ALIBABA CLOUD

Alibaba Cloud

表格存储

資料通道

Document Version: 20220630

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.資料匯入 05
1.1. 將Kafka資料同步到Tablestore 05
1.1.1. 概述 05
1.1.2. 同步資料到資料表 06
1.1.3. 同步資料到時序表 13
1.1.4. 配置說明 17
1.1.5. 錯誤處理 35
1.2. 將Tablestore資料表中資料同步到另一個資料表中 36
2.資料匯出 39
2.1. 將Tablestore資料同步到OSS 39
2.1.1. 概述 39
2.1.2. 全量匯出（指令碼模式） 40
2.1.3. 增量同步處理（指令碼模式） 45
2.2. 將Tablestore資料同步到MaxCompute 49
2.2.1. 全量匯出（指令碼模式） 49

1.資料匯入

1.1. 將Kafka資料同步到Tablestore

1.1.1. 概述

基於Tablestore Sink Connector，您可以將Apache Kafka中的資料大量匯入到Tablestore的資料表或者時序表中。

背景資訊

Kafka是一個分布式訊息佇列系統，不同的資料系統可以通過Kafka Connect工具將資料流輸入Kafka和從Kafka擷取資料流。

Tablestore團隊基於Kafka Connect開發了Tablestore Sink Connector。Tablestore Sink Connector會根據訂閱的主題（Topic）輪詢地從Kafka中拉取訊息記錄（Record），並對訊息記錄進行解析，然後將資料大量匯入到Tablestore。該Connector最佳化了資料匯入處理程序，並且支援個人化配置。

Tablestore是阿里雲自研的多模型結構化資料存放區，支援多種資料模型，包括寬表模型和時序模型。您可以將Kafka資料同步到Tablestore中的資料表（寬表模型中的表類型）或者時序表（時序模型中的表類型）。具體操作，請分別參見同步資料到資料表和同步資料到時序表。

功能特性

Tablestore Sink Connector的主要功能特性如下：

- 至少交付一次

  保證Kafka訊息記錄從Kafka主題向Tablestore至少交付一次。

- 資料對應

  Kafka主題中的資料先通過Converter進行還原序列化，您需要在Kafka Connect的worker配置或者connetor配置中修改key.converter和value.converter屬性，以確保配置合適的還原序列化轉換器。您可以選擇Kafka Connect帶有的JsonConverter，也可以選擇由第三方提供的其它Converter或者自訂Converter。

- 自動建立目標表

  當目標表缺失時，支援根據配置的主鍵列和屬性列白名單（如果有）自動建立目標表。

- 錯誤處理策略

  由於匯入資料時為大量操作，其中部分訊息記錄可能發生解析錯誤或者寫入錯誤。此時，您可以選擇立即終止任務或者忽略這些錯誤，您還可以選擇將產生錯誤的訊息記錄和錯誤資訊記錄在Kafka訊息系統中或者Tablestore中。

工作模式

Tablestore Sink Connector具有standalone（獨立）模式和distributed（分布式）模式兩種工作模式，請根據實際需要選擇。

- 在standalone模式下，所有任務都將在單個進程中執行，此模式更易於配置和使用。您可以使用standalone模式瞭解Tablestore Sink Connector的各種功能。
- 在distributed模式下，所有任務通過多個進程並存執行，此模式支援根據進程變化自動均衡任務以及在執行任務過程中提供容錯能力，穩定性更好。建議您使用distributed模式。

1.1.2. 同步資料到資料表

Tablestore Sink Connector會根據訂閱的主題輪詢地從Kafka中拉取訊息，並對訊息記錄進行解析，然後將資料大量匯入到Tablestore的資料表。

前提條件

- 已安裝Kafka，並且已啟動ZooKeeper和Kafka。更多資訊，請參見Kafka官方文檔。
- 已開通Tablestore服務，建立執行個體以及建立資料表。具體操作，請參見快速使用寬表模型。

  > **說明** 您也可以通過Tablestore Sink Connector自動建立目標資料表，此時需要配置auto.create為true。

- 已擷取AccessKey。具體操作，請參見擷取AccessKey。

步驟一：部署Tablestore Sink Connector

1. 通過以下任意一種方式擷取Tablestore Sink Connector。
   - 通過GitHub下載源碼並編譯。源碼的GitHub路徑為Tablestore Sink Connector源碼。
     a. 通過Git工具執行以下命令下載Tablestore Sink Connector源碼。

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

     b. 進入到下載的源碼目錄後，執行以下命令進行Maven打包。

     ```
     mvn clean package -DskipTests
     ```

     編譯完成後，產生的壓縮包（例如kafka-connect-tablestore-1.0.jar）會存放在target目錄。
   - 直接下載編譯完成的kafka-connect-tablestore壓縮包。
2. 將壓縮包複製到各個節點的$KAFKA_HOME/libs目錄下。

步驟二：啟動Tablestore Sink Connector

Tablestore Sink Connector具有standalone模式和distributed模式兩種工作模式。請根據實際選擇。

standalone模式的配置步驟如下：

1. 根據實際修改worker設定檔connect-standalone.properties和connetor設定檔connect-tablestore-sink-quickstart.properties。
   - worker設定檔connect-standalone.properties的配置樣本

     worker配置中包括Kafka串連參數、序列化格式、提交位移量的頻率等配置項。此處以Kafka官方樣本為例介紹。更多資訊，請參見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設定檔connect-tablestore-sink-quickstart.properties的配置樣本

  connetor配置中包括連接器類、Tablestore串連參數、資料對應等配置項。更多資訊，請參見配置說明。

```
# 設定連接器名稱。
name=tablestore-sink
# 指定連接器類。
connector.class=TableStoreSinkConnector
# 設定最大任務數。
tasks.max=1
# 指定匯出資料的Kafka的Topic列表。
topics=test
# 以下為Tablestore串連參數配置。
# Tablestore執行個體的Endpoint。
tablestore.endpoint=https://xxx.xxx.ots.aliyuncs.com
# 填寫AccessKey ID和AccessKey Secret。
tablestore.access.key.id =xxx
tablestore.access.key.secret=xxx
# Tablestore執行個體名稱。
tablestore.instance.name=xxx
# 用於指定Tablestore目標表名稱的格式字串，其中<topic>作為原始Topic的預留位置。預設值為<topic>
。
# Examples:
# table.name.format=kafka_<topic>，主題為test的訊息記錄將寫入表名為kafka_test的資料表。
# table.name.format=
# 主鍵模式，預設值為kafka。
# 將以<topic>_<partition>（Kafka主題和分區，用"_"分隔）和<offset>（訊息記錄在分區中的位移量
）作為Tablestore資料表的主鍵。
# primarykey.mode=
# 自動建立目標表，預設值為false。
auto.create=true
```

2. 進入到$KAFKA_HOME目錄後，執行以下命令啟動standalone模式。

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

distributed模式的配置步驟如下：

1. 根據實際修改worker設定檔connect-distributed.properties。

   worker配置中包括Kafka串連參數、序列化格式、提交位移量的頻率等配置項，還包括儲存各connectors相關資訊的Topic，建議您提前手動建立相應Topic。此處以Kafka官方樣本為例介紹。更多資訊，請參見Kafka Connect。

   - offset.storage.topic：用於儲存各connectors相關offset的Compact Topic。
   - config.storage.topic：用於儲存connector和task相關配置的Compact Topic，此Topic的Parition數必須設定為1。
   - status.storage.topic：用於儲存kafka connect狀態資訊的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.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. 進入到$KAFKA_HOME目錄後，執行以下命令啟動distributed模式。

   注意 您需要在每個節點上均啟動worker進程。

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

3. 通過REST API管理connectors。更多資訊，請參見REST API。

i. 在config路徑下建立connect-tablestore-sink-quickstart.json檔案並填寫以下樣本內容。

connetor設定檔以JSON格式串指定參數索引值對，包括連接器類、Tablestore串連參數、資料對應等配置項。更多資訊，請參見配置說明。

```
{
  "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. 執行以下命令啟動一個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` 為Kafka REST服務的地址，請根據實際修改。

步驟三：生產新的記錄

1. 進入到$KAFKA_HOME目錄後，執行以下命令啟動一個控制台生產者。

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

配置項說明請參見下表。

配置項	樣本值	描述
--broker-list	localhost:9092	Kafka叢集broker地址和連接埠。
--topic	test	主題名稱。啟動Tablestore Sink Connetor時預設會自動建立Topic，您也可以選擇手動建立。

2. 向主題test中寫入一些新的訊息。
   - Struct類型訊息

```
{
    "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類型訊息

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

3. 登入Tablestore控制台查看資料。

   Tablestore執行個體中將自動建立一張資料表，表名為test，表中資料如下圖所示。其中第一行資料為Map類型訊息匯入結果，第二行資料為Struct類型訊息匯入結果。

	Details	topic_partition(Primary Key)	offset(Primary Key)	id	price	product	quantity
	Details	test_3	0	AAAAAQ==			
	Details	test_34	0	1	50.0	foo	100

1.1.3. 同步資料到時序表

您可以使用kafka-connect-tablestore包將Kafka中資料寫入Tablestore的時序表中。本文主要介紹了如何配置Kafka寫入時序資料。

前提條件

- 已安裝Kafka，並且已啟動ZooKeeper和Kafka。更多資訊，請參見Kafka官方文檔。
- 已開通Tablestore服務，建立執行個體以及建立時序表。具體操作，請參見快速使用時序模型。

  > **說明** 您也可以通過Tablestore Sink Connector自動建立目標時序表，此時需要配置auto.create為true。

- 已擷取AccessKey。具體操作，請參見擷取AccessKey。

背景信息

Tablestore支援對時序資料進行儲存以及分析。更多資訊，請參見時序模型概述。

步驟一：部署Tablestore Sink Connector

1. 通過以下任意一種方式擷取Tablestore Sink Connector。
   - 通過GitHub下載源碼並編譯。源碼的GitHub路徑為Tablestore Sink Connector源碼。

a. 通過Git工具執行以下命令下載Tablestore Sink Connector源碼。

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

b. 進入到下載的源碼目錄後，執行以下命令進行Maven打包。

```
mvn clean package -DskipTests
```

編譯完成後，產生的壓縮包（例如kafka-connect-tablestore-1.0.jar）會存放在target目錄。

- 直接下載編譯完成的kafka-connect-tablestore壓縮包。

2. 將壓縮包複製到各個節點的$KAFKA_HOME/libs目錄下。

步驟二：啟動Tablestore Sink Connector

Tablestore Sink Connector具有standalone模式和distributed模式兩種工作模式。請根據實際選擇。

由於寫入時序資料時，Kafka側的訊息記錄必須為JSON格式，因此啟動Tablestore Sink Connector時需要使用Jsonconverter，且不需要提取schema以及不需要輸入key，請在connect-standalone.properties和connect-distributed.properties中按照如下樣本配置對應配置項。

