SpringMVC笔记

# SpringMVC架构

第一天：SpringMVC的基础知识

什么是SpringMVC?

SpringMVC框架原理(掌握)

前端控制器、处理器映射器、处理器适配器、视图解析器

SpringMVC的入门程序

目的：对前端控制、处理器映射器、处理器适配器、视图解析器学习

非注解的处理器映射器、非注解的处理器适配器

注解的处理器映射器、 处理器适配器

SpringMVC和MyBatis的整合

SpringMVC注解开发(重点)

常用的注解学习

参数绑定(简单类型、POJO、集合类型)

自定义参数绑定

第二天： SpringMVC的高级应用

数据回显

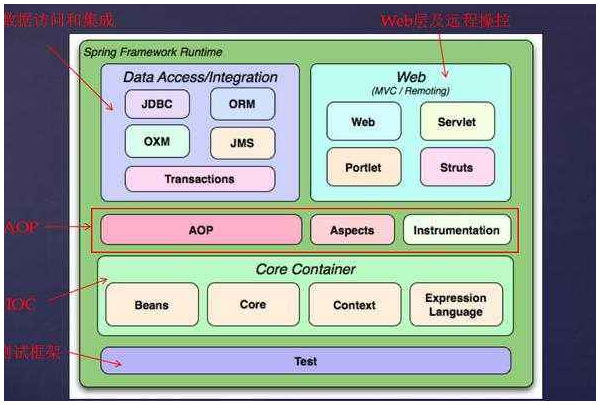
上传图片

JSON数据支持

RESTful支持

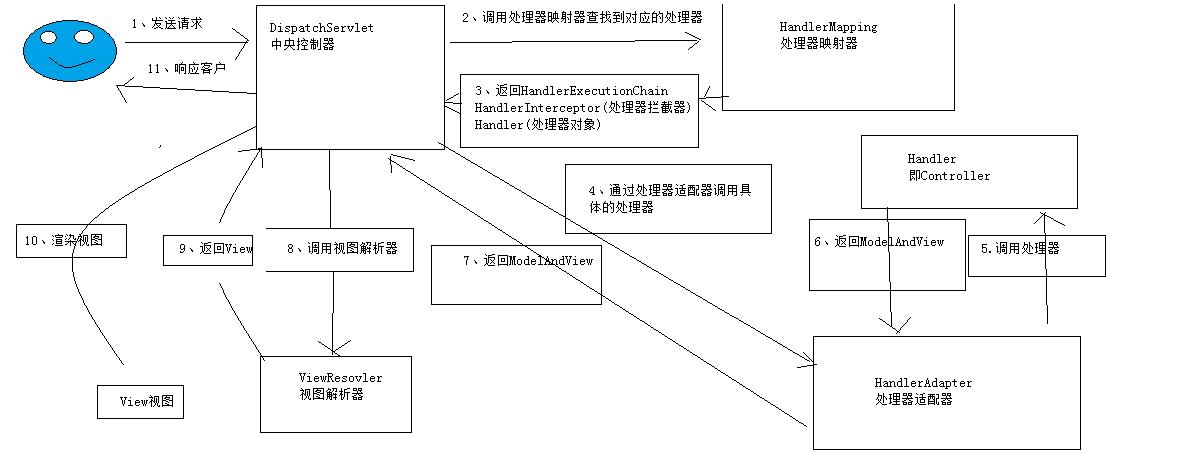
拦截器

## SpringMVC 介绍

SpringMVC是属于表现层的框架。它Spring框架的一部分  


## SpringMVC的结构

### 架构图



### 架构流程

1. 用户发送请求到前段控制器DispatchServlet
2. DispatchServlet接受到请求调用HandlerMapping处理器映射器
3. 处理器映射器根据请求的URL找到具体的处理器 生成处理器对象以及拦截器并 把所有的内容返回到DispatchServlet
4. DispatchServlet通过HandlerAdapter处理器适配器调用处理器
5. 执行处理器(Controller )
6. Controller执行完成 返回ModelAndView
7. HandlerAdapter将Controller执行的结果ModelAndView返回给DispatchServlet
8. DispatchServlet将ModelAndView传递给ViewReslover视图解析器
9. 视图解析器会返回具体的View
10. DispatchServlet对View进行渲染、将模型数据填充到视图
11. DispatchServlet响应客户

