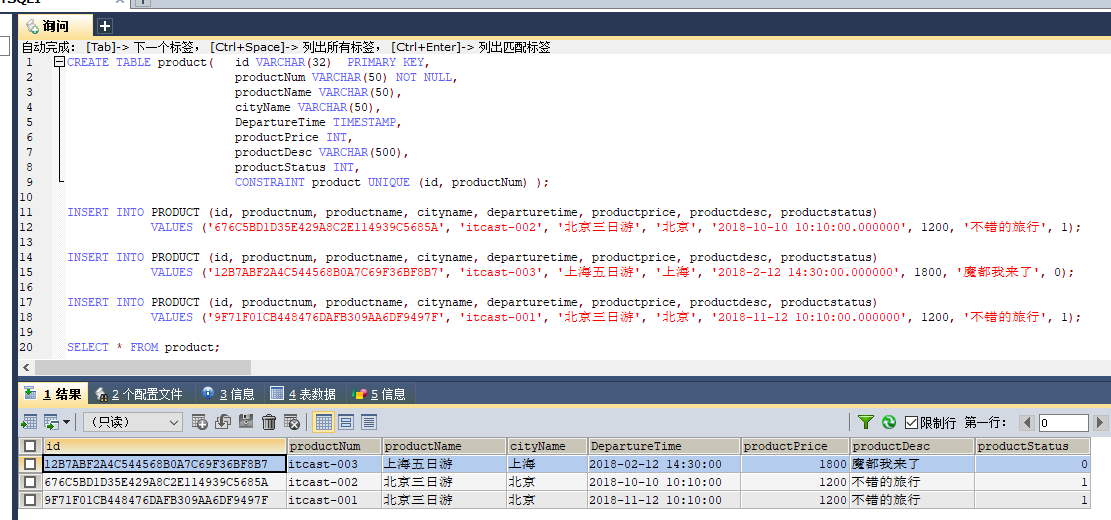
# Day02：课程介绍以及注意事项

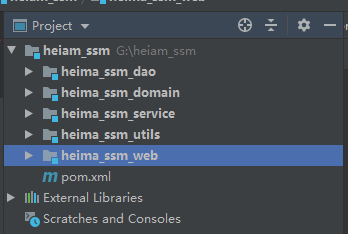
## 01：数据库的创建和表的结构：

课程采用的orcal数据库，我们自己改成mysql。部分知识需要百度！

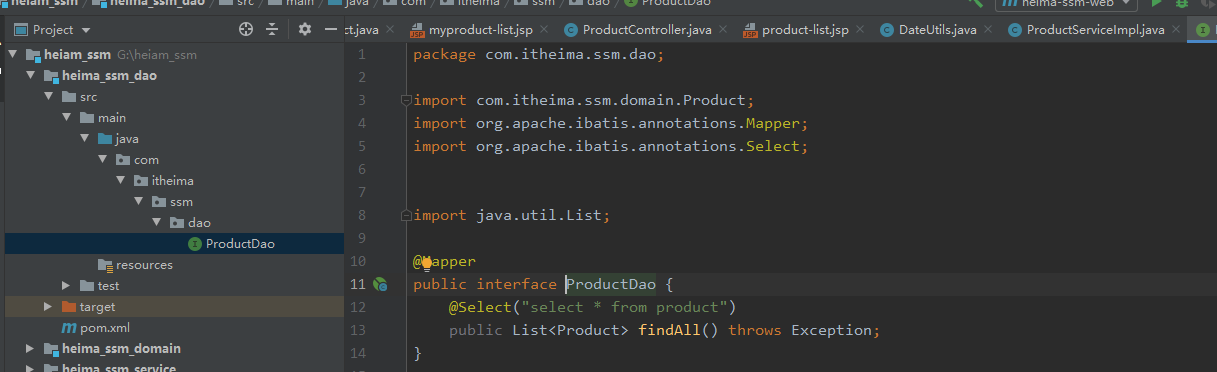


## 02：ssm环境搭建：

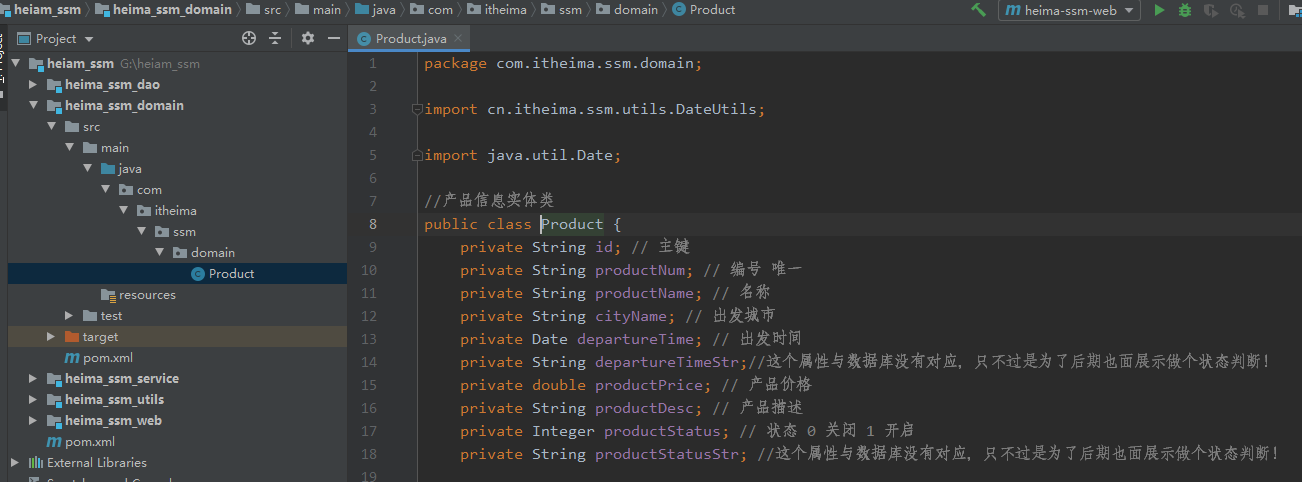
首先创建父项目和他的子模块，在pom文件中锁定版本以及导入jar包。



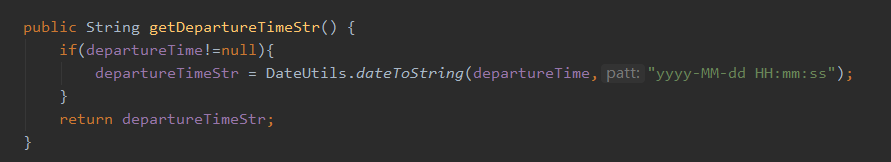
Dao模块中实现持久层的配置，这里采用注解式。而且实际上mapper注解可以不写，后面解释



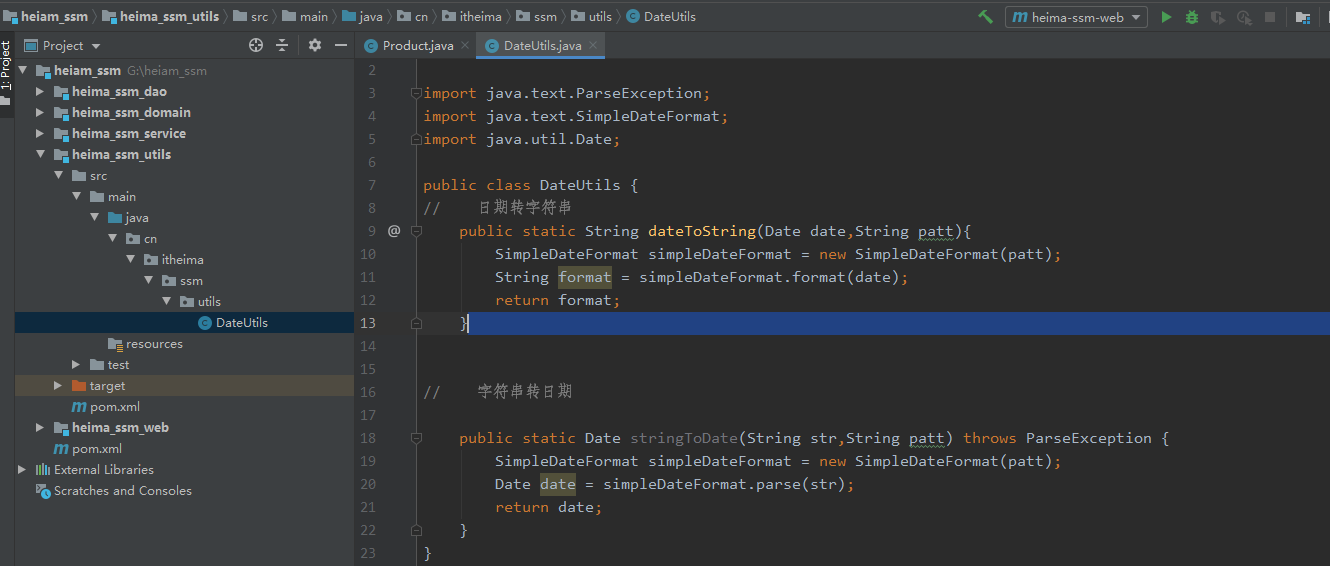
Domain模块中创建封装数据的实体类：注意这里的两个后缀为str的属性，他们并不对应书数据库的任何内容，只是用来识别数据库的一些数据，然后封装合适的数据给页面显示：



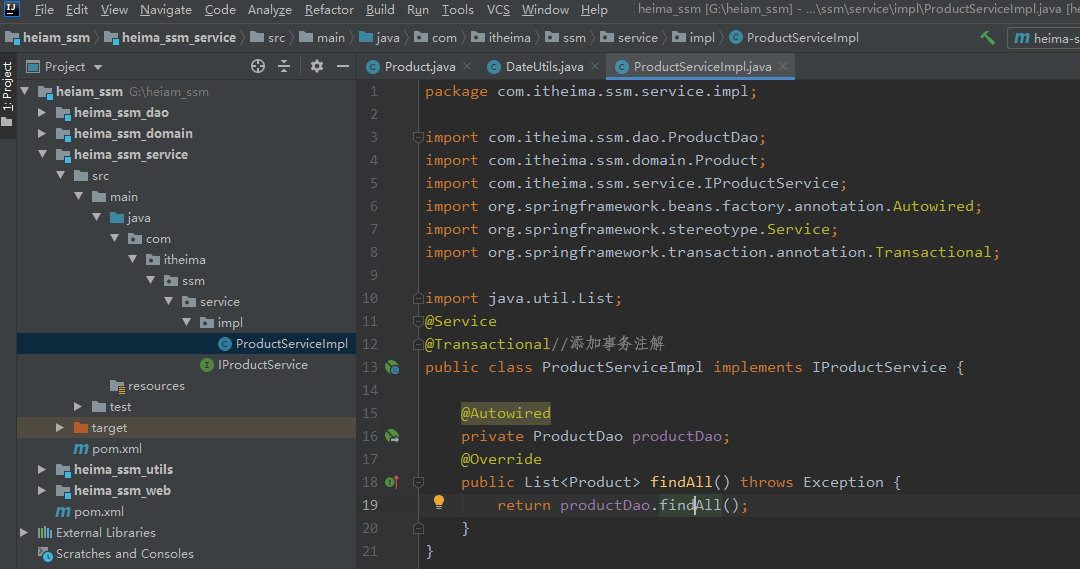
就比如这里的date显示，这里采用了utils模块中自定义的静态方法类！



静态工具包模块中完成我们的data数据的转变。以后也会添加更多的静态工具类



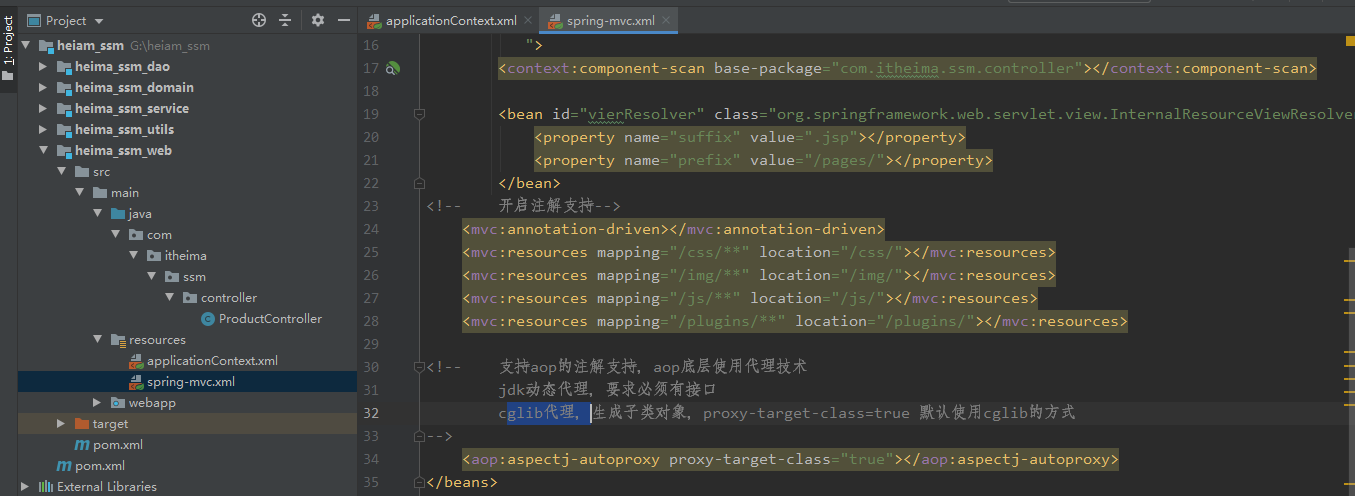
在service模块中完成业务层的配置！注意这里的事务采用的是注解式的配置。



最后配置controller模块，这里的重点（因为老师所有的配置都放到这里了，实际上这么做已经失去了模块化的作用了，其他的项目没办法调用我们的等模块了）。

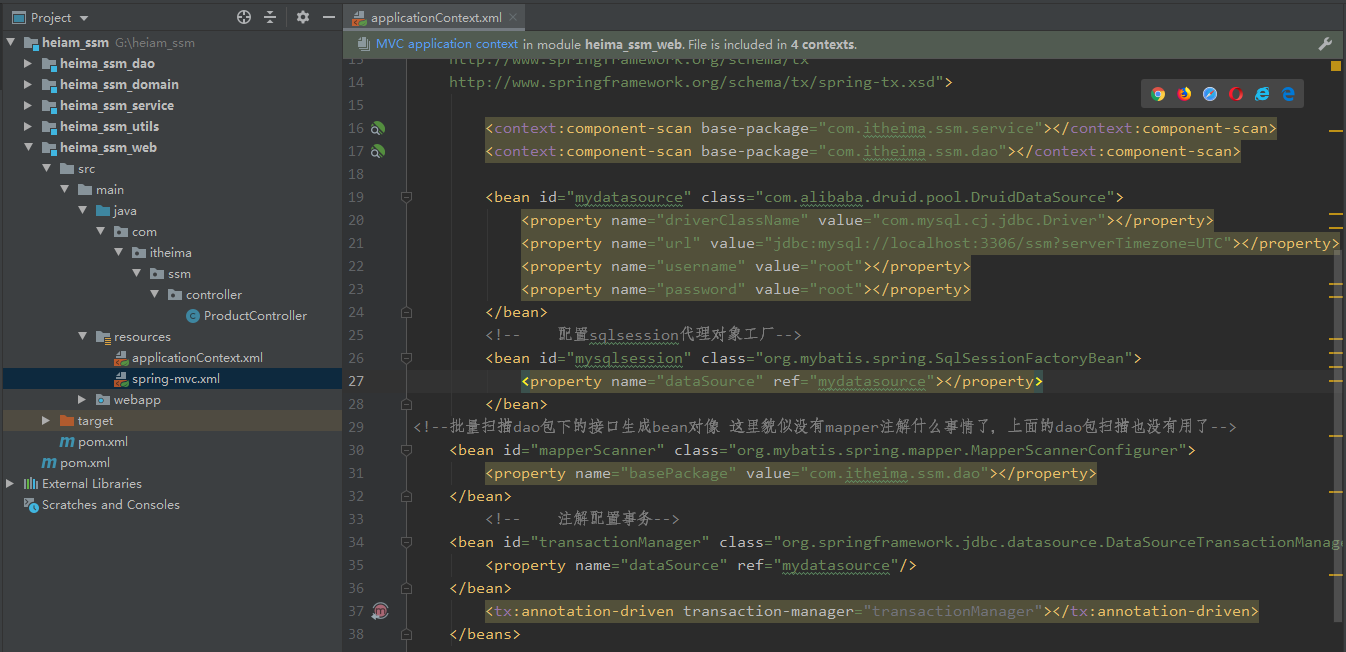


首先是mvc的配置：开启组件扫描；配置视图解析器；开启aop注解支持；静态资源放行；



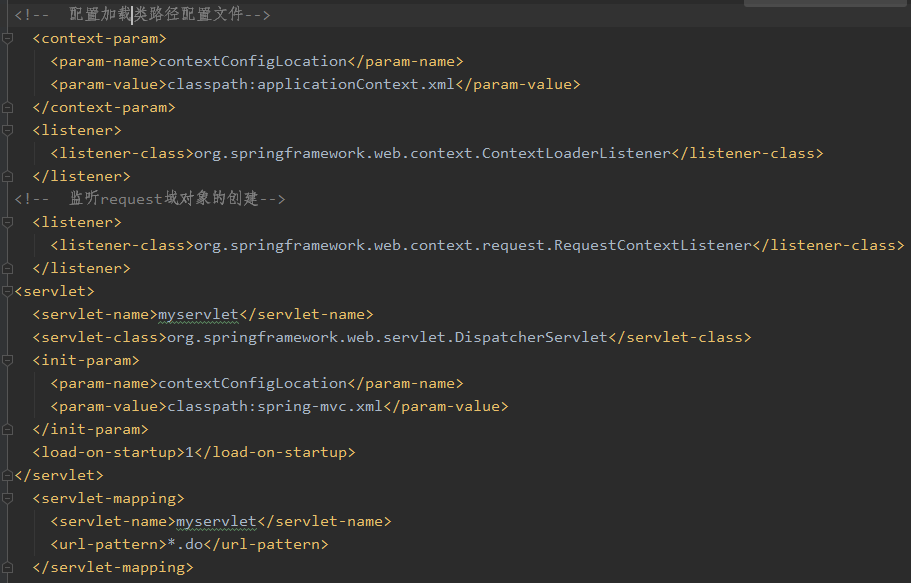
再是applicationContext.xml文件的配置（针对的是spring的整合）

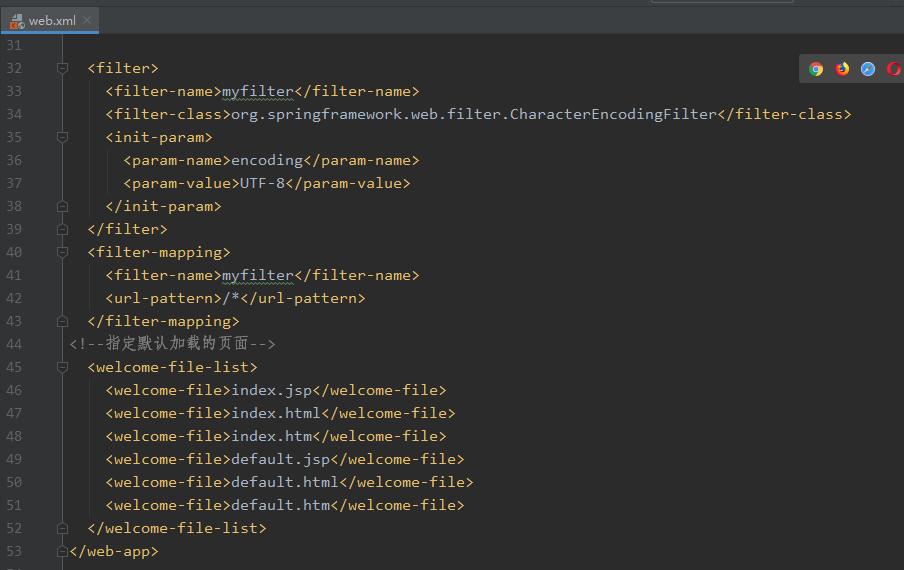
组件扫描；链接池配置；sqlsession工厂配置；批量扫描dao包下的接口生成对象（就是这里直接生成了dao接口的对象，我们的mapper才没有作用了？而且第二行的路径扫描也没有必要了！）；注解配置事务；



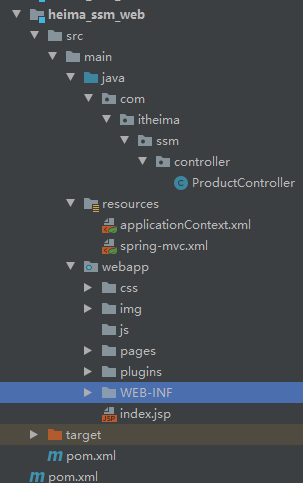
最后的web配置，这里实际上是配置tomcat启动配置

配置文件的加载路径（applicationContext.xml路径实际上是web/inf下的，这里配置后，更改为类路径）；配置监听器，一旦监听到全局对象的创建，就进行配置文件的解析；配置servlet前端解析器，指定mvc配置文件以及会拦截的路径为/\*.do(这是老师的命名习惯！不同与以前的拦截所有)；配置过滤器解决中文乱码问题；最后指定默认的展示页面！





