

## 1. swift-DRDS 的 Mybatis 解决方案与规范

在运用 DRDS 分布式数据库时，考虑到使用 DRUID 数据库连接池技术，在配合使用 Mybatis 持久化框架的时候，采用 Mybatis 整合 Spring，配置 Mapper 包扫描器的方式进行开发。

Mybatis 使用范例见：[swif-dao-drds](#) 项目的 [ScoreDemo](#) 相关的实体类、Mapper 接口类和 MapperXML 文件。

### 1.1. Mybatis 整合 Spring

配 Mybatis 整合 Spring 的主要工作包括：添加项目依赖，在 Spring 配置文件中配置管理 Mybatis 需要的 Bean。完成这两部分工作即可。

#### 1.1.1. 添加项目依赖

在已有的 Spring 基础环境的基础上，为了整合 Mybatis，首先需要加入相关的依赖包，包括：mybatis 核心包，spring-mybatis 整合包。

```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.3.0</version>
</dependency>
```

#### 1.1.2. 配置 Mybatis 的 Bean

Mybatis 框架的核心是 SqlSessionFactory，我们将 SqlSessionFactory 交给 Spring 管理，通过依赖注入方式来使用它。为了统一管理，我们创建一个 Spring 的配置文件：application-mybatis.xml 用于管理 Mybatis 相关的 Bean。

##### 1.1.2.1. SqlSessionFactory

我们在配置 SqlSessionFactory 的时候，需要配置两个属性，一个是数据源 DataSource，由于使用了 DRUID 数据库连接池，所以 dataSource 就是 DRUID（前面关于 DRDS 配置中已经进行了说明）；另一个是 configLocation，用于加载 Mybatis 的全局配置文件的属性。实际配置信息如下图：

```

<!-- 配置Mybatis的SqlSessionFactory -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <!-- 加载Mybatis的全局配置文件 -->
    <property name="configLocation" value="classpath:mybatis.xml"/>
</bean>

```

### 1.1.2.2. MapperScannerConfigurer

MapperScannerConfigurer 是 Mybatis 用于自动扫描 MapperXML 文件的扫描器类，它可以自动将指定包路径下 MapperXML 中的 MapperStatement 和 Mapper 接口纳入 Mybatis 的管理中。需要配置的属性是：包路径 basePackage，用于指定被扫描的包含 MapperXML 和 Mapper 接口的包路径，当需要多个包时，使用英文逗号分隔。另一个是 SqlSessionFactoryBeanName，用于注入 SqlSessionFactory。具体配置情况如下图：

```

<!-- 配置Mybatis自动扫描MapperXML文件的扫描器类 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <!-- 指定被扫描的包路径,多个包用半角逗号隔开 -->
    <property name="basePackage" value="swift.data.station.mapper,swift.data.tc.mapper"/>
    <!-- 此处不能使用ref,因为上面的加载会导致报错。必须用value -->
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
</bean>

```

### 1.1.2.3. DataSourceTransactionManager

dataSourceTransactionManager 是 Mybatis 的事务管理器，只需要配置 dataSource 数据源一个属性即可，如下图所示：

```

<!-- 配置Mybatis的事务管理器 -->
<bean id="transactionManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>

```

## 1.2. Mybatis 全局配置文件

在使用 Mybatis 过程中，相关的全局配置信息可以交给 Mybatis 的全局配置文件进行管理。我们将该全局配置文件命名为：mybatis.xml，并且放在 classpath 路径下。

### 1.2.1. 全局配置文件的结构

MyBatis 的 XML 配置文件包含了影响 MyBatis 的设置和属性信息。XML 文档的高层级结构如下图：

- configuration 配置
  - properties 属性
  - settings 设置
  - typeAliases 类型命名
  - typeHandlers 类型处理器
  - objectFactory 对象工厂
  - plugins 插件
  - environments 环境
    - environment 环境变量
      - transactionManager 事务管理器
      - dataSource 数据源
- 映射器

其中的环境配置和映射器配置不需要进一步配置，这是因为在 Mybatis 整合 Spring 的时候已经通过 Spring 配置管理。

configuration 配置可以用于配置一些全局的配置信息，常用的有 Settings 设置、TypeAliases 类型别名等。

### 1.2.1.1. Settings

Settings 可以用于管理 Mybatis 的缓存、超时时间等信息。对此暂时不需要做进一步配置，可以忽略。

### 1.2.1.2. TypeAliases

类型别名是为 Java 类型命名一个短的名字。它只和 MapperXML 配置有关，只用来减少类完全限定名的多余部分，比如：

```
<typeAliases>
  <typeAlias alias="GuandSurfH"
    type="swift.data.station.entity.GuandSurfH"/>
</typeAliases>
```

