映射文件

# 输入映射、输出映射POJO类时：getter与setter方法

传入POJO的属性时，默认调用**getXxx**方法获取属性值；

获取**POJO映射**时，首先调用**无参的构造方法**，然后通过**setXxx**设置属性值；如果没有无参构造方法，则会根据读取的值，调用相应的构造方法，如果恰好没有此构造方法，就会出错。但是如果具有无参构造方法，就不会报错。**所以，建议在创建POJO时，一定要创建无参的构造方法**，即使是私有的，也可以调用。

# 输入映射：parameterType

（parameterMap已经废弃）。

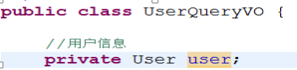
## 简单类型：8中基本数据类型包装类、String等。

**利用parameterType引入。**

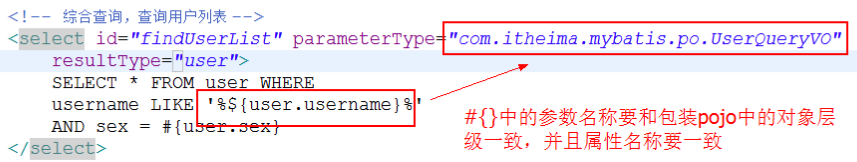
## POJO类型：属性为简单类型。

## 包装POJO类：属性有POJO类。

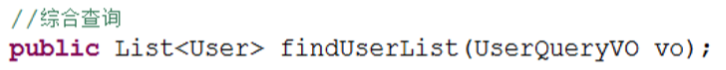
1. **包装POJO类**：



1. **包装POJO类的映射文件**：



1. **mapper接口中**：



## Map：同传递POJO对象一样，map的key相当于pojo的属性。

### 映射文件

<!-- 传递hashmap综合查询用户信息 -->

<select id=*"findUserByHashmap"* parameterType=*"hashmap"* resultType=*"user"*>

select \* from user where id=#{id} and username like '%${username}%'

</select>

上边红色标注的是**hashmap**的key。

### 测试代码

**Public void** testFindUserByHashmap()**throws** Exception{

//获取session

SqlSession session = sqlSessionFactory.openSession();

//获限mapper接口实例

UserMapper userMapper = session.getMapper(UserMapper.**class**);

//构造查询条件Hashmap对象

HashMap<String, Object> map = **new** HashMap<String, Object>();

map.put("id", 1);

map.put("username", "管理员");

//传递Hashmap对象查询用户列表

List<User>list = userMapper.findUserByHashmap(map);

//关闭session

session.close();

}

异常测试：

**传递的map中的key和sql中解析的key不一致。**

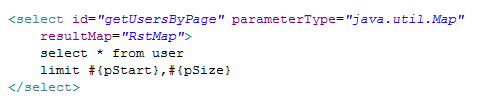
**测试结果没有报错，只是通过key获取值为空。**

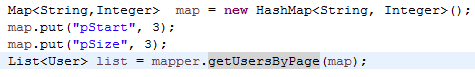
## 输入多个参数的时候可以采用Map参数进行包装

根据**#{key}**即可获取其对应的值。

例如分页查询：





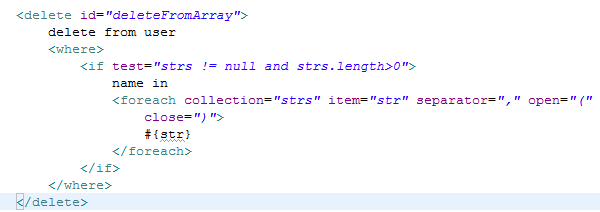


注意：Map作为入参时，请不要在接口方法中使用注解起别名。在#{key}直接利用键名就可以获取对应的值。

## forecah遍历数组

**parameterType**不需要指明。





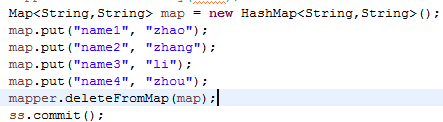


## foreach 遍历Map集合

**在foreach遍历Map集合**时，不需要利用**键key**。但是在简单传入参数时，是通过#{key}获取其对应的值的。







## foreach功能总结：

**foreach**的主要用在**构建in条件**中，它可以在SQL语句中进行迭代一个集合。**foreach元素**的属性主要有**item，index，collection，open，separator，close**。

**item**表示集合中每一个元素进行迭代时的别名；

**index指定一个名字，用于表示在迭代过程中，每次迭代到的位置**；

open表示该语句以什么开始；

separator表示在每次进行迭代之间以什么符号作为分隔符；

close表示以什么结束。

在使用foreach的时候最关键的也是最容易出错的就是**collection属性**，该属性是必须指定的，但是在不同情况下，该属性的值是不一样的，主要有以下3种情况： （collection属性名可以设置为接口方法指定的**@Param(“XXX”)**中的XXX。）