> **說明** 如果輸入了key，請按照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
```

此處以配置standalone模式為例介紹，distributed模式的配置步驟與同步資料到資料表時的distributed模式配置步驟類似，只需按照上述樣本在worker設定檔connect-distributed.properties中修改對應配置項以及在connetor檔案connect-tablestore-sink-quickstart.json中修改時序相關配置即可。具體操作，請參見步驟二：啟動Tablestore Sink Connector中distributed模式的配置步驟。

standalone模式的配置步驟如下：

1. 根據實際修改worker設定檔connect-standalone.properties和connetor設定檔connect-tablestore-sink-quickstart.properties。
   - worker設定檔connect-standalone.properties的配置樣本

     worker配置中包括Kafka串連參數、序列化格式、提交位移量的頻率等配置項。此處以Kafka官方樣本為例介紹。更多資訊，請參見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設定檔connect-tablestore-sink-quickstart.properties的配置樣本

  connetor配置中包括連接器類、Tablestore串連參數、資料對應等配置項。更多資訊，請參見配置說明。

```
# 設定連接器名稱。
name=tablestore-sink
# 指定連接器類。
connector.class=TableStoreSinkConnector
# 設定最大任務數。
tasks.max=1
```

```
tasks.max=1
# 指定匯出資料的Kafka的Topic列表。
topics=test
# 以下為Tablestore串連參數的配置。
# Tablestore執行個體的Endpoint。
tablestore.endpoint=https://xxx.xxx.ots.aliyuncs.com
# 指定認證模式。
tablestore.auth.mode=aksk
# 填寫AccessKey ID和AccessKey Secret。如果使用aksk認證，則需要填入這兩項。
tablestore.access.key.id=xxx
tablestore.access.key.secret=xxx
# 指定Tablestore執行個體名稱。
tablestore.instance.name=xxx
## STS認證相關配置，如果使用STS認證，則下列各項必填。此外aksk還需要在環境變數中配置配入ACCESS_ID和ACCESS_KEY。
#sts.endpoint=
#region=
#account.id=
#role.name=
# 定義目標表名稱的格式字串，字串中可包含<topic>作為原始Topic的預留位置。
# topics.assign.tables配置的優先順序更高，如果配置了topics.assign.tables，則忽略table.name.format的配置。
# 例如當設定table.name.format為kafka_<topic>時，如果kafka中主題名稱為test，則將映射到Tablestore的表名為kafka_test。
table.name.format=<topic>
# 指定Topic與目標表的映射關係，以"<topic>:<tablename>"格式映射Topic和表名，Topic和表名之間的分隔字元為半形冒號（:），不同映射之間分隔字元為半形逗號（,）。
# 如果預設，則採取table.name.format的配置。
# topics.assign.tables=test:test_kafka
# 是否自動建立目標表，預設值為false。
auto.create=true
# 以下為髒資料處理相關配置。
# 在解析Kafka Record或者寫入時序表時可能發生錯誤，您可以可通過以下配置進行處理。
# 指定容錯能力，可選值包括none和all，預設值為none。
# none表示任何錯誤都將導致Sink Task立即失敗。
# all表示跳過產生錯誤的Record，並記錄該Record。
runtime.error.tolerance=none
# 指定髒資料記錄模式，可選值包括ignore、kafka和tablestore，預設值為ignore。
# ignore表示忽略所有錯誤。
# kafka表示將產生錯誤的Record和錯誤資訊儲存在Kafka的另一個Topic中。
# tablestore表示將產生錯誤的Record和錯誤資訊儲存在Tablestore另一張資料表中。
runtime.error.mode=ignore
# 當髒資料記錄模式為kafka時，需要配置Kafka叢集地址和Topic。
# runtime.error.bootstrap.servers=localhost:9092
# runtime.error.topic.name=errors
# 當髒資料記錄模式為tablestore時，需要配置Tablestore中資料表名稱。
# runtime.error.table.name=errors
##以下為時序表新增配置。
# connector工作模式，預設為normal。
tablestore.mode=timeseries
# 時序表主鍵欄位對應。
tablestore.timeseries.test.measurement=m
tablestore.timeseries.test.dataSource=d
tablestore.timeseries.test.tags=region,level
# 時序表時間欄位對應。
```

```
tablestore.timeseries.test.time=timestamp
tablestore.timeseries.test.time.unit=MILLISECONDS
# 是否將時序資料欄位（field）的列名轉為小寫，預設為true。由於當前時序模型中時序表的列名不支援大寫字母，如果配置為false，且列名中有大寫字母，寫入會報錯。
tablestore.timeseries.toLowerCase=true
# 是否將所有非主鍵以及時間的欄位以field的形式儲存在時序表，預設為true，如果為false，則只儲存tablestore.timeseries.test.field.name中配置的欄位
tablestore.timeseries.mapAll=true
# 配置field欄位名稱，多個欄位名稱之間用半形冒號（,）分隔。
tablestore.timeseries.test.field.name=cpu
# 配置field欄位類型。取值範圍為double、integer、string、binary和boolean。
# 當field中包含多個欄位時，欄位類型必須和欄位名稱一一對應。多個欄位類型之間用半形冒號（,）分隔。
tablestore.timeseries.test.field.type=double
```

2. 進入到$KAFKA_HOME目錄後，執行以下命令啟動standalone模式。

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

步驟三：生產新的記錄

1. 進入到$KAFKA_HOME目錄後，執行以下命令啟動一個控制台生產者。

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

配置項說明請參見下表。

配置項	樣本值	描述
--broker-list	localhost:9092	Kafka叢集broker地址和連接埠。
--topic	test	主題名稱。啟動Tablestore Sink Connetor時預設會自動建立Topic，您也可以選擇手動建立。

2. 向主題test中寫入一些新的訊息。

> **注意** 如果要匯入資料到時序表，則向主題中寫入資料時必須輸入JSON格式的資料。

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

3. 登入Tablestore控制台查看資料。

1.1.4. 配置說明

啟動Tablestore Sink Connector時，您需要通過索引值映射向Kafka Connect進程傳遞參數。通過本文您可以結合配置樣本和配置參數說明瞭解Tablestore Sink Connector的相關配置。

配置樣本

當從Kafka同步資料到資料表或者時序表時配置項不同，且不同工作模式下相應設定檔的樣本不同。此處以同步資料到資料表中為例介紹配置樣本。同步資料到時序表的配置樣本中需要增加時序相關配置項。

- .properties格式設定檔的樣本，適用於standalone模式。

```
# 設定連接器名稱。
name=tablestore-sink
# 指定連接器類。
connector.class=TableStoreSinkConnector
# 設定最大任務數。
tasks.max=1
# 指定匯出資料的Kafka的Topic列表。
topics=test
# 以下為Tablestore串連參數的配置。
# Tablestore執行個體的Endpoint。
tablestore.endpoint=https://xxx.xxx.ots.aliyuncs.com
# 填寫AccessKey ID和AccessKey Secret。
tablestore.access.key.id=xxx
tablestore.access.key.secret=xxx
# Tablestore執行個體名稱。
tablestore.instance.name=xxx
# 以下為資料對應相關的配置。
# 指定Kafka Record的解析器。
# 預設的DefaulteEventParser已支援Struct和Map類型，您也可以使用自訂的EventParser。
event.parse.class=com.aliyun.tablestore.kafka.connect.parsers.DefaultEventParser
# 定義目標表名稱的格式字串，字串中可包含<topic>作為原始Topic的預留位置。
# topics.assign.tables配置的優先順序更高，如果配置了topics.assign.tables，則忽略table.name.format的配置。
# 例如當設定table.name.format為kafka_<topic>時，如果kafka中主題名稱為test，則將映射到Tablestore的表名為kafka_test。
table.name.format=<topic>
# 指定Topic與目標表的映射關係，以"<topic>:<tablename>"格式映射Topic和表名，Topic和表名之間的分隔字元為英文冒號（:），不同映射之間分隔字元為英文逗號（,）。
# 如果預設，則採取table.name.format的配置。
# topics.assign.tables=test:test_kafka
# 指定主鍵模式，可選值包括kafka、record_key和record_value，預設值為kafka。
# kafka表示以<connect_topic>_<connect_partition>和<connect_offset>作為資料表的主鍵。
# record_key表示以Record Key中的欄位作為資料表的主鍵。
# record_value表示以Record Value中的欄位作為資料表的主鍵。
primarykey.mode=kafka
# 定義匯入資料表的主鍵列名和資料類型。
# 屬性名稱格式為tablestore.<tablename>.primarykey.name和tablestore.<tablename>.primarykey.type。
# 其中<tablename>為資料表名稱的預留位置。
# 當主鍵模式為kafka時，無需配置該屬性，預設主鍵列名為{"topic_partition","offset"}，預設主鍵列資料類型為{string, integer}。
# 當主鍵模式為record_key或record_value時，必須配置以下兩個屬性。
# tablestore.test.primarykey.name=A,B
# tablestore.test.primarykey.type=string,integer
# 定義屬性列白名單，用於過濾Record Value中的欄位擷取所需屬性列。
# 預設值為空白，使用Record Value中的所有欄位作為資料表的屬性列。
# 屬性名稱格式為tablestore.<tablename>.columns.whitelist.name和tablestore.<tablename>.columns.whitelist.type。
# 其中<tablename>為資料表名稱的預留位置。
# tablestore.test.columns.whitelist.name=A,B
# tablestore.test.columns.whitelist.type=string,integer
# 以下為寫入Tablestore相關的配置。
# 指定寫入模式，可選值包括put和update，預設值為put
```

```
# 指定寫入模式，可選值包括put和update，預設值為put。
# put表示覆蓋寫。
# update表示更新寫。
insert.mode=put
# 是否需要保序，預設值為true。如果關閉保序模式，則有助於提升寫入效率。
insert.order.enable=true
# 是否自動建立目標表，預設值為false。
auto.create=false
# 指定刪除模式，可選值包括none、row、column和row_and_column，預設值為none。
# none表示不允許進行任何刪除。
# row表示允許刪除行。
# column表示允許刪除屬性列。
# row_and_column表示允許刪除行和屬性列。
delete.mode=none
# 寫入資料表時記憶體中緩衝隊列的大小，預設值為1024，單位為行數。此配置項的值必須為2的指數。
buffer.size=1024
# 寫入資料表時的回調線程數，預設值為核心數+1。
# max.thread.count=
# 寫入資料表時的最大請求並發數，預設值為10。
max.concurrency=10
# 寫入資料表時的分桶數，預設值為3。適當調大此配置項的值可提升並發寫入能力，但不應大於最大請求並發數。
bucket.count=3
# 寫入資料表時對緩衝區的重新整理時間間隔，預設值為10000，單位為毫秒。
flush.Interval=10000
# 以下為髒資料處理相關配置。
# 在解析Kafka Record或者寫入資料表時可能發生錯誤，您可以可通過以下配置進行處理。
# 指定容錯能力，可選值包括none和all，預設值為none。
# none表示任何錯誤都將導致Sink Task立即失敗。
# all表示跳過產生錯誤的Record，並記錄該Record。
runtime.error.tolerance=none
# 指定髒資料記錄模式，可選值包括ignore、kafka和tablestore，預設值為ignore。
# ignore表示忽略所有錯誤。
# kafka表示將產生錯誤的Record和錯誤資訊儲存在Kafka的另一個Topic中。
# tablestore表示將產生錯誤的Record和錯誤資訊儲存在Tablestore另一張資料表中。
runtime.error.mode=ignore
# 當髒資料記錄模式為kafka時，需要配置Kafka叢集地址和Topic。
# runtime.error.bootstrap.servers=localhost:9092
# runtime.error.topic.name=errors
# 當髒資料記錄模式為tablestore時，需要配置Tablestore中資料表名稱。
# runtime.error.table.name=errors
```

- .json格式設定檔的樣本，適用於distributed模式。

