阿里云
MaxCompute

开发

文档版本：20190716

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。
*## *	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 数据类型..... 1
2 常用命令..... 8
2.1 常用命令列表.....8
2.2 项目空间操作.....9
2.3 表操作.....13
2.4 实例操作.....19
2.5 资源操作.....23
2.6 函数操作.....25
2.7 SET操作.....27
2.8 时区配置操作.....29
2.9 其他操作.....31
3 数据上传下载.....33
3.1 数据上传下载概述.....33
3.2 数据通道服务连接.....34
3.3 数据上云的工具选择.....35
3.4 工具介绍.....36
3.5 Tunnel上传下载命令..... 38
3.6 批量数据通道SDK介绍.....50
3.6.1 批量数据通道概要.....50
3.6.2 TableTunnel.....51
3.6.3 InstanceTunnel.....52
3.6.4 UploadSession..... 53
3.6.5 DownloadSession.....55
3.6.6 TunnelBufferedWriter.....56
3.7 批量数据通道SDK示例.....57
3.7.1 示例概述.....57
3.7.2 简单上传示例.....58
3.7.3 简单下载示例.....60
3.7.4 多线程上传示例.....63
3.7.5 多线程下载示例.....65
3.7.6 BufferedWriter上传示例.....68
3.7.7 BufferedWriter多线程上传示例.....68
3.7.8 复杂数据类型上传下载示例.....69
3.8 数据集成导入导出数据..... 71
3.9 DataHub实时数据通道.....79
4 SQL及函数.....80
4.1 SQL概述.....80

4.2 运算符.....82
4.3 类型转换.....85
4.4 SQL限制项.....92
4.5 INSERT语句.....93
4.5.1 更新表数据（INSERT OVERWRITE and INTO）.....93
4.5.2 多路输出（MULTI INSERT）.....95
4.5.3 输出到动态分区（DYNAMIC PARTITION）.....96
4.6 DDL语句.....98
4.6.1 表操作.....98
4.6.2 生命周期操作.....109
4.6.3 分区和列操作.....110
4.6.4 视图操作.....113
4.7 Lateral View.....115
4.8 与其他SQL语法的差异.....117
4.9 SELECT语句.....121
4.9.1 SELECT语法介绍.....121
4.9.2 SELECT语序.....126
4.9.3 子查询.....127
4.9.4 交集、并集和补集.....130
4.9.5 JOIN.....133
4.9.6 SEMI JOIN.....135
4.9.7 MAPJOIN HINT.....135
4.9.8 HAVING子句.....136
4.9.9 GROUPING SETS.....137
4.9.10 SELECT TRANSFORM语法.....139
4.10 脚本模式.....145
4.11 COMMON TABLE EXPRESSION（CTE）.....149
4.12 EXPLAIN.....151
4.13 VALUES.....153
4.14 参数化视图.....156
4.15 内建函数.....159
4.15.1 与MySQL、Oracle内建函数对照表.....159
4.15.2 日期函数.....165
4.15.3 数学函数.....185
4.15.4 窗口函数.....205
4.15.5 聚合函数.....223
4.15.6 字符串函数.....236
4.15.7 其他函数.....261
4.16 UDF.....283
4.16.1 UDF概述.....283
4.16.2 Java UDF.....284
4.16.3 Python UDF.....298
4.16.4 UDTF使用说明.....304
4.16.5 UDAF/UDTF动态参数说明.....304
4.17 UDF示例程序.....306

4.17.1 JSON字符串获取示例.....306
4.17.2 JSON字符串值判断示例.....311
4.17.3 不同类型数据转换JSON STRING示例.....318
4.17.4 JSON字符串增加键值示例.....321
4.17.5 使用正则表达式替换字符串示例..... 324
4.17.6 获取字符串（不含分隔符）value示例..... 326
4.17.7 获取字符串（含有分隔符）value示例..... 329
4.17.8 获取指定日期格式的时间示例..... 331
4.17.9 取余函数示例..... 337
4.17.10 获取URL中指定位置的字符示例..... 341
4.18 UDT.....344
4.19 UDJ..... 354
4.20 附录..... 364
4.20.1 转义字符..... 364
4.20.2 LIKE字符匹配.....365
4.20.3 正则表达式规范..... 365
4.20.4 保留字与关键字..... 368
4.20.5 与Hive数据类型映射表..... 369
5 MapReduce..... 370
5.1 概要..... 370
5.1.1 MapReduce概述..... 370
5.1.2 扩展MapReduce..... 373
5.1.3 开源兼容MapReduce..... 373
5.2 功能介绍..... 378
5.2.1 基本概念.....378
5.2.2 作业提交.....379
5.2.3 输入与输出.....381
5.2.4 资源使用.....381
5.2.5 本地运行.....381
5.3 MR限制项汇总.....384
5.4 示例程序..... 387
5.4.1 WordCount示例.....387
5.4.2 MapOnly示例..... 389
5.4.3 多路输入输出示例..... 392
5.4.4 多任务示例.....396
5.4.5 二次排序示例.....398
5.4.6 使用资源示例.....401
5.4.7 使用Counter示例.....403
5.4.8 Grep示例..... 406
5.4.9 Join示例.....410
5.4.10 Sleep示例.....413
5.4.11 Unique示例..... 414
5.4.12 Sort示例.....417
5.4.13 分区表输入示例..... 419
5.4.14 Pipeline示例.....421

5.5 Java SDK..... 423
5.5.1 原生SDK概述.....423
5.5.2 兼容版本SDK概述.....429
6 Java沙箱.....443
7 外部表.....447
7.1 外部表概述..... 447
7.2 OSS的STS模式授权.....448
7.3 访问OSS非结构化数据..... 450
7.4 处理OSS的开源格式数据..... 463
7.5 输出到OSS的非结构化数据..... 470
7.6 访问OTS非结构化数据..... 478
8 PyODPS.....484
8.1 快速入门..... 484
8.2 安装指南..... 486
8.3 工具平台使用指南..... 488
8.3.1 工具平台使用概述.....488
8.3.2 从平台到自行部署.....488
8.3.3 在DataWorks上使用PyODPS.....489
8.4 基本操作..... 493
8.4.1 基本操作概述.....493
8.4.2 项目空间.....493
8.4.3 表.....493
8.4.4 SQL.....499
8.4.5 任务实例.....502
8.4.6 资源.....504
8.4.7 函数.....506
8.4.8 XFlow和模型..... 507
8.5 DataFrame..... 509
8.5.1 DataFrame概述..... 510
8.5.2 快速入门.....510
8.5.3 创建DataFrame..... 514
8.5.4 Sequence..... 516
8.5.5 Collection..... 518
8.5.6 执行.....526
8.5.7 列运算.....531
8.5.8 聚合操作.....546
8.5.9 排序、去重、采样、数据变换..... 551
8.5.10 对所有行和列调用自定义函数..... 557
8.5.11 MapReduce API.....560
8.5.12 数据合并..... 567
8.5.13 窗口函数..... 570
8.5.14 绘图..... 572
8.5.15 调试指南..... 575
8.6 交互体验增强..... 577

8.6.1 命令行增强 577
8.6.2 IPython增强 579
8.6.3 Jupyter Notebook增强 582
8.7 配置选项 586
8.8 API概述 588
8.9 常见问题与最佳实践 588
9 Spark 590
9.1 Spark概述 590
9.2 搭建开发环境 591
9.3 应用开发示例 593
9.3.1 Java/Scala开发示例 593
9.3.2 PySpark开发示例 599
9.4 运行模式 602
9.5 作业诊断 605
9.6 Spark常见问题 607
10 交互式分析 (Lightning) 609
10.1 概述 609
10.2 通过第三方工具连接服务 611
10.3 访问域名Endpoint 614
10.4 通过JDBC连接MaxCompute服务 616
10.4.1 JDBC驱动程序 616
10.4.2 配置JDBC连接 618
10.4.3 常见工具的连接 619
10.5 查询语法与内建函数 627
10.6 查看或取消查询作业 628
10.7 约束与限制 629
10.8 常见问题 629
11 图模型 634
11.1 图模型概述 634
11.2 Aggregator机制概述 637
11.3 Graph作业 645
11.4 SDK概述 648
11.5 开发和调试 649
11.6 应用限制 656
11.7 示例程序 656
11.7.1 单源最短距离 656
11.7.2 PageRank 659
11.7.3 K-均值聚类 662
11.7.4 BiPartiteMatchiing 667
11.7.5 强连通分量 670
11.7.6 连通分量 677
11.7.7 拓扑排序 680
11.7.8 线性回归 683
11.7.9 三角形计数 687

11.7.10 输入点表..... 689
11.7.11 输入边表..... 695
12 Job运行信息查看..... 701
12.1 使用Logview查看Job信息..... 701
12.2 巧用MaxCompute编译器的错误和警告..... 705
13 MaxCompute开源支持..... 708

1 数据类型

本文为您介绍MaxCompute2.0支持的数据类型，包括基本数据类型和复杂类型。

基本数据类型

MaxCompute支持多种数据类型，特别是MaxCompute升级到2.0后支持的基本数据类型更丰富，可参考本文查看基本数据类型和复杂类型。

由于原类型系统设计原因，如果您需要使用新数据类型系统，需首先设置`set odps.sql.type.system.odps2=true;`或`setproject odps.sql.type.system.odps2=true;`，否则可能产生报错`xxxx type is not enabled in current mode`。同时，需要注意对原有任务的影响，具体请参见本文中提到的新类型相关注意事项。

说明:

新数据类型开关`odps.sql.type.system.odps2`设置仅支持字母小写。

类型	是否2.0版本新增	常量定义	描述
TINYINT	是	1Y、-127Y	8位有符号整形，取值范围：-128~127。
SMALLINT	是	32767S、-100S	16位有符号整形，取值范围：-32768~32767。
INT	是	1000、-15645787	32位有符号整形，取值范围：-2^{31} ~2^{31} -1。详见注释1和注释2。
BIGINT	否	100000000000L、-1L	64位有符号整形，取值范围：-2^{63} +1~2^{63} -1。
FLOAT	是	无	32位二进制浮点型。
DOUBLE	否	3.1415926 1E+7	64位二进制浮点型。
DECIMAL	否	3.5BD、 99999999999.9999999BD	10进制精确数字类型，整形部分取值范围：-10^{36} +1~10^{36} -1，小数部分精确到10^{-18} 。详见注释5。
VARCHAR(n)	是	无	变长字符类型，n为长度，取值范围：1~65535。详见注释3。
STRING	否	“abc”、’ bcd’ 、” alibaba”、 ‘inc’	字符串类型，目前长度限制为8M。详见注释4。

类型	是否2.0版本新增	常量定义	描述
BINARY	是	无	二进制数据类型，目前长度限制为8M。
DATETIME	否	DATETIME ‘2017-11-11 00:00:00’	日期时间类型，取值范围：0000年1月1日~9999年12月31日，精确到毫秒。详见注释6。
TIMESTAMP	是	TIMESTAMP ‘2017-11-11 00:00:00.123456789’	与时区无关的时间戳类型，取值范围：0000年1月1日~9999年12月31日23.59:5 确到纳秒。 说明: 对于部分时区相关的函数，例如`cast(<a timestamp> as string)`，会要求与时区无关的TIMESTAMP按照当前时区相符的方式来展现。
BOOLEAN	否	True、False	BOOLEAN类型，取值范围：True、False。

上述的各种数据类型均可为Null。

说明:

新类型系统相关注意事项：

- 新类型使用范围：
  - MaxCompute SQL支持新数据类型。
  - 新版本MapReduce支持新数据类型。
  - SDK 0.27.2-public版本及以上、客户端0.27.0版本及以上支持新数据类型。
- 打开新类型

  由于历史原因，目前新数据类型默认不能使用，要使用需执行Flag开启（Flag命令仅支持小写）：

  - Session级别：SQL或者新版MR任务涉及使用新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需

在语句前加上set语句`set odps.sql.type.system.odps2=true;`，并与该语句一起提交执行。通过MaxCompute Studio提交的SQL工具默认自动加上该set语句提交执行。应用示例请参见创建和查看表。

- Project级别（需特别注意打开后的影响）：即支持对Project级别进行新类型打开。Project的Owner可根据需要对Project进行设置，命令为：

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

  setproject的详细说明请参见其他操作。

- 打开新类型`odps.sql.type.system.odps2`的主要影响：

  - 注释1：关键字INT语义的变化。打开新类型后SQL中的INT关键字是指32位整型，例如`cast(a as INT)`是将a类型转换成32位整型。未打开新类型会被转换成BIGINT，是64位。例如SQL中写`cast(a as INT)`相当于`cast(a as BIGINT)`，或`create table t(a INT)`相当于`create table a (BIGINT)`。为与MaxCompute原有模式兼容，新类型系统在未设定`odps.sql.type.system.odps2=true;`的情况下，仍

保留此转换，但会报告一个警告：提示INT被当作BIGINT处理了。如果您的脚本有此种情况，建议全部改写为BIGINT，避免混淆。

- 整型常量的语义变化

  例如`SELECT 1 + a;`中的整型常量1：

  - 未打开新类型时，会被当做BIGINT来处理（如果这个常量太长了，超过了BIGINT的值域，如100000000000000000000000000，则会被当做DOUBLE来处理）。
  - 打开新类型后， 整形常量会优先被解释为INT类型1就是32位整形的1（如果这个常量的值超过了INT的值域而又没有超过BIGINT的值域，则会解释为 BIGINT类型。如果超过了BIGINT的值域，则会解释为DOUBLE类型）。
  - 可能出现的兼容性问题：类型是INT导致后续一些操作调用到的函数原型不一致，包括落盘（即写入磁盘）后生成的新类型表会导致周边工具以及后续作业的行为改变。

- 隐式类型转换规则变化

  - 打开新类型时某些隐式类型转换会被禁用，包括STRING->BIGINT，STRING->DATETIME，DOUBLE->BIGINT，DECIMAL->DOUBLE，DECIMAL->BIGINT都是有精度损失或者报错的风险。这种情况可以通过cast函数做强制转换的方式来解决。
  - 隐式类型转换影响较大的地方是函数调用和insert，未开启新类型时能执行通过的SQL，在开启新后执行会报错。

- 支持的操作，内置函数，UDF不一样

  在未打开新类型时，部分以新类型作为参数和返回值的操作及内置函数会被直接忽略，打开新类型后才可以正常使用它们。主要影响如下：

  - 一些内置操作必须打开新类型Flag才能使用，例如返回INT类型的函数（接受INT作为参数的内置函数，基本都有BIGINT的重载），例

如YEAR、QUARTER、MONTH、DAY、HOUR、MINUTE、SECOND、MILLISECOND、NAN这些函数实际上可以用DATEPART内置函数来实现。

- UDF的Resolve不一致。例如UDF包含BIGINT和INT两个重载，原类型系统下由于无法识别INT，一定会执行BIGINT的重载，而新类型系统下可能会被解析到INT的重载。

- BIGINT关键字的解析不一样

  在旧类型系统下，单独一个整形常量，例如123，在旧类型下是BIGINT类型，在新类型下是INT类型。可能的兼容性问题：类型是INT会导致后续一些操作调用到的函数原型不一致，包括落盘（即写入磁盘）后生成的新类型表会导致周边工具以及后续作业的行为改变。

- 分区列的类型支持不同

  - 未打开新类型时分区列的类型实际上只有STRING一种。
  - 打开新类型后，分区列的类型完整支持STRING、VARCHAR、CHAR、TINYINT、SMALLINT、INT、BIGINT这几种类型。
  - 未打开新类型时有特殊的处理：INSERT操作中的分区字段，要当做STRING类型来处理。如输入：`insert overwrite table t partition (pt = 045) select ...;`，注意这里045没有引号，因此这里应该是当成整型来解析，即045实际上是45。但是未打开新类型时，前面的0还在，即045，打开新类型后将去掉这个特殊处理，即为45。

- 集合操作的最后的LIMIT语句行为变更

  例如`SELECT * FROM t1 UNION ALL SELECT * FROM t2 limit 10;`语句：

  - 未打开新类型时，实际是`SELECT * FROM t1 UNION ALL SELECT * FROM ( SELECT * FROM t2 limit 10) t2;`。
  - 打开新类型后，是`SELECT * FROM (SELECT * FROM t1 UNION ALL SELECT * FROM t2 ) t limit 10;`。

  具有相同行为的操作，除了交集、并集和补集、INTERSECT和EXCEPT等集合操作，还有LIMIT、ORDER BY、DISTRIBUTE BY、SORT BY、CLUSTER BY操作。

- IN表达式的类型解析不同

  IN表达式，例如`a in (1, 2, 3)`：

  - 未打开新类型时，要求IN后面的括号里的所有值，其类型必须一致。
  - 打开新类型后，IN后面的括号里的所有值支持隐式类型转换一致。

- 注释2：对于INT常量，如果超过INT取值范围，会转为BIGINT。如果超过BIGINT取值范围，会转为DOUBLE。在旧版MaxCompute中，因为历史原因，SQL脚本中的所有INT类型都被转换为BIGINT，如下所示。

```
create table a_bigint_table(a int); --这里的INT实际当作BIGINT处理。
select cast(id as int) from mytable; --这里的INT实际当作BIGINT处理。
```

- 注释3：VARCHAR类型常量可通过STRING常量的隐式转换表示。
- 注释4：STRING常量支持连接，例如abc和xyz会解析为abcxyz，不同部分可以写在不同行上。
- 注释5：insert一个常量到DECIMAL字段时，请注意常量的写法需与常量定义中的格式保持一致，例如下面示例代码中的3.5BD。

```
insert into test_tb(a) values (3.5BD)
```

- 注释6：目前查询显示的时间值不包含毫秒。Tunnel命令通过-dfp来指定时间格式，可以指定到毫秒显示，例如tunnel upload -dfp 'yyyy-MM-dd HH:mm:ss.SSS'。关于Tunnel命令的更多信息，请参见Tunnel上传下载命令。

复杂数据类型

MaxCompute2.0支持的复杂类型如下表所示。

类型	定义方法	构造方法
ARRAY	· array<int> · array<struct<a:int, b:string>>	· array(1, 2, 3) · array(array(1, 2), array(3, 4))
MAP	· map<string, string> · map<smallint, array<string>>	· map(“k1”, “v1”, “k2”, “v2”) · map(1S, array(‘a’, ‘b’), 2S, array(‘x’, ‘y’))
STRUCT	· struct<x:int, y:int> · struct<field1:bigint, field2:array<int>, field3:map<int, int>>	· named_struct(‘x’, 1, ‘y’, 2) · named_struct(‘field1’, 100L, ‘field2’, array(1, 2), ‘field3’, map(1, 100, 2, 200))

说明:

MaxCompute的复杂数据类型可以被任意嵌套使用，相关配套的内建函数说明请参见ARRAY、MAP、STRUCT。

PyODPS数据类型设置

如果您使用PyODPS，可以通过下列方法打开新数据类型开关：

- 如果通过execute_sql方式调用，可以执行 `o.execute_sql('set odps.sql.type.system.odps2=true;query_sql', hints={"odps.sql.submit.mode" : "script"})`
- 如果通过Dataframe调用，例如persist、execute、to_pandas等立即执行方法，可通过`hints`参数设置。下图中设置方法仅针对单个作业生效。

```
from odps.df import DataFrame
users = DataFrame(o.get_table('odps2_test'))
users.persist('copy_test',hints={'odps.sql.type.
```

- 如果通过Dataframe调用，且需要全局生效，需要使用Option参数 `options.sql.use_odps2_extension = True`。

2 常用命令

2.1 常用命令列表

本文为您列举MaxCompute客户端的常用命令。

最新的MaxCompute服务对常用命令做了调整，新的命令风格更贴近于Hive的使用方式，方便原有的Hadoop/Hive用户。

MaxCompute提供了对项目空间、表、资源及实例等对象的一系列操作。您可以通过客户端命令或SDK来操作这些对象。

说明:

- 本章节介绍的常用命令适用于MaxCompute客户端。
- 如果想了解如何安装、配置客户端，请参见准备工作>安装并配置客户端。
- 对于SDK的更多介绍，请参见MaxCompute SDK介绍。

常用命令

addaliasalter

costcreate

delete/dropdesc/describedownload/get

extended

flag/flagsfunctions

get

helphistory

jar/mapreduce

kill

lifecyclelist

odpscmd

partition

q/quit

resource

setshowstatus

tunnel

upload

waitwho

使用限制

在进行资源操作时，请注意每个资源文件的大小不能超过500MB，单个SQL、MapReduce任务所引用的资源总大小不能超过2048MB，更多限制请参见MR限制汇总。

2.2 项目空间操作

本文向您介绍进入项目空间、设置空间属性（权限和白名单功能等）的命令操作。

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。

创建项目空间即创建MaxCompute项目。

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，详情请参见项目空间列表。

查询项目空间

命令格式：

```
list projects;
```

用途：用于供主账号查看其创建的项目列表。

> 说明：
> 该命令从odpscmd 0.30.2版本开始支持。

SetProject

命令格式如下：

```
setproject <KEY>=<VALUE>;
```

行为说明如下：

- 您可以使用setproject命令设置Project属性。

  例如，以下示例是设置允许全表扫描的方法。

  ```
  setproject odps.sql.allow.fullscan=true;
  ```

- 当不指定<KEY>=<VALUE>时，显示当前Project的属性配置。命令格式如下：

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

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	项目空间是否允许全表扫描	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（不报错）/ false（报错）
odps.table.lifecycle	ProjectOwner	· optional：创建表时，lifecycle子句为可选设置，如果用户不设置生命周期，则该表永久有效。 · mandatory：lifecycle子句为必选设置，用户必须设置生命周期。 · inherit：如果用户不设置生命周期，则该表的生命周期为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.security.ip.whitelist	ProjectOwner	指定访问Project的IP白名单	IP列表，逗号分隔
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.sql.type.system.odps2	ProjectOwner	打开2.0新类型	true（允许）/ false（禁止）
odps.sql.hive.compatible	ProjectOwner	打开Hive兼容模式后，MaxCompute才可以支持Hive指定的各种语法，如inputRecordReader, outputRecordReader, Serde等。	true（允许）/ false（禁止）

说明:

打开新类型的主要影响：

- 某些隐式类型转换会被禁用， 包括STRING->BIGINT、STRING->DATETIME、 DOUBLE->BIGNIT、DECIMAL->DOUBLE、DECIMAL->BIGINT都是有精度损失或者报错的风险。当然依旧可以用`cast`来做强制转换。
- 常量类型会变化。单独一个整形常量，如123，在旧类型下是BIGINT类型，在新类型下是INT类型。
- UDF Resolve结果可能变化，例如UDF包含BIGINT和INT两个重载。旧类型下一定走BIGINT的重载，而新类型下可能会被解析到INT的重载。

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。
- VPC环境下当前仅支持针对SLB整段地址进行放通。
- 如果您误操作，将自己屏蔽，请通过提工单向阿里云技术支持寻求帮助。

白名单中IP列表的表示形式有以下三种，目前已支持IPV6格式。

- 单纯IP：例如101.132.236.134、FE80:0202:B3FF:FE1E:8329。
- 子网掩码：例如100.116.0.0/16、FE80:0101:4567:F456:0202:B3FF:1111:1111/126。
- 网段：例如101.132.236.134-101.132.236.144、FE80:0101:4567:F456:0202:B3FF:FE1E:8330-FE80:0101:4567:F456:0202:B3FF:FE1E:8331。

以上三种形式可以写在同一个命令中，用逗号分割。

使用命令行工具设置IP白名单的方法示例如下。

```
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会认为Project关闭了白名单功能。

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

2.3 表操作

本文将向您介绍如何通过MaxCompute客户端使用常用命令进行表的创建、删除、查看信息等操作。

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

```
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：

创建一张表。

> 说明：
>
> - 在创建表时，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。
> - 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。
- PARTITIONED BY：指定表的分区字段，旧版MaxCompute仅支持String类型。MaxCompute2.0对分区类型的支持进行了扩充，目前MaxCompute支持Tinyint、Smallint、Int、Bigint、Varchar和String分区类型。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。
- 分区限制：分区值不可以有双字节字符（如中文），必须是以英文字母A-Z开头。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。
- 数据类型：包括Bigint、Double、Boolean、Datetime、Decimal和String等多种数据类型。
- lifecycle指明此表的生命周期，单位：天。`create table like`语句不会复制源表的生命周期属性。
- 目前，在表中建的分区层次不能超过6级。一个表允许的分区个数支持按照具体的project配置，默认60,000个。

示例

```
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 。
```

```
create table test1 (key string); -- 创建非分区表，表名test1，字段名key，数据类型string。
create table test2 (key bigint) partitioned by (pt string, ds string);
 --创建分区表
create table test3 (key boolean) partitioned by (pt string, ds string
) lifecycle 100; -- 创建带有生命周期的表
create table test4 like test3; -- 除生命周期属性外，test4的其他属性（字段类型，分区类型等）均与test3完全一致
create table test5 as select * from test2; -- 这个操作会创建test5，但分区，生命周期信息不会被拷贝到目标表中。
-- 此操作仅会将 test2的数据复制到test5中（如果test2有数据的话，此示例中test2为空表，后续章节会介绍数据导入）。
```

说明:

- 添加分区请参见分区和列操作。
- 生命周期的修改请参见修改表的生命周期属性。
- 更多信息请参见表操作。

ChangeOwner

MaxCompute SQL支持通过changeowner命令来修改表的拥有人（表owner）。

命令格式

```
alter table table_name changeowner to 'ALIYUN$xxx@aliyun.com';
```

Drop Table

命令格式

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

行为说明：

- 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

```
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

```
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

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：表数据所占压缩后的存储容量大小，压缩比一般为5倍，单位Byte。

  说明:

  通过Describe Table查看到的Size包含了在回收站的数据Size。如果您需要清空回收站，可以先执行 purge table table_name;，然后再 Describe Table查看除回收站以外的数据大小。 您也可以执行show recyclebin; 查看本项目中回收站内的数据明细。

- 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

```
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视图操作。

查看分区信息

命令格式

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

行为说明

查看某个分区表具体的分区信息。

示例

```
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和Show Tables like

命令格式

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

行为说明

- 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

```
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

```
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

列出一张表中的所有分区。

示例

```
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

通过客户端命令操作您可以查看指定的实例信息、实例状态，停止实例以及获得实例任务运行日志信息等。

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];
SHOW P -p <project name>;
```

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，默认最大返回50条。注意执行该命令的用户需要有project的list权限。如需返回更多条记录，请使用-limit number参数，例如`show p -all -limit 100`表示返回当前项目下100条执行过的实例记录。
- project name：项目名称，您使用的账号必须已经是项目成员。
- 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  20150428xxxxxxxxxxxxxxxxx
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; -- 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 20131225123xxxxxxxxxxxxxx;
Success
```

查看ID为20131225123xxxxxxxxxxxxxx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;或top instance -all;
```

行为说明如下：

权限要求：用户为项目Owner或管理员角色

top instance返回项目中当前云账号所提交的正在执行的作业信息，包括INSTANCEID | Owner | Type | StartTime | Progress | Status | Priority | RuntimeUsage（CPU/MEM） | TotalUsage（CPU/MEM） | QueueingInfo（POS/LEN）等。

top instance -all返回当前项目下所有正在执行的作业，默认最大返回50条。如需返回更多条记录，请使用-limit number参数。

示例如下：

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

说明：

上述示例的命令在客户端运行。

Kill Instance

命令格式如下：

```
kill instance_id; --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 20131225123xxxxxxxxxxxxxxx;
```

停止ID为20131225123xxxxxxxxxxxxxxx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 ; --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@ $project_name> desc instance 20150715xxxxxxxxxxxxxx;
ID                                      20150715xxxxxxxxxxxxxx
Owner                                   ALIYUN$XXXXXX@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20150715xxxxxxxxxxxxxx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; --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 20170925161122379gxxxxxx;
ID = 20170925161122379g3xxxxxx
Log view:
http://logview.odps.aliyun.com/logview/?h=http://service.odps.aliyun.
com/api&p=alian&i=201709251611223xxxxxxdqp&token=XXXXXXvbiI6IjEifQ==
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 <PY | JAR> <localfile[.py |.jar]> [COMMENT 'cmt'][-f];
```

参数说明：

- file/archive/table/jar/py：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限制项汇总。

删除资源

命令格式如下:

```
DROP RESOURCE <resource_name>; --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 <resource_name> <path>;
```

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。

如果您想对函数进行操作，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理模块。

注册函数

命令格式如下：

```
CREATE FUNCTION <function_name> AS <package_to_class> USING <
resource_list>;
```

参数说明：

- function_name：UDF函数名，这个名字就是SQL中引用该函数所使用的名字。
- package_to_class：如果是Java UDF，这个名字就是从顶层包名一直到实现UDF类名的fully qualified class name；如果是Python UDF，这个名字就是Python脚本名.类名。这个名字须用引号引起来。
- 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。
  - 如果需要指定资源所在的project，写法为<project_name>/resources/<resource_name>。

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

- 假设Python UDF MyLower在项目空间test_project的脚本pyudf_test.py中，创建函数my_lower。如下所示：

```
create function my_lower as 'pyudf_test.MyLower' using 'test_project
/resources/pyudf_test.py';
```

- 假设Java UDF类com.aliyun.odps.examples.udf.UDTFResource是在udtfexample1.jar中，同时该函数还依赖file资源file_resource.txt，table资源table_resource1，以及Archive资源test_archive.zip，创建函数 test_udtf。如下所示：

```
create function test_udtf as 'com.aliyun.odps.examples.udf.
UDTFResource' using 'udtfexample1.jar, file_resource.txt, table_reso
urce1,test_archive.zip';
```

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 <function_name>;
```

示例如下：

```
DROP FUNCTION my_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自定义函数。
```

```
LIST 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 SET操作

本文向您介绍如何使用SET命令设置MaxCompute或用户自定义的系统变量，以及如何清除SET命令设置。

SET

命令格式

```
set <KEY>=<VALUE>
```

作用：设置MaxCompute或影响MaxCompute的行为。

MaxCompute支持的`set`命令Value如下所示。

属性名称	属性描述	取值范围
odps.sql.allow.fullscan	设置是否允许对分区表进行全表扫描，False不允许，True为允许。	True（允许）/False（禁止）。
odps.stage.mapper.mem	设置每个map worker的内存大小。	单位M，默认值1024M。
odps.stage.reducer.mem	设置每个reduce worker的内存大小。	单位M，默认值1024M。
odps.stage.joiner.mem	设置每个join worker的内存大小。	单位M，默认值1024M。
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。
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。
odps.stage.reducer.num	修改每个reduce阶段worker数量。	无默认值。
odps.stage.joiner.num	修改每个join阶段worker数量。	无默认值。
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。

属性名称	属性描述	取值范围
odps.sql.reshuffle.dynamicpt	避免拆分动态分区时产生过多小文件。如果生成的动态分区个数很少，建议将值设为False，以避免数据倾斜。	True/False，默认值为True。
odps.sql.type.system.odps2	SQL操作中涉及到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、DECIMAL）时，默认为False，需要设置为True。此设置为Session级别的设置。	True（允许）/False（禁止），默认为False。
odps.sql.hive.compatible	打开Hive兼容模式后，MaxCompute才可以支持Hive指定的各种语法，例如inputRecordReader、outputRecordReader、Serde等。此设置为Session级别的设置。	True（允许）/False（禁止），默认为False。

说明:

`set`操作命令仅支持字母小写。Project级别设置请参见SetProject。

`set odps.sql.executionengine.coldata.deep.buffer.size.max`:

- 作用：调整MaxCompute写表过程中为单个复杂类型的列预先申请缓存大小，以便提高写入性能。
- 使用场景
  1. 输出的表中含有的复杂数据类型过多。
  2. 输出的表中含有的某个单独的复杂类型变量Size过大。
- 使用说明
  - 默认值为67108864（64M），默认单位为Byte。
  - 如果输出的表有3个列的Schema是复杂类型，例如列类型为（STRING、MAP、STRUCT、ARRAY、BINARY），则默认情况下MaxCompute将

会为写表操作预留64M*3大小的内存。为每一列预先申请的缓存将会用来存放这一列`batch row count`行的数据。

- 如果预先可知道表的复杂类型变量占用的空间比较小，则建议调小此值。例如，如果每个复杂类型变量大小不会超过1024 Byte，同时`batch row count`值使用的是默认值（1024），则可以将Flag设置为1024*1024，如下所示。

```
set odps.sql.executionengine.coldata.deep.buffer.size.max=1048576;
```

- 如果您预先知道每个复杂类型的值都在7M到8M之间，同时指定了`batch row count`为32，则这个值可以被调整为8M*32。
- 如果任务的输出带有复杂类型，或者任务的`mapjoin`小表带有复杂类型，这个值的调整会影响到任务执行过程中使用的内存。根据前面的计算方法，值设的过大有可能导致任务OOM内存溢出。

`set odps.stage.mapper.split.size`:

- 作用：等同于`odps.sql.mapper.split.size`，用于调整每个`mapper`读取数据的大小，单位MB，示例如下 。

```
set odps.stage.mapper.split.size=256
```

Show Flags

命令格式

```
show flags; --显示set设置的参数。
```

说明:

运行`use project`命令会清除`set`命令设置的配置。

2.8 时区配置操作

本文为您介绍如何配置MaxCompute Project的时区。

时区配置的方式

MaxCompute Project时区默认是中国的东八区，DATETIME、TIMESTAMP、DATE类型字段以及相关时间内置函数都是按东八区进行计算。您可以通过以下两种方式进行时区的配置：

- Session级别：`set odps.sql.timezone=<timezoneid>;`语句需要与计算语句一起提交。

```
set odps.sql.timezone=Asia/Tokyo;
select getdate();
--结果如下。
output:
```

```
+------------+
| _c0        |
+------------+
| 2018-10-30 23:49:50 |
+------------+
```

- Project级别：Project Owner需要执行`setProject odps.sql.timezone=<timezoneid>;`语句。Project的时区一旦被设置，对应相关的时间计算会取设置后的时区，原有的任务数据将会受到影响。所以，请您谨慎考虑是否有必要设置时区。如果必要，建议只对新增的Project进行设置，不建议对已有数据的Project进行设置。

使用限制及注意事项

- 时区配置支持的范围：SQL内置日期函数、UDF、UDT、UDJ、`select transform`都支持获取Project Timezone属性来支持时区。
- 时区支持的格式为Asia/Shanghai这类格式（存在夏令时跳变），不支持GMT+9格式。
- 当SDK时区与Project时区不同时，DATETIME->STRING操作需设置GMT时区。
- 时区配置后，通过DataWorks执行相关SQL时，日期显示某些时间段会存在时间差异，例如1900~1928年的日期时间差异为5分52秒，1900年之前的日期时间差异为9秒。
- 为了保证MaxCompute在多个时区DATETIME类型数据的正确性，MaxCompute服务、Java SDK以及配套的Console将在近期进行版本更新（*-oversea*后缀的Java SDK/Console版本），更新后可能影响MaxCompute中已经存储的早于1928年的DATETIME类型数据的显示。
- 对于非中国东八区的区域，建议您同步更新Java SDK/Console版本，以保证SQL计算结果及Tunnel传输1900-01-01之后范围的数据的准确性和一致性。对于早于1900-01-01的DATETIME数据，SQL的计算显示结果和Tunnel传输数据仍然可能存在343秒的差异。对于新版本SDK/Console之前已经上传的早于1928-01-01的DATETIME数据，在新版本中日期时间会减少352秒。
- 如果继续使用不带有*-oversea*后缀的SDK/Console，SQL计算结果和Tunnel传输数据将存在差异。早于1900-01-01的数据差异为9秒，1900-01-01~1928-01-01的数据差异为352秒。

  说明：
  Java SDK/Console版本更新配置时区不影响DataWorks的时区配置，因此时区会存在差异，需要您对DataWorks中定时任务调度的影响进行计算评估。DataWorks服务器在日本Region的时区是GMT+9，在新加坡Region的时区是GMT+8。
- 通过JDBC连接的第三方客户端需要在客户端设置时区，保证与服务端时区设置的一致性。
- MapReduce支持时区功能。

- Spark支持时区功能。
  - 对于提交到MaxCompute计算集群的任务形式，可自动获取Project的时区。
  - 对于通过yarn-client模式启动（例如spark-shell，spark-sql，pyspark等）的设置，您需要手动配置Driver的启动参数（*spark-defaults.conf*），增加`spark.driver.extraJavaOptions -Duser.timezone=America/Los_Angeles`，`timezone`的值为将要使用的时区。
- PAI支持时区功能。
- Graph支持时区功能。

2.9 其他操作

本文向您介绍如何使用Alias命令读取资源数据和使用计量预估（Cost SQL）命令预估SQL计量信息。

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。

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 = 20150715113033121xxxxxxx
Input:65727592 Bytes
UDF:0
Complexity:1.0
```

3 数据上传下载

3.1 数据上传下载概述

本文为您概述如何将数据上传至MaxCompute或从MaxCompute下载数据，包括服务连接、SDK、工具和数据导入导出、上云等常见操作。

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。

说明:

对于离线数据的同步，推荐您优先使用数据集成。

使用限制

Tunnel Upload命令上传限制

- 上传不设速度限制，上传速度的瓶颈在网络带宽以及服务器性能。
- 重传retry有次数的限制，当重传次数超过限制，就会继续上传下一个block。上传完成后，可以通过`select count(*)`语句，检查是否有数据丢失。
- Tunnel命令不支持上传下载Array、Map和Struct类型的数据。
- Tunnel一个项目支持并发的连接数默认最高为2000个。
- 每个Tunnel的Session在服务端的生命周期为24小时，创建后24小时内均可使用，也可以跨进程/线程共享使用，但是必须保证同一个BlockId没有重复使用。
- 当遇到并发写入时，MaxCompute会根据ACID进行并发写的保障。关于ACID的具体语义，请参见ACID语义。

DataHub上传数据限制

- 每个字段的大小不能超过这个字段本身的限制，详情请参见数据类型。

  说明:

  STRING的长度不能超过8M。
- 上传的过程中，会将多条数据打包成一个Package来进行上传。

TableTunnel SDK接口限制

- BlockId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。

3.2 数据通道服务连接

不同网络环境下，您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。

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配置Endpoint。

3.3 数据上云的工具选择

MaxCompute平台提供丰富的数据上传下载工具，可以广泛应用于各种数据上云的应用场景，本文将介绍三种经典场景中的数据传输工具的选择。

Hadoop数据迁移

您可使用Sqoop和DataWorks进行Hadoop数据迁移。

- 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。
- 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，详情请参见Sqoop工具的介绍。

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，支持的数据库种类丰富，包括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，详情请参见什么是数据传输服务DTS。

日志采集

日志采集时，您可以选用Flume、Fluentd、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。

3.4 工具介绍

MaxCompute平台支持丰富的数据上传和下载工具（其中大部分工具已经在GitHub公开源代码，以开源社区的方式进行维护）。您可以根据实际应用场景，选择合适的工具进行数据的上传和下载。

阿里云数加产品

- DataWorks数据集成（Tunnel通道系列）

  DataWorks数据集成（即数据同步），是阿里巴巴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，详情请参见数据集成概述。

- MaxCompute客户端（Tunnel通道系列）
  - 客户端的安装和基本使用方法请参见客户端介绍。
  - 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（Tunnel通道系列）

  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（Tunnel通道系列）

  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（Tunnel通道系列）

  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 。详情请参见基于Kettle的MaxCompute插件实现数据上云。

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

- Flume（DataHub通道系列）

  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（DataHub通道系列）

  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。详情请参见 Fluentd插件介绍。

- LogStash（DataHub通道系列）

  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。具体示例请参见Logstash + DataHub + MaxCompute/StreamCompute 进行实时数据分析。

- OGG（DataHub通道系列）

  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。详情请参见基于OGG DataHub插件将Oracle数据同步上云。

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

3.5 Tunnel上传下载命令

MaxCompute主要通过Tunnel功能支持数据上传下载命令。本文向您介绍Upload、Show、Resume等MaxCompute Tunnel上传下载命令的使用说明。

Tunnel命令功能

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。

> 说明:
> 使用Tunnel命令过程中遇到的各类问题与最佳实践可参考Tunnel命令相关问题。

- 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天内的日志。

```
tunnel purge 5;
--清理前5天的日志。
```

Tunnel上传下载限制

- Tunnel功能及Tunnel SDK当前不支持外部表操作。您可以通过Tunnel直接上传数据到MaxCompute内部表，或者是通过OSS Python SDK上传到OSS后，在MaxCompute使用外部表做映射。关于外部表详情请参见外部表概述。
- Tunnel命令不支持上传下载ARRAY、MAP和STRUCT类型的数据。
- 每个Tunnel的Session在服务端的生命周期为24小时，创建后24小时内均可使用，也可以跨进程/线程共享使用，但是必须保证同一个BlockId没有重复使用。

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（1MiB＝1024*1024B）。
- -c：指定本地数据文件编码，默认为忽略。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。如果您想指定时间格式到毫秒级别，可以使用tunnel upload -dfp 'yyyy-MM-dd HH:mm:ss.SSS'，有关DateTime数据类型的详情请参见数据类型。
- -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，内容如下所示。

```
shopx,x_id,100
shopy,y_id,200
shopz,z_id
```

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: 20150610xxxxxxxxxx70a002ec60c
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 = 20150610xxxxxxxxxxvc61z5
+-----------+-------------+-------------+-----------+--------+
| 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。

说明:

当遇到并发写入时，MaxCompute会根据ACID进行并发写的保障。关于ACID的具体语义，请参见ACID语义。

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;
20150610xxxxxxxxxxx70a002ec60c  failed  'u --config-file /D:/console
/conf/odps_config.ini --project odpstest_ay52c_ay52 --endpoint http
://service.odps.aliyun.com/api --id UlxxxxxxxxxxxrI1 --key 2m4r3WvTxx
xxxxxxxx0InVke7UkvR d:\data.txt sale_detail/sale_date=201312,region=
hangzhou -s false'
```

说明：

20150610xxxxxxxxxxx70a002ec60c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

修改data.txt文件为如下内容：

```
shopx,x_id,100
shopy,y_id,200
```

修复执行上传数据，如下所示：

```
odps@ project_name>tunnel resume 20150610xxxxxxxxxxx70a002ec60c --
force;
start resume
20150610xxxxxxxxxxx70a002ec60c
Upload session: 20150610xxxxxxxxxxx70a002ec60c
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:

20150610xxxxxxxxxxx70a002ec60c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 = 20150610xxxxxxxxxxa741z5
 +-----------+-------------+-------------+-----------+--------+
 | shop_name | customer_id | total_price | sale_date | region |
 +-----------+-------------+-------------+-----------+--------+
 | shopx     | x_id        | 100.0       | 201312    | hangzhou|
 | shopy     | y_id        | 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，默认为忽略。**
- **-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

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: 20150610xxxxxxxxxx70a002ed0b9
    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：

```
shopx,x_id,100.0
shopy,y_id,200.0
```

下载Instance数据

方法一：使用tunnel download命令将特定Instance的执行结果下载到本地文件。

命令

```
tunnel download instance://<[project_name/]instance_id> <path>
```

参数说明：

- project_name：Instance所在的项目名称。
- instance_id：待下载数据的Instance ID。

示例

```
// 执行一条 select 查询。
odps@ odps_test_project>select * from wc_in;
ID = 20170724071705393ge3csfb8
... ...

// 使用Instance Tunnel Download命令下载执行结果到本地文件。
odps@ odps_test_project>tunnel download instance://2017072407
1705393ge3csfb8 result;
2017-07-24 15:18:47  -  new session: 2017072415184785b6516400090ca8
  total lines: 8
2017-07-24 15:18:47  -  file [0]: [0, 8), result
downloading 8 records into 1 file
2017-07-24 15:18:47  -  file [0] start
2017-07-24 15:18:48  -  file [0] OK. total: 44 bytes
download OK

// 查看结果。
cat result
slkdfj
hellp
apple
tea
peach
apple
tea
teaa
```

方法二：通过配置参数使SQL查询默认采用InstanceTunnel方式输出执行结果。

在MaxCompute Console中打开`use_instance_tunnel`选项后，执行的SELECT query会默认使用InstanceTunnel来下载结果，从而避免在MaxCompute平台获取SQL查询结果时所遇到的获取数据超时和获取数据量受限的问题。打开该配置有以下两种方法：

- **在最新版本的 Console中，*odps_config.ini*文件里已经默认打开此选项，并且`instance_tunnel_max_record`被默认设置成了10000。**

  ```
  # download sql results by instance tunnel
  use_instance_tunnel=true
  # the max records when download sql results by instance tunnel
  instance_tunnel_max_record=10000
  ```

  说明：

  `instance_tunnel_max_record`用来设置通过InstanceTunnel下载SQL查询结果的条数。若不设置该项，则下载条数不受限制。

- **将`set console.sql.result.instancetunnel`设置为true来开启此功能。**

  ```
  // 打开Instance tunnel选项。
  odps@ odps_test_tunnel_project>set console.sql.result.instancetunnel
  =true;
  OK

  // 运行select query。
  ```

```
odps@ odps_test_tunnel_project>select * from wc_in;
ID = 20170724081946458g14csfb8
Log view:
http://logview/xxxxx.....
+------------+
| key        |
+------------+
| slkdfj     |
| hellp      |
| apple      |
| tea        |
| peach      |
| apple      |
| tea        |
| teaa       |
+------------+
A total of 8 records fetched by instance tunnel.
```

说明:

如果用InstanceTunnel的方式来输出SELECT查询结果，系统在最后一行会打印一条提示，此示例中Instance的执行结果共有8条数据。同理，您可以将`set console.sql.result.instancetunnel`设置为false来关闭此功能。

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和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	· 有效位数16位 · 上传支持科学计数法表示 · 下载仅使用数字表示 · 最大值：1.7976931348623157E308 · 最小值：4.9E-324 · 无穷大：Infinity · 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

```
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

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

3.6 批量数据通道SDK介绍

3.6.1 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。

MaxCompute提供的数据上传和下载工具是基于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。
InstanceTunnel	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。

主要接口	描述
InstanceTunnel.DownloadSession	表示一个向MaxCompute Instance中下载数据的会话。但只能用于下载以SELECT关键字开头的用于获取数据的SQL Instance。

说明:

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

3.6.2 TableTunnel

TableTunnel是访问MaxCompute Tunnel服务的入口类，仅支持表数据（非视图）的上传和下载。

TableTunnel接口定义及说明

TableTunnel接口定义如下，详情请参见Java-sdk-doc。

```
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。
- TableTunnel提供创建UploadSession对象和DownloadSession对象的方法。数据的上传和下载分别由`TableTunnel.UploadSession`和`TableTunnel.DownloadSession`这两个会话实现。
- 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，对其它MaxCompute任务（例如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用Session ID来标识。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，无法预测数据的到达会造成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的数据后，再使用一个RecordWriter进行一次性批量写入。
- RecordReader的超时为300s。

3.6.3 InstanceTunnel

您可以通过InstanceTunnel下载以SELECT关键字开头的用于获取数据的SQL Instance。

InstanceTunnel接口定义如下，详情请参见Java-sdk-doc。

```
public class InstanceTunnel{
 public DownloadSession createDownloadSession(String projectName,
String instanceID);
 public DownloadSession createDownloadSession(String projectName,
String instanceID, boolean limitEnabled);
```

```
 public DownloadSession getDownloadSession(String projectName, String
id);
 }
```

参数说明：

- projectName：**Project名**
- instanceID：**Instance ID**

InstanceTunnel约束：虽然InstanceTunnel提供了非常方便的获取Instance执行结果的方式，但是为了保护用户数据安全，此功能也受到了权限约束。

- **使用InstanceTunnel下载数据时，若数据条数不超过10000，则只要对该Instance有Read权限的用户即可使用。此行为与使用Restful API获取查询数据的行为一致。**
- **使用InstanceTunnel下载数据时，若数据条数超过10000，则需要对Instance SQL查询语句中涉及到的所有源表进行权限检查，用户需要具有这些表的Read权限才可使用。**

3.6.4 UploadSession

本文为您介绍UploadSession接口。

UploadSession**接口定义**

```
public class UploadSession {
    UploadSession(Configuration conf, String projectName, String
tableName,
                  String partitionSpec) throws TunnelException;
    UploadSession(Configuration conf, String projectName, String
tableName,
                  String partitionSpec, String uploadId) throws
TunnelException;
    public void commit(Long[] blocks);
    public Long[] getBlockList();
    public String getId();
    public TableSchema getSchema();
    public UploadSession.Status getStatus();
    public Record newRecord();
    public RecordWriter openRecordWriter(long blockId);
    public RecordWriter openRecordWriter(long blockId, boolean
compress);
    public RecordWriter openBufferedWriter();
    public RecordWriter openBufferedWriter(boolean compress);
}
```

UploadSession**接口说明**

- **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。
- 六种状态说明：

1. UNKNOWN：Server端刚创建一个Session时设置的初始值。
2. NORMAL：创建Upload对象成功。
3. CLOSING：当调用complete方法（结束上传）时，服务端会先把状态置为CLOSING。
4. CLOSED：完成结束上传（即把数据移动到结果表所在目录）后。
5. EXPIRED：上传超时。
6. 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。

3.6.5 DownloadSession

本文向您介绍DownloadSession接口。

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
TunnelException
    public String getId()
    public long getRecordCount()
    public TableSchema getSchema()
    public DownloadSession.Status getStatus()
    public RecordReader openRecordReader(long start, long count)
    public RecordReader openRecordReader(long start, long count,
boolean compress)
}
```

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。

3.6.6 TunnelBufferedWriter

本文向您介绍TunnelBufferedWriter接口。

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—TunnelBufferedWriter。

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，这样可以避免连接超时。此外，如果数据上传失败，则会自动进行重试。
- 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），建议设置大于等于64MB，避免服务端产生过多小文件，影响性能。最小可设置为1MB，最大1000MB，通常无需设置，维持默认值64M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（默认设置为从4开始的指数递增的序列），一般情况下不建议调整此回避设置。

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

3.7 批量数据通道SDK示例

3.7.1 示例概述

本文向您推荐使用Tunnel服务地址和TunnelBufferedWriter上传数据。

- 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。

3.7.2 简单上传示例

Tunnel SDK是MaxCompute提供的离线批量数据通道服务，主要提供大批量离线数据上传和下载。Tunnel SDK仅提供每次批量大于等于64MB数据的场景。请使用DataHub实时数据通道处理小于64MB的流式数据场景，以获得更好的性能和体验。

典型的表数据上传流程

1. 创建TableTunnel。
2. 创建UploadSession。
3. 创建RecordWriter，写入Record。
4. 提交上传操作。

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";
    //默认情况下，使用公网进行传输。如果需要通过内网进行数据传输，必须设置
tunnelUrl变量。
    //此处给出华东2经典网络Tunnel Endpoint。
    private static String project = "<your project>";
    private static String table = "<your table name>";
    private static String partition = "<your partition spec>";
    public static void main(String args[]) {
        // 该部分为准备工作，仅需执行一次。
        Account account = new AliyunAccount(accessId, accessKey);
        Odps odps = new Odps(account);
        odps.setEndpoint(odpsUrl);
        odps.setDefaultProject(project);
        try {
            TableTunnel tunnel = new TableTunnel(odps);
            // tunnelUrl设置。
        tunnel.setEndpoint(tunnelUrl);
            // 确定写入分区。
        PartitionSpec partitionSpec = new PartitionSpec(partition);
            // 在服务端的本表本分区上创建一个有效期为24小时的Session。
        // 24小时内，该Session共计可以上传20000个分区（Block）数据。
        // 创建Session的耗时在秒级，且需要在服务端使用部分资源、创建临时目录
等，需要消耗较多的资源。
        // 强烈建议同一个分区的数据尽可能通过复用Session的方式上传。
```

```
            UploadSession uploadSession = tunnel.createUploadSession(
project, table, partitionSpec);
            System.out.println("Session Status is : " + uploadSession.
getStatus().toString());
            TableSchema schema = uploadSession.getSchema();
        // 数据准备完成后，打开Writer开始写入数据，将数据写入一个Block。
        // 每个Block仅能成功上传一次，不可重复上传。CloseWriter执行成功即代表该
Block上传成功，如果失败可重新上传该Block。
        // 同一个Session最多可以包含20000个BlockId（即0-19999）。如果超出
20000个需要执行Commit Session并重新创新一个新的Session。
        // 单个Block内如果写入的数据过少将产生大量小文件，严重影响计算性能。强烈
建议每次写入64MB以上的数据（同一Block支持写入100GB以内的数据）。
        // 如果创建一个Session后仅上传少量数据，不仅会造成小文件、空目录等问
题，还会严重影响上传性能（创建Session耗时数秒，而真正数据上传可能仅花费十几毫秒）。
            // Writer创建后任意一段时间内，如果任意连续两分钟内没有写入4KB以上的数
据，将会超时并自动断开连接。
            // 建议在创建Writer前，在内存中准备好可以直接写入的数据。
            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++) {
        // 数据写入服务端时，每写入8KB的数据将进行一次网络传输。
        // 如果120秒内无网络传输，服务端将自动关闭连接，该Writer将不可用，需要重
新创建Writer执行数据写入。
                recordWriter.write(record);
            }
            recordWriter.close();
            uploadSession.commit(new Long[]{0L});
            System.out.println("upload success!");
        } catch (TunnelException e) {
            // 建议重试一定次数。
            e.printStackTrace();
        } catch (IOException e) {
            // 建议重试一定次数。
            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，例如pt=’1’，ds=’2’。

因此程序中应该配置如下：

```
private static String partition = “pt=’XXX’,ds=’XXX’”;
```

说明：

文中使用的是华东2经典网络Tunnel Endpoint，其他Region的Tunnel Endpoint设置可以参考文档配置Endpoint。

3.7.3 简单下载示例

本文为您介绍如何使用MaxCompute Java SDK实现数据下载。

使用TableTunnel的DownloadSession接口实现数据下载

典型的表数据下载流程：

1. 创建TableTunnel。
2. 创建DownloadSession。
3. 创建RecordReader，读取Record。

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。
     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);//设置tunnelUrl。
        PartitionSpec partitionSpec = new PartitionSpec(partition);
          try {
                DownloadSession downloadSession = tunnel.createDown
loadSession(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();
        }
 }
```

说明:

文中给出的是华东2经典网络Tunnel Endpoint，其他region的Tunnel Endpoint设置可以参考文档配置Endpoint。

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。

使用InstanceTunnel的DownloadSession接口实现数据下载

```
Odps odps = OdpsUtils.newDefaultOdps(); // 初始化Odps对象。
    Instance i = SQLTask.run(odps, "select * from wc_in;");
    i.waitForSuccess();
    //创建InstanceTunnel。
    InstanceTunnel tunnel = new InstanceTunnel(odps);
    //根据instance id，创建DownloadSession。
    InstanceTunnel.DownloadSession session = tunnel.createDown
loadSession(odps.getDefaultProject(), i.getId());
    long count = session.getRecordCount();
    //输出结果条数。
    System.out.println(count);
    //获取数据的写法与TableTunnel一样。
    TunnelRecordReader reader = session.openRecordReader(0, count);
    Record record;
    while ((record = reader.read()) != null) {
      for (int col = 0; col < session.getSchema().getColumns().size();
 ++col) {
        //wc_in表字段均为 STRING，这里直接打印输出。
        System.out.println(record.get(col));
      }
    }
    reader.close();
```

通过使用SQLTask.getResultSet()静态方法实现数据下载

```
Odps odps = OdpsUtils.newDefaultOdps(); //初始化Odps对象。
    Instance i = SQLTask.run(odps, "select * from wc_in;");
    i.waitForSuccess();
    //根据instance对象，获取结果迭代器。
    ResultSet rs = SQLTask.getResultSet(i);
```

```
    for (Record r : rs) {
      //输出结果条数。
      System.out.println(rs.getRecordCount());
      for (int col = 0; col < rs.getTableSchema().getColumns().size();
 ++col) {
        //wc_in表字段均为STRING，这里直接打印输出。
        System.out.println(r.get(col));
      }
    }
```

3.7.4 多线程上传示例

本文通过代码示例向您介绍如何使用TableTunnel接口实现多线程上传。

```
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，支持指定设置或者不指定。设置

- **如果指定，按照指定的Endpoint路由。**
- **如果不指定，默认为公网。**
- **文中给出的是华东2经典网络Tunnel Endpoint，其他region的Tunnel Endpoint设置可以参考文档访问域名和数据中心。**

3.7.5 多线程下载示例

本文通过代码示例向您介绍如何使用TableTunnel接口实现多线程下载。

```
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请参见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.createDownloadSession(project,
table, partitionSpec);
            System.out.println("Session Status is : " + downloadSe
ssion.getStatus().toString());
            long count = downloadSession.getRecordCount();
            System.out.println("RecordCount is: " + count);
            ExecutorService pool = Executors.newFixedThreadPool(
threadNum);
            ArrayList<Callable<Long>> callers = new ArrayList<Callable
<Long>>();
            long start = 0;
            long step = count / threadNum;
            for (int i = 0; i < threadNum - 1; i++) {
                RecordReader recordReader = downloadSession.openRecord
Reader(step * i, step);
                callers.add(new DownloadThread(i, recordReader,
downloadSession.getSchema()));
            }
            RecordReader recordReader = downloadSession.openRecord
Reader(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。**
- **文中给出的是华东2经典网络Tunnel Endpoint，其他region的Tunnel Endpoint设置可以参考文档访问域名和数据中心。**

3.7.6 BufferedWriter上传示例

本文通过代码示例向您介绍如何使用BufferedWriter接口实现数据上传。

```
// 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();
```

3.7.7 BufferedWriter多线程上传示例

本文通过代码示例向您介绍如何使用BufferedWriter接口实现多线程上传。

```
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();
}
```

3.7.8 复杂数据类型上传下载示例

本文为您介绍如何通过Tunnel SDK上传或下载复杂数据类型。

MaxCompute支持ARRAY、MAP和STRUCT这三种复杂数据类型，具体用法请参见复杂数据类型。

上传复杂类型数据

代码示例

```
RecordWriter recordWriter = uploadSession.openRecordWriter(0);
ArrayRecord record = (ArrayRecord) uploadSession.newRecord();
//准备数据。
List arrayData = Arrays.asList(1, 2, 3);
Map<String, Long> mapData = new HashMap<String, Long>();
mapData.put("a", 1L);
mapData.put("c", 2L);
List<Object> structData = new ArrayList<Object>();
structData.add("Lily");
structData.add(18);
//将数据传入record。
record.setArray(0, arrayData);
record.setMap(1, mapData);
record.setStruct(2, new SimpleStruct((StructTypeInfo) schema.getColumn
(2).getTypeInfo(), structData));
//写入record。
recordWriter.write(record);
```

下载复杂类型数据

代码示例

```
RecordReader recordReader = downloadSession.openRecordReader(0, 1);
//读取record。
ArrayRecord record1 = (ArrayRecord)recordReader.read();
//获取array数据。
List field0 = record1.getArray(0);
```

```
List<Long> longField0 = record1.getArray(Long.class, 0);
//获取map数据。
Map field1 = record1.getMap(1);
Map<String, Long> typedField1 = record1.getMap(String.class, Long.
class, 1);
//获取struct数据。
Struct field2 = record1.getStruct(2);
```

上传下载运行实例

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
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.ArrayRecord;
import com.aliyun.odps.data.RecordReader;
import com.aliyun.odps.data.RecordWriter;
import com.aliyun.odps.data.SimpleStruct;
import com.aliyun.odps.data.Struct;
import com.aliyun.odps.tunnel.TableTunnel;
import com.aliyun.odps.tunnel.TableTunnel.UploadSession;
import com.aliyun.odps.tunnel.TableTunnel.DownloadSession;
import com.aliyun.odps.tunnel.TunnelException;
import com.aliyun.odps.type.StructTypeInfo;
public class TunnelComplexTypeSample {
  private static String accessId = "<your access id>";
  private static String accessKey = "<your access Key>";
  private static String odpsUrl = "<your odps endpoint>";
  private static String project = "<your project>";
  private static String table = "<your table name>";
  //一个分区表里的分区，例如："pt=\'1\',ds=\'2\'"。
  //如果该表不是一个分区表，则不需要以下语句。
  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);
      PartitionSpec partitionSpec = new PartitionSpec(partition);
      //---------- 上传数据 ---------------
      //为表创建上传会话。
      //表schema为{"col0": ARRAY<BIGINT>, "col1": MAP<STRING, BIGINT>,
 "col2": STRUCT<name:STRING,age:BIGINT>}。
      UploadSession uploadSession = tunnel.createUploadSession(project
, table, partitionSpec);
      //获取表schema。
      TableSchema schema = uploadSession.getSchema();
      //开启record writer。
      RecordWriter recordWriter = uploadSession.openRecordWriter(0);
      ArrayRecord record = (ArrayRecord) uploadSession.newRecord();
      //准备数据。
      List arrayData = Arrays.asList(1, 2, 3);
      Map<String, Long> mapData = new HashMap<String, Long>();
      mapData.put("a", 1L);
```

```
        mapData.put("c", 2L);
        List<Object> structData = new ArrayList<Object>();
        structData.add("Lily");
        structData.add(18);
        //将数据传入record。
        record.setArray(0, arrayData);
        record.setMap(1, mapData);
        record.setStruct(2, new SimpleStruct((StructTypeInfo) schema.
getColumn(2).getTypeInfo(), structData));
        //写入record。
        recordWriter.write(record);
        //关闭writer。
        recordWriter.close();
        //对uploadSession进行commit操作，上传结束。
        uploadSession.commit(new Long[]{0L});
        System.out.println("upload success!");
        //---------- 下载数据 ---------------
        //为表创建下载会话。
        //表schema为{"col0": ARRAY<BIGINT>, "col1": MAP<STRING, BIGINT>,
 "col2": STRUCT<name:STRING, age:BIGINT>}。
        DownloadSession downloadSession = tunnel.createDownloadSession(
project, table, partitionSpec);
        schema = downloadSession.getSchema();
        //开启record reader，示例为读取一个record。
        RecordReader recordReader = downloadSession.openRecordReader(0,
1);
        //读取record。
        ArrayRecord record1 = (ArrayRecord)recordReader.read();
        //获取array数据。
        List field0 = record1.getArray(0);
        List<Long> longField0 = record1.getArray(Long.class, 0);
        //获取map数据。
        Map field1 = record1.getMap(1);
        Map<String, Long> typedField1 = record1.getMap(String.class,
Long.class, 1);
        //获取struct数据。
        Struct field2 = record1.getStruct(2);
        System.out.println("download success!");
      } catch (TunnelException e) {
        e.printStackTrace();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
}
```

3.8 数据集成导入导出数据

本文向您介绍如何通过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。

详细信息请参见数据集成。

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。

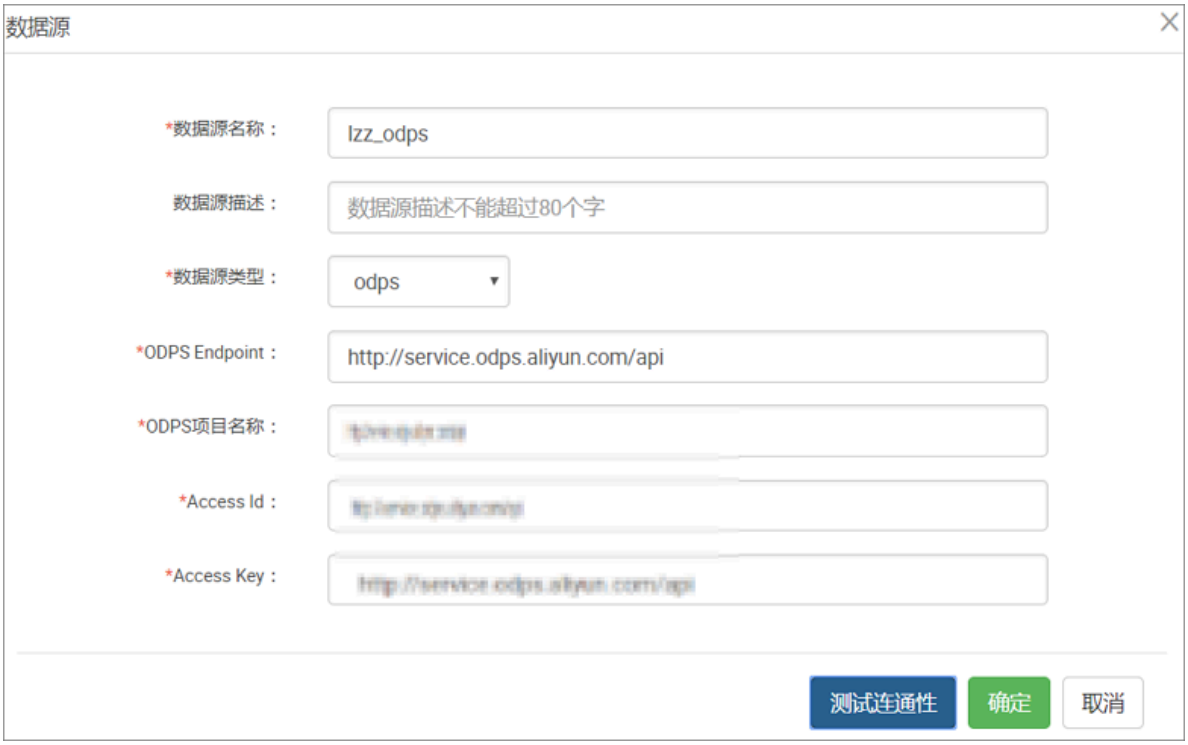
添加MaxCompute数据源

> **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。



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ODPS。
- 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AccessKeyID。
- Access Key： 与MaxCompute Project Owner云账号对应的AccessKeySecret，与AccessKeyID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。**

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。

3. 选择目标。

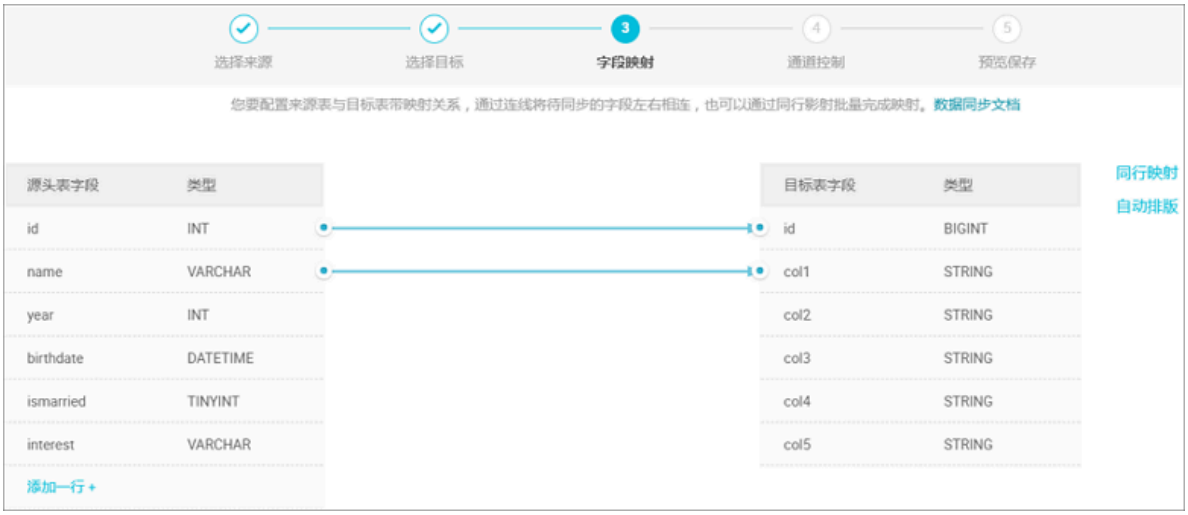
目标即MaxCompute（原ODPS），表可以是提前创建，也可以在此处单击快速建表。

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。

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。

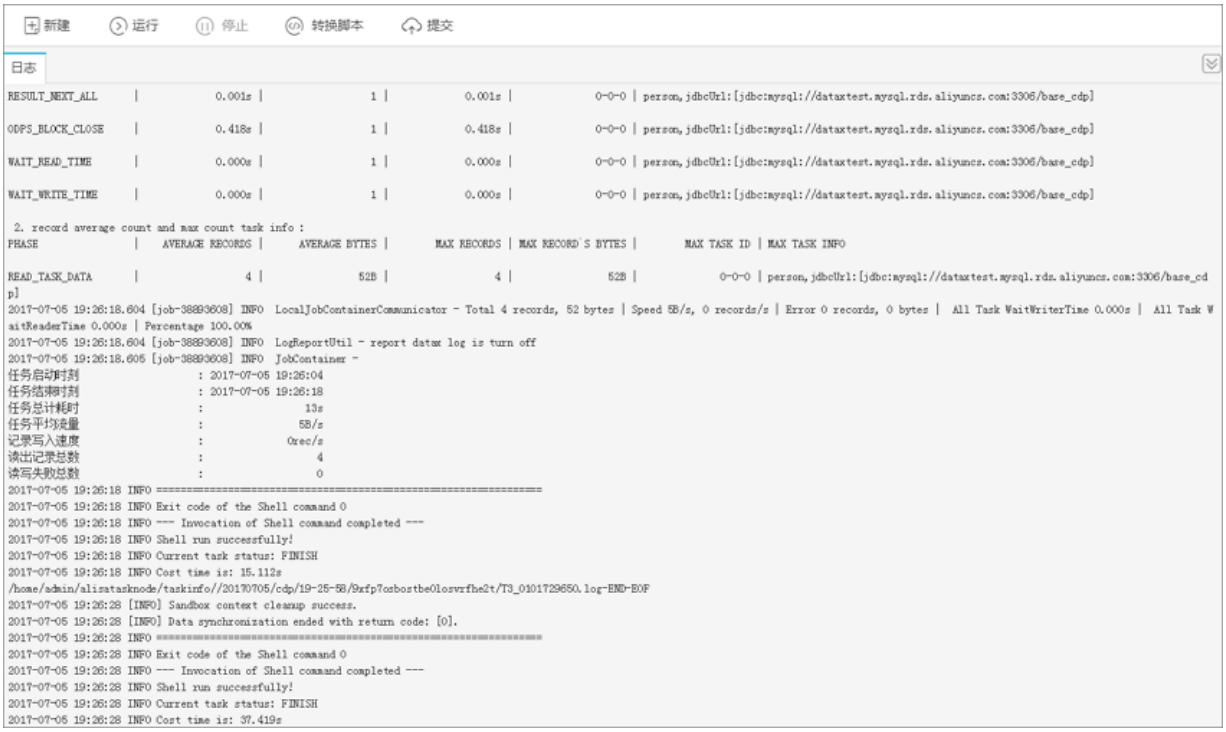
完成上述配置后，上下滚动鼠标可查看任务配置，如若无误，单击保存。

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。

查看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。

3.9 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。

DataHub同样提供流式数据归档的功能，支持流式数据归档至MaxCompute。DataHub实时数据通道的详情请参见DataHub文档。

DataHub提供了Java和Python两种语言的SDK，可供您使用。详情请参见下述文档：

- DataHub Java SDK介绍。
- DataHub Python SDK介绍。

4 SQL及函数

4.1 SQL概述

本文为您介绍MaxCompute SQL的关键字、类型转换说明、分区表、UNION ALL运算及使用限制。

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。它的语法是标准语法ANSI SQL92的一个子集，并有自己的扩展，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语法的差异。目前在MaxCompute中允许的最大SQL长度是3MB。

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。

使用限制

SQL限制项请参见SQL限制项，不支持的DDL及DML语法请参见与其他SQL语法的差异。

> 说明:
> 与其他SQL的差异和不兼容带来的限制，可参见与标准SQL的主要区别及解决方法及修改不兼容SQL实战解决。

其他限制：

- SCALAR SUBQUERY限制
- Insert values限制：values必须是常量
- MaxCompute最多允许256个表的union all/union
- Mapjoin的小表不能超过512MB
- 由于国际标准化组织发布的中国时区信息调整，执行相关SQL时，日期显示某些时间段会存在时间差异：1900-1928年的日期时间差异5分52秒，1900年之前的日期时间差异9秒。

4.2 运算符

运算符用于执行程序代码运算，本文向您介绍四种类型运算符：关系运算符、算术运算符、位运算符、逻辑运算符。

关系运算符

运算符	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（_）要用转义符表示（‘%’）或（‘_’）。 ```‘aaa’ like‘a__’= TRUE ‘aaa’ like‘a%’ = TRUE ‘aaa’ like‘aab’= FALSE ‘a%b’ like‘a\%b’= TRUE ‘axb’ like ‘a\%b’= FALSE```
A RLIKE B	A是字符串，B是字符串常量正则表达式。如果匹配成功，返回TRUE，否则返回FALSE。如果B为空串会报错退出。如果A或B为NULL，返回NULL。
A IN B	· B为一个集合，如果A在B中则返回TRUE，否则返回FALSE。 · 如果A为NULL，返回NULL。 · 如果B仅包含一个NULL元素，即A IN（NULL），则返回NULL；如果B集合中包含NULL元素（也包含其它元素），则将NULL视为B集合中其它元素的类型。 · B必须是常数集合且至少包含一个项，且所有项的数据类型需一致。

运算符	说明
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%';
```

在进行部分关系运算之前，您需要首先进行类型转换，否则可能返回null。如下举例，'2019-02-16 00:00:01'为DATETIME类型，而'2019-02-16'为STRING类型，在进行比较关系运算前需首先完成显示类型转换。

```
select cast('2019-02-16 00:00:01' AS string) > '2019-02-16';
select cast('2019-02-16 00:00:02' AS datetime) > '2019-02-16 00:00:01
';
```

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。
- 当将STRING和BIGINT类型数据进行数值比较时，两个数据都将自动转换成DOUBLE类型，因此在比较过程中可能存在精度丢失现象。这种情况下，您可以通过cast string as bigint的方式将STRING数值转换为BIGINT类型后再进行比较。

算术运算符

运算符	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。

运算符	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。 说明: 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。

位运算符

运算符	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。

逻辑运算符

运算符	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:

逻辑运算符只允许BOOLEAN类型参与运算，不支持隐式类型转换。

4.3 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	DATETIM	BOOLEA	DECIMA	FLOAT
BIGINT	–	Y	Y	N	Y	Y	Y
DOUBLE	Y	–	Y	N	Y	Y	Y
STRING	Y	Y	–	Y	Y	Y	Y
DATETIM	N	N	Y	–	N	N	N
BOOLEA	Y	Y	Y	N	–	Y	Y
DECIMA	Y	Y	Y	N	Y	–	Y
FLOAT	Y	Y	Y	N	Y	Y	-

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;
select cast(array(1,2,3) as array<STRING>);
select concat_ws(',', cast(array(1, 2) as array<STRING>));
```

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地类型转换，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的介绍请参见其他函数。

- 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。
- MaxCompute支持复杂类型的类型转换功能。其中复杂类型的隐式类型转换要求子类型能够隐式转换，而显示转换要求子类型能够显示转换。Struct类型转换不要求字段名称一致，但是要求字段的数量一致，且对应序号的字段能够隐式或显示转换。例如：
  - `array<bigINT>`能隐式转换或显示转换为`array<STRING>`。
  - `array<bigINT>`能显示转换为`array<INT>`，但是不能隐式转换。
  - `array<bigINT>`不能隐式转换或显示转换为 `array<datetime>`。
  - `struct<a:bigINT,b:INT>`能隐式转换为 `struct<col1:STRING,col2:bigINT>`，但是不能隐式或显示转换为`struct<a:STRING>`。

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：

From/To	BOO	TINY	SMA	INT	BIGI	FLOA	DOUBLE	DECIMAL	STRI	VARC	TIMI	BINARY
BOOLEAN to	Y	N	N	N	N	N	N	N	N	N	N	N
TINYINT to	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N
SMALLINT to	N	N	Y	Y	Y	Y	Y	Y	Y	Y	N	N
INT to	N	N	Y	Y	Y	Y	Y	Y	Y	Y	N	–
BIGINT to	N	N	N	N	Y	Y	Y	Y	Y	Y	N	N
FLOAT to	N	N	N	N	Y	Y	Y	Y	Y	Y	N	–
DOUBLE to	N	N	N	N	N	N	Y	Y	Y	Y	N	N
DECIMAL to	N	N	N	N	N	N	N	Y	Y	Y	N	N
STRING to	N	N	N	N	N	N	Y	Y	Y	Y	N	N
VARCHAR to	N	N	N	N	Y	Y	Y	Y	N	N	–	–

From/ To	BOO	TINY	SMA	INT	BIGI	FLOA	DOUBLE	DECIMAL	STRI	VARC	TIMI	BINARY
TIMESTAMP to	N	N	N	N	N	N	N	N	Y	Y	Y	N
BINARY to	N	N	N	N	N	N	N	N	N	N	N	Y

其中，Y表示可以转换，N表示不可以转换，– 表示不需要转换。

说明:

- 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，这个转换潜在有数据丢失风险，一般数据库系统都不允许，建议您谨慎操作。

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+、-、*、/、%，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)
```

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。

4.4 SQL限制项

本文为您介绍MaxCompute SQL的限制项。

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、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	进行MULTI-INSERT时，同时插入的数据表数量限制为256个。
UNION ALL	256个表	数量限制	最多允许256个表的UNION ALL。
MAPJOIN	128个小表	数量限制	MAPJOIN最多允许128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列在子查询中时，子查询的返回结果不可以超过1000行。
SQL语句长度	2M	长度限制	允许的SQL语句的最大长度。

边界名	最大值/限制条件	分类	说明
WHERE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	表中单个单元的最大长度。
IN的参数个数	1024	数量限制	IN的最大参数限制，例如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	视图不支持写入，不支持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	不允许为ABSTRACT或者STATIC。	操作限制	Java UDF函数不能是ABSTRACT或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限制项均无法修改。

4.5 INSERT语句

4.5.1 更新表数据（INSERT OVERWRITE and INTO）

本文向您介绍如何使用INSERT OVERWRITE和INSERT INTO两种命令更新表数据。

INSERT操作

命令格式

```
INSERT OVERWRITE|INTO TABLE tablename [PARTITION (partcol1=val1,
partcol2=val2 ...)] [(col1,col2 ...)]
select_statement
FROM from_statement;
```

说明:

- MaxCompute的insert语法与通常使用的MySQL或Oracle的insert语法有差别，在insert overwrite/into后需要加入table关键字，而非直接使用tablename。
- 当insert的目标表是分区表时，指定分区值[PARTITION (partcol1=val1, partcol2=val2 …)]语法中不允许使用函数等表达式。
- 目前insert overwrite不支持指定插入列的功能，暂时只能用insert into。
- 不支持insert into到Hash Clustering表。
- 当遇到并发写入时，MaxCompute会根据ACID进行并发写的保障。关于ACID的具体语义，请参见ACID语义。

在MaxCompute SQL处理数据的过程中，insert overwrite/into用于将计算的结果保存目标表中。

insert into与insert overwrite的区别是：insert into会向表或表的分区中追加数据，而insert overwrite会在向表或分区中插入数据前清空表中的原有数据。

说明:

在反复对同一个分区进行insert overwrite的时候，通过describe查看到的数据分区Size会不同。这是因为从同一个表的同一个分区select出来再insert overwrite回相同分区时，文件切分逻辑发生变化，从而导致数据的Size发生变化。数据的总长度在insert overwrite前后是不变的，您不必担心存储计费会存在问题。

在使用MaxCompute处理数据的过程中，insert overwrite/into是最常用到的语句，它们会将计算的结果保存到一个表中，以供下一步计算使用。例如计算sale_detail表中不同地区的销售额，操作如下。

```
create table sale_detail_insert like sale_detail;
alter table sale_detail_insert add partition(sale_date='2013', region
='china');
insert overwrite table sale_detail_insert partition (sale_date='2013
', region='china')
select shop_name, customer_id,total_price from sale_detail;
```

说明:

在进行insert更新表数据时，源表与目标表的对应关系依赖于在select子句中列的顺序，而不是表与表之间列名的对应关系。以下为合法的SQL语句。

```
insert overwrite table sale_detail_insert partition (sale_date='2013
', region='china')
select customer_id, shop_name, total_price from sale_detail;
-- 在创建sale_detail_insert表时，列的顺序为：
-- shop_name string, customer_id string, total_price bigint
-- 而从sale_detail向sale_detail_insert插入数据是，sale_detail的插入顺序为：
```

```
-- customer_id, shop_name, total_price
-- 此时，会将sale_detail.customer_id的数据插入sale_detail_insert.
shop_name。
-- 将sale_detail.shop_name的数据插入sale_detail_insert.customer_id。
```

向某个分区插入数据时，分区列不允许出现在select列表中。

```
insert overwrite table sale_detail_insert partition (sale_date='2013
', region='china')
select shop_name, customer_id, total_price, sale_date, region  from
sale_detail;
-- 报错返回，sale_date，region为分区列，不允许出现在静态分区的insert语句中。
```

同时，partition的值只能是常量，不可以出现表达式。以下为错误用法。

```
insert overwrite table sale_detail_insert partition (sale_date=
datepart('2016-09-18 01:10:00', 'yyyy') , region='china')
select shop_name, customer_id, total_price from sale_detail;
```

更新表数据到动态分区

动态分区使用注意事项：

- insert into partition时，如果分区不存在，会自动创建分区。
- 多个insert into partition作业并发时，如果分区不存在，会自动创建分区，但只会成功创建一个分区。
- 如果不能控制insert into partition作业并发，则只能通过预创建分区避免问题。

更多动态分区详情请参考输出到动态分区（DYNAMIC PARTITION）。

4.5.2 多路输出（MULTI INSERT）

MaxCompute SQL支持在一个语句中插入不同的结果表或者分区实现多路输出。

命令格式如下。

```
FROM from_statement
INSERT OVERWRITE | INTO TABLE tablename1 [PARTITION (partcol1=val1,
partcol2=val2 ...)]
select_statement1 [FROM from_statement]
[INSERT OVERWRITE | INTO TABLE tablename2 [PARTITION (partcol1=val3,
partcol2=val4 ...)]
select_statement2 [FROM from_statement]]
```

说明：

- 一般情况下，单个SQL中最多可以写128路输出。超过128路，则报语法错误。
- 在一个multi insert中：
  - 对于分区表，同一个目标分区不允许出现多次。
  - 对于未分区表，该表不能出现多次。

- 对于同一张分区表的不同分区，不能同时有insert overwrite和insert into操作，否则报错返回。

示例如下。

```
create table sale_detail_multi like sale_detail;
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2010
', region='china' )
select shop_name, customer_id, total_price where .....
insert overwrite table sale_detail_multi partition (sale_date='2011
', region='china' )
select shop_name, customer_id, total_price where .....;
-- 成功返回，将sale_detail的数据插入到sales里的2010年及2011年中国大区的销售记录中。
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2010
', region='china' )
select shop_name, customer_id, total_price
insert overwrite table sale_detail_multi partition (sale_date='2010
', region='china' )
select shop_name, customer_id, total_price;
-- 出错返回，同一分区出现多次。
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2010
', region='china' )
select shop_name, customer_id, total_price
insert into table sale_detail_multi partition (sale_date='2011',
region='china' )
select shop_name, customer_id, total_price;
-- 出错返回，同一张表的不同分区，不能同时有insert overwrite和insert into操作。
```

4.5.3 输出到动态分区（DYNAMIC PARTITION）

在使用INSERT OVERWRITE语句插入到一张分区表时，您可以在语句中指定分区的值，也可以在分区中指定一个分区列名（但不提供具体的值），并在SELECT子句中提供对应分区列的值。

动态分区语法

命令格式如下。

```
insert overwrite table tablename partition (partcol1, partcol2 ...)
select_statement from from_statement;
```

说明:

- select_statement字段中，后面的字段将提供目标表动态分区值。如果目标表只有一级动态分区，则select_statement最后一个字段值即为目标表的动态分区值。
- 在使用动态分区功能的SQL中，在分布式环境下，单个进程最多只能输出512个动态分区，否则会引发运行时异常。
- 任意动态分区SQL不允许生成超过2000个动态分区，否则会引发运行时异常。

- 动态生成的分区值不允许为NULL，也不支持含特殊字符和中文，否则会引发运行时异常`FAILED: ODPS-0123031:Partition exception - invalid dynamic partition value: province=xxx`。
- 如果目标表有多级分区，在运行`insert`语句时允许指定部分分区为静态，但是静态分区必须是高级分区。
- 如果目标表为Hash Clustering Table，则不支持动态分区。

动态分区示例

动态分区的示例如下。

```
create table total_revenues (revenue bigint) partitioned by (region
string);
insert overwrite table total_revenues partition(region)
select total_price as revenue, region
from sale_detail;
```

按照上述写法，在SQL运行之前，您无法得知会产生哪些分区。只有在`select`运行结束后，才能通过`region`字段产生的值确定会产生哪些分区。

其它示例如下。

```
create table sale_detail_dypart like sale_detail; --创建示例目标表。
```

- 示例一

```
insert overwrite table sale_detail_dypart partition (sale_date,
region)
select shop_name,customer_id,total_price,sale_date,region from
sale_detail;
-- 成功返回。
```

  - 此时`sale_detail`表中，`sale_date`的值决定目标表的`sale_date`分区值，`region`的值决定目标表的`region`分区值。
  - 动态分区中，`select_statement`字段和目标表动态分区的对应是按字段顺序决定的。上述示例中，`select`语句如果被写成`select shop_name,customer_id,total_price, region,sale_date from sale_detail;`，则`sale_detail`表中，`region`值决定目标表的`sale_date`分区值，`sale_date`的值决定目标表的`region`分区值。

- 示例二

```
insert overwrite table sale_detail_dypart partition (sale_date='2013
', region)
select shop_name,customer_id,total_price,region from sale_detail;
```

```
-- 成功返回，多级分区，指定一级分区。
```

- 示例三

```
insert overwrite table sale_detail_dypart partition (sale_date='2013
', region)
select shop_name,customer_id,total_price from sale_detail;
-- 失败返回，动态分区插入时，动态分区列必须在select列表中。
```

- 示例四

```
insert overwrite table sales partition (region='china', sale_date)
select shop_name,customer_id,total_price,region from sale_detail;
-- 失败返回，不能仅指定低级子分区，而动态插入高级分区。
```

旧版MaxCompute在进行动态分区时，如果分区列的类型与对应`select`列表中列的类型不严格一致，会发生报错。MaxCompute 2.0则支持隐式类型转换，示例如下。

```
create table parttable(a int, b double) partitioned by (p string);
insert into parttable partition(p) select key, value, current_ti
mestmap() from src;
select * from parttable;
```

执行上述语句后返回结果如下。

a	b	p
0	NULL	2017-01-23 22:30:47.130406621
0	NULL	2017-01-23 22:30:47.130406621

说明:

如果您的数据是有序的，动态分区插入会把数据随机打散，导致压缩率较低。推荐您使用Tunnel命令上传数据到动态分区，以获取较好的压缩率。

使用该命令的详细示例请参见RDS迁移至MaxCompute实现动态分区。

4.6 DDL语句

4.6.1 表操作

本文为您介绍如何通过客户端创建、查看、删除、重命名和修改表信息。

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], ...)]
```

```
[CLUSTERED BY (col_name [, col_name, ...]) [SORTED BY (col_name [
ASC | DESC] [, col_name [ASC | DESC] ...])] INTO number_of_buckets
BUCKETS] -- 用于创建Hash Clustering表时设置表的Shuffle和Sort属性。
[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
```

- 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：
  > 目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用`set`命令开启：
  >
  > - Session级别：要使用新数据类型，需在SQL语句前加上`set`语句`set odps.sql.type.system.odps2=true;`，并与SQL语句一起提交执行。
  > - Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：
  >
  >   ```
  >   setproject odps.sql.type.system.odps2=true;
  >   ```
  >
  >   对`setproject`的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

- `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。
- `clustered by`指定Hash Key。MaxCompute将对指定列进行Hash运算，按照Hash值分散到各个Bucket里面。
  - 为避免数据倾斜和热点，取得较好的并行执行效果，`clustered by`列适宜选择取值范围大，重复键值少的列。此外，为了达到`join`优化的目的，也应该考虑选取常用的Join/Aggregation Key，即类似于传统数据库中的主键。
  - `sorted by`用于指定在Bucket内字段的排序方式。建议`sorted by`和`clustered by`一致，以取得较好的性能。此外，当`sorted by`子句指定之后，MaxCompute将自动生成索引，并且在查询的时候利用索引来加快执行。
  - INTO number_of_buckets BUCKETS指定了哈希桶的数目。这个数字必须填写，且由数据量大小来决定。此外，缺省条件下MaxCompute只能支持最多1111个Reducer，所以此处最多也只支持1111个哈希桶。您可以使用`set odps.sql.reducer.instances=xxx;`来提升这个限制，但最大不得超过4000，否则会影响性能。

    选择哈希桶数目时，请您遵循以下两个原则：

    ■ 哈希桶大小适中：经验值是每个Bucket的大小在500M左右比较合理。例如，分区大小估计为500G，初略估算Bucket数目应该设为1000，这样平均每个Bucket大小约

为500M。对于特别大的表，500M的限制可以突破，每个Bucket在2-3G左右比较合适。同时，可以结合`set odps.sql.reducer.instances=xxx;`来突破1111个桶的限制。

- 对于需要经常`join`的两个表，哈希桶数目应设为一样，这样才能够优化`join`，省略掉Shuffle和Sort步骤。如果按照上述原则计算两个表的哈希桶数不一致，建议统一使用数字大的Bucket Number，保证合理的并发度和执行效率。

- Hash Clustering表还有以下优点：
  - 优化Bucket Pruning。
  - 优化Aggregation。
  - 优化存储。
- Hash Clustering表有以下限制：
  - 不支持`insert into`，只能通过`insert overwrite`来添加数据。
  - 不支持Tunnel直接Upload到Range Cluster表，因为Tunnel上传数据是无序的。

· 有关外部表的更多详情请参见访问OSS非结构化数据。

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，如果表`sale_detail`中存在数据，上面的示例会将表`sale_detail`的数据全部复制到表`sale_detail_ctas1`

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。

创建Hash Clustering表示例

```
CREATE TABLE T1 (a string, b string, c bigint) CLUSTERED BY (c) SORTED
 by (c) INTO 1024 BUCKETS; --创建Hash Clustering非分区表。
CREATE TABLE T1 (a string, b string, c bigint) PARTITIONED BY (dt
string) CLUSTERED BY (c) SORTED by (c) INTO 1024 BUCKETS; --创建Hash
Clustering分区表。
```

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

- 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
```

可见，除生命周期属性外，sale_detail_like的其它属性（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表操作。

说明:

通过describe table查看到的Size包含了在回收站的数据Size。如果您需要清空回收站，可以先执行purge table table_name，然后再执行describe table查看除回收站以外的数据大小。您也可以执行show recyclebin查看本项目中回收站内的数据明细。

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。

- 随着MaxCompute类型系统的不断扩充（参见数据类型），执行desc命令后可以返回的类型也随着增多。要使用MaxCompute新类型，SQL语句都需要配合设置开关新类型打开语句，但desc命令无需打开便可返回新类型。

  说明:

  若对desc table的输出结果有解析依赖，请及时更新适配解析MaxCompute新数据类型。

  示例

```
set odps.sql.type.system.odps2=true;
CREATE TABLE test_newtype (
    c1 tinyint
    ,c2 smallint
    ,c3 int
    ,c4 BIGINT
    ,c5 float
    ,c6 DOUBLE
    ,c7 decimal
    ,c8 binary
    ,c9 timestamp
    ,c10 ARRAY<map<BIGINT,BIGINT>>
    ,c11 map<STRING,ARRAY<BIGINT>>
    ,c12 STRUCT<s1:STRING,s2:BIGINT>
    ,c13 varchar(20))
LIFECYCLE 1
;
```

  desc test_newtype; 返回结果（省略部分结果）如下。

```
| Native Columns:
                |
+-------------------------------------------------------------------------------
+
| Field           | Type       | Label | Comment
                |
+-------------------------------------------------------------------------------
+
| c1              | tinyint    |       |
                |
| c2              | smallint   |       |
                |
| c3              | int        |       |
                |
| c4              | bigint     |       |
                |
| c5              | float      |       |
                |
| c6              | double     |       |
                |
| c7              | decimal    |       |
                |
| c8              | binary     |       |
                |
| c9              | timestamp  |       |
                |
| c10             | array<map<bigint,bigint>> |       |
                |
```

```
| c11              | map<string,array<bigint>> |       |
                  |
| c12              | struct<s1:string,s2:bigint> |       |
                  |
| c13              | varchar(20) |       |
                  |
+-------------------------------------------------------------------------------
+
```

您也可以使用`DESC EXTENDED table_name;`命令查看Hash Clustering Table的Clustering属性，如下所示，Clustering属性将显示在Extended Info中。

```
+-------------------------------------------------------------------------------
+
| Owner: ALIYUN$xxxxxxx@aliyun.com | Project: xxxxx |
| TableComment: |
+-------------------------------------------------------------------------------
+
| CreateTime: 2017-12-25 11:18:26 |
| LastDDLTime: 2017-12-25 11:18:26 |
| LastModifiedTime: 2017-12-25 11:18:26 |
| Lifecycle: 2 |
+-------------------------------------------------------------------------------
+
| InternalTable: YES | Size: 0 |
+-------------------------------------------------------------------------------
+
| Native Columns: |
+-------------------------------------------------------------------------------
+
| Field | Type | Label | Comment |
+-------------------------------------------------------------------------------
+
| a | string | | |
| b | string | | |
| c | bigint | | |
+-------------------------------------------------------------------------------
+
| Partition Columns: |
+-------------------------------------------------------------------------------
+
| dt | string | |
+-------------------------------------------------------------------------------
+
| Extended Info: |
+-------------------------------------------------------------------------------
+
| TableID: 91a3395d3ef64b4d9ee1d28527552864 |
| IsArchived: false |
| PhysicalSize: 0 |
| FileNum: 0 |
| ClusterType: hash |
| BucketNum: 1024 |
| ClusterColumns: [c] |
| SortColumns: [c ASC] |
```

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

对于聚集属性分区表，除了可以使用上述命令查看Table属性，还可以通过以下命令查看分区的属性。

```
DESC EXTENDED table_name partition(pt_spec);
```

使用`select`命令查看表中的详细信息，请参见SELECT语法介绍。

查看建表语句

查看建表语句的语法格式，如下所示。

```
SHOW CREATE TABLE <table_name>;
```

说明:

该命令可生成创建Table的SQL DDL语句，方便您通过SQL重建Schema。

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

```
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

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

修改表Owner

MaxCompute SQL支持通过changeowner命令来修改表的拥有人（表Owner），相应的语法格式如下所示。

```
alter table table_name changeowner to 'ALIYUN$xxx@aliyun.com';
```

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

```
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。

修改表的Hash Clustering属性

对于分区表，我们支持通过alter table语句增加或者去除Hash Clustering属性。

增加表的Hash Clustering属性语句，如下所示。

```
ALTER TABLE table_name
[CLUSTERED BY (col_name [, col_name, ...]) [SORTED BY (col_name [
ASC | DESC] [, col_name [ASC | DESC] ...])] INTO number_of_buckets
BUCKETS]
```

去除表的Hash Clustering属性的语法格式，如下所示。

```
ALTER TABLE table_name NOT CLUSTERED;
```

说明:

- alter table改变聚集属性，只对于分区表有效，非分区表一旦聚集属性建立就无法改变。
- 由于alter table只影响新分区，所以该语句不可以再指定partition alter table语句适用于存量表，在增加了新的聚集属性之后，新的分区将做Hash Cluster存储。

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;
```

4.6.2 生命周期操作

MaxCompute提供了数据生命周期管理功能，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

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。
- 非分区表不支持取消Lifecycle，只能修改Lifecycle。分区表可以取消某个具体分区的Lifecycle。

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

特定情况下，部分特定的分区不希望被生命周期功能自动回收掉，例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;
```

4.6.3 分区和列操作

本文为您介绍如何在MaxCompute进行添加、修改列或添加、删除分区的命令操作。

添加分区操作

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:

- 分区名必须小写。
- 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，则返回报错。
- 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，如果需要添加新的分区，必须指明全部的分区值。

例如为表`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。
```

删除分区操作

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`，则返回报错。

例如从表`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月杭州分区的销售记录。
```

添加列操作

添加列的语法格式，如下所示。

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

同时添加列和注释，如下所示。

```
ALTER TABLE table_name ADD COLUMNS (col_name1 type1 comment 'XXX',
col_name2 type2 comment 'XXX');
```

说明:

添加的新列不支持指定顺序，默认在最后一列。

修改列名操作

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、分区的注释

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:

- 如果table_name或partition_col不存在，则返回报错。
- 指定的partition_col_value不存在，则返回报错。
- 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填写。

4.6.4 视图操作

本文为您介绍如何通过DDL语句创建、删除、重命名视图。

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;
```

4.7 Lateral View

Lateral View和split，explode等UDTF一起使用，它能够将一行数据拆成多行数据，并在此基础上对拆分后的数据进行聚合。

单个Lateral View语句

语法定义如下：

```
lateralView: LATERAL VIEW [OUTER] udtf(expression) tableAlias AS
columnAlias (',' columnAlias) * fromClause: FROM baseTable (lateralVie
w)*
```

说明如下：

- Lateral view outer：当table function不输出任何一行时，对应的输入行在Lateral view结果中依然保留，且所有table function输出列为null。

示例如下：

假设我们有一张表pageAds，它有两列数据，第一列是pageid string，第二列是adid_list，即用逗号分隔的广告ID集合。

string pageid	Array<int> adid_list
“front_page”	[1, 2, 3]
“contact_page”	[3, 4, 5]

需求是要统计所有广告ID在所有页面中出现的次数，实现过程如下所示。

1. 拆分广告ID，如下所示：

```
SELECT pageid, adid
 FROM pageAds LATERAL VIEW explode(adid_list) adTable AS adid;
```

执行结果如下：

string pageid	int adid
“front_page”	1
“front_page”	2

string pageid	int adid
“front_page”	3
“contact_page”	3
“contact_page”	4
“contact_page”	5

2. 进行聚合的统计，语句如下：

```
SELECT adid, count(1)
    FROM pageAds LATERAL VIEW explode(adid_list) adTable AS adid
GROUP BY adid;
```

执行结果如下：

int adid	count(1)
1	1
2	1
3	2
4	1
5	1

多个Lateral View语句

一个from语句后可以跟多个Lateral View语句，后面的Lateral View语句能够引用它前面的所有表和列名。

以下面的表为例：

Array<int> col1	Array<string> col2
[1, 2]	[“a” , “b” , “c”]
[3, 4]	[“d” , “e” , “f”]

· 执行单个语句：

```
SELECT myCol1, col2 FROM baseTable
    LATERAL VIEW explode(col1) myTable1 AS myCol1;
```

执行结果如下所示：

int mycol1	Array<string> col2
1	[“a” , “b” , “c”]
2	[“a” , “b” , “c”]

int mycol1	Array<string> col2
3	[“d” , “e” , “f”]
4	[d” , “e” , “f”]

- 加上一个Lateral View语句，如下所示：

