阿里云
MaxCompute

用户指南

文档版本：20180806

法律声明

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

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

通用约定

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。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	**说明：** 您也可以通过按**Ctrl** + **A**选中全部文件。
>	多级菜单递进。	设置 > 网络 > 设置网络类型
粗体	表示按键、菜单、页面名称等UI元素。	单击 **确定**。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *`Instance_ID`*
[]或者[a|b]	表示可选项，至多选择一个。	`ipconfig` *`[-all|-t]`*
{}或者{a|b}	表示必选项，至多选择一个。	`swich` *`{stand | slave}`*

目录

法律声明.....I
通用约定.....I
1 数据上传下载.....1
1.1 数据上传/下载概述.....1
1.2 数据上云场景.....1
1.3 工具介绍.....2
1.4 Tunnel命令操作.....4
1.5 通过数据集成导入/导出.....14
1.6 批量数据通道SDK介绍.....21
1.6.1 批量数据通道概要.....21
1.6.2 TableTunnel.....22
1.6.3 UploadSession.....24
1.6.4 DownloadSession.....25
1.6.5 TunnelBufferedWriter.....27
1.7 批量数据通道SDK示例.....28
1.7.1 示例概述.....28
1.7.2 简单上传示例.....28
1.7.3 简单下载示例.....30
1.7.4 多线程上传示例.....32
1.7.5 多线程下载示例.....34
1.7.6 BufferedWriter多线程上传示例.....37
1.7.7 BufferedWriter上传示例.....38
1.8 DataHub实时数据通道.....38
1.9 数据通道服务连接.....38
2 常用命令.....39
2.2 项目空间操作.....39
2.3 表操作.....40
2.4 实例操作.....44
2.5 资源操作.....49
2.6 函数操作.....50
2.7 其他操作.....52
3 SQL.....57
3.1 SQL概述.....57
3.2 运算符.....58
3.3 类型转换.....61
3.4 DDL语句.....67
3.4.1 表操作.....67

3.4.2 生命周期操作....73
3.4.3 视图操作....74
3.4.4 分区/列操作....75
3.5 SELECT操作....79
3.6 INSERT操作....79
3.7 SELECT操作....79
3.8 SQL限制项汇总....79
3.9 内建函数....80
3.9.1 日期函数....80
3.9.2 数学函数....96
3.9.3 窗口函数....116
3.9.4 聚合函数....131
3.9.5 其他函数....144
3.9.6 字符串函数....163
3.10 UDF....184
3.10.2 Java UDF....184
3.10.3 Python UDF....196
3.11 附录....202
3.11.1 转义字符....202
3.11.2 LIKE字符匹配....203
3.11.3 正则表达式规范....204
3.11.4 保留字....206
4 MapReduce....208
4.1 概要....208
4.1.2 扩展MapReduce....208
4.1.3 开源兼容MapReduce....208
4.2 功能介绍....213
4.2.2 基本概念....213
4.2.3 输入与输出....214
4.2.4 资源使用....215
4.2.5 本地运行....215
4.3 示例程序....217
4.3.1 WordCount示例....218
4.3.2 MapOnly示例....220
4.3.3 多路输入输出示例....223
4.3.4 多任务示例....227
4.3.5 二次排序示例....229
4.3.6 使用资源示例....232
4.3.7 使用Counter示例....235
4.3.8 Grep示例....237

4.3.9 Join示例....241
4.3.10 Sleep示例....244
4.3.11 Unique示例....245
4.3.12 Sort示例....248
4.3.13 分区表输入示例....251
4.3.14 Pipeline示例....252
4.4 Java SDK....255
4.4.1 原生SDK概述....255
4.5 MR限制项汇总....261
5 Java沙箱....264
6 SDK....269
6.1 Java SDK....269
6.2 Python SDK....274
7 处理非结构化数据....290
7.1 访问 OSS 非结构化数据....290
7.2 访问OTS非结构化数据....302
8 图模型....308
8.1 图模型概述....308
8.2 功能概述....311
8.3 SDK概述....314
8.4 开发和调试....315
8.5 应用限制....322
8.6 示例程序....323
8.6.1 单源最短距离....323
8.6.2 PageRank....326
8.6.3 K-均值聚类....329
8.6.4 BiPartiteMatchiing....334
8.6.5 强连通分量....337
8.6.6 连通分量....345
8.6.7 拓扑排序....347
8.6.8 线性回归....350
8.6.9 三角形计数....355
8.6.10 输入点表....357
8.6.11 输入边表....363
8.7 Aggregator机制概述....369
9 安全指南....377
9.1 目标用户....377
9.2 快速开始....377
9.2.1 添加用户并授权....377

9.2.2 添加角色并通过ACL授权.....377
9.2.3 设置项目保护模式.....378
9.3 用户及授权管理.....378
9.5 用户管理.....378
9.6 角色管理.....383
9.7 授权.....385
9.8 权限查看.....388
9.9 项目空间的安全配置.....390
9.10 安全相关语句汇总.....391
9.10.1 项目空间的安全配置.....391
9.10.2 项目空间的权限管理.....391
9.11 跨项目空间的资源分享.....393
9.11.1 基于Package的跨项目空间的资源分享.....393
9.11.2 Package的使用方法.....393
9.12 列级别访问控制.....396

1 数据上传下载

1.1 数据上传/下载概述

本文对MaxCompute系统数据的上传和下载进行概述，包括服务连接、SDK、工具和数据上云场景。

总的来说，您可以通过DataHub实时数据通道和Tunnel批量数据通道两种途径进出MaxCompute系统。DataHub和Tunnel各自都提供了SDK，而基于这些SDK又衍生了许多用于数据上传/下载的工具，方便您在各种场景下的数据进行上传/下载的需求。

数据上传/下载的工具主要包括：DataWorks、DTS、OGG插件、Sqoop、Flume插件、LogStash插件、Fluentd插件、Kettle插件以及MaxCompute客户端等。

上述工具使用的底层数据通道，分类如下：

- DataHub通道系列
  - OGG插件
  - Flume插件
  - LogStash插件
  - Fluentd插件
- Tunnel通道系列
  - DataWorks
  - DTS
  - Sqoop
  - Kettle插件
  - MaxCompute客户端

基于上述丰富的数据上传/下载的工具，可以满足大部分常见的数据上云场景，后续的章节会对工具本身以及Hadoop数据迁移、数据库数据同步、日志采集等数据上云的场景进行介绍，为您进行技术方案选型时提供参考。

1.2 数据上云场景

利用MaxCompute平台的数据上传/下载工具，可以广泛用于各种数据上云的应用场景，本文将介绍几种常见的经典场景。

Hadoop数据迁移

Hadoop数据迁移有两种可选的工具，分别是Sqoop和DataWorks。

- Sqoop执行时，会在原来的Hadoop集群上执行MR作业，可以分布式地将数据传输到MaxCompute上，效率会比较高，详情请参见Sqoop工具的介绍。
- 使用DataWorks结合DataX进行Hadoop数据迁移的示例请参见*Hadoop*数据迁移新手教程。

数据库数据同步

数据库数据同步到MaxCompute需要根据数据库的类型和同步策略来选择相应的工具。

- 离线批量的数据库数据同步：可以选择DataWorks，支持的数据库种类比较丰富，有MySQL、SQL Server、PostgreSQL等，详情请参见数据同步简介，您也可以参见 创建同步任务进行实例操作。
- Oracle数据库数据实时同步时，可以选择OGG插件工具。
- RDS数据库数据实时同步时，可以选择DTS同步。

日志采集

日志采集时，您可以选用Flume、Flunted、LogStash等工具。具体场景示例请参见*Flume*收集网站日志数据到*MaxCompute*和海量日志数据分析与应用。

1.3 工具介绍

MaxCompute平台的数据上传和下载目前有着丰富的工具（其中大部分已经在GitHub上开源，走开源社区的维护方式）可以使用，各自有不同的应用场景，具体分为阿里云数加产品和开源产品两大类，本文将进行简要介绍。

阿里云数加产品

- **DataWorks之数据集成**

  DataWorks之数据集成（即数据同步），是阿里集团对外提供的稳定高效、弹性伸缩的数据同步平台，致力于为阿里云上各类异构数据存储系统提供离线全量和实时增量的数据同步、集成、交换服务。

  其中数据同步任务支持的数据源类型包括：MaxCompute、RDS（MySQL、SQL Server、PostgreSQL）、Oracle、FTP、ADS（AnalyticDB）、OSS、Memcache和DRDS，详情请参见数据同步简介，具体使用方法请参见创建数据同步任务。

- **MaxCompute客户端**

- 客户端的安装和基本使用请参见客户端介绍。
- 客户端基于批量数据通道的SDK实现了内置的tunnel命令，可对数据进行上传和下载，详情请参见*tunnel*命令的基本使用介绍。

> 说明：
> 该项目已经开源，您可进入*aliyun-odps-console*进行查看 。

- **DTS**

  数据传输*#Data Transmission#*服务*DTS*是阿里云提供的一种支持RDBMS（关系型数据库）、NoSQL、OLAP等多种数据源之间数据交互的数据服务。它提供了数据迁移、实时数据订阅及数据实时同步等多种数据传输功能。

  DTS可以支持RDS、MySQL实例的数据实时同步到MaxCompute表中，暂不支持其他数据源类型。详情请参见创建*RDS*到*MaxCompute*数据实时同步作业。

开源产品

- **Sqoop**

  Sqoop基于社区Sqoop 1.4.6版本开发，增强了对MaxCompute的支持，可以将数据从MySQL等关系数据库导入/导出到MaxCompute表中，也可以从HDFS/Hive导入数据到MaxCompute表中。详情请参见*MaxCompute Sqoop*。

> 说明：
> 该项目已经开源，您可进入*aliyun-maxcompute-data-collectors*进行查看。

- **Kettle**

  Kettle是一款开源的ETL工具，纯Java实现，可以在Windows、Unix和Linux上运行，提供图形化的操作界面，可以通过拖拽控件的方式，方便地定义数据传输的拓扑。

> 说明：
> 该项目已经开源，您可进入*aliyun-maxcompute-data-collectors*进行查看。

- **Flume**

  Apache Flume是一个分布式的、可靠的、可用的系统，可高效地从不同的数据源中收集、聚合和移动海量日志数据到集中式数据存储系统，支持多种Source和Sink插件。

Apache Flume的DataHub Sink插件可以将日志数据实时上传到DataHub，并归档到MaxCompute表中。详情请参见*flume_plugin*。

说明：

该项目已经开源，您可进入*aliyun-maxcompute-data-collectors*进行查看。

- **Fluentd**

  Fluentd是一个开源的软件，用来收集各种源头日志（包括Application Log、Sys Log及Access Log），允许您选择插件对日志数据进行过滤，并存储到不同的数据处理端（包括MySQL、Oracle、MongoDB、Hadoop、Treasure Data等）。

  Fluentd的DataHub插件可以将日志数据实时上传到DataHub，并归档到MaxCompute表中。

- **LogStash**

  LogStash是一款开源日志收集处理框架，logstash-output-datahub插件实现了将数据导入DataHub的功能。通过简单的配置即可完成数据的采集和传输，结合MaxCompute/StreamCompute可以轻松构建流式数据从采集到分析的一站式解决方案。

  LogStash的DataHub插件可以将日志数据实时上传到DataHub，并归档到MaxCompute表中。

- **OGG**

  OGG的DataHub插件可以支持将Oracle数据库的数据实时地以增量方式同步到DataHub中，并最终归档到MaxCompute表中。

说明：

该项目已经开源，您可进入*aliyun-maxcompute-data-collectors*进行查看。

1.4 Tunnel命令操作

功能简介

您可以通过客户端提供的*Tunnel*命令实现原有Dship工具的功能。

Tunnel命令，主要用于数据的上传和下载等功能。主要功能如下：

- **Upload**：支持文件或目录（指一级目录）的上传，每一次上传只支持数据上传到一张表或表的一个分区，有分区的表一定要指定上传的分区，多级分区一定要指定到末级分区。

```
tunnel upload log.txt test_project.test_table/p1="b1",p2="b2";
-- 将log.txt中的数据上传至项目空间test_project的表test_table（二级分区表）中的p1="b1",p2="b2"分区
```

```
tunnel upload  log.txt  test_table --scan=only;
-- 将log.txt中的数据上传至表 test_table 中。--scan参数表示需要扫描log.txt中的数据是否符合 test_table 的定义，如果不符合报错，并停止上传数据。
```

- **Download**：只支持下载到单个文件，每一次下载只支持下载一张表或一个分区到一个文件，有分区的表一定要指定下载的分区，多级分区一定要指定到末级分区。

```
tunnel download  test_project.test_table/p1="b1",p2="b2"  test_table.txt;
-- 将test_project.test_table表（二级分区表）中的数据下载到 test_table.txt 文件中
```

- **Resume**：因为网络或tunnel服务的原因出错，支持文件或目录的续传。可以继续上一次的数据上传操作，但Resume命令暂时没有对下载操作的支持。

```
tunnel resume;
```

- **Show**：显示历史任务信息。

```
tunnel show history -n 5;
--显示前5次上传/下载数据的详细命令
tunnel show log;
--显示最后一次上传/下载数据的日志
```

- **Purge**：清理session目录，默认清理3天内的。

```
tunnel purge 5;
--清理前5天的日志
```

Tunnel命令使用说明

Tunnel命令支持在客户端通过help子命令获取帮助信息，每个命令和选择支持短命令格式。

```
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**：上传数据到MaxCompute的表中。
- **download**：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。子命令使用提示：

```
odps@ project_name>tunnel help upload;
usage: tunnel upload [options] <path> <[project.]table[/partition]>
               upload data from local file
 -acp,-auto-create-partition <ARG>   auto create target partition if
not
                                     exists, default false
 -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 "\r\n
"
 -s,-scan <ARG>                      specify scan file
                                     action(true|false|only), default
true
 -sd,-session-dir <ARG>              set session dir, default
                                     D:\software\odpscmd_public\
plugins\ds
                                     hip
 -ss,-strict-schema <ARG>            specify strict schema mode. If
false,
                                     extra data will be abandoned and
                                     insufficient field will be filled
                                     with null. Default true
 -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：

- **-acp**：如果不存在，自动创建目标分区，默认关闭。
- **-bs**：每次上传至Tunnel的数据块大小，默认100MiB（MiB = 1024*1024B ）。
- **-c**：指定本地数据文件编码，默认为UTF-8。不设定，默认下载源数据。
- **-cp**：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时，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**：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，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时，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**：设置session目录。
- **-te**：指定tunnel的Endpoint。
- **-threads**：指定threads的数量，默认为1。
- **-tz**：指定时区。默认为本地时区：Asia/Shanghai。

示例如下：

- 创建目标表，如下所示：

```
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，如下所示：

```
alter table sale_detail add partition (sale_date='201312', region='
hangzhou');
```

- 准备数据文件data.txt，其内容如下所示：

```
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。

- 导入数据，如下所示：

```
odps@ project_name>tunnel u d:\data.txt sale_detail/sale_date=201312
,region=hangzhou -s false
Upload session: 20150610163922488087 0a002ec60c
Start upload:d:\data.txt
Total bytes:41   Split input to 1 blocks
2015-06-10 16:39:22     upload block: '1'
ERROR: column mismatch -,expected 3 columns, 2 columns found, please
 check data or delimiter
```

由于data.txt中有脏数据，数据导入失败。并给出session id及错误提示信息。

- 数据验证，如下所示：

```
odps@ odpstest_ay52c_ay52> select * from sale_detail where sale_date
='201312';
ID = 20150610084135370gyvc61z5
+-----------+-------------+-------------+-----------+--------+
| shop_name | customer_id | total_price | sale_date | region |
+-----------+-------------+-------------+-----------+--------+
+-----------+-------------+-------------+-----------+--------+
```

由于有脏数据，数据导入失败，表中无数据。

Show

显示历史记录。子命令使用提示：

```
odps@ project_name>tunnel help show;
usage: tunnel show history [options]
              show session information
 -n,-number <ARG>   lines
Example:
    tunnel show history -n 5
    tunnel show log
```

参数说明：

-n：指定显示行数。

示例如下：

```
odps@ project_name>tunnel show  history;
20150610163922488087 0a002ec60c  failed  'u --config-file /D:/console
/conf/odps_config.ini --project odpstest_ay52c_ay52 --endpoint http
://service.odps.aliyun.com/api --id UlVxOHuthHV1QrI1 --key 2m4r3WvTZb
sNJjybVXj0InVke7UkvR d:\data.txt sale_detail/sale_date=201312,region=
hangzhou -s false'
```

说明：

20150610163922488087 0a002ec60c是上节中导入数据失败时的运行ID。

Resume

修复执行历史记录，仅对上传数据有效。子命令使用提示：

```
odps@  project_name>tunnel help resume;
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文件为如下内容：

```
shop9,97,100
shop10,10,200
```

修复执行上传数据，如下所示：

```
odps@ project_name>tunnel resume 20150610163922488087 0a002ec60c --
force;
start resume
20150610163922488087 0a002ec60c
Upload session: 20150610163922488087 0a002ec60c
Start upload:d:\data.txt
Resume 1 blocks
2015-06-10 16:46:42     upload block: '1'
2015-06-10 16:46:42     upload block complete, blockid=1
upload complete, average speed is 0 KB/s
OK
```

说明：

20150610163922488087 0a002ec60c为上传失败的session ID。

数据验证，如下所示：

```
odps@ project_name>select * from sale_detail where sale_date='201312';
 ID = 20150610084801405g0a741z5
```

```
+-----------+-------------+-------------+-----------+--------+
| shop_name | customer_id | total_price | sale_date | region |
+-----------+-------------+-------------+-----------+--------+
| shop9     | 97          | 100.0       | 201312    | hangzhou |
| shop10    | 10          | 200.0       | 201312    | hangzhou |
+-----------+-------------+-------------+-----------+--------+
```

Download

子命令使用提示：

```
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
 -ci,-columns-index <ARG>           specify the columns index(starts
from
                                    0) to download, use comma to split
each
                                    index
 -cn,-columns-name <ARG>            specify the columns name to
download,
                                    use comma to split each name
 -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 "\r\n"
 -sd,-session-dir <ARG>             set session dir, default
                                    D:\software\odpscmd_public\plugins\
dshi
                                    p
 -te,-tunnel_endpoint <ARG>         tunnel endpoint
    -threads <ARG>                  number of threads, default 1
 -tz,-time-zone <ARG>               time zone, default local timezone:
                                    Asia/Shanghai
usage: tunnel download [options] instance://<[project/]instance_id> <
path>
              download instance result to local file
 -c,-charset <ARG>                  specify file charset, default
ignore.
                                    set ignore to download raw data
 -ci,-columns-index <ARG>           specify the columns index(starts
from
```

```
                                    0) to download, use comma to split
each
                                    index
 -cn,-columns-name <ARG>            specify the columns name to
download,
                                    use comma to split each name
 -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 "\r\n"
 -sd,-session-dir <ARG>             set session dir, default
                                    D:\software\odpscmd_public\plugins\
dshi
                                    p
 -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
    tunnel download instance://test_project/test_instance log.txt
```

参数说明：

- **-c**：本地数据文件编码，默认为UTF-8。
- **-ci**：指定列索引（从0）下载，使用逗号分隔。
- **-cn**：指定要下载的列名称，使用逗号分隔每个名称。
- **-cp**，**-compress**：指定是否压缩后再下载，减少网络流量，默认开启。
- **-dfp**：DateTime类型数据格式，默认为yyyy-MM-dd HH:mm:ss。
- **-e**：当下载double值时，如果需要，使用指数函数表示，否则最多保留20位。
- **-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下载数据。

> 说明：
>
> -h=true和threads>1即多线程不能一起使用。

- **-limit**：指定要下载的文件数量。
- **-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，默认为\r\n。
- **-sd**：设置session目录。
- **-te**：指定tunnel endpoint。
- **-threads**：指定threads的数量，默认为1。
- **-tz**：指定时区。默认为本地时区：Asia/Shanghai。

示例如下：

下载数据到*result.txt*文件中，如下所示：

```
$ ./tunnel download sale_detail/sale_date=201312,region=hangzhou
result.txt;
    Download session: 20150610165824528387 0a002ed0b9
    Total records: 2
    2015-06-10 16:58:24     download records: 2
    2015-06-10 16:58:24     file size: 30 bytes
    OK
```

验证*result.txt*的文件内容，如下所示：

```
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。子命令使用提示：

```
odps@ project_name>tunnel help purge;
usage: tunnel purge [n]
            force session history to be purged.([n] days before,
default
            3 days)
Example:
    tunnel purge 5
```

数据类型说明：

类型	描述
STRING	字符串类型，长度不能超过8MB。
BOOLEN	上传值只支持”true”，”false”，”0”，”1”。下载值为true/false且不区分大小写。
BIGINT	取值范围[-9223372036854775807，9223372036854775807]。

类型	描述
DOUBLE	1.有效位数16位 2.上传支持科学计数法表示 3.下载仅使用数字表示 4.最大值：1.7976931348623157E308 5.最小值：4.9E-324 6.无穷大：Infinity 7.无穷小：-Infinity
DATETIME	Datetime类型默认支持时区为GMT+8的数据上传，可以通过命令行指定用户数据日期格式的format pattern。

如果您上传DATETIME类型的数据，需要指定时间日期格式，具体格式请参见SimpleDateFormat。

```
"yyyyMMddHHmmss": 数据格式"20140209101000"
"yyyy-MM-dd HH:mm:ss"(默认)：数据格式"2014-02-09 10:10:00"
"yyyy年MM月dd日": 数据格式"2014年09月01日"
```

示例如下：

```
tunnel upload  log.txt  test_table -dfp  "yyyy-MM-dd HH:mm:ss"
```

空值：所有数据类型都可以有空值。

- 默认空字符串为空值。
- 可在命令行下通过**-null-indicator**参数来指定空值的字符串。

```
tunnel upload log.txt test_table -ni "NULL"
```

字符编码：您可以指定文件的字符编码，默认为UTF-8。

```
tunnel upload log.txt test_table -c "gbk"
```

分隔符：tunnel命令支持您自定义的文件分隔符，行分隔符选项为-record-delimiter，列分隔符选项为-field-delimiter。

分隔符说明如下：

- 支持多个字符的行列分隔符。
- 列分隔符不能够包含行分隔符。
- 转义字符分隔符，在命令行方式下只支持\r、\n和\t。

示例如下：

```
tunnel upload log.txt test_table -fd "||" -rd "\r\n"
```

1.5 通过数据集成导入/导出

您可以通过DataWorks中的数据集成创建数据同步任务，对MaxCompute 中的数据进行导入和导出操作。

前置条件

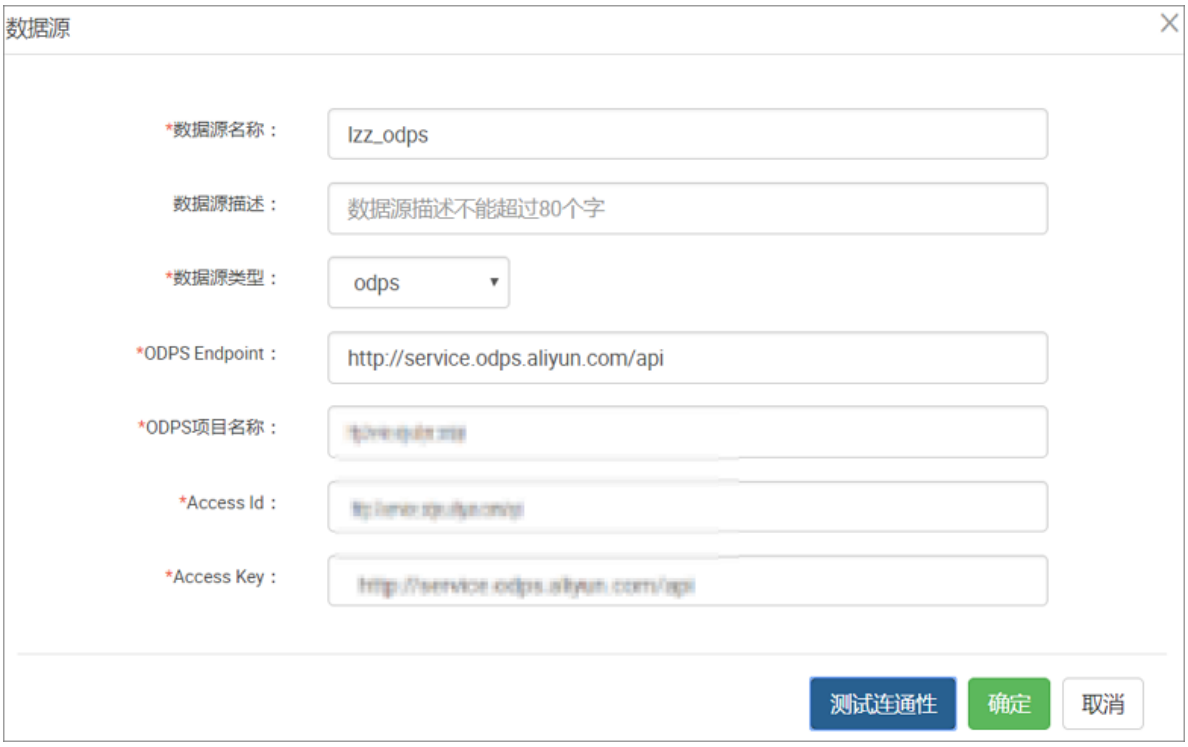
在开始数据的导入和导出操作前，您需要根据准备阿里云账号和购买并创建项目中的操作，做好准备工作。

添加MaxCompute数据源

> 说明：
>
> - 只有项目管理员角色才能够新建数据源，其他角色的成员仅能查看数据源。
> - 若需要添加的数据源是当前MaxCompute项目，则无需进行此操作。该项目创建成功时，即在数据集成的数据源中，默认将该项目添加为数据源，数据源名为odps_first。

操作步骤

1. 以项目管理员身份进入*DataWorks*管理控制台，单击项目列表下对应项目操作栏中的进入工作区。
2. 单击顶部菜单栏中的数据集成，导航至数据源页面。
3. 单击新增数据源。
4. 在新增数据源弹出框中填写相关配置项，如下图所示：



配置项说明如下：

- **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60个字符。
- **数据源描述**：对数据源进行简单描述，不得超过80个字符。
- **数据源类型**：当前选择的数据源类型MySQL。
- **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。
- **Access Id**：与MaxCompute Project Owner云账号对应的AccessID。
- **Access Key**：与MaxCompute Project Owner云账号对应的AccessKey，与AccessID成对使用。

5. 单击测试连通性。
6. 若测试连通性成功，单击保存即可。

> **说明**：
> 其他的数据源的配置请参见数据源配置。

通过数据集成导入数据

以将MySQL的数据导入到MaxCompute中为例，您可以通过向导模式和脚本模式两种方式配置同步任务。

向导模式配置同步任务

1. 新建向导模式的同步任务，如下图所示：

2. 选择来源。

   选择MySQL数据源及源头表mytest，数据浏览默认是收起的，选择后单击下一步，如下图所示：

3. 选择目标。

目标即MaxCompute（原ODPS），表可以是提前创建，也可以在此处单击**快速建表**，如下图所示：

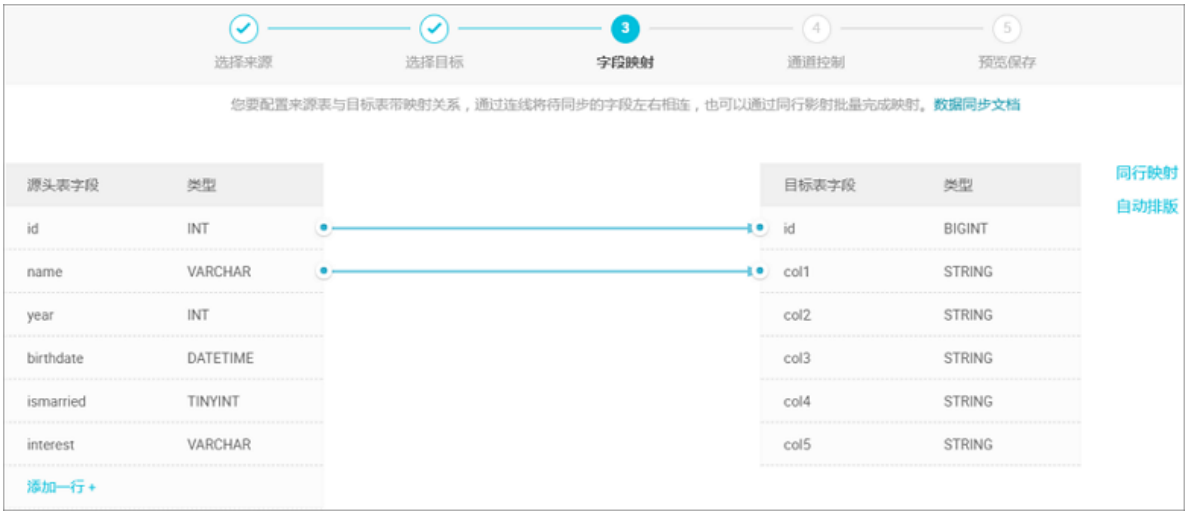
配置项说明：

- **分区信息**：必须指定到最后一级分区。例如把数据写入一个三级分区表，必须配置到最后一级分区，例如pt=20150101，type=1，biz=2。非分区表无此项配置。
- **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：导数据之前不清理任何数据，每次运行数据都是追加进去的，相当于insert into。

4. 映射字段。

选择字段的映射关系。需对字段映射关系进行配置，左侧源头表字段和右侧目标表字段为一一对应的关系。



5. 通道控制。

单击下一步，配置作业速率上限和脏数据检查规则。如下图所示：

配置项说明如下：

- 作业速率上限：是指数据同步作业可能达到的最高速率，其最终实际速率受网络环境、数据库配置等的影响。
- 作业并发数：从单同步作业来看，作业并发数*单并发的传输速率=作业传输总速率。

当作业速率上限已选定的情况下，应该如何选择作业并发数？

- 如果您的数据源是线上的业务库，建议您不要将并发数设置过大，以防对线上库造成影响。
- 如果您对数据同步速率特别在意，建议您选择最大作业速率上限和较大的作业并发数。

6. 预览保存。

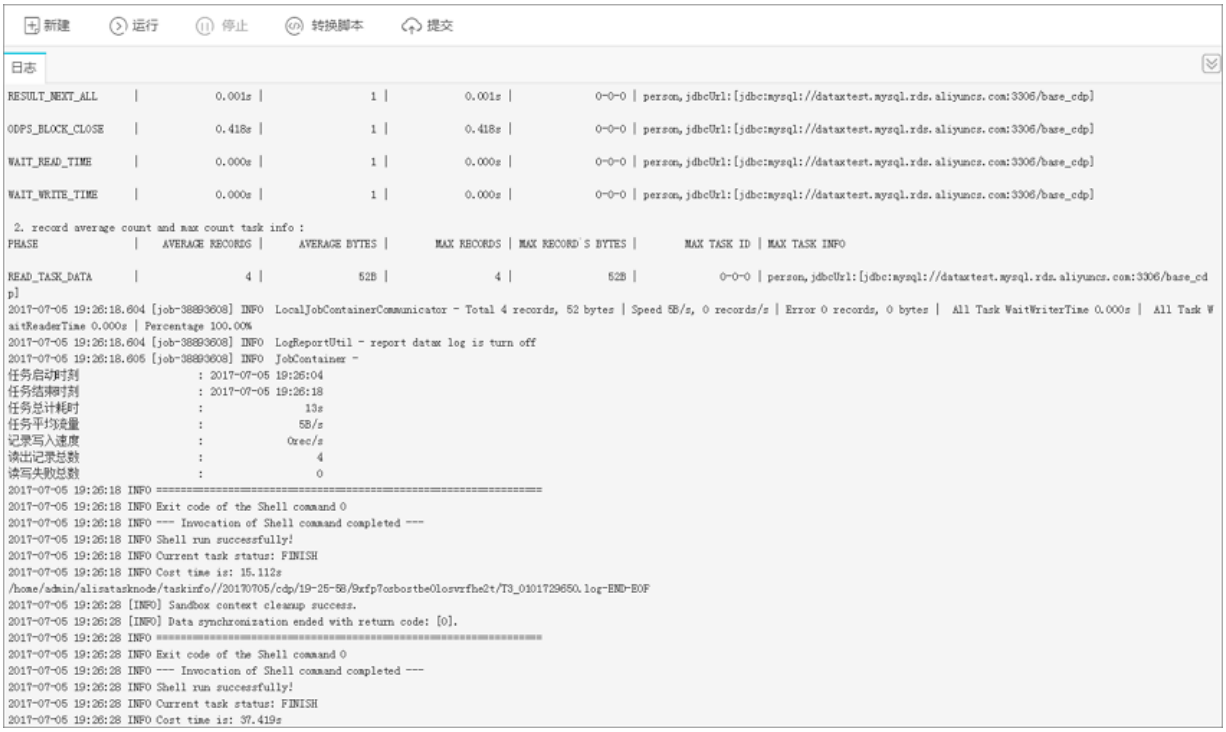
   完成上述配置后，上下滚动鼠标可查看任务配置，如若无误，单击保存，如下图所示：

运行同步任务

直接运行同步任务

只有在同步任务设置了系统变量参数，在运行时才会自动弹出弹框配置参数变量。如下图所示：

运行同步任务日志情况，如下图所示：



同步任务保存后，直接单击**运行**，任务便会立刻运行。您也可以单击**提交**，将同步任务提交到DataWorks的调度系统中，调度系统会按照配置属性在从第二天开始自动定时执行，相关调度的配置请参见调度配置介绍。

脚本模式配置同步任务

您可参考以下脚本进行配置同步任务操作，其他配置与任务运行同**向导模式**方式。

```
{
  "type": "job",
  "version": "1.0",
  "configuration": {
    "reader": {
      "plugin": "mysql",
      "parameter": {
        "datasource": "mysql",
        "where": "",
        "splitPk": "id",
        "connection": [
          {
            "table": [
              "person"
            ],
            "datasource": "mysql"
          }
        ],
        "connectionTable": "person",
        "column": [
          "id",
          "name"
        ]
```

```
      }
    },
    "writer": {
      "plugin": "odps",
      "parameter": {
        "datasource": "odps_first",
        "table": "a1",
        "truncate": true,
        "partition": "pt=${bdp.system.bizdate}",
        "column": [
          "id",
          "col1"
        ]
      }
    },
    "setting": {
      "speed": {
        "mbps": "1",
        "concurrent": "1"
      }
    }
  }
}
```

参考文档

- 不同类型数据源 *Reader* 的配置。
- 不同类型数据源 *Writer* 的配置。

1.6 批量数据通道SDK介绍

1.6.1 批量数据通道概要

MaxCompute Tunnel是MaxCompute的数据通道，您可以通过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）数据的上传和下载。

MaxCompute提供的数据上传和下载工具即是基于Tunnel SDK编写的。

如果您使用Maven，可以从*Maven*库中搜索**`odps-sdk-core`** 来获取不同版本的Java SDK，相关配置信息如下所示：

```
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-sdk-core</artifactId>
    <version>0.24.0-public</version>
</dependency>
```

本文将为您介绍Tunnel SDK的主要接口，不同版本的SDK在使用上有所差别，准确信息以*SDK Java Doc*为准。

主要接口	描述
TableTunnel	访问 MaxCompute Tunnel 服务的入口类。 您可以通过公网或者阿里云内网环境对 MaxCompute 及其 Tunnel 进行访问。当您在阿里云内网环境中使用 Tunnel 内网连接下载数据时，MaxCompute 不会将该操作产生的流量计入计费。
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。

说明：

- 关于SDK的更多详情请参见*SDK Java Doc*。
- 有关服务连接的详情请参见访问域名和数据中心。

1.6.2 TableTunnel

TableTunnel是访问MaxCompute Tunnel服务的入口类，TableTunnel.UploadSession接口表示一个向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。

接口定义如下：

```
public class TableTunnel {
 public DownloadSession createDownloadSession(String projectName,
String tableName);
 public DownloadSession createDownloadSession(String projectName,
String tableName, PartitionSpec partitionSpec);
 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);
 }
```

接口说明如下：

- 生命周期：从TableTunnel实例被创建开始，一直到程序结束。
- 提供创建Upload对象和Download对象的方法。
- 对一张表或partition上传下载的过程，称为一个session。session由一或多个到Tunnel RESTful API的HTTP Request组成。
- TableTunnel的上传session是INSERT INTO语义，即对同一张表或partition的多个/多次上传session互不影响，每个session上传的数据会位于不同的目录中。
- 在上传session中，每个RecordWriter对应一个HTTP Request，由一个block id标识，对应service端一个文件（实际上block id就是对应的文件名）。
- 同一session中，使用同一block id多次打开RecordWriter会导致覆盖行为，最后一个调用close()的RecordWriter所上传的数据会被保留。该特性可用于block的上传失败重传。

TableTunnel接口实现流程：

1. RecordWriter.write()将数据上传到临时目录的文件。
2. RecordWriter.close()将相应的文件从临时目录挪移到data目录。
3. session.commit()将相应data目录下的所有文件挪移到相应表所在目录，并更新表meta，即数据进表，对其他MaxCompute任务（如SQL、MR）可见。

TableTunnel接口限制：

- block id 的取值范围是[0, 20000)，单个block上传的数据限制为100G。
- session的超时时间为24小时。如果，大批量数据传送导致超过24小时，需要自行拆分成多个session。
- RecordWriter对应的HTTP Request超时为120s。如果120s内HTTP连接上没有数据流过，service端会主动关闭连接。

  说明：

  HTTP本身还有8k buffer，因此并不是每次调用RecordWriter.write()都能保证HTTP连接上有数据流过。TunnelRecordWriter.flush()可以将buffer内数据强制刷出。
- 对于日志类写入MaxCompute的场景，因为数据到达不可预测，容易造成RecordWriter超时。此时：
  - 不建议对每条数据打开一个RecordWriter（因为每个RecordWriter对应一个文件，会造成小文件过多，严重影响MaxCompute后续使用性能）。
  - 建议用户代码cache至少64M的数据再使用一个RecordWriter一次性批量写入。

- RecordReader的超时为300s。

1.6.3 UploadSession

接口定义如下：

```
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 TunnelExce
ption;
        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);
        public RecordWriter openBufferedWriter();
        public RecordWriter openBufferedWriter(boolean compress);
    }
```

接口说明如下：

- 生命周期：从创建Upload实例到结束上传。
- 创建**Upload**实例：您可以通过调用构造方法和**TableTunnel**两种方式进行创建。
  - 请求方式：同步。
  - Server端会为该Upload创建一个session， 生成唯一UploadId标识该Upload，客户端可以通过**`getId`**获取。
- 上传数据：
  - 请求方式：同步。
  - 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，也描述了数据在整个表中的位置，取值范围为[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列表不一致抛出异常。
  - Commit失败可以进行重试。
- 七种状态说明如下：
  - 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。

说明：

- 同一个Upload中Session的blockId不能重复。也就是说，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。
- 一个block大小上限100GB，建议大于64M的数据。
- 每个Session在服务端的生命周期为24小时。
- 上传数据时，Writer每写入8KB数据，便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，此时Writer将不可用，请重新打开一个新的Writer写入。
- 建议使用openBufferedWriter接口上传数据，该接口对您屏蔽了blockId的细节，并且内部带有数据缓存区，会自动进行失败重试，详情请参见TunnelBufferedWriter的介绍和示例。

1.6.4 DownloadSession

接口定义如下：

```
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)
    }
```

接口说明如下：

- 生命周期：从创建Download实例到下载结束。
- 创建**Download**实例：您可以通过调用构造方法和TableTunnel两种方式进行创建。
  - 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标识本次下载的Record的起始位置，从0开始，取值范围是>= 0， count标识本次下载的记录数，取值范围是>0。
- 查看下载：
  - 请求方式：同步。
  - 调用**getStatus**，可以获取当前Download状态。
- 四种状态说明如下：
  - 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。

1.6.5 TunnelBufferedWriter

一次完整的上传流程通常包括以下步骤：

1. 先对数据进行划分。
2. 为每个数据块指定block Id，即调用**`openRecordWriter(id)`**。
3. 用一个或多个线程分别将这些block上传上去，并在某个block上传失败以后，需要对整个block进行重传。
4. 在所有block都上传以后，向服务端提供上传成功的blockid list进行校验，即调用**`session.commit([1,2,3,…])`**。

   由于服务端对block管理，连接超时等的一些限制，上传过程逻辑变得比较复杂，为了简化上传过程，SDK提供了更高级的一种RecordWriter—TunnelBufferWriter。

接口定义如下：

```
public class TunnelBufferedWriter implements RecordWriter {
        public TunnelBufferedWriter(TableTunnel.UploadSession session
, CompressOption option) throws IOException;
        public long getTotalBytes();
        public void setBufferSize(long bufferSize);
        public void setRetryStrategy(RetryStrategy strategy);
        public void write(Record r) throws IOException;
        public void close() throws IOException;
    }
```

TunnelBufferedWriter 对象：

- 生命周期：从创建RecordWriter到数据上传结束。
- 创建TunnelBufferedWriter实例：通过调用UploadSession的**`openBufferedWriter`**接口创建。
- 数据上传：调用Write接口，数据会先写入本地缓存区，缓存区满后会批量提交到服务端，避免了连接超时，同时，如果上传失败会自动进行重试。
- 结束上传：调用**`close`**接口，最后再调用UploadSession的**`commit`**接口，即可完成上传。
- 缓冲区控制：可以通过**`setBufferSize`**接口修改缓冲区占内存的字节数（bytes），建议设置64M以上的大小，避免服务端产生过多小文件，影响性能，一般无须设置，维持默认值即可。
- 重试策略设置：您可以选择三种重试回避策略：指数回避（EXPONENTIAL_BACKOFF）、线性时间回避（LINEAR_BACKOFF）、常数时间回避（CONSTANT_BACKOFF）。

  例如：下面这段代码可以将Write的重试次数调整为6，每一次重试之前先分别回

避4s、8s、16s、32s、64s和128s（从4开始的指数递增的序列），这个也是默认的行为，一般情况不建议调整。

```
RetryStrategy retry
  = new RetryStrategy(6, 4, RetryStrategy.BackoffStrategy.EXPONENTIA
L_BACKOFF)
writer = (TunnelBufferedWriter) uploadSession.openBufferedWriter();
writer.setRetryStrategy(retry);
```

1.7 批量数据通道SDK示例

1.7.1 示例概述

- MaxCompute提供了两个服务地址供您进行选择。Tunnel服务地址的选择会直接影响您上传数据的效率及计量计费。详情请参见*Tunnel SDK*简介。
- 数据上传时，建议您使用TunnelBufferedWriter，详情请参见*BufferedWriter*相关的示例代码。
- 不同版本SDK会有不同，本示例仅供参考。请您注意版本变更。

1.7.2 简单上传示例

```
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 odpsUrl = "http://service.odps.aliyun.
com/api";
         private static String tunnelUrl = "http://dt.cn-shanghai.
maxcompute.aliyun-inc.com";
                         //设置tunnelUrl，若需要走内网时必须设置，否则默认公网。此处给的是华东2经典网络Tunnel Endpoint，其他region可以参考文档《访问域名和数据中心》。
         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);//tunnelUrl设置
                            PartitionSpec partitionSpec = new PartitionS
pec(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();
                }
        }
 }
```

构造器举例说明：

PartitionSpec(String spec)：通过字符串构造此类对象。

参数说明：

spec：分区定义字符串，比如pt='1'，ds='2'。

因此程序中应该配置如下：

```
private static String partition = "pt='XXX',ds='XXX'";
```

1.7.3 简单下载示例

```
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 odpsUrl = "http://service.odps.aliyun.
com/api";
         private static String tunnelUrl = "http://dt.cn-shanghai.
maxcompute.aliyun-inc.com";
                        //设置tunnelUrl，若需要走内网时必须设置，否则默认公
网。此处给的是华东2经典网络Tunnel Endpoint，其他region可以参考文档《访问域名和数
据中心》。
         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);//tunnelUrl设置
                 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();
        }
 }
```

本示例中，为了方便测试，数据通过System.out.println直接打印出来，在实际使用时，您可改写为直接输出到文本文件。

1.7.4 多线程上传示例

```
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());
                        }
                }
                for (int i = 0; i < 10; i++) {
                        try {
                                recordWriter.write(record);
                        } catch (IOException e) {
                                recordWriter.close();
                                e.printStackTrace();
                                return false;
                        }
                }
                recordWriter.close();
                return true;
        }
 }
 public class UploadThreadSample {
         private static String accessId = "<your access id>";
         private static String accessKey = "<your access Key>";
         private static String odpsUrl = "<http://service.odps.aliyun.
com/api>";
         private static String tunnelUrl = "<http://dt.cn-shanghai.
maxcompute.aliyun-inc.com>";
                        //设置tunnelUrl，若需要走内网时必须设置，否则默认公
网。此处给的是华东2经典网络Tunnel Endpoint，其他region可以参考文档《访问域名和数
据中心》。
         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);//tunnelUrl设置
                        PartitionSpec partitionSpec = new PartitionS
pec(partition);
                        UploadSession uploadSession = tunnel.
createUploadSession(project,
                                        table, partitionSpec);
                        System.out.println("Session Status is : "
```

```
                                                + uploadSession.getStatus().
toString());
                        ExecutorService pool = Executors.newFixedTh
readPool(threadNum);
                        ArrayList<Callable<Boolean>> callers = new
ArrayList<Callable<Boolean>>();
                        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();
                }
        }
 }
```

对于Tunnel Endpoint，支持指定或者不指定。

- 如果指定，按照指定的Endpoint路由。
- 如果不指定，默认为公网。

1.7.5 多线程下载示例

```
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 odpsUrl = "http://service.odps.aliyun.
com/api";
         private static String tunnelUrl = "http://dt.cn-shanghai.
maxcompute.aliyun-inc.com";
                        //设置tunnelUrl，若需要走内网时必须设置，否则默认公
网。此处给的是华东2经典网络Tunnel Endpoint，其他region可以参考文档《访问域名和数
据中心》。
         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);//tunnelUrl设置
                 PartitionSpec partitionSpec = new PartitionSpec(
partition);
                 DownloadSession downloadSession;
                 try {
                        downloadSession = tunnel.createDownloadSessio
n(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.newFixedTh
readPool(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();
                    }
            }
     }
```

说明：

对于Tunnel Endpoint，支持指定或者不指定。

- 如果指定，按照指定的Endpoint下载。
- 如果不指定，默认为公网Endpoint。

1.7.6 BufferedWriter多线程上传示例

```
class UploadThread extends Thread {
  private UploadSession session;
  private static int RECORD_COUNT = 1200;
  public UploadThread(UploadSession session) {
    this.session = session;
  }
  @Override
  public void run() {
    RecordWriter writer = up.openBufferedWriter();
    Record r = up.newRecord();
    for (int i = 0; i < RECORD_COUNT; i++) {
      r.setBigint(0, i);
      writer.write(r);
    }
    writer.close();
  }
};
```

```
public class Example {
  public static void main(String args[]) {
    // 初始化 MaxCompute 和 tunnel 的代码
    TableTunnel.UploadSession uploadSession = tunnel.createUplo
adSession(projectName, tableName);
    UploadThread t1 = new UploadThread(up);
    UploadThread t2 = new UploadThread(up);
    t1.start();
    t2.start();
    t1.join();
    t2.join();
    uploadSession.commit();
 }
```

1.7.7 BufferedWriter上传示例

```
// 初始化 MaxCompute 和 tunnel 的代码
RecordWriter writer = null;
TableTunnel.UploadSession uploadSession = tunnel.createUploadSession(
projectName, tableName);
try {
  int i = 0;
  // 生成TunnelBufferedWriter的实例
  writer = uploadSession.openBufferedWriter();
  Record product = uploadSession.newRecord();
  for (String item : items) {
    product.setString("name", item);
    product.setBigint("id", i);
    // 调用write接口写入数据
    writer.write(product);
    i += 1;
  }
} finally {
  if (writer != null) {
    // 关闭TunnelBufferedWriter
    writer.close();
  }
}
// uploadSession提交，结束上传
uploadSession.commit();
```

