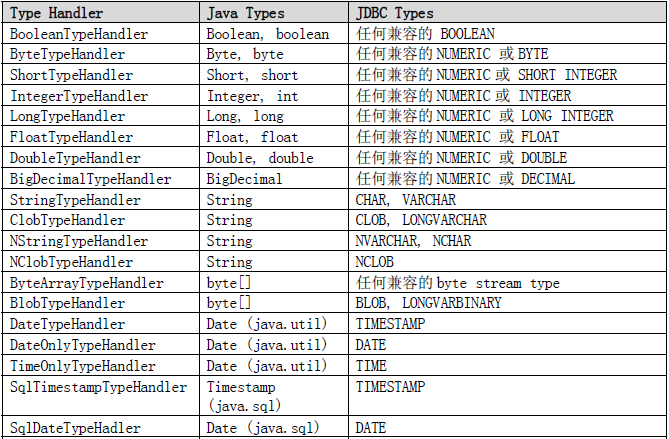
# MyBatis框架配置文件详解

## typeHandlers 类型转换器

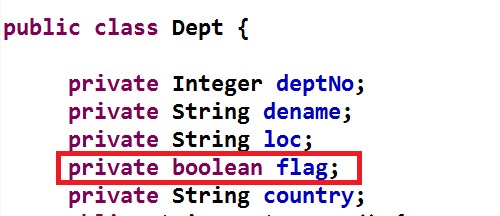
每当MyBatis 设置参数到PreparedStatement 或者从ResultSet 结果集中取得值时，就会使用TypeHandler 来处理数据库类型与java 类型之间转换。下表描述了默认TypeHandlers



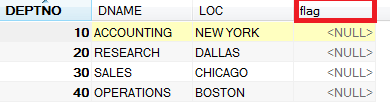
### 自定义类型转换器

假设表中字段是int类型,而实体类与之对应的属性是boolean类型,此时可以采用自定义类型转换器进行对应

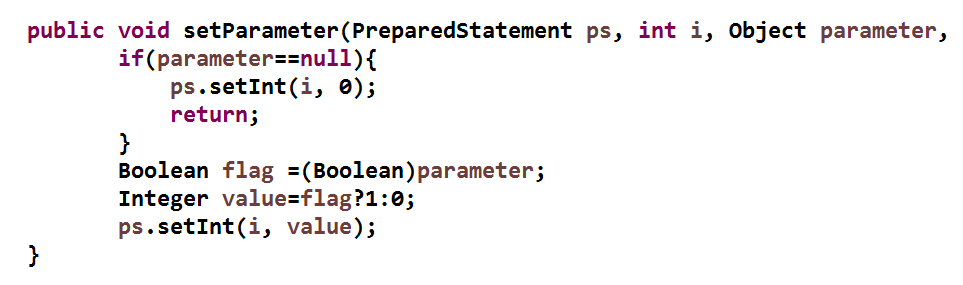
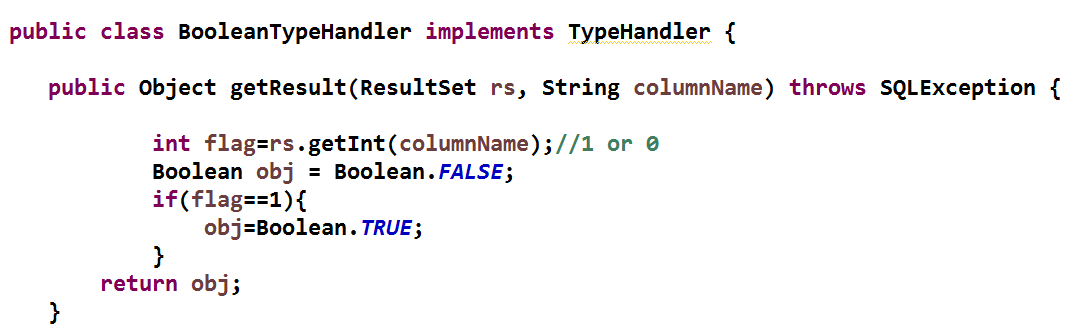
#### 实体类



