ALIBABA CLOUD

阿里云

表格存储Tablestore
数据同步迁移

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.迁移工具

使用DataWorks/DataX、表格存储的通道服务等迁移工具，可以在不影响业务的情况下实现表格存储数据的全量迁移、增量迁移和实时同步。您不仅可以将数据库迁移同步到表格存储，也可以实现表格存储数据表中数据的跨实例或者跨账号的迁移同步。

迁移工具选择

迁移数据时，可用的迁移工具包括Dataworks/DataX和表格存储的通道服务，请根据实际业务选择合适的迁移工具。

迁移工具	说明	应用场景
DataWorks/DataX	Dataworks数据集成是稳定高效、弹性伸缩的数据同步平台，底层实现依赖于DataX离线数据同步工具。适用于MySQL、Oracle、SQLServer等多种异构数据源之间的数据迁移同步。 DataWorks/DataX将不同数据源的同步抽象为从源头数据源读取数据的Reader插件，以及向目标端写入数据的Writer插件，详情请分别参见DataWorks数据集成或者DataX。	• 将MySQL数据同步迁移到表格存储 • 将HBase数据同步到表格存储 • 将MaxCompute数据同步到表格存储 • 将表格存储数据同步到MaxCompute • 将表格存储数据同步迁移到OSS • 将表格存储数据表中数据同步到另一个数据表
通道服务	通道服务（Tunnel Service）是基于表格存储数据接口之上的全增量一体化服务。适用于源表为表格存储数据表的数据迁移同步。 通道服务提供了增量、全量、增量加全量三种类型的分布式数据实时消费通道。通过为数据表建立数据通道，可以简单地实现对表中历史存量和新增数据的消费处理，详情请参见通道服务。	将表格存储数据表中数据同步到另一个数据表

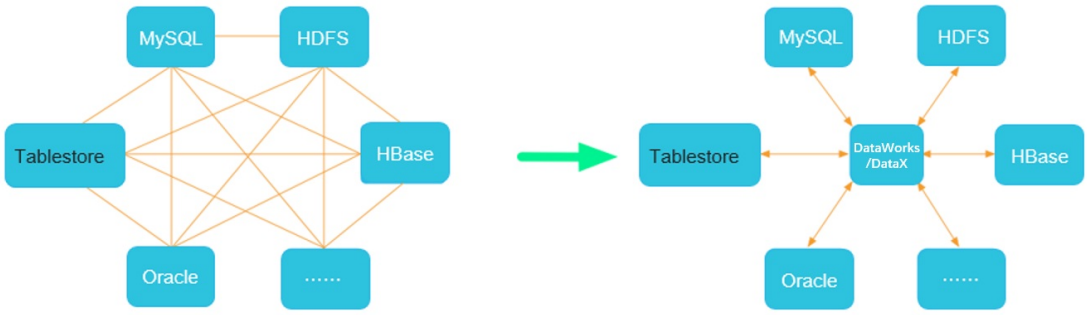
DataWorks/DataX

使用DataWorks/DataX可以实现如下功能：

- 将数据库数据迁移到表格存储

  DataWorks/DataX提供各种异构数据源之间稳定高效的数据同步功能，可以实现将多种数据库迁移到表格存储，如下图所示。

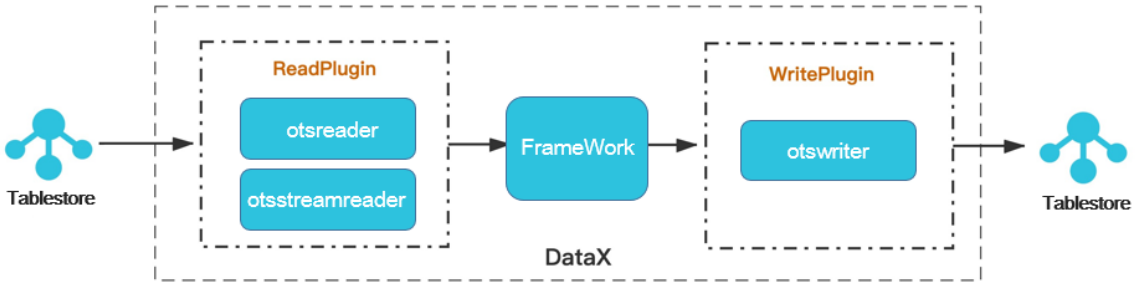
> **说明** DataWorks/DataX支持的数据源与读写插件详情请分别参见DataWorks支持的数据源与读写插件或者DataX支持的数据源与读写插件。

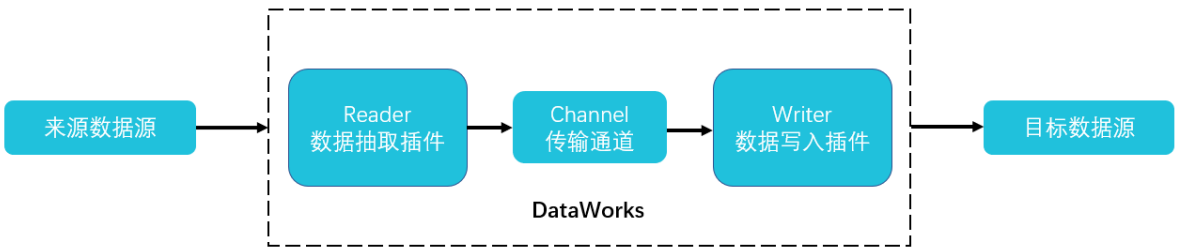


- 表格存储数据跨实例或者跨账号迁移同步

  通过在DataWorks/DataX中配置表格存储相关的Reader和Writer插件，即可以完成表格存储数据表的数据复制，如下图所示。表格存储相关的插件说明请参见下表。

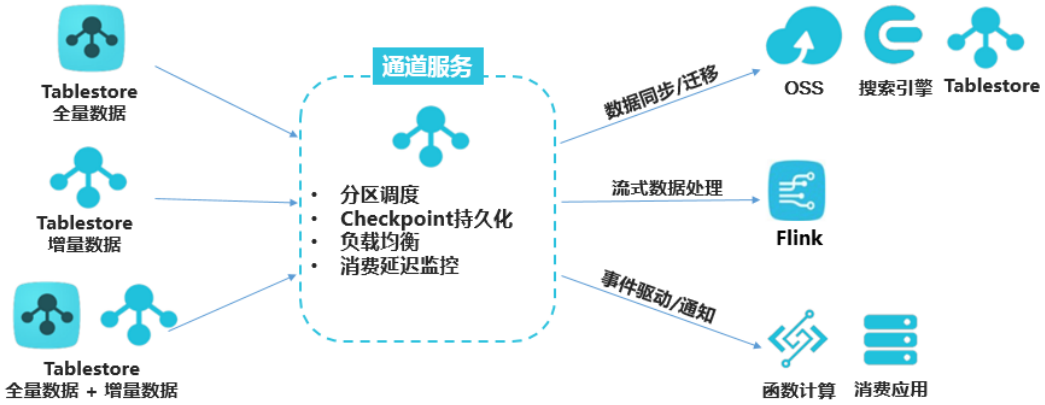
插件	说明
OTSReader	用于读取表格存储数据表的数据，并可以通过指定抽取数据范围实现数据增量抽取的需求。
OTSStreamReader	用于增量导出表格存储数据表的数据。
OTSWriter	用于向表格存储中写入数据。





通道服务

使用通道服务可以轻松构建高效和弹性的数据复制解决方案。



2.数据导入

2.1. 将MySQL数据同步到表格存储

2.1.1. 概述

您可以根据业务需求使用DataX、DTS或者canal工具将MySQL数据库中的数据同步迁移到表格存储（Tablestore）中。

使用场景

- 数据架构变化

  随着业务的变化，如果现有业务对数据库并发读写需求、扩展性和可用性需求较高，或需要复杂的检索，原有MySQL数据库的数据架构已经不能满足现在的业务需求，您可以选择将MySQL数据库中的数据迁移到表格存储中。

  表格存储是阿里云自研的多模型结构化数据存储，提供海量结构化数据存储且可以无限水平扩展。同时，表格存储提供强大查询功能，还支持在线、离线数据分析。此外，表格存储提供全托管服务，使用表格存储您无需担心软硬件预置、配置、故障、集群扩展、安全等问题，可以极大地减少管理成本。

- 大数据分析

  如果您的业务采用MySQL数据库，随着业务的发展，大数据分析场景逐渐增多，而MySQL数据库进行大数据分析需要结合流式组件、存储系统、计算组件等工具，操作复杂且难度大，您可以选择将MySQL数据库迁移到表格存储中实现大数据分析。表格存储具有良好的周边生态，可以对接MaxCompute、Blink等大数据分析工具，轻松实现流处理、批处理。

同步方案

请根据实际数据迁移场景选择合适的同步方案。

同步方案	说明
使用DataX同步	通过DataX，您可以将MySQL数据库（例如自建MySQL或RDS MySQL）中的全量数据同步到表格存储的数据表中。DataX只支持同步全量数据，不支持同步增量数据。
使用DTS同步	通过数据传输服务DTS（Data Transmission Service），您可以将MySQL数据库同步到表格存储实例，轻松实现数据的流转。
使用canal同步	对于中小规模的数据库或者个人开发者，通过canal，您可以将MySQL数据库中的全量数据或者增量数据同步到表格存储的数据表中。canal部署简单，易于运维，适用于中小规模MySQL数据同步。

2.1.2. 使用DataX同步

通过DataX，您可以将MySQL数据库中的全量数据同步到表格存储（Tablestore）的数据表中。DataX只支持同步全量数据，不支持同步增量数据。

前提条件

- 已创建表格存储实例并在实例详情页面获取实例的服务地址（Endpoint）。具体操作，请参见创建实例。

- 已创建表格存储数据表，用于存放迁移数据。具体操作，请参见创建数据表。

  > **说明** 创建数据表时，建议使用MySQL原主键或唯一索引作为表格存储数据表的主键。

- 已获取AccessKey（包括AccessKey ID和AccessKey Secret），用于进行签名认证。具体操作，请参见获取AccessKey。

背景信息

DataX通过MySQL驱动使用Reader中的MySQL连接串配置，直接发送SQL语句获取到查询数据，这些数据会缓存在本地JVM中，然后Writer线程将这些数据写入到表格存储的表中。更多信息，请参见DataX。

步骤一：下载DataX

您可以选择下载DataX的源代码进行本地编译或者直接下载编译好的压缩包。

- 下载DataX的源代码并编译。
  i. 通过Git工具执行以下命令下载DataX源代码。

  ```
  git clone https://github.com/alibaba/DataX.git
  ```

  ii. 进入到下载的源代码目录后，执行以下命令进行Maven打包。

  > **说明** 此步骤会在本地编译各种数据源的Writer和Reader，会花费较长的时间，需要耐心等待。

  ```
  mvn -U clean package assembly:assembly -Dmaven.test.skip=true
  ```

  编译完成后，进入*/target/datax/datax*目录，查看相应的目录。各目录说明请参见下表。

目录	说明
bin	存放可执行的datax.py文件，是DataX工具的入口。
plugin	存放支持各种类型数据源的Reader和Writer。
conf	存放core.json文件，文件中定义了一些默认参数值，例如channel流控、buffer大小等参数，建议使用默认值。

- 下载编译好的压缩包DataX压缩包。

步骤二：准备全量导出的JSON文件

在DataX中mysqlreader配置有querySQL模式和table模式两种模式，请根据实际选择。

- querySQL模式（单task）

  一般用于有条件的数据导出。在此模式下，DataX不会按照指定的column、table参数进行SQL的拼接，而是会略过这些配置（如果有）直接执行querySQL语句。task数量固定为1，因此在此模式下channel的配置无多线程效果。

  querySQL模式的数据导出示例如下：

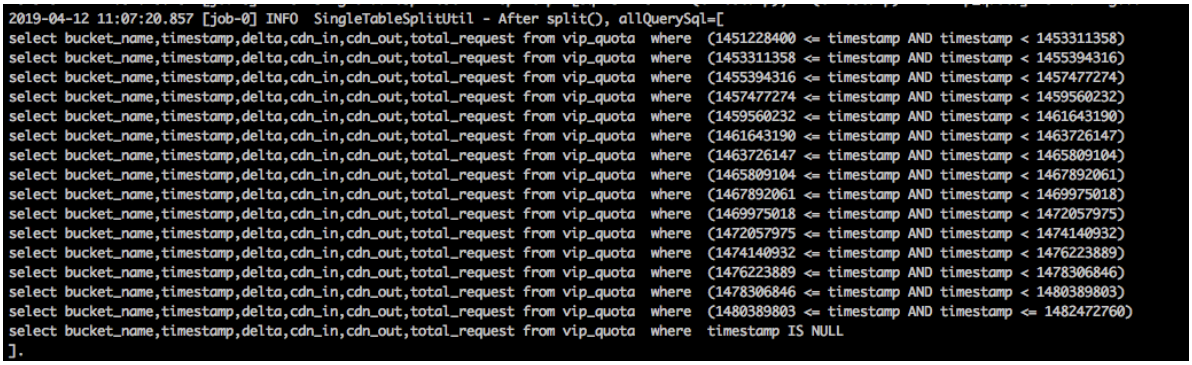
```
{
    "job": {
        "content": [
```

```
            {
                "reader": {
                    "name": "mysqlreader", #指定使用mysqlreader读取数据。
                    "parameter": {
                        "username": "username",#MySQL用户名。
                        "password": "password",#MySQL密码。
                        "connection": [
                            {
                                "querySql": [ #指定执行的SQL语句。
                                    "select bucket_name, delta , timestamp ,cdn_in, cdn_o
ut ,total_request from vip_quota where bucket_name='xxx' "
                                ],
                                "jdbcUrl": ["jdbc:mysql://192.168.0.8:3306/db1?useUnicode
=true&characterEncoding=UTF-8&autoReconnect=true" #jdbc连接串
                                ]
                            }
                        ]
                    }
                },
                "writer": {
                    "name": "otswriter",#指定使用otswriter进行数据写入。
                    "parameter": {#数据源配置。
                        "endpoint":"https://smoke-test.xxxx.ots.aliyuncs.com",#表格存储实例
的服务地址（Endpoint）。
                        "accessId":"xxxx",
                        "accessKey":"xxxx",
                        "instanceName":"smoke-test",#实例名。
                        "table":"vip_quota",#写入数据的目标table名称。
                        #以下为otswriter的限制项配置，默认可以不填，如果数据不符合以下规则，则数
据会被当成脏数据过滤掉。
                        "requestTotalSizeLimitation": 1048576, #单行数据大小限制，默认配置为1
MB，可不配置。
                        "attributeColumnSizeLimitation": 2097152, #单个属性列大小限制，默认
配置为2 MB，可不配置。
                        "primaryKeyColumnSizeLimitation": 1024, #单个主键列大小限制，默认配置
为1 KB，可不配置。
                        "attributeColumnMaxCount": 1024, #属性列个数限制，默认配置为1024，可
不配置。
                        "primaryKey":[# 主键名称和类型。
                            {"name":"bucket_name", "type":"string"},
                            {"name":"delta", "type":"int"},
                            {"name":"timestamp", "type":"int"}
                        ],
                        "column":[#其它column的名称和类型。
                            {"name":"cdn_in","type":"int"},
                            {"name":"cdn_out","type":"int"},
                            {"name":"total_request","type":"int"}
                        ],
                        "writeMode":"UpdateRow" #写入模式。
                    }
                }
            }
        ]
    }
```

```
}
```

- table模式（多task）

  在此模式下无需手动编写select语句，而是由DataX根据JSON中的column、table、splitPk配置项自行拼接SQL语句。观察执行日志如下：



  table模式的数据导出示例代码如下：

```
{
    "job": {
        "setting": {
            "speed": {
                 "channel": 3  #指定channel个数，该参数与并发数密切相关。
            },
            "errorLimit": {#容错限制。
                "record": 0,
                "percentage": 0.02
            }
        },
        "content": [
            {
                "reader": {
                    "name": "mysqlreader",#指定使用mysqlreader读取。
                    "parameter": {
                        "username": "username",#MySQL用户名。
                        "password": "password",#MySQL密码。
                        "column": [  #table模式下可以指定需要查询的列。
                            "bucket_name",
                            "timestamp" ,
                            "delta" ,
                            "cdn_in",
                            "cdn_out" ,
                            "total_request"
                        ],
                        "splitPk": "timestamp",#指定split字段。
                        "connection": [
                            {
                                "table": [#导出的表名。
                                    "vip_quota"
                                 ],
                                "jdbcUrl": ["jdbc:mysql://192.168.1.7:3306/db1"#jdbc连接串
。
                                ]
```

```
                                }
                            ]
                        }
                    },
                    "writer": {
                        "name": "otswriter",#指定使用otswriter进行写入。
                        "parameter": {#数据源配置。
                            "endpoint":"https://smoke-test.xxxx.ots.aliyuncs.com",#表格存储实例
的Endpoint。
                            "accessId":"xxx",
                            "accessKey":"xxx",
                            "instanceName":"smoke-test",#实例名。
                            "table":"vip_quota",#写入目标的table名称。
                            "primaryKey":[#主键名称和类型。
                                {"name":"bucket_name", "type":"string"},
                                {"name":"delta", "type":"int"},
                                {"name":"timestamp", "type":"int"}
                            ],
                            "column":[#其它column的名称和类型。
                                {"name":"haha","type":"int"},
                                {"name":"hahah","type":"int"},
                                {"name":"kengdie","type":"int"}
                            ],
                            "writeMode":"UpdateRow"#写入模式。
                        }
                    }
                }
            ]
        }
}
```

上述JSON文件中定义了一次数据导出导入的数据源信息和部分系统配置。配置主要包括如下两部分：

- setting：主要是speed（与速率、并发相关）和errorLimit（容错限制）。
  - channel：个数决定了reader和writer的个数上限。
  - splitPk：指定了splitPk字段，DataX会将MySQL表中数据按照splitPk切分成n段。splitPk的字段必须是整型或者字符串类型。

    由于DataX的实现方式是按照splitPk字段分段查询数据库表，那么splitPk字段的选取应该尽可能选择分布均匀且有索引的字段，例如主键ID、唯一键等字段。如果不指定splitPk字段，则DataX将不会进行数据的切分，并行度会变为1。

    > **说明** 为了保证同步数据的一致性，要么不配置splitPk字段使用单线程迁移数据，要么确保数据迁移期间停止该MySQL数据库的服务。

- content：主要是数据源信息，包含reader和writer两部分。

> **说明** 配置中的MySQL应该确保执行DataX任务的机器能够正常访问。

步骤三：执行同步命令

执行以下命令同步数据。

```
python datax.py  -j"-Xms4g -Xmx4g" mysql_to_ots.json
```

其中 `-j"-Xms4g -Xmx4g"` 可以限制占用JVM内存的大小。如果不指定，则DataX将使用 `conf/core.json` 中的配置，默认为1 GB。

步骤四：全量同步加速

DataX的数据同步涉及数据读取、数据交换和数据写入三部分，您可以对每个部分进行优化加速。

- 数据读取

  数据源读取有table模式和querySQL模式两种模式，选择table模式可以实现加速读取。具体操作，请参见步骤二：准备全量导出的JSON文件。

- 数据交换

  在数据交换部分，您可以通过以下方面进行同步优化。

  - JVM的内存

    发送给MySQL数据库SQL语句后会得到查询的数据集，并缓存在DataX的buffer中。除此之外，每个channel也维护了自己的record队列。如果存在并发，则channel的个数越多，也会需要更多的内存。因此您可以考虑指定JVM的内存大小参数，即在步骤三：执行同步命令中通过 `-j` 参数来指定JVM的内存大小。

  - channel的个数和流控参数

    在*conf/core.json*中，控制channel的关键参数如下图所示。

    > **说明**
    > - 一般情况下，channel队列本身配置的调整并不常见，但是在使用DataX时应该注意byte和record流控参数。这两个参数都是在flowControlInterval间隔中采样后根据采样值来决定是否进行流控。
    > - 为了提升同步效率，您可以适当提高channel的个数来提高并发数，以及调高每个channel的byte和record限制来提高DataX的吞吐量。
    > - 请综合考虑channel的个数和流控参数，保证理论峰值不会对服务器产生过高的压力。

```
"channel": {
    "class": "com.alibaba.datax.core.transport.channel.memory.MemoryChannel",
    "speed": {
        "byte": -1,
        "record": -1
    },
    "flowControlInterval": 20,
    "capacity": 512,
    "byteCapacity": 67108864
},
```

详细参数说明请参见下表。

参数	描述
capacity	限制channel中队列的大小，即最多缓存的record个数。
byteCapacity	限制record占用的内存大小，单位为字节。默认值为64 MB，如果不指定此参数，则占用内存大小会被配置为8 MB。