```
{
  "name": "tablestore-sink",
  "config": {
    // 指定連接器類。
    "connector.class":"TableStoreSinkConnector",
    // 設定最大任務數。
    "tasks.max":"3",
    // 指定匯出資料的Kafka的Topic列表。
    "topics":"test",
    // 以下為Tablestore串連參數的配置。
    // Tablestore執行個體的Endpoint。
    "tablestore.endpoint":"https://xxx.xxx.ots.aliyuncs.com",
```

```
    // 填寫AccessKey ID和AccessKey Secret。
    "tablestore.access.key.id":"xxx",
    "tablestore.access.key.secret":"xxx",
    // Tablestore執行個體名稱。
    "tablestore.instance.name":"xxx",
    // 以下為資料對應相關的配置。
    // 指定Kafka Record的解析器。
    // 預設的DefaulteEventParser已支援Struct和Map類型，您也可以使用自訂的EventParser。
    "event.parse.class":"com.aliyun.tablestore.kafka.connect.parsers.DefaultEventParser",
    // 定義目標表名稱的格式字串，字串中可包含<topic>作為原始Topic的預留位置。
    // topics.assign.tables配置的優先順序更高。如果配置了topics.assign.tables，則忽略table.name.format的配置。
    // 例如當設定table.name.format為kafka_<topic>時，如果kafka中主題名稱為test，則將映射到Tablestore的表名為kafka_test。
    "table.name.format":"<topic>",
    // 指定Topic與目標表的映射關係，以"<topic>:<tablename>"格式映射Topic和表名，Topic和表名之間的分隔字元為英文冒號（:），不同映射之間分隔字元為英文逗號（,）。
    // 如果預設，則採取table.name.format的配置。
    // "topics.assign.tables":"test:test_kafka",
    // 指定主鍵模式，可選值包括kafka、record_key和record_value，預設值為kafka。
    // kafka表示以<connect_topic>_<connect_partition>和<connect_offset>作為資料表的主鍵。
    // record_key表示以Record Key中的欄位作為資料表的主鍵。
    // record_value表示以Record Value中的欄位作為資料表的主鍵。
    "primarykey.mode":"kafka",
    // 定義匯入資料表的主鍵列名和資料類型。
    // 屬性名稱格式為tablestore.<tablename>.primarykey.name和tablestore.<tablename>.primarykey.type。
    // 其中<tablename>為資料表名稱的預留位置。
    // 當主鍵模式為kafka時，無需配置該屬性，預設主鍵列名為{"topic_partition","offset"}，預設主鍵列資料類型為{string, integer}。
    // 當主鍵模式為record_key或record_value時，必須配置以下兩個屬性。
    // "tablestore.test.primarykey.name":"A,B",
    // "tablestore.test.primarykey.type":"string,integer",
    // 定義屬性列白名單，用於過濾Record Value中的欄位擷取所需屬性列。
    // 預設值為空白，使用Record Value中的所有欄位作為資料表的屬性列。
    // 屬性名稱格式為tablestore.<tablename>.columns.whitelist.name和tablestore.<tablename>.columns.whitelist.type。
    // 其中<tablename>為資料表名稱的預留位置。
    // "tablestore.test.columns.whitelist.name":"A,B",
    // "tablestore.test.columns.whitelist.type":"string,integer",
    // 以下為寫入Tablestore相關的配置。
    // 指定寫入模式，可選值包括put和update，預設值為put。
    // put表示覆蓋寫。
    // update表示更新寫。
    "insert.mode":"put",
    // 是否需要保序，預設值為true。如果關閉保序模式，則有助於提升寫入效率。
    "insert.order.enable":"true",
    // 是否自動建立目標表，預設值為false。
    "auto.create":"false",
    // 指定刪除模式，可選值包括none、row、column和row_and_column，預設值為none。
    // none表示不允許進行任何刪除。
    // row表示允許刪除行。
    // column表示允許刪除屬性列。
    // row_and_column表示允許刪除行和屬性列。
```

