

## 1.动态SQL简介

**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。

```
Session session = this.getSession();
StringBuffer sb = new StringBuffer("select count(id) from House where 1=1");
if (title != null) {
    sb.append(" and title like '%" + title + "%'");
}
if (startTime != null) {
    sb.append(" and createdate>='" + DateUtils.convert(startTime) + "'");
}
if (endTime != null) {
    sb.append(" and createdate<='" + DateUtils.convert(endTime) + "'");
}
```

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：SQL语句可以根据用户的条件动态的生成和变化*

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**（**Struts2**语法）的表达式来消除其他元素。

**mybatis** 的动态sql语句是基于OGNL表达式的。可以方便的在 sql 语句中实现某些逻辑. 总体说来mybatis 动态SQL 语句主要有以下几类：

- **if** 语句 (简单的条件判断)
- **choose (when,otherwise)** ,相当于java 语言中的 switch ,与 jstl 中的 choose 很类似.
- **trim** (对包含的内容加上 prefix,或者 suffix 等，前缀，后缀)
- **where** (主要是用来简化sql语句中where条件判断的，能智能的处理 and or ,不必担心多余导致语法错误)
- **set** (主要用于更新时)
- **foreach** (在实现 mybatis in 语句查询时特别有用)

## 2.分支判断

### 2.1 if元素

根据 **username** 和 **sex** 来查询数据。如果**username**为空，那么将只根据**sex**来查询；反之只根据**username**来查询  
首先不使用 动态SQL 来书写

```
<select id="selectUserByUsernameAndSex" resultType="user"
parameterType="User">
    select * from user where username=#{username} and
sex=#{sex}
</select>
```

上面的查询语句，我们可以发现，如果 `#{username}` 为空，那么查询结果也是空，而不是报错，因为MySQL会认为传递的username的值为null（Java中字段的默认值），如何解决这个问题呢？使用 if 来判断

```
<select id="selectUserByUsernameAndSex"
resultType="user"
parameterType="User">
  select * from user where
  <if test="username != null and username != ''">
    username=#{username}
  </if>
  <if test="sex!= null and sex != ''">
    and sex=#{sex}
  </if>
</select>
```

这样写我们可以看到，如果 sex 等于 null，那么查询语句为 `select * from user where username=#{username}`，但是如果username 为空呢？那么查询语句为 `select * from user where and sex=#{sex}`，这是错误的 SQL 语句，如何解决呢？在这里有两种解决方式：

1. 将上面的SQL语句改为：

```
select * from user where 1 = 1
  <if test="username != null and username
!= ''">
    and username=#{username}
  </if>
  <if test="sex!= null and sex != ''">
    and sex=#{sex}
  </if>
```

2. 使用2.2节中的方式

## 2.2 动态SQL:if+where 语句

这个组合的作用：去除多余的AND或OR

```
<select id="selectUserByUsernameAndSex" resultType="User"
parameterType="User">
  select * from user
  <where>
    <if test="username != null">
      username=#{username}
    </if>
    <if test="sex != null">
      and sex=#{sex}
    </if>
  </where>
</select>
```

这个“where”标签会知道如果它包含的标签中有返回值的话，它就插入一个‘where’。如果where之后有多余的AND或OR，就会剔除掉这个AND或OR。

## 2.3 动态SQL:if+set 语句

这个组合的作用：去除多余的逗号，同时防止更新的时候表中的字段被设置为null

同理，上面的对于查询 SQL 语句包含 where 关键字，如果在进行更新操作的时候，含有 set 关键词，我们怎么处理呢？

```
<!-- 根据 id 更新 user 表的数据 -->
<update id="updateUserById" parameterType="User">
    update user u
    <set>
        <if test="username != null and username !=
        ''">
            u.username = #{username},
        </if>
        <if test="sex != null and sex != ''">
            u.sex = #{sex},
        </if>
    </set>
    where id=#{id}
</update>
```

这样写，如果第一个条件 username 为空，那么 sql 语句为：update user u set u.sex=? where id=?

如果第一个条件不为空，那么 sql 语句为：update user u set u.username = ?, u.sex = ? where id=?

如果出现 多余逗号 会自动去掉！！

## 2.4 动态SQL:choose(when,otherwise) 语句 了解

有时候，我们不想用到所有的查询条件，只想选择其中的一个，查询条件有一个满足即可，使用 choose 标签可以解决此类问题，类似于 Java 的 switch 语句

```
<select id="selectUserByChoose" resultType="User"
parameterType="User">
    select * from user
    <where>
        <choose>
            <when test="id != '' and id != null">
                id=#{id}
            </when>
            <when test="username != '' and username !=
            null">
                and username=#{username}
            </when>
            <otherwise>
                and sex=#{sex}
            </otherwise>
        </choose>
    </where>
</select>
```

```

        </otherwise>
    </choose>
</where>
</select>

```

也就是说，这里我们有三个条件，id,username,sex，只能选择一个作为查询条件；

上述动态SQL的执行情况：

如果 id 不为空，那么查询语句为：select \* from user where id=?;

如果 id 为空，那么看username 是否为空，如果不为空，那么语句为 select \* from user where username=?;

如果 username 为空，那么查询语句为 select \* from user where sex=?;

