4-8 ShardingJDBC-5.3 实现读写分离

主从的架构搭建步骤

基于 Docker 去创建 MySQL 的主从架构：

```
Bash
# 创建主从数据库文件夹
mkdir -p /usr/local/mysql/master1/conf
mkdir -p /usr/local/mysql/master1/data
mkdir -p /usr/local/mysql/slave1/conf
mkdir -p /usr/local/mysql/slave1/data

# 初始化主数据库配置文件
cd /usr/local/mysql/master1/conf
vi my.cnf

# 粘贴以下内容
[mysqld]
datadir = /usr/local/mysql/master1/data
character-set-server = utf8
lower-case-table-names = 1

# 主从复制-主机配置# 主服务器唯一 ID
server-id = 1
# 启用二进制日志
log-bin=mysql-bin
# 设置 logbin 格式
binlog_format = STATEMENT

# 初始化从数据库配置文件
cd /usr/local/mysql/slave1/conf
vi my.cnf

# 粘贴以下内容
[mysqld]
datadir = /usr/local/mysql/slave1/data
character-set-server = utf8
lower-case-table-names = 1

# 主从复制-从机配置# 从服务器唯一 ID
```

```
server-id = 2
# 启用中继日志
relay-log = mysql-relay

# 文件夹授权
chmod -R 777 /usr/local/mysql
```

Docker 部署 Mysql8.0

```
Ruby
Copy
# 拉取镜像
docker pull mysql:8.0# 查看镜像
docker images

# 构建主数据库容器
docker run --name=mysql-master-1 \
--privileged=true \
-p 8808:3306 \
-v /usr/local/mysql/master1/data/:/var/lib/mysql \
-v /usr/local/mysql/master1/conf/my.cnf:/etc/mysql/my.cnf \
-v /usr/local/mysql/master1/mysql-files/:/var/lib/mysql-files/ \
-e MYSQL_ROOT_PASSWORD=root \
-d mysql:8.0 --lower_case_table_names=1

docker ps

# 验证是否可以登录# 交互式进入容器
docker exec -it mysql-master-1 /bin/bash

# 登录（使用构建时指定的密码：qiyu_10981）
mysql -uroot -p

# 退出
quit
exit

# 构建从数据库容器
docker run --name=mysql-slave-1 \
--privileged=true \
-p 8809:3306 \
-v /usr/local/mysql/slave1/data/:/var/lib/mysql \
-v /usr/local/mysql/slave1/conf/my.cnf:/etc/mysql/my.cnf \
-v /usr/local/mysql/slave1/mysql-files/:/var/lib/mysql-files/ \
```

```
-e  MYSQL_ROOT_PASSWORD=root \
-d mysql:8.0 --lower_case_table_names=1
```

编写主数据库的复制配置文件

```
C#

- 主数据库创建用户 slave 并授权
# 创建用户,设置主从同步的账户名
create user 'qiyu-slave'@'%' identified with mysql_native_password
by 'qiyu-81710181';

# 授权
grant replication slave on *.* to 'qiyu-slave'@'%';

# 刷新权限
flush privileges;

# 查询 server_id 值
show variables like 'server_id';

# 也可临时（重启后失效）指定 server_id 的值（主从数据库的 server_id 不能
相同）
set global server_id = 1;

# 查询 Master 状态，并记录 File 和 Position 的值，这两个值用于和下边的从数
据库中的 change 那条 sql 中
的 master_log_file，master_log_pos 参数对齐使用
show master status;

# 重置下 master 的 binlog 位点
reset master;
```

编写从数据库的复制配置文件

```
Java
# 进入从数据库
# 注意：执行完此步骤后退出主数据库，防止再次操作导致 File 和 Position 的值
发生变化# 验证 slave 用户是否可用

# 查询 server_id 值
show variables like 'server_id';
```

```
# 也可临时（重启后失效）指定 server_id 的值（主从数据库的 server_id 不能相同）
set global server_id = 2;

# 若之前设置过同步，请先重置
stop slave;
reset slave;

# 设置主数据库
change master to master_host='主机的ip',master_port=8808,master_user='qiyu-slave',master_password='qiyu-pwd',master_log_file='mysql-bin.000001',master_log_pos=156;

# 开始同步
start slave;

# 若出现错误，则停止同步，重置后再次启动
stop slave;
reset slave;
start slave;

# 查询 Slave 状态
show slave status;
```

最后需要查看是否配置成功# 查看参数 Slave_IO_Running 和 Slave_SQL_Running 是否都为 yes，则证明配置成功。若为 no，则需要查看对应的 Last_IO_Error 或 Last_SQL_Error 的异常值。

注意点

这里要注意，如果 mysql 的同步出现了问题，如果不担心 binlog 丢失的话，其实可以尝试重制 master 的 binlog 信息，使用 reset master 命令去清空之前的 binlog 日志即可，注意正在写入的 binlog 不要随意清空，很危险。

ShardingJDBC 配置介绍

以下配置只适用于 shardingjdbc 的 5.3.2 版本

application.yml 配置中设置连接池的全局属性：

```Java
spring:
  datasource:
    driver-class-name:
org.apache.shardingsphere.driver.ShardingSphereDriver
    url: jdbc:shardingsphere:classpath:qiyu-db-sharding.yaml
    hikari:
      pool-name: qiyu-user-pool
      minimum-idle: 150
      maximum-pool-size: 300
      connection-init-sql: select 1
      connection-timeout: 4000
      max-lifetime: 60000
```

ShardingJDBC 的读写分离配置：

```Java
dataSources:
  user_master:  ##新表，重建的分表
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driver-class-name: com.mysql.cj.jdbc.Driver
    jdbcUrl: jdbc:mysql://cloud.db:8809/qiyu_live_user?
useUnicode=true&characterEncoding=utf8
    username: root
    password: root

  user_slave0:  ##新表，重建的分表
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driver-class-name: com.mysql.cj.jdbc.Driver
    jdbcUrl: jdbc:mysql://cloud.db:3307/qiyu_live_user?
useUnicode=true&characterEncoding=utf8
    username: root
    password: qiyu-71623@11L

rules:
    - !READWRITE_SPLITTING
      dataSources:
        user_ds:
         staticStrategy:
           writeDataSourceName: user_master
           readDataSourceNames:
              - user_slave0
    - !SINGLE
```

```
      defaultDataSource: user_ds ## 不分表分分库的默认数据源
    - !SHARDING
      tables:
        t_user:
          actualDataNodes: user_ds.t_user_${(0..99).collect()
{it.toString().padLeft(2,'0')}}
          tableStrategy:
            standard:
              shardingColumn: user_id
              shardingAlgorithmName: t_user-inline
      shardingAlgorithms:
        t_user-inline:
          type: INLINE
          props:
            algorithm-expression: t_user_${(user_id %
100).toString().padLeft(2,'0')}
props:
  sql-show: true
```

实现读写分离配置之后，再来验证下各个接口的请求清空