这里导入了大量的前端资源哦，具体的我们后端开发不需要了解！



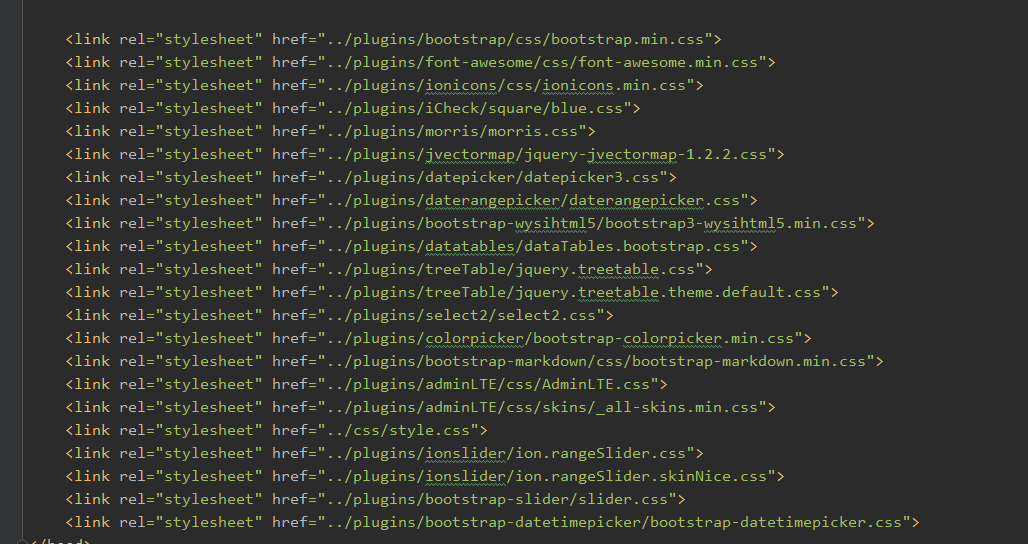
## 07：定制product.jsp页面：

如下，创建自己的list页面了解一下前端页面的结构和原理就好了：

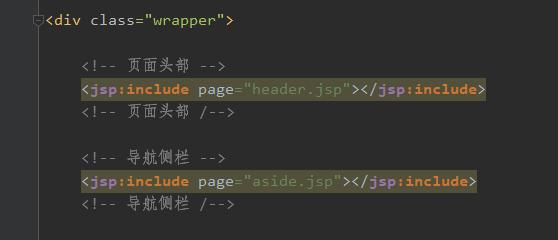
这里首先导入了jstl标签！后面会用到循环！



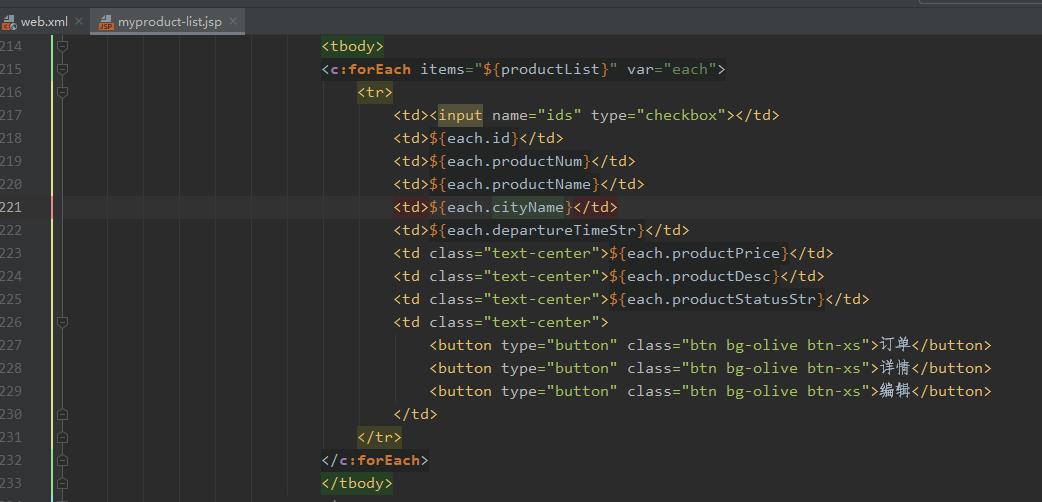
这里会导入大量的模板，需要注意路径的写法，具体情况会产生变化！



这里引入头部，侧栏等等，也是同一个文件夹下的，所有直接相对路径就完事！

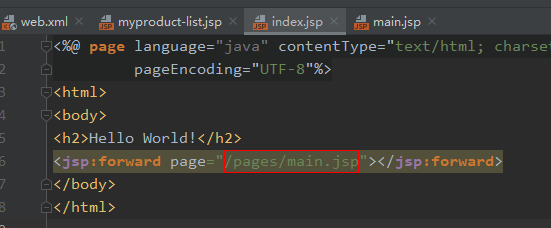


这里的主体展示采用了循环将所有查到的结果进行展示，用了el表达式以及jstl。最后结果和原来的效果一致！



## 08：定制main.jsp页面：

由于我们的欢迎页面太丑了，所以我们定制一个欢迎页面，在原本的页面实现转发就可以了！

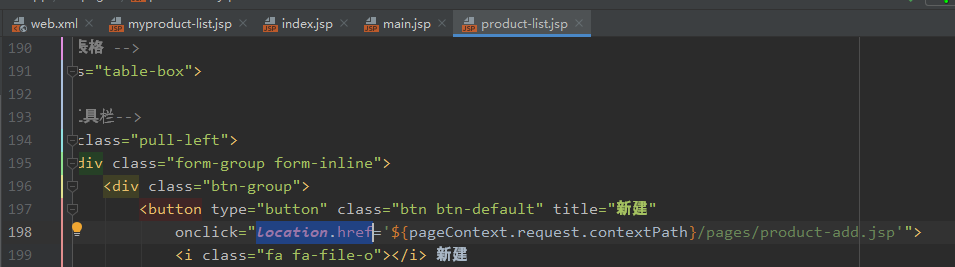


转发到的页面！

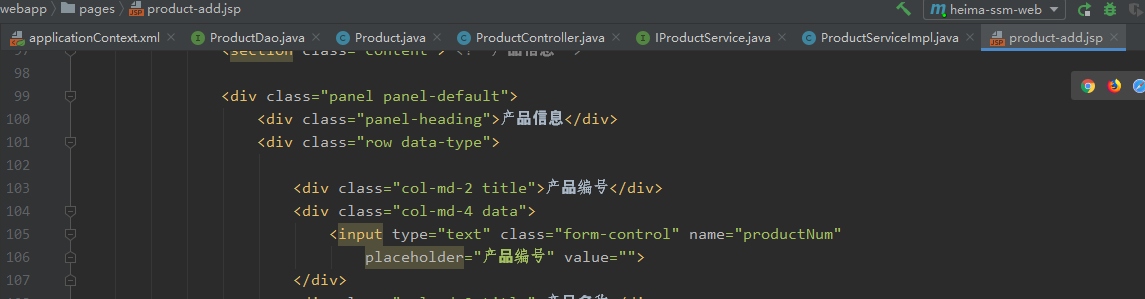


## 09：产品添加操作：

首先在新建按钮这里添加事件！跳转到add页面！



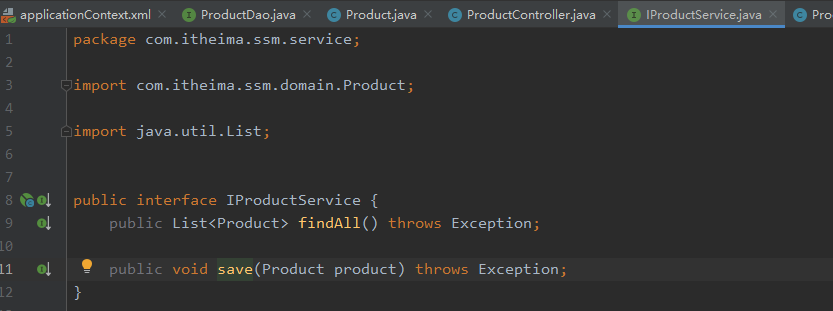
这里完成信息填写！



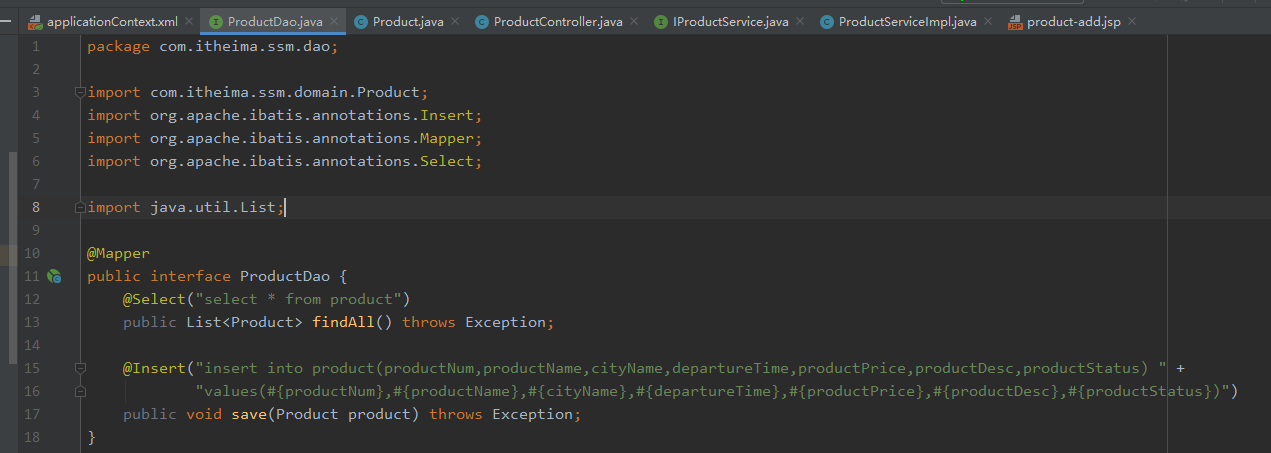
Controller类中添加如下方法！添加完成后实现转发



Service层中添加如下方法：



最后dao层我们添加方法，注解式实现方式。这里的departuretime提交的格式为string，但是我们要封装的格式式date，所以需要指定转换类型。以前介绍过一种，这里介绍另一种（注解式）。

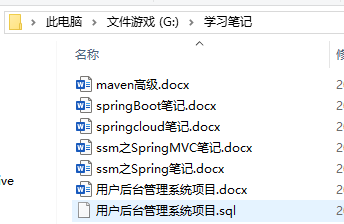


在product实体类中的departureTime属性上面添加注解指定我们的解析格式。这样的方式不同于以前，这个放在哪里，只有哪里生效。以前的方式是所有的都生效。



## 10：订单操作-创建表：

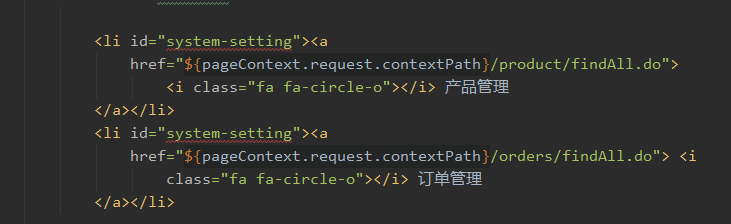
修改为mysql后的sql文件放在：



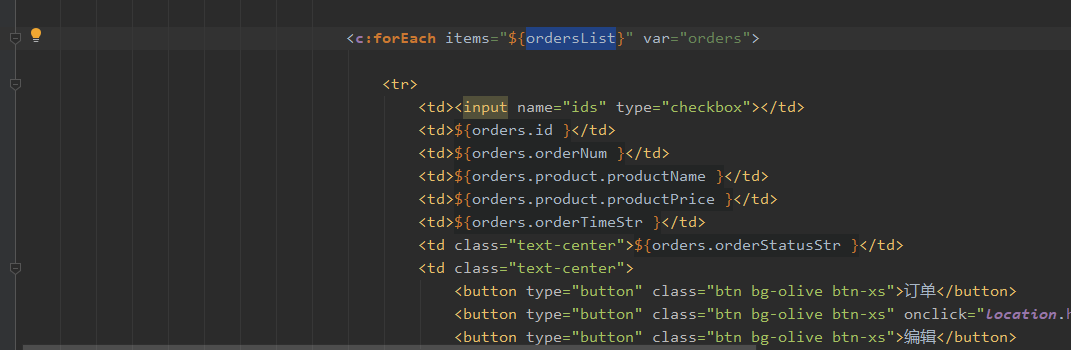
## 11：订单操作-查询所有订单代码实现：

这个环节最难的其实就是dao层的多表查询，因为涉及到多个表格，需要的注解比较麻烦！

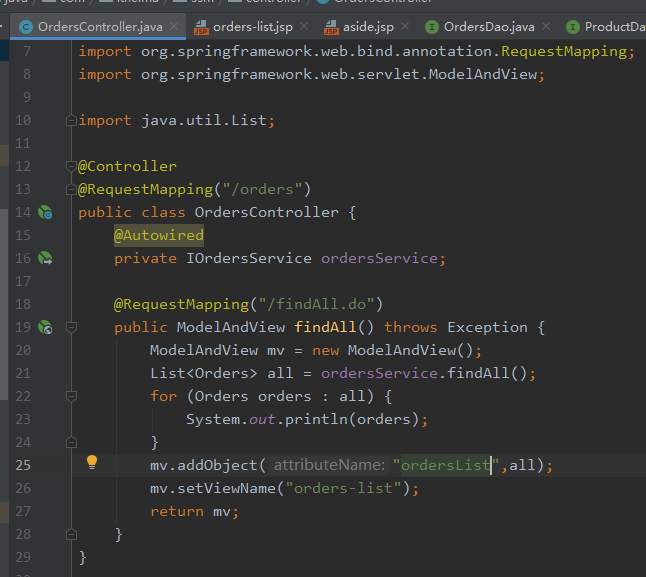
首先观察前端的页面！这里跳转的链接是orders/findAll.do



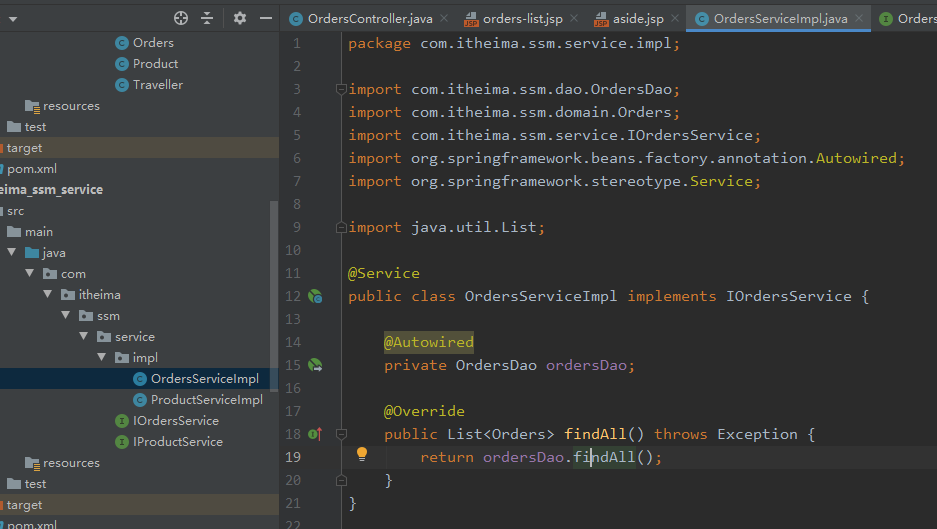
这里获取的json键名为ordersList。



那就首先创建一个表现层的controller。实现跳转功能。



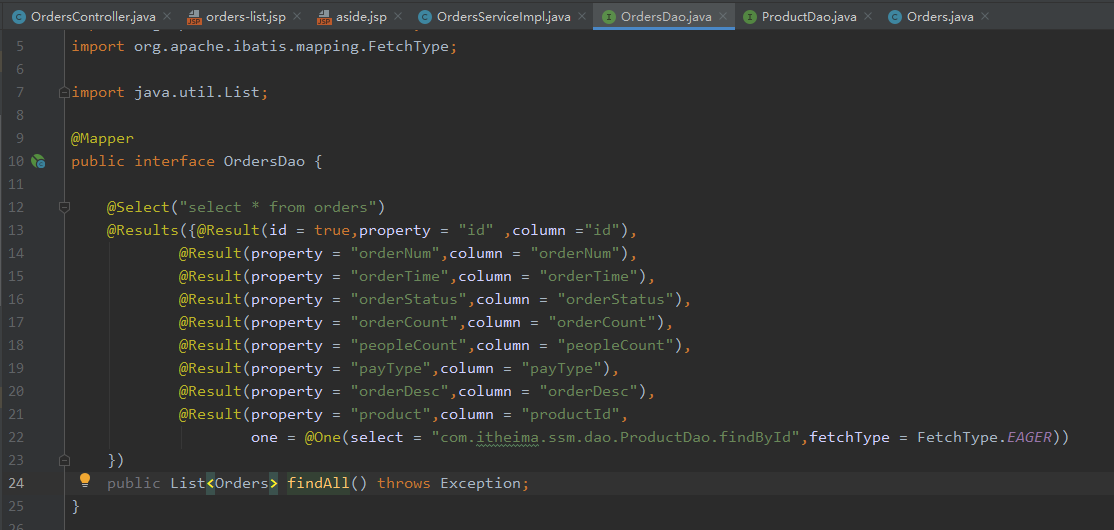
涉及到了order数据的封装，所以创建相关的业务层：

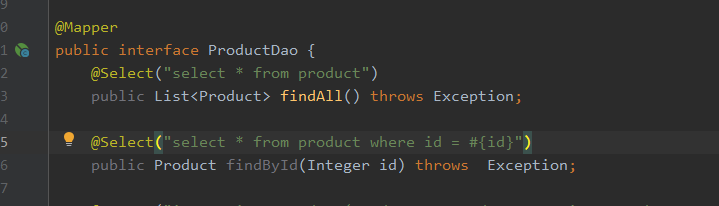


又调用了持久层：所以创建持久层dao接口（这里是难点！）

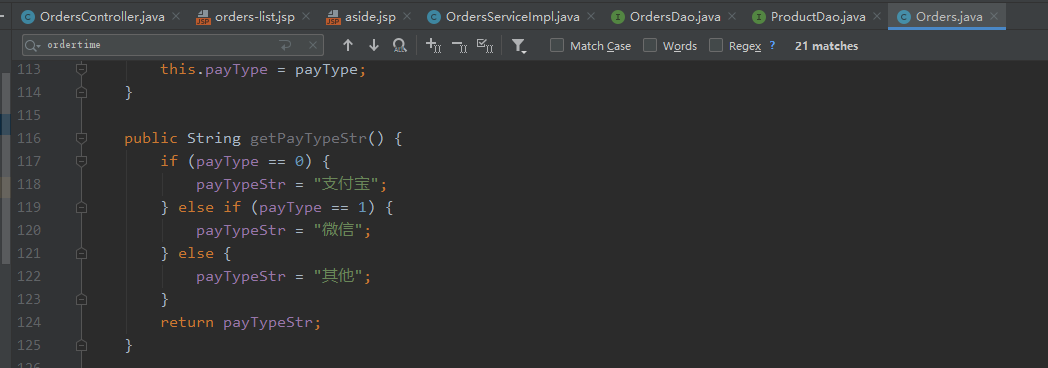
持久层的注解采用的results注解来指定我们的参数列表（用来做映射，防止属性和字段名字不相同），内部result注解指定特定的属性，这里需要将所有的都写上，即使二者一致。

重点是这里的product属性，他也是一个类用来封装产品的信息。和我们的order表存在一对一的关系。所以采用了多表查询的注解！fetchType指定立即查询。并且这里select指向了另一个表的id查询，所以需要创建这个方法！