参数	描述
byte	控流参数，限制通道的默认传输速率，-1表示不限制。
record	控流参数，限制通道的传输记录个数，-1表示不限制。

capacity和byteCapacity两个参数决定了每个channel能缓存的记录数量和内存占用情况。如果要调整相应参数，您需要按照DataX实际的运行环境进行配置。例如MySQL中每个record都比较大，那么可以考虑适当调高byteCapacity，调整该参数时还请同时考虑机器的内存情况。

```
{
    "core": {  #定义了全局的系统参数，如果不指定会使用默认值。
        "transport": {
            "channel": {
                "speed": {
                    "record": 5000,
                    "byte": 102400
                }
            }
        }
    },
    "job": {
        "setting": {
            "speed": {  #定义了单个channel的控制参数。
                "record": 10000,
            },
            "errorLimit": {
                "record": 0,
                "percentage": 0.02
            }
        },
        "content": [
            {
                "reader": {
                      .....#省略
                },
                "writer": {
                    .....#省略
                }
            }
        ]
    }
}
```

- 数据写入

  Tablestore是基于LSM设计的高性能高吞吐的分布式数据库产品，每一张表都会被切分为很多数据分区，数据分区分布在不同的服务器上，拥有极强的吞吐能力。如果写入能够分散在所有服务器上，则能够利用所有服务器的服务能力，更高速地写入数据，即表分区数量和吞吐能力正相关。

  新建的表默认分区数量都是1，分区数量会随着表不断写入数据而自动分裂不断增长，但是自动分裂的周期较长。对于新建表，如果需要马上进行数据导入，由于单分区很可能不够用导致数据导入不够顺畅，因此建议在新建表时对表进行预分区，在开始导入数据时即可获得极好的性能，而不用等待自动分裂。

```
{
    "job": {
        "setting": {
            ....#省略
        },
        "content": [
            {
                "reader": {
                      .....#省略
                },
                "writer": {
                    "name": "otswriter",
                    "parameter": {
                            .......
                        "writeMode":"UpdateRow",
                        "batchWriteCount":100
                    }
                }
            }
        ]
    }
}
```

2.1.3. 使用DTS同步

通过数据传输服务DTS（Data Transmission Service），您可以将MySQL数据库（例如自建MySQL或RDS MySQL）同步到表格存储（Tablestore）实例，轻松实现数据的流转。

前提条件

- 已创建RAM用户并为RAM用户授予AliyunOTSFullAccess权限（管理表格存储服务的权限）。具体操作，请参见配置RAM用户权限。

  > **注意** 由于在配置目标库时需要填写访问密钥AccessKey（AK）信息来执行授权，为避免阿里云账号泄露AccessKey带来的安全风险，建议您通过RAM用户来完成授权和AccessKey的创建。

- 已获取AccessKey（包括AccessKey ID和AccessKey Secret），用于进行签名认证。具体操作，请参见获取AccessKey。
- 已创建表格存储实例。具体操作，请参见创建实例。
- 已购买数据同步作业。具体操作，请参见购买流程。

  > **说明** 购买数据同步功能时，请选择源实例为MySQL，目标实例为Tablestore，并选择同步拓扑为单向同步。

背景信息

数据传输服务DTS（Data Transmission Service）支持RDBMS、NoSQL、OLAP等数据源间的数据交互，集数据同步、迁移、订阅、集成、加工于一体，助您构建安全、可扩展、高可用的数据架构。更多信息，请参见数据传输服务DTS。

注意事项

- DTS在执行全量数据初始化时将占用源库和目标库一定的读写资源，可能会导致数据库的负载上升，您需要在执行数据同步前评估数据同步对源库和目标库性能的影响，同时建议您在业务低峰期执行数据同步。
- 不支持同步DDL操作。如果同步过程中执行了DDL操作，请先移除同步对象，然后在Tablestore实例中删除该表，最后新增同步对象。

  关于同步对象移除和新增的具体操作，请分别参见移除同步对象和新增同步对象。
- 待同步的表数量不超过64个。如果业务需求会超过此限制，请提交工单提升目标Tablestore实例的表数量限制。
- 待同步的表或列名称符合如下规范：
  - 表或列的名称由大小写字母、数字或下划线（_）组成，且只能以字母或下划线开头。
  - 表或列的名称长度为1~255个字符。

同步初始化类型说明

同步初始化类型	说明
结构初始化	DTS将源库中待同步对象的结构定义信息同步至目标库，目前支持的对象为表。 警告 此场景属于异构数据库间的数据同步，DTS在执行结构初始化时数据类型无法完全对应，请谨慎评估数据类型的映射关系对业务的影响，详情请参见结构初始化涉及的数据类型映射关系。
全量数据初始化	DTS将源库中待同步对象的存量数据，全部同步到目标库中，作为后续增量同步数据的基线数据。
增量数据初始化	DTS在全量数据初始化的基础上，将源库的增量更新数据实时同步至目标库。 在增量数据初始化阶段，DTS支持同步的SQL语句为INSERT、UPDATE、DELETE。 警告 请勿在源库执行DDL语句，否则将导致数据同步失败。

操作步骤

1. 登录数据传输控制台。
2. 首次使用DTS时，请根据提示创建AliyunDTSDefaultRole权限，用于授予DTS访问云资源的权限。具体操作，请参见授予DTS访问云资源的权限。
   i. 在**提示**对话框，单击**前往RAM角色授权**。

ii. 在云资源访问授权界面，确认权限信息并单击**同意授权**。

iii. 授权完成后，返回DTS控制台。

3. 创建同步任务。

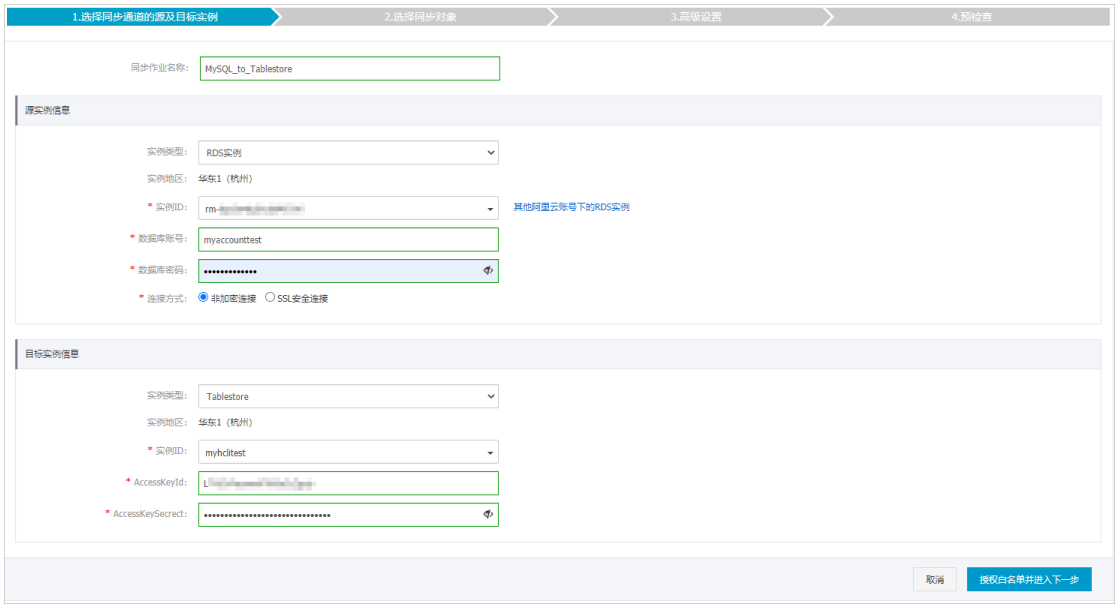
i. 在左侧导航栏，单击**数据同步**。

ii. 在**同步作业列表**区域选择同步的目标实例所属地域。

iii. 定位至已购买的数据同步实例，单击**配置同步链路**。

4. 选择同步通道的源及目标实例。

i. 配置数据同步作业的源库和目标库信息。



类型	配置	描述
无	同步作业名称	DTS会自动生成一个同步作业名称，建议配置具有业务意义的名称（无唯一性要求），便于后续识别。

类型	配置	描述
源实例信息	实例类型	选择**RDS实例**。
	实例地区	购买数据同步实例时选择的源实例地域信息，不可变更。
	实例ID	选择源RDS实例ID。
	数据库账号	填入源RDS的数据库账号，该账号需具备REPLICATION CLIENT、REPLICATION SLAVE、SHOW VIEW和所有同步对象的SELECT权限。 **说明** 当源RDS实例的数据库类型为**MySQL 5.5**或**MySQL 5.6**时，无需配置**数据库账号**和**数据库密码**。
	数据库密码	填入该数据库账号的密码。
	连接方式	根据需求选择**非加密连接**或**SSL安全连接**。如果设置为**SSL安全连接**，您需要提前开启RDS实例的SSL加密功能。具体操作，请参见设置SSL加密。
目标实例信息	实例类型	选择**Tablestore**。
	实例地区	购买数据同步实例时选择的目标实例地域信息，不可变更。
	实例ID	选择目标Tablestore实例ID。
	AccessKeyId	登录账号的AccessKey ID和AccessKey Secret。
	AccessKeySecrect	

ii. 单击**授权白名单并进入下一步**。
此步骤中，DTS会将DTS服务器的IP地址添加至源和目标实例的白名单中，以保障DTS可以正常连接源实例和目标实例。

5. 配置同步策略及对象信息。

i. 配置同步策略。

配置	说明
同步初始化	选中**结构初始化**、**全量数据初始化**和**增量数据初始化**，相关说明请参见同步初始化类型说明。
目标已存在表的处理模式	■ **预检查并报错拦截**：检查目标数据库中是否有同名的表。如果目标数据库中没有同名的表，则通过该检查项目；如果目标数据库中有同名的表，则在预检查阶段提示错误，数据同步作业不会被启动。 **说明** 如果目标库中同名的表不方便删除或重命名，您可以设置同步对象在目标实例中的名称来避免表名冲突。 ■ **忽略报错并继续执行**：跳过目标数据库中是否有同名表的检查项。 **警告** 选择为**忽略报错并继续执行**，可能导致数据不一致，给业务带来风险，例如： ■ 表结构一致的情况下，如果在目标库遇到与源库主键的值相同的记录，在初始化阶段会保留目标库中的该条记录；在增量同步阶段则会覆盖目标库的该条记录。 ■ 表结构不一致的情况下，可能会导致无法初始化数据、只能同步部分列的数据或同步失败。

配置	说明
多表归并	■ 选择为**是**：通常在OLTP场景中，为提高业务表响应速度，通常会做分库分表处理。此类场景中，您可以借助DTS的多表归并功能将源库中多个表结构相同的表（即各分表）同步至Tablestore实例的同一个表中。 **说明** ■ DTS会在Tablestore实例的同步目标表中增加 `__dts_data_source` 列（类型为varchar）来存储数据来源。DTS将以 `<dts数据同步实例ID>:<源数据库名>.<源表名>` 的格式写入列值用于区分表的来源，例如 `dts********:dtstestdata.customer1` 。 ■ 多表归并功能基于实例级别，如果需要让部分表执行多表归并，另一部分不执行多表归并，您需要为这两批表分别创建数据同步作业。 ■ 选择为**否**：默认选项。
同步操作类型	根据业务选中需要同步的操作类型，默认情况下都处于选中状态。
脏数据处理策略	选择数据写入错误时的处理策略： ■ **跳过** ■ **阻塞**
数据写入模式	■ **行更新**：使用PutRowChange会做行级别更新。 ■ **行覆盖**：使用UpdateRowChange会做行级别覆盖。
批量写入方式	批量写入调用接口。 ■ **BulkImportRequest**：离线写入。 ■ **BatchWriteRowRequest**：批量写入。 建议选择**BulkImportRequest**，读写效率更高，Tablestore实例计费便宜。
队列大小	Tablestore实例数据写入进程的队列长度。
线程数	Tablestore实例数据写入进程的回调处理线程数。
并发数	Tablestore实例的并发请求限制数。
分桶数	增量按序写入时的分桶并发数，适当调大可提升并发写入能力。 **说明** 该值需小于或等于并发数。

ii. 配置同步对象。

配置	说明
选择同步对象	在**源库对象**框中单击待同步的表（不能超过64个表），然后单击 › 图标将其移动至**已选择对象**框。 默认情况下，同步对象的名称保持不变。如果您需要同步对象在目标实例上名称不同，请使用对象名映射功能。具体操作，请参见设置同步对象在目标实例中的名称。 **警告** 如果选择**多表归并**为**是**，选择源库的多个表后，您需要通过对象名映射功能，将其改为Tablestore实例中的同一个表名，否则将导致数据不一致。
映射名称更改	如果需要更改同步对象在目标实例中的名称，请使用对象名映射功能。具体操作，请参见库表列映射。
源表DMS_ONLINE_DDL过程中是否复制临时表到目标库	如果源库使用数据管理DMS（Data Management Service）执行Online DDL变更，您可以选择是否同步Online DDL变更产生的临时表数据。 ■ **是**：同步Online DDL变更产生的临时表数据。 **说明** Online DDL变更产生的临时表数据过大，可能会导致同步任务延迟。 ■ **否**：不同步Online DDL变更产生的临时表数据，只同步源库的原始DDL数据。 **说明** 该方案会导致目标库锁表。

配置	说明
源、目标库无法连接重试时间	当源、目标库无法连接时，DTS默认重试720分钟（即12小时），您也可以自定义重试时间。如果DTS在设置的时间内重新连接上源、目标库，同步任务将自动恢复。否则，同步任务将失败。 **说明** 由于连接重试期间，DTS将收取任务运行费用，建议您根据业务需要自定义重试时间，或者在源和目标库实例释放后尽快释放DTS实例。

iii. （可选）在**已选择对象**区域框中，将鼠标指针放置在待同步的表上，并单击表名后出现的**编辑**，设置该表中各列在Tablestore实例中的数据类型。

iv. 单击**下一步**。

v. 在提示框中阅读注意事项后，单击**确认并执行下一步**。

6. 配置主键信息和预检查。

i. 配置待同步的表在Tablestore实例中的主键列信息。

> **说明** 关于Tablestore实例中主键的相关介绍，请参见主键。

ii. 单击**预检查并启动**。

> **说明**
>
> - 在同步作业正式启动之前，会先进行预检查。只有预检查通过后，才能成功启动同步作业。
> - 如果预检查失败，单击具体检查项后的，查看失败详情。
>
> - 您可以根据提示修复后重新进行预检查。
> - 如果无需修复告警检测项，您也可以选择**确认屏蔽**、**忽略告警项并重新进行预检查**，跳过告警检测项重新进行预检查。

iii. 在**预检查**对话框中显示**预检查通过**后，关闭**预检查**对话框，同步作业将正式开始。

7. 等待同步作业的链路初始化完成，直至处于**同步中**状态。

   您可以在**数据同步**页面，查看数据同步作业的状态。

2.1.4. 使用canal同步

对于中小规模的数据库或者个人开发者，您可以使用canal将MySQL数据同步到表格存储。canal部署简单，易于运维，适用于中小规模MySQL数据同步。

前提条件

- 已开启MySQL binlog功能，并且配置binlog-format为ROW模式。
- 已创建目标Tablestore表。具体操作，请参见创建数据表。

> **注意** 目标数据表中主键列的个数、顺序和数据类型必须与源数据表中主键列的个数、顺序和数据类型相匹配。

背景信息

canal基于MySQL数据库增量日志解析，提供增量数据订阅和消费，是阿里开源CDC工具，它可以获取MySQL binlog数据并解析，然后将数据变动传输给下游。基于canal，您可以实现从MySQL到其他数据库的实时同步。更多信息，请参见canal官网。

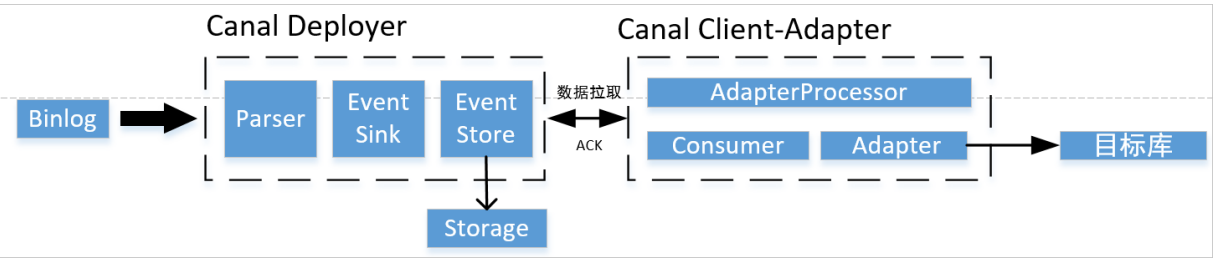
使用流程

使用canal将MySQL数据同步到表格存储的使用流程如下：

1. 部署Deployer服务，该服务负责从上游拉取binlog数据、记录位点等。具体操作，请参见步骤一：部署Deployer。
2. 部署Client-Adapter服务，该服务负责对接Deployer解析过的数据，并将数据传输到目标库中。具体操作，请参见步骤二：部署Client-Adapter。

   部署完成后，canal默认会自动同步MySQL增量数据。
3. 如果需要同步MySQL全量数据，请手动调用Client-Adapter服务的方法触发同步任务。具体操作，请参见步骤三：同步MySQL全量数据。

   待全量数据同步完成后，canal会自动开始增量同步。



步骤一：部署Deployer

> **说明** 由于Deployer包中不存在为表格存储定制的jar包或者配置，因此使用canal对接表格存储时的部署方式与使用canal对接MySQL时的部署方式类似。具体操作，请参见canal快速入门。

1. 下载canal.deployer包并解压。具体下载路径，请参见canal下载地址。

   解压后，您可以看到项目路径下的bin、conf、plugin、lib文件夹。
2. canal实例名称默认为example，如果需要自定义canal实例名称，例如改为test_ots，请执行以下操作。
   i. 修改canal.properties文件中canal.destinations的值为自定义的canal实例名称，其他配置均保持默认即可。

   ```
   canal.destinations = test_ots
   ```

   ii. 在conf路径下创建以canal实例名称命名的文件夹test_ots，并将conf/example路径下的instance.properties文件复制到conf/test_ots/路径下。
3. 修改instance.properties文件。instance.properties文件中的主要配置项说明请参见下表。

配置项	是否必填	示例值	描述

配置项	是否必填	示例值	描述
canal.instance.master.address	是	rm-bp15p07134rkvf7****.mysql.rds.aliyuncs.com:****	canal监听的数据库地址，格式为 `host:port` 。
canal.instance.rds.accesskey	否	LTAn********** *******	当MySQL为阿里云产品RDS库时，填写登录账号的AccessKey ID和AccessKey Secret，获取方式请参见为RAM用户创建访问密钥。如果非RDS库，无需填写此项。
canal.instance.rds.secretkey	否	zbnK********** **********	
canal.instance.rds.instanceId	否	rm-bp15p0713****	当MySQL为阿里云产品RDS库时，填写实例ID。如果非RDS库，无需填写此项。
canal.instance.dbUsername	是	test	数据库账号用户名。
canal.instance.dbPassword	是	db****	数据库账号密码。
canal.instance.filter.regex	是	.*\\..*	canal实例关注的表。通过正则表达式匹配。此处表示匹配所有数据库下的所有表。
canal.destinations	是	test_ots	canal实例名称，必须与配置文件所在上层路径相同。例如配置文件的路径为conf/test_ots/instance.properties，则canal实例名称为test_ots。

4. 通过bin路径下的脚本启动项目。

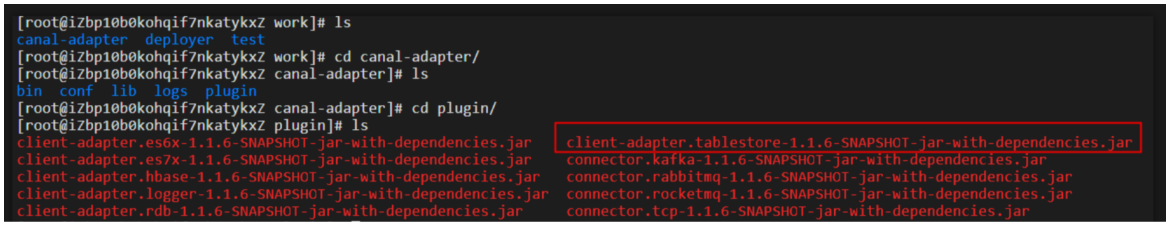
步骤二：部署Client-Adapter

1. 下载canal.adapter包并解压。具体下载路径，请参见canal下载地址。

   解压后，您可以看到项目路径下的bin、conf、plugin、lib文件夹。

2. 如果plugin路径下不存在以client-adapter.tablestore开头的jar包，请下载canal-adapter部署包并解压，然后使用该部署包的内容替代原有的canal.adapter包。

   > **说明** 如果plugin路径下存在以client-adapter.tablestore开头的jar包，则说明部署包中已包含canal对接表格存储的适配器代码，此时可以直接使用该部署包。



