# 课前准备工作

## 开发工具:Eclipse neon1版(绿色免安装)

## jar管理工具:Maven-3.3.9(绿色免安装)

## 数据库: MySql 5.x

## 项目类型:MAVEN工程

# MyBatis框架

## MyBatis框架简介

#### MyBatis框架就是对JDBC的封装.主要目的简化JDBC开发流程,实现事务松耦合管理,将实体类与SQL命令进行动态对应.

#### 起源于Apache的Ibatis项目,2010年迁移到Google.被正式命名为MyBatis.最后在2013年迁移到Github

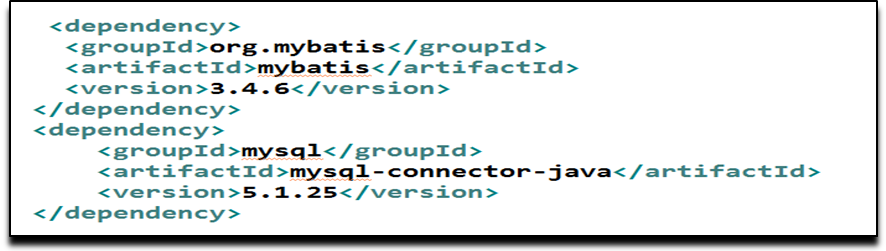
#### MyBatis框架使用简单,同时由于提供了中文官方文档,一般在一天左右即可以掌握.

## MyBatis下载

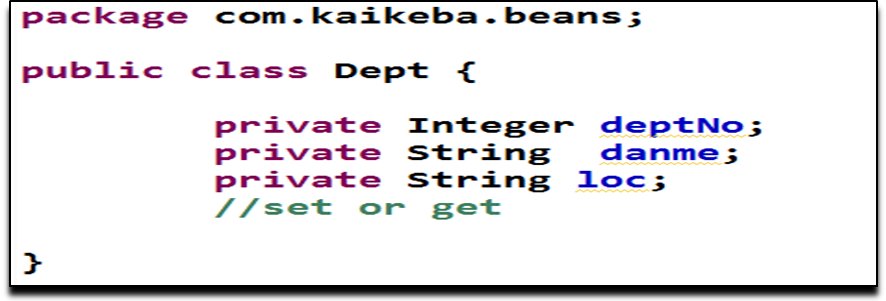
MyBatis的官网： <https://github.com/mybatis/mybatis-3>

# MyBatis开发流程

## 添加MyBatis依赖jar

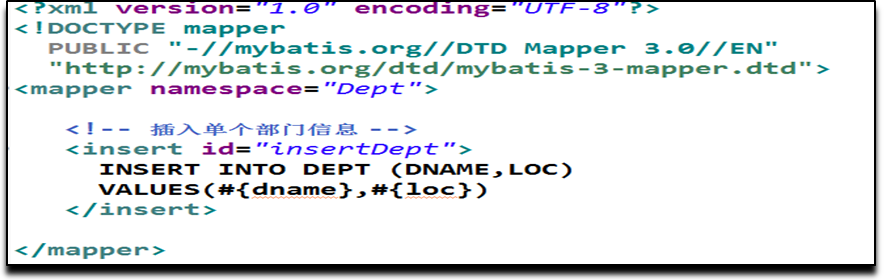


## 开发一个实体映射类



## 开发一个SQL映射文件

在src/main/resource下创建与当前表对应的SQL映射文件用于声明SQL语句

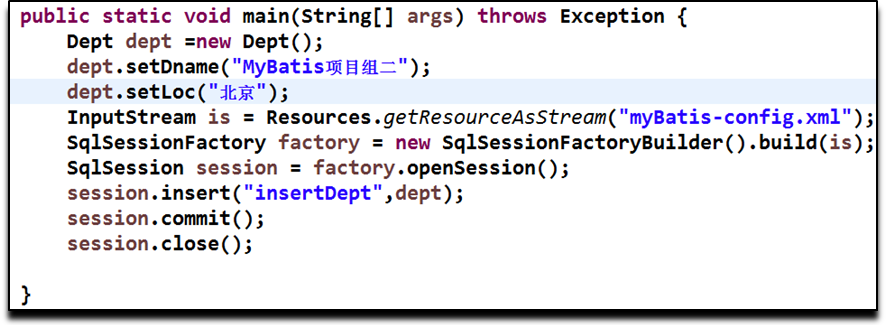


## 开发MyBatis核心配置文件

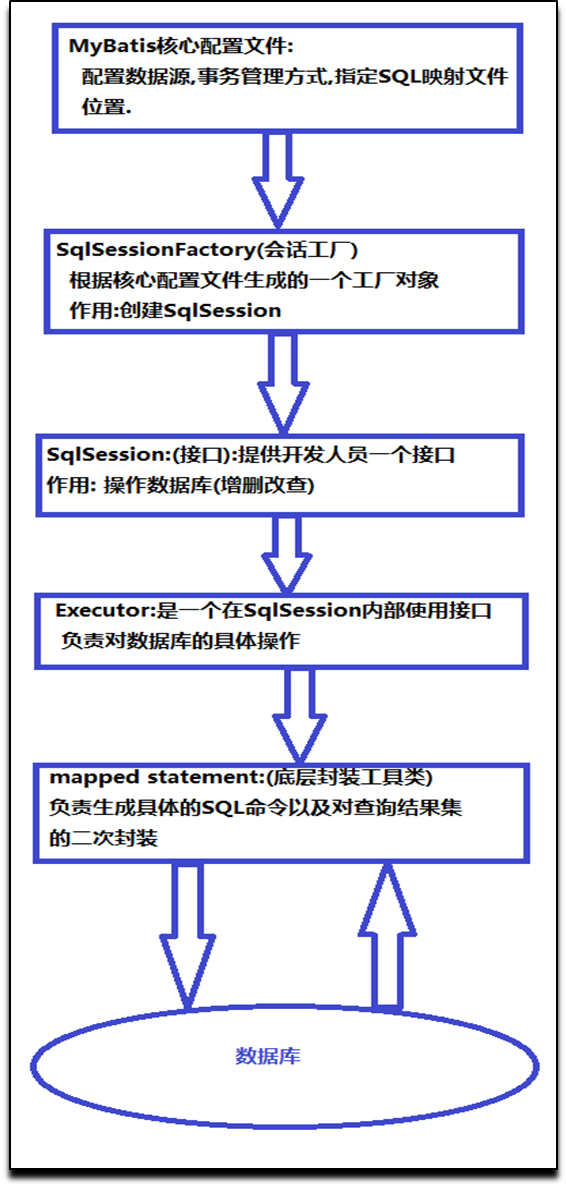
在src/main/resources下创建MyBatis-config.xml作为核心配置文件



## 开发MyBatis基本调用流程



# MyBatis工作原理与工作流程



# MyBatis单表增删改查操作

## API介绍

在SqlSession接口中提供了四个方法,实现简单的增删改查操作,分别是:

#### insert方法:实现插入

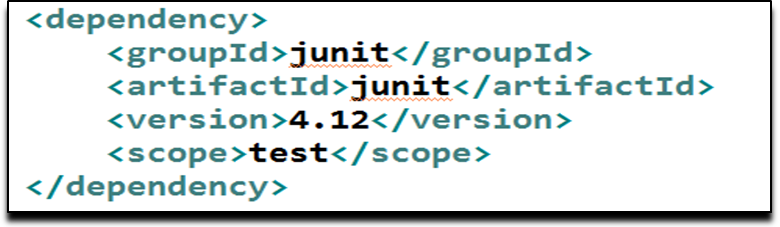
#### delete方法:实现删除

#### update方法:实现更新

#### select方法:实现查询

## 配置测试环境

### 添加Junit依赖jar



### 编写测试类

#### @Test关联的方法,是测试方法.方法声明时只能 public void.方法名可以随意定义

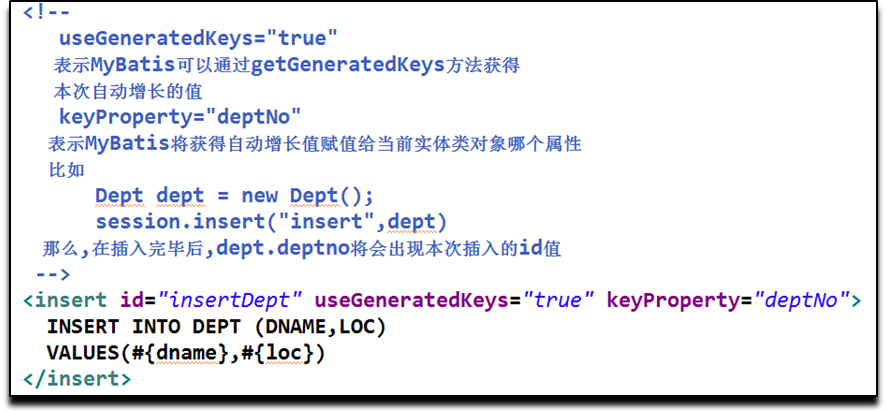
#### @Before关联的方法,会在测试方法之前执行

#### @After 关联的方法,会在测试方法之后执行

## 插入操作主键值获取

### 当前表支持主键自动增长

在JDBC技术中,可以通过Statement接口中getGeneratedKeys()方法获得本次插入后得到自动增长主键值.MyBatis框架也采用这个技术.因此MyBatis在插入完毕后也可以获得本次插入数据id.做法如下



### 当前表不支持主键自动增长

在Mysql数据库中,可以通过max函数获得当前表中最后一条插入数据id.

在MyBatis中,也可以通过这种方式来获得主键值



## 查询操作

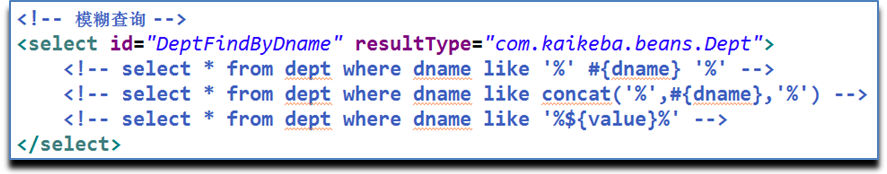
### 将查询结果封装为Map集合或则List集合

在SqlSession接口中,可以分别使用selectList方法和selectMap方法将查询结果分别封装为List集合和Map集合

### 查询单个记录

在SqlSession接口中,可以使用selectOne方法获得一个数据行并将数据行封装为一个实体类对象

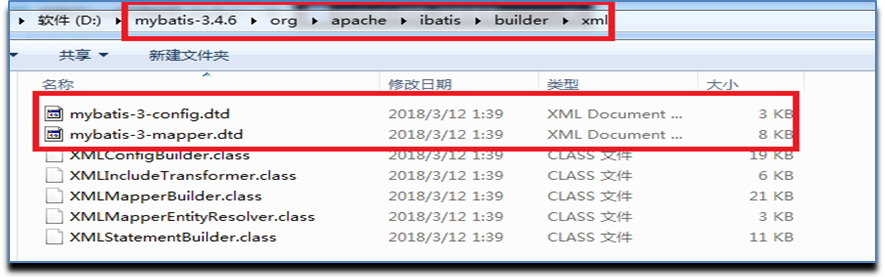
### 模糊查询



# MyBatis框架配置文件详解

## 在Mybatis配置文件获得帮助提升

我们在Mybatis的jar中发现如下两个约束文件mybatis-3-config.dtd和mybatis-3-mapper.dtd



mybatis-3-config.dtd是MyBatis核心配置文件约束

mybatis-3-mapper.data是MyBatis的SQL映射文件约束

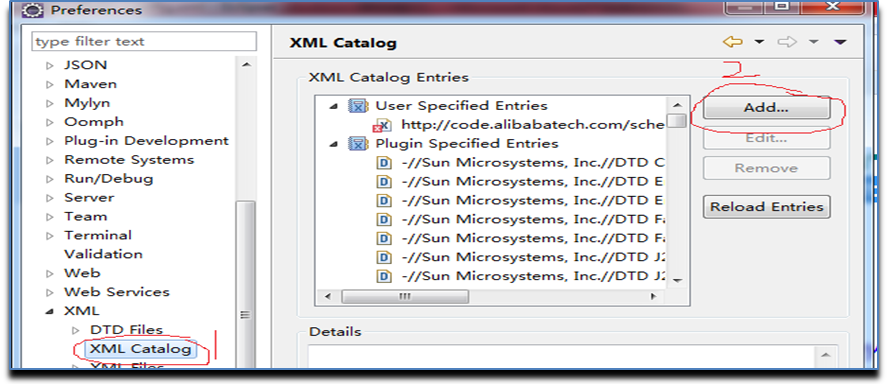
只要将这两个约束文件引入到Eclipse中,就可以在配置文件开发时获得帮助提示.