如此一来在 MapperXML 配置文件中编写 SQL Statement 语句的 resultType、parameterType 设置相应的 Java 类的时候就可以不需要写全类名，简化开发。比如：

```
<select id="get" resultType="GuandSurfH">
  SELECT * FROM t_guand_surf_h${_timeSuffix}
  WHERE ObserveTime=#{ObserveTime} AND StationID=#{StationID}
</select>
```

需要注意的是在配置的时候不要同 resultMap 属性混淆，resultMap 是在 MapperXML 文件中定义的。

下面是一些 Mybatis 定义好的 Java 类及其对应的别名，可以直接使用。

别名	映射的类型
_byte	byte
_long	long
_short	short
_int	int

_integer	int
_double	double
_float	float
_boolean	boolean
string	String
byte	Byte
long	Long
short	Short
int	Integer
integer	Integer
double	Double
float	Float
boolean	Boolean
date	Date
decimal	BigDecimal
bigdecimal	BigDecimal
object	Object
map	Map
hashmap	HashMap
list	List
arraylist	ArrayList
collection	Collection
iterator	Iterator

### 1.2.1.3. Mybatis 全局配置文件示例

Mybatis 全局配置文件本质上是一个 XML 文件，需要在文件头添加相应的信息。示例如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <!-- 配置别名，根据需求可以配置 -->
    <typeAliases>
        <typeAlias alias="GuandSurfH"
            type="swift.data.station.entity.GuandSurfH"/>
    </typeAliases>
    <settings>
        <!-- Mybatis默认没有启动延迟加载 -->
        <setting name="lazyLoadingEnabled" value="false"/>
        <setting name="aggressiveLazyLoading" value="false"/>
    </settings>
</configuration>
```

## 1.3. 接入 DRUID 数据源

在前面接入 DRDS 中已经提到了关于配置 DURID 连接池的说明，也就是通过 Spring 管理 DRUID 的 Bean。

## 1.4. 规定一些开发规范

这部分开发规范包括了 Mybatis 定义的、根据实际开发情况自定义的规范。遵循这样一套规范更有利于开发维护工作。

更多的规范还要进一步补充，可以根据实际应用的需要作进一步的调整。

### 1.4.1. 关于包名

#### 1.4.1.1. 实体类包名

实体类包主要用于存放实体类。

实体类的规范主要集中在统一类型的包命名上。我们定义形如：xxx.yyy.entity 的包用于存放相关的实体类，并且仅仅用于存放实体类。

#### 1.4.1.2. Mapper 包名

Mapper 包名都需要统一纳入 MapperScannerConfigurer 的 Bean 的管理。

Mapper 包统一规定用于存放 Mapper 接口类和 Mapper 接口类对应的 MapperXML 文件。并且我们规定，每个 Mapper 包的命名方式向其对应的实体类包相近，就是说它们唯一同一个上级包下的不同包，如实体类包名为：xxx.yyy.entity，那么 Mapper 包名则是形如：xxx.yyy.mapper 的格式。

### 1.4.2. 关于 MapperXML 和 Mapper 接口类命名

#### 1.4.2.1. MapperXML 文件名和 Mapper 类名

按照 Mybatis 的约定，当实体类名为 Xxx.java 的时候，其对应的 Mapper 接口类名为：XxxMapper.java；其对应的 MapperXML 文件名为：XxxMapper.xml。

#### 1.4.2.2. 命名空间命名

在 MapperXML 中，命名空间 namespace 是唯一的，我们使用该 MapperXML 相对应的 Mapper 接口类全类名作为命名空间。

### 1.4.3. 关于 Mapper 接口类的方法的传入参数

#### 1.4.3.1. 自定义 Criteria 类的使用

我们使用了自定义的一个传入参数信息类 Criteria.java 来接收 SQL 的传入参数信息。本质上，Criteria 是一个映射集合 Map，通过继承了 HashMap 类同时自定义实现了许多实用的方法用于设置传入参数。

在使用 Mybatis 过程中，传入参数除了简单类型和普通 Java 类以外，将 Criteria 规定为唯



一的合法传入参数信息类。当需要使用 Map、List 或者更加复杂的传入参数时，都通过 Criteria 类来进行设置。

允许使用多个 Criteria 作为传入参数，我们通过 Mybatis 提供的注解 @Param("VALUE") 来修饰 Criteria 参数。使得 MapperStatement 语句可以对多个 Criteria 参数进行区别处理。

#### 1.4.3.1.1. 接口方法示例

示例如下图：

