ShardingSphere

Apache ShardingSphere 数据迁移功能 & 实战

郭信泽 2022-10-15

郭信泽

ShardingSphere Active Contributor
SphereEx 研发工程师

- 目前专注于 ShardingSphere 弹性迁移、SphereEx 弹性扩缩容相关的研发工作

目录

Apache ShardingSphere 数据迁移介绍

随着业务持续发展，数据量和并发量达到一定程度，传统数据库可能面临性能、可扩展性、可用性等问题。

通过 ShardingSphere 提供的数据迁移方案可以助力传统数据库平滑切换到 ShardingSphere，内置于 ShardingSphere-Proxy。

目前支持的数据库类型

1. MySQL

2. PostgreSQL

3. openGauss

除此之外，也支持 Oracle 到 MySQL/PostgreSQL 的异构迁移。

Apache ShardingSphere 数据迁移介绍

数据迁移的要求

	数据库	版本支持	环境要求	权限要求
1	MySQL	5.1.15 ~ 8.0.x	`my.cnf` 配置 log-bin=binlog binlog-format=row binlog-row-image=full	GRANT REPLICATION SLAVE,REPLICATION CLIENT ON .TO ${username}@${host}
2	PostgreSQL	9.4及以上版本	`postgresql.conf` 配置 wal_level = logical max_wal_senders = 10 max_replication_slots = 10 max_connections = 600	`pg_hba.conf` 配置 host all ${username} 0.0.0.0/0 md5
3	openGauss	2.0.1 ~ 3.0.0	`postgresql.conf` 配置 wal_level = logical max_wal_senders = 10 max_replication_slots = 10 max_connections = 600 wal_sender_timeout = 0	`pg_hba.conf` 配置 host all ${username} 0.0.0.0/0 md5

