# 自定义分片模式

## 数据库表结构

创建db\_0数据库

CREATE TABLE `t\_order\_0` (

`order\_id` bigint(20) NOT NULL,

`user\_id` bigint(20) NOT NULL,

PRIMARY KEY (`order\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_bin;

CREATE TABLE `t\_order\_1` (

`order\_id` bigint(20) NOT NULL,

`user\_id` bigint(20) NOT NULL,

PRIMARY KEY (`order\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_bin;

## Maven依赖

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.3.RELEASE</version>  <relativePath /> <!-- lookup parent from repository -->  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <java.version>1.8</java.version>  </properties>  <dependencies>  <!-- jpa -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency>  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.3.2</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <optional>true</optional>  </dependency>  <!-- 引入shardingjdbc依赖信息 -->  <dependency>  <groupId>io.shardingjdbc</groupId>  <artifactId>sharding-jdbc-core</artifactId>  <version>2.0.3</version>  </dependency>  <dependency>  <groupId>com.dangdang</groupId>  <artifactId>sharding-jdbc-self-id-generator</artifactId>  <version>1.4.2</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.0.12</version>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build> |

## application配置

|  |
| --- |
| ###数据库访问连接  spring:  jdbc:  db0:  password: root  className: com.mysql.jdbc.Driver  url: jdbc:mysql://localhost:3306/%s?characterEncoding=utf-8  username: root  jpa:  database: mysql  show-sql: **true**  hibernate:  ## 自己建表  ddl-auto: none  application:  name: sharding-jdbc-first |

## 配置形式

分表算法类需要实现SingleKeyTableShardingAlgorithm<T>接口

### 分表算法

#### DataSourceConfig

|  |
| --- |
| // 数据源相关配置信息  @Configuration  **public** **class** DataSourceConfig {  @Value("${spring.jdbc.db0.className}")  **private** String className;  @Value("${spring.jdbc.db0.url}")  **private** String url;  @Value("${spring.jdbc.db0.username}")  **private** String username;  @Value("${spring.jdbc.db0.password}")  **private** String password;  @Bean  **public** IdGenerator getIdGenerator() {  **return** **new** CommonSelfIdGenerator();  }  @Bean  **public** DataSource getDataSource() {  **return** buildDataSource();  }  **private** DataSource buildDataSource() {  // 1.设置分库映射  Map<String, DataSource> dataSourceMap = **new** HashMap<>(2);  dataSourceMap.put("ds\_0", createDataSource("ds\_0"));  // dataSourceMap.put("ds\_1", createDataSource("ds\_1"));  // 设置默认db为ds\_0，也就是为那些没有配置分库分表策略的指定的默认库  // 如果只有一个库，也就是不需要分库的话，map里只放一个映射就行了，只有一个库时不需要指定默认库，  // 但2个及以上时必须指定默认库，否则那些没有配置策略的表将无法操作数据  DataSourceRule rule = **new** DataSourceRule(dataSourceMap, "ds\_0");  // 2.设置分表映射，将t\_order\_0和t\_order\_1两个实际的表映射到t\_order逻辑表  TableRule orderTableRule = TableRule.*builder*("t\_order").actualTables(Arrays.*asList*("t\_order\_0", "t\_order\_1"))  .dataSourceRule(rule).build();  // 3.具体的分库分表策略  ShardingRule shardingRule = ShardingRule.*builder*().dataSourceRule(rule)  .tableRules(Arrays.*asList*(orderTableRule))  // 根据userid分片字段  .tableShardingStrategy(**new** TableShardingStrategy("user\_id", **new** TableShardingAlgorithm())).build();  // 创建数据源  DataSource dataSource = ShardingDataSourceFactory.*createDataSource*(shardingRule);  **return** dataSource;  }  **private** DataSource createDataSource(String dataSourceName) {  // 使用druid连接数据库  DruidDataSource druidDataSource = **new** DruidDataSource();  druidDataSource.setDriverClassName(className);  druidDataSource.setUrl(String.*format*(url, dataSourceName));  druidDataSource.setUsername(username);  druidDataSource.setPassword(password);  **return** druidDataSource;  }  } |

#### TableShardingAlgorithm

|  |
| --- |
| **public** **class** TableShardingAlgorithm **implements** SingleKeyTableShardingAlgorithm<Long> {  // sql 中关键字 匹配符为 =的时候，表的路由函数  **public** String doEqualSharding(Collection<String> availableTargetNames, ShardingValue<Long> shardingValue) {  **for** (String tableName : availableTargetNames) {  **if** (tableName.endsWith(shardingValue.getValue() % 2 + "")) {  **return** tableName;  }  }  **throw** **new** IllegalArgumentException();  }  @Override  **public** Collection<String> doInSharding(Collection<String> availableTargetNames, ShardingValue<Long> shardingValue) {  **return** **null**;  }  @Override  **public** Collection<String> doBetweenSharding(Collection<String> availableTargetNames,  ShardingValue<Long> shardingValue) {  **return** **null**;  }  } |