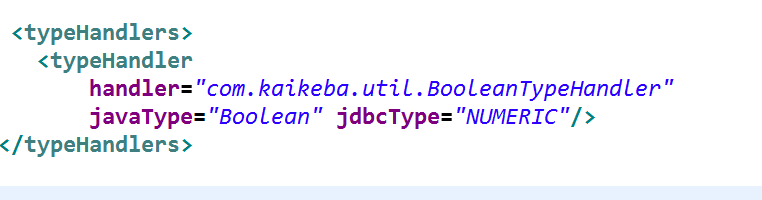
#### 表中字段



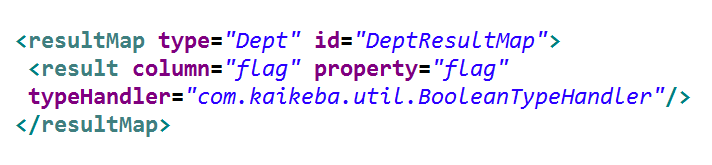
#### 开发自定义类型转换器



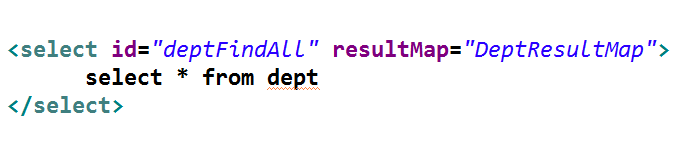
#### 在MyBatis核心配置文件注册自定义类型转换器



#### 在Mapper.xml文件中指定使用自定义类型转换器场合



#### 在查询Statement中指定对应的ResultMap

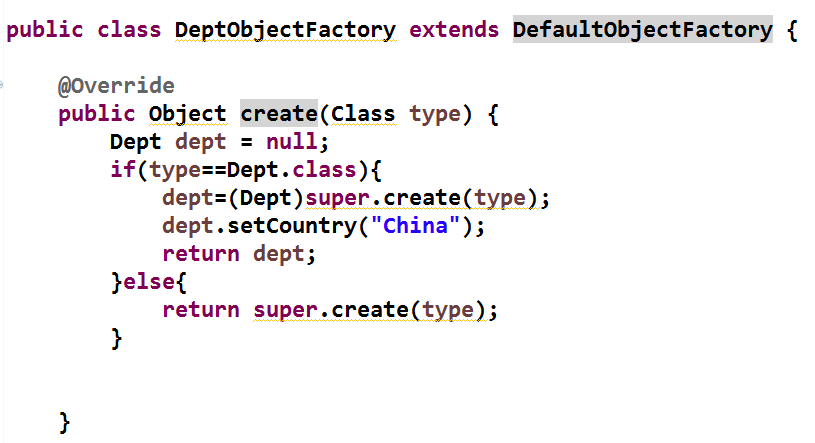


## objectFactory对象工厂

MyBatis 每次创建结果对象的新实例时，它都会使用一个对象工厂（ObjectFactory）实例来完成。 默认的对象工厂需要做的仅仅是实例化目标类，要么通过默认构造方法，要么在参数映射存在的时候通过参数构造方法来实例化。 如果想覆盖对象工厂的默认行为，则可以通过创建自己的对象工厂来实现。

### 自定义对象工厂

#### 继承与DefaultObjectFactory



继承DefaultObjectFactory的原因,可以利用父类中super.create方法帮助我们创建一个实例对象

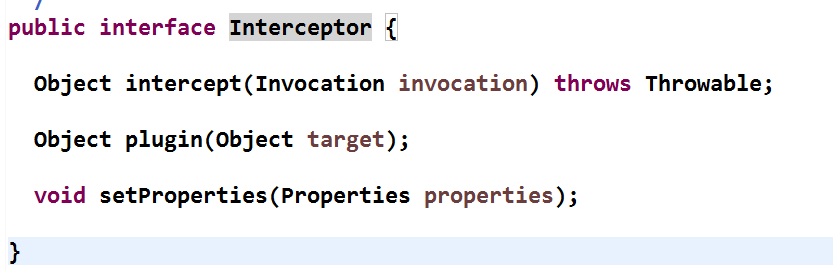
#### 在MyBatis核心文件中注册自定义工厂



## plugins拦截器

拦截器的一个作用就是我们可以拦截某些方法的调用，我们可以选择在这些被拦截的方法执行前后加上某些逻辑，也可以在执行这些被拦截的方法时执行自己的逻辑而不再执行被拦截的方法。Mybatis拦截器设计的一个初衷就是为了供用户在某些时候可以实现自己的逻辑而不必去动Mybatis固有的逻辑。打个比方，对于Executor，Mybatis中有几种实现：BatchExecutor、ReuseExecutor、SimpleExecutor和CachingExecutor。这个时候如果你觉得这几种实现对于Executor接口的query方法都不能满足你的要求，那怎么办呢？是要去改源码吗？当然不。我们可以建立一个Mybatis拦截器用于拦截Executor接口的query方法，在拦截之后实现自己的query方法逻辑，之后可以选择是否继续执行原来的query方法。