```
SELECT myCol1, myCol2 FROM baseTable
    LATERAL VIEW explode(col1) myTable1 AS myCol1
    LATERAL VIEW explode(col2) myTable2 AS myCol2;
```

执行结果如下所示：

int myCol1	string myCol2
1	“a”
1	“b”
1	“c”
2	“a”
2	“b”
2	“c”
3	“d”
3	“e”
3	“f”
4	“d”
4	“e”
4	“f”

4.8 与其他SQL语法的差异

本文将从SQL角度，将MaxCompute SQL与Hive、MySQL、Oracle、SQL Server进行对比，从而为您介绍MaxCompute不支持的DDL和DML语法。

MaxCompute不支持的DDL语法

语法	MaxCompute	Hive	MySQL	Oracle	SQL Server
CREATE TABLE—PRIMARY KEY	N	N	Y	Y	Y

语法	MaxCompute	Hive	MySQL	Oracle	SQL Server
CREATE TABLE—NOT NULL	N	N	Y	Y	Y
CREATE TABLE—CLUSTER BY	Y	Y	N	Y	Y
CREATE TABLE—EXTERNAL TABLE	Y (OSS, OTS, TDDL)	Y	N	N	N
CREATE TABLE—TEMPORARY TABLE	N	Y	Y	Y	Y （with #prefix）
INDEX—CREATE INDEX	N	Y	Y	Y	Y
VIRTUAL COLUMN	N	N (only 2 predefined)	N	Y	Y

MaxCompute不支持的DML语法

语法	MaxCompute	Hive	MySQL	Oracle	SQL Server
CTE	Y	Y	Y	Y	Y
SELECT—recursive CTE	N	N	N	Y	Y
SELECT—GROUP BY ROLL UP	Y	Y	Y	Y	Y
SELECT—GROUP BY CUBE	Y	Y	N	Y	Y
SELECT—GROUPING SET	Y	Y	N	Y	Y
SELECT—IMPLICT JOIN	Y	Y	N	Y	Y
SELECT—PIVOT	N	N	N	Y	Y

语法	MaxCompute	Hive	MySQL	Oracle	SQL Server
SEMI JOIN	Y	Y (corelated expression must be in WHERE , EXISTS must be corelated)	Y	N (has IN and EXISTS , but no SEMI JOIN grammer)	N (has IN and EXISTS, but no SEMI JOIN grammer)
SELEC TRANSFROM	Y	Y	N	N	N
SELECT—corelated subquery	Y	Y (corelated expression must be in WHERE , EXISTS must be corelated)	Y	Y	Y
ORDER BY NULLS FIRST/LAST	Y	Y	Y	Y	Y
LATERAL VIEW	Y	Y	N	Y (LATERAL keyword)	Y (CROSS APPLY keyword)
SET OPERATOR—UNION (disintct)	Y	Y	Y	Y	Y
SET OPERATOR—INTERSECT	Y	N	N	Y	Y
SET OPERATOR—MINUS/EXCEPT	Y	N	N	Y	Y（keyword EXCEPT）
INSERT INTO … VALUES	Y	Y	Y	Y	Y

语法	MaxCompute	Hive	MySQL	Oracle	SQL Server
INSERT INTO (ColumnList)	Y	Y	Y	Y	Y
UPDATE … WHERE	N	Y	Y	Y	Y
UPDATE … ORDER BY LIMIT	N	N	Y	N	Y
DELETE … WHERE	N	Y	Y	Y	Y
DELETE … ORDER BY LIMIT	N	N	Y	N	N
ANALYTIC —reusable WINDOWING CLUSUE	Y	Y	N	N	N (can implement with join)
ANALYTIC— CURRENT ROW	Y	Y	N	Y	Y
ANALYTIC— UNBOUNDED	Y	N	Y	Y	Y
ANALYTIC—RANGE …	N	Y	N	Y	Y
WHILE DO	N	N	Y	Y	Y

MaxCompute不支持的SCRIPTING语法

语法	MaxCompute	Hive	MySQL	Oracle	SQL Server
TABLE VARIABLE	Y	Y (TEMPORARY TABLE)	Y (TEMPORARY TABLE)	Y (TEMPLORARY TABLE)	Y
SCALER VARIABLE	Y	Y	Y (DECLARE x INT)	Y	Y
ERROR HANDLING —RAISE ERROR	N	N	Y	Y	Y
ERROR HANDLING —TRY CATCH	N	N	N	Y	Y

语法	MaxCompute	Hive	MySQL	Oracle	SQL Server
FLOW CONTROL—LOOP	N	N	Y	Y	Y
CURSOR	N	N	Y	Y	Y

4.9 SELECT语句

4.9.1 SELECT语法介绍

本文为您介绍MaxCompute SELECT语法格式及使用SELECT语法执行嵌套查询、排序操作、分组查询等操作的注意事项。

SELECT语法格式

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[ORDER BY order_condition]
[DISTRIBUTE BY distribute_condition [SORT BY sort_condition] ]
[LIMIT number]
```

使用`select`语句时的注意事项请参见本文后续部分。

列表达式

`select`操作从表中读取数据，要读的列可以用列名指定，或者用`*`代表所有的列，一个简单的`select`语句，如下所示。

```
select * from sale_detail;
```

只读取`sale_detail`的一列`shop_name`。

```
select shop_name from sale_detail;
```

在`where`中可以指定过滤的条件。

```
select * from sale_detail where shop_name like 'hang%';
```

当使用`select`语句屏显时，目前最多只能显示10000行结果。当`select`作为子句时，无此限制，`select`子句会将全部结果返回给上层查询。

使用说明：

- select分区表时禁止全表扫描。

  2018-01-10 20点后创建的新项目，默认情况下执行SQL时，针对该Project里的分区表不允许全表扫描，必须有分区条件指定需要扫描的分区，由此减少SQL的不必要I/O，从而减少计算资源的浪费，同时也减少了不必要的后付费模式的计算费用（后付费模式中，数据输入量是计量计费参数之一）。

  例如表定义是t1(c1,c2) partitioned by(ds)，在新项目里执行如下语句会被禁止，返回Error。

  ```
  select * from t1 where c1=1;
  select * from t1 where (ds='20180202' or c2=3);
  select * from t1 left outer join t2 on a.id =b.id and a.ds=b.ds and
  b.ds='20180101');
  --join进行关联时，若分区剪裁条件放在where中，则分区剪裁生效，如果放在on条件
  中，从表的分区剪裁会生效，主表则进行全表扫描。
  ```

  如果您需要对分区表进行全表扫描，可以在对分区表全表扫描的SQL语句前加set odps.sql.allow.fullscan=true;，并和SQL语句一起提交执行。假设sale_detail表为分区表，则要全表扫描需同时提交以下的简单查询命令。

  ```
  set odps.sql.allow.fullscan=true;
  select * from sale_detail;
  ```

  如果需要整个项目都允许全表扫描，可以通过开关自行打开或关闭（True/False），命令如下。

  ```
  setproject odps.sql.allow.fullscan=true;
  ```

- 在table_reference中支持使用嵌套子查询，如下所示。

  ```
  select * from (select region from sale_detail) t where region = '
  shanghai';
  ```

  说明:

  当使用select语句屏显时，目前最多只能显示10000行结果。当select作为子句时则无此限制，select子句会将全部结果返回给上层查询。

DISTINCT去重

您可以通过在字段前使用distinct去掉重复字段，只返回一个值；而使用all会返回字段中所有重复的值。不指定此选项时，默认效果和使用all的效果相同。

使用distinct只返回一行记录，如下所示。

```
select distinct region from sale_detail;
select distinct region, sale_date from sale_detail;
```

```
-- distinct多列，distinct的作用域是select的列集合，不是单个列。
```

SELECT_EXPR正则表达式

MaxCompute SQL支持使用正则表达式`select_expr`选列。

使用正则表达式`select_expr`：

- `SELECT `abc.*` FROM t;`选出t表中所有列名以abc开头的列。
- `SELECT `(ds)?+.+` FROM t;`选出t表中列名不为ds的所有列。
- `SELECT `(ds|pt)?+.+` FROM t;`选出t表中排除ds和pt两列的其它列。
- `SELECT `(d.*)?+.+` FROM t;`选出t表中排除列名以d开头的其它列。

说明：

在排除多个列时，如果col2是col1的前缀，则需保证col1写在col2的前面（较长的col写前面）。例如，一个表有2个分区无需被select，一个分区名为ds，另一个为dshh，则正确表达式为`SELECT `(dshh|ds)?+.+` FROM tbl;`；错误表达式为`SELECT `(ds|dshh)?+.+` FROM tbl;`。

WHERE子句过滤

`where`子句支持的过滤条件，如下表所示。

过滤条件	描述
>、<、=、>=、<=、<>	关系操作符。
like、rlike	like和rlike的source和pattern参数均仅接受STRING类型。
in、not in	如果在in/not in条件后加子查询，子查询只能返回一列值，且返回值的数量不能超过1000。

在`select`语句的`where`子句中，您可以指定分区范围，只扫描表的指定部分，避免全表扫描，如下所示。

```
SELECT sale_detail.*
FROM sale_detail
WHERE sale_detail.sale_date >= '2008'
AND sale_detail.sale_date <= '2014';
```

说明：

您可以通过`explain select`语句查看分区裁剪是否生效。普通的UDF或`join`的分区条件写法都有可能导致分区裁剪不生效，详情请参见分区剪裁合理性评估。

UDF支持分区裁剪，支持的方式是将UDF语句先当做一个小作业执行，再将执行的结果替换到原来UDF出现的位置 。实现的方式有以下两种：

- 在编写UDF的时候，UDF类上加入Annotation。

```
@com.aliyun.odps.udf.annotation.UdfProperty(isDeterministic=true)
```

说明：

com.aliyun.odps.udf.annotation.UdfProperty定义在*odps-sdk-udf.jar*文件中。您需要把引用的*odps-sdk-udf*版本提高到0.30.x或以上。

- 在SQL语句前设置Flag：`set odps.sql.udf.ppr.deterministic = true;`，此时SQL中所有的UDF均被视为`deterministic`。该操作执行的原理是做执行结果回填，但是结果回填存在限制，即最多回填1000个Partition。因此，如果UDF类加入Annotation，则可能会导致出现超过1000个回填结果的报错。此时您如果需要忽视此错误，可以通过设置Flag：`set odps.sql.udf.ppr.to.subquery = false;`全局关闭此功能。关闭后，UDF分区裁剪也会失效。

MaxCompute SQL的`where`子句支持`between...and`条件查询。

```
SELECT sale_detail.*
FROM sale_detail
WHERE sale_detail.sale_date BETWEEN '2008' AND '2014';
```

GROUP BY分组查询

通常，`group by`和聚合函数会配合使用。在`select`中包含聚合函数时有以下规则：

1. 用`group by`的Key可以是输入表的列名。
2. 也可以是由输入表的列构成的表达式，不允许是`select`语句的输出列的别名。
3. 规则1的优先级高于规则2。当规则1和规则2发生冲突时，即`group by`的Key既是输入表的列或表达式，又是`select`的输出列时，以规则1为准。

示例

```
select region from sale_detail group by region;
-- 直接使用输入表列名作为group by的列，可以运行。
select sum(total_price) from sale_detail group by region;
-- 以region值分组，返回每一组的销售额总量，可以运行。
select region, sum(total_price) from sale_detail group by region;
-- 以region值分组，返回每一组的region值(组内唯一)及销售额总量，可以运行。
select region as r from sale_detail group by r;
-- 使用select列的别名运行，报错返回。
select 2 + total_price as r from sale_detail group by 2 + total_price;
-- 必须使用列的完整表达式。
select region, total_price from sale_detail group by region;
-- 报错返回，select的所有列中，没有使用聚合函数的列，必须出现在group by中。
```

```
select region, total_price from sale_detail group by region,
total_price;
-- 可以运行。
```

通常，在SQL解析中，group by操作先于select操作，因此group by只能接受输入表的列或表达式为Key。

说明:

- 关于聚合函数的详情请参见聚合函数。
- order by用于对所有数据按照某几列进行全局排序。如果您希望按照降序对记录进行排序，可以使用desc关键字。由于是全局排序，order by必须与limit共同使用。在使用order by排序时，NULL会被认为比任何值都小，这个行为与MySQL一致，但是与Oracle不一致。

  与group by不同，order by后面必须加select列的别名。当select某列时，如果没有指定列的别名，则列名会被作为列的别名。

```
select * from sale_detail order by region;
-- 报错返回，order by没有与limit共同使用。
select * from sale_detail order by region limit 100;
select region as r from sale_detail order by region limit 100;
-- 报错返回，order by后面必须加列的别名。
select region as r from sale_detail order by r limit 100;
```

LIMIT NUMBER限制输出行数

limit number的number是常数，限制输出行数。当使用无limit的select语句直接从屏幕输出查看结果时，最多只输出10000行。每个项目空间的这个屏显最大限制可能不同，您可以通过setproject命令控制。

ORDER BY/SORT BY/DISTRIBUTE BY

- distribute by用于对数据按照某几列的值做Hash分片，必须使用select的输出列别名。

```
select region from sale_detail distribute by region;
-- 列名即是别名，可以运行。
select region as r from sale_detail distribute by region;
-- 报错返回，后面必须加列的别名。
select region as r from sale_detail distribute by r;
```

- sort by用于局部排序，语句前必须加distribute by。实际上sort by是对distribute by的结果进行局部排序。必须使用select的输出列别名。

```
select region from sale_detail distribute by region sort by region;
select region as r from sale_detail sort by region;
```

```
-- 没有distribute by，报错退出。
```

- order by不和distribute by/sort by共用，同时group by也不和distribute by/sort by共用，必须使用select的输出列别名。

说明:

- order by/sort by/distribute by的Key必须是select语句的输出列，即列的别名。列的别名可以为中文。
- 在MaxCompute SQL解析中，order by/sort by/distribute by是后于select操作的，因此它们只能接受select语句的输出列为Key。

4.9.2 SELECT语序

对于按照SELECT语法格式书写的SELECT语句，它的逻辑执行顺序与标准的书写语序并不相同。

示例一

```
SELECT  key
        ,MAX(value)
FROM    src t
WHERE   value > 0
GROUP BY key
HAVING  SUM(value) > 100
ORDER BY key
LIMIT   100
;
```

以上语句的逻辑执行顺序是FROM->WHERE->GROUY BY->HAVING->SELECT->ORDER BY->LIMIT:

- ORDER BY中只能引用SELECT列表中生成的列，而不是访问FROM的源表中的列。
- HAVING可以访问的是GROUP BY key和聚合函数。
- SELECT的时候，如果有GROUP BY，便只能访问GROUP BY key和聚合函数，而不是FROM中源表中的列。

为避免混淆，MaxCompute支持以执行顺序书写查询语句。

```
FROM    src t
WHERE   value > 0
GROUP BY key
HAVING  SUM(value) > 100
SELECT  key
        ,MAX(value)
ORDER BY key
LIMIT   100
```

```
;
```

示例二

```
SELECT  shop_name
        ,total_price
        ,region
FROM    sale_detail
WHERE   total_price > 150
DISTRIBUTE BY region
SORT BY region
;
```

以上语句的逻辑执行顺序是FROM->WHERE->SELECT->DISTRIBUTE BY->SORT BY。

4.9.3 子查询

本文为您介绍MaxCompute支持的子查询操作。

子查询基本定义

普通的select是从几张表中读数据，例如select column_1, column_2 … from table_name，但是查询的对象也可以是另一个select操作。

```
select * from (select shop_name from sale_detail) a;
```

说明:

子查询必须要有别名。

您也可以使用以下语句达到同样的效果，但它只能返回一条记录。

```
select (select a from table1) from table2;
```

在from子句中，子查询可以被当作一张表，与其它的表或子查询进行join操作。

```
create table shop as select * from sale_detail;
select a.shop_name, a.customer_id, a.total_price from
(select * from shop) a join sale_detail on a.shop_name = sale_detail.
shop_name;
```

IN SUBQUERY/NOT IN SUBQUERY

in subquery与left semi join用法类似。

```
select * from mytable1 where id in (select id from mytable2);
--等效于以下语句。
```

```
select * from mytable1 a left semi join mytable2 b on a.id = b.id;
```

目前MaxCompute不仅支持in subquery，还支持Correlated条件。

```
select * from mytable1 where id in (select id from mytable2 where
value = mytable1.value);
```

子查询中的where value = mytable1.value即是一个Correlated条件。MaxCompute 1.0版本不支持这种既引用了子查询中源表，又引用了外层查询源表的表达式。MaxCompute 2.0版本已支持这种用法，这样的过滤条件构成了semi join中on条件的一部分。

not in subquery与left anti join用法类似，但并不完全相同。示例如下。

```
select * from mytable1 where id not in (select id from mytable2);
--如果mytable2中的所有id都不为NULL，则等效于以下语句。
select * from mytable1 a left anti join mytable2 b on a.id = b.id;
```

如果mytable2中有任何为NULL的列，则not in表达式会为NULL，导致where条件不成立，无数据返回，此时与left anti join不同。

MaxCompute 1.0版本也支持 (not) in subquery不作为join条件。例如出现在非where语句中，或者虽然在where语句中，但无法转换为join条件。MaxCompute 2.0版本仍然支持这种用法，但是因为此时无法转换为semi join，必须启动一个单独的作业运行Subquery，所以不支持Correlated条件，示例如下。

```
select * from mytable1 where id in (select id from mytable2) or value
 > 0;
```

因为where中包含了or，所以无法转换为semi join，会单独启动作业执行子查询。

在处理分区表的时候，也会有特殊处理，示例如下。

```
SELECT * from sales_detail where ds in (select dt from sales_date);
```

如果ds是分区列，则select dt from sales_date会单独启动作业执行子查询，而不会转化为semi join，执行后的结果会依次与ds比较，sales_detail中ds值不在返回结果中的分区将不会被读取，保证分区裁剪仍然有效。

EXISTS SUBQUERY/NOT EXISTS SUBQUERY

使用exists subquery时，当Subquery中有至少一行数据时，返回True，否则返回False。使用not exists时，则相反。

目前只支持含有Correlated where条件的子查询。exists subquery和not exists subquery实现的方式是转换为left semi join或者left anti join。

示例一

```
SELECT * from mytable1 where exists (select * from mytable2 where id
 = mytable1.id);
--等效于以下语句。
SELECT * from mytable1 a left semi join mytable2 b on a.id = b.id;
```

示例二

```
select * from mytable1 where not exists (select * from mytable2 where
id = mytable1.id);
--等效于以下语句。
select * from mytable1 a left anti join mytable2 b on a.id = b.id;
```

SCALAR SUBQUERY

当Subquery的输出结果为单行单列时，可以做为标量使用。

```
select * from t1 where (select count(*) from t2 where t1.a = t2.a) > 1
;
--等效于以下语句。
select t1.* from t1 left semi join (select a, count(*) from t2 group
by a having count(*) > 1) t2 on t1.a = t2.a;
```

语句`select count(*) from t2 where t1.a = t2.a;`的输出结果是一个Row Set，可以判断出这条语句的输出有且仅有一行一列。因此它可以作为标量，即可以参与标量运算（‘>’）。但在实现过程中，会尽可能地转成`join`来处理，例如上述示例中的等效语句。

说明:

可以作为标量的Subquery必须是在编译阶段就能够确认其返回结果只有一行一列的查询。如果一个Subquery只能在实际运行过程中才能判断出它只产生一行数据（而无法在编译过程中判断），则编译器会报错。

目前编译器能够接受的语句需满足两个特征：

- 子查询的`select`列表里面用了聚合函数，且不在表值函数的参数列表中。
- 子查询中包含聚合函数的这一层查询没有`group by`语句。

同时还有两个限制：

- Scalar Subquery支持引用外层查询的列，当嵌套多层Scalar Subquery时，只支持引用直接外层的列。

  ```
  select * from t1 where (select count(*) from t2 where t1.a = t2.a)
   = 3; --允许的操作。
  ```

```
select * from t1 where (select count(*) from t2 where (select count
(*) from t3 where t3.a = t1.a) = 2) = 3; --不允许的操作，不能在子查询的子
查询中引用外部查询的列。
```

- Scalar Subquery只能where中使用。

```
select * from t1 where (select t1.b + count(*) from t2) = 3; --不能在
子查询的select中引用。
select (select count(*) from t2 where t2.a = t1.a) from t1; --不能在
外层查询的select中引用。
```

所有的满足一行一列输出值的子查询都可以按照上述示例进行重写（如果查询的结果只有一行，在外面嵌套一层max或min操作，其结果不变）。

4.9.4 交集、并集和补集

本文为您介绍UNOIN ALL、UNION DISTINCT并集，INTERSECT ALL、INTERSECT DISTINCT交集，EXCEPT ALL、EXCEPT DISTINCT补集等SQL语法。

语法格式如下。

```
select_statement UNION ALL select_statement;
select_statement UNION [DISTINCT] select_statement;
select_statement INTERSECT ALL select_statement;
select_statement INTERSECT [DISTINCT] select_statement;
select_statement EXCEPT ALL select_statement;
select_statement EXCEPT [DISTINCT] select_statement;
select_statement MINUS ALL select_statement;
select_statement MINUS [DISTINCT] select_statement;
```

- UNION：求两个数据集的并集，即将两个数据集合并成一个数据集。
- INTERSECT：求两个数据集的交集，即输出两个数据集均包含的记录。
- EXCEPT：求第二个数据集在第一个数据集中的补集，即输出第一个数据集包含而第二个数据集不包含的记录。
- MINUS：等同于EXCEPT。

示例

- UNION ALL示例

```
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4) t(a, b)
UNION ALL
SELECT * FROM VALUES (1, 2), (1, 4) t(a, b);
```

结果：将两个数据集合并。

```
+------------+------------+
| a          | b          |
+------------+------------+
| 1          | 2          |
| 1          | 4          |
| 1          | 2          |
```

```
| 1           | 2           |
| 3           | 4           |
+-------------+-------------+
```

- UNION DISTINCT示例

```
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4) t(a, b)
UNION
SELECT * FROM VALUES (1, 2), (1, 4) t(a, b);
```

结果：等同于SELECT DISTINCT * FROM (< UNION ALL的结果 >) t;语句。

```
+-------------+-------------+
| a           | b           |
+-------------+-------------+
| 1           | 2           |
| 1           | 4           |
| 3           | 4           |
+-------------+-------------+
```

- INTERSECT ALL示例

```
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (5, 6) t(a, b)
INTERSECT ALL
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (5, 7) t(a, b);
```

结果：INTERSECT ALL不做去重。可以理解为相同的行，每行后面有个隐藏的序号，可用于将各行区分开显示。

```
+-------------+-------------+
| a           | b           |
+-------------+-------------+
| 1           | 2           |
| 1           | 2           |
| 3           | 4           |
+-------------+-------------+
```

- INTERSECT DISTINCT示例

```
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (5, 6) t(a, b)
INTERSECT
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (5, 7) t(a, b);
```

结果：等同于SELECT DISTINCT * FROM (< INTERSECT ALL的结果 >) t;语句。

```
+-------------+-------------+
| a           | b           |
+-------------+-------------+
| 1           | 2           |
| 3           | 4           |
+-------------+-------------+
```

- EXCEPT ALL示例

```
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (3, 4), (5, 6), (7, 8)
t(a, b)
EXCEPT ALL
```

```
SELECT * FROM VALUES (3, 4), (5, 6), (5, 6), (9, 10) t(a, b);
```

结果：EXCEPT ALL不做去重。对于相同的行，每行后面有个隐藏的序号，将各行区分显示。

```
+------------+------------+
| a          | b          |
+------------+------------+
| 1          | 2          |
| 1          | 2          |
| 3          | 4          |
| 7          | 8          |
+------------+------------+
```

- EXCEPT DISTINCT示例

```
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (3, 4), (5, 6), (7, 8)
t(a, b)
EXCEPT
SELECT * FROM VALUES (3, 4), (5, 6), (5, 6), (9, 10) t(a, b);
```

结果：相当于SELECT DISTINCT * FROM left_branch EXCEPT ALL SELECT DISTINCT * FROM right_branch;。

```
+------------+------------+
| a          | b          |
+------------+------------+
| 1          | 2          |
| 7          | 8          |
+------------+------------+
```

说明：

- 集合操作的结果不一定会按序排列。
- 集合操作左右两个分支要求列个数必须一致。如果类型不一致，可能会进行隐式类型转换。由于兼容性原因，STRING类型和非STRING类型在集合操作中的隐式转换已被禁用。
- MaxCompute最多允许256个分支的集合操作，超出个数将报错。
- UNION后如果有CLUSTER BY、DISTRIBUTE BY、SORT BY、ORDER BY或者LIMIT子句，当设置set odps.sql.type.system.odps2=false;时，其作用于UNION的最后一个select_statement；当set odps.sql.type.system.odps2=true;时，作用于前面所有UNION的结果。示例如下。

  ```
  set odps.sql.type.system.odps2=true;
  SELECT explode(array(3, 1)) AS (a) UNION ALL SELECT explode(array
  (0, 4, 2)) AS (a) ORDER BY a LIMIT 3;
  ```

  返回结果如下。

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

```
| 0    |
| 1    |
| 2    |
+------+
```

4.9.5 JOIN

您可以使用JOIN操作连接两张表的查询结果。MaxCompute的JOIN操作分为左连接、右连接、全连接、内连接。支持多路连接，但不支持CROSS JOIN笛卡尔积，即无ON条件的连接。

命令格式如下。

```
join_table:
        table_reference join table_factor [join_condition]
        | table_reference {left outer|right outer|full outer|inner}
join table_reference join_condition
    table_reference:
        table_factor
        | join_table
    table_factor:
        tbl_name [alias]
        | table_subquery alias
        | ( table_references )
    join_condition:
        on equality_expression ( and equality_expression )
```

说明:

`equality_expression`是一个等式表达式。

- `left outer join`：左连接。返回左表中的所有记录，即使右表中没有与之匹配的记录。

```
select a.shop_name as ashop, b.shop_name as bshop from shop a
        left outer join sale_detail b on a.shop_name=b.shop_name;
    -- 由于表shop及sale_detail中都有shop_name列，因此需要在select子句中使用别名进行区分。
```

说明:

如果右边值不唯一，建议不要连续使用过多`left join`，以免在`join`的过程中产生数据膨胀，导致作业停滞。

- `right outer join`：右连接。返回右表中的所有记录，即使左表中没有与之匹配的记录。

```
select a.shop_name as ashop, b.shop_name as bshop from shop a
        right outer join sale_detail b on a.shop_name=b.shop_name;
```

- `full outer join`：全连接。返回左右表中的所有记录。

```
select a.shop_name as ashop, b.shop_name as bshop from shop a
```

```
        full outer join sale_detail b on a.shop_name=b.shop_name;
```

以下示例中，表A是test_table_a，表B是test_table_b，使用了join查询和where过滤查询。查询分区大于20180101的数据中origin和id一致的记录数，使用left join可以保留左表中test_table_a的全表记录。正常情况下，如果您使用left join，左表会保留全表记录；如果您使用right join，右表会保留全表记录。

错误示例如下。

```
SELECT s.id
        ,s.name
        ,s.origin
        ,d.value
FROM    test_table_a s
LEFT JOIN   test_table_b d
ON      s.origin = d.id
WHERE   s.ds > "20180101" AND d.ds>"20180101";
```

如果join在where条件之前， 会先做join操作，然后对join的结果做where条件过滤。您会发现获取的结果是两个表的交集，而不是全表。

修正后的SQL如下，通过这种方式可以显示全表。

```
SELECT s.id
        ,s.name
        ,s.origin
        ,d.value
FROM  (select * from  test_table_a where ds > "20180101" ) s
LEFT JOIN (select * from  test_table_b where ds > "20180101") d
ON s.origin = d.id;
```

表中存在至少一个匹配时，inner join返回行。关键字inner可省略。

```
select a.shop_name from shop a inner join sale_detail b on a.shop_name
=b.shop_name;
select a.shop_name from shop a join sale_detail b on a.shop_name=b.
shop_name;
```

连接条件，只允许and连接的等值条件。只有在MAPJOIN中，可以使用不等值连接或者使用or连接多个条件。

```
select a.* from shop a full outer join sale_detail b on a.shop_name=b.
shop_name
        full outer join sale_detail c on a.shop_name=c.shop_name;
    -- 支持多路join链接示例。
select a.* from shop a join sale_detail b on a.shop_name != b.
shop_name;
    -- 不支持不等值Join链接条件，报错返回。
```

IMPLICIT JOIN：MaxCompute支持以下join方式。

```
SELECT * FROM table1, table2 WHERE table1.id = table2.id;
--执行的效果相当于以下语句。
```

```
SELECT * FROM table1 JOIN table2 ON table1.id = table2.id;
```

4.9.6 SEMI JOIN

MaxCompute支持SEMI JOIN（半连接）。SEMI JOIN中，右表只用于过滤左表的数据而不出现在结果集中。支持LEFT SEMI JOIN和LEFT ANTI JOIN两种语法。

LEFT SEMI JOIN

当join条件成立时，返回左表中的数据。如果mytable1中某行的id在mytable2的所有id中出现过，则此行保留在结果集中。

示例如下。

```
SELECT * from mytable1 a LEFT SEMI JOIN mytable2 b on a.id=b.id;
```

只会返回mytable1中的数据，只要mytable1的id在mytable2的id中出现。

LEFT ANTI JOIN

当join条件不成立时，返回左表中的数据。如果mytable1中某行的id在mytable2的所有id中没有出现过，则此行保留在结果集中。

示例如下。

```
SELECT * from mytable1 a LEFT ANTI JOIN mytable2 b on a.id=b.id;
```

只会返回mytable1中的数据，只要mytable1的id在mytable2的id没有出现。

4.9.7 MAPJOIN HINT

当一个大表和一个或多个小表JOIN时，您可以使用MAPJOIN提升性能。

MAPJOIN的基本原理：在小数据量的场景中，SQL会将您指定的小表全部加载到执行JOIN操作的程序的内存中，从而加快JOIN的执行速度。

说明:

当您使用 MAPJOIN时，需要注意以下几点：

- LEFT OUTER JOIN的左表必须是大表。
- RIGHT OUTER JOIN的右表必须是大表。
- INNER JOIN的左表或右表均可以作为大表。
- FULL OUTER JOIN不能使用MAPJOIN。
- MAPJOIN支持小表为子查询。
- 使用MAPJOIN时，在引用小表或是子查询时，需要引用别名。

- 在MAPJOIN中，可以使用不等值连接或者使用OR连接多个条件。您可以通过不写ON语句或通过mapjoin on 1 = 1的形式，实现笛卡尔乘积的计算，例如select /* + mapjoin(a) */ a.id from shop a join table_name b on 1=1，但这样可能带来数据量膨胀问题。
- 目前，MaxCompute在MAPJOIN中最多支持指定256张小表，否则报语法错误。MAPJOIN HINT中多个小表用逗号隔开，例如/*+mapjoin(a,b,c)*/。
- 如果使用MAPJOIN，则小表占用的总内存不得超过640MB。由于MaxCompute是压缩存储，因此小表在被加载到内存后，数据大小会急剧膨胀。此处的640MB限制是加载到内存后的空间大小。

示例

```
select /* + mapjoin(a) */
        a.shop_name,
        b.customer_id,
        b.total_price
from shop a join sale_detail b
on a.shop_name = b.shop_name;
```

MaxCompute SQL不支持在普通JOIN的ON条件中使用不等值表达式、OR逻辑等复杂的JOIN条件，但是在MAPJOIN中可以进行上述操作。示例如下。

```
select /*+ mapjoin(a) */
        a.total_price,
        b.total_price
from shop a join sale_detail b
on a.total_price < b.total_price or a.total_price + b.total_price <
500;
```

MAPJOIN多个小表示例请参见用MAPJOIN缓存多张小表。

4.9.8 HAVING子句

MaxCompute SQL的WHERE关键字无法与合计函数一起使用，此时您可以使用HAVING子句来实现。

命令格式如下。

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
```

```
HAVING aggregate_function(column_name) operator value
```

例如有一张订单表Orders，包括客户名称（Customer），订单金额（OrderPrice），订单日期（Order_date），订单号（Order_id）四个字段。现在需要查找订单总额少于2000的客户，SQL语句如下所示。

```
SELECT Customer,SUM(OrderPrice) FROM Orders
GROUP BY Customer
HAVING SUM(OrderPrice)<2000
```

4.9.9 GROUPING SETS

对于经常需要对数据进行多维度的聚合分析的场景，您既需要对a列做聚合，也要对b列做聚合，同时要按照a、b两列同时做聚合，因此需要多次使用UNION ALL。使用GROUPING SETS可以快速解决此类问题。

MaxCompute中的GROUPING SETS是对SELECT语句中，GROUP BY子句的扩展，允许您采用多种方式对结果分组，而不必使用多个SELECT语句来实现这一目的。这样能够使MaxCompute的引擎给出更有效的执行计划，从而提高执行性能。

说明:

本文中很多示例采用MaxCompute Studio进行展示，建议您安装MaxCompute Studio后，再进行后续操作。

实现示例

准备数据

```
create table requests LIFECYCLE 20 as
select * from values
    (1, 'windows', 'PC', 'Beijing'),
    (2, 'windows', 'PC', 'Shijiazhuang'),
    (3, 'linux', 'Phone', 'Beijing'),
    (4, 'windows', 'PC', 'Beijing'),
    (5, 'ios', 'Phone', 'Shijiazhuang'),
    (6, 'linux', 'PC', 'Beijing'),
    (7, 'windows', 'Phone', 'Shijiazhuang')
as t(id, os, device, city);
```

您可以通过以下两种方式进行分组：

- 使用多个SELECT语句进行分组。

```
SELECT NULL, NULL, NULL, COUNT(*)
FROM requests
UNION ALL
SELECT os, device, NULL, COUNT(*)
FROM requests GROUP BY os, device
UNION ALL
SELECT null, null, city, COUNT(*)
```

```
FROM requests GROUP BY city;
```

- 使用GROUPING SETS进行分组。

```
SELECT os,device, city ,COUNT(*)
FROM requests
GROUP BY os, device, city GROUPING SETS((os, device), (city), ());
```

执行结果如下。

```
+----+--------+------+------------+
| os | device | city | cnt        |
+----+--------+------+------------+
| NULL | NULL   | NULL | 7          |
| NULL | NULL   | Beijing | 4          |
| NULL | NULL   | Shijiazhuang | 3          |
| ios | Phone  | NULL | 1          |
| linux | PC     | NULL | 1          |
| linux | Phone  | NULL | 1          |
| windows | PC     | NULL | 3          |
| windows | Phone  | NULL | 1          |
+----+--------+------+------------+
```

说明:

分组集中不使用的表达式，会使用NULL充当占位符，使得这些结果集可以做UNION操作。例如结果第1-5行的city列。

CUBE and ROLLUP函数

CUBE和ROLLUP可以认为是特殊的GROUPING SETS。CUBE会枚举指定列的所有可能组合作为GROUPING SETS，而ROLLUP会以按层级聚合的方式产生GROUPING SETS。

示例如下。

```
GROUP BY CUBE(a, b, c)
--等价于以下语句。
ROUPING SETS((a,b,c),(a,b),(a,c),(b,c),(a),(b),(c),())

GROUP BY ROLLUP(a, b, c)
--等价于以下语句。
GROUPING SETS((a,b,c),(a,b),(a), ())
```

GROUPING和GROUPING_ID函数

GROUPING SETS结果中使用NULL充当占位符，导致您会无法区分占位符NULL与数据中真正的NULL。因此，MaxCompute为您提供了GROUPING函数。GROUPING函数接受一个列名作为参数，如果结果对应行使用了参数列做聚合，返回0，此时意味着NULL来自输入数据。否则返回1，此时意味着NULL是GROUPING SETS的占位符。

MaxCompute还提供了GROUPING_ID函数，此函数接受一个或多个列名作为参数。结果是将参数列的GROUPING结果按照BitMap的方式组成整数。

示例如下。

```
SELECT a,b,c ,COUNT(*),
GROUPING(a) ga, GROUPING(b) gb, GROUPING(c) gc, GROUPING_ID(a,b,c)
groupingid
FROM VALUES (1,2,3) as t(a,b,c)
GROUP BY CUBE(a,b,c);
```

执行结果如下。

```
+------------+------------+------------+------------+------------
+------------+------------+------------+
| a          | b          | c          | _c3        | ga         | gb
        | gc         | groupingid |
+------------+------------+------------+------------+------------
+------------+------------+------------+
| NULL       | NULL       | NULL       | 1          | 1          | 1
        | 1          | 7          |
| NULL       | NULL       | 3          | 1          | 1          | 1
        | 0          | 6          |
| NULL       | 2          | NULL       | 1          | 1          | 0
        | 1          | 5          |
| NULL       | 2          | 3          | 1          | 1          | 0
        | 0          | 4          |
| 1          | NULL       | NULL       | 1          | 0          | 1
        | 1          | 3          |
| 1          | NULL       | 3          | 1          | 0          | 1
        | 0          | 2          |
| 1          | 2          | NULL       | 1          | 0          | 0
        | 1          | 1          |
| 1          | 2          | 3          | 1          | 0          | 0
        | 0          | 0          |
+------------+------------+------------+------------+------------
+------------+------------+------------+
```

4.9.10 SELECT TRANSFORM语法

SELECT TRANSFORM功能允许您指定启动一个子进程，将输入数据按照一定的格式通过stdin输入子进程，并且通过parse子进程的stdout输出获取输出数据。您无需编写UDF，便可以实现MaxCompute SQL不具备的功能。

命令格式如下。

```
SELECT TRANSFORM(arg1, arg2 ...)
(ROW FORMAT DELIMITED (FIELDS TERMINATED BY field_delimiter (ESCAPED
BY character_escape)?)?
(LINES SEPARATED BY line_separator)?
(NULL DEFINED AS null_value)?)?
USING 'unix_command_line'
(RESOURCES 'res_name'  (',' 'res_name') *)?
( AS col1, col2 ...)?
(ROW FORMAT DELIMITED (FIELDS TERMINATED BY field_delimiter (ESCAPED
BY character_escape)?)?
(LINES SEPARATED BY line_separator)? (NULL DEFINED AS null_value)?)?
```

说明：

- SELECT TRANSFORM关键字可以用MAP或REDUCE关键字替换，无论使用哪个关键字，语义是完全一样的。为使语法更清晰，推荐您使用SELECT TRANSFORM。
- arg1,arg2...是transform的参数，其格式和select子句的Item类似。默认格式下，参数的各个表达式的结果会在隐式转换成STRING后，用\t拼起来，输入到子进程中（该格式可以进行配置，详情请参见下文对ROW FORMAT的说明）。
- USING：指定要启动的子进程的命令。

  说明：

  - 大多数的MaxCompute SQL命令Using子句指定的是资源（Resources），但此处是为了和Hive的语法兼容。
  - Using中的格式和Shell的语法非常类似，但并非真的启动Shell来执行，而是直接根据命令的内容来创建了子进程。因此，很多Shell的功能不能使用，例如输入输出重定向、管道、循环等。若有需要，Shell本身也可以作为子进程命令来使用。
- RESOURCES子句：允许指定子进程能够访问的资源，支持以下两种方式指定资源。
  - 支持使用resources子句，例如using 'sh foo.sh bar.txt' resources 'foo.sh','bar.txt'。
  - 支持在SQL语句前使用set odps.sql.session.resources=foo.sh,bar.txt;来指定。注意这种配置是全局的，意味着整个SQL中所有的select transform都可以访问这个设置配置的资源。
- ROW FORMAT子句：允许自定义输入输出的格式。

  语法中有两个Row Format子句，第一个子句指定输入的格式，第二个指定输出的格式。 默认情况下使用\t来作为列的分隔符，\n作为行的分隔符，使用\N（注意是两个字符，反斜杠字符和字符N）表示NULL。

  说明：

  - field_delimiter、character_escape和line_separator只接受一个字符。如果指定的是字符串，则以第一个字符为准。
  - MaxCompute支持Hive指定格式的语法，例如inputRecordReader、outputRecordReader、SerDe等，但您需要打开Hive兼容模式才能使用。打开方式为在SQL语句前加set语句set odps.sql.hive.compatible=true;。Hive支持的语法详情请参见Hive文档。

- 如果使用Hive的inputRecordReader、outputRecordReader等自定义类，可能会降低执行性能。

- AS子句：指定输出列。
  - 输出列可以不指定类型，默认为STRING类型，例如as(col1, col2)。您也可以指定类型，例如as(col1:bigint, col2:boolean)。
  - 由于输出实际是parse子进程stdout获取的，如果指定的类型不是STRING，系统会隐式调用cast函数，而cast有可能出现Runtime Exception。
  - 输出列类型不支持部分指定部分不指定，例如as(col1, col2:bigint)。
  - as可以省略，此时默认stdout的输出中第一个\t之前的字段为Key，后面的部分全部为Value，相当于as(key, value)。

调用Shell脚本示例

假设通过Shell脚本生成50行数据，值是从1到50，对应data字段输出如下。

```
SELECT TRANSFORM(script) USING 'sh' AS (data)
FROM (
        SELECT  'for i in `seq 1 50`; do echo $i; done' AS script
      ) t
;
```

直接将Shell命令作为transform数据输入。

SELECT TRANSFORM不仅仅是语言支持的扩展。一些简单的功能，例如AWK、Python、Perl、Shell都支持直接在命令中写脚本，而不需要专门编写脚本文件、上传资源等，开发过程更简单。对于复杂的功能，您可以上传脚本文件来执行，例如下文将介绍的调用Python脚本示例。

调用Python脚本示例

准备Python文件，脚本文件名为*myplus.py*，如下所示。

```
#!/usr/bin/env python
import sys
line = sys.stdin.readline()
while line:
    token = line.split('\t')
    if (token[0] == '\\N') or (token[1] == '\\N'):
        print '\\N'
    else:
        print int(token[0]) + int(token[1])
```

```
    line = sys.stdin.readline()
```

将该Python脚本文件添加为MaxCompute资源（Resource）。

```
add py ./myplus.py -f;
```

说明:

您也可通过DataWorks控制台进行新增资源操作。

使用`select transform`语法调用资源。

```
Create table testdata(c1 bigint,c2 bigint); --创建测试表。
insert into Table testdata values (1,4),(2,5),(3,6); --测试表中插入测试数据。

--接下来执行select transform。
SELECT
TRANSFORM (testdata.c1, testdata.c2)
USING 'python myplus.py'resources 'myplus.py'
AS (result bigint)
FROM testdata;

--或执行以下语句。
set odps.sql.session.resources=myplus.py;
SELECT TRANSFORM (testdata.c1, testdata.c2)
USING 'python myplus.py'
AS (result bigint)
FROM testdata;
```

执行结果如下所示。

```
+-----+
| cnt |
+-----+
| 5   |
| 7   |
| 9   |
+-----+
```

Python脚本无需依赖MaxCompute的Python框架，也没有格式要求。

MaxCompute也支持直接将Python命令作为`transform`数据输入。例如调用Shell脚本示例也可以用Python命令实现。

```
SELECT TRANSFORM('for i in xrange(1, 50):  print i;') USING 'python'
AS (data);
```

调用Java脚本示例

与之前调用Python脚本类似，在本例中，您需要编辑好Java文件，导出Jar包，再通过`add file`方式将Jar包添加为MaxCompute资源，最后用于`select transform`调用。

1. 准备好Jar文件，脚本文件名为*Sum.jar*。Java代码如下所示。

```
package com.aliyun.odps.test;
import java.util.Scanner
public class Sum {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNext()) {
            String s = sc.nextLine();
            String[] tokens = s.split("\t");
            if (tokens.length < 2) {
                throw new RuntimeException("illegal input");
            }
            if (tokens[0].equals("\\N") || tokens[1].equals("\\N"))
 {
                System.out.println("\\N");
            }
            System.out.println(Long.parseLong(tokens[0]) + Long.
parseLong(tokens[1]));
        }
    }
}
```

2. 将Jar文件添加为MaxCompute的Resource。

```
add jar ./Sum.jar -f;
```

3. 使用`select transform`语法调用资源。

```
Create table testdata(c1 bigint,c2 bigint); --创建测试表。
insert into Table testdata values (1,4),(2,5),(3,6); --测试表中插入测试
数据。
--接下来执行Select Transform。
SELECT TRANSFORM(testdata.c1, testdata.c2) USING 'java -cp Sum.jar
com.aliyun.odps.test.Sum' resources 'Sum.jar' from testdata;
--或执行以下语句。
set odps.sql.session.resources=Sum.jar;
SELECT TRANSFORM(testdata.c1, testdata.c2) USING 'java -cp Sum.jar
com.aliyun.odps.test.Sum' FROM testdata;
```

执行结果如下所示。

```
+-----+
| cnt |
+-----+
| 5   |
| 7   |
| 9   |
+-----+
```

大多数Java的Utility可以直接使用本方法运行。

说明:

Java和Python虽然有现成的UDTF框架，但是用`select transform`编写更简单，并且不需要额外依赖，也没有格式要求，甚至可以实现直接使用离线脚本（Java和Python离线脚本的实际路径，可以从`JAVA_HOME`和`PYTHON_HOME`环境变量中得到）。

调用其他脚本语言

select transform不仅仅支持上述语言扩展，还支持其它的常用Unix命令或脚本解释器，例如AWK、Perl等。

- 调用AWK，将第二列原样输出，示例如下。

```
SELECT TRANSFORM(*) USING "awk '//{print $2}'" as (data) from
testdata;
```

- 调用Perl，示例如下。

```
SELECT TRANSFORM (testdata.c1, testdata.c2) USING "perl -e 'while($
input = <STDIN>){print $input;}'" FROM testdata;
```

说明:

目前由于MaxCompute计算集群上未部署PHP和Ruby，所以不支持调用这两种脚本。

串联使用示例

select transform还可以串联使用。例如使用distribute by和sort by对输入数据做预处理。

```
SELECT TRANSFORM(key, value) USING 'cmd2' from
(
    SELECT TRANSFORM(*) USINg 'cmd1' from
    (
        SELECt * FROM data distribute by col2 sort by col1
    ) t distribute by key sort by value
) t2;
```

或使用map、reduce关键字。

```
@a := select * from data distribute by col2 sort by col1;
@b := map * using 'cmd1' distribute by col1 sort by col2 from @a;
reduce * using 'cmd2' from @b;
```

SELECT TRANSFORM性能介绍

select transform与UDTF在不同场景下性能不同。经过多种场景对比测试，数据量较小时，大多数场景下select transform有优势，而数据量大时UDTF有优势。因为select transform的开发更加简便，所以select transform更适合做Ad Hoc的数据分析。

UDTF的优势

- UDTF的输出结果和输入参数是有类型的，而transform的子进程基于stdin和stdout传输数据，所有数据都当做STRING处理，因此select transform多了一步类型转换。

- `select transform`数据传输依赖于操作系统的管道，而目前管道的Buffer仅有4KB，且不能设置。`select transform`读空或Pipe写满会导致进程被挂起。
- UDTF的常量参数可以不用传输，而`select transform`无法利用这个优化。

`select transform`的优势

- 子进程和父进程是两个进程，而UDTF是单线程的。如果计算占比较高，数据吞吐量较小，`select transform`可以利用服务器的多核特性。
- 数据的传输通过更底层的系统调用来读写，效率比Java高。
- `select transform`支持的部分工具，例如AWK是Native代码实现的。理论上，与Java相比，使用`select transform`会更有性能优势。

4.10 脚本模式

脚本模式SQL即Script Mode SQL。

> 说明:
> 脚本模式SQL目前无法使用计量预估（Cost SQL命令）完成费用预估，具体费用请以实际费用账单为准，详情请参见账单详情。

MaxCompute当前的SQL引擎支持脚本模式。在脚本模式下编译脚本时，一个多语句的SQL脚本文件将被作为一个整体进行编译，无需您对单个语句进行编译。在提交运行时，SQL脚本文件会被整体提交，并生成一个执行计划，保证只需排队一次、执行一次，保证您能充分利用MaxCompute的资源。

Script Mode的SQL语句书写便利，您只需要按照业务逻辑，用类似于普通编程语言的方式书写，无需考虑如何组织语句。

工具支持

- MaxCompute Studio，详情请参见认识Studio。
- MaxCompute CMD命令行工具，详情请参见安装并配置客户端。
- DataWorks，详情请参见ODPS SCRIPT节点。

语法结构

```
--set
set odps.sql.type.system.odps2=true;
[set odps.stage.reducer.num=xxx;]
[...]
--ddl
create table table1 xxx;
[create table table2 xxx;]
[...]
```

```
--dml
@var1 := SELECT [ALL | DISTINCT] select_expr, select_expr, ...
        FROM table3
        [WHERE where_condition];
@var2 := SELECT [ALL | DISTINCT] select_expr, select_expr, ...
        FROM table4
        [WHERE where_condition];
@var3 := SELECT [ALL | DISTINCT] var1.select_expr, var2.select_expr
, ...
        FROM @var1 join @var2 on ...;
INSERT OVERWRITE|INTO TABLE [PARTITION (partcol1=val1, partcol2=val2
 ...)]
        SELECT [ALL | DISTINCT] select_expr, select_expr, ...
        FROM @var3;
[@var4 := SELECT [ALL | DISTINCT] var1.select_expr, var.select_expr
, ... FROM @var1
        UNION ALL | UNION
        SELECT [ALL | DISTINCT] var1.select_expr, var.select_expr
, ... FROM @var2;
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
        AS
        SELECT [ALL | DISTINCT] select_expr, select_expr, ...
        FROM var4;]
[...]
```

- 脚本模式支持SET语句、部分DDL语句（不支持结果为屏显类型的语句如desc、show）、DML语句。
- 一个脚本的完整形式是SET、DDL、DML语句按先后顺序排列。每种语句都可以包含0到多个具体的SQL语句，且不同类型的语句不能交错。
- 使用@声明变量。
- 一个脚本最多支持一个屏显结果的语句（如单独的SELECT语句），否则会发生报错。不建议在脚本中执行屏显的SELECT语句。
- 一个脚本最多支持一个CREATE TABLE AS语句并且必须是最后一句。推荐您将建表语句与INSERT语句分开写。
- 脚本模式下，一旦有一个语句失败，整个脚本的语句都不会执行成功。
- 脚本模式下，只有当所有输入的数据都准备好并到达，才会生成一个作业进行数据处理。
- 脚本模式下，如果一个表先被写再被读，则会发生报错，如下所示。

  ```
  insert overwrite table src2 select * from src where key > 0;
  @a := select * from src2;
  select * from @a;
  ```

  所以，为避免因表的先写后读产生的报错，应修改SQL脚本如下。

  ```
  @a := select * from src where key > 0;
  insert overwrite table src2 select * from @a;
  ```

```
select * from @a;
```

脚本模式SQL示例

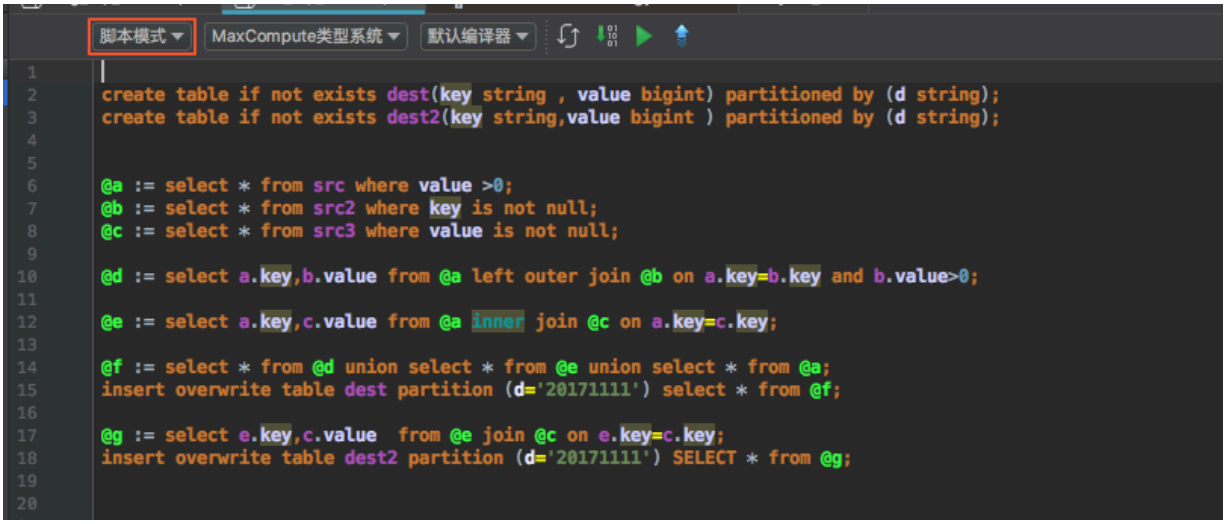
```
create table if not exists dest(key string , value bigint) partitioned
 by (d string);
create table if not exists dest2(key string,value bigint) partitioned
by (d string);
@a := select * from src where value >0;
@b := select * from src2 where key is not null;
@c := select * from src3 where value is not null;
@d := select a.key,b.value from @a left outer join @b on a.key=b.key
and b.value>0;
@e := select a.key,c.value from @a inner join @c on a.key=c.key;
@f := select * from @d union select * from @e union select * from @a;
insert overwrite table dest partition (d='20171111') select * from @f;
@g := select e.key,c.value from @e join @c on e.key=c.key;
insert overwrite table dest2 partition (d='20171111') SELECT * from @g
;
```

适用场景

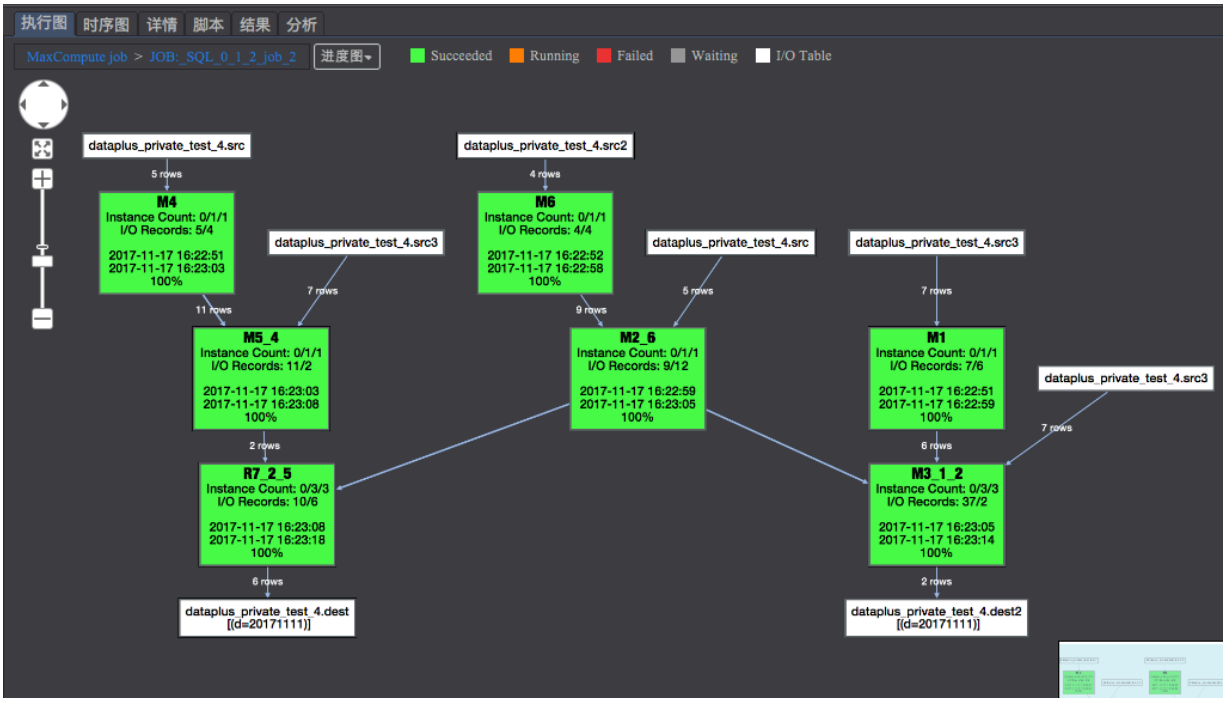
- 脚本模式适合用来改写本来要用层层嵌套子查询的单个语句，或者因为脚本复杂性而不得不拆成多个语句的脚本。
- 如果多个输入的数据源数据准备完成的时间间隔很长（例如一个凌晨1点可以准备好，一个上午7点可以准备好），则不适合通过table variable衔接拼装为一个大的脚本模式SQL。

使用MaxCompute Studio执行脚本模式

使用MaxCompute Studio脚本模式，首先请保证MaxCompute Studio完成安装、添加项目链接、建立MaxCompute SQL脚本文件，详情请参见IntelliJ IDEA安装步骤、项目空间连接管理、创建MaxCompute Script Module。编辑脚本页面如下。



脚本编译后提交运行，查看执行计划图。虽然脚本上是多个语句，但执行计划图是一个相通的DAG图。



通过客户端 （odpscmd）提交脚本模式

您需要使用0.27上版本的odpscmd提交脚本。建议您下载安装最新版本。安装后，请使用 `-s` 参数提交脚本。

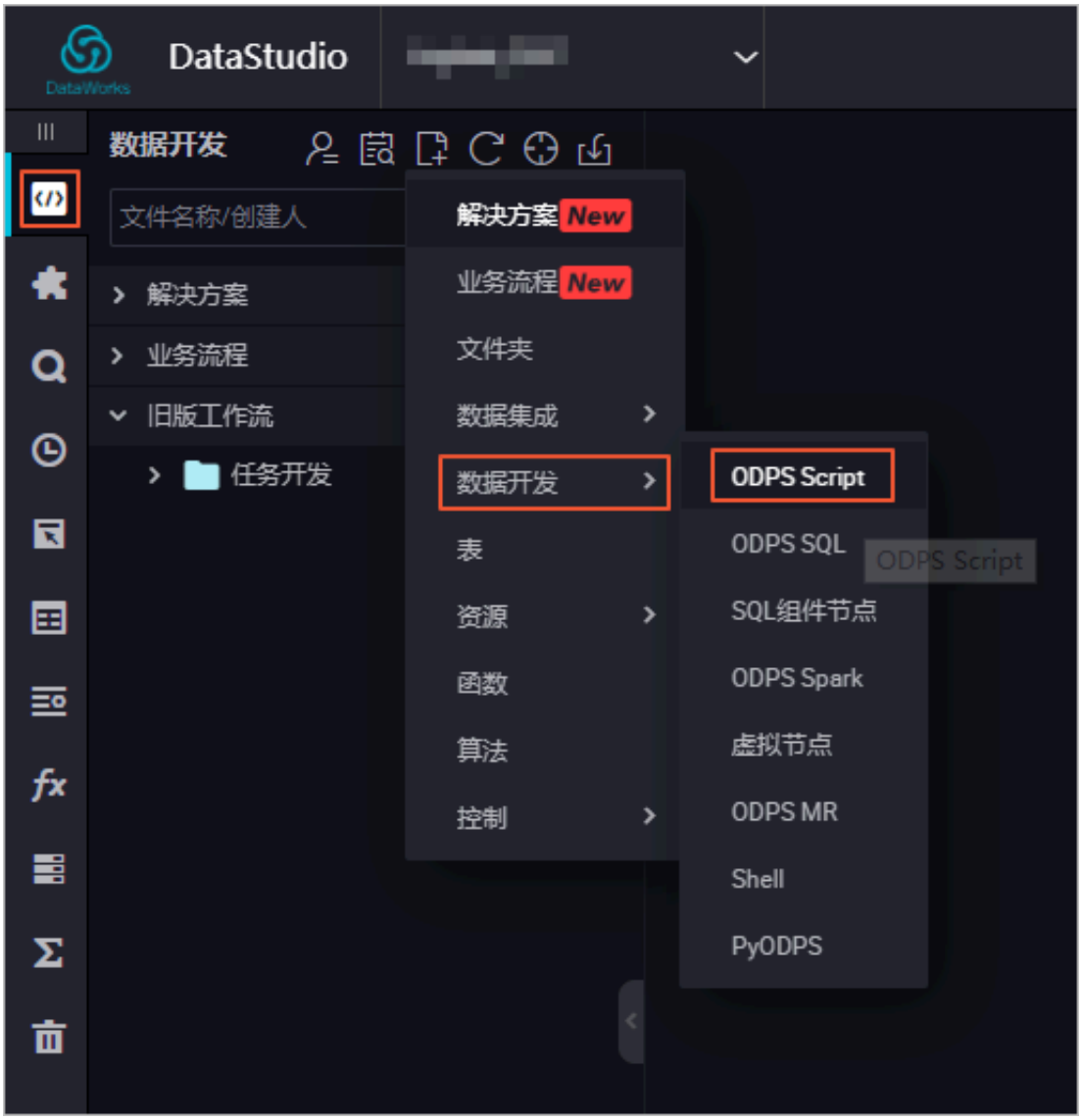
编辑脚本模式的源码*myscript.sql*文件，调用odpscmd命令执行如下。

```
odpscmd -s myscript.mxql;
```

> **说明:**
> `-s` 为odpscmd的命令行选项，类似于`-f`、`-e`，而非交互环境中的命令。odpscmd的交互环境中暂不支持脚本模式与表变量。

通过DataWorks使用脚本模式

在DataWorks中可以建立脚本模式的节点，如下图所示。



在此节点中进行脚本模式的脚本编辑，编辑好后点击工具栏的运行按钮，提交脚本到MaxCompute执行。您从输出信息的`logview url`中可以查看到执行计划图和结果。

4.11 COMMON TABLE EXPRESSION（CTE）

MaxCompute支持SQL标准的CTE，提高SQL语句的可读性与执行效率。

命令格式如下所示。

```
WITH
     cte_name AS
    (
        cte_query
    )
    [,cte_name2  AS
     (
     cte_query2
     )
```

```
    ,……]
```

- cte_name：CTE的名称，不能与当前WITH子句中的其他CTE的名称相同。查询中任何使用到cte_name标识符的地方，均指CTE。
- cte_query：一个SELECT语句。它产生的结果集用于填充CTE。

示例如下。

```
INSERT OVERWRITE TABLE srcp PARTITION (p='abc')
SELECT * FROM (
    SELECT a.key, b.value
    FROM (
        SELECT * FROM src WHERE key IS NOT NULL    ) a
    JOIN (
        SELECT * FROM src2 WHERE value > 0    ) b
    ON a.key = b.key
) c
UNION ALL
SELECT * FROM (
    SELECT a.key, b.value
    FROM (
        SELECT * FROM src WHERE key IS NOT NULL    ) a
    LEFT OUTER JOIN (
        SELECT * FROM src3 WHERE value > 0    ) b
    ON a.key = b.key AND b.key IS NOT NULL
)d;
```

顶层的UNION两侧各为一个JOIN，JOIN的左表是相同的查询。通过写子查询的方式，只能重复这段代码。

使用CTE的方式重写以上语句。

```
with
  a as (select * from src where key is not null),
  b as (select  * from src2 where value>0),
  c as (select * from src3 where value>0),
  d as (select a.key,b.value from a join b on a.key=b.key),
  e as (select a.key,c.value from a left outer join c on a.key=c.key
and c.key is not null)
insert overwrite table srcp partition (p='abc')
select * from d union all select * from e;
```

重写后，a对应的子查询只需写一次，便可在后面进行重用。您可以在CTE的WITH子句中指定多个子查询，像使用变量一样在整个语句中反复重用。除重用外，不必反复嵌套。

4.12 EXPLAIN

本文为您介绍MaxCompute SQL提供的EXPLAIN操作。该操作用于显示对应的DML语句执行计划的结构（执行计划是指用于执行SQL语义的程序）。

命令格式如下。

```
EXPLAIN <DML query>;
```

EXPLAIN的执行结果包含：

- 对应于该DML语句的所有Task的依赖结构。
- Task中所有Task的依赖结构。
- Task中所有Operator的依赖结构。

示例如下。

```
EXPLAIN
SELECT abs(a.key), b.value FROM src a JOIN src1 b ON a.value = b.value
;
```

EXPLAIN的输出结果会有以下三个部分：

- 首先是Job间的依赖关系：`job0 is root job`。因为该Query只需要一个Job（`job0`），所以只需要一行信息。
- 其次是Task间的依赖关系：

  ```
  In Job job0:
  root Tasks: M1_Stg1, M2_Stg1
  J3_1_2_Stg1 depends on: M1_Stg1, M2_Stg1
  ```

  `job0`包含三个Task。`M1_Stg1`和`M2_Stg1`这两个Task会先执行，执行完成后，再执行`J3_1_2_Stg1`。

  Task的命名规则如下：

  - 在MaxCompute中，共有四种Task类型：MapTask、ReduceTask、JoinTask和LocalWork。
  - Task名称的第一个字母表示了当前Task的类型，例如`M2Stg1`就是一个MapTask。
  - 紧跟着第一个字母后的数字，代表了当前Task的ID。这个ID在所有对应当前Query的Task中是唯一的。
  - 用下划线分隔的数字代表当前Task的直接依赖，例如`J3_1_2_Stg1`表示当前Task（ID为3）依赖ID为1、2的两个Task。

· 第三部分即Task中的Operator结构，Operator串描述了一个Task的执行语义。

```
In Task M1_Stg1:
  Data source: yudi_2.src                          # "Data source"描述了
当前Task的输入内容。
  TS: alias: a                                     # TableScanOperator
      RS: order: +                                 # ReduceSinkOperator
          keys:
               a.value
          values:
               a.key
          partitions:
               a.value
In Task J3_1_2_Stg1:
  JOIN: a INNER JOIN b                             # JoinOperator
      SEL: Abs(UDFToDouble(a._col0)), b._col5      # SelectOperator
          FS: output: None                         # FileSinkOperator
In Task M2_Stg1:
  Data source: yudi_2.src1
  TS: alias: b
      RS: order: +
          keys:
               b.value
          values:
               b.value
          partitions:
               b.value
```

- 各Operator的含义如下：
  - TableScanOperator：描述了Query语句中的FROM语句块的逻辑。EXPLAIN结果中会显示输入表的名称（Alias）。
  - SelectOperator：描述了Query语句中的SELECT语句块的逻辑。EXPLAIN结果中会显示向下一个Operator传递的列，多个列由逗号分隔。
    - 如果是列的引用，则显示成< alias >.< column_name >。
    - 如果是表达式的结果，则显示函数的形式，例如func1(arg1_1, arg1_2, func2(arg2_1, arg2_2))。
    - 如果是常量，则直接显示值内容。
  - FilterOperator：描述了Query语句中的WHERE语句块的逻辑。EXPLAIN结果中会显示一个WHERE条件表达式，形式类似SelectOperator的显示规则。
  - JoinOperator：描述了Query语句中的JOIN语句块的逻辑。EXPLAIN结果中会显示哪些表用哪种方式Join在一起。
  - GroupByOperator：描述了聚合操作的逻辑。如果Query中使用了聚合函数，就会出现该结构，EXPLAIN结果中会显示聚合函数的内容。
  - ReduceSinkOperator：描述了Task间数据分发操作的逻辑。如果当前Task的结果会传递给另一个Task，则必然需要在当前Task的最后，使用ReduceSinkOperator执行数据

分发操作。EXPLAIN的结果中会显示输出结果的排序方式、分发的Key、Value以及用来求Hash值的列。

- FileSinkOperator：描述了最终数据的存储操作。如果Query中有INSERT语句块，EXPLAIN结果中会显示目标表名称。
- LimitOperator：描述了Query语句中的LIMIT语句块的逻辑。EXPLAIN结果中会显示LIMIT数。
- MapjoinOperator：类似JoinOperator，描述了大表的JOIN操作。

说明：

如果Query足够复杂，EXPLAIN的结果太多，则会触发API的限制，无法得到完整的EXPLAIN结果。此时您可以拆分Query，对各部分分别进行EXPLAIN操作，以了解Job的结构。

4.13 VALUES

本文向您介绍INSERT…VALUES命令操作。

通常在业务测试阶段，需要创建一些数据进行简单的测试：

- 如果是为一个小表创建几条或十几条数据，您可以使用INSERT … VALUES方法在测试表中快速写入数据。
- 如果数据量较大，您可以通过Tunnel上传一个*txt*或*csv*数据文件，详情请参见导入数据；或者通过DataWorks的导入功能快速导入一个数据文件，详情请参见界面功能点介绍。

说明：

目前INSERT OVERWRITE不支持这种指定插入列的功能，只能使用INSERT INTO实现。

命令格式如下。

```
INSERT INTO TABLE tablename
[PARTITION (partcol1=val1, partcol2=val2,...)][(co1name1,colname2
,...)]
[VALUES (col1_value,col2_value,...),(col1_value,col2_value,...),...]
```

特定分区内插入数据

示例

```
drop table if exists srcp;
create table if not exists srcp (key string,value bigint) partitioned
by (p string);
```

```
insert into table srcp partition (p='abc') values ('a',1),('b',2),('c
',3);
```

INSERT … VALUES语句执行成功后，查询表srcp分区p=abc，结果如下。

```
+-----+-----------+---+
| key | value     | p |
+-----+-----------+---+
| a   | 1         | abc |
| b   | 2         | abc |
| c   | 3         | abc |
+-----+-----------+---+
```

非特定分区内插入数据

当表有很多列，而准备数据的时候希望只插入部分列的数据，此时可以使用插入列表功能。

示例

```
drop table if exists srcp;
create table if not exists srcp (key string,value bigint) partitioned
by (p string);
insert into table srcp partition (p)(key,p) values ('d','20170101'),('
e','20170101'),('f','20170101');
```

INSERT … VALUES语句执行成功后，查询表srcp分区p=20170101，结果如下。

```
+-----+-----------+---+
| key | value     | p |
+-----+-----------+---+
| d   | NULL      | 20170101 |
| e   | NULL      | 20170101 |
| f   | NULL      | 20170101 |
+-----+-----------+---+
```

说明:

- 目前INSERT OVERWRITE不支持这种指定插入列的功能，只能使用INSERT INTO实现。
- 对于在VALUES中没有指定的列，取缺省值NULL。插入列表功能不一定和VALUES一起用，对于INSERT INTO...SELECT...，同样可以使用。

VALUES TABLE功能

VALUES TABLE并不限于在INSERT语句中使用，任何DML语句都可以使用。

INSERT … VALUES有一个限制：VALUES必须是常量。但是当需要在插入的数据中进行一些简单的运算时，可使用MaxCompute的VALUES TABLE功能。

示例

```
drop table if exists srcp;
create table if not exists srcp (key string,value bigint) partitioned
by (p string);
```

```
insert into table srcp partition (p) select concat(a,b), length(a)+
length(b),'20170102' from values ('d',4),('e',5),('f',6) t(a,b);
```

INSERT … VALUES语句执行成功后，查询表srcp分区p=20170102，结果如下。

```
+-----+------------+---+
| key | value      | p |
+-----+------------+---+
| d4  | 2          | 20170102 |
| e5  | 2          | 20170102 |
| f6  | 2          | 20170102 |
+-----+------------+---+
```

说明:

其中的values (…), (…) t(a, b)相当于定义了一个名为t，列为a、b的表，类型分别为STRING、BIGINT，其中的类型从VALUES列表中推导。这样在不准备任何物理表时，您可以模拟一个有任意数据的、多行的表，并进行任意运算。

VALUES TABLE这个用法还可以取代select * from dual与union all组合的方式，拼出常量表，如下所示。

```
select 1 c from dual
union all
select 2 c from dual;
--等同于以下语句。
select * from values (1), (2) as t(c);
```

还有一种VALUES表的特殊形式，如下所示。

```
select abs(-1), length('abc'), getdate();
```

如上述语句所示，您可以不写FROM语句，直接执行SELECT（只要SELECT表达式列表中不出现上游表的数据）。其底层实现为从一个1行0列的匿名VALUES表中进行选取操作。这样，在您试图测试UDF或其它函数时，可免去手工创建DUAL表的过程。

说明:

- VALUES只支持常量不支持函数，但是ARRAY等复杂类型无法构造对应的常量，此时您可以通过以下语句实现在VALUES中使用ARRAY类型。

```
insert into table srcp (p='abc') select 'a', array('1', '2', '3');
```

- 通过VALUES写入DATETIME、TIMESTAMP类型，需要在VALUES中指定类型名称，如下所示。

```
insert into table srcp (p='abc') values (datetime'2017-11-11 00:00:
00',timestamp'2017-11-11 00:00:00.123456789');
```

4.14 参数化视图

本文为您介绍MaxCompute当前的SQL引擎支持的参数化视图功能。

MaxCompute传统的视图（VIEW）中，底层可以封装一段逻辑复杂的SQL脚本，调用者可以像读普通表一样调用视图，无需关心底层的实现。传统的视图实现了一定的封装与重用，因此被广泛地使用。但是传统的视图并不接受调用者传递的任何参数（例如调用者无法对视图读取的底层表进行数据过滤或传递其它参数），导致代码重用能力低下。MaxCompute当前的SQL引擎支持带参数的视图，支持传入任意表或者其它变量，定制视图的行为。

参数化视图定义

创建带参数的视图，语法如下。

```
-- view with parameters
-- param @a -a table parameter
-- param @b -a string parmeter
-- returns a table with schema (key string,value string)

create view if not exists pv1(@a table (k string,v bigint), @b string
)
as
select srcp.key,srcp.value from srcp join @a on srcp.key=a.k and srcp.
p=@b;
```

语法说明：

- 因为定义了参数，所以需要通过脚本模式操作。
- 创建的视图pv1有两个参数，即表参数和STRING参数，参数可以是任意的表或者基本类型。
- 支持使用子查询作为参数的值，例如select * from view_name((select 1 fromsrc where a > 0), 1);。
- 定义视图时，参数可以使用ANY关键字，表示任意类型。例如CREATE VIEW paramed_view (@a ANY) AS SELECT * FROM src WHERE case when @a is null then key1 else key2 end = key3;，定义了视图的第一个参数可以接受任意类型。但是ANY类

型不能参与如+、AND之类的需要明确类型才能做的运算。ANY类型通常用在TABLE参数中做PassThrough列，示例如下。

```
CREATE VIEW paramed_view (@a TABLE(name STRING, id ANY, age BIGINT
)) AS SELECT * FROM @a WHRER name = 'foo' and age < 25;
-- 调用示例。
SELECT * FROM param_view((SELECT name, id, age from students));
```

说明:

CREATE VIEW创建了视图后，可以用DESC来获取视图的描述。这个描述会包含视图的返回类型信息。视图的返回类型是在调用时重新推算的，它可能与创建视图时推导出来的不一致，例如ANY类型。

- 定义视图时，Table值参数还支持使用*表示任意多个列。这个*可以带类型，也可以使用ANY类型，示例如下。

```
CREATE VIEW paramed_view (@a TABLE(key STRING, * ANY), @b TABLE(key
STRING, * STRING)) AS SELECT a.* FROM @a JOIN @b ON a.key = b.key;
-- 调用示例。
SELECT name, address FROM param_view((SELECT school, name, age,
address FROM student), school) WHERE age < 20;
```

示例中的视图接受两个表值参数。第一个表值参数第一列是STRING类型，后面可以是任意多个任意类型的列；第二个表值参数的第一列是STRING类型，后面可以是任意多个STRING类型的列。注意事项如下：

- 变长部分必须写在表值参数定义语句的最后位置，即在*的后面不允许再出现其它列。因此，一个表值参数中最多只有一个变长列列表。
- 由于变长部分必须写在表值参数定义语句的最后位置，有时输入表的列不一定是按照这种顺序排列的，这时需要重排输入表的列，可以以子查询作为参数（参考上述示例），子查询外面必须加一层括号。
- 因为表值参数中变长部分没有名字，所以在视图定义过程中无法获得对这部分数据的引用，也无法对这些数据做运算。
- 虽然无法对变长部分做运算，但可以使用SELECT *这种通配符将变长部分的列传递出去。
- 表值参数的列与视图声明时指定的定长列部分不一定会完全一致。如果名字不一致，编译器会自动做重命名；如果类型不一致，编译器会做隐式转换（不能隐式转换时，会发生报错）。

调用参数化视图

调用上述定义中的pv1视图的示例如下。

```
@a := select * from src where value >0;
--call view with table varable an scalar
```

```
@b := select * from pv1(@a,'20170101');
@another_day := '20170102';
--call view with table name and scalar variable
@c := select * from pv1(src2, @another_day);
@d := select * from @c union all select * from @b;
with
t as(select * from src3)
select * from @c
union all
select * from @d
union all
select * from pv1(t,@another_day);
```

说明:

您可以使用不同的参数调用pv1。表参数可以是物理表、VIEW、表变量或者CTE中的表别名。普通参数可以是变量或者常量。

参数化视图的其它说明

- **参数化视图时，脚本中只能使用DML语句，不能使用insert或create table语句，也不能使用屏幕显示语句。**
- **参数化视图不一定只有一个SQL语句，也可以像脚本一样，包含多个语句。**

```
-- view with parameters
-- param @a -a table parameter
-- param @b -a string parmeter
-- returns a table with schema (key string,value string)
create view if not exists pv2(@a table (k string,v bigint), @b
string) as
BEGIN
@srcp := select * from srcp where p=@b;
@pv2 := select srcp.key,srcp.value from @srcp join @a on srcp.key=a.
k;
end;
```

说明:

begin到end之间的语句，就是这个视图的脚本。@pv2 :=...语句相当于其它语言中的return语句，方式为向一个与视图同名的隐含的表变量赋值。

- **在视图参数匹配时，实参和形参匹配的规则和普通的弱类型语言一样，如果传入的视图参数可以被隐式转换，则可与所定义的参数匹配，例如BIGINT的值可以匹配DOUBLE类型的参数。对于表变量，如果表a的Schema可以被用于插入到表b中，则意味着表a可以用来匹配和表b的Schema相同的表类型参数。**
- **您可以明确地声明返回类型，以提升代码的可读性。**

```
create view if not exists pv3(@a table (k string,v bigint), @b
string)
returns @ret table (x string,y string)
AS
begin
```

```
    @srcp := select * from srcp where p=@b;
    @ret := select srcp.key,srcp.value from @srcp join @a on srcp.
key=a.k;
end;
```

说明:

`returns @ret table (x string,y string)`**定义了:**

- **返回类型为**`table (x string, y string)`**，即返回给调用者的类型，可以在此处定制表的Schema。**
- **返回参数为**`@ret`**，为它赋值的操作会在视图的脚本中进行，此处相当于定义了返回的参数名。**

实际上，您可以把没有`begin/end`**和返回变量的视图，看作是此形式的简化形式。**

4.15 内建函数

4.15.1 与MySQL、Oracle内建函数对照表

本文帮助您根据MySQL、Oracle内建函数查找对应的MaxCompute内建函数。

函数类型	MaxCompute	HIVE	MySQL	Oracle
日期函数	DATEDIFF	DATEDIFF	DATEDIFF	MONTHS_BETWEEN
	DATEADD	DATE_ADD	DATE_ADD	无
	DATEPART	无	DATE_FORMAT	EXTRACT (datetime)
	DATETRUNC	TRUNC	DATE_FORMAT	EXTRACT (datetime)
	FROM_UNIXTIME	FROM_UNIXTIME	FROM_UNIXTIME	无
	GETDATE	CURRENT_DATE	NOW	CURRENT_DATE
	ISDATE	无	STR_TO_DATE（返回false则说明不能转换为date）	无
	LASTDAY	LAST_DAY	LAST_DAY	LAST_DAY
	TO_DATE	TO_DATE	STR_TO_DATE()	DATE
	TO_CHAR	无	DATE_FORMAT	TO_CHAR (datetime)

	UNIX_TIMESTAMP	UNIX_TIMESTAMP	UNIX_TIMESTAMP	无
	WEEKDAY	无	WEEKDAY	无
	WEEKOFYEAR	WEEKOFYEAR	WEEKOFYEAR	无
	YEAR	YEAR	YEAR	YEAR
	QUARTER	QUARTER	QUARTER	QUARTER
	MONTH	MONTH	MONTH	MONTH
	DAY	DAY	DAY	DAY
	DAYOFMONTH	DAYOFMONTH	DAYOFMONTH	无
	HOUR	HOUR	HOUR	HOUR
	MINUTE	MINUTE	MINUTE	MINUTE
	CURRENT_TIMESTAMP	CURRENT_TIMESTAMP	CURRENT_TIMESTAMP	CURRENT_TIMESTAMP
	ADD_MONTHS	ADD_MONTHS	ADDDATE	ADD_MONTHS
	LAST_DAY	LAST_DAY	LAST_DAY	无
	NEXT_DAY	NEXT_DAY	无	NEXT_DAY
	MONTHS_BETWEEN	MONTHS_BETWEEN	timestampdiff	MONTHS_BETWEEN
数学函数	ABS	ABS	ABS	ABS
	ACOS	ACOS	ACOS	ACOS
	ASIN	ASIN	ASIN	ASIN
	ATAN	ATAN	ATAN	ATAN
	CEIL	CEIL	CEIL	CEIL
	CONV	CONV	CONV	无
	COS	COS	COS	COS
	COSH	COSH	无	COSH
	COT	COT	COT	COT
	EXP	EXP	EXP	EXP
	FLOOR	FLOOR	FLOOR	FLOOR
	LN	LN	LN	LN
	LOG	LOG	LOG	LOG
	POW	POW	POW	POWER

	RAND	RAND	RAND	无
	ROUND	ROUND	ROUND	ROUND
	SIN	SIN	SIN	SIN
	SINH	SINH	无	SINH
	SQRT	SQRT	SQRT	SQRT
	TAN	TAN	TAN	TAN
	TANH	TANH	无	TANH
	TRUNC	TRUNC	TRUNCATE	TRUNC
	LOG2	LOG2	LOG2	LOG
	LOG10	LOG10	LOG10	LOG
	BIN	BIN	BIN	BITAND
	HEX	HEX	HEX	RAWTOHEX
	UNHEX	UNHEX	UNHEX	HEXTORAW
	RADIANS	RADIANS	RADIANS	RADIANS
	DEGREES	DEGREES	DEGREES	DEGREES
	SIGN	SIGN	SIGN	SIGN
	E	E	无	无
	PI	PI	PI	PI
	FACTORIAL	FACTORIAL	无	无
	CBRT	CBRT	无	无
	SHIFTLEFT	SHIFTLEFT	<<	无
	SHIFTRIGHT	SHIFTRIGHT	>>	无
	SHIFTRIGHTUNSIGNED	SHIFTRIGHT UNSIGNED	>>>	无
窗口函数	DENSE_RANK	DENSE_RANK	DENSE_RANK	DENSE_RANK
	RANK	RANK	RANK	RANK
	LAG	LAG	LAG	LAG
	LEAD	LEAD	LEAD	LEAD
	PERCENT_RANK	PERCENT_RANK	PERCENT_RANK	PERCENT_RANK
	ROW_NUMBER	ROW_NUMBER	ROW_NUMBER	ROW_NUMBER
	CLUSTER_SAMPLE	无	无	无

	NTILE	NTILE	NTILE	NTILE
聚合函数	COUNT	COUNT	COUNT	COUNT
	AVG	AVG	AVG	AVG
	MAX	MAX	MAX	MAX
	MIN	MIN	MIN	MIN
	MEDIAN	无	无	MEDIAN
	STDDEV	STDDEV	STDDEV	STDDEV
	STDDEV_SAMP	STDDEV_SAMP	STDDEV_SAMP	STDDEV_SAMP
	SUM	SUM	SUM	SUM
	WM_CONCAT	无	GROUP_CONCAT	WM_CONCAT
	COLLECT_LIST	COLLECT LIST	无	COLLECT
	COLLECT_SET	COLLECT SET	无	COLLECT
	VARIANCE/ VAR_POP	VARIANCE/ VAR_POP	VAR_POP	VARIANCE/ VAR_POP
	VAR_SAMP	VAR_SAMP	VAR_SAMP	VAR_SAMP
	COVAR_POP	COVAR_POP	无	COVAR_POP
	COVAR_SAMP	COVAR_SAMP	无	COVAR_SAMP
	PERCENTILE	PERCENTILE	无	无
字符串函数	CHAR_MATCHCOUNT	无	无	无
	CHR	CHR	CHAR	CHR
	CONCAT	CONCAT	CONCAT	CONCAT
	GET_JSON_OBJECT	GET_JSON_O BJECT	JSON_EXTRACT()	无
	INSTR	INSTR	INSTR	INSTR
	IS_ENCODING	无	无	无
	KEYVALUE	无	无	无
	LENGTH	LENGTH	LENGTH	LENGTH
	LENGTHB	LENGTHB	LENGTHB	LENGTHB
	MD5	MD5	MD5	无
	REGEXP_EXTRACT	REGEXP_EXT RACT	无	无
	REGEXP_INSTR	无	REGEXP_INSTR	REGEXP_INSTR

	REGEXP_REPLACE	REGEXP_REPLACE	REGEXP_REPLACE	REGEXP_REPLACE
	REGEXP_SUBSTR	无	REGEXP_SUBSTR	REGEXP_SUBSTR
	REGEXP_COUNT	无	无	REGEXP_COUNT
	SPLIT_PART	无	无	无
	SUBSTR	SUBSTR	SUBSTR	SUBSTR
	SUBSTRING	SUBSTRING	SUBSTRING	SUBSTR
	TOLOWER	LOWER	LOWER	LOWER
	TOUPPER	UPPER	UPPER	UPPER
	TRIM	TRIM	TRIM	TRIM
	LTRIM	LTRIM	LTRIM	LTRIM
	RTRIM	RTRIM	RTRIM	LTRIM
	REVERSE	REVERSE	REVERSE	REVERSE
	SPACE	SPACE	SPACE	SPACE
	REPEAT	REPEAT	REPEAT	REPEAT
	ASCII	ASCII	ASCII	ASCII
	CONCAT_WS	CONCAT_WS	CONCAT_WS	无
	LPAD	LPAD	LPAD	LPAD
	RPAD	RPAD	RPAD	RPAD
	REPLACE	无	REPLACE	REPLACE
	SOUNDEX	SOUNDEX	SOUNDEX	SOUNDEX
	SUBSTRING_INDEX	SUBSTRING_INDEX	SUBSTRING_INDEX	无
	TRANSLATE	TRANSLATE	无	TRANSLATE
	URL_DECODE	无	无	无
	URL_ENCODE	无	无	PERCENTILE_CONT
	CRC32	CRC32	CRC32	无
其他函数	CAST	CAST	CAST	CAST
	COALESCE	COALESCE	COALESCE	COALESCE

	DECODE	DECODE	无	DECODE
	GET_IDCARD_AGE	无	无	无
	GET_IDCARD_BIRTHDAY	无	无	无
	GET_IDCARD_SEX	无	无	无
	GREATEST	GREATEST	GREATEST	无
	ORDINAL	无	无	无
	LEAST	LEAST	LEAST	LEAST
	MAX_PT	无	无	无
	UUID	无	UUID	UID
	SAMPLE	无	无	无
	IF	IF	IF	IF
	CASE WHEN	CASE WHEN	CASE WHEN	CASE WHEN
	SPLIT	SPLIT	SPLIT	无
	STR_TO_MAP	STR_TO_MAP	无	无
	EXPLODE	EXPLODE	无	无
	MAP	MAP	无	无
	MAP_KEYS	MAP_KEYS	无	无
	MAP_VALUES	MAP_VALUES	无	无
	NVL	NVL	IFNULL	无
	ARRAY	ARRAY	无	无
	SIZE	SIZE	无	无
	ARRAY_CONTAINS	ARRAY_CONTAINS	无	无
	POSEXPLODE	POSEXPLODE	无	无
	TRANS_ARRAY	无	无	无
	INLINE	INLINE	无	无
	NAMED_STRUCT	无	无	无

4.15.2 日期函数

本文为您介绍MaxCompute SQL提供的日期函数。

DATEADD

- 命令格式

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

- 命令说明

  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。如果输入为STRING类型，则会隐式转换为DATETIME类型后参与运算，其它类型会抛异常。
  - delta：BIGINT类型，修改幅度。如果输入为STRING类型或DOUBLE型，则会隐式转换到BIGINT类型后参与运算，其他类型会引发异常。如果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、不支持的格式或其它类型会抛异常。

  > 说明:
  >
  > - 按照指定的单位增减delta时，会导致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、60进制进行计算。
  > - 当delta的单位是月时，如果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，否则将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。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。

- 返回值说明

  返回DATETIME类型。如果任一输入参数为NULL，则返回NULL。

- 示例

```
如果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。
如果trans_date = 2005-02-28 00:00:00, dateadd(transdate, 1, 'mm') =
2005-03-28 00:00:00
如果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。
如果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MaxCompute SQL中，DATETIME类型没有直接的常数表示方式，以下为错误的使用方式。

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

如果需要描述DATETIME类型常量，请参考以下用法。

```
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

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

- 命令说明

  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。如果输入为STRING类型，则会隐式转换为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。如果`datepart`不符合指定格式或者其它类型，则会发生异常。

  说明:

  计算时会按照`datepart`切掉低单位部分，然后再计算结果。

- 返回值说明

  返回BIGINT类型。如果任一输入参数为NULL，则返回NULL。如果`date1`小于`date2`，返回值可以为负数。

- 示例

```
如果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
如果start = 2018-06-04 19:33:23.234, end = 2018-06-04 19:33:23.250, 含毫秒的日期不属于标准DATETIME式样，不能直接隐式转换，此处需进行显示转换。
    datediff(to_date('2018-06-04 19:33:23.250', 'yyyy-MM-dd hh:mi:ss
.ff3'),to_date('2018-06-04 19:33:23.234', 'yyyy-MM-dd hh:mi:ss.ff3
') , 'ff3') = 16
```

DATEPART

- 命令格式

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

- 命令说明

  提取日期`date`中指定的时间单位`datepart`的值。

- 参数说明

  - `date`：DATETIME类型。如果输入为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。如果`datepart`不符合指定格式或者其它类型则会发生异常。

- 返回值说明

  返回BIGINT类型。如果任一输入参数为NULL，则返回NULL。

- 示例

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

DATETRUNC

- 命令格式

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

- 命令说明

  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。如果输入为STRING类型，则会隐式转换为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。如果`datepart`不符合指定格式或者其它类型则会发生异常。
- 返回值说明

  DATETIME类型。如果任一输入参数为NULL，则返回NULL。
- 示例

```
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
```

GETDATE

- 命令格式

```
datetime getdate()
```

- 命令说明

  获取当前系统时间。使用东八区时间作为MaxCompute标准时间。
- 返回值说明

  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

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

- 命令说明

  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。如果输入为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。例如`isdate("1234-yyyy", "yyyy-yyyy")`，会返回TRUE。
- 返回值说明

  返回BOOLEAN类型。如果任一输入参数为NULL，则返回NULL。

LASTDAY

- 命令格式

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

- 命令说明

  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。如果输入为STRING类型，则会隐式转换为DATETIME类型后参与运算，其它类型报异常。
- 返回值说明

  返回DATETIME类型。如果输入参数为NULL，则返回NULL。

TO_DATE

- 命令格式

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

- 命令说明

  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。如果输入为BIGINT、DOUBLE、DECIMAL或者DATETIME类型，则会隐式转换为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。例如`to_date("1234-2234", "yyyy-yyyy")` 会返回1234-01-01 00:00:00。

    `format`格式：yyyy为四位数的年，mm为2位数的月，dd为2位数的日，hh为24小时制的时，mi为2位数的分钟，ss为2位数秒，ff3为3位精度毫秒。
- 返回值说明

  返回DATETIME类型，格式为yyyy-mm-dd hh:mi:ss 。如果任一输入参数为NULL，则返回NULL。
- 示例

```
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') = null
-- 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') = null
-- 格式不符合，引发异常。
to_date('2010-24-01', 'yyyy') = null
-- 格式不符合，引发异常。
to_date('20181030 15-13-12.345','yyyymmdd hh-mi-ss.ff3')=2018-10-30 15:13:12
```

TO_CHAR

- 命令格式

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

- 命令说明

  将日期类型`date`按照`format`指定的格式转成字符串。

- 参数说明
  - `date`：DATETIME类型，要转换的日期值。如果输入为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。
- 返回值说明

  返回STRING类型。如果任一输入参数为NULL，则返回NULL。
- 示例

```
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') -- 会引发异常。
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

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

- 命令说明

  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。如果输入为STRING类型，则会隐式转换为DATETIME类型后参与运算，其它类型抛异常。当打开新数据类型Flag，隐式转换会失败，此时需要通过`case`函数转换，例如`unix_timestamp(cast(... as datetime))`。
- 返回值说明

  返回BIGINT类型。表示UNIX格式日期值，`date`为NULL时，返回NULL值。

- 示例

```
select unix_timestamp('2009-03-20 11:11:00'); --返回结果为1237518660。
```

FROM_UNIXTIME

- 命令格式

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

- 命令说明

  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。如果输入为STRING、DOUBLE、DECIMAL类型，则会隐式转换为BIGINT后参与运算。

- 返回值说明

  返回DATETIME类型的日期值。如果输入参数为NULL，则返回NULL。

  说明:
  HIVE兼容模式下（打开`set odps.sql.hive.compatible=true;` 后），为保持和HIVE兼容，如果您的入参是STRING类型，返回的也是STRING类型值。

- 示例

```
from_unixtime(123456789) = 1973-11-30 05:33:09
```

WEEKDAY

- 命令格式

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

- 命令说明

  返回`date`日期当前周的第几天。

- 参数说明

  `date`：DATETIME类型。如果输入为STRING类型，则会隐式转换为DATETIME类型后参与运算，其它类型抛异常。

- 返回值说明

  返回BIGINT类型。如果输入参数为NULL，则返回NULL。周一作为一周的第一天，返回值为0。其他日期依次递增，周日返回6。

WEEKOFYEAR

- 命令格式

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

- 命令说明

  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:
  这一周算上一年还是下一年，取决于这一周的大多数日期（4天以上）在哪一年。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。如果输入为STRING类型，则会隐式转换为DATETIME类型后参与运算，其它类型抛异常。

- 返回值说明

  返回BIGINT类型。如果输入参数为NULL，则返回NULL。

- 示例

```
select weekofyear(to_date("20141229", "yyyymmdd")) from dual;
--返回结果如下。
+------------+
| _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。
```

MaxCompute2.0扩展函数

升级到MaxCompute2.0后，产品扩展了部分日期函数。如果您用到的函数涉及新数据类型，在使用新函数的SQL前，需要加一个set语句。

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

如果您想同时提交，请执行以下语句。

```
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

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

- 命令说明

  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

```
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。
```

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用`set`命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上`set`语句`set odps.sql.type.system.odps2=true;`，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

对`setproject`的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

QUARTER

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回INT类型。如果输入参数为NULL，则返回NULL。

- 示例

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

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

MONTH

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回INT类型。

- 示例

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

说明：

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用`set`命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上`set`语句`set odps.sql.type.system.odps2=true;`，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

对`setproject`的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

DAY

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回INT类型。

- 示例

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

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用`set`命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上`set`语句`set odps.sql.type.system.odps2=true;`，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

对`setproject`的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

DAYOFMONTH

- 命令格式

```
INT dayofmonth(date)
```

- 命令说明

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。

- 返回值说明

返回INT类型。

- 示例

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

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

HOUR

- 命令格式

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

- 命令说明

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。

- 返回值说明

返回INT类型。

- 示例

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

说明：
目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。

- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

  对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

MINUTE

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回INT类型。

- 示例

  ```
  minute('2014-09-01 12:30:00') = 30
  minute('12:30:00') = 30
  minute('20140901120000') = null
  ```

说明：

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

  对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

SECOND

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回INT类型。

- 示例

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

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用`set`命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上`set`语句`set odps.sql.type.system.odps2=true;`，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

对`setproject`的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

CURRENT_TIMESTAMP

- 命令格式

```
timestamp current_timestamp()
```

- 命令说明

  返回当前TIMESTAMP类型的时间戳，值不固定。

- 返回值说明

  返回TIMESTAMP类型。

- 示例

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

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用`set`命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上`set`语句`set odps.sql.type.system.odps2=true;`，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

对`setproject`的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

FROM_UTC_TIMESTAMP

- 命令格式

```
timestamp from_utc_timestamp({any primitive type}*, string timezone)
```

- 命令说明

  将一个UTC时区的时间戳转换成一个指定时区的时间戳。

- 参数说明

  - `{any primitive type}*`：时间戳，类型包含TIMESTAMP、DATETIME、TINYINT、SMALLINT、INT、BIGINT。
  - `timezone`：指定需要转换的目标时区，例如PST。
  - 本函数只支持Asia、Shanghai这种格式，不支持GMT+9这种格式。

- 返回值说明

  返回TIMESTAMP类型。

- 示例

```
SELECT from_utc_timestamp(1501557840,'PST') ; --返回: 1970-01-18 09:
05:57.84。
SELECT from_utc_timestamp('1970-01-30 16:00:00','PST') ; --返回: 1970
-01-30 08:00:00.0。
SELECT from_utc_timestamp('1970-01-30','PST') ; --返回: 1970-01-29 16
:00:00.0。
```

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用`set`命令开启:

- **Session级别：要使用新数据类型，需在SQL语句前加上`set`语句`set odps.sql.type.system.odps2=true;`，并与SQL语句一起提交执行。**
- **Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为:**

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

  对`setproject`的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

ADD_MONTHS

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回STRING类型的日期，格式为yyyy-mm-dd。

- 示例

```
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
```

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用`set`命令开启：

- **Session级别：要使用新数据类型，需在SQL语句前加上`set`语句`set odps.sql.type.system.odps2=true;`，并与SQL语句一起提交执行。**
- **Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：**

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

对`setproject`的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

LAST_DAY

- **命令格式**

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

- **命令说明**

  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。**

- **返回值说明**

  返回STRING类型的日期，格式为yyyy-mm-dd。

- **示例**

```
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**

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

- **命令说明**

  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，例如MO、TUE、FRIDAY。
- 返回值说明

  返回STRING类型的日期，格式为yyyy-mm-dd。
- 示例

```
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

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

- 命令说明

  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。
- 返回值说明

  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`减去`date2`的天数除以31天。
- 示例

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

```
months_between('1996-09-30','1996-12-31') = -3.0
```

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- **Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。**
- **Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：**

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

对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

4.15.3 数学函数

本文向您介绍MaxCompute SQL支持的数学函数命令及说明。

ABS

- **命令格式**

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

- **命令说明**

  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类型。**
  - **输入为DOUBLE，返回DOUBLE类型。**
  - **输入为DECIMAL，返回DECIMAL类型。**
  - **输入为STRING，会隐式转换为DOUBLE类型后参与运算，其它类型抛异常。**

  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，但可能会损失精度。

- 返回值

  返回DOUBLE、BIGINT或DECIMAL类型，取决于输入参数的类型。如果输入参数为NULL，则返回NULL。

- 示例

```
abs(null)=null
abs(-1)=1
abs(-1.2)=1.2
abs("-2")=2.0
abs(12232083745629837659238745692 3748)=1.2232083745629837e32
```

下面是一个完整的abs函数在SQL中使用的示例，其他内建函数（除窗口函数、聚合函数外）的使用方式与其类似。

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

ACOS

- 命令格式

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

- 命令说明

  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。如果输入为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。

- 返回值说明

  返回DOUBLE类型或DECIMAL类型，值域在0~π之间。如果输入参数为NULL，则返回NULL。

- 示例

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

ASIN

- 命令格式

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

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

- 命令说明

  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。如果输入为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。

- 返回值说明

  返回DOUBLE类型或DECIMAL类型，值域在-π/2~π/2之间。如果输入参数为NULL，则返回NULL。

- 示例

