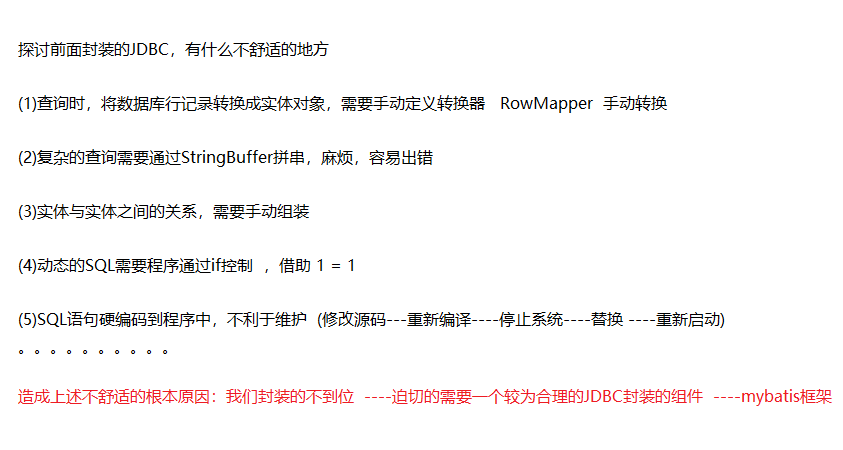
# 1.探讨现有JDBC的不足



# 2.MyBatis框架

## 2.1 简介

(1)MyBatis前生是ibatis框架

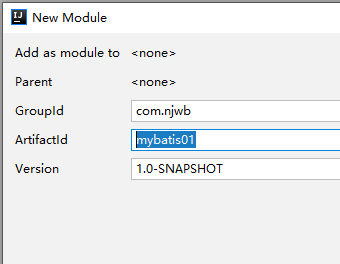
(2) MyBatis是一个持久层(DAO)的框架

(3)MyBatis对JDBC的高度封装，几乎消除了所有的JDBC代码，不需要手动处理参数和结果集的检索，MyBatis通过简单的XML或注解做映射。

## 2.2 搭建框架

以后遇到框架，想两件事：1、框架需要的jar包(maven管理) 2、配置文件(百度、开发手册、开发包。。。。。。)

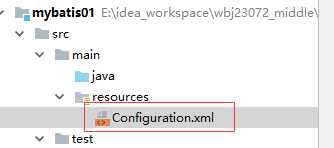
(1)创建maven工程



(2)编写pom.xml

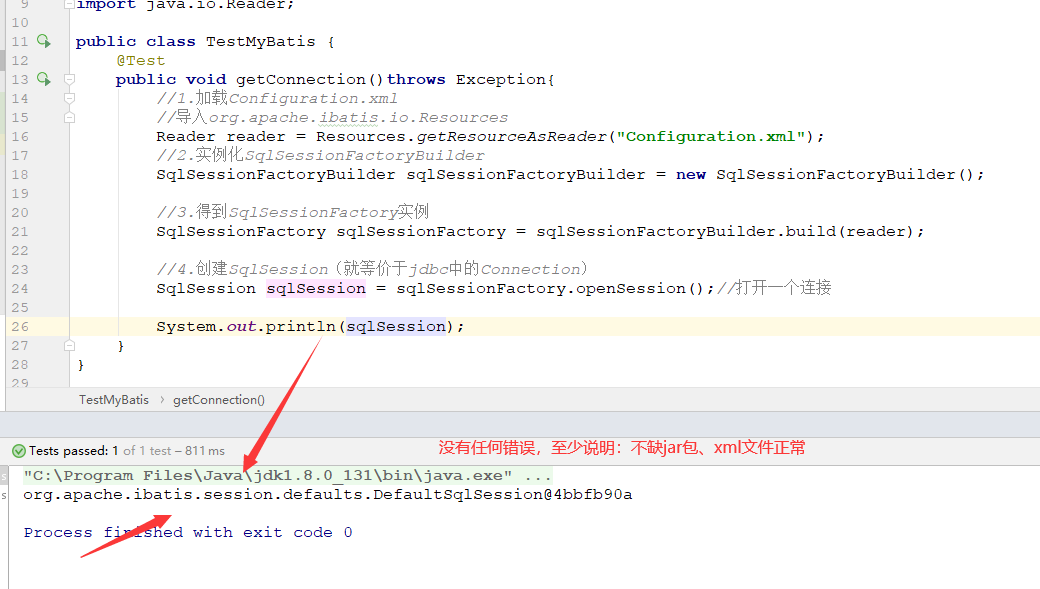
<**dependencies**>  
 *<!--驱动包-->* <**dependency**>  
 <**groupId**>com.oracle.database.jdbc</**groupId**>  
 <**artifactId**>ojdbc8</**artifactId**>  
 <**version**>12.2.0.1</**version**>  
 <**scope**>runtime</**scope**>  
 </**dependency**>  
 *<!--mybatis-->* <**dependency**>  
 <**groupId**>org.mybatis</**groupId**>  
 <**artifactId**>mybatis</**artifactId**>  
 <**version**>3.5.13</**version**>  
 </**dependency**>  
 *<!--测试包-->* <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.13.1</**version**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
  
</**dependencies**>

(3)src/main/resources下定义一个配置文件





(4)测试连接



# 3.CRUD操作

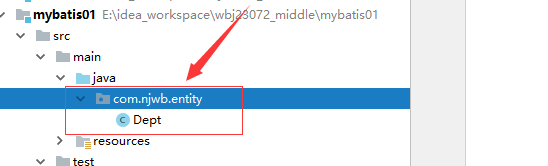
## 3.1 查询单个值

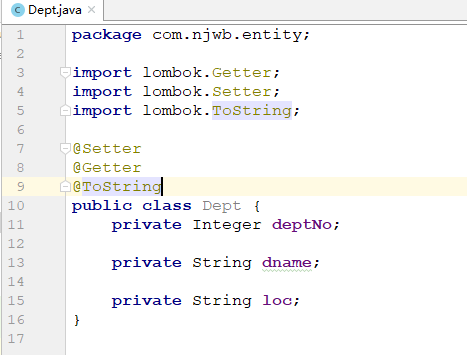
需求：根据部门编号查询部门信息

(1)引入lombok依赖

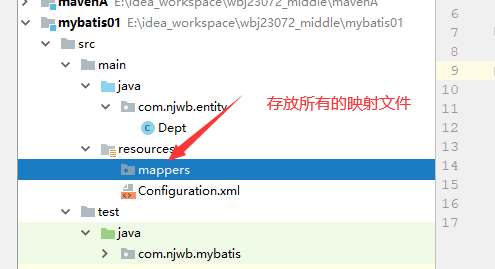
*<!--lombok插件-->*<**dependency**>  
 <**groupId**>org.projectlombok</**groupId**>  
 <**artifactId**>lombok</**artifactId**>  
 <**version**>1.18.28</**version**>  
 <**scope**>provided</**scope**>  
</**dependency**>

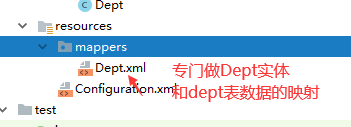
(2)定义实体类





(3)定义SQL映射文件





抄手册



(4)在Configruation.xml主配置文件中注册映射文件



(5)测试代码



底层原理：

(1)加载映射文件，解析



(2)selectOne(key,参数)

内部找到相应的Statement标签

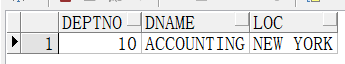
SQL : select deptno,dname,loc from dept where deptno = #{deptNo}

含有#{} 语法 认为SQL中有占位符

---->翻译 ----> select deptno,dname,loc from dept where deptno = ?

设置参数，根据parameterType：int类型，把selectOne(key,参数) 参数设置到占位符上

select deptno,dname,loc from dept where deptno = 10



(3)返回值

根据resultType = com.njwb.entity.Dept 就用Dept对象作为返回值对象

反射实例化Dept对象

对Dept属性赋值

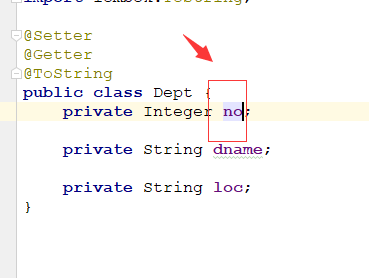
一行中的列 ----->dept对象中的属性

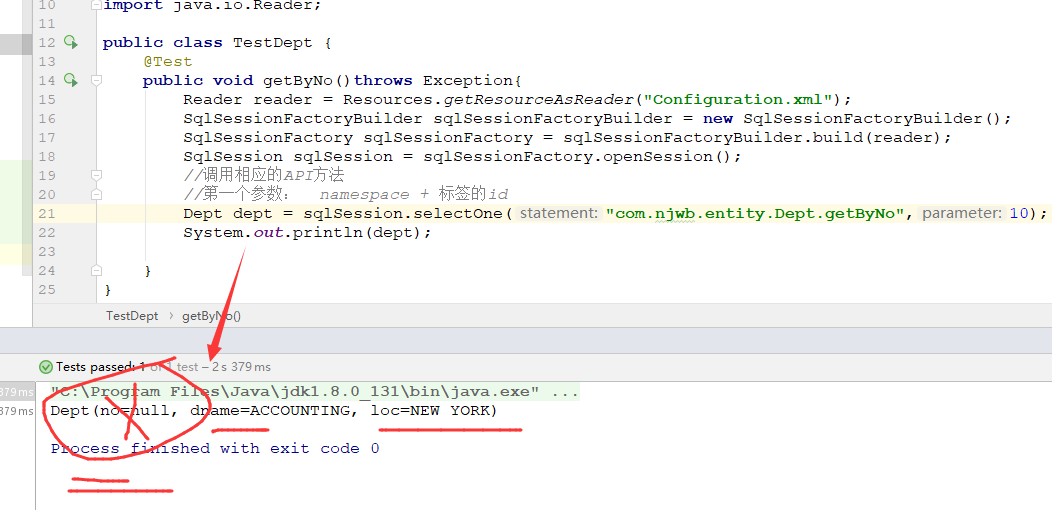
DEPTNO -------->dept对象的deptNo属性

set+DEPTNO 忽略大小写的方式找对应的setDeptNo方法

自动的将列值 映射到了实体的属性值上。

要求：如果数据库的字段和属性名忽略大小写的情况下还不一致 --->装不上的。





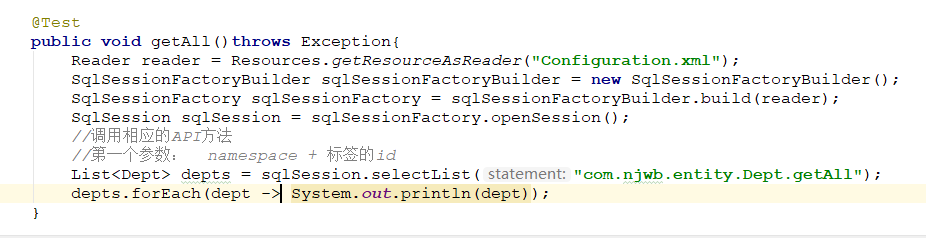
## 3.2 查询多个值

需求：查询所有的部门数据

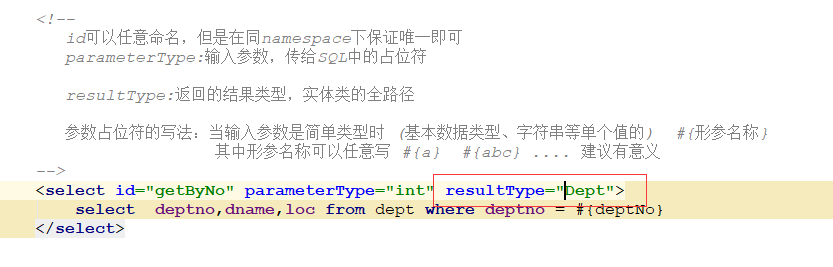
(1)Dept.xml中定义SQL语句



(2)测试



## 3.3 别名机制

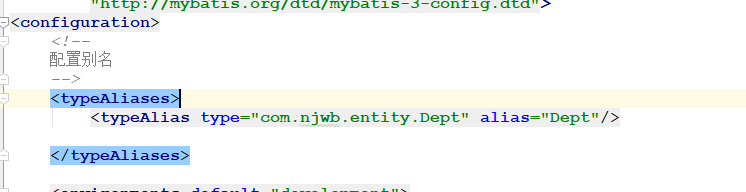




映射文件中出现的类型名称，要么写成包名+类名的全路径形式

要么写成别名(短一点)形式

主配置文件中配置别名：



再次运行 OK



建议采用下面的配置方法



测试

mybatis提供了一系列的内置别名

(1)基本数据类型，包装类型，全部写小写

(2)Java.lang.String 别名是 string

(3)java.util.Date 别名是date

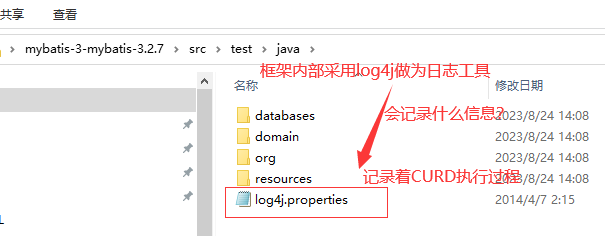
(4)java.util.List 别名是list

java.util.Map 别名是map

java.util.HashMap 别名是hashmap

java.util.ArrayList 别名是arraylist

## 3.4 放开日志



导入log4j包

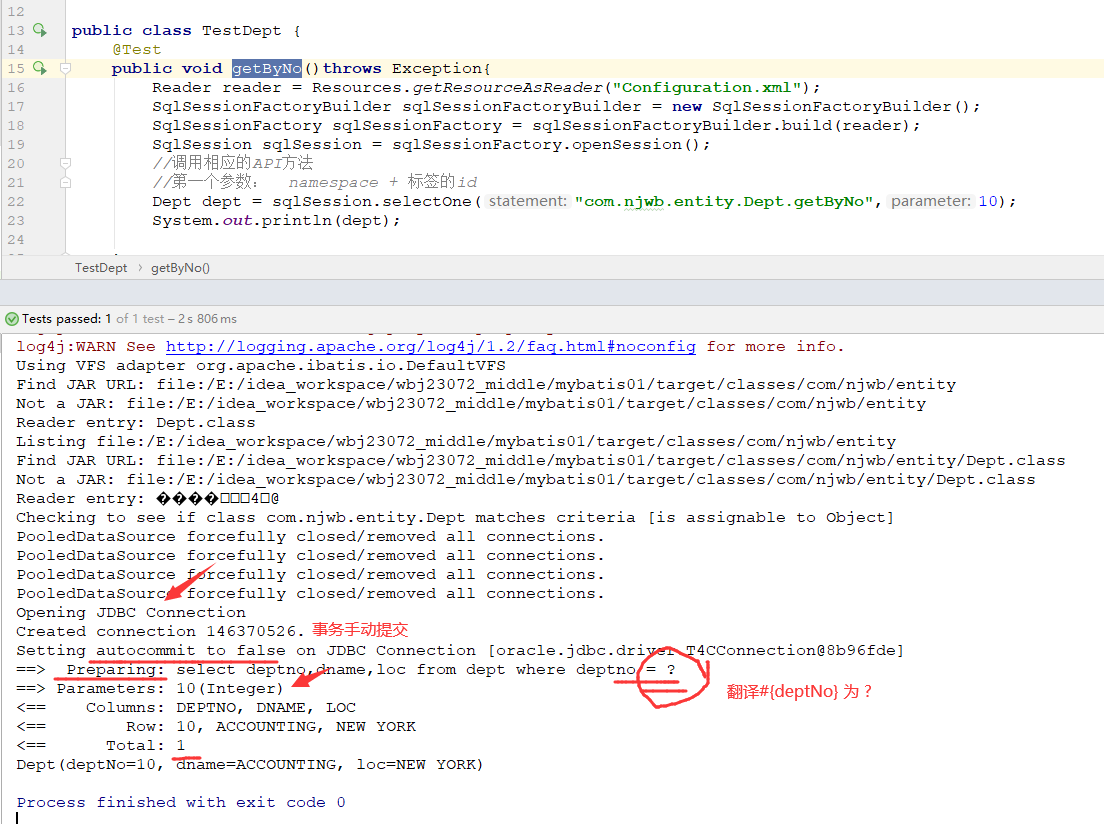
<**dependency**>  
 <**groupId**>log4j</**groupId**>  
 <**artifactId**>log4j</**artifactId**>  
 <**version**>1.2.17</**version**>  
</**dependency**>

放开日志

主配置文件中配置

<**settings**>  
 <**setting name="logImpl" value="STDOUT\_LOGGING"**/>  
</**settings**>

