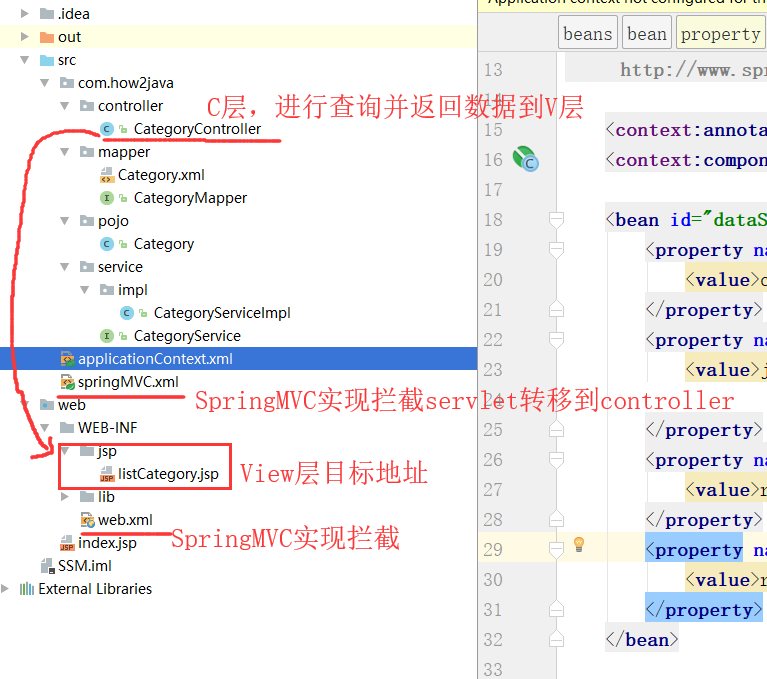
## 配置与架构

主要分为两个部分，

第一部分是SpringMVC自己，一部分是Spring+Mybatis

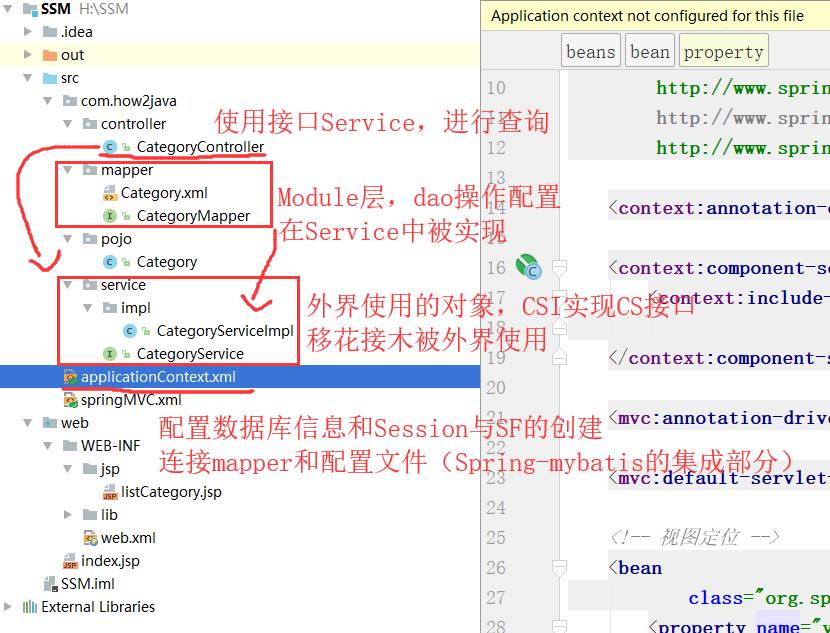
其中Controller部分是连接他们两的桥梁。

SpringMVC部分:



SpringMVC.xml文件中配置了拦截之后发送的Controller的地址和View层的视图定位。

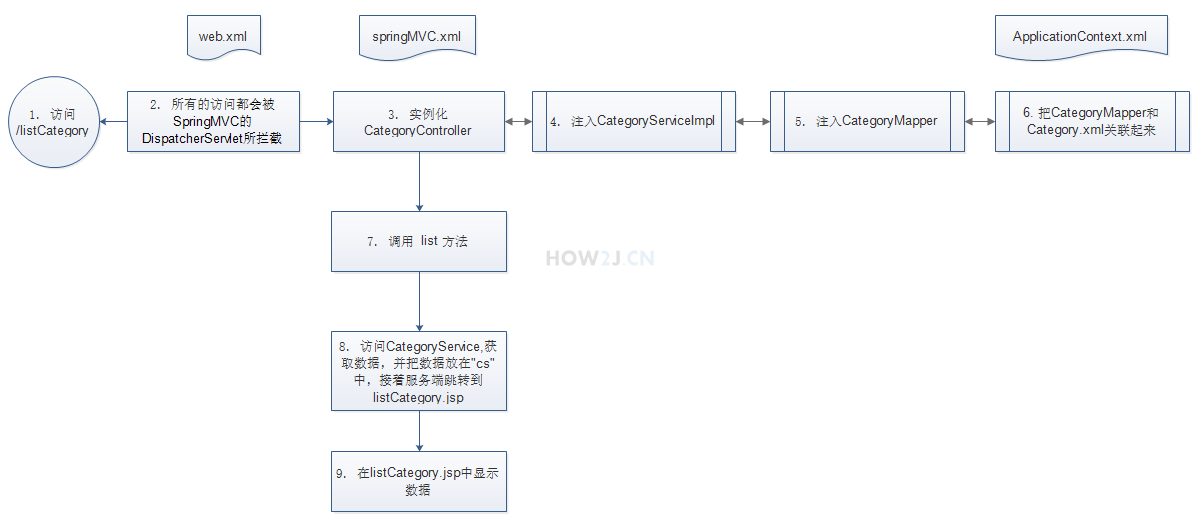
Spring+Mybatis部分:



架构差不多就是这样了

原文：

1. 首先浏览器上访问路径 /listCategory  
2. tomcat根据web.xml上的配置信息，拦截到了/listCategory，并将其交由DispatcherServlet处理。  
3. DispatcherServlet 根据springMVC的配置，将这次请求交由CategoryController类进行处理，所以需要进行这个类的实例化  
4. 在实例化CategoryController的时候，注入CategoryServiceImpl。 (自动装配实现了CategoryService接口的的实例，只有CategoryServiceImpl实现了CategoryService接口，所以就会注入CategoryServiceImpl)  
5. 在实例化CategoryServiceImpl的时候，又注入CategoryMapper  
6. 根据ApplicationContext.xml中的配置信息，将CategoryMapper和Category.xml关联起来了。  
7. 这样拿到了实例化好了的CategoryController,并调用 list 方法  
8. 在list方法中，访问CategoryService,并获取数据，并把数据放在"cs"上，接着服务端跳转到listCategory.jsp去  
9. 最后在listCategory.jsp 中显示数据



## 2.关于Service层：

Mapper可以直接操作数据库，也很方便，但是为了严谨起见，还是在Controller和Mapper之间加了一层Service接口去隔开直接操作，Controller只能通过给定的Service工具来进行dao操作。

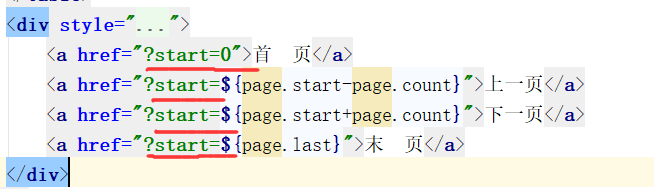
真正直接使用Mapper的是测试类(Test),只有做改动测试时才直接用mapper，在C层中只能用Service调用。

有了这一层后修改dao操作会很麻烦，要增加一个功能，比如下面的按page分页查list，就需要改4个文件。。。。。

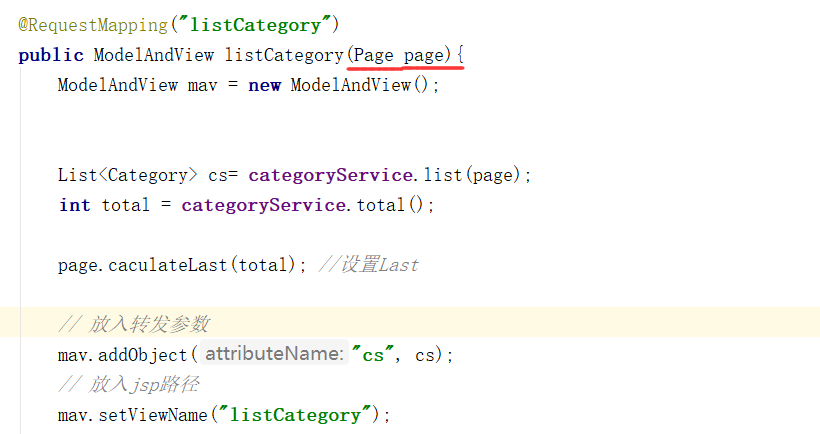
## 3.分页:

主要有一个知识点要注意：参数传入 Spring自动注入成对象

jsp页面  ulr链接地址有个start 它和page里面定义的属性 一致 spring mvc会自动帮我们 实例化

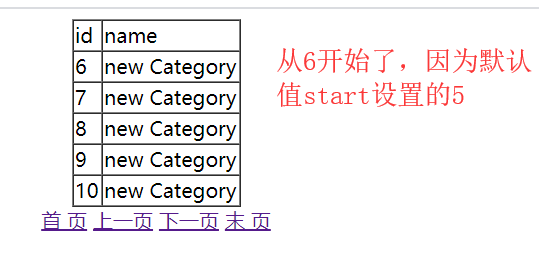


这里的Start和page对象里的属性Start的一样的名字，在接受参数时：



使用page接受的话，spring会自动生成page对象，并将传过来的start赋值到这个page对象上。

如果没有传（比如第一次访问页面，就会找那个类当做初始对象，其中的初始值当做默认值），比如将类中Start默认值设置为5后



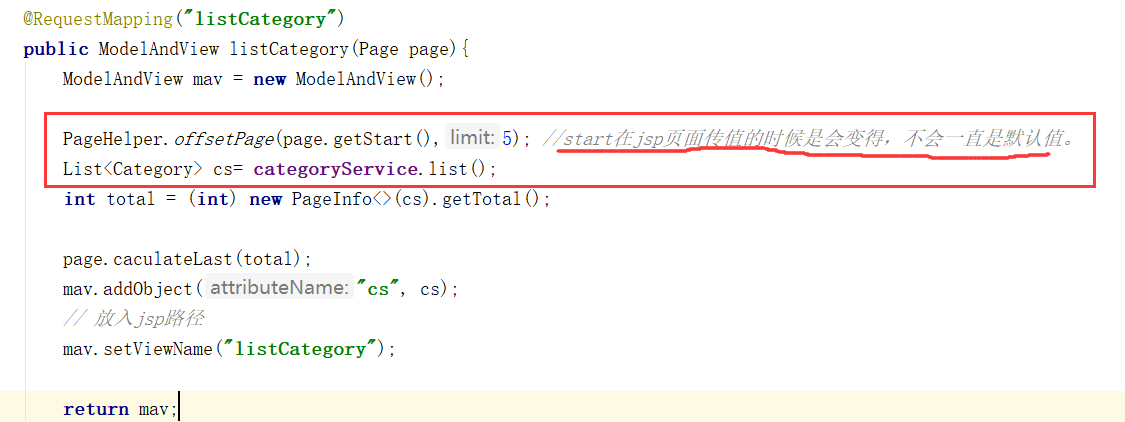
相当于传属性，而用对象去接受属性，不是原来的必须传一个相同格式的对象或字符串。

这里完成分页需要一个page类专门用来控制页数。



**使用pageHelper:**

导入相应的包，删掉原来有的list-limited查询（4个文件）



就这几句话就ok

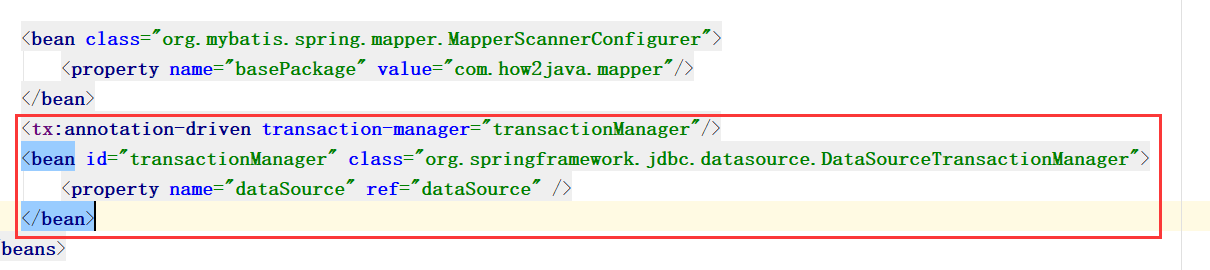
## 4.c3p0连接池配置：



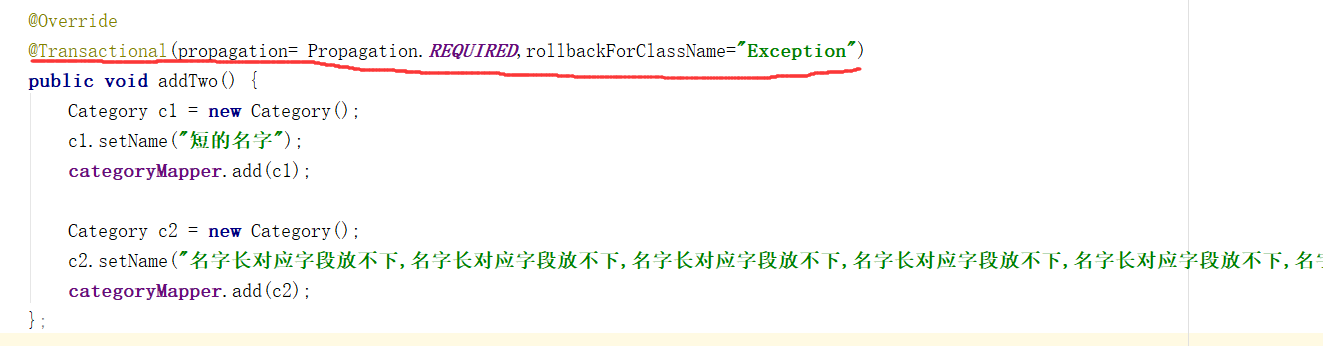
复制粘贴地址：<https://how2j.cn/k/ssm/ssm-c3p0/1142.html#nowhere>

