## 多数据源/动态数据源实现

比如在电商网站中可能会有这样的需求:根据用户所在城市不同,查询不同城市的商品数据。而在后台,这些不同城市的数据被分配在不同的数据库当中。很多人想当然的就觉得需要使用ShardingJDBC来实现多数据源管理。

这种场景就是一个典型的多数据源切换的场景。但是我们仔细分析这样的场景,他跟分库分表其实并不太相同。ShardingJDBC固然可以使用Hint策略实现快速的数据库分库查询。例如前端传进来一个cityId字段,然后在后台查询数据时,将cityId设置到HintManager中,通过定制Hint策略,将后续的SQL操作分配到目标数据源当中。Hint分片策略案例参考分库分表二: ShardingJDBC进阶实战案例上但是,其实这种场景跟分库分表还是有差距的,那么如果不使用ShardingJDBC如何实现多数据源的切换。

## 方案一: JDBC自带的多数据源方案

在springboot中整合好mybatis和web功能 ,能够实现普通的增删改查。 yml配置

```
1 spring:
    datasource:
       datasource1:
         url: jdbc:mysql://localhost/sharding test?
   useSSL=false&useUnicode=true&characterEncoding=utf-
   8&zeroDateTimeBehavior=convertToNull&transformedBitIsBoolean=true&tinyInt1isBit=false&a
   llowMultiQueries=true&serverTimezone=GMT%2B8&allowPublicKeyRetrieval=true
         username: root
         password: root
         driver-class-name: com.mysql.cj.jdbc.Driver
       datasource2:
         url: jdbc:mysql://xxx/xmkf zt?useSSL=false&useUnicode=true&characterEncoding=utf-
   8&zeroDateTimeBehavior=convertToNull&transformedBitIsBoolean=true&tinyInt1isBit=false&a
   llowMultiQueries=true&serverTimezone=GMT%2B8&allowPublicKeyRetrieval=true&rewriteBatche
   dStatements=true
10
         username: xxx
         password: xxx
11
         driver-class-name: com.mysql.cj.jdbc.Driver
12
```

#### 配置文件中配置了两个数据源。在两个的数据源中创建表

```
1 CREATE TABLE `sharding_user` (
```

```
id bigint(11) NOT NULL AUTO_INCREMENT,

user_name varchar(255) DEFAULT NULL,

age int(11) DEFAULT NULL,

source varchar(255) DEFAULT 'test',

create_time datetime DEFAULT NULL,

update_time datetime DEFAULT NULL,

PRIMARY KEY ('id')

BNGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;
```

# 实体类 service 那些就忽略了 和普通的mybatis调用是一样的配置所有数据源DataSource

```
1 @Configuration
  public class DataSourceConfig {
   @Bean(name = "dataSource1")
4
   @ConfigurationProperties(prefix = "spring.datasource.datasource1")
    public DataSource dataSource1(){
                  // 底层会自动拿到spring.datasource中的配置, 创建一个DruidDataSource
                  return DruidDataSourceBuilder.create().build();
8
    }
9
10
   @Bean(name = "dataSource2")
11
   @ConfigurationProperties(prefix = "spring.datasource.datasource2")
    public DataSource dataSource2(){
13
                  // 底层会自动拿到spring.datasource中的配置, 创建一个DruidDataSource
14
                  return DruidDataSourceBuilder.create().build();
15
16
    }
   //事务管理器
17
   @Bean
18
    public DataSourceTransactionManager transactionManager(DynamicDataSource dataSource){
19
                  DataSourceTransactionManager dataSourceTransactionManager = new
20
   DataSourceTransactionManager();
                  dataSourceTransactionManager.setDataSource(dataSource);
21
                   return dataSourceTransactionManager;
23
24
25
```

#### 配置动态数据源

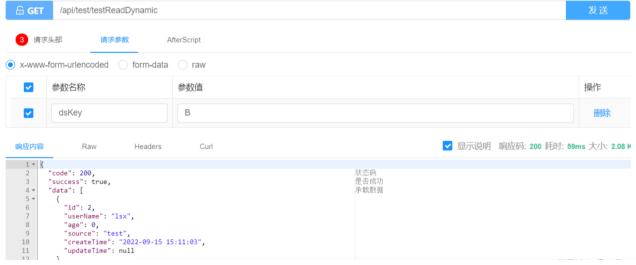
```
@Component("dynamicDataSource")
  @Primary
  public class DynamicDataSource extends AbstractRoutingDataSource {
   public static ThreadLocal<String> dataSourceName = new ThreadLocal<>();
5
   @Resource
   private DataSource dataSource1;
7
   @Resource
8
   private DataSource dataSource2;
9
10
   @Override
11
    protected Object determineCurrentLookupKey() {
12
                  //从当前线程中获取数据源
13
                  return dataSourceName.get();
14
15
16
   @Override
17
    public void afterPropertiesSet() {
18
                  Map<Object, Object> targetDataSources = new HashMap<>();
19
                  //假设B 代表北京的数据源
20
                  targetDataSources.put("B",dataSource1);
21
                  //假设x 代表厦门的数据源
22
                  targetDataSources.put("X",dataSource2);
                  //初始化 设置所有数据源
24
                  super.setTargetDataSources(targetDataSources);
                  //设置 默认数据源
26
                   super.setDefaultTargetDataSource(dataSource2);
27
                   super.afterPropertiesSet();
28
29
30
31
```