1.8 DataHub实时数据通道

*DataHub*是MaxCompute提供的流式数据处理（Streaming Data）服务，它提供流式数据的发布（Publish）和订阅 （Subscribe）的功能，让您可以轻松构建基于流式数据的分析和应用。

1.9 数据通道服务连接

DataHub和Tunnel在不同网络环境场景下，所使用的EndPoint会有所区别。您在不同网络环境下，需要选择不同的服务地址（Endpoint）来连接服务，否则将无法向服务发起请求。同时，不同的网络连接也会对您的计费产生影响。

具体的服务连接地址请参见访问域名和数据中心。

2 常用命令

2.2 项目空间操作

进入项目空间

命令格式如下：

```
use <project_name>;
```

行为说明如下：

- 进入指定的项目空间。进入该空间后可以直接操作该项目空间下的所有对象。
- 项目空间不存在或当前用户不在此项目空间中，则异常返回。

示例如下：

```
odps@ my_project>use my_project;    --my_project是用户有权限访问的一个
project
```

说明：

以上示例在客户端中运行。所有的 MaxCompute 命令关键字、项目空间名、表名、列名大小写不敏感。

成功运行命令后，您即可直接访问该项目空间下的对象。假设my_project项目空间下有表test_src，您运行如下命令：

```
odps @ my_project>select * from test_src;
```

MaxCompute便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，如果此表不存在，则报异常退出。如果您在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 @ my_project>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。您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，详情请参见项目管理。

2.3 表操作

本文将为您介绍如何通过客户端使用常用命令进行表操作。

您如果想对表进行操作，既可以通过客户端使用常用命令进行操作，也可以通过DataWorks中可视化的数据表管理方便地对表进行收藏、申请权限、查看分区信息等操作，详情请参见表详情页介绍。

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
```

行为说明如下：

创建一张表。

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。
- [LIFECYCLE days] days参数为生命周期时间，只接受正整数。单位：天。
- 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，则此表无需您干预，将会被MaxCompute自动回收（类似drop table操作）。
- 分区表：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。

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_deail 。
```

Drop Table

命令格式如下：

```
DROP TABLE [IF EXISTS] table_name; -- table_name：要删除的表名。
```

行为说明如下：

- 删除一张表。
- 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：

```
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>;      -- table_name：表名或视图名称
DESC extended <table_name>;--查看外部表信息
```

行为说明如下：

返回指定表的信息，具体返回包括以下信息：

- 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压缩后的大小，压缩比一般为5倍，单位Byte。
- Native Columns：非分区列的信息，包括列名、类型和备注。
- Partition Columns：分区列信息，包括分区名、类型和备注。
- Extended Info：外部表StorageHandler 、Location等信息。

示例如下：

```
odps@ project_name>DESC sale_detail;                -- 描述一张分区表
+------------------------------------------------------------------------------------
+
| Owner: ALIYUN$odpsuser@aliyun.com | Project: test_project
                |
| TableComment:
                |
+------------------------------------------------------------------------------------
+
| CreateTime:               2014-01-01 17:32:13
                |
| LastDDLTime:              2014-01-01 17:57:38
                |
| LastModifiedTime:         1970-01-01 08:00:00
                |
+------------------------------------------------------------------------------------
+
| InternalTable: YES      | Size: 0
                |
+------------------------------------------------------------------------------------
+
| Native Columns:
                |
+------------------------------------------------------------------------------------
+
| Field           | Type       | Comment
                |
+------------------------------------------------------------------------------------
+
| shop_name       | string     |
                |
| customer_id     | string     |
                |
| total_price     | double     |
                |
+------------------------------------------------------------------------------------
+
| Partition Columns:
                |
+------------------------------------------------------------------------------------
+
| sale_date       | string     |
                |
| region          | string     |
                |
+------------------------------------------------------------------------------------
+
```

说明：

- 上述示例中的命令在客户端中运行。
- 如果是非分区的表，将不会显示Partition Columns的相关信息。

- 如果描述的是一个视图（View），将不显示InternalTable选项，而是VirtualView选项，其值总是为YES。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。

查看分区信息

命令格式如下：

```
desc table_name partition(pt_spec)
```

行为说明如下：

查看某个分区表具体的分区信息。

示例如下：

```
odps@ project_name>desc meta.m_security_users partition (ds='20151010
');
+------------------------------------------------------------------------------------
+
| PartitionSize: 2109112
                |
+------------------------------------------------------------------------------------
+
| CreateTime:                2015-10-10 08:48:48
                |
| LastDDLTime:               2015-10-10 08:48:48
                |
| LastModifiedTime:          2015-10-11 01:33:35
                |
+------------------------------------------------------------------------------------
+
OK
```

Show Tables/Show Tables like

命令格式如下：

```
SHOW TABLES;
SHOW TABLES like 'chart';
```

行为说明如下：

- SHOW TABLES：列出当前项目空间下所有的表。
- SHOW TABLES like 'chart'：列出当前项目空间下表名与'chart'匹配上的表，支持正则表达式。

示例如下：

```
odps@ project_name>show tables;
odps@ project_name>show tables like 'ods_brand*';
ALIYUN$odps_user@aliyun.com:table_name
```

```
......
```

说明：

- 上述示例中的命令在客户端中运行。
- ALIYUN是系统提示符，表示您是阿里云用户。
- 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：指定查询的表名称（表不存在或非分区表报错）
```

行为说明如下：

列出一张表中的所有分区。

示例如下：

```
odps@ project_name>SHOW PARTITIONS table_name;
partition_col1=col1_value1/partition_col2=col2_value1
partition_col1=col1_value2/partition_col2=col2_value2
…
```

说明：

- 上述示例的命令在客户端运行。
- 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。

2.4 实例操作

Show Instances/Show P

命令格式如下：

```
SHOW INSTANCES [FROM startdate TO enddate] [number];
SHOW P [FROM startdate TO enddate] [number];
SHOW INSTANCES [-all];
SHOW P [-all];
```

行为说明如下：

返回由当前用户创建的实例信息。

参数说明如下：

- **startdate**、**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，返回您在三天内提交的实例。
- **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。
- **-all**：返回当前项目下所有执行过的实例。注意执行该命令的用户需要有project的list权限。
- 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和Suspended。

InstanceID以及实例对应的**SQL**如下所示：

```
StartTime             RunTime  Status   InstanceID
Owner       Query
2015-04-28 13:57:55   1s       Success  20150428055754916grvd5vj4
ALIYUN$xxxxx@aliyun-inner.com       select * from tab_pack_priv limit
20;
...    ...    ...    ...    ...    ...
...    ...    ...    ...    ...    ......
```

Instance 有以下 **6** 种状态：

- 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。

说明：

上述示例的命令在客户端运行。

Status Instance

命令格式如下：

```
STATUS ; -- instance_id: 实例的唯一标识符。指定查询哪个实例状态。
```

行为说明如下：

- 返回指定实例的状态，状态包括Success、Failed、Running和Cancelled。
- 如果此实例并非当前用户创建，则报异常返回。

示例如下：

```
odps@ $project_name>status 20131225123302267gk3u6k4y2;
Success
```

查看ID为20131225123302267gk3u6k4y2的实例状态，查询结果为Success。

说明：

上述示例的命令在客户端运行。

TOP INSTANCE

命令格式如下：

```
TOP INSTANCE;
```

行为说明如下：

返回当前project中正在执行的作业信息，包括ISNTANCEID | Owner | Type | StartTime | Progress | Status | Priority | RuntimeUsage（CPU/MEM）| TotalUsage（CPU/MEM）| QueueingInfo（POS/LEN）等。

示例如下：

```
odps@ $project_name>top instance;
```

说明：

上述示例的命令在客户端（版本为0.29.0或以上）运行。

Kill Instance

命令格式如下：

```
kill ; --instance_id：实例的唯一标识符。必须是状态为Running的实例的ID，否则抛异常
```

行为说明如下：

停止您指定的实例，此实例的状态必须为Running。

示例如下：

```
odps@ $project_name>kill 20131225123302267gk3u6k4y2;
```

停止ID为20131225123302267gk3u6k4y2的实例。

说明：

- 上述示例的命令在客户端运行。
- 此示例是一个异步的过程，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，而只代表系统已接收到请求，因此还要用**status**命令查看此instance的状态后才可以确定。

Desc Instance

命令格式如下：

```
desc instance ; --instance_id：实例的唯一标识符
```

行为说明如下：

根据具体的实例ID获得作业信息，包括具体的SQL、owner、startime、endtime和status等信息。

示例如下：

```
odps@ $project_name> desc instance 20150715103441522gond1qa2;
ID                                      20150715103441522gond1qa2
Owner                                   ALIYUN$maojing.mj@alibaba-inc.
com
StartTime                               2015-07-15 18:34:41
EndTime                                 2015-07-15 18:34:42
Status                                  Terminated
console_select_query_task_1436956481295 Success
Query                                   select * from mj_test;
```

查询ID为20150715103441522gond1qa2的实例对应的作业信息。

说明：

上述示例的命令在客户端运行。

Wait Instance

命令格式如下：

```
wait ; --instance_id：实例的唯一标识符
```

行为说明如下：

根据具体的实例ID获得任务运行日志信息，包含logview链接，再通过查看logview可以获得任务的详细日志。

示例如下：

```
wait 20170925161122379g357ldqp;
ID = 20170925161122379g357ldqp
Log view:
http://logview.odps.aliyun.com/logview/?h=http://service.odps.aliyun.
com/api&p=alian&i=20170925161122379g357ldqp&token=WnlzSGMwZG5vMUZxMGFT
Wk5hUElwYm1jb21VPSxPRFBTX09CTzoxMzI5MzgzMDA0NTQwNjUxLDE1MDcxOTE0MDYsey
JTdGF0ZW1lbnQiOlt7IkFjdGlvbiI6WyJvZHBzOlJlYWQiXSwiRWZmZWN0IjoiQWxsb3ci
LCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnByb2plY3RzL2FsaWFuL2luc3RhbmNlcy8yMD
E3MDkyNTE2MTEyMjM3OWczNTdsZHFwIl19XSwiVmVyc2lvbiI6IjEifQ==
Job Queueing...
Summary:
resource cost: cpu 0.05 Core * Min, memory 0.05 GB * Min
inputs:
        alian.bank_data: 41187 (588232 bytes)
outputs:
        alian.result_table: 8 (640 bytes)
Job run time: 2.000
Job run mode: service job
Job run engine: execution engine
M1:
        instance count: 1
        run time: 1.000
        instance time:
                min: 1.000, max: 1.000, avg: 1.000
        input records:
                TableScan_REL5213301: 41187  (min: 41187, max: 41187,
avg: 41187
)
        output records:
                StreamLineWrite_REL5213305: 8  (min: 8, max: 8, avg: 8
)
R2_1:
        instance count: 1
        run time: 2.000
        instance time:
                min: 2.000, max: 2.000, avg: 2.000
        input records:
                StreamLineRead_REL5213306: 8  (min: 8, max: 8, avg: 8)
        output records:
```

```
                    TableSink_REL5213309: 8  (min: 8, max: 8, avg: 8)
```

2.5 资源操作

本文将为您介绍如何通过客户端使用常用命令对资源进行操作。

您如果想对资源进行操作，既可以通过客户端使用常用命令进行操作，也可以通过DataWorks中可视化的在线数据开发工具对资源进行搜索、上传等操作，详情请参见资源管理。

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];
```

参数说明：

- **file/archive/table/jar**：表示资源类型，资源类型的介绍请参见资源#Resource#。
- **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，注意目前不支持添加外部表为Resource。
- **[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，不加该参数时默认文件名为资源名。Jar及Python类型资源不支持此功能。
- **[comment 'cmt']**：给资源添加注释。
- **[-f]**：当存在同名的资源时，此操作会覆盖原有资源。若不指定此选项，存在同名资源时，操作将失败。

示例如下：

```
odps@ odps_public_dev>add table sale_detail partition (ds='20150602')
as sale.res comment 'sale detail on 20150602' -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：

目前，每个资源文件的大小不能超过500M，单个SQL、MapReduce任务所引用的资源总大小不能超过2048M，更多限制请参见*MR*限制汇总。

删除资源

命令格式如下：

```
DROP RESOURCE ; --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;
```

行为说明如下：

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
```

下载资源

命令格式如下：

```
GET RESOURCE [:] ;
```

行为说明如下：

下载文件类型的资源到本地。资源类型必须为file、jar、archive和py，不支持table类型。

示例如下：

```
odps@ $project_name>get resource odps-udf-examples.jar d:\;
OK
```

2.6 函数操作

本文将为您介绍如何通过客户端使用常用命令对函数进行操作。

您如果想对函数进行操作，既可以通过客户端使用常用命令进行相关操作，也可以通过DataWorks中可视化的在线数据开发工具对资源进行新建、搜索等操作，详情请参见 文件目录中的函数管模块。

注册函数

命令格式如下：

```
CREATE FUNCTION  AS  USING ;
```

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，且必须使用引号。
- **resource_list**：UDF所用到的资源列表。
  - 此资源列表必须包括UDF代码所在的资源。
  - 如果您的代码中通过distributed cache接口读取资源文件，此列表中还要包括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，且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，如下所示：

```
CREATE FUNCTION test_lower AS 'org.alidata.odps.udf.examples.Lower'
    USING 'my_lower.jar';
```

说明：

- 与资源文件一样，同名函数只能注册一次。
- 一般情况下，您的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。如果您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 ;
```

示例如下：

```
DROP FUNCTION test_lower;
```

查看函数清单

命令格式如下：

```
list functions;      --查看当前项目空间中的所有的自定义函数
ls functions -p my_project;      --查看指定项目空间 my_project 下的所有自定义函数
```

2.7 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*MapReduce*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>;
```

行为说明如下：

为资源创建别名。

示例如下：

```
ADD TABLE src_part PARTITION (ds='20121208') AS res_20121208;
ADD TABLE src_part PARTITION (ds='20121209') AS res_20121209;
ALIAS resName=res_20121208;
jar -resources resName -libjars work.jar  -classpath ./work.jar com.company.MainClass args ...;  // 作业一
ALIAS resName=res_20121209;
jar -resources resName -libjars work.jar  -classpath ./work.jar com.company.MainClass args ...; // 作业二
```

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>"]
```

行为说明如下：

您可以使用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
set odps.sql.allow.fullscan=false/true  --设置是否允许对分区表进行全表扫描，
false不允许，true为允许。
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MaxCompute 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MaxCompute 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。
set odps.sql.type.system.odps2=true/false;   -- 默认为false，SQL（Create
、select、insert等操作）中涉及到新数据类型（TINYINT、SMALLINT、 INT、 FLOAT、
VARCHAR、TIMESTAMP BINARY）时需要设置为true。
```

Show Flags

命令格式如下：

```
show flags; --显示Set设置的参数
```

行为说明如下：

运行**Use Project**命令会清除掉Set命令设置的配置。

SetProject

命令格式如下：

```
setproject ["="];
```

行为说明如下：

- 您可以使用**setproject**命令设置Project属性。
- 当不指定< KEY >=< VALUE >时，显示当前Project的属性配置。

Project属性的详细说明如下：

属性名称	设置权限	属性描述	取值范围
odps.sql.allow.fullscan	ProjectOwner	项目是否允许全表扫描	true（允许）/false（禁止）
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.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
READ_TABLE_MAX_ROW	ProjectOwner	Select语句返回给客户端的数据条数	1~10000

odps.security.ip.whitelist示例

MaxCompute支持Project级别的IP白名单。

说明：

- 设置IP白名单后，只有白名单列表中的IP（console或者SDK所在的出口IP）能够访问这个Project。
- 设置IP白名单后，您需要等待五分钟后才会生效。

白名单中IP列表的表示格式有三种：

- 单纯IP：例如101.132.236.134。
- 子网掩码：100.116.0.0/16。
- 网段：101.132.236.134-101.132.236.144。

这三种格式可以写在同一个命令中，用逗号分割。

例如命令行工具设置IP白名单的方法：

```
setproject odps.security.ip.whitelist=101.132.236.134,100.116.0.0/16,
101.132.236.134-101.132.236.144;
```

IP白名单清空后MaxCompute就认为Project关闭了白名单功能。

```
setproject odps.security.ip.whitelist=;
```

setproject

命令格式如下：

```
setproject; --显示setproject设置的参数
```

计量预估（**Cost SQL**命令）

命令格式如下：

```
cost sql <SQL Sentence>;
```

行为说明如下：

预估出一条SQL的计量信息，包含输入数据的大小、UDF个数以及SQL复杂等级。

说明：

该信息不能够作为实际计费标准，仅具有参考意义。

示例如下：

```
odps@ $odps_project >cost sql select distinct project_name, user_name
 from meta.m_security_users distribute by project_name sort by
project_name;
ID = 20150715113033121gmsbjxl1
```

```
Input:65727592 Bytes
UDF:0
Complexity:1.0
```

3 SQL

3.1 SQL概述

MaxCompute SQL适用于海量数据（GB、TB、EB级别），离线批量计算的场合。MaxCompute作业提交后会有几十秒到数分钟不等的排队调度，所以适合处理跑批作业，一次作业批量处理海量数据，不适合直接对接需要每秒处理几千至数万笔事务的前台业务系统。

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，更多差异请参见与其他*SQL*语法差异。目前在MaxCompute中允许的最大SQL长度是2MB。

关键字

MaxCompute将SQL语句的关键字作为保留字。在对表、列或是分区命名时如若使用关键字，需给关键字加``符号进行转义，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
```

类型转换说明

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**。更多详情请参见类型转换。

- 显式类型转换：是指用cast将一种数据类型的值转换为另一种类型的值的行为。
- 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。隐式转换作用域包括各种运算符、内建函数等作用域。

分区表

MaxCompute SQL支持分区表。指定分区表会对您带来诸多便利，例如提高SQL运行效率、减少计费等。关于分区的详情请参见基本概念>分区。

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，否则会报异常。

Select Transform

Select Transform功能明显简化了对脚本代码的引用，支持Java、Python、Shell、Perl等语言，且编写过程简单，适合adhoc功能的实现。详情请参见*Select Transform*语法。

目前MaxCompute的select transform完全兼容了Hive的语法、功能和行为，包括input/output row format以及reader/writer。Hive上的脚本，大部分可以直接运行，部分脚本只需要经过稍微的改动即可运行。

3.2 运算符

关系操作符

操作符	说明
A=B	如果A或B为NULL，返回NULL；如果A等于B，返回TRUE，否则返回FALSE。
A<>B	如果A或B为NULL，返回NULL；如果A不等于B，返回TRUE，否则返回FALSE。
A<B	如果A或B为NULL，返回NULL；如果A小于B，返回TRUE，否则返回FALSE。
A<=B	如果A或B为NULL，返回NULL；如果A小于等于B，返回TRUE，否则返回FALSE。
A>B	如果A或B为NULL，返回NULL；如果A大于B，返回TRUE，否则返回FALSE。
A>=B	如果A或B为NULL，返回NULL；如果A大于等于B，返回TRUE，否则返回FALSE。
A IS NULL	如果A为NULL，返回TRUE，否则返回FALSE。
A IS NOT NULL	如果A不为NULL，返回TRUE，否则返回FALSE。

A LIKE B	如果A或B为NULL，返回NULL，A为字符串，B为要匹配的模式， 如果匹配，返回TRUE，否则返回FALSE。‘%’匹配任意多个字符，‘_’匹配单个字符。要匹配‘%’或‘_’需要用转义符表示‘\%’，‘_’。
	`‘aaa’ like‘a__’= TRUE`
	`‘aaa’ like‘a%’ = TRUE`
	`‘aaa’ like‘aab’= FALSE`
	`‘a%b’ like‘a\%b’= TRUE`

操作符	说明
	'axb' like 'a\%b'= FALSE
A RLIKE B	A是字符串，B是字符串常量正则表达式。如果匹配成功，返回TRUE，否则返回FALSE。 如果B为空串会报错退出。如果A或B为NULL，返回NULL。
A IN B	B是一个集合，如果A为NULL，返回NULL，如A在B中则返回TRUE，否则返回FALSE 若B仅有一个元素NULL，即A IN（NULL），则返回NULL。 若B含有NULL元素，将NULL视为B集合中其他元素的类型。 B必须是常数并且至少有一项，所有类型要一致
BETWEEN AND	表达式为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。

常见用法如下所示：

```
select * from user where user_id = '0001';
select * from user where user_name <> 'maggie';
select * from user where age > '50';
select * from user where birth_day >= '1980-01-01 00:00:00';
select * from user where is_female is null;
select * from user where is_female is not null;
select * from user where user_id in (0001,0010);
select * from user where user_name like 'M%';
```

由于Double值存在一定的精度差，因此，不建议您直接使用等号对两个Double类型的数据进行比较。您可以使用两个Double类型相减，然后取绝对值的方式进行判断。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，详情请参见*ABS*。
- 通常情况下，MaxCompute的Double类型能够保障14位有效数字。

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。

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。注：如果A和B为Bigint类型，返回结果为Double类型。
A % B	如果A或B为NULL，返回NULL；否则返回A模B的结果。
+A	仍然返回A。
-A	如果A为NULL，返回NULL，否则返回-A。

常见用法如下：

```
select age+10, age-10, age%10, -age, age*age, age/10 from user;
```

说明：

- 只有参数是String、Bigint或Double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，转换为Double类型。
- Bigint和Double类型共同参与计算时，会将Bigint隐式转换为Double再进行计算，返回结果为Double类型。
- A和B都是Bigint类型，进行A/B运算，返回结果为Double类型。进行上述其他运算，仍然返回Bigint类型。

位操作符

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。

说明：

位运算符不支持隐式转换，只允许Bigint类型。

逻辑操作符

```
操作符                  说明
A and B             TRUE and TRUE=TRUE
                    TRUE and FALSE=FALSE
```

```
                    FALSE and TRUE=FALSE
                    FALSE and NULL=FALSE
                    NULL and FALSE=FALSE
                    TRUE and NULL=NULL
                    NULL and TRUE=NULL
                    NULL and NULL=NULL
A or B              TRUE or TRUE=TRUE
                    TRUE or FALSE=TRUE
                    FALSE or TRUE=TRUE
                    FALSE or NULL=NULL
                    NULL or FALSE=NULL
                    TRUE or NULL=TRUE
                    NULL or TRUE=TRUE
                    NULL or NULL=NULL
NOT A               如果A是NULL，返回NULL
                    如果A是TRUE，返回FALSE
                    如果A是FALSE，返回TRUE
```

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。

3.3 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**。

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，如下表所示：

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。

示例如下：

```
select cast(user_id as double) as new_id from user;
```

```
select cast('2015-10-01 00:00:00' as datetime) as new_date from user;
```

说明：

- 将Double类型转为Bigint类型时，小数部分会被截断，例如`cast(1.6 as bigint) = 1`。
- 满足Double格式的String类型转换为Bigint时，会先将String转换为Double，再将Double转换为Bigint，因此，小数部分会被截断，例如`cast("1.6" as bigint) = 1`。
- 满足Bigint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`。
- 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，详情请参见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**，详情请参见*TO_CHAR*，而**to_date**函数同样支持从String类型到Datetime类型的转换，详情请参见*TO_DATE*。
- 关于cast的介绍请参见*CAST*。
- DECIMAL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。

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，如下表所示：

	boole	tinyint	smalli	int	bigint	float	doubl	decim	string	varch	timest	binary
boolean to	T	F	F	F	F	F	F	F	F	F	F	F
tinyint to	F	T	T	T	T	T	T	T	T	T	F	F
smallint to	F	F	T	T	T	T	T	T	T	T	F	F
int to	F	F	F	T	T	T	T	T	T	T	F	F
bigint to	F	F	F	F	T	T	T	T	T	T	F	F
float to	F	F	F	F	F	T	T	T	T	T	F	F
double to	F	F	F	F	F	F	T	T	T	T	F	F

	boolea	tinyint	smalli	int	bigint	float	doubl	decim	string	varcha	timest	binary
decimal to	F	F	F	F	F	F	F	T	T	T	F	F
string to	F	F	F	F	F	F	T	T	T	T	F	F
varchar to	F	F	F	F	F	F	T	T	T	T	F	F
timestamp to	F	F	F	F	F	F	F	F	T	T	T	F
binary to	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：

- MaxCompute2.0新增了DECIMAL类型与Datetime的常量定义方式，100BD就是数值为100的DECIMAL，Datetime`2017-11-11 00:00:00`就是Datetime类型的常量。常量定义的方便之处在于可以直接用到values子句和values表中。
- 旧版MaxCompute中，因为历史原因，Double可以隐式的转换为Bigint，这个转换潜在可能有数据丢失，一般数据库系统都不允许。

常见用法如下所示：

```
select user_id+age+'12345',
          concat(user_name,user_id,age)
  from user;
```

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，也会导致异常。
- 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，因此推荐您在类型不匹配时，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。详情请参见隐式类型转换的作用域。

- 关系运算符作用下的隐式转换

关系运算符包括=、<>、<、<=、>、>=、IS NULL、IS NOT NULL、LIKE、RLIKE和IN。由于LIKE、RLIKE和IN的隐式类型转换规则不同于其他关系运算符，将单独拿出章节对这三种关系运算符做出说明。本小节的说明不包含这三种特殊的关系运算符。

当不同类型的数据共同参与关系运算时，按照下述原则进行隐式类型转换。

From/To	Bigint	Double	String	Datetime	Boolean	Decimal
Bigint	–	Double	Double	N	N	Decimal
Double	Double	–	Double	N	N	Decimal
String	Double	Double	–	Datetime	N	Decimal
Datetime	N	N	Datetime	–	N	N
Boolean	N	N	N	N	–	N
Decimal	Decimal	Decimal	Decimal	N	N	-

说明：

- 如果进行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。
- 关系运算符的更多详情，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和IN。

  - LIKE和RLIKE的使用方式，如下所示：

    ```
    source like pattern;
    source rlike pattern;
    ```

    两者在隐式类型转换中的注意事项，如下所示：

    - 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。
  - IN的使用方式，如下所示：

    ```
    key in (value1, value2, ···)
    ```

    In的隐式转换规则：

    - 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、Double、String之间比较，统一转为Double，若Datetime和String之间比较，统一转为Datetime。除此之外不允许其它类型之间的转换。

- **算术运算符作用下的隐式转换**

  算术运算符包括+、-、* 、/、%、+和-，其隐式转换规则，如下所示：

  - 只有String、Bigint、Double和Decimal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。
  - Bigint和Double共同参与计算时，会将Bigint隐式转换为Double。
  - 日期型和布尔型不允许参与算数运算。

- **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。

内建函数涉及到隐式转换

MaxCompute SQL提供了大量的系统函数，以方便您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，如下所示：

- 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，详情请参见内建函数。

CASE WHEN作用下的隐式转换

CASE WHEN的详情介绍请参见*CASE WHEN*表达式。它的隐式转换规则，如下所示：

- 如果返回类型只有Bigint、Double，统一转为Double。
- 如果返回类型中有String类型，统一转为String，如果不能转则报错（如Boolean类型）。
- 除此之外不允许其它类型之间的转换。

String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`。

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

说明：

- 各个单位的值域中，如果首位为0，不可省略，例如`2014-1-9 12:12:12`就是非法的Datetime格式，无法从这个String类型数据转换为Datetime类型，必须写为`2014-01-09 12:12:12`。
- 只有符合上述格式描述的String类型才能够转换为Datetime类型，例如`cast(“2013-12-31 02:34:34” as datetime)`，将会把String类型`2013-12-31 02:34:34`转换为Datetime类型。同理，Datetime转换为String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，如下所示：

```
cast("2013/12/31 02/34/34" as datetime)
cast("20131231023434" as datetime)
cast("2013-12-31 2:34:34" as datetime)
```

dd部分的阈值上限取决于月份实际拥有的天数，如果超出对应月份实际拥有的天数，将会导致异常退出，如下所示：

```
cast("2013-02-29 12:12:12" as datetime)      -- 异常返回，2013年2月没有29日
cast("2013-11-31 12:12:12" as datetime)      -- 异常返回，2013年11月没有31日
```

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。详情请参见*TO_DATE*。

3.4 DDL语句

3.4.1 表操作

创建表

创建表的语法格式，如下所示：

```
CREATE [EXTERNAL] 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], ...)]
    [STORED BY StorageHandler] -- 仅限外部表
    [WITH SERDEPROPERTIES (Options)] -- 仅限外部表
    [LOCATION OSSLocation];-- 仅限外部表
    [LIFECYCLE days]
    [AS select_statement]
    CREATE TABLE [IF NOT EXISTS] table_name
    LIKE existing_table_name
```

说明如下：

- 创建表时，如果不指定if not exists选项而存在同名表，则返回出错。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。
- 表名与列名均对大小写不敏感，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。
- 单表的列定义个数最多1200个。
- 数据类型：Bigint、Double、Boolean、Datetime、Decimal和String等，MaxCompute2.0版本扩展了很多数据类型。

  说明：

  要使用新数据类型（Tinyint、Smallint、 Int、 Float、Varchar、TIMESTAMP BINARY），需在建表语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与建表语句一起提交执行。
- Partitioned by指定表的分区字段，目前支持Tinyint、Smallint、 Int、 Bigint、Varchar和String类型。

  分区值不允许有双字节字符（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。

- 一张表最多允许60000个分区，单表的分区层次不能超过6级。
- 注释内容是长度不超过1024字节的有效字符串。
- lifecycle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。
- 有关外部表的更多详情请参见处理非结构化数据。

示例如下：

假设创建表sale_detail来保存销售记录，该表使用销售时间（sale_date）和销售区域（region）作为分区列，建表语句如下所示：

```
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
```

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，如下所示：

```
create table sale_detail_ctas1 as
select * from sale_detail;
```

此时，如果sale_detail中存在数据，上面的示例会将sale_detail的数据全部复制到sale_detail_ctas1表中。

说明：

此处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，如下所示：

```
create table sale_detail_ctas2 as
        select shop_name,
            customer_id,
            total_price,
            '2013' as sale_date,
            'China' as region
```

```
        from sale_detail;
```

如果不加列的别名，如下所示：

```
create table sale_detail_ctas3 as
        select shop_name,
            customer_id,
            total_price,
            '2013',
            'China'
        from sale_detail;
```

则创建的表sale_detail_ctas3的第四、五列类似于_c5、_c6。

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，如下所示：

```
create table sale_detail_like like sale_detail;
```

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。

查看表信息

查看表信息的语法格式，如下所示：

```
desc <table_name>;
desc extended <table_name>;--查看外部表信息
```

示例如下：

- 假设查看上述示例中表sale_detail的信息，可输入如下命令：

  ```
  desc sale_detail;
  ```

  结果如下所示：

  ```
  odps@ $odps_project>desc sale_detail;
  +------------------------------------------------------------------------
  +
  | Owner: ALIYUN$maojing.mj@alibaba-inc.com | Project: $odps_project
                    |
  | TableComment:
     |
  +------------------------------------------------------------------------
  +
  | CreateTime:                2017-06-28 15:05:17
     |
  | LastDDLTime:               2017-06-28 15:05:17
     |
  | LastModifiedTime:          2017-06-28 15:05:17
     |
  ```

```
+------------------------------------------------------------------------
+
| InternalTable: YES      | Size: 0
   |
+------------------------------------------------------------------------
+
| Native Columns:
   |
+------------------------------------------------------------------------
+
| Field           | Type       | Label | Comment
   |
+------------------------------------------------------------------------
+
| shop_name       | string     |       |
   |
| customer_id     | string     |       |
   |
| total_price     | double     |       |
   |
+------------------------------------------------------------------------
+
| Partition Columns:
   |
+------------------------------------------------------------------------
+
| sale_date       | string     |
   |
| region          | string     |
   |
+------------------------------------------------------------------------
+
OK
```

- 假设查看上述示例表sale_detail_like中的信息，可输入如下命令：

```
desc sale_detail_like
```

结果如下所示：

```
odps@ $odps_project>desc sale_detail_like;
+------------------------------------------------------------------------
+
| Owner: ALIYUN$maojing.mj@alibaba-inc.com | Project: $odps_project
                  |
| TableComment:
   |
+------------------------------------------------------------------------
+
| CreateTime:               2017-06-28 15:42:17
   |
| LastDDLTime:              2017-06-28 15:42:17
   |
| LastModifiedTime:         2017-06-28 15:42:17
   |
+------------------------------------------------------------------------
+
| InternalTable: YES      | Size: 0
   |
+------------------------------------------------------------------------
+
```

```
| Native Columns:
   |
+------------------------------------------------------------------------
+
| Field           | Type       | Label | Comment
   |
+------------------------------------------------------------------------
+
| shop_name       | string     |       |
   |
| customer_id     | string     |       |
   |
| total_price     | double     |       |
   |
+------------------------------------------------------------------------
+
| Partition Columns:
   |
+------------------------------------------------------------------------
+
| sale_date       | string     |
   |
| region          | string     |
   |
+------------------------------------------------------------------------
+
OK
```

由上可见，除生命周期属性外，sale_detail_like的其他属性（字段类型，分区类型等）均与sale_detail完全一致。查看表信息的更多详情请参见*Describe Table*。

如果您查看表sale_detail_ctas1的信息，会发现sale_date、region两个字段仅会作为普通列存在，而不是表的分区。

删除表

删除表的语法格式，如下所示：

```
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。
- 删除外部表时，OSS上的数据不会被删除。

示例如下：

```
create table sale_detail_drop like sale_detail;
    drop table sale_detail_drop;
    --若表存在，成功返回；若不存在，异常返回；
    drop table if exists sale_detail_drop2;
```

```
    --无论是否存在 sale_detail_drop2 表，均成功返回。
```

重命名表

重命名表的语法格式，如下所示：

```
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，则报错。
- 如果table_name不存在，则报错。

示例如下：

```
create table sale_detail_rename1 like sale_detail;
alter table sale_detail_rename1 rename to sale_detail_rename2;
```

修改表的注释

修改表的注释的语法格式，如下所示：

```
ALTER TABLE table_name SET COMMENT 'tbl comment';
```

说明：

- table_name必须是已存在的表。
- comment最长1024字节。

示例如下：

```
alter table sale_detail set comment 'new coments for table sale_detail
';
```

通过MaxCompute的**desc**命令可以查看表中comment的修改，详情请参见常用命令>表操作中的Describe Table。

修改表的修改时间

MaxCompute SQL提供**touch**操作用来修改表的LastDataModifiedTime，可将表的LastDataModifiedTime修改为当前时间。

修改表的修改时间的语法格式，如下所示：

```
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。

清空非分区表里的数据

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`的方式将分区里的数据清除。

清空非分区表里的数据的语法格式，如下所示：

```
TRUNCATE TABLE table_name;
```

3.4.2 生命周期操作

修改表的生命周期

MaxCompute提供数据生命周期管理功能，以方便您释放存储空间，简化回收数据的流程。

修改表的生命周期属性的语法格式，如下所示：

```
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，单位为天。
- 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，则此表无需您干预，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，不能再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。
```

禁止生命周期

某些情况下，部分特定的分区不希望被生命周期功能自动回收掉，比如一个月的月初或双十一期间的数据，此时您可以禁止该分区被生命周期功能回收。

禁止生命周期的语法格式，如下所示：

```
ALTER TABLE table_name [partition_spec] ENABLE|DISABLE LIFECYCLE;
```

示例如下：

```
ALTER TABLE trans PARTITION(dt='20141111') DISABLE LIFECYCLE;
```

3.4.3 视图操作

创建视图

创建视图的语法格式，如下所示：

```
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。
- 视图只能包含一个有效的select语句。
- 视图可以引用其它视图，但不能引用自己，也不能循环引用。
- 不允许向视图写入数据，例如使用insert into或者insert overwrite操作视图。
- 当建好视图后，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。您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;
```

删除视图

删除视图的语法格式，如下所示：

```
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;
```

重命名视图

重命名视图的语法格式，如下所示：

```
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;
```

3.4.4 分区/列操作

添加分区

添加分区的语法格式，如下所示：

```
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：

- 仅支持新增分区，不支持新增分区字段。
- 如果未指定if not exists而同名的分区已存在，则出错返回。
- 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：

假设为表sale_detail添加一个分区，如下所示：

```
alter table sale_detail add if not exists partition (sale_date='201312
', region='hangzhou');
-- 成功添加分区，用来存储2013年12月杭州地区的销售记录。
alter table sale_detail add if not exists partition (sale_date='201312
', region='shanghai');
-- 成功添加分区，用来存储2013年12月上海地区的销售记录。
alter table sale_detail add if not exists partition(sale_date='
20111011');
-- 仅指定一个分区sale_date，出错返回
alter table sale_detail add if not exists partition(region='shanghai
');
-- 仅指定一个分区region，出错返回
```

删除分区

删除分区的语法格式，如下所示：

```
ALTER TABLE TABLE_NAME DROP [IF EXISTS] PARTITION 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：

假设从表sale_detail中删除一个分区，如下所示：

```
alter table sale_detail drop if exists partition(sale_date='201312',
region='hangzhou');
```

```
-- 成功删除2013年12月杭州分区的销售。
```

添加列

添加列的语法格式，如下所示：

```
ALTER TABLE table_name ADD COLUMNS (col_name1 type1, col_name2 type2
...)
```

> 说明：
> 添加的新列不支持指定顺序，默认在最后一列。

修改列名

修改列名的语法格式，如下所示：

```
ALTER TABLE table_name CHANGE COLUMN old_col_name RENAME TO new_col_na
me;
```

> 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。

修改列、分区注释

修改列、分区注释的语法格式，如下所示：

```
ALTER TABLE table_name CHANGE COLUMN col_name COMMENT comment_string;
```

> 说明：
> COMMENT内容最长为1024字节。

同时修改列名及列注释

同时修改列名及列注释的语法格式，如下所示：

```
ALTER TABLE table_name CHANGE COLUMN old_col_name new_col_name
column_type COMMENT column_comment;
```

> 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。

- COMMENT内容最长为1024字节。

修改表、分区的修改时间

MaxCompute SQL提供touch操作用来修改分区的LastDataModifiedTime。效果会将分区的LastDataModifiedTime修改为当前时间。

修改表、分区的修改时间的语法格式，如下所示：

```
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。

修改分区值

MaxCompute SQL支持通过rename操作更改对应表的分区值。

修改分区值的语法格式，如下所示：

```
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 SELECT操作

3.6 INSERT操作

3.7 SELECT操作

3.8 SQL限制项汇总

一些用户因没注意限制条件，业务启动后才发现限制条件，导致业务停止。为避免此类现象发生，方便用户查看，本文将对MaxCompute SQL限制项做以下汇总：

边界名	最大值/限制条件	分类	说明
表名长度	128字节	长度限制	表名，列名中不能有特殊字符，只能用英文的a-z,A-Z及数字和下划线_，且以字母开头
注释长度	1024字节	长度限制	注释内容是长度不超过1024字节的有效字符串
表的列定义	1200个	数量限制	单表的列定义个数最多1200个
单表分区数	60000	数量限制	一张表最多允许60000个分区
表的分区层级	6级	数量限制	在表中建的分区层次不能超过6级
表统计定义个数	100个	数量限制	表统计定义个数
表统计定义长度	64000	长度限制	表统计项定义长度
屏显	10000行	数量限制	SELECT语句屏显默认最多输出10000行
insert目标个数	256个	数量限制	multiins同时insert的数据表数量
UNION ALL	256个表	数量限制	最多允许256个表的UNION ALL
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
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	一个 partition 列 in subquery 时，subquery 的返回结果不可超过 1000 行。

边界名	最大值/限制条件	分类	说明
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	表中单个cell的最大长度
in的参数个数	1024	数量限制	in的最大参数限制，如in (1,2,3…,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，不可用insert操作
列的数据类	不允许	操作限制	不允许修改列的数据类型，列位置
java udf函数	不能是abstract或者static	操作限制	java udf函数不能是 abstract 或者static
最多查询分区个数	10000	数量限制	最多查询分区个数不能超过10000

说明：

以上MaxCompute SQL限制项均不可以被人为修改配置。

3.9 内建函数

3.9.1 日期函数

MaxCompute SQL提供了针对Datetime类型的操作函数。

DATEADD

命令格式如下：

```
datetime dateadd(datetime date, bigint delta, string datepart)
```

命令说明如下：

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，其它类型抛异常。
- **delta**：Bigint类型，修改幅度。若输入为String类型或Double型会隐式转换到Bigint类型后参与运算，其他类型会引发异常。若delta大于0，则增，否则减。
- **datepart**：String类型常量。此字段的取值遵循String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、不支持的格式会或其它类型抛异常。

返回值：

返回Datetime类型。若任一输入参数为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进制进行计算。
- 当delta的单位是月时，计算规则如下：

  若Datetime的月部分在增加delta值之后不造成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与Datetime类型转换的约定，即yyyy表示年，mm表示月，Datetime相关的内建函数如无特殊说明均遵守此约定。同时如果没有特殊说明，所有Datetime相关的内建函数的part部分也同样支持扩展的日期格式：年-year，月-month或mon，日-day，小时-hour。

示例如下：

```
若trans_date = 2005-02-28 00:00:00：
dateadd(trans_date, 1, 'dd') = 2005-03-01 00:00:00
-- 加一天，结果超出当年2月份的最后一天，实际值为下个月的第一天
dateadd(trans_date, -1, 'dd') = 2005-02-27 00:00:00
-- 减一天
dateadd(trans_date, 20, 'mm') = 2006-10-28 00:00:00
-- 加20个月，月份溢出，年份加1
若trans_date = 2005-02-28 00:00:00, dateadd(transdate, 1, 'mm') = 2005
-03-28 00:00:00
```

```
若trans_date = 2005-01-29 00:00:00, dateadd(transdate, 1, 'mm') = 2005
-02-28 00:00:00
-- 2005年2月没有29日，日期截取至当月最后一天
若trans_date = 2005-03-30 00:00:00, dateadd(transdate, -1, 'mm') =
2005-02-28 00:00:00
```

说明：

此处对trans_date的数值表示仅作示例使用，在文档中有关Datetime的介绍会经常使用到这种简易的表达方式。

在MaxCompute SQL中，Datetime类型没有直接的常数表示方式，如下使用方式是错误的：

```
select dateadd(2005-03-30 00:00:00, -1, 'mm') from tbl1;
```

如果一定要描述Datetime类型常量，请尝试如下方法：

```
select dateadd(cast("2005-03-30 00:00:00" as datetime), -1, 'mm') from
 tbl1;
-- 将String类型常量显式转换为Datetime类型
```

DATEDIFF

命令格式如下：

```
bigint datediff(datetime date1, datetime date2, string datepart)
```

命令说明如下：

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，其它类型抛异常。
- **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其它类型则会发生异常。

返回值：

返回Bigint类型。任一输入参数是NULL，返回NULL。如果date1小于date2，返回值可以为负数。

说明：

计算时会按照datepart切掉低单位部分，然后再计算结果。

示例如下：

```
若start = 2005-12-31 23:59:59，end = 2006-01-01 00:00:00:
    datediff(end, start, 'dd') = 1
    datediff(end, start, 'mm') = 1
    datediff(end, start, 'yyyy') = 1
    datediff(end, start, 'hh') = 1
    datediff(end, start, 'mi') = 1
    datediff(end, start, 'ss') = 1
    datediff('2013-05-31 13:00:00', '2013-05-31 12:30:00', 'ss') =
1800
    datediff('2013-05-31 13:00:00', '2013-05-31 12:30:00', 'mi') = 30
```

DATEPART

命令格式如下所示：

```
bigint datepart(datetime date, string datepart)
```

命令说明如下：

提取日期date中指定的时间单位datepart的值。

参数说明：

返回值：

- **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，其它类型抛异常。
- **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其它类型则会发生异常。

返回Bigint类型。若任一输入参数为NULL，返回NULL。

示例如下：

```
datepart('2013-06-08 01:10:00', 'yyyy')  =  2013
 datepart('2013-06-08 01:10:00', 'mm')  =  6
```

DATETRUNC

命令格式如下：

```
datetime datetrunc (datetime date, string datepart)
```

命令说明如下：

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，其它类型抛异常。
- **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其它类型则会发生异常。

返回值：

Datetime类型。任意一个参数为NULL时，返回NULL。

示例如下：

```
datetrunc('2011-12-07 16:28:46', 'yyyy') = 2011-01-01 00:00:00
 datetrunc('2011-12-07 16:28:46', 'month') = 2011-12-01 00:00:00
 datetrunc('2011-12-07 16:28:46', 'DD') = 2011-12-07 00:00:00
```

FROM_UNIXTIME

命令格式如下：

```
datetime from_unixtime(bigint unixtime)
```

命令说明如下：

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、Double、Decimal类型会隐式转换为Bigint后参与运算。

返回值：

返回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
```

GETDATE

命令格式如下：

```
datetime getdate()
```

命令说明如下：

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（2.0版本会精确到毫秒）。

ISDATE

命令格式如下：

```
boolean isdate(string date, string format)
```

命令说明如下：

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格式的日期值，若输入为Bigint、Double、Decimal或Datetime类型，会隐式转换为String类型后参与运算，其它类型报异常。
- **format**：String类型常量，不支持日期扩展格式。其它类型或不支持的格式会抛异常。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：

返回Boolean类型，如任意参数为NULL，返回NULL。

LASTDAY

命令格式如下：

```
datetime lastday(datetime date)
```

命令说明如下：

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，其它类型报异常。

返回值：

返回Datetime类型，如输入为NULL，返回NULL。

TO_DATE

命令格式如下：

```
datetime to_date(string date, string format)
```

命令说明如下：

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、Double、Decimal或者Datetime类型，会隐式转换为String类型后参与运算，为其它类型则抛异常，为空串时抛异常。
- **format**：String类型常量，日期格式。非常量或其他类型会引发异常。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，否则引发异常，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：

返回Datetime类型，格式为yyyy-mm-dd hh:mi:ss 。若任意一个输入的参数为NULL，则返回NULL值。

示例如下：

```
to_date('阿里巴巴2010-12*03', '阿里巴巴yyyy-mm*dd') = 2010-12-03 00:00:00
to_date('20080718', 'yyyymmdd') = 2008-07-18 00:00:00
to_date('200807182030','yyyymmddhhmi')=2008-07-18 20:30:00
to_date('2008718', 'yyyymmdd')
-- 格式不符合，引发异常
to_date('阿里巴巴2010-12*3', '阿里巴巴yyyy-mm*dd')
-- 格式不符合，引发异常
to_date('2010-24-01', 'yyyy')
```

```
-- 格式不符合，引发异常
```

TO_CHAR

命令格式如下：

```
string to_char(datetime date, string format)
```

命令说明如下：

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，其它类型抛异常。
- **format**：String类型常量。非常量或其他类型会引发异常。format中的日期格式部分会被替换成相应的数据，其它字符直接输出。

返回值：

返回String类型。若任意一个输入的参数为NULL，则返回NULL值。

示例如下：

```
to_char('2010-12-03 00:00:00', '阿里金融yyyy-mm*dd') = '阿里金融2010-12*
03'
    to_char('2008-07-18 00:00:00', 'yyyymmdd') = '20080718'
    to_char('阿里巴巴2010-12*3', '阿里巴巴yyyy-mm*dd') -- 引发异常
    to_char('2010-24-01', 'yyyy') -- 会引发异常
    to_char('2008718', 'yyyymmdd') -- 会引发异常
```

关于其他类型向String类型转换的详情请参见字符串函数>*TO_CHAR*。

UNIX_TIMESTAMP

命令格式如下：

```
bigint unix_timestamp(datetime date)
```

命令说明如下：

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，其它类型抛异常。

返回值：

返回Bigint类型，表示unix格式日期值，date为NULL时，返回NULL值。

WEEKDAY

命令格式如下：

```
bigint weekday (datetime date)
```

命令说明如下：

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，其它类型抛异常。

返回值：

返回Bigint类型，若输入参数为NULL，返回NULL值。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。

WEEKOFYEAR

命令格式如下：

```
bigint weekofyear(datetime date)
```

命令说明如下：

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：

关于这一周算上一年， 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。 算在前一年，就是前一年的最后一周，算在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，其它类型抛异常。

返回值：

返回Bigint类型。若输入为NULL，返回NULL值。

示例如下：