操作步骤如下: 以引入mybatis-3-config.dtd为例

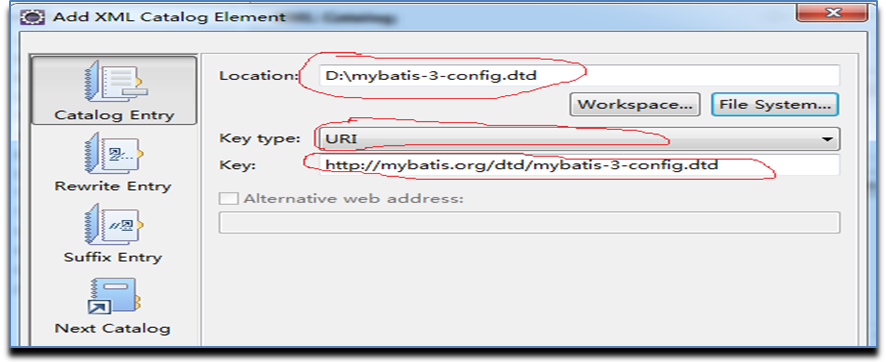
第一步:打开工程中MyBatis核心配置文件,复制约束文件地址



第二步:在window->Preferences下定位XML约束引入向导



第三步:指定添加约束文件位置



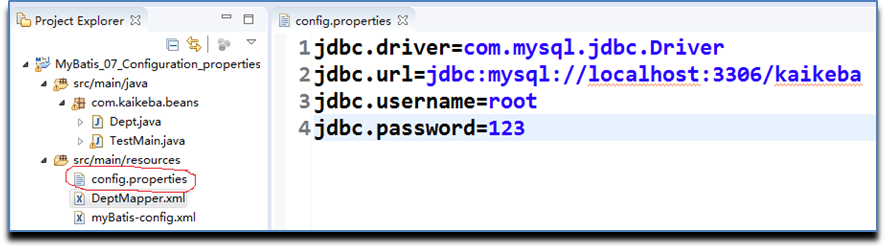
至此,我们就开在MyBatis核心配置文件获得提示帮助了

## properties

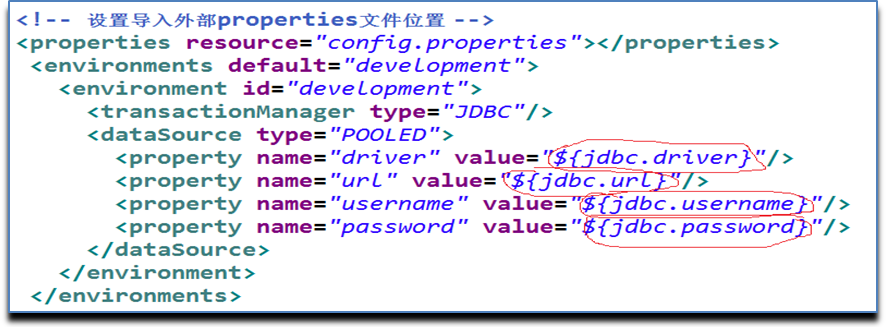
properties标签可以引入外部属性文件内容

操作如下

第一步:在工程下创建一个config.properties文件存储数据库访问信息

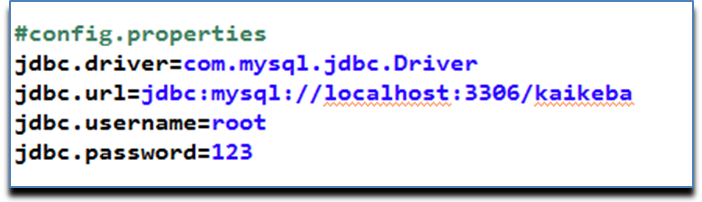


第二步:在核心配置文件引入config.propertites并修改数据源参数读取方式

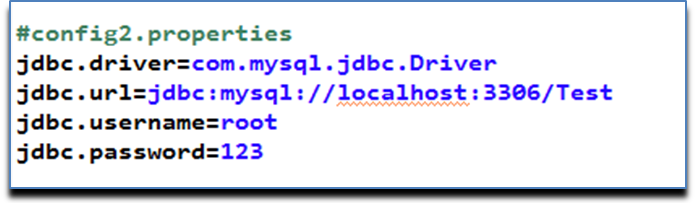


在默认情况下, SqlSessionFactoryBuilder会将properties标签指定的属性文件作为默认文件在开发环境(<environments>)使用.当然在一个项目中,也可以有多个properties属性文件,此时可以通过SqlSessionFactoryBuilder来动态指定开发环境(<environments>)依赖的properties属性文件

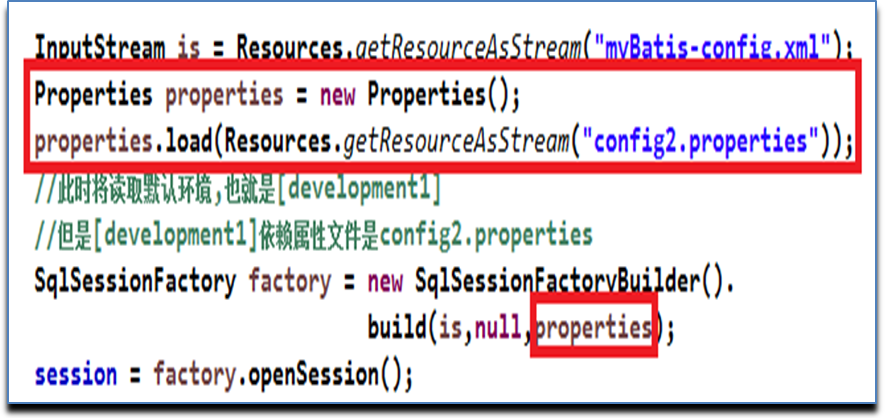
开发环境默认依赖的属性文件



在工程中新增的另一个属性文件



此时,如果我们需要开发环境依赖config2.properties,可以做如下操作



## settings

mybatis全局配置参数，全局参数将会影响mybatis的运行行为。比如：开启二级缓存、开启延迟加载。具体可配置情况如下：

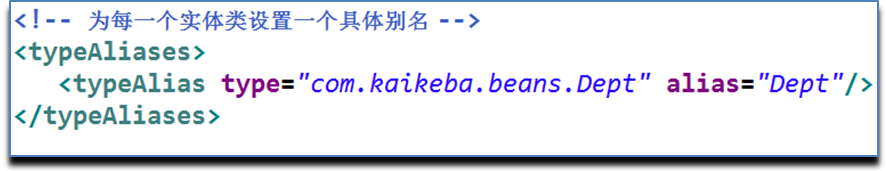


## typeAliases

在mapper.xml中，定义很多的statement，而statement需要parameterType指定输入参数的类型、需要resultType指定输出结果的映射类型。如果在指定类型时输入类型全路径，不方便进行开发，可以针对parameterType或resultType指定的类型定义一些别名，在mapper.xml中通过别名定义，方便开发。

typeAliases是MyBaits框架中提供别名转换器,可以对使用实体类名称设置一个简短的别名,从而简化开发负担.使用方式如下

方式一:为每一个实体类都设置一个别名

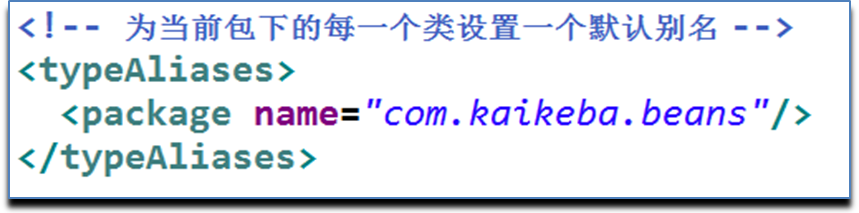


此时,在SQL映射文件就可以使用这个”Dept”别名了



如果工程中实体类个数比较多,那么用第一种方式就不会很方便.

方式二:为某个包下所有的类设置默认别名,此时别名就是当前类的简单名称



## environments 环境

MyBatis 可以配置多种环境。这会帮助你将 SQL 映射应用于多种数据库之中。但是要记得一个很重要的问题：你可以配置多种环境，但每个数据库对应一个 SqlSessionFactory。

所以，如果你想连接两个数据库，你需要创建两个 SqlSessionFactory 实例，每个数据库对应一个。而如果是三个数据库，你就需要三个实例，以此类推。

为了明确创建哪种环境，你可以将它作为可选的参数传递给 SqlSessionFactoryBuilder。

可以接受环境配置的两个方法签名是：

SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader, environment);

SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader,environment,properties);

如果环境被忽略，那么默认环境将会被加载，按照如下方式进行：

SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader);

SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader,properties);

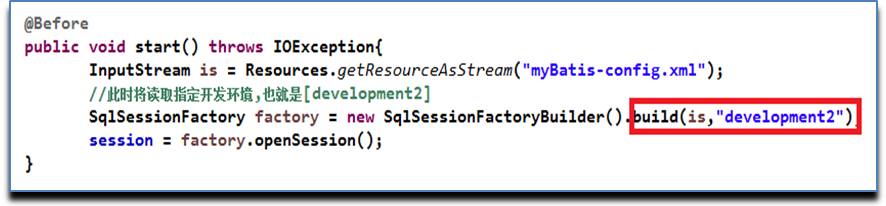
如下图所示,我们配置了两个开发环境,[development1]和[development2].默认的开发环境是[development1].