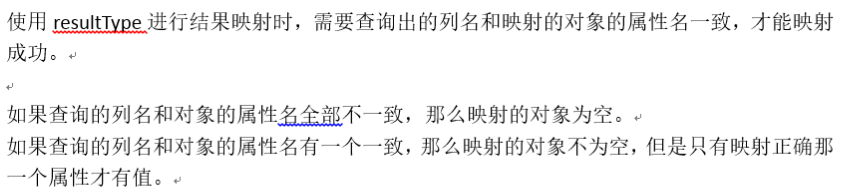
1. 如果传入的是单参数且参数类型是一个List的时候，collection属性值为list ；
2. 如果传入的是单参数且参数类型是一个array数组的时候，collection的属性值为array ；
3. **如果传入的参数是多个的时候，我们就需要把它们封装成一个Map了，当然单参数也可以封装成map，实际上如果你在传入参数的时候，在MyBatis里面也是会把它封装成一个Map的，map的key就是参数名，所以这个时候collection属性值就是传入的List或array对象在自己封装的map里面的key。**

# 输出映射：resultType和resultMap

## resultType介绍：

使用**resultType**必须满足：

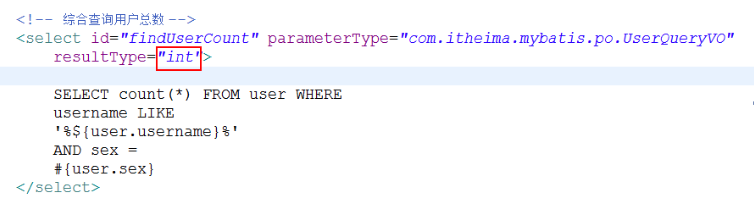
查询的**数据库中列名与POJO的属性名**必须一样，才可以成功，否则不能正常获取值，但不会报错。





### 简单类型：



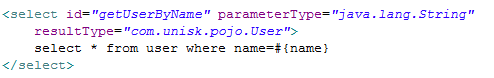


**mapper接口中的方法**：



### POJO类及List集合

要求数据库中**列名与POJO中的属性名及数据类型**必须一致。、





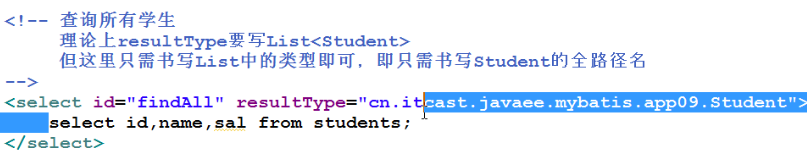
**重要:**

如果select查询的是一条记录可以直接利用POJO接收，如果查询的是多条记录，则返回的是一个List集合。但是resultType仍然是对应的List集合元素的数据类型。只需要在定义接口中的方法的时候，返回值类型设置为List<User>即可。





**返回集合的时候，返回类型只需要用集合元素的类型声明即可。**



## resultMap：

**子标签**可以见**ResultMap**内部标签。

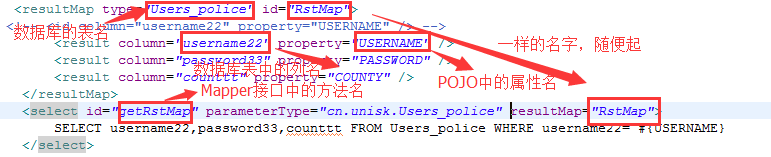
使用场景：**把数据库的列名与POJO类属性名建立映射关系。**



**经试验证明了：POJO类的属性名与数据库表中的列名不一样时，利用ResultMap进行一下映射，把数据库的列名与POJO类中的属性名进行关联**。

**ResultMap的内部也是通过ResultType进行实现的**。





# 多参数传入问题

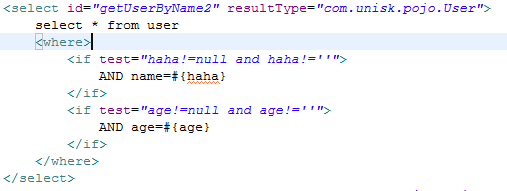
MyBatis可以实现**多参数传入**的问题。

很多博客都对此行了详细的总结。

在**Mapper接口方法**中，对参数利用**@Param(“xxx”)**进行注解注释，以便在**映射文件**中对应起来。如果不起名字的话，也可以利用#{1}、#{2}等进行传入参数。

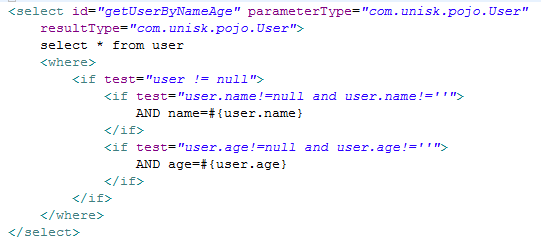
但是，一般不建议多参数传入的问题。如果需要传入多个参数，一般是把参数集中到**一个对象或一个Map集合**中，从而只需要**传入一个对象**即可。





建议封装多个参数：





# jdbcType与javaType的对应关系

<http://blog.csdn.net/haofeng82/article/details/34857991>

