数据传输服务

数据订阅（Kafka 版）

腾讯云

文档目录

数据订阅（Kafka 版）
数据订阅支持的数据库

最近更新时间：2024-12-02 09:52:52

数据订阅是指获取数据库中关键业务的数据变化信息，将这些信息包装为消息对象推送到 Kafka 中，方便下游业务订阅、获取和消费。腾讯云 DTS 支持通过 Kafka/Flink 客户端直接消费数据，方便用户搭建云数据库和异构系统之间的数据同步，如缓存更新、ETL（数据仓库技术）实时同步以及业务异步解耦等。
用户对订阅数据进行消费时，支持 ProtoBuf、Avro 和 JSON 三种形式。ProtoBuf 和 Avro 是二进制格式，效率更高，JSON 为轻量级的文本格式，更加简单易用。
DTS 支持对如下数据库类型进行数据订阅。

源数据库类型	源数据库版本	支持订阅的数据类型	订阅数据格式	Flink（DataStream API）订阅数据格式
MySQL	• 自建 MySQL 5.5、5.6、5.7、8.0 • 腾讯云 MySQL 5.5、5.6、5.7、8.0	• 数据更新 • 结构更新 • 全实例	Protobuf/Avro/JSON	Avro
MariaDB	• 自建 MariaDB 5.5、10.0、10.1 • 腾讯云 MariaDB （内核版本 Percona 5.7、MySQL 8.0、MariaDB 10.1）	• 数据更新 • 结构更新 • 全实例	Protobuf	不支持
Percona	自建 Percona 5.5、5.6、5.7、8.0	• 数据更新 • 结构更新 • 全实例	Protobuf	不支持
TDSQL MySQL	腾讯云 TDSQL MySQL（内核版本 MySQL 8.0、Percona 5.7）	• 数据更新 • 结构更新 • 全实例	Protobuf	Protobuf
TDSQL-C MySQL	腾讯云 TDSQL-C MySQL 5.7、8.0	• 数据更新 • 结构更新 • 全实例	Protobuf/Avro/JSON	Avro
TDSQL PostgreSQL 版	腾讯云 TDSQL PostgreSQL 版	数据更新	Protobuf	不支持
MongoDB	云数据库 MongoDB 3.6、4.0、4.2、4.4、5.0	数据更新	JSON	不支持

MySQL 系列数据订阅
创建 MySQL 或 TDSQL-C MySQL 数据订阅

最近更新时间：2024-10-16 16:11:21

本场景介绍使用 DTS 创建 MySQL 或 TDSQL-C MySQL 的数据订阅任务操作指导。

前提条件

- 已准备好待订阅的源端数据库，并且数据库版本符合要求，请参见 数据订阅支持的数据库 。
- 已在源库中开启 binlog。
- 已在源库中创建订阅账号，需要账号权限如下：REPLICATION CLIENT、REPLICATION SLAVE、PROCESS 和全部对象的 SELECT 权限。
  具体授权语句如下：

```
create user '迁移账号' IDENTIFIED BY '账号密码';
grant SELECT, REPLICATION CLIENT,REPLICATION SLAVE,PROCESS on *.* to '迁移账号'@'%';
flush privileges;
```

约束限制

- 订阅的消息保存在 DTS 内置 Kafka（单 Topic）中，目前默认保存时间为最近1天，单 Topic 的最大存储为500G，当数据存储时间超过1天，或者数据量超过500G时，内置 Kafka 都会开始清除最先写入的数据。所以请用户及时消费，避免数据在消费完之前就被清除。
- 数据消费的地域需要与订阅任务所属的地域相同。
- 当前不支持 geometry 相关的数据类型。
- 订阅任务过程中，如果进行修改订阅对象等操作会发生任务重启，重启后可能会导致用户在 kafka 客户端消费数据时出现重复。
  - DTS 是按最小数据单元进行传输的，增量数据每标记一个 checkpoint 位点就是一个数据单元，如果重启时，刚好一个数据单元传输已完成，则不会导致数据重复；如果重启时，一个数据单元还正在传输中，那么再次启动后需要重新拉取这个数据单元，以保证数据完整性，这样就会导致数据重复。
  - 2023年之后新建的 MySQL 或 TDSQL-C MySQL 订阅任务，DTS 已支持数据去重，消费 Demo 中包含了去重逻辑，不会产生数据重复；2023年之前的任务，用户如果对重复数据比较关注，需要自行在消费数据时设置去重逻辑。

支持订阅的 SQL 操作

操作类型	支持的 SQL 操作
DML	INSERT、UPDATE、DELETE
DDL	CREATE DATABASE、DROP DATABASE、CREATE TABLE、ALTER TABLE、DROP TABLE、RENAME TABLE

操作步骤

1. 登录 DTS 控制台，在左侧导航选择**数据订阅**，单击**新建数据订阅**。
2. 在新建数据订阅页，选择相应配置，单击**立即购买**。

- 计费模式：支持包年包月和按量计费。
- 地域：源库为腾讯云数据库，这里的地域需与源数据库实例的地域保持一致。源库为非腾讯云数据库，这里的地域选择与数据消费的地域保持一致，如果数据消费地域无特殊要求，这里选择离源数据库最近的一个地域即可。
- 数据库：请根据具体数据库类型进行选择。
- 版本：选择 **kafka 版**，支持通过 Kafka 客户端直接消费。
- 订阅实例名称：编辑当前数据订阅实例的名称。

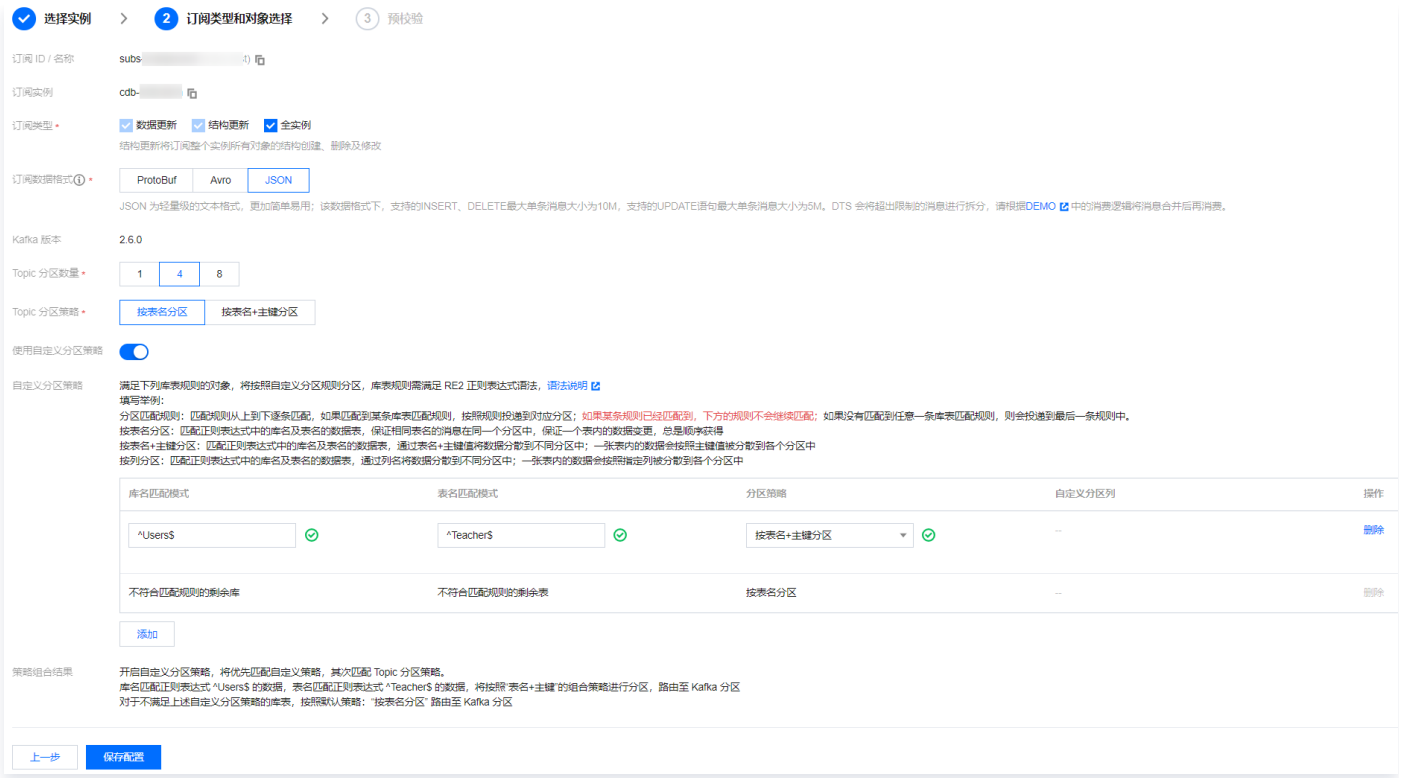
3. 购买成功后，返回数据订阅列表，单击**操作**列的**配置订阅**对刚购买的订阅进行配置，配置完成后才可以进行使用。

4. 在配置数据库订阅页面，选择源数据库相应配置，完成后单击**连通性测试**，测试通过后单击**下一步**。

配置项	说明
实例类型	购买时选择的数据库类型，不可修改。 建议选择主库进行订阅，订阅服务对源库的压力非常小。
接入类型	请根据您的场景选择，不同接入类型的准备工作请参考 准备工作概述 。 • 公网：源数据库可以通过公网 IP 访问。 • 云主机自建：源数据库部署在 腾讯云服务器 CVM 上。 • 专线接入：源数据库可以通过 专线接入 方式与腾讯云私有网络打通。 • VPN接入：源数据库可以通过 VPN 连接 方式与腾讯云私有网络打通。 • 云数据库：源数据库属于腾讯云数据库实例。 • 云联网：源数据库可以通过 云联网 与腾讯云私有网络打通。
公网	• 主机地址：源数据库 IP 地址或域名。 • 端口：源数据库使用的端口。

云主机自建	• 云主机实例：云服务器 CVM 的实例 ID。 • 端口：源数据库使用的端口。
专线接入	• 私有网络专线网关/：专线接入时只支持私有网络专线网关，请确认网关关联网络类型。 • 私有网络：选择私有网络专线网关和 VPN 网关关联的私有网络和子网。 • 主机地址：源数据库 IP 地址。 • 端口：源数据库使用的端口。
VPN 接入	• VPN 网关：VPN 网关，请选择通过 VPN 网关接入的 VPN 网关实例。 • 私有网络：选择私有网络专线网关和 VPN 网关关联的私有网络和子网。 • 主机地址：源数据库 IP 地址。 • 端口：源数据库使用的端口。
云数据库	云数据库实例：源数据库的实例 ID。 • 建议选择主库进行订阅，订阅服务对源库的压力非常小。 • 源数据库为腾讯云数据库 MySQL 时，支持选择只读实例、灾备实例，但使用只读实例时，需要先 提交工单 进行申请。腾讯云 MySQL 单节点（原基础版-已下线）作为源端，接入方式不能使用“云数据库”，可以使用“公网”接入方式；MySQL 单节点（云盘版）作为源端不限制，正常使用“云数据库”即可。 • 源数据库为腾讯云 TDSQL-C MySQL 时，支持选择只读实例。
云联网	• 主机地址：源数据库的主机 IP 地址。 • 端口：源数据库使用的端口。 • 私有网络云联网：云联网实例名称。 • 接入 VPC：接入 VPC 指的是云联网中接入订阅链路的 VPC。请在云联网关联的所有 VPC 中，选择除了源数据库所属 VPC 外的其他 VPC。 例如，广州地域数据库作为源数据库，则接入 VPC 选择其他地域，如成都 VPC 或者上海 VPC。 • 子网：已选择 VPC 网络的子网名称。 • 接入 VPC 地域：购买任务时选择的源数据库地域与接入 VPC 地域需要保持一致，如果不一致，DTS 会将购买任务中选择的源数据库地域，改为接入 VPC 地域。
账号/密码	账号/密码：源数据库的账号、密码。
连接方式	• SSL 安全连接指 DTS 与源数据库通过 SSL（Secure socket layer）安全连接，对传输链路进行加密。 • 选择 SSL 安全连接可能会增加数据库的连接响应时间，一般腾讯云内网链路相对较安全，无需开启 SSL 安全连接，采用公网/专线等传输方式，并且对数据安全要求较高的场景，需要开启 SSL 安全连接。选择 SSL 安全连接前，需要先在源数据库中开启 SSL 加密。
支持 XA 事务	勾选后 DTS 数据订阅可以识别源库进行的 XA 事务操作，在消费端获取的数据是按照 XA 事务的逻辑进行解析。 源库发生了 XA 事务回滚（rollback），DTS 可以识别 rollback，源库上 rollback 及 rollback 之前的 SQL 语句不会再传递到消费端。 当前仅 MySQL、TDSQL-C MySQL 的数据订阅支持 XA 事务。

5. 在订阅类型和对象选择页面，选择订阅类型，单击**保存配置**。



配置项	说明
订阅类型	• 数据更新：订阅源库全部对象的数据更新，包括数据 INSERT、UPDATE、DELETE 操作。 • 结构更新：订阅源库全部对象的结构创建、修改和删除。 • 全实例：订阅源库全部对象的数据更新和结构更新。
订阅数据格式	支持 ProtoBuf、Avro 和 JSON 三种格式。 ProtoBuf 和 Avro 采用二进制格式，消费效率更高，JSON 采用轻量级的文本格式，更加简单易用。
Topic 分区数量	设置数据投递到内置 kafka 中 Topic 的分区数量，增加分区数量可提高数据写入和消费的速度。单分区可以保障消息的顺序，多分区无法保障消息顺序，如果您对消费到消息的顺序有严格要求，请选择分区数量为1。
Topic 分区策略	• 按表名分区：将相同表名的数据写入同一个分区中。 • 表名+主键分区：将相同主键的数据会写入同一个分区。适用于热点数据的表，可以将热表数据分散到不同分区中，提升并发消费效率。
使用自定义分区策略	先通过正则表达式对订阅数据中的库名和表名进行匹配，将匹配到的数据按照表名、表名+主键、列进行分区投递。再将剩余未匹配到的数据，按照 Topic 分区策略进行投递。详情内容请参考 设置分区策略 。

6. 在预校验页面，预校验任务预计会运行2分钟 – 3分钟，预校验通过后，单击**启动**完成数据订阅任务配置。

> **说明：**
> 如果校验失败，请参考 校验不通过处理方法 进行修正，并重新进行校验。

7. 单击**启动**后，订阅任务会进行初始化，预计会运行3分钟 – 4分钟，初始化成功后进入**运行中**状态。
8. 新增消费组，数据订阅 Kafka 版支持用户创建多个消费组（单个订阅任务最多支持创建10个消费组），进行多点消费。数据订阅 Kafka 版消费依赖于 Kafka 的消费组，所以在消费数据前需要创建消费组。
9. 订阅实例进入运行中状态之后，就可以开始消费数据。Kafka 的消费需要进行密码认证，具体示例请参考 数据消费 Demo，我们提供了多种语言的 Demo 代码，也对消费的主要流程和关键的数据结构进行了说明。

创建 TDSQL MySQL 数据订阅

最近更新时间：2024-10-16 16:11:21

本场景介绍使用 DTS 创建腾讯云数据库 TDSQL MySQL 的数据订阅任务操作指导。

前提条件

- 已准备好待订阅的腾讯云数据库，并且数据库版本符合要求，请参见 数据订阅支持的数据库 。
- 已在源端实例中开启 binlog。
- 已在源端实例中创建订阅账号，需要账号权限如下：REPLICATION CLIENT、REPLICATION SLAVE、PROCESS 和全部对象的 SELECT 权限。
  具体授权语句如下：

```
create user '迁移账号' IDENTIFIED BY '账号密码';
grant SELECT, REPLICATION CLIENT,REPLICATION SLAVE,PROCESS on *.* to '迁移账号'@'%';
flush privileges;
```

约束限制

1. 订阅的消息保存在 DTS 内置 Kafka（单 Topic）中，目前默认保存时间为最近1天，单 Topic 的最大存储为500G，当数据存储时间超过1天，或者数据量超过500G时，内置 Kafka 都会开始清除最先写入的数据。所以请用户及时消费，避免数据在消费完之前就被清除。
2. 数据消费的地域需要与订阅任务所属的地域相同。
3. 当前不支持 geometry 相关的数据类型。
4. 数据订阅源是 TDSQL MySQL 版时，不支持直接执行授权语句授权，所以订阅账号的权限需要在 TDSQL 控制台 单击实例 ID，进入账号管理页中添加。
   订阅账号所需要的权限即上述授权语句中所示权限，对于为订阅账号进行 `__tencentdb__` 的授权操作，在控制台修改权限弹窗中选择对象级特权，勾选所有权限即可。
5. 数据订阅源是 TDSQL MySQL 版时，不支持订阅 二级分区 表。
   5.1 如果在订阅任务启动前源库创建了二级分区表，则校验任务不通过。
   5.2 如果在订阅任务运行中源库创建了二级分区表，那么订阅到二级分区表的数据是子表数据（订阅对象选择整库或者整实例，源库在订阅任务启动后创建的二级分区表也会被订阅，导致最终的结果订阅了二级分区表）。因为二级分区表的底层是通过子表实现，所以不建议用户在订阅任务过程中创建二级分区表，否则会导致如下示例的订阅数据差异。
   示例：源库表名为“test_a”是二级分区表，那么 DTS 订阅到该表的 DML 的表名为“test_a_tdsql_subp0/test_a_tdsql_subp1”。
6. 订阅任务过程中，如果进行修改订阅对象等操作会发生任务重启，重启后可能会导致用户在 kafka 客户端消费数据时出现重复。
   6.1 DTS 是按最小数据单元进行传输的，增量数据每标记一个 checkpoint 位点就是一个数据单元，如果重启时，刚好一个数据单元传输已完成，则不会导致数据重复；如果重启时，一个数据单元还正在传输中，那么再次启动后需要重新拉取这个数据单元，以保证数据完整性，这样就会导致数据重复。
   6.2 用户如果对重复数据比较关注，请自行在消费数据时设置去重逻辑。
7. 源端 TDSQL MySQL 中对表的数量有限制，整个实例最多为5000个，超出后 DTS 任务会报错；同时，表的数量太多会导致源端的访问耗时变大，引起性能抖动和下降。

注意事项

- 数据订阅源是 TDSQL MySQL 版的订阅任务，各个分片的 DDL 操作都会被订阅并投递到 Kafka，所以对于一个分表的 DDL 操作，会出现重复的 DDL 语句。例如，实例 A 有3个分片，订阅了一个分表 tableA，那么对于表 tableA 的 DDL 语句会订阅到3条。
- Kafka 中的每条消息的消息头中都带有分片信息，以 key/value 的形式存在消息头中，key 是 ShardId，value 是 SQL 透传 ID，可根据 SQL 透传 ID 区分该消息来自哪个分片。用户可以在 TDSQL 控制台 > **实例列表** > **分片管理**中查看 SQL 透传 ID。

支持订阅的 SQL 操作

操作类型	支持的 SQL 操作
DML	INSERT、UPDATE、DELETE
DDL	CREATE DATABASE、DROP DATABASE、CREATE TABLE、ALTER TABLE、DROP TABLE、RENAME TABLE

操作步骤

1. 登录 DTS 控制台，在左侧导航选择**数据订阅**，单击**新建数据订阅**。
2. 在新建数据订阅页，选择相应配置，单击**立即购买**。
   - 计费模式：支持包年包月和按量计费。
   - 地域：地域需与待订阅的数据库实例订阅保持一致。
   - 数据库：请根据具体数据库类型进行选择。
   - 版本：选择 **kafka 版**，支持通过 Kafka 客户端直接消费。
   - 订阅实例名称：编辑当前数据订阅实例的名称。
3. 购买成功后，返回数据订阅列表，单击**操作**列的**配置订阅**对刚购买的订阅进行配置，配置完成后才可以进行使用。
4. 在配置数据库订阅页面，选择相应配置，单击**下一步**。

配置项	说明
实例类型	购买时选择的数据库类型，不可修改。
接入类型	当前仅支持源数据库为腾讯云数据库实例的场景，请选择“云数据库”。
云数据库	云数据库实例：源数据库的实例 ID。 支持选择只读/灾备实例，但推荐选择主实例，订阅服务对源库的压力非常小。

账号/密码	账号/密码：源数据库的账号、密码。

5. 在订阅类型和对象选择页面，选择订阅类型，单击**保存配置**。

配置项	说明
订阅类型	• 数据更新：订阅源库全部对象的数据更新，包括数据 INSERT、UPDATE、DELETE 操作。 • 结构更新：订阅源库全部对象的结构创建、修改和删除。 • 全实例：订阅源库全部对象的数据更新和结构更新。
Kafka 版本	显示 DTS 内置 Kafka 的版本，不可修改。
Topic 分区数量	设置数据投递到内置 kafka 中 Topic 的分区数量，增加分区数量可提高数据写入和消费的速度。单分区可以保障消息的顺序，多分区无法保障消息顺序，如果您对消费到消息的顺序有严格要求，请选择分区数量为1。

6. 在预校验页面，预校验任务预计会运行2分钟 – 3分钟，预校验通过后，单击**启动**完成数据订阅任务配置。

> **说明：**
> 如果校验失败，请参考 校验不通过处理方法 进行修复，并重新进行校验。

7. 单击启动后，订阅任务会进行初始化，预计会运行3分钟 – 4分钟，初始化成功后进入**运行中**状态。
8. 新增消费组，数据订阅 Kafka 版支持用户创建多个消费组（单个订阅任务最多支持创建10个消费组），进行多点消费。数据订阅 Kafka 版消费依赖于 Kafka 的消费组，所以在消费数据前需要创建消费组。
9. 订阅实例进入运行中状态之后，就可以开始消费数据。Kafka 的消费需要进行密码认证，具体示例请参考 数据消费 Demo，我们提供了多种语言的 Demo 代码，也对消费的主要流程和关键的数据结构进行了说明。

创建 MariaDB 或 Percona 数据订阅

最近更新时间：2024-10-16 16:11:21

本场景介绍使用 DTS 创建 MariaDB、Percona 的数据订阅任务操作指导。因 MariaDB、Percona 的数据订阅操作一致，如下以 MariaDB 为例进行介绍。

前提条件

- 已准备好待订阅的源端数据库，并且数据库版本符合要求，请参见 数据订阅支持的数据库 。
- 已在源端实例中开启 Binlog。
- 已在源端实例中创建订阅账号，需要账号权限如下：REPLICATION CLIENT、REPLICATION SLAVE、PROCESS 和全部对象的 SELECT 权限。
  具体授权语句如下：

```
create user '账号' IDENTIFIED BY '密码';
grant SELECT, REPLICATION CLIENT,REPLICATION SLAVE,PROCESS on *.* to '账号'@'%';
flush privileges;
```

约束限制

- 订阅的消息保存在 DTS 内置 Kafka（单 Topic）中，目前默认保存时间为最近1天，单 Topic 的最大存储为500G，当数据存储时间超过1天，或者数据量超过500G时，内置 Kafka 都会开始清除最先写入的数据。所以请用户及时消费，避免数据在消费完之前就被清除。
- 数据消费的地域需要与订阅任务所属的地域相同。
- 当前不支持 geometry 相关的数据类型。
- 订阅任务过程中，如果进行修改订阅对象等操作会发生任务重启，重启后可能会导致用户在 kafka 客户端消费数据时出现重复。
  - DTS 是按最小数据单元进行传输的，增量数据每标记一个 checkpoint 位点就是一个数据单元，如果重启时，刚好一个数据单元传输已完成，则不会导致数据重复；如果重启时，一个数据单元还正在传输中，那么再次启动后需要重新拉取这个数据单元，以保证数据完整性，这样就会导致数据重复。
  - 用户如果对重复数据比较关注，请自行在消费数据时设置去重逻辑。

支持订阅的 SQL 操作

操作类型	支持的 SQL 操作
DML	INSERT、UPDATE、DELETE
DDL	CREATE DATABASE、DROP DATABASE、CREATE TABLE、ALTER TABLE、DROP TABLE、RENAME TABLE

操作步骤

1. 登录 DTS 控制台，在左侧导航选择**数据订阅**，单击**新建数据订阅**。
2. 在新建数据订阅页，选择相应配置，单击**立即购买**。
   - 计费模式：支持包年包月和按量计费。

