2 Sharding-JDBC入门使用

2.1不使用Spring

引入Maven依赖

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>sharding-jdbc-core</artifactId>
    <version>${sharding-sphere.version}</version>
</dependency>
```

基于Java编码的规则配置

Sharding-JDBC的分库分表通过规则配置描述，以下例子是根据user_id取模分库, 且根据order_id取模分表的两库两表的配置。

```
    // 配置真实数据源
    Map<String, DataSource> dataSourceMap = new HashMap<>();

    // 配置第一个数据源
    BasicDataSource dataSource1 = new BasicDataSource();
    dataSource1.setDriverClassName("com.mysql.jdbc.Driver");
    dataSource1.setUrl("jdbc:mysql://localhost:3306/ds0");
    dataSource1.setUsername("root");
    dataSource1.setPassword("");
    dataSourceMap.put("ds0", dataSource1);

    // 配置第二个数据源
    BasicDataSource dataSource2 = new BasicDataSource();
    dataSource2.setDriverClassName("com.mysql.jdbc.Driver");
    dataSource2.setUrl("jdbc:mysql://localhost:3306/ds1");
    dataSource2.setUsername("root");
    dataSource2.setPassword("");
    dataSourceMap.put("ds1", dataSource2);

    // 配置Order表规则
```

```
    TableRuleConfiguration orderTableRuleConfig = new
TableRuleConfiguration();
    orderTableRuleConfig.setLogicTable("t_order");
    orderTableRuleConfig.setActualDataNodes("ds${0..1}.t_order${0..1}");

    // 配置分库 + 分表策略
    orderTableRuleConfig.setDatabaseShardingStrategyConfig(new
InlineShardingStrategyConfiguration("user_id", "ds${user_id % 2}"));
    orderTableRuleConfig.setTableShardingStrategyConfig(new
InlineShardingStrategyConfiguration("order_id", "t_order${order_id %
2}"));

    // 配置分片规则
    ShardingRuleConfiguration shardingRuleConfig = new
ShardingRuleConfiguration();
    shardingRuleConfig.getTableRuleConfigs().add(orderTableRuleConfig);

    // 省略配置order_item表规则...
    // ...

    // 获取数据源对象
    DataSource dataSource =
ShardingDataSourceFactory.createDataSource(dataSourceMap,
shardingRuleConfig, new ConcurrentHashMap(), new Properties());
```

基于Yaml的规则配置

或通过Yaml方式配置，与以上配置等价：

```
dataSources:
  ds0: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ds0
    username: root
    password:
  ds1: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ds1
    username: root
    password:

tables:
```

```
  t_order:
    actualDataNodes: ds${0..1}.t_order${0..1}
    databaseStrategy:
      inline:
        shardingColumn: user_id
        algorithmInlineExpression: ds${user_id % 2}
    tableStrategy:
      inline:
        shardingColumn: order_id
        algorithmInlineExpression: t_order${order_id % 2}
  t_order_item:
    actualDataNodes: ds${0..1}.t_order_item${0..1}
    databaseStrategy:
      inline:
        shardingColumn: user_id
        algorithmInlineExpression: ds${user_id % 2}
    tableStrategy:
      inline:
        shardingColumn: order_id
        algorithmInlineExpression: t_order_item${order_id % 2}
    DataSource dataSource =
YamlShardingDataSourceFactory.createDataSource(yamlFile);
```

使用原生JDBC

通过ShardingDataSourceFactory或者YamlShardingDataSourceFactory工厂和规则配置对象获取ShardingDataSource，ShardingDataSource实现自JDBC的标准接口DataSource。然后可通过DataSource选择使用原生JDBC开发，或者使用JPA, MyBatis等ORM工具。 以JDBC原生实现为例：

```
DataSource dataSource =
YamlShardingDataSourceFactory.createDataSource(yamlFile);
String sql = "SELECT i.* FROM t_order o JOIN t_order_item i ON
o.order_id=i.order_id WHERE o.user_id=? AND o.order_id=?";
try (
        Connection conn = dataSource.getConnection();
        PreparedStatement preparedStatement = conn.prepareStatement(sql))
{
    preparedStatement.setInt(1, 10);
    preparedStatement.setInt(2, 1001);
    try (ResultSet rs = preparedStatement.executeQuery()) {
```

```
        while(rs.next()) {
            System.out.println(rs.getInt(1));
            System.out.println(rs.getInt(2));
        }
    }
}
```

2.2使用Spring

引入Maven依赖

```
<!-- for spring boot -->
<dependency>
    <groupId>io.shardingsphere</groupId>
    <artifactId>sharding-jdbc-spring-boot-starter</artifactId>
    <version>${sharding-sphere.version}</version>
</dependency>

