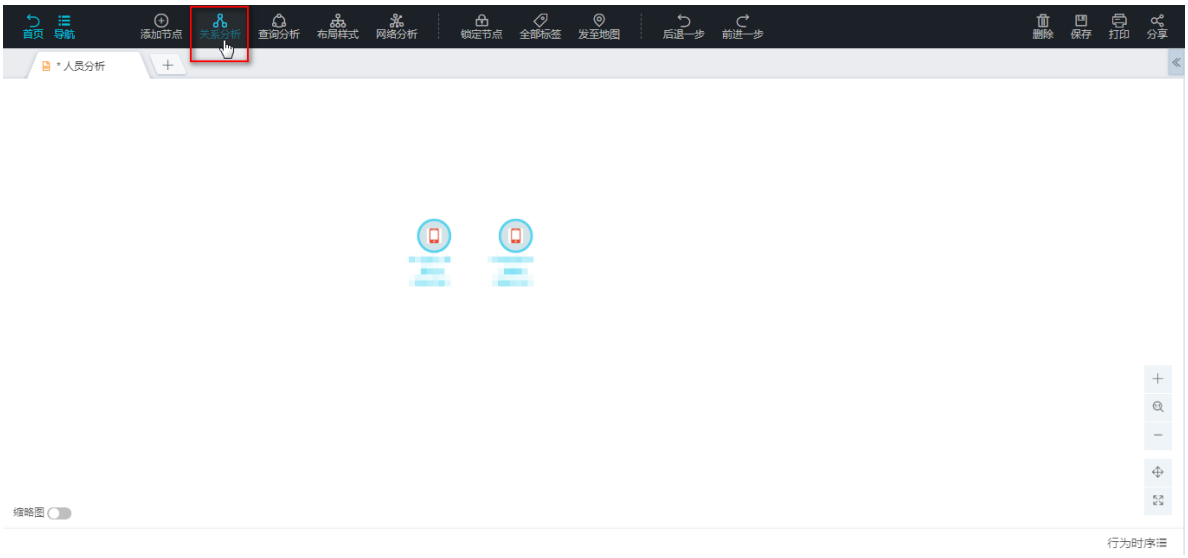
图 8-212: 简单方式



关联分析-高级方式

1. 选中节点，在工具栏上选择关联反查的图标，如图 *8-213:* 高级方式所示。

图 8-213: 高级方式



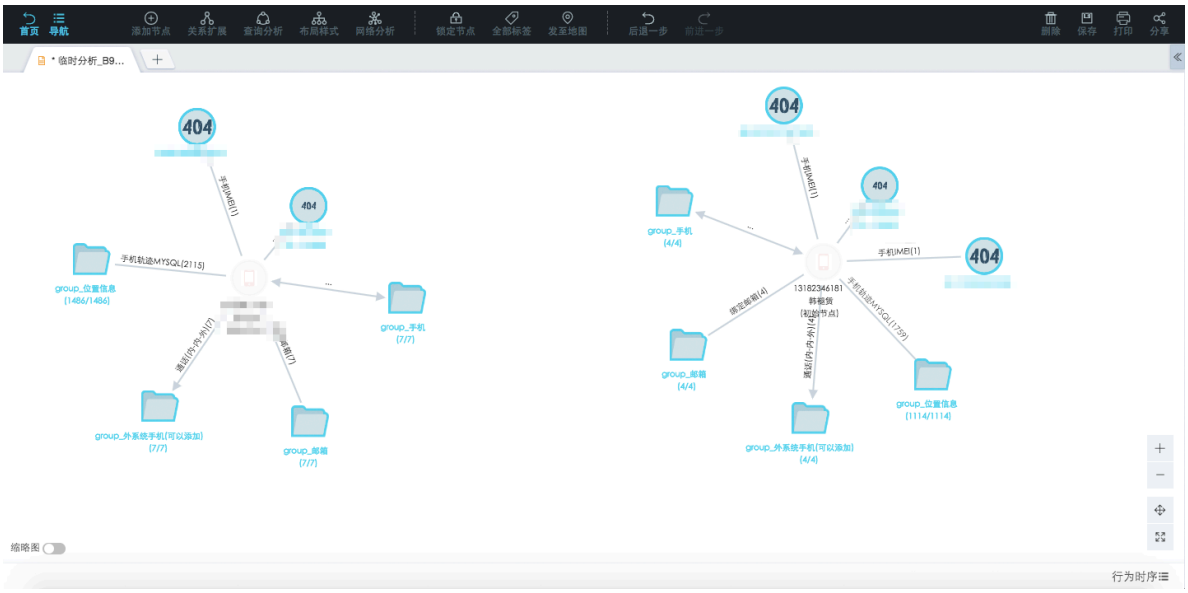
2. 在弹出的关系分析窗口中，进行对象选择、关系选择及关系条件的设置。

   如图 *8-214: 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、日期类型、数值类型、枚举类型、字符串相等、字符串模糊匹配的设定。

图 8-214: 设置关系类型

3. 单击**立即分析**按钮，完成目标节点的关系扩展，结果如图 *8-215: 关系分析结果-高级方式*所示。

图 8-215: 关系分析结果-高级方式



8.2.6.5 查询分析

8.2.6.5.1 群集分析

群集分析，实现批量一次性分析群体间个体之间任意两两之间的关系，包含直接关系和间接关系。

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添加节点。

操作步骤

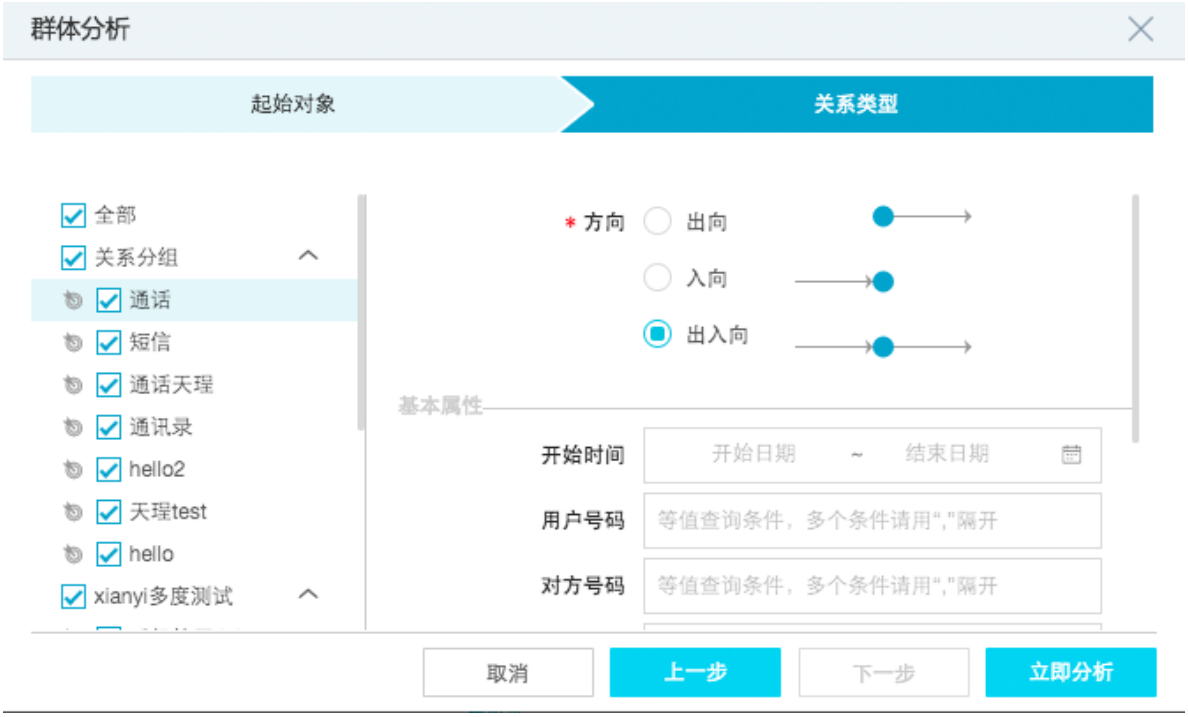
1. 登录关系网络分析工作台。
2. 在图区选中一批已添加的节点。
3. 在工具栏选择**查询分析** > **群集分析**，如图 *8-216:* 群集分析所示。

图 8-216: 群集分析

4. 在弹出的群体分析窗口中，依次进行实体选择、关系选择、关系条件的设置，如图 *8-217: 群体分析窗口*所示。

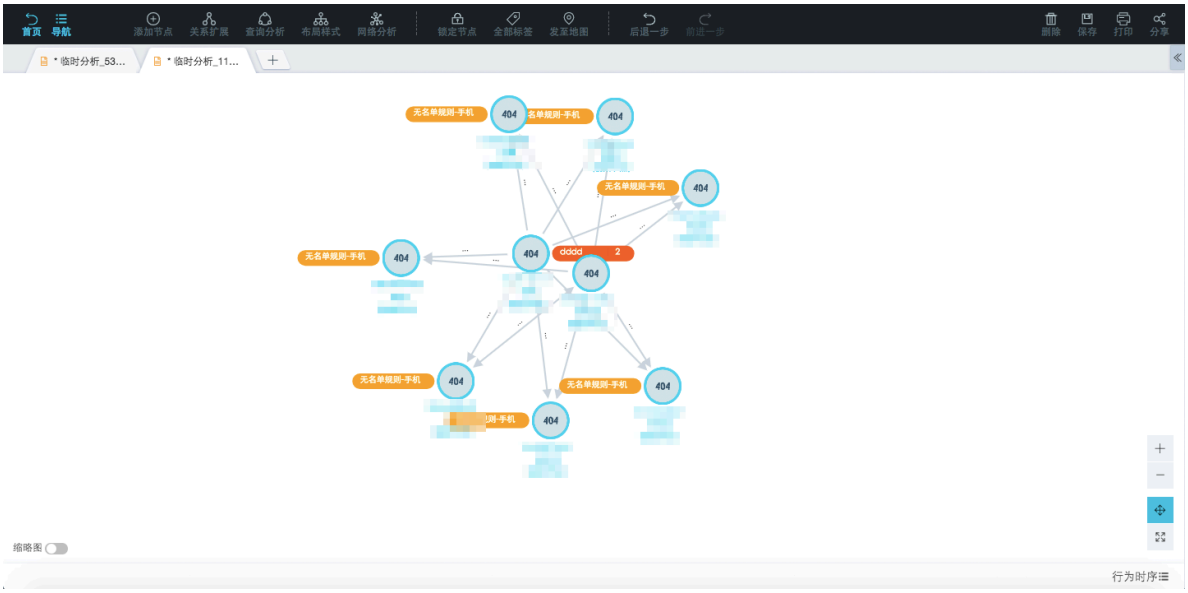
   关系条件设置支持时间类型、日期类型、数值类型、枚举类型、字符串相等、字符串模糊匹配的设定。

图 8-217: 群体分析窗口



5. 单击**立即分析**按钮，完成目标节点的群集分析，如图 *8-218: 群集分析结果*所示。

图 8-218: 群集分析结果



8.2.6.5.2 共同邻居分析

共同邻居分析，即分析一批相同或不同类型的对象的共同联系对象。

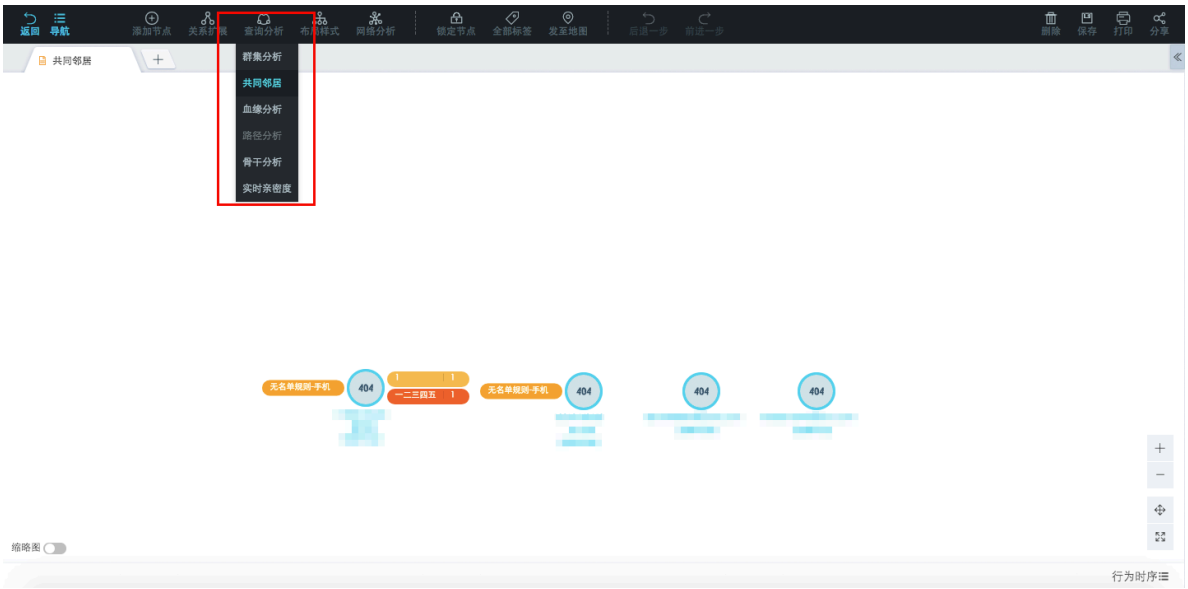
前提条件

- 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添加节点。

操作步骤

1. 登录关系网络分析工作台。
2. 在图区选中两个或多个已添加节点。
3. 在工具栏选择**查询分析** > **共同邻居**，如图 *8-219: 共同邻居*所示。

图 8-219: 共同邻居



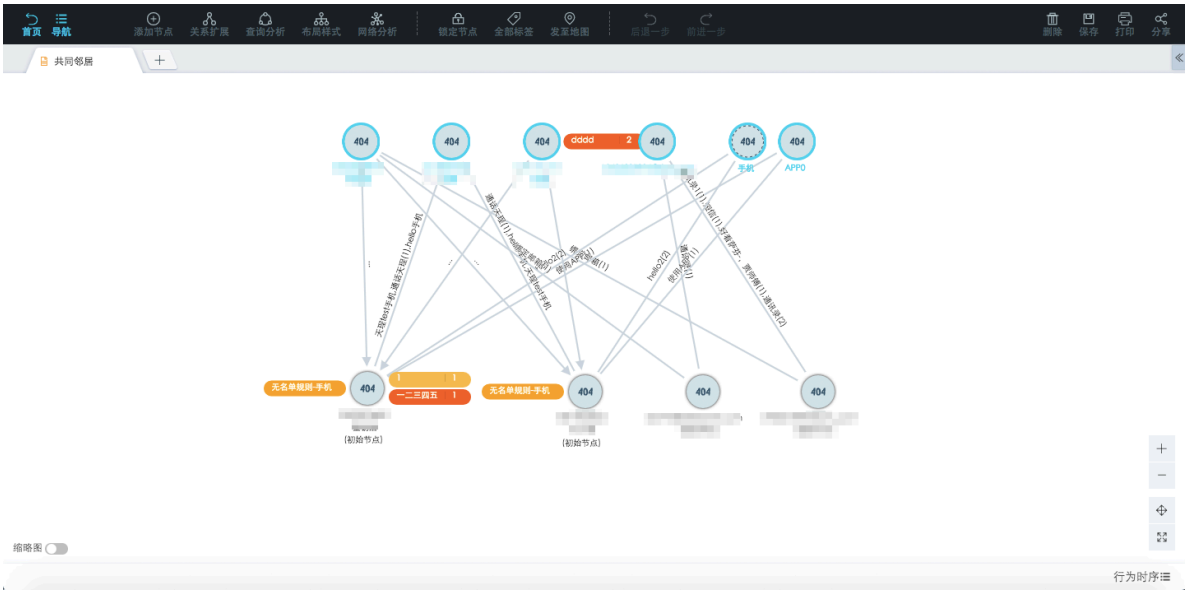
4. 在弹出的共同邻居窗口中，依次进行关联度设置、实体选择、关系选择、关系条件的设置，如图 *8-220: 共同邻居分析条件设定*所示。

   共同邻居关联度默认为查找二度邻居；关系条件设置支持时间类型、日期类型、数值类型、枚举类型、字符串相等、字符串模糊匹配的设定。

图 8-220: 共同邻居分析条件设定

5. 单击**立即分析**按钮，完成目标节点的共同邻居分析，结果如图 *8-221: 共同邻居分析结果*所示。

图 8-221: 共同邻居分析结果



8.2.6.5.3 血缘分析

血缘分析，即通过后台的血缘配置，查询特定类型实体的血缘关系。

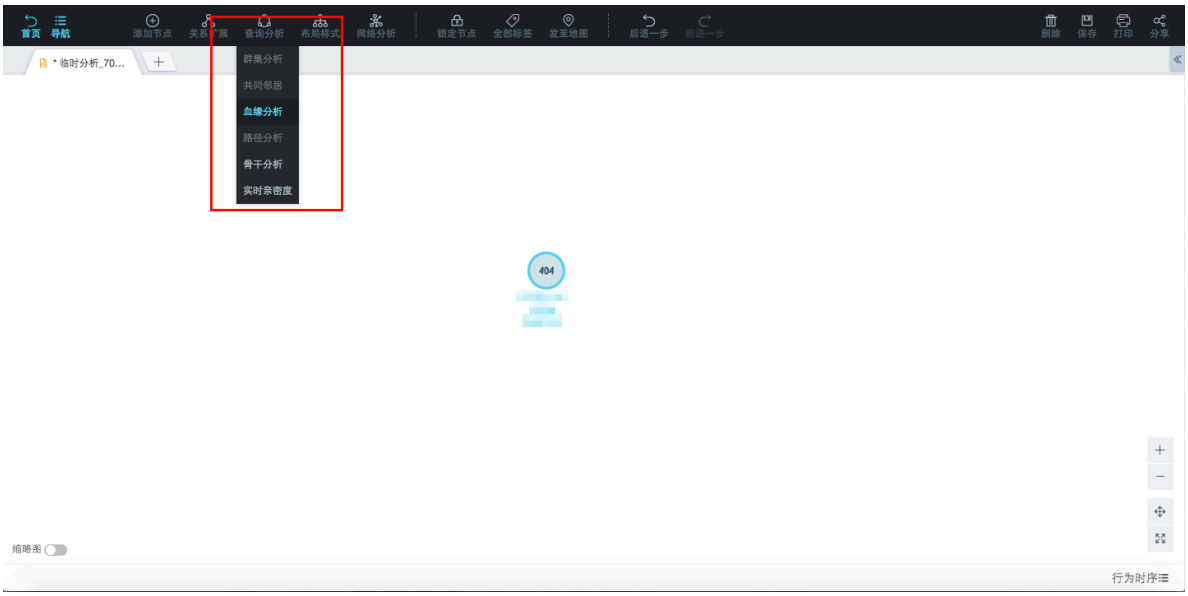
前提条件

- 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添加节点。

操作步骤

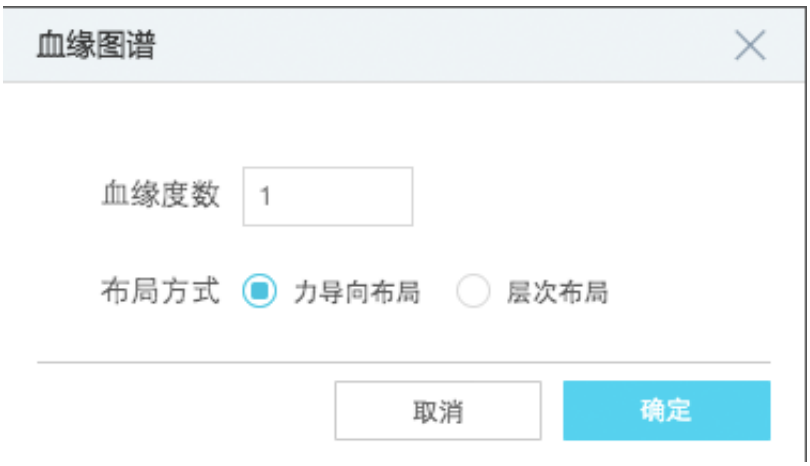
1. 登录关系网络分析工作台。
2. 在图区选中一个或多个需要进行血缘分析的已添加节点。
3. 在工具栏选择**查询分析** > **血缘分析**，如图 *8-222: 血缘分析*所示。

图 8-222: 血缘分析



4. 在弹出的血缘图谱窗口中，设置想要查询的血缘关系的度数，以及查看结果的布局方式，如图 *8-223:* 血缘图谱窗口所示。

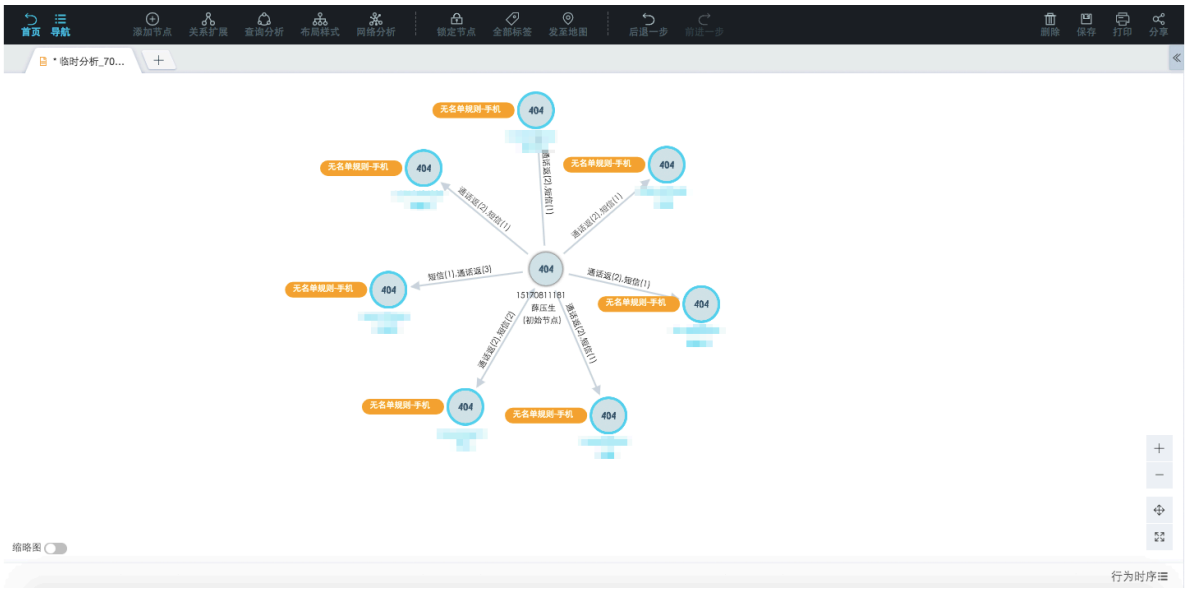
图 8-223: 血缘图谱窗口



5. 单击**确定**按钮，完成目标节点的血缘分析。

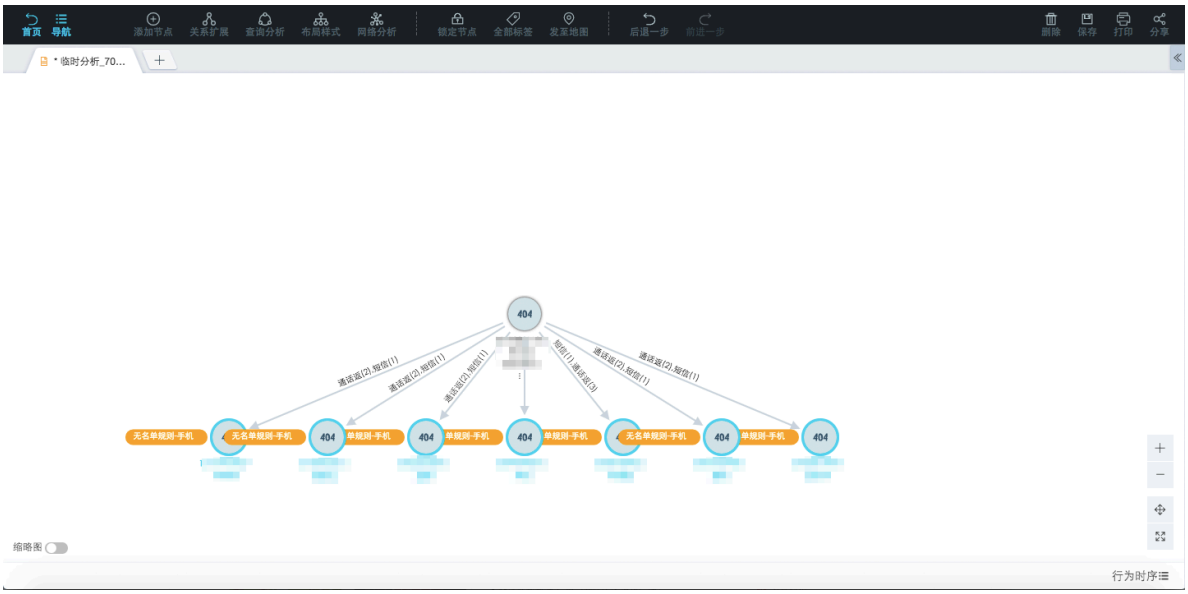
   假设用户设置了查询一度血缘关系，选择力导向布局的结果如图 *8-224:* 分析结果-力导向布局所示。

图 8-224: 分析结果-力导向布局



用户设置了查询一度血缘关系，选择层次布局的结果如图 *8-224: 分析结果-力导向布局*所示。

图 8-225: 分析结果-层次布局



8.2.6.5.4 路径分析

路径分析，即分析两个对象之间的关联路径。

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添加节点。

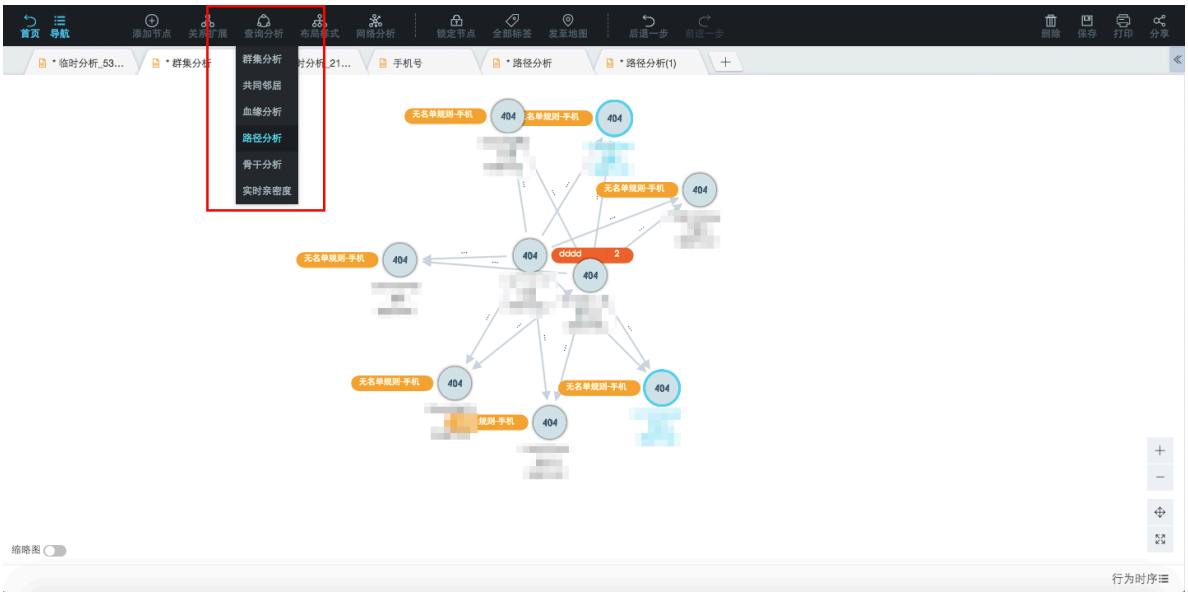
背景信息

用户可以选中两个目标对象分析其关联路径。

操作步骤

1. 登录关系网络分析工作台
2. 在图区选中两个已添加的节点。
3. 在工具栏选择**查询分析** > **路径分析**，如图 *8-226: 路径分析入口*所示。

图 8-226: 路径分析入口



4. 在弹出的路径分析窗口中，选择**全量数据**或**仅本页数据**进行分析，如图 *8-227: 路径分析窗口*所示。

图 8-227: 路径分析窗口

- 全量数据分析

  1. 需要设置关系条件、关联度数、是否显示热点数据，如图 *8-228:* 设定全量数据分析条件所示。

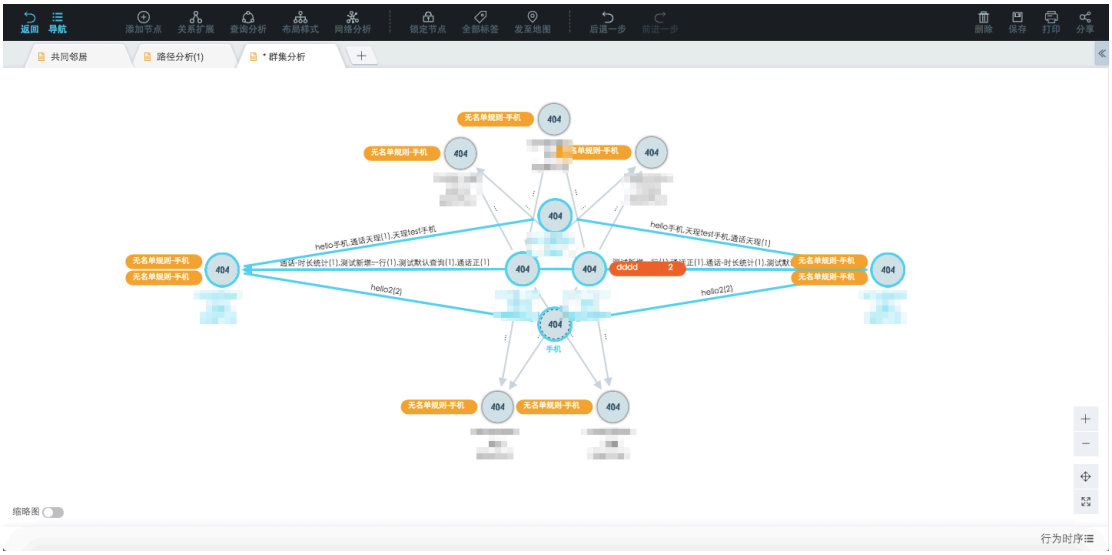
     如果勾选**显示热点数据**，会将热点数据显示出来，但不会扩展。

     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。

图 8-228: 设定全量数据分析条件

2. 单击**查询**按钮，进行路径分析，分析结果如图 *8-229: 全量数据分析结果*所示。

图 8-229: 全量数据分析结果

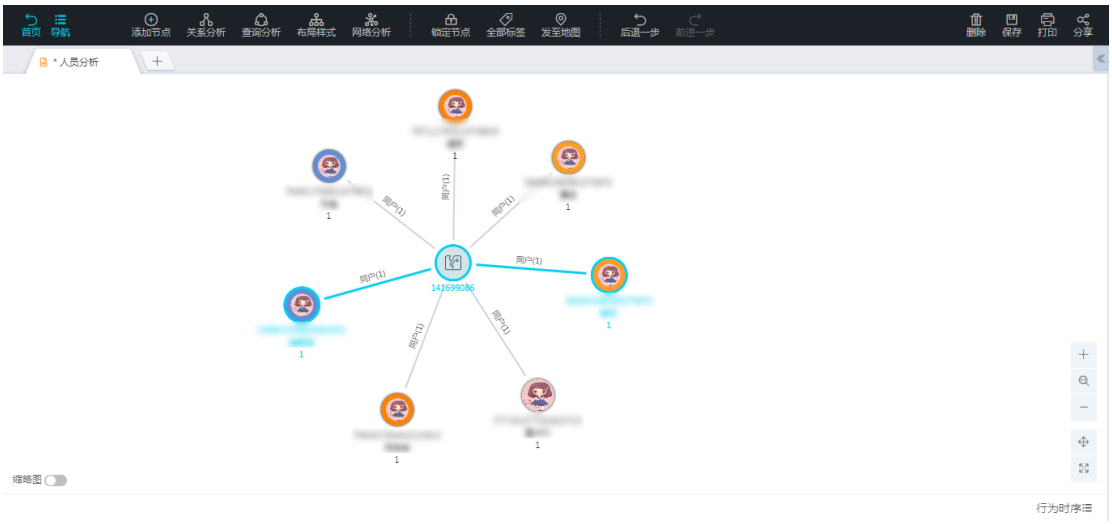


- 仅本页数据分析

  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关联度数的路径分析。

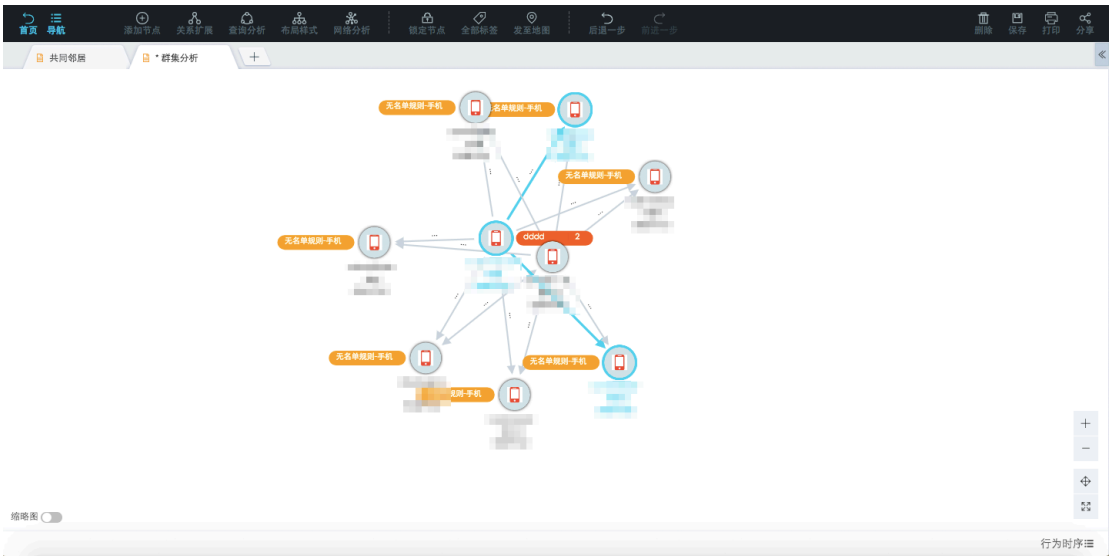
  - 最短路径分析结果如图 *8-230: 仅本页数据分析-最短路径分析结果*所示。

图 8-230: 仅本页数据分析-最短路径分析结果



- 指定关联度数的路径分析，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，分析结果如图 *8-231:* 仅本页数据分析-指定关联度数分析结果所示。

图 8-231: 仅本页数据分析-指定关联度数分析结果



8.2.6.5.5 骨干分析

骨干分析，即根据当前页面上的团伙网络图，通过智能业务算法，探索关系网络中核心骨干节点。

前提条件

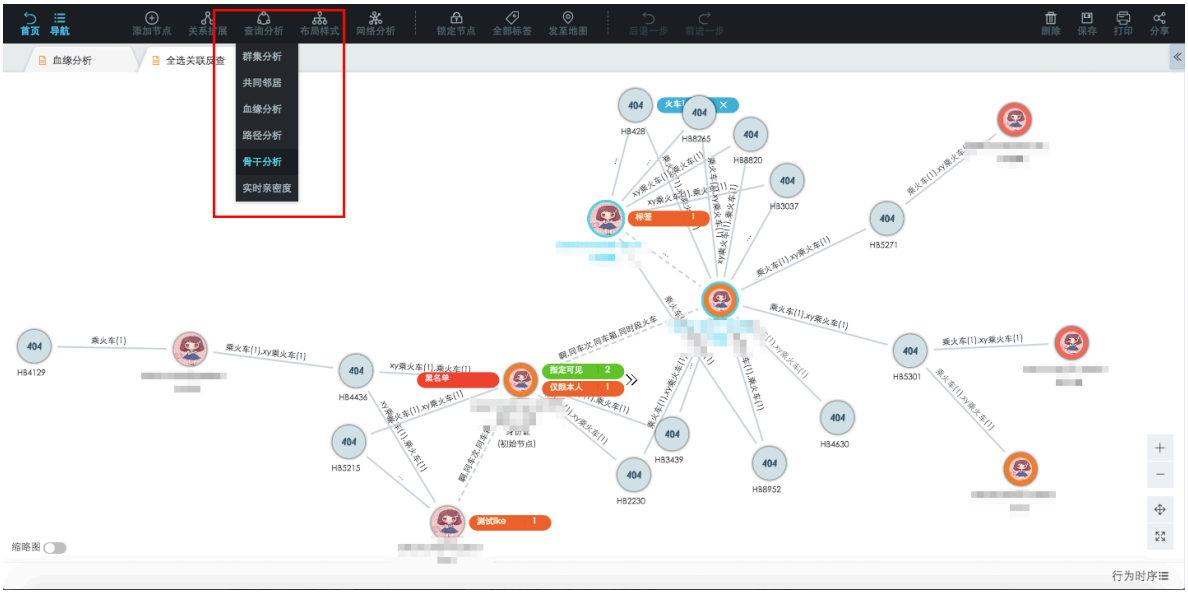
- 已存在分析文件，新建分析操作请参见新建分析。

- 已存在节点对象，新增节点操作请参见添加节点。

操作步骤

1. 登录关系网络分析工作台
2. 在图区选中已添加的节点。
3. 在工具栏选择**查询分析** > **骨干分析**，如图 *8-232: 骨干分析*所示。

图 8-232: 骨干分析

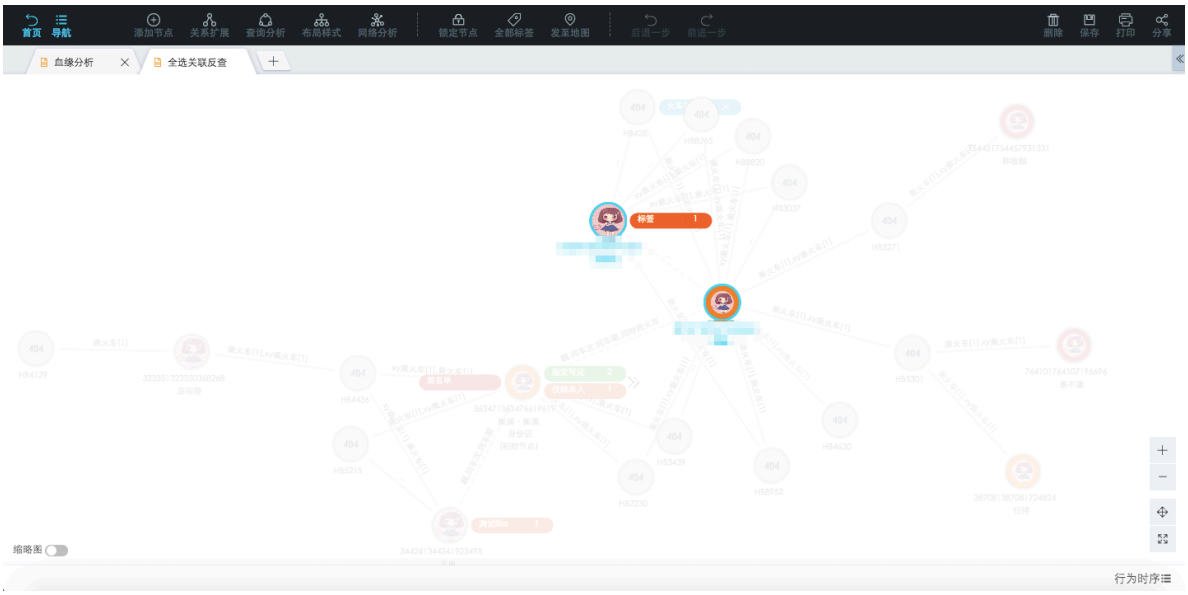


4. 选择想要分析的骨干节点的类型，如图 *8-233: 骨干节点类型*所示。

图 8-233: 骨干节点类型

5. 单击**确定**按钮，图区的骨干节点会高亮显示，如图 *8-234: 骨干分析结果*所示。

图 8-234: 骨干分析结果



8.2.6.5.6 实时亲密度

实时亲密度，即根据后台配置的实时亲密度关系，查询特定类型实体的实时亲密度关系。

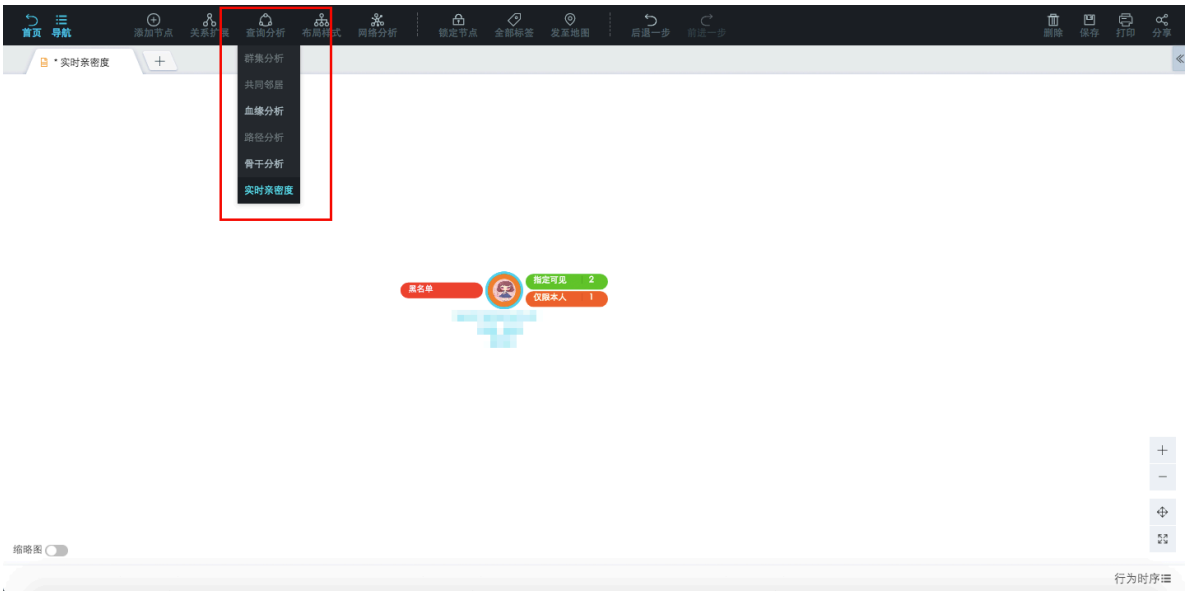
前提条件

- 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添加节点。

操作步骤

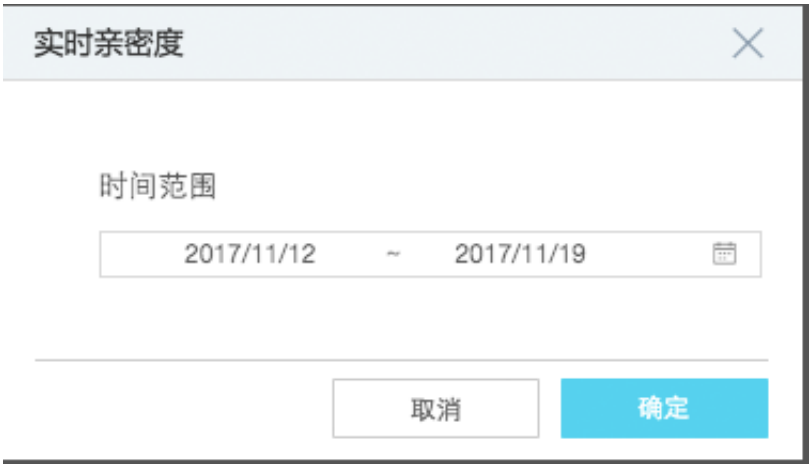
1. 登录关系网络分析工作台。
2. 在图区选中想要进行实时亲密度分析的一个或多个已添加的节点。
3. 在工具栏选择**查询分析** > **实时亲密度**，如图 *8-235:* 实时亲密度所示。

图 8-235: 实时亲密度



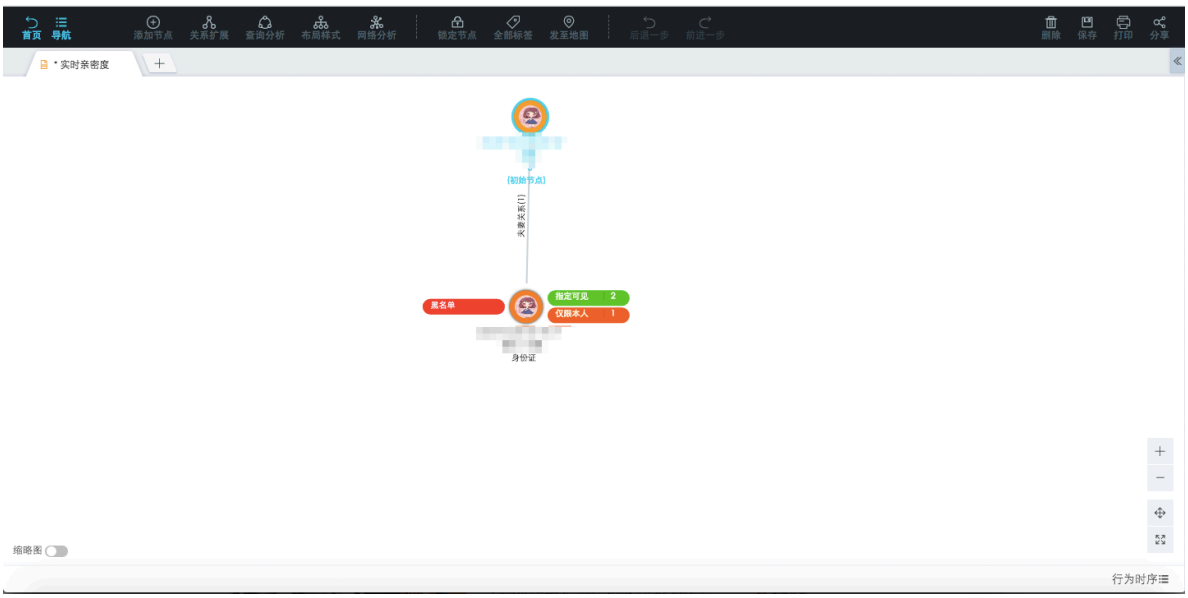
4. 在弹出的实时亲密度窗口中，设置实时亲密度的时间范围，如图 *8-236: 实时亲密度窗口*所示。

图 8-236: 实时亲密度窗口



5. 单击**确定**按钮，实时亲密度分析结果如图 *8-237: 实时亲密度分析结果*所示。

图 8-237: 实时亲密度分析结果



8.2.6.6 布局样式

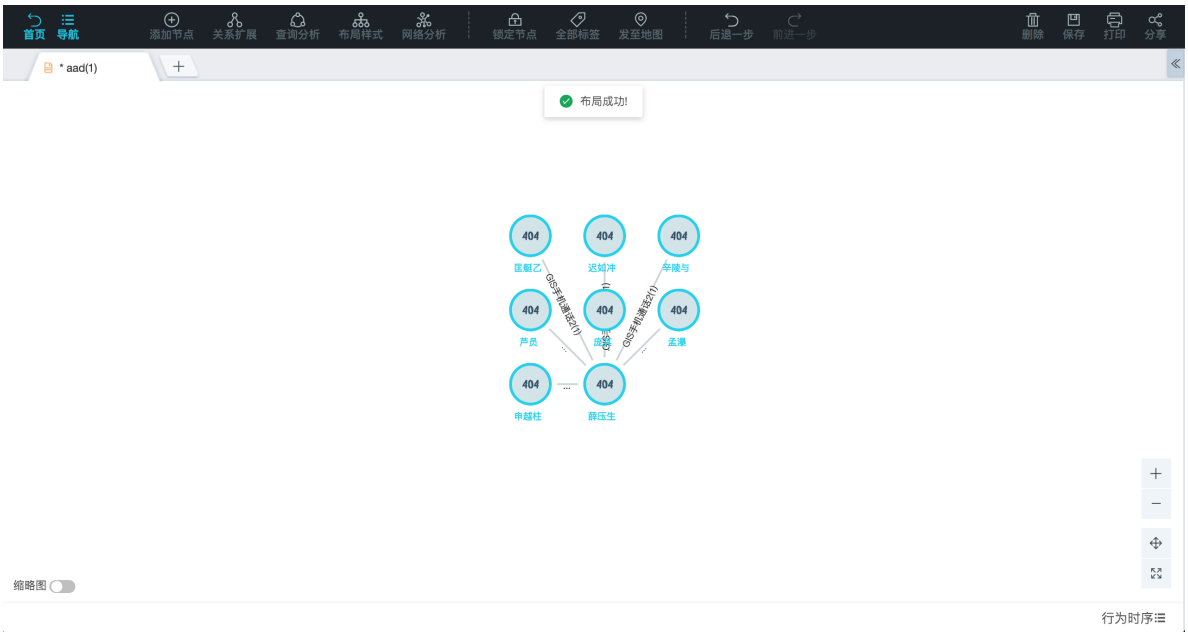
关系网络分析图区支持矩阵布局、圆环布局、横直线布局、竖直线布局、力导向布局和层次布局。

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。

1. 登录关系网络分析工作台。
2. 在图区选中两个或两个以上的对象节点。
3. 在工具栏选择**布局样式** > **矩阵布局**，如图 *8-238:* 矩阵布局所示。

图 8-238: 矩阵布局

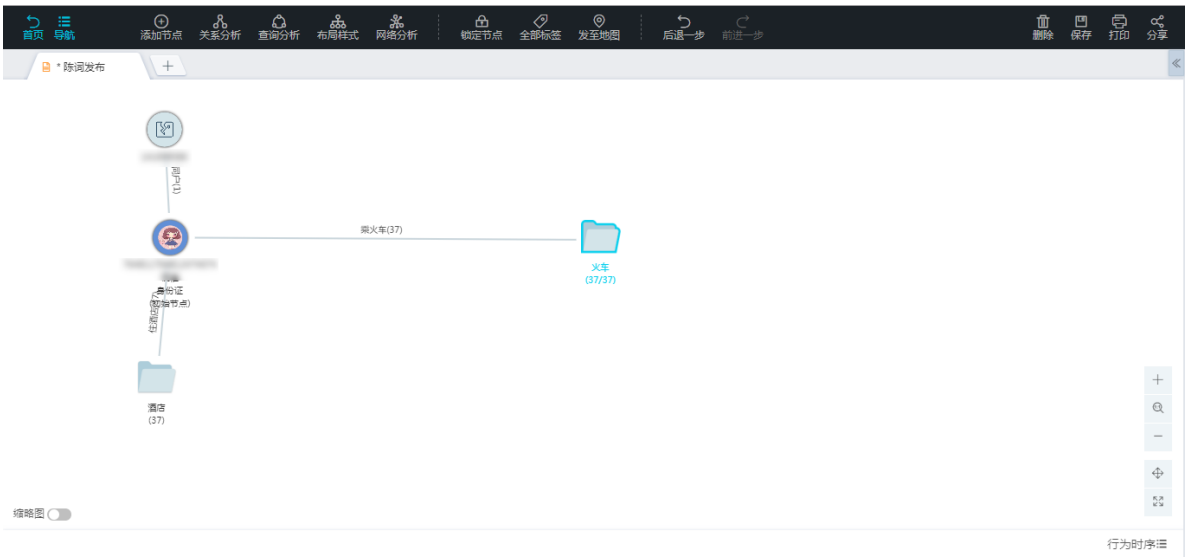


4. 矩阵布局可以配合**合并节点**功能一起使用，如图 *8-239: 合并节点*所示，能够在信息量大的情况下，将分析后的信息归整合并。

图 8-239: 合并节点

合并后的效果如图 *8-240: 合并结果*所示。

图 8-240: 合并结果

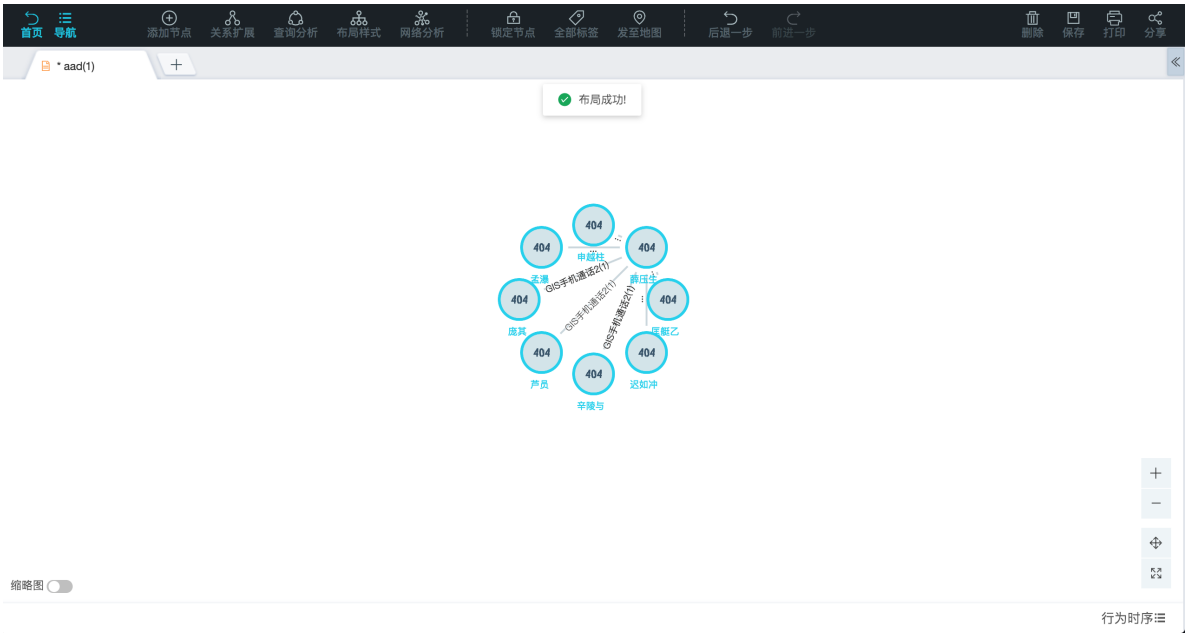


圆环布局

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。

1. 在图区选中两个或两个以上的对象节点。
2. 在工具栏选择**布局样式** > **圆环布局**，如图 *8-241:* 圆环布局所示。

图 8-241: 圆环布局

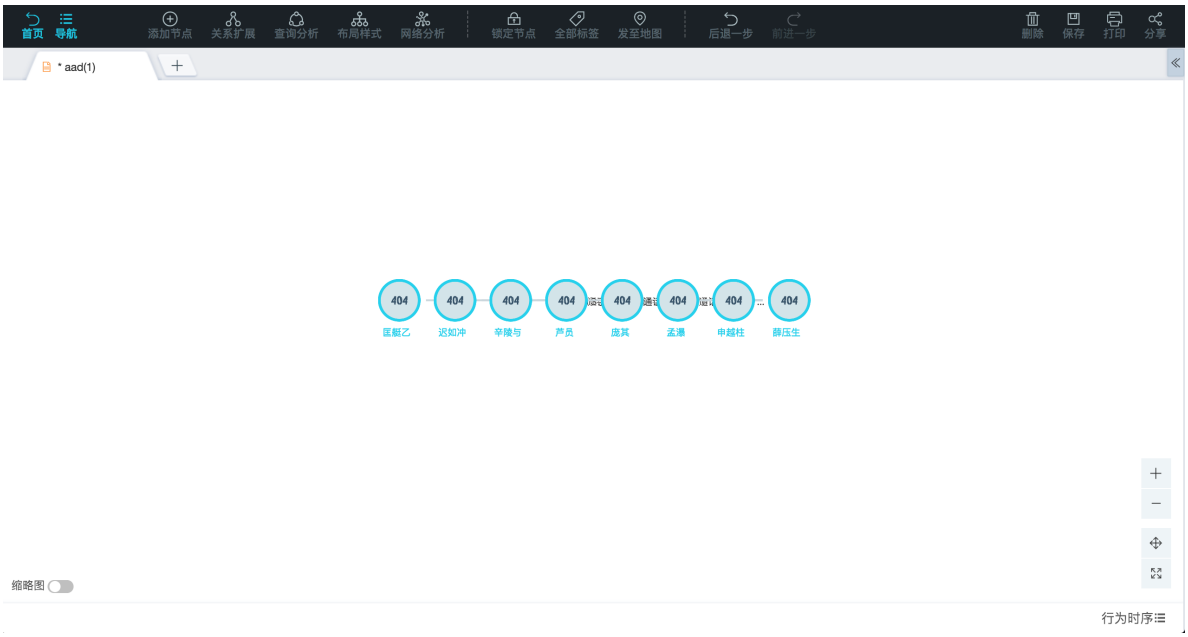


横直线布局

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。

1. 在图区选中两个或两个以上的对象节点。
2. 在工具栏选择**布局样式** > **横直线布局**，如图 *8-242:* 横直线布局所示。

图 8-242: 横直线布局

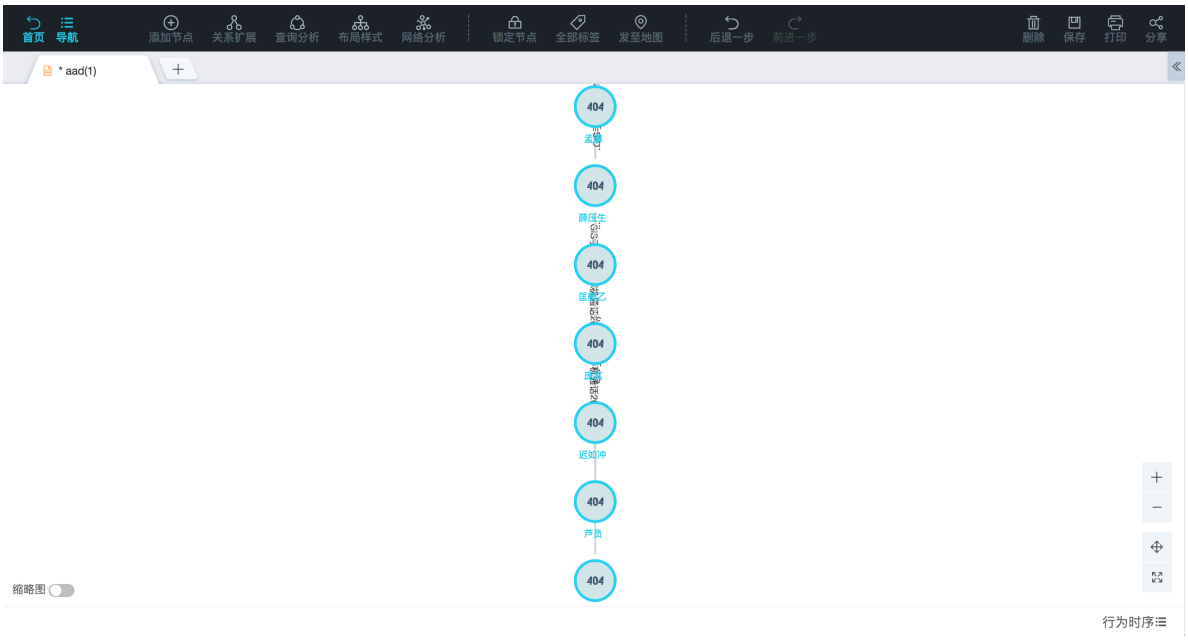


竖直线布局

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，方便用户从能从竖向流程分析。

1. 在图区选中两个或两个以上的对象节点。
2. 在工具栏选择**布局样式** > **竖直线布局**，如图 *8-243:* 竖直线布局所示。

图 8-243: 竖直线布局



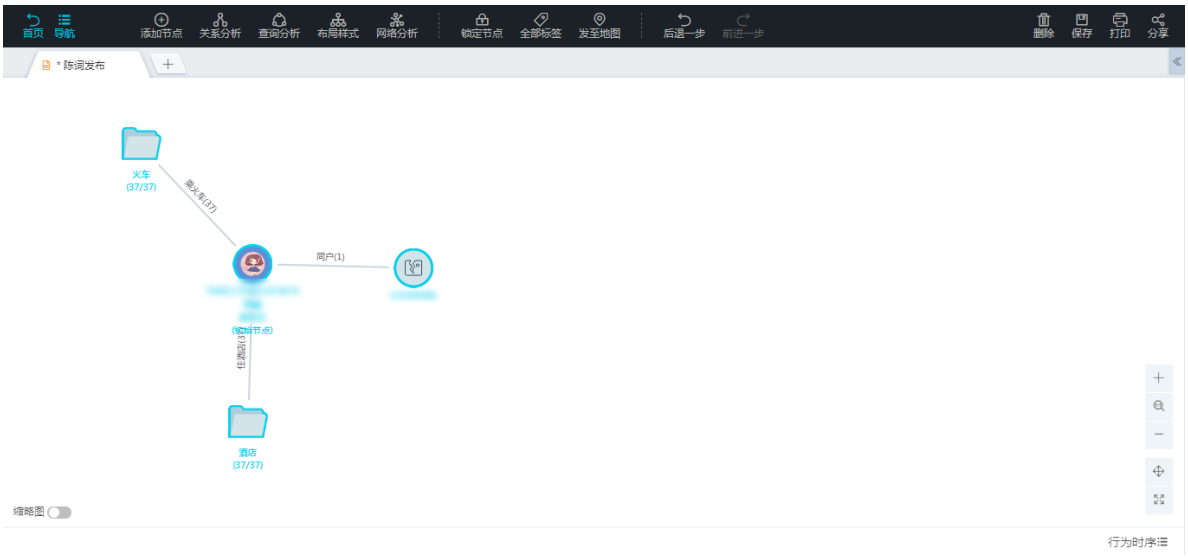
力导向布局

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。

1. 在图区选中两个或两个以上的对象节点。
2. 在工具栏选择**布局样式** > **力导向布局**，如图 *8-244: 力导向布局*所示。

图 8-244: 力导向布局

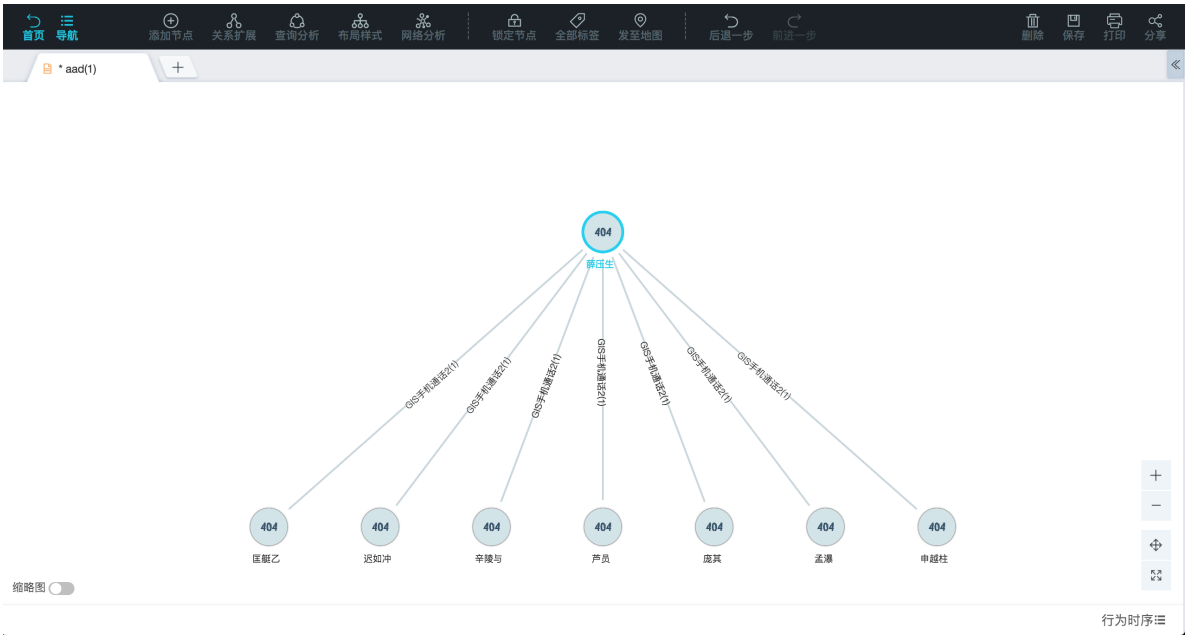


层次布局

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。

1. 在图区选中一个或多个对象节点。
2. 在工具栏选择**布局样式** > **层次布局**，如图 *8-245:* 层次布局所示。

图 8-245: 层次布局

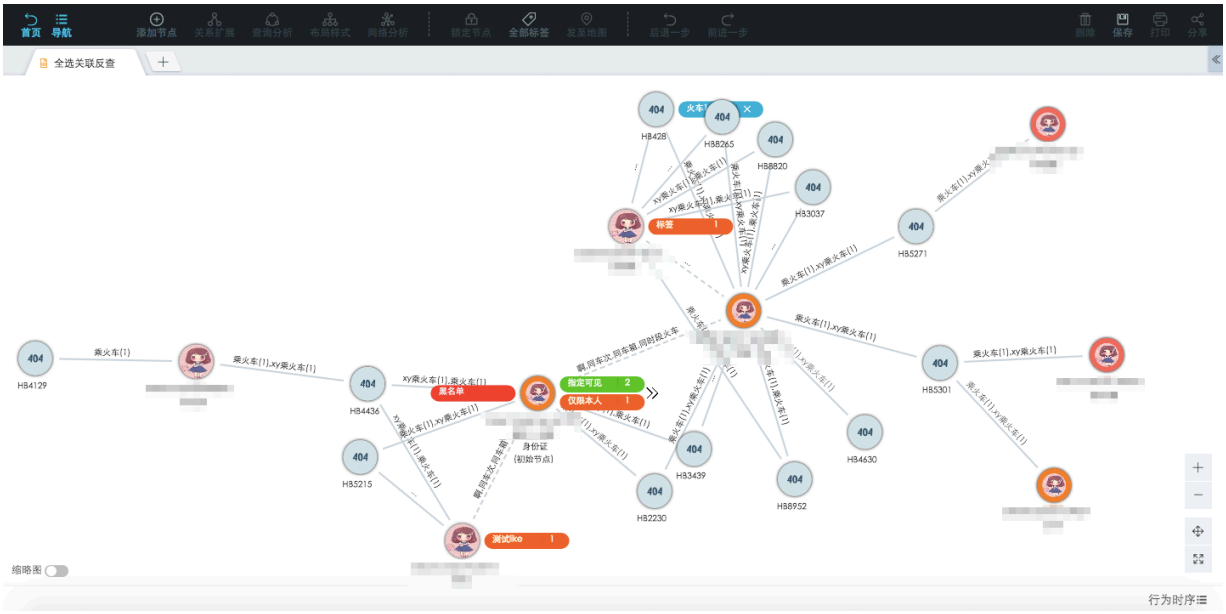


8.2.6.7 网络分析

网络分析是对页面上的网络图从位置关键度、关系紧密度、活动紧密度等不同的角度进行算法分析后，将结果节点高亮显示。

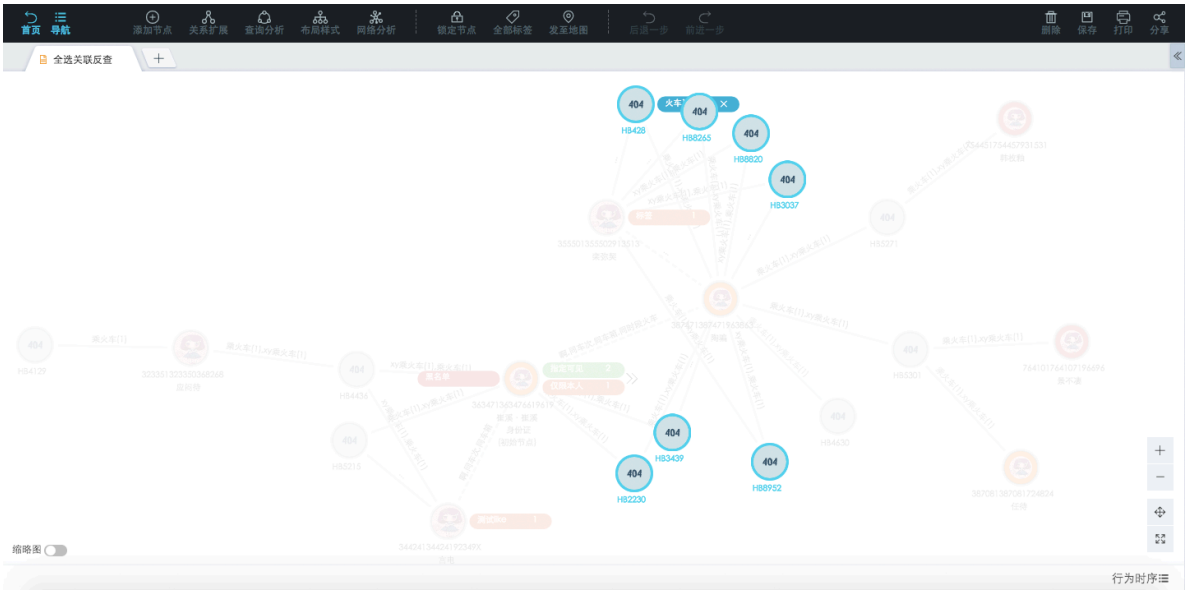
登录关系网络分析工作台，以图图 *8-246: 关系网络图示例*所示的网络图为例进行说明。