最后将我们的order中一些str后缀返回给页面的参数进行设定！



结果正确！



## 17：page-helper分页查询使用步骤：

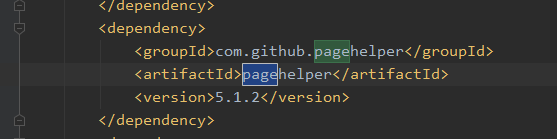
两种方式：

第一种：直接copy jar包进来。不推荐！

第二种：maven导入坐标！

步骤：

### ：导入依赖



### ：配置文件

如果是mybatis单独使用，需要.xml文件支持！

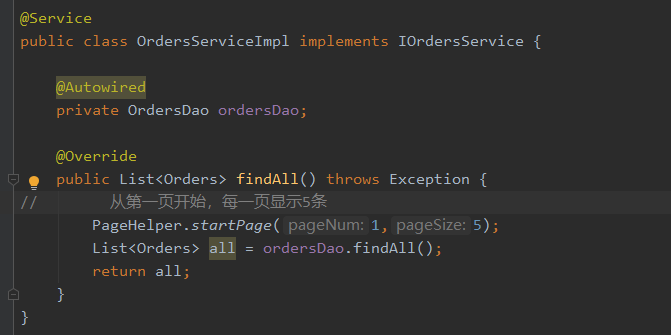
如果整合到ssm框架中，我们只需要引入插件就好了：

在我们的sqlsessionFactoryBean类中添加插件属性即可：标签具体作用注释，以及百度！



### ：分页插件使用

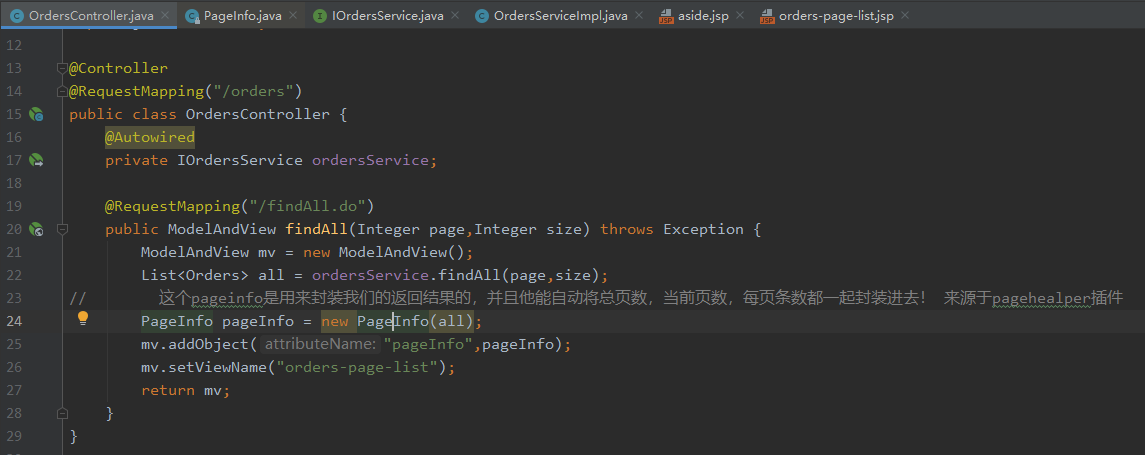
在我们真正使用sql的前面使用pagehelper.startpage方法就可以了，一定要挨着！



## 18：page-helper分页查询代码实现：

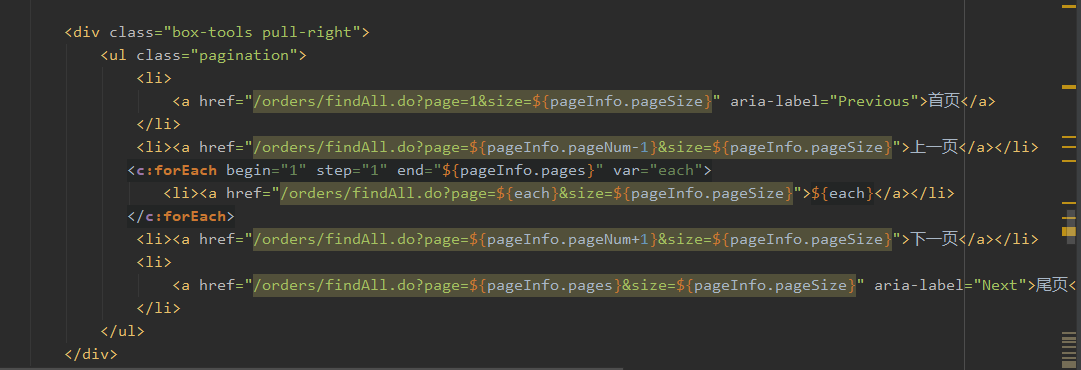
首先需要更改我们的controller表现层：

这里将我们的对象传入改变成为了pageinfo，他是pagehelper插件提供的实体类，里面一方面可以封装我们的返回list集合，另一方面封装了页数，当前页，总条数等等！



下面修改前端页面：

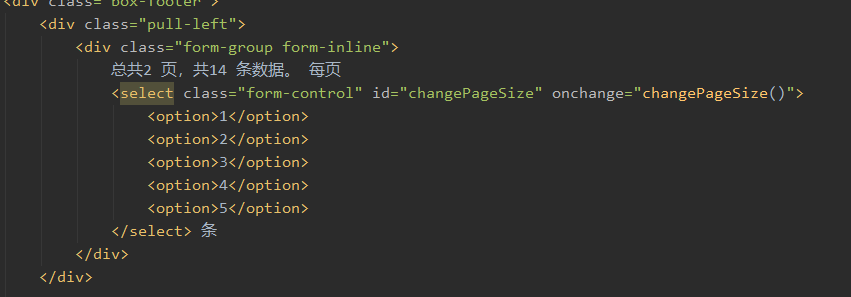
导航框的上一页下一页等等的href都进行了页码，页数绑定！



这里的下拉表单可以让我们选择每一页的显示条数！

里面的精髓在于onchange事件会触发自定义的函数：

更改条数并且重新查询！



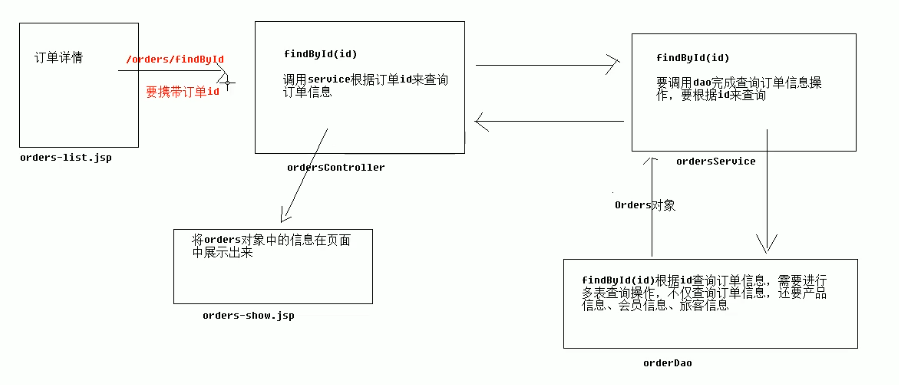


结果成功实现功能！

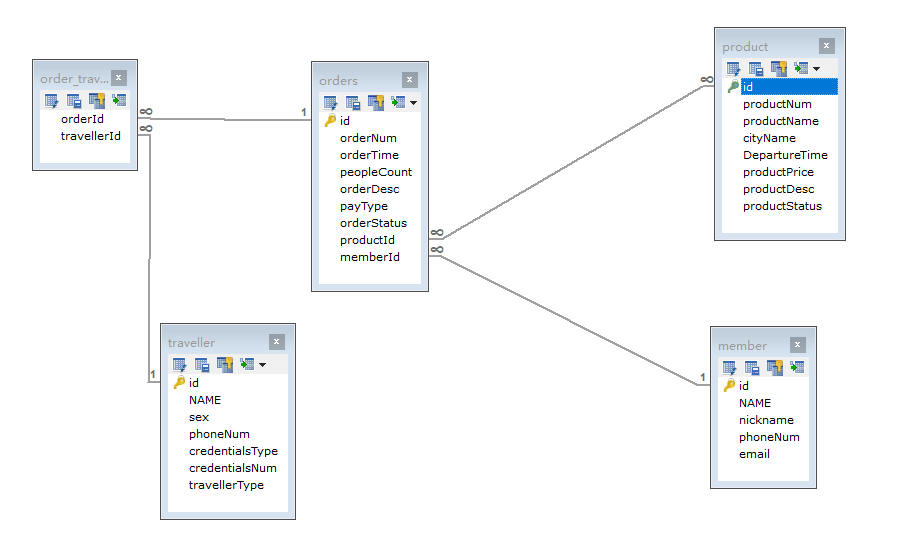
# Day03：课程介绍以及注意事项

## 01：订单操作-订单查询以及流程分析：

流程分析图：主要的难点集中在多表查询中!

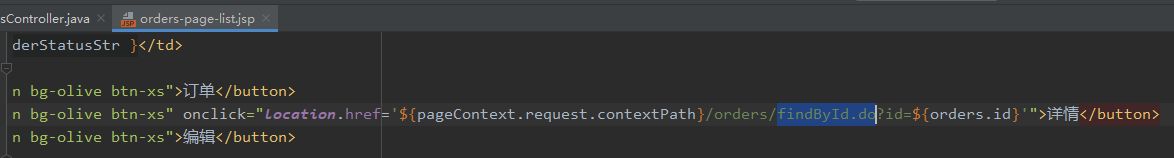


多表之间的关系：



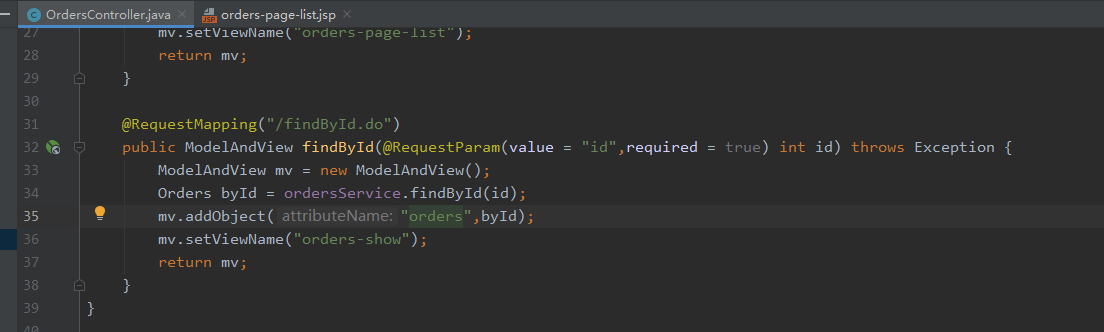
## 02：订单操作-订单查询代码实现：

首先查看前端页面：

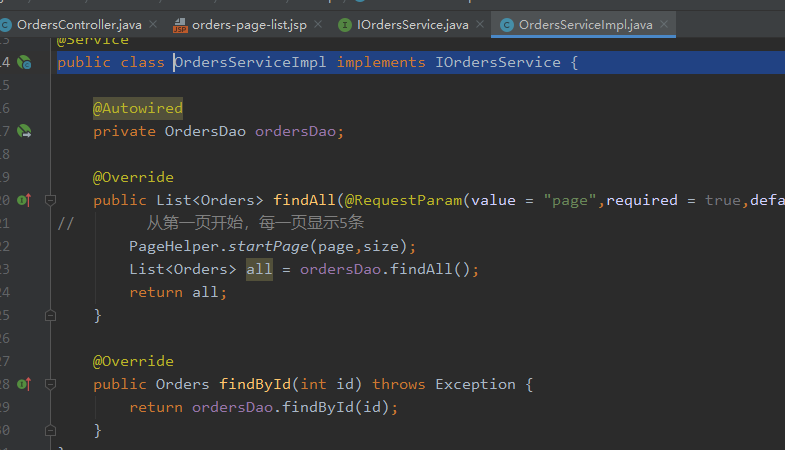


这里跳转orders/findById.do并且指定了id参数，所以我们在OrderController中创建对应的方法：

这里对order进行多表查询：返回show页面：

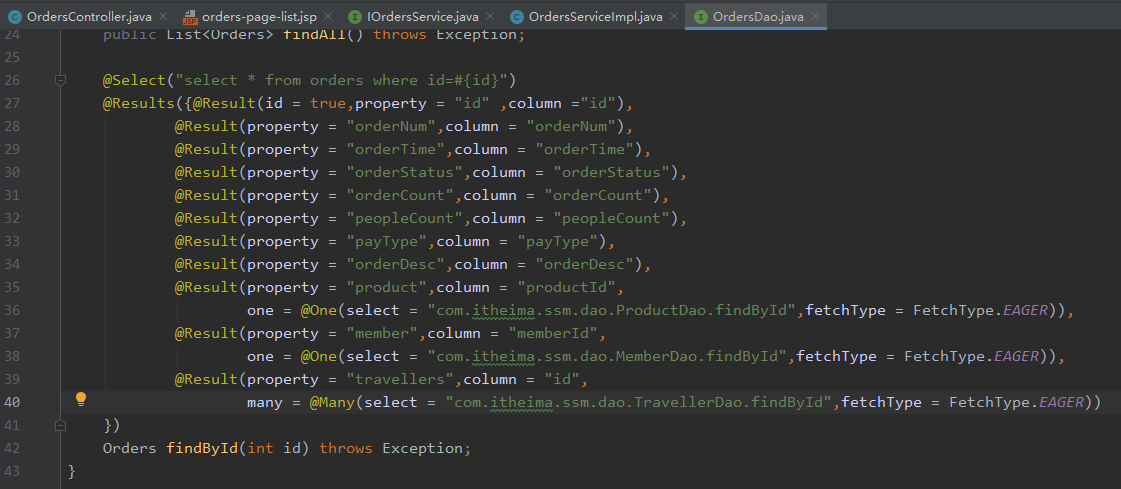


OrderService中创建findById方法进行业务层操作！

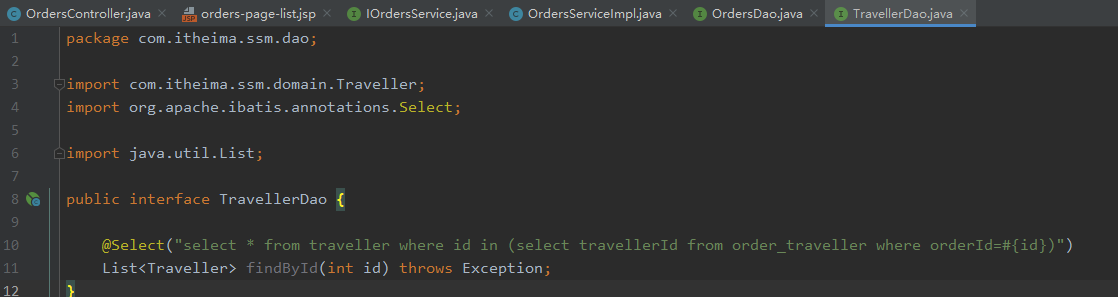


最后在OrderDao持久层进行注解多表查询！

这里是难点：针对member实体类，是多对一的关系，比较好实现，只需要进行指定id查询就好。



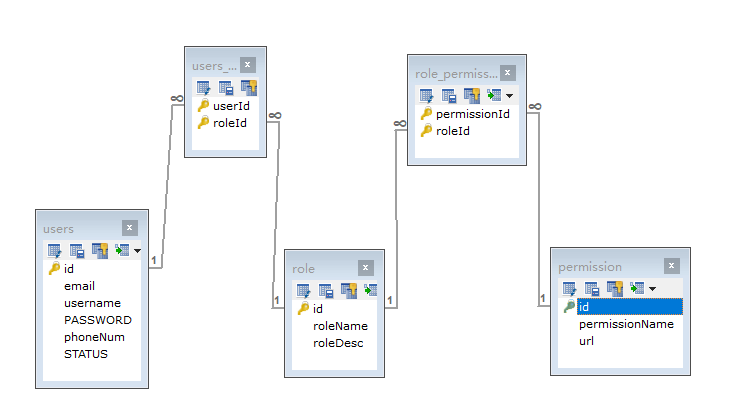
第三个traveller实体类，和order是多对多的关系，首先指定id查询，再在TravellerDao接口中进行注解：由于是多对多的关系，这里先根据orderid查找中间表中所有与指定orderid对应的travellerId，再在traveller表中查询所有关联travellerId的信息封装成list的实体类集合，返回到我们的order实体类中List<Traveller>属性中，完成了多表查询！



到此页面成功展示！

## 05：权限-表结构分析以及创建表：

表结构如下所示

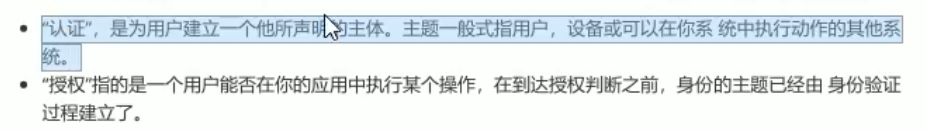


## 06：spring-security概述以及快速入门：

新创建一个项目：

这里仅作示例，和项目无关！

spring-security的两大功能：一个是认证，一个是授权！

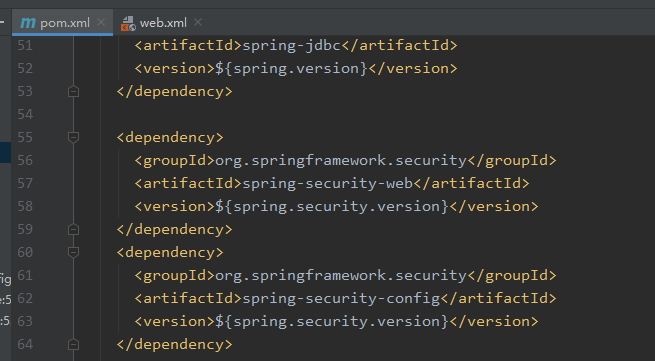


两种方式都可以使用spring-security，第一种是配置文件，第二种是数据库！（我们的授权，认证就是存在数据库的）

以下是示例，我们不涉及数据库，仅仅在配置文件中进行演示！

## 07：spring-security使用步骤：

### ：导入依赖



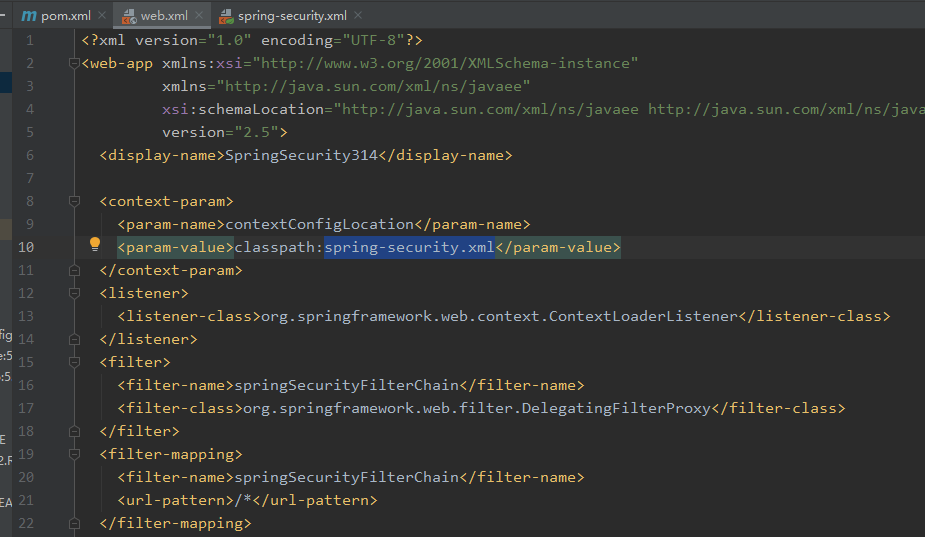
### ：配置web.xml文件

Web.xml配置文件中配置好监听器：

一旦监听到就对我们的spring-security.xml进行加载

在配置一个filter过滤器：

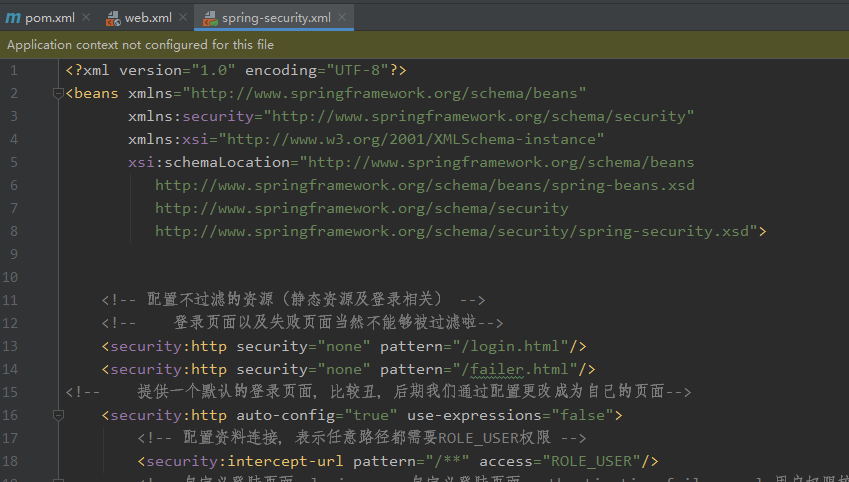
这里的过滤器名字有固定要求，具体原因查看源码：（这里配置对所有的路径都进行拦截）