- 地域：源库为腾讯云数据库，这里的地域需与源数据库实例的地域保持一致。源库为非腾讯云数据库，这里的地域选择与数据消费的地域保持一致，如果数据消费地域无特殊要求，这里选择离源数据库最近的一个地域即可。
- 数据库：请选择数据库类型。
- 版本：选择 **kafka 版**，支持通过 Kafka 客户端直接消费。
- 订阅实例名称：编辑当前数据订阅实例的名称。

3. 购买成功后，返回数据订阅列表，单击**操作**列的**配置订阅**对刚购买的订阅进行配置，配置完成后才可以进行使用。
4. 在配置数据库订阅页面，选择源数据库相应配置，完成后单击**连通性测试**，测试通过后单击**下一步**。

配置项	说明
实例类型	购买时选择的数据库类型。 建议选择主库进行订阅，订阅服务对源库的压力非常小。
接入类型	请根据您的场景选择，不同接入类型的准备工作请参考 准备工作概述 。 • 公网：源数据库可以通过公网 IP 访问。 • 云主机自建：源数据库部署在 腾讯云服务器 CVM 上。 • 专线接入：源数据库可以通过 专线接入 方式与腾讯云私有网络打通。 • VPN接入：源数据库可以通过 VPN 连接 方式与腾讯云私有网络打通。 • 云数据库：源数据库属于腾讯云数据库实例。 • 云联网：源数据库可以通过 云联网 与腾讯云私有网络打通。
公网	• 主机地址：源数据库 IP 地址或域名。 • 端口：源数据库使用的端口。
云主机自建	• 云主机实例：云服务器 CVM 的实例 ID。 • 端口：源数据库使用的端口。
专线接入	• 私有网络专线网关：专线接入时只支持私有网络专线网关，请确认网关关联网络类型。 • 私有网络：选择私有网络专线网关和 VPN 网关关联的私有网络和子网。 • 主机地址：源数据库 IP 地址。 • 端口：源数据库使用的端口。
VPN 接入	• VPN 网关：VPN 网关，请选择通过 VPN 网关接入的 VPN 网关实例。

	• 私有网络：选择私有网络专线网关和 VPN 网关关联的私有网络和子网。 • 主机地址：源数据库 IP 地址。 • 端口：源数据库使用的端口。
云数据库	云数据库实例：源数据库的实例 ID。 • 建议选择主库进行订阅，订阅服务对源库的压力非常小。 • 源数据库为腾讯云数据库 MariaDB 时，支持选择灾备/只读实例。
云联网	• 主机地址：源数据库的主机 IP 地址。 • 端口：源数据库使用的端口。 • 私有网络云联网：云联网实例名称。 • 接入 VPC：接入 VPC 指的是云联网中接入订阅链路的 VPC。请在云联网关联的所有 VPC 中，选择除了源数据库所属 VPC 外的其他 VPC。 例如，广州地域数据库作为源数据库，则接入 VPC 选择其他地域，如成都 VPC 或者上海 VPC。 • 子网：已选择 VPC 网络的子网名称。 • 接入 VPC 地域：购买任务时选择的源数据库地域与接入 VPC 地域需要保持一致，如果不一致，DTS 会将购买任务中选择的源数据库地域，改为接入 VPC 地域。
账号/密码	账号/密码：源数据库的账号、密码。

5. 在订阅类型和对象选择页面，选择订阅类型，单击**保存配置**。

配置项	说明
订阅类型	• 数据更新：订阅源库全部对象的数据更新，包括数据 INSERT、UPDATE、DELETE 操作。 • 结构更新：订阅源库全部对象的结构创建、修改和删除。 • 全实例：订阅源库全部对象的数据更新和结构更新。
Kafka 版本	显示 DTS 内置 Kafka 的版本，不可修改。
Topic 分区数量	设置数据投递到内置 kafka 中 Topic 的分区数量，增加分区数量可提高数据写入和消费的速度。单分区可以保障消息的顺序，多分区无法保障消息顺序，如果您对消费到消息的顺序有严格要求，请选择分区数量为1。
Topic 分区策略	• 按表名分区：将相同表名的数据写入同一个分区中。 • 表名 + 主键分区：将相同主键的数据会写入同一个分区。适用于热点数据的表，可以将热表数据分散到不同分区中，提升并发消费效率。

6. 在预校验页面，预校验任务预计会运行2分钟 – 3分钟，预校验通过后，单击**启动**完成数据订阅任务配置。

> **说明：**
> 如果校验失败，请参考 校验不通过处理方法 进行修正，并重新进行校验。

7. 订阅任务进行初始化，预计会运行3分钟 – 4分钟，初始化成功后进入**运行中**状态。
8. 新增消费组，数据订阅 Kafka 版支持用户创建多个消费组（单个订阅任务最多支持创建10个消费组），进行多点消费。数据订阅 Kafka 版消费依赖于 Kafka 的消费组，所以在消费数据前需要创建消费组。
9. 订阅实例进入运行中状态之后，就可以开始消费数据。Kafka 的消费需要进行密码认证，具体示例请参考 数据消费 Demo，我们提供了多种语言的 Demo 代码，也对消费的主要流程和关键的数据结构进行了说明。

消费 MySQL 系列订阅数据
消费 MySQL 系列订阅数据操作指导

最近更新时间：2024-09-24 18:02:21

操作场景

本操作适用于 MySQL/TDSQL-C MySQL/MariaDB/Percona/TDSQL MySQL 的数据订阅，数据订阅 Kafka 版（当前 Kafka Server 版本为V2.6.0）中，您可以通过0.11版本及以上的 Kafka 客户端 进行消费订阅数据，本文为您提供了 Java、Go、Python 语言的客户端消费 Demo 示例，方便您快速测试消费数据的流程，了解数据格式解析的方法。

注意事项

1. **Demo 并不包含消费数据的用法演示，仅对数据做了打印处理，您需要在此基础上自行编写数据处理逻辑**，您也可以使用其他语言的 Kafka 客户端消费并解析数据。
2. 目前不支持通过外网连接数据订阅的 Kafka 进行消费，只支持腾讯云内网的访问，并且订阅的数据库实例所属地域与数据消费的地域相同。
3. 数据格式选择 Avro、JSON 时，DTS 订阅对于大消息的处理存在如下限制。
   - 2024年3月25日之前创建的任务，DTS 订阅处理单条消息有一定上限，当源库中的单行数据超过5MB时，订阅任务可能会报错。
   - 2024年3月25日及之后创建的任务，当源库中的单行数据超过5MB时，订阅任务不会报错，使用最新的消费 Demo 即可获得大消息的消费数据。
4. 在订阅指定库/表对象（非源实例全部），并且采用 Kafka 单分区的场景中，DTS 解析增量数据后，仅将订阅对象的数据写入 Kafka Topic 中，其他非订阅对象的数据会转成空事务写入 Kafka Topic，所以在消费数据时会出现空事务。空事务的 Begin/Commit 消息中保留了事务的 GTID 信息，可以保证 GTID 的连续性和完整性，同时，在 MySQL/TDSQL-C MySQL 的消费 Demo 中，多个空事务也做了压缩处理以减少消息数量。
5. 为了保证数据可重入，DTS 订阅引入 Checkpoint 机制。消息写入 Kafka Topic 时，一般每10秒会插入一个 Checkpoint，用来标识数据同步的位点，在任务中断后再重启识别断点位置，实现断点续传。另外，消费端遇到 Checkpoint 消息会做一次 Kafka 消费位点提交，以便及时更新消费位点。
6. 数据格式选择 JSON 时，如果您使用过或者熟悉开源订阅工具 Canal，可以选择将这里消费出来的 JSON 格式数据转换成 Canal 工具兼容的数据格式，再进行后续处理，我们的 Demo 中已经提供了相关支持，在启动 Demo 的参数中添加参数 trans2canal 即可实现。目前该功能仅限 Java 语言支持。
7. 选择 ProtoBuf 和 JSON 数据格式时，对于源端 MySQL 的数值类型，DTS 写入 Kafka 时会转为字符串类型，其中 FLOAT 和 DOUBLE 类型在转为字符串时如果绝对值小于 1e-4 或大于等于 1e+6，那么结果将使用科学计数法表示。科学计数法以尾数乘以10的幂的形式表示浮点数，这样可以更清晰地表示其数量级和精确值，同时能够节省存储空间。如果在消费端用户需要转换为数值类型，请注意对科学记数法表示的浮点数字符串做特殊处理。
8. 订阅任务的数据格式选择 JSON 时，如果源库 MySQL 参数 `binlog_row_image` 设置为 `minimal`，因为 minimal 的实现为，只将有影响的列记录到 binlog，没有影响的列不记录，所以消费结果 UPDATE 语句中对于未更新字段的值是缺失的，DTS 为了与 NULL 值进行区分，使用 `##DTS_NA_VALUE##` 表示未更新字段的值。
   如果用户希望消费结果的 UPDATE 语句包含未更新字段的值，则需要将源端 MySQL 参数 `binlog_row_image` 设置为 `full`，然后重建订阅任务。
   ProtoBuf 与 Avro 格式由于是二进制格式，不需要通过特殊字符串区分缺失字段值与 NULL 值，所以不涉及该问题。

Demo 下载

- 在配置订阅任务中，您可以选择不同的订阅数据格式，ProtoBuf、Avro 和 JSON。ProtoBuf 和 Avro 采用二进制格式，消费效率更高，JSON 采用轻量级的文本格式，更加简单易用。订阅任务中选择了哪种格式，这里就需要下载对应格式的 Demo。
- 如下 Demo 示例中已包含了对应的 Protobuf/Avro/JSON 协议文件，您无需另外下载。如果您选择自行下载 Protobuf 协议文件，请使用 Protobuf 3.X 版本进行代码生成，以便数据结构可以正确兼容。
- Demo 中的逻辑讲解及关键参数说明，请参考 Demo 说明 。

Demo 语言	ProtoBuf（MySQL/MariaDB/TDSQL-C MySQL/Percona）	ProtoBuf（TDSQL MySQL）	Avro（MySQL/TDSQL-C MySQL）	Avro（MySQL/TDSQL-C MySQL）兼容阿里云订阅服务的数据格式	JSON（MySQL/TDSQL-C MySQL）
Go	地址	地址	地址	-	地址
Java	地址	地址	地址	地址	地址
Python	地址	地址	地址	-	地址

Java Demo 操作步骤

编译环境：Maven 或者 Gradle 包管理工具，JDK8。
运行环境：腾讯云服务器（需要与订阅实例相同地域，才能够访问到 Kafka 服务器的内网地址），安装 JRE8 。
操作步骤如下：

1. 创建新版数据订阅任务，详情请参见 数据订阅 Kafka 版 。
2. 创建一个或多个消费组，详情请参见 新增消费组 。
3. 下载 Java Demo ，然后解压该文件。
4. 进入解压后的目录，为方便使用，目录下分别放置了 Maven 模型文件、pom.xml 文件和 Gradle 相关配置文件，用户根据需要选用。
   - 使用 Maven 进行打包：mvn clean package 。
   - 使用 Gradle 进行打包：gradle fatJar 打包并包含所有依赖，或者 gradle jar 进行打包。
5. 运行 Demo。
   - 使用 Maven 打包后，进入目标文件夹 target ，运行

     ```
     java -jar sub_demo-1.0-SNAPSHOT-jar-with-dependencies.jar --brokers=xxx --topic=xxx --group=xxx --user=xxx --password=xxx --trans2sql
     ```

   - 使用 Gradle 打包后，进入文件夹 build/libs ，运行

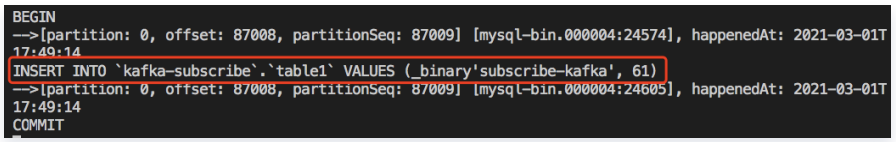
     ```
     java -jar sub_demo-with-dependencies-1.0-SNAPSHOT.jar --brokers=xxx --topic=xxx --group=xxx --user=xxx --password=xxx --trans2sql
     ```

   各参数详细说明如下：

参数	说明
brokers	数据订阅 Kafka 的内网访问地址，可在 订阅详情 页查看。
topic	数据订阅任务的订阅 topic，可在 订阅详情 页查看。
group	消费组名称。可在 消费管理 页查看。

user/password	消费账号和密码，可在 消费管理 页查看。
trans2sql	表示是否转换为 SQL 语句，携带该参数表示转换为 SQL 语句，不携带则不转换。
trans2canal	可选，仅对 JSON 数据格式支持，并限定 Java 语言。 数据格式选择 JSON 时，可以将 JSON 格式数据转换成 Canal 工具兼容的数据格式。携带该参数表示转换为 Canal 格式，不携带则不转换。

6. 观察消费情况。



用户也可以使用 IDE 进行编译打包，打包完成后，工程根目录 target 文件夹下的 sub_demo-1.0-SNAPSHOT-jar-with-dependencies 即为一个包含了所需依赖的可运行的 jar 包。

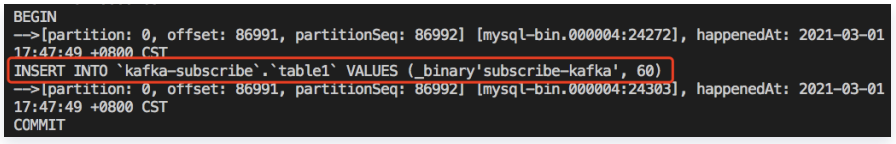
Golang Demo 操作步骤

编译环境：Golang 1.12 及以上版本，配置好 Go Module 环境。
运行环境：腾讯云服务器（需要与订阅实例相同地域，才能够访问到 Kafka 服务器的内网地址）。
操作步骤如下：

1. 创建新版数据订阅任务，详情请参见 数据订阅 Kafka 版 。
2. 创建一个或多个消费组，详情请参见 新增消费组 。
3. 下载 Golang Demo，然后解压该文件。
4. 进入解压后的目录，运行 `go build -o subscribe ./main` ，生成可执行文件 subscribe。
5. 运行如下代码。

   `/subscribe --brokers=xxx --topic=xxx --group=xxx --user=xxx --password=xxx --trans2sql=true`

   其中， `brokers` 为数据订阅 Kafka 的内网访问地址， `topic` 为数据订阅任务的订阅 topic，这两个可在 订阅详情 页查看， `group` 、 `user` 、 `password` 分别为消费组的名称、账号和密码，可在 消费管理 页查看， `trans2sql` 表示是否转换为 SQL 语句。
6. 观察消费情况。



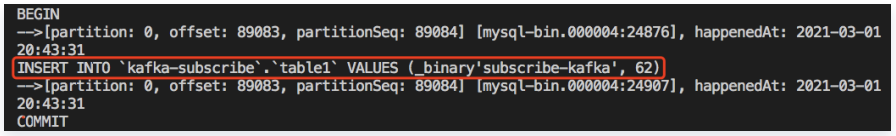
Python3 Demo 操作步骤

编译运行环境：腾讯云服务器（需要与订阅实例相同地域，才能够访问到 Kafka 服务器的内网地址），安装 Python3，pip3（用于依赖包安装）。
使用 pip3 安装依赖包：

```
pip install flag
pip install kafka-python
pip install protobuf
```

操作步骤如下：

1. 创建新版数据订阅任务，详情请参见 数据订阅 Kafka 版 。
2. 创建一个或多个消费组，详情请参见 新增消费组 。
3. 下载 Python3 Demo ，然后解压该文件。
4. 运行如下代码：
   `python main.py --brokers=xxx --topic=xxx --group=xxx --user=xxx --password=xxx --trans2sql=1`
   其中， `brokers` 为数据订阅 Kafka 的内网访问地址， `topic` 为数据订阅任务的订阅 topic，这两个可在 订阅详情 页查看， `group` 、 `user` 、 `password` 分别为消费组的名称、账号和密码，可在 消费管理 页查看， `trans2sql` 表示是否转换为 SQL 语句。
5. 观察消费情况。



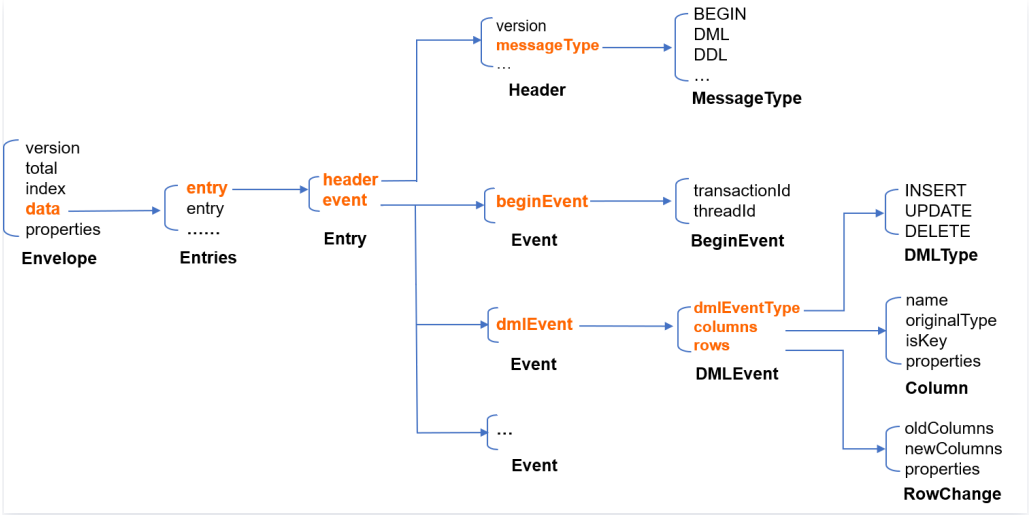
Demo 说明
ProtoBuf Demo 说明

最近更新时间：2024-08-22 14:42:31

Demo 关键逻辑讲解

消息生产逻辑

下文首先对消息生产逻辑进行简要说明，有助于用户理解消费逻辑。
我们采用 Protobuf 进行序列化，各语言 Demo 中均附带有 Protobuf 定义文件。文件中定义了几个关键结构：Envelope 是最终发送的 Kafka 消息结构；Entry 是单个订阅事件结构；Entries 是 Entry 的集合。主要数据结构关系如下所示：



生产过程如下：

1. 拉取 Binlog 消息，将每个 Binlog Event 编码为一个 Entry 结构体。

```
message Entry { //Entry 是单个订阅事件结构，一个事件相当于 MySQL 的一个 binlog event
 Header header = 1;  //事件头
 Event event   = 2;  //事件体
}


message Header {
 int32       version         = 1;   //Entry 协议版本
 SourceType  sourceType      = 2;   //源库的类型信息，包括 MySQL，Oracle 等类型
 MessageType messageType     = 3;   //消息的类型，也就是 Event 的类型，包括 BEGIN、
COMMIT、DML 等
 uint32 timestamp            = 4;   //Event 在原始 binlog 中的时间戳
 int64  serverId             = 5;   //源的 serverId
 string fileName             = 6;   //源 binlog 的文件名称
 uint64 position             = 7;   //事件在源 binlog 文件中的偏移量
 string gtid                 = 8;   //当前事务的 gtid
 string schemaName           = 9;   //变更影响的 schema
 string tableName            = 10;  //变更影响的 table
 uint64 seqId                = 11;  //全局递增序列号
```

```
 uint64 eventIndex          = 12;  //如果大的 event 分片，每个分片从0开始编号，当前版本无意义，留待后续扩展用
 bool   isLast              = 13;  //当前 event 是否 event 分片的最后一块，是则为 true，当前版本无意义，留待后续扩展用
 repeated KVPair properties = 15;
}


message Event {
 BeginEvent      beginEvent      = 1;  //binlog 中的 begin 事件
 DMLEvent        dmlEvent        = 2;  //binlog 中的 dml 事件
 CommitEvent     commitEvent     = 3;  //binlog 中的 commit 事件
 DDLEvent        ddlEvent        = 4;  //binlog 中的 ddl 事件
 RollbackEvent   rollbackEvent   = 5;  //rollback 事件，当前版本无意义
 HeartbeatEvent  heartbeatEvent  = 6;  //源库定时发送的心跳事件
 CheckpointEvent checkpointEvent = 7;  //订阅后台添加的 checkpoint 事件，每10秒自动生成一个，用于 Kafka 生产和消费位点管理
 repeated KVPair properties      = 15;
}
```

2. 为减少消息量，将多个 Entry 合并，合并后的结构为 Entries，Entries.items 字段即为 Entry 顺序列表。合并的数量以合并后不超过 Kafka 单个消息大小限制为标准。对单个 Event 就已超过大小限制的，则不再合并，Entries 中只有唯一 Entry 。

```
message Entries {
     repeated Entry items = 1; //entry list
}
```

3. 对 Entries 进行 Protobuf 编码得到二进制序列。
4. 将 Entries 的二进制序列放入 Envelope 的 data 字段。当存在单个 Binlog Event 过大时，二进制序列可能超过 Kafka 单个消息大小限制，此时我们会将其分割为多段，每段装入一个 Envelope。
   Envelope.total 和 Envelope.index 分别记录总段数和当前 Envelope 的序号（从0开始）。

```
message Envelope {
     int32  version                = 1; //protocol version, 决定了 data 内容如何解码
     uint32 total                  = 2;
     uint32 index                  = 3;
     bytes  data                   = 4; //当前 version 为1, 表示 data 中数据为 Entries 被 PB 序列化之后的结果
     repeated KVPair properties    = 15;
}
```

5. 对上一步生成的一个或多个 Envelope 依次进行 Protobuf 编码，然后投递到 Kafka 分区。同一个 Entries 分割后的多个 Envelope 顺序投递到同一个分区。

消息消费逻辑

下文对消费逻辑进行简要说明。我们提供的三种语言的 Demo 均遵循相同的流程。

1. 创建 Kafka 消费者。

2. 启动消费。
3. 依次消费原始消息，并根据消息中的分区找到分区对应的 partitionMsgConsumer 对象，由该对象对消息进行处理。
4. partitionMsgConsumer 将原始消息反序列化为 Envelope 结构。

```
// 将 Kafka 消息的 Value 值转换为 Envelope
envelope := subscribe.Envelope{}
err := proto.Unmarshal(msg.Value, &envelope)
```

5. partitionMsgConsumer 根据 Envelope 中记录的 index 和 total 连续消费一条或者多条消息，直到 Envlope.index 等于 Envelope.total-1（参见上面消费生产逻辑，表示收到了一个完整的 Entries ）。
6. 将收到的连续多条 Envelope 的 data 字段顺序组合到一起。将组合后的二进制序列用 Protobuf 解码为 Entries 。

```
if envelope.Index == 0 {
    pmc.completeMsg = envelope
} else {
    // 对进行过拆分的 Entries 二进制序列做拼接
    pmc.completeMsg.Data = append(pmc.completeMsg.Data, envelope.Data...)
}
if envelope.Index < envelope.Total-1 {
    return nil
}
// 将 Envelope.Data 反序列化为 Entries
entries := subscribe.Entries{}
err = proto.Unmarshal(pmc.completeMsg.Data, &entries)
```

7. 对 Entries.items 依次处理，打印原始 Entry 结构或者转化为 SQL 语句。
8. 当消费到 Checkpoint 消息时，做一次 Kafka 位点提交。Checkpoint 消息是订阅后台定时写入 Kafka 的特殊消息，每10秒一个。

数据库字段映射和存储

本节介绍数据库字段类型和序列化协议中定义的数据类型之间的映射关系。
源数据库（如 MySQL ）字段值在 Protobuf 协议中用如下所示的 Data 结构来存储。

```
message Data {
    DataType    dataType = 1;
    string      charset  = 2;  //DataType_STRING 的编码类型，值存储在 bv 里面
    string      sv       = 3;
//DataType_INT8/16/32/64/UINT8/16/32/64/Float32/64/DataType_DECIMAL 的字符串值
    bytes       bv       = 4;  //DataType_STRING/DataType_BYTES 的值
}
```

其中 DataType 字段代表存储的字段类型，可取枚举值如下图所示。

```
enum DataType {
    NIL     = 0; //值为 NULL
```

```
    INT8    = 1;
    INT16   = 2;
    INT32   = 3;
    INT64   = 4;
    UINT8   = 5;
    UINT16  = 6;
    UINT32  = 7;
    UINT64  = 8;
    FLOAT32 = 9;
    FLOAT64 = 10;
    BYTES   = 11;
    DECIMAL = 12;
    STRING  = 13;
    NA      = 14; //值不存在 (N/A)
}
```