数据迁移相关概念

原始数据所在的存储集群

源　　端

原始数据将要迁移的目标存储集群

目　标　端

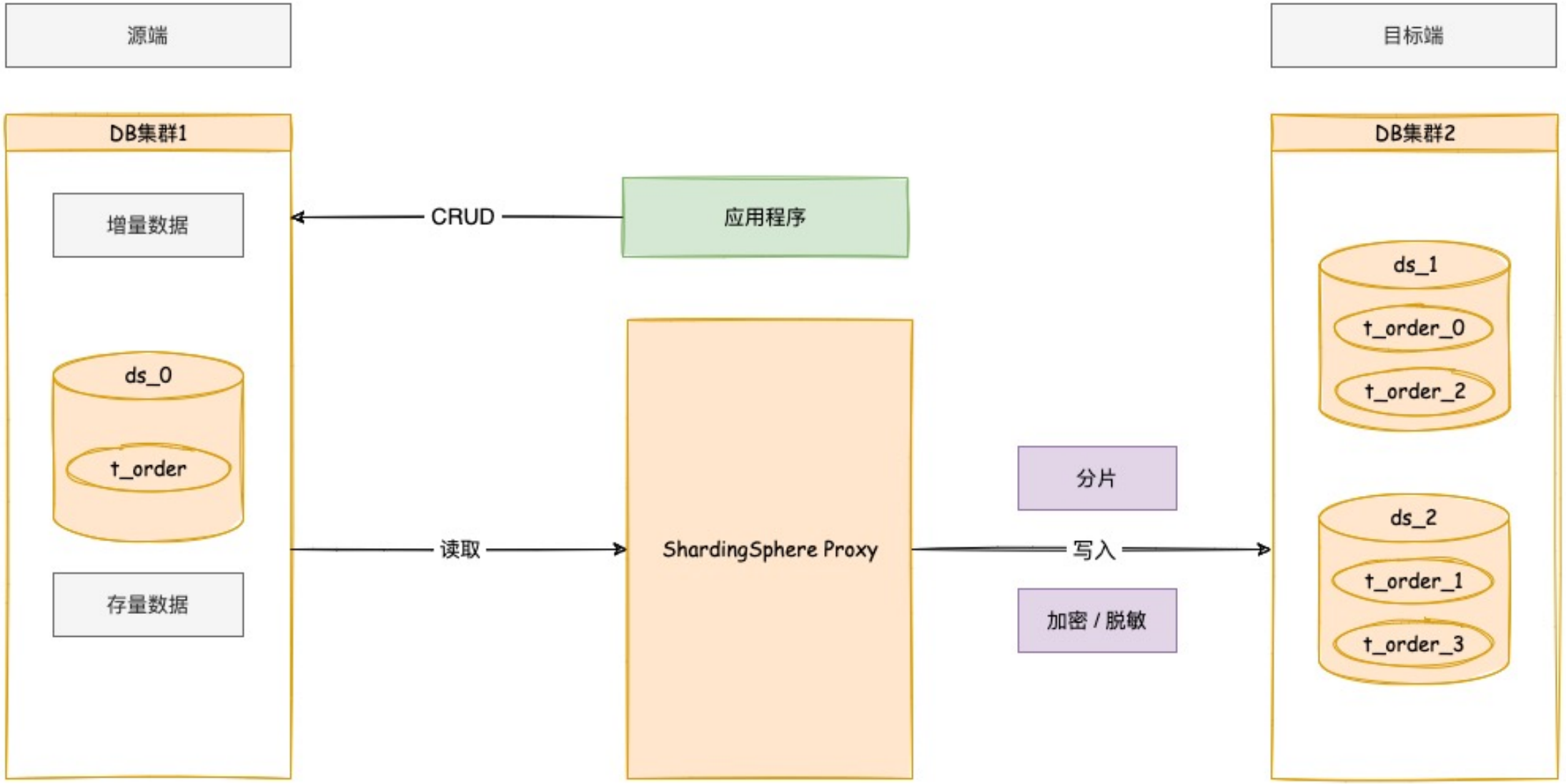
在数据迁移开始前，数据节点中已有的数据

存 量 数 据

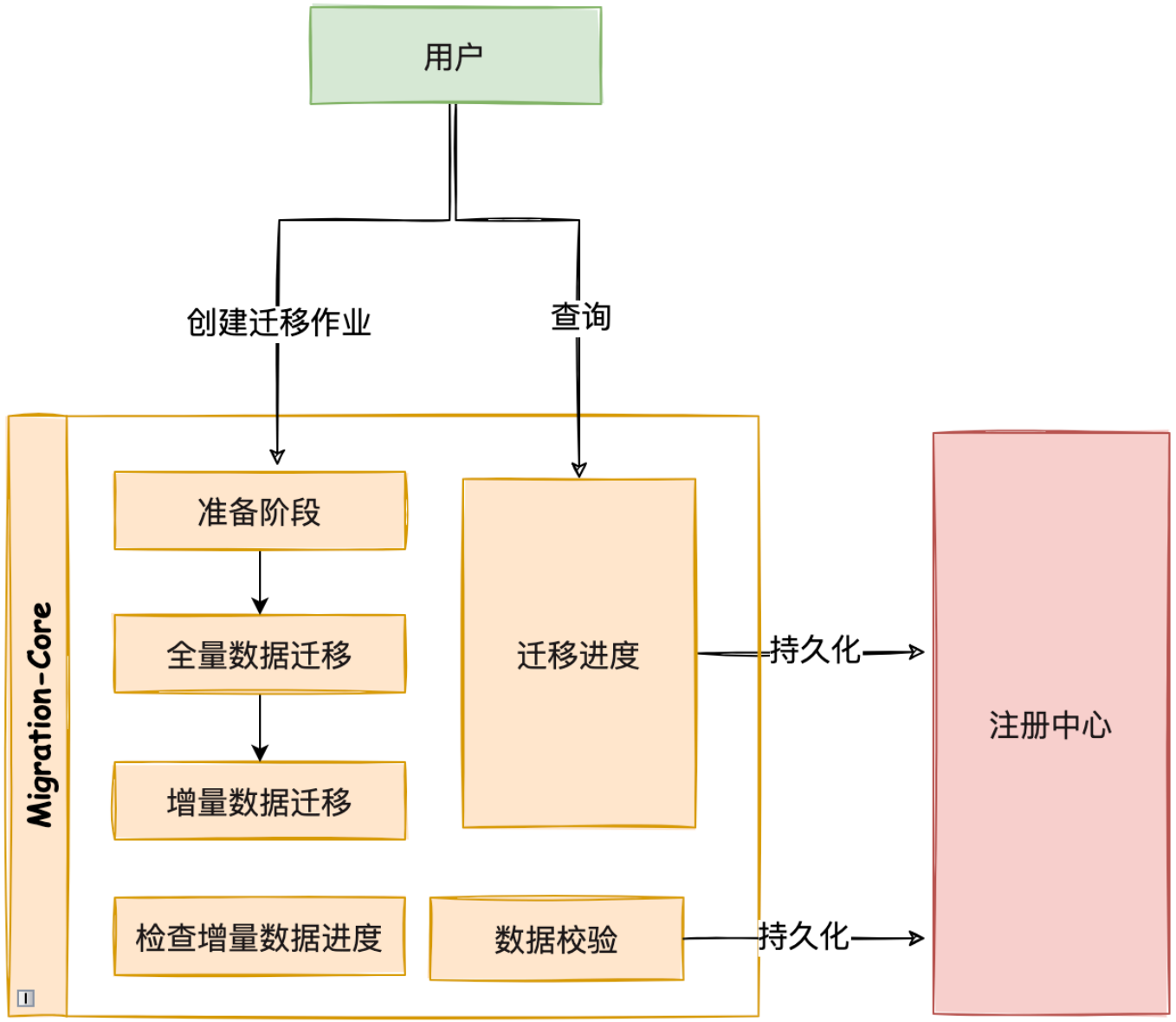
在数据迁移执行过程中，业务系统产生的新数据

增 量 数 据

数据迁移流程


源端
目标端
DB集群1
DB集群2
增量数据
CRUD
应用程序
ds_0
t_order
ds_1
t_order_0
t_order_2
分片
读取
ShardingSphere Proxy
写入
加密 / 脱敏
存量数据
ds_2
t_order_1
t_order_3