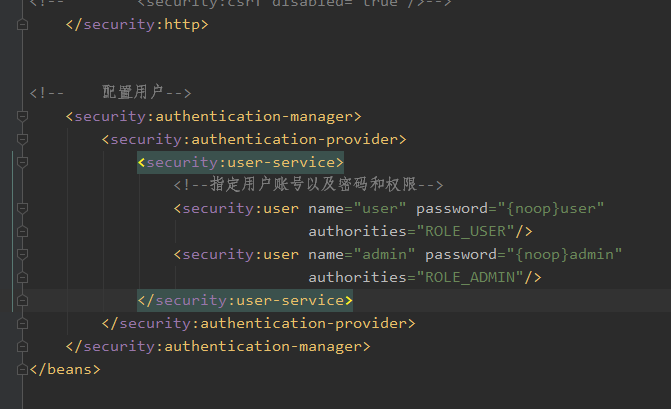
### ：配置spring-security.xml文件

首先要制定不进行过滤的资源：我们的登陆页面以及失败页面以及一些静态资源都不能进行拦截。

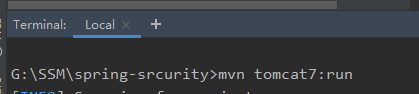
然后需要我们指定登录页面，这里开启了自动配置登陆页面！在内部我们也指定了登录的权限



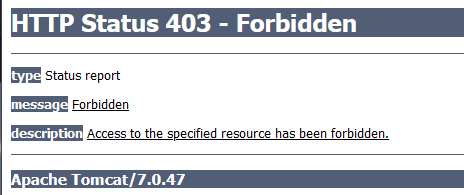
在这里我们采用配置文件的方式指定了两个用户以及他们的权限，后期这里要换成数据库！



这里将我们的项目进行启动。



采用admin进行登录发现无法登录，报错403就是表示权限不够。



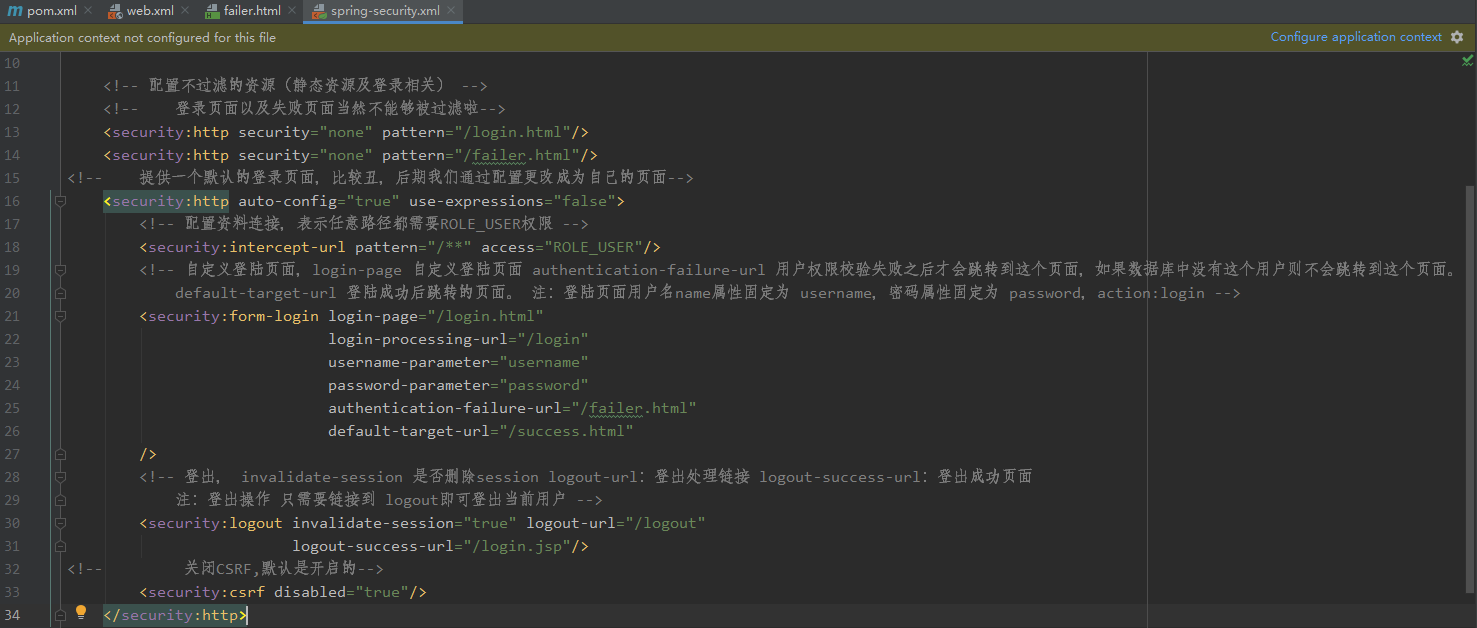
采用user进行登录就会成功且跳转到index.jsp页面！

## 08：spring-security使用自定义的页面：

前面我们指定的是开启自动配置的登陆页面，实际情况肯定不能满足，所以需要我们自主进行配置！

实际非常简单，只需要我们进行如下配置！具体解释注释已经解释清楚：

这里指定了登录的url以及登录的页面，以及登录成功后或者失败后的跳转页面！



## 09：用户登录-登录流程分析：

使用spring-security配合数据库进行认证操作，可以有很多种方式：

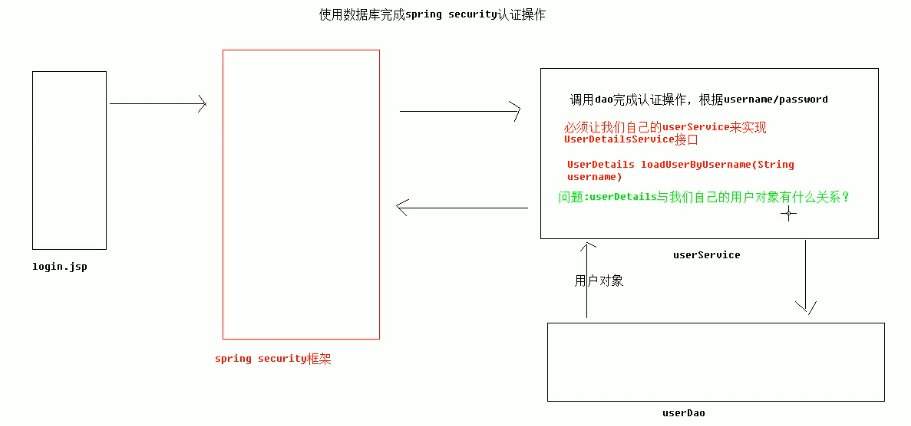
我们这里介绍一种：userDetails，以及userDeatilsService来完成操作！

他们两个都是接口，我们需要自己写实现类进行重写方法！



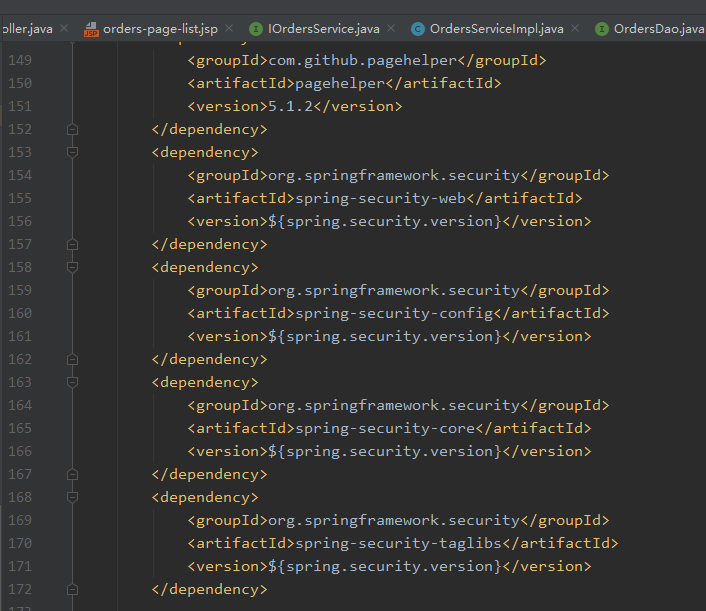
由于我们已经有spring-security框架了，所以不必要在配置controller了！我们已经在spring-security.xml文件中指定了登录的url，登录的页面，登录完成后结果的跳转页面！

另外一个需要我们注意的是，我们调用dao层返回的实体类需要封装到userDetails类中去才可以交给spring-security解析。（这一点类似前面分页的pageinfo类，他也是将我们dao层封装的实体类又进行了重新封装！）



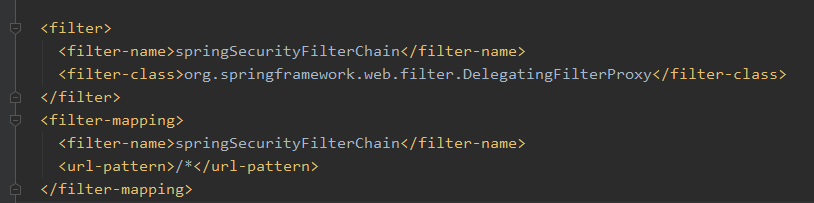
## 10：用户登录-代码实现1：

首先导入相关的maven坐标：

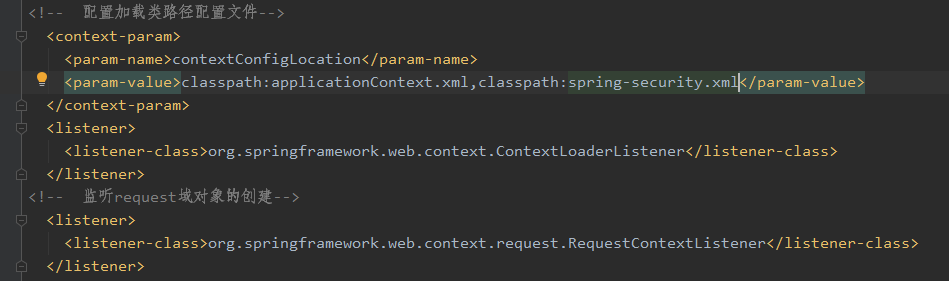


在web.xml文件中导入过滤器：

这里千万注意，过滤器的name是固定的，不可以低级随便写！



在监听器中添加我们的spring-security.xml配置文件！



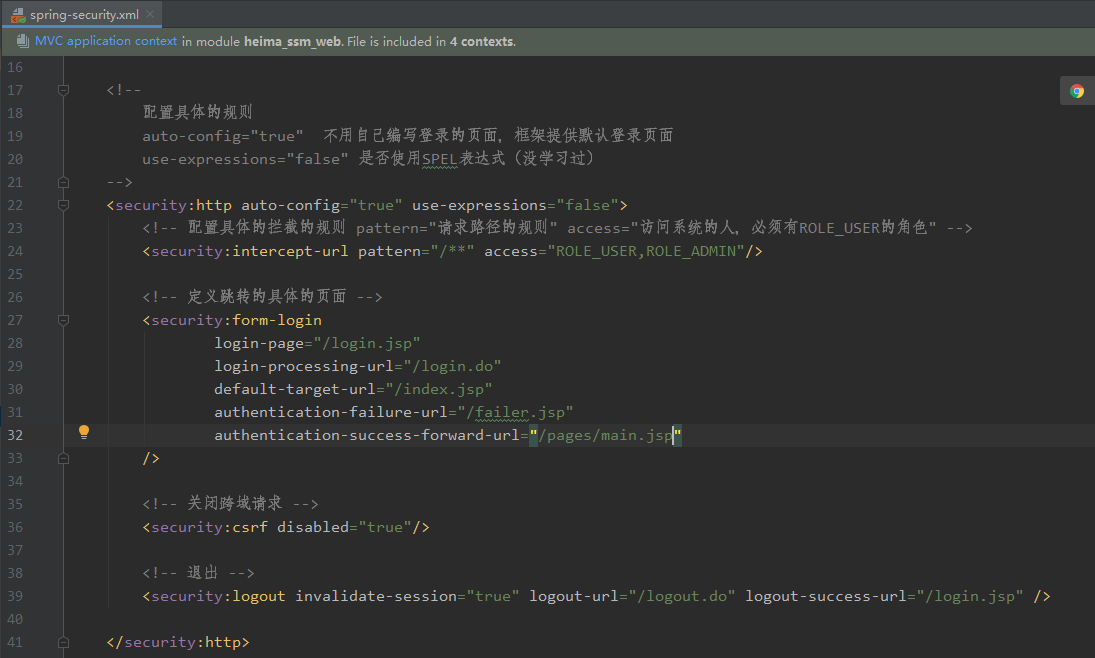
配置文件中我们做出如下配置：

首先配置不拦截的资源：我们没必要对登录页面本身进行拦截，一些必要的静态资源当然也不能拦截！



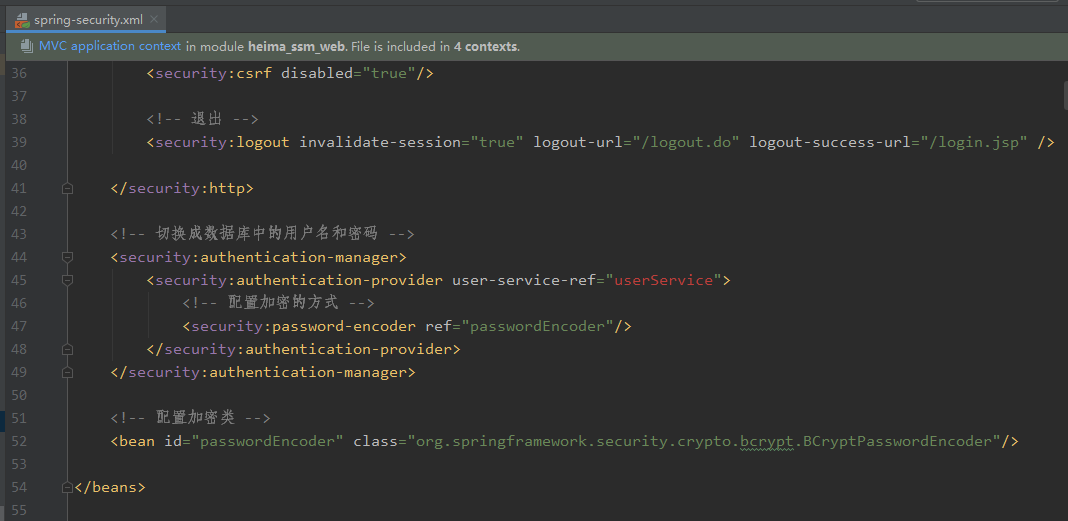
这里配置了我们登录的url，登录页面，登录后结果展示页面，以及我们的权限允许（这里允许role-user以及role-admin这两种权限【权限的种类在我们的用户表中有指定！】）这里也指定了我们用户退出的url，后面讲解。

这里没有像以前一样指定我们的username-parameter以及password-parameter参数，因为不指定的话默认就是username和password，我们数据库中的字段正好就是！这里也指定了我们的用户权限，只有指定的权限用户才可以进入页面！



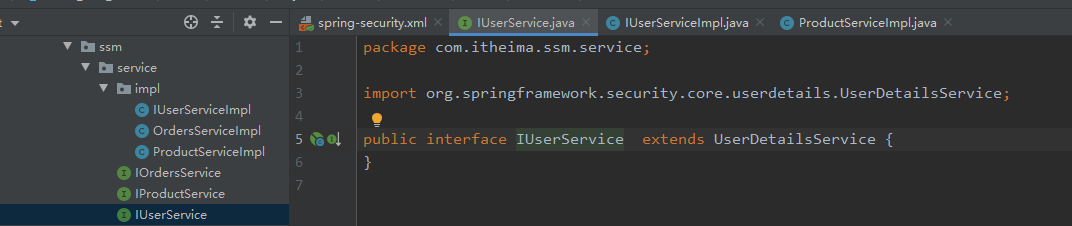
我们的spring-security本来是默认寻找配置文件自己注册的用户的账号密码，想让它知道采用数据库的话需要指定配置：如下

这里的userService就是我们要指定的service业务层。他直接跳过了controller表现层!



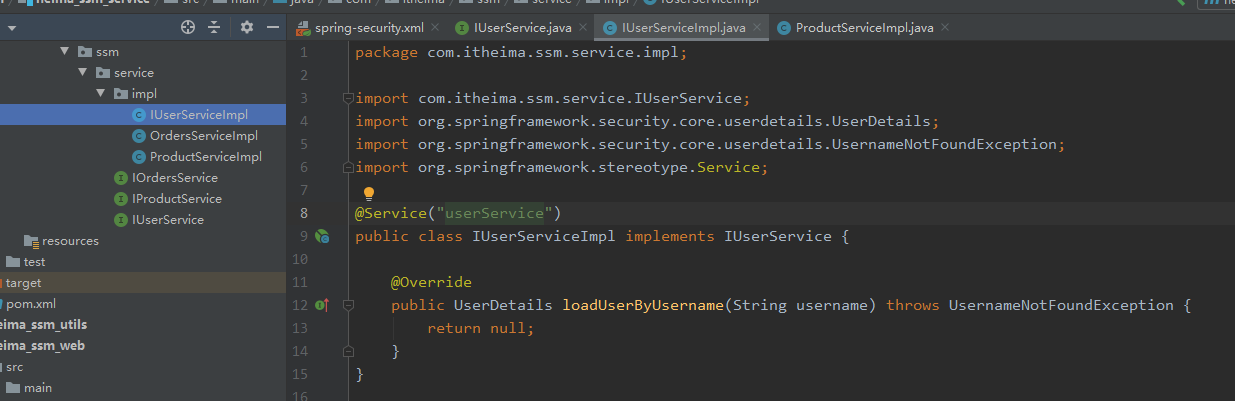
接下来我们完成service业务层的配置！

创建业务层接口继承userDetailsService接口



在业务层实现类中重写loadUserByUsername方法！

这里千万注意@servie中的参数userService与我们配置文件中的user-service-ref指定的参数一致，这样才能让登录跳转进我们service业务层！

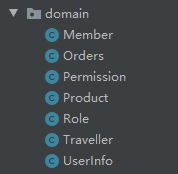


至此，框架基本搭建好了！

## 11：用户登录-代码实现2：

完成代码细节：

创建好我们用来封装数据的实体类：

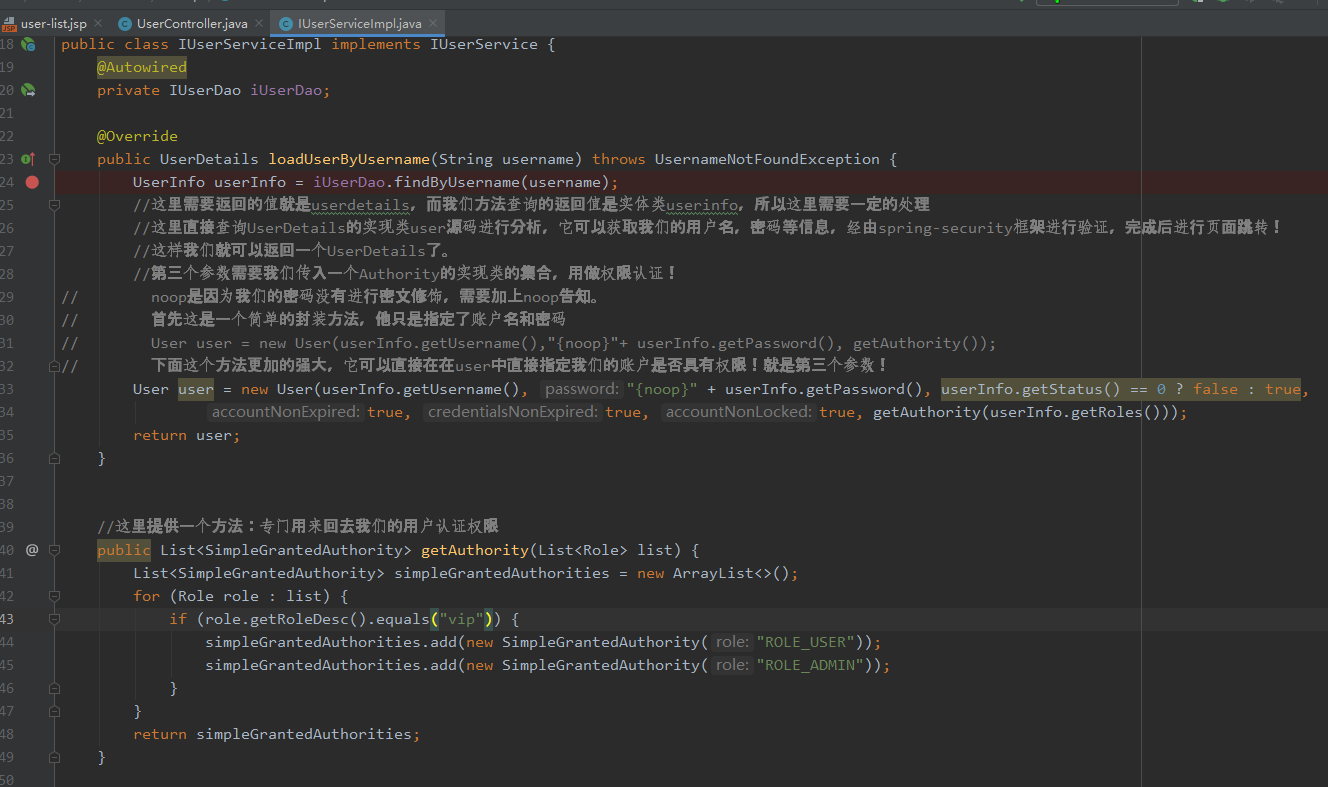


首先完成service业务层。

第一点需要明白的是我们spring-security框架需要返回一个userDrtails（接口，需要他的实现类）的类，而不是我们以前简单的一个实体类！这里的转换是我们的一个学习点！