其中 bv 字段存储 STRING 和 BYTES 类型的二进制表示，sv 字段存储 INT8/16/32/64/UINT8/16/32/64/DECIMAL 类型的字符串表示，charset 字段存储 STRING 的编码类型。
MySQL/TDSQL 原始类型与 DataType 映射关系如下（对 UNSIGNED 修饰的 MYSQL_TYPE_INT8/16/24/32/64 分别映射为 UINT8/16/32/32/64）：

说明：

1. 关于时区的相关说明。
   - `DATE`，`TIME`，`DATETIME` 类型不支持时区。
   - `TIMESTAMP` 类型支持时区，该类型字段表示：存储时，系统会从当前时区转换为 UTC（Universal Time Coordinated）进行存储；查询时，系统会从 UTC 转换为当前时区进行查询。
   - 综上，如下表中 "MYSQL_TYPE_TIMESTAMP" 和 "MYSQL_TYPE_TIMESTAMP_NEW" 字段会携带时区信息，用户在消费数据时可自行转换。（例如，DTS 输出的时间格式是带时区的字符串"2021-05-17 07:22:42 +00:00"，其中，"+00:00"表示 UTC 时间，用户在解析和转换的时候需要考虑时区信息。）
2. 对于源端 MySQL 的数值类型，DTS 写入 Kafka 时会转为字符串类型，其中 FLOAT 和 DOUBLE 类型在转为字符串时如果绝对值小于 1e-4 或大于等于 1e+6，那么结果将使用科学计数法表示。科学计数法以尾数乘以10的幂的形式表示浮点数，这样可以更清晰地表示其数量级和精确值，同时能够节省存储空间。如果在消费端用户需要转换为数值类型，请注意对科学记数法表示的浮点数字符串做特殊处理。

MySQL 字段类型（TDSQL 支持与 MySQL 相同的类型）	对应的 Protobuf DataType 枚举值
MYSQL_TYPE_NULL	NIL
MYSQL_TYPE_INT8	INT8
MYSQL_TYPE_INT16	INT16
MYSQL_TYPE_INT24	INT32
MYSQL_TYPE_INT32	INT32
MYSQL_TYPE_INT64	INT64
MYSQL_TYPE_BIT	INT64

MYSQL_TYPE_YEAR	INT64
MYSQL_TYPE_FLOAT	FLOAT32
MYSQL_TYPE_DOUBLE	FLOAT64
MYSQL_TYPE_VARCHAR	STRING
MYSQL_TYPE_STRING	STRING
MYSQL_TYPE_VAR_STRING	STRING
MYSQL_TYPE_TIMESTAMP	STRING
MYSQL_TYPE_DATE	STRING
MYSQL_TYPE_TIME	STRING
MYSQL_TYPE_DATETIME	STRING
MYSQL_TYPE_TIMESTAMP_NEW	STRING
MYSQL_TYPE_DATE_NEW	STRING
MYSQL_TYPE_TIME_NEW	STRING
MYSQL_TYPE_DATETIME_NEW	STRING
MYSQL_TYPE_ENUM	STRING
MYSQL_TYPE_SET	STRING
MYSQL_TYPE_DECIMAL	DECIMAL
MYSQL_TYPE_DECIMAL_NEW	DECIMAL
MYSQL_TYPE_JSON	BYTES
MYSQL_TYPE_BLOB	BYTES
MYSQL_TYPE_TINY_BLOB	BYTES
MYSQL_TYPE_MEDIUM_BLOB	BYTES
MYSQL_TYPE_LONG_BLOB	BYTES
MYSQL_TYPE_GEOMETRY	BYTES

Avro Demo 说明

最近更新时间：2024-09-24 18:02:21

消费程序说明

腾讯云 DTS 针对 Avro 数据格式，提供如下两种消费程序。

- Avro Demo（普通）：腾讯云 DTS 数据订阅 Avro 格式的消费程序。
- Avro Demo（兼容阿里云订阅服务的数据格式）：腾讯云 DTS 数据订阅 Avro 格式的消费程序，相对普通的程序，这个程序增加了一层转换操作，转化后可兼容阿里云订阅服务的数据格式，适用于从阿里云订阅服务切换过来的用户。

  具体场景为：用户之前使用阿里云 DTS 订阅服务（Avro 格式），后来切换到腾讯云，使用腾讯云 DTS 的订阅服务，腾讯云 DTS 提供的消费程序可兼容之前阿里云订阅服务的数据格式，这样减少用户在消费端的适配成本，仅对消费程序进行简单适配即可快速消费。

普通 Avro Demo 字段解释

Demo 中的文件说明如下，以 Java Demo 为例进行介绍。

- `consumerDemo-avro-java\src\main\resources\avro-tools-1.8.2.jar`：是用来生成 Avro 协议相关代码的工具。
- `consumerDemo-avro-java\src\main\java\com\tencent\subscribe\avro`：Avro 工具生成代码的目录。
- `consumerDemo-avro-java\src\main\resources\Record.avsc`：协议定义文件。

Record.avsc 中我们定义了14个结构（Avro 中叫做 schema），其中主要的数据结构为 Record，用于表示 binlog 中的一条数据，Record 的结构如下，其他数据结构可以在 Record.avsc 中查看：

```
 {
    "namespace": "com.tencent.subscribe.avro",     //Record.avsc 中的最后1个
schema，"name" 显示为 "Record"
    "type": "record",
    "name": "Record",      //"name" 显示为 "Record"，表示从 kafka 中消费的数据格式
    "fields": [
      {
        "name": "id",      //id 表示全局递增 ID，更多 record 取值解释如下表
        "type": "long",
        "doc": "unique id of this record in the whole stream"
      },
      {
        "name": "version",  //version 表示协议版本
        "type": "int",
        "doc": "protocol version"
      },
      {
        "name": "messageType",    //消息类型
        "aliases": [
          "operation"
        ],
        "type": {
          "namespace": "com.tencent.subscribe.avro",
          "name": "MessageType",
          "type": "enum",
```

```
            "symbols": [
              "INSERT",
              "UPDATE",
              "DELETE",
              "DDL",
              "BEGIN",
              "COMMIT",
              "HEARTBEAT",
              "CHECKPOINT",
              "ROLLBACK"
            ]
          }
        },
        {
          ......
        },
    }
```

Record 中的字段类型解释如下：

Record 中的字段名称	说明
id	全局递增 ID。
version	协议版本，当前版本为1。
messageType	消息类型，枚举 值："INSERT"，"UPDATE"，"DELETE"，"DDL"，"BEGIN"，"COMMIT"，"HEARTBEAT"，"CHECKPOINT"。
fileName	当前 record 所在的 binlog 文件名。
position	当前 record 的在 binlog 中结束的偏移量，格式为 End_log_pos@binlog 文件编号。例如，当前 record 位于文件 mysql-bin.000004 中，结束偏移量为2196，则其值为"2196@4"。
safePosition	当前事务在 binlog 中开始的偏移量，格式同上。
timestamp	写入 binlog 的时间，unix 时间戳，秒级。 binlog 记录的事务中对应 event header 里面的 timestamp，源端长事务操作可能会导致 timestamp 与 readerTimestamp 有时间差，这种属于正常情况。
gtid	当前的 gtid，如：c7c98333-6006-11ed-bfc9-b8cef6e1a231:9。
transactionId	事务 ID，只有 commit 事件才会生成事务 ID。
serverId	源库 serverId，查看源库 server_id 参考 SHOW VARIABLES LIKE 'server_id'。
threadId	提交当前事务的会话 ID，参考 SHOW processlist;。
sourceType	源库的数据库类型，当前版本只有 MySQL。
sourceVersion	源库版本，查看源库版本参考

	`select version();` 。
schemaName	库名。
tableName	表名。
objectName	格式为：库名.表名。
columns	表中各列的定义。
oldColumns	DML 执行前该行的数据，如果是 insert 消息，该数组为 null。数组中元素共有12种类型：Integer, Character, Decimal, Float, Timestamp, DateTime, TimestampWithTimeZone, BinaryGeometry, TextGeometry, BinaryObject, TextObject, EmptyObject，详见 demo 中定义。
newColumns	DML 执行后该行的数据，如果是 delete 消息，该数组为 null。数组中元素共有12种类型：Integer, Character, Decimal, Float, Timestamp, DateTime, TimestampWithTimeZone, BinaryGeometry, TextGeometry, BinaryObject, TextObject, EmptyObject，详见 demo 中定义。
sql	DDL 的 SQL 语句。
executionTime	DDL 执行时长，单位为秒。
heartbeatTimestamp	心跳消息的时间戳，秒级。只有 heartbeat 消息才有该字段。
syncedGtid	DTS 已解析 GTID 集合，格式形如：c7c98333-6006-11ed-bfc9-b8cef6e1a231:1-13。
fakeGtid	是否为构造的假 GTID，如未开启 gtid_mode，则 DTS 会构造一个 GTID。
pkNames	如果源库的表设有主键，则 DML 消息中会携带该参数，否则不会携带。
readerTimestamp	DTS 处理这条数据的时间，unix 时间戳，单位为毫秒数。
tags	QueryEvent 中的 status_vars，详细参考 QueryEvent 。
total	如果消息分片，记录分片总数。当前版本 (version=1) 无意义，预留扩展。
index	如果消息分片，记录当前分片的索引。当前版本 (version=1) 无意义，预留扩展。

Record 中描述列属性的字段为 "Field"，包含如下四个属性：

- name：列名。
- dataTypeNumber：是 binlog 中记录的数据类型。取值详见 MySQL 。
- isKey：是否主键。
- originalType：DDL 中定义的类型。

兼容阿里云订阅服务数据格式的 Avro Demo 字段解释

消费 Demo 中的关键文件说明如下：

文件/目录	说明
resources/Record.avs	腾讯云 avro 数据结构定义文件（为了便于转换为阿里云的格式，这里借用了 Record-

c	ali.avsc 中定义的基本类型如 Integer）。
resources/Record-ali.avsc	阿里云 avro 数据结构定义文件。
java/com/ClassConvertor	用于将腾讯云数据结构转阿里云数据结构的工具类。
java/com/UserBusinessProcess	用户业务逻辑类。Kafka 消息反序列化并转为阿里云的数据格式后传递到这里，然后执行一定的业务逻辑。 本 Demo 中仅实现了打印或者解析为 SQL 语句的功能。请用户根据业务需要自行在此扩展，也可以将之前阿里云消费程序中的业务处理代码迁移到此，简单适配后即可编译运行。

数据格式为 com.alibaba.dts.formats.avro.Record，具体字段说明如下：

字段名称	类型	说明
version	int	协议版本，当前版本为2。
id	long	全局递增 Id。
sourceTimestamp	long	当前 record 在 binlog event 里的时间戳，秒级。
sourcePosition	String	当前 record 的在 binlog 中结束的偏移量，格式为 End_log_pos@binlog 文件编号。比如，当前 record 位于文件 mysql-bin.000004 中，结束偏移量为 2196，则其值为"2196@4"。
safeSourcePosition	String	当前事务在 binlog 中开始的偏移量，格式同上。
sourceTxid	String	事务 Id，只有 COMMIT 记录才会携带该字段。
source	com.alibaba.dts.formats.avro.Source	源库类型与版本信息。
operation	com.alibaba.dts.formats.avro.Operation	消息类型，支持 INSERT、UPDATE、DELETE、DDL、BEGIN、COMMIT、ROLLBACK、HEARTBEAT、CHECKPOINT。
objectName	String	库名.表名。
processTimestamps	List	DTS 订阅服务处理数据的时间戳，未实现。
tags	Map<String,String>	一些额外信息，当前支持的 key 包括 thread_id, server_id, pk_uk_info, readerThroughoutTime, GTID, GTID_SET。
fields	Object	表中各列的定义，即表头。
beforeImages	Object	DML 执行前该行的数据，如果是 insert 消息，该数组为 null。数组中元素共有12种类型：Integer, Character, Decimal, Float, Timestamp, DateTime,

		TimestampWithTimeZone, BinaryGeometry, TextGeometry, BinaryObject, TextObject, EmptyObject。
afterImages	Object	DML 执行后该行的数据，如果是 delete 消息，该数组为 null。数组中元素共有12种类型：Integer, Character, Decimal, Float, Timestamp, DateTime, TimestampWithTimeZone, BinaryGeometry,TextGeometry, BinaryObject, TextObject, EmptyObject。
bornTimestamp	long	未实现，腾讯云 DTS 订阅服务中无对应字段。

数据库字段映射关系

如下为数据库（如 MySQL）字段类型和 Avro 协议中定义的数据类型之间的映射关系。

MySQL 类型	对应 Avro 中的类型
MYSQL_TYPE_NULL	EmptyObject
MYSQL_TYPE_INT8	Integer
MYSQL_TYPE_INT16	Integer
MYSQL_TYPE_INT24	Integer
MYSQL_TYPE_INT32	Integer
MYSQL_TYPE_INT64	Integer
MYSQL_TYPE_BIT	Integer
MYSQL_TYPE_YEAR	DateTime
MYSQL_TYPE_FLOAT	Float
MYSQL_TYPE_DOUBLE	Float
MYSQL_TYPE_VARCHAR	Character
MYSQL_TYPE_STRING	Character，如果原类型为 binary，则对应 BinaryObject
MYSQL_TYPE_VAR_STRING	Character，如果原类型为 varbinary，则对应 BinaryObject
MYSQL_TYPE_TIMESTAMP	Timestamp
MYSQL_TYPE_DATE	DateTime
MYSQL_TYPE_TIME	DateTime
MYSQL_TYPE_DATETIME	DateTime
MYSQL_TYPE_TIMESTAMP_NEW	Timestamp
MYSQL_TYPE_DATE_NEW	DateTime
MYSQL_TYPE_TIME_NEW	DateTime

MYSQL_TYPE_DATETIME_NEW	DateTime
MYSQL_TYPE_ENUM	TextObject
MYSQL_TYPE_SET	TextObject
MYSQL_TYPE_DECIMAL	Decimal
MYSQL_TYPE_DECIMAL_NEW	Decimal
MYSQL_TYPE_JSON	TextObject
MYSQL_TYPE_BLOB	BinaryObject
MYSQL_TYPE_TINY_BLOB	BinaryObject
MYSQL_TYPE_MEDIUM_BLOB	BinaryObject
MYSQL_TYPE_LONG_BLOB	BinaryObject
MYSQL_TYPE_GEOMETRY	BinaryObject

JSON Demo 说明

最近更新时间：2025-02-12 16:38:42

Demo 中我们采用 JSON 进行序列化，各语言 Demo 中均附带有 Record 定义文件。
Java Demo 中的定义文件路径为：`consumerDemo-json-java\src\main\java\json\FlatRecord.java`。

Record 中字段类型

Record 中的字段名称	说明
id	全局递增 ID。
version	协议版本，当前版本为1。
messageType	消息类型，枚举 值："INSERT"，"UPDATE"，"DELETE"，"DDL"，"BEGIN"，"COMMIT"，"HEARTBEAT"，"CHECKPOINT"。
fileName	当前 record 所在的 binlog 文件名。
position	当前 record 的在 binlog 中结束的偏移量，格式为 End_log_pos@binlog 文件编号。例如，当前 record 位于文件 mysql-bin.000004 中，结束偏移量为2196，则其值为"2196@4"。
safePosition	当前事务在 binlog 中开始的偏移量，格式同上。
timestamp	写入 binlog 的时间，unix 时间戳，秒级。 binlog 记录的事务中对应 event header 里面的 timestamp，源端长事务操作可能会导致 timestamp 与 readerTimestamp 有时间差，这种属于正常情况。
gtid	当前的 gtid，如：c7c98333-6006-11ed-bfc9-b8cef6e1a231:9。
transactionId	事务 ID，只有 commit 事件才会生成事务 ID。
serverId	源库 serverId，查看源库 server_id 参考 SHOW VARIABLES LIKE 'server_id'。
threadId	提交当前事务的会话 ID，参考 SHOW processlist;。
sourceType	源库的数据库类型，当前版本只有 MySQL。
sourceVersion	源库版本，查看源库版本参考 `select version();`。
schemaName	库名。
tableName	表名。
objectName	格式为：库名.表名。
columns	表中各列的定义。
oldColumns	DML 执行前该行的数据，如果是 insert 消息，该数组为 null。
newColumns	DML 执行后该行的数据，如果是 delete 消息，该数组为 null。
sql	DDL 的 SQL 语句。

executionTime	DDL 执行时长，单位为秒。
heartbeatTimestamp	心跳消息的时间戳，秒级。只有 heartbeat 消息才有该字段。
syncedGtid	DTS 已解析 GTID 集合，格式形如：c7c98333-6006-11ed-bfc9-b8cef6e1a231:1-13。
fakeGtid	是否为构造的假 GTID，如未开启 gtid_mode，则 DTS 会构造一个 GTID。
pkNames	如果源库的表设有主键，则 DML 消息中会携带该参数，否则不会携带。
readerTimestamp	DTS 处理这条数据的时间，unix时间戳，单位为毫秒数。
tags	QueryEvent 中的 status_vars，详细参考 QueryEvent 。
total	如果消息分片，记录分片总数。当前版本 (version = 1) 无意义，预留扩展。
index	如果消息分片，记录当前分片的索引。当前版本 (version = 1) 无意义，预留扩展。

Record 中 MySQL 列属性

- name：列名。
- dataTypeNumber：是 binlog 中记录的数据类型。取值详见 MySQL 。
- isKey：是否主键。
- originalType：DDL 中定义的类型。

MySQL 数据类型转换逻辑

在 JSON 协议中，将 MySQL 类型全部转换成了字符串。

- 对于 varchar 等字符串类型，全部转成了 UTF8 编码。
- 对于数值类型，全部转成了与值相同的字符串，如 "3.0"。
- 对于时间类型，格式为：YYYY-MM-DD HH:mm:ss.SSS。
- 对于时间戳类型，输出为毫秒数。
- 对于 binary、blob 等二进制类型，输出为与16进制值相同的字符串，如 "0xfff"。

使用 Flink 消费数据
使用 Flink 消费 MySQL 或 TDSQL-C MySQL 订阅数据

最近更新时间：2023-10-31 15:56:51

操作场景

数据订阅 Kafka 版（当前 Kafka Server 版本为V2.6.0）中，针对 Avro 格式的订阅数据，可以使用 Flink 客户端（仅支持客户端类型为 DataStream API）进行消费，本场景为您提供使用 flink-dts-connector 进行数据消费的 Demo 示例。

前提条件

1. 已 创建数据消费任务 。
2. 已 创建消费组 。
3. 已安装 Flink 并能够正常执行 Flink 任务。

注意事项

- **Demo 并不包含消费数据的用法演示，仅对数据做了打印处理，您需要在此基础上自行编写数据处理逻辑**，您也可以使用其他语言的 Kafka 客户端消费并解析数据。
- 目前不支持通过外网连接数据订阅的 Kafka 进行消费，只支持腾讯云内网的访问，并且订阅的数据库实例所属地域与数据消费的地域相同。
- DTS 订阅中内置的 Kafka 处理单条消息有一定上限，当源库中的单行数据超过5MB时，订阅任务可能会报错。
- 在订阅指定库/表对象（非源实例全部），并且采用 Kafka 单分区的场景中，DTS 解析增量数据后，仅将订阅对象的数据写入 Kafka Topic 中，其他非订阅对象的数据会转成空事务写入 Kafka Topic，所以在消费数据时会出现空事务。空事务的 Begin/Commit 消息中保留了事务的 GTID 信息，可以保证 GTID 的连续性和完整性，多个空事务也做了压缩处理以减少消息数量。
- 为了保证数据可重入，DTS 订阅引入 checkpoint 机制。消息写入 Kafka Server 时，一般每10秒会插入一个checkpoint，用来标识数据同步的位点，在任务中断后再重启识别断点位置，实现断点续传。另外，消费端遇到 checkpoint 消息会做一次 Kafka 消费位点提交，以此来实现消费端数据可重入。

消费 Demo 下载

Demo 中的逻辑讲解及关键参数说明，请参考 Avro Demo 说明（Flink）。

Demo 语言	Avro（MySQL/TDSQL-C MySQL）
Java	地址

Java Flink Demo 操作步骤

编译环境：Maven 或者 Gradle 包管理工具，JDK8。用户可自行选择打包工具，如下以 Maven 为例进行介绍。
运行环境：腾讯云服务器（需要与订阅实例相同地域，才能够访问到 Kafka 服务器的内网地址），安装 JRE8。
操作步骤：

1. 下载 Java Flink Demo，然后解压该文件。

2. 进入解压后的目录，为方便使用，目录下分别放置了 Maven 模型文件、pom.xml 文件，用户根据需要选用。
   `java -jar avro-tools-1.8.2.jar compile -string schema Record.avsc` ：代码生成路径。
3. 在 pom.xml 文件中修改 Flink 的版本，如下代码中的 version 需要与客户使用的 Flink 版本保持一致。

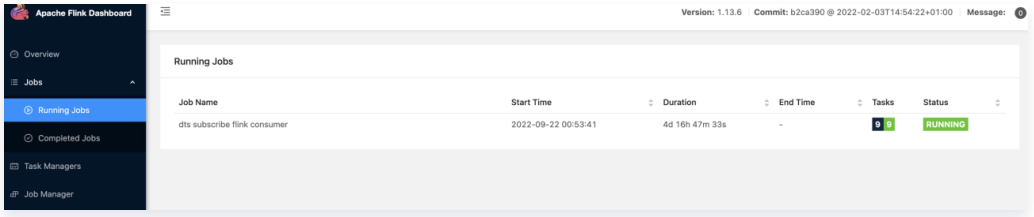
```
<dependency>
          <groupId>org.apache.flink</groupId>
          <artifactId>flink-connector-kafka_${scala.binary.version}</artifactId>
          <version>1.13.6</version>
</dependency>
<dependency>
          <groupId>org.apache.flink</groupId>
          <artifactId>flink-streaming-java_${scala.binary.version}</artifactId>
          <version>1.13.6</version>
          <scope>provided</scope>
</dependency>
<dependency>
          <groupId>org.apache.flink</groupId>
          <artifactId>flink-avro</artifactId>
          <version>1.13.6</version>
</dependency>
```

4. 进入 pom 文件所在的目录，使用 Maven 进行打包，也直接使用 IEDA 打包。
   使用 Maven 进行打包：mvn clean package。
5. 针对 Flink 客户端类型为 DataStream API 的场景，使用 Flink 客户端命令提交任务，启动消费。

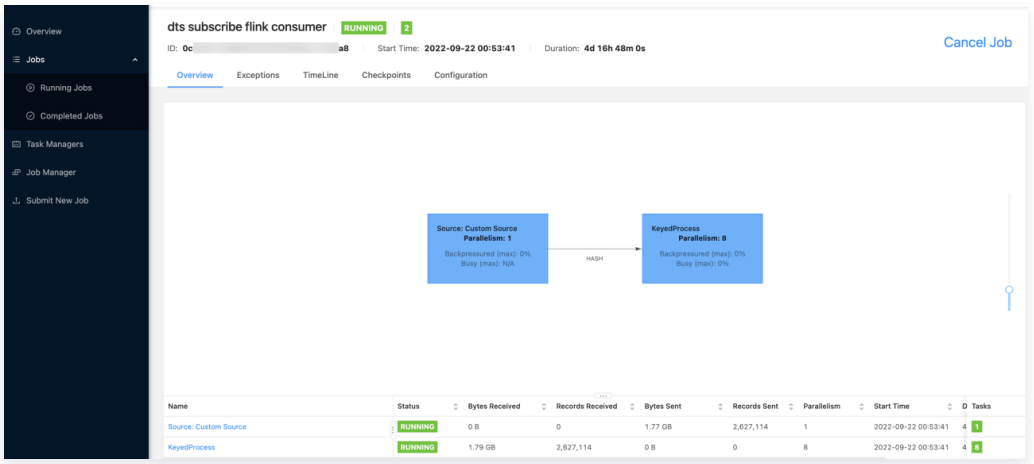
```
./bin/flink run consumerDemo-avro-flink-1.0-SNAPSHOT.jar --brokers xxx --topic xxx --group xxx --user xxx --password xxx –trans2sql
```

