Apache ShardingSphere

一 基本概念

1 什么是 Sharding Sphere

Apache ShardingSphere(Incubator) 是一套开源的分布式数据库中间件解决方案组成的生态圈，它由 Sharding-JDBC、Sharding-Proxy 和 Sharding-Sidecar（规划中）这 3 款相互独立，却又能够混合部署配合使用的产品组成。它们均提供标准化的数据分片、分布式事务和数据库治理功能，可适用于如 Java 同构、异构语言、云原生等各种多样化的应用场景。

ShardingSphere 定位为关系型数据库中间件，旨在充分合理地在分布式的场景下利用关系型数据库的计算和存储能力，而并非实现一个全新的关系型数据库。它通过关注不变，进而抓住事物本质。关系型数据库当今依然占有巨大市场，是各个公司核心业务的基石，未来也难于撼动，我们目前阶段更加关注在原有基础上的增量，而非颠覆。

Apache 官方发布从 4.0.0 版本开始。

2 什么是分库分表

数据库中的数据量不一定是可控的，在未进行分库分表的情况下，随着时间和业务的发展，库中的表会越来越多，表中的数据量也会越来越大，相应地，数据操作，增删改查的开销也会越来越大；另外，由于无法进行分布式式部署，而一台服务器的资源（CPU、磁盘、内存、IO 等）是有限的，最终数据库所能承载的数据量、数据处理能力都将遭遇瓶颈。。

分库分表就是为了解决由于数据量过大而导致数据库性能降低的问题，将原来独立的数据库拆分成若干数据库组成，将数据大表拆分成若干数据表组成，使得单一数据库、单一数据表的数据量变小，从而达到提升数据库性能的目的。

2.1 分库分表的方式

数据库的切分指的是通过某种特定的条件，将我们存放在同一个数据库中的数据分散存放到多个数据库（主机）中，以达到分散单台设备负载的效果，即分库分表。

数据的切分根据其切分规则的类型，可以分为 垂直切分 和水平切分。

（1）垂直切分： 把单一的表拆分成多个表，并分散到不同的数据库（主机）上。

（2）水平切分：根据表中数据的逻辑关系，将表中的数据按照某种条件拆分到多台数据库上。

2.2 垂直切分

一个数据库有多个表构成，每个表对应不同的业务，垂直切分是只按照业务将表进行分类，将其分布到不同的数据库上，这样就将数据分担到了不同的库上（专库专用）。

例如：

有如下几张表：

- 用户信息表（User）
- 课程信息表（Courses）
- 订单信息表（Orders）
- 针对以上案例，垂直切分就是根据每个表的不同业务进行切分。
- 比如 User 表，Courses 表和 Orders 表，将每个表切分到不同的数据库上。

垂直切分的优点如下：

（1）拆分后业务清晰，系统之间进行整合或扩展很容易。

（2）按照成本、应用的等级、应用的类型等奖表放到不同的机器上，便于管理，数据维护简单。

垂直切分的缺点如下：

（1）部分业务表无法关联(Join)，只能通过接口方式解决，提高了系统的复杂度。

（2）受每种业务的不同限制，存在单库性能瓶颈，不易进行数据扩展和提升性能。

（3）事务处理变得复杂。

2.3 水平切分

与垂直切分对比，水平切分不是将表进行分类，而是将其按照某个字段的某种规则分散到多个库中，在每个表中包含一部分数据，所有表加起来就是全量的数据。

简单来说，我们可以将对数据的水平切分理解为按照数据行进行切分，就是将表中的某些行切分到一个数据库表中，而将其他行切分到其他数据库表中。

这种切分方式根据单表的数据量的规模来切分，保证单表的容量不会太大，从而保证了单表的查询等处理能力

例如将用户的信息表拆分成 User1、User2 等，表结构是完全一样的。我们通常根据某些特定的规则来划分表，比如根据用户的 ID 来取模划分。

举例：

在博客类系统中，读取量一般都会很大。当同时有 100 万个用户在浏览时，如果是单表，则单表会进行 100 万次请求，如果是单库，数据库就会承受 100 万次的请求压力。

如果采取水平切分来减少每个单表的压力，将其分为 100 个表，并且分布在 10 个数据库中，每个表进行 1 万次请求，则每个数据库会承受 10 万次的请求压力，虽然这不可能绝对平均，但是这样，压力就减少了很多，并且是成倍减少的。