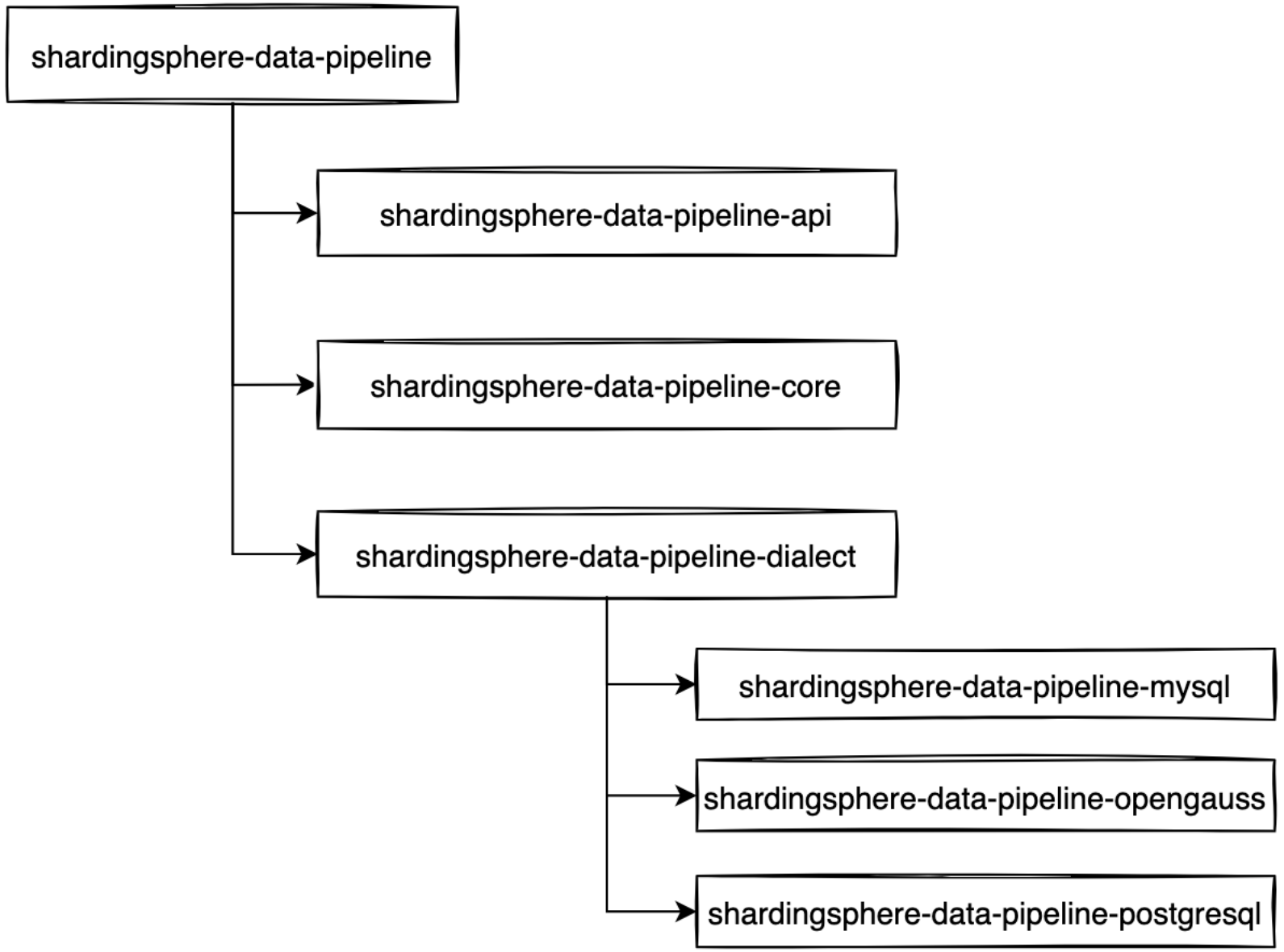
数据迁移阶段


用户
创建迁移作业
查询
Migration-Core
准备阶段
全量数据迁移
增量数据迁移
检查增量数据进度
迁移进度
数据校验
持久化
持久化
注册中心

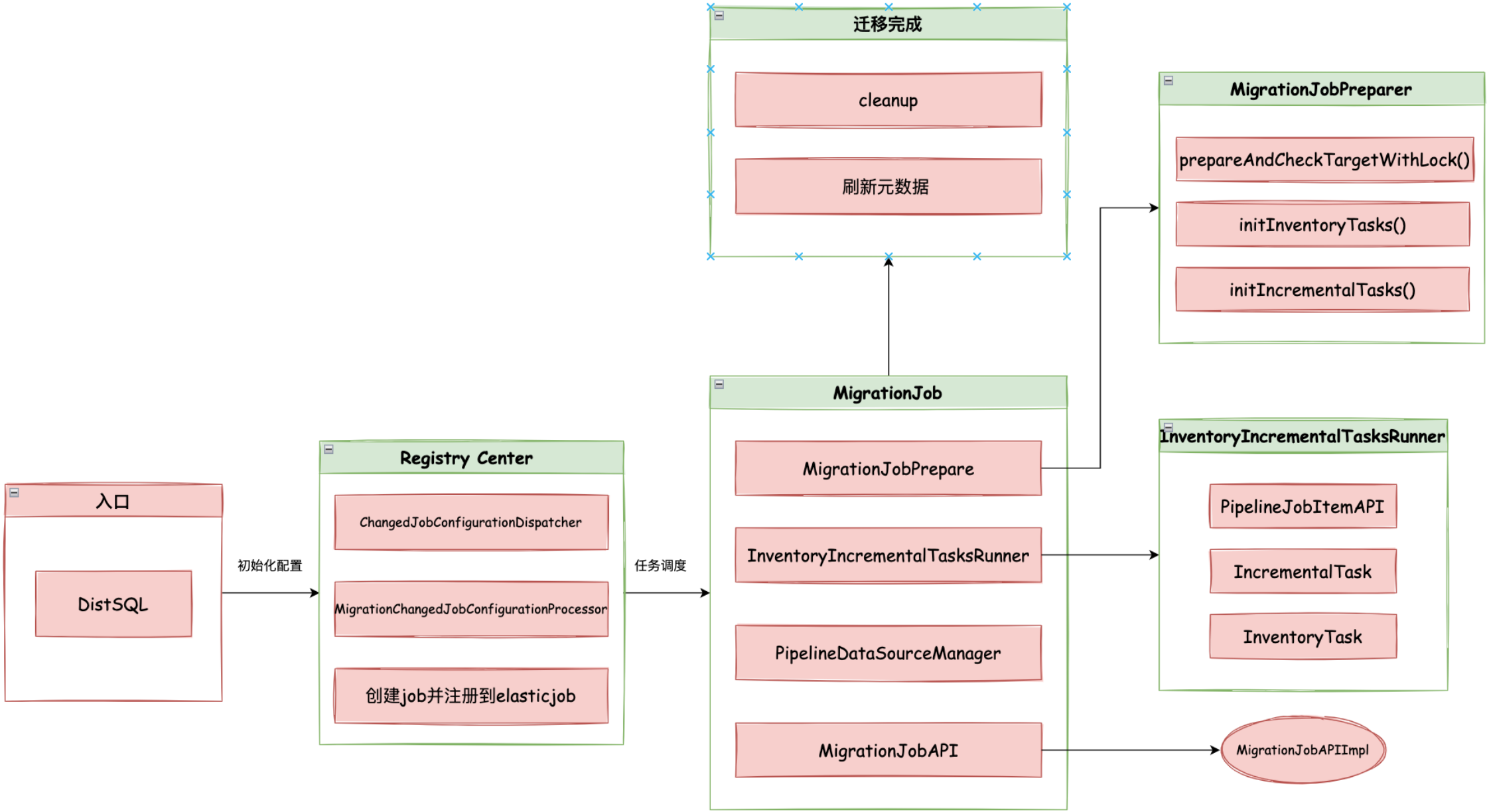
目录

Apache ShardingSphere 数据迁移源码解析

数据迁移模块结构

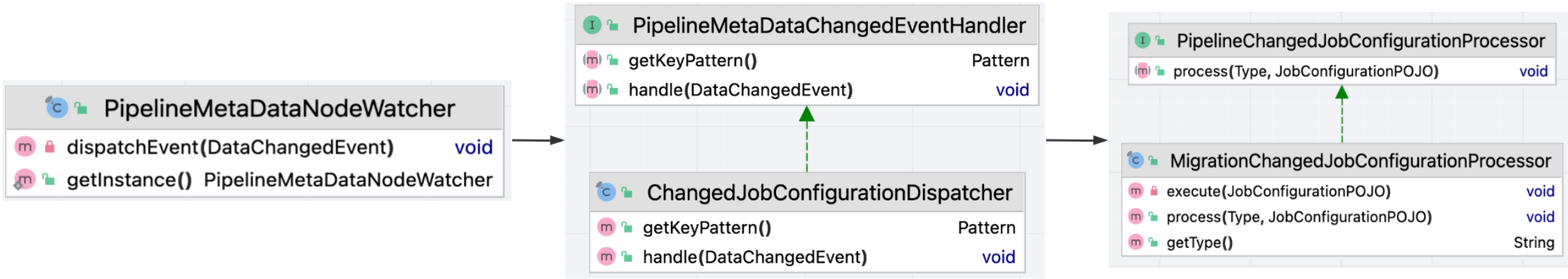


Apache ShardingSphere 数据迁移源码解析



Apache ShardingSphere 数据迁移—— Job 创建

迁移 job 的创建依赖注册中心（ZooKeeper）的事件监听，向指定的节点写入 job 的配置信息。
先后涉及的类如下：



Apache ShardingSphere 数据迁移—— Job 执行

数据迁移底层使用 ElasticJob 进行任务的分片和执行，具体的实现类是 MigrationJob

```
public final class MigrationJob extends AbstractPipelineJob implements SimpleJob, PipelineJob {
    ......
    @Override
    public void execute(final ShardingContext shardingContext) {
        MigrationJobConfiguration jobConfig =
YamlMigrationJobConfigurationSwapper.swapToObject(shardingContext.getJobParameter());
    ......
        MigrationJobItemContext jobItemContext = new MigrationJobItemContext(jobConfig, shardingItem, initProgress,
jobProcessContext, taskConfig, dataSourceManager);
        InventoryIncrementalTasksRunner tasksRunner = new InventoryIncrementalTasksRunner(jobItemContext,
jobItemContext.getInventoryTasks(), jobItemContext.getIncrementalTasks());
        runInBackground(() -> {
            prepare(jobItemContext);
            tasksRunner.start();
        });
    }
```

Apache ShardingSphere 数据迁移——准备阶段

Prepare 是数据迁移的第一个阶段，MigrationJobPreparer 的处理逻辑如下：

1. 检查源端数据库所需的用户权限和数据库配置项是否开启，比如 MySQL 需要打开 binlog，PostgreSQL 的 wal_level >= logical

