# Maven高级

## 二： maven基础知识回顾

Maven有两大功能：

一个是坐标依赖，帮我们简化了jar包的导入。（这个我们必须遵守）

一个时一键构建，内部集成tomcat，会帮我们编译，清除，打包等。（这个我们采用本地服务器实现也可以）

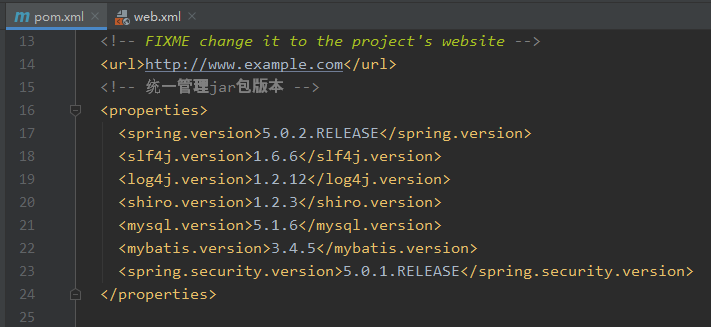
Maven仓库：本地仓库，远程仓库（私服），中央仓库。本地没有找私服，私服没有找远程或者我们进行本地上传。

Maven常用命令：clean，compile，test，package，install，deploy（上传本地的jar包）

Maven生命周期：清理生命周期，默认生命周期，站点生命周期。

## 三： 案例环境搭建

创建一个maven工程导入所有用到的jar包坐标。



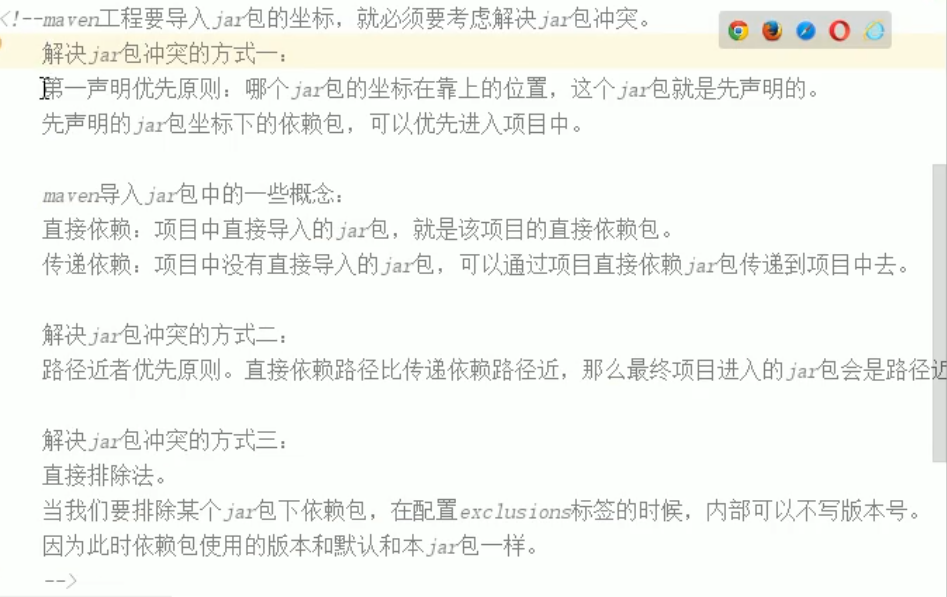
## 四： maven导入jar包的时候版本冲突问题

我们导入的maven坐标内部又会依赖于多个jar包，他们都会被导入进来，当公用一个jar依赖的时候，导入的版本就会不同，那么到底哪个会被采用呢？又要怎么解决呢？

这里提供了三种解决的方式：

1. jar包的依赖方式相同的时候，上面的先进入到项目中。
2. jar包的依赖方式不同的时候，直接依赖进入到项目中。
3. 我们也可在导入maven坐标的时候将某些固定的依赖剔除。他自然就不会影响我们的jar包引入了。

实际开发中我们采用第三种方式更多，因为大多数时候我们会直接把现成的ssm依赖导入，如果某些jar包需要指定版本，我们就将原来的jar包剔除，在最后补充上我们需要的版本jar包。

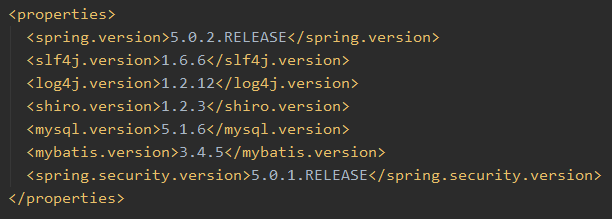


## 五： pom文件内部标签的解读

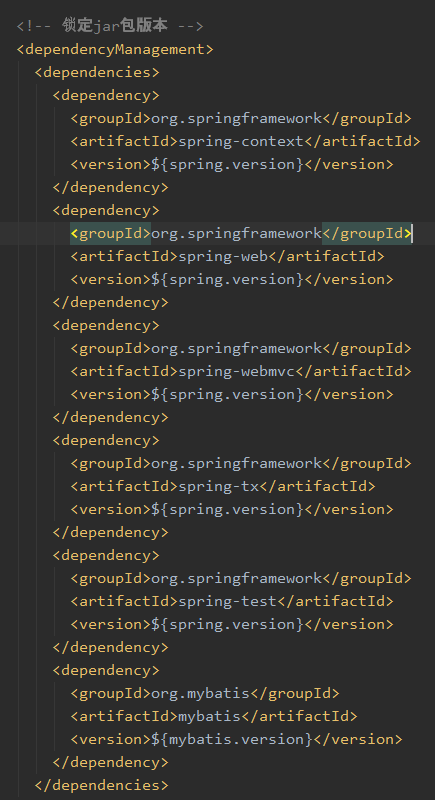
首先就是版本锁定：

子maven工程继承父工程的时候，会覆盖父工程的maven坐标，这时候可能会导致父工程无法正常运行，所以出现了版本锁定。一方面防止被子工程覆盖，另一方面也便于以后修改版本！

