ALIBABA CLOUD

阿里云

表格存储Tablestore

计算与分析

文档版本：20220707

阿里云

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
危险	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。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会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，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	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。	单击**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。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.MaxCompute

1.1. 使用MaxCompute访问表格存储

本文介绍如何在同一个云账号下实现表格存储和MaxCompute之间的无缝连接。

背景信息

MaxCompute是一项大数据计算服务，它能提供快速、完全托管的PB级数据仓库解决方案，使您可以经济并高效地分析处理海量数据。您只需通过一条简单的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。MaxCompute表是结构化的数据，而外部表可以不限于结构化数据。

表格存储与MaxCompute都有其自身的类型系统，两者之间的类型对应关系如下表所示。

Tablestore	MaxCompute
STRING	STRING
INTEGER	BIGINT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
BINARY	BINARY

准备工作

使用MaxCompute访问表格存储前，您需要完成以下准备工作：

1. 开通MaxCompute服务。
2. 创建工作空间。
3. 创建AccessKey。
4. 在RAM控制台授权MaxCompute访问表格存储的权限。具体请参见同账号授权和跨账号授权。
5. 在表格存储控制台创建实例和创建数据表。

   在本示例中，创建的表格存储实例和数据表信息如下：

   - 实例名称：cap1
   - 数据表名称：vehicle_track
   - 主键信息：vid(integer)，gt(integer)
   - 访问域名：`https://cap1.cn-hangzhou.ots-internal.aliyuncs.com`

     > **说明** 使用MaxCompute访问表格存储时，建议使用表格存储的私网地址。

   - 设置实例网络类型为**允许任意网络访问**。

步骤一：安装并配置客户端

1. 下载MaxCompute客户端并解压。

   > 说明 确保您的机器上已安装JRE 1.7或以上版本。

2. 编辑conf/odps_config.ini文件，对客户端进行配置，如下所示。

```
access_id=******************
access_key=**********************
 # Accesss ID及Access Key是用户的云账号信息，可登录阿里云官网，进入管理控制台 — accesskeys页面进行查看。
project_name=my_project
 # 指定用户想进入的项目空间。
end_point=https://service.odps.aliyun.com/api
 # MaxCompute服务的访问链接。
tunnel_endpoint=https://dt.odps.aliyun.com
 # MaxCompute Tunnel服务的访问链接。
log_view_host=http://logview.odps.aliyun.com
 # 当用户执行一个作业后，客户端会返回该作业的LogView地址。打开该地址将会看到作业执行的详细信息。
https_check=true
 # 决定是否开启HTTPS访问。
```

   > 说明 odps_config.ini文件中使用 `#` 作为注释，MaxCompute客户端内使用 `--` 作为注释。

3. 运行bin/odpscmd.bat，输入 `show tables;` 。

   如果显示当前MaxCompute项目中的表，则表示上述配置正确。

```
odps@ MCOTStest>show tables;

ALIYUN$document@aliyun-test.com:bank_data
ALIYUN$document@aliyun-test.com:result_table

OK
```

步骤二：创建外部表

创建一张MaxCompute的数据表（ots_vehicle_track）关联到Tablestore的某一张表（vehicle_track）。

关联的数据表信息如下。

- 实例名称：cap1

- 数据表名称：vehicle_track
- 主键信息：vid(int)，gt(int)
- 访问域名：`https://cap1.cn-hangzhou.ots-internal.aliyuncs.com`

```
CREATE EXTERNAL TABLE IF NOT EXISTS ots_vehicle_track
(
vid bigint,
gt bigint,
longitude double,
latitude double,
distance double ,
speed double,
oil_consumption double
)
STORED BY 'com.aliyun.odps.TableStoreStorageHandler' -- (1)
WITH SERDEPROPERTIES ( -- (2)
'tablestore.columns.mapping'=':vid, :gt, longitude, latitude, distance, speed, oil_consumpt
ion', -- (3)
'tablestore.table.name'='vehicle_track' -- (4)
)
LOCATION 'tablestore://cap1.cn-hangzhou.ots-internal.aliyuncs.com'; -- (5)
```

参数说明如下：

标号	参数	说明
(1)	com.aliyun.odps.TableStoreStorageHandler	MaxCompute内置的处理Tablestore数据的StorageHandler，定义了MaxCompute和Tablestore的交互，相关逻辑由MaxCompute实现。
(2)	SERDEPROPERITES	可以理解为提供参数选项的接口，在使用TableStoreStorageHandler时，有两个必须指定的选项，分别是tablestore.columns.mapping和tablestore.table.name。
(3)	tablestore.columns.mapping	必填选项。MaxCompute将要访问的Tablestore表的列，包括主键和属性列。其中，带 `:` 的表示Tablestore主键，例如本示例中的 `:vid`与 `:gt` ，其他均为属性列。在指定映射的时候，用户必须提供指定Tablestore表的所有主键，属性列无需全部提供，可以只提供需要通过MaxCompute来访问的属性列。
(4)	tablestore.table.name	需要访问的Tablestore表名。 如果指定的Tablestore表名错误（不存在），则会报错。MaxCompute不会主动创建Tablestore表。
(5)	LOCATION	指定访问的Tablestore的实例信息，包括实例名和endpoint等。

步骤三：通过外部表访问Tablestore数据

创建外部表后，Tablestore的数据便引入到了MaxCompute生态中，您可通过MaxCompute SQL命令来访问Tablestore数据。

```
//统计编号4以下的车辆在时间戳1469171387以前的平均速度和平均油耗。
select vid,count(*),avg(speed),avg(oil_consumption) from ots_vehicle_track where vid <4 and gt<1469171387  group by vid;
```

返回类似如下结果：

```
odps@ analysis_vehicle>select vid,count(*), avg(speed),avg(oil_consumption) from ots_vehicle_track where vid <4 and gt<1469171387  group by vid;

ID = 20170306160538185gsvlupk2
Log view:
http://logview.odps.aliyun.com/logview/?h=http://service.odps.aliyun.com/api&p=analysis_vehicle&i=20170306160538185gsvlupk2&token=ajhxdXZNWWVPcTNrWkFOS0xpSGtoM2tFWGEwPSxPRFBTX
09CTzoxNjA0MzM3MzgzMTc0NjE5LDE0ODk0MjExMzgseyJTdGF0ZW1lbnQiOlt7IkFjdGlvbiI6WyJvZHBzOlJlYWQiXSwiRWZmZWN0IjoiQWxsb3ciLCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnByb2plY3RzL2FuYWx5c2lzX3Zl
aGljbGUvaW5zdGFuY2VzLzIwMTcwMzA2MTYwNTM4MTg1Z3N2bHVwazIiXX1dLCJWZXJzaW9uIjoiMSJ9
Job Queueing.
--------------------------------------------------------------------------------
                STAGES        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  BACKUP
M1_job_0 .................    TERMINATED      1          1        0        0       0
R2_1_job_0 ...............    TERMINATED      1          1        0        0       0
--------------------------------------------------------------------------------
STAGES: 02/02    [==========================>>] 100%  ELAPSED TIME: 51.18 s
--------------------------------------------------------------------------------
Summary:

+------------+------------+------------+------------+
| vid        | _c1        | _c2        | _c3        |
+------------+------------+------------+------------+
| 0          | 47         | 0.11622589796672624 | 6.5155061805308145 |
| 1          | 47         | 0.11200649789552555 | 6.5673983001829885 |
| 2          | 47         | 0.098971763146085 | 6.738738527883797 |
| 3          | 47         | 0.11503916531605494 | 6.6038902692751895 |
+------------+------------+------------+------------+
```

1.2. 同账号授权

本文介绍如何使用阿里云访问控制的RAM角色实现同账号MaxCompute访问表格存储。

背景信息

RAM角色（RAM role）与RAM用户一样，都是RAM身份类型的一种。RAM角色是一种虚拟用户，没有确定的身份认证密钥，需要被一个受信的实体用户扮演才能正常使用。更多信息，请参见RAM角色概览。

操作步骤

1. 登录RAM控制台。
2. 创建用户角色AliyunODPSDefaultRole。
   i. 在左侧导航栏，选择**身份管理 > 角色**。
   ii. 单击**创建角色**。
   iii. 在**创建角色**面板，选择可信实体类型为**阿里云账号**，单击**下一步**。
   iv. 填写角色名称为AliyunODPSDefaultRole，并选择信任的云账号为当前云账号，单击**完成**。
3. 创建授权策略。
   i. 在左侧导航栏，选择**权限管理 > 权限策略**。
   ii. 单击**创建权限策略**。
   iii. 在**创建权限策略**页面，选择**脚本编辑**。

iv. 在**脚本编辑**页签，填写如下策略内容。

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

v. 单击**下一步：编辑基本信息**。

vi. 填写权限策略名称为AliyunODPSRolePolicy，单击**确定**。

4. 为RAM角色授权。

i. 在左侧导航栏，选择**身份管理 > 角色**。

ii. 找到AliyunODPSDefaultRole角色，单击**添加权限**。

iii. 在**添加权限**面板，设置选择权限为**自定义策略**，找到并单击AliyunODPSRolePolicy。

iv. 单击**确定**。

v. 单击**完成**。

5. 使用目标RAM用户扮演AliyunODPSDefaultRole角色。具体参见使用RAM角色。

完成以上步骤，目标RAM用户的下的MaxCompute即可访问表格存储。

1.3. 跨账号授权

本文介绍不同账号之间如何实现表格存储和MaxCompute之间的无缝连接。

> **说明** 如果需要了解同账号下的表格存储与MaxCompute对接操作，请参见同账号下使用MaxCompute访问表格存储。

准备工作

跨云账号需要两个阿里云账号，账号A将访问权限授予账号B，则运行MaxCompute时，账号B可以访问账号A下的表数据。账号基本信息如下：

> **说明** 以下信息仅为示例，在操作时请替换为实际使用的信息。

项目	表格存储	MaxCompute
阿里云账号名	账号A	账号B
UID	1234567890****	5678901234****

使用MaxCompute跨云账号访问表格存储前，您需要完成以下准备工作：

- 在MaxCompute产品详情页为账号B开通MaxCompute服务，并创建工作空间。具体操作，请参见创建工作空间。
- 分别获取账号A和账号B的AccessKey。具体操作，请参见获取AccessKey。
- 使用账号A创建可信实体为**阿里云账号**的RAM角色并设置信任策略内容。具体操作，请参见创建可信实体为阿里云账号的RAM角色和修改RAM角色的信任策略。

  > **说明** 策略内容中的5678901234****为账号B的UID。

  本示例中假设创建的RAM角色名称为AliyunODPSRoleForOtherUser，其信任策略内容示例如下，表示该RAM角色可以被账号B下的MaxCompute服务扮演。

```
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "5678901234****@odps.aliyuncs.com"
        ]
      }
    }
  ],
  "Version": "1"
}
```

- 记录账号A的RAM角色AliyunODPSRoleForOtherUser的ARN（例如acs:ram::1234567890****:role/aliyunodpsroleforotheruser），用于后续创建外表时配置。

- 创建权限策略并为账号A的RAM角色AliyunODPSRoleForOtherUser授权。具体操作，请参见创建自定义权限策略和为RAM角色授权。

  自定义权限策略示例如下，表示RAM角色具有读写账号A华东1（杭州）地域中名称为cap1的实例及其下所有表的权限。

```
{
  "Version": "1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ots:*",
      "Resource": [
      "acs:ots:cn-hangzhou:1234567890****:instance/cap1",
      "acs:ots:cn-hangzhou:1234567890****:instance/cap1/table/*"
      ],
      "Condition": {}
    }
  ]
}
```

关于自定义权限的更多信息，请参见自定义权限。

- 在表格存储控制台创建实例和创建数据表。具体操作，请参见创建实例和数据表。

  在本示例中，创建的表格存储实例和数据表信息如下：

  - 实例名称：cap1
  - 数据表名称：vehicle_track
  - 主键信息：vid (integer)，gt (integer)
  - 访问域名：`https://cap1.cn-hangzhou.ots-internal.aliyuncs.com`

    > **说明** 使用MaxCompute访问表格存储时，建议使用表格存储的私网地址。

  - 确保实例网络类型配置为**允许任意网络访问**。

使用MaxCompute访问表格存储

跨账号访问的操作与同账号下的访问相同，只是在创建外部表时要使用角色ARN（即rolearn）。

账号B通过MaxCompute创建外部表，指定准备工作中创建的角色ARN来访问表格存储。

具体操作，请参见使用MaxCompute访问表格存储。其中，在步骤二创建外部表时，使用如下示例代码：

```
CREATE EXTERNAL TABLE ads_log_ots_pt_external
(
vid bigint,
gt bigint,
longitude double,
latitude double,
distance double ,
speed double,
oil_consumption double
)
STORED BY 'com.aliyun.odps.TableStoreStorageHandler'
WITH SERDEPROPERTIES (
'tablestore.columns.mapping'=':vid, :gt, longitude, latitude, distance, speed, oil_consumpt
ion',
'tablestore.table.name'='vehicle_track',
'odps.properties.rolearn'='acs:ram::1234567890****:role/aliyunodpsroleforotheruser'
)
LOCATION 'tablestore://cap1.cn-hangzhou.ots-internal.aliyuncs.com'
USING 'odps-udf-example.jar'
```

1.4. 使用AccessKey访问表格存储

除了授权方式外，您还可以在 MaxCompute 中使用 AccessKey 访问表格存储的数据。

准备工作

获取表格存储资源所属账号的AccessKeyId 和 AccessKeySecret，如果该 AK 是资源所属账号的子账号，那么该子账号至少需要对表格存储相关的资源具有以下权限：

```
{
  "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"
    }
  ]
}
--您也可以自定义其他权限
```

在 MaxCompute 中使用 AccessKey 访问表格存储

同授权方式不同的是，需要在创建外表时在 `LOCATION` 中显示写入 AK 信息，其格式为：

```
LOCATION 'tablestore://${AccessKeyId}:${AccessKeySecret}@${InstanceName}.${Region}.ots-inte
rnal.aliyuncs.com'
```

假设需要访问的表格存储资源的信息为：

AccessKeyId	AccessKeySecret	实例名称	区域	网络模式
abcd	1234	cap1	cn-hangzhou	内网访问

创建外表的语句为：

```
CREATE EXTERNAL TABLE ads_log_ots_pt_external
(
vid bigint,
gt bigint,
longitude double,
latitude double,
distance double ,
speed double,
oil_consumption double
)
STORED BY 'com.aliyun.odps.TableStoreStorageHandler'
WITH SERDEPROPERTIES (
'tablestore.columns.mapping'=':vid, :gt, longitude, latitude, distance, speed, oil_consumpt
ion',
'tablestore.table.name'='vehicle_track'
)
LOCATION 'tablestore://abcd:1234@cap1.cn-hangzhou.ots-internal.aliyuncs.com'
```

对数据访问的操作步骤请参考使用MaxCompute访问表格存储中的步骤3.通过外部表访问 Table Store 数据。

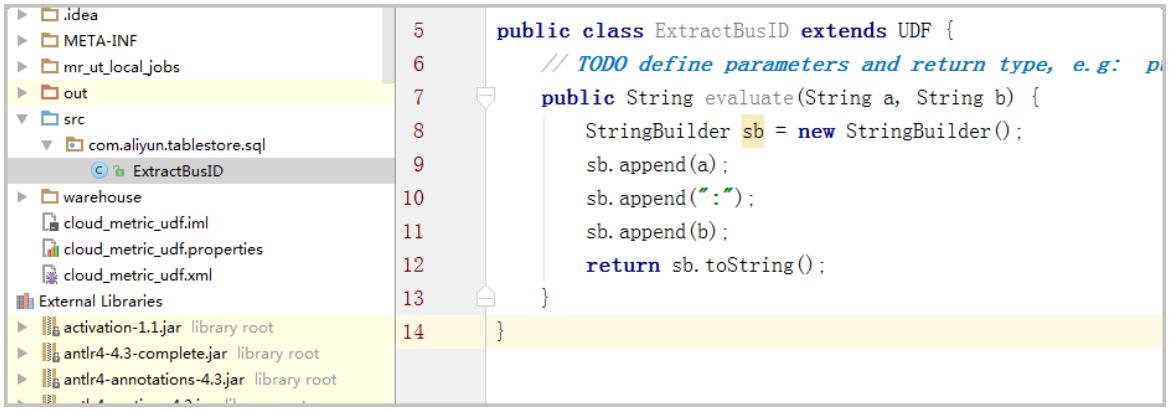
1.5. 使用UDF处理数据

如果您在表格存储里面的数据有着独特的结构，希望自定义开发逻辑来处理每一行数据，例如解析特定的JSON字符串，可以使用UDF（User Defined Function，即用户自定义函数）来处理。

操作步骤

1. 参考 MaxCompute Studio文档，在IntellijIdea中安装MaxCompute-Java/MaxCompute-Studio插件。插件安装完毕，即可直接开发。

