## Spring思想：

### 1.IOC （控制反转）

Spring是一个对象生产管理机器，其原理可以从IOC（控制反转）来体现。

平时，一个对象的生产，属性定义，都是自己完成的。（new，set）

但是使用Spring之后，对象控制权就交到Spring手上了，定义和属性赋值都是在Spring中完成，你能做的只是得到这个对象。这就叫做IOC，控制反转。

奇妙比喻：

买菜自己做和餐馆（Spring点菜）：

**传统方式：**相当于你自己去菜市场new 了一只鸡，不过是生鸡，要自己拔毛，去内脏，再上花椒，酱油，烤制，经过各种工序之后，才可以食用。  
**用 IOC：**相当于去馆子(Spring)点了一只鸡，交到你手上的时候，已经五味俱全，你就只管吃就行了。

原文：

**传统的方式：**  
通过new 关键字主动创建一个对象  
**IOC方式**  
对象的生命周期由Spring来管理，直接从Spring那里去获取一个对象。 IOC是**反转控制** (Inversion Of Control)的缩写，就像控制权从本来在自己手里，交给了Spring。



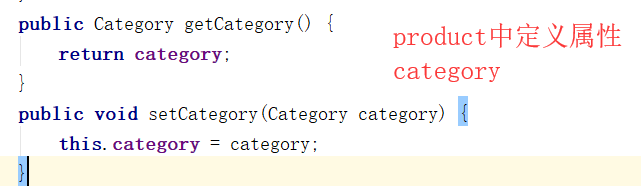
### 2.DI （注入对象）

Spring的第二大特性 注入对象。

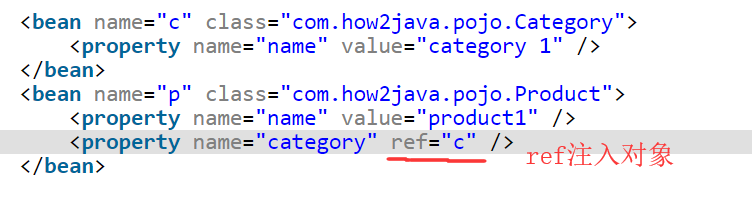
在上一个例子中，spring向一个Category对象注入了一个字符串“category 1”

怎么注入对象呢？

首先准备一个product类，其中定义一个Category，也是后面需要注入的对象。



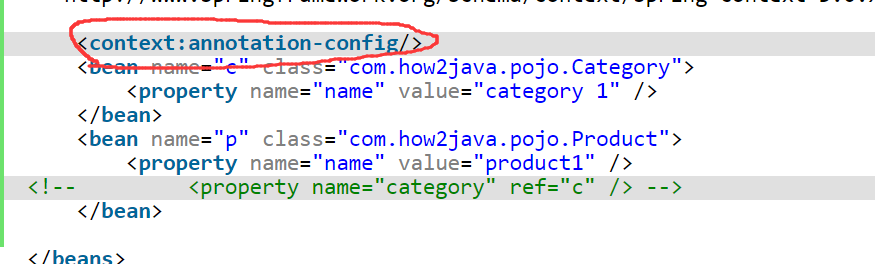
Ref



这样就把c对象注入了那个product。

还可以使用注解的方式去注入对象:

首先要在配置文件中设置使用



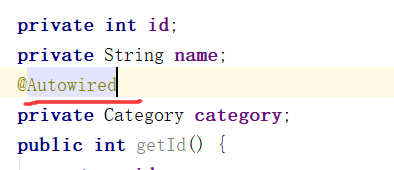
#### 1.自动编制注解(Autowired)：

这个方法很牛逼，是自动监测对象自动添加的，就是如果只实现了一个那种类(单列模式，那么进行autowired时就会吧那个类注入生成的类)，

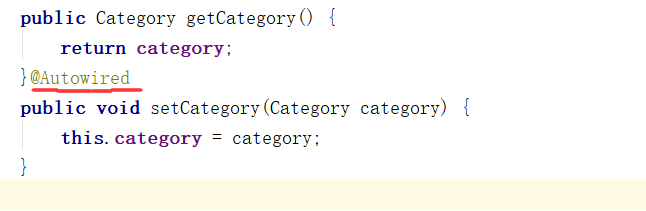
这种方式更多用在接口的autowired，声明一个接口用autowired注释，spring就会去找唯一实现这个接口方法的那个类。比如sps实现了sp接口,autowired修饰的sp就会自动注入变成sps这个类)。