Properties本质上就时一个map而已，标签名字就是键，标签内容就是值，只不过为了可读性，我们取名为spring.verson等！这里可以这样写是因为上面的约束条件！



然后再dependencyManagement标签内部锁定jar包版本，千万注意这里只负责锁定，不负责导入，所以导入版本依然需要<dependencies>标签来实现。其中的${}是我们xml文件的el表达式，无需多想！



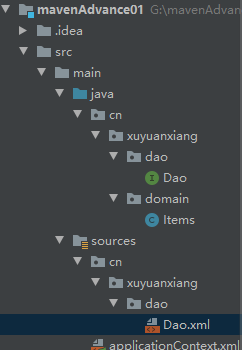
## 六： dao层整合

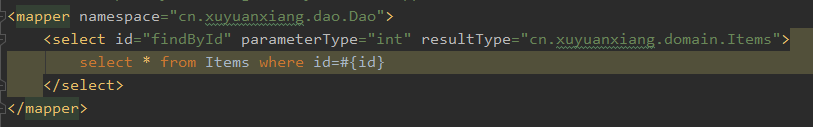
任何持久层框架都会为我们提供一个操作数据库的对象，mybatis用的就是sqlsession代理对象，所以我们要在spring配置文件中配置一个sqlsession对象，而这个对象又是多例的，线程不安全，所以又得为他配置一个工厂，这就又引出了sqlsessionfactoryBean对象的配置。而创建sqlsessionfactoryBean当然需要我们的连接池喽，所以才会有上面配置的数据库连接池！applicationContext.xml配置如下：



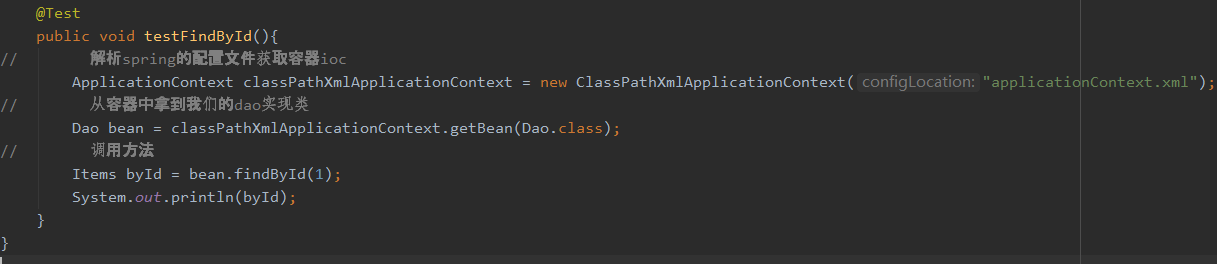
可以发现我们并没有在applicatioContext.xml中指定映射文件的路径！！！说明spring整合mybatis会自动解析！

映射文件配置：要求路径名与Dao层对仗:这样spring就会自动帮我们解析这个Dao.xml（值得注意的是这里的名字必须是Dao.xml，因为我们的接口名字是Dao，否则spring就不能自动解析！）