# SpringMVC入门

## SpringMVC入门案例

### 需求

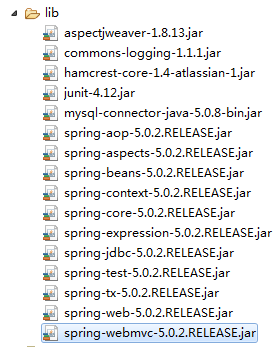
实现商品的查询列表

以商品订单管理为例

### 开发环境的准备

Spring5.0.2.RELEASE

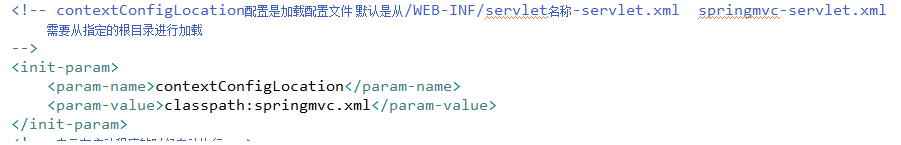
导入相关的jar包



### 配置前段控制器



### Springmvc配置文件



### 配置处理器映射器



### 配置处理器适配器

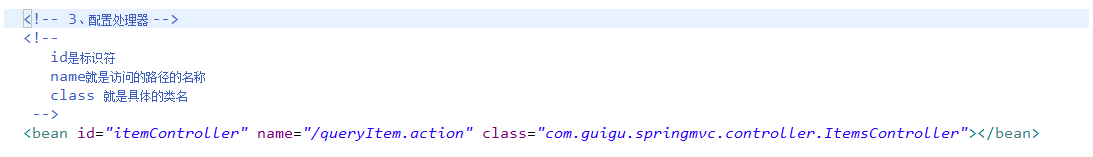


### 处理器的开发(Handler 即Controller)

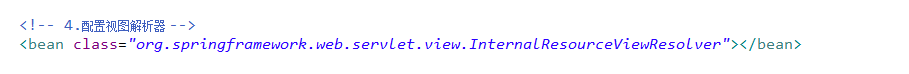




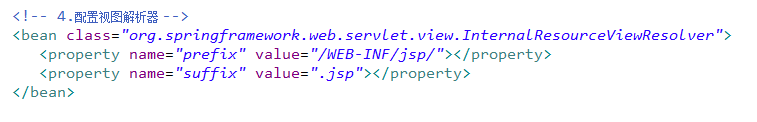
### 处理器配置



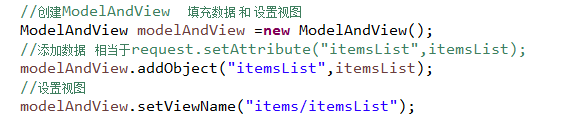
### 配置视图解析器



通过属性配置前缀和后缀



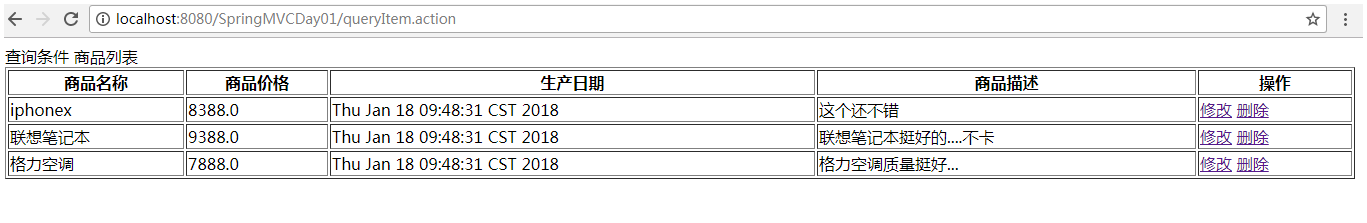
代码简化



### 视图开发



### 程序的测试



## 非注解方式的HandlerMapping处理器映射器

HandlerMapping负责根据request请求找到对应的Handler处理器以及拦截器 将他们封装到HandlerExecutionChain中 返回给中央处理器