图 8-246: 关系网络图示例



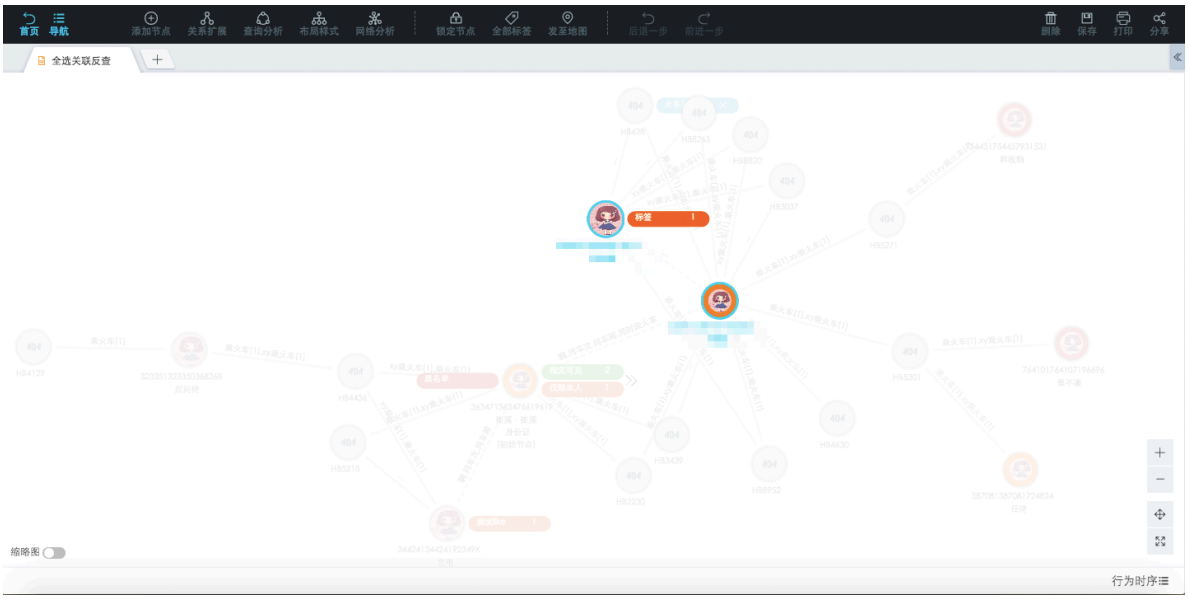
- 在工具栏选择**网络分析** > **位置关键度**，如图 *8-247: 位置关键度*所示。

图 8-247: 位置关键度



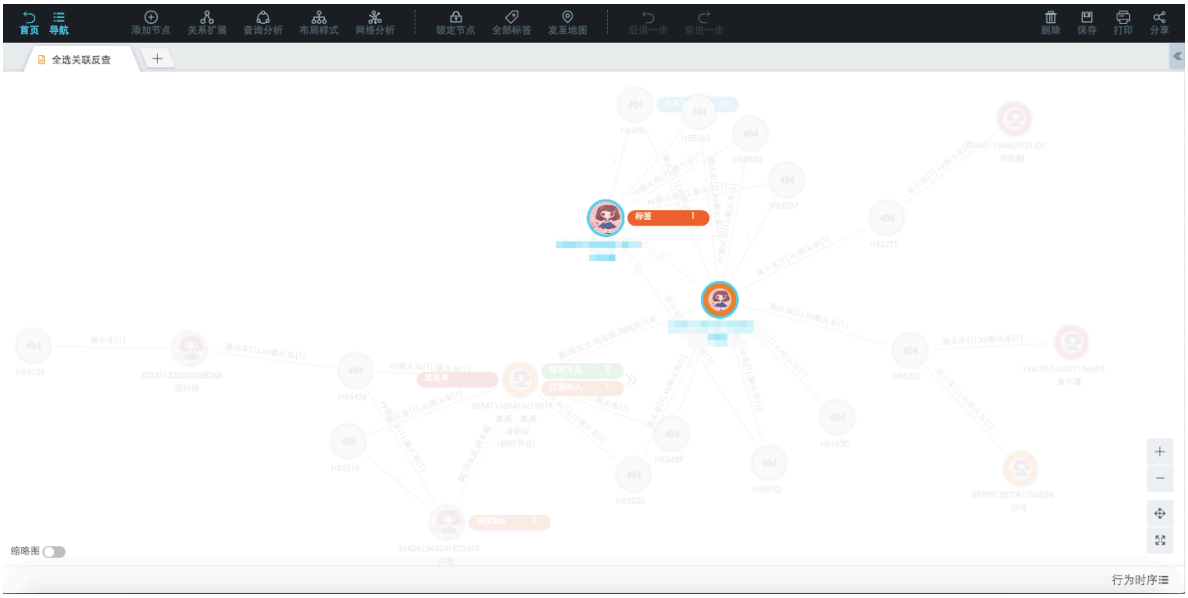
- 在工具栏选择**网络分析** > **关系紧密度**，如图 *8-248: 关系紧密度*所示。

图 8-248: 关系紧密度



- 在工具栏选择**网络分析 > 活动紧密度**，如图 *8-249: 活动紧密度*所示。

图 8-249: 活动紧密度



8.2.6.8 行为时序

行为时序区域即图区下方面板，包含行为明细清单、行为分析和时序分析。

8.2.6.8.1 行为明细

1. 登录关系网络分析工作台。

2. 有如下方式打开行为时序面板：
   - 在图区选择对象和关系，点击**行为时序**面板图标。
   - 在图区单击关系边，自动打开**行为时序**面板。
3. 查看行为明细清单，如图 *8-250: 查看行为明细清单*所示。

图 8-250: 查看行为明细清单

可以在右侧关系区域切换查看各关系的行为明细清单。

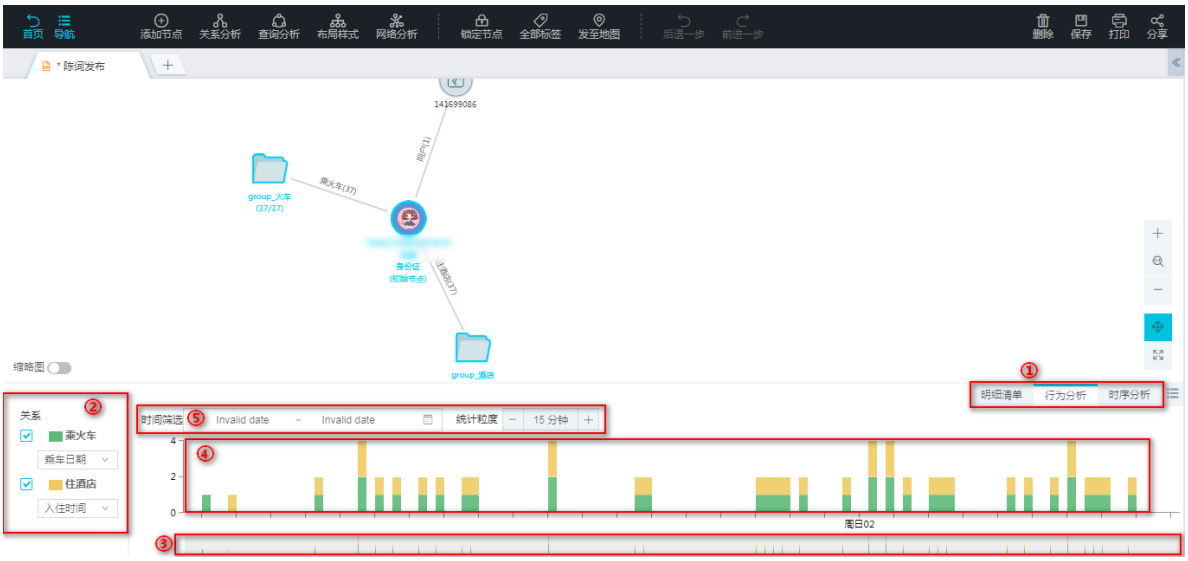
可以单击**数据导出**按钮，将所有关系的行为明细清单导出至Excel表格。

8.2.6.8.2 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。

1. 登录关系网络分析工作台。
2. 在图区中选择待进行行为分析的对象和关系。
3. 点击页面下方**行为时序** > **行为分析**，显示行为分析页签页面，如图 *8-251: 行为分析页签*所示。

图 8-251: 行为分析页签



行为分析区域说明如下：

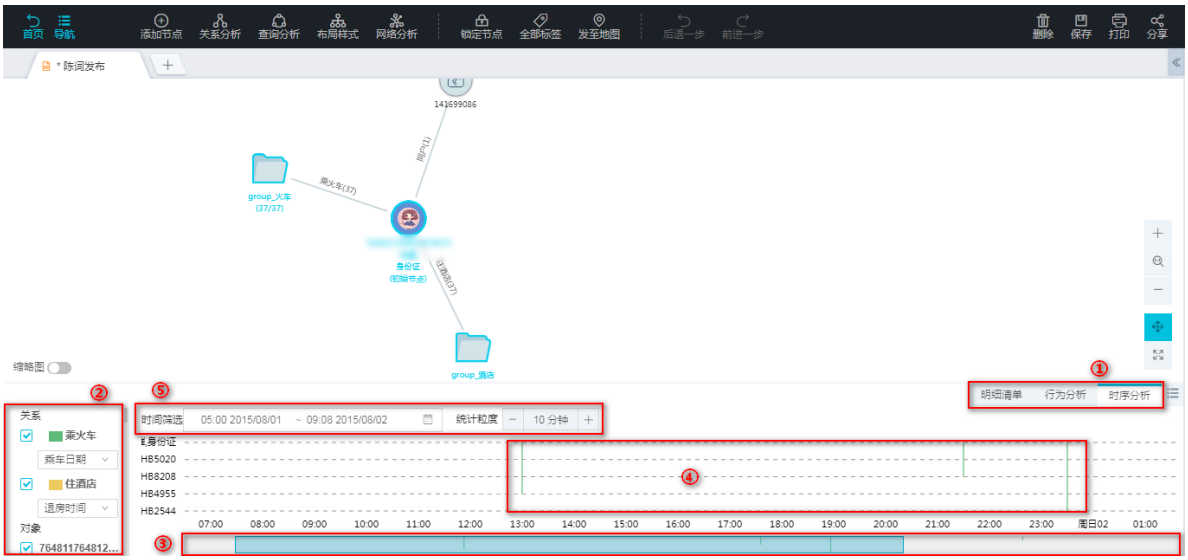
- 区域1：Tab页切换选择，选择**行为分析**Tab页。
- 区域2：关系筛选，筛选需要分析的关系。
- 区域3：缩略图筛选，通过下方缩略图进行范围定位，选中中间可移动缩略图。
- 区域4：选中行为，鼠标悬浮在柱状图上，可显示行为的明细信息。
- 区域5：设置筛选时间和统计粒度：
  - 可设置起始时间，进行时间筛选。
  - 点击+或-图标，进行统计粒度的增加和减小。

8.2.6.8.3 时序分析

时序分析，即在时间维度上详细展示每个事件发生细节。

1. 登录关系网络分析工作台。
2. 在图区中选择待进行时序分析的对象和关系。
3. 点击页面下方**行为时序** > **时序分析**，显示时序分析页签页面，如下图所示。

图 8-252: 分析结果



默认时序图中只允许五个对象进行展示分析。

时序分析区域说明如下：

- 区域1：Tab页切换选择，选择**时序分析**Tab页。
- 区域2：可进行关系和对象的选择，最多只允许同时对五个对象进行分析。
- 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：当鼠标移动到对应时序线上时，悬浮弹出时序明细。
- 区域5：设置筛选时间和统计粒度：
  - 可设置起始时间，进行时间筛选。
  - 点击+或-图标，进行统计粒度的增加和减小。

8.2.6.9 属性统计

图区右侧为属性统计区，可查看对象或关系的详情、统计和属性等信息。

8.2.6.9.1 详情信息

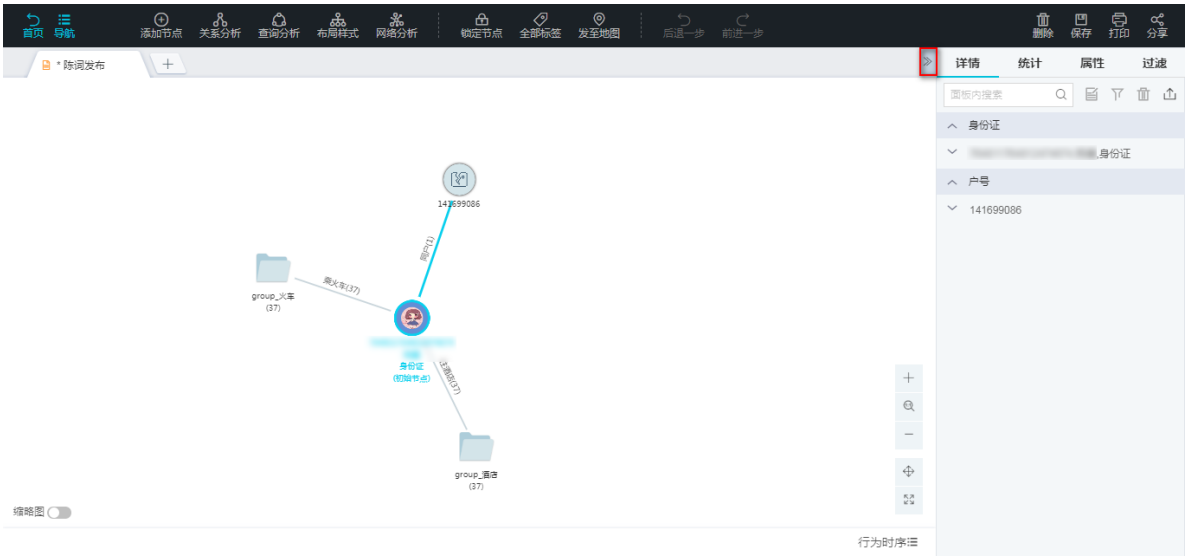
画布分析中经常会有非常多的实体对象，尤其当节点比较多的时候，业务用户需要将重点关注的某类对象突出显示。I+属性统计区域的详情页面提供的可视化分层展示功能，可以将用户重点关注的对象突出显示，以作区分。

操作步骤

1. 登录关系网络分析工作台。

2. 在图区选中一个或多个节点或关系。
3. 点击图区右侧属性统计区图标，展开属性统计区域，默认显示详情页签，如图 *8-253: 属性统计区域-详情*所示。

图 8-253: 属性统计区域-详情



详情页签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。

4. 单击收放按钮，可以展开或收起该点的信息，如图 *8-254: 对象详情信息*所示。

   当用户在右侧面板选中对象后，则在图区除该对象以外的实体和关系都会暗淡下来。用户可以选择需要重点关注的对象，系统支持通过键盘上的**Control** 键进行多选。

   当鼠标单击左边主图上的的节点或关系边，右边详情页面会对应的展示选定的节点或关系的全部属性信息，可以多选，如果多选，按住`Ctrl`键左键多选或框选。

图 8-254: 对象详情信息

详情页面可以针对选中内容进行如下操作：

- 文字标注：即添加文字标签，详情请参考添加用户标签。
- 选中：在图区中选中该节点。
- 删除：在图区中将选中的对象删除。
- 导出：将选中对象信息导出到Excel表格中。

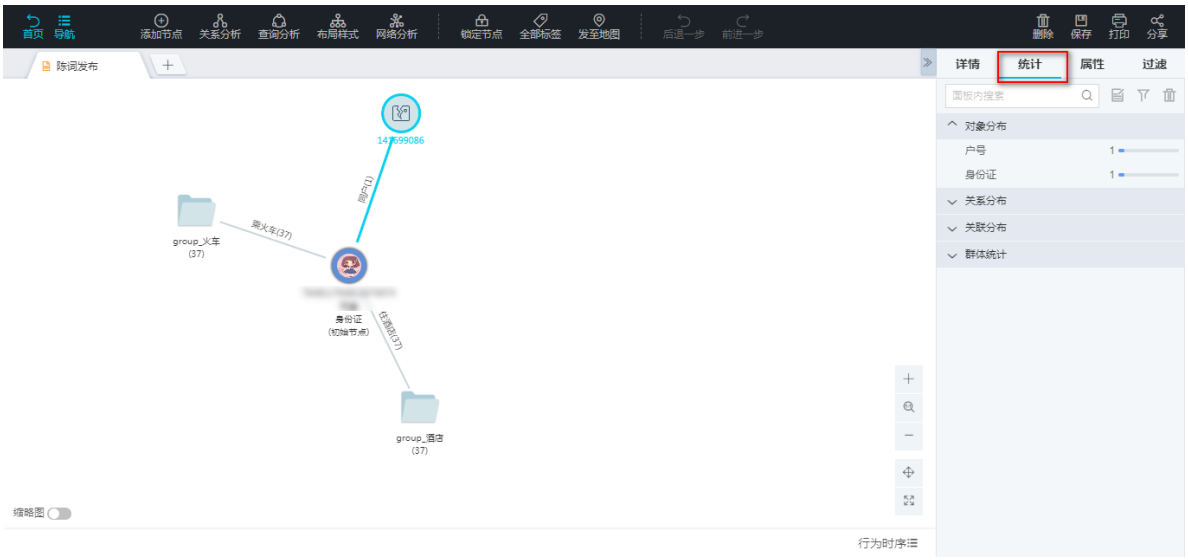
8.2.6.9.2 统计信息

统计页签页面展示对选中部分的对象或关系统计。

操作步骤

1. 登录关系网络分析工作台。
2. 在图区选中一个或多个节点或关系。
3. 点击图区右侧属性统计区图标，展开属性统计区域。
4. 点击切换到**统计**页签页面，如图 *8-255: 统计页签*所示。

图 8-255: 统计页签



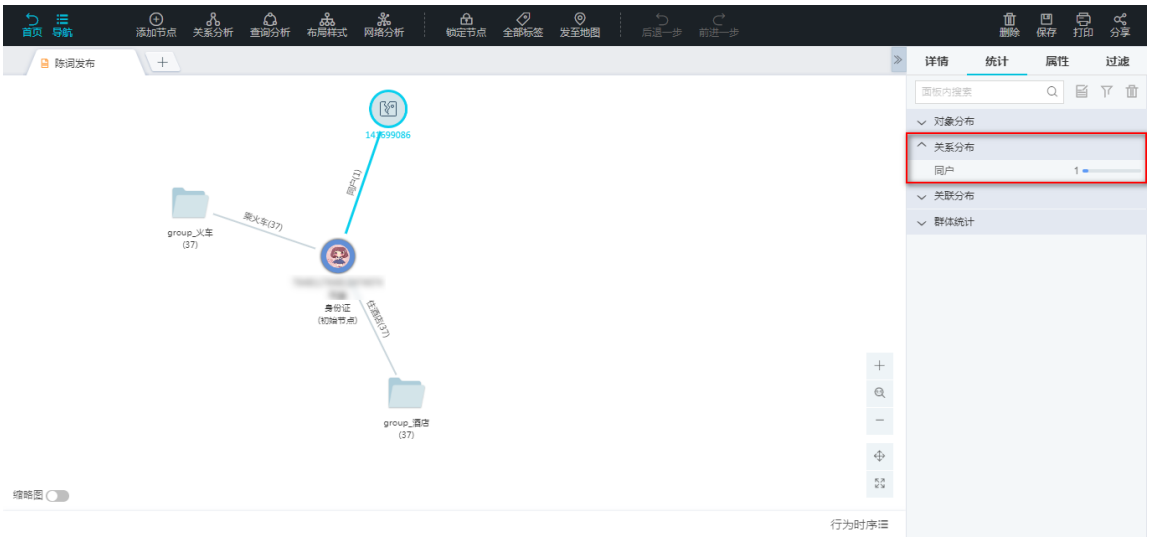
5. 单击收放按钮，可以展开或收起选中对象或关系的属性统计信息。

   - 查看对象分布，如图 *8-256: 对象分布*所示。

     图 8-256: 对象分布

   - 查看关系分布，如图 *8-257: 关系分布*所示。

图 8-257: 关系分布



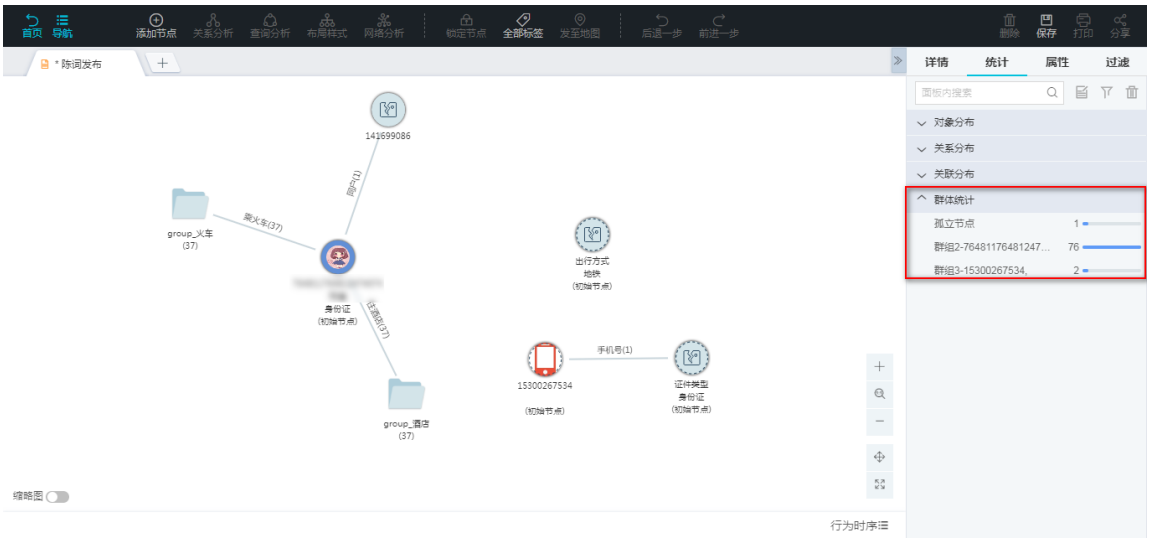
- 查看关联分布，如图 *8-258: 关联分布*所示。

图 8-258: 关联分布

- 查看群体分析，如图 *8-258: 关联分布*所示。

  群体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。

图 8-259: 群体分析



- 在群集统计中，将所有孤立的节点统称为一个群集，称之为**孤立节点**。
- 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，孤立节点群体在最上面，后面按照群集个数中依次降序排列。

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。

统计页面可以针对选中内容进行如下操作：

- 文字标注：即添加文字标签，详情请参考添加用户标签。
- 选中：在图区中选中该节点。
- 删除：在图区中将选中的对象删除。

8.2.6.9.3 属性信息

属性页签页面展示对选中部分的对象或关系的对象信息、关系信息和标签信息。

操作步骤

1. 登录关系网络分析工作台。
2. 在图区选中一个或多个节点或关系。
3. 点击图区右侧属性统计区图标，展开属性统计区域。
4. 点击切换到**属性**页签页面，如图 *8-260: 属性页签*所示。

图 8-260: 属性页签

5. 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。

   - 查看对象属性，如图 *8-261: 对象属性*所示。

     图 8-261: 对象属性

   - 查看关系属性，如图 *8-262: 关系属性*所示。

图 8-262: 关系属性

- 查看标签属性，如图 *8-263: 标签属性*所示。

图 8-263: 标签属性

属性页面可以针对选中内容进行如下操作：

- 文字标注：即添加文字标签，详情请参考添加用户标签。
- 选中：在图区中选中该节点。
- 删除：在图区中将选中的对象删除。

8.2.6.9.4 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。

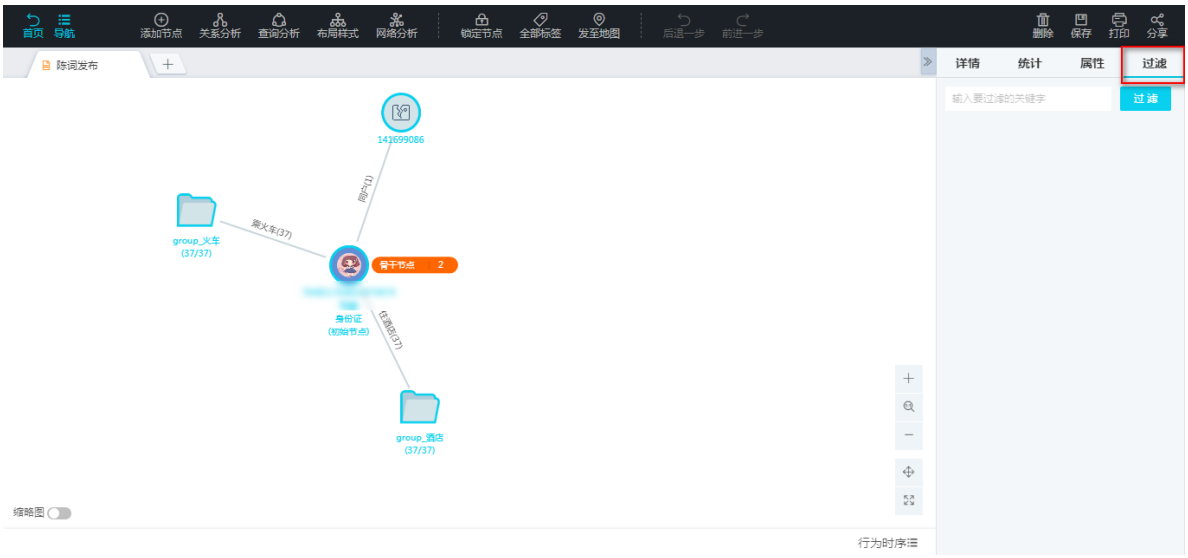
背景信息

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. 登录关系网络分析工作台。
2. 点击图区右侧属性统计区图标，展开属性统计区域。
3. 点击切换到**过滤**页签页面，如图 *8-264: 二次过滤入口*所示。

图 8-264: 二次过滤入口



4. 输入要过滤的关键字。

   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。

8.2.7 地图

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。

8.2.7.1 对象位置分析

位置用于在地图上定位对象，显示对象的空间特性，譬如人员的常驻地、户籍地等。只有配置了位置信息的对象才能添加到地图上。若某个特定的对象，即使配置了位置信息，由于数据上不存在该对象的位置数据，则该对象也无法添加到地图上。

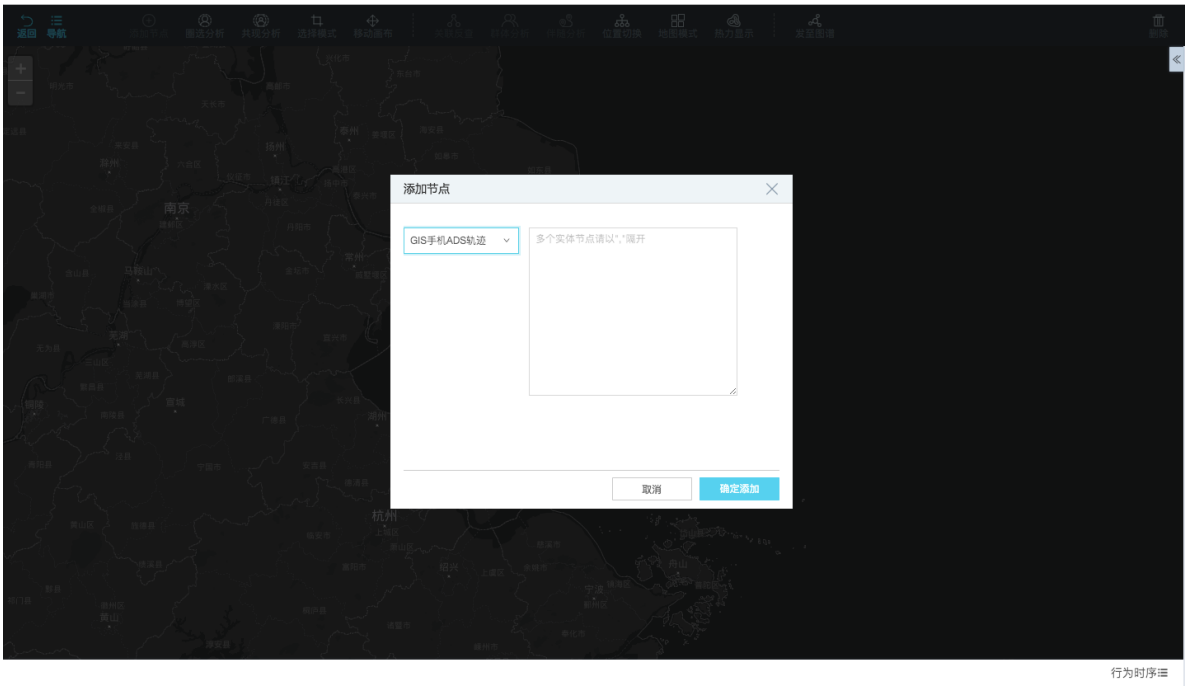
配置对象位置

对象位置由管理员在后台进行配置，请联系管理员进行配置。

添加对象节点

1. 点击页面上方的**添加节点**按钮，弹出添加节点窗口，如图 *8-265: 添加节点*所示。

图 8-265: 添加节点



2. 选择将要添加的对象类型，下拉列表中只会列出配置了位置信息的对象类型。
3. 输入要添加对象的主键，多个主键用换行或者逗号分隔。
4. 单击**确定**按钮，即可将节点添加到地图的相应位置上。

对象节点位置切换

管理员可能为同一种对象定义多种不同的位置数据，譬如说人员的户籍地与工作地点等。其中一种位置作为该对象的默认位置，用户添加对象时使用默认位置。根据业务需要，可以实时切换位置类型，地图上相关对象会根据配置重新布局，如图 *8-266: 位置切换*所示。

图 8-266: 位置切换

删除对象节点

1. 选中一个或多个对象节点。
2. 点击页面右上角的**删除**图标，有如下情况：
   - 删除所选：删除选中的对象节点。
   - 清空全部：删除所有的对象节点。

8.2.7.2 对象轨迹分析

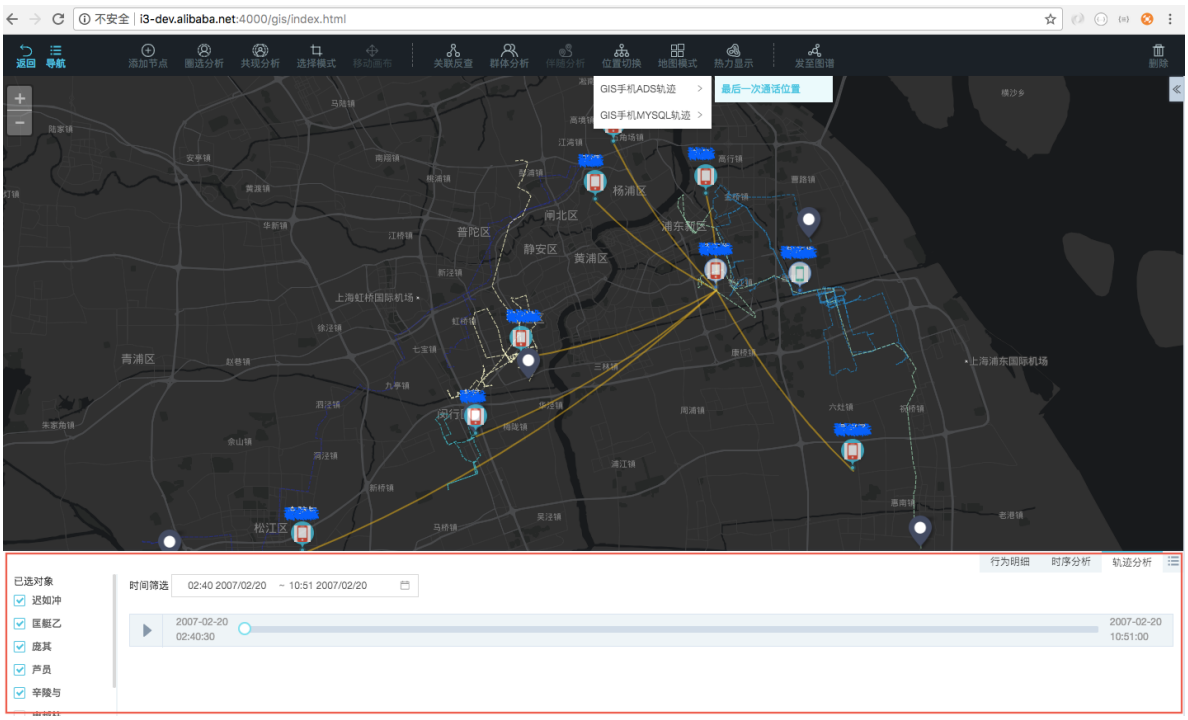
配置对象轨迹

对象轨迹由管理员在后台进行配置，请联系管理员进行配置。

对象轨迹分析

1. 将要分析的点添加到地图上，详细请参见添加对象节点。
2. 打开下侧面板（如果没有下侧面板按钮，请联系管理员赋予权限）后，切换至**轨迹分析**页签页面，如图 *8-267: 轨迹分析页签*所示。

图 8-267: 轨迹分析页签



3. 轨迹分析页签页面左侧为地图上所有点的列表，用户选择要分析的对象节点，并在右侧的时间控件上设定时间范围，则选中对象的轨迹将显示在地图区域。

   鼠标悬浮在对象节点上，相关的轨迹会高亮显示。

4. 点击**播放**图标，可以播放对象的轨迹。

圈选分析

圈选用于在特定时间与空间中找人，支持圆形圈选分析、矩形圈选分析和多边形圈选分析。

1. 选择圈选方式：

   - 点击页面上方**圈选分析** > **圆形圈选分析**。
   - 点击页面上方**圈选分析** > **矩形圈选分析**。
   - 点击页面上方**圈选分析** > **多边形圈选分析**。

2. 定义圈选空间：

   - 圆形圈选分析：直接按住鼠标左键然后拖拽即可定义，圆形圈选分析最大支持半径为10KM。
   - 矩形圈选分析：直接按住鼠标左键然后拖拽即可定义，矩形圈选分析最大支持边长为20KM。
   - 多边形圈选分析：通过鼠标在地图上单击定义顶点，请确保最后一个顶点与第一个顶点重合即可，多边形圈选分析最大支持边长为20KM。如图 *8-268: 多边形圈选分析*所示。

图 8-268: 多边形圈选分析

3. 定义圈选时间、模式和目标对象类型和基本属性，如图 *8-269:* 设置参数所示，单击**搜素对象**按钮。

图 8-269: 设置参数

4. 单击**添加对象**按钮将圈选结果添加到地图上，如图 *8-270:* 圈选结果所示。

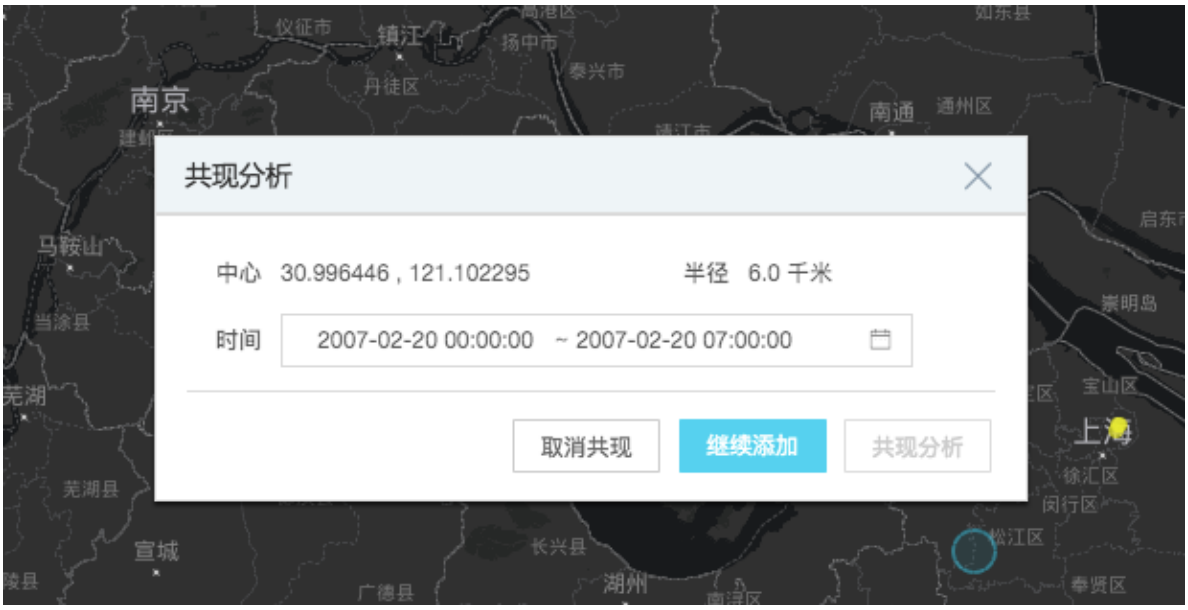
图 8-270: 圈选结果

共现分析

共现分析是在多个时空域上查找出现在不止一个时空域的对象，应用场景为串并案的嫌疑人筛查。支持圆形共现分析、矩形共现分析和多边形共现分析。

1. 选择共现分析方式：
   - 点击页面上方**共现分析 > 圆形共现分析**。
   - 点击页面上方**共现分析 > 矩形共现分析**。
   - 点击页面上方**共现分析 > 多边形共现分析**。
2. 定义共现空间：
   - 圆形共现分析：直接按住鼠标左键然后拖拽即可定义，圆形共现分析最大支持半径为10KM。
   - 矩形共现分析：直接按住鼠标左键然后拖拽即可定义，矩形共现分析最大支持边长为20KM。
   - 多边形共现分析：通过鼠标在地图上单击定义顶点，请确保最后一个顶点与第一个顶点重合即可，多边形共现分析最大支持边长为20KM。
3. 定义圈选时间，如图 *8-271:* 配置共现分析参数所示。

图 8-271: 配置共现分析参数



可以单击**继续添加**按钮继续圈选，步骤同第二步定义共现空间。

4. 单击**共现分析**按钮，在弹出的共现分析窗口中定义圈选命中个数和目标对象过滤条件后，单击**开始分析**按钮，如图 *8-272:* 共现分析窗口所示。

图 8-272: 共现分析窗口

5. 单击**添加对象**按钮，将共现结果添加到地图上，则会显示对象在所有圈选的时间的轨迹，如图 *8-273: 共现分析轨迹*所示。

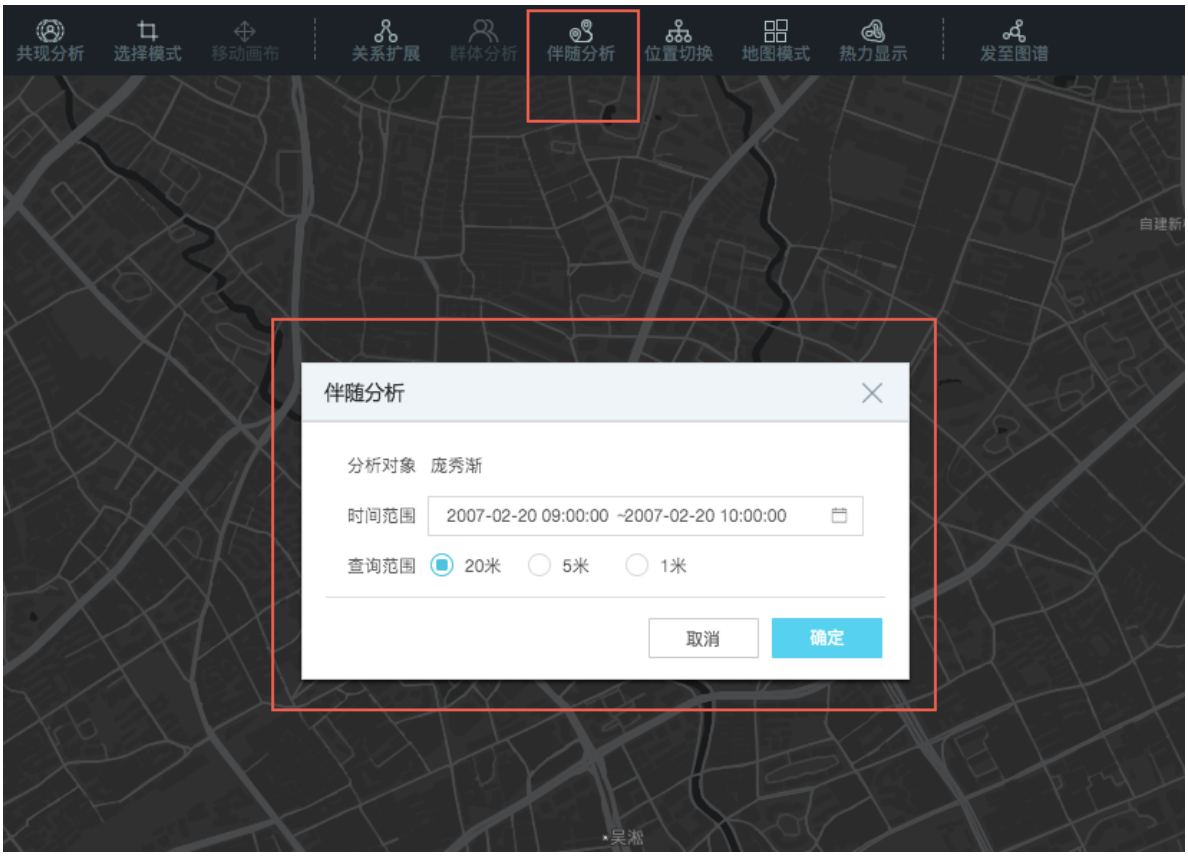
图 8-273: 共现分析轨迹

伴随分析

伴随分析，即计算与已知对象在特定时间区间轨迹相似的其他未知对象，用来发现作案同伙。伴随分析依赖对象的轨迹数据，并且轨迹上包括至少一个4到10位的geohash，如果对象未定义轨迹或者不包含geohash，则无法进行伴随分析，请联系管理员在后台配置对象的轨迹。

1. 在地图上选中已知对象或者将已知对象添加到地图上。
2. 单击页面上方的**伴随分析**按钮，弹出伴随分析窗口。
3. 设置分析的时间范围和查询范围后，单击**确定**按钮，如图 *8-274:* 伴随分析窗口所示。

图 8-274: 伴随分析窗口



4. 单击**确定**按钮，如图 *8-275:* 伴随结果所示，将计算出的伴随对象添加到地图上。

图 8-275: 伴随结果

8.2.7.3 地图模式

地图模式用于凸显地图上的重点区域，将重点区域以着色高亮的形式显示，以便于用户分析。

点击页面上方的**地图模式**图标，在下拉菜单中选择模式即可进行地图模式切换。

8.2.7.4 热力显示

热力图用于显示对象在地图上的密度分布，分为实时热力图与历史热力图。

1. 选择实时热力图或者历史热力图。
2. 配置对象属性与时间范围，如图 *8-276:* 热力图窗口所示。

图 8-276: 热力图窗口

3. 单击**显示热力**按钮，热力图显示效果如图 *8-277: 热力图显示效果*所示。

图 8-277: 热力图显示效果

8.2.7.5 对象关系网络分析

地图模块支持部分图谱模块的功能，主要包括关系分析、群体分析、属性统计、行为时序等，用法与图谱模块一致，详情可参考图谱模块。

发至图谱

在地图分析过程中可以将节点发送到图谱的关系网络中，使用图谱模块更加丰富的功能继续进行分析。

1. 在地图中选中一个或多个对象节点。
2. 在工具栏点击**发至图谱**图标，即可把地图的对象节点导入到图谱中去。

8.2.8 研判大厅

展示用户的个人分析和共享分析，并且可以对分析做一系列操作。 包含全部分析、个人文件、我的分享、收到的分享 以及按照条件进行搜索等模块。

8.2.8.1 全部分析

按照创建时间顺序展现登录用户所有的个人分析和共享分析。

登录关系网络分析工作台，点击页面上方的**研判大厅** > **全部**，进入全部分析页面，如图 *8-278: 全部分析页面*所示。

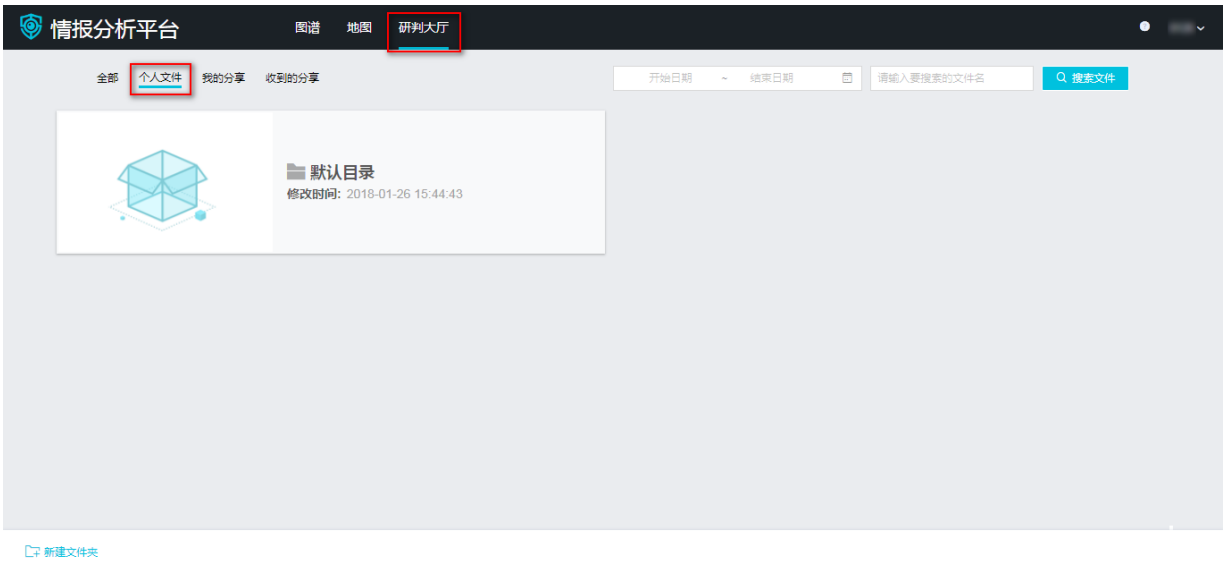
图 8-278: 全部分析页面

8.2.8.2 个人文件

按照创建时间顺序展现所有的个人目录，目录里展示个人分析，可对分析文件进行增、删、查、改等操作。

登录关系网络分析工作台，点击页面上方的**研判大厅** > **个人文件**，进入个人文件页面，如图 *8-279: 个人文件页面*所示。

图 8-279: 个人文件页面

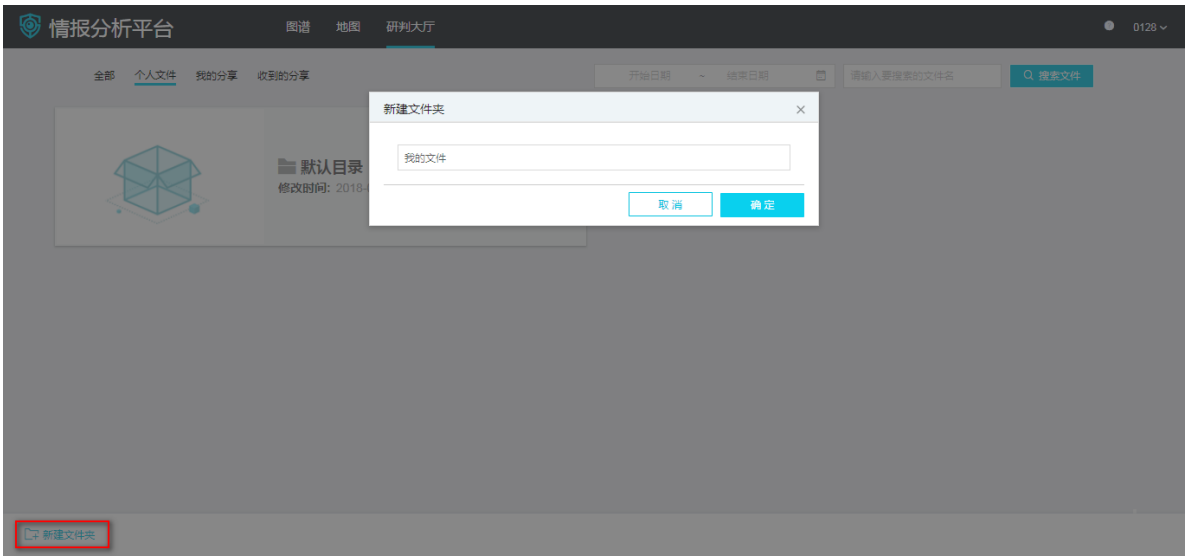


个人文件夹页面存在一个默认目录，不支持重命名和删除操作。

新建文件夹

1. 单击页面左下角的**新建文件夹**按钮。
2. 在弹出的新建文件夹窗口中，输入文件夹名称，如图 *8-280: 新建文件夹*所示。

图 8-280: 新建文件夹



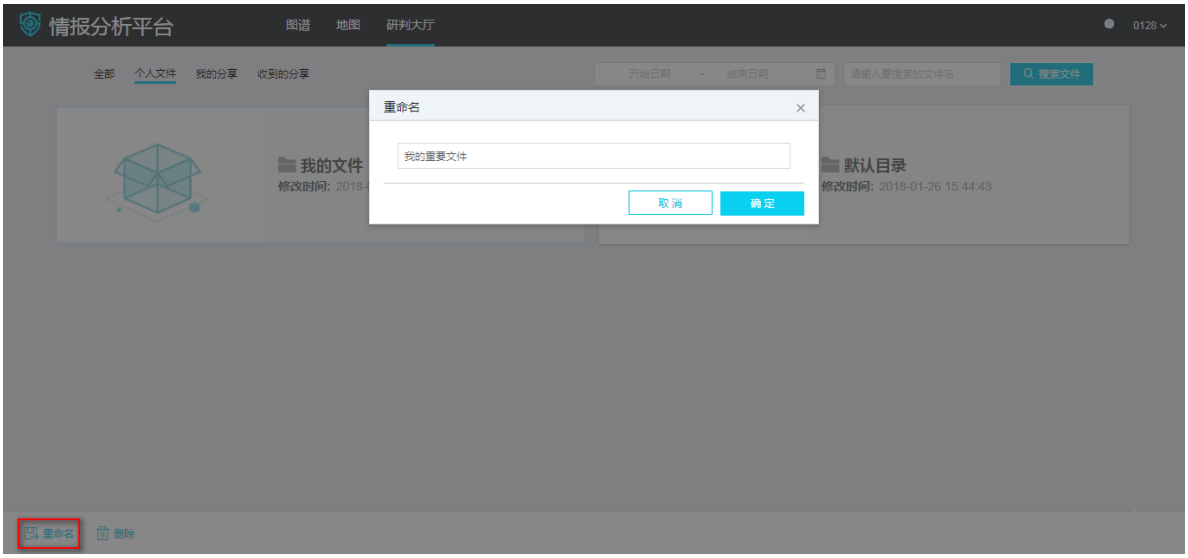
3. 单击**确定**按钮，完成新建文件夹操作。

重命名文件夹

1. 选中待重命名的文件夹。

2. 单击页面下方的**重命名**按钮。
3. 在弹出的重命名窗口中，输入新的文件夹名称，如图 *8-281: 重命名文件夹*所示。

图 8-281: 重命名文件夹

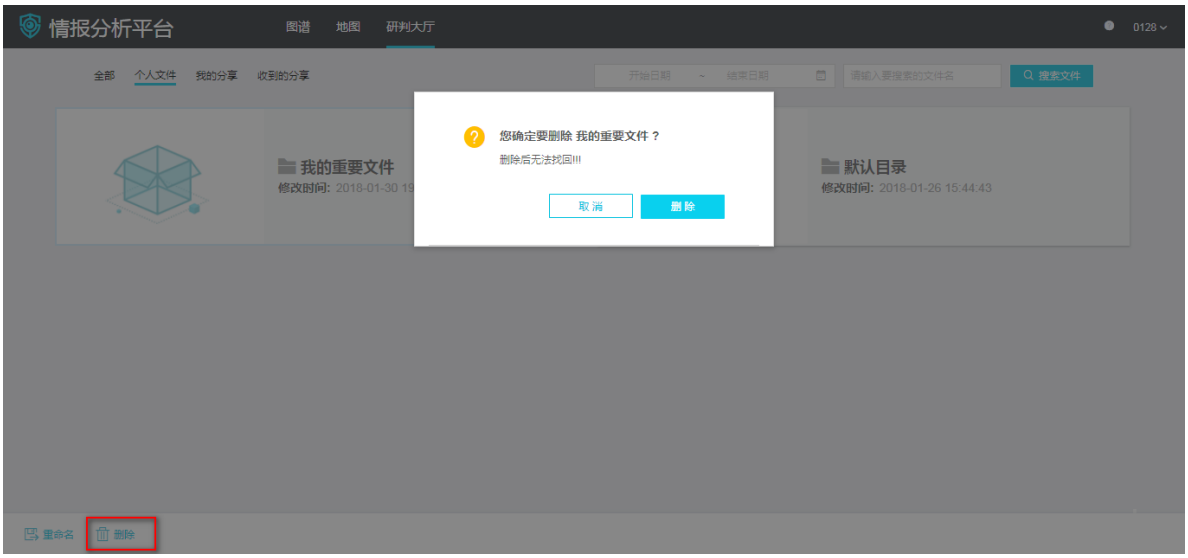


4. 单击**确定**按钮，完成重命名文件夹操作。

删除文件夹

1. 选中待删除的文件夹。
2. 单击页面下方的**删除**按钮，弹出删除窗口。
3. 单击**删除**按钮，完成删除文件夹操作，如图 *8-282: 删除文件夹*所示。

图 8-282: 删除文件夹



用户新建分析后可保存至个人目录下，进入个人目录后可对分析文件进行管理。

重命名分析

1. 选中待重命名的分析文件。
2. 单击页面下方的**重命名**按钮。
3. 在弹出的重命名窗口中，输入新的分析文件名称。
4. 单击**确定**按钮，完成重命名分析文件操作。

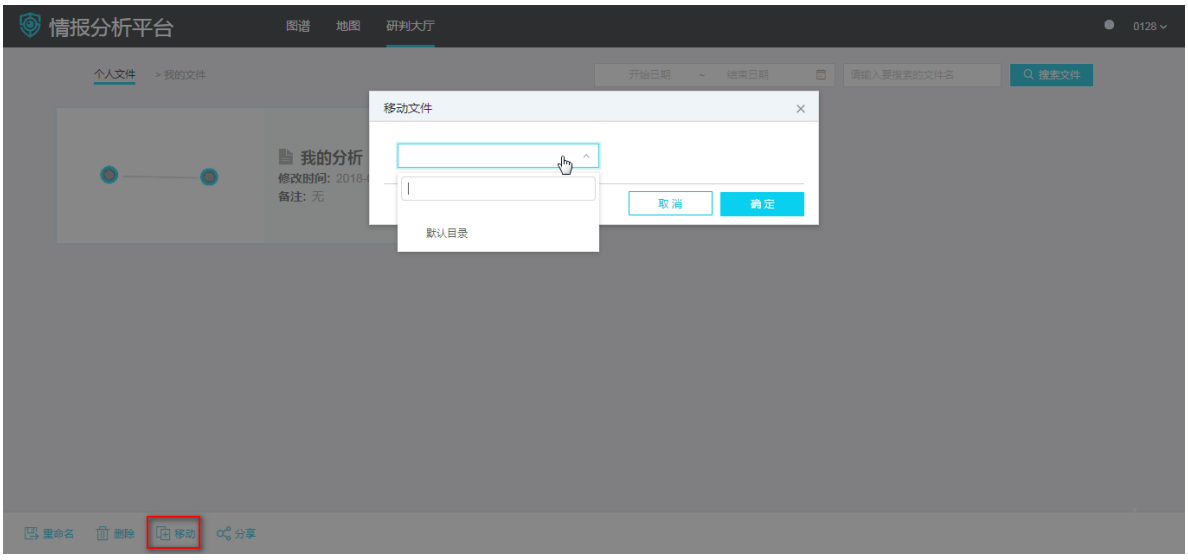
删除分析

1. 选中待删除的分析文件。
2. 单击页面下方的**删除**按钮，弹出删除窗口。
3. 单击**删除**按钮，完成删除分析文件操作。

移动分析

1. 选中待移动的分析文件。
2. 单击页面下方的**移动**按钮。
3. 在弹出的移动文件窗口中，选择目标目录，如图 *8-283: 移动分析文件*所示，可输入目录名称进行快速查询。

图 8-283: 移动分析文件



4. 单击**确定**按钮，完成移动分析文件操作。

分享分析

1. 选中待分享的分析文件。
2. 单击页面下方的**分享**按钮。
3. 在弹出的共享分析文件窗口中，选择要共享的成员，如图 *8-284:* 选择共享成员所示。

图 8-284: 选择共享成员

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。

4. 单击**分享**按钮，完成分享分析文件操作。

8.2.8.3 我的分享

I+支持将个人的分析共享给其他人，即将个人的思路和经验传递给其他分析用户，同时也支持将共享分析用户的智慧和经验统一起来，形成多人协作共享，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。

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

I+支持对共享分析文件的管理，包括删除、重命名、历史版本管理等操作。

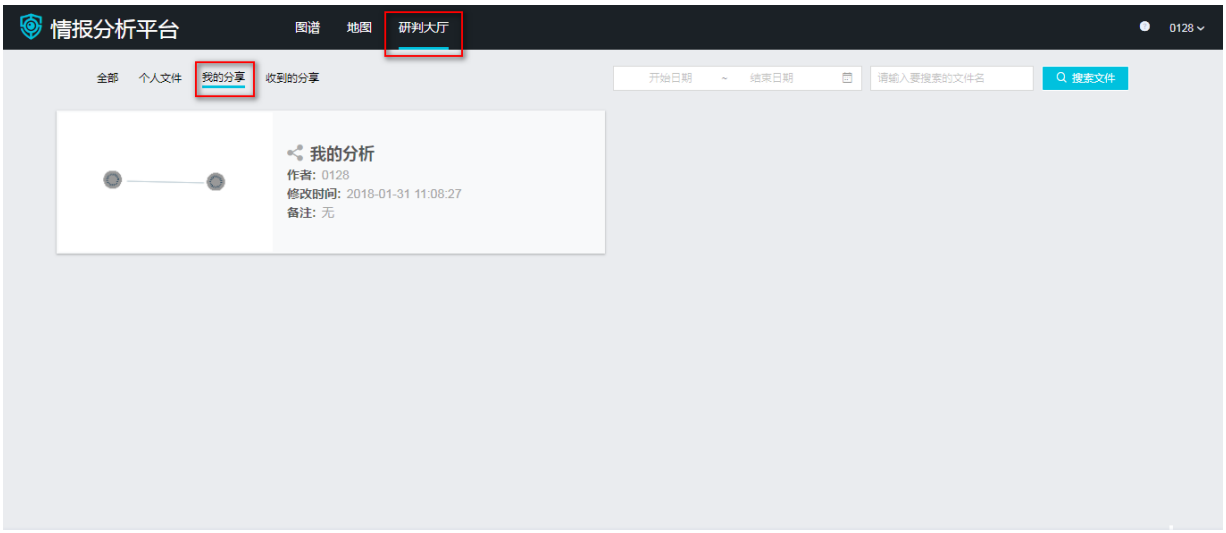
说明：
删除和重命名操作仅限发起者进行操作。

8.2.8.3.1 管理分享分析文件夹

我的分享页面是按照创建时间顺序展现登录用户分享的分析文件，有关分享分析文件的方法，请参见分享分析。

登录关系网络分析工作台，点击页面上方的**研判大厅** > **我的分享**，进入我的分享页面，如图 *8-285: 我的分享页面*所示。

图 8-285: 我的分享页面



更改分享权限

1. 选中待更改分享权限的分析文件夹。
2. 单击页面下方的**更改分享权限**按钮。
3. 在弹出的共享成员设置窗口中，添加或删除共享成员，如图 *8-286: 更改分享权限*所示。

图 8-286: 更改分享权限

4. 单击**分享**按钮，完成共享分析文件更改分享权限操作。

删除共享分析文件夹

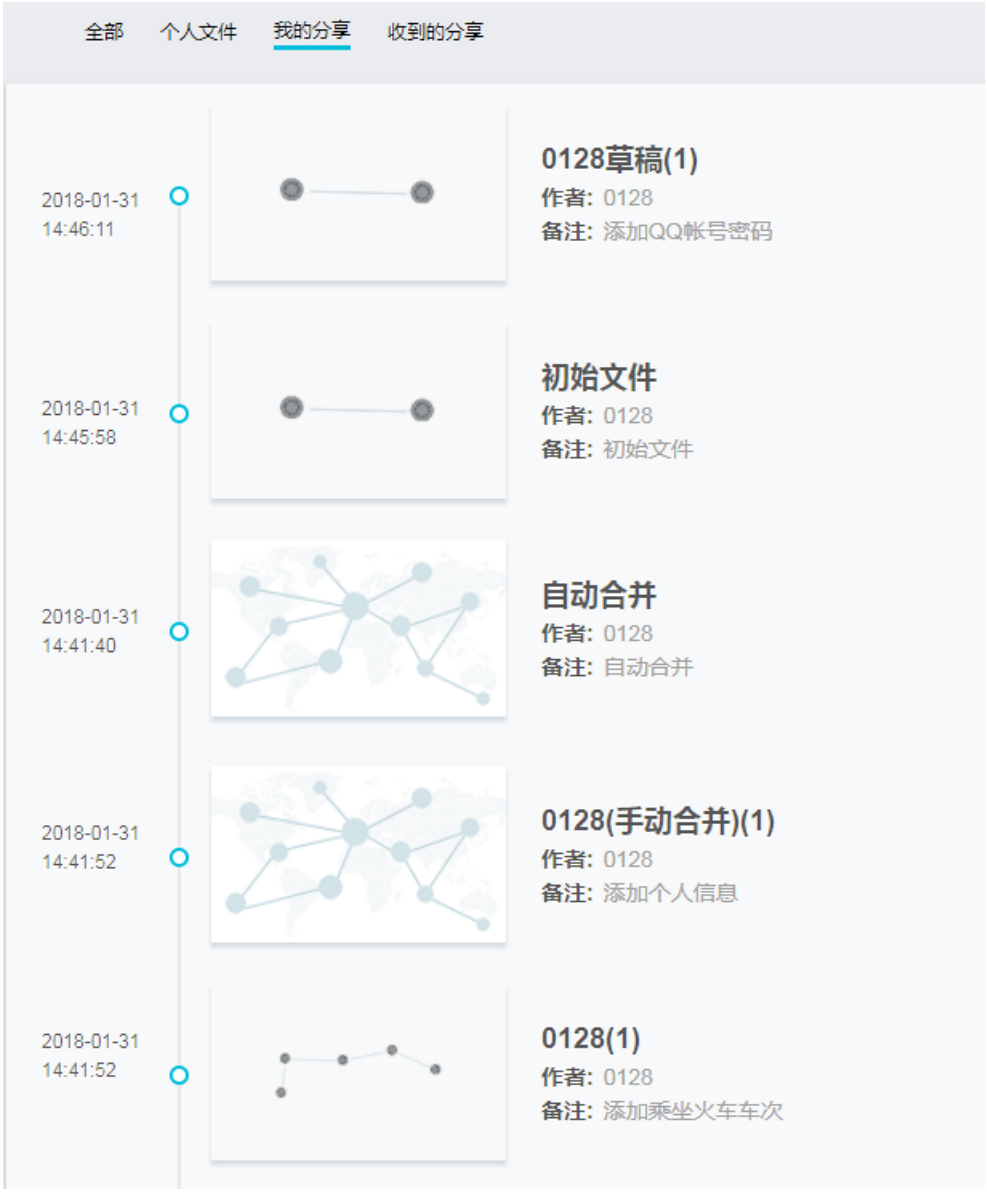
1. 选中待删除的分析文件夹。
2. 单击页面下方的**删除**按钮，弹出删除窗口。
3. 单击**删除**按钮，完成删除共享分析文件夹操作。

8.2.8.3.2 管理分享分析文件目录

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。

分享分析文件夹下的目录显示如图 *8-287: 分享分析文件夹目录*所示。

图 8-287: 分享分析文件夹目录



从上到下依次显示为：

- 草稿文件
- 初始文件
- 自动合并文件
- 手动合并文件
- 发布版本文件

有关分享分析文件的方法，请参考分享分析。

删除分析文件版本

1. 双击进入分享分析文件夹，默认存在初始文件和自动合并文件。

   初始文件和自动合并文件不支持删除操作。

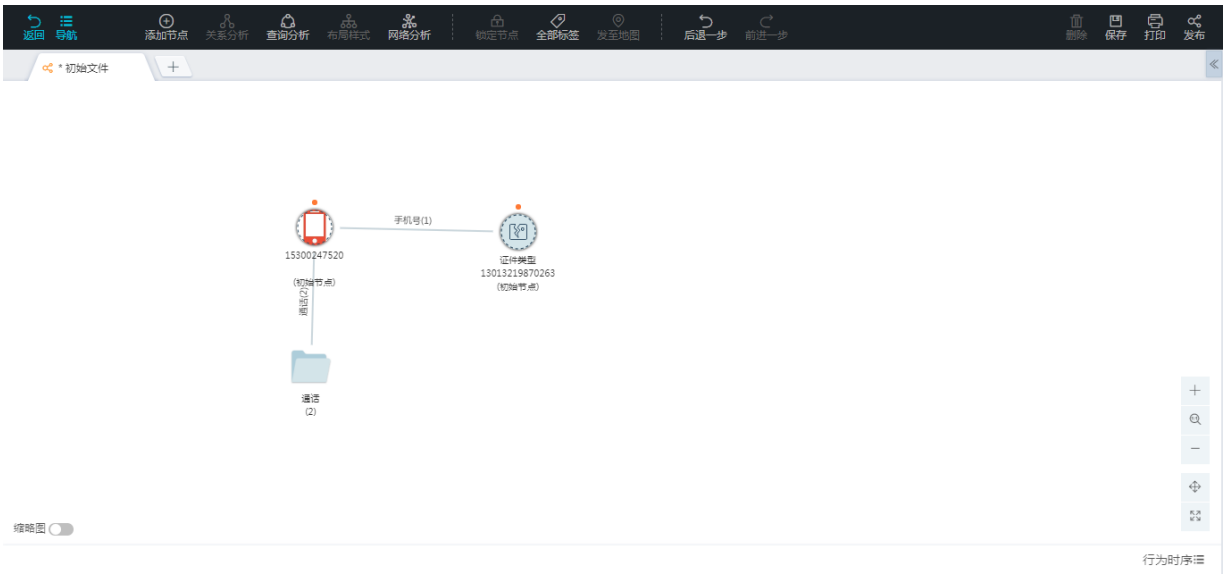
2. 选中待删除的草稿文件、手动合并文件或已发布版本文件。
3. 单击页面下方的**删除**按钮，弹出删除窗口。
4. 单击**删除**按钮，完成删除分析文件版本操作。

8.2.8.3.2.1 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 *8-288: 初始文件*所示。

图 8-288: 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。

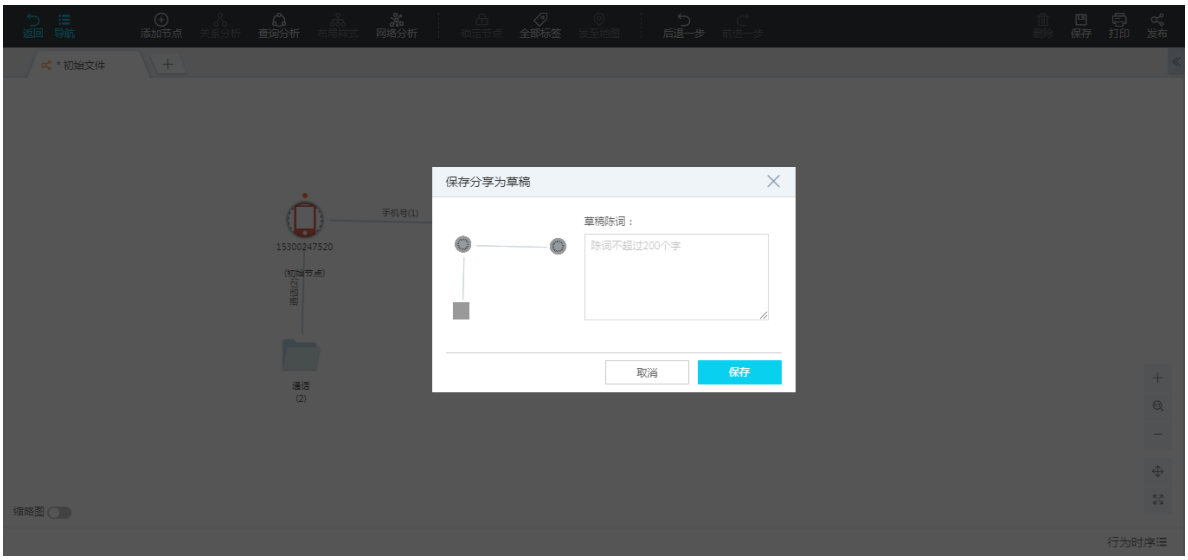
8.2.8.3.2.2 保存草稿

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. 点击页面上方的**保存**图标，弹出保存分享为草稿窗口，如图 *8-289: 保存分享为草稿窗口*所示。

图 8-289: 保存分享为草稿窗口



3. 输入草稿陈词描述，不超过200个字。
4. 单击**保存**按钮。

预期结果

草稿成功保存后，会在该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 *8-290:* 草稿文件所示。

图 8-290: 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)。

- 草稿文件作者：保存该草稿的用户。
- 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：草稿左侧显示的是该草稿文件生成的时间。

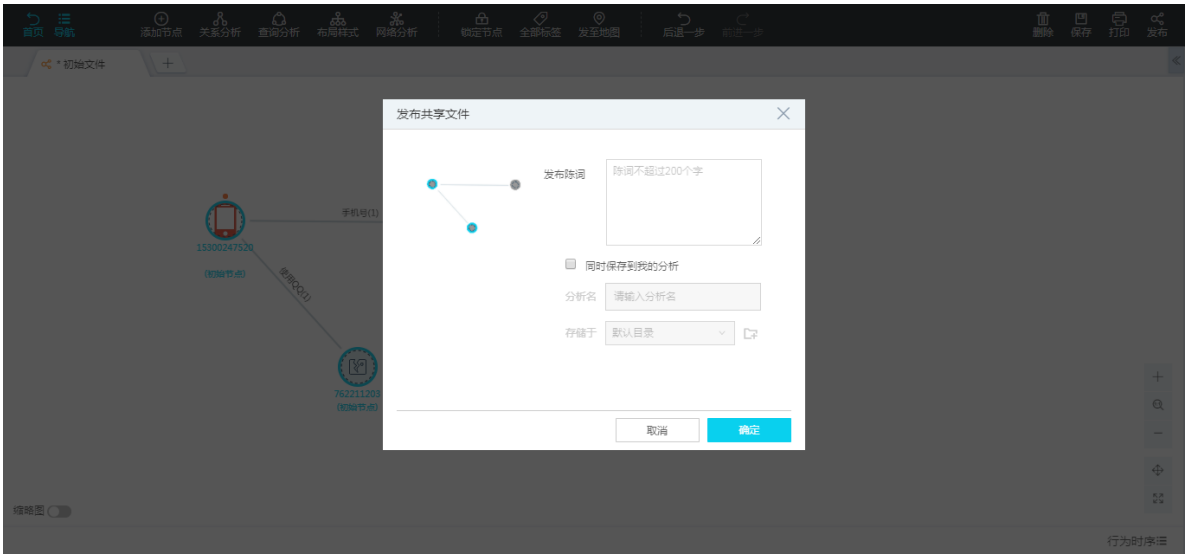
8.2.8.3.2.3 发布版本

对于分享给用户的分析文件，用户对其进行编辑后，可以发布为新版本的分析文件。发布的分析文件，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. 点击页面上方的**发布**图标，弹出发布共享文件窗口，如图 8-291: 发布版本所示。