2. 目标端建表（and schema）

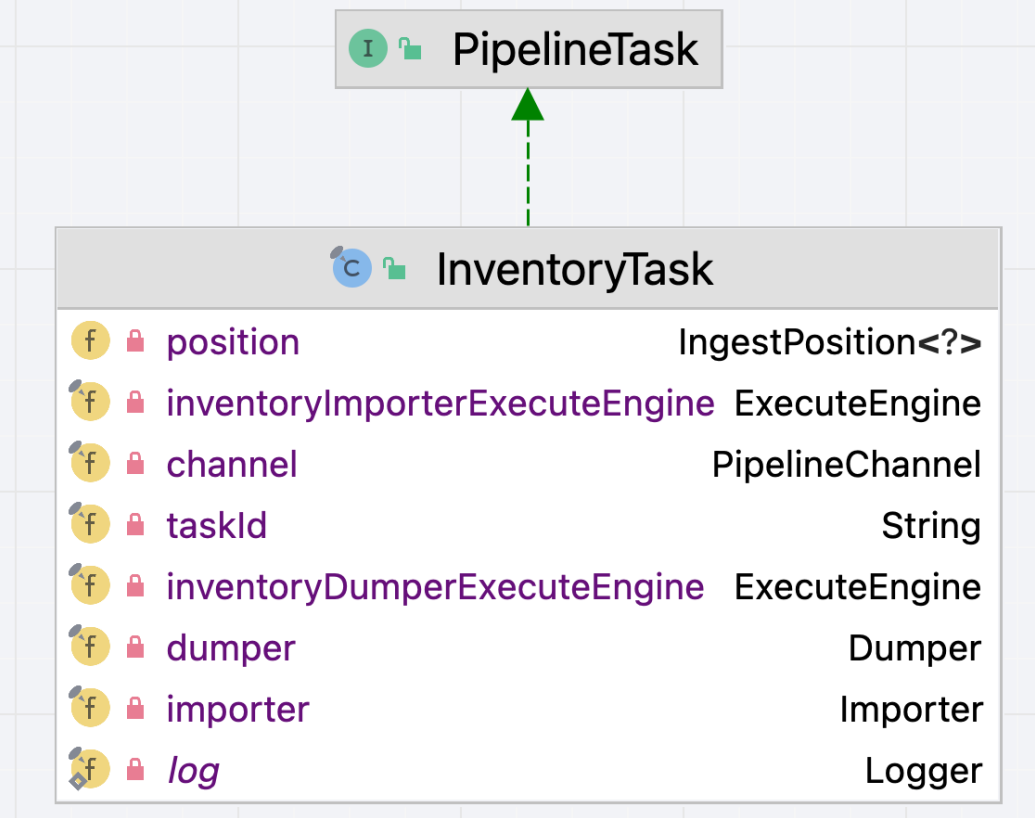
3. 检查目标端表是否为空

4. 初始化增量迁移的 Task，不同的数据库具有不同的实现，MySQL 是伪装成 slave，进行数据同步

5. 初始化全量迁移的 Task，计算每个 Task 负责迁移的数据范围

Apache ShardingSphere 数据迁移——全量迁移

目前要求迁移的表具有主键或者唯一键，作为数据段的切分的依据。每个 InventoryTask 中的 dumper 负责源端数据的导出，importer 负责目标端数据的导入，它们之间通过 PipelineChannel 联接



数据量大的情况下，通过限流的方式来控制迁移的速率，避免占用大量系统资源，目前支持两种限流方式，TPS 和 QPS，可以通过 DistSQL 进行配置

```
CREATE MIGRATION PROCESS CONFIGURATION (
READ(
  RATE_LIMITER
(TYPE(NAME='QPS',PROPERTIES('qps'='500')))
)
);
```

Apache ShardingSphere 数据迁移——增量迁移之 MySQL

增量迁移下，不同数据库差异较大，MySQL 用的是 binlog，需要把 BINLOG_FORMAT 设置为 ROW 模式，通过 netty 伪装成 MySQL 实例，并设置 master 为迁移源端数据库。步骤如下：

1. 准备阶段通过 show master status；获取主节点的 binlog 文件名和当前最新位点。

2. 增量同步阶段，先连接并登录到 MySQL 主节点（目前支持两种密码加密方式，**caching_sha2_password** 和 **mysql_native_password**）

3. 再发送 COM_REGISTER_SLAVE 事件注册为从节点，这里是模拟的从节点
参数提供从节点的 server_id / hostname / port。这里的 server_id 使用的是随机数

4. 再发送 COM_BINLOG_DUMP 事件开始接收 binlog stream
参数提供从节点的 server_id、主节点的 binlog-filename / binlog_pos

5. 最后循环解析事件流，放入内存队列；这个实现在网络层，生产者会从这里拉取数据、过滤掉不需要的表的事件、然后处理增删改事件

增量迁移也支持 MySQL MGR 高可用模式

Apache ShardingSphere 数据迁移——增量迁移之PostgreSQL

PostgreSQL 使用的是 logical replication slots 进行数据复制

1. 准备阶段创建逻辑复制槽并获取位点
- 通过 pg_create_logical_replication_slot SQL 函数创建逻辑复制槽，参数提供复制槽名称和解码输出插件名称（test_decoding）
- 通过 pg_current_xlog_location() / pg_current_wal_lsn() SQL 函数获取位点，不同版本的函数略有区别

2. 增量同步阶段，通过逻辑复制流获取数据变更
- 通过 PGConnection.getReplicationAPI().replicationStream().logical() 获取逻辑复制流，参数提供复制槽名称和位点

3. 最后生产者循环解析复制流、过滤不需要的表的事件、然后处理增删改事件

目前 PostgreSQL 插槽名称长度要求 < 64，超过的部分会自动截断。openGauss 则是直接报错，插件也从 test_decoding 变为了 mppdb_decoding（JSON格式）

Apache ShardingSphere 数据迁移——重复数据处理

迁移任务的进度是异步持久化的，定位点可能存在 1s 左右误差，当 Job 重启或者项目重启之后，可能出现部分已经写入目标端的数据再次写入，导致主键唯一键冲突，需要实现插入或者更新数据的幂等性。