```
    "delete.mode":"none",
    // 寫入資料表時記憶體中緩衝隊列的大小，預設值為1024，單位為行數。此配置項的值必須為2的指數。
    "buffer.size":"1024",
    // 寫入資料表時的回調線程數，預設值為核心數+1。
    // "max.thread.count":
    // 寫入資料表時的最大請求並發數，預設值為10。
    "max.concurrency":"10",
    // 寫入資料表時的分桶數，預設值為3。適當調大此配置項的值可提升並發寫入能力，但不應大於最大請求並發數。
    "bucket.count":"3",
    // 寫入資料表時對緩衝區的重新整理時間間隔，預設值為10000，單位為毫秒。
    "flush.Interval":"10000",
    // 以下為髒資料處理相關配置。
    // 在解析Kafka Record或者寫入資料表時可能發生錯誤，您可以通過以下配置進行處理。
    // 指定容錯能力，可選值包括none和all，預設值為none。
    // none表示任何錯誤都將導致Sink Task立即失敗。
    // all表示跳過產生錯誤的Record，並記錄該Record。
    "runtime.error.tolerance":"none",
    // 指定髒資料記錄模式，可選值包括ignore、kafka和tablestore，預設值為ignore。
    // ignore表示忽略所有錯誤。
    // kafka表示將產生錯誤的Record和錯誤資訊儲存在Kafka的另一個Topic中。
    // tablestore表示將產生錯誤的Record和錯誤資訊儲存在Tablestore另一張資料表中。
    "runtime.error.mode":"ignore"
    // 當髒資料記錄模式為kafka時，需要配置Kafka叢集地址和Topic。
    // "runtime.error.bootstrap.servers":"localhost:9092",
    // "runtime.error.topic.name":"errors",
    // 當髒資料記錄模式為tablestore時，需要配置Tablestore中資料表名稱。
    // "runtime.error.table.name":"errors",
  }
```

配置項說明

設定檔中的配置項說明請參見下表。其中時序相關配置項只有同步資料到時序表時才需要配置。

分類	配置項	類型	是否必選	樣本值	描述
	name	string	是	tablestore-sink	連接器（Connector）名稱。連接器名稱必須唯一。

分類	配置項	類型	是否必選	樣本值	描述
Kafka Connect常見配置	connector.class	class	是	TableStoreSinkConnector	連接器的Java類。 如果您要使用該連接器，請在connector.class配置項中指定Connector類的名稱，支援配置為Connector類的全名（com.aliyun.tablestore.kafka.connect.TableStoreSinkConnector）或別名（TableStoreSinkConnector）。 `connector.class=com.aliyun.tablestore.kafka.connect.TableStoreSinkConnector`
	tasks.max	integer	是	3	連接器支援建立的最大任務數。 如果連接器無法達到此並行度層級，則可能會建立較少的任務。
	key.converter	string	否	org.apache.kafka.connect.json.JsonConverter	覆蓋worker設定的預設key轉換器。
	value.converter	string	否	org.apache.kafka.connect.json.JsonConverter	覆蓋worker設定的預設value轉換器。
	topics	list	是	test	連接器輸入的Kafka Topic列表，多個Topic之間以英文逗號（,）分隔。 您必須為連接器設定topics來控制連接器輸入的Topic。
	tablestore.endpoint	string	是	https://xxx.xxx.ots.aliyuncs.com	Tablestore執行個體的服務地址。更多資訊，請參見服務地址。
	tablestore.mode	string	是	timeseries	根據資料同步到的表類型選擇模式。取值範圍如下： • normal（預設）：同步資料到Tablestore的資料表。 • timeseries：同步資料到Tablestore的時序表。

分類	配置項	類型	是否必選	樣本值	描述
連接器Connection配置	tablestore.access.key.id	string	是	LTAn**********	登入帳號的AccessKey ID和AccessKey Secret，擷取方式請參見擷取AccessKey。
	tablestore.access.key.secret	string	是	zbnK**********	
	tablestore.auth.mode	string	是	aksk	設定認證方式。取值範圍如下： • aksk（預設）：使用阿里雲帳號或者RAM使用者的AccessKey ID和Access Secret進行認證。請使用此認證方式。 • sts：使用STS臨時訪問憑證進行認證。對接雲kafka時使用。
	tablestore.instance.name	string	是	myotstest	Tablestore執行個體的名稱。

分類	配置項	類型	是否必選	樣本值	描述
	event.parse.class	class	是	DefaultEventParser	訊息解析器的Java類，預設值為DefaultEventParser。解析器用於從Kafka Record中解析出資料表的主鍵列和屬性列。 **注意** Tablestore對列值大小有限制。string類型和binary類型的主鍵列列值限制均為1 KB，屬性列列值限制均為2 MB。更多資訊，請參見通用限制。 如果資料類型轉換後列值超出對應限制，則將該Kafka Record作為髒資料處理。 如果使用預設的DefaultEventParser解析器，Kafka Record的Key或Value必須為Kafka Connect的Struct或Map類型。Struct中選擇的欄位必須為支援的資料類型，欄位會根據資料類型映射錶轉換為Tablestore資料類型寫入資料表。Map中的實值型別也必須為支援的資料類型，支援的資料類型同Struct，最終會被轉換為binary類型寫入資料表。Kafka和Tablestore的資料類型映射關係請參見附錄：Kafka和Tablestore資料類型映射。 如果Kafka Record為不相容的資料格式，則您可以通過實現com.aliyun.tablestore.kafka.connect.parsers.EventParser定義的介面來自訂解析器。

分類	配置項	類型	是否必選	樣本值	描述
	table.name.format	string	否	kafka_<topic>	目標資料表名稱的格式字串，預設值為<topic>。字串中可包含<topic>作為原始Topic的預留位置。例如當設定table.name.format為kafka_<topic>時，如果kafka中主題名稱為test，則映射到Tablestore的表名為kafka_test。 此配置項的優先順序低於topics.assign.tables配置項，如果配置了topics.assign.tables，則忽略table.name.format的配置。
	topics.assign.tables	list	是	test:destTable	指定topic與Tablestore表之間的映射關係，格式為 `<topic_1>:<tablename_1>,<topic_2>:<tablename_2>` 。多個映射關係之間以英文逗號（,）分隔，例如test:destTable表示將Topic名為test的訊息記錄寫入資料表destTable中。 此配置項的優先順序高於table.name.format配置項，如果配置了topics.assign.tables，則忽略table.name.format的配置。

分類	配置項	類型	是否必選	樣本值	描述
連接器Data Mapping配置	primarykey.mode	string	否	kafka	資料表的主鍵模式。取值範圍如下： • kafka：以<connect_topic>_<connect_partition>（Kafka主題和分區，用底線"_"分隔）和<connect_offset>（該訊息記錄在分區中的位移量）作為資料表的主鍵。 • record_key：以Record Key中的欄位（Struct類型）或者鍵（Map類型）作為資料表的主鍵。 • record_value：以Record Value中的欄位（Struct類型）或者鍵（Map類型）作為資料表的主鍵。 請配合tablestore.<tablename>.primarykey.name和tablestore.<tablename>.primarykey.type使用。此配置項不區分大小寫。

分類	配置項	類型	是否必選	樣本值	描述
	tablestore.<tablename>.primarykey.name	list	否	A,B	資料表的主鍵列名，其中<tablename>為資料表名稱的預留位置，包含1~4個主鍵列，以英文逗號（,）分隔。 主鍵模式不同時，主鍵列名的配置不同。 • 當設定主鍵模式為kafka時，以 `topic_partition,offset` 作為資料表主鍵列名稱。在該主鍵模式下，您可以不配置此主鍵列名。如果配置了主鍵列名，則不會覆蓋預設主鍵列名。 • 當設定主鍵模式為record_key時，從Record Key中提取與配置的主鍵列名相同的欄位（Struct類型）或者鍵（Map類型）作為資料表的主鍵。在該主鍵模式下主鍵列名必須配置。 • 當設定主鍵模式為record_value時，從Record Value中提取與配置的主鍵列名相同的欄位（Struct類型）或者鍵（Map類型）作為資料表的主鍵。在該主鍵模式下主鍵列名必須配置。 Tablestore資料表的主鍵列是有順序的，此屬性的配置應注意主鍵列名順序，例如PRIMARY KEY（A, B, C）與PRIMARY KEY（A, C, B）是不同的兩個主鍵結構。

分類	配置項	類型	是否必選	樣本值	描述
	tablestore.<tablename>.primarykey.type	list	否	string,integer	資料表的主鍵列資料類型，其中<tablename>為資料表名稱的預留位置，包含1~4個主鍵列，以英文逗號（,）分隔，順序必須與tablestore.<tablename>.primarykey.name相對應。此屬性配置不區分大小寫。資料類型的可選值包括integer、string、binary和auto_increment。 主鍵模式不同時，主鍵資料類型的配置不同。 • 當主鍵模式為kafka時，以 `string, integer` 作為資料表主鍵資料類型。 在該主鍵模式下，您可以不配置此主鍵列資料類型。如果配置了主鍵列資料類型，則不會覆蓋預設主鍵列資料類型。 • 當主鍵模式為record_key或者record_value時，指定相應主鍵列的資料類型。在該主鍵模式下主鍵列資料類型必須配置。 如果指定的資料類型與Kafka Schema中定義的資料類型發生衝突，則會產生解析錯誤，您可以配置Runtime Error相關屬性來應對解析錯誤。 當配置此配置項為auto_increment時，表示主鍵自增列，此時Kafka Record中可以預設該欄位，寫入資料表時會自動插入自增列。
	tablestore.<tablename>.columns.whitelist.name	list	否	A,B	資料表的屬性列白名單中屬性列名稱，其中<tablename>為資料表名稱的預留位置，以英文逗號（,）分隔。 如果配置為空白，則使用Record Value中的所有欄位（Struct類型）或者鍵（Map類型）作為資料表的屬性列，否則用於過濾得到所需屬性列。

分類	配置項	類型	是否必選	樣本值	描述
	tablestore.<tablename>.columns.whitelist.type	list	否	string, integer	資料表的屬性列白名單中屬性列資料類型，其中<tablename>為資料表名稱的預留位置，以英文逗號（,）分隔，順序必須與tablestore.<tablename>.columns.whitelist.name相對應。此屬性配置不區分大小寫。資料類型的可選值包括integer、string、binary、boolean和double。
	insert.mode	string	否	put	寫入模式。取值範圍如下： • put（預設）：對應Tablestore的PutRow操作，即新寫入一行資料會覆蓋原資料。 • update：對應Tablestore的UpdateRow操作，即更新一行資料，可以增加一行中的屬性列，或者更新已存在的屬性列的值。 此屬性配置不區分大小寫。
	insert.order.enable	boolean	否	true	寫入資料表時是否需要保序。取值範圍如下： • true（預設）：寫入時保持Kafka訊息記錄的順序。 • false：寫入順序無保證，但寫入效率會提升。
	auto.create	boolean	否	false	是否需要自動建立目標表，支援自動建立資料表或者時序表。取值範圍如下： • true：自動建立目標表。 • false（預設）：不自動建立目標表。

分類	配置項	類型	是否必選	樣本值	描述
連接器Write配置	delete.mode	string	否	none	刪除模式，僅當同步資料到資料表且主鍵模式為record_key時才有效。取值範圍如下： • none（預設）：不允許進行任何刪除。 • row：允許刪除行。當Record Value為空白時會刪除行。 • column：允許刪除屬性列。當Record Value中欄位值（Struct類型）或者索引值（Map類型）為空白時會刪除屬性列。 • row_and_column：允許刪除行和屬性列。 此屬性配置不區分大小寫。 刪除操作與insert.mode的配置相關。更多資訊，請參見附錄：刪除語義。
	buffer.size	integer	否	1024	寫入資料表時記憶體中緩衝隊列的大小，預設值為1024，單位為行數。此配置項的值必須是2的指數。
	max.thread.count	integer	否	3	寫入資料表時的回調線程數，預設值為 `CPU核心數+1` 。
	max.concurrency	integer	否	10	寫入資料表時的最大請求並發數。
	bucket.count	integer	否	3	寫入資料表時的分桶數，預設值為3。適當調大此配置項的值可提升並發寫入能力，但不應大於最大請求並發數。
	flush.Interval	integer	否	10000	寫入資料表時對緩衝區的重新整理時間間隔，預設值為10000，單位為毫秒。

分類	配置項	類型	是否必選	樣本值	描述
	runtime.error.tolerance	string	否	none	解析Kafka Record或者寫入表時產生錯誤的處理策略。取值範圍如下： • none（預設）：任何錯誤都將導致Sink Task立即失敗。 • all：跳過產生錯誤的Record，並記錄該Record。 此屬性配置不區分大小寫。

分類	配置項	類型	是否必選	樣本值	描述
連接器Runtime Error配置	runtime.error.mode	string	否	ignore	解析Kafka Record或者寫入表時產生錯誤，對錯誤的Record的處理策略。取值範圍如下： • ignore（預設）：忽略所有錯誤。 • kafka：將產生錯誤的Record和錯誤資訊儲存在Kafka的另一個Topic中，此時需要配置runtime.error.bootstrap.servers和runtime.error.topic.name。記錄運行錯誤的Kafka Record的Key和Value與原Record一致，Header中增加ErrorInfo欄位來記錄運行錯誤資訊。 • tablestore：將產生錯誤的Record和錯誤資訊儲存在Tablestore另一張資料表中，此時需要配置runtime.error.table.name。記錄運行錯誤的資料表主鍵列為 `topic_partition（string類型）`，`offset（integer類型）`，並且屬性列為key（bytes類型）、value（bytes類型）和error_info（string類型）。 kafka模式下需要對Kafka Record的Header、Key和Value進行序列化轉換，tablestore模式下需要對Kafka Record的Key和Value進行序列化轉換，此處預設使用org.apache.kafka.connect.json.JsonConverter，並且配置schemas.enable為true，您可以通過JsonConverter還原序列化得到未經處理資料。關於Converter的更多資訊，請參見Kafka Converter。
	runtime.error.bootstrap.servers	string	否	localhost:9092	用於記錄運行錯誤的Kafka叢集地址。

分類	配置項	類型	是否必選	樣本值	描述
	runtime.error.topic.name	string	否	errors	用於記錄運行錯誤的Kafka Topic名稱。
	runtime.error.table.name	string	否	errors	用於記錄運行錯誤的Tablestore表名稱。
	tablestore.timeseries.<tablename>.measurement	string	是	mName	將JSON中的key值為指定值對應的value值作為_m_name欄位寫入對應時序表中。 如果設定此配置項為<topic>，則將kafka記錄的topic作為_m_name欄位寫入時序表中。 配置項名稱中<tablename>為時序表名稱的預留位置，請根據實際修改，例如時序表名稱為test，則配置項名稱為tablestore.timeseries.test.measurement。
	tablestore.timeseries.<tablename>.dataSource	string	是	ds	將JSON中的key值為ds對應的value值作為_data_source欄位寫入對應時序表中。 配置項名稱中<tablename>為時序表名稱的預留位置，請根據實際修改。
	tablestore.timeseries.<tablename>.tags	list	是	region,level	將JSON中key值為region和level所對應的value值作為tags欄位寫入對應時序表中。 配置項名稱中<tablename>為時序表名稱的預留位置，請根據實際修改。
	tablestore.timeseries.<tablename>.time	string	是	timestamp	將JSON中key值為timestamp對應的value值作為_time欄位寫入對應時序表中。 配置項名稱中<tablename>為時序表名稱的預留位置，請根據實際修改。

分類	配置項	類型	是否必選	樣本值	描述
時序相關配置項	tablestore.timeseries.<tablename>.time.unit	string	是	MILLISECONDS	tablestore.timeseries.<tablename>.time值的時間戳記單位。取值範圍為SECONDS、MILLISECONDS、MICROSECONDS、NANOSECONDS。 配置項名稱中<tablename>為時序表名稱的預留位置，請根據實際修改。
	tablestore.timeseries.<tablename>.field.name	list	否	cpu,io	將JSON中key值為cpu和io的索引值對作為_field_name以及_field_name的值寫入對應時序表。 配置項名稱中<tablename>為時序表名稱的預留位置，請根據實際修改。
	tablestore.timeseries.<tablename>.field.type	string	否	double,integer	tablestore.timeseries.<tablename>.field.name中欄位對應的資料類型。取值範圍為double、integer、string、binary、boolean。多個資料類型之間用半形冒號（,）分隔。 配置項名稱中<tablename>為時序表名稱的預留位置，請根據實際修改。
	tablestore.timeseries.mapAll	boolean	否	false	將輸入JSON中的非主鍵欄位和時間欄位都作為field儲存到時序表中。 當配置項取值為false時，tablestore.timeseries.<tablename>.field.name和tablestore.timeseries.<tablename>.field.type必填。
	tablestore.timeseries.toLowerCase	boolean	否	true	將field中的key（輸入資料中非主鍵或者時間的key，或者配置在tablestore.timeseries.<tablename>.field.name中的key）轉為小寫寫入時序表。
	tablestore.timeseries.rowsPerBatch	integer	否	50	寫入tablestore時，一次請求支援寫入的最大行數。最大值為200，預設值為200。

附錄：Kafka和Tablestore資料類型映射

Kafka和Tablestore資料類型映射關係請參見下表。

Kafka Schema Type	Tablestore資料類型
STRING	STRING
INT8、INT16、INT32、INT64	INTEGER
FLOAT32、FLOAT64	DOUBLE
BOOLEAN	BOOLEAN
BYTES	BINARY

附錄：刪除語義

> **說明** 只有同步資料到資料表時才支援此功能。

當同步資料到資料表且Kafka訊息記錄的value中存在空值時，根據寫入模式（insert.mode）和刪除模式（delete.mode）的不同設定，資料寫入到Tablestore資料表中的處理方式不同，詳細說明請參見下表。

insert.mode	put				update			
delete.mode	none	row	column	row_and_column	none	row	column	row_and_column
value為空白值	覆蓋寫	刪行	覆蓋寫	刪行	髒資料	刪行	髒資料	刪行
value所有欄位值均為空白值	覆蓋寫	覆蓋寫	覆蓋寫	覆蓋寫	髒資料	髒資料	刪列	刪列
value部分欄位值為空白值	覆蓋寫	覆蓋寫	覆蓋寫	覆蓋寫	忽略空值	忽略空值	刪列	刪列

1.1.5. 錯誤處理

在將Kafka資料匯入到Tablestore的過程中可能產生錯誤，如果您不希望導致Sink Task立即失敗，您可以配置錯誤處理策略。

可能產生的錯誤類型如下：

- Kafka Connect Error

  此類錯誤發生在Sink Task執行資料匯入前，例如使用Converter進行還原序列化或者使用Kafka Transformations對訊息記錄進行輕量級修改時產生錯誤，您可以配置由Kafka提供的錯誤處理選項。

如果您想要跳過此類錯誤，請在connector設定檔中配置屬性 `errors.tolerance=all` 。更多資訊，請參見Kafka Connect Configs。

- Tablestore Sink Task Error

  此類錯誤發生在Sink Task執行資料匯入時，例如解析訊息記錄或者寫入Tablestore時產生錯誤，您可以配置由Tablestore Sink Connector提供的錯誤處理選項。

  如果您想要跳過此類錯誤，請在connector設定檔中配置屬性 `errors.tolerance=all` 。更多資訊，請參見配置說明。同時，您還可以選擇錯誤報表的方式，如果需要將產生錯誤的訊息記錄儲存到Tablestore中獨立的一張資料表中，請進行如下配置：

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

  在distributed模式下，您也可以通過REST API來管理connector和task。如果發生錯誤導致connector或者task停止，您可以選擇手動重啟connector或者task。

  i. 檢查connector和task狀態。

  - 查看connector狀態。

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

  - 查看task狀態。

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

  其中 `http://localhost:8083/connectors` 為Kafka REST服務的地址，name必須與設定檔中的name（連接器名稱）相同，taskid包含在connector狀態資訊中。

  您還可以通過執行以下命令擷取taskid。

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

  ii. 手動重啟connector或者task。

  - 重啟connector。

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

  - 重啟task。

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

1.2. 將Tablestore資料表中資料同步到另一個資料表中

使用通道服務或者DataWorks/DataX，實現將Tablestore資料表中資料同步到另一個資料表中。

前提條件

已建立目標資料表。具體操作，請參見建立資料表。

> **注意** 目標資料表的列必須與來源資料表中待遷移的列一一對應。

使用通道服務遷移同步

建立來源資料表的通道後，使用SDK進行遷移同步。遷移過程中可以自訂業務處理邏輯對資料進行處理。

1. 使用Tablestore控制台或SDK建立來源資料表的通道並記錄通道ID，具體操作請分別參見快速入門或使用SDK。
2. 使用SDK遷移資料。

   範例程式碼如下：