下图是一个简单的UDF定义，将两个字符串连接。MaxCompute支持更复杂的UDF，包括自定义窗口执行逻辑等。更多信息，请参见开发和调试UDF。

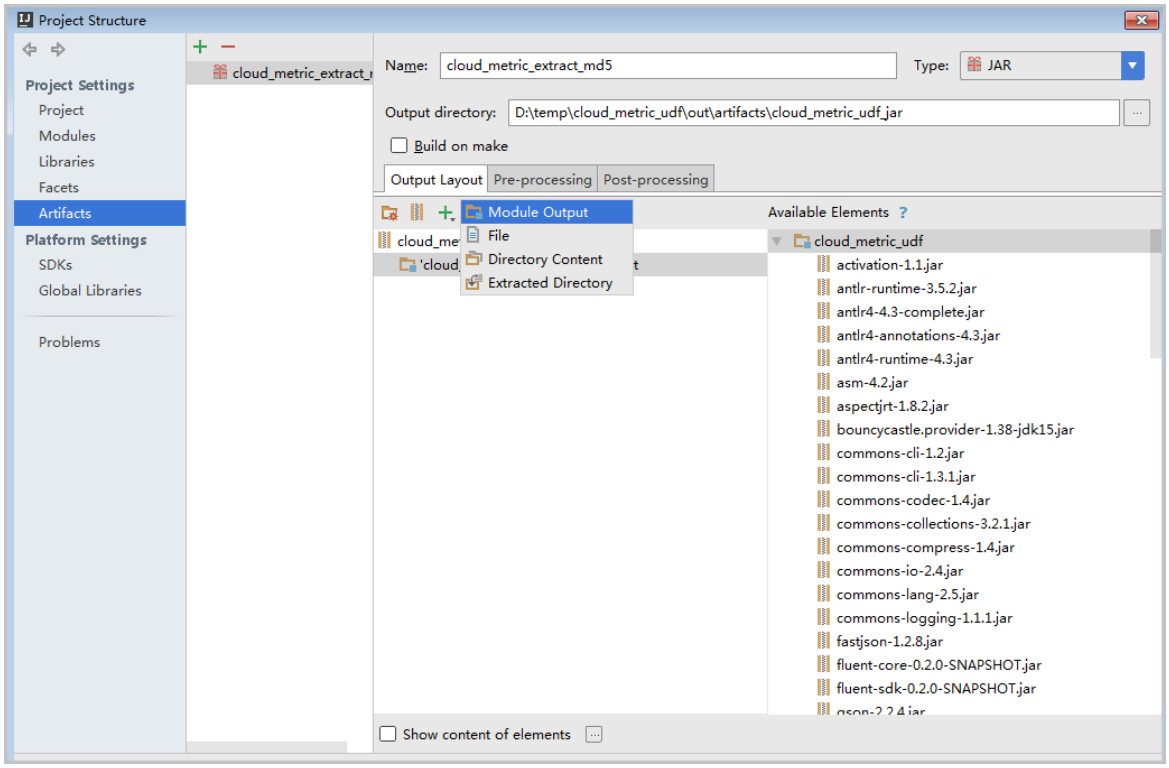


2. 参考MaxCompute Studio文档，在IntelliJ中安装MaxCompute-Java/MaxCompute-Studio插件。插件安装完毕，即可直接开发。

   下图是一个简单的UDF定义，将两个字符串连接。MaxCompute支持更复杂的UDF，包括自定义窗口执行逻辑等。更多信息，请参见开发和调试UDF。

3. 包之后可以上传到MaxCompute。

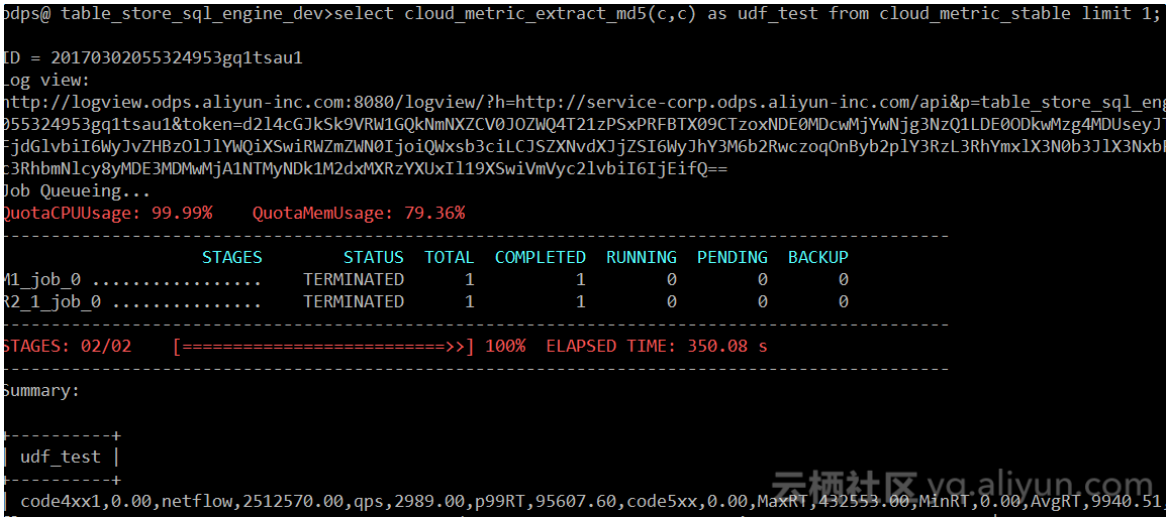
   选择**File > Project Structure > Artifacts**，输入*Name*和*Output directory*后，单击**+**选择输出模块。打包后通过ODPS Project Explorer来上传资源、创建函数，然后就可以在SQL中调用。



4. 运行**bin/odpscmd.bat**。

```
// 选出来1行数据，并将name/name传入UDF，返回两个string的累加.
select cloud_metric_extract_md5(name, name) as udf_test from test_table limit 1;
```

返回结果如下：



1.6. 常见问题

本文主要为您介绍使用MaxCompute访问表格存储的相关常见问题。

- FAILED: ODPS-0010000:System internal error - fuxi job failed, WorkerPackageNotExist

  原因：需要设置 set odps.task.major.version=unstructured_data。

- FAILED: ODPS-0010000:System internal error - std::exception:Message: a timeout was reached

  原因：一般情况下是表格存储的 endpoint 填写错误，导致 MaxCompute 无法访问。

- logview invalid end_point

  原因：在执行过程中，会返回 logview URL 地址，如果使用浏览器访问该地址返回错误，可能是配置不对，请检查 MaxCompute 配置。

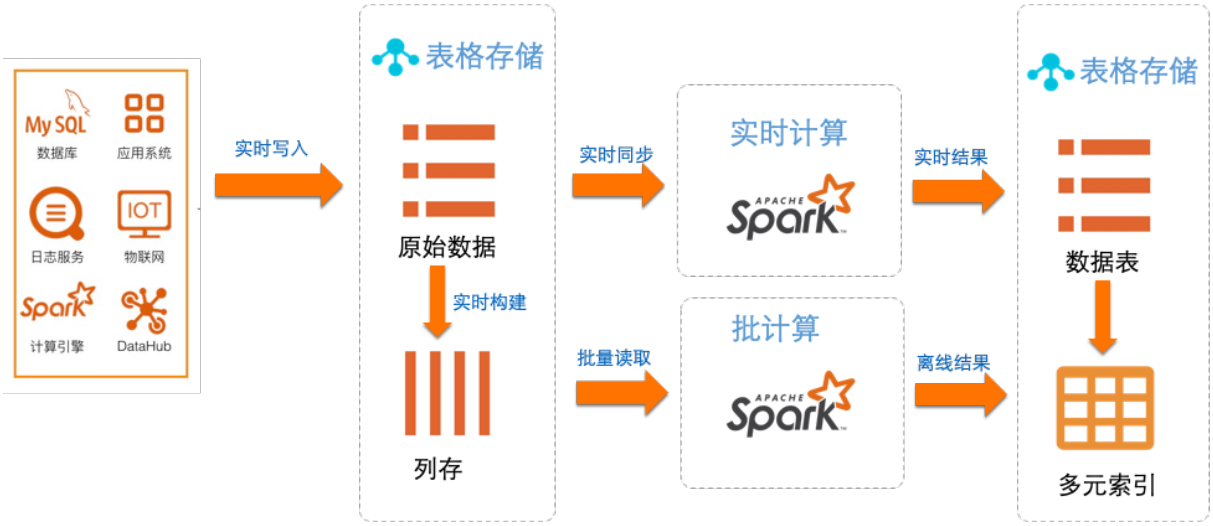
如果仍未解决问题，请提交工单。

2.Spark/SparkSQL

2.1. 概述

使用Spark计算引擎访问表格存储时，您可以通过E-MapReduce SQL或者DataFrame编程方式对表格存储中数据进行复杂的计算和高效的分析。

应用场景



功能特性

对于批计算，除了基础功能外，Tablestore On Spark提供了如下核心优化功能：

- 索引选择：数据查询效率的关键在于选择合适的索引方式，根据过滤条件选择最匹配的索引方式增加查询效率。表格存储的索引方式可选全局二级索引或者多元索引。

  > 说明
  >
  > 全局二级索引和多元索引的更多信息请参见海量结构化数据存储技术揭秘：Tablestore存储和索引引擎详解。

- 表分区裁剪：根据过滤条件进行逻辑分区（Split）的细化匹配，提前筛选出无效的Split，降低服务端的数据出口量。
- Projection和Filter下推：将Projection列和Filter条件下推到服务端，降低每个分区的数据出口量。
- 动态指定Split大小：支持调整每个逻辑分区(Split)中的数据量和分区数，每个Split会和RDD（Resilient Distributed DataSet）的Partition绑定，可用于加速Spark任务的执行。

  > 说明
  >
  > 通过ComputeSplitsBySize接口可以获取逻辑分区(Split)，该接口将全表数据在逻辑上划分成若干接近指定大小的分片，并返回这些分片之间的分割点以及分片所在机器的提示。一般用于计算引擎规划并发度等执行计划。

对于流计算，基于通道服务，利用CDC（数据变更捕获）技术完成Spark的mini batch流式消费和计算，同时提供了at-least-once一致性语义。在流计算中每个分区和RDD的Partition一一绑定，通过扩展表的分区，可以完成数据吞吐量的线性扩展。

场景案例

- Tablestore结合Spark的流批一体SQL实战
- 海量结构化数据的冷热分层一体化
- Tablestore+Delta Lake（快速开始）

使用方式

根据业务场景可以选择通过E-MapReduce SQL或者DataFrame编程方式使用Spark访问表格存储。

- E-MapReduce SQL方式

  此方式使用标准的SQL语句进行业务数据操作和访问，操作便捷，可以将已有的业务逻辑进行无缝迁移。

- DataFrame方式

  此方式需要一定的编程基础，但可以组合实现复杂的业务逻辑，适用于比较复杂和灵活的场景。

数据访问方式

表格存储为Spark批计算提供KV查询（数据表或全局二级索引）和多元索引查询两种数据访问方式，以支持海量结构化数据快速读写和丰富的查询分析能力。

两种数据访问方式的区别如下：

- KV查询方式在过滤字段是主键的场景下效率较高，但不适合过滤字段变动较大且过滤字段中非主键列较多的场景，KV查询方式也不支持地理位置查询。
- 多元索引查询方式适用于如下数据访问场景中：

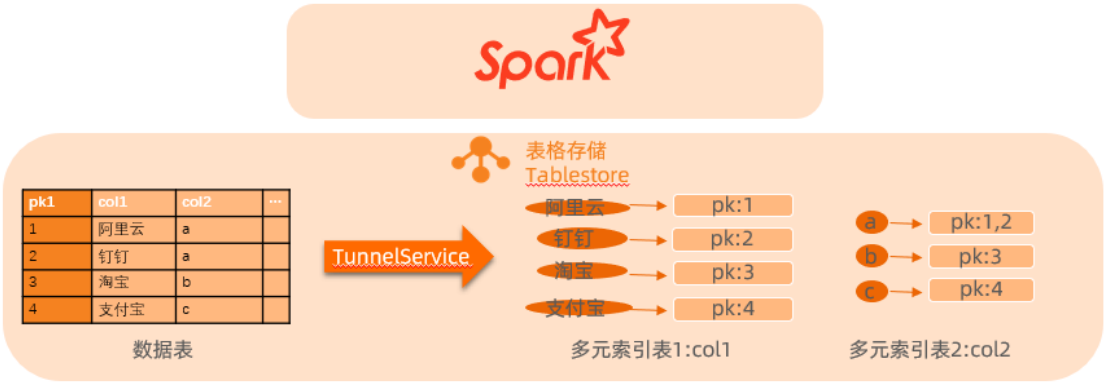
  > **说明**
  >
  > 多元索引基于倒排索引和列式存储，可以解决大数据的复杂查询难题，提供类似于ElasticSearch的全文检索、模糊查询、地理位置查询、统计聚合等查询和分析功能。

  - 少量且对延时要求较高的实时数据分析场景。
  - 非主键的过滤字段较多，过滤字段无法被全局二级索引主键或者数据表主键包含。
  - 过滤字段的筛选效率较高，可以通过某一字段条件过滤掉大部分数据。

    例如 `select * from table where col = 1000;` 中，col是非主键列，且col = 1000字段条件可以过滤掉大部分数据。
  - 查询条件中包含地理位置查询。

结合下图以 `select * from table where col1 like '阿%' or col2 = 'a';` SQL语句为例介绍两种查询方式的使用样例。

- 通过多元索引方式访问数据时，在多元索引的col1中可以获取col1的值为'阿%'的一行数据pk1 = 1，在多元索引的col2中可以获取col2的值为'a'的两行数据pk1 = 1和pk1 = 2，然后将两次中间结果的数据进行union，即可得到满足条件的数据 `pk1 = 1，col1 = '阿里云'，col2 = 'a'` 。



- 通过KV查询方式访问数据时，查询主体是表格存储的数据表，数据表只能通过主键查询。如果查询的SQL语句中的过滤字段不是数据表的主键，则需要进行全表扫描。

  由于col1不是数据表的主键，表格存储会进行全表扫描找到col1的值为'阿%'的一行数据；由于col2不是数据表的主键，表格存储会再次进行全表扫描找到col2的值为'a'的两行数据，然后将两次中间结果的数据进行union。

  此时也可以通过构建一个主键为col1、col2的索引表支持该查询，但此种方式的灵活性较低。

2.2. 数据类型

了解Spark数据类型、Scala中的值类型、表格存储中多元索引数据类型和表格存储表中数据类型的对应关系。使用过程中请确保Spark、Scala和表格存储中字段或值的数据类型相匹配。

基础数据类型

基础数据类型的支持情况请参见下表。

Spark数据类型	Scala中的值类型	多元索引数据类型	表中数据类型
ByteType	Byte	Long	Integer
ShortType	Short	Long	Integer
IntegerType	Int	Long	Integer
LongType	Long	Long	Integer
FloatType	Float	Double	Double
DoubleType	Double	Double	Double

Spark数据类型	Scala中的值类型	多元索引数据类型	表中数据类型
StringType	String	Keyword/Text	String
BinaryType	Array[Byte]	Binary	Binary
BooleanType	Boolean	Boolean	Boolean
地理坐标（String JSON）	String（JSON）	*Geopoint*	String（JSON）

地理位置类型（Geopoint类型）

多元索引支持地理位置查询方式，将其提供到计算层，使Spark在能查询分析基础类型数据的同时，也可以结合地理位置对数据进行查询分析。

地理位置查询包括地理距离查询、地理矩形查询和地理多边形范围查询三种查询方式。地理位置查询广泛应用于物联网设备位置信息、骑手订单、打卡位置信息、快递地理信息等场景中。使用方式如下：

- 使用表格存储的多元索引查询，详情请参见地理距离查询。
- 使用Spark SQL方式查询。
  - 地理半径圆查询

    ```
    select * from table where  val_geo = '{"centerPoint":"3,0", "distanceInMeter": 100000}' and name like 'ali%'
    ```

  - 地理矩形查询

    ```
    select * from table where geo = '{"topLeft":"8,0", "bottomRight": "0,10"}' and id in { 123 , 321 }
    ```

  - 地理多边形查询

    ```
    select * from table where geo = '{"points":["5,0", "5,1", "6,1", "6,10"]}'
    ```

地理位置数据类型的支持情况请参见下表。

Spark数据类型	Scala中的值类型	多元索引数据类型	表中数据类型
地理坐标（String JSON）半径圆	String（JSON）	Geopoint	STRING（JSON）
地理坐标（String JSON）矩形	String（JSON）	Geopoint	STRING（JSON）

Spark数据类型	Scala中的值类型	多元索引数据类型	表中数据类型
地理坐标（String JSON） 多边形	String（JSON）	Geopoint	STRING（JSON）

2.3. E-MapReduce SQL

2.3.1. 批计算

通过在E-MapReduce集群中使用Spark SQL访问表格存储。对于批计算，Tablestore on Spark提供索引选择、分区裁剪、Projection列和Filter下推、动态指定分区大小等功能，利用表格存储的全局二级索引或者多元索引可以加速查询。

