springmvc第一天 springmvc的基础知识

springmvc课程安排：

第一天： 基础知识

springmvc框架(重点)

mvc在b/s系统中应用方式

springmvc框架原理（DispatcherServlet前端控制器、处理器映射器、处理器适配器、视图解析器）

springmvc入门程序

需求：商品列表查询

常用的处理器映射器，处理器适配器

注解的处理器映射器，处理器适配器用于注解开发(重点)

Mybatis和Springmvc整合

注解开发基础

常用的注解使用

参数绑定

自定义参数绑定

springmvc和struts的区别

第二天： 高级知识

使用springmvc上传图片

集合参数的绑定

validation校验器

异常处理器使用（用于系统异常处理）

RESTful支持

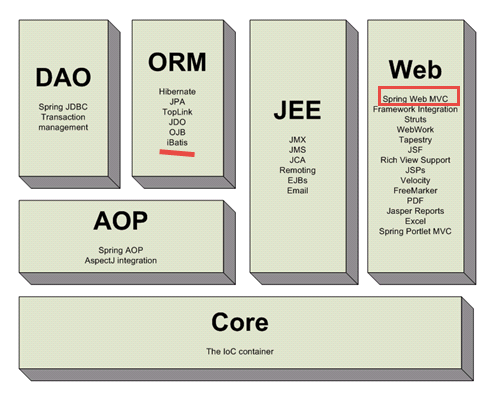
拦截器

学生练习

# springmvc框架

## springmvc是什么？

springmvc是spring的一个模块，提供web层解决方案（基于mvc设计架构）



## mvc在b/s系统的应用

mvc是一个设计模式，在b/s系统的应用：

request

**C （controller）控制器**

**作用：接收用户请求**

**及响应结果**

response

**M （model）**

**模型**

**pojo**

**action**

**service**

**dao**

请求模型进行处理

返回处理结果

**V (view)视图**

填充模型数据渲染视图

注意：b/s系统模型无法将数据推送到视图

## springmvc的框架

DispatcherServlet

前端控制器

1 request请求

url

Handler处理器

HandlerMapping处理器映射器

2请求查找Handler

3 HandlerExecutionChain{

HandlerInterceptor

HandlerInterceptor

Handler

}

HandlerAdapter处理器适配器

4请求执行Handler

5执行

6ModelAndView

7向前端控制器返回

ModelAndView

ViewResolver视图解析器

8请求进行视图解析

9返回view

View视图

10渲染视图

向view填充模型数据

11 response

第一步：用户发起request请求，请求至DispatcherServlet前端控制器

第二步：DispatcherServlet前端控制器请求HandlerMapping处理器映射器查找Handler

DispatcherServlet：前端控制器，相当于中央调度器，各个组件都和前端控制器进行交互，降低了各个组 件之间耦合度。

第三步：HandlerMapping处理器映射器，根据url及一些配置规则（xml配置、注解配置）查找Handler，将Handler返回给DispatcherServlet前端控制器

第四步：DispatcherServlet前端控制器调用适配器执行Handler，有了适配器通过适配器去扩展对不同Handler执行方式（比如：原始servlet开发，注解开发）

第五步：适配器执行Handler

Handler是后端控制器，当成模型。

第六步：Handler执行完成返回ModelAndView

ModelAndView：springmvc的一个对象，对Model和view进行封装。

第七步：适配器将ModelAndView返回给DispatcherServlet

第八步：DispatcherServlet调用视图解析器进行视图解析，解析后生成view

视图解析器根据逻辑视图名解析出真正的视图。

View：springmvc视图封装对象，提供了很多view，jsp、freemarker、pdf、excel。。。

第九步：ViewResolver视图解析器给前端控制器返回view

第十步：DispatcherServlet调用view的渲染视图的方法，将模型数据填充到request域 。

第十一步：DispatcherServlet向用户响应结果(jsp页面、json数据。。。。)

DispatcherServlet：前端控制器，由springmvc提供

HandlerMappting：处理器映射器，由springmvc提供

HandlerAdapter：处理器适配器，由springmvc提供

Handler：处理器，需要程序员开发

ViewResolver：视图解析器，由springmvc提供

View：真正视图页面需要由程序编写

# 入门程序

## 需求

实现商品列表查询

## 需要的jar包

使用spring3.2.0（带springwebmvc模块）



## 前端控制器

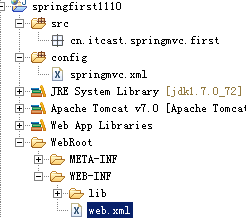
在web.xml中配置：



## springmvc.xml

在springmvc.xml中配置springmvc架构三大组件（处理器映射器、适配器、视图解析器）

## 工程结构



## 处理器映射器

在springmvc.xml中配置：

BeanNameUrlHandlerMapping： 根据请求url（XXXX.action）匹配spring容器bean的 name

找到对应的bean（程序员编写的Handler）

<!-- 根据bean的name进行查找Handler 将action的url配置在bean的name中 -->

<bean

class=*"org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"* />

所有处理器映射器都实现HandlerMapping接口。

## 处理器适配器

在springmvc.xml配置：

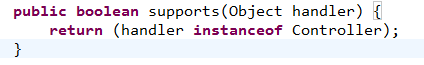
<bean class=*"org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"/*>