在开发时,如果需要使用[development2],此时可以通过

SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader, environment);

来制定

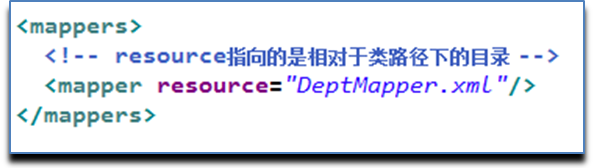


## databaseIdProvider 数据库厂商标识

## mappers 映射器

mappers标签是MyBatis框架提供的SQL映射文件的加载器,用于指定项目中SQL映射文件的位置.用法如下

### 第一种（常用） <mapper resource=" " /> resource指向的是相对于类路径下的目录 如：<mapper resource="sqlmap/User.xml" />



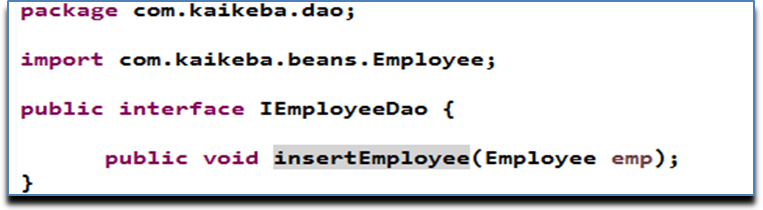
### 第二种 <mapper url=" " /> 使用完全限定路径 如：<mapper url="file:///D:\workspace\mybatis1\config\sqlmap\User.xml" />



### 第三种 <mapper class=" " /> 使用mapper接口类路径 如：<mapper class="cn.kang.mapper.UserMapper"/> 注意：此种方法要求mapper接口名称和mapper映射文件名称相同，且放在同一个目录中。

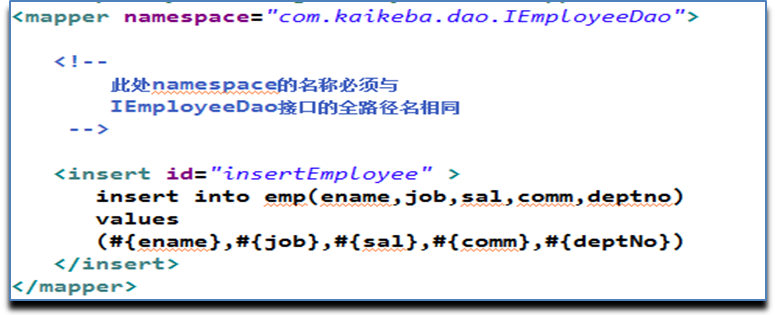
操作如下

第一步:在[src/main/java]创建接口[com.kaikeba.dao.IEmployeeDao]

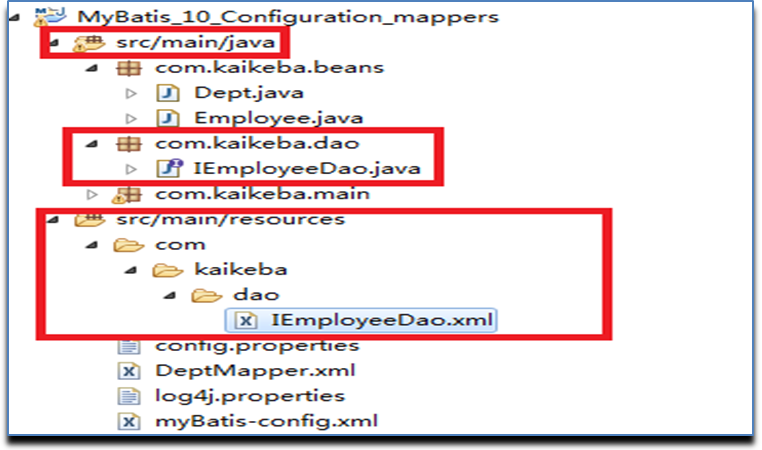


第二步:在[src/main/resources]下的[com.kaikeba.dao]文件夹下创建[IemployeeDao.xml]

的SQL映射文件



这两个文件在MAVEN工程位置如下



### 第四种（推荐） <package name=""/> 注册指定包下的所有mapper接口 如：<package name="cn.kang.mapper"/> 注意：此种方法要求mapper接口名称和mapper映射文件名称相同，且放在同一个目录中。



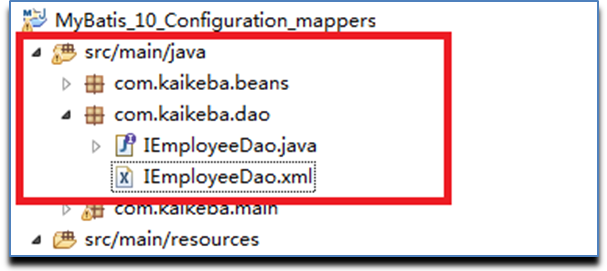
### MyBatis中接口和对应mapper文件配置深入解析

首先要说明的问题是，Mybatis中接口和对应的mapper文件不一定要放在同一个包下，放在一起的目的是为了Mybatis进行**自动扫描**，并且要注意此时**java接口的名称和mapper文件的名称要相同**，否则会报异常，由于此时Mybatis会自动解析对应的接口和相应的配置文件，所以就不需要配置mapper文件的位置了。

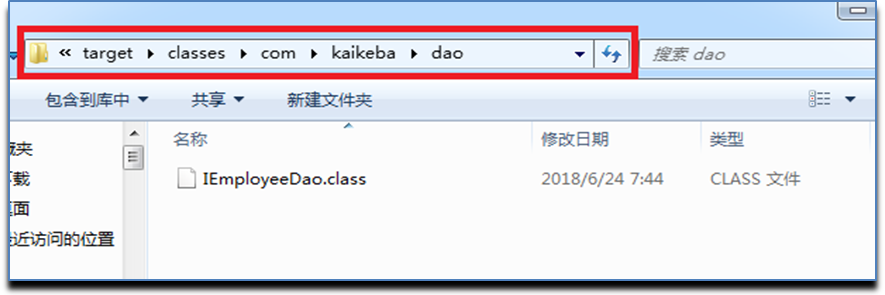
#### 默认MAVEN构建

如果在工程中使用了maven构建工具，那么就会出现一个问题：我们知道在典型的maven工程中，目录结构有：src/main/java和src/main/resources，前者是用来存放java源代码的，后者则是存放一些资源文件，比如配置文件等，**在默认的情况下maven打包的时候，对于src/main/java目录只打包源代码，而不会打包其他文件**。所以此时如果把对应的mapper文件放到src/main/java目录下时，不会打包到最终的jar文件夹中，也不会输出到target文件夹中，由于在进行单元测试的时候执行的是/target目录下/test-classes下的代码，所以在测试的时候也不会成功。