水平切分的优点：

（1）单库单表的数据保持在一定的量级，有助于性能的提高。

（2）切分的表的结构相同，应用层改造较少，只需要增加路由规则即可。

（3）提高了系统的稳定性和负载能力。

水平切分的缺点如下：

（1）切分后，数据是分散的，很难利用数据库的 Join 操作，跨库 Join 性能较差。

（2）分片事务的一致性难以解决，数据扩容的难度和维护量极大。

2.4 分库分表带来的问题

综上所述，垂直切分和水平切分的共同点如下：

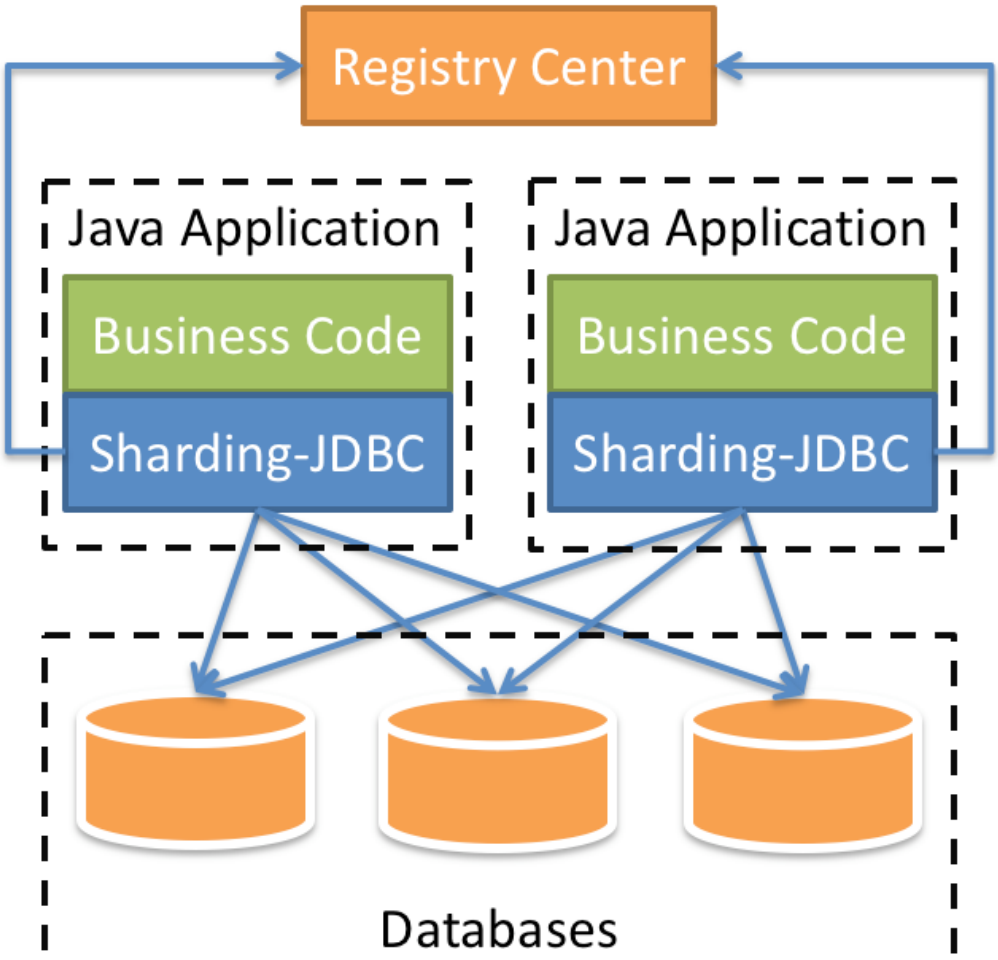
- 存在跨节点 Join 的问题。
- 存在跨节点合并排序、分页的问题。
- 存在多数据源管理的问题。

二 Sharding-JDBC

1 简介

Sharding-JDBC 是当当网研发的开源分布式数据库中间件，从 3.0 开始 Sharding-JDBC 被包含在 Sharding-Sphere 中，之后该项目进入进入 Apache 孵化器，4.0 版本之后的版本为 Apache 版本。

Sharding-JDBC 是 ShardingSphere 的第一个产品，也是 ShardingSphere 的前身。 它定位为轻量级 Java 框架，在 Java 的 JDBC 层提供的额外服务。它使用客户端直连数据库，以 jar 包形式提供服务，无需额外部署和依赖，可理解为增强版的 JDBC 驱动，完全兼容 JDBC 和各种 ORM 框架。



Sharding-JDBC的核心功能为数据分片和读写分离，通过Sharding-JDBC，应用可以透明的使用jdbc访问已经分库分表、读写分离的多个数据源，而不用关心数据源的数量以及数据如何分布。

- 适用于任何基于 JDBC 的 ORM 框架，如：JPA, Hibernate, Mybatis, Spring JDBC Template 或直接使用 JDBC。
- 支持任何第三方的数据库连接池，如：DBCP, C3P0, BoneCP, Druid, HikariCP 等。
- 支持任意实现 JDBC 规范的数据库。目前支持 MySQL，Oracle，SQLServer，PostgreSQL 以及任何遵循 SQL92 标准的数据库。

2 快速入门（水平分表）

2.1 需求说明

使用Sharding-JDBC完成对订单表的水平分表，通过快速入门程序的开发，快速体验Sharding-JDBC的使用
方法。
创建两张结构相同的表，atguigu_order_1和atguigu _order_2，这两张表是订单表拆分后的表，通过Sharding-Jdbc向订单表插入数据，
按照一定的分片规则，主键为偶数的进入atguigu _order_1，另一部分数据进入atguigu _order_2，通过Sharding-Jdbc 查询数
据，根据 SQL 语句的内容从 atguigu _order_1 或 atguigu _order_2 查询数据。