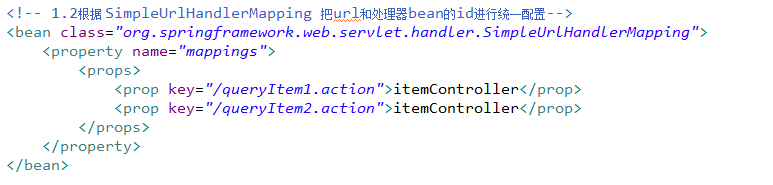
### BeanNameUrlHandlerMapping

根据请求的url与spring容器中的定义的bean的name属性进行匹配 从而从spring容器中 找到对应的bean实例



### SimpleUrlHandlerMapping

SimpleUrlHandlerMapping是BeanNameUrlHandlerMapping的增强版，他可以将url和处理器的bean的id进行统一的配置

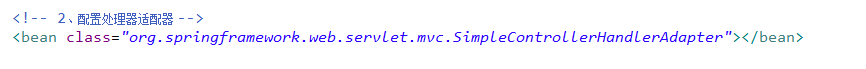


## 非注解方式的HandlerAdapter处理器适配器

HandlerAdatper会根据适配器接口后端控制器进行包装，包装后可以对处理器进行执行。

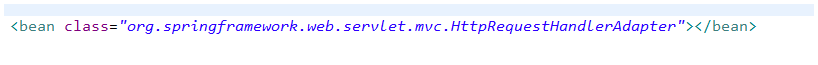
### SimpleControllerHandlerAdapter

SimpleControllerHandlerAdapter那么Controller类要实现Controller接口



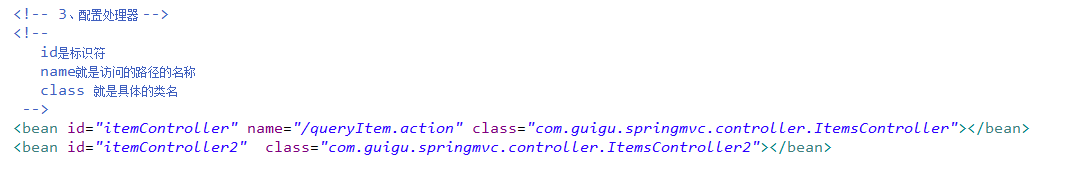
### HttpRequestHandlerAdapter

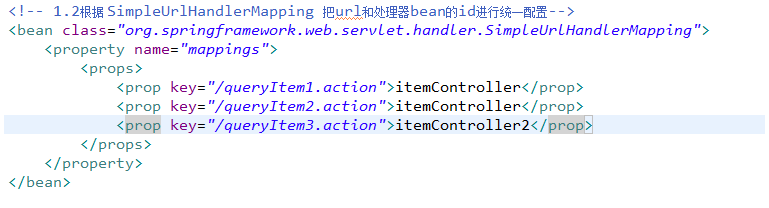
配置文件



Controller类

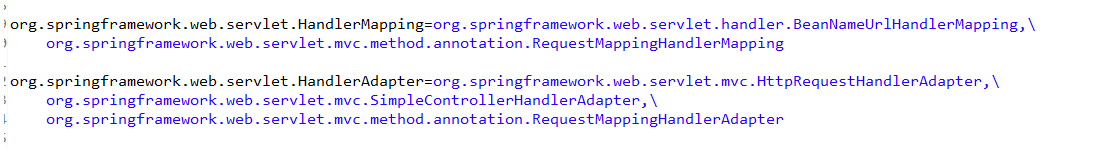
 





## 注解的映射器和适配器

中央处理器的配置文件的非注解形式的内容

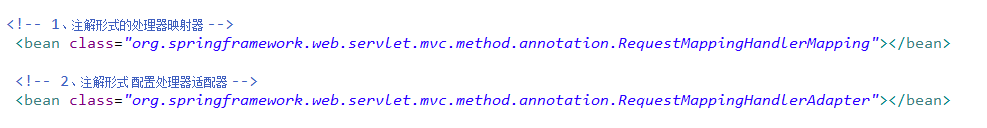


系统指定的映射器和适配器的注解方式

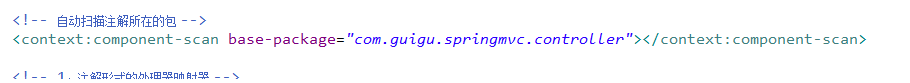
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping

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

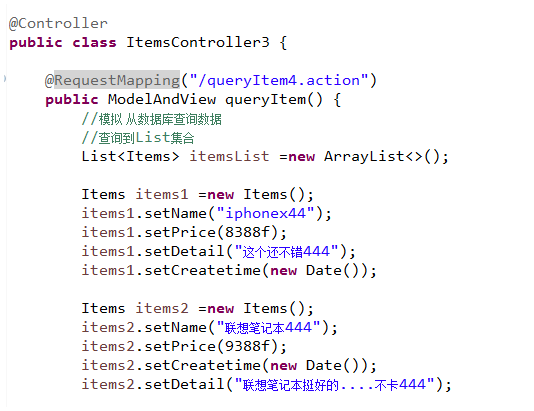
### 在配置文件中配置注解的适配器和映射器



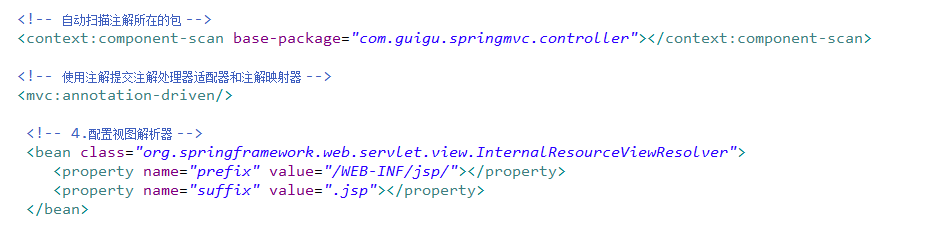
### 组件扫描



### Controller代码

### 使用mvc:annotation-driven 自动加载



# SpringMVC整合MyBatis

整合目标： 控制层采用SpringMVC、持久层使用MyBatis

## 需求

实现商品查询列表、是从mysql数据库查询商品信息

## 整合思路

第一步：整合DAO层

MyBatis和Spring整合 通过Spring管理Mapper接口  
 使用mapper的扫描器自动扫描mapper接口在spring中进行注册

第二步： 整合service层

使用spring管理service接口

使用配置方式将service接口配置在spring配置文件中 或者注解的形式

实现事务管理

第三步： 整合SpringMVC

由于SpringMVC本身就是Spring的一个模块 所以不需要整合

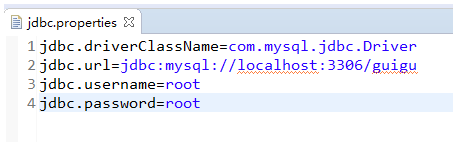