```
asin(1)=1.5707963267948966
asin(-1)=-1.5707963267948966
```

ATAN

- 命令格式

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

- 命令说明

  该函数用于计算number的反正切函数。

- 参数说明

  number：DOUBLE类型。如果输入为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。

- 返回值说明

  返回DOUBLE类型，值域在-π/2~π/2之间。如果输入参数为NULL，则返回NULL。

- 示例

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

CEIL

- 命令格式

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

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

- 命令说明

  向上取整，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。如果输入为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。

- 返回值说明

  返回BIGINT类型。如果任一输入参数为NULL，则返回NULL。

- 示例

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

CONV

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回STRING类型。如果任一输入参数为NULL，则返回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

```
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

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

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

- 命令说明

  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。如果输入为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。

- 返回值说明

  返回DOUBLE类型或DECIMAL类型。如果输入参数为NULL，则返回NULL。

- 示例

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

COSH

- 命令格式

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

- 命令说明

  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。如果输入为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。

- 返回值说明

  返回DOUBLE类型或DECIMAL类型。如果输入参数为NULL，则返回NULL。

COT

- 命令格式

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

- 命令说明

  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。如果输入为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。

- 返回值说明

  返回DOUBLE类型或DECIMAL类型。如果输入参数为NULL，则返回NULL。

EXP

- 命令格式

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

- 命令说明

  该函数用于计算number的指数函数。

- 返回值说明

  返回number的指数值。

- 参数说明

  number：DOUBLE类型或DECIMAL类型。如果输入为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。

- 返回值说明

  返回DOUBLE类型或DECIMAL类型。如果输入参数为NULL，则返回NULL。

FLOOR

- 命令格式

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

- 命令说明

  向下取整，函数返回不大于number的最大整数值。

- 参数说明

  number：DOUBLE类型或DECIMAL类型。如果输入为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。

- 返回值说明

  返回BIGINT类型。如果输入参数为NULL，则返回NULL。

- 示例

```
floor(1.2)=1
floor(1.9)=1
floor(0.1)=0
floor(-1.2)=-2
floor(-0.1)=-1
floor(0.0)=0
floor(-0.0)=0
```

LN

- 命令格式

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

- 命令说明

  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。

  - 如果输入为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。
  - 如果输入参数为NULL，则返回NULL；如果输入参数为负数或0，则抛异常。

- 返回值说明

  返回DOUBLE类型或DECIMAL类型。

LOG

- 命令格式

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

- 命令说明

  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。如果输入为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。如果输入为STRING类型或BINGINT类型，会隐式转换为DOUBLE类型后参与运算，其他类型抛异常。

- 返回值说明

  返回DOUBLE类型或DECIMAL类型的对数值。

  - 如果任一输入参数为NULL，则返回NULL。
  - 如果任一输入参数为负数或0，则引发异常。
  - 如果base为1（会引发一个除零行为），也会引发异常。

POW

- 命令格式

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

- 命令说明

  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。如果输入为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。如果输入为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。

- 返回值说明

  返回DOUBLE类型或DECIMAL类型。如果任一输入参数为NULL，则返回NULL。

RAND

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回DOUBLE类型。

- 示例

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

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

ROUND

- 命令格式

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

- 命令说明

  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。如果输入为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。

- 返回值说明

  返回DOUBLE类型或DECIMAL类型。如果任一输入参数为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

```
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

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

- 命令说明

  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。如果输入为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。

- 返回值说明

  返回DOUBLE类型或DECIMAL类型。如果输入参数为NULL，则返回NULL。

SINH

- 命令格式

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

- 命令说明

  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。如果输入为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。

- 返回值说明

  返回DOUBLE类型或DECIMAL类型。如果输入参数为NULL，则返回NULL。

SQRT

- 命令格式

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

- 命令说明

  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。如果输入为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。

- 返回值说明

  返回DOUBLE类型或DECIMAL类型。如果输入参数为NULL，则返回NULL。

TAN

- 命令说明

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

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

- 命令说明

  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。如果输入为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。

- 返回值说明

  返回DOUBLE类型或DECIMAL类型。如果输入参数为NULL，则返回NULL。

TANH

- 命令格式

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

- 命令说明

  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。如果输入为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。

- 返回值说明

  返回DOUBLE类型或DECIMAL类型。如果输入参数为NULL，则返回NULL。

TRUNC

- 命令格式

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

- 命令说明

  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。如果输入为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。

- 返回值说明

  返回值类型为DOUBLE或DECIMAL类型。如果任一输入参数为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

```
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
```

MaxCompute2.0扩展函数

下文将为您详细介绍新扩展的函数。

LOG2

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回DOUBLE类型。如果输入参数为0或NULL，则返回NULL。

- 示例

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

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

LOG10

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回DOUBLE类型。如果输入参数为0或NULL，则返回NULL。

- 示例

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

BIN

- 命令格式

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

- 命令说明

  该函数用于返回`number`的二进制代码表示。

- 参数说明

  `number`：BIGINT类型。

- 返回值说明

  返回STRING类型。如果输入参数为0，则返回0；如果输入参数为NULL，则返回NULL。

- 示例

```
bin(0)='0'
bin(null)='null'
bin(12)='1100'
```

HEX

- 命令格式

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

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

- 命令说明

  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。

- 返回值说明

  返回STRING类型。如果输入参数为0，则返回0；如果输入参数为NULL，则会抛异常。

- 示例

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

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

UNHEX

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回BINARY类型。如果输入参数为0，则返回失败；如果输入参数为NULL，则返回NULL。

- 示例

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

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

RADIANS

- 命令格式

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

- 命令说明

  该函数用于将角度转换为弧度。

- 参数说明

  number：DOUBLE类型数据。

- 返回值说明

  返回DOUBLE类型。如果输入参数为NULL，则返回NULL。

- 示例

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

```
radians(null)=null
```

DEGREES

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回DOUBLE类型。如果输入参数为NULL，则返回NULL。

- 示例

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

SIGN

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回DOUBLE类型。如果输入参数为0，则返回0.0；如果输入参数为NULL，则返回NULL。

- 示例

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

```
sign(null)=null
```

E

- 命令格式

```
Double e()
```

- 命令说明

  该函数用于返回e的值。

- 返回值说明

  返回DOUBLE类型。

- 示例

```
e()=2.718281828459045
```

PI

- 命令格式

```
Double pi()
```

- 命令说明

  该函数用于返回π的值。

- 返回值说明

  返回DOUBLE类型。

- 示例

```
pi()=3.141592653589793
```

FACTORIAL

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回BIGINT类型。如果输入参数为0，则返回1；如果输入参数为NULL或[0，20]之外的值，则返回NULL。

- 示例

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

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用`set`命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上`set`语句`set odps.sql.type.system.odps2=true;`，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

对`setproject`的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

CBRT

- 命令格式

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

- 命令说明

  该函数用于计算`number`的立方根。

- 参数说明

  `number`：DOUBLE类型数据。

- 返回值说明

  返回DOUBLE类型。如果输入参数为NULL，则返回NULL。

- 示例

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

```
cbrt(null)=null
```

SHIFTLEFT

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回INT或BIGINT类型。

- 示例

```
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）。
```

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

SHIFTRIGHT

- 命令格式

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

```
Bigint shiftright(Bigint number1, Int number2)
```

- 命令说明

  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。

- 返回值说明

  返回INT或BIGINT类型。

- 示例

```
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）。
```

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

SHIFTRIGHTUNSIGNED

- 命令格式

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

- 命令说明

  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。
- 返回值说明

  返回INT或BIGINT类型。
- 示例

```
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)。
```

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

4.15.4 窗口函数

您可以在MaxCompute SQL中使用窗口函数，灵活地分析处理工作。

说明:

- 窗口函数只能出现在select子句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。
- 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。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。该限制适用于所有内建窗口函数。

  说明:
  例如group by UID，同1个UID对应的记录视为同一个窗口，即1个UID对应记录数最多包含1亿行数据（建议不超过500万行）。
- 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。

说明:

- x、y必须为大于等于0的整数常量，限定范围0~10000，值为0时表示当前行。您必须指定order by才可以用rows方式指定窗口范围。
- order by用来指定数据在一个窗口内如何排序。当遇到相同的order by值的时候，排序结果不稳定。为了减少随机性，应当尽可能保持order by值的唯一性。
- 并非所有的窗口函数都可以使用rows指定开窗方式。支持这种用法的窗口函数有AVG、COUNT、MAX、MIN、STDDEV和SUM。

COUNT

- 命令格式

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

- 命令说明

  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。

说明：
当指定distinct关键字时，不能使用order by。

- 返回值说明

  返回BIGINT类型。

- 示例

  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          |       -- 窗口为起始点。
| 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

```
avg([distinct] expr) over(partition by [col1, col2…]
```

```
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

- 命令说明

  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。

  > 说明：
  > 指明distinct关键字时，不能使用order by。

- 返回值说明

  返回DOUBLE类型。

MAX

- 命令格式

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

- 命令说明

  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。

  > 说明：
  > 指明distinct关键字时，不能使用order by。

- 返回值说明

  返回值的类型同expr类型。

MIN

- 命令格式

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

- 命令说明

  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。

  > 说明：
  > 指明distinct关键字时，不能使用order by。

- 返回值说明

  返回值类型同expr类型。

MEDIAN

- 命令格式

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

- 命令说明

  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。
  - 单`key`记录数限制为1000000。

- 返回值说明

  返回值类型同DOUBLE类型。

STDDEV

- 命令格式

```
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

  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。

      说明：

      - 当指定`distinct`关键字时，不能使用`order by`。
      - `stddev`还有一个别名函数`stddev_pop`，用法和`stddev`一样。
  - `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。
- 返回值说明

  输入值为DECIMAL类型时，返回DECIMAL类型，否则返回DOUBLE类型。
- 示例

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

STDDEV_SAMP

- 命令格式

```
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

  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。

      说明:
      指明`distinct`关键字时，不能使用`order by`。
  - `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。
- 返回值说明

  输入值为DECIMAL类型时，返回DECIMAL类型，否则返回DOUBLE类型。

SUM

- 命令格式

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

- 命令说明

  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。

  说明：
  指明`distinct`关键字时，不能使用`order by`。
- 返回值说明
  - 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。

DENSE_RANK

- 命令说明

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

- 命令说明

  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。
- 返回值说明

  返回BIGINT类型。
- 示例

  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

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

- 命令说明

  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。
- 返回值说明

  返回BIGINT类型。
- 示例

  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

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

- 命令说明

  按偏移量取当前行之前第几行的值。如果当前行号为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。

- 返回值说明

  返回值类型同expr类型。

- 示例

```
select seq, lag(seq+100, 1) over (partition by window order by seq)
as r from sliding_window;
--执行结果如下。
+------------+------------+
| seq        | r          |
+------------+------------+
| 0          | NULL       |
| 1          | 100        |
| 2          | 101        |
| 3          | 102        |
| 4          | 103        |
| 5          | 104        |
| 6          | 105        |
| 7          | 106        |
| 8          | 107        |
| 9          | 108        |
+------------+------------+
```

LEAD

- 命令格式

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

```
[order by [col1[asc|desc], col2[asc|desc]…]])
```

- 命令说明

  按偏移量取当前行之后第几行的值。如果当前行号为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。

- 返回值说明

  返回值类型同expr类型。

- 示例

```
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

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

- 命令说明

  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。

- 返回值说明

  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)。

ROW_NUMBER

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回BIGINT类型。

- 示例

  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

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

- 命令说明

  该函数用于分组抽样。

- 参数说明

  - x：BIGINT类型常量，x>=1。如果指定参数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。

- 返回值说明

  返回BOOLEAN类型。

- 示例

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

如果您需要从每组中抽取约10%的值，可以使用以下MaxCompute SQL完成。

```
select key, value
    from (
        select key, value, cluster_sample(10, 1) over(partition by
key) as flag
        from tbl
        ) sub
    where flag = true;
--执行结果如下。
+-----+------------+
| key | value      |
+-----+------------+
| groupa | 0.167537127858695 |
| groupb | 0.135751224393789 |
+-----+------------+
```

CUME_DIST

- 命令格式

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

- 命令说明

  求累计分布，相当于求分组中值小于等于当前值的行数占分组总行数的比例。

- 参数说明

  比较大小的值由`order by`列指定。

- 返回值

  分组中值小于等于当前值的行数占分组总行数的比例。

- 示例

  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，再求sal在同一组内的前百分之几。

```
SELECT deptno
, ename
, sal
, concat(round(cume_dist() OVER(PARTITION BY deptno ORDER BY sal
desc)*100,2),'%') as cume_dist
FROM emp;
--执行结果如下。
+------------+-------+------------+-----------+
| deptno     | ename | sal        | cume_dist |
+------------+-------+------------+-----------+
| 10         | JACCKA | 5000.0     | 33.33%    |
| 10         | KING  | 5000.0     | 33.33%    |
| 10         | CLARK | 2450.0     | 66.67%    |
| 10         | WELAN | 2450.0     | 66.67%    |
| 10         | TEBAGE | 1300.0     | 100.0%    |
| 10         | MILLER | 1300.0     | 100.0%    |
| 20         | SCOTT | 3000.0     | 40.0%     |
| 20         | FORD  | 3000.0     | 40.0%     |
| 20         | JONES | 2975.0     | 60.0%     |
| 20         | ADAMS | 1100.0     | 80.0%     |
| 20         | SMITH | 800.0      | 100.0%    |
| 30         | BLAKE | 2850.0     | 16.67%    |
| 30         | ALLEN | 1600.0     | 33.33%    |
| 30         | TURNER | 1500.0     | 50.0%     |
| 30         | MARTIN | 1250.0     | 83.33%    |
| 30         | WARD  | 1250.0     | 83.33%    |
| 30         | JAMES | 950.0      | 100.0%    |
+------------+-------+------------+-----------+
```

NTILE

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回BIGINT类型。

- 示例

  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          |
```

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

4.15.5 聚合函数

聚合（Aggregate）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

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

- 命令说明

  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`*`，即当`count(*)`时，返回所有行数。

- 返回值说明

  返回BIGINT类型。

- 示例

```
-- 例如表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`一起使用。例如表`test_src`存在以下两列数据（`key`为STRING类型，`value`为DOUBLE类型。

```
-- test_src的数据如下。
+-----+-------+
| key | value |
+-----+-------+
| a | 2.0 |
+-----+-------+
| a | 4.0 |
+-----+-------+
```

```
| b | 1.0 |
+-----+-------+
| b | 3.0 |
+-----+-------+
-- 此时执行如下语句，结果如下。
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

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

- 命令说明

  用于计算平均值。

- 参数说明

  `value`：DOUBLE类型或DECIMAL类型。如果输入为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

  如果输入DECIMAL类型，则返回DECIMAL类型。输入其他合法的类型时，会返回DOUBLE类型。

- 示例

```
-- 例如表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

```
max(value)
```

- 命令说明

  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。

- 返回值说明

  返回值的类型与`value`类型相同。

- 示例

```
-- 例如表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

```
MIN(value)
```

- 命令说明

  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

```
-- 例如表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

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

- 命令说明

  用于计算中位数。

- 参数说明

  `number`：DOUBLE类型或者DECIMAL类型。如果输入为STRING类型或BIGINT类型，会隐式转换到DOUBLE类型后参与运算。输入其他类型时会发生报错。

- 返回值说明

  返回DOUBLE类型或者DECIMAL类型。

- 示例

```
-- 例如表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

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

- 命令说明

  用于计算总体标准差。

- 参数说明

  number：DOUBLE类型或者DECIMAL类型。如果输入为STRING类型或BIGINT类型，会隐式转换到DOUBLE类型后参与运算。输入其他类型时会发生报错。

- 返回值说明

  返回DOUBLE类型或者DECIMAL类型。

- 示例

```
-- 例如表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

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

- 命令说明

  用于计算样本标准差。

- 参数说明

  number：DOUBLE类型或者DECIMAL类型。如果输入为STRING类型或BIGINT类型会隐式转换到DOUBLE类型后参与运算。输入其他类型时会发生报错。

- 返回值说明

  返回DOUBLE类型或者DECIMAL类型。

- 示例

```
-- 例如表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

```
sum(value)
```

- 命令说明

  用于计算汇总值。

- 参数说明

  value：DOUBLE、DECIMAL或BIGINT类型。如果输入为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。

- 返回值说明

  输入BIGINT类型时，返回BIGINT类型。输入为DOUBLE或STRING类型时，返回DOUBLE类型。

- 示例

```
-- 例如表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

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

- 命令说明

  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。如果输入为BIGINT，DOUBLE或者DATETIME类型，会隐式转换为STRING后参与运算。输入其他类型时会发生报错。
- 返回值说明

  返回STRING类型。

说明:

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

COLLECT_LIST

- 命令格式

```
ARRAY collect_list(col)
```

- 命令说明

  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:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用`set`命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上`set`语句`set odps.sql.type.system.odps2=true;`，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

对`setproject`的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

COLLECT_SET

- 命令格式

```
ARRAY collect_set(col)
```

- 命令说明

  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。

- 返回值说明

  返回ARRAY类型。

说明：

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

VARIANCE/VAR_POP

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回DOUBLE类型。

- 示例

  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:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

  对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

VAR_SAMP

- 命令格式

```
DOUBLE var_samp(col)
```

- 命令说明

  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。

- 返回值说明

  返回DOUBLE类型。

- 示例

  `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：

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用`set`命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上`set`语句`set odps.sql.type.system.odps2=true;`，并与SQL语句一起提交执行。

- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

  对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

COVAR_POP

- 命令格式

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

- 命令说明

  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

  test表（c1为BIGINT类型，c2为BIGINT类型）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：

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

  对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

COVAR_SAMP

- 命令格式

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

- 命令说明

  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

  test表（c1为BIGINT类型，c2为BIGINT类型）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：

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

  对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

PERCENTILE

- 命令格式

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

- 命令说明

  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

  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 |
  ```

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

select percentile(c1,array(0,0.3,0.5,1))from var_test;
--执行结果如下。
+------+
| _c0 |
+------+
| [8, 8.9, 9.5, 11] |
+------+
```

4.15.6 字符串函数

本文为您介绍MaxCompute支持的CHAR_MATCHCOUNT、CHR、CONCAT、GET_JSON_OBJECT、INSTR、IS_ENCODING等字符串函数。

CHAR_MATCHCOUNT

- 命令格式

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

- 命令说明

  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。

- 返回值说明

  返回BIGINT类型。如果任一输入参数为NULL，则返回NULL。

- 示例

```
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

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

- 命令说明

  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。如果输入为STRING类型或DOUBLE类型或DECIMAL类型，则会隐式转换到BIGINT类型后参与运算，其它类型抛异常。

- 返回值说明

  返回STRING类型。参数范围是0~255，超过此范围会引发异常。如果输入参数为NULL，则返回NULL。

CONCAT

- 命令格式

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

- 命令说明

  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

  输入参数为STRING类型。如果输入参数为BIGINT、DOUBLE、DECIMAL或者DATETIME类型，则会隐式转换为STRING后参与运算，其它类型抛异常。

- 返回值类型

  返回STRING类型。如果没有参数或者任一参数为NULL，则返回NULL。

- 示例

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

GET_JSON_OBJECT

- 命令格式

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

- 命令说明

  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。每次调用该函数时，都会读一次原始数据，因此反复调用可能会造成性能和费用的浪费。利用`GET_JSON_OBJECT`结合UDTF，您可以轻松转换JSON格式日志数据，避免多次调用函数，详情请参见利用MaxCompute内建函数及UDTF转换JSON格式日志数据。

- 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`$`开头。关于`path`的说明，请参见LanguageManual UDF。
    - `$`：表示根节点。
    - `.`：表示子节点。
    - `[]`：`[number]`表示数组下标，数组的格式为`key[sub1][sub2][sub3]……`。
    - `*`：`Wildcard for []`，返回整个数组。`*`不支持转义。
- 返回值说明
  - 如果`json`为空或者非法的`json`格式，则返回NULL。
  - 如果`json`合法，`path`也存在，则返回对应字符串。
  - 当前不支持一个Object中出现两次相同的Key，例如`{a:1, a:0}`，否则可能造成无法解析。
  - 当前不支持Emoji表情字符串。
- 示例一

```
+-----+
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

```
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

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

- 命令说明

  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，要搜索的字符串。如果输入为BIGINT、DOUBLE、DECIMAL或者DATETIME类型，则会隐式转换为STRING后参与运算，其它类型抛异常。
  - str2：STRING类型，要搜索的子串。如果输入为BIGINT、DOUBLE、DECIMAL或者DATETIME类型，则会隐式转换为STRING后参与运算，其它类型抛异常。
  - start_position：BIGINT类型，其它类型抛异常。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，则抛异常。

- 返回值说明

  - 返回BIGINT类型。
  - 如果在str1中未找到str2，则返回0。
  - 如果任一输入参数为NULL，则返回NULL。
  - 如果str2为空串，则总能匹配成功，例如instr(‘abc’, ‘’)会返回 1。

- 示例

```
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

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

- 命令说明

  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，空字符串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。
  - 如果任一输入参数为NULL，则返回NULL。

- 返回值说明

  返回BOOLEAN类型。如果str能够成功转换，则返回True，否则返回False。

- 示例

```
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

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

- 命令说明

  将srcStr（源字符串）按split1分成Key-Value对，并按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。
- 返回值说明
  - 返回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

```
keyvalue('0:1\;1:2', 1) = '2'
```

说明：

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拆分后变为如下。

```
0 1/
1 2
```

返回key为1所对应的Value值2。

示例二

```
keyvalue("\;decreaseStore:1\;xcard:1\;isB2C:1\;tf:21910\;cart:1\;
shipping:2\;pf:0\;market:shoes\;instPayAmount:0\;","\;",":","tf") =
 "21910" value:21910。
```

说明：

源字符串如下所示。

```
“\;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对如下所示。

```
decreaseStore:1, xcard:1, isB2C:1, tf:21910, cart:1, shipping:2, pf:0
, market:shoes, instPayAmount:0
```

按照split2":"拆分后，结果如下所示。

```
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为21910。

LENGTH

- 命令格式

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

- 命令说明

  返回字符串str的长度。

- 参数说明

  - str：STRING类型。如果输入为BIGINT、DOUBLE、DECIMAL或者DATETIME类型，则会隐式转换为STRING后参与运算，其它类型抛异常。
  - 返回BIGINT类型。如果输入参数为NULL，则返回NULL；如果输入参数为非UTF-8编码格式，则返回-1。

- 示例

```
length('hi! 中国') = 6
```

LENGTHB

- 命令格式

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

- 命令说明

  返回字符串str的以字节为单位的长度。

- 参数说明

  - str：STRING类型。如果输入为BIGINT、DOUBLE、DECIMAL或者DATETIME类型，则会隐式转换为STRING后参与运算，其它类型抛异常。
  - 返回BIGINT类型。如果输入参数为NULL，则返回NULL。

- 示例

```
lengthb('hi! 中国') = 10
```

MD5

- 命令格式

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

- 命令说明

  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，则会隐式转换成STRING类型参与运算，其它类型抛异常。

- 返回值说明

  返回STRING类型。如果输入参数为NULL，则返回NULL。

REGEXP_EXTRACT

- 命令格式

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

- 命令说明

  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，如果pattern为空串或者pattern中没有指定group，则抛异常。
  - occurrence：BIGINT类型常量，必须大于等于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。如果occurrence等于0，则返回满足整个pattern的子串。

  说明:
  数据以UTF-8格式保存。对于中文字符可以用对应的16进制编码表示，中文字符编码的范围是[\x{4e00}-\x{9fa5}]。
- 返回值说明

  返回STRING类型。如果任一输入参数为NULL，则返回NULL。
- 示例

```
regexp_extract('foothebar', 'foo(.*?)(bar)', 1) = the
regexp_extract('foothebar', 'foo(.*?)(bar)', 2) = bar
regexp_extract('foothebar', 'foo(.*?)(bar)', 0) = foothebar
regexp_extract("阿里巴巴", "([\\x{4e00}-\\x{9fa5}]+)", 1) = 阿里巴巴
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"\"作为转义字符。
regexp_extract('foothebar', 'foothebar')
-- 异常返回，pattern中没有指定group。
```

REGEXP_INSTR

- 命令格式

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

- 命令说明

  返回字符串source从start_position开始，和pattern第n次（nth_occurrence）匹配的子串的起始、结束位置。

- 参数说明
  - source：STRING类型，待搜索的字符串。
  - pattern：STRING类型常量，pattern为空串时抛异常。
  - start_position：BIGINT类型常量，搜索的开始位置。不指定时默认值为1，其它类型或值小于等于0时抛异常。
  - 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。其它类型或值小于等于0时抛异常。
  - 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。
- 返回值说明

  返回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。如果任一输入参数为NULL，则返回NULL。
- 示例

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

REGEXP_REPLACE

- 命令格式

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

- 命令说明

  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。

  说明:

  当引用不存在的组时，行为未定义。

- 返回值说明
  - 返回STRING类型。当引用不存在的组时，不进行替换。
  - 如果任一输入参数为NULL，则返回NULL。
  - 如果replace_string为NULL且pattern有匹配，则返回NULL。
  - 如果replace_string为NULL但pattern不匹配，则返回原串。
- 示例

```
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

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

- 命令说明

  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。
- 返回值说明

  返回STRING类型。如果任一输入参数为NULL，则返回NULL。没有匹配时也返回NULL。

- 示例

```
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

```
bigint regexp_count(string source, string pattern[, bigint
start_position])
```

- 命令说明

  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，其它类型抛异常。
  - pattern：STRING类型常量，要匹配的模型。pattern为空串时抛异常，其它类型抛异常。
  - start_position：BIGINT类型常量，必须大于0。其它类型或值小于等于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。

- 返回值说明

  返回BIGINT类型。如果没有匹配成功，则返回0；如果任一输入参数为NULL，则返回NULL。

- 示例

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

SPLIT_PART

- 命令格式

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

- 命令说明

  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，则会隐式转换到STRING类型后参加运算，其它类型抛异常。
  - 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，其它类型抛异常。
  - 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，非常量或其他类型抛异常。此参数可省略，缺省时表示最后一部分。
- 返回值说明
  - 返回STRING类型。
  - 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，start大于6，返回空串。
  - 如果separator不存在于str中，且start指定为1，返回整个str。如果str为空串，则输出空串。
  - 如果separator为空串，则返回原字符串str。
  - 如果end大于片段个数，按片段个数处理。
  - 如果任一输入参数为NULL，则返回NULL。
- 示例

```
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

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

- 命令说明

  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，如果输入为BIGINT、DECIMAL、DOUBLE或者DATETIME类型，则会隐式转换为STRING后参与运算，其它类型抛异常。
  - start_position：BIGINT类型，起始位置为1。当start_position为0时，返回空串。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，往前数依次为-2，-3…，其它类型抛异常。
  - length：BIGINT类型，表示子串的长度值。值大于0，其它类型或值小于等于0抛异常。
- 返回值说明

  返回STRING类型。如果任一输入参数为NULL，则返回NULL。

  说明:

  当length被省略时，返回到str结尾的子串。
- 示例

```
substr("abc", 2) = "bc"
substr("abc", 2, 1) = "b"
substr("abc",-2,2) = "bc"
substr("abc",-3) = "abc"
```

SUBSTRING

- 命令格式

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

- 命令说明

  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。如果输入为其它类型，则返回NULL或报错。
  - start_position：INT类型，起始位置为1。当start_position为0时，返回空串。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，最后一个字符为-1，往前数依次为-2，-3…，其它类型抛异常。
  - length：BIGINT类型，表示子串的长度。值大于0，其它类型或值小于等于0抛异常。
- 返回值说明

  返回STRING类型。如果任一输入参数为NULL，则返回NULL。

  说明:

当length被省略时，返回到str结尾的子串。

- 示例

```
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'
```

TOLOWER

- 命令格式

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

- 命令说明

  输出英文字符串source对应的小写字符串。

- 参数说明

  source：STRING类型。如果输入为BIGINT、DOUBLE、DECIMAL或者DATETIME类型，则会隐式转换为STRING后参与运算，其它类型抛异常。

- 返回值说明

  返回STRING类型。如果输入参数为NULL，则返回NULL。

- 示例

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

TOUPPER

- 命令格式

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

- 命令说明

  输出英文字符source串对应的大写字符串。

- 参数说明

  source：STRING类型。如果输入为BIGINT、DOUBLE、DECIMAL或者DATETIME类型，则会隐式转换为STRING后参与运算，其它类型抛异常。

- 返回值说明

  返回STRING类型。如果输入参数为NULL，则返回NULL。

- 示例

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

TO_CHAR

- 命令格式

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

- 命令说明

  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，请参见另一同名函数TO_CHAR。

- 返回值说明

  返回STRING类型。如果输入参数为NULL，则返回NULL。

- 示例

```
to_char(123) = '123'
to_char(true) = 'TRUE'
to_char(1.23) = '1.23'
to_char(null) = NULL
```

TRIM

- 命令格式

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

- 命令说明

  将输入字符串`str`去除左右空格。

- 参数说明

  `str`：STRING类型。如果输入为BIGINT、DECIMAL、DOUBLE或者DATETIME类型，则会隐式转换为STRING后参与运算，其它类型抛异常。

- 返回值说明

  返回STRING类型。如果输入参数为NULL，则返回NULL。

LTRIM

- 命令格式

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

- 命令说明

  将输入的字符串`str`去除左边空格。

- 参数说明

  `str`：STRING类型。如果输入为BIGINT、DECIMAL、DOUBLE或者DATETIME类型，则会隐式转换为STRING后参与运算，其它类型抛异常。

- 返回值说明

  返回为STRING类型。如果输入参数为NULL，则返回NULL。

- 示例

```
select ltrim(' abc ') from dual;
--执行结果如下。
+------+
| _c0 |
+------+
| abc |
+------+
```

RTRIM

- 命令格式

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

- 命令说明

  将输入的字符串`str`去除右边空格。

- 参数说明

  `str`：STRING类型。如果输入为BIGINT、DECIMAL、DOUBLE或者DATETIME类型，则会隐式转换为STRING后参与运算，其它类型抛异常。

- 返回值说明

  返回STRING类型。如果输入参数为NULL，则返回NULL。

- 示例

```
select rtrim('a abc ') from dual;
--执行结果如下。
+------+
| _c0 |
+------+
| a abc |
```

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

REVERSE

- 命令格式

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

- 命令说明

  返回倒序字符串。

- 参数说明

  `str`：STRING类型。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING后参与运算，其它类型抛异常。

- 返回值说明

  返回STRING类型。如果输入参数为NULL，则返回NULL。

- 示例

```
select reverse('abcedfg') from dual;
--执行结果如下。
+------+
| _c0 |
+------+
| gfdecba |
+------+
```

REPEAT

- 命令格式

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

- 命令说明

  返回重复`n`次后的`str`字符串。

- 参数说明

  - `str`：STRING类型。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING后参与运算，其它类型抛异常。
  - `n`：BIGINT类型。长度不超过2M。如果为空，则抛异常。

- 返回值说明

  返回STRING类型。

- 示例

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

ASCII

- 命令格式

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

- 命令说明

  返回字符串`str`第一个字符的ASCII码。

- 参数说明

  `str`：STRING类型。如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING后参与运算，其它类型抛异常。

- 返回值说明

  返回BIGINT类型。

- 示例

```
select ascii('abcde') from dual; --返回97。
```

MaxCompute 2.0扩展函数

升级到MaxCompute 2.0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，需加以下`set`语句。

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

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。

下文将为您详细介绍新扩展的字符串函数。

CONCAT_WS

- 命令格式

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

- 命令说明

  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。如果不指定，则返回值会异常。
- 返回值说明

  返回STRING类型。如果没有输入参数或者某个输入参数为NULL，则返回NULL。
- 示例

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

LPAD

- 命令格式

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

- 命令说明

  用b字符串将a字符串向左补足到len位。
- 参数说明
  - len：INT整型。
  - a、b等：STRING类型。
- 返回值说明

  返回STRING类型。如果len小于a的位数，则返回a从左开始截取len位字符；如果len为0，则返回空。
- 示例

```
lpad('abcdefgh',10,'12')='12abcdefgh'
lpad('abcdefgh',5,'12')='abcde'
lpad('abcdefgh',0,'12') --返回空
```

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。

- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

  对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

RPAD

- 命令格式

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

- 命令说明

  用b字符串将a字符串向右补足到len位。

- 参数说明

  - len：INT整型。
  - a、b等：STRING类型。

- 返回值说明

  返回STRING类型。如果len小于a的位数，则返回a从左开始截取len位字符；如果len为0，则返回空。

- 示例

  ```
  rpad('abcdefgh',10,'12')='abcdefgh12'
  rpad('abcdefgh',5,'12')='abcde'
  rpad('abcdefgh',0,'12') -- 返回空。
  ```

说明：

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。

- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

  对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

REPLACE

- 命令格式

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

- 命令说明

  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

  参数均为STRING类型。

- 返回值说明

  返回STRING类型。如果任一输入参数为NULL，则返回NULL。

- 示例

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

SOUNDEX

- 命令格式

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

- 命令说明

  将普通字符串转换成soundex字符串。

- 参数说明

  a为STRING类型。

- 返回值说明

  返回STRING类型。如果输入参数为NULL，则返回NULL。

- 示例

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

SUBSTRING_INDEX

- 命令格式

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

- 命令说明

  截取字符串a第count分隔符之前的字符串。如果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。

- 返回值说明

  返回STRING类型。如果任一输入参数为NULL，则返回NULL。

- 示例

```
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
```

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

  对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

TRANSLATE

- 命令格式

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

- 命令说明

  将str1出现在str2中的字符串替换成str3中的字符串。

- 返回值说明

  返回STRING类型。如果任一输入参数为NULL，则返回NULL。

- 示例

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

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

  对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

URL_DECODE

- 命令格式

```
STRING URL_DECODE(STRING input[, STRING encoding])
```

- 命令说明

  将输入字符串从application/x-www-form-urlencoded MIME格式转为正常字符串，是`url_encoding`的逆过程。

  - a-z、A-Z保持不变。
  - （.）、（-）、（*）和（_）保持不变。
  - （+）转为空格。
  - %xy格式的序列转为对应的字节值，连续的字节值根据输入的`encoding`名称解成对应的字符串。
  - 其余的字符保持不变。
  - 函数最终的返回值是UTF-8编码的字符串。

- 参数说明

  - `input`：要输入的字符串。
  - `encoding`：指定的编码格式，只支持GBK格式。

- 返回值说明

  返回STRING类型。如果输入参数为NULL，则返回NULL。

- 示例

```
url_decode('%E7%A4%BA%E4%BE%8Bfor+url_encode%3A%2F%2F+%28fdsf%29')=
 "示例for url_encode:// (fdsf)"
url_decode('Exaple+for+url_encode+%3A%2F%2F+dsf%28fasfs%29', 'GBK')
 = "Exaple for url_encode :// dsf(fasfs)"
```

说明:

`URL_DECODE`函数的`encoding`设置只支持GBK格式。

URL_ENCODE

- 命令格式

```
STRING URL_ENCODE(STRING input[, STRING encoding])
```

- 命令说明

  将输入字符串编码为application/x-www-form-urlencoded MIME格式。

  - a-z、A-Z保持不变。
  - （.）、（-）、（*）和（_）保持不变。
  - 空格转为（+）。
  - 其余字符根据指定的`encoding`转为字节值，然后将每个字节值表示为%xy的格式，xy是该字符值的十六进制表示方式。

- 参数说明

  - `input`：要输入的字符串。
  - `encoding`：指定的编码格式，只支持GBK格式。

- 返回值说明

  返回STRING类型。如果输入参数为NULL，则返回NULL。

- 示例

```
url_encode('示例for url_encode:// (fdsf)') = "%E7%A4%BA%E4%BE%8Bfor+
url_encode%3A%2F%2F+%28fdsf%29"
url_encode('Exaple for url_encode :// dsf(fasfs)', 'GBK') = "Exaple+
for+url_encode+%3A%2F%2F+dsf%28fasfs%29"
```

说明：

`URL_ENCODE`函数的`encoding`设置只支持GBK格式。

4.15.7 其他函数

本文为您介绍如何使用CAST、DECODE、LEAST、ARRAY、SPLIT、MAP等函数。

CAST

函数声明格式如下。

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

用途：将表达式的结果转换成目标类型，例如`cast('1' as bigint)`将字符串`'1'`转为整数类型的`1`，如果转换不成功或遇到不支持的类型转换，则会引发异常。

函数说明：

- `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，则会直接将它们转为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。
- 使用`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

函数声明格式如下。

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

用途：返回列表中第一个非NULL的值，如果列表中所有的值都是NULL，则返回NULL。

参数说明：

- `expr`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。
- 至少要有一个参数，否则会引发异常。

返回值说明：

- 返回值类型和参数类型相同。

DECODE

函数声明格式如下。

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

用途：实现`if-then-else`分支选择的功能。

参数说明：

- `expression`：要比较的表达式。
- `search`：与`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。

返回值说明：

- 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，否则会引发异常。
> - 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

```
select
decode(customer_id,
1, 'Taobao',
2, 'Alipay',
3, 'Aliyun',
Null, 'N/A',
'Others') as result
from sale_detail;
```

示例中的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

函数声明格式如下。

```
get_idcard_age(idcardno)
```

用途：根据身份证号返回当前的年龄。返回值为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。

返回值说明：

- 返回BIGINT类型。输入为NULL时，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

函数声明格式如下。

```
get_idcard_birthday(idcardno)
```

用途：根据身份证号返回出生日期。

参数说明：

- idcardno：STRING类型，15位或18位身份证号。在计算时，会根据省份代码以及最后一位校验码检查身份证的合法性。如果校验不通过，则返回NULL。

返回值说明：

- 返回DATETIME类型。输入为NULL时，返回NULL。

GET_IDCARD_SEX

函数声明格式如下。

```
get_idcard_sex(idcardno)
```

用途：根据身份证号返回性别，值为M（男）或F（女）。

参数说明：

- idcardno：STRING类型，15位或18位身份证号。在计算时会根据省份代码以及最后一位校验码检查身份证的合法性。如果校验不通过，则返回NULL。

返回值说明：

- 返回STRING类型。输入为NULL时，返回NULL。

GREATEST

函数声明格式如下。

```
greatest(var1, var2, ...)
```

用途：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。如果所有的参数值都为NULL，则返回NULL。

返回值说明：

- 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

函数声明格式如下。

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

用途：将输入变量按从小到大排序后，返回nth指定位置的值。

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。

返回值说明：

- 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为最小值。

示例

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

LEAST

函数声明格式如下。

```
least(var1, var2, …)
```

用途：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。如果所有参数值都为NULL，则返回NULL。

返回值说明：

- 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为最小值。

MAX_PT

函数声明格式如下。

```
max_pt(table_full_name)
```

用途：对于分区的表，此函数返回该分区表的一级分区的最大值，按字母排序，且该分区下有对应的数据文件。

参数说明：

- `table_full_name`：STRING类型，指定表名（必须带上Project名，例如*prj.src*），您必须对此表有读权限。

返回值说明：

- 返回最大的一级分区的值。

示例

例如`tbl`是分区表，该表对应的分区如下，且都有数据文件。

```
pt =‘20120901’
pt =‘20120902’
```

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

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

说明：

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

UUID

函数声明格式如下。

```
string TRING uuid()
```

用途：返回一个随机ID，形式为29347a88-1e57-41ae-bb68-a9edbdd94212。

说明：
UUID返回的是一个随机的全局ID，其重复的概率很小。

SAMPLE

函数声明格式如下。

```
boolean sample(x, y, column_name1,column_name2,...)
```

用途：对所有读入的column_name的值，sample根据x、y的设置做采样，并过滤掉不满足采样条件的行。

参数说明：

- x、y：BIGINT类型，表示哈希为x份，取第y份。y可省略，省略时取第一份，如果省略参数中的y，则必须同时省略column_name。x、y为整型常量，大于0，其它类型或小于等于0时抛异常，如果y>gt，x也抛异常。x、y任一输入为NULL时，返回NULL。
- column_name为采样的目标列。该参数可以省略，省略时将根据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。

返回值说明：

- 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，以获得较好的输出结果。

示例

例如存在表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提供以下两种`case when`的语法格式：

- ```
  case value
  when value1 then result1
  when value2 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，灵活地返回不同的值。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，则统一转为STRING后，再返回。如果无法进行类型转换（例如BOOLEAN类型无法转换为STRING类型），则会发生报错。
- 除此之外不允许其它类型之间的转换。

IF表达式

表达式语法如下。

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

用途：判断`testCondition`是否为真。如果为真，返回`valueTrue`，否则返回另一个值（`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。

返回值说明：

- 返回值类型和参数`valueTrue`或者`valueFalseOrNull`的类型一致。

示例

```
select if(1=2,100,200) from dual;
--结果如下。
+------------+
| _c0 |
+------------+
| 200 |
+------------+
```

MaxCompute 2.0扩展支持的其他函数类型

SPLIT

函数声明格式如下。

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

用途：通过`pat`将`str`分割后返回数组。

参数说明：

- `str`：STRING类型，指被分隔的字符串。
- `pat`：STRING类型的分隔符，支持正则。

返回值说明：

- 返回`array <STRING>`。元素是`str`被`pat`分隔后的结果。

示例

```
select split("a,b,c",",") from dual;
--结果如下。
+------+
| _c0 |
+------+
| [a, b, c] |
```

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

STR_TO_MAP

函数声明格式如下。

```
str_to_map(text [, delimiter1 [, delimiter2]])
```

用途：使用delimiter1将text分割成K-V对，然后使用delimiter2分割每个K-V对的K和V。

参数说明：

- text：STRING类型，指被分割的字符串。
- delimiter1：STRING类型，分隔符，不指定时默认为逗号（,）。
- delimiter2：STRING类型，分隔符，不指定时默认为句号（=。

返回值说明：

- 返回MAP<STRING, STRING>。元素是text被delimiter1和delimiter2分割后的K-V结果。

示例

```
select str_to_map('test1&1-test2&2','-','&');
--结果如下。
+-------------+
| a           |
+-------------+
| {test1:1, test2:2} |
```

EXPLODE

函数声明格式如下。

```
explode (var)
```

用途：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ARRAY，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。

返回值说明：

- 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

```
explode(array(null, 'a', 'b', 'c')) col
```

MAP

函数声明格式如下。

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

用途：使用给定的Key-Value对建立MAP。

参数说明：

- **所有key类型一致（包括隐式转换后类型一致），必须是基本类型。**
- **所有value类型一致（包括隐式转换后类型一致），可为任意类型。**

返回值说明：

- **返回MAP类型。**

示例

例如表t_table的字段为c1 BIGINT, c2 STRING, c3 STRING, c4 BIGINT, c5 BIGINT，数据如下。

```
+------------+----+----+------------+------------+
| c1         | c2 | c3 | c4         | c5         |
+------------+----+----+------------+------------+
| 1000       | k11 | k21 | 86         | 15         |
| 1001       | k12 | k22 | 97         | 2          |
| 1002       | k13 | k23 | 99         | 1          |
+------------+----+----+------------+------------+
```

执行SQL。

```
select  map(c2,c4,c3,c5) from t_table;
--结果如下。
+------+
| _c0  |
+------+
| {k11:86, k21:15} |
| {k12:97, k22:2} |
| {k13:99, k23:1} |
```

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

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- **Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。**
- **Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：**

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

  对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

MAP_KEYS

函数声明格式如下。

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

用途：将参数MAP中的所有Key作为数组返回。

参数说明：

- **参数为MAP类型的数据。**

返回值说明：

- **返回ARRAY类型。输入NULL时，则返回NULL。**

示例

例如表t_table_map的字段为c1 BIGINT,t_map MAP<STRING,BIGINT>，数据如下。

```
+------------+-------+
| c1         | t_map |
+------------+-------+
| 1000       | {k11:86, k21:15} |
| 1001       | {k12:97, k22:2} |
| 1002       | {k13:99, k23:1} |
+------------+-------+
```

执行SQL。

```
select  c1,map_keys(t_map) from t_table_map;
--结果如下。
```

```
+------------+------+
| c1         | _c1  |
+------------+------+
| 1000       | [k11, k21] |
| 1001       | [k12, k22] |
| 1002       | [k13, k23] |
+------------+------+
```

MAP_VALUES

函数声明格式如下。

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

用途：将参数MAP中的所有Values作为数组返回。

参数说明：

- **参数为MAP类型的数据。**

返回值说明：

- **返回ARRAY类型。输入NULL时，返回NULL。**

示例

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

ARRAY

函数声明格式如下。

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

用途：使用给定的Value构造ARRAY。

参数说明：

- **参数可为任意类型，但是所有参数的类型必须一致。**

返回值说明：

- **返回ARRAY类型。**

示例

例如表`t_table`的字段为`c1 BIGINT, c2 STRING, c3 STRING, c4 BIGINT, c5 BIGINT`，数据如下。

```
+------------+----+----+------------+------------+
| c1         | c2 | c3 | c4         | c5         |
+------------+----+----+------------+------------+
```

```
| 1000        | k11 | k21 | 86          | 15          |
| 1001        | k12 | k22 | 97          | 2           |
| 1002        | k13 | k23 | 99          | 1           |
+-------------+-----+-----+-------------+-------------+
```

执行SQL。

```
select array(c2,c4,c3,c5) from t_table;
--结果如下。
+------+
| _c0  |
+------+
| [k11, 86, k21, 15] |
| [k12, 97, k22, 2] |
| [k13, 99, k23, 1] |
+------+
```

说明:

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用`set`命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上`set`语句`set odps.sql.type.system.odps2=true;`，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

  对`setproject`的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

SIZE

函数声明格式如下。

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

用途：`size(map)`返回给定`map`中K/V对数。`size(array)`返回给定的`array`中的元素数目。

参数说明：

- `map`：MAP类型的数据。
- `array`：ARRAY类型的数据。

返回值说明：

- 返回INT类型。

示例

```
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。
```

ARRAY_CONTAINS

函数声明格式如下。

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

用途：检测给定ARRAY `a`中是否包含`v`。

参数说明：

- `a`：ARRAY类型的数据。
- `v`：必须与ARRAY数组中的数据类型一致。

返回值说明：

- 返回BOOLEAN类型。

示例

例如表`t_table_array`的字段为`c1 BIGINT, t_array ARRAY<STRING>`，数据如下。

```
+------------+---------+
| c1         | t_array |
+------------+---------+
| 1000       | [k11, 86, k21, 15] |
| 1001       | [k12, 97, k22, 2] |
| 1002       | [k13, 99, k23, 1] |
+------------+---------+
```

执行SQL。

```
select c1, array_contains(t_array,'1') from t_table_array;
--结果如下。
+------------+------+
| c1         | _c1  |
+------------+------+
| 1000       | false |
| 1001       | false |
| 1002       | true |
```

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

SORT_ARRAY

函数声明格式如下。

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

用途：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。

返回值说明：

- 返回ARRAY类型。

示例

例如表`t_array`的字段为`c1 ARRAY<STRING>,c2 ARRAY<INT> ,c3 ARRAY<STRING>`，数据如下。

```
+------------+---------+--------------+
| c1         | c2      | c3           |
+------------+---------+--------------+
| [a, c, f, b]  | [4, 5, 7, 2, 5, 8]  |  [你，我，他] |
+------------+---------+--------------+
```

执行SQL。

```
select sort_array(c1),sort_array(c2),sort_array(c3) from t_array;
--结果如下。
[a, b, c, f] [2, 4, 5, 5, 7, 8] [他，你，我]
```

说明：

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用`set`命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上`set`语句`set odps.sql.type.system.odps2=true;`，并与SQL语句一起提交执行。

- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

  对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

POSEXPLODE

函数声明格式如下：

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

用途：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：ARRAY类型的数据。数组中的数据可为任意类型。

返回值说明：

- 返回表生成的函数。

示例

```
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：

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。

- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

  对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

STRUCT

函数声明格式如下。

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

用途：使用给定value列表建立STRUCT。

参数说明：

- value：各value可为任意类型。

返回值说明：

- 返回STRUCT类型。Field的名称依次为col1，col2，…。

示例

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

说明：

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

  对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

NAMED_STRUCT

函数声明格式如下。

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

用途：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。

返回值说明：

- 返回STRUCT类型。Field的名称依次为col1，col2，…。

示例

```
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：

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用set命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上set语句set odps.sql.type.system.odps2=true;，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

  对setproject的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

INLINE

函数声明格式如下。

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

用途：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：数组中的`value`可为任意类型。

返回值说明：

- 返回表生成的函数。

示例

例如表`t_table`的字段为`t_struct struct<user_id:bigint,user_name:string,married:string,weight:double>`，表数据如下。

```
+----------+
| t_struct |
+----------+
| {user_id:10001, user_name:LiLei, married:N, weight:63.5} |
| {user_id:10002, user_name:HanMeiMei, married:Y, weight:43.5} |
+----------+
```

执行SQL。

```
select inline(array(t_struct)) from t_table;
--结果如下。
+------------+-----------+---------+------------+
| user_id    | user_name | married | weight     |
+------------+-----------+---------+------------+
| 10001      | LiLei     | N       | 63.5       |
| 10002      | HanMeiMei | Y       | 43.5       |
+------------+-----------+---------+------------+
```

说明：

目前MaxCompute SQL中使用到新数据类型（TINYINT、SMALLINT、INT、FLOAT、VARCHAR、TIMESTAMP、BINARY）时，需要用`set`命令开启：

- Session级别：要使用新数据类型，需在SQL语句前加上`set`语句`set odps.sql.type.system.odps2=true;`，并与SQL语句一起提交执行。
- Project级别：支持对Project级别进行新类型打开。Project Owner可根据需要对Project进行设置，命令为：

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

  对`setproject`的详细说明请参见其他操作。关于Project级别开启数据类型时的注意事项，请参见数据类型。

TRANS_ARRAY

函数声明格式如下。

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

用途：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，值必须>=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，例如Hangzhou;Beijing;shanghai，是以分号（;）分隔的数组。

返回值说明：

- 返回转置后的行，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的使用限制：

- 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

- 例如表t_table中的数据如下。

```
+----------+----------+------------+
| login_id | login_ip | login_time |
+----------+----------+------------+
| wangwangA | 192.168.0.1,192.168.0.2 | 20120101010000,2012010201
0000 |
| wangwangB | 192.168.45.10,192.168.67.22,192,168.6.3 | 2012011101
0000,20120112010000,20120223080000 |
```

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

执行SQL。

```
select trans_array(1, ",", login_id, login_ip, login_time) as (
login_id,login_ip,login_time) from t_table;
结果如下。
+----------+----------+------------+
| login_id | login_ip | login_time |
+----------+----------+------------+
| wangwangB | 192.168.45.10 | 20120111010000 |
| wangwangB | 192.168.67.22 | 20120112010000 |
| wangwangB | 192.168.6.3 | 20120223080000 |
| 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
```

会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
```

NVL

函数声明格式如下。

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

用途：如果value值为NULL，返回default_value，否则返回value。两个参数数据类型需要一致。

示例

例如表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 |
+-----+------------+-----+
```

4.16 UDF

4.16.1 UDF概述

UDF全称为User Defined Function，即用户自定义函数。MaxCompute提供多种内建函数来满足您的计算需求，同时您还可以通过创建自定义函数来满足不同的计算需求。

UDF的使用方法与普通的内建函数类似。Java和MaxCompute的数据类型的对应关系，请参见参数与返回值类型。

UDF分类

MaxCompute支持的UDF有三种，如下表所示。

UDF 分类	描述
UDF（User Defined Scalar Function）	用户自定义标量值函数（User Defined Scalar Function）。其输入与输出是一对一的关系，即读入一行数据，写出一条输出值。
UDTF（User Defined Table Valued Function）	自定义表值函数。用来解决一次函数调用输出多行数据场景。它是唯一能够返回多个字段的自定义函数，而UDF只能一次计算输出一条返回值。
UDAF（User Defined Aggregation Function）	自定义聚合函数。其输入与输出是多对一的关系， 即将多条输入记录聚合成一条输出值。它可以与SQL中的GROUP BY语句联用。具体语法请参见聚合函数。

说明:

- UDF广义的说法代表了自定义标量函数、自定义聚合函数及自定义表函数三种类型的自定义函数的集合。狭义的说法仅代表用户自定义标量函数。在帮助文档中将经常出现UDF这一名词，请您根据上下文内容理解其具体含义。
- 在SQL语句中使用自定义函数时，可能提示内存不足。这是由于计算的数据量过大并且存在倾斜，导致任务超出默认分配的内存。这种情况下，您可以通过`set odps.sql.udf.joiner.jvm.memory=xxxx;set odps.sql.udf.joiner.jvm.memory=xxxx;`语句手动设置更大的内存。

跨项目分享

MaxCompute的UDF支持跨项目分享，即在project_a中可以使用project_b的UDF。对应的语句为`select other_project:udf_in_other_project(arg0, arg1) as res from table_t;`。具体示例请参见跨项目空间Table、Resource、Function分享示例。

UDF SDK依赖安装

如果您使用Maven，可以从 Maven库中搜索odps-sdk-udf，从而获取不同版本的Java SDK，相关配置信息如下所示。

```
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-sdk-udf</artifactId>
   <version>0.29.10-public</version>
</dependency>
```

4.16.2 Java UDF

MaxCompute的UDF包括UDF、UDAF和UDTF三种函数。本文为您介绍如何通过Java实现这三种函数。

参数与返回值类型

MaxCompute 2.0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，以及Writable参数。

- Java UDF使用新基本类型的方法，如下所示：
  - UDTF通过`@Resolve`注解获取Signature，例如`@Resolve("smallint->varchar(10)")`。
  - UDF通过反射分析`evaluate`获取Signature，此时MaxCompute内置类型与Java类型符合一一映射关系。
  - UDAF通过`@Resolve`注解获取Signature，MaxCompute2.0支持在注解中使用新类型，例如`@Resolve("smallint->varchar(10)")`。

- Java UDF使用复杂类型的方法，如下所示：
  - UDTF通过@Resolve注解指定Signature，例如@Resolve("array<string>,struct<a1:bigint,b1:string>,string->map<string,bigint>,struct<b1:bigint>")。
  - UDF通过evaluate方法的Signature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通过@Resolve注解来获取Signature。MaxCompute 2.0版支持在注解中使用新类型，例如@Resolve("smallint->varchar(10)")。

    说明：
    - 您可以使用type,*实现任意个数的传参，例如@resolve("string,*->array<string>")，请注意此处ARRAY后需要加Subtype。
    - com.aliyun.odps.data.Struct从反射看不出Field Name和Field Type，所以需要用@Resolve注解来辅助。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注解，因此一个UDF中带有STRUCT参数或返回值的重载只能有一个。

MaxCompute数据类型与Java类型的对应关系，如下所示。

MaxCompute Type	Java Type
Tinyint	java.lang.Byte
Smallint	java.lang.Short
Int	java.lang.Integer
Bigint	java.lang.Long
Float	java.lang.Float
Double	java.lang.Double
Decimal	java.math.BigDecimal
Boolean	java.lang.Boolean
String	java.lang.String
Varchar	com.aliyun.odps.data.Varchar

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

说明:

- 在UDF中使用输入或输出参数的类型请务必使用Java Type，否则会报错ODPS-0130071。
- Java中对应的数据类型以及返回值数据类型是对象，首字母需大写。
- 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。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 。

说明:

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。例如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，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。

说明:

如果1个SQL中同时使用2个UDF，执行时2个UDF会共享一个Classpath，不会隔离。如果2个UDF引用的Resource资源都包含同一个Class，则Classloader会引用哪个Class是不确定的。因此，请尽量避免此种情况。

- 新版的MaxCompute支持定义Java UDF时，使用Writable类型作为参数和返回值。下面为MaxCompute类型和Java Writable类型的映射关系。

MaxCompute Type	Java Writable Type
tinyint	ByteWritable
smallint	ShortWritable
int	IntWritable
bigint	LongWritable
float	FloatWritable
double	DoubleWritable
decimal	BigDecimalWritable
boolean	BooleanWritable
string	Text
varchar	VarcharWritable
binary	BytesWritable
datetime	DatetimeWritable
timestamp	TimestampWritable
interval_year_month	IntervalYearMonthWritable
interval_day_time	IntervalDayTimeWritable
array	N/A
map	N/A

MaxCompute Type	Java Writable Type
struct	N/A

使用Writable类型实现Concat的示例如下：

```
package com.aliyun.odps.udf.example;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.udf.UDF;
public class MyConcat extends UDF {
  private Text ret = new Text();
  public Text evaluate(Text a, Text b) {
    if (a == null || b == null) {
      return null;
    }
    ret.clear();
    ret.append(a.getBytes(), 0, a.getLength());
    ret.append(b.getBytes(), 0, b.getLength());
    return ret;
  }
}
```

说明：

所有的Writable类型所在的Package为`com.aliyun.odps.io`。如果您要使用该类型，可到API文档地址下载`odps-sdk-commons`包。

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	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。

如果您想了解使用Intellij IDEA开发工具完成完整的Java UDF开发示例，请参见IntelliJ IDEA Java UDF开发最佳实践。使用Eclipse开发工具完成完整的Java UDF开发示例，请参见Eclipse Java UDF开发最佳实践。

其他UDF示例

以下示例定义了一个有三个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了STRUCT作为参数或者返回值，因此要求必须要对UDF Class打上`@Resolve`注解，指定STRUCT的具体类型。

```
import com.aliyun.odps.udf.UDF;
import com.aliyun.odps.udf.annotation.Resolve;
@Resolve("struct,string->string")
public class UdfArray extends UDF {
    public String evaluate(List vals, Long len) {
        return vals.get(len.intValue());
    }

    public String evaluate(Map 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。

```
create function my_index as 'UdfArray' using 'myjar.jar';
select id, my_index(array('red', 'yellow', 'green'), colorOrdinal) as
color_name from co
```

UDAF

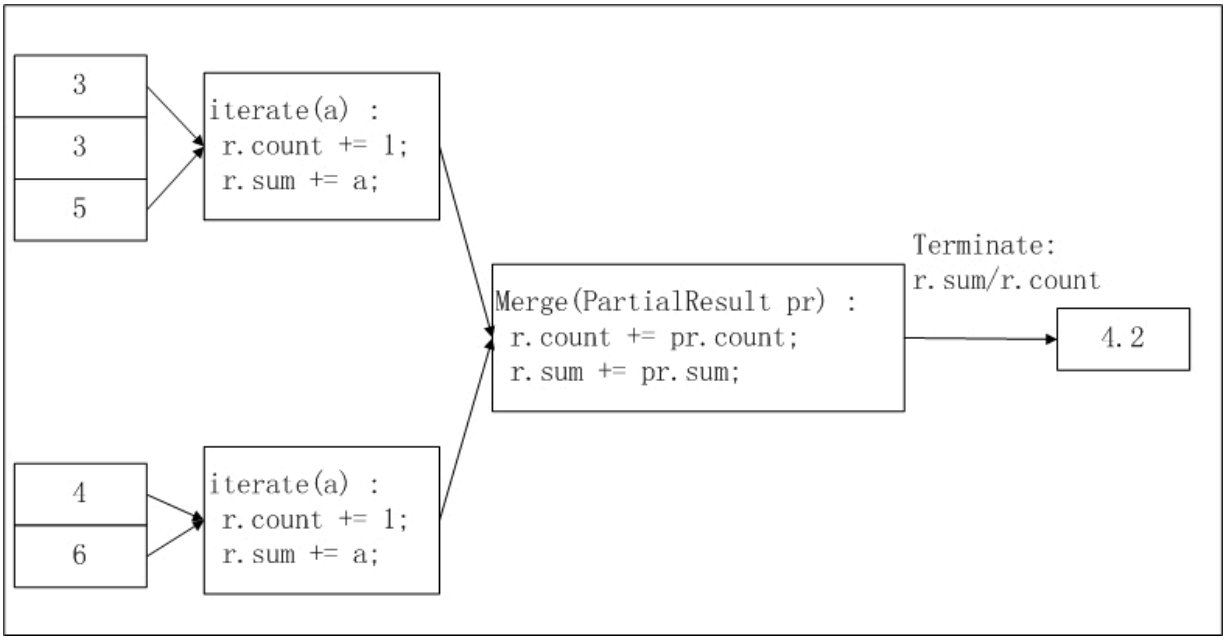
实现Java UDAF类需要继承`com.aliyun.odps.udf.Aggregator`，并实现以下几个接口。

```
import com.aliyun.odps.udf.ContextFunction;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.UDFException;
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;
  }
}
```

使用Writable类型实现Concat，示例如下。

```
package com.aliyun.odps.udf.example;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.udf.UDF;
public class MyConcat extends UDF {
  private Text ret = new Text();
  public Text evaluate(Text a, Text b) {
    if (a == null || b == null) {
```

```
      return null;
    }
    ret.clear();
    ret.append(a.getBytes(), 0, a.getLength());
    ret.append(b.getBytes(), 0, b.getLength());
    return ret;
  }
}
```

说明:

- `Writable[] writables`：表示一行数据。代码中是指传入的列，例如`writables[0]`表示第一列，`writables[1]`表示第二列。
- `iterate(Writable writable,Writable[] writables)`方法：`writable`是指一个阶段性的汇总数据，在不同的Map任务中，`group by`后得出的数据（可理解为一个集合），每行执行一次。
- `merge()`方法：将不同的Map直接结算的结果进行汇总。
- `terminate()`方法：返回数据。
- `newBuffer()`方法：创建初始返回结果的值。
- 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`，则需要清空。
- 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。
- STRING对应的Writable类型为Text。
- 所有的Writable类型所在的Package为`com.aliyun.odps.io`。如果您要使用该类型，可到API文档地址下载`odps-sdk-commons`包。

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。

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。您需要在`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被调用。
public void forward(Object …o) throws UDFException	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中可以这样使用此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`的结果，如下所示。

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

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. **运行该UDTF**。

```
select mp_udtf("10","20") as (a, b, fileResourceLineCount) from tmp1
;
```

返回结果如下。

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

以下示例定义了一个有三个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，因此要求必须对UDF Class添加`@Resolve`注解，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Hive风格的UDF，部分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。如果UDF是在其他版本的Hive/Hadoop开发的，则可能需要使用此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的使用请参考以下链接:

- HivePlugins
- DeveloperGuide UDTF
- 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需要注意以下几点：

- 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，还需注意MaxCompute的Java沙箱限制。

4.16.3 Python UDF

MaxCompute UDF包括UDF、UDAF和UDTF三种函数。本文为您介绍如何通过MaxCompute 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。具体标准库的可用模块说明如下：

- 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。例如，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外比较常用的第三方库，做为标准库的补充，例如Numpy。

> 说明:
> 第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，因此第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。
- 星号（*）代表变长参数。使用变长参数时，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，禁止执行此函数。执行时，UDF函数的参数会以函数签名指定的类型传给您。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数据类型对应的Python类型如下。

MaxCompute 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])`增加了参数`silent`。当`silent`为True时，如果`value`无法转为INT，则会返回None（不会抛出异常）。

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。

```
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。

Python UDF的使用示例请参考使用MaxCompute分析IP来源最佳实践。

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marshallableObject（例如LIST、DICT），并且`buffer`的大小不应该随数据量递增。在极限情况下，`buffer` Marshal过后的大小不应该超过2MB。
- `BaseUDAF.iterate(buffer[, args, ...])`：实现此方法将`args`聚合到中间值`buffer`中。

- `BaseUDAF.merge(buffer, pbuffer)`：实现此方法将两个中间值`buffer`聚合到一起，即将`pbuffer`合并到`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。您可以继承此类实现`process`、`close`等方法。
- `BaseUDTF.__init__()`：初始化方法。继承类如果需要实现这个方法，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，所有输出参数都将被视为STRING类型。因此在调用forward时，就必须将所有输出值转成STRING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)使用说明:**
  - **返回指定名字的资源内容。resource_name为STRING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Object后，调用者应该调用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)使用说明:**
  - **返回指定资源表的内容。resource_name为STRING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)
```

4.16.4 UDTF使用说明

本文为您介绍Java UDTF和Python UDTF的使用说明。

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;
```

UDTF的使用限制如下：

- 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
```

4.16.5 UDAF/UDTF动态参数说明

MaxCompute的UDAF和UDTF曾经使用Resolve注解决定函数的Signature，示例如下。这种方式的局限性在于输入参数和输出参数是固定的，无法实现方法重载。

```
@com.aliyun.odps.udf.annotation.Resolve("BIGINT->DOUBLE")
public class UDTFClass extends UDTF {
    ...
}
```

说明：

示例定义了一个UDTF，该UDTF接受的参数类型为BIGINT，返回值类型为DOUBLE。

现在MaxCompute对Resolve注解的语法做了如下扩展：

- 您可以在参数列表中使用*，表示接受任意长度、任意类型的输入参数。例如@Resolve('double,*->String')表示接受第一个是DOUBLE类型，后接任意长度、任意类型的参数列表。此时，您需要自己编写代码判断输入的个数和参数类型，然后对它们进行相应的操作（您可以对比C语言里面的printf函数来理解此操作）。

  说明:
  星号用在返回值列表中时，表示的是不同的含义，详见下文。

- 您可以在参数列表中使用any关键字，表示任意类型的参数。例如@Resolve('double,any->string')表示接受第一个是DOUBLE类型，后接任意类型的参数列表。

  说明:
  any不能用在返回值列表中，也不能用在复杂类型的子类型中（例如ARRAY）。

- UDTF的返回值可以使用星号，表示返回任意个STRING类型。返回值的个数与调用函数时设置的alias个数有关。例如@Resolve("ANY,ANY->DOUBLE,*")，调用方式是UDTF(x, y) as (a, b, c)，此处as后面设置了三个alias，即a、b、c。编辑器会认定a为DOUBLE类型（Annotation中返回值第一列的类型是给定的），b和c为STRING类型。因为这里给出了三个返回值，所以UDTF在调用forward时，必须forward长度为3的数组，否则会出现运行时报错。

  说明:
  这种错误无法在编译时报出，因此UDTF的调用者在SQL中设置alias个数时，必须遵循该UDTF定义的规则。由于Aggregator的返回值个数固定是1，所以这个功能对UDAF来说并无意义。

UDTF示例

```
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.UDTF;
import com.aliyun.odps.udf.annotation.Resolve;
import org.json.JSONException;
import org.json.JSONObject;
@Resolve("STRING,*->STRING,*")
public class JsonTuple extends UDTF {
  private Object[] result = null;
  @Override
  public void process(Object[] input) throws UDFException {
    if (result == null) {
      result = new Object[input.length];
    }
    try {
      JSONObject obj = new JSONObject((String)input[0]);
      for (int i = 1; i < input.length; i++) {
        // 返回值要求变长部分都是STRING。
        result[i] = String.valueOf(obj.get((String)(input[i])));
```

```
      }
      result[0] = null;
    } catch (JSONException ex) {
      for (int i = 1; i < result.length; i++) {
        result[i] = null;
      }
      result[0] = ex.getMessage();
    }
    forward(result);
  }
}
```

以上UDTF示例中，返回值个数会根据输入参数的个数来决定。输出参数中的第一个参数是一个JSON文本，后面的参数是需要从JSON中解析的Key。返回值中的第一个返回值是解析JSON过程中的出错信息，如果没有出错，则会根据输入的Key依次输出从JSON中解析出来的内容。使用示例如下。

```
-- 根据输入参数的个数定制输出alias个数。
SELECT my_json_tuple(json, 'a', 'b') as exceptions, a, b FROM jsons;

-- 变长部分可以一列都没有。
SELECT my_json_tuple(json) as exceptions, a, b FROM jsons;

-- 下面这个SQL会出现运行时错误，因为alias个数与实际输出个数不符。
-- 注意编译时无法发现这个错误。
SELECT my_json_tuple(json, 'a', 'b') as exceptions, a, b, c FROM jsons
;
```

当本文介绍的这些扩展无法满足您的业务需求时，建议您使用UDT实现Aggregator和UDTF的功能。详情请参见UDT文档中的聚合操作的实现示例和表值函数的实现示例。

4.17 UDF示例程序

4.17.1 JSON字符串获取示例

本文为您介绍如何通过Java UDF实现根据PATH路径获取JSON字符串。

JSON字符串获取UDF说明

`UDFGetJsonObject(String jsonString, String pathString)`:

- 函数功能

  解析JSON的字符串`json_string`，返回`path`指定的内容。如果输入的JSON字符串无效，则返回NULL。`$`代表根节点，`.`取子节点，`[]`取数组下标。`path`只支持字母、数字、下划线。

- 参数说明

  - `jsonString`：JSON字符串。
  - `pathString`：路径。

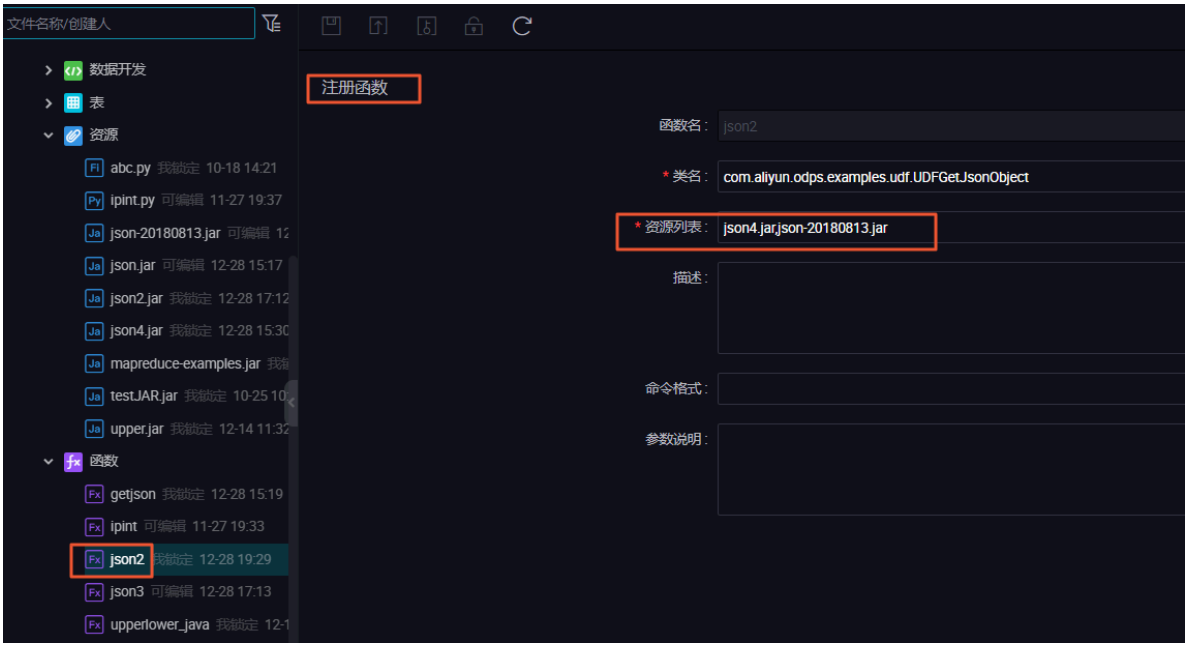
UDF使用示例

- 资源上传

  在MaxCompute公共云运行时，由于沙箱等限制，需要把*org.json*包和UDF Jar包分别作为资源上传至MaxCompute，并且在DataWorks创建函数时同时引用两个资源。您可以在MVN手动下载*org.json*包。

- 注册函数

  *UDFGetJsonObject.java*测试成功后，需要注册函数。本示例中，*json4.jar*是UDF打包后生成的Jar，而`json-20180813.jar`是`org.json`包。



- 使用示例

  成功注册UDF后，执行命令。

```
select UDFGetJsonObject('{"grade":"一年级","persons":[{"age":"a1","
name":"n1"},{"age":"a2","name":"n2"}]}', '$.persons[0].age') from
dual;
```

  执行结果为`a1`。

UDF代码示例

```
package com.aliyun.odps.examples.udf; //package名称，可以根据您的情况定义。
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.regex.Matcher;
```

```
import java.util.regex.Pattern;
//阿里云UDF。
import com.aliyun.odps.udf.UDF;

public class UDFGetJsonObject extends UDF {
  //声明私有不可变Pattern正则表达式模式。
  private final Pattern patternKey = Pattern.compile("^([a-zA-Z0-9_\
\-]+).*");
  private final Pattern patternIndex = Pattern.compile("\\[([0-9]+|\
\*)\\]");

  // 使用链接哈希映射的LRU表缓存。
  // 定义HashCache静态内部子类，继承LinkedHashMap。
  static class HashCache<K, V> extends LinkedHashMap<K, V> {

    private static final int CACHE_SIZE = 16;
    private static final int INIT_SIZE = 32;
    // 声明私有常量HashMap的LOAD_FACTOR装载因子。
    private static final float LOAD_FACTOR = 0.6f;

    HashCache() {
      super(INIT_SIZE, LOAD_FACTOR);
    }

    private static final long serialVersionUID = 1;

    // 重载LinkedHashMap方法，用于判断是否超过Cache大小并清理缓存。
    @Override
    protected boolean removeEldestEntry(Map.Entry<K, V> eldest) {
      return size() > CACHE_SIZE;
    }

  }
  //使用HashCache构造方法创建Map类型缓存区域。
  static Map<String, Object> extractObjectCache = new HashCache<String
, Object>();
  static Map<String, String[]> pathExprCache = new HashCache<String,
String[]>();
  static Map<String, ArrayList<String>> indexListCache = new HashCache
<String, ArrayList<String>>();
  static Map<String, String> mKeyGroup1Cache = new HashCache<String,
String>();
  static Map<String, Boolean> mKeyMatchesCache = new HashCache<String
, Boolean>();

  String result = new String();

  public UDFGetJsonObject() {
  }

  /**
   * Extract json object from a json string based on json path
specified, and
   * return json string of the extracted json object. It will return
null if the
   * input json string is invalid.
   *
   * A limited version of JSONPath supported: $ : Root object . :
Child operator
   * [] : Subscript operator for array * : Wildcard for []
   *
   * Syntax not supported that's worth noticing: '' : Zero length
string as key
```

```
   * .. : Recursive descent &amp;amp;#064; : Current object/element () :
Script
   * expression ?() : Filter (script) expression. [,] : Union operator
   * [start:end:step] : array slice operator
   *
   * @param jsonString
   *          the json string.
   * @param pathString
   *          the json path expression.
   * @return json string or null when an error happens.
   */
  public String evaluate(String jsonString, String pathString) {
    // 创建evaluate方法，与HIVE一样，MaxCompute的UDF通常使用evaluate方法,
jsonString作为输入的JSON字符串，pathString作为输入的路径。
    // 判断如果输入的JSON字符串路径为空则返回null。
    if (jsonString == null || jsonString.equals("") || pathString ==
null
        || pathString.equals("")) {
      return null;
    }

    try {
      // Cache pathExpr
      // 获取string数组类型的输入路径。
      String[] pathExpr = pathExprCache.get(pathString);
      if (pathExpr == null) {
        // 用“.”作为分隔，获取输入路径的string数组pathExpr，-1表示如果最后几位
是分隔符也会继续切割。
        pathExpr = pathString.split("\\.", -1);
        // 使用put方法将pathString、pathExpr缓存到pathExprCache。
        pathExprCache.put(pathString, pathExpr);
      }
      // 如果path首位不为$，则返回null。
      if (!pathExpr[0].equalsIgnoreCase("$")) {
        return null;
      }
      // Cache extractObject
      Object extractObject = extractObjectCache.get(jsonString);
      if (extractObject == null) {
        extractObject = new JSONObject(jsonString);
        // 使用put方法缓存JSON字符串内容。
        extractObjectCache.put(jsonString, extractObject);
      }
      for (int i = 1; i < pathExpr.length; i++) {
        // 根据路径参数pathExpr使用extract方法取出JSON字符串。
        extractObject = extract(extractObject, pathExpr[i]);
      }
      // 将获取的object类型的JSON字符串以string类型输出。
      return extractObject.toString();
    } catch (Exception e) {
      // 发生异常时返回null。
      return null;
    }
  }

  private Object extract(Object json, String path) throws JSONExcept
ion {
    // Cache patternkey.matcher(path).matches()
    Matcher mKey = null;
    Boolean mKeyMatches = mKeyMatchesCache.get(path);
    if (mKeyMatches == null) {
      // 如果缓存中不存在path。
      mKey = patternKey.matcher(path);
      // 如果路径可以匹配到JSON字符串，返回True，如果无法匹配，返回False。
```

```
      mKeyMatches = mKey.matches() ? Boolean.TRUE : Boolean.FALSE;
      mKeyMatchesCache.put(path, mKeyMatches);
    }
    if (!mKeyMatches.booleanValue()) {
      return null;
    }

    // Cache mkey.group(1)
    // 根据path获取对应的JSON。
    String mKeyGroup1 = mKeyGroup1Cache.get(path);
    if (mKeyGroup1 == null) {
      // 判断缓存中是否已存在JSON。
      if (mKey == null) {
        mKey = patternKey.matcher(path);
      }
      mKeyGroup1 = mKey.group(1);
      // 根据path缓存对应的JSON。
      mKeyGroup1Cache.put(path, mKeyGroup1);
    }
    // 获取JSON。
    json = extract_json_withkey(json, mKeyGroup1);

    // Cache indexList
    // 根据获取对应的数组类型JSON。
    ArrayList<String> indexList = indexListCache.get(path);
    if (indexList == null) {
      // 判断缓存中是否已存在数组类型JSON。
      Matcher mIndex = patternIndex.matcher(path);
      indexList = new ArrayList<String>();
      while (mIndex.find()) {
        indexList.add(mIndex.group(1));
      }
      indexListCache.put(path, indexList);
    }

    if (indexList.size() > 0) {
      // 根据path缓存对应数组类型的JSON。
      json = extract_json_withindex(json, indexList);
    }

    return json;
  }
  // 创建数组类型JSON对象。
  ArrayList<Object> jsonList = new ArrayList<Object>();
  // 创建获取数组类型JSON的方法。
  private Object extract_json_withindex(Object json, ArrayList<String
> indexList)
      throws JSONException {

    jsonList.clear();
    jsonList.add(json);
    Iterator<String> itr = indexList.iterator();
    while (itr.hasNext()) {
      String index = itr.next();
      ArrayList<Object> tmp_jsonList = new ArrayList<Object>();
      // 如果path路径中包含通配符，则获取所有通配符匹配的JSON。
      if (index.equalsIgnoreCase("*")) {
        for (int i = 0; i < (jsonList).size(); i++) {
          try {
            JSONArray array = (JSONArray) (jsonList).get(i);
            for (int j = 0; j < array.length(); j++) {
              tmp_jsonList.add(array.get(j));
            }
          } catch (Exception e) {
```

```
            continue;
          }
        }
        jsonList = tmp_jsonList;
      } else {
        // 如果不存在通配符，则遍历JSON。
        for (int i = 0; i < (jsonList).size(); i++) {
          try {
            tmp_jsonList.add(((JSONArray) (jsonList).get(i)).get(
Integer
                .parseInt(index)));
          } catch (ClassCastException e) {
            continue;
          } catch (JSONException e) {
            return null;
          }
          jsonList = tmp_jsonList;
        }
      }
    }
    return (jsonList.size() > 1) ? new JSONArray(jsonList) : jsonList.
get(0);
  }
  // 创建获取普通JSON的方法。
  private Object extract_json_withkey(Object json, String path)
      throws JSONException {
    if (json.getClass() == org.json.JSONArray.class) {
      JSONArray jsonArray = new JSONArray();
      for (int i = 0; i < ((JSONArray) json).length(); i++) {
        Object josn_elem = ((JSONArray) json).get(i);
        try {
          Object json_obj = ((JSONObject) josn_elem).get(path);
          if (json_obj.getClass() == org.json.JSONArray.class) {
            for (int j = 0; j < ((JSONArray) json_obj).length(); j++)
 {
              jsonArray.put(((JSONArray) json_obj).get(j));
            }
          } else {
            jsonArray.put(json_obj);
          }
        } catch (Exception e) {
          continue;
        }
      }
      return (jsonArray.length() == 0) ? null : jsonArray;
    } else {
      return ((JSONObject) json).get(path);
    }
  }
}
```

4.17.2 JSON字符串值判断示例

本文为您介绍如何通过Java UDF实现根据PATH路径获取JSON字符串并判断指定值是否存在。

JSON字符串值判断UDF说明

`UDFGetJsonObject(String jsonString, String pathString):`

- 函数功能

  根据路径path得到JSON串中的值，与JSON字符串获取示例中的get_json_object类似。它还可用于判断指定的PATH中是否包含Value。$代表根节点，.取子节点，[]取数组下标。path只支持字母、数字、下划线。

- 参数说明

  - jsonString：JSON字符串。
  - pathString：路径。

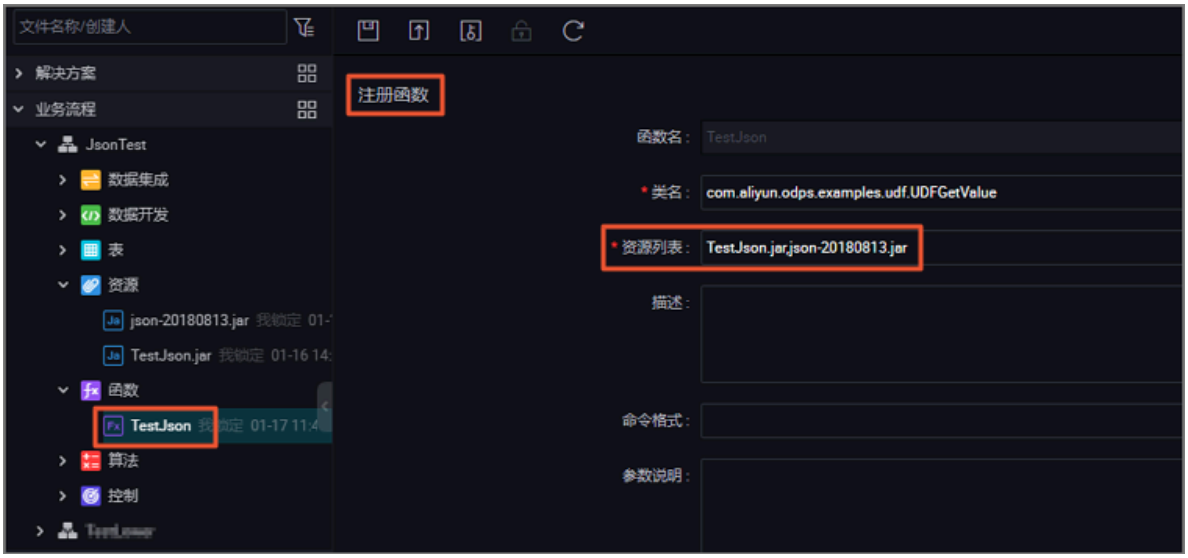
UDF使用示例

- 资源上传

  在MaxCompute公共云运行时，由于沙箱等限制，需要把*org.json*包和UDF Jar包分别作为资源上传至MaxCompute，并且在DataWorks创建函数时同时引用两个资源。您可以在MVN手动下载*org.json*包。

- 注册函数

  *TestJson.jar*是UDF打包后生成的Jar，而*json-20180813.jar*是org.json包。



- 使用示例

  成功注册UDF后，执行以下命令：

  1. ```
     select TestJson('{"LogA\":[{"id":"11562,20508"}, {"id":"11563,
     20509"}]}', "LogA[0]") from dual;
     ```

     执行结果为`{"id":"11562,20508"}`。

  2. ```
     select bi_udf:bi_get_value('{"LogA\":[{"id":"11562,20508"}, {"id
     ":"11563,20509"}]}', "LogA[0].id",  "20508") from dual;
     ```

     执行结果为`true`。

  3. ```
     select bi_udf:bi_get_value('{"LogA\":[{"id":"11562,20508"}, {"id
     ":"11563,20509"}]}', "LogA[0].id",  "20508",  "&") from dual;
     ```

     执行结果为`true`。

UDF代码示例

```
package com.aliyun.odps.examples.udf; //package名称，可以根据您的情况定义。
import com.aliyun.odps.io.Text;
import com.aliyun.odps.udf.UDF; //阿里云UDF。
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class UDFGetValue extends UDF {
    //声明私有不可变Pattern正则表达式模式。
    private final Pattern patternKey = Pattern.compile("^([a-zA-
Z0-9_\\-]+).*");
    private final Pattern patternIndex = Pattern.compile("\\[([0-
9]+|*)\\]");
    //使用HashCache构造方法创建Map类型缓存区域。
    private static Map<String, Object> extractObjectCache = new
  HashCache<String, Object>();
    private static Map<String, String[]> pathExprCache = new
HashCache<String, String[]>();
    private static Map<String, ArrayList<String>> indexListCache =
  new HashCache<String, ArrayList<String>>();
    private static Map<String, String> mKeyGroup1Cache = new
HashCache<String, String>();
    private static Map<String, Boolean> mKeyMatchesCache = new
HashCache<String, Boolean>();
    private Text result = new Text();
    private ArrayList<Object> jsonList = new ArrayList<Object>();

    // An LRU cache using a linked hash map, LRU 表缓存。
    //定义HashCache静态内部子类，继承LinkedHashMap。
    private static class HashCache<K, V> extends LinkedHashMap<K,
  V> {
```
```

```
        private  static  final  int  CACHE_SIZE  =  16;
        private  static  final  int  INIT_SIZE  =  32;
        //声明私有常量HashMap的LOAD_FACTOR装载因子。
        private  static  final  float  LOAD_FACTOR  =  0.6f;
        private  static  final  long  serialVersionUID  =  1;
        HashCache()  {
            super(INIT_SIZE,  LOAD_FACTOR);
        }
        //重载LinkedHashMap方法，用于判断是否超过Cache大小并清理缓存。
        @Override
        protected  boolean  removeEldestEntry(Map.Entry<K,  V>  eldest
)  {
            return  size()  >  CACHE_SIZE;
        }
    }

    public UDFGetValue()  {
    }

    /**
     *  @param  jsonStr  传入的Json字符串。
     *  @param  path  Json路径。
     *  @return  返回对应的值。
     */
    public  String  evaluate(String  jsonStr,  String  path)  {
        //创建evaluate方法，与HIVE一样，MaxCompute的UDF通常使用evaluate方
法，jsonString作为输入的JSON字符串，pathString作为输入的路径。
        //判断如果输入的JSON字符串路径为空则返回null。
        if  (jsonStr  ==  null  ||  jsonStr  ==  ""  ||  path  ==
null  ||  path  ==  "")  {
            return null;
        }
        String  jResult  =  evaluateJson(jsonStr,  path);
        if  (jResult  ==  null  ||  "".equals(jResult))
            return  null;
        result.set(jResult);
        return result.toString();
    }

    /**
     * @param jsonStr 传入的Json字符串。
     * @param path Json路径。
     * @param value。
     * @return 判断Json字符串中，指定的path中，是否包含有value(完全包含),返回"
true" 或者"false"。
     */
    public String evaluate(String jsonStr, String path, String value)
 {
        if (jsonStr == null || jsonStr == "" || path == null
                || path == "" || value == null || "".equals(value)) {
            return null;
        }
        String jResult = evaluateJson(jsonStr, path);
        if (jResult == null || "".equals(jResult)) {
            result.set("false");
            return result.toString();
        }
        String[] sV = value.split(",");
        String[] dV = jResult.split(",");
        int sameCount = 0;
        for (int i = 0; i < sV.length; i++) {
            for (int j = 0; j < dV.length; j++) {
                if (sV[i].equals(dV[j])) {
                    sameCount++;
```

```
                    break;
                }
            }
        }
        if (sameCount == sV.length) {
            result.set("true");
        } else {
            result.set("false");
        }
        return result.toString();
    }

    /**
     * @param jsonStr 传入的Json字符串。
     * @param path Json路径。
     * @param value。
     * @param flag 可以取值"|"或者"&","|"表示value中任意一个存在即可，"&"表示
value中的值都需要包含。
     * @return 判断Json字符串中，指定的path中，是否包含有value,返回"true" 或
者"false"。
     */
    public String evaluate(String jsonStr, String path, String value,
String flag) {
        if ("&".equals(flag))
            return evaluate(jsonStr, path, value);
        if (jsonStr == null || jsonStr.equals("") || path == null ||
path.equals("")
                || value == null || "".equals(value) || flag == null
                || !"|".equals(flag)) {
            result.set("false");
            return result.toString();
        }
        String jResult = evaluateJson(jsonStr, path);
        if (jResult == null || "".equals(jResult)) {
            result.set("false");
            return result.toString();
        }
        String[] sV = value.split(",");
        String[] dV = jResult.split(",");
        for (int i = 0; i < sV.length; i++) {
            for (int j = 0; j < dV.length; j++) {
                if (sV[i].equals(dV[j])) {
                    result.set("true");
                    return result.toString();
                }
            }
        }
        result.set("false");
        return result.toString();
    }

    /**
     * Extract json object from a json string based on json path
specified, and
     * return json string of the extracted json object. It will return
 null if
     * the input json string is invalid.
     *
     * A limited version of JSONPath supported: $ : Root object . :
Child
     * operator [] : Subscript operator for array * : Wildcard for []
     *
     * Syntax not supported that's worth noticing: '' : Zero length
string as
```

```
     * key .. : Recursive descent &amp;amp;#064; : Current object/element
 () :
     * Script expression ?() : Filter (script) expression. [,] : Union
 operator
     * [start:end:step] : array slice operator
     *
     * @param jsonString the json string.
     * @param pathString the json path expression.
     * @return json string or null when an error happens.
     */
    private String evaluateJson(String jsonString, String pathString)
 {
        try {
            pathString = "$." + pathString;
            // Cache pathExpr
            //获取string数组类型的输入路径。
            String[] pathExpr = pathExprCache.get(pathString);
            if (pathExpr == null) {
                //用“.”作为分隔,获取输入路径的string数组pathExpr, -1表示如果
最后几位是分隔符也会继续切割。
                pathExpr = pathString.split("\\.", -1);
                //使用put方法将pathString、pathExpr缓存到pathExprCache。
                pathExprCache.put(pathString, pathExpr);
            }
            //如果path首位不为$, 则返回null。
            if (!pathExpr[0].equalsIgnoreCase("$")) {
                return null;
            }
            // Cache extractObject
            Object extractObject = extractObjectCache.get(jsonString);
            if (extractObject == null) {
                extractObject = new JSONObject(jsonString);
                Object put = extractObjectCache.put(jsonString,
extractObject);
            }
            for (int i = 1; i < pathExpr.length; i++) {
                //根据路径参数pathExpr使用extract方法取出JSON字符串。
                extractObject = extract(extractObject, pathExpr[i]);
            }
            //将获取的object类型的JSON字符串以string类型输出。
            return extractObject.toString();
        } catch (Exception e) {
            //发生异常时返回null。
            return null;
        }
    }

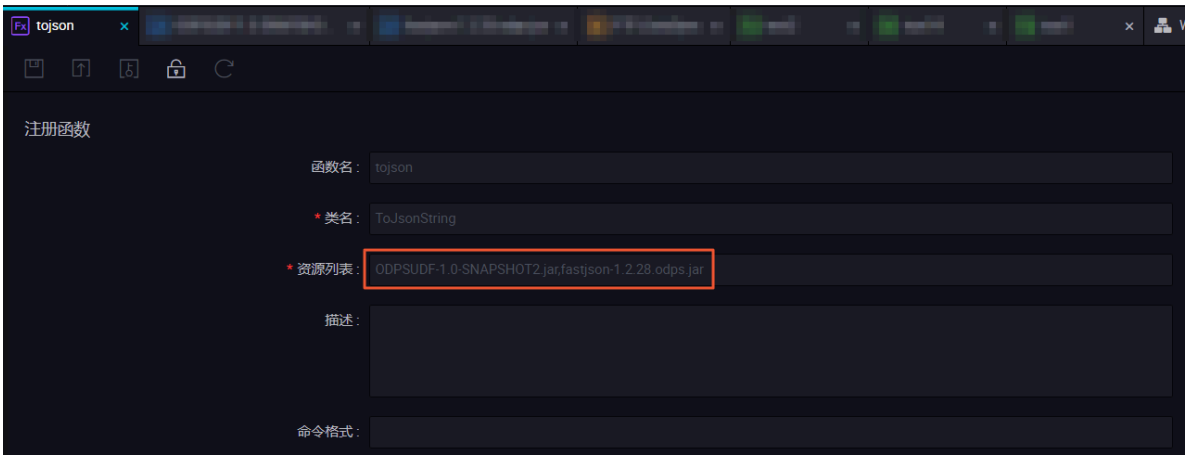
    private Object extract(Object json, String path) throws JSONExcept
ion {
        // Cache patternkey.matcher(path).matches()
        Matcher mKey = null;
        Boolean mKeyMatches = mKeyMatchesCache.get(path);
        //如果缓存中不存在path。
        if (mKeyMatches == null) {
            mKey = patternKey.matcher(path);
            //如果路径可以匹配到JSON字符串, 返回True, 如果无法匹配, 返回False。
            mKeyMatches = mKey.matches() ? Boolean.TRUE : Boolean.
FALSE;
            mKeyMatchesCache.put(path, mKeyMatches);
        }
        if (!mKeyMatches.booleanValue()) {
            return null;
        }
```

```
        // Cache mkey.group(1)
        //根据path获取对应的JSON。
        String mKeyGroup1 = mKeyGroup1Cache.get(path);
        //判断缓存中是否已存在JSON。
        if (mKeyGroup1 == null) {
            if (mKey == null) {
                mKey = patternKey.matcher(path);
            }
            mKeyGroup1 = mKey.group(1);
            // 根据path缓存对应的JSON。
            mKeyGroup1Cache.put(path, mKeyGroup1);
        }
        //获取JSON。
        json = extract_json_withkey(json, mKeyGroup1);

        // Cache indexList
        //根据获取对应的数组类型JSON。
        ArrayList<String> indexList = indexListCache.get(path);
        //判断缓存中是否已存在数组类型JSON。
        if (indexList == null) {
            Matcher mIndex = patternIndex.matcher(path);
            indexList = new ArrayList<String>();
            while (mIndex.find()) {
                indexList.add(mIndex.group(1));
            }
            indexListCache.put(path, indexList);
        }

        if (indexList.size() > 0) {
            // 根据path缓存对应数组类型的JSON。
            json = extract_json_withindex(json, indexList);
        }

        return json;
    }

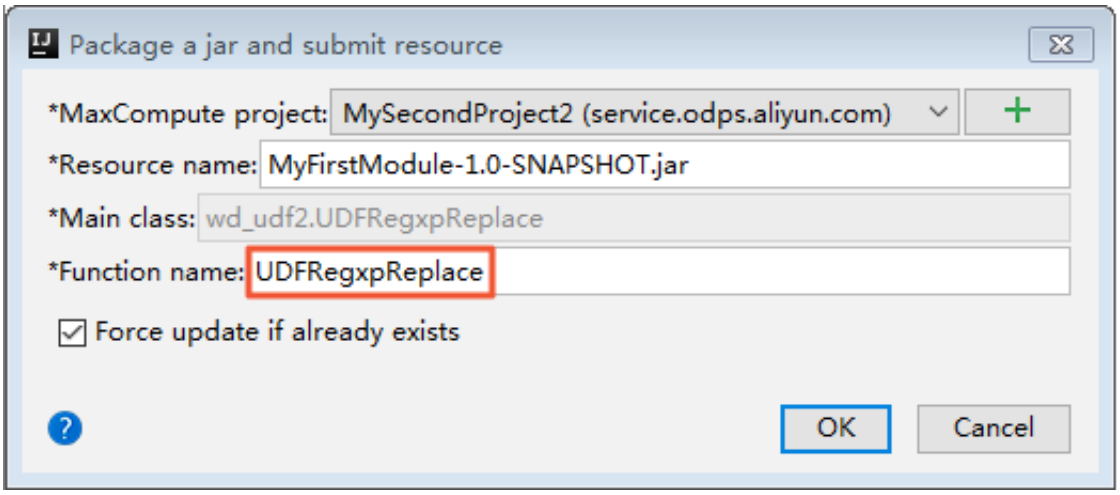
    private Object extract_json_withindex(Object json, ArrayList<
String> indexList) throws JSONException {

        jsonList.clear();
        jsonList.add(json);
        Iterator<String> itr = indexList.iterator();
        while (itr.hasNext()) {
            String index = itr.next();
            //创建数组类型JSON对象。
            ArrayList<Object> tmp_jsonList = new ArrayList<Object>();
            //创建获取数组类型JSON的方法。
            if (index.equalsIgnoreCase("*")) {
                for (int i = 0; i < (jsonList).size(); i++) {
                    try {
                        JSONArray array = (JSONArray) (jsonList).get(i
);
                        for (int j = 0; j < array.length(); j++) {
                            tmp_jsonList.add(array.get(j));
                        }
                    } catch (Exception e) {
                        continue;
                    }
                }
                jsonList = tmp_jsonList;
              //如果不存在通配符，则遍历JSON。
            } else {
                for (int i = 0; i < (jsonList).size(); i++) {
                    try {
```

```
                        tmp_jsonList.add(((JSONArray) (jsonList).get(i
))
                                .get(Integer.parseInt(index)));
                    } catch (ClassCastException e) {
                        continue;
                    } catch (JSONException e) {
                        return null;
                    }
                    jsonList = tmp_jsonList;
                }
            }
        }
        return (jsonList.size() > 1) ? new JSONArray(jsonList) :
jsonList
                .get(0);
    }
    //创建获取普通JSON的方法。
    private Object extract_json_withkey(Object json, String path)
throws JSONException {
        if (json.getClass() == JSONArray.class) {
            JSONArray jsonArray = new JSONArray();
            for (int i = 0; i < ((JSONArray) json).length(); i++) {
                Object josn_elem = ((JSONArray) json).get(i);
                try {
                    Object json_obj = ((JSONObject) josn_elem).get(
path);
                    if (json_obj.getClass() == JSONArray.class) {
                        for (int j = 0; j < ((JSONArray) json_obj).
length(); j++) {
                            jsonArray.put(((JSONArray) json_obj).get(j
));
                        }
                    } else {
                        jsonArray.put(json_obj);
                    }
                } catch (Exception e) {
                    continue;
                }
            }
            return (jsonArray.length() == 0) ? null : jsonArray;
        } else {
            return ((JSONObject) json).get(path);
        }
    }
}
```

4.17.3 不同类型数据转换JSON STRING示例

本文为您介绍如何通过Java UDF实现不同类型数据转换为JSON STRING类型。

在您的数据使用过程中，常常需要将不同类型的数据转换为JSON STRING。例如，您的数据是ARRAY类型，由于Tunnel上传下载目前不支持复杂数据类型，可能需要先将ARRAY类型数据转换为JSON STRING，再进行下载。

转换JSONStringUDF说明

- **函数名：**`ToJsonString`。

- 功能：将MaxCompute对象（字段）转换为JSON字符串，包括必要的字符串转义。您可以与`PutJsonValues`配合使用。
- 参数：任意类型的参数，但不支持DATETIME类型。

UDF使用示例

- 资源上传

  在MaxCompute公共云运行时，由于沙箱等限制，您需要把*fastjson.JSON*包和UDF Jar包分别作为资源上传至MaxCompute，并且在DataWorks创建函数时同时引用两个资源。您可以在MVN手动下载*fastjson.JSON*包。

- 注册函数

  *ToJsonStrin.java*运行成功后，将其注册函数。

  本例中*ODPSUDF-1.0-SNAPSHOT2.jar*是UDF打包后生成的Jar，而*fastjson-1.2.28.odps.jar*是*fastjson.JSON*包。