```
public UnionResult selectOne(Criteria criteria);

public List<UnionResult> selectList(@Param("criteria1") Criteria criteria1,
                                     @Param("criteria2") Criteria criteria2);
```

#### 1.4.3.1.2. MapperXML 示例

如此一来，就可以在 MapperXML 中通过占位符#{aaa.bbb}或\${aaa.bbb}方式进行参数匹配。示例如下图：

```
<select id="selectList" parameterType="java.util.Map" resultMap="GuandSurfH">
    SELECT StationID, Temperature FROM t_guand_surf_h_201610 WHERE
        <if test="criteria1 != null">
            StationID=#{criteria1.stationID}
        </if>
    UNION ALL
    SELECT StationID, Temperature FROM t_guand_surf_h_201610 WHERE
        <if test="criteria2 != null">
            StationID=#{criteria2.stationID}
        </if>
</select>
```

#### 1.4.3.2. 封装对象类 JavaBean 的使用

我们根据业务需求，除了使用自定义的 Criteria 类之外，定义了封装输入参数信息的 JavaBean 对象。直接将对象作为输入参数，由 Mybatis 的 OGNL 支持可以自动映射相应的参数占位符。

为了便于区别输入参数信息类和数据表实体类的区别，我们统一规定其命名方式为：“实体类名+“Example””作为输入参数信息类的类名。如下图 Mapper 接口类方法中的例子：

```
public List<ScoreDemo> avgOfHighTemperature(@Param("criteria") ScoreDemoExample example);
```

##### 1.4.3.2.1. 接口示例方法

如下图所示，通过 @Param 注解修饰让该对象等同于使用 Criteria 自定义类，不同在于，Mybatis 映射 Criteria (本质上是一个 HashMap) 是通过映射该自定义的 Key 去寻找相应的 Value，而对于 JavaBean 对象，Mybatis 则是通过 SetterGetter 方法去查找相应的值。

```
public List<ScoreDemo> rateOfHeavyRain(@Param("criteria") ScoreDemoExample example);  
public List<ScoreDemo> rateOfHeavyRain(@Param("criteria") Criteria criteria);
```

#### 1.4.3.2.2. MapperXML 示例

由于 Mybatis 对 OGNL 的支持，配合 @Param 注解，对于 JavaBean 的输入参数，MapperXML 中的定义可以更加简单，如下图：

```
<!-- ##### 暴雨 NA / (NA+NB+NC) 子句测试 ##### -->  
<select id="heavyRain" resultMap="ScoreResult">  
    <include refid="Heavy_Rain"/>  
</select>
```

可以看到，输入参数 parameterType 都可以直接省略不写。

#### 1.4.4. 关于占位符、元素 Id 的命名

由于 Mybatis 的属性占位符支持 OGNL，所以可以通过英文句点 “.” 的方式映射 Java Bean 对象或者 Map 的 key 的对应值。如：#{criteria.location.province}，\${criteria.location.province} 等。

##### 1.4.4.1. 在 XML 中设定值的占位符的命名

由于 SQL 片段中的占位符可以直接在引用在片段的时候定义值，于是我们统一规定，这一类在引用的时候才定义的占位符统一使用下划线开头的命名方式，形如：\_scoreFunction。

以下为示例：

被引用的片段如下：

```
<sql id="Base_Select_SQL_Except_Typhoon_Heavy_Rain_Or_Not">  
    SELECT ${_forecasterIdOrForecastDate}, ${_scoreFunction} Score,ItemId  
    <where>  
        JobId  
        <foreach collection="criteria.jobIds" item="jobId" open="IN(" clo
```

引用 SQL 的语句定义如下：