```
public class TunnelTest {
    public static void main(String[] args){
       TunnelClient tunnelClient = new TunnelClient("endpoint",
               "accessKeyId","accessKeySecret","instanceName");
        TunnelWorkerConfig config = new TunnelWorkerConfig(new SimpleProcessor());
        //tunnelId可以在控制台通道管理頁面查看，或調用describeTunnelRequest查詢。
        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{
       //目標tablestore連線物件。
       TunnelClient tunnelClient = new TunnelClient("endpoint",
               "accessKeyId","accessKeySecret","instanceName");
       @Override
        public void process(ProcessRecordsInput processRecordsInput) {
            //ProcessRecordsInput中返回了增量或全量資料。
            List<StreamRecord> list = processRecordsInput.getRecords();
            for(StreamRecord streamRecord : list){
                switch (streamRecord.getRecordType()){
                    case PUT:
                        //自訂業務處理邏輯。
                        //putRow
                        break;
                    case UPDATE:
                        //updateRow
                        break;
                    case DELETE:
                        //deleteRow
                        break;
                }
                System.out.println(streamRecord.toString());
            }
        }
        @Override
        public void shutdown() {
        }
    }
}
```

使用Dataworks/DataX遷移同步

通過DataWorks/DataX實現Tablestore資料的遷移同步，此處以DataWorks為例介紹遷移操作。

1. 新增Tablestore資料來源。

   分別以來源資料表和目標資料表所在執行個體新增Tablestore資料來源。具體操作，請參見步驟三：建立同步任務。

2. 建立同步任務節點。

   i. 以專案系統管理員身份登入DataWorks控制台。

   > 說明 僅專案系統管理員角色可以新增資料來源，其他角色的成員僅可查看資料來源。

   ii. 選擇地區，在左側導覽列，單擊**工作空間列表**。

   iii. 在**工作空間列表**頁面，單擊工作空間操作中的**進入資料開發**。

   iv. 在DataStudio控制台的**資料開發**頁面，單擊**商務程序**節點下的目標商務程序。

   如果需要建立商務程序，請參見步驟二：建立商務程序。

   v. 在**Data Integration**節點上右鍵選擇**建立 > 離線同步**。

   vi. 在**建立節點**對話方塊，輸入**節點名稱**。

3. 配置資料來源。

   i. 在**Data Integration**節點下，雙擊同步任務節點。

   ii. 在同步任務節點的編輯頁面的**選擇資料來源**地區，配置資料來源和資料去向。

   選擇**資料來源**和**資料去向**的**資料來源**為OTS並分別設定為來源資料表和目標資料表對應的資料來源，然後單擊  表徵圖或者**點擊轉換為指令碼**，進行指令碼配置。

   > 說明 Tablestore僅支援指令碼模式，配置指令碼的具體操作請分別參見配置Tablestore（OTS）Reader和配置Tablestore（OTS）Writer。

   iii. 單擊  表徵圖，儲存資料來源配置。

4. 運行同步任務。

   i. 單擊  表徵圖。

   ii. 在**參數**對話方塊，選擇調度的資源群組。

   iii. 單擊**確定**，開始運行任務。

   運行結束後，在**作業記錄**頁簽中可以查看任務是否成功和匯出的資料行數。

5. （可選）定時執行同步任務。具體操作，請參見步驟四：設定周期和依賴。

2.資料匯出

2.1. 將Tablestore資料同步到OSS

2.1.1. 概述

Tablestore中的增量資料及全量資料可以通過Data Integration的指令碼模式同步到OSS中。

Tablestore是構建在阿里雲飛天分布式系統之上的分布式NoSQL資料存放區服務，根據99.99%的高可用以及11個9的資料可靠性的標準設計。Tablestore通過資料分區和負載平衡技術，實現資料規模與訪問並發的無縫擴充，提供海量結構化資料的儲存和即時訪問。

OSS（Object Storage Service）是海量、安全、低成本、高可靠的雲端儲存體服務，提供99.99999999%的資料可靠性。使用RESTful API可以在互連網任何位置儲存和訪問，容量和處理能力彈性擴充，多種儲存類型供選擇全面最佳化儲存成本。

使用情境

Tablestore：提供專業的資料持久化儲存服務，以及面向使用者的即時高並發低延遲讀寫操作。

OSS：提供極低成本的備份功能。

使用方式

- 寫

  直接寫入Tablestore。

- 讀

  直接讀取Tablestore。

- 備份

  自動備份。

- 恢複

  使用Data Integration（OSSreader + OTSwriter）重新寫回Tablestore

限制

- 整行寫入

  使用Tablestore Stream功能，要求每次寫入Tablestore的資料必須是整行資料。目前類似物聯網資料的時序資料寫入方式都是整行寫入，後續基本無修改。

- 同步延時

  目前使用的是周期調度，每隔5分鐘調度一次，並且外掛程式有5分鐘延遲，同步總延遲為5~10分鐘。

開通服務

- 開通Tablestore服務。具體操作，請參見開通Tablestore服務。
- 開通OSS服務。

資料通道

離線

- 全量匯出到OSS
  - 指令碼模式
- 增量同步處理到OSS
  - 指令碼模式
- 全量匯入到Tablestore
  - 指令碼模式

2.1.2. 全量匯出（指令碼模式）

通過DataWorks控制台將Tablestore中的全量資料同步到OSS中。同步到OSS中的檔案可自由下載，也可作為Tablestore資料的備份存於OSS中。

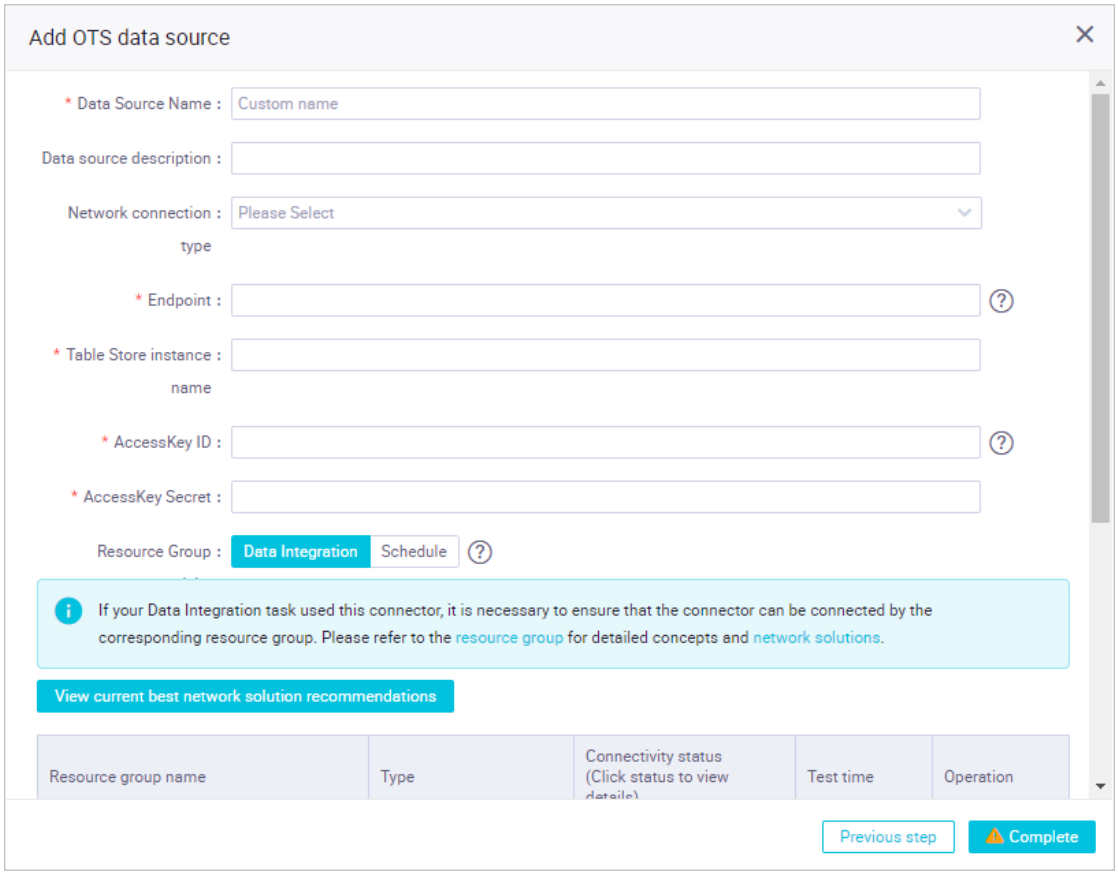
步驟一：新增Tablestore資料來源

將Tablestore添加為資料來源，具體操作步驟如下：

1. 進入Data Integration。
   i. 以專案系統管理員身份登入DataWorks控制台。

   > **說明** 僅專案系統管理員角色可以新增資料來源，其他角色的成員僅可查看資料來源。

   ii. 選擇地區，在左側導覽列，單擊**工作空間列表**。
   iii. 在**工作空間列表**頁面，單擊工作空間操作地區的**進入Data Integration**。
2. 新增資料來源。
   i. 在Data Integration控制台，單擊**資料來源管理**。
   ii. 在**資料來源管理**頁面，單擊**新增資料來源**。
   iii. 在**新增資料來源**對話方塊的**NoSQL**地區，選擇資料來源類型為**OTS**。
   iv. 在**新增OTS資料來源**對話方塊，配置參數。



參數	說明
資料來源名稱	資料來源的名稱。
資料來源描述	資料來源的描述資訊。
Endpoint	填寫目標Tablestore執行個體的服務地址。 ■ 如果Tablestore執行個體和OSS在同一個地區，填寫經典網地址。 ■ 如果Tablestore執行個體和OSS不在同一個地區，填寫公網地址。 ■ 不能填寫VPC地址。
Tablestore執行個體名稱	Tablestore執行個體的名稱。
AccessKey ID	登入賬戶的AccessKey ID和AccessKey Secret，擷取方式請參見為RAM使用者建立存取金鑰。
AccessKey Secret	

v. 單擊**測試連通性**，測試資料來源的連通狀態。

3. 單擊**完成**。

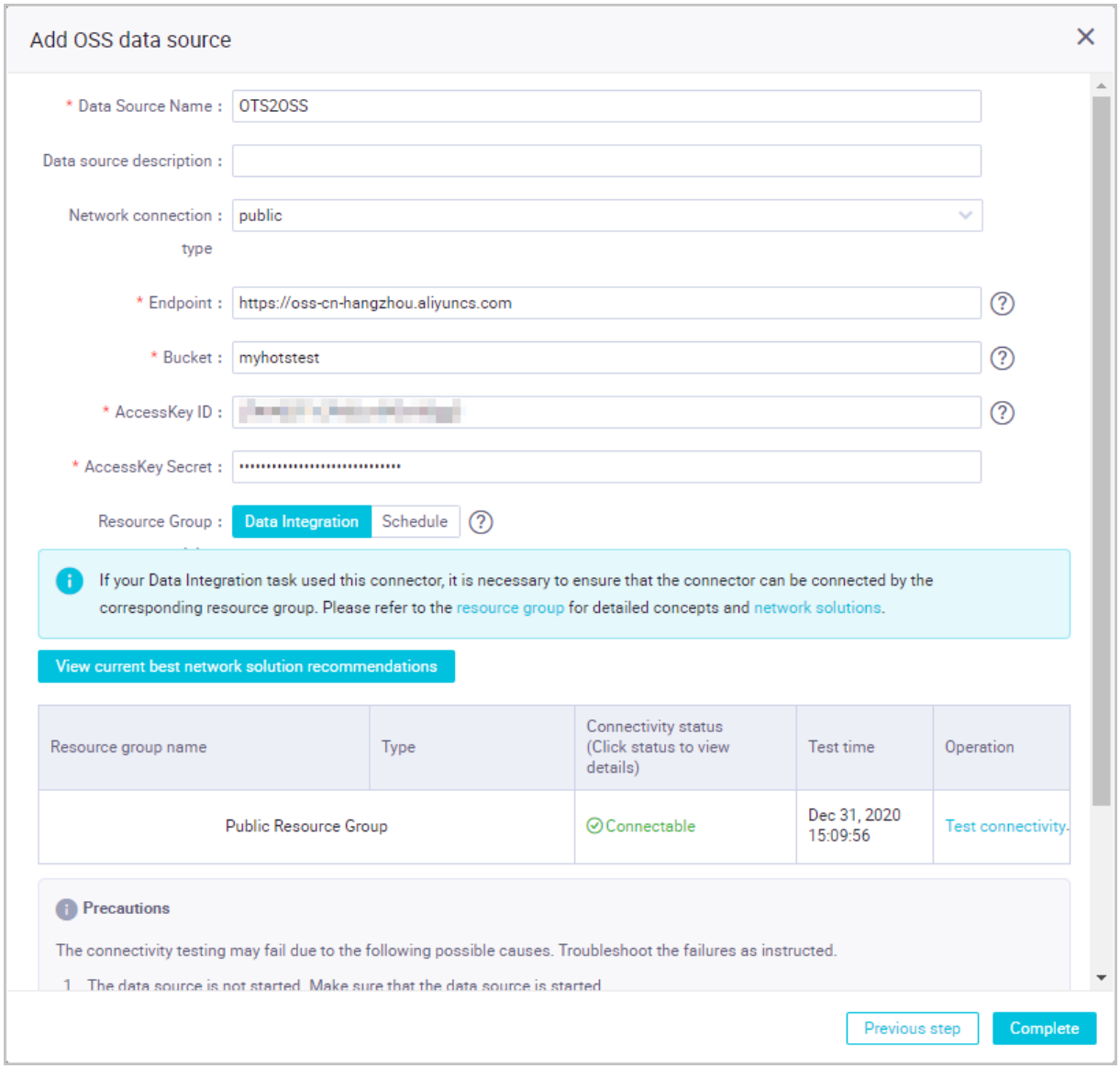
   在**資料來源管理**頁面，會顯示該資料來源資訊。

步驟二：新增OSS資料來源

操作與步驟一類似，只需在Semi-structuredstorage地區，選擇資料來源類型為OSS。

> **說明** 配置OSS資料來源的參數時，請注意Endpoint中不包括Bucket的名稱。

本樣本中，該資料來源名稱使用OTS2OSS，如下圖所示。



步驟三：建立同步任務

建立並配置Tablestore到OSS的同步任務，具體操作步驟如下：

1. 進入資料開發。
   i. 以專案系統管理員身份登入DataWorks控制台。
   ii. 選擇地區，在左側導覽列，單擊**工作空間列表**。
   iii. 在**工作空間列表**頁面，單擊工作空間操作地區的**進入資料開發**。
2. 在DataStudio控制台的**資料開發**頁面，單擊**商務程序**節點下的目標商務程序。

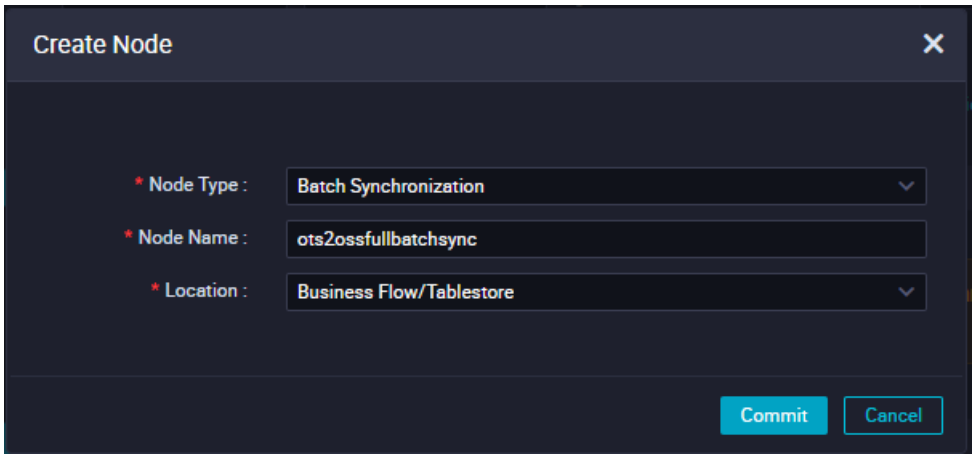
   如果需要建立商務程序，請參見步驟二：建立商務程序。
3. 建立同步任務節點。

每個同步任務都需建立一個相應的節點。

i. 在Data Integration節點上右鍵選擇**建立 > 離線同步**。

您也可以將滑鼠移至上方在  表徵圖，選擇**Data Integration > 離線同步**來建立節點。

ii. 在**建立節點**對話方塊，輸入節點名稱，選擇一個目標檔案夾。



iii. 單擊**提交**。

4. 配置資料來源。

i. 在Data Integration節點下，雙擊同步任務節點。

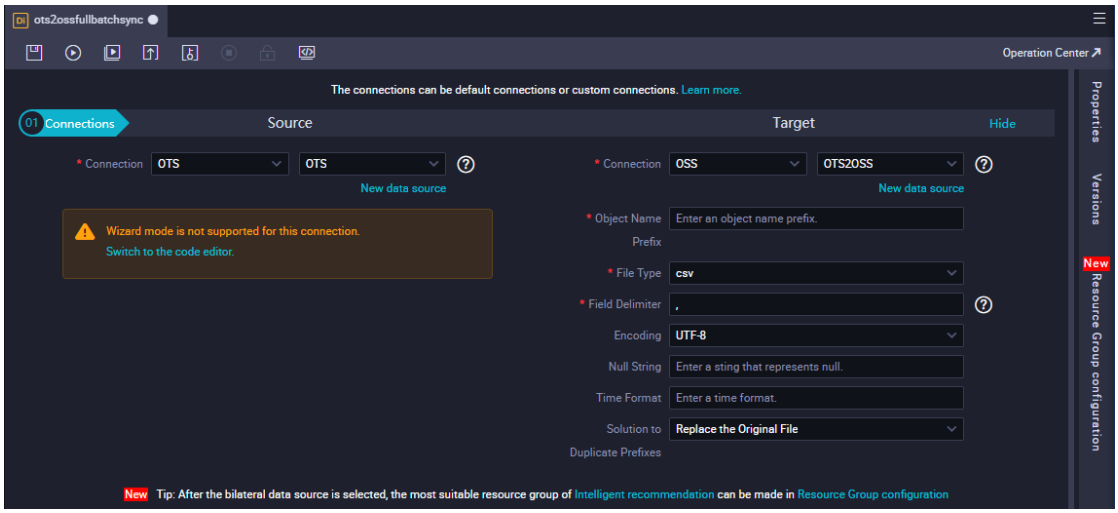
ii. 在同步任務節點的編輯頁面的**選擇資料來源**地區，配置資料來源和資料去向。

- 配置資料來源。

  選擇**資料來源**的**資料來源**為OTS。

- 配置資料去向。

  選擇**資料去向**的**資料來源**為OSS，並配置資料來源。



iii. 單擊  表徵圖或者**點擊轉換為指令碼**，進行指令碼配置。

Tablestore僅支援指令碼模式配置，使用過程中涉及Tablestore（OTS） Reader和OSS Writer外掛程式的配置。具體操作，請參見配置Tablestore（OTS） Reader和配置OSS Writer。

在指令碼配置頁面，請根據如下樣本完成配置。