```
select weekofyear(to_date("20141229", "yyyymmdd")) from dual;
返回结果：
+------------+
| _c0        |
+------------+
| 1          |
+------------+
 -虽然20141229属于2014年，但是这一周的大多数日期是在2015年，因此返回结果为1，表示是2015年的第一周。
 select weekofyear(to_date("20141231", "yyyymmdd")) from dual；--返回结果为1。
 select weekofyear(to_date("20151229", "yyyymmdd")) from dual；--返回结果为53。
```

新扩展的日期函数

升级到MaxCompute2.0后，产品扩展了部分日期函数。您如果用到的函数涉及新数据类型，请在使用新函数的SQL前，需要加个set语句：

```
set odps.sql.type.system.odps2=true;--开启新类型
```

若想同时提交，执行以下语句：

```
set odps.sql.type.system.odps2=true;
select year('1970-01-01 12:30:00')=1970 from dual;
```

下文将为您介绍新扩展的函数的详情。

YEAR

命令格式如下：

```
INT year(string date)
```

命令说明如下：

返回一个日期的年。

参数说明：

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值。

返回值：

返回INT类型。

示例如下：

```
year('1970-01-01 12:30:00')=1970
year('1970-01-01')=1970
year('70-01-01')=70
year(1970-01-01)=null
year('1970/03/09')=null
year(null)返回异常
```

QUARTER

命令格式如下：

```
INT quarter(datetime/timestamp/string date )
```

命令说明如下：

返回一个日期的季度，范围是1-4。

参数说明：

date：Datetime/Timestamp/String类型日期值，日期格式至少包含yyyy-mm-dd，其他会返回null值。

返回值：

返回INT类型，输入null，则返回null值。

示例如下：

```
quarter('1970-11-12 10:00:00')=4
quarter('1970-11-12')=4
```

MONTH

命令格式如下：

```
INT month(string date)
```

命令说明如下：

返回一个日期的月份。

参数说明：

date：String类型日期值，其他类型将返回异常。

返回值：

返回INT类型。

示例如下：

```
month('2014-09-01')=9
month('20140901')=null
```

DAY

命令格式如下：

```
INT day(string date)
```

命令说明如下：

返回一个日期的天。

参数说明：

date：String类型日期值（格式为yyyy-mm-dd、yyyy-mm-dd hh:mi:ss）其他类型将返回异常。

返回值：

返回INT类型。

示例如下：

```
day('2014-09-01')=1
day('20140901')=null
```

DAYOFMONTH

命令格式如下：

```
INT dayofmonth(date)
```

命令说明如下：

返回年/月/日中的具体日期。

例如2017年10月13日，执行命令`int dayofmonth(2017-10-13)`返回结果为13。

参数说明：

date：String类型日期值，其他类型将返回异常。

返回值：

返回INT类型。

示例如下：

```
dayofmonth('2014-09-01')=1
dayofmonth('20140901')=null
```

HOUR

命令格式如下：

```
INT hour(string date)
```

命令说明如下：

返回一个日期的小时。

参数说明：

date：String类型日期值，其他类型将返回异常。

返回值：

返回INT类型。

示例如下：

```
hour('2014-09-01 12:00:00')=12
hour('12:00:00')=12
hour('20140901120000')=null
```

MINUTE

命令格式如下：

```
INT minute(string date)
```

命令说明如下：

返回一个日期的分钟。

参数说明：

date：String类型日期值，其他类型将返回异常。

返回值：

返回INT类型。

示例如下：

```
minute('2014-09-01 12:30:00')=30
```

```
minute('12:30:00')=30
minute('20140901120000')=null
```

SECOND

命令格式如下：

```
INT second(string date)
```

命令说明如下：

返回一个日期的秒钟。

参数说明：

date：String类型日期值，其他类型将返回异常。

返回值：

返回INT类型。

示例如下：

```
second('2014-09-01 12:30:45')=45
second('12:30:45')=45
second('20140901123045')=null
```

CURRENT_TIMESTAMP

命令格式如下：

```
timestamp current_timestamp()
```

命令说明如下：

返回当前timestamp类型的时间戳，值不固定。

返回值：

返回timestamp类型。

示例如下：

```
select current_timestamp() from dual;--返回'2017-08-03 11:50:30.661'
```

ADD_MONTHS

命令格式如下：

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

命令说明如下：

返回开始日期startdate增加nummonths个月后的日期。

参数说明：

- **startdate**：String类型，格式至少包含年-月-日的日期，否则返回null值。
- **num_months**：Int型数值。

返回值：

返回String类型的日期，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
```

LAST_DAY

命令格式如下：

```
string last_day(string date)
```

命令说明如下：

返回该日期所在月份的最后一天日期。

参数说明：

date：String类型，格式为yyyy-MM-dd HH:mi:ss或yyyy-MM-dd。

返回值：

返回String类型的日期，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
```

NEXT_DAY

命令格式如下：

```
string next_day(string startdate, string week)
```

命令说明如下：

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。

参数说明：

- **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：

返回String类型的日期，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
```

MONTHS_BETWEEN

命令格式如下：

```
double months_between(datetime/timestamp/string date1, datetime/
timestamp/string date2)
```

命令说明如下：

返回日期date1和date2之间的月数。

参数说明：

- **date1**：Datetime/Timestamp/String类型，格式为yyyy-MM-dd HH:mi:ss或yyyy-MM-dd。
- **date2**：Datetime/Timestamp/String类型，格式为yyyy-MM-dd HH:mi:ss或yyyy-MM-dd。

返回值：

返回Double类型。

- 当date1晚于date2，返回值为正。当date2晚于date1，返回值为负。
- 当date1和date2分别对应两个月的最后一天，返回整数月，否则计算方式为date1-date2的天数除以31天。

示例如下：

```
months_between('1997-02-28 10:30:00', '1996-10-30')=3.9495967741935485
months_between('1996-10-30','1997-02-28 10:30:00' )=-3.9495967741
935485
months_between('1996-09-30','1996-12-31')=-3.0
```

3.9.2 数学函数

ABS

命令格式如下：

```
Double abs(Double number)
Bigint abs(Bigint number)
Decimal abs(Decimal number)
```

命令说明如下：

该函数用于返回number的绝对值。

参数说明：

number：当number为Double、Bigint或Decimal类型时。

- 输入为Bigint，返回Bigint。
- 输入为Double，返回Double 类型。
- 输入为Decimal，返回Decimal 类型。

若输入为String类型，会隐式转换为Double类型后参与运算，其它类型抛异常。

返回值：

返回Double、Bigint或Decimal类型，取决于输入参数的类型。若输入为null，则返回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，不再举例。

```
select abs(id) from tbl1;
-- 取tbl1表内id字段的绝对值
```

ACOS

命令格式如下：

```
Double acos(Double number)
Decimal acos(Decimal number)
```

命令说明如下：

该函数用于计算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，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型，值域在0 ~ π之间。若number为null，则返回null。

示例如下：

```
acos("0.87") = 0.5155940062460905
acos(0) = 1.5707963267948966
```

ASIN

命令格式如下：

```
Double asin(Double number)
```

```
Decimal asin(Decimal number))
```

命令说明如下：

该函数用于计算number的反正弦函数。

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，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回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
```

ATAN

命令格式如下：

```
Double atan(Double number)
```

命令说明如下：

该函数用于计算number的反正切函数。

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，其他类型抛异常。

返回值：

返回Double类型，值域在-π/2~π/2之间。若number为null，则返回null。

示例如下：

```
atan(1) = 0.7853981633974483
atan(-1) = -0.7853981633974483
```

CEIL

命令格式如下：

```
Bigint ceil(Double value)
```

```
Bigint ceil(Decimal value)
```

命令说明如下：

该函数返回不小于输入值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，其他类型抛异常。

返回值：

返回Bigint类型。任意一个参数输入为null，则返回null。

示例如下：

```
ceil(1.1) = 2
ceil(-1.1) = -1
```

CONV

命令格式如下：

```
String conv(String input, Bigint from_base, Bigint to_base)
```

命令说明如下：

该函数为进制转换函数。

参数说明：

- **input**：以String表示的要转换的整数值，接受Bigint、Double的隐式转换。
- **from_base**、**to_base**：以十进制表示的进制的值，可接受的的值为2、8、10和16。接受String及Double的隐式转换。

返回值：

返回String类型。任意一个参数输入为null，返回null。转换过程以64位精度工作，溢出时报异常。输入如果是负值，即以（-）开头，报异常。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'
```

COS

命令格式如下：

```
Double cos(Double number)
Decimal cos(Decimal number)
```

命令说明如下：

该函数用于计算number的余弦函数，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，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，则返回null。

示例如下：

```
cos(3.1415926/2)=2.6794896585028633e-8
cos(3.1415926)=-0.9999999999999986
```

COSH

命令格式如下：

```
Double cosh(Double number)
Decimal cosh(Decimal number)
```

命令说明如下：

该函数用于计算number的双曲余弦函数。

参数说明：

number：Double类型或Decimal类型。若输入为String类型或Bigint类型，会隐式转换为Double类型后，参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，返回null。

COT

命令格式如下：

```
Double cot(Double number)
Decimal cot(Decimal number)
```

命令说明如下：

该函数用于计算number的余切函数，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，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，则返回null。

EXP

命令格式如下：

```
Double exp(Double number)
 Decimal exp(Decimal number)
```

命令说明如下：

该函数用于计算number的指数函数。

返回值：

返回number的指数值。

参数说明：

number：Double类型或Decimal类型。若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，则返回null。

FLOOR

命令格式如下：

```
Bigint floor(Double number)
```

```
Bigint floor(Decimal number)
```

命令说明如下：

该函数用于向下取整。

返回值：

返回比number小的整数值。

参数说明：

number：Double类型或Decimal类型，若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Bigint类型。若number为null，则返回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
```

LN

命令格式如下：

```
Double ln(Double number)
Decimal ln(Decimal number)
```

命令说明如下：

该函数用于返回number的自然对数。

参数说明：

number：Double类型或Decimal类型。

- 若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。
- 若number为null，则返回null。若number为负数或零，则执行报错。

返回值：

返回Double类型或Decimal类型。

LOG

命令格式如下：

```
Double log(Double base, Double x)
Decimal log(Decimal base, Decimal x)
```

命令说明如下：

该函数用于返回以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，会隐式转换为Double类型后参与运算，其他类型抛异常。
- **x**：Double类型或Decimal类型，若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型的对数值。

- 若base和x中存在null，则返回null。
- 若base和x中某一个值为负数或 0，会引发异常。
- 若base为1（会引发一个除零行为），也会引发异常。

POW

命令格式如下：

```
Double pow(Double x, Double y)
Decimal pow(Decimal x, Decimal y)
```

命令说明如下：

该函数用于返回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，会隐式转换为Double类型后参与运算，其他类型抛异常。
- **Y**：Double类型或Decimal类型，若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若x或y为null，则返回null。

RAND

命令格式如下：

```
Double rand(Bigint seed)
```

命令说明如下：

该函数以seed为种子，返回Double类型的随机数，返回值区间是的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：

返回Double类型。

示例如下：

```
select rand() from dual;
select rand(1) from dual;
```

ROUND

命令格式如下：

```
Double round(Double number, [Bigint Decimal_places])
Decimal round(Decimal number, [Bigint Decimal_places])
```

命令说明如下：

该函数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，会隐式转换为Double类型后参与运算，其他类型抛异常。
- **Decimal_place**：Bigint类型常量，四舍五入计算到小数点后的位置，其他类型参数会引发异常。如果省略表示四舍五入到个位数，默认值为0。

返回值：

返回Double类型或Decimal类型。若**number**或**Decimal_places**为null，则返回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
```

SIN

命令格式如下所示：

```
Double sin(Double number)
Decimal sin(Decimal number)
```

命令说明如下：

该函数用于计算number的正弦函数，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，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，则返回null。

SINH

命令格式如下：

```
Double sinh(Double number)
Decimal sinh(Decimal number)
```

命令说明如下：

该函数用于计算number的双曲正弦函数。

参数说明：

number：Double类型或Decimal类型。若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，则返回null。

SQRT

命令格式如下：

```
Double sqrt(Double number)
Decimal sqrt(Decimal number)
```

命令说明如下：

该函数用于计算number的平方根。

参数说明：

number：Double类型或Decimal类型，必须大于0，小于0时引发异常。若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，则返回null。

TAN

命令说明如下：

```
Double tan(Double number)
Decimal tan(Decimal number)
```

命令说明如下：

该函数用于计算number的正切函数，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，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，则返回null。

TANH

命令格式如下：

```
Double tanh(Double number)
Decimal tanh(Decimal number)
```

命令说明如下：

该函数用于计算number的双曲正切函数。

参数说明：

number：Double类型或Decimal类型。若输入为String类型或Bigint类型，会隐式转换为Double类型后参与运算，其他类型抛异常。

返回值：

返回Double类型或Decimal类型。若number为null，则返回null。

TRUNC

命令格式如下：

```
Double trunc(Double number[, Bigint Decimal_places])
Decimal trunc(Decimal number[, Bigint Decimal_places])
```

命令说明如下：

该函数用于将输入值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，会隐式转换为Double类型后参与运算，其他类型抛异常。
- **Decimal_places**：Bigint类型常量，要截取到的小数点位置，其他类型参数会隐式转为Bigint，省略此参数时默认到截取到个位数。

返回值：

返回值类型为Double或Decimal类型。若**number**或**Decimal_places**为null，则返回null。

说明：

- 返回Double类型时，返回的结果的显示可能不符合预期，如示例`trunc(125.815, 1)`（这个Double类型显示问题任何系统都存在）。

- 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，则返回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
```

新扩展数学函数

升级到MaxCompute2.0后，产品扩展部分数学函数，新函数若用到新数据类型时，在使用新函数的SQL前，需要加一个set语句：

```
set odps.sql.type.system.odps2=true;
```

下文将为您详细介绍新扩展的函数。

LOG2

命令格式如下：

```
Double log2(Double number)
Double log2(Decimal number)
```

命令说明如下：

该函数用于以2为底，返回number的对数。

参数说明：

number：Double或Decimal类型。

返回值：

返回Double类型。若输入为0或null，则返回null。

示例如下：

```
log2(null)=null
log2(0)=null
```

```
log2(8)=3.0
```

LOG10

命令格式如下：

```
Double log10(Double number)
Double log10(Decimal number)
```

命令说明如下：

该函数用于以10为底，返回number的对数。

参数说明：

number：Double或Decimal类型。

返回值：

返回Double类型。若输入为0或null，则返回null。

示例如下：

```
log10(null)=null
log10(0)=null
log10(8)=0.9030899869919435
```

BIN

命令格式如下：

```
String bin(Bigint number)
```

命令说明如下：

该函数用于返回number的二进制代码表示。

参数说明：

number：Bigint类型。

返回值：

返回String类型。若输入为0，返回0，输入为null，则返回null。

示例如下：

```
bin(0)='0'
bin(null)='null'
```

```
bin(12)='1100'
```

HEX

命令格式如下：

```
String hex(Bigint number)
String hex(String number)
String hex(BINARY number)
```

命令说明如下：

该函数用于将整数或字符转换为十六进制格式。

参数说明：

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：

返回String类型。若输入为0，返回0，输入为null，则返回异常。

示例如下：

```
hex(0)=0
hex('abc')='616263'
hex(17)='11'
hex('17')='3137'
hex(null)异常返回失败
```

说明：

当输入参数为Binary类型时，请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

UNHEX

命令格式如下：

```
BINARY unhex(String number)
```

命令说明如下：

该函数用于返回十六进制字符串所代表的字符串。

参数说明：

number：为十六进制字符串。

返回值：

返回Binary类型，若输入0，返回失败，若输入为null，则返回null。

示例如下：

```
unhex('616263')='abc'
unhex(616263)='abc'
```

RADIANS

命令格式如下：

```
Double radians(Double number)
```

命令说明如下：

该函数用于将角度转换为弧度。

参数说明：

number：Double类型数据。

返回值：

返回Double类型，若输入为null，返回null。

示例如下：

```
radians(90)=1.5707963267948966
radians(0)=0.0
radians(null)=null
```

DEGREES

命令格式如下：

```
Double degrees(Double number)
Double degrees(Decimal number)
```

命令说明如下：

该函数用于将弧度转换为角度。

参数说明：

number：Double或Decimal类型数据。

返回值：

返回Double类型，若输入null，则返回null。

示例如下：

```
degrees(1.5707963267948966)=90.0
degrees(0)=0.0
degrees(null)=null
```

SIGN

命令格式如下：

```
Double sign(Double number)
Double sign(Decimal number)
```

命令说明如下：

该函数用于取输入数据的符号，1.0表示正，-1.0表示负，否则0.0。

参数说明：

number：Double或Decimal类型数据。

返回值：

返回Double类型，若输入0，则返回0.0，输入为null，则返回null。

示例如下：

```
sign(-2.5)=-1.0
sign(2.5)=1.0
sign(0)=0.0
sign(null)=null
```

E

命令格式如下：

```
Double e()
```

命令说明如下：

该函数用于返回e的值。

返回值：

返回Double类型。

示例如下：

```
e()=2.718281828459045
```

PI

命令格式如下：

```
Double pi()
```

命令说明如下：

该函数用于返回π的值。

返回值：

返回Double类型。

示例如下：

```
pi()=3.141592653589793
```

FACTORIAL

命令格式如下：

```
Bigint factorial(Int number)
```

命令说明如下：

该函数用于返回number的阶乘。

参数说明：

number：Int类型数据，且在[0..20]之间。

返回值：

返回Bigint类型，输入为0，则返回1，输入为null或[0..20]之外的值，返回null。

示例如下：

```
factorial(5)=120 --5!= 5*4*3*2*1=120
```

CBRT

命令格式如下：

```
Double cbrt(Double number)
```

命令说明如下：

该函数用于计算number的立方根。

参数说明：

number：Double类型数据。

返回值：

返回Double类型，输入为null，返回null。

示例如下：

```
cbrt(8)=2
cbrt(null)=null
```

SHIFTLEFT

命令格式如下：

```
Int shiftleft(Tinyint|Smallint|Int number1, Int number2)
Bigint shiftleft(Bigint number1, Int number2)
```

命令说明如下：

该函数用于按位左移（**<<**）。

参数说明：

- **number1**：Tinyint|Smallint|Int|Bigint整型数据。
- **number2**：Int整型数据。

返回值：

返回Int或Bingint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）
```

SHIFTRIGHT

命令格式如下：

```
Int shiftright(Tinyint|Smallint|Int number1, Int number2)
Bigint shiftright(Bigint number1, Int number2)
```

命令说明如下：

该函数用于按位右移（**>>**）。

参数说明：

- **number1**：Tinyint|Smallint|Int|Bigint整型数据。
- **number2**：Int整型数据。

返回值：

返回Int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）
```

SHIFTRIGHTUNSIGNED

命令格式如下：

```
Int shiftrightunsigned(Tinyint|Smallint|Int number1, Int number2)
Bigint shiftrightunsigned(Bigint number1, Int number2)
```

命令说明如下：

该函数用于无符号按位右移（**>>>**）。

参数说明：

- **number1**：Tinyint|Smallint|Int|Bigint整型数据。
- **number2**：Int整型数据。

返回值：

返回Int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)
```

3.9.3 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。

窗口函数的语法声明如下：

```
window_func() over (partition by [col1,col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] windowing_clause)
```

- **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，有以下两种方式：
  - 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窗口范围是从前或后第x行到当前行。
  - 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。

说明：

并非所有的窗口函数都可以用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。

COUNT

命令格式如下：

```
Bigint count([distinct] expr) over(partition by [col1, col2…]
 [order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

命令说明如下：

该函数用于计算计数值。

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：

返回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。
```

AVG

命令格式如下：

```
avg([distinct] expr) over(partition by [col1, col2…]
 [order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

命令说明如下：

该函数用于计算平均值。

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，Decimal类型。
  - 当输入值为String、Bigint类型时，会隐式转换到Double类型后参与运算，其它类型抛异常。
  - 当输入值为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：

返回Double类型。

说明：

指明distinct关键字时，不能写order by。

MAX

命令格式如下：

```
max([distinct] expr) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

命令说明如下：

该函数用于计算最大值。

参数说明：

- **expr**：除Boolean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：

返回值的类型同expr类型。

说明：

指明distinct关键字时，不能写order by。

MIN

命令格式如下：

```
min([distinct] expr) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

命令说明如下：

该函数用于计算最小值。

参数说明：

- **expr**：除Boolean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：

返回值类型同expr类型。

说明：

指明distinct关键字时，不能写order by。

MEDIAN

命令格式如下：

```
Double median(Double number1,number2...) over(partition by [col1, col2
…])
Decimal median(Decimal number1,number2...) over(partition by [col1,
col2…])
```

命令说明如下：

该函数用于计算中位数最小值。

参数说明：

- **number1,number1**…：Double类型或Decimal类型的1到255个数字。
  - 当输入值为String类型或Bigint类型，会隐式转换到Double类型后参与运算，其他类型抛异常。
  - 当输入值为null时，返回null。
  - 如果传入的参数是Double类型，会默认转成Double的Array。
- **partition by [col1, col2…]**：指定开窗口的列。

返回值：

返回值类型同Double类型。

STDDEV

命令格式如下：

```
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]…]] [windowing_clause])
```

命令说明如下：

该函数用于计算总体标准差。

参数说明：

- **expr**：Double类型或Decimal类型。
  - 当输入值为String类型或Bigint类型时，会隐式转换到Double类型后参与运算，其他类型抛异常。
  - 当输入值为null时，该行不参与计算。
  - 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：

输入值为Decimal类型时，返回Decimal类型，否则返回Double类型。

示例如下：

```
select window, seq, stddev_pop('1\01') over (partition by window order
 by seq) from dual;
```

说明：

- 当指定distinct关键字时，不能写order by。
- **stddev**还有一个别名函数**stddev_pop**，用法和**stddev**一样。

STDDEV_SAMP

命令格式如下：

```
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])
```

命令说明如下：

该函数用于计算样本标准差。

参数说明：

- **expr**：Double类型或Decimal类型。
  - 当输入值为String类型或Bigint类型时，会隐式转换到Double类型后参与运算，其他类型抛异常。
  - 当输入值为null时，该行不参与计算。
  - 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：

输入值为Decimal类型时，返回Decimal类型，否则返回Double类型。

说明：

指明distinct关键字时，不能写order by。

SUM

命令格式如下：

```
sum([distinct] expr) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

命令说明如下：

该函数用于计算汇总值。

参数说明：

- **expr**：Double类型、Decimal类型或Bigint类型。
  - 当输入值为String类型时，会隐式转换到Double类型后参与运算，其他类型抛异常。
  - 当输入值为null时，该行不参与计算。
  - 当指定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：

- 输入值为Bigint类型时，返回Bigint类型。
- 输入值为Decimal类型时，返回Decimal类型。
- 输入值为Double类型或String类型时，返回Double类型。

说明：

指明distinct关键字时，不能写order by。

DENSE_RANK

命令说明如下：

```
Bigint dense_rank() over(partition by [col1, col2…]
order by [col1[asc|desc], col2[asc|desc]…])
```

命令说明如下：

该函数用于计算连续排名。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：

返回Bigint类型。

示例如下：

假设表emp中的数据如下所示：

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

RANK

命令格式如下：

```
Bigint rank() over(partition by [col1, col2…]
order by [col1[asc|desc], col2[asc|desc]…])
```

命令说明如下：

该函数用于计算排名。col2相同的行数据获得排名顺序下降。

参数说明：

- **partition by [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：

返回Bigint类型。

示例如下：

假设表emp中的数据如下所示：

```
| 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          |
```

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

LAG

命令格式如下：

```
lag(expr , Bigint offset, default) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]])
```

命令说明如下：

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。输入值为String、Double到Bigint的隐式转换，offset>0。
- **default**：当offset指定的范围越界时的缺省值，常量，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：

返回值类型同expr类型。

LEAD

命令格式如下：

```
lead(expr , Bigint offset, default) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]])
```

命令说明如下：

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**：可选，Bigint类型常量。输入值为String、Decimal、Double到Bigint的隐式转换，offset>0。
- **default**：可选，当offset指定的范围越界时的缺省值，常量。
- **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：

返回值类型同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;
```

PERCENT_RANK

命令格式如下：

```
percent_rank() over(partition by [col1, col2…]
order by [col1[asc|desc], col2[asc|desc]…])
```

命令说明如下：

该函数用于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：

返回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。

ROW_NUMBER

命令格式如下：

```
row_number() over(partition by [col1, col2…]
order by [col1[asc|desc], col2[asc|desc]…])
```

命令说明如下：

该函数用于计算行号，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：

返回Bigint类型。

示例如下：

假设表emp中的数据如下所示：

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

CLUSTER_SAMPLE

命令格式如下：

```
boolean cluster_sample([Bigint x, Bigint y])
over(partition by [col1, col2..])
```

命令说明如下：

该函数用于分组抽样。

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：

返回Boolean类型。

示例如下：

假设表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;
+-----+------------+
| key | value      |
+-----+------------+
| groupa | 0.167537127858695 |
| groupb | 0.135751224393789 |
+-----+------------+
```

NTILE

命令格式如下：

```
BIGINT ntile(BIGINT n) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause]))
```

命令说明如下：

用于将分组数据按照顺序切分成n片，并返回当前切片值，如果切片不均匀，默认增加第一个切片的分布。

参数说明：

n：Bigint数据类型。

返回值：

返回Bigint类型。

示例如下：

假设表emp中的数据如下所示：

```
| 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按SAL高到低切分为3组，并获得职工自己组内的序号。

```
select  deptno, ename,sal,NTILE(3) OVER(PARTITION BY deptno ORDER BY
sal desc) AS nt3 from emp;
--执行结果如下
+------------+-------+------------+------------+
| deptno     | ename | sal        | nt3        |
+------------+-------+------------+------------+
| 10         | JACCKA | 5000.0     | 1          |
| 10         | KING  | 5000.0     | 1          |
| 10         | WELAN | 2450.0     | 2          |
| 10         | CLARK | 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     | 2          |
| 20         | SMITH | 800.0      | 3          |
| 30         | BLAKE | 2850.0     | 1          |
| 30         | ALLEN | 1600.0     | 1          |
| 30         | TURNER | 1500.0     | 2          |
| 30         | MARTIN | 1250.0     | 2          |
| 30         | WARD  | 1250.0     | 3          |
| 30         | JAMES | 950.0      | 3          |
+------------+-------+------------+------------+
```

3.9.4 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。

COUNT

命令格式如下：

```
bigint count([distinct|all] value)
```

命令说明如下：

该函数用于计算记录数。

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：

返回Bigint类型。

示例如下：

```
-- 如表tbla有列col1类型为 Bigint
+------+
| COL1 |
+------+
| 1 |
+------+
| 2 |
+------+
| NULL |
+------+
select count(*) from tbla; -- 值为3,
select count(col1) from tbla; -- 值为2
```

聚合函数可以和group by一同使用，例如：假设存在表test_src，存在如下两列：key string类型，value double类型。

```
-- test_src的数据为
+-----+-------+
| key | value |
+-----+-------+
| a | 2.0 |
+-----+-------+
| a | 4.0 |
+-----+-------+
| b | 1.0 |
+-----+-------+
| b | 3.0 |
+-----+-------+
-- 此时执行如下语句，结果为：
select key, count(value) as count from test_src group by key;
+-----+-------+
| key | count |
+-----+-------+
| a | 2 |
+-----+-------+
| b | 2 |
+-----+-------+
```

```
-- 聚合函数将对相同key值得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。
```

AVG

命令格式如下：

```
double avg(double value)
decimal avg(decimal value)
```

命令说明如下：

该函数用于计算平均值。

参数说明：

value：Double类型或Decimal 类型，若输入为String或Bigint，会隐式转换到Double类型后参与运算，其它类型抛异常。当value值为NULL 时，该行不参与计算。Boolean类型不允许参与计算。

返回值：

若输入Decimal类型，返回Decimal类型，其他合法输入类型返回Double类型。

示例如下：

```
-- 如表tbla有一列value，类型为 Bigint
+-------+
| value |
+-------+
| 1 |
| 2 |
| NULL |
+-------+
-- 则对该列计算avg结果为(1+2)/2=1.5
select avg(value) as avg from tbla;
+------+
| avg |
+------+
| 1.5 |
+------+
```

MAX

命令格式如下：

```
max(value)
```

命令说明如下：

该函数用于计算最大值。

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：

返回值的类型与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
```

MIN

命令格式如下：

```
MIN(value)
```

命令说明如下：

该函数用于计算最小值。

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。

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
```

MEDIAN

命令格式如下：

```
double median(double number)
decimal median(decimal number)
```

命令说明如下：

该函数用于计算中位数。

参数说明：

number：Double类型或者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，其他类型报错。

返回值：

返回Double类型或者Decimal类型。

示例如下：

```
-- 如表tbla有一列value，类型为 Bigint
+------+
| value|
+------+
| 1 |
+------+
| 2 |
+------+
| 3 |
+------+
| 4 |
+------+
| 5 |
+------+
select MEDIAN(value) from tbla; -- 返回值为 3.0
```

STDDEV

命令格式如下：

```
double stddev(double number)
decimal stddev(decimal number)
```

命令说明如下：

该函数用于计算总体标准差。

参数说明：

number：Double类型或者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，其他类型时报错。

返回值：

返回Double类型或者Decimal类型。

示例如下：

```
-- 如表tbla有一列value，类型为 Bigint
+------+
| value|
+------+
| 1 |
+------+
| 2 |
+------+
| 3 |
+------+
| 4 |
+------+
| 5 |
+------+
select STDDEV(value) from tbla; -- 返回值为 1.4142135623730951
```

STDDEV_SAMP

命令格式如下：

```
double stddev_samp(double number)
decimal stddev_samp(decimal number)
```

命令说明如下：

该函数用于计算样本标准差。

参数说明：

number：Double类型或者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，其他类型时报错。

返回值：

返回Double类型或者Decimal类型。

示例如下：

```
-- 如表tbla有一列value，类型为 Bigint
+------+
| value|
+------+
| 1 |
```

```
+------+
| 2 |
+------+
| 3 |
+------+
| 4 |
+------+
| 5 |
+------+
select STDDEV_SAMP(value) from tbla; -- 返回值为 1.5811388300841898
```

SUM

命令格式如下：

```
sum(value)
```

命令说明如下：

该函数用于计算汇总值。

参数说明：

value：Double、Decimal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时，返回Bigint，输入为Double或String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
```

WM_CONCAT

命令格式如下：

```
string wm_concat(string separator, string str)
```

命令说明如下：

该函数用指定的spearator做分隔符，链接str中的值。

参数说明：

- separator：String类型常量，分隔符。其他类型或非常量将引发异常。
- str：String类型，若输入为Bigint，Double或者Datetime类型，会隐式转换为String后参与运算，其它类型报异常。

返回值：

返回String类型。

说明：

语句`select wm_concat(',', name) from test_src;`中，若test_src为空集合，此条语句返回NULL值。

COLLECT_LIST

命令格式如下：

```
ARRAY collect_list(col)
```

命令说明如下：

该函数在给定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：

返回ARRAY类型。

说明：

请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

COLLECT_SET

命令格式如下：

```
ARRAY collect_set(col)
```

命令说明如下：

该函数在给定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：

返回ARRAY类型。

说明：

请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

VARIANCE/VAR_POP

命令格式如下：

```
DOUBLE variance(col)
DOUBLE var_pop(col)
```

命令说明如下：

该函数用于求指定数字列的方差。

参数说明：

col：数值类型列，其他类型返回null。

返回值：

返回double类型。

示例如下：

test表的c1列数据如下所示：

```
+------------+
| c1 |
+------------+
```

```
| 8 |
| 9 |
| 10 |
| 11 |
+------------+
```

执行下述语句求该表c1列的方差。

```
select variance(c1) from test;
--或
select var_pop(c1) from test;
--执行结果如下：
+------------+
| _c0 |
+------------+
| 1.25 |
+------------+
```

说明：

若涉及新数据类型，请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

VAR_SAMP

命令格式如下：

```
DOUBLE var_samp(col)
```

命令说明如下：

该函数用于求指定数字列的样本方差。

参数说明：

col：数值类型列，其他类型返回null。

返回值：

返回double类型。

示例如下：

test表的c1列数据如下所示：

```
+------------+
| c1 |
+------------+
| 8 |
| 9 |
| 10 |
| 11 |
```

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

执行下述语句求该表c1列的方差。

```
select var_samp(c1) from test;
--执行结果如下：
+------------+
| _c0 |
+------------+
| 1.6666666666666667 |
+------------+
```

说明：

若涉及新数据类型，请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

COVAR_POP

命令格式如下：

```
DOUBLE covar_pop(col1, col2)
```

命令说明如下：

该函数用于求指定两个数字列的总体协方差。

参数说明：

col1/col2：数值类型列，其他类型返回null。

示例如下：

test表（c1 bigint，c2 bigint）数据如下所示：

```
+------------+------------+
| c1 | c2 |
+------------+------------+
| 3 | 2 |
| 14 | 5 |
| 50 | 14 |
| 26 | 75 |
+------------+------------+
```

执行下述语句求c1，c2的总体协方差。

```
select covar_pop(c1,c2) from test;
--结果如下：
+------------+
| _c0 |
+------------+
| 123.49999999999997|
```

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

说明：

若涉及新数据类型，请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

COVAR_SAMP

命令格式如下：

```
DOUBLE covar_samp(col1, col2)
```

命令说明如下：

该函数用于求指定两个数字列的样本协方差。

参数说明：

col1/col2：数字类型列，其他类型返回null。

示例如下：

test表（c1 bigint，c2 bigint）数据如下所示：

```
+------------+------------+
| c1 | c2 |
+------------+------------+
| 3 | 2 |
| 14 | 5 |
| 50 | 14 |
| 26 | 75 |
+------------+------------+
```

执行下述语句求c1，c2的样本协方差。

```
select covar_samp(c1,c2) from test;
--结果如下：
+------------+
| _c0 |
+------------+
| 164.66666666666663|
+------------+
```

说明：

若涉及新数据类型，请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

PERCENTILE

命令格式如下：

```
DOUBLE percentile(BIGINT col, p)
array percentile(BIGINT col, array(p1 [, p2]...))
```

命令说明如下：

返回指定列精确的第p位百分数，p必须在0和1之间。

> 说明：
> 只有整数值才能计算出真正的百分位数。

参数说明：

- col：bigint类型。
- p：必须是大于等0小于等于1。

示例如下：

test表的c1列数据如下所示：

```
+------------+
| c1 |
+------------+
| 8 |
| 9 |
| 10 |
| 11 |
+------------+
```

执行下述语句求该表c1列的第P位百分数。

```
select percentile(c1,0),percentile(c1,0.3),percentile(c1,0.5),
percentile(c1,1) from var_test;
--执行结果如下：
+------------+------------+------------+------------+
| _c0 | _c1 | _c2 | _c3 |
+------------+------------+------------+------------+
| 8.0 | 8.9 | 9.5 | 11.0 |
+------------+------------+------------+------------+
--执行
select percentile(c1,array(0,0.3,0.5,1))from var_test;
结果如下：
+------+
| _c0 |
+------+
| [8, 8.9, 9.5, 11] |--返回array的话看起来有问题，不应该是8.0，11.0么？
```

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

3.9.5 其他函数

CAST

命令格式如下：

```
cast(expr as <type>)
```

命令说明如下：

该函数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引发异常。

说明：

- cast(double as bigint)，将Double值转换成Bigint。
- cast(string as bigint)在将字符串转为Bigint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。

COALESCE

命令格式如下：

```
coalesce(expr1, expr2, ...)
```

命令说明如下：

该函数用于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：

expri是要测试的值，所有这些值类型必须相同或为Null，否则会引发异常。

返回值：

返回值类型和参数类型相同。

说明：

至少要有一个参数，否则引发异常。

DECODE

命令格式如下：

```
decode(expression, search, result[, search, result]...[, default])
```

命令说明如下：

该函数用于实现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引发异常。所有的search和expression类型必须一致，否则报异常。
- 如果decode中的search选项有重复时且匹配时，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;
```

说明：

- 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。
- 上述示例中，当customer_id的值为Null时，decode函数返回N/A。

GET_IDCARD_AGE

命令格式如下：

```
get_idcard_age(idcardno)
```

命令说明如下：

该函数用于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：

返回Bigint类型，输入为Null，返回Null。如果当前年份减去出生年份差值大于100，返回Null。

GET_IDCARD_BIRTHDAY

命令格式如下：

```
get_idcard_birthday(idcardno)
```

命令说明如下：

该函数用于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，则返回Null。

返回值：

返回Datetime类型，输入为Null，返回Null。

GET_IDCARD_SEX

命令格式如下：

```
get_idcard_sex(idcardno)
```

命令说明如下：

该函数用于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，则返回Null。

返回值：

返回String类型，输入为Null，返回Null。

GREATEST

命令格式如下：

```
greatest(var1, var2, …)
```

命令说明如下：

该函数用于返回输入参数中最大的一个。

参数说明：

var1，var2可以为Bigint，Double，Decimal，Datetime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，Bigint，Decimal，String之间的比较转为Double类型。
  - String，Datetime的比较转为Datetime类型。

- 不允许其它的隐式转换。

ORDINAL

命令格式如下：

```
ordinal(bigint nth, var1, var2, …)
```

命令说明如下：

该函数用于将输入变量按从小到大排序后，返回nth指定位置的值。

参数说明：

- nth：Bigint类型，指定要返回的位置为Null时，返回Null。
- var1，var2：类型可以为Bigint，Double，Datetime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。
- 当有类型转换时：
  - Double，Bigint，String之间的转换返回Double类型。
  - String，Datetime之间的转换返回Datetime类型。
  - 不允许其它的隐式转换。
- Null为最小。

示例如下：

```
ordinal(3, 1, 3, 2, 5, 2, 4, 6) = 2
```

LEAST

命令格式如下：

```
least(var1, var2, …)
```

命令说明如下：

该函数用于返回输入参数中最小的一个。

参数说明：

var1，var2可以为Bigint，Double，Decimal，Datetime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。
- 当有类型转换时：
  - Double，Bigint，String之间的转换返回Double类型。
  - String，Datetime之间的转换返回Datetime类型。
  - Decimal和Double，Bigint，String之间比较时转为Decimal类型。
  - 不允许其它的隐式类型转换。
- Null为最小。

MAX_PT

命令格式如下：

```
max_pt(table_full_name)
```

命令说明如下：

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），您必须对此表有读权限。

返回值：

返回最大的一级分区的值。

示例如下：

假设tbl是分区表，该表对应的分区如下，且都有数据文件：

```
pt ='20120901'
```

```
pt ='20120902'
```

则以下语句中max_pt返回值为'20120902'，MaxCompute SQL语句读出pt='20120902'分区下的数据。

```
select * from tbl where pt=max_pt('myproject.tbl');
```

说明：

如果只是用alter table的方式新加了一个分区，但是此分区中并无任何数据文件，则此分区不会做为返回值。

UUID

命令格式如下：

```
string uuid()
```

命令说明如下：

该函数用于返回一个随机ID，示例样式为`29347a88-1e57-41ae-bb68-a9edbdd94212`。

SAMPLE

命令格式如下：

```
boolean sample(x, y, column_name)
```

命令说明如下：

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，其它类型或小于等于0时抛异常，若y&gt，x也抛异常。x，y任一输入为Null时，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，则报异常。

返回值：

返回Boolean类型。

说明：

为避免Null值带来的数据倾斜，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
```

CASE WHEN表达式

MaxCompute提供两种case when的语法格式，如下所示：

```
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。

根据shop_name的不同情况得出所属区域，示例如下：

```
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类型只有Bigint，Double，统一转为Double后，再返回。
- 如果result类型中有String类型，统一转为String后，再返回。如果不能转则报错（如Boolean型）。
- 除此之外不允许其它类型之间的转换。

IF

命令格式如下：

```
if(testCondition, valueTrue, valueFalseOrNull)
```

命令说明如下：

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：要判断的表达式，Boolean类型。
- valueTrue：表达式testCondition为True时，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;
返回值：
+------------+
| _c0 |
+------------+
| 200 |
+------------+
```

NVL

命令格式如下：

```
nvl(T value, T default_value)
```

命令说明如下：

如果value值为NULL，返回default_value，否则返回value。

示例如下：

如表t_data的3个列分别为c1 string，c2 bigint，c3 datetime。表中数据如下所示：

```
+----+------------+------------+
| c1 | c2 | c3 |
+----+------------+------------+
| NULL | 20 | 2017-11-13 05:00:00 |
| ddd | 25 | NULL |
| bbb | NULL | 2017-11-12 08:00:00 |
| aaa | 23 | 2017-11-11 00:00:00 |
+----+------------+------------+
```

通过nvl函数将c1中为null的值输出为’00000’，c2中为null的值输出为0，c3中为null的值输出为’-‘，执行如下SQL：

```
SELECT nvl(c1,'00000'),nvl(c2,0) nvl(c3,'-') from nvl_test;
--结果如下：
+-----+------------+-----+
| _c0 | _c1 | _c2 |
+-----+------------+-----+
| bbb | 0 | 2017-11-12 08:00:00 |
| ddd | 25 | - |
| 00000 | 20 | 2017-11-13 05:00:00 |
| aaa | 23 | 2017-11-11 00:00:00 |
+-----+------------+-----+
```

MaxCompute2.0扩展支持的其他函数类型

SPLIT

命令格式如下：

```
split(str, pat)
```

命令说明如下：

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;
结果如下：
+------+
| _c0 |
```

```
+------+
| [a, b, c] |
+------+
```

说明：

请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

EXPLODE

命令格式如下：

```
explode (var)
```

命令说明如下：

该函数用于将一行数据转为多行的UDTF。

- 如果var是array，则将列中存储的array转为多行。
- 如果var是map，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，令一列存储value。

参数说明：

`var`：`array<T>` 类型或者 `map<K, V>` 类型。

返回值：

返回转换后的行。

说明：

UDTF在使用上有以下限制：

- 在一个select中只能有一个UDTF，不可出现其它的列。
- 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
```

说明：

请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

MAP

命令格式如下：

```
MAP map(K key1, V value1, K key2, V value2, ...)
```

命令说明如下：

该函数用于使用给定的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：

返回`map<K,V>`类型。

示例如下：

```
select  map('a',123,'b',456) from dual;
```

结果如下：

```
{a:123, b:456}
```

说明：

请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

MAP_KEYS

命令格式如下：

```
ARRAY map_keys(map<K, V> )
```

命令说明如下：

该函数用于将参数map中的所有key作为数组返回。

参数说明：

`map<K , V>`：map类型的数据。

返回值：

返回`array<K>`类型，输入Null，则返回Null。

示例如下：

```
select map_keys(map('a',123,'b',456)) from dual;
结果如下：
[a, b]
```

说明：

请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

MAP_VALUES

命令格式如下：

```
ARRAY map_values(map<K, V>)
```

命令说明如下：

该函数用于将参数map中的所有values作为数组返回。

参数说明：

`map<K , V>`：map类型的数据。

返回值：

返回`array<V>`类型，输入Null，返回Null。

示例如下：

```
select map_values(map('a',123,'b',456));
--结果如下：
[123, 456]
```

说明：

请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

ARRAY

命令格式如下：

```
ARRAY array(value1,value2, ...)
```

命令说明如下：

该函数用于使用给定的value构造array。

参数说明：

value：value可为任意类型，但是所有value的类型必须一致。

返回值：

返回array类型。

示例如下：

```
select array(123,456,789) from dual;
```

结果如下：

```
[123, 456, 789]
```

说明：

请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

SIZE

命令格式如下：

```
INT size(map)
INT size(array)
```

命令说明如下：

- `size(map<K , V>)`返回给定map中K/V对数。
- `size(array<T>)`返回给定的array中的元素数目。

参数说明：

- `map<K, V>`：map类型的数据。
- `array<T>`：array类型的数据。

返回值：

返回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
```

说明：

请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

ARRAY_CONTAINS

命令格式如下：

```
boolean array_contains(ARRAY<T> a,value v)
```

命令说明如下：

该函数用于检测给定array a中是否包含v。

参数说明：

- a：array类型的数据。
- v：给出的v必须与array数组中的数据类型一致。

返回值：

返回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
```

说明：

请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

SORT_ARRAY

命令格式如下：

```
ARRAY sort_array(ARRAY<T>)
```

命令说明如下：

该函数用于为给定的数组排序。

参数说明：

ARRAY<T>：array类型的数据，数组中的数据可为任意类型。

返回值：

返回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;
结果如下：
[a, b, c, f] [2, 4, 5, 5, 7, 8] [他, 你, 我]
```

说明：

请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

POSEXPLODE

命令格式如下：

```
posexplode(ARRAY<T>)
```

命令说明如下：

该函数用于将给定的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<T>：array类型的数据，数组中的数据可为任意类型。

返回值：

返回表生成的函数。

示例如下：

```
select posexplode(array('a','c','f','b')) from dual;
结果如下：
+------------+-----+
| pos | val |
+------------+-----+
| 0 | a |
| 1 | c |
| 2 | f |
| 3 | b |
+------------+-----+
```

说明：

请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

STRUCT

命令格式如下：

```
STRUCT struct(value1,value2, ...)
```

函数说明：

该函数用于使用给定value列表建立struct。

参数说明：

value：各value可为任意类型。

返回值：

返回`STRUCT<col1:T1, col2:T2, ...>`类型。field的名称依次为col1，col2，…。

示例如下：

```
select struct('a',123,'ture',56.90) from dual;
结果如下：
{col1:a, col2:123, col3:ture, col4:56.9}
```

说明：

请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

NAMED_STRUCT

命令格式如下：

```
STRUCT named_struct(string name1, T1 value1, string name2, T2 value2
, ...)
```

函数说明：

该函数用于使用给定的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：

返回STRUCT<name1:T1, name2:T2, ...>类型，生成struct的field的名称依次为name1，name2，…。

示例如下：

```
select named_struct('user_id',10001,'user_name','LiLei','married','F
','weight',63.50) from dual;
结果如下：
{user_id:10001, user_name:LiLei, married:F, weight:63.5}
```

说明：

请在用到该函数的SQL语句前加set odps.sql.type.system.odps2=true;，并与SQL一起提交运行，以便正常使用新数据类型。

INLINE

命令格式如下：

```
inline(array<struct<f1:T1, f2:T2, ...>>)
```

命令说明如下：

该函数用于将给定的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<f1:T1, f2:T2, ...>：数组中的value可为任意类型。

返回值：

返回表生成的函数。

示例如下：

```
select inline(array(named_struct('user_id',10001,'user_name','LiLei','
married','F','weight',63.50))) from dual;
结果如下：
+------------+-----------+---------+------------+
| user_id | user_name | married | weight |
+------------+-----------+---------+------------+
| 10001 | LiLei | F | 63.5 |
+------------+-----------+---------+------------+
```

说明：

请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

TRANS_ARRAY

命令格式如下：

```
trans_array (num_keys, separator, key1,key2,…,col1, col2,col3) as (
key1,key2,…,col1, col2)
```

命令说明如下：

用于将一行数据转为多行的UDTF，将列中存储的以固定分隔符格式分隔的数组转为多行。

参数说明：

- num_keys： Bigint类型常量，必须>=0。在转为多行时作为转置key的列的个数。
- Key：是指在将一行转为多行时，在多行中重复的列。
- separator：String类型常量，用于将字符串拆分成多个元素的分隔符。为空时报异常。
- keys：转置时作为key的列， 个数由num_keys指定。如果num_keys指定所有的列都作为key（即num_keys等于所有列的个数），则只返回一行。
- cols： 要转为行的数组，keys之后的所有列视为要转置的数组，必须为String类型，存储的内容是字符串格式的数组，如”Hangzhou;Beijing;shanghai”，是以”;”分隔的数组。

返回值：

转置后的行，新的列名由as指定。作为key的列类型保持不变，其余所有的列是String类型。拆分成的行数以个数多的数组为准，不足的补NULL。

说明：

UDTF使用上的限制如下：

- 所有作为key的列必须处在前面，而要转置的列必须放在后面。
- 在一个select中只能有一个udtf，不可以再出现其它的列。
- 不可以与group by/cluster by/distribute by/sort by一起使用。

示例如下：

表中的数据如下所示：

```
Login_id LOGIN_IP LOGIN_TIME
wangwangA 192.168.0.1,192.168.0.2 20120101010000,20120102010000
```