测试



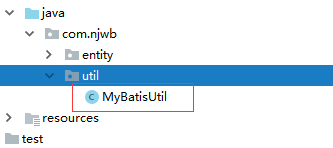
## 3.5 新增

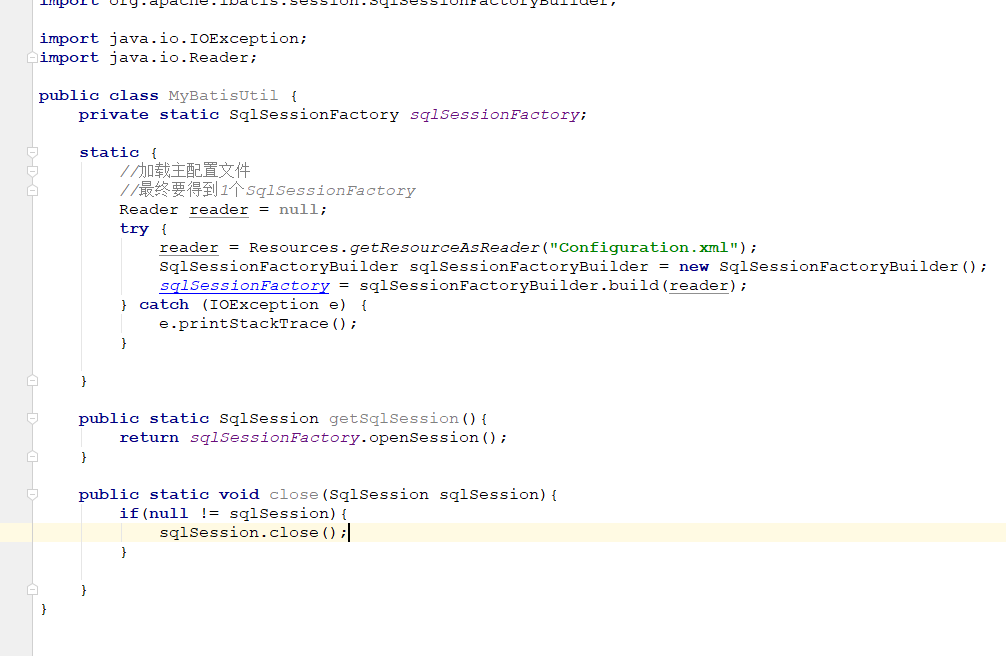
新增一个部门

(1)定义SQL



(2)封装一个工具类MyBatisUtil





(3)测试添加

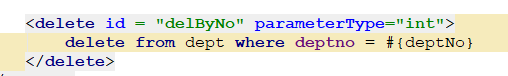


## 3.6 修改





## 3.7 删除





## 3.8 总结

使用MyBatis框架时，关键点

(1)在映射文件中定义合适的标签

(2)选择适当的API方法

selectOne

selectList

insert

delete

update

# 3.输入参数探讨

SQL语句的输入参数

ibatis时期使用**~~parameterMap~~**，mybatis时期则使用新的parameterType

## 3.1 简单类型参数

简单值 ： 单个值

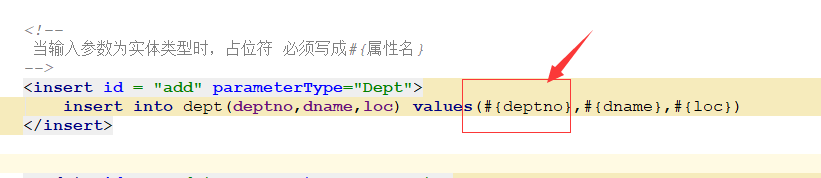
比如：基本数据类型、String等

当parameterType采用简单类型参数时，SQL中的占位符#{任意值}

## 3.2 实体参数

当parameterType采用实体类型参数时，SQL中的占位符#{属性名}

通过反射获取数据





## 3.3 Map输入参数

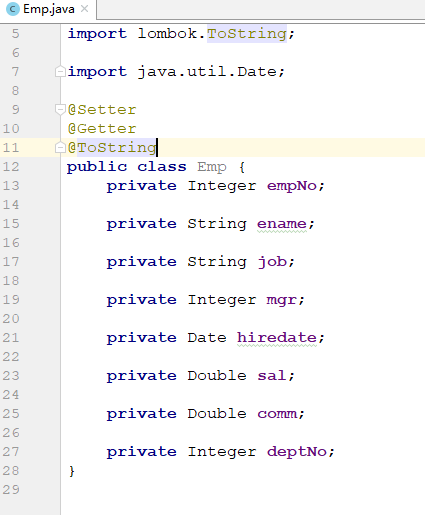
当parameterType采用Map类型参数时，SQL中的占位符#{key名}

需求：按照薪资降序排列分页查询员工信息，每页显示5条

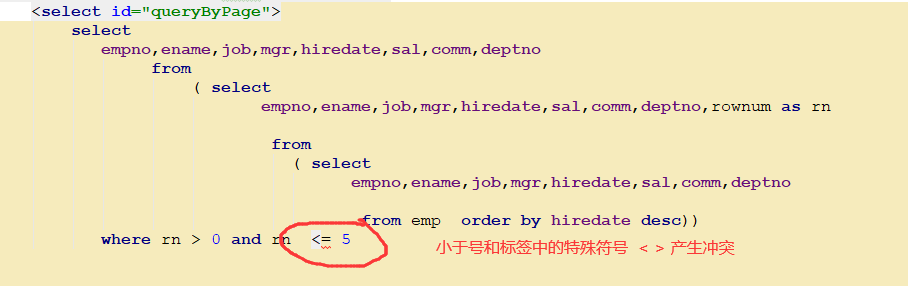
(1)SQL



(2)Emp实体类

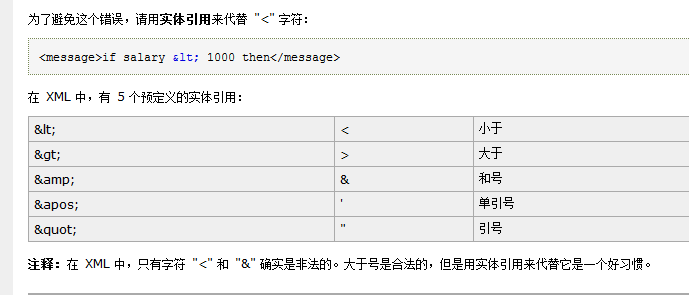


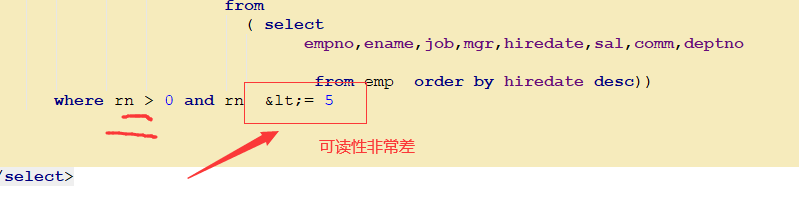
(3)Emp.xml映射文件



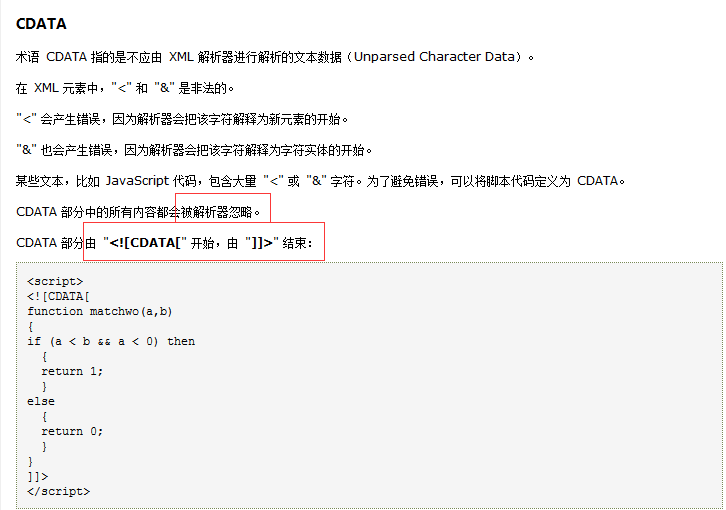
如何解决：

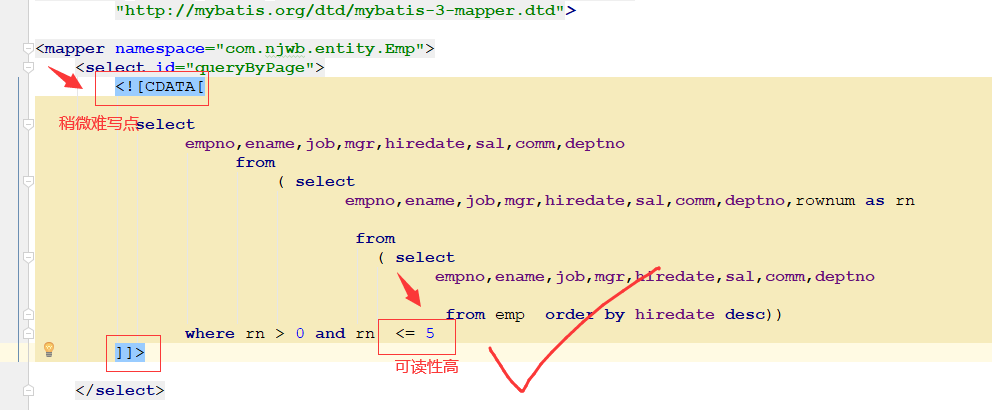
a.转移字符



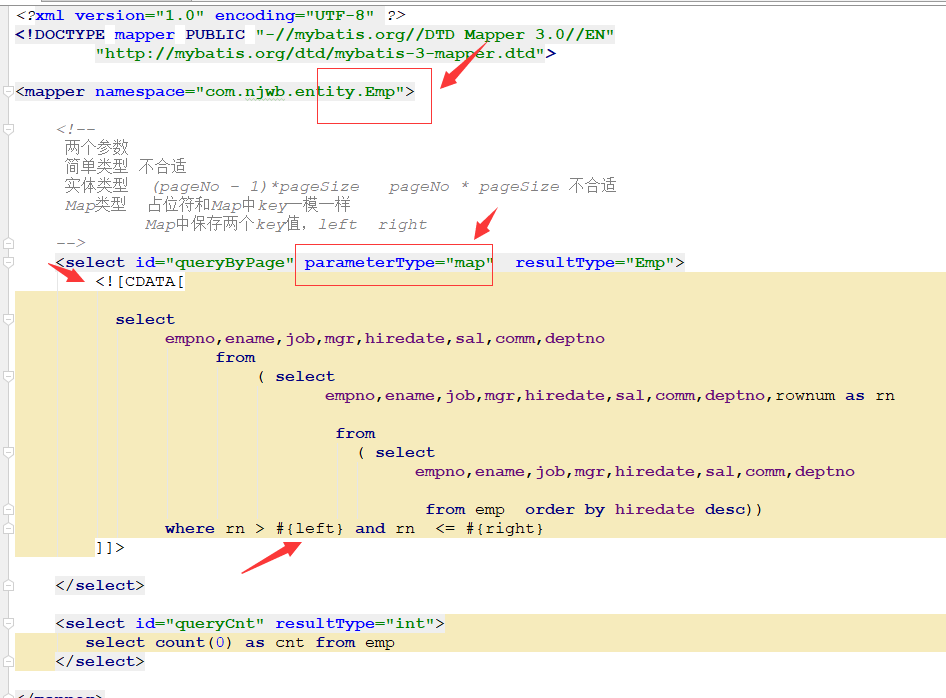


b.CDATA

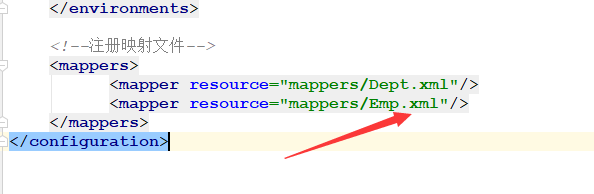




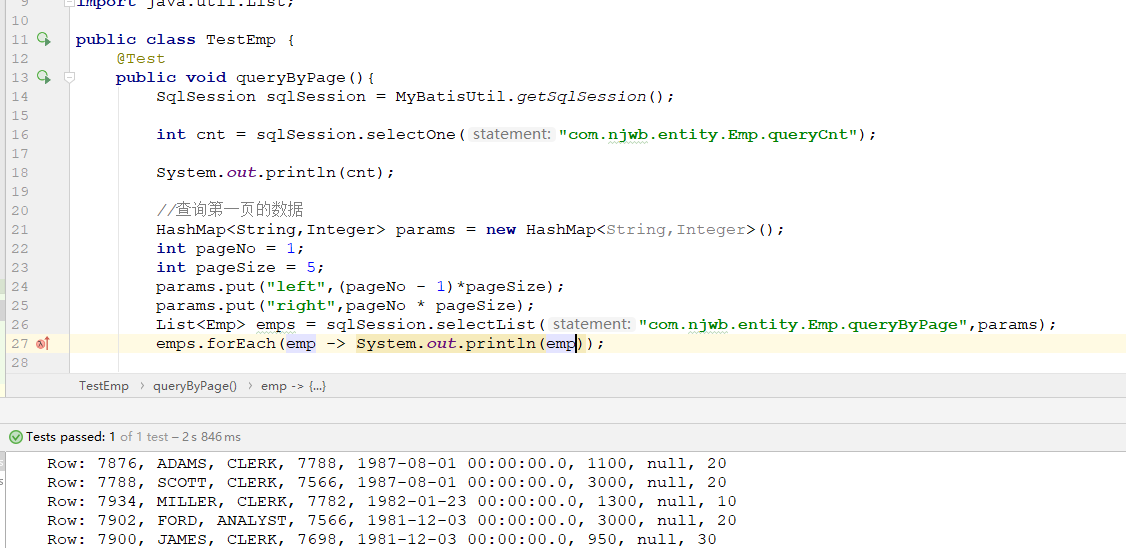
最终



(4)注册映射文件



(5)测试查询



# 4.输出参数探讨

## 4.1 resultType

适合使用

(1)简单类型

如：返回记录总数 整数 resultType = “int”

(2)实体类型

要求：查询的列必须和实体中的属性名一致(忽略大小写)