对于拦截器Mybatis为我们提供了一个Interceptor接口，通过实现该接口就可以定义我们自己的拦截器。我们先来看一下这个接口的定义：

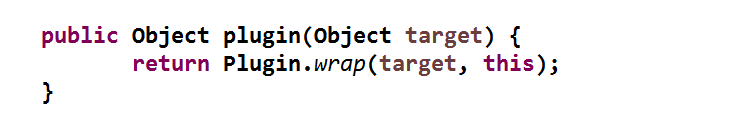
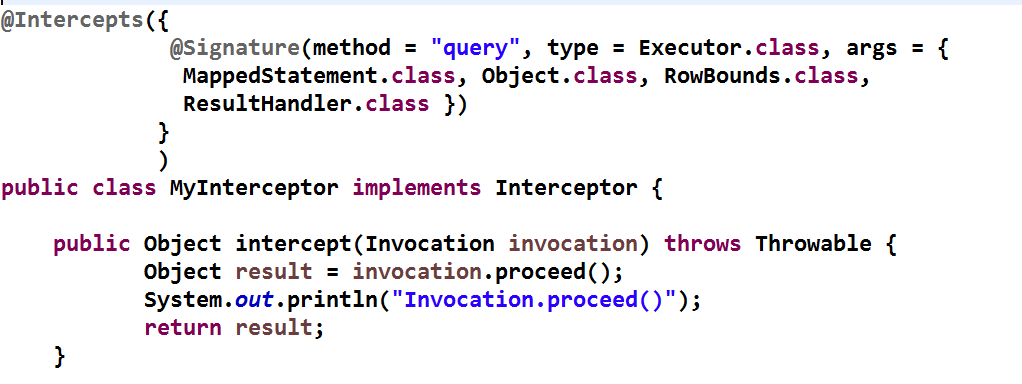


我们可以看到在该接口中一共定义有三个方法，intercept、plugin和setProperties。plugin方法是拦截器用于封装目标对象的，通过该方法我们可以返回目标对象本身，也可以返回一个它的代理。当返回的是代理的时候我们可以对其中的方法进行拦截来调用intercept方法，当然也可以调用其他方法，这点将在后文讲解。setProperties方法是用于在Mybatis配置文件中指定一些属性的。

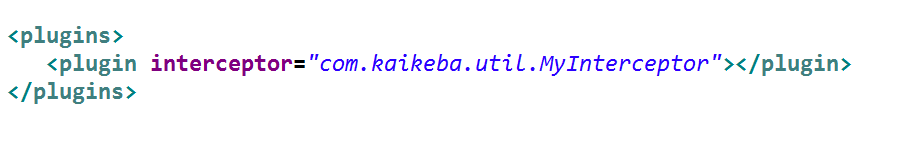
       定义自己的Interceptor最重要的是要实现plugin方法和intercept方法，在plugin方法中我们可以决定是否要进行拦截进而决定要返回一个什么样的目标对象。而intercept方法就是要进行拦截的时候要执行的方法。

       对于plugin方法而言，其实Mybatis已经为我们提供了一个实现。Mybatis中有一个叫做Plugin的类，里面有一个静态方法wrap(Object target,Interceptor interceptor)，通过该方法可以决定要返回的对象是目标对象还是对应的代理。

对于实现自己的Interceptor而言有两个很重要的注解，一个是@Intercepts，其值是一个@Signature数组。@Intercepts用于表明当前的对象是一个Interceptor，而@Signature则表明要拦截的接口、方法以及对应的参数类型。来看一个自定义的简单Interceptor：



然后在MyBatis核心配置文件注册自定义拦截器



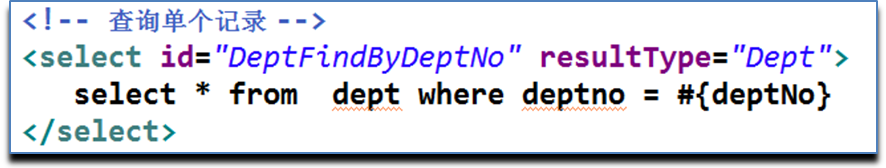
MyBatis自定义拦截器，可以拦截接口只有四种.Executor.clas，StatementHandler.class,