```
<include refid="Base Select SQL Except Typhoon Heavy Rain Or Not">  
    <property name="_scoreFunction" value="ROUND(avg(Score),3)"/>  
    <property name="_forecasterIdOrForecastDate" value="ForecasterId"/>  
    <property name="_itemId" value="2"/>  
</include>
```


有一点需要注意的是，在定义占位符属性值的时候，存在一个值的作用域的问题。比如说在嵌套引用的关系中，都对同一个占位符\${..}赋值。

这个时候，Mybatis 将会按照就近原则，从最近的嵌套元素中匹配相应的值，如下图所示，同一个属性设置两次，起作用的将会是由红色下划线标识的部分。

```

<sql id="Wind_Direct_SQL">
  <include refid="Base_Select_SQL_Except_Typhoon_Heavy_Rain_Or_Not">
    <property name="_scoreFunction" value="IFNULL(ROUND(sum(Ext1)/sum(Ext1),2),0)>
    <property name="_forecatorIdOrForecastDate" value="ForecatorId"/>
    <property name="_itemId" value="6"/>
  </include>
</sql>
<!-- ##### 风向 nacd /nt 子句测试 ##### -->
<select id="windDirect" resultMap="ScoreResult">
  <include refid="Wind_Direct_SQL">
    <property name="_forecatorIdOrForecastDate" value="ForecastDate"/>
  </include>
</select>

```



由此，我们也规定：对于这类属性占位符的赋值，尽可能在嵌套引用的最外层处理，最理想的状态是在<select>等 MapperStatement 中在进行赋值，这样的 SQL 片段更加有利于提高重用性。具体应用场景需要根据该占位符的作用来灵活地选择方案。

#### 1.4.4.2. 输入参数占位符命名

我们统一规定，采用驼峰式方式为属性占位符命名，即：以字母开头，并且首个单词首字母小写其他单词首字母大写的方式，如：#{departmentId}或\${departmentId}。

对于复杂的属性映射，在每个占位符中使用英文句点分隔，命名方式不因存在该句点而变化，如：#{criteria.departmentId}或\${criteria.departmentId}。

#### 1.4.4.3. <sql>元素 Id 的命名

为了增加辨识度，我们统一规定，所有的<sql>元素的 Id 命名均采用以下划线分隔、首字母大写的命名方式。如：

```
<sql id="Base_Union_Rate_Of_Rain_Or_Not">
```

当该条 SQL 包含较为完整的 SELECT/UPDATE、FROM、WHERE 等要素以致足以构成一条可用 SQL 的时候，就用“\_SQL”作为元素 Id 的结尾。如：

```
<sql id="Typhoon_72H_SQL">
```

#### 1.4.4.4. MapperStatement 的 Id 的命名

MapperStatement 就是 Mybatis 生成的用于执行的 SQL 语句所在的元素节点。也就是<select>,<update>,<delete>,<insert>元素的内容。其 Id 值就是对应的 MapperStatement 的 Id。

按照规定，每一条 MapperStatement 的 Id 都在对应的 Mapper 接口类中有相应的同名方法。所以我们在命名 MapperStatement 的时候，同样规定采用驼峰式的命名方式。

值得注意的是，我们可以在 Mapper 接口类中重载这些方法，Mybatis 只要形参中定义的输入参数最终能够匹配该 MapperStatement 中所需要的属性占位符即可。例子如下：



```
public interface ScoreDemoMapper {

    public List<ScoreDemo> rateOfHeavyRain(@Param("criteria") ScoreDemoExample example);
    public List<ScoreDemo> rateOfHeavyRain(@Param("criteria") Criteria criteria);
}
```

## 1.4.5. 关于 SqlSessionFactory 的使用

### 1.4.5.1. 注入 SqlSessionFactory 依赖

在开发 DAO 层的时候,主要是通过使用 SqlSessionFactory 来创建 SqlSession,再由 SqlSession 去完成数据库相关的操作。对于 SqlSessionFactory 的使用,主要推荐两种使用方式。

其一: 通过 Setter 注入方式将其作为数据库操作基类的一个属性,如下图所示:

```
public class ScoreDetailRepository {

    SqlSessionFactory sqlSessionFactory;
    public void setSqlSessionFactory(SqlSessionFactory sqlSessionFactory) {
        this.sqlSessionFactory = sqlSessionFactory;
    }
}
```

其二: 通过注解方式依赖注入,如下图所示:

```
public class ScoreDetailRepository {

    @Autowired
    SqlSessionFactory sqlSessionFactory;
}
```

两者相比较而言, Setter 注入方式需要在依赖 SqlSessionFactory 的类中配置 sqlSessionFactory 的属性,而注解方式不需要。但是从单元测试角度而言, Setter 注入方式更加方便。

### 1.4.5.2. SqlSession 的使用

我们已经知道,操作 Mybatis 执行 SQL 语句的关键是通过 SqlSession。在这里,关于 SqlSession 的使用主要有以下几点需要特别注意:

1. SqlSession 的作用域必须控制在一个方法内而不是类的全局。如下图:

```
@Test
public void testRateOfRainOrNot() {
    try(SqlSession sqlSession = sqlSessionFactory.openSession()) {
```

2. SqlSession 是通过 `sqlSessionFactory.openSession(boolean autoCommit)` 创建的,当形参为空时默认是 false,但是当需要操作的 SQL 涉及数据的变化时,就需要显性地将形参设置为 true。

3. 建议使用:

```
try(SqlSession sqlSession = sqlSessionFactory.openSession()) { //coding here }
```

的方式作为 SqlSession 的创建方式。

## 1.5. MapperXML 的基本使用

建议查看 swift-dao-drds 中的范例（ScoreDemoMapper.xml 文件）的使用。

### 1.5.1. select

如下图示例，这是最基础的一条 SELECT 查询语句，该语句 id 为“get”，需要跟 Mapper 接口类的一个方法名相同。输入参数 parameterType 为“GuandSurfH”，这里再次强调 parameterType、resultType 的值是在全局配置文件中定义的别名。而返回结果集的“GuandSurfH”，则是在 MapperXML 文件中定义的。

```
<select id="get" parameterType="GuandSurfH" resultMap="GuandSurfH">
    SELECT * FROM t_guand_surf_h${_timeSuffix}
    WHERE ObserveTime=#{ObserveTime} AND StationID=#{StationID}
</select>
```

#### 参数占位符说明

#{ObserveTime}

这就告诉 MyBatis 创建一个 PreparedStatement（预处理语句）参数。使用 JDBC，这样的参数在 SQL 中会由一个“?”来标识，并被传递到一个新的预处理语句中。

\${\_timeSuffix}

这是一个拼接字符串占位符，其参数会直接作为一个字符串拼接 SQL 语句中去。

就像下面这样：

// 相似的 JDBC 代码

```
String get="SELECT * FROM t_guand_surf_h201610 WHERE ObserveTime=? AND StationID=?";
```

```
PreparedStatement ps = conn.prepareStatement(get);
```

```
// .....
```

### 1.5.2. insert、update、delete

数据修改语句 insert，update 和 delete 在它们的实现中非常相似。

```
<insert id="create" parameterType="map">
    INSERT INTO t_guand_surf_h${_timeSuffix} VALUES (
        StationID=#{StationID}, ObserveTime=#{ObserveTime}, Wind
    )
</insert>
```

```
<update id="update" parameterType="map">
    UPDATE t_guand_surf_h${_timeSuffix} SET ObserveTime=#{ObserveTime}, Wind=#{Wind}
    WHERE StationID=#{StationID}
</update>
```

```
<delete id="delete" parameterType="map">
    DELETE FROM t_guand_surf_h${_timeSuffix}
    WHERE StationID=#{StationID}
</delete>
```

### 1.5.3. sql

sql 这个元素可以被用来定义可重用的 SQL 代码段，可以包含在其他语句中。如下图示例：

```
<sql id="select_Station_WindSpeed">
    SELECT StationID, WindSpeed FROM t_guand_surf_h_${yyyyMM}
    WHERE ObserveTime > #{Min_ObserveTime}
    AND ObserveTime < #{Max_ObserveTime}
</sql>
```

在定义好 sql 元素的片段之后，可以在 MapperXML 范围内引用该 SQL 片段，方法是使用 include 元素，如下图：