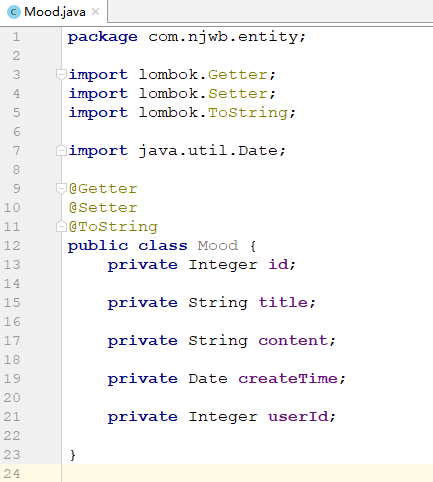
## 4.2 resultMap

当字段和实体属性名不一致(忽略大小写)的时候，手动配置映射

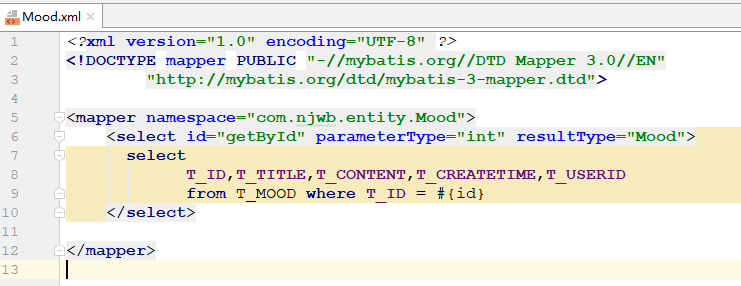
需求：根据id查询Mood对象

(1)SQL

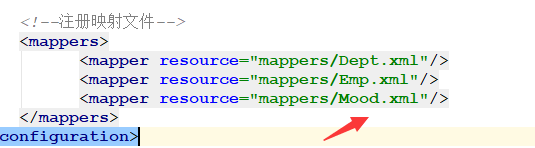
(2)Mood实体



(3)Mood.xml



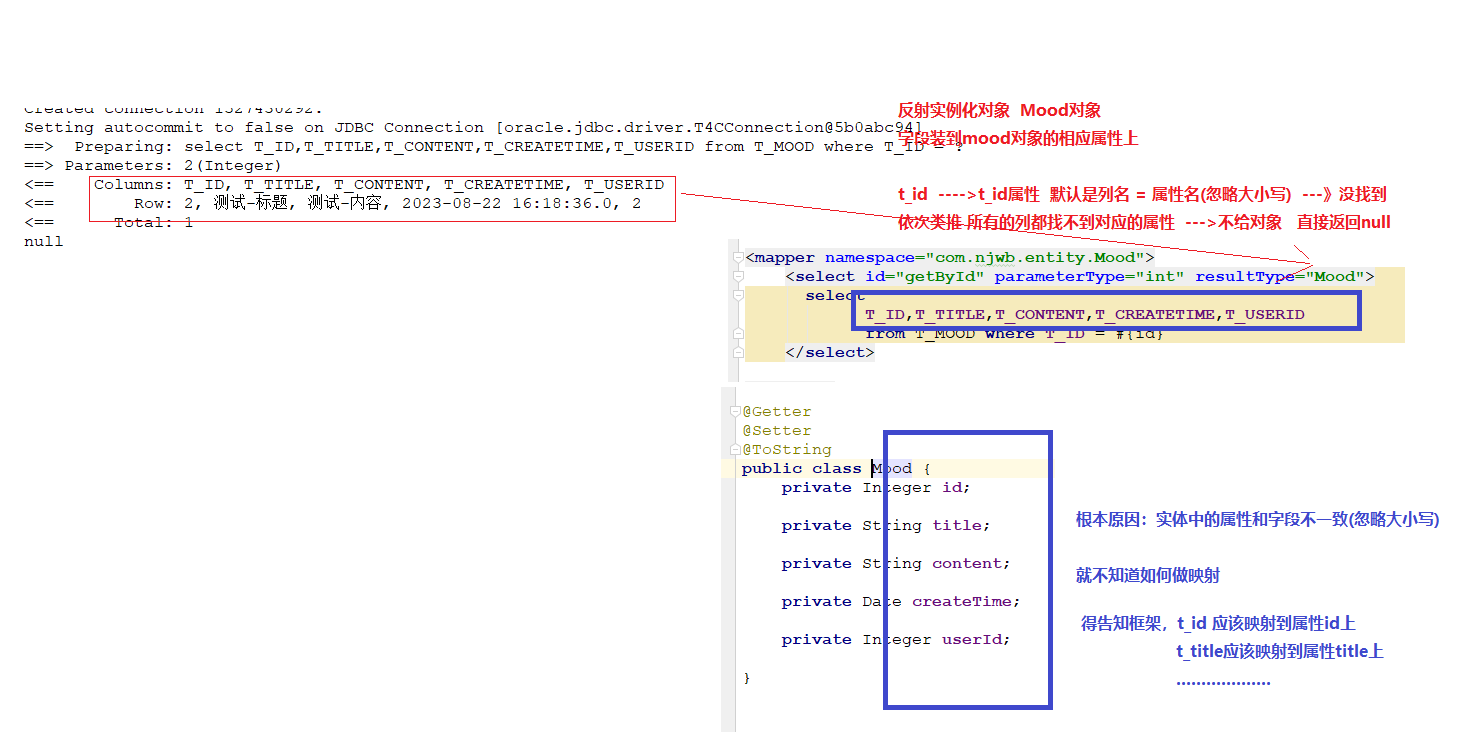
(4)注册映射文件



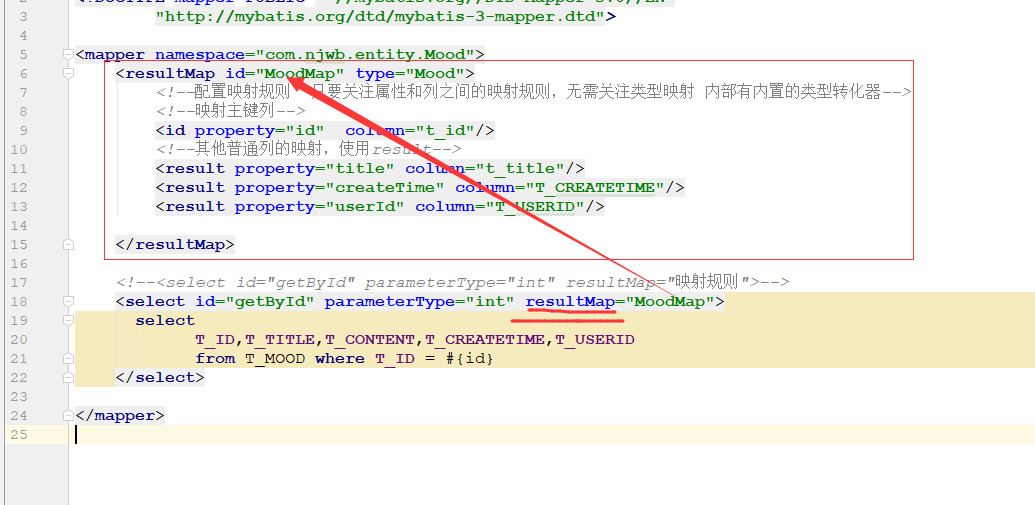
(5)测试



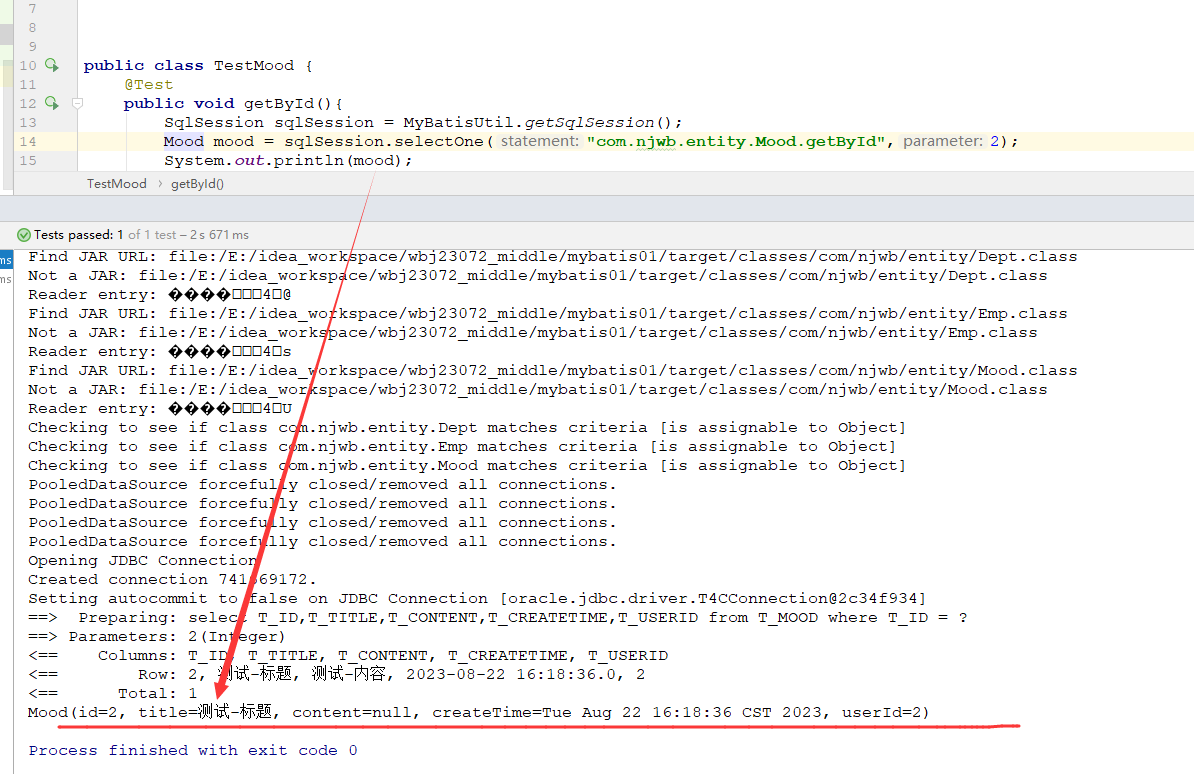
问题：能够将该数据查询出来，但是结果为null？



解决办法：



测试



小结：

resultMap使用的频率远远的高于resultType

# 5.范围和生命周期

不正确的使用会造成严重的并发问题。

SqlSessionFactoryBuilder：

作用：构建一个SqlSessionFactory，可以被实例化，使用，用完之后就可以丢弃，所以它的生命周期比较短，在方法范围内，且作为局部变量使用(方法执行时，入栈，执行完毕后出栈，局部变量结束了声明周期)



SqlSessionFactory：

作用：创建SqlSession连接

SqlSessionFactory应该在整个应用期间有效，且唯一，比较好的做法：单例

更好的做法：后面将学习的依赖注入容器。



SqlSession:

作用：CRUD操作

该对象是线程不安全对象，只能用在方法级别(多线程下，每个线程都拥有自己独立的方法栈区)

固定的模式

public someMethod(){

SqlSession

try{

CRUD

}finally{

关闭连接

}

}

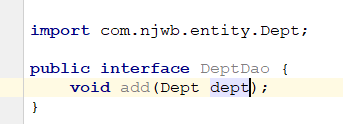
# 6.MyBatis下的DAO模式

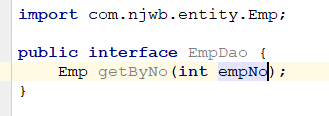
只要保证SqlSession对象在DAO方法内部定义即可----不会造成线程不安全事件。