```
{
"type": "job",    # 不能修改。
"version": "1.0",  # 不能修改。
"configuration": {
 "setting": {
   "errorLimit": {
     "record": "0"  # 當錯誤個數超過record個數時，匯入任務會失敗。
   },
   "speed": {
     "mbps": "1",  # 匯入速率，單位是MB/s。
     "concurrent": "1"  # 並發度。
   }
 },
 "reader": {
   "plugin": "ots",  # 不能修改。
   "parameter": {
     "datasource": "",   # Data Integration中的資料來源名稱，需要提前配置完成，此處可選擇配置Tablestore的資料來源或者填寫明文的AccessKeyID等鑒權資訊，建議使用資料來源。
     "table": "",    # Tablestore中的資料表名稱。
     "column": [   # 需要匯出到OSS的列名，不能設定為空白。
       {
         "name": "column1"    # Tablestore中列名，此列需要匯入到OSS。
       },
       {
         "name": "column2"    # Tablestore中列名，此列需要匯入到OSS。
       }
     ],
     "range": {
       "begin": [
         {
           "type": "INF_MIN"    # Tablestore中第一列主鍵的起始位置。如果需要匯出全量，此處請配置為INF_MIN；如果只需匯出部分，則按需要配置。當資料表存在多個主鍵列時，此處begin中需要配置對應主鍵列資訊。
         }
       ],
       "end": [
         {
           "type": "INF_MAX"    # Tablestore中第一列主鍵的結束位置。如果需要匯出全量，此處請配置為INF_MAX；如果只需匯出部分，則按需要配置。當資料表存在多個主鍵列時，此處end中需要配置對應主鍵列資訊。
         }
       ],
       "split": [  # 用於配置Tablestore的資料表的分區資訊，可以加速匯出，下一個版本會自動處理。
       ]
     }
   }
 },
 "writer": {
   "plugin": "oss",
   "parameter": {
     "datasource": "",  # 配置OSS的資料來源。
```