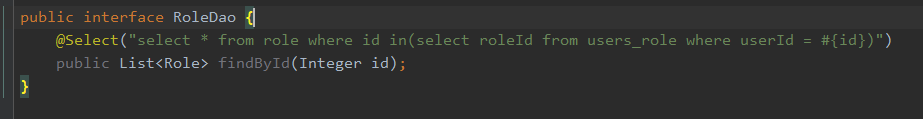
首先获取我们的实体类：

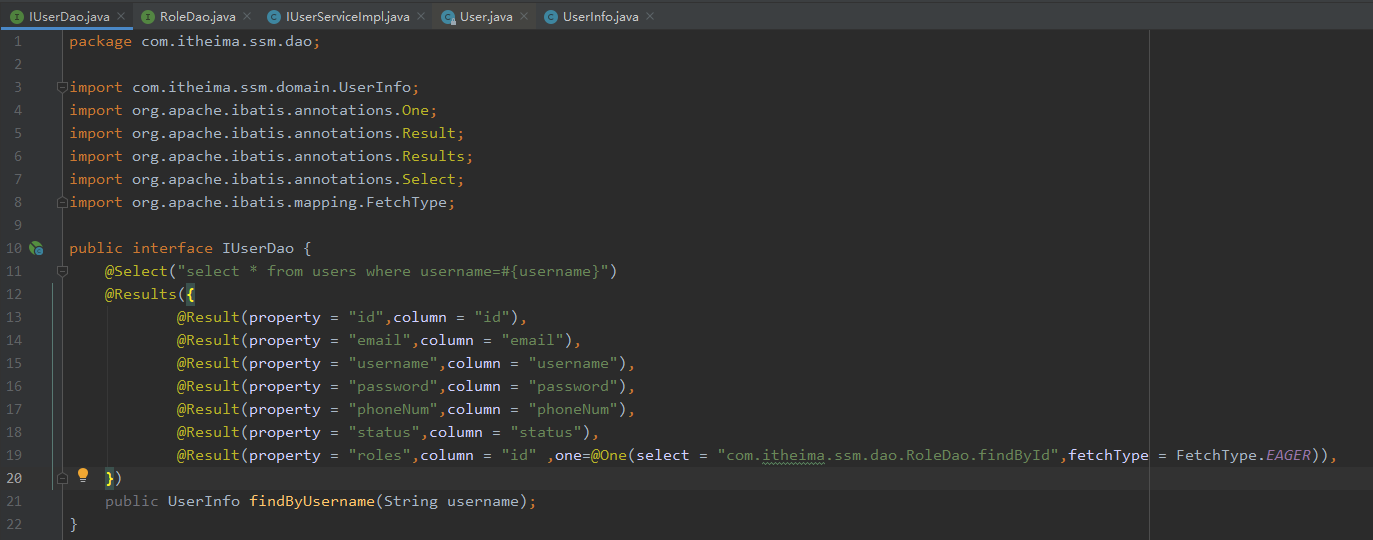
将实体类的username，password等参数填入user类中，其中第三个参数可以直接指定我们当前账户是否具有权限登录！最后一个参数可以指定我们当前用户的账户具有的权限！



在我们的dao层进行封装！

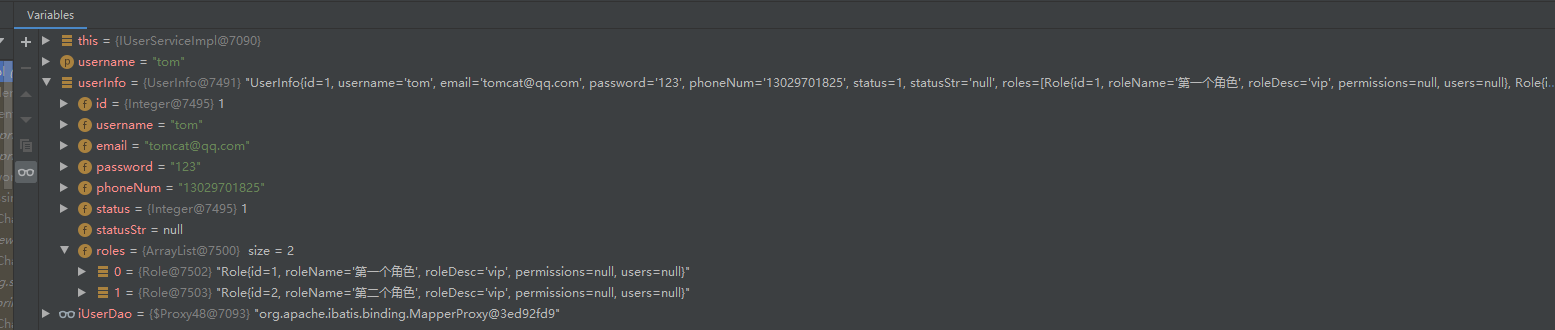
这里进行了多表查询，因为我们的users表和role表是多对多的关系，原理和我们以前的多表查询一致！





最后debug启动，在我们的user实体类中发现如下数据：

证明封装成功。页面跳转也正常！



## 12：用户退出：

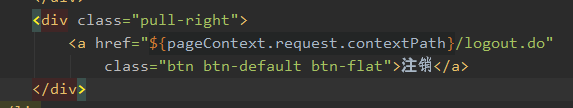
在spring-security中我们完成退出工作非常简单！

只需要在我们的spring-security.xml文件中进行一定的配置：

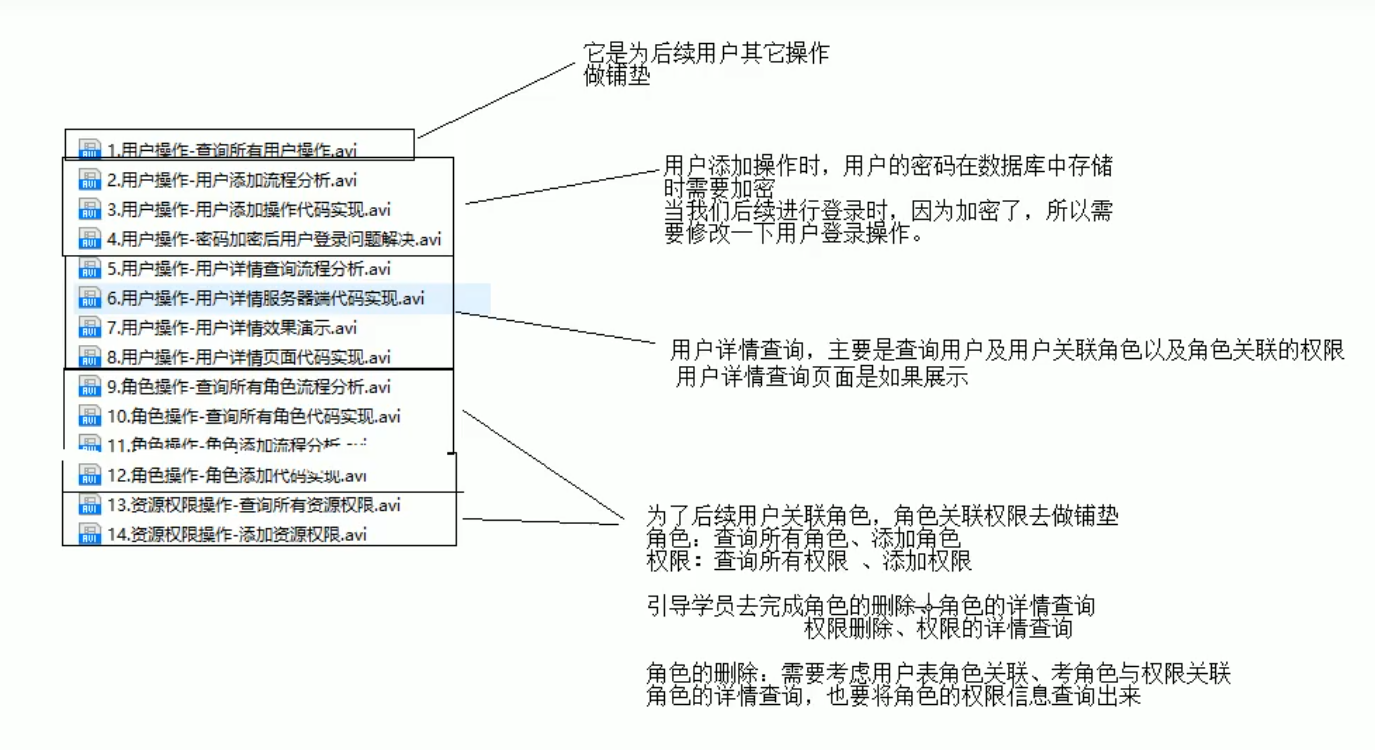
在配置标签中开启退出功能！指定我们的退出触发url以及退出后跳转的页面就可以了！