解决办法：类似于幂等消费 MQ 重复消息，使用数据库提供的 insert or update 进行幂等插入。

MySQL：insert into ... on duplicate key update

PostgreSQL：insert into ... on conflict ... do update

Apache ShardingSphere 数据迁移——一致性校验

前提条件：当增量数据同步持续一段时间没有同步新数据，

1. 数据一致性校验之前，可以停写源端库，或者不停写。

如果停写，那所有校验算法都可以全面校验所有数据的一致性。

如果不停写，需要允许部分增量数据不校验。支持增量数据变更的一致性校验还在规划中

2. 支持数据分段校验。发现部分不一致即可及时中断

3. 支持的校验算法

- CRC32_MATCH，

只有 MySQL 支持，SELECT BIT_XOR(CAST(CRC32(columnName) AS UNSIGNED)) AS checksum FROM tableName，该 CRC32() 函数计算循环冗余校验值并返回 32 位无符号值

- DATA_MATCH

每行数据逐个比对各个 column 的字段值，效率比 CRC32 稍微差点，但是支持所有数据库

目录

数据迁移实操展示

准备工作：

- ShardingSphere-Proxy 5.2.1 or master branch & cluster mode
- MySQL（本次使用 MySQL8）
- ZooKeeper 3.8.0

数据迁移实操展示

配置初始化

```
mysql> create database sharding_db;
Query OK, 0 rows affected (0.05 sec)

mysql> use sharding_db;
Database changed
mysql> ADD RESOURCE ds_0 (
    ->     URL="jdbc:mysql://127.0.0.1:3306/target_ds_0?serverTimezone=UTC&useSSL=false",
    ->     USER="root",
    ->     PASSWORD="root",
    ->     PROPERTIES("maxPoolSize"="50","idleTimeout"="60000")
    -> );
Query OK, 0 rows affected (1.45 sec)

mysql> ADD RESOURCE ds_1 (
    ->     URL="jdbc:mysql://127.0.0.1:3306/target_ds_1?serverTimezone=UTC&useSSL=false",
    ->     USER="root",
    ->     PASSWORD="root",
    ->     PROPERTIES("maxPoolSize"="50","idleTimeout"="60000")
    -> );
Query OK, 0 rows affected (0.16 sec)

mysql> CREATE SHARDING TABLE RULE t_order(
    -> RESOURCES(ds_0,ds_1),
    -> SHARDING_COLUMN=order_id,
    -> TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="6")),
    -> KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
    -> );
Query OK, 0 rows affected (2.91 sec)

mysql> ADD MIGRATION SOURCE RESOURCE source_ds (
    ->     URL="jdbc:mysql://127.0.0.1:3306/source_ds?serverTimezone=UTC&useSSL=false",
    ->     USER="root",
    ->     PASSWORD="root",
    ->     PROPERTIES("maxPoolSize"="50","idleTimeout"="60000")
    -> );
Query OK, 0 rows affected (0.24 sec)
```

数据迁移实操展示

执行迁移

```
mysql> MIGRATE TABLE source_ds.t_order INTO sharding_db.t_order;
Query OK, 0 rows affected (0.83 sec)

mysql> show migration list;
+-----------------------------------+---------+---------------------+--------+---------------------+-----------+
| id                                | tables  | sharding_total_count | active | create_time         | stop_time |
+-----------------------------------+---------+---------------------+--------+---------------------+-----------+
| j01019761922f8fab5ce3c61496c22f733586 | t_order | 1                   | true   | 2022-10-12 09:14:36 | NULL      |
+-----------------------------------+---------+---------------------+--------+---------------------+-----------+
1 row in set (0.47 sec)

mysql> show migration status 'j01019761922f8fab5ce3c61496c22f733586';
+------+-------------+--------------------------+--------+-------------------------+---------------------------------+---------------------------+---------------+
| item | data_source | status                   | active | processed_records_count | inventory_finished_percentage | incremental_idle_seconds | error_message |
+------+-------------+--------------------------+--------+-------------------------+---------------------------------+---------------------------+---------------+
| 0    | source_ds   | EXECUTE_INCREMENTAL_TASK | true   | 8                       | 100                             | 4                         |               |
+------+-------------+--------------------------+--------+-------------------------+---------------------------------+---------------------------+---------------+
1 row in set (0.04 sec)
```