所有的适配器都是实现了HandlerAdapter接口。

程序员编写Handler根据适配器的要求编写。

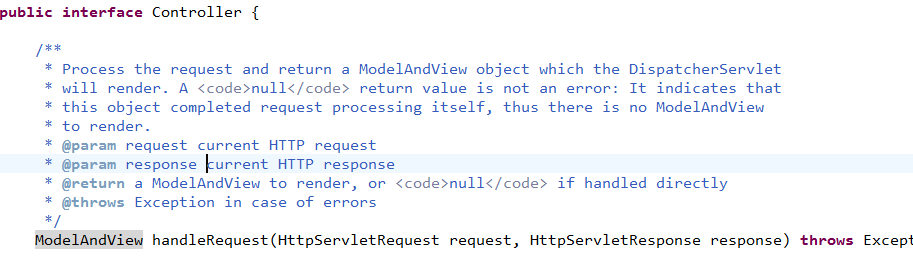
SimpleControllerHandlerAdapter适配器要求：

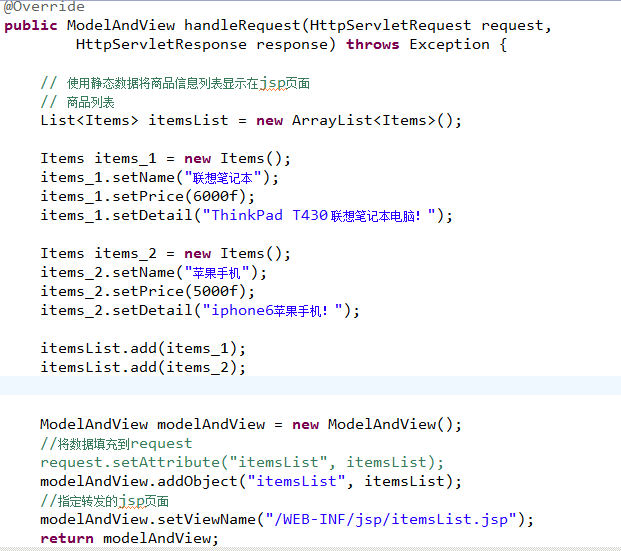
通过supports方法知道Handler必须要实现哪个接口：



## Handler编写

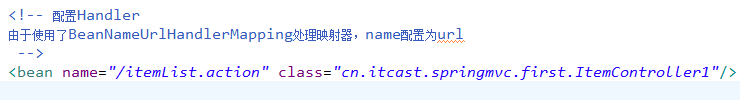
需要实现Controller接口：





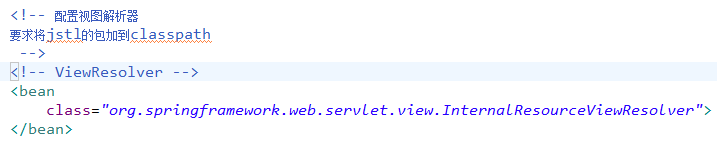
## 配置Handler

在springmvc.xml配置Handler由spring管理Handler。



## 配置视图解析器

配置视图解析，能够解析jsp视图：



## 工程部署

访问：http://localhost:8080/springfirst1110/itemList.action



## 异常

### HandlerMapping没有找到Handler

HandlerMapping没有找到Handler，404后边不显示jsp路径



### jsp页面地址错误

handlerMapping找到了Handler，但是页面找不到



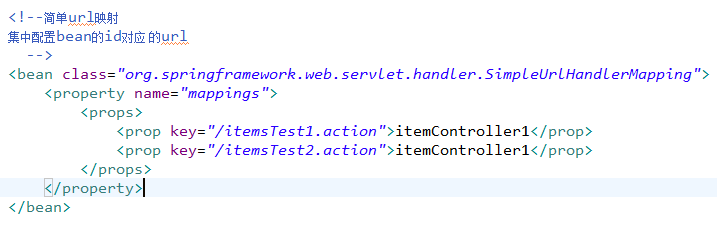
# 其它非注解处理器映射器和适配器

## BeanNameUrlHandlerMapping(映射器)

根据请求url（XXXX.action）匹配spring容器bean的 name

找到对应的bean（程序员编写的Handler）

## SimpleUrlHandlerMapping(映射器)



注意：在springmvc.xml配置了多个处理器映射器，**多个处理器映射器可以共存**。

## SimpleControllerHandlerAdapter(适配器)

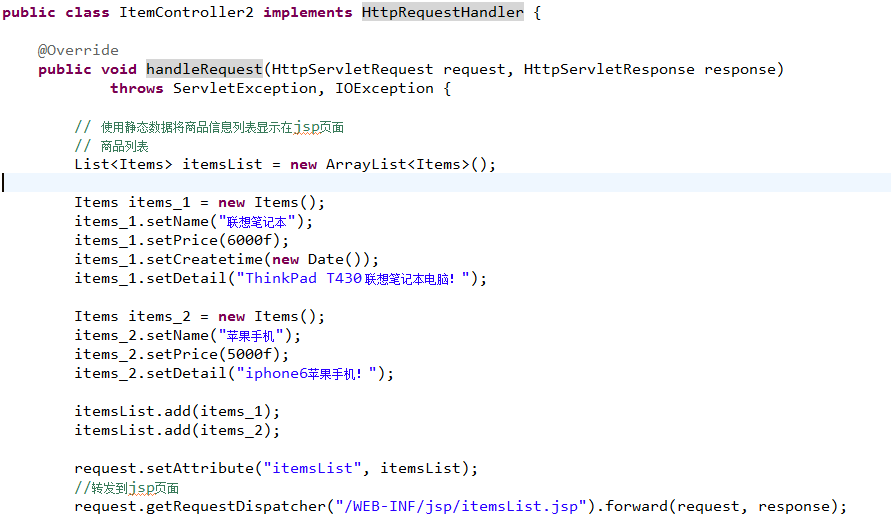
要求程序员编写的Handler(Controller)需要**实现 Controller接口**。

## HttpRequestHandlerAdapter（适配器）

在springmvc.xml配置：HttpRequestHandlerAdapter

要求Handler实现 **HttpRequestHandler接口**

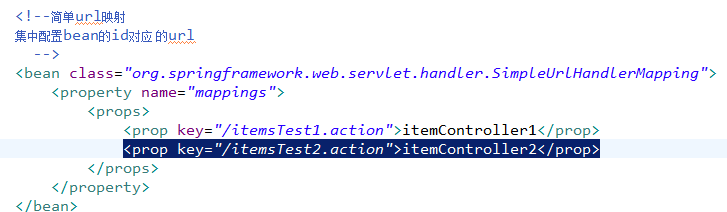
### 开发Handler



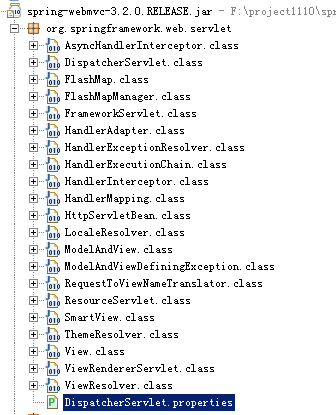
### 配置Handler



使用简单url映射器：



# DispatcherServlet.properoties



DispatcherServlet前端控制器加载 DispatcherServlet.properoties 配置文件，从而默认加载各各组件，

如果在springmvc.xml中配置了处理器映射器和适配器，以sprintmvc.xml中配置的为准

# 注解映射器和适配器

## 注解映射器

spring3.1之前默认加载映射器是

org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping，3.1之后要使用：

org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping

在springmvc.xml中配置RequestMappingHandlerMapping：

使用RequestMappingHandlerMapping需要在Handler 中

使用**@controller**标识此类是一个控制器

使用**@requestMapping**指定Handler方法所对应的url。

## 注解适配器

spring3.1之前默认加载映射器是

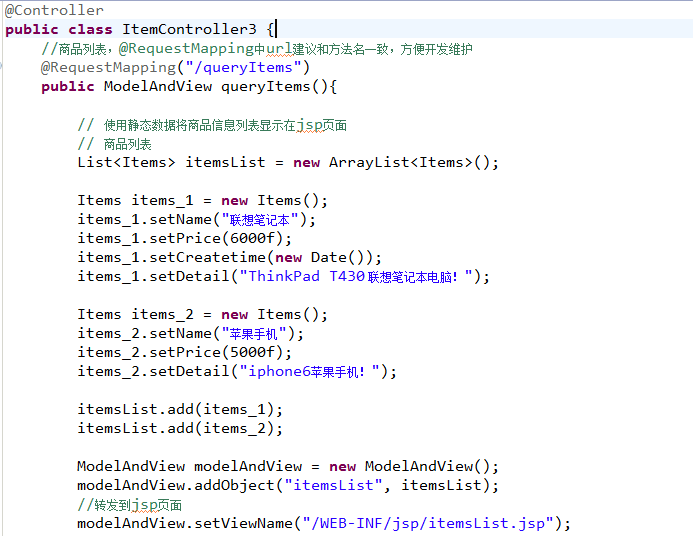
org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter，3.1之后要使用：

org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter

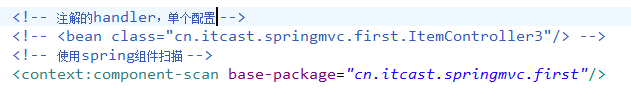
在springmvc.xml中配置RequestMappingHandlerAdapter：

RequestMappingHandlerAdapter，不要求Handler实现任何接口，它需要和RequestMappingHandlerMapping注解映射器配对使用，主要解析Handler方法中的形参（参数）。

## 注解开发Hanlder



## 配置Handler



建议使用组件扫描，**组件扫描可以扫描@Controller、@Service、@component、@Repsitory**

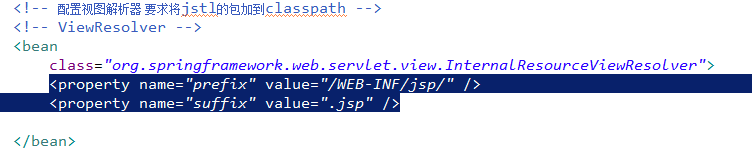
# 小结

DispatcherServlet：前端控制器，相当于中央调度器，可以降低组件之间的耦合度。

HandlerMapping：处理器映射器,负责根据url查找Handler

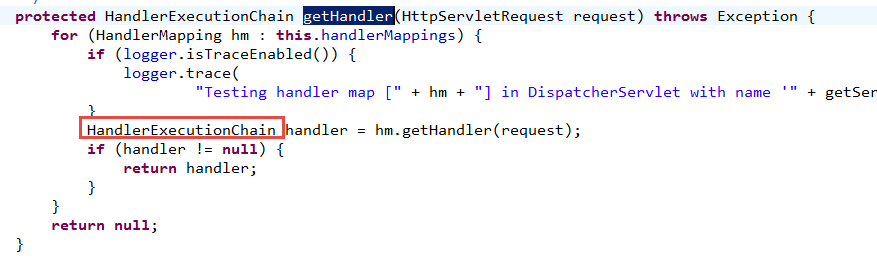
HandlerAdapter：处理器适配器，负责根据适配器要求的规则（需要实现什么接口，需要使用什么注解，一些开发规则规范）去执行处理器。可以通过扩展适配器支持不同类型的Handler。