使用trans_array函数：

```
trans_array(1, ",", login_id, login_ip, login_time) as (login_id,
login_ip,login_time)
```

产生的数据如下所示：

```
Login_id Login_ip Login_time
wangwangA 192.168.0.1 20120101010000
wangwangA 192.168.0.2 20120102010000
```

如果表中的数据如下所示：

```
Login_id LOGIN_IP LOGIN_TIME
wangwangA 192.168.0.1,192.168.0.2 20120101010000
```

则对数组中不足的数据补NULL：

```
Login_id Login_ip Login_time
wangwangA 192.168.0.1 20120101010000
wangwangA 192.168.0.2 NULL
```

3.9.6 字符串函数

CHAR_MATCHCOUNT

命令格式如下：

```
bigint char_matchcount(string str1, string str2)
```

命令说明如下：

该函数用于计算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。
- bigint：返回值为bigint类型。任一输入为NULL返回NULL。

示例如下：

```
char_matchcount('abd','aabc') = 2
-- str1中得两个字符串'a','b'在str2中出现过
```

CHR

命令格式如下：

```
string chr(bigint ascii)
```

命令说明如下：

该函数用于将给定ASCII码ascii转换成字符。

参数说明：

- ascii：Bigint类型ASCII值，若输入为string类型或double类型或decimal类型会隐式转换到bigint类型后参与运算，其它类型抛异常。
- String：返回值为String类型。参数范围是0~255，超过此范围会引发异常。输入值为NULL返回NULL。

CONCAT

命令格式如下：

```
string concat(string a, string b...)
```

命令说明如下：

该函数的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等为String类型，若输入为bigint，decimal，double或datetime类型会隐式转换为string后参与运算，其它类型报异常。
- String：返回值为String类型。如果没有参数或者某个参数为NULL，结果均返回NULL。

示例如下：

```
concat('ab','c') = 'abc'
concat() = NULL
concat('a', null, 'b') = NULL
```

GET_JSON_OBJECT

命令格式如下：

```
STRING GET_JSON_OBJECT(STRING json,STRING path)
```

命令说明如下：

该函数用于在一个标准json字符串中，按照path抽取指定的字符串。

参数说明：

- json： String类型，标准的json格式字符串。
- path：**String**类型，用于描述在**json**中的**path**，以**$**开头。关于新实现中json path的说明，请参见：*JsonPath*，$表示根节点，“.”表示child，”[number]”表示数组下标，对于数组，格式为key[sub1][sub2][sub3]……，[*]返回整个数组，*不支持转义。
- String：返回值为String类型。

说明：

- 如果json为空或者非法的json格式，返回NULL；
- 如果path为空或者不合法（json中不存在）返回NULL；
- 如果json合法，path也存在则返回对应字符串。

示例一如下：

```
+----+
json
+----+
{"store":
{"fruit":[{"weight":8,"type":"apple"},{"weight":9,"type":"pear"}],
"bicycle":{"price":19.95,"color":"red"}
},
"email":"amy@only_for_json_udf_test.net",
"owner":"amy"
}
```

通过以下查询，可以提取 json 对象中的信息：

```
odps> SELECT get_json_object(src_json.json, '$.owner') FROM src_json;
amy
```

```
odps> SELECT get_json_object(src_json.json, '$.store.fruit\[0]') FROM
src_json;
{"weight":8,"type":"apple"}
odps> SELECT get_json_object(src_json.json, '$.non_exist_key') FROM
src_json;
NULL
```

示例二如下：

```
get_json_object('{"array":[["aaaa",1111],["bbbb",2222],["cccc",3333
]]}','$.array[1][1]')= "2222"
get_json_object('{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["
h0","h1","h2"]},"iii":"jjj"}','$.ccc.hhh[*]') = "["h0","h1","h2"]"
get_json_object('{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["
h0","h1","h2"]},"iii":"jjj"}','$.ccc.hhh[1]') = "h1"
```

INSTR

命令格式如下：

```
bigint instr(string str1, string str2[, bigint start_position[, bigint
 nth_appearance]])
```

命令说明如下：

该函数用于计算子串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，decimal，double或datetime类型会隐式转换为string后参与运算，其它类型报异常。
- str2：String类型，要搜索的子串，若输入为bigint，decimal，double或datetime类型会隐式转换为string后参与运算，其它类型报异常。
- start_position：bigint类型，其它类型会抛异常，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为其它类型或小于等于0会抛异常。
- bigint：返回值为bigint类型。

说明：

- 如果在str1中未找到str2，返回0。
- 任一输入参数为NULL返回NULL。
- 如果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
```

IS_ENCODING

命令格式如下：

```
boolean is_encoding(string str, string from_encoding, string
to_encoding)
```

命令说明如下：

用于判断输入字符串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，输入为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。输入为NULL返回NULL。
- boolean：返回值为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
```

KEYVALUE

命令格式如下：

```
KEYVALUE(STRING srcStr,STRING split1,STRING split2, STRING key)
KEYVALUE(STRING srcStr,STRING key) //split1 = ";"，split2 = ":"
```

命令说明如下：

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输入待拆分的字符串。
- 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。
```

LENGTH

命令格式如下：

```
bigint length(string str)
```

命令说明如下：

返回字符串str的长度。

参数说明：

- str：String类型，若输入为bigint，double，decimal或datetime类型会隐式转换为string后参与运算，其它类型报异常。
- bigint：返回值为bigint类型。若str是NULL返回NULL。如果str非UTF-8编码格式，返回-1。

示例如下：

```
length('hi! 中国') = 6
```

LENGTHB

命令格式如下：

```
bigint lengthb(string str)
```

命令说明如下：

返回字符串str的以字节为单位的长度。

参数说明：

- str：String类型，若输入为bigint，double，decimal或者datetime类型会隐式转换为string后参与运算，其它类型报异常。
- bigint：返回值为bigint类型。若str是NULL返回NULL。

示例如下：

```
lengthb('hi! 中国') = 10
```

MD5

命令格式如下：

```
string md5(string value)
```

命令说明如下：

计算输入字符串value的md5值。

参数说明：

- value：String 类型，如果输入类型是bigint，double，decimal或者datetime会隐式转换成string类型参与运算，其它类型报异常。输入为NULL，返回NULL。
- String：返回值为String类型。

REGEXP_EXTRACT

命令格式如下：

```
string regexp_extract(string source, string pattern[, bigint
occurrence])
```

命令说明如下：

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，待搜索的字符串。
- pattern：String类型常量，pattern为空串时抛异常，pattern中如果没有指定group，抛异常。
- occurrence：Bigint类型常量，必须 >=0，其它类型或小于0时抛异常，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。
- String：返回值为String类型，任一输入为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')
```

```
-- 异常返回，pattern中没有指定group
```

REGEXP_INSTR

命令格式如下：

```
bigint regexp_instr(string source, string pattern[,
bigint start_position[, bigint nth_occurrence[, bigint return_option
]])
```

命令说明如下：

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。任一输入参数为NULL时返回NULL。

参数说明：

- source：String类型，待搜索的字符串。
- pattern：String类型常量，pattern为空串时抛异常。
- start_position：Bigint类型常量，搜索的开始位置。不指定时默认值为1，其它类型或小于等于0的值会抛异常。
- 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。小于等于0或者其它类型抛异常。
- return_option：Bigint类型常量，值为0或1，其它类型或不允许的值会抛异常。0表示返回匹配的开始位置，1表示返回匹配的结束位置。
- bigint：返回值为bigint类型。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。

示例如下：

```
regexp_instr("i love www.taobao.com", "o[[:alpha:]]{1}", 3, 2) = 14
```

REGEXP_REPLACE

命令格式如下：

```
string regexp_replace(string source, string pattern, string replace_st
ring[, bigint occurrence])
```

命令说明如下：

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为空串时抛异常。
- 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。其它类型或小于0抛异常。可缺省，默认值为0。
- String：返回值为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 有匹配，返回NULL，replace_string为NULL 但pattern不匹配，则返回原串。

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。
```

REGEXP_SUBSTR

命令格式如下：

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

命令说明如下：

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为空串时抛异常。
- start_position：Bigint常量，必须大于0。其它类型或小于等于0时抛异常，不指定时默认为1，表示从 source的第一个字符开始匹配。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，其它类型或小于等于0时抛异常。不指定时默认为1，表示返回第一次匹配的子串。不指定时默认为1，表示返回第一次匹配的子串。
- String：返回值为String类型。任一输入参数为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"
```

REGEXP_COUNT

命令格式如下：

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

命令说明如下：

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，其它类型报异常。
- pattern：String类型常量，要匹配的模型，pattern为空串时抛异常，其它类型报异常。
- start_position：Bigint类型常量，必须大于0。其它类型或小于等于0时抛异常，不指定时默认为1，表示从source的第一个字符开始匹配。
- bigint：返回值为bigint类型。没有匹配时返回0。任一输入参数为NULL返回NULL。

示例如下：

```
regexp_count('abababc', 'a.c') = 1
regexp_count('abcde', '[[:alpha:]]{2}', 3) = 1
```

SPLIT_PART

命令格式如下：

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

命令说明如下：

依照分隔符separator拆分字符串str，返回从第start部分到第end部分的子串（闭区间）。

参数说明：

- Str：String类型，要拆分的字符串。如果是bigint，double，decimal或者datetime类型会隐式转换到string类型后参加运算，其它类型报异常。
- Separator：String类型常量，拆分用的分隔符，可以是一个字符，也可以是一个字符串，其它类型会引发异常。
- start：Bigint类型常量，必须大于0。非常量或其它类型抛异常。返回段的开始编号（从1开始），如果没有指定end，则返回start指定的段。
- end：Bigint类型常量，大于等于start，否则抛异常。返回段的截止编号，非常量或其他类型会引发异常。可省略，缺省时表示最后一部分。
- String：返回值为String类型。若任意参数为NULL，返回NULL；若separator为空串，返回原字符串str。

说明：

- 如果separator不存在于str中，且start指定为1，返回整个str。若输入为空串，输出为空串。
- 如果start的值大于切分后实际的分段数，例如：字符串拆分完有6个片段，但start大于6，返回空串""。
- 若end大于片段个数，按片段个数处理。

示例如下：

```
split_part('a,b,c,d', ',', 1) = 'a'
split_part('a,b,c,d', ',', 1, 2) = 'a,b'
split_part('a,b,c,d', ',', 10) = ''
```

SUBSTR

命令格式如下：

```
string substr(string str, bigint start_position[, bigint length])
```

命令说明如下：

返回字符串str从start_position开始往后数，长度为length的子串。

参数说明：

- str：String类型，若输入为bigint，decimal，double或者datetime类型会隐式转换为string后参与运算，其它类型报异常。

- 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，往前数依次就是-2，-3…，其它类型抛异常。
- length：Bigint类型，大于0，其它类型或小于等于 0 抛异常。子串的长度。
- String：返回值为String类型。若任一输入为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"
```

SUBSTRING

命令格式如下：

```
string  substring(string|binary str, int start_position[, int length])
```

命令说明如下：

返回字符串str从start_position开始往后数，长度为length的子串。

参数说明：

- str：String | Binary类型，若输入为其它类型返回null或报错。
- start_position：Int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，往前数依次就是-2，-3…，其它类型抛异常。
- length：Bigint类型，大于0，其它类型或小于等于0抛异常。子串的长度。
- String：返回值为String类型。若任一输入为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：

```
substring('abc', 2) = 'bc'
substring('abc', 2, 1) ='"b'
substring('abc',-2,2)='bc'
substring('abc',-3,2)='ab'
```

```
substring(BIN(2345),2,3)='001'
```

说明：

当输入的数据类型为binary时，请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

TOLOWER

命令格式如下：

```
string tolower(string source)
```

命令说明如下：

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，double，decimal或者datetime类型会隐式转换为string后参与运算，其它类型报异常。
- String：返回值为String类型。输入为NULL时返回NULL。

示例如下：

```
tolower("aBcd") = "abcd"
tolower("哈哈Cd") = "哈哈cd"
```

TOUPPER

命令格式如下：

```
string toupper(string source)
```

命令说明如下：

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，double，decimal或者datetime类型会隐式转换为string后参与运算，其它类型报异常。
- String：返回值为String类型。输入为NULL时返回NULL。

示例如下：

```
toupper("aBcd") = "ABCD"
toupper("哈哈Cd") = "哈哈CD"
```

TO_CHAR

命令格式如下：

```
string to_char(boolean value)
string to_char(bigint value)
string to_char(double value)
string to_char(decimal value)
```

命令说明如下：

将Boolean类型、bigint类型、decimal类型或者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、decimal类型或者double类型输入，其它类型抛异常。对 datetime类型的格式化输出请参考另一同名函数TO_CHAR 。
- String：返回值为String类型。如果输入为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
```

TRIM

命令格式如下：

```
string trim(string str)
```

命令说明如下：

将输入字符串str去除左右空格。

参数说明：

- str：String类型，若输入为bigint，decimal，double或者datetime类型会隐式转换为string后参与运算，其它类型报异常。
- String：返回值为String类型。输入为NULL时返回NULL。

LTRIM

命令格式如下：

```
string ltrim(string str)
```

命令说明如下：

将输入的字符串str去除左边空格。

参数说明：

- str：String类型，若输入为bigint，decimal，double或者datetime类型会隐式转换为string后参与运算，其它类型报异常。
- String：返回值为String类型。输入为NULL时返回NULL。

示例如下：

```
select ltrim(' abc ') from dual;
返回：
+-----+
| _c0 |
+-----+
| abc |
+-----+
```

RTRIM

命令格式如下：

```
string rtrim(string str)
```

命令说明如下：

将输入的字符串str去除右边空格。

参数说明：

- str：String类型，若输入为bigint，decimal，double或者datetime类型会隐式转换为string后参与运算，其它类型报异常。
- String：返回值为String类型。输入为NULL时返回NULL。

示例如下：

```
select rtrim('a abc ') from dual;
返回：
+-----+
| _c0 |
+-----+
```

```
| a abc |
+-----+
```

REVERSE

命令格式如下：

```
STRING REVERSE(string str)
```

命令说明如下：

返回倒序字符串。

参数说明：

- str：String类型，若输入为bigint，double，decimal或datetime类型会隐式转换为string后参与运算，其它类型报异常。
- String：返回值为String类型。输入为NULL时返回NULL。

示例如下：

```
select reverse('abcedfg') from dual;
返回：
+-----+
| _c0 |
+-----+
| gfdecba |
+-----+
```

SPACE

命令格式如下：

```
STRING SPACE(bigint n)
```

命令说明如下：

空格字符串函数，返回长度为n的字符串。

参数说明：

- n: bigint类型。长度不超过2M。如果为空，则抛异常。
- String：返回值为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。
```

REPEAT

命令格式如下：

```
STRING REPEAT(string str, bigint n)
```

命令说明如下：

返回重复n次后的str字符串。

参数说明：

- str：String类型，若输入为bigint，double，decimal或datetime类型会隐式转换为string后参与运算，其它类型报异常。
- n: Bigint 类型。长度不超过2M。如果为空，则抛异常。
- String：返回值为String类型。

示例如下：

```
select repeat('abc',5) from lxw_dual;
返回：abcabcabcabcabc
```

ASCII

命令格式如下：

```
Bigint ASCII(string str)
```

命令说明如下：

返回字符串str第一个字符的ascii码。

参数说明：

- str：String类型，若输入为bigint，double，decimal或datetime类型会隐式转换为string后参与运算，其它类型报异常。
- Bigint：返回值为Bigint类型。

示例如下：

```
select ascii('abcde') from dual;
返回值：97
```

MaxCompute2.0扩展支持的字符串函数如下：

CONCAT_WS

命令格式如下：

```
string concat_ws(string SEP, string a, string b...)
string concat_ws(string SEP, array)
```

命令说明如下：

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类型的分隔符，若不指定，返回的值异常。
- a/b等：String类型，若输入为bigint，decimal，double或datetime类型会隐式转换为string后参与运算，其它类型报异常。

返回值：

为String类型。如果没有参数或者某个参数为NULL，结果均返回NULL。

示例如下：

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

LPAD

命令格式如下：

```
string lpad(string a, int len, string b)
```

命令说明如下：

用b字符串将a字符串向左补足到len位。

> 说明：
> 需在使用该函数的sql语句前加语句`set odps.sql.type.system.odps2=true`，否则会报错。

参数说明：

- len：int整型。
- a/b等：String类型。

返回值：

为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
```

RPAD

命令格式如下：

```
string rpad(string a, int len, string b)
```

命令说明如下：

用b字符串将a字符串向右补足到len位。

说明：

需在使用该函数的sql语句前加语句`set odps.sql.type.system.odps2=true`,否则会报错。

参数说明：

- len：int整型。
- a/b等：String类型。

返回值：

为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
```

REPLACE

命令格式如下：

```
string replace(string a, string OLD, string NEW)
```

命令说明如下：

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：

参数说明：参数都为string类型。

返回值：

为String类型，若某个参数为null，则返回null。

示例如下：

```
replace('ababab','abab','12')='12ab'
replace('ababab','cdf','123')='ababab'
replace('123abab456ab',null,'abab')=null
```

SOUNDEX

命令格式如下：

```
string soundex(string a)
```

命令说明如下：

将普通字符串转换成soundex字符串。

参数说明：为string类型。

返回值：为String类型，若输入为null，则返回null。

示例如下：

```
soundex('hello')='H400'
```

SUBSTRING_INDEX

命令格式如下：

```
string substring_index(string a, string SEP, int count))
```

命令说明如下：

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为string类型，count为int类型。

返回值：

返回String类型，若输入为null，则返回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
```

TRANSLATE

命令格式如下：

```
string translate(string|varchar str1, string|varchar str2, string|
varchar str3)
```

命令说明如下：

将str1出现在str2中的字符串替换成str3中的字符串。

返回值：

返回string类型，若某个参数为null，则返回null。

示例如下：

```
translate('MaxComputer','puter','pute')='MaxCompute'
translate('aaa','b','c')='aaa'
translate('MaxComputer','puter',null)=null
```

说明：

若涉及新数据类型（varchar）请在用到该函数的SQL语句前加`set odps.sql.type.system.odps2=true;`，并与SQL一起提交运行，以便正常使用新数据类型。

3.10 UDF

3.10.2 Java UDF

MaxCompute 的 UDF 包括：UDF，UDAF 和 UDTF 三种函数，本文将重点介绍如何通过 Java 实现这三种函数。

参数与返回值类型

MaxCompute2.0 版本升级后，Java UDF 支持的数据类型从原来的 Bigint，String，Double，Boolean 扩展了更多基本的数据类型，同时还扩展支持了 ARRAY，MAP，STRUCT 等复杂类型。

- Java UDF 使用新基本类型的方法，如下所示：

- UDTF 通过 @Resolve 注解来获取 signature，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。
- UDAF暂时还不支持新数据类型。

- JAVA UDF 使用复杂类型的方法，如下所示：
  - UDTF 通过 @Resolve annotation 来指定 sinature，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。
  - UDAF暂时还不支持。

> 说明：
>
> - com.aliyun.odps.data.Struct 从反射看不出 field name 和 field type，所以需要用 @Resolve annotation 来辅助。即如果需要在 UDF 中使用 struct，要求在 UDF class 上也标注上 @Resolve 注解，这个注解只会影响参数或返回值中包含 com.aliyun.odps.data.Struct 的重载。
> - 目前 *class* 上只能提供一个 *@Resolve annotation*#因此一个 *UDF* 中带有 *struct* 参数或返回值的重载只能有一个。

MaxCompute 数据类型与 Java 类型的对应关系，如下所示：

MaxCompute Type	Java Type
Tinyint	java.lang.Byte
Smallint	java.lang.Short
Int	java.lang.Integer
Bigint	java.lang.Long
Float	java.lang.Float
Double	java.lang.Double
Decimal	java.math.BigDecimal

MaxCompute Type	Java Type
Boolean	java.lang.Boolean
String	java.lang.String
Varchar	com.aliyun.odps.data.Varchar
Binary	com.aliyun.odps.data.Binary
Datetime	java.util.Date
Timestamp	java.sql.Timestamp
Array	java.util.List
Map	java.util.Map
Struct	com.aliyun.odps.data.Struct

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 Array 类型对应的 Java 类型是 List，而不是数组。

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。这意味着您可以在 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 。

特别注意：不同的jar包最好不要有类名相同但实现功能逻辑不一样的类。如，UDF(UDAF/UDTF)： udf1、 udf2分别对应资源udf1.jar、udf2.jar，如果两个jar包里都包含一个com.aliyun.UserFunction.class 类，当同一个sql中同时使用到这两个udf时，系统会随机加载其中一个类，那么就会导致udf执行行为不一致甚至编译失败。

UDF 的示例如下：

```
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(); }
```

```
}
```

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。

UDAF

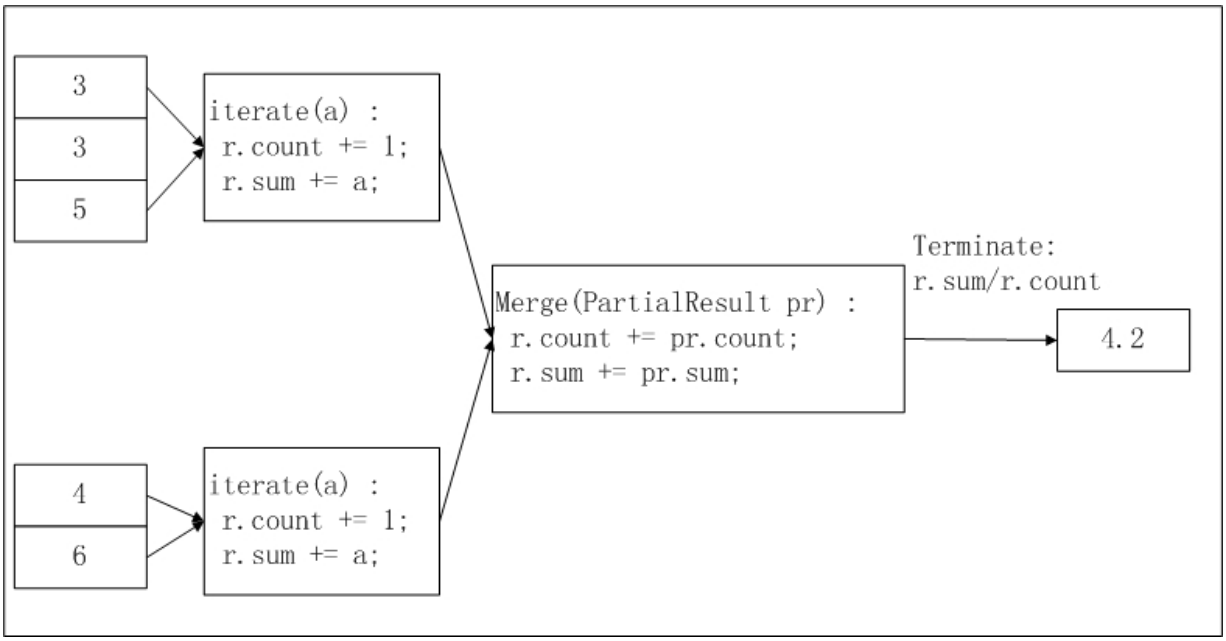
实现 Java UDAF 类需要继承 com.aliyun.odps.udf.Aggregator，并实现如下几个接口：

```
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，不能为null，但是args里面的元素可以为null，代表对应的输入数据是null
   * @throws UDFException
   */
  abstract public void iterate(Writable buffer, Writable[] args) throws UDFException;
  /**
   * 生成最终结果
   * @param buffer
   * @return Object UDAF的最终结果
   * @throws UDFException
   */
  abstract public Writable terminate(Writable buffer) throws UDFException;
  abstract public void merge(Writable buffer, Writable partial) throws UDFException;
}
```

其中最重要的是 iterate，merge 和 terminate 三个接口，UDAF 的主要逻辑依赖于这三个接口的实现。此外，还需要您实现自定义的 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（有关分片的描述请参见 *MapReduce*），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您手动配置完成。

UDAF 的计算过程分为两个阶段：

- 第一阶段：每个 worker 统计分片内数据的个数及汇总值，您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。

计算平均值的 UDAF 的代码示例，如下所示：

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.udf.Aggregator;
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.annotation.Resolve;
@Resolve("double->double")
public class AggrAvg extends Aggregator {
  private static class AvgBuffer implements Writable {
    private double sum = 0;
    private long count = 0;
    @Override
    public void write(DataOutput out) throws IOException {
      out.writeDouble(sum);
      out.writeLong(count);
    }
    @Override
    public void readFields(DataInput in) throws IOException {
      sum = in.readDouble();
      count = in.readLong();
```

```
    }
  }
  private DoubleWritable ret = new DoubleWritable();
  @Override
  public Writable newBuffer() {
    return new AvgBuffer();
  }
  @Override
  public void iterate(Writable buffer, Writable[] args) throws
UDFException {
    DoubleWritable arg = (DoubleWritable) args[0];
    AvgBuffer buf = (AvgBuffer) buffer;
    if (arg != null) {
      buf.count += 1;
      buf.sum += arg.get();
    }
  }
  @Override
  public Writable terminate(Writable buffer) throws UDFException {
    AvgBuffer buf = (AvgBuffer) buffer;
    if (buf.count == 0) {
      ret.set(0);
    } else {
      ret.set(buf.sum / buf.count);
    }
    return ret;
  }
  @Override
  public void merge(Writable buffer, Writable partial) throws
UDFException {
    AvgBuffer buf = (AvgBuffer) buffer;
    AvgBuffer p = (AvgBuffer) partial;
    buf.sum += p.sum;
    buf.count += p.count;
  }
}
```

说明：

- Writable 的 readFields 方法， 由于partial的writable对象是重用的，同一个对象的readFields方法会被调用多次。该方法期望每次调用的时候重置整个对象，如果对象中包含collection，需要清空。
- 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*。

UDTF

Java UDTF 需要继承 com.aliyun.odps.udf.UDTF 类。这个类需要实现 4 个接口，如下表所示：

接口定义	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 的程序示例，如下所示：

```
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);
         }
       }
     }
```

说明：

以上只是程序示例，关于如何在 MaxCompute 中运行 UDTF 的方法与 UDF 类似，详情请参见：运行 *UDF*。

在 SQL 中可以这样使用这个 UDTF，假设在 MaxCompute 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，如下所示：

```
+----+----+
| c0 | c1 |
+----+----+
| A  | 1  |
| B  | 1  |
| C  | 2  |
| D  | 2  |
+----+----+
```

使用说明

UDTF 在 SQL 中的常用方式如下：

```
select user_udtf(col0, col1, col2) as (c0, c1) from my_table;
      select user_udtf(col0, col1, col2) as (c0, c1) from
          (select * from my_table distribute by key sort by key) t;
      select reduce_udtf(col0, col1, col2) as (c0, c1) from
          (select col0, col1, col2 from
              (select map_udtf(a0, a1, a2, a3) as (col0, col1, col2)
from my_table) t1
           distribute by col0 sort by col0, col1) t2;
```

但使用 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 value, user_udtf(key) as mycol ...
```

- UDTF 不能嵌套使用。

```
select user_udtf1(user_udtf2(key)) as mycol...
```

- 不支持在同一个 select 子句中与 group by / distribute by / sort by 联用。

```
select user_udtf(key) as mycol ... group by mycol
```

其他 UDTF 示例

在 UDTF 中，您可以读取 MaxCompute 的 资源。利用 UDTF 读取 MaxCompute 资源的示例，如下所示：

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_reso
urce1").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（my_udtf）。

```
create function mp_udtf as com.aliyun.odps.examples.udf.UDTFResource
 using
'udtfexample1.jar, file_resource.txt, table_resource1, table_reso
urce2';
```

4. 在 MaxCompute 创建资源表 table_resource1、table_resource2 和物理表 tmp1，并插入相应的数据。

5. 运行该 UDTF。

```
select mp_udtf("10","20") as (a, b, fileResourceLineCount) from tmp1
;
返回：
+-------+------------+-------+
| a | b      | fileResourceLineCount |
+-------+------------+-------+
| 10    | 2          | fileResourceLineCount=3|tableResource1Record
Count=0|tableResource2RecordCount=0 |
| 10    | 2          | fileResourceLineCount=3|tableResource1Record
Count=0|tableResource2RecordCount=0 |
```

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

复杂数据类型示例

如以下代码，定义了一个有三个 overloads 的 UDF，其中第一个用了 array 作为参数，第二个用了 map 作为参数，第三个用了 struct。由于第三个 overloads 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;
  }
}
```

您可以直接将复杂类型传入 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;
```

hive udf兼容示例

MaxCompute 2.0支持了Hive风格的UDF，有部分的hive UDF、UDTF可以直接在MaxCompute上使用。

说明：

目前支持兼容的Hive版本为2.1.0; 对应Hadoop版本为2.7.2。UDF如果是在其他版本的Hive/Hadoop开发的，可能需要使用此Hive/Hadoop版本重新编译。

示例如下：

```
package com.aliyun.odps.compiler.hive;
import org.apache.hadoop.hive.ql.exec.UDFArgumentException;
import org.apache.hadoop.hive.ql.metadata.HiveException;
import org.apache.hadoop.hive.ql.udf.generic.GenericUDF;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInsp
ectorFactory;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
public class Collect extends GenericUDF {
  @Override
```

```
  public ObjectInspector initialize(ObjectInspector[] objectInspectors
) throws UDFArgumentException {
    if (objectInspectors.length == 0) {
      throw new UDFArgumentException("Collect: input args should >= 1
");
    }
    for (int i = 1; i < objectInspectors.length; i++) {
      if (objectInspectors[i] != objectInspectors[0]) {
        throw new UDFArgumentException("Collect: input oi should be
the same for all args");
      }
    }
    return ObjectInspectorFactory.getStandardListObjectInspector(
objectInspectors[0]);
  }
  @Override
  public Object evaluate(DeferredObject[] deferredObjects) throws
HiveException {
    List<Object> objectList = new ArrayList<>(deferredObjects.length);
    for (DeferredObject deferredObject : deferredObjects) {
      objectList.add(deferredObject.get());
    }
    return objectList;
  }
  @Override
  public String getDisplayString(String[] strings) {
    return "Collect";
  }
}
```

说明：

对应hive udf的使用请参考：

- *https://cwiki.apache.org/confluence/display/Hive/HivePlugins*
- *https://cwiki.apache.org/confluence/display/Hive/DeveloperGuide+UDTF*
- *https://cwiki.apache.org/confluence/display/Hive/GenericUDAFCaseStudy*

该udf可以将任意类型、数量的参数打包成array输出。假设输出jar包名为 test.jar：

```
--添加资源
Add jar test.jar;
--创建function
CREATE FUNCTION hive_collect as 'com.aliyun.odps.compiler.hive.Collect
' using 'test.jar';
--使用function
set odps.sql.hive.compatible=true;
select hive_collect(4y,5y,6y) from dual;
+------+
| _c0  |
+------+
| [4, 5, 6] |
```

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

说明：

该udf可以支持所有的类型，包括array，map，struct等复杂类型。

使用兼容hive的udf需要注意：

- MaxCompute的add jar命令会永久地在project中创建一个resource，所以创建udf时需要指定jar包，无法自动将所有jar包加入classpath。
- 在使用兼容的hive UDF的时候，需要在sql前加set语句`set odps.sql.hive.compatible=true;`语句，set语句和sql语句一起提交执行。
- 在使用兼容的hive UDF时，还要注意MaxCompute的*JAVA*沙箱限制。

3.10.3 Python UDF

MaxCompute UDF包括UDF、UDAF和UDTF三种函数，本文将重点介绍如何通过Python实现这三种函数。

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，以下行为会被禁止：

- 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，您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：

- 所有纯Python实现（不依赖扩展模块）的模块都可用。
- 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

- 部分模块功能受限。比如沙箱限制了您的代码最多可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。

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**。

说明：

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。

参数与返回值类型

参数与返回值的指定方式，如下所示：

```
@odps.udf.annotate(signature)
```

Python UDF目前支持的MaxCompute SQL数据类型包括Bigint、String、Double、Boolean和Datetime。SQL语句在执行之前，必须确定所有函数的参数类型和返回值类型。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，命令格式如下：

```
arg_type_list '->' type_list
arg_type_list: type_list | '*' | ''
type_list: [type_list ','] type
type: 'bigint' | 'string' | 'double' | 'boolean' | 'datetime'
```

- 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。

合法的signature示例如下：

```
'bigint,double->string'            # 参数为bigint、double，返回值为string

'bigint,boolean->string,datetime'  # UDTF参数为bigint、boolean，返回值为
string,datetime
```

```
'*->string'                    # 变长参数，输入参数任意，返回值为string
'->double'                     # 参数为空，返回值为double
```

Query语义解析阶段会检查到不符合函数签名的用法，抛出错误禁止执行。执行时，UDF函数的参数会以函数签名指定的类型传给您。您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：

ODPS SQL Type	Bigint	String	Double	Boolean	Datetime
Python Type	int	str	float	bool	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。您可以通过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无法转为Int ，不会抛出异常，而是返回None 。

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
```

说明：

Python UDF必须通过annotate指定函数签名。

UDAF

- 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（比如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，即将pbuffer merge到buffer中。
- BaseUDAF.terminate(buffer)：实现此方法将中间值buffer转换为MaxCompute SQL的基本类型。

UDAF求平均值的示例如下：

```
@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]
```

UDTF

- 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，便会输出一条记录。 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。

UDTF的示例如下：

```
#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)
```

说明：

Python UDTF也可以不加annotate指定参数类型和返回值类型。这样，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成Str类型。

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。

- odps.distcache.get_cache_file(resource_name)：
  - 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，会抛出异常。
  - 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：

```
@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)
```

- odps.distcache.get_cache_table(resource_name)：

  - 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，会抛出异常。
  - 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)
```

3.11 附录

3.11.1 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。

如下表达方式均可：

```
"I'm a happy manong."
'I\'m a happy manong.'
```

在 MaxCompute SQL 中，反斜线“\”是转义符，用来表达字符串中的特殊字符，或将其后跟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 值转为相应的字符。

对于以下情况，会将其解释为特殊字符：

转义	字符
\b	backspace
\t	tab
\n	newline
\r	carriage-return
\’	单引号
\”	双引号
\ \	反斜线
\;	分号
\Z	control-Z
\0或\00	结束符

```
select length('a\tb') from dual;
```

结果是 3，表示字符串里实际有三个字符，“\t”被视为一个字符。在转义符后的其它字符被解释为其本身。

```
select 'a\ab',length('a\ab') from dual;
```

结果是“aab”，3。“\a”被解释成了普通的“a”。

3.11.2 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_。

```
    'abcd' like 'ab%' -- true
    'abcd' like 'ab\%' -- true
    'ab\_cde' like 'ab\\_c%'; -- true
```

说明：

关于字符串的字符集，目前 MaxCompute SQL 支持 UTF-8 的字符集，如果数据是以其它格式编码，可能计算出的结果不正确。

3.11.3 正则表达式规范

MaxCompute SQL中的正则表达式采用的是PCRE的规范，匹配时按字符进行，支持的元字符如下表所示：

元字符	说明
^	行首
$	行尾
.	任意字符
*	匹配零次或多次
+	匹配1次或多次
?	匹配零次或1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。
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,例中模式匹配a或b
[a-d]	匹配a，b，c，d任一字符
[^ab]	^表示非，匹配任一非a非b的字符
[::]	见下表POSIX字符组
\	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

POSIX字符组	说明	范围
[[:alnum:]]	字母字符和数字字符	[a-zA-Z0-9]
[[:alpha:]]	字母	[a-zA-Z]

POSIX字符组	说明	范围
[[:ascii:]]	ASCII字符	[\x00-\x7F]
[[:blank:]]	空格字符和制表符	[\t]
[[:cntrl:]]	控制字符	[\x00-\x1F\x7F]
[[:digit:]]	数字字符	[0-9]
[[:graph:]]	空白字符之外的字符	[\x21-\x7E]
[[:lower:]]	小写字母字符	[a-z]
[[:print:]]	[:graph:]和空白字符	[\x20-\x7E]
[[:punct:]]	标点符号	[][!”#$%&’()*+,./:;<=>? @\^_`{|}~-]
[[:space:]]	空白字符	[\t\r\n\v\f]
[[:upper:]]	大写字母字符	[A-Z]
[[:xdigit:]]	十六进制字符	[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。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，如果在要匹配字符“\”， 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 | false |
```

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

> 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这两个字符时，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 |
+---------+------+
```

3.11.4 保留字

本文将为您介绍 MaxCompute SQL 中的所有保留字。

> 说明：
>
> - 在对表、列或分区进行命名时，请不要使用保留字，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
  DECIMAL
    BUCKET    BUCKETS    BY    CASCADE    CASE    CAST    CFILE
    CHANGE    CLUSTER    CLUSTERED    CLUSTERSTATUS    COLLECTION
COLUMN    COLUMNS
    COMMENT    COMPUTE    CONCATENATE    CONTINUE    CREATE    CROSS
  CURRENT
    CURSOR    DATA    DATABASE    DATABASES    DATE    DATETIME
DBPROPERTIES
    DEFERRED    DELETE    DELIMITED    DESC    DESCRIBE    DIRECTORY
  DISABLE
    DISTINCT    DISTRIBUTE    DOUBLE    DROP    ELSE    ENABLE    END
    ESCAPED    EXCLUSIVE    EXISTS    EXPLAIN    EXPORT    EXTENDED
  EXTERNAL
    FALSE    FETCH    FIELDS    FILEFORMAT    FIRST    FLOAT
FOLLOWING
    FORMAT    FORMATTED    FROM    FULL    FUNCTION    FUNCTIONS
GRANT
    GROUP    HAVING    HOLD_DDLTIME    IDXPROPERTIES    IF    IMPORT
  IN
    INDEX    INDEXES    INPATH    INPUTDRIVER    INPUTFORMAT    INSERT
    INT
```

```
    INTERSECT    INTO    IS    ITEMS    JOIN    KEYS    LATERAL
    LEFT    LIFECYCLE    LIKE    LIMIT    LINES    LOAD    LOCAL
    LOCATION    LOCK    LOCKS    LONG    MAP    MAPJOIN    MATERIALIZ
ED
    MINUS    MSCK    NOT    NO_DROP    NULL    OF    OFFLINE
    ON    OPTION    OR    ORDER    OUT    OUTER    OUTPUTDRIVER
    OUTPUTFORMAT    OVER    OVERWRITE    PARTITION    PARTITIONED
PARTITIONPROPERTIES    PARTITIONS
    PERCENT    PLUS    PRECEDING    PRESERVE    PROCEDURE    PURGE
RANGE
    RCFILE    READ    READONLY    READS    REBUILD    RECORDREADER
RECORDWRITER
    REDUCE    REGEXP    RENAME    REPAIR    REPLACE    RESTRICT
REVOKE
    RIGHT    RLIKE    ROW    ROWS    SCHEMA    SCHEMAS    SELECT
    SEMI    SEQUENCEFILE    SERDE    SERDEPROPERTIES    SET    SHARED
   SHOW
    SHOW_DATABASE    SMALLINT    SORT    SORTED    SSL    STATISTICS
  STORED
    STREAMTABLE    STRING    STRUCT    TABLE    TABLES    TABLESAMPLE
   TBLPROPERTIES
    TEMPORARY    TERMINATED    TEXTFILE    THEN    TIMESTAMP
TINYINT    TO
    TOUCH    TRANSFORM    TRIGGER    TRUE    UNARCHIVE    UNBOUNDED
  UNDO
    UNION    UNIONTYPE    UNIQUEJOIN    UNLOCK    UNSIGNED    UPDATE
  USE
    USING    UTC    UTC_TMESTAMP    VIEW    WHEN    WHERE    WHILE
DIV
```

4 MapReduce

4.1 概要

4.1.2 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（比如 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， 针对上述的情况，可以在 Reduce 后直接执行下一次的 Reduce 操作，而不需要中间插入一个 Map 操作。因此，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，比如 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）。这样的好处是：您可以复用之前的 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 I/O 模型。

与 *MaxCompute* 相比，MR2 在 Map/Reduce 等函数编写方式上基本一致。较大的不同点发生在作业时。更多详情请参见 扩展 *MapReduce* 示例。

4.1.3 开源兼容MapReduce

MaxCompute（原ODPS）有一套原生的MapReduce编程模型和接口，简单来说，这套接口的输入输出都是MaxCompute中的Table，处理的数据以Record为组织形式，它可以很好地描述Table中的数据处理过程。

但是与社区的Hadoop相比，编程接口差异较大。Hadoop用户如果要将原来的Hadoop MR作业迁移到MaxCompute的MR中执行，需要重写MR的代码，使用MaxCompute的接口进行编译和调试，运行正常后再打成一个Jar包才能放到MaxCompute的平台来运行。这个过程十分繁琐，需要耗费很

多的开发和测试人力。如果能够完全不改或者少量地修改原来的Hadoop MapReduce代码，便可在MaxCompute平台上运行，将会比较理想。

现在MaxCompute平台提供了一个Hadoop MapReduce到MaxCompute MR的适配工具，已经在一定程度上实现了Hadoop MapReduce作业的二进制级别的兼容，即您可以在不改代码的情况下通过指定一些配置，便可将原来在Hadoop上运行的MapReduce Jar包拿过来直接运行在MaxCompute上。您可通过此处下载开发插件。目前该插件处于测试阶段，暂时还不能支持您自定义comparator和自定义key类型。

下文将以WordCount程序为例，为您介绍HadoopMR插件的基本使用方式。

说明：

- 更多开源兼容性的介绍请参见开源版本*SDK*兼容性。
- 更多Hadoop MapReduce SDK的介绍请参见*MapReduce*官方文档。

下载HadoopMR插件

请单击此处下载插件，包名为hadoop2openmr-1.0.jar。

说明：

这个Jar包中已经包含hadoop-2.7.2版本的相关依赖，在作业的Jar包中请不要携带Hadoop的依赖，避免版本冲突。

准备Jar包

编译导出WordCount的Jar包（wordcount_test.jar），WordCount程序的源码如下所示：

```
package com.aliyun.odps.mapred.example.hadoop;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.IOException;
import java.util.StringTokenizer;
public class WordCount {
    public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(Object key, Text value, Context context
        ) throws IOException, InterruptedException {
```

```
            StringTokenizer itr = new StringTokenizer(value.toString
());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }
    public static class IntSumReducer
        extends Reducer<Text,IntWritable,Text,IntWritable> {
        private IntWritable result = new IntWritable();
        public void reduce(Text key, Iterable<IntWritable> values,
            Context context
        ) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

准备测试数据

1. 创建输入表和输出表。

   ```
   create table if not exists wc_in(line string);
   create table if not exists wc_out(key string, cnt bigint);
   ```

2. 通过Tunnel命令将数据导入输入表中。

   需要导入文本文件data.txt的数据内容如下：

   ```
   hello maxcompute
   ```

```
hello mapreduce
```

您可通过MaxCompute客户端的Tunnel命令将data.txt的数据导入wc_in中，如下所示：

```
tunnel upload data.txt wc_in;
```

准备表与HDFS文件路径的映射关系配置

配置文件命名为：wordcount-table-res.conf

```
{
  "file:/foo": {
    "resolver": {
      "resolver": "com.aliyun.odps.mapred.hadoop2openmr.resolver.
TextFileResolver",
      "properties": {
          "text.resolver.columns.combine.enable": "true",
          "text.resolver.seperator": "\t"
      }
    },
    "tableInfos": [
      {
        "tblName": "wc_in",
        "partSpec": {},
        "label": "__default__"
      }
    ],
    "matchMode": "exact"
  },
  "file:/bar": {
    "resolver": {
      "resolver": "com.aliyun.odps.mapred.hadoop2openmr.resolver.
BinaryFileResolver",
      "properties": {
          "binary.resolver.input.key.class" : "org.apache.hadoop.io.
Text",
          "binary.resolver.input.value.class" : "org.apache.hadoop.io.
LongWritable"
      }
    },
    "tableInfos": [
      {
        "tblName": "wc_out",
        "partSpec": {},
        "label": "__default__"
      }
    ],
    "matchMode": "fuzzy"
  }
}
```

配置项说明：

整个配置是一个JSON文件，描述HDFS上的文件与MaxCompute上的表之间的映射关系，一般要配置输入和输出两部分，一个HDFS路径对应一个resolver配置，tableInfos配置以及matchMode配置。

- resolver：用于配置如何对待文件中的数据，目前有com.aliyun.odps.mapred.hadoop2openmr.resolver.TextFileResolver和com.aliyun.odps.mapred.hadoop2ope个内置的resolver可以选用。除了指定好resolver的名字，还需要为相应的resolver配置一些properties指导它正确的进行数据解析。
  - TextFileResolver：对于纯文本的数据，输入输出都会当成纯文本对待。当作为输入resolver配置时，需要配置的properties有：text.resolver.columns.combine.enable和text.resolver.seperator，当text.resolver.columns.combine.enable配置为true时，会把输入表的所有列按照text.resolver.seperator指定的分隔符组合成一个字符串作为输入。否则，会把输入表的前两列分别作为key，value。
  - BinaryFileResolver：可以处理二进制的数据，自动将数据转换为MaxCompute可以支持的数据类型，如：Bigint，Bool，Double等。当作为输出resolver配置时，需要配置的properties有：binary.resolver.input.key.class和binary.resolver.input.value.class，分别代表中间结果的key和value类型。
- tableInfos：您配置HDFS对应的MaxCompute表，目前只支持配置表的名字tblName，而partSpec和label请保持和示例一致。
- matchMode：路径的匹配模式，可选项为exact和fuzzy，分别代表精确匹配和模糊匹配，如果设置为fuzzy，则可以通过正则来匹配HDFS的输入路径。

作业提交

使用MaxCompute命令行工具odpscmd提交作业。MaxCompute命令行工具的安装和配置方法请参见客户端用户手册。在odpscmd运行如下命令：

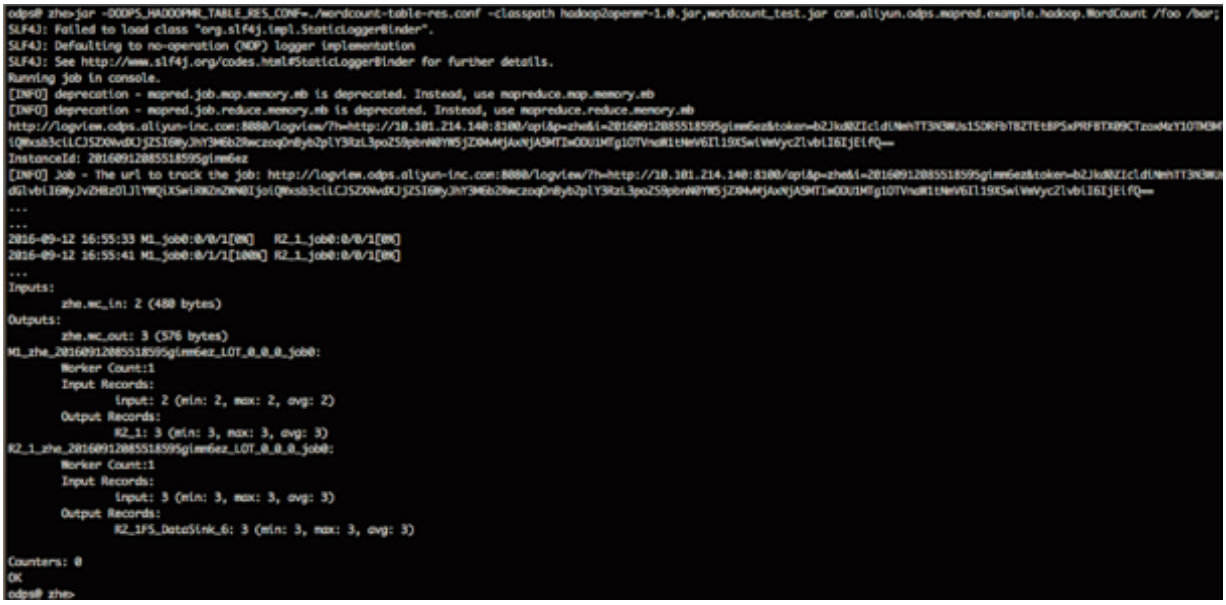
```
jar -DODPS_HADOOPMR_TABLE_RES_CONF=./wordcount-table-res.conf -
classpath hadoop2openmr-1.0.jar,wordcount_test.jar com.aliyun.odps.
mapred.example.hadoop.WordCount /foo /bar;
```