图 8-291: 发布版本



3. 进行如下操作：
   - 发布陈词：输入发布陈词备注信息，不超过200个字。
   - 同时保存到我的分析：勾选该项后，该分析会显示在我的分析页签下，需输入分析名称和该文件的存储目录。
4. 单击**确定**按钮。

预期结果

文件发布成功后，会在该分享分析目录下显示该发布文件，如图 8-292: 发布文件成功所示。

图 8-292: 发布文件成功

- 发布文件显示位置：位于**自动合并**或**手动合并**下方。
- 发布文件命名格式：当前用户的用户名+（版本号），如test123(1)。
- 发布文件作者：发布该文件的用户。
- 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：发布文件左侧显示的是该发布文件生成的时间。

8.2.8.3.2.4 自动合并

当有新的版本发布的时候，不论是哪个分享成员，还是分享者自己，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。

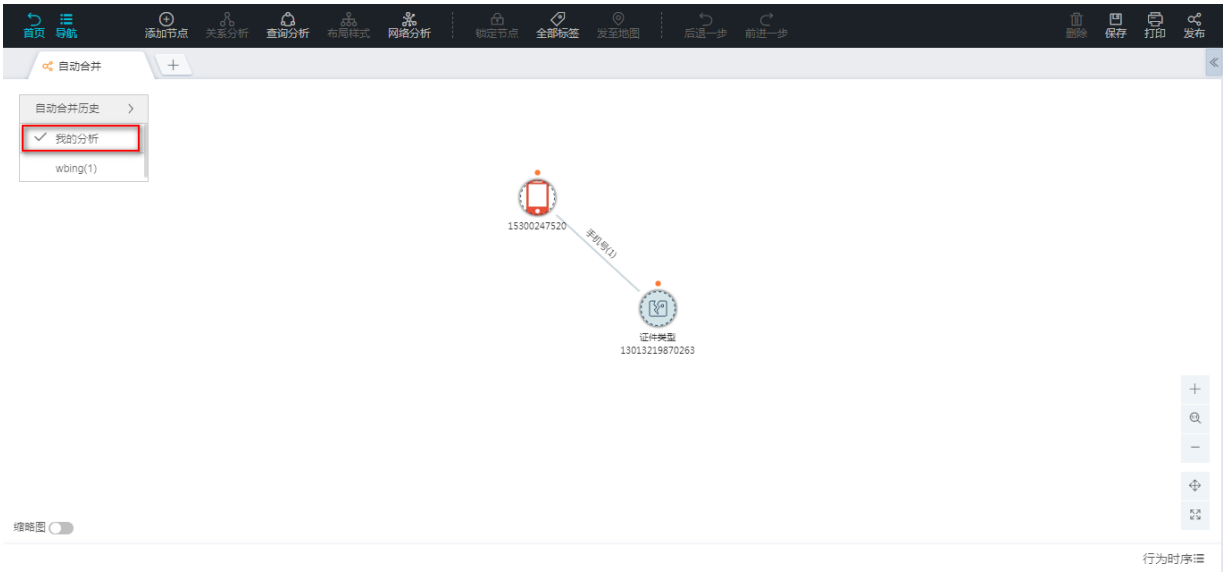
发布了一个新版本后，自动合并文件会有更新，如图 *8-293:* 自动合并文件所示。

图 8-293: 自动合并文件

在图区左上角，可以看到不同版本的列表，默认展示最新的版本。

用户可以选择回溯版本。通过选择**自动合并历史**列表中的不同版本，查看以前的版本，如图 *8-294: 版本回溯*所示。

图 8-294: 版本回溯



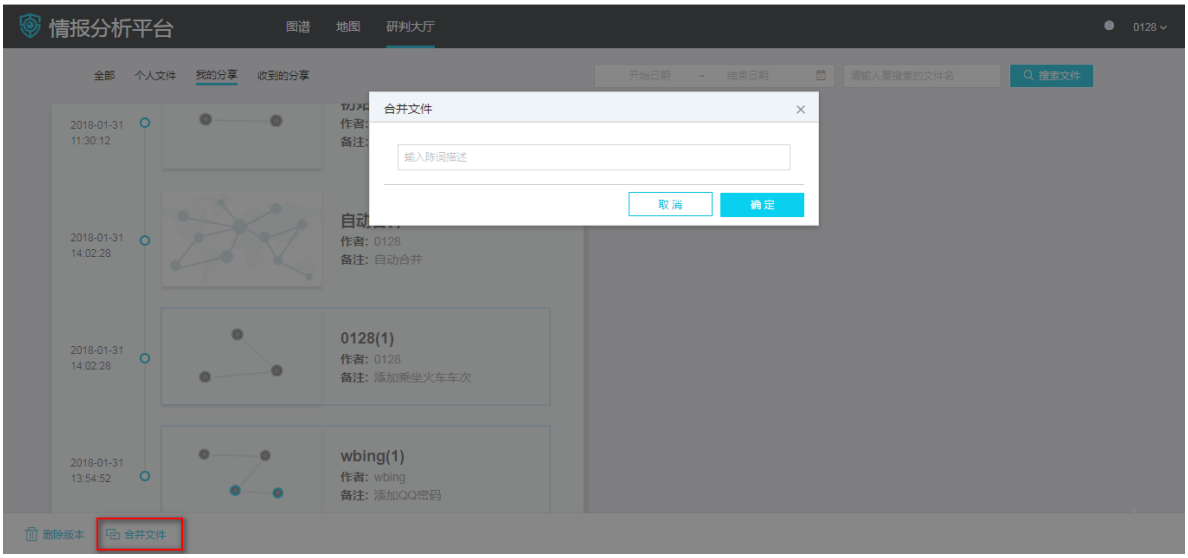
8.2.8.3.2.5 手动合并

用户可以将同一个分享分析的多个版本进行合并。只有发布的版本文件支持合并。

操作步骤

1. 双击进入分享分析文件夹，选中两个或多个已发布的版本文件，可以通过键盘上的**Control**键来选择。
2. 点击页面左下角**合并文件**图标，弹出合并文件窗口，如图 *8-295: 合并文件窗口*所示。

图 8-295: 合并文件窗口



3. 输入陈词描述信息后，单击**确定**按钮。

预期结果

文件合并成功后，如图 *8-296: 查看结果*所示。

图 8-296: 查看结果

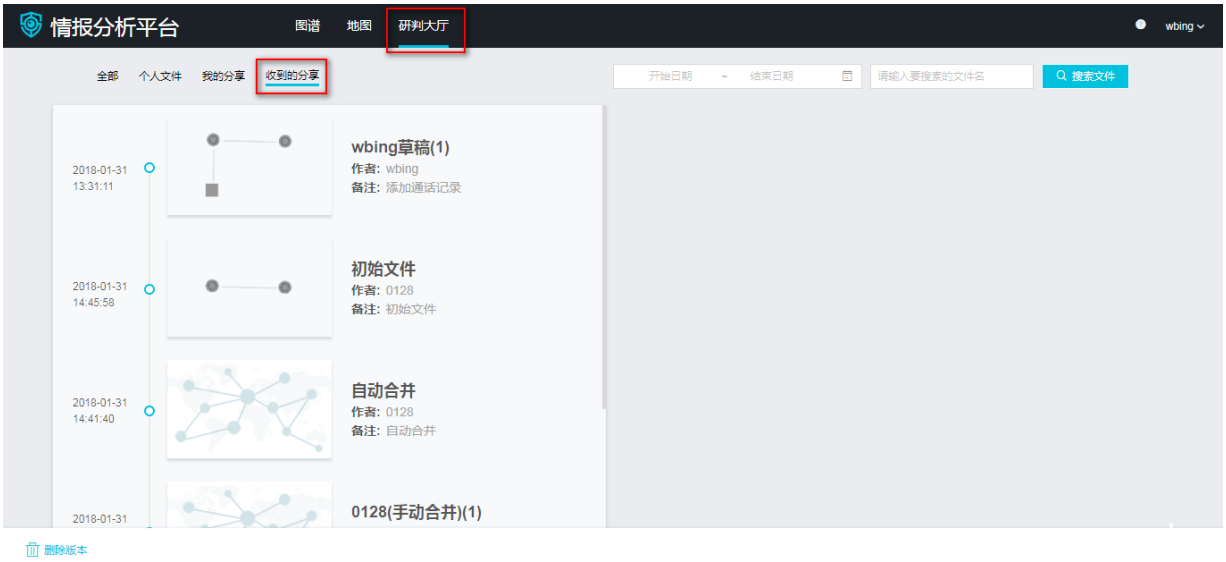
当版本中有不一致的地方，取并集。

8.2.8.4 收到的分享

用户在收到的分享页面可查看其他用户共享给自己的分析文件，从而对分析文件进行修改操作，不支持删除共享文件操作。

登录关系网络分析工作台，点击页面上方的**研判大厅** > **收到的分享**，进入收到的分享页面，如图 *8-297: 收到的分享页面*所示。

图 8-297: 收到的分享页面



8.2.8.5 搜索分析文件

I+支持用户通过时间或者关键字对分析文件进行搜索，以方便用户在大量分析文件中快速定位到所需要的分析文件。

1. 登录关系网络分析工作台。
2. 点击页面上方的**研判大厅**，默认进入个人文件页签页面。
3. 用户可根据实际需要点击切换到全部、个人文件、我的分享或收到的分享页签页面。
4. 在页面右上方选择起始时间范围、输入关键字，单击**搜索**按钮，即可搜索出符合条件的分析文件。

8.2.9 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“iplus1688”，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：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。

10. 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。

11. 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。

12. 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。

13. 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。

14. 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。

15.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。

9 Quick BI

9.1 什么是Quick BI

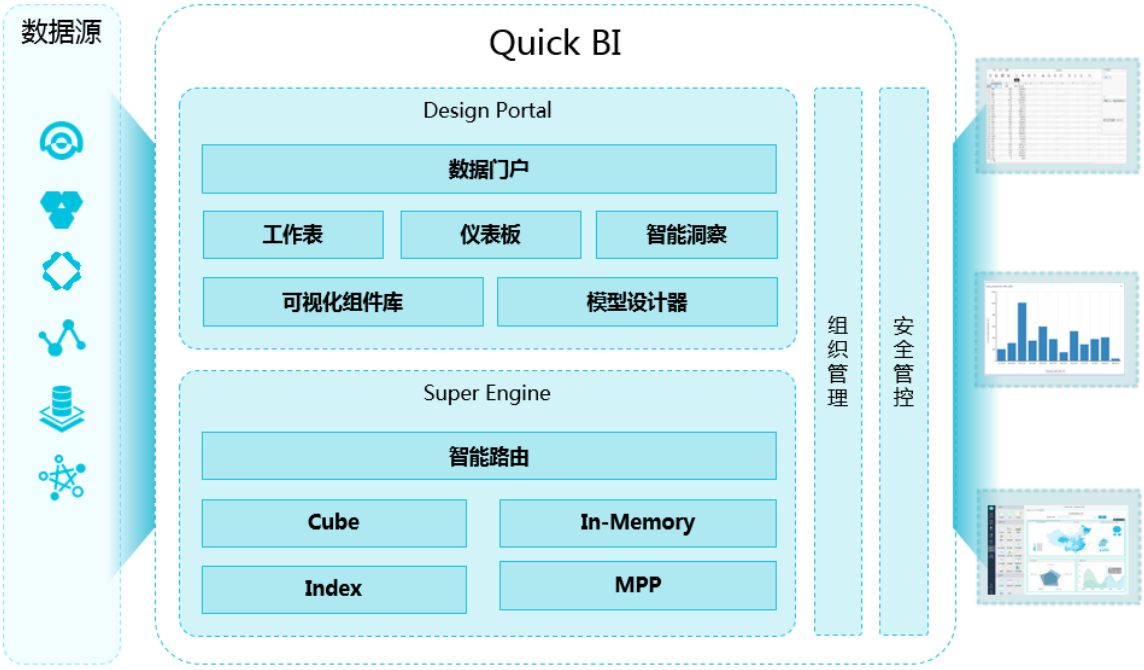
本章节将为您介绍Quick BI的产品特点。

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，例如连接MaxCompute (ODPS)、RDS、Analytic DB、HybirdDB (Greenplum) 等云数据源，连接ECS上您自建的MySQL数据库和SQL Server数据库，也支持连接VPC数据源。

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 9-1: Quick BI 产品架构



9.2 登录Quick BI

本章节将为您介绍如何登录Quick BI。

背景信息

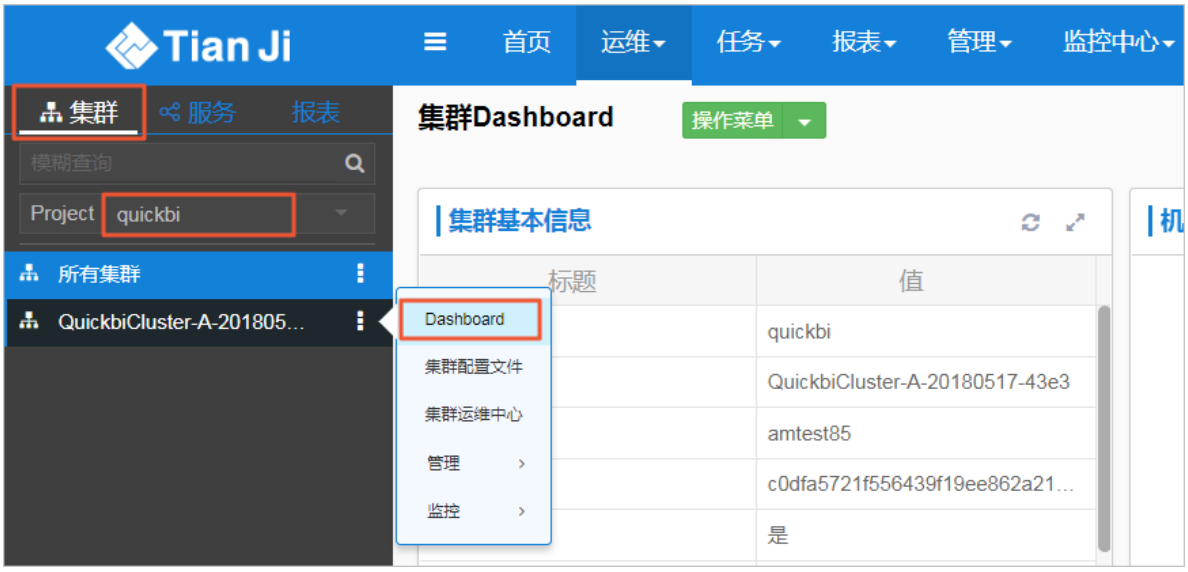
您需要先从天基上获取Quick BI的域名，然后用获取后的域名登录Quick BI控制台。

操作步骤

1. 登录天基，如图 *9-2: 天基*所示。

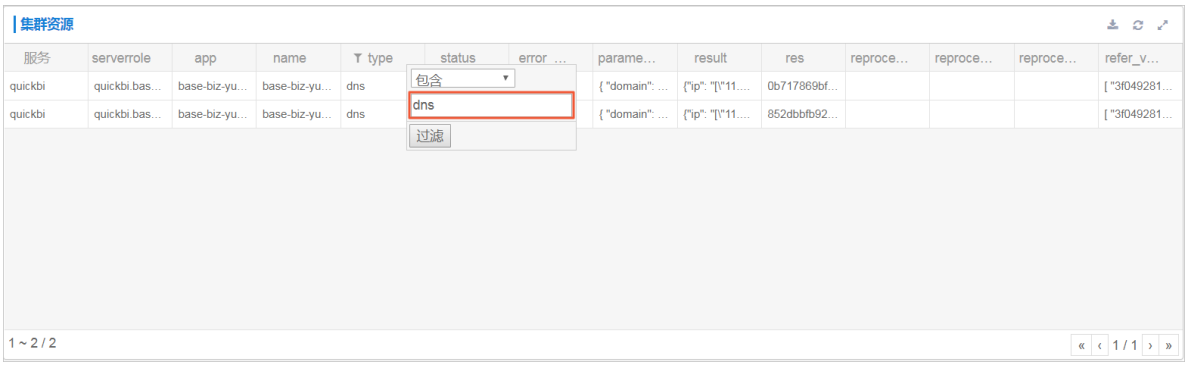
   天基

   图 9-2: 天基



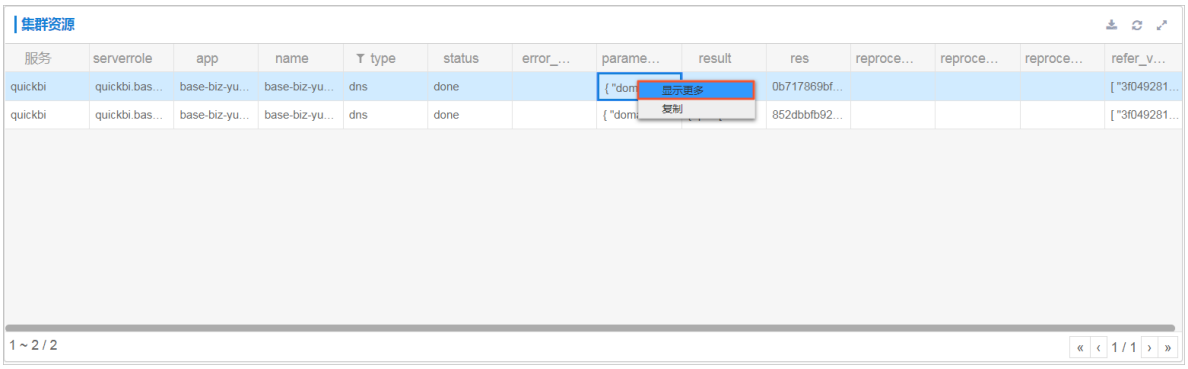
2. 选择**集群**标签页， 然后在Project下拉列表中选择**quickbi** 。
3. 找到筛选出的项目，单击鼠标右键，然后选择 **Dashboard**。
4. 在**集群资源**列表中，单击列表的**type**表头，以dns进行筛选，如图 *9-3: 集群资源*所示。

   图 9-3: 集群资源



5. 在筛选结果中选择一个目标集群，在parameters选项中，单击鼠标右键，然后选择**显示更多**，如图 *9-4: 显示更多*所示。

图 9-4: 显示更多



6. 域名地址会自动列在页面上方，如图 *9-5:* 查看域名所示。

图 9-5: 查看域名

7. 复制Quick BI域名到浏览器地址栏，按照系统提示登录Quick BI控制台。

9.3 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：

1. 新建数据源
2. 选择数据源中的表来创建数据集
3. 通过自定义SQL来创建数据集（可选）

9.3.1 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。

- 来自云数据库（MaxCompute，MySQL，SQL Server，AnalyticDB，HybirdDB for MySQL，HybirdDB for PostgreSQL，PostgreSQL，PPAS）
- 来自自建数据源（MySQL，SQL Server，Oracle）
- 来自VPC数据源

说明：

SQL Server暂不支持使用视图。

9.3.1.1 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 *9-6:* 数据源列表所示。

图 9-6: 数据源列表

9.3.1.2 新建数据源

背景信息

数据集、工作表、电子表格、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**工作空间**，进入工作空间管理页面。
2. 单击**数据源**标签页进入数据源管理页面。
3. 单击**新建**，如图 *9-7:* 新建数据源所示。

图 9-7: 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。
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**。

9.3.1.2.1 来自云数据库

本章节将为您介绍来自云数据库数据源的新建流程。

9.3.1.2.1.1 MaxCompute

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 *9-8:* 添加数据源所示。

图 9-8: 添加数据源

2. 输入所需要的数据源连接信息。

   - 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请登录大数据管家的云账号管理来获取Access ID
   - Access Key：阿里云官网管控台Access Key Secret

   请登录大数据管家的云账号管理来获取Access ID和Access Key Secret。更多云账号管理，请参阅，云账号管理。

3. 单击**连接测试**，进行数据源连通性测试，如图 *9-9:* 连接测试所示。

图 9-9: 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。

9.3.1.2.1.2 MySQL

背景信息

受限于RDS的白名单策略，在添加MySQL数据源时，您需要先查询到可用的IP地址，手动添加到RDS控制台上。

操作步骤

1. 登录Quick BI控制台。
2. 单击**数据源** > **新建**。
3. 选择MySQL，输入所需要的数据源连接信息，如图 *9-10: 添加 MySQL 数据源*所示。

图 9-10: 添加 MySQL 数据源

- 显示名称：输入数据源显示名称（可自定义）
- 数据库地址：输入主机名或IP地址
- 端口号：输入端口号，默认为3306
- 数据库：输入数据库名称
- 用户名：输入用户名
- 密码：输入密码

> **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。

- 如果数据源来自VPC，请勾选VPC数据源。

4. 单击**连接测试**，进行数据源连通性测试。

5. 连通性测试成功后，单击**添加**。

   数据源不能重复添加，如果该数据源已经存在，系统会给出错误提示。

9.3.1.2.1.3 SQL Server

背景信息

受限于RDS的白名单策略，在添加SQL Server数据源时，您需要先查询到可用的IP地址，手动添加到RDS控制台上。

SQL Server数据源的配置方法与MySQL数据源的配置方法类似。多了一个SQL Server数据源特有的配置项：schema。

操作步骤

1. 单击SQL Server图标，输入所需要的数据源连接信息，如图 *9-11:* 添加 *SQL Server*数据源所示。

图 9-11: 添加 SQL Server数据源

- 显示名称：输入数据源显示名称（可自定义）
- 数据库地址：输入主机名或IP地址
- 端口号：输入端口号，默认为1433
- 数据库：输入数据库名称
- Schema：dbo
- 用户名：输入用户名
- 密码：输入密码
- 如果数据源来自VPV，请勾选来自VPC。

2. 单击**连接测试**，进行数据源连通性测试。

3. 连接性测试成功后，单击**添加**。

9.3.1.2.1.4 Analytic DB

背景信息

Analytic DB也叫ADS。

操作步骤

1. 单击Analytic DB图标，输入所需要的数据源连接信息，如图 *9-12: 添加 Analytic DB 数据源*所示。

图 9-12: 添加 Analytic DB 数据源

- 显示名称：输入数据源显示名称（可自定义）
- 数据库地址：输入主机名或IP地址
- 端口号：输入端口号, 默认是3306
- 数据库：输入数据库名称
- Access Id：输入阿里云官网管控台Access Key ID
- Access Key：输入阿里云官网管控台Access Key Secret

2. 单击**连接测试**按钮，进行数据源连通性测试。
3. 连接性测试成功后，单击**添加**。

9.3.1.2.1.5 HybirdDB for MySQL

背景信息

在添加HybirdDB for MySQL数据源时，您需要先查询到可用的IP地址，手动添加到HybirdDB for MySQL控制台上。

HybirdDB for MySQL数据源的配置方法与MySQL数据源的配置方法类似。默认端口会变成HybirdDB for MySQL特有的端口。

操作步骤

1. 单击HybirdDB for MySQL，输入所需要的数据源连接信息，如图 *9-13: 添加HybirdDB for MySQL数据源*所示。

图 9-13: 添加HybirdDB for MySQL数据源

- 显示名称：输入数据源显示名称（可自定义）
- 数据库地址：输入主机名或IP地址

- 端口号：输入端口号，默认为3306
- 数据库：输入数据库名称
- 用户名：输入用户名
- 密码：输入密码

2. 单击**连接测试**，进行数据源连通性测试。
3. 连接性测试成功后，单击**添加**。

9.3.1.2.1.6 HybirdDB for PostgreSQL

背景信息

HybridDB for PostgreSQL数据源的添加方法与SQL Server数据源的添加方法类似。默认端口会变成HybridDB for PostgreSQL特有的端口。

操作步骤

1. 单击HybridDB for PostgreSQL，输入所需要的数据源连接信息，如图 *9-14: 添加HybirdDB for PostgreSQL* 所示。

图 9-14: 添加HybirdDB for PostgreSQL

- 显示名称：输入数据源显示名称（可自定义）
- 数据库地址：输入主机名或IP地址
- 端口号：输入端口号，默认为5432
- 数据库：输入数据库名称
- Schema：public
- 用户名：输入用户名
- 密码：输入密码

2. 单击**连接测试**，进行数据源连通性测试。
3. 连接性测试成功后，单击**添加**。

9.3.1.2.1.7 PostgreSQL

背景信息

PostgreSQL数据源的添加方法与HybirdDB for PostgreSQL数据源的添加方法类似。

操作步骤

1. 单击PostgreSQL，输入所需要的数据源连接信息，如图 *9-15: 添加 PostgreSQL 数据源*所示。

图 9-15: 添加 PostgreSQL 数据源

- 显示名称：输入数据源显示名称（可自定义）
- 数据库地址：输入主机名或IP地址
- 端口号：输入端口号，默认为5432
- 数据库：输入数据库名称

- Schema：public
- 用户名：输入用户名
- 密码：输入密码

2. 单击**连接测试**，进行数据源连通性测试。
3. 连接性测试成功后，单击**添加**。

9.3.1.2.1.8 PPAS

背景信息

PPAS数据源的添加方法与PostgreSQL数据源的添加方法类似。

操作步骤

1. 单击PPAS，输入所需要的数据源连接信息，如图 *9-16:* 添加*PPAS*数据源所示。

图 9-16: 添加PPAS数据源

- 显示名称：输入数据源显示名称
- 数据库地址：输入主机名或IP地址
- 端口号：输入端口号，默认为5432
- 数据库：输入数据库名称
- Schema：public
- 用户名：输入用户名
- 密码：输入密码