点击注销自然就执行url，跳转到我们的login.jsp页面，同时后台会自动的销毁session等数据！



# Day04：课程介绍以及课后练习

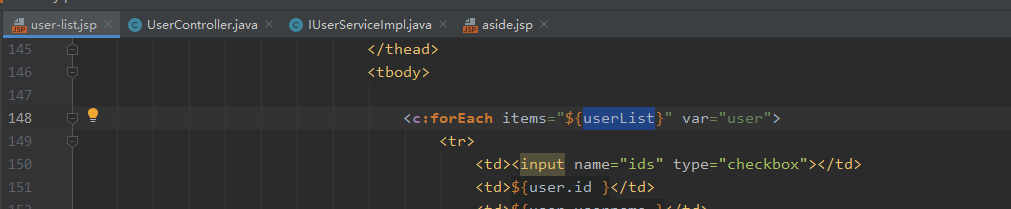


## 01：用户操作-查询所有用户：

首先观察页面这里跳转的路径：

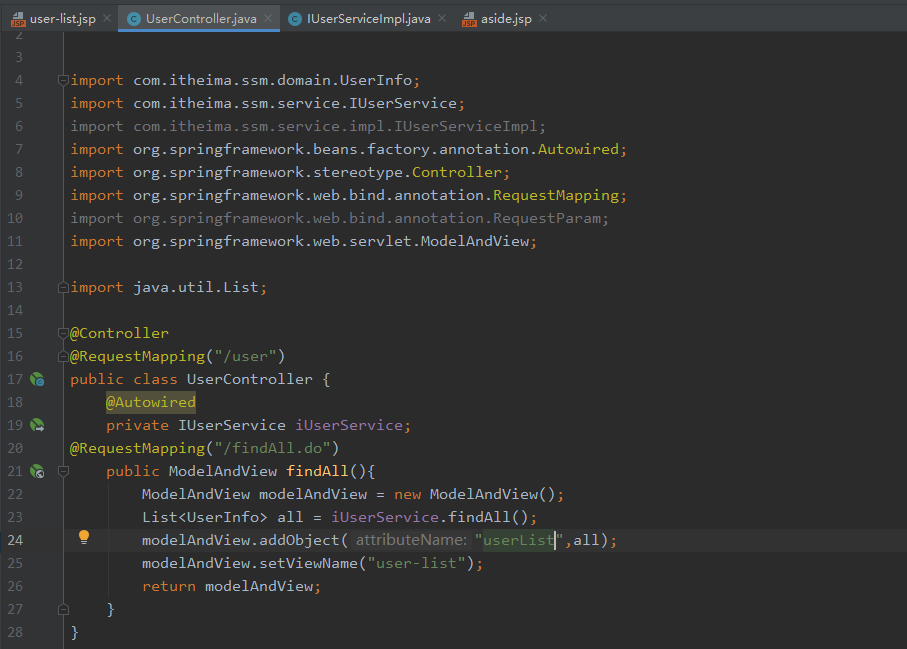


再观察我们返回的json键名：



根据以上两点来创建controller表现层！

创建好的controller中我们需要调用业务层userservice中的findAll方法：

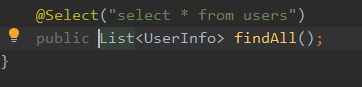


根据需求再我们的userservice层进行方法的创建：

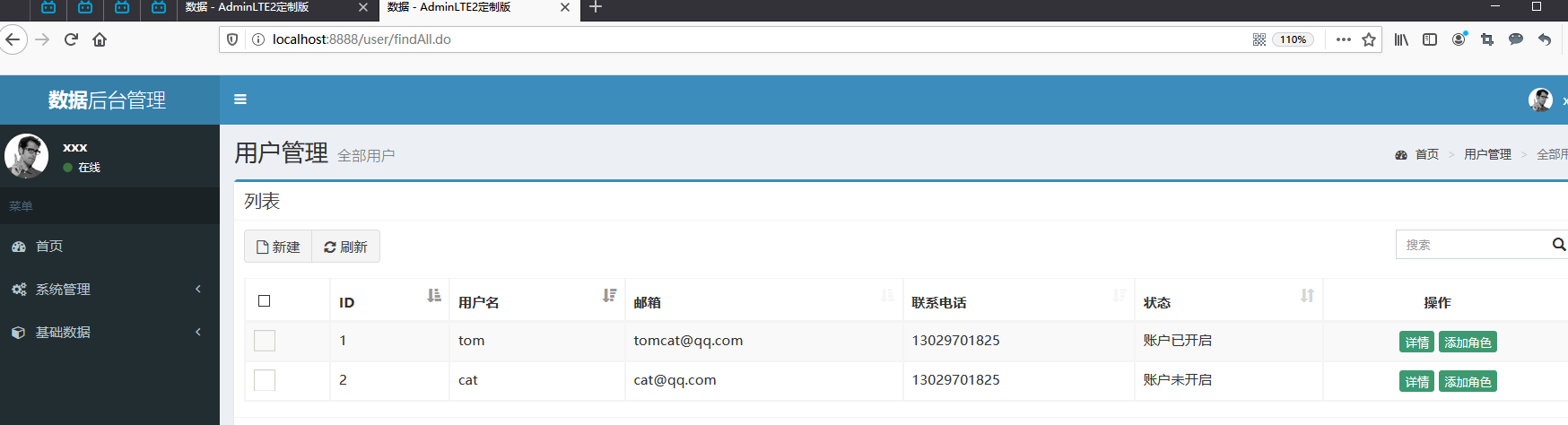
这里又调用了我们的userdao业务层中的findall方法：



根据需求我们继续创建方法：

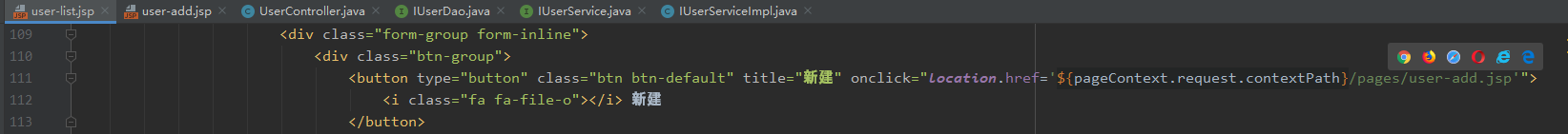


最后打开页面进行查询：

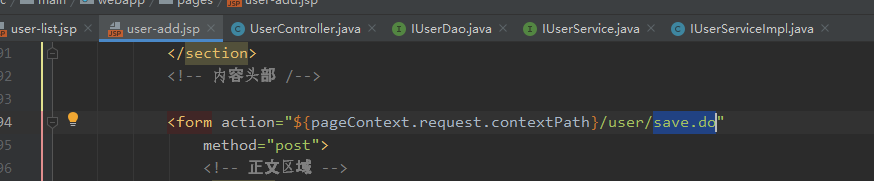


## 02：用户操作-添加用户：

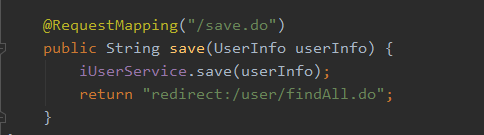
首先观察页面：新建用户的页面如下



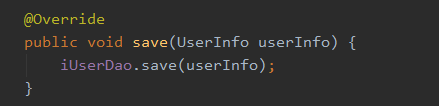
提交表单后的跳转路径：



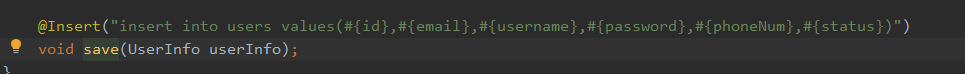
根据以上创建我们的usercontroller方法：



调用了我们的userservice中的save方法：我们需要创建：

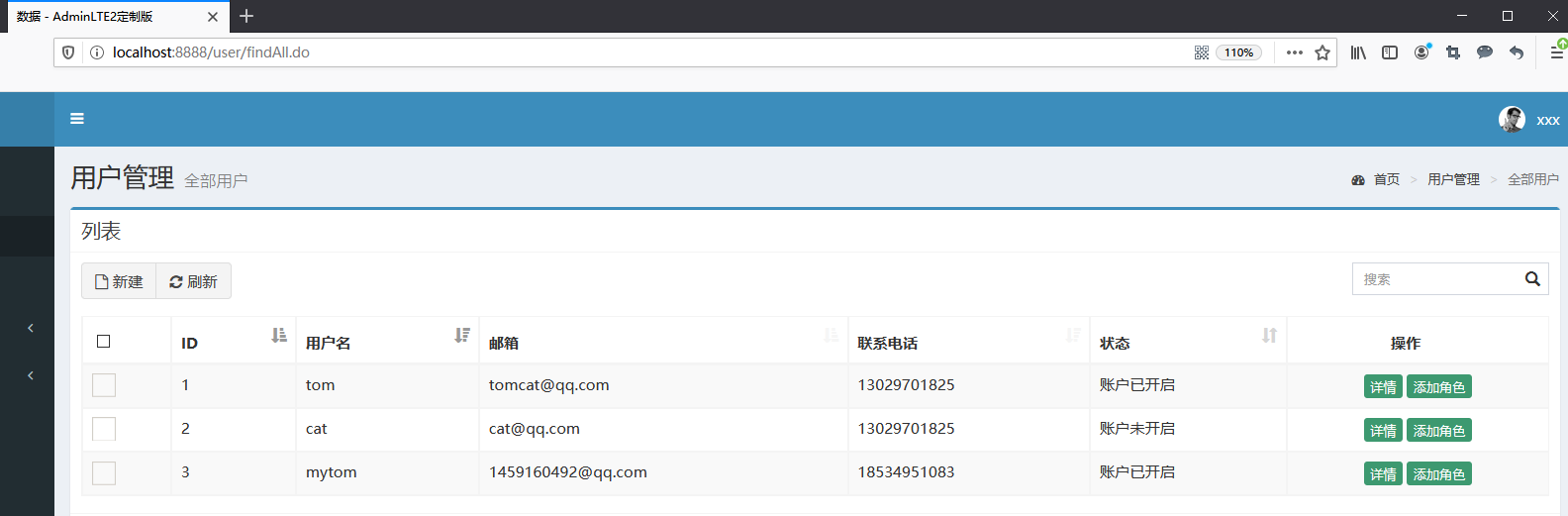


又调用了userDao中的save方法，我们需要创建：



最后实现添加功能效果如下：

三号用户就是我们手动添加的！

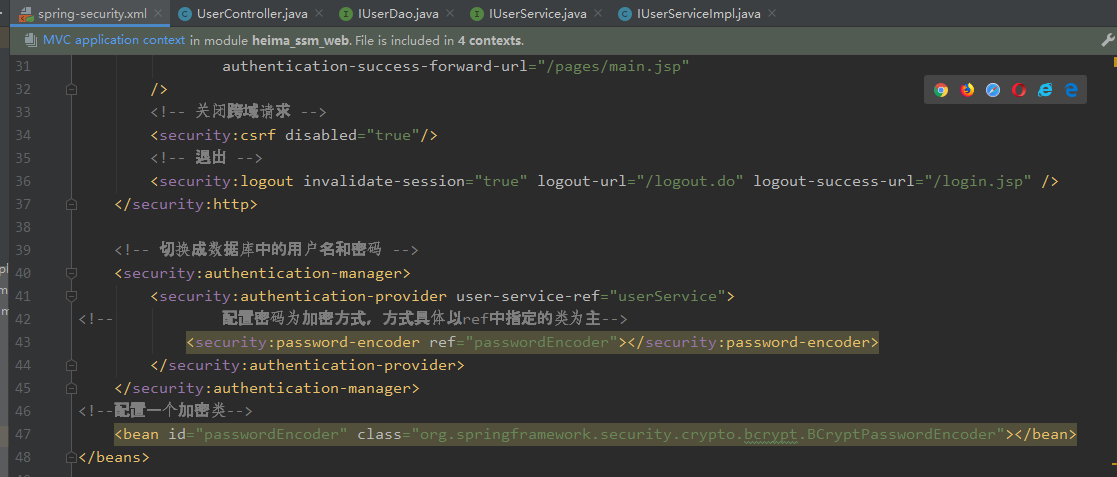


## 04：密码加密后用户登录问题解决：

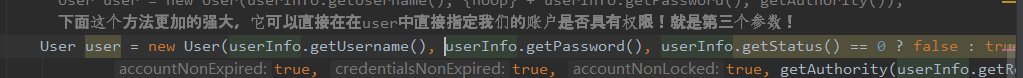
首先在spring-security的 配置文件中添加一些标签：

第一处修改指定我们要开启密码加密：

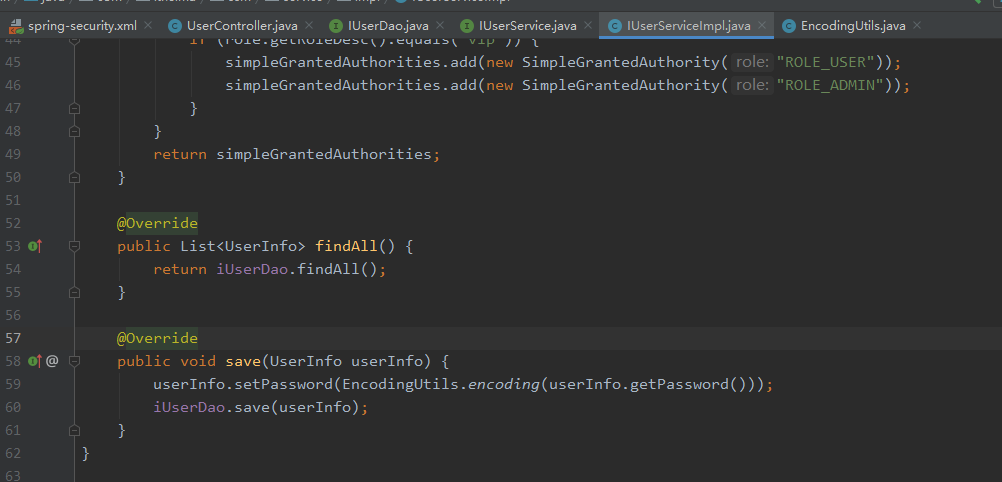
第二处修改指定我们加密的具体方式！



同样：这里以前password前面的“{noop}”也就不需要了！

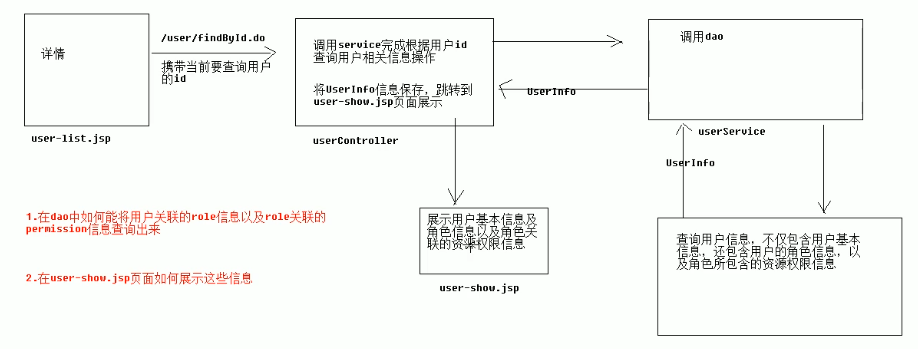


最后在添加用户里面进行密码加密操作！



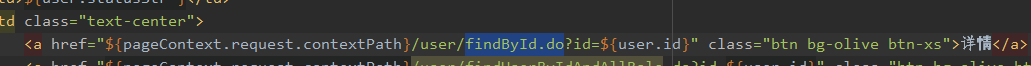
## 05：用户操作-用户详情查询分析：

这里详情查询会涉及到用户，角色以及角色中的权限的多表操作，是这里的难点！

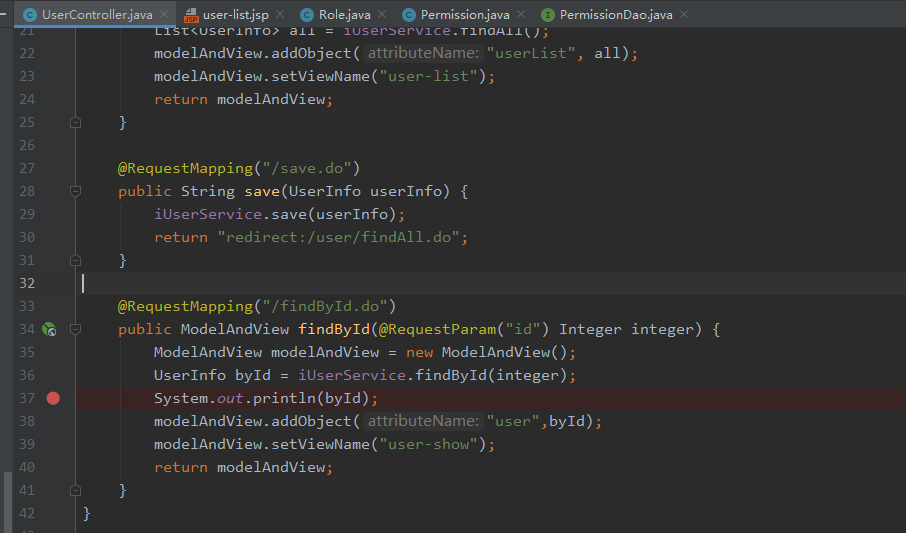


## 06：用户操作-用户详情查询代码：

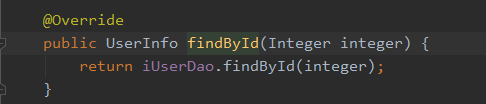
首先观察页面：



根据这里的跳转url创建我们usercontroller中的方法：返回user-show.jsp页面：

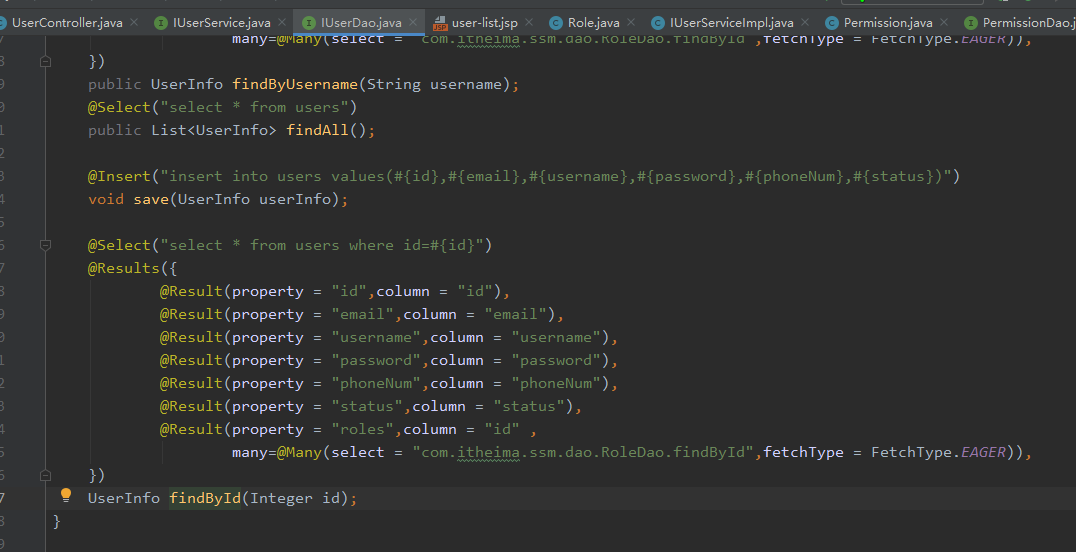


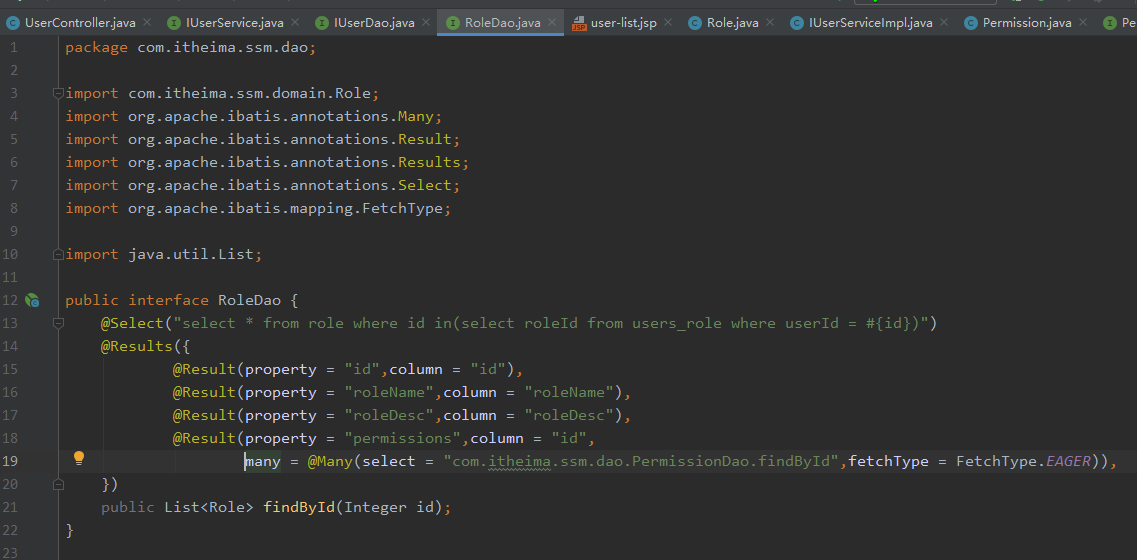
上面调用了findById方法，我们在userservice业务层中创建对应的方法：

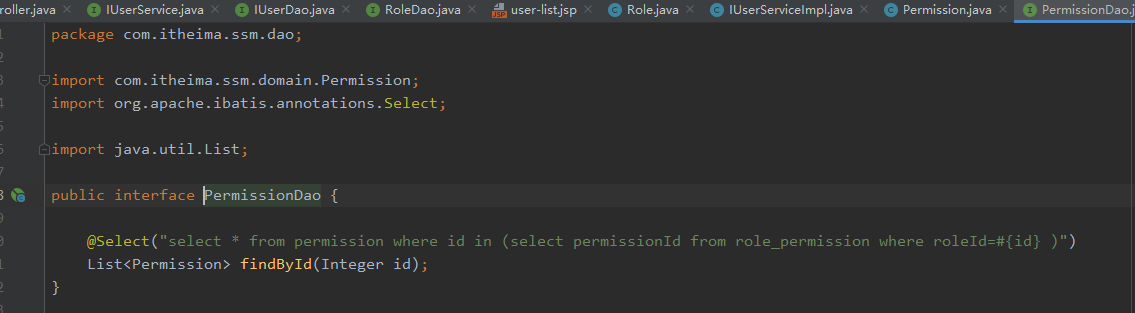


这里又调用了userdao持久层的findbyid方法，我们也进行创建：

这里的难点就是我们需要进行多表查询：即需要查询到我们的user表数据，也需要查询user关联的role表的数据（多对多），又需要查询role关联的permssion表数据（又是一个多对多）：

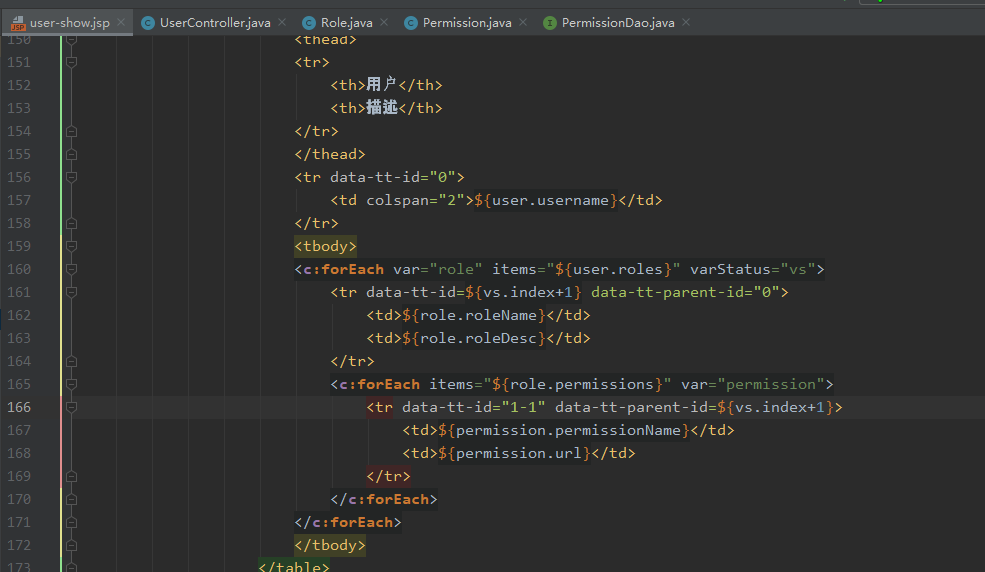




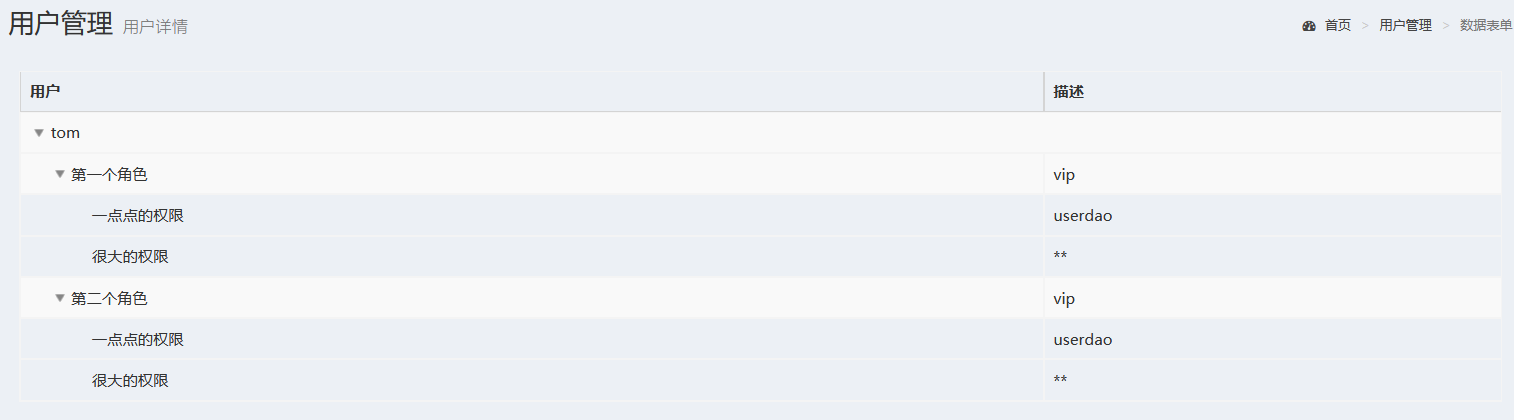


上面的三个dao共同完成了我们的userinfo数据封装。

最后返回json数据格式，键名为user：在我们的前端jsp页面中通过下面处理方式进行树结构展示：



最后页面结果展示：成功！

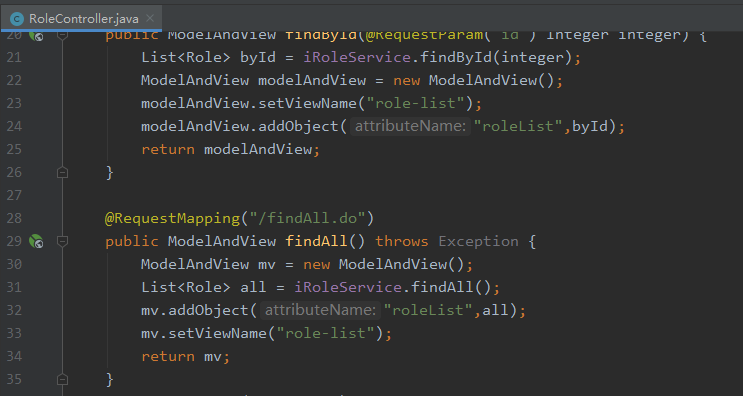


## 09：角色操作-查询所有角色代码：

先观察我们的页面跳转url：

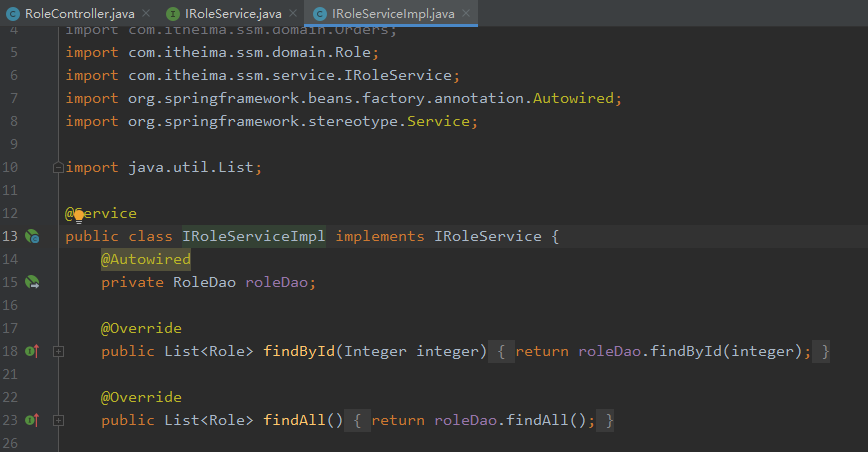


据此创建RoleController中的findall.do方法：

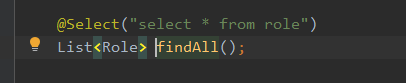


以上调用了RoleService中的findAll方法：

我们继续创建：



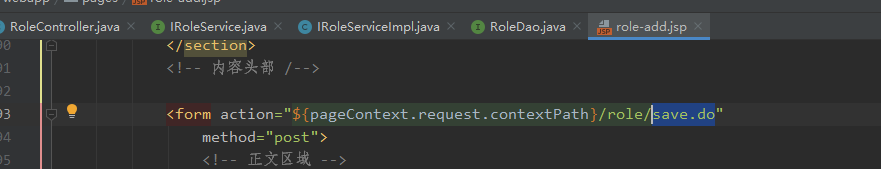
这里又调用了RoleDao中的findAll方法，我们继续创建：



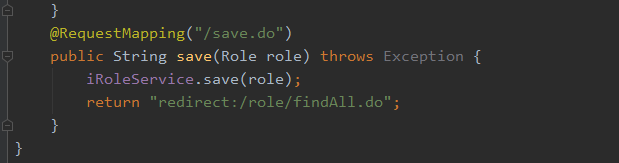
最后实现页面显示所有的角色！

## 10：角色操作-添加角色代码：

首先观察页面跳转路径：



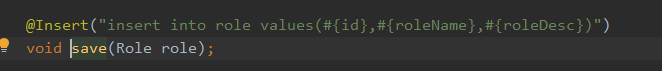
据此我们创建对应的方法：



创建service业务层save：



创建dao持久层save：



## 13：资源权限操作-查询所有的资源权限代码：

同角色查询！

## 14：资源权限操作-添加资源权限代码：

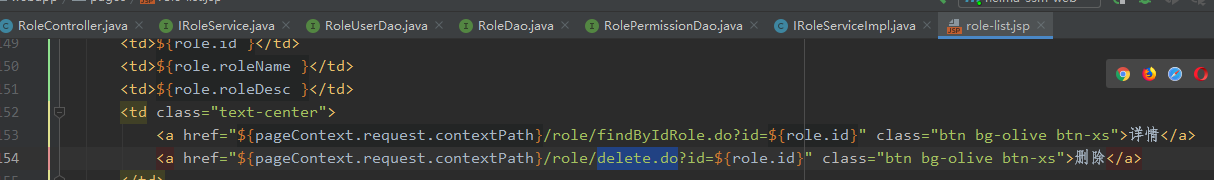
同角色添加！

## 15：作业之角色详情查询：

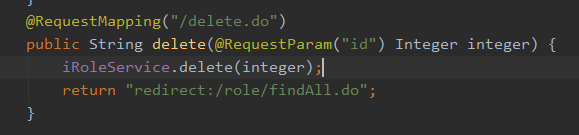
同用户详情查询，相比反而更简单！

## 16：作业之角色删除：

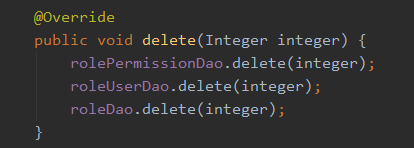
首先观察页面跳转路径：



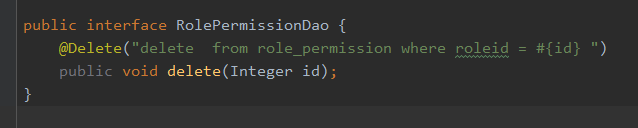
在controller创建我们的delete方法：



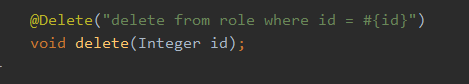
在roleservice业务层创建对应的删除方法：这里需要注意的是，并不是简简单单的删除role就可以实现功能的，因为这里包含了两个外键关联，我们需要首先删除外键关联表中的数据，才能对我们的role中的数据进行操作！