## Jar包

Spring的所需的基础jar包、MyBatis基础jar包、数据库驱动、数据库连接池、mybatis-spring整合的jar包

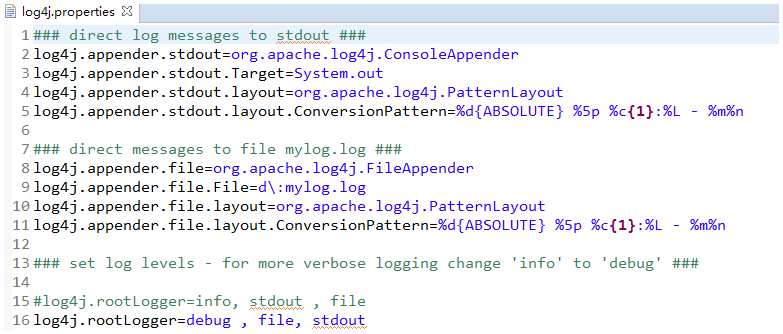
## DAO

目标： Spring管理 SqlSessionFactory 和Mapper

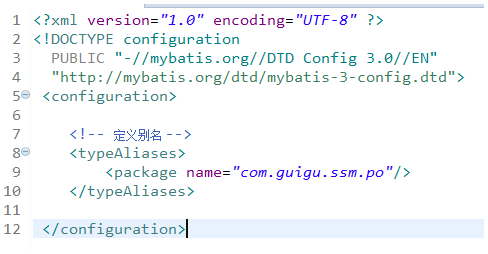
### db.properties



### log4j.properties



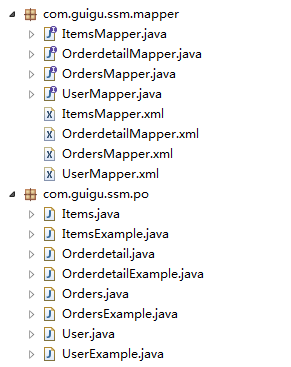
### SqlMapConfig.xml



### applicationContext-dao.xml



### 通过逆向工程生成PO类以及mapper(单表增删改查)

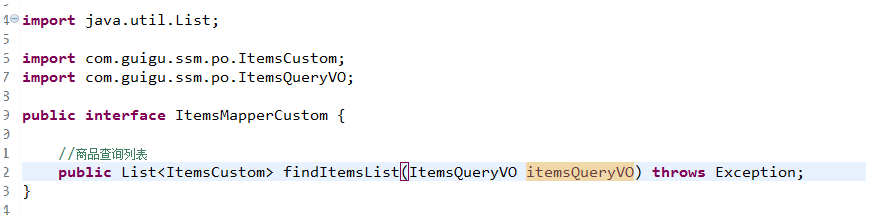


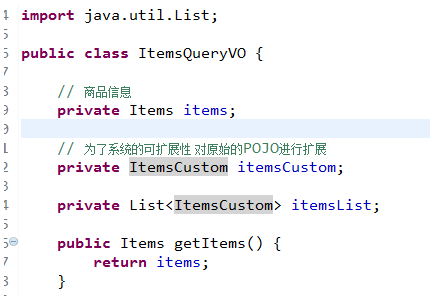
### 自定义ItemsMapper.xml

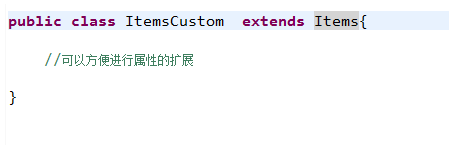
为了扩展进行自定义



### ItemsMapper.java





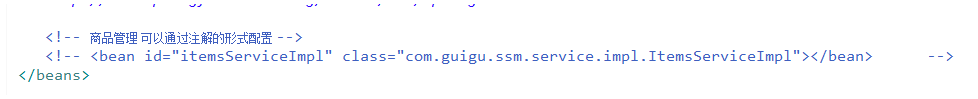