- 使用示例

```
SELECT tojson(array(2.0,3.0,4.0)) FROM dual;
```

运行结果如下。

```
[2.0,3.0,4.0]
```

> **说明:**
> `tojson`是您使用DataWorks或Intellij IDEA注册的函数名称。

UDF代码示例

```
import com.aliyun.odps.udf.UDF; //阿里云UDF。
import java.math.BigDecimal;
import com.alibaba.fastjson.JSON;  //引入fastjson.JSON类。
import java.util.Arrays;
```

```
import java.util.List;

public class ToJsonString extends UDF {
//下列代码调用fastjson.JSON的toJSONString方法，将各种类型的数据转换为JSON
string,您可以根据您的需要增加更多的数据类型。
    public String evaluate(String obj) {
        return JSON.toJSONString(obj);
    }
    public String evaluate(Long obj) {
        return JSON.toJSONString(obj);
    }
    public String evaluate(Double obj) {
        return JSON.toJSONString(obj);
    }
    public String evaluate(Boolean obj) {
        return JSON.toJSONString(obj);
    }
    public String evaluate(BigDecimal obj) {
        return JSON.toJSONString(obj);
    }
//MaxCompute中Array类型数据在JAVA UDF中类型为List，例如Array<double>类型可以
在UDF中通过List<Double>进行匹配。
    public String evaluate(List<Double> obj) {
        return JSON.toJSONString(obj);
    }
    public String evaluate(double[] obj) {
        return JSON.toJSONString(obj);
    }
    public String evaluate(int[] obj) {
        return JSON.toJSONString(obj);
    }
}
```

单元测试

```
//下列代码为测试代码，用于测试数据转换是否生效，您在实际使用过程中可注释掉。
    public static void main(String[] args) {
        //System.out.println(new ToJsonString().evaluate(null));//java
不能调用，但odps可以。
        System.out.println(new ToJsonString().evaluate("中文\t"));
        System.out.println(new ToJsonString().evaluate(123L));
        System.out.println(new ToJsonString().evaluate(1.123));
        System.out.println(new ToJsonString().evaluate(true));
        //System.out.println(new ToJsonString().evaluate(new Date
()));//udf/udaf不支持datetime类型。
        System.out.println(new ToJsonString().evaluate(BigDecimal.TEN
));
        System.out.println(new ToJsonString().evaluate(Arrays.asList(1
.0,2.0,3.0,4.0)));
        double[] args1 = {1,2,3,4};
        System.out.println(new ToJsonString().evaluate(args1));
       }
```

```
    }
```

4.17.4 JSON字符串增加键值示例

本文为您介绍如何通过Java UDF实现向基础JSON字符串中增加（覆盖）一个或多个键值对（支持嵌套JSON），接受奇数个参数。

JSON字符串增加键值UDF说明

- 函数名：`PutJsonValues`。
- 功能：向基础JSON串中增加（覆盖）一个或多个键值对支持嵌套JSON，接受奇数个参数。可以和不同类型数据转换JSON STRING示例中的`ToJsonObeject`配合使用（请先完成`ToJsonObeject`函数的注册）。
- 参数说明：
  - 参数1：基础JSON串。
  - 参数2：键1。
  - 参数3：值1。
  - 参数4：键2。
  - 参数5：值2。

UDF使用示例

- 资源上传

  实际在MaxCompute公共云运行时，由于沙箱等限制，需要把*fastjson.JSON*包和UDF Jar包分别作为资源上传至MaxCompute，并且在DataWorks创建函数时同时引用两个资源。您可以在MVN手动下载*fastjson.JSON*包。

- 注册函数

  *PutJsonValues.java*测试通过后，将其注册函数。

  本例中*PutJsonValues.jar*是UDF打包后生成的Jar，*ODPSUDF-1.0-SNAPSHOT2.jar*是上例`ToJsonString`打包后生成的Jar，而*fastjson-1.2.28.odps.jar*是*fastjson.JSON*包。

  说明：

  - 本例的操作步骤详情请参见资源。
  - 一个UDF从发布到服务端供生产使用，需要经过打包、上传、注册这三个步骤。您可以使用一键发布功能一次性完成这些步骤（Studio会依次执行`mvn clean package`、上传Jar和注册UDF这三个步骤）。详情请参见打包、上传和注册。
  - 通过客户端使用常用命令进行资源上传操作，请参见资源操作。

- 使用示例

  成功注册UDF后，执行以下命令。

```
SELECT tojson(NULL)
,  tojson(concat('123', chr(3)))
,  tojson(123)
,  tojson(12.3)
,  tojson(true)
,  tojson(CAST(123  AS  DECIMAL))
,  putJsonValuesTest('{}', 'a', '123', 'b', tojson('abc'), 'c', '{"
c1":567}')
FROM dual;
```

  运行结果：

```
null
"123\u0003"
123
```

```
12.3
true
123
{"a":123,"b":"abc","c":{"c1":567}}
```

UDF代码示例

```
import com.alibaba.fastjson.JSON; //阿里云UDF。
import com.alibaba.fastjson.JSONObject; //引入fastjson.JSON类。
import com.aliyun.odps.udf.UDF;

public class PutJsonValues extends UDF {
//MaxCompute中String类型数据与JAVA UDF中类型一致，String... keyValues为可变形参，表示参数个数不定。
    public  PutJsonValues(){
    }

    public  String evaluate(String jsonBase, String... keyValues) {
        if (jsonBase == null) {
            return null;
        }
        JSONObject outputObj = JSON.parseObject(jsonBase);
        for (int i = 1; i < keyValues.length; i = i + 2) {
            String key = keyValues[i - 1];
            String value = keyValues[i];
            if (key == null || value == null) {
                continue;
            }
            Object valueObj = JSON.parse(value);
            outputObj.put(key, valueObj);
        }
        return JSON.toJSONString(outputObj);
    }
}
```

说明:

- `null`代表"null"。`a`的值是数值没有引号。`b`的值是字符串，需要使用`tojson`转义为json元素（json字符串）。`{"c1":567}`本身已经是一个合法的json元素，因此可以直接传入。
- `PutJsonValues`、tojson是您使用DataWorks或Intellij IDEA注册的函数名称。

单元测试

```
//下列代码为测试代码，用于测试数据转换是否生效，您在实际使用过程中可注释掉。
public static void main(String[] args) {
    PutJsonValues putJsonValues=new PutJsonValues();
    System.out.println(putJsonValues.hashCode();

    String js=putJsonValues.evaluate("{}", "k", "null");
    System.out.println(js);
    System.out.println(new PutJsonValues().evaluate("{}", "a", "123",
 "b", "234", "c", "345"));
    System.out.println(new PutJsonValues().evaluate("{}", "k", "\"abc
\""));
    System.out.println(new PutJsonValues().evaluate("{}", "k", "{\"kk
\":123}"));
```

```
}
```

4.17.5 使用正则表达式替换字符串示例

本文为您介绍如何通过Java UDF实现使用正则表达式替换字符串。

替换字符串UDF说明

```
String UDFRegxpReplace(String s, String regex, String replacement):
```

- 函数名：`UDFRegxpReplace`。
- 功能：替换满足正则表达式的字符串。它与MaxCompute自带的内建函数`REGEXP_REPLACE`函数的不同点为：正则表达式可支持变量。
- 参数说明：
  - `s`：字符串。
  - `regex`：正则表达式。
  - `replacement`：要替换的字符串。
  - 参数类型：STRING类型。

UDF使用示例

- 注册函数

  *UDFRegxpReplace.java*测试通过后，将其注册函数使用。



> **说明：**
> 一个UDF从发布到服务端供生产使用，需要经过打包、上传、注册这三个步骤。您可以使用一键发布功能一次性完成这些步骤（Studio会依次执行`mvn clean package`、上传Jar和注册UDF这三个步骤）。详情请参见打包、上传和注册。

- 使用示例

  成功注册UDF后，执行以下命令。

  ```
  select UDFRegxpReplace("foobar", "oo|ar", "") from dual;
  ```

  执行结果如下。

  ```
  +-----+
  | _c0 |
  +-----+
  | fb  |
  +-----+
  ```

UDF代码示例

```
// package名称，可以根据您的情况定义。
package com.aliyun.odps.examples.udf;
import com.aliyun.odps.udf.annotation.UdfProperty;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

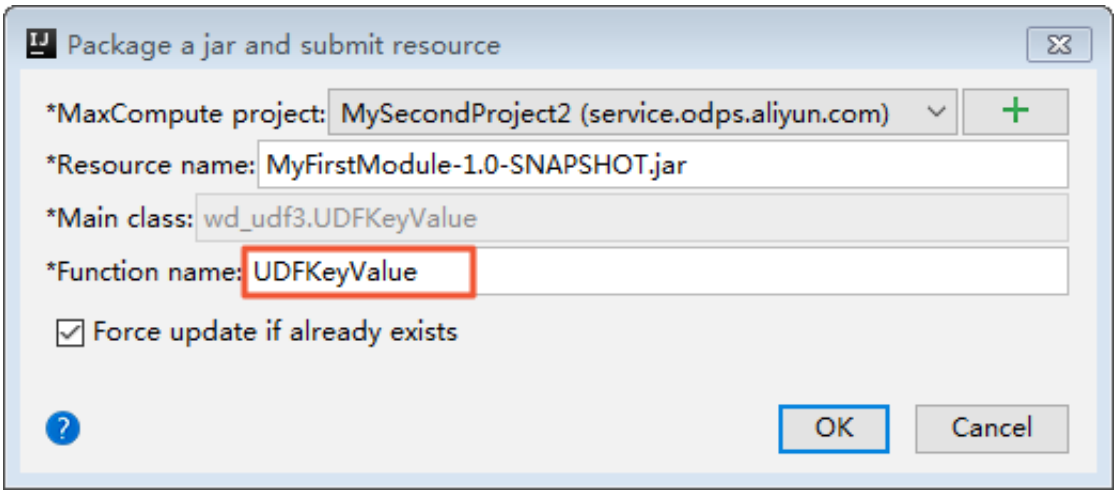
@UdfProperty(isDeterministic=true)
public class UDFRegxpRplace extends UDF {

    private String lastRegex = new String();
    private Pattern p = null;

    private String lastReplacement = new String();
    private String replacementString = "";

    public String evaluate(String s, String regex, String replacement)
 {
        if (s == null || regex == null || replacement == null) {
            return null;
        }
        // 如果正则表达式已更改，再次编译正则表达式。
        if (!regex.equals(lastRegex) || p == null) {
            lastRegex = regex;
            p = Pattern.compile(regex.toString());
        }
        Matcher m = p.matcher(s.toString());
        // 如果替换更改，再次执行toString。
        if (!replacement.equals(lastReplacement)) {
            lastReplacement = replacement;
            replacementString = replacement.toString();
        }

        StringBuffer sb = new StringBuffer();
        while (m.find()) {
            m.appendReplacement(sb, replacementString);
        }
        m.appendTail(sb);

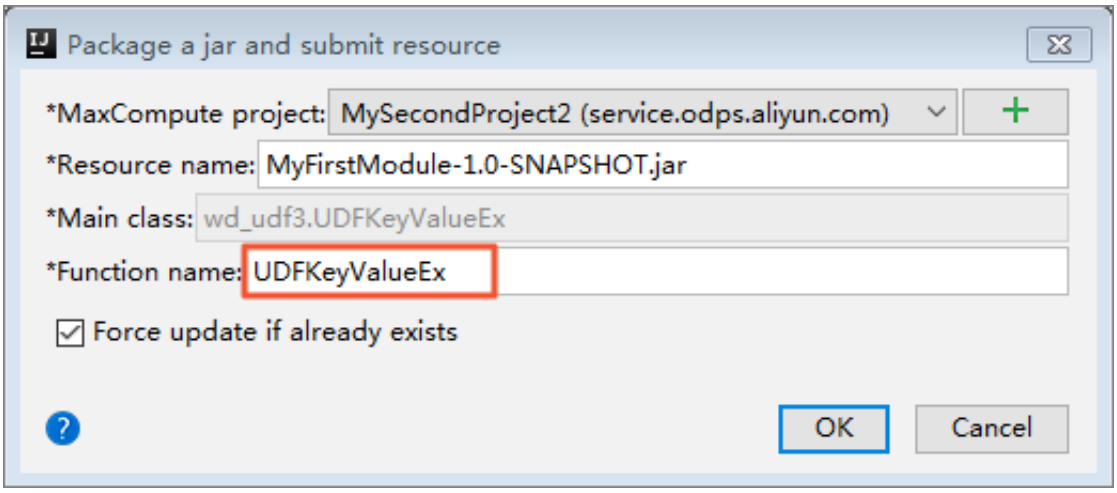
        return sb.toString();
    }
```

```
}
```

4.17.6 获取字符串（不含分隔符）value示例

本文为您介绍如何获取不包含分隔符的字符串中指定key对应的value值。

获取字符串（不含分隔符）value的UDF说明

- 函数名：UDFKeyValue
- 功能：从字符串中获得指定key的value值。

  说明：

  该UDF不适用于字符串本身包含分隔符的情况，如果需要处理该情况请使用UDFKeyValueEx。
- 参数说明

  UDFKeyValue(String str, String splitor1, String splitor2, String key):

  - str：字符串。
  - splitor1：通过splitor1分割出key:value对。
  - splitor2：对分割出来的key:value对，使用splitor2进行分割。
  - key：要获取的参数值。

  UDFKeyValue(String keyValues, String split):

  - str：字符串。
  - key：要获取的参数值。splitor1为;，splitor2为:。

UDF使用示例

1. 注册函数

*UDFKeyValue.java*测试通过后，将其注册函数。



> **说明:**
> 一个UDF从发布到服务端供生产使用，需要经过打包、上传、注册这三个步骤。您可以使用一键发布功能一次性完成这些步骤（Studio会依次执行`mvn clean package`、上传Jar和注册UDF这三个步骤）。详情请参见打包、上传和注册。

2. 使用示例

成功注册UDF后，执行以下命令：

- 示例一

```
select UDFKeyValue('a:1;b:2;','a') from dual;
```

运行结果如下。

```
+-----+
| _c0 |
+-----+
| 1   |
+-----+
```

- 示例二

```
select UDFKeyValue('a:1;b:2;','\;',':','a') from dual;
```

运行结果如下。

```
+-----+
| _c0 |
+-----+
| 1   |
```

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

UDF代码示例

```
//package名称，可以根据您的情况定义。
package com.aliyun.odps.examples.udf;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.udf.UDF;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

public class UDFKeyValue extends UDF{
        // 存储所有键：值对。
    private Map<String, String> mKeyValueache = new ConcurrentHashMap<
String, String>();
    private Text result = new Text();

    public UDFKeyValue() {
    }

    public String evaluate(String str, String keyname){
        // 默认split1=; split2=:。
        return evaluate(str, ";", ":", keyname);
    }

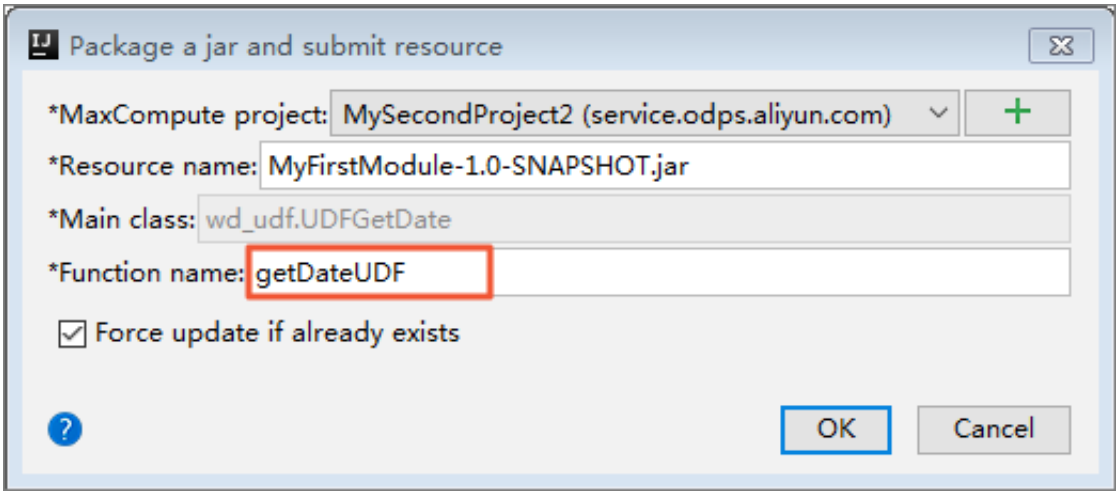
    public String evaluate(String str, String split1, String split2,
String keyname) {
        try {
                        // 通过splitor1分割出key:value对。
            if (str == null || "".equals(str))
                return null;
            String[] values1 = str.split(split1);
            mKeyValueache.clear();
            int i = 0;
                        // 对分割出来的key:value对，使用splitor2进行分割。
            while (i < values1.length) {
                storeKeyValue(values1[i], split2);
                i++;
            }
            String resultValue = getKeyValue(keyname);
            if (resultValue == null)
                return null;
            result.set(new Text(resultValue));
                        // 获取结果值。
            return result.toString();
        } catch (Exception e) {
            return null;
        }
    }

    private boolean storeKeyValue(String keyValues, String split) {
        if (keyValues == null || "".equals(keyValues))
            return false;
        if (mKeyValueache == null)
            mKeyValueache = new ConcurrentHashMap<String, String>();
        String[] keyValueArr = keyValues.split(split);
        if (keyValueArr.length == 2) {
            mKeyValueache.put(keyValueArr[0], keyValueArr[1]);
            return true;
        }
        return false;
    }
```

```
    private String getKeyValue(String keyName) {
        if (keyName == null ||
            "".equals(keyName) ||
            mKeyValueache == null ||
            mKeyValueache.size() == 0)
            return null;
        return mKeyValueache.get(keyName);
    }
}
```

4.17.7 获取字符串（含有分隔符）value示例

本文为您介绍如何通过Java UDF获取含有二级分隔符字符串中指定key对应的value值。

获取字符串（含有分隔符）value的UDF说明

- 函数名：`UDFKeyValueEx`。
- 功能：取出key所对应的value值。先按一级分隔符进行分割，然后按二级分隔符进行分隔，再取出key所对应的value值。与UDFKeyValue主要的不同在于，该UDF适用于value值本身含有二级分隔的情况。
- 参数说明

  `UDFKeyValueEx(String str, String split1, String split2, String keyname)`:

  - `str`：字符串。
  - `split1`：一级分隔符，为正则表达式。
  - `split2`：二级分隔符， 非正则表达式。
  - `keyname`：要获取的key值。

  `UDFKeyValueEx(String str, String keyname)`:

  - `str`：字符串。
  - `keyname`：要获取的key值。默认一级分隔符为`;`，二级分隔符为`:`。

UDF使用示例

1. 注册函数

*UDFKeyValueEx.java*测试通过后，将其注册函数。



说明:

- Studio提供了一键发布功能，即依次执行`mvn clean package`、上传Jar和注册UDF这三个步骤。您也可以在DataWorks选择资源管理，点击资源上传，选择Jar包完成上传，详情请参见资源。
- 通过客户端使用常用命令进行资源上传操作，请参见资源操作。

2. 使用示例

成功注册UDF后，执行以下命令：

- 示例一

```
select UDFKeyValueEx('a:b;c:d', ';', ':', 'a') from dual;
```

运行结果如下。

```
+------+
| _c0 |
+------+
| b    |
+------+
```

- 示例二

```
select UDFKeyValueEx('a:b:c;c:d', ';', ':', 'a:b') from dual;
```

运行结果如下。

```
+------+
| _c0 |
```

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

UDF代码示例

```
// package名称，可以根据您的情况定义。
package com.aliyun.odps.examples.udf;
import com.taobao.bi.odps.udf.UDF;

public class UDFKeyValueEx extends UDF {

    public String evaluate(String str, String keyname) {
                // 默认fault split1=; split2=:。
        return evaluate(str, ";", ":", keyname);
    }

    public String evaluate(String str, String split1, String split2,
String keyname) {
        try {
            if (str==null || split1==null || split2==null || keyname==
null){
                return null;
            }
                        // 使用二级分隔符。
            String keySplit = keyname+split2;
                        // 遍历字符串。
            for (String subStr : str.split(split1)){
                if (subStr.startsWith(keySplit)){
                    if (keySplit.length() < subStr.length()){
                                                //返回结果值。
                        return subStr.substring(keySplit.length());
                    }else{
                        return null;
                    }
                }
            }

            // 没有key对应的value。
            return null;
        } catch (Exception e) {
            return null;
        }
    }
}
```

4.17.8 获取指定日期格式的时间示例

本文为您介绍如何通过evaluate()方法获取指定格式时间。

获取指定时间UDF说明

- 函数名：UDFGetDate。
- 功能：获取指定格式时间。

- 参数说明：

  `String UDFGetDate(String date, Long days)`：

  - `date`：表示时间。格式为yyyyMMddHHmmss、yyyyMMdd、yyyy-MM-dd、yyyy-MM-dd HH:mm:ss。
  - `days`：间隔的天数。值可以为正或者负。
  - 返回日期格式：yyyyMMdd、yyyy-MM-dd。

  `String UDFGetDate(String date, Long months)`：

  - `date`：表示时间。格式为yyyyMMddHHmmss、yyyyMMdd、yyyy-MM-dd、yyyy-MM-dd HH:mm:ss。
  - `months`：表示间隔月份。如果值为2，则表示求指定月份向后加两个月的月份；如果值为-2，则表示求指定月份向前减两个月的月份。

UDF使用示例

- 注册函数

  *UDFGetDate.java*测试通过后，将其注册函数使用。



> **说明：**
> 一个UDF从发布到服务端供生产使用，需要经过打包、上传、注册这三个步骤。您可以使用一键发布功能一次性完成这些步骤（Studio会依次执行`mvn clean package`、上传Jar和注册UDF这三个步骤）。详情请参见打包、上传和注册。

- **使用示例**

  成功注册UDF后，执行以下命令：

  - **示例一**

    ```
    select getDateUDF(20140503,-2) from dual;
    ```

    运行结果如下。

    ```
    +-----+
    | _c0 |
    +-----+
    | 20140501 |
    +-----+
    ```

  - **示例二**

    ```
    select getDateUDF("20100405",2,10) from dual;
    ```

    运行结果如下。

    ```
    +-----+
    +-----+
    | _c0 |
    +-----+
    | 2010-06-10 |
    +-----+
    ```

  - **示例三**

    ```
    select getDateUDF("20100405",2,'first') from dual;
    ```

    运行结果如下。

    ```
    +-----+
    | _c0 |
    +-----+
    | 2010-06-01 |
    +-----+
    ---+
    ```

  - **示例四**

    ```
    select getDateUDF("20100405",-2,"10") from dual;
    ```

    运行结果如下。

    ```
    +-----+
    | _c0 |
    +-----+
    | 2010-02-10 |
    ```

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

UDF代码示例

```
package com.aliyun.odps.examples.udf; // package名称，可以根据您的情况定
义。
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

import com.aliyun.odps.io.Text;
import com.aliyun.odps.udf.UDF;
import com.aliyun.odps.udf.annotation.UdfProperty;
@UdfProperty(isDeterministic=true)
public class UDFGetDate extends UDF{
  public String evaluate(String riqi, Long days) {
        try {
                        // 判断日期。
            String year = riqi.substring(0, 2);
            if (!"18".equals(year) && !"19".equals(year) && !"20".
equals(year)
                    && !"21".equals(year)) {
                return "err";
            }

            String date = "";
            if (riqi.endsWith(".0")) {
                riqi = riqi.substring(0, riqi.length() - 2);
            }

            if (riqi.indexOf("-") >= 0) {
                riqi = riqi.substring(0, 10);
                riqi = riqi.replaceAll("-", "");
            } else {
                riqi = riqi.substring(0, 8);
            }
            try {
                                // 对指定日期进行处理：（+/-间隔天数）。
                date = getDay(riqi, days.intValue());
            } catch (Exception e) {
                return "err";
            }
            return date;
        } catch (Exception e) {
            return "err";
        }
    }

    public String evaluate(String riqi, Long n, String flag) {
        try {
                        // 判断日期。
            String year = riqi.substring(0, 2);
            if (!"18".equals(year) && !"19".equals(year) && !"20".
equals(year)
                    && !"21".equals(year)) {
                return "err";
            }
        } catch (Exception e) {
            return "err";
        }
```

```
        try {
            String date = "";
            if (riqi.endsWith(".0")) {
                riqi = riqi.substring(0, riqi.length() - 2);
            }

            if (riqi.indexOf("-") >= 0) {
                riqi = riqi.substring(0, 10);
                riqi = riqi.replaceAll("-", "");
            } else {
                riqi = riqi.substring(0, 8);
            }
            try {
                                    // 如果flag为first,则日期为获取月份和当月的
第一天。
                if ("first".equals(flag)) {
                    date = getMonthStartDate(riqi, n.intValue());
                } else if ("last".equals(flag)) {
                    date = getMonthEndDate(riqi, n.intValue());
                } else {
                                            // 如果flag为last,则日期为获取月份
和第last天。
                    date = getMonthDate(riqi, n.intValue(), flag);
                }
            } catch (Exception e) {
                return "err";
            }
            return date;
        } catch (Exception e) {
            return "err";
        }
    }

    public String evaluate(Long n) {
        if (null == n)
            return "err";
        return getNmonthAgo(n.intValue());
    }

    private String getMonthDate(String riqi, int n, String flag) {
        Calendar ca = Calendar.getInstance();
        Date dtBegin = new Date();
        try {
            dtBegin = new SimpleDateFormat("yyyyMMdd").parse(riqi);
        } catch (ParseException e1) {
            e1.printStackTrace();
        }
        int day = Integer.parseInt(flag);
        ca.setTime(dtBegin);
        ca.add(Calendar.MONTH, n);
        ca.set(Calendar.HOUR_OF_DAY, 0);
        ca.set(Calendar.MINUTE, 0);
        ca.set(Calendar.SECOND, 0);
        ca.set(Calendar.DAY_OF_MONTH, day);

        Date firstDate = ca.getTime();

        return ymdFormat(firstDate);
    }

    private String getNmonthAgo(int n) {
        Calendar ca = Calendar.getInstance();
        Date dtBegin = new Date();
        ca.setTime(dtBegin);
```

```
        ca.add(Calendar.MONTH, n);
        Date firstDate = ca.getTime();

        return ymdhmsFormat(firstDate);
    }
        // 对指定日期进行处理：（+/-间隔天数）。
    private String getDay(String strBeginDate, int n) {
        Date dtBegin = new Date();
        try {
            dtBegin = new SimpleDateFormat("yyyyMMdd").parse(
strBeginDate);
        } catch (ParseException e1) {
            e1.printStackTrace();
        }
        Calendar cld = Calendar.getInstance();
        cld.setTime(dtBegin);
        int day = cld.get(Calendar.DAY_OF_YEAR);

        String strYear = strBeginDate.substring(0, 4);
        String strInputPath = "";
        String strDate = "";
        try {
            int nDays = 1;

            cld.setTime(dtBegin);
            cld.set(Calendar.DAY_OF_YEAR, day + n);
            Date dt = cld.getTime();
            strDate = new SimpleDateFormat("yyyyMMdd").format(dt);
            strYear = strDate.substring(0, 4);

            strInputPath = strDate;

        } catch (NumberFormatException e) {
            e.printStackTrace();
        }

        return strInputPath;
    }
        // 对指定日期进行+flag处理。
    private String getMonthStartDate(String riqi, int n) {
        Calendar ca = Calendar.getInstance();
        Date dtBegin = new Date();
        try {
            dtBegin = new SimpleDateFormat("yyyyMMdd").parse(riqi);
        } catch (ParseException e1) {
            e1.printStackTrace();
        }
        ca.setTime(dtBegin);
        ca.add(Calendar.MONTH, n);
        ca.set(Calendar.HOUR_OF_DAY, 0);
        ca.set(Calendar.MINUTE, 0);
        ca.set(Calendar.SECOND, 0);
        ca.set(Calendar.DAY_OF_MONTH, 1);

        Date firstDate = ca.getTime();

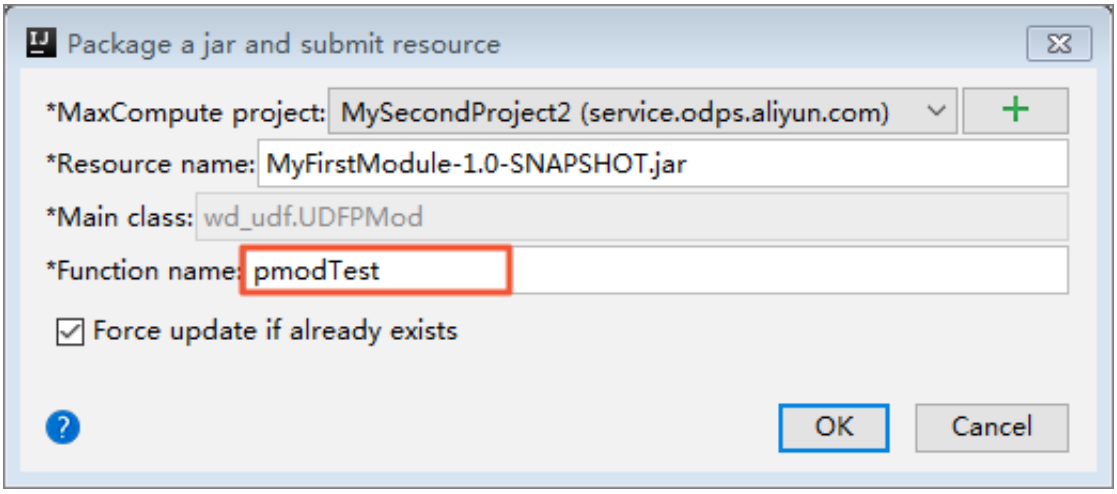
        return ymdFormat(firstDate);
    }

    private String getMonthEndDate(String riqi, int n) {
        Calendar ca = Calendar.getInstance();
        Date dtBegin = new Date();
        try {
            dtBegin = new SimpleDateFormat("yyyyMMdd").parse(riqi);
```

```
        } catch (ParseException e1) {
            e1.printStackTrace();
        }
        ca.setTime(dtBegin);
        ca.add(Calendar.MONTH, n);
        ca.set(Calendar.HOUR_OF_DAY, 23);
        ca.set(Calendar.MINUTE, 59);
        ca.set(Calendar.SECOND, 59);
        ca.set(Calendar.DAY_OF_MONTH, 1);
        ca.add(Calendar.MONTH, 1);
        ca.add(Calendar.DAY_OF_MONTH, -1);

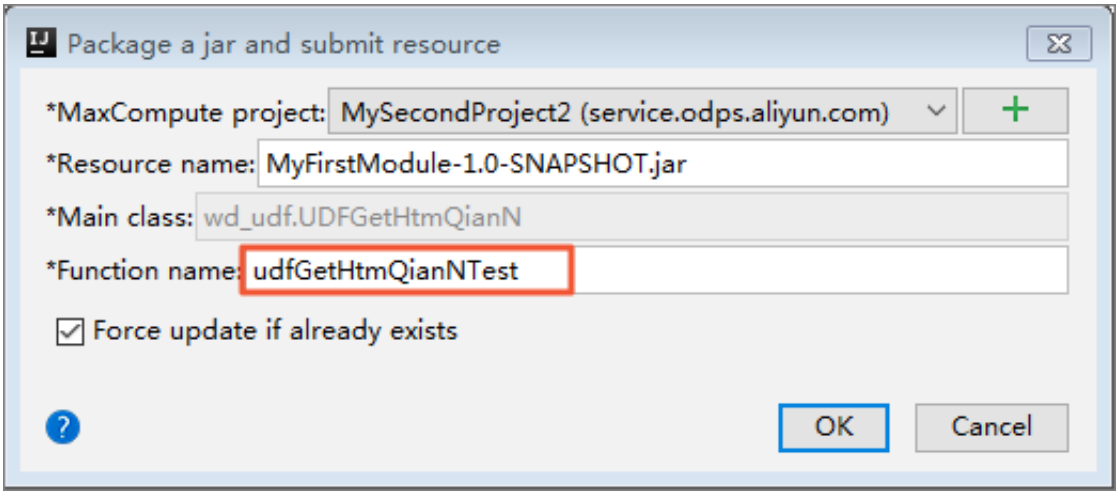
        Date lastDate = new Date(ca.getTime().getTime());

        return ymdFormat(lastDate);
    }

    private String ymdFormat(Date date) {
        if (date == null) {
            return "";
        }
                // 获取指定日期格式。
        DateFormat ymdFormat = new SimpleDateFormat("yyyy-MM-dd");
        return ymdFormat.format(date);
    }

    private String ymdhmsFormat(Date date) {
        if (date == null) {
            return "";
        }
                // 获取指定日期格式。
        DateFormat ymdFormat = new SimpleDateFormat("yyyyMMddHHmmss");
        return ymdFormat.format(date);
    }
}
```

4.17.9 取余函数示例

本文为您介绍如何通过Java UDF实现不同类型数值相除获取余数。

取余函数UDF说明

`Integer UDFPMod(Long a, Long b)`:

- 函数名：`UDFPMod`。
- 功能：返回a除以b的余数。
- 参数说明：
  - `a`：INTEGER类型数值1。
  - `b`：INTEGER类型数值2。

`Long UDFPMod(Long a, Long b)`:

- 函数名：`UDFPMod`。
- 功能：返回a除以b的余数。

- 参数说明：
  - a：LONG类型数值1。
  - b：LONG类型数值2。

`Double UDFPMod(Long a, Long b)`：

- 函数名：`UDFPMod`。
- 功能：返回a除以b的余数。
- 参数说明：
  - a：DOUBLE类型数值1。
  - b：DOUBLE类型数值2。

UDF使用示例

- 注册函数

  *UDFPMod.java*测试通过后，将其注册为函数使用。



> **说明：**
> 一个UDF从发布到服务端供生产使用，需要经过打包、上传、注册这三个步骤。您可以使用一键发布功能一次性完成这些步骤（Studio会依次执行`mvn clean package`、上传Jar和注册UDF这三个步骤）。详情请参见打包、上传和注册。

- 使用示例

  成功注册UDF后，执行以下命令：

  - 示例一

    ```
    select pmodTest(-9L,-4L) from dual;
    ```

    运行结果如下。

    ```
    +-----+
    | _c0 |
    +-----+
    | -1  |
    +-----+
    ```

  - 示例二

    ```
    select pmodTest(9.0,-4.0) from dual;
    ```

    运行结果如下。

    ```
    +-----+
    | _c0 |
    +-----+
    | -3.0 |
    +-----+
    ```

  - 示例三

    ```
    select pmodTest(9,4) from dual;
    ```

    运行结果如下。

    ```
    +-----+
    | _c0 |
    +-----+
    | 1   |
    +-----+
    ```

UDF代码示例

- UDF实现示例

```
package com.aliyun.odps.examples.udf// package名称，可以根据您的情况定义。
import com.aliyun.odps.udf.UDF;
public class UDFPMod extends UDF {
    // 使用Long类型的数据。
    public String evaluate(Long  a,  Long  b)  {
        if  ((a  == null) || (b == null) || b==0L)  {
            return  null;
        }
        Long d=((a % b) + b) % b;
        return d.toString();
    }
    // 使用Double类型的数据。
```

```
    public String evaluate(Double a, Double b)  {
        if  ((a == null) || (b == null) || b==0.0)  {
            return  null;
        }
        Double d=((a % b) + b) % b;
        return d.toString() ;
    }
    // 使用Integer类型的数据。
    public String evaluate(Integer a, Integer b)  {
        if  ((a == null) || (b == null) || b==0)  {
            return  null;
        }
        Integer d=((a % b) + b) % b;
        return d.toString();
    }
}
```

说明:

示例中为返回值类型做了toString处理，对测试结果无影响。

- **UDF单元测试**

```
package com.aliyun.odps.examples.udf// package名称，可以根据您的情况定义。
import org.junit.Test;
import static org.junit.Assert.assertEquals;
public class TestUDFPMod {
    private UDFPMod udf = new UDFPMod();
    @Test
    public void test_null(){
        assertEquals(null, udf.evaluate(null, 1L));
        assertEquals(null, udf.evaluate(1L, null));
        assertEquals(null, udf.evaluate((Long)null, null));

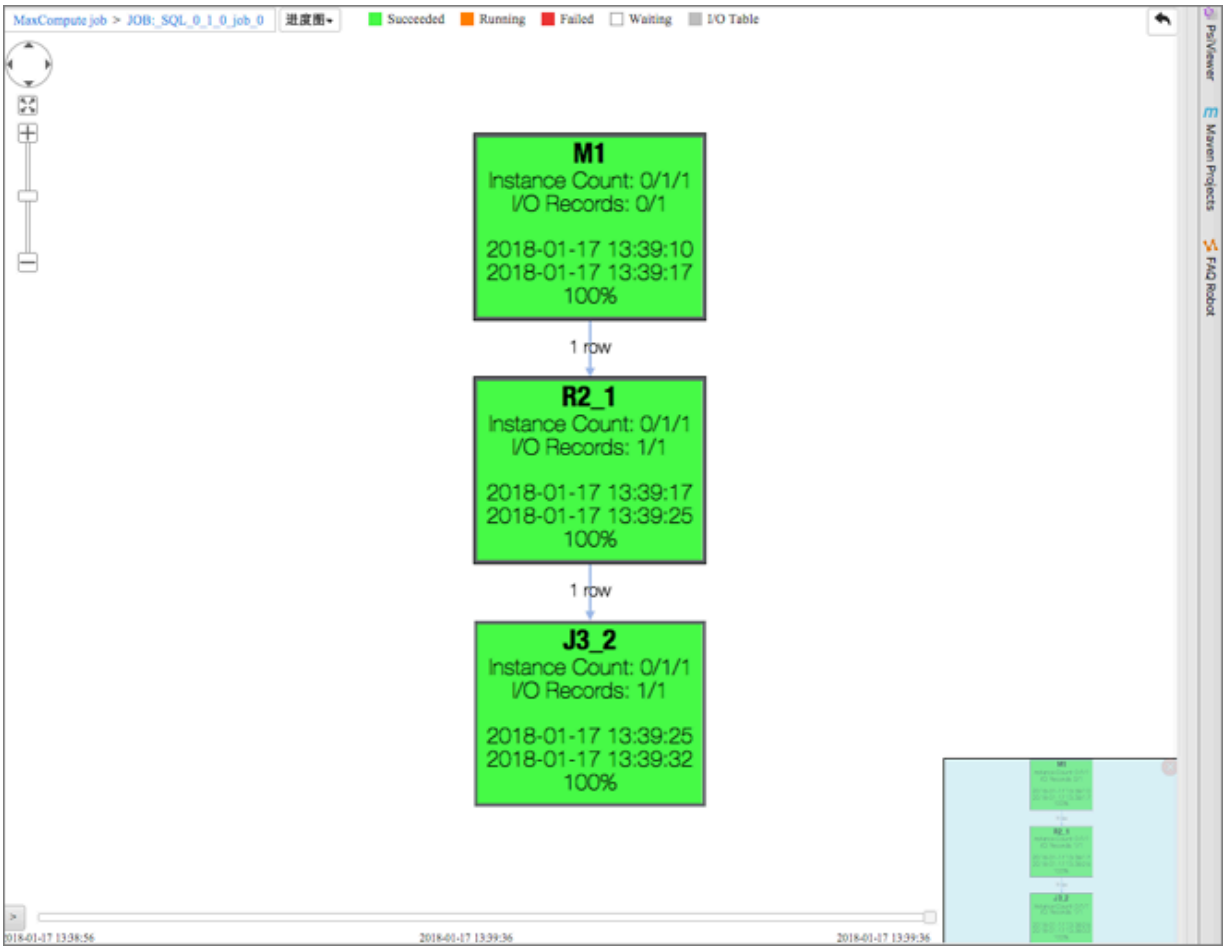
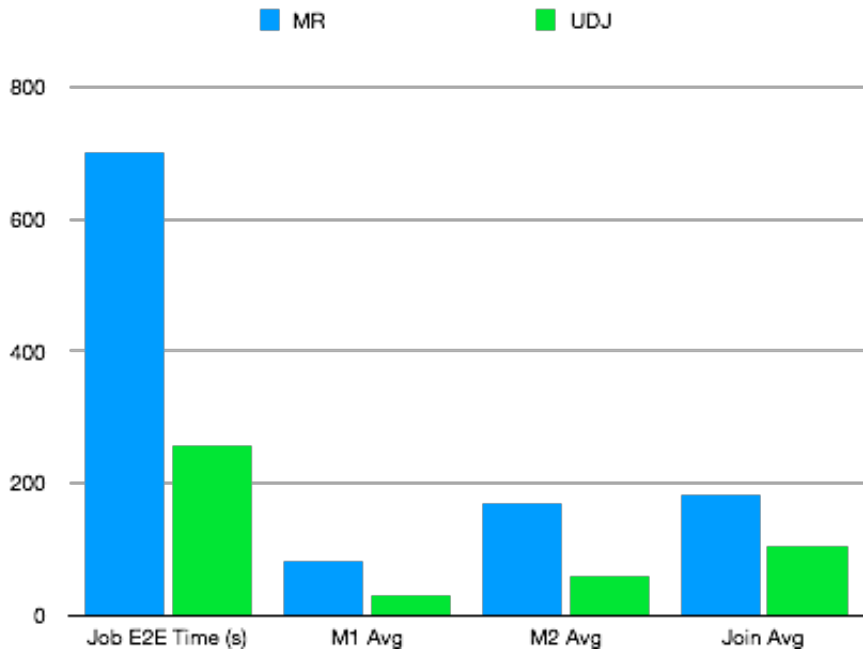
        assertEquals(null, udf.evaluate(null, 1.0));
        assertEquals(null, udf.evaluate(1.0, null));
        assertEquals(null, udf.evaluate((Double)null, null));
    }
    @Test
    public void test_正常情况(){
        assertEquals(1L, udf.evaluate(9L, 4L));
        assertEquals(3L, udf.evaluate(-9L, 4L));
        assertEquals(-1L, udf.evaluate(-9L, -4L));
        assertEquals(-3L, udf.evaluate(9L, -4L));
        // 原来抛异常，现返回null。
        assertEquals(null, udf.evaluate(9L, 0L));

        assertEquals(1.0, udf.evaluate(9.0, 4.0));
        assertEquals(3.0, udf.evaluate(-9.0, 4.0));
        assertEquals(-1.0, udf.evaluate(-9.0, -4.0));
        assertEquals(-3.0, udf.evaluate(9.0, -4.0));
        // 原来返回Nan，现返回null。
        assertEquals(null, udf.evaluate(9.0, 0.0));
    }
```

```
}
```

4.17.10 获取URL中指定位置的字符示例

本文为您介绍如何使用indexOf()、substring()、lastIndexOf()等字符串处理方法获取URL中指定位置的字符。

获取URL中指定位置的字符UDF说明

`String UDFGetHtmQianN(String url, Long n);`

- 函数名：`UDFGetHtmQianN`。
- 功能：获取URL中指定位置的字符。
- 参数说明：
  - `url`：STRING类型URL。
  - `n`：LONG类型长度值。

UDF使用示例

- 注册函数

  *UDFGetHtmQianN.java*测试通过后，将其注册为函数使用。



> **说明：**
> 一个UDF从发布到服务端供生产使用，需要经过打包、上传、注册这三个步骤。您可以使用一键发布功能一次性完成这些步骤（Studio会依次执行`mvn clean package`、上传Jar和注册UDF这三个步骤）。详情请参见打包、上传和注册。

- 使用示例

  成功注册UDF后，执行以下命令：

  - 示例一

    ```
    select udfGetHtmQianNTest("http://www.taobao.com", 1) from dual;
    ```

    运行结果如下。

    ```
    +-----+
    | _c0 |
    +-----+
    |     |
    +-----+
    ```

    说明：

    这里的返回结果不为NULL。

  - 示例二

    ```
    select udfGetHtmQianNTest("http://www.taobao.com/a.htm", 1) from
    dual;
    ```

    运行结果如下。

    ```
    +-----+
    | _c0 |
    +-----+
    | a   |
    +-----+
    ```

  - 示例三

    ```
    select udfGetHtmQianNTest("http://www.taobao.com/a-b-c-d.htm", 3)
    from dual;
    ```

    运行结果如下。

    ```
    +-----+
    | _c0 |
    +-----+
    | b   |
    +-----+
    ```

UDF代码示例

- UDF实现示例

  ```
  package com.aliyun.odps.examples.udf// package名称，可以根据您的情况定
  义。
  import com.aliyun.odps.udf.UDF;
  public  class  UDFGetHtmQianN  extends  UDF  {
      public  String  evaluate(String  url,  Long  n)  {
  ```

```
        try {
            // 是取url字符串中第一次出现.htm时.的索引位置，数值类型为int。
            int index = url.indexOf(".htm");
            if (index < 0) {
                return "";
            }
            // 从0开始数，其中不包括index位置的字符。
            String a = url.substring(0, index);
            // 返回指定字符/在此字符串中最后一次出现处的索引。
            index = a.lastIndexOf("/");
            // 取得的字符串长度为: a.length() - (index + 1)
            String b = a.substring(index + 1, a.length());
            // 用"-"拆分字符串b,获取字符串数组。
            String[] c = b.split("-");
            // 此处c.length>=n时,才有返回值。
            if (c.length < n) {
                return "";
            }
            if (n == 0) {
                return "";
            }
            // 返回字符串数组中指定下标的字符。
            String d = c[c.length - n.intValue()];
            return d;
        } catch (Exception e) {
            return "err";
        }
    }
}
```

· **UDF单元测试**

```
package com.aliyun.odps.examples.udf// package名称，可以根据您的情况定
义。
import org.junit.Test;
import static org.junit.Assert.assertEquals;
public class TestUDFGetHtmQianN {
    private UDFGetHtmQianN udf = new UDFGetHtmQianN();

    @Test
    public void test_url(){
        Long n = 1L;
        assertEquals("err", udf.evaluate(null, n));
        // 原来抛异常，现返回null。
        assertEquals("", udf.evaluate("", n));

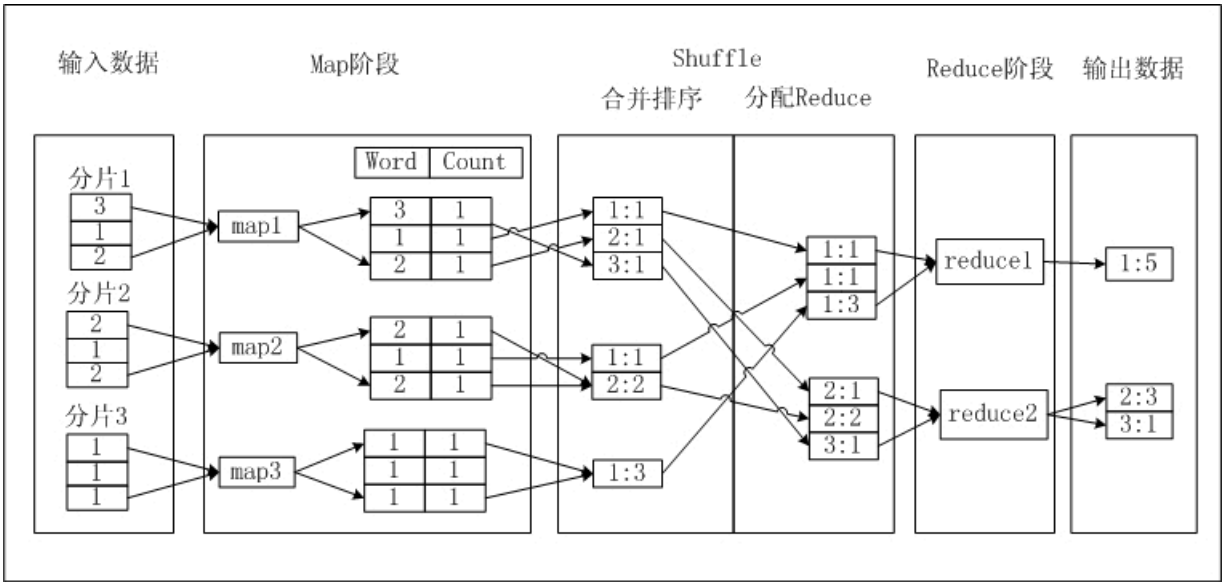
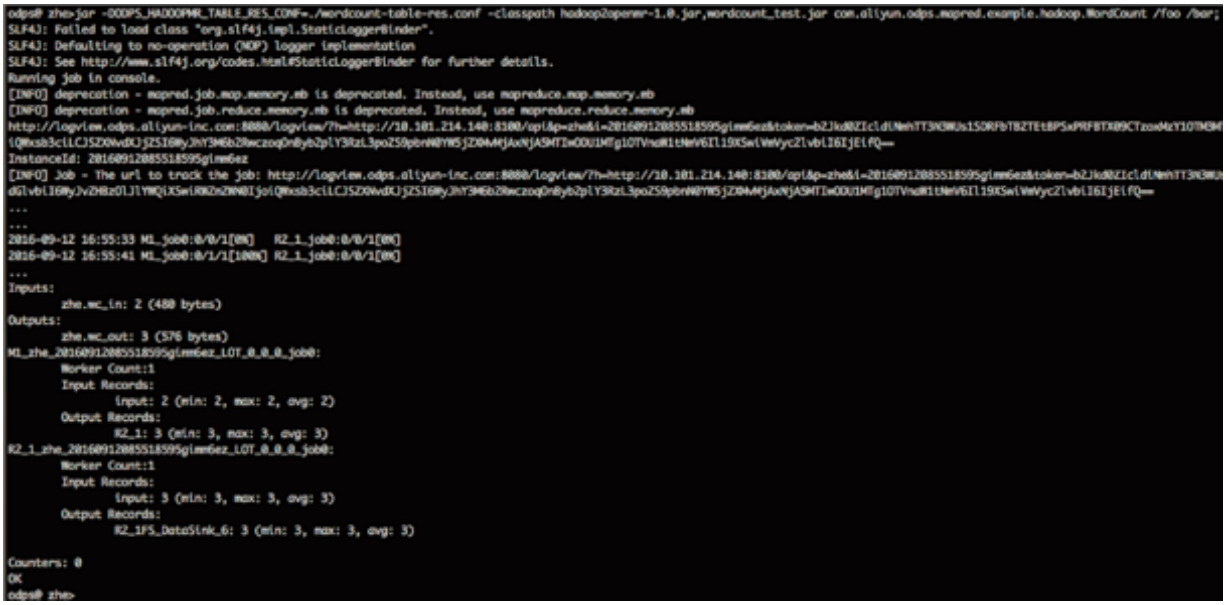
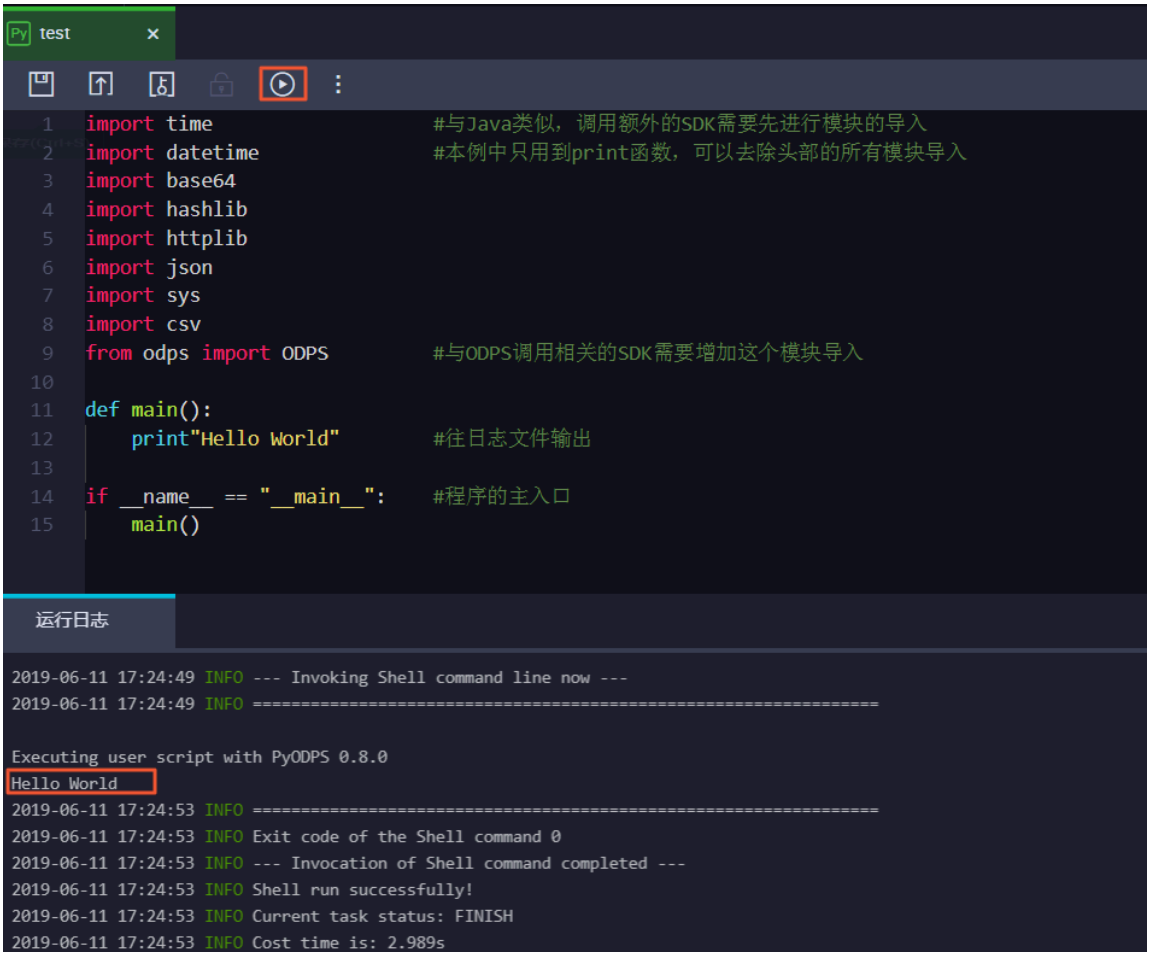
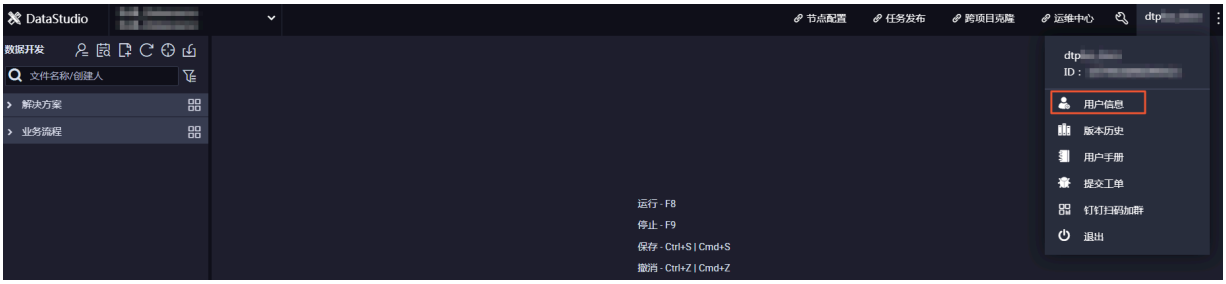
        assertEquals("", udf.evaluate("http://www.taobao.com", n
));
        assertEquals("a", udf.evaluate("http://www.taobao.com/a.htm
", n));
        assertEquals("b", udf.evaluate("http://www.taobao.com/a-b.
htm", n));
    }

    @Test
    public void test_index(){
        String url = "http://www.taobao.com/a-b-c.htm";
        assertEquals("err", udf.evaluate(url, null));
        // 原来抛异常，现返回null。
        assertEquals("err", udf.evaluate(url, -1L));
        assertEquals("", udf.evaluate(url, 0L));

        assertEquals("c", udf.evaluate(url, 1L));
```

```
        //  原来返回-，  现返回null。
        assertEquals("",  udf.evaluate(url,  100L));
    }
}
```

4.18 UDT

MaxCompute基于新一代的SQL引擎再推出新功能User Defined Type（UDT）。MaxCompute的UDT功能允许您在SQL中直接引用第三方语言的类或对象，获取其数据内容或调用其方法。

应用场景

UDT的常用场景如下：

- 场景1：某些可以用其它语言实现的功能，例如只需调用一次Java内置类的方法即可实现的功能，MaxCompute的内置函数却没有简单的方法实现那些功能。如果使用UDF实现，整个过程会过于繁杂。
- 场景2：SQL中需要调用第三方库实现相关功能。希望能够在SQL中直接调用，而不需要再Wrap一层UDF。
- 场景3：Select Transform支持把脚本写到SQL语句中，提升可读性和代码易维护性。但是某些语言无法这样使用，例如Java源代码必须经过编译才能执行，希望类似这些语言也可以直接写到SQL中。

UDT概述

其它的SQL引擎中UDT的概念不同于MaxCompute UDT的概念。很多SQL引擎中的概念比较像MaxCompute的STRUCT复杂类型，而某些语言提供了调用第三方库的功能，例如Oracle的Create Type。相比之下，MaxCompute的UDT更像Create Type的概念，Type中不仅包含数据域，还包含方法。而且MaxCompute不需要用特殊的DDL语法来定义类型的映射，而是在SQL中直接使用。

示例如下：

```
set odps.sql.type.system.odps2=true;
-- 打开新类型，因为下面的操作会用到INTEGER，即INT类型。
SELECT java.lang.Integer.MAX_VALUE;
```

上述语句的输出结果如下。

```
+------------+
| max_value |
+------------+
| 2147483647 |
```

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

和Java语言一样，*java.lang*包可以省略，所以上述示例可以简写为如下语句。

```
set odps.sql.type.system.odps2=true;
SELECT Integer.MAX_VALUE;
```

可以看到，上述示例在Select列表中直接写上了类似于Java表达式的表达式，而这个表达式按照Java的语义来执行，这个表达式表现出的能力就是MaxCompute的UDT。

UDT所提供的所有扩展能力可以用UDF实现。如果您想使用UDF实现上述示例，请进行以下操作。

1. 定义一个UDF的类。

```
package com.aliyun.odps.test;
public class IntegerMaxValue extends com.aliyun.odps.udf.UDF {
  public Integer evaluate() {
     return Integer.MAX_VALUE;
  }
}
```

2. 将上面的UDF编译，并打成Jar包。然后上传Jar包，并创建Function。

```
add jar odps-test.jar;
create function integer_max_value as 'com.aliyun.odps.test.
IntegerMaxValue' using 'odps-test.jar';
```

3. 在SQL中使用。

```
select integer_max_value();
```

UDT简化了上述一系列的过程，方便您使用其它语言扩展SQL的功能。

实现原理

概述中的示例体现了Java静态域访问的能力，UDT的能力远不限于此。以下示例为您介绍UDT的执行过程和具体功能。

```
-- 示例数据。
@table1 := select * from values ('10000000000000000000') as t(x);
@table2 := select * from values (100L) as t(y);
-- 代码逻辑。
@a := select new java.math.BigInteger(x) x from @table1;          --
new创建对象。
@b := select java.math.BigInteger.valueOf(y) y from @table2;
 -- 静态方法调用。
```

```
select /*+mapjoin(b)*/ x.add(y).toString() from @a a join @b b;
 -- 实例方法调用。
```

输出结果如下所示。

```
100000000000000000100
```

上述示例还体现了一种用UDF比较不好实现的功能：子查询的结果允许UDT类型的列。例如上面变量a的x列是`java.math.BigInteger`类型，而不是内置类型。UDT类型的数据可以被带到下一个Operator中再调用其它方法，甚至可以参与数据Shuffle。



如上图可知，该UDT共有三个Stage：M1、R2和J3。如果您熟悉MapReduce原理便会知道，由于`join`的存在需要做数据Reshuffle，所以会出现多个Stage。通常，不同的Stage是在不同的进程、不同的物理机器上运行的。

查看M1，可以看到M1只执行了`new java.math.BigInteger(x)`操作。

查看J3，可以看到J3在不同阶段执行了`java.math.BigInteger.valueOf(y)`和`x.add(y).toString()`操作。这几个操作不仅分阶段执行，而且在不同的进程、不同的物理机器上执行。UDT把这个过程封装起来，将这个过程变得看起来和在同一个JVM中执行的效果几乎一样。

详细功能说明：

- 目前UDT仅支持Java语言，后续版本会加入对其它语言的支持。
- UDT同样允许您上传自己的Jar包，并且直接引用。当前提供了一些Flag以方便您使用。
  - `set odps.sql.session.resources`指定引用的资源，可以指定多个，用逗号隔开，例如`set odps.sql.session.resources=foo.sh,bar.txt;`。

    说明:

    这个Flag和Select Transform中指定资源的Flag相同，所以这个Flag会同时影响两个功能。 例如UDT概述中UDF的Jar包，用UDT来使用。

    ```
    set odps.sql.type.system.odps2=true;
    set odps.sql.session.resources=odps-test.jar;
    --指定要引用的Jar，这些Jar一定要事先上传到Project，并且需要是Jar类型的资源。
    select new com.aliyun.odps.test.IntegerMaxValue().evaluate();
    ```
  - `odps.sql.session.java.imports`指定默认的Java Package，可以指定多个，用逗号隔开。和Java的`import`语句类似，可以提供完整类路径，例如`java.math.BigInteger`，也可以使用`*`。暂不支持`static import`。

    UDT概述中UDF的Jar包，用UDT使用还可以有如下写法。

    ```
    set odps.sql.type.system.odps2=true;
    set odps.sql.session.resources=odps-test.jar;
    set odps.sql.session.java.imports=com.aliyun.odps.test.*;
    -- 指定默认的Package。
    select new IntegerMaxValue().evaluate();
    ```
- UDT支持的操作如下：
  - 实例化对象的`new`操作。
  - 实例化数组的`new`操作，包括使用初始化列表创建数组，例如`new Integer[] { 1, 2, 3 }`。
  - 方法调用，包括静态方法调用（因此能用工厂方法构建对象）。
  - 域访问，包括静态域。

    说明:

    - 仅支持公有方法和公有域的访问。
    - UDT中的标识符是大小写敏感的，包括Package、类名、方法名和域名。
    - UDT支持类型转换，支持SQL的风格，例如`cast(1 as java.lang.Object)`。但不支持Java风格的类型转换，例如`(Object)1`。
    - 暂不支持匿名类和Lambda表达式（后续版本可能会支持）。

- 暂不支持无返回值的函数调用（因为UDT均出现在Expression中，没有返回值的函数调用无法嵌入到Expression中，这个问题在后续的版本中会有解决方案）。

- Java SDK的类都是默认可用的。

  说明:
  目前Runtime使用的JDK版本是JDK1.8，可能不支持更新版本的JDK。

- 所有的运算符都是MaxCompute SQL的语义，不是UDT的语义。例如`String.valueOf(1) + String.valueOf(2)`的结果是3（STRING隐式转换为DOUBLE，并且DOUBLE相加），而不是12（Java中STRING相加是Concatenate的语义）。

  除了STRING的相加操作比较容易混淆外，另一个比较容易混淆的是=操作。SQL中的=不是赋值而是判断相等。而对于Java对象来说，判断相等应该用Equals方法，而非=操作（在UDT场景下，同一对象的概念是不能保证的，详情请参见下文的说明）。

- 内置类型与特定Java类型有一一映射关系，详情请参见Java UDF中的数据类型映射表，这个映射关系在UDT也有效。

  - 内置类型的数据能够直接调用其映射到的Java类型的方法，例如`'123'.length()`，`1L.hashCode()`。
  - UDT类型能够直接参与内置函数或者UDF的运算，例如`chr(Long.valueOf('100'))`，其中`Long.valueOf`返回的是`java.lang.Long`类型的数据，而内置函数`Chr`接受的数据类型是内置类型BIGINT。
  - Java的PRIMITIVE类型可以自动转化为其BOXING类型，并应用前两条规则。

  说明:
  部分内置的新数据类型需要先设置`set odps.sql.type.system.odps2=true;`才可使用，否则会报错。

- UDT对泛型有比较完整的支持，例如`java.util.Arrays.asList(new java.math.BigInteger('1'))`，编译器可以根据参数类型判断出该方法的返回值是`java.util.List<java.math.BigInteger>`类型。

  说明:
  构造函数需要指定类型参数，否则使用`java.lang.Object`，这一点和Java保持一致。`new java.util.ArrayList(java.util.Arrays.asList('1', '2'))`的结果是`java.util.ArrayList<Object>`类型，而`new java.util.ArrayList<String>(java.util.Arrays.asList('1', '2'))`的结果是`java.util.ArrayList<String>`类型。

- UDT对同一对象的概念是模糊的。这是由数据的Reshuffle导致的。从第一部分join的示例可以看出，对象有可能会在不同进程、不同物理机器之间传输，在传输过程中同一个对象的两个引用后面可能分别引用了不同的对象（例如对象先被Shuffle到两台机器，然后下次又Shuffle回一起）。

  所以在使用UDT时，应该使用equals方法判断相等，避免使用=判断相等。

  某行某列的对象，其内部包含的各个数据对象的相关性是可以保证的。不同行或者不同列的对象的数据相关性是不保证的。
- 目前UDT不能用作Shuffle Key，包括join、group by、distribute by、sort by、order by、cluster by等结构的Key。

  并不是说UDT不能用在这些结构中，UDT可以在Expression中间的任意阶段使用，但不能作为最终输出。例如，不可以使用语句group by new java.math.BigInteger('123')，但可以使用语句group by new java.math.BigInteger('123').hashCode()。因为hashCode方法的返回值是int.class类型，可以当做内置类型INT来使用（应用上述内置类型与特定Java类型规则）。
- UDT扩展了类型转换规则：
  - UDT对象可以被隐式类型转换为其基类对象。
  - UDT对象可以被强制类型转换为其基类或子类对象。
  - 没有继承关系的两个对象之间遵守原来的类型转换规则，注意这时可能会导致内容的变化。例如java.lang.Long类型的数据是可以强制转换为java.lang.Integer的，应用的是内置类型的BIGINT强制转换为INT的过程，而这个过程会导致数据内容的变化，甚至可能导致精度的损失。
- 目前UDT对象不能落盘，这意味着不能将UDT对象insert到表中（实际上DDL不支持UDT，创建不出这样的表），当然，隐式类型转换变成了内置类型的除外。同时，屏显的最终结果也不能是UDT类型，对于屏显的场景，由于所有的Java类都有toString()方法，而java.lang.String类型是合法的。所以Debug时，可以用这种方法观察UDT的内容。

  您也可以设置set odps.sql.udt.display.tostring=true;，这样MaxCompute会自动帮您把所有以UDT为最终输出的列Wrap上java.util.Objects.toString(...)，以方便调试。这个Flag只对屏显语句生效，对insert语句不生效，所以它只用在调试的过程中。

  内置类型支持BINARY，因此支持自己实现序列化的过程，将byte[]的数据落盘。下次读出时再还原回来。

  某些类可能自带序列化和反序列化的方法，例如Protobuffer。目前UDT依旧不支持落盘，需要您自己调用序列化反序列化方法，变成BINARY数据类型来落盘。

- UDT不仅可以实现Scalar函数的功能，配合内置函数collect list和explode，UDT还可以实现Aggregator和Table Function功能。
- UDT支持资源（Resource）的访问，您可以在SQL中通过`com.aliyun.odps.udf.impl.UDTExecutionContext.get()`静态方法获取`ExecutionContext`对象，从而访问当前的`ExecutionContext`，进而访问资源（例如文件资源和表格资源）。

使用Java数组示例

```
set odps.sql.type.system.odps2=true;
set odps.sql.udt.display.tostring=true;
select
    new Integer[10],     -- 创建一个10个元素的数组。
    new Integer[] {c1, c2, c3},  -- 通过初始化列表创建一个长度为3的数组。
    new Integer[][] { new Integer[] {c1, c2}, new Integer[] {c3, c4
} },  -- 创建多维数组。
    new Integer[] {c1, c2, c3} [2], -- 通过下标操作访问数组元素。
    java.util.Arrays.asList(c1, c2, c3);     -- 这个创建了一个List<
Integer>，这个也能当做array<int>来用，所以这是另一个创建内置ARRAY数据的方法。
from values (1,2,3,4) as t(c1, c2, c3, c4);
```

使用JSON示例

UDT的Runtime自带一个JSON的依赖（2.2.4），因此可以直接使用JSON。

```
set odps.sql.type.system.odps2=true;
set odps.sql.session.java.imports=java.util.*,java,com.google.gson.*;
 -- 同时import多个Package时用逗号隔开。
@a := select new Gson() gson;    -- 构建gson对象。
select
gson.toJson(new ArrayList<Integer>(Arrays.asList(1, 2, 3))), -- 将任意
对象转成JSON字符串。
cast(gson.fromJson('["a","b","c"]', List.class) as List<String>) -- 反
序列化JSON字符串，注意gson的接口，直接反序列化出来是List<Object>类型，所以这里强
转成了List<String>，方便后续使用。
from @a;
```

相比于内置函数get_json_object，上述用法不仅使用方便，在需要对JSON字符串多个部分做内容提取时，先将JSON字符串反序列成格式化数据，提升工作效率。

除JSON外，MaxCompute Runtime自带的依赖还包括：`commons-logging (1.1.1)`、`commons-lang (2.5)`、`commons-io (2.4)`、`protobuf-java (2.4.1)`。

复杂类型操作示例

内置类型ARRAY和MAP与`java.util.List`和`java.util.Map`存在映射关系。结果如下：

- Java中实现了`java.util.List`或者`java.util.Map`接口类的对象，都可参与MaxCompute SQL的复杂类型操作。
- MaxCompute中ARRAY、MAP的数据，能够直接调用List或者MAP的接口。

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

```
set odps.sql.session.java.imports=java.util.*;
select
    size(new ArrayList<Integer>()),        -- 对ArrayList数据调用内置函数
Size。
    array(1,2,3).size(),                   -- 对内置类型ARRAY调用List的方
法。
    sort_array(new ArrayList<Integer>()),  -- 对ArrayList的数据进行排序。
    al[1],                                 -- 虽然Java的List不支持下标操
作，但ARRAY支持。
    Objects.toString(a),        -- 之前不支持将ARRAY类型cast成STRING，现在
有绕过方法了。
    array(1,2,3).subList(1, 2)             -- 求出subList。
from (select new ArrayList<Integer>(array(1,2,3)) as al, array(1,2,3)
as a) t;
```

聚合操作的实现示例

UDT实现聚合的原理是，先用内置函数collect_set或collect_list函数将数据转变成List，之后对该List应用UDT的标量方法求得这一组数据的聚合值。

下述示例实现对BigInteger求中位数（由于数据是java.math.BigInteger类型的，所以不能直接用内置的median函数）。

```
set odps.sql.session.java.imports=java.math.*;
@test_data := select * from values (1),(2),(3),(5) as t(value);
@a := select collect_list(new BigInteger(value)) values from @
test_data;  -- 先把数据聚合成List。
@b := select sort_array(values) as values, values.size() cnt from @a;
 -- 求中位数的逻辑，先将数据排序。
@c := select if(cnt % 2 == 1, new BigDecimal(values[cnt div 2]), new
 BigDecimal(values[cnt div 2 - 1].add(values[cnt div 2])).divide(new
BigDecimal(2))) med from @b;
-- 最终结果。
select med.toString() from @c;
```

由于collect_list会先把所有数据都收集到一块，是没有办法实现Partial Aggregate的，所以这个做法的效率会比内置的Aggregator或者UDAF低，所以在内置Aggregator能实现的情况下，应尽量使用内置的Aggregator。同时把一个Group的所有数据都收集到一起，会增加数据倾斜的风险。

但是另一方面，如果UDAF本身的逻辑就是要将所有数据收集到一块（例如类似内置函数wm_concat的功能），此时使用上述方法，反而可能比UDAF（注意不是内置Aggregator）高。

表值函数的实现示例

表值函数允许输入多行多列数据，输出多行多列数据。可以按照下述原理实现：

1. 对于输入多行多列数据，可以参考聚合函数实现的示例。
2. 要实现多行的输出，可以让UDT方法输出一个Collection类型的数据（List或者MAP），然后调用explode函数，将Collections展开成多行。

3. UDT本身就可以包含多个数据域，通过调用不同的getter方法获取各个域的内容即可展开成多列。

下述示例实现将一个JSON字符串的内容展开的功能。

```
@a := select '[{"a":"1","b":"2"},{"a":"1","b":"2"}]' str; -- 示例数据。
@b := select new com.google.gson.Gson().fromJson(str, java.util.List.
class) l from @a; -- 反序列化JSON。
@c := select cast(e as java.util.Map<Object,Object>) m from @b lateral
 view explode(l) t as e;  -- 用explode打成多行。
@d := select m.get('a') as a, m.get('b') as b from @c; -- 展开成多列。
select a.toString() a, b.toString() b from @d; -- 最终结果输出（注意变量d
的输出中a，b两列是Object类型）。
```

函数重载实现示例

MaxCompute的UDF使用重载evaluate方法的方式来重载函数。这种方式不支持泛型，所以当您需要定义一个可以接受任何数据类型的函数时，必须为每种类型都写一个evaluate函数。但是，这种方法依然无法实现个别输入类型（例如ARRAY）的重载函数。在没有提供Resolve注解（Annotation）的情况下，Python UDF或UDTF会根据参数个数决定输入参数，同时支持变长参数，但这种过于灵活的机制也会导致编译器无法静态找到某些错误。

通过UDT实现函数重载，可以很好地解决以上问题。 UDT支持泛型、类继承、变长参数，为您提供灵活的函数定义方式，示例如下。

```
public class UDTClass {
    // 这个函数接受一个数值类型（可以是TINYINT、SMALLINT、INT、BIGINT、FLOAT
、DOUBLE以及任何以Number为基类的UDT），返回DOUBLE。
    public static Double doubleValue(Number input) {
        return input.doubleValue();
    }
    // 这个方法，接受一个数值类型参数和一个任意类型的参数，返回值类型与第二个参数的
类型相同。
    public static <T extends Number, R> R nullOrValue(T a, R b) {
        return a.doubleValue() > 0 ? b : null;
    }
    // 这个方法接受一个任意元素类型的ARRAY或List，返回BIGINT。
    public static Long length(java.util.List<? extends Object> input)
 {
        return input.size();
    }
    // 注意这个在不做强制转换的情况下参数只能接受UDT的java.util.Map<Object,
Object>对象。如果需要传入任何MAP对象，例如map<bigint,bigint>可以考虑：
    // 1. 定义函数时使用java.util.Map<? extends Object, ? extends Object
>。
    // 2. 调用时强转，例如UDTClass.mapSize(cast(mapObj as java.util.Map<
Object, Object>))。
    public static Long mapSize(java.util.Map<Object, Object> input) {
        return input.size();
    }
```

```
}
```

特定场景下，UDF需要通过`com.aliyun.odps.udf.ExecutionContext`（在`setup`方法中传入）获取一些上下文；UDT也可以通过`com.aliyun.odps.udt.UDTExecutionContext.get()`方法获取这样的一个`ExecutionContext`对象。

功能/性能/安全性

UDT在功能方面的优势如下：

- 使用简单，无需定义任何Function。
- 支持JDK的所有功能，扩展了SQL的能力。
- 代码可与SQL放于同一文件，便于管理。
- 可直接使用其它类库，代码重用率高。
- 可以使用面向对象的思想设计某些功能。

后续待完善功能如下：

- 支持无返回值的函数调用，或支持（有返回值但忽略返回值）直接取操作数本身的函数调用。例如，调用List的`add`方法会返回执行完`add`操作的List。
- 支持匿名类和Lambda表达式。
- 支持用作Shuffle Key。
- 支持Java外的其他语言，例如Python。

因为UDT和UDF的执行过程非常接近，所以UDT与UDF的性能几乎一致。优化后的计算引擎使得UDT在特定场景下的性能更高。

- UDT对象只有在跨进程时才需要做序列化和反序列化，因此在执行不需要数据Reshuffle的操作（如`join`或`ggregate`）时，UDT可节省序列化和反序列化的开销。
- 因为UDT的Runtime基于Codegen而非反射实现的，所以不存在反射带来的性能损失。在您使用过程中，连续多个UDT操作会合并在一个FunctionCall里一起执行。例如在之前的示例中，`values[x].add(values[y]).divide(java.math.BigInteger.valueOf(2))`实际上只会发生一次的UDT方法调用。所以，UDT操作的单元虽然较小，却并不会因多次函数调用而造成接口额外开销。

在安全控制方面，UDT和UDF完全一样，都会受到Java沙箱Policy的限制。所以如果要使用受限的操作，需要打开沙箱隔离，或者申请沙箱白名单。

4.19 UDJ

目前基于MaxCompute 2.0计算引擎，MaxCompute在UDF框架中新近引入的一种新扩展机制：UDJ(User Defined Join)，来实现灵活的跨表、多表自定义操作，同时减少不得不通过MR等方式对分布式系统底层细节的操作。这是MaxCompute基于新一代体系机构发展NewSQL数据处理框架的重要一步。

UDJ概述

目前MaxCompute内置了多种Join操作，包括inner/right join、outer/left jion、outer/full join、outer/semi/anti-semi join等。这些内置的join操作功能强大，能够满足很大一部分需求，但是其标准的join实现，无法满足很多跨表操作的需求。

很多情况下，可以通过UDF（User Defined Function）描述您的代码框架，但现有的UDF/UDTF/UDAF接口主要是针对在单个数据表上的操作而设计，一旦涉及多表的自定义操作，经常还需要依赖于内置join +各种UDF/UDTF, 并配合比较复杂的SQL语句来完成。甚至在一些多表操作的场景上，您不得不放弃SQL而转向传统的完全自定义MapReduce，才能完成所需的计算。

无论是join+各种UDF/UDTF+复杂SQL还是自定义MR门槛都比较高，同时还会带来一些问题：

- 对于计算平台而言，多个复杂的join和散布在SQL语言各处的代码揉合在一起，将带来多处的逻辑黑盒，这点不利于生成最优的执行计划。
- 使用MR，不仅更大程度上限制了系统进行执行优化的可能性，而且由于MR绝大部分代码由Java完成，在执行效率上会远低于MaxCompute基于LLVM 代码生成器产生的深度优化native运行时。

UDJ使用

接下来用一个样例来详细介绍MaxCompute的UDJ使用。

- 样例定义

  两个日志表，分别是payment和user_client_log。

  - payment (user_id string,time datetime,pay_info string)表中保存了用户的支付记录，一笔支付记录包含用户ID、支付时间和支付内容。
  - user_client_log(user_id string,time datetime,content string)表保存了用户的客户端日志，每一条日志包含了用户ID、日志时间和日志内容。

  需求：对于每一条客户端日志，找出该用户在payment表里时间最接近的一条支付记录，将其中的支付内容和日志内容合并输出。

  样例数据如下：

  payment

user_id	time	pay_info
2656199	2018-02-13 22:30:00	gZhvdySOQb
8881237	2018-02-13 08:30:00	pYvotuLDIT
8881237	2018-02-13 10:32:00	KBuMzRpsko

  user_client_log

user_id	time	content
8881237	2018-02-13 00:30:00	click MpkvilgWSmhUuPn
8881237	2018-02-13 06:14:00	click OkTYNUHMqZzlDyL

user_id	time	content
8881237	2018-02-13 10:30:00	click OkTYNUHMqZzlDyL

其中user_client_log的一条记录

user_id	time	content
8881237	2018-02-13 00:30:00	click MpkvilgWSmhUuPn

和payment中时间最接近的一条是

user_id	time	pay_info
8881237	2018-02-13 08:30:00	pYvotuLDIT

因此，这两条记录合并为：

8881237	2018-02-13 00:30:00	click MpkvilgWSmhUuPn, pay pYvotuLDIT

上面样例数据全部合并的结果如下：

user_id	time	content
8881237	2018-02-13 00:30:00	click MpkvilgWSmhUuPn, pay pYvotuLDIT
8881237	2018-02-13 06:14:00	click OkTYNUHMqZzlDyL, pay pYvotuLDIT
8881237	2018-02-13 10:30:00	click OkTYNUHMqZzlDyL, pay KBuMzRpsko

- 使用内置JOIN

  若使用标准join解决这个需求，会发现这个需求除了需要按照user_id进行关联以外，还需要知道payment中哪一条记录和user_client_log中的记录的时间差异值最小，勉强写SQL伪代码则类似于：

```
SELECT
  p.user_id,
  p.time,
  merge(p.pay_info, u.content)
FROM
  payment p RIGHT OUTER JOIN user_client_log u
```

```
ON p.user_id = u.user_id and abs(p.time - u.time) = min(abs(p.time
 - u.time))
```

关联时需要知道相同user_id下的p.time与u.time差异最小的值，而聚合函数不能出现在关联条件上。因此，这个看似简单的需求，无法通过标准的关联操作实现。

但在分布式系统中，Join操作实际上是将两个表按照某个(或多个)字段进行分组，并将同组数据集中到一处，而标准SQL中的Join对于关联后的操作有限，此时，若能提供一个通用编程语言接口，使用过程通过插件式的方式实现这个接口，在这个接口中将关联后的分组数据进行自定义处理并输出既可解决这个问题，这也是UDJ要解决的问题。

- 使用Java编写UDJ代码

  接下来详细介绍如何通过UDJ实现这个内置join无法实现的需求。

  前置条件

  这是一个新功能，我们需要一个比较新的SDK：

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-udf</artifactId>
  <version>0.29.10-public</version>
  <scope>provided</scope>
</dependency>
```

  新的SDK中包含了一个新的抽象类UDJ，我们通过继承这个类，来实现UDJ的功能：

```
package com.aliyun.odps.udf.example.udj;
import com.aliyun.odps.Column;
import com.aliyun.odps.OdpsType;
import com.aliyun.odps.Yieldable;
import com.aliyun.odps.data.ArrayRecord;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.udf.DataAttributes;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.UDJ;
import com.aliyun.odps.udf.annotation.Resolve;
import java.util.ArrayList;
import java.util.Iterator;
/** 对于右表的每个记录，找到最近的左表记录和
 * 合并两条记录。
 */
@Resolve("->string,bigint,string")
public class PayUserLogMergeJoin extends UDJ {
  private Record outputRecord;
  /** 将在数据处理阶段之前调用。 用户可以实施
   * 这个方法做初始化工作。
   */
  @Override
  public void setup(ExecutionContext executionContext, DataAttrib
utes dataAttributes) {
    //
    outputRecord = new ArrayRecord(new Column[]{
      new Column("user_id", OdpsType.STRING),
      new Column("time", OdpsType.BIGINT),
      new Column("content", OdpsType.STRING)
```

```
    });
  }
  /** 重写此方法以实现连接逻辑。
   * @param key 当前连接键
   * @param 左表对应当前键的记录组
   * @param right对应当前键的右表记录组
   * @param output用于输出UDJ的结果
   */
  @Override
  public void join(Record key, Iterator<Record> left, Iterator<
Record> right, Yieldable<Record> output) {
    outputRecord.setString(0, key.getString(0));
    if (!right.hasNext()) {
      // 空右组，什么都不做。
      return;
    } else if (!left.hasNext()) {
      // 空左组。 输出右侧组的所有记录而不合并。
      while (right.hasNext()) {
        Record logRecord = right.next();
        outputRecord.setBigint(1, logRecord.getDatetime(0).getTime
());
        outputRecord.setString(2, logRecord.getString(1));
        output.yield(outputRecord);
      }
      return;
    }
    ArrayList<Record> pays = new ArrayList<>();
    // 左侧记录组将从开始到结束迭代
    // 对于右组的每个记录，但迭代器无法重置。
    // 所以我们将左边的每个记录保存到ArrayList。
    left.forEachRemaining(pay -> pays.add(pay.clone()));
    while (right.hasNext()) {
      Record log = right.next();
      long logTime = log.getDatetime(0).getTime();
      long minDelta = Long.MAX_VALUE;
      Record nearestPay = null;
      // 迭代左边的所有记录，找到有的记录
      // 时间上的微小差异。
      for (Record pay: pays) {
        long delta = Math.abs(logTime - pay.getDatetime(0).getTime
());
        if (delta < minDelta) {
          minDelta = delta;
          nearestPay = pay;
        }
      }
      // 将日志记录与最近的支付记录合并，并输出到结果。
      outputRecord.setBigint(1, log.getDatetime(0).getTime());
      outputRecord.setString(2, mergeLog(nearestPay.getString(1),
log.getString(1)));
      output.yield(outputRecord);
    }
  }
  String mergeLog(String payInfo, String logContent) {
    return logContent + ", pay " + payInfo;
  }
  @Override
  public void close() {
  }
}
```

说明:

在本例中没有处理记录中的NULL值，为了使程序简洁便于演示，这里假设数据中没有NULL值。

从代码中可以看到，在每次调用UDJ的join方法时，两张表中各有一组对应着同一个key数据，提供给了我们。因此，只需遍历右表(user_client_log)的分组，对于每一个log记录，遍历一遍左表(payment)的分组，找出时间相差最小的记录，将日志内容合并然后输出即可。

这里假设同一个用户的支付记录数比较少，可以预先将左表分组全部加载到内存(通常情况下，内存足以存放一个用户在一天内产生的支付数据)。但若这个假设不成立应该如何解决？文章后面内容使用SORT BY预排序会解决这个问题。

- 在MaxCompute中创建UDJ

  编写完UDJ的Java代码后，需要将这部分代码插件式的嵌入到MaxCompute SQL中进行执行。再此之前，需要将代码注册到MaxCompute。假设上述代码打包成了*odps-udj-example.jar*。

  通过add jar命令将其当做jar资源上传到MaxCompute:

  ```
  add jar odps-udj-example.jar;
  ```

  通过create function语句注册UDJ函数，指定UDJ在SQL中的函数名`pay_user_log_merge_join`，以及关联上它对应的jar资源odps-udj-example.jar和在jar包中的类名`com.aliyun.odps.udf.example.udj.PayUserLogMergeJoin`:

  ```
  create function pay_user_log_merge_join
    as 'com.aliyun.odps.udf.example.udj.PayUserLogMergeJoin'
    using 'odps-udj-example.jar';
  ```

- 使用MaxCompute SQL进行UDJ查询

  UDJ注册好了以后，就可以在MaxCompute SQL中使用了。

  1. 创建示例源表：

     ```
     create table payment(user_id string,time datetime,pay_info string
     );
     create table user_client_log(user_id string,time datetime,content
     string);
     ```

  2. 制造演示数据：

     ```
     --制造payment表数据
     INSERT OVERWRITE TABLE payment VALUES
     ('1335656', datetime '2018-02-13 19:54:00', 'PEqMSHyktn'),
     ('2656199', datetime '2018-02-13 12:21:00', 'pYvotuLDIT'),
     ('2656199', datetime '2018-02-13 20:50:00', 'PEqMSHyktn'),
     ('2656199', datetime '2018-02-13 22:30:00', 'gZhvdySOQb'),
     ('8881237', datetime '2018-02-13 08:30:00', 'pYvotuLDIT'),
     ('8881237', datetime '2018-02-13 10:32:00', 'KBuMzRpsko'),
     ```

```
('9890100', datetime '2018-02-13 16:01:00', 'gZhvdySOQb'),
('9890100', datetime '2018-02-13 16:26:00', 'MxONdLckwa')
;
--制造user_client_log表数据
INSERT OVERWRITE TABLE user_client_log VALUES
('1000235', datetime '2018-02-13 00:25:36', 'click FNOXAibRjkIaQPB
'),
('1000235', datetime '2018-02-13 22:30:00', 'click GczrYaxvkiPultZ
'),
('1335656', datetime '2018-02-13 18:30:00', 'click MxONdLckpAFUHRS
'),
('1335656', datetime '2018-02-13 19:54:00', 'click mKRPGOciFDyzTgM
'),
('2656199', datetime '2018-02-13 08:30:00', 'click CZwafHsbJOPNitL
'),
('2656199', datetime '2018-02-13 09:14:00', 'click nYHJqIpjevkKToy
'),
('2656199', datetime '2018-02-13 21:05:00', 'click gbAfPCwrGXvEjpI
'),
('2656199', datetime '2018-02-13 21:08:00', 'click dhpZyWMuGjBOTJP
'),
('2656199', datetime '2018-02-13 22:29:00', 'click bAsxnUdDhvfqaBr
'),
('2656199', datetime '2018-02-13 22:30:00', 'click XIhZdLaOocQRmrY
'),
('4356142', datetime '2018-02-13 18:30:00', 'click DYqShmGbIoWKier
'),
('4356142', datetime '2018-02-13 19:54:00', 'click DYqShmGbIoWKier
'),
('8881237', datetime '2018-02-13 00:30:00', 'click MpkvilgWSmhUuPn
'),
('8881237', datetime '2018-02-13 06:14:00', 'click OkTYNUHMqZzlDyL
'),
('8881237', datetime '2018-02-13 10:30:00', 'click OkTYNUHMqZzlDyL
'),
('9890100', datetime '2018-02-13 16:01:00', 'click vOTQfBFjcgXisYU
'),
('9890100', datetime '2018-02-13 16:20:00', 'click WxaLgOCcVEvhiFJ
')
;
```

3. **在MaxCompute SQL中使用刚刚创建好的UDJ函数：**

```
SELECT r.user_id, from_unixtime(time/1000) as time, content FROM (
SELECT user_id, time as time, pay_info FROM payment
) p JOIN (
SELECT user_id, time as time, content FROM user_client_log
) u
ON p.user_id = u.user_id
USING pay_user_log_merge_join(p.time, p.pay_info, u.time, u.
content)
r
AS (user_id, time, content)
```

```
;
```

UDJ的语法与标准Join语法类似，主要差异多了using字句：

- pay_user_log_merge_join是注册的UDJ在SQL中的函数名；
- (p.time, p.pay_info, u.time, u.content)是UDJ中分别用到的左右表的列；
- r是UDJ结果的别名，用于其他地方引用UDJ的结果；
- (user_id, time, content)是UDJ产生的结果的列名。

运行上面这条SQL，结果为：

```
+---------+------------+---------+
| user_id | time       | content |
+---------+------------+---------+
| 1000235 | 2018-02-13 00:25:36 | click FNOXAibRjkIaQPB |
| 1000235 | 2018-02-13 22:30:00 | click GczrYaxvkiPultZ |
| 1335656 | 2018-02-13 18:30:00 | click MxONdLckpAFUHRS, pay
PEqMSHyktn |
| 1335656 | 2018-02-13 19:54:00 | click mKRPGOciFDyzTgM, pay
PEqMSHyktn |
| 2656199 | 2018-02-13 08:30:00 | click CZwafHsbJOPNitL, pay
pYvotuLDIT |
| 2656199 | 2018-02-13 09:14:00 | click nYHJqIpjevkKToy, pay
pYvotuLDIT |
| 2656199 | 2018-02-13 21:05:00 | click gbAfPCwrGXvEjpI, pay
PEqMSHyktn |
| 2656199 | 2018-02-13 21:08:00 | click dhpZyWMuGjBOTJP, pay
PEqMSHyktn |
| 2656199 | 2018-02-13 22:29:00 | click bAsxnUdDhvfqaBr, pay
gZhvdySOQb |
| 2656199 | 2018-02-13 22:30:00 | click XIhZdLaOocQRmrY, pay
gZhvdySOQb |
| 4356142 | 2018-02-13 18:30:00 | click DYqShmGbIoWKier |
| 4356142 | 2018-02-13 19:54:00 | click DYqShmGbIoWKier |
| 8881237 | 2018-02-13 00:30:00 | click MpkvilgWSmhUuPn, pay
pYvotuLDIT |
| 8881237 | 2018-02-13 06:14:00 | click OkTYNUHMqZzlDyL, pay
pYvotuLDIT |
| 8881237 | 2018-02-13 10:30:00 | click OkTYNUHMqZzlDyL, pay
KBuMzRpsko |
| 9890100 | 2018-02-13 16:01:00 | click vOTQfBFjcgXisYU, pay
gZhvdySOQb |
| 9890100 | 2018-02-13 16:20:00 | click WxaLgOCcVEvhiFJ, pay
MxONdLckwa |
+---------+------------+---------+
```

如上述代码，通过使用UDJ完成了内置join无法轻松完成的需求。

· UDJ预排序功能

前面的UDJ代码抛出过这样的问题：为了找到payment中相差最小的一条记录，需要反复对payment表的数据进行iterator遍历，所以事先将相同user_id的payment记录全部加载到了ArrayList，当同一个用户一天之内的支付行为比较少时，这个方式可用，但在其它场景中，有

时候同组内的数据可能非常大，大到无法在内存中放下，此时就需要通过其他方式解决。本小节将介绍通过SORT BY预排序解决这个场景。

当某个用户的支付数据量非常巨大导致无法将payment放在内存中时，仔细分析会发现组内所有数据如果已经按照time排好序，那么这个问题可以这样解决：只需要比较两边iterator最”顶端”的数据，就可以实现这个功能。

java UDJ代码如下：

```
@Override
public void join(Record key, Iterator<Record> left, Iterator<Record
> right, Yieldable<Record> output) {
  outputRecord.setString(0, key.getString(0));
  if (!right.hasNext()) {
    return;
  } else if (!left.hasNext()) {
    while (right.hasNext()) {
      Record logRecord = right.next();
      outputRecord.setBigint(1, logRecord.getDatetime(0).getTime());
      outputRecord.setString(2, logRecord.getString(1));
      output.yield(outputRecord);
    }
    return;
  }
  long prevDelta = Long.MAX_VALUE;
  Record logRecord = right.next();
  Record payRecord = left.next();
  Record lastPayRecord = payRecord.clone();
  while (true) {
    long delta = logRecord.getDatetime(0).getTime() - payRecord.
getDatetime(0).getTime();
    if (left.hasNext() && delta > 0) {
      //两个记录之间的时间差正在减少，我们仍然可以操作
      //探索左侧组以尝试获得更小的增量。
      lastPayRecord = payRecord.clone();
      prevDelta = delta;
      payRecord = left.next();
    } else {
     //到达最小delta点。 检查最后的工资记录，
     //输出合并结果并准备处理下一条记录
     //右组.
      Record nearestPay = Math.abs(delta) < prevDelta ? payRecord :
lastPayRecord;
      outputRecord.setBigint(1, logRecord.getDatetime(0).getTime());
      String mergedString = mergeLog(nearestPay.getString(1),
logRecord.getString(1));
      outputRecord.setString(2, mergedString);
      output.yield(outputRecord);
      if (right.hasNext()) {
        logRecord = right.next();
        prevDelta = Math.abs(
          logRecord.getDatetime(0).getTime() - lastPayRecord.
getDatetime(0).getTime()
        );
      } else {
        break;
      }
    }
  }
```

```
}
```

SQL语句中，只需要对前面的例子稍作修改，在UDJ语句尾部增加SORT BY子句，指定UDJ组内左右表分别都按照各自的time字段进行排序。

说明:

注意：UDJ代码修改后需要更新UDJ对应的jar包：

```
SELECT r.user_id, from_unixtime(time/1000) as time, content FROM (
  SELECT user_id, time as time, pay_info FROM payment
) p JOIN (
  SELECT user_id, time as time, content FROM user_client_log
) u
ON p.user_id = u.user_id
USING pay_user_log_merge_join(p.time, p.pay_info, u.time, u.content
)
r
AS (user_id, time, content)
SORT BY p.time, u.time
;
```

执行结果将和前面的方式执行结果一样。

该方式主要是使用SORT BY子句对UDJ的数据进行预排序，在这个过程中最多只需要同时缓存3条记录，就可以实现和之前算法的相同的功能。

UDJ的性能

当前通常情况下，在面对复杂的跨表计算需求时，因为缺乏UDJ这种机制，不得不借助MR来实现整套跨表计算的分布式流程。这类场景业务复杂的BU（比如搜索，广告等）尤为常见。

在此用一个线上线上真实的MR作业进行测试验证UDJ的性能。该MR作业实现一套比较复杂的算法将两个表合并一起，用UDJ对该MR进行改写，并且验证新的UDJ实现结果的正确性。性能方面，在并发度相同的情况下两者性能对比如下：



由图可见，UDJ接口的引入，一方面更加方便的描述对多表数据进行操作的复杂逻辑，一方面大幅度提高了性能（代码只有在UDJ内被调用，其上下游的逻辑（比如这个例子中的整个mapper逻辑）则完全通过MaxCompute高效的native运行时完成）。在Java代码中，由于UDJ对MaxCompute运行时引擎和Java接口之间的数据交互逻辑做了深度的优化，通过UDJ实现的join逻辑，也比其对等的reducer更高效。

4.20 附录

4.20.1 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

转义	字符
\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”。

4.20.2 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_' -- false
    'ab_cde' like 'ab\\_c%'; -- true
```

注意:
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。

4.20.3 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

元字符	说明
.	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]
[[: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]

POSIX字符组	说明	范围
[[: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 |
```

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

4.20.4 保留字与关键字

本文将为您介绍MaxCompute SQL中的所有保留字与关键字。

注意:

- 在对表、列或分区进行命名时，请不要使用保留字与关键字，可能会报错。
- 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
```

4.20.5 与Hive数据类型映射表

MaxCompute与Hive的数据类型映射表，如下所示：

Hive 数据类型	MaxCompute 数据类型
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
DEICIMAL	DEICIMAL
STRING	STRING
VARCHAR	VARCHAR
CHAR	STRING
BINARY	BINARY
TIMESTAMP	TIMESTAMP
DATE	Datetime
ARRAY	ARRAY
`MAP<key,value>`	MAP
STRUCT	STRUCT
UNION	不支持

5 MapReduce

5.1 概要

5.1.1 MapReduce概述

本文为您介绍MaxCompute支持的MapReduce编程接口及使用限制。

MaxCompute提供三个版本的MapReduce编程接口。

- MaxCompute MapReduce：MaxCompute的原生接口，执行速度更快，开发更便捷，不暴露文件系统。
- MaxCompute 扩展MapReduceMR2）：对MaxCompute MapReduce的扩展，支持更复杂的作业调度逻辑。MapReduce的实现方式与MaxCompute原生接口一致。
- Hadoop兼容版本MapReduce：高度兼容Hadoop MapReduce，与MaxCompute MapReduce MR2不兼容。

以上三个版本在基本概念、作业提交、输入与输出、资源使用等方面基本一致，仅各版本的Java SDK有所不同。本文仅对MapReduce的基本原理做简单介绍，更多详情请参见Hadoop Map/Reduce教程。

> 说明:
> 您无法通过MapReduce读写外部表中的数据。

应用场景

MapReduce支持下列场景：

- 搜索：网页爬取、倒排索引、PageRank。
- Web访问日志分析：
  - 分析和挖掘用户在Web上的访问、购物行为特征，实现个性化推荐。
  - 分析用户访问行为。
- 文本统计分析：
  - 热门小说的字数统计（WordCount）、词频TFIDF分析。
  - 学术论文、专利文献的引用分析和统计。
  - 维基百科数据分析等。
- 海量数据挖掘：非结构化数据、时空数据、图像数据的挖掘。
- 机器学习：监督学习、无监督学习、分类算法如决策树、SVM等。

- 自然语言处理：
  - 基于大数据的训练和预测。
  - 基于语料库构建单词同现矩阵，频繁项集数据挖掘、重复文档检测等。
- 广告推荐：用户点击（CTR）和购买行为（CVR）预测。

处理流程

MapReduce处理数据过程主要分成Map和Reduce两个阶段。首先执行Map阶段，再执行Reduce阶段。Map和Reduce的处理逻辑由用户自定义实现，但要符合MapReduce框架的约定。处理流程如下所示：