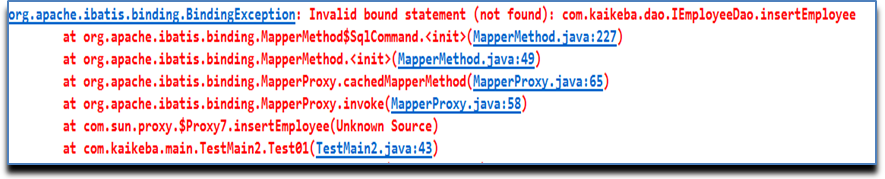
如下情况



此时我们在target中并不会发现有IemployeeDao.xml文件存在

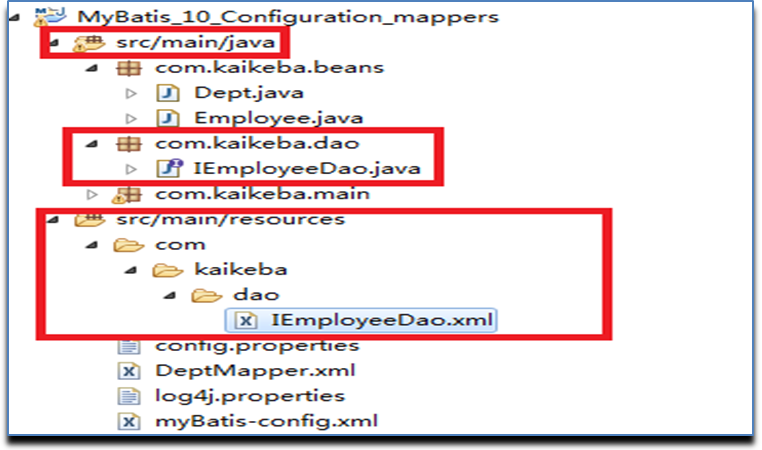


此时运行程序会抛出如下异常

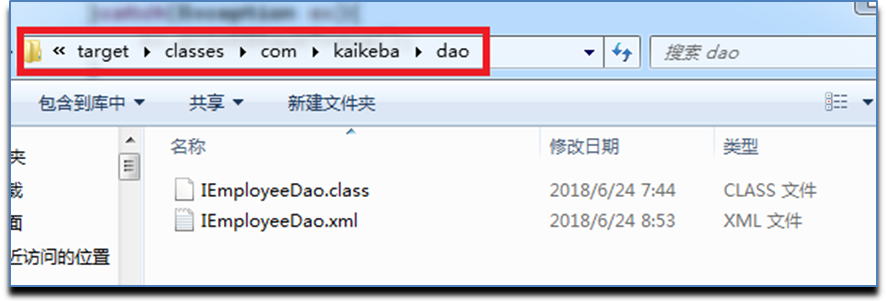


**为了实现在maven默认环境下打包时，Mybatis的接口和mapper文件在同一包中，可以通过将接口文件放在src/main/java某个包中，而在src/main/resources目录中建立同样的包**，这是一种约定优于配置的方式，这样在maven打包的时候就会将src/main/java和src/main/resources相同包下的文件合并到同一包中。

如下图所示



此时在target下,我们是可以看到IemployeeDao.xml文件的



此时程序可以正常运行

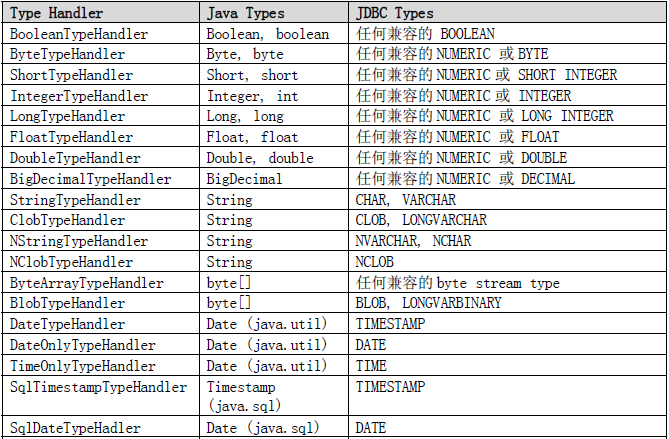
#### 更改MAVEN配置

如果不想将接口和mapper文件分别放到src/main/java和src/main/resources中，而是全部放到src/main/java，那么在构建的时候需要指定maven打包需要包括xml文件，具体配置如下：



## 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



# MyBatis框架Mapper配置文件详解

## Mapper配置文件标签介绍

#### insert – 映射插入语句

#### update – 映射更新语句

#### delete – 映射删除语句

#### select – 映射查询语句

#### sql – 可被其他语句引用的可重用语句块

#### resultMap-确定实体类属性与表中字段对应关系

## mapper中nameSpace

<mapper>标签是SQL映射文件中根目录标签.在这个标签中只有输一个属性

<mapper namesapce=””>

### namespace属性有什么作用呢?

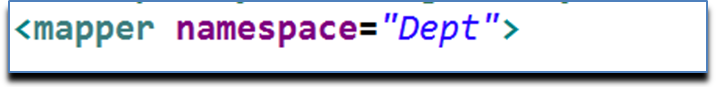
在MyBatis中，Mapper中的namespace用于绑定Dao接口的，即面向接口编程。

它的好处在于当使用了namespace之后就可以不用写接口实现类，业务逻辑会直接通过这个绑定寻找到相对应的SQL语句进行对应的数据处理

### namespace属性赋值规则

#### 规则1: 短名称（比如“selectAllThings”）如果全局唯一也可以作为一个单独的引用

nameSpace的值可以是一个简短的唯一字符串



#### 规则2:接口完全限定名

也可以是当前工程中一个接口完整路径



## parameterType属性

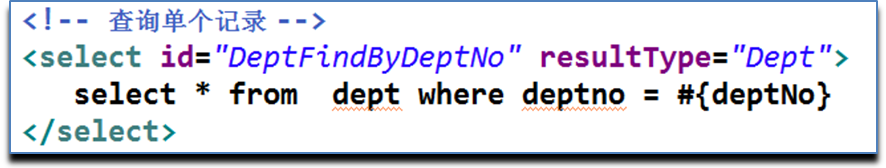
在<insert>,<update>,<select>,<delete>标签中,可以通过parameterType指定输入参数的类型，类型可以是简单类型、hashmap、pojo的包装类型。

parameterType属性是可以省略的.MyBatis框架可以根据SqlSession接口中方法的参数

来判断输入参数的实际数据类型.

## 参数(#{参数名})

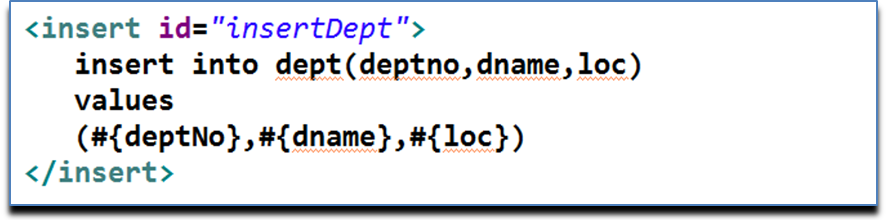
#{}实现的是向prepareStatement中的预处理语句中设置参数值，sql语句中#{}表示一个占位符即?



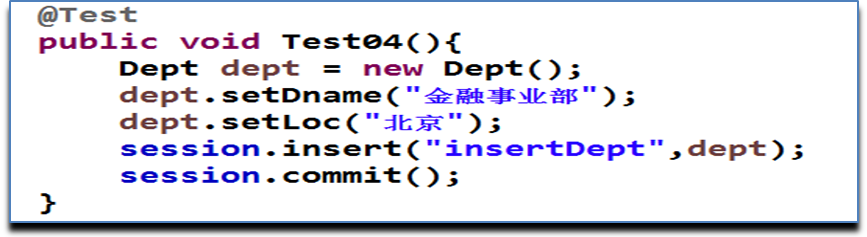
使用#{参数名},将参数的内容添加到sql语句中指定位置.