下面分别在对应的dao持久层中进行删除！



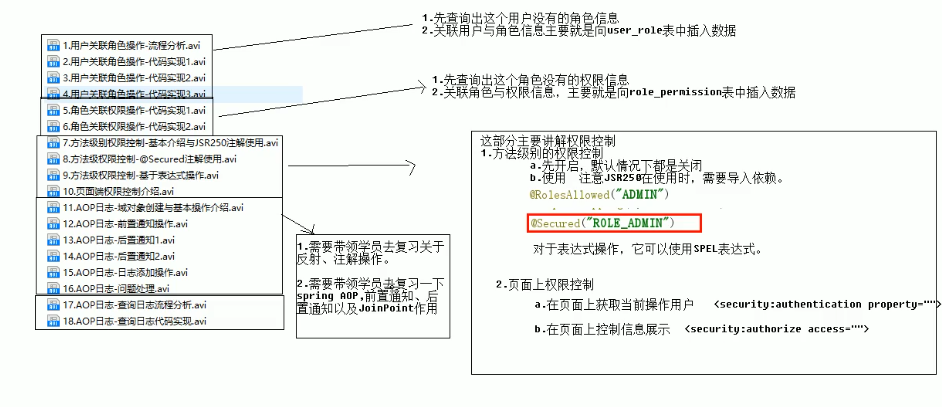




## 17：作业之资源权限详情查询以及删除操作：

同上，也是反而更简单，因为关联的表减少了。

# Day05：课程介绍以及难点分析

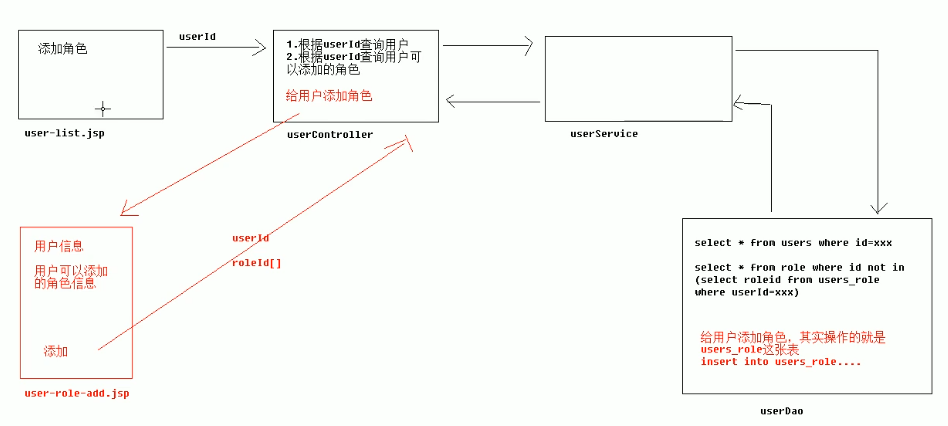


以上就是我们今日的内容，这里的aop日志是难点。

## 01：用户关联角色操作-流程分析：

大概流程就是：首先在user-list页面点击添加角色：

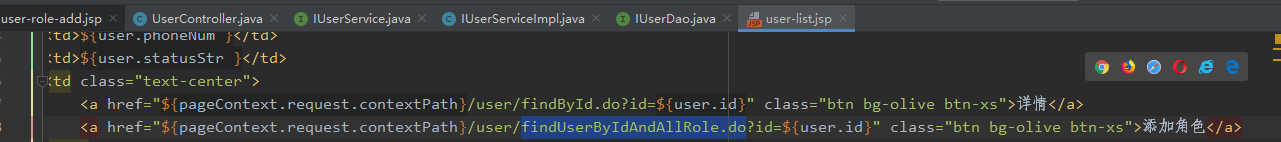
Controller表现层就会自动调用dao层实现当前user以及可以添加的角色信息，将这些信息再返回user-role-add页面，在进行添加操作，我们controller又会自动调用dao对user-role这个多对多中间表进行添加，最后就实现了用户添加角色的操作!



## 02：用户关联角色操作-代码实现：

首先观察前端页面跳转路径：

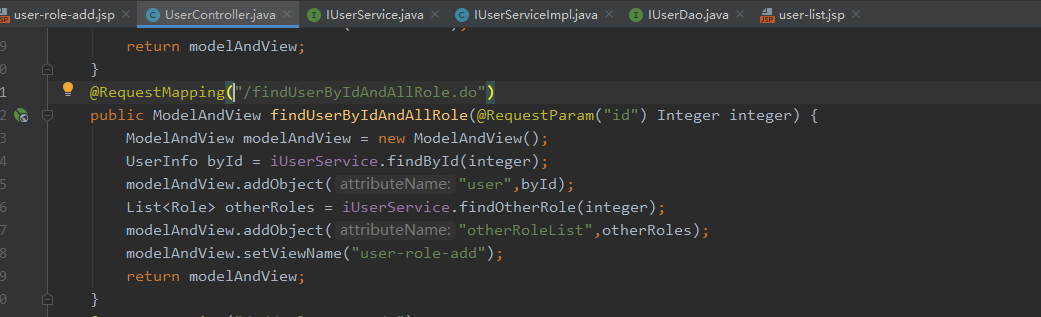
这里点击添加角色直接跳转到了controller中



在usercontroller添加对应的处理方法：

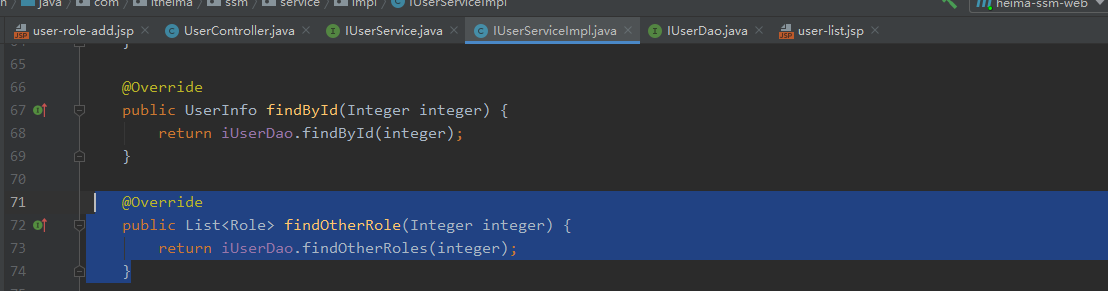
这里进行了查询：

主要是查询当前用户还可以添加的角色列表，最后将用户信息，以及用户仍然可以添加的角色都查询出来进行页面返回！



由于上面调用了findOtherRole这个新方法：

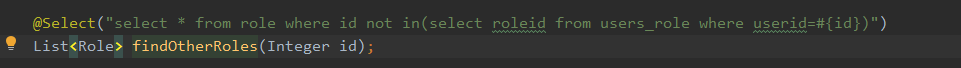
我们在userservice中创建这个方法：



这里又进一步调用了userdao层中的findOtherRole方法：

我们继续创建：

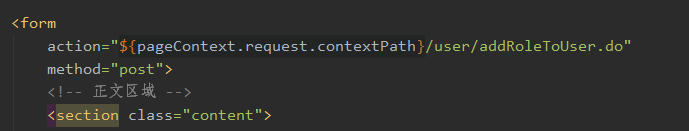
这里采用了查询的放式not in保证了查询到的role都是未添加的role！



最后回显到页面user-role-add.jsp页面

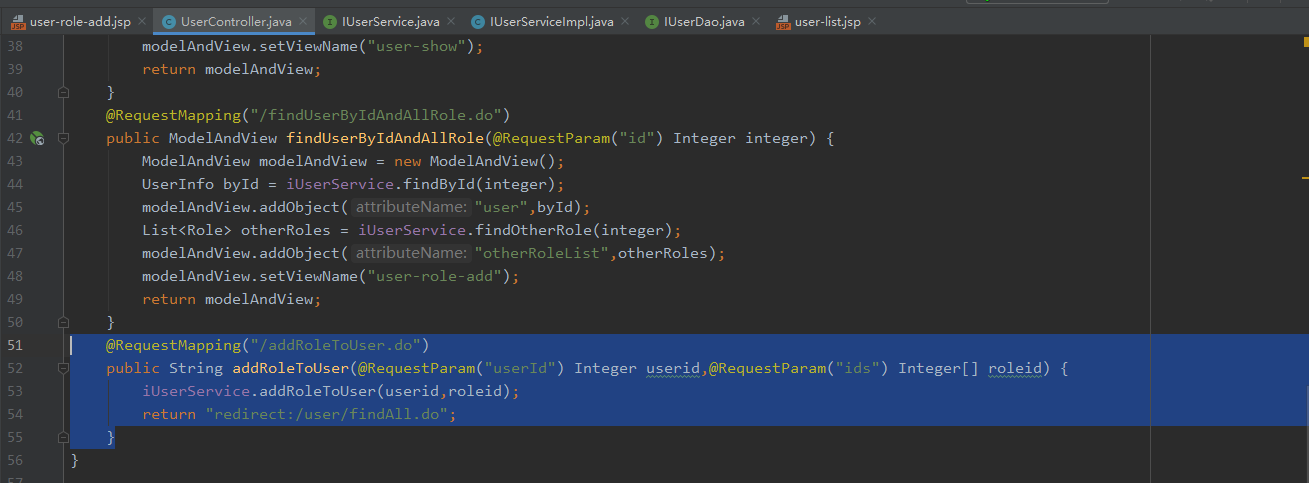
我们继续观察页面路径的跳转：

表单提交路径跳转到了usercontroller表现层中：



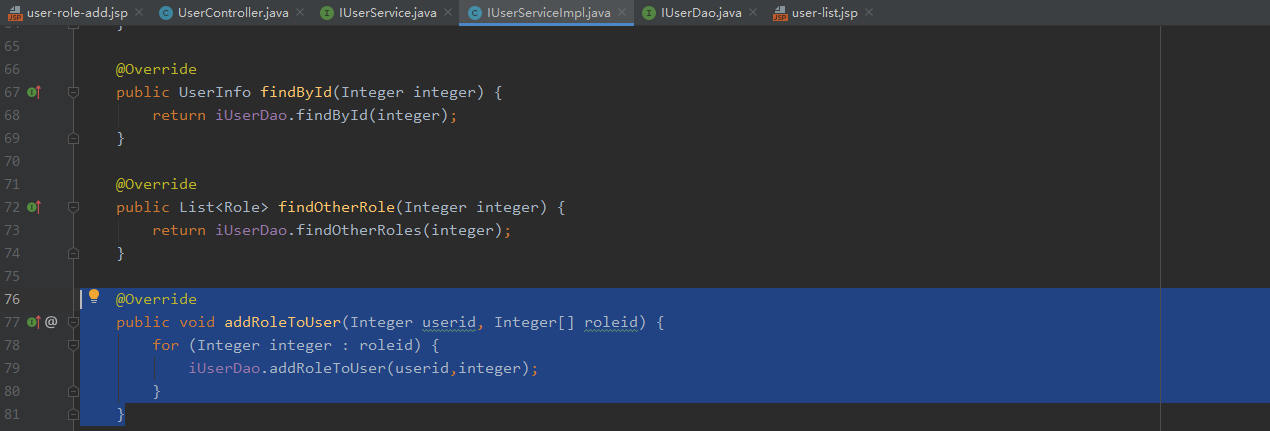
我们在usercontroller层中进行方法的创建：

如下：方法中获取了两个参数，一个是user的id属性，另一个是我们选中要添加的角色的id属性，由于我们可以多选，所以这里的id采用的是一个数组的方式进行收集！



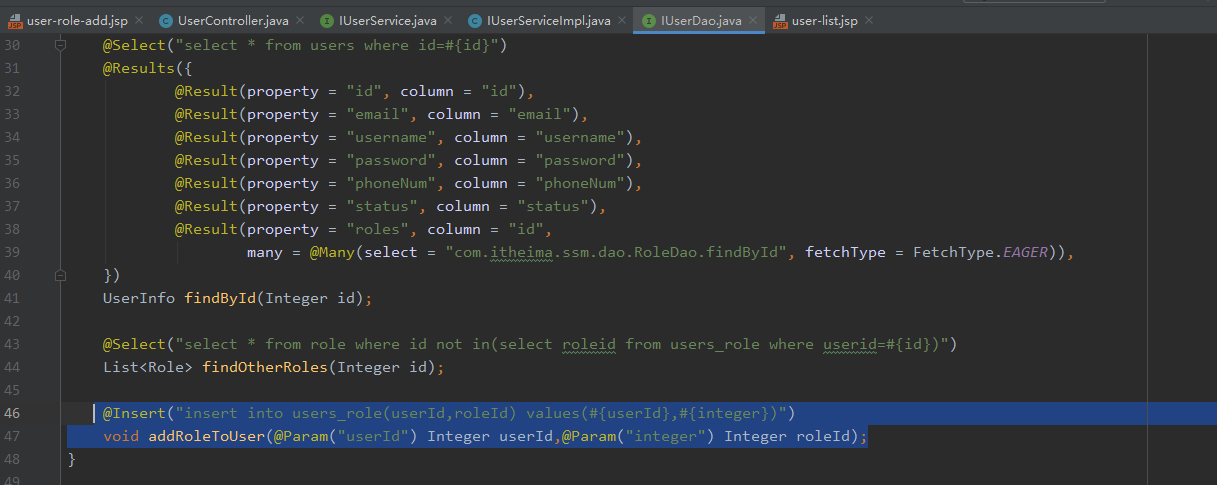
这里调用了userservice中的方法：我们进行创建：

由于传入的是一个数组，而dao层一次只能添加一个，所以这里我们设计一个循环将我们的数组中的id依次调用dao进行添加！



上面涉及到了dao的添加操作：

我们继续创建：



这里需要千万注意的是我们在方法的参数前面添加了@param注解，这是因为这里涉及到了多个参数，并且都是int类型的，mybatis不知道该如何进行匹配了（事实上她可不是按照前后顺序进行匹配的！），所以我们要在每一个参数的前面都添加上@param注解，指定我们的参数与sql语句中的什么参数进行匹配！

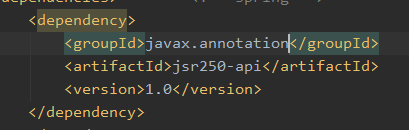
最后重定向到查询所有的用户界面，实现关联的功能！

## 05：角色关联权限操作-代码实现：

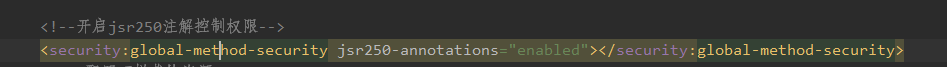
原理类似上面的关联，role与permission存在多对多的关系。本质上一样的！

## 07：方法级别的权限控制-基本介绍JSR250注解的使用：

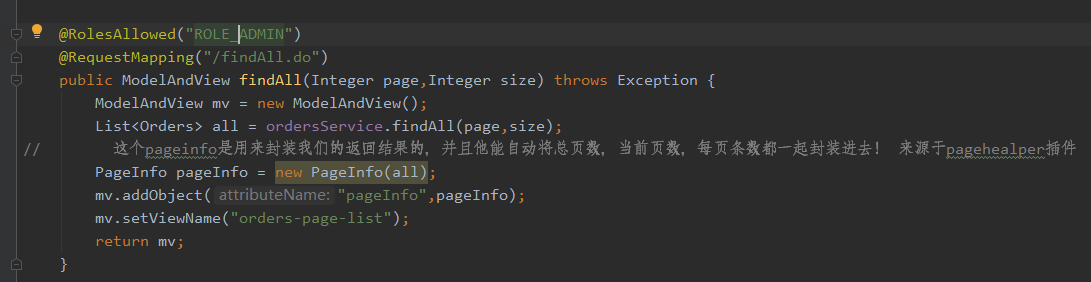
首先在pom.xml文件中导入所需要的依赖：



在spring-security中配置开启jsr250权限控制（默认是关闭的）

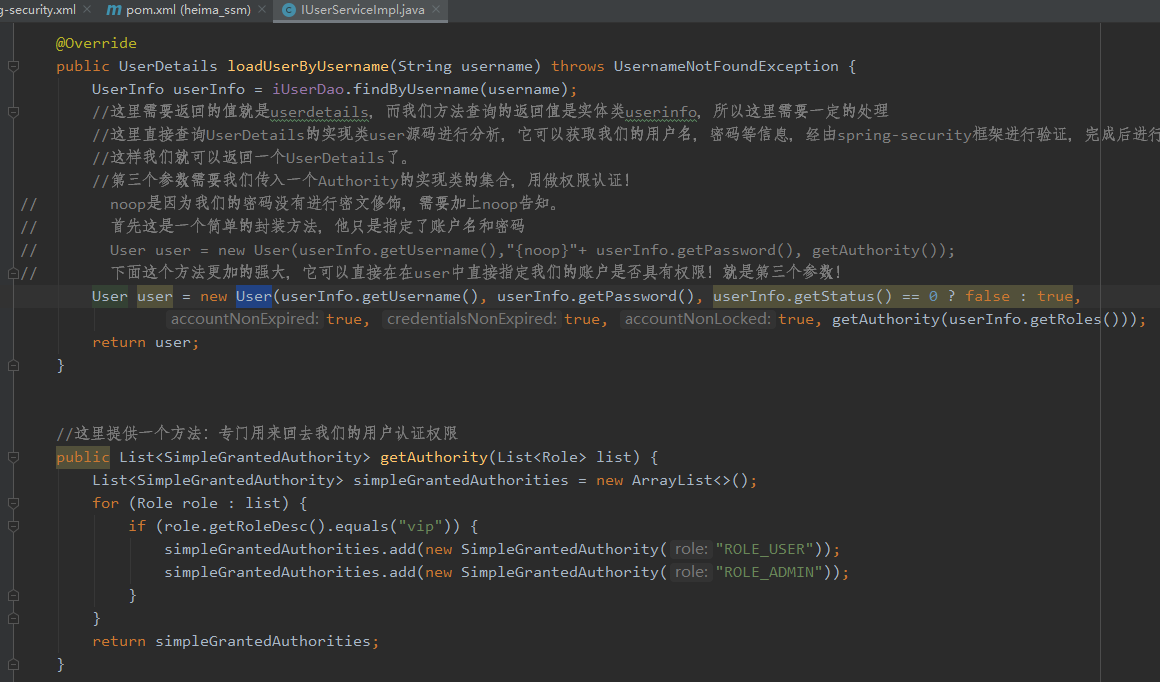


在我们需要加权限的方法头上添加@RoleAllows注解：



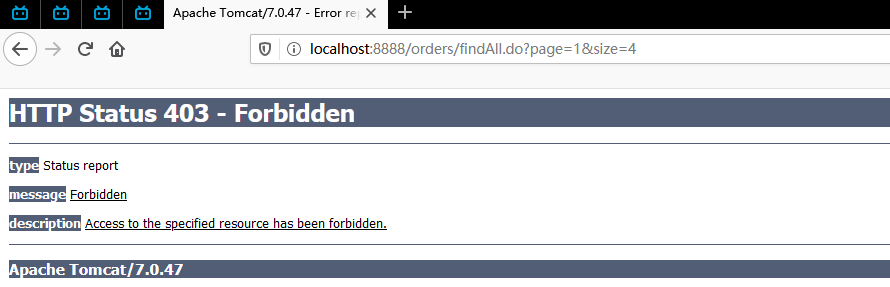
权限的添加我们是在登录检验中完成的，一旦用户的描述是vip，我们就为他添加两个权限：