2. 单击**连接测试**，进行数据源连通性测试。
3. 连接性测试成功后，单击**添加**。

9.3.1.2.2 来自自建数据源

本章节将为您介绍如何用Quick BI添加自建数据库下的数据源。

9.3.1.2.2.1 MySQL

背景信息

自建数据源下的MySQL数据源的配置方法与云数据库中的MySQL数据源的配置方法类似。

操作步骤

1. 单击MySQL，输入所需要的数据源连接信息，如图 *9-17: 添加MySQL数据源*所示。

图 9-17: 添加MySQL数据源

- 显示名称：输入数据源显示名称
- 数据库地址：输入主机名或IP地址
- 端口：输入端口地址，默认为3306
- 数据库：输入数据库名称

- 用户名：输入用户名
- 密码：输入密码

2. 单击**连接测试**，进行数据源连通性测试。
3. 连接性测试成功后，单击**添加**。
4. 您需要打开自建数据源下的防火墙，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
   ```

9.3.1.2.2.2 SQL Server

背景信息

自建数据源下的SQL Server数据源的配置方法与云数据库下的SQL Server数据源的配置方法类似。

操作步骤

1. 单击SQL Server ，输入所需要的数据源连接信息，如图 *9-18:* 添加*SQL Server*数据源所示。

图 9-18: 添加SQL Server数据源

- 显示名称：输入数据源显示名称
- 数据库地址：输入主机名或IP地址
- 端口：输入端口地址，默认为1433
- 数据库：输入数据库名称
- Schema：dbo
- 用户名：输入用户名
- 密码：输入密码

2. 单击**连接测试**按钮，进行数据源连通性测试。
3. 连接性测试成功后，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。

9.3.1.2.2.3 PostgreSQL

背景信息

自建数据源下的PostgreSQL数据源的配置方法与云数据库中的PostgreSQL数据源的配置方法类似。

操作步骤

1. 单击PostgreSQL，输入所需要的数据源连接信息，如图 *9-19: 添加PostgreSQL数据源*所示。

图 9-19: 添加PostgreSQL数据源

- 显示名称：输入数据源显示名称
- 数据库地址：输入主机名或IP地址
- 端口：输入端口地址，默认为5432
- 数据库：输入数据库名称
- Schema：public
- 用户名：输入用户名
- 密码：输入密码

2. 单击**连接测试**按钮，进行数据源连通性测试。
3. 连接性测试成功后，单击**添加**。

9.3.1.2.2.4 Oracle

操作步骤

1. 单击Oracle，输入所需要的数据源连接信息，如图 *9-20: 添加Oracle数据源*所示。

图 9-20: 添加Oracle数据源

- 显示名称：输入数据源显示名称
- 数据库地址：输入主机名或IP地址
- 端口：输入端口地址，默认为1521
- 数据库：输入数据库名称
- Schema：输入Schema值
- 用户名：输入用户名
- 密码：输入密码

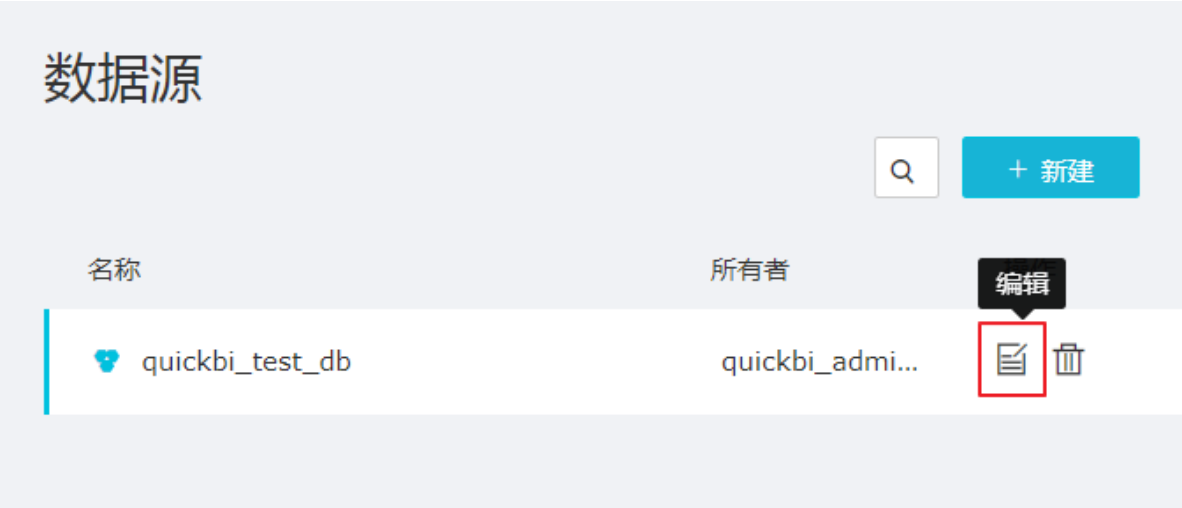
2. 单击**连接测试**按钮，进行数据源连通性测试。
3. 连接性测试成功后，单击**添加**。

9.3.1.3 编辑数据源

操作步骤

1. 在数据源列表中，选择一个数据源。
2. 单击**编辑**，可以对该数据源进行编辑，如图 *9-21: 编辑数据源*所示。

图 9-21: 编辑数据源



9.3.1.4 查询数据源

操作步骤

1. 在Quick BI控制台单击**数据源**，进入数据源管理页面。
2. 在搜索框中输入关键字查询需要的数据源，如图 *9-22: 查询数据源*所示。

图 9-22: 查询数据源

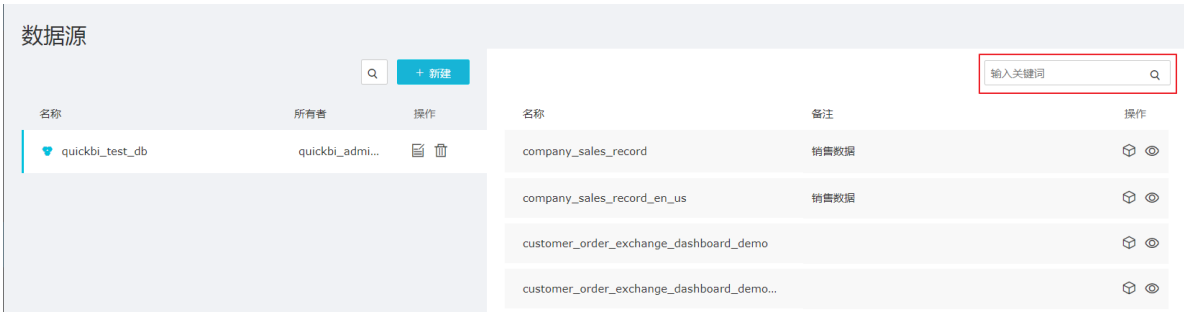
3. 单击**查询**图标，进行数据源查询。

9.3.1.5 查询数据源中包含的表

操作步骤

1. 在Quick BI控制台，单击**数据源**，进入数据源管理页面。
2. 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。
3. 在右侧页面的搜索框中输入关键字段搜索需要的表，如图 *9-23:* 查询数据源中包含的表所示。

图 9-23: 查询数据源中包含的表



4. 单击**查询**图标，进行表查询。

9.3.1.6 查询数据源中表的详情

操作步骤

1. 在Quick BI控制台，单击**数据源**，进入数据源管理页面。

2. 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。
3. 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 *9-24: 查询数据源中表的详情*所示。

图 9-24: 查询数据源中表的详情

9.3.1.7 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 *9-25: 删除数据源*所示。

图 9-25: 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，如图 *9-26: 删除*所示。

图 9-26: 删除

9.3.2 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。

9.3.2.1 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

9.3.2.1.1 从数据源创建

操作步骤

1. 在Quick BI控制台，单击**数据源**图标，进入数据源管理页面。
2. 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。
3. 选中其中一个表，单击后面的**创建数据集**，如图 9-27: 从数据源创建所示。

图 9-27: 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 *9-28: 新添加数据集*所示。

图 9-28: 新添加数据集

9.3.2.1.2 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。

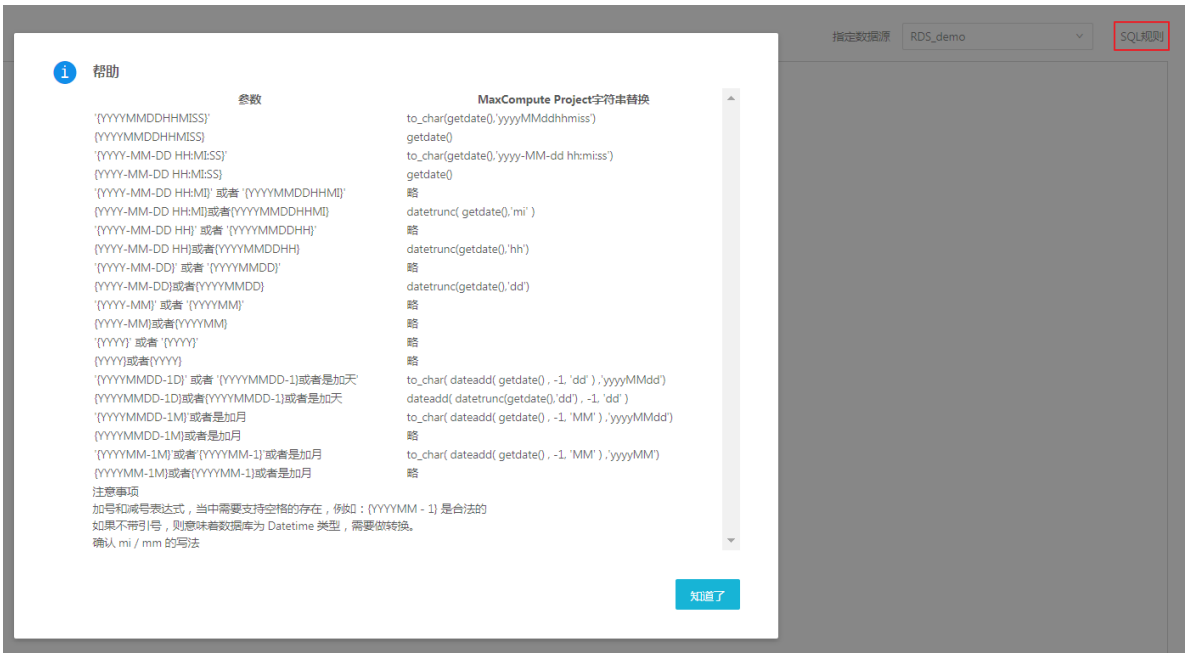
操作步骤

1. 在Quick BI控制台，单击**数据源**，进入数据源管理页面。
2. 在列表中选择一个来自MaxCompute的数据源。
3. 在页面右侧，单击**自定义SQL**来创建数据集，如图 *9-29: 自定义SQL*所示。

图 9-29: 自定义SQL

4. 编辑数据源内容，如需帮助，可单击 **SQL规则**，打开SQL帮助页面，如图 *9-30: SQL规则*所示。

图 9-30: SQL规则



5. 单击**保存**，将该数据源直接保存为数据集。

9.3.2.2 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。在配置面板的**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**配置面板**图标，进入组织管理页面，如图 *9-31:* 配置面板所示。

图 9-31: 配置面板

2. 单击**偏好设置**标签页，选择数据集字段的的默认名称，如图 *9-32:* 设置数据集默认名称所示。

图 9-32: 设置数据集默认名称

9.3.2.3 编辑数据集

操作步骤

1. 在Quick BI控制台，单击**数据集**图标，进入数据集管理页面。
2. 选择一个数据集，单击数据集名称进入数据集编辑页面。

9.3.2.3.1 编辑维度

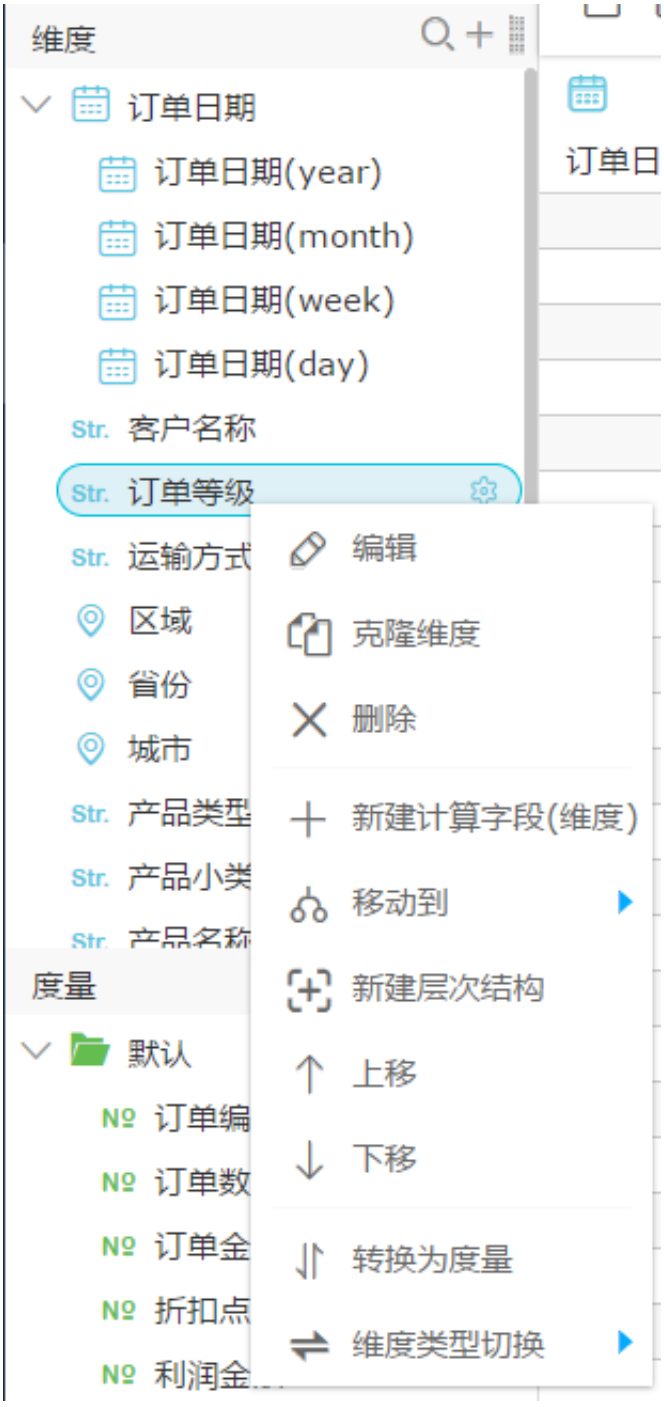
背景信息

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，编辑菜单如下图所示。

图 9-33: 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。

9.3.2.3.2 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 *9-34:* 度量编辑菜单所示。

图 9-34: 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。

9.3.2.3.3 工具按钮

Quick BI提供以下工具按钮，如图 *9-35:* 工具按钮区所示，供您来编辑数据集。

图 9-35: 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 *9-36: 新建*所示。

图 9-36: 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 *9-37: 同步*所示。

图 9-37: 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 *9-37: 同步表结构*所示。

图 9-37: 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 *9-39: 保存*所示。

图 9-39: 保存

- 保存：保存数据集。

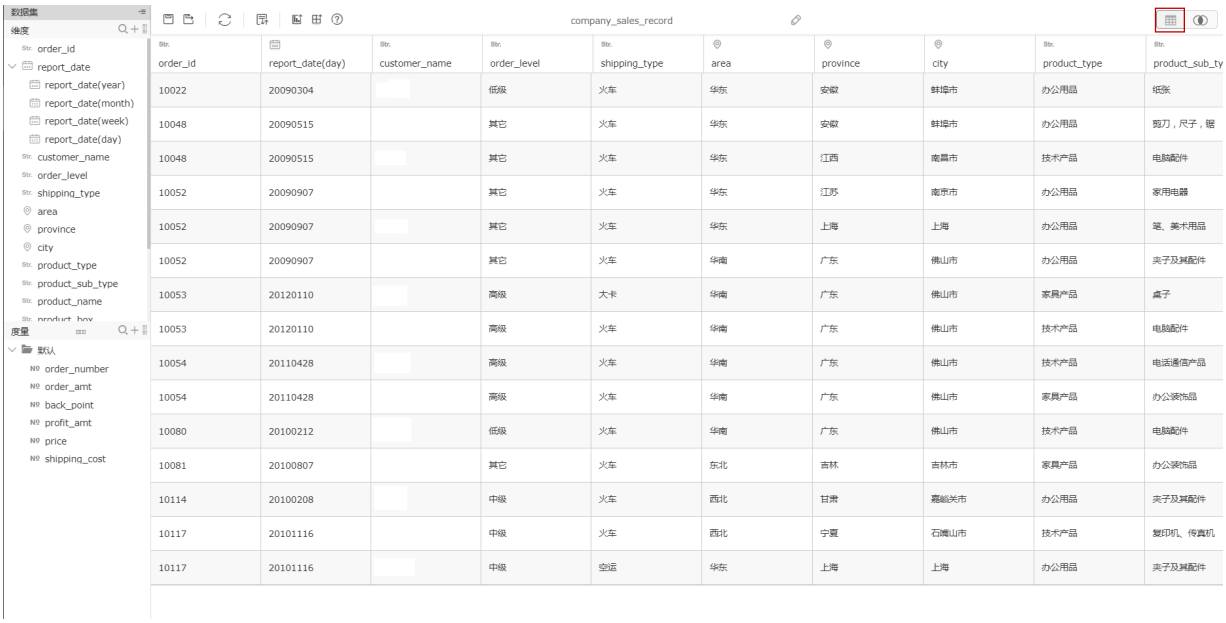
  当您在数据集编辑界面进行新建字段（度量）、删除字段、维度度量转化等操作后，需要保存数据集，进行刷新数据查看效果。
- 另存为：把当前的数据集另存为新的数据集，此操作可以用于快速复制一个新的数据集或备份数据集等场景。

9.3.2.3.4 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 *9-40: 数据预览*所示。

图 9-40: 数据预览



9.3.2.3.5 关联工作表

操作步骤

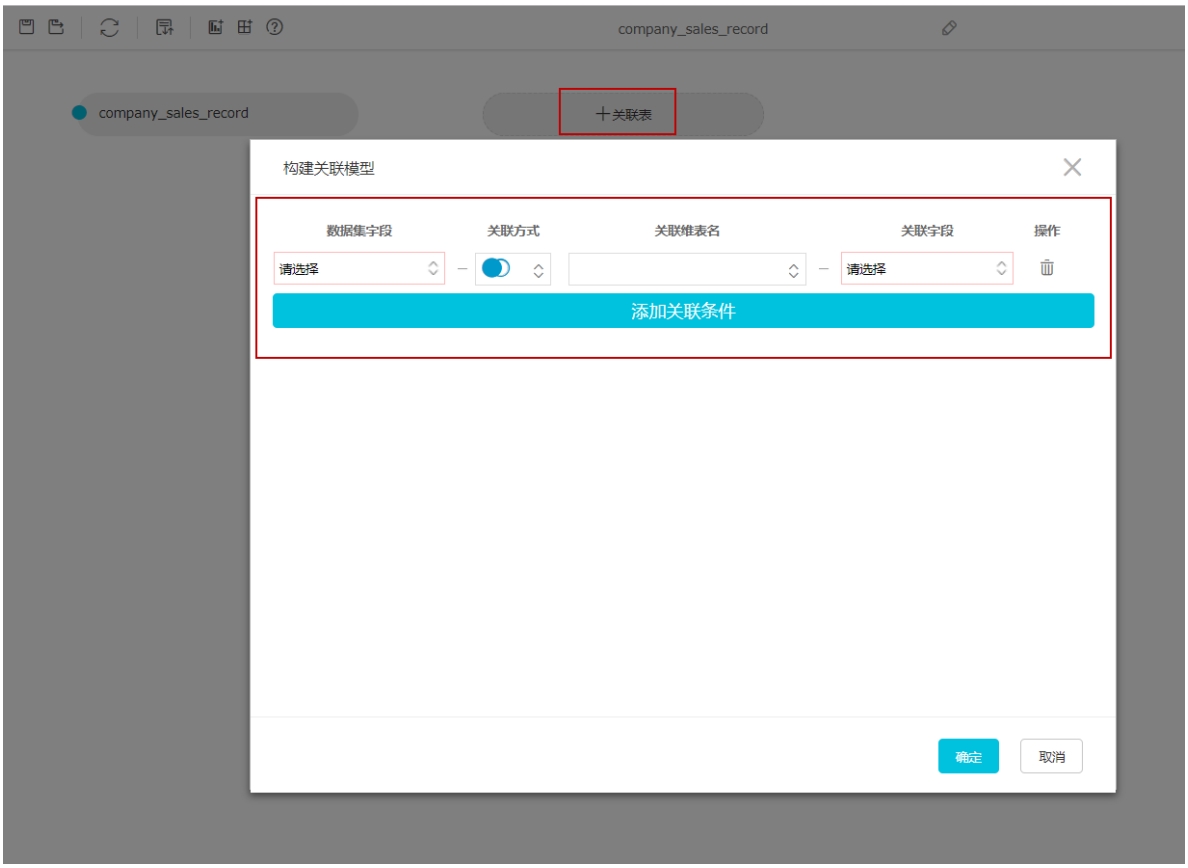
1. 单击**关联**图标，可切换到关联模式，如图 *9-41: 数据关联*所示。

   图 9-41: 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 *9-42: 构建关联模型*所示。

图 9-42: 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 *9-43: 添加关联条件*所示。

图 9-43: 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。

9.3.2.3.6 关联工作表示例

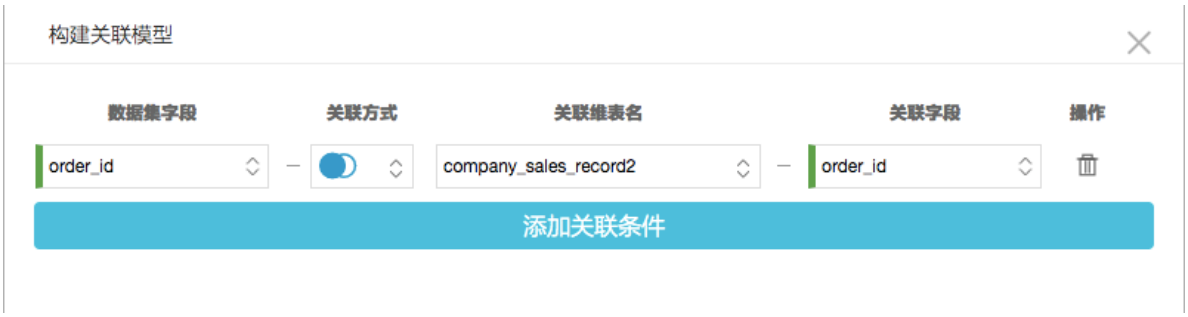
背景信息

把表company_sals_record1和表company_sals_record2以id进行**左外关联**。

操作步骤

1. 在数据集列表中，单击company_sals_record1，进入数据集编辑页面。
2. 单击**关联**图标，进入数据表关联页面。
3. 单击**+ 关联表**按钮，打开**构建关联模型**对话框。
4. 单击下拉箭头，选择需要关联的字段和关联方式，如图 *9-44:* 数据表左外关联所示。

图 9-44: 数据表左外关联



5. 单击**确定**按钮，完成关联模型构建。
6. 单击**预览**图标，进入到数据预览页面。
7. 单击**保存**图标，保存新关联的模型。
8. 单击**刷新**图标，查看关联后的数据，如图 *9-45:* 数据表左外关联预览所示。

图 9-45: 数据表左外关联预览

9.3.2.3.7 钻取

背景信息

以company_sales_record数据集和个人空间下的工作表为例，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

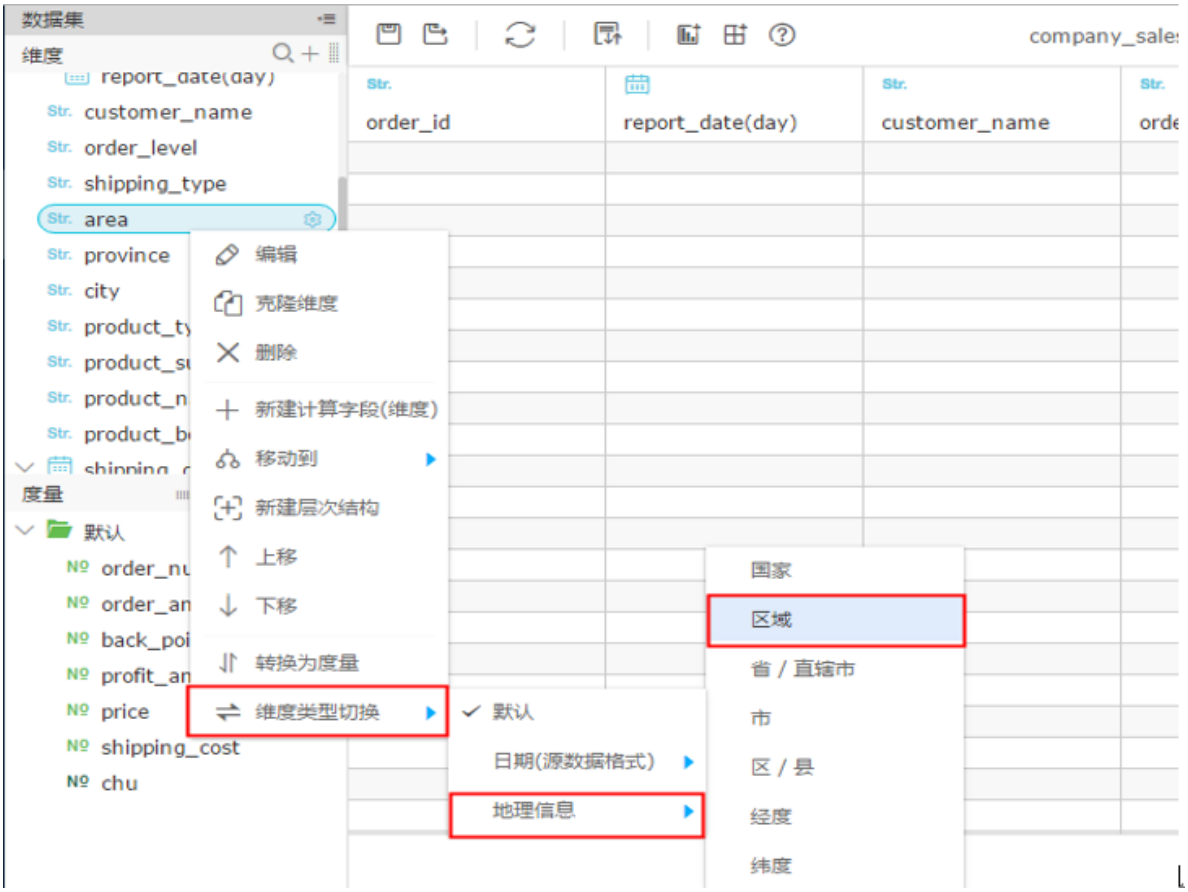
1. 在**数据集**列表中，单击company_sales_record，进入数据集编辑页面。

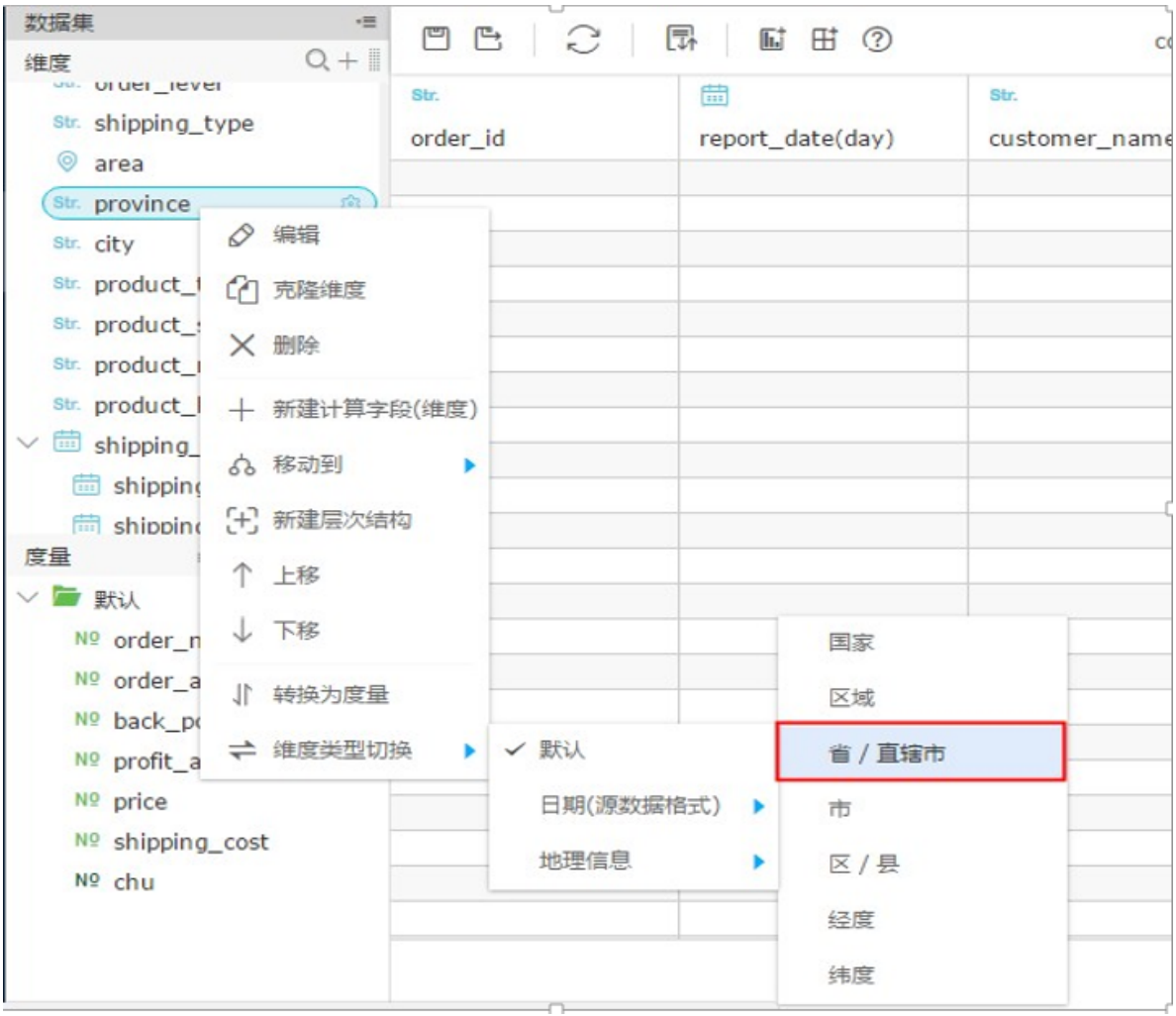
   切换字段类型

2. 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 *9-46: 切换字段类型*所示。

图 9-46: 切换字段类型

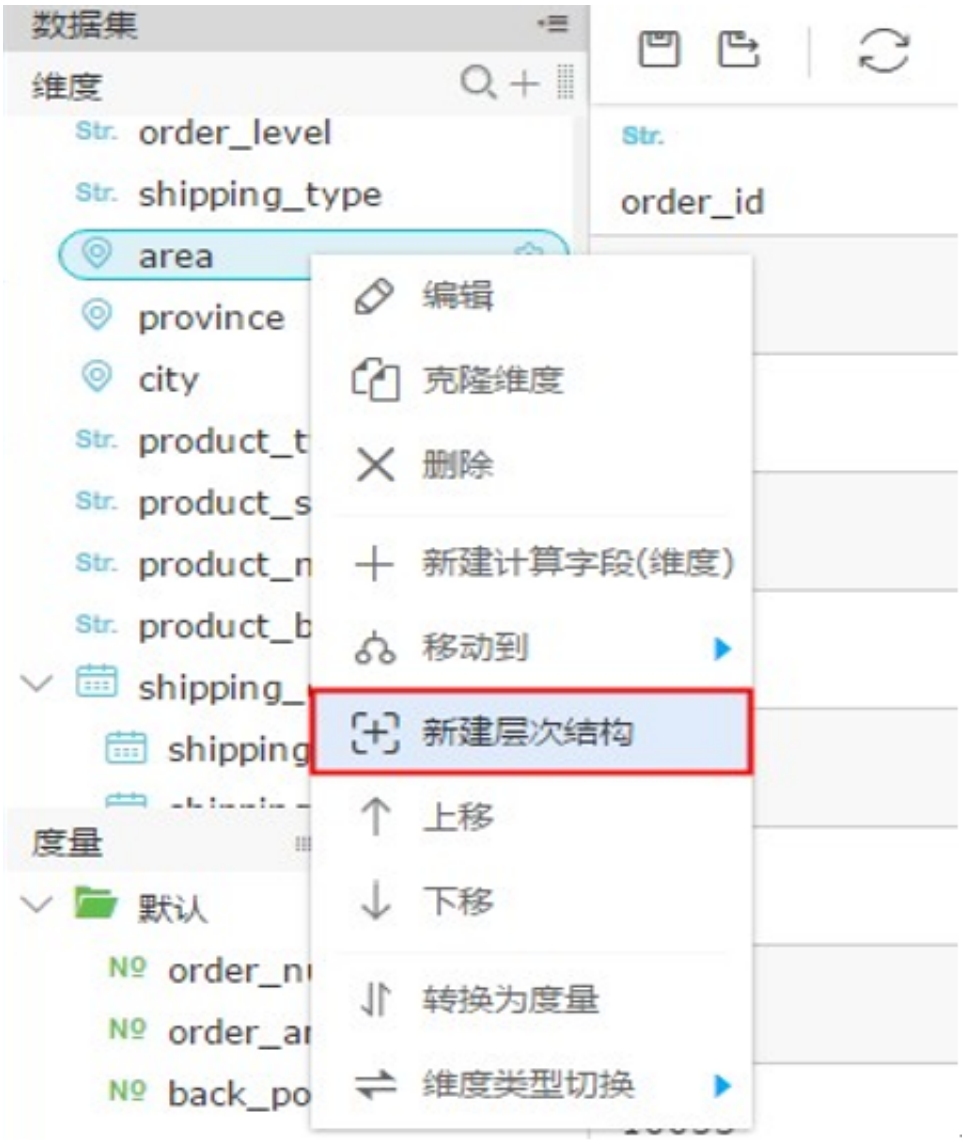




或创建具有地理信息的层次结构。

3. 在维度列表中，选择一个字段。
4. 单击鼠标右键，打开编辑菜单。
5. 选择**新建层次结构**，如图 *9-47: 新建层次结构*所示。

图 9-47: 新建层次结构

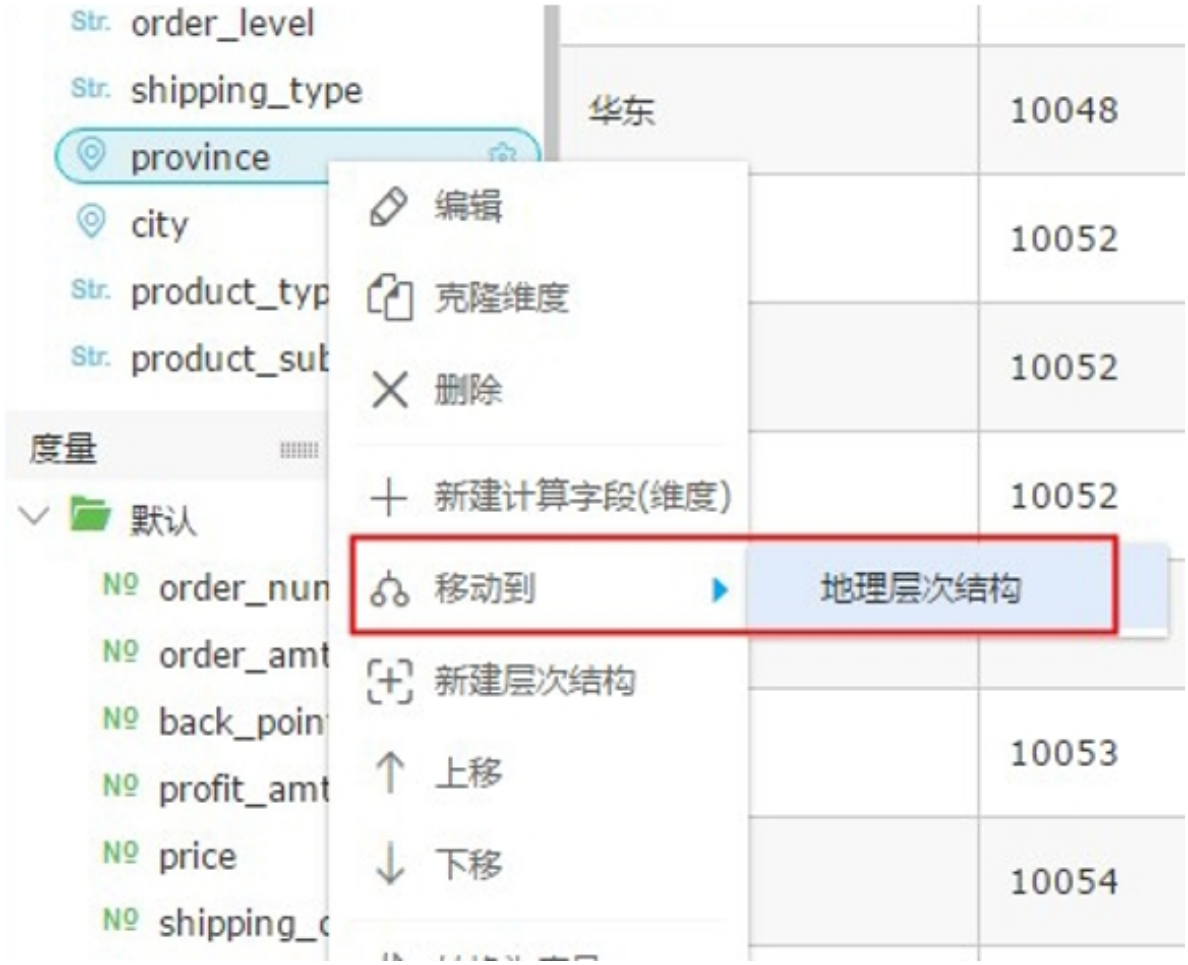


6. 输入层次结构名称，单击**确定**，如图 *9-48: 层次结构名称*所示。

图 9-48: 层次结构名称

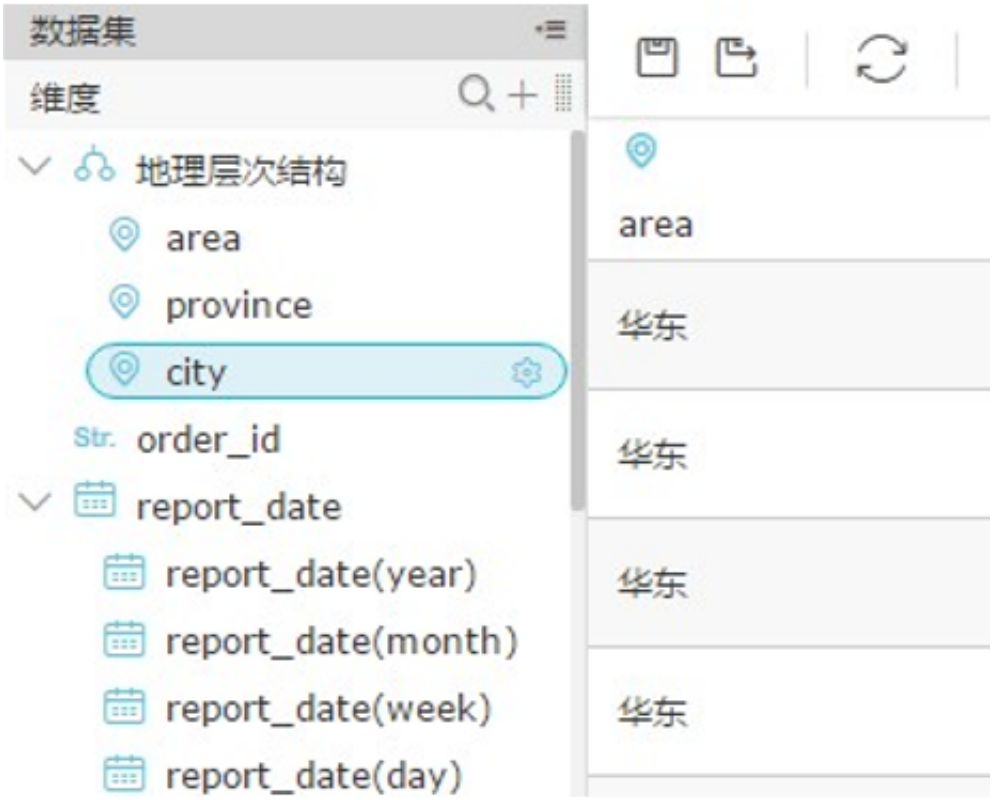
7. 将具有地理含义的维度字段移动到新建的层次结构中，如图 *9-49: 添加维度字段*所示。

图 9-49: 添加维度字段



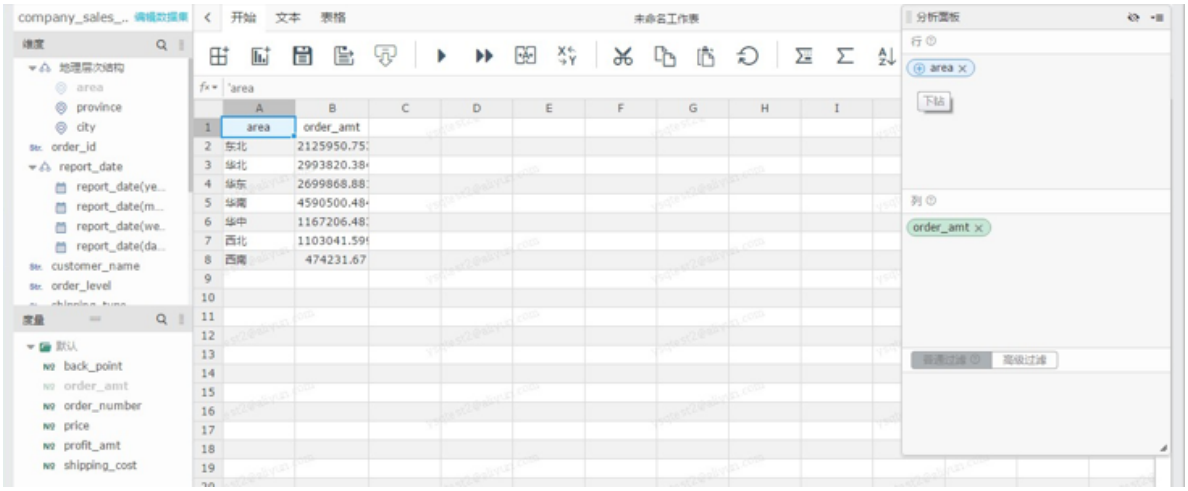
转换完成以后，维度列表字段如图 *9-50: 切换后的维度字段*所示。

图 9-50: 切换后的维度字段



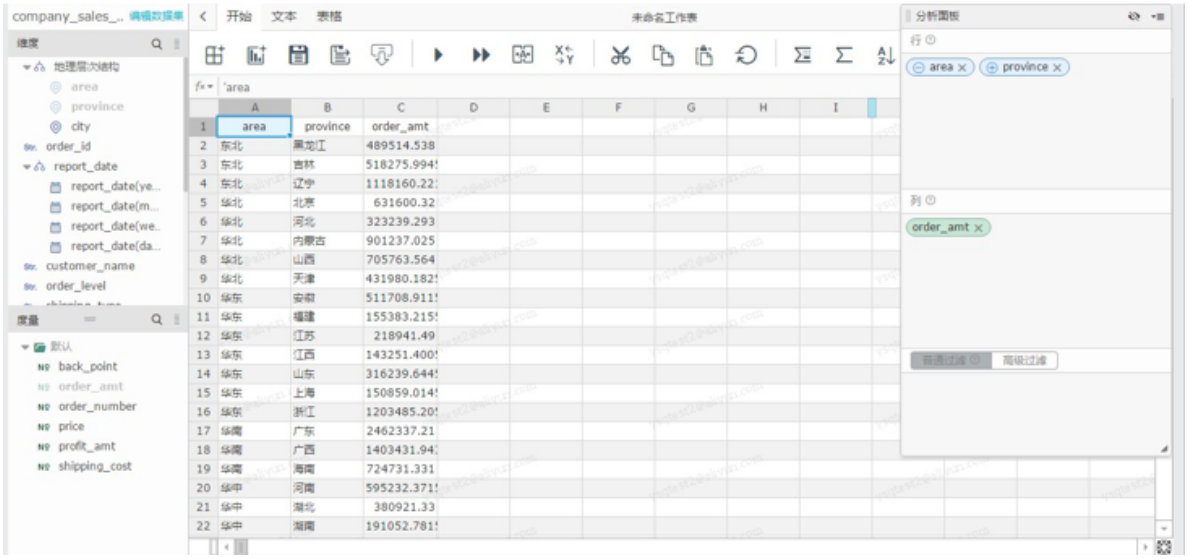
8. 单击**保存**，保存数据集。
9. 将area，order_amt字段依次添加到分析面板中，如图 *9-51: 在分析面板中添加字段*所示。

图 9-51: 在分析面板中添加字段



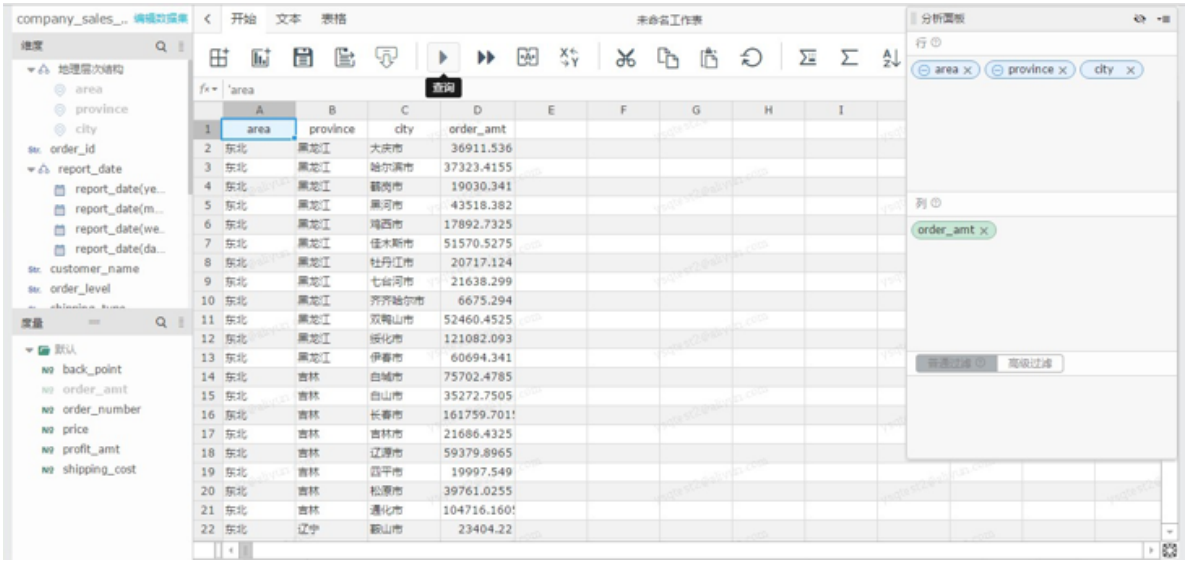
鼠标指向area时，左侧会出现加号，单击加号即可下钻到各省份。

图 9-51: 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 *9-52: 省份字段下钻*所示。

图 9-52: 省份字段下钻

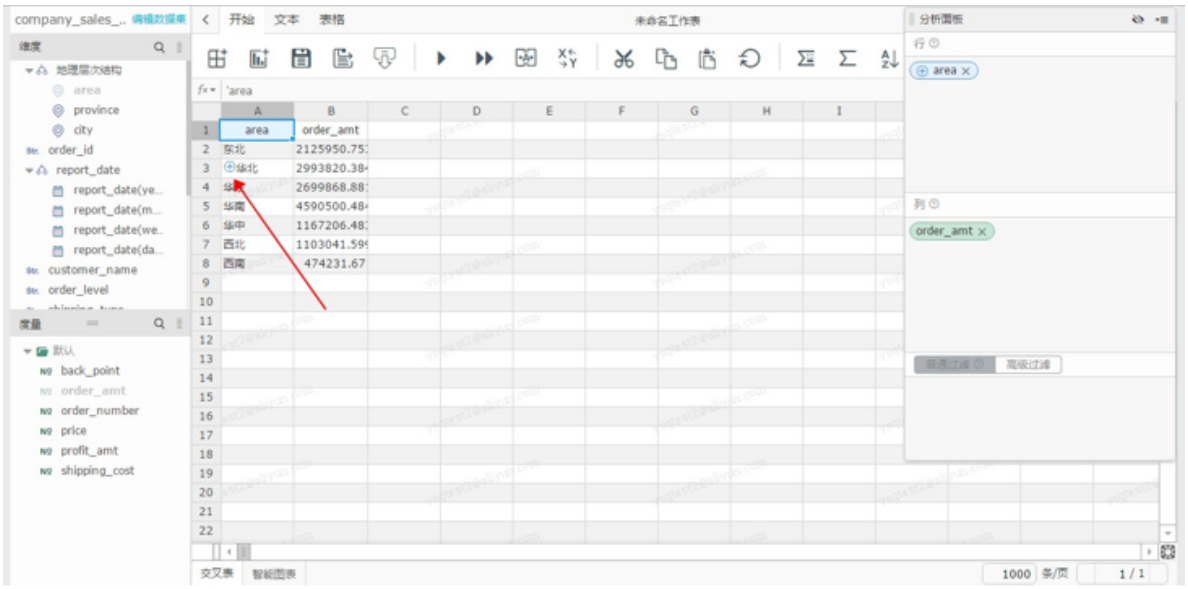


province和area左侧都由加号变为了减号，指向会出现提示上钻信息，即从城市上钻到省/直辖市，或者由省/直辖市上钻到区域。

实现局部概念下的钻取。

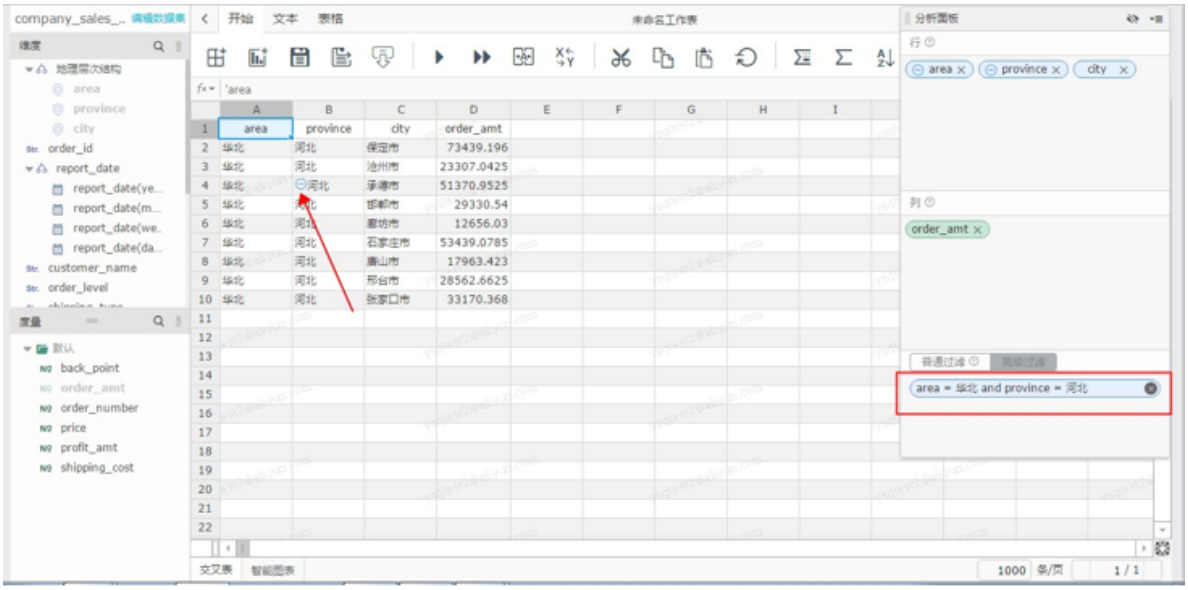
如下图红箭头指向华北区域时，左侧会出现加号，说明可以下钻到华北并下钻到各省，效果相当于添加了过滤条件：area = 华北。

图 9-53: 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 9-54: 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。

10.单击**保存**，保存编辑好的工作表。

9.3.2.3.8 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。

9.3.2.3.8.1 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。

9.3.2.3.8.2 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。
- 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

9.3.2.3.8.3 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。

9.3.2.3.8.4 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**数据集**，进入数据集管理页面。
2. 单击company_sales_record，进入数据集编辑页面。
3. 在度量列表，单击加号，进入新建计算字段的对话框，如图 *9-56: 新建计算字段*所示。

图 9-56: 新建计算字段

4. 填写度量名称及表达式，如图 *9-57: 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。

> **说明**：
> 填写表达式时，必需使用英文输入法。

图 9-57: 新建计算字段表达式

5. 选择一个数据类型。

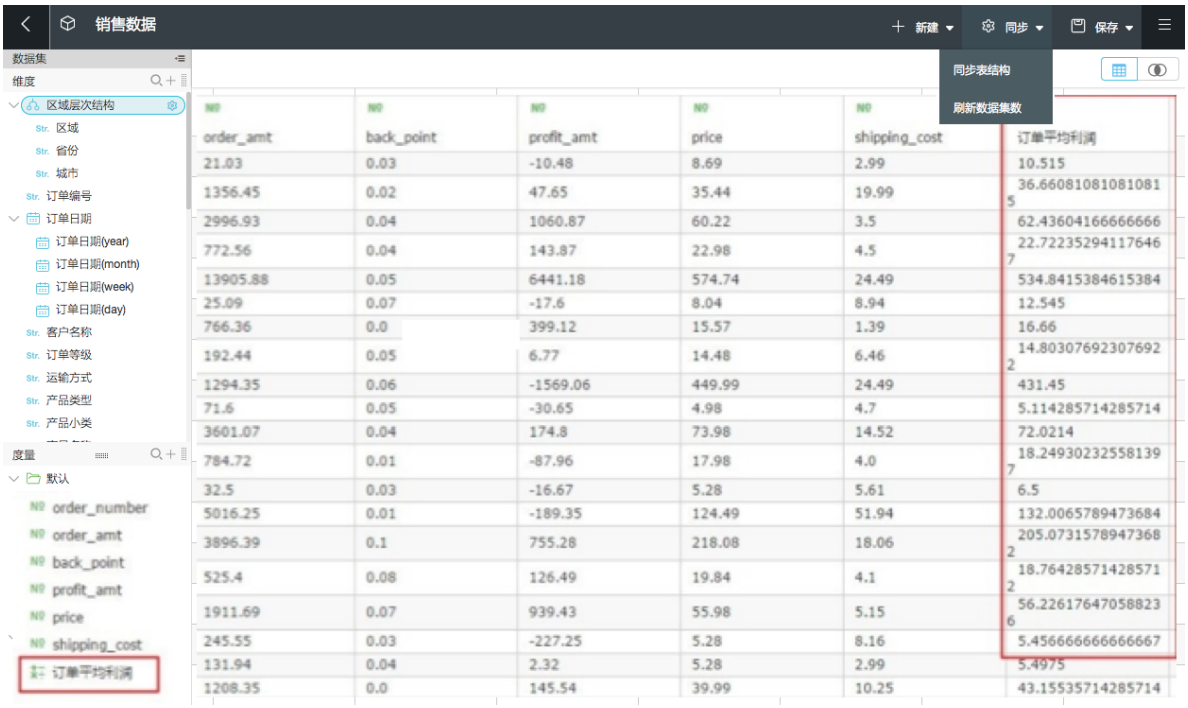
   例如，平均值为一个数值，那么在数据类型中，请选择**数值**选项。

6. 单击**确定**，完成字段新建。

7. 单击**保存**，保存数据集。

8. 单击**刷新数据集**，查看新建的计算字段，如图 *9-58: 刷新数据集*所示。

图 9-58: 刷新数据集



9.3.2.4 删除数据集

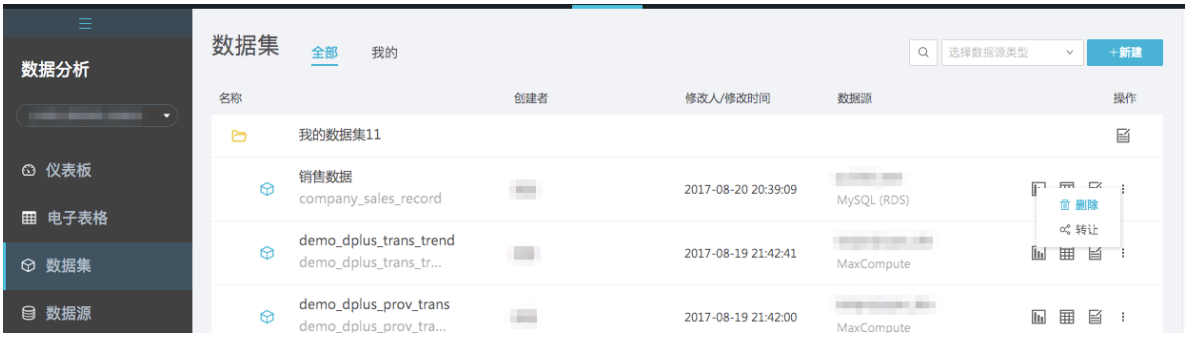
背景信息

如果有基于该数据集的电子表格和仪表板，删除该数据集后，会导致关联的电子表格和仪表板失效。

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 *9-59: 删除数据集*所示。

图 9-59: 删除数据集

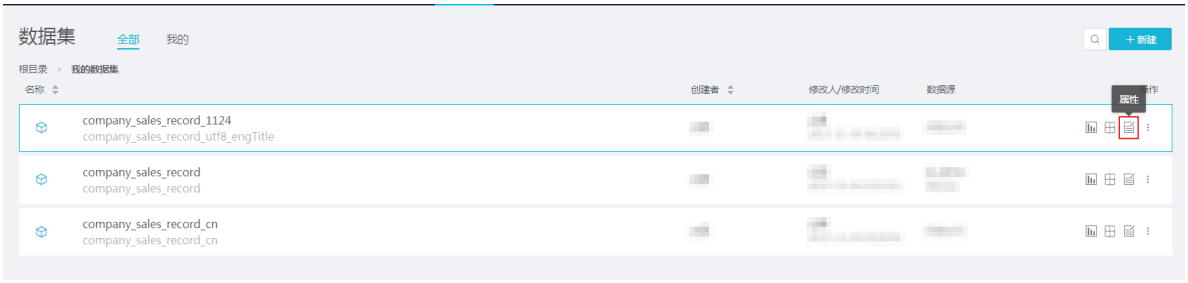


9.3.2.5 重命名数据集

操作步骤

1. 在**数据集**列表页面，选择一个数据集。
2. 单击**属性**，可重命名当前数据集，如图 *9-60: 重命名数据集*所示。

图 9-60: 重命名数据集

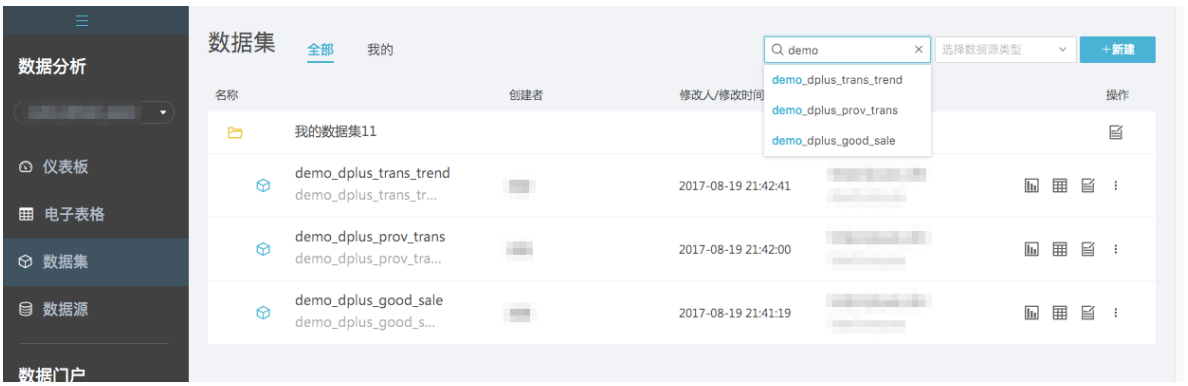


9.3.2.6 查询数据集

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 *9-61: 查询数据集*所示。

图 9-61: 查询数据集



9.3.2.7 新建数据集文件夹

背景信息

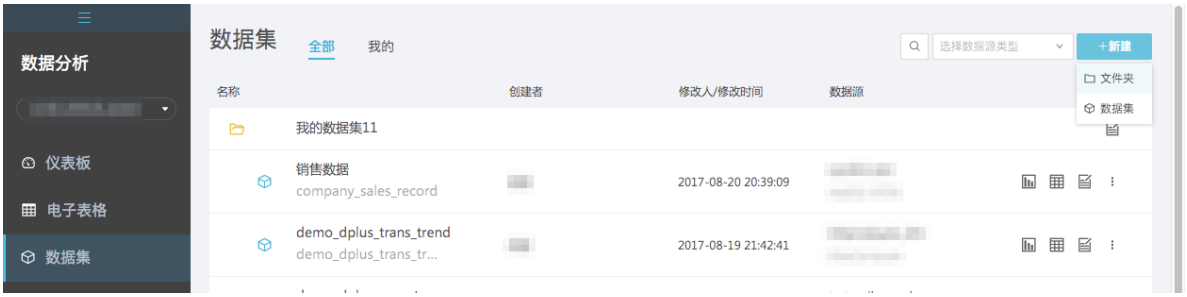
您可以在**数据集**列表页创建多个文件夹以便更好的管理数据集。

操作步骤

1. 在**数据集**列表页面，选择**我的**标签页。

2. 选择**新建** > **文件夹**，如图 *9-62: 新建数据集文件夹*所示。

图 9-62: 新建数据集文件夹

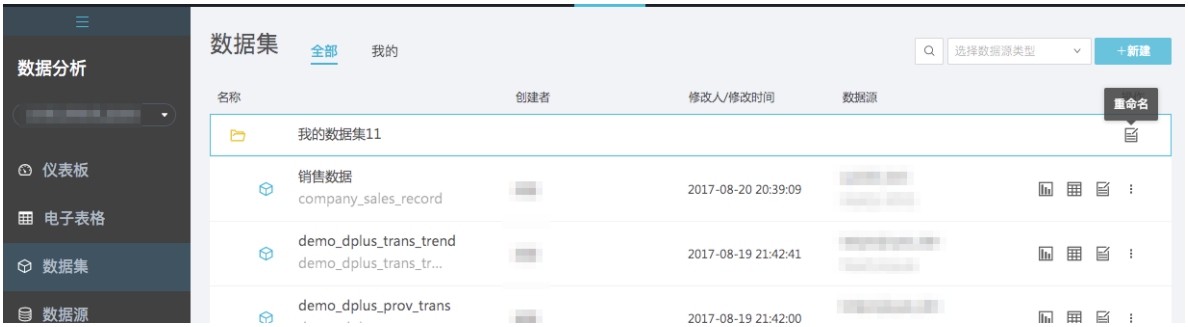


9.3.2.8 重命名数据集文件夹

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 *9-63: 数据集文件夹重命名*所示。

图 9-63: 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。

9.3.2.9 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。

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 *9-64: 行级权限控制*所示。

图 9-64: 行级权限控制

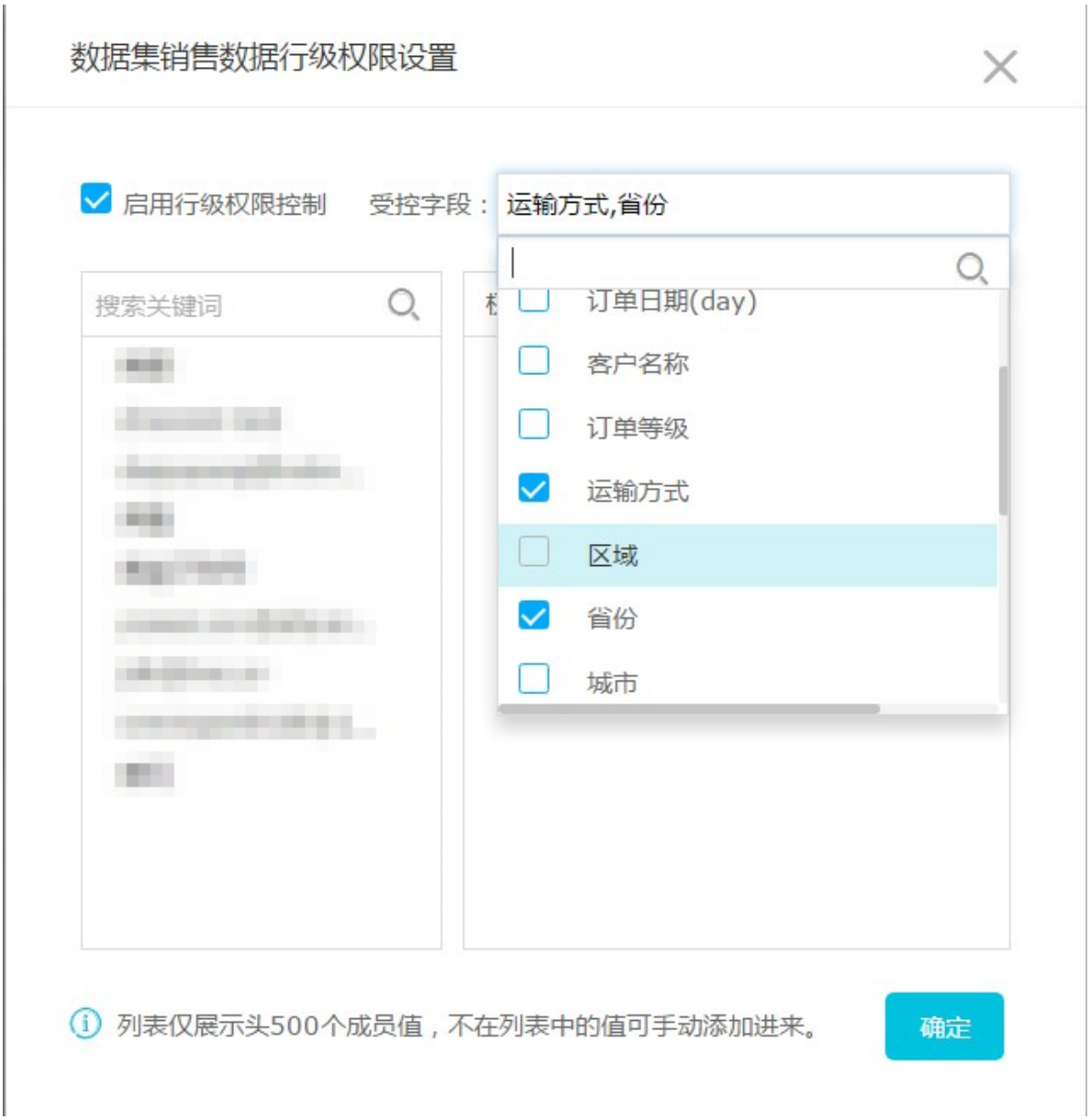
4. 选择数据集上需要受控的字段，如图 *9-65:* 选择受控字段所示。

   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 9-65: 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 9-66: 为用户1设置行级权限

- 为用户2设置行级权限。

图 9-67: 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 *9-68: 控制效果*所示。

图 9-68: 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 *9-68: 无权限提示*所示。

图 9-68: 无权限提示

9.4 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的数据查询。

9.4.1 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。

9.4.1.1 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，仪表板通过引导、拖拽和单击字段，让数据展示得更直观明了；从数据分析层面看，通过友好的页面提示，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。

9.4.1.2 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 9-70: 图表样式和控件

新增

- 新增了对字段的过滤筛选。

图 9-71: 过滤器

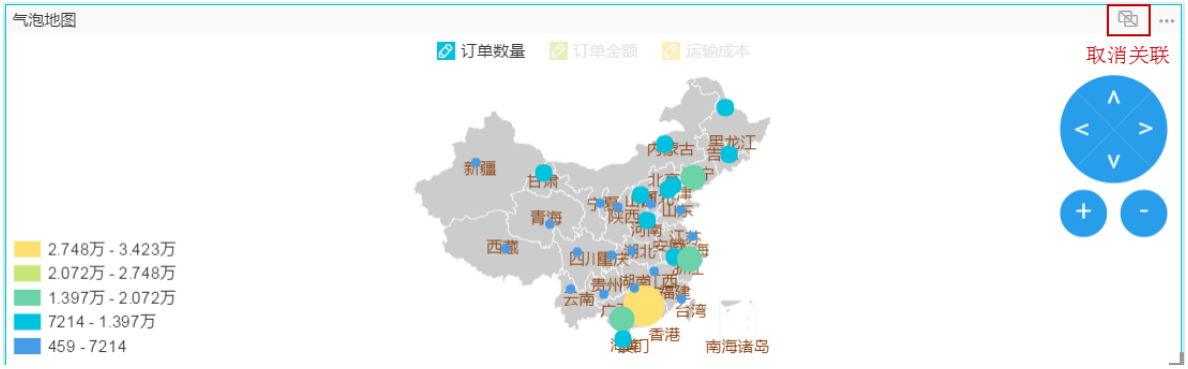
- 查询控件可支持非同源关联。

图 9-72: 非同源关联

- 图表的高级联动功能新增了取消关联。

> **说明**：
> 取消关联的功能只是将被关联的图表恢复到最初的样式，两张图表之间的关联依旧存在。

图 9-73: 取消关联



9.4.1.3 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 9-1: 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	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
	标进展以及可能具有的模式。		
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

9.4.1.4 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 *9-74:* 图表标签页所示。

图 9-74: 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 9-2: 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
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

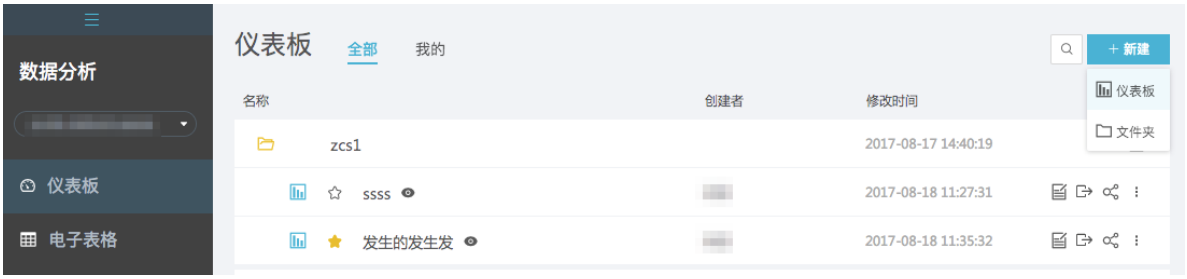
图表名称	数据图表要素	数据要素构成
		于等于12；扇区长度上有且仅有1个度量
词云图	词大小，词标签	词大小有且仅有1个维度；词标签有且仅有1个度量
旋风漏斗	对比主题，对比指标	对比主题有且仅有1个维度；对比指标最少1个度量
树图	树父子节点标签，树父子节点指标	树父子节点标签最少2个维度，树父子节点指标最少1个度量
来源去向图	前一页面，当前页面，后一页面；前一页面PV，前一页面UV，当前页面PV，当前页面UV，下一页面PV，下一页面UV，路径转化率，页面跳出率	所有数据要素，分别只取1个维度或者1个度量

9.4.2 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 *9-75:* 新建仪表板所示。

图 9-75: 新建仪表板



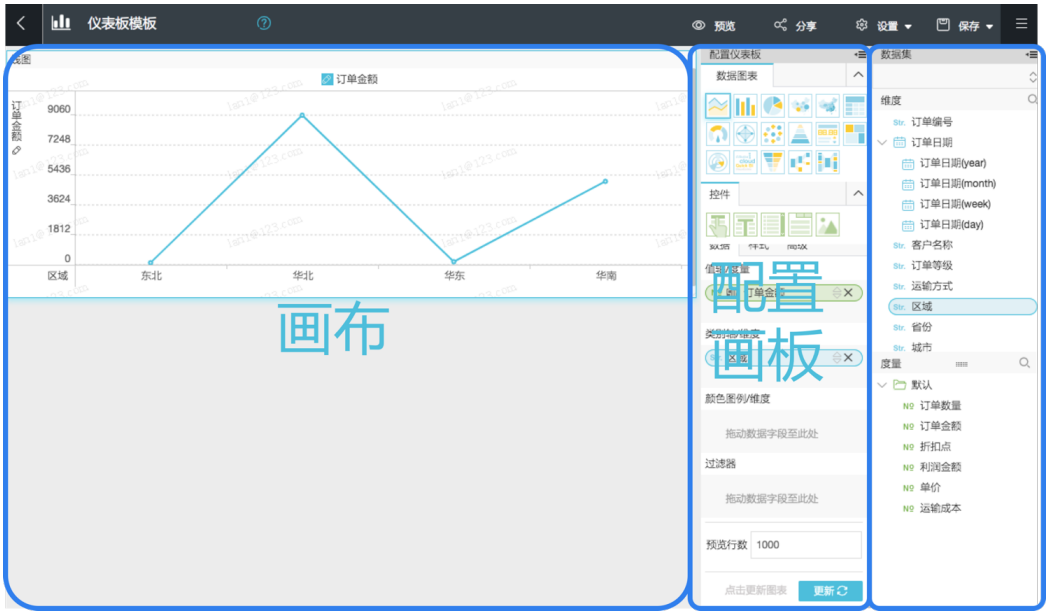
9.4.3 仪表板分区

仪表板编辑页面可分为以下三个区域，如图 *9-76:* 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 9-76: 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。

9.4.3.1 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。

9.4.3.1.1 切换数据集

背景信息

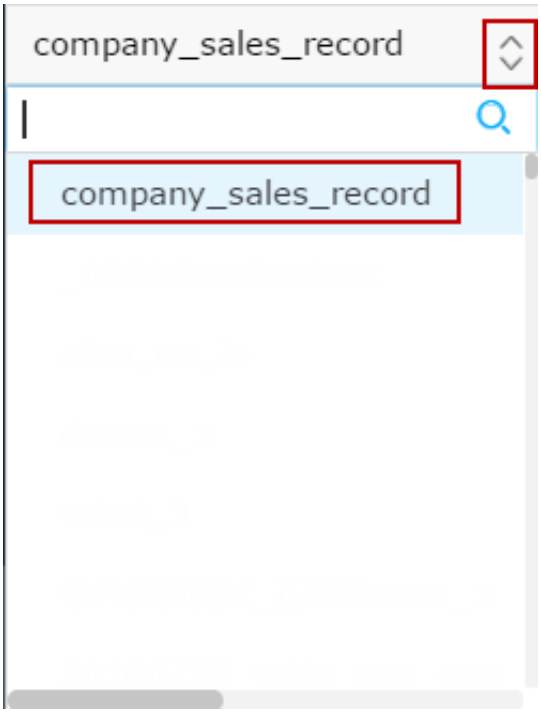
如果您在下拉菜单中找不到需要的数据集，可以单击左侧导航栏中的**数据**图标，返回到数据集列表页面，检查数据集是否创建成功。

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 *9-77: 切换数据集*所示。

图 9-77: 切换数据集

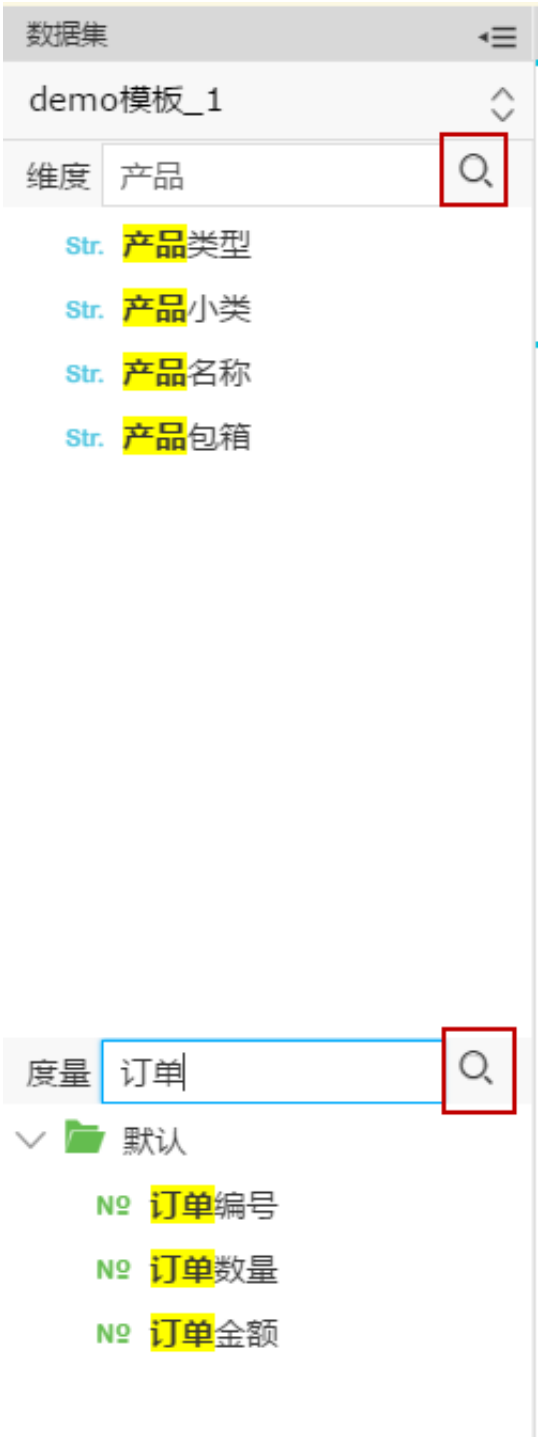


9.4.3.1.2 搜索维度字段和度量字段

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 *9-78: 搜索维度字段和度量字段*所示。

图 9-78: 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。

9.4.3.2 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。

9.4.3.2.1 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 *9-79:* 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 9-79: 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。

9.4.3.2.2 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 *9-80: 颜色图例*所示。

图 9-80: 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 *9-81:* 颜色图例效果所示。

图 9-81: 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 *9-82:* 图例配色方案所示。

图 9-82: 图例配色方案

9.4.3.2.3 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

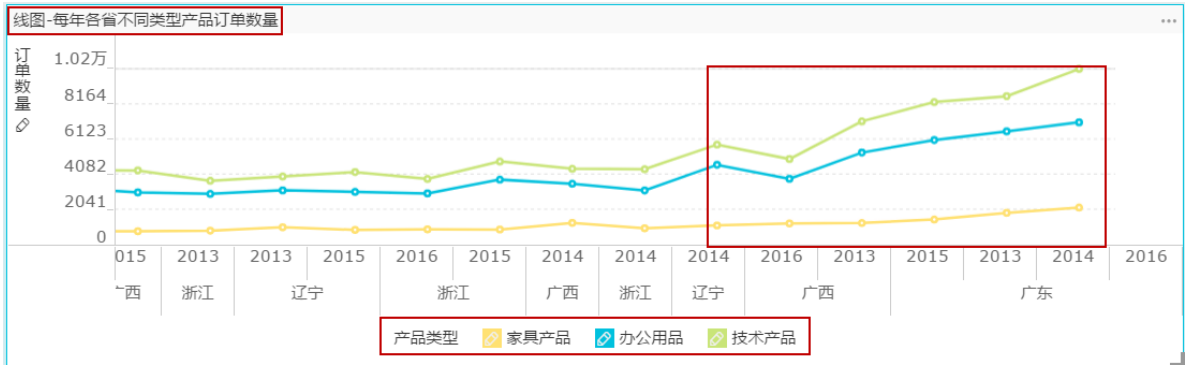
1. 选中一个字段，例如订单数量。
2. 单击字段后面向上的小三角图标，如图 *9-83:* 设置排序所示。

   向上的三角图标为升序，向下的三角图标为降序。

图 9-83: 设置排序

3. 选择完成后，单击**更新**，更新后的图表如图 *9-84:* 排序效果所示。

图 9-84: 排序效果



9.4.3.2.4 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。

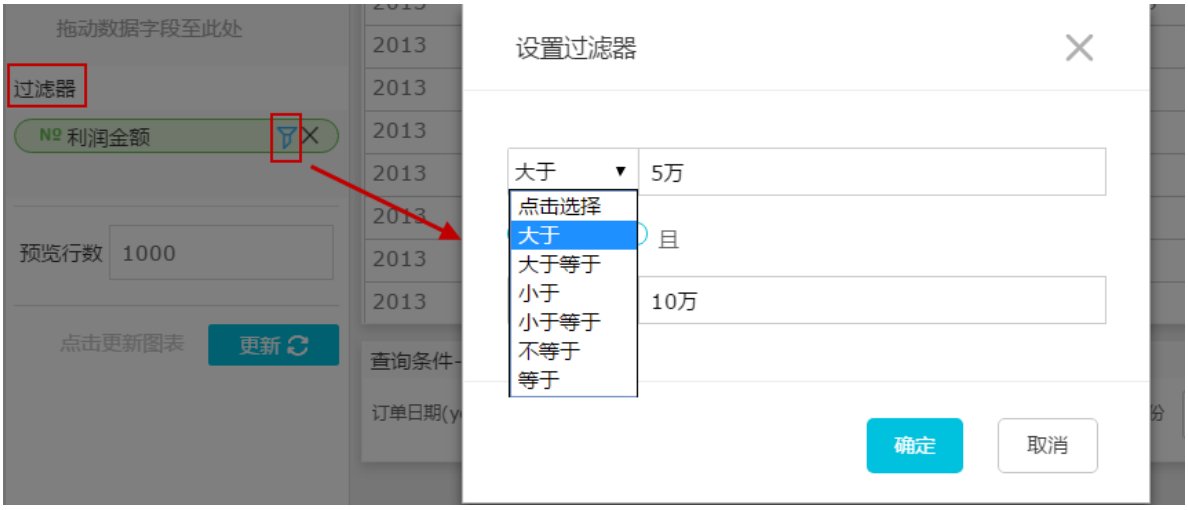
以**利润金额**为例。

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 *9-85: 设置过滤器*所示。

图 9-85: 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 *9-86: 设置过滤范围*所示。

图 9-86: 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。

9.4.3.2.5 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 *9-87: 高级标签页*所示。

图 9-87: 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 *9-88: 关联设置*所示。

图 9-88: 关联设置

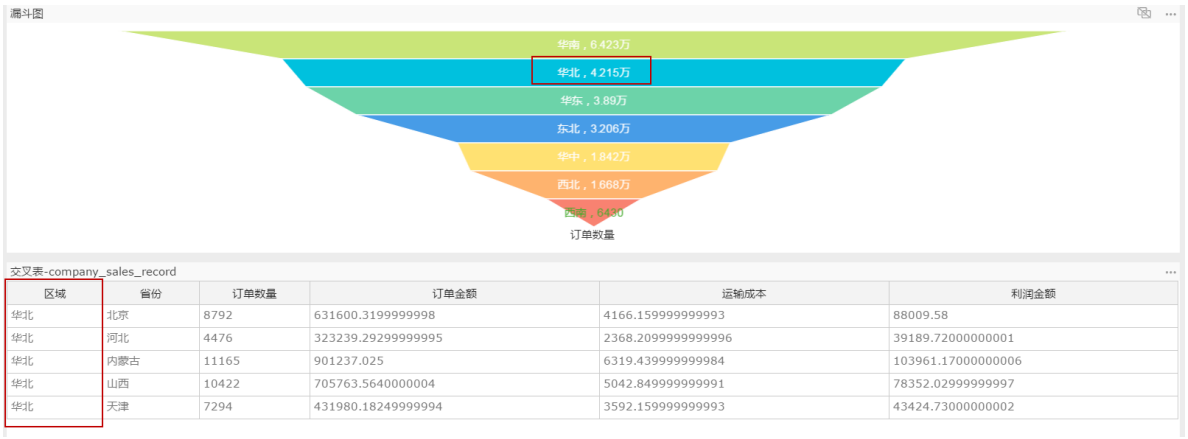
如果选择错误，系统会自动给出错误提示。

5. 在仪表板展示区上方，单击**预览**图标，进入预览页面。

图 9-89: 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 *9-90: 关联效果*所示。

图 9-90: 关联效果



7. 在图表的右上角，单击**取消关联**图标，被关联的交叉表会恢复到最初的状态。

9.4.3.3 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

9.4.3.3.1 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 *9-91:* 仪表板工具栏所示。

图 9-91: 仪表板工具栏

9.4.3.3.2 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：

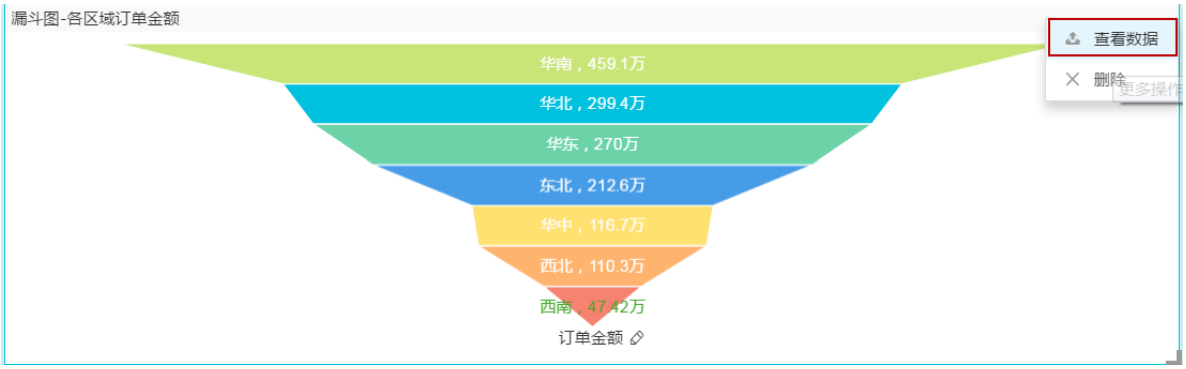
图表可拖拽的区域仅限于仪表板展示区（画布）内。

9.4.3.3.3 查看图表数据

操作步骤

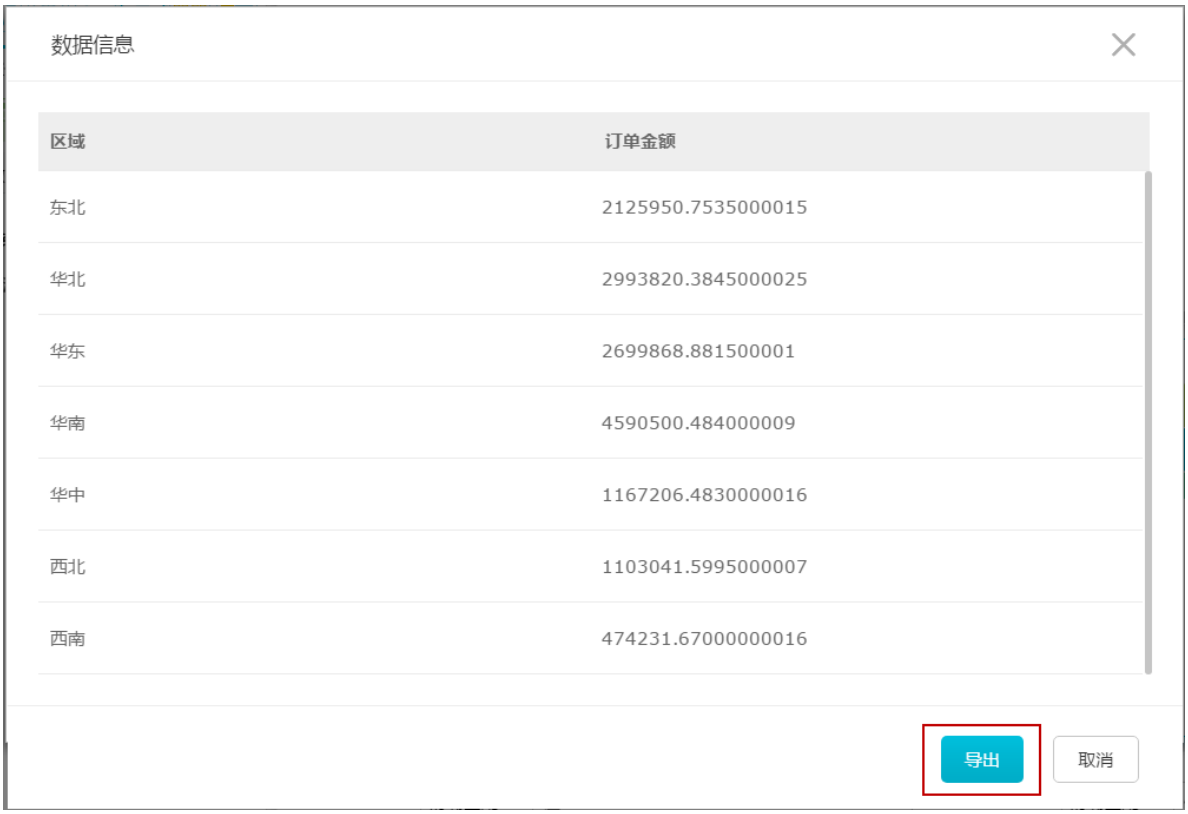
1. 选中一个图表，例如漏斗图。
2. 鼠标指向图表右上方。
3. 单击**查看数据**，如图 *9-92: 查看图表数据*所示。

图 9-92: 查看图表数据



4. 单击**导出**，下载数据到本地，如图 *9-93: 导出图表数据*所示。

图 9-93: 导出图表数据

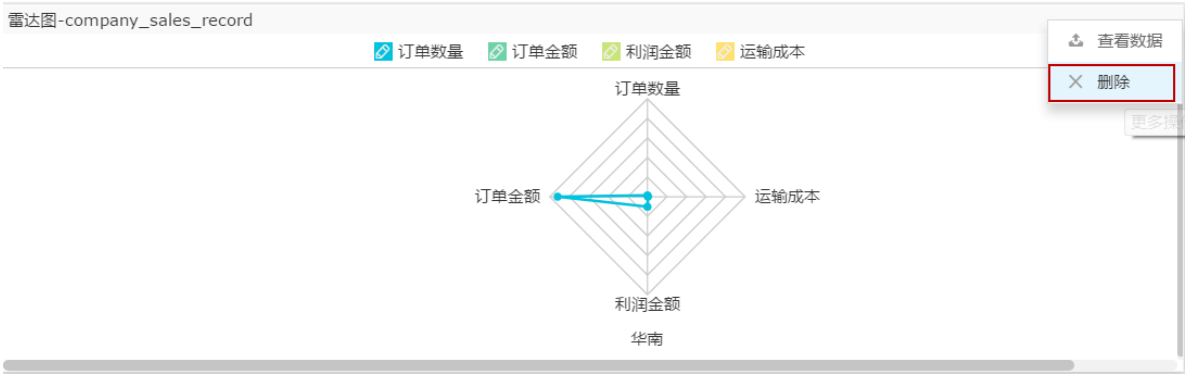


9.4.3.3.4 删除图表

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 *9-94: 删除图表*所示。

图 9-94: 删除图表



9.4.3.3.5 切换图表

背景信息

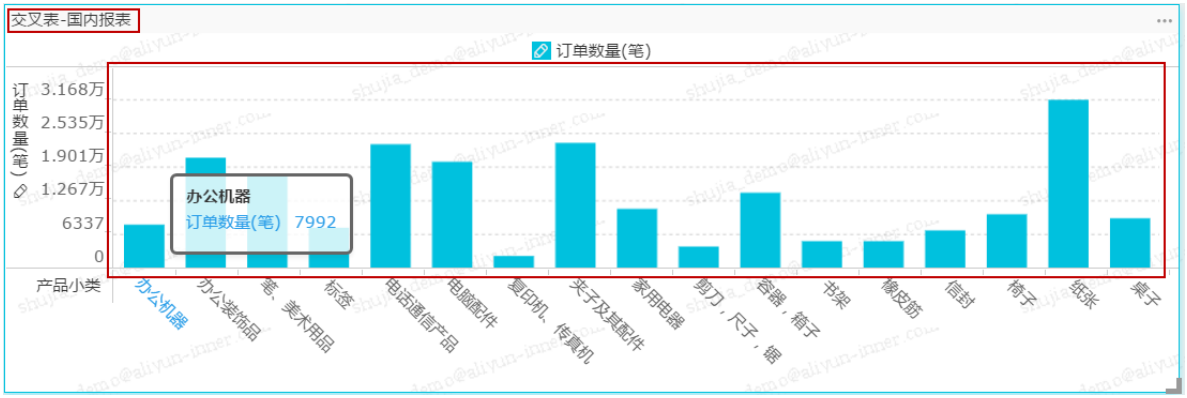
在仪表板展示区（画布），您可随意切换图表样式。

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 9-95: 切换图表所示。

图 9-95: 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 9-95: 系统提示所示。

图 9-95: 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。

9.4.3.3.6 引导功能

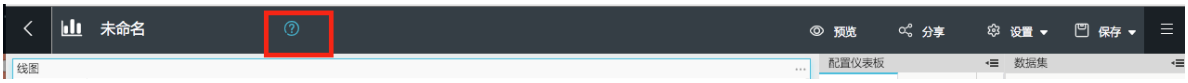
背景信息

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 *9-97: 引导图标*所示。

图 9-97: 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 *9-98: 引导功能*所示。

图 9-98: 引导功能

9.4.3.3.7 控件

仪表板展示区域支持以下五种控件。

- 查询条件
- 文本框
- IFRAME
- TAB
- PIC

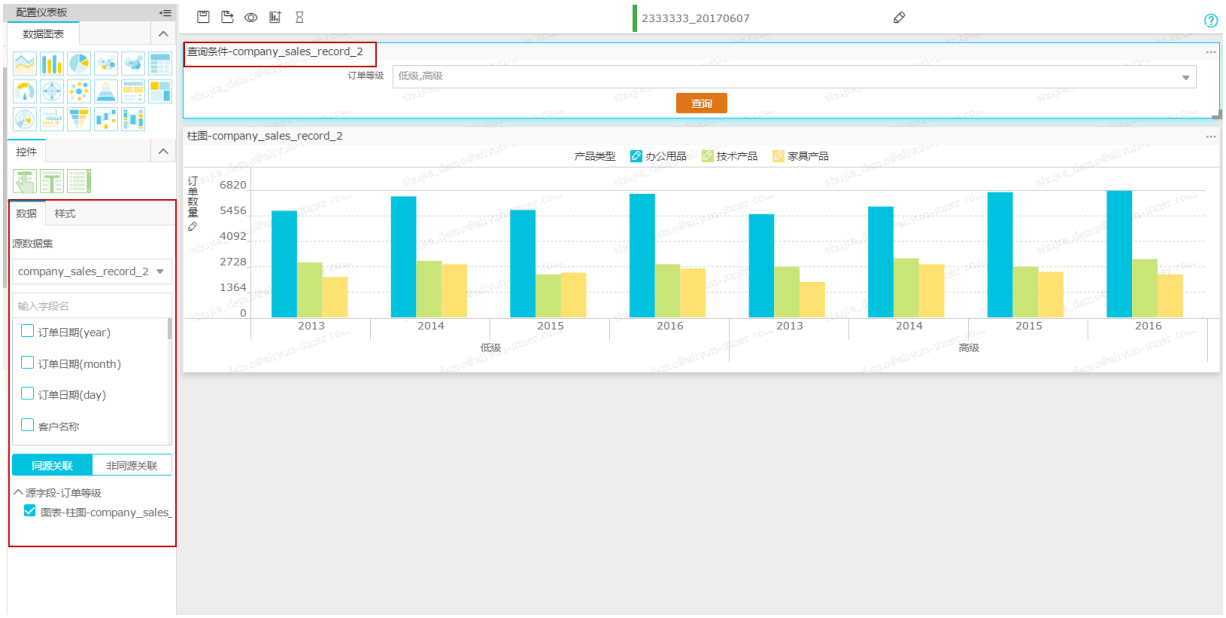
9.4.3.3.7.1 查询条件

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

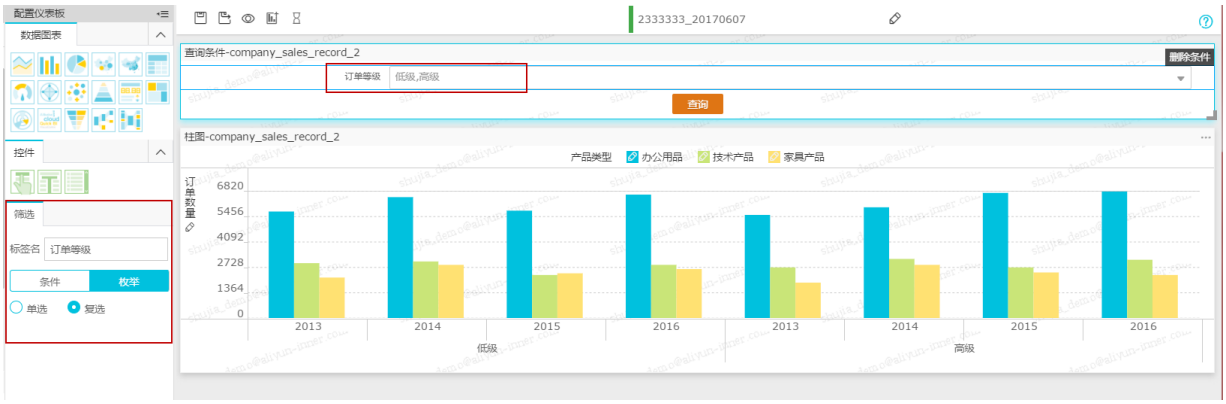
单击查询条件控件，可打开查询条件控件编辑菜单，如图 *9-99:* 查询条件所示。

图 9-99: 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 *9-100:* 过滤条件编辑菜单所示。

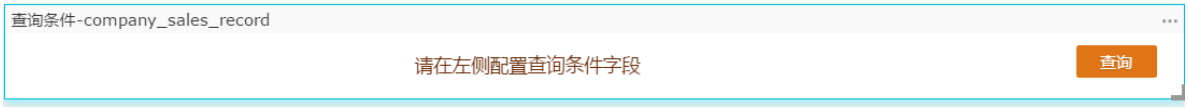
图 9-100: 过滤条件编辑菜单



操作步骤

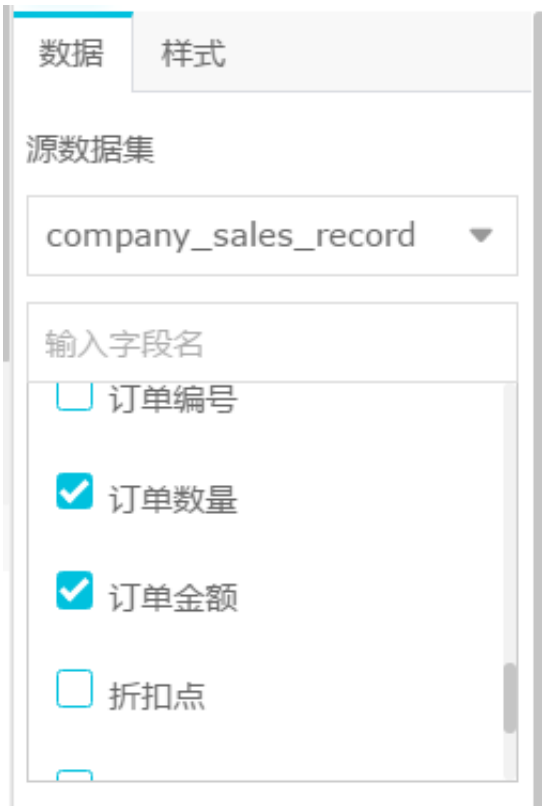
1. 双击**查询条件**图标，如图 *9-101: 查询条件图例*所示。

   图 9-101: 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 *9-102: 查询条件设置*所示。

图 9-102: 查询条件设置



目前，控件支持同源关联和非同源关联。

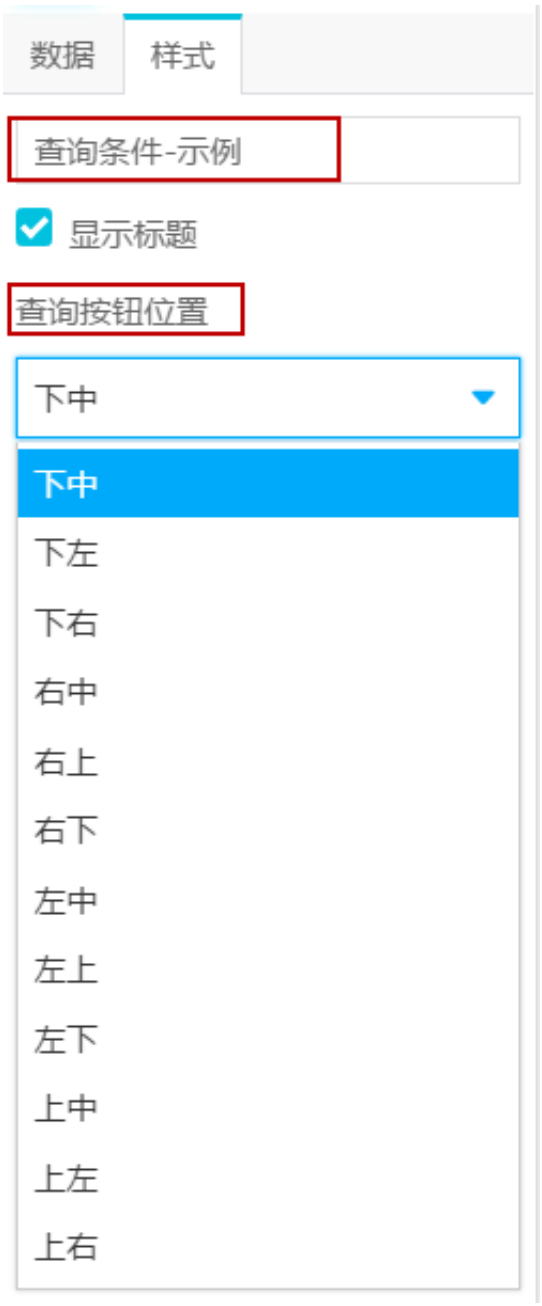
同源关联示例

3. 选择同源关联，按字段类型选择图表，如图 *9-103:* 同源关联所示。

图 9-103: 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 *9-104:* 查询条件编辑所示。

图 9-104: 查询条件编辑



编辑完成后，控件如图 *9-105:* 查询条件展示所示。

图 9-105: 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 *9-106:* 查询条件设置所示。

图 9-106: 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 *9-107:* 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 9-107: 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 *9-108: 海外报表*所示。

图 9-108: 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 *9-109: 国内报表*所示。

图 9-109: 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 *9-110: 非同源关联*所示。

图 9-110: 非同源关联

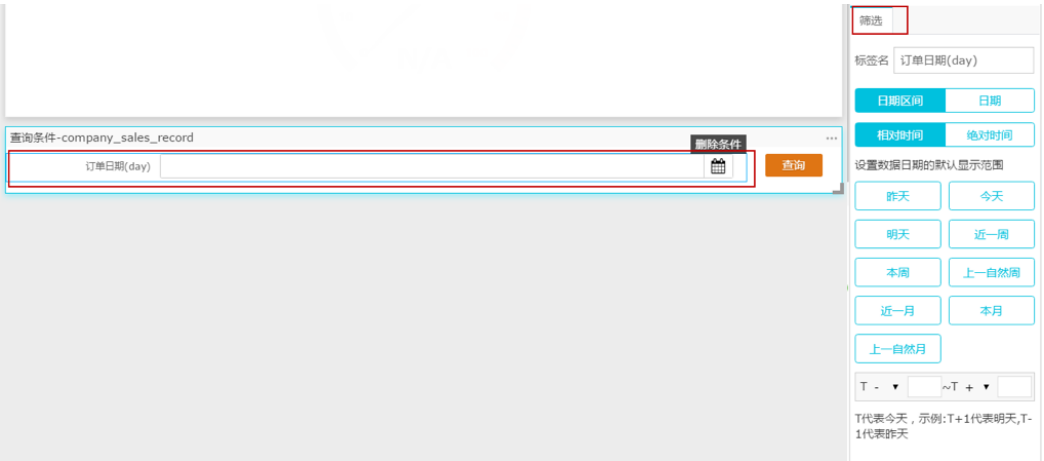
选择完成后，该控件就可以对两张来自不同数据集的图表进行查询了。

9.4.3.3.7.1.1 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 *9-111:* 日期查询所示。

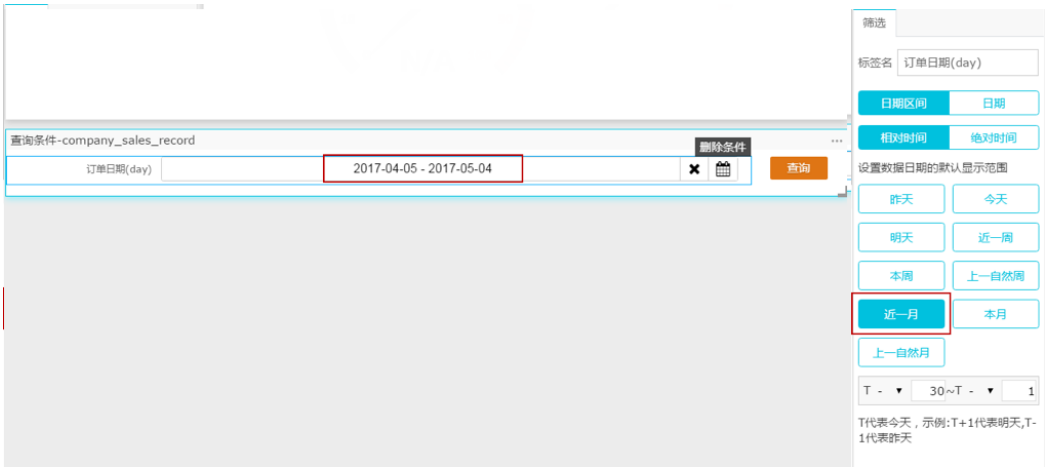
图 9-111: 日期查询



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 *9-112: 编辑日期范围*所示。

图 9-112: 编辑日期范围



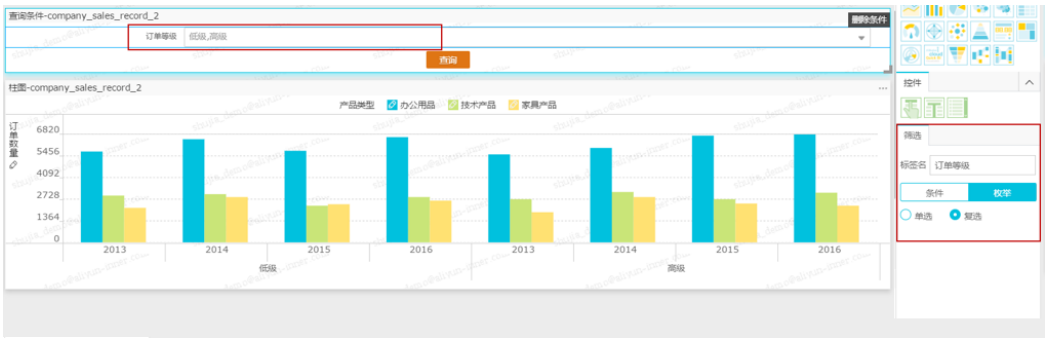
5. 单击**查询**。

9.4.3.3.7.1.2 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 *9-113: 文本查询*所示。

图 9-113: 文本查询

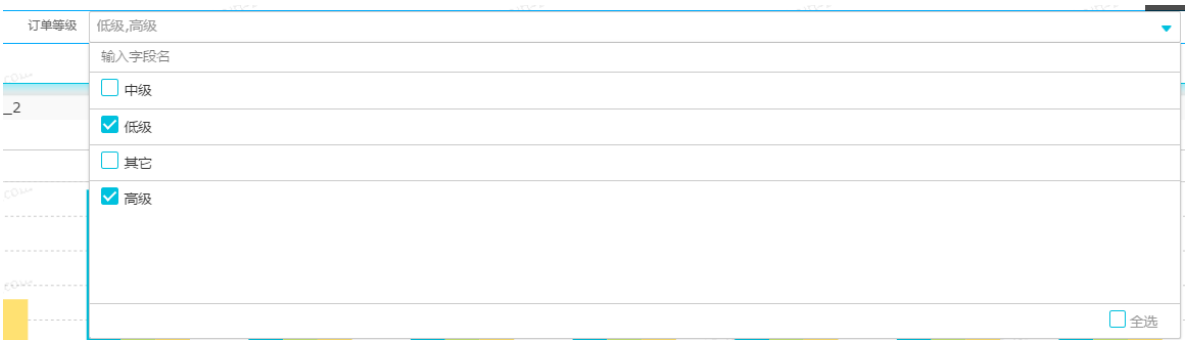


4. 选择一个文本查询条件，如枚举。

   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 *9-114: 枚举*所示。

图 9-114: 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 *9-115: 枚举查询*所示。

图 9-115: 枚举查询

9.4.3.3.7.2 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 *9-116: 文本框*所示。

图 9-116: 文本框

9.4.3.3.7.3 IFRAME

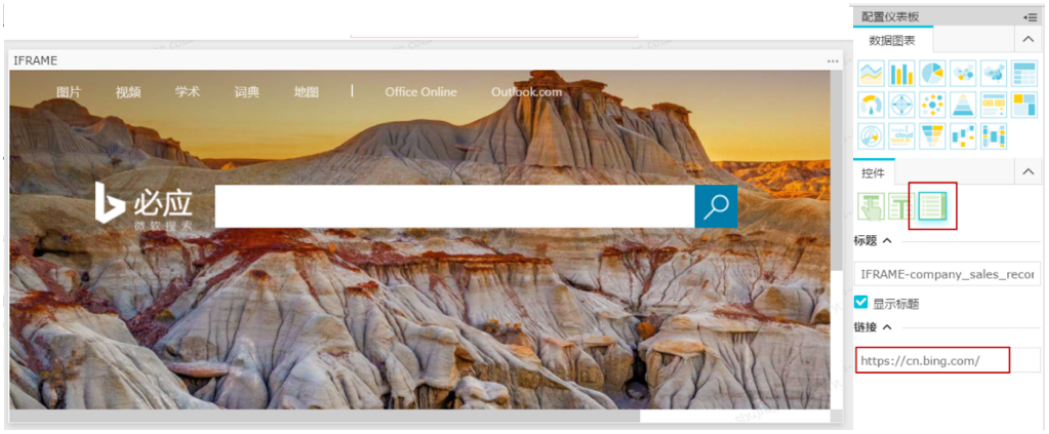
背景信息

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 *9-117: IFRAME*所示。

图 9-117: 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。

9.4.3.3.7.4 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 *9-118: TAB*编辑菜单所示。

图 9-118: TAB编辑菜单

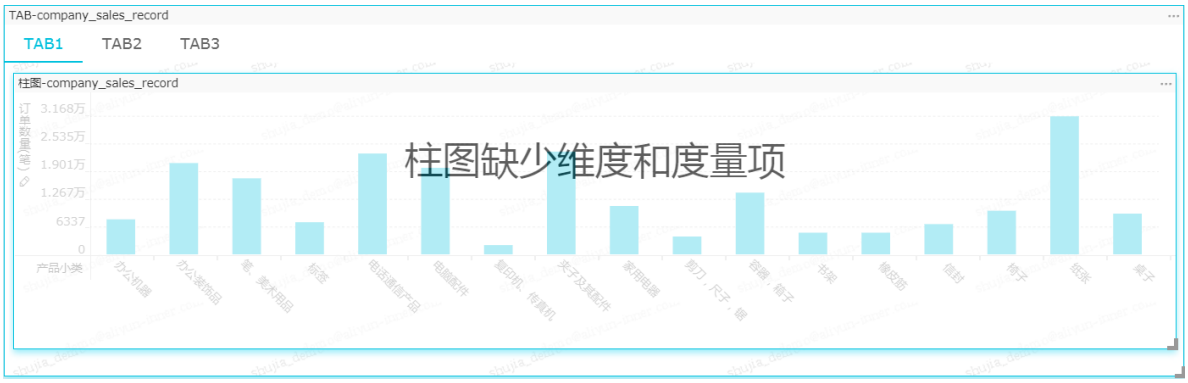
3. 选择一个TAB页来插入图表，如图 *9-119: TAB*标签页所示。

   单击TAB1，TAB1的颜色会被标蓝。

图 9-119: TAB标签页

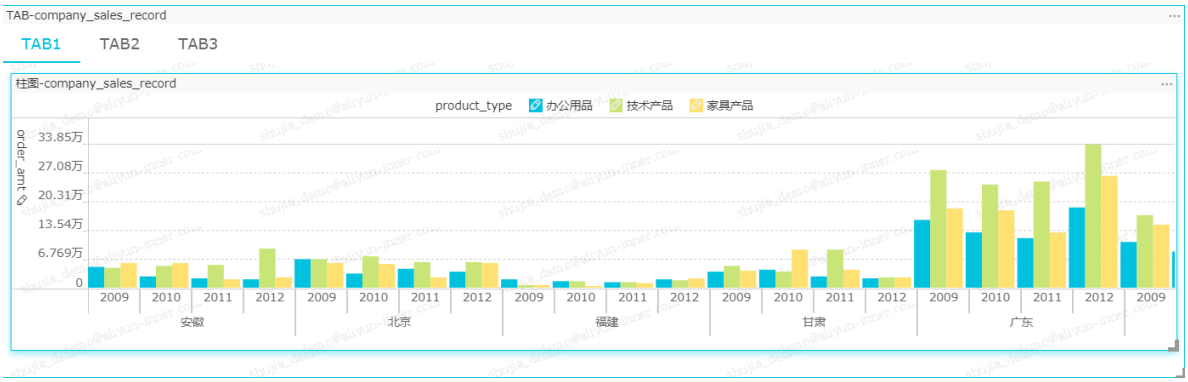
4. 双击图表图标，将图表添加到TAB1上，如图 *9-120:* 添加图表所示。

图 9-120: 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 *9-121: TAB*标签页展示所示。

图 9-121: 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。

9.4.3.3.7.5 PIC

背景信息

您可以通过PIC功能，在仪表板中插入图片，并根据展示的需求，调整图片的位置和效果。

操作步骤

1. 双击PIC图标。
2. 输入图片的链接地址。
3. 选择图片的展示效果，如图 *9-122: PIC*编辑菜单所示。

图 9-122: PIC编辑菜单

9.4.4 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。

9.4.4.1 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 *9-123: 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 9-123: 线图的类别轴、值轴

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**数据集**图标，进入数据集管理页面。
3. 找到company_sales_record数据集，单击后面的**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 *9-124:* 选择线图字段所示。

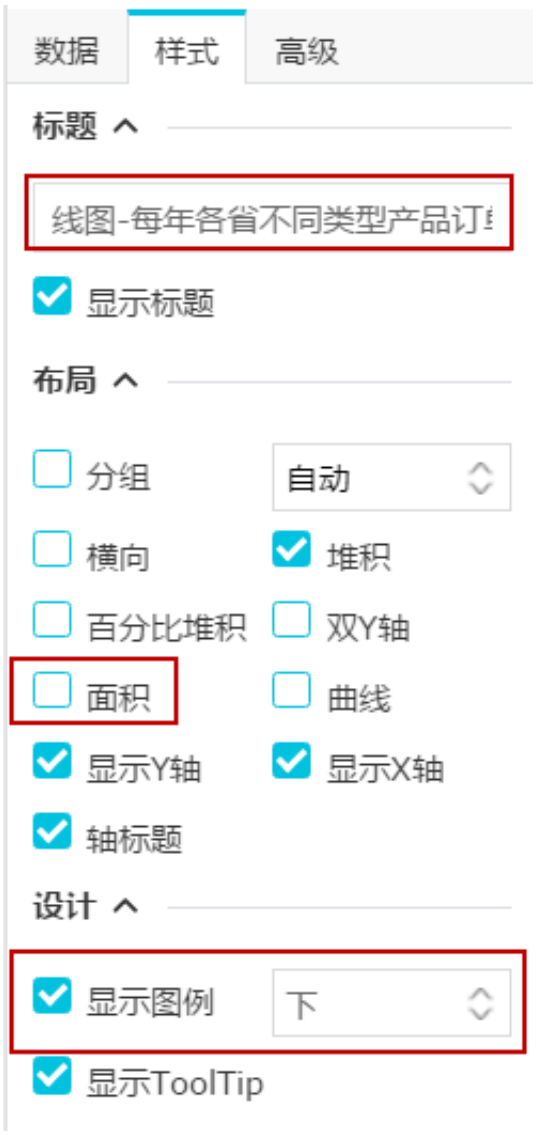
   说明：
   请确保省份字段的维度类型已经从字符串切换为了地理信息。

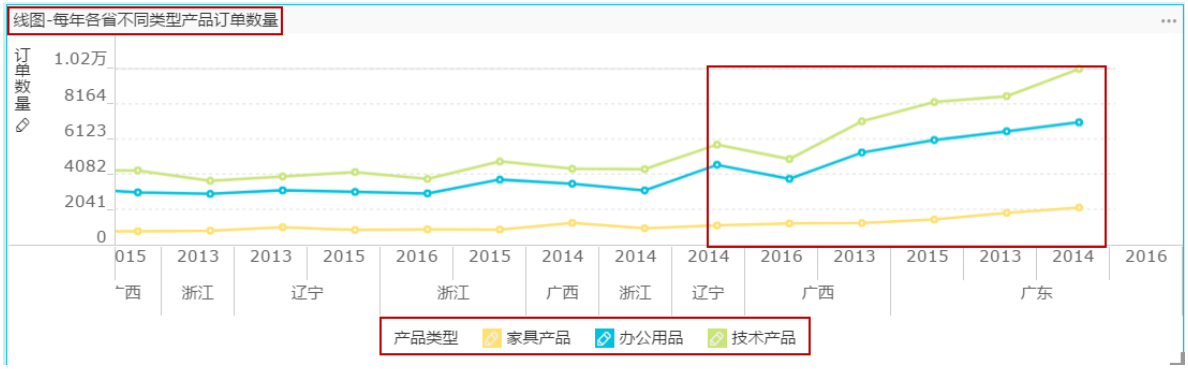
   切换维度字段类型，请参见编辑维度。

图 9-124: 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 *9-125:* 编辑后的线图所示。

图 9-125: 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。

9.4.4.2 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

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。

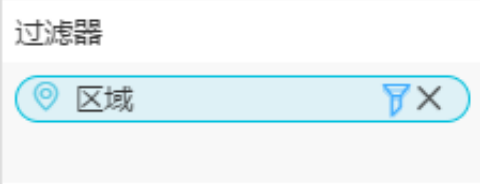
场景一：以company_sales_record数据集为例，用柱图展示，华东各省不同产品的运输成本比较。

操作步骤

1. 登录Quick BI控制台。
2. 在左侧导航栏中单击**数据集**，进入数据集管理页面。
3. 找到company_sales_record数据集，单击后面的**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 *9-126: 过滤器*所示。

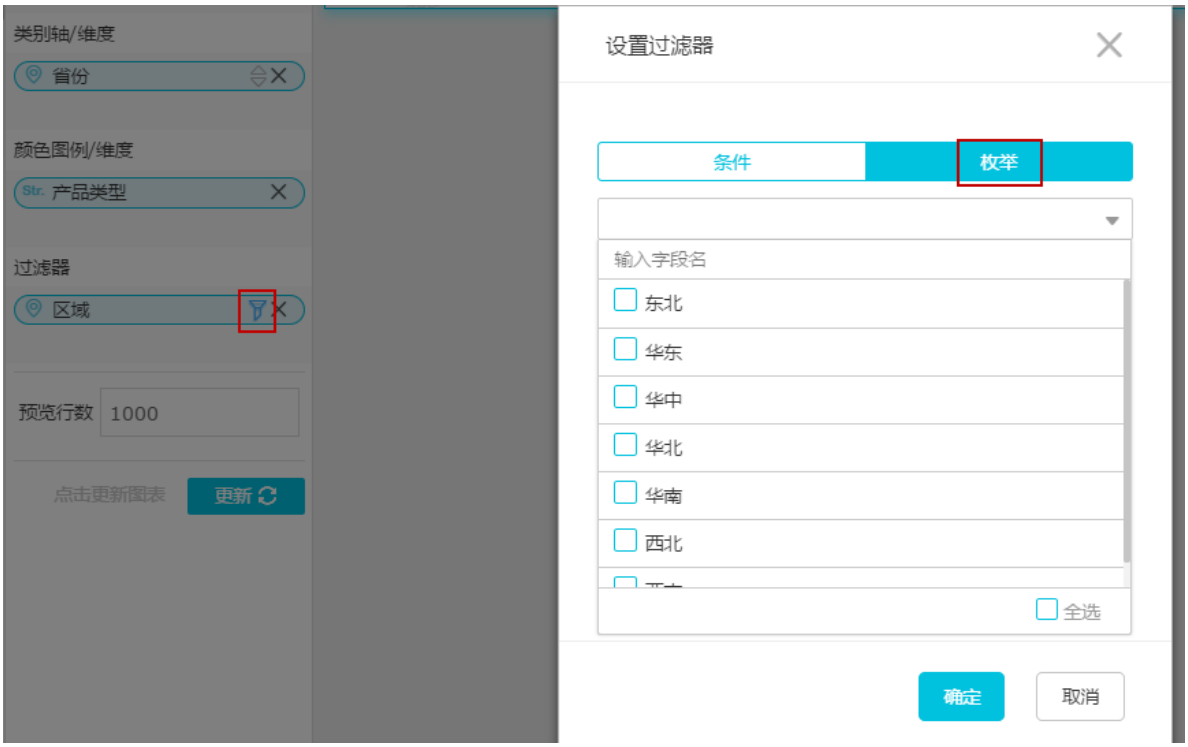
   我们需要通过过滤器，将**华东**从区域中筛选出来。

   图 9-126: 过滤器

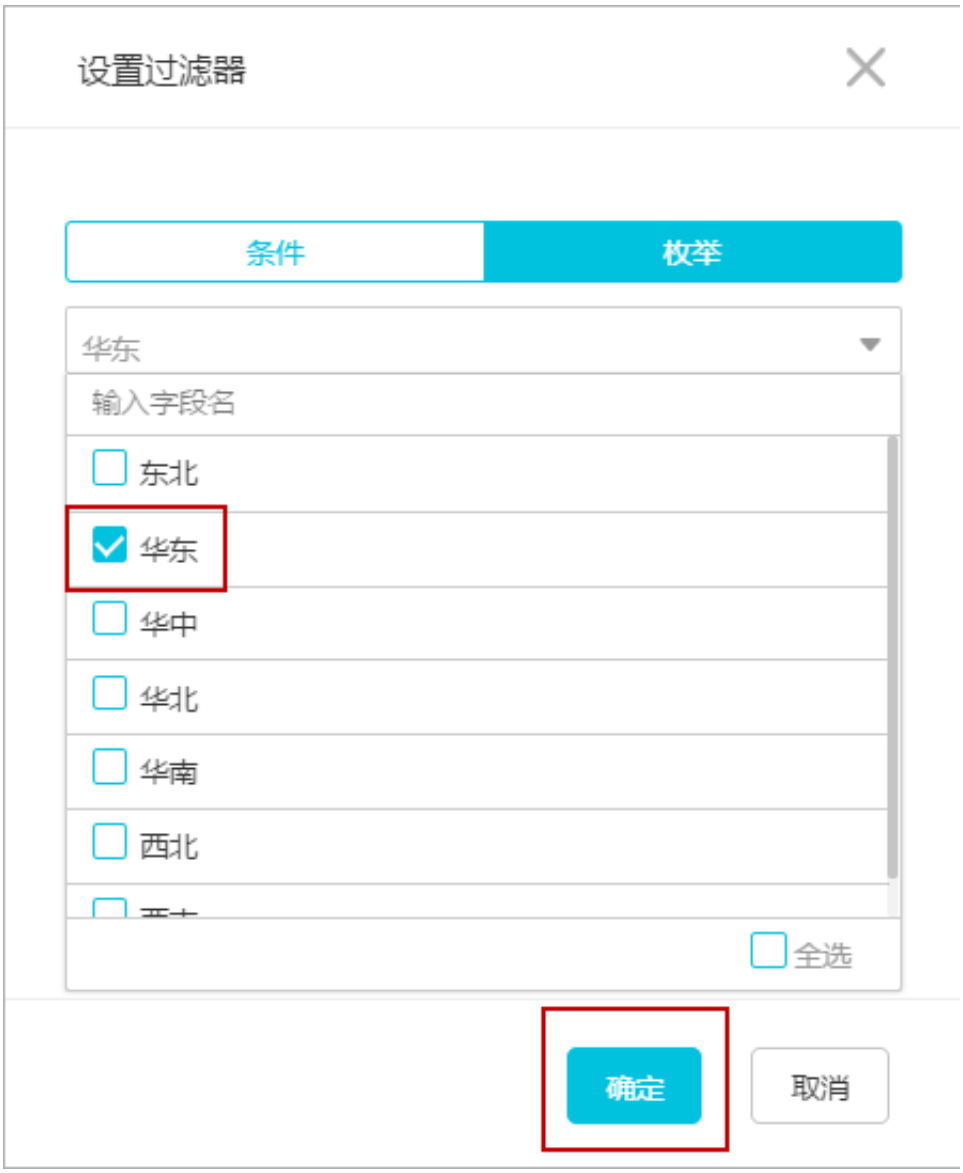
6. 单击**过滤**图标，设置过滤条件。
7. 选择**枚举**，系统会自动列出区域下所有可选择的选项，如图 *9-127: 枚举*所示。

图 9-127: 枚举



8. 选择**华东**，并单击**确定**，如图 *9-128:* 选择枚举选项所示。

图 9-128: 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。

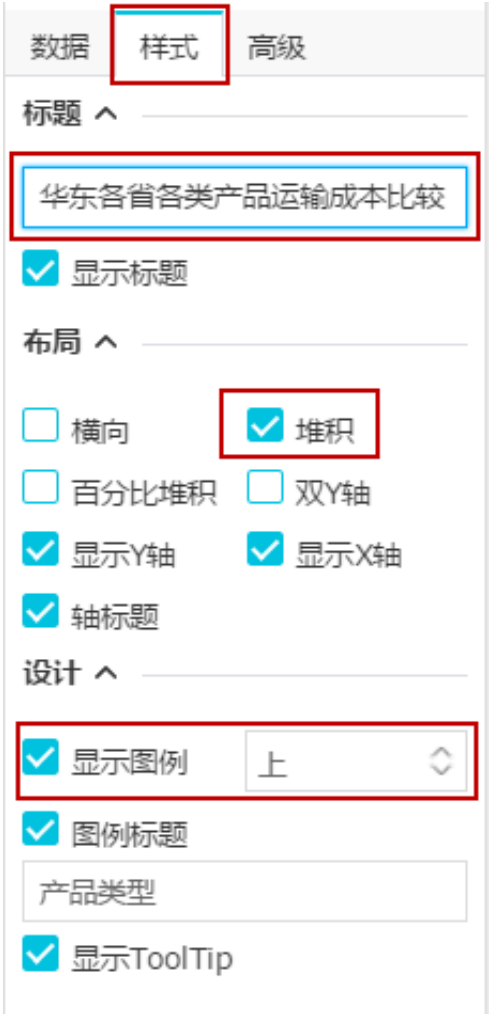
11. 将**产品类型**字段拖拽到**颜色图例**中，如图 *9-129: 启用颜色图例*所示。

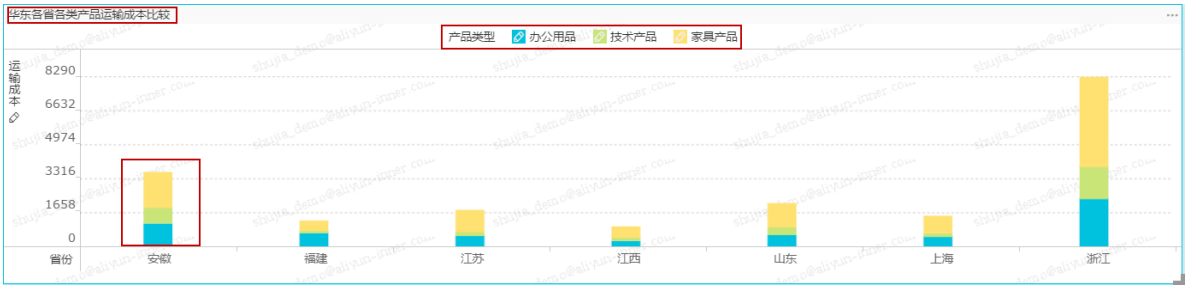
图 9-129: 启用颜色图例

12.单击**更新**，更新图表。

13.单击**样式**标签页，选择**堆积**，如图 *9-130:* 编辑后的柱图所示。

图 9-130: 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。

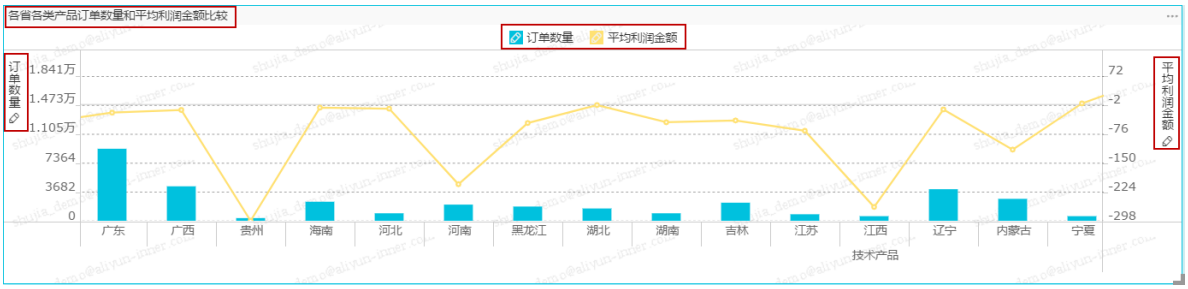
在维度列表中，找到省份和产品类型，并将它们依次添加到类别轴区域中；在度量列表中，找到订单数量和平均利润金额，并将它们依次添加到值轴区域中。

图 9-131: 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 *9-132:* 编辑后的柱图所示。

图 9-132: 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。

9.4.4.3 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。

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**数据集**，进入数据集管理页面。
3. 找到company_sales_record数据集，单击后面的**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，并将其添加到扇区大小区域中，如图 *9-133:* 选择饼图字段所示。

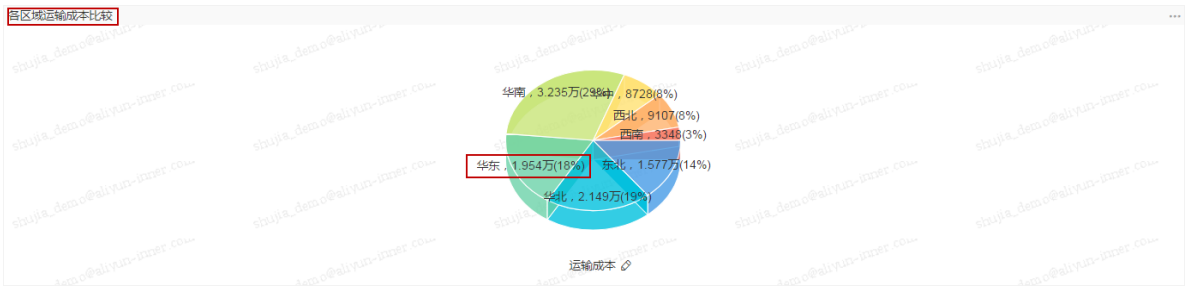
说明：

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 9-133: 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 *9-134:* 编辑后的饼图所示。

图 9-134: 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。

9.4.4.4 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。

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**数据集**，进入数据集管理页面。
3. 找到company_sales_record数据集，单击后面的**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。

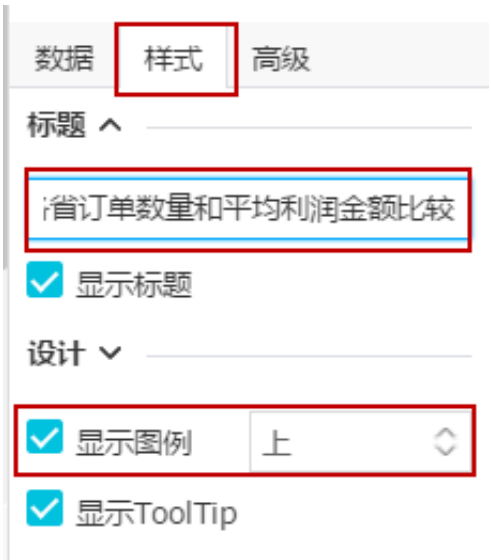
   在维度列表中，找到省份，并将其添加到地理区域中；在度量列表中，找到订单数量和平均利润金额，并将依次添加到气泡大小区域中，如图 *9-135:* 选择气泡地图字段所示。

   > **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 9-135: 选择气泡地图字段

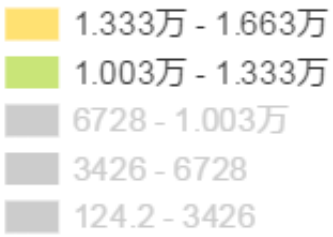
6. 单击**更新**，更新图表。
7. 单击**样式**标签页，更改图表的显示标题和显示图例，如图 *9-136: 编辑后的气泡地图*所示。

图 9-136: 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。

9.4.4.5 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。

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**数据集**，进入数据集管理页面。
3. 找到company_sales_record数据集，单击后面的**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 *9-137:* 选择色彩地图字段所示。

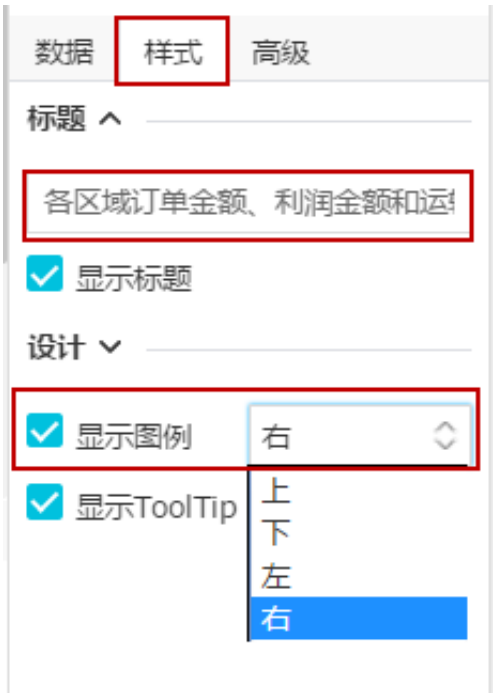
说明：

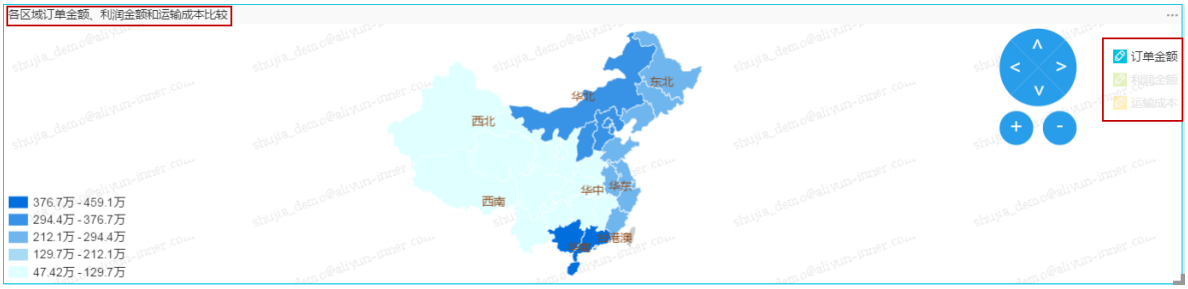
请确保区域字段的维度类型已经从字符串切换为了地理信息，切换维度字段类型，请参见编辑维度。

图 9-137: 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 *9-138:* 编辑后的色彩地图所示。

图 9-138: 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。

9.4.4.6 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。

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**数据集**，进入数据集管理页面。
3. 找到company_sales_record数据集，单击后面的**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 *9-139: 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 9-139: 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 *9-140: 编辑后的交叉表*所示。

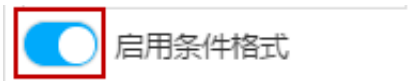
图 9-140: 编辑后的交叉表

	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 *9-141: 启用条件格式*所示。

图 9-141: 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 *9-142: 规则设置*所示。

图 9-142: 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 *9-143: 数值显示样式*所示。

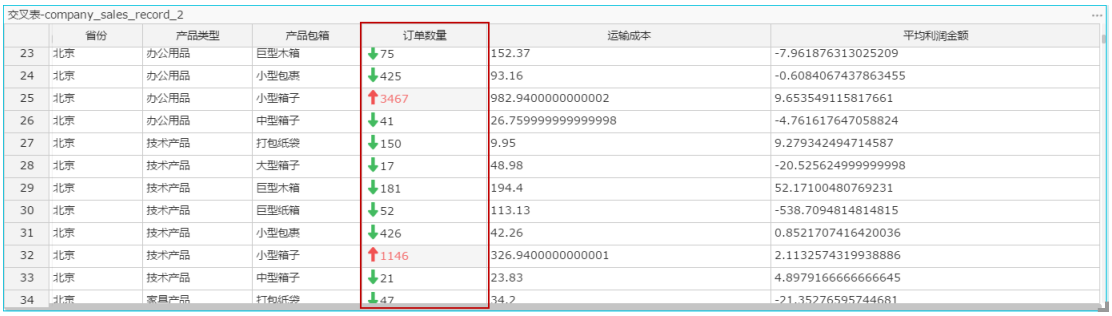
图 9-143: 数值显示样式

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 *9-143:* 样式编辑示例所示。

图 9-143: 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 9-143: 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	-21.352765957446811

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。