说明：

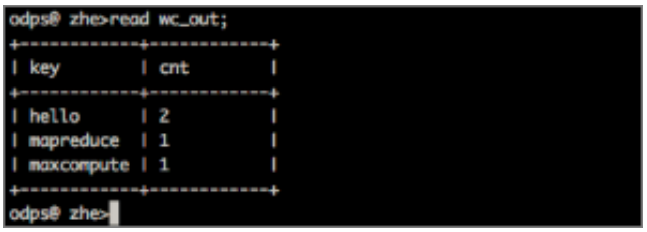
- wordcount-table-res.conf是配置了/foo /bar的映射。
- wordcount_test.jar包是您的Hadoop MapReduce的jar包。
- com.aliyun.odps.mapred.example.hadoop.WordCount是您要运行的作业类名。
- /foo /bar是指在HDFS上的路径，在JSON配置文件中映射成了wc_in和wc_out。
- 配置好映射后，您需要通过datax或DataWorks数据集成功能，手动导入Hadoop HDFS的输入文件到wc_in进行MR计算，并手动导出结果数据wc_out到您的HDFS输出目录/bar。

- 此处假设已经将hadoop2openmr-1.0.jar、wordcount_test.jar和wordcount-table-res.conf放置到odpscmd的当前目录，否则在指定配置和-classpath的路径时需要做相应的修改。

运行过程如下图所示：



当作业运行完成后，便可查看结果表wc_out的内容，验证作业是否成功结束，结果是否符合预期。



4.2 功能介绍

4.2.2 基本概念

Map/Reduce

Map 和 Reduce 分别支持对应的 map/reduce 方法，setup 及 cleanup 方法。setup 方法在 map/reduce 方法之前调用，每个 Worker 调用且仅调用一次。

cleanup 方法在 map/reduce 方法之后调用，每个 Worker 调用且仅调用一次。

相关的使用示例请参见 示例程序。

排序

支持将 Map 输出的 key record 中的某几列作为排序（Sort）列，不支持您自定义的比较器（comparator）。您可以在排序列中选择某几列作为 Group 列，不支持您自定义的 Group 比较器。Sort 列一般用来对您的数据进行排序，而 Group 列一般用来进行二次排序。

相关的使用示例请参见 二次排序源代码。

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。

归并

归并（Combiner）函数将 Shuffle 阶段相邻的 Record 进行归并。您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。

相关的使用示例请参见 示例程序。

4.2.3 输入与输出

- MaxCompute MapReduce 的输入、输出，支持 MaxCompute 内置类型的 Bigint，Double，String，Datetime 及 Boolean 类型，不支持您自定义类型。
- 接受多表输入，且输入表的 Schema 可以不同。在 map 函数中，您可以获取当前 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。

有关输入输出的使用示例请参见 多路输入输出示例。

4.2.4 资源使用

您可以在 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，以供您的代码使用。

相关的使用示例请参见 资源使用示例。

4.2.5 本地运行

基本阶段介绍

本地运行前提：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。

本地运行结束后：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，不需重复下载。

本地运行和分布式环境运行差异

在本地运行的过程中，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，MaxCompute 会限制引用资源的大小，详情请参见 *MR* 限制汇总。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，没有此限制。

本地运行示例

本地运行的示例如下：

```
    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
```

关于 WordCount 示例的代码介绍请参见 *WordCount 示例*。

如果您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，您在 Jar 命令中读写的表文件数据会被下载到这级目录下。
- <__schema__>文件中的内容表示表的元信息，其文件格式定义如下：

  ```
  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
  ```

  其中，列名与列类型使用英文冒号分隔，列与列之间使用英文逗号分隔。<__schema__>文件的最前面需要声明 Project 名字及 Table 名字，即`project_name.table_name`，并通过英文逗号与列的定义做分隔，即：`project_name.table_name,col1_name:col1_type,col2_name:col2_type,……`。
- tables 目录中 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，如下所示：

```
my_project.wc_in,key:STRING,value:STRING
```

data 的文件内容，如下所示：

```
0,2
```

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。如果再次运行此示例，将直接使用 wc_in 目录下的数据，不会再次下载。

说明：

仅在 MapReduce 的本地运行模式下支持从 MaxCompute 中下载数据的功能，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，如下所示：

```
my_project.wc_out,key:STRING,cnt:BIGINT
```

data 的文件内容，如下所示：

```
0,1
2,1
```

客户端会从 MaxCompute 中下载 wc_out 表的元信息，并保存到《*_schema_*》文件中。而本地运行后，生成的结果数据，则保存到 data 文件中。

说明：

- 您也可以自行编辑 《*_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。

4.3 示例程序

4.3.1 WordCount示例

测试准备

1. 准备好测试程序的 Jar 包，假设名字为 mapreduce-examples.jar，本地存放路径为 *data\resources*。
   - 创建测试表。

     ```
     create table wc_in (key string, value string);
     create table wc_out(key string, cnt bigint);
     ```

   - 添加测试资源。

     ```
     add jar data\resources\mapreduce-examples.jar -f;
     ```

2. 准备好 WordCount 测试表和资源。
3. 使用 tunnel 导入数据。

   ```
   tunnel upload data wc_in;
   ```

   导入 wc_in 表的数据文件 data 的内容，如下所示：

   ```
   hello,odps
   ```

测试步骤

在 odpscmd 中执行 WordCount，如下所示：

```
jar -resources mapreduce-examples.jar -classpath data\resources\
mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.WordCount wc_in wc_out
```

预期结果

作业成功结束后，输出表 wc_out 中的内容，如下所示：

```
+------------+------------+
| key        | cnt        |
+------------+------------+
| hello      | 1          |
| odps       | 1          |
+------------+------------+
```

代码示例

```
    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);
      }
    }
```

4.3.2 MapOnly示例

对于 MapOnly 的作业，Map 直接将 < Key, Value > 信息输出到 MaxCompute 的表中，您只需要指定输出表即可，不需指定 Map 输出的 Key/Value 元信息。

测试准备

1. 准备好测试程序的 Jar 包，假设名字为 mapreduce-examples.jar，本地存放路径为 data\resources。
2. 准备好 MapOnly 的测试表和资源。
   - 创建测试表。

     ```
     create table wc_in (key string, value string);
     ```

```
create table wc_out(key string, cnt bigint);
```

- 添加测试资源。

```
add jar data\resources\mapreduce-examples.jar -f;
```

3. 使用 tunnel 导入数据。

```
tunnel upload data wc_in;
```

导入 wc_in 表的数据文件 data 的内容，如下所示：

```
hello,odps
hello,odps
```

测试步骤

在 odpscmd 中执行 MapOnly，如下所示：

```
jar -resources mapreduce-examples.jar -classpath data\resources\
mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.MapOnly wc_in wc_out map
```

预期结果

作业成功结束后，输出表 wc_out 中的内容，如下所示：

```
+------------+------------+
| key        | cnt        |
+------------+------------+
| hello      | 1          |
| hello      | 1          |
+------------+------------+
```

代码示例

```
    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);
      }
```

```
    }
```

4.3.3 多路输入输出示例

测试准备

1. 准备好测试程序的 Jar 包，假设名字为 mapreduce-examples.jar，本地存放路径为 *data\resources*。
2. 准备好多路输入输出的测试表和资源。
   - 创建测试表。

     ```
     create table wc_in1(key string, value string);
     create table wc_in2(key string, value string);
     create table mr_multiinout_out1 (key string, cnt bigint);
     create table mr_multiinout_out2 (key string, cnt bigint)
     partitioned by (a string, b string);
     alter table mr_multiinout_out2 add partition (a='1', b='1');
     alter table mr_multiinout_out2 add partition (a='2', b='2');
     ```

   - 添加测试资源。

     ```
     add jar data\resources\mapreduce-examples.jar -f;
     ```

3. 使用 tunnel 导入数据。

   ```
   tunnel upload data1 wc_in1;
   tunnel upload data2 wc_in2;
   ```

   导入 wc_in1 表的数据文件 data 的内容，如下所示：

   ```
   hello,odps
   ```

   导入 wc_in2 表的数据文件 data 的内容，如下所示：

   ```
   hello,world
   ```

测试步骤

在 odpscmd 中执行 MultipleInOut，如下所示：

```
jar -resources mapreduce-examples.jar -classpath data\resources\
mapreduce-examples.jar
```

```
com.aliyun.odps.mapred.open.example.MultipleInOut wc_in1,wc_in2
mr_multiinout_out1,mr_multiinout_out2|a=1/b=1|out1,mr_multiinout_out2|
a=2/b=2|out2;
```

预期结果

作业成功结束后，mr_multiinout_out1 中的内容，如下所示：

```
+------------+------------+
| key        | cnt        |
+------------+------------+
| default    | 1          |
+------------+------------+
```

mr_multiinout_out2 中内容，如下所示：

```
+--------+------------+---+---+
| key    | cnt        | a | b |
+--------+------------+---+---+
| odps   | 1          | 1 | 1 |
| world  | 1          | 1 | 1 |
| out1   | 1          | 1 | 1 |
| hello  | 2          | 2 | 2 |
| out2   | 1          | 2 | 2 |
+--------+------------+---+---+
```

代码示例

```
    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.
     **/
    public class MultipleInOut {
      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.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, "out2");
          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);
      }
    }
```

4.3.4 多任务示例

测试准备

1. 准备好测试程序的 Jar 包，假设名字为 mapreduce-examples.jar，本地存放路径为 *data\resources*。
2. 准备好 MultiJobs 测试表和资源。
   - 创建测试表。

     ```
     create table mr_empty (key string, value string);
     create table mr_multijobs_out (value bigint);
     ```

   - 添加测试资源。

     ```
     add table mr_multijobs_out as multijobs_res_table -f;
     add jar data\resources\mapreduce-examples.jar -f;
     ```

测试步骤

在 odpscmd 中执行 MultiJobs，如下所示：

```
jar -resources mapreduce-examples.jar,multijobs_res_table -classpath
data\resources\mapreduce-examples.jar
 com.aliyun.odps.mapred.open.example.MultiJobs mr_multijobs_out;
```

预期结果

作业成功结束后，输出表 mr_multijobs_out 中的内容，如下所示：

```
+------------+
| value      |
```

```
+------------+
| 0          |
+------------+
```

代码示例

```
    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
     *
     **/
    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.");
        }
      }
    }
```

4.3.5 二次排序示例

测试准备

1. 准备好测试程序的 Jar 包，假设名字为 mapreduce-examples.jar，本地存放路径为 *data\resources*。
2. 准备好 SecondarySort 的测试表和资源。

- 创建测试表。

```
create table ss_in(key bigint, value bigint);
create table ss_out(key bigint, value bigint)
```

- 添加测试资源。

```
add jar data\resources\mapreduce-examples.jar -f;
```

3. 使用 tunnel 导入数据。

```
tunnel upload data ss_in;
```

导入 ss_in 表的数据文件 data 的内容，如下所示：

```
1,2
2,1
1,1
2,2
```

测试步骤

在 odpscmd 中执行 SecondarySort，如下所示：

```
jar -resources mapreduce-examples.jar -classpath data\resources\
mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.SecondarySort ss_in ss_out;
```

预期结果

作业成功结束后，输出表 ss_out 中的内容，如下所示：

```
+------------+------------+
| key        | value      |
+------------+------------+
| 1          | 1          |
| 1          | 2          |
| 2          | 1          |
| 2          | 2          |
+------------+------------+
```

代码示例

```
    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.
     *
     **/
    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);
      }
    }
```

4.3.6 使用资源示例

测试准备

1. 准备好测试程序的 Jar 包，假设名字为 mapreduce-examples.jar，本地存放路径为 *data\resources*。
2. 准备好测试表和资源。
   - 创建测试表。

     ```
     create table mr_upload_src(key bigint, value string);
     ```

   - 添加测试资源。

     ```
     add jar data\resources\mapreduce-examples.jar -f;
     ```

```
add file data\resources\import.txt -f;
```

- import.txt的数据内容，如下所示：

```
1000,odps
```

测试步骤

在odpscmd中执行Upload，如下所示：

```
jar -resources mapreduce-examples.jar,import.txt -classpath data\
resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Upload import.txt mr_upload_src;
```

预期结果

作业成功结束后，输出表mr_upload_src中的内容，如下所示：

```
+------------+------------+
| key        | value      |
+------------+------------+
| 1000       | odps       |
+------------+------------+
```

代码示例

```
    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
     *
     **/
    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);
      }
    }
```

实际上，您可通过以下两种方式设置JobConf。

- 通过SDK中JobConf的接口设置，本示例即是通过此方法实现。
- 在Jar命令行中，通过 -**conf**参数指定新的JobConf文件。

4.3.7 使用Counter示例

测试准备

1. 准备好测试程序的 Jar 包，假设名字为 mapreduce-examples.jar，本地存放路径为 *data\resources*。
2. 准备好 UserDefinedCounters 测试表和资源。
   - 创建测试表。

     ```
     create table wc_in (key string, value string);
     create table wc_out(key string, cnt bigint);
     ```

   - 添加测试资源。

     ```
     add jar data\resources\mapreduce-examples.jar -f;
     ```

3. 使用 tunnel 导入数据。

   ```
   tunnel upload data wc_in;
   ```

   导入 wc_in 表的数据文件 data 的内容，如下所示：

   ```
   hello,odps
   ```

测试步骤

在 odpscmd 中执行 UserDefinedCounters，如下所示：

```
jar -resources mapreduce-examples.jar -classpath data\resources\
mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.UserDefinedCounters wc_in wc_out
```

预期结果

作业成功结束后，可以看到 Counters 的输出，如下所示：

```
Counters: 3
com.aliyun.odps.mapred.open.example.UserDefinedCounters$MyCounter
MAP_TASKS=1
REDUCE_TASKS=1
TOTAL_TASKS=2
```

输出表 wc_out 中的内容，如下所示：

```
+------------+------------+
| key        | cnt        |
+------------+------------+
| hello      | 1          |
| odps       | 1          |
```

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

代码示例

```
    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
     *
     **/
    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);
      }
    }
```

4.3.8 Grep示例

测试准备

1. 准备好测试程序的 Jar 包，假设名字为 mapreduce-examples.jar，本地存放路径为 *data\resources*。
2. 准备好 Grep 测试表和资源。

- 创建测试表。

```
create table mr_src(key string, value string);
create table mr_grep_tmp (key string, cnt bigint);
create table mr_grep_out (key bigint, value string);
```

- 添加测试资源。

```
add jar data\resources\mapreduce-examples.jar -f;
```

3. 使用 tunnel 导入数据。

```
tunnel upload data mr_src;
```

导入 mr_src 表的数据文件 data 的内容，如下所示：

```
hello,odps
hello,world
```

测试步骤

在 odpscmd 中执行 Grep，如下所示：

```
jar -resources mapreduce-examples.jar -classpath data\resources\
mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Grep mr_src mr_grep_tmp mr_grep_ou
t hello;
```

预期结果

作业成功结束后，输出表 mr_grep_out 中的内容，如下所示：

```
+------------+------------+
| key        | value      |
+------------+------------+
| 2          | hello      |
+------------+------------+
```

代码示例

```
    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.
     *
     **/
    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" });
        @SuppressWarnings("unused")
        RunningJob rjSort = JobClient.runJob(sortJob);
      }
    }
```

4.3.9 Join示例

MaxCompute MapReduce 框架自身并不支持 Join 逻辑，但您可以在自己的 Map/Reduce 函数中实现数据的 Join，当然这需要您做一些额外的工作。

假设需要 Join 两张表 mr_Join_src1(key bigint, value string) 和 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。

测试准备

1. 准备好测试程序的 Jar 包，假设名字为 mapreduce-examples.jar，本地存放路径为 *data\resources*。
2. 准备好 Join 的测试表和资源。
   - 创建测试表。

     ```
     create table mr_Join_src1(key bigint, value string);
     create table mr_Join_src2(key bigint, value string);
     create table mr_Join_out(key bigint, value1 string, value2 string
     );
     ```

   - 添加测试资源。

     ```
     add jar data\resources\mapreduce-examples.jar -f;
     ```

3. 使用 tunnel 导入数据。

   ```
   tunnel upload data1 mr_Join_src1;
   ```

```
tunnel upload data2 mr_Join_src2;
```

导入 mr_Join_src1 数据的内容，如下所示：

```
 1,hello
 2,odps
```

导入 mr_Join_src2 数据的内容，如下所示：

```
1,odps
3,hello
4,odps
```

测试步骤

在 odpscmd 中执行 Join，如下所示：

```
jar -resources mapreduce-examples.jar -classpath data\resources\
mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Join mr_Join_src1 mr_Join_src2
mr_Join_out;
```

预期结果

作业成功结束后，输出表 mr_Join_out 中的内容，如下所示：

```
+------------+------------+------------+
| key        | value1     | value2     |
+------------+------------+------------+
|  1         | hello      |  odps      |
+------------+------------+------------+
```

代码示例

```
    package com.aliyun.odps.mapred.open.example;
    import java.io.IOException;
    import java.util.ArrayList;
    import java.util.Iterator;
    import java.util.List;
    import org.apache.commons.logging.Log;
    import org.apache.commons.logging.LogFactory;
    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.conf.JobConf;
    import com.aliyun.odps.mapred.utils.InputUtils;
    import com.aliyun.odps.mapred.utils.OutputUtils;
    import com.aliyun.odps.mapred.utils.SchemaUtils;
    /**
     * Join, mr_Join_src1/mr_Join_src2(key bigint, value string),
mr_Join_out(key
     * bigint, value1 string, value2 string)
     *
     */
```

```
    public class Join {
      public static final Log LOG = LogFactory.getLog(Join.class);
      public static class JoinMapper extends MapperBase {
        private Record mapkey;
        private Record mapvalue;
        private long tag;
        @Override
        public void setup(TaskContext context) throws IOException {
          mapkey = context.createMapOutputKeyRecord();
          mapvalue = context.createMapOutputValueRecord();
          tag = context.getInputTableInfo().getLabel().equals("left
") ? 0 : 1;
        }
        @Override
        public void map(long key, Record record, TaskContext context)
            throws IOException {
          mapkey.set(0, record.get(0));
          mapkey.set(1, tag);
          for (int i = 1; i < record.getColumnCount(); i++) {
            mapvalue.set(i - 1, 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 = key.getBigint(0);
          List<Object[]> leftValues = new ArrayList<Object[]>();
          while (values.hasNext()) {
            Record value = values.next();
            long tag = (Long) key.get(1);
            if (tag == 0) {
              leftValues.add(value.toArray().clone());
            } else {
              for (Object[] leftValue : leftValues) {
                int index = 0;
                result.set(index++, k);
                for (int i = 0; i < leftValue.length; i++) {
                  result.set(index++, leftValue[i]);
                }
                for (int i = 0; i < value.getColumnCount(); i++) {
                  result.set(index++, value.get(i));
                }
                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("value:
string"));
        job.setPartitionColumns(new String[]{"key"});
        job.setOutputKeySortColumns(new String[]{"key", "tag"});
        job.setOutputGroupingColumns(new String[]{"key"});
        job.setNumReduceTasks(1);
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).
label("left").build(), job);
        InputUtils.addTable(TableInfo.builder().tableName(args[1]).
label("right").build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[2]).
build(), job);
        JobClient.runJob(job);
      }
    }
```

4.3.10 Sleep示例

测试准备

1. 准备好测试程序的 Jar 包，假设名字为 mapreduce-examples.jar，本地存放路径为 *data\resources*。
2. 准备好 SleepJob 的测试资源。

```
add jar data\resources\mapreduce-examples.jar -f;
```

测试步骤

在 odpscmd 中执行 Sleep，如下所示：

```
  jar -resources mapreduce-examples.jar -classpath data\resources\
mapreduce-examples.jar
  com.aliyun.odps.mapred.open.example.Sleep 10;
  jar -resources mapreduce-examples.jar -classpath data\resources\
mapreduce-examples.jar
  com.aliyun.odps.mapred.open.example.Sleep 100;
```

预期结果

作业成功结束后，对比不同 Sleep 时长的运行时间，可以看到效果。

代码示例

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.conf.JobConf;
```

```
public class Sleep {
  private static final String SLEEP_SECS = "sleep.secs";
  public static class MapperClass extends MapperBase {
    @Override
    public void setup(TaskContext context) throws IOException {
      try {
        Thread.sleep(context.getJobConf().getInt(SLEEP_SECS, 1) * 1000
);
      } catch (InterruptedException e) {
        throw new RuntimeException(e);
      }
    }
  }
  public static void main(String[] args) throws Exception {
    if (args.length != 1) {
      System.err.println("Usage: Sleep <sleep_secs>");
      System.exit(-1);
    }
    JobConf job = new JobConf();
    job.setMapperClass(MapperClass.class);
    job.setNumReduceTasks(0);
    job.setNumMapTasks(1);
    job.set(SLEEP_SECS, args[0]);
    JobClient.runJob(job);
  }
}
```

4.3.11 Unique示例

测试准备

1. 准备好测试程序的 Jar 包，假设名字为 mapreduce-examples.jar，本地存放路径为 *data\resources*。
2. 准备好 Unique 的测试表和资源。
   - 创建测试表。

     ```
     create table ss_in(key bigint, value bigint);
     create table ss_out(key bigint, value bigint);
     ```

   - 添加测试资源。

     ```
     add jar data\resources\mapreduce-examples.jar -f;
     ```

3. 使用 tunnel 导入数据。

   ```
   tunnel upload data ss_in;
   ```

   导入 ss_in 表的数据文件 data 的内容，如下所示：

   ```
   1,1
   1,1
   2,2
   ```

```
2,2
```

测试步骤

在 odpscmd 中执行 Unique，如下所示：

```
jar -resources mapreduce-examples.jar -classpath data\resources\
mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Unique ss_in ss_out key;
```

预期结果

作业成功结束后，输出表 ss_out 中的内容，如下所示：

```
+------------+------------+
| key        | value      |
+------------+------------+
| 1          |   1        |
| 2          |   2        |
+------------+------------+
```

代码示例

```
    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
     *
     **/
    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);
        }
      }
    }
```

4.3.12 Sort示例

测试准备

1. 准备好测试程序的 Jar 包，假设名字为 mapreduce-examples.jar，本地存放路径为 *data\resources*。
2. 准备好 Sort 的测试表和资源。
   - 创建测试表。

     ```
     create table ss_in(key bigint, value bigint);
     ```

```
create table ss_out(key bigint, value bigint);
```

- 添加测试资源。

```
add jar data\resources\mapreduce-examples.jar -f;
```

3. 使用 tunnel 导入数据。

```
tunnel upload data ss_in;
```

导入 ss_in 表的数据文件 data 的内容，如下所示：

```
 2,1
 1,1
 3,1
```

测试步骤

在 odpscmd 中执行 Sort，如下所示：

```
jar -resources mapreduce-examples.jar -classpath data\resources\
mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Sort ss_in ss_out;
```

预期结果

作业成功结束后，输出表 ss_out 中的内容，如下所示：

```
+------------+------------+
| key        | value      |
+------------+------------+
| 1          | 1          |
| 2          | 1          |
| 3          | 1          |
+------------+------------+
```

代码示例

```
    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.
     *
```

```
     **/
    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.");
      }
```

```
    }
```

4.3.13 分区表输入示例

本文将为您介绍两个把 Partition 作为输入输出的示例，仅供参考。

示例一：

```
     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);
    }
```

示例二：

```
    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。

4.3.14 Pipeline示例

测试准备

1. 准备好测试程序的 Jar 包，假设名字为 mapreduce-examples.jar，本地存放路径为 *data\resources*。
2. 准备好 Pipeline 的测试表和资源。
   - 创建测试表。

     ```
     create table wc_in (key string, value string);
     create table wc_out(key string, cnt bigint);
     ```

   - 添加测试资源。

     ```
     add jar data\resources\mapreduce-examples.jar -f;
     ```

3. 使用 tunnel 导入数据。

   ```
   tunnel upload data wc_in;
   ```

   导入 wc_in 表的数据文件 data 的内容，如下所示：

   ```
   hello,odps
   ```

测试步骤

在 odpscmd 中执行 WordCountPipeline，如下所示：

```
jar -resources mapreduce-examples.jar -classpath data\resources\
mapreduce-examples.jar
```

```
com.aliyun.odps.mapred.open.example.WordCountPipeline wc_in wc_out;
```

预期结果

作业成功结束后，输出表 wc_out 中的内容，如下所示：

```
+------------+------------+
| key        | cnt        |
+------------+------------+
| hello      | 1          |
| odps       | 1          |
+------------+------------+
```

代码示例

```
    package com.aliyun.odps.mapred.open.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 WordCountPipelineTest {
      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 value;
        @Override
        public void setup(TaskContext context) throws IOException {
          value = 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);
          }
          value.set(0, count);
          context.write(key, value);
        }
      }
      public static class IdentityReducer extends ReducerBase {
        private Record result;
        @Override
        public void setup(TaskContext context) throws IOException {
          result = context.createOutputRecord();
        }
        @Override
        public void reduce(Record key, Iterator<Record> values,
TaskContext context)
            throws IOException {
          while (values.hasNext()) {
            result.set(0, key.get(0));
            result.set(1, values.next().get(0));
            context.write(result);
          }
        }
      }
      public static void main(String[] args) throws OdpsException {
        if (args.length != 2) {
          System.err.println("Usage: WordCountPipeline <in_table> <
out_table>");
          System.exit(2);
        }
        Job job = new Job();
        /***
         * 构造Pipeline的过程中，如果不指定Mapper的OutputKeySortColumns，
PartitionColumns，OutputGroupingColumns，
         * 框架会默认使用其OutputKey作为此三者的默认配置
         ***/
        Pipeline pipeline = Pipeline.builder()
            .addMapper(TokenizerMapper.class)
            .setOutputKeySchema(
                    new Column[] { new Column("word", OdpsType.STRING
) })
            .setOutputValueSchema(
                    new Column[] { new Column("count", OdpsType.BIGINT
) })
            .setOutputKeySortColumns(new String[] { "word" })
            .setPartitionColumns(new String[] { "word" })
            .setOutputGroupingColumns(new String[] { "word" })
            .addReducer(SumReducer.class)
            .setOutputKeySchema(
                    new Column[] { new Column("word", OdpsType.STRING
) })
            .setOutputValueSchema(
                    new Column[] { new Column("count", OdpsType.BIGINT
)})
            .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);
    }
}
```

4.4 Java SDK

4.4.1 原生SDK概述

本文将会为您介绍较为常用的 MapReduce 核心接口。

如果您使用 Maven，可以从 *Maven* 库 中搜索“odps-sdk-mapred”获取不同版本的 Java SDK，相关配置信息如下：

```
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-sdk-mapred</artifactId>
    <version>0.26.2-public</version>
</dependency>
```

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。

MapperBase

主要函数接口：

主要接口	描述
void cleanup(TaskContext context)	在Map阶段结束时，map方法之后调用。

主要接口	描述
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。

ReducerBase

主要函数接口：

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。

TaskContext

主要函数接口：

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。
void write(Record key, Record value)	Map写记录到中间结果，可以在Map函数中多次调用。 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

主要接口	描述
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：

MaxCompute 的 TaskContext 接口中提供了 progress 功能，但此功能是在防止 Worker 长时间运行未结束，被框架误认为超时而被杀的情况出现。此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，建议您在 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。

JobConf

主要函数接口：

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
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

主要接口	描述
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256
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。

说明：

- 通常情况下，GroupingColumns 包含在 KeySortColumns 中，KeySortColumns 和 PartitionColumns 要包含在 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。

JobClient

主要函数接口：

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

RunningJob

主要函数接口：

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 waitForCompletion()	等待直至作业实例结束。一般使用于异步方式提交的作业。
JobStatus getJobStatus()	查询作业实例运行状态。
void killJob()	结束此作业。
Counters getCounters()	获取Conter信息。

InputUtils

主要函数接口：

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。

OutputUtils

主要函数接口：

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。

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，您可以在 main 函数中构建一个 Map 后连续接两个 Reduce 的 MapReduce 任务。如果您比较熟悉 MapReduce 的基础功能，即可轻松使用 MR2。

说明：

- 建议您在使用 MR2 功能前，首先了解 MapReduce 的基础用法。
- JobConf 仅能够配置 Map 后接单 Reduce 的 MapReduce 任务。

数据类型

MapReduce 支持的数据类型有：Bigint，String，Double，Boolean，datetime 和 Decimal 类型。MaxCompute 数据类型与 Java 数据类型的对应关系，如下所示：

MaxCompute SQL Type	Bigint	String	Double	Boolean	Datetime	Decimal
Java Type	Long	String	Double	Boolean	Date	BigDecimal

4.5 MR限制项汇总

为避免您因没注意限制条件，业务启动后才发现限制条件，导致业务停止的现象发生，本文将对 MaxCompute MR 限制项进行汇总，以方便您查看。

MaxCompute MR 限制项汇总，如下表所示：

边界名	边界值	分类	配置项名称	默认值	是否可配置	说明
instance 内存占用	[256MB，12GB]	内存限制	odps.stage.mapper(reducer).mem 和 odps.stage.mapper(reducer).jvm.mem	2048M + 1024M	是	单个 map instance 或 reduce instance 占用 memory，由框架 memory（默认2048M）和 jvm 的heap memory（默认1024M）两部分 。
resource 数量	256个	数量限制	不涉及	无	否	单个 job 引用的 resource 数量不超过 256 个，table、archive 按照一个单位计算 。
输入路数和输出路数	1024 个和 256 个	数量限制	不涉及	无	否	单个job 的输入路数不能超过 1024（同一个表的一个分区算一路输入，总的不同表个数不能超过 64 个），单个 job 的输出路数不能超过 。
counter 数量	64个	数量限制	不涉及	无	否	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 。
map instance	[1，100000]	数量限制	odps.stage.mapper.num	无	是	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.

边界名	边界值	分类	配置项名称	默认值	是否可配置	说明
						stage.mapper.num 直接设置，最终个数范围 [1, 100000].
reduce instance	[0，2000]	数量限制	odps.stage.reducer.num	无	是	单个 job 默认 reduce instance 个数为 map instance 个数的1/4，用户设置作为最终的 reduce instance 个数，范围 [0, 2000]。可能出现这样的情形：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 。
重试次数	3	数量限制	不涉及	无	否	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。
local debug 模式	instance 个数不超100	数量限制	不涉及	无	否	local debug 模式下，默认 map instance 个数为 2，不能超过 100；默认 reduce instance 个数为 1，不能超过 100；默认一路输入下载记录数 100，不能超过 10000 。
重复读取 resource次数	64次	数量限制	不涉及	无	否	单个 map instance 或 reduce instance 重复读一个 resource 次数限制<=64 次 。
resource 字节数	2G	长度限制	不涉及	无	否	单个 job 引用的resource 总计字节数大小不超过 2G 。

边界名	边界值	分类	配置项名称	默认值	是否可配置	说明
split size	[1，)	长度限制	odps.stage.mapper.split.size	256M	是	框架会参考设置的 split size 值来划分 map，决定 map 的个数 。
string 列内容长度	8M	长度限制	不涉及	无	否	MaxCompute 表string 列内容长度不允许 。
worker 运行超时时间	[1，3600]	时间限制	odps.function.timeout	600	是	map 或者 reduce worker 在无数据读写且没有通过context.progress()主动发送心态的情况下的超时时间，默认值是 600s。
MR引用Table资源支持的字段类型	BIGINT、DOUBLE、STRING、DATETIME、BOOLEAN	数据类型限制	不涉及	无	否	MR任务引用到Table资源时，若table表字段有其他类型字段执行报错

5 Java沙箱

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，不允许启动子进程执行 Linux 命令。
- 不允许访问网络，包括获取本地 IP 地址等，都会被禁止。
- Java 反射限制：suppressAccessChecks 权限被禁止，无法 setAccessible 某个 private 的属性或方法，以达到读取 private 属性或调用 private 方法的目的。

具体的说，在代码中，直接使用下面这些方法会报 access denied 异常。

访问本地文件的相关方法，如下所示：

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，如下所示：

```
    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, DomainComb
iner combiner)
    public DomainCombiner getDomainCombiner()
```

6 SDK

6.1 Java SDK

本文将为您介绍较为常用的 MaxCompute 核心接口，更多详情请参见 *SDK Java Doc*。

您可以通过 Maven 管理配置新 SDK 的版本。Maven 的配置信息如下（最新版本可以随时到 *search.maven.org* 搜索 odps-sdk-core 获取）：

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-core</artifactId>
  <version>0.26.2-public</version>
</dependency>
```

MaxCompute 提供的 SDK 包整体信息，如下表所示：

包名	描述
odps-sdk-core	MaxCompute 的基础功能，例如：对表，Project 的操作，以及 Tunnel 均在此包中
odps-sdk-commons	一些 Util 封装
odps-sdk-udf	UDF 功能的主体接口
odps-sdk-mapred	MapReduce 功能
odps-sdk-graph	Graph Java SDK，搜索关键词“odps-sdk-graph”

AliyunAccount

阿里云认证账号。输入参数为 accessId 及 accessKey，是阿里云用户的身份标识和认证密钥。此类用来初始化 MaxCompute。

MaxCompute

MaxCompute SDK 的入口，您可通过此类来获取项目空间下的所有对象集合，包括：*Projects*，*Tables*，*Resources*，*Functions*，*Instances*。

说明：

MaxCompute 原名 ODPS，因此在现有的 SDK 版本中，入口类仍命名为 ODPS。

您可以通过传入 AliyunAccount 实例来构造 MaxCompute 对象。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    odps.setDefaultProject("my_project");
    for (Table t : odps.tables()) {
        ....
    }
```

Projects

Projects 是 MaxCompute 中所有项目空间的集合。集合中的元素为 Project。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    Project p = odps.projects().get("my_exists");
    p.reload();
    Map<String, String> properties = prj.getProperties();
    ...
```

Project

Project 是对项目空间信息的描述，可以通过 Projects 获取相应的项目空间。

SQLTask

SQLTask 是用于运行、处理 SQL 任务的接口。可以通过 run 接口直接运行 SQL。(注意：每次只能提交运行一个**SQL**语句。)

run 接口返回 Instance 实例，通过 Instance 获取 SQL 的运行状态及运行结果。程序示例如下：

```
    import java.util.List;
    import com.aliyun.odps.Instance;
    import com.aliyun.odps.Odps;
    import com.aliyun.odps.OdpsException;
    import com.aliyun.odps.account.Account;
    import com.aliyun.odps.account.AliyunAccount;
    import com.aliyun.odps.data.Record;
    import com.aliyun.odps.task.SQLTask;
    public class testSql {
    private static final String accessId = "";
    private static final String accessKey = "";
    private static final String endPoint = "http://service.odps.aliyun
.com/api";
    private static final String project = "";
    private static final String sql = "select category from iris;";
    public static void
    main(String[] args) {
      Account account = new AliyunAccount(accessId, accessKey);
       Odps odps = new Odps(account);
```

```
        odps.setEndpoint(endPoint);
        odps.setDefaultProject(project);
        Instance i;
       try {
          i = SQLTask.run(odps, sql);
          i.waitForSuccess();
          List<Record> records = SQLTask.getResult(i);
          for(Record r:records){
             System.out.println(r.get(0).toString());
          }
       } catch (OdpsException e) {
          e.printStackTrace();
       }
    }
  }
```

说明：

如果您想创建表，需要通过 SQLTask 接口，而不是 Table 接口。您需要将 创建表的语句传入 SQLTask。

Instances

Instances 是 MaxCompute 中所有实例（Instance）的集合，集合中的元素为 Instance。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    odps.setDefaultProject("my_project");
    for (Instance i : odps.instances()) {
        ....
    }
```

Instance

Instance 是对实例信息的描述，可以通过 Instances 获取相应的实例。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    Instance ins = odps.instances().get("instance id");
    Date startTime = instance.getStartTime();
    Date endTime = instance.getEndTime();
    ...
    Status instanceStatus = instance.getStatus();
    String instanceStatusStr = null;
    if (instanceStatus == Status.TERMINATED) {
      instanceStatusStr = TaskStatus.Status.SUCCESS.toString();
      Map<String, TaskStatus> taskStatus = instance.getTaskStatus();
      for (Entry<String, TaskStatus> status : taskStatus.entrySet()) {
```

```
        if (status.getValue().getStatus() != TaskStatus.Status.SUCCESS
) {
          instanceStatusStr = status.getValue().getStatus().toString
();
          break;
        }
      }
    } else {
      instanceStatusStr = instanceStatus.toString();
    }
    ...
    TaskSummary summary = instance.getTaskSummary("instance name");
    String s = summary.getSummaryText();
```

Tables

Tables 是 MaxCompute 中所有表的集合，集合中的元素为 Table。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    odps.setDefaultProject("my_project");
    for (Table t : odps.tables()) {
        ....
    }
```

Table

Table 是对表信息的描述，可以通过 Tables 获取相应的表。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    Table t = odps.tables().get("table name");
    t.reload();
    Partition part = t.getPartition(new PartitionSpec(tableSpec[1]));
    part.reload();
    ...
```

Resources

Resources 是 MaxCompute 中所有资源的集合。集合中的元素为 Resource。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    odps.setDefaultProject("my_project");
    for (Resource r : odps.resources()) {
        ....
```

```
    }
```

Resource

Resource 是对资源信息的描述，可以通过 Resources 获取相应的资源。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    Resource r = odps.resources().get("resource name");
    r.reload();
    if (r.getType() == Resource.Type.TABLE) {
     TableResource tr = new TableResource(r);
     String tableSource = tr.getSourceTable().getProject() + "."
         + tr.getSourceTable().getName();
     if (tr.getSourceTablePartition() != null) {
       tableSource += " partition(" + tr.getSourceTablePartition().
toString()
           + ")";
     }
     ....
    }
```

创建文件资源的示例，如下所示：

```
    String projectName = "my_porject";
    String source = "my_local_file.txt";
    File file = new File(source);
    InputStream is = new FileInputStream(file);
    FileResource resource = new FileResource();
    String name = file.getName();
    resource.setName(name);
    odps.resources().create(projectName, resource, is);
```

创建表资源的示例，如下所示：

```
    TableResource resource = new TableResource(tableName, tablePrj,
partitionSpec);
    //resource.setName(INVALID_USER_TABLE);
    resource.setName("table_resource_name");
    odps.resources().update(projectName, resource);
```

Functions

Functions 是 MaxCompute 中所有函数的集合。集合中的元素为 Function。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    odps.setDefaultProject("my_project");
    for (Function f : odps.functions()) {
        ....
```

```
    }
```

Function

Function 是对函数信息的描述，可以通过 Functions 获取相应的函数。程序示例如下：

```
    Account account = new AliyunAccount("my_access_id", "my_access_key
");
    Odps odps = new Odps(account);
    String odpsUrl = "<your odps endpoint>";
    odps.setEndpoint(odpsUrl);
    Function f = odps.functions().get("function name");
    List<Resource> resources = f.getResources();
```

创建函数的示例，如下所示：

```
    String resources = "xxx:xxx";
    String classType = "com.aliyun.odps.mapred.open.example.WordCount
";
    ArrayList<String> resourceList = new ArrayList<String>();
    for (String r : resources.split(":")) {
      resourceList.add(r);
    }
    Function func = new Function();
    func.setName(name);
    func.setClassType(classType);
    func.setResources(resourceList);
    odps.functions().create(projectName, func);
```

6.2 Python SDK

PyODPS是MaxCompute的Python版本的SDK，它提供了对MaxCompute对象的基本操作和DataFrame框架，可以轻松地在MaxCompute上进行数据分析。更多详情请参见*Github*项目和包括所有接口、类的细节等内容的详细文档。

- 关于PyODPS的更多详情请参见*PyODPS*云栖社区专辑。
- 欢迎各位开发者参与到PyODPS的生态开发中，详情请参见*GitHub*文档。
- 欢迎提交issue和merge request，加快PyODPS生态成长，更多详情请参见代码。
- 钉钉技术交流群：**11701793**

安装

PyODPS支持Python2.6以上（包括Python3），系统安装pip后，只需运行`pip install pyodps`，PyODPS的相关依赖便会自动安装。

快速开始

首先，用阿里云账号初始化一个MaxCompute的入口，如下所示：

```
from odps import ODPS
odps = ODPS('**your-access-id**', '**your-secret-access-key**', '**
your-default-project**',
            endpoint='**your-end-point**')
```

根据上述操作初始化后，便可对表、资源、函数等进行操作。

项目空间

项目空间是MaxCompute的基本组织单元，类似于Database的概念。

您可通过`get_project`获取到某个项目空间，如下所示：

```
project = odps.get_project('my_project')  # 取到某个项目
project = odps.get_project()              # 取到默认项目
```

说明：

- 如果不提供参数，则获取到默认项目空间。
- 通过`exist_project`，可以查看某个项目空间是否存在。
- 表是MaxCompute的数据存储单元。

表操作

通过调用`list_tables`可以列出项目空间下的所有表，如下所示：

```
for table in odps.list_tables():
    # 处理每张表
```

通过调用`exist_table`可以判断表是否存在，通过调用`get_table`可以获取表。

```
>>> t = odps.get_table('dual')
>>> t.schema
odps.Schema {
  c_int_a                 bigint
  c_int_b                 bigint
  c_double_a              double
  c_double_b              double
  c_string_a              string
  c_string_b              string
  c_bool_a                boolean
  c_bool_b                boolean
  c_datetime_a            datetime
  c_datetime_b            datetime
}
>>> t.lifecycle
-1
```

```
>>> print(t.creation_time)
2014-05-15 14:58:43
>>> t.is_virtual_view
False
>>> t.size
1408
>>> t.schema.columns
[<column c_int_a, type bigint>,
 <column c_int_b, type bigint>,
 <column c_double_a, type double>,
 <column c_double_b, type double>,
 <column c_string_a, type string>,
 <column c_string_b, type string>,
 <column c_bool_a, type boolean>,
 <column c_bool_b, type boolean>,
 <column c_datetime_a, type datetime>,
 <column c_datetime_b, type datetime>]
```

创建表的Schema

初始化的方法有两种，如下所示：

- 通过表的列和可选的分区来初始化。

```
>>> from odps.models import Schema, Column, Partition
>>> columns = [Column(name='num', type='bigint', comment='the column
')]
>>> partitions = [Partition(name='pt', type='string', comment='the
partition')]
>>> schema = Schema(columns=columns, partitions=partitions)
>>> schema.columns
[<column num, type bigint>, <partition pt, type string>]
```

- 通过调用Schema.from_lists，虽然调用更加方便，但显然无法直接设置列和分区的注释。

```
>>> schema = Schema.from_lists(['num'], ['bigint'], ['pt'], ['string
'])
>>> schema.columns
[<column num, type bigint>, <partition pt, type string>]
```

创建表

您可以使用表的Schema来创建表，操作如下所示：

```
>>> table = odps.create_table('my_new_table', schema)
>>> table = odps.create_table('my_new_table', schema, if_not_exists=
True)  # 只有不存在表时才创建
>>> table = o.create_table('my_new_table', schema, lifecycle=7)  # 设置
生命周期
```

也可以使用逗号连接的字段名 字段类型字符串组合来创建表，操作如下所示：

```
>>> # 创建非分区表
>>> table = o.create_table('my_new_table', 'num bigint, num2 double',
if_not_exists=True)
>>> # 创建分区表可传入 (表字段列表, 分区字段列表)
```

```
>>> table = o.create_table('my_new_table', ('num bigint, num2 double',
 'pt string'), if_not_exists=True)
```

在未经设置的情况下，创建表时，只允许使用bigint、double、decimal、string、datetime、boolean、map和array类型。

如果您的服务位于公共云，或者支持tinyint、struct等新类型，可以设置`options.sql.use_odps2_extension = True`，以打开这些类型的支持，示例如下：

```
>>> from odps import options
>>> options.sql.use_odps2_extension = True
>>> table = o.create_table('my_new_table', 'cat smallint, content
struct<title:varchar(100), body string>')
```

获取表数据

您可通过以下三种方法获取表数据。

- 通过调用`head`获取表数据，但仅限于查看每张表开始的小于1万条的数据，如下所示：

```
>>> t = odps.get_table('dual')
>>> for record in t.head(3):
>>>     print(record[0])  # 取第0个位置的值
>>>     print(record['c_double_a'])  # 通过字段取值
>>>     print(record[0: 3])  # 切片操作
>>>     print(record[0, 2, 3])  # 取多个位置的值
>>>     print(record['c_int_a', 'c_double_a'])  # 通过多个字段取值
```

- 通过在table上执行`open_reader`操作，打开一个reader来读取数据。您可以使用with表达式，也可以不使用。

```
# 使用with表达式
>>> with t.open_reader(partition='pt=test') as reader:
>>>     count = reader.count
>>>     for record in reader[5:10]  # 可以执行多次，直到将count数量的
record读完，这里可以改造成并行操作
>>>         # 处理一条记录
>>>
>>> # 不使用with表达式
>>> reader = t.open_reader(partition='pt=test')
>>> count = reader.count
>>> for record in reader[5:10]
>>>     # 处理一条记录
```

- 通过使用Tunnel API读取表数据，`open_reader`操作其实也是对Tunnel API的封装。

写入数据

类似于open_reader，table对象同样可以执行open_writer来打开writer，并写数据。如下所示：

```
>>> # 使用 with 表达式
>>> with t.open_writer(partition='pt=test') as writer:
>>>     writer.write(records)  # 这里records可以是任意可迭代的records，默认写到block 0
>>>
>>> with t.open_writer(partition='pt=test', blocks=[0, 1]) as writer:
 # 这里同是打开两个block
>>>     writer.write(0, gen_records(block=0))
>>>     writer.write(1, gen_records(block=1))  # 这里两个写操作可以多线程并行，各个block间是独立的
>>>
>>> # 不使用 with 表达式
>>> writer = t.open_writer(partition='pt=test', blocks=[0, 1])
>>> writer.write(0, gen_records(block=0))
>>> writer.write(1, gen_records(block=1))
>>> writer.close()  # 不要忘记关闭 writer，否则数据可能写入不完全
```

同样，向表中写入数据也是对Tunnel API的封装，更多详情请参见数据上传下载通道。

删除表

删除表的操作，如下所示：

```
>>> odps.delete_table('my_table_name', if_exists=True)  #  只有表存在时删除
>>> t.drop()  # Table对象存在的时候可以直接执行drop函数
```

表分区

- 基本操作

  遍历表的全部分区，如下所示：

  ```
  >>> for partition in table.partitions:
  >>>     print(partition.name)
  >>> for partition in table.iterate_partitions(spec='pt=test'):
  >>>     # 遍历二级分区
  ```

  判断分区是否存在，如下所示：

  ```
  >>> table.exist_partition('pt=test,sub=2015')
  ```

  获取分区，如下所示：

  ```
  >>> partition = table.get_partition('pt=test')
  >>> print(partition.creation_time)
  ```

```
2015-11-18 22:22:27
>>> partition.size
0
```

- 创建分区

```
>>> t.create_partition('pt=test', if_not_exists=True) # 不存在的时候才创建
```

- 删除分区

```
>>> t.delete_partition('pt=test', if_exists=True)  # 存在的时候才删除
>>> partition.drop()  # Partition对象存在的时候直接drop
```

SQL

PyODPS支持MaxCompute SQL的查询，并可以读取执行的结果。

- 执行**SQL**

```
>>> odps.execute_sql('select * from dual')  #  同步的方式执行，会阻塞直到SQL执行完成
>>> instance = odps.run_sql('select * from dual')  # 异步的方式执行
>>> instance.wait_for_success()  # 阻塞直到完成
```

- 读取**SQL**执行结果

  运行SQL的instance能够直接执行`open_reader`的操作，一种情况是SQL返回了结构化的数据。

```
>>> with odps.execute_sql('select * from dual').open_reader() as reader:
>>>     for record in reader:
>>>         # 处理每一个record
```

另一种情况是SQL可能执行的比如`desc`，这时通过`reader.raw`属性取到原始的SQL执行结果。

```
>>> with odps.execute_sql('desc dual').open_reader() as reader:
>>>     print(reader.raw)
```