1. 在正式执行Map前，需要将输入数据进行分片。所谓分片，就是将输入数据切分为大小相等的数据块，每一块作为单个Map Worker的输入被处理，以便于多个Map Worker同时工作。
2. 分片完毕后，多个Map Worker便可同时工作。每个Map Worker在读入各自的数据后，进行计算处理，最终输出给Reduce。Map Worker在输出数据时，需要为每一条输出数据指定一个Key，这个Key值决定了这条数据将会被发送给哪一个Reduce Worker。Key值和Reduce Worker是多对一的关系，具有相同Key的数据会被发送给同一个Reduce Worker，单个Reduce Worker有可能会接收到多个Key值的数据。
3. 在进入Reduce阶段之前，MapReduce框架会对数据按照Key值排序，使得具有相同Key的数据彼此相邻。如果您指定了合并操作（Combiner），框架会调用Combiner，将具有相同Key的数据进行聚合。Combiner的逻辑可以由您自定义实现。与经典的MapReduce框架协议不同，在MaxCompute中，Combiner的输入、输出的参数必须与Reduce保持一致，这部分的处理通常也叫做洗牌（Shuffle）。
4. 接下来进入Reduce阶段。相同Key的数据会到达同一个Reduce Worker。同一个Reduce Worker会接收来自多个Map Worker的数据。每个Reduce Worker会对Key相同的多个数据进行Reduce操作。最后，一个Key的多条数据经过Reduce的作用后，将变成一个值。

说明：

上文仅是对MapReduce框架做简单介绍，更多详情请查阅功能介绍。

下文将以WordCount为例，为您介绍MaxCompute MapReduce各个阶段的概念。

假设存在一个文本a.txt，文本内每行是一个数字，您要统计每个数字出现的次数。文本内的数字称为Word，数字出现的次数称为Count。如果MaxCompute MapReduce完成这一功能，需要经历以下流程，如下图所示：



操作步骤

1. 输入数据：对文本进行分片，将每片内的数据作为单个Map Worker的输入。
2. Map阶段：Map处理输入，每获取一个数字，将数字的Count 设置为1，并将此<Word, Count>对输出，此时以Word作为输出数据的Key。
3. Shuffle>合并排序：在Shuffle阶段前期，首先对每个Map Worker的输出，按照Key值（即Word值）进行排序。排序后进行Combiner操作，即将Key值（Word值）相同的Count累加，构成一个新的<Word, Count>对。此过程被称为合并排序。
4. Shuffle>分配Reduce：在Shuffle阶段后期，数据被发送到Reduce端。Reduce Worker收到数据后依赖Key值再次对数据排序。
5. Reduce阶段：每个Reduce Worker对数据进行处理时，采用与Combiner相同的逻辑，将Key值（Word 值）相同的Count累加，得到输出结果。
6. 输出结果数据。

说明:
由于MaxCompute的所有数据都被存放在表中，因此MaxCompute MapReduce的输入、输出只能是表，不允许您自定义输出格式，不提供类似文件系统的接口。

使用限制

MapReduce使用限制汇总。

有关本地运行的MapReduce使用限制，您可以参考本地运行和分布式环境运行差异。

5.1.2 扩展MapReduce

相比于传统的MapRudce，MaxCompute提供的扩展MapReduce模型（简称MR2）改变了底层的调度和IO模型，可避免作业时冗余的IO操作。

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。

MR2 模型产生背景

传统的MapReduce模型要求在经过每一轮MapReduce操作后，得到的数据结果必须存储到分布式文件系统中（例如，HDFS或MaxCompute数据表）。MapReduce模型通常由多个MapReduce作业组成，每个作业执行完成后都需要将数据写入磁盘，然而后续的Map任务很可能只需要读取一遍这些数据，为之后的Shuffle阶段做准备，这种情况就产生了冗余的磁盘IO操作。

MaxCompute的计算调度逻辑可以支持更复杂的编程模型，针对上述情况，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。

与Hadoop Chain Mapper/Reducer对比

Hadoop Chain Mapper/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。

Chain Mapper/Reducer基于传统的MapReduce模型，仅可以在原有的Mapper或Reducer后增加一个或多个Mapper操作（不允许增加Reducer）。这样的优点是：您可以复用之前的Mapper业务逻辑，把一个Map或Reduce拆成多个Mapper阶段，同时本质上并没有改变底层的调度和I/O模型。

5.1.3 开源兼容MapReduce

本文为您介绍开源兼容MapReduce的应用背景，以及HadoopMR插件的基本使用方式。

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，二者的编程接口差异较大。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。您可通过单击此处下载开发插件。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*openmr_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.mapr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。

```
odps@ zhe>read wc_out;
+------------+------------+
| key        | cnt        |
+------------+------------+
| hello      | 2          |
| mapreduce  | 1          |
| maxcompute | 1          |
+------------+------------+
odps@ zhe>
```

5.2 功能介绍

5.2.1 基本概念

本章节向您介绍如何实现MapReduce排序、自定义哈希函数、归并功能。

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。

5.2.2 作业提交

本文向您介绍如何使用MaxCompute客户端运行和提交MapReduce作业。

MaxCompute客户端提供一个Jar命令用于运行MapReduce作业，具体语法如下所示：

```
jar [<GENERIC_OPTIONS>] <MAIN_CLASS> [ARGS];
        -conf <configuration_file>          Specify an application
configuration file
        -resources <resource_name_list>     file\table resources used
in mapper or reducer, seperate by comma
        -classpath <local_file_list>        classpaths used to run
mainClass
        -D <name>=<value>                   Property value pair, which
will be used to run mainClass
        -l                                  Run job in local mode
```

举例如下：

```
jar -conf \home\admin\myconf -resources a.txt,example.jar -classpath
 ..\lib\example.jar:.\other_lib.jar -D java.library.path=.\native;
```

其中 `<GENERIC_OPTIONS>` 包括（均为可选参数）：

- `-conf <configuration file>`：指定JobConf配置文件。
- `-resources <resource_name_list>`：MapReduce作业运行时使用的资源声明。一般情况下，resource_name_list中需要指定Map/Reduce函数所在的资源名称。

  > **说明：**
  >
  > 如果您在Map/Reduce函数中读取了其他MaxCompute资源，则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，需要前面加上PROJECT/resources/，例如：-resources otherproject/resources/resfile。

有关如何在Map/Reduce函数中读取资源的示例，请参见资源使用示例。

- -classpath <local_file_list>：本地执行时的classpath，主要用于指定main函数所在的Jar包的本地路径（包含相对路径和绝对路径）。

  包名之间使用系统默认的文件分割符作分割。通常：Windows系统中使用分号（;）；Linux系统中使用逗号（,）。如果您在云端服务器运行MR任务，则使用逗号（,）进行分隔。

  说明：
  通常，您可能更习惯于将main函数与Map/Reduce函数编写在一个包中，例如：WordCount代码示例。因此，在执行示例程序时，-resources及-classpath的参数中都出现了mapreduce-examples.jar。但二者意义不同：-resources引用的是Map/Reduce函数，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。

- -D <prop_name>=<prop_value>：本地执行时，<mainClass>的Java属性，可以定义多个。
- -l：以本地模式执行MapReduce作业，主要用于程序调试。

您可以通过-conf选项指定JobConf配置文件，该文件可以影响SDK中JobConf的设置。

JobConf配置文件的示例如下：

```
<configuration>
       <property>
          <name>import.filename</name>
          <value>resource.txt</value>
       </property>
    </configuration>
```

在上述示例中，通过JobConf配置文件定义一个名为import.filename的变量，该变量的值为resource.txt。

您可以在MapReduce程序中通过JobConf接口获取该变量的值。您通过SDK中JobConf接口可以达到相同的目的，详情请参见资源使用示例 。

示例如下：

```
add jar data\mapreduce-examples.jar;
    jar -resources mapreduce-examples.jar -classpath data\mapreduce-
examples.jar
        org.alidata.odps.mr.examples.WordCount wc_in wc_out;
    add file data\src.txt;
    add jar data\mapreduce-examples.jar;
```

```
    jar -resources src.txt,mapreduce-examples.jar -classpath data\
mapreduce-examples.jar
        org.alidata.odps.mr.examples.WordCount wc_in wc_out;
    add file data\a.txt;
    add table wc_in as test_table;
    add jar data\work.jar;
    jar -conf odps-mapred.xml -resources a.txt,test_table,work.jar
        -classpath data\work.jar:otherlib.jar
        -D import.filename=resource.txt org.alidata.odps.mr.examples.
WordCount args;
```

5.2.3 输入与输出

本文向您介绍MapReduce的输入与输出。

- MaxCompute MapReduce的输入、输出，支持MaxCompute内置类型的Bigint、Double、String、Datetime和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label，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。

5.2.4 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使用资源示例。

5.2.5 本地运行

本文向您介绍本地和分布式环境运行MapReduce的差异和MapReduce本地运行示例。

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*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*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。

5.3 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，而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256。

边界名	边界值	分类	配置项名称	默认值	是否可配置	说明
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.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 。

边界名	边界值	分类	配置项名称	默认值	是否可配置	说明
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。
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。
MR是否支持读取OSS数据	不涉及	功能限制	不涉及	无	否	MR不支持读取OSS数据。
MR是否支持MaxCompute 2.0新类型	不涉及	功能限制	不涉及	无	否	MR不支持MaxCompute 2.0新类型。

5.4 示例程序

5.4.1 WordCount示例

本文向您介绍MapReduce WordCount示例程序。

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();
        }
        //combiner实现的接口和reducer一样，可以立即为在mapper本地执行的一个
reduce，作用是减少mapper的输出量
        @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);
       //设置mapper中间结果的key和value的schema，mapper的中间结果输出也是
record的形式
        job.setMapOutputKeySchema(SchemaUtils.fromString("word:string
"));
        job.setMapOutputValueSchema(SchemaUtils.fromString("count:
bigint"));
        //设置输入和输出的表信息
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).
build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[1]).
build(), job);
        JobClient.runJob(job);
      }
    }
```

5.4.2 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);
          //Main函数在jobconf里设置了option.mapper.setup为true，才会执行下
面的逻辑
          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);
          //Main函数在jobconf里设置了option.mapper.map为true，才会执行下面
的逻辑
          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);
          //Main函数在jobconf里设置了option.mapper.cleanup为true，才会执行
下面的逻辑
          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);
        //对于MapOnly的作业，必须显式设置reducer的个数为0
        job.setNumReduceTasks(0);
        //设置输入输出的表信息
        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]);
        //jobconf中可以设置自定义的key,value值，在mapper中通过context的
getJobConf可以获取到相关的设置
          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);
      }
```

```
    }
```

5.4.3 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 {
          //对于不同的输出需要创建不同的record，通过label来区分
          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");
        }
      }
      //将分区字符串如"ds=1/pt=2"转为map的形式
      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);
      }
    }
```

5.4.4 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;
          //读取资源表里面的数据，这个表是上一个job的输出表
          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);
          //设置counter，counter在作业成功结束后，可以在main函数中获取到
          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"));
        //maponly作业需要显式设置reducer的数目为0
        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);
          //maponly作业需要显式设置reducer的数目为0
          decJob.setNumReduceTasks(0);
          RunningJob rJob = JobClient.runJob(decJob);
          iterCount--;
          //如果迭代次数已经达到，则退出循环
          if (rJob.getCounters().findCounter("multijobs", "value").
getValue() == 0) {
            break;
          }
        }
        if (iterCount != 0) {
          throw new IOException("Job failed.");
        }
      }
    }
```

5.4.5 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);
      }
    }
```

5.4.6 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);
        //设置资源名字，可以在map中通过jobconf获取到
        job.set("import.filename", args[0]);
        //maponly作业需要显式设置reducer的数目为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。

5.4.7 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);
        //在作业成功结束后，可以获取到job里面的自定义counter的值
        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);
      }
    }
```

5.4.8 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);
        //设置grepJob的grep的正则表达式
        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);
        //grepJob的输出作为sortJob的输入
        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);
      }
```

```
    }
```

5.4.9 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();
        }
        //reduce函数每次的输入会是key相同的所有record
        @Override
        public void reduce(Record key, Iterator<Record> values,
TaskContext context)
            throws IOException {
          long k = key.getBigint(0);
          List<Object[]> leftValues = new ArrayList<Object[]>();
          //由于设置了outputKeySortColumn是key + tag组合，这样可以保证
reduce函数的输入record中，left表的record数据在前面
          while (values.hasNext()) {
            Record value = values.next();
            long tag = (Long) key.get(1);
            //左表的数据会先缓存到内存中
            if (tag == 0) {
              leftValues.add(value.toArray().clone());
            } else {
              //碰到右表的数据，会与所有左表的数据进行join输出，此时左表的数据已
经全部在内存里了
         //这个实现只是一个功能展示，性能比较低，不建议用于实际生产
              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);
```

```
    }
  }
```

5.4.10 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 {
    //由于没有输入数据，map函数不会被执行，相关逻辑只能写在setup中
    @Override
    public void setup(TaskContext context) throws IOException {
      try {
        //获取jobconf中设置的sleep秒数，进行sleep
        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);
```

```
    //这个实例也是一个MapOnly的，所以需要设置reducer个数为0
    job.setNumReduceTasks(0);
    //由于没有输入表，mapper的个数需要由用户明确进行指定
    job.setNumMapTasks(1);
    job.set(SLEEP_SECS, args[0]);
    JobClient.runJob(job);
  }
}
```

5.4.11 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];
        }
        // reduce的输入分组是由setOutputGroupingColumns的设置来决定的，这个
参数如果不设置
        // 默认就是MapOutputKeySchema
        // 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);
          }
        }
      }
```

5.4.12 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);
        //为了全局有序，这里设置了reducer的个数为1，所有的数据都会集中到一个
reducer上面
        //只能用于小数据量，大数据量需要考虑其他的方式，比如TeraSort
        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.");
      }
    }
```

5.4.13 分区表输入示例

本文为您介绍两个把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。**

5.4.14 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();
        //将pipeline的设置到jobconf中，如果需要设置combiner，是通过jobconf来
设置
        job.setPipeline(pipeline);
        //设置输入输出表
        job.addInput(TableInfo.builder().tableName(args[0]).build());
        job.addOutput(TableInfo.builder().tableName(args[1]).build());
        //作业提交并等待结束
        job.submit();
        job.waitForCompletion();
        System.exit(job.isSuccessful() == true ? 0 : 1);
      }
    }
```

5.5 Java SDK

5.5.1 原生SDK概述

本文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。
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label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。
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。

注意:

- MaxCompute的TaskContext接口中提供了progress功能，但此功能是为防止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。
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)	设置分片大小，单位MB，默认值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，再按照KeySortColumns排序后，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之后，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，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	Java Type
Bigint	Long
String	String
Double	Double
Boolean	Boolean
Datetime	Date
Decimal	BigDecimal

5.5.2 兼容版本SDK概述

MaxCompute兼容版本的MapReduce与Hadoop MapReduce兼容性的详细列表，如下表所示：

类型	接口	是否兼容
Mapper	void map(KEYIN key, VALUEIN value , org.apache.hadoop.mapreduce. Mapper.Context context)	是
Mapper	void run(org.apache.hadoop. mapreduce.Mapper.Context context)	是
Mapper	void setup(org.apache.hadoop. mapreduce.Mapper.Context context)	是
Reducer	void cleanup(org.apache.hadoop. mapreduce.Reducer.Context context)	是

类型	接口	是否兼容
Reducer	void reduce(KEYIN key, VALUEIN value, org.apache.hadoop.mapreduce.Reducer.Context context)	是
Reducer	void run(org.apache.hadoop.mapreduce.Reducer.Context context)	是
Reducer	void setup(org.apache.hadoop.mapreduce.Reducer.Context context)	是
Partitioner	int getPartition(KEY key, VALUE value, int numPartitions)	是
MapContext（继承TaskInputOutputContext）	InputSplit getInputSplit()	否，抛异常
ReduceContext	nextKey()	是
ReduceContext	getValues()	是
TaskInputOutputContext	getCurrentKey()	是
TaskInputOutputContext	getCurrentValue()	是
TaskInputOutputContext	getOutputCommitter()	否，抛异常
TaskInputOutputContext	nextKeyValue()	是
TaskInputOutputContext	write(KEYOUT key, VALUEOUT value)	是
TaskAttemptContext	getCounter(Enum<?> counterName)	是
TaskAttemptContext	getCounter(String groupName, String counterName)	是
TaskAttemptContext	setStatus(String msg)	空实现
TaskAttemptContext	getStatus()	空实现
TaskAttemptContext	getTaskAttemptID()	否，抛异常
TaskAttemptContext	getProgress()	否，抛异常
TaskAttemptContext	progress()	是
Job	addArchiveToClassPath(Path archive)	否
Job	addCacheArchive(URI uri)	否
Job	addCacheFile(URI uri)	否
Job	addFileToClassPath(Path file)	否
Job	cleanupProgress()	否
Job	createSymlink()	否，抛异常

类型	接口	是否兼容
Job	failTask(TaskAttemptID taskId)	否
Job	getCompletionPollInterval(Configuration conf)	空实现
Job	getCounters()	是
Job	getFinishTime()	是
Job	getHistoryUrl()	是
Job	getInstance()	是
Job	getInstance(Cluster ignored)	是
Job	getInstance(Cluster ignored, Configuration conf)	是
Job	getInstance(Configuration conf)	是
Job	getInstance(Configuration conf, String jobName)	空实现
Job	getInstance(JobStatus status, Configuration conf)	否，抛异常
Job	getJobFile()	否，抛异常
Job	getJobName()	空实现
Job	getJobState()	否，抛异常
Job	getPriority()	否，抛异常
Job	getProgressPollInterval(Configuration conf)	空实现
Job	getReservationId()	否，抛异常
Job	getSchedulingInfo()	否，抛异常
Job	getStartTime()	是
Job	getStatus()	否，抛异常
Job	getTaskCompletionEvents(int startFrom)	否，抛异常
Job	getTaskCompletionEvents(int startFrom, int numEvents)	否，抛异常
Job	getTaskDiagnostics(TaskAttemptID taskid)	否，抛异常
Job	getTaskOutputFilter(Configuration conf)	否，抛异常

类型	接口	是否兼容
Job	getTaskReports(TaskType type)	否，抛异常
Job	getTrackingURL()	是
Job	isComplete()	是
Job	isRetired()	否，抛异常
Job	isSuccessful()	是
Job	isUber()	空实现
Job	killJob()	是
Job	killTask(TaskAttemptID taskId)	否
Job	mapProgress()	是
Job	monitorAndPrintJob()	是
Job	reduceProgress()	是
Job	setCacheArchives(URI[] archives)	否，抛异常
Job	setCacheFiles(URI[] files)	否，抛异常
Job	setCancelDelegationTokenUponJobCompletion(boolean value)	否，抛异常
Job	setCombinerClass(Class<? extends Reducer> cls)	是
Job	setCombinerKeyGroupingComparatorClass(Class<? extends RawComparator> cls)	是
Job	setGroupingComparatorClass(Class<? extends RawComparator> cls)	是
Job	setInputFormatClass(Class<? extends InputFormat> cls)	空实现
Job	setJar(String jar)	是
Job	setJarByClass(Class<?> cls)	是
Job	setJobName(String name)	空实现
Job	setJobSetupCleanupNeeded(boolean needed)	空实现
Job	setMapOutputKeyClass(Class<?> theClass)	是
Job	setMapOutputValueClass(Class<?> theClass)	是

类型	接口	是否兼容
Job	setMapperClass(Class<? extends Mapper> cls)	是
Job	setMapSpeculativeExecution(boolean speculativeExecution)	空实现
Job	setMaxMapAttempts(int n)	空实现
Job	setMaxReduceAttempts(int n)	空实现
Job	setNumReduceTasks(int tasks)	是
Job	setOutputFormatClass(Class<? extends OutputFormat> cls)	否，抛异常
Job	setOutputKeyClass(Class<?> theClass)	是
Job	setOutputValueClass(Class<?> theClass)	是
Job	setPartitionerClass(Class<? extends Partitioner> cls)	是
Job	setPriority(JobPriority priority)	否，抛异常
Job	setProfileEnabled(boolean newValue)	空实现
Job	setProfileParams(String value)	空实现
Job	setProfileTaskRange(boolean isMap, String newValue)	空实现
Job	setReducerClass(Class<? extends Reducer> cls)	是
Job	setReduceSpeculativeExecution(boolean speculativeExecution)	空实现
Job	setReservationId(ReservationId reservationId)	否，抛异常
Job	setSortComparatorClass(Class<? extends RawComparator> cls)	否，抛异常
Job	setSpeculativeExecution(boolean speculativeExecution)	是
Job	setTaskOutputFilter(Configuration conf, org.apache.hadoop.mapreduce.Job.TaskStatusFilter newValue)	否，抛异常
Job	setupProgress()	否，抛异常
Job	setUser(String user)	空实现

类型	接口	是否兼容
Job	setWorkingDirectory(Path dir)	空实现
Job	submit()	是
Job	toString()	否，抛异常
Job	waitForCompletion(boolean verbose)	是
Task Execution & Environment	mapreduce.map.java.opts	空实现
Task Execution & Environment	mapreduce.reduce.java.opts	空实现
Task Execution & Environment	mapreduce.map.memory.mb	空实现
Task Execution & Environment	mapreduce.reduce.memory.mb	空实现
Task Execution & Environment	mapreduce.task.io.sort.mb	空实现
Task Execution & Environment	mapreduce.map.sort.spill.percent	空实现
Task Execution & Environment	mapreduce.task.io.soft.factor	空实现
Task Execution & Environment	mapreduce.reduce.merge.inmem.thresholds	空实现
Task Execution & Environment	mapreduce.reduce.shuffle.merge.percent	空实现
Task Execution & Environment	mapreduce.reduce.shuffle.input.buffer.percent	空实现
Task Execution & Environment	mapreduce.reduce.input.buffer.percent	空实现
Task Execution & Environment	mapreduce.job.id	空实现
Task Execution & Environment	mapreduce.job.jar	空实现
Task Execution & Environment	mapreduce.job.local.dir	空实现
Task Execution & Environment	mapreduce.task.id	空实现
Task Execution & Environment	mapreduce.task.attempt.id	空实现
Task Execution & Environment	mapreduce.task.is.map	空实现
Task Execution & Environment	mapreduce.task.partition	空实现
Task Execution & Environment	mapreduce.map.input.file	空实现
Task Execution & Environment	mapreduce.map.input.start	空实现
Task Execution & Environment	mapreduce.map.input.length	空实现
Task Execution & Environment	mapreduce.task.output.dir	空实现

类型	接口	是否兼容
JobClient	cancelDelegationToken(Token < DelegationTokenIdentifier> token)	否，抛异常
JobClient	close()	空实现
JobClient	displayTasks(JobID jobId, String type, String state)	否，抛异常
JobClient	getAllJobs()	否，抛异常
JobClient	getCleanupTaskReports(JobID jobId)	否，抛异常
JobClient	getClusterStatus()	否，抛异常
JobClient	getClusterStatus(boolean detailed)	否，抛异常
JobClient	getDefaultMaps()	否，抛异常
JobClient	getDefaultReduces()	否，抛异常
JobClient	getDelegationToken(Text renewer)	否，抛异常
JobClient	getFs()	否，抛异常
JobClient	getJob(JobID jobid)	否，抛异常
JobClient	getJob(String jobid)	否，抛异常
JobClient	getJobsFromQueue(String queueName)	否，抛异常
JobClient	getMapTaskReports(JobID jobId)	否，抛异常
JobClient	getMapTaskReports(String jobId)	否，抛异常
JobClient	getQueueAclsForCurrentUser()	否，抛异常
JobClient	getQueueInfo(String queueName)	否，抛异常
JobClient	getQueues()	否，抛异常
JobClient	getReduceTaskReports(JobID jobId)	否，抛异常
JobClient	getReduceTaskReports(String jobId)	否，抛异常
JobClient	getSetupTaskReports(JobID jobId)	否，抛异常
JobClient	getStagingAreaDir()	否，抛异常
JobClient	getSystemDir()	否，抛异常
JobClient	getTaskOutputFilter()	否，抛异常
JobClient	getTaskOutputFilter(JobConf job)	否，抛异常
JobClient	init(JobConf conf)	否，抛异常

类型	接口	是否兼容
JobClient	isJobDirValid(Path jobDirPath, FileSystem fs)	否，抛异常
JobClient	jobsToComplete()	否，抛异常
JobClient	monitorAndPrintJob(JobConf conf, RunningJob job)	否，抛异常
JobClient	renewDelegationToken(Token< DelegationTokenIdentifier> token)	否，抛异常
JobClient	run(String[] argv)	否，抛异常
JobClient	runJob(JobConf job)	是
JobClient	setTaskOutputFilter(JobClient. TaskStatusFilter newValue)	否，抛异常
JobClient	setTaskOutputFilter(JobConf job, JobClient.TaskStatusFilter newValue)	否，抛异常
JobClient	submitJob(JobConf job)	是
JobClient	submitJob(String jobFile)	否，抛异常
JobConf	deleteLocalFiles()	否，抛异常
JobConf	deleteLocalFiles(String subdir)	否，抛异常
JobConf	normalizeMemoryConfigValue(long val)	空实现
JobConf	setCombinerClass(Class<? extends Reducer> theClass)	是
JobConf	setCompressMapOutput(boolean compress)	空实现
JobConf	setInputFormat(Class<? extends InputFormat> theClass)	否，抛异常
JobConf	setJar(String jar)	否，抛异常
JobConf	setJarByClass(Class cls)	否，抛异常
JobConf	setJobEndNotificationURI(String uri)	否，抛异常
JobConf	setJobName(String name)	空实现
JobConf	setJobPriority(JobPriority prio)	否，抛异常
JobConf	setKeepFailedTaskFiles(boolean keep)	否，抛异常
JobConf	setKeepTaskFilesPattern(String pattern)	否，抛异常

类型	接口	是否兼容
JobConf	setKeyFieldComparatorOptions(String keySpec)	否，抛异常
JobConf	setKeyFieldPartitionerOptions(String keySpec)	否，抛异常
JobConf	setMapDebugScript(String mDbgScript)	空实现
JobConf	setMapOutputCompressorClass(Class<? extends CompressionCodec> codecClass)	空实现
JobConf	setMapOutputKeyClass(Class<?> theClass)	是
JobConf	setMapOutputValueClass(Class<?> theClass)	是
JobConf	setMapperClass(Class<? extends Mapper> theClass)	是
JobConf	setMapRunnerClass(Class<? extends MapRunnable> theClass)	否，抛异常
JobConf	setMapSpeculativeExecution(boolean speculativeExecution)	空实现
JobConf	setMaxMapAttempts(int n)	空实现
JobConf	setMaxMapTaskFailuresPercent(int percent)	空实现
JobConf	setMaxPhysicalMemoryForTask(long mem)	空实现
JobConf	setMaxReduceAttempts(int n)	空实现
JobConf	setMaxReduceTaskFailuresPercent(int percent)	空实现
JobConf	setMaxTaskFailuresPerTracker(int noFailures)	空实现
JobConf	setMaxVirtualMemoryForTask(long vmem)	空实现
JobConf	setMemoryForMapTask(long mem)	是
JobConf	setMemoryForReduceTask(long mem)	是
JobConf	setNumMapTasks(int n)	是

类型	接口	是否兼容
JobConf	setNumReduceTasks(int n)	是
JobConf	setNumTasksToExecutePerJvm(int numTasks)	空实现
JobConf	setOutputCommitter(Class<? extends OutputCommitter> theClass)	否，抛异常
JobConf	setOutputFormat(Class<? extends OutputFormat> theClass)	空实现
JobConf	setOutputKeyClass(Class<?> theClass)	是
JobConf	setOutputKeyComparatorClass(Class <? extends RawComparator> theClass)	否，抛异常
JobConf	setOutputValueClass(Class<?> theClass)	是
JobConf	setOutputValueGroupingComparator(Class<? extends RawComparator> theClass)	否，抛异常
JobConf	setPartitionerClass(Class<? extends Partitioner> theClass)	是
JobConf	setProfileEnabled(boolean newValue)	空实现
JobConf	setProfileParams(String value)	空实现
JobConf	setProfileTaskRange(boolean isMap, String newValue)	空实现
JobConf	setQueueName(String queueName)	否，抛异常
JobConf	setReduceDebugScript(String rDbgScript)	空实现
JobConf	setReducerClass(Class<? extends Reducer> theClass)	是
JobConf	setReduceSpeculativeExecution(boolean speculativeExecution)	空实现
JobConf	setSessionId(String sessionId)	空实现
JobConf	setSpeculativeExecution(boolean speculativeExecution)	否，抛异常
JobConf	setUseNewMapper(boolean flag)	是
JobConf	setUseNewReducer(boolean flag)	是
JobConf	setUser(String user)	空实现

类型	接口	是否兼容
JobConf	setWorkingDirectory(Path dir)	空实现
FileInputFormat	不涉及	否，抛异常
TextInputFormat	不涉及	是
InputSplit	mapred.min.split.size.	否，抛异常
FileSplit	map.input.file	否，抛异常
RecordWriter	不涉及	否，抛异常
RecordReader	不涉及	否，抛异常
OutputFormat	不涉及	否，抛异常
OutputCommitter	abortJob(JobContext jobContext, int status)	否，抛异常
OutputCommitter	abortJob(JobContext context, JobStatus.State runState)	否，抛异常
OutputCommitter	abortTask(TaskAttemptContext taskContext)	否，抛异常
OutputCommitter	abortTask(TaskAttemptContext taskContext)	否，抛异常
OutputCommitter	cleanupJob(JobContext jobContext)	否，抛异常
OutputCommitter	cleanupJob(JobContext context)	否，抛异常
OutputCommitter	commitJob(JobContext jobContext)	否，抛异常
OutputCommitter	commitJob(JobContext context)	否，抛异常
OutputCommitter	commitTask(TaskAttemptContext taskContext)	否，抛异常
OutputCommitter	needsTaskCommit(TaskAttemptContext taskContext)	否，抛异常
OutputCommitter	needsTaskCommit(TaskAttemptContext taskContext)	否，抛异常
OutputCommitter	setupJob(JobContext jobContext)	否，抛异常
OutputCommitter	setupJob(JobContext jobContext)	否，抛异常
OutputCommitter	setupTask(TaskAttemptContext taskContext)	否，抛异常
OutputCommitter	setupTask(TaskAttemptContext taskContext)	否，抛异常
Counter	getDisplayName()	是

类型	接口	是否兼容
Counter	getName()	是
Counter	getValue()	是
Counter	increment(long incr)	是
Counter	setValue(long value)	是
Counter	setDisplayName(String displayName)	是
DistributedCache	CACHE_ARCHIVES	否，抛异常
DistributedCache	CACHE_ARCHIVES_SIZES	否，抛异常
DistributedCache	CACHE_ARCHIVES_TIMESTAMPS	否，抛异常
DistributedCache	CACHE_FILES	否，抛异常
DistributedCache	CACHE_FILES_SIZES	否，抛异常
DistributedCache	CACHE_FILES_TIMESTAMPS	否，抛异常
DistributedCache	CACHE_LOCALARCHIVES	否，抛异常
DistributedCache	CACHE_LOCALFILES	否，抛异常
DistributedCache	CACHE_SYMLINK	否，抛异常
DistributedCache	addArchiveToClassPath(Path archive, Configuration conf)	否，抛异常
DistributedCache	addArchiveToClassPath(Path archive, Configuration conf, FileSystem fs)	否，抛异常
DistributedCache	addCacheArchive(URI uri, Configuration conf)	否，抛异常
DistributedCache	addCacheFile(URI uri, Configuration conf)	否，抛异常
DistributedCache	addFileToClassPath(Path file, Configuration conf)	否，抛异常
DistributedCache	addFileToClassPath(Path file, Configuration conf, FileSystem fs)	否，抛异常
DistributedCache	addLocalArchives(Configuration conf, String str)	否，抛异常
DistributedCache	addLocalFiles(Configuration conf, String str)	否，抛异常
DistributedCache	checkURIs(URI[] uriFiles, URI[] uriArchives)	否，抛异常

类型	接口	是否兼容
DistributedCache	createAllSymlink(Configuration conf, File jobCacheDir, File workDir)	否，抛异常
DistributedCache	createSymlink(Configuration conf)	否，抛异常
DistributedCache	getArchiveClassPaths(Configuration conf)	否，抛异常
DistributedCache	getArchiveTimestamps(Configuration conf)	否，抛异常
DistributedCache	getCacheArchives(Configuration conf)	否，抛异常
DistributedCache	getCacheFiles(Configuration conf)	否，抛异常
DistributedCache	getFileClassPaths(Configuration conf)	否，抛异常
DistributedCache	getFileStatus(Configuration conf, URI cache)	否，抛异常
DistributedCache	getFileTimestamps(Configuration conf)	否，抛异常
DistributedCache	getLocalCacheArchives(Configuration conf)	否，抛异常
DistributedCache	getLocalCacheFiles(Configuration conf)	否，抛异常
DistributedCache	getSymlink(Configuration conf)	否，抛异常
DistributedCache	getTimestamp(Configuration conf, URI cache)	否，抛异常
DistributedCache	setArchiveTimestamps(Configuration conf, String timestamps)	否，抛异常
DistributedCache	setCacheArchives(URI[] archives, Configuration conf)	否，抛异常
DistributedCache	setCacheFiles(URI[] files, Configuration conf)	否，抛异常
DistributedCache	setFileTimestamps(Configuration conf, String timestamps)	否，抛异常
DistributedCache	setLocalArchives(Configuration conf, String str)	否，抛异常
DistributedCache	setLocalFiles(Configuration conf, String str)	否，抛异常
IsolationRunner	不涉及	否，抛异常

类型	接口	是否兼容
Profiling	不涉及	空实现
Debugging	不涉及	空实现
Data Compression	不涉及	是
Skipping Bad Records	不涉及	否，抛异常
Job Authorization	mapred.acls.enabled	否，抛异常
Job Authorization	mapreduce.job.acl-view-job	否，抛异常
Job Authorization	mapreduce.job.acl-modify-job	否，抛异常
Job Authorization	mapreduce.cluster.administrators	否，抛异常
Job Authorization	mapred.queue.queue-name.acl-administer-jobs	否，抛异常
MultipleInputs	不涉及	否，抛异常
Multi{anchor:_GoBack}pleOutputs	不涉及	是
org.apache.hadoop.mapreduce.lib.db	不涉及	否，抛异常
org.apache.hadoop.mapreduce.security	不涉及	否，抛异常
org.apache.hadoop.mapreduce.lib.jobcontrol	不涉及	否，抛异常
org.apache.hadoop.mapreduce.lib.chain	不涉及	否，抛异常
org.apache.hadoop.mapreduce.lib.db	不涉及	否，抛异常

6 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，例如MR Main则不受此限制），具体限制如下所示。

- 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。

在代码中，直接使用下文访问本地文件的方法会报access denied异常。

- 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, DomainCombiner
 combiner)
public DomainCombiner getDomainCombiner()
```

7 外部表

7.1 外部表概述

MaxCompute作为阿里云大数据平台的核心计算组件，拥有强大的计算能力。MaxCompute能够调度大量的节点做并行计算，同时对分布式计算中的failover、重试等功能都有体系化的处理管理机制。

外部表产生背景

MaxCompute SQL作为分布式数据处理的主要入口，可实现EB级别的离线数据的快速处理和存储。随着大数据业务的不断扩展，新的数据使用场景在不断产生，MaxCompute计算框架也在不断的演化。MaxCompute原来主要面对内部特殊格式数据的强大计算能力，正一步步的开放给不同的外部数据。

现阶段MaxCompute SQL处理的主要是以cfile列格式存储在内部MaxCompute表格中的结构化数据。对于 MaxCompute表外的各种用户数据（包括文本以及各种非结构化的数据），您需要通过不同工具导入MaxCompute表再进行计算。以OSS为例，如果您需要在MaxCompute中处理OSS上的数据，通常有以下两种方式：

- 通过OSS SDK或者其他工具从OSS下载数据，然后再通过MaxCompute Tunnel将数据导入表里。
- 编写UDF，在UDF里直接调用OSS SDK访问OSS数据。

但这两种做法都有不足之处：

- 第一种需要在MaxCompute系统外部做一次中转，如果OSS数据量太大，还需要考虑如何并发来加速，无法充分利用 MaxCompute大规模计算的能力。
- 第二种通常需要申请UDF网络访问权限，还需要开发者自己控制作业并发数，解决数据如何分片的问题。

本节将介绍的外部表的功能，旨在提供除MaxCompute内部表格以外的其他数据的处理能力。在这个框架中，通过一条简单的DDL语句，即可在MaxCompute上创建一张外部表，建立MaxCompute表与外部数据源的关联，提供各种数据的接入和输出能力。创建好的外部表可以像普通的MaxCompute表一样使用（大部分场景），充分利用MaxCompute SQL的强大计算功能。

说明：

- 使用外部表功能，外部表的数据不会复制一份放在MaxCompute并带来存储费用。
- 外部表当前仅支持全量搜索，不支持列剪裁。
- Tunnel功能及Tunnel SDK当前不支持外部表操作。您可以通过Tunnel直接上传数据到MaxCompute内部表，或者是通过OSS Python SDK上传到OSS后，在MaxCompute使用外部表做映射。

外部表示例

您可以通过以下示例，深入了解通过MaxCompute外部表功能处理各种非结构化数据的方法：

- 访问OSS和TableStore（OTS）非结构化数据，请参见 访问OSS非结构化数据和访问OTS非结构化数据。
- 外部表访问OSS的账号，在RAM中自定义授权MaxCompute访问OSS的权限，请参见OSS的STS模式授权。
- MaxCompute的非结构化框架支持通过INSERT方式将MaxCompute的数据直接输出到OSS，请参见输出到OSS的非结构化数据。
- 处理各种开源格式数据，请参见处理OSS的开源格式数据。

7.2 OSS的STS模式授权

本文向您介绍如何在RAM中自定义授权MaxCompute访问OSS的权限。

您可以通过DataWorks配合MaxCompute对外部表进行可视化的创建、搜索、查询、配置、加工和分析。详情请参见外部表。

创建外部表时Location访问OSS的账号支持传入明文`AccessKeyId`和`AccessKeySecret`，但这样做有泄露账号的风险。在某些场景下，这种风险是不可容忍的，因此MaxCompute提供了更为安全的方式来访问OSS。

MaxCompute结合了阿里云的访问控制服务（RAM）和令牌服务（STS）来解决账号的安全问题。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，可以直接在RAM控制台进行一键授权操作。
- 自定义授权。

  1. 首先需要在RAM中授权MaxCompute访问OSS的权限。创建角色，角色名如`AliyunODPSDefaultRole`或`AliyunODPSRoleForOtherUser`，并将策略内容设置为：

```
--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"
}
```

2. 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
```

3. 再将权限`AliyunODPSRolePolicy`授权给该角色。

说明:

授权完成后，查看角色详情获取Role的Ran信息，后续创建OSS外部表时需指定这个Ran信息。

7.3 访问OSS非结构化数据

本文将为您介绍如何在MaxCompute上访问OSS的数据。

您可以通过DataWorks配合MaxCompute对外部表进行可视化的创建、搜索、查询、配置、加工和分析，详情请参见外部表。关于处理非结构化数据的原理性介绍，请参见外部表概述。

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，例如角色AliyunODPSDefaultRole、AliyunODPSRoleForOtherUser。

2. 修改角色策略内容设置。

```
--当MaxCompute和OSS的Owner是同一个账号，设置如下。
{
"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，设置如下。
```

```
{
"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。

```
{
"Version": "1",
"Statement": [
{
 "Action": [
   "oss:ListBuckets",
   "oss:GetObject",
   "oss:ListObjects",
   "oss:PutObject",
   "oss:DeleteObject",
   "oss:AbortMultipartUpload",
   "oss:ListParts"
 ],
 "Resource": "*",
 "Effect": "Allow"
}
]
}
--可自定义其它权限。
```

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，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

创建外部表语句示例如下。

```
CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_csv_external
(
vehicleId int,
recordId int,
patientId int,
calls int,
```

```
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
STORED BY 'com.aliyun.odps.CsvStorageHandler' -- (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)
```

示例说明：

- `com.aliyun.odps.CsvStorageHandler`**是内置的处理CSV格式文件的**`StorageHandler`**，它定义了如何读写CSV文件。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，以下为错误用法。**

    ```
    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。**

您可以执行如下语句，查看创建好的外部表结构信息。

```
desc extended <table_name>;
```

在返回的信息里，除了包含和内部表一样的基础信息，Extended Info还包含外部表StorageHandler 、Location等信息。

查询外部表

外部表创建成功后，您可以像使用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:

目前您只能通过MaxCompute SQL操作外部表，不能通过MaxCompute MapReduce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。输出结果如下所示。

```
+------------+------------+-----------+
| recordId   | patientId  | direction |
+------------+------------+-----------+
| 1          | 51         | S         |
| 3          | 48         | NE        |
| 4          | 30         | W         |
| 5          | 47         | S         |
| 7          | 53         | N         |
| 8          | 63         | SW        |
| 10         | 31         | N         |
+------------+------------+-----------+
```

读取gzip压缩的CSV/TSV数据

MaxCompute目前只支持通过内置Extractor读取OSS上gzip压缩的CSV/TSV数据。与读取非压缩文件相比，主要区别是SERDEPROPERTIES指定的属性。

说明:

如果OSS上的数据文件为归档文件，请登录OSS，对文件进行解冻。

创建外部表的代码示例如下。

```
CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_csv_external
```

```
(
vehicleId bigint,
recordId bigint,
patientId bigint,
calls bigint,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
STORED BY 'com.aliyun.odps.CsvStorageHandler'
WITH SERDEPROPERTIES (
 'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole'
 [,'odps.text.option.gzip.input.enabled'='true']
 [,'name3'='value3']
)
LOCATION 'oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/Demo/
SampleData/CSV/AmbulanceData/';
```

目前SERDEPROPERTIES还支持的属性项，如下所示。

属性名	属性值	默认值	说明
odps.text.option.gzip.input.enabled	True/False	False	打开/关闭读压缩。
odps.text.option.gzip.output.enabled	True/False	False	打开/关闭写压缩。
odps.text.option.header.lines.count	非负整数	0	跳过文本文件头N行。
odps.text.option.null.indicator	字符串	空字符串	在解析或者写出NULL值时代表NULL的字符串。
odps.text.option.ignore.empty.lines	True/False	True	是否忽略空行。
odps.text.option.encoding	UTF-8/UTF-16/US-ASCII	UTF-8	指定文本的字符编码。
odps.text.option.delimiter	单个字符	英文逗号（,）	指定文本的列分隔符。

说明:

当关联OSS压缩数据的外部表，读（Select）、写（Insert）操作也需要进行，那么创建这个外部表时，需要同时将odps.text.option.gzip.input.enabled和odps.text.option.gzip.output.enabled两个属性设置为True。

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。例如`/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是外部表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;
```

- 创建外部表

  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，其中delimeter是您自定义的分割方法名称。

```
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。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。在此示例中，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。您可以用同样的方法利用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：您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EXTERNAL TABLE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（例如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），则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。有效的路径存储layout如下。

```
osscmd ls oss://oss-odps-test/log_data/
2017-01-14 08:03:35 128MB Standard oss://oss-odps-test/log_data/year
=2016/month=06/day=01/logfile
2017-01-14 08:04:12 127MB Standard oss://oss-odps-test/log_data/year
=2016/month=06/day=01/logfile.1
2017-01-14 08:05:02 118MB Standard oss://oss-odps-test/log_data/year
=2016/month=06/day=02/logfile
2017-01-14 08:06:45 123MB Standard oss://oss-odps-test/log_data/year
=2016/month=07/day=10/logfile
2017-01-14 08:07:11 115MB Standard oss://oss-odps-test/log_data/year
=2016/month=08/day=08/logfile
...
```

说明:

因为数据是离线准备的，即通过osscmd或者其它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，分区的详情请参见分区。在数据准备好并且PARTITION信息引入MaxCompute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，则执行如下语句。

```
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），数据的layout如下。

```
osscmd ls oss://oss-odps-test/log_data_customized/
2017-01-14 08:03:35 128MB Standard oss://oss-odps-test/log_data_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可参看前面的示例，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时增加了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，您也可以正确地实现对子目录数据的分区访问了。

7.4 处理OSS的开源格式数据

本文将为您介绍对于存储在OSS上的各种流行的开源数据格式（ORC、PARQUET、SEQUENCEFILE、RCFILE、AVRO和TEXTFILE）如何通过非结构化框架在MaxCompute进行处理。

您可以通过DataWorks配合MaxCompute对外部表进行可视化的创建、搜索、查询、配置、加工和分析。详情请参见外部表。

访问OSS非结构化数据为您介绍了如何在MaxCompute上访问存储在OSS上的文本、音频、图像等格式的数据。非结构框架会直接调用开源社区的实现来进行开源数据格式的解析，并且与MaxCompute系统无缝对接。

说明:

处理OSS的开源格式数据前，需要首先对OSS进行STS模式授权。

创建External Table

MaxCompute非结构化数据框架通过External Table与各种数据的关联，关联OSS上开源格式数据的External Table建表的DDL语句格式如下所示。

```
DROP TABLE [IF EXISTS] <external_table>;
CREATE EXTERNAL TABLE [IF NOT EXISTS] <external_table>
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
[ROW FORMAT SERDE '<serde class>'
  [WITH SERDEPROPERTIES ('odps.properties.rolearn'='${roleran}' [,'
name2'='value2',...])]
]
STORED AS <file format>
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

说明:

该语法格式与Hive的语法相当接近，但需注意以下问题。

- STORED AS关键字，在该语法格式中不是普通非结构化外表用的STORED BY关键字，这是目前在读取开源兼容数据时独有的。

  STORED AS后面接的是文件格式名字，例如*ORC/PARQUET/RCFILE/SEQUENCEFILE/TEXTFILE*等。
- 外部表的column schemas必须与具体OSS上存储的数据的schema相符合。
- `ROW FORMAT SERDE`：非必选选项，只有在使用一些特殊的格式上，比如TEXTFILE时才需要使用。
- `WITH SERDEPROPERTIES`：当关联OSS权限使用STS模式授权时，需要该参数指定`odps.properties.rolearn`属性，属性值为RAM中具体使用的Role的Arn的信息。您可以在配置`STORED AS <file format>`的同时也通过`<serde class>`说明file format文件格式。以ORC文件格式为例，如下所示。

  ```
  CREATE EXTERNAL TABLE [IF NOT EXISTS] <external_table>
  (<column schemas>)
  [PARTITIONED BY (partition column schemas)]
  ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
  WITH SERDEPROPERTIES ('odps.properties.rolearn'='${roleran}'
  STORED AS ORC
  LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/'
  ```

  不同`file format`对应的`serde class`如下：

  - CFILE: com.aliyun.apsara.serde.CFileSerDe
  - ALIORC: com.aliyun.apsara.serde.AliOrcSerDe
  - SEQUENCEFILE: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
  - TEXTFILE: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
  - RCFILE: org.apache.hadoop.hive.serde2.columnar.LazyBinaryColumnarSerDe
  - ORC: org.apache.hadoop.hive.ql.io.orc.OrcSerde
  - ORCFILE: org.apache.hadoop.hive.ql.io.orc.OrcSerde
  - PARQUET: org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe
  - AVRO: org.apache.hadoop.hive.serde2.avro.AvroSerDe
- 如果不使用STS模式授权，则无需指定`odps.properties.rolearn`属性，直接在Location传入明文`AccessKeyId`和`AccessKeySecret`。
- Location如果关联OSS，需使用明文AK，写法如下所示。

  ```
  LOCATION 'oss://${accessKeyId}:${accessKeySecret}@${endpoint}/${
  bucket}/${userPath}/'
  ```
- 访问OSS外部表，目前不支持使用外网Endpoint。
- 目前STORE AS单个文件大小不能超过3G，如果文件过大，建议split拆分。

关联OSS的PARQUET数据

假设有一些PARQUET文件存放在一个OSS路径上，每个文件都是PARQUET格式，存放的schema为16列（4列Bigint、4列Double和8列String）的数据，建表DDL语句示例如下所示。

```
CREATE EXTERNAL TABLE tpch_lineitem_parquet
(
  l_orderkey bigint,
  l_partkey bigint,
  l_suppkey bigint,
  l_linenumber bigint,
  l_quantity double,
  l_extendedprice double,
  l_discount double,
  l_tax double,
  l_returnflag string,
  l_linestatus string,
  l_shipdate string,
  l_commitdate string,
  l_receiptdate string,
  l_shipinstruct string,
  l_shipmode string,
  l_comment string
)
STORED AS PARQUET
LOCATION 'oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.
aliyuncs.com/bucket/parquet_data/';
```

默认PARQUET数据不压缩，如果您需要在MaxCompute上压缩PARQUET数据，需要`set odps.sql.hive.compatible=true;`。支持压缩类型包括：SNAPPY, GZIP。

关联OSS的TEXT数据

如果数据是每行以JSON格式，存储成OSS上TEXTFILE文件，同时数据在OSS通过多个目录组织，这时可以使用MaxCompute分区表和数据关联，建分区表DDL语句示例如下所示。

```
CREATE EXTERNAL TABLE tpch_lineitem_textfile
(
  l_orderkey bigint,
  l_partkey bigint,
  l_suppkey bigint,
  l_linenumber bigint,
  l_quantity double,
  l_extendedprice double,
  l_discount double,
  l_tax double,
  l_returnflag string,
  l_linestatus string,
  l_shipdate string,
  l_commitdate string,
  l_receiptdate string,
  l_shipinstruct string,
  l_shipmode string,
  l_comment string
)
PARTITIONED BY (ds string)
```

```
ROW FORMAT serde 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS TEXTFILE
LOCATION 'oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.
aliyuncs.com/bucket/text_data/';
```

如果OSS表目录下面的子目录是以Partition Name方式组织，示例如下。

```
oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.
com/bucket/text_data/ds=20170102/'
oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.
com/bucket/text_data/ds=20170103/'
...
```

则可以使用以下DDL语句ADD PARTITION。

```
ALTER TABLE tpch_lineitem_textfile ADD PARTITION(ds="20170102");
ALTER TABLE tpch_lineitem_textfile ADD PARTITION(ds="20170103");
```

如果OSS分区目录不是按这种方式组织，或者根本不在表目录下，示例如下。

```
oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.
com/bucket/text_data_20170102/;
oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.
com/bucket/text_data_20170103/;
...
```

这种情况下，可以使用以下DDL语句ADD PARTITION。

```
ALTER TABLE tpch_lineitem_textfile ADD PARTITION(ds="20170102")
LOCATION 'oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.
aliyuncs.com/bucket/text_data_20170102/';
ALTER TABLE tpch_lineitem_textfile ADD PARTITION(ds="20170103")
LOCATION 'oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.
aliyuncs.com/bucket/text_data_20170103/';
...
```

TEXT数据支持SERDEPROPERTIES（key：默认值）

```
Fields terminator: '\001'
Escape delimitor: '\\'
Collection items terminator: '\002'
Map keys terminator: '\003'
Lines terminator: '\n'
Null defination: '\\N'
```

关联OSS的CSV数据

DDL语句格式如下所示。

```
CREATE EXTERNAL TABLE [IF NOT EXISTS]
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
  WITH SERDEPROPERTIES
    ('separatorChar'=',', 'quoteChar'='"', 'escapeChar'='\\')
STORED AS TEXTFILE
```

```
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

从上述语句可以看出，CSV数据DDL语句支持SERDEPROPERTIES（key：默认值）

```
separatorChar: ','
quoteChar: '"'
escapeChar: '\'
```

说明:

hive OpenCSVSerde只支持string类型。

OpenCSVSerde当前不属于Builtin Serde，DML语句执行时您需要设置set odps.sql.hive.compatible=true;

关联OSS的JSON数据

DDL语句格式如下所示，支持SERDEPROPERTIES。

```
CREATE EXTERNAL TABLE [IF NOT EXISTS]
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS TEXTFILE
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

关联OSS的ORC数据

DDL语句格式如下所示。

```
CREATE EXTERNAL TABLE [IF NOT EXISTS]
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
STORED AS ORC
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

关联OSS的AVRO数据

DDL语句格式如下所示。

```
CREATE EXTERNAL TABLE [IF NOT EXISTS]
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
STORED AS AVRO
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

关联OSS的SEQUENCEFILE数据

```
CREATE EXTERNAL TABLE [IF NOT EXISTS]
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
STORED AS SEQUENCEFILE
```

```
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

读取以及处理OSS的开源格式数据

对比前文的两个创建外部表示例，可以看出对于不同文件类型，只要简单修改STORED AS后的格式名。在下述示例中将只集中描述对上面PARQUET数据对应的外表（tpch_lineitem_parquet）的处理。如果要处理不同的文件类型，只要在DDL创建外表时指定是PARQUET/ORC/TEXTFILE/RCFILE/TEXTFILE即可，处理数据的语句一样。

- 直接读取以及处理OSS的开源数据

  创建数据外表进行关联后，直接对外表就可以进行与普通MaxCompute表一样的操作，如下所示。

```
SELECT l_returnflag, l_linestatus,
SUM(l_extendedprice*(1-l_discount)) AS sum_disc_price,
AVG(l_quantity) AS avg_qty,
COUNT(*) AS count_order
FROM tpch_lineitem_parquet
WHERE l_shipdate <= '1998-09-02'
GROUP BY l_returnflag, l_linestatus;
```

  外表tpch_lineitem_parquet被当作一个普通的内部表一样使用，不同之处在于，MaxCompute内部的计算引擎是直接从OSS读取对应的PARQUET数据进行处理的。

  前文创建的关联textfile的外部分区表tpch_lineitem_textfile，因为使用了`ROW FORMAT + STORED AS`，需要手动设置flag `set odps.sql.hive.compatible=true;`（只使用STORED AS，odps.sql.hive.compatible默认为FALSE）再创建外部表或读取数据，否则会有报错。

```
SELECT * FROM tpch_lineitem_textfile LIMIT 1;
FAILED: ODPS-0123131:User defined function exception - Traceback:
com.aliyun.odps.udf.UDFException: java.lang.ClassNotFoundException:
com.aliyun.odps.hive.wrapper.HiveStorageHandlerWrapper
--需要手动设置hive兼容flag
set odps.sql.hive.compatible=true;
SELECT * FROM tpch_lineitem_textfile LIMIT 1;
+------------+------------+------------+-------------+------------
+-----------------+------------+------------+-------------
+--------------+------------+-------------+--------------
+----------------+------------+-----------+
| l_orderkey | l_partkey  | l_suppkey  | l_linenumber | l_quantity
 | l_extendedprice | l_discount | l_tax      | l_returnflag
 | l_linestatus | l_shipdate | l_commitdate | l_receiptdate |
l_shipinstruct | l_shipmode | l_comment |
+------------+------------+------------+-------------+------------
+-----------------+------------+------------+-------------
+--------------+------------+-------------+--------------
+----------------+------------+-----------+
| 5640000001 | 174458698  | 9458733    | 1            | 14.0
  | 23071.58        | 0.08       | 0.06       | N            | O
          | 1998-01-26 | 1997-11-16   | 1998-02-18    | TAKE BACK
RETURN | SHIP       | cuses nag silently. quick |
```

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

说明:

直接使用外表，每次读取数据都需要涉及外部OSS的I/O操作，且MaxCompute系统本身针对内部存储做的许多高性能优化都用不上，如此一来性能上就会有所损失。 因此如果是需要对数据进行反复计算以及对计算的高效性比较敏感的场景，推荐使用下文介绍的用法：先将数据导入MaxCompute内部再进行计算。

SQL（create、select、insert等操作）中涉及到这几个复杂数据类型，需在SQL语句前加语句`set odps.sql.type.system.odps2=true;`，执行时set语句和SQL语句一起提交执行。详情请参见数据类型。

- **将*OSS*的开源数据导入*MaxCompute*再进行计算**

  首先创建一个与外部表schema一样的内部表tpch_lineitem_internal，然后将OSS上的开源数据导入MaxCompute内部表，以MaxCompute内部数据存储格式进行存储。

```
CREATE TABLE tpch_lineitem_internal LIKE tpch_lineitem_parquet;
INSERT OVERWRITE TABLE tpch_lineitem_internal;
SELECT * FROM tpch_lineitem_parquet;
```

  接下来直接对内部表进行同样的操作。

```
SELECT l_returnflag, l_linestatus,
SUM(l_extendedprice*(1-l_discount)) AS sum_disc_price,
AVG(l_quantity) AS avg_qty,
COUNT(*) AS count_order
FROM tpch_lineitem_internal
WHERE l_shipdate <= '1998-09-02'
GROUP BY l_returnflag, l_linestatus;
```

  通过此方式将数据先导入MaxCompute系统进行存储，对同样数据进行计算处理会更高效。

处理OSS数据常见问题

作业报错：`Inline data exceeds the maximun allowed size`。

问题原因：OSS Store对于每一个小文件有一个大小限制，如果超过3GB则报错。

处理方法：针对该问题，您可以通过调整以下两个flag值进行处理。其原理是通过flag调整执行计划，控制每个reducer写入外部表OSS的数据大小，使得OSS Store文件不超过3GB的限制。

```
set odps.sql.mapper.split.size=256; #调整每个mapper读取table数据的大小，单位是MB。
```

```
set odps.sql.reducer.instances=100; #调整执行计划的reducer数量。
```

7.5 输出到OSS的非结构化数据

MaxCompute的非结构化框架支持通过insert方式将MaxCompute的数据直接输出到OSS，MaxCompute也是通过外部表关联OSS，进行数据输出。

您可以通过DataWorks配合MaxCompute对外部表进行可视化的创建、搜索、查询、配置、加工和分析。详情请参见外部表。

访问OSS非结构化数据为您介绍了MaxCompute如何通过外部表的关联进行访问并处理存储在OSS的非结构化数据。

输出数据到OSS通常是两种情况：

- MaxCompute内部表输出到关联OSS的外部表。
- MaxCompute处理外部表后结果直接输出到关联OSS的外部表。

与访问OSS数据一样，MaxCompute支持通过内置StorageHandler 和自定义StorageHandler进行输出。

通过内置StorageHandler输出到OSS

使用MaxCompute内置的StorageHandler，可以非常方便的按照约定格式输出数据到OSS进行存储。我们只需要创建一个外部表，指明内置的StorageHandler，就能以这张表为关联，相关逻辑由系统实现。

目前MaxCompute支持2个内置StorageHandler：

- `com.aliyun.odps.CsvStorageHandler`定义如何读写csv格式数据，数据格式约定：英文逗号,为列分隔符，换行符为`\n`。
- `com.aliyun.odps.TsvStorageHandler`定义如何读写csv格式数据，数据格式约定：`\t`为列分隔符，换行符为`\n`。
- 创建EXTERNAL TABLE

```
CREATE EXTERNAL TABLE [IF NOT EXISTS] <external_table>
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
STORED BY '<StorageHandler>'
[WITH SERDEPROPERTIES (  'odps.properties.rolearn'='${roleran}') ]
```

```
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

- STORED BY，如果需求输出到OSS上的数据文件是TSV文件，则用内置`com.aliyun.odps.TsvStorageHandler`；如果需求输出到OSS上的数据文件是CSV文件，则用内置`com.aliyun.odps.CsvStorageHandler`。
- WITH SERDEPROPERTIES，当关联OSS权限使用“STS模式授权”的“自定义授权”时，需要该参数指定’odps.properties.rolearn’属性，属性值为RAM 中具体使用的自定义role的Arn的信息。

  说明:
  关于STS模式授权的介绍，请参见访问OSS非结构化数据。
- LOCATION，指定对应OSS存储的文件路径。若WITH SERDEPROPERTIES中不设置`odps.properties.rolearn`属性，且授权方式是采用明文AK，则LOCATION为：

  ```
  LOCATION 'oss://${accessKeyId}:${accessKeySecret}@${endpoint}/${
  bucket}/${userPath}/'
  ```

· 通过对External Table的 INSERT 操作实现数据输出到OSS。

说明:
insert到OSS的单个文件的大小不能超过5G。

通过External Table关联上OSS存储路径后，可以对External Table做标准的SQL INSERT OVERWRITE/INSERT INTO操作既可将数据输出到OSS。

```
INSERT OVERWRITE|INTO TABLE <external_tablename> [PARTITION (
partcol1=val1, partcol2=val2 ...)]
select_statement
FROM <from_tablename>
[WHERE where_condition];
```

- *from_tablename*：可以是内部表，也可以是外部表（包括关联的OSS或OTS的外部表）。
- INSERT将按照外部表‘STORED BY’指定 ‘StorageHandler’的格式（即TSV或CSV）写到OSS上。

INSERT 操作成功完成后，就可以看到OSS上的对应LOCATION产生了一系列文件。

如：external table 对应的location是*oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/tsv_output_folder/* 则，在OSS对应路径中可以看到生成一系列文件：

```
osscmd ls oss://oss-odps-test/tsv_output_folder/
2017-01-14 06:48:27 39.00B Standard oss://oss-odps-test/tsv_output
_folder/.odps/.meta
2017-01-14 06:48:12 4.80MB Standard oss://oss-odps-test/tsv_output
_folder/.odps/20170113224724561g9m6csz7/M1_0_0-0.tsv
```

```
2017-01-14 06:48:05 4.78MB Standard oss://oss-odps-test/tsv_output
_folder/.odps/20170113224724561g9m6csz7/M1_1_0-0.tsv
2017-01-14 06:47:48 4.79MB Standard oss://oss-odps-test/tsv_output
_folder/.odps/20170113224724561g9m6csz7/M1_2_0-0.tsv
...
```

可以看到，通过前面LOCATION指定的oss-odps-test这个OSS bucket下的tsv_output_folder文件夹下产生了一个’.odps’文件夹，这其中将有一些’.tsv’文件，以及一个’.meta’文件。 类似的文件结构是MaxCompute往OSS上输出所特有的：

- 通过MaxCompute对一个OSS地址，使用INSERT INTO/OVERWRITE 外部表来做输出操作，所有的数据将在指定的LOCATION下的’.odps’文件夹产生。
- ‘.odps’文件夹中的’.meta’文件为MaxCompute额外写出的宏数据文件，其中用于记录当前文件夹中有效的数据。 正常情况下，如果INSERT操作成功完成的话，可以认为当前文件夹的所有数据均是有效数据。 只有在有作业失败的情况下需要对这个宏数据进行解析。即使是在作业中途失败或被kill的情况下，对于INSERT OVERWRITE操作，再跑一次成功即可。
- 如果是分区表，将在tsv_output_folder文件夹下根据insert语句指定的分区值生成对应的分区子目录然后分区子目录里才是’.odps’文件夹。如`test/tsv_output_folder/一级分区名=分区值/n级分区名=分区值/.odps/20170113224724561g9m6csz7/M1_2_0-0.tsv`。

对于MaxCompute内置的TSV/CSV StorageHandler 处理来说，产生的文件数目与对应SQL stage的并发度是相同。

若`INSER OVERWITE ... SELECT ... FROM ...;`的操作在源数据表(from_tablename)上分配了1000个mapper，那么最后将产生了1000个TSV/CSV文件。

通过自定义StorageHandler输出到OSS

除了使用内置的StorageHandler来实现在OSS上输出TSV/CSV常见文本格式，MaxCompute非结构化框架提供了通用的SDK，支持对外输出自定义数据格式文件。

与内置StorageHandler一样需要先创建EXTERNAL TABLE，再通过对External Table的insert操作实现数据输出到OSS。不同点在于创建外部表时，STORED BY是需要指定自定义的StorageHandler。

说明:

MaxCompute非结构化框架通过StorageHandler这个接口来描述对各种数据存储格式的处理。具体来说，StorageHandler作为一个wrapper class, 允许您指定自定义的Exatractor（用于数据的读入，解析，处理等）和Outputer（用于数据的

处理和输出等）。自定义的StorageHandler应该继承OdpsStorageHandler，实现getExtractorClass和getOutputerClass接口。

下面引用访问OSS非结构化数据中自定义Extractor访问OSS的TextStorageHandler例子，为您介绍MaxCompute如何通过自定义StorageHandler将数据输出到OSS的txt文件，且以‘|’为列分隔符，以‘\n’为换行符。

说明:

MaxCompute Studio配置好 MaxCompute Java Module后，可以在examples中看到对应的示例代码。或者点击此处也看到完整代码。

- 定义Outputer

  输出逻辑都必须实现Outputer接口

```
package com.aliyun.odps.examples.unstructured.text;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.io.OutputStreamSet;
import com.aliyun.odps.io.SinkOutputStream;
import com.aliyun.odps.udf.DataAttributes;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.Outputer;
import java.io.IOException;
public class TextOutputer extends Outputer {
    private SinkOutputStream outputStream;
    private DataAttributes attributes;
    private String delimiter;
    public TextOutputer (){
        // default delimiter, this can be overwritten if a delimiter
 is provided through the attributes.
        this.delimiter = "|";
    }
    @Override
    public void output(Record record) throws IOException {
        this.outputStream.write(recordToString(record).getBytes());
    }
    // no particular usage of execution context in this example
    @Override
    public void setup(ExecutionContext ctx, OutputStreamSet
outputStreamSet, DataAttributes attributes) throws IOException {
        this.outputStream = outputStreamSet.next();
        this.attributes = attributes;
    }
    @Override
    public void close() {
        // no-op
    }
    private String recordToString(Record record){
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < record.getColumnCount(); i++)
        {
            if (null == record.get(i)){
                sb.append("NULL");
            }
            else{
                sb.append(record.get(i).toString());
```

```
            }
            if (i != record.getColumnCount() - 1){
                sb.append(this.delimiter);
            }
        }
        sb.append("\n");
        return sb.toString();
    }
}
```

Outputer接口有三个：setup、output和close, 这和Extractor的setup、extract和close三个接口基本上对称。 其中setup()和close()在一个outputer中只会调用一次。 可以在setup里面做初始化准备工作，另外通常需要把setup()传递进来的这三个参数保存成ouputer的class variable, 方便之后output()或者close()接口中使用。 而close()这个接口用于代码的扫尾工作。

通常情况下大部分的数据处理发生在output(Record)这个接口内。MaxCompute系统根据当前outputer分配处理的每个输入Record调用一次 output(Record)。假设在一个output(Record) 调用返回的时候，代码已经消费完这个Record, 因此在当前output(Record)返回后，系统会将Record所使用的内存作它用，所以当Record中的信息在跨多个output()函数调用被使用时，需要调用当前处理的record.clone()方法，将当前record保存下来。

- 定义Extractor

  Exatractor用于数据的读入，解析，处理等，如果输出的表最终不需要再通过MaxCompute进行读取等，可以无需定义。

```
package com.aliyun.odps.examples.unstructured.text;
import com.aliyun.odps.Column;
import com.aliyun.odps.data.ArrayRecord;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.io.InputStreamSet;
import com.aliyun.odps.udf.DataAttributes;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.Extractor;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
/**
 * 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;
        this.attributes = attributes;
        // check if "delimiter" attribute is supplied via SQL query
        String columnDelimiter = this.attributes.getValueByKey("
delimiter");
        if ( columnDelimiter != null)
        {
            this.columnDelimiter = columnDelimiter;
        }
        System.out.println("TextExtractor using delimiter [" + this.
columnDelimiter + "].");
        // note: more properties can be inited from attributes if
needed
    }
    @Override
    public Record extract() throws IOException {
        String line = readNextLine();
        if (line == null) {
            return null;
        }
        return textLineToRecord(line);
    }
    @Override
    public void close(){
        // no-op
    }
    private Record textLineToRecord(String line) throws IllegalArg
umentException
    {
        Column[] outputColumns = this.attributes.getRecordColumns();
        ArrayRecord record = new ArrayRecord(outputColumns);
        if (this.attributes.getRecordColumns().length != 0){
            // string copies are needed, not the most efficient one
, but suffice as an example here
            String[] parts = line.split(columnDelimiter);
            int[] outputIndexes = this.attributes.getNeededIndexes
();
            if (outputIndexes == null){
                throw new IllegalArgumentException("No outputIndexes
 supplied.");
            }
            if (outputIndexes.length != outputColumns.length){
                throw new IllegalArgumentException("Mismatched
output schema: Expecting "
                        + outputColumns.length + " columns but get "
 + parts.length);
            }
            int index = 0;
            for(int i = 0; i < parts.length; i++){
                // only parse data in columns indexed by output
indexes
                if (index < outputIndexes.length && i == outputInde
xes[index]){
                    switch (outputColumns[index].getType()) {
                        case STRING:
                            record.setString(index, parts[i]);
                            break;
                        case BIGINT:
                            record.setBigint(index, Long.parseLong(
parts[i]));
                            break;
```

```
                        case BOOLEAN:
                            record.setBoolean(index, Boolean.
parseBoolean(parts[i]));
                            break;
                        case DOUBLE:
                            record.setDouble(index, Double.
parseDouble(parts[i]));
                            break;
                        case DATETIME:
                        case DECIMAL:
                        case ARRAY:
                        case MAP:
                        default:
                            throw new IllegalArgumentException("Type
 " + outputColumns[index].getType() + " not supported for now.");
                    }
                    index++;
                }
            }
        }
        return record;
    }
    /**
     * Read next line from underlying input streams.
     * @return The next line as String object. If all of the
contents of input
     * streams has been read, return null.
     */
    private String readNextLine() throws IOException {
        if (firstRead) {
            firstRead = false;
            // the first read, initialize things
            currentReader = moveToNextStream();
            if (currentReader == null) {
                // empty input stream set
                return null;
            }
        }
        while (currentReader != null) {
            String line = currentReader.readLine();
            if (line != null) {
                return line;
            }
            currentReader = moveToNextStream();
        }
        return null;
    }
    private BufferedReader moveToNextStream() throws IOException {
        InputStream stream = inputs.next();
        if (stream == null) {
            return null;
        } else {
            return new BufferedReader(new InputStreamReader(stream
));
        }
    }
}
```

详情请参见访问OSS非结构化数据文档。

- **定义StorageHandler**

```
package com.aliyun.odps.examples.unstructured.text;
```

```
import com.aliyun.odps.udf.Extractor;
import com.aliyun.odps.udf.OdpsStorageHandler;
import com.aliyun.odps.udf.Outputer;
public class TextStorageHandler extends OdpsStorageHandler {
    @Override
    public Class<? extends Extractor> getExtractorClass() {
        return TextExtractor.class;
    }
    @Override
    public Class<? extends Outputer> getOutputerClass() {
        return TextOutputer.class;
    }
}
```

若表无需读取可不用指定Extractor接口。

- 编译打包

  将自定义代码编译打包，并作为jar资源上传到MaxCompute。如打的jar包名为*odps-TextStorageHandler.jar*，则上传为MaxCompute Resource的示例如下。

```
add jar odps-TextStorageHandler.jar;
```

- 创建external表

  与使用内置StorageHandler类似，需要建立一个外部表，不同的是这次需要指定数据输出到外部表时候，使用自定义的StorageHandler。

```
CREATE EXTERNAL TABLE IF NOT EXISTS output_data_txt_external
(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
STORED BY 'com.aliyun.odps.examples.unstructured.text.TextStorag
eHandler'
WITH SERDEPROPERTIES(
    'delimiter'='|'
    [,'odps.properties.rolearn'='${roleran}'])
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/'
USING 'odps-TextStorageHandler.jar';
```

说明:

若需用odps.properties.rolearn属性，详情请参见访问OSS非结构化数据的STS模式授权的自定义授权。若不用，可以参考一键授权或者在LOCATION上用明文AK。

- 通过对External Table的insert操作实现数据输出到OSS

  通过自定义StorageHandler 创建External Table关联上OSS存储路径后，可以对External Table做标准的SQL的insert overwrite/into操作既可将数据输出到OSS方式与内置StorageHandler一样。

```
INSERT OVERWRITE|INTO TABLE <external_tablename> [PARTITION (
partcol1=val1, partcol2=val2 ...)]
select_statement
FROM <from_tablename>
[WHERE where_condition];
```

  insert操作执行成功后，与内置StorageHandler一样，可以在OSS对应LOCATION路径看到生成一系列文件于*.odps*文件夹中。

7.6 访问OTS非结构化数据

本文将进一步为您介绍如何将来自Table Store（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。

您可以通过DataWorks配合MaxCompute对外部表进行可视化的创建、搜索、查询、配置、加工和分析。详情请参见外部表。

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Table Store文档对其进行了解。

MaxCompute与Table Store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Table Store存储时，推荐您使用Table Store私网地址，即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*。

Table Store与MaxCompute都有其自身的类型系统。在MaxCompute处理Table Store数据时，两者之间的类型对应关系如下所示。

MaxCompute Type	Table Store 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。因此，MaxCompute结合了阿里云的访问控制服务（RAM）和令牌服务（STS）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，例如角色AliyunODPSDefaultRole、AliyunODPSRoleForOtherUser。

2. 修改策略内容设置。

```
--当MaxCompute和Table Store的Owner是同一个账号时，进行如下设置。
{
"Statement": [
{
 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
}
--当MaxCompute和Table Store的Owner不是同一个账号时，进行如下设置。
{
"Statement": [
{
 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "MaxCompute的Owner云账号的UID@odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
}
```

说明：

您可以单击右上角的头像，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。

```
{
"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"
}
]
}
--还可自定义其它权限。
```

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Table Store表数据的描述引入到MaxCompute的meta系统内部后，即可实现对Table Store数据的处理。本节通过下述示例为您说明MaxCompute对接Table Store的一些概念和实现。

创建外部表语句示例如下。

```
DROP TABLE IF EXISTS ots_table_external;
CREATE EXTERNAL TABLE IF NOT EXISTS ots_table_external
(
odps_orderkey bigint,
odps_orderdate string,
odps_custkey bigint,
odps_orderstatus string,
odps_totalprice double
)
STORED BY 'com.aliyun.odps.TableStoreStorageHandler' -- (1)
WITH SERDEPROPERTIES ( -- (2)
'tablestore.columns.mapping'=':o_orderkey,:o_orderdate,o_custkey,
o_orderstatus,o_totalprice', -- ①
'tablestore.table.name'='ots_tpch_orders' -- ②
```

```
'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole'
 --③
)
LOCATION 'tablestore://odps-ots-dev.cn-shanghai.ots-internal.aliyuncs.
com'; --  (3)
```

示例说明：

- `com.aliyun.odps.TableStoreStorageHandler`**是MaxCompute内置的处理Table Store数据的StorageHandler，定义了MaxCompute和Table Store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，有三个必须指定的选项，分别是**`tablestore.columns.mapping`、`tablestore.table.name`、`odps.properties.rolearn`。

  1. `tablestore.columns.mapping`**：用于描述MaxCompute将访问的Table Store表的列，包括主键和属性列。**
     - **以冒号（：）开头用于表示Table Store主键，例如示例中的**`:o_orderkey`**和**`:o_orderdate`**，其它的均为属性列。**
     - **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，无需提供全部的属性列，只需提供需要通过MaxCompute访问的属性列。提供的属性列必须是Table Store表的列，否则即使外表可以创建成功，查询时也会报错。**
  2. `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不会主动创建Table Store表。**
  3. `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RAM角色管理进行获取。**
- **LOCATION Clause：用来指定Table Store的Instance名、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。**

您可以执行如下语句，查看创建好的外部表结构信息。

```
desc extended <table_name>;
```

在返回的信息里，除了包含和内部表一样的基础信息，Extended Info还包含外部表StorageHandler 、Location等信息。

查询外部表

创建完外部表后，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;
```

使用常见的MaxCompute SQL语句访问Table Store时，所有的操作细节（例如列名的选择）是在MaxCompute内部处理完成的。上述SQL示例中，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，这是因为在创建外部表的DDL语句中，已经做了对应的mapping。您也可以在创建外部表时按需选择保留原始的Table Store主键/列名。

如果您需要对一份数据做多次计算，相比每次从Table Store去远程读数据，更高效的方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。同时因为表存储在MaxCompute内部，所以访问速度会比访问外部的Table Store更快。这种方法非常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上完成创建（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 Store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，如果您需要对internal_orders中的数据进行处理后再写回Table Store，则可通过对外部表执行`insert overwrite table`操作实现，示例如下。

```
INSERT OVERWRITE TABLE ots_table_external
SELECT odps_orderkey, odps_orderdate, odps_custkey, CONCAT(odps_custk
ey, 'SHIPPED'), CEIL(odps_totalprice)
```

```
FROM internal_orders;
```

说明:

如果ODPS表内数据本身有一定的顺序，例如已经按照Primary Key做过一次排序，则在写入到OTS表时，会导致压力集中在一个OTS分区上面，无法充分利用分布式写入的特点。因此，当出现这种情况时，建议您通过`distribute by rand()`先将数据打散，示例如下。

```
INSERT OVERWRITE TABLE ots_table_external
SELECT odps_orderkey, odps_orderdate, odps_custkey, CONCAT(odps_custk
ey, 'SHIPPED'), CEIL(odps_totalprice)
FROM (SELECT * FROM internal_orders DISTRIBUTE BY rand()) t;
```

对于Table Store这种KV数据的NoSQL存储介质，MaxCompute的输出将只影响相对应主键所在的行，例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Table Store行上面，也只会更新在创建外部表（ots_table_external）时指定的属性列，而不会修改未在外部表中出现的数据列。

说明:

- **将MaxCompute中的数据写入OTS时一次不能超过4MB，否则需要用户剔除掉超大数据再写入。此时可能会产生报错。**

  ```
  ODPS-0010000:System internal error - Output to TableStore failed
  with exception:
  TableStore BatchWrite request id XXXXX failed with error code
  OTSParameterInvalid and message:The total data size of BatchWrite
  Row request exceeds the limit
  ```

- **将数据批量写入或分行写入，都算一次操作。详细描述请参考BatchWriteRow。因此如果批量写入数据量太大，也可以分行写入。**
- **将数据批量写入时请注意不要有重复行，否则可能产生如下报错。**

  ```
  ErrorCode: OTSParameterInvalid, ErrorMessage: The input parameter
  is invalid
  ```

  详细描述请参考使用BatchWriteRow一次提交100条数据的时候报OTSParameterInvalid错误。

更多详情请参见MaxCompute访问TableStore（OTS）数据。

8 PyODPS

8.1 快速入门

本文用于帮助您快速开始使用PyODPS，并且能够用于实际项目。

前提条件

请您首先确保已开通MaxCompute及DataWorks服务，登录DataWorks数据开发控制台，详情请参见开通MaxCompute。

背景信息

PyODPS是MaxCompute的Python版本的SDK，提供简单方便的Python编程接口。MaxCompute大多数开发都可以通过SQL方式实现，但对于复杂的业务场景以及自定义UDF函数都需使用Python的能力，应用场景举例如下：

- 接口对接

  例如：MaxCompute表的每条数据，都需要通过身份证号码和外部HTTP的验证接口进行打通。

- 异步调用

  例如：对于上千个处理逻辑相似的任务，如果对于不同任务都单独创建一个节点，不利于管理，并且在同一时间会占用过多资源。利用PyODPS可以异步生成SQL执行任务队列，通过队列实现任务数的并发，方便统一管理所有节点。

- UDF

  例如：当MaxCompute内置函数无法满足需求时，需要快速开发UDF自定义函数。例如，要针对DECIMAL类型进行千分位数字格式化展示。

- 提升SQL性能

  例如：要计算库存事务是否遵循先进先出的规则。由于只有一张表，每条数据都需要跟历史数据进行比对，而通常库存事务表体量巨大，时间复杂度是$O(N^2)$，往往无法运行处历史至今的结果。用Python结合Cache循环每条记录，只需遍历一次即可，时间复杂度是O(N)，可极大提升效率。

- 其他SQL很难解决的场景

  例如：金额分摊模型，10元由3个人均分，把尾差摊到其中一人身上。

操作步骤

1. 新建PyODPS节点

   为方便您快速开始，本文中使用DataWorks PyODPS节点进行开发，详情请参见PyODPS节点。

   > 说明:
   > PyODPS节点底层的Python版本为2.7 。
   >
   > PyODPS节点获取到本地处理的数据不能超过50MB，节点运行时占用内存不能超过1G，否则节点任务会被系统Kill，请避免在PyODPS任务中写额外的Python数据处理代码。
   >
   > 在DataWorks上编写代码并进行调试效率较低，为提升运行效率，建议本地安装IDE进行代码开发，详情请参见安装指南。

   a) 新建业务流程

   右键单击数据开发下的业务流程，选择新建业务流程。

   b) 新建节点

   右键单击数据开发，选择新建数据开发节点 > PyODPS。

2. 编辑PyODPS节点

   a) 编写程序代码

   以下为Hello World测试代码。

```
import time                          #与Java类似，调用额外的SDK需要先进行模块的导入。
import datetime                      #本例中只用到print函数。
import base64
import hashlib
import httplib
import json
import sys
import csv
from odps import ODPS                #与ODPS调用相关的SDK需要增加这个模块导入。
def main():
    print"Hello World"               #往日志文件输出。
if __name__ == "__main__":           #程序的主入口。
```

```
    main()
```

b) 运行代码

完成编辑后，点击运行按钮即可，您可以在下方的运行日志中看到运行结果。



此时您已完成了第一段代码。代码的主入口通过`if __name__ == "__main__"`进行判断，该语句只有直接运行此脚本时才会生效（name=main）。当作为模块被其他代码文件引用时，不会生效（name=py文件名）。

8.2 安装指南

PyODPS是MaxCompute的Python版本的SDK，提供对MaxCompute对象的基本操作及DataFrame框架，帮助您在MaxCompute上使用Python进行数据分析。本文为您介绍如何安装PyODPS及使用限制。

使用限制

- 读取SQL限制
- DataWorks PyODPS节点使用限制

- 由于沙箱的限制，部分pandas计算后端执行本地调试通过的程序，无法在MaxCompute上调试通过。

安装步骤

如果您的服务器可以访问公网，推荐使用Python包管理工具（PIP）安装PyODPS环境。您可以参见PIP安装说明安装PIP工具。

说明:

如果您想加快下载速度，推荐使用 阿里云镜像 。

前提条件

安装PyOPDS环境前，您的Python环境需要满足以下条件：

- `setuptools 3.0`或以上版本
- `requests 2.4.0`或以上版本

参考安装命令如下。

```
pip install setuptools>=3.0
pip install requests>=2.4.0
```

同时，建议您安装以下工具，提升Tunnel上传的速度：

- `greenlet 0.4.10`或以上版本
- `cython 0.19.0`或以上版本

参考安装命令如下。

```
pip install greenlet>=0.4.10  # 可选，安装后能加速Tunnel上传。
pip install cython>=0.19.0  # 可选，不建议Windows用户安装。
```

说明:

如果您的服务器是Windows系统，请确认您安装合适版本的Visual C++ 和Cython，否则无法实现Tunnel上传加速。

安装PyODPS命令如下。

```
pip install pyodps
```

使用如下命令检查安装是否完成。

```
python -c "from odps import ODPS"
```

如果您使用的Python不是系统默认的Python版本，安装完PIP后，您可以执行下列命令进行Python版本切换。

```
/home/tops/bin/python2.7 -m pip install setuptools>=3.0
```

8.3 工具平台使用指南

8.3.1 工具平台使用概述

PyODPS可在DataWorks等数据开发平台中作为数据开发节点调用。这些平台提供了PyODPS运行环境和调度执行的能力，无需您手动创建ODPS入口对象。如果您想从上述平台迁移到自行部署的PyODPS环境，请仔细阅读下列章节：

- 从平台到自行部署
- DataWorks用户使用指南

8.3.2 从平台到自行部署

如果您需要在本地调试PyODPS，或平台的资源限制不能满足您的要求，此时您可能需要从平台迁移到自己部署的PyODPS环境。

安装PyODPS的步骤请参见安装指南。

您需要在自己的环境中手动创建之前平台为您创建的ODPS入口对象。之后，您可以在之前的平台中使用下列语句生成创建ODPS入口对象所需要的语句模板，手动修改为可用的代码。

```
print("\nfrom odps import ODPS\no = ODPS(%r, '<access-key>', %r, '<
endpoint>')\n" % (o.account.access_id, o.project))
```

其中，<access-key>和<endpoint>需要手动替换成可用的值。这两项信息您可以在DataWorks中点击用户信息获得。



您可以参见配置Endpoint获得Endpoint值，或联系项目管理员。

第三个参数为项目名称。在您从DataWorks平台迁移时，如果手动填写该参数（而非使用上述语句自动生成），需注意DataWorks中的项目名称（下称 <DataWorks Project Name>）与实际的MaxCompute项目名称并不是同一个概念。在DataWorks中，`<DataWorks Project Name>_dev` 表示DataWorks开发环境中作业运行的MaxCompute项目名称，而 `<DataWorks Project Name>` 表示DataWorks生产环境中作业运行的MaxCompute项目名称。开发和生产环境详情请参见简单模式和标准模式的区别。

修改完成后，将上述代码放置到所有代码的最顶端即可。

8.3.3 在DataWorks上使用PyODPS

本文为您介绍如何在DataWorks上使用PyODPS和使用限制。

新建工作流节点

在工作流节点中会包含PYODPS类型节点。

ODPS入口

DataWorks的PyODPS节点中，将会包含一个全局变量odps或者o，即ODPS入口。您不需要手动定义ODPS入口。

```
print(o.exist_table('pyodps_iris'))
```

执行SQL

详情请参见执行SQL文档。

> **说明:**
> Dataworks上默认未开启instance tunnel，即instance.open_reader默认使用result接口（最多一万条记录）。开启instance tunnel后，您可以通过reader.count获取记录数。如果您需要迭代获取全部数据，则需要关闭limit限制。

您可以通过下列语句在全局范围内打开instance tunnel并关闭limit限制。

```
options.tunnel.use_instance_tunnel = True
options.tunnel.limit_instance_tunnel = False  # 关闭limit限制，读取全部数据。

with instance.open_reader() as reader:
```

```
    # 通过instance tunnel可读取全部数据。
```

您也可以通过在open_reader上添加tunnel=True，实现仅对本次open_reader开启instance tunnel。同时，您还可以添加 limit=False，实现仅对本次关闭limit限制。

```
with instance.open_reader(tunnel=True, limit=False) as reader:
# 本次open_reader使用instance tunnel接口，且能读取全部数据。
```

说明:

若您未开启instance tunnel，可能导致获取数据格式错误，解决方法请参见PyODPS常见问题。

DataFrame

- 执行

  在DataWorks的环境里，DataFrame概述 的执行需要显式调用立即执行的方法（如execute，head等）。

  ```
  from odps.df import DataFrame
  iris = DataFrame(o.get_table('pyodps_iris'))
  for record in iris[iris.sepal_width < 3].execute():  # 调用立即执行的方法，处理每条record。
  ```

  如果您需要在print时调用立即执行，需要开启options.interactive。

  ```
  from odps import options
  from odps.df import DataFrame
  options.interactive = True  # 在开始处打开开关。
  iris = DataFrame(o.get_table('pyodps_iris'))
  print(iris.sepal_width.sum())  # print时会立即执行。
  ```

- 打印详细信息

  通过设置options.verbose选项。在DataWorks上，默认已经处于打开状态，运行过程会打印logview等详细过程。

获取调度参数

与DataWorks中的SQL节点不同，为了避免侵入代码，PyODPS节点不会在代码中替换 ${param_name}这样的字符串，而是在执行代码前，在全局变量中增加一个名为args的dict，调度

参数可以在此获取。例如，在节点基本属性 > 参数中设置ds=${yyyymmdd}，则可以通过以下方式在代码中获取该参数。

```
print('ds=' + args['ds'])
```

```
ds=20161116
```

说明:

如果您需要获取名为ds=${yyyymmdd}的分区，则可以使用如下方法。

```
o.get_table('table_name').get_partition('ds=' + args['ds'])
```

受限功能

由于缺少matplotlib等包，如下功能可能受限：

- DataFrame的plot函数。
- DataFrame自定义函数需要提交到MaxCompute执行。由于Python沙箱限制，第三方库只支持所有的纯粹Python库以及Numpy，因此不能直接使用Pandas。
- DataWorks中执行的非自定义函数代码可以使用平台预装的Numpy和Pandas。其他带有二进制代码的三方包不被支持。

由于兼容性原因，在DataWorks中，options.tunnel.use_instance_tunnel默认设置为False。如果如果需要全局开启instance tunnel，需要手动将该值设置为True。

由于实现的原因，Python的atexit包不被支持，请使用try-finally结构实现相关功能。

使用限制

- PyODPS节点底层的Python版本为2.7。
- PyODPS节点获取到本地处理的数据不能超过50MB，节点运行时占用内存不能超过1G，否则节点任务会被系统Kill。请避免在PyODPS任务中写额外的Python数据处理代码。
- 在DataWorks上编写代码并进行调试效率较低，为提升运行效率，建议本地安装IDE进行代码开发，详情请参见安装指南。
- 在DataWorks上使用PyODPS时，为了防止对DataWorks的Gate Way造成压力，对内存和CPU都有限制。该限制由DataWorks统一管理。如果您发现有Got killed报错，即表名内存使用超限，进程被Kill。因此，请尽量避免本地的数据操作。通过PyODPS发起的SQL和DataFrame任务（除to_pandas外）不受此限制。

8.4 基本操作

8.4.1 基本操作概述

PyODPS提供直接针对ODPS对象的基本操作接口，您可以使用符合Python习惯的编程方式操作MaxCompute。

PyODPS的基本操作主要包括以下几个部分：

- 项目空间
- 表
- SQL
- 任务实例
- 资源
- 函数
- XFlow和模型

8.4.2 项目空间

项目空间是ODPS的基本组织单元，详情请参见项目空间。

- 您可以使用ODPS入口对象的`get_project`方法获取到项目空间。

```
project = o.get_project('my_project')  # 取到某个特定项目。
project = o.get_project()              # 取到默认项目。
```

如果不提供`my_project`参数，则获取到默认项目空间。

- 您可以使用`exist_project`方法检验某个项目空间是否存在。

8.4.3 表

本文向您介绍ODPS表的基本操作。包括创建表、创建表的Schema、同步表更新、获取表数据、删除表、表分区操作以及如何转换为DataFrame对象。

基本操作

表是MaxCompute的数据存储单元。您可以使用ODPS入口对象的`list_tables`来列出项目空间下的所有表。

```
for table in o.list_tables():
# 处理每个表。
```

您可以通过调用`exist_table`来判断表是否存在，通过调用`get_table`来获取表。

```
t = o.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.comment
'Dual Table Comment'
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>]
t.schema['c_int_a']
<column c_int_a, type bigint>
t.schema['c_int_a'].comment
'Comment of column c_int_a'
```

通过提供`project`参数，您可以实现跨Project获取表。

```
t = o.get_table('dual', project='other_project')
```

创建表的Schema

有两种方法完成初始化：

- 第一种方式即通过表的列、以及可选的分区来进行初始化。

```
from odps.models import Schema, Column, Partition
columns = [Column(name='num', type='bigint', comment='the column'),
           Column(name='num2', type='double', comment='the column2
')]
partitions = [Partition(name='pt', type='string', comment='the
partition')]
schema = Schema(columns=columns, partitions=partitions)
schema.columns
[<column num, type bigint>,
 <column num2, type double>,
 <partition pt, type string>]
schema.partitions
```

```
[<partition pt, type string>]
schema.names  # 获取非分区字段的字段名
['num', 'num2']
schema.types  # 获取非分区字段的字段类型
[bigint, double]
```

- 第二种方法是使用Schema.from_lists。这种方法更容易调用，但无法直接设置列和分区的注释。

```
schema = Schema.from_lists(['num', 'num2'], ['bigint', 'double'], ['
pt'], ['string'])
schema.columns
[<column num, type bigint>,
 <column num2, type double>,
 <partition pt, type string>]
```

创建表

您可以使用表Schema来创建表，方法如下。

```
table = o.create_table('my_new_table', schema)
table = o.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。
```

更简单的方式是采用 字段名及字段类型 字符串来创建表，方法如下。

```
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。如果您使用的是位于公共云上的服务，或者支持TINYINT、STRUCT等新类型，可以设置options.sql.use_odps2_extension = True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>')
```

同步表更新

当一个表被其他程序更新，例如改变了Schema之后，可以调用 `reload` 方法同步表的更新。

```
table.reload()
```

行记录Record

Record表示表的一行记录，您在Table对象上调用new_record即可创建一个新的Record。

```
t = o.get_table('mytable')
r = t.new_record(['val0', 'val1'])  # 值的个数必须等于表schema的字段数。
r2 = t.new_record()  #  您也可以不传入值。
r2[0] = 'val0' # 可以通过偏移设置值。
r2['field1'] = 'val1'  # 也可以通过字段名设置值。
r2.field1 = 'val1'  # 通过属性设置值。

print(record[0])  # 取第0个位置的值。
print(record['c_double_a'])  # 通过字段取值。
print(record.c_double_a)  # 通过属性取值。
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。
```

获取表数据

获取表数据的方法很多，如果您仅需要查看每个表的最开始的小于1万条数据，可以使用head方法。

```
t = o.get_table('dual')
for record in t.head(3):
# 处理每个Record对象。
```

或在Table上执行open_reader操作来读取数据。

使用with表达式的写法如下所示。

```
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，此处可以改造成并行操作。
# 处理一条记录。
```

不使用with表达式的写法如下所示。

```
reader = t.open_reader(partition='pt=test')
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。
```

更简单的调用方法是使用ODPS对象的`read_table`方法，举例如下。

```
for record in o.read_table('test_table', partition='pt=test'):
# 处理一条记录。
```

写入数据

类似于`open_reader`，Table对象同样能执行`open_writer`来开启writer，并写入数据。

使用ith表达式的写法如下所示。

```
with t.open_writer(partition='pt=test') as writer:
records = [[111, 'aaa', True],                # 这里可以是list。
           [222, 'bbb', False],
           [333, 'ccc', True],
           [444, '中文', False]]
writer.write(records)  # 这里records可以是可迭代对象。

records = [t.new_record([111, 'aaa', True]),    # 也可以是Record对象。
           t.new_record([222, 'bbb', False]),
           t.new_record([333, 'ccc', True]),
           t.new_record([444, '中文', False])]
writer.write(records)

with t.open_writer(partition='pt=test', blocks=[0, 1]) as writer:
 # 此处同时打开两个block。
     writer.write(0, gen_records(block=0))
     writer.write(1, gen_records(block=1))  # 此处两个写操作可以多线程并
行，各个block间是独立的。
```

不使用ith表达式的写法如下所示。

```
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()  # 请不要忘记关闭writer，否则数据可能写入不完全。
```

如果分区不存在，可以使用`create_partition`参数指定创建分区，举例如下。

```
with t.open_writer(partition='pt=test', create_partition=True) as
writer:
    records = [[111, 'aaa', True],                # 这里可以是list。
               [222, 'bbb', False],
               [333, 'ccc', True],
               [444, '中文', False]]
    writer.write(records)  # 这里records可以是可迭代对象。
```

说明:

每次调用write_table，MaxCompute都会在服务端生成一个文件。这一操作需要较大的时间开销，同时过多的文件会降低后续的查询效率。因此，建议在使用write_table方法时，一次性写入多组数据，或者传入一个Generator对象。

删除表

```
o.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。
```

创建DataFrame

PyODPS提供了DataFrame框架DataFrame框架，支持更方便地方式来查询和操作ODPS数据。

使用to_df方法，即可转化为DataFrame对象。

```
table = o.get_table('my_table_name')
 df = table.to_df()
```

表分区

- 基本操作

  判断是否为分区表：

  ```
  if table.schema.partitions:
   print('Table %s is partitioned.' % table.name)
  ```

  遍历表全部分区：

  ```
  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：

  ```
  table.exist_partition('pt=test,sub=2015')
  ```

  获取分区：

  ```
  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)  # 分区。不存在时候才创建。
  ```

- 删除分区

  ```
  t.delete_partition('pt=test', if_exists=True)  # 存在时才删除。
  ```

```
partition.drop()  # Partition对象存在的时候直接drop。
```

数据上传下载通道

ODPS Tunnel是MaxCompute的数据通道，用户可以通过Tunnel向ODPS中上传或者下载数据。

说明:

不推荐直接使用Tunnel接口，推荐您直接使用表的写和读接口。 如果安装了 Cython，在安装PyODPS时会编译C代码，加速Tunnel的上传和下载。

- 上传示例

```
from odps.tunnel import TableTunnel

table = o.get_table('my_table')

tunnel = TableTunnel(odps)
upload_session = tunnel.create_upload_session(table.name, partition_
spec='pt=test')

with upload_session.open_record_writer(0) as writer:
    record = table.new_record()
    record[0] = 'test1'
    record[1] = 'id1'
    writer.write(record)

    record = table.new_record(['test2', 'id2'])
    writer.write(record)

upload_session.commit([0])
```

- 下载示例

```
from odps.tunnel import TableTunnel

tunnel = TableTunnel(odps)
download_session = tunnel.create_download_session('my_table',
partition_spec='pt=test')

with download_session.open_record_reader(0, download_session.count)
as reader:
    for record in reader:
# 处理每条记录。
```

说明:

PyODPS不支持上传外部表(OSS/OTS)。

8.4.4 SQL

PyODPS支持ODPS SQL的查询和执行结果的读取。

`execute_sql`和`run_sql`方法的返回值是运行实例 。

说明:

在ODPS Console中可执行的命令并非都是ODPS接受的SQL语句。 在调用非DDL/DML语句时，请使用其它方法，例如，调用GRANT/REVOKE等语句时，请使用`run_security_query`方法；调用PAI命令时，请使用`run_xflow`或`execute_xflow`方法。

执行SQL

```
o.execute_sql('select * from dual')  #  同步的方式执行，会阻塞直到SQL执行完成。
instance = o.run_sql('select * from dual')  # 异步的方式执行。
print(instance.get_logview_address())  # 获取logview地址。
instance.wait_for_success()  # 阻塞直到完成。
```

设置运行参数

您可以通过设置`hints`来设置运行时参数，参数类型是dict。

```
o.execute_sql('select * from pyodps_iris', hints={'odps.sql.mapper.
split.size': 16})
```

如果全局配置设置了`sql.settings`，则每次运行时都会添加相关的运行时参数。

```
from odps import options
options.sql.settings = {'odps.sql.mapper.split.size': 16}
o.execute_sql('select * from pyodps_iris')  # 会根据全局配置添加hints。
```

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 o.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可能执行了`desc`命令，这时可以通过`reader.raw`取到原始的SQL执行结果。

```
with o.execute_sql('desc dual').open_reader() as reader:
print(reader.raw)
```

如果`options.tunnel.use_instance_tunnel == True`，在调用`open_reader`时，PyODPS会默认调用Instance Tunnel，否则会调用旧的Result接口。如果您使用了版本较低的MaxCompute服务，或者调用Instance Tunnel出现了问题，PyODPS会给出警告并自动降级到旧的Result接口，您可根据警告信息判断导致降级的原因。如果Instance Tunnel的结果不合预期，请将该选项设为False。在调用`open_reader`时，也可以使用 `tunnel`参数来指定使用何种结果接口。

```
# 使用Instance Tunnel。
```

```
with o.execute_sql('select * from dual').open_reader(tunnel=True) as
reader:
    for record in reader:
    # 处理每一个record。
# 使用Results接口。
with o.execute_sql('select * from dual').open_reader(tunnel=False) as
reader:
    for record in reader:
    # 处理每一个record。
```

PyODPS默认不限制从Instance读取的数据规模。对于受保护的Project，通过Tunnel下载数据将受限。此时，如果未设置options.tunnel.limit_instance_tunnel，则数据量限制会被自动打开。此时，可下载的数据条数受到Project配置限制，通常该限制为10000条。如果您需要手动限制下载数据的规模，可以为open_reader方法增加limit选项， 或者设置options.tunnel.limit_instance_tunnel = True。

