## 参数个数不同

### 单个参数

mybatis不会做特殊处理，

**#{参数名/任意名}：取出参数值。**

### 多个参数

mybatis会做特殊处理。

多个参数会被封装成 一个map，

**key：param1...paramN,或者参数的索引也可以**

**value：传入的参数值**

**#{}就是从map中获取指定的key的值；**

异常：

org.apache.ibatis.binding.BindingException:

Parameter 'id' not found.

Available parameters are [1, 0, param1, param2]

操作：

方法：public Employee getEmpByIdAndLastName(Integer id,String lastName);

取值：#{id},#{lastName}

#### 命名参数（注解）

明确指定封装参数时map的key；**@Param("id")**

多个参数会被封装成 一个map，

**key：使用@Param注解指定的值**

**value：参数值**

**public Employee getEmp(@Param("id")Integer id,String lastName);**

**#{指定的key}取出对应的参数值**

#### POJO（数据模型）

如果多个参数正好是我们业务逻辑的数据模型，我们就可以直接传入pojo；

**#{属性名}：取出传入的pojo的属性值**

#### Map（键值对）

如果多个参数不是业务模型中的数据，没有对应的pojo，不经常使用，为了方便，我们也可以传入map

**#{key}：取出map中对应的值**

#### TO（数据封装）

如果多个参数不是业务模型中的数据，但是经常要使用，推荐来编写一个TO（Transfer Object）数据传输对象（**把这组数据封装成类**，按pojo的方式获取）

Page{

int index;

int size;

}

## 思考案例

**public Employee getEmp(@Param("id")Integer id,String lastName);**

取值：id==>#{id/param1} lastName==>#{param2}

**public Employee getEmp(Integer id,@Param("e")Employee emp);**

取值：id==>#{param1} lastName===>#{param2.lastName/e.lastName}

## **集合或数组为参数**

如果是Collection（List、Set）类型或者是数组，也会特殊处理。也是把传入的list或者数组封装在map中。

**key**：**Collection**（**collection**）,**List** (**list**），**数组**(**array**)

**public Employee getEmpById(List<Integer> ids);**

取值：取出第一个id的值：  **#{list[0]}**

## 结合源码，mybatis怎么处理参数

**总结：**

参数多时会封装map，为了不混乱，我们可以使用@Param来指定封装时使用的key；

#{key}就可以取出map中的值；

**(@Param("id")Integer id,@Param("lastName")String lastName);**

ParamNameResolver解析参数封装map的；

//1、names：{0=id, 1=lastName}；构造器的时候就确定好了

确定流程：

**1.获取每个标了param注解的参数的@Param的值：id，lastName； 赋值给name;**

**2.每次解析一个参数给map中保存信息：（key：参数索引，value：name的值）**

**name的值：**

标注了param注解：注解的值

没有标注：

1.**全局配置：useActualParamName（jdk1.8）：name=参数名**

2.**name=map.size()；相当于当前元素的索引**

{0=id, 1=lastName,2=2}

args【1，"Tom",'hello'】:

**public** Object getNamedParams(Object[] args) {

**final** **int** paramCount = names.size();

//1、参数为null直接返回

**if** (args == **null** || paramCount == 0) {

**return** **null**;

//2、如果只有一个元素，并且没有Param注解；args[0]：单个参数直接返回

} **else** **if** (!hasParamAnnotation && paramCount == 1) {

**return** args[names.firstKey()];

//3、多个元素或者有Param标注

} **else** {

**final** Map<String, Object> param = **new** ParamMap<Object>();

**int** i = 0;

//4、遍历names集合；{0=id, 1=lastName,2=2}

**for** (Map.Entry<Integer, String> entry : names.entrySet()) {

//names集合的value作为key; names集合的key又作为取值的参考args[0]:args【1，"Tom"】:

//eg:{id=args[0]:1,lastName=args[1]:Tom,2=args[2]}

param.put(entry.getValue(), args[entry.getKey()]);

// add generic param names (param1, param2, ...)param

//额外的将每一个参数也保存到map中，使用新的key：param1...paramN

//效果：有Param注解可以#{指定的key}，或者#{param1}

**final** String genericParamName = GENERIC\_NAME\_PREFIX + String.valueOf(i + 1);

// ensure not to overwrite parameter named with @Param

**if** (!names.containsValue(genericParamName)) {

param.put(genericParamName, args[entry.getKey()]);

}

i++;

}

**return** param;

}

}

}

## 参数值的获取#{ }或${ }

#{}：可以获取map中的值或者pojo对象属性的值；

${}：可以获取map中的值或者pojo对象属性的值；

**select \* from tbl\_employee where id=${id} and last\_name=#{lastName}**

**Preparing: select \* from tbl\_employee where id=2 and last\_name=?**

**区别：**

**#{ }:是以预编译的形式，将参数设置到sql语句中；PreparedStatement；防止sql注入**

**${ }:取出的值直接拼装在sql语句中；会有安全问题；**

大多情况下，我们去参数的值都应该去使用#{ }，而${ }有特定的使用场景

**${ }：** 原生jdbc不支持占位符的地方我们就可以使用${ }进行取值

**比如**分表、排序。。。；按照年份分表拆分

**select \* from ${year}\_salary where xxx;**

**select \* from tbl\_employee order by ${f\_name} ${order}**

**#{ }:** 更丰富的用法：

规定参数的一些规则：

**javaType、 jdbcType、 mode（存储过程）、 numericScale、**

**resultMap、 typeHandler、 jdbcTypeName、 expression（未来准备支持的功能）；**

jdbcType通常需要在某种特定的条件下被设置：

在我们数据为null的时候，有些数据库可能不能识别mybatis对null的默认处理。比如Oracle（报错）；

JdbcType OTHER：无效的类型；因为mybatis对所有的null都映射的是原生Jdbc的OTHER类型，oracle不能正确处理;

由于全局配置中：jdbcTypeForNull=OTHER；oracle不支持；两种办法

1、**#{email,jdbcType=NULL};**

2、**jdbcTypeForNull=NULL**

**<setting name="jdbcTypeForNull" value="NULL"/>**