Resource

资源在MaxCompute上常用在UDF和MapReduce中。

列出所有资源还是可以使用`list_resources`，判断资源是否存在使用`exist_resource`。删除资源时，可以调用`delete_resource`，或者直接对于Resource对象调用`drop`方法。

在PyODPS中，主要支持两种资源类型，一种是文件，另一种是表。

- 文件资源

文件资源包括基础的file类型、以及py、jar和archive。

说明：

在DataWorks中，py格式的文件资源请以file形式上传，详情请参见*Python UDF*文档。

创建文件资源

创建文件资源可以通过给定资源名、文件类型、以及一个file-like的对象（或者是字符串对象）来创建，示例如下：

```
resource = odps.create_resource('test_file_resource', 'file',
file_obj=open('/to/path/file'))  # 使用file-like的对象
resource = odps.create_resource('test_py_resource', 'py', file_obj='
import this')  # 使用字符串
```

读取和修改文件资源

对文件资源调用open方法，或者在MaxCompute入口调用open_resource都能打开一个资源，打开后的对象会是file-like的对象。类似于Python内置的open方法，文件资源也支持打开的模式。示例如下：

```
>>> with resource.open('r') as fp:  # 以读模式打开
>>>     content = fp.read()  # 读取全部的内容
>>>     fp.seek(0)  # 回到资源开头
>>>     lines = fp.readlines()  # 读成多行
>>>     fp.write('Hello World')  # 报错，读模式下无法写资源
>>>
>>> with odps.open_resource('test_file_resource', mode='r+') as fp:
 # 读写模式打开
>>>     fp.read()
>>>     fp.tell()  # 当前位置
>>>     fp.seek(10)
>>>     fp.truncate()  # 截断后面的内容
>>>     fp.writelines(['Hello\n', 'World\n'])  # 写入多行
>>>     fp.write('Hello World')
>>>     fp.flush()  # 手动调用会将更新提交到ODPS
```

所有支持的打开类型包括：

- r：读模式，只能打开不能写。
- w：写模式，只能写入而不能读文件，注意用写模式打开，文件内容会被先清空。
- a：追加模式，只能写入内容到文件末尾。
- r+：读写模式，能任意读写内容。
- w+：类似于r+，但会先清空文件内容。
- a+：类似于r+，但写入时只能写入文件末尾。

同时，PyODPS中，文件资源支持以二进制模式打开，打开如说一些压缩文件等等就需要以这种模式， 因此`rb`就是指以二进制读模式打开文件，`r+b`是指以二进制读写模式打开。

- 表资源

  创建表资源

```
>>> odps.create_resource('test_table_resource', 'table', table_name
='my_table', partition='pt=test')
```

  更新表资源

```
>>> table_resource = odps.get_resource('test_table_resource')
>>> table_resource.update(partition='pt=test2', project_name='
my_project2')
```

DataFrame

PyODPS提供了DataFrame API，它提供了类似pandas的接口，但是能充分利用MaxCompute的计算能力。完整的DataFrame文档请参见*DataFrame*。

DataFrame的示例如下：

说明：

在所有步骤开始前，需要创建MaxCompute对象。

```
>>> o = ODPS('**your-access-id**', '**your-secret-access-key**',
             project='**your-project**', endpoint='**your-end-point
**'))
```

此处以movielens 100K作为示例，假设已经有三张表，分别是`pyodps_ml_100k_movies`（电影相关的数据），`pyodps_ml_100k_users`（用户相关的数据），`pyodps_ml_100k_ratings`（评分有关的数据）。

只需传入Table对象，便可创建一个DataFrame对象。如下所示：

```
>>> from odps.df import DataFrame
```

```
>>> users = DataFrame(o.get_table('pyodps_ml_100k_users'))
```

通过dtypes属性来查看这个DataFrame有哪些字段，分别是什么类型，如下所示：

```
>>> users.dtypes
```

通过head方法，可以获取前N条数据，方便快速预览数据。如下所示：

```
>>> users.head(10)
```

	user_id	age	sex	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213
5	6	42	M	executive	98101
6	7	57	M	administrator	91344
7	8	36	M	administrator	05201
8	9	29	M	student	01002
9	10	53	M	lawyer	90703

有时候，并不需要都看到所有字段，便可以从中筛选出一部分。如下所示：

```
>>> users[['user_id', 'age']].head(5)
```

	user_id	age
0	1	24
1	2	53
2	3	23
3	4	24
4	5	33

有时候只是排除个别字段。如下所示：

```
>>> users.exclude('zip_code', 'age').head(5)
```

	user_id	sex	occupation
0	1	M	technician
1	2	F	other
2	3	M	writer
3	4	M	technician
4	5	F	other

排除掉一些字段的同时，想要通过计算得到一些新的列，比如将sex为M的置为True，否则为False，并取名叫sex_bool。如下所示：

```
>>> users.select(users.exclude('zip_code', 'sex'), sex_bool=users.sex
 == 'M').head(5)
```

	user_id	age	occupation	sex_bool
0	1	24	technician	True
1	2	53	other	False
2	3	23	writer	True
3	4	24	technician	True
4	5	33	other	False

若想知道年龄在20到25岁之间的人有多少个，如下所示：

```
>>> users.age.between(20, 25).count().rename('count')
943
```

若想知道男女用户分别有多少，如下所示：

```
>>> users.groupby(users.sex).count()
```

	sex	count
0	F	273
1	M	670

若想将用户按职业划分，从高到底，获取人数最多的前10个职业，如下所示：

```
>>> df = users.groupby('occupation').agg(count=users['occupation'].
count())
>>> df.sort(df['count'], ascending=False)[:10]
```

	occupation	count
0	student	196
1	other	105
2	educator	95
3	administrator	79
4	engineer	67
5	programmer	66
6	librarian	51
7	writer	45
8	executive	32
9	scientist	31

DataFrame API提供了value_counts方法来快速达到同样的目的。如下所示：

```
>>> users.occupation.value_counts()[:10]
```

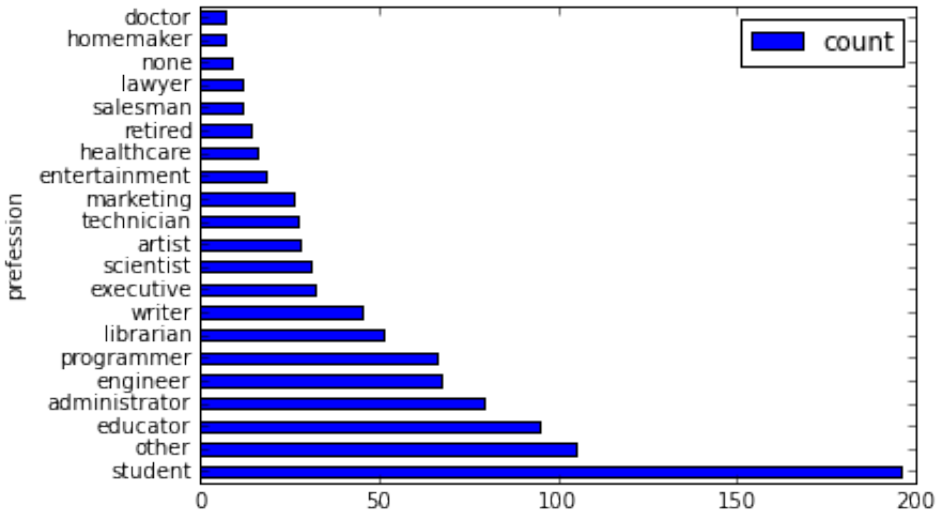
	occupation	count
0	student	196
1	other	105
2	educator	95
3	administrator	79
4	engineer	67
5	programmer	66
6	librarian	51
7	writer	45
8	executive	32
9	scientist	31

使用更直观的图来查看这份数据，如下所示：

```
>>> %matplotlib inline
```

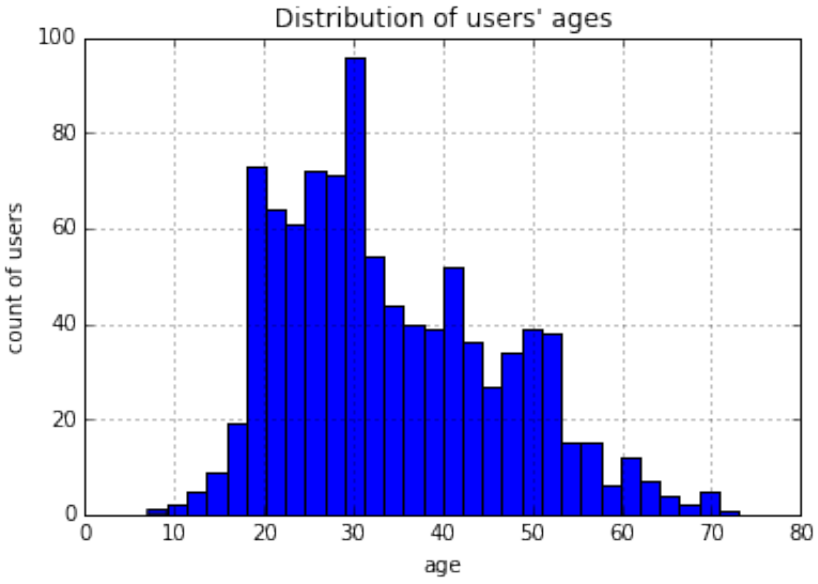
使用横向的柱状图来可视化，如下所示：

```
>>> users['occupation'].value_counts().plot(kind='barh', x='occupation
',
ylabel='prefession')
```



将年龄分成30组，查看各年龄分布的直方图，如下所示：

```
>>> users.age.hist(bins=30, title="Distribution of users' ages",
xlabel='age', ylabel='count of users')
```



使用join把这三张表进行联合后，把它保存成一张新的表。如下所示：

```
>>> movies = DataFrame(o.get_table('pyodps_ml_100k_movies'))
>>> ratings = DataFrame(o.get_table('pyodps_ml_100k_ratings'))
>>> o.delete_table('pyodps_ml_100k_lens', if_exists=True)
>>> lens = movies.join(ratings).join(users).persist('pyodps_ml_
100k_lens')
>>> lens.dtypes
```

```
odps.Schema {
  movie_id                                 int64
  title                                    string
  release_date                             string
  video_release_date                       string
  imdb_url                                 string
  user_id                                  int64
  rating                                   int64
  unix_timestamp                           int64
  age                                      int64
  sex                                      string
  occupation                               string
  zip_code                                 string
}
```

把0到80岁的年龄，分成8个年龄段，如下所示：

```
>>> labels = ['0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-
69', '70-79']
>>> cut_lens = lens[lens, lens.age.cut(range(0, 81, 10), right=False,
labels=labels).rename('年龄分组')]
```

取分组和年龄唯一的前10条数据来进行查看，如下所示：

```
>>> cut_lens['年龄分组', 'age'].distinct()[:10]
```

	年龄分组	age
0	0-9	7
1	10-19	10
2	10-19	11
3	10-19	13
4	10-19	14
5	10-19	15
6	10-19	16
7	10-19	17
8	10-19	18
9	10-19	19

对各个年龄分组下，用户的评分总数和评分均值进行查看，如下所示：

```
>>> cut_lens.groupby('年龄分组').agg(cut_lens.rating.count().rename('评分总数'), cut_lens.rating.mean().rename('评分均值'))
```

	年龄分组	评分均值	评分总数
0	0-9	3.767442	43
1	10-19	3.486126	8181
2	20-29	3.467333	39535
3	30-39	3.554444	25696
4	40-49	3.591772	15021
5	50-59	3.635800	8704
6	60-69	3.648875	2623
7	70-79	3.649746	197

Configuration

PyODPS提供了一系列的配置选项，可通过`odps.options`获得。可配置的MaxCompute选项，如下所示：

- 通用配置

选项	说明	默认值
end_point	MaxCompute Endpoint	None
default_project	默认Project	None
log_view_host	LogView主机名	None
log_view_hours	LogView保持时间（小时）	24
local_timezone	使用的时区，True表示本地时间，False表示UTC，也可用 pytz 的时区	1
lifecycle	所有表生命周期	None
temp_lifecycle	临时表生命周期	1
biz_id	用户ID	None
verbose	是否打印日志	False
verbose_log	日志接收器	None

选项	说明	默认值
chunk_size	写入缓冲区大小	1496
retry_times	请求重试次数	4
pool_connections	缓存在连接池的连接数	10
pool_maxsize	连接池最大容量	10
connect_timeout	连接超时	5
read_timeout	读取超时	120
completion_size	对象补全列举条数限制	10
notebook_repr_widget	使用交互式图表	True
sql.settings	ODPS SQL运行全局hints	None
sql.use_odps2_extension	启用MaxCompute2.0语言扩展	False

- 数据上传/下载配置

选项	说明	默认值
tunnel.endpoint	Tunnel Endpoint	None
tunnel.use_instance_tunnel	使用Instance Tunnel获取执行结果	True
tunnel.limited_instance_tunnel	限制Instance Tunnel获取结果的条数	True
tunnel.string_as_binary	在string类型中使用bytes而非unicode	False

- **DataFrame**配置

选项	说明	默认值
interactive	是否在交互式环境	根据检测值
df.analyze	是否启用非ODPS内置函数	True
df.optimize	是否开启DataFrame全部优化	True
df.optimizes.pp	是否开启DataFrame谓词下推优化	True
df.optimizes.cp	是否开启DataFrame列剪裁优化	True

选项	说明	默认值
df.optimizes.tunnel	是否开启DataFrame使用tunnel优化执行	True
df.quote	ODPS SQL后端是否用``来标记字段和表名	True
df.libraries	DataFrame运行使用的第三方库（资源名）	None

- **PyODPS ML配置**

选项	说明	默认值
ml.xflow_project	默认Xflow工程名	algo_public
ml.use_model_transfer	是否使用ModelTransfer获取模型PMML	True
ml.model_volume	在使用ModelTransfer时使用的Volume名称	pyodps_volume

7 处理非结构化数据

7.1 访问 OSS 非结构化数据

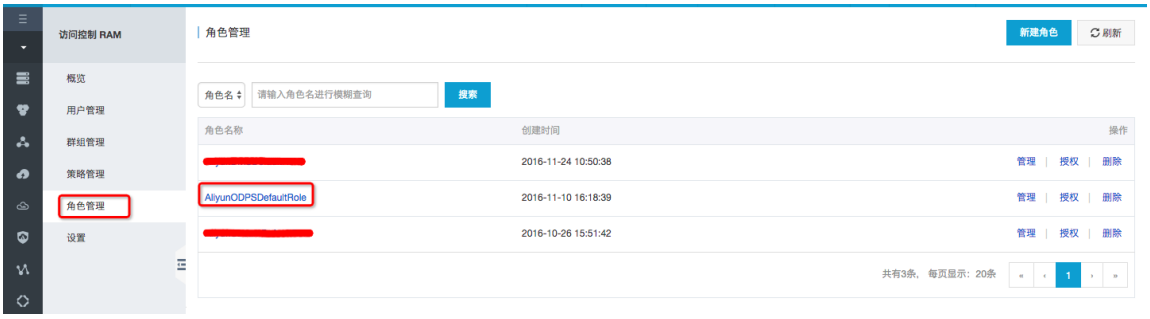
本文将带您实现在 MaxCompute 上轻松访问 OSS 的数据，关于处理非结构化数据的原理性介绍请参见 前言。

STS模式授予权限

MaxCompute需要直接访问OSS的数据，前提需要将OSS的数据相关权限赋给MaxCompute的访问账号，您可通过以下两种方式授予权限：

- 当**MaxCompute和OSS的owner是同一个账号时**，可以直接登录阿里云账号后，点击此处完成一键授权。
- 自定义授权。

1. 首先需要在 *RAM* 中授予 MaxCompute 访问 OSS 的权限。登录 *RAM* 控制台（若MaxCompute和OSS不是同一个账号，此处需由OSS账号登录进行授权），通过控制台中的 角色管理 创建角色 ，角色名如AliyunODPSDefaultRole或AliyunODPSRoleForOtherUser。如下图所示：



2. 修改角色策略内容设置，如下所示：

```
--当MaxCompute和OSS的Owner是同一个账号
{
"Statement": [
{
 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
}
```

```
--当MaxCompute和OSS的Owner不是同一个账号
{
"Statement": [
{
 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "MaxCompute的Owner云账号id@odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
}
```

3. 授予角色访问OSS必要的权限 **AliyunODPSRolePolicy** 。如下所示：

```
{
"Version": "1",
"Statement": [
{
 "Action": [
   "oss:ListBuckets",
   "oss:GetObject",
   "oss:ListObjects",
   "oss:PutObject",
   "oss:DeleteObject",
   "oss:AbortMultipartUpload",
   "oss:ListParts"
 ],
 "Resource": "*",
 "Effect": "Allow"
}
]
}
--可自定义其他权限
```

4. 将权限 **AliyunODPSRolePolicy** 授权给该角色。

内置extractor访问 OSS 数据

访问外部数据源时，需要用户自定义不同的 Extractor，同时您也可以使用 MaxCompute 内置的 Extractor，来读取按照约定格式存储的 *OSS*数据。只需要创建一个外部表，便可把这张表作为源表进行查询。

假设有一份 CSV 数据存在 *OSS*上，endpoint 为`oss-cn-shanghai-internal.aliyuncs.com`，bucket 为`oss-odps-test`，数据文件的存放路径为*`/demo/vehicle.csv`*。

创建外部表

创建外部表，语句如下：

```
CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_csv_external
(
```

```
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
STORED BY 'com.aliyun.odps.CsvStorageHandler' -- (1)
WITH SERDEPROPERTIES (
 'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole'
) -- (2)
LOCATION 'oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/
Demo/'; -- (3)(4)
```

上述语句，说明如下：

- `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。您只需指明这个名字，相关逻辑已经由系统实现。
- `odps.properties.rolearn`中的信息是 RAM 中`AliyunODPSDefaultRole`的`Arn`信息。您可以通过 RAM 控制台中的 角色详情 获取。
- 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。
  - 建议您使用 OSS 提供的内网域名，否则将产生 OSS 流量费用。
  - 建议您存放 OSS 数据的区域对应您开通 MaxCompute 的区域。由于 MaxCompute 只有在部分区域部署，我们不承诺跨区域的数据连通性。
  - OSS 的连接格式为`oss://oss-cn-shanghai-internal.aliyuncs.com/Bucket`名称/目录名称/。目录后不要加文件名称，如下的集中用法都是错误的：

    ```
    http://oss-odps-test.oss-cn-shanghai-internal.aliyuncs.com/Demo/
     -- 不支持http连接
    https://oss-odps-test.oss-cn-shanghai-internal.aliyuncs.com/Demo/
     -- 不支持https连接
    oss://oss-odps-test.oss-cn-shanghai-internal.aliyuncs.com/Demo
     -- 连接地址错误
    oss://oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/
    Demo/vehicle.csv  -- 不必指定文件名
    ```

- 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。

  更多有关外部表的说明请参见 *DDL* 语句。

如果想查看创建好的外部表结构信息，可以执行语句：

```
desc extended <table_name>;
```

在返回的信息里，除了跟内部表一样的基础信息外，“Extended Info”包含外部表StorageHandler 、Location 等信息。

查询外部表

外部表创建成功后，便可如同普通表一样使用这个外部表。假设/demo/vehicle.csv数据如下：

```
1,1,51,1,46.81006,-92.08174,9/14/2014 0:00,S
1,2,13,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,3,48,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,4,30,1,46.81006,-92.08174,9/14/2014 0:00,W
1,5,47,1,46.81006,-92.08174,9/14/2014 0:00,S
1,6,9,1,46.81006,-92.08174,9/14/2014 0:00,S
1,7,53,1,46.81006,-92.08174,9/14/2014 0:00,N
1,8,63,1,46.81006,-92.08174,9/14/2014 0:00,SW
1,9,4,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,10,31,1,46.81006,-92.08174,9/14/2014 0:00,N
```

执行如下 SQL 语句：

```
select recordId, patientId, direction from ambulance_data_csv_external
 where patientId > 25;
```

说明：

目前外部表只能通过MaxCompute SQL操作，MaxCompute MapReduce目前无法操作外部表。

这条语句会提交一个作业，调用内置 csv extractor，从 OSS 读取数据进行处理。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         |
+------------+------------+-----------+
```

自定义 Extractor 访问 OSS

当 OSS 中的数据格式比较复杂，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。

例如有一个 txt 数据文件，并不是 CSV 格式，记录之间的列通过|分隔。比如/demo/SampleData/CustomTxt/AmbulanceData/vehicle.csv数据如下：

```
1|1|51|1|46.81006|-92.08174|9/14/2014 0:00|S
1|2|13|1|46.81006|-92.08174|9/14/2014 0:00|NE
1|3|48|1|46.81006|-92.08174|9/14/2014 0:00|NE
1|4|30|1|46.81006|-92.08174|9/14/2014 0:00|W
1|5|47|1|46.81006|-92.08174|9/14/2014 0:00|S
1|6|9|1|46.81006|-92.08174|9/14/2014 0:00|S
1|7|53|1|46.81006|-92.08174|9/14/2014 0:00|N
1|8|63|1|46.81006|-92.08174|9/14/2014 0:00|SW
1|9|4|1|46.81006|-92.08174|9/14/2014 0:00|NE
1|10|31|1|46.81006|-92.08174|9/14/2014 0:00|N
```

- 定义 **Extractor**

  写一个通用的 Extractor，将分隔符作为参数传进来，可以处理所有类似格式的 text 文件。如下所示：

```
/**
 * 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 语句传参。
    if ( columnDelimiter != null)
    {
      this.columnDelimiter = columnDelimiter;
    }
    // note: more properties can be inited from attributes if needed
  }
  @Override
  public Record extract() throws IOException {//extactor() 调用返回一
条 Record，代表外部表中的一条记录。
    String line = readNextLine();
    if (line == null) {
```

```
      return null; // 返回 NULL 来表示这个表中已经没有记录可读。
    }
    return textLineToRecord(line); // textLineToRecord 将一行数据按照
delimiter 分割为多个列。
  }
  @Override
  public void close(){
    // no-op
  }
}
```

textLineToRecord 将数据分割的完整实现请参见 此处。

定义 **StorageHandler**

StorageHandler 作为 External Table 自定义逻辑的统一入口。

```
package com.aliyun.odps.udf.example.text;
public class TextStorageHandler extends OdpsStorageHandler {
  @Override
  public Class<? extends Extractor> getExtractorClass() {
    return TextExtractor.class;
  }
  @Override
  public Class<? extends Outputer> getOutputerClass() {
    return TextOutputer.class;
  }
}
```

编译打包

将自定义代码编译打包，并上传到 MaxCompute。

```
add jar odps-udf-example.jar;
```

- 创建 **External** 表

  与使用内置 Extractor 相似，首先需要创建一张外部表，不同的是在指定外部表访问数据的时候，需要使用自定义的 StorageHandler。

  创建外部表语句如下:

```
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.TextStorageHandler' --
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。
'odps.properties.rolearn'='acs:ram::xxxxxxxxxxxxx:role/aliyunodps
defaultrole'
)
LOCATION 'oss://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。
```

- 查询外部表

  执行如下 SQL 语句：

```
select recordId, patientId, direction from ambulance_data_txt_e
xternal where patientId > 25;
```

自定义 **Extractor** 访问非文本文件数据

在前面我们看到了通过内置与自定义的 Extractor 可以轻松处理存储在 OSS 上的 CSV 等文本数据。接下来以语音数据（wav 格式文件）为例，为您介绍如何通过自定义的 Extractor 访问并处理 OSS 上的非文本文件。

这里从最终执行的 **SQL** 开始，介绍以 **MaxCompute SQL** 为入口，处理存放在 **OSS** 上的语音文件的使用方法。

创建外部表表sql如下：

```
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://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/
dev/SpeechSentenceTest/'
USING 'odps-udf-example.jar,sm_random_5_utterance.text.label';
```

如上所示，同样需要创建外部表，然后通过外部表的 Schema 定义了希望通过外部表从语音文件中抽取出来的信息：

- 一个语音文件中的语句信噪比（SNR）：sentence_snr。
- 对应语音文件的名字：id。

创建外部表后，通过标准的 Select 语句进行查询，则会触发 Extractor 运行计算。此处便可感受到，在读取处理 OSS 数据时，除了可以对文本文件做简单的反序列化处理，还可以通过自定义

Extractor 实现更复杂的数据处理抽取逻辑。比如：在这个例子中，我们通过自定义的`com.aliyun.odps.udf.example.speech.SpeechStorageHandler` 中封装的 Extractor, 实现了对语音文件计算平均有效语句信噪比的功能，并将抽取出来的结构化数据直接进行 SQL 运算（WHERE sentence_snr > 10），最终返回所有信噪比大于 10 的语音文件以及对应的信噪比值。

在 OSS 地址`oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/dev/SpeechSentenceTest/`上，存储了原始的多个 WAV 格式的语音文件，MaxCompute 框架将读取该地址上的所有文件，并在必要的时候进行文件级别的分片，自动将文件分配给多个计算节点处理。每个计算节点上的 Extractor 则负责处理通过 InputStreamSet 分配给该节点的文件集。具体的处理逻辑则与用户单机程序相仿，您不需关心分布计算中的种种细节，按照类单机方式实现其用户算法即可。

定制化的`SpeechSentenceSnrExtractor`主体逻辑，说明如下：

首先在`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_A
TTRIBUTE_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());
    }
  }
```

Extractor() 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。

上述示例对实现进行了简化，同时也没有包括涉及语音处理的算法逻辑，具体实现请参见 MaxCompute SDK 在开源社区中提供的 样例代码。

```
@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);
    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
    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 {
    //  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 validBuffe
rLen){
    // computing the snr value for the wav file (supplied as byte
buffer array), skipped here
  }
```

执行查询，如下所示：

```
select sentence_snr, id
    from speech_sentence_snr_external
where sentence_snr > 10.0;
```

获得计算结果，如下所示：

```
-------------------------------------------------------------
| 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   |
```

```
------------------------------------------------------------
```

综上所述，通过自定义 Extractor，便可在 SQL 语句上分布式地处理多个 OSS 上的语音数据文件。同样的方法，也可以方便的利用 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。

数据的分区

在前面的例子中，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是读取“目录”下面的所有数据，**包括子目录中的所有文件**。在数据量比较大，尤其是对于随着时间不断积累的数据目录，对全目录扫描可能带来不必要的I/O以及数据处理时间。 解决这个问题通常有两种做法：

- 直接的方法:用户对数据存放地址做好规划，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。
- 数据分区方法:EXTERNAL TABLE与内部表一样，**支持分区表的功能**，可以通过这个功能来对数据做系统化的管理。

本章节主要介绍EXTERNAL TABLE的分区功能。

- **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，MaxComput没有数据的管理权，因此如果需要使用分区表功能，在OSS上数据文件的存放路径必须符合一定的格式，路径格式如下：

  ```
  partitionKey1=value1\partitionKey2=value2\...
  ```

  场景示例

  将每天产生的LOG文件存放在OSS上，并需要通过MaxCompute进行数据处理，数据处理时需按照粒度为“天”来访问一部分数据。假设这些LOG文件为CSV格式且可以用内置extractor访问（复杂自定义格式用法也类似），那么**外部分区表定义数据如下**：

  ```
  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'
    WITH SERDEPROPERTIES (
  ```

```
 'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrol
e'
)
  LOCATION 'oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/
log_data/';
```

如上建表语句，和前面的例子区别在于定义EXTERNAL TABLE时，通过PARTITIONED BY的语法指定该外部表为分区表，该例子是一个三层分区分区表，分区的key分别是 year， month 和 day。

为了让分区生效，在OSS上存储数据时需要遵循location的路径格式。如有效的路径存储layout:

```
osscmd ls oss://oss-odps-test/log_data/
2017-01-14 08:03:35 128MB Standard oss://oss-odps-test/log_data/year
=2016/month=06/day=01/logfile
2017-01-14 08:04:12 127MB Standard oss://oss-odps-test/log_data/year
=2016/month=06/day=01/logfile.1
2017-01-14 08:05:02 118MB Standard oss://oss-odps-test/log_data/year
=2016/month=06/day=02/logfile
2017-01-14 08:06:45 123MB Standard oss://oss-odps-test/log_data/year
=2016/month=07/day=10/logfile
2017-01-14 08:07:11 115MB Standard oss://oss-odps-test/log_data/year
=2016/month=08/day=08/logfile
...
```

说明：

因为数据是离线准备的,即通过osscmd或者其他OSS工具上载到OSS存储服务，所以数据路径格式也在上载时决定。

通过**ALTER TABLE ADD PARTITIONDDL**语句，即可把这些分区信息引入MaxCompute。

对应的DDL语句：

```
ALTER TABLE log_table_external ADD PARTITION (year = '2016', month =
 '06', day = '01')
ALTER TABLE log_table_external ADD PARTITION (year = '2016', month =
 '06', day = '02')
ALTER TABLE log_table_external ADD PARTITION (year = '2016', month =
 '07', day = '10')
ALTER TABLE log_table_external ADD PARTITION (year = '2016', month =
 '08', day = '08')
...
```

说明：

以上这些操作与标准的MaxCompute内部表操作一样，对分区表概念不熟悉的可以参考文档。

在数据准备好并且PARTITION信息引入ODPS之后，即可通过SQL语句对OSS外表数据的分区进行操作。

此时分析数据时，可以指定指需分析某天的数据，如只想分析2016年6月1号当天，有多少不同的IP出现在LOG里面，可以通过如下语句实现：

```
SELECT count(distinct(ip)) FROM log_table_external WHERE year = '
2016' AND month = '06' AND day = '01';
```

该语句对log_table_external这个外表对应的目录,将只访问`log_data/year=2016/month=06/day=01`子目录下的文件(logfile和logfile.1), 不会对整个log_data/目录作全量数据扫描，避免大量无用的**I/O**操作。

同样如果只希望对2016年下半年的数据做分析，则：

```
SELECT count(distinct(ip)) FROM log_table_external
WHERE year = '2016' AND month > '06';
```

只访问OSS上面存储的下半年的LOG数据。

- 分区数据在**OSS**上的自定义路径

  如果事先存在OSS上的历史数据，但是又不是根据`partitionKey1=value1\partitionKey2=value2\...`路径格式来组织存放，也需要通过MaxCompute的分区方式来进行访问计算时，MaxCompute也提供了通过自定义路径来引入partition的方法。

  假设OSS数据路径只有简单的分区值（而无分区key信息）,也就是数据的layout为：

```
osscmd ls oss://oss-odps-test/log_data_customized/
2017-01-14 08:03:35 128MB Standard oss://oss-odps-test/log_data_c
ustomized/2016/06/01/logfile
2017-01-14 08:04:12 127MB Standard oss://oss-odps-test/log_data_c
ustomized/2016/06/01/logfile.1
2017-01-14 08:05:02 118MB Standard oss://oss-odps-test/log_data_c
ustomized/2016/06/02/logfile
2017-01-14 08:06:45 123MB Standard oss://oss-odps-test/log_data_c
ustomized/2016/07/10/logfile
2017-01-14 08:07:11 115MB Standard oss://oss-odps-test/log_data_c
ustomized/2016/08/08/logfile
...
```

外部表建表DDL可参看前面的例子，同样在建表语句里指定好分区key。

不同的子目录指定到不同的分区，可通过类似如下自定义分区路径的DDL语句实现：

```
ALTER TABLE log_table_external ADD PARTITION (year = '2016', month = '
06', day = '01')
```

```
LOCATION 'oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/
log_data_customized/2016/06/01/';
```

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了。

7.2 访问OTS非结构化数据

表格存储（Table Store）是构建在阿里云飞天分布式系统之上的NoSQL数据存储服务，提供海量结构化数据的存储和实时访问。您可以通过*TableStore*文档对其进行了解。

MaxCompute与TableStore是两个独立的大数据计算和存储服务，所以两者之间的网络必须保证连通性。MaxCompute公共云服务访问TableStore存储时，推荐您使用TableStore私网地址，也就是host名以ots-internal.aliyuncs.com作为结尾的地址，例如：tablestore://odps-ots-dev.cn-shanghai.ots-internal.aliyuncs.com。

前文为您介绍了如何访问*OSS*非结构化数据，本文将进一步为您介绍如何将来自TableStore（OTS）的数据纳入MaxCompute上的计算生态，实现多种数据源之间的无缝连接。

TableStore与MaxCompute都有其自身的类型系统。在MaxCompute处理TableStore数据时，两者之间的类型对应关系如下所示：

MaxCompute Type	TableStore Type
STRING	STRING
BIGINT	INTEGER
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
BINARY	BINARY

STS模式授权

MaxCompute计算服务访问Table Store数据需要有一个安全的授权通道。在此问题上，MaxCompute结合了阿里云的访问控制服务（RAM）和令牌服务（STS）来实现对数据的安全访问。

您可以通过以下两种方式授予权限：

- 当MaxCompute和Table Store的Owner是同一个账号时，登录阿里云账号后，单击此处完成一键授权。
- 自定义授权

1. 首先在RAM控制台中授予MaxCompute访问Table Store的权限。

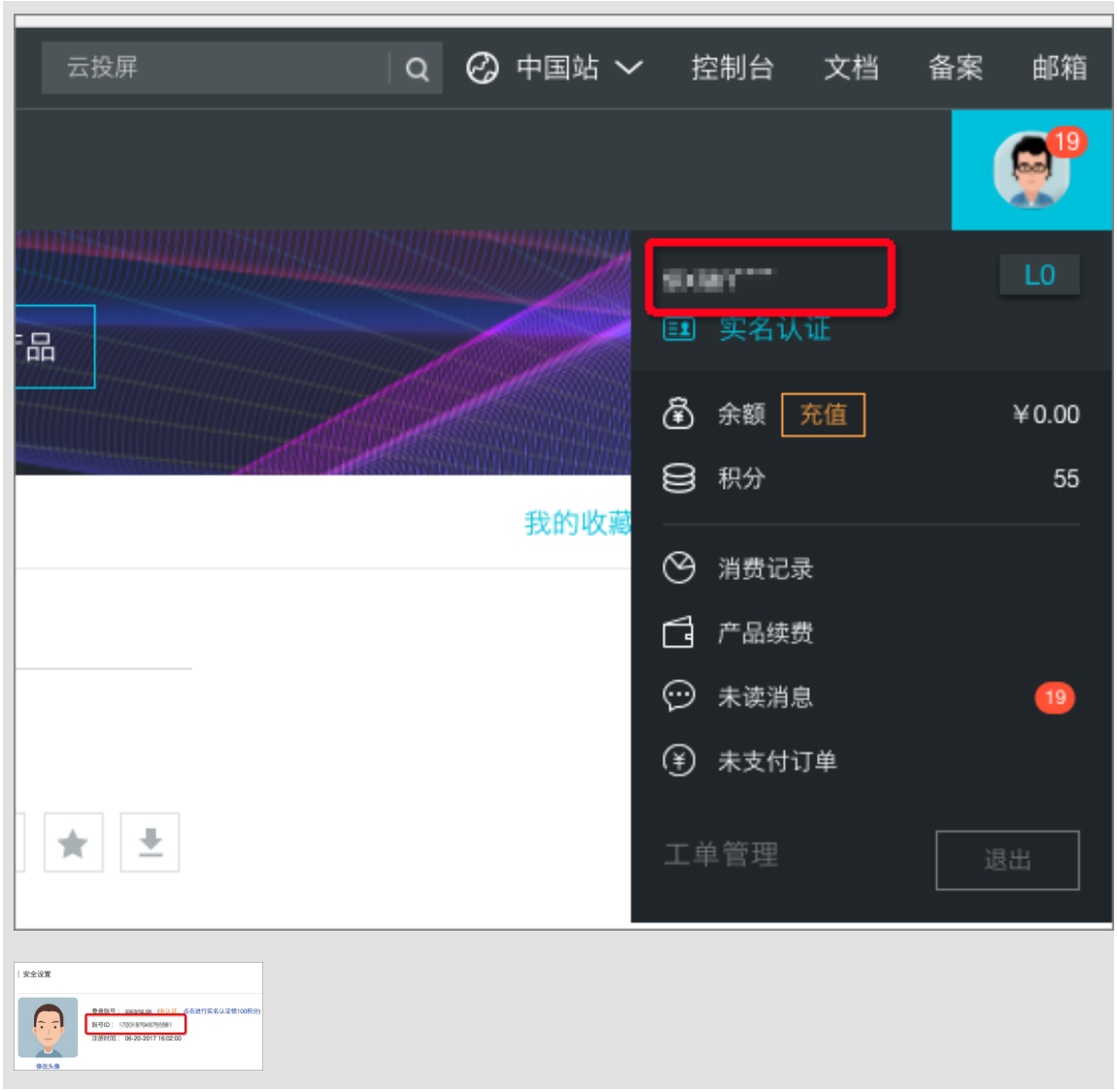
   登录 *RAM*控制台（若MaxCompute和Table Store不是同一个账号，此处需由Table Store账号登录进行授权），创建角色，角色名叫AliyunODPSDefaultRole或AliyunODPSRoleForOtherUser 。

2. 修改策略内容设置，如下所示：

```
--当MaxCompute和Table Store的Owner是同一个账号
{
"Statement": [
{
 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
}
--当MaxCompute和Table Store的Owner不是同一个账号
{
"Statement": [
{
 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "MaxCompute的Owner云账号的UID@odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
}
```

说明：

您可单击右上角的登录账号，进入账号管理页面查看云账号的UID。



3. 编辑该角色的授权策略AliyunODPSRolePolicy，如下所示：

```
{
"Version": "1",
"Statement": [
{
 "Action": [
   "ots:ListTable",
   "ots:DescribeTable",
   "ots:GetRow",
   "ots:PutRow",
   "ots:UpdateRow",
   "ots:DeleteRow",
   "ots:GetRange",
   "ots:BatchGetRow",
   "ots:BatchWriteRow",
   "ots:ComputeSplitPointsBySize"
 ],
 "Resource": "*",
 "Effect": "Allow"
}
]
}
```

```
--还可自定义其他权限
```

4. 将权限AliyunODPSRolePolicy授权给该角色。

创建外部表

MaxCompute通过创建外部表，把对TableStore表数据的描述引入到MaxCompute的meta系统内部后，即可轻松实现对TableStore数据的处理。本节将以下述示例为例，来为您说明 axCompute对接TableStore的一些概念和实现。

建外部表语句如下：

```
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' -- (1)
WITH SERDEPROPERTIES ( -- (2)
'tablestore.columns.mapping'=':o_orderkey,:o_orderdate,o_custkey,
o_orderstatus,o_totalprice', -- (3)
'tablestore.table.name'='ots_tpch_orders' -- (4)
'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole'
)
LOCATION 'tablestore://odps-ots-dev.cn-shanghai.ots-internal.aliyuncs.
com'; -- (5)
```

语句说明如下所示：

- 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。
- SERDEPROPERITES是提供参数选项的接口，在使用TableStoreStorageHandler时，有两个必须指定的选项，分别是下面将会介绍的tablestore.columns.mapping 、tablestore.table.name和odps.properties.rolearn。
  - tablestore.columns.mapping选项：必选项，用来描述MaxCompute将访问的Table Store表的列，包括主键和属性列。
    - 以:打头的用来表示Table Store主键，例如此语句中的`:o_orderkey`和`:o_orderdate`，其他的均为属性列。
    - Table Store支持1-4个主键，主键类型为String、Integer和Binary，其中第一个主键为分区键。

- 在指定映射时，您必须提供指定Table Store表的所有主键，对于属性列则没有必要全部提供，可以只提供需要通过MaxCompute来访问的属性列。
- tablestore.table.name：需要访问的Table Store表名。如果指定的Table Store表名错误（不存在），则会报错，MaxCompute不会主动去创建Table Store表。
- odps.properties.rolearn中的信息是RAM中AliyunODPSDefaultRole的Arn信息。您可以通过RAM控制台中的角色详情进行获取。

- LOCATION clause：用来指定Table Storeinstance名字、endpoint等具体信息。这里的Table Store数据的安全访问建立在前文介绍的RAM/STS授权的前提上。

如果您想要查看创建好的外部表结构信息，可以执行如下语句：

```
desc extended <table_name>;
```

在返回的信息里，除了跟内部表一样的基础信息外，Extended Info包含外部表StorageHandler、Location等信息。

查询外部表

创建External Table后，Table Store的数据便引入到了MaxCompute生态中，即可通过正常的MaxCompute SQL语法访问Table Store数据，如下所示：

```
SELECT odps_orderkey, odps_orderdate, SUM(odps_totalprice) AS
sum_total
FROM ots_table_external
WHERE odps_orderkey > 5000 AND odps_orderkey < 7000 AND odps_orderdate
 >= '1996-05-03' AND odps_orderdate < '1997-05-01'
GROUP BY odps_orderkey, odps_orderdate
HAVING sum_total> 400000.0;
```

由上可见，使用常见的MaxCompute SQL语法，访问Table Store的所有细节由MaxComput 内部处理。这包括在列名的选择上，比如上述SQL中，使用的列名是odps_orderkey，odps_totalprice等，而不是原始Table Store中的主键名o_orderkey或属性列名o_totalprice，因为在创建External Table的DDL语句中，已经做了对应的mapping。当然您也可根据自己的需求在创建External Table时选择保留原始的TableStore主键/列名。

如果需要对一份数据做多次计算，相较每次从Table Store去远程读数据，有个更高效的办法是先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 ;
```

现在internal_orders就是一个MaxCompute表了，也拥有所有MaxCompute内部表的特性，包括高效的压缩列存储数据格式、完整的内部宏数据以及统计信息等。同时因为存储在MaxCompute内部，访问速度会比访问外部的Table Store更快，尤其适用于需要进行多次计算的热点数据。

MaxCompute导出数据到Table Store

说明：

MaxCompute不会主动创建外部的Table Store表，所以在对Table Store表进行数据输出之前，必须保证该表已经在Table Store上创建过（否则将报错）。

根据上面的操作，您已创建了外部表ots_table_external来打通MaxCompute与Table Storeb数据表ots_tpch_orders的链路，同时还有一份存储在MaxCompute内部表internal_orders的数据，现在希望对internal_orders中的数据进行一定处理后再写回 Table Store，可通过对外部表做**INSERT OVERWITE TABLE**操作来实现，如下所示：

```
INSERT OVERWRITE TABLE ots_table_external
SELECT odps_orderkey, odps_orderdate, odps_custkey, CONCAT(odps_custk
ey, 'SHIPPED'), CEIL(odps_totalprice)
FROM internal_orders;
```

对于Table Store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，比如示例中只影响所有odps_orderkey + odps_orderdate这两个主键值能对应行上的数据。而且在这些Tabele Store行上面，也只会去更新在创建External Table（ots_table_external）时指定的属性列，而不会去修改未在External Table中出现的数据列。更多详情请参见*MaxCompute访问TableStore#OTS#数据*。

8 图模型

8.1 图模型概述

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 支持以下图编辑操作：

- 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，点与边的关系需要您来维护。

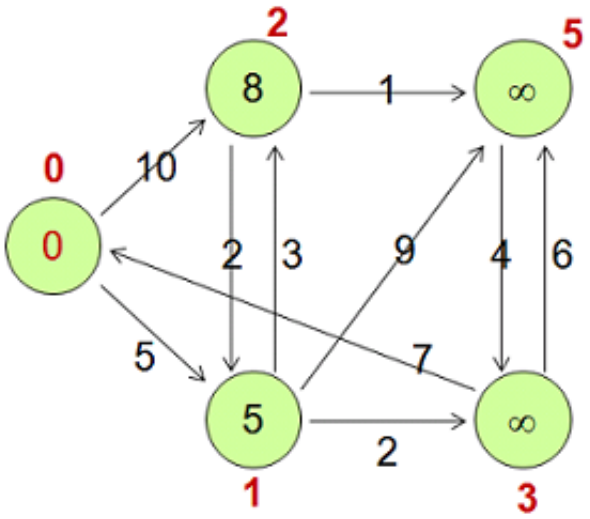
通过迭代对图进行编辑、演化，最终求解出结果，典型应用有：*PageRank*，单源最短距离算法，*K*-均值聚类算法 等。您可以使用 MaxCompute Graph 提供的接口 Java SDK 编写图计算程序。

Graph 数据结构

MaxCompute Graph 能够处理的图必须是是一个由点（Vertex）和边（Edge）组成的有向图。由于 MaxCompute 仅提供二维表的存储结构，因此需要您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，您可以根据自身的业务场景做决定。

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），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）。



例如，上图由下面的点组成：

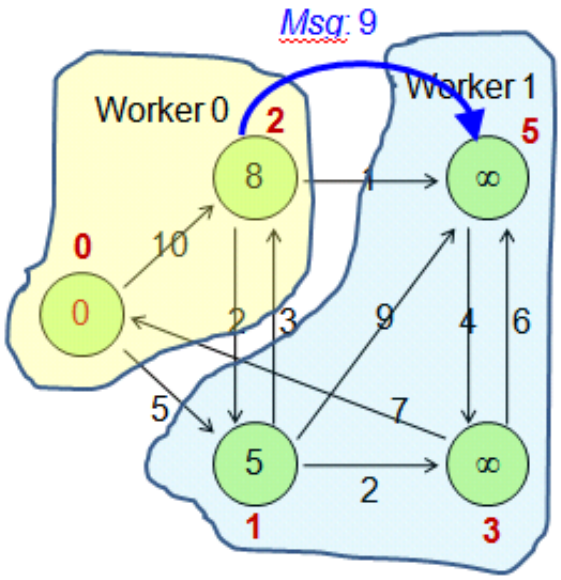
Vertex	<ID, Value, Halted, Edges>
v0	<0, 0, false, [<1, 5 >, <2, 10 >] >
v1	<1, 5, false, [<2, 3>, <3, 2>, <5, 9>]>
v2	<2, 8, false, [<1, 2>, <5, 1 >]>
v3	<3, Long.MAX_VALUE, false, [<0, 7>, <5, 6>]>
v5	<5, Long.MAX_VALUE, false, [<3, 4 >]>

Graph 程序逻辑

图加载

图加载：框架调用您自定义的 GraphLoader，将输入表的记录解析为点或边。

分布式化：框架调用您自定义的 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。



例如，上图假设 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。

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。
- 在您实现的 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，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，您无需关心。

迭代终止

满足以下任意一条，迭代即终止。

- 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。

伪代码描述如下所示：

```
// 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();
```

8.2 功能概述

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。

简要介绍如下：

```
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
```

其中 <GENERIC_OPTIONS> 包括以下参数（均为可选）：

- -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。

  大多数情况下，用户更习惯于将 main 函数与 Graph 作业编写在一个包中，例如：单源最短距离算法。因此，在执行示例程序时，-resources 及 -classpath 的参数中都出现了用户的 Jar 包，但二者意义不同：-resources 引用的是 Graph 作业，运行于分布式环境中，而 -classpath 引用的是 main 函数，运行于本地，指定的 Jar 包路径也是本地文件路径。包名之间使用系统默认的文件分割符作分割（通常情况下，windows 系统是分号，linux 系统是冒号）。
- -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 >：Graph 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。如果您在 Graph 作业中读取了其他 MaxCompute 资源，那么，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，使用跨项目空间使用资源时，需要在前面加上：PROJECT_NAME/resources/。示例：`-resources otherproject/resources/resfile;`。

同时，您也可以直接运行 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，如下所示：

```
public static void main(String[] args) throws Exception {
  if (args.length < 2)
    printUsage();
  Account account = new AliyunAccount(accessId, accessKey);
  Odps odps = new Odps(account);
  odps.setEndpoint(endPoint);
  odps.setDefaultProject(project);
  SessionState ss = SessionState.get();
  ss.setOdps(odps);
  ss.setLocalRun(false);
  String resource = "mapreduce-examples.jar";
  GraphJob job = new GraphJob();
  // 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.addCacheResourcesToClassPath(resource);
  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");
}
```

输入及输出

MaxCompute Graph 作业不支持您自定义输入与输出格式。

定义作业输入，支持多路输入，如下所示：