## 5.事务配置：

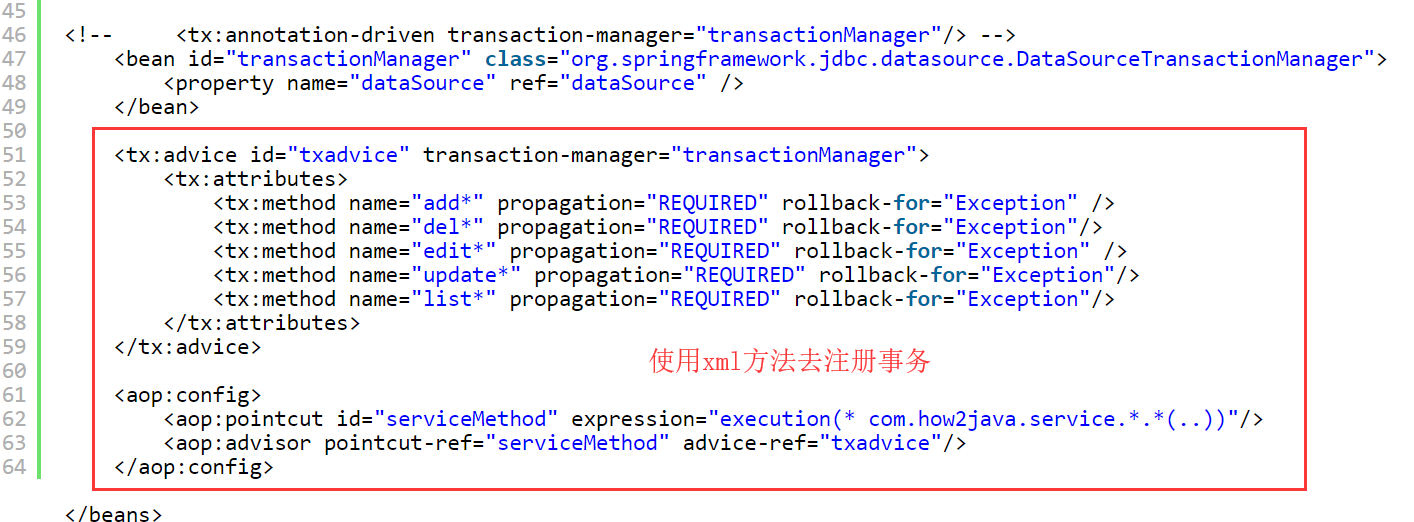
在Spring-Mybati中开启事务。



指定的方法配置事务，成功就全部成功，失败则全部失败。



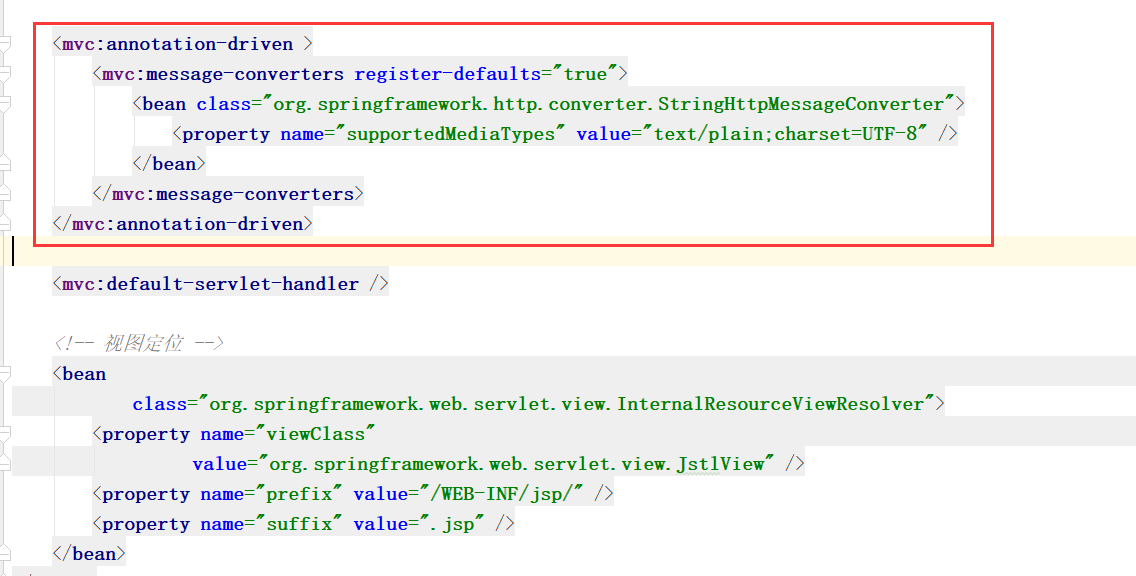
还有使用xml的方式去注册事务：



## 6.接口设计：



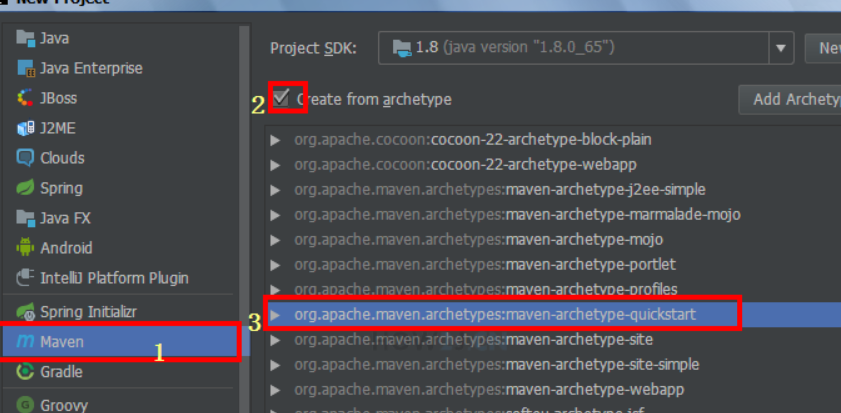
需要配置一下，方便接受json文件：



## 7.Maven:

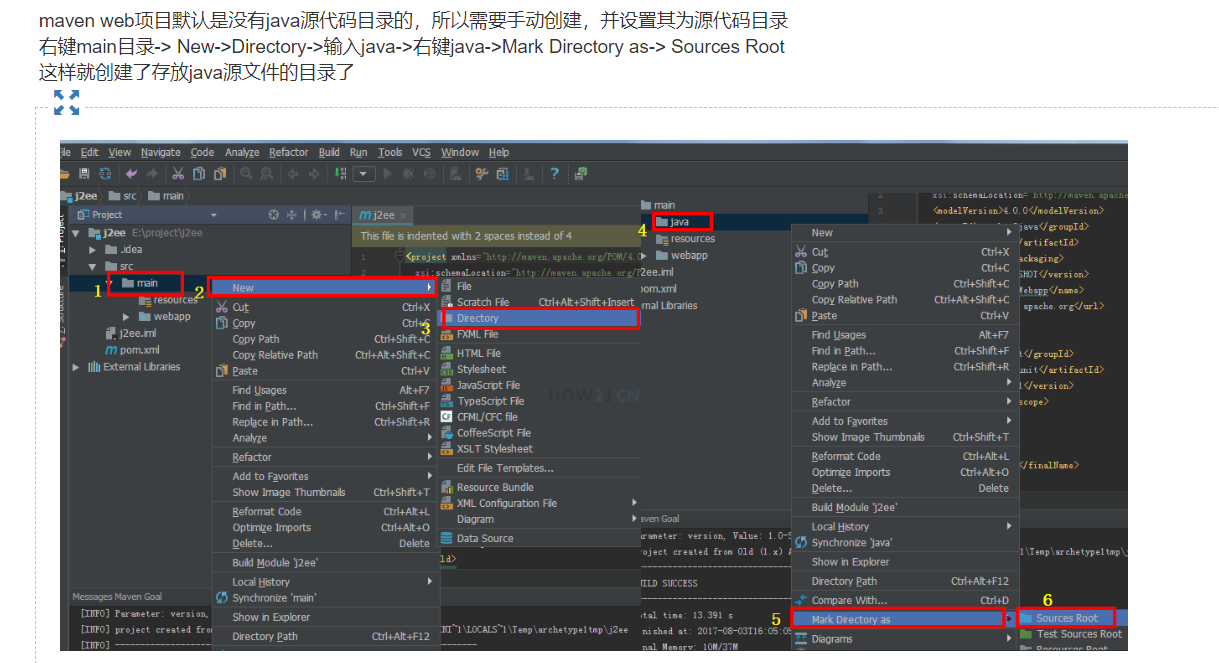
Maven不只是一个工具，更是一种构建java项目的风格。

