

03-MyBatis基于XML的详细使用-参数、返回结果处理

03-MyBatis基于XML的详细使用-参数、返回结果处理

1、参数的获取方式

2、参数的传递方式

3、处理集合返回结果

4、自定义结果集---resultMap

1、参数的取值方式

在xml文件中编写sql语句的时候有两种取值的方式，分别是#{ }和\${ }，下面来看一下他们之间的区别：

```
1  <!--获取参数的方式：
2      1.#{ } ==> jdbc String sql=" SELECT id,user_name FROM EMP WHERE id=?"
3          1.会经过JDBC当中PreparedStatement的预编译，会根据不同的数据类型来编译成对应数据库所对应的SQL。
4          2.能够有效的防止SQL注入。 推荐使用！！
5          特殊用法：
6          自带很多内置参数的属性：通常不会使用。了解
7          javaType、jdbcType、mode、numericScale、resultMap、typeHandler。
8          比如 需要改变默认的NULL==>OTHER:#{id,javaType=NULL}
9          想保留小数点后两位：#{id,numericScale=2}
10
11     2.${ } ==> jdbc String sql=" SELECT id,user_name FROM EMP WHERE id="+id
12         1.不会进行预编译，会直接将输入进来的数据拼接在SQL中。
13         2.存在SQL注入的风险。不推荐使用。
14         特殊用法：
15             1.调试情况下可以临时使用。
16             2.实现一些特殊功能：前提一定要保证数据的安全性。
17             比如：动态表、动态列。动态SQL。
18 -->
19 <select id="SelectEmp"  resultType="Emp"  resultMap="emp_map"  >
20     SELECT id,user_name,create_date FROM EMP where id=#{id}
21 </select>
```

2、select的参数传递

```
1
2  <!--
```

参数传递的处理:

1. 单个参数: `SelectEmp(Integer id);`

mybatis 不会做任何特殊要求

获取方式:

`#:{输入任何字符获取参数}`

2. 多个参数: `Emp SelectEmp(Integer id,String username);`

mybatis 会进行封装

会将传进来的参数封装成map:

1个值就会对应2个map项 : `id==> {key:arg0 ,value:id的值},{key:param1 ,value:id的值}`

`username==> {key:arg1 ,value:id的值},{key:param2 ,value:username的值}`

获取方式:

没使用了@Param:

`id====> #{arg0} 或者 #{param1}`

`username====> #{arg1} 或者 #{param2}`

除了使用这种方式还有别的方式,因为这种方式参数名没有意义:

设置参数的别名: `@Param(""): SelectEmp(@Param("id") Integer id,@Param("username") String username);`

当使用了@Param:

`id====> #{id} 或者 #{param1}`

`username====> #{username} 或者 #{param2}`

3. javaBean的参数:

单个参数: `Emp SelectEmp(Emp emp);`

获取方式: 可以直接使用属性名

`emp.id====>#{id}`

`emp.username====>#{username}`

多个参数: `Emp SelectEmp(Integer num,Emp emp);`

`num==> #{param1} 或者 @Param("num") Integer num`

`emp==> 必须加上对象别名: emp.id==> #{param2.id} 或者 @Param("emp")Emp emp`

`emp.username==> #{param2.username} 或者 @Param("emp")String username`

4. 集合或者数组参数:

`Emp SelectEmp(List<String> usernames);`

如果是list,MyBatis会自动封装为map:

`{key:"list":value:usernames}`

没用@Param("")要获得: `usernames.get(0) ==> #{list[0]}`

`:usernames.get(0) ==> #{arg0[0]}`

有@Param("usernames")要获得: `usernames.get(0) ==> #{usernames[0]}`

`:usernames.get(0) ==> #{param1[0]}`

如果是数组,MyBatis会自动封装为map:

`{key:"array":value:usernames}`

```

43         没用@param("")要获得: usernames.get(0)  =====>  #{array[0]}
44         : usernames.get(0)  =====>  #{arg0[0]}
45         有@param("usernames")要获得: usernames.get(0)  =====>  #{usernames[0]}
46         : usernames.get(0)  =====>  #{param1[0]}
47     5.map参数
48     和javaBean的参数传递是一样。
49     一般情况下:
50         请求进来的参数 和pojo对应, 就用pojo
51         请求进来的参数 没有和pojo对应, 就用map
52         请求进来的参数 没有和pojo对应上, 但是使用频率很高, 就用TO、DTO (就是单独为这些参数创
53
54     -->
55
56     <!--
57     接口: SelectEmp(String username,@Param("id") Integer id);
58         username=====>  #{arg0}  #{param1}
59         id=====>  #{id}  #{param2}
60     接口: SelectEmp(@Param("beginDate") String beginDate,
61         String endDate,
62         Emp emp);
63         beginDate=====>  #{beginDate}  #{param1}
64         endDate=====>  #{arg1}  #{param2}
65         emp.id=====>#{arg2.id}  #{param2.id}
66     接口: SelectEmp(List<Integer> ids,
67         String[] usernames,
68         @Param("beginDate") String beginDate,
69         String endDate,);
70         ids.get(0)=====>  #{list[0]}  #{param1[0]}
71         usernames[0]=====>  #{array[0]}  #{param2[0]}
72         beginDate=====>  #{beginDate}  #{param3}
73         end=====>  #{arg3}  #{param4}
74     -->
75

```

3、处理集合返回结果

EmpDao.xml

```

1  <!--当返回值的结果是集合的时候, 返回值的类型依然写的是集合中具体的类型-->
2      <select id="selectAllEmp" resultType="cn.tulingxueyuan.bean.Emp">

```

```

3      select  * from emp
4  </select>
5  <!--在查询的时候可以设置返回值的类型为map，当mybatis查询完成之后会把列的名称作为key
6      列的值作为value，转换到map中
7      -->
8      <select id="selectEmpByEmpReturnMap" resultType="map">
9          select * from emp where empno = #{empno}
10     </select>
11
12     <!--注意，当返回的结果是一个集合对象的是，返回值的类型一定要写集合具体value的类型
13     同时在dao的方法上要添加@MapKey的注解，来设置key是什么结果
14     @MapKey("empno")
15     Map<Integer,Emp> getAllEmpReturnMap();-->
16     <select id="getAllEmpReturnMap" resultType="cn.tulingxueyuan.bean.Emp">
17         select * from emp
18     </select>

```

UserDao.java

```

1  package cn.tulingxueyuan.dao;
2
3  import cn.tulingxueyuan.bean.Emp;
4  import org.apache.ibatis.annotations.MapKey;
5  import org.apache.ibatis.annotations.Param;
6
7  import java.util.List;
8  import java.util.Map;
9
10 public interface EmpDao {
11
12     public Emp findEmpByEmpno(Integer empno);
13
14     public int updateEmp(Emp emp);
15
16     public int deleteEmp(Integer empno);
17
18     public int insertEmp(Emp emp);
19
20     Emp selectEmpByNoAndName(@Param("empno") Integer empno, @Param("ename") String ename);
21     Emp selectEmpByNoAndName2(Map<String,Object> map);

```

```

22
23     List<Emp> selectAllEmp();
24
25     Map<String,Object> selectEmpByEmpReturnMap(Integer empno);
26
27     @MapKey("empno")
28     Map<Integer,Emp> getAllEmpReturnMap();
29 }

```

4、自定义结果集---resultMap

EmpMapper.xml

```

1  <!--1.声明resultMap自定义结果集    resultMap 和 resultType 只能使用一个。
2      id 唯一标识， 需要和<select 上的resultMap 进行对应
3      type 需要映射的pojo对象， 可以设置别名
4      autoMapping 自动映射，（默认=true） 只要字段名和属性名遵循映射规则就可以自动映射，但是不建议
5      extends 如果多个resultMap有重复映射，可以声明父resultMap,将公共的映射提取出来， 可以减少
6  -->
7  <resultMap id="emp_map" type="emp" autoMapping="false" extends="common_map">
8      <result column="create_date" property="cjsj"></result>
9  </resultMap>
10
11  <resultMap id="common_map" type="emp" autoMapping="false" >
12      <!-- <id> 主键必须使用 对底层存储有性能作用
13              column 需要映射的数据库字段名
14              property 需要映射的pojo属性名
15      -->
16      <id column="id" property="id"></id>
17      <result column="user_name" property="username"></result>
18  </resultMap>
19
20  <!--2.使用resultMap 关联 自定义结果集的id-->
21  <select id="SelectEmp" resultType="Emp" resultMap="emp_map" >
22      SELECT id,user_name,create_date FROM EMP where id=#{id}
23  </select>

```