```
GraphJob job = new GraphJob();
job.addInput(TableInfo.builder().tableName(“tblname”).build());  //表作为输入
job.addInput(TableInfo.builder().tableName(“tblname”).partSpec("pt1=a/pt2=b").build()); //分区作为输入
//只读取输入表的 col2 和 col0 列，在 GraphLoader 的 load 方法中，record.get(0) 得到的是col2列，顺序一致
job.addInput(TableInfo.builder().tableName(“tblname”).partSpec("pt1=a/pt2=b").build(), new String[]{"col2", "col0"});
```

说明：

- 关于作业输入定义，更多详情请参见 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。
- 限制：暂时不支持分区过滤条件。更多应用限制请参见 应用限制。

定义作业输出，支持多路输出，通过 label 标识每路输出。如下所示：

```
GraphJob job = new GraphJob();
//输出表为分区表时需要给到最末一级分区
job.addOutput(TableInfo.builder().tableName("table_name").partSpec("pt1=a/pt2=b").build());
// 下面的参数 true 表示覆盖tableinfo指定的分区，即INSERT OVERWRITE语义，false表示INSERT INTO语义
job.addOutput(TableInfo.builder().tableName("table_name").partSpec("pt1=a/pt2=b").lable("output1").build(), true);
```

说明：

- 关于作业输出定义，更多详情请参见 GraphJob 的 addOutput 相关方法说明。
- 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，如上面的 output1。

- 更多应用限制请参见 应用限制。

读取资源

- **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 的以下两种方法指定：

```
void addCacheResources(String resourceNames)
void addCacheResourcesToClassPath(String resourceNames)
```

- **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 resourceNa
me);
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 方法中读取资源，然后保存在 Worker Value 中，之后通过 getWorkerValue 方法获取。
  - 建议用上面的流接口，边读边处理，内存耗费少。
  - 更多应用限制请参见 应用限制。

8.3 SDK概述

如果您使用 Maven，可以从 *Maven* 库 中搜索“odps-sdk-graph”来获取不同版本的 Java SDK。相关配置信息，如下所示：

```
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-sdk-graph</artifactId>
    <version>0.20.7</version>
```

```
</dependency>
```

主要接口	说明
GraphJob	GraphJob继承自JobConf，用于定义、提交和管理一个 MaxCompute Graph 作业。
Vertex	Vertex 是图的点的抽象，包含属性：id，value，halted，edges，通过 GraphJob 的 setVertexClass 接口提供 Vertex 实现。
Edge	Edge 是图的边的抽象，包含属性：destVertexId, value，图数据结构采用邻接表，点的出边保存在点的 edges 中。
GraphLoader	GraphLoader 用于载入图，通过 GraphJob 的 setGraphLoaderClass 接口提供 GraphLoader 实现。
VertexResolver	VertexResolver 用于自定义图拓扑修改时的冲突处理逻辑，通过 GraphJob的 setLoadingVertexResolverClass 和 setComputingVertexResolverClass 接口提供图加载和迭代计算过程中的图拓扑修改的冲突处理逻辑。
Partitioner	Partitioner 用于对图进行划分使得计算可以分片进行，通过GraphJob的 setPartitionerClass 接口提供 Partitioner 实现，默认采用 HashPartitioner，即对点 ID 求哈希值然后对 Worker 数目取模。
WorkerComputer	WorkerComputer允许在 Worker 开始和退出时执行用户自定义的逻辑，通过GraphJob的 setWorkerComputerClass 接口提供WorkerComputer 实现。
Aggregator	Aggregator 的 setAggregatorClass(Class …) 定义一个或多个 Aggregator。
Combiner	Combiner 的 setCombinerClass 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。

8.4 开发和调试

MaxCompute 并未为您提供 Graph 开发插件，但您仍然可以基于 Eclipse 开发 MaxCompute Graph 程序，推荐的开发流程，如下所示：

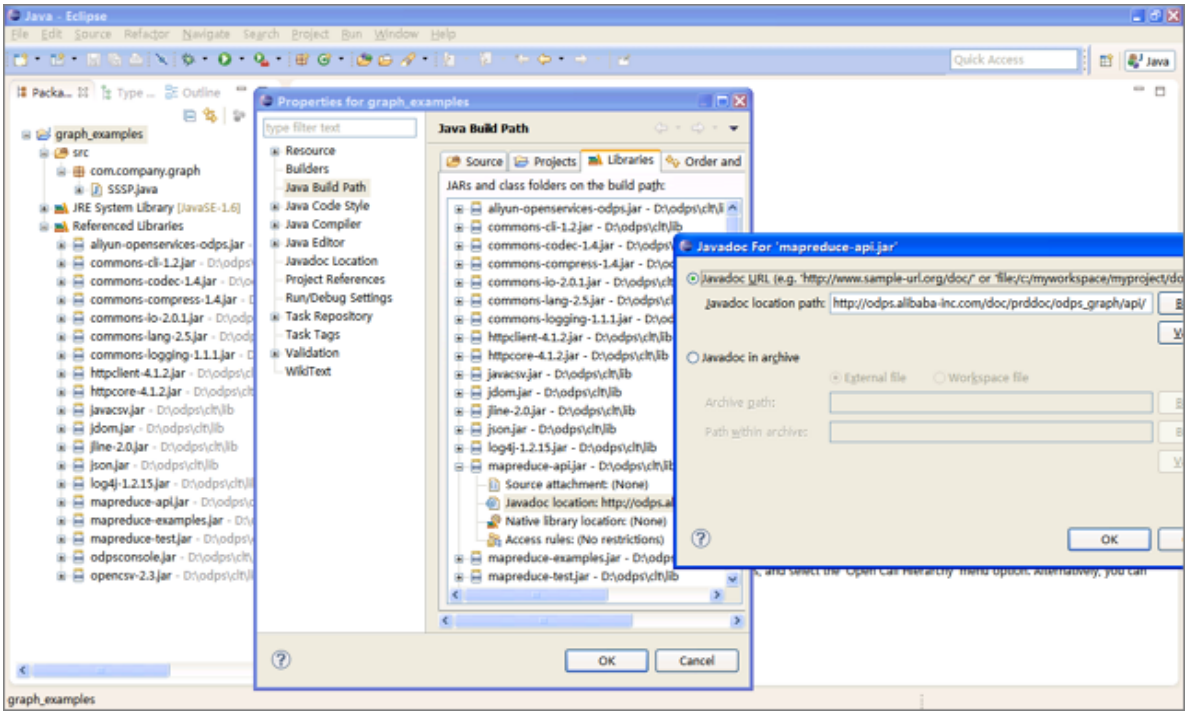
1. 编写 Graph 代码，使用本地调试进行基本的测试。
2. 进行集群调试，验证结果。

开发示例

本节以 *SSSP* 算法为例，为您介绍如何用 Eclipse 开发和调试 Graph 程序。

操作步骤

1. 创建 Java 工程，假设工程名为 graph_examples。
2. 将 MaxCompute 客户端 lib 目录下的 Jar 包加到 Eclipse 工程的 Build Path 中。配置好的 Eclipse 工程如下图所示：



3. 开发 MaxCompute Graph 程序。

   实际开发过程中，常常会先 copy 一个例子（例如*SSSP*），然后进行修改。在本示例中，我们仅修改了 package 的路径为：package com.aliyun.odps.graph.example。

4. 编译打包。

   在 Eclipse 环境中，右键单击源代码目录（图中的 src 目录），Export -> Java -> JAR file 生成 Jar 包，选择目标 Jar 包的保存路径，例如：`D:\\odps\\clt\\odps-graph-example-sssp.jar`。

5. 使用 MaxCompute 客户端运行 SSSP，更多详情请参见 运行 *Graph*。

> **说明：**
> 相关的开发步骤请参见 *Graph* 开发插件介绍。

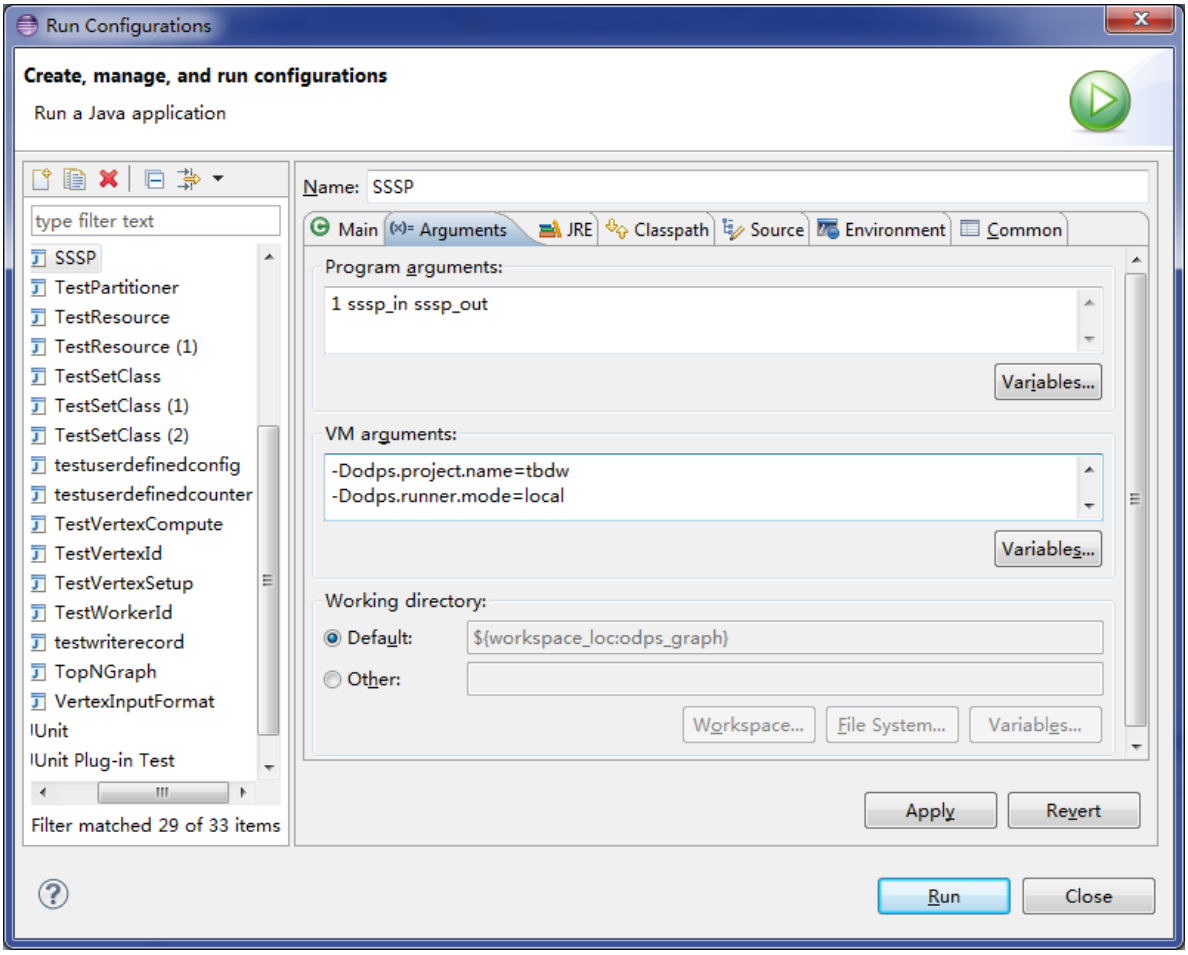
本地调试

MaxCompute Graph 支持本地调试模式，可以使用 Eclipse 进行断点调试。

操作步骤

1. 下载一个 odps-graph-local 的 maven 包。
2. 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（Run As -> Run Configurations…）。
3. 在 Arguments tab 页中，设置 Program arguments 参数为 `1 sssp_in sssp_out`，作为主程序的输入参数。
4. 在 Arguments tab 页中，设置 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>
```



5. 对于本地模式（即不指定 odps.end.point 参数），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"
```

关于 warehouse 的介绍请参见 *MapReduce* 本地运行。

6. 单击 **Run**，即可本地跑 SSSP。

说明：

参数设置可参见 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。若不指定，只从 warehouse 读取表或资源的 meta 和数据，不存在则抛异常。若指定，会先从 warehouse 读取，不存在时会远程连接 MaxCompute 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 中本地跑 SSSP 的调试输出信息，如下所示：

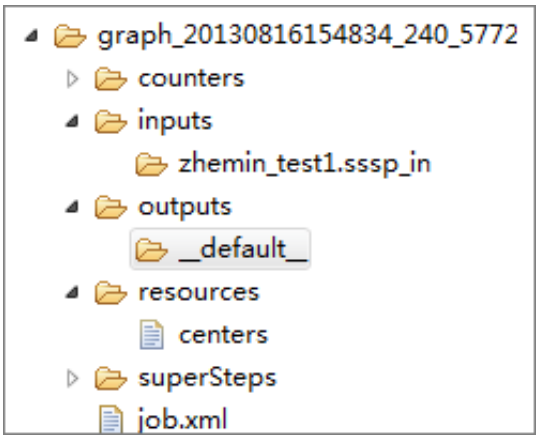
```
Counters: 3
        com.aliyun.odps.graph.local.COUNTER
                TASK_INPUT_BYTE=211
                TASK_INPUT_RECORD=5
                TASK_OUTPUT_BYTE=161
                TASK_OUTPUT_RECORD=5
 graph task finish
```

说明：

上述示例中，需要本地 warehouse 下有 sssp_in 及 sssp_out 表。sssp_in 及 sssp_out 的详细信息请参见 运行 *Graph*。

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：



本地运行的 Graph 作业，临时目录包括以下几个目录和文件：

- 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，但是也不能超过 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。

> 说明：
> 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`。

集群调试

通过本地的调试后，您即可提交作业到集群进行测试。

操作步骤

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。

> 说明：
> 集群运行 Graph 的详细介绍请参见 运行 *Graph*。

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。

作业参数配置

对性能有所影响的 GraphJob 配置项，如下所示：

- 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，范围为 [1, 1000]，默认值为 -1，worker 数由作业输入字节数和 split size 决定。
- setWorkerCPU(int)：Map CPU 资源，100 为 1cpu 核，[50，800] 之间，默认值为 200。
- setWorkerMemory(int)：Map 内存资源，单位 MB，范围为 [256M，12G]，默认值为 4096M。
- setMaxIteration(int)：设置最大迭代次数，默认为 -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，范围为 [0, 9]，默认值为 9，数值越大优先级越小。

通常情况下：

- 可以考虑使用 setNumWorkers 方法增加 worker 数目。
- 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。
- 加大 worker 的 CPU 或内存。
- 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，如下所示：

- 若 splitNum == workerNum，每个 worker 负责载入一个 split。
- 若 splitNum > workerNum，每个 worker 负责载入一个或多个 split。
- 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：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：

- 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。

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。

减少数据输入量

数据量大时，读取磁盘中的数据可能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。您可选择如下方法：

- 减少输入数据量：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，这样也可以减少输入的数据量，提高作业性能。

内置 **Jar** 包

下面这些 Jar 包会默认加载到运行 Graph 程序的 JVM 中，您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

说明：

在 JVM 的 Classpath 里，上述内置 Jar 包会放在您的 Jar 包的前面，所以可能产生版本冲突，例如：您的程序中使用了 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，这时只能看 1.3 版本里是否有满足您需求的实现，或者等待 MaxCompute 升级新版本。

8.5 应用限制

MaxCompute Graph 的应用限制，如下所示：

- 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，超过抛异常。
- 单个 worker 占用 CPU 默认为 200，范围为 [50, 800]。
- 单个 worker 占用 memory 默认为 4096，范围为 [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，您可自行设置，范围为：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。

8.6 示例程序

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，简称为 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，如下所示：

- 初始化：源点 s 到 s 自身的距离（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。

由算法基本原理可以看出，此算法非常适合使用 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，迭代结束。

代码示例

单源最短距离的代码，如下所示：

```
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";

  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();
      }
    }

    @Override
    public void cleanup(
        WorkerContext<LongWritable, LongWritable, LongWritable,
LongWritable> context)
        throws IOException {
      context.write(getId(), getValue());
    }
  }

  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());
      }
    }

  }

  public static class SSSPVertexReader extends
      GraphLoader<LongWritable, LongWritable, LongWritable, LongWritab
le> {

    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, LongWritable, LongWritable,
LongWritable> context)
        throws IOException {
      SSSPVertex vertex = new SSSPVertex();
      vertex.setId((LongWritable) record.get(0));
      String[] edges = record.get(1).toString().split(",");
      for (int i = 0; i < edges.length; i++) {
        String[] ss = edges[i].split(":");
        vertex.addEdge(new LongWritable(Long.parseLong(ss[0])),
            new LongWritable(Long.parseLong(ss[1])));
      }

      context.addVertexRequest(vertex);
    }

  }

  public static void main(String[] args) throws IOException {
    if (args.length < 2) {
      System.out.println("Usage: <startnode> <input> <output>");
      System.exit(-1);
    }

    GraphJob job = new GraphJob();
    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");
  }
}
```

上述代码，说明如下：

- 第 19 行：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。

- 第 58 行：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。
- 第 70 行：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。
- 第 83 行：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。
- 第 106 行：主程序（main 函数），定义 GraphJob，指定 Vertex/GraphLoader/Combiner 等的实现，指定输入输出表。

8.6.2 PageRank

PageRank 算法是计算网页排名的经典算法：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，那么存在连接 A 到 B 的边。

算法基本原理，如下所示：

- 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。

由算法基本原理可以看出，此算法非常适合使用 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。

代码示例

```
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);

  public static class PageRankVertex extends
      Vertex<Text, DoubleWritable, NullWritable, DoubleWritable> {

    @Override
    public void compute(
        ComputeContext<Text, DoubleWritable, NullWritable, DoubleWrit
able> 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, DoubleWrit
able> context)
        throws IOException {
      context.write(getId(), getValue());
    }
  }

  public static class PageRankVertexReader extends
      GraphLoader<Text, DoubleWritable, NullWritable, DoubleWritable>
 {

    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<Text, DoubleWritable, NullWritable, DoubleWrit
able> 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();

    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");
  }
}
```

上述代码，说明如下：

- 第 23 行：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。
- 第 55 行：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。

- 第 88 行：主程序（main 函数），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。

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：以空间中 k 个点为中心进行聚类，对最靠近它们的点进行归类。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，算法描述如下：

1. 适当选择 k 个类的初始中心。
2. 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。
3. 利用均值等方法更新该类的中心值。
4. 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。

代码示例

K-均值聚类算法的代码，如下所示：

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.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);

  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());
    }

  }

  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();
    }

  }

  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()
```

```
              - ((DoubleWritable) oldCenter.get(j)).get();
          sum += v * v;
        }
        double dist = Math.sqrt(sum);
        LOG.info("old center: " + oldCenter + ", new center: " +
newCenter
            + ", dist: " + dist);
        // converge threshold for each center: 0.05
        converged = dist < 0.05d;
      }

      if (converged || context.getSuperstep() == context.getMaxIter
ation() - 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);
  }

  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.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");
  }
}
```

上述代码，说明如下：

- 第 26 行：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）。
- 第 38 行：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。
- 第 83 行：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。
- 第 236 行：主程序（main 函数），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。
- 第 243 行：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，以提升加载图时的性能。

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。

算法描述，如下所示：

- 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。

代码示例

BiPartiteMatchiing 算法的代码，如下所示：

```
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> [maxIterati
on]");
  }
  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();
  }
}
```

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，如下所示：

- 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。

算法包含以下四部分：

- 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，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。该步结束后如果仍有活跃顶点，则回到修剪步。

代码示例

强连通分量的代码，如下所示：

```
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, LongWritab
le> 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, LongWritab
le> context)
        throws IOException {
      context.write(getId(), getValue().getSccID());
    }
  }
  /**
   * The SCCAggrValue maintains global stage and graph updated and
active status.
   * updated is true only if one vertex is updated.
   * active is true only if one vertex is active.
   */
  public static class SCCAggrValue implements Writable {
    IntWritable stage = new IntWritable(STAGE_TRANSPOSE_1);
    BooleanWritable updated = new BooleanWritable(false);
    BooleanWritable active = new BooleanWritable(false);
    public void setStage(int stage) {
      this.stage.set(stage);
    }
    public int getStage() {
      return this.stage.get();
    }
    public void setUpdated(boolean updated) {
      this.updated.set(updated);
    }
    public boolean getUpdated() {
      return this.updated.get();
    }
    public void setActive(boolean active) {
      this.active.set(active);
    }
    public boolean getActive() {
      return this.active.get();
    }
    @Override
    public void write(DataOutput out) throws IOException {
      this.stage.write(out);
      this.updated.write(out);
      this.active.write(out);
    }
    @Override
    public void readFields(DataInput in) throws IOException {
      this.stage.readFields(in);
      this.updated.readFields(in);
      this.active.readFields(in);
    }
  }
  /**
   * The job of SCCAggregator is to schedule global stage in every
superstep.
   */
  public static class SCCAggregator extends Aggregator<SCCAggrValue> {
    @SuppressWarnings("rawtypes")
    @Override
```

```
    public SCCAggrValue createStartupValue(WorkerContext context)
throws IOException {
      return new SCCAggrValue();
    }
    @SuppressWarnings("rawtypes")
    @Override
    public SCCAggrValue createInitialValue(WorkerContext context)
        throws IOException {
      return (SCCAggrValue) context.getLastAggregatedValue(0);
    }
    @Override
    public void aggregate(SCCAggrValue value, Object item) throws
IOException {
      MyValue v = (MyValue)item;
      if ((value.getStage() == STAGE_FW_REST || value.getStage() ==
STAGE_BW_REST)
          && v.isUpdated()) {
        value.setUpdated(true);
      }
      // only active vertex invoke aggregate()
      value.setActive(true);
    }
    @Override
    public void merge(SCCAggrValue value, SCCAggrValue partial)
        throws IOException {
      boolean updated = value.getUpdated() || partial.getUpdated();
      value.setUpdated(updated);
      boolean active = value.getActive() || partial.getActive();
      value.setActive(active);
    }
    @SuppressWarnings("rawtypes")
    @Override
    public boolean terminate(WorkerContext context, SCCAggrValue value
)
        throws IOException {
      // If all vertices is inactive, job is over.
      if (!value.getActive()) {
        return true;
      }
      // state machine
      switch (value.getStage()) {
      case STAGE_TRANSPOSE_1:
        value.setStage(STAGE_TRANSPOSE_2);
        break;
      case STAGE_TRANSPOSE_2:
        value.setStage(STAGE_TRIMMING);
        break;
      case STAGE_TRIMMING:
        value.setStage(STAGE_FW_START);
        break;
      case STAGE_FW_START:
        value.setStage(STAGE_FW_REST);
        break;
      case STAGE_FW_REST:
        if (value.getUpdated()) {
          value.setStage(STAGE_FW_REST);
        } else {
          value.setStage(STAGE_BW_START);
        }
        break;
      case STAGE_BW_START:
        value.setStage(STAGE_BW_REST);
```

```
        break;
      case STAGE_BW_REST:
        if (value.getUpdated()) {
          value.setStage(STAGE_BW_REST);
        } else {
          value.setStage(STAGE_TRIMMING);
        }
        break;
      }
      value.setActive(false);
      value.setUpdated(false);
      return false;
    }
  }
  public static class SCCVertexReader extends
  GraphLoader<LongWritable, MyValue, NullWritable, LongWritable> {
    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, MyValue, NullWritable,
LongWritable> context)
    throws IOException {
      SCCVertex vertex = new SCCVertex();
      vertex.setId((LongWritable) record.get(0));
      String[] edges = record.get(1).toString().split(",");
      for (int i = 0; i < edges.length; i++) {
        try {
          long destID = Long.parseLong(edges[i]);
          vertex.addEdge(new LongWritable(destID), NullWritable.get
());
        } catch(NumberFormatException nfe) {
          System.err.println("Ignore " + nfe);
        }
      }
      context.addVertexRequest(vertex);
    }
  }
  public static void main(String[] args) throws IOException {
    if (args.length < 2) {
      System.out.println("Usage: <input> <output>");
      System.exit(-1);
    }
    GraphJob job = new GraphJob();
    job.setGraphLoaderClass(SCCVertexReader.class);
    job.setVertexClass(SCCVertex.class);
    job.setAggregatorClass(SCCAggregator.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    long startTime = System.currentTimeMillis();
    job.run();
    System.out.println("Job Finished in "
        + (System.currentTimeMillis() - startTime) / 1000.0 + "
seconds");
  }
```

```
}
```

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。

代码示例

连通分量的代码，如下所示：

```
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");
  }
}
```

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。

算法步骤如下：

1. 从图中找到一个没有入边的顶点，并输出。
2. 从图中删除该点，及其所有出边。
3. 重复以上步骤，直到所有点都已输出。

代码示例

拓扑排序算法的代码，如下所示：

```
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, LongWritab
le> {
    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, LongWritable, NullWritable,
LongWritable> context)
        throws IOException {
      TopologySortVertex vertex = new TopologySortVertex();
      vertex.setId((LongWritable) record.get(0));
      vertex.setValue(new LongWritable(0));
      String[] edges = record.get(1).toString().split(",");
      for (int i = 0; i < edges.length; i++) {
        long edge = Long.parseLong(edges[i]);
        if (edge >= 0) {
          vertex.addEdge(new LongWritable(Long.parseLong(edges[i])),
              NullWritable.get());
```

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

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，流程如下：

1. 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）。
2. 对每个样本，计算最小平方误差。
3. 对最小平方误差求和，根据下降速率更新权重。
4. 重复迭代直到收敛。

代码示例

```
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,......
 **/
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, NullWritab
le> context)
        throws IOException {
      LinearRegressionVertex vertex = new LinearRegressionVertex();
      vertex.setId(recordNum);
      vertex.setValue(new Tuple(record.getAll()));
      context.addVertexRequest(vertex);
```

```
    }
  }
  public static class LinearRegressionAggregator extends
      Aggregator<GradientWritable> {
    @SuppressWarnings("rawtypes")
    @Override
    public GradientWritable createInitialValue(WorkerContext context)
        throws IOException {
      if (context.getSuperstep() == 0) {
        /* set initial value, all 0 */
        GradientWritable grad = new GradientWritable();
        grad.lastTheta = new Tuple();
        grad.currentTheta = new Tuple();
        grad.tmpGradient = new Tuple();
        grad.count = new LongWritable(1);
        grad.lost = new DoubleWritable(0.0);
        int n = (int) Long.parseLong(context.getConfiguration()
            .get("Dimension"));
        for (int i = 0; i < n; i++) {
          grad.lastTheta.append(new DoubleWritable(0));
          grad.currentTheta.append(new DoubleWritable(0));
          grad.tmpGradient.append(new DoubleWritable(0));
        }
        return grad;
      } else
        return (GradientWritable) context.getLastAggregatedValue(0);
    }
    public static double vecMul(Tuple value, Tuple theta) {
      /* perform this partial computing: y(i)-hθ(x(i)) for each sample
 */
      /* value denote a piece of sample and value(0) is y */
      double sum = 0.0;
      for (int j = 1; j < value.size(); j++)
        sum += Double.parseDouble(value.get(j).toString())
            * Double.parseDouble(theta.get(j).toString());
      Double tmp = Double.parseDouble(theta.get(0).toString()) + sum
          - Double.parseDouble(value.get(0).toString());
      return tmp;
    }
    @Override
    public void aggregate(GradientWritable gradient, Object value)
        throws IOException {
      /*
       * perform on each vertex--each sample i：set theta(j) for each
sample i
       * for each dimension
       */
      double tmpVar = vecMul((Tuple) value, gradient.currentTheta);
      /*
       * update 2:local worker aggregate(), perform like merge() below
. This
       * means the variable gradient denotes the previous aggregated
value
       */
      gradient.tmpGradient.set(0, new DoubleWritable(
          ((DoubleWritable) gradient.tmpGradient.get(0)).get() +
tmpVar));
      gradient.lost.set(Math.pow(tmpVar, 2));
      /*
       * calculate (y(i)-hθ(x(i))) x(i)(j) for each sample i for each
       * dimension j
       */
```

```
      for (int j = 1; j < gradient.tmpGradient.size(); j++)
        gradient.tmpGradient.set(j, new DoubleWritable(
            ((DoubleWritable) gradient.tmpGradient.get(j)).get() +
tmpVar
                * Double.parseDouble(((Tuple) value).get(j).toString
()())));
    }
    @Override
    public void merge(GradientWritable gradient, GradientWritable
partial)
        throws IOException {
      /* perform SumAll on each dimension for all samples. */
      Tuple master = (Tuple) gradient.tmpGradient;
      Tuple part = (Tuple) partial.tmpGradient;
      for (int j = 0; j < gradient.tmpGradient.size(); j++) {
        DoubleWritable s = (DoubleWritable) master.get(j);
        s.set(s.get() + ((DoubleWritable) part.get(j)).get());
      }
      gradient.lost.set(gradient.lost.get() + partial.lost.get());
    }
    @SuppressWarnings("rawtypes")
    @Override
    public boolean terminate(WorkerContext context, GradientWritable
gradient)
        throws IOException {
      /*
       * 1. calculate new theta 2. judge the diff between last step
and this
       * step, if smaller than the threshold, stop iteration
       */
      gradient.lost = new DoubleWritable(gradient.lost.get()
          / (2 * context.getTotalNumVertices()));
      /*
       * we can calculate lost in order to make sure the algorithm is
running on
       * the right direction (for debug)
       */
      System.out.println(gradient.count + " lost:" + gradient.lost);
      Tuple tmpGradient = gradient.tmpGradient;
      System.out.println("tmpGra" + tmpGradient);
      Tuple lastTheta = gradient.lastTheta;
      Tuple tmpCurrentTheta = new Tuple(gradient.currentTheta.size());
      System.out.println(gradient.count + " terminate_start_last:" +
lastTheta);
      double alpha = 0.07; // learning rate
      // alpha =
      // Double.parseDouble(context.getConfiguration().get("Alpha"));
      /* perform theta(j) = theta(j)-alpha*tmpGradient */
      long M = context.getTotalNumVertices();
      /*
       * update 3: add (/M) on the code. The original code forget this
 step
       */
      for (int j = 0; j < lastTheta.size(); j++) {
        tmpCurrentTheta
            .set(
                j,
                new DoubleWritable(Double.parseDouble(lastTheta.get(j)
                    .toString())
                    - alpha
                    / M
```

```
                    * Double.parseDouble(tmpGradient.get(j).toString
()));
      }
      System.out.println(gradient.count + " terminate_start_current:"
          + tmpCurrentTheta);
      // judge if convergence is happening.
      double diff = 0.00d;
      for (int j = 0; j < gradient.currentTheta.size(); j++)
        diff += Math.pow(((DoubleWritable) tmpCurrentTheta.get(j)).get
()
            - ((DoubleWritable) lastTheta.get(j)).get(), 2);
      if (/*
           * Math.sqrt(diff) < 0.00000000005d ||
           */Long.parseLong(context.getConfiguration().get("Max_Iter_N
um")) == 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");
  }
```

```
}
```

8.6.9 三角形计数

三角形计数算法用于计算通过每个顶点的三角形个数。

算法实现的流程如下：

1. 每个顶点将其 ID 发送给所有出边邻居。
2. 存储入边和出边邻居并发送给出边邻居。
3. 对每条边计算其终点的交集数量，并求和，结果输出到表。
4. 将表中的输出结果求和并除以三，即得到三角形个数。

代码示例

三角形计数算法的代码，如下所示：

```
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");
  }
}
```

8.6.10 输入点表

输入点表的代码，如下所示：

```
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。
 *
 * ODPS Graph的作业输入都为ODPS的Table，假设作业输入有两张表，一张存储点的信
息，一张存储边的信息。
 * 存储点信息的表的格式，如：
 * +------------------------+
 * | VertexID | VertexValue |
 * +------------------------+
 * |       id0|            9|
```

```
 * +------------------------+
 * |       id1|             7|
 * +------------------------+
 * |       id2|             8|
 * +------------------------+
 *
 * 存储边信息的表的格式，如
 * +-----------------------------------+
 * | VertexID | DestVertexID| EdgeValue|
 * +-----------------------------------+
 * |       id0|          id1|         1|
 * +-----------------------------------+
 * |       id0|          id2|         2|
 * +-----------------------------------+
 * |       id2|          id1|         3|
 * +-----------------------------------+
 *
 * 结合两张表的数据，表示id0有两条出边，分别指向id1和id2；id2有一条出边，指向id1
；id1没有出边。
 *
 * 对于此种类型的数据，在GraphLoader::load(LongWritable, Record,
MutationContext)
 * ，可以使用 MutationContext#addVertexRequest(Vertex)向图中请求添加点，使
用
 * link MutationContext#addEdgeRequest(WritableComparable, Edge)向图中
请求添加边，然后，在
 * link VertexResolver#resolve(WritableComparable, Vertex, VertexChan
ges, 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。
   *
   * 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LoadingVer
texResolver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, LongWritab
le> {
    private boolean isEdgeData;
    /**
     * 配置VertexInputLoader。
     *
     * @param conf
     *          作业的配置参数，在main中使用GraphJob配置的，或者在console中
set的
     * @param workerId
```

```
     *          当前工作的worker的序号，从0开始，可以用于构造唯一的vertex id
     * @param inputTableInfo
     *          当前worker载入的输入表信息，可以用于确定当前输入是哪种类型的数
据，即Record的格式
     **/
    @Override
    public void setup(Configuration conf, int workerId, TableInfo
inputTableInfo) {
      isEdgeData = conf.get(EDGE_TABLE).equals(inputTableInfo.
getTableName());
    }
    /**
     * 根据Record中的内容，解析为对应的边，并请求添加到图中。
     *
     * @param recordNum
     *          记录序列号，从1开始，每个worker上单独计数
     * @param record
     *          输入表中的记录，三列，分别表示初点、终点、边的权重
     * @param context
     *          上下文，请求将解释后的边添加到图中
     **/
    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, LongWritable, LongWritable,
LongWritable> context)
        throws IOException {
      if (isEdgeData) {
        /**
         * 数据来源于存储边信息的表。
         *
         * 1、第一列表示初始点的ID
         **/
        LongWritable sourceVertexID = (LongWritable) record.get(0);
        /**
         * 2、第二列表示终点的ID
         **/
        LongWritable destinationVertexID = (LongWritable) record.get(1
);
        /**
         * 3、地三列表示边的权重
         **/
        LongWritable edgeValue = (LongWritable) record.get(2);
        /**
         * 4、创建边，由终点ID和边的权重组成
         **/
        Edge<LongWritable, LongWritable> edge = new Edge<LongWritable
, LongWritable>(
            destinationVertexID, edgeValue);
        /**
         * 5、请求给初始点添加边
         **/
        context.addEdgeRequest(sourceVertexID, edge);
        /**
         * 6、如果每条Record表示双向边，重复4与5 Edge<LongWritable,
LongWritable> edge2 = new
         * Edge<LongWritable, LongWritable>( sourceVertexID, edgeValue
);
```

```
           * context.addEdgeRequest(destinationVertexID, edge2);
           **/
        } else {
          /**
           * 数据来源于存储点信息的表。
           *
           * 1、第一列表示点的ID
           **/
          LongWritable vertexID = (LongWritable) record.get(0);
          /**
           * 2、第二列表示点的值
           **/
          LongWritable vertexValue = (LongWritable) record.get(1);
          /**
           * 3、创建点，由点的ID和点的值组成
           **/
          MyVertex vertex = new MyVertex();
          /**
           * 4、初始化点
           **/
          vertex.setId(vertexID);
          vertex.setValue(vertexValue);
          /**
           * 5、请求添加点
           **/
          context.addVertexRequest(vertex);
        }
      }
    }
    /**
     * 汇总GraphLoader::load(LongWritable, Record, MutationContext)生成的
键值对，类似于
     * com.aliyun.odps.mapreduce.Reducer中的reduce。对于唯一的Vertex ID，所
有关于这个ID上
     * 添加\删除、点\边的行为都会放在VertexChanges中。
     *
     * 注意：此处并不只针对load方法中添加的有冲突的点或边才调用（冲突是指添加多个相
同Vertex对象，添加重复边等），
     * 所有在load方法中请求生成的ID都会在此处被调用。
     **/
    public static class LoadingResolver extends
        VertexResolver<LongWritable, LongWritable, LongWritable,
LongWritable> {
      /**
       * 处理关于一个ID的添加或删除、点或边的请求。
       *
       * VertexChanges有四个接口，分别与MutationContext的四个接口对应：
       * VertexChanges::getAddedVertexList()与
       * MutationContext::addVertexRequest(Vertex)对应，
       * 在load方法中，请求添加的ID相同的Vertex对象，会被汇总在返回的List中
       * VertexChanges::getAddedEdgeList()与
       * MutationContext::addEdgeRequest(WritableComparable, Edge)
       * 对应，请求添加的初始点ID相同的Edge对象，会被汇总在返回的List中
       * VertexChanges::getRemovedVertexCount()与
       * MutationContext::removeVertexRequest(WritableComparable)
       * 对应，请求删除的ID相同的Vertex，汇总的请求删除的次数作为返回值
       * VertexChanges#getRemovedEdgeList()与
```

```
     * MutationContext#removeEdgeRequest(WritableComparable,
WritableComparable)
     * 对应，请求删除的初始点ID相同的Edge对象，会被汇总在返回的List中
     *
     * 用户通过处理关于这个ID的变化，通过返回值声明此ID是否参与计算，如果返回的
Vertex不为null，
     * 则此ID会参与随后的计算，如果返回null，则不会参与计算。
     *
     * @param vertexId
     *          请求添加的点的ID，或请求添加的边的初点ID
     * @param vertex
     *          已存在的Vertex对象，数据载入阶段，始终为null
     * @param vertexChanges
     *          此ID上的请求添加\删除、点\边的集合
     * @param hasMessages
     *          此ID是否有输入消息，数据载入阶段，始终为false
     **/
    @Override
    public Vertex<LongWritable, LongWritable, LongWritable, LongWritab
le> resolve(
        LongWritable vertexId,
        Vertex<LongWritable, LongWritable, LongWritable, LongWritable
> vertex,
        VertexChanges<LongWritable, LongWritable, LongWritable,
LongWritable> vertexChanges,
        boolean hasMessages) throws IOException {
      /**
       * 1、获取Vertex对象，作为参与计算的点。
       **/
      MyVertex computeVertex = null;
      if (vertexChanges.getAddedVertexList() == null
          || vertexChanges.getAddedVertexList().isEmpty()) {
        computeVertex = new MyVertex();
        computeVertex.setId(vertexId);
      } else {
        /**
         * 此处假设存储点信息的表中，每个Record表示唯一的点。
         **/
        computeVertex = (MyVertex) vertexChanges.getAddedVertexList().
get(0);
      }
      /**
       * 2、将请求给此点添加的边，添加到Vertex对象中，如果数据有重复的可能，根据
算法需要决定是否去重。
       **/
      if (vertexChanges.getAddedEdgeList() != null) {
        for (Edge<LongWritable, LongWritable> edge : vertexChanges
            .getAddedEdgeList()) {
          computeVertex.addEdge(edge.getDestVertexId(), edge.getValue
());
        }
      }
      /**
       * 3、将Vertex对象返回，添加到最终的图中参与计算。
       **/
      return computeVertex;
    }
  }
  /**
```

```
   * 确定参与计算的Vertex的行为。
   *
   **/
  public static class MyVertex extends
      Vertex<LongWritable, LongWritable, LongWritable, LongWritable> {
    /**
     * 将vertex的边，按照输入表的格式再写到结果表。输入表与输出表的格式和数据都相
同。
     *
     * @param context
     *          运行时上下文
     * @param messages
     *          输入消息
     **/
    @Override
    public void compute(
        ComputeContext<LongWritable, LongWritable, LongWritable,
LongWritable> context,
        Iterable<LongWritable> messages) throws IOException {
      /**
       * 将点的ID和值，写到存储点的结果表
       **/
      context.write("vertex", getId(), getValue());
      /**
       * 将点的边，写到存储边的结果表
       **/
      if (hasEdges()) {
        for (Edge<LongWritable, LongWritable> edge : getEdges()) {
          context.write("edge", getId(), edge.getDestVertexId(),
              edge.getValue());
        }
      }
      /**
       * 只迭代一轮
       **/
      voteToHalt();
    }
  }
  /**
   * @param args
   * @throws IOException
   */
  public static void main(String[] args) throws IOException {
    if (args.length < 4) {
      throw new IOException(
          "Usage: VertexInputFormat <vertex input> <edge input> <
vertex output> <edge output>");
    }
    /**
     * GraphJob用于对Graph作业进行配置
     */
    GraphJob job = new GraphJob();
    /**
     * 1、指定输入的图数据，并指定存储边数据所在的表。
     */
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addInput(TableInfo.builder().tableName(args[1]).build());
    job.set(EDGE_TABLE, args[1]);
    /**
```

```
     * 2、指定载入数据的方式，将Record解释为Edge，此处类似于map，生成的 key为
vertex的ID，value为edge。
     */
    job.setGraphLoaderClass(VertexInputLoader.class);
    /**
     * 3、指定载入数据阶段，生成参与计算的vertex。此处类似于reduce，将map 生成
的edge合并成一个vertex。
     */
    job.setLoadingVertexResolverClass(LoadingResolver.class);
    /**
     * 4、指定参与计算的vertex的行为。每轮迭代执行vertex.compute方法。
     */
    job.setVertexClass(MyVertex.class);
    /**
     * 5、指定图作业的输出表，将计算生成的结果写到结果表中。
     */
    job.addOutput(TableInfo.builder().tableName(args[2]).label("vertex
").build());
    job.addOutput(TableInfo.builder().tableName(args[3]).label("edge
").build());
    /**
     * 6、提交作业执行。
     */
    job.run();
  }
}
```

8.6.11 输入边表

输入边表的代码，如下所示：

```
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。
 *
 * ODPS Graph的作业输入都为ODPS的Table，假设作业输入有两张表，一张存储点的信
息，一张存储边的信息。
 * 存储点信息的表的格式，如：
 * +------------------------+
 * | VertexID | VertexValue |
 * +------------------------+
 * |       id0|            9|
```

```
 * +------------------------+
 * |       id1|             7|
 * +------------------------+
 * |       id2|             8|
 * +------------------------+
 *
 * 存储边信息的表的格式，如
 * +-----------------------------------+
 * | VertexID | DestVertexID| EdgeValue|
 * +-----------------------------------+
 * |       id0|          id1|         1|
 * +-----------------------------------+
 * |       id0|          id2|         2|
 * +-----------------------------------+
 * |       id2|          id1|         3|
 * +-----------------------------------+
 *
 * 结合两张表的数据，表示id0有两条出边，分别指向id1和id2；id2有一条出边，指向id1
；id1没有出边。
 *
 * 对于此种类型的数据，在GraphLoader::load(LongWritable, Record,
MutationContext)
 * ，可以使用 MutationContext#addVertexRequest(Vertex)向图中请求添加点，使
用
 * link MutationContext#addEdgeRequest(WritableComparable, Edge)向图中
请求添加边，然后，在
 * link VertexResolver#resolve(WritableComparable, Vertex, VertexChan
ges, 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LoadingVer
texResolver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, LongWritab
le> {
    private boolean isEdgeData;
    /**
     * 配置VertexInputLoader。
     *
     * @param conf
     *          作业的配置参数，在main中使用GraphJob配置的，或者在console中
set的
     * @param workerId
```

```
     *          当前工作的worker的序号，从0开始，可以用于构造唯一的vertex id
     * @param inputTableInfo
     *          当前worker载入的输入表信息，可以用于确定当前输入是哪种类型的数
据，即Record的格式
     **/
    @Override
    public void setup(Configuration conf, int workerId, TableInfo
inputTableInfo) {
      isEdgeData = conf.get(EDGE_TABLE).equals(inputTableInfo.
getTableName());
    }
    /**
     * 根据Record中的内容，解析为对应的边，并请求添加到图中。
     *
     * @param recordNum
     *          记录序列号，从1开始，每个worker上单独计数
     * @param record
     *          输入表中的记录，三列，分别表示初点、终点、边的权重
     * @param context
     *          上下文，请求将解释后的边添加到图中
     **/
    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, LongWritable, LongWritable,
LongWritable> context)
        throws IOException {
      if (isEdgeData) {
        /**
         * 数据来源于存储边信息的表。
         *
         * 1、第一列表示初始点的ID
         **/
        LongWritable sourceVertexID = (LongWritable) record.get(0);
        /**
         * 2、第二列表示终点的ID
         **/
        LongWritable destinationVertexID = (LongWritable) record.get(1
);
        /**
         * 3、地三列表示边的权重
         **/
        LongWritable edgeValue = (LongWritable) record.get(2);
        /**
         * 4、创建边，由终点ID和边的权重组成
         **/
        Edge<LongWritable, LongWritable> edge = new Edge<LongWritable
, LongWritable>(
            destinationVertexID, edgeValue);
        /**
         * 5、请求给初始点添加边
         **/
        context.addEdgeRequest(sourceVertexID, edge);
        /**
         * 6、如果每条Record表示双向边，重复4与5 Edge<LongWritable,
LongWritable> edge2 = new
         * Edge<LongWritable, LongWritable>( sourceVertexID, edgeValue
);
```

```
           * context.addEdgeRequest(destinationVertexID, edge2);
           **/
        } else {
          /**
           * 数据来源于存储点信息的表。
           *
           * 1、第一列表示点的ID
           **/
          LongWritable vertexID = (LongWritable) record.get(0);
          /**
           * 2、第二列表示点的值
           **/
          LongWritable vertexValue = (LongWritable) record.get(1);
          /**
           * 3、创建点，由点的ID和点的值组成
           **/
          MyVertex vertex = new MyVertex();
          /**
           * 4、初始化点
           **/
          vertex.setId(vertexID);
          vertex.setValue(vertexValue);
          /**
           * 5、请求添加点
           **/
          context.addVertexRequest(vertex);
        }
      }
    }
    /**
     * 汇总GraphLoader::load(LongWritable, Record, MutationContext)生成的
键值对，类似于
     * com.aliyun.odps.mapreduce.Reducer中的reduce。对于唯一的Vertex ID，所
有关于这个ID上
     * 添加\删除、点\边的行为都会放在VertexChanges中。
     *
     * 注意：此处并不只针对load方法中添加的有冲突的点或边才调用（冲突是指添加多个相
同Vertex对象，添加重复边等），
     * 所有在load方法中请求生成的ID都会在此处被调用。
     **/
    public static class LoadingResolver extends
        VertexResolver<LongWritable, LongWritable, LongWritable,
LongWritable> {
      /**
       * 处理关于一个ID的添加或删除、点或边的请求。
       *
       * VertexChanges有四个接口，分别与MutationContext的四个接口对应：
       * VertexChanges::getAddedVertexList()与
       * MutationContext::addVertexRequest(Vertex)对应，
       * 在load方法中，请求添加的ID相同的Vertex对象，会被汇总在返回的List中
       * VertexChanges::getAddedEdgeList()与
       * MutationContext::addEdgeRequest(WritableComparable, Edge)
       * 对应，请求添加的初始点ID相同的Edge对象，会被汇总在返回的List中
       * VertexChanges::getRemovedVertexCount()与
       * MutationContext::removeVertexRequest(WritableComparable)
       * 对应，请求删除的ID相同的Vertex，汇总的请求删除的次数作为返回值
       * VertexChanges#getRemovedEdgeList()与
```

```
     * MutationContext#removeEdgeRequest(WritableComparable,
WritableComparable)
     * 对应，请求删除的初始点ID相同的Edge对象，会被汇总在返回的List中
     *
     * 用户通过处理关于这个ID的变化，通过返回值声明此ID是否参与计算，如果返回的
Vertex不为null，
     * 则此ID会参与随后的计算，如果返回null，则不会参与计算。
     *
     * @param vertexId
     *          请求添加的点的ID，或请求添加的边的初点ID
     * @param vertex
     *          已存在的Vertex对象，数据载入阶段，始终为null
     * @param vertexChanges
     *          此ID上的请求添加\删除、点\边的集合
     * @param hasMessages
     *          此ID是否有输入消息，数据载入阶段，始终为false
     **/
    @Override
    public Vertex<LongWritable, LongWritable, LongWritable, LongWritab
le> resolve(
        LongWritable vertexId,
        Vertex<LongWritable, LongWritable, LongWritable, LongWritable
> vertex,
        VertexChanges<LongWritable, LongWritable, LongWritable,
LongWritable> vertexChanges,
        boolean hasMessages) throws IOException {
      /**
       * 1、获取Vertex对象，作为参与计算的点。
       **/
      MyVertex computeVertex = null;
      if (vertexChanges.getAddedVertexList() == null
          || vertexChanges.getAddedVertexList().isEmpty()) {
        computeVertex = new MyVertex();
        computeVertex.setId(vertexId);
      } else {
        /**
         * 此处假设存储点信息的表中，每个Record表示唯一的点。
         **/
        computeVertex = (MyVertex) vertexChanges.getAddedVertexList().
get(0);
      }
      /**
       * 2、将请求给此点添加的边，添加到Vertex对象中，如果数据有重复的可能，根据
算法需要决定是否去重。
       **/
      if (vertexChanges.getAddedEdgeList() != null) {
        for (Edge<LongWritable, LongWritable> edge : vertexChanges
            .getAddedEdgeList()) {
          computeVertex.addEdge(edge.getDestVertexId(), edge.getValue
());
        }
      }
      /**
       * 3、将Vertex对象返回，添加到最终的图中参与计算。
       **/
      return computeVertex;
    }
  }
  /**
```

```
   * 确定参与计算的Vertex的行为。
   *
   **/
  public static class MyVertex extends
      Vertex<LongWritable, LongWritable, LongWritable, LongWritable> {
    /**
     * 将vertex的边，按照输入表的格式再写到结果表。输入表与输出表的格式和数据都相
同。
     *
     * @param context
     *          运行时上下文
     * @param messages
     *          输入消息
     **/
    @Override
    public void compute(
        ComputeContext<LongWritable, LongWritable, LongWritable,
LongWritable> context,
        Iterable<LongWritable> messages) throws IOException {
      /**
       * 将点的ID和值，写到存储点的结果表
       **/
      context.write("vertex", getId(), getValue());
      /**
       * 将点的边，写到存储边的结果表
       **/
      if (hasEdges()) {
        for (Edge<LongWritable, LongWritable> edge : getEdges()) {
          context.write("edge", getId(), edge.getDestVertexId(),
              edge.getValue());
        }
      }
      /**
       * 只迭代一轮
       **/
      voteToHalt();
    }
  }
  /**
   * @param args
   * @throws IOException
   */
  public static void main(String[] args) throws IOException {
    if (args.length < 4) {
      throw new IOException(
          "Usage: VertexInputFormat <vertex input> <edge input> <
vertex output> <edge output>");
    }
    /**
     * GraphJob用于对Graph作业进行配置
     */
    GraphJob job = new GraphJob();
    /**
     * 1、指定输入的图数据，并指定存储边数据所在的表。
     */
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addInput(TableInfo.builder().tableName(args[1]).build());
    job.set(EDGE_TABLE, args[1]);
    /**
```

```
     * 2、指定载入数据的方式，将Record解释为Edge，此处类似于map，生成的 key为
vertex的ID，value为edge。
     */
    job.setGraphLoaderClass(VertexInputLoader.class);
    /**
     * 3、指定载入数据阶段，生成参与计算的vertex。此处类似于reduce，将map 生成
的edge合并成一个vertex。
     */
    job.setLoadingVertexResolverClass(LoadingResolver.class);
    /**
     * 4、指定参与计算的vertex的行为。每轮迭代执行vertex.compute方法。
     */
    job.setVertexClass(MyVertex.class);
    /**
     * 5、指定图作业的输出表，将计算生成的结果写到结果表中。
     */
    job.addOutput(TableInfo.builder().tableName(args[2]).label("vertex
").build());
    job.addOutput(TableInfo.builder().tableName(args[3]).label("edge
").build());
    /**
     * 6、提交作业执行。
     */
    job.run();
  }
}
```