ParameterHandler.class,ResultSetHandler.class

# MyBatis框架Mapper配置文件详解

## 参数(#{参数名})

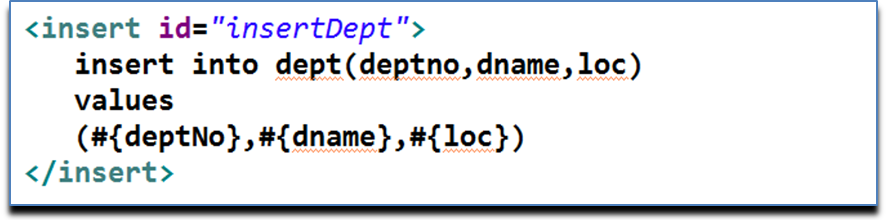
#{}实现的是向prepareStatement中的预处理语句中设置参数值，sql语句中#{}表示一个占位符即?



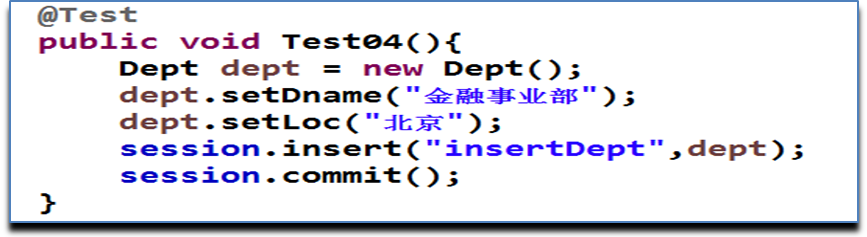
使用#{参数名},将参数的内容添加到sql语句中指定位置.

如果当前sql语句中只有一个参数,此时参数名称可以随意定义

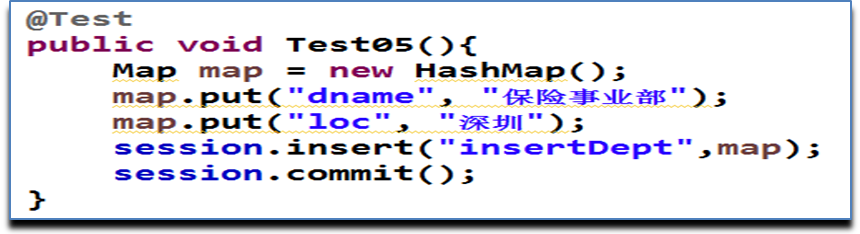
但是,如果当前sql语句有多个参数,此时参数名称应该是与当前表关联[实体类的属性名]或则[Map集合关键字]