- `brokers` 为数据订阅 Kafka 的内网访问地址，`topic` 为数据订阅任务的订阅 topic，这两个可在 订阅详情 页查看。
- `group` 、`user` 、`password` 分别为消费组的名称、账号和密码，可在 消费管理 页查看。
- `trans2sql` 表示是否转换为 SQL 语句，java 代码中，携带该参数表示转换为 SQL 语句，不携带则不转换。

6. 观察消费情况。
   - 查看正在运行的任务。



- 查看任务详情。


Overview
Jobs
Running Jobs
Completed Jobs
Task Managers
Job Manager
Submit New Job
dts subscribe flink consumer
RUNNING
2
Start Time: 2022-09-22 00:53:41
Duration: 4d 16h 48m 0s
Cancel Job
Overview
Exceptions
TimeLine
Checkpoints
Configuration
Source: Custom Source
Parallelism: 1
Backpressured (max): 0%
Busy (max): N/A
HASH
KeyedProcess
Parallelism: 8
Backpressured (max): 0%
Busy (max): 0%
Name
Status
Bytes Received
Records Received
Bytes Sent
Records Sent
Parallelism
Start Time
Tasks
Source: Custom Source
RUNNING
0 B
0
1.77 GB
2,627,114
1
2022-09-22 00:53:41
KeyedProcess
RUNNING
1.79 GB
2,627,114
0 B
0
8
2022-09-22 00:53:41

使用 Flink 消费 TDSQL MySQL 订阅数据

最近更新时间：2023-10-31 15:56:51

操作场景

数据订阅 Kafka 版（当前 Kafka Server 版本为V2.6.0）中，针对 Protobuf 格式的订阅数据，可以使用 Flink 客户端（仅支持客户端类型为 DataStream API）进行消费，本场景为您提供使用 **consumer-demo-tdsql-pb-flink** 进行数据消费的 Demo 示例，方便您快速测试消费数据的流程，了解数据格式解析的方法。

前提条件

1. 已 创建数据消费任务 。
2. 已 创建消费组 。
3. 已安装 Flink 运行环境，并能够正常执行 Flink 任务。

注意事项

- **Demo 并不包含消费数据的用法演示，仅对数据做了打印处理，您需要在此基础上自行编写数据的处理逻辑。**
- 目前不支持通过外网连接数据订阅的 Kafka 进行消费，**只支持腾讯云内网的访问**，并且订阅的数据库实例所属地域与数据消费的地域相同。
- 在订阅指定库/表对象（非源实例全部），并且采用 Kafka 单分区的场景中，DTS 解析增量数据后，仅将订阅对象的数据写入 Kafka Topic 中，其他非订阅对象的数据会转成空事务写入 Kafka Topic，所以在消费数据时会出现空事务。空事务的 Begin/Commit 消息中保留了事务的 GTID 信息，可以保证 GTID 的连续性和完整性。
- 为了保证数据可重入，DTS 订阅引入 Checkpoint 机制。消息写入 Kafka Topic 时，一般每10秒会插入一个 Checkpoint，用来标识数据同步的位点，在任务中断后再重启识别断点位置，实现断点续传。另外，消费端遇到 Checkpoint 消息会做一次 Kafka 消费位点提交，以便及时更新消费位点。

消费 Demo下载

Demo 中的逻辑讲解及关键参数说明，请参考 Protobuf Demo 说明（Flink）。
在配置订阅任务中，TDSQL MySQL 支持的订阅数据格式为 Protobuf。Protobuf 采用二进制格式，消费效率更高。如下 Demo 中已包含 Protobuf 协议文件，无需另外下载。如果您选择自行下载 Protobuf 协议文件，请使用 Protobuf 3.X 版本进行代码生成，以便数据结构可以正确兼容。

Demo 语言	Protobuf（TDSQL MySQL）
Java	地址

Java Flink Demo 操作步骤

编译环境： Maven 包管理工具，JDK8。用户可自行选择打包工具，如下以 Maven 为例进行介绍。
运行环境： 腾讯云服务器（需要与订阅实例相同地域，才能够访问到 Kafka 服务器的内网地址），安装 JRE8。
操作步骤：

1. 下载 **consumer-demo-tdsql-pb-flink.zip** ，然后解压该文件。
2. 进入解压后的目录，为方便使用，目录下已放置了 pom.xml 文件，用户需要修改 Flink 的版本，确保 Flink 集群的版本与 pom.xml 依赖的 Flink 的版本相同。
3. 如下代码中的 ${flink.version} 需要与集群的 Flink 版本保持一致。

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-streaming-java_${scala.binary.version}</artifactId>
    <version>${flink.version}</version>
    <scope>provided</scope>
</dependency>

<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-connector-kafka_${scala.binary.version}</artifactId>
    <version>${flink.version}</version>
    <exclusions>
        <exclusion>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
        </exclusion>
    </exclusions>
</dependency>

<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-clients_${scala.binary.version}</artifactId>
    <version>${flink.version}</version>
</dependency>
```

4. 进入 pom 文件所在的目录，使用 Maven 进行打包，也直接使用 IEDA 打包。
   使用 Maven 进行打包：mvn clean package。
5. 针对 Flink 客户端类型为 DataStream API 的场景，使用 Flink 客户端命令提交 job 到 Flink 集群，启动消费。

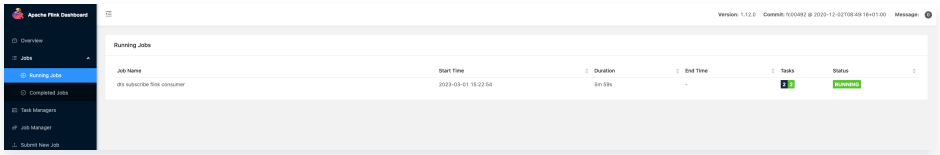
```
./bin/flink run consumer-demo-tdsql-pb-flink.jar --brokers xxx --topic xxx --group xxx --user xxx --password xxx --trans2sql
```

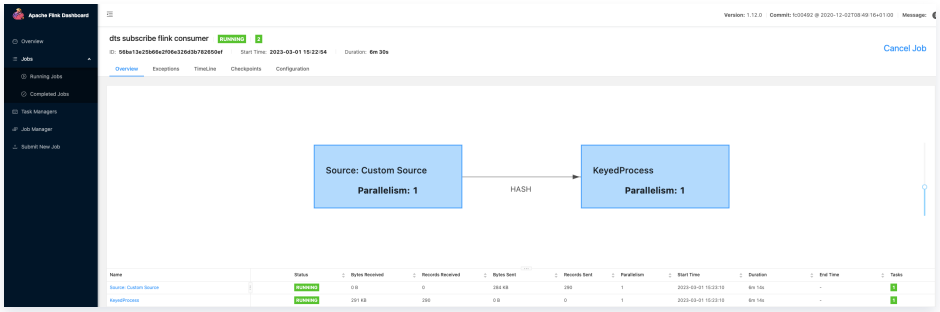
- `brokers` 为数据订阅 Kafka 的内网访问地址，`topic` 为数据订阅任务的订阅 topic，这两个可在 订阅详情 页查看。
- `group`、`user`、`password` 分别为消费组的名称、账号和密码，可在 消费管理 页查看。
- `trans2sql` 表示是否转换为 SQL 语句，java 代码中，携带该参数表示转换为 SQL 语句，不携带则不转换。

6. 在源数据库执行 DML 语句并观察 Flink 中所提交的 job 的消费情况。

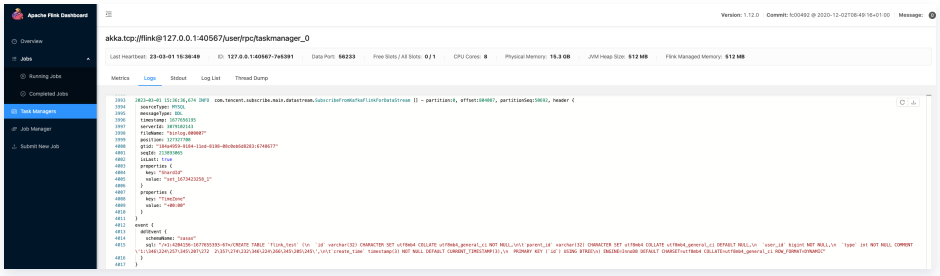
```
CREATE TABLE `flink_test` (
  `id` varchar(32) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
    `parent_id` varchar(32) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci
DEFAULT NULL,
  `user_id` bigint NOT NULL,
  `type` int NOT NULL COMMENT '1:支出  2: 收入',
    `create_time` timestamp(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
  PRIMARY KEY (`id`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci
ROW_FORMAT=DYNAMIC
```

7. 观察 Flink 中提交的 job 的消费情况。

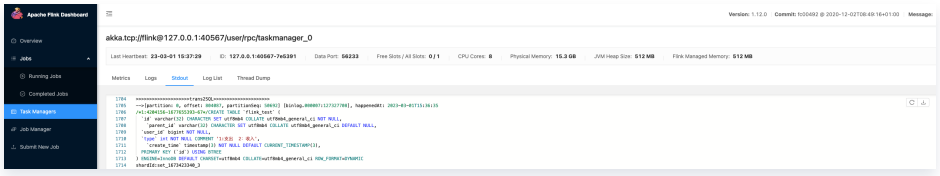




在 Task Managers 上查看具体的 task 的日志。



在 Task Managers 上查看具体的 Stdout 信息。



Demo 说明
Avro Demo 说明（Flink）

最近更新时间：2024-03-18 10:41:21

Demo 关键逻辑讲解

Demo 中的文件说明如下：

- `consumerDemo-avro-flink\src\main\resources\avro-tools-1.8.2.jar`：用来生成 Avro 协议相关代码的工具。
- `consumerDemo-avro-flink\src\main\java\com\tencent\subscribe\avro`：Avro 工具生成代码的目录。
- `consumerDemo-avro-flink\src\main\resources\Record.avsc`：协议定义文件。

Record.avsc 中我们定义了14个结构（Avro 中叫做 schema），其中主要的数据结构为 Record，用于表示 binlog 中的一条数据，Record 的结构如下，其他数据结构可以在 Record.avsc 中查看：

```
{
    "namespace": "com.tencent.subscribe.avro",     //Record.avsc 中的最后1个
schema，"name" 显示为 "Record"
    "type": "record",
    "name": "Record",      //"name" 显示为 "Record"，表示从 kafka 中消费的数据格式
    "fields": [
      {
        "name": "id",      //id 表示全局递增 ID，更多 record 取值解释如下表
        "type": "long",
        "doc": "unique id of this record in the whole stream"
      },
      {
        "name": "version",  //version 表示协议版本
        "type": "int",
        "doc": "protocol version"
      },
      {
        "name": "messageType",    //消息类型
        "aliases": [
          "operation"
        ],
        "type": {
          "namespace": "com.tencent.subscribe.avro",
          "name": "MessageType",
          "type": "enum",
          "symbols": [
            "INSERT",
            "UPDATE",
            "DELETE",
            "DDL",
            "BEGIN",
            "COMMIT",
            "HEARTBEAT",
            "CHECKPOINT",
```

```
            "ROLLBACK"
          ]
        }
      },
      {
        ……
      },
  }
```

Record 中的字段类型解释如下：

Record 中的字段名称	说明
id	全局递增 ID。
version	协议版本，当前版本为1。
messageType	消息类型，枚举 值："INSERT"，"UPDATE"，"DELETE"，"DDL"，"BEGIN"，"COMMIT"，"HEARTBEAT"，"CHECKPOINT"。
fileName	当前 record 所在的 binlog 文件名。
position	当前 record 的在 binlog 中结束的偏移量，格式为 End_log_pos@binlog 文件编号。例如，当前 record 位于文件 mysql-bin.000004 中，结束偏移量为2196，则其值为"2196@4"。
safePosition	当前事务在 binlog 中开始的偏移量，格式同上。
timestamp	写入 binlog 的时间，unix 时间戳，秒级。
gtid	当前的 gtid，如：c7c98333-6006-11ed-bfc9-b8cef6e1a231:9。
transactionId	事务 ID，只有 commit 事件才会生成事务 ID。
serverId	源库 serverId，查看源库 server_id 参考 SHOW VARIABLES LIKE 'server_id'。
threadId	提交当前事务的会话 ID，参考 SHOW processlist;。
sourceType	源库的数据库类型，当前版本只有 MySQL。
sourceVersion	源库版本，查看源库版本参考 `select version();`。
schemaName	库名。
tableName	表名。
objectName	格式为：库名.表名。
columns	表中各列的定义。
oldColumns	DML 执行前该行的数据，如果是 insert 消息，该数组为 null。数组中元素共有12种类型：Integer, Character, Decimal, Float, Timestamp, DateTime, TimestampWithTimeZone, BinaryGeometry, TextGeometry, BinaryObject, TextObject, EmptyObject，详见 demo 中定义。

newColumns	DML 执行后该行的数据，如果是 delete 消息，该数组为 null。数组中元素共有12种类型：Integer, Character, Decimal, Float, Timestamp, DateTime, TimestampWithTimeZone, BinaryGeometry, TextGeometry, BinaryObject, TextObject, EmptyObject，详见 demo 中定义。
sql	DDL 的 SQL 语句。
executionTime	DDL 执行时长，单位为秒。
heartbeatTimestamp	心跳消息的时间戳，秒级。只有 heartbeat 消息才有该字段。
syncedGtid	DTS 已解析 GTID 集合，格式形如：c7c98333-6006-11ed-bfc9-b8cef6e1a231:1-13。
fakeGtid	是否为构造的假 GTID，如未开启 gtid_mode，则 DTS 会构造一个 GTID。
pkNames	如果源库的表设有主键，则 DML 消息中会携带该参数，否则不会携带。
readerTimestamp	DTS 处理这条数据的时间，unix 时间戳，单位为毫秒数。
tags	QueryEvent 中的 status_vars，详细参考 QueryEvent 。
total	如果消息分片，记录分片总数。当前版本 (version=1) 无意义，预留扩展。
index	如果消息分片，记录当前分片的索引。当前版本 (version=1) 无意义，预留扩展。

Record 中描述列属性的字段为 "Field"，包含如下四个属性：

- name：列名。
- dataTypeNumber：是 binlog 中记录的数据类型。取值详见 MySQL 。
- isKey：是否主键。
- originalType：DDL 中定义的类型。

数据库字段映射关系

如下为数据库（如 MySQL）字段类型和 Avro 协议中定义的数据类型之间的映射关系。

MySQL 类型	对应 Avro 中的类型
MYSQL_TYPE_NULL	EmptyObject
MYSQL_TYPE_INT8	Integer
MYSQL_TYPE_INT16	Integer
MYSQL_TYPE_INT24	Integer
MYSQL_TYPE_INT32	Integer
MYSQL_TYPE_INT64	Integer
MYSQL_TYPE_BIT	Integer
MYSQL_TYPE_YEAR	DateTime

MYSQL_TYPE_FLOAT	Float
MYSQL_TYPE_DOUBLE	Float
MYSQL_TYPE_VARCHAR	Character
MYSQL_TYPE_STRING	Character，如果原类型为 binary，则对应 BinaryObject
MYSQL_TYPE_VAR_STRING	Character，如果原类型为 varbinary，则对应 BinaryObject
MYSQL_TYPE_TIMESTAMP	Timestamp
MYSQL_TYPE_DATE	DateTime
MYSQL_TYPE_TIME	DateTime
MYSQL_TYPE_DATETIME	DateTime
MYSQL_TYPE_TIMESTAMP_NEW	Timestamp
MYSQL_TYPE_DATE_NEW	DateTime
MYSQL_TYPE_TIME_NEW	DateTime
MYSQL_TYPE_DATETIME_NEW	DateTime
MYSQL_TYPE_ENUM	TextObject
MYSQL_TYPE_SET	TextObject
MYSQL_TYPE_DECIMAL	Decimal
MYSQL_TYPE_DECIMAL_NEW	Decimal
MYSQL_TYPE_JSON	TextObject
MYSQL_TYPE_BLOB	BinaryObject
MYSQL_TYPE_TINY_BLOB	BinaryObject
MYSQL_TYPE_MEDIUM_BLOB	BinaryObject
MYSQL_TYPE_LONG_BLOB	BinaryObject
MYSQL_TYPE_GEOMETRY	BinaryObject

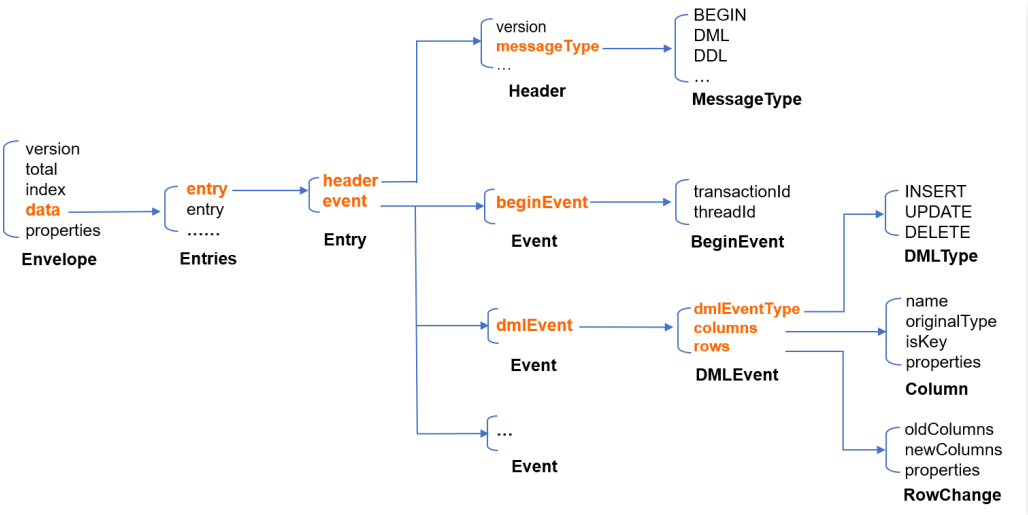
Protobuf Demo 说明（Flink）

最近更新时间：2024-08-27 10:03:11

Demo 关键逻辑讲解

消息生产逻辑

下文首先对消息生产逻辑进行简要说明，有助于用户理解消费逻辑。
我们采用 Protobuf 进行序列化，Demo 中均附带有 Protobuf 定义文件。文件中定义了几个关键结构：Envelope 是最终发送的 Kafka 消息结构；Entry 是单个订阅事件结构；Entries 是 Entry 的集合。主要数据结构关系如下所示：



生产过程如下：
1. 拉取 Binlog 消息，将每个 Binlog Event 编码为一个 Entry 结构体。

```
message Entry { //Entry 是单个订阅事件结构，一个事件相当于 MySQL 的一个 binlog event
 Header header = 1;  //事件头
 Event event   = 2;  //事件体
}


message Header {
 int32       version         = 1;   //Entry 协议版本
 SourceType  sourceType      = 2;   //源库的类型信息，包括 MySQL，Oracle 等类型
 MessageType messageType     = 3;   //消息的类型，也就是 Event 的类型，包括 BEGIN、
COMMIT、DML 等
 uint32 timestamp            = 4;   //Event 在原始 binlog 中的时间戳
 int64  serverId             = 5;   //源的 serverId
 string fileName             = 6;   //源 binlog 的文件名称
 uint64 position             = 7;   //事件在源 binlog 文件中的偏移量
 string gtid                 = 8;   //当前事务的 gtid
 string schemaName           = 9;   //变更影响的 schema
 string tableName            = 10;  //变更影响的 table
 uint64 seqId                = 11;  //全局递增序列号
 uint64 eventIndex           = 12;  //如果大的 event 分片，每个分片从0开始编号，当前版本无意
义，留待后续扩展用
```

```
 bool   isLast              = 13;  //当前 event 是否 event 分片的最后一块，是则为 true，
当前版本无意义，留待后续扩展用
 repeated KVPair properties = 15;
}


message Event {
 BeginEvent       beginEvent       = 1;  //binlog 中的 begin 事件
 DMLEvent         dmlEvent         = 2;  //binlog 中的 dml 事件
 CommitEvent      commitEvent      = 3;  //binlog 中的 commit 事件
 DDLEvent         ddlEvent         = 4;  //binlog 中的 ddl 事件
 RollbackEvent    rollbackEvent    = 5;  //rollback 事件，当前版本无意义
 HeartbeatEvent   heartbeatEvent   = 6;  //源库定时发送的心跳事件
 CheckpointEvent checkpointEvent = 7;  //订阅后台添加的 checkpoint 事件，每10秒自动生成一
个，用于 Kafka 生产和消费位点管理
 repeated KVPair properties       = 15;
}
```

2. 为减少消息量，将多个 Entry 合并，合并后的结构为 Entries，Entries.items 字段即为 Entry 顺序列表。合并的数量以合并后不超过 Kafka 单个消息大小限制为标准。对单个 Event 就已超过大小限制的，则不再合并，Entries 中只有唯一 Entry 。

```
message Entries {
     repeated Entry items = 1; //entry list
}
```

3. 对 Entries 进行 Protobuf 编码得到二进制序列。
4. 将 Entries 的二进制序列放入 Envelope 的 data 字段。当存在单个 Binlog Event 过大时，二进制序列可能超过 Kafka 单个消息大小限制，此时我们会将其分割为多段，每段装入一个 Envelope。

```
message Envelope {
     int32  version                 = 1; //protocol version, 决定了 data 内容如何解码
     uint32 total                   = 2;
     uint32 index                   = 3;
     bytes  data                    = 4; //当前 version 为1, 表示 data 中数据为
Entries 被 PB 序列化之后的结果
     repeated KVPair properties     = 15;
}
```

5. 对上一步生成的一个或多个 Envelope 依次进行 Protobuf 编码，然后投递到 Kafka 分区。同一个 Entries 分割后的多个 Envelope 顺序投递到同一个分区。

消息消费逻辑

下文对消费逻辑进行简要说明。

1. Flink 消费端需要创建一个 FlinkKafkaConsumer，需要指定消费主题，以及自定义一个基于 Protobuf 协议的消息反序列化器。

```
// 创建一个 Flink-kafka 消费者
```

```
FlinkKafkaConsumer<RecordMsgObject> consumer =
                        new FlinkKafkaConsumer<>(topic, new
DeserializeProtobufToRecordMsgObject(), props);
```

2. DeserializeProtobufToRecordMsgObject 将原始消息反序列化为 RecordMsgObject 对象。

```
// 自定义反序列化器，将原始消息反序列化为 RecordMsgObject 对象
@Override
public RecordMsgObject deserialize(ConsumerRecord<byte[], byte[]> record) throws
Exception {
    RecordMsgObject obj = new RecordMsgObject();
    obj.topic = record.topic();
    obj.partition = record.partition();
    obj.offset = record.offset();
    obj.partitionSeq = getPartitionSeq(record);
    obj.key = new String(record.key());
    obj.headers = record.headers();
    // 这里收到的就是 Envelope 的二进制的值
    obj.value = record.value();
    return obj;
}
```

3. 将收到的消息按照消息的 partition 分组处理，分组处理逻辑由 SubscribeMsgProcess 实现。

```
//将收到的消息按照 parttition 分区处理
stream.keyBy(RecordMsgObject::getPartition)
      .process(new SubscribeMsgProcess(trans2sql)).setParallelism(1);
```

4. SubscribeMsgProcess 中将收到的二进制序列用 Protobuf 解码为 Envelope 。

```
// 反序列出 envlope，本 demo 默认一个 envlope 就可以存储完整个 bnlog-event 的数据
SubscribeProtobufData.Envelope envelope =
SubscribeProtobufData.Envelope.parseFrom(record.value);
if (1 != envelope.getVersion()) {
    throw new IllegalStateException(String.format("unsupported version: %d",
envelope.getVersion()));
}
```

> ⚠ **注意：**
> 本 Demo 演示时默认单个 Binlog Event 不会超过 Kafka 单个消息大小限制，如果超过单个消息大小限制，Demo 必须在消费时引入 Flink 的高级特性“状态”去拼接 Envelope 来得到完整消息体，此种场景需要用户参考 Flink 官网，并根据自己的业务场景进行处理。

5. 将收到的 Envelope 的 data 字段的二进制序列用 Protobuf 解码为 Entries 。

```
// 反序列得到 Entries
ByteString envelopeData = envelope.getData();
SubscribeProtobufData.Entries entries;
if (1 == envelope.getTotal()) {
    entries = SubscribeProtobufData.Entries.parseFrom(envelopeData.toByteArray());
} else {
    entries =
SubscribeProtobufData.Entries.parseFrom(shardMsgMap.get(shardId).toByteArray());
    shardMsgMap.remove(shardId);
}
```

6. 对 Entries.items 依次处理，打印原始 Entry 结构或者转化为 SQL 语句。