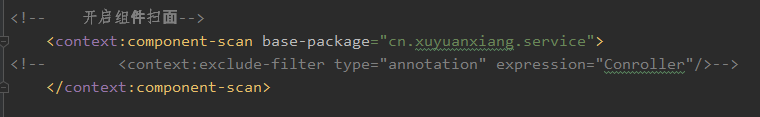
最后完成测试！



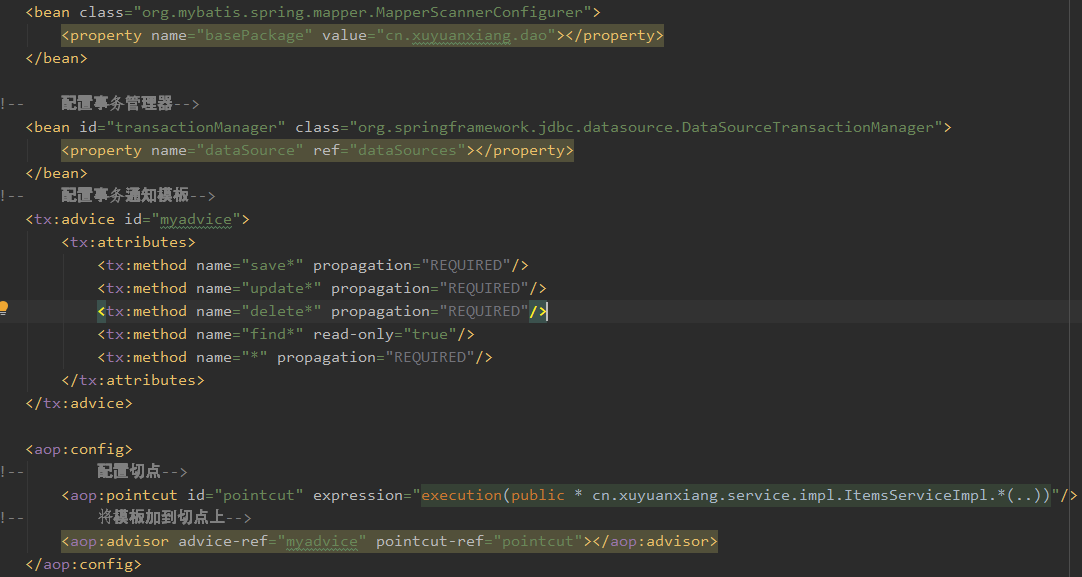
## 七： service层整合

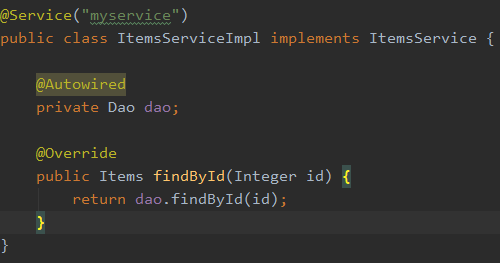
自己写的类尽量使用注解，框架的类使用配置文件。这是公司标准！

首先在applicationContext.xml文件中开启组件扫面，将我们的service包下面的组件扫描添加到容器中！

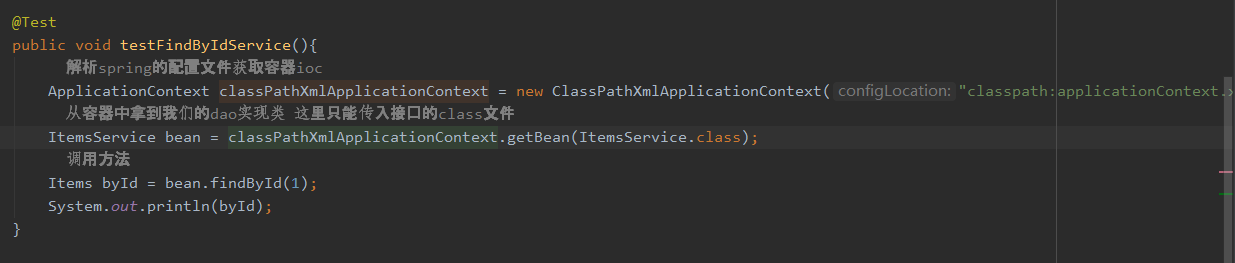


再applicationContext.xml文件中配置好事务管理器。



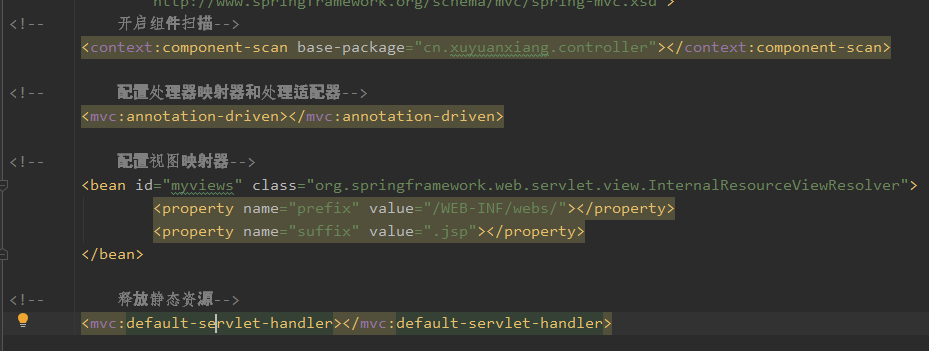


编写一个测试类进行测试（这里为啥必须传入一个接口的class文件，为啥传入“myservice”不可以了！）

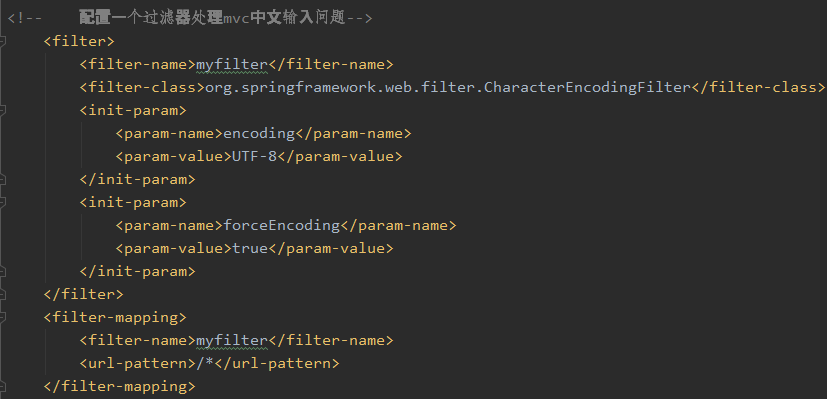


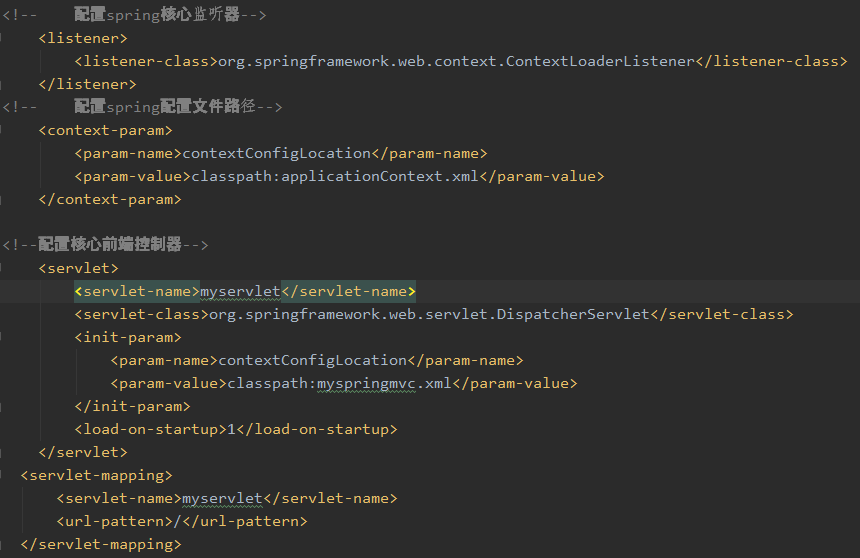
## 八： web层整合

首先创建一个springmvc的配置文件，配置好主要的组件（处理适配器，处理映射器，视图解析器），再释放静态资源。（这里的操作是将servlet采用httpservlet进行处理而不是dispatcherservlet，自然不会过滤静态资源！）