```
<include refid="select_Station_WindSpeed"/>
```

该元素的内容将作为一个完整的语句片段被包含在其他的语句中，并且支持 sql 元素的嵌套引用。可以结合 foreach 元素设计开发复杂的动态 SQL 语句，如多表 UNION、JOIN 查询等。

### 1.5.4. resultMap

resultMap 是定义在 MapperXML 文件中的用于映射 SQL 返回结果的自定义集合。可以根据需求自定义查询结果集中的列名与 JavaBean 实体类的映射关系。

## 1.6. 一些开发维护的建议

为了应对多变的需求的开发维护，总结一些开发新功能、维护改进旧功能的小技巧。这是建立在基础环境搭建完成的前提下的。提供了一些简单的开发步骤说明，更复杂的场景还需要进一步思考选择，不必拘泥。

### 1.6.1. 开发新需求

开发新需求主要指的是，需要重新开始一个子模块功能（诸如：station、typhoon、score 都单独认为是一个子模块功能类）的开发。我们推荐按照以下大致步骤进行。

#### 1.6.1.1. 创建实体类

从数据库表作为出发点，首先创建命名新的包名用于存放数据表对应的实体类，然后根据数据表编写设计实体类。Setter 和 Getter 方法是必不可少的。

#### 1.6.1.2. 创建 Mapper 接口和 MapperXML

首先根据前面提到的命名规范，创建 Mapper 的包用于存放 Mapper 接口和 MapperXML 文件。然后根据实体类的情况创建相应的 XxxMapper 接口类和 XxxMapper.xml 文件。



### 1.6.1.3. 加到 Mybatis 扫描器

找到 Mybatis 相关的 Spring 的配置文件，找到 Mybatis 包扫描器 `MapperScannerConfigurer`，在其属性 `basePackage` 值中加入新增的 `Mapper` 接口和 `MapperXML` 所在包的包名，多个包使用英文逗号分隔，如下图：

```
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
  <!-- 指定被扫描的包路径,多个包用半角逗号隔开 -->
  <property name="basePackage"
    value="swift.data.station.mapper,swift.data.tc.mapper,swift.data.score.mapper"/>
  <!-- 此处不能使用ref,因为上面的加载会导致报错。必须用value -->
  <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
</bean>
```

### 1.6.1.4. 编写 MapperXML

首先，每一个 Mybatis 的 `MapperXML` 文件都必须有以下的文档头：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
```

接着，在根节点 `<mapper>` 中需要设置命名空间属性 `namespace`，在命名方式规范中有提到。

然后，首先拿到需要运行的 SQL 语句，根据 SQL 语句返回结果集考虑如何选择定义 `ResultMap`。

在 `ResultMap` 中，其 `type` 属性值建议仍然使用其对应的实体类的全类名，除非返回结果集在对应的实体类中已经无法适用，否则不建议重新创建定义新的实体类去接收返回结果对象。

如下图，需要返回的结果集不同，但是仍然可以基于相同的实体类，那些没有被映射的字段将会是默认值：

```
<resultMap type="swift.data.score.entity.ScoreDemo" id="ScoreResult">
  <result column="ForecasterId" property="forecasterId"/>
  <result column="DepartmentId" property="departmentId"/>
  <result column="Score" property="score"/>
  <result column="ItemId" property="itemId"/>
</resultMap>
<resultMap type="swift.data.score.entity.ScoreDemo" id="ProvinceStatisticResult">
  <result column="ForecasterId" property="forecasterId"/>
  <result column="Score" property="score"/>
  <result column="ItemId" property="itemId"/>
  <result column="DepartmentId" property="departmentId"/>
  <result column="ForecastDate" property="forecastDate"/>
</resultMap>
```

## 1.6.2. 维护原有业务

维护原有业务主要包括：在原有的基础上进行的功能修改、删减、增加等操作。而且是有 SQL 语句变化需求的操作。

具体说就是在 SQL 语句层面的维护，业务层的维护修改不涉及 SQL 语句故而不属于该范畴。这一部分需要功能需求入手，主要说明了需要注意的事项。



### 1.6.2.1. 功能修改、删减

对于功能修改删减而言，我们可以从被调用的 Mapper 接口类的方法作为起点，在对应的 MapperXML 中找到与该方法名同名的 MapperStatement，通过修改 MapperStatement 对功能进行修改。

如果 MapperStatement 中包含了引用语句，尽可能不要改动被引用的 SQL 片段，你不知道哪里还有引用，确实需要修改引用语句的，你也要确保所有引用该片段的语句功能变化是受你控制的。

