一、功能说明

该工具包的作用是集成ShardingJdbc进行分库分表, 借助其自由的读写分离和分布式事务能力提供相应场景的支持。

背景

- 随着平台的活跃用户日益激增, 数据量累计日益庞大, 许许多多的查询/统计业务响应迟缓, 短则10s长则数十秒, 严重影响用户体验。
- Apache基金会孵化的ShardingSphere项目, 借助其完全兼容 JDBC 和各种 ORM 框架且0代码侵入的优势, 已经成为该场景解决方案的主流。
- 因此特编本文用于集成该方案提供分库分表甚至分布式事务的能力。

二、工具使用

- 添加依赖: pom.xml

```
<!-- 读写分离依赖包 -->
<dependency>
    <groupId>cn.gmlee</groupId>
    <artifactId>tools-shardingJdbc</artifactId>
    <version>3.0.2</version>
</dependency>
```

- 启用工具

> 在启动类上或其他配置类上扫描包: cn.gmlee.tools

```
@SpringBootApplication
@ComponentScan("cn.gmlee.tools")
public class XxxApp {
    public static void main(String[] args) {
        SpringApplication.run(XxxApp.class, args);
    }
}
```

- 添加配置: application.properties

```
# 多数据源配置 (支持读写分离)
spring.shardingsphere.datasource.names=master0,master0slave0,pos,site
spring.shardingsphere.datasource.master0.type=com.alibaba.druid.pool.DruidDataSource
spring.shardingsphere.datasource.master0.driver-class-name=com.mysql.cj.jdbc.Driver
spring.shardingsphere.datasource.master0.url=jdbc:mysql://172.19.193.244:3306/db_lala_pos_order?useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
spring.shardingsphere.datasource.master0.username=lala_pos_rw
spring.shardingsphere.datasource.master0.password=vLqUeUxC1mD7ZwPZlF_G
spring.shardingsphere.datasource.master0.validation-query=SELECT 1 FROM DUAL
spring.shardingsphere.datasource.master0.timeBetweenEvictionRunsMillis=60000
```

```
spring.shardingsphere.datasource.master0.minEvictableIdleTimeMillis=600000
spring.shardingsphere.datasource.master0.testOnBorrow=true

spring.shardingsphere.datasource.master0slave0.type=com.alibaba.druid.pool.DruidDataSource
spring.shardingsphere.datasource.master0slave0.driver-class-name=com.mysql.cj.jdbc.Driver
spring.shardingsphere.datasource.master0slave0.url=jdbc:mysql://172.19.193.244:3306/db_lala_pos_order?useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
spring.shardingsphere.datasource.master0slave0.username=lala_pos_rw
spring.shardingsphere.datasource.master0slave0.password=vLqUeUxC1mD7ZwPZlF_G
spring.shardingsphere.datasource.master0slave0.validation-query=SELECT 1 FROM DUAL
spring.shardingsphere.datasource.master0slave0.timeBetweenEvictionRunsMillis=60000
spring.shardingsphere.datasource.master0slave0.minEvictableIdleTimeMillis=600000
spring.shardingsphere.datasource.master0slave0.testOnBorrow=true

# 打印SQL
spring.shardingsphere.props.sql.show=true

# 分库分表主表策略配置 (是否分库分表由是否配置策略决定)
spring.shardingsphere.sharding.tables.t_pos_order_midway.actual-data-nodes=ds$->{0..0}.t_pos_order_midway$->{1..2}
spring.shardingsphere.sharding.tables.t_pos_order_midway.table-strategy.inline.sharding-column=order_id
spring.shardingsphere.sharding.tables.t_pos_order_midway.table-strategy.inline.algorithm-expression=t_pos_order_midway$->{order_id.toBigInteger() % 2 + 1}

# 分库分表主表策略配置 (是否分库分表由是否配置策略决定)
spring.shardingsphere.sharding.tables.t_pos_order_log.actual-data-nodes=ds$->{0..0}.t_pos_order_log$->{1..12}
spring.shardingsphere.sharding.tables.t_pos_order_log.table-strategy.inline.sharding-column=order_id
spring.shardingsphere.sharding.tables.t_pos_order_log.table-strategy.inline.algorithm-expression=t_pos_order_log$->{order_id.toBigInteger() % 12 + 1}

# 默认分库配置: 无分库策略的库不分库
spring.shardingsphere.sharding.default-database-strategy.none=Timi°
# 默认分表配置: 无分表策略的表不分表
spring.shardingsphere.sharding.default-table-strategy.none=Timi°

# 默认数据源: 不分库的数据从这个库操作, 支持读写分离
spring.shardingsphere.sharding.default-data-source-name=ds0

# 读写分离配置
spring.shardingsphere.sharding.master-slave-rules.ds0.master-data-source-name=master0
spring.shardingsphere.sharding.master-slave-rules.ds0.slave-data-source-names=master0slave0


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

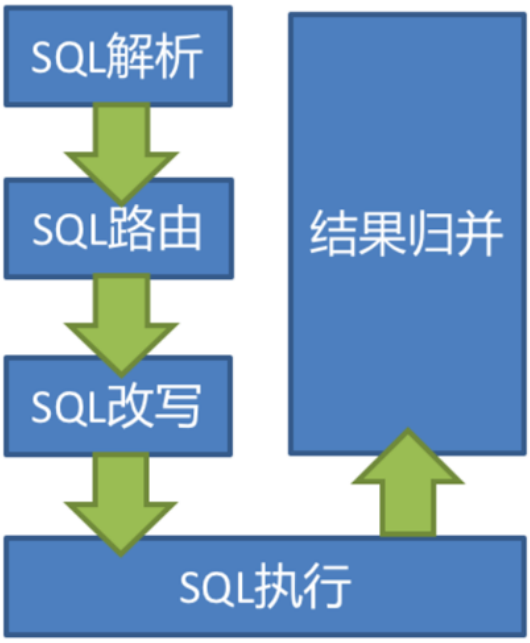
```
spring.shardingsphere.datasource.pos.driver-class-
name=com.mysql.cj.jdbc.Driver
spring.shardingsphere.datasource.pos.url=jdbc:mysql://172.19.193.244:3306/db
_lala_pos?useUnicode=true&characterEncoding=UTF-
8&serverTimezone=Asia/Shanghai
spring.shardingsphere.datasource.pos.username=lala_pos_rw
spring.shardingsphere.datasource.pos.password=vLqUeUxC1mD7ZwPZlF_G