2.2 创建数据库

在atguigu _db中创建atguigu _order_1、atguigu _order_2表

```
CREATE DATABASE ` atguigu _db` CHARACTER SET 'utf8' COLLATE 'utf8_general_ci';

DROP TABLE IF EXISTS ` atguigu _order_1`;
CREATE TABLE ` atguigu _order_1` (
`order_id` bigint(20) NOT NULL COMMENT '订单id',
`price` decimal(10, 2) NOT NULL COMMENT '订单价格',
`user_id` bigint(20) NOT NULL COMMENT '下单用户id',
`status` varchar(50) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL
COMMENT '订单状态',
PRIMARY KEY (`order_id`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT =
Dynamic;
DROP TABLE IF EXISTS ` atguigu _order_2`;
CREATE TABLE ` atguigu _order_2` (
`order_id` bigint(20) NOT NULL COMMENT '订单id',
`price` decimal(10, 2) NOT NULL COMMENT '订单价格',
`user_id` bigint(20) NOT NULL COMMENT '下单用户id',
`status` varchar(50) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL
COMMENT '订单状态',
PRIMARY KEY (`order_id`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT =
Dynamic;
```

2.3 创建 SpringBoot 工程引入 maven 依赖

```
<dependency>
 <groupId>org.apache.shardingsphere</groupId>
 <artifactId>sharding-jdbc-spring-boot-starter</artifactId>
 <version>4.0.0-RC1</version>
</dependency>
```

2.4 创建配置文件编写分片规则

```
server.port=56081
spring.application.name = sharding-jdbc-simple-demo

server.servlet.context-path = /sharding-jdbc-simple-demo
spring.http.encoding.enabled = true
spring.http.encoding.charset = UTF-8
spring.http.encoding.force = true

spring.main.allow-bean-definition-overriding = true
mybatis.configuration.map-underscore-to-camel-case = true
# 以下是分片规则配置
# 定义数据源
spring.shardingsphere.datasource.names = m1
spring.shardingsphere.datasource.m1.type                                    =
com.alibaba.druid.pool.DruidDataSource
spring.shardingsphere.datasource.m1.driver-class-name = com.mysql.jdbc.Driver
spring.shardingsphere.datasource.m1.url  =  jdbc:mysql://localhost:3306/atguigu
_db?useUnicode=true
spring.shardingsphere.datasource.m1.username = root
spring.shardingsphere.datasource.m1.password = root
# 指定t_order表的数据分布情况，配置数据节点
spring.shardingsphere.sharding.tables.t_order.actual-data-nodes = m1. atguigu
_order_$->{1..2}
# 指定t_order表的主键生成策略为SNOWFLAKE
spring.shardingsphere.sharding.tables.t_order.key-generator.column=order_id
spring.shardingsphere.sharding.tables.t_order.key-generator.type=SNOWFLAKE
# 指定t_order表的分片策略，分片策略包括分片键和分片算法
spring.shardingsphere.sharding.tables.t_order.table-strategy.inline.sharding-
column = order_id
spring.shardingsphere.sharding.tables.t_order.table-strategy.inline.algorithm
-expression =
atguigu _order_$->{order_id % 2 + 1}
# 打开sql输出日志
spring.shardingsphere.props.sql.show = true
swagger.enable = true
logging.level.root = info
logging.level.org.springframework.web = info
logging.level.com.itheima.dbsharding = debug
logging.level.druid.sql = debug
```

1.首先定义数据源m1，并对m1进行实际的参数配置。
2.指定atguigu _order表的数据分布情况，他分布在m1. atguigu _order_1，m1. atguigu _order_2
3.指定atguigu _order表的主键生成策略为SNOWFLAKE，SNOWFLAKE是一种分布式自增算法，保证id全局唯一
4.定义atguigu _order分片策略，order_id为偶数的数据落在atguigu _order_1，为奇数的落在atguigu _order_2，分表策略的表达式为
atguigu _order_$->{order_id % 2 + 1}

2.5 编写程序，循环向表里面添加数据，查看最终的效果

2.6 执行流程分析

（1）解析sql，获取片键值，在本例中是order_id

（2）Sharding-JDBC通过规则配置 atguigu_order_$->{order_id % 2 + 1}，知道了当order_id为偶数时，应该往atguigu_order_1表插数据，为奇数时，往t_order_2插数据。

