

05-MyBatis基于XML的详细使用——动态sql

05-MyBatis基于XML的详细使用——动态sql

1、动态sql

1、if

where

trim

3、foreach

3、choose、when、otherwise

4、set

1、动态sql

动态 SQL 是 MyBatis 的强大特性之一。如果你使用过 JDBC 或其它类似的框架，你应该能理解根据不同条件拼接 SQL 语句有多痛苦，例如拼接时要确保不能忘记添加必要的空格，还要注意去掉列表最后一个列名的逗号。利用动态 SQL，可以彻底摆脱这种痛苦。

使用动态 SQL 并非一件易事，但借助可用于任何 SQL 映射语句中的强大的动态 SQL 语言，MyBatis 显著地提升了这一特性的易用性。

如果你之前用过 JSTL 或任何基于类 XML 语言的文本处理器，你对动态 SQL 元素可能会感觉似曾相识。在 MyBatis 之前的版本中，需要花时间了解大量的元素。借助功能强大的基于 OGNL 的表达式，MyBatis 3 替换了之前的大部分元素，大大精简了元素种类，现在要学习的元素种类比原来的一半还要少。

- if
- choose (when, otherwise)
- trim (where, set)
- foreach
- bind
- sql片段

1、if

EmpDao.xml

```
1 <select id="getEmpByCondition" resultType="cn.tulingxueyuan.bean.Emp">
2     select * from emp where
```

```

3      <if test="empno!=null">
4          empno > #{empno} and
5      </if>
6      <if test="ename!=null">
7          ename like #{ename} and
8      </if>
9      <if test="sal!=null">
10         sal > #{sal}
11     </if>
12 </select>

```

EmpDao.java

```

1 public List<Emp> getEmpByCondition(Emp emp);

```

Test.java

@Test

```

1 public void test10() {
2
3     SqlSession sqlSession = sqlSessionFactory.openSession();
4     try {
5         EmpDao mapper = sqlSession.getMapper(EmpDao.class);
6         Emp emp = new Emp();
7         emp.setEmpno(6500);
8         emp.setEname("%E%");
9         emp.setSal(500.0);
10        List<Emp> empByCondition = mapper.getEmpByCondition(emp);
11        for (Emp emp1 : empByCondition) {
12            System.out.println(emp1);
13        }
14    } catch (Exception e) {
15        e.printStackTrace();
16    } finally {
17        sqlSession.close();
18    }
19 }

```

看起来测试是比较正常的，但是大家需要注意的是如果我们传入的参数值有缺失会怎么办呢？这个时候拼接的sql语句就会变得有问题，例如不传参数或者丢失最后一个参数，那么语句中就会多一个where

或者and的关键字，因此在mybatis中也给出了具体的解决方案：

where

where 元素只会在子元素返回任何内容的前提下才插入 “WHERE” 子句。而且，若子句的开头为 “AND” 或 “OR”，*where* 元素也会将它们去除。

```
1 <select id="getEmpByCondition" resultType="cn.tulingxueyuan.bean.Emp">
2     select * from emp
3     <where>
4         <if test="empno!=null">
5             empno > #{empno}
6         </if>
7         <if test="ename!=null">
8             and ename like #{ename}
9         </if>
10        <if test="sal!=null">
11            and sal > #{sal}
12        </if>
13    </where>
14 </select>
```

现在看起来没有什么问题了，但是我们的条件添加到了拼接sql语句的前后，那么我们该如何处理呢？

trim

```
1 <!--
2     trim截取字符串：
3     prefix: 前缀，为sql整体添加一个前缀
4     prefixOverrides: 去除整体字符串前面多余的字符
5     suffixOverrides: 去除后面多余的字符串
6     -->
7 <select id="getEmpByCondition" resultType="cn.tulingxueyuan.bean.Emp">
8     select * from emp
9
10    <trim prefix="where" prefixOverrides="and" suffixOverrides="and">
11        <if test="empno!=null">
12            empno > #{empno} and
13        </if>
14        <if test="ename!=null">
15            ename like #{ename} and
16        </if>
17        <if test="sal!=null">
```

```

18         sal > #{sal} and
19     </if>
20 </trim>
21 </select>