```
// 遍历每个 Entry,根据 Entry 类型去打印 sql
for (SubscribeProtobufData.Entry entry : entries.getItemsList()) {
    onEntry(record.partition, record.offset, ps, entry, trans2sql);
}
```

Table API & Flink SQL

本 Demo 只展示了客户端类型为 DataStream API 的模式，Flink 的客户端模式为 Table API & Flink SQL的场景需要用户自行处理。使用 Table API & Flink SQL 的客户端模式有两种方式：

1. 使用 DataStream 转化成 Table 的客户端形式，具体可以参考：DataStream API Integration 。
2. 基于 Table API & Flink SQL 自定义一个 connector，具体可以参考：User-defined Sources & Sinks 。

数据库字段映射和存储

本节介绍数据库字段类型和序列化协议中定义的数据类型之间的映射关系。
源数据库字段值在 Protobuf 协议中用如下所示的 Data 结构来存储。

```
message Data {
    DataType     dataType = 1;
    string       charset  = 2;  //DataType_STRING 的编码类型，值存储在 bv 里面
    string       sv       = 3;
//DataType_INT8/16/32/64/UINT8/16/32/64/Float32/64/DataType_DECIMAL 的字符串值
    bytes        bv       = 4;  //DataType_STRING/DataType_BYTES 的值
}
```

其中 DataType 字段代表存储的字段类型，可取枚举值如下图所示。

```
enum DataType {
    NIL     = 0; //值为 NULL
    INT8    = 1;
    INT16   = 2;
    INT32   = 3;
    INT64   = 4;
    UINT8   = 5;
    UINT16  = 6;
```

```
    UINT32  = 7;
    UINT64  = 8;
    FLOAT32 = 9;
    FLOAT64 = 10;
    BYTES   = 11;
    DECIMAL = 12;
    STRING  = 13;
    NA      = 14; //值不存在 (N/A)
}
```

其中 bv 字段存储 STRING 和 BYTES 类型的二进制表示，sv 字段存储 INT8/16/32/64/UINT8/16/32/64/DECIMAL 类型的字符串表示，charset 字段存储 STRING 的编码类型。
TDSQL MySQL 原始类型与 DataType 映射关系如下（对 UNSIGNED 修饰的 MYSQL_TYPE_INT8/16/24/32/64 分别映射为 UINT8/16/32/32/64）：

说明：

1. 关于时区的相关说明。
   - `DATE`，`TIME`，`DATETIME` 类型不支持时区。
   - `TIMESTAMP` 类型支持时区，该类型字段表示：存储时，系统会从当前时区转换为 UTC（Universal Time Coordinated）进行存储；查询时，系统会从 UTC 转换为当前时区进行查询。
   - 综上，如下表中 "MYSQL_TYPE_TIMESTAMP" 和 "MYSQL_TYPE_TIMESTAMP_NEW" 字段会携带时区信息，用户在消费数据时可自行转换。（例如，DTS 输出的时间格式是带时区的字符串"2021-05-17 07:22:42 +00:00"，其中，"+00:00"表示 UTC 时间，用户在解析和转换的时候需要考虑时区信息。）
2. 对于源端 MySQL 的数值类型，DTS 写入 Kafka 时会转为字符串类型，其中 FLOAT 和 DOUBLE 类型在转为字符串时如果绝对值小于 1e-4 或大于等于 1e+6，那么结果将使用科学计数法表示。科学计数法以尾数乘以10的幂的形式表示浮点数，这样可以更清晰地表示其数量级和精确值，同时能够节省存储空间。如果在消费端用户需要转换为数值类型，请注意对科学记数法表示的浮点数字符串做特殊处理。

TDSQL MySQL 字段类型	对应的 Protobuf DataType 枚举值
MYSQL_TYPE_NULL	NIL
MYSQL_TYPE_INT8	INT8
MYSQL_TYPE_INT16	INT16
MYSQL_TYPE_INT24	INT32
MYSQL_TYPE_INT32	INT32
MYSQL_TYPE_INT64	INT64
MYSQL_TYPE_BIT	INT64
MYSQL_TYPE_YEAR	INT64
MYSQL_TYPE_FLOAT	FLOAT32
MYSQL_TYPE_DOUBLE	FLOAT64

MYSQL_TYPE_VARCHAR	STRING
MYSQL_TYPE_STRING	STRING
MYSQL_TYPE_VAR_STRING	STRING
MYSQL_TYPE_TIMESTAMP	STRING
MYSQL_TYPE_DATE	STRING
MYSQL_TYPE_TIME	STRING
MYSQL_TYPE_DATETIME	STRING
MYSQL_TYPE_TIMESTAMP_NEW	STRING
MYSQL_TYPE_DATE_NEW	STRING
MYSQL_TYPE_TIME_NEW	STRNG
MYSQL_TYPE_DATETIME_NEW	STRING
MYSQL_TYPE_ENUM	STRING
MYSQL_TYPE_SET	STRING
MYSQL_TYPE_DECIMAL	DECIMAL
MYSQL_TYPE_DECIMAL_NEW	DECIMAL
MYSQL_TYPE_JSON	BYTES
MYSQL_TYPE_BLOB	BYTES
MYSQL_TYPE_TINY_BLOB	BYTES
MYSQL_TYPE_MEDIUM_BLOB	BYTES
MYSQL_TYPE_LONG_BLOB	BYTES
MYSQL_TYPE_GEOMETRY	BYTES

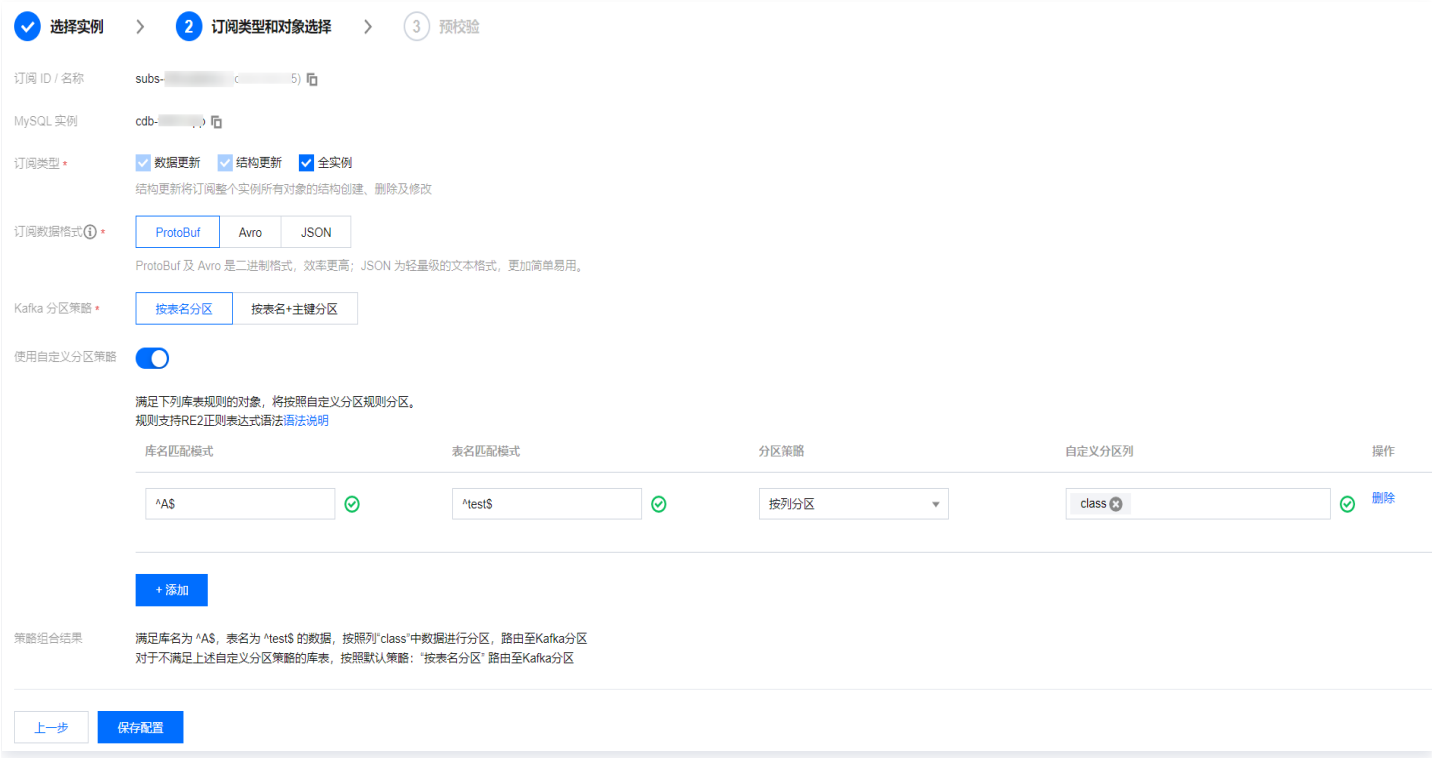
订阅高级操作
设置分区策略

最近更新时间：2023-11-17 17:14:45

操作场景

当用户选择 Kafka 多分区时，可以通过设置分区策略，将业务相互关联的数据路由到同一个分区中，这样方便用户处理消费数据。DTS 支持按表名、按表名+主键、按列名进行分区，将订阅数据经过哈希规则路由到 Kafka 各分区。

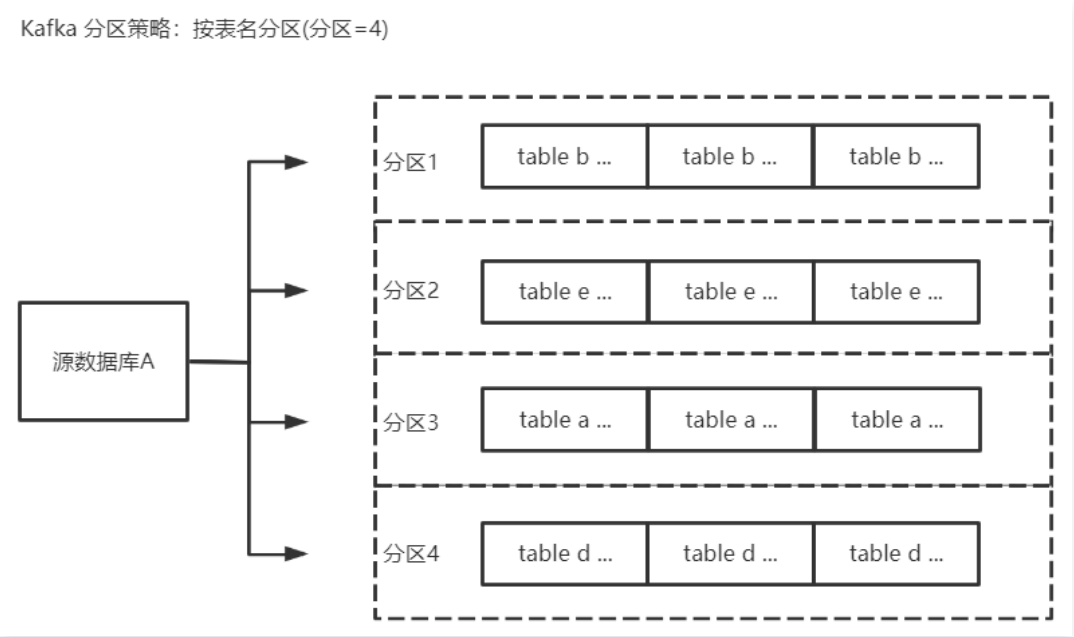
- Kafka 分区策略 – 按表名分区：将源库的订阅数据按照表名进行分区，设置后相同表名的数据会写入同一个 Kafka 分区中。在数据消费时，同一个表内的数据变更总是顺序获得。
- Kafka 分区策略 – 按表名+主键分区：将源库的订阅数据按照表名+主键进行分区，适用于热点数据，设置后热点数据的表，通过表名+主键的方式将数据分散到不同分区中，提升并发消费效率。
- 自定义分区策略：先通过正则表达式对订阅数据中的库名和表名进行匹配，匹配后的数据再按照表名、表名+主键、列进行分区。



Kafka 分区策略

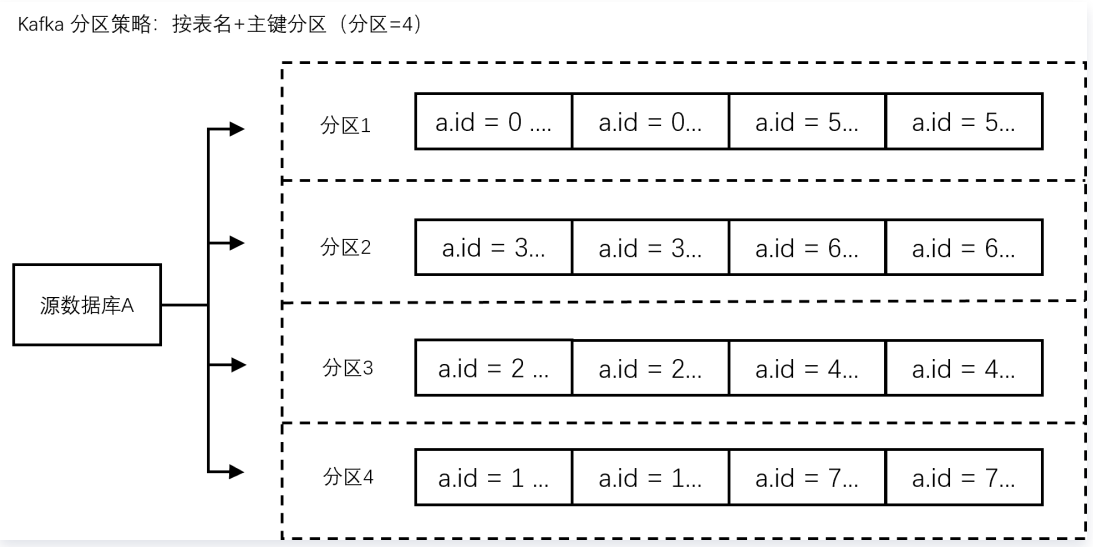
按表名分区

当源数据库变更时，将变更的数据写入订阅 Topic 中，选择**按表名分区**后，相同表名的订阅数据会写入同一个 Kafka 分区中。在消费数据时，可以保证同一个表内的数据变更总是顺序获得。



按表名+主键分区

当仅指定按照表名分区时，如果一张表为热点数据，则按照表名分区，该 Kafka 分区的压力会非常大。通过表名+主键分区将数据散列到不同分区中，提升并发消费效率。



自定义分区策略

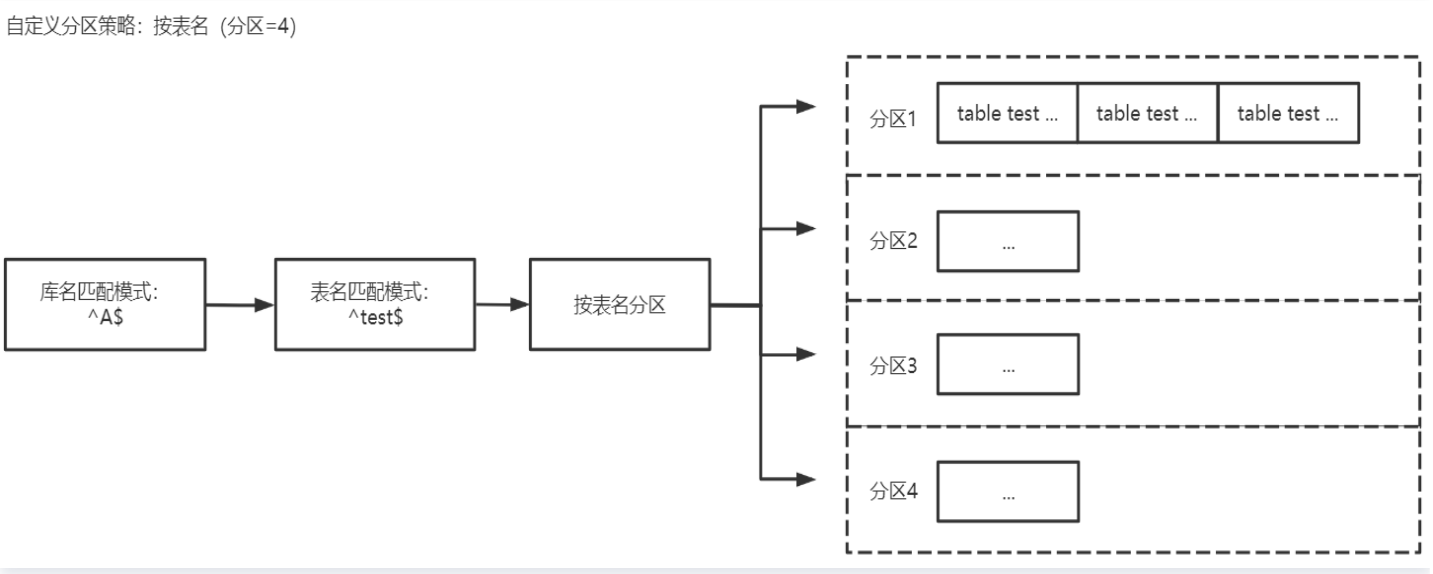
自定义分区策略，是通过正则表达式对库名和表名进行匹配，匹配后的数据再按照表名、表名+主键、列进行分区。

匹配规则

- **库表、表名的匹配规则支持 RE2 正则表达式，具体语法请参考 语法说明 。**
- **库名的匹配规则，按照正则表达式来匹配库名。表名的匹配规则，按照正则表达式来匹配表名的数据。如果库名和表名需要精确匹配，需要加开始和结束符，如 `test` 表应该为 `^test$` 。**
- **列名的匹配规则，按照等值 `==` 来匹配，大小写不敏感。**
- **优先匹配自定义分区策略，当自定义分区策略未匹配到时，则应用 Kafka 分区策略的设置进行匹配。当自定义分区策略有多条时，自上向下逐条匹配。**

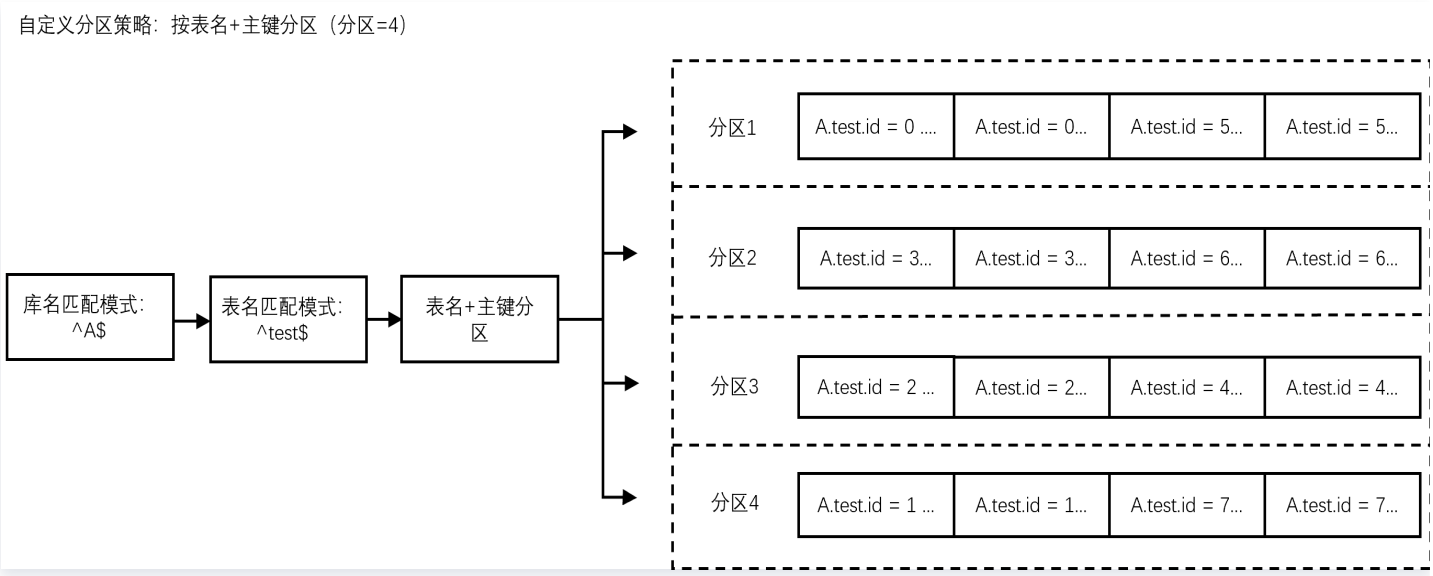
按表名分区

库名匹配模式填入 `^A$` **，表名匹配模式填入** `^test$` **，选择按表名分区后，** `A` **库中** `test` **的订阅数据会写入到同一个分区中，** `test` **除外其他未匹配到的库表数据会根据 Kafka 分区策略中设置的策略进行写入。**



按表名+主键分区

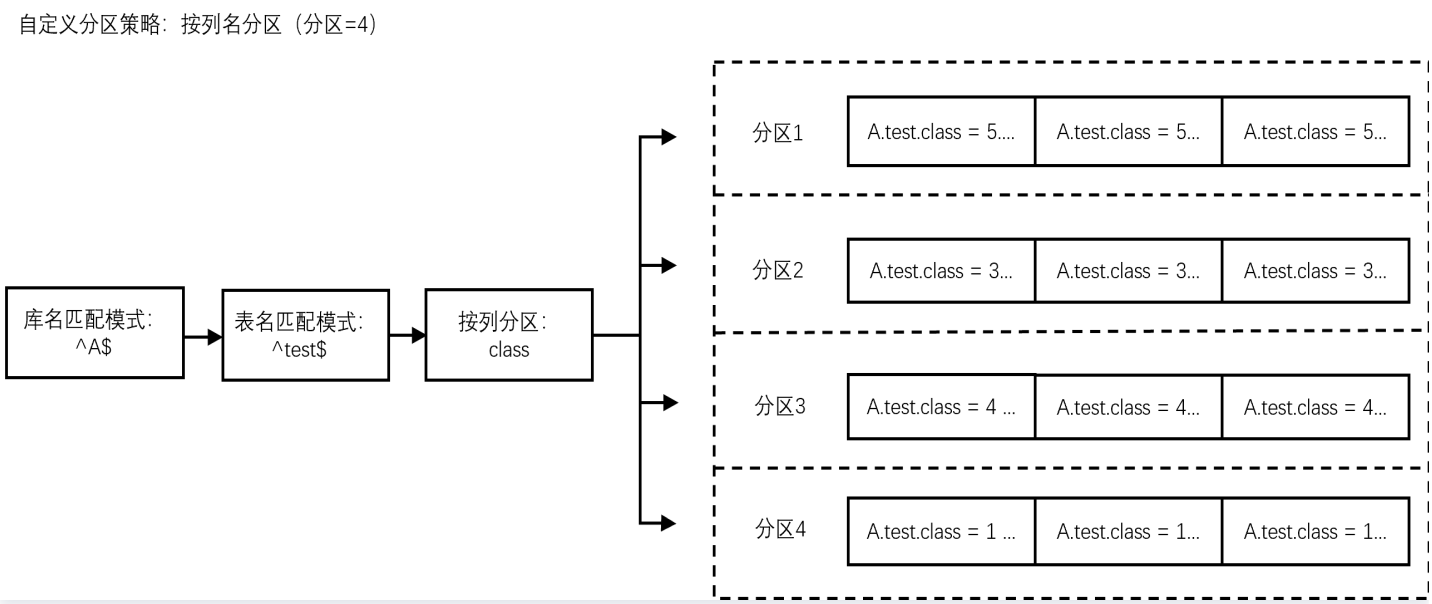
库名匹配模式填入 `^A$` **，表名匹配模式填入** `^test$` **，选择按表名+主键分区后，** `A` **库中** `test` **的订阅数据会根据主键数据的不同，散列写入到不同的分区中，最终主键数据相同的数据都写入了同一个分区。** `test` **除外其他未匹配到的库表数据会根据 Kafka 分区策略中设置的策略进行写入。**



按列分区

库名匹配模式填入 `^A$` **，表名匹配模式填入** `^test$` **，自定义分区列填入** `class` **，选择按列名分区后，** `A` **库中表** `test` **列名为** `class` **的订阅数据将被散列写入到不同分区中，最终同一列的数据更新都写入到了一个分区中。** `test` **除外其他未匹配到的库表数**

据会根据 Kafka 分区策略中设置的策略进行写入。



前置校验不通过处理
连接 DB 检查

最近更新时间：2024-06-12 14:35:41

检查详情

源数据库和目标数据库需要能正常连通，如果未连通，会报连接失败。

问题原因

- 源数据库所在网络或服务器设置了安全组或防火墙 。
- 源数据库对来源 IP 地址进行了限制 。
- 网络端口未放通 。
- 数据库账号或密码不正确。

修复方法

请按照问题原因中的对应链接进行处理。

周边检查

最近更新时间：2024-05-21 11:41:05

检查详情

建议源数据库环境变量参数 `innodb_stats_on_metadata` 设置为 `OFF` 。

修复方法

修改 innodb_stats_on_metadata 参数

- `innodb_stats_on_metadata` 参数开启时，每当查询 `information_schema` 元数据库里的表，Innodb 就会更新 `information_schema.statistics` 表，导致访问时间变长。关闭后可加快对于 schema 库表的访问。
- MySQL 5.6.6 之前版本 `innodb_stats_on_metadata` 参数预设值为 ON，需要修改为 OFF。MySQL 5.6.6 及其以后的版本预设值为 OFF，不存在问题。

1. 登录源数据库。
2. 修改 `innodb_stats_on_metadata` 为 `OFF` 。