在创建maven项目时，可以勾选快速创建一个已存在的maven模板。

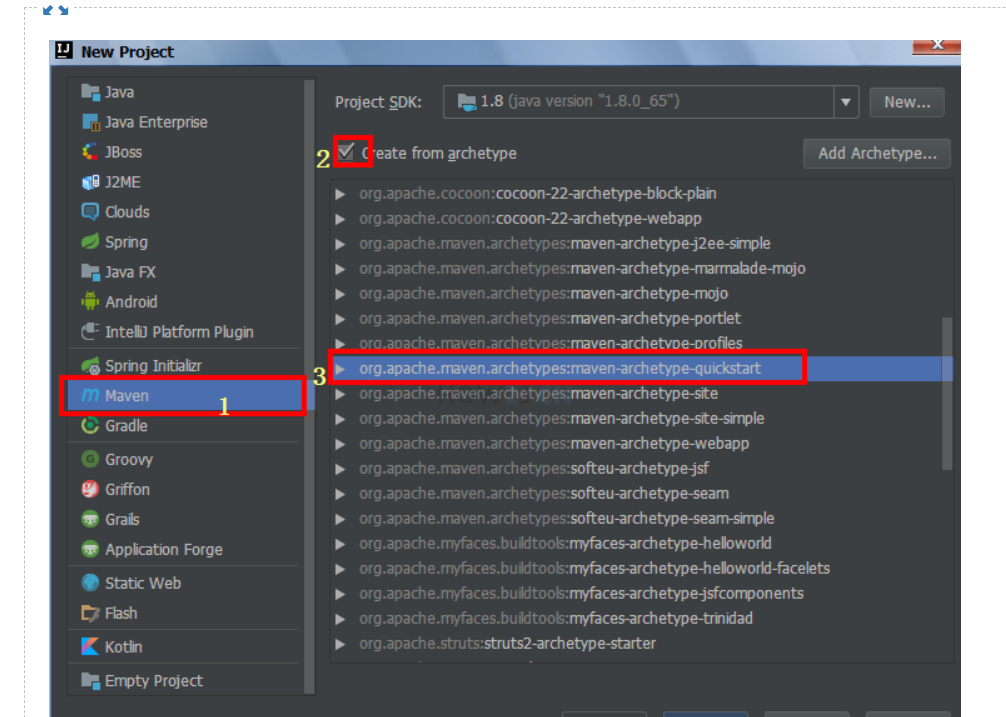


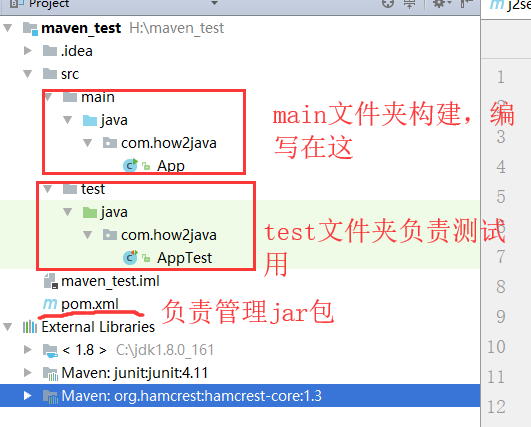
还可以配置webapp版本的。

App版本的模板，需要自己创建java目录:

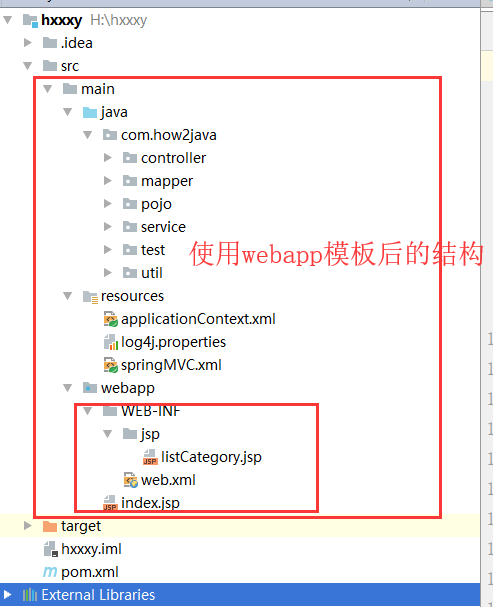


当然，前提是要在idea中把maven的配置全部写好。（省的后面又得配）





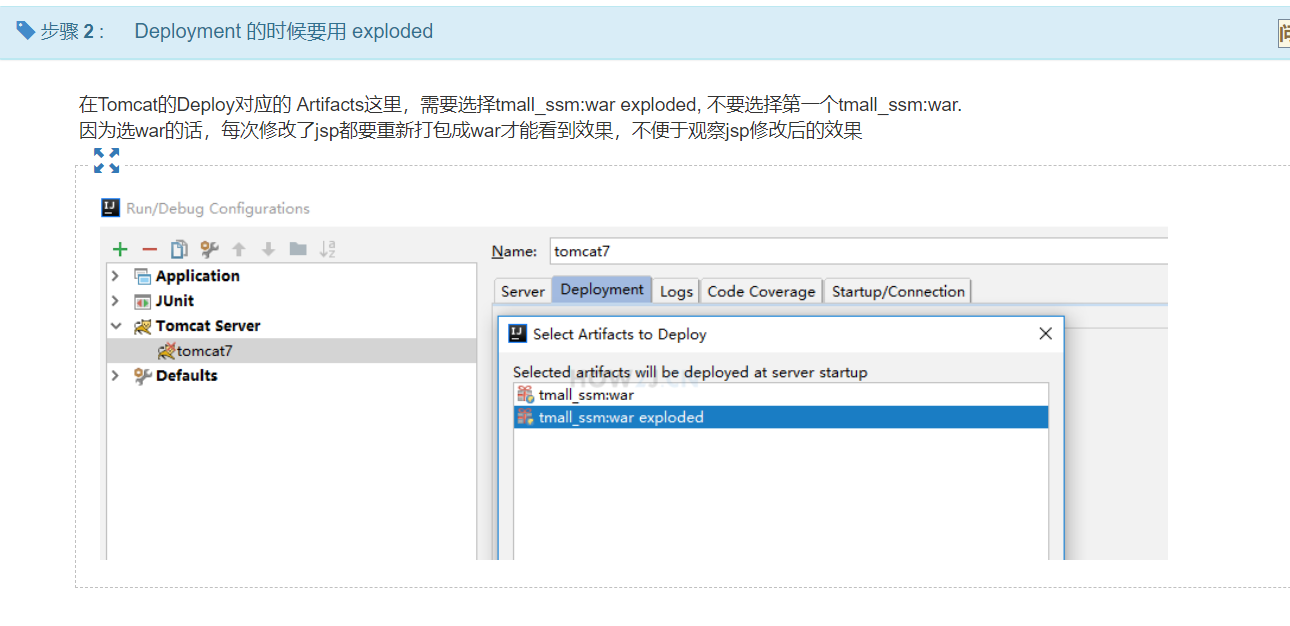
Webapp模板的maven:



将一个存在的maven项目导入的话，source from existing project 然后选择哪个project的pom.xml文件然后一直next即可。

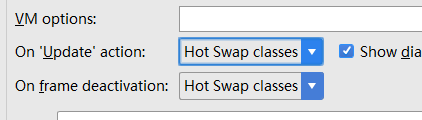
8.Tomcat的三个特别方便的技巧

使用exploded

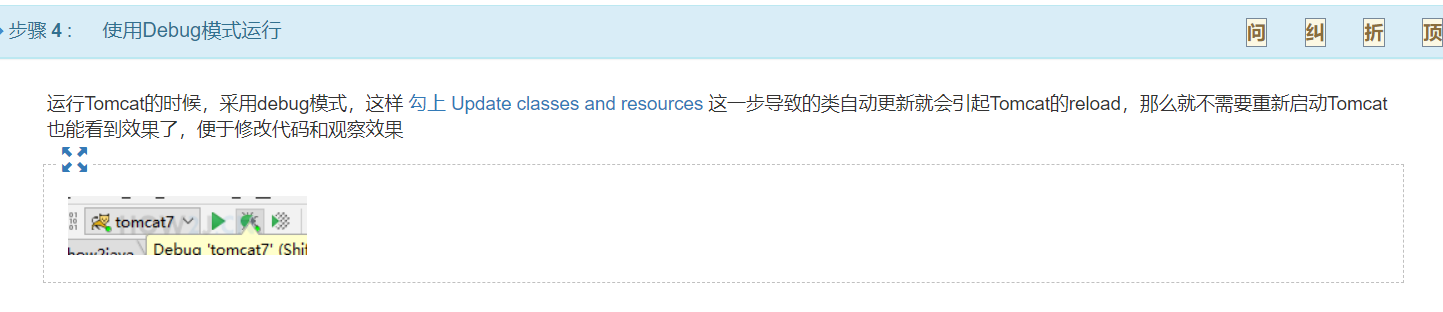


1. 更新的快捷：

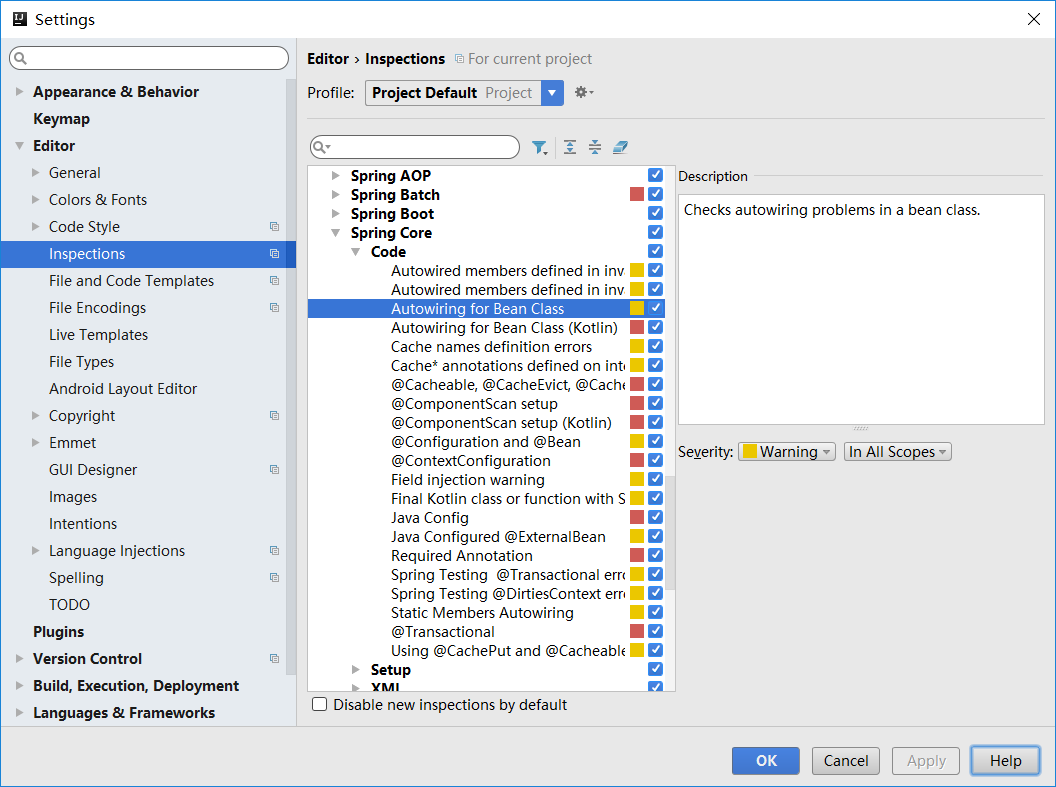




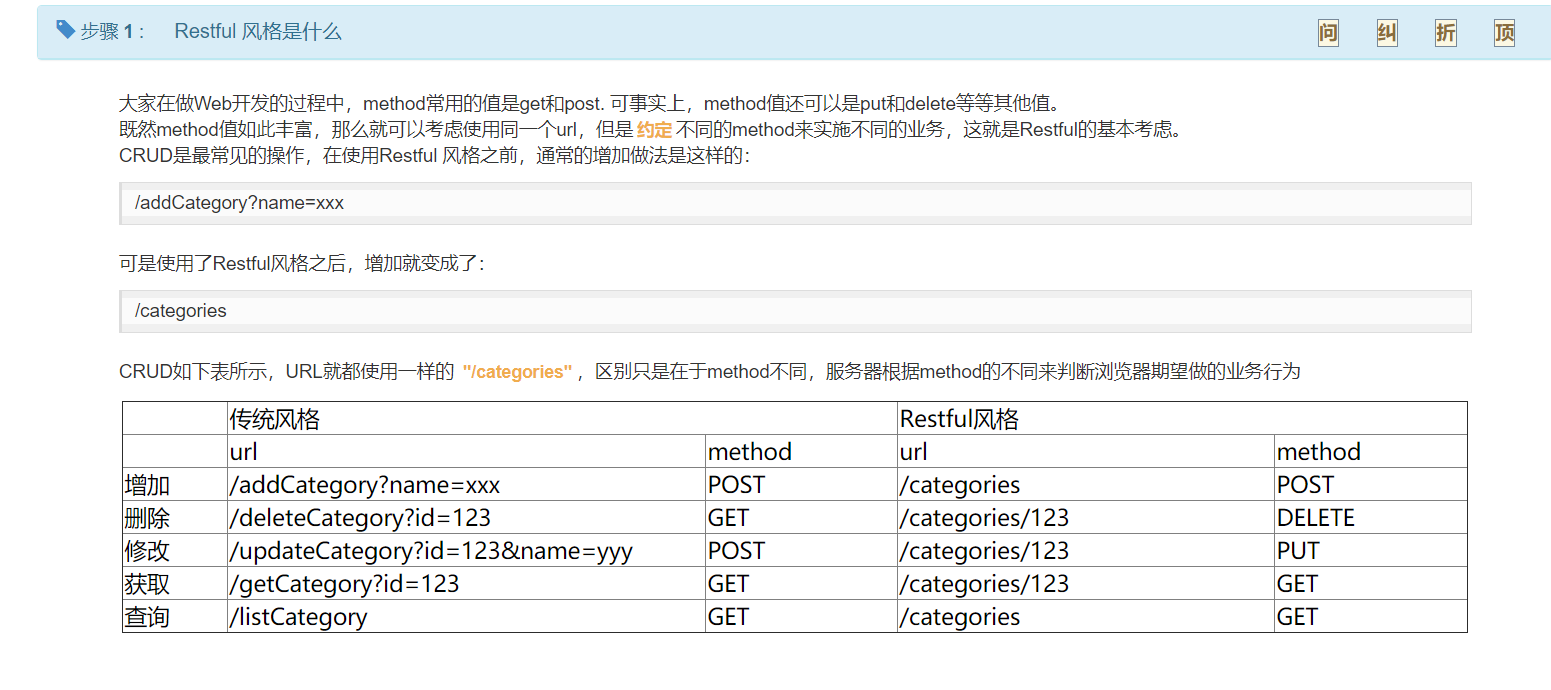
再使用debug，方便观察变化:

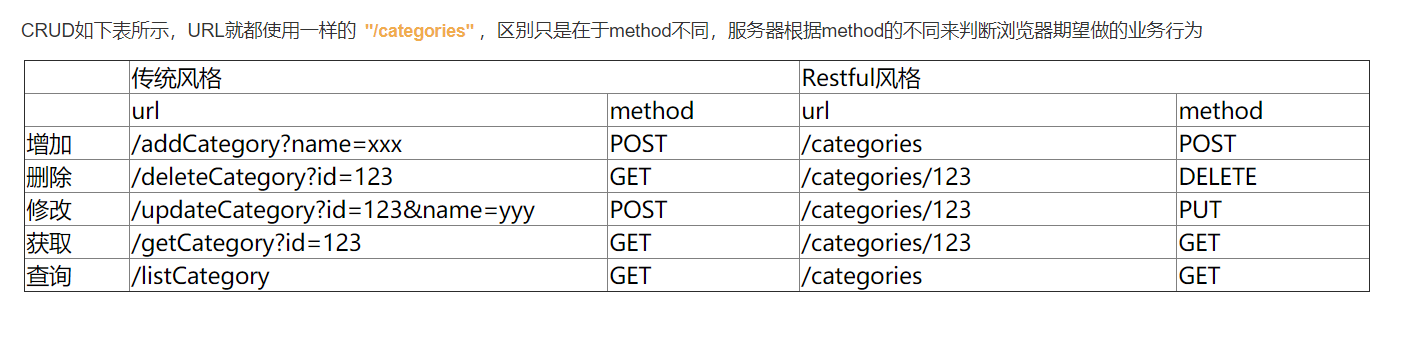


设置报错级别：



## 8.Restful：





## 9.SSM项目 多对多的实现：

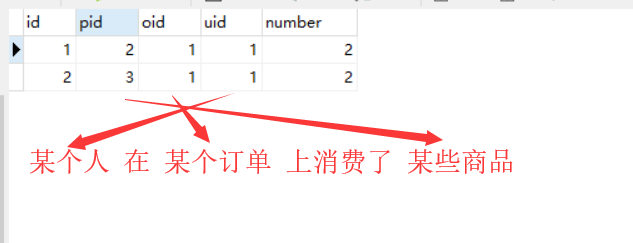
### 1.产品product和属性property是多对多的关系。

所以在设计数据库时需要设计一个中间表propertyValue

PropertyValueServiceImpl实现了PropertyValueService：  
1. get(int ptid, int pid)  
根据属性id和产品id获取PropertyValue对象（这里取得的是某产品的某一条属性）  
  