9.4.4.7 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。

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。

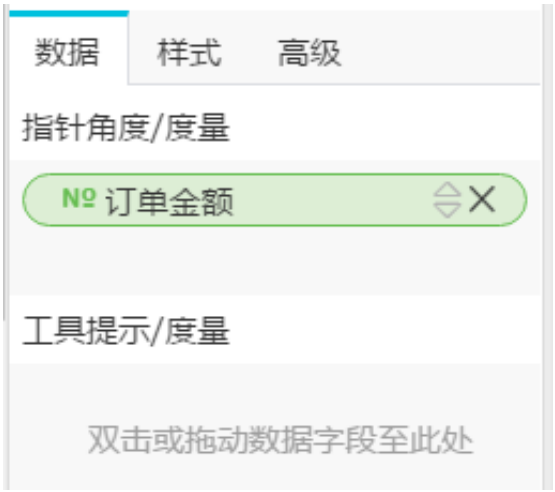
操作步骤

1. 登录Quick BI控制台。
2. 在左侧导航栏中单击**数据集**，进入数据集管理页面。
3. 找到company_sales_record数据集，单击后面的**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 *9-146: 选择仪表盘字段*所示。

图 9-146: 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 *9-147: 编辑仪表盘样式*所示。

图 9-147: 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 *9-148:* 设置展示区间所示。

图 9-148: 设置展示区间

9. 单击色块，更改区间显示的配色方案，如图 *9-149:* 更改区间的配色方案所示。

图 9-149: 更改区间的配色方案

10.单击**更新**，更新图表，如图 *9-150:* 编辑后的仪表盘所示。

图 9-150: 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。

9.4.4.8 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。

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**数据集**，进入数据集管理页面。
3. 找到company_sales_record数据集，单击后面的**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。

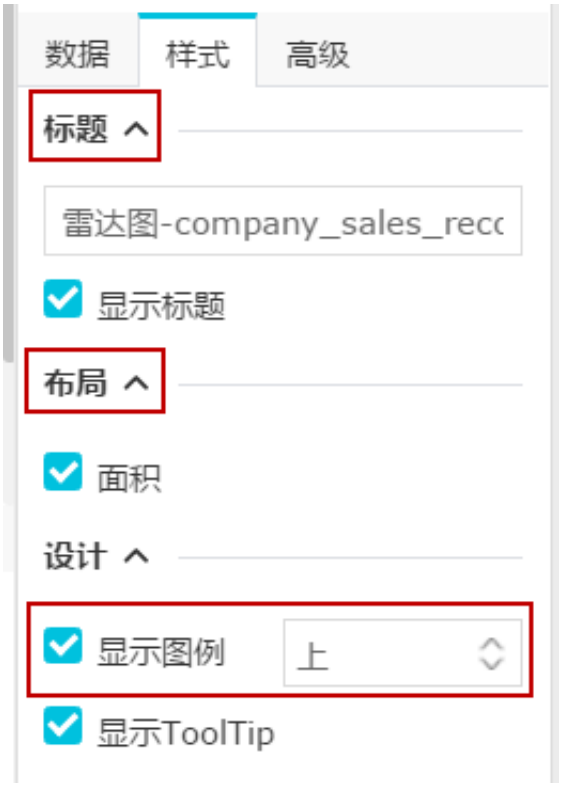
   在维度列表中，找到区域，并将其添加到分支标签区域中；在度量列表中，找到订单数量和订单金额，并将它们依次添加到分支长度区域中，如图 *9-151:* 选择雷达图字段所示。

   > **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 9-151: 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 *9-152:* 编辑后的雷达图所示。

图 9-152: 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。

9.4.4.9 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。

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。

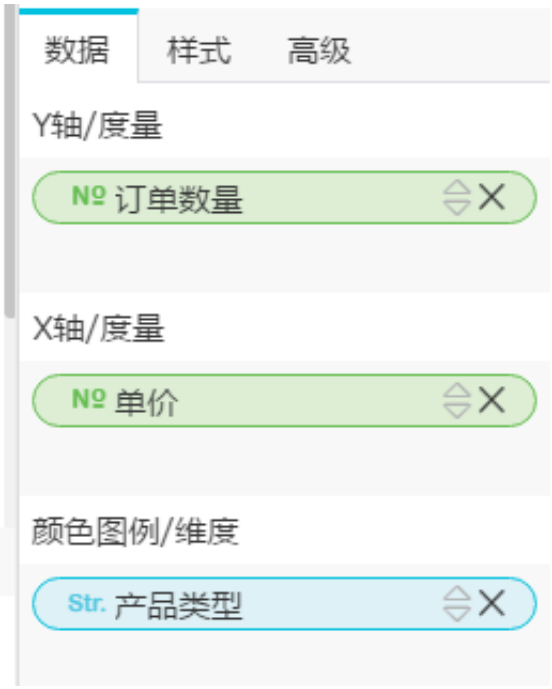
操作步骤

1. 登录Quick BI控制台。
2. 在左侧导航栏中单击**数据集**，进入数据集管理页面。
3. 找到company_sales_record数据集，单击后面的**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。

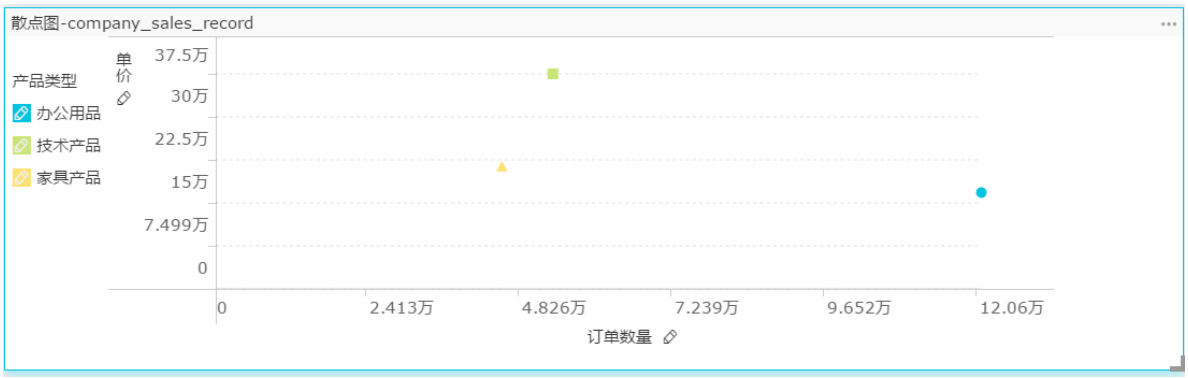
   在维度列表中，找到产品类型，并将其添加到颜色图例区域；在度量列表中，找到单价和订单数量，并将它们依次添加到Y轴和X轴区域，如图 *9-153: 选择散点图字段*所示。

图 9-153: 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 *9-154: 编辑后的散点图*所示。

图 9-154: 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。

9.4.4.10 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。

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。

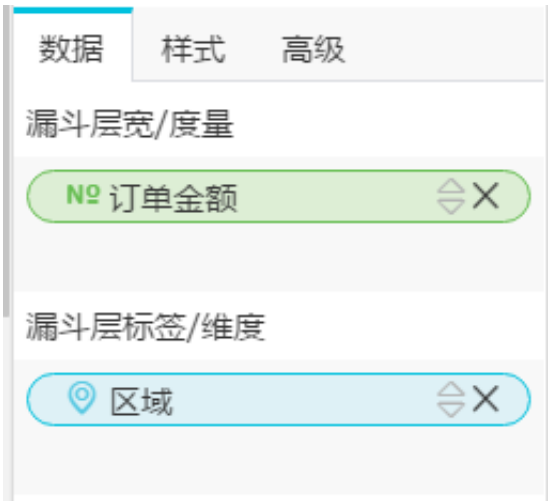
操作步骤

1. 登录Quick BI控制台。
2. 在左侧导航栏中，单击**数据集**，进入数据集管理页面。
3. 找到company_sales_record数据集，单击后面的**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 *9-155:* 选择漏斗图字段所示。

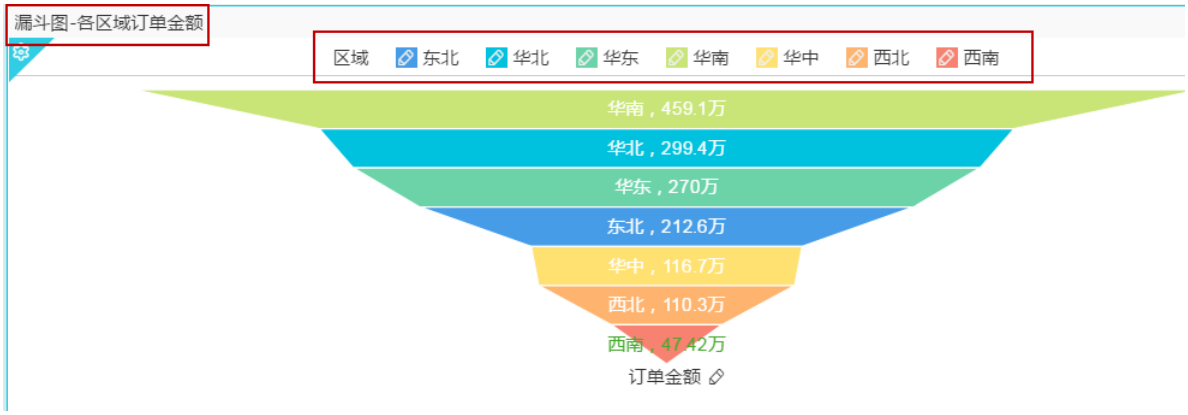
   > **说明**：
   > 请确保区域字段的维度类型已经从字符串切换为了地理信息，切换维度字段类型，请参见编辑维度。

图 9-155: 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 *9-156:* 编辑后的漏斗图所示。

图 9-156: 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。

9.4.4.11 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。

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**数据集**，进入数据集管理页面。
3. 找到company_sales_record数据集，单击后面的**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 *9-157:* 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 9-157: 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 *9-158:* 编辑后的指标看板所示。

图 9-158: 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。

9.4.4.12 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。

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

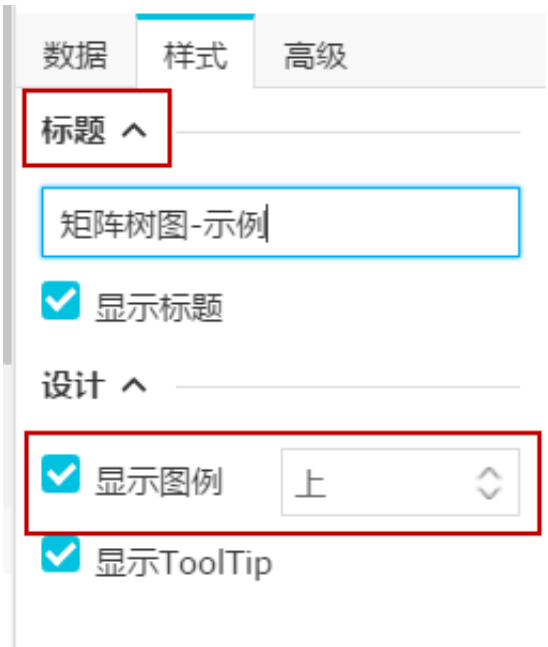
1. 登录Quick BI控制台。
2. 在左侧导航栏中，单击**数据集**，进入数据集管理页面。
3. 找到company_sales_record数据集，单击后面的**新建仪表板**，进入仪表板编辑页面。
4. 在仪表板配置区，双击**矩阵树图**图标。
5. 在**数据**标签页，选择需要的维度字段和度量字段。

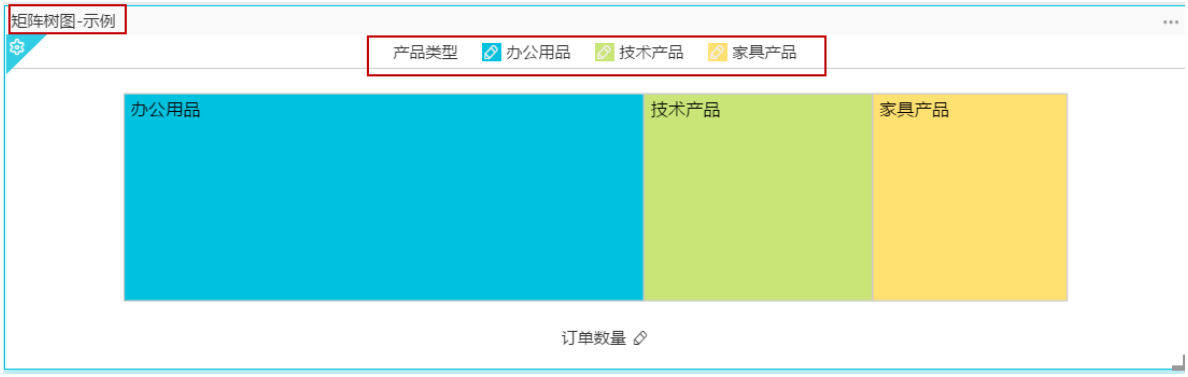
   在维度列表中，找到产品类型，并将其添加到色块标签区域；在度量列表中，找到订单数量，并将其添加到色块大小区域，如图 *9-159:* 选择矩阵树图字段所示。

图 9-159: 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 *9-160:* 编辑后的矩阵树图所示。

图 9-160: 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。

9.4.4.13 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。

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**数据集**，进入数据集管理页面。
3. 找到company_sales_record数据集，单击后面的**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。

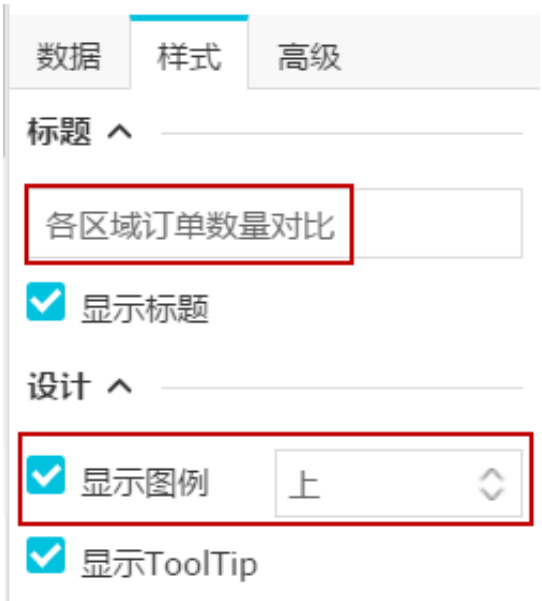
   在维度列表中，找到区域，并将其添加到扇区标签区域中；在度量列表中，找到订单数量，并将其添加到扇区长度区域中，如图 *9-161: 选择极坐标图字段*所示。

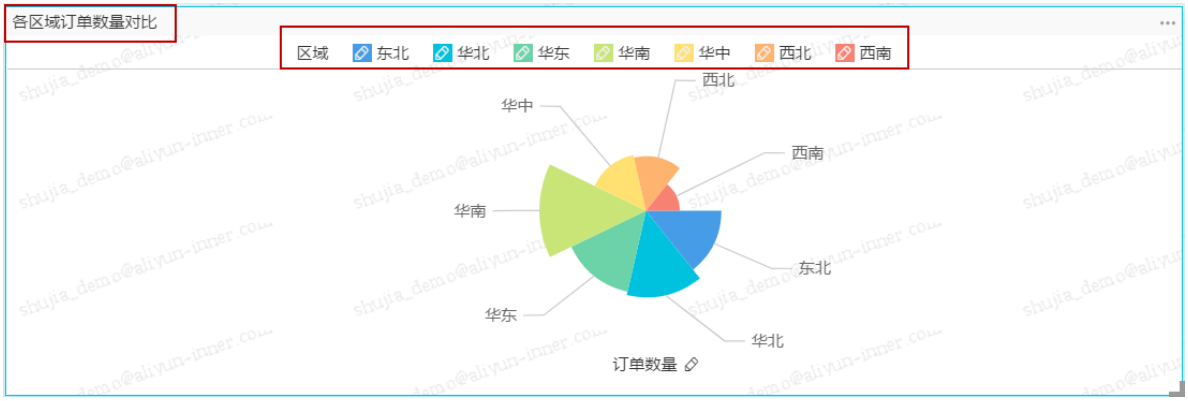
说明：
请确保区域字段的维度类型已经从字符串切换为了地理信息。切换维度字段类型，请参见编辑维度。

图 9-161: 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 *9-162:* 编辑后的极坐标图所示。

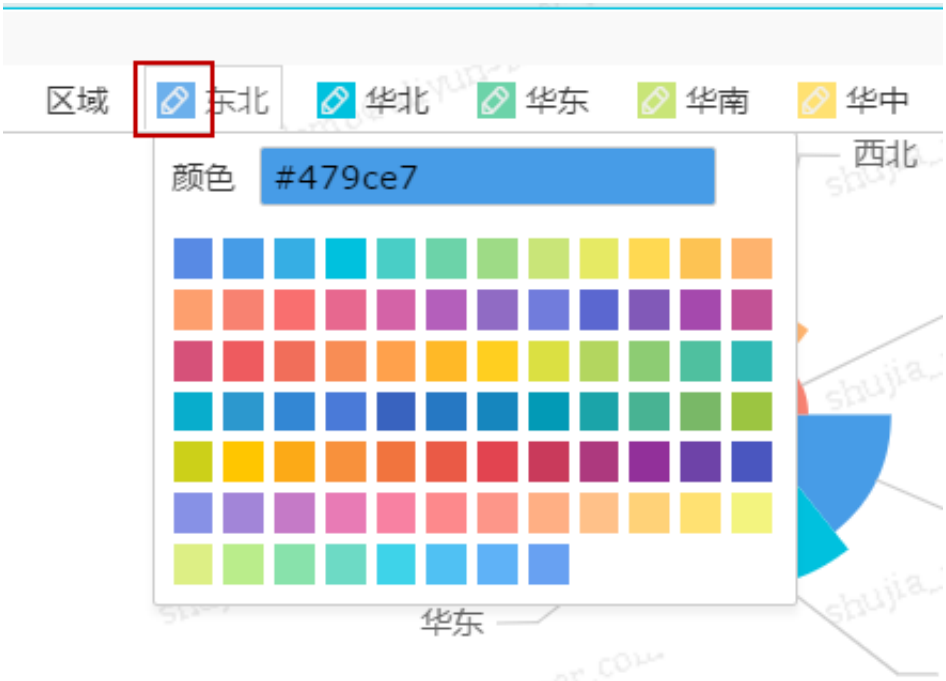
图 9-162: 编辑后的极坐标图





8. 单击图例前面的色块，更改显示图例的配色方案。

图 9-163: 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。

9.4.4.14 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。

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**数据集**，进入数据集管理页面。
3. 找到company_sales_record数据集，单击后面的**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 *9-164: 选择词云图字段*所示。

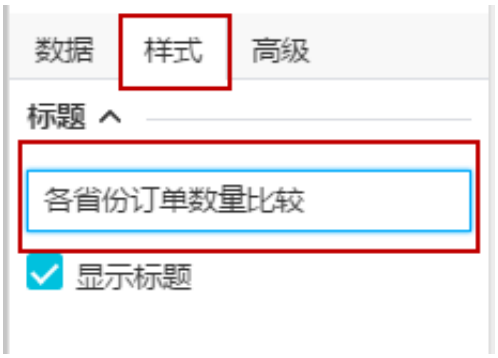
   > **说明**：
   > 请确保省份字段的维度类型已经从字符串切换为了地理信息。切换维度字段类型，请参见编辑维度。

   图 9-164: 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 *9-165: 编辑后的词云图*所示。

图 9-165: 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。

9.4.4.15 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。

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**数据集**图标，进入数据集管理页面。
3. 找到company_sales_record数据集，单击后面的**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 *9-166: 选择旋风漏斗字段*所示。

   图 9-166: 选择旋风漏斗字段

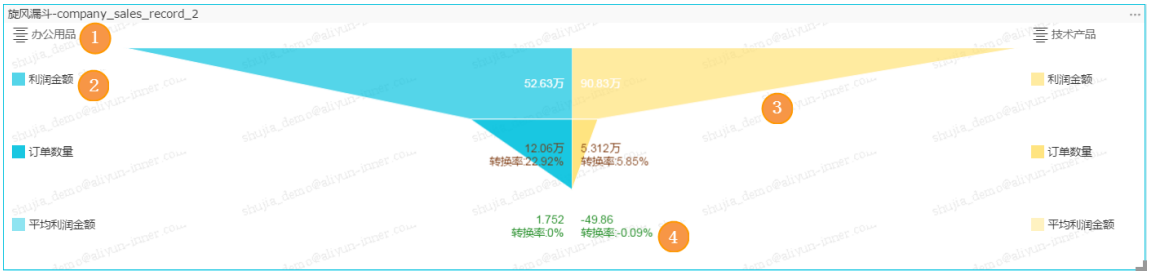
6. 单击**更新**，更新图表。
7. 在**样式**标签页，更改图表的显示标题、图形布局、图例位置、配色方案和转化率显示。

   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 9-167: 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 *9-168: 编辑后的旋风漏斗图*所示。

图 9-168: 编辑后的旋风漏斗图

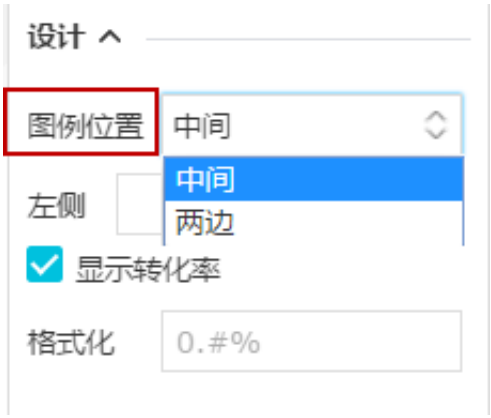


- （1）鼠标指向产品类型字段，可以任意切换产品类型，如图 *9-168: 切换产品类型*所示。

图 9-168: 切换产品类型

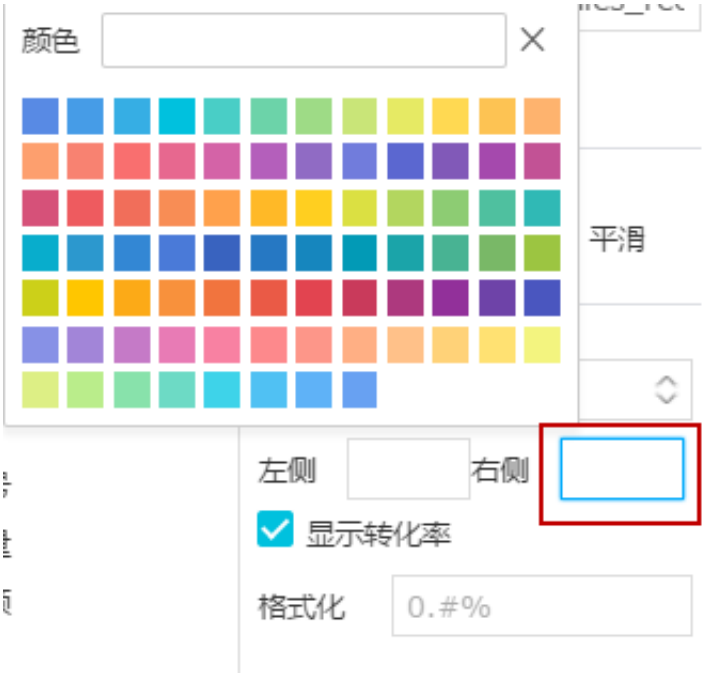
- （2）调整显示图例的展示位置，如图 9-169: 图例位置所示。

图 9-169: 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 9-170: 更改配色方案所示。

图 9-170: 更改配色方案



- （4）手动调整转化率的格式，如图 9-171: 转化率格式所示。

图 9-171: 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。

9.4.4.16 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

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：各区域下各省份不同产品的订单数量比较

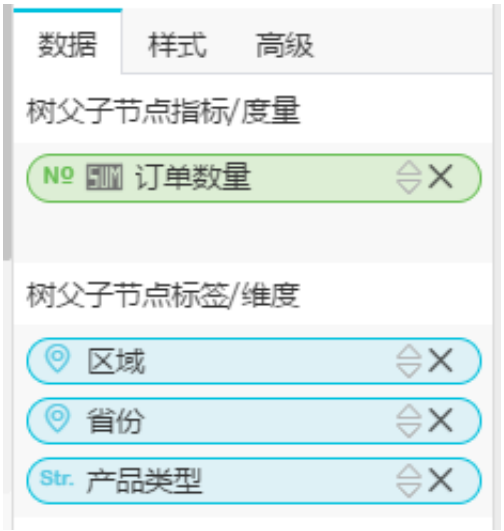
操作步骤

1. 登录 Quick BI 控制台。
2. 在左侧导航栏中，单击**数据集**，进入数据集管理页面。
3. 找到company_sales_record数据集，单击后面的**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 *9-173:* 选择树图字段所示。

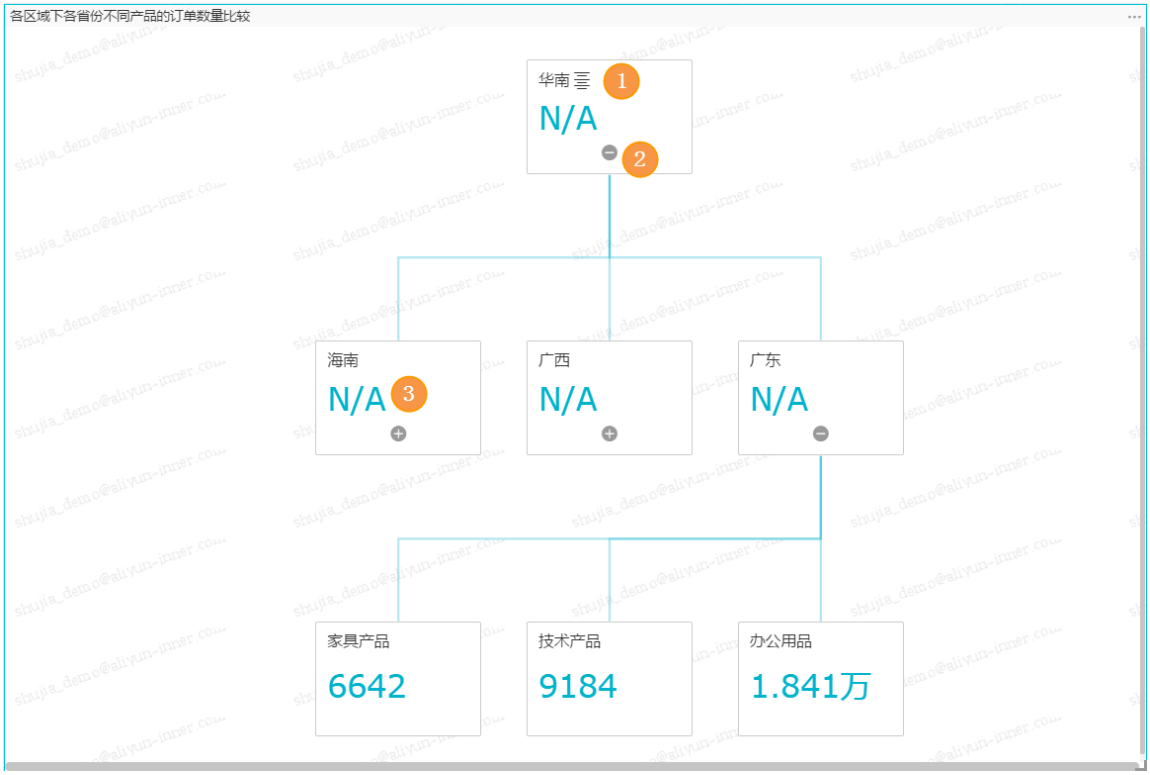
说明：
请确保区域字段和省份字段的维度类型已经从字符串切换为了地理信息。切换维度字段类型，请参见编辑维度。

图 9-173: 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 *9-174:* 图表布局所示。

图 9-174: 图表布局

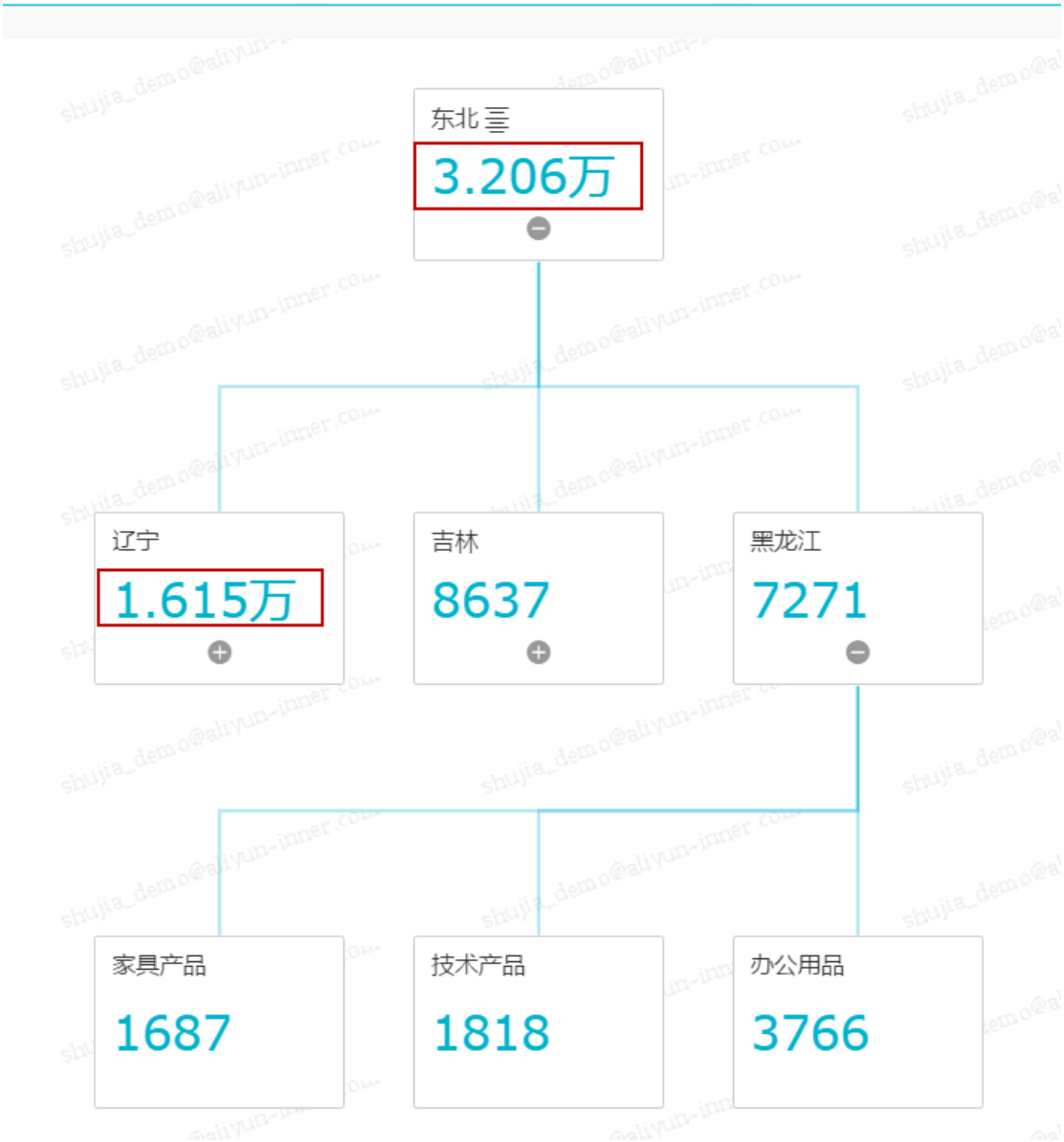


- （1）鼠标指向区域字段，可以任意切换区域名称，如图 *9-174:* 切换区域所示。

图 9-174: 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 *9-175: 自动汇总*所示。

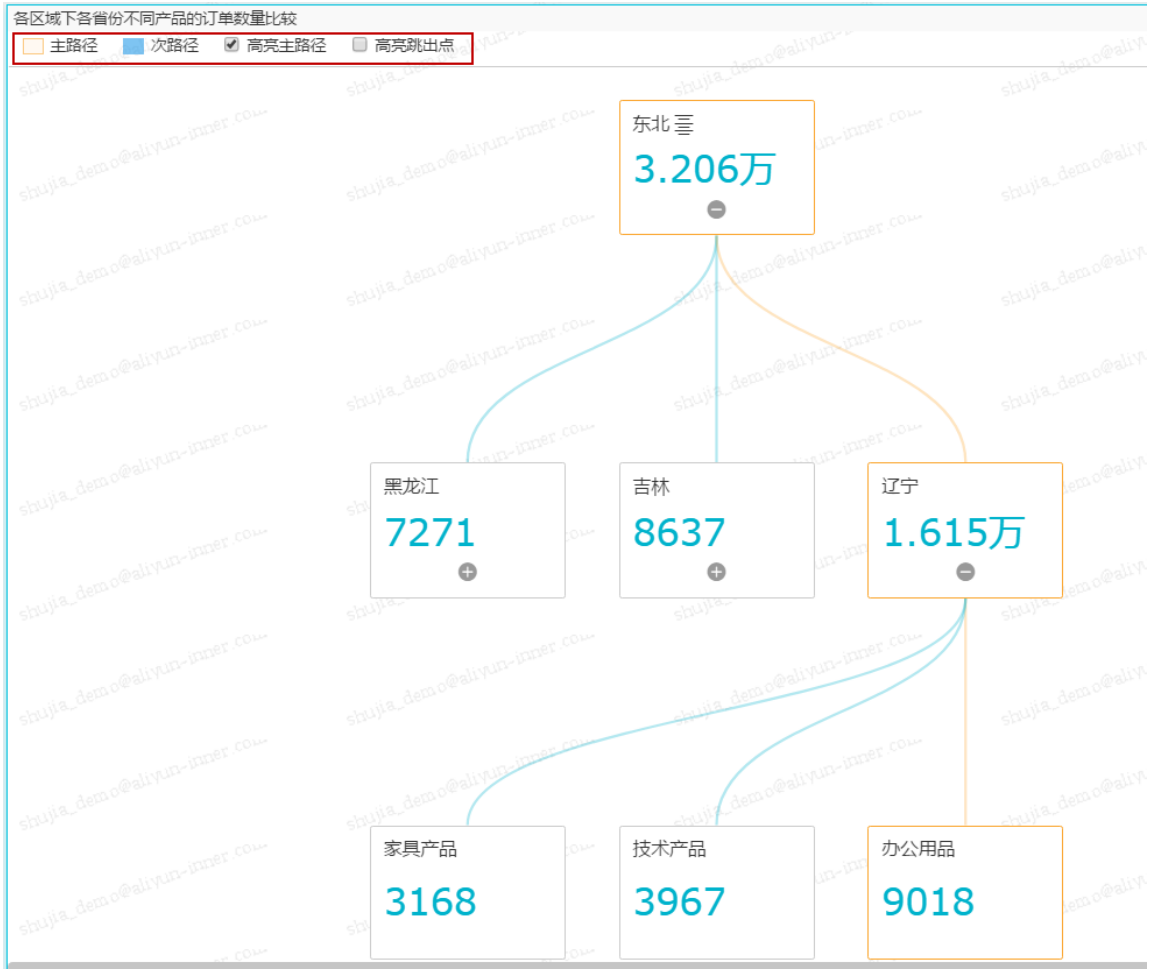
图 9-175: 自动汇总



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 *9-177:* 编辑后的树图所示。

图 9-177: 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

背景信息

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

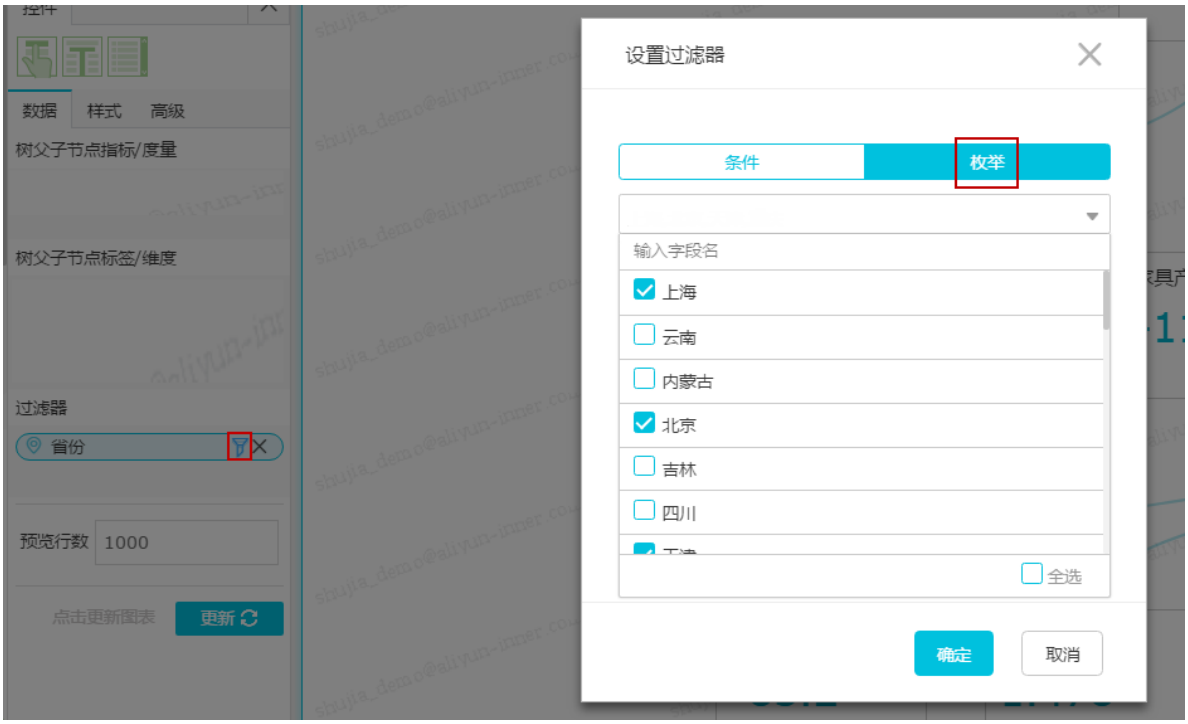
1. 在维度列表中，找到省份字段，并将其添加到过滤器中。

   我们需要通过过滤器将直辖市从省份中筛选出来。

2. 单击**筛选**图标，选择**枚举**，如下图所示。

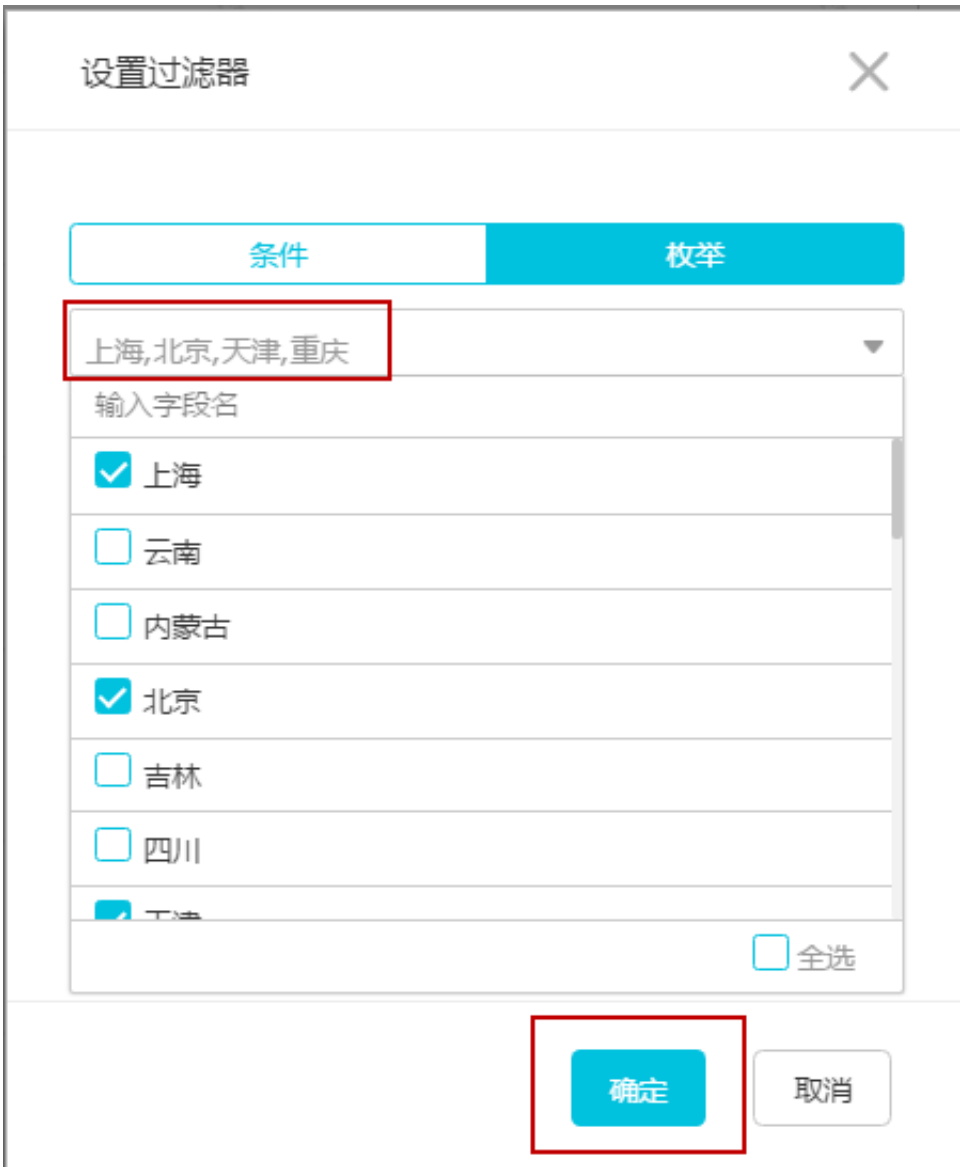
   系统会自动将省份字段中所有可选的信息列图 *9-178:* 枚举在下拉菜单中。

图 9-178: 枚举



3. 选择需要的城市，或者手动输入字段名。
4. 单击**确定**，完成过滤条件设置，如图 *9-179:* 选择枚举条件所示。

图 9-179: 选择枚举条件



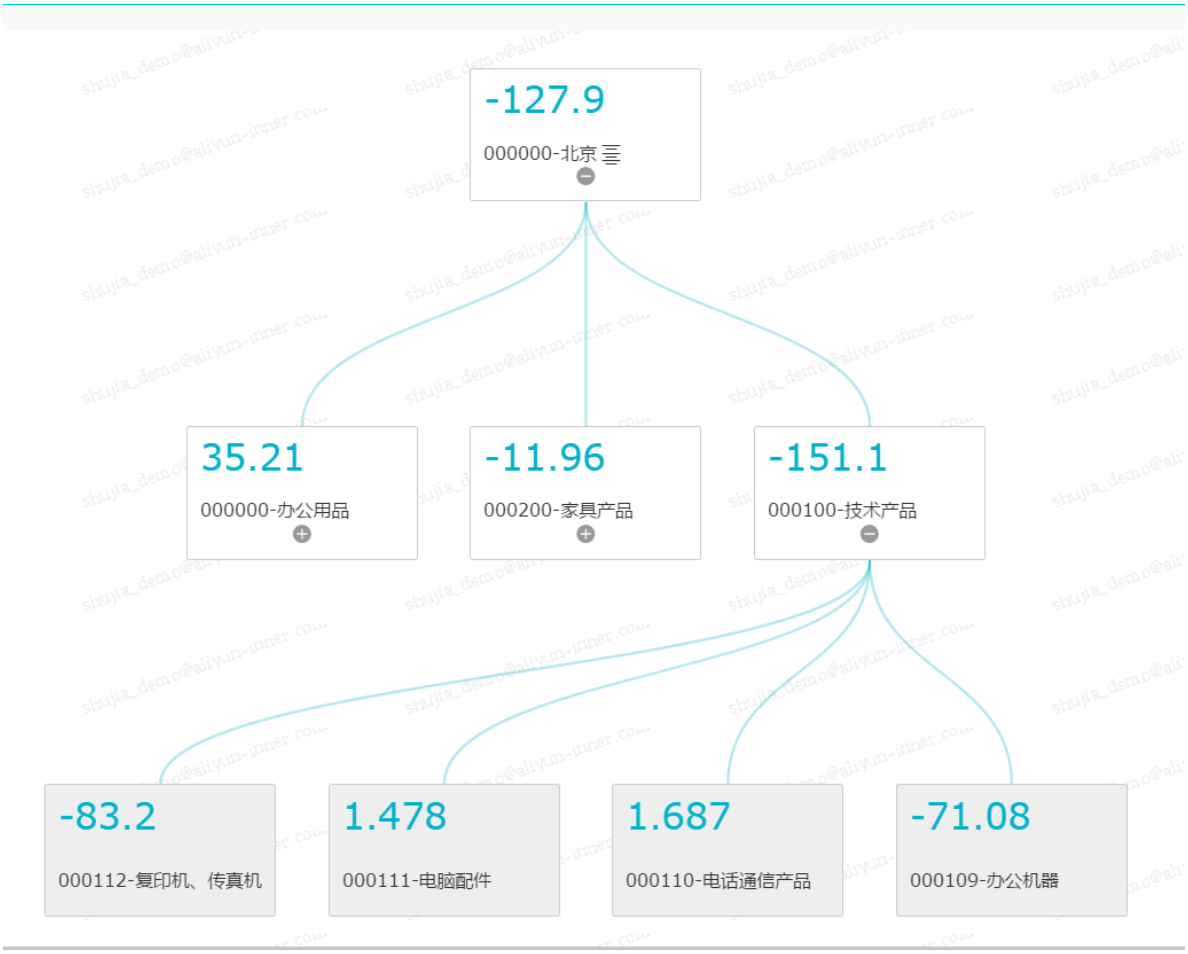
5. 继续从维度列表中，找到城市，产品类型和产品小类，并将它们依次添加到树父子节点标签区域，如图 *9-180:* 选择树图字段所示。

   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 9-180: 选择树图字段

6. 单击**更新**，更新图表。
7. 单击**样式**标签页，更改图标的显示标题，布局和设计，如图 *9-181:* 编辑后的树图所示。

图 9-181: 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。

9.4.4.17 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。

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**数据集**，进入数据集管理页面。
3. 找到page_source_target_day_stat数据集，单击后面的**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。

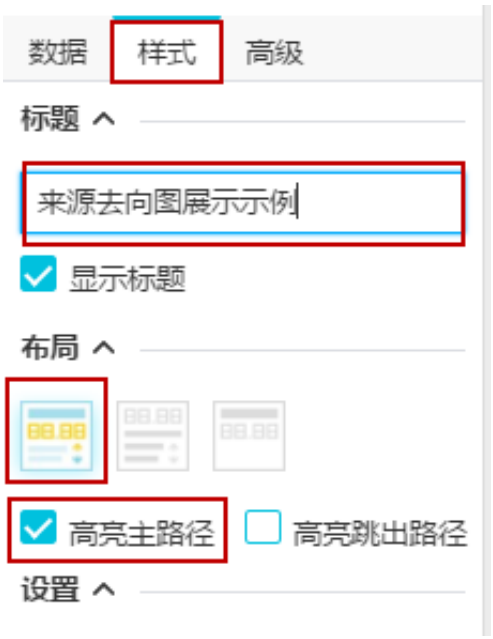
   在维度列表中，找到前一页面，当前页面和后一页面的名称，并将它们依次填入对应的区域。它们的排列顺序就是图表将要展示的层级关系；在度量列表，找到页面路径转化率和页面跳出率，并将它们依次填入到对应的区域，继续在度量列表中找到三个PV数值和UV数值，也将它们依次填入到对应的区域（也可以只选择三个PV数值或者三个UV数值填入）。

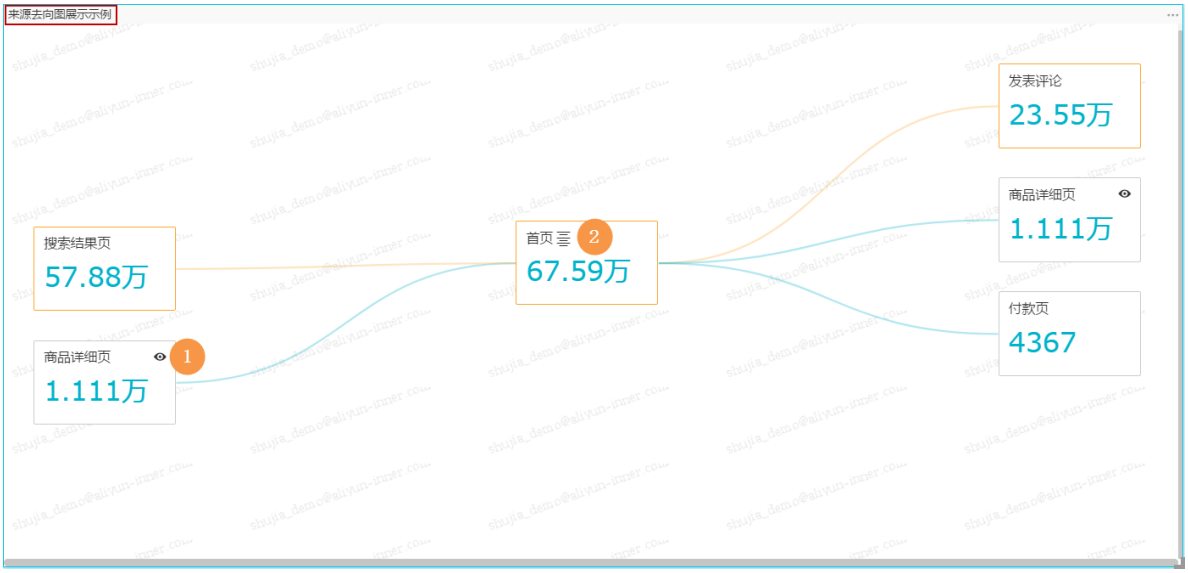
图 9-182: 选择来源去向图字段

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 *9-183:* 编辑后的来源去向图所示。

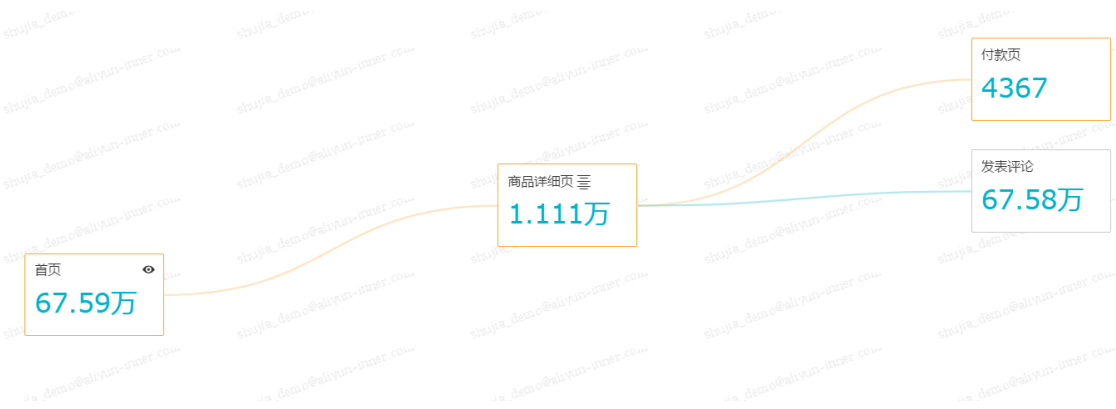
图 9-183: 编辑后的来源去向图





- （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 *9-183: 查看页面来源去向*所示。

图 9-183: 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 *9-184:* 切换分析页面所示。

图 9-184: 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。

9.4.5 查询仪表板

操作步骤

1. 在Quick BI控制台，单击**仪表板**，进入仪表板管理页面。
2. 在搜索框中，通过输入仪表板的关键字来查询仪表板。

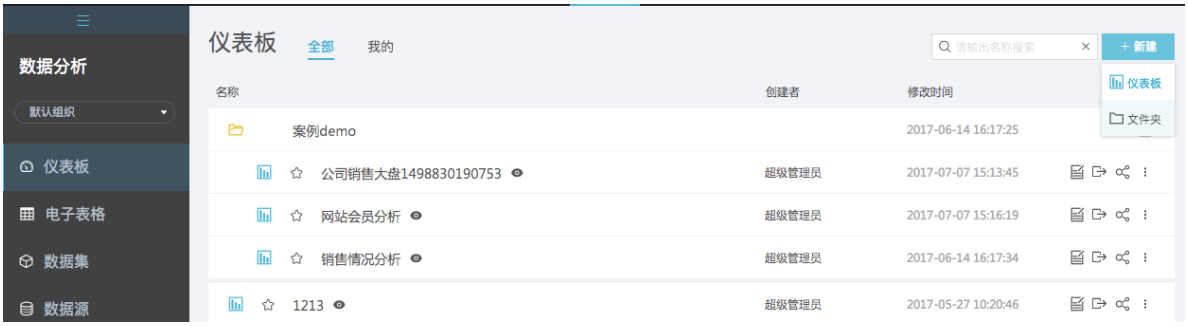
9.4.6 新建仪表板文件夹

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 *9-186: 新建文件夹*所示。

图 9-186: 新建文件夹



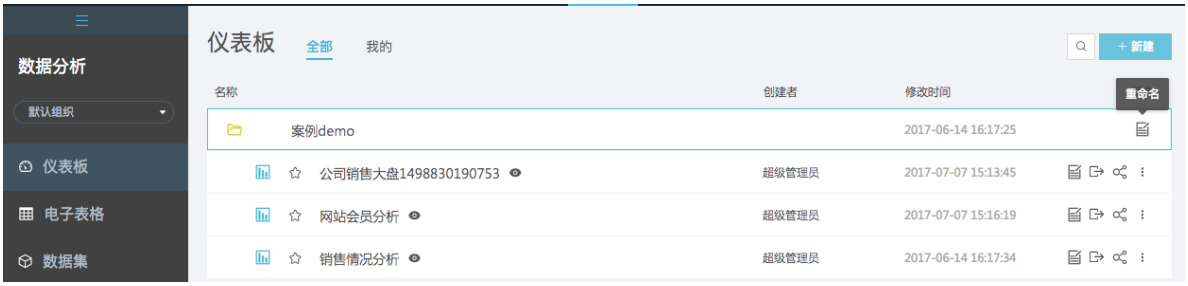
3. 输入一个文件夹名称，单击页面空白处即可。

9.4.7 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 *9-187: 重命名仪表板文件夹*所示。

图 9-187: 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。

9.4.8 分享仪表板

操作步骤

1. 在Quick BI控制台，单击**仪表板**，进入仪表板管理页面。
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 *9-188: 仪表板分享*所示。

图 9-188: 仪表板分享

5. 单击**保存**。

9.4.9 公开仪表板

操作步骤

1. 在Quick BI控制台，单击**仪表板**，进入仪表板管理页面。
2. 选择一个要公开的仪表板，单击后面的省略号图标或者鼠标右键。
3. 选择**公开**选项。
4. 选择一个公开的截止日期，并单击**公开**，如图 *9-189:* 仪表板公开所示。

图 9-189: 仪表板公开

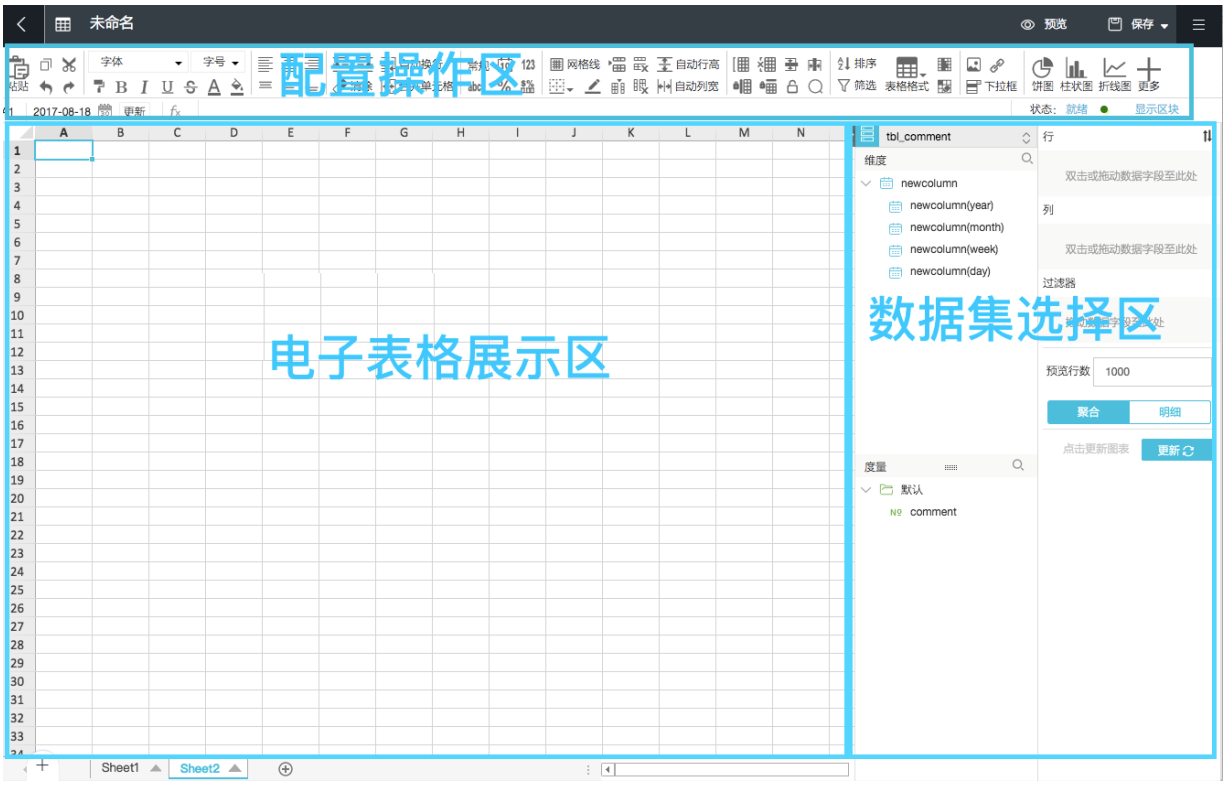
9.5 电子表格操作

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 9-190: 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。

9.5.1 新建电子表格

背景信息

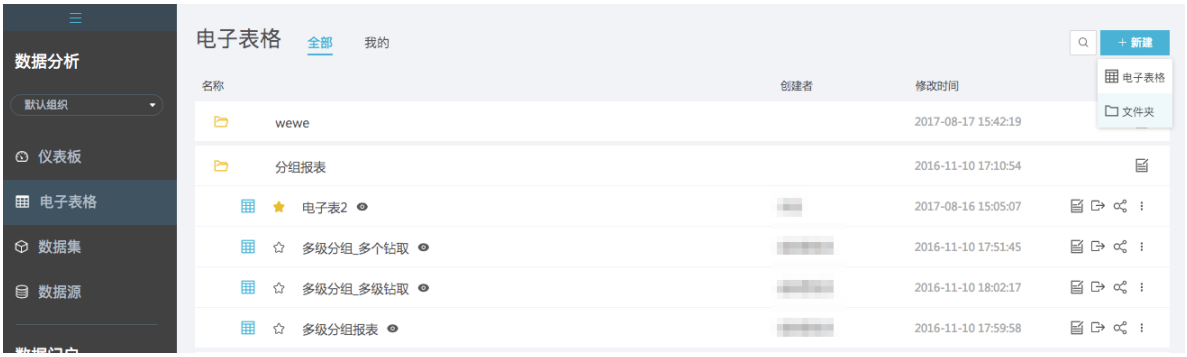
电子表格功能只适用于工作空间下，个人空间不支持该功能。

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 *9-191: 新建电子表格*所示。

图 9-191: 新建电子表格



9.5.2 切换数据集

在数据集选择区，您可以选择或切换数据集。

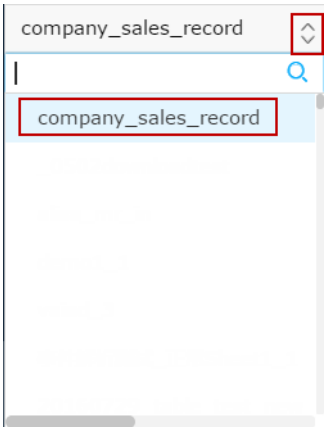
背景信息

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 *9-192: 切换数据集*所示。

图 9-192: 切换数据集



9.5.3 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

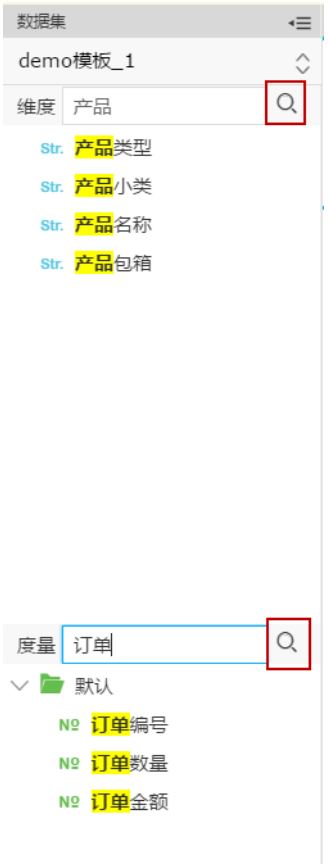
选择好数据集后，系统会自动将不同类型的列分别显示在维度和度量区域。

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. 单击搜索图标，，搜索相关字段。

图 9-193: 搜索字段



9.5.4 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控制台。
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 *9-194: 字体样式*所示。

图 9-194: 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 *9-195: 字体大小*所示。

图 9-195: 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。

9.5.5 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控制台。
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 *9-196: 对齐方式*所示。

图 9-196: 对齐方式

9.5.6 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控制台。
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 *9-197:* 显示格式所示。

图 9-197: 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。

9.5.7 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控制台。
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 *9-198:* 样式、单元格和窗口所示。

   图 9-198: 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。

9.5.8 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控制台。
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 *9-199: 插入超链接*所示。

图 9-199: 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 *9-200: 下拉框设置*所示。

图 9-200: 下拉框设置

12. 单击**确定**，插入下拉框。

9.5.9 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控制台。
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 *9-201:* 表格格式所示。

图 9-201: 表格格式

9.5.10 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控制台。
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 *9-202:* 高亮所示。

图 9-202: 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 *9-203: 数据栏*所示。

图 9-203: 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 *9-204: 图标*所示。

图 9-204: 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**。

9.5.11 查询电子表格

操作步骤

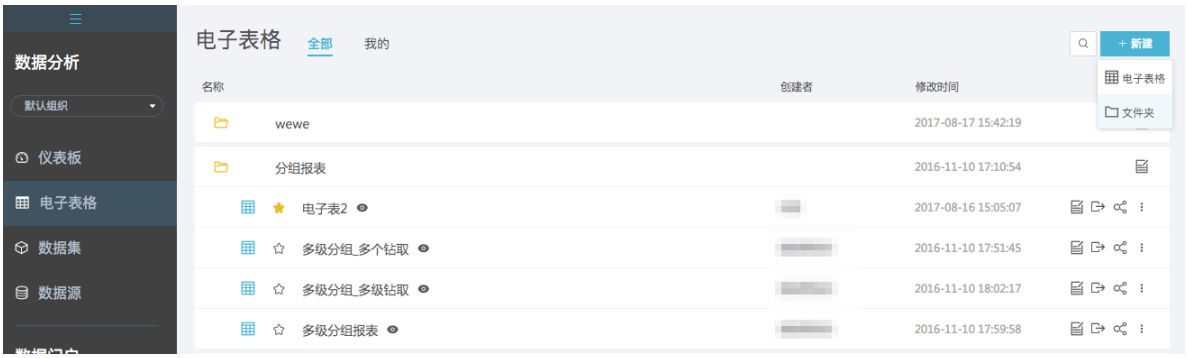
1. 登录Quick BI控制台。
2. 在左侧导航栏中，单击**电子表格**。
3. 在搜索框中输入电子表格的关键字。
4. 单击搜索图标，查询电子表格。

9.5.12 新建电子表格文件夹

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** > **文件夹**如图 *9-205: 新建文件夹*所示。

图 9-205: 新建文件夹



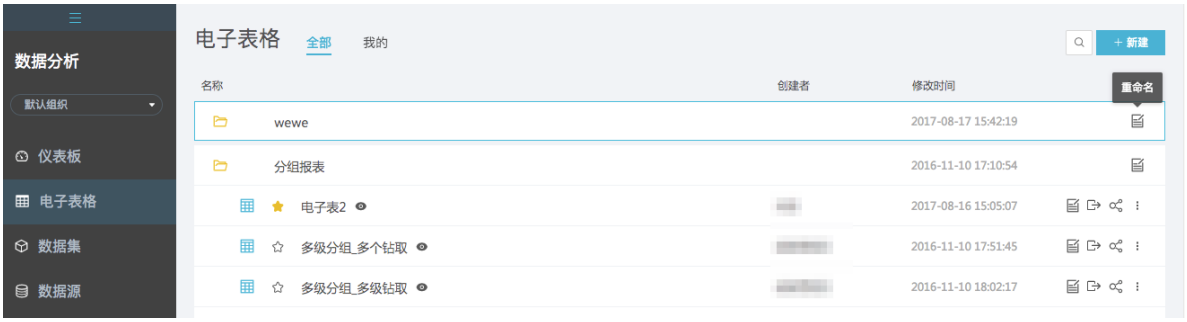
4. 输入一个文件夹名称，单击页面空白处即可。

9.5.13 重命名电子表格文件夹

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. 在列表中，选择一个电子表格。
4. 单击**重命名**图标，如图 *9-206:* 重命名文件夹所示。

图 9-206: 重命名文件夹



9.5.14 分享电子表格

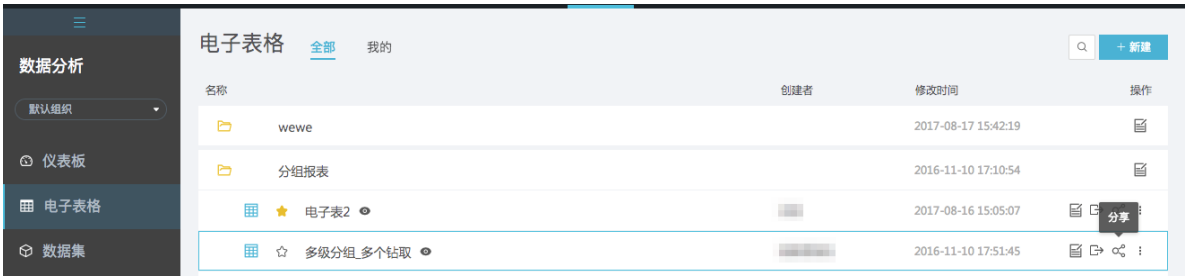
制作好的电子表格可以分享给他人。

操作步骤

1. 登录Quick BI控制台。
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 *9-207: 分享电子表格*所示。

图 9-207: 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。

9.5.15 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控制台。
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**，并单击**公开**。

9.6 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。

9.6.1 新建数据门户

背景信息

数据门户的操作只适用于工作空间下，个人空间不支持该功能。

操作步骤

1. 登录Quick BI控制台。

2. 在左侧导航栏中，单击**数据门户**，进入数据门户管理页面。
3. 单击**新建** > **数据门户**，如图 *9-208: 新建数据门户*所示。

图 9-208: 新建数据门户

9.6.2 设置模板

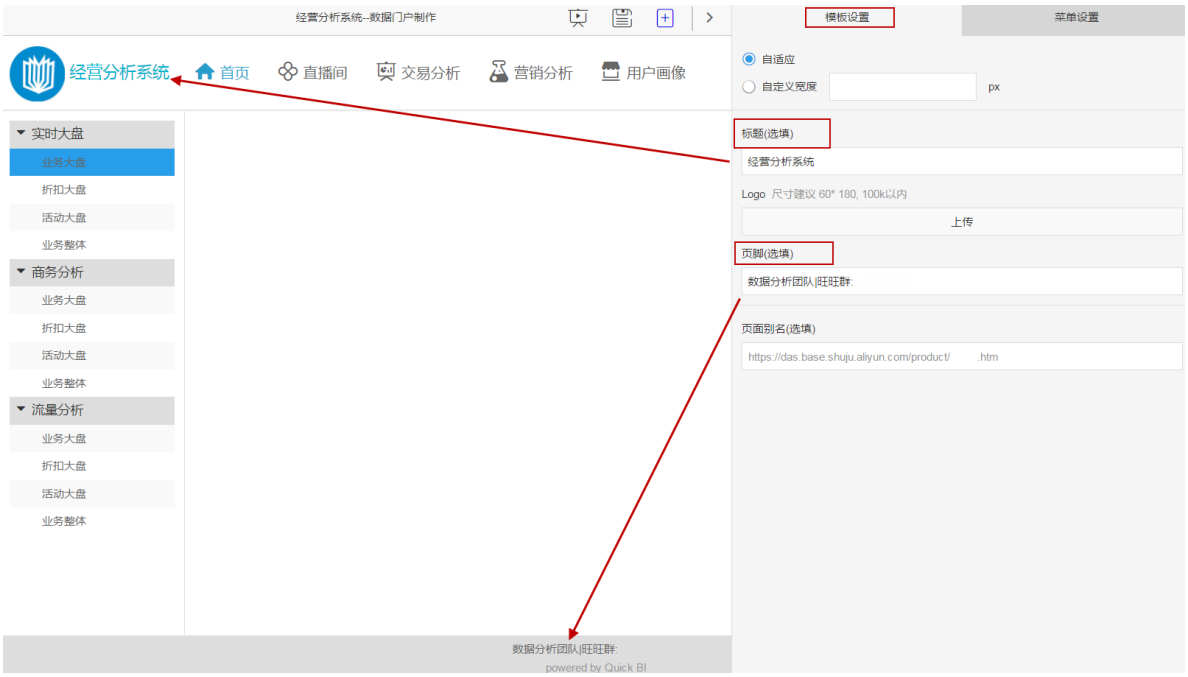
在模板设置的标签页中，可以设置模板的标题、上传logo以及编辑页脚。

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**，进入数据门户管理页面。
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 *9-209: 模板设置*所示。

图 9-209: 模板设置



9.6.3 设置菜单

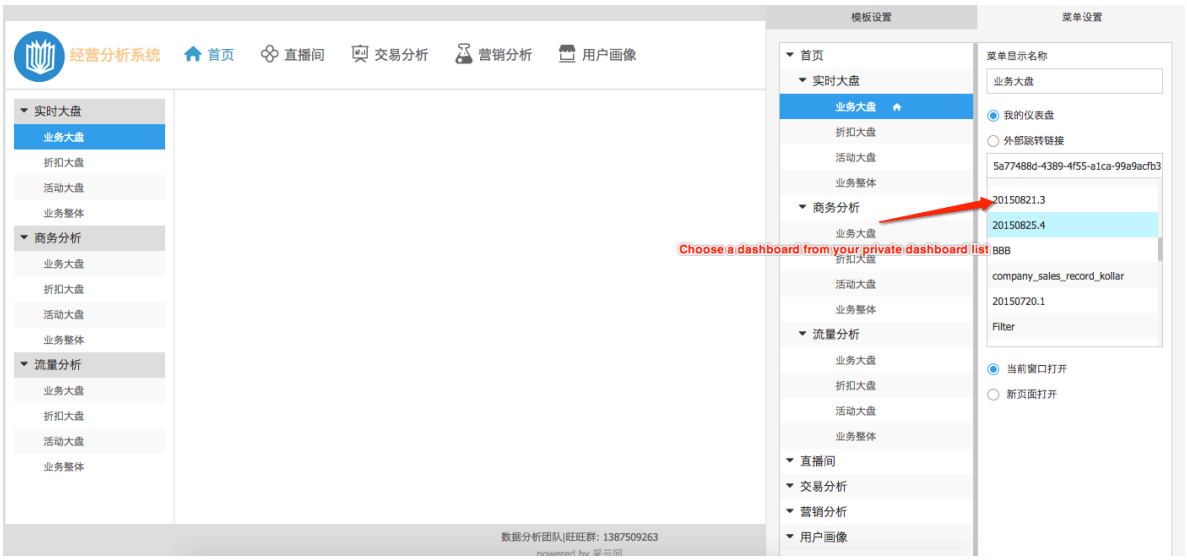
在菜单设置的标签页中，可以设置菜单名称、单击菜单要打开的URL等信息。

操作步骤

1. 登录Quick BI控制台。
2. 在左侧导航栏中，单击**数据门户**，进入数据门户管理页面。
3. 在数据门户列表页，单击一个数据门户名称。

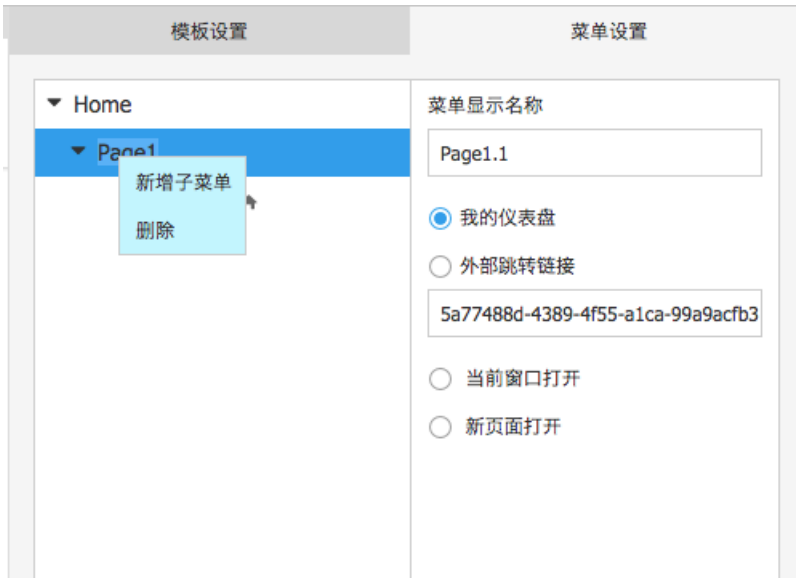
   新建数据门户，请参阅，新建数据门户。
4. 单击**菜单设置**标签页，编辑门户模板，如图 *9-210:* 菜单设置所示。

图 9-210: 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 *9-210:* 编辑菜单结构所示。

图 9-210: 编辑菜单结构



- 支持对仪表板和工作表的引用。

9.7 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。

9.7.1 创建组织

背景信息

一个账号最多只允许开通或加入一个组织。账号需要在大数据管家云账号管理中申请。

操作步骤

1. 在Quick BI控制台，单击**配置面板**，进入组织管理页面。
2. 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 *9-212:* 创建组织所示。

图 9-212: 创建组织

9.7.2 修改组织信息

背景信息

组织管理员可以对组织的属性进行再次编辑，如图 *9-213:* 修改组织信息所示。

图 9-213: 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。

工作空间的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工作空间中。工作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。

只有组织的管理员才有权限管理组织中的成员。组织的创建者默认就是组织管理员。

在一个组织中，组织成员可分为两种：管理员和普通成员。

操作步骤

1. 进入组织管理页面。
2. 选择**组织配置**标签页。
3. 手动修改组织信息，然后单击**修改组织信息**，完成信息修改。

9.7.3 退出组织

操作步骤

1. 登录组织管理页面。
2. 单击**退出组织**，可退出当前的组织，如图 *9-214:* 退出组织所示。

图 9-214: 退出组织

9.7.4 添加组织成员

背景信息

您可以通过添加阿里云账号和RAM子账号来逐一添加组织成员。

如果需要添加的成员较多，您也可以通过批量添加组织成员功能添加成员。

通过阿里云账号添加成员

操作步骤

1. 进入组织管理页面。
2. 单击**组织成员**标签页。
3. 单击**添加组织成员**。
4. 选择**添加阿里云账号**标签页。
5. 手动输入阿里云账号和昵称，并勾选是否将该成员设为组织管理员，如图 *9-215:* 添加组织成员所示。

图 9-215: 添加组织成员

6. 单击**确定**，完成组织成员添加。

通过RAM子账号添加组织成员

操作步骤

1. 进入组织管理页面。
2. 单击**组织成员**标签页。
3. 单击**添加组织成员**。
4. 选择**添加RAM子账号**标签页。
5. 手动输入RAM子账号和昵称，并勾选是否将该成员设为组织管理员，如图 *9-216: 添加组织成员*所示。

图 9-216: 添加组织成员

6. 单击**确定**，完成组织成员添加。

   如果该用户已经被其他组织吸纳，系统会给出添加成员失败的提示。

批量添加组织成员

操作步骤

1. 单击**批量添加组织成员**。
2. 单击**选择Excel文件**，从本地上传组织成员名单，如图 *9-217:* 批量添加组织成员所示。

图 9-217: 批量添加组织成员

3. 单击**确定**，完成批量组织成员添加。

9.7.5 编辑组织成员

背景信息

您可以设置成员在组织中的角色和成员的昵称，便于查找成员。

操作步骤

1. 进入组织管理页面。
2. 单击**组织成员**标签页。
3. 选择其中一个组织成员，单击后面的**编辑**。
4. 修改组织成员的信息，如图 *9-218:* 编辑组织成员所示。

图 9-218: 编辑组织成员

5. 单击**确定**，完成组织成员信息编辑。

9.7.6 移除组织成员

背景信息

只有管理员有权限移除组织成员。移除组织成员时，若组织成员仍在某工作空间中，需要先将该成员从工作空间中移除，否则系统会给出该成员无法移除的提示。

说明：
移除成员的操作不可撤销。一旦成员被移除后，需要将其重新添加才能回到组织中，请谨慎操作。

操作步骤

1. 进入组织管理页面。
2. 单击**组织成员**标签页。
3. 选择一个组织成员，单击后面的**移除**。
4. 单击**确定**，完成组织成员移除。

9.7.7 查看用户所在的工作空间

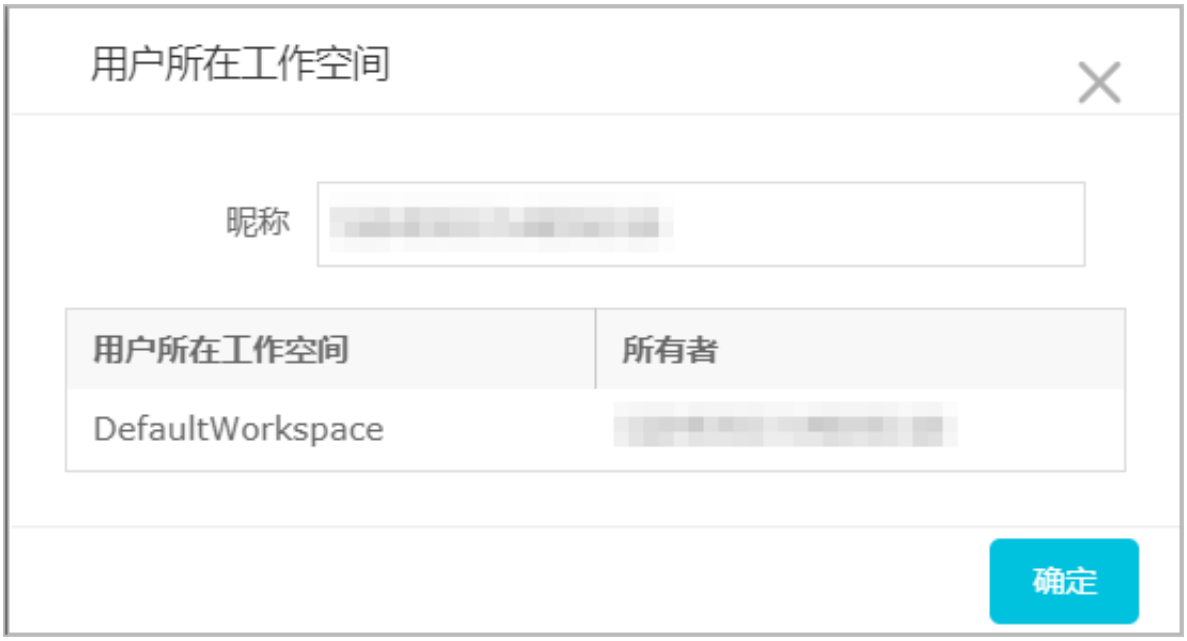
背景信息

您可以查看组织中用户所在的工作空间。

操作步骤

1. 进入组织管理页面。
2. 单击**组织成员**标签页。
3. 选择一个组织成员，单击后面的工作空间图标，如图 *9-219: 查看用户所在的工作空间*所示。

图 9-219: 查看用户所在的工作空间



4. 单击**确定**，关闭对话框。

9.7.8 查询组织成员

背景信息

您可以通过昵称或者账号查询组织成员。

操作步骤

1. 进入组织管理页面。
2. 单击**组织成员**标签页。
3. 输入成员昵称或者账号，单击**查询**图标，如图 *9-220: 查询组织成员*所示。

图 9-220: 查询组织成员

9.7.9 工作空间管理

工作空间的管理由空间的管理员来完成。工作空间管理员是由创建该空间的组织管理员来任命的。工作空间管理员也可以把空间中的其他成员设置为工作空间管理员。

工作空间管理包括以下工作。

- 新建工作空间
- 编辑工作空间
- 设置默认工作空间

9.7.9.1 什么是工作空间

工作空间是组织成员进行协同开发的工作空间，在工作空间中，空间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，电子表格、仪表板和数据门户的创建和修改。 这些数据对象会保存在其所在的工作空间中，不同的工作空间中有不同的数据对象。

工作空间有以下主要属性：

- 工作空间名称
- 工作空间说明
- 功能权限：电子表格可导出，默认允许，改为不允许后该工作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，该工作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，该工作空间中的作品不能被分享给空间外的人。
- 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。修改后影响该工作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。

您可以通过输入成员账号进行组织内的模糊匹配，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：工作空间管理员、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 9-3: 功能导航入口

权限点	开发者	分析师	阅览者
数据	有	无	无
电子表格	有	有	有
仪表板	有	有	有
数据门户	有	有	有

- 数据

表 9-4: 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
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
使用数据集	有	有	无

- 电子表格

表 9-5: 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
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 9-6: 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
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 9-7: 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
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

9.7.9.2 个人空间和工作空间的差异

用户工作的空间叫个人空间。 个人空间和工作空间的主要差异表现在以下几个方面。

- 个人空间在用户首次登录时自动创建，而工作空间需要组织管理员手动创建。
- 个人空间无法新建和删除。
- 个人空间不允许添加其他用户，也没有协作共享的功能。
- 个人空间中数据对象的分享范围为任何一个阿里云Quick BI用户，而工作空间中数据对象的分享范围是工作空间内部。

9.7.10 新建工作空间

操作步骤

1. 进入组织权限管理页面。
2. 选择**工作空间管理** > **新建工作空间**。
3. 在新弹出的页面中，输入一个工作空间名称，如图 *9-221:* 新建工作空间所示。

图 9-221: 新建工作空间

4. 单击**确定**，完成工作空间新建。

9.7.11 修改工作空间

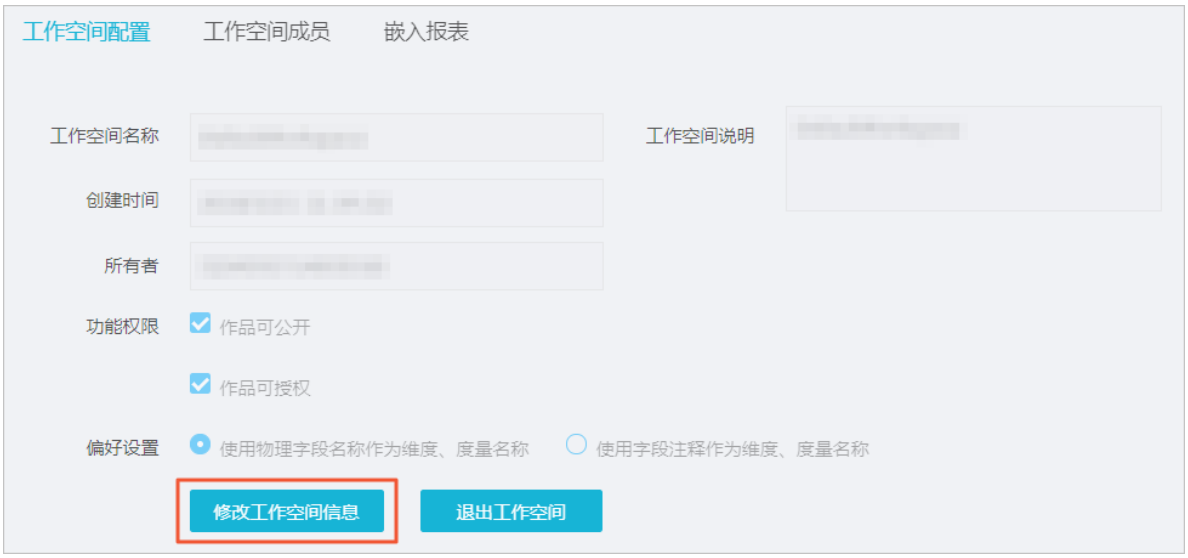
背景信息

个人空间只允许空间的所有者修改配置，而工作空间只有工作空间管理员才可以修改工作空间的配置。

操作步骤

1. 进入组织权限管理页面。
2. 选择**工作空间管理**，进入工作空间管理页面。
3. 单击**工作空间配置**标签页。
4. 单击**修改工作空间信息**，并手动修改空间信息。

图 9-222: 修改工作空间



5. 单击**确定**，完成工作空间的修改。

9.7.12 退出工作空间

操作步骤

1. 进入组织权限管理页面。
2. 选择**工作空间管理**标签页。
3. 选择一个工作空间，单击**工作空间配置**标签页。
4. 单击**退出工作空间**，可退出当前选择的工作空间，如图 *9-223:* 退出工作空间所示。

图 9-223: 退出工作空间

9.7.13 转让工作空间

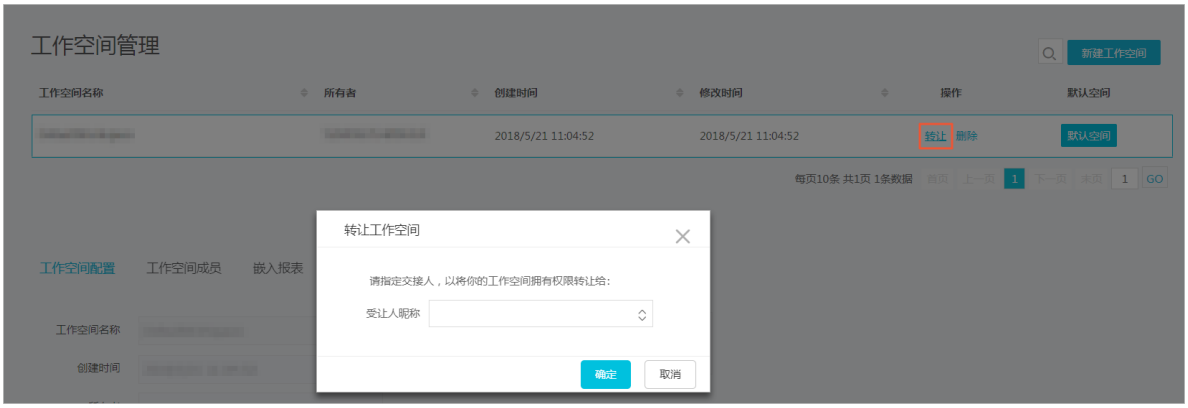
背景信息

如果一个工作空间的创建者想从组织中被移除，可以通过转让工作空间操作将拥有的工作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。

操作步骤

1. 进入组织权限管理页面。
2. 选择**工作空间管理**标签页。
3. 选择一个工作空间，单击**转让**。
4. 输入被转让人昵称，然后单击**确定**，完成工作空间转让，如图 *9-224: 转让工作空间*所示。

图 9-224: 转让工作空间

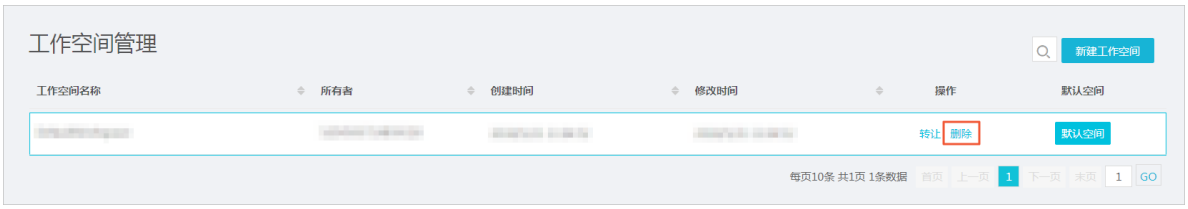


9.7.14 删除工作空间

操作步骤

1. 进入组织权限管理页面。
2. 选择**工作空间管理**标签页。
3. 选择一个工作空间，单击**删除**，如图 *9-225: 删除工作空间*所示。

图 9-225: 删除工作空间



9.7.15 添加工作空间成员

操作步骤

1. 进入组织权限管理页面。
2. 选择**工作空间成员**标签页。
3. 单击**添加工作空间成员**。
4. 输入成员的账号并分配角色，如图 *9-226:* 添加工作空间成员所示。

图 9-226: 添加工作空间成员

9.7.16 编辑工作空间成员

操作步骤

1. 进入组织权限管理页面。
2. 选择**工作空间成员**标签页。
3. 选中某一位成员，单击**编辑**。
4. 手动修改工作空间成员信息，然后单击**确定**，完成工作空间成员编辑。

9.7.17 删除工作空间成员

操作步骤

1. 进入组织权限管理页面。
2. 选择**工作空间成员**标签页。
3. 选中某一位成员，单击**删除**。

4. 选择一个受让人，将被删除者的作品转移到受让人名下。
5. 单击**确定**，完成工作空间成员删除。

9.7.18 查询工作空间成员

操作步骤

1. 在工作空间管理页面，选择工作空间。
2. 单击**工作空间成员**标签页，进入工作空间成员列表。
3. 在搜索框内，输入成员昵称或账号。
4. 单击查询图标，查询工作空间成员，如图 *9-227: 查询工作空间成员*所示。

图 9-227: 查询工作空间成员

9.8 权限管理

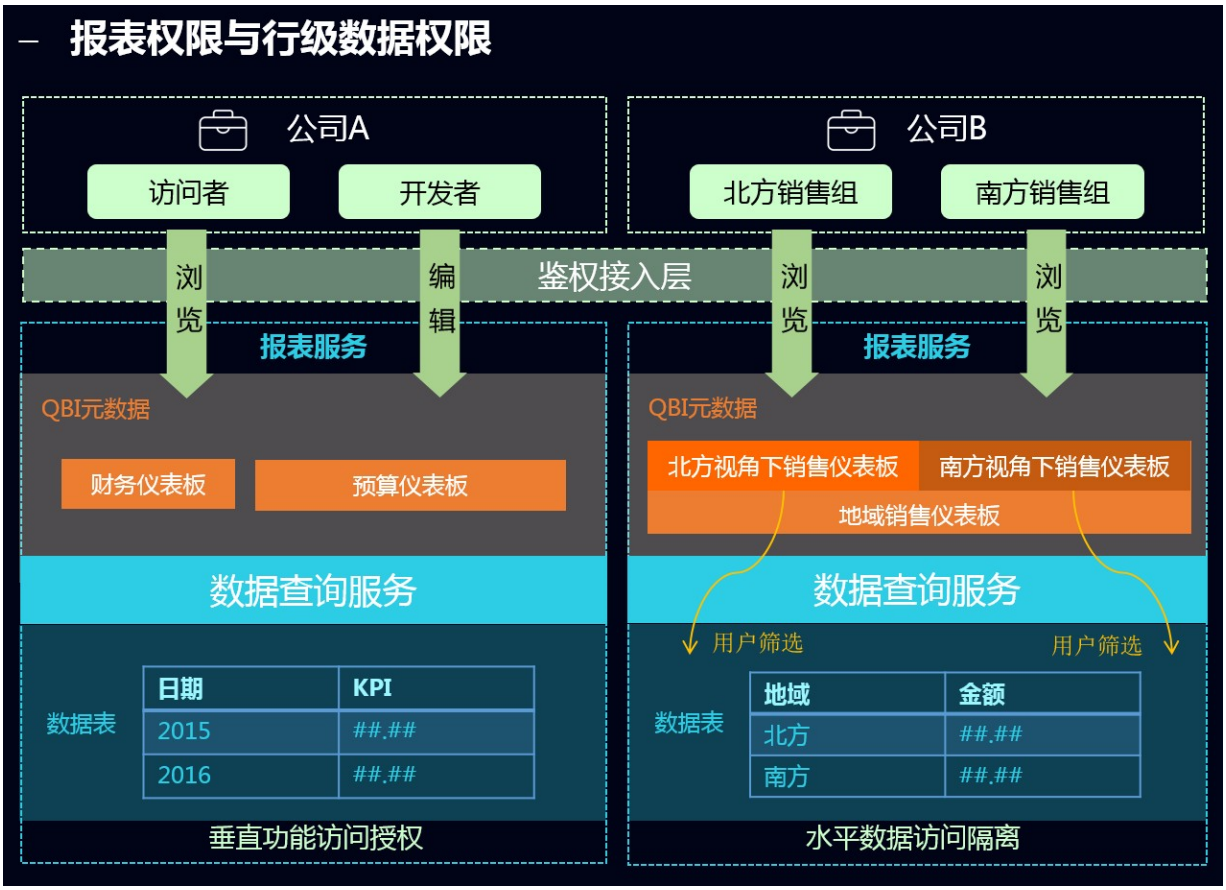
权限管理包括数据对象管理和行级权限管理。

数据对象包括数据源、数据集、电子表格、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工作空间下数据对象的管理。

9.8.1 数据对象管理

数据对象包括 数据源、数据集、电子表格、仪表板、数据门户等。

图 9-228: 数据对象管理



工作空间中数据对象的分享

支持电子表格、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。
- 如果工作空间属性中设置了不允许分享，则该工作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。

工作空间内的作品默认工作空间成员进入所在工作空间就可以看到并进行阅览。

工作空间内的作品也可以分享给组织内的个人，被分享人可以不在当前作品所在的工作空间。

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 9-229: 阅览被分享的作品所示。

图 9-229: 阅览被分享的作品

工作空间中的仪表板也支持公开，所有人都可以访问被公开的仪表板，所以不建议您将包含业务数据的仪表板公开。

9.8.2 行级权限

详细内容，请参阅，设置数据集行级权限。

9.8.3 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，其他人没有权限。

个人空间中数据对象的分享

支持电子表格、 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。

10 流计算StreamCompute

10.1 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，流计算具备以下三大类特点：

- 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。

10.2 快速开始

10.2.1 登录流计算控制台

本章节将为您介绍如何登录流计算控制台。

前提条件

管理员已为您在BCC控制台上创建云账号。

背景信息

您需要先从天基上获取流计算集群的域名，然后用获取后的域名登录流计算控制台。

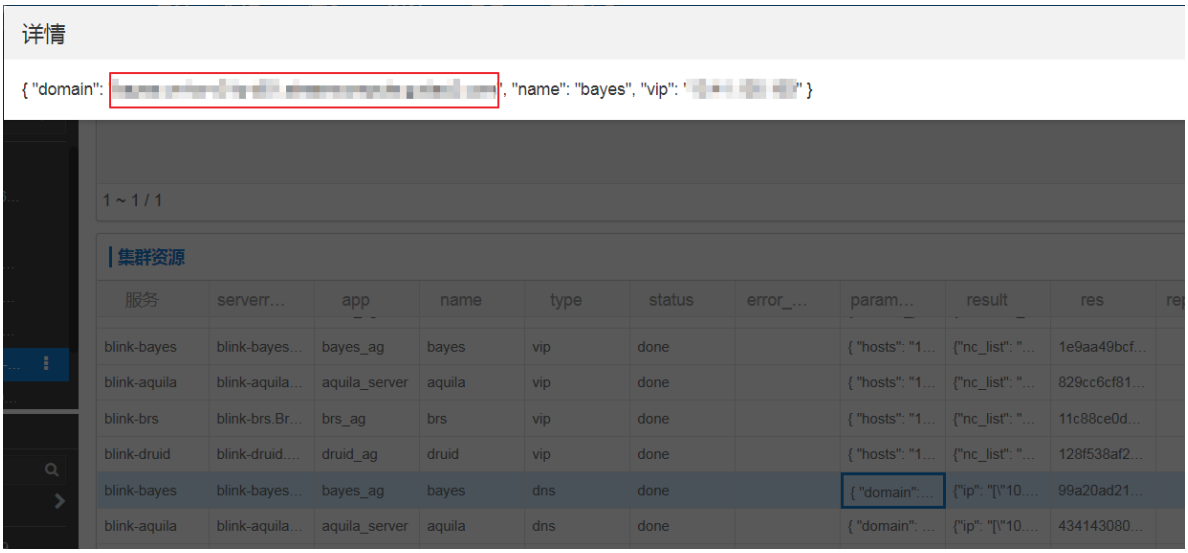
操作步骤

1. 登录天基。
2. 在上方导航栏中点击**运维** > **集群运维**，进入**集群运维**界面。
3. 在**Project**下拉菜单中选择**blink**查询流计算集群，点击查询到的目标流计算集群名称，进入**集群Dashboard**页面。

图 10-1: 查询集群

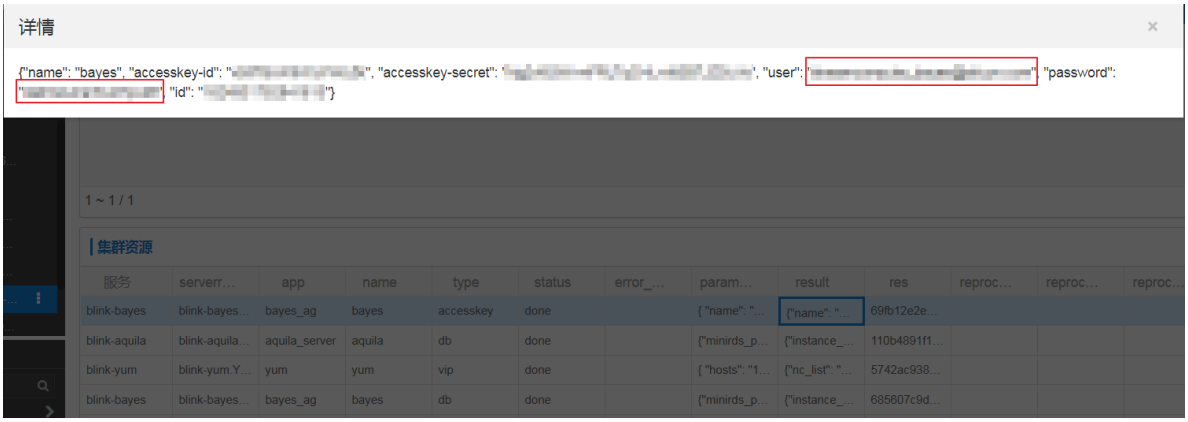
4. 往下拖动滚动条，在**集群资源**区域框中选中**app**为**bayes_ag**、**type**为**dns**的行，右键单击该行的**parameters**单元格，选择**显示更多**，弹出**详情**提示框，在提示框中查看该流计算集群的域名。

图 10-2: 查看域名



5. 在**集群资源**区域框中选中**app**为**bayes_ag**、**type**为**accesskey**的行，右键单击该行的**result**单元格，选择**显示更多**，弹出**详情**提示框，在提示框中查看云资源管控平台的用户名和密码。

图 10-3: 查看用户名和密码



6. 复制流计算集群域名到浏览器地址栏，回车后，进入云资源管控平台登录界面。
7. 选择**我是主账号**页签，输入流计算集群的云账号信息（即上述获取的用户名和密码），然后点击**登录**，进入流计算控制台页面。

10.2.2 安全实时监控案例

随着科技的进步，数据的安全越来越被世人所关注，安全实时监控报警尤为重要。要做到在流式的数据中进行实时监控报警，既需要保证数据的正确性、又要保证数据的实效性。阿里云流计算BlinkSQL团队顺利解决了这一难题，对流式数据的表和黑名单的表单进行维表的实时JOIN。以下案例就是对实时监控报警的完美诠释。

10.2.2.1 准备工作

背景信息

源表为简化问题，我们将源源不断的数据抽象简化为二维表datahub_IpPlace。

表 10-1: datahub_IpPlace

字段名	类型	注释
name	varchar	名字
Place	varchar	地址

dim表rds_dim

表 10-2: rds_dim

字段名	类型	注释
name	varchar	名字
Place	varchar	地址

JOIN的结果表rds_IpPlace:

表 10-3: rds_IpPlace

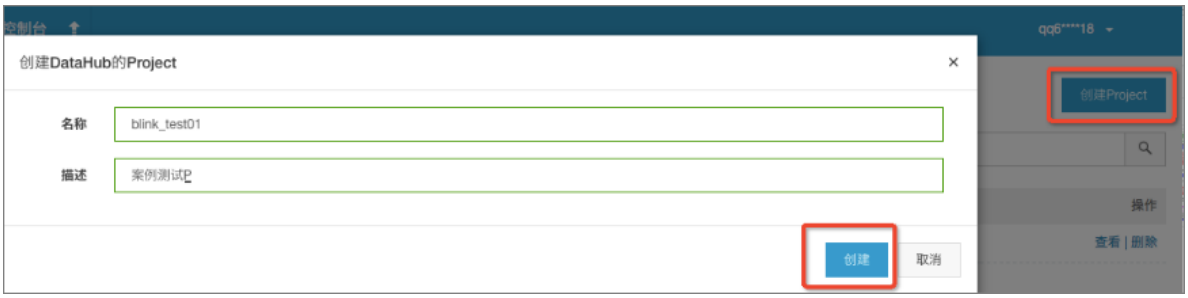
字段名	类型	注释
name	varchar	名字
Place	varchar	地址

创建数据源表和结果表

操作步骤

1. 登录DataHub创建项目。

图 10-4: 创建项目



2. 在该项目的详情页面中创建Topic，创建的源表schema示意图如下图所示。

图 10-5: Schema示意图

3. 在结果表所要归属的数据库实例后，点击**操作**列的图标，选择**查询详情**。

图 10-6: 云数据库管理

4. 在**基本信息**页面，点击**登录DMS**。

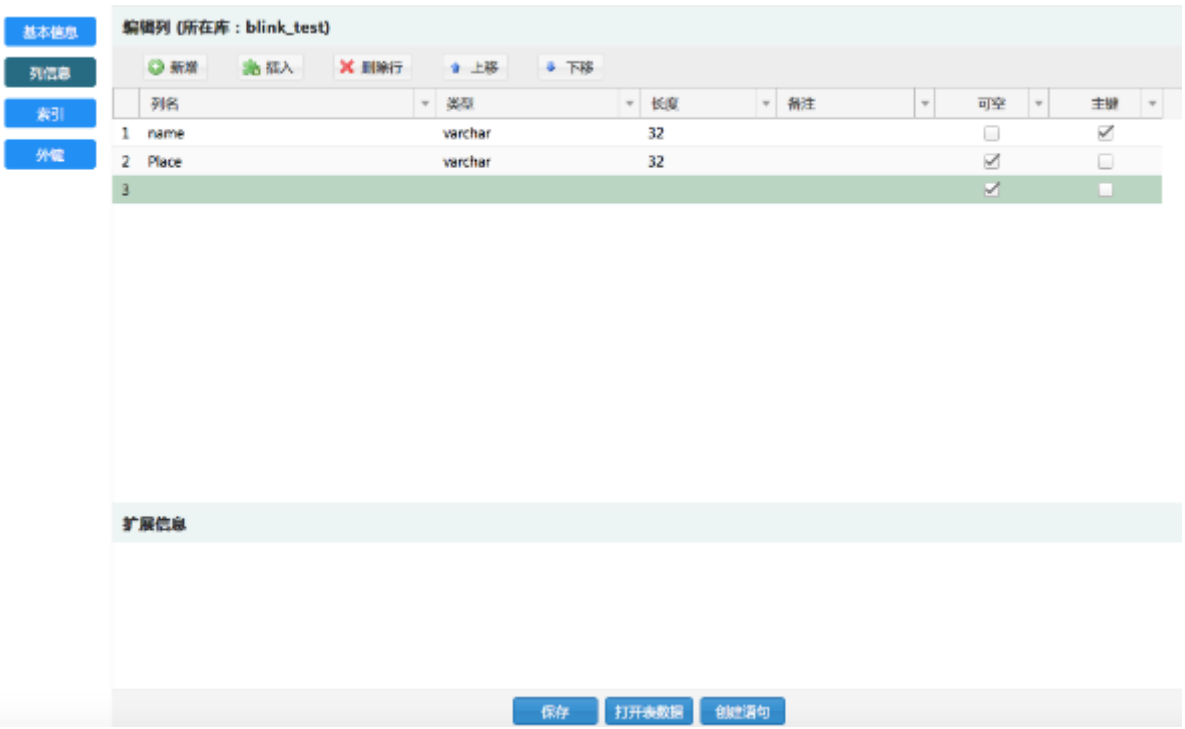
图 10-7: 基本信息页面

5. 登录数据库，然后在RDS中创建结果表。

图 10-8: 登录数据库

图 10-9: 维表schema示意图

图 10-10: 结果表schema示意图



6. 上传DataHub数据。

   进入**DataHub**界面，选择上述创建完成的DataHub Topic进行数据上传，单击导航栏的**数据采集 > 文件上传**。

图 10-11: 上传DataHub源表的数据

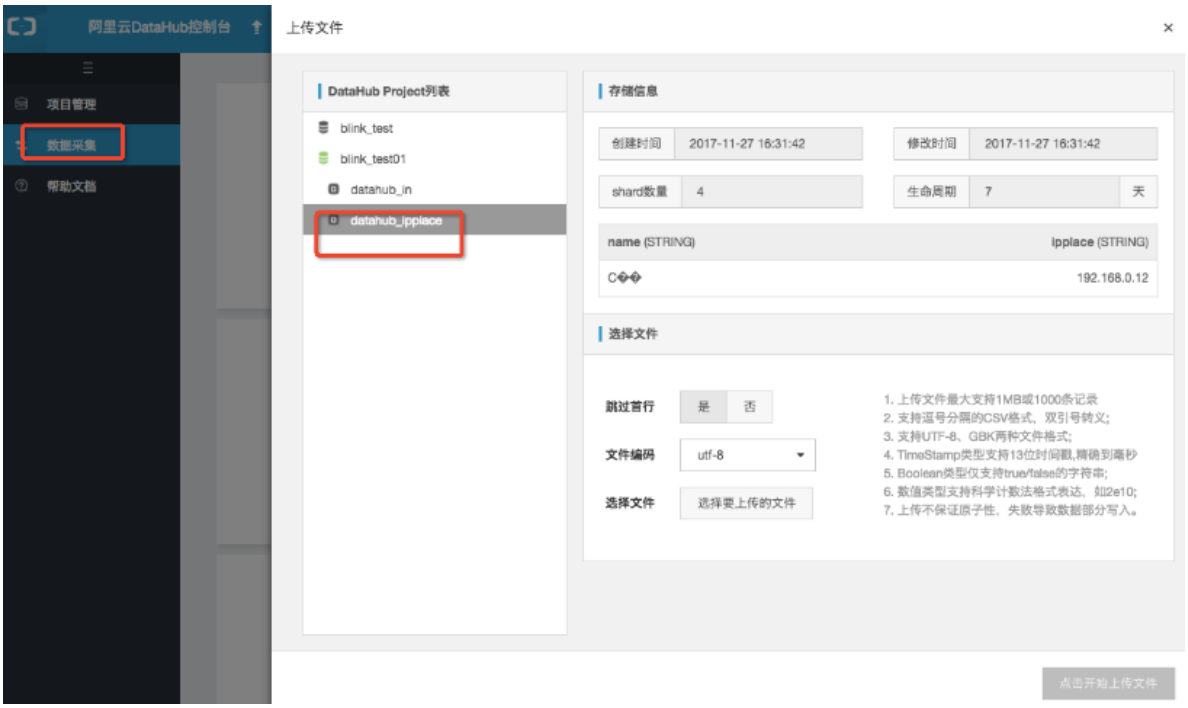


图 10-12: 上传RDS_DIM表的数据

10.2.2.2 开发

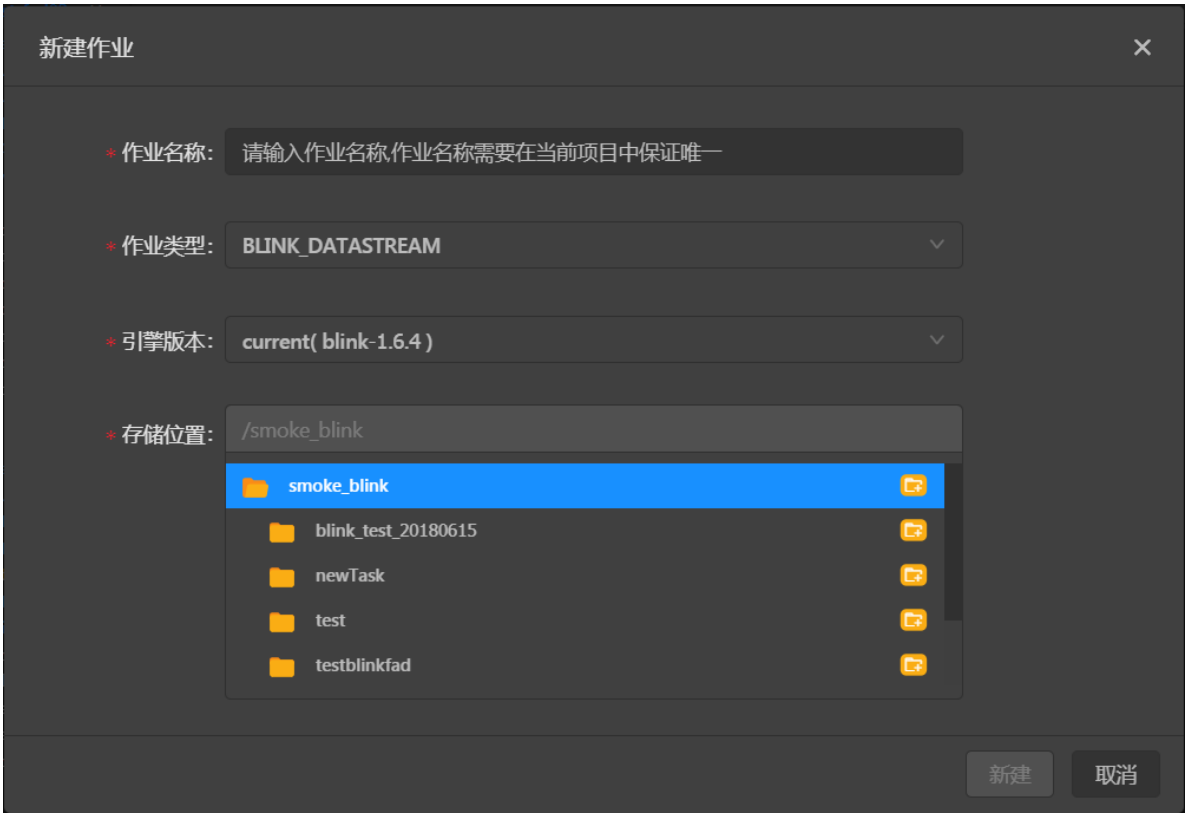
背景信息

一旦完成数据采集工作，您就可以专心研究BlinkSQL的编写工作。同样，为方便新手用户上手流计算，我们在**开发作业** > **smoke_blink** > **新手任务**中提供了商品排行榜的教学作业bj_dim_join 。单击这个作业，即可查看BlinkSQL情况。

操作步骤

1. 登录流计算控制台，进入阿里云流计算产品首页。
2. 点击头部导航栏**开发**页签，进入数据开发的 IDE页面。
3. 新建作业。

图 10-13: 新建作业



4. 以DataHub为数据的源表，使用数据存储自动生成DataHub的参数和您的schema信息。

图 10-14: 数据存储

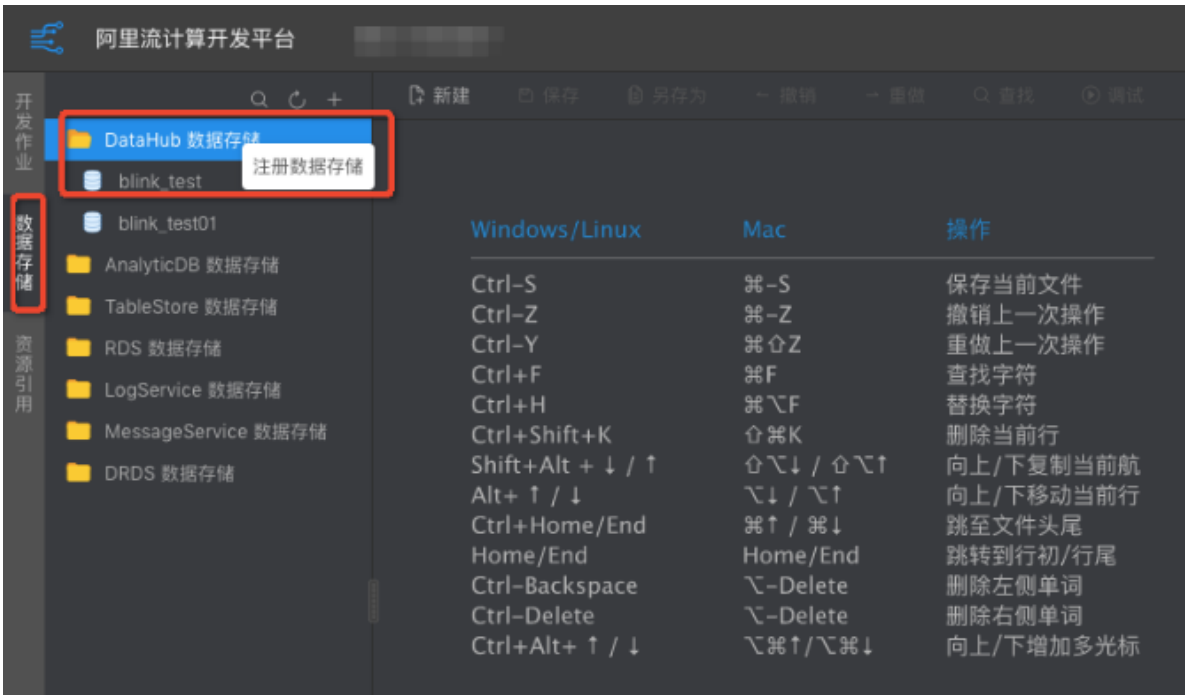


图 10-15: 注册数据存储

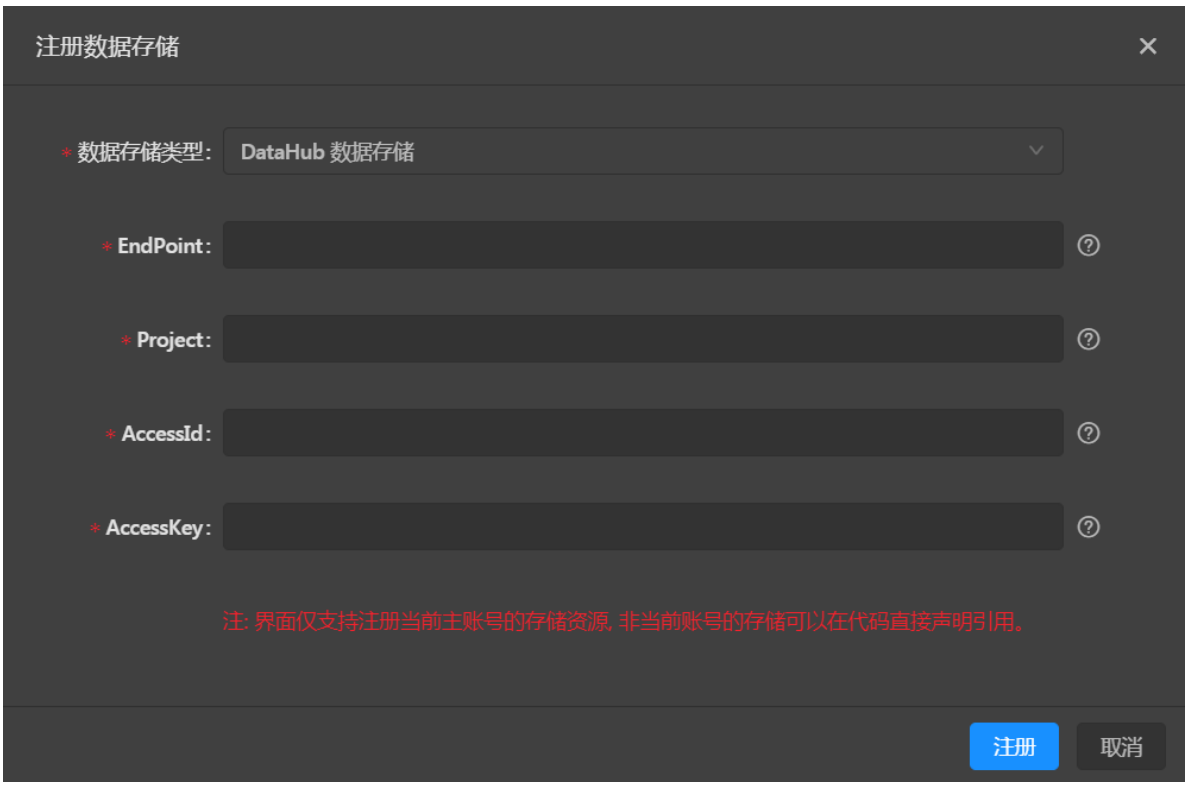
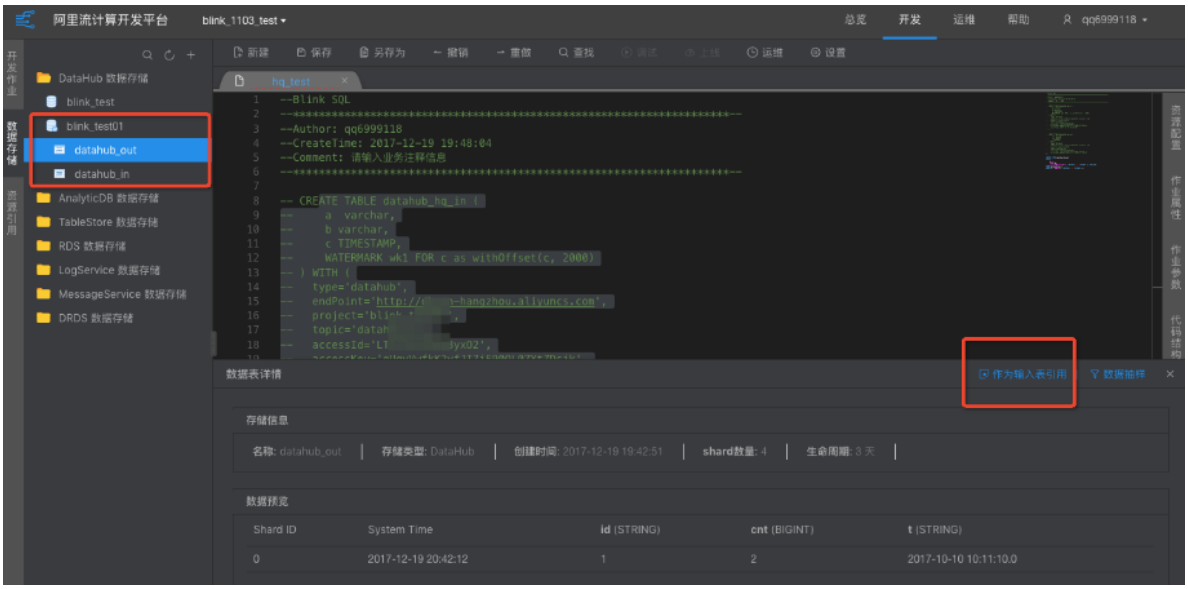


图 10-16: 作为输入表引入



5. 以RDS为数据的维表，手动填写RDS的参数和您的schema信息（维表在不支持数据存储的功能）。

图 10-17: 填写信息