8.7 Aggregator机制概述

本文将为您介绍 Aggregator 的执行机制、相关 API，并以 Kmeans Clustering 为例，说明 Aggregator 的具体用法。

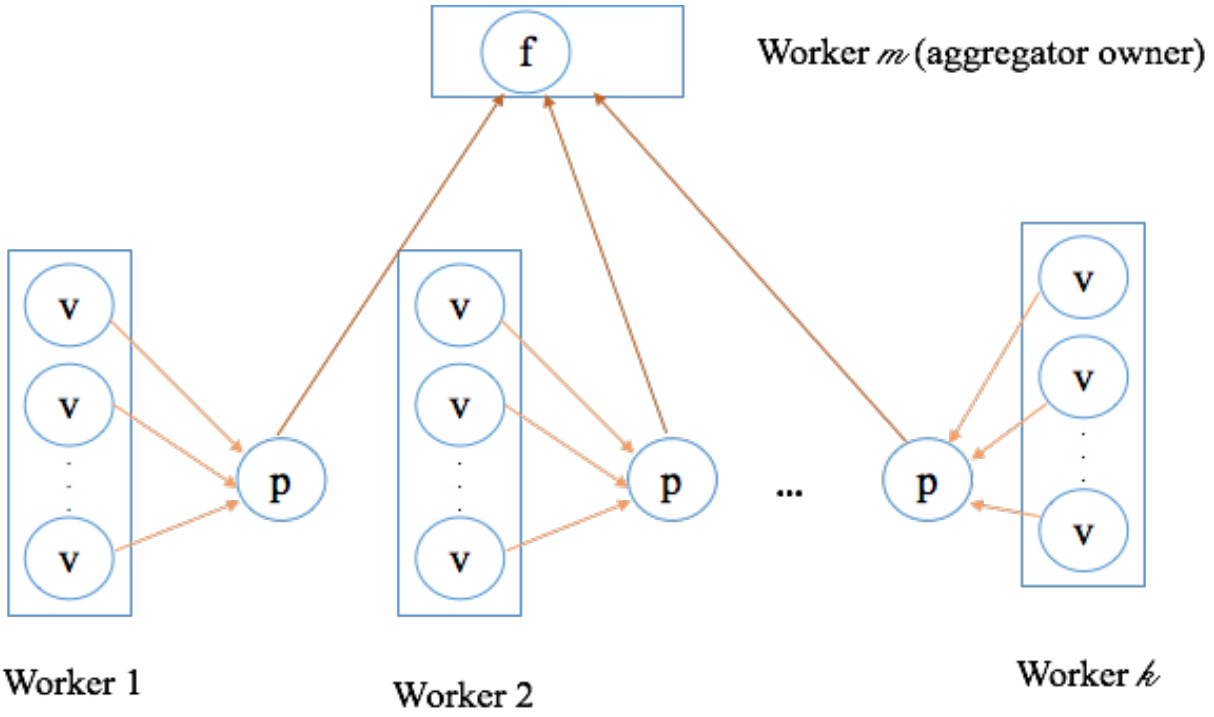
Aggregator 是 MaxCompute Graph 作业中常用的 feature 之一，特别适用于解决机器学习问题。MaxCompute Graph 中，Aggregator 用于汇总并处理全局信息。

Aggregator 机制

Aggregator 的逻辑分为两部分：

- 一部分在所有 Worker 上执行，即分布式执行。
- 另一部分只在 AggregatorOwner 所在的 Worker 上执行，即单点。

其中在所有 Worker 上执行的操作包括创建初始值及局部聚合，然后将局部聚合结果发送给 AggregatorOwner 所在的 Worker 上。AggregatorOwner 所在的 Worker 上聚合普通 Worker 发送过来的局部聚合对象，得到全局聚合结果，然后判断迭代是否结束。全局聚合的结果会在下一轮超步分发给所有 Worker，供下一轮迭代使用。如下图所示：



Aggregator 的 API

Aggregator 共提供了五个 API 供您实现。API 的调用时机及常规用途，如下所示：

- createStartupValue(context)

  该 API 在所有 Worker 上执行一次，调用时机是所有超步开始之前，通常用以初始化 Aggregator Value。在第 0 轮超步中，调用 WorkerContext.getLastAggregatedValue() 或 ComputeContext.getLastAggregatedValue() 可以获取该 API 初始化的 AggregatorValue 对象。
- createInitialValue(context)

  该 API 在所有 Worker 上每轮超步开始时调用一次，用以初始化本轮迭代所用的 Aggregator Value。通常操作是通过 WorkerContext.getLastAggregatedValue() 得到上一轮迭代的结果，然后执行部分初始化操作。
- aggregate(value, item)

  该 API 同样在所有 Worker 上执行，与上述 API 不同的是，该 API 由用户显示调用 ComputeContext#aggregate(item) 来触发，而上述两个 API，则由框架自动调用。该 API 用以执行局部聚合操作，其中第一个参数 value 是本 Worker 在该轮超步已经聚合的结果（初始值是 createInitialValue 返回的对象），第二个参数是您的代码调用 ComputeContext#aggregate(item) 传入的参数。该 API 中通常用 item 来更新 value 实现聚合。所有 aggregate 执行完后，得到

的 value 就是该 Worker 的局部聚合结果，然后由框架发送给 AggregatorOwner 所在的 Worker 。

- merge(value, partial)

  该 API 执行于 AggregatorOwner 所在 Worker，用以合并各 Worker 局部聚合的结果，达到全局聚合对象。与 Aggregate 类似，value 是已经聚合的结果，而 partial 待聚合的对象，同样用 partial 更新 value。

  假设有 3 个 worker，分别是 w0、w1、w2，其局部聚合结果是 p0、p1、p2。假设发送到 AggregatorOwner 所在 Worker 的顺序为 p1、p0、p2。那么 merge 执行次序为：

  1. 首先执行 merge(p1, p0)，这样 p1 和 p0 就聚合为 p1。
  2. 然后执行 merge(p1', p2)，p1 和 p2 聚合为 p1，而 p1 即为本轮超步全局聚合的结果。

  由上述示例可见，当只有一个 worker 时，不需要执行 merge 方法，也就是说 merge() 不会被调用。

- terminate(context, value)

  当 AggregatorOwner 所在 Worker 执行完 merge() 后，框架会调用 terminate(context, value) 执行最后的处理。其中第二个参数 value，即为 merge() 最后得到全局聚合，在该方法中可以对全局聚合继续修改。执行完 terminate() 后，框架会将全局聚合对象分发给所有 Worker，供下一轮超步使用。terminate() 方法的一个特殊之处在于，如果返回 true，则整个作业就结束迭代，否则继续执行。在机器学习场景中，通常判断收敛后返回 true 以结束作业。

Kmeans Clustering 示例

下面以典型的 KmeansClustering 为例，为您介绍 Aggregator 的具体用法。

说明：

完整代码见附件，此处为解析代码。

- **GraphLoader 部分**

  GraphLoader 部分用以加载输入表，并转换为图的点或边。这里我们输入表的每行数据为一个样本，一个样本构造一个点，并用 Vertex 的 value 来存放样本。

  首先定义一个 Writable 类 KmeansValue 作为 Vertex 的 value 类型：

```
public static class KmeansValue implements Writable {
  DenseVector sample;
  public KmeansValue() {
  }
```

```
  public KmeansValue(DenseVector v) {
    this.sample = v;
  }
  @Override
  public void write(DataOutput out) throws IOException {
    wirteForDenseVector(out, sample);
  }
  @Override
  public void readFields(DataInput in) throws IOException {
    sample = readFieldsForDenseVector(in);
  }
}
```

KmeansValue 中封装一个 DenseVector 对象来存放一个样本，这里 DenseVector 类型来自 *matrix-toolkits-java*，而 wirteForDenseVector() 及 readFieldsForDenseVector() 用以实现序列化及反序列化，可参见附件中的完整代码。

自定义的 KmeansReader 代码，如下所示：

```
public static class KmeansReader extends
  GraphLoader<LongWritable, KmeansValue, NullWritable, NullWritable>
 {
  @Override
  public void load(
      LongWritable recordNum,
      WritableRecord record,
      MutationContext<LongWritable, KmeansValue, NullWritable,
NullWritable> context)
      throws IOException {
    KmeansVertex v = new KmeansVertex();
    v.setId(recordNum);
    int n = record.size();
    DenseVector dv = new DenseVector(n);
    for (int i = 0; i < n; i++) {
      dv.set(i, ((DoubleWritable)record.get(i)).get());
    }
    v.setValue(new KmeansValue(dv));
    context.addVertexRequest(v);
  }
}
```

KmeansReader 中，每读入一行数据（一个 Record）创建一个点，这里用 recordNum 作为点的 ID，将 record 内容转换成 DenseVector 对象并封装进 VertexValue 中。

- **Vertex 部分**

  自定义的 KmeansVertex 代码如下。逻辑非常简单，每轮迭代要做的事情就是将自己维护的样本执行局部聚合。具体逻辑参见下面 Aggregator 的实现。

```
public static class KmeansVertex extends
  Vertex<LongWritable, KmeansValue, NullWritable, NullWritable> {
  @Override
  public void compute(
      ComputeContext<LongWritable, KmeansValue, NullWritable,
NullWritable> context,
```

```
      Iterable<NullWritable> messages) throws IOException {
    context.aggregate(getValue());
  }
}
```

- **Aggregator 部分**

  整个 Kmeans 的主要逻辑集中在 Aggregator 中。首先是自定义的 KmeansAggrValue，用以维护要聚合及分发的内容。

```
public static class KmeansAggrValue implements Writable {
  DenseMatrix centroids;
  DenseMatrix sums; // used to recalculate new centroids
  DenseVector counts; // used to recalculate new centroids
  @Override
  public void write(DataOutput out) throws IOException {
    wirteForDenseDenseMatrix(out, centroids);
    wirteForDenseDenseMatrix(out, sums);
    wirteForDenseVector(out, counts);
  }
  @Override
  public void readFields(DataInput in) throws IOException {
    centroids = readFieldsForDenseMatrix(in);
    sums = readFieldsForDenseMatrix(in);
    counts = readFieldsForDenseVector(in);
  }
}
```

  KmeansAggrValue 中维护了三个对象，其中 centroids 是当前的 K 个中心点，假定样本是 m 维的话，centroids 就是一个 K*m 的矩阵。sums 是和 centroids 大小一样的矩阵，每个元素记录了到特定中心点最近的样本特定维之和，例如 sums(i,j) 是到第 i 个中心点最近的样本的第 j 维度之和。

  counts 是个 K 维的向量，记录到每个中心点距离最短的样本个数。sums 和 counts 一起用以计算新的中心点，也是要聚合的主要内容。

  接下来是自定义的 Aggregator 实现类 KmeansAggregator，我们按照上述 API 的顺序逐个分析其实现。

  1. createStartupValue() 的实现。

```
public static class KmeansAggregator extends Aggregator<KmeansAggr
Value> {
public KmeansAggrValue createStartupValue(WorkerContext context)
throws IOException {
KmeansAggrValue av = new KmeansAggrValue();
byte[] centers = context.readCacheFile("centers");
String lines[] = new String(centers).split("\n");
int rows = lines.length;
int cols = lines[0].split(",").length; // assumption rows >= 1
av.centroids = new DenseMatrix(rows, cols);
av.sums = new DenseMatrix(rows, cols);
av.sums.zero();
```

```
av.counts = new DenseVector(rows);
av.counts.zero();
for (int i = 0; i < lines.length; i++) {
  String[] ss = lines[i].split(",");
  for (int j = 0; j < ss.length; j++) {
    av.centroids.set(i, j, Double.valueOf(ss[j]));
  }
}
return av;
}
```

我们在该方法中初始化一个 KmeansAggrValue 对象，然后从资源文件 centers 中读取初始中心点，并赋值给 centroids。而 sums 和 counts 初始化为 0。

2. createInitialValue() 的实现。

```
@Override
public void aggregate(KmeansAggrValue value, Object item)
  throws IOException {
DenseVector sample = ((KmeansValue)item).sample;
// find the nearest centroid
int min = findNearestCentroid(value.centroids, sample);
// update sum and count
for (int i = 0; i < sample.size(); i ++) {
  value.sums.add(min, i, sample.get(i));
}
value.counts.add(min, 1.0d);
}
```

该方法中调用 findNearestCentroid()（实现见附件）找到样本 item 距离最近的中心点索引，然后将其各个维度加到 sums 上，最后 counts 计数加 1。

以上三个方法执行于所有 worker 上，实现局部聚合。接下来看下在 AggregatorOwner 所在 Worker 执行的全局聚合相关操作：

1. merge 的实现：

```
@Override
public void merge(KmeansAggrValue value, KmeansAggrValue partial)
  throws IOException {
value.sums.add(partial.sums);
value.counts.add(partial.counts);
}
```

merge 的实现逻辑很简单，就是把各个 worker 聚合出的 sums 和 counts 相加即可。

2. terminate() 的实现。

```
@Override
public boolean terminate(WorkerContext context, KmeansAggrValue
value)
   throws IOException {
 // Calculate the new means to be the centroids (original sums)
 DenseMatrix newCentriods = calculateNewCentroids(value.sums, value.
counts, value.centroids);
```

```
 // print old centroids and new centroids for debugging
 System.out.println("\nsuperstep: " + context.getSuperstep() +
     "\nold centriod:\n" + value.centroids + " new centriod:\n" +
newCentriods);
 boolean converged = isConverged(newCentriods, value.centroids, 0.
05d);
 System.out.println("superstep: " + context.getSuperstep() + "/"
     + (context.getMaxIteration() - 1) + " converged: " + converged
);
 if (converged || context.getSuperstep() == context.getMaxIteration
() - 1) {
   // converged or reach max iteration, output centriods
   for (int i = 0; i < newCentriods.numRows(); i++) {
     Writable[] centriod = new Writable[newCentriods.numColumns()];
     for (int j = 0; j < newCentriods.numColumns(); j++) {
       centriod[j] = new DoubleWritable(newCentriods.get(i, j));
     }
     context.write(centriod);
   }
   // true means to terminate iteration
   return true;
 }
 // update centriods
 value.centroids.set(newCentriods);
 // false means to continue iteration
 return false;
}
```

teminate() 中首先根据 sums 和 counts 调用 calculateNewCentroids() 求平均计算出新的中心点。然后调用 isConverged() 根据新老中心点欧拉距离判断是否已经收敛。如果收敛或迭代次数达到最大数，则将新的中心点输出并返回 true，以结束迭代。否则更新中心点并返回 false 以继续迭代。其中 calculateNewCentroids() 和 isConverged() 的实现见附件。

- **main 方法**

  main 方法用以构造 GraphJob，然后设置相应配置，并提交作业。代码如下：

```
public static void main(String[] args) throws IOException {
  if (args.length < 2)
    printUsage();
  GraphJob job = new GraphJob();
  job.setGraphLoaderClass(KmeansReader.class);
  job.setRuntimePartitioning(false);
  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");
```

```
}
```

说明：

job.setRuntimePartitioning(false) 设置为 false 后，各个 worker 加载的数据不再根据 Partitioner 重新分区，即谁加载的数据谁维护。

总结

本文介绍了 MaxCompute Graph 中的 Aggregator 机制，API 含义以及 Kmeans Clustering 示例。总的来说，Aggregator 的基本步骤，如下所示：

1. 每个 worker 启动时执行 createStartupValue 用以创建 AggregatorValue。
2. 每轮迭代开始前，每个 worker 执行 createInitialValue 来初始化本轮的 AggregatorValue。
3. 一轮迭代中每个点通过 context.aggregate() 来执行 aggregate() 实现 worker 内的局部迭代。
4. 每个 Worker 将局部迭代结果发送给 AggregatorOwner 所在的 Worker。
5. AggregatorOwner 所在 worker 执行多次 merge，实现全局聚合。
6. AggregatorOwner 所在 Worker 执行 terminate 用以对全局聚合结果做处理并决定是否结束迭代。

附件

本文所需附件，请参见 *Kmeans*。

9 安全指南

9.1 目标用户

本章节文档主要面向 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 多租户的数据安全体系，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。

9.2 快速开始

9.2.1 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。

操作步骤：（由项目空间管理员来操作）

```
    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
```

9.2.2 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。

操作方法：

```
    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;
```

9.2.3 设置项目保护模式

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，比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。

操作方法：Jack需要如下操作

```
    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提供了一种机制来支持受保护项目空间的数据流出，即设置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。

```
    use prj1;
    add trustedproject prj2;
```

9.3 用户及授权管理

9.5 用户管理

任意非项目空间 Owner 用户必须被加入 MaxCompute 项目空间中，并被授予相对应权限，方能操作 MaxCompute 中的数据、作业、资源及函数。本文将介绍项目空间 Owner 如何将其他用户（包括 RAM 子账号）加入和移出 MaxCompute，如何给用户授权。

如果您是项目空间 Owner，建议您仔细阅读本文；如果您是普通用户，建议您向 Owner 提出申请，被加入对应的项目空间后再阅读后续章节。

本文的操作均在客户端运行，Linux 系统下运行 **./bin/odpscmd**，Windows 下运行 **./bin/odpscmd.bat**。

添加用户

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
```

云账号的<username>既可以是在 www.aliyun.com 上注册过的有效邮箱地址，也可以是执行此命令的云账号的某个 RAM 子账号，示例如下：

```
add user ALIYUN$odps_test_user@aliyun.com;
add user RAM$ram_test_user;
```

假设 Alice 的云账号为 alice@aliyun.com，那么当 Alice 执行上述两条语句后，通过`list users;`命令即可看到如下结果：

```
RAM$alice@aliyun.com:ram_test_user
ALIYUN$odps_test_user@aliyun.com
```

这表明云账号 odps_test_user@aliyun.com 以及 Alice 通过 RAM 创建的子账号 ram_test_user 已经被加入到了该项目空间中。

添加 RAM 子账号

添加 RAM 子账号有以下两种方式：

- 通过大数据开发套件进行操作，详情请参见 如何添加成员及授权。
- 通过 MaxCompute 客户端常用命令进行操作，详情如下：

> **说明：**
>
> - MaxCompute 只允许主账号将自身的 RAM 子账号加入到项目空间中，不允许加入其它云账号的 RAM 子账号，因此在`add user`时，无需在 RAM 子账号前指定主账号名称，MaxCompute 默认判定命令的执行者即是子账号对应的主账号。
> - MaxCompute 只能够识别 RAM 的账号体系，不能识别 RAM 的权限体系。即用户可以将自身的任意 RAM 子账号加入 MaxCompute 的某一个项目中，但 MaxCompute 在对该 RAM 子账号做权限验证时，并不会考虑 RAM 中的权限定义。

默认情况下，MaxCompute 项目只能够识别阿里云账号系统，用户可以通过`list accountproviders;`命令查看该项目所支持的账号系统，通常情况下仅会看到 ALIYUN 账号，例如下所示：

```
odps@ ****>list accountproviders;
ALIYUN
```

说明：

只有项目空间的Owner 有权限进行`accountproviders`的相关操作。

由上可见，只能看到`ALIYUN`账号体系，如果想添加对 RAM 账号的支持，可以执行`add accountprovider ram;`如下所示：

```
odps@ odps_pd_inter>add accountprovider ram;
OK
```

添加用户成功后，此用户仍不能操作 MaxCompute，需要对用户授予一定权限，用户才能在所拥有的权限范围内操作 MaxCompute。关于授权的更多操作，请参见 授权。

用户授权

添加用户后，项目空间 Owner 或者项目空间管理员需要给该用户进行授权，只有用户获得权限后，才能执行操作。

MaxCompute 提供了 ACL 授权，跨项目空间数据分享及项目空间数据保护等多种策略。下面列举两个常见场景，更多详情请参见 *ACL*授权。

场景一：

假设 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，并申请查看 Table 列表，提交作业和创建表的权限。项目空间的 admin role 或者该项目空间 owner 在客户端执行如下命令：

```
use prj1; --进入项目空间 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
```

场景二：

假设用户云账号为 bob@aliyun.com，已经被添加到某个项目空间（$user_project_name），需要给它授予建表、获取表信息和执行的权限。

项目空间的 admin role或者该项目空间 owner 在客户端可以执行如下命令：

```
grant CreateTable on PROJECT $user_project_name to USER ALIYUN$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 on Table $user_table_name to USER ALIYUN$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 $user_function_name to USER ALIYUN$bob@
aliyun.com;
  -- 向 bob@aliyun.com 授予名为 "$user_function_name" 的 Function 的
Execute（执行）权限
```

给 RAM 子账号授权

通过`list accountproviders;`来查看账号支持情况，如下所示：

```
odps@ ****>list accountproviders;
ALIYUN, RAM
```

由上可见，这个项目空间已经能够支持RAM账号体系，即可以向这个项目空间添加 RAM 子账号并授予某张表的`Describe`权限，如下所示：

```
odps@ ****>add user ram$bob@aliyun.com:Alice;
OK: DisplayName=RAM$bob@aliyun.com:Alice
odps@ ****>grant Describe on table src to user ram$bob@aliyun.com:
Alice;
OK
```

此时，*bob@aliyun.com*的 RAM 子账号*Alice*就可以通过自己的**AccessKeyID**及**AccessKeySecret**登录 MaxCompute ，并对表*src*进行desc操作。

说明：

- 获取 RAM 子账号AccessKeyID及AccessKeySecret的相关操作请参见 *RAM 介绍*。
- 更多 MaxCompute 添加/删除用户的操作请参见本文相关内容。
- 更多有关授权的操作请参见 授权。

删除用户

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，如下所示：

```
remove user
```

说明：

- 当一个用户被移除后，该用户不再拥有访问该项目空间资源的任何权限。
- 移除一个用户之前，如果该用户已被赋予某些角色，则需要先撤销该用户的所有角色。关于角色的介绍请参考 角色管理。
- 当一个用户被移除后，与该用户有关的 *ACL* 授权 仍然会被保留。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。

Alice 执行下述两条命令，即可移除相关用户：

```
remove user ALIYUN$odps_test_user@aliyun.com;
remove user RAM$ram_test_user;
```

查看用户是否移除，命令如下：

```
LIST USERS;
```

查看结果将不会看到这两个账号。此时，表明这两个账号已经被移出项目空间。

删除 RAM 子账号

同样，您也可以通过`remove user`命令删除自身的 RAM 子账号。示例如下：

```
odps@ ****>revoke describe on table src from user ram$bob@aliyun.com:
Alice;
OK
-- 回收子账号 Alice 权限
odps@ ****>remove user ram$bob@aliyun.com:Alice;
Confirm to "remove user ram$bob@aliyun.com:Alice;" (yes/no)? yes
OK
-- 删除子账号
```

如果您是项目空间的 owner，也可以通过`remove accountprovider`将 RAM 账号系统从当前项目中删除，如下所示：

```
odps@ ****>remove accountprovider ram;
Confirm to "remove accountprovider ram;" (yes/no)? yes
OK
odps@ ****>list accountproviders;
```

```
ALIYUN
```

9.6 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：可以访问项目空间内的所有对象、对用户或角色进行管理、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：

- 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。
- DataWorks中成员角色类型对应的 MaxCompute 角色，以及各角色的平台权限详情，请参见 项目管理中的项目成员管理模块。

创建角色

创建角色的命令格式，如下所示：

```
CREATE ROLE ;
```

示例如下：

假设要创建一个 player 角色，需在客户端输入如下命令：

```
create role player;
```

添加用户到角色

添加用户到角色的命令格式，如下所示：

```
GRANT <roleName> TO <full_username> ;
```

示例如下：

假设要将用户 bob@aliyun.com 加入 player 角色中，需在客户端输入如下命令：

```
grant player to bob@aliyun.com;
```

给角色授权

给角色授权的语句与给用户授权相似，更多详情请参见 用户授权。

说明：

给角色授权后，该角色下的所有用户拥有相同的权限。

示例如下：

假设 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 和 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，提交作业和读取表 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;
```

删除角色中的用户

删除角色中的用户的命令格式，如下所示：

```
REVOKE <roleName> FROM <full_username>;
```

示例如下：

假设将用户 bob@aliyun.com 从 player 角色中删除，需在客户端输入如下命令：

```
revoke player from bob@aliyun.com;
```

删除角色

删除角色的命令格式，如下所示：

```
DROP ROLE <roleName>;
```

示例如下：

假设要删除 player 角色，需在客户端输入如下命令：

```
drop role player;
```

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。

9.7 授权

添加用户 后，项目空间 Owner 或者项目空间管理员需要给该用户进行授权，只有用户获得权限后，才能执行操作。所谓授权，即授予用户对 MaxCompute 中的表，任务，资源等客体的某种操作权限，包括：读、写、查看等。

MaxCompute 提供了 ACL 授权，跨项目空间数据分享及项目空间数据保护等多种策略。授权操作一般涉及到三个要素：主体（Subject，可以是用户也可以是角色），客体（Object）和操作（Action）。在 MaxCompute 中，主体是指用户或角色，客体是指项目空间中的各种类型对象。

ACL 授权中，MaxCompute 的客体包括：*Project*，*Table*，*Function*，*Resource*，*Instance*，操作与特定对象类型有关，不同类型的对象支持的操作也不尽相同。

MaxCompute 项目空间支持如下的对象类型及操作：

客体（**Object**）	操作（**Action**）	说明
Project	Read	查看项目空间自身（不包括项目空间的任何对象）的信息，如 CreateTime 等
Project	Write	更新项目空间自身（不包括项目空间的任何对象）的信息，如 Comments
Project	List	查看项目空间所有类型的对象列表
Project	CreateTable	在项目空间中创建 Table
Project	CreateInstance	在项目空间中创建 Instance
Project	CreateFunction	在项目空间中创建 Function
Project	CreateResource	在项目空间中创建 Resource
Project	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	Execute	执行
Function	All	具备上述所有权限
Resource	Read	读取
Resource	Write	更新
Resource	Delete	删除
Resource	All	具备上述所有权限
Instance	Read	读取
Instance	Write	更新
Instance	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 操作，Table 类型的 Select、Alter、Update、Drop 操作需要与 Project 对象的 CreateInstance 操作权限配合使用。
- 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。这与 MaxCompute 的内部实现相关。同样，Table 的 Select 权限也要与 CreateInstance 权限配合使用，当跨项目操作如在项目A里select项目B的table，则需要有项目A的CreateInstance和项目B的table select权限 。
- 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 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）被看作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。

- 类似**SQL92**的授权方法

  MaxCompute 支持的授权方法类似于 SQL92 定义的 GRANT/REVOKE 语法，它通过简单的授权语句来完成对已存在的项目空间对象的授权或撤销授权。授权语法如下：

  ```
  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
  ```

  熟悉 SQL92 定义的 GRANT/REVOKE 语法或者熟悉 Oracle 数据库安全管理的用户容易发现，MaxCompute 的 ACL 授权语法并不支持 [WITH GRANT OPTION] 授权参数。也就是说，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。那么，所有的授权操作都必须由具有以下三种身份之一的用户来完成：

  - 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** 授权的应用示例

  假设云账号用户 alice@aliyun.com 是新加入到项目空间 test_project_a 的成员，Allen 是加入到 bob@aliyun.com 中的 RAM 子账号。在 test_project_a 中，他们需要提交作业、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，如下所示：

```
    use test_project_a; --打开项目空间
    add user aliyun$alice@aliyun.com; --添加用户
    add user ram$bob@aliyun.com:Allen; --添加RAM子账号
    create role worker; --创建角色
    grant worker TO aliyun$alice@aliyun.com; --角色指派
    grant worker TO ram$bob@aliyun.com:Alice;  --角色指派
    grant CreateInstance, CreateResource, CreateFunction, CreateTabl
e, List ON PROJECT test_project TO ROLE worker; --对角色授权
```

- 跨项目空间**Table**、**Resource**、**Function**分享示例

  接前一个示例，aliyun$alice@aliyun.com 和 ram$bob@aliyun.com:Allen 在test_project_a 拥有了一定的权限后，这两个用户还需查询test_project_b中的table prj_b_test_table，且需要用到test_project_b中的UDF prj_b_test_udf。

  test_project_b的管理员执行的授权操作如下：

```
    use test_project_b; --打开项目空间
    add user aliyun$alice@aliyun.com; --添加用户
    add user ram$bob@aliyun.com:Allen; --添加RAM子账号
    create role prj_a_worker; --创建角色
    grant prj_a_worker TO aliyun$alice@aliyun.com; --角色指派
    grant prj_a_worker TO ram$bob@aliyun.com:Alice;  --角色指派
    grant Describe , Select  ON TABLE prj_b_test_table TO ROLE
prj_a_worker; --对角色授予table权限
    grant Read  ON Function prj_b_test_udf TO ROLE prj_a_worker;
 --对角色授予udf权限
    grant Read  ON Resource prj_b_test_udf_resource TO ROLE
prj_a_worker; --对角色授予实现udf的Resource的权限
--授权后，这两个用户在test_project_a中查询表和使用udf的方式如下：
use test_project_a;
select test_project_b:prj_b_test_udf(arg0, arg1) as res from
test_project_b.prj_b_test_table;
```

说明：

若是在test_project_a 中创建udf，则授权时只需进行Resource授权后，使用写法如下：

```
create function function_name as 'com.aliyun.odps.compiler.udf.
PlaybackJsonShrinkUdf' using 'test_project_b/resources/odps-compiler-
playback.jar' -f;
```

9.8 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。

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。
- 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
```

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，仅由 ProjectOwner
 和 Admin 才能有执行权限 。
```

示例如下：

假设需要查看用户云账号 bob@aliyun.com 在当前项目空间的权限，需要在客户端可以执行如下命令：

```
show grants for ALIYUN$bob@aliyun.com;
```

查看RAM子帐号权限：

```
show grants for RAM$主帐号:子帐号;
```

示例如下：

```
show grants for RAM$bob@aliyun.com:Alice;
```

查看指定角色的权限

```
describe role ; --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  [on type ];--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。

9.9 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，ProjectOwner 可以定制适合自己的外部账号支持和鉴权模型。

MaxCompute 支持多种正交的授权机制，如 ACL 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，您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 CheckPermissionUsingACL=true/false
      --激活/冻结ACL授权机制，默认为true
    set 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 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 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
```

说明：

您也可以通过DataWorks进行可视化操作，完成项目空间的相关安全配置，详情请参见 项目配置。

9.10 安全相关语句汇总

9.10.1 项目空间的安全配置

鉴权配置

语句	说明
show SecurityConfiguration	查看项目空间的安全配置
set CheckPermissionUsingACL=true/false	激活/冻结ACL授权机制
set CheckPermissionUsingPolicy=true/false	激活/冻结Policy授权机制
set ObjectCreatorHasAccessPermission=true/false	允许/禁止对象创建者默认拥有访问权限
set ObjectCreatorHasGrantPermission=true/false	允许/禁止对象创建者默认拥有授权权限

数据保护

语句	说明
set ProjectProtection=false	关闭数据保护机制
list TrustedProjects	查看可信项目空间列表
add TrustedProject `<projectName>`	添加可信项目空间
remove TrustedProject `<projectName>`	移除可信项目空间

9.10.2 项目空间的权限管理

用户管理

语句	说明
list users	查看所有已添加进来的用户

语句	说明
add user <username>	添加一个用户
remove user <username>	移除一个用户

角色管理

语句	说明
list roles	查看所有已创建的角色
create role <rolename>	创建一个角色
drop role <rolename>	删除一个角色
grant <rolelist> to <username>	对用户指派一个或多个角色
revoke <rolelist> from <username>	撤销对用户的角色指派

ACL授权

语句	说明
grant <privList> on <objType> <objName> to user <username>	对用户授权
grant <privList> on <objType> <objName> to role <rolename>	对角色授权
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

权限审查

语句	说明
whoami	查看当前用户信息
show grants [for <username>] [on type <objectType>]	查看用户权限和角色
show acl for <objectName> [on type <objectType>]	查看具体对象的授权信息
describe role <roleName>	查看角色的授权信息和角色指派

9.11 跨项目空间的资源分享

9.11.1 基于Package的跨项目空间的资源分享

假设您是项目空间的Owner或管理员（admin角色），某个主账号下多个项目空间，其中项目空间prj1里有一批资源（包括tables、Resources、自定义functions）可以分享给其他项目空间使用，但是若把其他项目空间的user都add到这个prj1项目空间并逐个进行授权操作，需要操作非常多的步骤，对于prj1项目空间来说引入很多跟本项目业务无关（假设存在）的user非常不方便管理。

那么你可以使用本节介绍的跨项目空间的资源分享功能。

如果资源需要精细控制单人使用，且申请人是本业务项目团队成员，那么建议你使用项目空间的用户与授权管理功能：项目空间的用户与授权管理

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。

如果不使用Package，对于下面的场景我们无法有效的解决：

Alifinance项目空间的成员若要访问Alipay项目空间的数据，则需要Alipay项目空间管理员执行繁琐的授权操作：首先需要将Alifiance项目空间中的用户添加到Alipay项目空间中，再分别对这些新加入的用户进行普通授权。

实际上，Alipay项目空间管理员并不期望对Alifiance项目空间中的每个用户都进行授权管理，而更期望有一种机制能使得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。

9.11.2 Package的使用方法

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。

下面分别介绍Package创建者和Package使用者所涉及的操作。

Package创建者

- 创建**Package**

```
create package ;
```

说明：

- 仅project的owner有权限进行该操作。
- 目前创建的package名称不能超过128个字符。

- 将分享的资源添加到**Package**

```
    add project_object to package package_name [with privileges
privileges];--将对象添加到Package
    remove project_object from package package_name;--将对象从Package
移除
    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，添加后不可被更新。若有需要，只能删除和重新添加。

- 许可其他项目空间使用**Package**

```
allow project <prjname> to install package <pkgname> [using label <
number>];
```

- 撤销其他项目空间使用**Package**的许可

```
disallow project <prjname> to install package <pkgname>;
```

- 删除**Package**

```
delete package <pkgname>;
```

- 查看已创建和已安装的**Package**列表

```
show packages;
```

- 查看**Package**详细信息

```
describe package <pkgname>;
```

Package使用者

- 安装**Package**

```
install package <pkgname>;
```

对于安装Package来说，要求pkgName的格式为：`<projectName>.<packageName>`。

> 说明：
> 仅project的Owner有权限进行该操作。

- 卸载**Package**

```
uninstall package <pkgname>;
```

对于卸载Package来说，要求pkgName的格式为：`<projectName>.<packageName>`

- 查看**Package**

```
    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
```

- 使用方项目给本项目其他成员授权访问**Package**

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您的资源)，您必须拥有对该Package的Read权限。

如果请求者没有Read权限，则需要向ProjectOwner或Admin申请。ProjectOwner或Admin可以通过ACL授权机制来完成。

比如，如下的ACL授权允许云账号用户LIYUN$odps_test@aliyun.com访问Package里的资源：

```
    use prj2;
    install package prj1.testpkg;
    grant read on package prj1.testpackage to user aliyun$odps_test@
aliyun.com;
```

场景示例

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(比如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自主授权，无需Jack参与。

操作步骤：

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
```

9.12 列级别访问控制

基于标签的安全（LabelSecurity）是项目空间级别的一种强制访问控制策略（Mandatory Access Control, MAC），它的引入是为了让项目空间管理员能更加灵活地控制用户对列级别敏感数据的访问。

理解MaxCompute中的MAC与DAC差异

在MaxCompute中，强制访问控制机制(MAC)独立于自主访问控制机制(DAC)。为了便于理解，MAC与DAC的关系可以用下面的例子来做个类比。

对于一个国家来说（类比MaxCompute的一个项目空间），这个国家公民要想开车(类比读数据操作)，必须先申请获得驾照(类比申请SELECT权限)。这些就属于DAC考虑的范畴。

但由于这个国家交通事故率一直居高不下，于是该国新增了一条法律：禁止酒驾。此后，所有想开车的人除了持有驾照之外，还必须不能喝酒。类比MaxCompute，这个禁止酒驾就相当于禁止读取敏感度高的数据。这就属于MAC考虑的范畴。

数据的敏感等级分类

LabelSecurity需要将数据和访问数据的人进行安全等级划分。在政府和金融机构，一般将数据的敏感度标记分为四类：0级 (不保密, Unclassified), 1级 (秘密, Confidential), 2级 (机密, Sensitive), 3级 (高度机密, Highly Sensitive)。MaxCompute也遵循这一分类方法。ProjectOwner需要定义明确的数据敏感等级和访问许可等级划分标准，默认时所有用户的访问许可等级为0级，数据安全级别默认为0级。

LabelSecurity对敏感数据的粒度可以支持列级别，管理员可以对表的任何列设置敏感度标记(Label)，一张表可以由不同敏感等级的数据列构成。

对于view，也支持和表同样的设置，即管理员可以对view设置label等级，view的等级和它对应的基表的label等级是独立的，在view创建时，默认的等级也是0。

LabelSecurity默认安全策略

在对数据和user分别设置安全等级标记之后，LabelSecurity的默认安全策略如下：

- (No-ReadUp) 不允许user读取敏感等级高于用户等级的数据，除非有显式授权。
- (Trusted-User) 允许user写任意等级的数据，新创建的数据默认为0级（不保密）。

说明：

- 在一些传统的强制访问控制系统中，为了防止数据在项目空间内部的任意分发，一般还支持更多复杂的安全策略，比如不允许用户写敏感等级不高于用户等级的数据(No-WriteDown)。但在MaxCompute平台中，考虑到项目空间管理员对数据敏感等级的管理成本，默认安全策略并不支持No-WriteDown. 如果项目空间管理员有类似需求，可以通过修改项目空间安全配置(`Set ObjectCreatorHasGrantPermission=false`)以达到控制目的。

- 如果是为了控制数据在不同项目空间之间的流动，则可以将项目空间设置为受保护状态(ProjectProtection)。设置之后，只允许用户在项目空间内访问数据，这样可以有效防止数据流出道项目空间之外。

项目空间中的LabelSecurity安全机制默认是关闭的，ProjectOwner可以自行开启。

LabelSecurity安全机制一旦开启，上述的默认安全策略将被强制执行。当用户访问数据表时，除了必须拥有Select权限外，还必须获得读取敏感数据的相应许可等级。或者说，通过LabelSecurity只是通过CheckPermission的必要而非充分条件。

LabelSecurity操作

- **LabelSecurity机制的开/关**

```
Set LabelSecurity=true|false;
  --开启或关闭LabelSecurity机制，默认为false。
  --此操作必须由ProjectOwner完成，其他操作则可以由Admin角色完成。
```

- **给user设置安全许可标签**

```
SET LABEL <number> TO USER <username>; -- number取值范围:[0, 9]，该操作只能由ProjectOwner或Admin角色完成
--举例:
ADD USER aliyun$yunma@aliyun.com; --添加用户，默认的安全许可标签为0级
ADD USER ram$yunma@aliyun.com:Allen;--添加yunma@aliyun.com的RAM子账号用户Allen
SET LABEL 3 TO USER aliyun$yunma@aliyun.com;
  --设置yunma的安全许可标签为3级，他能访问敏感等级不超过3级的数据
SET LABEL 1 TO USER ram$yunma@aliyun.com:Allen;
  --设置yunma的子账号Allen的安全许可标签为1级，他能访问敏感等级不超过1级的数据
```

- **给数据设置敏感等级标签**

```
SET LABEL <number> TO TABLE tablename[(column_list)]; -- number取值范围:[0, 9]，该操作只能由ProjectOwner或Admin角色完成
--举例:
SET LABEL 1 TO TABLE t1; --设置表t1的label为1级
SET LABEL 2 TO TABLE t1(mobile, addr); --将t1的mobile,addr两列的label设置为2级
SET LABEL 3 TO TABLE t1; --设置表t1的label为3级. 注意此时mobile,addr两列的label仍为2级
```

说明：

从上面例子可以看出，显式地对列设置的标签会覆盖对表设置的标签，而不会考虑标签设置的顺序以及敏感等级的高低。

- 显式授权低级别用户访问特定的高敏感级数据表

```
 --授权：
 GRANT LABEL <number> ON TABLE <tablename>[(column_list)] TO USER <
username> [WITH EXP <days>]; --默认过期时间是180天
 --撤销授权：
 REVOKE LABEL ON TABLE <tablename>[(column_list)] FROM USER <
username>;
 --清理过期的授权：
 CLEAR EXPIRED GRANTS;
 --举例：
 GRANT LABEL 2 ON TABLE t1 TO USER ram$yunma@aliyun.com:Allen WITH
EXP 1; --显式授权Allen访问t1表中敏感度不超过2级的数据，授权有效期为1天
 GRANT LABEL 3 ON TABLE t1(col1, col2) TO USER ram$yunma@aliyun.com
:Allen WITH EXP 1; --显式授权alice访问t1(col1, col2)中敏感度不超过3级的数
据，授权有效期为1天
 REVOKE LABEL ON TABLE t1 FROM USER ram$yunma@aliyun.com:Allen; --撤
销alice对t1表的敏感数据访问
```

说明：

目前取消用户对table的label权限，会同时取消该用户对table字段的label的权限

- 查看一个用户能访问哪些敏感数据集

```
 SHOW LABEL [<level>] GRANTS [FOR USER <username>];
   --省略 [FOR USER <username>]时，查看当前用户所能访问的敏感数据集
   --省略<level>时，将显示所有label等级的授权；若指定<level>，则只显示指定等
级的授权
```

- 查看一个敏感数据表能被哪些用户访问

```
 SHOW LABEL [<level>] GRANTS ON TABLE <tablename>;
   --显示指定table上的Label授权
```

- 查看一个用户对一个数据表的所有列级别的**Label**权限

```
 SHOW LABEL [<level>] GRANTS ON TABLE <tablename> FOR USER <username
>;
   --显示指定用户对指定table上列级别的Label授权
```

- 查看一个表中所有列的敏感等级

```
DESCRIBE <tablename>;
```

- 控制**package**安装者对**package**中敏感资源的许可访问级别

```
 ALLOW PROJECT <prjName> TO INSTALL PACKAGE <pkgName> [USING LABEL <
number>];
```

```
--由package创建者授权，授予package安装者对package中敏感资源的许可访问级别
```

说明：

- 省略[USING LABEL <number>]时，默认为0级，即只可以访问非敏感数据。
- 跨项目空间访问敏感数据时，package安装者的项目空间中的所有用户都将使用此命令中许可的访问级别

LabelSecurity应用场景示例

- 限制项目空间中所有非**Admin**用户对一张表的某些敏感的列的读访问

  场景说明：user_profile是某项目空间中的一张含有敏感数据的表，它包含有100列，其中有5列包含敏感数据：id_card, credit_card, mobile, user_addr, birthday. 当前的DAC机制中已经授权了所有用户对该表的Select操作。ProjectOwner希望除了Admin之外，所有用户都不允许访问那5列敏感数据。

  ProjectOwner操作步骤如下：

```
set LabelSecurity=true;
  --开启LabelSecurity机制
set label 2 to table user_profile(mobile, user_addr, birthday);
  --将指定列的敏感等级设置为2
set label 3 to table user_profile(id_card, credit_card);
  --将指定列的敏感等级设置为3
```

说明：

以上操作执行之后，所有非Admin用户都将无法访问那5列数据。如果有人因业务需要确实要访问这些敏感数据，那么需要ProjectOwner或Admin的授权。

Alice是项目空间中的一员，由于业务需要，她要申请访问user_profile的mobile列的数据，需要访问1周时间。项目空间管理员操作步骤如下：

```
GRANT LABEL 2 ON TABLE user_profile TO USER ALIYUN$alice@aliyun.com
WITH EXP 7;
```

说明：

敏感等级为2的数据一共有三列：mobile, user_addr, birthday。那么，当上述授权成功后，Alice将能访问这三列数据。这里存在轻微的过渡授权。如果管理员合理设置数据列的敏感度，那么这种过渡授权是可以避免的。

- 限制项目空间中已获得敏感数据访问许可的用户在项目空间内对敏感数据的复制与肆意传播

场景说明：在上一个场景中，Alice由于业务需要而获得了等级为2的敏感数据的访问权限。但管理员仍然担心Alice可能会将user_profile表中的敏感等级为2的那些数据复制到她自己新建的另一张表user_profile_copy中，从而进一步将user_profile_copy表自主授权给Bob访问。如何限制Alice的这种行为？

考虑到安全易用性和管理成本，LabelSecurity的默认安全策略允许WriteDown，即允许用户向敏感等级不高于用户等级的数据列写入数据，因此MaxCompute还无法从根本上解决此问题。但这里可以限制用户的自主授权行为：允许对象创建者只能访问自己创建的数据，而不允许自主授权该对象给其他用户。操作如下：

```
SET ObjectCreatorHasAccessPermission=true;
  --允许对象创建者操作对象
SET ObjectCreatorHasGrantPermission=false;
  --不允许对象创建者授权对象给其他用户
```