（3）于是Sharding-JDBC根据order_id的值改写sql语句，改写后的SQL语句是真实所要执行的SQL语句。

（4）执行改写后的真实sql语句

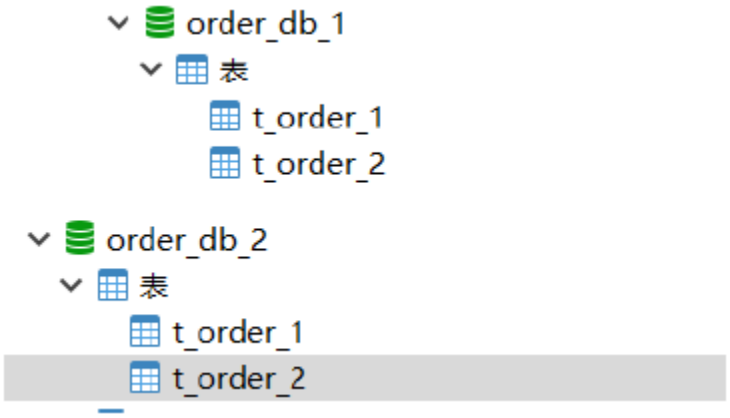
（5）将所有真正执行 sql 的结果进行汇总合并，返回。

2.6 其他整合方式介绍

3 水平分库

水平分库是把同一个表的数据按一定规则拆到不同的数据库中，每个库可以放在不同的服务器上。接下来看一下如何使用Sharding-JDBC实现水平分库，咱们继续对快速入门中的例子进行完善。

(1)将原有数据库拆分为order_db_1、order_db_2



(2)分片规则修改

由于数据库拆分了两个，这里需要配置两个数据源。

分库需要配置分库的策略，和分表策略的意义类似，通过分库策略实现数据操作针对分库的数据库进行操作。

```
# 定义多个数据源
spring.shardingsphere.datasource.names = m1,m2
spring.shardingsphere.datasource.m1.type = com.alibaba.druid.pool.DruidDataSource
spring.shardingsphere.datasource.m1.driver-class-name = com.mysql.jdbc.Driver
spring.shardingsphere.datasource.m1.url =
jdbc:mysql://localhost:3306/order_db_1?useUnicode=true
spring.shardingsphere.datasource.m1.username = root
spring.shardingsphere.datasource.m1.password = root
spring.shardingsphere.datasource.m2.type = com.alibaba.druid.pool.DruidDataSource
spring.shardingsphere.datasource.m2.driver-class-name = com.mysql.jdbc.Driver
spring.shardingsphere.datasource.m2.url =
```

```
jdbc:mysql://localhost:3306/order_db_2?useUnicode=true
spring.shardingsphere.datasource.m2.username = root
spring.shardingsphere.datasource.m2.password = root
...
# 分库策略，以user_id为分片键，分片策略为user_id % 2 + 1，user_id为偶数操作m1数据源，否则操作m2。
spring.shardingsphere.sharding.tables.t_order.database-strategy.inline.sharding-column = user_id
spring.shardingsphere.sharding.
```

分库策略定义方式解析：

```
#分库策略，如何将一个逻辑表映射到多个数据源
spring.shardingsphere.sharding.tables.<逻辑表名称>.database-strategy.<分片策略>.<分片策略属性名>= #
分片策略属性值
#分表策略，如何将一个逻辑表映射为多个实际表
spring.shardingsphere.
```

Sharding-JDBC支持以下几种分片策略：

不管理分库还是分表，策略基本一样。

standard：标准分片策略，对应StandardShardingStrategy。提供对SQL语句中的=, IN和BETWEEN AND的

分片操作支持。StandardShardingStrategy只支持单分片键，提供PreciseShardingAlgorithm和

RangeShardingAlgorithm两个分片算法。PreciseShardingAlgorithm是必选的，用于处理=和IN的分片。

RangeShardingAlgorithm是可选的，用于处理BETWEEN AND分片，如果不配置RangeShardingAlgorithm，SQL中的BETWEEN AND将按照全库路由处理。

complex：符合分片策略，对应ComplexShardingStrategy。复合分片策略。提供对SQL语句中的=, IN和

BETWEEN AND的分片操作支持。ComplexShardingStrategy支持多分片键，由于多分片键之间的关系复

杂，因此并未进行过多的封装，而是直接将分片键值组合以及分片操作符透传至分片算法，完全由应用开发

者实现，提供最大的灵活度。

inline：行表达式分片策略，对应InlineShardingStrategy。使用Groovy的表达式，提供对SQL语句中的=和

IN的分片操作支持，只支持单分片键。对于简单的分片算法，可以通过简单的配置使用，从而避免繁琐的Java

代码开发，如: `t_user_$->{u_id % 8}` 表示t_user表根据u_id模8，而分成8张表，表名称为 `t_user_0` 到

`t_user_7` 。

hint：Hint分片策略，对应HintShardingStrategy。通过Hint而非SQL解析的方式分片的策略。对于分片字段

非SQL决定，而由其他外置条件决定的场景，可使用SQL Hint灵活的注入分片字段。例：内部系统，按照员工

登录主键分库，而数据库中并无此字段。SQL Hint支持通过Java API和SQL注释(待实现)两种方式使用。

none：不分片策略，对应 NoneShardingStrategy。不分片的策略。

5 垂直分库

垂直分库是指按照业务将表进行分类，分布到不同的数据库上面，每个库可以放在不同的服务器上，它的核心理念是专库专用。接下来看一下如何使用Sharding-JDBC实现垂直分库。

⑴创建数据库

创建数据库 user_db 和表 t_user

```
CREATE DATABASE `user_db` CHARACTER SET 'utf8' COLLATE 'utf8_general_ci';

DROP TABLE IF EXISTS `t_user`;
CREATE TABLE `t_user` (
`user_id` bigint(20) NOT NULL COMMENT '用户id',
`fullname` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL COMMENT
'用户姓名',
`user_type` char(1) DEFAULT NULL COMMENT '用户类型',
PRIMARY KEY (`user_id`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT =

Dynamic;
```