3. 修改配置文件。

   i. 修改conf路径下application.yml文件中表格存储相关的配置信息，详细配置项说明请参见下表，其

他配置项的说明请参见ClientAdapter配置项说明。

配置项	是否必填	示例值	描述
canal.conf:canalAdapters:instance	是	test_ots	canal实例名称，必须与部署Depolyer时自定义的canal实例名称相同。
canal.conf:canalAdapters:outerAdapters:-name:	是	tablestore	定义适配器类型。设置此项为tablestore，表示此适配器下游写入Tablestore库。
canal.conf:canalAdapters:outerAdapters:properties:tablestore.endpoint	是	https://test-2009.cn-hangzhou.ots.aliyuncs.com	填写目标Tablestore实例的服务地址。
canal.conf:canalAdapters:outerAdapters:properties:tablestore.accessSecretId	是	LTAn**********	登录账号的AccessKey ID和AccessKey Secret，获取方式请参见为RAM用户创建访问密钥。
canal.conf:canalAdapters:outerAdapters:properties:tablestore.accessSecretKey	是	zbnK**********	
canal.conf:canalAdapters:outerAdapters:properties:tablestore.instanceName	是	test-2009	Tablestore实例的名称。
canal.conf:syncBatchSize	否	1000	一个请求批次中涉及的行数，统计DML（Data Manipulation Language）操作中的数据大小。 如果一次拉取到的DML中涉及到的行数过大，且行数大于此参数值，则该批次数据会被拆分处理。
canal.conf:retries	否	3	Adapterprocessor层的重试次数。默认值为3。

配置项	是否必填	示例值	描述
canal.conf: timeout	否	1000	Adapterprocessor层通过consumer向上游拉取数据的超时时间。单位为毫秒。 如果设置timeout为0，则拉取数据时，Adapterprocessor层会处于阻塞状态直到拉取到数据。
canal.conf： consumerProperties:canal.tcp.batch.size	否	500	consumer向上游拉取的DML个数。
canal.conf: terminateOnException	否	true	默认为false。如果配置为true，则数据同步重试后仍失败，程序会暂停实时同步任务，等待用户手动处理。

application.yml文件的完整配置示例如下：

```
server:
  port: 8081
spring:
  jackson:
    date-format: yyyy-MM-dd HH:mm:ss
    time-zone: GMT+8
    default-property-inclusion: non_null
canal.conf:
  mode: tcp #tcp kafka rocketMQ rabbitMQ
  flatMessage: true
  zookeeperHosts:
  syncBatchSize: 1000
  retries: 3
  timeout:
  accessKey:
  secretKey:
  terminateOnException: true
  consumerProperties:
    # canal tcp consumer
    canal.tcp.server.host: 127.0.0.1:11111
    canal.tcp.zookeeper.hosts:
    canal.tcp.batch.size: 500
    canal.tcp.username:
    canal.tcp.password:
  srcDataSources:
    defaultDS:
      url: jdbc:mysql://rm-bp15po.mysql.rds.aliyuncs.com:3306/test_ots?useUnicode=t
rue
      username: ****
      password: ****
  canalAdapters:
  - instance: test_ots # canal instance Name or mq topic name
    groups:
    - groupId: g1
      outerAdapters:
      - name: logger
      - name: tablestore
        key: ts
        properties:
          tablestore.endpoint: https://test-2009.cn-hangzhou.ots.aliyuncs.com
          tablestore.accessSecretId: ****
          tablestore.accessSecretKey: ****
          tablestore.instanceName: test-2009
```

ii. 在conf/tablestore路径下创建.yml格式的文件并配置。

> **说明** 如果不存在conf/tablestore路径，请手动创建该路径。

a. 在conf/tablestore路径下创建.yml格式的文件，例如mytest.yml。

b. 在mytest.yml文件中填写以下内容并配置相应参数。

```
dataSourceKey: defaultDS
destination: test_ots
groupId: g1
outerAdapterKey: ts
threads: 8
updateChangeColumns: false
dbMapping:
  database: test_ots
  table: order_contract_canal
  targetTable: canal_target_order
  targetPk:
    oId: oId
  targetColumns:
    oId:
    create_time: createTime$string
    pay_time: $string
    update_time: updateTime
  etlCondition:
  commitBatch: 200 # 批量提交的行数。
```

配置项	是否必填	示例值	描述
dataSourceKey	是	defaultDS	该任务的源数据库标识，您可以在application.yml中canal.conf:srcDataSources下找到该标识对应的数据库。
destination	是	test_ots	canal实例名称，必须与application.yml中的canal.conf:canalAdapters下instance值相同。
groupId	是	g1	分组ID，与application.yml中canal.conf: canalAdapters: groups下groupId值相同即可。在MQ模式下才需要使用，此处无需关注。
outerAdapterKey	是	ts	使用的Adapter标识，必须与application.yml中canal.conf:canalAdapters: groups:outerAdapters下key值相同。
threads	是	8	Bucket数量，默认值为1，对应于tablestorewriter中的bucket数量。
dbMapping.database	是	test_ots	源数据库名称。
dbMapping.table	是	order_contract_canal	源表名称。
dbMapping.targetTable	是	canal_target_order	目标表名称。

配置项	是否必填	示例值	描述
dbMapping.targetPk	是	oId: oId	主键配置，格式为 `id: target_id`，即 **源表主键：目标表主键** 。 当表存在多个主键时需要配置多个，多个主键配置顺序必须与Tablestore中的主键顺序相同。 dbMapping.targetPk中支持配置主键列自增，格式为 `$$: target_id` ，表示在目标表中生成一列名称为target_id的主键且该主键列为自增列。当上游数据写入Tablestore时，canal adapter会自动填充该列的值。关于主键列自增的更多信息，请参见主键列自增。

配置项	是否必填	示例值	描述
dbMapping.targetColumns	是	create_time:createTime$string	配置需要同步的列名以及列映射，支持配置类型转换。 **注意** 在dbMapping.targetPk中配置的非自增的主键列也需要在此处再进行配置，自增主键列不需要再进行配置。 如果不配置类型转换，则canal会根据源表中字段类型推断目标字段类型。更多信息，请参见源表和目标Tablestore表中字段映射。 支持配置的格式包含如下4种，请根据实际需要配置。 **说明** 在配置映射的字段类型时，字段类型大小写不敏感。 ■ `id: target_id$string`：表示源表中id字段同步到目标表后为target_id字段，且字段类型映射为string。 ■ `id: target_id`：表示源表中id字段同步到目标表后为target_id字段。 ■ `id:`：表示id字段同步前后字段名不变，字段类型采用默认映射。 ■ `id: $string`：等同于`id: id$string`，表示id字段同步前后字段名不变，且字段类型映射为string。
dbMapping.etlCondition	否	where create_time > "2021-01-01"	全量抽取数据时的过滤条件，其中字段名称为源表字段名称。
dbMapping.commitBatch	否	200	一次批量RPC请求导入的行数，对应于tablestorewriter中的maxBatchRowsCount，默认为writerConfig中的默认值200。
updateChangeColumns	否	false	行覆盖或行更新。默认值为false，表示行覆盖，即一行数据更新时，使用该行最新的整行值覆盖Tablestore中的旧行。如果设置为true，则表示行更新，即一行数据更新时，只对变化的字段进行操作。

4. 通过bin路径下的脚本启动项目。

步骤三：同步MySQL全量数据

执行以下命令调用Client-Adapter服务的方法触发同步任务。此时，canal会先中止增量数据传输，然后同步全量数据。待全量数据同步完成后，canal会自动进行增量数据同步。

- 命令格式

```
curl "hostip:port/etl/type/key/task" -X POST
```

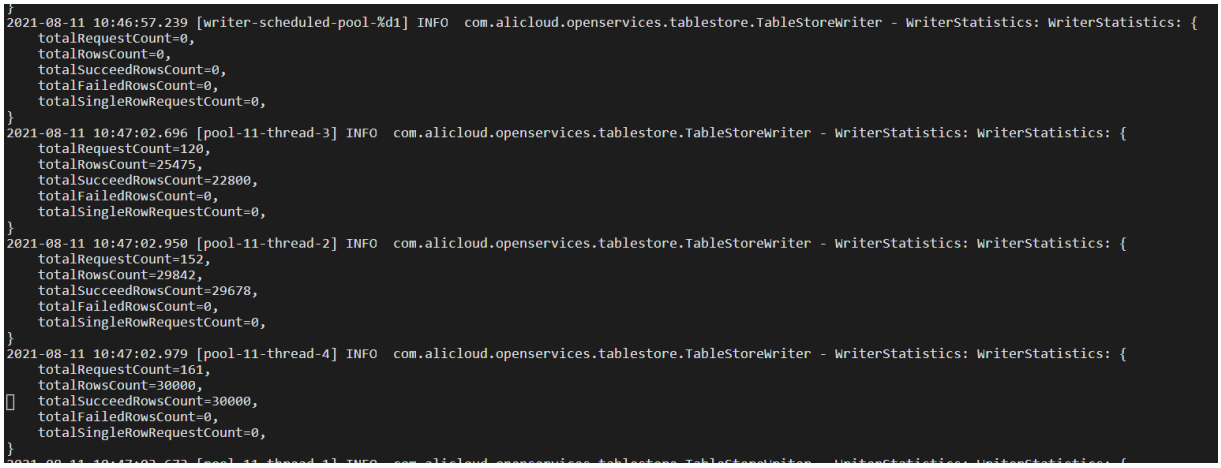
- 示例

```
curl "localhost:8081/etl/tablestore/ts/mytest.yml" -X POST
```

详细配置项说明请参见下表。

配置项	是否必选	示例	描述
hostip	是	localhost	部署canal服务的机器IP地址和端口。 当在部署canal服务的机器上执行此命令时，可设置hostip为localhost。
port	是	8081	
type	是	tablestore	下游数据库类型，必须设置为tablestore。
key	是	ts	使用的Adapter标识，必须与application.yml中canal.conf: canalAdapters: groups: outerAdapters下key值相同。
task	是	mytest.yml	任务配置文件的名称，必须与步骤二：部署Client-Adapter中创建的.yml格式的文件名称相同。

全量数据同步开始后，您可以在日志中查看Adapter中TablestoreWriter的传输日志变化，如下图所示。



源表和目标Tablestore表中字段映射

canal支持源表到目标Tablestore表的字段名称以及字段类型映射，对应于dbMapping.targetColumns字段中的映射配置。支持作为目标类型配置在$后面的字段类型请参见下表。

源表中字段类型	目标Tablestore表中字段类型
string	string
int	int
integer	
bool	bool
boolean	
binary	binary
double	double
float	
decimal	

2.2. 将Kafka数据同步到表格存储

2.2.1. 概述

基于Tablestore Sink Connector，您可以将Apache Kafka中的数据批量导入到表格存储（Tablestore）的数据表或者时序表中。

背景信息

Kafka是一个分布式消息队列系统，不同的数据系统可以通过Kafka Connect工具将数据流输入Kafka和从Kafka获取数据流。

表格存储团队基于Kafka Connect开发了Tablestore Sink Connector。Tablestore Sink Connector会根据订阅的主题（Topic）轮询地从Kafka中拉取消息记录（Record），并对消息记录进行解析，然后将数据批量导入到Tablestore。该Connector优化了数据导入过程，并且支持个性化配置。

表格存储是阿里云自研的多模型结构化数据存储，支持多种数据模型，包括宽表模型和时序模型。您可以将Kafka数据同步到表格存储中的数据表（宽表模型中的表类型）或者时序表（时序模型中的表类型）。具体操作，请分别参见同步数据到数据表和同步数据到时序表。

功能特性

Tablestore Sink Connector的主要功能特性如下：

- 至少交付一次

  保证Kafka消息记录从Kafka主题向Tablestore至少交付一次。

- 数据映射

  Kafka主题中的数据先通过Converter进行反序列化，您需要在Kafka Connect的worker配置或者connetor配置中修改key.converter和value.converter属性，以确保配置合适的反序列化转换器。您可以选择Kafka Connect带有的JsonConverter，也可以选择由第三方提供的其它Converter或者自定义Converter。

- 自动创建目标表

当目标表缺失时，支持根据配置的主键列和属性列白名单（如果有）自动创建目标表。

- 错误处理策略

  由于导入数据时为批量操作，其中部分消息记录可能发生解析错误或者写入错误。此时，您可以选择立即终止任务或者忽略这些错误，您还可以选择将产生错误的消息记录和错误信息记录在Kafka消息系统中或者表格存储中。

工作模式

Tablestore Sink Connector具有standalone（独立）模式和distributed（分布式）模式两种工作模式，请根据实际需要选择。

- 在standalone模式下，所有任务都将在单个进程中执行，此模式更易于配置和使用。您可以使用standalone模式了解Tablestore Sink Connector的各种功能。
- 在distributed模式下，所有任务通过多个进程并行执行，此模式支持根据进程变化自动均衡任务以及在执行任务过程中提供容错能力，稳定性更好。建议您使用distributed模式。

2.2.2. 同步数据到数据表

Tablestore Sink Connector会根据订阅的主题轮询地从Kafka中拉取消息，并对消息记录进行解析，然后将数据批量导入到Tablestore的数据表。

前提条件

- 已安装Kafka，并且已启动ZooKeeper和Kafka。更多信息，请参见Kafka官方文档。
- 已开通表格存储服务，创建实例以及创建数据表。具体操作，请参见快速使用宽表模型。

  > **说明** 您也可以通过Tablestore Sink Connector自动创建目标数据表，此时需要配置auto.create为true。

- 已获取AccessKey。具体操作，请参见获取AccessKey。

步骤一：部署Tablestore Sink Connector

1. 通过以下任意一种方式获取Tablestore Sink Connector。
   - 通过GitHub下载源码并编译。源码的GitHub路径为Tablestore Sink Connector源码。
     a. 通过Git工具执行以下命令下载Tablestore Sink Connector源码。

     ```
     git clone https://github.com/aliyun/kafka-connect-tablestore.git
     ```

     b. 进入到下载的源码目录后，执行以下命令进行Maven打包。

     ```
     mvn clean package -DskipTests
     ```

     编译完成后，生成的压缩包（例如kafka-connect-tablestore-1.0.jar）会存放在target目录。
   - 直接下载编译完成的kafka-connect-tablestore压缩包。
2. 将压缩包复制到各个节点的$KAFKA_HOME/libs目录下。

步骤二：启动Tablestore Sink Connector

Tablestore Sink Connector具有standalone模式和distributed模式两种工作模式。请根据实际选择。

standalone模式的配置步骤如下：

1. 根据实际修改worker配置文件connect-standalone.properties和connetor配置文件connect-tablestore-sink-quickstart.properties。
   - worker配置文件connect-standalone.properties的配置示例

     worker配置中包括Kafka连接参数、序列化格式、提交偏移量的频率等配置项。此处以Kafka官方示例为例介绍。更多信息，请参见Kafka Connect。

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# These are defaults. This file just demonstrates how to override some settings.
bootstrap.servers=localhost:9092
# The converters specify the format of data in Kafka and how to translate it into Con
nect data. Every Connect user will
# need to configure these based on the format they want their data in when loaded fro
m or stored into Kafka
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
# Converter-specific settings can be passed in by prefixing the Converter's setting w
ith the converter we want to apply
# it to
key.converter.schemas.enable=true
value.converter.schemas.enable=true
offset.storage.file.filename=/tmp/connect.offsets
# Flush much faster than normal, which is useful for testing/debugging
offset.flush.interval.ms=10000
# Set to a list of filesystem paths separated by commas (,) to enable class loading i
solation for plugins
# (connectors, converters, transformations). The list should consist of top level dir
ectories that include
# any combination of:
# a) directories immediately containing jars with plugins and their dependencies
# b) uber-jars with plugins and their dependencies
# c) directories immediately containing the package directory structure of classes of
plugins and their dependencies
# Note: symlinks will be followed to discover dependencies or plugins.
# Examples:
# plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/connectors,
#plugin.path=
```

   - connetor配置文件connect-tablestore-sink-quickstart.properties的配置示例

connetor配置中包括连接器类、表格存储连接参数、数据映射等配置项。更多信息，请参见配置说明。

```
# 设置连接器名称。
name=tablestore-sink
# 指定连接器类。
connector.class=TableStoreSinkConnector
# 设置最大任务数。
tasks.max=1
# 指定导出数据的Kafka的Topic列表。
topics=test
# 以下为Tablestore连接参数配置。
# Tablestore实例的Endpoint。
tablestore.endpoint=https://xxx.xxx.ots.aliyuncs.com
# 填写AccessKey ID和AccessKey Secret。
tablestore.access.key.id =xxx
tablestore.access.key.secret=xxx
# Tablestore实例名称。
tablestore.instance.name=xxx
# 用于指定表格存储目标表名称的格式字符串，其中<topic>作为原始Topic的占位符。默认值为<topic>。
# Examples:
# table.name.format=kafka_<topic>，主题为test的消息记录将写入表名为kafka_test的数据表。
# table.name.format=
# 主键模式，默认值为kafka。
# 将以<topic>_<partition>（Kafka主题和分区，用"_"分隔）和<offset>（消息记录在分区中的偏移量）作为Tablestore数据表的主键。
# primarykey.mode=
# 自动创建目标表，默认值为false。
auto.create=true
```

2. 进入到$KAFKA_HOME目录后，执行以下命令启动standalone模式。

```
bin/connect-standalone.sh config/connect-standalone.properties config/connect-tablestore-sink-quickstart.properties
```

distributed模式的配置步骤如下：

1. 根据实际修改worker配置文件connect-distributed.properties。

   worker配置中包括Kafka连接参数、序列化格式、提交偏移量的频率等配置项，还包括存储各connectors相关信息的Topic，建议您提前手动创建相应Topic。此处以Kafka官方示例为例介绍。更多信息，请参见Kafka Connect。

   - offset.storage.topic：用于存储各connectors相关offset的Compact Topic。
   - config.storage.topic：用于存储connector和task相关配置的Compact Topic，此Topic的Parition数必须设置为1。
   - status.storage.topic：用于存储kafka connect状态信息的Compact Topic。

```
##
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
```

```
"
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
##
# This file contains some of the configurations for the Kafka Connect distributed worke
r. This file is intended
# to be used with the examples, and some settings may differ from those used in a produ
ction system, especially
# the `bootstrap.servers` and those specifying replication factors.
# A list of host/port pairs to use for establishing the initial connection to the Kafka
cluster.
bootstrap.servers=localhost:9092
# unique name for the cluster, used in forming the Connect cluster group. Note that thi
s must not conflict with consumer group IDs
group.id=connect-cluster
# The converters specify the format of data in Kafka and how to translate it into Conne
ct data. Every Connect user will
# need to configure these based on the format they want their data in when loaded from
or stored into Kafka
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
# Converter-specific settings can be passed in by prefixing the Converter's setting wit
h the converter we want to apply
# it to
key.converter.schemas.enable=true
value.converter.schemas.enable=true
# Topic to use for storing offsets. This topic should have many partitions and be repli
cated and compacted.
# Kafka Connect will attempt to create the topic automatically when needed, but you can
always manually create
# the topic before starting Kafka Connect if a specific topic configuration is needed.
# Most users will want to use the built-in default replication factor of 3 or in some c
ases even specify a larger value.
# Since this means there must be at least as many brokers as the maximum replication fa
ctor used, we'd like to be able
# to run this example on a single-broker cluster and so here we instead set the replica
tion factor to 1.
offset.storage.topic=connect-offsets
offset.storage.replication.factor=1
#offset.storage.partitions=25
# Topic to use for storing connector and task configurations; note that this should be
a single partition, highly replicated,
# and compacted topic. Kafka Connect will attempt to create the topic automatically whe
n needed, but you can always manually create
# the topic before starting Kafka Connect if a specific topic configuration is needed.
# Most users will want to use the built-in default replication factor of 3 or in some c
ases even specify a larger value.
# Since this means there must be at least as many brokers as the maximum replication fa
ctor used, we'd like to be able
# to run this example on a single-broker cluster and so here we instead set the replica
```

```
tion factor to 1.
config.storage.topic=connect-configs
config.storage.replication.factor=1
# Topic to use for storing statuses. This topic can have multiple partitions and should
be replicated and compacted.
# Kafka Connect will attempt to create the topic automatically when needed, but you can
always manually create
# the topic before starting Kafka Connect if a specific topic configuration is needed.
# Most users will want to use the built-in default replication factor of 3 or in some c
ases even specify a larger value.
# Since this means there must be at least as many brokers as the maximum replication fa
ctor used, we'd like to be able
# to run this example on a single-broker cluster and so here we instead set the replica
tion factor to 1.
status.storage.topic=connect-status
status.storage.replication.factor=1
#status.storage.partitions=5
# Flush much faster than normal, which is useful for testing/debugging
offset.flush.interval.ms=10000
# These are provided to inform the user about the presence of the REST host and port co
nfigs
# Hostname & Port for the REST API to listen on. If this is set, it will bind to the in
terface used to listen to requests.
#rest.host.name=
#rest.port=8083
# The Hostname & Port that will be given out to other workers to connect to i.e. URLs t
hat are routable from other servers.
#rest.advertised.host.name=
#rest.advertised.port=
# Set to a list of filesystem paths separated by commas (,) to enable class loading iso
lation for plugins
# (connectors, converters, transformations). The list should consist of top level direc
tories that include
# any combination of:
# a) directories immediately containing jars with plugins and their dependencies
# b) uber-jars with plugins and their dependencies
# c) directories immediately containing the package directory structure of classes of p
lugins and their dependencies
# Examples:
# plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/connectors,
#plugin.path=
```

2. 进入到$KAFKA_HOME目录后，执行以下命令启动distributed模式。

   > **注意** 您需要在每个节点上均启动worker进程。

   ```
   bin/connect-distributed.sh config/connect-distributed.properties
   ```

3. 通过REST API管理connectors。更多信息，请参见REST API。

i. 在config路径下创建connect-tablestore-sink-quickstart.json文件并填写以下示例内容。

connetor配置文件以JSON格式串指定参数键值对，包括连接器类、表格存储连接参数、数据映射等配置项。更多信息，请参见配置说明。

