03-MyBatis基于XML的详细使用-参数、返回结果处理

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：

```
<!--获取参数的方式：
1.#{} ==> jdbc String sql=" SELECT id,user_name FROM EMP WHERE id=?"
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数据。
2.能够有效的防止SQL注入。 推荐使用！！
特殊用法：
自带很多内置参数的属性：通常不会使用。了解
javaType、jdbcType、mode、numericScale、resultMap、typeHandler.
比如 需要改变默认的NULL===>OTHER:#{id,javaType=NULL}
想保留小数点后两位：#{id,numericScale=2}

 2.${} ==> jdbc String sql=" SELECT id,user_name FROM EMP WHERE id="+id
 1.不会进行预编译，会直接将输入进来的数据拼接在SQL中。
 2.存在SQL注入的风险。不推荐使用。
 特殊用法：
 1.调试情况下可以临时使用。
 2.实现一些特殊功能:前提一定要保证数据的安全性。
 比如：动态表、动态列. 动态SQL.
-->
<select id="SelectEmp" resultType="Emp" resultMap="emp_map" >
 SELECT id,user_name,create_date FROM EMP where id=#{id}
</select>
```

2、select的参数传递

```
1
2 <!--
3  参数传递的处理：
4  1.单个参数:SelectEmp(Integer id);
5  mybatis 不会做任何特殊要求
6  获取方式：
7  #:{输入任何字符获取参数}
8
9  2.多个参数:Emp SelectEmp(Integer id,String username);
10  mybatis 会进行封装
11  会将传进来的参数封装成map:
12  1个值就会对应2个map项 ： id===> {key:arg0 ,value:id的值},{key:param1 ,value:id的值}
13  username===> {key:arg1 ,value:id的值},{key:param2 ,value:id的值}
14  获取方式：
15  没使用了@Param：
16  id=====> #{arg0} 或者 #{param1}
17  username=====> #{arg1} 或者 #{param2}
18  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:
19  设置参数的别名：@Param("")：SelectEmp(@Param("id") Integer id,@Param("username") String username);
20  当使用了@Param:
21  id=====> #{id} 或者 #{param1}
22  username=====> #{username} 或者 #{param2}
23
24  3. javaBean的参数:
25  单个参数：Emp SelectEmp(Emp emp);
26  获取方式：可以直接使用属性名
27  emp.id=====>#{id}
28  emp.username=====>#{username}
29  多个参数：Emp SelectEmp(Integer num,Emp emp);
30  num===> #{param1} 或者 @Param
31  emp===> 必须加上对象别名： emp.id===> #{param2.id} 或者 @Param("emp")Emp emp ====>#{emp.id}
32  emp.username===> #{param2.username} 或者 @Param("emp")Emp emp ====>#{emp.username}
33  4.集合或者数组参数：
34  Emp SelectEmp(List<String> usernames);
35  如果是list,MyBatis会自动封装为map:
36  {key:"list":value:usernames}
```

```
没用@Param("")要获得:usernames.get(0) =====> #{list[0]}
:usernames.get(0) =====> #{agr0[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}
没用@Param("")要获得:usernames.get(0) =====> #{array[0]}
:usernames.get(0) =====> #{agr0[0]}
有@Param("usernames")要获得:usernames.get(0) =====> #{usernames[0]}
:usernames.get(0) =====> #{param1[0]}
5.map参数
和javaBean的参数传递是一样。
一般情况下:
请求进来的参数 和pojo对应，就用pojo
请求进来的参数 没有和pojo对应，就用map
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建一个对应的javaBean出来,使参数传递更规范、更重用）

-->

<!--
接口：SelectEmp(String username,@Param("id") Integer id);
username====> #{arg0} #{param1}
id====> #{id} #{param2}
接口：SelectEmp(@Param("beginDate") String beginDate,
String endDate,
Emp emp);
beginDate====> #{beginDate} #{param1}
endDate====> #{arg1} #{param2}
emp.id====>#{arg2.id} #{param2.id}
接口：SelectEmp(List<Integer> ids,
String[] usernames,
@Param("beginDate") String beginDate,
String endDate,);
ids.get(0)=====> #{list[0]} #{param1[0]}
usernames[0]=====> #{array[0]} #{param2[0]}
beginDate====> #{beginDate} #{param3}
end====> #{arg3} #{param4}
-->

```

3、处理集合返回结果

EmpDao.xml

```
<!--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-->
 <select id="selectAllEmp" resultType="cn.tulingxueyuan.bean.Emp">
 select * from emp
 </select>
<!--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
 列的值作为value，转换到map中
 -->
 <select id="selectEmpByEmpReturnMap" resultType="map">
 select * from emp where empno = #{empno}
  </select>

 <!--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
 同时在dao的方法上要添加@MapKey的注解，来设置key是什么结果
 @MapKey("empno")
 Map<Integer,Emp> getAllEmpReturnMap();-->
 <select id="getAllEmpReturnMap" resultType="cn.tulingxueyuan.bean.Emp">
 select * from emp
 </select>
```

UserDao.java

```
package cn.tulingxueyuan.dao;

import cn.tulingxueyuan.bean.Emp;
import org.apache.ibatis.annotations.MapKey;
import org.apache.ibatis.annotations.Param;

import java.util.List;
import java.util.Map;

public interface EmpDao {

 public Emp findEmpByEmpno(Integer empno);

 public int updateEmp(Emp emp);

 public int deleteEmp(Integer empno);

```

```
18   public int insertEmp(Emp emp);
19
20   Emp selectEmpByNoAndName(@Param("empno") Integer empno, @Param("ename")
String ename,@Param("t") String tablename);
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 resultType 和 resultMap 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，哪怕属性名和字段名一一对应上了也要显示的配置映射
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， 可以减少子resultMap的映射冗余
6  -->
7  <resultMap id="emp_map" type="emp" autoMapping="false" extends="common_ma
p">
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>
```