⑵在 Sharding-JDBC 规则中修改

```
# 新增m0数据源，对应user_db
spring.shardingsphere.datasource.names = m0,m1,m2
...
spring.shardingsphere.datasource.m0.type = com.alibaba.druid.pool.DruidDataSource
spring.shardingsphere.datasource.m0.driver-class-name = com.mysql.jdbc.Driver

spring.shardingsphere.datasource.m0.url                                          =
jdbc:mysql://localhost:3306/user_db?useUnicode=true
spring.shardingsphere.datasource.m0.username = root
spring.shardingsphere.datasource.m0.password = root
....
# t_user分表策略，固定分配至m0的t_user真实表
spring.shardingsphere.sharding.tables.t_user.actual-data-nodes = m$->{0}.t_user
spring.shardingsphere.sharding.tables.t_user.table-strategy.inline.sharding-column
= user_id
spring.shardingsphere.sharding.tables.t_user.table-strategy.inline.algorithm-expres

sion = t_user
```

6 公共表

公共表属于系统中数据量较小，变动少，而且属于高频联合查询的依赖表。参数表、数据字典表等属于此类型。可以将这类表在每个数据库都保存一份，所有更新操作都同时发送到所有分库执行。接下来看一下如何使用Sharding-JDBC实现公共表。

⑴创建数据库

分别在 user_db、order_db_1、order_db_2 中创建 t_dict 表：

```
CREATE TABLE `t_dict` (
`dict_id` bigint(20) NOT NULL COMMENT '字典id',
`type` varchar(50) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL COMMENT '字典
类型',
`code` varchar(50) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL COMMENT '字典
```

```
编码',
`value` varchar(50) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL COMMENT '字典值',
PRIMARY KEY (`dict_id`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;
```

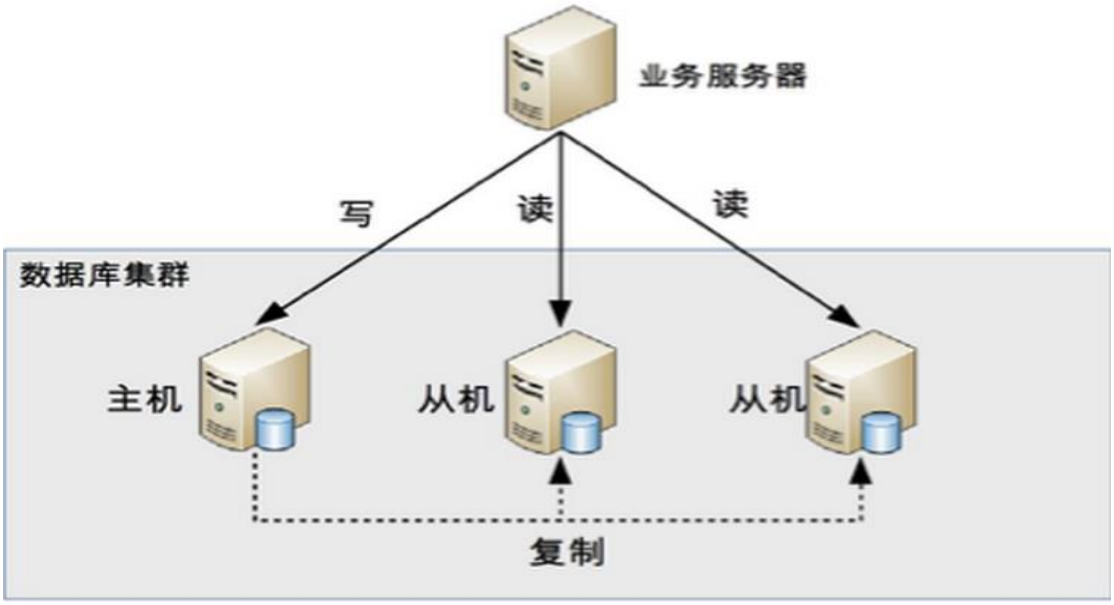
(2)在 Sharding-JDBC 规则中修改

```
# 指定t_dict为公共表
spring.shardingsphere.sharding.broadcast-tables=t_dict
```