比如要在product对象中注入一个category c。

就直接在product类中定义Category或者是setCategory的地方使用注解即可，spring会自己找该注入那个类。



或者在set前加：

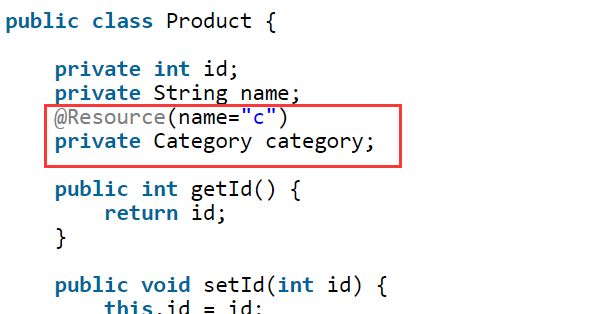


这样product对象就会自动注入Category对象了。

#### 2.指定对象添加的方式(Resource)：

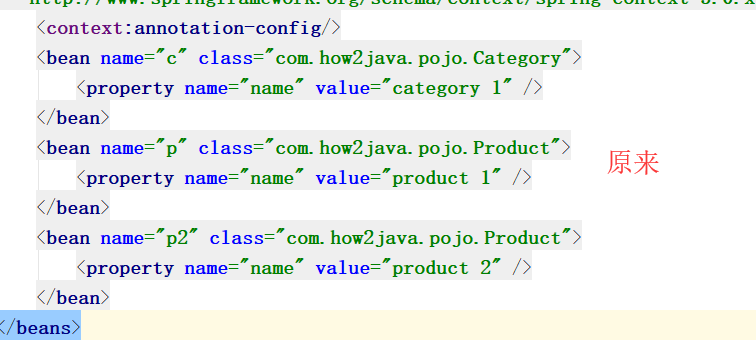
这种方法才是常见方法，在类中定义category的时候进行定向注解:

@Resource(name="c")

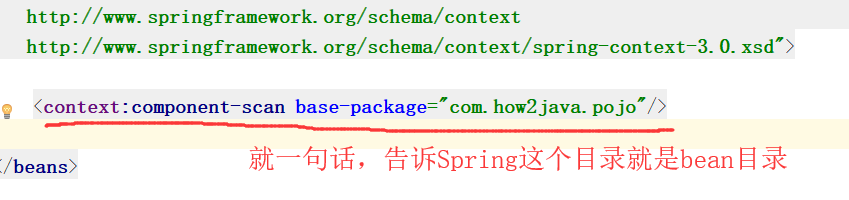


#### 3.对bean本身的注解创建方式

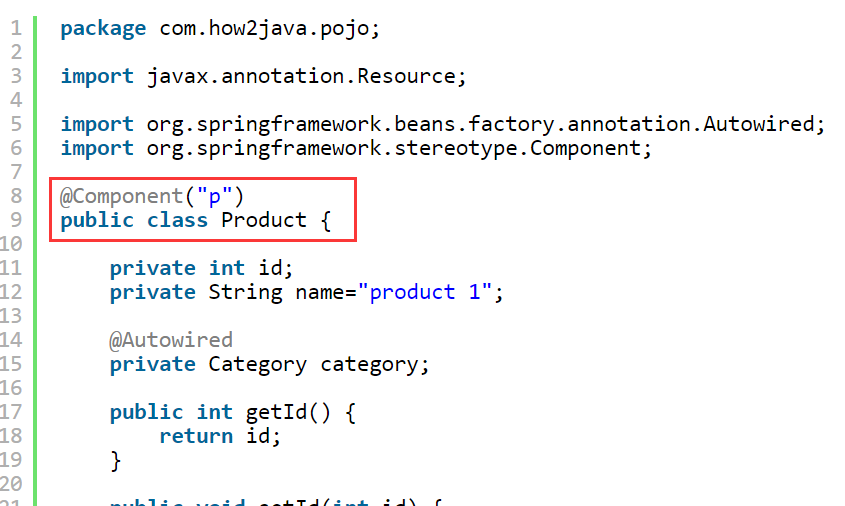
原来Spring中创建一个对象要写在配置文件中，现在可以使用注解来直接生产一个对象。