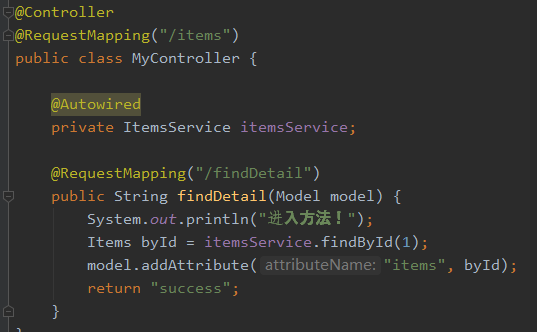


再web.xml文件中配置servlet容器（tomcat）三大组件

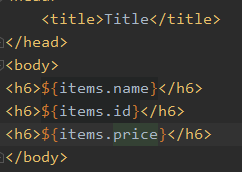




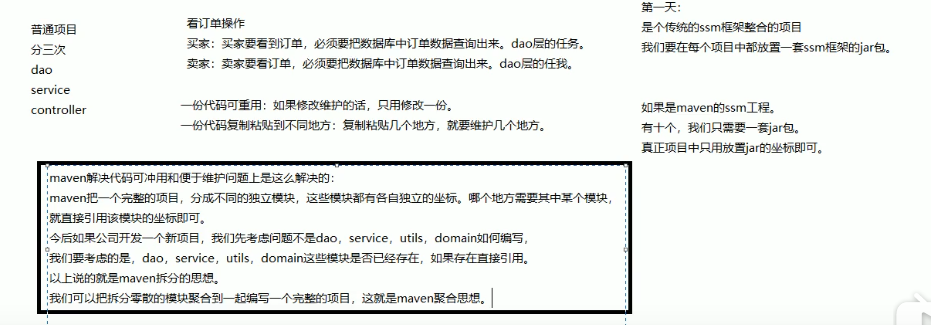
Mycontroller类实现前端功能



页面回显！值得注意的是页面回显使用的是jsp的el表达式，并不能支持lombok包下的注解(@data,@NoArgsConstuctor)



## 九： maven拆分和聚合的思想

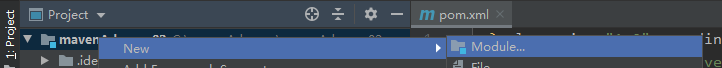


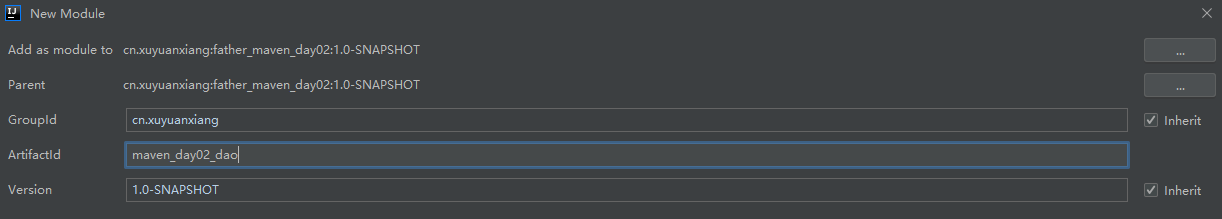
我们的maven工程里有dao，service，controller，utils等等功能模块，如果再来一个功能类似的maven工程岂不是又要写一遍？

实际上大可不必，因为maven早就有解决这个问题，我们写的dao，service，等等都可以当作一个模块，需要的时候就调用，大家共用就行了，没必要每一个都写一份（那样岂不是和以前的javaee一样了！）