7 读写分离

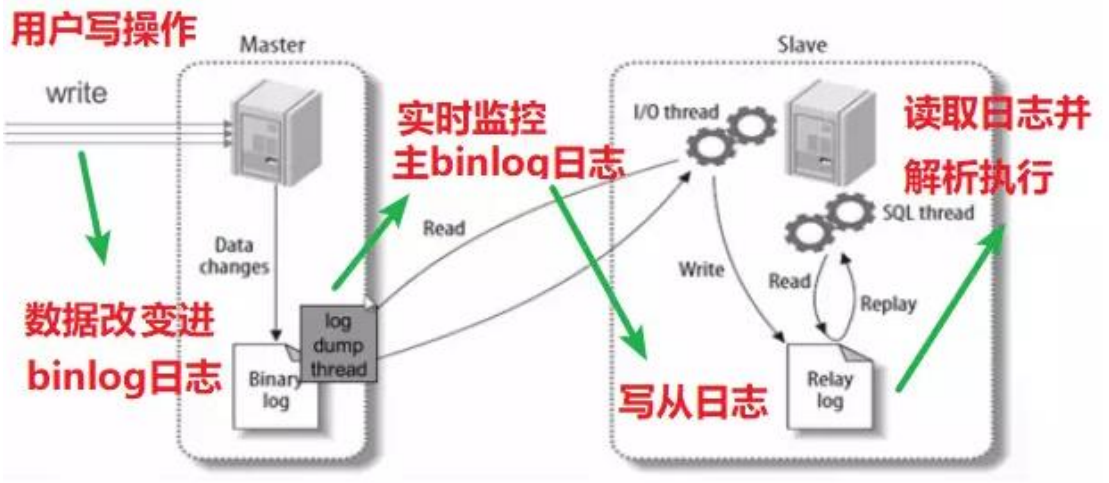
为了确保数据库产品的稳定性，很多数据库拥有双机热备功能。也就是，第一台数据库服务器，是对外提供增删改业务的生产服务器；第二台数据库服务器，主要进行读的操作。

原理：让主数据库（master）处理事务性增、改、删操作，而从数据库（slave）处理SELECT查询操作。

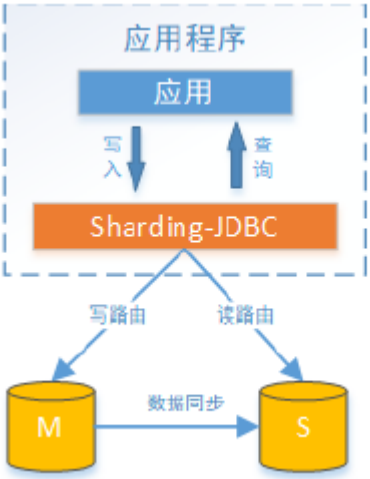


主从复制：**当主服务器有写入（insert/update/delete）语句时候，从服务器自动获取**

读写分离：**insert/update/delete语句操作一台服务器，select操作另一个服务器**

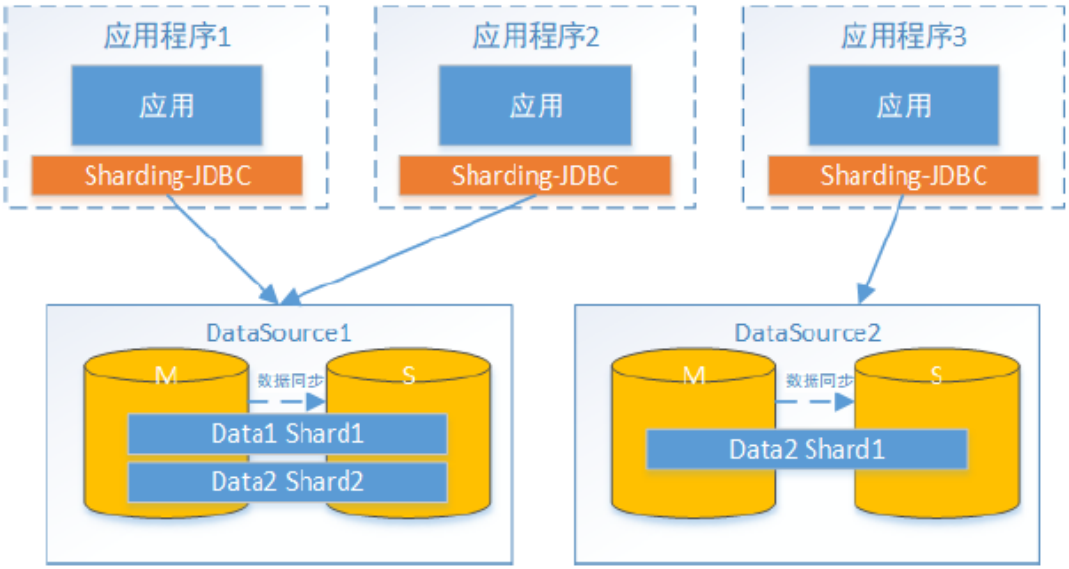


Sharding-JDBC读写分离则是根据SQL语义的分析，将读操作和写操作分别路由至主库与从库。它提供透明化读写分离，让使用方尽量像使用一个数据库一样使用主从数据库集群。



Sharding-JDBC提供一主多从的读写分离配置，可独立使用，也可配合分库分表使用，同一线程且同一数据库连接内，如有写入操作，以后的读操作均从主库读取，用于保证数据一致性。

Sharding-JDBC不提供主从数据库的数据同步功能，需要采用其他机制支持。



（1）新增 mysql 实例

复制原有 mysql 如：D:\mysql-5.7.25(作为主库) -> D:\mysql-5.7.25-s1(作为从库)，并修改以下从库的 my.ini：

```
[mysqld]
#设置3307端口
port = 3307
# 设置mysql的安装目录
basedir=D:\mysql-5.7.25-s1
# 设置mysql数据库的数据的存放目录
datadir=D:\mysql-5.7.25-s1\data
```