前提条件

- 已创建E-MapReduce Hadoop集群。具体操作，请参见创建集群。

  创建集群时，请确保打开**挂载公网**开关，将集群挂载到公网，用于Shell远程登录服务器。

  > 说明
  >
  > 本文使用Shell命令演示，如果需要使用E-MapReduce的图形化页面进行数据开发。具体操作，请参见数据开发。

- 已上传emr-datasources_shaded_2.11-2.2.0-SNAPSHOT.jar包到EMR Header服务器。

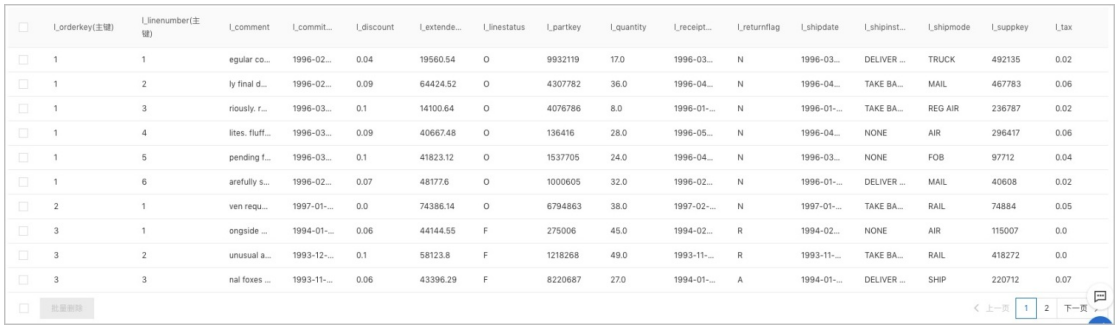
Spark连接表格存储数据表和全局二级索引

Spark连接到表格存储数据表和全局二级索引后，通过Spark外表查询数据时，系统会根据查询条件中设置的列条件自动选择索引表进行查询。

1. 在表格存储侧创建数据表或全局二级索引。

i. 创建表格存储的数据表。具体操作，请参见概述。

本示例中数据表名称为tpch_lineitem_perf，主键列为l_orderkey（LONG类型）、l_linenumber（LONG类型），属性列分别为l_comment（STRING类型）、l_commitdate（STRING类型）、l_discount（DOUBLE类型）、l_extendedprice（DOUBLE类型）、l_linestatus（STRING类型）、l_partkey（LONG类型）、l_quantity（DOUBLE类型）、l_receiptdate（STRING类型）等14列，数据条数为384016850，数据样例如下图所示。



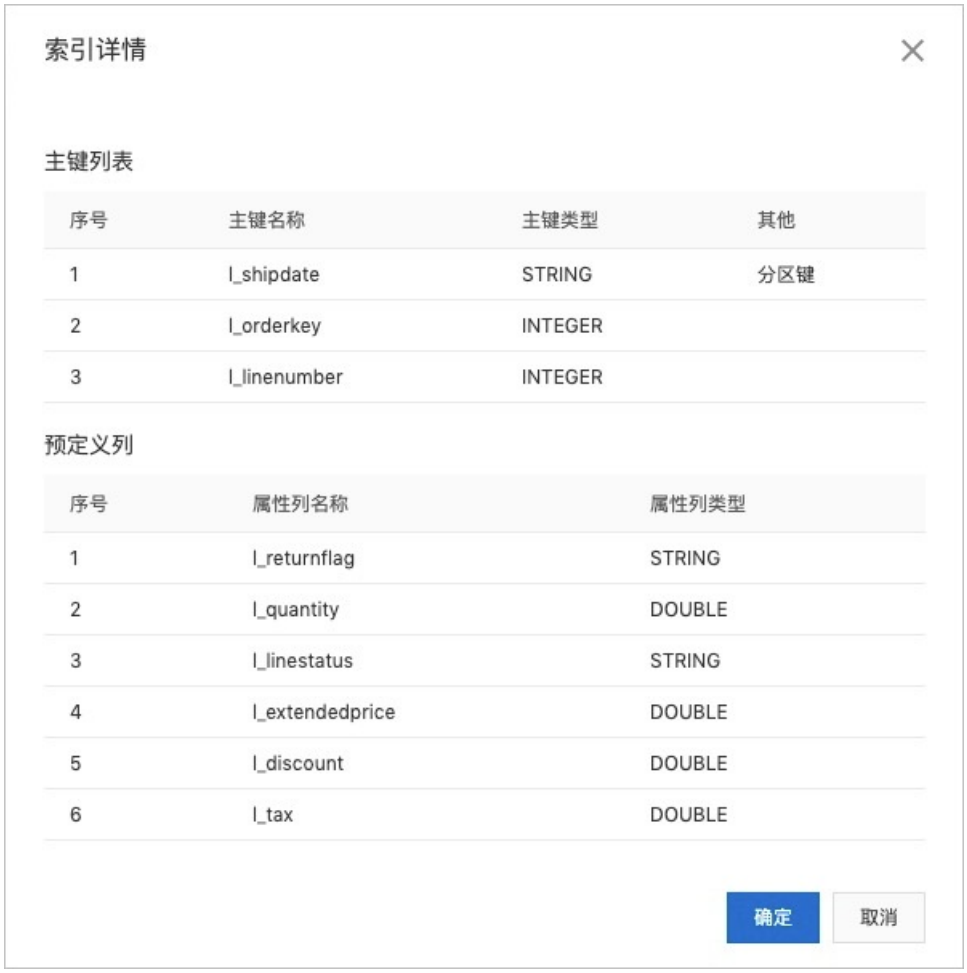
l_orderkey(主键)	l_linenumber(主键)	l_comment	l_commit...	l_discount	l_extende...	l_linestatus	l_partkey	l_quantity	l_receipt...	l_returnflag	l_shipdate	l_shipinst...	l_shipmode	l_suppkey	l_tax
1	1	egular co...	1996-02...	0.04	19560.54	O	9932119	17.0	1996-03...	N	1996-03...	DELIVER ...	TRUCK	492135	0.02
1	2	ly final d...	1996-02...	0.09	64424.52	O	4307782	36.0	1996-04...	N	1996-04...	TAKE BA...	MAIL	467783	0.06
1	3	riously. r...	1996-03...	0.1	14100.64	O	4076786	8.0	1996-01-...	N	1996-01-...	TAKE BA...	REG AIR	236787	0.02
1	4	lites. fluff...	1996-03...	0.09	40667.48	O	136416	28.0	1996-05...	N	1996-04...	NONE	AIR	296417	0.06
1	5	pending f...	1996-03...	0.1	41823.12	O	1537705	24.0	1996-04...	N	1996-03...	NONE	FOB	97712	0.04
1	6	arefully s...	1996-02...	0.07	48177.6	O	1000605	32.0	1996-02...	N	1996-01-...	DELIVER ...	MAIL	40608	0.02
2	1	ven requ...	1997-01-...	0.0	74386.14	O	6794863	38.0	1997-02-...	N	1997-01-...	TAKE BA...	RAIL	74884	0.05
3	1	ongside ...	1994-01-...	0.06	44144.55	F	275006	45.0	1994-02...	R	1994-02...	NONE	AIR	115007	0.0
3	2	unusual a...	1993-12-...	0.1	58123.8	F	1218268	49.0	1993-11-...	R	1993-11-...	TAKE BA...	RAIL	418272	0.0
3	3	nal foxes ...	1993-11-...	0.06	43396.29	F	8220687	27.0	1994-01-...	A	1994-01-...	DELIVER ...	SHIP	220712	0.07

批量删除 上一页 1 2 下一页

ii. （可选）在数据表上创建全局二级索引。具体操作，请参见使用SDK。

> **说明**
>
> 当查询条件中需要使用数据表的非主键列，建议创建全局二级索引加速查询。

全局二级索引支持在指定列上建立索引，生成的索引表中数据按指定的索引列进行排序，数据表的每一个数据写入都将自动以异步方式同步到索引表。



2. 在EMR集群侧创建Spark外表。

i. 登录EMR Header服务器。

ii. 执行如下命令启动spark-sql命令行，用于Spark外表创建和后续的SQL实战操作。

其中Spark的标准启动参数为 `--num-executors 32 --executor-memory 2g --executor-cores 2` ，可以根据具体的集群配置进行自定义调整。<Version>表示上传JAR包的版本信息，请根据实际填写，例如2.1.0-SNAPSHOT。

```
spark-sql --jars emr-datasources_shaded_2.11-<Version>.jar --master yarn --num-exec
utors 32 --executor-memory 2g --executor-cores 2
```

iii. 创建Spark外表同时连接全局二级索引。

- 参数

参数	说明
endpoint	表格存储实例访问地址，EMR集群中使用VPC地址。
access.key.id	阿里云账号的AccessKey ID。
access.key.secret	阿里云账号的AccessKey Secret。
instance.name	实例名称。
table.name	表格存储的数据表名称。
split.size.mbs	每个Split的大小，默认值为100 MB。
max.split.count	数据表计算出的最大Split数，并发数和Spark的Split个数对应，默认值为1000。
catalog	数据表的Schema定义。

- 示例

```
DROP TABLE IF EXISTS tpch_lineitem;
CREATE TABLE tpch_lineitem
USING tablestore
OPTIONS(
endpoint="http://vehicle-test.cn-hangzhou.vpc.tablestore.aliyuncs.com",
access.key.id="",
access.key.secret="",
instance.name="vehicle-test",
table.name="tpch_lineitem_perf",
split.size.mbs=10,
max.split.count=1000,
catalog='{"columns":{"l_orderkey":{"type":"long"},"l_partkey":{"type":"long"},"l_
suppkey":{"type":"long"},"l_linenumber":{"type":"long"},"l_quantity":{"type":"dou
ble"},"l_extendedprice":{"type":"double"},"l_discount":{"type":"double"},"l_tax":
{"type":"double"},"l_returnflag":{"type":"string"},"l_linestatus":{"type":"string
"},"l_shipdate":{"type":"string"},"l_commitdate":{"type":"string"},"l_receiptdate
":{"type":"string"},"l_shipinstruct":{"type":"string"},"l_shipmode":{"type":"stri
ng"},"l_comment":{"type":"string"}}}'
);
```

3. SQL查询实战。

   如下是不同查询需求的SQL查询样例，请根据实际业务组合使用SQL查询。

- 全表查询
  - SQL语句：`SELECT COUNT(*) FROM tpch_lineitem;`
  - SQL总耗时：36.199s、34.711s、34.801s，平均耗时35.237s。
- 主键查询
  - SQL语句：
    ```
    SELECT COUNT(*) FROM tpch_lineitem WHERE l_orderkey = 1 AND l_linenumber = 1;
    ```
  - 表格存储服务端：GetRow操作，平均耗时为0.585 ms。
- 非主键查询，未开启全局二级索引
  - SQL语句：`SELECT count(*) FROM tpch_lineitem WHERE l_shipdate = '1996-06-06';`
  - SQL总耗时：37.006s、37.269s、37.17s，平均耗时37.149s。
- 非主键查询，开启全局二级索引
  - SQL语句：`SELECT count(*) FROM tpch_lineitem WHERE l_shipdate = '1996-06-06';`
  - SQL总耗时（开启l_shipdate列的全局二级索引）：1.686s、1.651s、1.784s，平均耗时1.707s。

Spark连接表格存储数据表和多元索引

Spark连接到表格存储数据表和多元索引后，通过Spark外表查询数据时，系统会自动使用设置的多元索引进行查询。

1. 在表格存储侧创建数据表和多元索引。
   i. 创建数据表。具体操作，请参见概述。

   本示例中数据表名称为geo_table，主键列为pk1（String类型），属性列分别为val_keyword1（String类型）、val_keyword2（String类型）、val_keyword3（String类型）、val_bool（Boolean类型）、val_double（Double类型）、val_long1（Long类型）、val_long2（Long类型）、val_text（String类型）和val_geo（String类型），数据条数为208912382，数据样例如下图所示。

pk1(主键)	val_bool	val_double	val_geo	val_keyword1	val_keyword2	val_keyword3	val_long1	val_long2	val_text
000000111235...	true	5115.675982414764	77.095359344091...	99a		ddeeffhgiijjkkllnmo...	76219039	9095	jkmn srrt vxy0
00000011223...	true	5749.532177337978	6.9595843091486...	666	aa9	_ba	34193251	3304	
00000011223...	false	6544.788811853295	43.646137629808...	877	888	8__aaeegfhhiijjkkml...	65625257	5310	fgij pppq tuwx 001...
00000012445...	true	893.466367126703	39.879665239814...	dde	ussttuuzy	jlkmnnooqprrsttuu...	29781870	1923	bcef iiij kmln oppr ...
00000012445...	false	7546.537986812851	69.786751502895...	dde		ooooppqqrttsssssr...	77670653	709	ttsu yyyz 2235 677...
00000012455...	true	774.8470049349321	15.749635372889...	667	8_99ap	sttuwwxxyyyyxxxx...	55490380	447	
000000y11235...	false	8761.384128437592	29.955553294736...	dde		tttsuuv	14035455	5506	fggh mmnn qrsv yy...
000000yghjkl...	false	2943.7331796206...	73.554041223022...	011	11z1056d	kmlnoopprqssttuuv...	54210039	103	wzz2
000000z11223...	false	1677.20249416672...	3.14253294563 49...	jlk	ggghhiilm	vwwwvvxwyyzzzyy...	87747663	7729	ihhj abbd fhik mop...
000000z11223...	false	5991.067741621071	42.977362790263...	yyz	00011224	vvuwxxyy0zzzzzyy...	35861398	1460	9__a

ii. 在数据表上创建多元索引。具体操作，请参见创建及使用多元索引。

创建多元索引时，根据字段类型选择对应的多元索引Mapping。

> 说明
>
> 创建多元索引时，地理位置字段需选择字段类型为地理位置而非字符串类型。

创建多元索引后，多元索引会自动开始同步数据表中的数据，待多元索引进入增量状态时，表示多元索引完成构建。

2. 在EMR集群侧创建Spark外表。

i. 登录EMR Header服务器。

ii. 创建Spark外表同时连接多元索引。

- 参数

参数	说明
endpoint	表格存储实例访问地址，EMR集群中使用VPC地址。
access.key.id	阿里云账号的AccessKey ID。
access.key.secret	阿里云账号的AccessKey Secret。
instance.name	实例名称。
table.name	表格存储的数据表名称。
search.index.name	多元索引名称。
max.split.count	多元索引Parallel Scan的查询并发度，并发数和Spark的Split数对应。
push.down.range.long	与Long类型的列做大小（>=、>、<、<=）比较的谓词是否下推。更多信息，请参见批计算谓词下推配置。 类型为Boolean，默认值为true，表示与Long类型的列做大小比较的谓词下推。 设置为false时，表示与Long类型的列做大小比较的谓词不下推。
push.down.range.string	与String类型的列做大小（>=、>、<、<=）比较的谓词是否下推。更多信息，请参见批计算谓词下推配置。 类型为Boolean，默认值为true，表示与String类型的列做大小比较的谓词下推。 设置为false时，表示与String类型的列做大小比较的谓词不下推。

- 示例

```
DROP TABLE IF EXISTS geo_table;
CREATE TABLE geo_table (
pk1 STRING, val_keyword1 STRING, val_keyword2 STRING, val_keyword3 STRING,
val_bool BOOLEAN, val_double DOUBLE, val_long1 LONG, val_long2 LONG,
val_text STRING, val_geo STRING COMMENT "geo stored in string format"
)
USING tablestore
OPTIONS(
endpoint="https://sparksearchtest.cn-hangzhou.vpc.tablestore.aliyuncs.com",
access.key.id="",
access.key.secret="",
instance.name="sparksearchtest",
table.name="geo_table",
search.index.name="geo_table_index",
max.split.count=64,
push.down.range.long = false,
push.down.range.string = false
);
```

3. SQL查询实战。

   如下是不同查询需求的SQL查询样例，请根据实际业务组合使用SQL查询。

   - 使用多元索引全表查询
     - SQL语句：`SELECT COUNT(*) FROM geo_table;`
     - SQL耗时：测试数据208912382条，配置64个Parallel Scan并发，实际耗时165.208s，平均QPS约126.45万。

```
208912382
Time taken: 165.208 seconds, Fetched 1 row(s)
20/06/29 20:55:11 INFO [main] SparkSQLCLIDriver: Time taken: 165.208 seconds, Fetch
ed 1 row(s)
```

   - 组合条件查询
     - SQL语句：

```
 SELECT val_long1, val_long2, val_keyword1, val_double FROM geo_table WHERE
(val_long1 > 17183057 AND val_long1 < 27183057) AND (val_long2 > 1000 AND val_long2 <
5000) LIMIT 100;
```

     - SQL耗时：Spark会将Projection列和Filter下推到多元索引，实际耗时2.728s，极大加快查询效率。

```
21423964        4017    aaa     2501.9901650365096
21962236        2322    eio     2775.9021545044116
Time taken: 2.894 seconds, Fetched 100 row(s)
20/06/30 18:51:24 INFO [main] SparkSQLCLIDriver: Time taken: 2.894 second
```

   - 地理位置查询

     地理位置查询包括地理距离查询、地理长方形查询和地理多边形范围查询三种地理位置查询方式。示例中val_geo为地理位置字段名，地理坐标的格式都为"纬度,经度"。

- 地理距离查询

  语法为val_geo = '{"centerPoint":"中心点坐标", "distanceInMeter": 距离中心点的距离}'。

  SQL语句：

```
SELECT COUNT(*) FROM geo_table WHERE val_geo =
'{"centerPoint":"6.530045901643962,9.05358919674954", "distanceInMeter": 3000.0}';
```

- 地理长方形查询

  语法为val_geo = '{"topLeft":"矩形框的左上角的坐标", "bottomRight": "矩形框的右下角的坐标"}'。

  SQL语句：

```
SELECT COUNT(*) FROM geo_table WHERE val_geo =
'{"topLeft":"6.257664116603074,9.1595116589601", "bottomRight":
"6.153593333442616,9.25968497923747"}';
```