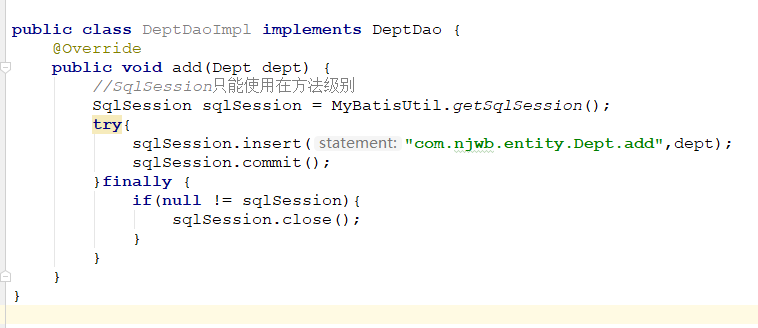
需求：采用DAO模式实现

根据员工编号查询员工信息

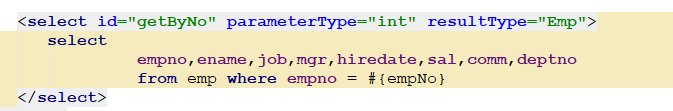
添加一个部门



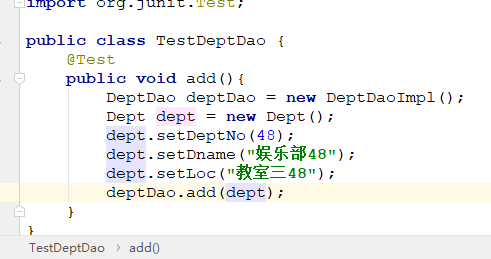


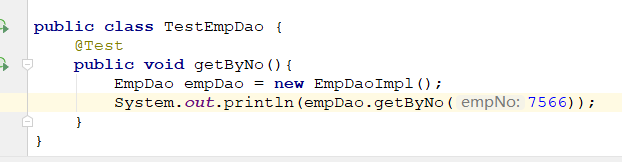




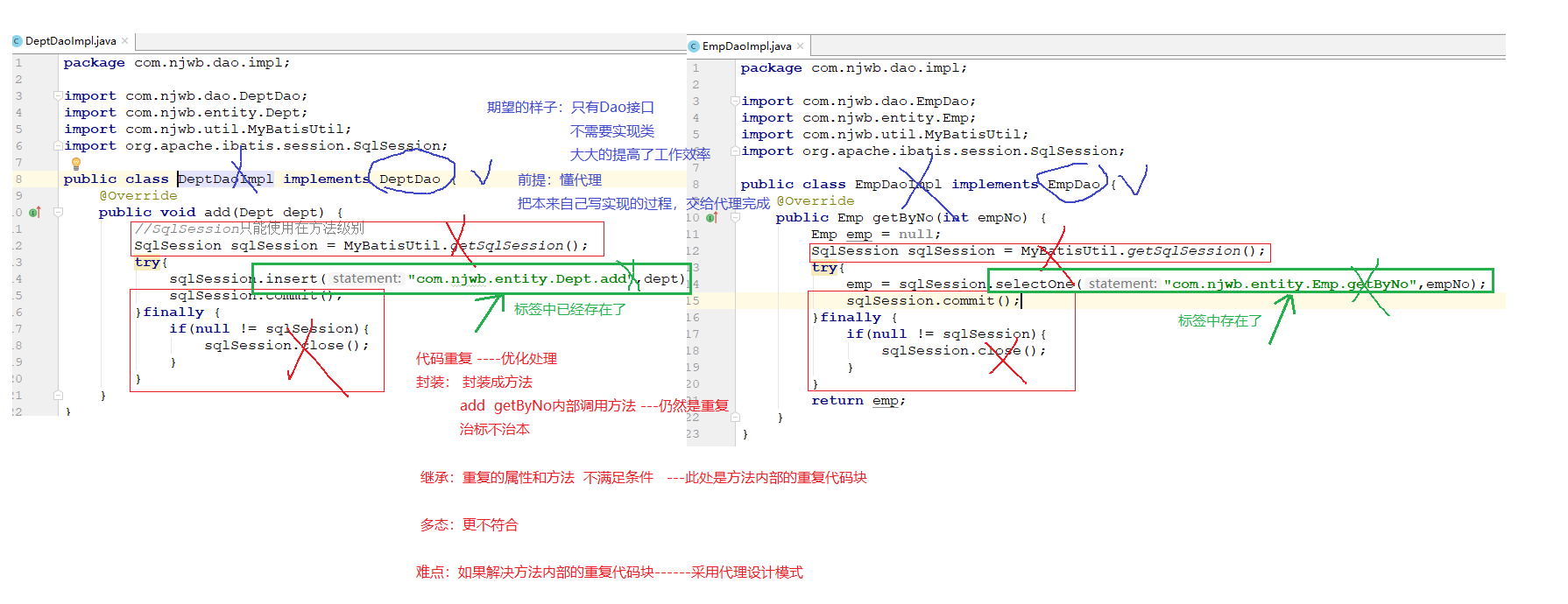


测试

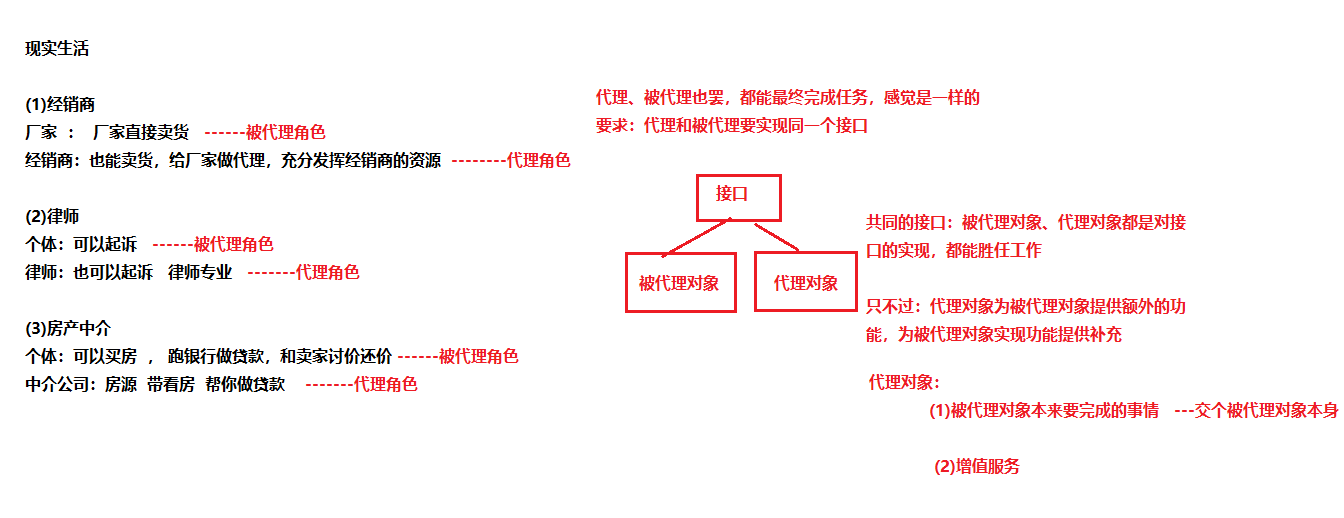




分析：



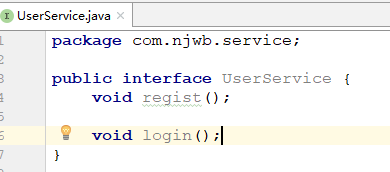
# 7.代理设计模式

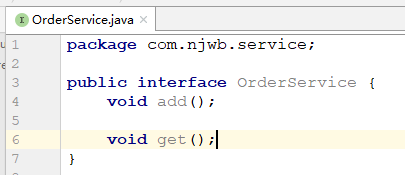


案例：某系统含有多个功能，比如UserService、OrderService。。。。。。。。。

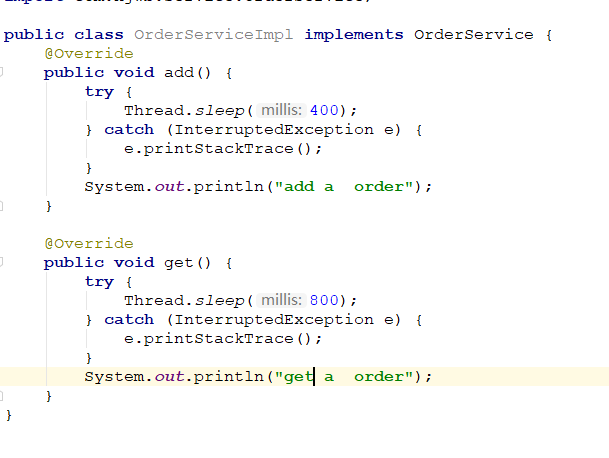
开发好了以后，版本上线了，做：系统监控，监控功能接口的执行效率。

模拟一下现有的功能接口









增加监控程序----->违背开闭原则





足够的耐心、细心 相信能顺利的完成工作

测试：



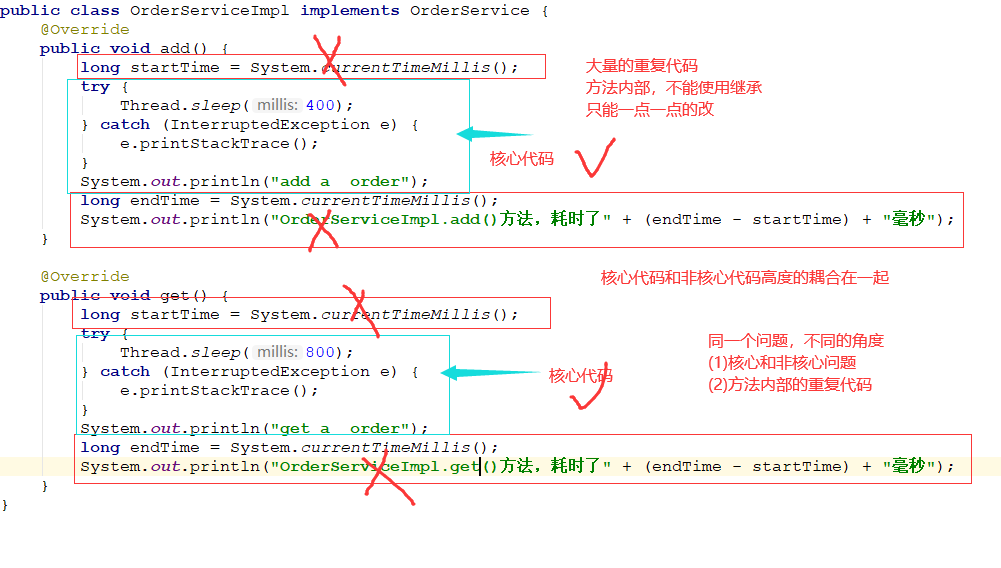
程序运行了一段时间，稳定了 ----->撤销监控

删掉所有的业务类中的监控代码 ----->违背了开闭原则 ---->愿意干

剧本又变了，过了一段时间，好像系统又慢了----->增加监控

如此反复----------->程序员心态要炸了。

分析炸的根本原因：



非核心的功能代码交给代理完成

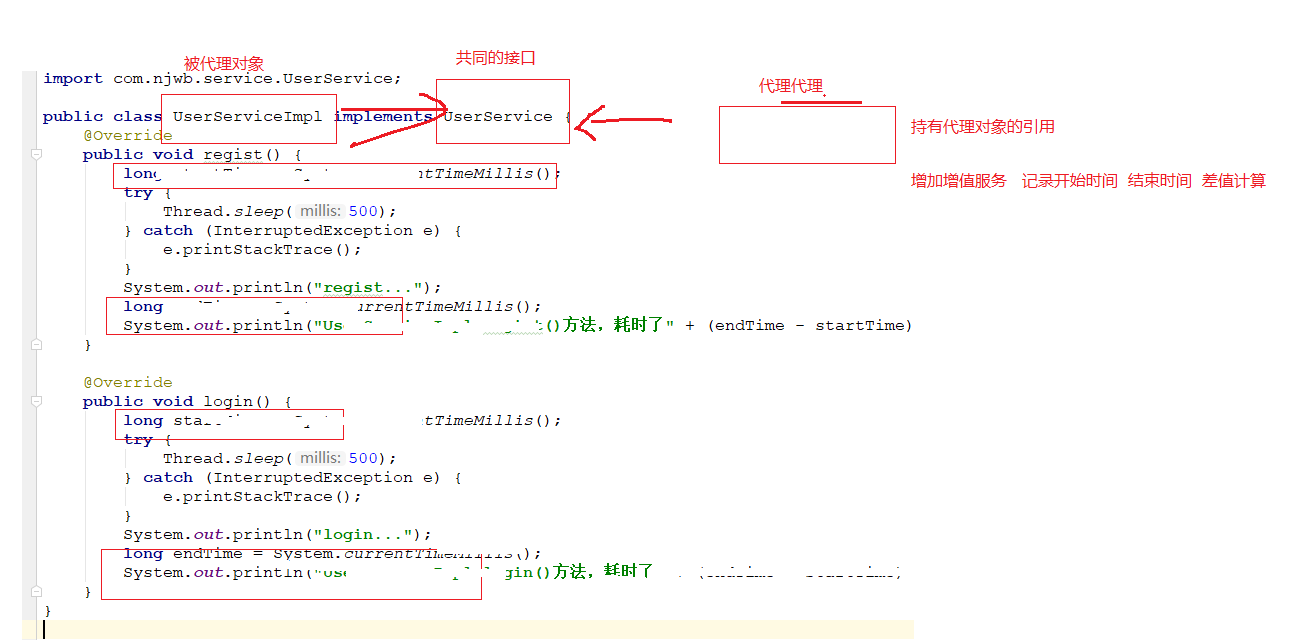
代理的关键点：

(1)谁是代理对象，谁是被代理对象

(2)谁是代理对象和被代理对象共同的接口

(3)谁持有谁的引用，谁增加增值服务

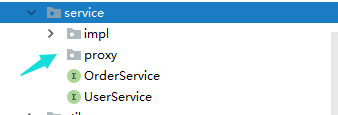
以UserServiceImpl为例



(1)删除被代理对象UserServiceImpl中的监控代码，只保留核心功能

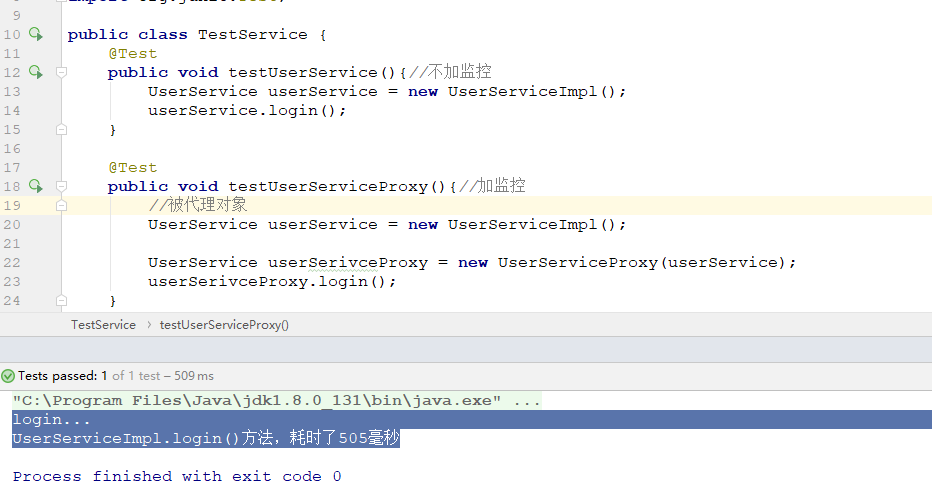


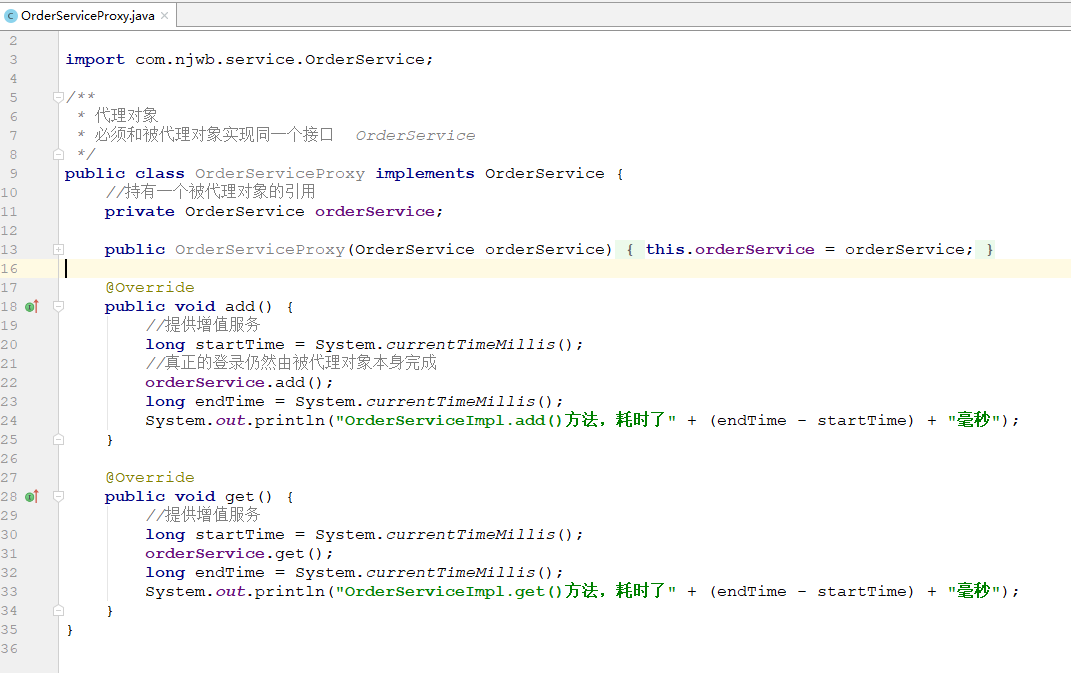
(2)新增代理对象





(3)运行





采用代理解决了：被代理对象中的重复代码/非核心代码与核心代码彻底的解耦合。

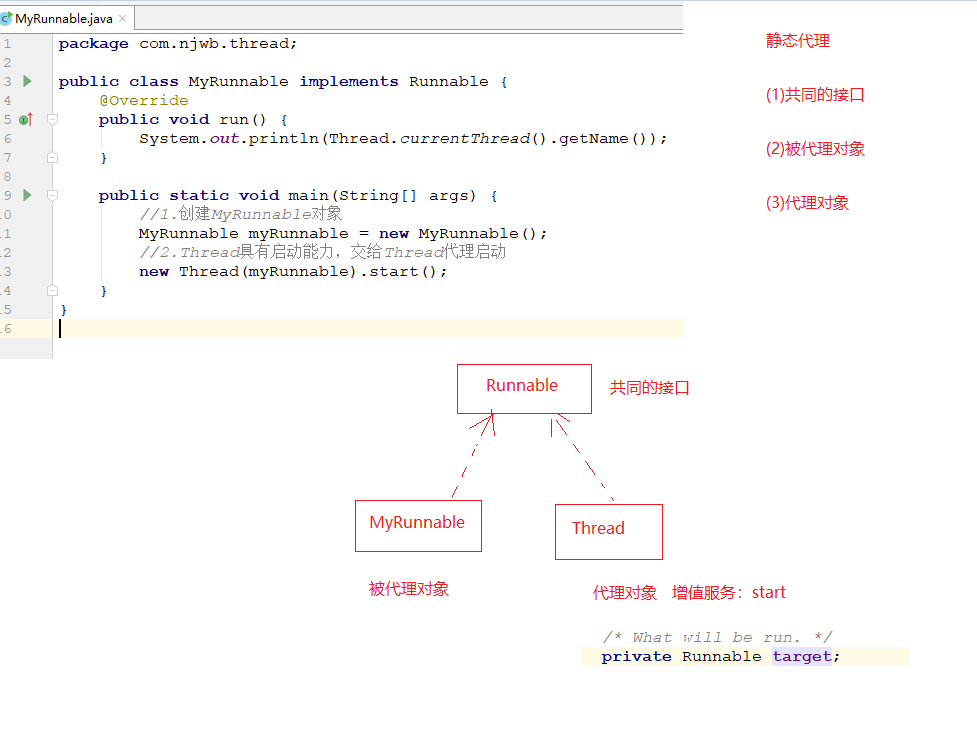
带来了新的问题：

(1)新增代理类，导致类的泛滥

(2)代理类之间又出现了重复代码

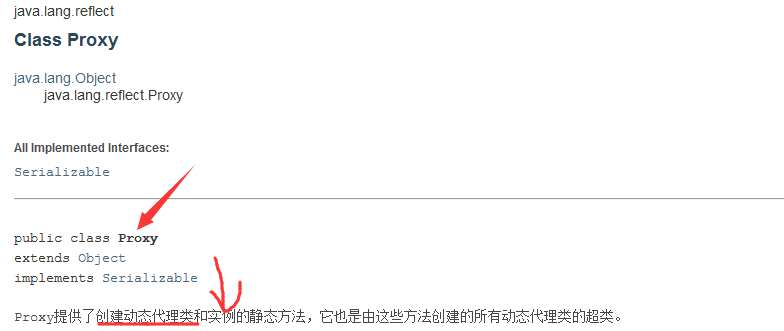
以新增代理类为代价的称之为：静态代理

在多线程中：

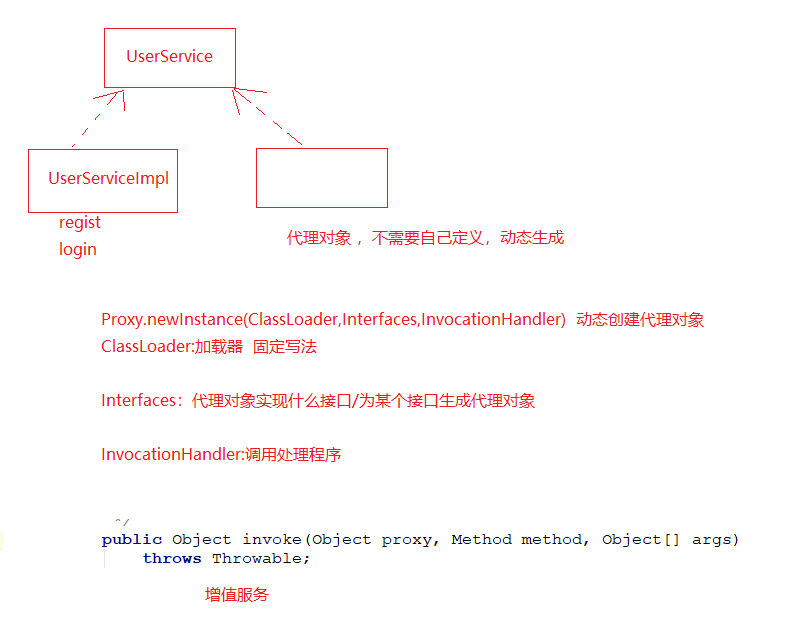


代理类对象通过动态生成。

JDK反射生成动态代理

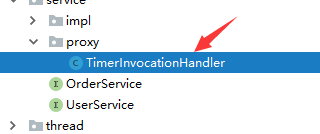








测试

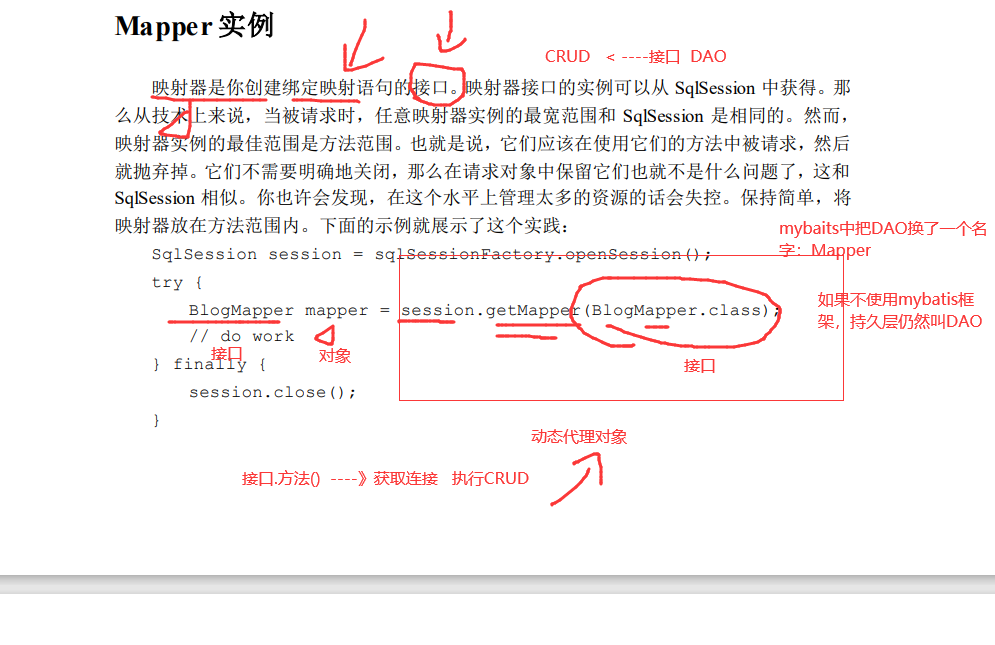




# 8.Mapper代理模式

## 8.1 基本实现

重点！！！！



遵循一些规范：MyBatis内部要求

规范一、映射文件必须和接口在同一路径下，且映射文件的名称和接口名一致。

规范二、映射文件中的namespace必须是接口的全路径

规范三、SQL标签中的id名称必须和接口中的方法名一模一样

规范四、SQL标签中的输入参数类型必须和接口中方法的形参类型一模一样

规范五、SQL标签中的返回值类型必须和接口中方法的返回值类型一模一样

创建maven工程mybatis02

前期的准备：

(1)工程

(2)pom.xml文件

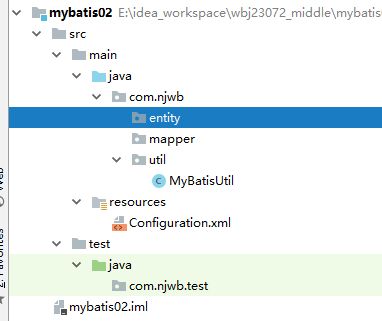
(3)基础的包结构

实体包 entity

工具包 util

持久层包 改为mapper 就是之前的dao

测试包





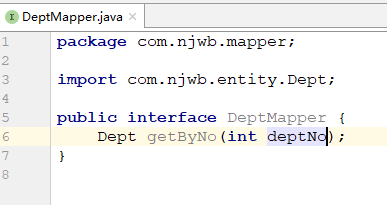
需求：

根据部门编号查询部门信息

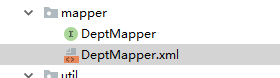
(1)实体类Dept

(2)DeptMapper接口(相当于原来的DeptDao)