上述SQL语句在调用时,我们可以分别采用如下两种方式输入参数



使用#{}读取实体类对象属性内容



使用#{}读取map集合中关键字的值

## #{}和${}区别

在MyBatis中提供了两种方式读取参数的内容到SQL语句中,分别是

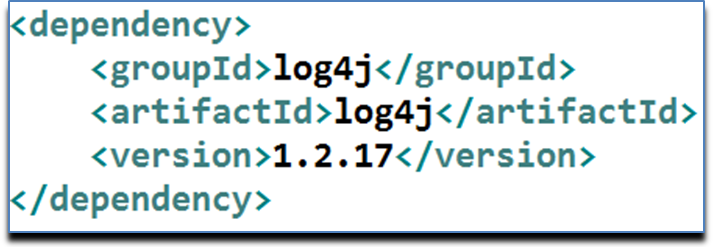
#{参数名} :实体类对象或则Map集合读取内容

${参数名} :实体类对象或则Map集合读取内容

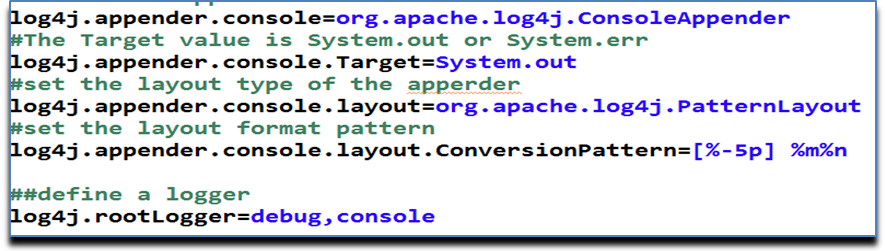
为了能够看到两种方式的区别,需要看到MyBatis执行时输送的SQL情况.因此

需要借助Log4J日志进行观察

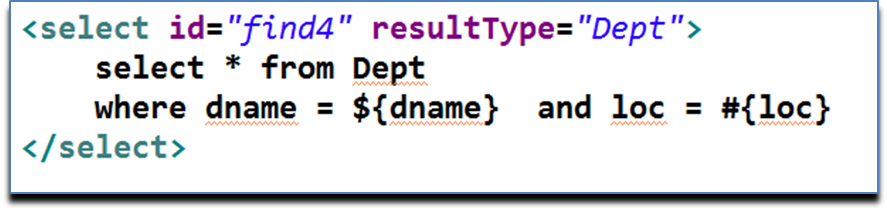
第一步: 加载Log4j日志工具包到项目



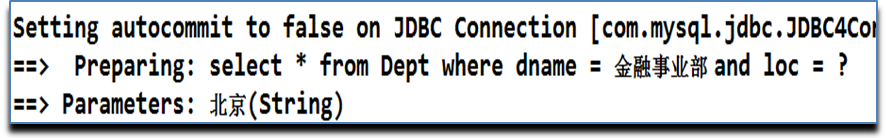
第二步:将Log4j配置文件添加到src/main/resources下



接下来,我们可以查看



输出结果



从这里我们可以看出两者的区别:

#{} : 采用预编译方式,可以防止SQL注入

${}: 采用直接赋值方式,无法阻止SQL注入攻击

在大多数情况下,我们都是采用#{}读取参数内容.但是在一些特殊的情况下,我们还是需要使用${}读取参数的.

比如 有两张表,分别是emp\_2017 和 emp\_2018 .如果需要在查询语句中动态指定表名,就只能使用${}

<select>

select \* from emp\_${year}

<select>

再比如.需要动态的指定查询中的排序字段,此时也只能使用${}

<select>

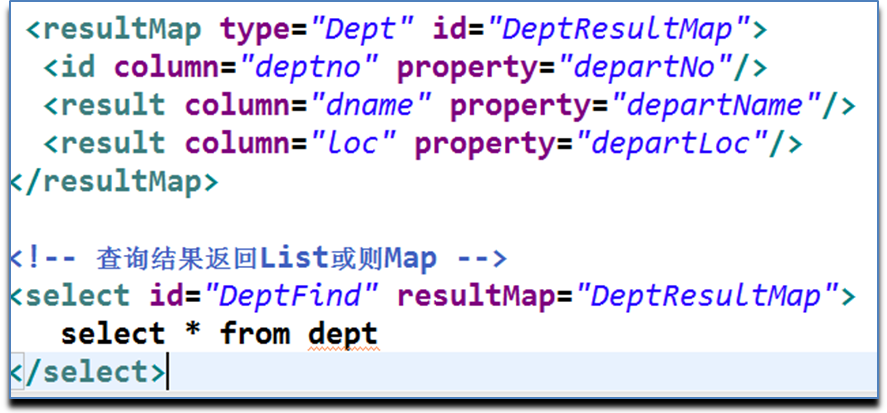
select \* from dept order by ${name}

</select>

简单来说,在JDBC不支持使用占位符的地方,都可以使用${}

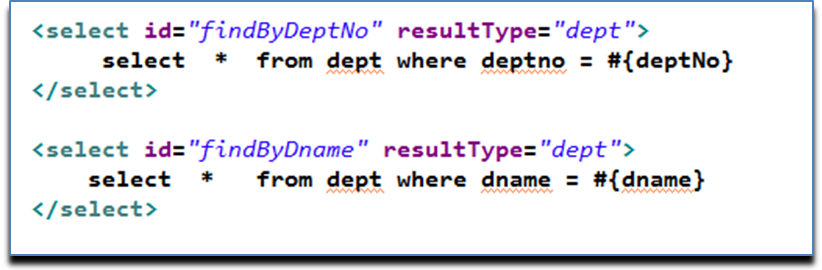
## resultMap

MyBatis框架中是根据表中字段名到实体类定位同名属性的.如果出现了实体类属性名与表中字段名不一致的情况,则无法自动进行对应.此时可以使用resultMap来重新建立实体类与字段名之间对应关系.