## 2.5 动态SQL:trim 语句 了解

trim标记是一个格式化的标记，可以完成set或者是where标记的功能

①、用 trim 改写上面第二点的 if+where 语句

```

<select id="selectUserByUsernameAndSex"
resultType="user" parameterType="User">
    select * from user
    <trim prefix="where" prefixOverrides="and | or">
        <if test="username != null">
            and username=#{username}
        </if>
        <if test="sex != null">
            and sex=#{sex}
        </if>
    </trim>
</select>

```

prefix: 前缀

prefixoverride: 去掉第一个and或者是or

②、用 trim 改写上面第三点的 if+set 语句

```

<!-- 根据 id 更新 user 表的数据 -->
<update id="updateUserById" parameterType="User">
    update user u
    <trim prefix="set" suffixOverrides=",">
        <if test="username != null and username !=
        ''">
            u.username = #{username},
        </if>
        <if test="sex != null and sex != ''">
            u.sex = #{sex},
        </if>
    </trim>
    where id=#{id}
</update>

```

suffix: 后缀

suffixoverride: 去掉最后一个逗号（也可以是其他的标记，就像是上面前缀中的and一样）

## 2.6 动态SQL: SQL 片段

有时候可能某个 sql 语句我们用的特别多，为了增加代码的重用性，简化代码，我们需要将这些代码抽取出来，然后使用时直接调用。

比如：假如我们需要经常根据用户名和性别来进行联合查询，那么我们就把这个代码抽取出来，如下：

```

<!-- 定义 sql 片段 -->
<sql id="selectUserByUsernameAndSexSQL">
    <if test="username != null and username != ''">
        AND username = #{username}
    </if>
    <if test="sex != null and sex != ''">
        AND sex = #{sex}
    </if>
</sql>

```

引用 sql 片段

```

<select id="selectUserByUsernameAndSex" resultType="user"
    parameterType="User">
    select * from user
    <trim prefix="where" prefixOverrides="and | or">
        <!-- 引用 sql 片段，如果refid 指定的不在本文件中，那
        么需要在前面加上 namespace -->
        <include
        refid="selectUserByUsernameAndSexSQL"></include>
        <!-- 在这里还可以引用其他的 sql 片段 -->
    </trim>
</select>

```

注意：

- ①、最好基于 单表来定义 sql 片段，提高片段的可重用性
- ②、在 sql 片段中不要包括 where

## 2.7 动态SQL: foreach 语句

### 2.7.1 语法

foreach元素中的属性：

1. **item**：配置的是循环中的当前元素，这个元素的值是什么，那么在循环中使用的#{xxx}中的xxx就是什么。
2. **index**：指的是当前元素在集合的位置下标；
3. **collection**：指的是传递到这个中的参数类型（首字母小写），它可以是：array、list（或collection）、Map集合的键的名字、POJO包装类中数组或集合类型的属性名字。

注意：

1. 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作为参数。当你使用可迭代对象或数组时，**index**是当前迭代次数，**item**是本次迭代获取的元素；当你使用字典（Map.Entry对象的集合）时，**index**是键，**item**是值。
2. 再使用时最关键也是最重要的是**collection**属性，该属性是必须指定的，而且在不同情况下，该属性的值是不一样的，主要有以下三种情况：
  - a. 如果传递的参数是单参数且参数类型是一个数组或者List的时候**collection**属性值分别是array和list(或collection)。
  - b. 如果传递的参数是多个的时候，就需要把它们封装成一个Map了当然单参数也可以封装成Map集合，这时**collection**属性值九尾Map的键。
  - c. 如果传递的参数是POJO包装类的时候，**collection**属性值就为该包装类中需要进行遍历的数组或者集合的属性名。

### 2.7.2 示例

需求：我们需要查询 user 表中 id 分别为1,2,3的用户

sql语句：select \* from user where id=1 or id=2 or id=3  
select \* from user where id in (1,2,3)

- 单参数List的类型

```

<select id="dynamicForeachTest" resultType="User">
    select * from user where id in
    <!--循环列表-->
    <foreach collection="list" index="index"
item="item" open="("
    separator="," close=")">
        #{item}
    </foreach>
</select>

```

```

public List<User> dynamicForeachTest(List<Integer> ids);

```

测试

```

@Test
public void testSelectUserByListId1(){
    List<Integer> ids = new ArrayList<Integer>();
    ids.add(1);
    ids.add(2);
    ids.add(3);
    List<User> listUser =
mapper.dynamicForeachTest(ids);
    for(User u : listUser){
        System.out.println(u);
    }
}

```

- 单参数array数组的类型

```

<select id="dynamicForeach2Test" resultType="User">
    select * from user where id in
    <!--循环数组-->
    <foreach collection="array" index="index"
item="item" open="(" separator="," close=")">
        #{item}
    </foreach>
</select>

```

```

public List<User> dynamicForeach2Test(Integer[] ids);

```

```
@Test
public void dynamicForeach2Test(){
    Integer[] ids={1,2,3};
    List<User> listUser =
mapper.dynamicForeach2Test(ids);
    for(User u : listUser){
        System.out.println(u);
    }
}
```

*IF + WHERE*

*SET*

*Foreach*