然后将从库安装为 windows 服务，注意配置文件位置：

```
D:\mysql-5.7.25-s1\bin>mysqld install mysqls1
 --defaults-file="D:\mysql-5.7.25-s1\my.ini"
```

由于从库是从主库复制过来的，因此里面的数据完全一致，可使用原来的账号、密码登录。

（2）修改主、从库的配置文件(my.ini)，新增内容如下：

主库

```
[mysqld]
#开启日志
log-bin = mysql-bin
#设置服务id，主从不能一致
server-id = 1
#设置需要同步的数据库
binlog-do-db=user_db
#屏蔽系统库同步
```

```
binlog-ignore-db=mysql
binlog-ignore-db=information_schema
binlog-ignore-db=performance_schema
```

从库

```
[mysqld]
#开启日志
log-bin = mysql-bin
#设置服务id，主从不能一致
server-id = 2
#设置需要同步的数据库
replicate_wild_do_table=user_db.%
#屏蔽系统库同步
replicate_wild_ignore_table=mysql.%
replicate_wild_ignore_table=information_schema.%
replicate_wild_ignore_table=performance_schema.%
```

重启主库和从库

（3）授权主从复制专用账号

```
#切换至主库bin目录，登录主库
mysql -h localhost -uroot -p
#授权主备复制专用账号
GRANT REPLICATION SLAVE ON *.* TO 'db_sync'@'%' IDENTIFIED BY 'db_sync';
#刷新权限
FLUSH PRIVILEGES;
#确认位点 记录下文件名以及位点
show master status;
```

（4）设置从库向主库同步数据、并检查连接

```
#切换至从库bin目录，登录从库
mysql -h localhost -P3307 -uroot -p
#先停止同步
STOP SLAVE;
#修改从库指向到主库，使用上一步记录的文件名以及位点
CHANGE MASTER TO
master_host = 'localhost',
master_user = 'db_sync',
master_password = 'db_sync',
master_log_file = 'mysql-bin.000002',
master_log_pos = 154;
#启动同步
START SLAVE;
#查看从库状态Slave_IO_Runing和Slave_SQL_Runing都为Yes说明同步成功，如果不为Yes，请检查
error_log，然后
排查相关异常。
show slave status\G
#注意 如果之前此备库已有主库指向 需要先执行以下命令清空
STOP SLAVE IO_THREAD FOR CHANNEL '';
reset slave all;
```

（5）实现 sharding-jdbc 读写分离

```
# 增加数据源s0，使用上面主从同步配置的从库。
```

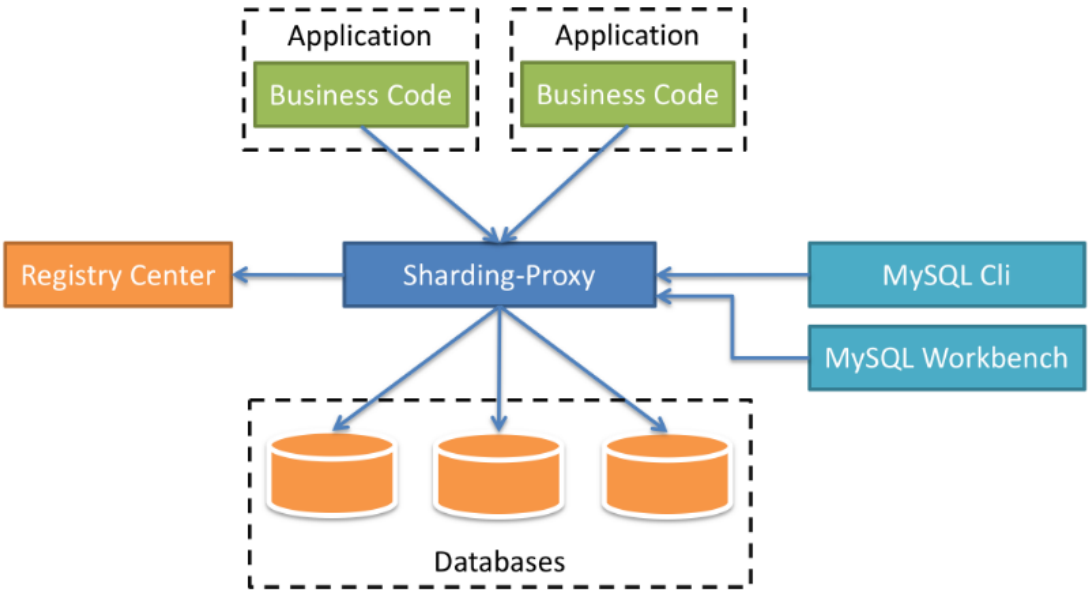
```
spring.shardingsphere.datasource.names = m0,m1,m2,s0
...
spring.shardingsphere.datasource.s0.type = com.alibaba.druid.pool.DruidDataSource
spring.shardingsphere.datasource.s0.driver-class-name = com.mysql.jdbc.Driver
spring.shardingsphere.datasource.s0.url                                          =
jdbc:mysql://localhost:3307/user_db?useUnicode=true
spring.shardingsphere.datasource.s0.username = root
spring.shardingsphere.datasource.s0.password = root
....
# 主库从库逻辑数据源定义 ds0为user_db
spring.shardingsphere.sharding.master-slave-rules.ds0.master-data-source-name=m0
spring.shardingsphere.sharding.master-slave-rules.ds0.slave-data-source-names=s0
# t_user分表策略，固定分配至ds0的t_user真实表
spring.shardingsphere.sharding.tables.t_user.actual-data-nodes = ds0.t_user
....
```