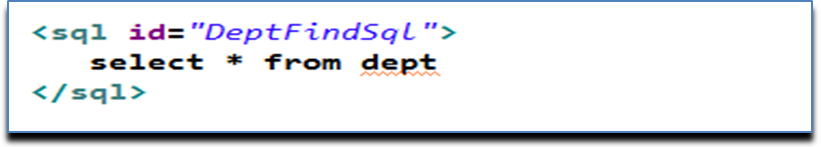
## sql标签

首先,我们如下两条SQL映射

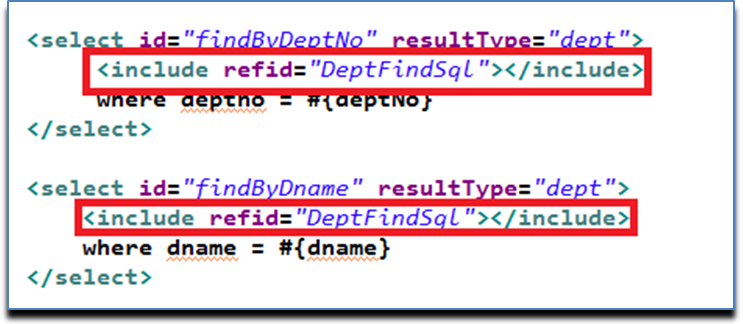


这两条查询映射中要查询的表以及查询的字段是完全一致的.因此可以<sql>标签

将[select \* from dept]进行抽取.



在需要使用到这个查询的地方,通过<include>标签进行引用



# MyBatis动态SQL

## 什么是MyBatis动态SQL

根据用户提供的参数,动态决定查询语句依赖的查询条件或则SQL语句的内容

## 动态SQL依赖标签

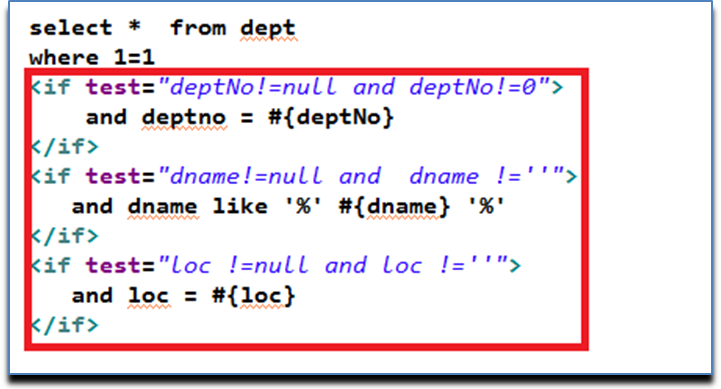
### if

### choose、when、otherwise

### trim、where、set

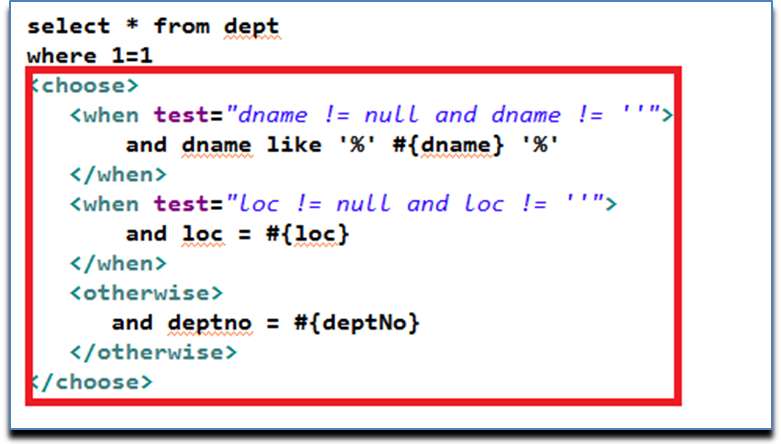
### foreach

## if使用



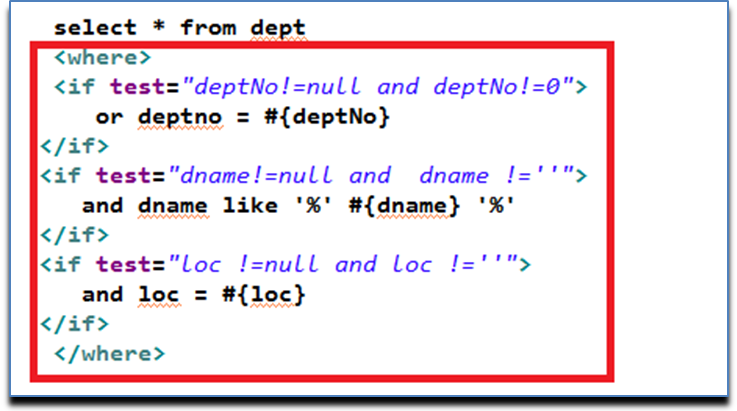
## ****choose、when、otherwise****

类似于Java中的switch case default. 只有一个条件生效,也就是只执行满足的条件when,没有满足的条件就执行otherwise,表示默认条件