```
create table rds_intput(
     Place varchar,
    `name`  varchar,
  primary key(Place),
  PERIOD FOR SYSTEM_TIME--这行声明定义了维表的变化周期，即表明该表是一张会变化的表
)with(
  type = 'rds',
  url='填写自己的URL',
  tableName='填写自己的表名',
  userName='填写自己的账号',
  password='填写自己的密码'
);
```

以RDS为数据的结果表，使用数据存储自动生成RDS的参数和您的schema信息。

图 10-18: 注册数据存储

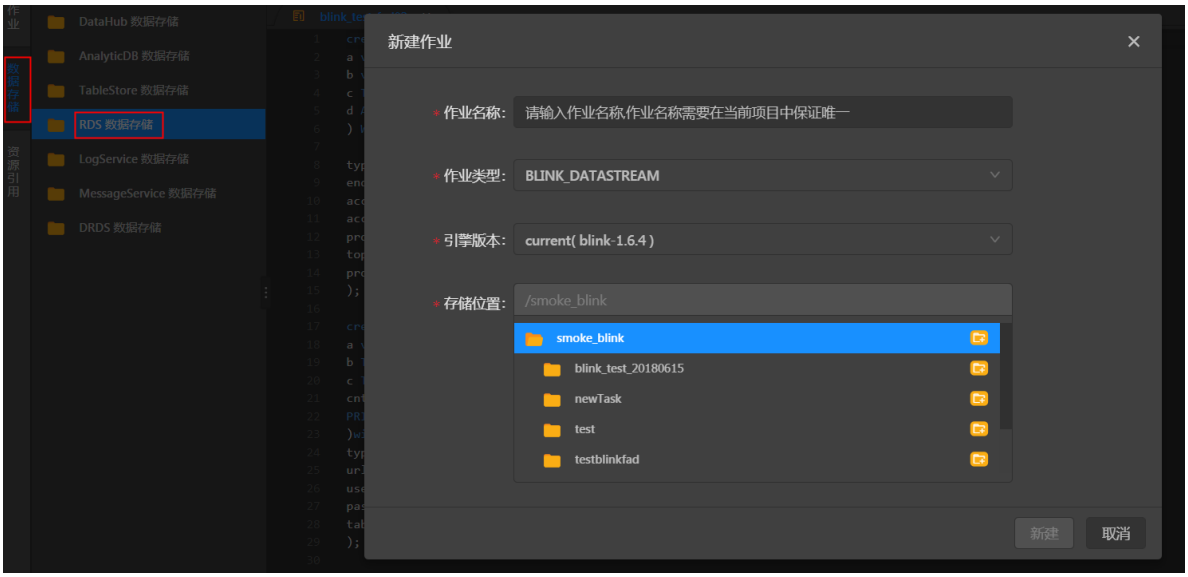
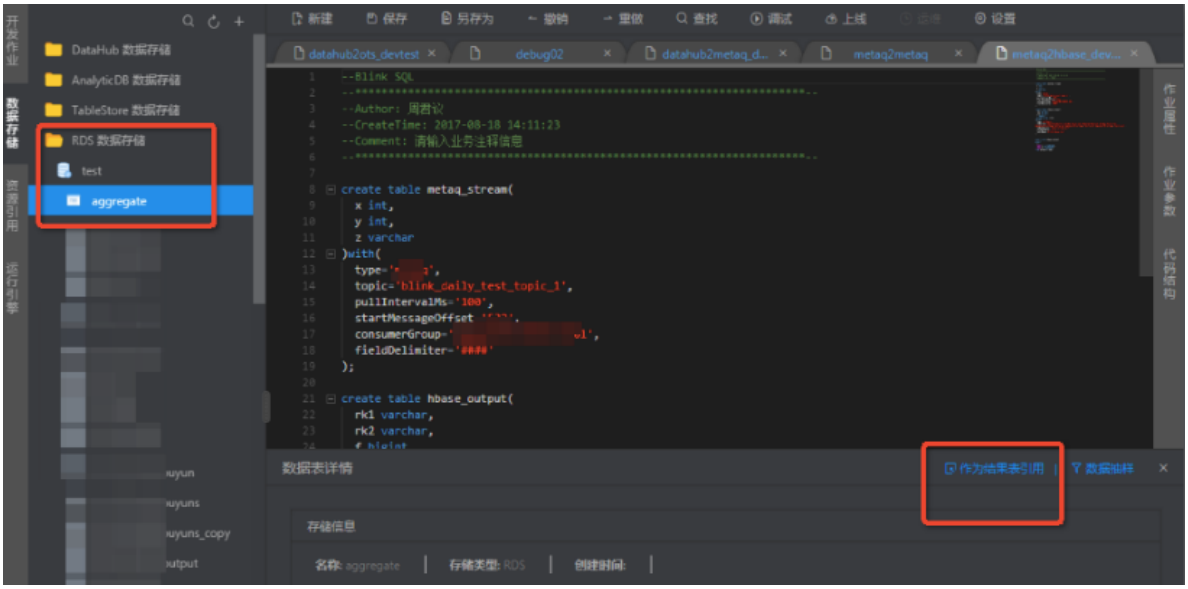


图 10-19: 作为结果表引用



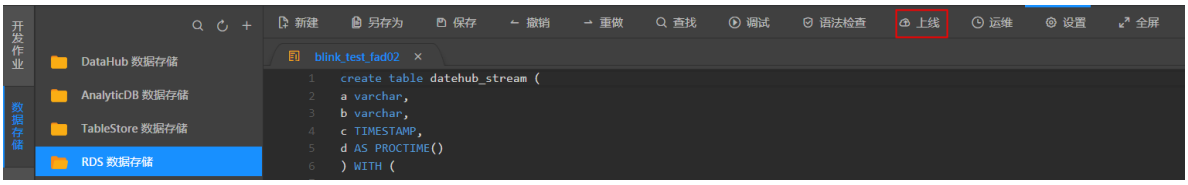
6. 编写业务逻辑的SQL。