spring.shardingsphere.datasource.site.type=com.alibaba.druid.pool.DruidDataS
ource
spring.shardingsphere.datasource.site.driver-class-
name=com.mysql.cj.jdbc.Driver
spring.shardingsphere.datasource.site.url=jdbc:mysql://172.19.193.244:3306/d
b_lala_pos_site?useUnicode=true&characterEncoding=UTF-
8&serverTimezone=Asia/Shanghai
spring.shardingsphere.datasource.site.username=lala_pos_rw
spring.shardingsphere.datasource.site.password=vLqUeUxC1mD7ZwPZlF_G
```

三、工具原理

逻辑

- 该解决方案可以理解为为增强版的 JDBC 驱动, 完全基于JDBC对SQL进行改写/对结果集进行归并
- 通过配置的解析对真实表进行相应配置的切割, 得到相应数量的逻辑表并赋予下标/索引
- 执行数据库操作时根据配置的分片键/策略解析逻辑表, 改写最终的执行语句, 并将结果进行归并



代码

- 文中采用的是默认分片算法: PreciseShardingAlgorithm, 可选四种详见ShardingAlgorithm子接口
- 当sql执行时携带其参数调用分片算法的 doSharding(Collection, PreciseShardingValue) 方法
- 由开法则对真实表进行逻辑表的选择并输出给组件进行sql改造并执行

数据

- 解析配置所获得的逻辑表、策略均存于应用内存中
- 查询获得的结果集在归并前存储于应用内存中 (gc回收)

四、全局事务

- 启用工具: @EnableGlobalTransactional

  > 在shardingJdbc的基础上再启用seata分布式事务解决方案

  ```
  @SpringBootApplication
  @EnableGlobalTransactional // 开启全局事务
  @ComponentScan("cn.gmlee.tools")
  public class XxxApp {
      public static void main(String[] args) {
          SpringApplication.run(XxxApp.class, args);
      }
  }
  ```

- 追加配置: application.properties

  ```
  # Seata
  seata.txServiceGroup=SEATA_GROUP
  seata.applicationId=seata-server

  seata.registry.type=nacos
  seata.registry.nacos.application=seata-server
  seata.registry.nacos.cluster=DEFAULT
  seata.registry.nacos.group=SEATA_GROUP
  seata.registry.nacos.username=username
  seata.registry.nacos.password=password
  seata.registry.nacos.server-addr=${spring.cloud.nacos.config.server-addr}

  seata.config.type=nacos
  seata.config.nacos.group=SEATA_GROUP
  seata.config.nacos.username=username
  seata.config.nacos.password=password
  seata.config.nacos.server-addr=${spring.cloud.nacos.discovery.server-addr}
  ```

- 添加配置: seata.conf

  > 该配置只有shardingJdbc项目需要添加, 普通的微服务加入全局事务只需要上述配置即可