```
{
  "name": "tablestore-sink",
  "config": {
    "connector.class":"TableStoreSinkConnector",
    "tasks.max":"1",
    "topics":"test",
    "tablestore.endpoint":"https://xxx.xxx.ots.aliyuncs.com",
    "tablestore.access.key.id":"xxx",
    "tablestore.access.key.secret":"xxx",
    "tablestore.instance.name":"xxx",
    "table.name.format":"<topic>",
    "primarykey.mode":"kafka",
    "auto.create":"true"
  }
}
```

ii. 执行以下命令启动一个Tablestore Sink Connector。

```
curl -i -k  -H "Content-type: application/json" -X POST -d @config/connect-tablesto
re-sink-quickstart.json http://localhost:8083/connectors
```

其中 `http://localhost:8083/connectors` 为Kafka REST服务的地址，请根据实际修改。

步骤三：生产新的记录

1. 进入到$KAFKA_HOME目录后，执行以下命令启动一个控制台生产者。

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
```

配置项说明请参见下表。

配置项	示例值	描述
--broker-list	localhost:9092	Kafka集群broker地址和端口。
--topic	test	主题名称。启动Tablestore Sink Connetor时默认会自动创建Topic，您也可以选择手动创建。

2. 向主题test中写入一些新的消息。
   - Struct类型消息

```
{
    "schema":{
        "type":"struct",
        "fields":[
            {
                "type":"int32",
                "optional":false,
                "field":"id"
            },
            {
                "type":"string",
                "optional":false,
                "field":"product"
            },
            {
                "type":"int64",
                "optional":false,
                "field":"quantity"
            },
            {
                "type":"double",
                "optional":false,
                "field":"price"
            }
        ],
        "optional":false,
        "name":"record"
    },
    "payload":{
        "id":1,
        "product":"foo",
        "quantity":100,
        "price":50
    }
}
```

- Map类型消息

```
{
    "schema":{
        "type":"map",
        "keys":{
            "type":"string",
            "optional":false
        },
        "values":{
            "type":"int32",
            "optional":false
        },
        "optional":false
    },
    "payload":{
        "id":1
    }
}
```

3. 登录表格存储控制台查看数据。

   表格存储实例中将自动创建一张数据表，表名为test，表中数据如下图所示。其中第一行数据为Map类型消息导入结果，第二行数据为Struct类型消息导入结果。

	数据详情	topic_partition(主键)	offset(主键)	id	price	product	quantity
	数据详情	test_3	0	AAAAAQ==			
	数据详情	test_34	0	1	50.0	foo	100

2.2.3. 同步数据到时序表

您可以使用kafka-connect-tablestore包将Kafka中数据写入Tablestore的时序表中。本文主要介绍了如何配置Kafka写入时序数据。

前提条件

- 已安装Kafka，并且已启动ZooKeeper和Kafka。更多信息，请参见Kafka官方文档。
- 已开通表格存储服务，创建实例以及创建时序表。具体操作，请参见快速使用时序模型。

  > **说明** 您也可以通过Tablestore Sink Connector自动创建目标时序表，此时需要配置auto.create为true。

- 已获取AccessKey。具体操作，请参见获取AccessKey。

背景信息

表格存储支持对时序数据进行存储以及分析。更多信息，请参见时序模型概述。

步骤一：部署Tablestore Sink Connector

1. 通过以下任意一种方式获取Tablestore Sink Connector。
   - 通过GitHub下载源码并编译。源码的GitHub路径为Tablestore Sink Connector源码。
     a. 通过Git工具执行以下命令下载Tablestore Sink Connector源码。

     ```
     git clone https://github.com/aliyun/kafka-connect-tablestore.git
     ```

b. 进入到下载的源码目录后，执行以下命令进行Maven打包。

```
mvn clean package -DskipTests
```

编译完成后，生成的压缩包（例如kafka-connect-tablestore-1.0.jar）会存放在target目录。

- 直接下载编译完成的kafka-connect-tablestore压缩包。

2. 将压缩包复制到各个节点的$KAFKA_HOME/libs目录下。

步骤二：启动Tablestore Sink Connector

Tablestore Sink Connector具有standalone模式和distributed模式两种工作模式。请根据实际选择。

由于写入时序数据时，Kafka侧的消息记录必须为JSON格式，因此启动Tablestore Sink Connector时需要使用Jsonconverter，且不需要提取schema以及不需要输入key，请在connect-standalone.properties和connect-distributed.properties中按照如下示例配置对应配置项。

> **说明** 如果输入了key，请按照key的格式配置key.converter和key.converter.schemas.enable。

```
value.converter=org.apache.kafka.connect.json.JsonConverter
value.converter.schemas.enable=false
```

此处以配置standalone模式为例介绍，distributed模式的配置步骤与同步数据到数据表时的distributed模式配置步骤类似，只需按照上述示例在worker配置文件connect-distributed.properties中修改对应配置项以及在connetor文件connect-tablestore-sink-quickstart.json中修改时序相关配置即可。具体操作，请参见步骤二：启动Tablestore Sink Connector中distributed模式的配置步骤。

standalone模式的配置步骤如下：

1. 根据实际修改worker配置文件connect-standalone.properties和connetor配置文件connect-tablestore-sink-quickstart.properties。
   - worker配置文件connect-standalone.properties的配置示例

     worker配置中包括Kafka连接参数、序列化格式、提交偏移量的频率等配置项。此处以Kafka官方示例为例介绍。更多信息，请参见Kafka Connect。

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# These are defaults. This file just demonstrates how to override some settings.
bootstrap.servers=localhost:9092
# The converters specify the format of data in Kafka and how to translate it into Con
nect data. Every Connect user will
# need to configure these based on the format they want their data in when loaded fro
m or stored into Kafka
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
# Converter-specific settings can be passed in by prefixing the Converter's setting w
ith the converter we want to apply
# it to
key.converter.schemas.enable=true
value.converter.schemas.enable=false
offset.storage.file.filename=/tmp/connect.offsets
# Flush much faster than normal, which is useful for testing/debugging
offset.flush.interval.ms=10000
# Set to a list of filesystem paths separated by commas (,) to enable class loading i
solation for plugins
# (connectors, converters, transformations). The list should consist of top level dir
ectories that include
# any combination of:
# a) directories immediately containing jars with plugins and their dependencies
# b) uber-jars with plugins and their dependencies
# c) directories immediately containing the package directory structure of classes of
plugins and their dependencies
# Note: symlinks will be followed to discover dependencies or plugins.
# Examples:
# plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/connectors,
#plugin.path=
```

- connetor配置文件connect-tablestore-sink-quickstart.properties的配置示例

  connetor配置中包括连接器类、表格存储连接参数、数据映射等配置项。更多信息，请参见配置说明。

```
# 设置连接器名称。
name=tablestore-sink
# 指定连接器类。
connector.class=TableStoreSinkConnector
# 设置最大任务数。
tasks.max=1
```

```
tasks.max=1
# 指定导出数据的Kafka的Topic列表。
topics=test
# 以下为Tablestore连接参数的配置。
# Tablestore实例的Endpoint。
tablestore.endpoint=https://xxx.xxx.ots.aliyuncs.com
# 指定认证模式。
tablestore.auth.mode=aksk
# 填写AccessKey ID和AccessKey Secret。如果使用aksk认证，则需要填入这两项。
tablestore.access.key.id=xxx
tablestore.access.key.secret=xxx
# 指定Tablestore实例名称。
tablestore.instance.name=xxx
## STS认证相关配置，如果使用STS认证，则下列各项必填。此外aksk还需要在环境变量中配置配入ACCESS_ID和ACCESS_KEY。
#sts.endpoint=
#region=
#account.id=
#role.name=
# 定义目标表名称的格式字符串，字符串中可包含<topic>作为原始Topic的占位符。
# topics.assign.tables配置的优先级更高，如果配置了topics.assign.tables，则忽略table.name.format的配置。
# 例如当设置table.name.format为kafka_<topic>时，如果kafka中主题名称为test，则将映射到Tablestore的表名为kafka_test。
table.name.format=<topic>
# 指定Topic与目标表的映射关系，以"<topic>:<tablename>"格式映射Topic和表名，Topic和表名之间的分隔符为半角冒号（:），不同映射之间分隔符为半角逗号（,）。
# 如果缺省，则采取table.name.format的配置。
# topics.assign.tables=test:test_kafka
# 是否自动创建目标表，默认值为false。
auto.create=true
# 以下为脏数据处理相关配置。
# 在解析Kafka Record或者写入时序表时可能发生错误，您可以可通过以下配置进行处理。
# 指定容错能力，可选值包括none和all，默认值为none。
# none表示任何错误都将导致Sink Task立即失败。
# all表示跳过产生错误的Record，并记录该Record。
runtime.error.tolerance=none
# 指定脏数据记录模式，可选值包括ignore、kafka和tablestore，默认值为ignore。
# ignore表示忽略所有错误。
# kafka表示将产生错误的Record和错误信息存储在Kafka的另一个Topic中。
# tablestore表示将产生错误的Record和错误信息存储在Tablestore另一张数据表中。
runtime.error.mode=ignore
# 当脏数据记录模式为kafka时，需要配置Kafka集群地址和Topic。
# runtime.error.bootstrap.servers=localhost:9092
# runtime.error.topic.name=errors
# 当脏数据记录模式为tablestore时，需要配置Tablestore中数据表名称。
# runtime.error.table.name=errors
##以下为时序表新增配置。
# connector工作模式，默认为normal。
tablestore.mode=timeseries
# 时序表主键字段映射。
tablestore.timeseries.test.measurement=m
tablestore.timeseries.test.dataSource=d
tablestore.timeseries.test.tags=region,level
# 时序表时间字段映射。
```

```
tablestore.timeseries.test.time=timestamp
tablestore.timeseries.test.time.unit=MILLISECONDS
# 是否将时序数据字段（field）的列名转为小写，默认为true。由于当前时序模型中时序表的列名不支持大写字母，如果配置为false，且列名中有大写字母，写入会报错。
tablestore.timeseries.toLowerCase=true
# 是否将所有非主键以及时间的字段以field的形式存储在时序表，默认为true，如果为false，则只存储tablestore.timeseries.test.field.name中配置的字段
tablestore.timeseries.mapAll=true
# 配置field字段名称，多个字段名称之间用半角冒号（,）分隔。
tablestore.timeseries.test.field.name=cpu
# 配置field字段类型。取值范围为double、integer、string、binary和boolean。
# 当field中包含多个字段时，字段类型必须和字段名称一一对应。多个字段类型之间用半角冒号（,）分隔。
tablestore.timeseries.test.field.type=double
```

2. 进入到$KAFKA_HOME目录后，执行以下命令启动standalone模式。

```
bin/connect-standalone.sh config/connect-standalone.properties config/connect-tablestore-sink-quickstart.properties
```

步骤三：生产新的记录

1. 进入到$KAFKA_HOME目录后，执行以下命令启动一个控制台生产者。

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
```

配置项说明请参见下表。

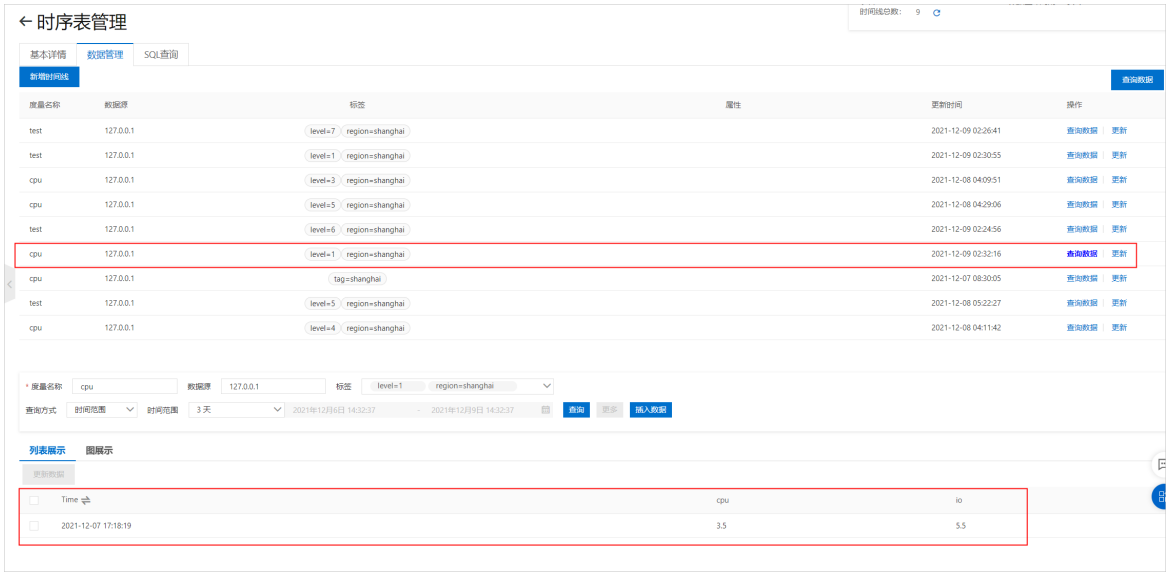
配置项	示例值	描述
--broker-list	localhost:9092	Kafka集群broker地址和端口。
--topic	test	主题名称。启动Tablestore Sink Connetor时默认会自动创建Topic，您也可以选择手动创建。

2. 向主题test中写入一些新的消息。

> 注意 如果要导入数据到时序表，则向主题中写入数据时必须输入JSON格式的数据。

```
{"m":"cpu","d":"127.0.0.1","region":"shanghai","level":1,"timestamp":1638868699090,"io":5.5,"cpu":"3.5"}
```

3. 登录表格存储控制台查看数据。



2.2.4. 配置说明

启动Tablestore Sink Connector时，您需要通过键值映射向Kafka Connect进程传递参数。通过本文您可以结合配置示例和配置参数说明了解Tablestore Sink Connector的相关配置。

配置示例

当从Kafka同步数据到数据表或者时序表时配置项不同，且不同工作模式下相应配置文件的示例不同。此处以同步数据到数据表中为例介绍配置示例。同步数据到时序表的配置示例中需要增加时序相关配置项。

- .properties格式配置文件的示例，适用于standalone模式。

```
# 设置连接器名称。
name=tablestore-sink
# 指定连接器类。
connector.class=TableStoreSinkConnector
# 设置最大任务数。
tasks.max=1
# 指定导出数据的Kafka的Topic列表。
topics=test
# 以下为Tablestore连接参数的配置。
# Tablestore实例的Endpoint。
tablestore.endpoint=https://xxx.xxx.ots.aliyuncs.com
# 填写AccessKey ID和AccessKey Secret。
tablestore.access.key.id=xxx
tablestore.access.key.secret=xxx
# Tablestore实例名称。
tablestore.instance.name=xxx
# 以下为数据映射相关的配置。
# 指定Kafka Record的解析器。
# 默认的DefaulteEventParser已支持Struct和Map类型，您也可以使用自定义的EventParser。
event.parse.class=com.aliyun.tablestore.kafka.connect.parsers.DefaultEventParser
# 定义目标表名称的格式字符串，字符串中可包含<topic>作为原始Topic的占位符。
# topics.assign.tables配置的优先级更高，如果配置了topics.assign.tables，则忽略table.name.format的配置。
# 例如当设置table.name.format为kafka_<topic>时，如果kafka中主题名称为test，则将映射到Tablestore的表名为kafka_test。
```

```
table.name.format=<topic>
# 指定Topic与目标表的映射关系，以"<topic>:<tablename>"格式映射Topic和表名，Topic和表名之间的分隔符为英文冒号（:），不同映射之间分隔符为英文逗号（,）。
# 如果缺省，则采取table.name.format的配置。
# topics.assign.tables=test:test_kafka
# 指定主键模式，可选值包括kafka、record_key和record_value，默认值为kafka。
# kafka表示以<connect_topic>_<connect_partition>和<connect_offset>作为数据表的主键。
# record_key表示以Record Key中的字段作为数据表的主键。
# record_value表示以Record Value中的字段作为数据表的主键。
primarykey.mode=kafka
# 定义导入数据表的主键列名和数据类型。
# 属性名格式为tablestore.<tablename>.primarykey.name和tablestore.<tablename>.primarykey.type。
# 其中<tablename>为数据表名称的占位符。
# 当主键模式为kafka时，无需配置该属性，默认主键列名为{"topic_partition","offset"}，默认主键列数据类型为{string, integer}。
# 当主键模式为record_key或record_value时，必须配置以下两个属性。
# tablestore.test.primarykey.name=A,B
# tablestore.test.primarykey.type=string,integer
# 定义属性列白名单，用于过滤Record Value中的字段获取所需属性列。
# 默认值为空，使用Record Value中的所有字段作为数据表的属性列。
# 属性名格式为tablestore.<tablename>.columns.whitelist.name和tablestore.<tablename>.columns.whitelist.type。
# 其中<tablename>为数据表名称的占位符。
# tablestore.test.columns.whitelist.name=A,B
# tablestore.test.columns.whitelist.type=string,integer
# 以下为写入Tablestore相关的配置。
# 指定写入模式，可选值包括put和update，默认值为put。
# put表示覆盖写。
# update表示更新写。
insert.mode=put
# 是否需要保序，默认值为true。如果关闭保序模式，则有助于提升写入效率。
insert.order.enable=true
# 是否自动创建目标表，默认值为false。
auto.create=false
# 指定删除模式，可选值包括none、row、column和row_and_column，默认值为none。
# none表示不允许进行任何删除。
# row表示允许删除行。
# column表示允许删除属性列。
# row_and_column表示允许删除行和属性列。
delete.mode=none
# 写入数据表时内存中缓冲队列的大小，默认值为1024，单位为行数。此配置项的值必须为2的指数。
buffer.size=1024
# 写入数据表时的回调线程数，默认值为核数+1。
# max.thread.count=
# 写入数据表时的最大请求并发数，默认值为10。
max.concurrency=10
# 写入数据表时的分桶数，默认值为3。适当调大此配置项的值可提升并发写入能力，但不应大于最大请求并发数。
bucket.count=3
# 写入数据表时对缓冲区的刷新时间间隔，默认值为10000，单位为毫秒。
flush.Interval=10000
# 以下为脏数据处理相关配置。
# 在解析Kafka Record或者写入数据表时可能发生错误，您可以可通过以下配置进行处理。
# 指定容错能力，可选值包括none和all，默认值为none。
```

```
# none表示任何错误都将导致Sink Task立即失败。
# all表示跳过产生错误的Record，并记录该Record。
runtime.error.tolerance=none
# 指定脏数据记录模式，可选值包括ignore、kafka和tablestore，默认值为ignore。
# ignore表示忽略所有错误。
# kafka表示将产生错误的Record和错误信息存储在Kafka的另一个Topic中。
# tablestore表示将产生错误的Record和错误信息存储在Tablestore另一张数据表中。
runtime.error.mode=ignore
# 当脏数据记录模式为kafka时，需要配置Kafka集群地址和Topic。
# runtime.error.bootstrap.servers=localhost:9092
# runtime.error.topic.name=errors
# 当脏数据记录模式为tablestore时，需要配置Tablestore中数据表名称。
# runtime.error.table.name=errors
```