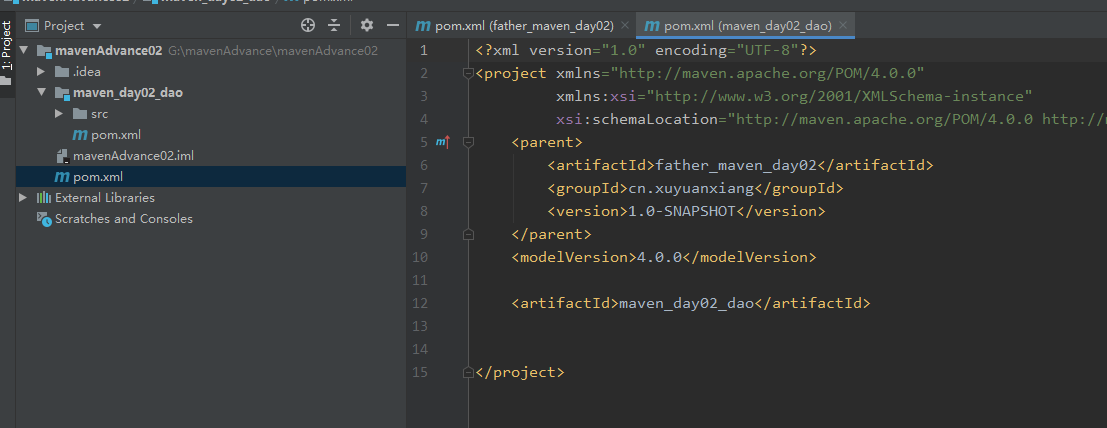
## 十： maven父子工程创建

如下创建父工程和模块！父工程其实只负责jar包管理，所以内部甚至代码都不用写！

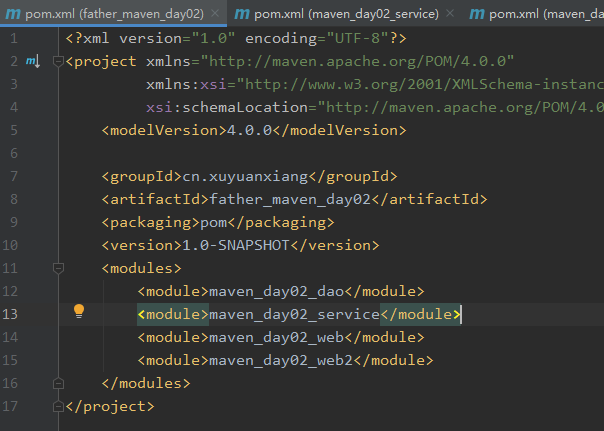




子模块中会出现parent标签



父工程pom文件里会出现modules标签把子模块包含进来！



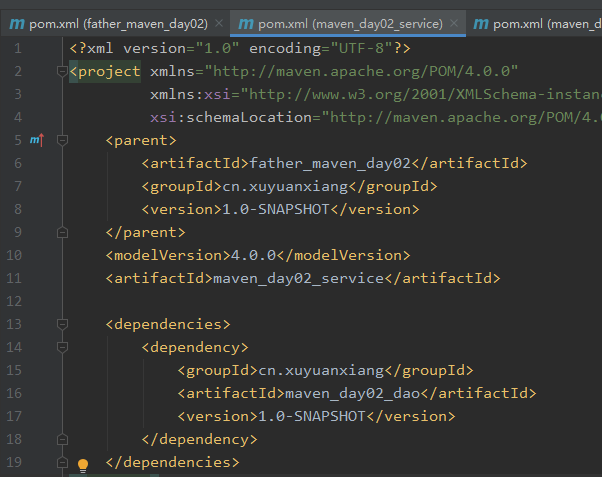
## 十一： 工程和模块的关系

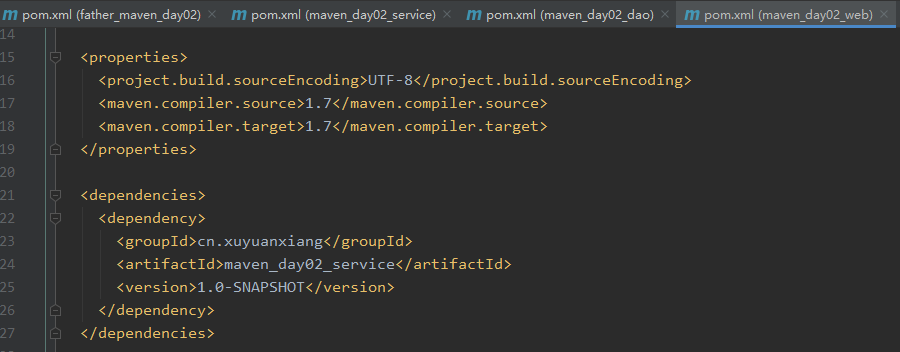
工程和模块没有大小区分，二者都不能称之为完整的项目！完整的项目是看代码能否实现指定功能的。工程天生只能使用自己内部资源，天生独立，可以和其他模块或者工程建立练习。