```
set global innodb_stats_on_metadata = OFF;
```

3. 查看配置是否生效。

```
show global variables like '%innodb_stats_on_metadata%';
```

系统显示结果类似如下：

```
mysql> show global variables like '%innodb_stats_on_metadata%';
+---------------------------+-------+
| Variable_name             | Value |
+---------------------------+-------+
| innodb_stats_on_metadata | OFF   |
+---------------------------+-------+
1 row in set (0.00 sec)
```

4. 重新执行校验任务。

版本检查

最近更新时间：2024-05-21 11:41:05

检查详情

源库版本符合 数据订阅支持的数据库 中的版本要求。

修复方法

如果源库版本不支持，请升级版本或者使用更高版本的数据库实例。

源实例权限检查

最近更新时间：2024-05-21 11:41:05

检查详情

检查用户是否具备对数据库的操作权限，迁移/同步/订阅中，每个数据库类型的操作指导有详细的权限要求，请参考对应内容：

- 数据迁移权限要求
- 数据同步权限要求
- 数据订阅权限要求

修复方法

用户若不具备操作权限，请按照检查要求中的对应权限要求对用户进行授权，然后重新执行校验任务。

Binlog 参数检查

最近更新时间：2024-09-02 14:30:32

检查详情

源数据库 binlog 相关参数需要按照如下要求配置，如果校验不通过，请参考本文后续指导进行修复。

- `log_bin` 变量必须设置为 `ON` 。
- `binlog_format` 变量必须设置为 `ROW` 。
- 如果源数据库为 MySQL 5.6 及以上版本，`gtid_mode` 只支持设置为 `ON` 和 `OFF` ，建议将 `gtid_mode` 设置为 `ON` ，设置为 `OFF` 会报警告，设置为 `ON_PERMISSIVE` 和 `OFF_PERMISSIVE` 会报错。
- `server_id` 参数需要手动设置，且值不能设置为0。
- 不允许设置 `do_db` ， `ignore_db` 。
- 对于源实例为从库的情况， `log_slave_updates` 变量必须设置为 `ON` 。
- 建议源库 Binlog 日志至少保留3天及以上，否则可能会因任务暂停/中断时间大于 Binlog 日志保留时间，造成任务无法续传，进而导致任务失败。

修复方法

开启 binlog

`log_bin` 是 binlog 的开关控制参数，需要将 binlog 打开，以便记录所有的数据库表结构和表数据变更日志。
如发生类似报错，请参考如下指导进行修复：

1. 登录源数据库。
2. 参考如下内容修改源数据库的配置文件 `my.cnf` 。

   因 log_bin 参数修改后需要重启数据库后才能生效，如果 binlog_format、server_id、binlog_row_image、expire_logs_days 这几个参数也在校验阶段提示报错，请一并修改完成后再重启数据库，让所有参数都生效。

   > **说明：**
   > `my.cnf` 配置文件的默认路径为 `/etc/my.cnf` ，现场以实际情况为准。

   ```
   log_bin = MYSQL_BIN
   binlog_format = ROW
   server_id = 2      //建议设为大于1的整数，此处仅为示例
   binlog_row_image = FULL
   expire_logs_days = 3      //修改 binlog 的保留时间，建议大于等于3天
   ```

3. 参考如下命令重启源数据库。

   ```
   [$Mysql_Dir]/bin/mysqladmin -u root -p shutdown
   [$Mysql_Dir]/bin/safe_mysqld &
   ```

   > **说明：**
   > [$Mysql_Dir] 指源数据库的安装路径，请替换为实际的源数据库安装目录。

4. 确认 binlog 功能是否已启用。

```
show variables like '%log_bin%';
```

系统显示结果类似如下：

```
mysql> show variables like '%log_bin%';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| log_bin          | ON    |
+------------------+-------+
1 row in set (0.00 sec)
```

请使用如下对应的命令查看其他参数的修改结果。
show variables like '%binlog_format%';
show variables like '%binlog_row_image%';
show global variables like '%server_id%';

5. 重新执行校验任务。

修改 binlog_format 参数

`binlog_format` 为 binlog 的记录模式，有以下三种：

- `STATEMENT`：每一条会修改数据的 SQL 都会记录到 master 的 binlog 中，slave 在复制的时候，会执行和原来 master 端相同的 SQL。该模式可以减少 binlog 日志量，但是对某些特定的函数进行复制时，slave 端不能正确复制。
- `ROW`：binlog 日志中会记录成每一行数据修改的形式，然后在 slave 端再对相同的数据进行修改。该模式会保证 master 和 slave 的正确复制，但是 binlog 日志量会增加。
- `MIXED`：前两种模式的结合，MySQL 会根据执行的每一条具体的 SQL 语句来区分对待记录的日志形式，在 `STATEMENT` 和 `ROW` 之间选择一种。

综上，为了保证 master 和 slave 的正确复制，`binlog_format` 参数需要设置为 `ROW`。如发生类似报错，请参考如下指导进行修复。

> **说明：**
> 该参数修改需要重置数据库上的所有连接才能生效，当源库为从库时，还需重启主从同步 SQL 线程，避免当前业务连接继续使用修改前的模式写入。

1. 登录源数据库。
2. 参考如下命令修改 `binlog_format`。

```
set global binlog_format = ROW;
```

3. 重启线程使配置生效，然后通过如下命令查看参数修改是否生效。

```
show variables like '%binlog_format%';
```

系统显示结果类似如下：

```
mysql> show variables like '%binlog_format%';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| binlog_format | ROW   |
+---------------+-------+
1 row in set (0.00 sec)
```

4. 重新执行校验任务。

修改 gtid_mode 参数

GTID（Global Transaction Identifier，全局事务标识），用于在 binlog 中唯一标识一个事务，使用 GTID 可以避免事务重复执行导致数据混乱或者主从不一致。
GTID 是 MySQL 5.6 的新特性，所以 MySQL 5.6 及之后版本存在此问题。DTS 只支持 `gtid_mode` 设置为 `ON` 和 `OFF`，建议将 `gtid_mode` 设置为 `ON`，否则校验时会报警告。
警告不影响迁移或同步任务进行，但是会对业务造成一定的影响：设置 GTID 后，在增量数据同步阶段，如果源实例发生 HA 切换，DTS 服务切换重连，任务几乎无感知；反之，任务会中断后失败，且不可恢复。
`gtid_mode` 的取值如下，在修改 `gtid_mode` 的值时，只能从以下四个值逐级修改，例如，需要从 `OFF` 修改为 `ON`，则 `gtid_mode` 修改顺序为 `OFF` <-> `OFF_PERMISSIVE` <-> `ON_PERMISSIVE` <-> `ON`。

- `OFF`：主库所有新启的事务以及从库的事务都要求是匿名事务。
- `OFF_PERMISSIVE`：主库新启的事务是匿名事务，但从库事务允许是匿名的或者是 GTID 事务，但不允许只是 GTID 模式。
- `ON_PERMISSIVE`：主库新启的事务是 GTID 事务，从库事务允许是匿名的或者是 GTID 事务。
- `ON`：主库新启的事务是 GTID 事务，从库的事务也要求是 GTID 事务。

如果发生类似警告，请按照如下指导进行修复：

1. 登录源数据库。
2. 在主从复制结构两边都设置 `gtid_mode = OFF_PERMISSIVE`。
   MySQL 5.7.6 之前的版本需要在 `my.cnf` 配置文件中修改并重启数据库才能生效，5.7.6 及之后的版本通过全局命名修改，不需要重启数据库，但是需要重置所有业务连接。

```
set global gtid_mode = OFF_PERMISSIVE;
```

3. 在主从复制结构两边都设置 `gtid_mode = ON_PERMISSIVE`。

```
set global gtid_mode = ON_PERMISSIVE;
```

4. 在各个实例节点上执行如下命令，检查匿名事务是否消耗完毕，参数值为 `0` 则代表消耗完毕。

```
show variables like '%ONGOING_ANONYMOUS_TRANSACTION_COUNT%';
```

系统显示结果类似如下：

```
mysql> show variables like '%ONGOING_ANONYMOUS_TRANSACTION_COUNT%';
+-----------------------------------+-------+
| Variable_name                     | Value |
```

```
+-------------------------------------+-------+
| Ongoing_anonymous_transaction_count | 0     |
+-------------------------------------+-------+
1 row in set (0.00 sec)
```

5. 在主从复制结构两边都设置 `gtid_mode = ON` 。

```
set global gtid_mode = ON;
```

6. 在 `my.cnf` 文件中添加如下内容，后续重启数据库后使初始值生效。

> **说明：**
> `my.cnf` 配置文件的默认路径为 `/etc/my.cnf` ，现场以实际情况为准。

```
gtid_mode = on
enforce_gtid_consistency = on
```

7. 重新执行校验任务。

修改 server_id 参数

`server_id` 参数需要手动设置，且值不能设置为0。该参数系统预设值为 `1` ，如果查询该参数显示为 `1` 不一定正确，需要手动进行配置。

1. 登录源数据库。
2. 参考如下内容修改 `server_id` 。

```
set global server_id = 2;  //建议设为大于1的整数，此处仅为示例
```

3. 通过如下命令查看参数修改是否生效。

```
show global variables like '%server_id%';
```

系统显示结果类似如下：

```
mysql> show global variables like '%server_id%';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 2     |
+---------------+-------+
1 row in set (0.00 sec)
```

4. 重新执行校验任务。

删除 do_db，ignore_db 设置

binlog 会记录数据库所有执行的 DDL 和 DML 语句，而 do_db，ignore_db 则是设置 binlog 记录的过滤条件。

- `binlog_do_db`：只记录指定数据库的二进制日志，默认全部记录。
- `binlog_ignore_db`：不记录指定的数据库的二进制日志。

设置 do_db，ignore_db 后，会导致一些跨库操作 binlog 记录不全，主从复制出现异常，因此不建议设置。如发生类似报错，请参考如下指导进行修复：

1. 登录源数据库。
2. 修改源数据库的配置文件 `my.cnf`，删除 do_db，ignore_db相关设置。

> **说明：**
> `my.cnf` 配置文件的默认路径为 `/etc/my.cnf`，现场以实际情况为准。

3. 参考如下命令重启源数据库。

```
[$Mysql_Dir]/bin/mysqladmin -u root -p shutdown
[$Mysql_Dir]/bin/safe_mysqld &
```

> **说明：**
> [$Mysql_Dir] 指源数据库的安装路径，请替换为实际的源数据库安装目录。

4. 确认参数修改是否生效。

```
show master status;
```

系统显示结果类似如下：

```
mysql> show master status;
+---------------+----------+--------------+------------------+-------------------+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+---------------+----------+--------------+------------------+-------------------+
| binlog.000011 | 154      |              |                  |                   |
+---------------+----------+--------------+------------------+-------------------+
```

5. 重新执行校验任务。

修改 log_slave_updates 参数

在主从复用结构中，从库开启 `log-bin` 参数，直接在从库操作数据时，可以记录在 binlog 中，但是从库从主库上复制数据时，不能记录在 binlog 中，所以从库作为其他从库的主库时，需要打开 `log_slave_updates` 参数。

1. 登录源数据库。
2. 在源数据库的配置文件 `my.cnf` 中增加如下内容。

> **说明：**
> `my.cnf` 配置文件的默认路径为 `/etc/my.cnf`，现场以实际情况为准。

```
log_slave_updates = ON
```

3. 参考如下命令重启源数据库。

```
[$Mysql_Dir]/bin/mysqladmin -u root -p shutdown
[$Mysql_Dir]/bin/safe_mysqld &
```

> **说明：**
> [$Mysql_Dir] 指源数据库的安装路径，请替换为实际的源数据库安装目录。

4. 查看配置是否生效。

```
show variables like '%log_slave_updates%';
```

系统显示结果类似如下：

```
mysql> show variables like '%log_slave_updates%';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| log_slave_updates | ON    |
+-------------------+-------+
1 row in set (0.00 sec)
```

5. 重新执行校验任务。

TDSQL PostgreSQL 数据订阅
创建 TDSQL PostgreSQL 版数据订阅

最近更新时间：2024-10-16 16:11:21

本场景介绍使用 DTS 创建腾讯云 TDSQL PostgreSQL 版 的数据订阅任务操作指导。

前提条件

- 已准备好待订阅的 TDSQL PostgreSQL 版，并且数据库版本符合要求，请参见 数据订阅支持的数据库 。
- 已在源端实例中创建订阅账号，需要账号权限如下： LOGIN 和 REPLICATION 权限。
  LOGIN 和 REPLICATION 授权请 提交工单 处理。
- 订阅账号必须拥有被订阅表的 select 权限，如果是整库订阅，那么订阅账号要拥有该 schema 下所有表的 select 权限，具体授权语句如下：

```
grant SELECT on all tables in schema "schema_name" to '迁移账号' ;
```

- 用户必须拥有 pg_catalog.pgxc_node 表的 select 权限，具体授权语句如下：

```
grant SELECT on pg_catalog.pgxc_node to '迁移账号';
```

- DN 节点的 wal_level 必须是 logical。
- 被订阅的表如果是全复制表（建表语句中有 distribute by replication 关键字），必须拥有主键；被订阅的表如果不是全复制表，必须拥有主键或 REPLICA IDENTITY 为 FULL；修改表的 REPLICA IDENTITY 为 FULL 的语句：

```
alter table '表名' REPLICA IDENTITY FULL;
```

约束限制

- 订阅的消息保存在 DTS 内置 Kafka（单 Topic）中，目前默认保存时间为最近1天，单 Topic 的最大存储为500G，当数据存储时间超过1天，或者数据量超过500G时，内置 Kafka 都会开始清除最先写入的数据。所以请用户及时消费，避免数据在消费完之前就被清除。
- 数据消费的地域需要与订阅任务所属的地域相同。
- 当前不支持 gtsvector, pg_dependencies, pg_node_tree, pg_ndistinct, xml 相关的数据类型。
- 数据订阅源是 TDSQL PostgreSQL 版时，不支持直接执行授权语句授权，所以订阅账号的权限需要在 TDSQL 控制台 单击实例 ID，获取实例登录信息后，通过客户端登录数据库进行账号授权。
- 订阅任务过程中，如果进行修改订阅对象等操作会发生任务重启，重启后可能会导致用户在 kafka 客户端消费数据时出现重复。
  - DTS 是按最小数据单元进行传输的，增量数据每标记一个 checkpoint 位点就是一个数据单元，如果重启时，刚好一个数据单元传输已完成，则不会导致数据重复；如果重启时，一个数据单元还正在传输中，那么再次启动后需要重新拉取这个数据单元，以保证数据完整性，这样就会导致数据重复。
  - 用户如果对重复数据比较关注，请自行在消费数据时设置去重逻辑。

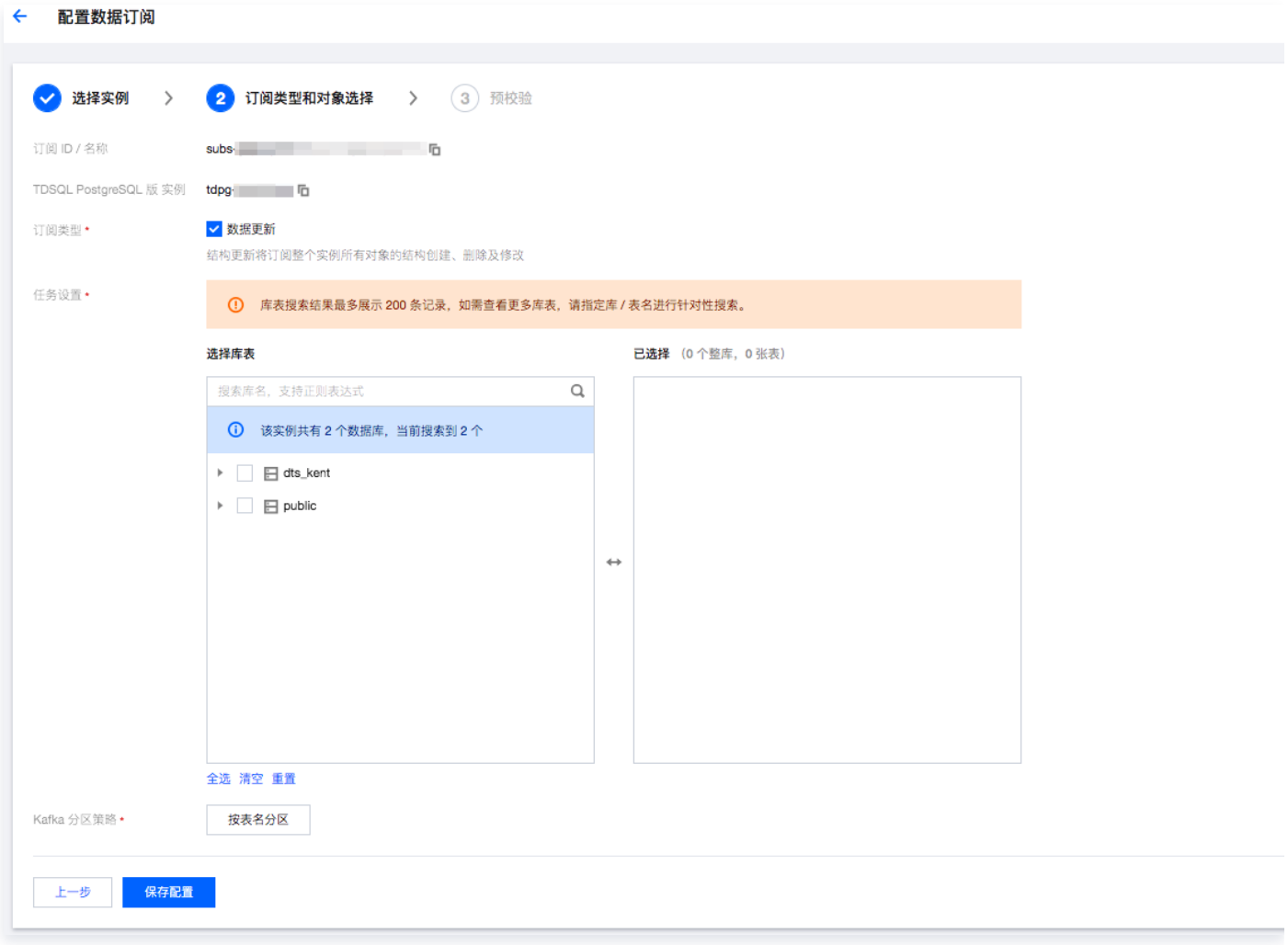
支持订阅的 SQL 操作

操作类型	支持的 SQL 操作
DML	INSERT、UPDATE、DELETE

操作步骤

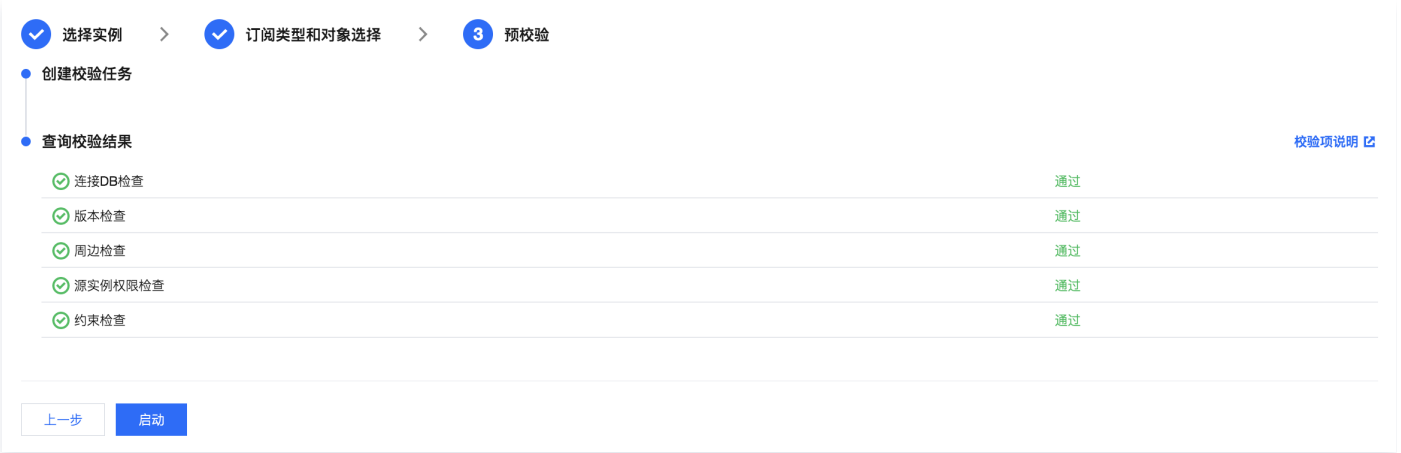
1. 登录 DTS 控制台，在左侧导航选择**数据订阅**，单击**新建数据订阅**。
2. 在新建数据订阅页，选择相应配置，单击**立即购买**。
   - 计费模式：支持包年包月和按量计费。
   - 地域：地域需与待订阅的数据库实例订阅保持一致。
   - 数据库：请根据具体数据库类型进行选择。
   - 版本：选择 **kafka 版**，支持通过 Kafka 客户端直接消费。
   - 订阅实例名称：编辑当前数据订阅实例的名称。
3. 购买成功后，返回数据订阅列表，单击**操作**列的**配置订阅**对刚购买的订阅进行配置，配置完成后才可以进行使用。
4. 在配置数据库订阅页面，选择相应配置，单击**下一步**。
   - 实例：选择对应数据库实例，只读实例和灾备实例不支持数据订阅。
   - 数据库账号：添加订阅实例的账号和密码，账号的 LOGIN 、REPLICATION 权限和全部对象、pg_catalog.pgxc_node 表的 SELECT 权限。
5. 在订阅类型和对象选择页面，选择订阅类型，单击**保存配置**。
   - 订阅类型为数据更新（订阅选择对象的数据更新，包括数据 INSERT、UPDATE、DELETE 操作）。

- Kafka 分区策略：支持按表名分区。



6. 在预校验页面，预校验任务预计会运行2分钟 – 3分钟，预校验通过后，单击**启动**完成数据订阅任务配置。

> **说明：**
> 如果校验失败，请根据失败提示在待订阅实例中进行修正，并重新进行校验。



7. 单击启动后，订阅任务会进行初始化，预计会运行3分钟 – 4分钟，初始化成功后进入**运行中**状态。
8. 新增消费组，数据订阅 Kafka 版支持用户创建多个消费组（单个订阅任务最多支持创建10个消费组），进行多点消费。数据订阅 Kafka 版消费依赖于 Kafka 的消费组，所以在消费数据前需要创建消费组。

9. 订阅实例进入运行中状态之后，就可以开始消费数据。Kafka 的消费需要进行密码认证，具体示例请参考 数据消费 Demo，我们提供了多种语言的 Demo 代码，也对消费的主要流程和关键的数据结构进行了说明。

消费 TDSQL PostgreSQL 订阅数据

最近更新时间：2024-10-15 10:27:52

操作场景

数据订阅 Kafka 版（当前 Kafka Server 版本为V2.6.0）中，您可以通过0.11版本及以上的 Kafka 客户端 进行消费订阅数据，本文为您提供了消费 Demo 示例，方便您快速测试消费数据的流程，了解数据格式解析的方法。

注意事项

- **Demo 并不包含消费数据的用法演示，仅对数据做了打印处理，您需要在此基础上自行编写数据处理逻辑**，您也可以使用其他语言的 Kafka 客户端消费并解析数据。
- 目前不支持通过外网连接数据订阅的 Kafka 进行消费，只支持腾讯云内网的访问，并且订阅的数据库实例所属地域与数据消费的地域相同。
- 在订阅指定库/表对象（非源实例全部），并且采用 Kafka 单分区的场景中，DTS 解析增量数据后，仅将订阅对象的数据写入 Kafka Topic 中，其他非订阅对象的数据会转成空事务写入 Kafka Topic，所以在消费数据时会出现空事务。空事务的 Begin/Commit 消息中保留了事务的 GTID 信息，可以保证 GTID 的连续性和完整性。
- 为了保证数据可重入，DTS 订阅引入 Checkpoint 机制。消息写入 Kafka Topic 时，一般每10秒会插入一个 Checkpoint，用来标识数据同步的位点，在任务中断后再重启识别断点位置，实现断点续传。另外，消费端遇到 Checkpoint 消息会做一次 Kafka 消费位点提交，以便及时更新消费位点。

消费 Demo 下载

TDSQL PostgreSQL 数据订阅当前仅支持 ProtoBuf 的数据格式，如下 Demo 中已包含 Protobuf 协议文件，无需另外下载。
如果您选择自行下载 Protobuf 协议文件，请使用 Protobuf 3.X 版本进行代码生成，以便数据结构可以正确兼容。
TDSQL PostgreSQL 消费 Demo 的逻辑说明与 MySQL 的类似，请参考 MySQL Demo 说明。

Demo 语言	ProtoBuf Demo 下载
Go	地址
Java	地址
Python	地址

Java Demo 操作步骤

编译环境：Maven 或者 Gradle 包管理工具，JDK8。用户可自行选择打包工具，如下以 Maven 为例进行介绍。
运行环境：腾讯云服务器（需要与订阅实例相同地域，才能够访问到 Kafka 服务器的内网地址），安装 JRE8。
操作步骤如下：

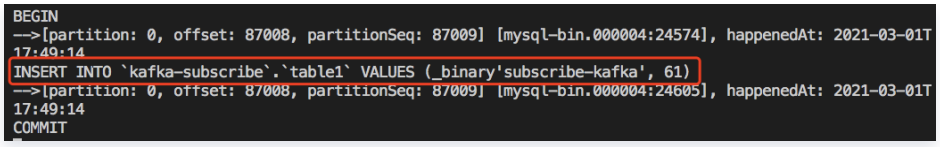
1. 创建新版数据订阅任务，详情请参见 创建 TDSQL PostgreSQL 版数据订阅。
2. 创建一个或多个消费组，详情请参见 新增消费组。
3. 下载 Java Demo，然后解压该文件。
4. 进入解压后的目录，为方便使用，目录下分别放置了 Maven 模型文件、pom.xml 文件，用户根据需要选用。
   使用 Maven 进行打包：mvn clean package。
5. 运行 Demo。
   使用 Maven 打包后，进入目标文件夹 target，运行如下代码。