- .json格式配置文件的示例，适用于distributed模式。

```
{
  "name": "tablestore-sink",
  "config": {
    // 指定连接器类。
    "connector.class":"TableStoreSinkConnector",
    // 设置最大任务数。
    "tasks.max":"3",
    // 指定导出数据的Kafka的Topic列表。
    "topics":"test",
    // 以下为Tablestore连接参数的配置。
    // Tablestore实例的Endpoint。
    "tablestore.endpoint":"https://xxx.xxx.ots.aliyuncs.com",
    // 填写AccessKey ID和AccessKey Secret。
    "tablestore.access.key.id":"xxx",
    "tablestore.access.key.secret":"xxx",
    // Tablestore实例名称。
    "tablestore.instance.name":"xxx",
    // 以下为数据映射相关的配置。
    // 指定Kafka Record的解析器。
    // 默认的DefaulteEventParser已支持Struct和Map类型，您也可以使用自定义的EventParser。
    "event.parse.class":"com.aliyun.tablestore.kafka.connect.parsers.DefaultEventParser",
    // 定义目标表名称的格式字符串，字符串中可包含<topic>作为原始Topic的占位符。
    // topics.assign.tables配置的优先级更高。如果配置了topics.assign.tables，则忽略table.name.format的配置。
    // 例如当设置table.name.format为kafka_<topic>时，如果kafka中主题名称为test，则将映射到Tablestore的表名为kafka_test。
    "table.name.format":"<topic>",
    // 指定Topic与目标表的映射关系，以"<topic>:<tablename>"格式映射Topic和表名，Topic和表名之间的分隔符为英文冒号（:），不同映射之间分隔符为英文逗号（,）。
    // 如果缺省，则采取table.name.format的配置。
    // "topics.assign.tables":"test:test_kafka",
    // 指定主键模式，可选值包括kafka、record_key和record_value，默认值为kafka。
    // kafka表示以<connect_topic>_<connect_partition>和<connect_offset>作为数据表的主键。
    // record_key表示以Record Key中的字段作为数据表的主键。
    // record_value表示以Record Value中的字段作为数据表的主键。
    "primarykey.mode":"kafka",
    // 定义导入数据表的主键列名和数据类型。
    // 属性名格式为tablestore.<tablename>.primarykey.name和tablestore.<tablename>.primarykey.type。
```

```
    // 其中<tablename>为数据表名称的占位符。
    // 当主键模式为kafka时，无需配置该属性，默认主键列名为{"topic_partition","offset"}，默认主键列数据类型为{string, integer}。
    // 当主键模式为record_key或record_value时，必须配置以下两个属性。
    // "tablestore.test.primarykey.name":"A,B",
    // "tablestore.test.primarykey.type":"string,integer",
    // 定义属性列白名单，用于过滤Record Value中的字段获取所需属性列。
    // 默认值为空，使用Record Value中的所有字段作为数据表的属性列。
    // 属性名格式为tablestore.<tablename>.columns.whitelist.name和tablestore.<tablename>.columns.whitelist.type。
    // 其中<tablename>为数据表名称的占位符。
    // "tablestore.test.columns.whitelist.name":"A,B",
    // "tablestore.test.columns.whitelist.type":"string,integer",
    // 以下为写入Tablestore相关的配置。
    // 指定写入模式，可选值包括put和update，默认值为put。
    // put表示覆盖写。
    // update表示更新写。
    "insert.mode":"put",
    // 是否需要保序，默认值为true。如果关闭保序模式，则有助于提升写入效率。
    "insert.order.enable":"true",
    // 是否自动创建目标表，默认值为false。
    "auto.create":"false",
    // 指定删除模式，可选值包括none、row、column和row_and_column，默认值为none。
    // none表示不允许进行任何删除。
    // row表示允许删除行。
    // column表示允许删除属性列。
    // row_and_column表示允许删除行和属性列。
    "delete.mode":"none",
    // 写入数据表时内存中缓冲队列的大小，默认值为1024，单位为行数。此配置项的值必须为2的指数。
    "buffer.size":"1024",
    // 写入数据表时的回调线程数，默认值为核数+1。
    // "max.thread.count":
    // 写入数据表时的最大请求并发数，默认值为10。
    "max.concurrency":"10",
    // 写入数据表时的分桶数，默认值为3。适当调大此配置项的值可提升并发写入能力，但不应大于最大请求并发数。
    "bucket.count":"3",
    // 写入数据表时对缓冲区的刷新时间间隔，默认值为10000，单位为毫秒。
    "flush.Interval":"10000",
    // 以下为脏数据处理相关配置。
    // 在解析Kafka Record或者写入数据表时可能发生错误，您可以通过以下配置进行处理。
    // 指定容错能力，可选值包括none和all，默认值为none。
    // none表示任何错误都将导致Sink Task立即失败。
    // all表示跳过产生错误的Record，并记录该Record。
    "runtime.error.tolerance":"none",
    // 指定脏数据记录模式，可选值包括ignore、kafka和tablestore，默认值为ignore。
    // ignore表示忽略所有错误。
    // kafka表示将产生错误的Record和错误信息存储在Kafka的另一个Topic中。
    // tablestore表示将产生错误的Record和错误信息存储在Tablestore另一张数据表中。
    "runtime.error.mode":"ignore"
    // 当脏数据记录模式为kafka时，需要配置Kafka集群地址和Topic。
    // "runtime.error.bootstrap.servers":"localhost:9092",
    // "runtime.error.topic.name":"errors",
    // 当脏数据记录模式为tablestore时，需要配置Tablestore中数据表名称。
```

```
    // "runtime.error.table.name":"errors",
}
```

配置项说明

配置文件中的配置项说明请参见下表。其中时序相关配置项只有同步数据到时序表时才需要配置。

分类	配置项	类型	是否必选	示例值	描述
Kafka Connect常见配置	name	string	是	tablestore-sink	连接器（Connector）名称。连接器名称必须唯一。
	connector.class	class	是	TableStoreSinkConnector	连接器的Java类。 如果您要使用该连接器，请在connector.class配置项中指定Connector类的名称，支持配置为Connector类的全名（com.aliyun.tablestore.kafka.connect.TableStoreSinkConnector）或别名（TableStoreSinkConnector）。 `connector.class=com.aliyun.tablestore.kafka.connect.TableStoreSinkConnector`
	tasks.max	integer	是	3	连接器支持创建的最大任务数。 如果连接器无法达到此并行度级别，则可能会创建较少的任务。
	key.converter	string	否	org.apache.kafka.connect.json.JsonConverter	覆盖worker设置的默认key转换器。
	value.converter	string	否	org.apache.kafka.connect.json.JsonConverter	覆盖worker设置的默认value转换器。
	topics	list	是	test	连接器输入的Kafka Topic列表，多个Topic之间以英文逗号（,）分隔。 您必须为连接器设置topics来控制连接器输入的Topic。

分类	配置项	类型	是否必选	示例值	描述
连接器Connection配置	tablestore.endpoint	string	是	https://xxx.xxx.ots.aliyuncs.com	Tablestore实例的服务地址。更多信息，请参见服务地址。
	tablestore.mode	string	是	timeseries	根据数据同步到的表类型选择模式。取值范围如下： • normal（默认）：同步数据到表格存储的数据表。 • timeseries：同步数据到表格存储的时序表。
	tablestore.access.key.id	string	是	LTAn**********	登录账号的AccessKey ID和AccessKey Secret，获取方式请参见获取AccessKey。
	tablestore.access.key.secret	string	是	zbnK**********	
	tablestore.auth.mode	string	是	aksk	设置认证方式。取值范围如下： • aksk（默认）：使用阿里云账号或者RAM用户的AccessKey ID和Access Secret进行认证。请使用此认证方式。 • sts：使用STS临时访问凭证进行认证。对接云kafka时使用。
	tablestore.instance.name	string	是	myotstest	Tablestore实例的名称。

分类	配置项	类型	是否必选	示例值	描述
	event.parse.class	class	是	DefaultEventParser	消息解析器的Java类，默认值为DefaultEventParser。解析器用于从Kafka Record中解析出数据表的主键列和属性列。 **注意** Tablestore对列值大小有限制。string类型和binary类型的主键列列值限制均为1 KB，属性列列值限制均为2 MB。更多信息，请参见通用限制。 如果数据类型转换后列值超出对应限制，则将该Kafka Record作为脏数据处理。 如果使用默认的DefaultEventParser解析器，Kafka Record的Key或Value必须为Kafka Connect的Struct或Map类型。Struct中选择的字段必须为支持的数据类型，字段会根据数据类型映射表转换为Tablestore数据类型写入数据表。Map中的值类型也必须为支持的数据类型，支持的数据类型同Struct，最终会被转换为binary类型写入数据表。Kafka和Tablestore的数据类型映射关系请参见附录：Kafka和Tablestore数据类型映射。 如果Kafka Record为不兼容的数据格式，则您可以通过实现com.aliyun.tablestore.kafka.connect.parsers.EventParser定义的接口来自定义解析器。

分类	配置项	类型	是否必选	示例值	描述
	table.name.format	string	否	kafka_<topic>	目标数据表名称的格式字符串，默认值为<topic>。字符串中可包含<topic>作为原始Topic的占位符。例如当设置table.name.format为kafka_<topic>时，如果kafka中主题名称为test，则映射到Tablestore的表名为kafka_test。 此配置项的优先级低于topics.assign.tables配置项，如果配置了topics.assign.tables，则忽略table.name.format的配置。
	topics.assign.tables	list	是	test:destTable	指定topic与Tablestore表之间的映射关系，格式为 `<topic_1>:<tablename_1>,<topic_2>:<tablename_2>` 。多个映射关系之间以英文逗号（,）分隔，例如test:destTable表示将Topic名为test的消息记录写入数据表destTable中。 此配置项的优先级高于table.name.format配置项，如果配置了topics.assign.tables，则忽略table.name.format的配置。

分类	配置项	类型	是否必选	示例值	描述
连接器Data Mapping配置	primarykey.mode	string	否	kafka	数据表的主键模式。取值范围如下： • kafka：以<connect_topic>_<connect_partition>（Kafka主题和分区，用下划线"_"分隔）和<connect_offset>（该消息记录在分区中的偏移量）作为数据表的主键。 • record_key：以Record Key中的字段（Struct类型）或者键（Map类型）作为数据表的主键。 • record_value：以Record Value中的字段（Struct类型）或者键（Map类型）作为数据表的主键。 请配合tablestore.<tablename>.primarykey.name和tablestore.<tablename>.primarykey.type使用。此配置项不区分大小写。

分类	配置项	类型	是否必选	示例值	描述
	tablestore.<tablename>.primarykey.name	list	否	A,B	数据表的主键列名，其中<tablename>为数据表名称的占位符，包含1~4个主键列，以英文逗号（,）分隔。 主键模式不同时，主键列名的配置不同。 • 当设置主键模式为kafka时，以 `topic_partition,offset` 作为数据表主键列名称。在该主键模式下，您可以不配置此主键列名。如果配置了主键列名，则不会覆盖默认主键列名。 • 当设置主键模式为record_key时，从Record Key中提取与配置的主键列名相同的字段（Struct类型）或者键（Map类型）作为数据表的主键。在该主键模式下主键列名必须配置。 • 当设置主键模式为record_value时，从Record Value中提取与配置的主键列名相同的字段（Struct类型）或者键（Map类型）作为数据表的主键。在该主键模式下主键列名必须配置。 Tablestore数据表的主键列是有顺序的，此属性的配置应注意主键列名顺序，例如PRIMARY KEY（A, B, C）与PRIMARY KEY（A, C, B）是不同的两个主键结构。

分类	配置项	类型	是否必选	示例值	描述
	tablestore.<tablename>.primarykey.type	list	否	string,integer	数据表的主键列数据类型，其中<tablename>为数据表名称的占位符，包含1~4个主键列，以英文逗号（,）分隔，顺序必须与tablestore.<tablename>.primarykey.name相对应。此属性配置不区分大小写。数据类型的可选值包括integer、string、binary和auto_increment。 主键模式不同时，主键数据类型的配置不同。 • 当主键模式为kafka时，以 `string, integer` 作为数据表主键数据类型。 在该主键模式下，您可以不配置此主键列数据类型。如果配置了主键列数据类型，则不会覆盖默认主键列数据类型。 • 当主键模式为record_key或者record_value时，指定相应主键列的数据类型。在该主键模式下主键列数据类型必须配置。 如果指定的数据类型与Kafka Schema中定义的数据类型发生冲突，则会产生解析错误，您可以配置Runtime Error相关属性来应对解析错误。 当配置此配置项为auto_increment时，表示主键自增列，此时Kafka Record中可以缺省该字段，写入数据表时会自动插入自增列。
	tablestore.<tablename>.columns.whitelist.name	list	否	A,B	数据表的属性列白名单中属性列名称，其中<tablename>为数据表名称的占位符，以英文逗号（,）分隔。 如果配置为空，则使用Record Value中的所有字段（Struct类型）或者键（Map类型）作为数据表的属性列，否则用于过滤得到所需属性列。

分类	配置项	类型	是否必选	示例值	描述
	tablestore.<tablename>.columns.whitelist.type	list	否	string,integer	数据表的属性列白名单中属性列数据类型，其中<tablename>为数据表名称的占位符，以英文逗号（,）分隔，顺序必须与tablestore.<tablename>.columns.whitelist.name相对应。此属性配置不区分大小写。数据类型的可选值包括integer、string、binary、boolean和double。
	insert.mode	string	否	put	写入模式。取值范围如下： • put（默认）：对应Tablestore的PutRow操作，即新写入一行数据会覆盖原数据。 • update：对应Tablestore的UpdateRow操作，即更新一行数据，可以增加一行中的属性列，或者更新已存在的属性列的值。 此属性配置不区分大小写。
	insert.order.enable	boolean	否	true	写入数据表时是否需要保序。取值范围如下： • true（默认）：写入时保持Kafka消息记录的顺序。 • false：写入顺序无保证，但写入效率会提升。
	auto.create	boolean	否	false	是否需要自动创建目标表，支持自动创建数据表或者时序表。取值范围如下： • true：自动创建目标表。 • false（默认）：不自动创建目标表。

分类	配置项	类型	是否必选	示例值	描述
连接器Write配置	delete.mode	string	否	none	删除模式，仅当同步数据到数据表且主键模式为record_key时才有效。取值范围如下： • none（默认）：不允许进行任何删除。 • row：允许删除行。当Record Value为空时会删除行。 • column：允许删除属性列。当Record Value中字段值（Struct类型）或者键值（Map类型）为空时会删除属性列。 • row_and_column：允许删除行和属性列。 此属性配置不区分大小写。 删除操作与insert.mode的配置相关。更多信息，请参见附录：删除语义。
	buffer.size	integer	否	1024	写入数据表时内存中缓冲队列的大小，默认值为1024，单位为行数。此配置项的值必须是2的指数。
	max.thread.count	integer	否	3	写入数据表时的回调线程数，默认值为 `CPU核数+1` 。
	max.concurrency	integer	否	10	写入数据表时的最大请求并发数。
	bucket.count	integer	否	3	写入数据表时的分桶数，默认值为3。适当调大此配置项的值可提升并发写入能力，但不应大于最大请求并发数。
	flush.Interval	integer	否	10000	写入数据表时对缓冲区的刷新时间间隔，默认值为10000，单位为毫秒。

分类	配置项	类型	是否必选	示例值	描述
连接器Runtime Error配置	runtime.error.tolerance	string	否	none	解析Kafka Record或者写入表时产生错误的处理策略。取值范围如下： • none（默认）：任何错误都将导致Sink Task立即失败。 • all：跳过产生错误的Record，并记录该Record。 此属性配置不区分大小写。
	runtime.error.mode	string	否	ignore	解析Kafka Record或者写入表时产生错误，对错误的Record的处理策略。取值范围如下： • ignore（默认）：忽略所有错误。 • kafka：将产生错误的Record和错误信息存储在Kafka的另一个Topic中，此时需要配置runtime.error.bootstrap.servers和runtime.error.topic.name。记录运行错误的Kafka Record的Key和Value与原Record一致，Header中增加ErrorInfo字段来记录运行错误信息。 • tablestore：将产生错误的Record和错误信息存储在Tablestore另一张数据表中，此时需要配置runtime.error.table.name。记录运行错误的数据表主键列为 `topic_partition（string类型）`，`offset（integer类型）`，并且属性列为key（bytes类型）、value（bytes类型）和error_info（string类型）。

分类	配置项	类型	是否必选	示例值	描述
					kafka模式下需要对Kafka Record的Header、Key和Value进行序列化转换，tablestore模式下需要对Kafka Record的Key和Value进行序列化转换，此处默认使用org.apache.kafka.connect.json.JsonConverter，并且配置schemas.enable为true，您可以通过JsonConverter反序列化得到原始数据。关于Converter的更多信息，请参见Kafka Converter。
	runtime.error.bootstrap.servers	string	否	localhost:9092	用于记录运行错误的Kafka集群地址。
	runtime.error.topic.name	string	否	errors	用于记录运行错误的Kafka Topic名称。
	runtime.error.table.name	string	否	errors	用于记录运行错误的Tablestore表名称。
	tablestore.timeseries.<tablename>.measurement	string	是	mName	将JSON中的key值为指定值对应的value值作为_m_name字段写入对应时序表中。 如果设置此配置项为<topic>，则将kafka记录的topic作为_m_name字段写入时序表中。 配置项名称中<tablename>为时序表名称的占位符，请根据实际修改，例如时序表名称为test，则配置项名称为tablestore.timeseries.test.measurement。
	tablestore.timeseries.<tablename>.dataSource	string	是	ds	将JSON中的key值为ds对应的value值作为_data_source字段写入对应时序表中。 配置项名称中<tablename>为时序表名称的占位符，请根据实际修改。

分类	配置项	类型	是否必选	示例值	描述
时序相关配置项	tablestore.timeseries.<tablename>.tags	list	是	region,level	将JSON中key值为region和level所对应的value值作为tags字段写入对应时序表中。 配置项名称中<tablename>为时序表名称的占位符，请根据实际修改。
	tablestore.timeseries.<tablename>.time	string	是	timestamp	将JSON中key值为timestamp对应的value值作为_time字段写入对应时序表中。 配置项名称中<tablename>为时序表名称的占位符，请根据实际修改。
	tablestore.timeseries.<tablename>.time.unit	string	是	MILLISECONDS	tablestore.timeseries.<tablename>.time值的时间戳单位。取值范围为SECONDS、MILLISECONDS、MICROSECONDS、NANOSECONDS。 配置项名称中<tablename>为时序表名称的占位符，请根据实际修改。
	tablestore.timeseries.<tablename>.field.name	list	否	cpu,io	将JSON中key值为cpu和io的键值对作为_field_name以及_field_name的值写入对应时序表。 配置项名称中<tablename>为时序表名称的占位符，请根据实际修改。
	tablestore.timeseries.<tablename>.field.type	string	否	double,integer	tablestore.timeseries.<tablename>.field.name中字段对应的数据类型。取值范围为double、integer、string、binary、boolean。多个数据类型之间用半角冒号（,）分隔。 配置项名称中<tablename>为时序表名称的占位符，请根据实际修改。

分类	配置项	类型	是否必选	示例值	描述
	tablestore.timeseries.mapAll	boolean	否	false	将输入JSON中的非主键字段和时间字段都作为field存储到时序表中。 当配置项取值为false时，tablestore.timeseries.<tablename>.field.name和tablestore.timeseries.<tablename>.field.type必填。
	tablestore.timeseries.toLowerCase	boolean	否	true	将field中的key（输入数据中非主键或者时间的key，或者配置在tablestore.timeseries.<tablename>.field.name中的key）转为小写写入时序表。
	tablestore.timeseries.rowsPerBatch	integer	否	50	写入tablestore时，一次请求支持写入的最大行数。最大值为200，默认值为200。

附录：Kafka和Tablestore数据类型映射

Kafka和Tablestore数据类型映射关系请参见下表。

Kafka Schema Type	Tablestore数据类型
STRING	STRING
INT8、INT16、INT32、INT64	INTEGER
FLOAT32、FLOAT64	DOUBLE
BOOLEAN	BOOLEAN
BYTES	BINARY

附录：删除语义

> **说明** 只有同步数据到数据表时才支持此功能。

当同步数据到数据表且Kafka消息记录的value中存在空值时，根据写入模式（insert.mode）和删除模式（delete.mode）的不同设置，数据写入到表格存储数据表中的处理方式不同，详细说明请参见下表。

insert.mode	put				update			
delete.mode	none	row	column	row_and_column	none	row	column	row_and_column

insert.mode	put				update			
value为空值	覆盖写	删行	覆盖写	删行	脏数据	删行	脏数据	删行
value所有字段值均为空值	覆盖写	覆盖写	覆盖写	覆盖写	脏数据	脏数据	删列	删列
value部分字段值为空值	覆盖写	覆盖写	覆盖写	覆盖写	忽略空值	忽略空值	删列	删列

2.2.5. 错误处理

在将Kafka数据导入到表格存储的过程中可能产生错误，如果您不希望导致Sink Task立即失败，您可以配置错误处理策略。

可能产生的错误类型如下：

- Kafka Connect Error

  此类错误发生在Sink Task执行数据导入前，例如使用Converter进行反序列化或者使用Kafka Transformations对消息记录进行轻量级修改时产生错误，您可以配置由Kafka提供的错误处理选项。

  如果您想要跳过此类错误，请在connector配置文件中配置属性 `errors.tolerance=all` 。更多信息，请参见Kafka Connect Configs。

- Tablestore Sink Task Error

  此类错误发生在Sink Task执行数据导入时，例如解析消息记录或者写入Tablestore时产生错误，您可以配置由Tablestore Sink Connector提供的错误处理选项。

  如果您想要跳过此类错误，请在connector配置文件中配置属性 `errors.tolerance=all` 。更多信息，请参见配置说明。同时，您还可以选择错误报告的方式，如果需要将产生错误的消息记录保存到Tablestore中独立的一张数据表中，请进行如下配置：

  ```
  runtime.error.tolerance=all
  runtime.error.mode=tablestore
  runtime.error.table.name=error
  ```

  在distributed模式下，您也可以通过REST API来管理connector和task。如果发生错误导致connector或者task停止，您可以选择手动重启connector或者task。

  i. 检查connector和task状态。

  - 查看connector状态。

    ```
    curl http://localhost:8083/connectors/{name}/status
    ```

  - 查看task状态。

    ```
    curl http://localhost:8083/connectors/{name}/tasks/{taskid}/status
    ```

  其中 `http://localhost:8083/connectors` 为Kafka REST服务的地址，name必须与配置文件中的name（连接器名称）相同，taskid包含在connector状态信息中。

您还可以通过执行以下命令获取taskid。

```
curl http://localhost:8083/connectors/{name}/tasks
```

ii. 手动重启connector或者task。

- 重启connector。

```
curl -X POST http://localhost:8083/connectors/{name}/restart
```