## Service

目标： service由spring进行管理

Spring对service进行事务控制

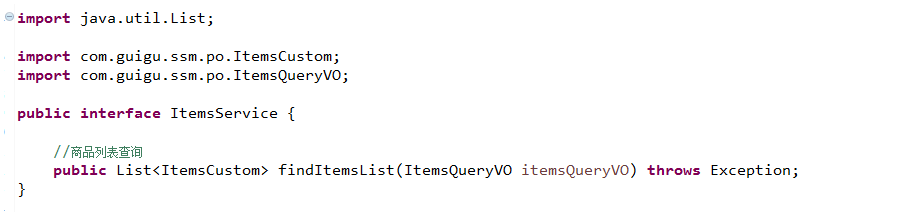
### applicationContext-service.xml

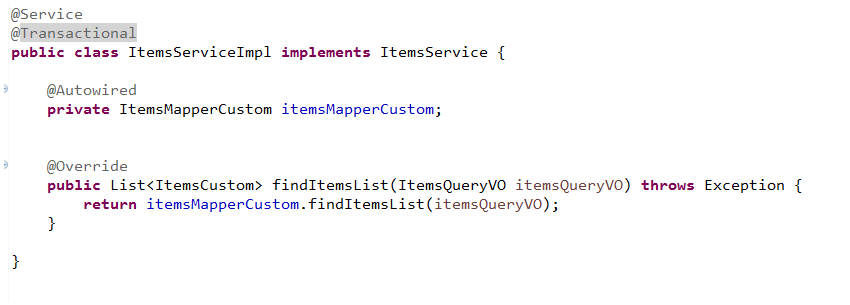


### applicationContext-transaction.xml



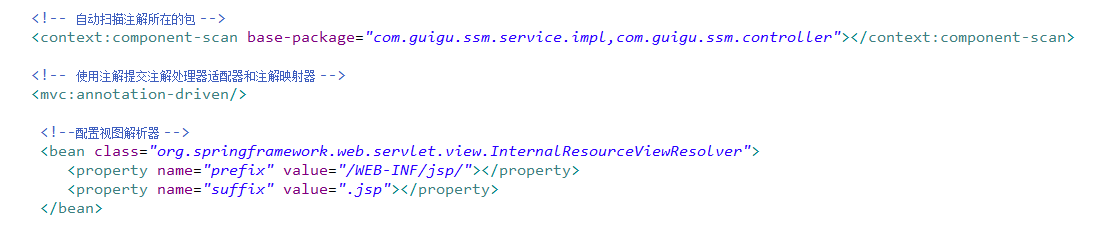
### ItemsService和ItemsServiceImpl





## Action(Controller)

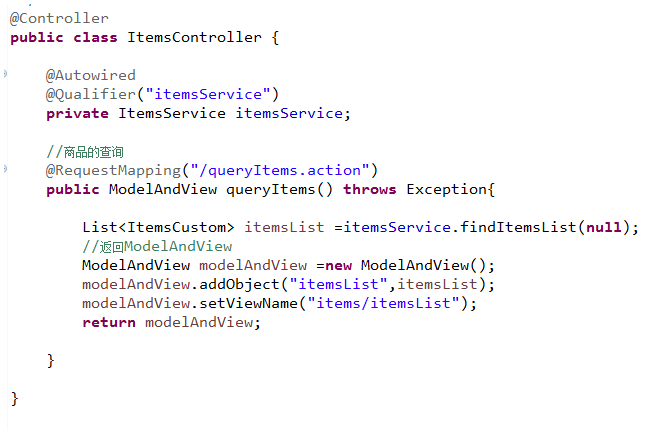
### Springmvc.xml



### Web.xml



### ItemsController



## JSP页面



# 注解开发—基础

## 需求

使用SSM架构完成商品信息维护

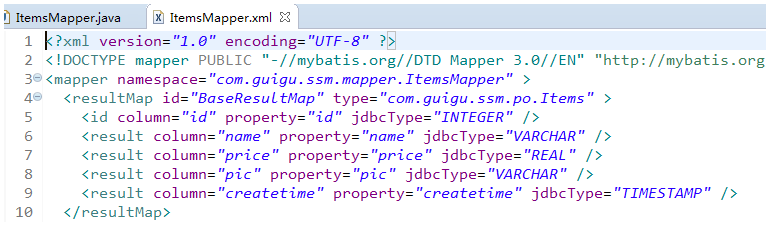
操作流程：

1. 进入商品查询列表页面
2. 点击修改 从数据库查询 页面显示要修改的内容 根据id查询 商品信息
3. 在商品修改页面 修改商品信息 修改完成后点击提交