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

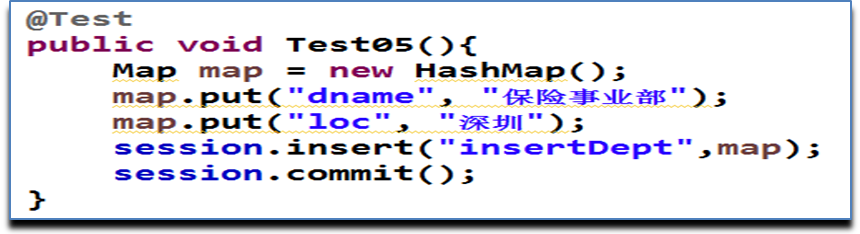
但是,如果当前sql语句有多个参数,此时参数名称应该是与当前表关联[实体类的属性名]或则[Map集合关键字]



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

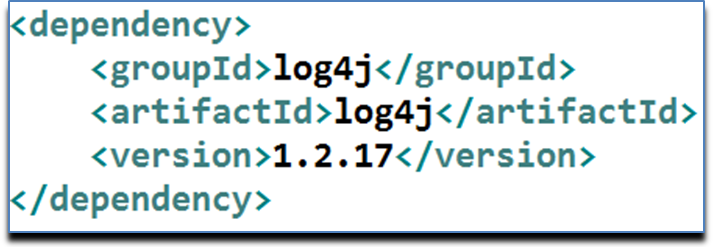
#{参数名} :实体类对象或则Map集合读取内容

${参数名} :实体类对象或则Map集合读取内容

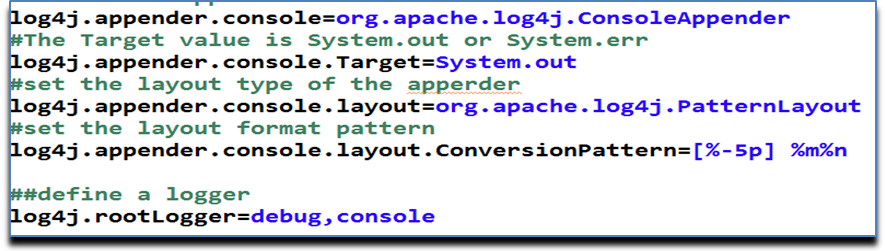
为了能够看到两种方式的区别,需要看到MyBatis执行时输送的SQL情况.因此

需要借助Log4J日志进行观察

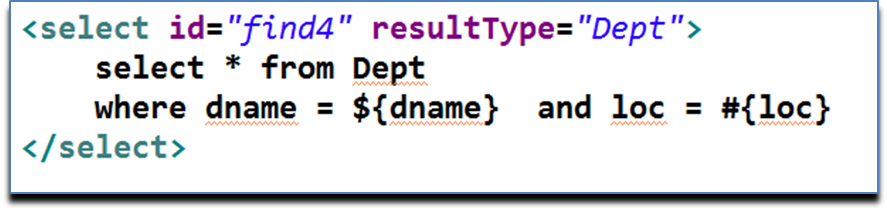
第一步: 加载Log4j日志工具包到项目



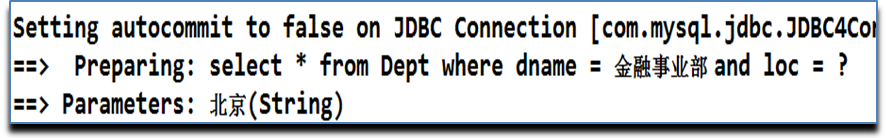
第二步:将Log4j配置文件添加到src/main/resources下



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${}

## resultType属性

#### resultType属性存在<select>标签.负责将查询结果进行映射.

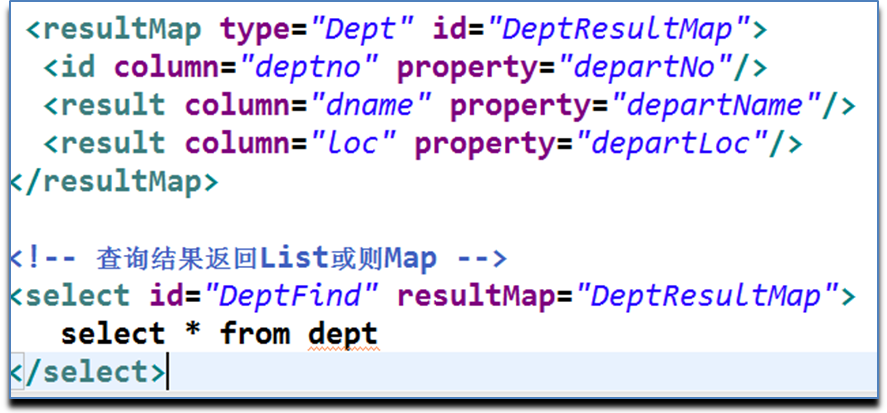
#### resultType属性可以指定一个基本类型也可以是一个实体类类型

#### 使用resultType属性为实体类类型时，只有查询出来的列名和实体类中的属性名一致，才可以映射成功. 如果查询出来的列名和pojo中的属性名全部不一致，就不会创建实体类对象.但是只要查询出来的列名和实体类中的属性有一个一致，就会创建实体类对象

#### resultType属性无法与resultMap属性同时出现.

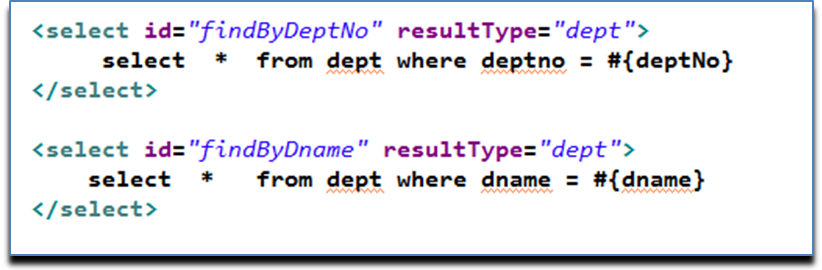
## resultMap

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.



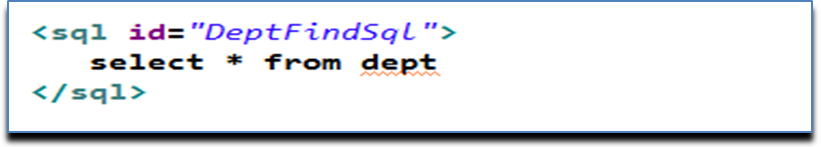
## sql标签

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.