```
INSERT INTO rds_output
SELECT
    t.`name`,
    w.Place
FROM datahub_input1 as t
JOIN rds_intput FOR SYSTEM_TIME AS OF PROCTIME() as w
ON t.Place = w.Place
```

7. 调试BlinkSQL。

8. 作业上线。调试完成后，经验证逻辑无误后，单击数据开发中的**上线作业**，即可完成作业上线工作。上线作业操作将当前用户的改动提交到数据运维中，即可在生产环境下进行作业启动等生产运维工作。

图 10-20: 作业上线



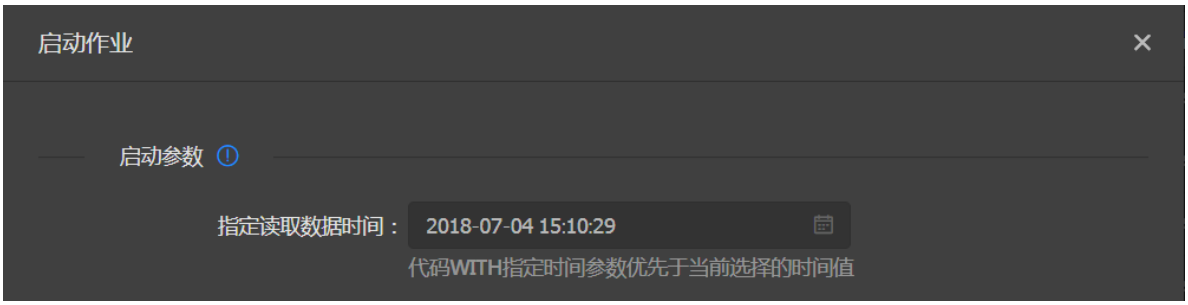
10.2.2.3 运维

操作步骤

1. 选中数据运维页面的bj_dim_join作业，单击**启动**。
2. 选择启动位点，由于刚传入数据，可以把启动时间设置的往前一点。

   指定读取数据时间是指从源头数据存储的指定时间点开始读取数据。

图 10-21: 启动作业



> **说明**：
> **透明升级**部分请忽略。

3. 作业运行中。

图 10-22: 运行拓扑图

4. 运行起来后，进入RDS数据库查看最终数据输出，和最终流式计算结果一致。这样即验证了业务代码的正确性。

图 10-23: 查看RDS数据

10.2.3 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？

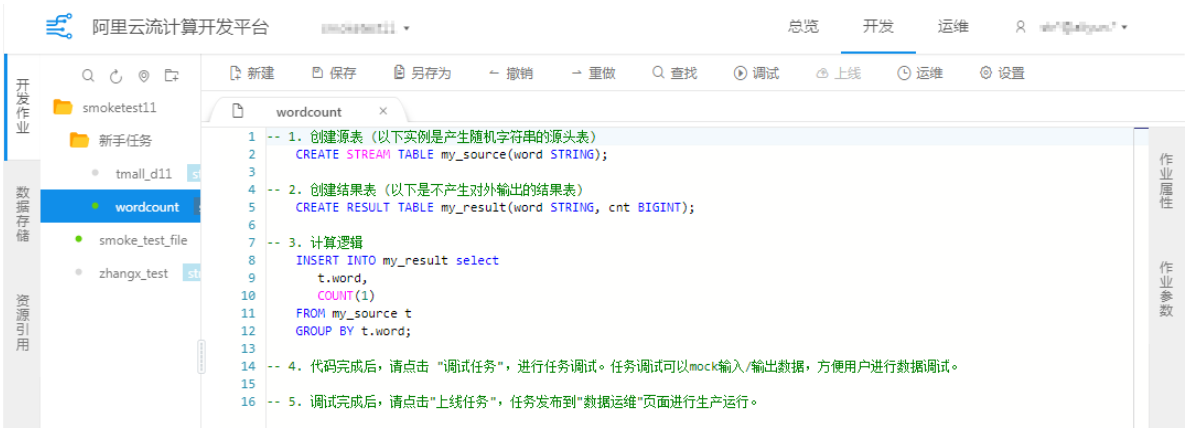
A：大数据的 WordCount 任务好比编程教学中的 “Hello World” ，通常均作为新手用户必不可少的入门任务。下面就以阿里云流计算的 WordCount 为例，讲解如何开发一个流式版本的 WordCount。新手用户通过 WordCount 任务，可以学习基本的 BlinkSQL 语法格式，以及任务编写/发布基本操作。

10.2.3.1 代码开发

1. 登录流计算控制台，进入阿里云流计算产品首页。
2. 点击头部导航栏**开发**页签，进入数据开发的 IDE页面。
3. 在阿里云流计算内置的名称为**新手任务**文件夹下，新建**wordcount**作业，代码如下。

```
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;
```

图 10-24: wordcount作业



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：

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。

10.2.3.2 代码调试

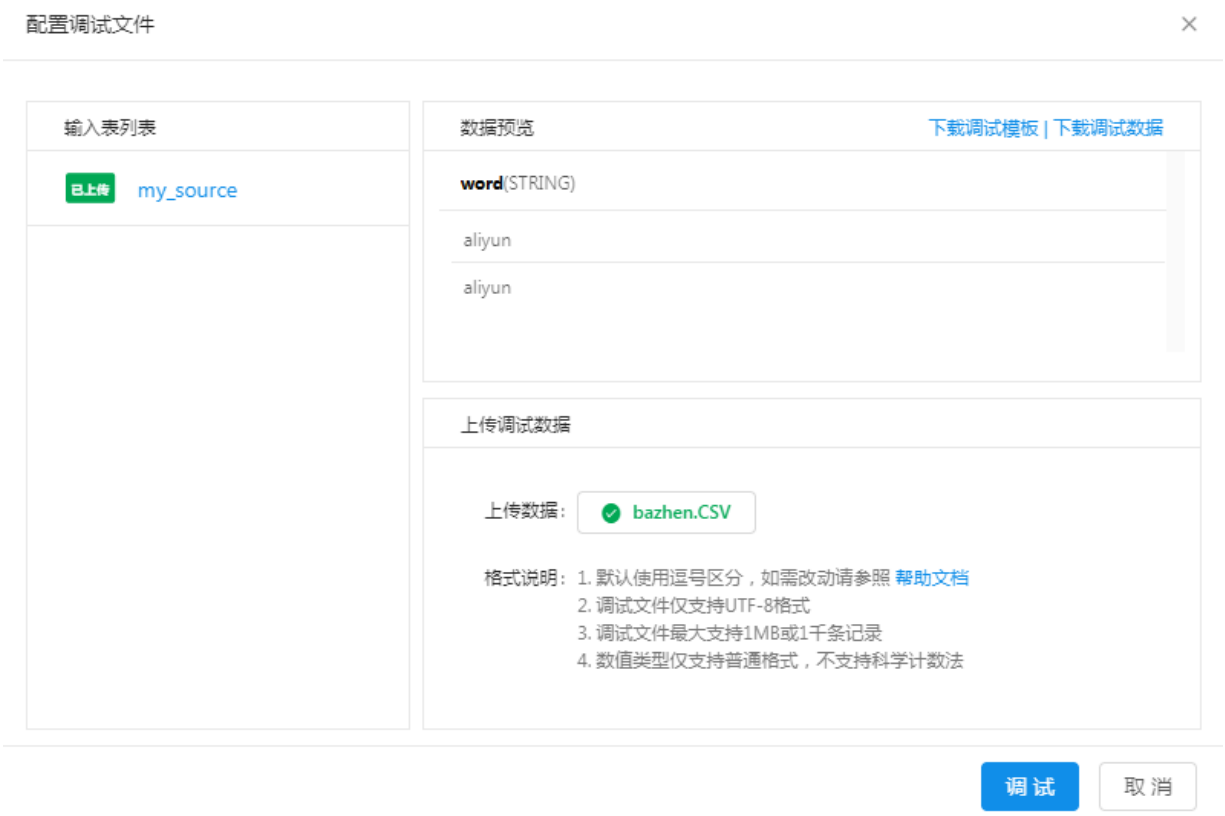
为方便用户调试 BlinkSQL，阿里云流计算提供了在线调试/Debug 功能，方便用户构造调试数据并方便的进行回归测试。阿里云流计算提供的调试功能非常强大，可以将流式存储、静态存储、结果存储全部进行模拟调试，方便用户构造各类数据进行 SQL 正确性验证。

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 10-25: 上传调试数据

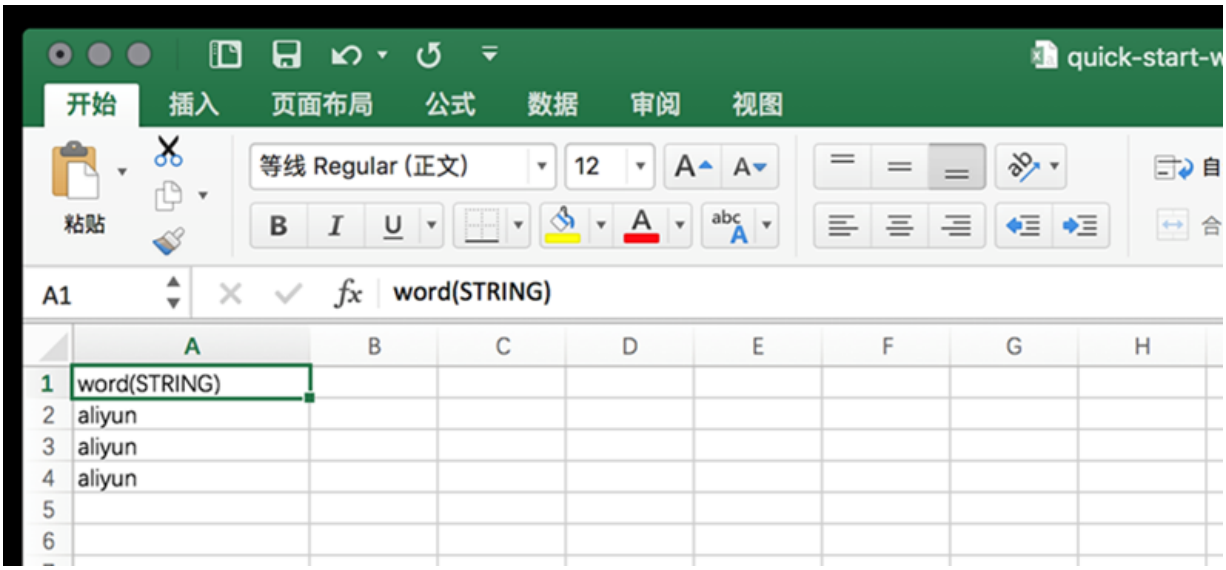


上图左侧是当前 BlinkSQL 用户声明的数据表引用清单，包括流式输入表、静态维表的清单，用户需要为每个输入表提供调试数据方能够进行调试运行。为了方便用户进行数据调试，阿里云流计算针对每个表自动构建了数据模板，并在自动构建的模板文件第一行附带提供了字段名、字段类型的Title，格式为**字段名**（字段类型）。用户可以根据Title的提示信息进行修改自己需要的测试数据。在调试页面用户点击**下载调试模板**并根据自己的测试策略填写调试数据。流计算对于上传的调试数据有较严格的定义：

- 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 10-26: 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。

10.2.3.3 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。

预期结果

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 *10-27:* 任务启动所示。

图 10-27: 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，为何这个计算任务既没有流式数据输入，也没有数据输出，那整体是否还能运行起来。原因在于：我们在定义上述`my_source`、`my_result`表时并未指定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。

10.3 操作指导

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、数据开发等方面指导用户如何在云上进行流式数据实时化分析。

10.3.1 管理项目

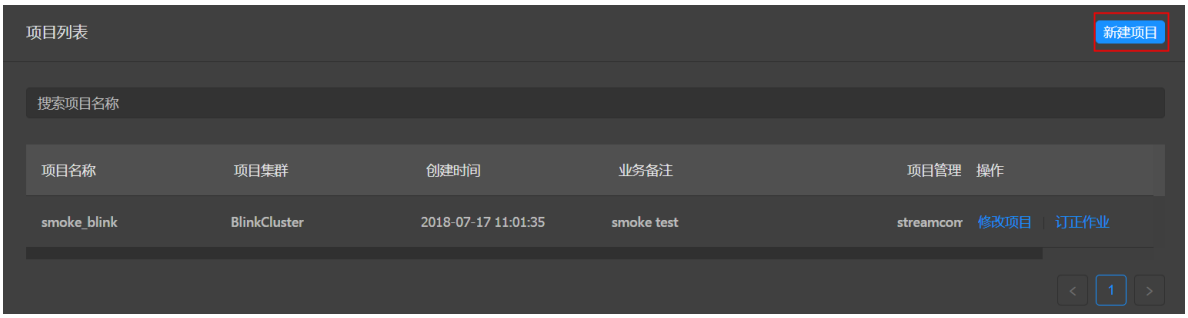
本节介绍如何创建和搜索项目。

创建项目

为了便于用户对开发项目进行管理，您可以通过登录admin账号创建新的项目。

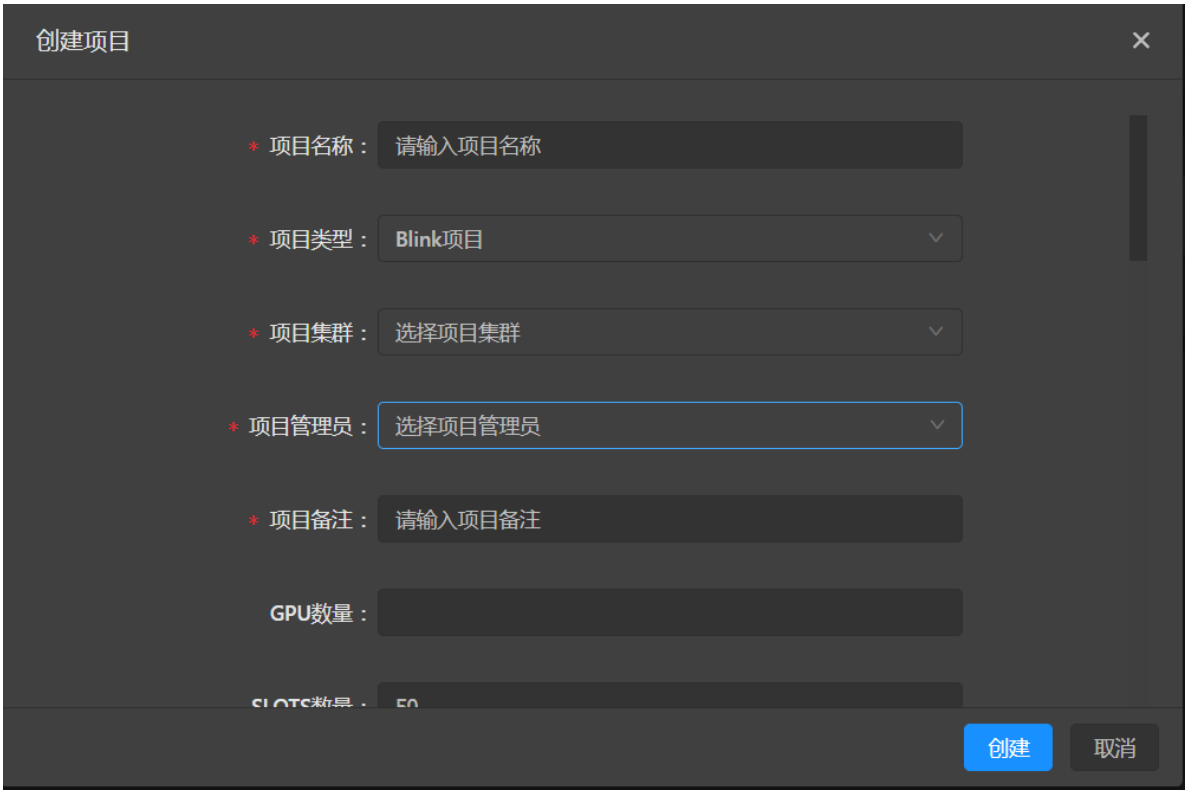
1. 在浏览器地址栏中输入`https://xxxx/#/admin`，其中*xxxx*为当前流计算登录账号登录状态下浏览器地址栏中#前面的内容。
2. 点击页面右上角的用户名称，在菜单中选择**项目管理**，进入**项目管理**页面。
3. 在左侧导航栏中依次选择**管理** > **项目管理**，进入**项目列表**页面。
4. 点击右上角的**创建项目**按钮。

图 10-28: 创建项目



5. 配置项目。

图 10-29: 配置项目



- 项目名称：输入新创建的项目名称。
- 项目类型：专有云默认选择**Blink项目**。
- 项目集群：整个项目中的作业要运行在的集群。
- 项目管理员：只有项目管理员才能运行这个项目中的作业。
- 项目备注：为该项目添加说明信息。
- GPU数量：指定该项目要占用集群的GPU的数量。

- SLOTS数量：指定该项目要占用的计算单元（CU）的数量，1CU包括1核CPU加4G内存。
- 支持报警方式：当作业运行出现异常时，可以通过设置的报警方式得到报警信息，包括短信告警和旺旺告警。
- 支持作业类型：保持默认选项即可。
- 支持存储类型：保持默认选项即可。
- 最大数据存储：可添加的数据存储数量。保持默认选项即可。
- 最大文件版本数：作业代码文件的版本保存数量。保持默认选项即可。
- 最大文件夹数：该项目中可创建的文件夹的数量。保持默认选项即可。
- 最大文件层级：该项目中可创建的文件夹的层级。保持默认选项即可。
- 最大文件数：该项目中可创建的作业代码文件的数量。保持默认选项即可。
- 最大资源数：用户可上传的JAR包和DICTIONARY资源的最大数量。保持默认选项即可。
- 最大资源引用数：用户可引用的JAR包和DICTIONARY资源的最大数量。保持默认选项即可。
- 监控报警：对作业是否启用监控报警功能。保持默认即可。
- 数据收集：对作业运行期间的数据是否进行收集。保持默认即可。
- 数据展示：保持默认即可。
- 数据存储：是否开启注册数据存储功能。默认开启，保持默认即可。
- 数据引擎：保持默认即可。
- 线上日志：是否记录作业运行的日志。默认开启，保持默认即可。
- 资源管理：是否可以上传JAR包等资源。默认开启，保持默认即可。

6. 点击**创建**完成项目配置。

搜索项目

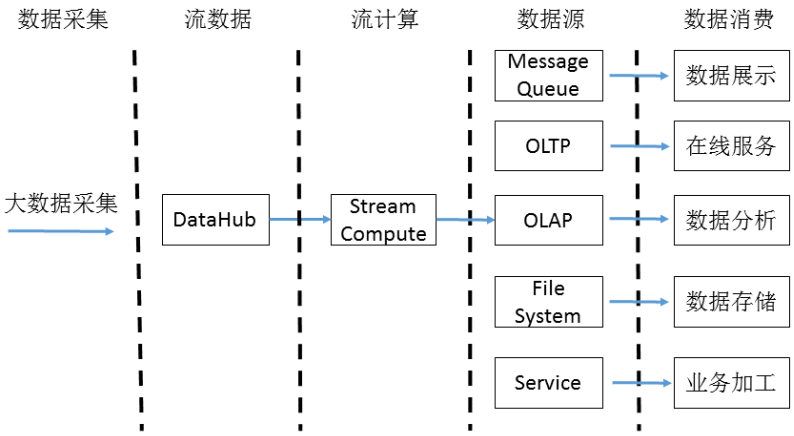
您可以在上方搜索栏中，输入项目名称的关键字或者全称来快速定位指定的项目。

10.3.2 数据采集

图 10-30: 数据采集

常言道巧妇难为无米之炊，所有的大数据分析系统都基于数据需要采集进入大数据系统的前提。为最大化利用您现有的流式存储系统，阿里云流计算对接了DataHub流式存储，让您不用进行数据采集、数据集成，即可享受现有的数据流式存储。

图 10-31: 系统架构



对DataHub流式存储，需要您使用DataHub数据集成工具进行数据采集。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。

10.3.3 数据存储

10.3.3.1 存储概览

为方便您管理数据存储，通过提前注册数据存储，您能够享受到更多一站式流计算开发平台提供的便利性。

说明：
数据存储注册需要提前授权，请参见《角色授权》。

10.3.3.1.1 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。

表 10-4: 流式存储

支持情况	输入	输出
DataHub	支持	支持

10.3.3.1.2 存储使用

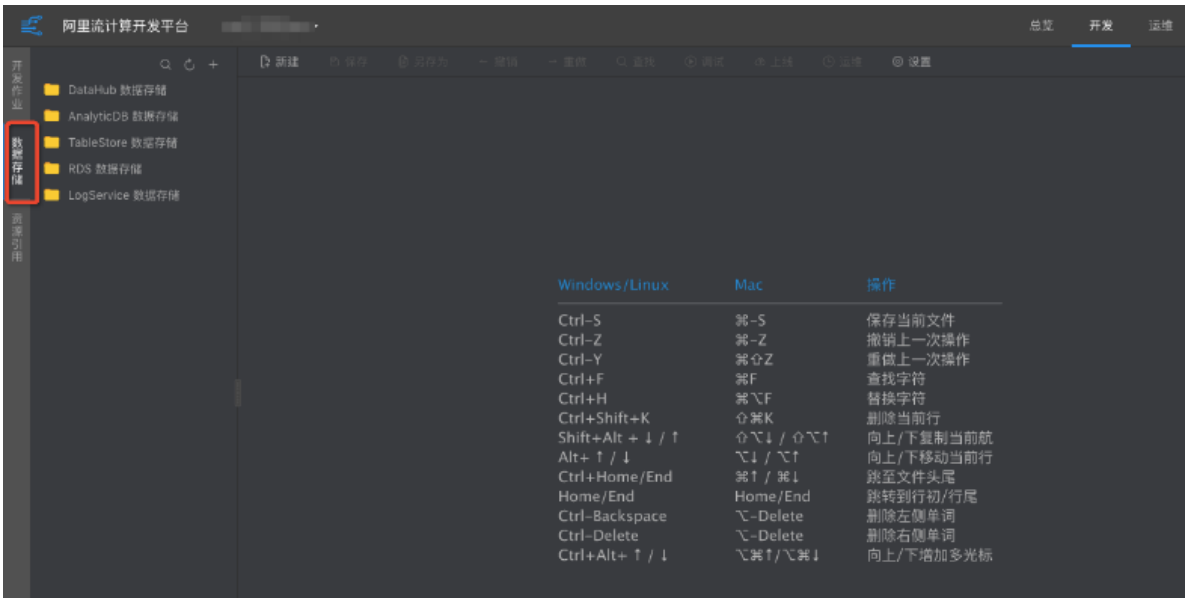
注册的数据源将为以下使用场景提供服务。

说明：
对于不属于当前用户的资源使用，您可以直接用AccessId/AccessKey在DDL定义语句中直接写出，此时您无法界面化操作数据源，但是作业可以直接运行。

- **数据注册**

  您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，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**。进入**开发**界面，单击左侧导航栏的**数据存储** > **+**，即可进入图 *10-32:* 数据存储注册页面。

图 10-32: 数据存储注册页面



说明：

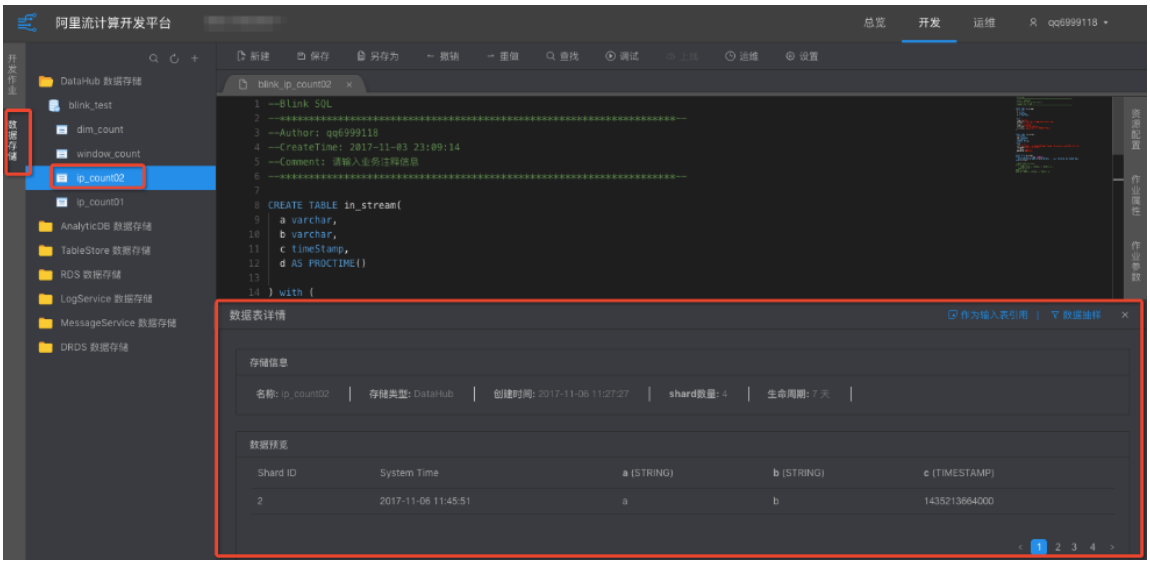
流计算数据存储功能当前仅支持同个一级部门账号属主下的存储资源，不支持跨一级部门账号授权。

- **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，单击**数据存储**，选择某个数据存储类型，即可预览数据。本节以DataHub为例为您介绍数据预览功能。

  1. 导航至**数据存储** > **DataHub存储**。
  2. 选择具体的Project和需要预览的Topic，双击即可进入查看数据存储。

图 10-33: 数据表详情



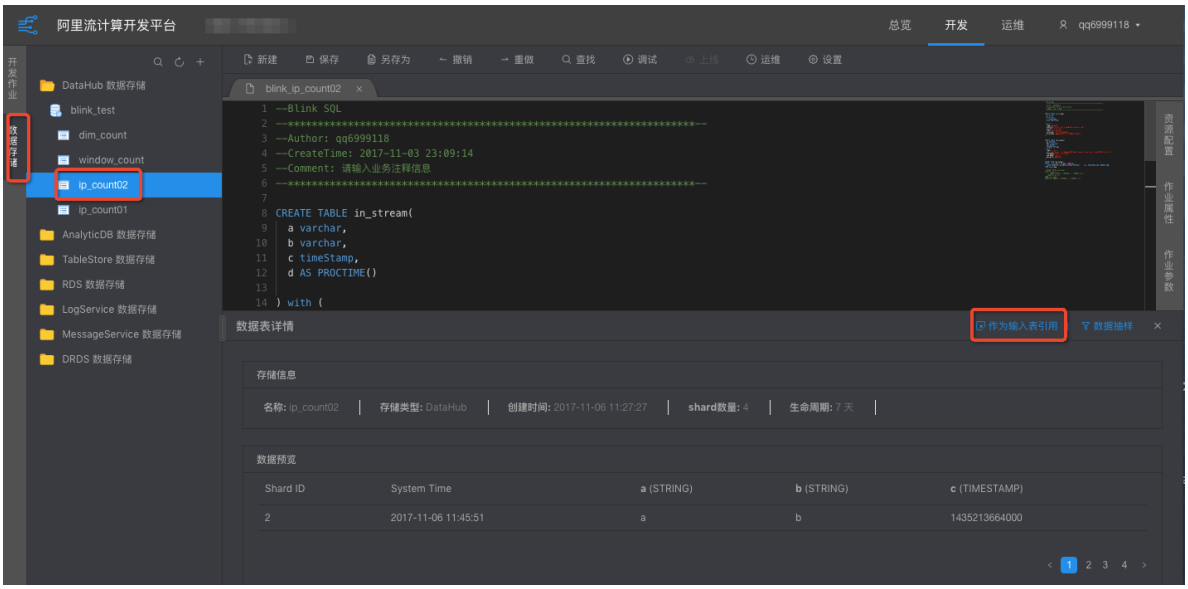
- **自动生成DDL**

  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 TABLE in_stream(
  a varchar,
  b varchar,
  c timeStamp,
) with (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyuncs.com',
  project='blink_test',
  topic='ip_count02',
  accessId='LTAIYtafPsXXXX',
  accessKey='gUqyVwfkK2vfJI7jF90QXXXXX'
);
```

  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，帮助您一键生成建表DDL语句。

  进入**开发**页面，点击需要编辑的作业，然后点击**数据存储**页签，选择输入表，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：跨一级部门账号的资源引用

当前流计算界面不支持跨一级部门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个一级部门账号属主下的存储资源，不支持跨一级部门账号授权。如果您需要跨一级部门账号使用，可以考虑直接在DDL语句中手写外部数据引用。例如A部门用户需要使用B部门用户的资源，那么可以在DDL定义如下内容。

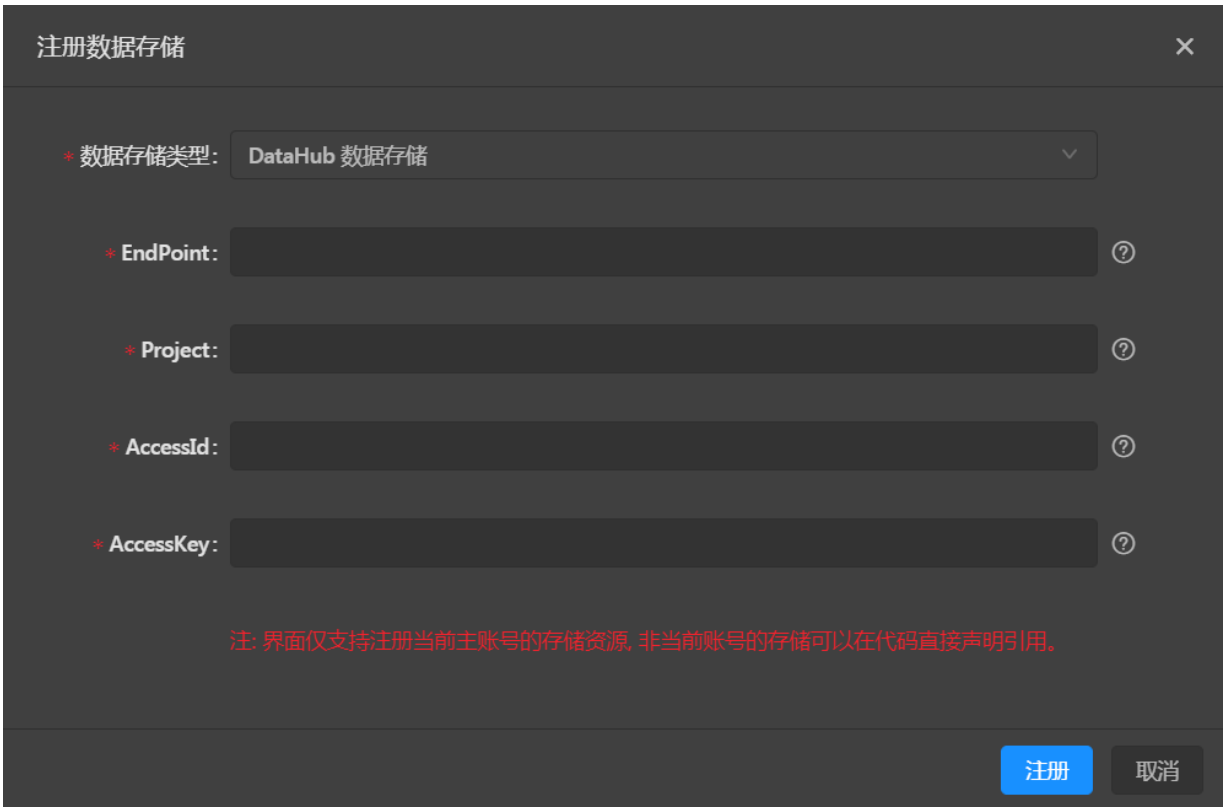
```
CREATE TABLE in_stream(
  a varchar,
  b varchar,
  c timeStamp,
) with (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyuncs.com',
  project='blink_test',
  topic='ip_count02',
  accessId='B用户授权的AccessId',
  accessKey='B用户授权的AccessKey'
);
```

10.3.3.2 大数据总线(DataHub)

注册

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。

图 10-34: DataHub数据存储



- **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，请联系您的管理员。

  > **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。

  > **说明**：
  > 跨一级部门属主的数据存储不能注册。例如A部门用户拥有DataHub的ProjectA，但B部门用户希望在流计算使用ProjectA，目前流计算暂不支持这类使用情况。

- **AccessId**

  填写当前账户的AccessKey ID。

- **AccessKey**

  填写当前账户的AccessKey Secret，以便流计算可以访问DataHub的Project。

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**

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DataHub的Project。请登录DataHub控制台，您可以访问DataHub控制台看您是否有权限访问您的Project。
- 请确认您是DataHub Project的属主， 特别注意，跨一级部门属主的数据存储不能注册。例如A部门用户拥有DataHub的ProjectA，但B部门用户希望在流计算使用ProjectA，目前流计算暂不支持这类使用情况。
- 请确认您填写的DataHub的Endpoint和Project完全正确。
- 请确认您填写的DataHub Endpoint是经典网络地址，而非VPC地址。目前流计算暂不支持VPC内部地址。
- 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：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。

10.3.4 数据开发

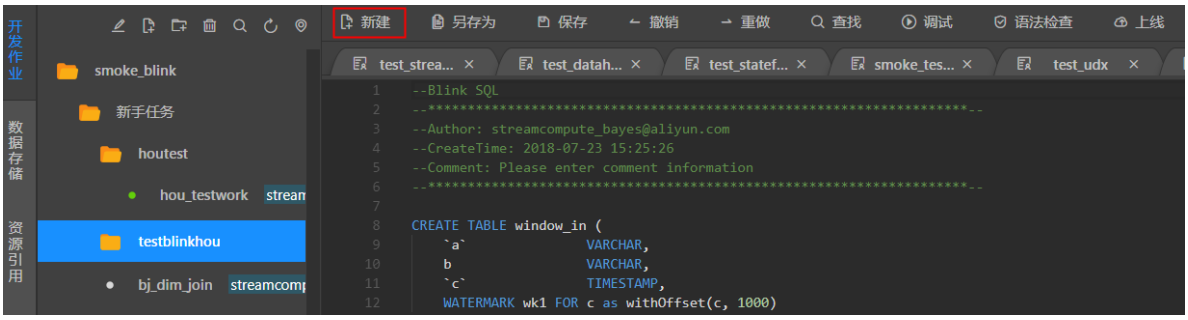
10.3.4.1 创建作业

本节介绍如何创建新的作业。

操作步骤

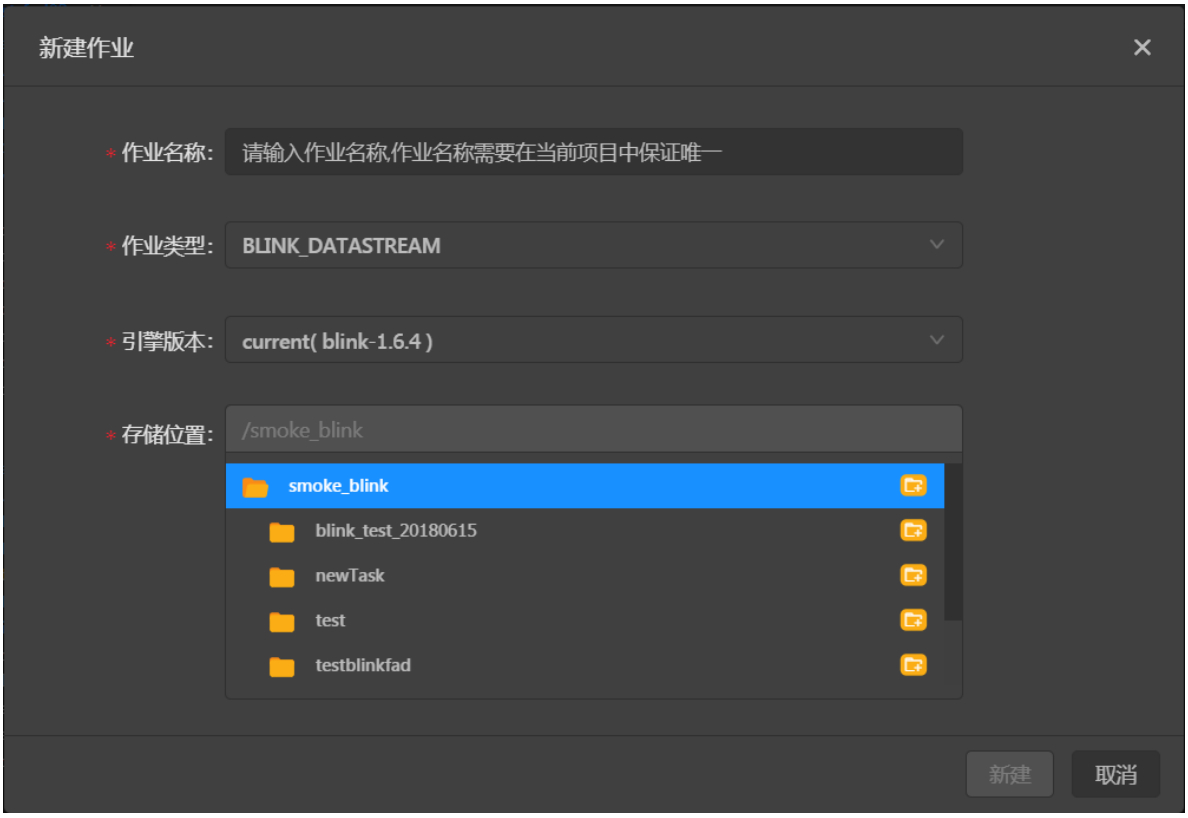
1. 登录流计算控制台，进入阿里云流计算产品首页。
2. 点击头部导航栏**开发**页签，进入数据开发页面。
3. 在菜单栏中点击**新建**，如下图所示。

图 10-35: 新建作业



4. 在**新建作业**对话框中配置相关参数。

图 10-36: 配置作业



- 作业名称：输入新作业的名称，需以字母开头，只能包含小写英文字母、数字、下划线（_），长度限制为3~64个字符。
- 作业类型：可以创建三种作业类型，分别是：BLINK_DATASTREAM、BLINK_TABLEAPI和BLINK_SQL。
- 引擎版本：保持默认版本即可。

- 存储位置：在文件夹目录中，指定该作业的代码文件所属的文件夹。您还可以点击现有文件夹右侧的图标，新建子文件夹。

5. 点击**新建**完成作业配置。

10.3.4.2 开发阶段

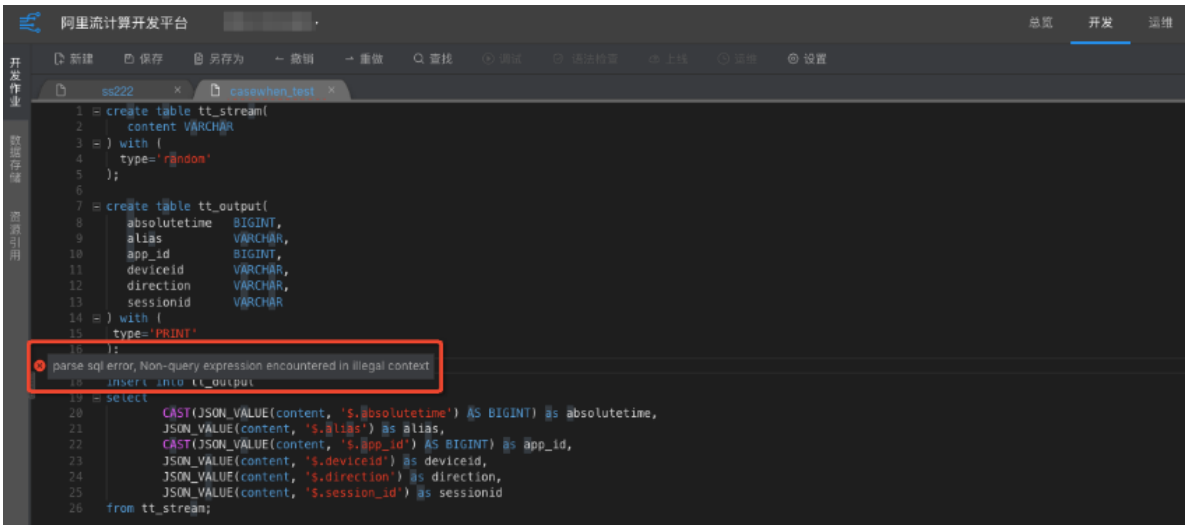
10.3.4.2.1 SQL辅助

数据开发提供了一套完整的在线SQL IDE工具，支持如下功能辅助您进行BlinkSQL开发。

- **BlinkSQL语法检查**

  您在修改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。

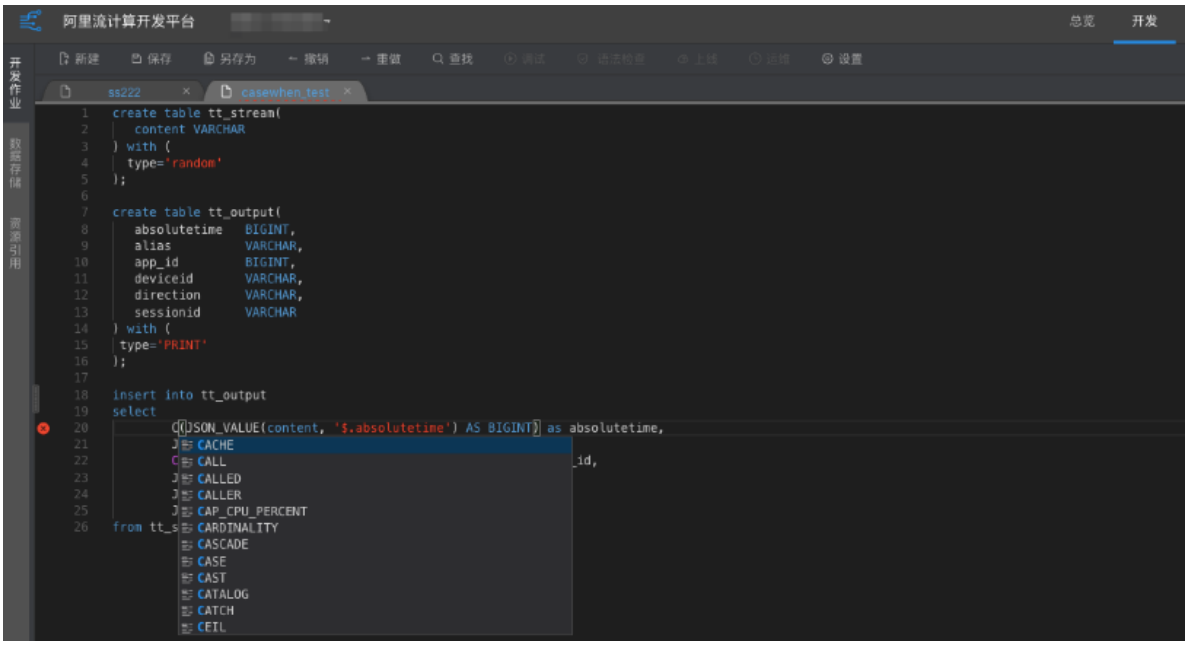
  图 10-37: 异常信息图



- BlinkSQL智能提示

  您在输入BlinkSQL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。

图 10-38: 异常信息图



- BlinkSQL语法高亮

  针对BlinkSQL关键字，提供不同颜色的语法高亮功能，以区分BlinkSQL不同结构。

10.3.4.2.2 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。您每次提交即可生成一个代码版本，代码版本为追踪修改以及日后回滚所用。

- 版本管理

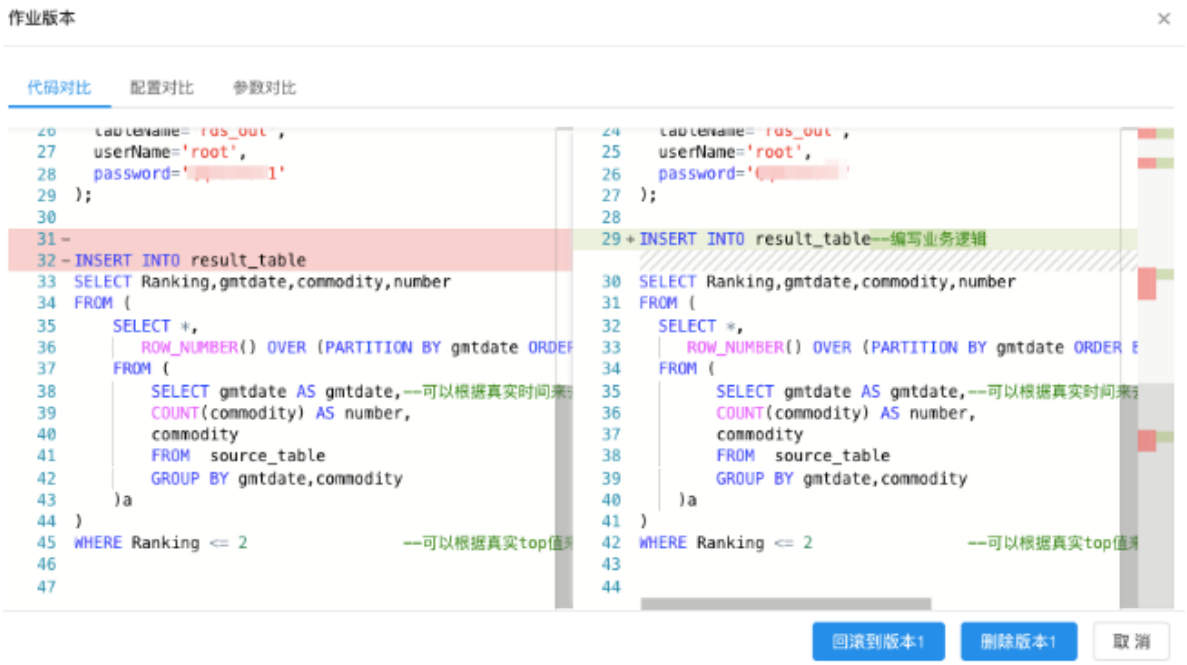
  您每次提交一个作业，发布线上，流计算均会生成一份代码快照用于日后的代码追踪使用。单击**数据开发 > 作业属性** ，在**作业属性**下展示了该作业的所有版本信息。

- 版本清理

  您每次提交一个作业，发布线上，流计算均会生成一份代码快照用于日后的代码追踪使用。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，报错提示您需要删除部分旧版本作业。

  此时导航至**开发 > 作业属性**，单击**版本列表**下的**删除**，删除过期且业务不需要的版本后，即可再次进行上线作业操作。

图 10-39: 代码对比



10.3.4.2.3 数据存储管理

开发页面提供了一整套数据存储管理的便捷工具，您通过**开发**页面注册数据源，即可享受到多种遍历的数据存储服务，如下所示。

- **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**

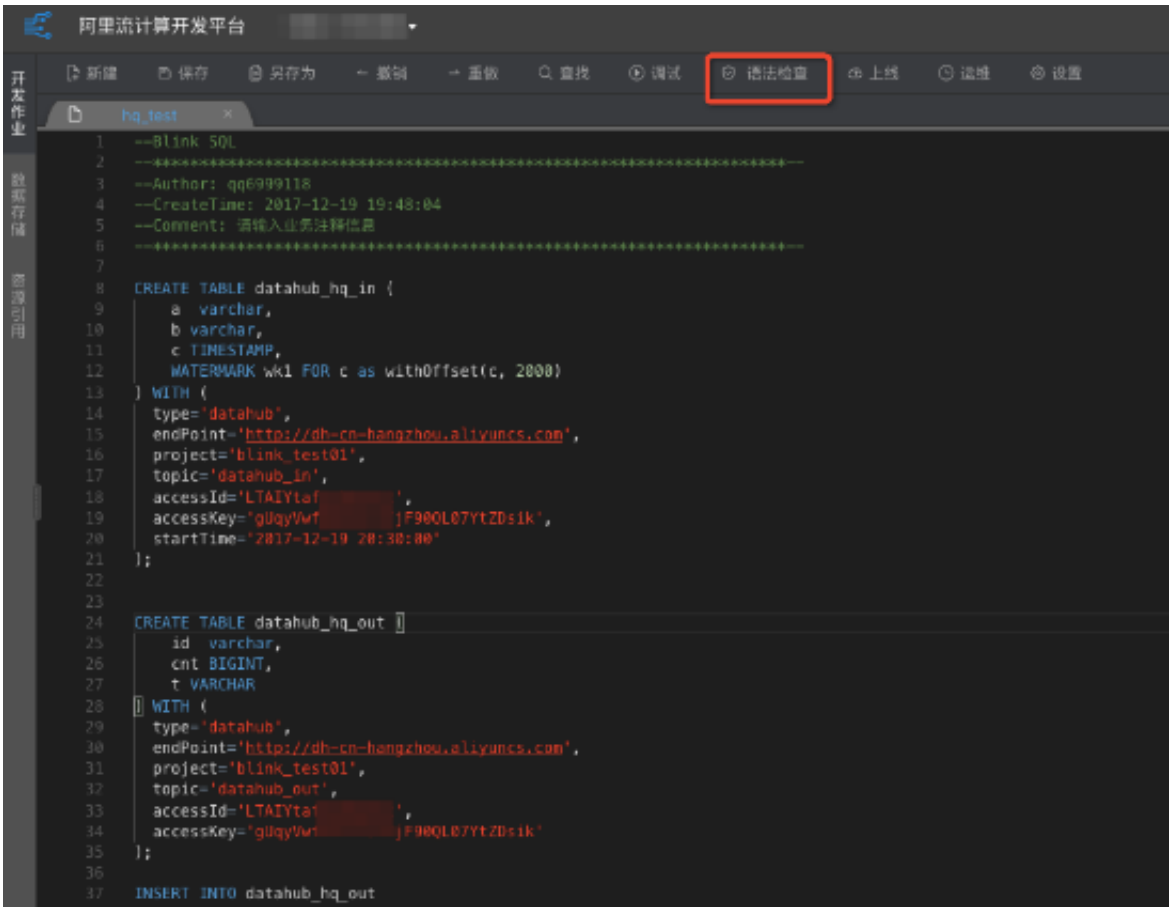
  流计算DDL生成工作大部分均属于比较机械的翻译工作，即将需要映射的数据存储DDL语句人工翻译为流计算的DDL语句。流计算提供辅助生成DDL工作，进一步减少您手工编写流式任务的复杂度，有效降低人工编写SQL的错误率，并最终提供流计算业务产出效率。

10.3.4.3 调试阶段

数据开发模块为您提供了一套模拟的运行环境，您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。当您写完所有的业务逻辑，接下来的操作步骤如下所示。

1. 语法检查。检测SQL中是否有语法错误，如果有错误会直接弹出错误。

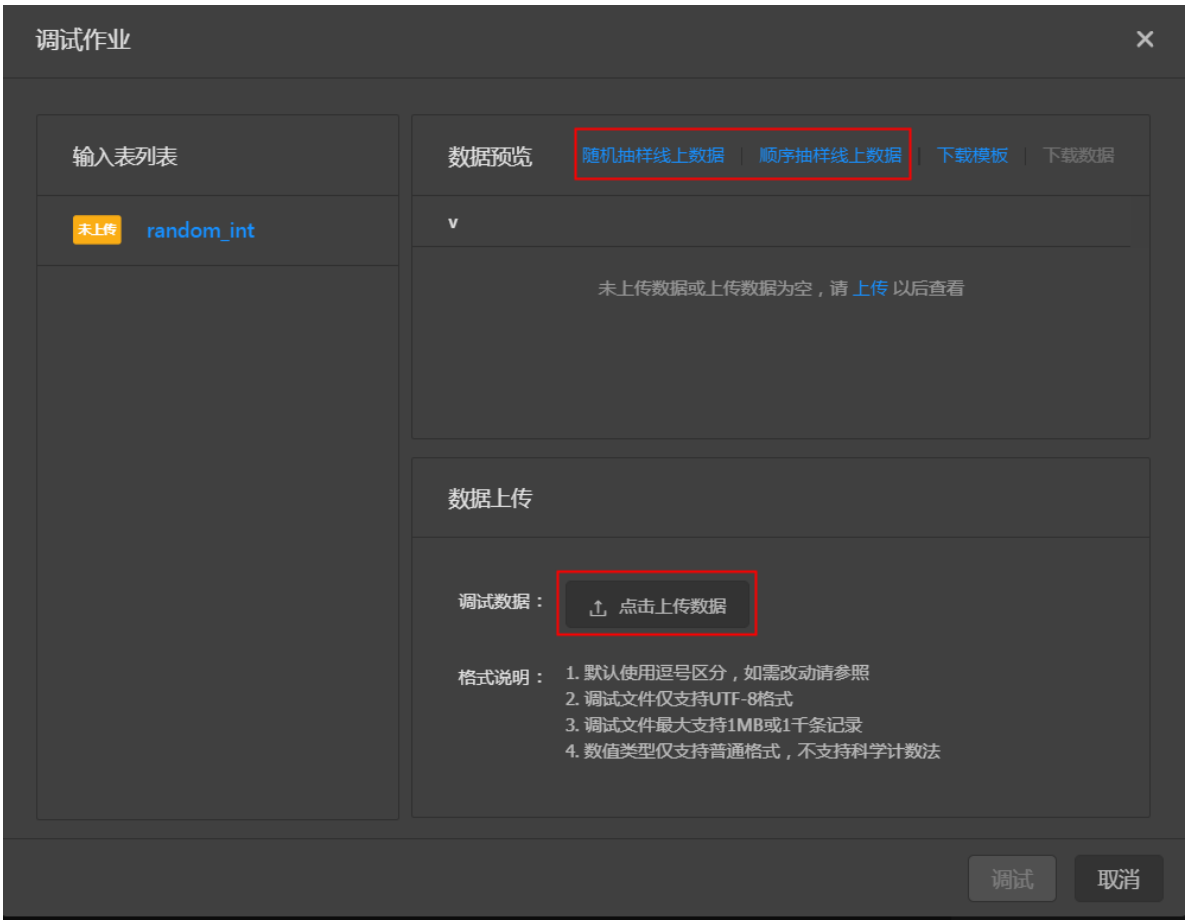
图 10-40: 语法检查



2. 进行作业调试。为方便您调试作业，流计算支持以下两种调试模式。

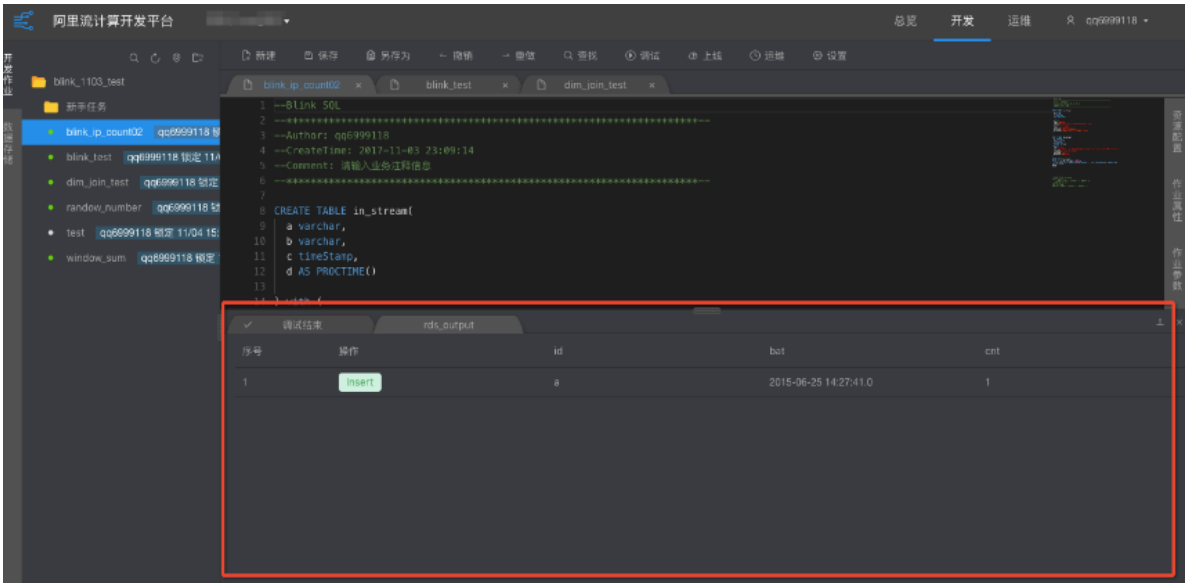
   - 自行构建测试数据。
   - 从数据源表中抽取数据，一种是随机抽取数据，另一种是顺序抽取数据。**但是注意必须使用了数据存储中注册功能才可以使用。抽取数据建议使用纯文本，不要使用Excel，否则会导致数据测试不准确。**

图 10-41: 配置调试文件



3. 查看调试结果。

图 10-42: 调试结果



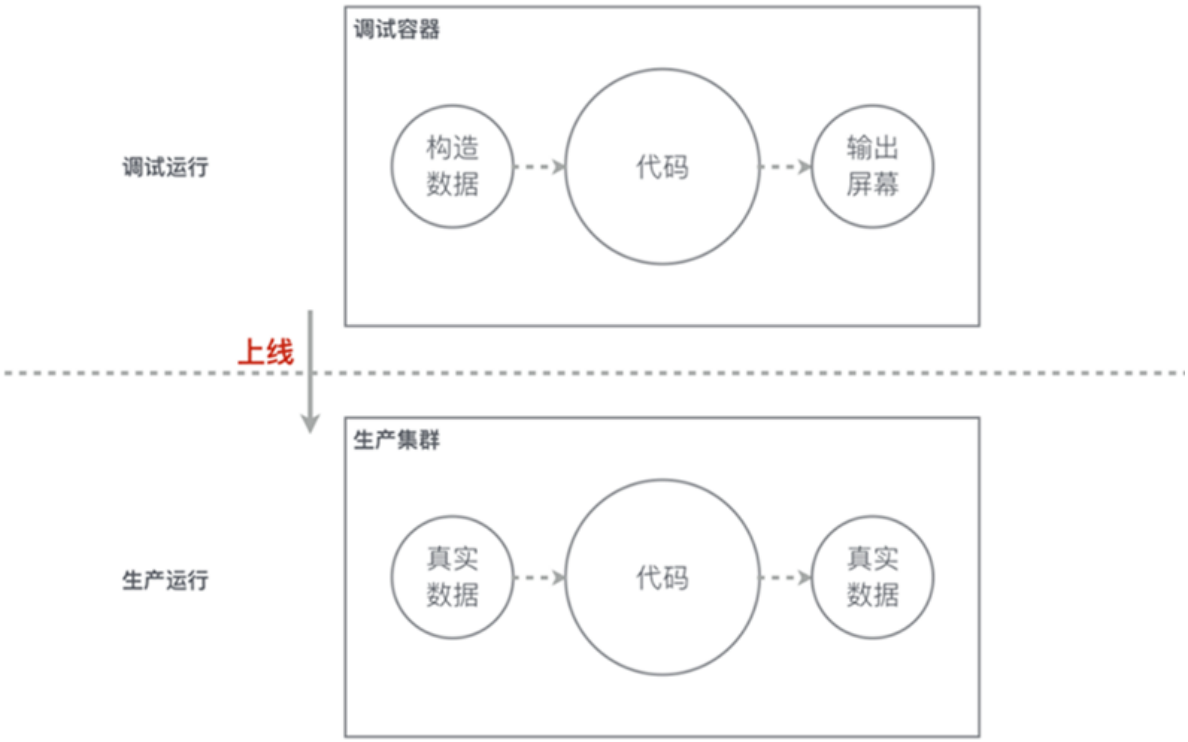
在该环境运行BlinkSQL可以实现以下功能。

- **生产完全隔离**

  调试环境下，所有的BlinkSQL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，让您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。假设您的结果数据输出到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。

  图 10-43: 生产调试



- **支持构造测试数据**

调试环境下，所有的BlinkSQL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。调试环境要求您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**。

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，不必担心测试数据构造困难。

说明：

强烈建议您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，例如您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。但一旦您需要使用JSON作为字段内容，字段内容即包含了逗号，此时您需要指定分割符为其他字符。