模块天生不独立，他需要寄生在父工程上，一旦模块建立，他就可以使用父工程的所有资源。

子模块之间天生没有什么关系，但是我们可以建立关系，例如service调用dao。

建立关系的方式就是在pom文件中导入器坐标而已。例如：（也可以与其他工程的模块建立联系！）

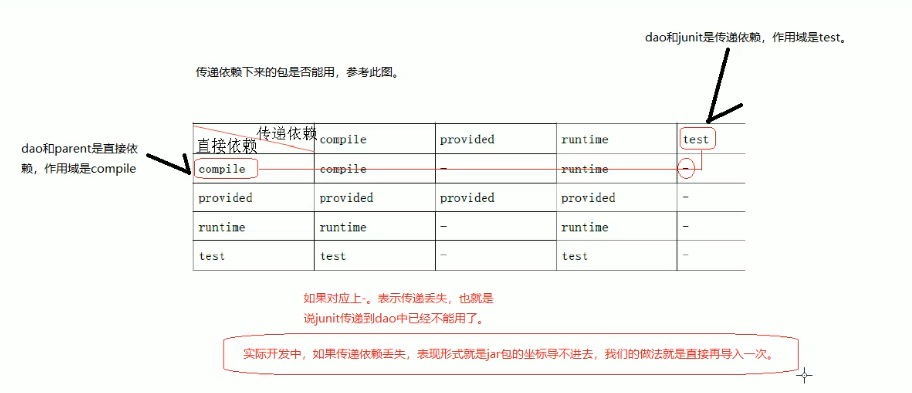




父子工程是不用建立关系的，子模块想用父工程资源

平级之间的引用称之为依赖~！这个是需要后天建立的，例如上面。

## 十二： 依赖关系的jar包是否可用？

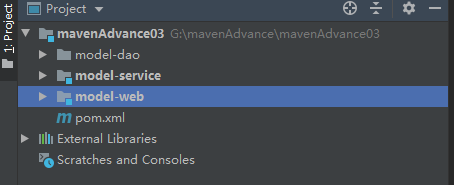
依赖的时候不写作用域默认是compile

## 十三： 父子工程代码填充

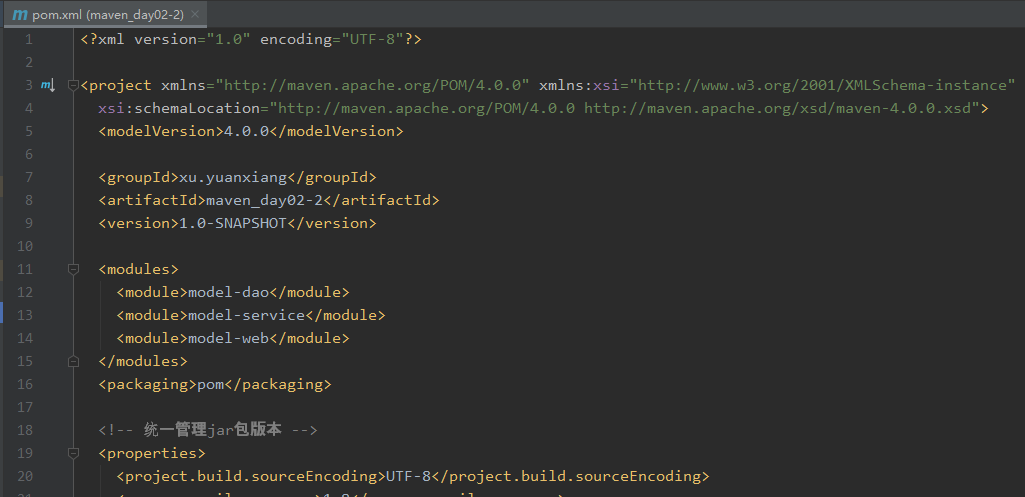
首先创建一个根项目：用来做pom中maven的坐标导入以及版本锁定。

分别在根项目下面创建三个模块，用来实现dao，service以及controller。