数据迁移实操展示

执行数据校验

```
mysql> CHECK MIGRATION 'j01019761922f8fab5ce3c61496c22f733586' BY TYPE (NAME='DATA_MATCH');
Query OK, 0 rows affected (0.07 sec)

mysql> SHOW MIGRATION CHECK STATUS 'j01019761922f8fab5ce3c61496c22f733586';
+----------+--------+---------------------+-------------------+-------------------------+-------------------------+-------------------+-----------------+
| tables   | result | finished_percentage | remaining_seconds | check_begin_time        | check_end_time          | duration_seconds | error_message |
+----------+--------+---------------------+-------------------+-------------------------+-------------------------+-------------------+-----------------+
| t_order | true   | 100                 | 0                 | 2022-10-12 09:16:18.842 | 2022-10-12 09:16:19.502 | 0                 |                 |
+----------+--------+---------------------+-------------------+-------------------------+-------------------------+-------------------+-----------------+
1 row in set (0.02 sec)

mysql> REFRESH TABLE METADATA;
Query OK, 0 rows affected (0.18 sec)

mysql> select * from t_order;
+-----+----------+---------+--------+----------------------+----------+------+
| id  | order_id | user_id | status | t_unsigned_mediumint | username | pwd  |
+-----+----------+---------+--------+----------------------+----------+------+
|  66 |       66 |      66 | fff    |             16777215 | 6        | 6    |
|   2 |        2 |       2 | OK     |                    0 | 2        | 2    |
|   4 |        4 |       4 | OK     |                   25 | 4        | 4    |
| 100 |      100 |     100 | udpate |                  111 | 7        | 7    |
|   1 |        1 |       1 | newnew |                  123 | 1        | 1    |
|   3 |        3 |       3 | NO     |                   34 | 3        | 3    |
|   5 |        5 |       5 | OK     |                   89 | 5        | 5    |
| 101 |      101 |     101 | update |                   99 | 8        | 8    |
+-----+----------+---------+--------+----------------------+----------+------+
8 rows in set (0.04 sec)
```

数据迁移实操展示

刷新元数据及数据验证

```
mysql> REFRESH TABLE METADATA;
Query OK, 0 rows affected (0.12 sec)

mysql> select count(*) from t_order;
+----------+
| count(*) |
+----------+
|      500 |
+----------+
1 row in set (0.02 sec)

mysql> insert into t_order(order_id,user_id,status) values(9999999, 1, "just test");
Query OK, 1 row affected (0.02 sec)

mysql> select count(*) from t_order;
+----------+
| count(*) |
+----------+
|      501 |
+----------+
1 row in set (0.02 sec)

mysql> select * from t_order where order_id = 9999999;
+----------+---------+-----------+-------------+------------+-----------+--------------+---------------------+--------------------+--------------------+---------+----------+-----------+-------------+
----------+---------+---------+--------+-------+----------+-------------+--------+-------------+--------+--------+-------------+--------+-------+---------+
| order_id | user_id | status    | t_mediumint | t_smallint | t_tinyint | t_unsigned_int | t_unsigned_mediumint | t_unsigned_smallint | t_unsigned_tinyint | t_float | t_double | t_decimal | t_timestamp |
t_datetime | t_date | t_time | t_year | t_bit | t_binary | t_varbinary | t_blob | t_mediumblob | t_char | t_text | t_mediumtext | t_enum | t_set | t_json |
+----------+---------+-----------+-------------+------------+-----------+--------------+---------------------+--------------------+--------------------+---------+----------+-----------+-------------+
----------+---------+---------+--------+-------+----------+-------------+--------+-------------+--------+--------+-------------+--------+-------+---------+
|  9999999 |       1 | just test |        NULL |       NULL |      NULL |           NULL |                NULL |               NULL |               NULL |    NULL |     NULL |      NULL | NULL        |
NULL       | NULL    | NULL    | NULL   | NULL  | NULL     | NULL        | NULL   | NULL        | NULL   | NULL   | NULL        | NULL   | NULL  | NULL    |
+----------+---------+-----------+-------------+------------+-----------+--------------+---------------------+--------------------+--------------------+---------+----------+-----------+-------------+
----------+---------+---------+--------+-------+----------+-------------+--------+-------------+--------+--------+-------------+--------+-------+---------+
1 row in set (0.05 sec)
```

欢迎关注我们！

技术干货

加入交流群

SphereEx 官网：https://sphere-ex.com

Apache ShardingSphere Website：https://shardingsphere.apache.org

Apache ShardingSphere GitHub： https://github.com/apache/shardingsphere

Apache ShardingSphere Slack Channel：https://apacheshardingsphere.slack.com

ShardingSphere

谢谢观看