#### 测试代码:

```
@GetMapping("/testWriteDynamic")
   @ApiOperation(value = "测试动态数据源插入")
2
   @ApiOperationSupport(order = 9, author = "lsx")
   public R testWriteDynamic(@RequestParam(value = "dsKey", defaultValue = "B") String
  dsKey){
                  List<ShardingUser> list = new ArrayList<>();
5
                  for (int i = 0; i < 20; i++) {
6
                          ShardingUser user = new ShardingUser();
                          user.setAge(i);
                          user.setUserName("lsx");
9
                          user.setCreateTime(new Date());
10
                          list.add(user);
11
                  }
                  //从前端获取数据源 默认为北京的数据源
13
                  DynamicDataSource.dataSourceName.set(dsKey);
14
                  shardingUserService.saveBatch(list);
15
                  return R.success("成功");
16
17
    }
18
   @GetMapping("/testReadDynamic")
19
   @ApiOperation(value = "测试动态数据源读取")
20
   @ApiOperationSupport(order = 10, author = "lsx")
21
   public R testReadDynamic(@RequestParam(value = "dsKey", defaultValue = "X") String
  dsKey){
                  //从前端获取数据源 默认为厦门的数据源
23
                  DynamicDataSource.dataSourceName.set(dsKey);
24
                  List<ShardingUser> list = shardingUserService.list();
25
                  return R.data(list);
26
   }
27
```

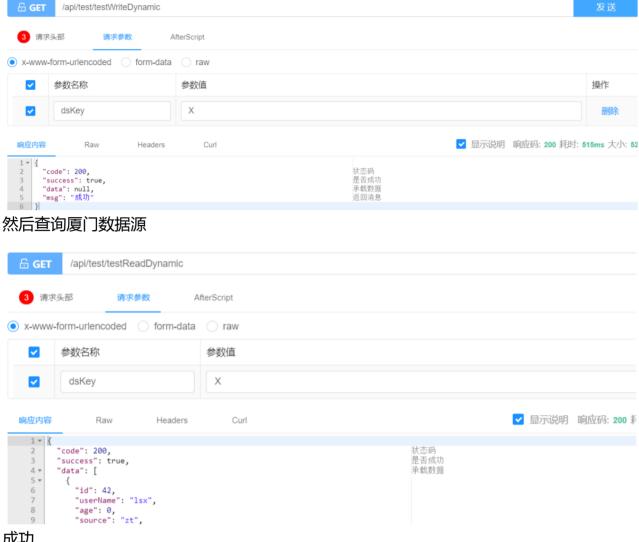
#### 测试插入北京数据源





查询成功

测试插入数据到厦门数据源



成功

这种方案的好处是根据前端传入的参数决定使用哪个数据源。

# 方案二: 使用动态数据源框架

引入依赖

```
1 <dependency>
2 <groupId>com.baomidou</groupId>
  <artifactId>dynamic-datasource-spring-boot-starter</artifactId>
4 <version>3.5.0</version>
5 </dependency>
```

# yml配置

```
1 spring:
    datasource:
```

```
type: com.alibaba.druid.pool.DruidDataSource
      #使用dynamicDatasource框架
4
      dynamic:
5
        #设置默认的数据源或者数据源组
6
        primary: XM
        #严格匹配数据源,默认false. true未匹配到指定数据源时抛异常,false使用默认数据源
8
        strict: false
        datasource:
          XM:
            url: jdbc:mysql://localhost/sharding_test?
12
  useSSL=false&useUnicode=true&characterEncoding=utf-
  8&zeroDateTimeBehavior=convertToNull&transformedBitIsBoolean=true&tinyInt1isBit=false&a
   llowMultiQueries=true&serverTimezone=GMT%2B8&allowPublicKeyRetrieval=true
            username: root
13
            password: root
14
            driver-class-name: com.mysql.cj.jdbc.Driver
          BJ:
16
            url: jdbc:mysql://xxx/xmkf zt?
17
  useSSL=false&useUnicode=true&characterEncoding=utf-
  8&zeroDateTimeBehavior=convertToNull&transformedBitIsBoolean=true&tinyInt1isBit=false&a
  llowMultiQueries=true&serverTimezone=GMT%2B8&allowPublicKeyRetrieval=true&rewriteBatche
   dStatements=true
18
            username: xxx
            password: xxx
19
            driver-class-name: com.mysql.cj.jdbc.Driver
```

引入了这个框架,配置好数据源就不需要在有额外的配置了。这个框架自动帮我们配置好动态数据源 用法

在service 实现类添加@DS()注解 通过DS注解指定数据源 方法上的DS注解会覆盖类上的DS注解

```
public List<ShardingUser> queryList() {
    return this.list();
}

13 }

14 }
```

### 测试代码

```
@GetMapping("/testWriteDynamic")
   @ApiOperation(value = "测试动态数据源插入")
   @ApiOperationSupport(order = 9, author = "lsx")
   public R testWriteDynamic(@RequestParam(value = "dsKey", defaultValue = "B") String
   dsKey){
                   for (int i = 0; i < 20; i++) {
                           ShardingUser user = new ShardingUser();
                           user.setAge(i);
                           user.setUserName("lsx");
                           user.setCreateTime(new Date());
                           shardingUserService.saveEntity(user);
11
                   return R.success("成功");
12
13
14
   @GetMapping("/testReadDynamic")
15
   @ApiOperation(value = "测试动态数据源读取")
16
   @ApiOperationSupport(order = 9, author = "lsx")
17
   public R testReadDynamic(@RequestParam(value = "dsKey",defaultValue = "X") String
18
   dsKey){
                   List<ShardingUser> list = shardingUserService.list();
19
                   return R.data(list);
20
   }
21
```

## 上述插入和查询都会去xm的数据源操作