如果您使用的MaxCompute只能支持旧Result接口，并且需要读取所有的数据，您可将SQL结果写入另一张表后用读表接口读取（可能受到Project安全设置的限制）。

从PyODPS v0.7.7.1开始，您可以通过open_reader方法使用Instance tunnel来获取全量数据。

```
instance = o.execute_sql('select * from movielens_ratings limit 20000
')
with instance.open_reader() as reader:
    print(reader.count)
    # for record in reader 遍历这2万条数据，这里通过切片只取10条。
    for record in reader[:10]:
        print(record)
```

设置alias

在运行时，如果某个UDF引用的资源是动态变化的，您可以将旧的资源名alias到新的资源，无需重新删除或创建新的UDF。

```
from odps.models import Schema
myfunc = '''\
from odps.udf import annotate
from odps.distcache import get_cache_file

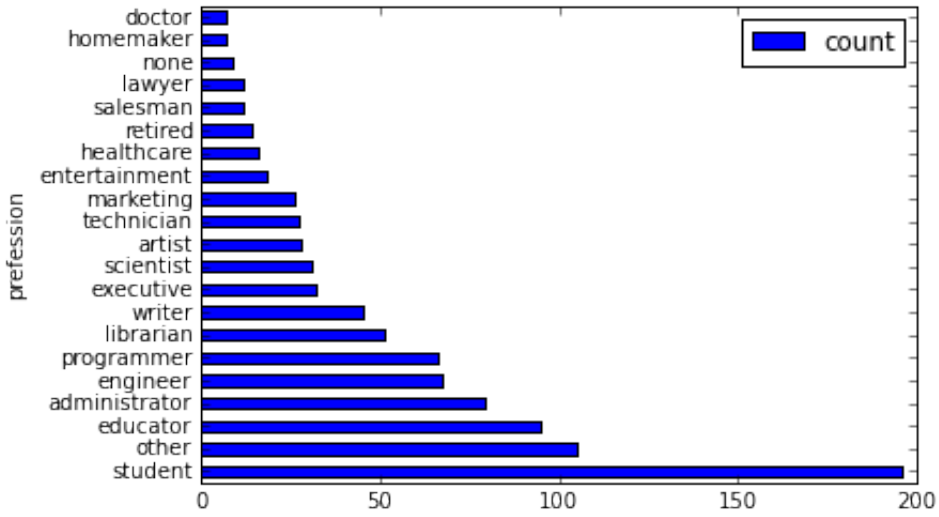
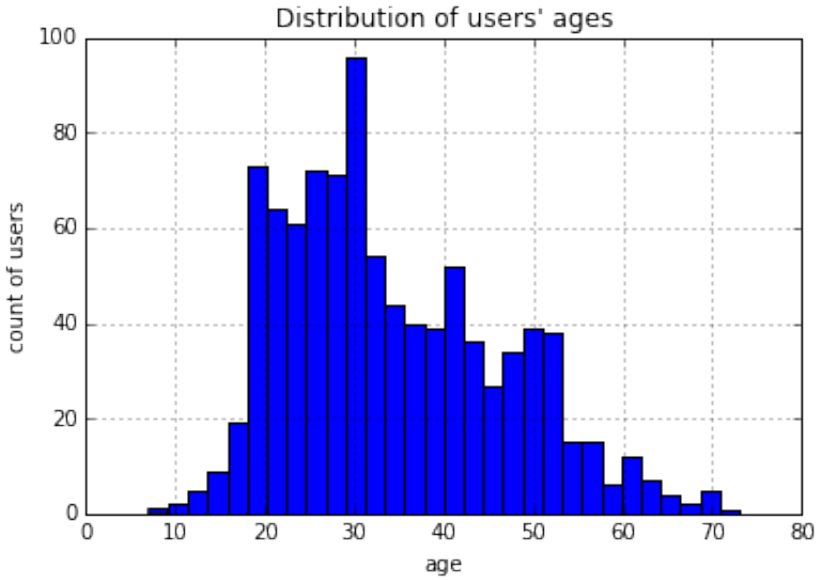
@annotate('bigint->bigint')
class Example(object):
    def __init__(self):
        self.n = int(get_cache_file('test_alias_res1').read())

    def evaluate(self, arg):
        return arg + self.n
'''
res1 = o.create_resource('test_alias_res1', 'file', file_obj='1')
o.create_resource('test_alias.py', 'py', file_obj=myfunc)
o.create_function('test_alias_func',
                  class_type='test_alias.Example',
                  resources=['test_alias.py', 'test_alias_res1'])

table = o.create_table(
```

```
    'test_table',
    schema=Schema.from_lists(['size'], ['bigint']),
    if_not_exists=True
)

data = [[1, ], ]
# 写入一行数据，只包含一个值1。
o.write_table(table, 0, [table.new_record(it) for it in data])

with o.execute_sql(
    'select test_alias_func(size) from test_table').open_reader() as
reader:
    print(reader[0][0])
res2 = o.create_resource('test_alias_res2', 'file', file_obj='2')
# 把内容为1的资源alias成内容为2的资源，您不需要修改UDF或资源。
with o.execute_sql(
    'select test_alias_func(size) from test_table',
    aliases={'test_alias_res1': 'test_alias_res2'}).open_reader() as
reader:
    print(reader[0][0])
```

在交互式环境执行SQL

在IPython和Jupyter里支持使用SQL插件的方式运行SQL和参数化查询。

设置biz_id

在少数情形下，提交SQL时需要同时提交`biz_id`，否则执行会报错。此时，您可以通过设置全局`options`里的`biz_id`解决此类报错。

```
from odps import options
options.biz_id = 'my_biz_id'
o.execute_sql('select * from pyodps_iris')
```

8.4.5 任务实例

Task是ODPS的基本计算单元，例如SQLTask。

基本操作

Task在执行时会被实例化， 以ODPS实例的形式存在。您可以调用`list_instances`来获取项目空间下的所有的Instance。`exist_instance`用于判断是否存在某Instance；`get_instance`用于获取实例。

```
for instance in o.list_instances():
    print(instance.id)
    o.exist_instance('my_instance_id')
```

您可以在ODPS入口使用`stop_instance`，或通过Instance对象调用`stop`方法来停止一个Instance。

获取Logview地址

对于SQL等任务，通过调用get_logview_address方法即可获取Logview地址。

```
# 从已有的instance对象获取Logview地址。
instance = o.run_sql('desc pyodps_iris')
print(instance.get_logview_address())
# 从instance id获取Logview地址。
instance = o.get_instance('20160426055209945g9k5pvyi2')
print(instance.get_logview_address())
```

对于XFlow任务，需要先枚举其子任务，再获取子任务的Logiew。

```
instance = o.run_xflow('AppendID', 'algo_public', {'inputTableName': '
input_table', 'outputTableName': 'output_table'})
for sub_inst_name, sub_inst in o.get_xflow_sub_instances(instance).
items():
    print('%s: %s' % (sub_inst_name, sub_inst.get_logview_address()))
```

任务实例状态

一个Instance的状态可以是Running、Suspended或者 Terminated，用户可以通过status属性来获取状态。is_terminated方法返回当前Instance是否已经执行完毕，is_successful方法返回当前Instance是否正确地执行完毕， 任务处于运行中或者执行失败都会返回False。

```
instance = o.get_instance('20160426055209945g9k5pvyi2')
instance.status
<Status.TERMINATED: 'Terminated'>
    from odps.models import Instance
    instance.status == Instance.Status.TERMINATED
    True
    instance.status.value
    'Terminated'
```

调用wait_for_completion方法会阻塞直到Instance执行完成。wait_for_success方法同样会阻塞，但如果最终任务执行失败，会抛出相关异常。

子任务操作

一个Instance在运行时，可能包含一个或者多个子任务，这些子任务称为Task（注意：这里的Task不同于ODPS的计算单元）。

您可以通过get_task_names方法获取所有的Task任务。该方法返回一个所有子任务的名称列表。

```
instance.get_task_names()
```

```
['SQLDropTableTask']
```

得到Task的名称后，您可以通过get_task_result方法获取这个Task的执行结果。

get_task_results方法是以字典的形式返回每个Task的执行结果。

```
instance = o.execute_sql('select * from pyodps_iris limit 1')
instance.get_task_names()
['AnonymousSQLTask']
instance.get_task_result('AnonymousSQLTask')
'"sepallength","sepalwidth","petallength","petalwidth","name"\n5.1,3.5
,1.4,0.2,"Iris-setosa"\n'
instance.get_task_results()
OrderedDict([('AnonymousSQLTask', "sepallength","sepalwidth","
petallength","petalwidth","name"\n5.1,3.5,1.4,0.2,"Iris-setosa"\n')])
```

您可以通过get_task_progress方法获取在任务实例运行时Task当前的运行进度。

```
while not instance.is_terminated():
    for task_name in instance.get_task_names():
        print(instance.id, instance.get_task_progress(task_name).
get_stage_progress_formatted_string())
        time.sleep(10)
20160519101349613gzbzufck2 2016-05-19 18:14:03 M1_Stg1_job0:0/1/1[100
%]
```

8.4.6 资源

PyODPS主要支持两种资源类型，即文件和表。

资源常用在UDF和MapReduce中。

list_resources用于列出所有资源，exist_resource用于判断资源是否存在。您可以调用delete_resource或直接通过Resource对象调用drop方法删除资源。

文件资源

文件资源包括基础的*file*类型，以及*py*、*jar*、*archive*类型。

- 创建文件资源

  创建文件资源可以通过给定资源名、文件类型和一个file-like的对象（或字符串对象）来创建。

  ```
  resource = o.create_resource('test_file_resource', 'file', file_obj=
  open('/to/path/file'))  # 使用file-like的对象创建文件资源。
  ```

```
resource = o.create_resource('test_py_resource', 'py', file_obj='
import this')  # 使用字符串创建文件资源。
```

- 读取和修改文件资源

  对文件资源调用open方法，或在ODPS入口调用open_resource都可以打开一个资源， 打开后的对象是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:  # 以读模式打开资源。
     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 o.open_resource('test_file_resource', mode='r+') as fp:  # 读写
模式打开资源。
     fp.read()
     fp.tell()  # 定位当前位置。
     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，文件内容会先被清空。
  - a：追加模式，只能写入内容到文件末尾。
  - r+：读写模式，可以任意读写内容。
  - 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，例如一些压缩文件需要以这种模式打开。rb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

```
o.create_resource('test_table_resource', 'table', table_name='
my_table', partition='pt=test')
```

- 更新表资源

```
table_resource = o.get_resource('test_table_resource')
```

```
table_resource.update(partition='pt=test2', project_name='my_project
2')
```

- 获取表及分区

```
table_resource = o.get_resource('test_table_resource')
table = table_resource.table
print(table.name)
partition = table_resource.partition
print(partition.spec)
```

- 读写内容

```
table_resource = o.get_resource('test_table_resource')
with table_resource.open_writer() as writer:
 writer.write([0, 'aaaa'])
 writer.write([1, 'bbbbb'])
 with table_resource.open_reader() as reader:
  for rec in reader:
   print(rec)
```

8.4.7 函数

您可以编写自定义函数并在MaxCompute SQL中使用它们。

基本操作

您可以调用MaxCompute入口对象的`list_functions`来获取项目空间下的所有函数。

`exist_function`用于判断是否存在函数；`get_function`用于获取函数对象。

创建函数

```
# 引用当前project中的资源。
resource = o.get_resource('my_udf.py')
function = o.create_function('test_function', class_type='my_udf.Test
', resources=[resource])
# 引用其他project中的资源。
resource2 = o.get_resource('my_udf.py', project='another_project')
function2 = o.create_function('test_function2', class_type='my_udf.
Test', resources=[resource2])
```

删除函数

```
o.delete_function('test_function')
function.drop()  # Function对象存在时直接调用drop方法。
```

更新函数

只需对函数调用`update`方法即可。

```
function = o.get_function('test_function')
new_resource = o.get_resource('my_udf2.py')
function.class_type = 'my_udf2.Test'
function.resources = [new_resource, ]
```

```
function.update()  # 更新函数。
```

8.4.8 XFlow和模型

XFlow实现了MaxCompute对算法包的封装，您可以使用PyODPS可以执行XFlow。

XFlow

您可以同步调用或异步调用下面的PAI命令。

```
PAI -name AlgoName -project algo_public -Dparam1=param_value1 -Dparam2
=param_value2 ...
```

- 同步调用

```
inst = o.execute_xflow('AlgoName', 'algo_public', parameters={'
param1': 'param_value1', 'param2': 'param_value2', ...})
```

- 异步调用

```
inst = o.run_xflow('AlgoName', 'algo_public', parameters={'param1':
 'param_value1', 'param2': 'param_value2', ...})
```

参数不应包含命令两端的引号，也不应该包含末尾的分号。

这两个方法都会返回一个Instance对象。由于XFlow的一个Instance包含若干个子Instance，您需要使用下面的方法来获得每个Instance的LogView。

```
for sub_inst_name, sub_inst in o.get_xflow_sub_instances(inst).items
():
    print('%s: %s' % (sub_inst_name, sub_inst.get_logview_address()))
```

`get_xflow_sub_instances`返回的是Instance当前的子Instance，可能会随时间变化，因而可能需要定时查询。为简化这一步骤，您可以使用`iter_xflow_sub_instances`方法。该方法返回一个迭代器，会阻塞执行直到发现新的子Instance或者主Instance结束。

```
# 此处建议使用异步调用。
inst = o.run_xflow('AlgoName', 'algo_public', parameters={'param1': '
param_value1', 'param2': 'param_value2', ...})
for sub_inst_name, sub_inst in o.iter_xflow_sub_instances(inst):  # 此
处将进行异步等待。
    print('%s: %s' % (sub_inst_name, sub_inst.get_logview_address()))
```

在调用`run_xflow`或者`execute_xflow`时，也可以指定运行参数，指定的方法与SQL类似。

```
parameters = {'param1': 'param_value1', 'param2': 'param_value2', ...}
o.execute_xflow('AlgoName', 'algo_public', parameters=parameters,
hints={'odps.xxx.yyy': 10})
```

使用`options.ml.xflow_settings`可以配置全局设置。

```
from odps import options
options.ml.xflow_settings = {'odps.xxx.yyy': 10}
```

```
parameters = {'param1': 'param_value1', 'param2': 'param_value2', ...}
o.execute_xflow('AlgoName', 'algo_public', parameters=parameters)
```

PAI命令的详细介绍，请参见机器学习PAI 。

XFlow模型

在线模型是ODPS提供的模型在线部署能力。

- 部署离线模型上线

  PyODPS提供了离线模型的部署功能。部署方法如下。

  ```
  model = o.create_online_model('online_model_name', 'offline_mo
  del_name')
  ```

- 部署自定义Pipeline上线

  含有自定义Pipeline的在线模型可自行构造`ModelPredictor`对象，示例如下。

  ```
  from odps.models.ml import ModelPredictor, ModelProcessor,
  BuiltinProcessor, PmmlProcessor, PmmlRunMode
  predictor = ModelPredictor(target_name='label')
  predictor.pipeline.append(BuiltinProcessor(offline_model_name='
  sample_offlinemodel',
                                             offline_model_project='
  online_test'))
  predictor.pipeline.append(PmmlProcessor(pmml='data_preprocess.xml',
                                          resources='online_test/
  resources/data_preprocess.xml',
                                           run_mode=PmmlRunMode.
  Converter))
   predictor.pipeline.append(CustomProcessor(class_name='SampleProc
  essor',
                                             lib='libsample_processor.
  so',
                                             resources='online_test/
  resources/sample_processor.tar.gz'))
  model = o.create_online_model('online_model_name', predictor)
  ```

  其中，`BuiltinProcessor`、`PmmlProcessor`和`CustomProcessor`分别指ODPS OfflineModel形成的Pipeline节点、PMML模型文件形成的Pipeline节点和用户自行开发的Pipeline节点。

- 在线模型操作

  与其它ODPS对象类似，创建后，可列举、获取和删除在线模型。

  ```
   models = o.list_online_models(prefix='prefix')
   model = o.get_online_model('online_model_name')
  ```

```
 o.delete_online_model('online_model_name')
```

可使用模型名和数据进行在线预测，输入数据可以是Record，也可以是字典或数组和Schema的组合。

```
data = [[4, 3, 2, 1], [1, 2, 3, 4]]
result = o.predict_online_model('online_model_name', data, schema=['
sepal_length', 'sepal_width', 'petal_length', 'petal_width'])
```

也可为模型设置ABTest。参数中的`modelx`可以是在线模型名，也可以是`get_online_model`获得的模型对象本身，而`percentagex`表示`modelx`在ABTest中所占的百分比，范围为0至100。

```
result = o.config_online_model_ab_test('online_model_name', model1,
percentage1, model2, percentage2)
```

修改模型参数可以通过修改`OnlineModel`对象的属性，再调用`update`方法实现，如下所示。

```
model = o.get_online_model('online_model_name')
model.cpu = 200
model.update()
```

与其它对象不同的是，在线模型的创建和删除较为耗时。PyODPS默认`create_online_model`和`delete_online_model`以及`OnlineModel`的`update`方法在整个操作完成后才返回。用户可以通过`async`选项控制是否要在模型创建请求提交后立即返回， 然后自己控制等待。例如，下列语句

```
model = o.create_online_model('online_model_name', 'offline_mo
del_name')
```

等价于

```
model = o.create_online_model('online_model_name', 'offline_mo
del_name', async=True)
model.wait_for_service()
```

而下列语句

```
o.delete_online_model('online_model_name')
```

等价于

```
o.delete_online_model('* online_model_name *', async=True)
 model.wait_for_deletion()
```

8.5 DataFrame

8.5.1 DataFrame概述

PyODPS提供了DataFrame API，它提供了类似pandas的接口，但是能充分利用ODPS的计算能力。同时能在本地使用同样的接口，用pandas进行计算。

- 快速开始
- 创建DataFrame
- Sequence
- Collection
- 执行
- MapReduce API
- 列运算
- 聚合操作
- 排序、去重、采样、数据变换
- 数据合并
- 窗口函数
- 绘图
- 调试指南

8.5.2 快速入门

本文为您介绍如何创建和操作DataFrame对象。

本文将以movielens 100K进行举例。现在已经有了三张表，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）。

DataFrame对象操作

1. 当运行环境没有提供ODPS对象时，您需要自己创建该对象。

```
from odps import ODPS
o = ODPS('**your-access-id**', '**your-secret-access-key**',
         project='**your-project**', endpoint='**your-end-point**')
```

2. 创建一个DataFrame对象非常容易，只需传入Table对象即可。

```
from odps.df import DataFrame
users = DataFrame(o.get_table('pyodps_ml_100k_users'))
```

3. 您可以通过`dtypes`属性查看这个DataFrame的字段及字段类型。

```
users.dtypes
odps.Schema {
  user_id              int64
  age                  int64
```

```
  sex                 string
  occupation          string
  zip_code            string
}
```

4. 通过head方法，您可以取前N条数据并进行快速预览。

```
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
```

5. 如果您不需要看到所有字段，则可以进行如下操作：

   · 从中筛选出一部分字段。

```
users[['user_id', 'age']].head(5)
   user_id  age
0        1   24
1        2   53
2        3   23
3        4   24
4        5   33
```

   · 只做排除个别字段操作。

```
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，通过计算得到一些新的列。例如将sex为M的置为True，否则为False，并取名为sex_bool。

```
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
```

6. 查看男用户和女用户的个数。

```
users.groupby(users.sex).agg(count=users.count())
   sex  count
0    F    273
```

```
1    M    670
```

7. 将用户按职业划分，从高到底进行排序，查看人数最多的前10职业。

```
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
```

或者通过value_counts方法快速实现。该方法返回的行数受到options.df.odps.sort.limit的限制，详情请参见配置选项。

```
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
```

8. 通过更直观的图查看这份数据。

```
%matplotlib inline
```

9. 通过横向的柱状图实现可视化。

```
users['occupation'].value_counts().plot(kind='barh', x='occupation
', ylabel='prefession')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10653cfd0>
```



10.将年龄分为30组，查看年龄分布的直方图。

```
users.age.hist(bins=30, title="Distribution of users' ages", xlabel
='age', ylabel='count of users')
<matplotlib.axes._subplots.AxesSubplot at 0x10667a510>
```



11.此时，只需要使用`join`将这三张表联合起来，然后保存为一张新的表。

```
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
}
```

12.将年龄（0到80岁）分成8个年龄段。

```
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')]
```

13.取分组和年龄唯一的前10条查看。

```
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
```

14.最后，查看在各个年龄分组下，用户的评分总数和评分均值。

```
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
```

8.5.3 创建DataFrame

本文为您介绍如何创建DataFrame。

在使用DataFrame时，您需要了解关于这三类对象的操作，即`Collection`（`DataFrame`）、`Sequence`、`Scalar`。 这三种对象分别表示表结构（或者二维结构）、列（一维结构）、标量。这些对象仅在使用Pandas数据创建对象后，才会包含实际数据， 而在ODPS表上创建的对象中并不包含实际的数据，仅包含对这些数据的操作，实质的存储和计算会在ODPS中进行。

创建DataFrame

通常，您唯一需要直接创建的Collection对象是DataFrame，用于引用数据源，可能是一个ODPS表、ODPS分区、Pandas DataFrame或sqlalchemy Table（数据库表）。 使用这几种数据源时，相关的操作相同，这意味着您可以不更改数据处理的代码，仅修改输入、输出的指向， 便可以将在小数据量上运行的本地测试代码迁移到ODPS上，而迁移的正确性由PyODPS来保证。

创建DataFrame，只需将Table对象、pandas DataFrame对象或者sqlalchemy Table对象传入即可。

```
from odps.df import DataFrame

# 从ODPS表创建DataFrame。
 iris = DataFrame(o.get_table('pyodps_iris'))
 iris2 = o.get_table('pyodps_iris').to_df()  # 使用表的to_df方法。

# 从ODPS分区创建DataFrame。
 pt_df = DataFrame(o.get_table('partitioned_table').get_partition('pt=
20171111'))
 pt_df2 = o.get_table('partitioned_table').get_partition('pt=20171111
').to_df()  # 使用分区的to_df方法。

# 从Pandas DataFrame创建DataFrame。
 import pandas as pd
 import numpy as np
 df = DataFrame(pd.DataFrame(np.arange(9).reshape(3, 3), columns=list
('abc')))

# 从sqlalchemy Table创建DataFrame。
 engine = sqlalchemy.create_engine('mysql://root:123456@localhost/
movielens')
 metadata = sqlalchemy.MetaData(bind=engine) # 需要绑定到engine。
 table = sqlalchemy.Table('top_users', metadata, extend_existing=True
, autoload=True)
 users = DataFrame(table)
```

在用Pandas DataFrame初始化时，对于NUMPY OBJECT类型或者STRING类型，PyODPS DataFrame会尝试进行类型的推断， 如果一整列都为空，则会报错。为避免报错，您可以通过指定`unknown_as_string`为True，将这些列指定为STRING类型。您也可以指定`as_type`参数，进行强制类型转换。如果类型为基本类型，会在创建PyODPS DataFrame时进行强制类型转换。如果Pandas DataFrame中包含LIST或者DICT列，该列的类型不会被推断，必须手动使用`as_type`指定。`as_type`参数类型必须是DICT。

```
df2 = DataFrame(df, unknown_as_string=True, as_type={'null_col2': '
float'})
df2.dtypes
odps.Schema {
  sepallength           float64
  sepalwidth            float64
  petallength           float64
  petalwidth            float64
  name                  string
  null_col1             string   # 无法识别，通过unknown_as_string设置成
string类型。
```

```
  null_col2              float64   # 强制转换成float类型。
}
 df4 = DataFrame(df3, as_type={'list_col': 'list<int64>'})
 df4.dtypes
odps.Schema {
  id          int64
  list_col    list<int64>   # 无法识别且无法自动转换，通过as_type设置。
}
```

说明:

PyODPS不支持上传外部表（OSS/OTS）。

8.5.4 Sequence

SequenceExpr代表二维数据集中的一列。SequenceExpr只可以从一个Collection中获取，不支持手动创建SequenceExpr。

获取列

您可以使用`collection.column_name`取出一列。

```
iris.sepallength.head(5)
   sepallength
0          5.1
1          4.9
2          4.7
3          4.6
4          5.0
```

或者使用`df[column_name]`的形式实现（前提是列名存储在一个字符串变量中）。

```
iris['sepallength'].head(5)
   sepallength
0          5.1
1          4.9
2          4.7
3          4.6
4          5.0
```

列类型

DataFrame包括自己的类型系统，在使用Table初始化的时候，ODPS的类型会被转换。这样做的好处是可以支持更多的计算后端。目前，DataFrame的执行后端支持ODPS SQL、Pandas和数据库（MySQL和Postgres）。

PyODPS DataFrame包括的类型为`INT8`、`INT16`、`INT32`、`INT64`、`FLOAT32`、`FLOAT64`、`BOOLEAN`、`STRING`、`DECIMAL`、`DATETIME`、`LIST`、`DICT`。

ODPS的字段和DataFrame的类型映射关系如下。

ODPS类型	DataFrame类型
BIGINT	INT64
DOUBLE	FLOAT64
STRING	STRING
DATETIME	DATETIME
BOOLEAN	BOOLEAN
DECIMAL	DECIMAL
ARRAY<VALUE_TYPE>	LIST<VALUE_TYPE>
MAP<KEY_TYPE, VALUE_TYPE>	DICT<KEY_TYPE, VALUE_TYPE>

LIST和DICT必须填写其包含值的类型，否则会报错。目前DataFrame暂不支持MaxCompute 2.0中新增的TIMESTAMP及STRUCT类型，后续版本将会陆续支持新增的类型。

在Sequence中可以通过`sequence.dtype`获取数据类型。

```
iris.sepallength.dtype
float64
```

如果要修改一列的类型，可以使用`astype`方法。该方法输入一个类型，并返回类型转换后的Sequence。

```
iris.sepallength.astype('int')
   sepallength
0            5
1            4
2            4
3            4
4            5
```

列名

在DataFrame的计算过程中，每个Sequence必须要有列名。通常，DataFrame会为每个Sequence起一个名字。

```
iris.groupby('name').sepalwidth.max()
   sepalwidth_max
0             4.4
1             3.4
2             3.8
```

上述示例中，`sepalwidth`取最大值后被命名为`sepalwidth_max`。除部分操作外（例如在给一个Sequence加上一个Scalar时，结果会自动被命名为这个Sequence的名字），为Sequence命名的操作需要您自己手动完成。

Sequence提供`rename`方法对一列进行重命名。

```
iris.sepalwidth.rename('sepal_width').head(5)
   sepal_width
0          3.5
1          3.0
2          3.2
3          3.1
4          3.6
```

简单的列变换

您可以对一个Sequence进行运算，返回一个新的Sequence，这种操作类似于对简单的Python变量进行运算。Sequence支持对数值列进行四则运算，而对字符串则仅支持字符串的相加操作。

```
(iris.sepallength + 5).head(5)
   sepallength
0         10.1
1          9.9
2          9.7
3          9.6
4         10.0
```

```
(iris.sepallength + iris.sepalwidth).rename('sum_sepal').head(5)
   sum_sepal
0        8.6
1        7.9
2        7.9
3        7.7
4        8.6
```

两列共同参与运算时，PyODPS无法确定最终显示的列名，需要您手动指定。详细的列变换说明，请参见列运算。

8.5.5 Collection

DataFrame中所有二维数据集上的操作都属于CollectionExpr，可视为一张ODPS表或一张电子表单，DataFrame对象也是CollectionExpr的特例。CollectionExpr中包含针对二维数据集的列操作、筛选、变换等大量操作。

获取类型

`dtypes`可以用来获取CollectionExpr中所有列的类型。`dtypes`返回的是Schema类型 。

```
iris.dtypes
odps.Schema {
  sepallength           float64
  sepalwidth            float64
  petallength           float64
  petalwidth            float64
  name                  string
```

```
}
```

列选择和增删

- 列选择

  如果您需要从一个CollectionExpr中选取部分列，产生新的数据集，可以使用`expr[columns]`语法。

```
iris['name', 'sepallength'].head(5)
          name  sepallength
0  Iris-setosa          5.1
1  Iris-setosa          4.9
2  Iris-setosa          4.7
3  Iris-setosa          4.6
4  Iris-setosa          5.0
```

  说明:

  如果需要选择的列只有一列，需要在Columns后加上逗号或者显示标记为列表，例如`iris[iris.sepal_length, ]`或`iris[[iris.sepal_length]]`，否则返回的将是一个Sequence对象，而不是Collection。

- 列删除

  如果您需要在新的数据集中排除已有数据集的某些列，可使用`exclude`方法。

```
iris.exclude('sepallength', 'petallength')[:5]
   sepalwidth  petalwidth         name
0         3.5         0.2  Iris-setosa
1         3.0         0.2  Iris-setosa
2         3.2         0.2  Iris-setosa
3         3.1         0.2  Iris-setosa
4         3.6         0.2  Iris-setosa
```

  PyODPS 0.7.2版（及以上版本）支持另一种写法，即在数据集上直接排除相应的列。

```
del iris['sepallength']
del iris['petallength']
iris[:5]
   sepalwidth  petalwidth         name
0         3.5         0.2  Iris-setosa
1         3.0         0.2  Iris-setosa
2         3.2         0.2  Iris-setosa
3         3.1         0.2  Iris-setosa
```

```
4          3.6          0.2   Iris-setosa
```

- 列增加

  如果您需要在已有Collection中添加某一列变换的结果，可以使用`expr[expr, new_sequence]`语法， 新增的列会作为新Collection的一部分。

  以下示例将`iris`中的`sepalwidth`列加一后重命名为`sepalwidthplus1`并追加到数据集末尾，形成新的数据集。

```
iris[iris, (iris.sepalwidth + 1).rename('sepalwidthplus1')].head(5)
   sepallength  sepalwidth  petallength  petalwidth         name  \
0          5.1         3.5          1.4         0.2  Iris-setosa
1          4.9         3.0          1.4         0.2  Iris-setosa
2          4.7         3.2          1.3         0.2  Iris-setosa
3          4.6         3.1          1.5         0.2  Iris-setosa
4          5.0         3.6          1.4         0.2  Iris-setosa

   sepalwidthplus1
0              4.5
1              4.0
2              4.2
3              4.1
4              4.6
```

  使用`expr[expr, new_sequence]`语法需要注意的是，变换后的列名与原列名可能相同。如果需要与原Collection合并， 请将该列重命名。

  PyODPS 0.7.2版（及以上版本）支持直接在当前数据集中追加。

```
iris['sepalwidthplus1'] = iris.sepalwidth + 1
iris.head(5)
   sepallength  sepalwidth  petallength  petalwidth         name  \
0          5.1         3.5          1.4         0.2  Iris-setosa
1          4.9         3.0          1.4         0.2  Iris-setosa
2          4.7         3.2          1.3         0.2  Iris-setosa
3          4.6         3.1          1.5         0.2  Iris-setosa
4          5.0         3.6          1.4         0.2  Iris-setosa

   sepalwidthplus1
0              4.5
1              4.0
2              4.2
3              4.1
4              4.6
```

- 同时增删列

  您也可以先将原列通过`exclude`方法进行排除，再将变换后的新列并入，而不必担心重名。

```
iris[iris.exclude('sepalwidth'), iris.sepalwidth * 2].head(5)
   sepallength  petallength  petalwidth         name  sepalwidth
0          5.1          1.4         0.2  Iris-setosa         7.0
1          4.9          1.4         0.2  Iris-setosa         6.0
2          4.7          1.3         0.2  Iris-setosa         6.4
3          4.6          1.5         0.2  Iris-setosa         6.2
```

```
4          5.0          1.4          0.2  Iris-setosa          7.2
```

PyODPS 0.7.2版（及以上版本）支持直接在当前数据集上覆盖的操作，写法如下。

```
iris['sepalwidth'] = iris.sepalwidth * 2
iris.head(5)
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.1         7.0          1.4         0.2  Iris-setosa
1          4.9         6.0          1.4         0.2  Iris-setosa
2          4.7         6.4          1.3         0.2  Iris-setosa
3          4.6         6.2          1.5         0.2  Iris-setosa
4          5.0         7.2          1.4         0.2  Iris-setosa
```

增删列以创建新Collection的另一种实现方式是调用`select`方法，将需要选择的列作为参数输入。如果您需要重命名，可以使用`keyword`参数输入，并将新的列名作为参数名。

```
iris.select('name', sepalwidthminus1=iris.sepalwidth - 1).head(5)
          name  sepalwidthminus1
0  Iris-setosa               2.5
1  Iris-setosa               2.0
2  Iris-setosa               2.2
3  Iris-setosa               2.1
4  Iris-setosa               2.6
```

您也可以传入一个Lambda表达式，它接收的参数为上一步的运算结果。在执行时，PyODPS会检查这些Lambda表达式，传入上一步生成的Collection并将其替换为正确的列。

```
iris['name', 'petallength'][[lambda x: x.name]].head(5)
          name
0  Iris-setosa
1  Iris-setosa
2  Iris-setosa
3  Iris-setosa
4  Iris-setosa
```

PyODPS 0.7.2版（及以上版本）支持对数据进行条件赋值。

```
iris[iris.sepallength > 5.0, 'sepalwidth'] = iris.sepalwidth * 2
iris.head(5)
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.1        14.0          1.4         0.2  Iris-setosa
1          4.9         6.0          1.4         0.2  Iris-setosa
2          4.7         6.4          1.3         0.2  Iris-setosa
3          4.6         6.2          1.5         0.2  Iris-setosa
4          5.0         7.2          1.4         0.2  Iris-setosa
```

引入常数和随机数

- DataFrame支持在Collection中追加一列常数。追加常数需要使用 Scalar，引入时需要手动指定列名。

```
from odps.df import Scalar
iris[iris, Scalar(1).rename('id')][:5]
   sepallength  sepalwidth  petallength  petalwidth         name  id
```

```
0           5.1          3.5           1.4          0.2  Iris-setosa   1
1           4.9          3.0           1.4          0.2  Iris-setosa   1
2           4.7          3.2           1.3          0.2  Iris-setosa   1
3           4.6          3.1           1.5          0.2  Iris-setosa   1
4           5.0          3.6           1.4          0.2  Iris-setosa   1
```

如果需要指定一个空值列，您可以使用NullScalar，但需要提供字段类型。

```
from odps.df import NullScalar
iris[iris, NullScalar('float').rename('fid')][:5]
   sepal_length  sepal_width  petal_length  petal_width     category
   fid
0           5.1          3.5           1.4          0.2  Iris-setosa
  None
1           4.9          3.0           1.4          0.2  Iris-setosa
  None
2           4.7          3.2           1.3          0.2  Iris-setosa
  None
3           4.6          3.1           1.5          0.2  Iris-setosa
  None
4           5.0          3.6           1.4          0.2  Iris-setosa
  None
```

在PyODPS 0.7.12（及以上版本）中，引入了简化写法。

```
iris['id'] = 1
iris
   sepallength  sepalwidth  petallength  petalwidth         name  id
0           5.1          3.5           1.4          0.2  Iris-setosa   1
1           4.9          3.0           1.4          0.2  Iris-setosa   1
2           4.7          3.2           1.3          0.2  Iris-setosa   1
3           4.6          3.1           1.5          0.2  Iris-setosa   1
4           5.0          3.6           1.4          0.2  Iris-setosa   1
```

需要注意的是，这种写法无法自动识别空值的类型，所以在增加空值列时，仍然要使用如下代码。

```
iris['null_col'] = NullScalar('float')
iris
   sepallength  sepalwidth  petallength  petalwidth         name
null_col
0           5.1          3.5           1.4          0.2  Iris-setosa
  None
1           4.9          3.0           1.4          0.2  Iris-setosa
  None
2           4.7          3.2           1.3          0.2  Iris-setosa
  None
3           4.6          3.1           1.5          0.2  Iris-setosa
  None
4           5.0          3.6           1.4          0.2  Iris-setosa
  None
```

- DataFrame也支持在Collection中增加一列随机数列，该列类型为FLOAT，范围为0~1，每行数值均不同。 追加随机数列需要使用RandomScalar，参数为随机数种子，可省略。

```
from odps.df import RandomScalar
iris[iris, RandomScalar().rename('rand_val')][:5]
   sepallength  sepalwidth  petallength  petalwidth         name
rand_val
```

```
0           5.1          3.5           1.4          0.2   Iris-setosa  0.
000471
1           4.9          3.0           1.4          0.2   Iris-setosa  0.
799520
2           4.7          3.2           1.3          0.2   Iris-setosa  0.
834609
3           4.6          3.1           1.5          0.2   Iris-setosa  0.
106921
4           5.0          3.6           1.4          0.2   Iris-setosa  0.
763442
```

过滤数据

Collection提供了数据过滤的功能。查询sepallength大于5的数据，示例如下。

```
iris[iris.sepallength > 5].head(5)
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.1         3.5          1.4         0.2  Iris-setosa
1          5.4         3.9          1.7         0.4  Iris-setosa
2          5.4         3.7          1.5         0.2  Iris-setosa
3          5.8         4.0          1.2         0.2  Iris-setosa
4          5.7         4.4          1.5         0.4  Iris-setosa
```

多种查询方式的示例：

- 与（&）条件

```
iris[(iris.sepallength < 5) & (iris['petallength'] > 1.5)].head(5)
   sepallength  sepalwidth  petallength  petalwidth             name
0          4.8         3.4          1.6         0.2      Iris-setosa
1          4.8         3.4          1.9         0.2      Iris-setosa
2          4.7         3.2          1.6         0.2      Iris-setosa
3          4.8         3.1          1.6         0.2      Iris-setosa
4          4.9         2.4          3.3         1.0  Iris-versicolor
```

- 或（|）条件

```
iris[(iris.sepalwidth < 2.5) | (iris.sepalwidth > 4)].head(5)
   sepallength  sepalwidth  petallength  petalwidth             name
0          5.7         4.4          1.5         0.4      Iris-setosa
1          5.2         4.1          1.5         0.1      Iris-setosa
2          5.5         4.2          1.4         0.2      Iris-setosa
3          4.5         2.3          1.3         0.3      Iris-setosa
4          5.5         2.3          4.0         1.3  Iris-versicolor
```

说明：

与和或条件必须使用&和|，不能使用and和or。

- 非（~）条件

```
iris[~(iris.sepalwidth > 3)].head(5)
   sepallength  sepalwidth  petallength  petalwidth         name
0          4.9         3.0          1.4         0.2  Iris-setosa
1          4.4         2.9          1.4         0.2  Iris-setosa
2          4.8         3.0          1.4         0.1  Iris-setosa
3          4.3         3.0          1.1         0.1  Iris-setosa
```

```
4          5.0          3.0           1.6          0.2   Iris-setosa
```

- 显式调用filter方法，提供多个与条件。

```
iris.filter(iris.sepalwidth > 3.5, iris.sepalwidth < 4).head(5)
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.0         3.6          1.4         0.2  Iris-setosa
1          5.4         3.9          1.7         0.4  Iris-setosa
2          5.4         3.7          1.5         0.2  Iris-setosa
3          5.4         3.9          1.3         0.4  Iris-setosa
4          5.7         3.8          1.7         0.3  Iris-setosa
```

- 对于连续的操作，使用Lambda表达式。

```
iris[iris.sepalwidth > 3.8]['name', lambda x: x.sepallength + 1]
          name  sepallength
0  Iris-setosa          6.4
1  Iris-setosa          6.8
2  Iris-setosa          6.7
3  Iris-setosa          6.4
4  Iris-setosa          6.2
5  Iris-setosa          6.5
```

- 对于Collection，如果它包含一个列是BOOLEAN类型，则可以直接使用该列作为过滤条件。

```
df.dtypes
odps.Schema {
  a boolean
  b int64
}
df[df.a]
      a  b
0  True  1
1  True  3
```

因此，对Collection取单个Sequence的操作时，只有BOOLEAN列是合法的，即对Collection作过滤操作。

```
df[df.a, ]        # 取列操作。
df[[df.a]]        # 取列操作。
df.select(df.a)   # 显式取列。
df[df.a]          # a列是boolean列，执行过滤操作。
df.a              # 取单列。
df['a']           # 取单列。
```

- 同时，您可以使用Pandas中的query方法，通过查询语句做数据的筛选，在表达式中直接使用列名如sepallength进行操作。在查询语句中，&和and都表示与操作，|和or都表示或操作。

```
iris.query("(sepallength < 5) and (petallength > 1.5)").head(5)
   sepallength  sepalwidth  petallength  petalwidth             name
0          4.8         3.4          1.6         0.2      Iris-setosa
1          4.8         3.4          1.9         0.2      Iris-setosa
2          4.7         3.2          1.6         0.2      Iris-setosa
3          4.8         3.1          1.6         0.2      Iris-setosa
```

```
4          4.9          2.4           3.3          1.0   Iris-versicolor
```

当表达式中需要使用到本地变量时，需要在该变量前加一个@前缀。

```
var = 4
iris.query("(iris.sepalwidth < 2.5) | (sepalwidth > @var)").head(5)
   sepallength  sepalwidth  petallength  petalwidth            name
0          5.7          4.4          1.5          0.4      Iris-setosa
1          5.2          4.1          1.5          0.1      Iris-setosa
2          5.5          4.2          1.4          0.2      Iris-setosa
3          4.5          2.3          1.3          0.3      Iris-setosa
4          5.5          2.3          4.0          1.3  Iris-versicolor
```

目前query支持的语法，如下表所示。

语法	说明
name	没有@前缀的都当做列名处理，有前缀的会获取本地变量。
operator	支持部分运算符：+、-、*、/、//、%、**、==、!=、<、<=、>、>=、in、not in。
bool	与或非操作，其中&和and表示与，|和or表示或。
attribute	取对象属性。
index, slice, subscript	切片操作。

并列多行输出

对于LIST及MAP类型的列，explode方法会将该列转换为多行输出。您也可以使用apply方法实现多行输出。为了进行聚合等操作，常常需要将这些输出和原表中的列合并。此时可以使用DataFrame提供的并列多行输出功能， 写法为将多行输出函数生成的集合与原集合中的列名一起映射。

并列多行输出的示例如下。

```
df
   id          a              b
0   1  [a1, b1]  [a2, b2, c2]
1   2      [c1]      [d2, e2]
df[df.id, df.a.explode(), df.b]
   id   a              b
0   1  a1  [a2, b2, c2]
1   1  b1  [a2, b2, c2]
2   2  c1      [d2, e2]
df[df.id, df.a.explode(), df.b.explode()]
   id   a   b
0   1  a1  a2
1   1  a1  b2
2   1  a1  c2
3   1  b1  a2
```

```
4   1  b1  b2
5   1  b1  c2
6   2  c1  d2
7   2  c1  e2
```

如果多行输出方法对某个输入不产生任何输出，默认输入行将不在最终结果中出现。如果需要在结果中出现该行，可以设置`keep_nulls=True`。此时，与该行并列的值将输出为空值。

```
df
   id         a
0   1  [a1, b1]
1   2        []
df[df.id, df.a.explode()]
   id   a
0   1  a1
1   1  b1
df[df.id, df.a.explode(keep_nulls=True)]
   id     a
0   1    a1
1   1    b1
2   2  None
```

使用`explode`方法实现并列多行输出的示例，请参见集合类型相关操作，使用`apply`方法实现并列多行输出的示例，请参见对一行数据使用自定义函数。

限制条数

```
iris[:3]
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.1         3.5          1.4         0.2  Iris-setosa
1          4.9         3.0          1.4         0.2  Iris-setosa
2          4.7         3.2          1.3         0.2  Iris-setosa
```

目前对于ODPS SQL，后端切片不支持`start`和`step`方法，但支持`limit`方法。

```
iris.limit(3)
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.1         3.5          1.4         0.2  Iris-setosa
1          4.9         3.0          1.4         0.2  Iris-setosa
2          4.7         3.2          1.3         0.2  Iris-setosa
```

说明:

切片操作只能作用于Collection，不能作用于Sequence。

8.5.6 执行

本文为您介绍DataFrame操作支持的执行方法。

延迟执行

DataFrame上的所有操作并不会立即执行，只有当显式调用`execute`方法，或者调用立即执行的方法时（内部调用的就是`execute`），才会执行这些操作。这些立即执行的方法如下表所示。

方法	说明	返回值
persist	将执行结果保存到ODPS表。	PyODPS DataFrame
execute	执行并返回全部结果。	ResultFrame
head	查看开头N行数据，这个方法会执行所有结果，并取开头N行数据。	ResultFrame
tail	查看结尾N行数据，这个方法会执行所有结果，并取结尾N行数据。	ResultFrame
to_pandas	转化为Pandas DataFrame或者Series，wrap参数为True的时候，返回PyODPS DataFrame对象。	wrap为True时，返回PyODPS DataFrame；wrap为False（默认）时，返回Pandas DataFrame。
plot，hist，boxplot	画图有关。	无。

说明:

在交互式环境下，PyODPS DataFrame会在打印或者`repr`的时候，调用`execute`方法，您无需手动调用`execute`。

```
>>> iris[iris.sepallength < 5][:5]
   sepallength  sepalwidth  petallength  petalwidth         name
0          4.9         3.0          1.4         0.2  Iris-setosa
1          4.7         3.2          1.3         0.2  Iris-setosa
2          4.6         3.1          1.5         0.2  Iris-setosa
3          4.6         3.4          1.4         0.3  Iris-setosa
4          4.4         2.9          1.4         0.2  Iris-setosa
```

如果您需要关闭自动调用执行，请进行手动设置。

```
>>> from odps import options
>>> options.interactive = False
>>>
>>> iris[iris.sepallength < 5][:5]
Collection: ref_0
  odps.Table
    name: odps_test_sqltask_finance.`pyodps_iris`
    schema:
      sepallength           : double
      sepalwidth            : double
      petallength           : double
      petalwidth            : double
      name                  : string

Collection: ref_1
  Filter[collection]
    collection: ref_0
    predicate:
      Less[sequence(boolean)]
```

```
        sepallength = Column[sequence(float64)] 'sepallength' from
collection ref_0
        Scalar[int8]
          5

Slice[collection]
  collection: ref_1
  stop:
    Scalar[int8]
      5
```

此时打印repr对象，会显示整个抽象语法树。

说明:

ResultFrame是结果集合，不能参与后续计算。

ResultFrame可以迭代取出每条记录。

```
>>> result = iris.head(3)
>>> for r in result:
>>>     print(list(r))
[5.0999999999999996, 3.5, 1.3999999999999999, 0.20000000000000001, u'
Iris-setosa']
[4.9000000000000004, 3.0, 1.3999999999999999, 0.20000000000000001, u'
Iris-setosa']
[4.7000000000000002, 3.2000000000000002, 1.3, 0.20000000000000001, u'
Iris-setosa']
```

ResultFrame也支持在安装有Pandas的前提下转换为Pandas DataFrame，或使用Pandas后端的PyODPS DataFrame。

```
pd_df = iris.head(3).to_pandas()  # 返回Pandas DataFrame。
 wrapped_df = iris.head(3).to_pandas(wrap=True)  # 返回使用Pandas后端的
PyODPS DataFrame。
```

保存执行结果为ODPS表

对于Collection，您可以调用persist方法，参数为表名，用于返回一个新的DataFrame对象。

```
>>> iris2 = iris[iris.sepalwidth < 2.5].persist('pyodps_iris2')
>>> iris2.head(5)
   sepallength  sepalwidth  petallength  petalwidth            name
0          4.5         2.3          1.3         0.3     Iris-setosa
1          5.5         2.3          4.0         1.3 Iris-versicolor
2          4.9         2.4          3.3         1.0 Iris-versicolor
3          5.0         2.0          3.5         1.0 Iris-versicolor
4          6.0         2.2          4.0         1.0 Iris-versicolor
```

persist可以传入partitions参数，用于创建一个分区表。它的分区是partitions所指定的字段。

```
iris3 = iris[iris.sepalwidth < 2.5].persist('pyodps_iris3', partitions
=['name'])
 iris3.data
odps.Table
```

```
  name: odps_test_sqltask_finance.`pyodps_iris3`
  schema:
    sepallength           : double
    sepalwidth            : double
    petallength           : double
    petalwidth            : double
  partitions:
    name                  : string
```

如果您需要写入已经存在的表的某个分区，persist可以传入partition参数，指明写入表的哪个分区（例如ds=******）。该DataFrame的每个字段的类型都必须相同，且都存在于该表中。drop_partition和create_partition参数只在此时有效，分别表示是否要删除（如果分区存在）或创建（如果分区不存在）该分区。

```
iris[iris.sepalwidth < 2.5].persist('pyodps_iris4', partition='ds=test
', drop_partition=True, create_partition=True)
```

写入表时，您还可以指定表的生命周期。例如下列语句将表的生命周期指定为10天。

```
iris[iris.sepalwidth < 2.5].persist('pyodps_iris5', lifecycle=10)
```

如果数据源中没有ODPS对象，例如数据源仅为Pandas，在persist时需要手动指定ODPS入口对象，或者将需要的入口对象标明为全局对象。

```
# 假设入口对象为o。
# 指定入口对象。
df.persist('table_name', odps=o)
# 或者可将入口对象标记为全局。
o.to_global()
df.persist('table_name')
```

保存执行结果为Pandas DataFrame

您可以使用to_pandas方法，如果wrap参数为True，将返回PyODPS DataFrame对象。

```
type(iris[iris.sepalwidth < 2.5].to_pandas())
pandas.core.frame.DataFrame
type(iris[iris.sepalwidth < 2.5].to_pandas(wrap=True))
odps.df.core.DataFrame
```

将表中的数据读出，PyODPS可以执行open_reader方法，通过reader.to_pandas()转成Pandas DataFrame。

立即运行设置运行参数

对于立即执行的方法，例如execute、persist、to_pandas等，您可以通过以下方法设置它们运行时的参数（仅对ODPS SQL后端有效）：

- 设置全局参数。详情请参考SQL设置运行参数 。

- 在这些立即执行的方法上，使用hints参数，可以确保这些参数只作用于当前的计算过程。

```
iris[iris.sepallength < 5].to_pandas(hints={'odps.sql.mapper.split.
size': 16})
```

运行时显示详细信息

如果您需要查看运行时Instance的Logview，则应该修改全局配置。

```
from odps import options
 options.verbose = True

 iris[iris.sepallength < 5].exclude('sepallength')[:5].execute()
Sql compiled:
SELECT t1.`sepalwidth`, t1.`petallength`, t1.`petalwidth`, t1.`name`
FROM odps_test_sqltask_finance.`pyodps_iris` t1
WHERE t1.`sepallength` < 5
LIMIT 5
logview:
http://logview

   sepalwidth  petallength  petalwidth         name
0         3.0          1.4         0.2  Iris-setosa
1         3.2          1.3         0.2  Iris-setosa
2         3.1          1.5         0.2  Iris-setosa
3         3.4          1.4         0.3  Iris-setosa
4         2.9          1.4         0.2  Iris-setosa
```

您可以指定自己的日志记录函数。

```
my_logs = []
 def my_logger(x):
     my_logs.append(x)

 options.verbose_log = my_logger

 iris[iris.sepallength < 5].exclude('sepallength')[:5].execute()
   sepalwidth  petallength  petalwidth         name
0         3.0          1.4         0.2  Iris-setosa
1         3.2          1.3         0.2  Iris-setosa
2         3.1          1.5         0.2  Iris-setosa
3         3.4          1.4         0.3  Iris-setosa
4         2.9          1.4         0.2  Iris-setosa

 print(my_logs)
['Sql compiled:', 'SELECT t1.`sepalwidth`, t1.`petallength`, t1.`
petalwidth`, t1.`name` \nFROM odps_test_sqltask_finance.`pyodps_iri
s` t1 \nWHERE t1.`sepallength` < 5 \nLIMIT 5', 'logview:', u'http://
logview']
```

缓存中间Collection计算结果

DataFrame的计算过程中，部分Collection被多处使用。如果您需要查看中间过程的执行结果，可以使用cache标记某个需要被优先计算的Collection。

说明:

cache会延迟执行，调用cache不会触发立即计算。

```
cached = iris[iris.sepalwidth < 3.5].cache()
 df = cached['sepallength', 'name'].head(3)
 df
   sepallength         name
0          4.9  Iris-setosa
1          4.7  Iris-setosa
2          4.6  Iris-setosa
 cached.head(3)  # 由于cached已经被计算，所以能立刻取到计算结果。
   sepallength         name
0          4.9  Iris-setosa
1          4.7  Iris-setosa
2          4.6  Iris-setosa
```

异步和并行执行

DataFrame支持异步操作，对于立即执行的方法，包括execute、persist、head、tail、to_pandas（其他方法不支持），传入async参数，即可以将这个操作异步执行，timeout参数指定超时时间，异步返回的是 Future对象。

```
from odps.df import Delay
 delay = Delay()  # 创建Delay对象。

 df = iris[iris.sepal_width < 5].cache()  # 有一个共同的依赖。
 future1 = df.sepal_width.sum().execute(delay=delay)  # 立即返回future对象，此时并没有执行。
 future2 = df.sepal_width.mean().execute(delay=delay)
 future3 = df.sepal_length.max().execute(delay=delay)
 delay.execute(n_parallel=3)  # 并发度是3，此时才真正执行。
|========================================|   1 /  1  (100.00%)
  21s
 future1.result()
 458.10000000000014
 future2.result()
3.0540000000000007
```

上述示例中，共同依赖的对象会先执行，然后再以并发度为3分别执行future1到future3。当n_parallel为1时，执行时间会达到37s。

delay.execute也接受async操作指定是否异步执行，当异步执行的时候，也可以用timeout参数指定超时时间。

8.5.7 列运算

本文为您介绍DataFrame API中的列运算。

列运算

```
from odps.df import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
lens = DataFrame(o.get_table('pyodps_ml_100k_lens'))
```

为一个Sequence加上一个常量或执行sin函数时，这些操作将作用于Sequence中的每个元素。

NULL相关（isnull，notnull，fillna）

DataFrame API提供了几个和NULL相关的内置函数，例如`isnull`用于判断某字段是否为NULL，`notnull`用于判断某字段是否为非NULL，`fillna`用于将NULL填充为您指定的值。

```
>>> iris.sepallength.isnull().head(5)
   sepallength
0        False
1        False
2        False
3        False
4        False
```

逻辑判断（ifelse，switch）

`ifelse`作用于BOOLEAN类型的字段，当条件成立时，返回第0个参数，否则返回第1个参数。

```
>>> (iris.sepallength > 5).ifelse('gt5', 'lte5').rename('cmp5').head(5
)
   cmp5
0   gt5
1  lte5
2  lte5
3  lte5
4  lte5
```

`switch`用于多条件判断的情况。

```
>>> iris.sepallength.switch(4.9, 'eq4.9', 5.0, 'eq5.0', default='noeq
').rename('equalness').head(5)
   equalness
0       noeq
1      eq4.9
2       noeq
3       noeq
4      eq5.0
```

```
>>> from odps.df import switch
>>> switch(iris.sepallength == 4.9, 'eq4.9', iris.sepallength == 5.0,
 'eq5.0', default='noeq').rename('equalness').head(5)
   equalness
0       noeq
1      eq4.9
2       noeq
3       noeq
4      eq5.0
```

PyODPS 0.7.8以上版本支持根据条件修改数据集某一列的一部分值，写法如下。

```
>>> iris[iris.sepallength > 5, 'cmp5'] = 'gt5'
>>> iris[iris.sepallength <= 5, 'cmp5'] = 'lte5'
>>> iris.head(5)
   cmp5
0   gt5
1  lte5
2  lte5
3  lte5
```

```
4  lte5
```

数学运算

对于数字类型的字段，支持加法（+）、减法（-）、乘法（*）和除法（/）等操作，也支持log、sin等数学计算。

```
>>> (iris.sepallength * 10).log().head(5)
   sepallength
0     3.931826
1     3.891820
2     3.850148
3     3.828641
4     3.912023
```

```
>>> fields = [iris.sepallength,
>>>           (iris.sepallength / 2).rename('sepallength除以2'),
>>>           (iris.sepallength ** 2).rename('sepallength的平方')]
>>> iris[fields].head(5)
   sepallength  sepallength除以2  sepallength的平方
0          5.1              2.55             26.01
1          4.9              2.45             24.01
2          4.7              2.35             22.09
3          4.6              2.30             21.16
4          5.0              2.50             25.00
```

算术运算支持如下操作。

算术操作	说明
abs	绝对值。
sqrt	平方根。
sin	无。
sinh	无。
cos	无。
cosh	无。
tan	无。
tanh	无。
arccos	无。
arccosh	无。
arcsin	无。
arcsinh	无。
arctan	无。
arctanh	无。
exp	指数函数。

算术操作	说明
expm1	指数减1。
log	传入参数表示底是几。
log2	无。
log10	无。
log1p	log(1+x)。
radians	给定角度计算弧度。
degrees	给定弧度计算角度。
ceil	不小于输入值的最小整数。
floor	向下取整，返回比输入值小的整数值。
trunc	将输入值截取到指定小数点位置。

Sequence也支持自身与其它Sequence或Scalar的比较。

```
>>> (iris.sepallength < 5).head(5)
   sepallength
0        False
1         True
2         True
3         True
4        False
```

虽然DataFrame API不支持连续操作，例如`3 <= iris.sepallength <= 5`，但是`between`函数可以用于判断`iris.sepallength`是否在某个区间。

```
>>> (iris.sepallength.between(3, 5)).head(5)
   sepallength
0        False
1         True
2         True
3         True
4         True
```

默认情况下，`between`包含两边的区间，如果计算开区间，则需要设置`inclusive=False`。

```
>>> (iris.sepallength.between(3, 5, inclusive=False)).head(5)
   sepallength
0        False
1         True
2         True
3         True
```

```
4        False
```

STRING相关操作

DataFrame API提供了一系列针对STRING类型的Sequence或者Scalar的操作。

```
>>> fields = [
>>>     iris.name.upper().rename('upper_name'),
>>>     iris.name.extract('Iris(.*)', group=1)
>>> ]
>>> iris[fields].head(5)
    upper_name     name
0  IRIS-SETOSA  -setosa
1  IRIS-SETOSA  -setosa
2  IRIS-SETOSA  -setosa
3  IRIS-SETOSA  -setosa
4  IRIS-SETOSA  -setosa
```

STRING的相关操作如下。

STRING操作	说明
capitalize	无。
contains	包含某个字符串，如果`regex`参数为True，则是包含某个正则表达式，默认为True。
count	指定字符串出现的次数。
endswith	以某个字符串结尾。
startswith	以某个字符串开头。
extract	抽取出某个正则表达式，如果group不指定，则返回满足整个pattern的子串；否则，返回第几个group。
find	返回第一次出现的子串位置，若不存在则返回-1。
rfind	从右查找返回子串第一次出现的位置，不存在则返回-1。
replace	将某个pattern的子串全部替换成另一个子串，n参数若指定，则替换n次。
get	返回某个位置上的字符串。
len	返回字符串的长度。
ljust	若未达到指定的`width`的长度，则在右侧填充`fillchar`指定的字符串（默认空格）。
rjust	若未达到指定的`width`的长度，则在左侧填充`fillchar`指定的字符串（默认空格）。

STRING操作	说明
lower	变为全部小写。
upper	变为全部大写。
lstrip	在左侧删除空格（包括空行符）。
rstrip	在右侧删除空格（包括空行符）。
strip	在左右两侧删除空格（包括空行符）。
split	将字符串按分隔符拆分为若干个字符串（返回list<string>类型）。
pad	在指定的位置（left，right或者both）用指定填充字符（用`fillchar`指定，默认空格）来对齐。
repeat	重复指定n次。
slice	切片操作。
swapcase	对调大小写。
title	同`str.title`。
zfill	长度没达到指定`width`，则左侧填充0。
isalnum	同`str.isalnum`。
isalpha	同`str.isalpha`。
isdigit	是否都是数字，同`str.isdigit`。
isspace	是否都是空格，同`str.isspace`。
islower	是否都是小写，同`str.islower`。
isupper	是否都是大写，同`str.isupper`。
istitle	同`str.istitle`。
isnumeric	同`str.isnumeric`。
isdecimal	同`str.isdecimal`。
todict	将字符串按分隔符拆分为一个Dict，传入的两个参数分别为项目分隔符和Key-Value分隔符（返回dict<string, string>类型）
strptime	按格式化读取成时间，时间格式和Python标准库相同，详细参考 Python 时间格式化

时间相关操作

对于DATETIME类型Sequence或者Scalar，可以调用时间相关的内置函数。

```
>>> df = lens[[lens.unix_timestamp.astype('datetime').rename('dt')]]
>>> df[df.dt,
>>>    df.dt.year.rename('year'),
>>>    df.dt.month.rename('month'),
>>>    df.dt.day.rename('day'),
>>>    df.dt.hour.rename('hour')].head(5)
                   dt  year  month  day  hour
0  1998-04-08 11:02:00  1998      4    8    11
1  1998-04-08 10:57:55  1998      4    8    10
2  1998-04-08 10:45:26  1998      4    8    10
3  1998-04-08 10:25:52  1998      4    8    10
4  1998-04-08 10:44:19  1998      4    8    10
```

与时间相关的属性如下。

时间相关属性	说明
year	无。
month	无。
day	无。
hour	无。
minute	无。
second	无。
weekofyear	返回日期位于那一年的第几周。周一作为一周的第一天。
weekday	返回日期当前周的第几天。
dayofweek	同weekday。
strftime	格式化时间，时间格式和Python标准库相同，详请请参见 Python 时间格式化

PyODPS也支持时间的加减操作，例如可以通过以下方法得到前3天的日期。两个日期列相减得到相差的毫秒数。

```
>>> df
                           a                          b
0 2016-12-06 16:43:12.460001 2016-12-06 17:43:12.460018
1 2016-12-06 16:43:12.460012 2016-12-06 17:43:12.460021
2 2016-12-06 16:43:12.460015 2016-12-06 17:43:12.460022
>>> from odps.df import day
>>> df.a - day(3)
                           a
0 2016-12-03 16:43:12.460001
1 2016-12-03 16:43:12.460012
2 2016-12-03 16:43:12.460015
```

```
>>> (df.b - df.a).dtype
int64
>>> (df.b - df.a).rename('a')
         a
0  3600000
1  3600000
2  3600000
```

支持的时间类型如下表所示。

属性	说明
year	无。
month	无。
day	无。
hour	无。
minute	无。
second	无。
millisecond	无。

集合类型相关操作

PyODPS支持的集合类型有List和Dict。这两个类型都可以使用下标获取集合中的某个项目。`len`方法用于获得集合的大小。

同时，两种集合均有`explode`方法，用于展开集合中的内容。对于List，`explode`默认返回一列，当传入参数`pos`时， 将返回两列，其中一列为值在数组中的编号（类似Python的`enumerate`函数）。对于Dict，`explode`会返回两列， 分别表示keys及values。`explode`中也可以传入列名，作为最后生成的列。

示例如下。

```
>>> df
   id         a                              b
0   1  [a1, b1]  {'a2': 0, 'b2': 1, 'c2': 2}
1   2      [c1]           {'d2': 3, 'e2': 4}
>>> df[df.id, df.a[0], df.b['b2']]
   id   a    b
0   1  a1    1
1   2  c1  NaN
>>> df[df.id, df.a.len(), df.b.len()]
   id  a  b
0   1  2  3
1   2  1  2
>>> df.a.explode()
    a
0  a1
1  b1
2  c1
>>> df.a.explode(pos=True)
```

```
   a_pos   a
0      0  a1
1      1  b1
2      0  c1
>>> # 指定列名。
>>> df.a.explode(['pos', 'value'], pos=True)
   pos value
0    0    a1
1    1    b1
2    0    c1
>>> df.b.explode()
  b_key  b_value
0    a2        0
1    b2        1
2    c2        2
3    d2        3
4    e2        4
>>> # 指定列名。
>>> df.b.explode(['key', 'value'])
  key  value
0  a2      0
1  b2      1
2  c2      2
3  d2      3
4  e2      4
```

`explode`也可以和并列多行输出结合，以将原有列和`explode`的结果相结合，示例如下。

```
>>> df[df.id, df.a.explode()]
   id   a
0   1  a1
1   1  b1
2   2  c1
>>> df[df.id, df.a.explode(), df.b.explode()]
   id   a b_key  b_value
0   1  a1    a2        0
1   1  a1    b2        1
2   1  a1    c2        2
3   1  b1    a2        0
4   1  b1    b2        1
5   1  b1    c2        2
6   2  c1    d2        3
7   2  c1    e2        4
```

除了下标、`len`和`explode`两个共有方法以外，List还支持下列方法。

List 操作	说明
contains(v)	列表是否包含某个元素。
sort	返回排序后的列表（返回值为List）。

Dict还支持下列方法。

dict操作	说明
keys	获取Dict keys（返回值为List）。
values	获取Dict values（返回值为List）。

其它元素操作（isin，notin，cut）

isin用于判断Sequence里的元素是否在某个集合元素里；notin反之。

```
>>> iris.sepallength.isin([4.9, 5.1]).rename('sepallength').head(5)
   sepallength
0         True
1         True
2        False
3        False
4        False
```

cut提供离散化的操作，可以将Sequence的数据拆成几个区段。

```
>>> iris.sepallength.cut(range(6), labels=['0-1', '1-2', '2-3', '3-4',
 '4-5']).rename('sepallength_cut').head(5)
   sepallength_cut
0             None
1              4-5
2              4-5
3              4-5
4              4-5
```

include_under和include_over可以分别包括向下和向上的区间。

```
>>> labels = ['0-1', '1-2', '2-3', '3-4', '4-5', '5-']
>>> iris.sepallength.cut(range(6), labels=labels, include_over=True).
rename('sepallength_cut').head(5)
   sepallength_cut
0               5-
1              4-5
2              4-5
3              4-5
4              4-5
```

使用自定义函数及第三方库

DataFrame函数支持对Sequence使用map，它会对它的每个元素调用自定义函数。

```
>>> iris.sepallength.map(lambda x: x + 1).head(5)
   sepallength
0          6.1
1          5.9
2          5.7
3          5.6
4          6.0
```

说明:

目前，受限于Python UDF，自定义函数无法支持将List/Dict类型作为输入或输出。

如果map前后，Sequence的类型发生了变化，则需要显式指定map后的类型。

```
>>> iris.sepallength.map(lambda x: 't'+str(x), 'string').head(5)
   sepallength
0         t5.1
1         t4.9
```

```
2          t4.7
3          t4.6
4          t5.0
```

如果在函数中包含闭包，则函数外闭包变量值的变化会引起函数内该变量值的变化。

```
>>> dfs = []
>>> for i in range(10):
>>>     dfs.append(df.sepal_length.map(lambda x: x + i))
```

结果为dfs中每个SequenceExpr均为df.sepal_length+9。为解决此问题，可以将函数作为另一函数的返回值，或者使用partial。两个示例如下。

```
>>> dfs = []
>>> def get_mapper(i):
>>>     return lambda x: x + i
>>> for i in range(10):
>>>     dfs.append(df.sepal_length.map(get_mapper(i)))
```

```
>>> import functools
>>> dfs = []
>>> for i in range(10):
>>>     dfs.append(df.sepal_length.map(functools.partial(lambda v, x:
x + v, i)))
```

map也支持使用现有的UDF函数，传入的参数是str类型（函数名）或者Function对象，详情请参见函数。

map传入Python函数的实现使用了MaxCompute Python UDF。因此，如果您所在的Project不支持Python UDF，则map函数无法使用。除此以外，所有Python UDF的限制在此都适用。

目前，第三方库（包含C）只能使用Numpy，第三方库使用详情请参见使用第三方Python库。

除了调用自定义函数，DataFrame还提供了很多内置函数，这些函数中部分使用了map函数来实现。因此，如果您所在Project未开通Python UDF，则无法使用这些函数（注意：阿里云公共服务暂不提供对Python UDF的支持）。

说明:

由于字节码定义的差异，Python 3下使用新语言特性（例如yield from）时，代码在使用Python 2.7的MaxCompute Worker上执行时会发生错误。因此，建议您在Python 3下使用MapReduce API编写生产作业前，先确认相关代码是否能正常执行。

- 引用资源

  自定义函数也能读取MaxCompute上的资源（表资源或文件资源），或者引用一个collection作为资源。此时，自定义函数需要写成函数闭包或callable的类。两个示例如下。

```
>>> file_resource = o.create_resource('pyodps_iris_file', 'file',
file_obj='Iris-setosa')
>>>
>>> iris_names_collection = iris.distinct('name')[:2]
>>> iris_names_collection
       sepallength
0      Iris-setosa
1  Iris-versicolor
```

```
>>> def myfunc(resources):  # resources按调用顺序传入。
>>>     names = set()
>>>     fileobj = resources[0] # 文件资源是一个file-like的object。
>>>     for l in fileobj:
>>>         names.add(l)
>>>     collection = resources[1]
>>>     for r in collection:
>>>         names.add(r.name)  # 这里可以通过字段名或者偏移来取。
>>>     def h(x):
>>>         if x in names:
>>>             return True
>>>         else:
>>>             return False
>>>     return h
>>>
>>> df = iris.distinct('name')
>>> df = df[df.name,
>>>         df.name.map(myfunc, resources=[file_resource, iris_names
_collection], rtype='boolean').rename('isin')]
>>>
>>> df
              name   isin
0      Iris-setosa   True
1  Iris-versicolor   True
2   Iris-virginica  False
```

说明:

分区表资源在读取时不包含分区字段。

- 使用第三方Python库

  您可以把第三方Python包作为资源上传到MaxCompute，支持的格式有*whl*、*egg*、*zip*以及*tar.gz*。在全局或者在立即执行的方法时，指定需要使用的包文件，即可以在自定义函数中使用第三方库。第三方库的依赖库，也必须指定，否则依然会导致导入错误。

  - 您可以通过PyODPS的资源上传接口来完成资源的上传。

    下面以*python-dateutil*包进行举例：

    1. 使用`pip download`命令，下载包以及其依赖到某个路径。下载后会出现两个包：*six-1.10.0-py2.py3-none-any.whl*和*python_dateutil-2.5.3-py2.py3-none-any.whl*注意：这里需要下载支持Linux环境的包）。

       ```
       $ pip download python-dateutil -d /to/path/
       ```

    2. 分别把两个文件上传到MaxCompute资源。

       ```
       >>> # 这里要确保资源名的后缀是正确的文件类型。
       >>> odps.create_resource('six.whl', 'file', file_obj=open('six-
       1.10.0-py2.py3-none-any.whl', 'rb'))
       >>> odps.create_resource('python_dateutil.whl', 'file',
       file_obj=open('python_dateutil-2.5.3-py2.py3-none-any.whl', 'rb
       '))
       ```

    3. 现在有个DataFrame，只有一个STRING类型字段。

       ```
       >>> df
                     datestr
       0  2016-08-26 14:03:29
       1  2015-08-26 14:03:29
       ```

    4. 全局配置使用到的三方库如下。

       ```
       >>> from odps import options
       >>>
       >>> def get_year(t):
       >>>     from dateutil.parser import parse
       >>>     return parse(t).strftime('%Y')
       >>>
       >>> options.df.libraries = ['six.whl', 'python_dateutil.whl']
       >>> df.datestr.map(get_year)
          datestr
       0     2016
       1     2015
       ```

       或者通过运行方法的`libraries`参数指定使用到的第三方库。

       ```
       >>> def get_year(t):
       >>>     from dateutil.parser import parse
       >>>     return parse(t).strftime('%Y')
       >>>
       >>> df.datestr.map(get_year).execute(libraries=['six.whl', '
       python_dateutil.whl'])
          datestr
       ```

```
0     2016
1     2015
```

PyODPS默认支持执行仅包含Python且不含文件操作的第三方库。在较新版本的MaxCompute服务下，PyODPS也支持执行带有二进制代码或带有文件操作的Python库。这些库的后缀必须是*cp27-cp27m-manylinux1_x86_64*，以`archive`方式上传，*whl*后缀的包需要重命名为*zip*。同时，作业需要开启`odps.isolation.session.enable`选项，或者在Project级别开启`isolation`。以下示例为您展示如何上传并使用`scipy`中的特殊函数。

```
>>> # 对于含有二进制代码的包，必须使用archive方式上传资源，whl后缀需要改为zip。
>>> odps.create_resource('scipy.zip', 'archive', file_obj=open('scipy-0.19.0-cp27-cp27m-manylinux1_x86_64.whl', 'rb'))
>>>
>>> # 如果Project开启了isolation，下面的选项不是必需的。
>>> options.sql.settings = { 'odps.isolation.session.enable': True }
>>>
>>> def psi(value):
>>>     # 建议在函数内部import第三方库，以防止不同操作系统下二进制包结构差异造成执行错误。
>>>     from scipy.special import psi
>>>     return float(psi(value))
>>>
>>> df.float_col.map(psi).execute(libraries=['scipy.zip'])
```

对于只提供源码的二进制包，可以在Linux Shell中打包成Wheel再上传，Mac和Windows中生成的Wheel包无法在MaxCompute中使用。

```
python setup.py bdist_wheel
```

- 您也可以通过MaxCompute Console上传资源。

1. 现在主流的Python包都提供了whl包，提供了各平台包含二进制文件的包，因此找到可以在MaxCompute上运行的包是第一步。
2. 其次，要想在MaxCompute上运行，需要包含所有的依赖包，这个是比较繁琐的。各个包的依赖情况如下表所示。

包名	依赖
pandas	numpy，python-dateutil，pytz，six
scipy	numpy
scikit-learn	numpy，scipy

说明:

其中numpy已包含，您只需上传python-dateutil、pytz、pandas、scipy、sklearn、six包，pandas、scipy和scikit-learn即可使用。

3. 您可进入python-dateutils找到python-dateutil-2.6.0.zip进行下载。

4. 重命名为python-dateutil.zip，通过MaxCompute Console上传资源。

```
add archive python-dateutil.zip;
```

> 说明:
> pytz和six的上传方式同上，分别找到 pytz-2017.2.zip和six-1.11.0.tar.gz进行下载和上传资源操作。

5. 对于Pandas这种包含c的包，需要找到名字中包含cp27-cp27m-manylinux1_x86_64的whl包，这样才能在MaxCompute上正确执行。因此，您需要找到pandas-0.20.2-cp27-cp27m-manylinux1_x86_64.whl进行下载，然后把后缀改成zip，在MaxCompute Console中执行`add archive pandas.zip;`进行上传。scipy和scikit-learn包的操作同上。

所有包需要下载的资源如下表所示。

包名	文件名	上传资源名
python-dateutil	python-dateutil-2.6.0.zip	python-dateutil.zip
pytz	pytz-2017.2.zip	pytz.zip
six	six-1.11.0.tar.gz	six.tar.gz
pandas	pandas-0.20.2-cp27-cp27m-manylinux1_x86_64.zip	pandas.zip
scipy	scipy-0.19.0-cp27-cp27m-manylinux1_x86_64.zip	scipy.zip
scikit-learn	scikit_learn-0.18.1-cp27-cp27m-manylinux1_x86_64.zip	sklearn.zip

- 使用计数器

```
from odps.udf import get_execution_context
def h(x):
    ctx = get_execution_context()
    counters = ctx.get_counters()
    counters.get_counter('df', 'add_one').increment(1)
    return x + 1
df.field.map(h)
```

Logview的JSONSummary中即可找到计数器值。

调用MaxCompute内建或者已定义函数

如果您需要调用MaxCompute上的内建或者已定义函数来生成列，您可以使用`func`接口，该接口默认函数返回值为STRING，可以用`rtype`参数指定返回值。

```
>>> from odps.df import func
>>>
>>> iris[iris.name, func.rand(rtype='float').rename('rand')][:4]
>>> iris[iris.name, func.rand(10, rtype='float').rename('rand')][:4]
>>> # 调用ODPS上定义的UDF，列名无法确定时需要手动指定。
>>> iris[iris.name, func.your_udf(iris.sepalwidth, iris.sepallength,
rtype='float').rename('new_col')]
>>> # 从其它Project调用UDF，也可通过name参数指定列名。
>>> iris[iris.name, func.your_udf(iris.sepalwidth, iris.sepallength,
rtype='float', project='udf_project', name='new_col')]
```

说明:

Pandas后端不支持执行带有`func`的表达式。

8.5.8 聚合操作

本文为您介绍DataFrame支持的聚合操作以及如何实现分组聚合、编写自定义聚合。DataFrame提供对列进行HyperLogLog计数的接口。

```
from odps.df import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
```

使用`describe`函数，查看DataFrame里数字列的数量、最大值、最小值、平均值以及标准差。

```
>>> print(iris.describe())
    type  sepal_length  sepal_width  petal_length  petal_width
0  count    150.000000   150.000000    150.000000   150.000000
1   mean      5.843333     3.054000      3.758667     1.198667
2    std      0.828066     0.433594      1.764420     0.763161
3    min      4.300000     2.000000      1.000000     0.100000
4    max      7.900000     4.400000      6.900000     2.500000
```

使用单列执行聚合操作。

```
>>> iris.sepallength.max()
```

```
7.9
```

如果要在消除重复后的列上进行聚合，可以先调用unique方法，再调用相应的聚合函数。

```
>>> iris.name.unique().cat(sep=',')
u'Iris-setosa,Iris-versicolor,Iris-virginica'
```

如果所有列支持同一种聚合操作，也可以直接在整个DataFrame上执行聚合操作。

```
>>> iris.exclude('category').mean()
   sepal_length  sepal_width  petal_length  petal_width
1      5.843333     3.054000      3.758667     1.198667
```

在DataFrame上执行count获取的是DataFrame的总行数。

```
>>> iris.count()
150
```

PyODPS支持的聚合操作，如下表所示。

聚合操作	说明
count（或size）	数量。
nunique	不重复值数量。
min	最小值。
max	最大值。
sum	求和。
mean	均值。
median	中位数。
quantile(p)	p分位数，仅在整数值下可取得准确值。
var	方差。
std	标准差。
moment	n阶中心矩（或n阶矩）。
skew	样本偏度（无偏估计）。
kurtosis	样本峰度（无偏估计）。
cat	按sep做字符串连接操作。
tolist	组合为LIST。

说明:

不同于Pandas，对于列上的聚合操作，无论是在ODPS还是Pandas后端，PyODPS DataFrame都会忽略空值。这一逻辑与SQL类似。

分组聚合

DataFrame API提供了groupby执行分组操作，分组后的一个主要操作是通过调用agg或者aggregate方法，执行聚合操作。 最终的结果列中会包含分组的列，以及聚合的列。

```
>>> iris.groupby('name').agg(iris.sepallength.max(), smin=iris.
sepallength.min())
              name  sepallength_max  smin
0      Iris-setosa              5.8   4.3
1  Iris-versicolor              7.0   4.9
2   Iris-virginica              7.9   4.9
```

DataFrame提供了value_counts操作，用于返回按某列分组后，将每个组的个数从大到小进行排列。

- 使用groupby实现。

  ```
  >>> iris.groupby('name').agg(count=iris.name.count()).sort('count',
  ascending=False).head(5)
                name  count
  0   Iris-virginica     50
  1  Iris-versicolor     50
  2      Iris-setosa     50
  ```

- 使用value_counts实现。

  ```
  >>> iris['name'].value_counts().head(5)
                name  count
  0   Iris-virginica     50
  1  Iris-versicolor     50
  2      Iris-setosa     50
  ```

对于聚合后的单列操作，您也可以直接取出列名。但此时只能使用聚合函数。

```
>>> iris.groupby('name').petallength.sum()
   petallength_sum
0            73.2
1           213.0
2           277.6
```

```
>>> iris.groupby('name').agg(iris.petallength.notnull().sum())
              name  petallength_sum
0      Iris-setosa               50
1  Iris-versicolor               50
2   Iris-virginica               50
```

分组时也支持对常量进行分组，但是需要使用Scalar初始化。

```
>>> from odps.df import Scalar
>>> iris.groupby(Scalar(1)).petallength.sum()
   petallength_sum
```

```
0            563.8
```

编写自定义聚合

对字段调用agg或者aggregate方法调用自定义聚合。自定义聚合需要提供一个类，这个类需要提供以下方法：

- buffer()：返回一个Mutable的Object（例如LIST、DICT），buffer大小不应随数据而递增。
- __call__(buffer, *val)：将值聚合到中间buffer。
- merge(buffer, pbuffer)：将pbuffer聚合到buffer中。
- getvalue(buffer)：返回最终值。

计算平均值的示例如下。

```
class Agg(object):

    def buffer(self):
        return [0.0, 0]

    def __call__(self, buffer, val):
        buffer[0] += val
        buffer[1] += 1

    def merge(self, buffer, pbuffer):
        buffer[0] += pbuffer[0]
        buffer[1] += pbuffer[1]

    def getvalue(self, buffer):
        if buffer[1] == 0:
            return 0.0
        return buffer[0] / buffer[1]
```

执行结果如下。

```
>>> iris.sepalwidth.agg(Agg)
3.0540000000000007
```

如果最终类型和输入类型发生了变化，则需指定类型。

```
>>> iris.sepalwidth.agg(Agg, 'float')
```

自定义聚合也可以用在分组聚合中。

```
>>> iris.groupby('name').sepalwidth.agg(Agg)
   petallength_aggregation
0                    3.418
1                    2.770
2                    2.974
```

当对多列调用自定义聚合，可以使用agg方法。

```
class Agg(object):
```

```
    def buffer(self):
        return [0.0, 0.0]

    def __call__(self, buffer, val1, val2):
        buffer[0] += val1
        buffer[1] += val2

    def merge(self, buffer, pbuffer):
        buffer[0] += pbuffer[0]
        buffer[1] += pbuffer[1]

    def getvalue(self, buffer):
        if buffer[1] == 0:
            return 0.0
        return buffer[0] / buffer[1]r[0] / buffer[1]
```

执行结果如下。

```
>>> from odps.df import agg
>>> to_agg = agg([iris.sepalwidth, iris.sepallength], Agg, rtype='
float')  # 对两列调用自定义聚合。
>>> iris.groupby('name').agg(val=to_agg)
              name       val
0      Iris-setosa  0.682781
1  Iris-versicolor  0.466644
2   Iris-virginica  0.451427
```

如果您需要调用ODPS上已经存在的UDAF，指定函数名即可。

```
>>> iris.groupby('name').agg(iris.sepalwidth.agg('your_func'))  # 对单
列聚合。
>>> to_agg = agg([iris.sepalwidth, iris.sepallength], 'your_func',
rtype='float')
>>> iris.groupby('name').agg(to_agg.rename('val'))  # 对多列聚合。
```

说明:

目前，因受限于Python UDF，自定义聚合无法支持将LIST/DICT类型作为初始输入或最终输出结果。

HyperLogLog计数

DataFrame提供了对列进行HyperLogLog计数的接口`hll_count`，这个接口是个近似的估计接口。当数据量很大时，它可以较快地对数据的唯一个数进行估计。

使用该接口计算海量用户UV时，可以快速得出估计值。

```
>>> df = DataFrame(pd.DataFrame({'a': np.random.randint(100000, size=
100000)}))
>>> df.a.hll_count()
63270
>>> df.a.nunique()
```

```
63250
```

说明:

提供的splitter参数会对每个字段进行分隔，再计算唯一数。

8.5.9 排序、去重、采样、数据变换

您可以对DataFrame对象执行排序、去重、采样、数据变换操作。

```
from odps.df import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
```

排序

排序操作只能作用于Collection。

- **调用sort或者sort_values方法进行排序。**

```
>>> iris.sort('sepalwidth').head(5)
   sepallength  sepalwidth  petallength  petalwidth            name
0          5.0         2.0          3.5         1.0  Iris-versicolor
1          6.2         2.2          4.5         1.5  Iris-versicolor
2          6.0         2.2          5.0         1.5   Iris-virginica
3          6.0         2.2          4.0         1.0  Iris-versicolor
4          5.5         2.3          4.0         1.3  Iris-versicolor
```

- **设置参数ascending=False;进行降序排列。**

```
>>> iris.sort('sepalwidth', ascending=False).head(5)
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.7         4.4          1.5         0.4  Iris-setosa
1          5.5         4.2          1.4         0.2  Iris-setosa
2          5.2         4.1          1.5         0.1  Iris-setosa
3          5.8         4.0          1.2         0.2  Iris-setosa
4          5.4         3.9          1.3         0.4  Iris-setosa
```

- **设置-实现降序排列。**

```
>>> iris.sort(-iris.sepalwidth).head(5)
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.7         4.4          1.5         0.4  Iris-setosa
1          5.5         4.2          1.4         0.2  Iris-setosa
2          5.2         4.1          1.5         0.1  Iris-setosa
3          5.8         4.0          1.2         0.2  Iris-setosa
4          5.4         3.9          1.3         0.4  Iris-setosa
```

- **实现多字段排序。**

```
>>> iris.sort(['sepalwidth', 'petallength']).head(5)
   sepallength  sepalwidth  petallength  petalwidth            name
0          5.0         2.0          3.5         1.0  Iris-versicolor
1          6.0         2.2          4.0         1.0  Iris-versicolor
2          6.2         2.2          4.5         1.5  Iris-versicolor
3          6.0         2.2          5.0         1.5   Iris-virginica
```

```
4          4.5          2.3          1.3          0.3       Iris-setosa
```

- 多字段排序时，如果是升序降序不同，`ascending`参数可以用于传入一个列表，长度必须等同于排序的字段，它们的值都是BOOLEAN类型。

```
>>> iris.sort(['sepalwidth', 'petallength'], ascending=[True, False
]).head(5)
   sepallength  sepalwidth  petallength  petalwidth            name
0          5.0         2.0          3.5         1.0  Iris-versicolor
1          6.0         2.2          5.0         1.5   Iris-virginica
2          6.2         2.2          4.5         1.5  Iris-versicolor
3          6.0         2.2          4.0         1.0  Iris-versicolor
4          6.3         2.3          4.4         1.3  Iris-versicolor
```

下面代码可以实现同样的效果。

```
>>> iris.sort(['sepalwidth', -iris.petallength]).head(5)
   sepallength  sepalwidth  petallength  petalwidth            name
0          5.0         2.0          3.5         1.0  Iris-versicolor
1          6.0         2.2          5.0         1.5   Iris-virginica
2          6.2         2.2          4.5         1.5  Iris-versicolor
3          6.0         2.2          4.0         1.0  Iris-versicolor
4          6.3         2.3          4.4         1.3  Iris-versicolor
```

说明：

由于ODPS要求排序必须指定个数，所以在ODPS后端执行时，会通过`options.df.odps.sort.limit`指定排序个数，这个值默认是10000。如果需要排序尽量多的数据，可以把这个值设到较大的值。但是，这样可能会导致OOM。

去重

- 您可以通过以下三种方式调用`distinct`方法，对Collection进行去重操作。

  - ```
    >>> iris[['name']].distinct()
                   name
    0      Iris-setosa
    1  Iris-versicolor
    2   Iris-virginica
    ```

  - ```
    >>> iris.distinct('name')
                   name
    0      Iris-setosa
    1  Iris-versicolor
    2   Iris-virginica
    ```

  - ```
    >>> iris.distinct('name', 'sepallength').head(3)
              name  sepallength
    0  Iris-setosa          4.3
    1  Iris-setosa          4.4
    ```
    ```

```
2  Iris-setosa          4.5
```

- 您可以调用unique对Sequence进行去重操作，但是调用unique的Sequence不能用在列选择中。

```
>>> iris.name.unique()
              name
0      Iris-setosa
1  Iris-versicolor
2   Iris-virginica
```

错误示例如下。

```
>>> iris[iris.name, iris.name.unique()]
```

采样

要对一个Collection的数据采样，可以调用sample方法。PyODPS支持以下四种采样方式：

> **说明：**
> 除了按份数采样外，其余方法如果要在ODPS DataFrame上执行，需要Project支持XFlow，否则，这些方法只能在Pandas DataFrame后端上执行。

- 按份数采样

  在这种采样方式下，数据被分为parts份，可选择选取的份数序号。

```
>>> iris.sample(parts=10)  # 分成10份，默认取第0份。
>>> iris.sample(parts=10, i=0)  # 手动指定取第0份。
>>> iris.sample(parts=10, i=[2, 5])   # 分成10份，取第2和第5份。
>>> iris.sample(parts=10, columns=['name', 'sepalwidth'])  # 根据name和sepalwidth的值做采样。
```

- 按比例、条数采样

  在这种采样方式下，用户指定需要采样的数据条数或采样比例。指定replace参数为True可启用放回采样。

```
>>> iris.sample(n=100)  # 选取100条数据。
>>> iris.sample(frac=0.3)  # 采样30%的数据。
```

- 按权重列采样

  在这种采样方式下，您可以指定权重列和数据条数、采样比例。指定replace参数为True可启用放回采样。

```
>>> iris.sample(n=100, weights='sepal_length')
```

```
>>> iris.sample(n=100, weights='sepal_width', replace=True)
```

- 分层采样

  在这种采样方式下，您可以指定用于分层的标签列，同时为需要采样的每个标签指定采样比例（`frac`参数）或条数（`n`参数）。暂不支持放回采样。

```
>>> iris.sample(strata='category', n={'Iris Setosa': 10, 'Iris
Versicolour': 10})
>>> iris.sample(strata='category', frac={'Iris Setosa': 0.5, 'Iris
Versicolour': 0.4})
```

数据缩放

DataFrame支持通过最大、最小值或平均值、标准差对数据进行缩放。示例数据如下。

```
name  id  fid
0  name1   4  5.3
1  name2   2  3.5
2  name2   3  1.5
3  name1   4  4.2
4  name1   3  2.2
5  name1   3  4.1
```

- 使用`min_max_scale`方法进行归一化。

```
df.min_max_scale(columns=['fid'])
    name  id       fid
0  name1   4  1.000000
1  name2   2  0.526316
2  name2   3  0.000000
3  name1   4  0.710526
4  name1   3  0.184211
5  name1   3  0.684211
```

- `min_max_scale`还支持使用`feature_range`参数指定输出值的范围。例如，需要使输出值在(-1, 1) 范围内，示例如下。

```
df.min_max_scale(columns=['fid'], feature_range=(-1, 1))
    name  id       fid
0  name1   4  1.000000
1  name2   2  0.052632
2  name2   3 -1.000000
3  name1   4  0.421053
4  name1   3 -0.631579
5  name1   3  0.368421
```

- 如果需要保留原始值，可以使用`preserve`参数。此时，缩放后的数据将会以新增列的形式追加到数据中，列名默认为原列名追加_*scaled*后缀，该后缀可使用`suffix`参数更改。

```
df.min_max_scale(columns=['fid'], preserve=True)
    name  id  fid  fid_scaled
0  name1   4  5.3    1.000000
1  name2   2  3.5    0.526316
2  name2   3  1.5    0.000000
3  name1   4  4.2    0.710526
```

```
4  name1   3  2.2    0.184211
5  name1   3  4.1    0.684211
```

- min_max_scale也支持使用group参数指定一个或多个分组列，在分组列中分别取最值进行缩放。

```
df.min_max_scale(columns=['fid'], group=['name'])
    name  id       fid
0  name1   4  1.000000
1  name1   4  0.645161
2  name1   3  0.000000
3  name1   3  0.612903
4  name2   2  1.000000
5  name2   3  0.000000
```

结果显示，name1和name2两组均按组中的最值进行了缩放。

- 使用std_scale方法可以依照标准正态分布对数据进行调整。std_scale同样支持preserve参数保留原始列以及使用group进行分组。

```
df.std_scale(columns=['fid'])
    name  id       fid
0  name1   4  1.436467
1  name2   2  0.026118
2  name2   3 -1.540938
3  name1   4  0.574587
4  name1   3 -0.992468
5  name1   3  0.496234
```

空值处理

DataFrame支持筛去空值以及填充空值的功能。示例数据如下。

```
id   name   f1   f2   f3   f4
0   0  name1  1.0  NaN  3.0  4.0
1   1  name1  2.0  NaN  NaN  1.0
2   2  name1  3.0  4.0  1.0  NaN
3   3  name1  NaN  1.0  2.0  3.0
4   4  name1  1.0  NaN  3.0  4.0
5   5  name1  1.0  2.0  3.0  4.0
6   6  name1  NaN  NaN  NaN  NaN
```

- 使用dropna，删除subset中包含空值的行。

```
df.dropna(subset=['f1', 'f2', 'f3', 'f4'])
   id   name   f1   f2   f3   f4
0   5  name1  1.0  2.0  3.0  4.0
```

- 如果行中包含非空值则不删除，可以使用how='all'。

```
df.dropna(how='all', subset=['f1', 'f2', 'f3', 'f4'])
   id   name   f1   f2   f3   f4
0   0  name1  1.0  NaN  3.0  4.0
1   1  name1  2.0  NaN  NaN  1.0
2   2  name1  3.0  4.0  1.0  NaN
3   3  name1  NaN  1.0  2.0  3.0
4   4  name1  1.0  NaN  3.0  4.0
```

```
5   5  name1  1.0  2.0  3.0  4.0
```

- 使用thresh参数指定行中至少要有多少个非空值。

```
df.dropna(thresh=3, subset=['f1', 'f2', 'f3', 'f4'])
   id   name   f1   f2   f3   f4
0   0  name1  1.0  NaN  3.0  4.0
2   2  name1  3.0  4.0  1.0  NaN
3   3  name1  NaN  1.0  2.0  3.0
4   4  name1  1.0  NaN  3.0  4.0
5   5  name1  1.0  2.0  3.0  4.0
```

- 调用fillna方法，使用常数或已有的列来填充未知值。
  - 使用常数填充未知值。

```
df.fillna(100, subset=['f1', 'f2', 'f3', 'f4'])
   id   name     f1     f2     f3     f4
0   0  name1    1.0  100.0    3.0    4.0
1   1  name1    2.0  100.0  100.0    1.0
2   2  name1    3.0    4.0    1.0  100.0
3   3  name1  100.0    1.0    2.0    3.0
4   4  name1    1.0  100.0    3.0    4.0
5   5  name1    1.0    2.0    3.0    4.0
6   6  name1  100.0  100.0  100.0  100.0
```

  - 使用一个已有的列填充未知值。

```
df.fillna(df.f2, subset=['f1', 'f2', 'f3', 'f4'])
   id   name   f1   f2   f3   f4
0   0  name1  1.0  NaN  3.0  4.0
1   1  name1  2.0  NaN  NaN  1.0
2   2  name1  3.0  4.0  1.0  4.0
3   3  name1  1.0  1.0  2.0  3.0
4   4  name1  1.0  NaN  3.0  4.0
5   5  name1  1.0  2.0  3.0  4.0
6   6  name1  NaN  NaN  NaN  NaN
```

- DataFrame提供了向前、向后填充的功能。您可以按照表格中的取值范围为method参数进行赋值。

取值	含义
bfill、backfill	向前填充。
ffill、pad	向后填充。

示例

```
df.fillna(method='bfill', subset=['f1', 'f2', 'f3', 'f4'])
   id   name   f1   f2   f3   f4
0   0  name1  1.0  3.0  3.0  4.0
1   1  name1  2.0  1.0  1.0  1.0
2   2  name1  3.0  4.0  1.0  NaN
3   3  name1  1.0  1.0  2.0  3.0
4   4  name1  1.0  3.0  3.0  4.0
5   5  name1  1.0  2.0  3.0  4.0
6   6  name1  NaN  NaN  NaN  NaN
```

```
df.fillna(method='ffill', subset=['f1', 'f2', 'f3', 'f4'])
   id   name   f1   f2   f3   f4
0   0  name1  1.0  1.0  3.0  4.0
1   1  name1  2.0  2.0  2.0  1.0
2   2  name1  3.0  4.0  1.0  1.0
3   3  name1  NaN  1.0  2.0  3.0
4   4  name1  1.0  1.0  3.0  4.0
5   5  name1  1.0  2.0  3.0  4.0
6   6  name1  NaN  NaN  NaN  NaN
```

您也可以使用ffill、bfill函数简化代码。ffill等价于fillna(method='ffill');bfill等价于fillna(method='bfill')。

8.5.10 对所有行和列调用自定义函数

本文为您介绍如何对所有行和列数据使用自定义函数。

对一行数据使用自定义函数

如果您需要对一行数据使用自定义函数，可以使用apply方法。参数axis的值必须设为1，表示对行进行操作。apply的自定义函数接收一个参数，参数为上一步Collection的一行数据。您可以通过属性或者偏移获得一个字段的数据。

- reduce为True时，表示返回结果为Sequence，否则返回结果为Collection。 names和types参数分别指定返回的Sequence或Collection的字段名和类型。 如果未指定类型，则会默认为STRING类型。

```
>>> iris.apply(lambda row: row.sepallength + row.sepalwidth, axis=1
, reduce=True, types='float').rename('sepaladd').head(3)
   sepaladd
0       8.6
1       7.9
2       7.9
```

- 在apply的自定义函数中，reduce为False时，您可以使用yield关键字返回多行结果。

```
>>> iris.count()
150
>>>
>>> def handle(row):
>>>     yield row.sepallength - row.sepalwidth, row.sepallength +
row.sepalwidth
>>>     yield row.petallength - row.petalwidth, row.petallength +
row.petalwidth
>>>
>>> iris.apply(handle, axis=1, names=['iris_add', 'iris_sub'], types
=['float', 'float']).count()
300
```

- 您也可以在函数中注释返回的字段和类型，无需在函数调用时再指定它们。

```
>>> from odps.df import output
>>>
>>> @output(['iris_add', 'iris_sub'], ['float', 'float'])
>>> def handle(row):
```

```
>>>     yield row.sepallength - row.sepalwidth, row.sepallength +
row.sepalwidth
>>>     yield row.petallength - row.petalwidth, row.petallength +
row.petalwidth
>>>
>>> iris.apply(handle, axis=1).count()
300
```

- 您也可以使用map-only的map_reduce，该操作与axis=1的apply操作是等价的。

```
>>> iris.map_reduce(mapper=handle).count()
300
```

- 如果您想调用ODPS上已经存在的UDTF，函数指定为函数名即可。

```
>>> iris['name', 'sepallength'].apply('your_func', axis=1, names=['
name2', 'sepallength2'], types=['string', 'float'])
```

- 使用apply对行操作，且reduce为False时，您可以使用并列多行输出与已有的行结合，用于后续聚合等操作。

```
>>> from odps.df import output
>>>
>>> @output(['iris_add', 'iris_sub'], ['float', 'float'])
>>> def handle(row):
>>>     yield row.sepallength - row.sepalwidth, row.sepallength +
row.sepalwidth
>>>     yield row.petallength - row.petalwidth, row.petallength +
row.petalwidth
>>>
>>> iris[iris.category, iris.apply(handle, axis=1)]
```

对所有列调用自定义聚合

调用apply方法，当不指定axis，或者axis值为0时，您可以通过传入一个自定义聚合类对所有Sequence进行聚合操作。

```
class Agg(object):

    def buffer(self):
        return [0.0, 0]

    def __call__(self, buffer, val):
        buffer[0] += val
        buffer[1] += 1

    def merge(self, buffer, pbuffer):
        buffer[0] += pbuffer[0]
        buffer[1] += pbuffer[1]

    def getvalue(self, buffer):
        if buffer[1] == 0:
            return 0.0
        return buffer[0] / buffer[1]
```

```
>>> iris.exclude('name').apply(Agg)
   sepallength_aggregation  sepalwidth_aggregation  petallengt
h_aggregation  petalwidth_aggregation
```

```
0                  5.843333                     3.054                     3.
758667                 1.198667
```