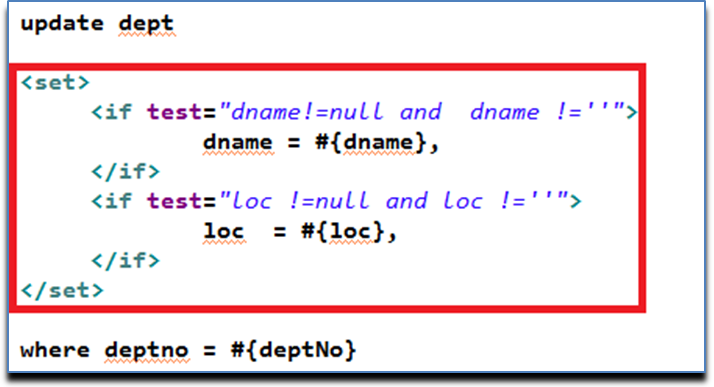
## where

<where>可以自动的将第一个条件前面的逻辑运算符(or ,and)去掉

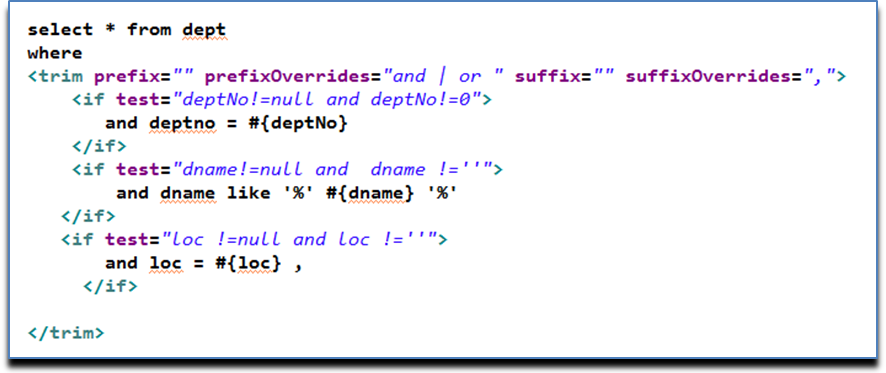


## set

会在成功拼接的条件前加上SET单词且最后一个”,”号会被无视掉



## trim

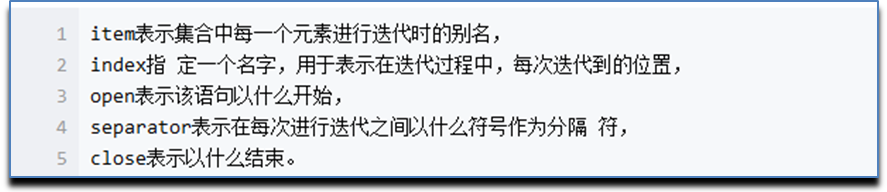


## foreach

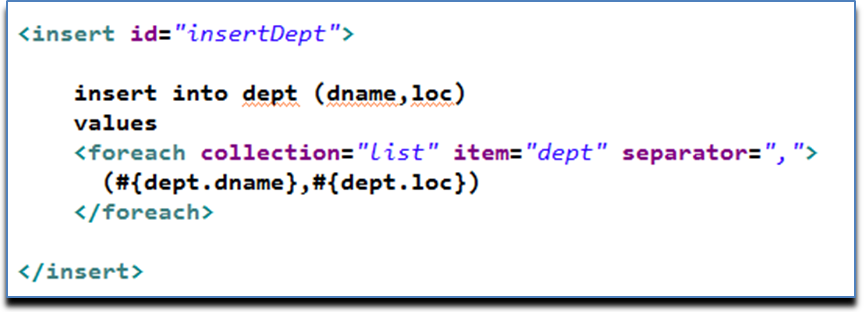
foreach标签用于对集合内容进行遍历,将得到内容作为SQL语句的一部分.