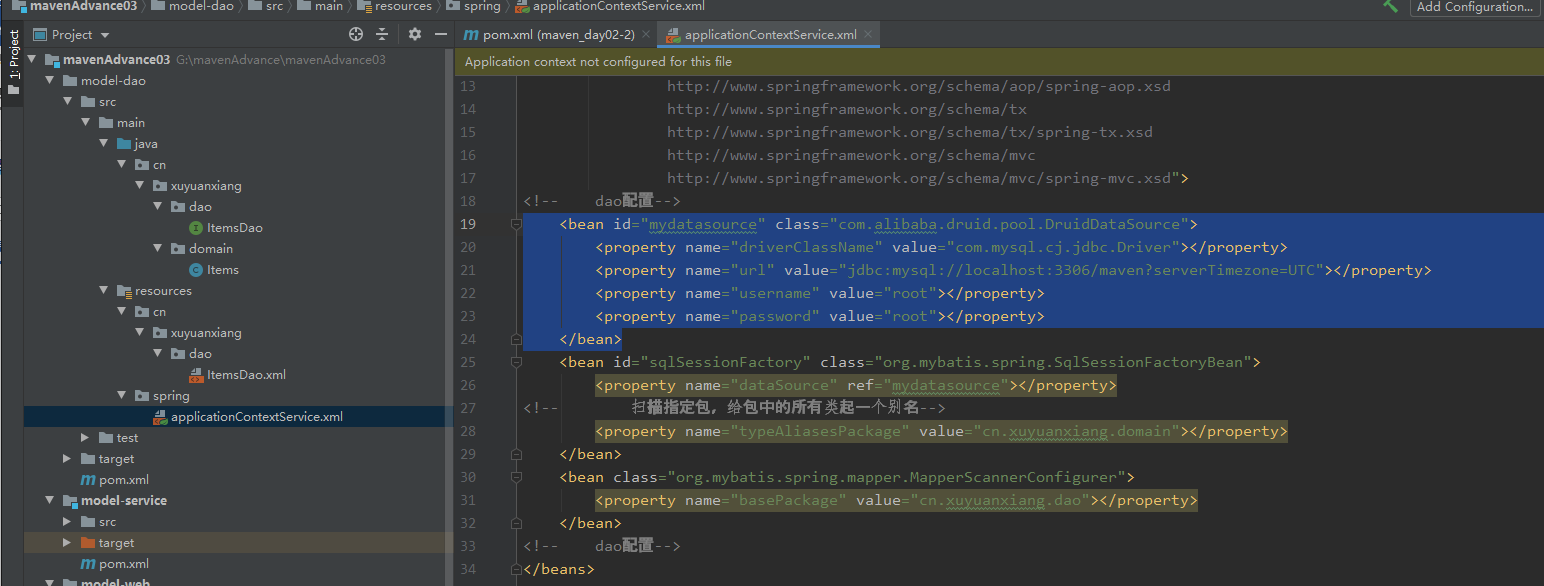
在根项目的pom文件中导入坐标：



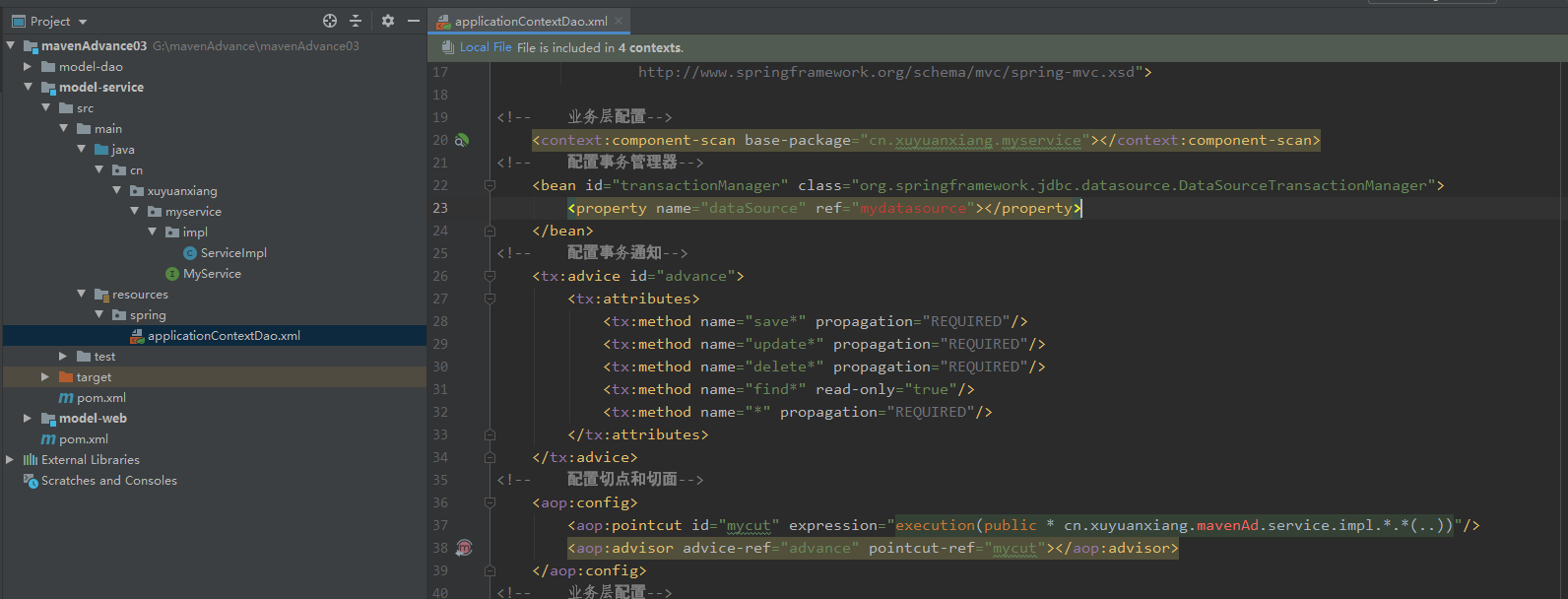
在根项目的pom文件中导入坐标：



完成dao模块：我们这里只需要关注dao层的内容就行了，无需做什么事务配置等等，只需要实现dao接口的代理。

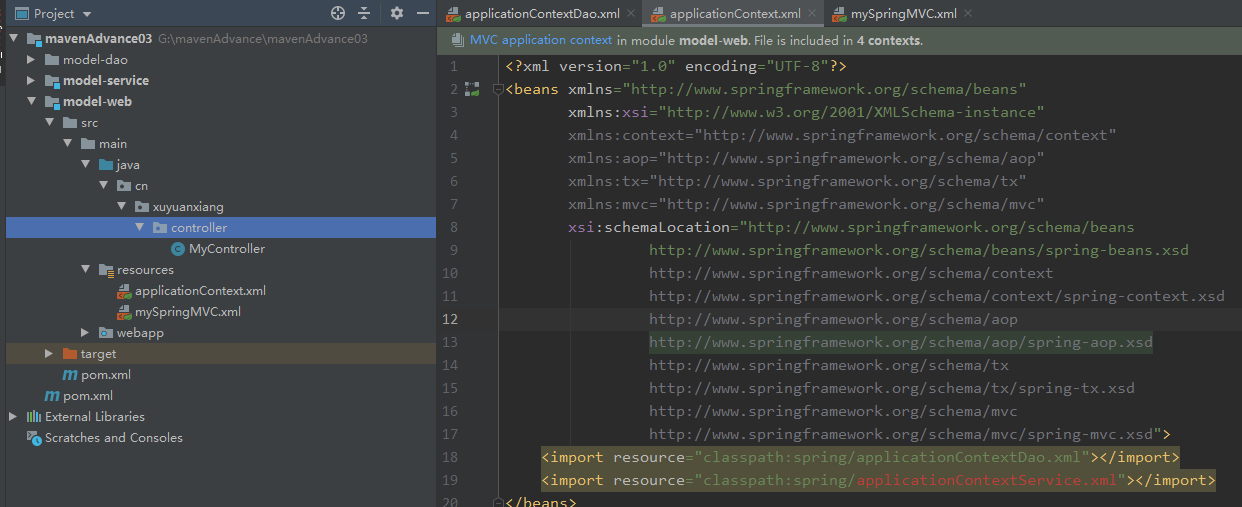


完成service模块，只需要关注service就行了，什么mybatis以及封装数据的类都不需要了，我们只需要在pom中引入dao模块的坐标，所有的dao模块内容都会对应的进入到我们的service模块中，即使我们这里会爆红，也没有问题。

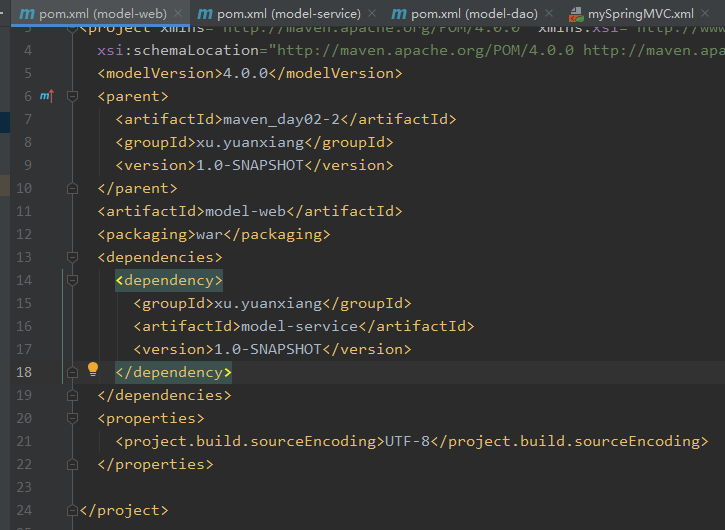
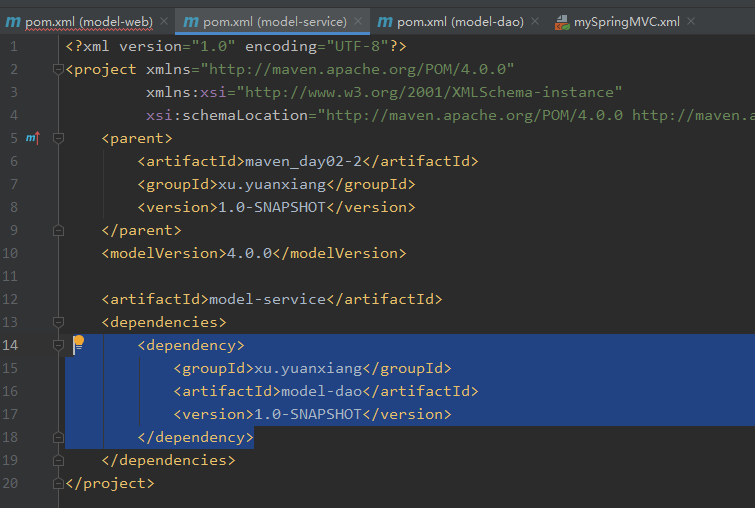