说明:

目前，受限于Python UDF，自定义函数无法支持将LIST/DICT类型作为初始输入或最终输出结果。

引用资源

类似于对map方法的Resources参数，每个Resource可以是ODPS上的资源（表资源或文件资源），或者引用一个Collection作为资源。

当`axis`值为1，即在行上操作时，您需要写一个函数闭包或者Callable的类。 而对于列上的聚合操作，您只需在 `__init__` 函数里读取资源即可。

```
>>> words_df
                     sentence
0                 Hello World
1                Hello Python
2  Life is short I use Python
>>>
>>> import pandas as pd
>>> stop_words = DataFrame(pd.DataFrame({'stops': ['is', 'a', 'I']}))
>>>
>>> @output(['sentence'], ['string'])
>>> def filter_stops(resources):
>>>     stop_words = set([r[0] for r in resources[0]])
>>>     def h(row):
>>>         return ' '.join(w for w in row[0].split() if w not in
stop_words),
>>>     return h
>>>
>>> words_df.apply(filter_stops, axis=1, resources=[stop_words])
                sentence
0            Hello World
1           Hello Python
2  Life short use Python
```

说明:

这里的`stop_words`存放于本地，但在真正执行时会被上传到ODPS作为资源引用。

使用第三方Python库

使用方法类似在map中使用第三方Python库 。

- **在全局指定使用的库。**

```
>>> from odps import options
```

```
>>> options.df.libraries = ['six.whl', 'python_dateutil.whl']
```

- 在立即执行的方法中，局部指定使用的库。

```
>>> df.apply(my_func, axis=1).to_pandas(libraries=['six.whl', '
python_dateutil.whl'])
```

说明:

由于字节码定义的差异，Python 3下使用新语言特性（例如`yield from`）时，代码在使用Python 2.7的ODPS Worker上执行时会发生错误。因而建议在Python 3下使用MapReduce API编写生产作业前，先确认相关代码是否能正常undefined执行。

8.5.11 MapReduce API

本文为您介绍MapReduce API。

PyODPS DataFrame支持MapReduce API，您可以分别编写`map`和`reduce`函数（`map_reduce`可以只有`mapper`或者`reducer`过程）。

`wordcount`的示例如下。

```
>>> #encoding=utf-8
>>> from odps import ODPS
>>> from odps import options
>>> options.verbose = True
>>> o = ODPS('your-access-id', 'your-secret-access-key',project='
DMP_UC_dev', endpoint='http://service-corp.odps.aliyun-inc.com/api')
>>> from odps.df import DataFrame

>>> def mapper(row):
>>>     for word in row[0].split():
>>>         yield word.lower(), 1
>>>
>>> def reducer(keys):
>>>     # 这里使用list而不是cnt=0，否则h内的cnt会被认为是局部变量，其中的赋值无
法输出。
>>>     cnt = [0]
>>>     def h(row, done):  # done表示这个key已经迭代结束。
>>>         cnt[0] += row[1]
>>>         if done:
>>>             yield keys[0], cnt[0]
>>>     return h
>>>
>>> word_count = DataFrame(o.get_table('zx_word_count'))
>>> table = word_count.map_reduce(mapper, reducer, group=['word', ],
                          mapper_output_names=['word', 'cnt'],
                          mapper_output_types=['string', 'int'],
                          reducer_output_names=['word', 'cnt'],
                          reducer_output_types=['string', 'int'])
         word  cnt
0         are    1
1         day    1
2      doing?    1
3   everybody    1
4       first    1
5       hello    2
```

```
6         how    1
7          is    1
8          so    1
9         the    1
10       this    1
11      world    1
12        you    1
```

group参数用于指定reduce按哪些字段做分组，如果不指定，会按全部字段做分组。reducer需要接收聚合的keys进行初始化，并能继续处理按这些keys聚合的每行数据。done表示与这些keys相关的所有行是否都迭代完成。

为了方便，此处写成了函数闭包的方式，您也可以写成Callable的类。

```
class reducer(object):
    def __init__(self, keys):
        self.cnt = 0

    def __call__(self, row, done):  # done表示这个key已经迭代结束。
        self.cnt += row.cnt
        if done:
            yield row.word, self.cnt
```

使用output进行注释会让代码更简单些。

```
>>> from odps.df import output
>>>
>>> @output(['word', 'cnt'], ['string', 'int'])
>>> def mapper(row):
>>>     for word in row[0].split():
>>>         yield word.lower(), 1
>>>
>>> @output(['word', 'cnt'], ['string', 'int'])
>>> def reducer(keys):
>>>     # 这里使用list而不是cnt=0，否则h内的cnt会被认为是局部变量，其中的赋值无法输出。
>>>     cnt = [0]
>>>     def h(row, done):  # done表示这个key已经迭代结束。
>>>         cnt[0] += row.cnt
>>>         if done:
>>>             yield keys.word, cnt[0]
>>>     return h
>>>
>>> word_count = DataFrame(o.get_table('zx_word_count'))
>>> table = word_count.map_reduce(mapper, reducer, group='word')
         word  cnt
0         are    1
1         day    1
2      doing?    1
3   everybody    1
4       first    1
5       hello    2
6         how    1
7          is    1
8          so    1
9         the    1
10       this    1
11      world    1
```

```
12        you    1
```

在迭代的时候，可以使用sort参数实现按指定列排序，通过ascending参数指定升序降序。ascending参数可以是一个BOOL值，表示所有的sort字段是相同升序或降序， 也可以是一个列表，长度必须和sort字段长度相同。

指定COMBINER

combiner表示在map_reduce API里表示在mapper端，就先对数据进行聚合操作，它的用法和reducer是完全一致的，但不能引用资源。 并且， combiner的输出的字段名和字段类型必须和mapper完全一致。

上面的例子，您可以使用reducer作为combiner在mapper端对数据做初步的聚合，减少Shuffle出去的数据量。

```
>>> words_df.map_reduce(mapper, reducer, combiner=reducer, group='word
')
```

引用资源

在MapReduce API里，您可以分别指定mapper和reducer所要引用的资源。

以下示例为对mapper里的单词做停词过滤，在reducer里对白名单的单词数量加5。

```
>>> white_list_file = o.create_resource('pyodps_white_list_words', '
file', file_obj='Python\nWorld')
>>>
>>> @output(['word', 'cnt'], ['string', 'int'])
>>> def mapper(resources):
>>>     stop_words = set(r[0].strip() for r in resources[0])
>>>     def h(row):
>>>         for word in row[0].split():
>>>             if word not in stop_words:
>>>                 yield word, 1
>>>     return h
>>>
>>> @output(['word', 'cnt'], ['string', 'int'])
>>> def reducer(resources):
>>>     d = dict()
>>>     d['white_list'] = set(word.strip() for word in resources[0])
>>>     d['cnt'] = 0
>>>     def inner(keys):
>>>         d['cnt'] = 0
>>>         def h(row, done):
>>>             d['cnt'] += row.cnt
>>>             if done:
>>>                 if row.word in d['white_list']:
>>>                     d['cnt'] += 5
>>>                 yield keys.word, d['cnt']
>>>         return h
>>>     return inner
>>>
>>> words_df.map_reduce(mapper, reducer, group='word',
>>>                     mapper_resources=[stop_words], reducer_re
sources=[white_list_file])
```

```
     word  cnt
0   hello    2
1    life    1
2  python    7
3   world    6
4   short    1
5     use    1
```

使用第三方Python库

使用方法类似在MAP中使用第三方Python库。

- 在全局指定使用的库。

```
>>> from odps import options
>>> options.df.libraries = ['six.whl', 'python_dateutil.whl']
```

- 在立即执行的方法中，局部指定使用的库。

```
>>> df.map_reduce(mapper=my_mapper, reducer=my_reducer, group='key
').execute(libraries=['six.whl', 'python_dateutil.whl'])
```

说明:

由于字节码定义的差异，Python 3下使用新语言特性（例如`yield from`）时，代码在使用 Python 2.7的ODPS Worker上执行时会发生错误。因此，建议您在Python 3下使用MapReduce API编写生产作业前，先确认相关代码是否能正常执行。

重排数据

当数据在集群上分布不均匀时，您可以调用`reshuffle`接口对数据重排。

```
>>> df1 = df.reshuffle()
```

默认会按随机数做哈希来分布，但您也可以按指定列做分布，并且可以指定重排后的排序顺序。

```
>>> df1.reshuffle('name', sort='id', ascending=False)
```

布隆过滤器

PyODPS DataFrame提供了`bloom_filter`接口进行布隆过滤器的计算。

给定某个Collection和它的某个列计算的`sequence1`，对另外一个`sequence2`进行布隆过滤时，`sequence1`不存在于`sequence2`中的数据一定会被过滤掉， 但可能无法完全过滤。所以这种过滤方式是一种近似的过滤方法。这样的好处是对Collection进行快速过滤一些无用数据。这在一边数据量远大过另一边数据量（大部分数据并不会参与`join`运算）时的大规模`join`场景很有用。例

如，在join用户的浏览数据和交易数据时，用户的浏览数据量远大于交易数据量，可以利用交易数据先对浏览数据进行布隆过滤， 然后再join可以很好地提升性能。

```
>>> df1 = DataFrame(pd.DataFrame({'a': ['name1', 'name2', 'name3', '
name1'], 'b': [1, 2, 3, 4]}))
>>> df1
       a  b
0  name1  1
1  name2  2
2  name3  3
3  name1  4
>>> df2 = DataFrame(pd.DataFrame({'a': ['name1']}))
>>> df2
       a
0  name1
>>> df1.bloom_filter('a', df2.a) # 这里第0个参数可以是个计算表达式如：df1.a
 + '1'。
       a  b
0  name1  1
1  name1  4
```

示例说明：

- 由于数据量很小，df1中的a为name2和name3的行都被正确过滤掉了，当数据量很大的时候，可能无法过滤全部数据。
- 之前提join场景中，少量数据不能被过滤的情况并不会影响数据的正确性，但可以较大地提升join的性能。
- 您可以传入capacity和error_rate来设置数据的量以及错误率，默认值是3000和0.01。

说明：

要注意，调大capacity或者减小error_rate会增加内存的使用，所以应当根据实际情况选择一个合理的值。

Collection对象操作请参见DataFrame执行。

透视表（PIVOT_TABLE）

PyODPS DataFrame提供了透视表的功能。示例的表数据如下。

```
>>> df
     A    B      C  D  E
0  foo  one  small  1  3
1  foo  one  large  2  4
2  foo  one  large  2  5
3  foo  two  small  3  6
4  foo  two  small  3  4
5  bar  one  large  4  5
6  bar  one  small  5  3
7  bar  two  small  6  2
```

```
8  bar  two  large  7  1
```

- 使用透视表功能时，rows参数为必选参数，表示按一个或者多个字段做取平均值的操作。

```
>>> df['A', 'D', 'E'].pivot_table(rows='A')
     A  D_mean  E_mean
0  bar     5.5    2.75
1  foo     2.2    4.40
```

- rows可以提供多个字段，表示按多个字段做聚合。

```
>>> df.pivot_table(rows=['A', 'B', 'C'])
     A    B      C  D_mean  E_mean
0  bar  one  large     4.0     5.0
1  bar  one  small     5.0     3.0
2  bar  two  large     7.0     1.0
3  bar  two  small     6.0     2.0
4  foo  one  large     2.0     4.5
5  foo  one  small     1.0     3.0
6  foo  two  small     3.0     5.0
```

- 指定values显示指定要计算的列。

```
>>> df.pivot_table(rows=['A', 'B'], values='D')
     A    B    D_mean
0  bar  one  4.500000
1  bar  two  6.500000
2  foo  one  1.666667
3  foo  two  3.000000
```

- 计算值列时，默认会计算平均值。通过aggfunc指定一个或者多个聚合函数。

```
>>> df.pivot_table(rows=['A', 'B'], values=['D'], aggfunc=['mean', '
count', 'sum'])
     A    B    D_mean  D_count  D_sum
0  bar  one  4.500000        2      9
1  bar  two  6.500000        2     13
2  foo  one  1.666667        3      5
3  foo  two  3.000000        2      6
```

- 将原始数据的某一列的值，作为新的Collection的列。

```
>>> df.pivot_table(rows=['A', 'B'], values='D', columns='C')
     A    B  large_D_mean  small_D_mean
0  bar  one           4.0           5.0
1  bar  two           7.0           6.0
2  foo  one           2.0           1.0
3  foo  two           NaN           3.0
```

- 使用fill_value填充空值。

```
>>> df.pivot_table(rows=['A', 'B'], values='D', columns='C',
fill_value=0)
     A    B  large_D_mean  small_D_mean
0  bar  one             4             5
1  bar  two             7             6
2  foo  one             2             1
```

```
3  foo  two             0             3
```

KEY-VALUE字符串转换

DataFrame提供了将Key-Value对展开为列，以及将普通列转换为Key-Value列的功能。

- 将Key-Value对展开为列，示例数据如下。

```
>>> df
    name                kv
0  name1  k1=1,k2=3,k5=10
1  name1    k1=7.1,k7=8.2
2  name2    k2=1.2,k3=1.5
3  name2      k9=1.1,k2=1
```

通过`extract_kv`方法将Key-Value字段展开。

```
>>> df.extract_kv(columns=['kv'], kv_delim='=', item_delim=',')
   name   kv_k1  kv_k2  kv_k3  kv_k5  kv_k7  kv_k9
0  name1    1.0    3.0    NaN   10.0    NaN    NaN
1  name1    7.0    NaN    NaN    NaN    8.2    NaN
2  name2    NaN    1.2    1.5    NaN    NaN    NaN
3  name2    NaN    1.0    NaN    NaN    NaN    1.1
```

其中，需要展开的字段名由`columns`指定，Key和Value之间的分隔符，以及Key-Value对之间的分隔符分别由`kv_delim`和`item_delim`这两个参数指定，默认分别为半角冒号和半角逗号。输出的字段名为原字段名和Key值的组合，通过_相连。缺失值默认为NONE，可通过`fill_value`选择需要填充的值。

```
>>> df.extract_kv(columns=['kv'], kv_delim='=', fill_value=0)
   name   kv_k1  kv_k2  kv_k3  kv_k5  kv_k7  kv_k9
0  name1    1.0    3.0    0.0   10.0    0.0    0.0
1  name1    7.0    0.0    0.0    0.0    8.2    0.0
2  name2    0.0    1.2    1.5    0.0    0.0    0.0
3  name2    0.0    1.0    0.0    0.0    0.0    1.1
```

- 将多列数据转换为一个Key-Value列，示例数据如下。

```
>>> df
   name    k1   k2   k3    k5   k7   k9
0  name1  1.0  3.0  NaN  10.0  NaN  NaN
1  name1  7.0  NaN  NaN   NaN  8.2  NaN
2  name2  NaN  1.2  1.5   NaN  NaN  NaN
3  name2  NaN  1.0  NaN   NaN  NaN  1.1
```

通过`to_kv`方法转换为Key-Value表示的格式。

```
>>> df.to_kv(columns=['k1', 'k2', 'k3', 'k5', 'k7', 'k9'], kv_delim
='=')
    name                kv
0  name1  k1=1,k2=3,k5=10
1  name1    k1=7.1,k7=8.2
2  name2    k2=1.2,k3=1.5
```

```
3  name2      k9=1.1,k2=1
```

·

DataFrame创建及其对象操作请参见创建DataFrame。

8.5.12 数据合并

本文向您介绍DataFrame支持的数据表的JOIN操作、UNION操作等数据合并操作。

导入表数据，创建示例数据。

```
>>> from odps.df import DataFrame
>>> movies = DataFrame(o.get_table('pyodps_ml_100k_movies'))
>>> ratings = DataFrame(o.get_table('pyodps_ml_100k_ratings'))
>>> movies.dtypes
odps.Schema {
  movie_id                              int64
  title                                 string
  release_date                          string
  video_release_date                    string
  imdb_url                              string
}
>>> ratings.dtypes
odps.Schema {
  user_id                     int64
  movie_id                    int64
  rating                      int64
  unix_timestamp              int64
}
```

JOIN操作

DataFrame支持对两个Collection执行join的操作。

· 如果不指定join的条件，则DataFrame API会寻找名字相同的列，并作为join的条件。

```
>>> movies.join(ratings).head(3)
   movie_id              title  release_date  video_release_date
                                       imdb_url  user_id  rating
unix_timestamp
0         3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...       49       3
     888068877
1         3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...      621       5
     881444887
2         3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...      291       3
     874833936
```

· 显式指定join的条件。在join时，如果on条件在两表中所选取的字段名称相同，则只会保留其中一个表的此字段。

```
>>> movies.join(ratings, on='movie_id').head(3)
   movie_id              title  release_date  video_release_date
                                       imdb_url  user_id  rating
unix_timestamp
```

```
0         3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...       49       3
     888068877
1         3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...      621       5
     881444887
2         3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...      291       3
     874833936
```

- 在执行其他类型的join操作时，例如left_join，两表中相同名称的字段会被重命名。

```
>>> movies.left_join(ratings, on='movie_id').head(3)
   movie_id_x              title  release_date  video_release_date
                                        imdb_url  user_id
movie_id_y  rating  unix_timestamp
0           3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...       49
  3       3       888068877
1           3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...      621
  3       5       881444887
2           3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...      291
  3       3       874833936
```

示例中，movie_id被重命名为movie_id_x、movie_id_y，这与suffixes参数有关，该参数的默认值为('_x', '_y')。当遇到重名的列时，重名的列会被重命名为指定的后缀名。

```
>>> ratings2 = ratings[ratings.exclude('movie_id'), ratings.movie_id
.rename('movie_id2')]
>>> ratings2.dtypes
odps.Schema {
  user_id                      int64
  rating                       int64
  unix_timestamp               int64
  movie_id2                    int64
}
>>> movies.join(ratings2, on=[('movie_id', 'movie_id2')]).head(3)
   movie_id              title  release_date  video_release_date
                                        imdb_url  user_id  rating
unix_timestamp  movie_id2
0         3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...       49       3
     888068877          3
1         3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...      621       5
     881444887          3
2         3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...      291       3
     874833936          3
```

- 也可以在on中直接写等于表达式。

```
>>> movies.join(ratings2, on=[movies.movie_id == ratings2.movie_id2
]).head(3)
   movie_id              title  release_date  video_release_date
                                        imdb_url  user_id  rating
unix_timestamp  movie_id2
```

```
0         3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...       49       3
     888068877          3
1         3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...      621       5
     881444887          3
2         3  Four Rooms (1995)   01-Jan-1995
http://us.imdb.com/M/title-exact?Four%20Rooms%...      291       3
     874833936          3
```

- 在self-join时，可以调用view方法分别取字段。

```
>>> movies2 = movies.view()
>>> movies.join(movies2, movies.movie_id == movies2.movie_id)[movies
, movies2.movie_id.rename('movie_id2')].head(3)
   movie_id            title_x release_date_x video_release_date_x
 \
0         2   GoldenEye (1995)    01-Jan-1995                True
1         3  Four Rooms (1995)    01-Jan-1995                True
2         4  Get Shorty (1995)    01-Jan-1995                True

                                          imdb_url_x  movie_id2
0  http://us.imdb.com/M/title-exact?GoldenEye%20(...          2
1  http://us.imdb.com/M/title-exact?Four%20Rooms%...          3
2  http://us.imdb.com/M/title-exact?Get%20Shorty%...          4
```

除了支持join，DataFrame还支持left_join、right_join和outer_join。在执行上述外连接操作时，默认会为重名列加上_x和 _y后缀，可通过在suffixes参数中传入一个二元tuple进行自定义后缀。

- 如果需要在外连接中避免对位次相等的列取重复列，可以指定merge_columns选项。该选项会自动选择两列中的非空值作为新列的值。

```
>>> movies.left_join(ratings, on='movie_id', merge_columns=True)
```

使用mapjoin时，只需将mapjoin设为True，执行时会对右表做mapjoin操作。您也可以对分别来自ODPS和Pandas的Collection进行join，或者对分别来自ODPS和数据库的Collection进行join，该计算会在ODPS上执行。

UNION操作

如果两张表的字段和类型都一致（字段的顺序可以不同），则可以使用union或者concat将它们合并成一张表。

```
>>> mov1 = movies[movies.movie_id < 3]['movie_id', 'title']
>>> mov2 = movies[(movies.movie_id > 3) & (movies.movie_id < 6)]['
title', 'movie_id']
>>> mov1.union(mov2)
   movie_id              title
0         1   Toy Story (1995)
1         2   GoldenEye (1995)
2         4  Get Shorty (1995)
```

```
3         5     Copycat (1995)
```

您也可以对分别来自ODPS和Pandas的Collection进行union操作，或者对分别来自ODPS和数据库的Collection进行union操作，相关的计算会在ODPS上执行。

8.5.13 窗口函数

本文为您介绍DataFrame API支持使用窗口函数。

```
grouped = iris.groupby('name')
grouped.mutate(grouped.sepallength.cumsum(), grouped.sort('sepallength
').row_number()).head(10)
          name  sepallength_sum  row_number
0  Iris-setosa            250.3           1
1  Iris-setosa            250.3           2
2  Iris-setosa            250.3           3
3  Iris-setosa            250.3           4
4  Iris-setosa            250.3           5
5  Iris-setosa            250.3           6
6  Iris-setosa            250.3           7
7  Iris-setosa            250.3           8
8  Iris-setosa            250.3           9
9  Iris-setosa            250.3          10
```

- 窗口函数可以用在列选择中。

```
iris['name', 'sepallength', iris.groupby('name').sort('sepallength
').sepallength.cumcount()].head(5)
          name  sepallength  sepallength_count
0  Iris-setosa          4.3                  1
1  Iris-setosa          4.4                  2
2  Iris-setosa          4.4                  3
3  Iris-setosa          4.4                  4
4  Iris-setosa          4.5                  5
```

- 窗口函数按标量聚合时，与分组聚合的处理方式一致。

```
from odps.df import Scalar
iris.groupby(Scalar(1)).sort('sepallength').sepallength.cumcount()
```

DataFrame API支持的窗口函数如下。

窗口函数	说明
cumsum	计算累积和。
cummean	计算累积均值。
cummedian	计算累积中位数。
cumstd	计算累积标准差。
cummax	计算累积最大值。
cummin	计算累积最小值。
cumcount	计算累积和。

窗口函数	说明
lag	按偏移量取当前行之前第几行的值。如果当前行号为rn，则取行号为rn-offset的值。
lead	按偏移量取当前行之后第几行的值。如果当前行号为rn则取行号为rn+offset的值。
rank	计算排名。
dense_rank	计算连续排名。
percent_rank	计算一组数据中某行的相对排名。
row_number	计算行号，从1开始。
qcut	将分组数据按顺序切分成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。
nth_value	返回分组中的第n个值。
cume_dist	计算分组中值小于等于当前值的行数占分组总行数的比例。

各个窗口函数支持的参数说明如下。

- `rank`、`dense_rank`、`percent_rank`和`row_number`支持下列参数。

参数名	说明
sort	排序关键字，默认为空。
ascending	排序时，是否依照升序排序，默认值为True。

- `lag`和`lead`除了支持`rank`所支持的参数，还支持下列参数。

参数名	说明
offset	取数据的行距离当前行的行数。
default	当offset指定的行不存在时的返回值。

- `cumsum`、`cummax`、`cummin`、`cummean`、`cummedian`、`cumcount`和`cumstd`除了支持`rank`所支持的参数外，还支持下列参数。

参数名	说明
unique	是否排除重复值，默认False。
preceding	窗口范围起点。
following	窗口范围终点。

8.5.14 绘图

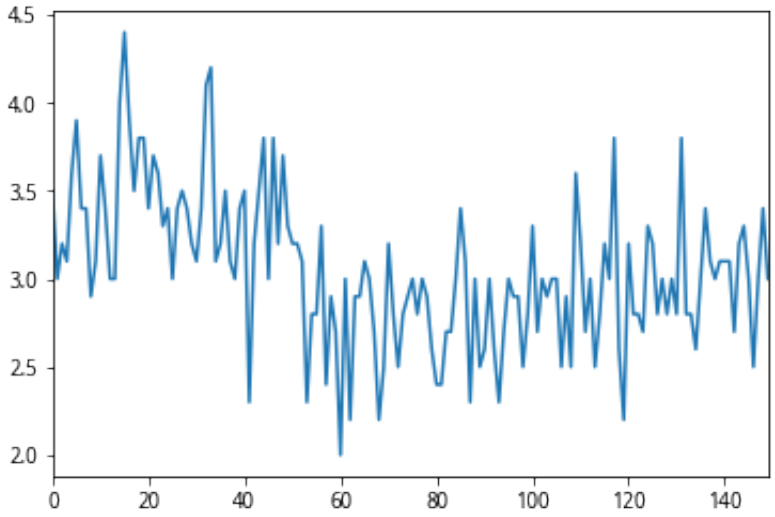
本文为您介绍PyODPS DataFrame提供的绘图方法。

如果您需要使用绘图，请先安装Pandas和Matplotlib。您可以在Jupyter中运行以下示例代码。

绘图

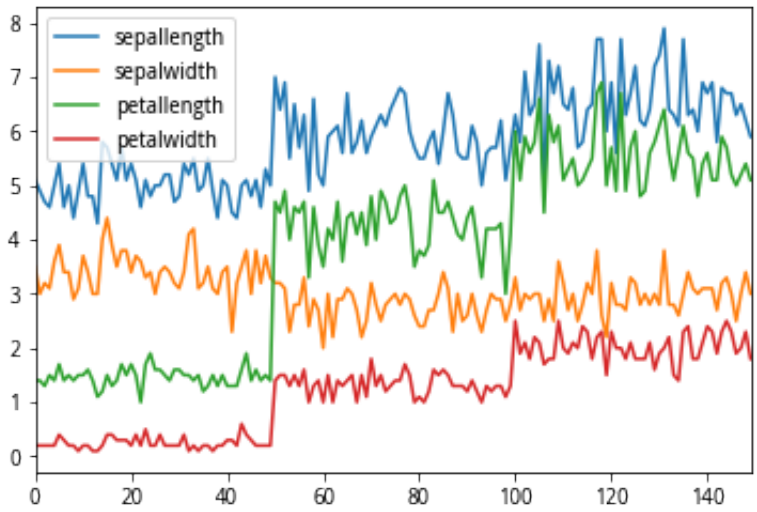
- 单线图

```
>>> from odps.df import DataFrame
>>> iris = DataFrame(o.get_table('pyodps_iris'))
>>> %matplotlib inline
>>> iris.sepalwidth.plot()
<matplotlib.axes._subplots.AxesSubplot at 0x10c2b3510>
```



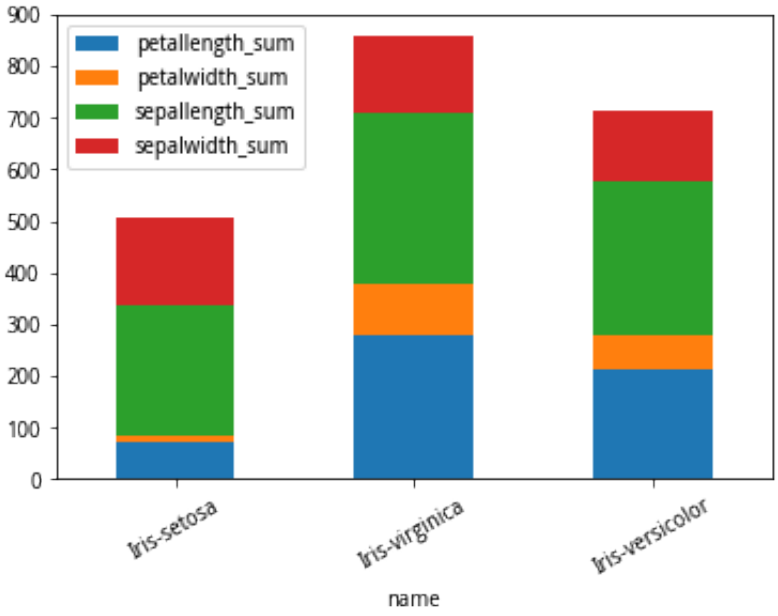
- 多线图

```
>>> iris.plot()
<matplotlib.axes._subplots.AxesSubplot at 0x10db7e690>
```



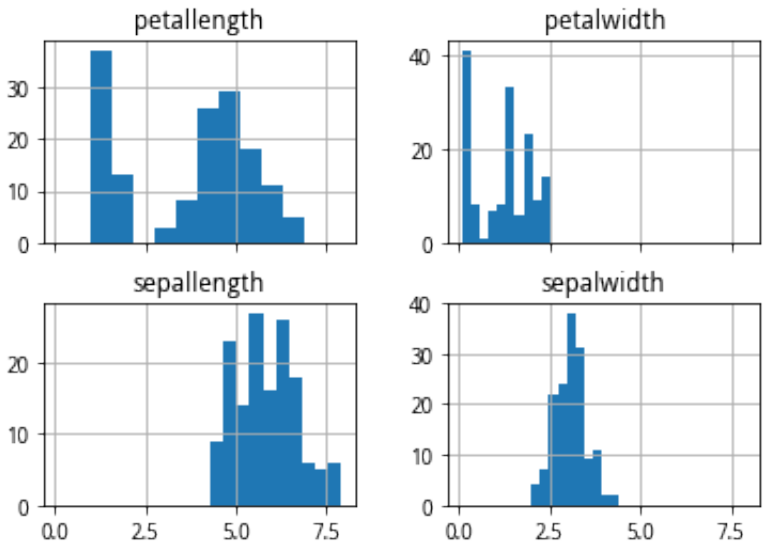
- **竖向柱状图**

```
>>> iris.groupby('name').sum().plot(kind='bar', x='name', stacked=
True, rot=30)
<matplotlib.axes._subplots.AxesSubplot at 0x10c5f2090>
```



- **直方图**

```
>>> iris.hist(sharex=True)
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x10e013f90
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x10e2d1c10
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x10e353f10
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x10e3c4410
>]], dtype=object)
```



参数`kind`表示了绘图的类型，支持的绘图类型如下表所示。

kind	说明
line	线图。
bar	竖向柱状图。
barh	横向柱状图。
hist	直方图。
box	box图。
kde	核密度估计。
density	和kde相同。
area	area图。
pie	饼图。
scatter	散点图。
hexbin	hexbin图。

详细的参数说明请参见pandas.DataFrame.plot。

除了上表所示的参数，`plot`函数还增加了以下几种参数，方便进行绘图。

参数	说明
xlabel	x轴名。
ylabel	y轴名。
xlabelsize	x轴名大小。
ylabelsize	y轴名大小。
labelsize	轴名大小。
title	标题。
titlesize	标题大小。
annotate	是否标记值。

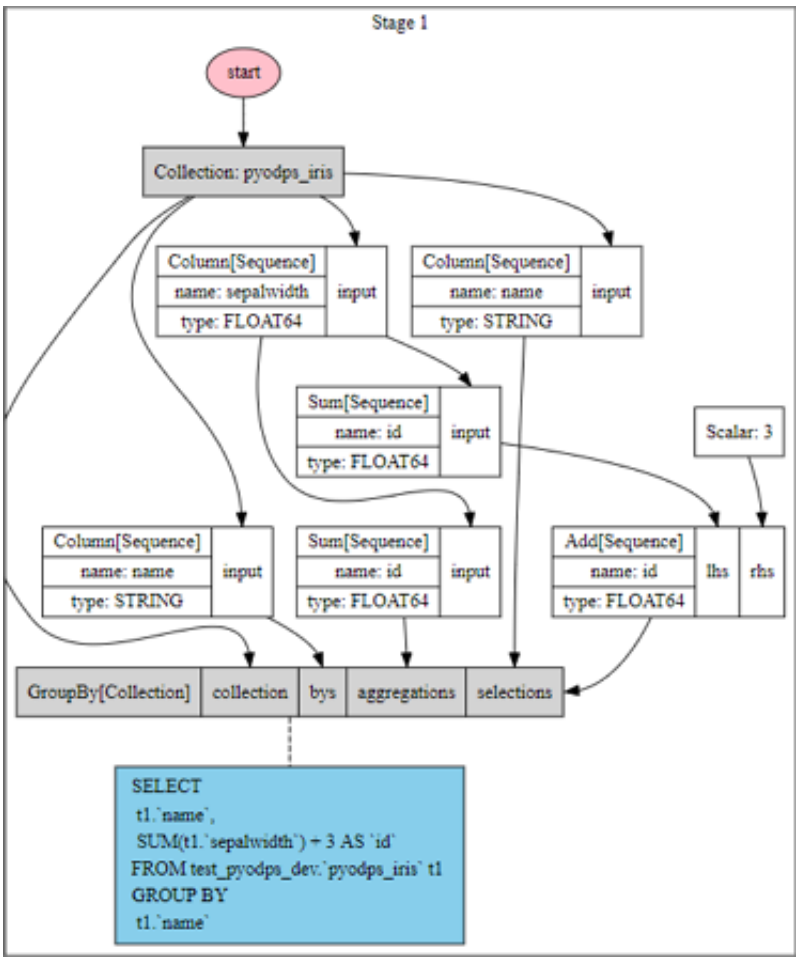
8.5.15 调试指南

由于PyODPS DataFrame本身会对整个操作执行优化，为了更直观地反应整个过程， 您可以使用可视化的方式显示整个表达式的计算过程。

可视化DataFrame

可视化需要依赖graphviz软件和*graphviz*Python包。

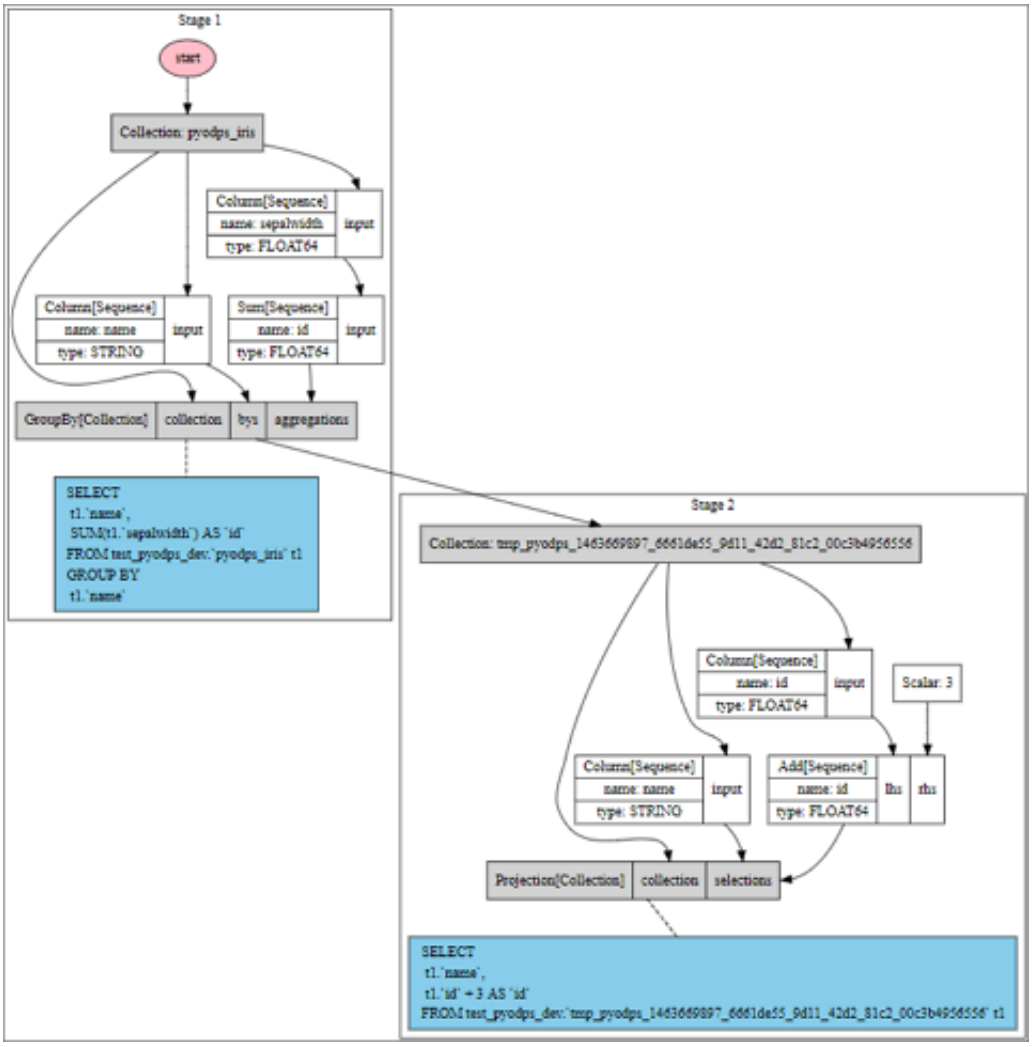
```
>>> df = iris.groupby('name').agg(id=iris.sepalwidth.sum())
>>> df = df[df.name, df.id + 3]
>>> df.visualize()
```



由上图可见，这个计算过程中，PyODPS DataFrame将`groupby`和列筛选做了操作合并。

```
>>> df = iris.groupby('name').agg(id=iris.sepalwidth.sum()).cache()
>>> df2 = df[df.name, df.id + 3]
```

```
>>> df2.visualize()
```



此时，由于执行了Cache操作，整个执行计划将会分成两步执行。

ODPS SQL后端查看编译结果

您可以直接调用`compile`方法查看ODPS SQL后端编译到SQL的结果。

```
>>> df = iris.groupby('name').agg(sepalwidth=iris.sepalwidth.max())
>>> df.compile()
Stage 1:

SQL compiled:

SELECT
  t1.`name`,
  MAX(t1.`sepalwidth`) AS `sepalwidth`
FROM test_pyodps_dev.`pyodps_iris` t1
GROUP BY
```

```
    t1.`name`
```

使用Pandas计算后端执行本地调试

对于来自ODPS表的DataFrame，部分操作不会编译到ODPS SQL执行，而会使用Tunnel下载表数据。这个下载的过程很快，且无需等待ODPS SQL任务的调度。利用这个特性，您可以快速下载小部分ODPS数据到本地，使用Pandas计算后端进行代码编写和调试。这些操作包括：

- 非分区表：对其进行选取全部或有限条数据、列筛选的操作（不包括列的各种计算），以及计算其数量。
- 不选取分区或筛选前几个分区字段的分区表：对其选取全部或有限条数据、列筛选的操作，以及计算其数量。

例如`iris`这个DataFrame来自非分区表，以下操作会使用Tunnel进行下载。

```
>>> iris.count()
>>> iris['name', 'sepalwidth'][:10]
```

例如有个DataFrame来自分区表（有三个分区字段，即`ds`、`hh`、`mm`），以下操作会使用Tunnel下载。

```
>>> df[:10]
>>> df[df.ds == '20160808']['f0', 'f1']
>>> df[(df.ds == '20160808') & (df.hh == 3)][:10]
>>> df[(df.ds == '20160808') & (df.hh == 3) & (df.mm == 15)]
```

因此您可以使用`to_pandas`方法将部分数据下载到本地进行调试。

```
>>> DEBUG = True
>>> if DEBUG:
>>> df = iris[:100].to_pandas(wrap=True)
>>> else:
>>> df = iris
```

最后把`DEBUG`设置为False，便可以在ODPS上执行完整的计算。

说明：

由于沙箱的限制，本地调试通过的程序不一定能在ODPS上也跑通。

8.6 交互体验增强

8.6.1 命令行增强

PyODPS提供了命令行下的增强工具。

您配置好帐号后，下次登录时无需再次输入帐号信息。 配置和调用的步骤如下。

1. 导入PyODPS的增强工具。

```
>>> from odps.inter import setup, enter, teardown
```

2. 配置帐号。

```
>>> setup('**your-access_id**', '**your-access-key**', '**your-
project**', endpoint='**your-endpoint**')
```

说明:

在不指定room这个参数时，会被配置到default的room里。

3. 之后，在任何命令行打开的地方，都可以直接调用。

```
>>> room = enter()
>>> o = room.odps
>>> o.get_table('dual')
odps.Table
  name: odps_test_sqltask_finance.`dual`
  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
```

说明:

在重新setup room后，ODPS入口对象并不会自动替换，需要再次调用enter()获得新的room对象。

在配置好账号及调用出对象后，您可以对room里的对象进行存储，取出，删除等操作，此外您也可以删除整个room。

- 常用的ODPS表或者资源都可以存放在room里。

```
>>> room.store('存储表示例', o.get_table('dual'), desc='简单的表存储示
例')
```

调用display方法，可将已经存储的对象以表格的形式打印出来。

```
>>> room.display()
default name          desc
存储表示例      简单的存储表示例
```

```
iris            安德森鸢尾花卉数据集
```

- 通过room['存储表示例']或room.iris命令，可以取出存储的对象。

```
>>> room['存储表示例']
odps.Table
  name: odps_test_sqltask_finance.`dual`
  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
```

- 调用drop方法可以对room内的对象执行删除操作。

```
>>> room.drop('存储表示例')
>>> room.display()
default name          desc
iris            安德森鸢尾花卉数据集
```

- 调用teardown()方法可以删除某个room。如果该方法没有传入的参数，则会删除默认的room。

```
teardown()
```

8.6.2 IPython增强

PyODPS提供了IPython的插件，来更方便得操作ODPS。

IPython增强

针对命令行增强，PyODPS也有相应的命令。

- 执行以下代码加载插件。

```
%load_ext odps
%enter
```

执行结果如下。

```
<odps.inter.Room at 0x11341df10>
```

- 此时全局会包含o和odps变量，即ODPS入口。您可以通过o.get_table或odps.get_table命令调用表。

```
o.get_table('dual')
```

```
odps.get_table('dual')
```

执行结果如下。

```
odps.Table
  name: odps_test_sqltask_finance.`dual`
  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
```

- 调用以下命令将已储存的对象以表格形式打印出来。

```
%stores
```

default name	desc
iris	安德森鸢尾花卉数据集

对象名补全

PyODPS拓展了IPython原有的代码补全功能，支持在书写类似`o.get_xxx`的语句时，自动补全对象名。以下示例中，<tab>不是实际输入的字符，而是当所有输入完成后，将光标移动到相应位置， 并按Tab键。

- 直接使用Tab键补全对象。

```
o.get_table(<tab>
```

- 如果已知需要补全对象的前缀，也可以在使用此功能时输入前缀。

```
o.get_table('tabl<tab>
```

输入上述语句并按Tab键后IPython会自动补全前缀为`tabl`的表。

- 对象名补全也支持补全不同Project下的对象名，支持的用法如下。

```
o.get_table(project='project_name', name='tabl<tab>
o.get_table('tabl<tab>', project='project_name')
```

- 如果匹配的对象有多个，IPython会给出一个列表。其最大长度由`options.completion_size`给出， 默认值为10。

SQL命令

PyODPS还提供了SQL插件，用于执行ODPS SQL。

- 您可以使用单个百分号输入是单行SQL命令。

```
In [37]: %sql select * from pyodps_iris limit 5
|==========================================|   1 /  1  (100.00%)
     3s
Out[37]:
   sepallength  sepalwidth  petallength  petalwidth         name
0          5.1         3.5          1.4         0.2  Iris-setosa
1          4.9         3.0          1.4         0.2  Iris-setosa
2          4.7         3.2          1.3         0.2  Iris-setosa
3          4.6         3.1          1.5         0.2  Iris-setosa
4          5.0         3.6          1.4         0.2  Iris-setosa
```

- 多行SQL可以使用`%%sql`命令。

```
In [38]: %%sql
   ....: select * from pyodps_iris
   ....: where sepallength < 5
   ....: limit 5
   ....:
|==========================================|   1 /  1  (100.00%)
    15s
Out[38]:
   sepallength  sepalwidth  petallength  petalwidth         name
0          4.9         3.0          1.4         0.2  Iris-setosa
1          4.7         3.2          1.3         0.2  Iris-setosa
2          4.6         3.1          1.5         0.2  Iris-setosa
3          4.6         3.4          1.4         0.3  Iris-setosa
4          4.4         2.9          1.4         0.2  Iris-setosa
```

- 如果您需要执行参数化SQL查询，则需以`:参数`的方式替换需要的参数。

```
In [1]: %load_ext odps

In [2]: mytable = 'dual'

In [3]: %sql select * from :mytable
|==========================================|   1 /  1  (100.00%)
     2s
Out[3]:
   c_int_a  c_int_b  c_double_a  c_double_b  c_string_a  c_string_b
c_bool_a  \
0        0        0       -1203           0           0       -1203
    True

  c_bool_b         c_datetime_a         c_datetime_b
0    False  2012-03-30 23:59:58  2012-03-30 23:59:59
```

- 设置SQL运行参数时，可以通过`%set`设置到全局，或者在SQL的cell里用SET进行局部设置。

  以下语句会进行局部设置，不会影响全局的配置。

```
In [17]: %%sql
         set odps.sql.mapper.split.size = 16;
```

```
        select * from pyodps_iris;
```

以下设置会进行全局设置，后续运行的SQL都会使用这个设置。

```
In [18]: %set odps.sql.mapper.split.size = 16
```

持久化Pandas DataFrame到ODPS表

PyODPS还提供把Pandas DataFrame上传到ODPS表的命令。您只需使用`%persist`命令即可，此命令的第1个参数`df`是前面的变量名，第2个参数`pyodps_pandas_df`是ODPS表名。

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.arange(9).reshape(3, 3), columns=list('abc'))
%persist df pyodps_pandas_df
```

8.6.3 Jupyter Notebook增强

PyODPS针对Jupyter Notebook下的探索性数据分析功能进行了增强， 包括结果探索功能以及进度展示功能。

结果探索

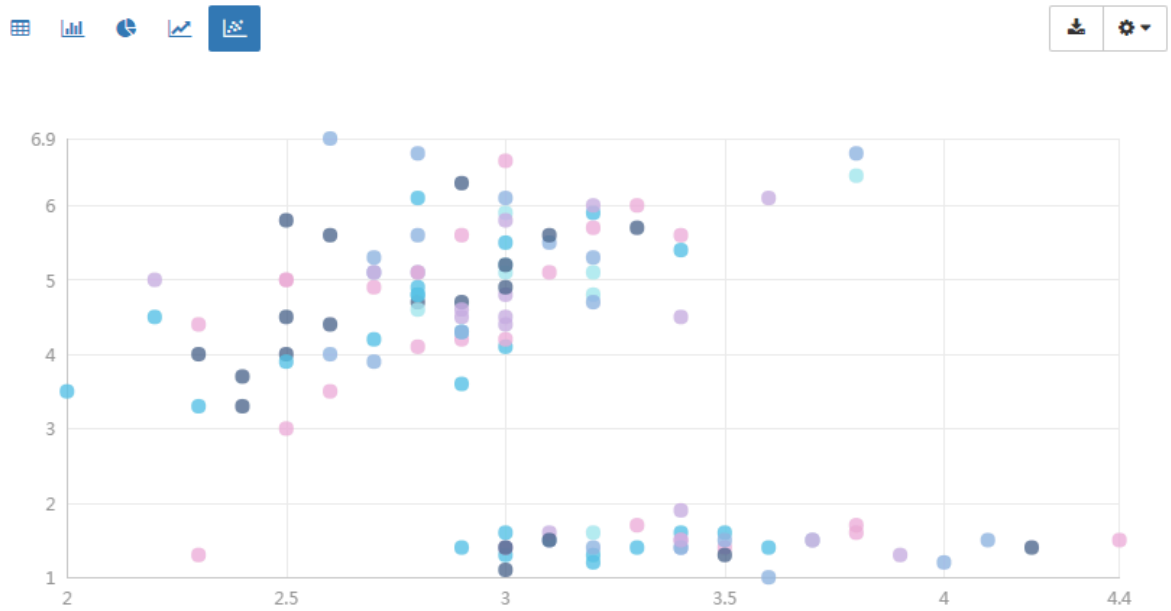
PyODPS在Jupyter Notebook中为SQL Cell和DataFrame提供了数据探索功能。对于已拉到本地的数据，可使用交互式的数据探索工具浏览数据，交互式地绘制图形。

1. 当执行结果为DataFrame时，PyODPS会读取执行结果，并以分页表格的形式展示出来。您可以单击页号或前进、后退按钮，在数据中导航。

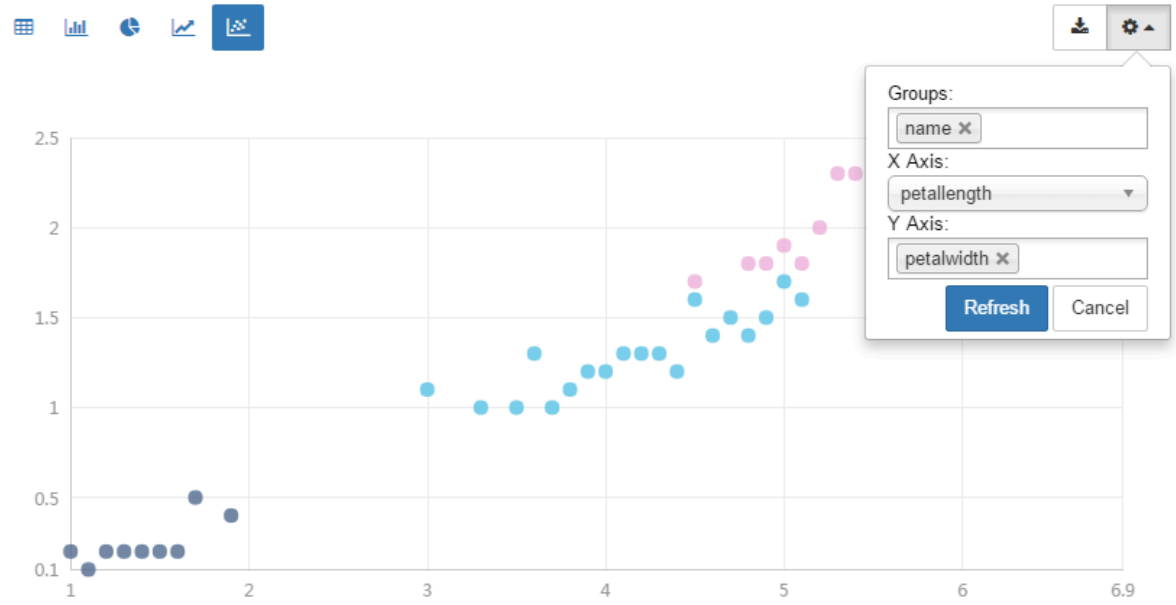
sepallength	sepalwidth	petallength	petalwidth	name
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa

« 1 2 3 4 5 6 7 ... 15 »

2. 结果区的顶端为模式选择区。除数据表外，也可以选择柱状图、饼图、折线图和散点图。使用默认的字段选择（即前三个字段）时， 绘制的散点图如下。

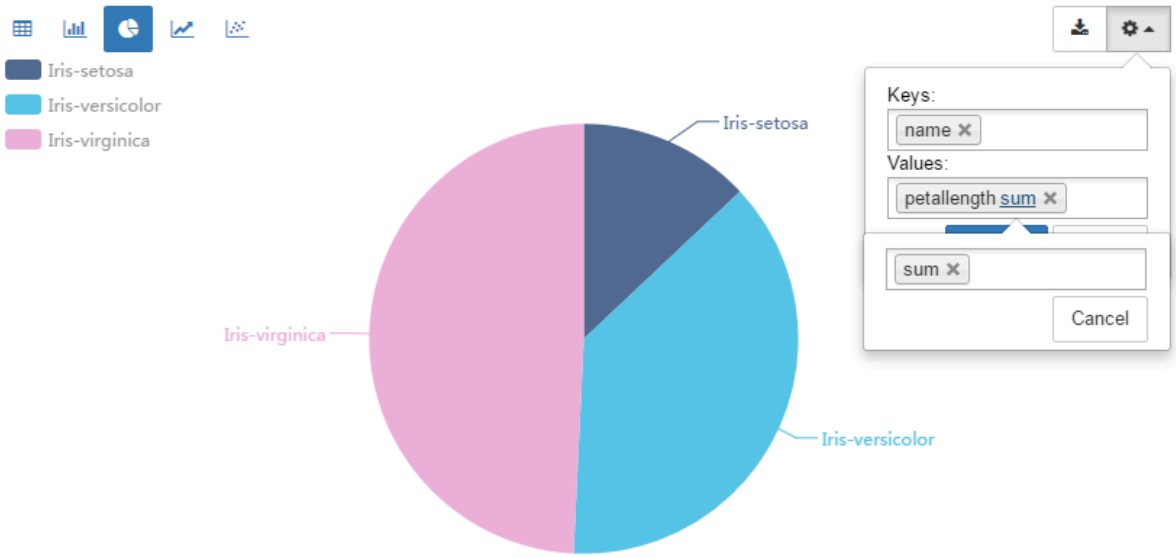


3. 在绘图模式下，单击右上角的配置按钮可以修改图表设置。将name设置为分组列，X轴选择为petallength，Y轴选择为petalwidth。由下图可见，在petallength-petalwidth维度下，数据对name有较好的区分度。



对于柱状图和饼图，值字段支持选择聚合函数。PyODPS对柱状图的默认聚合函数为sum，对饼图则为count。如需修改聚合函数， 可以在值字段名称后的聚合函数名上单击，选择所需的聚合函数。

对于折线图，需要避免X轴包含空值，否则可能导致图像不符合预期。



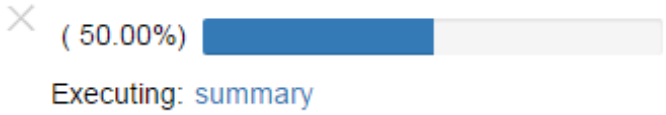
4. 完成绘图后，可单击下载保存绘制的图表。

> **说明:**
> 使用此功能前，您需要安装Pandas和ipywidgets。

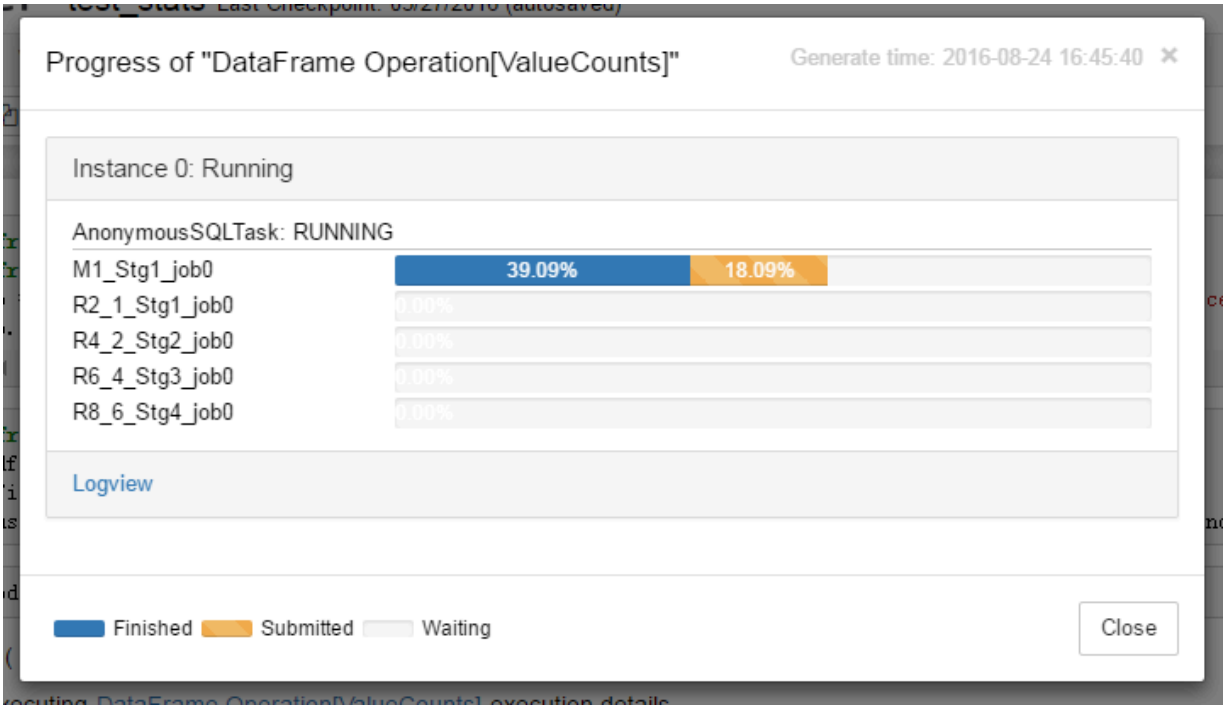
进度展示

大型作业通常需要执行较长的时间，因此PyODPS提供了进度展示功能。当在Jupyter Notebook中执行DataFrame、机器学习作业或通过执行`%sql`编写的SQL语句时，Jupyter Notebook会显示当前正在执行的作业列表及总体进度。

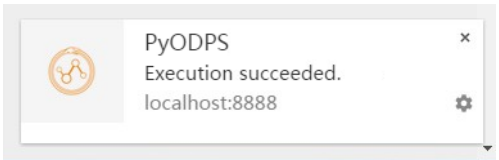
```
In [*]: from odps import ODPS
from odps.df.examples import create_ionosphere
o = ODPS(access_id, secret_access_key, project=project, endpoint=endpoint)
df = create_ionosphere(o)['a01', 'a02', 'a03', 'a04', 'class']
df.calc_summary()
```



当点击某个作业名称上的链接时，会弹出一个对话框，显示该作业中每个Task的具体执行进度。



当作业运行成功后，将弹出作业是否成功的提示信息。



8.7 配置选项

本文为您介绍PyODPS提供的配置选项。

您可以通过`odps.options`获得PyODPS提供的配置选项。

```
from odps import options
# 设置所有输出表的生命周期（lifecycle选项）。
options.lifecycle = 30
# 使用Tunnel下载string类型时使用bytes（tunnel.string_as_binary选项）。
options.tunnel.string_as_binary = True
# 用ODPS执行PyODPS DataFrame时，可以参照下面dataframe相关配置，在sort时将limit设置为一个比较大的值。
options.df.odps.sort.limit = 100000000
```

通用配置

选项	说明	默认值
end_point	ODPS 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
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
api_proxy	API代理服务器。	None
data_proxy	数据代理服务器。	None
completion_size	对象补全列举条数限制。	10

选项	说明	默认值
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MaxCompute 2.0语言扩展。	False

数据上传下载配置

选项	说明	默认值
tunnel.endpoint	Tunnel的Endpoint。	None
tunnel.use_instance_tunnel	使用Instance Tunnel获取执行结果。	True
tunnel.limit_instance_tunnel	是否限制Instance Tunnel获取结果的条数。	None
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
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
df.supersede_libraries	使用自行上传的Numpy替换服务中的版本。	False

选项	说明	默认值
df.odps.sort.limit	DataFrame有排序操作时，默认添加的Limit条数。	10000

机器学习配置

选项	说明	默认值
ml.xflow_settings	Xflow执行配置。	None
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。	False
ml.model_volume	在使用ModelTransfer时使用的 Volume名称。	pyodps_volume

8.8 API概述

本文为您提供了PyODPS API文档链接，其中包含各函数的参数说明及示例。

- ODPS详解（Definitions）
- PyODPS DataFrame指南（DataFrame Reference）

8.9 常见问题与最佳实践

本文为您提供PyODPS的最佳实践及常见问题，方便您更高效地开发PyODPS程序。

常见问题

PyODPS安装、使用、开发过程中常见问题列表请参见PyODPS常见问题列表：

- PyODPS安装常见问题
- PyODPS开发常见问题
- PyODPS使用常见问题

PyODPS DataFrame处理笛卡尔积的方式

请参见云栖社区了解如何使用PyODPS DataFrame API执行笛卡尔积的操作。

DataWorks PyODPS节点实现结巴中文分词

请参见PyODPS节点实现结巴中文分词了解如何在DataWorks中使用PyODPS节点实现结巴中文分词。

PyODPS实现SQL的循环

请参见云栖社区了解如何通过使用Python内嵌SQL的方式实现SQL的循环。

Pandas计算后端进行高效本地Debug

请参见云栖社区了解如何使用Pandas计算后端进行高效本地Debug。

说明:

如果您对以上参考内容有任何疑问，请直接在云栖社区的相关文章中进行交流。

9 Spark

9.1 Spark概述

MaxCompute Spark是MaxCompute提供的兼容开源的Spark计算服务。它在统一的计算资源和数据集权限体系之上，提供Spark计算框架，支持用户以熟悉的开发使用方式提交运行Spark作业，以满足更丰富的数据处理分析场景。

关键特性

- 支持原生多版本Spark作业

  社区原生Spark运行在MaxCompute里，完全兼容Spark的API，支持多个Spark版本同时运行。MaxCompute Spark提供原生的Spark WebUI供用户查看。
- 统一的计算资源

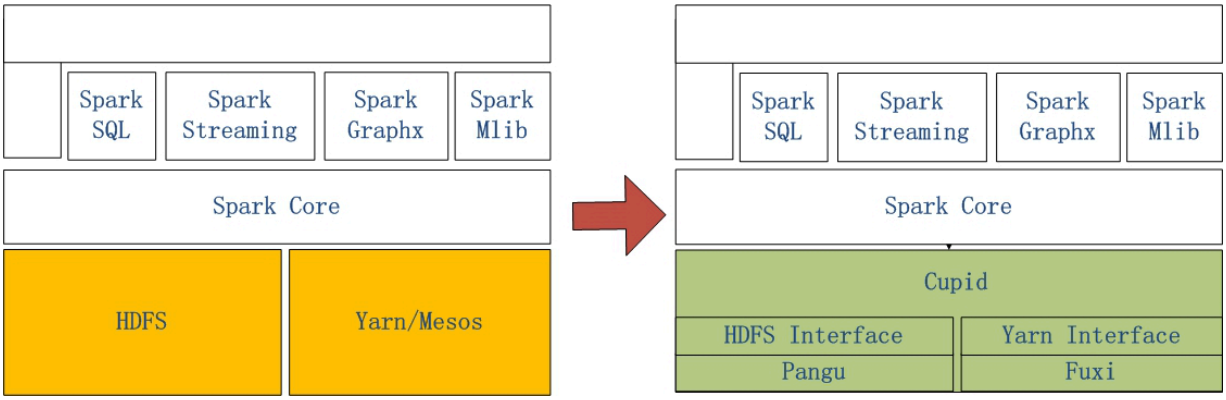
  MaxCompute Spark像MaxCompute SQL/MR等任务类型一样，运行在MaxCompute项目开通的统一计算资源中。
- 统一的数据和权限管理

  完全遵循MaxCompute项目的权限体系，在访问用户权限范围内安全地查询数据。
- 与开源系统相同的使用体验

  MaxCompute Spark与社区开源Spark保持相同的体验（例如开源应用的UI界面、在线交互等），完全符合Spark用户使用习惯。开源应用的调试过程中需要使用开源UI，MaxCompute Spark提供原生的开源实时UI和查询历史日志的功能。其中，对于部分开源应用还支持交互式体验，在后台引擎运行后即可进行实时交互。

系统结构

MaxCompute Spark是阿里云通过Spark on MaxCompute的解决方案，让原生Spark能够运行在MaxCompute当中。



左侧是原生Spark的架构图，右边Spark on MaxCompute运行在阿里云自研的Cupid的平台之上，该平台可以原生支持开源社区Yarn所支持的计算框架，如Spark等。

约束与限制

目前MaxCompute Spark支持以下适用场景：

- Java/Scala所有离线场景，GraphX、Mllib、RDD、Spark-SQL、PySpark等。
- 读写MaxCompute Table。
- OSS非结构化存储支持。

暂不支持以下场景（后续版本将陆续支持）：

- 读写VPC环境下的服务，如RDS、Redis、ECS上部署的服务等。
- Streaming场景。
- 交互式类需求Spark-Shell、Spark-SQL-Shell、PySpark-Shell等。

9.2 搭建开发环境

本文为您介绍如何搭建MaxCompute Spark开发环境。

下载MaxCompute Spark客户端

MaxCompute Spark发布包集成了MaxCompute认证功能。作为客户端工具，它用于通过Spark-submit方式提交作业到MaxCompute项目中运行。目前提供了面向Spark1.x和Spark2.x的2个发布包：

- spark-1.6.3
- spark-2.3.0

如果您需要开发Spark1.x应用，请使用spark-1.6.3版本客户端；如果您需要开发Spark2.x应用，请使用spark-2.3.0版本客户端。

设置环境变量

- JAVA_HOME设置

```
# 尽量使用JDK 1.7+，1.8+最佳。
export JAVA_HOME=/path/to/jdk
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
export PATH=$JAVA_HOME/bin:$PATH
```

- SPARK_HOME设置

```
export SPARK_HOME=/path/to/spark_extracted_package
export PATH=$SPARK_HOME/bin:$PATH
```

- PySpark的用户请安装Python2.7版本，并设置PATH。

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

配置*Spark-defaults.conf*

*$SPARK_HOME/conf*路径下存在*spark-defaults.conf.template*文件，它可以作为*spark-defaults.conf*的模版。您需要在该文件中设置MaxCompute相关的账号及服务域名后，才可以提交Spark任务到MaxCompute。默认配置内容如下，将空白处填入实际的账号信息，其余的配置信息可保持不变。

```
# 配置MaxCompute项目及访问账号信息。
spark.hadoop.odps.project.name = XXX
spark.hadoop.odps.access.id = XXX
spark.hadoop.odps.access.key = XXX

# 配置Spark客户端连接访问MaxCompute项目的Endpoint，请根据网络环境以及Region填写对应MaxCompute Endpoint。
spark.hadoop.odps.end.point = http://service.cn.maxcompute.aliyun.com/api
# Spark运行环境endpoint，请配置为所在region的MaxCompute VPC内网Endpoint。
spark.hadoop.odps.runtime.end.point = http://service.cn.maxcompute.aliyun-inc.com/api

# 以下配置保持不变。
spark.sql.catalogImplementation=odps
spark.hadoop.odps.task.major.version = cupid_v2
spark.hadoop.odps.cupid.container.image.enable = true
spark.hadoop.odps.cupid.container.vm.engine.type = hyper
```

配置依赖

- 配置访问MaxCompute表所需的依赖

  Spark作业访问MaxCompute表，需要依赖odps-spark-datasource模块。Maven坐标如下。

```
<!-- Spark-2.x请依赖此模块 -->
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-spark-datasource_2.11</artifactId>
```

```
    <version>3.3.3-public</version>
</dependency>

<!-- Spark-1.x请依赖此模块 -->
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-spark-datasource_2.10</artifactId>
  <version>3.3.3-public</version>
</dependency>
```

- 配置访问OSS所需的依赖

  如果作业需要访问OSS，直接添加以下依赖即可。

```
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>hadoop-fs-oss</artifactId>
    <version>3.3.3-public</version>
</dependency>
```

9.3 应用开发示例

9.3.1 Java/Scala开发示例

本文为您介绍Java/Scala的开发示例。

下载示例工程

您可以执行如下指令下载示例工程。

```
git clone git@github.com:aliyun/aliyun-cupid-sdk.git
cd aliyun-cupid-sdk
git checkout 3.3.3-public
# Spark-2.x示例工程目录
cd spark/spark-2.x/spark-examples
# Spark-1.x示例工程目录
cd spark/spark-1.x/spark-examples
# 打包，将在target目录生成shaded jar包。
mvn clean package
```

配置Spark-1.x的依赖

*pom.xml*须知：用户构建Spark应用时，通过MaxCompute提供的Spark客户端提交应用，因此需要添加以下依赖。

```
<properties>
    <spark.version>1.6.3</spark.version>
    <cupid.sdk.version>3.3.3-public</cupid.sdk.version>
    <scala.version>2.10.4</scala.version>
    <scala.binary.version>2.10</scala.binary.version>
</properties>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
```

```
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-mllib_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-streaming_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>cupid-sdk</artifactId>
    <version>${cupid.sdk.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>hadoop-fs-oss</artifactId>
    <version>${cupid.sdk.version}</version>
</dependency>
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-spark-datasource_${scala.binary.version}</
artifactId>
    <version>${cupid.sdk.version}</version>
</dependency>
<dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-library</artifactId>
    <version>${scala.version}</version>
</dependency>
<dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-actors</artifactId>
    <version>${scala.version}</version>
</dependency>
```

说明:

scope的定义：

- **spark-core、spark-sql等所有Spark社区发布的包，使用`provided` scope。**
- **odps-spark-datasource使用默认的`compile` scope。**

Spark-1.x案例说明

- WordCount

  详细代码

  提交方式

```
Step 1. build aliyun-cupid-sdk
Step 2. properly set spark.defaults.conf
Step 3. bin/spark-submit --master yarn-cluster --class \
      com.aliyun.odps.spark.examples.WordCount \
      ${path to aliyun-cupid-sdk}/spark/spark-1.x/spark-examples/
target/spark-examples_2.10-version-shaded.jar
```

- Spark-SQL on MaxCompute Table

  详细代码

  提交方式

```
# 运行可能会报Table Not Found的异常，这是因为用户的MaxCompute Project中没有
代码中指定的表。
# 可参考代码中的各种接口，实现对应Table的SparkSQL应用。
Step 1. build aliyun-cupid-sdk
Step 2. properly set spark.defaults.conf
Step 3. bin/spark-submit --master yarn-cluster --class \
      com.aliyun.odps.spark.examples.sparksql.SparkSQL \
      ${path to aliyun-cupid-sdk}/spark/spark-1.x/spark-examples/
target/spark-examples_2.10-version-shaded.jar
```

- GraphX PageRank

  详细代码

  提交方式

```
Step 1. build aliyun-cupid-sdk
Step 2. properly set spark.defaults.conf
Step 3. bin/spark-submit --master yarn-cluster --class \
      com.aliyun.odps.spark.examples.graphx.PageRank \
      ${path to aliyun-cupid-sdk}/spark/spark-1.x/spark-examples/
target/spark-examples_2.10-version-shaded.jar
```

- Mllib Kmeans-ON-OSS

  详细代码

  提交方式

```
# 在代码中填入OSS账号信息，再编译提交。
conf.set("spark.hadoop.fs.oss.accessKeyId", "***")
conf.set("spark.hadoop.fs.oss.accessKeySecret", "***")
conf.set("spark.hadoop.fs.oss.endpoint", "oss-cn-hangzhou-zmf.
aliyuncs.com")
Step 1. build aliyun-cupid-sdk
Step 2. properly set spark.defaults.conf
Step 3. bin/spark-submit --master yarn-cluster --class \
```

```
      com.aliyun.odps.spark.examples.mllib.KmeansModelSaveToOss \
      ${path to aliyun-cupid-sdk}/spark/spark-1.x/spark-examples/
target/spark-examples_2.10-version-shaded.jar
```

- **OSS UnstructuredData**

  详细代码

  提交方式

```
# 在代码中填入OSS账号信息，再编译提交。
conf.set("spark.hadoop.fs.oss.accessKeyId", "***")
conf.set("spark.hadoop.fs.oss.accessKeySecret", "***")
conf.set("spark.hadoop.fs.oss.endpoint", "oss-cn-hangzhou-zmf.
aliyuncs.com")
Step 1. build aliyun-cupid-sdk
Step 2. properly set spark.defaults.conf
Step 3. bin/spark-submit --master yarn-cluster --class \
      com.aliyun.odps.spark.examples.oss.SparkUnstructuredDat
aCompute \
      ${path to aliyun-cupid-sdk}/spark/spark-1.x/spark-examples/
target/spark-examples_2.10-version-shaded.jar
```

配置Spark-2.x的依赖

*pom.xml*须知：用户构建Spark应用时，通过MaxCompute提供的Spark客户端提交应用，因此需要添加以下依赖。

```
<properties>
    <spark.version>2.3.0</spark.version>
    <cupid.sdk.version>3.3.3-public</cupid.sdk.version>
    <scala.version>2.11.8</scala.version>
    <scala.binary.version>2.11</scala.binary.version>
</properties>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-mllib_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-streaming_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>com.aliyun.odps</groupId>
```

```
    <artifactId>cupid-sdk</artifactId>
    <version>${cupid.sdk.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>hadoop-fs-oss</artifactId>
    <version>${cupid.sdk.version}</version>
</dependency>
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-spark-datasource_${scala.binary.version}</
artifactId>
    <version>${cupid.sdk.version}</version>
</dependency>
<dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-library</artifactId>
    <version>${scala.version}</version>
</dependency>
<dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-actors</artifactId>
    <version>${scala.version}</version>
</dependency>
```

说明:

scope的定义:

- **spark-core、spark-sql等所有Spark社区发布的包，使用**`provided`**scope。**
- **odps-spark-datasource使用默认的**`compile`**scope。**

Spark-2.x案例说明

- **WordCount**

  详细代码

  提交方式

```
Step 1. build aliyun-cupid-sdk
Step 2. properly set spark.defaults.conf
Step 3. bin/spark-submit --master yarn-cluster --class \
      com.aliyun.odps.spark.examples.WordCount \
      ${path to aliyun-cupid-sdk}/spark/spark-2.x/spark-examples/
target/spark-examples_2.11-version-shaded.jar
```

- **Spark-SQL on MaxCompute Table**

  详细代码

  提交方式

```
# 运行可能会报Table Not Found的异常，因为用户的MaxCompute Project中没有代码
中指定的表。
# 可以参考代码中的各种接口，实现对应Table的SparkSQL应用。
Step 1. build aliyun-cupid-sdk
```

```
Step 2. properly set spark.defaults.conf
Step 3. bin/spark-submit --master yarn-cluster --class \
      com.aliyun.odps.spark.examples.sparksql.SparkSQL \
      ${path to aliyun-cupid-sdk}/spark/spark-2.x/spark-examples/
target/spark-examples_2.11-version-shaded.jar
```

- **GraphX PageRank**

  详细代码

  提交方式

```
Step 1. build aliyun-cupid-sdk
Step 2. properly set spark.defaults.conf
Step 3. bin/spark-submit --master yarn-cluster --class \
      com.aliyun.odps.spark.examples.graphx.PageRank \
      ${path to aliyun-cupid-sdk}/spark/spark-2.x/spark-examples/
target/spark-examples_2.11-version-shaded.jar
```

- **Mllib Kmeans-ON-OSS**

  详细代码

  提交方式

```
# 在代码中填入OSS账号信息，再编译提交。
val spark = SparkSession
      .builder()
      .config("spark.hadoop.fs.oss.accessKeyId", "***")
      .config("spark.hadoop.fs.oss.accessKeySecret", "***")
      .config("spark.hadoop.fs.oss.endpoint", "oss-cn-hangzhou-zmf.
aliyuncs.com")
      .appName("KmeansModelSaveToOss")
      .getOrCreate()
Step 1. build aliyun-cupid-sdk
Step 2. properly set spark.defaults.conf
Step 3. bin/spark-submit --master yarn-cluster --class \
      com.aliyun.odps.spark.examples.mllib.KmeansModelSaveToOss \
      ${path to aliyun-cupid-sdk}/spark/spark-2.x/spark-examples/
target/spark-examples_2.11-version-shaded.jar
```

- **OSS UnstructuredData**

  详细代码

  提交方式

```
# 在代码中填入OSS账号信息，再编译提交。
val spark = SparkSession
      .builder()
      .config("spark.hadoop.fs.oss.accessKeyId", "***")
      .config("spark.hadoop.fs.oss.accessKeySecret", "***")
      .config("spark.hadoop.fs.oss.endpoint", "oss-cn-hangzhou-zmf.
aliyuncs.com")
      .appName("SparkUnstructuredDataCompute")
      .getOrCreate()
Step 1. build aliyun-cupid-sdk
Step 2. properly set spark.defaults.conf
Step 3. bin/spark-submit --master yarn-cluster --class \
```

```
      com.aliyun.odps.spark.examples.oss.SparkUnstructuredDat
aCompute \
      ${path to aliyun-cupid-sdk}/spark/spark-2.x/spark-examples/
target/spark-examples_2.11-version-shaded.jar
```

9.3.2 PySpark开发示例

本文为您介绍PySpark开发示例。

若需要访问MaxCompute表，则需要编译datasource包。具体步骤请参见配置依赖。

SparkSQL应用示例（Spark1.6）

详细代码

```
from pyspark import SparkContext, SparkConf
from pyspark.sql import OdpsContext
if __name__ == '__main__':
    conf = SparkConf().setAppName("odps_pyspark")
    sc = SparkContext(conf=conf)
    sql_context = OdpsContext(sc)
    sql_context.sql("DROP TABLE IF EXISTS spark_sql_test_table")
    sql_context.sql("CREATE TABLE spark_sql_test_table(name STRING,
num BIGINT)")
    sql_context.sql("INSERT INTO TABLE spark_sql_test_table SELECT '
abc', 100000")
    sql_context.sql("SELECT * FROM spark_sql_test_table").show()
    sql_context.sql("SELECT COUNT(*) FROM spark_sql_test_table").show
()
```

提交运行

```
./bin/spark-submit \
--jars cupid/odps-spark-datasource_xxx.jar \
example.py
```

SparkSQL应用示例（Spark2.3）

详细代码

```
from pyspark.sql import SparkSession
if __name__ == '__main__':
    spark = SparkSession.builder.appName("spark sql").getOrCreate()
    spark.sql("DROP TABLE IF EXISTS spark_sql_test_table")
    spark.sql("CREATE TABLE spark_sql_test_table(name STRING, num
BIGINT)")
    spark.sql("INSERT INTO spark_sql_test_table SELECT 'abc', 100000")
    spark.sql("SELECT * FROM spark_sql_test_table").show()
    spark.sql("SELECT COUNT(*) FROM spark_sql_test_table").show()
```

Cluster模式提交运行

```
spark-submit --master yarn-cluster \
--jars cupid/odps-spark-datasource_xxx.jar \
```

```
example.py
```

Local模式运行

```
cd $SPARK_HOME
./bin/spark-submit --master local[4] \
--driver-class-path cupid/odps-spark-datasource_xxx.jar \
/path/to/odps-spark-examples/spark-examples/src/main/python/spark_sql.
py
```

说明:

- Local模式访问表依赖Tunnel，需要在*Spark-defaults.conf*配置tunnel_end_point配置项的内容。
- Local模式要用--driver-class-path而非--jars。

Package依赖

由于MaxCompute集群无法自由安装Python库，PySpark依赖其它Python库/插件/项目时，通常需在本地打包后通过Spark-submit上传。对于特定依赖，打包环境需与线上环境保持一致。您可根据依赖复杂度使用以下两种打包方式。

- 本地打egg包

  例如mllib需要numpy和setuptool两个插件，需要制作对应的egg包，通过--py-files上传。 具体步骤如下:

  说明:

  由于运行Spark作业的线上环境是Python2.7，所以本地必须使用Python2.7打包。

  1. 制作numpy、setuptools egg包。具体操作步骤如下:
     a. 下载numpy和setuptools安装包。
     b. 进入setuptools源码路径，执行Python setup.py bdist_egg后，会在dist目录生成egg文件。
     c. 进入numpy源码路径，执行Python setupeggs.py bdist_egg后，会在dist目录生成egg文件。
  2. 提交Spark作业

     ```
     cd $SPARK_HOME
     ./bin/spark-submit --master yarn-cluster \
     --jars cupid/odps-spark-datasource_2.11-3.3.2-hotfix1.jar \
     --py-files /path/to/numpy-1.7.1-py2.7-lunux-x85_64.egg,/path/to/
     setuptools-33.1.1-py2.7.egg \
     ```

```
app.py
```

- **本地打Python包**

  如果依赖链太长，导致打egg包无法满足需求，或依赖包含so文件等无法通过zipimport引入的文件，则需要把所有依赖下载后打Python包，再把整个Python包上传。 具体步骤如下：

  1. 添加配置

  ```
  spark.pyspark.python=./public.python-2.7-ucs4.zip/python-2.7-ucs4/bin/python2.7
  ```

  2. 提交Spark作业

  ```
  cd $SPARK_HOME
  ./bin/spark-submit --master yarn-cluster \
  --jars cupid/odps-spark-datasource_2.11-3.3.2-hotfix1.jar \
  --archives ./python-2.7-ucs4.zip app.py
  ```

  如果您不想通过`--archives`的方式提交，也可以通过公共资源的方式提交。

  1. 添加配置

  ```
  spark.hadoop.odps.cupid.resources=public.python-2.7-ucs4.zip
  spark.pyspark.python=./public.python-2.7-ucs4.zip/python-2.7-ucs4/bin/python2.7
  ```

  2. 提交Spark作业

  ```
  cd $SPARK_HOME
  ./bin/spark-submit --master yarn-cluster \
  --jars cupid/odps-spark-datasource_2.11-3.3.2-hotfix1.jar app.py
  ```

  如果有其他依赖，可参考以下脚本进行自主打包。

  ```
  work_root=`dirname $0`
  work_root=`cd ${work_root}; pwd`
  # Step 1 compile python
  # 1.1 python source code
  cd ${work_root}
  if [ ! -f Python-2.7.13.tgz ]; then
      wget https://www.python.org/ftp/python/2.7.13/Python-2.7.13.tgz
  fi
  # 1.2 configure && make && make install
  if [ ! -d ${work_root}/Python-2.7.13 ]; then
      cd ${work_root}
      tar xf ${work_root}/Python-2.7.13.tgz
  fi
  if [ -d ${work_root}/python-2.7-ucs4 ]; then
      rm -rf ${work_root}/python-2.7-ucs4
  fi
  cd ${work_root}/Python-2.7.13
  ./configure --prefix=${work_root}/python-2.7-ucs4 --enable-unicode=ucs4
  sed -i 's/#.*zlib zlibmodule.c/zlib zlibmodule.c/g' Modules/Setup
  make -j20
  make install
  # 1.3 install pip
  ```

```
cd ${work_root}
if [ ! -f get-pip.py ]; then
    curl -s https://bootstrap.pypa.io/get-pip.py -o ${work_root}/get
-pip.py
fi
${work_root}/python-2.7-ucs4/bin/python ${work_root}/get-pip.py
# 1.4 install numpy
${work_root}/python-2.7-ucs4/bin/pip install numpy
# 1.6 make python zip
if [ -f ${work_root}/python-2.7-ucs4.zip ]; then
    rm -rf ${work_root}/python-2.7-ucs4.zip
fi
cd ${work_root}
zip -r ${work_root}/python-2.7-ucs4.zip python-2.7-ucs4
```

9.4 运行模式

目前MaxCompute Spark支持以下三种运行方式：Local模式、Cluster模式和在DataWorks中执行模式。

Local模式

Local模式主要是为方便用户调试应用代码。在Local模式下，MaxCompute Spark的使用方式与社区相同，还添加了用Tunnel读写MaxCompute表的功能。您可以在IDE和命令行中使用该模式需要添加配置`spark.master=local[N]`，其中N表示执行该模式所需的CPU资源。Local模式下读写表是通过读写Tunnel完成的，需要在*Spark-defaults.conf*中增加Tunnel配置项（请根据MaxCompute项目所在的Region及网络环境填写对应的Endpoint地址，详情请参见配置Endpoint）。命令行执行该模式的方式如下。

```
1.bin/spark-submit --master local[4] \
--class com.aliyun.odps.spark.examples.SparkPi \
${path to aliyun-cupid-sdk}/spark/spark-2.x/spark-examples/target/
spark-examples_2.11-version-shaded.jar
```

Cluster模式

在Cluster模式中，您需要指定自定义程序入口main。main结束（Success or Fail）时，对应的Spark作业就会结束。使用场景适合于离线作业，可与阿里云DataWorks产品结合进行作业调度。命令行提交方式如下。

```
1.bin/spark-submit --master yarn-cluster \
-class SparkPi \
${ProjectRoot}/spark/spark-2.x/spark-examples/target/spark-examples_2.
11-version-shaded.jar
```

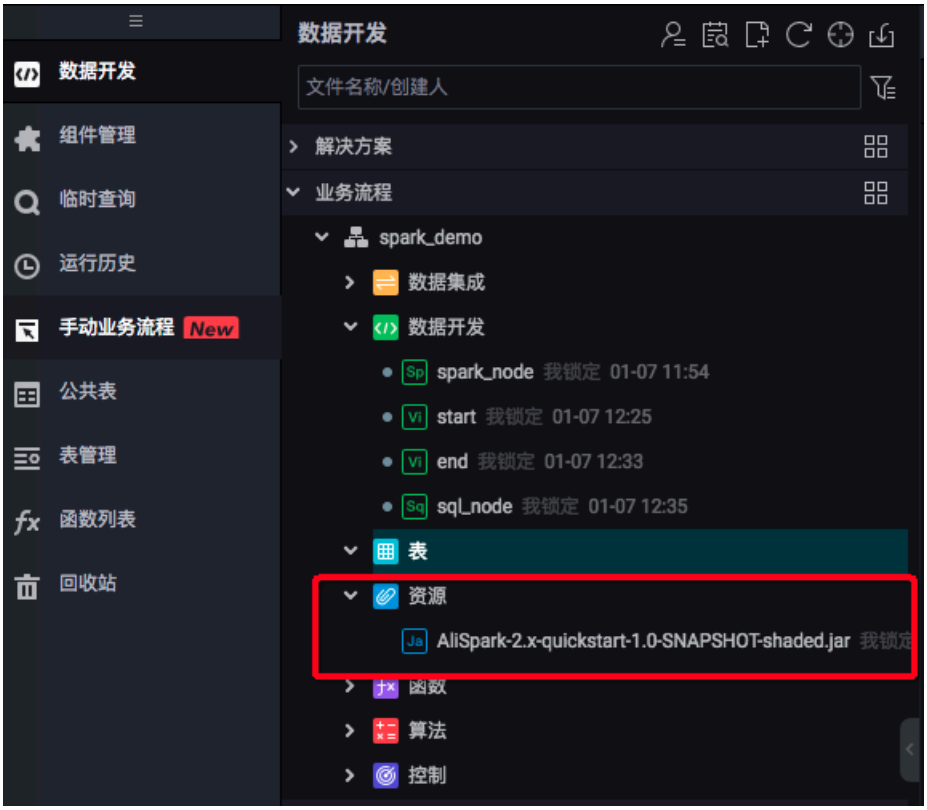
DataWorks执行模式

说明:

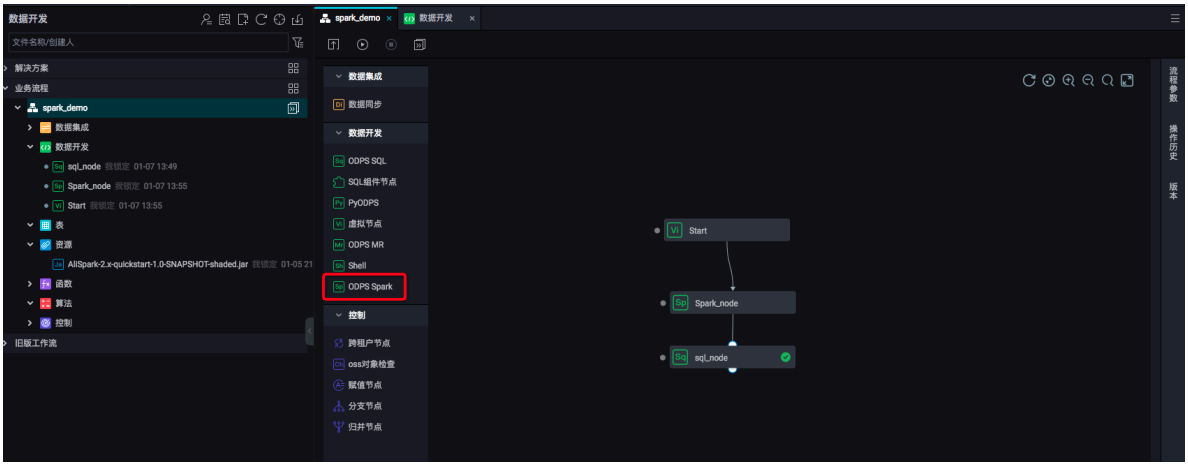
DataWorks的Spark节点目前在灰度发布中，如果您有在DataWorks中调用Spark节点的需求，请通过工单或加入钉钉群：21969532（MaxCompute Spark支持群）进行申请。

用户可以在DataWorks中运行MaxCompute Spark离线作业（Cluster模式），以方便与其它类型执行节点集成和调度。具体步骤如下：

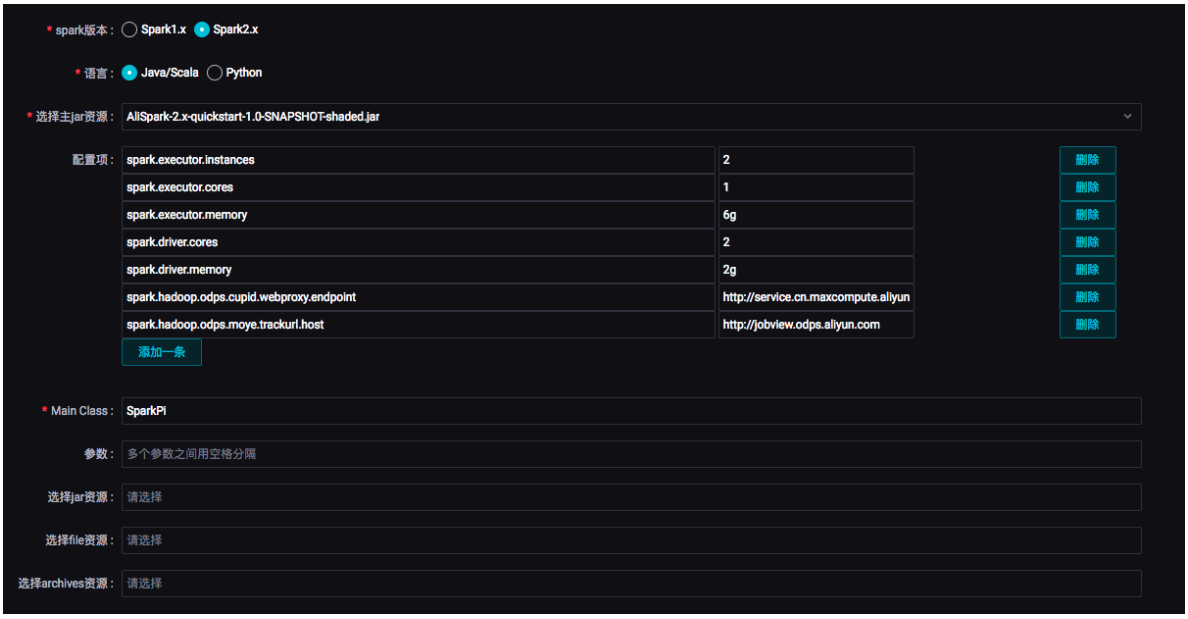
1. 用户需要在DataWorks的业务流程中上传并提交（单击提交按钮）资源。



2. 在创建的业务流程中，从数据开发组件中选择ODPS Spark节点。



3. 双击拖拽到工作流的Spark节点，对Spark作业进行任务定义。选择Spark的版本、任务使用的开发语言，并指定任务所使用的资源文件。这里的资源文件就是第一步在业务流程中预先上传并发布的资源文件。同时，您还可以指定提交作业时的配置项，例如Executor的数量、内存大小等配置项。同时还需设置MaxCompute Spark服务的Endpoint配置项：`spark.hadoop.odps.cupid.webproxy.endpoint`（取值填写项目所在Region的Endpoint，例如*http://service.cn.maxcompute.aliyun-inc.com/api*）。



4. 手动执行Spark节点，可以查看该任务的执行日志，从打印出来的日志中可以获取该任务的Logview和Jobview的URL，编译进一步查看与诊断。

Spark作业定义完成后，即可在业务流程中对不同类型服务进行编排、统一调度执行。

9.5 作业诊断

提交作业后，您需要根据作业日志来检查作业是否已正常提交并执行。MaxCompute为Spark作业提供了Logview工具以及Spark Web-UI，帮助开发者进行作业诊断。

通过Spark-submit方式（Dataworks执行Spark任务时也会产生相应日志）提交作业。

```
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class  SparkPi /tmp/spark-2.x
-demo/target/AliSpark-2.x-quickstart-1.0-SNAPSHOT-shaded.jar
```

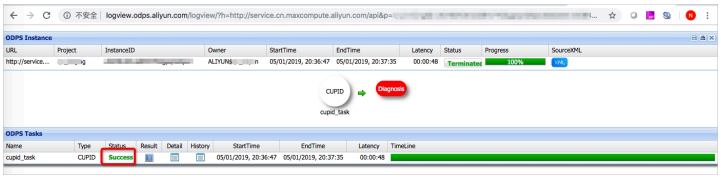
作业提交成功后，MaxCompute会创建一个Instance，在日志中会打印Instance的Logview。

```
19/01/05 20:36:47 INFO YarnClientImplUtil: logview url: http://logview
.odps.aliyun.com/logview/?h=http://service.cn.maxcompute.aliyun.com/
api&p=qn_beijing&i=xxx&token=xxx
成功标准：<看到以下输出，可能会有其他日志一并输出>
19/01/05 20:37:34 INFO Client:
   client token: N/A
   diagnostics: N/A
   ApplicationMaster host: 11.220.xxx.xxx
   ApplicationMaster RPC port: 30002
   queue: queue
   start time: 1546691807945
```

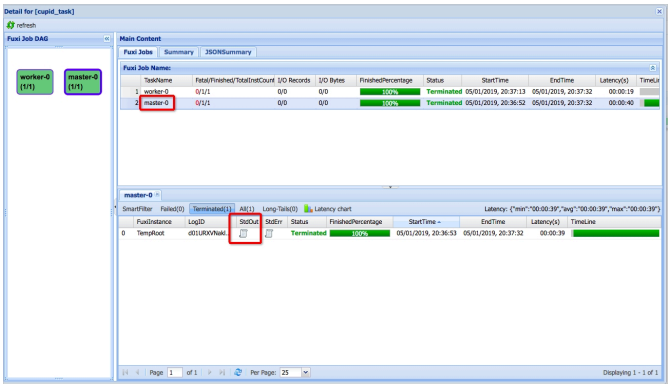
```
    final status: SUCCEEDED
    tracking URL: http://jobview.odps.aliyun.com/proxyview/jobview/?h=
http://service.cn.maxcompute.aliyun-inc.com/api&p=project_name&i=xxx&t
=spark&id=application_xxx&metaname=xxx&token=xxx
```

使用Logview工具诊断作业

1. 通过日志输出的Logview，在浏览器中可以查看CUPID类型的任务执行的基本信息。



2. 单击TaskName下名为master-0的任务条，在下方FuxiInstance栏中，通过All过滤。

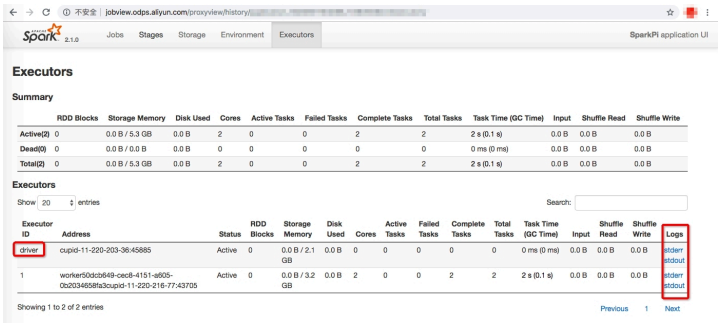


3. 单击TempRoot的StdOut，可以查看SparkPi的输出结果。

使用Web-UI诊断作业

日志中打印出上述的TrackingUrl，表示您的作业已经提交到MaxCompute集群。这个TrackingUrl非常关键，它既是SparkWebUI，也是HistoryServer的URL。

1. 在浏览器中打开这个URL，可以追踪Spark作业的运行情况。



2. 单击driver对应的stdout即可以查看Spark作业的输出内容。

9.6 Spark常见问题

本文为您介绍使用Spark过程中可能遇到的常见问题。

如何将开源Spark代码迁移到Spark on MaxCompute

分以下三种情形：

- 作业无需访问MaxCompute表和OSS。

  用户jar包可直接运行，具体步骤请参见搭建开发环境。注意，对于Spark或Hadoop的依赖必须设成`provided`。

- 作业需要访问MaxCompute表。

  配置相关依赖后重新打包即可。配置依赖的步骤请参见配置依赖。

- 作业需要访问OSS。

  配置相关依赖后重新打包即可。配置依赖的步骤请参见配置依赖。

如何通过Spark访问VPC环境内服务

Spark默认支持对MaxCompute表的访问，支持对OSS、OTS等外部数据源的访问处理。对VPC内的服务访问，目前默认暂不支持访问，如有需求可以通过工单和我们联系。

*spark-defaults.conf*提供的id、key错误

```
Stack:
com.aliyun.odps.OdpsException: ODPS-0410042:
Invalid signature value - User signature dose not match
```

请检查*spark-defaults.conf*提供的id、key和在阿里云官网管理控制台用户信息管理中的AccessKey ID、Access Key Secret是否一致。

提示权限不足

```
Stack:
com.aliyun.odps.OdpsException: ODPS-0420095:
Access Denied - Authorization Failed [4019], You have NO privilege '
odps:CreateResource' on {acs:odps:*:projects/*}
```

请project owner授权grant resource的read以及create权限。

项目未支持Spark任务运行