viewResolver：视图解析器，根据逻辑视图名解析成真正的视图，



真正视图地址==前缀+逻辑视图名+后缀

1、DispatcherServlet通过HandlerMapping查找Handler

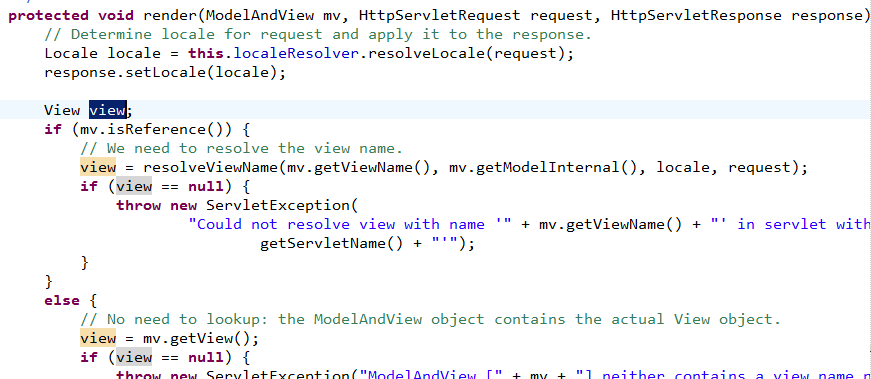


2、DispatcherServlet通过适配器HandlerAdapter去执行Handler，得到ModelAndview



3、视图解析

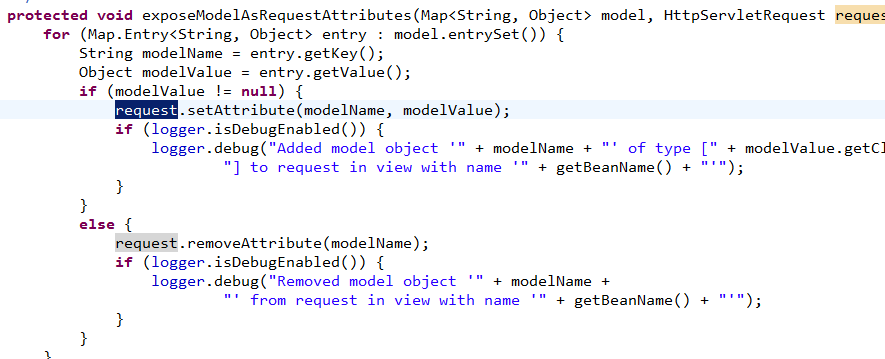
视图解析完成得到一个view：



4、进行视图渲染

将Model中的数据 填充到request域。





# springmvc和mybatis整合工程搭建

## 整合思路

在mybatis和spring整合的基础上 添加springmvc（由于springmvc和spring是一家人，不需要加其他乱七八糟的中间的包了，比如说整合包之类的）。

spring要管理springmvc编写的Handler（Controller）、mybatis的SqlSessionFactory、mapper

第一步：整合dao，spring和mybatis整合

第二步：整合service，spring管理service接口，service中可以调用spring容器中dao(mapper)

第三步：整合controller，spring管理controller接口，在controller调用service

## jar包

mybatis：3.2.7

spring：3.2.0

mybatis的jar

mybatis和spring整合包

spring的所有jar包(包括 springmvc的包)

数据库驱动包

log4j日志..



## 工程结构

### 配置文件

applicationContext-dao.xml---配置数据源、SqlSessionFactory、mapper扫描器

applicationContext-service.xml---配置service接口

applicationContext-transaction.xml--事务管理

sprintmvc.xml---springmvc的配置，配置处理器映射器、适配器、视图解析器

SqlMapConfig.xml---mybatis的配置文件，配置别名、settings、mapper

#### applicationContext-dao.xml

配置mybatis的数据源、sqlSessionFactory、mapper扫描器

#### applicationContext-transation.xml

事务有：注解事务控制和声明式事务控制

在此配置文件配置事务，声明式事务控制。

好处：可以有效的规范代码



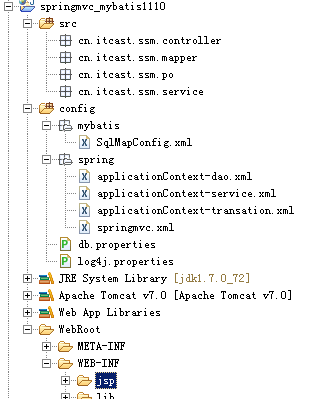
### 前端控制器配置



### 配置springmvc.xml



### 工程结构



## 商品列表开发

### 需求

查询商品列表

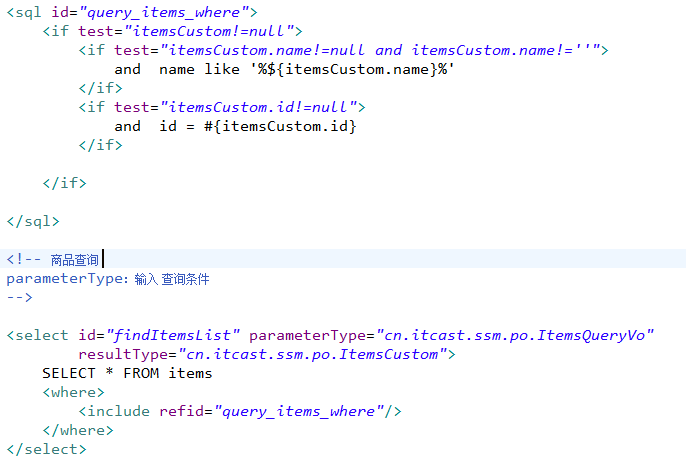
### mapper

功能描述：根据条件查询商品信息，返回商品列表

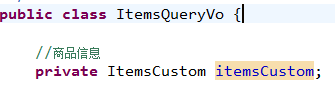
一般情况下针对综合查询mapper需要自定义mapper。

首先针对单表进行逆向工程，生成代码。

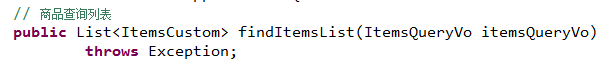
#### mapper.xml



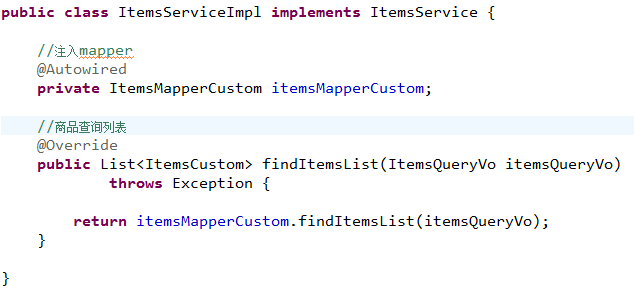
#### 包装类：



#### mapper.java



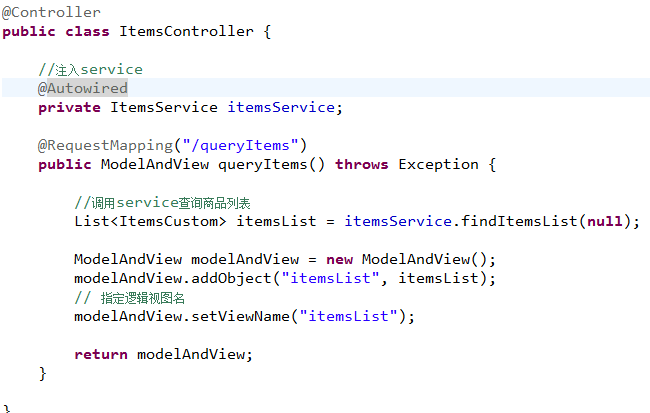
### service



### 在applicationContext-service.xml中配置service



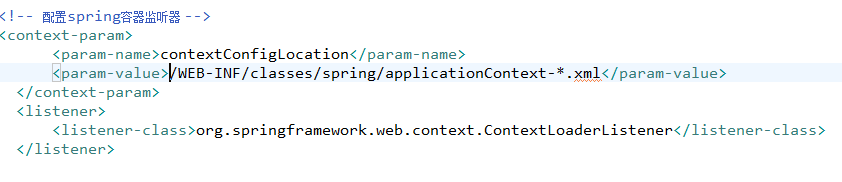
### controller



### jsp



### 在web.xml配置spring监听器



# 注解开发基础

## 商品修改

### 需求

功能描述：商品信息修改

操作流程：

1、在商品列表页面点击修改连接

2、打开商品修改页面，显示了当前商品的信息

根据商品id查询商品信息

3、修改商品信息，点击提交。

更新商品信息

### mapper

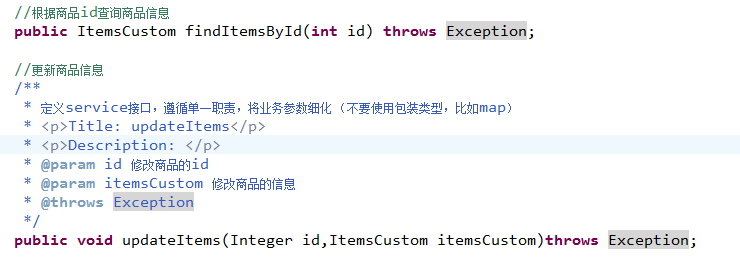
使用逆向工程生成代码：

根据商品id查询商品信息

更新商品信息



### service



## @RequestMapping

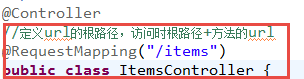
### 设置方法对应的url（完成url映射）

一个方法对应一个url



### 窄化请求映射

在class上定义根路径



好处：更好的规范系统 的url，避免 url冲突。

### 限制http请求的方法

通过requestMapping限制url请求的http方法，

没有指定时，默认post和get都可以，无限制

如果限制请求必须是post，如果get请求就抛出异常：

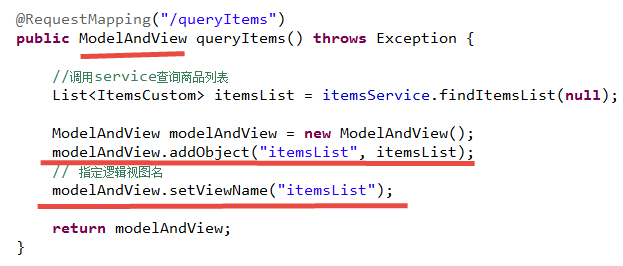


商品修改方法，限制为http的get：



## controller方法返回值

### 返回ModelAndView



### 返回字符串

如果controller方法返回jsp页面，可以简单将方法返回值类型定义 为字符串，最终返回逻辑视图名。



### 返回void



使用此方法，容易输出json、xml格式的数据：

通过response指定响应结果，例如响应json数据如下：

response.setCharacterEncoding("utf-8");

response.setContentType("application/json;charset=utf-8");

response.getWriter().write("json串");