### 分库算法

分库算法类需要实现SingleKeyDatabaseShardingAlgorithm<T>接口

#### DataSourceConfig

|  |
| --- |
| // 数据源相关配置信息  @Configuration  **public** **class** DataSourceConfig {  @Value("${spring.jdbc.db0.className}")  **private** String className;  @Value("${spring.jdbc.db0.url}")  **private** String url;  @Value("${spring.jdbc.db0.username}")  **private** String username;  @Value("${spring.jdbc.db0.password}")  **private** String password;  @Bean  **public** IdGenerator getIdGenerator() {  **return** **new** CommonSelfIdGenerator();  }  @Bean  **public** DataSource getDataSource() {  **return** buildDataSource();  }  **private** DataSource buildDataSource() {  // 1.设置分库映射  Map<String, DataSource> dataSourceMap = **new** HashMap<>(2);  dataSourceMap.put("ds\_0", createDataSource("ds\_0"));  dataSourceMap.put("ds\_1", createDataSource("ds\_1"));  // 设置默认db为ds\_0，也就是为那些没有配置分库分表策略的指定的默认库  // 如果只有一个库，也就是不需要分库的话，map里只放一个映射就行了，只有一个库时不需要指定默认库，  // 但2个及以上时必须指定默认库，否则那些没有配置策略的表将无法操作数据  DataSourceRule rule = **new** DataSourceRule(dataSourceMap, "ds\_0");  // 2.设置分表映射，将t\_order\_0和t\_order\_1两个实际的表映射到t\_order逻辑表  TableRule orderTableRule = TableRule.*builder*("t\_order").dataSourceRule(rule).build();  // 3.具体的分库分表策略  ShardingRule shardingRule = ShardingRule.*builder*().dataSourceRule(rule)  .tableRules(Arrays.*asList*(orderTableRule))  .databaseShardingStrategy(**new** DatabaseShardingStrategy("user\_id", **new** DatabaseShardingAlgorithm()))  .build();  // 创建数据源  DataSource dataSource = ShardingDataSourceFactory.*createDataSource*(shardingRule);  **return** dataSource;  }  **private** DataSource createDataSource(String dataSourceName) {  // 使用druid连接数据库  DruidDataSource druidDataSource = **new** DruidDataSource();  druidDataSource.setDriverClassName(className);  druidDataSource.setUrl(String.*format*(url, dataSourceName));  druidDataSource.setUsername(username);  druidDataSource.setPassword(password);  **return** druidDataSource;  }  } |

#### DatabaseShardingAlgorithm

|  |
| --- |
| **public** **class** DatabaseShardingAlgorithm **implements** SingleKeyDatabaseShardingAlgorithm<Long> {  @Override  **public** String doEqualSharding(Collection<String> databases, ShardingValue<Long> shardingValue) {  **for** (String tableName : databases) {  System.***out***.println("tableName:" + tableName + ",----" + shardingValue.getValue());  **if** (tableName.endsWith(shardingValue.getValue() % 2 + "")) {  **return** tableName;  }  }  **throw** **new** IllegalArgumentException();  }  @Override  **public** Collection<String> doInSharding(Collection<String> availableTargetNames, ShardingValue<Long> shardingValue) {  **return** **null**;  }  @Override  **public** Collection<String> doBetweenSharding(Collection<String> availableTargetNames,  ShardingValue<Long> shardingValue) {  **return** **null**;  }  } |

## 配置形式

### Maven依赖

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.4.RELEASE</version>  <relativePath />  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  <dependency>  <groupId>io.shardingsphere</groupId>  <artifactId>sharding-jdbc-spring-boot-starter</artifactId>  <version>3.0.0.M3</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.1.9</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  </dependency>  </dependencies> |

### application配置

|  |
| --- |
| spring:  jpa:  show-sql: **true**  hibernate:  ddl-auto: none  database-platform: org.hibernate.dialect.MySQL5InnoDBDialect  sharding:  jdbc:  ####ds1  datasource:  names: ds1  ds1:  password: root  type: com.alibaba.druid.pool.DruidDataSource  driver-class-name: com.mysql.jdbc.Driver  url: jdbc:mysql://127.0.0.1:3306/ds\_1  username: root  config:  sharding:  tables:  t\_order:  table-strategy:  inline:  #### 根据userid 进行分片  sharding-column: user\_id  algorithm-expression: ds\_1.t\_order\_$->{user\_id % 2}  actual-data-nodes: ds1.t\_order\_$->{0..1}  props:  sql:  ### 开启分片日志  show: **true** |

## 日志分析实现原理

|  |
| --- |
| **public** **interface** OrderRepository **extends** CrudRepository<OrderEntity, Long> {  @Query(value = "SELECT order\_id ,user\_id FROM t\_order where order\_id in (?1);", nativeQuery = **true**)  **public** List<OrderEntity> findExpiredOrderState(List<String> bpIds);  @Query(value = "SELECT order\_id ,user\_id FROM t\_order where user\_id=:userId ", nativeQuery = **true**)  **public** List<OrderEntity> findByUserId(@Param("userId") Long userId);  } |