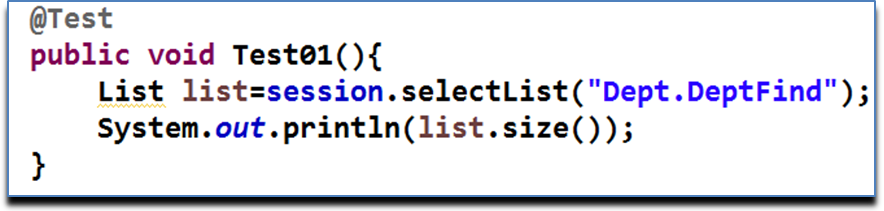
在需要使用到这个查询的地方,通过<include>标签进行引用



# MyBatis接口编程

## 传统的MyBatis开发流程问题

在讲Mybatis接口式编程之前，我们先回忆一下前面是如何调用映射文件中的SQL代码的。通常情况下，都是使用SqlSession实例的selectXXX(selectOne, selectList, selectMap)方法来执行映射文件中相应的SQL语句的，这些方法都有一个共同的特征，那就是第一个参数都是String类型的，我们需要使用这个参数明确告之Mybatis我们是需要执行映射文件的哪一个元素下的SQL语句，所以这个参数内容应该是映射文件的名称空间加上相应元素的id值，如：



这里存在一些潜在的问题：

* 为了确保名称空间的唯一性，通常会使用相对较长的、且有一定含义的字符串来作为其值，这样就很难保证我们在代码不出现拼写错误的情况，即使是直接从映射文件拷贝过来的，也存在不经意间被修改的可能性；
* 从selectXXX方法的签名可以看到，她的第二个参数是Object类型，那么如果我们传入的参数类型与映射文件中由parameterType属性指定的类型不一致时，将会出现不可预知的错误
* 同样，selectXXX方法返回值使用了泛型，我们须确保用于接收其返回值的变量类型与映射文件中属性resultType指定的类型相一致