- 地理多边形范围查询

  语法为val_geo = '{"points":["坐标1", "坐标2", ... "坐标n-1", "坐标n"]}'。

  SQL语句：

```
SELECT COUNT(*) FROM geo_table WHERE val_geo = '{"points":
["6.530045901643962,9.05358919674954", "6.257664116603074,9.1595116589601",
"6.160393397574926,9.256517839929597", "6.16043846779313,9.257192872563525"]}';
```

2.3.2. 流计算

通过在E-MapReduce集群中使用Spark SQL访问表格存储。对于流计算，基于通道服务，利用CDC（数据变更捕获）技术完成Spark的mini batch流式消费和计算，同时提供了at-least-once一致性语义。

前提条件

- 已创建E-MapReduce Hadoop集群。具体操作，请参见创建集群。

  创建集群时，请确保打开**挂载公网**开关，将集群挂载到公网，用于Shell远程登录服务器。

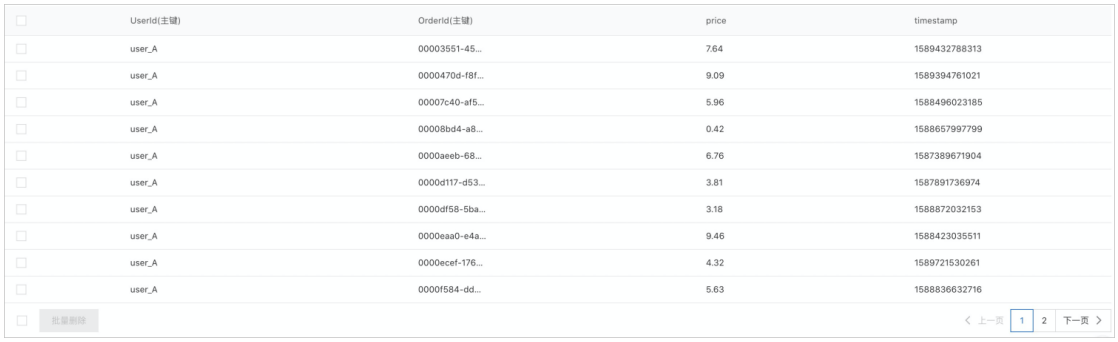
> **说明**
>
> 本文使用Shell命令演示，如果需要使用E-MapReduce的图形化页面进行数据开发。具体操作，请参见数据开发。

- 已上传emr-datasources_shaded_2.11-2.2.0-SNAPSHOT.jar包到EMR Header服务器。

快速开始

1. 在表格存储侧创建数据表和通道。
   i. 创建Source表和Sink表。具体操作，请参见概述。

   Source表名称为OrderSource，主键列分别为UserId（用户ID）和OrderId（订单ID），属性列分别为price（价格）和timestamp（订单时间），Source表如下图所示。



   ii. Sink表名称为OrderStreamSink，主键列分别为begin（开始时间）、end（结束时间），属性列分别为count（订单数）和totalPrice（订单总金额）。其中开始时间和结束时间的格式为“yyyy-MM-dd HH:mm:ss”，例如2019-11-27 14:54:00。

   iii. 在Source表上创建通道。具体操作，请参见快速入门。

   通道列表中会显示该通道的通道名、通道ID、通道类型等信息。其中通道ID用于后续的流式处理。

2. 在EMR集群侧创建Spark外表。
   i. 登录EMR Header服务器。

ii. 执行如下命令启动spark-sql命令行，用于Spark外表创建和后续的SQL实战操作。

其中Spark的标准启动参数为 `--num-executors 32 --executor-memory 2g --executor-cores 2` ，可以根据具体的集群配置进行自定义调整。<Version>表示上传JAR包的版本信息，请根据实际填写，例如2.1.0-SNAPSHOT。

```
spark-sql --jars emr-datasources_shaded_2.11-<Version>.jar --master yarn --num-exec
utors 32 --executor-memory 2g --executor-cores 2
```

iii. 创建Source外表order_source（对应表格存储的OrderSource表）。

- 参数

参数	说明
endpoint	表格存储实例访问地址，EMR集群中使用VPC地址。
access.key.id	阿里云账号的AccessKey ID。
access.key.secret	阿里云账号的AccessKey Secret。
instance.name	实例名称。
table.name	表格存储的数据表名称。
catalog	数据表的Schema定义。

- 示例

```
DROP TABLE IF EXISTS order_source;
CREATE TABLE order_source
USING tablestore
OPTIONS(
endpoint="http://vehicle-test.cn-hangzhou.vpc.tablestore.aliyuncs.com",
access.key.id="",
access.key.secret="",
instance.name="vehicle-test",
table.name="OrderSource",
catalog='{"columns": {"UserId": {"type": "string"}, "OrderId": {"type": "string"}
,"price": {"type": "double"}, "timestamp": {"type": "long"}}}'
);
```

3. 实时流计算。

实时流计算将实时统计一个窗口周期时间内的订单数和订单金额统计，并将聚合结果写回表格存储的数据表中。

i. 创建流计算的Sink外表order_stream_sink（对应表格存储的OrderStreamSink表）。

创建Sink外表与创建Source外表的参数设置中只有catalog字段有差别，其他参数设置均相同。

ii. 在Source外表order_source上创建视图。

创建视图时需要设置Source表上通道的通道ID。

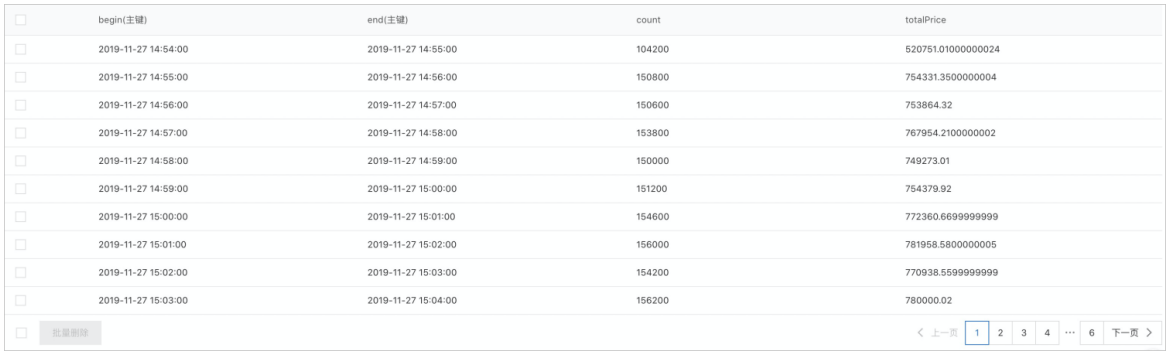
iii. 在视图上运行Stream SQL作业进行实时聚合，且将聚合结果实时写回表格存储的OrderStreamSink表。

```
//创建Sink外表order_stream_sink（对应表格存储的OrderStreamSink表）。
DROP TABLE IF EXISTS order_stream_sink;
CREATE TABLE order_stream_sink
USING tablestore
OPTIONS(
endpoint="http://vehicle-test.cn-hangzhou.vpc.tablestore.aliyuncs.com",
access.key.id="",
access.key.secret="",
instance.name="vehicle-test",
table.name="OrderStreamSink",
catalog='{"columns": {"begin": {"type": "string"},"end": {"type": "string"}, "count": {
"type": "long"}, "totalPrice": {"type": "double"}}}'
);

//在order_source表上创建视图order_source_stream_view。
CREATE SCAN order_source_stream_view ON order_source USING STREAM
OPTIONS(
tunnel.id="4987845a-1321-4d36-9f4e-73d6db63bf0f",
maxoffsetsperchannel="10000");

//在视图order_source_stream_view上运行Stream SQL作业，如下样例会按30s粒度进行订单数和订单金额的聚合。
//将聚合结果实时写回表格存储OrderStreamSink表。
CREATE STREAM job1
options(
checkpointLocation='/tmp/spark/cp/job1',
outputMode='update'
)
INSERT INTO order_stream_sink
SELECT CAST(window.start AS String) AS begin, CAST(window.end AS String) AS end, count(
*) AS count, CAST(sum(price) AS Double) AS totalPrice FROM order_source_stream_view GRO
UP BY window(to_timestamp(timestamp / 1000), "30 seconds");
```

运行Stream SQL后，可以实时得到聚合结果，聚合结果样例如下图所示，聚合结果保存在OrderStreamSink表中。



begin(主键)	end(主键)	count	totalPrice
2019-11-27 14:54:00	2019-11-27 14:55:00	104200	520751.01000000024
2019-11-27 14:55:00	2019-11-27 14:56:00	150800	754331.3500000004
2019-11-27 14:56:00	2019-11-27 14:57:00	150600	753864.32
2019-11-27 14:57:00	2019-11-27 14:58:00	153800	767954.2100000002
2019-11-27 14:58:00	2019-11-27 14:59:00	150000	749273.01
2019-11-27 14:59:00	2019-11-27 15:00:00	151200	754379.92
2019-11-27 15:00:00	2019-11-27 15:01:00	154600	772360.6699999999
2019-11-27 15:01:00	2019-11-27 15:02:00	156000	781958.5800000005
2019-11-27 15:02:00	2019-11-27 15:03:00	154200	770938.5599999999
2019-11-27 15:03:00	2019-11-27 15:04:00	156200	780000.02

批量删除 上一页 1 2 3 4 ··· 6 下一页

2.4. DataFrame编程

2.4.1. 批计算

使用Spark的DataFrame方式访问表格存储，并在本地和集群上分别进行运行调试。

前提条件

- 了解Spark访问表格存储的依赖包，并在使用时通过Maven方式引入项目中。
  - Spark相关：spark-core、spark-sql、spark-hive
  - Spark Tablestore connector：emr-tablestore-<version>.jar
  - Tablestore Java SDK：tablestore-<version>-jar-with-dependencies.jar

  其中<version>表示相应依赖包的版本号，请以实际为准。
- 已在表格存储侧创建数据表。具体操作，请参见创建数据表。
- 已获取AccessKey（包括AccessKey ID和AccessKey Secret）。具体操作，请参见获取AccessKey。

快速开始

通过项目样例了解快速使用批计算的操作。

1. 从GitHub下载项目样例的源码，具体下载路径请参见TableStoreSparkDemo。

   项目中包含完整的依赖和使用样例，具体的依赖请参见项目中的pom文件。
2. 阅读TableStoreSparkDemo项目的README文档，并安装最新版的Spark Tablestore connector和Tablestore Java SDK到本地Maven库。
3. 修改Sample代码。

   以TableStoreBatchSample为例，对此示例代码的核心代码说明如下：
   - format("tablestore")表示使用ServiceLoader方式加载Spark Tablestore connector，具体配置请参见项目中的META-INF.services。
   - instanceName、tableName、endpoint、accessKeyId、accessKeySecret分别表示表格存储的实例名称、数据表名称、实例endpoint、阿里云账号的AccessKey ID和AccessKey Secret。
   - catalog是一个JSON串，包含字段名和类型，如下示例中的数据表有salt（Long类型）、UserId（String类型）、OrderId（String类型）、price（Double类型）和timestamp（Long类型）五个字段。

     最新版本中支持使用Schema方式替换catalog的配置，请根据实际选择。

- split.size.mbs表示每个Split的切分大小，默认值为100，单位为MB，可不配置。

  此值越小产生的Split会越多，对应Spark的Task也会越多。

```
    val df = sparkSession.read
    .format("tablestore")
    .option("instance.name", instanceName)
    .option("table.name", tableName)
    .option("endpoint", endpoint)
    .option("access.key.id", accessKeyId)
    .option("access.key.secret", accessKeySecret)
    .option("split.size.mbs", 100)
    .option("catalog", dataCatalog)
    // 最新版本支持使用Schema方式替换catalog的配置。
    //.schema("salt LONG, UserId STRING, OrderId STRING, price DOUBLE, timestamp LONG
")
    .load()

  val dataCatalog: String =
    s"""
       |{"columns": {
       |    "salt": {"type":"long"},
       |    "UserId": {"type":"string"},
       |    "OrderId": {"type":"string"},
       |    "price": {"type":"double"},
       |    "timestamp": {"type":"long"}
       | }
       |}""".stripMargin
```

运行调试

根据需求修改示例代码后，可在本地或者通过Spark集群进行运行调试。以TableStoreBatchSample为例说明调试过程。

- 本地调试

  以IntelliJ IDEA为例说明。

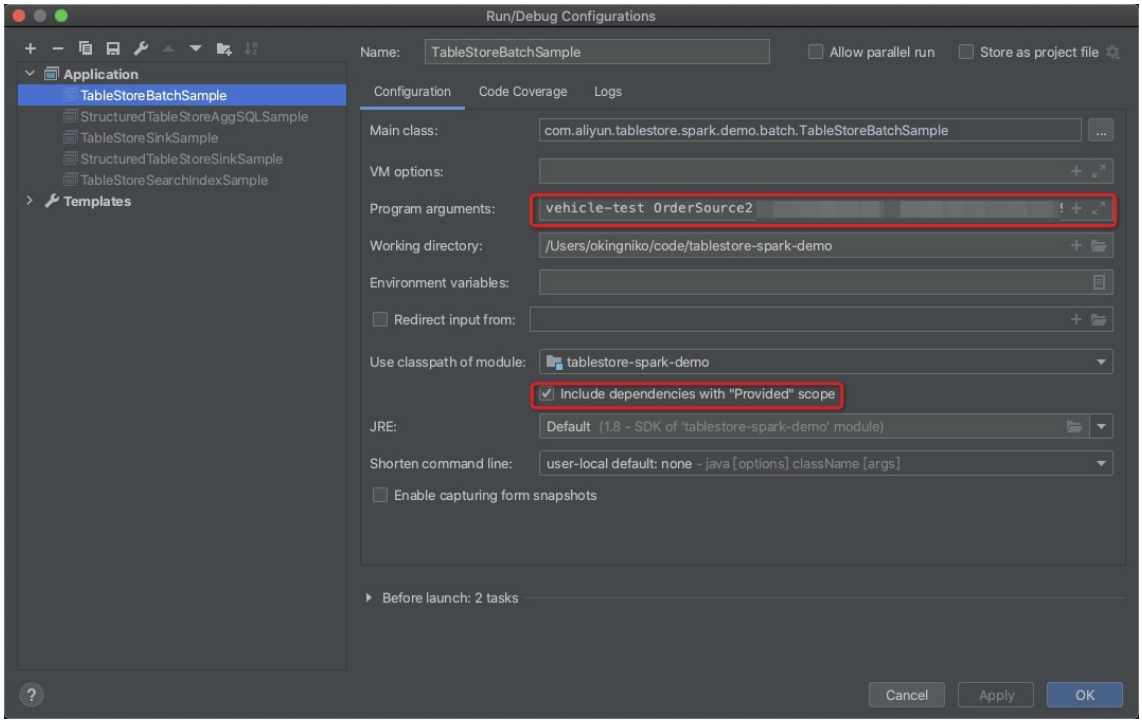
  > **说明**
  >
  > 本文测试使用的环境为Spark 2.4.3、Scala 2.11.7和Java SE Development Kit 8，如果使用中遇到问题，请联系表格存储技术支持。

  i. 在系统参数中，配置实例名称、数据表名称、实例endpoint、阿里云账号的AccessKey ID和AccessKey Secret等参数。

     您也可以自定义参数的加载方式。

  ii. 选择**include dependencies with "provided" scope**，单击**OK**。

  iii. 运行示例代码程序。



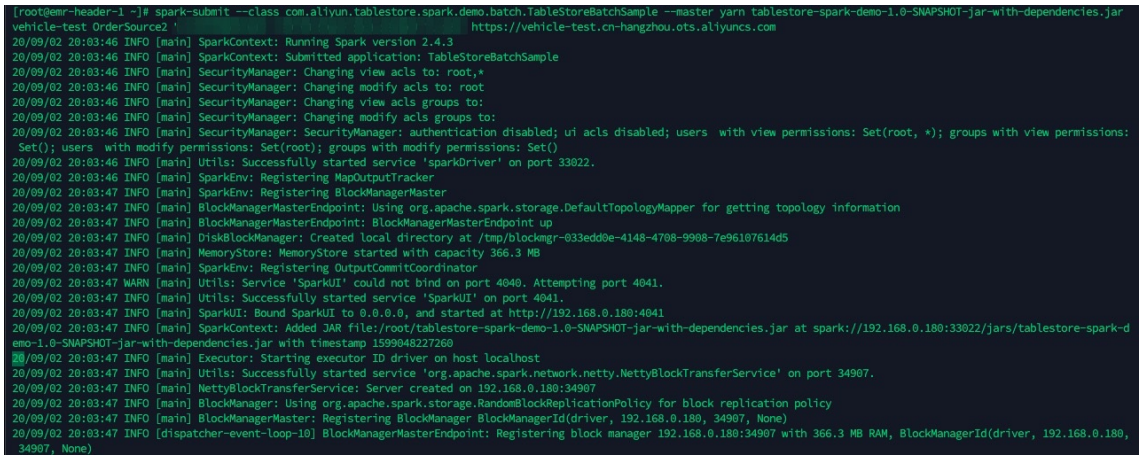
- 通过Spark集群调试

  以spark-submit方式为例说明。示例代码中的master默认为 `local[*]` ，在Spark集群上运行时可以去掉，使用spark-submit参数传入。

  i. 执行**mvn -U clean package**命令打包，包的路径为 `target/tablestore-spark-demo-1.0-SNAPSHOT-jar-with-dependencies.jar` 。

  ii. 上传包到Spark集群的Driver节点，并使用spark-submit提交任务。

```
spark-submit --class com.aliyun.tablestore.spark.demo.batch.TableStoreBatchSample --master yarn tablestore-spark-demo-1.0-SNAPSHOT-jar-with-dependencies.jar <ots-instanceName> <ots-tableName> <access-key-id> <access-key-secret> <ots-endpoint>
```



