

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。

yaml配置

```
1  spring:
2    datasource:
3      datasource1:
4        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
5        username: root
6        password: root
7        driver-class-name: com.mysql.cj.jdbc.Driver
8      datasource2:
9        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
11       password: xxx
12       driver-class-name: com.mysql.cj.jdbc.Driver
```

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` (
```

```

2  `id` bigint(11) NOT NULL AUTO_INCREMENT,
3  `user_name` varchar(255) DEFAULT NULL,
4  `age` int(11) DEFAULT NULL,
5  `source` varchar(255) DEFAULT 'test',
6  `create_time` datetime DEFAULT NULL,
7  `update_time` datetime DEFAULT NULL,
8  PRIMARY KEY (`id`)
9  ) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;
10

```

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
2  public class DataSourceConfig {
3
4      @Bean(name = "dataSource1")
5      @ConfigurationProperties(prefix = "spring.datasource.datasource1")
6      public DataSource dataSource1(){
7          // 底层会自动拿到spring.datasource中的配置， 创建一个DruidDataSource
8          return DruidDataSourceBuilder.create().build();
9      }
10
11      @Bean(name = "dataSource2")
12      @ConfigurationProperties(prefix = "spring.datasource.datasource2")
13      public DataSource dataSource2(){
14          // 底层会自动拿到spring.datasource中的配置， 创建一个DruidDataSource
15          return DruidDataSourceBuilder.create().build();
16      }
17      //事务管理器
18      @Bean
19      public DataSourceTransactionManager transactionManager(DynamicDataSource dataSource){
20          DataSourceTransactionManager dataSourceTransactionManager = new
21          DataSourceTransactionManager();
22          dataSourceTransactionManager.setDataSource(dataSource);
23          return dataSourceTransactionManager;
24      }
25  }

```

配置动态数据源

```

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

```

测试代码：

```

1  @GetMapping("/testWriteDynamic")
2  @ApiOperation(value = "测试动态数据源插入")
3  @ApiOperationSupport(order = 9, author = "lsx")
4  public R testWriteDynamic(@RequestParam(value = "dsKey",defaultValue = "B") String
    dsKey){
5
6      List<ShardingUser> list = new ArrayList<>();
7      for (int i = 0; i < 20; i++) {
8          ShardingUser user = new ShardingUser();
9          user.setAge(i);
10         user.setUserName("lsx");
11         user.setCreateTime(new Date());
12         list.add(user);
13     }
14     //从前端获取数据源 默认为北京的数据源
15     DynamicDataSource.dataSourceName.set(dsKey);
16     shardingUserService.saveBatch(list);
17     return R.success("成功");
18 }
19
20 @GetMapping("/testReadDynamic")
21 @ApiOperation(value = "测试动态数据源读取")
22 @ApiOperationSupport(order = 10, author = "lsx")
23 public R testReadDynamic(@RequestParam(value = "dsKey",defaultValue = "X") String
    dsKey){
24
25     //从前端获取数据源 默认为厦门的数据源
26     DynamicDataSource.dataSourceName.set(dsKey);
27     List<ShardingUser> list = shardingUserService.list();
28     return R.data(list);
29 }

```

测试插入北京数据源

GET

/api/test/testWriteDynamic

3 请求头部

请求参数

AfterScript

x-www-form-urlencoded

form-data

raw

参数名称

参数值

dsKey

B

响应内容

Raw

Headers

Curl

显示说明

响应码: 200 耗时: 1ms

1

{

2

"code": 200,

3

"success": true,

4

"data": null,

5

"msg": "成功"

6

}

状态码

是否成功

承载数据

返回消息

测试查询厦门数据源

GET

/api/test/testReadDynamic

发送

3 请求头部

请求参数

AfterScript

x-www-form-urlencoded

form-data

raw

参数名称

参数值

操作

dsKey

X

删除

响应内容

Raw

Headers

Curl

显示说明

响应码: 200 耗时: 372ms 大小: 5B

1

{

2

"code": 200,

3

"success": true,

4

"data": [],

5

"msg": "操作成功"

6

}

状态码

是否成功

承载数据

返回消息

由于厦门数据源没有插入数据所以为空

测试查询北京数据源

GET

/api/test/testReadDynamic

发送

3 请求头部

请求参数

AfterScript

x-www-form-urlencoded

form-data

raw

参数名称

参数值

操作

dsKey

B

删除

响应内容

Raw

Headers

Curl

显示说明

响应码: 200 耗时: 59ms 大小: 2.08 KB

1

{

2

"code": 200,

3

"success": true,

4

"data": [

5

{

6

"id": 2,

7

"userName": "lsx",

8

"age": 0,

9

"source": "test",

10

"createTime": "2022-09-15 15:11:03",

11

"updateTime": null

12

}

13

]