```
        "object": "",  # Object的目錄，無需包括Bucket名稱，例如tablestore/20171111/。如果是定時匯出，則此處需要使用變數，例如tablestore/${date}，然後在配置調度參數時配置${date}的值。
        "writeMode": "truncate", # 當同名檔案存在時系統進行的操作，全量匯出時請使用truncate，可選值包括truncate、append和nonConflict，truncate表示會清理已存在的同名檔案，append表示會加到已存在的同名檔案內容後面，nonConflict表示當同名檔案存在時會報錯。
        "fileFormat": "csv", # 檔案類型，可選值包括csv、txt和parquet格式。
        "encoding": "UTF-8",  # 編碼類別型。
        "nullFormat": "null", # 定義null值的字串標識符方式，可以是Null 字元串。
        "dateFormat": "yyyy-MM-dd HH:mm:ss",  # 時間格式。
        "fieldDelimiter": "," # 每一列的分隔字元。
    }
  }
}
}
```

iv. 單擊表徵圖，儲存資料來源配置。

> **說明**
>
> - 由於全量匯出一般是一次性的，所以無需配置自動調度參數。如果需要配置調度參數，請參見增量同步處理中的調度參數配置。
> - 如果在指令碼配置中存在變數，例如存在*${date}*，則需要在運行同步任務時設定變數的具體值。

5. 運行同步任務。

   i. 單擊表徵圖。

   ii. 在**參數**對話方塊，選擇調度的資源群組。

   iii. 單擊**確定**，開始運行任務。

   運行結束後，在**作業記錄**頁簽中可以查看任務是否成功和匯出的資料行數。

步驟四：查看匯出到OSS中的資料

1. 登入OSS管理主控台。
2. 選擇相應Bucket和檔案名稱，下載後可查看內容是否符合預期。

2.1.3. 增量同步處理（指令碼模式）

通過DataWorks控制台將Tablestore中的增量資料同步到OSS中。

步驟一：新增Tablestore資料來源

如果已新增Tablestore資料來源，請跳過此步驟。

新增Tablestore資料來源的操作請參見步驟一：新增Tablestore資料來源。

步驟二：新增OSS資料來源

如果已新增OSS資料來源，請跳過此步驟。

新增OSS資料來源的操作請參見步驟二：新增OSS資料來源。

步驟三：配置定時同步任務

建立並配置Tablestore到OSS的增量資料同步任務，具體操作步驟如下：

1. 進入資料開發。
   i. 以專案系統管理員身份登入DataWorks控制台。

   > **說明** 僅專案系統管理員角色可以新增資料來源，其他角色的成員僅可查看資料來源。

   ii. 選擇地區，在左側導覽列，單擊**工作空間列表**。
   iii. 在**工作空間列表**頁面，單擊工作空間操作地區的**進入資料開發**。
2. 在DataStudio控制台的**資料開發**頁面，單擊**商務程序**節點下的目標商務程序。

   如果需要建立商務程序，請參見步驟二：建立商務程序。
3. 建立同步任務節點。

   每個同步任務都需建立一個相應的節點。
   i. 在**Data Integration**節點上右鍵選擇**建立 > 離線同步**。

   您也可以將滑鼠移至上方在  表徵圖，選擇**Data Integration > 離線同步**來建立節點。

   ii. 在**建立節點**對話方塊，輸入節點名稱，選擇一個目標檔案夾。

   iii. 單擊**提交**。
4. 配置資料來源。
   i. 在**Data Integration**節點下，雙擊同步任務節點。
   ii. 在同步任務節點的編輯頁面的**選擇資料來源**地區，配置資料來源和資料去向。
   - 配置資料來源。

     選擇**資料來源**的**資料來源**為**OTS Stream**，選擇資料來源和資料表，可根據需要配置任務開始時間、結束時間、狀態表的名稱、最大重試次數等。
   - 配置資料去向。

     選擇**資料去向**的**資料來源**為**OSS**，選擇資料來源，配置Object首碼、文本類型、列的分隔字元等。

   iii. 單擊  表徵圖，進行指令碼配置。

   使用過程時涉及OTSStream Reader和OSS Writer外掛程式的配置。具體操作，請參見配置OTSStream Reader和配置OSS Writer。

   在指令碼配置頁面，請根據如下樣本完成配置。

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

```
"setting": {
"errorLimit": {
 "record": "0"  # 允許出錯的個數，當錯誤超過這個數目的時候同步任務會失敗。
},
"speed": {
 "mbps": "1",  # 每次同步任務的最大流量。
 "concurrent": "1"   # 每次同步任務的並發度。
}
},
"reader": {
"plugin": "otsstream",  # Reader外掛程式的名稱。
"parameter": {
 "datasource": "", # Tablestore的資料來源名稱，如果有此項則無需配置endpoint，accessId，accessKey和instanceName。
 "dataTable": "", # Tablestore中的資料表名稱。
 "statusTable": "TablestoreStreamReaderStatusTable", # 儲存Tablestore Stream狀態的表，一般無需修改。
 "startTimestampMillis": "",  # 開始匯出的時間點，由於是增量匯出，需要迴圈啟動此任務，則此處每次啟動時的時間都不同，因此需要設定一個變數，例如${start_time}。
 "endTimestampMillis": "",  # 結束匯出的時間點。此處也需要設定一個變數，例如${end_time}。
 "date": "yyyyMMdd",  # 匯出該日期的資料，此功能與startTimestampMillis和endTimestampMillis重複，需要刪除。
 "mode": "single_version_and_update_only", # Tablestore Stream匯出資料的格式，目前需要設定為single_version_and_update_only。如果配置模板中無此項，則需要增加。
 "column":[  # 設定資料表中需要匯出到OSS中的列，如果配置模板中無此項則需要增加，具體列個數由使用者自訂設定。
           {
              "name": "uid"  # 列名，此處是Tablestore中的主鍵。
           },
           {
              "name": "name"  # 列名，此處是Tablestore中的屬性列。
           },
 ],
 "isExportSequenceInfo": false, # single_version_and_update_only模式下只能設定為false。
 "maxRetries": 30 # 最大重試次數。
}
},
"writer": {
"plugin": "oss", # Writer外掛程式的名稱。
"parameter": {
 "datasource": "", # OSS的資料來源名稱。
 "object": "",  # 備份到OSS的檔案名稱首碼，建議使用"Tablestore執行個體名/表名/date"，例如"instance/table/{date}"。
 "writeMode": "truncate", # 當同名檔案存在時系統進行的操作，可選值包括truncate、append和nonConflict，truncate表示會清理已存在的同名檔案，append表示會加到已存在的同名檔案內容後面，nonConflict表示當同名檔案存在時會報錯。
 "fileFormat": "csv", # 檔案類型，可選值包括csv、txt和parquet格式。
 "encoding": "UTF-8", # 編碼類別型。
 "nullFormat": "null", # 定義null值的字串標識符方式，可以是Null 字元串。
 "dateFormat": "yyyy-MM-dd HH:mm:ss", # # 時間格式。
 "fieldDelimiter": "," # 每一列的分隔字元。
}
}
```