### redirect重定向

如果方法重定向到另一个url，方法返回值为“redirect:url路径”

使用redirect进行重定向，request数据无法共享，url地址栏会发生变化的。

### forward转发

方法返回值为“forward:url路径”

使用forward进行请求转发，request数据可以共享，url地址栏不会发生变化。

## 参数绑定

### 参数绑定过程

http请求

HanlderAdapter(适配器)

Handler

以方法为单位进行编写

方法形参

key、value数据

早期springmvc是使用

PropertyEditor属性编辑器进行参数绑定(仅支持由字符串传为其它类型)

后期springmvc是使用

converter转换器进行参数绑定（支持任意类型转换）

将key/value数据绑定到方法形参

### 默认支持的参数类型

处理器形参中添加如下类型的参数处理适配器会默认识别并进行赋值。

#### HttpServletRequest

通过request对象获取请求信息

#### HttpServletResponse

通过response处理响应信息

#### HttpSession

通过session对象得到session中存放的对象

#### Model

通过model向页面传递数据，如下：

//调用service查询商品信息

Items item = itemService.findItemById(id);

model.addAttribute("item", item);

页面通过${item.XXXX}获取item对象的属性值。

### @RequestParam

作用：用于绑定单个请求参数（通常指request域中的参数）

如果request请求的参数名和controller方法的形参数名称一致，适配器自动进行参数绑定。如果不一致可以通过

@RequestParam 指定request请求的参数名绑定到哪个方法形参上。

对于必须要传入的参数，通过@RequestParam中属性required设置为true，如果不传此参数则报错。默认值为true

对于有些参数如果不传入，还需要设置默认值，使用@RequestParam中属性defaultvalue设置默认值。

### 可以绑定简单类型

可以绑定整型、字符串、单精/双精度、日期、布尔型。

### 可以绑定简单pojo类型

简单pojo类型只包括简单类型的属性。

绑定过程：

request请求的参数名称和pojo的属性名一致，就可以绑定成功。

问题：

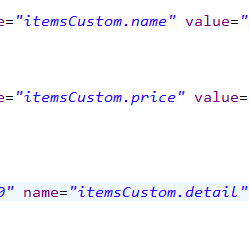
如果controller方法形参中有多个pojo且pojo中有重复的属性，使用简单pojo绑定无法有针对性的绑定，

比如：方法形参有items和User，pojo同时存在name属性，从http请求过程的name无法有针对性的绑定到items或user（就乱套了）。

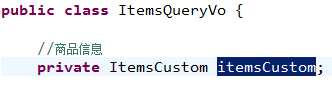
### 可以绑定包装的pojo

包装的pojo里边包括了pojo。

页面参数定义：



包装类型的属性也是itemsCustom：



按照上边的规则进行包装类型的绑定。

### 自定义绑定

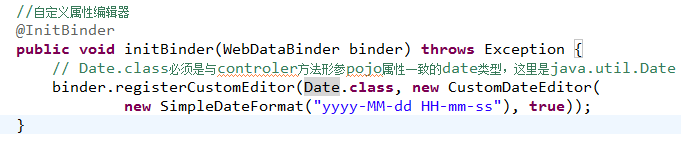
如：springmvc没有提供默认的对日期类型的绑定，需要自定义日期类型的绑定。

当然也可以自定义其他的

#### 自定义绑定使用属性编辑器(早期：PropertyEditor)

##### 使用WebDataBinder（了解）

在controller类中定义：



使用这种方法问题是无法在多个controller共用。

可以在父类中写这个方法然后继承，但是不推荐，因为尽量避免这种情况，而且只能单继承，以后万一还有其他的东西继承怎么办呢

##### 使用WebBindingInitializer（了解）

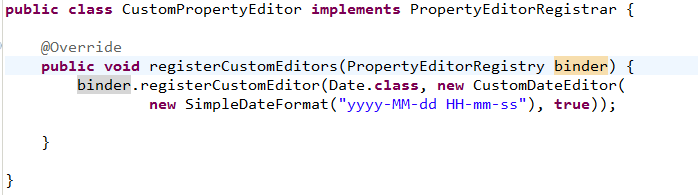
使用WebBindingInitializer让多个controller共用属性编辑器。

自定义WebBindingInitializer，注入到处理器适配器中。

如果想多个controller需要共同注册相同的属性编辑器，可以实现PropertyEditorRegistrar接口，并注入webBindingInitializer中。

如下：

编写CustomPropertyEditor：

配置如下：

<!-- 注册属性编辑器 -->

<bean id=*"customPropertyEditor"* class=*"cn.itcast.ssm.propertyeditor.CustomPropertyEditor"*></bean>

<!-- 自定义webBinder -->

<bean id=*"customBinder"*

class=*"org.springframework.web.bind.support.ConfigurableWebBindingInitializer"*>

<property name=*"propertyEditorRegistrars"*>

<list>

<ref bean=*"customPropertyEditor"*/>

</list>

</property>

</bean>

<!--注解适配器 -->

<bean

class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"*>

<property name=*"**webBindingInitializer"* ref=*"customBinder"*></property>

</bean>

#### 自定义参数绑定使用转换器(架构师掌握 后期：Converter)

##### 实现Converter接口：

定义日期类型转换器和字符串去除前后空格转换器。



##### 配置转换器

参考教案：

配置方式1针对不使用<mvc:annotation-driven>

配置方式2针对使用<mvc:annotation-driven>的配置

## 问题处理

### post乱码

在web.xml中加入：

<filter>

<filter-name>CharacterEncodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>utf-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CharacterEncodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

以上可以解决post请求乱码问题。

对于get请求中文参数出现乱码解决方法有两个：

修改tomcat配置文件添加编码与工程编码一致，如下：

<Connector URIEncoding="utf-8" connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>

另外一种方法对参数进行重新编码：

String userName new

String(request.getParamter("userName").getBytes("ISO8859-1"),"utf-8")

ISO8859-1是tomcat默认编码，需要将tomcat编码后的内容按utf-8编码

springmvc第二天 注解开发高级知识

# 复习

springmvc框架：

用户请求url到DispatcherServlet前端控制器，相当于中央调度器，降低系统各组件之间耦合度（利于增强可扩展）

DispatcherServlet前端控制器通过HandlerMapping处理器映射器根据url找到Handler。

DispatcherServlet前端控制器通过HandlerAdapter处理器适配器执行Handler。

DispatcherServlet前端控制器拿着Handler返回的ModelAndView通过视图解析器ViewResolver去进行视图解析。

视图解析：将程序中写的逻辑视图名，转成真正的视图（springmvc通过view表示各个不同类型的视图）。

DispatcherServlet前端控制器调用View的渲染方法进行视图渲染（将ModelAndView中的Model放到request域，填充到视图）。

要掌握springmvc的注解开发，企业中常用springmvc注解开发。

使用专门注解处理器映射器（RequestMappingHandlerMapping）和处理器适配器（RequestMappingHandlerAdapter）。

<mvc:annotation-driven/>可以代替上边的处理器映射器和适配器的配置。

在Handler(Controller)中定义很多的方法，一个方法通过RequestMapping和url进行映射。

方法返回值：ModelAndView、string(jsp的逻辑视图名)、void（通过response将数据输出成json）

方法输入参数（形参）-参数绑定：springmvc需要将请求的key/value(串，id=001&type=t002)、解析、绑定到Handler(Controller)中方法的形参上，将请求的内容绑定到参数。

springmvc默认支持很多种类型的参数绑定。

默认支持哪些类型：

HttpServletRequest、response、session、Model(用于将数据填充到request域)