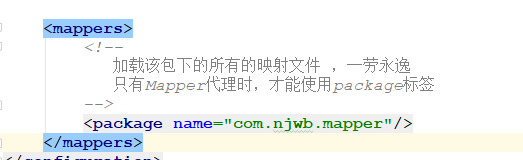
定义一个接口方法 ，用于查询部门信息



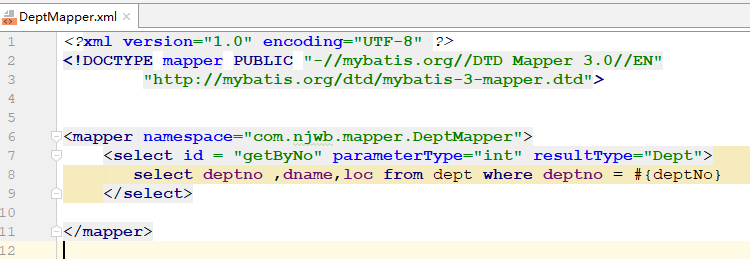
(3)定义DetpMapper.xml文件



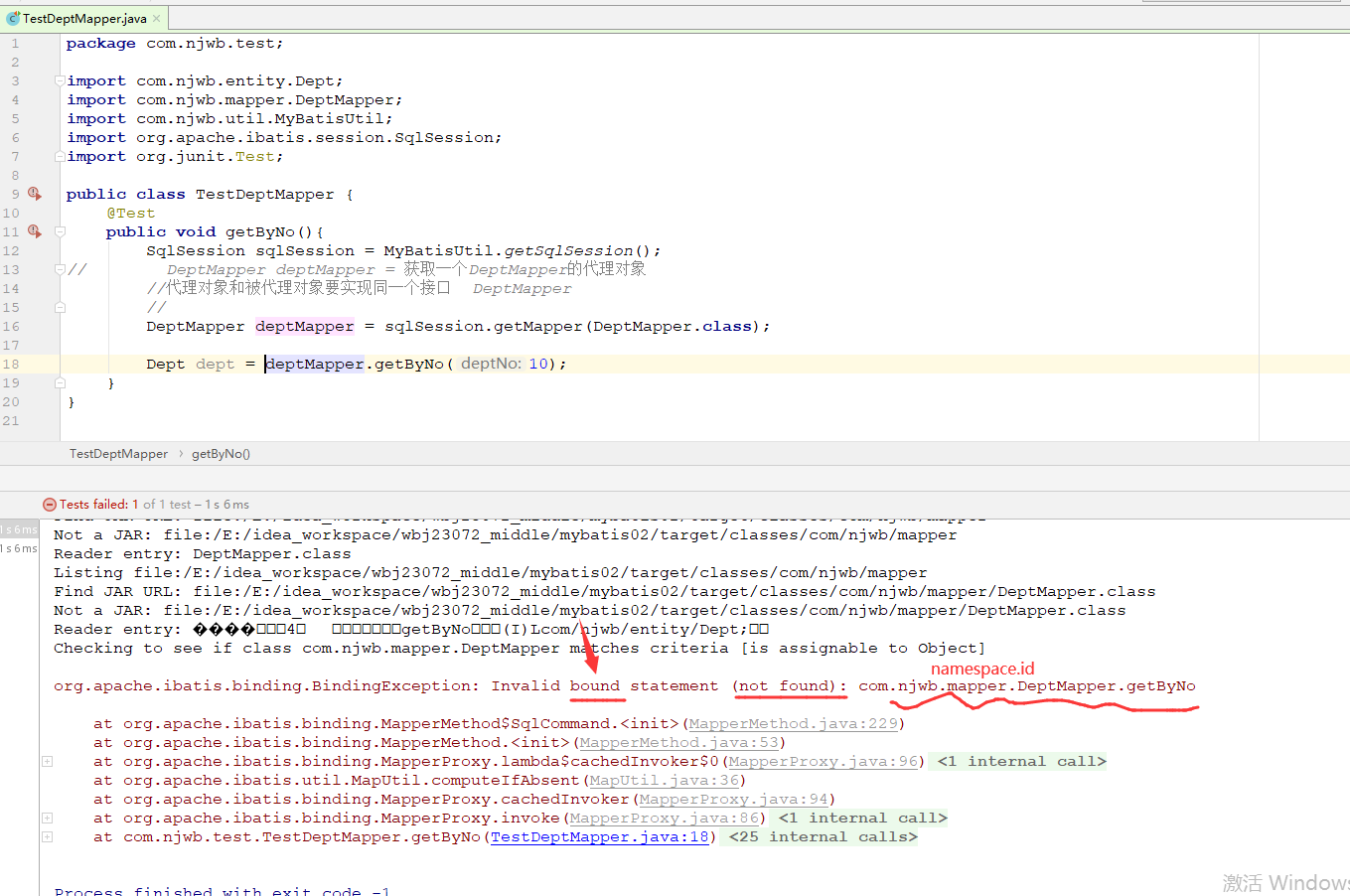
(4)在主配置文件中注册映射文件



(5)在映射文件中定义SQL标签



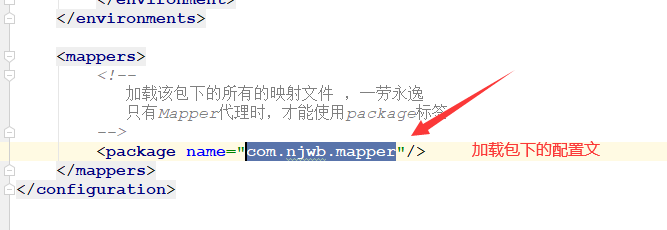
(6)测试

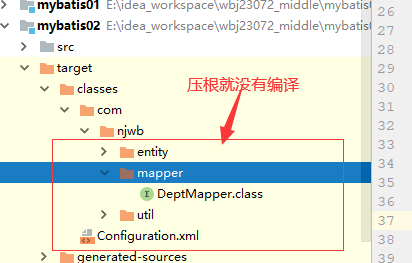


找不到的可能性：

a、名称错了

b、没有被加载





说明：maven工程默认会编译src/main/resources目录下的配置文件

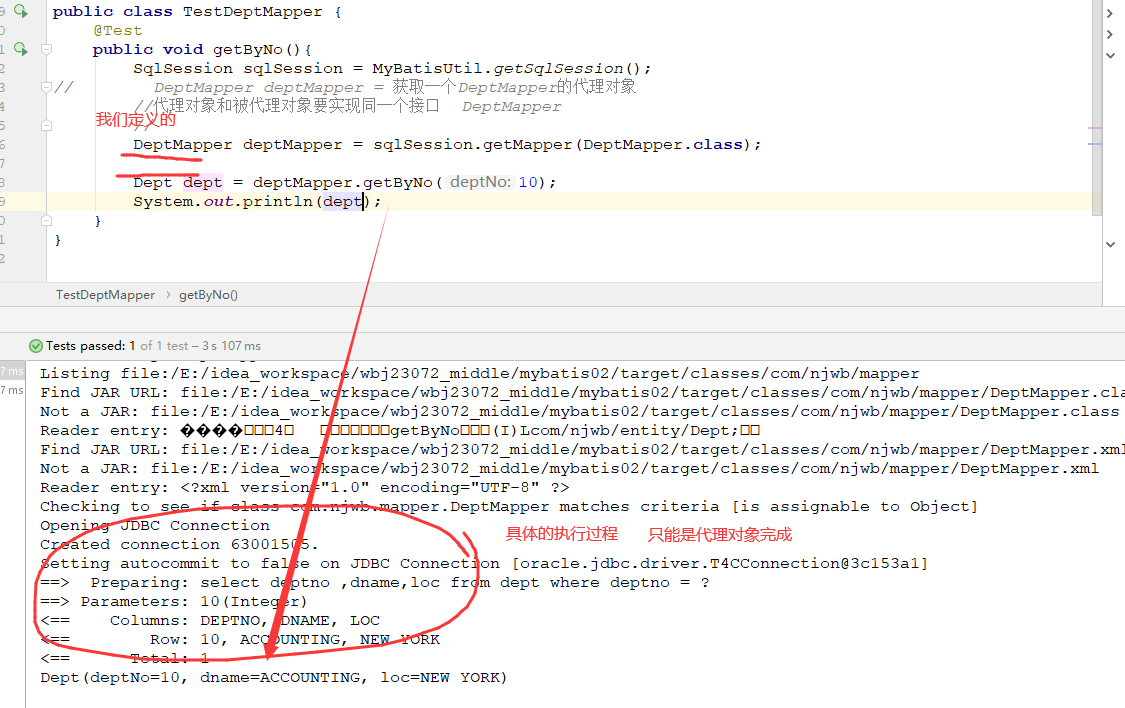
也就是说src/main/java目录下的配置文件不认识

---->告知maven,你在编译的时候，一并把src/main/java下的配置文件也编译一遍

在pom.xml中配置

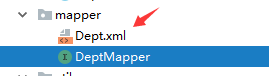


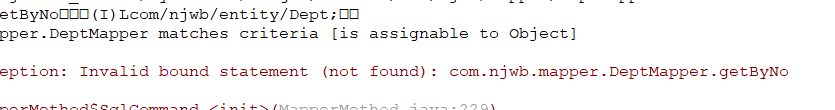
再测试



## 8.2 常见的问题

(1)映射文件不符合要求

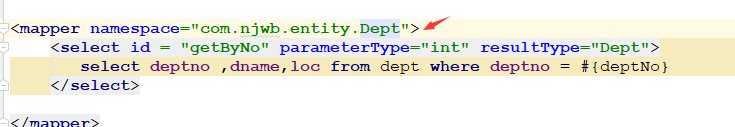




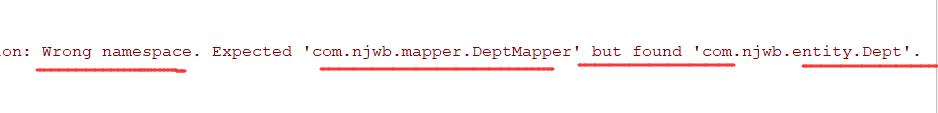
分析：Dept.xml没有被加载，先决于是否存在同名接口。

mybatis:先按照主配置文件中的package -----》mapper接口的路径 ---->扫描mapper包下的所有接口，根据接口的名称构造出同名的xml，进而加载XML文件。

(2)namespace不正确



解析的过程就报错了

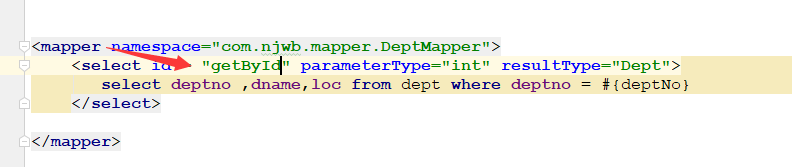


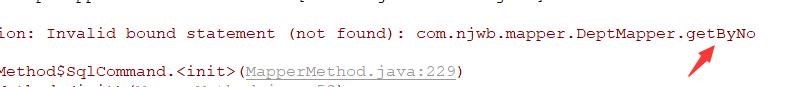
mybatis:

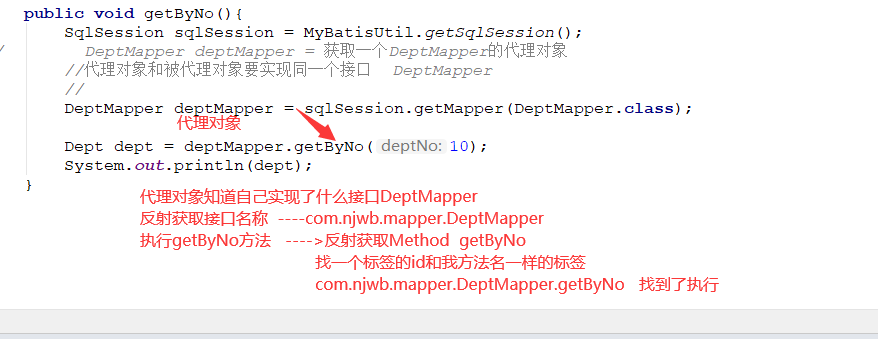
先加载接口DeptMapper ----》Class -----》接口的名称 ---->com.njwb.mapper.DeptMapper--->

就应该是namespace ----》解析XML，比对namespace是否正确

(3)标签ID名称和接口的方法名称不一致







## 8.3 几点疑问

1、mybatis的代理能够帮助我们执行CURD操作，也就是说不需要我们自己调用

sqlSession.insert() ----><insert> insert语句 --->insert方法

sqlSession.delete() ---->< delete > delete语句 ---> delete方法

sqlSession.update()----><update > update语句 ---> update方法

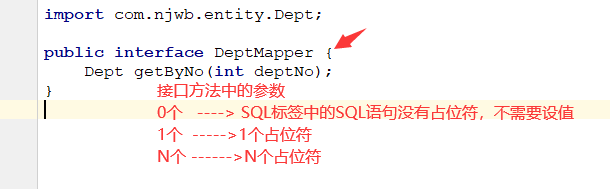
sqlSession.selectOne()----><select> select语句 ---> selectOne方法

根据接口方法的返回值类型，单个和多个

sqlSession.selectList()----><select> select语句 ---> selectList方法

mybatis是如何知道要调用相应的方法？

2、接口中方法参数最多有一个



SQL标签中的输入参数类型必须和接口中方法的参数类型一模一样。

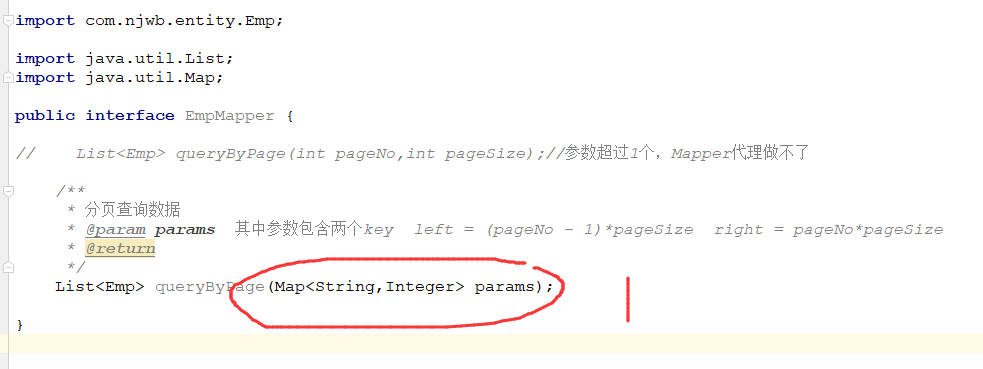
parameterType只能接收一个参数 ----》Mapper接口中的方法的参数最多不能超过一个---》

改造接口中方法的参数 ----->实体？Map?