```
}
}
```

iv. 單擊表徵圖，儲存資料來源配置。

5. 運行同步任務。

   i. 單擊表徵圖。

   ii. 在**參數**對話方塊，選擇調度的資源群組。

   iii. 單擊**確定**，開始運行任務。

   運行結束後，在**作業記錄**頁簽中可以查看任務是否成功和匯出的資料行數。

   Tablestore的增量資料可以在延遲5~10分鐘的基礎上自動同步到OSS中。

6. 配置調度參數。

   通過**調度配置**，可以配置同步任務的執行時間、重跑屬性、調度依賴等。

   i. 在Data Integration節點下，雙擊同步任務節點。

   ii. 在同步任務節點的編輯頁面的右側單擊**調度配置**，進行調度參數配置，詳情請參見步驟四：設定周期和依賴。

7. 提交同步任務。

   將同步任務提交到調度系統後，調度系統會根據配置的調度參數，自動定時執行同步任務。

   i. 在同步任務節點的編輯頁面，單擊表徵圖。

   ii. 在**提交新版本**對話方塊，輸入備忘資訊。

   iii. 單擊**確認**。

步驟四：查看同步任務

1. 進入營運中心。

   > **說明** 您也可以在DataStudio控制台的右上方單擊**營運中心**，快速進入營運中心。

   i. 以專案系統管理員身份登入DataWorks控制台。

   ii. 選擇地區，在左側導覽列，單擊**工作空間列表**。

   iii. 在**工作空間列表**頁面，單擊工作空間操作地區的**進入營運中心**。

2. 在營運中心控制台，選擇**周期任務營運 > 周期任務**。

3. 在**周期任務**頁面，查看提交的同步任務詳情。

   - 在左側導覽列中，選擇**周期任務營運 > 周期執行個體**，可以查看當天需要啟動並執行周期任務。單擊執行個體名稱，可以查看任務運行詳情。
   - 當單個任務在運行中或運行結束後，可以查看日誌。

步驟五：查看匯出到OSS中的資料

1. 登入OSS管理主控台。
2. 選擇相應Bucket和檔案名稱，下載後可查看內容是否符合預期。

2.2. 將Tablestore資料同步到MaxCompute

2.2.1. 全量匯出（指令碼模式）

通過DataWorks控制台將Tablestore中的全量資料匯出到MaxCompute中。

步驟一：新增Tablestore資料來源

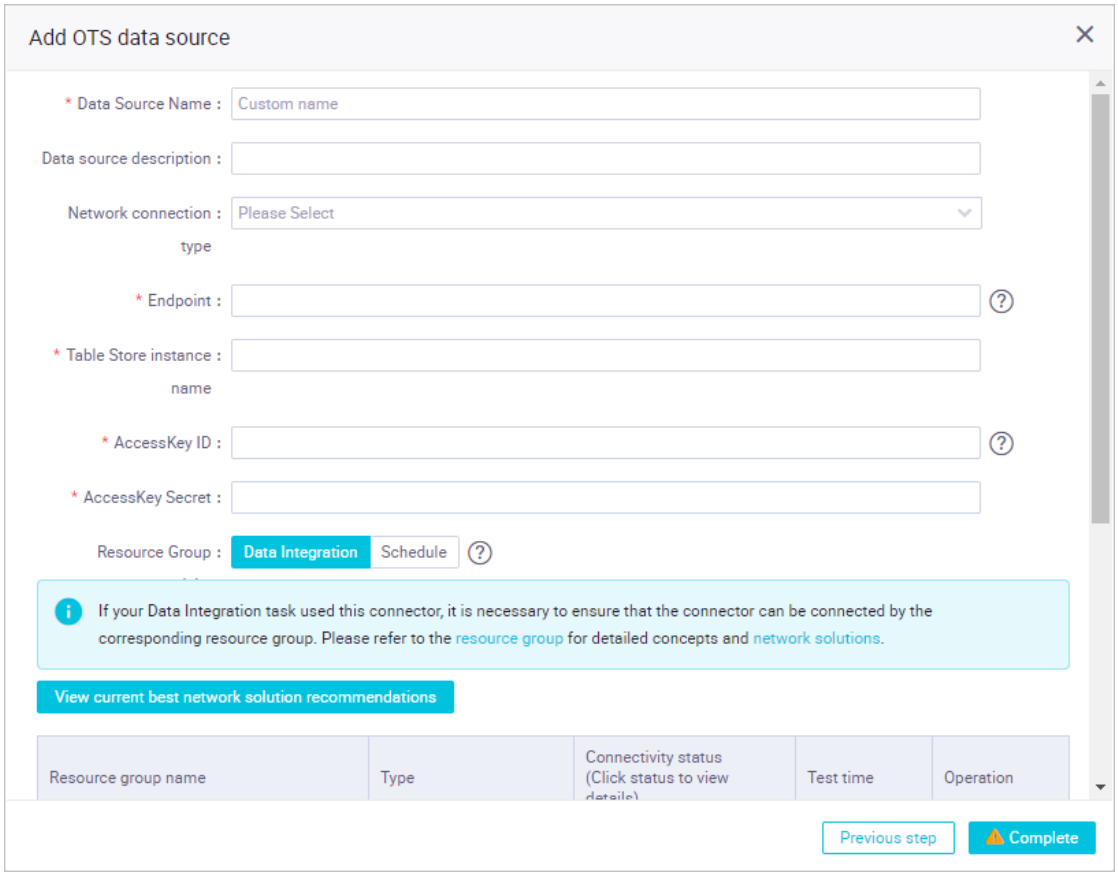
將Tablestore資料庫添加為資料來源，具體操作步驟如下：

1. 進入Data Integration。
   i. 以專案系統管理員身份登入DataWorks控制台。

      > **說明** 僅專案系統管理員角色可以新增資料來源，其他角色的成員僅可查看資料來源。

   ii. 選擇地區，在左側導覽列，單擊**工作空間列表**。
   iii. 在**工作空間列表**頁面，單擊工作空間操作地區的**進入Data Integration**。
2. 新增資料來源。
   i. 在Data Integration控制台，單擊**資料來源管理**。
   ii. 在**資料來源管理**頁面，單擊**新增資料來源**。
   iii. 在**新增資料來源**對話方塊的**NoSQL**地區，選擇資料來源類型為**OTS**。

iv. 在**新增OTS資料來源**對話方塊，配置參數。



參數	說明
資料來源名稱	資料來源的名稱，例如gps_data。
資料來源描述	資料來源的描述資訊。
Endpoint	填寫目標Tablestore執行個體的服務地址。 ■ 如果Tablestore執行個體和MaxCompute在同一個地區，填寫經典網地址。 ■ 如果Tablestore執行個體和MaxCompute不在同一個地區，填寫公網地址。 ■ 不能填寫VPC地址。
Tablestore執行個體名稱	Tablestore執行個體的名稱。
AccessKey ID	登入賬戶的AccessKey ID和AccessKey Secret，擷取方式請參見為RAM使用者建立存取金鑰。
AccessKey Secret	

v. 單擊**測試連通性**，測試資料來源的連通狀態。

3. 單擊**完成**。

在**資料來源管理**頁面，會顯示該資料來源資訊。

步驟二：新增MaxCompute資料來源

操作與步驟一類似，只需在Big Data Storage地區，選擇資料來源類型為MaxCompute(ODPS)。

本樣本中，該資料來源名稱使用OTS2ODPS，如下圖所示。

步驟三：配置同步任務

建立並配置Tablestore到MaxCompute的同步任務，具體操作步驟如下：

1. 進入資料開發。
   i. 以專案系統管理員身份登入DataWorks控制台。
   ii. 選擇地區，在左側導覽列，單擊**工作空間列表**。
   iii. 在**工作空間列表**頁面，單擊工作空間操作地區的**進入資料開發**。
2. 在DataStudio控制台的**資料開發**頁面，單擊**商務程序**節點下的目標商務程序。

   如果需要建立商務程序，請參見步驟二：建立商務程序。
3. 建立同步任務節點。

   每個同步任務都需建立一個相應的節點。
   i. 在**Data Integration**節點上右鍵選擇**建立 > 離線同步**。

      您也可以將滑鼠移至上方在  表徵圖，選擇**Data Integration > 離線同步**來建立節點。
   ii. 在**建立節點**對話方塊，輸入節點名稱，選擇一個目標檔案夾。
   iii. 單擊**提交**。
4. 配置資料來源。
   i. 在**Data Integration**節點下，雙擊同步任務節點。
   ii. 在同步任務節點的編輯頁面的**選擇資料來源**地區，配置資料來源和資料去向。
      - 配置資料來源。

        選擇**資料來源**的**資料來源**為OTS。
      - 配置資料去向。

        選擇**資料去向**的**資料來源**為ODPS，並選擇對應的**表**。
   iii. 單擊  表徵圖或者**點擊轉換為指令碼**，進行指令碼配置。

      Tablestore僅支援指令碼模式配置，使用過程中涉及Tablestore（OTS） Reader和MaxCompute Writer外掛程式的配置。具體操作，請參見配置Tablestore（OTS） Reader和。

      在指令碼配置頁面，請根據如下樣本完成配置。

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

```
    "record": "0"     # 能夠允許的最大錯誤數。
  },
  "speed": {
    "mbps": "1",    # 最大的流量，單位為MB。
    "concurrent": "1"  # 並發數。
  }
},
"reader": {
  "plugin": "ots",  # 讀取的外掛程式名稱。
  "parameter": {
    "datasource": "",  # 資料來源名稱。
    "table": "",  # 資料表名稱。
    "column": [  # 需要匯出到MaxCompute中去的Tablestore中的列名。
      {
        "name": "column1"
      },
      {
        "name": "column2"
      },
      {
        "name": "column3"
      },
      {
        "name": "column4"
      },
      {
        "name": "column5"
      }
    ],
    "range": {  # 需要匯出的資料範圍，如果是全量匯出，則需要從INF_MIN到INF_MAX。
      "begin": [ # 需要匯出資料的起始位置，最小的位置是INF_MIN。begin中的配置項數目個數和Tablestore中相應表的主鍵列個數一致。
        {
          "type": "INF_MIN"
        },
        {
          "type": "INF_MIN"
        },
        {
          "type": "STRING",  # 此配置項表示第三列的起始位置是begin1。
          "value": "begin1"
        },
        {
          "type": "INT",  # 此配置項表示第四列的起始位置是0。
          "value": "0"
        }
      ],
      "end": [  # 匯出資料的結束位置。
        {
          "type": "INF_MAX"
        },
        {
          "type": "INF_MAX"
        },
```

```
          {
            "type": "STRING",
            "value": "end1"
          },
          {
            "type": "INT",
            "value": "100"
          }
        ],
        "split": [  # 配置分區範圍，一般可以不配置，如果效能較差，可以提交工單或者加入DingTalk群23307953聯絡Tablestore技術支援人員處理。
          {
            "type": "STRING",
            "value": "splitPoint1"
          },
          {
            "type": "STRING",
            "value": "splitPoint2"
          },
          {
            "type": "STRING",
            "value": "splitPoint3"
          }
        ]
      }
    }
  },
  "writer": {
    "plugin": "odps",  # MaxCompute寫入的外掛程式名。
    "parameter": {
      "datasource": "",  # MaxCompute的資料來源名稱。
      "column": [],  # MaxCompute中的列名，列名順序需對應TableStore中的列名順序。
      "table": "",  # MaxCompute中的表名，需要提前建立好，否則任務執行會失敗。
      "partition": "",  # 如果表為分區表，則必填。如果表為非分區表，則不能填寫。需要寫入資料表的分區資訊，必須指定到最後一級分區。
      "truncate": false  # 是否清空之前的資料。
    }
  }
  }
  }
```

您可以通過begin和end來設定匯出的資料範圍，假設表包含pk1（String類型）和pk2（Integer類型）兩個主鍵列。

- 如果需要匯出全表資料，則配置樣本如下：

```
"begin": [ # 需要匯出資料的起始位置。
  {
    "type": "INF_MIN"
  },
  {
    "type": "INF_MIN"
  }
],
"end": [  # 需要匯出資料的結束位置。
  {
    "type": "INF_MAX"
  },
  {
    "type": "INF_MAX"
  }
],
```

- 如果需要匯出pk1="tablestore" 的行，則配置樣本如下：

```
"begin": [ # 匯出資料的起始位置。
  {
    "type": "STRING",
    "value": "tablestore"
  },
  {
    "type": "INF_MIN"
  }
],
"end": [  # 匯出資料的結束位置。
  {
    "type": "STRING",
    "value": "tablestore"
  },
  {
    "type": "INF_MAX"
  }
],
```

iv. 單擊  表徵圖，儲存資料來源配置。

5. 運行同步任務。

i. 單擊  表徵圖。

ii. 在**參數**對話方塊，選擇調度的資源群組。

iii. 單擊**確定**，開始運行任務。

運行結束後，在**作業記錄**頁簽中可以查看任務是否成功和匯出的資料行數。

6. 配置調度參數。

通過**調度配置**，可以配置同步任務的執行時間、重跑屬性、調度依賴等。

i. 在Data Integration節點下，雙擊同步任務節點。

ii. 在同步任務節點的編輯頁面的右側單擊**調度配置**，進行調度參數配置，詳情請參見步驟四：設定周期和依賴。

7. 提交同步任務。

i. 在同步任務節點的編輯頁面，單擊表徵圖。

ii. 在**提交新版本**對話方塊，輸入備忘資訊。

iii. 單擊**確認**。

將同步任務提交到調度系統後，調度系統會根據配置的調度參數，自動定時執行同步任務。

步驟四：查看同步任務

1. 進入營運中心。

> **說明** 您也可以在DataStudio控制台的右上方單擊**營運中心**，快速進入營運中心。

i. 以專案系統管理員身份登入DataWorks控制台。

ii. 選擇地區，在左側導覽列，單擊**工作空間列表**。

iii. 在**工作空間列表**頁面，單擊工作空間操作地區的**進入營運中心**。

2. 在營運中心控制台，選擇**周期任務營運 > 周期任務**。
3. 在**周期任務**頁面，查看提交的同步任務詳情。
   - 在左側導覽列中，選擇**周期任務營運 > 周期執行個體**，可以查看當天需要啟動並執行周期任務。單擊執行個體名稱，可以查看任務運行詳情。
   - 當單個任務在運行中或運行結束後，可以查看日誌。

步驟五：查看匯入到MaxCompute中的資料

1. 進入資料地圖。

i. 以專案系統管理員身份登入DataWorks控制台。

ii. 選擇地區，在左側導覽列，單擊**工作空間列表**。

iii. 在**工作空間列表**頁面，單擊工作空間操作地區的**進入資料地圖**。

2. 在資料地圖控制台的導覽列，選擇**我的資料 > 我管理的資料**。
3. 在**我管理的資料**頁簽，單擊匯入資料的表名稱。
4. 在表詳情頁面，單擊**資料預覽**頁簽，查看匯入的資料。