```
java -jar consumerDemo-avro-1.0-SNAPSHOT.jar --brokers xxx --topic xxx --group xxx --user xxx --password xxx --trans2sql
```

- `brokers` 为数据订阅 Kafka 的内网访问地址，`topic` 为数据订阅任务的订阅 topic，这两个可在 订阅详情 页查看。
- `group`、`user`、`password` 分别为消费组的名称、账号和密码，可在 消费管理 页查看。
- `trans2sql` 表示是否转换为 SQL 语句，java 代码中，携带该参数表示转换为 SQL 语句，不携带则不转换。

> **说明**
> 携带 trans2sql 时，将使用 `javax.xml.bind.DatatypeConverter.printHexBinary()` 将 byte 值转成16进制，请使用 JDK1.8 版本及以上避免不兼容。如果不需要转 SQL，可以注释此处代码。

6. 观察消费情况。



Golang Demo 操作步骤

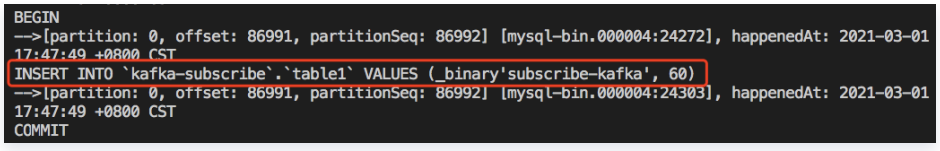
编译环境：Golang 1.12 及以上版本，配置好 Go Module 环境。
运行环境：腾讯云服务器（需要与订阅实例相同地域，才能够访问到 Kafka 服务器的内网地址）。
操作步骤如下：

1. 创建新版数据订阅任务，详情请参见 数据订阅 Kafka 版 。
2. 创建一个或多个消费组，详情请参见 新增消费组 。
3. 下载 Golang Demo，然后解压该文件。
4. 进入解压后的目录，运行 `go build -o subscribe ./main/main.go`，生成可执行文件 subscribe。
5. 运行如下代码：

```
./subscribe --brokers=xxx --topic=xxx --group=xxx --user=xxx --password=xxx --trans2sql=true
```

- `brokers` 为数据订阅 Kafka 的内网访问地址，`topic` 为数据订阅任务的订阅 topic，这两个可在 订阅详情 页查看。
- `group`、`user`、`password` 分别为消费组的名称、账号和密码，可在 消费管理 页查看。
- `trans2sql` 表示是否转换为 SQL 语句。

6. 观察消费情况。



Python3 Demo 操作步骤

编译运行环境：腾讯云服务器（需要与订阅实例相同地域，才能够访问到 Kafka 服务器的内网地址），安装 Python3，pip3（用于依赖包安装）。
使用 pip3 安装依赖包：

```
pip install flag
pip install kafka-python
pip install avro
```

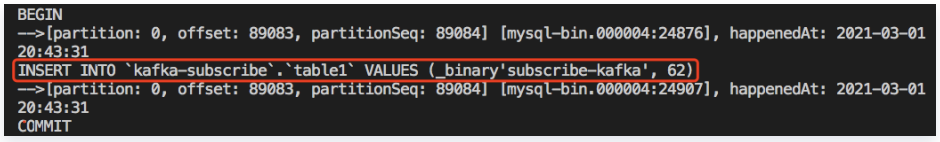
操作步骤如下：

1. 创建新版数据订阅任务，详情请参见 数据订阅 Kafka 版 。
2. 创建一个或多个消费组，详情请参见 新增消费组 。
3. 下载 Python3 Demo ，然后解压该文件。
4. 运行如下代码：

```
python main.py --brokers=xxx --topic=xxx --group=xxx --user=xxx --password=xxx --trans2sql=1
```

- `brokers` 为数据订阅 Kafka 的内网访问地址， `topic` 为数据订阅任务的订阅 topic，这两个可在 订阅详情 页查看。
- `group` 、 `user` 、 `password` 分别为消费组的名称、账号和密码，可在 消费管理 页查看。
- `trans2sql` 表示是否转换为 SQL 语句。

5. 观察消费情况。



前置校验不通过处理
连接 DB 检查

最近更新时间：2024-11-15 10:47:42

检查详情

源数据库和目标数据库需要能正常连通，如果未连通，会报连接失败。

问题原因

- 源数据库所在网络或服务器设置了安全组或防火墙 。
- 源数据库对来源 IP 地址进行了限制 。
- 网络端口未放通 。
- 数据库账号或密码不正确。

修复方法

请按照问题原因中的对应链接进行处理。

版本检查

最近更新时间：2024-10-15 10:27:52

检查详情

登录数据库 `select tbase_version();` 必须返回版本为 Tbase_V2.xx。

修复方法

请选择 TDSQL PostgreSQL 支持的对应版本实例 ID。

周边检查

最近更新时间：2024-10-16 18:01:41

检查详情

DN 节点的 wal_level 必须是 logical。

修复方法

1. 登录源数据库。
2. 找到 postgresql.conf 文件，打开此文件，修改 `wal_level` 。

```
wal_level = logical
```

3. 修改完成后，重启数据库实例。

源实例权限检查

最近更新时间：2024-10-15 10:27:52

检查详情

用户必须拥有 REPLICATION 权限，对应 pg_roles.rolreplication。
用户必须拥有被订阅表的 select 权限，如果是整库订阅，那么用户要拥有该 schema 下所有表的 select 权限。
具体授权请参考 TDSQL PostgreSQL 版数据订阅 。

修复方法

用户可能不具备操作权限，请按照检查要求中的对应权限要求对用户进行授权，然后重新执行校验任务。

约束检查

最近更新时间：2024-10-16 18:01:41

检查详情

被订阅的表必须拥有主键或 REPLICA IDENTITY 为 FULL。

修复方法

当校验不通过时，请修改对应的表格。

MongoDB 数据订阅支持能力

最近更新时间：2024-12-02 09:52:52

功能类别	支持能力
源端版本	腾讯云 MongoDB 3.6（仅支持订阅集合）、4.0、4.2、4.4、5.0的副本集与分片集群。 支持灾备实例和只读实例。
订阅对象	数据库、集合
订阅类型	Change Stream
数据格式	JSON
设置 Topic 分区数	支持
Topic 分区策略	按集合名分区、自定义分区策略
任务关键操作	修改订阅对象

使用说明

最近更新时间：2025-02-10 15:04:21

订阅操作说明

1. DTS 订阅任务地域需要和云数据库所属地域保持一致。
2. 在源数据库中删除已选订阅对象的指定库或者集合后，该库或者集合的订阅数据（Change Stream）将会被无效化，即使在源数据库中重建该库或者集合也无法续订数据，需要重置订阅任务，重新勾选订阅对象。

消费操作说明

1. 数据消费时，不支持通过外网连接数据订阅的 Kafka 进行消费，只支持腾讯云内网的访问，并且消费的地域，需要和 DTS 订阅任务的地域保持一致。
2. 为了保证数据可重入，DTS 订阅引入 Checkpoint 机制。消息写入 Kafka Topic 时，一般每10秒会插入一个 Checkpoint，用来标识数据同步的位点，在任务中断后再重启识别断点位置，实现断点续传。另外，消费端遇到 Checkpoint 消息会做一次 Kafka 消费位点提交，以便及时更新消费位点。

内置 Kafka 说明

1. 订阅的消息保存在 DTS 内置 Kafka（单 Topic）中，目前默认保存时间为最近1天，单 Topic 的最大存储为500G，当数据存储时间超过1天，或者数据量超过500G时，内置 Kafka 都会开始清除最先写入的数据。所以请用户及时消费，避免数据在消费完之前就被清除。
2. DTS 中内置的 Kafka 处理单条消息有一定上限，当源库中的单行数据超过10MB时，这条数据在消费端可能会被丢弃。

支持订阅的事件

事件	说明
DELETE	从集合中删除文档时发生。
DROP	从数据库中删除集合时发生。
DROPDATABASE	删除数据库时发生。
INSERT	在一个操作将文档添加到集合时发生。
INVALIDATE	当操作使变更流无效时发生此事件。
RENAME	重命名集合时发生。
REPLACE	当更新操作从集合中删除文档并将其替换为新文档时发生此事件。
UPDATE	当一个操作更新集合中的文档时发生。

创建 MongoDB 订阅任务

最近更新时间：2024-10-16 16:11:21

操作场景

本场景介绍使用 DTS 创建腾讯云数据库 MongoDB 的数据订阅任务操作指导。

前提条件

- 已准备好待订阅的腾讯云数据库。
- 建议在源端实例中创建只读账号，可参考如下语法进行操作。源库为腾讯云 MongoDB 的，也可 在 MongoDB 控制台创建只读账号 。

```
# 创建全实例只读账号
use admin
db.createUser({
  user: "username",
  pwd: "password",
  roles:[
         {role: "readAnyDatabase",db: "admin"}
        ]
})

# 创建指定库只读账号
use admin
db.createUser({
  user: "username",
  pwd: "password",
  roles:[
         {role: "read",db: "指定库的库名"}
        ]
})
```

订阅配置步骤

1. 登录 DTS 控制台，在左侧导航选择**数据订阅**，单击**新建数据订阅**。
2. 在新建数据订阅页，选择相应配置，单击**立即购买**。

参数名称	配置方式
计费模式	支持**包年包月**和**按量计费**。计费详情请参见 数据传输服务计费模式 。
地域	DTS 订阅服务的地域需要与源数据库实例所属地域保持一致。
数据库	选择 MongoDB。
版本	默认为 Kafka 版。
标签	给数据订阅服务指定标签，可选。

订阅实例名称	• 创建后命名：创建数据订阅服务之后，系统随机分配实例名称，用户可以再自行修改名称。 • 立即命名：在下方输入框，直接设置数据订阅服务名称。
自动续费	建议勾选。若不勾选，请关注服务到期预警信息。具体信息，请参见 欠费说明 。
购买时长	购买时长越长，享受的优惠越多。
购买数量	单次购买最多支持10个任务。

3. 购买成功后，返回数据订阅列表，选择刚才购买的任务，在**操作**列单击**配置订阅**。

4. 在配置数据库订阅页面，配置源库信息后，单击**测试连通性**，通过后单击**下一步**。
   - 接入类型：目前仅支持**云数据库**。
   - 云数据库实例：选择云数据库实例 ID。
   - 数据库账号/密码：添加订阅实例的账号和密码，账号具有只读权限。

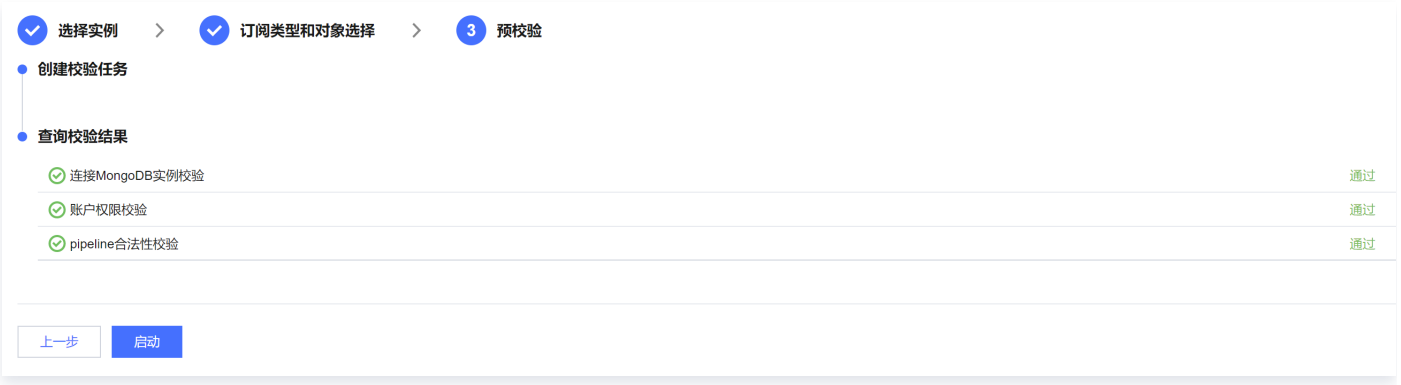
5. 在订阅类型和对象选择页面，选择订阅参数后，单击**保存配置**。

参数	说明
订阅类型	默认Change Stream，不可修改。
订阅数据格式	当前仅支持 JSON 格式。
订阅级别	订阅级别，包括全实例、库和集合。 • 全实例：订阅全实例数据。 • 库：订阅库级别的数据，选择后，下面的任务设置中，只能选择一个库。 • 集合：订阅集合级别的数据，选择后，下面的任务设置中，只能选择一个集合。
任务设置	**订阅级别**选择“库”和“集合”后需要配置。勾选需要订阅的库、集合。 仅支持选择一个库或者一个集合。
输出聚合设置	勾选后，聚合管道的执行顺序由页面上的配置顺序决定。更多聚合管道信息及其使用示例，请参见 MongoDB 官网文档 。
Topic 分区数量	设置 Topic 分区数量，增加分区数量可提高数据写入和消费的速度。单分区可以保障消息的顺序，多分区无法保障消息顺序，如果您对消费到消息的顺序有严格要求，请选择分区数量为1。

Topic 分区策略	当 Topic 分区数量不为1时，可以设置分区策略。 • 按集合名分区：将源库的订阅数据按照集合名进行分区，设置后相同集合名的数据会写入同一个 Kafka 分区中。 • 自定义分区策略：先通过正则表达式对订阅数据中的库名和集合名进行匹配，匹配到的数据再按照集合名分区、集合名 + objectid 分区。

6. 在预校验页面，预校验任务预计会运行2分钟 – 3分钟，预校验通过后，单击**启动**完成数据订阅任务配置。

> **说明：**
> 如果校验失败，请参考 校验不通过处理方法 进行修正，并重新进行校验。



7. 订阅任务进行初始化，预计会运行3分钟 – 4分钟，初始化成功后进入**运行中**状态。

后续操作

创建完订阅任务后，需要进行数据消费操作，详情请参考 消费 MongoDB 订阅数据 。

1. 新增消费组。

   数据订阅的消费依赖于 Kafka 的消费组，所以在消费数据前需要创建消费组。数据订阅 Kafka 版支持用户创建多个消费组（单个订阅任务最多支持创建10个消费组），进行多点消费。

2. 消费订阅数据。

   订阅任务进入运行中状态之后，就可以开始消费数据。Kafka 的消费需要进行密码认证，具体内容请参考上述链接中的 Demo，我们提供了多种语言的 Demo 代码，也对消费的主要流程和关键的数据结构进行了说明。

消费 MongoDB 订阅数据

最近更新时间：2024-06-04 17:11:22

操作场景

数据订阅 Kafka 版（当前 Kafka Server 版本为V2.6.0）中，您可以通过0.11版本及以上的 Kafka 客户端 进行消费订阅数据，本文为您提供了消费 Demo 示例，方便您快速测试消费数据的流程，了解数据格式解析的方法。

注意事项

- **Demo 并不包含消费数据的用法演示，仅对数据做了打印处理，您需要在此基础上自行编写数据处理逻辑**，您也可以使用其他语言的 Kafka 客户端消费并解析数据。
- 目前不支持通过外网连接数据订阅的 Kafka 进行消费，只支持腾讯云内网的访问，并且订阅的数据库实例所属地域与数据消费的地域相同。

消费 Demo 下载

MongoDB 数据订阅当前仅支持 JSON 数据格式，如下 Demo 示例中已包含 JSON 协议文件，无需另外下载。
MongoDB 消费 Demo 采用原生 changeStream 格式，详情请参考 MongoDB 官网 changeStream 格式说明 。

Demo 语言	JSON Demo 下载
Go	地址
Java	地址
Python	地址

Java Demo 操作步骤

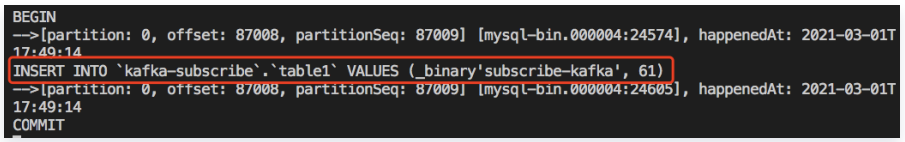
编译环境：Maven 包管理工具，JDK8。用户可自行选择打包工具，如下以 Maven 为例进行介绍。
运行环境：腾讯云服务器（需要与订阅实例相同地域，才能够访问到 Kafka 服务器的内网地址），安装 JRE8。
操作步骤如下：

1. 创建新版数据订阅任务，详情请参见 新建 MongoDB 数据订阅 。
2. 创建一个或多个消费组，详情请参见 新增消费组 。
3. 下载 Java Demo ，然后解压该文件。
4. 进入解压后的目录，为方便使用，目录下分别放置了 Maven 模型文件、pom.xml 文件，用户根据需要选用。
   使用 Maven 进行打包：mvn clean package。
5. 运行 Demo。
   使用 Maven 打包后，进入目标文件夹 target ，运行如下代码：

```
java -jar consumerDemo-json-1.0-SNAPSHOT.jar --brokers xxx --topic xxx --group xxx --user xxx
--password xxx --trans2sql --trans2canal
```

- `brokers` 为数据订阅 Kafka 的内网访问地址，`topic` 为数据订阅任务的订阅 topic，这两个可在 订阅详情 页查看。
- `group` 、`user` 、`password` 分别为消费组的名称、账号和密码，可在 消费管理 页查看，
- `trans2sql` 表示是否转换为 SQL 语句，java 代码中，携带该参数表示转换为 SQL 语句，不携带则不转换。
- `trans2canal` 表示是否转换为 Canal 格式打印出来，携带该参数表示转换为 Canal 格式，不携带则不转换。（当前仅 JSON 数据格式涉及该参数）

6. 观察消费情况。



Golang Demo 操作步骤

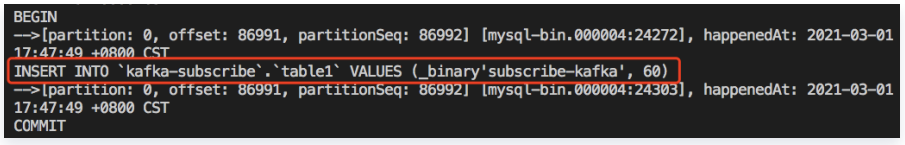
编译环境：Golang 1.12 及以上版本，配置好 Go Module 环境。
运行环境：腾讯云服务器（需要与订阅实例相同地域，才能够访问到 Kafka 服务器的内网地址）。
操作步骤如下：

1. 创建新版数据订阅任务，详情请参见 数据订阅 Kafka 版 。
2. 创建一个或多个消费组，详情请参见 新增消费组 。
3. 下载 Golang Demo，然后解压该文件。
4. 进入解压后的目录，运行 `go build -o subscribe ./main/main.go` ，生成可执行文件 subscribe。
5. 运行如下代码：

```
./subscribe --brokers=xxx --topic=xxx --group=xxx --user=xxx --password=xxx --trans2sql=true
```

- `brokers` 为数据订阅 Kafka 的内网访问地址， `topic` 为数据订阅任务的订阅 topic，这两个可在 订阅详情 页查看。
- `group` 、 `user` 、 `password` 分别为消费组的名称、账号和密码，可在 消费管理 页查看。
- `trans2sql` 表示是否转换为 SQL 语句。

6. 观察消费情况。



Python3 Demo 操作步骤

编译运行环境：腾讯云服务器（需要与订阅实例相同地域，才能够访问到 Kafka 服务器的内网地址），安装 Python3，pip3（用于依赖包安装）。
使用 pip3 安装依赖包：

```
pip install flag
pip install kafka-python
```

操作步骤如下：

1. 创建新版数据订阅任务，详情请参见 数据订阅 Kafka 版 。
2. 创建一个或多个消费组，详情请参见 新增消费组 。
3. 下载 Python3 Demo ，然后解压该文件。
4. 运行如下代码：

```
python main.py --brokers=xxx --topic=xxx --group=xxx --user=xxx --password=xxx --trans2sql=1
```

- `brokers` 为数据订阅 Kafka 的内网访问地址， `topic` 为数据订阅任务的订阅 topic，这两个可在 订阅详情 页查看。
- `group` 、 `user` 、 `password` 分别为消费组的名称、账号和密码，可在 消费管理 页查看。
- `trans2sql` 表示是否转换为 SQL 语句。

5. 观察消费情况。