2. list(int pid)  
根据产品id获取所有的属性值  
  
3. init方法（这个很重要）  
3.1 这个方法的作用是初始化PropertyValue。 为什么要初始化呢？ 因为对于PropertyValue的管理，没有增加，只有修改。 所以需要通过初始化来进行自动地增加，以便于后面的修改。  
3.2 首先根据产品获取分类，然后获取这个分类下的所有属性集合  
3.3 然后用属性和id产品id去查询，看看这个属性和这个产品，是否已经存在属性值了。  
3.4 如果不存在，那么就创建一个属性值，并设置其属性和产品，接着插入到数据库中。  
这样就完成了属性值的初始化。  
  
4. update（PropertyValue pv)  
更新

详情页：<https://how2j.cn/k/tmall_ssm/tmall_ssm-1483/1483.html#nowhere>

### 2.关于商品和订单以及用户的多对多关系：



### Order订单和orderItem订单项的关系（一对多嵌套）：

首先获取所有的List<Order>取得所有的订单（这个时候就可以对每一个Order去setUser了），然后针对每一个订单order进行订单项的填充fill

在一个order中肯定是List<orderItem>，而每一个orderItem都有对应的product。所以在orderService中提供了fill()方法来填充order。

这个fill方法可以装配好每一个订单项，然后把订单项全部打包填充给对应的Order。

Fill方法首先提供order，通过order的id来查找有哪些订单项。

找到订单项List后，开始对这个List<orderItem>进行遍历，对订单项填充必要的信息（如product：根据orderItem的pid去setProduct）。

最后获得完整的List<order>,并返回给前端

List<order> 🡪对每个order填好user –>对每个order进行orderItem的fill（product填充） 🡪获得完整的List<order>返回给前端

其实也就是根据id填充对象（user，product）使其信息完整的过程，思路是从宏观到微观。

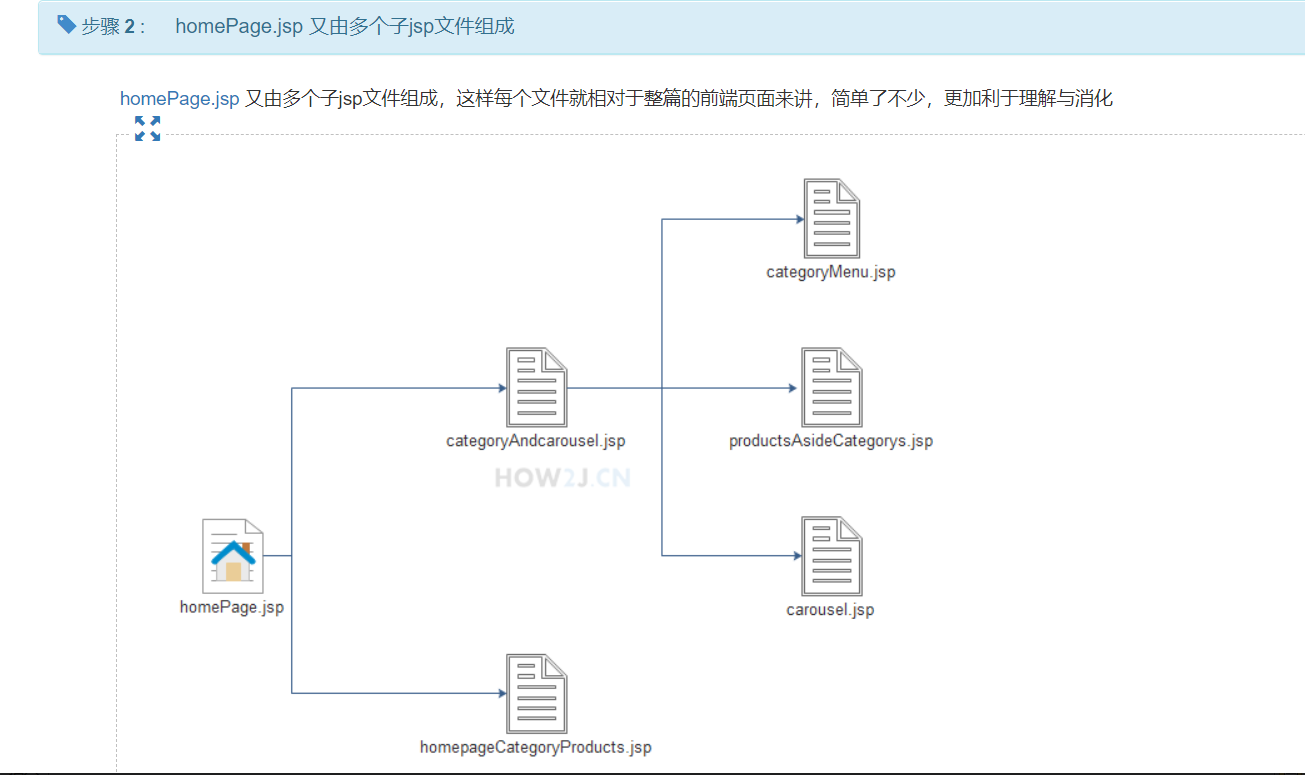
## 10.关于example

Example目前没看懂，好像是先生成一个实例，然后去数据库里看看有没有和这个实例相匹配的词条。

## 11.前端的实现：

### 1.真正的开发前端都是模块化的开发：





### 2.关于一个图片如何做到定位和浮动出现：

猫耳浮出的效果：





2.产品列表分类的出现和消失：



这个和原来的板块设计很类似，都是鼠标移入就出现，移开区域就消失。

通过jq就可以设置，只是这个是通过cid去控制的。

