# 动态 SQL

MyBatis 的强大特性之一便是它的动态 SQL。如果你有使用 JDBC 或其他类似框架的经验，你就能体会到根据不同条件拼接 SQL 语句有多么痛苦。拼接的时候要确保不能忘了必要的空格，还要注意省掉列名列表最后的逗号。利用动态 SQL 这一特性可以彻底摆脱这种痛苦。

通常使用动态 SQL 不可能是独立的一部分,MyBatis 当然使用一种强大的动态 SQL 语言来改进这种情形,这种语言可以被用在任意的 SQL 映射语句中。

动态 SQL 元素和使用 JSTL 或其他类似基于 XML 的文本处理器相似。在 MyBatis 之前的版本中,有很多的元素需要来了解。MyBatis 3 大大提升了它们,现在用不到原先一半的元素就可以了。MyBatis 采用功能强大的基于 OGNL 的表达式来消除其他元素。

1. If
2. Choose(when ,otherwise)
3. Trim(when ,set)
4. foreach

### 

### **if**

动态 SQL 通常要做的事情是有条件地包含 where 子句的一部分。比如:

<select id="findActiveBlogWithTitleLike"

resultType="Blog">

SELECT \* FROM BLOG

WHERE state = ‘ACTIVE’

<if test="title != null">

AND title like #{title}

</if></select>

这条语句提供了一个可选的文本查找类型的功能。如果没有传入“title”，那么所有处于“ACTIVE”状态的BLOG都会返回；反之若传入了“title”，那么就会把模糊查找“title”内容的BLOG结果返回（就这个例子而言，细心的读者会发现其中的参数值是可以包含一些掩码或通配符的）。

如果想可选地通过“title”和“author”两个条件搜索该怎么办呢？首先，改变语句的名称让它更具实际意义；然后只要加入另一个条件即可。

<select id="findActiveBlogLike"

resultType="Blog">

SELECT \* FROM BLOG WHERE state = ‘ACTIVE’

<if test="title != null">

AND title like #{title}

</if>

<if test="author != null and author.name != null">

AND author\_name like #{author.name}

</if></select>

### **choose, when, otherwise**

有些时候，我们不想用到所有的条件语句，而只想从中择其一二。针对这种情况，MyBatis 提供了 choose 元素，它有点像 Java 中的 switch 语句。

还是上面的例子，但是这次变为提供了“title”就按“title”查找，提供了“author”就按“author”查找，若两者都没有提供，就返回所有符合条件的BLOG（实际情况可能是由管理员按一定策略选出BLOG列表，而不是返回大量无意义的随机结果）。

<select id="findActiveBlogLike"

resultType="Blog">

SELECT \* FROM BLOG WHERE state = ‘ACTIVE’

<choose>

<when test="title != null">

AND title like #{title}

</when>

<when test="author != null and author.name != null">

AND author\_name like #{author.name}

</when>

<otherwise>

AND featured = 1

</otherwise>

</choose></select>

### **trim, where, set**

前面几个例子已经合宜地解决了一个臭名昭著的动态 SQL 问题。现在考虑回到“if”示例，这次我们将“ACTIVE = 1”也设置成动态的条件，看看会发生什么。

<select id="findActiveBlogLike"

resultType="Blog">

SELECT \* FROM BLOG

WHERE

<if test="state != null">

state = #{state}

</if>

<if test="title != null">

AND title like #{title}

</if>

<if test="author != null and author.name != null">

AND author\_name like #{author.name}

</if></select>

如果这些条件没有一个能匹配上将会怎样？最终这条 SQL 会变成这样：

SELECT \* FROM BLOG

WHERE

这会导致查询失败。如果仅仅第二个条件匹配又会怎样？这条 SQL 最终会是这样:

SELECT \* FROM BLOG

WHERE

AND title like ‘someTitle’

这个查询也会失败。这个问题不能简单的用条件句式来解决，如果你也曾经被迫这样写过，那么你很可能从此以后都不想再这样去写了。

MyBatis 有一个简单的处理，这在90%的情况下都会有用。而在不能使用的地方，你可以自定义处理方式来令其正常工作。一处简单的修改就能得到想要的效果：

<select id="findActiveBlogLike"

resultType="Blog">

SELECT \* FROM BLOG

<where>

<if test="state != null">

state = #{state}

</if>

<if test="title != null">

AND title like #{title}

</if>

<if test="author != null and author.name != null">

AND author\_name like #{author.name}

</if>

</where></select>

where 元素知道只有在一个以上的if条件有值的情况下才去插入“WHERE”子句。而且，若最后的内容是“AND”或“OR”开头的，where 元素也知道如何将他们去除。