需求：员工的分页查询

Emp

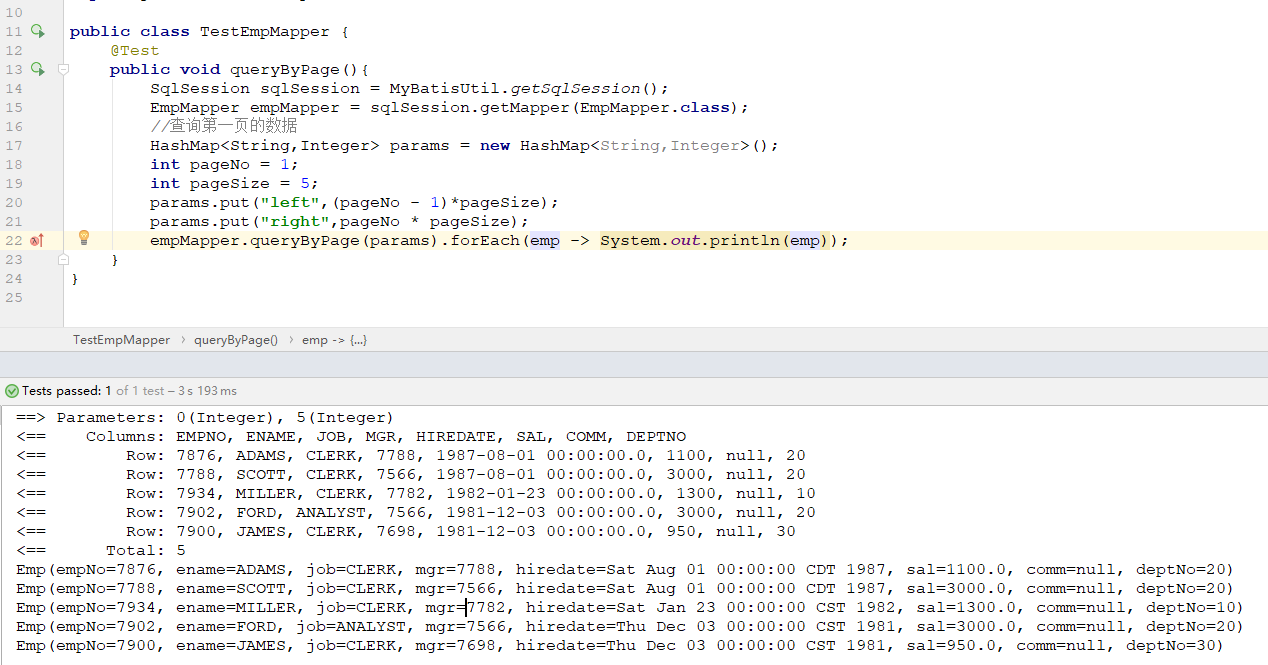
EmpMapper



EmpMapper.xml 定义SQL语句



测试



## 8.4 底层源码鉴赏

详情见视频。

# 9.注解版本

和XML类似，也可以作为程序的配置

常用的注解：

@Insert

@Update

@Delete

@Select

@Results

@Result

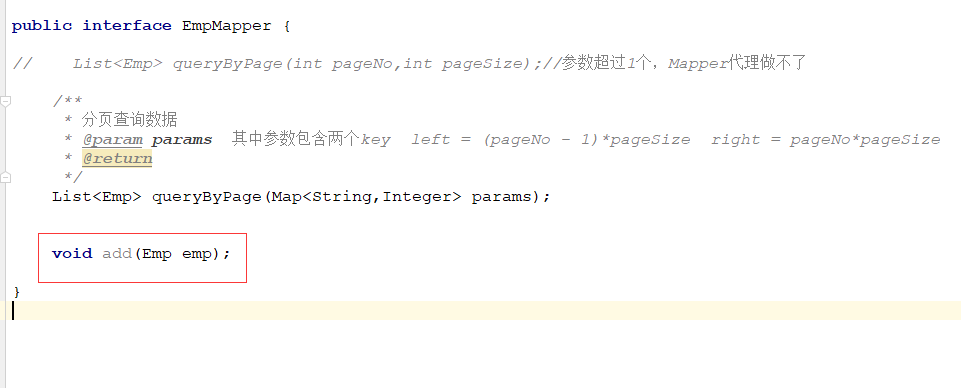
@ResultMap

等

员工的CRUD操作

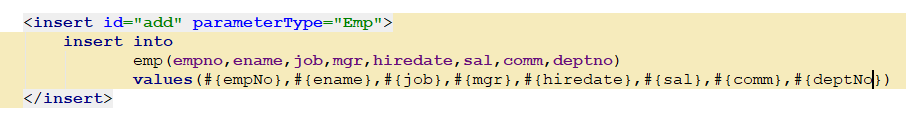
Emp实体

EmpMapper接口 定义方法

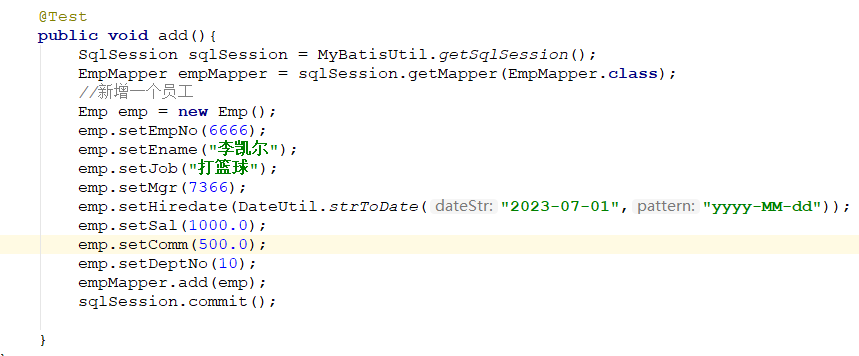


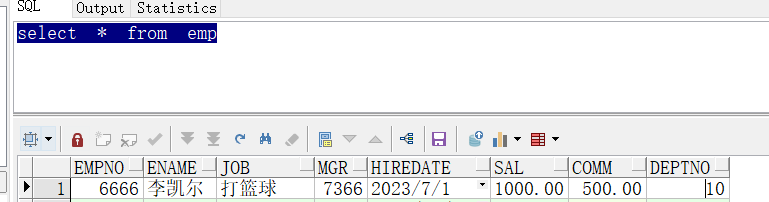
EmpMapper.xml

定义SQL语句



测试



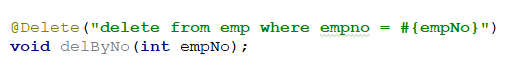


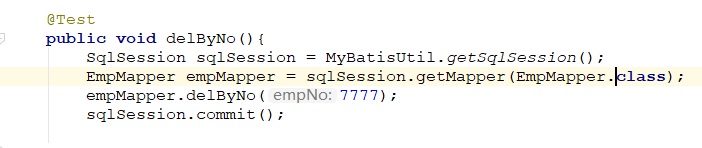
思考：

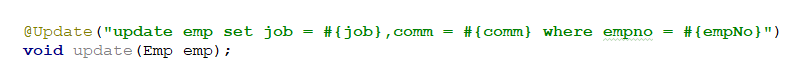
对于比较简单的SQL语句，没有必要再写SQL标签，使用注解来简化。

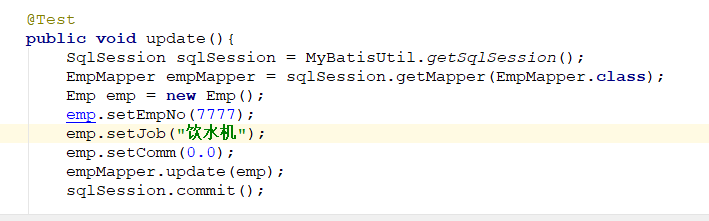
删除insert标签，在Mapper接口采用合适的注解替代



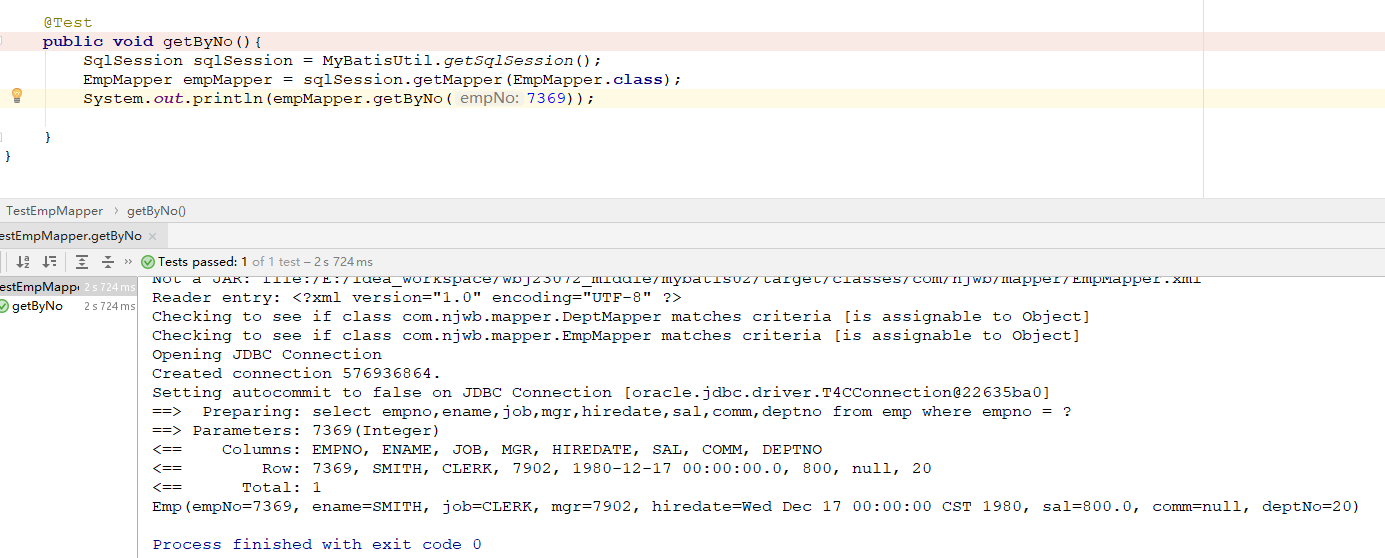








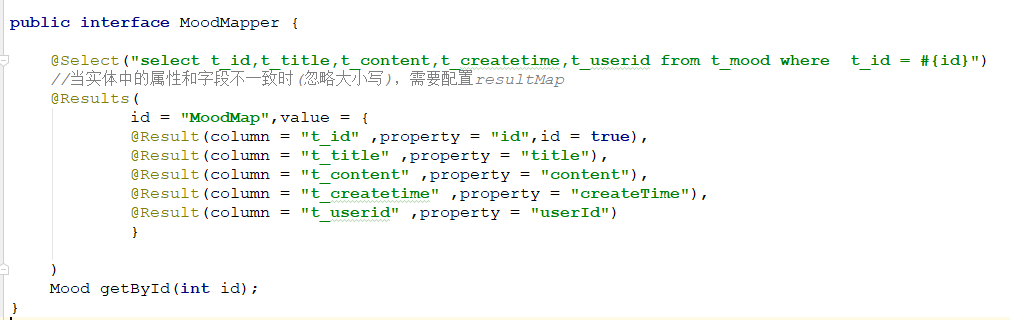




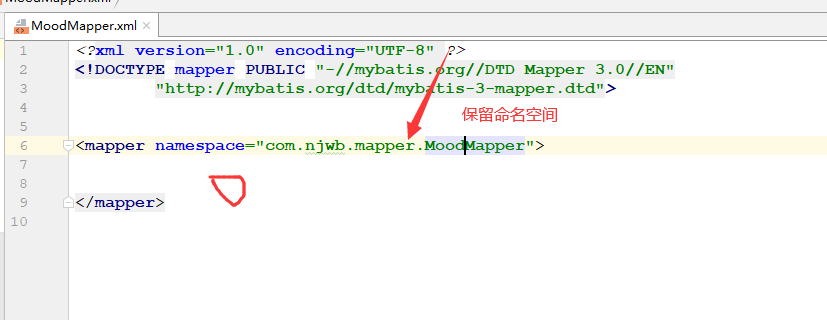
需求：根据心情id查询心情对象

Mood

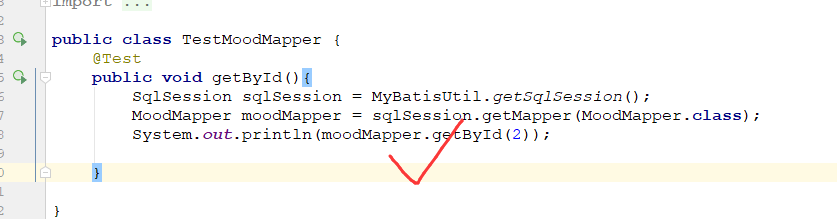
MoodMapper



MoodMapper.xml



测试





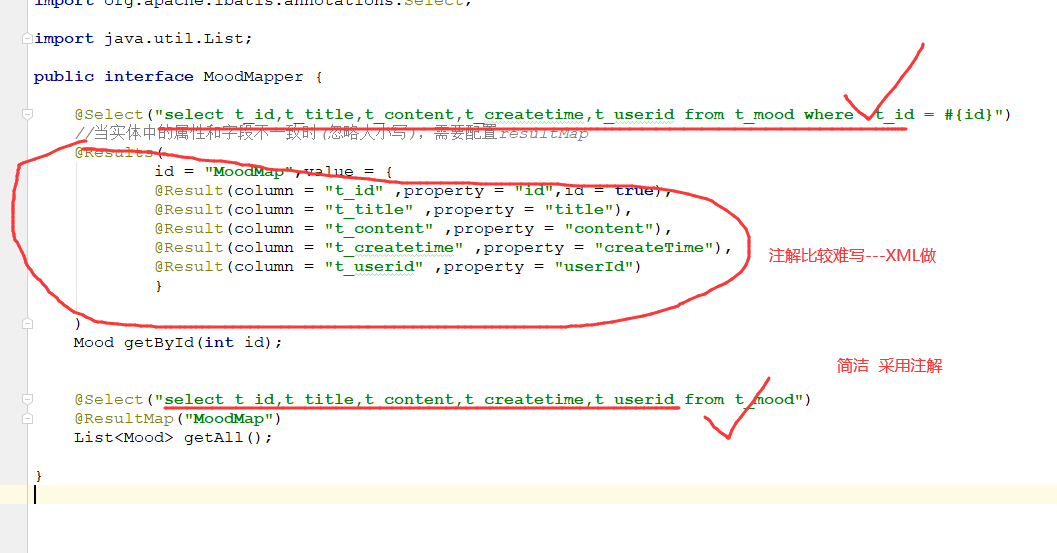
小结：

写映射有两种方式

(1)XML 工整

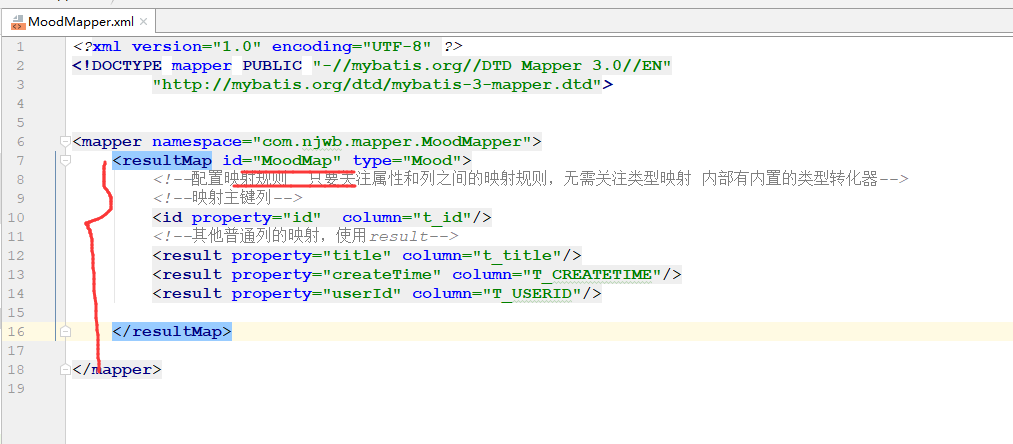
(2)注解 简洁

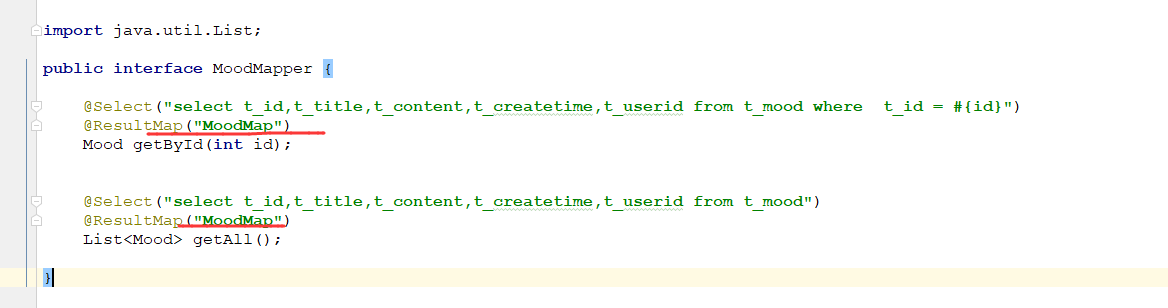
当需要简洁时可以选择注解，如果注解写起来比较麻烦时，请使用XML ----最佳的实践



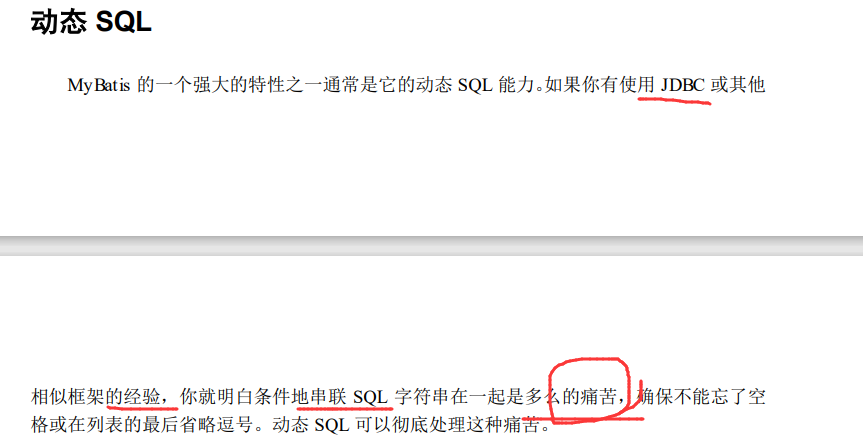
最佳实践：

把实体的映射规则写在XML中





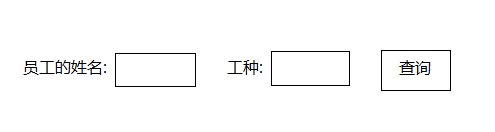
# 10.动态SQL



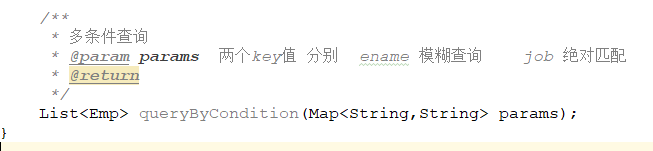
## 10.1 if标签

判断用的

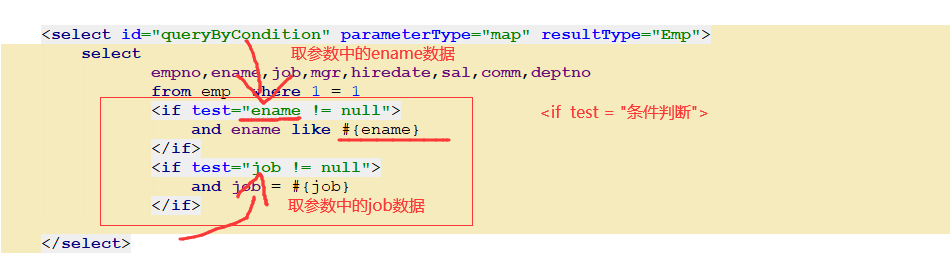
需求：多条件查询



EmpMapper接口中新增的方法



EmpMapper.xml中定义SQL



测试



## 10.2 choose标签

语法:类似于多重if

<choose>

<when test = “条件”> </when>

<when test = “条件”> </when>

<when test = “条件”> </when>

…….