- 重启task。

```
curl -X POST http://localhost:8083/connectors/{name}/tasks/{taskId}/restart
```

2.3. 将HBase数据同步到表格存储

使用DataX将HBase数据库中的全量数据同步到表格存储（Tablestore）中。

准备工作

- 开通表格存储服务，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。

  > **说明** 创建数据表时建议使用HBase原主键或唯一索引作为表格存储数据表的主键。

- 获取AccessKey用来进行签名认证。AccessKey包括AccessKey ID和AccessKey Secret。具体操作，请参见获取AccessKey。

步骤一：下载DataX

DataX是一个异构数据源离线同步工具，本示例中使用DataX将HBase中的数据同步到表格存储。

您可以选择下载DataX的源代码（开源）进行本地编译或者直接下载编译好的压缩包。

- 下载DataX的源代码
  - GitHub下载

    安装Git工具，然后执行以下操作：

    ```
    git clone https://github.com/alibaba/DataX.git
    ```

  - 本地下载
- 下载编译好的压缩包
  - 在DataX的GitHub地址中获取下载链接：

    ```
    https://github.com/alibaba/DataX
    ```

  - 本地下载

步骤二：Maven打包

如果您下载的是已编译好的DataX压缩包，请跳过此步骤。

> **说明** 此步骤会在本地编译各种数据源的Writer和Reader，花费时间较长，请耐心等待。

1. 进入到下载的源代码目录，并执行如下命令编译源代码。

```
mvn -U clean package assembly:assembly -Dmaven.test.skip=true
```

2. 编译完成后，进入*/target/datax/datax*目录，其中的目录说明请参见下表。

目录	说明
bin	存放可执行的datax.py文件，是整个DataX工具的入口。
plugin	存放支持各种类型数据源的Reader和Writer。
conf	存放core.json文件，该文件中定义了一些缺省参数值，例如channle流控、buffer大小等参数，一般无需修改。

步骤三：准备全量导出的JSON文件

DataX提供了HbaseReader插件从HBase中读取数据。在底层实现上，HbaseReader通过HBase的Java客户端连接远程HBase服务，并通过Scan方式读取指定rowkey范围内的数据，然后将读取的数据使用DataX自定义的数据类型拼装为抽象的数据集，并传递给下游Writer处理。

您可以根据HBase版本选择相应的Reader插件导出JSON文件。

> **说明** DataX仅提供HBase0.94、HBase1.1以及HBase2.0的导出插件，如需导出其他版本的HBase数据，请参考HBase API实现导出工具。

- Hbase0.94 XReader
- Hbase1.1 XReader
- Hbase2.0 XReader：Hbase2.0 XReader插件从Phoenix读取数据。

配置样例

使用Hbase1.1 XReader配置一个从HBase抽取数据到本地的作业（normal 模式）：

```
{
    "job": {
        "setting": {
            "speed": {
                "channel": 1
            }
        },
        "content": [
            {
                "reader": {
                    "name": "hbase11xreader",
                    "parameter": {
                        "hbaseConfig": {
                            "hbase.zookeeper.quorum": "xxxf"
                        },
                        "table": "users",
                        "encoding": "utf-8",
                        "mode": "normal",
                        "column": [
                            {
                                "name": "rowkey",
                                "type": "string"
```

```
                            },
                            {
                                "name": "info: age",
                                "type": "string"
                            },
                            {
                                "name": "info: birthday",
                                "type": "date",
                                "format":"yyyy-MM-dd"
                            },
                            {
                                "name": "info: company",
                                "type": "string"
                            },
                            {
                                "name": "address: contry",
                                "type": "string"
                            },
                            {
                                "name": "address: province",
                                "type": "string"
                            },
                            {
                                "name": "address: city",
                                "type": "string"
                            }
                        ],
                        "range": {
                            "startRowkey": "",
                            "endRowkey": "",
                            "isBinaryRowkey": true
                        }
                    }
                },
                "writer": {
                    "name": "txtfilewriter",
                    "parameter": {
                        "path": "/Users/shf/workplace/datax_test/hbase11xreader/result",
                        "fileName": "qiran",
                        "writeMode": "truncate"
                    }
                }
            }
        ]
    }
}
```

步骤四：执行同步命令

执行如下命令同步数据。

```
python datax.py  -j"-Xms4g -Xmx4g" hbase_to_ots.json
```

其中 `-j"-Xms4g -Xmx4g"` 可以限制占用JVM内存的大小；如果不指定，将会使用 `conf/core.json` 中的配置，默认为1 GB。

相关操作

HBase数据迁移到表格存储后，您可以使用Tablestore SDK或者Tablestore HBase Client读取表格存储的数据，具体操作，请参见从HBase Client迁移到Tablestore HBase Client。

2.4. 将MaxCompute数据同步到表格存储

通过DataWorks控制台将MaxCompute中的全量数据同步到表格存储。

背景信息

表格存储能够支持千万TPS以及毫秒级延迟的服务能力，拥有强大的读写能力，同时表格存储还提供多元索引等强大的索引功能，满足各种搜索场景。您可以将MaxCompute计算分析后的数据同步到表格存储中，提升应用的读写和搜索效率。

步骤一：新增数据源

将表格存储数据库添加为数据源，具体操作步骤如下：

1. 以项目管理员身份登录DataWorks控制台。

   > **说明** 仅项目管理员角色可以新增数据源，其他角色的成员仅可查看数据源。

2. 选择区域，在左侧导航栏，单击**工作空间列表**。
3. 在**工作空间列表**页面，单击工作空间操作区域的**进入数据集成**。
4. 在数据集成控制台，单击**数据源管理**。
5. 新增数据源。
   i. 在**数据源管理**页面，单击**新增数据源**。
   ii. 在**新增数据源**对话框的**NoSQL**区域，选择数据源类型为**OTS**。
   iii. 在**新增OTS数据源**对话框，配置参数。

参数	说明
数据源名称	数据源的名称，例如gps_data。
数据源描述	数据源的描述信息。
Endpoint	填写目标Tablestore实例的服务地址。 ■ 如果Tablestore实例和MaxCompute在同一个region，填写经典网地址。 ■ 如果Tablestore实例和MaxCompute不在同一个region，填写公网地址。 ■ 不能填写VPC地址。
Table Store实例名称	Tablestore实例的名称。
AccessKey ID	登录账户的AccessKeyID和AccessKeySecret，获取方式请参见为RAM用户创建访问密钥。
AccessKey Secret	

iv. 单击**测试连通性**，测试数据源的连通状态。

6. 单击**完成**。

在**数据源管理**页面，会显示该数据源信息。

步骤二：新建同步任务

新建并配置MaxCompute到表格存储的同步任务，具体操作步骤如下：

1. 以项目管理员身份登录DataWorks控制台。

   > **说明** 仅项目管理员角色可以新增数据源，其他角色的成员仅可查看数据源。

2. 选择区域，在左侧导航栏，单击**工作空间列表**。
3. 在**工作空间列表**页面，单击工作空间操作中的**进入数据开发**。
4. 在DataStudio控制台的**数据开发**页面，单击**业务流程**节点下的目标业务流程。

   如果需要新建业务流程，请参见创建业务流程。

5. 新建同步任务节点。

   每个同步任务都需创建一个相应的节点。

   i. 在**数据集成**节点上右键选择**新建 > 离线同步**。

   ii. 在**新建节点**对话框，输入**节点名称**。

   iii. 单击**提交**。

6. 配置数据源。

   i. 在**数据集成**节点下，双击同步任务节点。

ii. 在同步任务节点的编辑页面的**选择数据源**区域，配置数据来源和数据去向。

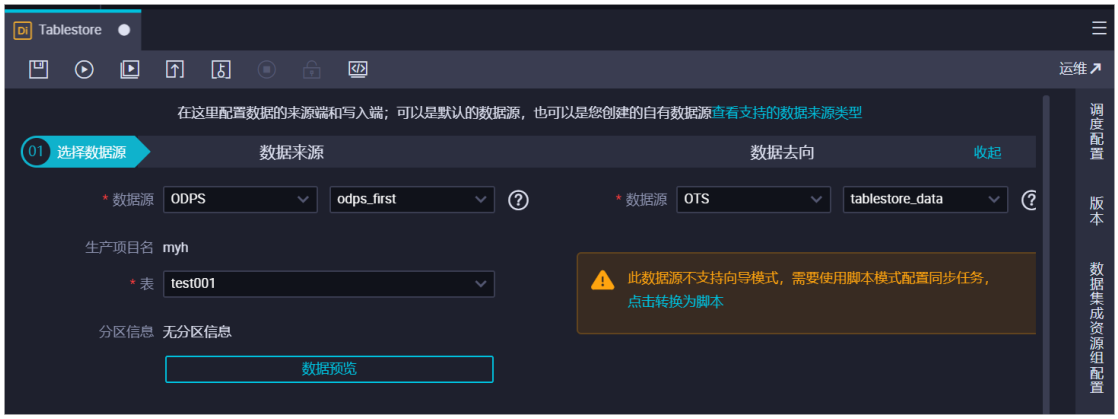
- 配置数据来源。

  选择**数据来源**的**数据源**为ODPS，并选择对应的**表**。

- 配置数据去向。

  选择**数据去向**的**数据源**为OTS，并单击 图标或者**点击转换为脚本**，进行脚本配置。

> **说明** 表格存储仅支持脚本模式，配置脚本的详情请参见配置Tablestore（OTS）Writer。



iii. 单击 图标，保存数据源配置。

7. 运行同步任务。

   i. 单击 图标。

   ii. 在**参数**对话框，选择调度的资源组。

   iii. 单击**确定**，开始运行任务。

   运行结束后，在**运行日志**页签中可以查看任务是否成功和导出的数据行数。

步骤三：定时执行同步任务

1. 以项目管理员身份登录DataWorks控制台。

   > **说明** 仅项目管理员角色可以新增数据源，其他角色的成员仅可查看数据源。

2. 选择区域，在左侧导航栏，单击**工作空间列表**。
3. 在**工作空间列表**页面，单击工作空间操作中的**进入数据开发**。
4. 在DataStudio控制台的**数据开发**页面，单击**业务流程**节点下的目标业务流程。
5. 配置调度参数。

   通过**调度配置**，可以配置同步任务的执行时间、重跑属性、调度依赖等。

   i. 在**数据集成**节点下，双击同步任务节点。

   ii. 在同步任务节点的编辑页面的右侧单击**调度配置**，进行调度参数配置，详情请参见配置调度和依赖属性。

6. 提交同步任务。
   i. 在同步任务节点的编辑页面，单击图标。
   ii. 在**提交新版本**对话框，输入备注信息。
   iii. 单击**确认**。
   将同步任务提交到调度系统后，调度系统会根据配置的调度参数，自动定时执行同步任务。

步骤四：查看同步任务

1. 以项目管理员身份登录DataWorks控制台。

   > **说明** 仅项目管理员角色可以新增数据源，其他角色的成员仅可查看数据源。

2. 选择区域，在左侧导航栏，单击**工作空间列表**。
3. 在**工作空间列表**页面，单击工作空间操作中的**进入运维中心**。
4. 在运维中心控制台，选择**周期任务运维 > 周期任务**。
5. 在**周期任务**页面，查看提交的同步任务详情。

2.5. 将表格存储数据表中数据同步到另一个数据表

使用通道服务、DataWorks或者DataX将表格存储数据表中的数据同步到另一个数据表。

前提条件

已创建目标数据表。具体操作，请参见创建数据表。

> **注意** 目标数据表的列必须与源数据表中待迁移的列一一对应。

使用通道服务迁移同步

创建源数据表的通道后，使用SDK进行迁移同步。迁移过程中可以自定义业务处理逻辑对数据进行处理。

1. 使用表格存储控制台或SDK创建源数据表的通道并记录通道ID，具体操作请分别参见快速入门或使用SDK。
2. 使用SDK迁移数据。

   示例代码如下：

```
public class TunnelTest {
    public static void main(String[] args){
       TunnelClient tunnelClient = new TunnelClient("endpoint",
               "accessKeyId","accessKeySecret","instanceName");
        TunnelWorkerConfig config = new TunnelWorkerConfig(new SimpleProcessor());
        //tunnelId可以在表格存储控制台通道管理页面查看或者调用describeTunnelRequest查询。
        TunnelWorker worker = new TunnelWorker("tunnelId", tunnelClient, config);
        try {
            worker.connectAndWorking();
        } catch (Exception e) {
            e.printStackTrace();
            worker.shutdown();
            tunnelClient.shutdown();
        }
    }
    public static class SimpleProcessor implements IChannelProcessor{
       //目标tablestore连接对象。
       TunnelClient tunnelClient = new TunnelClient("endpoint",
               "accessKeyId","accessKeySecret","instanceName");
       @Override
        public void process(ProcessRecordsInput processRecordsInput) {
            //ProcessRecordsInput中返回了增量或全量数据。
            List<StreamRecord> list = processRecordsInput.getRecords();
            for(StreamRecord streamRecord : list){
                switch (streamRecord.getRecordType()){
                    case PUT:
                        //自定义业务处理逻辑。
                        //putRow
                        break;
                    case UPDATE:
                        //updateRow
                        break;
                    case DELETE:
                        //deleteRow
                        break;
                }
                System.out.println(streamRecord.toString());
            }
        }
        @Override
        public void shutdown() {
        }
    }
}
```

使用DataWorks或者DataX迁移同步

通过DataWorks或者DataX实现表格存储数据的迁移同步，此处以DataWorks为例介绍迁移操作。

1. 新增表格存储数据源。

   分别以源数据表和目标数据表所在实例新增表格存储数据源。具体操作，请参见步骤一：新增数据源。

2. 新建同步任务节点。

   i. 以项目管理员身份登录DataWorks控制台。

   > **说明** 仅项目管理员角色可以新增数据源，其他角色的成员仅可查看数据源。

   ii. 在左侧导航栏，单击**工作空间列表**后，选择地域。

   iii. 在**工作空间列表**页面，单击目标工作空间操作栏中的**数据开发**。

   iv. 在DataStudio控制台的**数据开发**页面，单击**业务流程**节点下的目标业务流程。

   如果需要新建业务流程，请参见创建业务流程。

   v. 在**数据集成**节点上右键选择**新建节点 > 离线同步**。

   vi. 在**新建节点**对话框，输入**名称**，单击**提交**。

3. 配置数据源。

   i. 在**数据集成**节点下，双击同步任务节点。

   ii. 在同步任务节点的编辑页面的**选择数据源**区域，配置数据来源和数据去向。

   选择**数据来源**和**数据去向**的**数据源**为**OTS**并分别设置为源数据表和目标数据表对应的数据源，然后单击  图标或者单击**点击转换为脚本**，进行脚本配置。

   > **注意** 表格存储仅支持脚本模式配置。

   - 配置Tablestore（OTS） Reader

     Tablestore（OTS） Reader插件实现了从Tablestore读取数据，通过您指定的抽取数据范围，可以方便地实现数据增量抽取的需求。具体操作，请参见配置Tablestore（OTS） Reader。

   - 配置Tablestore（OTS） Writer

     Tablestore（OTS） Writer通过Tablestore官方Java SDK连接到Tablestore服务端，并通过SDK写入Tablestore服务端 。Tablestore Writer本身对于写入过程进行了诸多优化，包括写入超时重试、异常写入重试、批量提交等功能。具体操作，请参见配置Tablestore（OTS） Writer。

   iii. 单击  图标，保存数据源配置。

4. 运行同步任务。

   i. 单击  图标。

   ii. 在**参数**对话框，选择调度的资源组。

   iii. 单击**确定**，开始运行任务。

   运行结束后，在**运行日志**页签中可以查看任务是否成功和导出的数据行数。

5. （可选）定时执行同步任务。具体操作，请参见步骤三：定时执行同步任务。

3.数据导出

3.1. 将表格存储数据同步到OSS

3.1.1. 概述

Tablestore中的增量数据及全量数据可以通过数据集成的脚本模式同步到OSS中。

Tablestore是构建在阿里云飞天分布式系统之上的分布式NoSQL数据存储服务，根据99.99%的高可用以及11个9的数据可靠性的标准设计。表格存储通过数据分片和负载均衡技术，实现数据规模与访问并发的无缝扩展，提供海量结构化数据的存储和实时访问。

OSS（对象存储）是海量、安全、低成本、高可靠的云存储服务，提供99.99999999%的数据可靠性。使用RESTful API可以在互联网任何位置存储和访问，容量和处理能力弹性扩展，多种存储类型供选择全面优化存储成本。

使用场景

Tablestore：提供专业的数据持久化存储服务，以及面向用户的实时高并发低延迟读写操作。

OSS：提供极低成本的备份功能。

使用方式

- 写

  直接写入Tablestore。

- 读

  直接读取Tablestore。

- 备份

  自动备份。

- 恢复

  使用数据集成（OSSreader + OTSwriter）重新写回Tablestore

限制

- 整行写入

  使用Tablestore Stream功能，要求每次写入Tablestore的数据必须是整行数据。目前类似物联网数据的时序数据写入方式都是整行写入，后续基本无修改。

- 同步延时

  目前使用的是周期调度，每隔5分钟调度一次，并且插件有5分钟延迟，同步总延迟为5~10分钟。

开通服务

- 开通表格存储服务。具体操作，请参见开通表格存储服务。
- 开通OSS服务。

数据通道

离线

- 全量导出到OSS
  - 脚本模式
- 增量同步到OSS
  - 脚本模式
- 全量导入到Tablestore
  - 脚本模式

3.1.2. 全量导出（脚本模式）

通过DataWorks控制台将表格存储中的全量数据同步到OSS中。同步到OSS中的文件可自由下载，也可作为表格存储数据的备份存于OSS中。

步骤一：新增表格存储数据源

将表格存储添加为数据源，具体操作步骤如下：

1. 进入数据集成。
   i. 以项目管理员身份登录DataWorks控制台。

      > **说明** 仅项目管理员角色可以新增数据源，其他角色的成员仅可查看数据源。

   ii. 选择地域，在左侧导航栏，单击**工作空间列表**。
   iii. 在**工作空间列表**页面，单击工作空间操作区域的**进入数据集成**。
2. 新增数据源。
   i. 在数据集成控制台，单击**数据源管理**。
   ii. 在**数据源管理**页面，单击**新增数据源**。
   iii. 在**新增数据源**对话框的**NoSQL**区域，选择数据源类型为**OTS**。
   iv. 在**新增OTS数据源**对话框，配置参数。

参数	说明
数据源名称	数据源的名称。
数据源描述	数据源的描述信息。
Endpoint	填写目标Tablestore实例的服务地址。 ■ 如果Tablestore实例和OSS在同一个地域，填写经典网地址。 ■ 如果Tablestore实例和OSS不在同一个地域，填写公网地址。 ■ 不能填写VPC地址。
Table Store实例名称	Tablestore实例的名称。
AccessKey ID	登录账户的AccessKey ID和AccessKey Secret，获取方式请参见为RAM用户创建访问密钥。
AccessKey Secret	

v. 单击**测试连通性**，测试数据源的连通状态。

3. 单击**完成**。

在**数据源管理**页面，会显示该数据源信息。

步骤二：新增OSS数据源

操作与步骤一类似，只需在**Semi-structuredstorage**区域，选择数据源类型为**OSS**。

> **说明** 配置OSS数据源的参数时，请注意Endpoint中不包括Bucket的名称。

本示例中，该数据源名称使用OTS2OSS，如下图所示。

步骤三：新建同步任务

新建并配置表格存储到OSS的同步任务，具体操作步骤如下：

1. 进入数据开发。
   i. 以项目管理员身份登录DataWorks控制台。
   ii. 选择地域，在左侧导航栏，单击**工作空间列表**。
   iii. 在**工作空间列表**页面，单击工作空间操作区域的**进入数据开发**。

2. 在DataStudio控制台的**数据开发**页面，单击**业务流程**节点下的目标业务流程。

   如果需要新建业务流程，请参见创建业务流程。

3. 新建同步任务节点。

   每个同步任务都需创建一个相应的节点。

   i. 在**数据集成**节点上右键选择**新建 > 离线同步**。

   您也可以将鼠标悬停在  图标，选择**数据集成 > 离线同步**来新建节点。

   ii. 在**新建节点**对话框，输入节点名称，选择一个目标文件夹。

   iii. 单击**提交**。

4. 配置数据源。

   i. 在**数据集成**节点下，双击同步任务节点。

   ii. 在同步任务节点的编辑页面的**选择数据源**区域，配置数据来源和数据去向。

   - 配置数据来源。

     选择**数据来源**的**数据源**为**OTS**。

   - 配置数据去向。

     选择**数据去向**的**数据源**为**OSS**，并配置数据源。

   iii. 单击  图标或者**点击转换为脚本**，进行脚本配置。

表格存储仅支持脚本模式配置，使用过程中涉及Tablestore（OTS） Reader和OSS Writer插件的配置。具体操作，请参见配置Tablestore（OTS） Reader和配置OSS Writer。