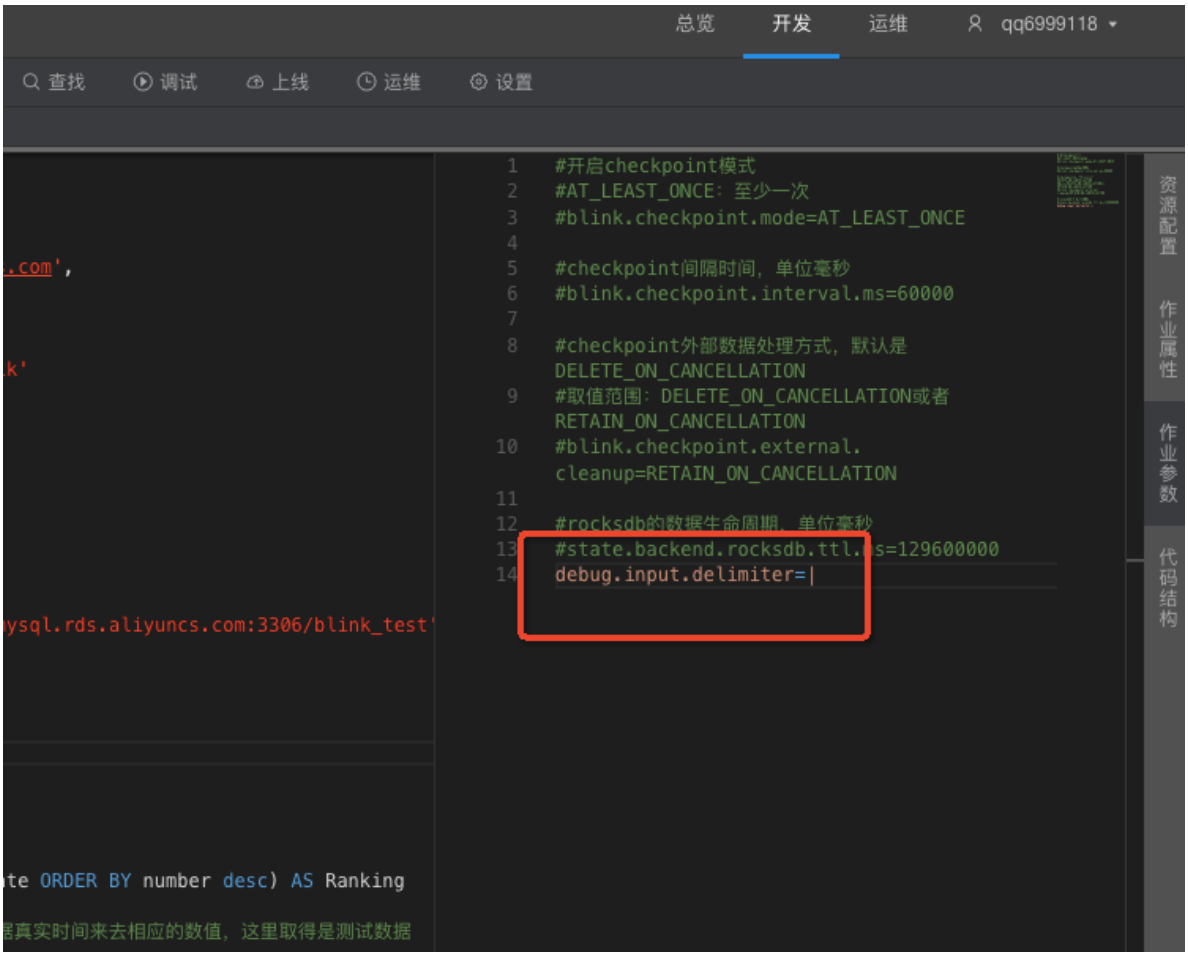
说明：

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。

```
id|name|age
1|alicloud|13
2|stream|1
```

  此时您需要针对该数据存储的作业参数设置debug.input.delimiter=|。

图 10-44: 调试分隔符



10.3.4.4 上线阶段

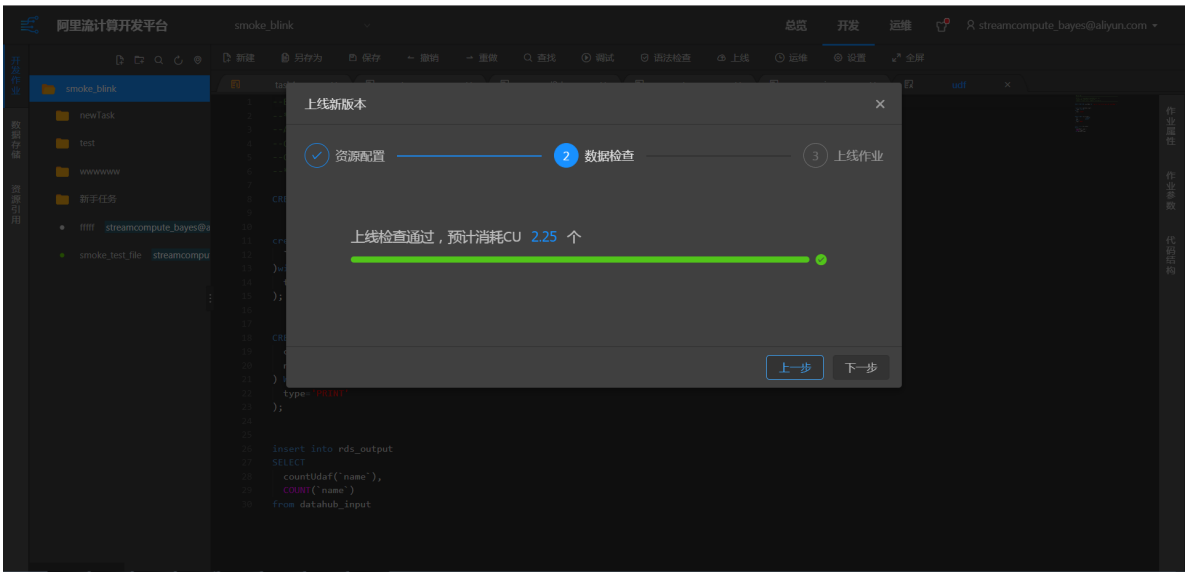
操作步骤

1. 完成开发、调试阶段，验证BlinkSQL正确无误后，单击**上线**。
2. 选择**智能CU配置**，第一次不需要指定CU数量，直接用系统默认配置即可。点击**下一步**。

图 10-45: 资源配置

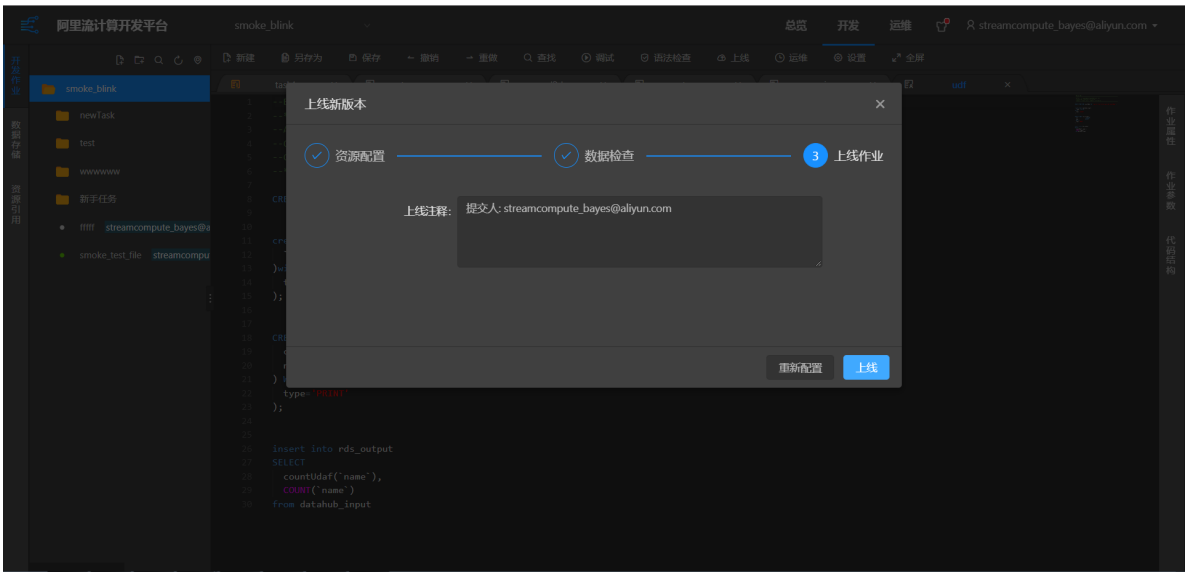
3. 检查数据。完成后，点击**下一步**。

图 10-46: 数据检查



4. 点击**上线**完成作业的上线。

图 10-47: 上线作业

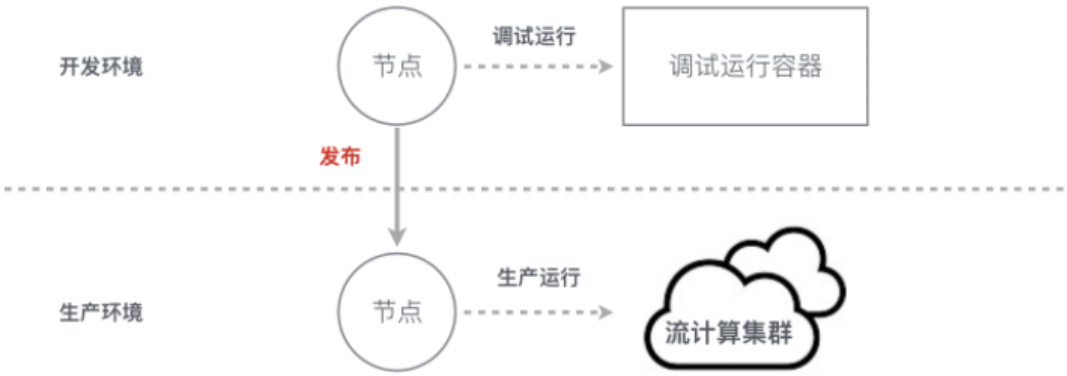


5. 进入**运维**页面，点击相应的作业进行启动。

图 10-48: 启动作业

在**开发**页面中，用户所有的修改、调试操作均和**运维**页面下的真正运行的生产调试完全隔离，从而保证了开发、生产互不干扰。

图 10-49: 开发和生产隔离



11 实时数据分发平台DataHub

11.1 什么是实时数据分发平台

实时数据分发平台（DataHub）是流式数据（Streaming Data）的处理平台，提供对流式数据的发布（Publish）、订阅（Subscribe）及分发功能，让用户可以轻松构建基于流式数据的分析和应用。

DataHub服务可以对各种移动设备、应用软件、网站服务、传感器等产生的大量流式数据进行持续不断的采集、存储和处理。用户可以编写应用程序或者使用流计算引擎来处理写入到DataHub的流式数据（例如：实时web访问日志、应用日志、各种事件等），并且能够产出各种实时的数据处理结果（例如：实时图表、报警信息、实时统计等）。

DataHub服务基于阿里云自研的飞天操作平台，具有高可用、低延迟、高可扩展、高吞吐的特点。DataHub与阿里云流计算引擎StreamCompute无缝连接，用户可以轻松使用SQL进行流数据分析。

DataHub服务也提供分发流式数据到各种云产品的功能，目前支持分发到MaxCompute、OSS等。

11.2 快速开始

11.2.1 登录实时数据分发平台控制台

本节以Chrome浏览器为例，介绍云产品用户如何登录到实时数据分发平台控制台。

前提条件

登录实时数据分发平台控制台前，需要您确认以下信息：

- 管理员已在BCC控制台上创建云账号。
- 已获取云账号对应的AK信息。
- 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. 登录数据中心管理框架，如下图所示。

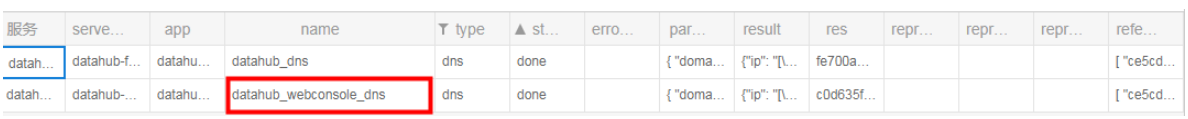
图 11-1: 数据中心管理框架

2. 在左侧选择**集群**，通过模糊查询DataHub，筛选出DataHub对应的集群。
3. 选择筛选出的DataHub集群，将鼠标指向后面的菜单项图标并选择**Dashboard**，进入DataHub的**集群Dashboard**界面。
4. 在**集群资源**列表中，单击列表的**type**表头，以dns进行筛选，如下图所示。

图 11-2: 筛选条件

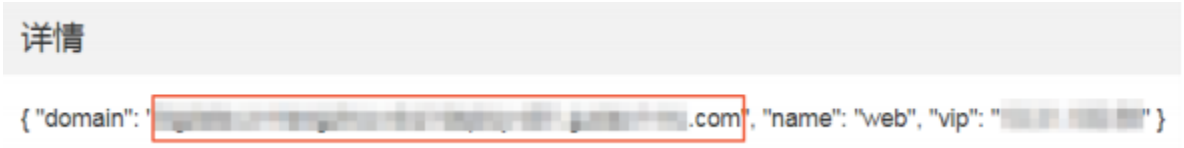
5. 在筛选结果中查找**name**为**datahub_webconsole_dns**的行，如下图所示。

图 11-3: 筛选结果



6. 右键单击该行的**parameters**单元格，并选择**显示更多**，在弹出的**详情**信息框中查看DataHub的域名，如下图所示。

图 11-4: 域名获取



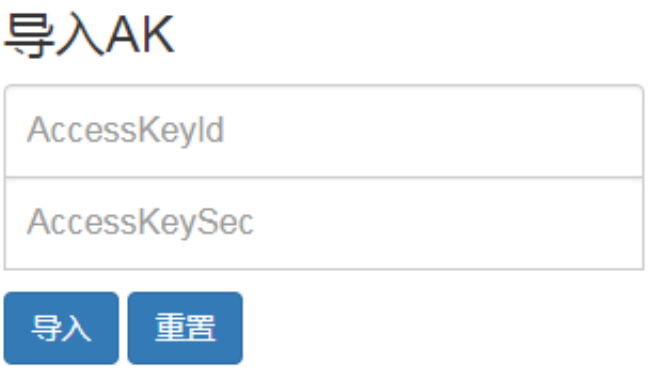
7. 打开Chrome浏览器。
8. 在地址栏中，输入查找到的DataHub的域名，按回车键。

   显示登录界面如下图所示。

图 11-5: 登录界面

9. 输入从部署人员处获取的阿里云账号及密码并单击**登录**。
10. 在跳转到的第二层登录页面，输入从部署人员处获取的阿里云账号对应的AK信息后，单击**导入**，进入实时数据分发平台控制台。

图 11-6: 输入AK信息并登录控制台

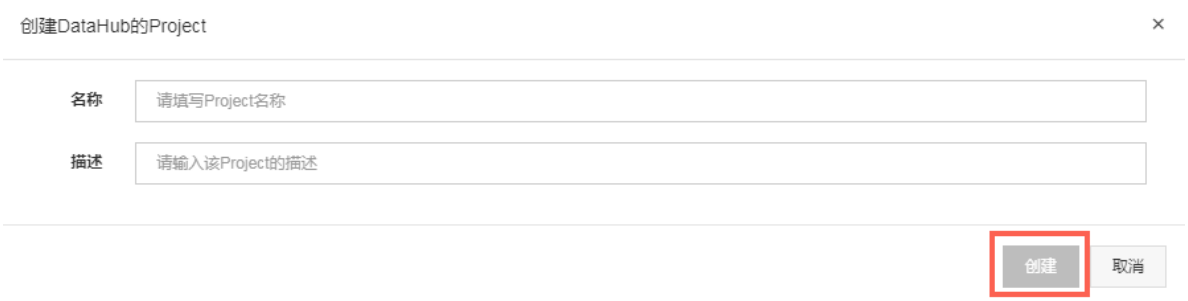


11.2.2 创建Project

1. 登录实时数据分发平台控制台。
2. 在默认进入的**项目管理**页签，单击**创建Project**，在弹出窗口中填写Project名称以及Project描述后，单击**创建**，完成创建Project操作。

> **说明**：
> 名称和描述均为必填项，填写缺失会导致无法创建，其中名称项不支持中文输入。

图 11-7: 创建Project



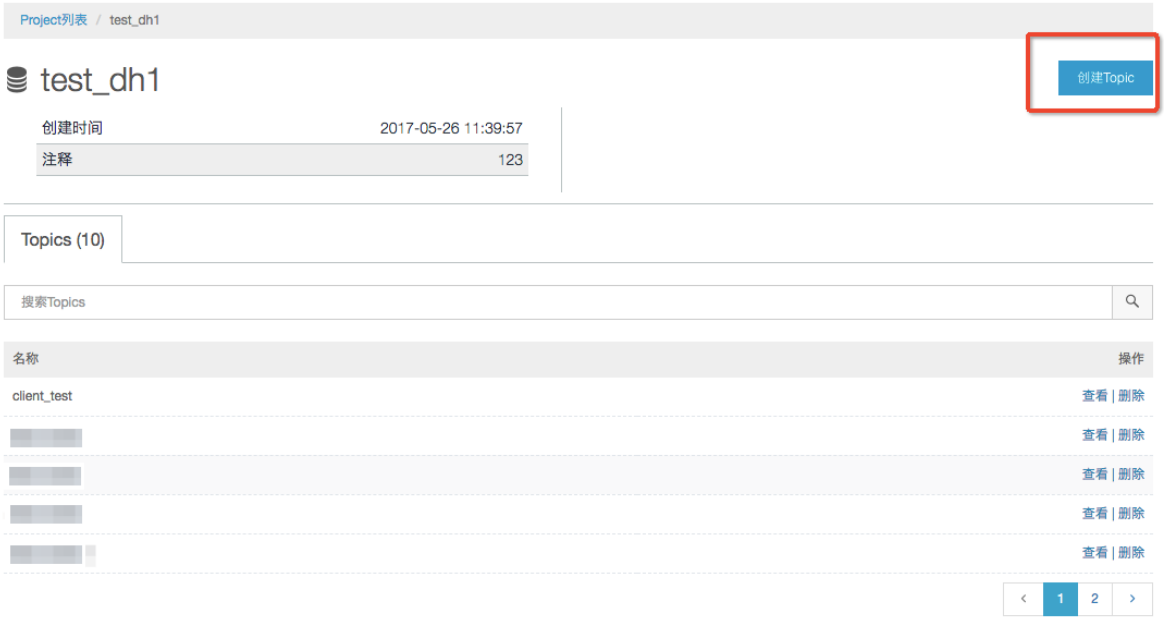
11.2.3 创建Topic

1. 登录实时数据分发平台控制台。
2. 在默认进入的**项目管理**页签，单击Project列表后的**查看**，进入Project详情页面。

图 11-8: 查看Project详情

3. 在Project详情页面单击**创建Topic**，在该Project中进行创建Topic操作。

图 11-9: 创建Topic



4. 在弹出的Topic创建页面中配置好Topic相关信息后，单击**创建**，完成创建Topic操作。

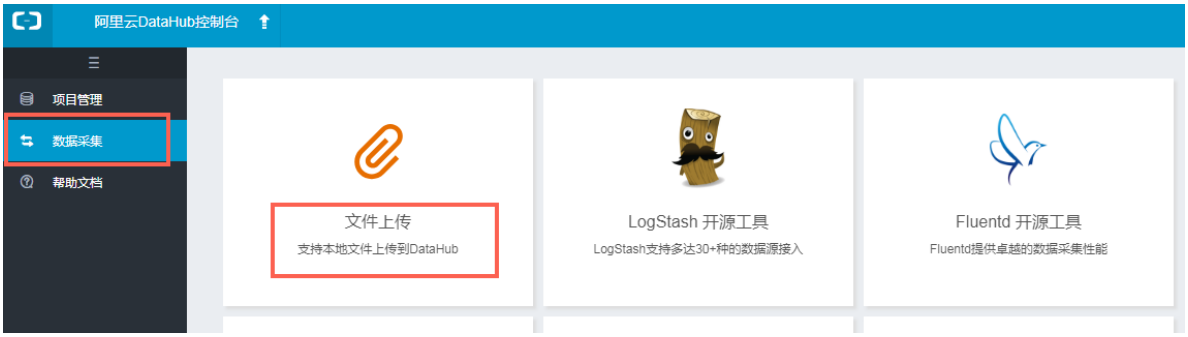
图 11-10: 配置Topic信息

11.2.4 文件上传

使用文件上传功能，用户不再需要额外对数据进行编码解析（例如使用SDK编程后解析数据或者配置日志采集等），即可以将想要导入datahub的数据（例如文本数据data.txt）直接通过上传功能完成导入。

1. 登录实时数据分发平台控制台。
2. 切换到**数据采集**页签，单击右侧的**文件上传**。

图 11-11: 数据采集-文件上传



3. 查看详细的文件上传要求后，设置存储信息并选择要上传的文件，单击**点击开始上传文件**，进行文件上传操作。

图 11-12: 上传操作

4. 上传成功后，弹出文件上传成功提示框。

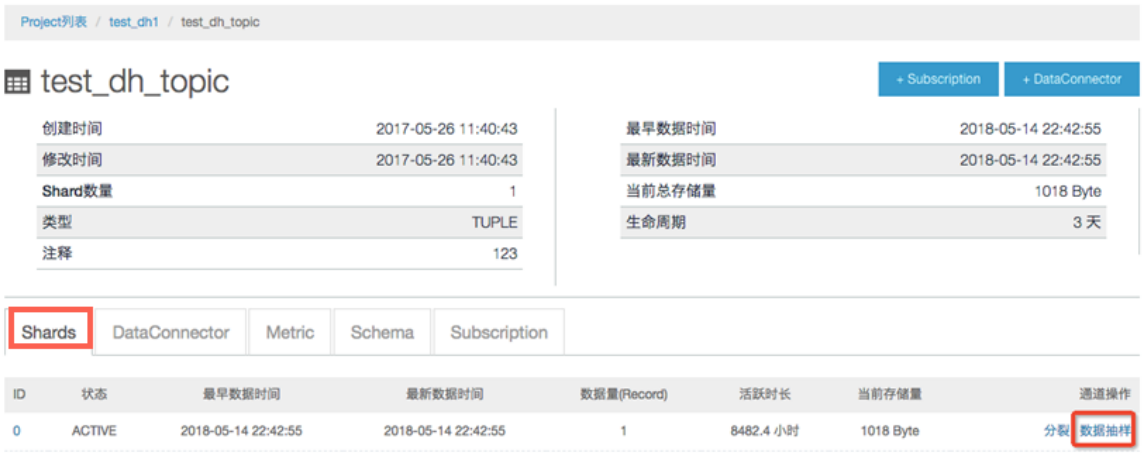
图 11-13: 上传成功

11.2.5 数据抽样

1. 登录实时数据分发平台控制台。

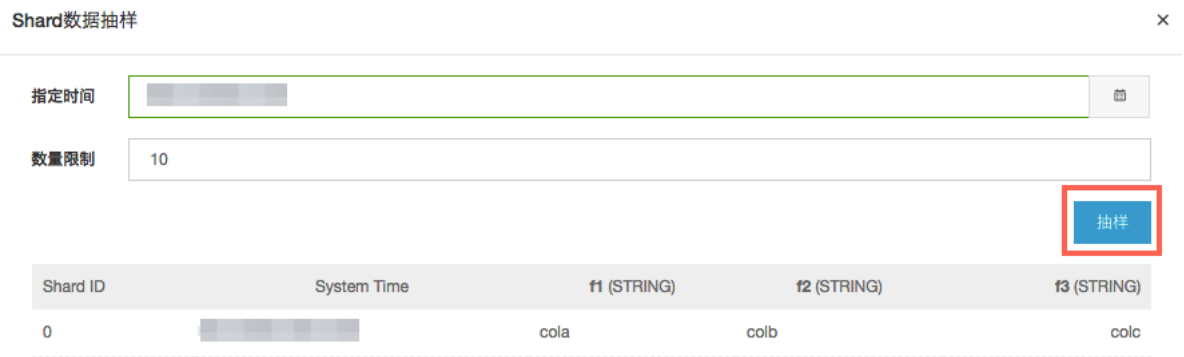
2. 在默认进入的**项目管理**页签，按先后顺序，分别单击Project列表后及Topic列表后的**查看**，进入Topic详情页面。
3. 在Topic的详情页面，单击shard列表右侧的**数据抽样**，对该Shard进行数据抽样查看。

图 11-14: Topic详情--数据抽样



4. 进入数据抽样页面，指定时间与记录条数限制后，单击**抽样**，数据抽样会显示指定时间之后写入到该Shard中的小于等于所指定限制数量的记录。

图 11-15: 数据抽样操作



11.3 使用指南

11.3.1 DataHub权限控制

DataHub采用阿里云RAM进行访问控制。用户对DataHub资源的访问，通过RAM进行鉴权。阿里云主账号拥有所属资源的所有权限，子用户在创建时并没有任何权限，不能访问任何资源，用户需要在RAM中对该子用户进行授权操作。下面章节将介绍DataHub在RAM下的访问控制体系。

说明：

一般来说，专有云部门账号一般是主账号，不需要权限赋值；专有云个人账号一般是子账号，需要通过主账号对子账号做特殊的赋权。

11.3.1.1 DataHub资源

DataHub在RAM的访问控制中的资源体系包含Project、Topic和Subscription。其中Subscription是指对某个特定Project下的Topic的一次订阅。目前支持Project、Topic和Subscription级别的鉴权，不支持Shard的访问控制。

DataHub的每一种资源在RAM中都有两种资源描述：所有该种资源、某一个特定的该种资源。例如：指定某用户在特定region下的某个Project，在RAM中的资源描述为*acs:dhs:$region:$accountid:projects/$projectName*，其中*$region*、*$accountid*、*$projectName*分别为指定的区域、用户ID和项目名。

表 11-1: 资源描述

资源类型	RAM中的资源描述
SingleProject	acs:dhs:$region:$accountid:projects/$projectName
AllProject	acs:dhs:$region:$accountid:projects/*
SingleTopic	acs:dhs:$region:$accountid:projects/$projectName/topics/$topicName
AllTopic	acs:dhs:$region:$accountid:projects/$projectName/topics/*
SingleSubscription	acs:dhs:$region:$accountid:projects/$projectName/topics/$topicName/subscriptions/$subId
AllSubscription	acs:dhs:$region:$accountid:projects/$projectName/topics/$topicName/subscriptions/*

11.3.1.2 DataHub API

DataHub Api包括Project、Topic、Shard、Subcription以及Record五个级别，针对每个资源都有不同的API，每个API对应的RAM Action和RAM Resource不尽相同。每种API的对应的授权策略Action及Resource如下所示。

Project API

表 11-2: Project API

DataHub API	RAM Action	资源类型
CreateProject	dhs:CreateProject	AllProject
ListProject	dhs:ListProject	AllProject
DeleteProject	dhs:DeleteProject	SingleProject
GetProject	dhs:GetProject	SingleProject
UpdateProject	dhs: UpdateProject	SingleProject

Topic API

表 11-3: Topic API

DataHub API	RAM Action	资源类型
CreateTopic	dhs:CreateTopic	AllTopic
ListTopic	dhs:ListTopic	AllTopic
DeleteTopic	dhs:DeleteTopic	SingleTopic
GetTopic	dhs:GetTopic	SingleTopic
UpdateTopic	dhs: UpdateTopic	SingleTopic

Subscription API

表 11-4: Subscription API

DataHub API	RAM Action	资源类型
CreateSubscription	dhs:CreateSubscription	AllSubscription
ListSubscription	dhs:ListSubscription	AllSubscription
DeleteSubscription	dhs:DeleteSubscription	SingleSubscription
GetSubscription	dhs:GetSubscription	SingleSubscription
UpdateSubscription	dhs: UpdateSubscription	SingleSubscription
CommitOffset	dhs:CommitOffset	SingleSubscription
GetOffset	dhs:GetOffset	SingleSubscription

Shard API

表 11-5: Shard API

DataHub API	RAM Action	资源类型
ListShard	dhs:ListShard	SingleTopic
MergeShard	dhs:MergeShard	SingleTopic
SplitShard	dhs:SplitShard	SingleTopic

Shard API

表 11-6: Shard API

DataHub API	RAM Action	资源类型
PutRecords	dhs:PutRecords	SingleTopic
GetRecords	dhs:GetRecords	SingleTopic
GetCursor	dhs:GetRecords	SingleTopic

11.3.1.3 DataHub支持的Condition

DataHub在RAM下支持的Condition如下表所示。

表 11-7: DataHub支持的Condition

条件(Condition)	功能	合法取值
acs:SourceIp	指定IP网段	普通IP，支持*通配
acs:SecureTransport	是否是https协议	true/false
acs:MFAPresent	是否多设备认证	true/false
acs:CurrentTime	指定访问时间	ISO8601格式

11.3.1.4 DataHub Policy示例

11.3.1.4.1 AliyunDataHubFullAccess

示例如下，仅供参考：

```
{
  "Version": "1",
  "Statement": [
    {
```

```
      "Action": "dhs:*",
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

11.3.1.4.2 AliyunDataHubReadOnlyAccess

示例如下，仅供参考：

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": ["dhs:List*", "dhs:Get*"],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

11.3.2 DataHub数据采集

除了直接使用SDK和Web控制台中的文件上传功能进行数据发布之外，DataHub还提供多种数据采集工具，方便用户快速实现/迁移数据收集方案。其中包括Fluentd、Logstash以及Oracle Golden Gate。

11.3.2.1 Fluentd

该插件是基于Fluentd开发的输出插件，主要是将采集到的数据写入DataHub。该插件遵守fluentd输出插件开发规范，安装方便，可以很方便地将采集到的数据写到DataHub。

产品安装

- 通过ruby gem安装。

  ```
  gem install fluent-plugin-datahub
  ```

  注意：
  RubyGem源建议更改为https://ruby.taobao.org/。

- 本地安装。

  当前Fluentd仅支持Linux环境，同时要求用户安装Ruby。目前支持两种安装模式，对于没有安装过fluentd的客户提供一键fluent+datahub的全部安装模式，对于曾经安装了fluentd的客户提供datahub写入插件的单独安装模式。

- 一键安装：如果之前没安装过fluentd，请下载*Fluentd*完全安装包，并使用如下命令安装插件。

  注意：
  完全安装包提供的fluentd是fluentd-0.12.23.gem的版本。

  ```
  $ tar -xzvf fluentd-with-datahub-0.12.23.tar.gz
  $ cd fluentd-with-dataHub
  $ sudo sh install.sh
  ```

- 单独安装：如果之前安装过fluentd，请下载*Fluentd datahub*插件包，并使用gem命令安装datahub插件。

  ```
  $ sudo gem install --local fluent-plugin-dataHub-0.0.2.gem
  ```

使用案例

案例一：CSV文件上传。

下面将以增量的CSV文件为例，说明如何使用fluentd将增量的csv文件准实时上传到DataHub数据。CSV文件的格式示例如下所示：

```
0,qe614c760fuk8judu01tn5x055rpt1,true,100.1,14321111111
1,znv1py74o8ynn87k66o32ao4x875wi,true,100.1,14321111111
2,7nm0mtpgo1q0ubuljjjx9b000ybltl,true,100.1,14321111111
3,10t0n6pvonnan16279w848ukko5f6l,true,100.1,14321111111
4,0ub584kw88s6dczd0mta7itmta10jo,true,100.1,14321111111
5,1ltfpf0jt7fhvf0oy4lo8m3z62c940,true,100.1,14321111111
6,zpqsfxqy9379lmcehd7q8kftntrozb,true,100.1,14321111111
7,ce1ga9aln346xcj761c3iytshyzuxg,true,100.1,14321111111
8,k5j2id9a0ko90cykl40s6ojq6gruyi,true,100.1,14321111111
9,ns2zcx9bdip5y0aqd1tdicf7bkdmsm,true,100.1,14321111111
10,54rs9cm1xau2fk66pzyz62tf9tsse4,true,100.1,14321111111
```

上述CSV文件中每行一条record，按照(,)区分字段。保存在本地路径/temp/test.csv中。DataHub Topic格式如下所示。

表 11-8: DataHub Topic格式

字段名称	字段类型
id	BIGINT
name	STRING
gender	BOOLEAN
salary	DOUBLE

字段名称	字段类型
my_time	TIMESTAMP

对于以上数据文件以及Topic Schema，编辑Fluentd配置文件后，使用如下命令启动fluentd，即可完成CSV文件数据采集进入DataHub，即CSV文件上传。

```
${FLUENTD_HOME}/fluentd-with-dataHub/bin/fluentd -c fluentd_test.conf
```

对应的Fluentd配置文件如下所示：

```
<source>
  @type tail
  path /xxx/yyy（用户的文件路径）
  tag test1
  format csv
  keys id,name,gender,salary,my_time
</source>

<match test1>
  @type dataHub
  access_id your_app_id
  access_key your_app_key
  endpoint http://ip:port
  project_name test_project
  topic_name fluentd_performance_test_1
  column_names ["id", "name", "gender", "salary", "my_time"]
  flush_interval 1s
  buffer_chunk_limit 3m
  buffer_queue_limit 128
  dirty_data_continue true
  dirty_data_file /xxx/yyy（脏数据记录文件路径）
  retry_times 3
  put_data_batch_size 1000
</match>
```

案例二：Log4J日志采集。

Log4j的日志格式示例如下所示：

```
11:48:43.439 [qtp1847995714-17] INFO  AuditInterceptor - [c2un5sh7cu
52ek6am1ui1m5h] end /web/v1/project/tefe4mfurtix9kwwyrvfqd0m/node/
0m0169kapshvgc3ujskwkk8g/health GET, 4061 ms
```

对应的Fluentd配置文件如下所示：

```
<source>
  @type tail
  path bayes.log
  tag test
  format /(?<request_time>\d\d:\d\d:\d\d.\d+)\s+\[(?<thread_id>[\w
\-]+)\]\s+(?<log_level>\w+)\s+(?<class>\w+)\s+-\s+\[(?<request_id>\w
+)\]\s+(?<detail>.+)/
</source>
```

```
<match test>
  @type dataHub
  access_id your_access_id
  access_key your_access_key
  endpoint http://ip:port
  project_name test_project
  topic_name dataHub_fluentd_out_1
  column_names ["thread_id", "log_level", "class"]
</match>
```

插件配置

表 11-9: 读插件配置

配置项	描述
tag test1	指定路由，和<match>会进行路由正则匹配
format csv	数据按照csv方式采集
keys id,name,gender,salary,my_time	指定需要采集的列名，必须和目的DataHub表的列名一致

表 11-10: 写插件配置

配置项	描述
shard_id 0	指定shard_id写入，默认round-robin方式写入
shard_keys ["id"]	指定用作分区key，用key值hash后作为写入shard的索引
flush_interval 1	fluentd每一秒钟至少写一次, 默认60s
buffer_chunk_limit 3m	块大小，支持“k”（KB）、“m”（MB）单位，建议值3m
buffer_queue_limit 128	块队列大小，此值与buffer_chunk_limit共同决定整个缓冲区大小
put_data_batch_size 1000	每1000条record写一次DataHub
retry_times 3	重试次数
retry_interval 3	重试间隔（单位：s）
dirty_data_continue true	遇到增数据是否继续，若为true 遇到脏数据会重试，重试次数用完，会将脏数据写入脏数据文件
dirty_data_file /xxx/yyy	指定脏数据文件的位置
column_names ["id"]	指定需要采集的列

11.3.2.2 Logstash

Logstash是一种分布式日志收集框架，经常与ElasticSearch、Kibana配置，组成著名的ELK技术栈，非常适合用来做日志数据的分析。 阿里云流计算为了方便用户将更多数据采集进入DataHub，提供了针对Logstash的DataHub Output/Input插件。使用Logstash，用户可以轻松享受到Logstash开源社区多达30+种数据源支持（file、syslog、redis、log4j、apache log或nginx log）；同时Logstash还支持filter对传输字段自定义加工等功能。下面将以Logstash日志采集为例，介绍如何使用Logstash快速将日志数据采集进入DataHub。

产品安装

Logstash安装要求JRE 7版本及以上，否则部分功能无法使用。DataHub logstash插件提供两种安装方式。

- 一键安装：一键安装包，支持将Logstash和DataHub插件一键安装，下载地址：一键安装包，当前提供了免安装的版本，解压即可使用:。

  使用如下命令执行完成后，logstash即安装成功。

  ```
  $ tar -xzvf logstash-with-datahub-2.3.0.tar.gz
  $ cd logstash-with-datahub-2.3.0
  ```

- 单独安装。
  - 安装Logstash：参考Logstash官网提供的安装教程，进行安装工作。官网地址为：*https://www.elastic.co/guide/en/logstash/current/index.html*。
  - 上传数据至DataHub：请使用：*DataHub Logstash Output*插件。
  - 下载DataHub中数据：请使用：*DataHub Logstash Input*插件。

使用案例

案例一：Log4j日志采集。

下面将以Log4j日志采集为例，介绍如何使用Logstash采集并结构化日志数据。Log4j的日志格式示例如下所示：

```
20:04:30.359 [qtp1453606810-20] INFO  AuditInterceptor - [13pn9kdr5t
l84stzkmaa8vmg] end /web/v1/project/fhp4clxfbu0w3ym2n7ee6ynh/
statistics?executionName=bayes_poc_test GET, 187 ms
```

针对上述Log4j文件，用户希望将数据结构化并采集进入DataHub。DataHub Topic格式如下所示。

表 11-11: DataHub Topic格式

字段名称	字段类型
request_time	STRING
thread_id	STRING
log_level	STRING
class_name	STRING
request_id	STRING
detail	STRING

Logstash任务配置如下所示：

```
input {
    file {
        path => "${APP_HOME}/log/bayes.log"
        start_position => "beginning"
    }
}

filter{
    grok {
        match => {
           "message" => "(?<request_time>\d\d:\d\d:\d\d\.\d+)\s+\[(?<
thread_id>[\w\-]+)\]\s+(?<log_level>\w+)\s+(?<class_name>\w+)\s+\-\s+
\[(?<request_id>\w+)\]\s+(?<detail>.+)"
        }
    }
}

output {
    datahub {
        access_id => "Your accessId"
        access_key => "Your accessKey"
        endpoint => "Endpoint"
        project_name => "project"
        topic_name => "topic"
        #shard_id => "0"
        #shard_keys => ["thread_id"]
        dirty_data_continue => true
        dirty_data_file => "/Users/ph0ly/trash/dirty.data"
        dirty_data_file_max_size => 1000
    }
}
```

案例二：csv文件数据采集。

下面将以标准csv文件为例，介绍如何使用Logstash采集csv文件数据。CSV文件的格式示例如下所示：

```
1111,1.23456789012E9,true,14321111111000000,string_dataxxx0,
```

```
2222,2.23456789012E9,false,14321111111000000,string_dataxxx1
```

针对上述csv文件，DataHub Topic格式如下所示。

表 11-12: DataHub Topic格式

字段名称	字段类型
col1	BIGINT
col2	DOUBLE
col3	BOOLEAN
col4	TIMESTAMP
col5	STRING

Logstash任务配置如下所示：

```
input {
    file {
        path => "${APP_HOME}/data.csv"
        start_position => "beginning"
    }
}

filter{
    csv {
        columns => ['col1', 'col2', 'col3', 'col4', 'col5']
    }
}

output {
    datahub {
        access_id => "Your accessId"
        access_key => "Your accessKey"
        endpoint => "Endpoint"
        project_name => "project"
        topic_name => "topic"
        #shard_id => "0"
        #shard_keys => ["thread_id"]
        dirty_data_continue => true
        dirty_data_file => "/Users/ph0ly/trash/dirty.data"
        dirty_data_file_max_size => 1000
    }
}
```

案例三：消费Datahub数据写到文件。

Logstash任务配置如下所示：

```
input {
    datahub {
        access_id => "Your accessId"
        access_key => "Your accessKey"
```

```
        endpoint => "Endpoint"
        project_name => "test_project"
        topic_name => "test_topic"
        interval=> 5
        #cursor => {
        #    "0"=>"20000000000000000000000003110091"
        #    "2"=>"20000000000000000000000003110091"
        #    "1"=>"20000000000000000000000003110091"
        #    "4"=>"20000000000000000000000003110091"
        #    "3"=>"20000000000000000000000003110000"
        #}
        shard_ids => []
        pos_file => "/home/admin/logstash/logstash-2.3.0/pos_file"
    }
}

output {
    file {
        path => "/home/admin/logstash/logstash-2.3.0/output"
    }
}
```

运行Logstash

使用如下命令启动Logstash。

```
$LOGSTASH_HOME/bin/logstash -f <上述配置文件> -b 256
```

说明：

-f指定配置文件路径，-b指定每次向DataHub发送的记录条数，默认为125。

参数说明

Datahub output插件参数介绍：

- access_id(Required)：阿里云access id。
- access_key(Required)：阿里云access key。
- endpoint(Required)：阿里云datahub的服务地址。
- project_name(Required)：datahub项目名称。
- topic_name(Required)：datahub topic名称。
- retry_times(Optional)：重试次数，-1为无限重试、0为不重试、>0表示需要有限次数，默认值为-1。
- retry_interval(Optional)：下一次重试的间隔，单位为秒，默认值为5。
- shard_keys(Optional)：数组类型，数据落shard的字段名称，插件会根据这些字段的值计算shash将每条数据落某个shard。需注意，shard_keys和shard_id都未指定，默认轮询落shard。

- shard_id(Optional)：所有数据落指定的shard。需注意，shard_keys和shard_id都未指定，默认轮询落shard。
- dirty_data_continue(Optional)：脏数据是否继续运行，默认为false，如果指定true，则遇到脏数据直接无视，继续处理数据。当开启该开关，必须指定@dirty_data_file文件。
- dirty_data_file(Optional)：脏数据文件名称，当数据文件名称，在@dirty_data_continue开启的情况下，需要指定该值。需要特别注意，脏数据文件将被分割成两个部分.part1和.part2，其中part1作为更早的脏数据，part2作为更新的数据。
- dirty_data_file_max_size(Optional)：脏数据文件的最大大小，该值保证脏数据文件最大大小不超过这个值，目前该值仅是一个参考值。

Datahub input插件参数介绍：

- access_id(Required)：阿里云access id。
- access_key(Required)：阿里云access key。
- endpoint(Required)：阿里云datahub的服务地址。
- project_name(Required)：datahub项目名称。
- topic_name(Required)：datahub topic名称。
- retry_times(Optional)：重试次数，-1为无限重试、0为不重试、>0表示需要有限次数。
- retry_interval(Optional)：下一次重试的间隔，单位为秒。
- shard_ids(Optional)：数组类型，需要消费的shard列表，空列表默认全部消费。
- cursor(Optional)：消费起点，默认为空，表示从头开始消费。
- pos_file(Required)：checkpoint记录文件，必须配置，优先使用checkpoint恢复消费offset。

11.3.2.3 Oracle Golden Gate

随着数据规模的不断扩大，传统的RDBMS难以满足OLAP的需求，本章节将介绍如何将Oracle的数据实时同步到阿里云的大数据处理平台当中，并利用大数据工具对数据进行分析。

OGG（Oracle Golden Gate）是一个基于日志的结构化数据备份工具，一般用于Oracle数据库之间的主从备份以及Oracle数据库到其他数据库（DB2、MySQL等）的同步。OGG的部署分为源端和目标端两部分组成，主要有Manager、Extract、Pump、Collector、Replicat等组件。

- Manager：在源端和目标端都会有且只有一个Manager进程存在，负责管理其他进程的启停和监控等。

- Extract：负责从源端数据库表或者事务日志中捕获数据，有初始加载和增量同步两种模式可以配置。初始加载模式是直接将源表数据同步到目标端，而增量同步模式是分析源端数据库的日志，将变动的记录传到目标端，本章节介绍的是增量同步模式。
- Pump：Extract从源端抽取的数据会先写到本地磁盘的Trail文件，Pump进程会负责将Trail文件的数据投递到目标端。
- Collector：目标端负责接收来自源端的数据，生成Trail文件。
- Replicat：负责读取目标端的Trail文件，转化为相应的DDL和DML语句作用到目标数据库，实现数据同步。

DataHub OGG插件可以实现Replicat的功能，通过分析Trail文件，将数据的变化记录写入DataHub中，使用流计算对DataHub中的数据进行实时分析，同时可以将数据归档到MaxCompute中进行离线处理。

产品安装

安装的环境要求如下所示：

- 源端已安装好Oracle 。
- 已获取源端OGG安装包（建议版本Oracle GoldenGate V12.1.2.1）。
- 已获取目标端OGG Adapters安装包（建议版本Oracle GoldenGate Application Adapters 12.1.2.1）。
- 已安装java 7。

安装步骤如下所示：

1. 源端OGG安装。
   a. 将已获取的源端OGG安装包解压后，会显示如下目录：

```
drwxr-xr-x install
drwxrwxr-x response
-rwxr-xr-x runInstaller
drwxr-xr-x stage
```

   b. 由于目前oracle一般采取response安装的方式，需要在response/oggcore.rsp中配置安装依赖，具体示例如下：

```
oracle.install.responseFileVersion=/oracle/install/rspfmt_ogg
install_response_schema_v12_1_2
# 需要目前与oracle版本对应
INSTALL_OPTION=ORA11g
# goldegate主目录
SOFTWARE_LOCATION=/home/oracle/u01/ggate
```

```
# 初始不启动manager
START_MANAGER=false
# manger端口
MANAGER_PORT=7839
# 对应oracle的主目录
DATABASE_LOCATION=/home/oracle/u01/app/oracle/product/11.2.0/
dbhome_1
# 暂可不配置
INVENTORY_LOCATION=
# 分组（目前暂时将oracle和ogg用同一个账号ogg_test，实际可以给ogg单独账号）
UNIX_GROUP_NAME=oinstall
```

c. 执行如下命令，安装OGG。

```
runInstaller -silent -responseFile {YOUR_OGG_INSTALL_FILE_PATH}/
response/oggcore.rsp
```

说明：

本参考示例中，安装完成后OGG的目录在*/home/oracle/u01/ggate*，安装日志在*/home/oracle/u01/ggate/cfgtoollogs/oui*目录下，当**silentInstall{时间}.log**文件里出现如下提示，表明安装成功：

```
The installation of Oracle GoldenGate Core was successful.
```

d. 执行如下命令，并在提示符下键入命令`CREATE SUBDIRS`，生成OGG需要的各种目录。

```
/home/oracle/u01/ggate/ggsci
```

2. 源端Oracle配置。

以dba分身进入*sqlplus#sqlplus / as sysdba*，完成如下配置。

```
# 创建独立的表空间
create tablespace ATMV datafile '/home/oracle/u01/app/oracle/oradata
/uprr/ATMV.dbf' size 100m autoextend on next 50m maxsize unlimited;

 # 创建ogg_test用户，密码也为ogg_test
create user ogg_test identified by ogg_test default tablespace ATMV;

 # 给ogg_test赋予充分的权限
grant connect,resource,dba to ogg_test;

 # 检查附加日志情况
Select SUPPLEMENTAL_LOG_DATA_MIN, SUPPLEMENTAL_LOG_DATA_PK,
SUPPLEMENTAL_LOG_DATA_UI, SUPPLEMENTAL_LOG_DATA_FK, SUPPLEMENT
AL_LOG_DATA_ALL from v$database;

 # 增加数据库附加日志及回退
alter database add supplemental log data;
alter database add supplemental log data (primary key, unique,
foreign key) columns;
 # rollback
```

```
alter database drop supplemental log data (primary key, unique,
foreign key) columns;
alter database drop supplemental log data;

 # 全字段模式，注意：在该模式下的delete操作也只有主键值
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
 # 开启数据库强制日志模式
alter database force logging;
 # 执行marker_setup.sql 脚本
@marker_setup.sql
 # 执行@ddl_setup.sql
@ddl_setup.sql
 # 执行role_setup.sql
@role_setup.sql
 # 给ogg用户赋权
grant GGS_GGSUSER_ROLE to ogg_test;
 # 执行@ddl_enable.sql，开启DDL trigger
@ddl_enable.sql
 # 执行优化脚本
@ddl_pin ogg_test
 # 安装sequence support
@sequence.sql
#
alter table sys.seq$ add supplemental log data (primary key) columns
;
```

3. 源端OGG mgr配置。

   启动ggsci后执行如下操作。

   1. 配置mgr。

      ```
      edit params mgr
      PORT 7839
      DYNAMICPORTLIST  7840-7849
      USERID ogg_test, PASSWORD ogg_test
      PURGEOLDEXTRACTS ./dirdat/*, USECHECKPOINTS, MINKEEPDAYS 7
      LAGREPORTHOURS 1
      LAGINFOMINUTES 30
      LAGCRITICALMINUTES 45
      PURGEDDLHISTORY MINKEEPDAYS 3, MAXKEEPDAYS 7
      PURGEMARKERHISTORY MINKEEPDAYS 3, MAXKEEPDAYS 7
      ```

   2. 启动mgr（运行日志在ggate/dirrpt中）。

      ```
      start mgr
      ```

   3. 查看mgr状态。

      ```
      info mgr
      ```

   4. 查看mgr配置。

      ```
      view params mgr
      ```

4. 源端OGG extract配置。

启动ggsci后执行如下操作。

1. 配置extract（extract是示例组名，可以任意取）。

```
edit params extractEXTRACT extract
SETENV (NLS_LANG="AMERICAN_AMERICA.AL32UTF8")
DBOPTIONS   ALLOWUNUSEDCOLUMN
USERID ogg_test, PASSWORD ogg_test
REPORTCOUNT EVERY 1 MINUTES, RATE
NUMFILES 5000
DISCARDFILE ./dirrpt/ext_test.dsc, APPEND, MEGABYTES 100
DISCARDROLLOVER AT 2:00
WARNLONGTRANS 2h, CHECKINTERVAL 3m
EXTTRAIL ./dirdat/st, MEGABYTES 200
DYNAMICRESOLUTION
TRANLOGOPTIONS CONVERTUCS2CLOBS
TRANLOGOPTIONS RAWDEVICEOFFSET 0
DDL &
INCLUDE MAPPED OBJTYPE 'table' &
INCLUDE MAPPED OBJTYPE 'index' &
INCLUDE MAPPED OBJTYPE 'SEQUENCE' &
EXCLUDE OPTYPE COMMENT
DDLOPTIONS  NOCROSSRENAME  REPORT
TABLE     OGG_TEST.*;
SEQUENCE  OGG_TEST.*;

GETUPDATEBEFORES
```

2. 增加extract进程（ext后的名字要跟上面示例的示例组名对应，即extract）。

```
add ext extract,tranlog, begin now
```

3. 删除某废弃进程DP_TEST。

```
delete ext DP_TEST
```

4. 添加抽取进程，每个队列文件大小为200M。

```
add exttrail ./dirdat/st,ext extract, megabytes 200
```

5. 启动抽取进程（运行日志在ggate/dirrpt中）

```
start extract extract
```

说明：

Extract配置完成后，数据库的变更可以在*ggate/dirdat*目录下的文件中看到。

5. 生成def文件。

a. 启动源端ggsci后执行如下命令，生成defgen文件，并且拷贝到目标端dirdef下。

```
edit params defgen
DEFSFILE ./dirdef/ogg_test.def
USERID ogg_test, PASSWORD ogg_test
```

```
table OGG_TEST.*;
```

b. 在shell中执行如下命令，生成ogg_test.def。

```
./defgen paramfile ./dirprm/defgen.prm
```

6. 目标端OGG安装及配置。

a. 解压目标端OGG安装包。

b. 将源端中**dirdef/ogg_test.def**文件拷贝到目标端的dirdef下。

c. 启动ggsci后执行如下命令，创建必须目录。

```
create subdirs
```

d. 编辑mgr配置。

```
edit params mgr
PORT 7839
DYNAMICPORTLIST  7840-7849
PURGEOLDEXTRACTS ./dirdat/*, USECHECKPOINTS, MINKEEPDAYS 7
LAGREPORTHOURS 1
LAGINFOMINUTES 30
LAGCRITICALMINUTES 45
PURGEDDLHISTORY MINKEEPDAYS 3, MAXKEEPDAYS 7
PURGEMARKERHISTORY MINKEEPDAYS 3, MAXKEEPDAYS 7
```

e. 启动mgr。

```
start mgr
```

7. 源端OGG pump配置。

启动ggsci后执行如下操作。

1. 编辑pump配置。

```
edit params pump
EXTRACT pump
RMTHOST xx.xx.xx.xx, MGRPORT 7839, COMPRESS
PASSTHRU
NUMFILES 5000
RMTTRAIL ./dirdat/st
DYNAMICRESOLUTION
TABLE      OGG_TEST.*;
```

```
SEQUENCE   OGG_TEST.*;
```

2. 添加投递进程，从某一个队列开始投递。

```
add ext pump,exttrailsource ./dirdat/st
```

3. 投递进程，每个队文件大小为200m。

```
add rmttrail ./dirdat/st,ext pump,megabytes 200
```

4. 启动pump进程。

```
start pump
```

说明：

进程启动后，结合上面目标端OGG的配置，可以在目标端的dirdat目录下看到投递过来的trailfile。

8. DataHub插件安装和配置。

a. 配置环境变量*JAVA_HOME, LD_LIBRARY_PATH*，并将环境变量配置到~/.bash_profile中，参示例如下，仅供参考。

```
export JAVA_HOME=/xxx/xxx/jrexx
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:$JAVA_HOME/lib/amd64:$
JAVA_HOME/lib/amd64/server
```

b. 修改环境变量后，重启目标端OGG的mgr进程。

c. 下载*DataHub OGG*插件并解压。

d. 修改conf路径下的javaue.properties和log4j.properties文件，将{YOUR_HOME}替换为解压后的路径。

```
gg.handlerlist=ggdatahub
gg.handler.ggdatahub.type=com.aliyun.odps.ogg.handler.datahub.
DatahubHandler
gg.handler.ggdatahub.configureFileName={YOUR_HOME}/datahub-ogg-
plugin/conf/configure.xml
goldengate.userexit.nochkpt=false
goldengate.userexit.timestamp=utc
gg.classpath={YOUR_HOME}/datahub-ogg-plugin/lib/*
gg.log.level=debug
jvm.bootoptions=-Xmx512m -Dlog4j.configuration=file:{YOUR_HOME}/
datahub-ogg-plugin/conf/log4j.properties -Djava.class.path=ggjava/
ggjava.jar
```

e. 修改conf路径下的configure.xml文件，修改方式下面示例中的注释。

```
<?xml version="1.0" encoding="UTF-8"?>
<configue>

    <defaultOracleConfigure>
```

```
        <!-- oracle sid, 必选-->
        <sid>100</sid>
        <!-- oracle schema, 可以被mapping中的oracleSchema覆盖, 两者必
须有一个非空-->
        <schema>ogg_test</schema>
    </defaultOracleConfigure>

    <defalutDatahubConfigure>
        <!-- datahub endpoint, 必填-->
        <endPoint>YOUR_DATAHUB_ENDPOINT</endPoint>
        <!-- datahub project, 可以被mapping中的datahubProject, 两者
必须有一个非空-->
        <project>YOUR_DATAHUB_PROJECT</project>
        <!-- datahub accessId, 可以被mapping中的datahubAccessId覆
盖, 两者必须有一个非空-->
        <accessId>YOUR_DATAHUB_ACCESS_ID</accessId>
        <!-- datahub accessKey, 可以被mapping中的datahubAccessKey覆
盖, 两者必须有一个非空-->
        <accessKey>YOUR_DATAHUB_ACCESS_KEY</accessKey>
        <!-- 数据变更类型同步到datahub对应的字段，可以被columnMapping中
的ctypeColumn覆盖 -->
        <ctypeColumn>optype</ctypeColumn>
        <!-- 数据变更时间同步到datahub对应的字段，可以被columnMapping中
的ctimeColumn覆盖 -->
        <ctimeColumn>readtime</ctimeColumn>
        <!-- 数据变更序号同步到datahub对应的字段, 按数据变更先后递增, 不保
证连续, 可以被columnMapping中的cidColumn覆盖 -->
        <cidColumn>record_id</cidColumn>
    </defalutDatahubConfigure>

    <!-- 默认最严格，不落文件 直接退出 无限重试-->

    <!-- 运行每批上次的最多纪录数, 可选, 默认1000-->
    <batchSize>1000</batchSize>

    <!-- 默认时间字段转换格式, 可选, 默认yyyy-MM-dd HH:mm:ss-->
    <defaultDateFormat>yyyy-MM-dd HH:mm:ss</defaultDateFormat>

    <!-- 脏数据是否继续, 可选, 默认false-->
    <dirtyDataContinue>true</dirtyDataContinue>

    <!-- 脏数据文件, 可选, 默认datahub_ogg_plugin.dirty-->
    <dirtyDataFile>datahub_ogg_plugin.dirty</dirtyDataFile>

    <!-- 脏数据文件最大size, 单位M, 可选, 默认500-->
    <dirtyDataFileMaxSize>200</dirtyDataFileMaxSize>

    <!-- 重试次数, -1:无限重试 0:不重试 n:重试次数, 可选, 默认-1-->
    <retryTimes>0</retryTimes>

    <!-- 重试间隔, 单位毫秒, 可选, 默认3000-->
    <retryInterval>4000</retryInterval>

    <!-- 点位文件, 可选, 默认datahub_ogg_plugin.chk-->
    <checkPointFileName>datahub_ogg_plugin.chk</checkPointFileName
>
```

```
    <mappings>
        <mapping>
            <!-- oracle schema, 见上描述-->
            <oracleSchema></oracleSchema>
            <!-- oracle table, 必选-->
            <oracleTable>t_person</oracleTable>
            <!-- datahub project, 见上描述-->
            <datahubProject></datahubProject>
            <!-- datahub AccessId, 见上描述-->
            <datahubAccessId></datahubAccessId>
            <!-- datahub AccessKey, 见上描述-->
            <datahubAccessKey></datahubAccessKey>
            <!-- datahub topic, 必选-->
            <datahubTopic>t_person</datahubTopic>
            <ctypeColumn></ctypeColumn>
            <ctimeColumn></ctimeColumn>
            <cidColumn></cidColumn>
            <columnMapping>
                <!--
                src:oracle字段名称, 必须;
                dest:datahub field, 必须;
                destOld:变更前数据落到datahub的field, 可选;
                isShardColumn: 是否作为shard的hashkey, 可选, 默认为
false, 可以被shardId覆盖
                isDateFormat: timestamp字段是否采用DateFormat格式转
换, 默认true. 如果是false, 源端数据必须是long
                dateFormat: timestamp字段的转换格式, 不填就用默认值
                -->
                <column src="id" dest="id" isShardColumn="true"
isDateFormat="false" dateFormat="yyyy-MM-dd HH:mm:ss"/>
                <column src="name" dest="name" isShardColumn="true
"/>
                <column src="age" dest="age"/>
                <column src="address" dest="address"/>
                <column src="comments" dest="comments"/>
                <column src="sex" dest="sex"/>
                <column src="temp" dest="temp" destOld="temp1"/>
            </columnMapping>

            <!--指定shard id, 优先生效, 可选-->
            <shardId>1</shardId>
        </mapping>
    </mappings>
</configue>
```

f. 在ggsci下启动datahub writer。

```
edit params dhwriter
extract dhwriter
getEnv (JAVA_HOME)
getEnv (LD_LIBRARY_PATH)
getEnv (PATH)
CUSEREXIT ./libggjava_ue.so CUSEREXIT PASSTHRU INCLUDEUPD
ATEBEFORES, PARAMS "{YOUR_HOME}/datahub-ogg-plugin/conf/javaue.
properties"
sourcedefs ./dirdef/ogg_test.def
```

```
table OGG_TEST.*;
```

g. 添加dhwriter。

```
add extract dhwriter, exttrailsource ./dirdat/st
```

h. 启动dhwriter。

```
start dhwriter
```

使用案例

用户在Oracle数据库有一张商品订单表orders（oid int、pid int、num int），该表有三列，分别为订单ID、商品ID和商品数量。用户需要将这个表通过DataHub OGG进行增量数据同步，具体数据同步步骤如下。

说明：

进行增量数据同步前，需要先将源表已有的数据通过DataX同步到MaxCompute中。

1. 在DataHub上创建相应的Topic，Topic的schema如下所示。

```
string record_id, string optype, string readtime, bigint oid_before
, bigint oid_after, bigint pid_before, bigint pid_after, bigint
num_before, bigint num_after
```

2. DataHub OGG插件按照上述的安装流程进行部署配置，其中列的Mapping配置如下所示。

```
<ctypeColumn>optype</ctypeColumn>
            <ctimeColumn>readtime</ctimeColumn>
            <cidColumn>record_id</cidColumn>
            <columnMapping>
                <column src="oid" dest="oid_after" destOld="
oid_before" isShardColumn="true"/>
                <column src="pid" dest="pid_after" destOld="
pid_before"/>
                <column src="num" dest="num_after" destOld="
num_before"/>
            </columnMapping>
```

说明：

其中optype和readtime字段是记录数据库的数据变更类型和时间，optype有“I”、“D”、“U”三种取值，分别对应为“增”、“删”、“改”三种数据变更操作。

3. DataHub OGG插件部署完成并成功运行后，插件会源源不断的将源表的数据变更记录输送至datahub中。

11.3.3 DataHub数据归档

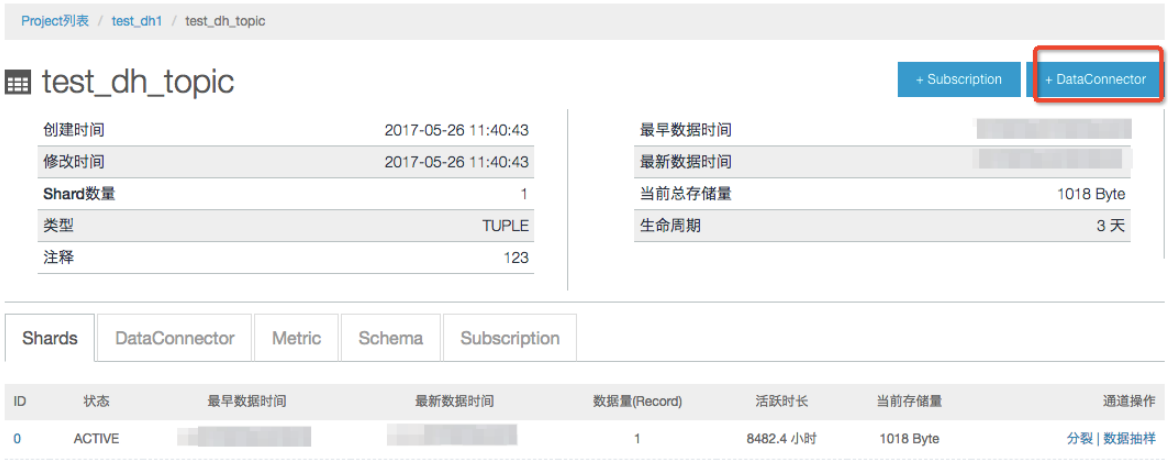
在DataHub WebConsole中，提供的数据归档功能可以将DataHub中的实时数据同步到离线数据仓库，例如：MaxCompute中，方便用户对历史数据进行分析处理。

11.3.3.1 归档MaxCompute

11.3.3.1.1 创建归档

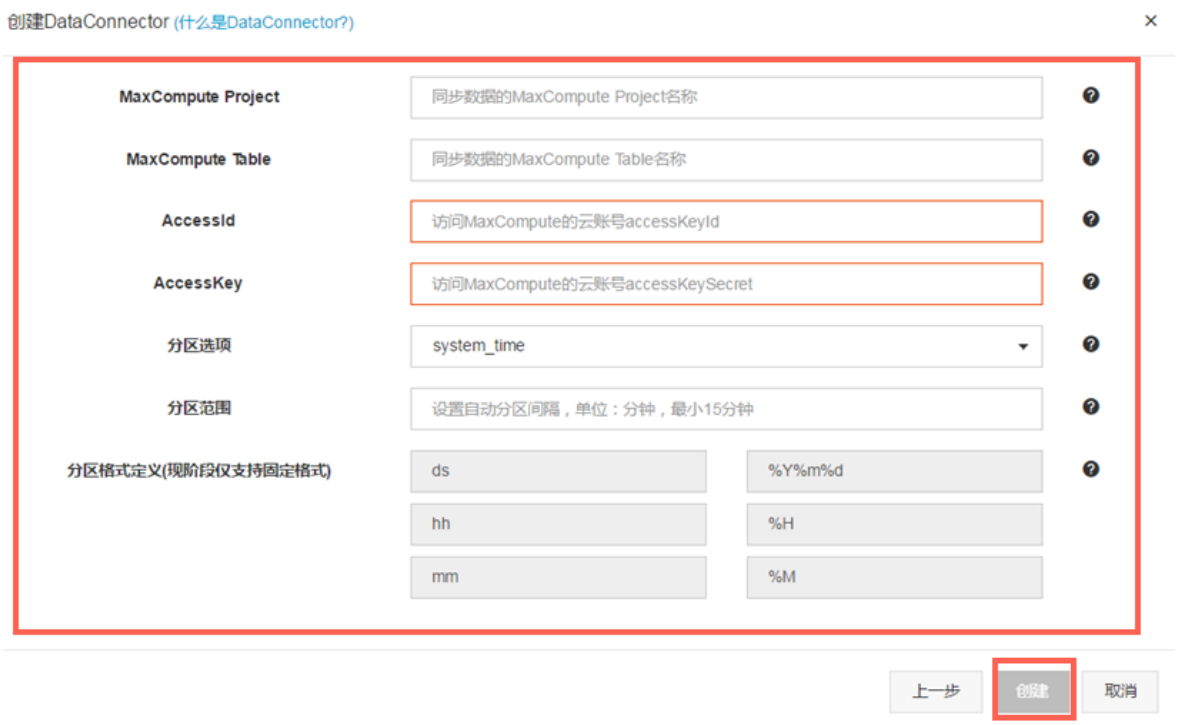
1. 登录实时数据分发平台控制台。
2. 在默认进入的**项目管理**页签，按先后顺序，分别单击Project列表后及Topic列表后的**查看**，进入Topic详情页面。
3. 在Topic详情页面中，单击**+DataConnector**并选择**MaxCompute**，进入创建**MaxCompute DataConnector**页面。

图 11-16: +DataConnector



4. 在创建DataConnector页面，配置相关信息后，单击**创建**，完成创建归档操作。

图 11-17: 配置相关信息



说明：

相关配置信息的详情如下所示。

- MaxCompute Project：指定Topic需要同步到的MaxCompute project名称。
- MaxCompute Table：指定Topic需要同步到的MaxCompute table名称。
- AccessId/AccessKey：访问MaxCompute所需要的ak对，该ak必须是子用户ak，并且具有指定的MaxCompute table下CreateInstance、Desc、Alter权限。
- 分区选项：分为SYSTEM_TIME、EVENT_TIME、USER_DEFINE三种模式。其中，SystemTime模式会使用写入时间转化为字符串进行分区；EventTime模式会根据topic中固定的event_time字段时间进行分区（需要在创建Topic时增加一个TIMESTAMP类型，名称为event_time的字段，并且写入数据时向这个字段写入其微秒时间）；UserDefine模式将会直接使用用户自定义的分区字段字符串分区。
- 分区范围：划分分区的时间间隔，在SYSTEM_TIME、EVENT_TIME两种模式下生效，最短为15分钟。
- 分区格式定义：目前仅支持固定格式，未来将会开放为自定义格式。目前格式下，若为15分钟的分区范围，则会产生*ds=20170704,hh=01,mm=15*这样的分区。

11.3.3.1.2 查看归档详情

归档详情提供归档任务的基本信息，包括DataHub的Topic信息、需要归档到的MaxCompute项目和表的信息以及整体归档进度信息（包括最新写入时间、已归档时间和归档延迟）。

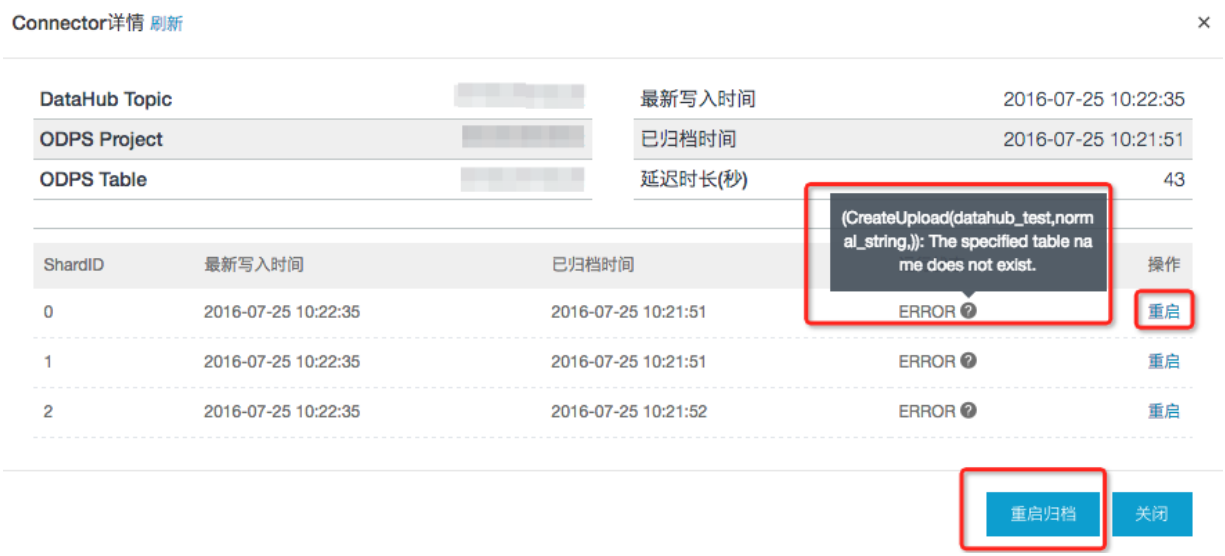
说明：

最新写入时间表示指定Topic中最新的写入记录的系统时间；已归档时间表示在该时间之前的记录已全部归档；延迟时长为最新写入时间和已归档时间的差值，单位为秒。

图 11-18: 归档详情--概况

归档详情中可以查看该DataConnector归档任务中每个shard的归档状况。当运行状态为**ERROR**时，鼠标移动到帮助图标会显示该shard归档任务终止的详细报错信息，并且该shard的重启功能可用，用户可根据报错信息进行修复。例如：因为删除MaxCompute partition导致的任务终止，可以通过MaxCompute控制台、SDK等创建Partition，修复完成后，单击该shard的**重启**，该shard的归档任务将重新被调度，并且从上次终止的记录开始进行归档。若所有shard或者大部分shard处于非**RUNNING**状态，可单击**重启归档**，重启所有shard的归档任务。

图 11-19: 归档详情--shard



11.3.4 DataHub统计信息

DataHub在WebConsole中向用户提供Topic级别的统计信息，用户可以通过Metric界面查看准实时的Topic级别的QPS和流量等信息。目前提供的指标有：

- 读写request/second。
- 读写record/second。
- 读写throughput/second（单位KB）。
- 读写请求latency/request（单位微秒）。

进入Topic详情页面后，单击**Metric**，查看Metric统计信息，如下图所示。

图 11-20: Metric统计信息

DataHub还支持可以通过指定时间段，读取指定时间内的统计信息。目前仅支持一次最多查询4个小时内的统计信息，暂不支持一次查询中的时间跨天。如下图所示，选定一个时间将会显示选定时间之前的n小时内的metric信息。

图 11-21: 指定时间的Metric统计信息