<otherwise></otherwise>

</choose>

要实现if else的效果

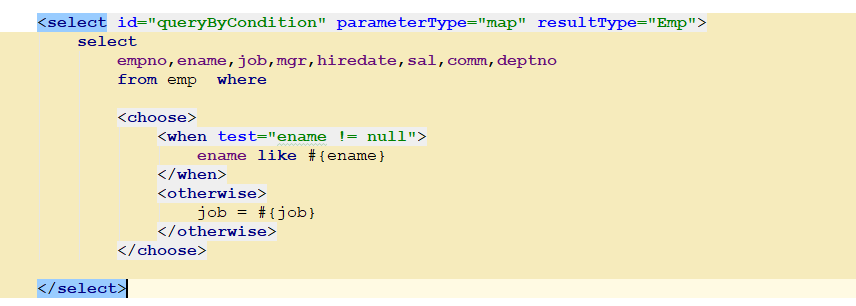
<choose>

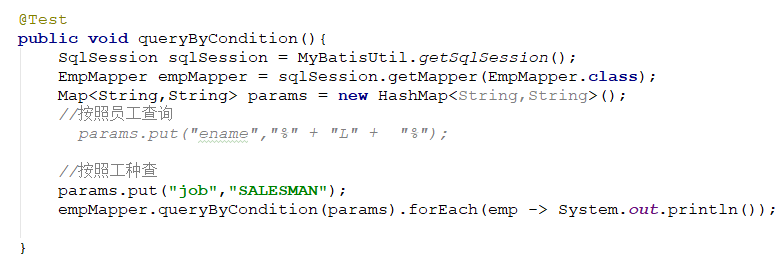
<when test = “条件”> </when>

<otherwise></otherwise>

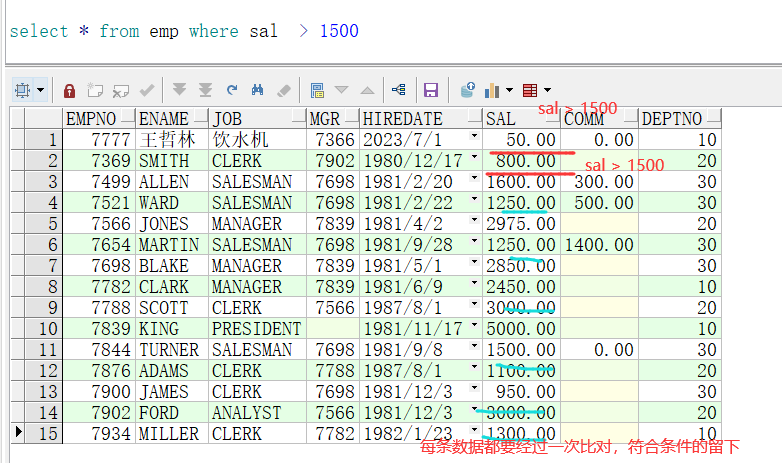
</choose>

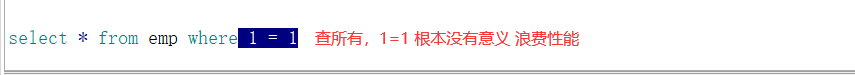
需求：员工的姓名和工种二选一查询



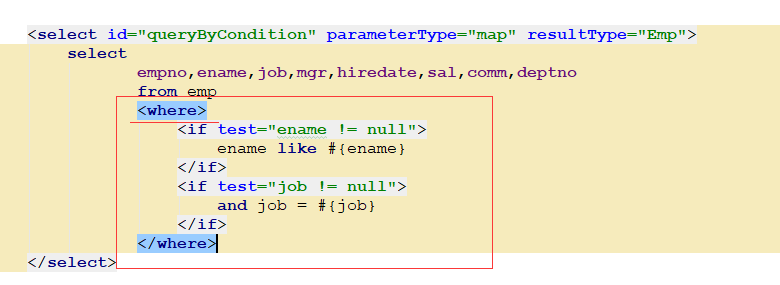


## 10.3 where标签









测试，4种情况

<where>:当where标签中有条件成立时，会动态生成where关键字

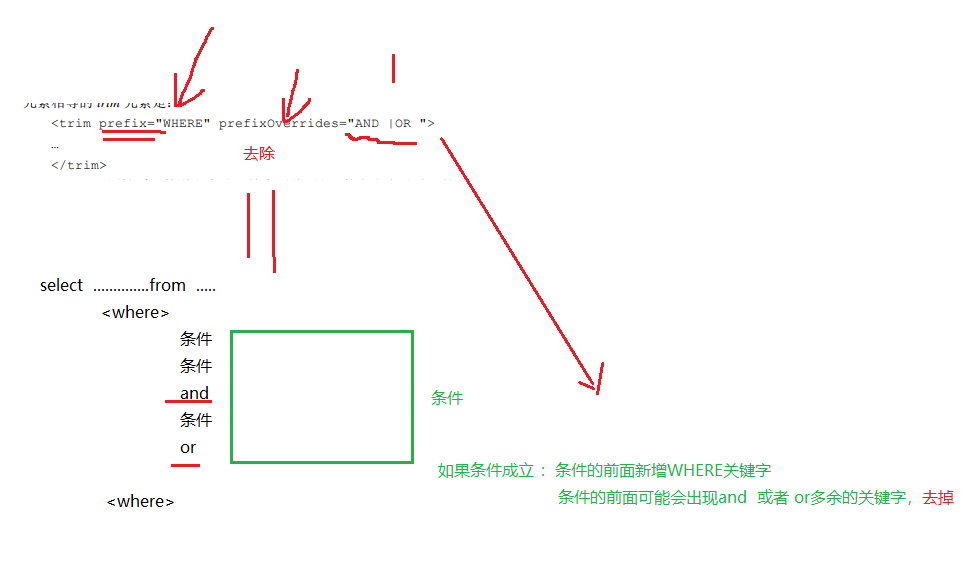
遇到紧接着where关键字后面的or 、and会自动去除

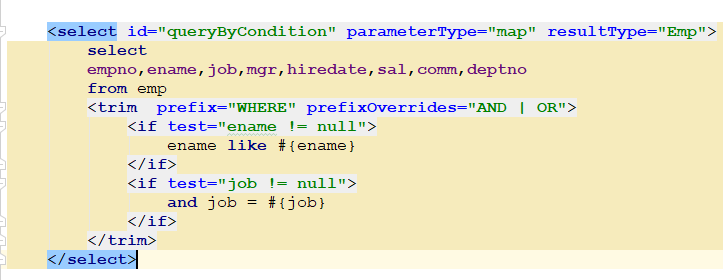
where ~~or~~

where ~~and~~

## 10.4 trim标签

使用trim替换where标签





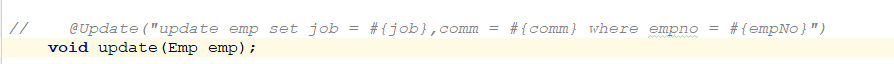
测试

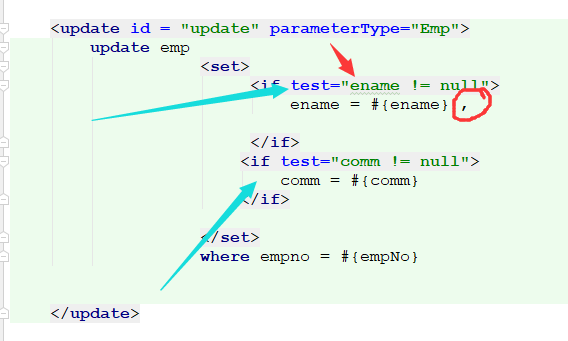
## 10.5 set标签

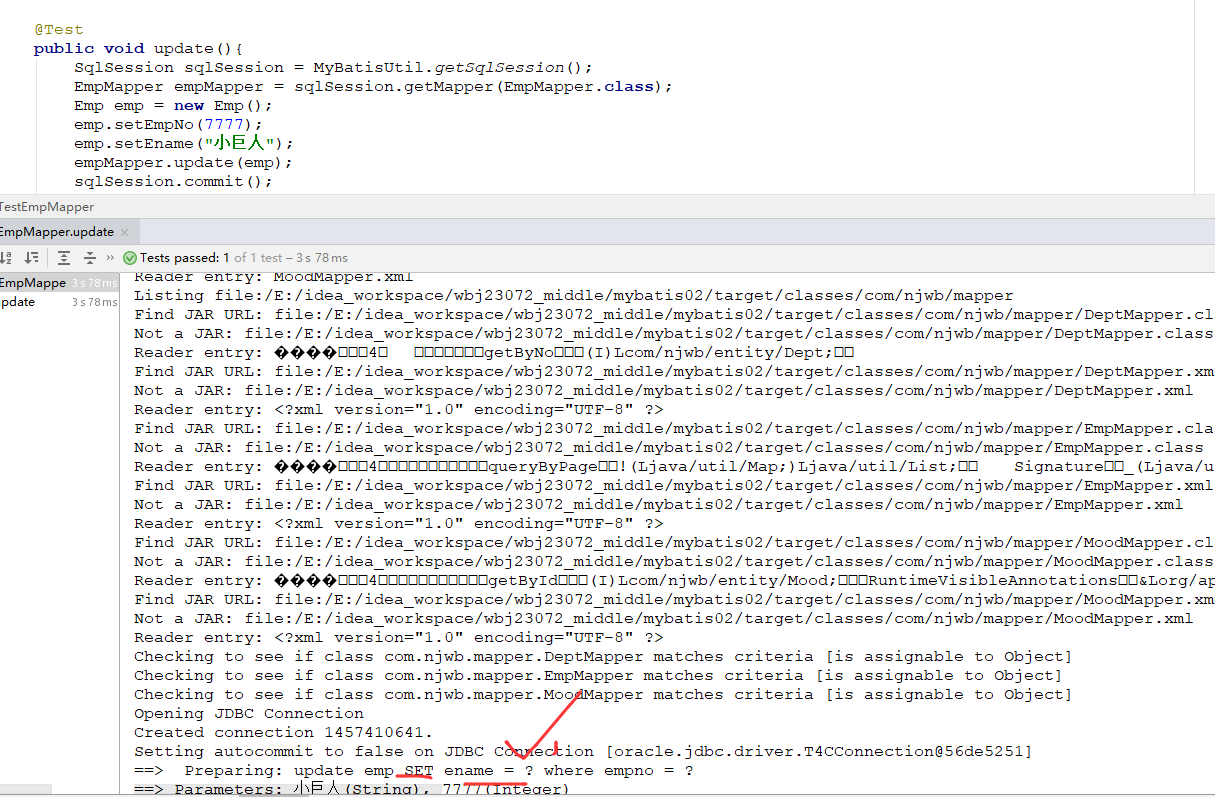
<set> :如果条件成立，条件的前面会动态的增加SET关键字，遇到条件末尾出现多余的逗号会去除。