```

3、foreach

动态 SQL 的另一个常见使用场景是对集合进行遍历（尤其是在构建 IN 条件语句的时候）。

```

1  <!--foreach是对集合进行遍历
2      collection="deptnos" 指定要遍历的集合
3      close="" 表示以什么结束
4      index="" 给定一个索引值
5      item="" 遍历的每一个元素的值
6      open="" 表示以什么开始
7      separator="" 表示多个元素的分隔符
8  -->
9  <select id="getEmpByDeptnos" resultType="Emp">
10      select * from emp where deptno in
11          <foreach collection="deptnos" close=")" index="idx" item="deptno" open="(" separator=",">
12              #{deptno}
13          </foreach>
14  </select>

```

3、choose、when、otherwise

有时候，我们不想使用所有的条件，而只是想从多个条件中选择一个使用。针对这种情况，MyBatis 提供了 choose 元素，它有点像 Java 中的 switch 语句。

```

1  <select id="getEmpByConditionChoose" resultType="cn.tulingxueyuan.bean.Emp">
2      select * from emp
3      <where>
4          <choose>
5              <when test="empno!=null">
6                  empno > #{empno}
7              </when>
8              <when test="ename!=null">
9                  ename like #{ename}
10             </when>
11             <when test="sal!=null">
12                 sal > #{sal}

```

```

13             </when>
14             <otherwise>
15                 1=1
16             </otherwise>
17         </choose>
18     </where>
19 </select>

```

4、set

用于动态更新语句的类似解决方案叫做 *set*。*set* 元素可以用于动态包含需要更新的列，忽略其它不更新的列。

```

1 <update id="updateEmpByEmpno">
2     update emp
3     <set>
4         <if test="empno!=null">
5             empno=#{empno},
6         </if>
7         <if test="ename!=null">
8             ename = #{ename},
9         </if>
10        <if test="sal!=null">
11            sal = #{sal}
12        </if>
13    </set>
14    <where>
15        empno = #{empno}
16    </where>
17 </update>

```

- **bind**

bind 元素允许你在 OGNL 表达式以外创建一个变量，并将其绑定到当前的上下文。比如：

```

1
2 <select id="selectBlogsLike" resultType="Blog">
3     <bind name="pattern" value="'%' + _parameter.getTitle() + '%" />
4     SELECT * FROM BLOG
5     WHERE title LIKE #{pattern}
6 </select>

```

- **sql**

这个元素可以用来定义可重用的 SQL 代码片段，以便在其它语句中使用。参数可以静态地（在加载的时候）确定下来，并且可以在不同的 include 元素中定义不同的参数值。比如：

```
1 <sql id="userColumns"> ${alias}.id,${alias}.username,${alias}.password </sql>
```

这个 SQL 片段可以在其它语句中使用，例如：

```
1 <select id="selectUsers" resultType="map">
2   select
3     <include refid="userColumns"><property name="alias" value="t1"/></include>,
4     <include refid="userColumns"><property name="alias" value="t2"/></include>
5   from some_table t1
6   cross join some_table t2
7 </select>
```

MyBatis常用OGNL表达式

```
1 e1 or e2
2 e1 and e2
3 e1 == e2,e1 eq e2
4 e1 != e2,e1 neq e2
5 e1 lt e2: 小于
6 e1 lte e2: 小于等于, 其他gt (大于),gte (大于等于)
7 e1 in e2
8 e1 not in e2
9 e1 + e2,e1 * e2,e1/e2,e1 - e2,e1%e2
10 !e,not e: 非, 求反
11 e.method(args)调用对象方法
12 e.property对象属性值
13 e1[ e2 ]按索引取值, List,数组和Map
14 @class@method(args)调用类的静态方法
15 @class@field调用类的静态字段值
```