14

}

状态码

是否成功

承载数据

查询成功

测试插入数据到厦门数据源

GET

/api/test/testWriteDynamic

发送

3 请求头部

请求参数

AfterScript

☒ x-www-form-urlencoded

☐ form-data

☐ raw

<input checked="" type="checkbox"/>	参数名称	参数值	操作
<input checked="" type="checkbox"/>	dsKey	X	删除

响应内容

Raw

Headers

Curl

☒ 显示说明 响应码: 200 耗时: 515ms 大小: 62

1 {

2 "code": 200,

3 "success": true,

4 "data": null,

5 "msg": "成功"

6 }

状态码

是否成功

承载数据

返回消息

然后查询厦门数据源

GET

/api/test/testReadDynamic

发送

3 请求头部

请求参数

AfterScript

☒ x-www-form-urlencoded

☐ form-data

☐ raw

<input checked="" type="checkbox"/>	参数名称	参数值
<input checked="" type="checkbox"/>	dsKey	X

响应内容

Raw

Headers

Curl

☒ 显示说明 响应码: 200 耗时: 515ms 大小: 62

1 {

2 "code": 200,

3 "success": true,

4 "data": [

5 {

6 "id": 42,

7 "userName": "lsx",

8 "age": 0,

9 "source": "zt",

状态码

是否成功

承载数据

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>
3 <artifactId>dynamic-datasource-spring-boot-starter</artifactId>
4 <version>3.5.0</version>
5 </dependency>
```

yaml配置

```
1 spring:
2   datasource:
```

```

3     type: com.alibaba.druid.pool.DruidDataSource
4     #使用dynamicDatasource框架
5     dynamic:
6         #设置默认的数据源或者数据源组
7         primary: XM
8         #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
9         strict: false
10        datasource:
11            XM:
12                url: jdbc:mysql://localhost/sharding_test?
13                useSSL=false&useUnicode=true&characterEncoding=utf-
14                8&zeroDateTimeBehavior=convertToNull&transformedBitIsBoolean=true&tinyInt1isBit=false&a
15                llowMultiQueries=true&serverTimezone=GMT%2B8&allowPublicKeyRetrieval=true
16                username: root
17                password: root
18                driver-class-name: com.mysql.cj.jdbc.Driver
19            BJ:
20                url: jdbc:mysql://xxx/xmkf_zt?
21                useSSL=false&useUnicode=true&characterEncoding=utf-
22                8&zeroDateTimeBehavior=convertToNull&transformedBitIsBoolean=true&tinyInt1isBit=false&a
23                llowMultiQueries=true&serverTimezone=GMT%2B8&allowPublicKeyRetrieval=true&rewriteBatche
24                dStatements=true
25                username: xxx
26                password: xxx
27                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

```

1  @Service
2  @DS("bj")
3  public class ShardingUserServicceImpl extends ServiceImpl<ShardingUserMapper,
4  ShardingUser> implements IShardingUserService{
5      @DS("xm")
6      @Override
7          public boolean saveEntity(ShardingUser entity) {
8              return this.save(entity);
9          }
10     @DS("xm")
11     @Override

```

```
11         public List<ShardingUser> queryList() {
12             return this.list();
13         }
14     }
```

测试代码

```
1  @GetMapping("/testWriteDynamic")
2  @ApiOperation(value = "测试动态数据源插入")
3  @ApiOperationSupport(order = 9, author = "lsx")
4  public R testWriteDynamic(@RequestParam(value = "dsKey",defaultValue = "B") String
    dsKey){
5
6      for (int i = 0; i < 20; i++) {
7          ShardingUser user = new ShardingUser();
8          user.setAge(i);
9          user.setUserName("lsx");
10         user.setCreateTime(new Date());
11         shardingUserService.saveEntity(user);
12     }
13     return R.success("成功");
14 }
15 @GetMapping("/testReadDynamic")
16 @ApiOperation(value = "测试动态数据源读取")
17 @ApiOperationSupport(order = 9, author = "lsx")
18 public R testReadDynamic(@RequestParam(value = "dsKey",defaultValue = "X") String
    dsKey){
19     List<ShardingUser> list = shardingUserService.list();
20     return R.data(list);
21 }
```

上述插入和查询都会去xm的数据源操作