<!-- for spring namespace -->
<dependency>
    <groupId>io.shardingsphere</groupId>
    <artifactId>sharding-jdbc-spring-namespace</artifactId>
    <version>${sharding-sphere.version}</version>
</dependency>
```

基于Spring boot的规则配置

```
sharding.jdbc.datasource.names=ds0,ds1

sharding.jdbc.datasource.ds0.type=org.apache.commons.dbcp2.BasicDataSource
sharding.jdbc.datasource.ds0.driver-class-name=com.mysql.jdbc.Driver
sharding.jdbc.datasource.ds0.url=jdbc:mysql://localhost:3306/ds0
sharding.jdbc.datasource.ds0.username=root
sharding.jdbc.datasource.ds0.password=

sharding.jdbc.datasource.ds1.type=org.apache.commons.dbcp2.BasicDataSource
sharding.jdbc.datasource.ds1.driver-class-name=com.mysql.jdbc.Driver
sharding.jdbc.datasource.ds1.url=jdbc:mysql://localhost:3306/ds1
sharding.jdbc.datasource.ds1.username=root
sharding.jdbc.datasource.ds1.password=
```

```
sharding.jdbc.config.sharding.default-database-strategy.inline.sharding-column=user_id
sharding.jdbc.config.sharding.default-database-strategy.inline.algorithm-expression=ds$->{user_id % 2}

sharding.jdbc.config.sharding.tables.t_order.actual-data-nodes=ds$->{0..1}.t_order$->{0..1}
sharding.jdbc.config.sharding.tables.t_order.table-strategy.inline.sharding-column=order_id
sharding.jdbc.config.sharding.tables.t_order.table-strategy.inline.algorithm-expression=t_order$->{order_id % 2}

sharding.jdbc.config.sharding.tables.t_order_item.actual-data-nodes=ds$->{0..1}.t_order_item$->{0..1}
sharding.jdbc.config.sharding.tables.t_order_item.table-strategy.inline.sharding-column=order_id
sharding.jdbc.config.sharding.tables.t_order_item.table-strategy.inline.algorithm-expression=t_order_item$->{order_id % 2}
```

基于Spring命名空间的规则配置

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

 xmlns:sharding="http://shardingsphere.io/schema/shardingsphere/sharding"
    xsi:schemaLocation="http://www.springframework.org/schema/beans

 http://www.springframework.org/schema/beans/spring-beans.xsd

 http://shardingsphere.io/schema/shardingsphere/sharding

 http://shardingsphere.io/schema/shardingsphere/sharding/sharding.xsd
                        ">
    <bean id="ds0" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/ds0" />
        <property name="username" value="root" />
        <property name="password" value="" />
    </bean>
```

```
    <bean id="ds1" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/ds1" />
        <property name="username" value="root" />
        <property name="password" value="" />
    </bean>

    <sharding:inline-strategy id="databaseStrategy" sharding-
column="user_id" algorithm-expression="ds$->{user_id % 2}" />
    <sharding:inline-strategy id="orderTableStrategy" sharding-
column="order_id" algorithm-expression="t_order$->{order_id % 2}" />
    <sharding:inline-strategy id="orderItemTableStrategy" sharding-
column="order_id" algorithm-expression="t_order_item$->{order_id % 2}" />

    <sharding:data-source id="shardingDataSource">
        <sharding:sharding-rule data-source-names="ds0,ds1">
            <sharding:table-rules>
                <sharding:table-rule logic-table="t_order" actual-data-
nodes="ds$->{0..1}.t_order$->{0..1}" database-strategy-
ref="databaseStrategy" table-strategy-ref="orderTableStrategy" />
                <sharding:table-rule logic-table="t_order_item" actual-
data-nodes="ds$->{0..1}.t_order_item$->{0..1}" database-strategy-
ref="databaseStrategy" table-strategy-ref="orderItemTableStrategy" />
            </sharding:table-rules>
        </sharding:sharding-rule>
    </sharding:data-source>
</beans>
```

在Spring中使用DataSource

直接通过注入的方式即可使用DataSource，或者将DataSource配置在JPA、Hibernate或MyBatis中使用。

```
@Resource
private DataSource dataSource;
```

2.3 数据源配置

yaml格式

```yaml
dataSources:
  ds0: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ds0
    username: root
    password:
  ds1: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ds1
    username: root
    password:

props:
  sql.show: true
```

spring boot 配置

```properties
sharding.jdbc.datasource.names=ds0,ds1

sharding.jdbc.datasource.ds0.type=org.apache.commons.dbcp.BasicDataSource
sharding.jdbc.datasource.ds0.driver-class-name=com.mysql.jdbc.Driver
sharding.jdbc.datasource.ds0.url=jdbc:mysql://localhost:3306/ds0
sharding.jdbc.datasource.ds0.username=root
sharding.jdbc.datasource.ds0.password=

sharding.jdbc.datasource.ds1.type=org.apache.commons.dbcp.BasicDataSource
sharding.jdbc.datasource.ds1.driver-class-name=com.mysql.jdbc.Driver
sharding.jdbc.datasource.ds1.url=jdbc:mysql://localhost:3306/ds1
sharding.jdbc.datasource.ds1.username=root
sharding.jdbc.datasource.ds1.password=
```