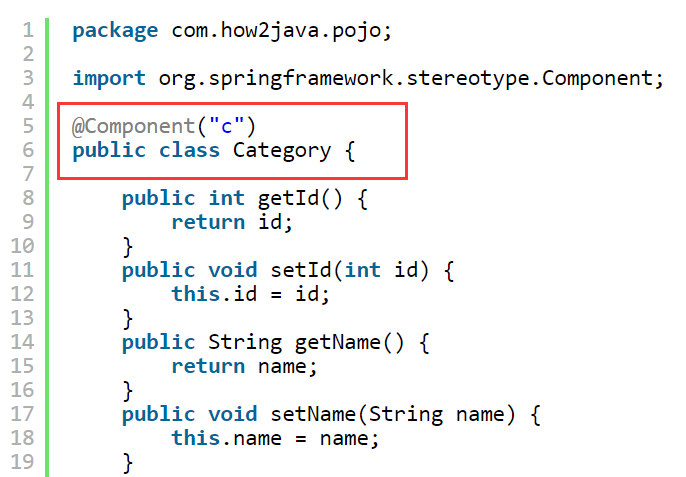


现在：

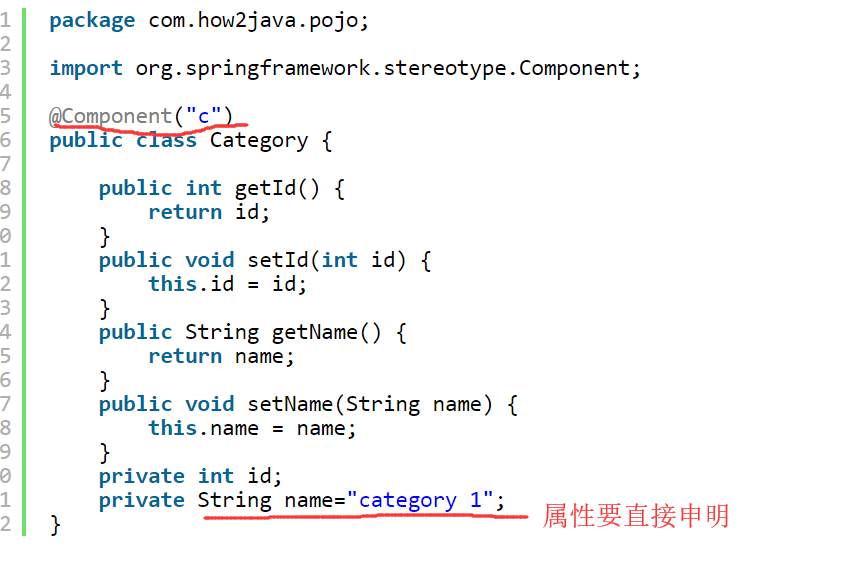


然后在类上面注解@component告诉Spring这是bean就ok了。





此时 属性要在类中直接声明（之前是在配置文件中声明的）。



但是这样不就变成了单例了么。。。。

### 3.AOP面向切面编程

#### 1.概念

Aop = aspect orient program

面向切面编程的思想是 核心功能 和 辅助功能（切面功能） 分离开发的思想。

各自开始时将耦合性降到最低，互相独立模块开发互不影响，最后用一种“需求拼接”的方式打包完成整个项目的编程思想（有点架构内味了）

奇妙比喻：

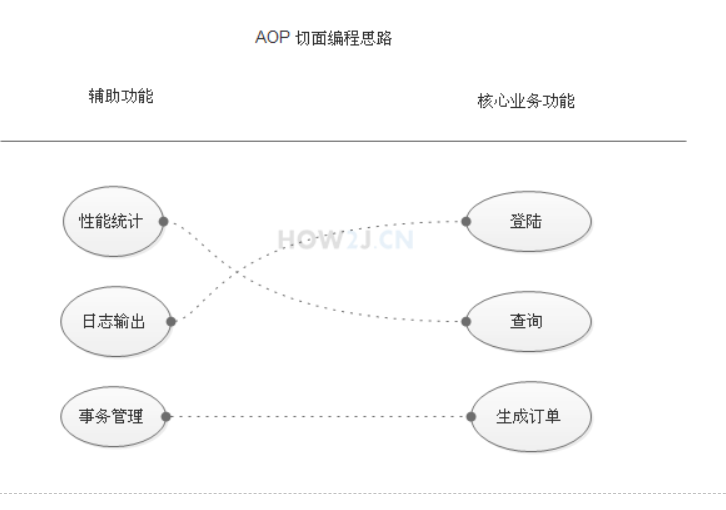
主食（核心功能）+ 不同配菜（切面功能） = 一顿饭（一个程序）。

原文：

AOP 即 Aspect Oriented Program 面向切面编程  
首先，在面向切面编程的思想里面，把功能分为**核心业务功能**，和**周边功能**。  
所谓的核心业务，比如登陆，增加数据，删除数据都叫核心业务  
所谓的周边功能，比如性能统计，日志，事务管理等等  
  
周边功能在Spring的面向切面编程AOP思想里，即被定义为**切面**  
  
在面向切面编程AOP的思想里面，核心业务功能和切面功能分别**独立进行开发**  
然后把切面功能和核心业务功能 **"编织"** 在一起，这就叫AOP

关于编制：

1. 功能分两大类，辅助功能和核心业务功能  
   2. 辅助功能和核心业务功能**彼此独立**进行开发  
   3. 比如登陆功能，即便是没有性能统计和日志输出，也可以正常运行  
   4. 如果有需要，就把"日志输出" 功能和 "登陆" 功能 **编织**在一起，这样登陆的时候，就可以看到日志输出了  
   5. 辅助功能，又叫做**切面**，这种能够**选择性的，低耦合的**把切面和核心业务功能结合在一起的编程思想，就叫做切面编程

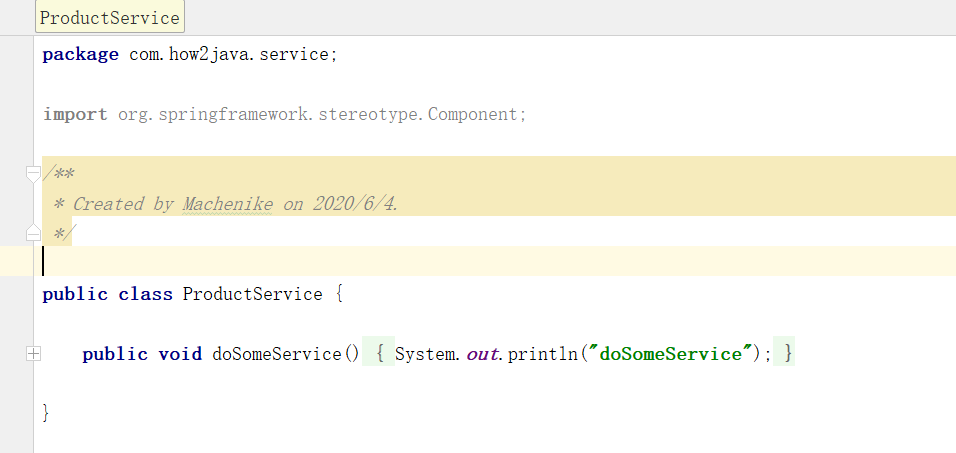


要什么功能自选打包。

#### 2.实例（配置文件，注解）：

一个Service（核心功能）要加上一个log（切面功能）功能：

1. 核心Service:



1. 配置使用切面:



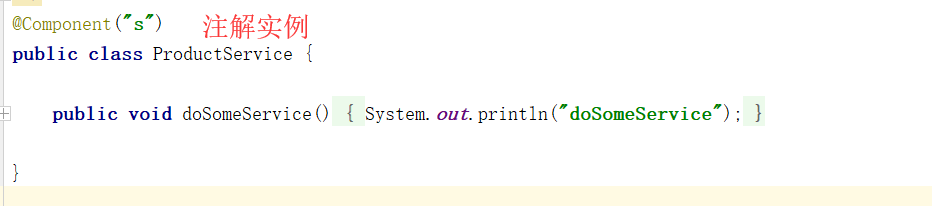
配置文件：



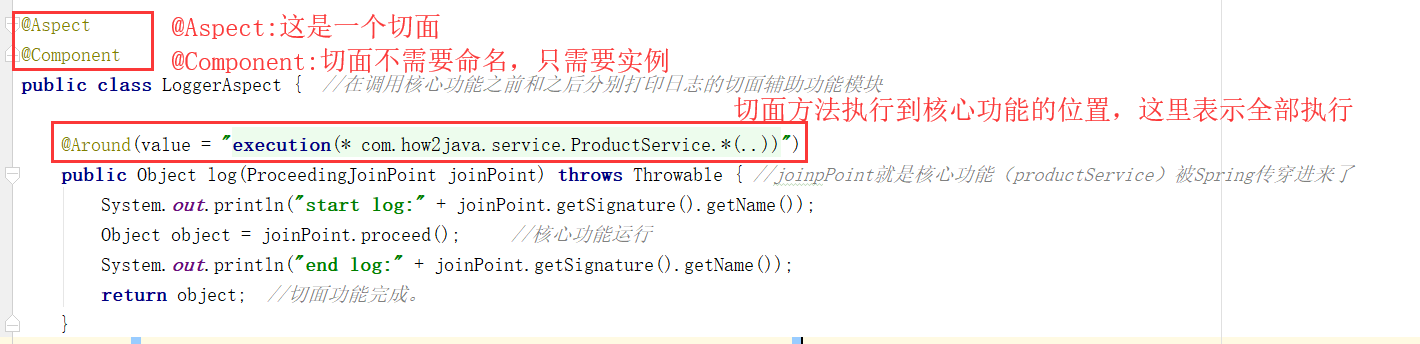
既然有配置文件，那么肯定也可以使用注解的方式。

2.注解方式：

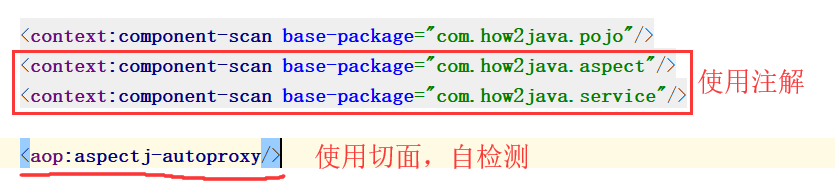
核心功能：



切面功能：



配置文件：



运行结果是一样的。

## 2.SpringMVC

### 1.基本原理及配置

原理图：



配置：

1.Web.xml:

这里原本是访问的入口，但是现在是spring结构，因此要将所有的访问转移到Spring-servlet当中。



2.Springmvc-servlet配置：

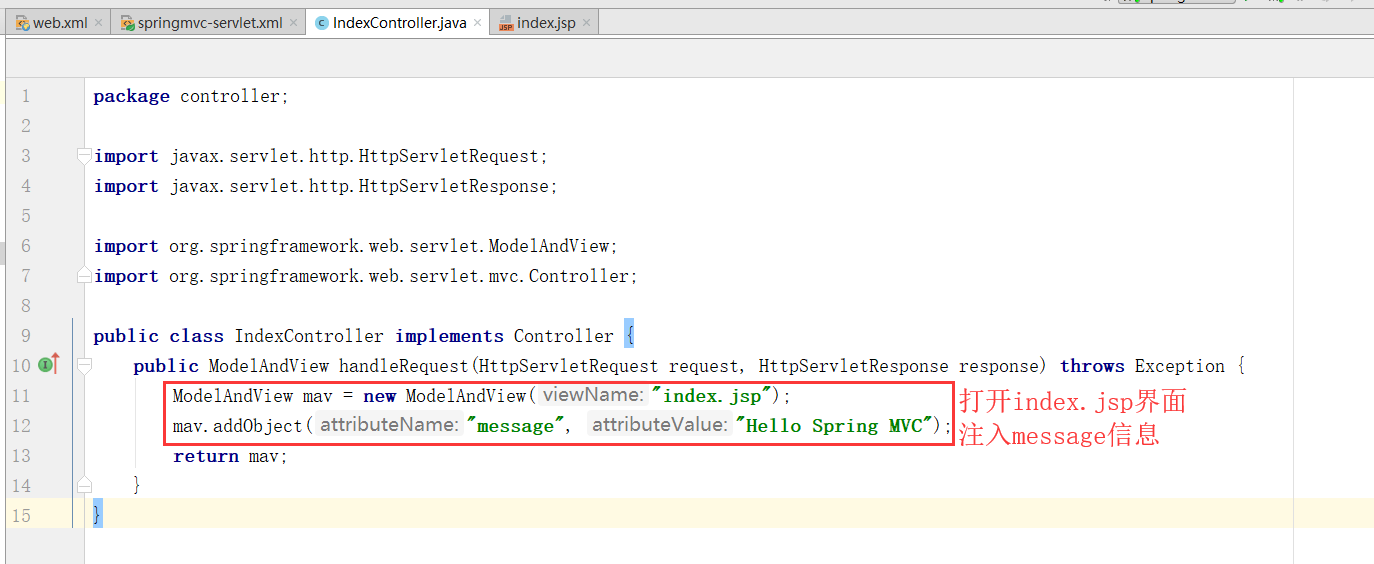
接收到了来自web.xml的访问请求后，这里才是实际上真正的servlet

在这里可以进行路由分配操作：



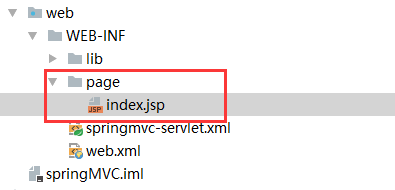
3.Controller：

控制器用来打开指定页面并传输内容



### 2.自定义文件路由(视图定位)

一个好的项目肯定是吧各类文件分类的，比如page里面全是html，css里面全是css文件，但是又为了让spring能够很好地访问这些页面，就需要做设置：



比如路径是这样的，要让controller去访问：



效果：



### 3.注解的方式实现配置

经典的配置转注解。

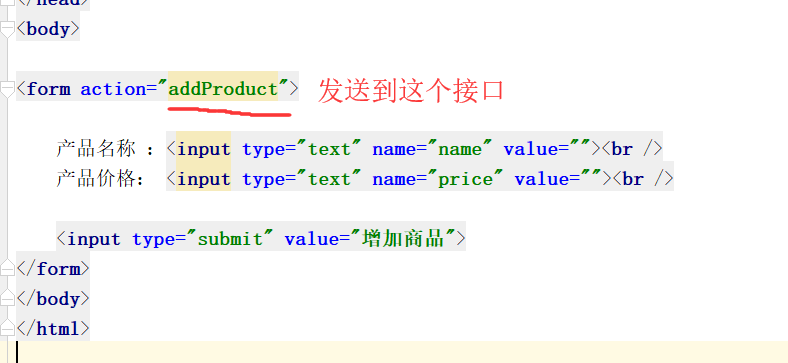
Controller：



配置xml:



### 4.表单提交：



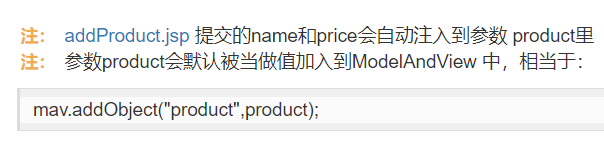
如果有配置文件，就会走到配置文件中寻找路由方向。

Web.xml --> SpringMVC-servlet.xml->Controllor

如果使用的是注解的方式：



这个括号里的形参会自动将传过来的参数进行赋值并返回，这就是Spring的牛逼的地方。



### 5.客户端跳转：

之前都是服务端跳转：

客户端跳转就是在服务端跳转当中加一句话：

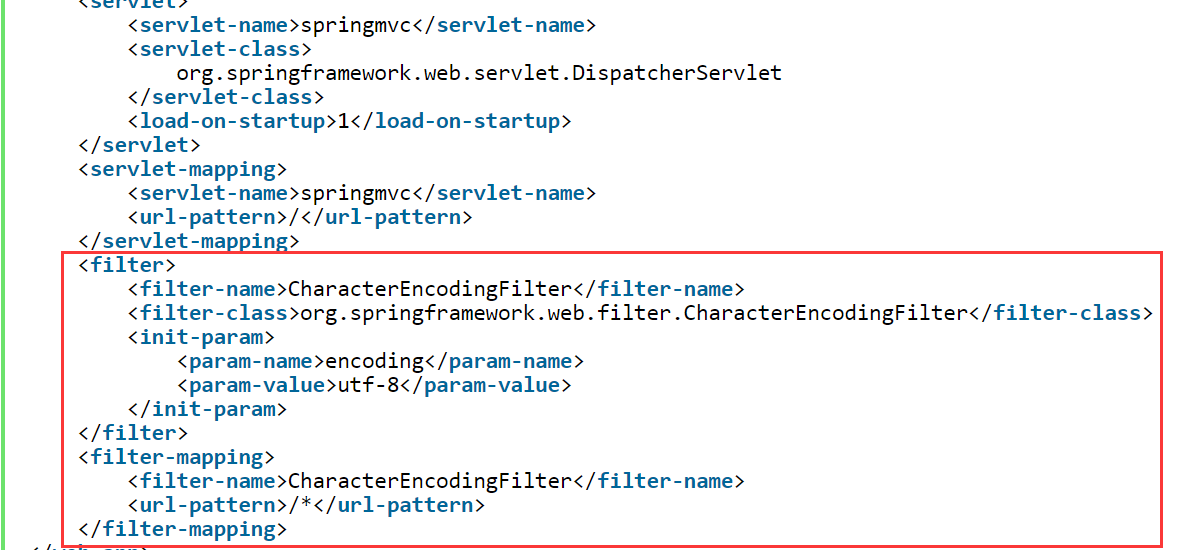


### 6.使用session:



### 7.处理中文问题（filter）：

在web.xml中添加中文过滤器:





<https://how2j.cn/k/tmall_ssm/tmall_ssm-1518/1518.html#nowhere>

### 8.文件上传：

Spring的文件上传比较方便，它自带了一个多媒体类（MultipartFile

），这个类可以代替所有的非结构化数据（图片，音乐）

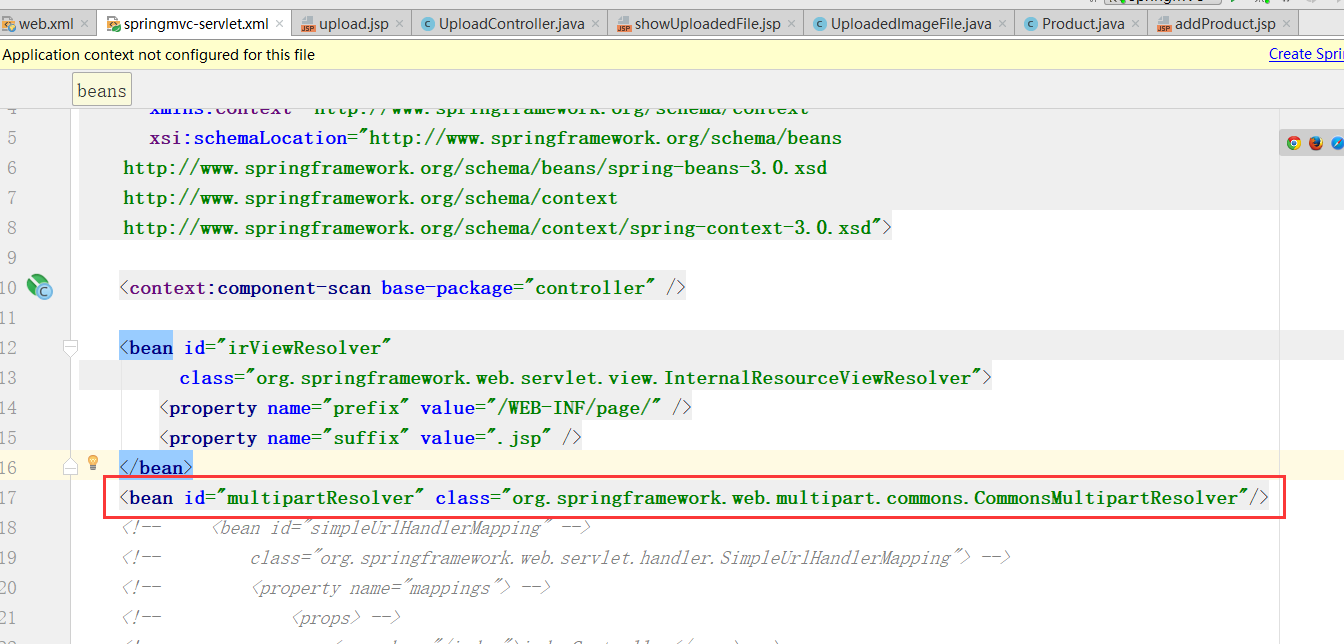
上传的文件肯定需要注入对象方便使用，所以需要先创建一个类用来接收上传的文件类型。



注意这里的属性名 image 要和前端传值的name保持一致。



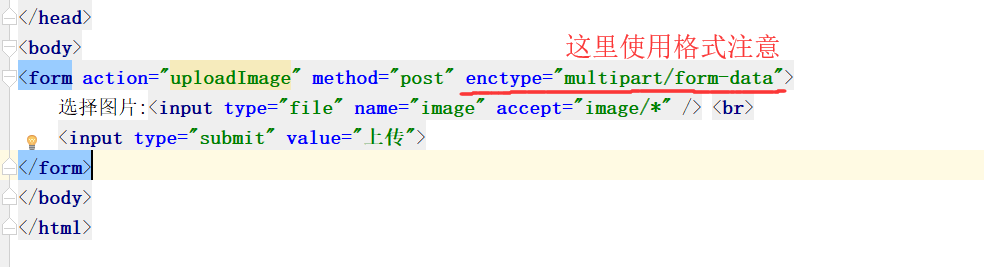
Spring配置，使用多媒体文件。

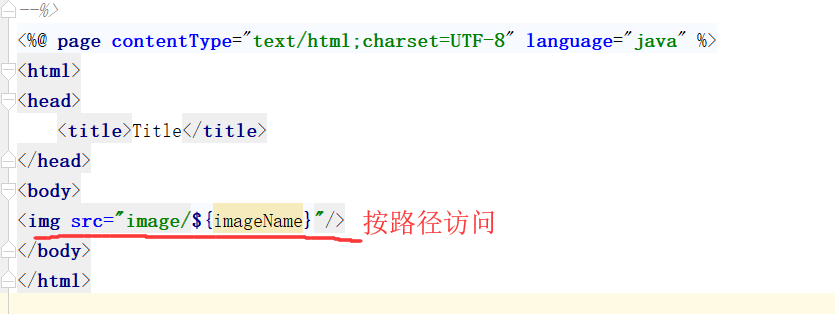


Web.xml:



前端：





Controller，核心部分：

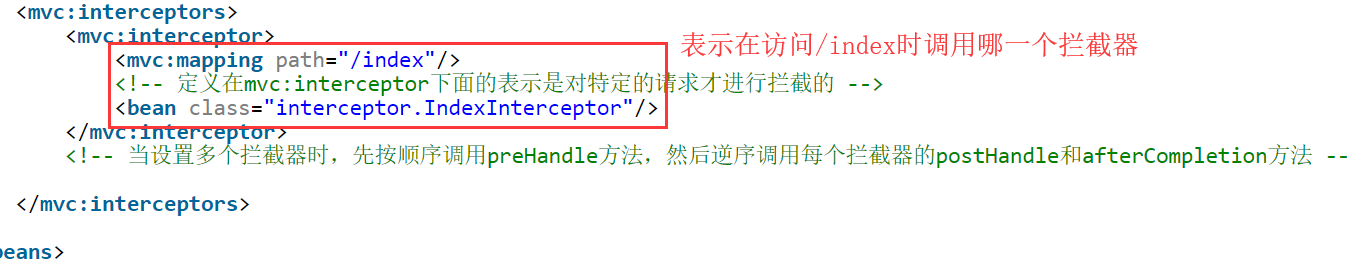


### 9.拦截器（生命周期控制器）

拦截器可以说成一个在页面执行的时候进行的插件操作，他在页面执行的前后都会行动。

配置:

Springmvc-servlet



拦截器写法:



这里简单介绍下，就是三个部分，但是在多拦截器的情况下，这三个方法会有很多的操作变换。

第一个preHandle:

是控制是否执行这个MVC，和调用其他拦截器的规则。

\* 在业务处理器处理请求之前被调用

\* 如果返回false

\* 从当前的拦截器往回执行所有拦截器的afterCompletion(),再退出拦截器链（倒着执行after）

\* 如果返回true

\* 执行下一个拦截器,直到所有的拦截器都执行完毕 （把所有的preHandle全部执行一遍）

\* 再执行被拦截的Controller

\* 然后进入拦截器链,

\* 从最后一个拦截器往回执行所有的postHandle()

\* 接着再从最后一个拦截器往回执行所有的afterCompletion()

简单地说就是，

如果失败了，就从这个拦截器开始倒着进行前面全部的afterHandle，然后退出。

如果全部成功了，

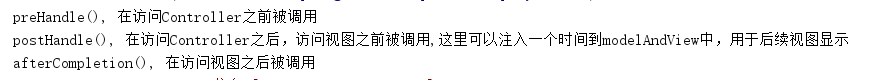
就先执行所有的preHandle(),

然后执行controller中的内容，

然后从最后一个拦截器倒着往前执行postHandle() (注意，这个时候只是执行完了controller，还没有进行view层的渲染，postHandle可以往即将执行的view层添加东西)

最后再从最后一个拦截器往前执行一遍所有的afterHandle（这个时候所有的view层已经执行完毕。）

总共从后往前跑了两次。



总结：(true和false是在preHandle中就决定了的)

True：

\*preHandle 🡪 C 🡪（倒）\*postHandle 🡪 V 🡪（倒）\*afterHandle

False:

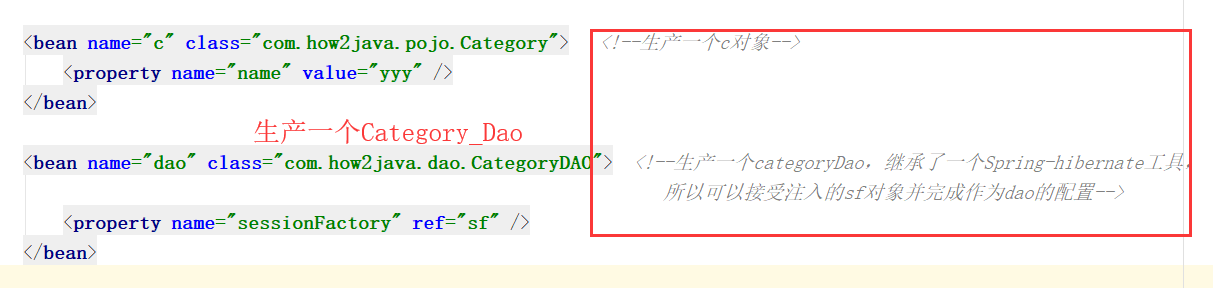
\*preHandle(有一个false)