2.4.2. 流计算

使用Spark的DataFrame方式访问表格存储，并在本地和集群上分别进行运行调试。

前提条件

- 了解Spark访问表格存储的依赖包，并在使用时通过Maven方式引入项目中。
  - Spark相关：spark-core、spark-sql、spark-hive
  - Spark Tablestore connector：emr-tablestore-<version>.jar
  - Tablestore Java SDK：tablestore-<version>-jar-with-dependencies.jar

  其中<version>表示相应依赖包的版本号，请以实际为准。
- 已在表格存储侧创建Source表和在Source表上创建通道。具体操作，请参见创建数据通道。
- 已获取AccessKey（包括AccessKey ID和AccessKey Secret）。具体操作，请参见获取AccessKey。

快速开始

通过项目样例了解快速使用流计算的操作。

1. 从GitHub下载项目样例的源码，具体下载路径请参见TableStoreSparkDemo。

   项目中包含完整的依赖和使用样例，具体的依赖请参见项目中的pom文件。
2. 阅读TableStoreSparkDemo项目的README文档，并安装最新版的Spark Tablestore connector和Tablestore Java SDK到本地Maven库。
3. 修改Sample代码。

   以StructuredTableStoreAggSQLSample为例，对此示例代码的核心代码说明如下：
   - format("tablestore")表示使用ServiceLoader方式加载Spark Tablestore connector，具体配置请参见项目中的META-INF.services。
   - instanceName、tableName、tunnel.id、endpoint、accessKeyId、accessKeySecret分别表示表格存储的实例名称、数据表名称、通道ID、实例endpoint、阿里云账号的AccessKey ID和AccessKey Secret。
   - catalog是一个JSON串，包含字段名和类型，如下示例中的数据表有UserId（STRING类型）、OrderId（STRING类型）、price（DOUBLE类型）和timestamp（LONG类型）四个字段。
   - maxoffsetsperchannel表示每一个mini-batch中每一个channel（分区）最多读取的数据量，默认值为10000。

```
  val ordersDF = sparkSession.readStream
    .format("tablestore")
    .option("instance.name", instanceName)
    .option("table.name", tableName)
    .option("tunnel.id", tunnelId)
    .option("endpoint", endpoint)
    .option("access.key.id", accessKeyId)
    .option("access.key.secret", accessKeySecret)
    .option("maxoffsetsperchannel", maxOffsetsPerChannel) //默认值为10000。
    .option("catalog", dataCatalog)
    .load()
    .createTempView("order_source_stream_view")

val dataCatalog: String =
  s"""
     |{"columns": {
     |    "UserId": {"type":"string"},
     |    "OrderId": {"type":"string"},
     |    "price": {"type":"double"},
     |    "timestamp": {"type":"long"}
     | }
     |}""".stripMargin
```

运行调试

根据需求修改示例代码后，可在本地或者通过Spark集群进行运行调试。以StructuredTableStoreAggSQLSample为例说明调试过程。

- 本地调试

  以IntellIj IDEA为例说明。

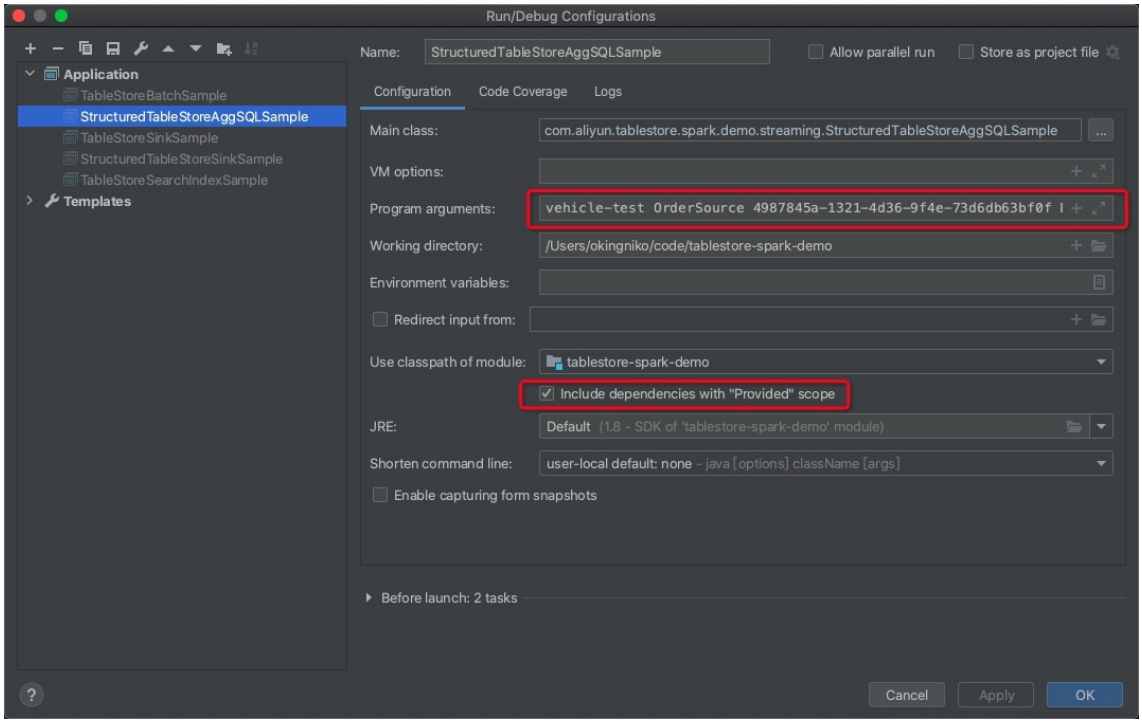
  > **说明**
  >
  > 本文测试使用的环境为Spark 2.4.3、Scala 2.11.7和Java SE Development Kit 8，如果使用中遇到问题，请联系表格存储技术支持。

  i. 在系统参数中，配置实例名称、数据表名称、实例endpoint、阿里云账号的AccessKey ID和AccessKey Secret等参数。

     您也可以自定义参数的加载方式。

  ii. 选择**include dependencies with "provided" scope**，单击**OK**。

  iii. 运行示例代码程序。



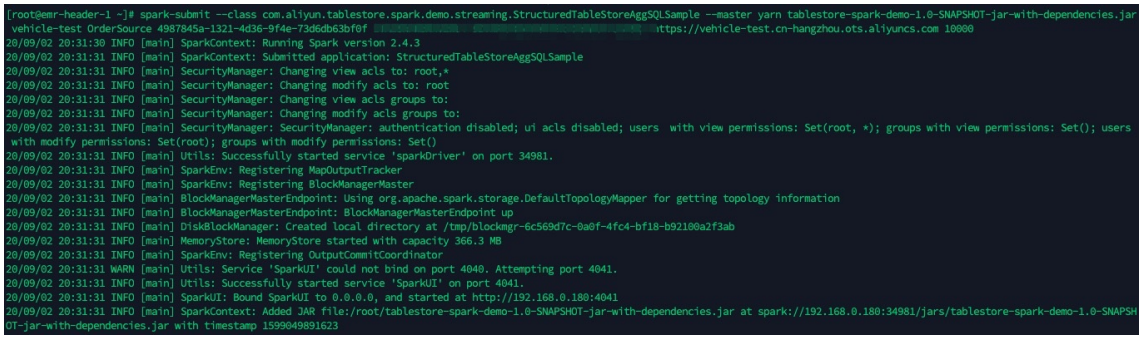
- 通过Spark集群调试

  以spark-submit方式为例说明。示例代码中的master默认为 `local[*]` ，在Spark集群上运行时可以去掉，使用spark-submit参数传入。

  i. 执行**mvn -U clean package**命令打包，包的路径为 `target/tablestore-spark-demo-1.0-SNAPSHOT-jar-with-dependencies.jar` 。

  ii. 上传包到Spark集群的Driver节点，并使用spark-submit提交任务。

```
spark-submit --class com.aliyun.tablestore.spark.demo.streaming.StructuredTableStoreAggSQLSample --master yarn tablestore-spark-demo-1.0-SNAPSHOT-jar-with-dependencies.jar <ots-instanceName> <ots-tableName> <ots-tunnelId> <access-key-id> <access-key-secret> <ots-endpoint> <max-offsets-per-channel>
```



2.5. 细则剖析

2.5.1. 批计算谓词下推配置

批计算中的多元索引查询方式可以自定义谓词下推配置。目前只能设置与Long、String类型的列做大小比较的谓词是否下推。

背景信息

谓词下推适用于当多元索引中多字段过滤的中间结果数据量较大，则中间结果的合并较为耗时的场景。此时可以将某些字段的过滤从存储层（表格存储）提到计算层（Spark）处理，提高查询效率。

例如 `select * from table where a = 10 and  b < 999999999;` ，如果a = 10返回的结果只有1000条，b < 999999999返回的结果有一亿条，则在存储层将1000条结果与一亿条结果做合并比较耗时，此时把b < 999999999提到计算层，Spark只需要对存储层返回的1000条数据作过滤，大大降低了存储层的压力。

谓词支持

Spark谓词支持情况请参见下表。

Spark	是否支持	SQL语句举例
And	是	select * from table where a > 1 and b < 0;
Or	是	select * from table where a > 1 or b < 0;
Not	是	select * from table where a != 1;
EqualTo	是	select * from table where a = 1;
Not+EqualTo	是	select * from table where a != 1;
IsNull	是	select * from table where a is null; 返回table数据表中没有a列的行。
In	是	select * from table where a in {1,2,3}; 默认最大限制为1024。
LessThan	是	select * from table where a < 10; 当SQL语句中使用该谓词的列的数据类型为Long或者String时，可以自定义谓词下推配置。
LessThanOrEqual	是	select * from table where a <=10; 当SQL语句中使用该谓词的列的数据类型为Long或者String时，可以自定义谓词下推配置。

Spark	是否支持	SQL语句举例
GreaterThan	是	select * from table where a > 10; 当SQL语句中使用该谓词的列的数据类型为Long或者String时，可以自定义谓词下推配置。
GreaterThanOrEqual	是	select * from table where a >= 10; 当SQL语句中使用该谓词的列的数据类型为Long或者String时，可以自定义谓词下推配置。
StringStartsWith	是	select * from table where a like "tablestore%"; 返回table数据表中a列以tablestore为前缀（prefix）的行。
地理坐标（String JSON）中心距离	是	select * from table where val_geo = '{"centerPoint":"3,0", "distanceInMeter": 100000}'; 由中心点和距离决定的地理圆圈范围。
地理坐标（String JSON）长方形	是	select * from table where geo = '{"topLeft":"8,0", "bottomRight": "0,10"}'; 由左上角和右下角决定的地理长方形范围。
地理坐标（String JSON）多边形	是	select * from table where geo = '{"points":["5,0", "5,1", "6,1", "6,10"]}'; 由多个点组成的地理多边形范围。

谓词下推配置

谓词下推配置的参数需要在创建Spark外表时配置。谓词下推规则如下：

- 当过滤条件中的逻辑谓词只存在AND和NOT时，可以自定义谓词是否下推。
- 当过滤条件中的逻辑谓词存在OR时，谓词全部下推，自定义的谓词下推配置（例如E-MapReduce SQL方式的参数配置push.down.range.long=false和push.down.range.string=false）不会生效。

不同的逻辑谓词和下推配置组合使用时，SQL语句举例及预期效果请参见下表。

逻辑谓词	下推配置	SQL语句举例	预期效果
全为AND	• push.down.range.long=true • push.down.range.string=true	select * from table where val_long1 > 1000 and val_long1 is null and name like 'table%' and pk in {12341,213432};	SQL正常。 谓词全部下推。

逻辑谓词	下推配置	SQL语句举例	预期效果
全为AND	• push.down.range.long=false • push.down.range.string=true	select * from table where val_long1 > 1 and name like 'table%';	与Long类型做大小比较的谓词不会下推。 该谓词由Spark计算层进行过滤，Spark计算层获取name like 'table%'的数据，val_long1 > 1由业务代码进行过滤。
全为AND	• push.down.range.long=true • push.down.range.string=false	select * from table where val_string1 > 'string1' and name like 'table%';	与String类型做大小比较的谓词都不会下推。 该谓词交由Spark计算层进行过滤，Spark计算层获取name like 'table%'的数据，val_string1 > 'string1'由业务代码进行过滤。
全为AND	存在地理位置类型列	select * from table where val_geo = '{"centerPoint":"3,0", "distanceInMeter": 100000}' and val_long1 = 37691900 and val_long2 > 2134234;	SQL正常。 val_long2 > 2134234能否由Spark计算层过滤取决于push.down.range.long的配置。
存在OR	• push.down.range.long=true • push.down.range.string=true	select * from table where val_long1 > 1000 or val_long1 is null or name like 'table%' and pk in {12341,213432};	SQL正常。 谓词全部下推。
存在OR	存在地理位置类型列	select * from table where val_geo = '{"centerPoint":"3,0", "distanceInMeter": 100000}' or val_long1 = 37691900;	SQL正常。 谓词全部下推。
存在OR	• push.down.range.long=false • push.down.range.string=true • SQL语句中存在rangeLong的过滤字段	select * from table where val_long1 > 1 or name like 'table%';	此时自定义谓词配置不生效，谓词全部下推。

逻辑谓词	下推配置	SQL语句举例	预期效果
存在OR	• push.down.range.long=true • push.down.range.string=false • SQL语句中存在rangeString的过滤字段	select * from table where val_string1 > 'string1' or name like 'table%';	此时自定义谓词配置不生效，谓词全部下推。

2.5.2. 流计算实现细节

了解对接Structured Streaming的微批模式的过程，以及表格存储对接Spark Structured Streaming的详细接入流程。

背景信息

在对接Spark Structured Streaming的微批模式时，以Spark DataSource v1接口为例说明过程。

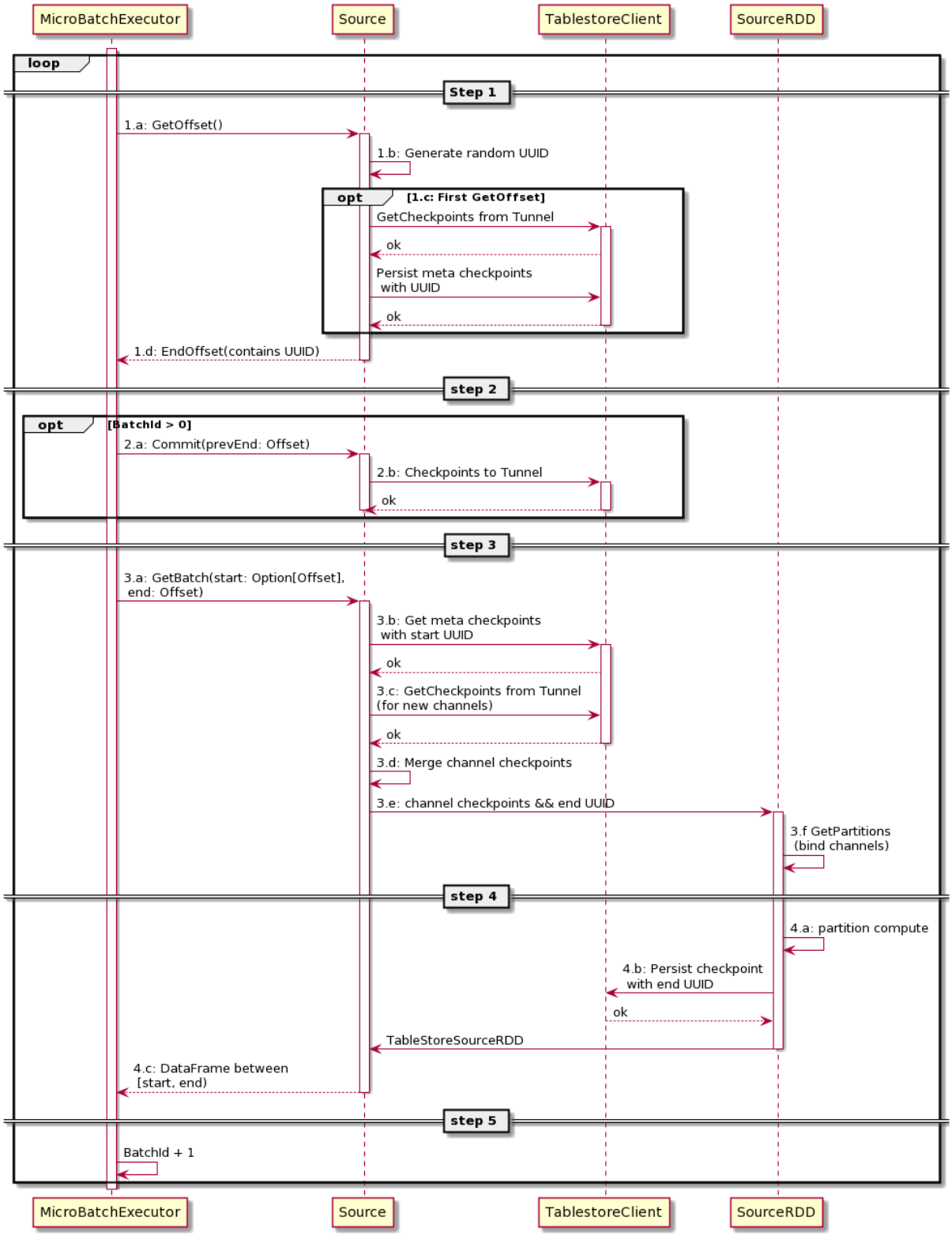
1. 调用GetOffset方法获取当前批次可以读取到的最大Offset（EndOffset）。
2. 调用GetBatch方法获取当前批次的StartOffset（上一个批次的EndOffset）到EndOffset间的数据并进行转换。
3. 执行自定义的Spark计算逻辑。