@requestParam注解：用于绑定单个请求参数，常用于简单类型参数（Integer、String 、Float。。。）绑定。

不用@requestParam要求请求参数的名称和方法形参名一致方可绑定。

对于简单类型参数中的日期型，建议使用自定义参数绑定，对日期型数据个性化定义日期的格式。

自定义参数绑定：建议使用Converter进行参数绑定。

还可以绑定pojo、包装的pojo。

# 课程安排

注解开发：

数据回显：表单提交出现错误，重新回到表单，用户重新填写数据，刚才提交的参数在页面上回显。

集合类型（String[]、List<>、map(自学)）的参数绑定。

springmvc上传图片（重点）

json数据交互（提交json数据、响应json数据）（重点）

**Validation(springmvc使用校验方式，使用Hibernate Validator（和Hibernate的ORM没有任何关系）)**

**异常处理器（可以用于系统的统一异常处理，架构的内容）**

springmvc提供RESTful支持

拦截器（用于权限控制）

学生练习

# 数据回显

## 需求

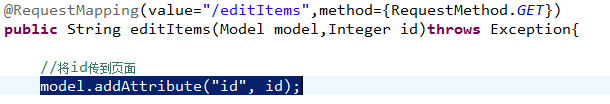
表单提交出现错误，重新回到表单，用户重新填写数据，刚才提交的参数在页面上回显。

## 对简单类型的数据回显

对商品修改数据回显：

注意在进入修改页面的controller方法中和提交修改商品信息方法中**model.addAttribute**方法设置的key一致。

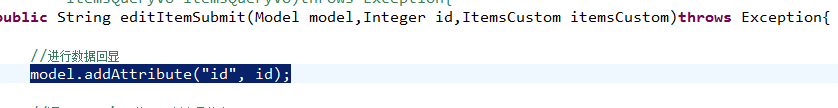
修改商品显示方法：



修改商品页面：



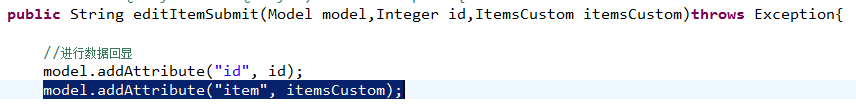
修改商品提交方法：



## pojo类型数据回显

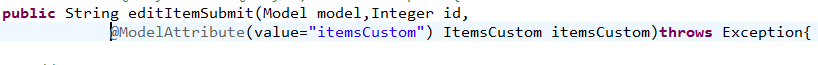
### 方法1：使用Model.addAttribute

使用Model.addAttribute方法进行数据回显：



### 方法2：使用@ModelAttribute

**使用@ModelAttribute，作用：**将请求pojo数据放到Model中回显到页面



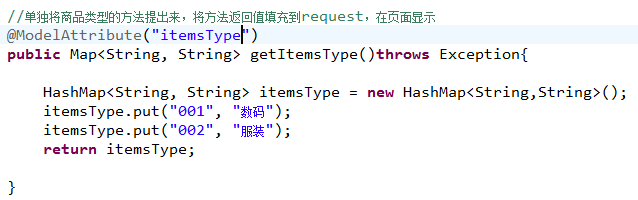
**在ModelAttribute方法指定的名称就是要填充到Model中的key，在页面中就要通过key取数据。**

有一种情况不用@ModelAttribute也行，只要上一次提交的key和这里形参名字一致也行，建议还是使用@ModelAttribute，因为这个可以指定名称，相对比较灵活（如果忘记怎么回事，可以参考springmvc第二天的视频第一集《1数据回显》）

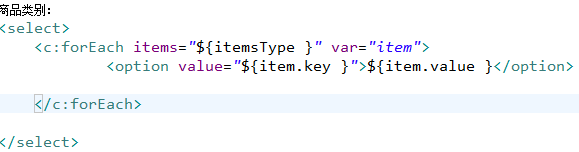
### @ModelAttribute将方法返回值传到页面

**@ModelAttribute还可以将方法的返回值填充到Model中（就是填充到request中一个意思），传到页面**

需求：将商品类别信息在商品信息查询页面显示。



页面：



**使用@ModelAttribute的好处及用法技巧（哪些方法需要用它）**

将公用的取数据的方法返回值传到页面，不用在每一个controller方法通过Model将数据传到页面（每次在每个方法中都调很麻烦）。

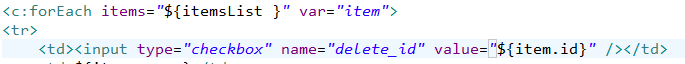
# 参数绑定集合类型

## 绑定数组

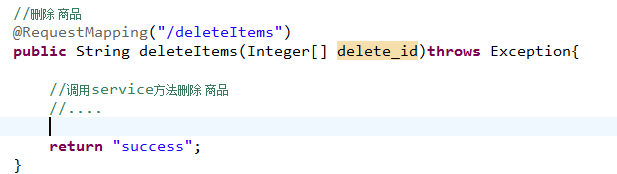
需求：在商品查询列表页面，用户选择要删除的商品，批量删除商品。

在controller方法中如何将批量提交的数据绑定成数组类型。

### 页面定义



### controller方法定义

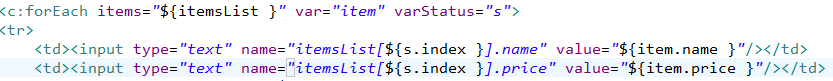


## 绑定List<Object>

需求：批量修改商品信息提交。

先进入批量修改商品页面，填写信息，点击提交。

### 页面定义

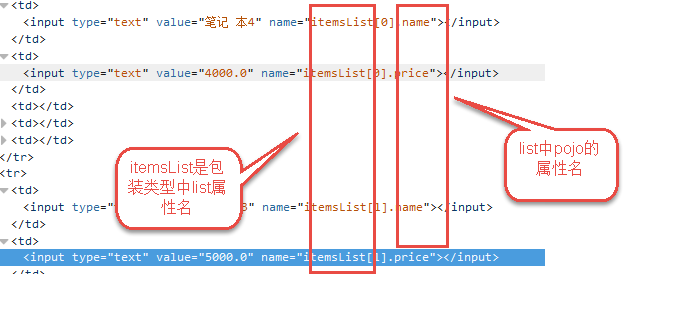


注释：

itemsList：controller方法形参包装类型中list的属性名。

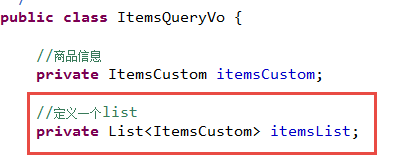
itemsList[0]或itemsList[1]...，[]中是序号，从0开始。

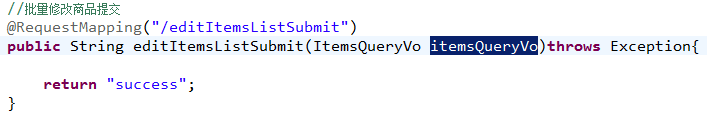
itemsList[].name：name就是controller方法形参包装类型中list中pojo的属性名