🡪 (从false的那一个preHandle的拦截器开始)(倒) \*afterHandle

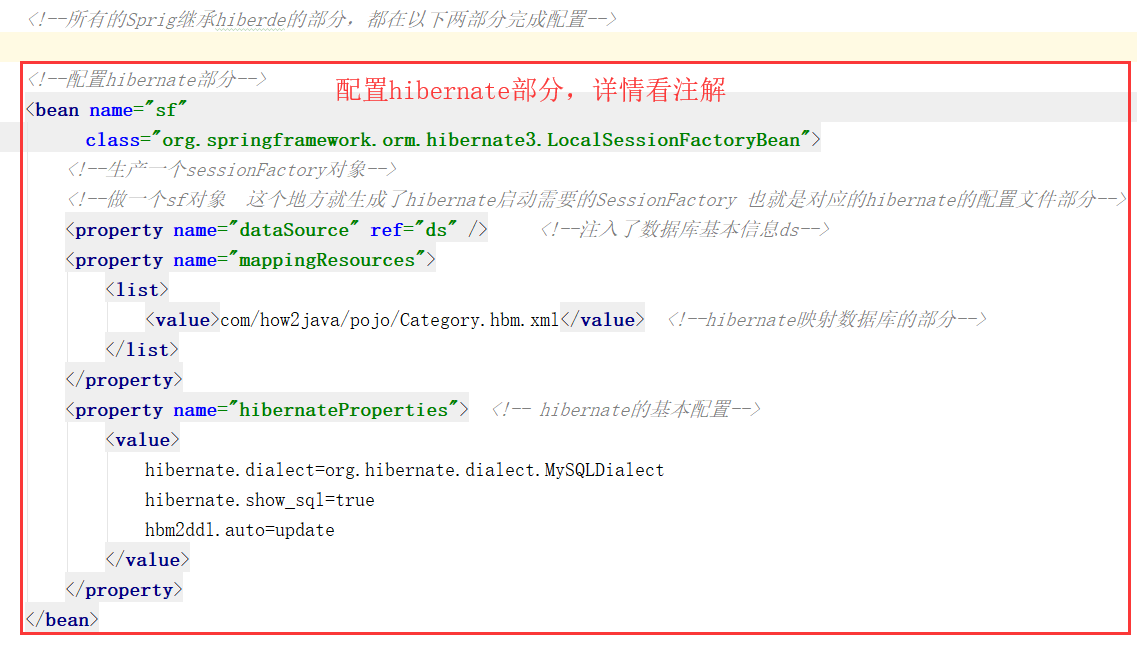
## 3.Spring\_hibernate:

### 1.基本配置和使用：

Spring继承hibernate，在Spring的配置文件中就可以实现连接数据库和继承hibernate



关键点就在下面的配置部分，继承spring和hibernate

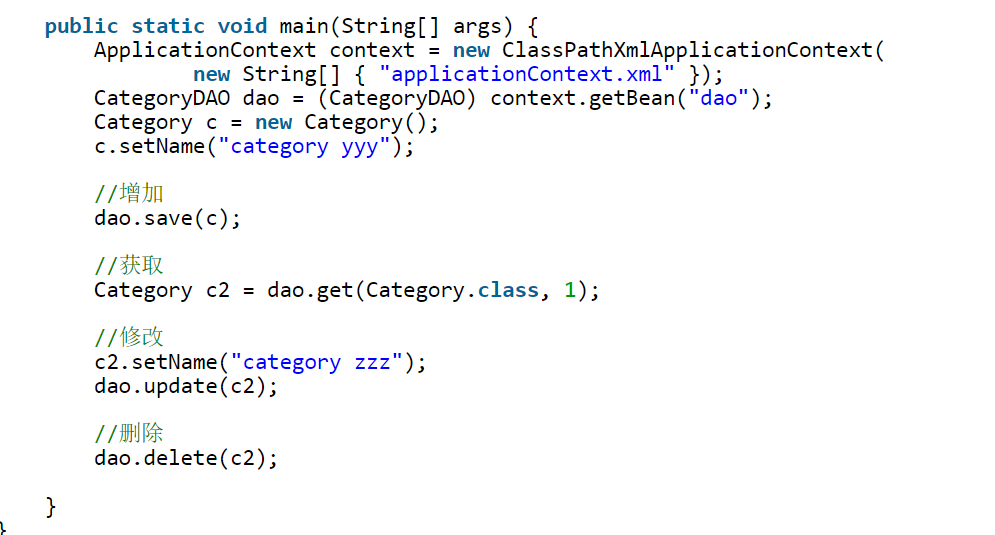








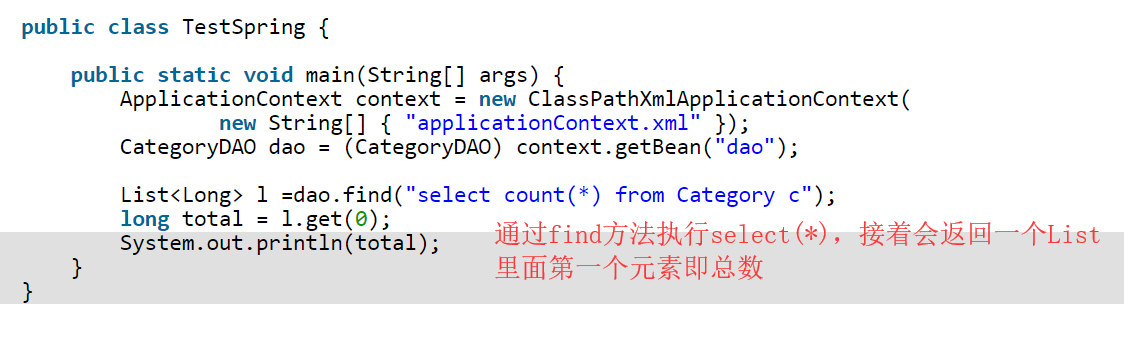
增删改查：



### 2. 分页，总数：

使用Criteria工具 很方便实现分页。





### 3.查询：



### 4.使用连接池和Oracle

<https://how2j.cn/k/spring-hibernate/spring-hibernate-c3p0/362.html#nowhere>

<https://how2j.cn/k/spring-hibernate/spring-hibernate-oracle/106.html#nowhere>