数据源配置说明

```
dataSources: # 配置数据源列表,必须是有效的jdbc配置,目前仅支持MySQL与PostgreSQL,另外通过一些未公开(代码中可查,但可能会在未来有变化)的变量,可以配置来兼容其他支持JDBC的数据库,但由于没有足够的测试支持,可能会有严重的兼容性问题,配置时候要求至少有一个
  master_ds_0: # 数据源名称,可以是合法的字符串,目前的校验规则中,没有强制性要求,只要是合法的yaml字符串即可,但如果要用于分库分表配置,则需要有有意义的标志(在分库分表配置中详述),以下为目前公开的合法配置项目,不包含内部配置参数
    # 以下参数为必备参数
    url: •jdbc:mysql://127.0.0.1:3306/demo_ds_slave_1?serverTimezone=UTC&useSSL=false # 这里的要求合法的jdbc连接串即可,目前尚未兼容MySQL 8.x,需要在maven编译时候,升级MySQL JDBC版本到5.1.46或者47版本(不建议升级到JDBC的8.x系列版本,需要修改源代码,并且无法通过很多测试case)
    username: root # MySQL用户名
    password: password # MySQL用户的明文密码
    # 以下参数为可选参数,给出示例为默认配置,主要用于连接池控制
    connectionTimeoutMilliseconds: 30000 #连接超时控制
    idleTimeoutMilliseconds: 60000 # 连接空闲时间设置
    maxLifetimeMilliseconds: 0 # 连接的最大持有时间,0为无限制
    maxPoolSize: 50 # 连接池中最大维持的连接数量
    minPoolSize: 1 # 连接池的最小连接数量
    maintenanceIntervalMilliseconds: 30000 # 连接维护的时间间隔 atomikos框架需求
```

2.4 属性配置

配置示例

```
props:
  sql.show: true
```

可配置属性说明

```
props:
  sql.show: #是否开启SQL显示，默认值: false
  acceptor.size: # accept连接的线程数量,默认为cpu核数2倍
  executor.size: #工作线程数量最大，默认值: 无限制
  max.connections.size.per.query: # 每个查询可以打开的最大连接数量,默认为1
  check.table.metadata.enabled: #是否在启动时检查分表元数据一致性，默认值: false
  proxy.frontend.flush.threshold: # proxy的服务时候,对于单个大查询,每多少个网络包返回一次
  proxy.transaction.type: # 默认LOCAL,proxy的事务模型 允许LOCAL,XA,BASE三个值LOCAL无分布式事务,XA则是采用atomikos实现的分布式事务 BASE目前尚未实现
  proxy.opentracing.enabled: # 是否启用opentracing
  proxy.backend.use.nio: # 是否采用netty的NIO机制连接后端数据库,默认False ,使用epoll机制
  proxy.backend.max.connections: # 使用NIO而非epoll的话,proxy后台连接每个netty客户端允许的最大连接数量(注意不是数据库连接限制) 默认为8
  proxy.backend.connection.timeout.seconds: #使用nio而非epoll的话,proxy后台连接的超时时间,默认60s
```

2.5 spring boot 示例

maven 依赖 pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.3.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.study.mike</groupId>
    <artifactId>sharding-jdbc-study</artifactId>
    <version>0.0.1-SNAPSHOT</version>
```

```
    <name>sharding-jdbc-study</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>2.0.0</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>io.shardingsphere</groupId>
            <artifactId>sharding-jdbc-spring-boot-starter</artifactId>
            <version>3.1.0</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.1.14</version>
```

```
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

配置 application.yml

```
sharding:
  jdbc:
    datasource:
      names: ds0,ds1
      ds0:
        type: com.alibaba.druid.pool.DruidDataSource
        driver-class: com.mysql.jdbc.Driver
        url: jdbc:mysql://localhost:3306/db1?
useUnicode=true&characterEncoding=utf-8&serverTimezone=UTC
        username: mike
        password: Mike666!
        maxPoolSize: 50
        minPoolSize: 1
      ds1:
        type: com.alibaba.druid.pool.DruidDataSource
        driver-class: com.mysql.jdbc.Driver
        url: jdbc:mysql://localhost:3306/db2?
useUnicode=true&characterEncoding=utf-8&serverTimezone=UTC
        username: mike
        password: Mike666!
        maxPoolSize: 50
        minPoolSize: 1
    config:
```

```
    sharding:
      default-data-source-name: ds0
    props:
      sql.show: true
```