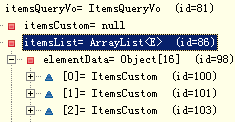


### controller方法定义

使用包装类型接收页面批量提交的数据，绑定成List。







# springmvc和struts的区别

springmvc是通过方法的形参接收参数，在使用时可以以单例方式使用，建议使用单例。

struts是通过成员变量接收参数，在使用时必须以多例方式使用。

springmvc是基于方法开发，struts基于类开发。

springmvc将一个请求的Method和Handler进行关联绑定，一个method对应一个Handler。

springmvc开发以方法为单位进行开发，方法更贴近service(业务方法)。

经过实际测试，发现struts标签解析速度比较慢，建议在实际开发时使用jstl。

# 商品图片上传

## 需求

在商品修改页面，增加图片上传的功能。

操作流程：

用户进入商品修改页面

上传图片

点击提交（提交的是图片和商品信息）

再次进入修改页面，图片在商品修改页面展示

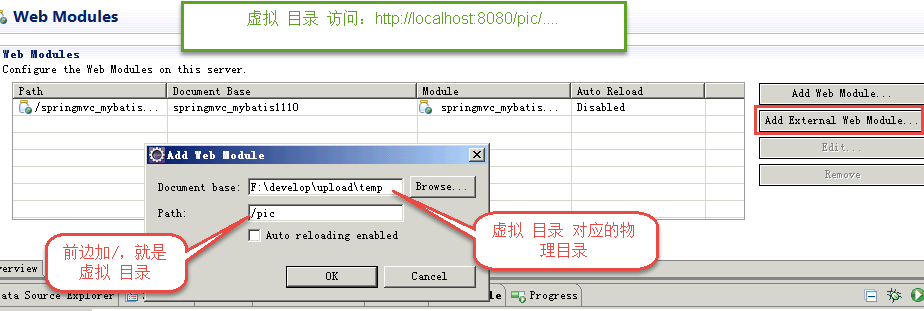
## 图片存储问题

切记：不要把图片上传到工程目录 ，不方便进行工程维护。

实际电商项目中使用专门图片服务器(http，比如apache、tomcat)。

本教程使用图片虚拟目录，通过虚拟目录 访问硬盘上存储的图片目录 。

虚拟目录 设置：



**注意：**

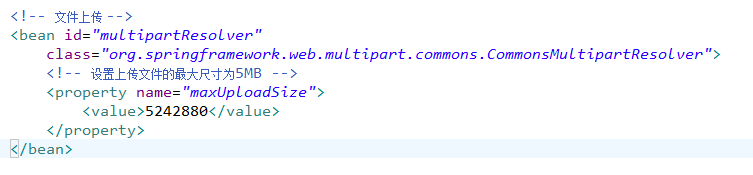
**图片目录中尽量进行目录分级存储，提高访问速度（提高I/O）。**

## 配置图片上传解析器

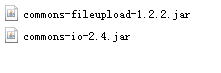
springmvc使用commons-fileupload进行图片上传。

commons-fileupload对应的springmvc的图片上传解析器：

*org.springframework.web.multipart.commons.CommonsMultipartResolver*



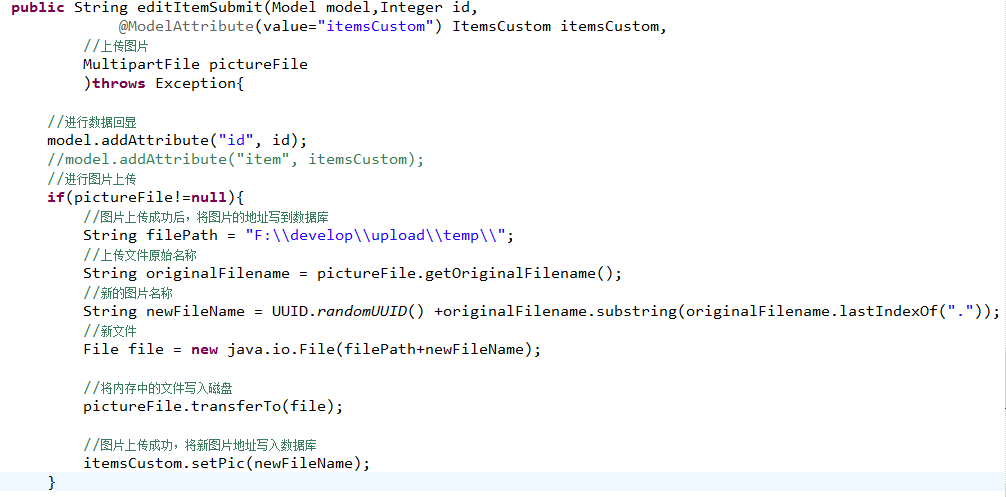
加入commons-fileupload的jar包



## 编写上传图片的页面



## 编写controller方法



# json数据的交互

## 需求

json数据格式是比较简单容易理解，json数据格式常用于远程接口传输，http传输json数据，非常方便页面进行提交/请求结果解析，对json数据的解析。

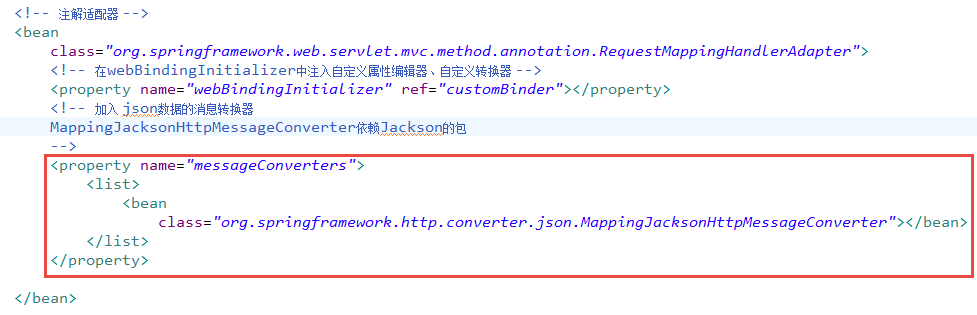
## springmvc解析json加入json解析包

Springmvc默认用MappingJacksonHttpMessageConverter对json数据进行转换，需要加入jackson的包，如下：



## 在处理器适配器中注入MappingJacksonHttpMessageConverter

让处理器适配器支持json数据解析，需要注入MappingJacksonHttpMessageConverter。



## @RequestBody和@ResponseBody

@RequestBody：将请求的json数据转成java对象。

@ResponseBody：将java对象转成json数据输出。

http请求数据是json格式（json串）

contentType=appliation/json;charset=utf-8

@ResponseBody方法返回值

controller方法形参（@RequestBody pojo对象）

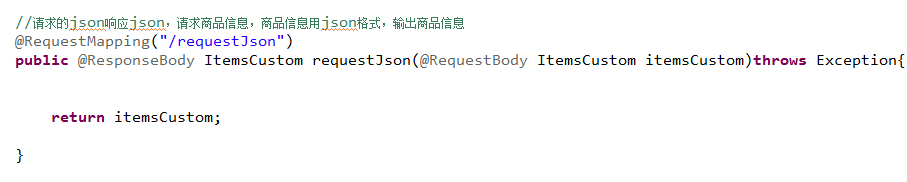
使用@RequestBody将json转成java对象(pojo)

@ResponseBody可以将java对象转成json输出

如果请求的是key/value就不需要使用@RequestBody

## 请求json响应json

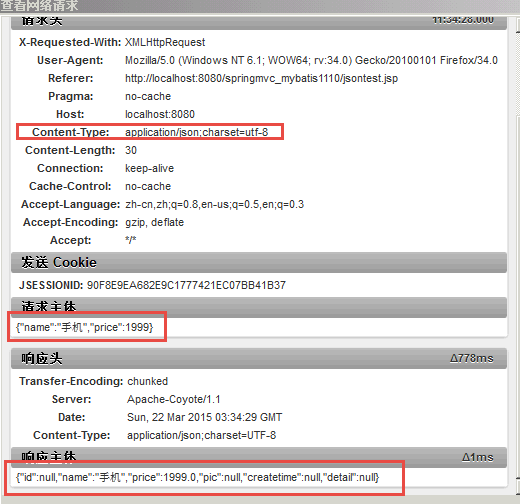
controller方法：



页面：

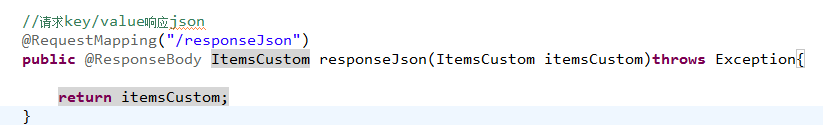


测试跟踪：

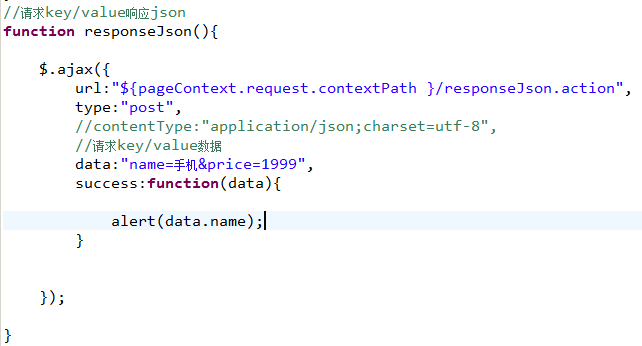


## 请求key/value响应json

controller方法：



页面：



测试：



## 小结

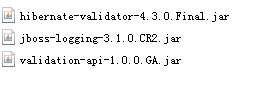
如果前端处理没有特殊要求建议使用第二种，请求key/value，响应json，方便客户端解析请求结果。

# validation校验（了解）

对前端的校验大多数通过js在页面校验，这种方法比较简单，如果对安全性考虑，还要在后台校验。

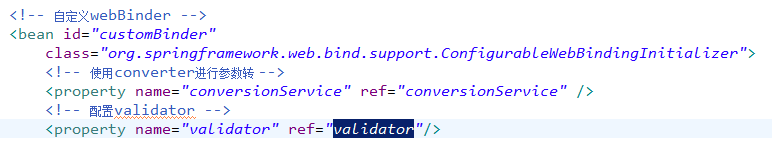
springmvc使用JSR-303（javaEE6规范的一部分）校验规范（一些接口，实现是各个厂家提供的），springmvc使用的是Hibernate Validator（和Hibernate的ORM不是一回事）