如果 where 元素没有按正常套路出牌，我们还是可以通过自定义 trim 元素来定制我们想要的功能。比如，和 where 元素等价的自定义 trim 元素为：

<trim prefix="WHERE" prefixOverrides="AND |OR ">

... </trim>

prefixOverrides 属性会忽略通过管道分隔的文本序列（注意此例中的空格也是必要的）。它带来的结果就是所有在 prefixOverrides 属性中指定的内容将被移除，并且插入 prefix 属性中指定的内容。

类似的用于动态更新语句的解决方案叫做 set。set 元素可以被用于动态包含需要更新的列，而舍去其他的。比如：

<update id="updateAuthorIfNecessary">

update Author

<set>

<if test="username != null">username=#{username},</if>

<if test="password != null">password=#{password},</if>

<if test="email != null">email=#{email},</if>

<if test="bio != null">bio=#{bio}</if>

</set>

where id=#{id}</update>

这里，set 元素会动态前置 SET 关键字，同时也会消除无关的逗号，因为用了条件语句之后很可能就会在生成的赋值语句的后面留下这些逗号。

若你对等价的自定义 trim 元素的样子感兴趣，那这就应该是它的真面目：

<trim prefix="SET" suffixOverrides=",">

...</trim>

注意这里我们忽略的是后缀中的值，而又一次附加了前缀中的值。

### **foreach**

动态 SQL 的另外一个常用的必要操作是需要对一个集合进行遍历，通常是在构建 IN 条件语句的时候。比如：

<select id="selectPostIn" resultType="domain.blog.Post">

SELECT \*

FROM POST P

WHERE ID in

<foreach item="item" index="index" collection="list"

open="(" separator="," close=")">

#{item}

</foreach></select>

foreach 元素的功能是非常强大的，它允许你指定一个集合，声明可以用在元素体内的集合项和索引变量。它也允许你指定开闭匹配的字符串以及在迭代中间放置分隔符。这个元素是很智能的，因此它不会偶然地附加多余的分隔符。

**注意** 你可以将任何可迭代对象（如列表、集合等）和任何的字典或者数组对象传递给foreach作为集合参数。当使用可迭代对象或者数组时，index是当前迭代的次数，item的值是本次迭代获取的元素。当使用字典（或者Map.Entry对象的集合）时，index是键，item是值。

到此我们已经完成了涉及 XML 配置文件和 XML 映射文件的讨论。下一部分将详细探讨 Java API，这样才能从已创建的映射中获取最大利益。

### **bind**

bind 元素可以从 OGNL 表达式中创建一个变量并将其绑定到上下文。比如：

<select id="selectBlogsLike" resultType="Blog">

<bind name="pattern" value="'%' + \_parameter.getTitle() + '%'" />

SELECT \* FROM BLOG

WHERE title LIKE #{pattern}</select>

### **Multi-db vendor support**

一个配置了“\_databaseId”变量的 databaseIdProvider 对于动态代码来说是可用的，这样就可以根据不同的数据库厂商构建特定的语句。比如下面的例子：

<insert id="insert">

<selectKey keyProperty="id" resultType="int" order="BEFORE">

<if test="\_databaseId == 'oracle'">

select seq\_users.nextval from dual

</if>

<if test="\_databaseId == 'db2'">

select nextval for seq\_users from sysibm.sysdummy1"

</if>

</selectKey>

insert into users values (#{id}, #{name})</insert>

### **动态 SQL 中可插拔的脚本语言**

MyBatis 从 3.2 开始支持可插拔的脚本语言，因此你可以在插入一种语言的驱动（language driver）之后来写基于这种语言的动态 SQL 查询。

可以通过实现下面接口的方式来插入一种语言：

public interface LanguageDriver {

ParameterHandler createParameterHandler(MappedStatement mappedStatement, Object parameterObject, BoundSql boundSql);

SqlSource createSqlSource(Configuration configuration, XNode script, Class<?> parameterType);

SqlSource createSqlSource(Configuration configuration, String script, Class<?> parameterType);}

一旦有了自定义的语言驱动，你就可以在 mybatis-config.xml 文件中将它设置为默认语言：

<typeAliases>

<typeAlias type="org.sample.MyLanguageDriver" alias="myLanguage"/></typeAliases><settings>

<setting name="defaultScriptingLanguage" value="myLanguage"/></settings>

除了设置默认语言，你也可以针对特殊的语句指定特定语言，这可以通过如下的 lang 属性来完成：

<select id="selectBlog" lang="myLanguage">

SELECT \* FROM BLOG</select>

或者在你正在使用的映射中加上注解 @Lang 来完成：

public interface Mapper {

@Lang(MyLanguageDriver.class)

@Select("SELECT \* FROM BLOG")

List<Blog> selectBlog();