```
Exception in thread "main" org.apache.hadoop.yarn.exceptions.
YarnException: com.aliyun.odps.OdpsException: ODPS-0420095: Access
Denied - The task is not in release range: CUPID
```

首先需要确认项目所在的region中，是否已经提供了MaxCompute Spark服务。同时，检查*Spark-defaults.conf*配置信息是否与产品文档中的要求一致。如果region已经支持，请通过工单或加入钉钉群：21969532（MaxCompute Spark支持群）进行咨询。

运行时报错`No space left on device`

Spark使用网盘进行本地存储。Shuffle数据和BlockManager溢出的数据均存储在网盘上。网盘的大小通过参数`spark.hadoop.odps.cupid.disk.driver.device_size`控制，默认20g，最大100g。如果调整到100g仍然报出此错误，则需要分析具体原因。常见的原因为数据倾斜，在Shuffle或者Cache过程中数据集中分布在某些Block，此时可以缩小单个Executor的并发（`spark.executor.cores`），增加Executor的数量（`spark.executor.instances`）。

10 交互式分析 (Lightning)

10.1 概述

MaxCompute Lightning是MaxCompute产品的交互式查询服务，支持以PostgreSQL协议及语法连接访问Maxcompute项目，让您使用熟悉的工具以标准SQL查询分析MaxCompute项目中的数据，快速获取查询结果。

您可使用主流BI工具（如Tableau、帆软等）或SQL客户端轻松连接到MaxCompute项目，开展BI分析或即席查询。或者利用MaxCompute Lightning的快速查询特性，将项目表数据封装成API对外服务，无需数据迁移就能够支持更丰富的应用场景。

MaxCompute Lightning提供无服务器计算（Serverless）的服务方式，您无需管理任何基础设施，只需为运行的查询付费。

关键特性

- 兼容PostgreSQL

  MaxCompute Lightning提供兼容PostgreSQL协议的JDBC/ODBC接口，所有支持PostgreSQL数据库的工具或应用使用默认驱动都可以轻松地连接到MaxCompute项目。您也可以使用更广泛的PostgreSQL生态工具来分析MaxCompute的数据。
- 显著提升性能

  针对MaxCompute表的快速查询进行了优化，特别是在小数据集、并发场景下有更好的性能表现。从而能够支撑更丰富的应用场景，如固定报表、API开放等。
- 统一的权限管理

  作为MaxCompute产品内的服务，通过MaxCompute Lightning连接到MaxCompute项目的访问完全遵循Maxcompute项目的权限体系，在访问用户权限范围内安全地查询数据。
- 开箱即用，按查询付费

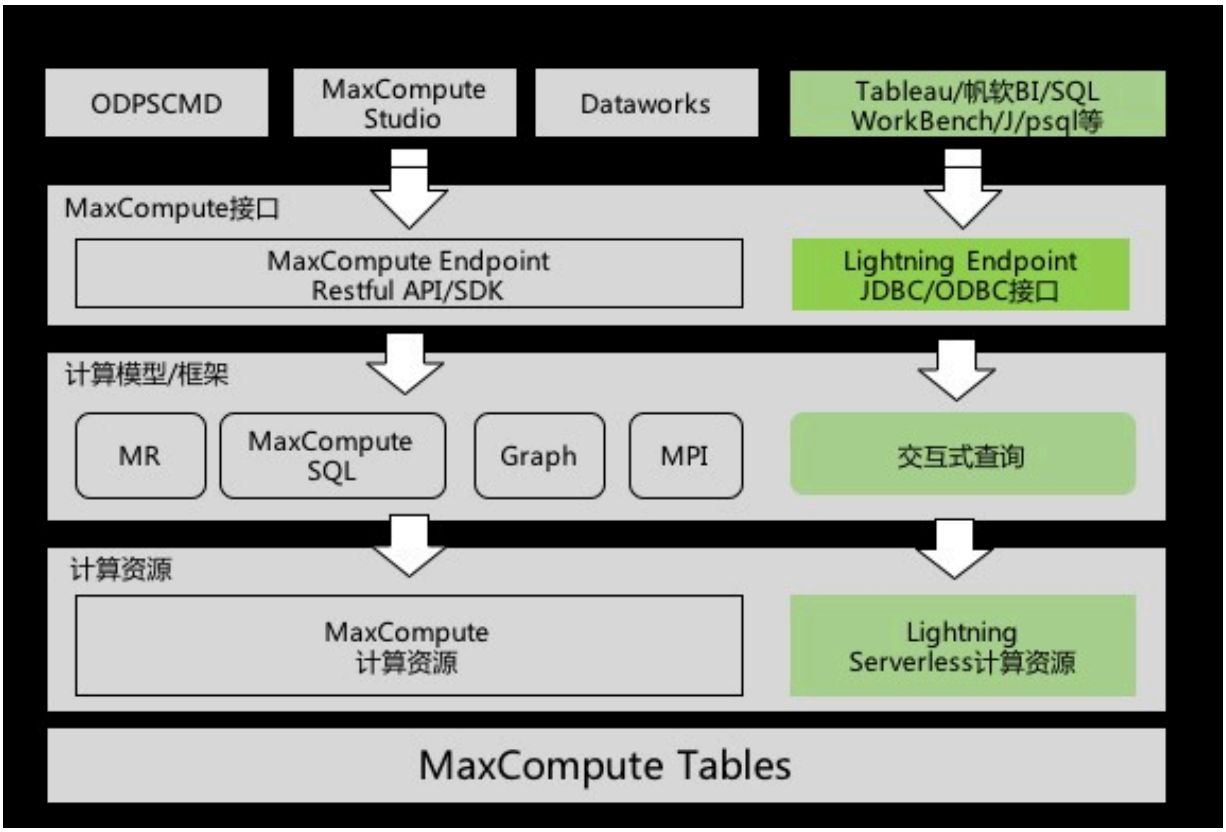
  MaxCompute Lightning是在MaxCompute已有的计算资源之外提供的无服务器的计算服务，您不需要设置、管理或运维MaxCompute Lightning资源，通过MaxCompute Lightning连接后即可开展查询。

  使用MaxCompute Lightning时，只需为每次查询所实际处理的数据量付费，不查询时不产生费用。

系统结构

作为Maxcompute的交互式查询服务，MaxCompute Lightning提供了配套的接入访问域名地址(Endpoint)，客户端工具及应用通过PostgreSQL驱动连接访问Lightning JDBC/ODBC接口服务，在MaxCompute项目统一的权限体系下安全地访问项目数据。

通过该服务接口连接并提交的查询任务，都将使用MaxCompute Lightning的Serverless计算资源以保障交互式查询的服务质量。



应用场景

- 即席查询（Ad Hoc）

  利用MaxCompute Lightning面向小规模数据集（百GB规模内）查询性能优化的特性，使用者可以直接对MaxCompute表开展低时延的查询操作，而不需要再把数据再导入到其它各种系统进行加速（比如ADS、RDS），节约资源和管理成本。

  场景特点：查询的数据对象自由不固定，逻辑相对复杂，期望快速获取查询结果并调整查询逻辑，对查询时延的要求在几十秒内。使用者往往是掌握SQL技能的数据分析师，希望使用熟悉的客户端工具来开展查询分析。

- Reporting报表分析

  对MaxCompute项目中通过ETL加工汇总后的结果数据制作分析报表，提供给管理者和业务人员定期查看。

  场景特点：查询的数据对象通常为聚合后的结果数据， 数据量较小、查询逻辑固定且较简单。时延要求低：秒级返回（例如大部分查询不超过5秒，不同查询根据其数据规模和查询复杂度有较大差异）。
- 面向在线应用的消费场景

  直接将MaxCompute项目中的数据封装成为restful api，支撑在线应用。

  场景特点：利用MaxCompute Lightning作为加速查询引擎，结合阿里云Dataworks的数据服务组件，零开发、无运维地将MaxCompute的表数据开放为API服务。

使用限制

- Lightning常见限制
- DDL、DML、查询及UDF限制

10.2 通过第三方工具连接服务

本教程将引导您使用主流的第三方工具连接MaxCompute Lightning服务，查看指定MaxCompute项目下的数据表，进行BI分析。

前提条件

- 开通MaxCompute并创建项目。

  使用MaxCompute Lightning需要您已经有创建好的MaxCompute项目。

  如果您还没有开通阿里云MaxCompute项目，请参见开通MaxCompute。
- 创建表并导入数据。

  使用MaxCompute Lightning前需要您创建表并导入数据，详情请参见创建和查看表。

- 获取云账号信息。

  使用MaxCompute Lightning前需要MaxCompute项目用户的Accesss ID及Access Key云账号信息。

  您可登录阿里云官网，进入管理控制台用户信息管理页面进行查看。如果您的子账号没有查看权限，请联系主账号管理员索取，同时确定该子账号有权限查看指定的数据表。

准备连接的客户端工具

MaxCompute Lightning兼容PostgreSQL接口，所有支持PostgreSQL数据库连接的客户端工具都可以用于连接MaxCompute Lightning。

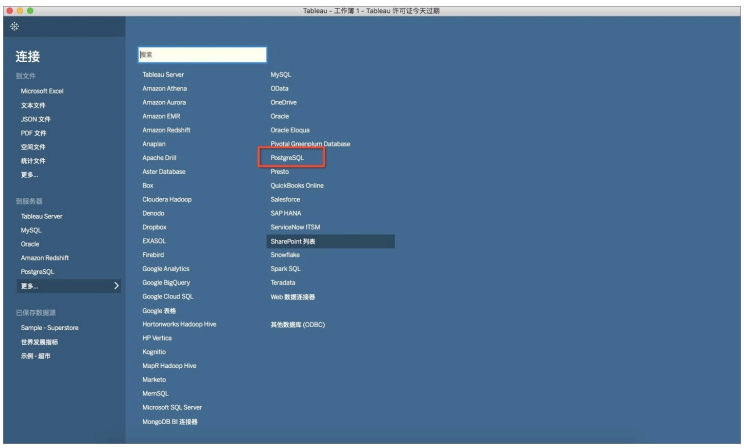
本文选择大家熟悉的BI工具Tableau Desktop进行示例，相关工具请到Tableau官网进行下载。其他常见的客户端工具，例如SQL Workbench/J、PSQL、帆软BI、MicroStrategy BI等，都可以像连接PostgreSQL数据库一样配置服务连接。

连接服务并开展分析

连接服务成功后，您可以查看指定MaxCompute项目下的数据表，进行BI分析。

1. 连接服务器时选择PostgreSQL。

   打开Tableau Desktop，选择连接 > 到服务器 > 更多 > PostgreSQL。

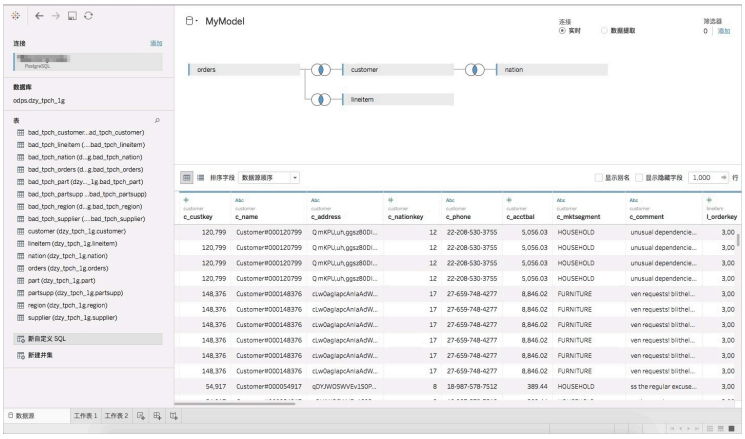


2. 填写服务连接及用户认证信息。

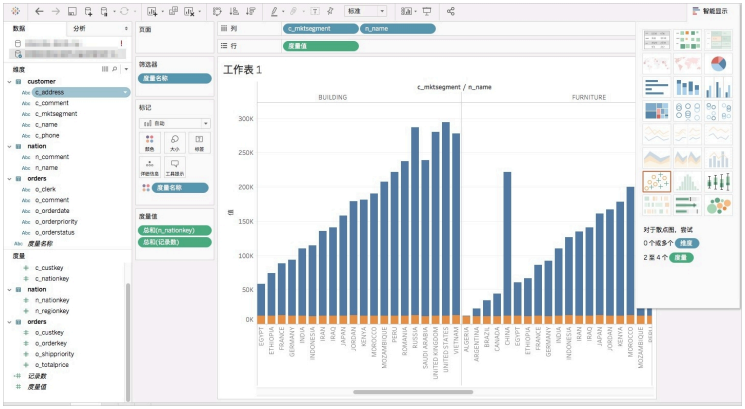
参数	说明
服务器	填写所在区域对应的MaxCompute Lightning EndPoint，如华东2区域的Endpoint为：`lightning.cn-shanghai.maxcompute.aliyun.com`，详情请参见访问域名。
端口	443
数据库	MaxCompute项目名
身份验证	用户名和密码
用户名/密码	您的Access Key ID/Access Key Secret
SSL连接	勾选需要SSL

3. 获取项目表信息，创建数据源/模型。

配置好联系信息并登录后，Tableau会加载所连接的MaxCompute项目下的表，您可以根据需要选择对应的表创建模型和工作表。



选择特定数据的维度和指标，创建工作表。



至此，您已经使用Tableau工具成功连接MaxCompute Lightning服务，可以对连接到MaxCompute项目内的数据进行BI分析。

> **说明:**
> 为了获得更好的性能和体验，建议您使用Tableau支持的TDC文件方式对Lightning数据源进行连接定制优化，详情请参见Tableau Desktop。

10.3 访问域名Endpoint

MaxCompute Lightning提供了单独的域名访问地址（Endpoint），通过该地址您可以访问到阿里云不同区域的MaxCompute Lightning服务。

MaxCompute Lightning在公共云的不同Region及网络环境下的服务连接对照表如下。

表 10-1: 外网网络下Region和服务连接对照表

Region	开服状态	外网Endpoint
华东1	已开服	lightning.cn-hangzhou.maxcompute.aliyun.com
华东2	已开服	lightning.cn-shanghai.maxcompute.aliyun.com
华北2	已开服	lightning.cn-beijing.maxcompute.aliyun.com
华南1	已开服	lightning.cn-shenzhen.maxcompute.aliyun.com
香港	公测中	lightning.cn-hongkong.maxcompute.aliyun.com
亚太东南1	公测中	lightning.ap-southeast-1.maxcompute.aliyun.com
其他区域	暂未开服	-

表 10-2: 经典网络下Region和服务连接对照表

Region	开服状态	经典网络Endpoint
华东1	已开服	lightning.cn-hangzhou.maxcompute.aliyun-inc.com
华东2	已开服	lightning.cn-shanghai.maxcompute.aliyun-inc.com
华北2	已开服	lightning.cn-beijing.maxcompute.aliyun-inc.com
华南1	已开服	lightning.cn-shenzhen.maxcompute.aliyun-inc.com
香港	公测中	lightning.cn-hongkong.maxcompute.aliyun-inc.com
亚太东南1	公测中	lightning.ap-southeast-1.maxcompute.aliyun-inc.com
其他区域	暂未开服	-

表 10-3: VPC网络下Region和服务连接对照表

Region	开服状态	VPC网络Endpoint
华东1	已开服	lightning.cn-hangzhou.maxcompute.aliyun-inc.com
华东2	已开服	lightning.cn-shanghai.maxcompute.aliyun-inc.com
华东2金融云	已开服	lightning.cn-shanghai-finance.maxcompute.aliyun-inc.com
华北2	已开服	lightning.cn-beijing.maxcompute.aliyun-inc.com
华南1	已开服	lightning.cn-shenzhen.maxcompute.aliyun-inc.com
香港	公测中	lightning.cn-hongkong.maxcompute.aliyun-inc.com
亚太东南1	公测中	lightning.ap-southeast-1.maxcompute.aliyun-inc.com
其他区域	暂未开服	-

10.4 通过JDBC连接MaxCompute服务

10.4.1 JDBC驱动程序

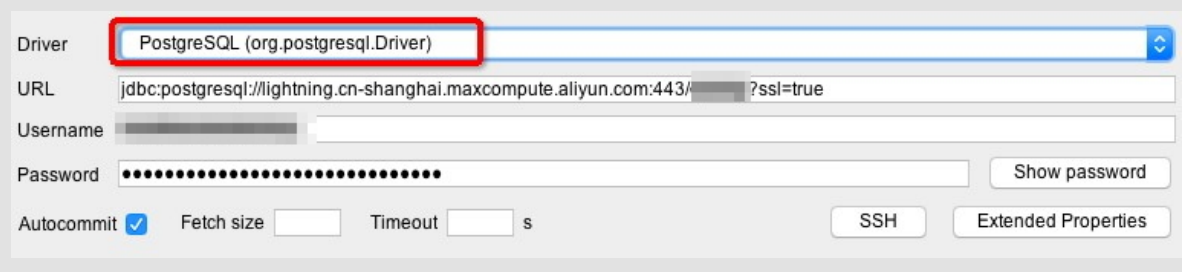
MaxCompute提供完全兼容PostgreSQL消息协议的Java数据库连接（JDBC）接口，使用者可以通过JDBC将SQL客户端工具连接到MaxCompute Lightning服务。

MaxCompute Lightning支持PostgeSQL官方驱动连接，同时也提供了为Lightning服务而优化的驱动程序供选择。

Maxcompute Lightning查询引擎基于PostgreSQL 8.2，当前仅支持对已有MaxCompute表进行SELECT查询，更多详情请参见查询语法及函数。

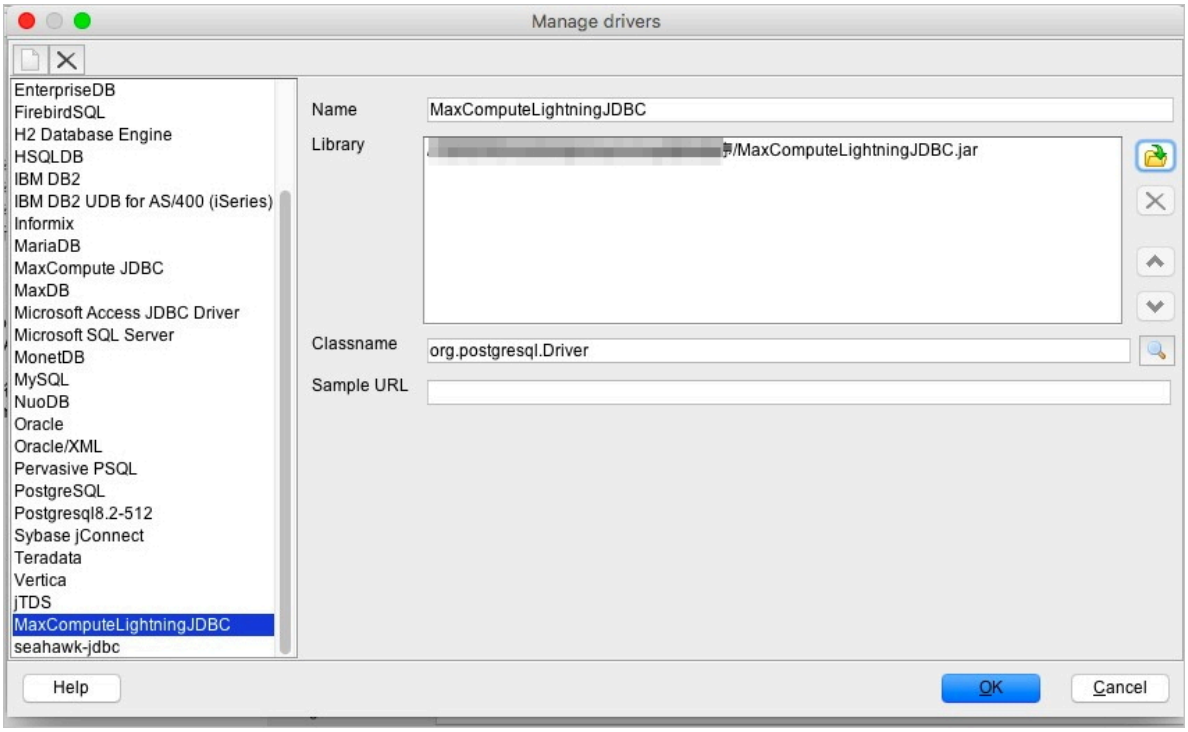
1. 使用PostgeSQL官方提供的JDBC驱动程序。

> 说明:
> 通常，客户端工具默认集成了PostgreSQL数据库的驱动，直接使用工具自带的驱动即可。如果未集成，可从客户端官网下载。以SQL Workbench/J客户端为例，创建连接时可选择PostgreSQL官方驱动。

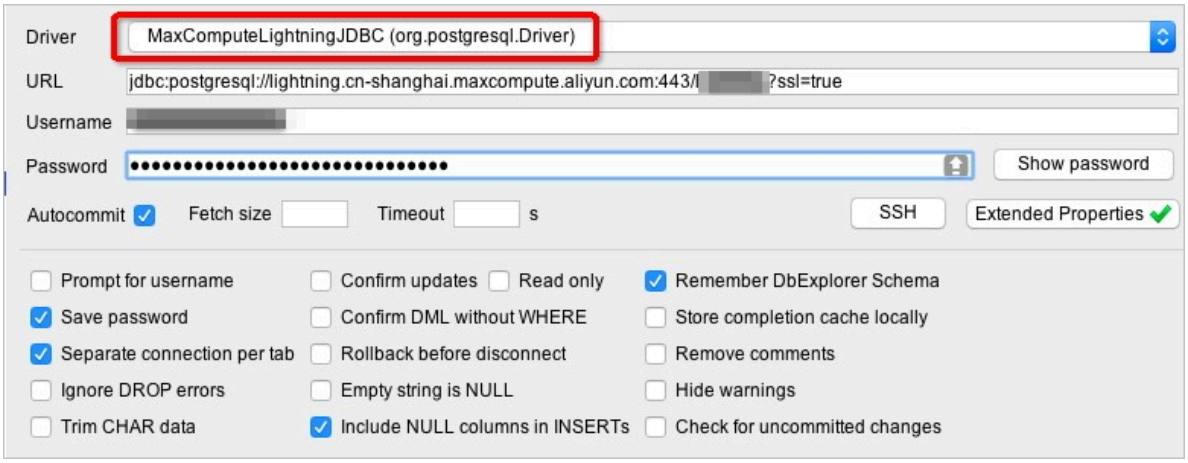


2. 使用阿里云MaxCompute Lightning优化过的JDBC驱动，以获取更好的性能。

   将下载后的MaxCompute Lightning的JDBC驱动程序保存为MaxComputeLightningJDBC.jar文件。以SQL Workbench/J客户端为例，在驱动管理菜单中，添加MaxCompute Lightning JDBC驱动程序项界面如下。



   在创建连接时，从Driver列表中选择刚才添加的MaxCompute Lightning JDBC驱动。



10.4.2 配置JDBC连接

您需要获得您的MaxCompute项目JDBC URL，才能将SQL客户端工具连接到该项目。

Maxcompute Lightning查询引擎基于PostgreSQL 8.2，当前仅支持对已有MaxCompute表进行SELECT查询，更多详情请参见查询语法及函数。

JDBC URL命名方式如下。

```
jdbc:postgresql://endpoint:port/database
```

连接参数说明如下。

参数	取值	说明
endpoint	所在区域不同网络环境下的Lightning访问域名	详情请参见访问域名，例如通过外网访问上海Region的服务使用`lightning.cn-shanghai.maxcompute.aliyun.com`
port	443	无
database	填写MaxCompute的项目名称	无
user	访问用户的Access Key ID	无
password	访问用户的Access Key Secret	无
ssl	true	MaxCompute Lightning服务端默认开启SSL服务，客户端需要使用SSL进行连接。
prepareThreshold	0	可选。需要使用JDBC PrepareStatement功能时， 建议设置`prepareThreshold=0`。

连接URL举例：`jdbc:postgresql://lightning.cn-shanghai.maxcompute.aliyun.com:443/myproject`

之后，您还需要配置`user`、`password`、SSL连接参数后才能连接成功。

您也可以将参数添加到JDBC URL中，直接连接到MaxCompute项目，举例如下。

```
jdbc:postgresql://lightning.cn-shanghai.maxcompute.aliyun.com:443/
myproject?ssl=true& prepareThreshold=0&user=xxx&password=yyy
```

说明：

- `lightning.cn-shanghai.maxcompute.aliyun.com`是华东2区域的Endpoint。
- `myproject`是需要访问的MaxCompute项目名称。
- `ssl=true`是指定通过SSL方式连接。

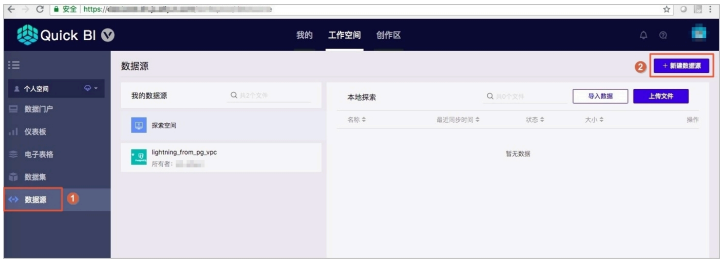
- xxx是访问用户的Access Key ID。
- yyy是访问用户的Access Key Secret。

10.4.3 常见工具的连接

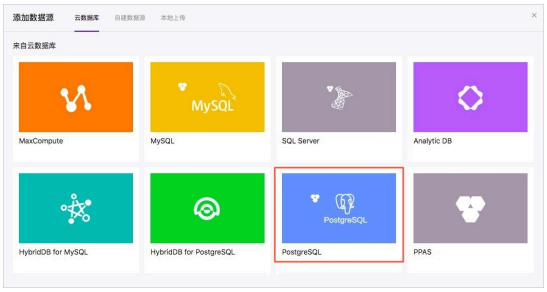
本文将为您介绍几款常见客户端工具的接入说明，除此之外，原则上支持PostgresSQL的工具都可以通过Lightning来对接访问MaxCompute。

阿里云Quick BI

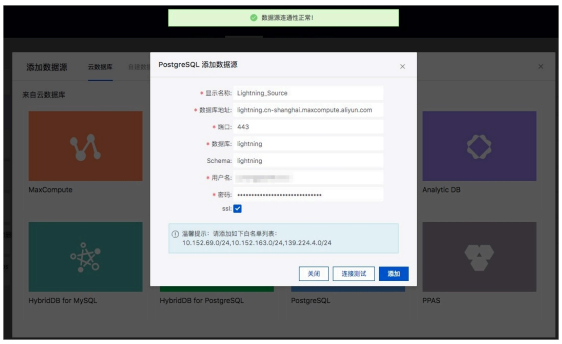
1. 登录Quick BI控制台，单击左侧导航栏中的数据源。
2. 单击数据源管理页面右上角的新建数据源。



3. 选择云数据库或自建数据源中的PostgeSQL数据库类型添加数据源。



4. 填写对话框中的MaxCompute Lightning的连接信息并测试连接连通状态。



参数	说明
数据库地址	MaxCompute Lightning对应区域的Endpoint，可使用公网访问的Endpoint，也可以使用经典网络及VPC网络访问的Endpoint。
数据库	需要访问的MaxCompute项目的名称。
Schema	MaxCompute项目名称。

参数	说明
用户名/密码	在MaxCompute项目中用户的Access Key ID/Access Key Secret。
ssl	请勾选
连接测试	连通正常时添加数据源以便在QuickBI中进行分析。

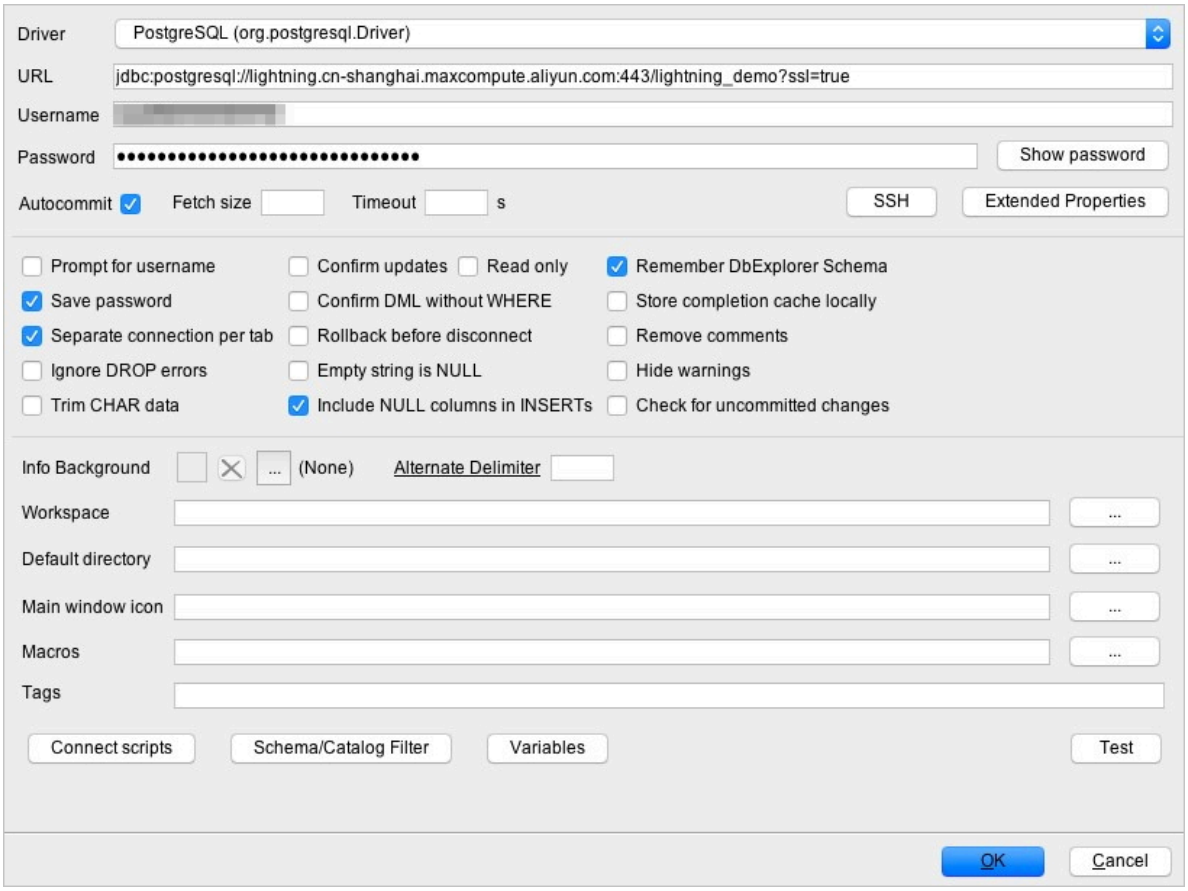
SQL Workbench/J

SQL Workbench/J是一款流行的免费、跨平台SQL查询分析工具，使用SQL Workbench/J可以通过PostgreSQL驱动连接MaxCompute Lightning服务。

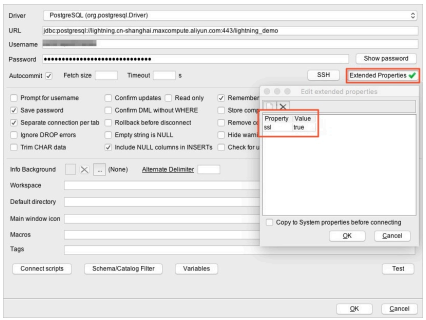
1. 下载并安装SQL Workbench/J 。

2. 启动SQL Workbench/J，创建数据库连接。

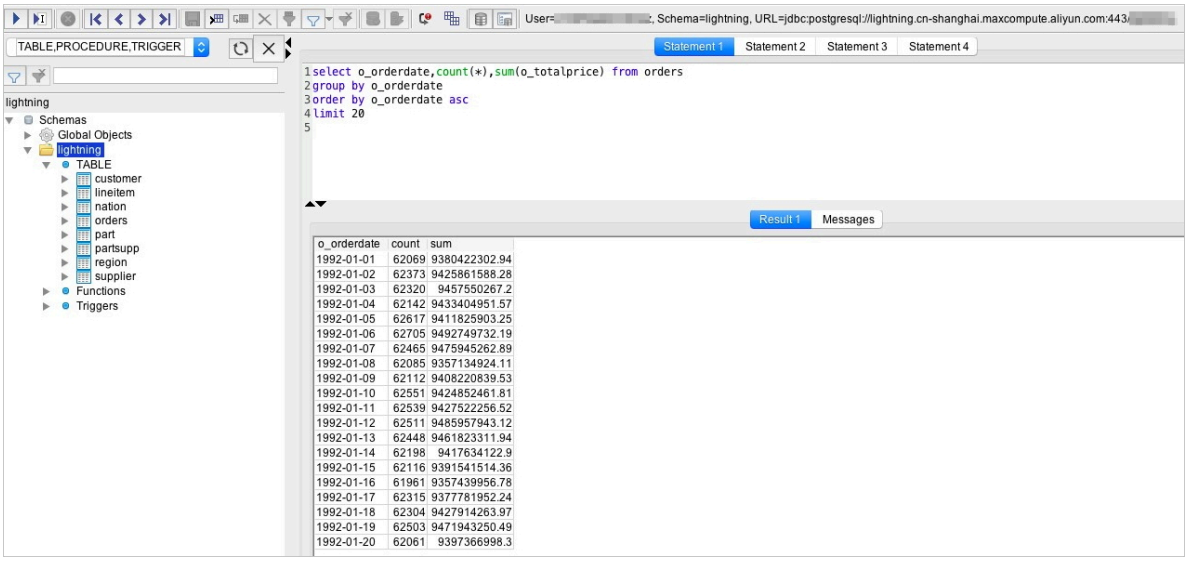
选择PostgreSQL驱动，连接MaxCompute项目所对应的Lightning URL地址，同时输入访问用户的用户名和密码，即Access Key ID和Access Key Secret。



您也可通过扩展属性（Extended Properities）设置ssl取值为true。



3. 连接后，在Workbench工作区查看MaxCompute项目的表数据、查询分析。



psql工具连接

psql是PostgreSQL的一个命令行交互式客户端工具，在本机安装PostgreSQL数据库将默认安装psql客户端。

通过psql在命令行下可以连接MaxCompute Lightning，语法与连接PostgreSQL数据库一致。

```
psql -h <endpoint> -U <userid> -d <databasename> -p <port>
```

参数说明：

- <endpoint>：MaxCompute Lightning的Endpoint，详情请参见访问域名Endpoint。
- <userid>：访问用户Access Key ID。
- <databasename>：Maxcompute项目名。
- <port>：443

执行后，在psql密码提示符处，输入<userid>用户的密码，即访问用户的Access Key Secret。

示例如下：

```
[~]$psql -h lightning.cn-shanghai.maxcompute.aliyun.com -U                z -d          -p 443
Password for user
psql (10.3, server 8.2.15)
Type "help" for help.

lightning=> \d
              List of relations
 Schema    |       Name        | Type  |   Owner
-----------+-------------------+-------+------------
 lightning | customer          | table | proxy_role
 lightning | lineitem          | table | proxy_role
 lightning | nation            | table | proxy_role
 lightning | orders            | table | proxy_role
 lightning | orders_partition  | table | proxy_role
 lightning | orders_partition2 | table | proxy_role
 lightning | orders_partition3 | table | proxy_role
 lightning | part              | table | proxy_role
 lightning | partsupp          | table | proxy_role
 lightning | region            | table | proxy_role
 lightning | supplier          | table | proxy_role
 lightning | test              | table | proxy_role
(12 rows)

lightning=> select * from orders limit 10;
 o_orderkey | o_custkey | o_orderstatus | o_totalprice | o_orderdate | o_orderpriority |     o_clerk     | o_shippriority |                    o_comment
------------+-----------+---------------+--------------+-------------+-----------------+-----------------+----------------+-----------------------------------------------
   21638945 |   9836128 | O             |    149664.71 | 1997-08-01  | 5-LOW           | Clerk#000000016 |              0 | ular requests are quickly ironic reque
   21638946 |    781658 | O             |     70792.99 | 1997-03-08  | 1-URGENT        | Clerk#000066763 |              0 | ons wake even, bold requests. slyly bold requests snooze slyly fin
   21638947 |  12320353 | O             |    231895.68 | 1996-01-13  | 3-MEDIUM        | Clerk#000078663 |              0 | lithely regular deposits affix q
   21638948 |   8140645 | F             |    128980.5  | 1995-02-18  | 4-NOT SPECIFIED | Clerk#000082968 |              0 |  regular, even requests? furiously enticing ins
   21638949 |   4800883 | P             |    289947.16 | 1995-05-08  | 4-NOT SPECIFIED | Clerk#000035612 |              0 | d theodolites among the slow dolphins ca
   21638950 |   3428813 | F             |    243629.29 | 1993-09-08  | 1-URGENT        | Clerk#000024564 |              0 | ic, ironic excuses haggle silent instructions.
   21638951 |  13056718 | F             |    187252.37 | 1994-09-05  | 1-URGENT        | Clerk#000045762 |              0 | rouches engage blithely among the blithely regu
   21638976 |   9240733 | F             |     130092.7 | 1993-04-09  | 2-HIGH          | Clerk#000020449 |              0 | nto beans. furiously express Tiresias above the regular a
   21638977 |  13996019 | F             |    247587.28 | 1994-02-16  | 3-MEDIUM        | Clerk#000071761 |              0 | posits haggle. deposits
   21638978 |   1084246 | F             |    275689.34 | 1993-09-01  | 4-NOT SPECIFIED | Clerk#000085923 |              0 |  the blithely regular deposits. requests kindle fluffily. ideas nag s
(10 rows)
```

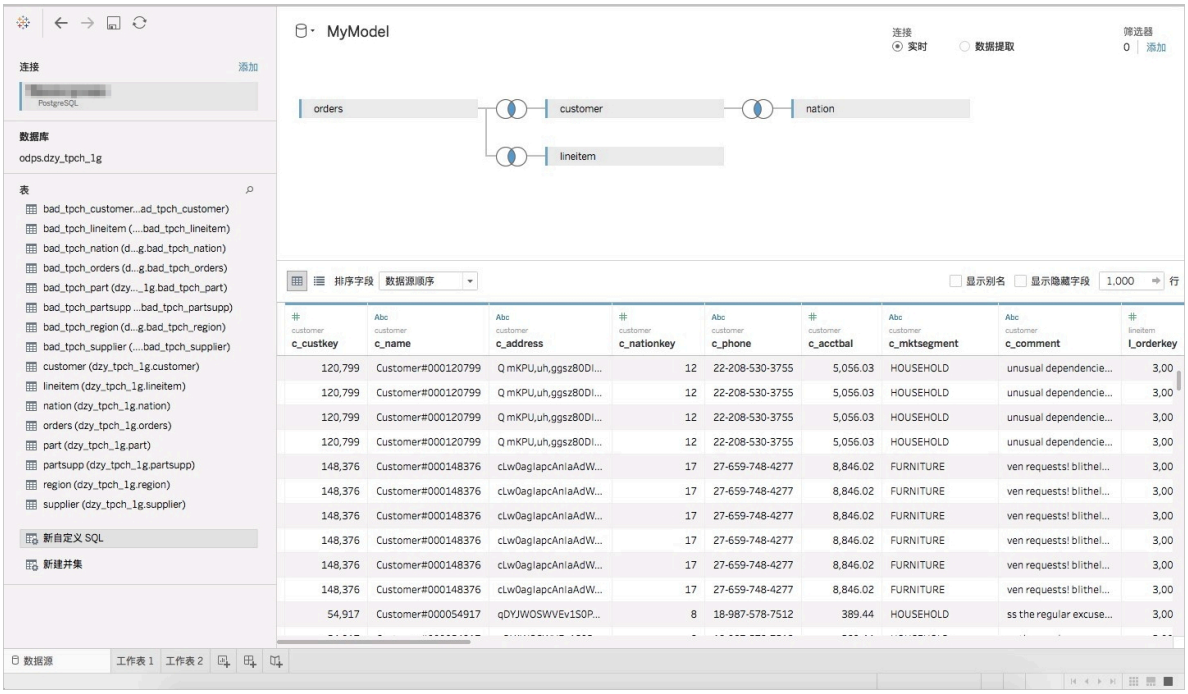
说明:

psql默认优先通过ssl方式连接。

Tableau Desktop

1. **使用BI工具，选择PostgreSQL数据源，配置连接。配置连接时，需勾选需要SSL。**

2. 登录后，通过Tableau创建工作表进行可视化分析。



说明:

时间函数会默认返回UTC时区的时间。如果需要返回北京时间，则需要进行如下配置。

```
SET timezone = 'Asia/Shanghai';
```

为了获得更好的性能和体验，建议您使用Tableau支持的tdc文件方式，对Lightning数据源进行连接定制优化。具体操作如下：

1. 将如下xml内容保存为*postgresql.tdc*文件。

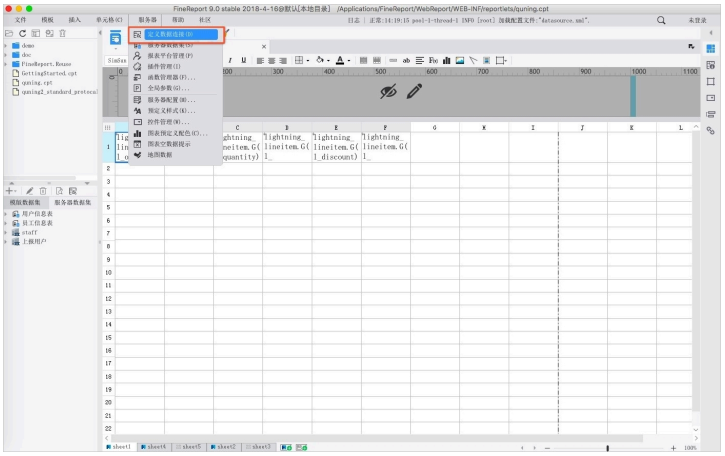
```
<?xml version='1.0' encoding='utf-8' ?>
<connection-customization class='postgres' enabled='true' version='8
.10'>
<vendor name='postgres'/>
<driver name='postgres'/>
<customizations>
<customization name='CAP_CREATE_TEMP_TABLES' value='no' />
<customization name='CAP_STORED_PROCEDURE_TEMP_TABLE_FROM_BUFFER'
value='no' />
<customization name='CAP_CONNECT_STORED_PROCEDURE' value='no' />
<customization name='CAP_SELECT_INTO' value='no' />
<customization name='CAP_SELECT_TOP_INTO' value='no' />
<customization name='CAP_ISOLATION_LEVEL_SERIALIZABLE' value='yes' /
>
<customization name='CAP_SUPPRESS_DISCOVERY_QUERIES' value='yes' />
<customization name='CAP_SKIP_CONNECT_VALIDATION' value='yes' />
<customization name='CAP_ODBC_TRANSACTIONS_SUPPRESS_EXPLICIT_COMMIT
' value='yes' />
<customization name='CAP_ODBC_TRANSACTIONS_SUPPRESS_AUTO_COMMIT'
value='yes' />
<customization name='CAP_ODBC_REBIND_SKIP_UNBIND' value='yes' />
<customization name='CAP_FAST_METADATA' value='no' />
```

```
<customization name='CAP_ODBC_METADATA_SUPPRESS_SELECT_STAR' value='
yes' />
<customization name='CAP_ODBC_METADATA_SUPPRESS_EXECUTED_QUERY'
value='yes' />
<customization name='CAP_ODBC_UNBIND_AUTO' value='yes' />
<customization name='SQL_TXN_CAPABLE' value='0' />
<customization name='CAP_ODBC_CURSOR_FORWARD_ONLY' value='yes' />
<customization name='CAP_ODBC_TRANSACTIONS_COMMIT_INVALIDATES
_PREPARED_QUERY' value='yes' />
</customizations>
</connection-customization>
```

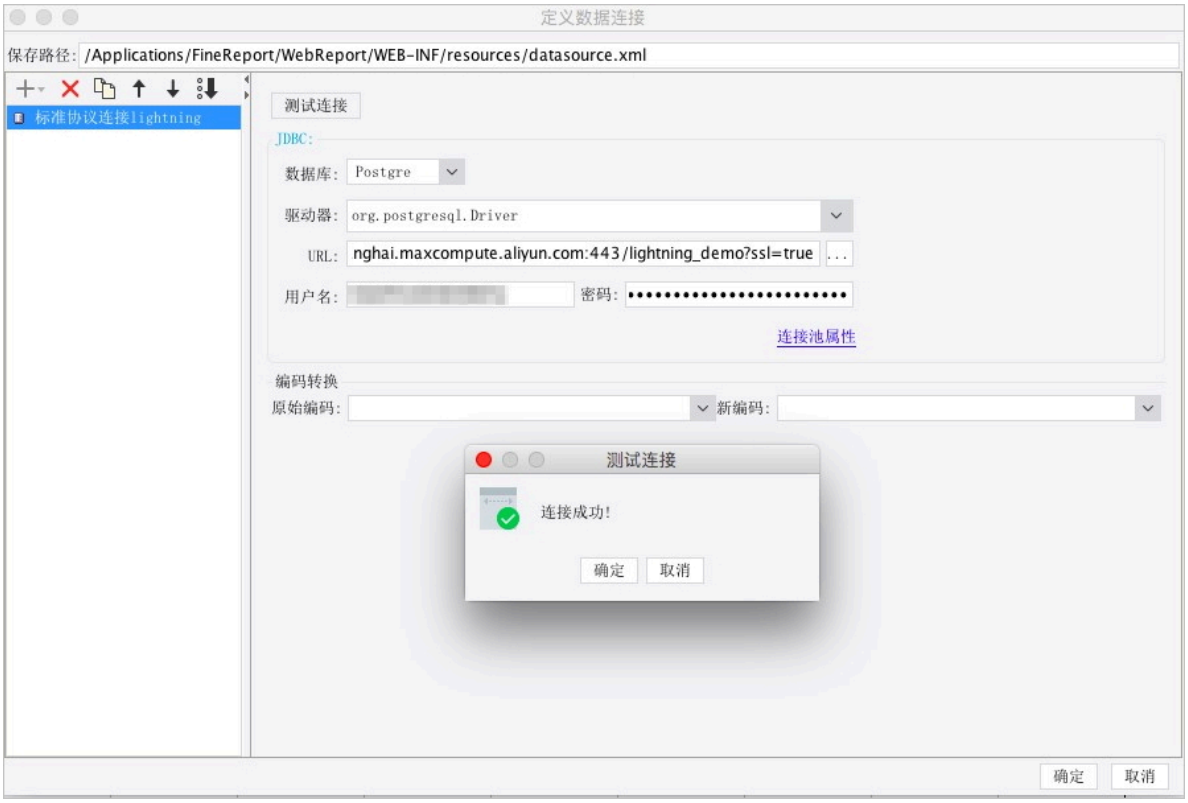
2. 将文件保存到*\My Documents\My Tableau Repository\Datasources*目录下。如果是Tableau Server，Windows下请保存在*C:\ProgramData\Tableau\Tableau Server\data\tabsvc\vizqlserver\Datasources*；Linux下请保存在*/var/opt/tableau/tableau_server/data/tabsvc/vizqlserver/Datasources/*。
3. 重新打开Tableau，使用PostgreSQL数据源连接MaxCompute Lightning服务。关于tdc文件定制数据源的更多内容，请参见Tableau官方帮助文档。

帆软Report

1. 打开帆软Report，选择服务器 > 定义数据库连接。



2. 添加JDBC连接。



参数说明如下：

参数	说明
数据库	Postgre
驱动器	帆软Report自带的org.postgresql.Driver
URL	`jdbc:postgresql://<MaxCompute Lightning Endpoint>:443/<Project_Name>?ssl=true&prepareThreshold=0` 例如：`jdbc:postgresql://lightning.cn-shanghai.maxcompute.aliyun.com:443/lightning_demo?ssl=true&prepareThreshold=0`
用户名/密码	访问用户的Access Key ID和Access Key Secret

10.5 查询语法与内建函数

在PostgreSQL官方函数基础上，MaxCompute Lightning还支持查询语法、MAX_PT等内建函数。

查询语法

MaxCompute Lightning查询引擎基于PostgreSQL 8.2。当前仅支持对已有MaxCompute表进行SELECT查询，相关语法参见PostgreSQL官方文档。

函数

MAX_PT

命令格式：

```
max_pt(table_full_name)
```

命令说明：

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，用于指定表名（必须携带project名称，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：

假设tbl为分区表，对应分区如下，且都包含数据文件：

```
pt ='20120901'
pt ='20120902'
```

则以下语句中分区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');
```

10.6 查看或取消查询作业

Lightning为用户提供了一个系统视图表stv_recents，通过查询该表能够获取当前用户正在执行中的所有查询作业。

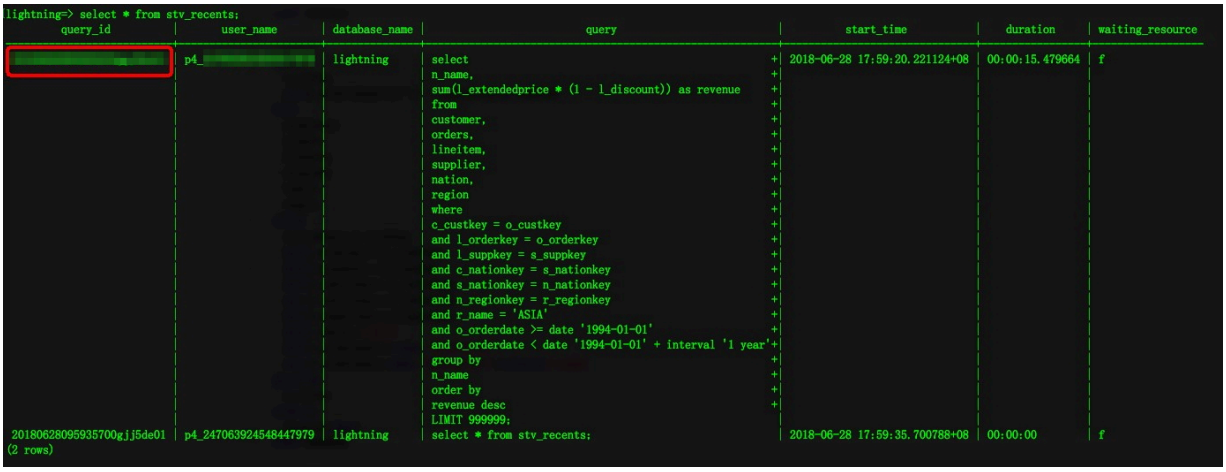
查看运行中的查询

您可以执行查询命令查看这些作业的ID、用户、SQL语句、发起开始时间、运行时间、是否等待资源（t为等待资源未执行，f为获取资源并执行中）等信息。

执行查询命令：

```
select * from stv_recents;
```

查询结果如下所示：



取消正在运行的查询

通过查询stv_recents获得运行中查询的信息，如果想要取消某个查询，可以执行下述语句：

```
select cancel('query_id');
```

其中的query_id，是运行中查询的query_id信息，示例如下：

10.7 约束与限制

本文为您介绍使用MaxCompute Lightning服务的约束与限制。

DDL/DML的约束限制

MaxCompute Lightning目前不支持update、create、delete、insert操作，仅支持对MaxCompute表进行select，后续版本将陆续开放相关能力。

查询约束限制

- 查询分区表时，扫描分区数的最大值为1024。
- 目前不支持创建和使用View。
- 目前不支持的数据类型：MAP、ARRAY、TINYINT、BINARY、TIMESTAMP以及带精度的DECIMAL（陆续支持中）。
- 每个查询中对单张表最大的数据扫描量为1TB。
- 提交的查询语句的长度不超过100KB。
- 查询超时时间为1小时。
- 由于数据服务通过Lightning查询MaxCompute，Lightning兼容PostgreSQL，而PostgreSQL中没有Datetime数据类型，所以如果您的数据有Datetime类型，数据服务中会映射为PostgreSQL的Timestamp类型进行查询。

UDF约束限制

- 当前不支持在MaxCompute Lightning中使用MaxCompute创建的UDF。
- 当前不支持在MaxCompute Lightning中创建和使用PostgreSQL UDF（支持使用PostgreSQL内建函数）。
- 当前不支持MaxCompute内建函数。

Query并发约束

单个MaxCompute项目的MaxCompute Lightning查询并发数限制为20。

10.8 常见问题

本文为您整理应用MaxCompute Lightning过程中的常见问题。

- Q：如何使用Lightning，有WEB控制台可以使用吗?

  A：Lightning没有提供WEB控制台接口，可以通过SQL客户端/BI工具等连接Lightning的JDBC/ODBC接口，即可访问对应的MaxCompute项目。

- Q：还没有建表的情况下，能够使用MaxCompute Lightning查询什么数据?

  A：您使用阿里云账号，通过MaxCompute Lightning能够访问到指定MaxCompute项目中的数据表，如果项目中还没有数据表，您需要首先利用Dataworks或者odpscmd客户端等工具，在MaxCompute项目中进行创建并加载、加工数据。这时再通过MaxCompute Lightning连接到该项目，这时就能查看到项目内有权限的数据表，对这些表进行查询。
- Q：MaxCompute Lightning是否限制查询的数据量？查询多大规模的数据性能较好?

  A：目前每次查询对单表的扫描数据量限制为1TB，数据量越小查询性能越好。

  说明:

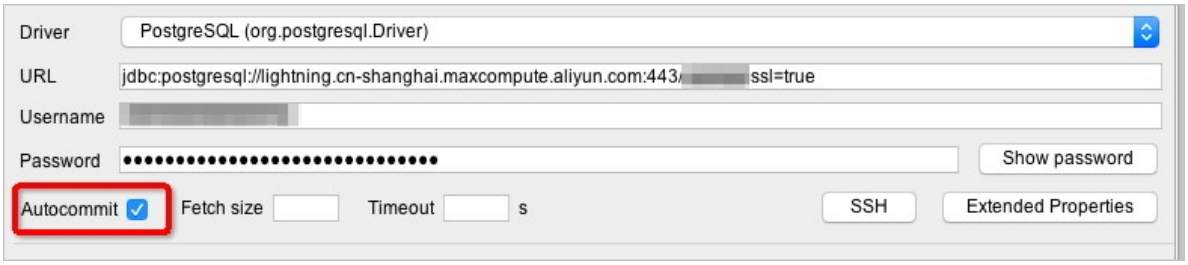
  建议不要扫描超过100GB的表数据。超过100GB的表数据虽然仍可查询，但查询性能会随数据规模增长逐渐下降。如果需要扫描量超过100GB的查询，建议您根据性能表现考虑使用MaxCompute SQL。
- Q：使用BI工具，通过拖拽方式选择一张分区表进行分析时，提示报错：`ERROR: AXF Exception: specified partitions count in odps table: <project_name.table_name> is: xxx, exceeds the limitation of xxx, please add stricter partition filter`。

  A：MaxCompute Lightning为保障查询性能，对分区表的分区数量进行了限制，一次查询所扫描的单表最大分区数量不能超过1024。由于部分BI工具使用拖拽方式选择表直接进行分析，不能BI前端指定分区条件，导致请求扫描的分区数超限制、触发了Lightning限制而报错提示。建议先对查询的数据表进行加工处理，处理为非分区表或分区小于1024的表再进行分析。
- Q：连接时提示创建数据连接失败：`ERROR: SSL required`。

  A：MaxCompute Lightning要求SSL连接服务，需要客户端指定以SSL方式连接。如果使用客户端工具，可以选择SSL连接选项。如果没有相关选项，可以在JDBC URL连接串中增加SSL参数，需要替换为您项目所在region的endpoint、连接的项目名称，例如`jdbc:postgresql://lightning.cn-shanghai.maxcompute.aliyun.com:443/myproject?ssl=true`。

- Q：使用Workbench/J客户端查询时提示报错`Error:current transaction is aborted ,commands ignored until end of transaction block`。

  A：使用的客户端请勾选Autocommit选项。



- Q：请求连接时返回提示报错`ERROR: Failed to login the project: <project name>. Lightning service has not been activated for your project, please contact the administrator.`。

  A：MaxCompute项目还没有开通Lightning服务，需要申请开通后使用。

- Q：MaxCompute项目名称大小写敏感吗?

  A：服务连接时，MaxCompute项目名称大小写敏感，连接时的项目名称请全部使用小写连接。

- Q：是否推荐直接通过Lightning拉取全量明细数据?

  A：通过SQL客户端工具/BI工具对数据量大的表执行`select *` 不加限制条件时，大量数据需要从服务侧端抽取到客户端侧，从而导致查询时间将大大延长。应尽量在服务器侧对明细数据进行分析，返回聚合后的较少量的数据以降低端到端的查询时延，减少或避免直接通过Lightning拉取明细数据。

- Q：Lightning如何处理数字开头的字段?

  A：MaxCompute支持以数字开头的字段，例如`1_day_active_users`。Lightning使用的Postgres的SQL语法，不支持数字开头的字段。遇到MaxCompute中数字开头的字段时，在Lightning的SQL查询中可以对该字段增加双引号进行查询。例如：`select bizdate ,"1_day_active_users","7_day_active_users" from t_active_users;`。

- Q：在应用程序中使用JDBC连接MaxCompute Lightning服务时建议如何配置?

  A：在应用程序中使用JDBC连接MaxCompute Lightning服务时，建议配置prepareThreshold=0，autocommit=True。

- Q：是否可以通过Psycopg2来连接访问Lightning?

  A：可以，请参考如下示例。

```
#!/usr/bin/env python
# coding=utf-8
```

```
import psycopg2
import sys

def query_lightning(lightning_conf, sql):
    """Query data through Lightning by sql

    Args:
        lightning_conf: a map contains settings of 'dbname', 'user',
 'password', 'host', 'port'
        sql:  query submit to Lightning

    Returns:
        result: the query result in format of list of rows
    """
    result = None
    conn = None
    conn_str = None
    try:
        conn_str = ("dbname={dbname} "
                    "user={user} "
                    "password={password} "
                    "host={host} "
                    "port={port}").format(**lightning_conf)
    except Exception, e:
        print >> sys.stderr, ("Invalid Lightning' configuration "
                              "{}".format(e))
        sys.exit(1)

    try:
        conn = psycopg2.connect(conn_str)
        conn.set_session(autocommit=True)  # This will disable
transaction
        # started with keyword BEGIN,
        # which is currently not
        # supported by Lightning' public service

        cur = conn.cursor()
        # execute Lightning' query
        cur.execute(sql)
        # get result
        result = cur.fetchall()
    except Exception, e:
        print >> sys.stderr, ("Failed to query data through "
                              "Lightning: {}".format(e))
    finally:
        if conn:
            conn.close()

    return result

if __name__ == "__main__":
    # step1. setup configuration
    lightning_conf = {
        "dbname": "project",
        "user": "accessId",  # AccessId
        "password": "accessKey",  # AccessKey
        "host": "lightning.cn-shanghai.maxcompute.aliyun-inc.com",
 # Lightning Endpoint
        "port": 443
    }

    # step2. issue a query
```

```
    result = query_lightning(lightning_conf, "select count(1) from
dual")
    # step3. print result
    if result:
        for i in xrange(0, len(result)):
            print
            "Got %d row from Lightning:%s" % (i + 1, result[i])
```

- **Q：使用MaxCompute Lightning有哪些性能优化建议?**

  A：

  1. **使用分区表，并在查询时指定分区，减少扫描的数据量。**
  2. **查询中只对需要列进行查询，减少对不必要列的查询。例如`select * from table`修改为`select a,b from table;`**
  3. **对于数据量较大的表，查询主要以等值(例如`select id,value from table where id=123)`查询为主要查询场景时，建议可以创建或者将原表修改为Hash Clustering表，利用CLUSTERED字段加速该字段的filter效率.**
  4. **对于数据量不大但查询时长明显较大时，需要判断是否存在数据倾斜问题:可以通过ANALYZE EXPLAIN方式查看执行计划，获取诊断信息。**

11 图模型

11.1 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，需要您同时在代码中维护点与边的关系。

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

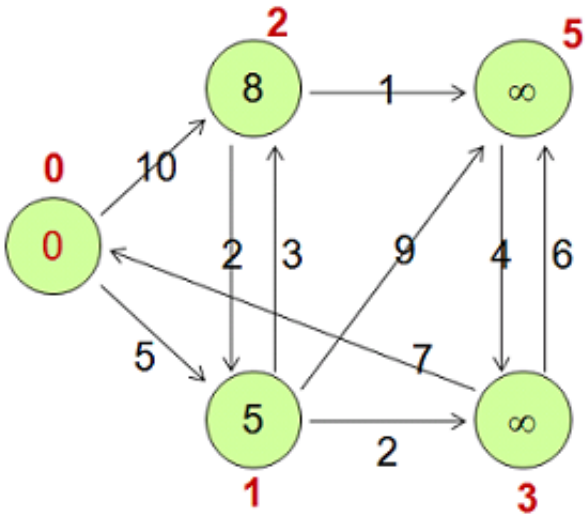
MaxCompute Graph能够处理的图必须是一个由点（Vertex）和边（Edge）组成的有向图。由于MaxCompute仅提供二维表的存储结构，因此需要您自行将图数据分解为二维表格式存储在MaxCompute中。

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，上图可以表述为以下二维表格式。

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 >]>

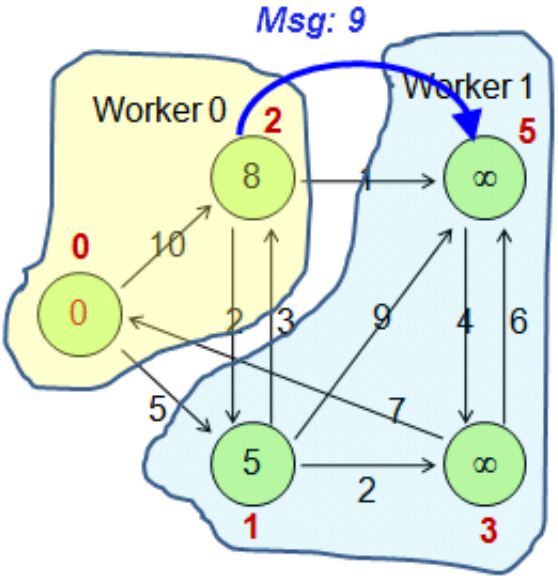
Graph 程序逻辑

Graph程序主要包含图加载、迭代计算、迭代终止等处理步骤。

- 图加载

  图加载包含两个步骤：

  - 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，则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，详情请参见Aggregator机制概述。
  - 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，无需人工干预。

- 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。

  Graph程序的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();
```

使用限制

详情请参见MaxCompute Graph作业输入输出限制。

11.2 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。

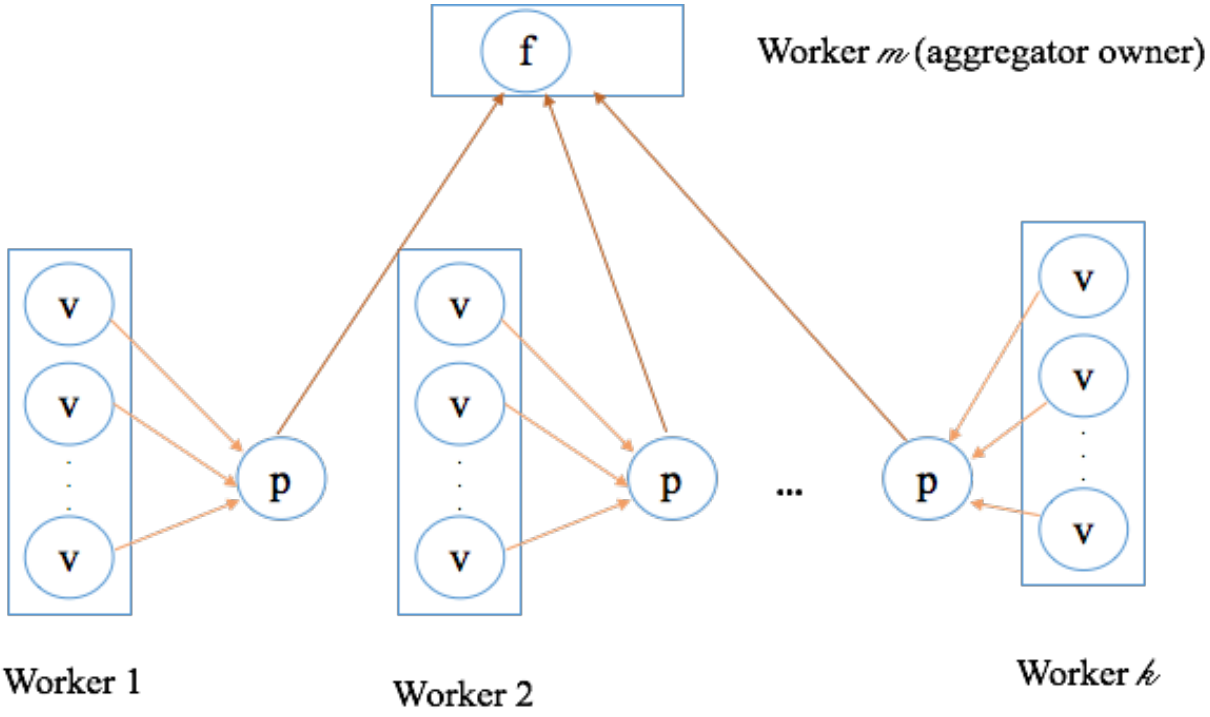
Aggregator是MaxCompute Graph作业中常用的特征，特别适用于解决机器学习问题。MaxCompute Graph中，Aggregator用于汇总并处理全局信息。

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Aggregator Owner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Aggregator Owner所在的Worker上。Aggregator Owner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（迭代）分发给所有Worker，供下一轮迭代使用。



Aggregator的API

Aggregator共提供了5个API供您实现。API的调用时机及常规用途如下：

- `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`AggregatorValue`。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`AggregatorValue`。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Aggregator Owner所在的Worker。

- `merge(value, partial)`

  该API执行于Aggregator Owner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。例如，发送到Aggregator Owner所在Worker的顺序为p1、p0、p2，则`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，即`merge()`不会被调用。

- `terminate(context, value)`

  当Aggregator Owner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Kmeans Clustering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, NullWritab
le> {
    @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维护了3个对象：

  - centroids是当前的K个中心点。如果样本是m维，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，按照上述API的顺序分析其实现。

  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;
    }
```

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 KmeansAggrValue createInitialValue(WorkerContext context)
    throws IOException {
    KmeansAggrValue av = (KmeansAggrValue)context.getLastAgg
regatedValue(0);
    // reset for next iteration
    av.sums.zero();
    av.counts.zero();
    return av;
}
```

该方法首先获取上一轮迭代的KmeansAggrValue，然后将sums和counts清零，只保留了上一轮迭代出的centroids。

3. aggregate()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3个方法执行于所有Worker上，实现局部聚合。在Aggregator Owner所在Worker执行的全局聚合相关操作如下：

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.getMaxIter
ation() - 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。

```
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。

11.3 Graph作业

本文为您介绍如何运行MaxCompute Graph作业。

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，则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());
//下面的参数中，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

- 通过GraphJob读取资源。

  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)
```

- 通过WorkerContext读取资源。

  您可以通过相应上下文对象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:**
>
> - Graph也支持在WorkerComputer的setup方法中读取资源，保存在Worker Value中，之后通过getWorkerValue方法获取。
> - 如果您需要在读取资源的同时并行进行资源处理，可以使用上述流接口。该读取方式可以减少内存的消耗。
> - 更多应用限制请参见应用限制。

11.4 SDK概述

本文为您介绍几种不同版本的Java SDK的相关配置信息。

如果您使用Maven，可以从Maven库中搜索odps-sdk-graph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。

主要接口	说明
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，例如修改图拓扑结构、发送消息、写结果、读取资源等。

11.5 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。

开发流程如下：

1. 编写Graph代码，使用本地调试进行基本的测试。
2. 进行集群调试，验证结果。

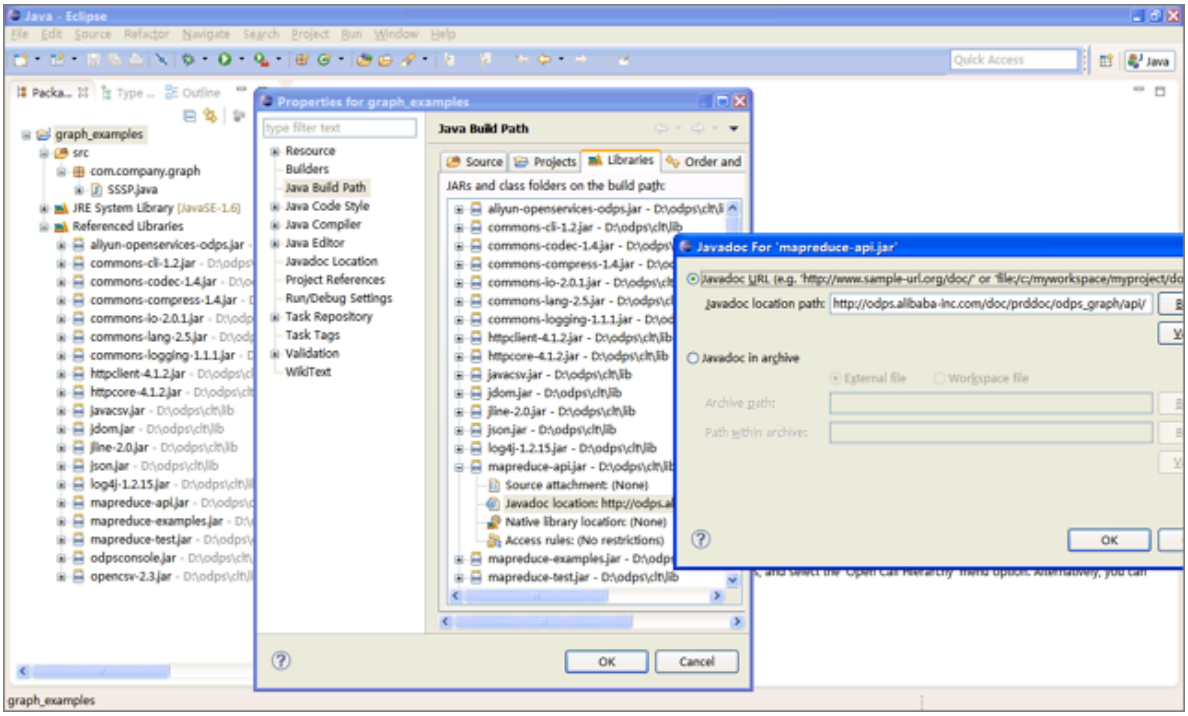
开发示例

本节以SSSP算法为例，为您介绍如何使用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，通常会先复制一个例子（例如单源最短距离），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

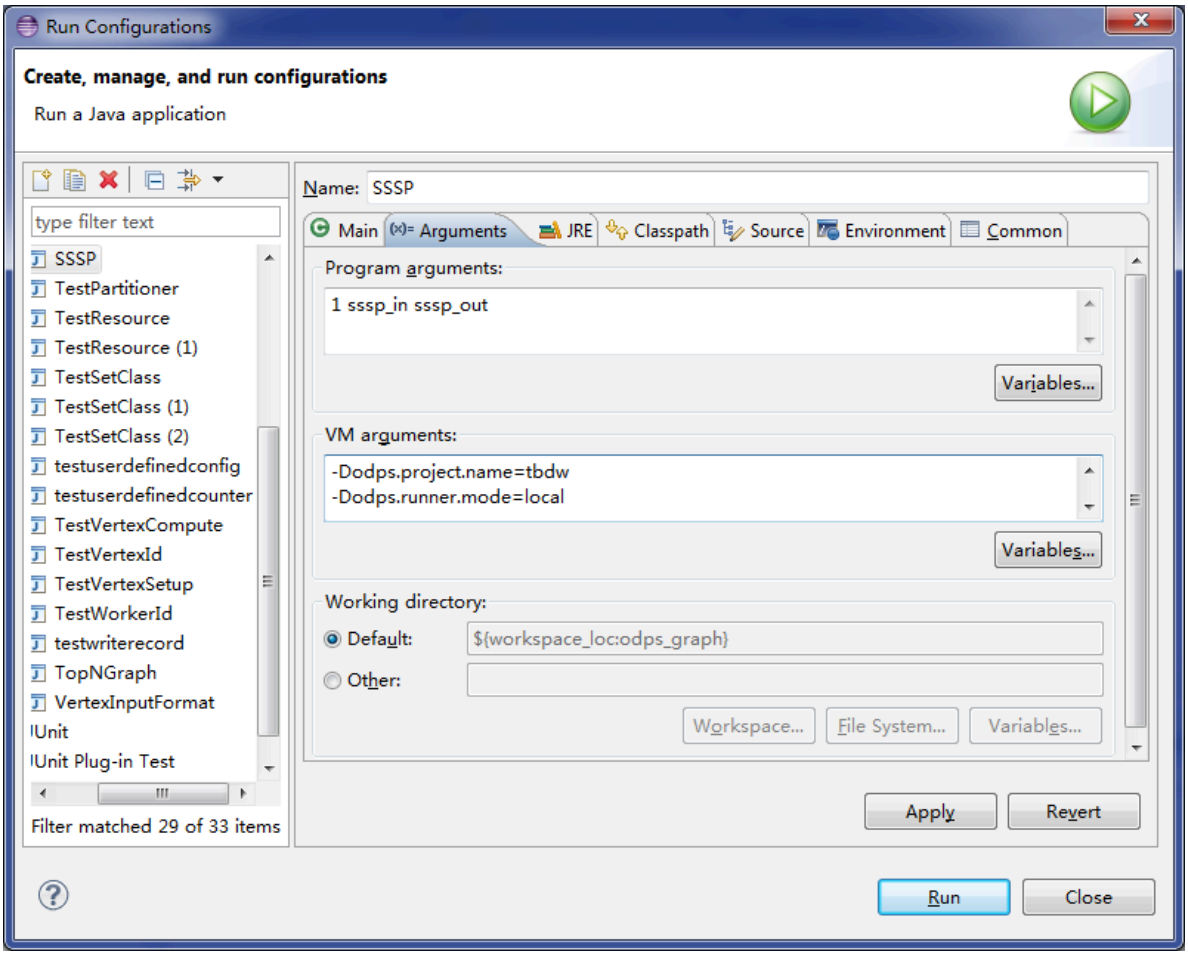
MaxCompute Graph支持本地调试模式，您可以使用Eclipse进行断点调试。

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。

```
-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，输入数据如下所示。

```
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。

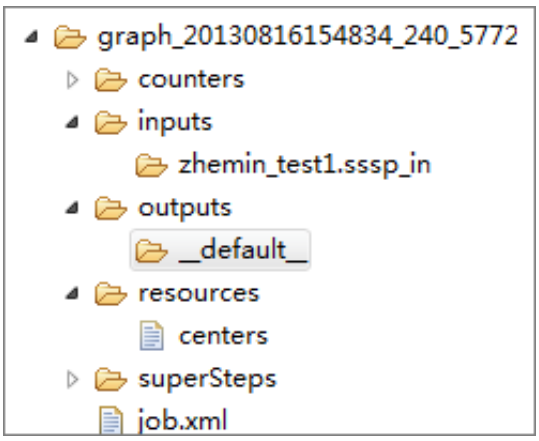
```
Counters: 3
        com.aliyun.odps.graph.local.COUNTER
                TASK_INPUT_BYTE=211
                TASK_INPUT_RECORD=5
                TASK_OUTPUT_BYTE=161
                TASK_OUTPUT_RECORD=5
 graph task finish
```

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的详细信息请参见编写Graph（可选）。

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中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的详细介绍请参见编写Graph（可选）。

性能调优

下文主要从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。

  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。

- 改进业务逻辑。

  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。您可通过如下方法进行改进。

  - 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中，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版本中是否有满足您需求的实现，或者等待MaxCompute升级新版本。

11.6 应用限制

本文为您介绍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、_、#、.和-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。

目前MaxCompute Graph处于公测阶段，各开服Region仅提供少量的资源用来公测使用。

- 华东2（上海）：仅提供2000CU。
- 华南1（深圳）：仅提供500CU。
- 其他开服Region：仅提供100CU。

11.7 示例程序

11.7.1 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。

11.7.2 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。

11.7.3 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。

11.7.4 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();
  }
}
```

11.7.5 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。

这一算法示例基于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");
  }
}
```

11.7.6 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");
  }
```

```
}
```

11.7.7 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");
  }
```

```
}
```

11.7.8 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");
  }
}
```

11.7.9 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");
  }
}
```

11.7.10 输入点表

本文为您提供输入点表的代码。

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、第三列表示边的权重
         **/
        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();
  }
}
```

11.7.11 输入边表

本文为您提供输入边表的代码。

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、第三列表示边的权重
         **/
        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();
  }
}
```

12 Job运行信息查看

12.1 使用Logview查看Job信息

Logview是MaxCompute Job提交后查看和Debug任务的工具。

通过Logview可看到一个Job的如下内容：

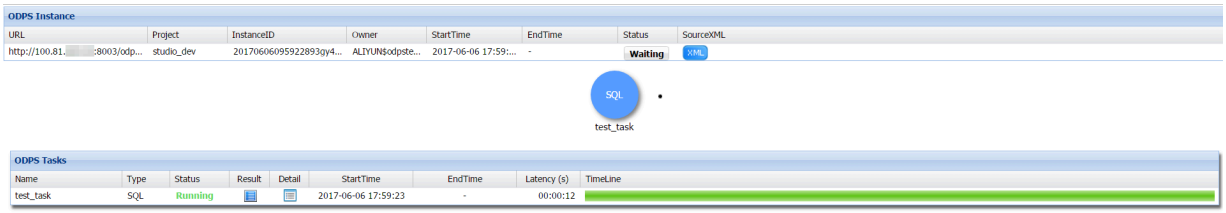
- 任务的运行状态。
- 任务的运行结果。
- 任务的细节和每个步骤的进度。

Job提交到MaxCompute后，会生成Logview的链接。您可以直接在浏览器上打开Logview链接，进入查看Job的信息。

> **说明：**
> 每个Job的Logview页面的有效期为一周。

Logview功能组件

下面结合具体的Logview Web UI界面，为您介绍每个组件的含义。



Logview的首页分成上下两部分：

- Instance信息
- Task信息

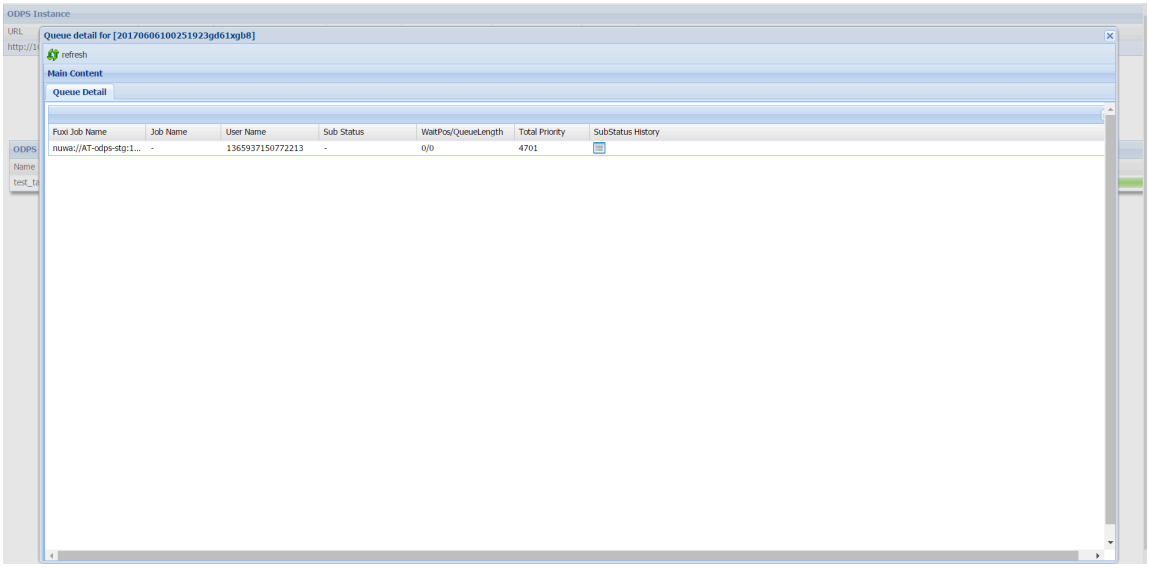
Instance信息

在Logview页面中，上半部分是您提交的SQL任务对应的MaxCompute Instance信息，包含URL链接、项目、InstanceID、提交人、开始时间、结束时间和状态（Status）等。

- Instance信息部分，您可以单击Status查看当前队列的Status详细信息，包含四种状态：
  - Waiting：说明当前作业正在MaxCompute中处理，并没有提交到分布式调度系统（Fuxi）中运行。
  - Waiting List : n：说明作业已提交至分布式调度系统（Fuxi）排队等待运行，当前在队列中处于第n位。
  - Running：作业在分布式调度系统（Fuxi）中运行。

> 说明:
> 处于Terminated状态的Instance由于作业已处理完成，因此不存在队列信息。

- 单击Status查看队列详细信息：
  - Sub Status：表示当前子状态信息。
  - WaitPos：表示排队位置，如果是0表示正在运行，如果为-表示尚未到Fuxi。
  - QueueLength：表示Fuxi中总的队列长度。
  - Total Priority：表示作业运行时经过系统判断后授予的优先级。
  - SubStatus History：单击后，可以查看作业执行的详细历史状态，包含状态码、状态描述、开始时间、持续时间等（某些版本暂时无历史信息）。

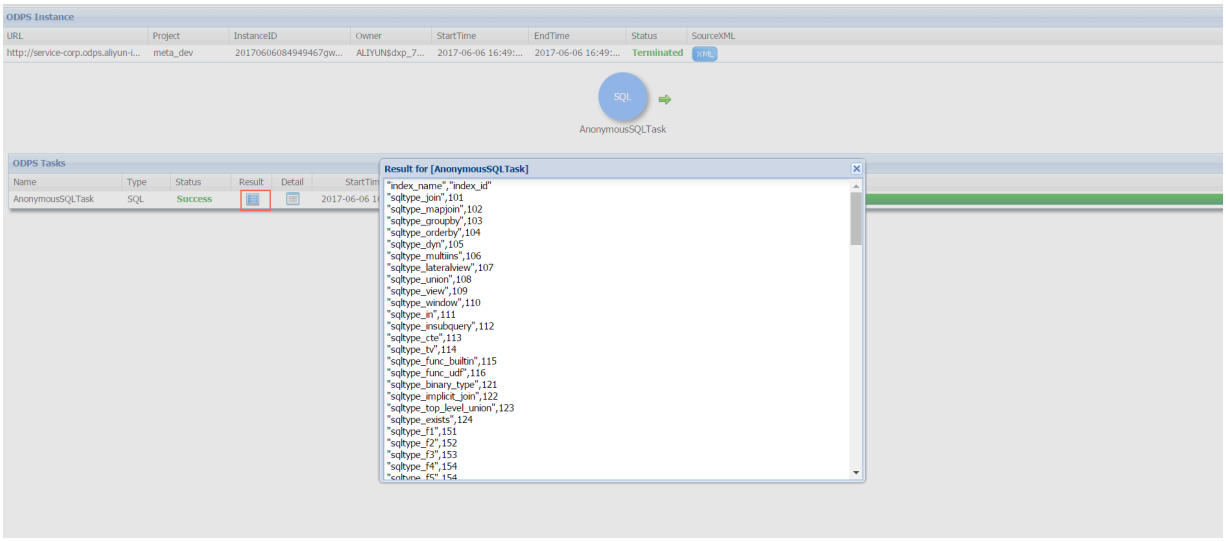


Task 信息

在Logview页面中，下半部分为该Instance对应的task信息，包括任务名称、类型、状态、运行结果（Result）、详细信息（Detail）、开始时间、结束时间、总运行时间（Latency）、进度。其中，Latency指运行总共消耗的时间，与其他页面的Latency含义类似。

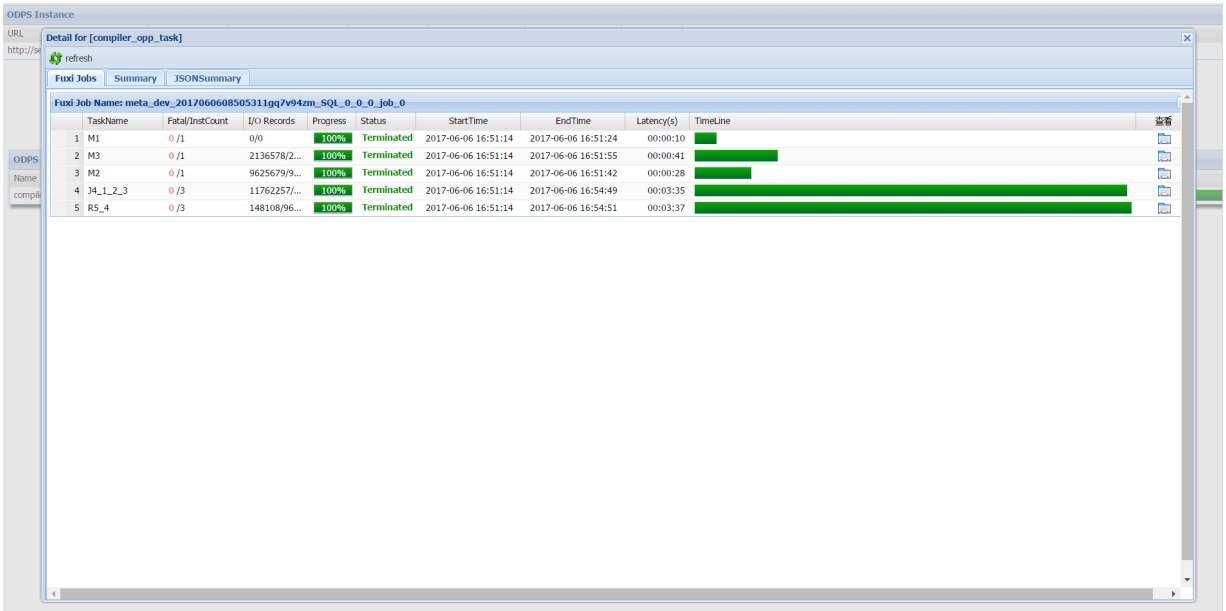
Result：

在Job运行结束后，可以看到运行结果。例如一条select SQL的结果如下图所示。



Detail：

一个Job在运行中和结束后，均可以单击Detail 来查看任务运行的具体情况。



在MaxCompute Task详细信息对话框中，您可以查看以下信息：

- **一个MaxCompute Task可以由一个或多个Fuxi Job组成。例如，如果您提交的SQL任务较为复杂，MaxCompute将自动向分布式调度系统（Fuxi）提交多个Fuxi Job。**
- **每个Fuxi Job可以由一个或多个Fuxi Task组成。例如，简单的MapReduce任务通常会产生两个Fuxi Task，即Map Task（M1）和Reduce Task（R2）。而一个比较复杂的SQL任务也可能产生多个Fuxi Task。**
- **查看所产生的Fuxi Task的任务名称。任务名称一般由字母和数字组成，其中字母表示任务类型（例如，M代表Map Task），数字标识任务编号以及依赖关系（例如，R5_4表示**

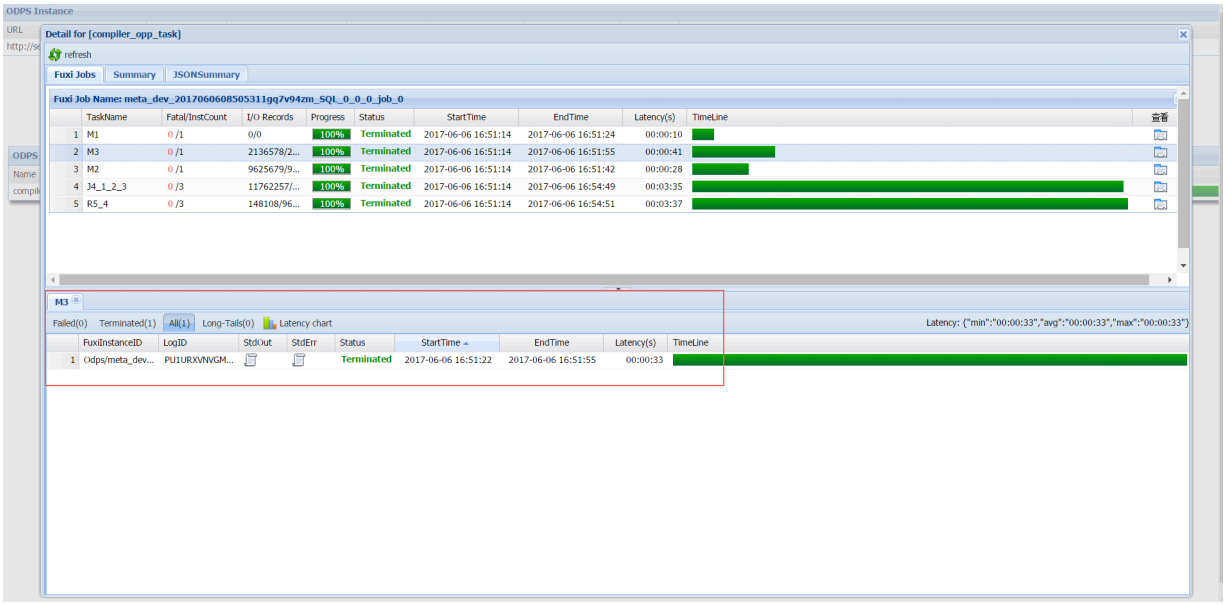
该Reduce Task需要J4任务执行结束后才会开始执行；J4_1_2_3则表示该Join Task需要M1、M2、M3这三个任务都执行完成后才会开始执行）。

> 说明:
> I/O Records表示该Fuxi Task的输入和输出的记录数。

- I/O Records表示这个task的输入和输出的records数。

单击Fuxi Task最右侧的查看列中的按钮或双击该Fuxi Task，您可以查看该Fuxi Task对应的Instance信息。

> 说明:
> 每个Fuxi Task由一个或多个Fuxi Instance组成。当Fuxi Task的输入数据量持续增大时，MaxCompute将自动为该Task启动更多的节点来处理数据，每个节点即对应一个Fuxi Instance。



在页面下方的Fuxi Instance信息部分，Logview将处于不同阶段的Instance进行分组展示。例如，您可以选择Failed组，查看运行出错的节点。同时，您可以单击StdOut和StdErr列中的按钮查看标准输出信息和标准错误信息。

> 说明:
> 您在所提交的MaxCompute任务中编写的打印信息也将在标准输出信息和标准错误信息中展示。

使用Logview排查问题

- 出错的任务

  当有任务出错时，您可以在Logview页面的Result中看到错误的提示信息，也可以在Detail页面中通过Fuxi Instance的stderr，查看具体某个Instance出错的信息。

- 数据倾斜

  运行缓慢有时是由于在某个Fuxi Task的所有Fuxi Instance中，有个别Instance形成长尾造成的。长尾的现象由同一个Task内任务分配不均导致。这时可以在任务运行完后，在Summay标签页中看运行结果。在每个Task中都可以看到如下输出：

  ```
  output records:
  R2_1_Stg1: 199998999  (min: 22552459, max: 177446540, avg: 99999499)
  ```

  此处如果min和max值相差较大，不在一个数量级，说明在这一阶段出现了数据倾斜。例如在执行字段Join时，某个字段中有一个值出现的比例很高，则在这一字段上执行Join就会出现数据倾斜。

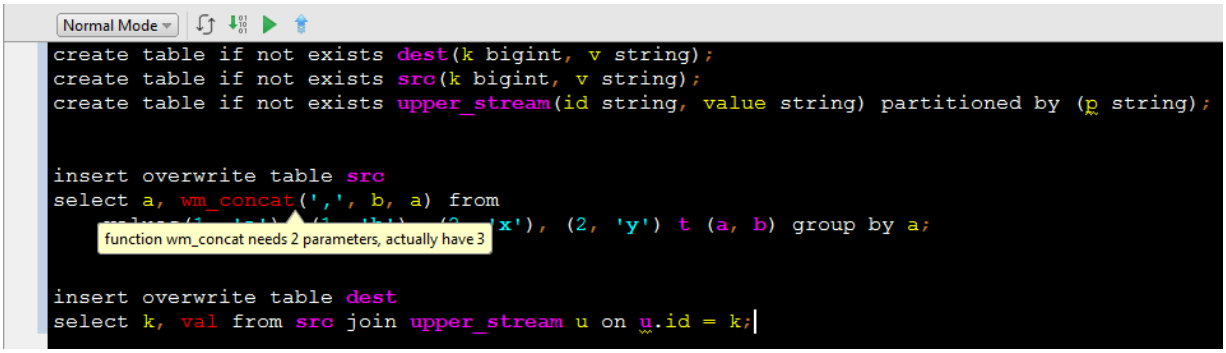
12.2 巧用MaxCompute编译器的错误和警告

MaxCompute编译器基于MaxCompute2.0新一代的SQL引擎，显著提升了SQL语言编译过程的易用性与语言的表达能力。本文将为您介绍编译器的易用性改进。

编译器的易用性改进

为了充分展示MaxCompute编译器的易用性改进，推荐您配合使用MaxCompute Studio。

首先，请安装 MaxCompute Studio，添加MaxCompute项目并创建工程，然后新建MaxCompute脚本文件，如下所示：

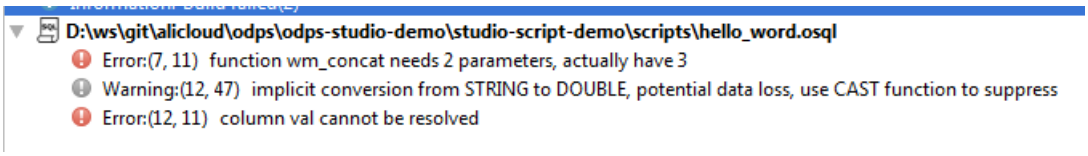


由上图可以发现以下问题：

- 第一个insert语句中wm_concat函数使用有错误。
- 第二个insert语句中有一个错误和一个警告，错误是列名写错了，警告则是MaxCompute中，比较Bigint与Double时，会隐式转换为Double。因为从String到Double是有可能在运

行时导致错误的转换，所以MaxCompute编译器会在此警告，请您确定此行为是否是您希望的。

鼠标停止在错误或者警告上，会直接提示具体错误或者警告信息。如果您不修改错误，直接提交，会被MaxCompute Studio阻拦，如下图所示：

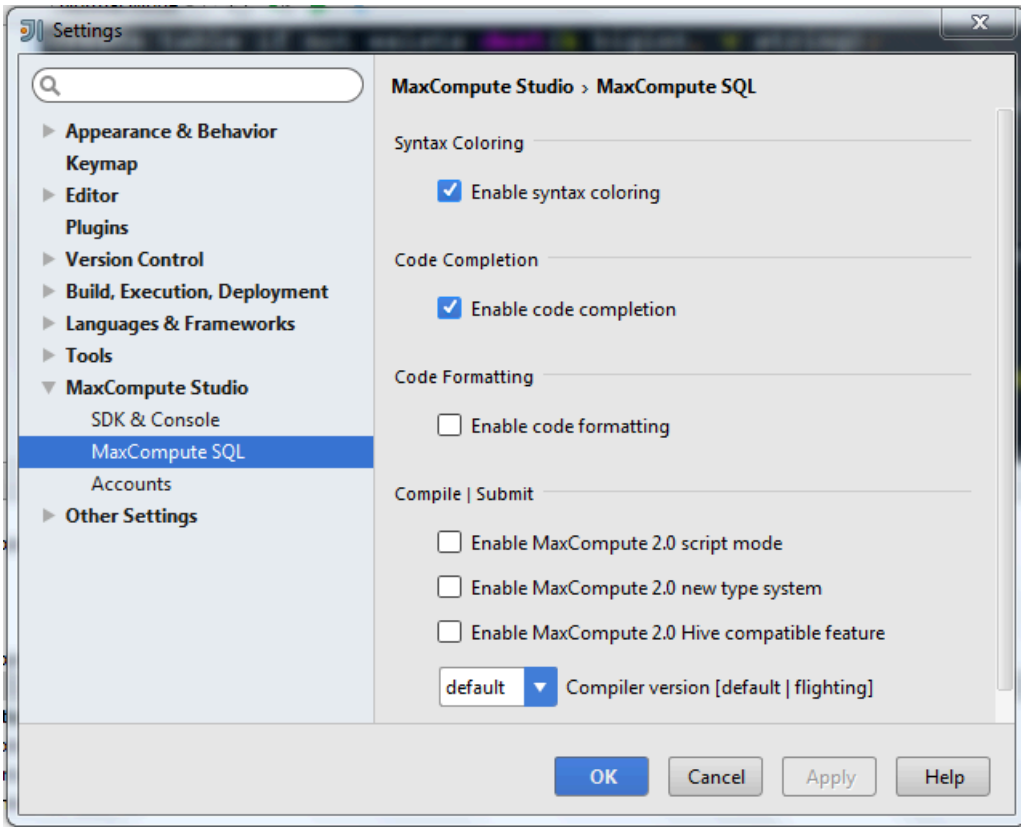


所以，请您按照提示修改错误和警告，如下所示：

```
create table if not exists dest(k bigint, v string);
create table if not exists src(k bigint, v string);
create table if not exists upper_stream(id string, value string) partitioned by (dt string);


insert overwrite table src
select a, wm_concat(',', b) from
    values(1, 'a'), (1, 'b'), (2, 'x'), (2, 'y') t (a, b) group by a;


insert overwrite table dest
select k, value from src join upper_stream u on u.id = string(k);
```

修改完毕后，再次提交脚本，便可以顺畅运行。

您也可以通过MaxCompute Studio把所有警告都设定为错误，如下图所示：



通过以上设置，可以保证不会不小心漏掉任何有可能的错误。

建议您在提交任何脚本之前，都使用MaxCompute Studio对脚本进行静态编译检查，并强烈推荐将警告设定为错误，在提交前修改所有的警告，这样可以避免浪费时间和资源。另外，提交有错误的脚本会被扣您的计算健康分，导致以后提交任务的优先级被下调，未来没有修改的警告也会被纳入到健康分体系，所以使用MaxCompute Compiler和Studio，可以永不降级。

警告中有很多情况是不安全的隐式类型转换，如果确实是您想要的转换，可以用cast (xxx as) 的方式消除警告；如果觉得这么写麻烦，MaxCompute编译器还提供一种简洁的方式（xxx），如上面修改过的脚本所示。

13 MaxCompute开源支持

MaxCompute是阿里巴巴自研的一站式的快速、完全托管的TB/PB级数据仓库解决方案。本章节将为您介绍MaxCompute相关的开源功能。

SDK

MaxCompute提供Java SDK和Python SDK接口实现创建、查看、删除MaxCompute表等操作。通过SDK，您可以通过编辑代码灵活地操作MaxCompute。MaxCompute提供的SDK如下：

- Java SDK

  使用说明可参考MaxCompute文档SDK 参考 > Java SDK。

  较为常用的MaxCompute核心接口详情请参见ODPS SDK Java DOC。更多信息可参考GitHub ODPS SDK for Java Developers。

  服务支持方式：您可通过访问官方文档和在线提交工单的方式获取支持。

- Python SDK

  PyODPS是MaxCompute的Python版本的SDK，提供对MaxCompute对象的基本操作和DataFrame框架，让您可以轻松地在MaxCompute上进行数据分析。更多详情请参见GitHub项目aliyun-odps-python-sdk和包括所有接口、类的细节等内容的详细PyODPS文档。

  - 了解更多关于PyODPS的开发内容，请参见PyODPS开发指南。在您开始使用PyODPS之前请先安装PyODPS，详细内容请参见PyODPS安装指南。
  - 关于PyODPS的社区信息请参见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PyODPS DataFrame概述。
  - 欢迎反馈问题和需求，加快PyODPS生态成长，更多详情请参见GitHub ODPS Python SDK and data analysis framework。

  服务支持方式：您可通过访问官方文档和在线提交工单的方式获取支持。

MaxCompute rodps

MaxCompute R语言插件：RODPS。具体使用说明请参考GitHub ODPS Plugin for R。

服务支持方式：在Github ODPS Plugin for R中留言或新建Issue。

MaxCompute JDBC

odps-jdbc是MaxCompute官方提供的JDBC驱动，它向Java程序提供了一套执行SQL任务的接口。项目托管在GiHub ODPS JDBC。

服务支持方式：在GitHub ODPS JDBC中留言或新建Issue。

Mars

Mars是一个基于张量的统一分布式计算框架。使用Mars进行科学计算，不仅使得完成大规模科学计算任务从MapReduce实现上千行代码降低到Mars数行代码，更在性能上有大幅提升。

Mars已经在Github上开源代码，您可以一起参与共建Mars。请参见GitHub开源代码Mars 。

关于Mars的更多说明信息请参见Mars开发指南。您还可通过云栖文章了解Mars，请参见Mars相关文档。

服务支持方式：在Github Mars中留言或新建Issue。

Data collector

MaxCompute数据集成工具插件，包括Flume、OGG、kettle、Sqoop、Hive 。

Data Collector是MaxCompute主要开源数据采集工具的集合，包括：

- Flume插件
- OGG插件
- Sqoop
- Kettle插件
- Hive Data Transfer UDTF

  从这些数据采集工具的实现技术上来看，Flume和OGG插件是基于DataHub的SDK实现，而Sqoop、Kettle以及Hive Data Transfer UDTF是基于Tunnel的SDK实现。DataHub和Tunnel分别是MaxCompute系统的实时和批量数据通道。Flume插件和OGG插件是面向于实时数据传输，而Sqoop，Kettle和Hive Data Transfer UDTF则是适合用于离线批量的数据传输。

参考源代码请参见GitHub源码Aliyun MaxCompute Data Collectors。插件说明请参见wiki。

服务支持方式：在Github Aliyun MaxCompute Data Collectors中留言或新建Issue。