建议是首先考虑新增一个 SQL 片段去适应新的功能需求，弃用原来引用的 SQL 片段。如果功能改动很大，请酌情选择涉及的相关引用了该 SQL 片段的语句。

### 1.6.2.2. 功能新增

功能新增主要场景有：当需要加入新的 SQL 语句或者需要在原有的 SQL 语句中加入某些查询列、查询条件、排序条件等内容。

对于加入新的 SQL 语句，可以直接创建一条新的 SQL 片段，查找原有的基础 SQL 片段中是否有可重用的，引入进来使用即可。

对于需要在原有 SQL 语句中加入一些列、条件的场景，主要有两种解决办法。

其一：在原有的 SQL 语句中，通过拼接字符串的占位符\${...}，然后在引用该 SQL 的地方对这些占位符进行赋值（如加入列名、查询条件等）即可。

其二：重新创建新的 SQL 片段，更换原来的 MapperStatement 的引用。

可以根据应用场景灵活选择一种方案。

## 1.7. 可能遇到的问题

### 1.7.1. 集合类转换异常问题

报错信息：

org.apache.ibatis.exceptions.PersistenceException:

### Error querying database. Cause: java.lang.IllegalArgumentException: invalid comparison: java.util.Arrays\$ArrayList and java.lang.String

### Cause: java.lang.IllegalArgumentException: invalid comparison: java.util.Arrays\$ArrayList and java.lang.String

说明：

在使用 Mybatis 编写 MapperXML 中，我们已经知道可以通过

```
<if test="criteria.itemIds != null and criteria.itemIds != ''">
```

的方式判断输入参数是否为 null 等。

但是如果被判断的是一个集合对象，当使用如上判断方式就会出错。因为集合无法判断是否不等于空字符串。当输入参数不是集合类，也就不存在该问题。

解决方法：

去掉判断语句中的集合不等于空字符串的判断即可。

## 1.7.2. 无法映射结果集问题

报错信息：

org.apache.ibatis.exceptions.PersistenceException:

### Error querying database. Cause: org.apache.ibatis.executor.ExecutorException: A query was run and no Result Maps were found for the Mapped Statement 'swift.data.score.mapper.ScoreDemoMapper.getTsScoreProvinceStatisticor'. It's likely that neither a Result Type nor a Result Map was specified.

说明：

如果漏写了 ResultType 或者 resultMap（二者不可同时使用），将会出现该报错信息。

解决方法：

只需要在 MapperStatement 中补上 ResultType 或者 resultMap 即可。

## 1.7.3. 多个参数映射失败问题

报错信息：

org.apache.ibatis.exceptions.PersistenceException:

### Error querying database. Cause: org.apache.ibatis.binding.BindingException: Parameter 'criteria' not found. Available parameters are [1, jobType, param1, param2]

### Cause: org.apache.ibatis.binding.BindingException: Parameter 'criteria' not found. Available parameters are [1, jobType, param1, param2]

说明：

当 Mapper 接口类中的方法是多个参数的时候，需要添加 @Param 注解修饰参数，因为 Mybatis 会默认以 “param1, param2, ...” 的形式去解析输入参数，参数名并没有实际意义。

更简单的情况是，当只有一个输入参数时，类似的报错是值没有定义该 Parameter。

解决方法：

为每个输入参数都用 @Param 注解定义其输入参数映射值 Parameter 即可避免该问题。或者检查提示的 Parameter 是否已经定义。

## 1.7.4. 符号格式错误问题

报错信息：

org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'scoreDemoMapper' defined in file [D:\qxt\swift-edas-2\swift-dao-drds\target\classes\swift\data\score\mapper\ScoreDemoMapper.class]: Invocation of init method failed; nested exception is java.lang.IllegalArgumentException: org.apache.ibatis.builder.BuilderException: Error creating document instance. Cause: org.xml.sax.SAXParseException; lineNumber: 31; columnNumber: 22; 元素内容必须由格式正确的字符数据或标记组成。

说明：

这个报错信息指出 XML 格式错误，可能是某些节点没有闭合等，但是很大程度是因为使用了诸如 “<”、“>” 等影响 XML 解析的字符。

解决方法：

对这类型的字符需要使用格式转换符替换，如小于号 “<” 使用 “&lt;” 代替，大于号 “>” 使用 “&gt;” 代替。常见的几种替换符如下：

&lt;	<	小于
------	---	----

&gt;	>	大于
&amp;	&	和号
&apos;	'	省略号
&quot;	"	引号

还可以使用 **CDATA** 的形式替换特殊的字符，即使用：“<![CDATA[”开始，由“]]>”结束的形式避免其中的内容被 XML 解析。如下图：

```
<![CDATA[ AND ForecastTime >= #{criteria.startTime} ]]>
```