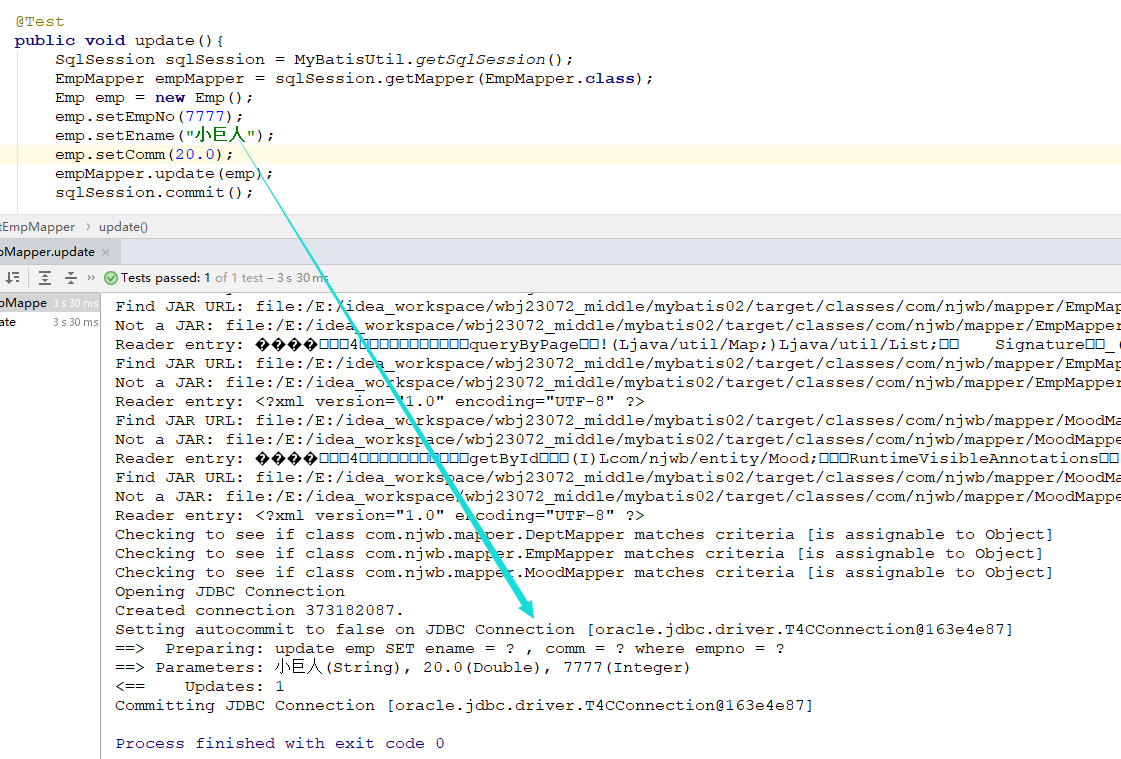
修改员工

如果值为null，就不修改，不为null，则正常修改

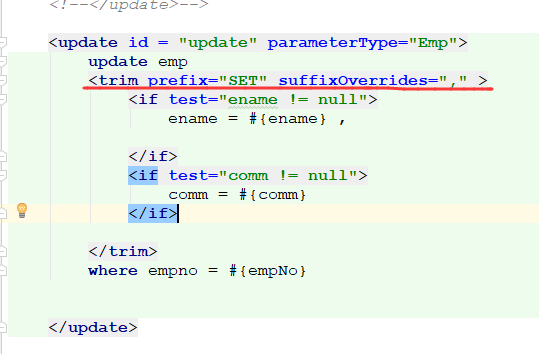




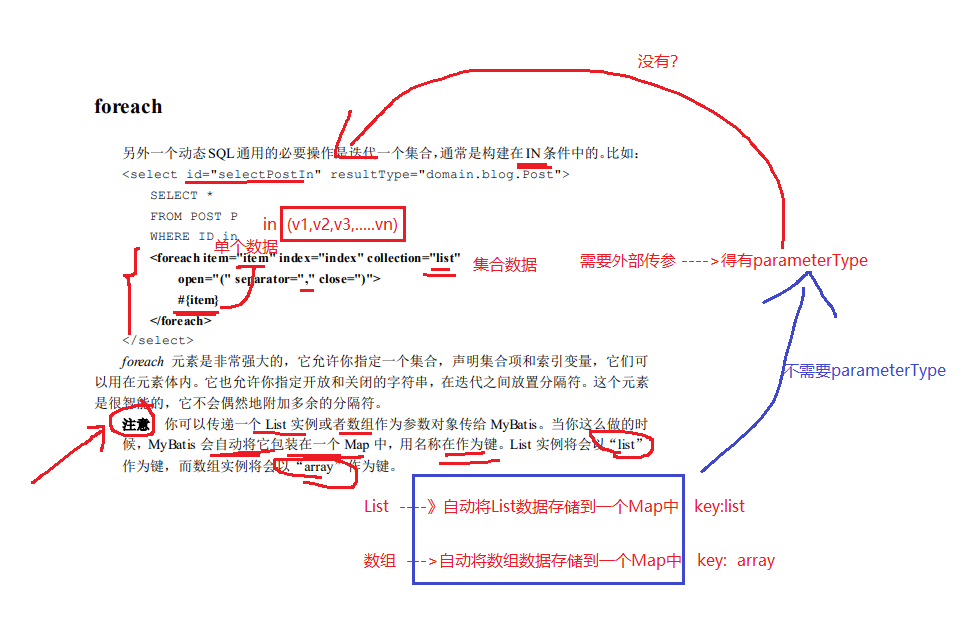




同样可以使用trim标签替换

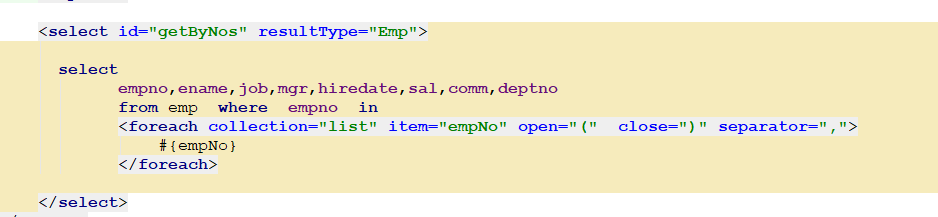


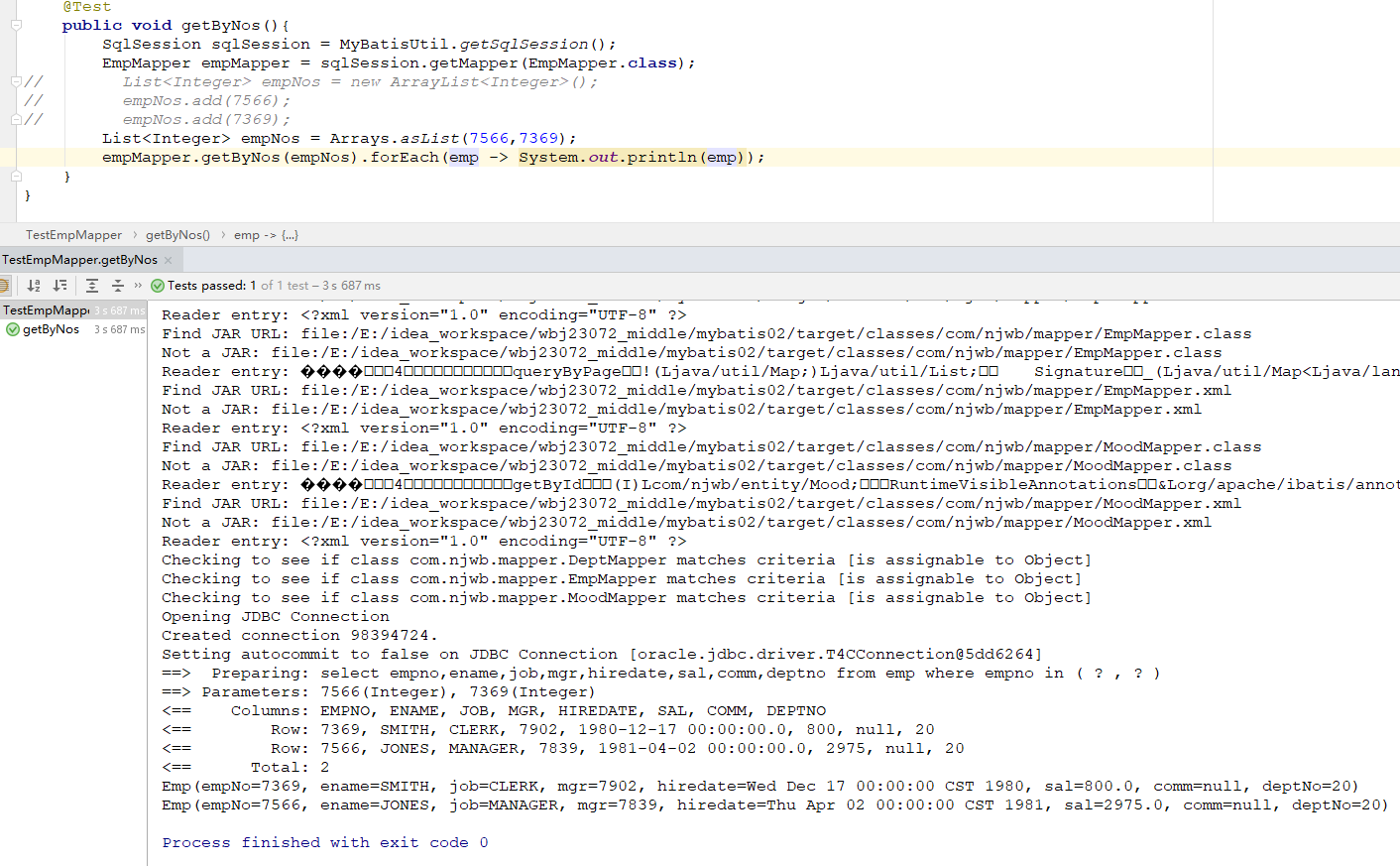
## 10.6 foreach标签



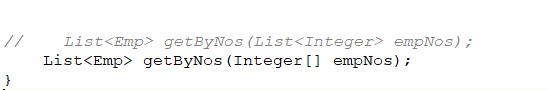
需求：根据多个员工编号查询员工信息

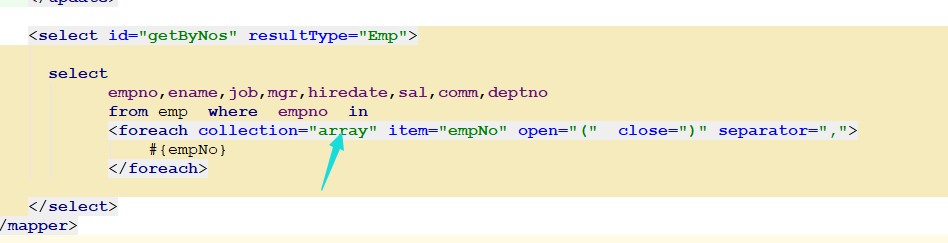




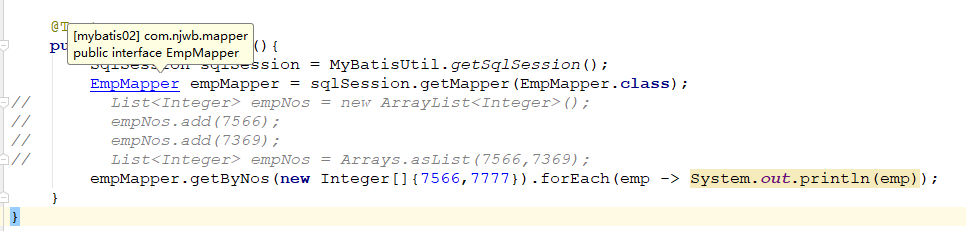


改为数组





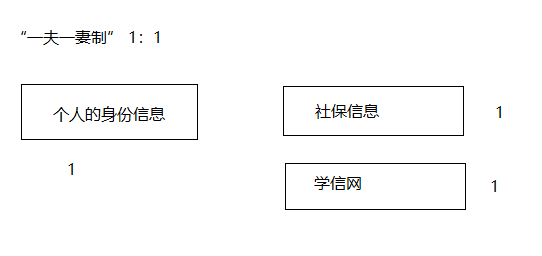
测试



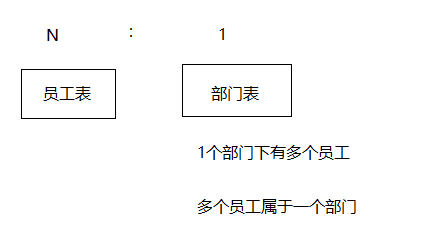
# 11.关联查询

表与表之间的关系

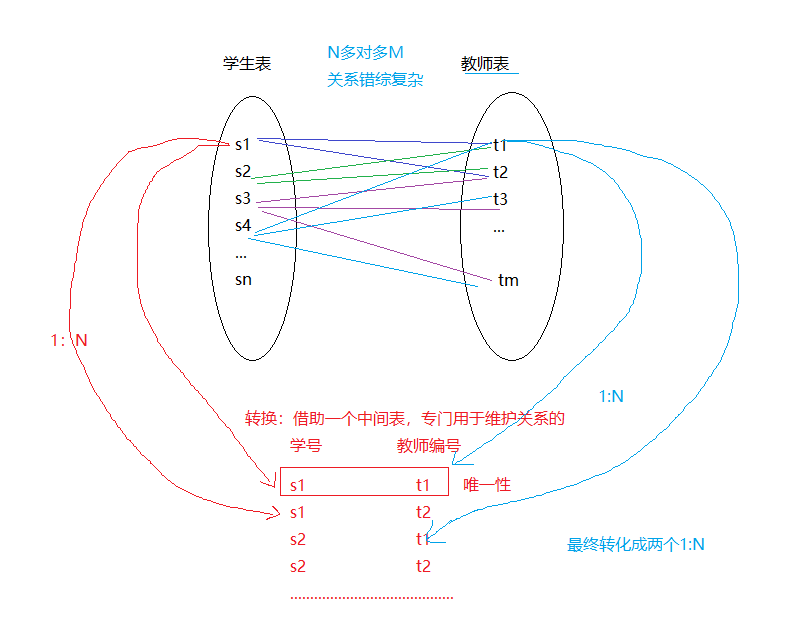
一对一的关系：其实就是1:N的特例 N看成1



一对多和多对一的关系 1:N



多对多的关系 ：N:M



结论：1:N或者N:1是最普遍的

表与表之间有关系 ------------->实体与实体之间有关系 组合 实体与实体之间也有1:N或N:1的关系

## 11.1 多对一的关系

需求：

给定员工编号，查询员工信息及关联的部门信息

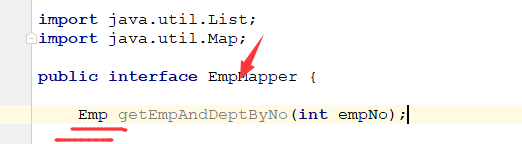
分析：

主查 员工 ----->EmpMapper接口中写方法 ---->返回Emp ---->包含关联信息--->什么关系？多对一的关系 ---->Emp持有一个Dept属性

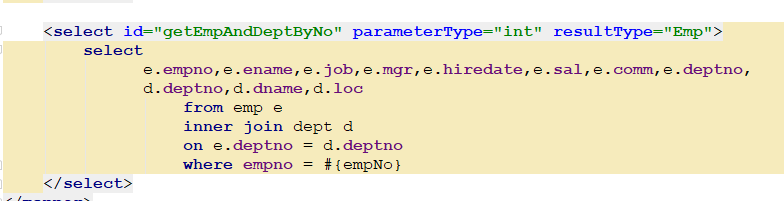
(1)修改实体类Emp



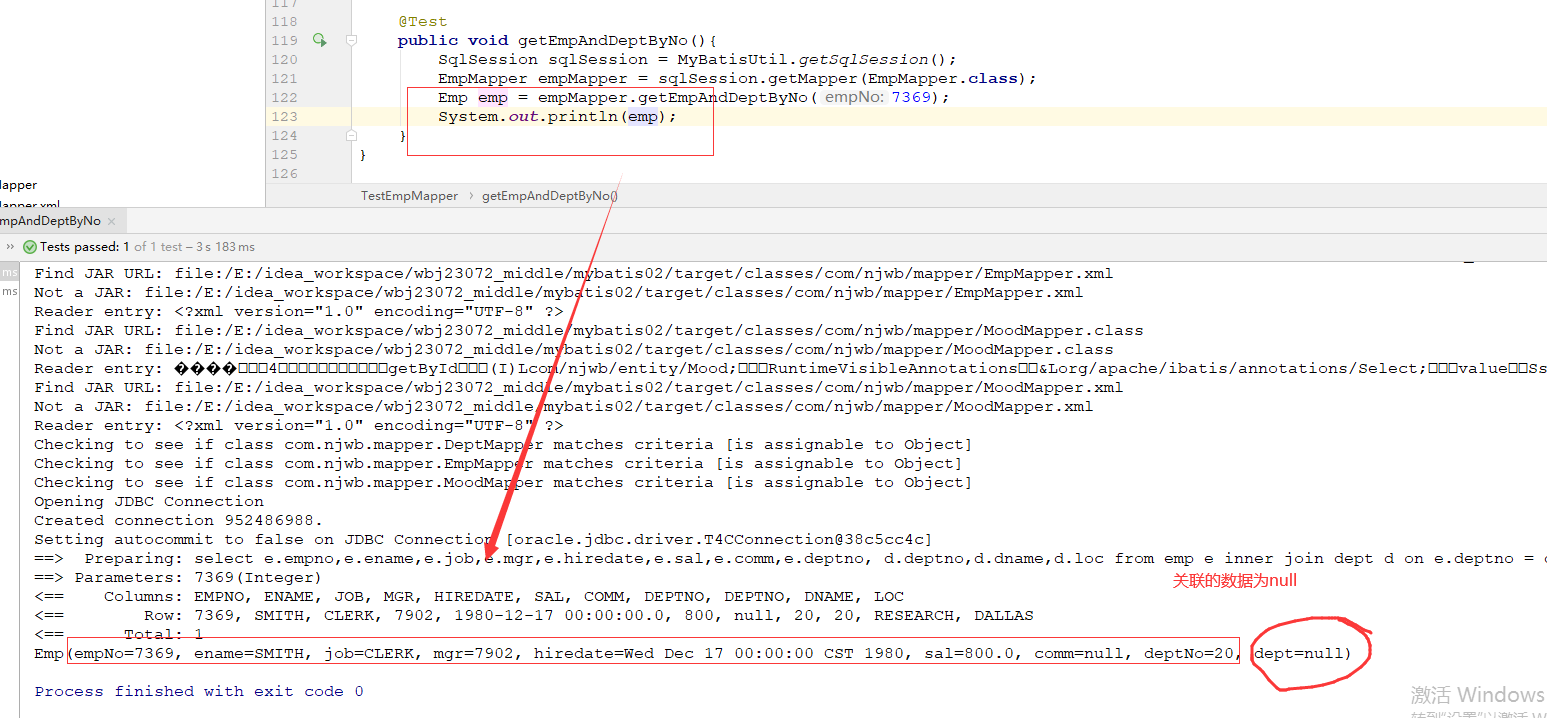
(2)在EmpMapper接口中新增方法



(3)定义SQL语句



(4)测试



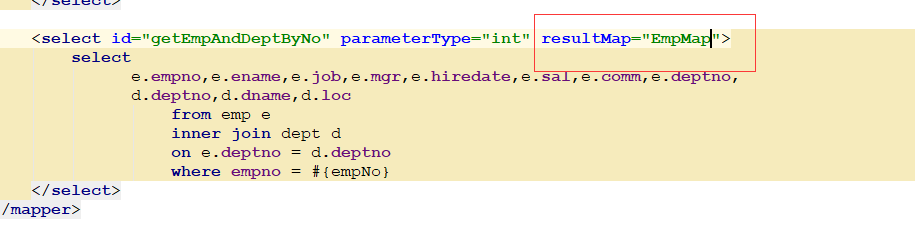
分析：



因此：当出现关联查询时，一律使用resultMap

整改：





测试



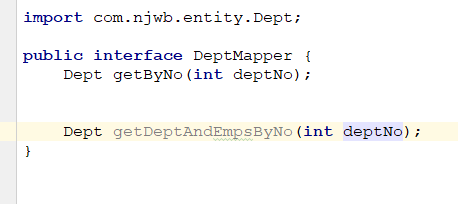
## 11.2 一对多关联

需求：给定部门编号，查询部门信息及管理的员工信息

(1)修改部门实体



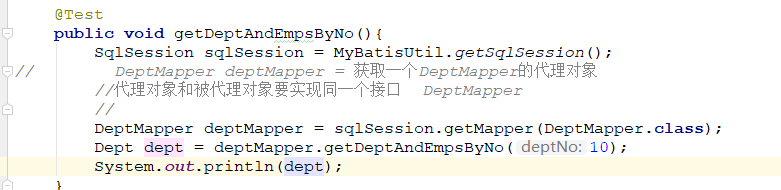
(2)DeptMapper接口中新增方法



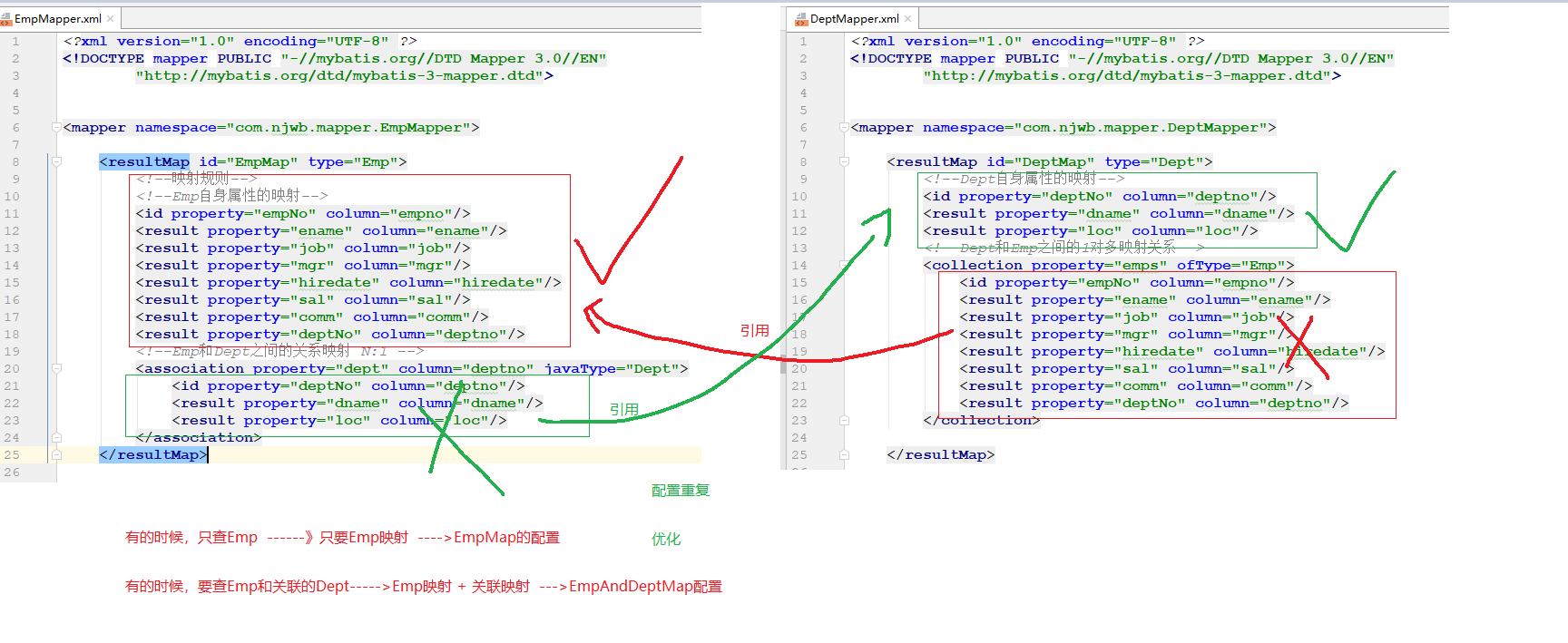
(3)定义SQL

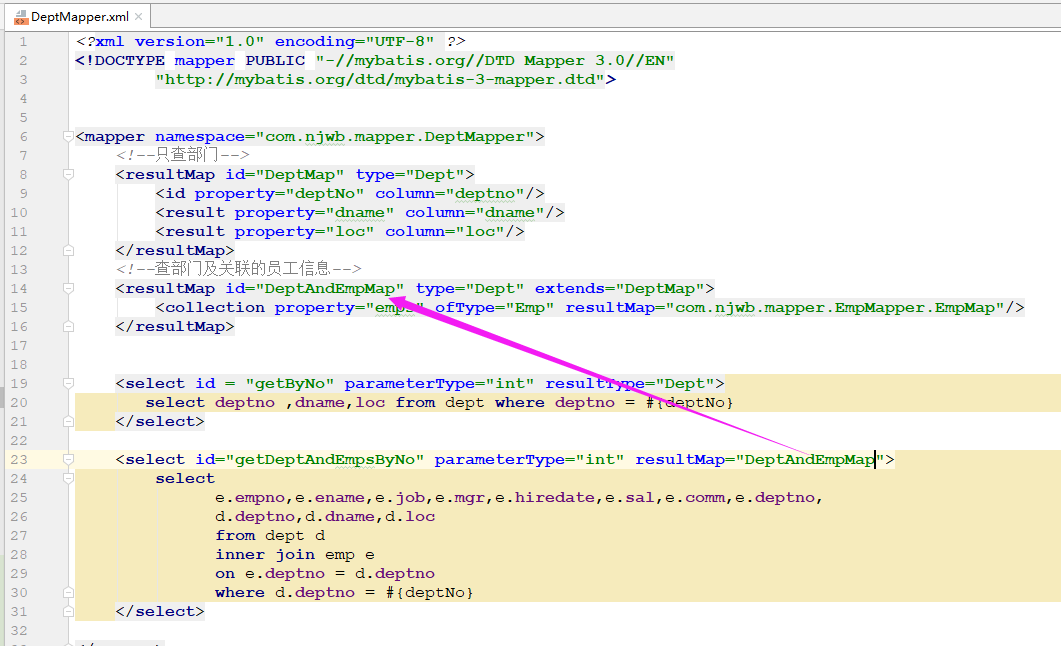


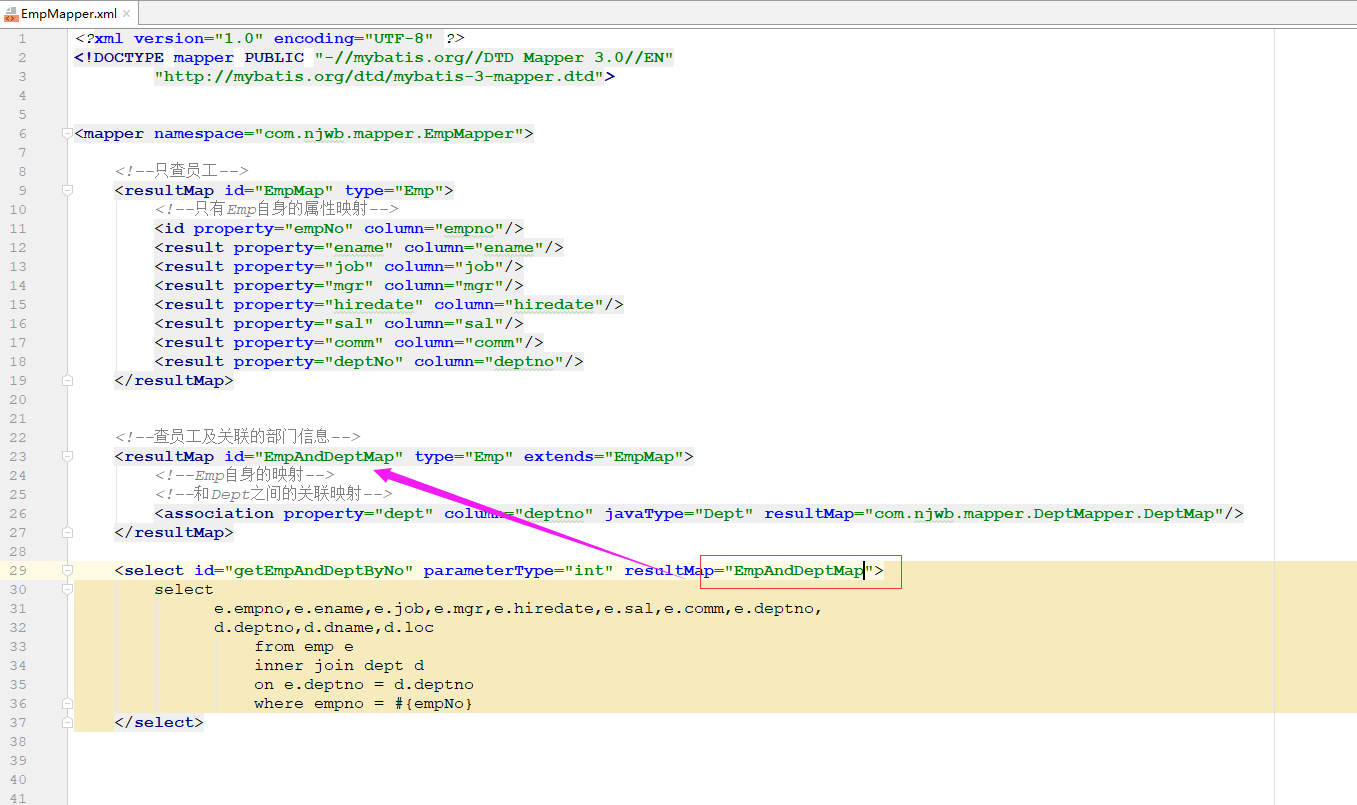
(4)测试



## 11.3 优化配置







如何使用？

结果是否包含关联数据