  ```
  client {
      application.id = seata-server
      transaction.service.group = SEATA_GROUP
  }
  ```

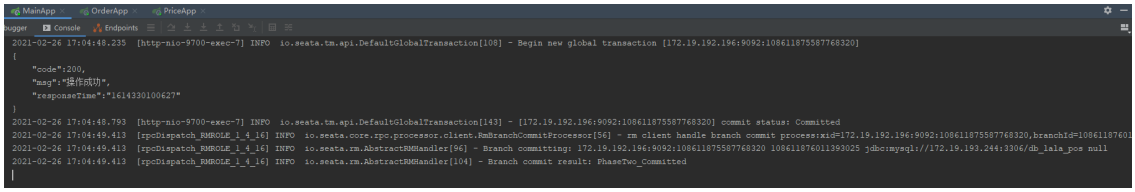
五、使用示例

5.1 成功案例

- pos-main: 分布式事务入口所在的微服务

> 省略controller资源代码

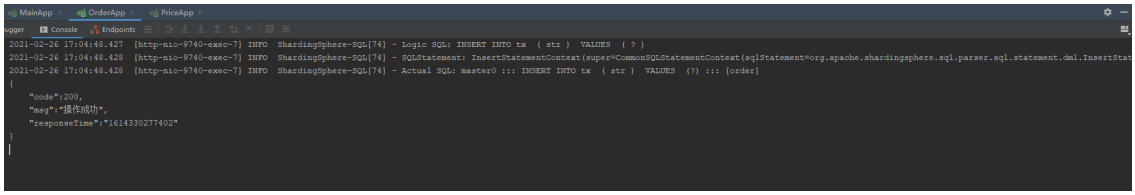
```
@Override
@GlobalTransactional // 全局事务
public void order(Tx tx) {
    tx.setId(null);
    txMapper.insert(tx);
    JsonResult result = posOrderApi.come();
    System.out.println(JsonUtil.format(JsonUtil.toJson(result)));
    AssertUtil.isOk(result, String.format("服务调用异常: %s", "order"));
}
```



- pos-order: 分布式事务的分支1事务所在的微服务, 值得一提的是: 该服务在数据库方面 **同时启用了分库分表、分布式事务** 解决方案

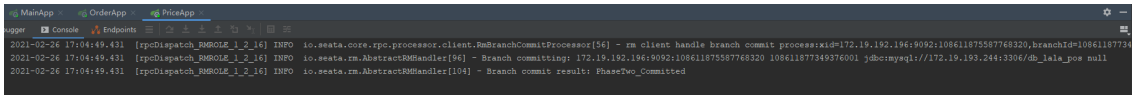
> 此处打印了本文的主角: ShardingJdbc的执行日志: 逻辑日志、真实日志, 详见下图第一行、第三行..

```
@Override
@Transactional // 本地事务
public void order(Tx tx) {
    tx.setId(null);
    txMapper.insert(tx);
    JsonResult result = posPriceApi.come();
    System.out.println(JsonUtil.format(JsonUtil.toJson(result)));
    AssertUtil.isOk(result, String.format("服务调用异常: %s", "price"));
}
```



- pos-price: 分布式事务的分支2事务所在的微服务

```
@Override
@Transactional // 开起本地事务
public void order(Tx tx) {
    tx.setId(null);
    txMapper.insert(tx);
}
```



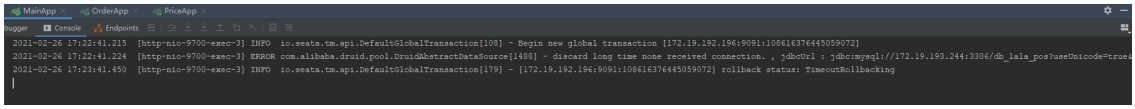
5.2 回滚案例

> 本次回滚采用的是在pos-price被调用的代码中打断点直至默认的60000ms事务超时后事务回滚, 详见seata AT模式

- pos-main: 分布式事务入口所在的微服务

  > 省略controller资源代码

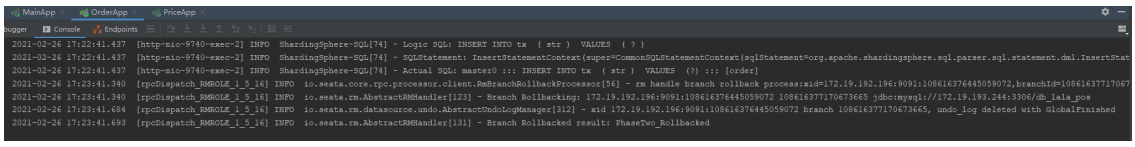
```
@Override
@GlobalTransactional // 全局事务
public void order(Tx tx) {
    tx.setId(null);
    txMapper.insert(tx);
    JsonResult result = posOrderApi.come();
    System.out.println(JsonUtil.format(JsonUtil.toJson(result)));
    AssertUtil.isOk(result, String.format("服务调用异常: %s", "order"));
}
```



- pos-order: 分布式事务的分支1事务所在的微服务, 值得一提的是: 该服务在数据库方面 **同时启用了分库分表、分布式事务** 解决方案

  > 此处打印了本文的主角: ShardingJdbc的执行日志: 逻辑日志、真实日志, 详见下图第一行、第三行..