在此种微批模式对接中，上游数据库的流式接口需要提供灵活且准确的Seek功能，能够实时获取到每个分区的起始游标或结束游标等，从而进行Spark微批模式中Offset的预估。

在分布式NoSQL数据库中，对于基于数据捕获变更（CDC）技术的流式接口，该接口是一种持续流的模式，通常不具备灵活的Seek接口，对接Structured Streaming的微批接口有较大的难度。如果在GetOffset阶段进行提前的数据拉取，可以获取到预期的EndOffset，但是需要提前进行一次额外的RDD的计算并将其进行持久化缓存，大大降低了Source的性能。

接入流程

下图为表格存储对接Structured Streaming方案的UML时序图。



图中MicroBatchExecutor为Spark的微批处理框架，Source为Structured Streaming抽象的接口类，SourceRDD为Spark的RDD抽象类，TablestoreClient为表格存储的客户端。实线表示详细操作，虚线表示请求成功返回。

从整体上看，时序流程是GetOffset、 Commit和GetBatch三个步骤的循环执行，分别对应于Spark Streaming的一个Batch执行。

详细接入流程步骤说明如下：

1. 调用GetOffset方法获取当前批次可以读取到的最大Offset（EndOffset）。
   i. MicroBatchExecutor调用Source中GetOffset方法，获取当前批次的最大可达的Offset，即EndOffset。
   ii. Source中生成一个随机的UUID字符串，UUID会与Offset一一对应。
   iii. （可选）获取表格存储的通道信息。

   > **说明**
   > 只有首次调用Source中GetOffset方法时，才执行此步骤；否则系统会跳过此步骤。

      a. 从表格存储的Tunnel服务端获取当前所有Channel（分区）的消费位点（Checkpoint）。
      b. 将获取到所有Channel的消费位点持久化到Meta表中，建立UUID和checkpoints的映射关系。
   iv. 将UUID包装成Offset（Offset与UUID可以互相转换），并返回给MicroBatchExecutor作为当前批次的EndOffset。
2. （可选）将checkpoints持久化到Tunnel服务端。

   > **说明**
   > 只有当前批次的BatchId大于0时，才执行此步骤；否则系统会跳过此步骤

   i. MicroBatchExecutor调用Source的Commit逻辑。
   ii. 将上一个批次的EndOffset（当前批次的StartOffset）对应的checkpoints持久化到Tunnel服务端。

   > **说明**
   > 正常情况下Commit逻辑无需处理额外内容，此操作的目的如下：
   > - 持久化到Tunnel服务端后，可以实时显示消费进度。
   > - Tunnel为了实现数据保序有父子分区关系，需要由客户端将父分区已消费结束checkpoint返回服务端，子分区才能加载。

3. 调用GetBatch方法获取当前批次的StartOffset（上一个批次的EndOffset）到EndOffset间的数据并进行转换。
   i. MicroBatchExecutor根据当前批次的StartOffset和GetOffset返回的EndOffset调用Source中的GetBatch操作，以期获得当前批次的数据，供计算逻辑使用。
   ii. 根据StartOffset对应的UUID从Meta表中获取Tunnel对应的Channel列表的实时消费位点。
   iii. 从Tunnel服务端定时获取Channel的消费位点。

   由于表的分区变化可能会存在新的分区（例如子分区），所以需要定时获取Channel的消费位点。
   iv. 合并从Meta表中获取的实时消费位点和从Tunnel服务端定时获取消费位点checkpoints，得到到当前所有Channel的消费位点。
   v. 根据最新的checkpoints和EndOffset对应的UUID等信息来构建SourceRDD。

   RDD是Spark中最基本的数据抽象，它代表一个不可变、可分区、元素可并行计算的集合。

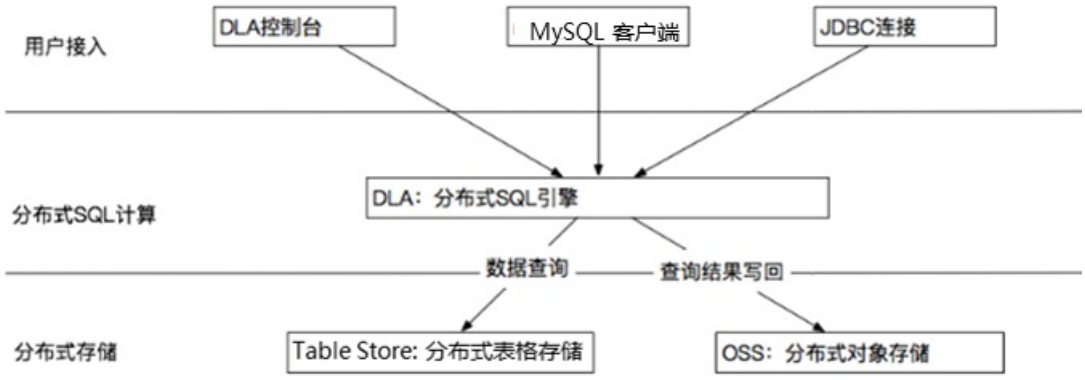
vi. 在SourceRDD中将通道的channel和RDD的partition进行绑定，因此每个Channel都会在Spark的执行节点上分布式的进行数据并行转换和处理。

4. 执行自定义的Spark计算逻辑。
   i. 执行每个RDD partition上的计算逻辑，此处会进行channel上的数据读取，并更新内存中的每个channel的消费位点。
   ii. 每个RDD partition在本批次执行完成后（例如读到指定条数或无新数据），将channel对应的最新消费位点持久化到Meta表对应UUID的行中，即填充EndOffset对应UUID的checkpoints，每个批次结束后，新的批次的StartOffset对应的checkpoints都能够在Meta表中查询到。
   iii. 待所有RDD partition的计算逻辑完成后，返回当前批次对应Offset范围内的数据给MicroBatchExecutor。
5. 当前批次的所有逻辑计算完成后，递增BatchId，从步骤1重新开始，循环执行。

3.Data Lake Analytics

3.1. 背景信息

Tablestore中接入Data Lake Analytics（简称 DLA）服务的方式，为您提供一种快速的OLAP（On-Line Analytical Processing）解决方案，可通过DLA访问表格存储中的数据。DLA是阿里云一款通用的SQL查询引擎，使用通用的SQL语言（兼容MySQL 5.7绝大部分查询语法）可在Tablestore中进行灵活的数据分析。



如架构图所示，OLAP查询架构涉及阿里云DLA、Tablestore和OSS三款产品。

- DLA：负责分布式SQL查询计算。在实际运行过程中将SQL查询请求进行任务拆解，产生若干可并行化的子任务，提升数据计算和查询能力。
- Tablestore：数据存储层，用于接收DLA的各类子查询任务。如果您在Tablestore中已经有存量数据，可以直接在DLA上建立映射视图，从而体验SQL计算的便捷服务。
- OSS：分布式对象存储系统，主要用于保存查询结果。

如果您要在Tablestore中体验分布式SQL计算，必须开通Tablestore、DLA和OSS服务。

> **说明**
> - 开通OSS服务的主要原因是DLA默认将查询结果集数据写入OSS，因此需要引入一个额外的存储依赖。您仅需开通OSS服务，无需预先创建OSS存储空间。
> - 目前开服公测的区域是华东2（上海），对应的实例是该Region内所有的容量型实例。在开通DLA服务时，需要先填写公测申请。详情请参考准备工作文档。

3.2. 准备工作

如果您要在Tablestore中体验分布式SQL计算，需开通Tablestore、Data Lake Analytics和OSS服务。本文主要为您介绍如何开通这些服务。

> **说明** 完成Tablestore、Data Lake Analytics和OSS服务接入后，实际查询将会产生相应的费用。在实际查询过程中如果您的账号欠费，则查询失败。

开通Tablestore服务

如果您已经开通Tablestore服务，并且已创建实例和数据表，则忽略该步骤。

如果您首次使用Tablestore，请开通Tablestore并创建实例和数据表。具体操作，请参见通过控制台使用或者通过命令行工具使用。

开通OSS服务

如果您首次使用OSS，需开通OSS服务。具体操作，请参见开通OSS服务。

开通Data Lake Analytics服务

如果您首次使地域用Data Lake Analytics，需先开通Data Lake Analytics服务。开通DLA服务后，按照如下步骤申请DLA账号：

1. 登录DLA管理控制台，选择开通对应地域的DLA服务实例，例如华东2（上海）地域，然后单击**初始化服务**。

   > **说明** 不同的地域对应不同的账号，且不同地域之间的DLA账号不能混用。

2. 在云产品开通页面，单击**立即开通**。

   > **说明** 账号创建完成之后会收到相关邮件（邮箱为阿里云的注册邮箱），内含该region的DLA账号和密码，注意查收。

3. 选择地域，单击右上角的**Tablestore授权**。
4. 在**云资源访问授权**页面，单击**同意授权**，授权 DLA 访问 Tablestore 中的用户实例数据，如下图所示：

3.3. 使用DLA服务

开通服务后，可通过控制台、MySQL Client以及JDBC这三种方式接入DLA服务并进行SQL查询。

Tablestore和DLA元信息映射逻辑

- 库和表等概念映射

Tablestore	DLA
实例（instance）	schema或database
表（table）	table
主键列（pk）	column，isPrimaryKey=true，isNullable=false
非主键列（column）	column，isPrimaryKey=false，isNullable=<用户的DDL定义>

- 字段的映射关系

Tablestore	DLA
INTEGER（8 Bytes）	Bigint（8 Bytes）
STRING	Varchar
BINARY	Varbinary（目前主键中不支持）
DOUBLE	Double
BOOLEAN	Byte

控制台访问DLA

控制台访问DLA步骤如下：

1. 使用邮件中随附的该地域的用户名和密码登录数据库。
2. 为Tablestore中的实例表格数据建立映射。

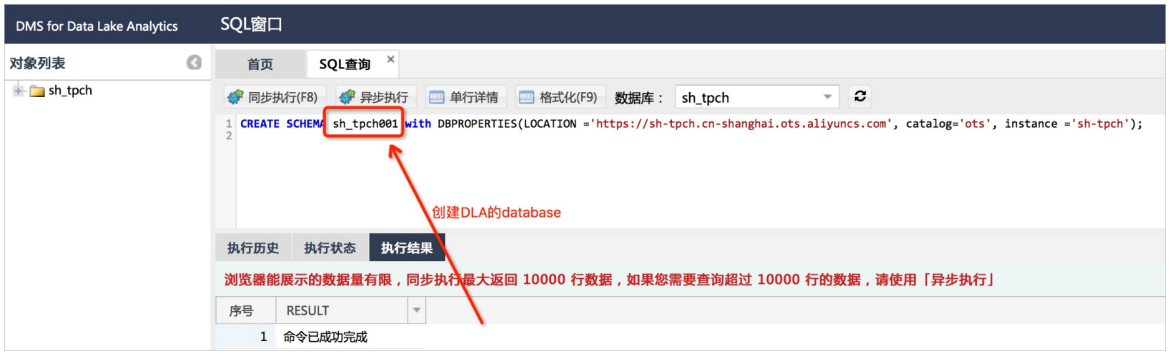
假设您在上海地域已创建一个名为sh_tpch的实例，该实例包含表格test001，表格内包含2行测试数据。该实例建立映射的步骤如下：

1. 将Tablestore的实例映射成DLA的一个DataBase实例。建立DLA的Database映射前，首先需要在Tablestore中创建实例，如创建一个名为sh-tpch的实例，对应的endpoint为sh-tpch.cn-shanghai.ots.aliyuncs.com。

   完成测试实例创建后，执行下列语句建立Database映射：

   ```
   CREATE SCHEMA sh_tpch001 with DBPROPERTIES(LOCATION ='https://sh-tpch.cn-shanghai.ots.aliyuncs.com', catalog='ots', instance ='sh-tpch');
   ```

   > **说明** 使用MySQL Client时，可以使用create database或create schema语句创建database映射。但是控制台目前只支持create schema语句创建database映射。



2. 在tp_tpch001的Database下建立表格的映射。在建立DLA的表格映射前，首先需要在Tablestore中创建数据表。

   数据表创建完成后，执行下列语句建立表格映射：

   ```
   CREATE EXTERNAL TABLE test001 (pk0 int NOT NULL , primary key(pk0));
   ```

   > **说明** 建立DLA映射表时，指定的Primary Key必须与Tablestore表格定义Primary Key列表一致。Primary Key用于唯一确定一行的数据，一旦映射表的Primary Key列表与Tablestore表格的PK不一致，可能导致SQL查询结果出现非预期错误。

   例如，您的Tablestore实例sh_tpch中包含test001表格，其中只有一列pk0。使用show命令可查看该表已创建成功。

3. 使用select语句执行SQL查询。
   - 查出所有数据：

     ```
     select * from test001;
     ```

- 执行count统计：

```
select count(*) from test001;
```

- 执行sum统计：

```
select sum(pk0) from test001;
```

执行SQL查询时，可以选择同步执行结果，返回满足条件的前 10000 条记录；如要获取大结果集数据，请选择异步执行，并使用show query_id的方式异步获取结果：

```
show query_task where id = '59a05af7_1531893489231';
```

其他执行语句，请查看如下说明文档：

- CREATE SCHEMA
- CREATE TABLE
- SELECT
- SHOW
- DROP TABLE
- DROP SCHEMA

MySQL Client访问DLA

您可以使用标准的MySQL Client快速接入DLA的数据实例，其连接语句为：

```
mysql -h service.cn-shanghai.datalakeanalytics.aliyuncs.com -P 10000 -u <username> -p <pass
word> -c -A
```

> **说明** 其他操作语句与控制台访问一致。

JDBC访问DLA

您还可以使用标准的Java API访问DLA，其连接语句为：

```
jdbc:mysql://service.cn-shanghai.datalakeanalytics.aliyuncs.com:10000/
```

> **说明** 其他操作语句与控制台访问一致。

4.Hive/HadoopMR

4.1. 环境准备

本文为您介绍使用Hive/HadoopMR访问表格存储中的表前的环境准备。

使用Hive/HadoopMR来访问表格存储中的表

通过表格存储及E-MapReduce官方团队发布的依赖包，可以直接使用Hive及HadoopMR来访问表格存储中的数据并进行数据分析。

安装JDK-7+

1. 下载并安装JDK-7+安装包。
   - Linux/macOS系统：使用系统自带的包管理器安装
   - Windows系统：具体下载路径请参见JDK安装包
2. 按照以下示例进行安装检查。

```
$ java -version
java version "1.8.0_77"
Java(TM) SE Runtime Environment (build 1.8.0_77-b03)
Java HotSpot(TM) 64-Bit Server VM (build 25.77-b03, mixed mode)
```

安装并启动Hadoop环境

1. 下载2.6.0版本以上的Hadoop安装包，具体下载路径请参见Hadoop安装包。
2. 解压并安装，根据实际集群情况安装Hadoop服务。
3. 按照如下示例启动Hadoop环境。

```
$ bin/start-all.sh
# 检查服务是否成功启动。
$ jps
24017 NameNode
24835 Jps
24131 DataNode
24438 ResourceManager
5114 HMaster
24287 SecondaryNameNode
24527 NodeManager
```

4. 在*/etc/profile*中添加Hadoop路径，并执行**source /etc/profile**命令使配置生效。

```
export HADOOP_HOME=/data/hadoop/hadoop-2.6.0
export PATH=$PATH:$HADOOP_HOME/bin
```

下载及安装Hive环境

1. 下载类型为bin.tar.gz的Hive安装包，具体下载路径请参见Hive安装包。
2. 按照如下示例解压安装包。

```
$ mkdir /home/admin/hive-2.1.0
$ tar -zxvf apache-hive-2.1.0-bin.tar.gz -C /home/admin/
$ mv /home/admin/apache-hive-2.1.0-bin /home/admin/hive-2.1.0/
```

3. 按照如下示例初始化schema。

```
# 进入指定的目录。
$ cd /home/admin/hive-2.1.0/
# 初始化，如果是mysql则derby可以直接替换成mysql。
# 如果执行出错可以删除rm -rf metastore_db/之后重新执行。
$ ./bin/schematool -initSchema -dbType derby
```

4. 按照如下示例启动Hive环境。

```
$ ./bin/hive
# 检查服务是否成功启动。
hive> show databases;
OK
default
Time taken: 0.207 seconds, Fetched: 1 row(s)
```