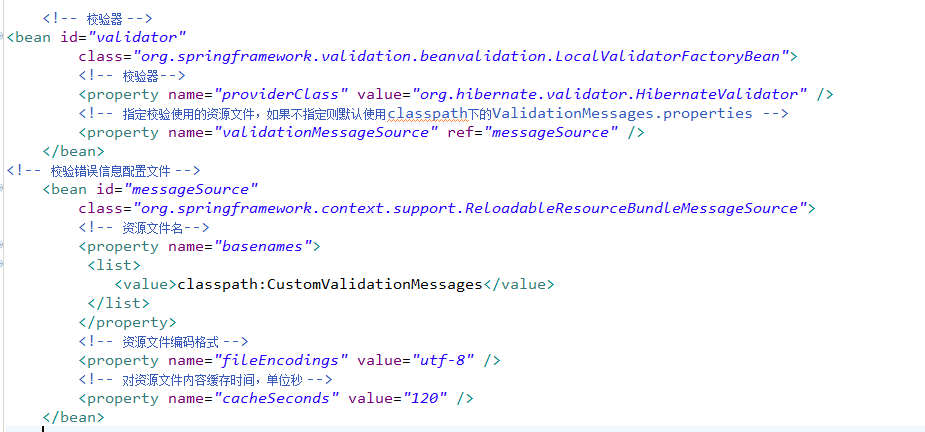
## 加入Hibernate Validator的jar



## 在处理器适配器中配置校验器

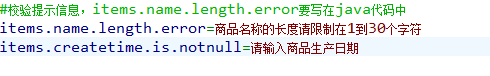






## 创建CustomValidationMessages

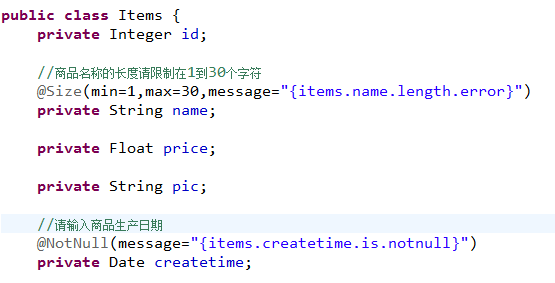
在classpath下创建CustomValidationMessages.properties



## 校验规则

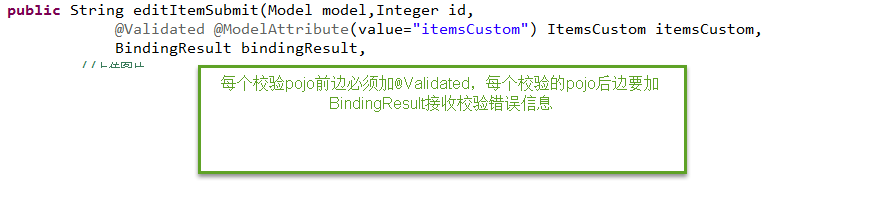
需求：

商品信息提交时校验 ，商品生产日期不能为空，商品名称长度在1到30字符之间

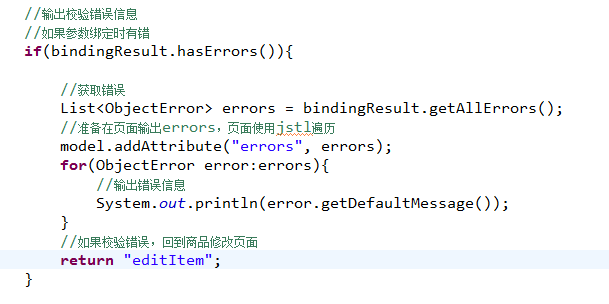


## 捕获错误

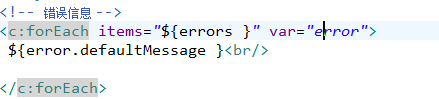
需要修改controller方法，在要校验的pojo前边加上@Validated，



错误信息输出：



## 在页面上展示错误



## 分组校验

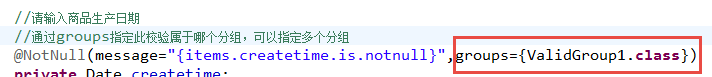
需求：

针对不同的controller方法通过分组校验达到个性化校验的目的，修改商品修改功能，只校验生产日期不能为空。

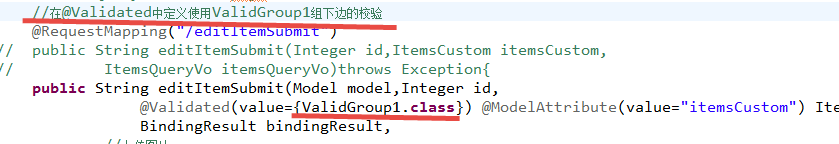
第一步：创建分组接口



第二步：定义校验规则属于哪个分组



第三步：捕获错误，在controller方法定义使用校验的分组



# 统一异常处理

## 需求

一般项目中都需要作异常处理，基于系统架构的设计考虑，使用统一的异常处理方法。

系统中异常类型有哪些？

包括预期可能发生的异常、运行时异常（RuntimeException），运行时异常不是预期会发生的。

针对预期可能发生的异常，在代码手动处理异常可以try/catch捕获，可以向上抛出。

针对运行时异常，只能通过规范代码质量、在系统测试时详细测试等排除运行时异常。

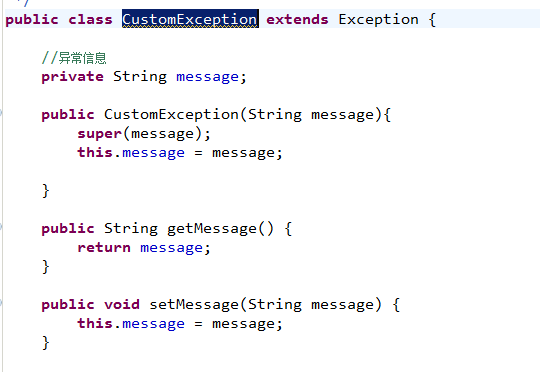
## 统一异常处理解决方案

### 定义异常

针对预期可能发生的异常，定义很多异常类型，这些异常类型通常继承于Exception。

这里定义一个系统自定义异常类：

CustomException，用于测试。

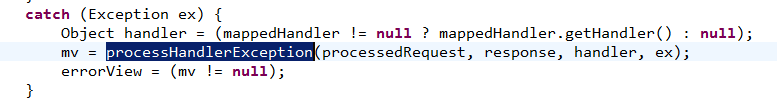


### 异常处理

要在一个统一异常处理的类中要处理系统抛出的所有异常，根据异常类型来处理。

统一异常处理的类是什么？

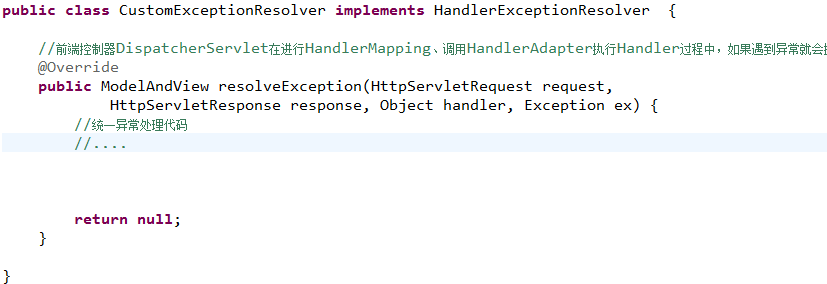
前端控制器DispatcherServlet在进行HandlerMapping、调用HandlerAdapter执行Handler过程中，如果遇到异常，进行异常处理。



在系统中自定义统一的异常处理器，写系统自己的异常处理代码。

#### 定义统一异常处理器类

统一异常处理器实现HandlerExceptionResolver接口。



#### 配置统一异常处理器



#### 异常处理逻辑

根据不同的异常类型进行异常处理。

系统自定义的异常类是CustomException ，在controller方法中、service方法中手动抛出此类异常。

针对系统自定义的CustomException异常，就可以直接从异常类中获取异常信息，将异常处理在错误页面展示。

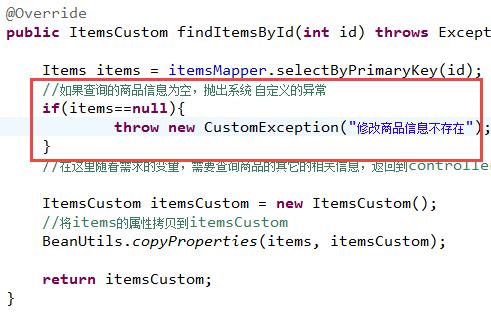
针对非CustomException异常，对这类重新构造成一个CustomException，异常信息为“未知错误”，此类错误需要在系统测试阶段去排除。

在统一异常处理器CustomExceptionResolver中实现上边的逻辑。



#### 测试抛出异常由统一异常处理器捕获

可以在controller方法、service方法、dao实现类中抛出异常，要求dao、service、controller遇到异常全部向上抛出异常，方法向上抛出异常throws Exception



图解：

http请求

DispatcherServlet

Handler(Controller)

service

mapper

左边的箭头就是请求处理的过程

有异常向上抛出

有异常向上抛出

有异常向上抛出

DispatcherServlet统一try/catch捕获异常调用异常处理器（全系统只有一个）

异常处理器需要实现HandlerExceptionResolver接口

要求dao、service、controller遇到异常全部向上抛出异常，方法向上抛出异常throws Exception

# RESTful支持

## 什么是RESTful

RESTful软件开发理念，RESTful对http进行非常好的诠释。

RESTful即Representational State Transfer的缩写。

综合上面的解释，我们总结一下什么是RESTful架构：

　　（1）每一个URI代表一种资源；

　　（2）客户端和服务器之间，传递这种资源的某种表现层；

　　（3）客户端通过四个HTTP动词，对服务器端资源进行操作，实现"表现层状态转化"。

## url的RESTful实现

非RESTful的http的url：http://localhost:8080/items/editItems.action?id=1&....