在脚本配置页面，请根据如下示例完成配置。

```
{
"type": "job",     # 不能修改。
"version": "1.0",   # 不能修改。
"configuration": {
 "setting": {
   "errorLimit": {
     "record": "0"   # 当错误个数超过record个数时，导入任务会失败。
   },
   "speed": {
     "mbps": "1",   # 导入速率，单位是MB/s。
     "concurrent": "1"   # 并发度。
   }
 },
 "reader": {
   "plugin": "ots",   # 不能修改。
   "parameter": {
     "datasource": "",    # 数据集成中的数据源名称，需要提前配置完成，此处可选择配置Tablestore的数据源或者填写明文的AccessKeyID等鉴权信息，建议使用数据源。
     "table": "",     # Tablestore中的数据表名称。
     "column": [    # 需要导出到OSS的列名，不能设置为空。
       {
         "name": "column1"     # Tablestore中列名，此列需要导入到OSS。
       },
       {
         "name": "column2"    # Tablestore中列名，此列需要导入到OSS。
       }
     ],
     "range": {
       "begin": [
         {
           "type": "INF_MIN"    # Tablestore中第一列主键的起始位置。如果需要导出全量，此处请配置为INF_MIN；如果只需导出部分，则按需要配置。当数据表存在多个主键列时，此处begin中需要配置对应主键列信息。
         }
       ],
       "end": [
         {
           "type": "INF_MAX"    # Tablestore中第一列主键的结束位置。如果需要导出全量，此处请配置为INF_MAX；如果只需导出部分，则按需要配置。当数据表存在多个主键列时，此处end中需要配置对应主键列信息。
         }
       ],
       "split": [  # 用于配置Tablestore的数据表的分区信息，可以加速导出，下一个版本会自动处理。
       ]
     }
   }
 },
 "writer": {
```

```
    "plugin": "oss",
    "parameter": {
      "datasource": "",   # 配置OSS的数据源。
      "object": "",   # Object的前缀，无需包括Bucket名称，例如tablestore/20171111/。如果是定时导出，则此处需要使用变量，例如tablestore/${date}，然后在配置调度参数时配置${date}的值。
      "writeMode": "truncate", # 当同名文件存在时系统进行的操作，全量导出时请使用truncate，可选值包括truncate、append和nonConflict，truncate表示会清理已存在的同名文件，append表示会加到已存在的同名文件内容后面，nonConflict表示当同名文件存在时会报错。
      "fileFormat": "csv", # 文件类型，可选值包括csv、txt和parquet格式。
      "encoding": "UTF-8",   # 编码类型。
      "nullFormat": "null", # 定义null值的字符串标识符方式，可以是空字符串。
      "dateFormat": "yyyy-MM-dd HH:mm:ss",   # 时间格式。
      "fieldDelimiter": "," # 每一列的分隔符。
    }
  }
}
}
```

iv. 单击图标，保存数据源配置。

> **说明**
>
> - 由于全量导出一般是一次性的，所以无需配置自动调度参数。如果需要配置调度参数，请参见增量同步中的调度参数配置。
> - 如果在脚本配置中存在变量，例如存在*${date}*，则需要在运行同步任务时设置变量的具体值。

5. 运行同步任务。

   i. 单击图标。

   ii. 在**参数**对话框，选择调度的资源组。

   iii. 单击**确定**，开始运行任务。

   运行结束后，在**运行日志**页签中可以查看任务是否成功和导出的数据行数。

步骤四：查看导出到OSS中的数据

1. 登录OSS管理控制台。
2. 选择相应Bucket和文件名，下载后可查看内容是否符合预期。

3.1.3. 增量同步（脚本模式）

通过DataWorks控制台将表格存储中的增量数据同步到OSS中。

步骤一：新增表格存储数据源

如果已新增表格存储数据源，请跳过此步骤。

新增表格存储数据源的操作请参见步骤一：新增表格存储数据源。

步骤二：新增OSS数据源

如果已新增OSS数据源，请跳过此步骤。

新增OSS数据源的操作请参见步骤二：新增OSS数据源。

步骤三：配置定时同步任务

新建并配置表格存储到OSS的增量数据同步任务，具体操作步骤如下：

1. 进入数据开发。
   i. 以项目管理员身份登录DataWorks控制台。

   > **说明** 仅项目管理员角色可以新增数据源，其他角色的成员仅可查看数据源。

   ii. 选择地域，在左侧导航栏，单击**工作空间列表**。
   iii. 在**工作空间列表**页面，单击工作空间操作区域的**进入数据开发**。
2. 在DataStudio控制台的**数据开发**页面，单击**业务流程**节点下的目标业务流程。

   如果需要新建业务流程，请参见创建业务流程。
3. 新建同步任务节点。

   每个同步任务都需创建一个相应的节点。
   i. 在**数据集成**节点上右键选择**新建 > 离线同步**。

   您也可以将鼠标悬停在  图标，选择**数据集成 > 离线同步**来新建节点。

   ii. 在**新建节点**对话框，输入节点名称，选择一个目标文件夹。

   iii. 单击**提交**。
4. 配置数据源。
   i. 在**数据集成**节点下，双击同步任务节点。

ii. 在同步任务节点的编辑页面的**选择数据源**区域，配置数据来源和数据去向。

- 配置数据来源。

  选择**数据来源**的**数据源**为**OTS Stream**，选择数据源和数据表，可根据需要配置任务开始时间、结束时间、状态表的名称、最大重试次数等。

- 配置数据去向。

  选择**数据去向**的**数据源**为**OSS**，选择数据源，配置Object前缀、文本类型、列的分隔符等。

iii. 单击  图标，进行脚本配置。

使用过程时涉及OTSStream Reader和OSS Writer插件的配置。具体操作，请参见配置OTSStream Reader和配置OSS Writer。

在脚本配置页面，请根据如下示例完成配置。

```
{
"type": "job",
"version": "1.0",
"configuration": {
"setting": {
"errorLimit": {
 "record": "0"  # 允许出错的个数，当错误超过这个数目的时候同步任务会失败。
},
"speed": {
 "mbps": "1",  # 每次同步任务的最大流量。
 "concurrent": "1"   # 每次同步任务的并发度。
}
},
"reader": {
"plugin": "otsstream",  # Reader插件的名称。
"parameter": {
 "datasource": "", # Tablestore的数据源名称，如果有此项则无需配置endpoint，accessId，accessKey和instanceName。
 "dataTable": "", # Tablestore中的数据表名称。
 "statusTable": "TablestoreStreamReaderStatusTable", # 存储Tablestore Stream状态的表，一般无需修改。
 "startTimestampMillis": "",  # 开始导出的时间点，由于是增量导出，需要循环启动此任务，则此处每次启动时的时间都不同，因此需要设置一个变量，例如${start_time}。
 "endTimestampMillis": "",  # 结束导出的时间点。此处也需要设置一个变量，例如${end_time}。
```

```
 "date": "yyyyMMdd",  # 导出该日期的数据，此功能与startTimestampMillis和endTimestampMil
lis重复，需要删除。
 "mode": "single_version_and_update_only", # Tablestore Stream导出数据的格式，目前需要
设置为single_version_and_update_only。如果配置模板中无此项，则需要增加。
 "column":[  # 设置数据表中需要导出到OSS中的列，如果配置模板中无此项则需要增加，具体列个数由
用户自定义设置。
          {
             "name": "uid"  # 列名，此处是Tablestore中的主键。
          },
          {
             "name": "name"  # 列名，此处是Tablestore中的属性列。
          },
 ],
 "isExportSequenceInfo": false, # single_version_and_update_only模式下只能设置为false
。
 "maxRetries": 30 # 最大重试次数。
}
},
"writer": {
"plugin": "oss", # Writer插件的名称。
"parameter": {
 "datasource": "", # OSS的数据源名称。
 "object": "",  # 备份到OSS的文件名前缀，建议使用"Tablestore实例名/表名/date"，例如"insta
nce/table/{date}"。
 "writeMode": "truncate", # 当同名文件存在时系统进行的操作，可选值包括truncate、append和n
onConflict，truncate表示会清理已存在的同名文件，append表示会加到已存在的同名文件内容后面，n
onConflict表示当同名文件存在时会报错。
 "fileFormat": "csv", # 文件类型，可选值包括csv、txt和parquet格式。
 "encoding": "UTF-8", # 编码类型。
 "nullFormat": "null", # 定义null值的字符串标识符方式，可以是空字符串。
 "dateFormat": "yyyy-MM-dd HH:mm:ss", # # 时间格式。
 "fieldDelimiter": "," # 每一列的分隔符。
}
}
}
}
```

iv. 单击  图标，保存数据源配置。

5. 运行同步任务。

   i. 单击  图标。

   ii. 在**参数**对话框，选择调度的资源组。

   iii. 单击**确定**，开始运行任务。

   运行结束后，在**运行日志**页签中可以查看任务是否成功和导出的数据行数。

   表格存储的增量数据可以在延迟5~10分钟的基础上自动同步到OSS中。

6. 配置调度参数。

   通过**调度配置**，可以配置同步任务的执行时间、重跑属性、调度依赖等。

   i. 在**数据集成**节点下，双击同步任务节点。

ii. 在同步任务节点的编辑页面的右侧单击**调度配置**，进行调度参数配置，详情请参见配置调度和依赖属性。

7. 提交同步任务。

   将同步任务提交到调度系统后，调度系统会根据配置的调度参数，自动定时执行同步任务。

   i. 在同步任务节点的编辑页面，单击  图标。

   ii. 在**提交新版本**对话框，输入备注信息。

   iii. 单击**确认**。

步骤四：查看同步任务

1. 进入运维中心。

   > **说明** 您也可以在DataStudio控制台的右上角单击**运维中心**，快速进入运维中心。

   i. 以项目管理员身份登录DataWorks控制台。

   ii. 选择地域，在左侧导航栏，单击**工作空间列表**。

   iii. 在**工作空间列表**页面，单击工作空间操作区域的**进入运维中心**。

2. 在运维中心控制台，选择**周期任务运维 > 周期任务**。

3. 在**周期任务**页面，查看提交的同步任务详情。

   - 在左侧导航栏中，选择**周期任务运维 > 周期实例**，可以查看当天需要运行的周期任务。单击实例名称，可以查看任务运行详情。
   - 当单个任务在运行中或运行结束后，可以查看日志。

步骤五：查看导出到OSS中的数据

1. 登录OSS管理控制台。
2. 选择相应Bucket和文件名，下载后可查看内容是否符合预期。

3.2. 将表格存储数据同步到MaxCompute

3.2.1. 全量导出（脚本模式）

通过DataWorks控制台将表格存储中的全量数据导出到MaxCompute中。

步骤一：新增表格存储数据源

将表格存储数据库添加为数据源，具体操作步骤如下：

1. 进入数据集成。

   i. 以项目管理员身份登录DataWorks控制台。

   > **说明** 仅项目管理员角色可以新增数据源，其他角色的成员仅可查看数据源。

   ii. 选择地域，在左侧导航栏，单击**工作空间列表**。

   iii. 在**工作空间列表**页面，单击工作空间操作区域的**进入数据集成**。

2. 新增数据源。

i. 在数据集成控制台，单击**数据源管理**。
ii. 在**数据源管理**页面，单击**新增数据源**。
iii. 在**新增数据源**对话框的**NoSQL**区域，选择数据源类型为**OTS**。
iv. 在**新增OTS数据源**对话框，配置参数。

参数	说明
数据源名称	数据源的名称，例如gps_data。
数据源描述	数据源的描述信息。
Endpoint	填写目标Tablestore实例的服务地址。 ■ 如果Tablestore实例和MaxCompute在同一个地域，填写经典网地址。 ■ 如果Tablestore实例和MaxCompute不在同一个地域，填写公网地址。 ■ 不能填写VPC地址。
Table Store实例名称	Tablestore实例的名称。

参数	说明
AccessKey ID	登录账户的AccessKey ID和AccessKey Secret，获取方式请参见为RAM用户创建访问密钥。
AccessKey Secret	

v. 单击**测试连通性**，测试数据源的连通状态。

3. 单击**完成**。

在**数据源管理**页面，会显示该数据源信息。

步骤二：新增MaxCompute数据源

操作与步骤一类似，只需在**Big Data Storage**区域，选择数据源类型为**MaxCompute(ODPS)**。

本示例中，该数据源名称使用OTS2ODPS，如下图所示。

步骤三：配置同步任务

新建并配置表格存储到MaxCompute的同步任务，具体操作步骤如下：

1. 进入数据开发。
   i. 以项目管理员身份登录DataWorks控制台。
   ii. 选择地域，在左侧导航栏，单击**工作空间列表**。
   iii. 在**工作空间列表**页面，单击工作空间操作区域的**进入数据开发**。
2. 在DataStudio控制台的**数据开发**页面，单击**业务流程**节点下的目标业务流程。

   如果需要新建业务流程，请参见创建业务流程。
3. 新建同步任务节点。

   每个同步任务都需创建一个相应的节点。

i. 在**数据集成**节点上右键选择**新建 > 离线同步**。

您也可以将鼠标悬停在  图标，选择**数据集成 > 离线同步**来新建节点。

ii. 在**新建节点**对话框，输入节点名称，选择一个目标文件夹。

iii. 单击**提交**。

4. 配置数据源。

i. 在**数据集成**节点下，双击同步任务节点。

ii. 在同步任务节点的编辑页面的**选择数据源**区域，配置数据来源和数据去向。

- 配置数据来源。

  选择**数据来源**的**数据源**为OTS。

- 配置数据去向。

  选择**数据去向**的**数据源**为ODPS，并选择对应的**表**。

iii. 单击  图标或者**点击转换为脚本**，进行脚本配置。

表格存储仅支持脚本模式配置，使用过程中涉及Tablestore（OTS） Reader和MaxCompute Writer插件的配置。具体操作，请参见配置Tablestore（OTS） Reader和MaxCompute Writer。

在脚本配置页面，请根据如下示例完成配置。

```
{
"type": "job",
"version": "1.0",
"configuration": {
```

```
"setting": {
  "errorLimit": {
    "record": "0"     # 能够允许的最大错误数。
  },
  "speed": {
    "mbps": "1",    # 最大的流量，单位为MB。
    "concurrent": "1"  # 并发数。
  }
},
"reader": {
  "plugin": "ots",  # 读取的插件名称。
  "parameter": {
    "datasource": "",  # 数据源名称。
    "table": "",  # 数据表名称。
    "column": [  # 需要导出到MaxCompute中去的表格存储中的列名。
      {
        "name": "column1"
      },
      {
        "name": "column2"
      },
      {
        "name": "column3"
      },
      {
        "name": "column4"
      },
      {
        "name": "column5"
      }
    ],
    "range": {  # 需要导出的数据范围，如果是全量导出，则需要从INF_MIN到INF_MAX。
      "begin": [ # 需要导出数据的起始位置，最小的位置是INF_MIN。begin中的配置项数目个数和表格存储中相应表的主键列个数一致。
        {
          "type": "INF_MIN"
        },
        {
          "type": "INF_MIN"
        },
        {
          "type": "STRING",  # 此配置项表示第三列的起始位置是begin1。
          "value": "begin1"
        },
        {
          "type": "INT",  # 此配置项表示第四列的起始位置是0。
          "value": "0"
        }
      ],
      "end": [  # 导出数据的结束位置。
        {
          "type": "INF_MAX"
        },
        {
```

```
          "type": "INF_MAX"
        },
        {
          "type": "STRING",
          "value": "end1"
        },
        {
          "type": "INT",
          "value": "100"
        }
      ],
      "split": [  # 配置分区范围，一般可以不配置，如果性能较差，可以提交工单或者加入钉钉群23307953联系表格存储技术支持人员处理。
        {
          "type": "STRING",
          "value": "splitPoint1"
        },
        {
          "type": "STRING",
          "value": "splitPoint2"
        },
        {
          "type": "STRING",
          "value": "splitPoint3"
        }
      ]
    }
  }
},
"writer": {
  "plugin": "odps",  # MaxCompute写入的插件名。
  "parameter": {
    "datasource": "",  # MaxCompute的数据源名称。
    "column": [],  # MaxCompute中的列名，列名顺序需对应TableStore中的列名顺序。
    "table": "",  # MaxCompute中的表名，需要提前创建好，否则任务执行会失败。
    "partition": "",  # 如果表为分区表，则必填。如果表为非分区表，则不能填写。需要写入数据表的分区信息，必须指定到最后一级分区。
    "truncate": false  # 是否清空之前的数据。
  }
}
}
}
```

您可以通过begin和end来配置导出的数据范围，假设表包含pk1（String类型）和pk2（Integer类型）两个主键列。

- 如果需要导出全表数据，则配置示例如下：

```
"begin": [ # 需要导出数据的起始位置。
  {
    "type": "INF_MIN"
  },
  {
    "type": "INF_MIN"
  }
],
"end": [  # 需要导出数据的结束位置。
  {
    "type": "INF_MAX"
  },
  {
    "type": "INF_MAX"
  }
],
```

- 如果需要导出pk1="tablestore" 的行，则配置示例如下：

```
"begin": [ # 导出数据的起始位置。
  {
    "type": "STRING",
    "value": "tablestore"
  },
  {
    "type": "INF_MIN"
  }
],
"end": [  # 导出数据的结束位置。
  {
    "type": "STRING",
    "value": "tablestore"
  },
  {
    "type": "INF_MAX"
  }
],
```

iv. 单击  图标，保存数据源配置。

5. 运行同步任务。

i. 单击  图标。

ii. 在**参数**对话框，选择调度的资源组。

iii. 单击**确定**，开始运行任务。

运行结束后，在**运行日志**页签中可以查看任务是否成功和导出的数据行数。

6. 配置调度参数。

通过**调度配置**，可以配置同步任务的执行时间、重跑属性、调度依赖等。

i. 在**数据集成**节点下，双击同步任务节点。

ii. 在同步任务节点的编辑页面的右侧单击**调度配置**，进行调度参数配置，详情请参见配置调度和依赖属性。

7. 提交同步任务。

i. 在同步任务节点的编辑页面，单击  图标。

ii. 在**提交新版本**对话框，输入备注信息。

iii. 单击**确认**。

将同步任务提交到调度系统后，调度系统会根据配置的调度参数，自动定时执行同步任务。

步骤四：查看同步任务

1. 进入运维中心。

> **说明** 您也可以在DataStudio控制台的右上角单击**运维中心**，快速进入运维中心。

i. 以项目管理员身份登录DataWorks控制台。

ii. 选择地域，在左侧导航栏，单击**工作空间列表**。

iii. 在**工作空间列表**页面，单击工作空间操作区域的**进入运维中心**。

2. 在运维中心控制台，选择**周期任务运维 > 周期任务**。
3. 在**周期任务**页面，查看提交的同步任务详情。
   - 在左侧导航栏中，选择**周期任务运维 > 周期实例**，可以查看当天需要运行的周期任务。单击实例名称，可以查看任务运行详情。
   - 当单个任务在运行中或运行结束后，可以查看日志。

步骤五：查看导入到MaxCompute中的数据

1. 进入数据地图。

i. 以项目管理员身份登录DataWorks控制台。

ii. 选择地域，在左侧导航栏，单击**工作空间列表**。

iii. 在**工作空间列表**页面，单击工作空间操作区域的**进入数据地图**。

2. 在数据地图控制台的导航栏，选择**我的数据 > 我管理的数据**。
3. 在**我管理的数据**页签，单击导入数据的表名称。
4. 在表详情页面，单击**数据预览**页签，查看导入的数据。

3.2.2. 增量同步（脚本模式）

通过DataWorks控制台将表格存储中的增量数据导出到MaxCompute中。

步骤一：新增表格存储数据源

如果已新增表格存储数据源，请跳过此步骤。

新增表格存储数据源的操作请参见步骤一：新增表格存储数据源。

步骤二：新增MaxCompute数据源

如果已新增MaxCompute数据源，请跳过此步骤。

新增MaxCompute数据源的操作请参见步骤二：新增MaxCompute数据源。

步骤三：配置同步任务

新建并配置表格存储到MaxCompute的增量数据同步任务，具体操作步骤如下：

1. 进入数据开发。
   i. 以项目管理员身份登录DataWorks控制台。
   ii. 选择地域，在左侧导航栏，单击**工作空间列表**。
   iii. 在**工作空间列表**页面，单击工作空间操作区域的**进入数据开发**。
2. 在DataStudio控制台的**数据开发**页面，单击**业务流程**节点下的目标业务流程。

   如果需要新建业务流程，请参见创建业务流程。
3. 新建同步任务节点。

   每个同步任务都需创建一个相应的节点。
   i. 在**数据集成**节点上右键选择**新建 > 离线同步**。

      您也可以将鼠标悬停在  图标，选择**数据集成 > 离线同步**来新建节点。
   ii. 在**新建节点**对话框，输入节点名称，选择一个目标文件夹。

   iii. 单击**提交**。
4. 配置数据源。
   i. 在**数据集成**节点下，双击同步任务节点。

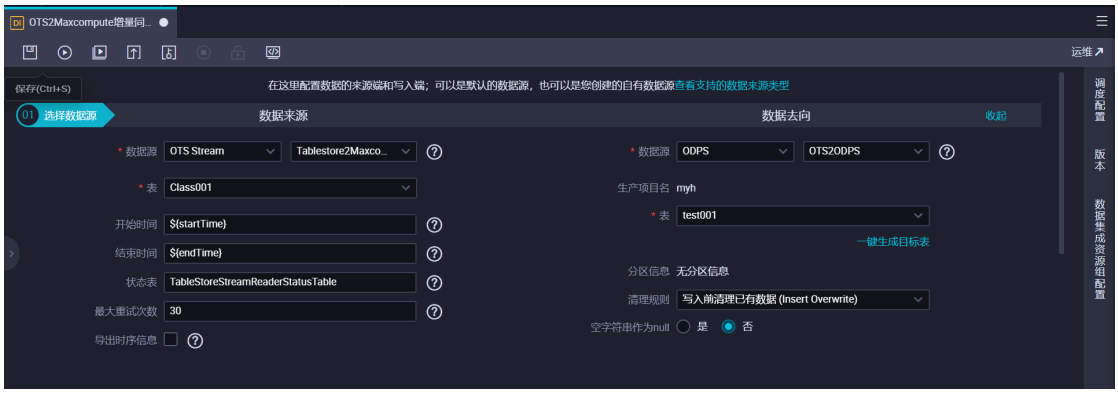
ii. 在同步任务节点的编辑页面的**选择数据源**区域，配置数据来源和数据去向。