下载表格存储的Java SDK

1. 在Maven库中下载4.1.0版本以上的Java SDK相关依赖包，具体下载路径请参见Java SDK历史迭代版本。
   Java SDK相关依赖包会随最新的Java SDK发布，请下载最新的相关依赖包。
2. 按照如下示例将SDK拷贝到Hive目录下。

```
$ mv tablestore-4.1.0-jar-with-dependencies.jar /home/admin/hive-2.1.0/
```

下载阿里云EMR SDK

下载EMR SDK依赖包，具体下载路径请参见EMR SDK依赖包。

4.2. 使用教程

本文主要为您介绍如何使用Hive或者HadoopMR访问表格存储中的表。

数据准备

在表格存储中准备一张数据表pet，name是唯一的一列主键，数据示例请参见下表。

> **说明** 表中空白部分无需写入，因为表格存储是schema-free的存储结构，没有值也无需写入NULL。

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	

name	owner	species	sex	birth	death
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	
Puffball	Diane	hamster	f	1999-03-30	

前提条件

已准备好Hadoop、Hive、JDK环境以及表格存储Java SDK和EMR SDK依赖包。更多信息，请参见准备工作。

Hive访问示例

1. HADOOP_HOME及HADOOP_CLASSPATH可以添加到*/etc/profile*中，示例如下：

```
export HADOOP_HOME=${您的Hadoop安装目录}
export HADOOP_CLASSPATH=emr-tablestore-1.4.2.jar:tablestore-4.3.1-jar-with-dependencies
.jar:joda-time-2.9.4.jar
```

2. 执行**bin/hive**命令进入hive后，创建外表。示例如下：

```
CREATE EXTERNAL TABLE pet
  (name STRING, owner STRING, species STRING, sex STRING, birth STRING, death STRING)
  STORED BY 'com.aliyun.openservices.tablestore.hive.TableStoreStorageHandler'
  WITH SERDEPROPERTIES(
    "tablestore.columns.mapping"="name,owner,species,sex,birth,death")
  TBLPROPERTIES (
    "tablestore.endpoint"="YourEndpoint",
    "tablestore.access_key_id"="YourAccessKeyId",
    "tablestore.access_key_secret"="YourAccessKeySecret",
    "tablestore.table.name"="pet");
```

具体配置项说明请参见下表。

配置项	说明

配置项	说明
WITH SERDEPROPERTIES	字段映射配置，包括tablestore.columns.mapping选项配置。 在默认的情况下，外表的字段名即为表格存储上表的列名（主键列名或属性列名）。但有时外表的字段名和表上列名并不一致（例如处理大小写或字符集相关的问题），此时需要指定tablestore.columns.mapping。该参数为一个英文逗号分隔的字符串，每个分隔之间不能添加空格，每一项都是表中列名，顺序与外表字段保持一致。 **说明** 表格存储的列名支持空白字符，所以空白也会被认为是表中列名的一部分。
TBLPROPERTIES	表的属性配置。包括如下选项： ○ tablestore.endpoint（必选）：访问表格存储的服务地址，您可以在表格存储控制台上查看实例的Endpoint信息。关于服务地址的更多信息，请参见服务地址。 ○ tablestore.instance（可选）：表格存储的实例名称。如果不填写，则为tablestore.endpoint的第一段。关于实例的更多信息，请参见实例。 ○ tablestore.access_key_id（必选）：阿里云账号或者RAM用户的AccessKey ID。更多信息，请参见获取AccessKey。 当要使用STS服务临时访问资源时，请设置此参数为临时访问凭证的AccessKey ID。 ○ tablestore.access_key_secret（必选）：阿里云账号或者RAM用户的AccessKey Secret。更多信息，请参见获取AccessKey。 当要使用STS服务临时访问资源时，请设置此参数为临时访问凭证的AccessKey Secret。 ○ tablestore.sts_token（可选）：临时访问凭证的安全令牌。当要使用STS服务临时访问资源时，才需要设置此参数。更多信息，请参见配置临时用户权限。 ○ tablestore.table.name（必选）：表格存储中对应的表名。

3. 查询表中数据。
   - 执行SELECT * FROM pet;命令查询表中所有行数据。

     返回结果示例如下：

```
Bowser  Diane   dog     m       1979-08-31      1995-07-29
Buffy   Harold  dog     f       1989-05-13      NULL
Chirpy  Gwen    bird    f       1998-09-11      NULL
Claws   Gwen    cat     m       1994-03-17      NULL
Fang    Benny   dog     m       1990-08-27      NULL
Fluffy  Harold  cat     f       1993-02-04      NULL
Puffball        Diane   hamster f       1999-03-30      NULL
Slim    Benny   snake   m       1996-04-29      NULL
Whistler        Gwen    bird    NULL    1997-12-09      NULL
Time taken: 5.045 seconds, Fetched 9 row(s)
```

- 执行SELECT * FROM pet WHERE birth > "1995-01-01";命令查询表中birth列值大于1995-01-01的行数据。

  返回结果示例如下：

```
Chirpy  Gwen    bird    f       1998-09-11      NULL
Puffball        Diane   hamster f       1999-03-30      NULL
Slim    Benny   snake   m       1996-04-29      NULL
Whistler        Gwen    bird    NULL    1997-12-09      NULL
Time taken: 1.41 seconds, Fetched 4 row(s)
```

返回结果示例如下：

```
Bowser  Diane   dog     m       1979-08-31      1995-07-29
Buffy   Harold  dog     f       1989-05-13      NULL
Chirpy  Gwen    bird    f       1998-09-11      NULL
Claws   Gwen    cat     m       1994-03-17      NULL
Fang    Benny   dog     m       1990-08-27      NULL
Fluffy  Harold  cat     f       1993-02-04      NULL
Puffball        Diane   hamster f       1999-03-30      NULL
Slim    Benny   snake   m       1996-04-29      NULL
Whistler        Gwen    bird    NULL    1997-12-09      NULL
Time taken: 5.045 seconds, Fetched 9 row(s)
hive> SELECT * FROM pet WHERE birth > "1995-01-01";
Chirpy  Gwen    bird    f       1998-09-11      NULL
Puffball        Diane   hamster f       1999-03-30      NULL
Slim    Benny   snake   m       1996-04-29      NULL
Whistler        Gwen    bird    NULL    1997-12-09      NULL
Time taken: 1.41 seconds, Fetched 4 row(s)
```

HadoopMR访问示例

以下示例介绍如何使用HadoopMR程序统计数据表pet的行数。

- 构建Mappers和Reducers

```
public class RowCounter {
public static class RowCounterMapper
extends Mapper<PrimaryKeyWritable, RowWritable, Text, LongWritable> {
    private final static Text agg = new Text("TOTAL");
    private final static LongWritable one = new LongWritable(1);
    @Override
    public void map(
        PrimaryKeyWritable key, RowWritable value, Context context)
        throws IOException, InterruptedException {
        context.write(agg, one);
    }
}
public static class IntSumReducer
extends Reducer<Text,LongWritable,Text,LongWritable> {
    @Override
    public void reduce(
        Text key, Iterable<LongWritable> values, Context context)
        throws IOException, InterruptedException {
        long sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }
        context.write(key, new LongWritable(sum));
    }
}
}
```

数据源每从表格存储上读出一行，都会调用一次mapper的map()。PrimaryKeyWritable和RowWritable两个参数分别对应这行的主键以及该行的内容。您可以通过调用PrimaryKeyWritable.getPrimaryKey()和RowWritable.getRow()取得表格存储Java SDK定义的主键对象及行对象。

- 配置表格存储作为mapper的数据源。

```
private static RangeRowQueryCriteria fetchCriteria() {
    RangeRowQueryCriteria res = new RangeRowQueryCriteria("YourTableName");
    res.setMaxVersions(1);
    List<PrimaryKeyColumn> lower = new ArrayList<PrimaryKeyColumn>();
    List<PrimaryKeyColumn> upper = new ArrayList<PrimaryKeyColumn>();
    lower.add(new PrimaryKeyColumn("YourPkeyName", PrimaryKeyValue.INF_MIN));
    upper.add(new PrimaryKeyColumn("YourPkeyName", PrimaryKeyValue.INF_MAX));
    res.setInclusiveStartPrimaryKey(new PrimaryKey(lower));
    res.setExclusiveEndPrimaryKey(new PrimaryKey(upper));
    return res;
}
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "row count");
    job.addFileToClassPath(new Path("hadoop-connector.jar"));
    job.setJarByClass(RowCounter.class);
    job.setMapperClass(RowCounterMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(LongWritable.class);
    job.setInputFormatClass(TableStoreInputFormat.class);
    TableStoreInputFormat.setEndpoint(job, "https://YourInstance.Region.ots.aliyuncs.com/
");
    TableStoreInputFormat.setCredential(job, "YourAccessKeyId", "YourAccessKeySecret");
    TableStoreInputFormat.addCriteria(job, fetchCriteria());
    FileOutputFormat.setOutputPath(job, new Path("output"));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

示例中使用job.setInputFormatClass(TableStoreInputFormat.class)将表格存储设置为数据源，除此之外，还需要：

- 把hadoop-connector.jar部署到集群上并添加到classpath中。路径为addFileToClassPath()指定hadoop-connector.jar的本地路径。代码中假定hadoop-connector.jar在当前路径。
- 访问表格存储需要指定入口和身份。通过TableStoreInputFormat.setEndpoint()和TableStoreInputFormat.setCredential()设置访问表格存储需要指定的Endpoint和AccessKey信息。
- 指定一张表用来计数。

> **说明**
>
> - 每调用一次addCriteria()可以在数据源里添加一个Java SDK定义的RangeRowQueryCriteria对象。可以多次调用addCriteria()。RangeRowQueryCriteria对象与表格存储Java SDK GetRange接口所用的RangeRowQueryCriteria对象具有相同的限制条件。
> - 使用RangeRowQueryCriteria的setFilter()和addColumnsToGet()可以在表格存储的服务器端过滤掉不必要的行和列，减少访问数据的大小，降低成本，提高性能。
> - 通过添加对应多张表的多个RangeRowQueryCriteria，可以实现多表的union。
> - 通过添加同一张表的多个RangeRowQueryCriteria，可以做到更均匀的切分。TableStore-Hadoop Connector会根据一些策略将用户传入的范围切细。

程序运行示例

1. 设置HADOOP_CLASSPATH。

```
HADOOP_CLASSPATH=hadoop-connector.jar bin/hadoop jar row-counter.jar
```

2. 执行**find output -type f**命令查找output目录下的所有文件。

   返回结果示例如下：

```
output/_SUCCESS
output/part-r-00000
output/._SUCCESS.crc
output/.part-r-00000.crc
```

3. 执行**cat out/part-r-00000**命令统计运行结果中的行数。

```
TOTAL    9
```

类型转换说明

表格存储支持的数据类型与Hive或者Spark支持的数据类型不完全相同。

下表列出了从表格存储的数据类型（行）转换到Hive或者Spark数据类型（列）的支持情况。

类型转换	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	BOOLEAN	STRING	BINARY
INTEGER	支持，损失精度	支持，损失精度	支持，损失精度	支持	支持，损失精度	支持，损失精度	不支持	不支持	不支持
DOUBLE	支持，损失精度	支持，损失精度	支持，损失精度	支持，损失精度	支持，损失精度	支持	不支持	不支持	不支持
BOOLEAN	不支持	不支持	不支持	不支持	不支持	不支持	支持	不支持	不支持
STRING	不支持	不支持	不支持	不支持	不支持	不支持	不支持	支持	不支持
BINARY	不支持	不支持	不支持	不支持	不支持	不支持	不支持	不支持	支持

5.函数计算
5.1. 使用函数计算

通过函数计算访问表格存储，对表格存储增量数据进行实时计算。

背景信息

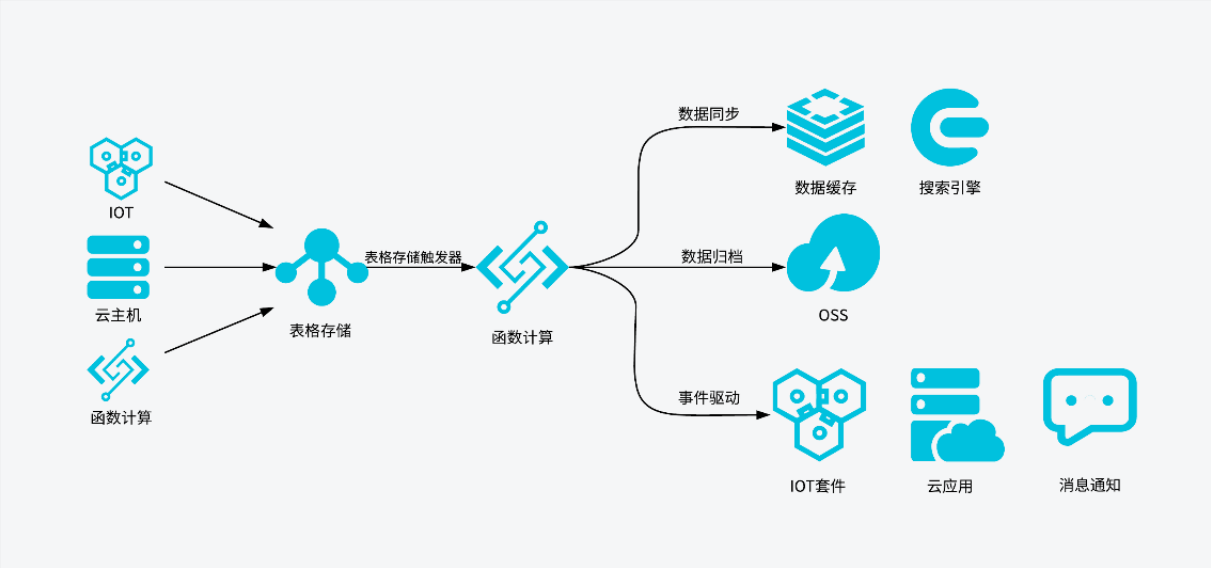
函数计算（Function Compute）是一个事件驱动的服务，通过函数计算，无需管理服务器等运行情况，只需编写代码并上传。函数计算准备计算资源，并以弹性伸缩的方式运行用户代码，而您只需根据实际代码运行所消耗的资源进行付费。更多信息，请参见什么是函数计算。函数计算示例请参见表格存储函数计算示例。

Tablestore Stream是用于获取Tablestore表中增量数据的一个数据通道，通过创建Tablestore触发器，能够实现Stream和函数计算的自动对接，让计算函数中自定义的程序逻辑自动处理Tablestore表中发生的数据修改。关于Tablestore触发器的更多信息，请参见Tablestore触发器。

使用场景

如下图所示，使用函数计算可以实现如下任务。

- 数据同步：同步表格存储中的实时数据到数据缓存、搜索引擎或者其他数据库实例中。
- 数据归档：增量归档表格存储中的数据到OSS等做冷备份。
- 事件驱动：利用触发器触发函数调用IoT套件、云应用API或者做消息通知等。



准备工作

- 已创建表格存储实例和数据表。具体操作，请参见创建实例和创建数据表。
- 已开通函数计算服务。具体操作，请参见开通函数计算服务。

注意事项

请确保表格存储数据表与函数计算服务处于同一地域。

步骤一：为数据表开启Stream功能

使用触发器功能需要先在表格存储控制台开启数据表的Stream功能，才能在函数计算中处理写入表格存储中的增量数据。

1. 登录表格存储控制台。
2. 在**概览**页面，单击实例名称或在操作列单击**实例管理**。
3. 在**实例详情**页签的**数据表列表**区域，单击数据表名称后选择**实时消费通道**页签或单击 ⋮ 后选择**实时消费通道**。
4. 在**实时消费通道**页签，单击Stream信息对应的**开启**。
5. 在**开启Stream功能**对话框，设置日志过期时长，单击**开启**。

   日志过期时长取值为非零整数，单位为小时，最长时长为168小时。

   > **说明** 日志过期时长设置后不能修改，请谨慎设置。

步骤二：配置Tablestore触发器

在函数计算控制台创建Tablestore触发器来处理Tablestore数据表的实时数据流。

1. 创建函数计算的服务。
   i. 登录函数计算控制台。
   ii. 在左侧导航栏，单击**服务及函数**。
   iii. 在顶部菜单栏，选择地域。
   iv. 在**服务列表**页面，单击**创建服务**。
   v. 在**创建服务**面板，填写服务名称和描述，按需设置日志与链路追踪功能。

      关于参数说明的更多信息，请参见管理服务。
   vi. 单击**确定**。

      创建完成后，在**服务列表**页面，您可以查看到已创建的服务及其配置信息。
2. 创建函数计算的函数。

   > **说明** 系统支持使用标准Runtime从零创建、使用自定义运行时平滑迁移Web Server、使用容器镜像创建、使用模板创建等方式创建函数。此处以**使用标准Runtime从零创建**方式为例介绍，其他创建方式，请分别参见使用容器镜像创建函数和使用模板创建函数。

   i. 在**服务列表**页面，单击目标服务名称。
   ii. 在**函数管理**页面，单击**创建函数**。
   iii. 在**创建函数**页面，选择创建函数的方式为**使用标准Runtime从零创建**。
   iv. 在**基本设置**区域，根据下表说明填写函数基本信息。

参数	是否必填	说明	本文示例

参数	是否必填	说明	本文示例
函数名称	否	填写自定义的函数名称。 **说明** 如果不填写名称，函数计算会自动为您创建。	Function
运行环境	是	选择您熟悉的语言，例如Python、Java、PHP、Node.js等。函数计算支持的运行环境，请参见管理函数。	Python 3.6
代码上传方式	否	默认选择**使用示例代码**创建函数，函数创建完成后自带函数计算平台为其配置的示例代码。您也可以选择以下三种方式上传您自己的代码。 ■ **通过ZIP包上传代码**：选择函数代码ZIP包。 ■ **通过文件夹上传代码**：选择包含函数代码的文件夹。 ■ **通过OSS上传代码**：选择上传函数代码的**Bucket名称**和**文件名称**。	使用示例代码
启动命令	否	程序的启动命令。如果不配置启动命令，您需要在代码的根目录手动创建一个名称为bootstrap的启动脚本，您的程序通过此脚本来启动。 **说明** 仅当您选择**使用自定义运行时平滑迁移Web Server**时，需配置此参数。	npm run start
监听端口	是	您的代码中的HTTP Server所监听的端口。 **说明** 仅当您选择**使用自定义运行时平滑迁移Web Server**时，需配置此参数。	9000
请求处理程序类型	是	选择请求处理程序类型。取值范围如下： ■ **处理事件请求**：通过定时器、调用API/SDK或其他阿里云服务的触发器来触发函数执行。 ■ **处理HTTP请求**：用于处理HTTP请求或Websocekt请求的函数。如果您的使用场景是Web场景，建议您**使用自定义运行时平滑迁移Web Server**。具体信息，请参见函数计算控制台。 使用表格存储触发器时，此参数必须设置为**处理事件请求**。	处理事件请求

参数	是否必填	说明	本文示例
实例类型	是	选择适合您的实例类型。取值范围如下： ■ **弹性实例** ■ **性能实例** 更多信息，请参见实例类型及使用模式。关于各种实例类型的计费详情，请参见计费概述。	弹性实例
内存规格	是	设置函数执行内存。 ■ **选择输入**：在下拉列表中选择所需内存。 ■ **手动输入**：仅适用于弹性实例，单击**手动输入内存大小**，可自定义函数执行内存。取值范围[128, 3072]，单位为MB。 **说明** 输入的内存必须为64 MB的倍数。	512 MB
实例并发度	是	设置函数实例的并发度，具体信息，请参见设置实例并发数。	1
请求处理程序	是	设置请求处理程序，函数计算的运行时会加载并调用您的请求处理程序处理请求。	index.handler

函数创建完成后，在**函数管理**页面，即可查看已创建的函数。

v. 在**配置触发器**区域，根据下表说明填写触发器相关参数。

参数	操作	本文示例
触发器类型	选择**表格存储Tablestore**。 说明 当**请求处理程序类型**选择**处理HTTP请求**时，此参数默认为**HTTP触发器**。	表格存储Tablestore
名称	自定义填写触发器名称。	Tablestore-trigger
实例	在下拉列表中选择已创建的Tablestore实例。	distribute-test
表格	在下拉列表中选择已创建的数据表。	source_data
角色名称	选择**AliyunTableStoreStreamNotificationRole**。 说明 如果您第一次创建该类型的触发器，则需要单击**确定**后，在弹出的对话框中选择**立即授权**，并根据系统提示完成角色创建和授权。	AliyunTableStoreStreamNotificationRole

vi. 单击**创建**。

创建好的触发器会自动显示在**触发器管理**页签。

说明 您也可以在表格存储控制台中数据表的**触发器管理**页签，查看和创建Tablestore触发器。

步骤三：验证测试

函数计算支持函数的在线调试功能，您可以构建触发的Event，并测试代码逻辑是否符合期望。

由于Tablestore触发函数服务的Event是CBOR格式，该数据格式是一种类似JSON的二进制格式，可以按照如下方法进行在线调试。

1. 编写代码。
   i. 在**函数管理**页面，单击函数名称。

ii. 在函数详情页面，单击**函数代码**页签，在代码编辑器中编写代码。

> **说明** 此处以Python函数代码为例介绍。如果您想使用其他运行环境，更多代码示例，请参见表格存储触发函数计算示例。

在代码中使用 `records = json.loads(event)` 处理自定义的测试触发事件。