```
BEGIN
-->[partition: 0, offset: 89083, partitionSeq: 89084] [mysql-bin.000004:24876], happenedAt: 2021-03-01
20:43:31
INSERT INTO `kafka-subscribe`.`table1` VALUES (_binary'subscribe-kafka', 62)
-->[partition: 0, offset: 89083, partitionSeq: 89084] [mysql-bin.000004:24907], happenedAt: 2021-03-01
20:43:31
COMMIT
```

前置校验不通过处理
连接 MongoDB 实例校验

最近更新时间：2025-03-04 14:19:52

检查详情

源数据库和目标数据库需要能正常连通，如果未连通，会报连接失败。

问题原因

- 源数据库所在网络或服务器设置了安全组或防火墙。
- 源数据库对来源 IP 地址进行了限制。
- 网络端口未放通。
- 数据库账号或密码不正确。

修复方法

请按照问题原因中的对应原因进行处理。

账号权限检查

最近更新时间：2024-06-04 17:11:22

检查详情

检查用户是否具备对数据库的操作权限，具体可参考如下对应文档。
数据迁移权限要求： 创建 MongoDB 订阅任务

修复方法

用户若不具备操作权限，请按照检查要求中的对应权限要求对用户进行授权，然后重新执行校验任务。

Pipeline 合法性校验

最近更新时间：2024-06-04 17:11:22

MongoDB 的数据订阅中，DTS 会按照 Change Stream 原生的 pipeline 合法性要求 对源库进行校验，对于和用户数据无关 pipeline 语法性错误，给出界面提示，请用户按照提示修改。

任务管理
任务状态说明

最近更新时间：2024-10-16 17:40:11

说明：

- 仅使用包年包月计费模式的任务显示**续费**的操作。
- 仅使用按量计费模式的任务显示**按量计费转包年包月**的操作。

状态	说明	支持的操作类型
未启动	完成购买操作，未配置订阅任务。	配置订阅、销毁/退货
校验中	订阅任务校验进行中。	销毁/退货
校验通过	订阅任务校验顺利完成。	配置、启动、校验、销毁/退货
校验不通过	订阅任务校验不通过。	配置订阅、校验、销毁/退货
启动中	订阅任务准备运行。	销毁/退货
任务运行	订阅任务运行中。	修改订阅对象、重置订阅、销毁/退货
停止中	重置订阅对象时，订阅任务停止中。	-
任务出错	订阅任务异常出错，任务中断。	恢复、销毁/退货
已隔离	对任务进行**销毁/退货**操作后，任务进入隔离状态。 隔离的任务，用户可以主动发起下线操作，也可以等隔离1天后系统自动下线。	下线

查看订阅详情

最近更新时间：2024-10-16 17:40:11

数据订阅任务创建成功后，用户可查看订阅任务详情。

前提条件

已 创建数据订阅任务 。

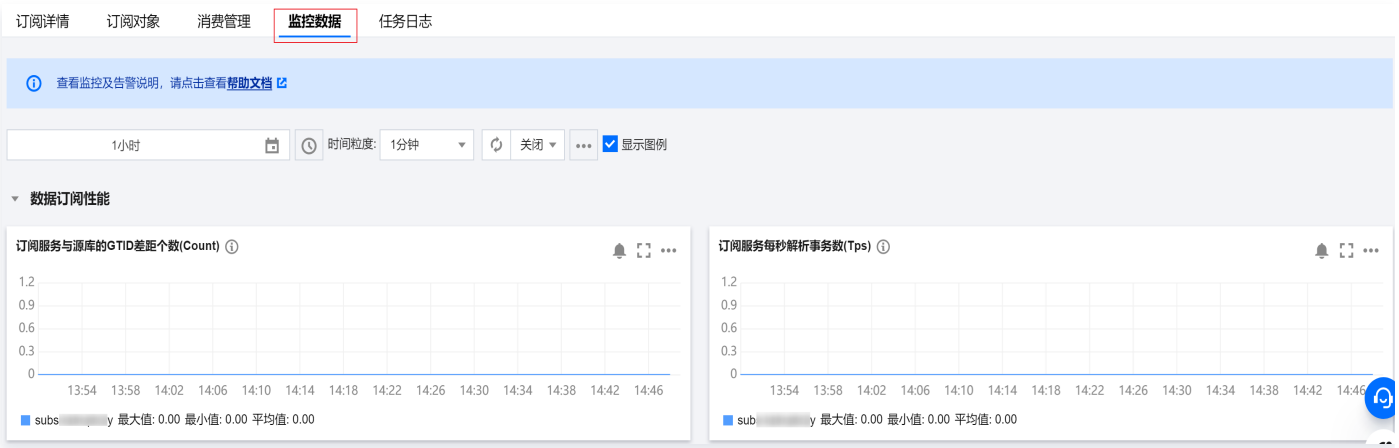
操作步骤

1. 登录 DTS 控制台，在左侧导航选择**数据订阅**，进入数据订阅页面。
   - 方式一：选择指定的订阅任务，单击任务名称。
   - 方式二：选择指定的订阅任务，在**操作**列单击**查看订阅详情**。

2. 切换不同页签，查看订阅详情、订阅对象、消费管理信息。


订阅详情
订阅对象
消费管理
监控数据
任务日志
基础信息
订阅 ID / 名称
订阅类型
全实例
订阅数据格式
ProtoBuf
源实例类型
MySQL
源实例 ID
接入类型
云数据库
所属地域
华南地区（广州）
帐号
root
标签
支持 XA 事务
已关闭
订阅信息
订阅 topic
内网地址
内部 IP 与端口
Kafka 分区数量
1
Kafka 默认分区策略
按表名分区
计费信息
源实例所在地域
华南地区（广州）
创建时间
2022-12-21 17:20:34
计费模式
按量计费

3. 查看监控数据。


订阅详情
订阅对象
消费管理
监控数据
任务日志
查看监控及告警说明，请点击查看帮助文档
1小时
时间粒度:
1分钟
关闭
显示图例
数据订阅性能
订阅服务与源库的GTID差距个数(Count)
订阅服务每秒解析事务数(Tps)
1.2
0.9
0.6
0.3
0
13:54
13:58
14:02
14:06
14:10
14:14
14:18
14:22
14:26
14:30
14:34
14:38
14:42
14:46
最大值: 0.00 最小值: 0.00 平均值: 0.00

4. 查看任务日志。


订阅详情
订阅对象
消费管理
监控数据
任务日志
全部
今天
昨天
近 7 天
近 30 天
选择日期
选择日期
操作时间
状态
2022-12-21 17:27:16
任务运行
2022-12-21 17:26:27
启动中
2022-12-21 17:23:46
任务运行
2022-12-21 17:23:32
启动中
2022-12-21 17:23:27
校验通过
2022-12-21 17:22:52
校验中

修改订阅对象

最近更新时间：2024-10-16 17:40:11

操作场景

订阅任务启动后，用户如需增加、删除订阅对象，可通过本操作进行。
当前支持如下参数修改，其他参数的修改请参考 重置数据订阅 。

- 修改订阅类型，增加、删除订阅对象。
- 修改 Kafka 分区策略，自定义分区策略。

前提条件

已创建 数据订阅 Kafka 版 。

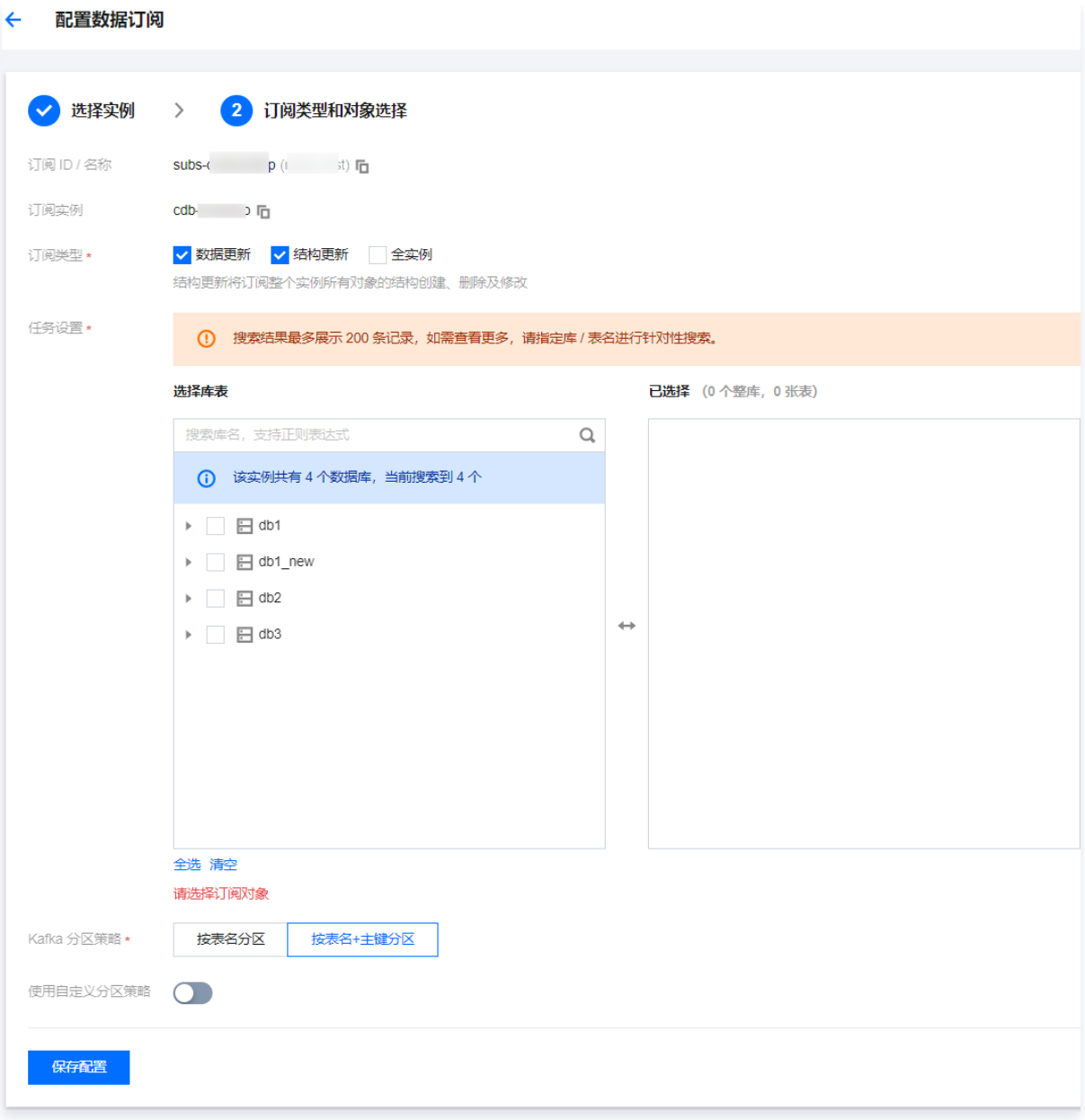
注意事项

1. 修改订阅对象后，原内网地址不变，仍绑定在该订阅任务上。内网地址可通过**查看订阅详情** > **订阅信息**进行查看。
2. 修改订阅对象操作后，会导致订阅任务会重启，发生秒级的暂停，重启后系统会识别源库的 Binlog 中断位点并继续订阅，数据不会中断。
3. MySQL/TDSQL-C MySQL/MariaDB/Percona/TDSQL PostgreSQL 订阅任务中，修改订阅对象操作后会发生任务重启，重启后可能会导致用户在 kafka 客户端消费数据时出现重复。MongoDB 订阅任务中，重启后不会发生数据重复。
   - MySQL/TDSQL-C MySQL/MariaDB/Percona/TDSQL PostgreSQL 订阅任务中，DTS 是按最小数据单元进行传输的，每标记一个 checkpoint 位点就是一个数据单元，如果重启时，刚好一个数据单元传输已完成，则不会导致数据重复；如果重启时，一个数据单元还正在传输中，那么再次启动后需要重新拉取这个数据单元，以保证数据完整性，这样就会导致数据重复。
   - MySQL/TDSQL-C MySQL 当前已支持数据去重，消费 Demo 中包含了去重逻辑，不会产生数据重复，MariaDB/Percona/TDSQL PostgreSQL，用户如果对重复数据比较关注，请自行在消费数据时设置去重逻辑。
4. 新增对象后，订阅的数据内容，针对的是任务重启后源库产生的增量数据。

操作步骤

1. 登录 DTS 控制台 ，在左侧导航选择**数据订阅**，进入数据订阅页面。
2. 在数据订阅列表，选择所需的数据订阅，选择**操作**列的**更多** > **修改订阅对象**，进入修改订阅对象页面。

3. 在修改订阅对象页面，重新选择订阅类型，勾选订阅对象，或者修改分区策略，然后单击**保存配置**。



4. 返回订阅列表，订阅任务进入启动中状态，进行任务预检查并初始化。启动成功后，订阅任务进入运行状态，即可使用 Kafka 客户端消费订阅数据。

按量计费转包年包月

最近更新时间：2024-10-16 17:40:11

操作场景

DTS 支持将按量计费任务转为包年包月。用户转换计费模式后，DTS 会生成包年包月的续费订单，需要用户及时支付续费订单才能转换成功。

操作步骤

1. 登录 DTS 控制台，在左侧导航选择**数据订阅**，进入数据订阅页面。
2. 在数据订阅列表，选择所需修改的数据订阅，选择**操作**列的**更多 > 按量计费转包年包月**。

3. 在弹出的对话框中，设置续费时长等信息，确认无误后，单击**立即转换**。

重置订阅

最近更新时间：2024-10-16 17:40:11

操作场景

重置数据订阅，即停止原任务，删除原订阅任务中的参数配置和订阅数据，删除后不可再恢复。重置后任务状态为“未配置”，用户可以重新配置所有参数。重置操作适用于不需要的订阅任务，通过重置后快速配置新任务，用户无需将老任务退费，再购买新任务，方便操作。
在 修改订阅对象 操作中，仅支持增加、删除订阅对象，修改分区策略，如果用户需要其他参数，如订阅数据格式，Kafka 分区，支持 XA 事务等，确认原任务不再需要后，可以通过重置来重新配置。

前提条件

- 原订阅任务不再需要。
- 数据订阅任务状态为**任务运行**或**任务出错**。

操作步骤

1. 登录 DTS 控制台，在左侧导航选择**数据订阅**，进入数据订阅页面。
2. 在数据订阅列表，选择所需的数据订阅，选择**操作**列的**更多 > 重置订阅**。

3. 在弹出的对话框，确认无误后，单击**确定**。

> **注意：**
> - 订阅重置后，会解除订阅任务和源数据库的绑定关系，同时恢复至“未配置”状态，您可以再次进行初始化配置。
> - 重置订阅操作将停止订阅源库增量数据，订阅任务中已存储的增量数据将被删除，请谨慎操作。
> - 重置订阅后，原内网地址不变，仍绑定在该订阅任务上。内网地址可通过**查看订阅详情 > 订阅信息**进行查看。

4. 在数据订阅列表，任务状态已变为“未启动”，单击**操作**列的**配置订阅**，重新配置任务。


新建数据订阅
编辑标签
更多操作
数据库智能选择
new
EventBridge
new
多个关键字用竖线 | 分隔，多个过滤标签用回车键分隔
订阅 ID / 名称
监控 / 状态
订阅配置
数据库
消费时间起点
SDK 当前消费时间位点
计费模式
创建时间
到期时间
操作
subs-
Kafka 版
未启动
Topic: --
接实例: --
MySQL
--
--
按量计费
2023-02-07 15:18:56
--
配置订阅
更多

销毁退货实例

最近更新时间：2024-10-16 17:40:11

操作场景

- 销毁：对于已结束或者失败等不再需要的任务，用户可销毁任务，任务销毁后就不会存在，也不再产生计费。
- 退货：使用包年包月购买的任务，用户不需要使用，可以进行退货处理。相关的退费请参考 退费说明 。

销毁或退货后任务将进入回收站，不可恢复，回收站保留7天后自动下线，请谨慎处理。

操作步骤

1. 登录 DTS 控制台 ，在左侧导航选择**数据订阅**页，选择指定任务，在**操作**列选择**更多 > 销毁 /退货**。

2. 在弹出的对话框中，确认无误后勾选**已阅读并同意销毁规则**，然后单击**立即销毁**。

消费管理
新增消费组

最近更新时间：2024-10-16 16:11:21

操作场景

数据订阅 Kafka 版支持用户创建多个消费组，进行多点消费。通过创建多个消费组，您可以进行多次消费，增加消费通道降低使用成本，提升消费数据速度。

> **说明：**
> 请您先创建一个用于测试的消费组，待后续消费程序的功能都测试完成，进行正式的消费数据时，再创建一个正式的消费组，防止测试数据和正式数据在一个消费组中无法区分。

前提条件

已创建 数据订阅 Kafka 版。

注意事项

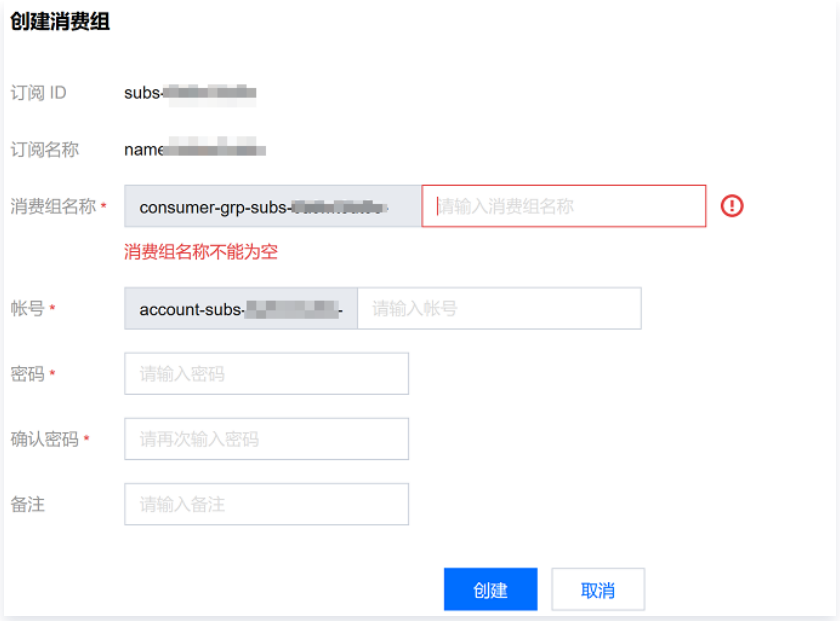
- 数据订阅任务须为运行中状态。
- 一个数据订阅任务最多创建10个消费组，如需增加消费组，请另外新建同步任务。
- 一个消费组只能关联一个 User 账号，多个消费者可使用同一个 User 账号进行消费。

操作步骤

1. 登录 DTS 控制台，在左侧导航选择**数据订阅**，进入数据订阅页面。
2. 在数据订阅列表，选择所需的数据订阅，单击订阅名或**操作**列的**查看订阅详情**，进入订阅管理页面。

3. 在订阅管理页面，选择**消费管理**页，单击**新增消费组**。

4. 在弹窗的新建消费对话框，设置消费组信息，单击**创建**。
   - 消费组名称：根据实际情况创建消费组名称，方便使用和识别。
   - 账号：设置消费组账号。
   - 密码：设置消费组账号对应密码。
   - 确认密码：再次输入相同密码。
   - 备注：设置备注信息，方便备注信息记录。


创建消费组
订阅 ID
subs-
订阅名称
name
消费组名称 *
consumer-grp-subs-
请输入消费组名称
消费组名称不能为空
帐号 *
account-subs-
-
请输入帐号
密码 *
请输入密码
确认密码 *
请再次输入密码
备注
请输入备注
创建
取消

管理消费组

最近更新时间：2024-05-21 11:41:02

操作场景

数据订阅 Kafka 版支持对消费组进行管理，包括修改密码、删除消费组、修改消费位点 offset 等。
同时，用户在消费管理中可查看 Client ID（显示消费该组数据的客户端 ID）、分区编号、消费点、消费点写入分区时间、消费延迟等信息。

前提条件

已创建 消费组 。

注意事项

在同一个消费组中，当有其他 Kafka 客户端正在消费数据时，不支持修改消费位点。

修改消费组密码

1. 登录 DTS 控制台，在左侧导航选择**数据订阅**，进入数据订阅页面。
2. 在数据订阅列表，选择所需的数据订阅，单击订阅名或**操作**列的**查看订阅详情**，进入订阅管理页面。
3. 在订阅管理页面，选择**消费管理**页，单击**操作**列的**修改密码**。

4. 在弹出的对话框，修改密码，单击**修改**。

删除消费组

1. 登录 DTS 控制台，在左侧导航选择**数据订阅**，进入数据订阅页面。
2. 在数据订阅列表，选择所需的数据订阅，单击订阅名或**操作**列的**查看订阅详情**，进入订阅管理页面。
3. 在订阅管理页面，选择**消费管理**页，单击**操作**列的**删除**。
4. 在弹出的对话框，确认无误后，单击**确定**。

> **注意**
> 删除消费组后，该消费组中的消费点位将被删除，但数据订阅中的数据不会被删除，请谨慎操作。

修改消费位点 offset

最近更新时间：2024-05-21 11:41:05

操作场景

本操作指导用户修改消费位点 offset。

操作步骤

1. 登录 DTS 控制台，在左侧导航选择**数据订阅**，进入数据订阅页面。
2. 在数据订阅列表，选择所需的数据订阅，单击订阅名或**操作**列的**查看订阅详情**，进入订阅管理页面。
3. 在订阅管理页面，选择**消费管理**页，选择对应的消费组，单击**修改 offset**。

4. 在弹出的对话框中，选择 offset 对象，单击**下一步**。

5. 设置消费数据的重置方式，然后单击**完成**。
   - 从最新位置开始消费：消费位点改为 Kafka 消息的最新位点。
   - 从最开始位置开始消费：消费位点改为 Kafka 消息最开始存储的位点。
   - 按时间点进行消费位置重置：用户输入对应的时间，DTS 找到用户输入时间对应的消费位点，然后改为从该消费位点开始消费。当用户设置的时间超过 Kafka 消息的时间，消费位点将以 Kafka 消息中的最大位点为准；当用户设置的时间早于 Kafka 消息的时间，消费位点将以 Kafka 消息中的最小位点为准。


offset设置
对象选择
2
offset设置
重置方式
从最新位置开始消费
从最开始位置开始消费
按时间点进行消费位置重置
日期和时间
选择时间
请选择重置时间
分区编号
消费点
未消费记录数
0
10079
22
1
10108
22
2
10078
22
3
10078
22
上一步
完成

多消费者设置 Client ID

最近更新时间：2024-08-27 15:19:11

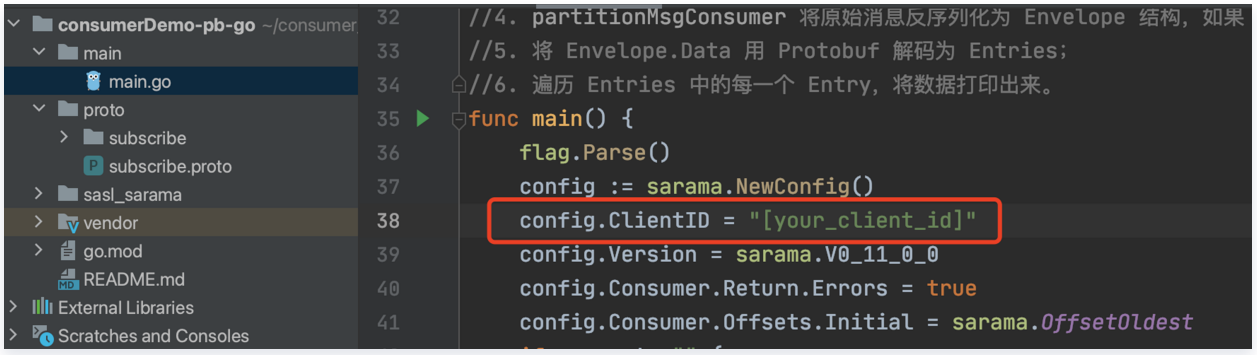
操作场景

用户使用多个消费端消费时，每个消费端可以先设置 Client ID，然后在 DTS 控制台中展示具体的 Client ID，这样可以帮助用户定位消费程序异常问题。

操作步骤

1. 在每个消费程序中，设置 Client ID。如果启动多个消费者，建议为不同消费者设置不同的 Client ID 值。

   如下以 golang 消费程序为例进行介绍，其他语言的消费程序操作类似。
   在 config 中设置 ClientID。如果没有设置，默认的 ClientID 值为 "sarama"。
   config.ClientID = "[your_client_id]"



2. 开始消费后，在 DTS 控制台打开对应的订阅任务，单击**消费管理**即可看到正在消费的消费者 Client ID。

   对比这里的 Client ID 和消费程序中设置的 Client ID 是否一致。
   - 如果一致，消费程序符合预期。
   - 如果不一致，可以帮助定位消费程序异常问题。例如出现的 Client ID 不是用户设置的，则为未知消费程序正在消费数据。