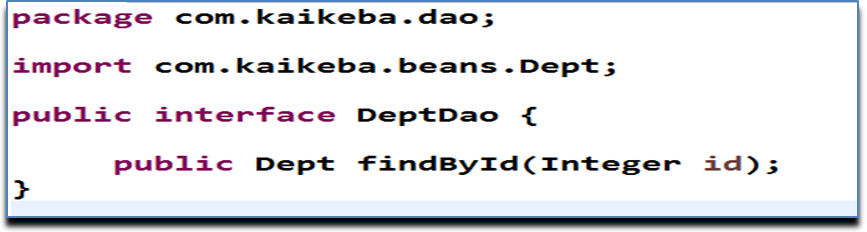
Mybatis为了规避上述风险,提供了接口编程

## 什么是接口编程

接口式编程，我们可以简单的理解为Mybatis为映射文件定义了一个代理接口，以后全部通过这个接口来和映射文件交互，而不再是使用以前方法。

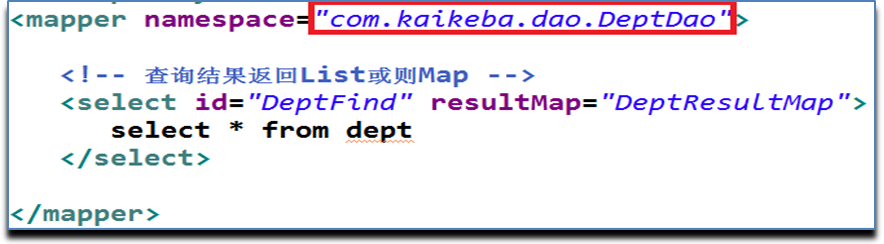
## 接口编程实现

#### 声明一个接口

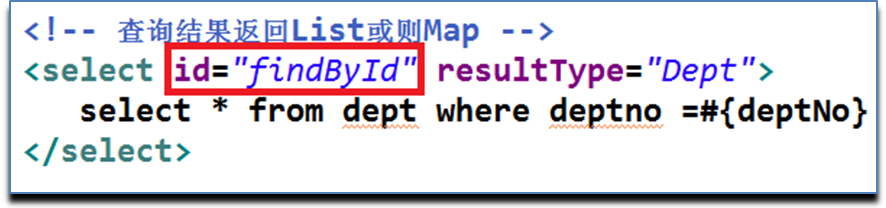


#### 修改Mapper文件命名空间

命名空间应该是接口完整名称

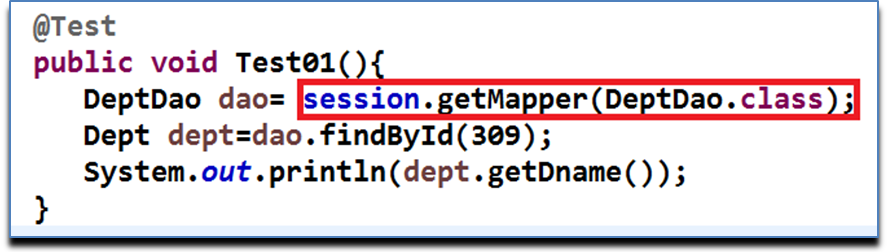


#### 修改Mapper文件SQL语句ID



#### 测试调用

#### 



# 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

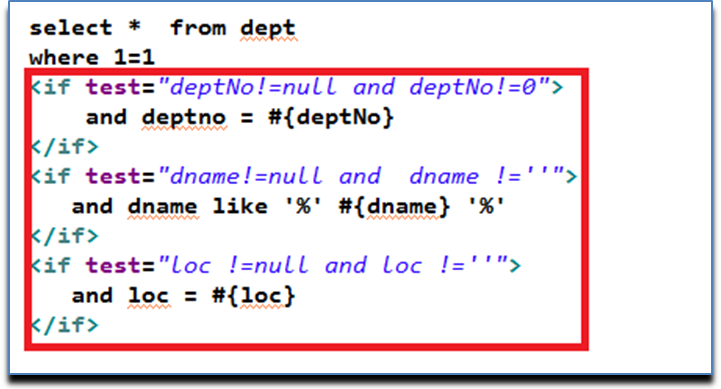
### if

### choose、when、otherwise

### trim、where、set

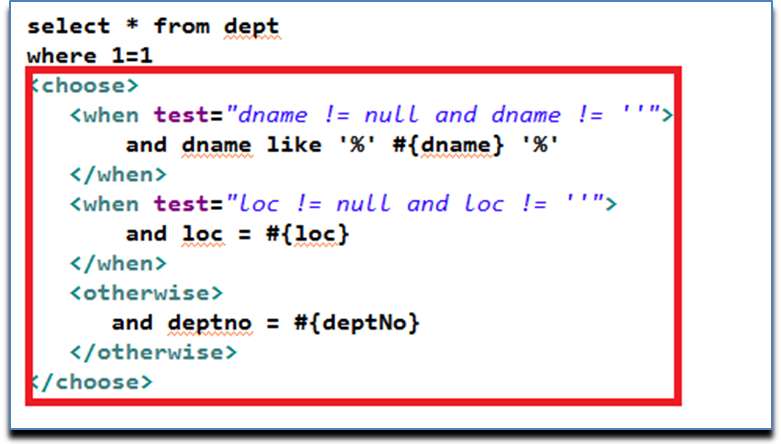
### foreach

## if使用



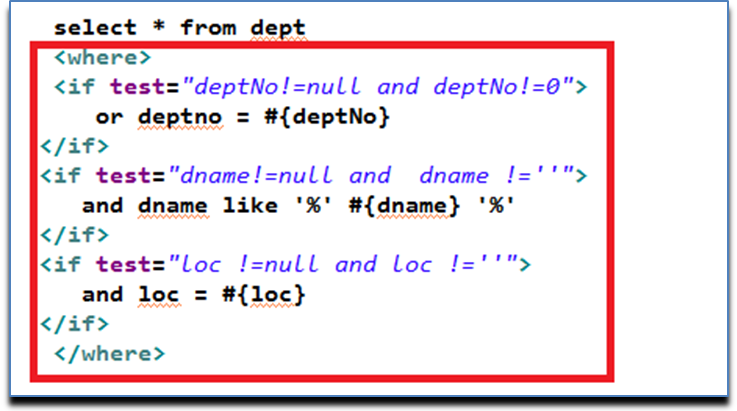
## ****choose、when、otherwise****

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



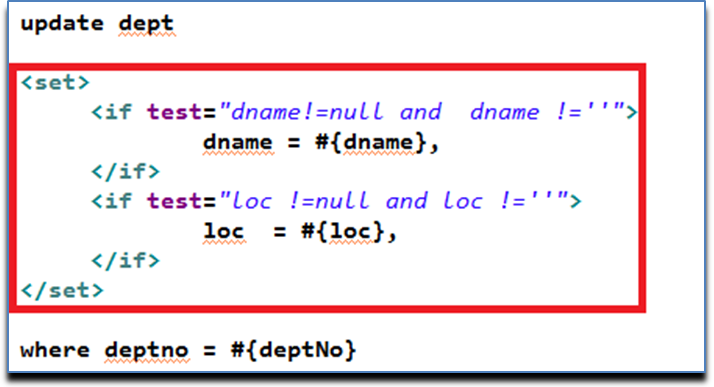
## where

<where>可以自动的将第一个条件前面的逻辑运算符(or ,and)去掉

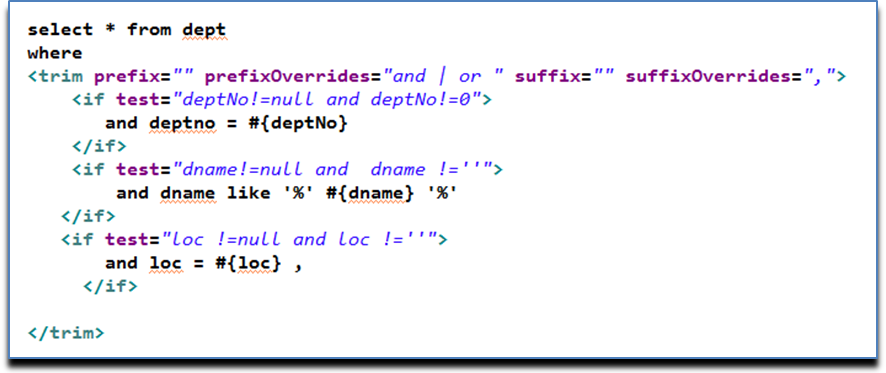


## set

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

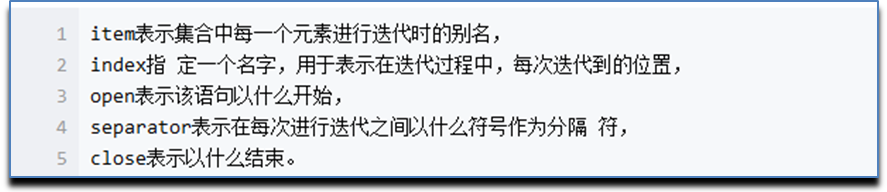


## foreach

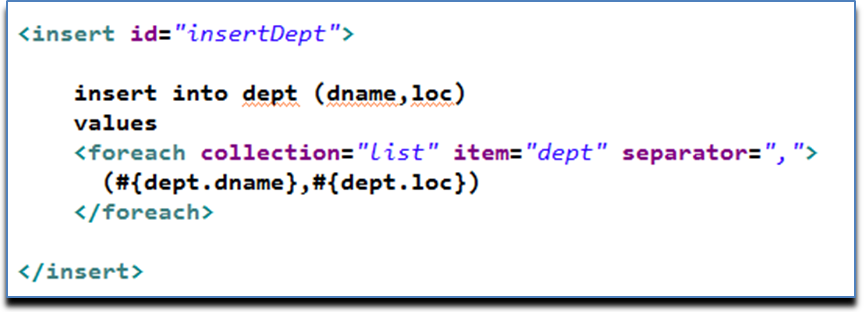
foreach标签用于对集合内容进行遍历,将得到内容作为SQL语句的一部分.

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

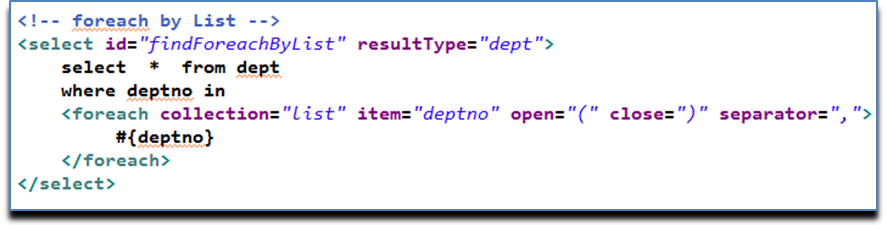
foreach元素的属性主要有 item，index，collection，open，separator，close。



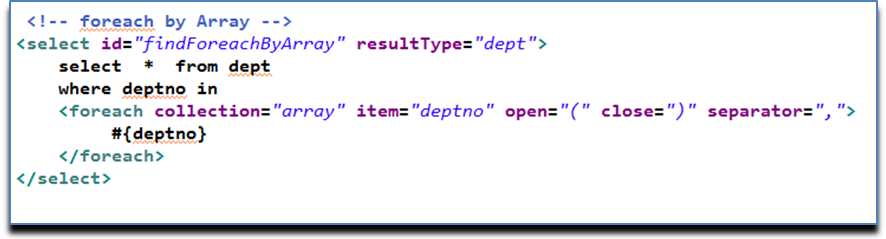
案例1.使用foreach实现批处理添加



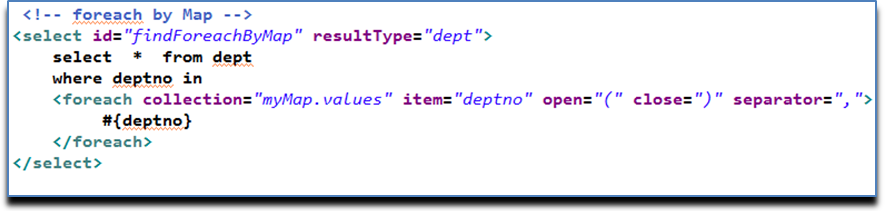
案例2.使用foreach遍历list集合作为查询条件



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

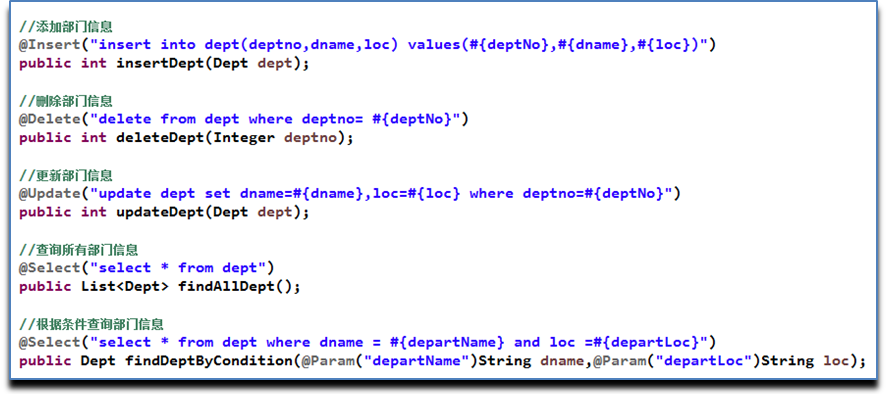
## CRUD操作

### @Insert

### @Update

### @Delete

### @Select



### @SelectProvider

### @InsertProvider

### @DeleteProvider

### @UpdateProvider