最后完成controller模块的编写，这里需要注意的是我们既然扫描的是类路径下的applicationContext.xml文件，这里就需要创建这个文件，同时需要引入前两个模块的applicationContext.xml文件：即使爆红也是对的（爆红是因为我们的service没有进行install，也就没有把自身的模块注册到本地中去，当然找不到对应的路径了，我们手动install之后九没事情了。）！

不要以为service引入了dao，这里只引入service模块就可以了。实践证明不可以！

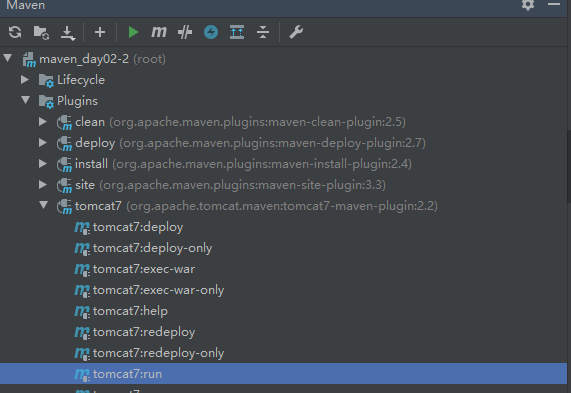


最后补上web模块引入service模块，service模块引入dao模块的pom信息：

## 十四： 父子工程的三种启动方式

第一种：root根目录下的tomcat7：run启动！



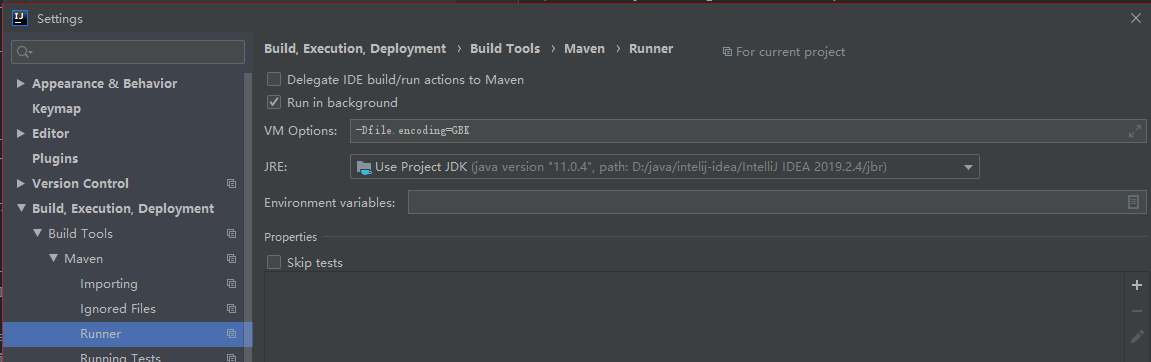
Bug满天飞：

第一个：中文乱码：

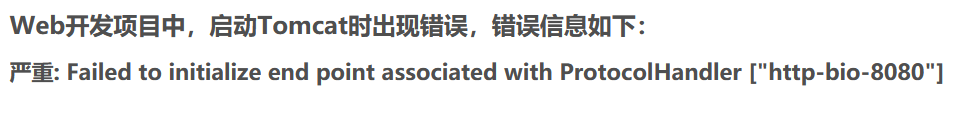


解决方案：（因为maven默认是gbk编码格式）

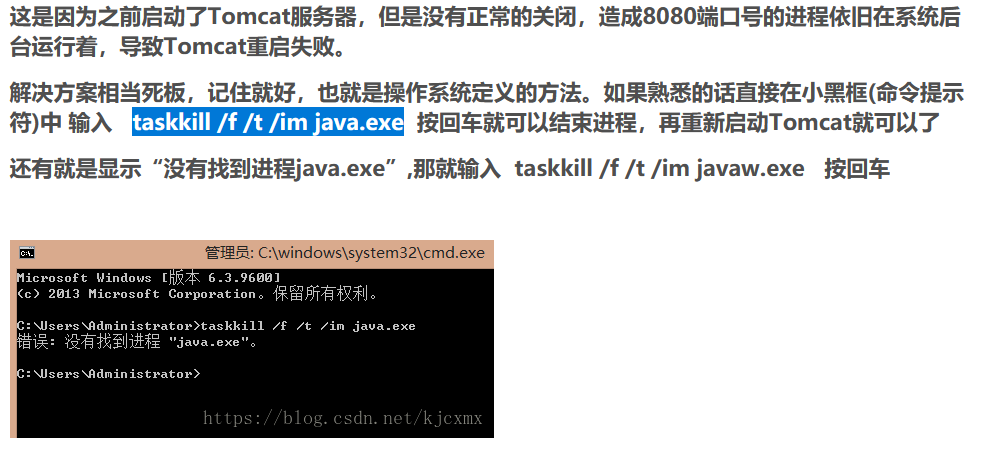
添加编码后发现上面的信息仅仅是不支持编译版本5！



第二个：



这里显示的端口号8080被占用！可能是我们一顿乱启动tomcat，后台依然存在服务器！解决方法很固定，黑窗口输入命令行：关闭所有java项目！



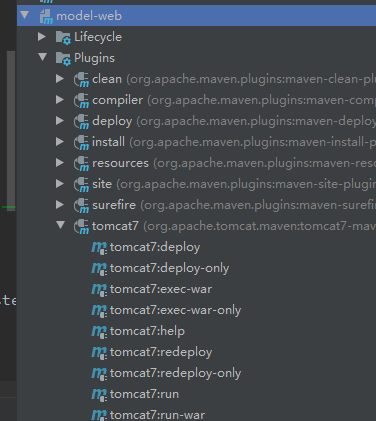
再次启动成功！

第二种：

在web模块中直接调用run命令。

这里需要注意的是我们的web引用了service和dao，所以service和dao都必须已经install才行。因为maven启动时先从本地找，没有install的话，本地没有，远程更没有，报错！

启动成功!



第三种：

以上两种都是maven采用的内置tomcat插件

第三种采用外置tomcat



也可以启动成功！