User以及admin。

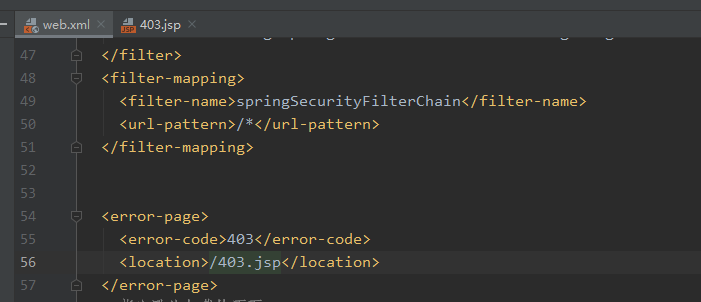


此时仅有vip描述的用户才能够查看订单！

Newtom账号就没有权限！

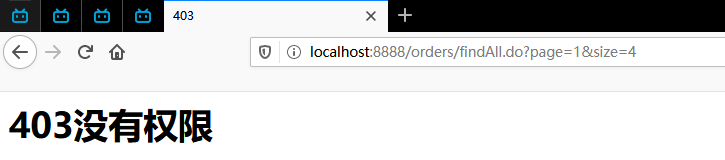


但是这里的页面不好看，我们可以自定义403状态码的页面！



这里指定了403状态码跳转类路径下的403.jsp页面！

效果展示：

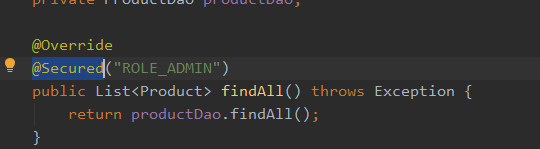


## 08：方法级别的权限控制-@Secured注解的使用：

首先在spring-security.xml配置文件中开启注解：



在需要加权限的方法头上添加@secured注解：



## 09：方法级别的权限控制-基于表达式使用：

首先在spring-security.xml文件中开启表达式权限控制：



开启以后共有四个注解可以使用，我们讲解最常用的一个：

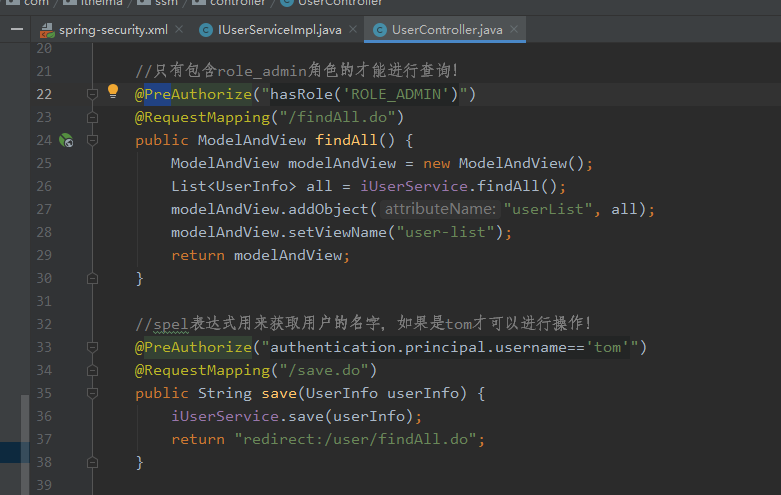
@PreAuthorize注解：

第一个hasRole指定了只有包含role\_admin角色的用户才能够访问查询所有用户！

第二个指定了当前登录用户的用户名必须是tom才可以进行添加用户保存的操作！

还有很多其他的用法！

总之就是这种方式他强大在于他支持spel表达式！所以很灵活。



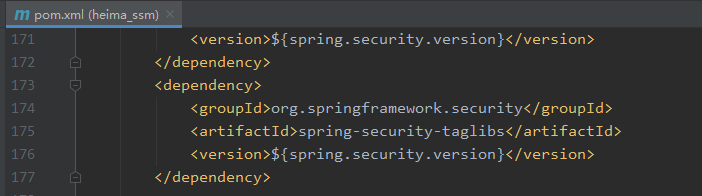
Web实测发现只有tom用户才能够完成用户添加操作，其他用户即使可以查看，删除等操作，也是不可以完成保存操作的！

## 10：页面端权限控制介绍：

使用步骤：



第一步：导入maven坐标：



第二步：jsp页面导入标签：

这里的第二个标签就是我们导入的页面权限控制！



第三步：使用（获取当前登录的用户）：

如下使用就可以获取我们当前登录用户的一些内容！



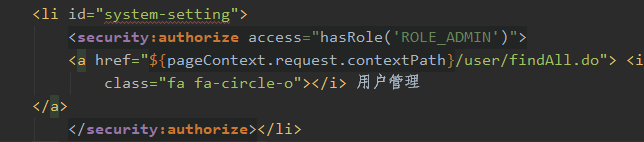
最后页面显示：



第三步：使用（指定显示）：

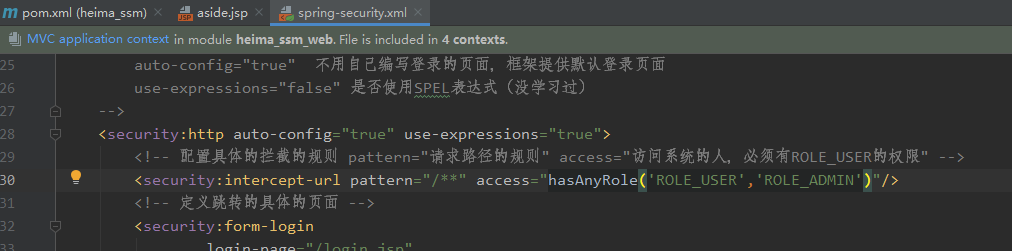
如下使用就可以指定我们的标签是否显示：

只有当我们当前角色具有role\_admin的时候才能够显示用户管理标签！

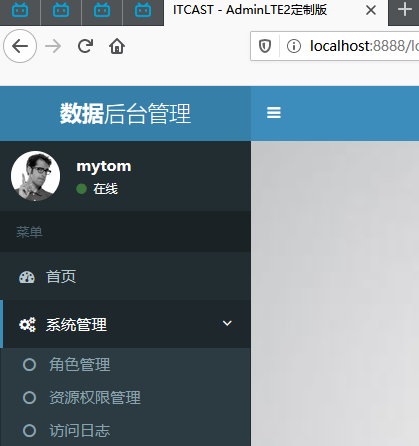


这里使用了spel表达式，所以我们需要打开支持spel表达式的开关：

配置文件中做出如下修改：

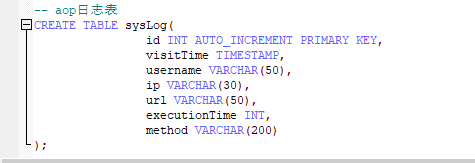


最后这里的mytom用户以及没有权利查看用户管理标签了！

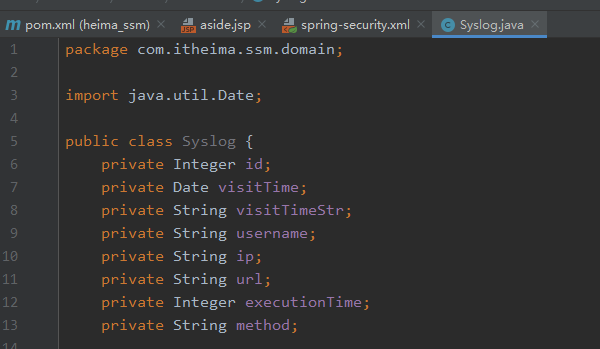


## 11：aop日志记录访问信息介绍：

首先创建表：用来插入或者查询访问的信息！



在domain包下面创建用来封装数据库参数的实体类：



## 12：aop日志配置前置通知：

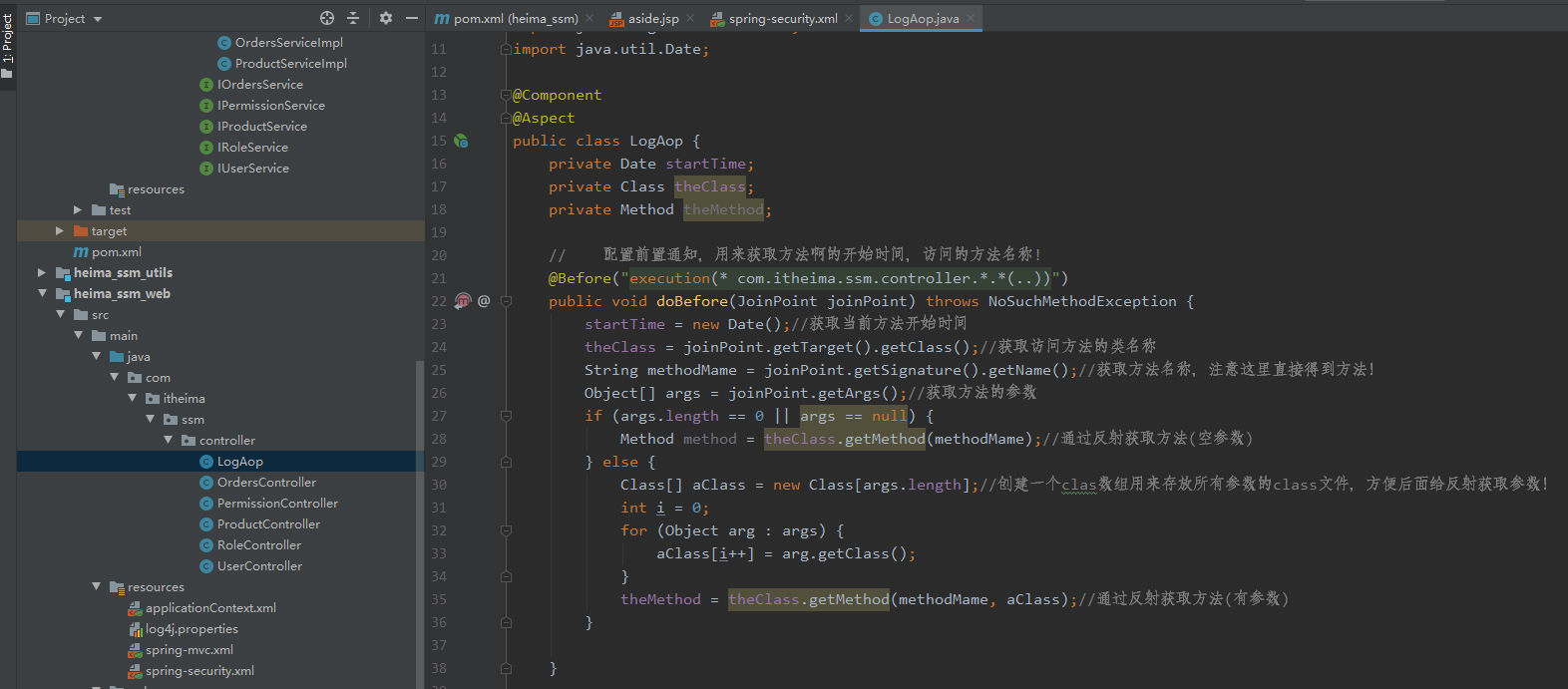
在前置通知中我们可以获取方法运行的开始时间，调用的方法，类的名称！

这里指定@component注解，代表他是我们自定义的类交给spring托管！

@Aspect注解表示这是一个切面类

@Before注解表示，这里的通知类型是前置通知，就是在他参数内指定的方法运行之前要运行的方法！

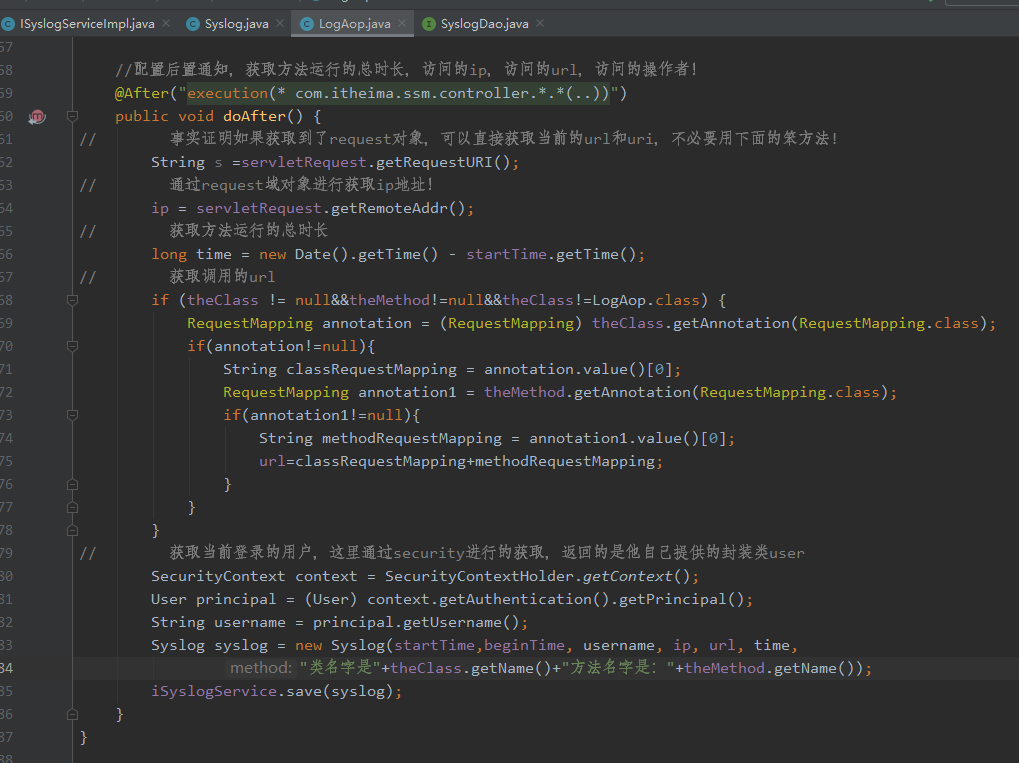
在前置方法中我们使用了joinpoint参数来获取切点的类以及方法。同时涉及到一点点反射的知识。这里获取对应参数的class数组的时候，要求参数是对象不能是int等基本类型，因为他没有.class文件！



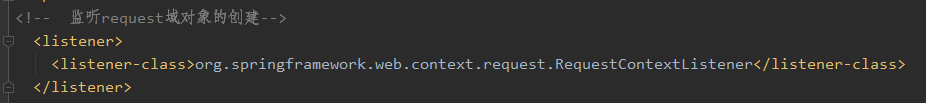
## 13：aop日志配置后置通知：

在后置通知配置中完成运行时间的计算，url和ip地址的获取（因为这里获取需要用到request对象，所以我们在web.xml文件中配置了一个request的监听器，一旦检测到controller中的方法运行，自动生成request对象！），这里获取url采用了两种方式，都可!

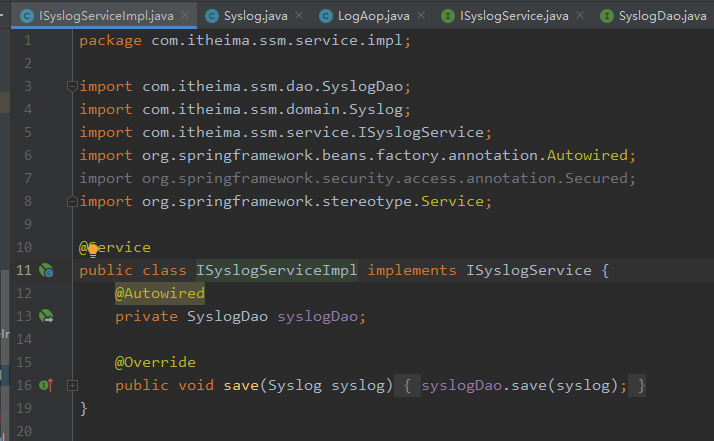
这里获取当前登录用户方法比较麻烦：要通过security框架获取到我们前面使用的user封装类对象！最后将我们所有参数封装到实体类中供dao层插入数据库使用！



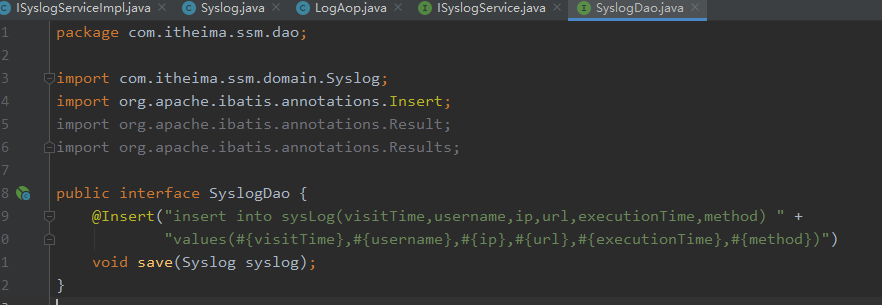
这里就是配置监听器生成request对象！



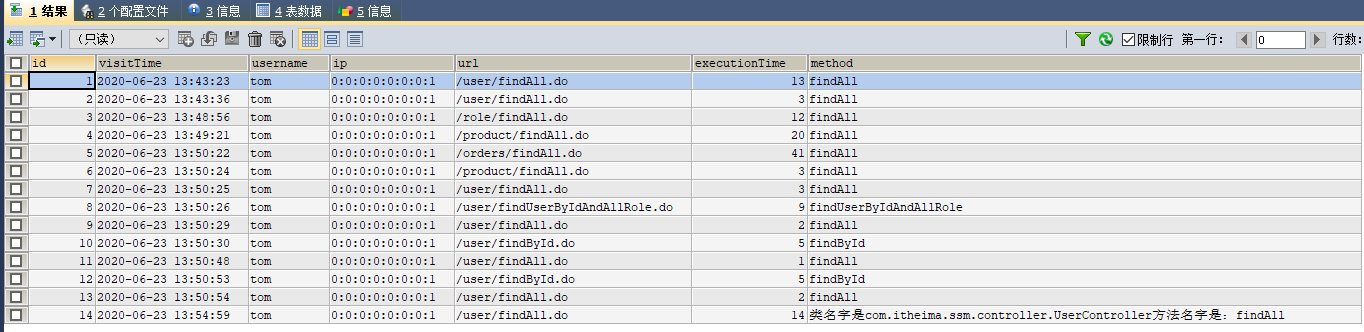
后置通知最后调用了SyslogServicesave的save方法，我们进行创建：



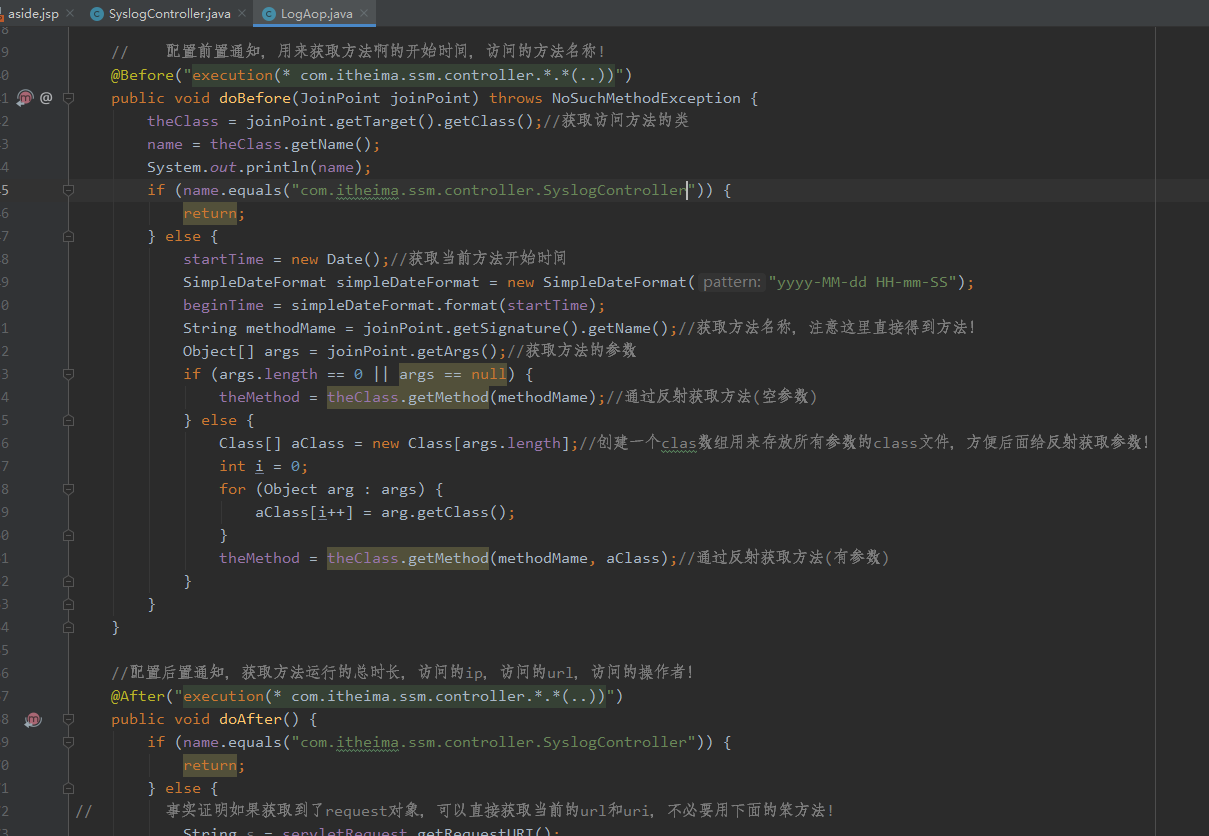
这里又调用了SyslogDao层中的save方法，我们进行创建：



最后结果显示数据库中封装了我们的操作信息！



但是这里会将我们点击访问日志的操作也记录进去，没有实际意义，所以我们在aop切面中进行方法的筛选：我们将获取到的类名称为syslogcontroller的所有请求都过滤掉：



最后这里访问日志查询就不会进行记录了！