在实际开发过程中主要用于in语句的构建和批量添加操作

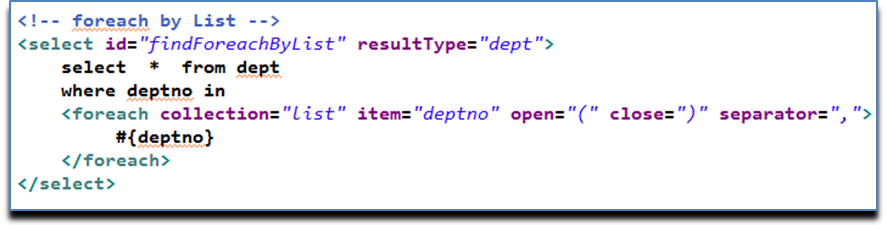
foreach元素的属性主要有 item，index，collection，open，separator，close。



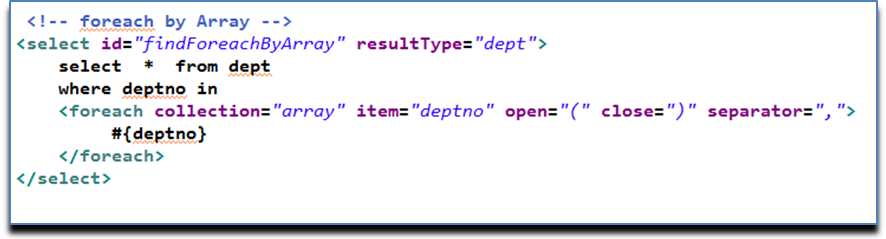
案例1.使用foreach实现批处理添加



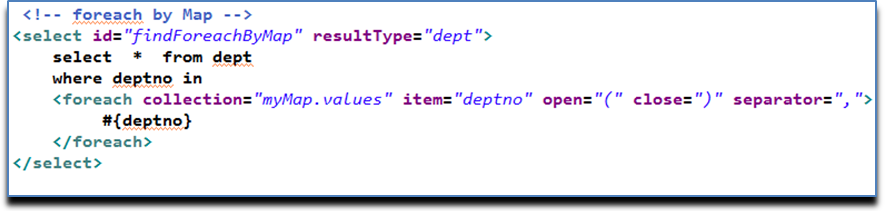
案例2.使用foreach遍历list集合作为查询条件



案例3.使用foreach遍历数组作为查询条件



案例4.使用foreach遍历Map作为查询条件



# MyBatis级联操作

在实际开发中,我们操作的表往往不是一个独立个体.它们往往根据业务依赖关系,形成一对一,一对多,多对多关联关系.为了保证业务数据的完整性.我们在操作某一张表的时候也要对与这张表关联其它表进行操作.这样的操作就成为级联操作

## 级联操作分类

### 级联查询

### 级联删除

### 级联更新

### 级联添加

## 级联查询

### 一对多查询

### 多对一查询

### 多对多查询

# MyBatis注解开发

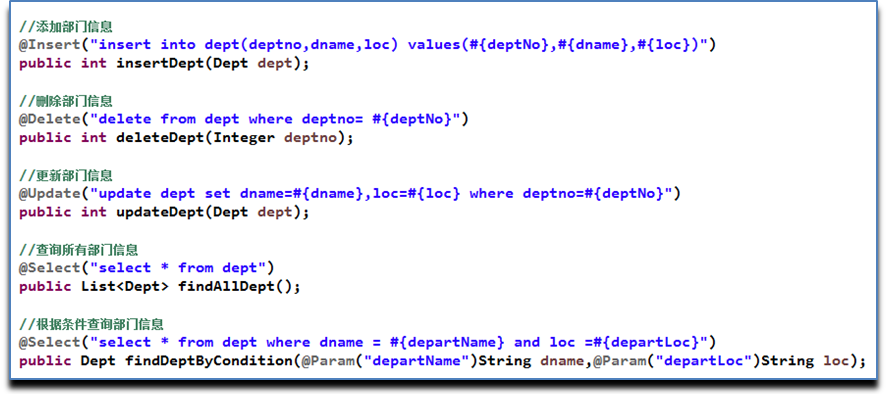
## CRUD操作

### @Insert

### @Update

### @Delete

### @Select



### @SelectProvider

### @InsertProvider

### @DeleteProvider

### @UpdateProvider