- 配置数据来源。

  选择**数据来源**的**数据源**为**OTS Stream**，选择数据源和数据表，可根据需要配置任务开始时间、结束时间、状态表的名称、最大重试次数等。

- 配置数据去向。

  选择**数据去向**的**数据源**为**ODPS**，选择数据源和表，可根据需要配置清理规则、是否把空字符串作为null等。



iii. 单击  图标，进行脚本配置。

使用过程时涉及OTSStream Reader和OSS Writer插件的配置。具体操作，请参见配置OTSStream Reader和配置MaxCompute Writer。

在脚本配置页面，请根据如下示例完成配置。

```
{
    "type": "job",
    "steps": [
        {
            "stepType": "otsstream",     # Reader插件的名称。
            "parameter": {
                "mode": "single_version_and_update_only",     # 配置导出模式，默认不设置，为列模式。
                "statusTable": "TableStoreStreamReaderStatusTable",     # 存储TableStore Stream状态的表，一般无需修改。
                "maxRetries": 30,     # 从TableStore中读取增量数据时，每次请求的最大重试次数，默认为30。重试之间有间隔，重试30次的总时间约为5分钟，一般无需修改。
                "isExportSequenceInfo": false,     # 是否导出时序信息，时序信息包含了数据的写入时间等。默认该值为false，即不导出。single_version_and_update_only模式下只能为false。
                "datasource": "",     # Tablestore的数据源名称，如果有此项则无需配置endpoint、accessId、accessKey和instanceName。
                "column": [     # 按照需求添加需要导出TableStore中的列，您可以自定义个数。
                    {
                        "name": "h"     # 列名示例，可以是主键或属性列。
                    },
                    {
                        "name": "n"
                    },
                    {
```

```
                        "name": "s"
                    }
                ],
                "startTimeString": "${startTime}",     # 增量数据的时间范围（左闭右开）的左边界，格式为yyyymmddhh24miss，单位毫秒。输入$[yyyymmddhh24miss-10/24/60]，表示调度时间减去10分钟。
                "table": "",     # TableStore中的表名。
                "endTimeString": "${endTime}"     # 增量数据的时间范围（左闭右开）的右边界，格式为yyyymmddhh24miss，单位为毫秒。输入$[yyyymmddhh24miss-5/24/60]，表示调度时间减去5分钟。注意，endTime需要比调度时间提前5分钟及以上。
            },
            "name": "Reader",
            "category": "reader"
        },
        {
            "stepType": "odps",     # Writer插件的名称。
            "parameter": {
                "partition": "",     # 需要写入数据表的分区信息。
                "truncate": false,     # 清理规则，写入前是否清理已有数据。
                "compress": false,     # 是否压缩。
                "datasource": "",     # 数据源名。
                "column": [     # 需要导入的字段列表。
                    "h",
                    "n",
                    "s"
                ],
                "emptyAsNull": false,     # 空字符串是否作为null，默认是。
                "table": ""     # 表名。
            },
            "name": "Writer",
            "category": "writer"
        }
    ],
    "version": "2.0",
    "order": {
        "hops": [
            {
                "from": "Reader",
                "to": "Writer"
            }
        ]
    },
    "setting": {
        "errorLimit": {
            "record": "0"     # 允许出错的个数，当错误超过这个数目的时候同步任务会失败。
        },
        "speed": {
            "throttle": false,     #同步速率不限流。
            "concurrent": 2     # 每次同步任务的并发度。
        }
    }
}
```

iv. 单击  图标，保存数据源配置。

5. 运行同步任务。

i. 单击  图标。

ii. 在**参数**对话框，选择调度的资源组。

iii. 单击**确定**，开始运行任务。

运行结束后，在**运行日志**页签中可以查看任务是否成功和导出的数据行数。

表格存储的增量数据可以在延迟5~10分钟的基础上自动同步到MaxCompute中。

6. 配置调度参数。

通过**调度配置**，可以配置同步任务的执行时间、重跑属性、调度依赖等。

i. 在**数据集成**节点下，双击同步任务节点。

ii. 在同步任务节点的编辑页面的右侧单击**调度配置**，进行调度参数配置，详情请参见配置调度和依赖属性。

7. 提交同步任务。

i. 在同步任务节点的编辑页面，单击  图标。

ii. 在**提交新版本**对话框，输入备注信息。

iii. 单击**确认**。

将同步任务提交到调度系统后，调度系统会根据配置的调度参数，自动定时执行同步任务。

步骤四：查看同步任务

1. 进入运维中心。

> **说明** 您也可以在DataStudio控制台的右上角单击**运维中心**，快速进入运维中心。

i. 以项目管理员身份登录DataWorks控制台。

ii. 选择地域，在左侧导航栏，单击**工作空间列表**。

iii. 在**工作空间列表**页面，单击工作空间操作区域的**进入运维中心**。

2. 在运维中心控制台，选择**周期任务运维 > 周期任务**。

3. 在**周期任务**页面，查看提交的同步任务详情。

- 在左侧导航栏中，选择**周期任务运维 > 周期实例**，可以查看当天需要运行的周期任务。单击实例名称，可以查看任务运行详情。
- 当单个任务在运行中或运行结束后，可以查看日志。

步骤五：查看导入到MaxCompute中的数据

1. 进入数据地图。

i. 以项目管理员身份登录DataWorks控制台。

ii. 选择地域，在左侧导航栏，单击**工作空间列表**。

iii. 在**工作空间列表**页面，单击工作空间操作区域的**进入数据地图**。

2. 在数据地图控制台的导航栏，选择**我的数据 > 我管理的数据**。
3. 在**我管理的数据**页签，单击导入数据的表名称。
4. 在表详情页面，单击**数据预览**页签，查看导入的数据。

3.2.3. 将表格存储的增量数据转换为全量数据格式

通过DataWorks控制台，您可以在MaxCompute中使用merge_udf.jar包将表格存储的增量数据转换为全量数据格式。

前提条件

- 已导出表格存储全量数据到MaxCompute，且已配置同步表格存储增量数据到MaxCompute。具体操作，请分别参见全量导出（脚本模式）和增量同步（脚本模式）。
- 已下载merge_udf.jar包。具体下载路径请参见merge_udf.jar。

步骤一：新建JAR资源

通过新建JAR资源，将下载的merge_udf.jar包上传到MaxCompute中。

1. 进入**数据开发**页面。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 选择工作空间所在地域后，单击相应工作空间后的**进入数据开发**。
2. 新建JAR资源。
   i. 在**数据开发**页面，将鼠标悬停在+新建图标，选择**MaxCompute > 资源 > JAR**。

      > **注意** 只有在**工作空间配置**页面添加MaxCompute引擎后，当前页面才会显示MaxCompute节点。具体操作，请参见配置工作空间。

      您也可以打开相应的业务流程，右键单击**MaxCompute**，选择**新建 > 资源 > JAR**。

      如果您需要创建业务流程，请参见创建业务流程。

   ii. 在**新建资源**对话框，填写**资源名称**，并选择**目标文件夹**。

      > **说明**
      > - 资源名称无需与上传的文件名保持一致，JAR资源的后缀必须为.jar。
      > - 如果该JAR包已经在MaxCompute（ODPS）客户端上传过，则需要取消选择**上传为ODPS资源**，否则上传会报错。

   iii. 单击**点击上传**，选择相应的文件进行上传。
   iv. 单击**新建**。

3. 在资源编辑页面，提交资源到调度开发服务器端。
   i. 单击工具栏中的图标。
   ii. 在**提交新版本**对话框，输入变更描述。
   iii. 单击**确认**。

步骤二：新建并注册函数

1. 新建函数。
   i. 在**数据开发**页面，将鼠标悬停在+新建图标，选择**MaxCompute > 函数**。

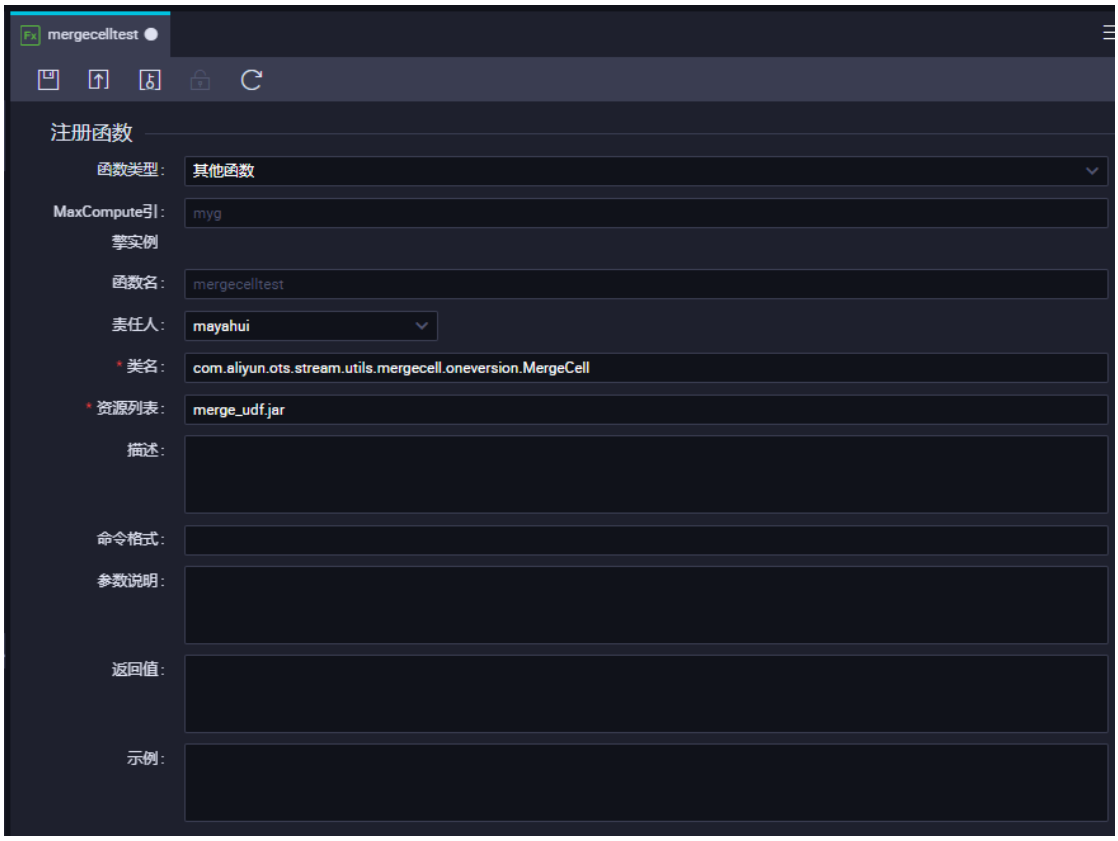
      您也可以打开相应的业务流程，右键单击**MaxCompute**，选择**新建 > 函数**。
   ii. 在**新建函数**对话框，输入**函数名称**，并选择**目标文件夹**。
   iii. 单击**新建**。

2. 注册函数。
   i. 在**注册函数**页面，选择函数类型为**其他函数**。

ii. 填写资源列表为步骤一：新建JAR资源中的资源名称，并根据表类型和模式填写对应的类名。

根据表类型不同可以选择的模式不同，单版本表只能选择单版本模式，多版本表可以选择多版本模式V1或者多版本模式V2。关于模式选择的更多信息，请参见附录：模式选择。

表类型	模式	类名
单版本表	单版本模式	com.aliyun.ots.stream.utils.mergecell.oneversion.MergeCell
多版本表	多版本模式V1	com.aliyun.ots.stream.utils.mergecell.multiversion.MergeCellV1
	多版本模式V2	com.aliyun.ots.stream.utils.mergecell.multiversion.MergeCellV2



3. 在函数编辑页面，提交函数到调度开发服务器端。

i. 单击工具栏中的图标。

ii. 在**提交新版本**对话框，输入变更描述。

iii. 单击**确认**。

步骤三：编写ODPS SQL并运行

1. 新建ODPS SQL节点。

i. 在**数据开发**页面，将鼠标悬停在+新建图标，选择**MaxCompute > ODPS SQL**。

您也可以打开相应的业务流程，右键单击**MaxCompute**，选择**新建 > ODPS SQL**。

ii. 在**新建节点**对话框，填写节点名称，并选择目标文件夹。

iii. 单击**提交**。

2. 在节点编辑页面，编写ODPS SQL语句并运行。

   ODPS SQL语句的语法如下：

```
select function_name(para_list)
 as (custom_para_list)
from(
    select * from stream_table_name distribute by primary_keys sort by primary_keys,Seq
uenceID
)t;
```

   其中*function_name*为步骤二：新建并注册函数中的函数名称，*para_list*为附录：模式选择中的参数列表，*custom_para_list*为自定义参数列表，*stream_table_name*为增量表的名称，*primary_keys*为表格存储数据表的主键列表，*SequenceID*为增量表的sequenceid。

附录：模式选择

模式包括单版本模式、多版本模式V1和多版本模式V2，请根据表类型选择合适的模式。

- 单版本模式
  - 类名：com.aliyun.ots.stream.utils.mergecell.oneversion.MergeCell

- 参数列表

  参数列表的格式为 `pknum,colnum,colnames,pknames,colname,version,colvalue,optype,sequenceinfo` 。具体参数说明请参见下表，请根据实际填写pknum、colnum、colnames和pknames，其他参数保持参数名称即可。

参数	类型	描述
INT pknum	常量	表格存储数据表的主键个数。
INT colnum	常量	表格存储数据表中的属性列个数。
List<String> colnames	常量	要合并增量变更的属性列名称。
List<String> pknames	变量	表格存储数据表的主键列表。
STRING colname	变量	属性列。
BIGINT version	变量	版本号。
STRING colvalue	变量	增量值。
STRING optype	变量	增量操作类型。
STRING sequenceinfo	变量	自增保序sequenceid。

- 示例

  当表格存储数据表的主键为 `pk1,pk2` 且属性列为 `col1,col2,col3` 时，则在MaxCompute中创建的增量表Schema为 `pk1,pk2,colname,version,colvalue,optype,sequenceinfo` ，如果要合并增量变更的属性列为 `col1,col2,col3` ，设置参数列表为 `2,3,"col1","col2","col3",pk1,pk2,colname,version,colvalue,optype,sequenceinfo` ，ODPS SQL语句示例如下：

```
select mergeCell(2,3,"col1","col2","col3",pk1,pk2,colname,version,colvalue,optype,sequenceinfo)
 as (pk1,pk2,col1,col1_is_deleted,col2,col2_is_deleted,col3,col3_is_deleted)
from(
    select * from stream_table_name distribute by pk1,pk2 sort by pk1,pk2,sequenceInfo
)t;
```

  执行ODPS SQL语句后的输出结果请参见下表。

pk1	pk2	col1	co1_is_deleted	col2	col2_is_deleted	col3	col3_is_deleted
test	0	\N	\N	20	\N	\N	true

  输出结果的说明如下：

  - 当col列和col_is_deleted列均为 `\N` 时，表示col列无任何增量操作。
  - 当col列为具体的值且col_is_deleted列为 `\N` 时，表示col列的值被修改为对应值。
  - 当col列为 `\N` 且col_is_deleted列为true时，表示col列被删除。

- 多版本模式V1

- 类名：com.aliyun.ots.stream.utils.mergecell.multiversion.MergeCellV1
- 参数列表

  参数列表的格式为 `pknum,colnum,colnames,pknames,colname,version,colvalue,optype,sequenceinfo` 。具体参数说明请参见下表，请根据实际填写pknum、colnum、colnames和pknames，其他参数保持参数名称即可。

参数	类型	描述
INT pknum	常量	表格存储数据表的主键个数。
INT colnum	常量	表格存储数据表中的属性列个数。
List<String> colnames	常量	要合并增量变更的属性列名称。
List<String> pknames	变量	表格存储数据表的主键列表。
STRING colname	变量	属性列。
BIGINT version	变量	版本号。
STRING colvalue	变量	增量值。
STRING optype	变量	增量操作类型。
STRING sequenceinfo	变量	自增保序sequenceid。

- 示例

  当表格存储数据表的主键为 `pk1,pk2` 且属性列为 `col1,col2,col3` 时，则在MaxCompute中创建的增量表Schema为 `pk1,pk2,colname,version,colvalue,optype,sequenceinfo` ，如果要合并增量变更的属性列为 `col1,col2,col3` ，设置参数列表为 `2,3,"col1","col2","col3",pk1,pk2,colname,version,colvalue,optype,sequenceinfo` ，ODPS SQL语句示例如下：

```
select mergeCell(2,3,"col1","col2","col3",pk1,pk2,colname,version,colvalue,optype,sequenceinfo)
 as (pk1,pk2,version,col1,col1_is_deleted,col2,col2_is_deleted,col3,col3_is_deleted)
from(
    select * from stream_table_name distribute by pk1,pk2 sort by pk1,pk2,sequenceinfo
)t;
```

  执行ODPS SQL语句后的输出结果请参见下表。

pk1	pk2	version	col1	co1_is_deleted	col2	col2_is_deleted	col3	col3_is_deleted
test	0	123	\N	\N	20	\N	\N	true

  输出结果的说明如下：

  - 当version列为具体的值，且col列和col_is_deleted列均为 `\N` 时，表示col列对应版本没有任何增量操作。
  - 当version列和col列均为具体的值，且col_is_deleted列为 `\N` 时，表示col列对应版本的值被修改为具体的值。
  - 当version列为具体的值，col列为 `\N` ，且col_is_deleted列为true时，表示col列对应版本被删除。
  - 当version列和col列均为 `\N` ，且col_is_deleted列为true时，表示存在删除一列所有版本的操作。

- 多版本模式V2
  - 类名：com.aliyun.ots.stream.utils.mergecell.multiversion.MergeCellV2

- 参数列表

  参数列表的格式为`pknum,colnum,maxversion,colnames,pknames,colname,version,colvalue,optype,sequenceinfo`。具体参数说明请参见下表，请根据实际填写pknum、colnum、maxversion、colnames和pknames，其他参数保持参数名称即可。

参数	类型	描述
BIGINT pknum	常量	表格存储数据表的主键个数。
BIGINT colnum	常量	表格存储数据表中的属性列个数。
BIGINT maxversion	常量	最大版本数。
List<String> colnames	常量	要合并增量变更的属性列名称。
List<String> pknames	变量	表格存储数据表的主键列表。
STRING colname	变量	属性列。
BIGINT version	变量	版本号。
STRING colvalue	变量	增量值。
STRING optype	变量	增量操作类型。
STRING sequenceinfo	变量	自增保序sequenceid。

- 示例

  当表格存储数据表的主键为 `pk1,pk2` ，属性列为 `col1,col2,col3` 且最大版本数为3时，则在MaxCompute中创建的增量表Schema为 `pk1,pk2,colname,version,colvalue,optype,sequenceinfo` ，如果要合并增量变更的属性列为 `col1,col2,col3` ，设置参数列表为 `2,3,3,"col1","col2","col3",pk1,pk2,colName,version,colValue,opType,sequenceInfo` ，ODPS SQL语句示例如下：

```
select mergeCell(2,3,3,"col1","col2","col3",pk1,pk2,colname,version,colvalue,opyype,sequenceinfo)
 as (pk1,pk2,col1,col2,col3)
from(
    select * from stream_table_name distribute by pk1,pk2 sort by pk1,pk2,sequenceinfo
)t;
```

执行ODPS SQL语句后的输出结果请参见下表。

pk1	pk2	col1	col2	col3
test	02	{"data":[{"version":1621330803390,"value":"value001"},{"version":1621330795198,"value":"value002"},{"version":1621330785936,"value":"value003"}],"needDeleteAllVersionFirst":true,"deleteVersions":[]}	\N	\N

输出结果的说明如下：

- data表示新写入的数据列表，按照版本号降序排序。最多保留maxversion个版本的数据。
- needDeleteAllVersionFirst表示该列是否需要删除原有全部版本。当出现删除一行DeleteRow或删除一列的所有版本DeleteColumns时，该值为true，否则为false。
- deleteVersions表示该列需要删除的版本列表，按照版本号降序排序。最多保留maxversion个版本。

  deleteVersions中的版本号不会与data中的版本号相同，当needDeleteAllVersionFirst为true时，deleteVersions为空列表。