```
@Override
@Transactional // 本地事务
public void order(Tx tx) {
    tx.setId(null);
    txMapper.insert(tx);
    JsonResult result = posPriceApi.come();
    System.out.println(JsonUtil.format(JsonUtil.toJson(result)));
    AssertUtil.isOk(result, String.format("服务调用异常: %s", "price"));
}
```



- pos-price: 分布式事务的分支2事务所在的微服务

```
@Override
@Transactional // 开起本地事务
public void order(Tx tx) {
    tx.setId(null);
    txMapper.insert(tx);
}
```

> 此处示例项目没做做好超时的日志输出控制, 没有相应日志输出, 但其全局事务已生效, 数据库展现了数据最终一致性。

六、特别提示

> 如果文中三个项目所使用的数据库不一致, 则需要相互注册/配置数据源
>
> 原理请阅读Seata-分布式解决方案集成简要说明中的AT模式, 或其他权威AT模式的原理介绍!
>
> 恰好示例项目则用的不是同一个数据库, 现贴出互相注册/配置的数据源

- pos-main: 分布式事务入口所在的微服务

```
# 这个是pos-main微服务自己的读写分离数据源
spring.datasource.master.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.master.url=jdbc:mysql://ip:port/db_lala_pos?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
spring.datasource.master.username=username
spring.datasource.master.password=password

spring.datasource.slave.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.slave.url=jdbc:mysql://ip:port/db_lala_pos?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
spring.datasource.slave.username=username
spring.datasource.slave.password=password

# 这下面分别是order、site微服务的数据源
spring.datasource.order.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.order.url=jdbc:mysql://ip:port/db_lala_pos_order?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
spring.datasource.order.username=username
spring.datasource.order.password=password
# 这个site微服务文中没有提到, 若不需要该服务加入到全局事务则不需要配置
spring.datasource.site.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.site.url=jdbc:mysql://ip:port/db_lala_pos_site?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
spring.datasource.site.username=username
spring.datasource.site.password=password
```

- pos-order: 分布式事务的分支1事务所在的微服务, 值得一提的是: 该服务在数据库方面 **同时启用了分库分表、分布式事务** 解决方案

```
# 此处省略order微服务自己的分库分表读写分离数据源配置
# 以下是pos通用数据库微服务和site微服务的数据源注册/配置
spring.shardingsphere.datasource.pos.type=com.alibaba.druid.pool.DruidDataSo
urce
spring.shardingsphere.datasource.pos.driver-class-
name=com.mysql.cj.jdbc.Driver
spring.shardingsphere.datasource.pos.url=jdbc:mysql://ip:port/db_lala_pos?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
spring.shardingsphere.datasource.pos.username=username
spring.shardingsphere.datasource.pos.password=password

spring.shardingsphere.datasource.site.type=com.alibaba.druid.pool.DruidDataS
ource
spring.shardingsphere.datasource.site.driver-class-
name=com.mysql.cj.jdbc.Driver
spring.shardingsphere.datasource.site.url=jdbc:mysql://ip:port/db_lala_pos_s
ite?useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
spring.shardingsphere.datasource.site.username=username
spring.shardingsphere.datasource.site.password=password
```

- pos-price: 分布式事务的分支2事务所在的微服务

```
# 此处省略price微服务自己的读写分离数据源配置
# 以下是order微服务和site微服务的数据源注册/配置
spring.datasource.order.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.order.url=jdbc:mysql://ip:port/db_lala_pos_order?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
spring.datasource.order.username=username
spring.datasource.order.password=password

spring.datasource.site.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.site.url=jdbc:mysql://ip:port/db_lala_pos_site?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
spring.datasource.site.username=username
spring.datasource.site.password=password
```

源码

文中示例是企业在线项目, 不便展示源码, 敬请谅解!