RESTful的url是简洁的：http://localhost:8080/items/editItems/1

参数通过url传递，rest接口返回json数据

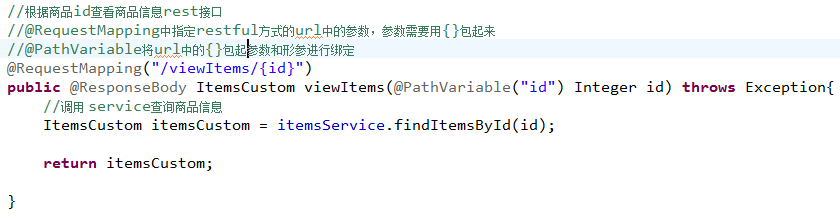
### 需求

根据id查看商品信息，商品信息查看的链接使用RESTful方式实现，商品信息以json返回。

### 第一步更改DispatcherServlet配置



### 第二步参数通过url传递

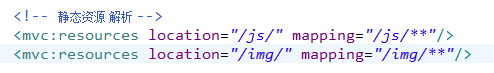


### 设置静态资源解析

当DispatcherServlet拦截/开头的所有请求，对静态资源的访问就报错：



需要通过设置对静态资源进行解析.



访问/js/\*\*的url从工程/js/下解析。

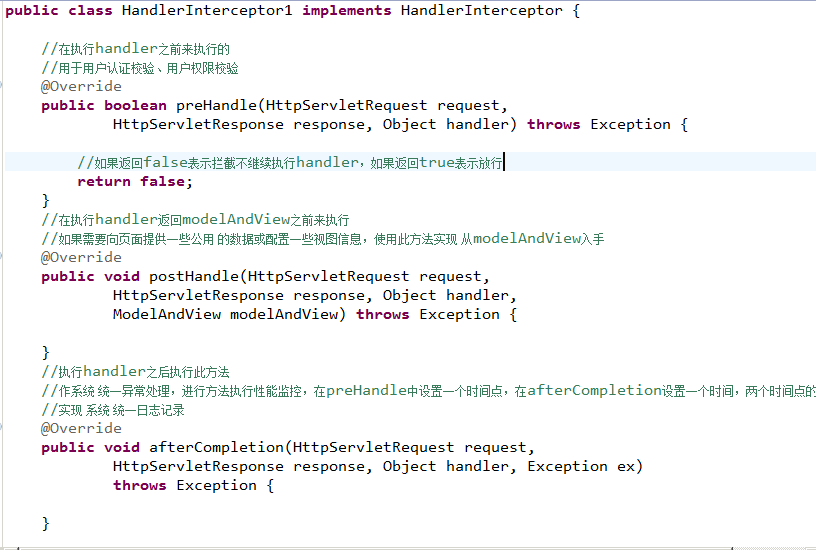
# springmvc拦截器

## 拦截器的应用场合

用户请求到DispatherServlet中，DispatherServlet调用HandlerMapping查找Handler，HandlerMapping返回一个拦截的链儿（多个拦截器），springmvc中的拦截器是通过HandlerMapping发起的。

在企业开发，使用拦截器实现用户认证（用户登陆后进行身份校验拦截），用户权限拦截。

## springmvc拦截器方法



## 测试拦截器

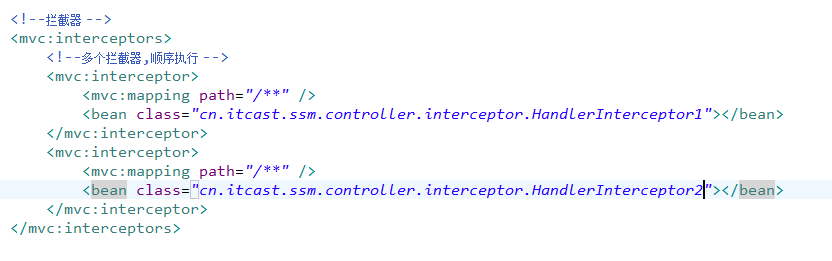
### 定义两个拦截器



### 配置拦截器

配置全局的拦截器，DispatcherServlet将配置的全局拦截器加载到所有的HandlerMapping。

在springmvc.xml中配置：



### 测试1 （1 号和2号都放行）

测试结果：

HandlerInterceptor1...preHandle

HandlerInterceptor2...preHandle

HandlerInterceptor2...postHandle

HandlerInterceptor1...postHandle

HandlerInterceptor2...afterCompletion

HandlerInterceptor1...afterCompletion

总结：

执行preHandle是顺序执行。

执行postHandle、afterCompletion是倒序执行

### 测试2 （1 号放行和2号不放行）

测试结果：

HandlerInterceptor1...preHandle

HandlerInterceptor2...preHandle

HandlerInterceptor1...afterCompletion

总结：

如果preHandle不放行，postHandle、afterCompletion都不执行。

只要链中有一个拦截器不放行，controller不能执行完成

### 测试3 （1 号不放行和2号不放行）

测试结果：

HandlerInterceptor1...preHandle

总结：

只有前边的拦截器preHandle方法放行，下边的拦截器的preHandle才执行。

### 日志拦截器或异常拦截器要求

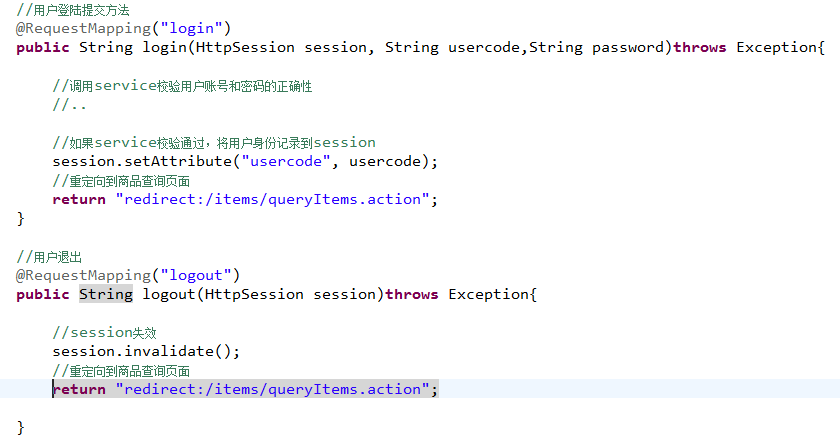
将日志拦截器或异常拦截器放在拦截器链儿中第一个位置，且preHandle方法放行

## 拦截器应用（用户认证拦截）

### 需求

用户访问系统的资源(url)，如果用户没有进行身份认证，进行拦截，系统跳转登陆页面，如果用户已经认证通过，用户可以继续访问系统 的资源。

### 用户登陆及退出功能开发



### 用户身份认证校验拦截器

拦截实现思路：

http请求url

如果url是公开地址

放行

如果用户在session中存在

放行

公开地址：

不需要认证即可访问的url

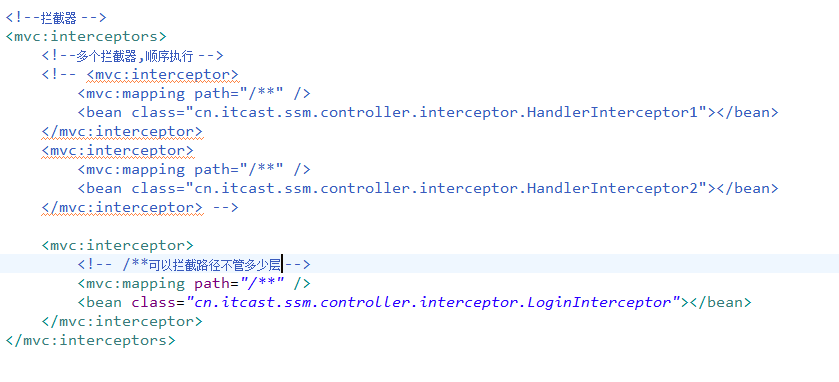
如果用户在session中不存在

登陆页面

### 拦截器



在springmvc.xml中配置拦截器：



# 学生练习

商品查询添加查询条件：商品名称、价格范围

添加商品功能开发

删除商品功能开发