三 Sharding-Proxy

1 简介

Sharding-Proxy 是 ShardingSphere 的第二个产品。 它定位为透明化的数据库代理端，提供封装了数据库二进制协议的服务端版本，用于完成对异构语言的支持。 目前先提供 MySQL/PostgreSQL 版本，它可以使用任何兼容 MySQL/PostgreSQL 协议的访问客户端(如：MySQL Command Client, MySQL Workbench, Navicat 等)操作数据，对 DBA 更加友好。

- 向应用程序完全透明，可直接当做 MySQL/PostgreSQL 使用。
- 适用于任何兼容 MySQL/PostgreSQL 协议的的客户端



2 安装

下载 Sharding-Proxy 的最新发行版，地址:https://github.com/sharding-sphere/sharding-sphere-doc/raw/master/dist/sharding-proxy-3.0.0.tar.gz

上传服务器，解压，进入 conf 目录，有 2 个重要的配置文件：server.yaml 和 config-sharding.yaml

3 配置

（1）配置 server.yaml，把下面的配置的 # 注释打开，改为符合自己的配置

```
authentication:

username: root

password: 123456

props:

#max.connections.size.per.query: 1

acceptor.size: 8 # 用于设置接收客户端请求的工作线程个数，默认为 CPU 核数*2

executor.size: 4 # 工作线程数量，默认值: CPU 核数

proxy.transaction.type: XA

proxy.transaction.enabled: true # 是否开启事务, 目前仅支持 XA 事务，默认为不开启

proxy.opentracing.enabled: false # 是否开启链路追踪功能，默认为不开启

sql.show: true # 是否开启 SQL 显示，默认值: false
```

（2）配置分表规则， config-sharding.yaml，把最下面 mysql 的注释改为符合自己的配置

```
# 应用连接 Sharding-Proxy 的数据库名称
schemaName: sharding_db

dataSources:
  ds0:
    url:              jdbc:mysql://localhost:3306/ds0?useUnicode=true&characterEncoding=UTF-8&useSSL=false
    username: root
    password: 123456
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 30
  ds1:
    url:              jdbc:mysql://localhost:3306/ds1?useUnicode=true&characterEncoding=UTF-8&useSSL=false
    username: root
    password: 123456
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 30

shardingRule:
  tables:
    t_order:
      actualDataNodes: ds${0..1}.t_order${0..1}
```

```
    tableStrategy:
      inline:
        shardingColumn: order_id
        algorithmExpression: t_order${order_id % 2}
#     keyGenerator:
#       type: SNOWFLAKE
#       column: order_id
  t_order_item:
    actualDataNodes: ds${0..1}.t_order_item${0..1}
    tableStrategy:
      inline:
        shardingColumn: order_id
        algorithmExpression: t_order_item${order_id % 2}
#     keyGenerator:
#       type: SNOWFLAKE
#       column: order_item_id
 bindingTables:
   - t_order,t_order_item
 defaultDatabaseStrategy:
   inline:
     shardingColumn: member_id
     algorithmExpression: ds${member_id % 2}
 defaultTableStrategy:
   none:
```

（3）启动 Sharding-Proxy，如果是 windows 系统，双击 start.bat，启动后 sharding-proxy 目录会有日志文件 logs/stdout.log

使用 Mysql 客户端连接 Sharding-Proxy，然后创建表，该代理层会根据分库分表规则自动在后端对应的分库中创建表

3 SpringBoot 接入

（1）新建项目，引入依赖

pom 依赖导入

```
<!-- druid 连接池 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid-spring-boot-starter</artifactId>

<version>1.1.9</version>

</dependency>

<dependency>

<groupId>io.shardingsphere</groupId>

<artifactId>sharding-transaction-spring</artifactId>
```

```
<version>3.1.0</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-aop</artifactId>

</dependency>
```

（2）配置 sharding-proxy 相关参数。连接 proxy

```
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3307/sharding_db?useServerPrepStmts=true&cachePrepStmts=true

spring.datasource.username=root

spring.datasource.password=123456

#spring.datasource.druid.driver-class-name=com.mysql.cj.jdbc.Driver

#初始化时建立物理连接的个数

spring.datasource.druid.initial-size=3

#最小连接池数量

spring.datasource.druid.min-idle=3

#最大连接池数量

spring.datasource.druid.max-active=10

#获取连接时最大等待时间

spring.datasource.druid.max-wait=60000
```

（3） 最后，启动应用 springboot 应用即可。