```
import logging
import cbor
import json
 def get_attribute_value(record, column):
     attrs = record[u'Columns']
     for x in attrs:
         if x[u'ColumnName'] == column:
             return x['Value']
 def get_pk_value(record, column):
     attrs = record[u'PrimaryKey']
     for x in attrs:
         if x['ColumnName'] == column:
             return x['Value']
 def handler(event, context):
     logger = logging.getLogger()
     logger.info("Begin to handle event")
     #records = cbor.loads(event)
     records = json.loads(event)
     for record in records['Records']:
         logger.info("Handle record: %s", record)
         pk_0 = get_pk_value(record, "pk_0")
         attr_0 = get_attribute_value(record, "attr_0")
     return 'OK'
```

2. 验证测试。

i. 在**函数代码**页签，单击**测试函数**右侧的 图标后，从下拉列表中，选择**配置测试参数**。

ii. 在**配置测试参数**对话框，选择**创建新测试事件**或**编辑已有测试事件**页签，选择**事件模板**为**Tablestore**，并填写事件名称和事件内容。单击**确定**。

关于事件内容数据格式的更多信息，请参见附录：数据处理。

iii. 单击**测试函数**，查看是否符合期望。

当使用 `records=json.loads(event)` 测试OK后，可以修改为 `records = cbor.loads(event)` 并单击**部署代码**，当Tablestore中有数据写入时，相关的函数逻辑就会触发。

附录：数据处理

配置测试参数时，需要按照特定的数据格式设置事件内容。

- 数据格式

  Tablestore触发器使用CBOR格式对增量数据进行编码构成函数计算的Event。增量数据的具体数据格式如下：

```
{
    "Version": "string",
    "Records": [
        {
            "Type": "string",
            "Info": {
                "Timestamp": int64
            },
            "PrimaryKey": [
                {
                    "ColumnName": "string",
                    "Value": formated_value
                }
            ],
            "Columns": [
                {
                    "Type": "string",
                    "ColumnName": "string",
                    "Value": formated_value,
                    "Timestamp": int64
                }
            ]
        }
    ]
}
```

- 成员定义

  成员定义请参见下表。

参数	说明
Version	payload版本号，目前为Sync-v1。类型为string。
Records	数据表中的增量数据行数组。包含如下内部成员： ○ Type：数据行类型，包含PutRow、UpdateRow和DeleteRow。类型为string。 ○ Info：包含Timestamp内部成员。Timestamp表示该行的最后修改UTC时间。类型为int64。
PrimaryKey	主键列数组。包含如下内部成员： ○ ColumnName：主键列名称。类型为string。 ○ Value：主键列内容。类型为formated_value，支持integer、string和blob。

参数	说明
Columns	属性列数组。包括如下内部成员： ○ Type：属性列类型，包含Put、DeleteOneVersion和DeleteAllVersions。类型为string。 ○ ColumnName：属性列名称。类型为string。 ○ Value：属性列内容。类型为formated_value，支持integer、boolean、double、string和blob。 ○ Timestamp：属性列最后修改UTC时间。类型为int64。

- 数据示例

```
{
    "Version": "Sync-v1",
    "Records": [
        {
            "Type": "PutRow",
            "Info": {
                "Timestamp": 1506416585740836
            },
            "PrimaryKey": [
                {
                    "ColumnName": "pk_0",
                    "Value": 1506416585881590900
                },
                {
                    "ColumnName": "pk_1",
                    "Value": "2017-09-26 17:03:05.8815909 +0800 CST"
                },
                {
                    "ColumnName": "pk_2",
                    "Value": 1506416585741000
                }
            ],
            "Columns": [
                {
                    "Type": "Put",
                    "ColumnName": "attr_0",
                    "Value": "hello_table_store",
                    "Timestamp": 1506416585741
                },
                {
                    "Type": "Put",
                    "ColumnName": "attr_1",
                    "Value": 1506416585881590900,
                    "Timestamp": 1506416585741
                }
            ]
        }
    ]
}
```

附录：使用已有函数计算创建Tablestore触发器

在表格存储控制台，使用已有函数计算的服务以及服务下的函数创建Tablestore触发器。

1. 登录表格存储控制台。
2. 在**概览**页面，单击实例名称或在操作列单击**实例管理**。
3. 在**实例详情**页签的**数据表列表**区域，单击数据表名称。
4. 在**触发器管理**页签，单击**使用已有函数计算**。
5. 在**创建触发器**对话框，选择函数计算服务以及服务下的函数，并填写触发器名称，单击**确定**。

5.2. 使用函数计算清洗数据

表格存储高并发的写入性能以及低廉的存储成本非常适合物联网、日志、监控数据的存储。将数据写入到表格存储时，您可以通过函数计算对新增的数据做简单的清洗，将清洗后的数据写回到表格存储的另一种数据表中。同时，您也可以实时访问表格存储中的原始数据和结果数据。

样例场景

假设写入表格存储的为日志数据，且日志数据包括如下三个字段。为了便于日志查询，您需要将level>1的日志写入到表格存储的另一张数据表result中。

字段名称	类型	说明
id	整型	日志ID。
level	整型	日志的等级，数值越大表示日志等级越高。
message	字符串	日志的内容。

步骤一：为数据表开启Stream功能

使用触发器功能需要先在表格存储控制台开启数据表的Stream功能，才能在函数计算中处理写入表格存储中的增量数据。

1. 登录表格存储控制台。
2. 在**概览**页面，单击实例名称或在操作列单击**实例管理**。
3. 在**实例详情**页签的**数据表列表**区域，单击数据表名称后选择**实时消费通道**页签或单击 ⋮ 后选择**实时消费通道**。
4. 在**实时消费通道**页签，单击Stream信息对应的**开启**。
5. 在**开启Stream功能**对话框，设置日志过期时长，单击**开启**。

   日志过期时长取值为非零整数，单位为小时，最长时长为168小时。

   > **说明** 日志过期时长设置后不能修改，请谨慎设置。

步骤二：配置Tablestore触发器

在函数计算控制台创建Tablestore触发器来处理Tablestore数据表的实时数据流。

1. 创建函数计算的服务。
   i. 登录函数计算控制台。
   ii. 在左侧导航栏，单击**服务及函数**。
   iii. 在顶部菜单栏，选择地域。
   iv. 在**服务列表**页面，单击**创建服务**。
   v. 在**创建服务**面板，填写服务名称和描述，按需设置日志与链路追踪功能。
   关于参数说明的更多信息，请参见管理服务。
   vi. 单击**确定**。
   创建完成后，在**服务列表**页面，您可以查看到已创建的服务及其配置信息。
2. 创建函数计算的函数。

> **说明** 系统支持使用标准Runtime从零创建、使用自定义运行时平滑迁移Web Server、使用容器镜像创建、使用模板创建等方式创建函数。此处以**使用标准Runtime从零创建**方式为例介绍，其他创建方式，请分别参见使用容器镜像创建函数和使用模板创建函数。

   i. 在**服务列表**页面，单击目标服务名称。
   ii. 在**函数管理**页面，单击**创建函数**。
   iii. 在**创建函数**页面，选择创建函数的方式为**使用标准Runtime从零创建**。
   iv. 在**基本设置**区域，根据下表说明填写函数基本信息。

参数	是否必填	说明	本文示例
函数名称	否	填写自定义的函数名称。 **说明** 如果不填写名称，函数计算会自动为您创建。	Function
运行环境	是	选择您熟悉的语言，例如Python、Java、PHP、Node.js等。函数计算支持的运行环境，请参见管理函数。	Python 3.6
代码上传方式	否	默认选择**使用示例代码**创建函数，函数创建完成后自带函数计算平台为其配置的示例代码。您也可以选择以下三种方式上传您自己的代码。 ■ **通过ZIP包上传代码**：选择函数代码ZIP包。 ■ **通过文件夹上传代码**：选择包含函数代码的文件夹。 ■ **通过OSS上传代码**：选择上传函数代码的**Bucket名称**和**文件名称**。	使用示例代码

参数	是否必填	说明	本文示例
启动命令	否	程序的启动命令。如果不配置启动命令，您需要在代码的根目录手动创建一个名称为bootstrap的启动脚本，您的程序通过此脚本来启动。 说明 仅当您选择**使用自定义运行时平滑迁移Web Server**时，需配置此参数。	npm run start
监听端口	是	您的代码中的HTTP Server所监听的端口。 说明 仅当您选择**使用自定义运行时平滑迁移Web Server**时，需配置此参数。	9000
请求处理程序类型	是	选择请求处理程序类型。取值范围如下： ■ **处理事件请求**：通过定时器、调用API/SDK或其他阿里云服务的触发器来触发函数执行。 ■ **处理HTTP请求**：用于处理HTTP请求或Websocekt请求的函数。如果您的使用场景是Web场景，建议您**使用自定义运行时平滑迁移Web Server**。具体信息，请参见函数计算控制台。 使用表格存储触发器时，此参数必须设置为**处理事件请求**。	处理事件请求
实例类型	是	选择适合您的实例类型。取值范围如下： ■ **弹性实例** ■ **性能实例** 更多信息，请参见实例类型及使用模式。关于各种实例类型的计费详情，请参见计费概述。	弹性实例
内存规格	是	设置函数执行内存。 ■ **选择输入**：在下拉列表中选择所需内存。 ■ **手动输入**：仅适用于弹性实例，单击**手动输入内存大小**，可自定义函数执行内存。取值范围[128, 3072]，单位为MB。 说明 输入的内存必须为64 MB的倍数。	512 MB
实例并发度	是	设置函数实例的并发度，具体信息，请参见设置实例并发数。	1
请求处理程序	是	设置请求处理程序，函数计算的运行时会加载并调用您的请求处理程序处理请求。	index.handler

函数创建完成后，在**函数管理**页面，即可查看已创建的函数。

v. 在**配置触发器**区域，根据下表说明填写触发器相关参数。

参数	操作	本文示例
触发器类型	选择**表格存储Tablestore**。 **说明** 当**请求处理程序类型**选择**处理HTTP请求**时，此参数默认为**HTTP触发器**。	表格存储Tablestore
名称	自定义填写触发器名称。	Tablestore-trigger
实例	在下拉列表中选择已创建的Tablestore实例。	distribute-test
表格	在下拉列表中选择已创建的数据表。	source_data
角色名称	选择**AliyunTableStoreStreamNotificationRole**。 **说明** 如果您第一次创建该类型的触发器，则需要单击**确定**后，在弹出的对话框中选择**立即授权**，并根据系统提示完成角色创建和授权。	AliyunTableStoreStreamNotificationRole

vi. 单击**创建**。

创建好的触发器会自动显示在**触发器管理**页签。

说明 您也可以在表格存储控制台中数据表的**触发器管理**页签，查看和创建Tablestore触发器。

步骤三：验证测试

创建触发器后，通过在表格存储中写入和查询数据验证数据清洗是否成功。

1. 编写代码。

i. 在**函数管理**页面，单击函数名称。

ii. 在函数详情页面，单击**函数代码**页签，在代码编辑器中编写代码。

此处以Python函数代码为例介绍。其中INSTANCE_NAME（表格存储的实例名称）、REGION（使用的区域）、ENDPOINT（服务地址）需要根据情况进行修改。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import cbor
import json
import tablestore as ots
INSTANCE_NAME = 'distribute-test'
REGION = 'cn-shanghai'
ENDPOINT = 'http://%s.%s.vpc.tablestore.aliyuncs.com'%(INSTANCE_NAME, REGION)
RESULT_TABLENAME = 'result'
def _utf8(input):
    return str(bytearray(input, "utf-8"))
def get_attrbute_value(record, column):
    attrs = record[u'Columns']
    for x in attrs:
        if x[u'ColumnName'] == column:
            return x['Value']
def get_pk_value(record, column):
    attrs = record[u'PrimaryKey']
    for x in attrs:
        if x['ColumnName'] == column:
            return x['Value']
#由于已经授权了AliyunOTSFullAccess权限，此处获取的credentials具有访问表格存储的权限。
def get_ots_client(context):
    creds = context.credentials
    client = ots.OTSClient(ENDPOINT, creds.accessKeyId, creds.accessKeySecret, INST
ANCE_NAME, sts_token = creds.securityToken)
    return client
def save_to_ots(client, record):
    id = int(get_pk_value(record, 'id'))
    level = int(get_attrbute_value(record, 'level'))
    msg = get_attrbute_value(record, 'message')
    pk = [(_utf8('id'), id),]
    attr = [(_utf8('level'), level), (_utf8('message'), _utf8(msg)),]
    row = ots.Row(pk, attr)
    client.put_row(RESULT_TABLENAME, row)
def handler(event, context):
    records = cbor.loads(event)
    #records = json.loads(event)
    client = get_ots_client(context)
    for record in records['Records']:
        level = int(get_attrbute_value(record, 'level'))
        if level > 1:
            save_to_ots(client, record)
        else:
            print "Level <= 1, ignore."
```

2. 向source_data数据表中写入数据，依次填入id、level和message信息，并在result表中查询清洗后的数据。
   - 当向soure_data表中写入level>1的数据时，数据会同步到result表中。
   - 当向soure_data表中写入level<=1的数据时，数据不会同步到result表中。