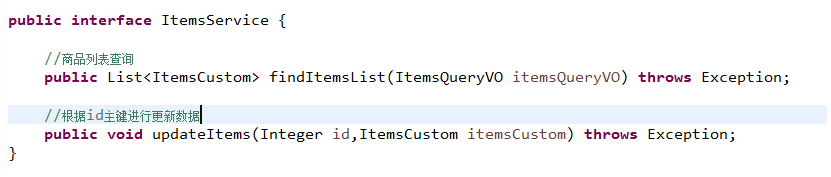
## 商品修改

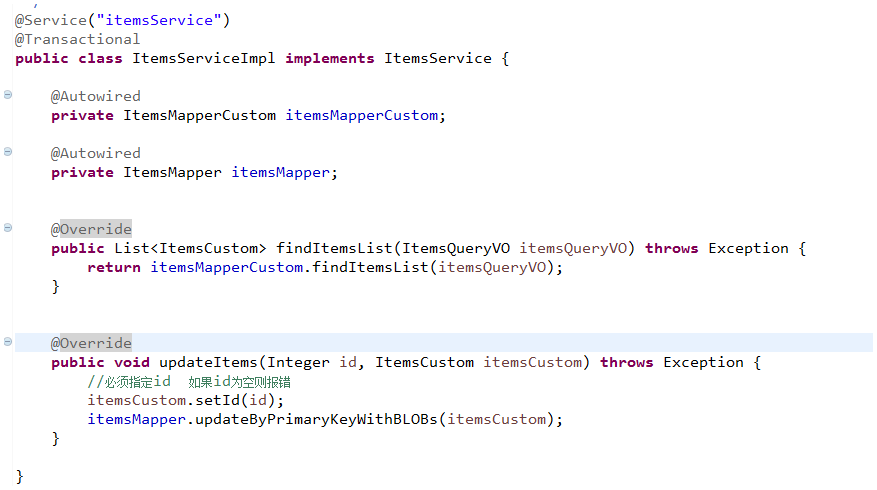
### Dao

通过逆向工程生成的ItemsMapper.java和ItemsMapper.xml

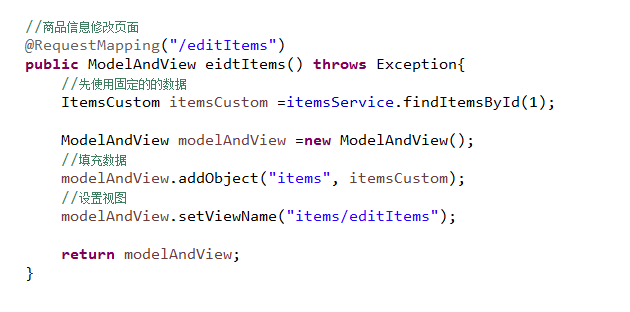


### Service





### Controller



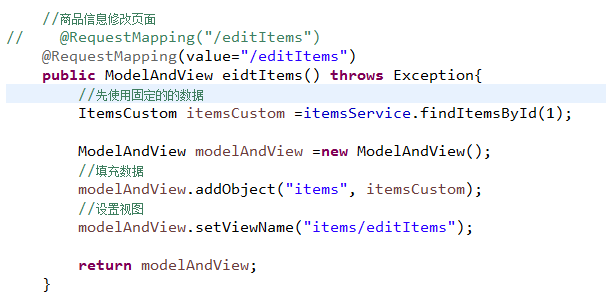
### Jsp页面



## @RequestMapping

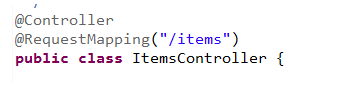
通过RequestMapping注解可以定义不同的处理器映射规则

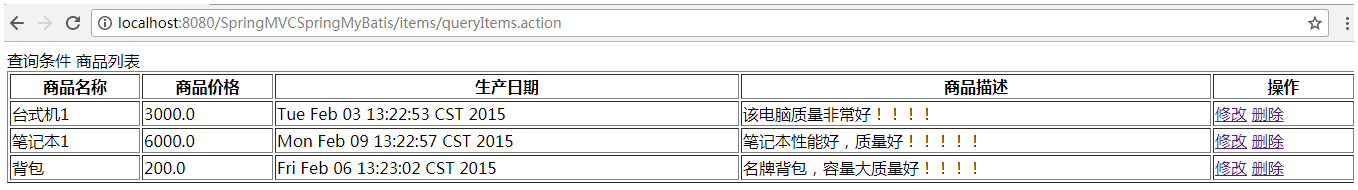
### URL路径映射



### 窄化请求映射

可以在类上添加前缀 限定访问路径信息



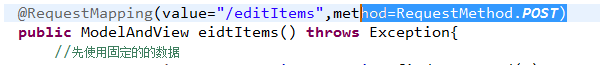


### 请求方法的限定

**限定GET方法**



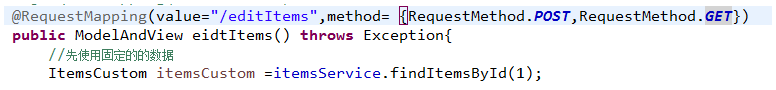
**限定POST方法**



如果使用get方式请求目标资源将会报错 405

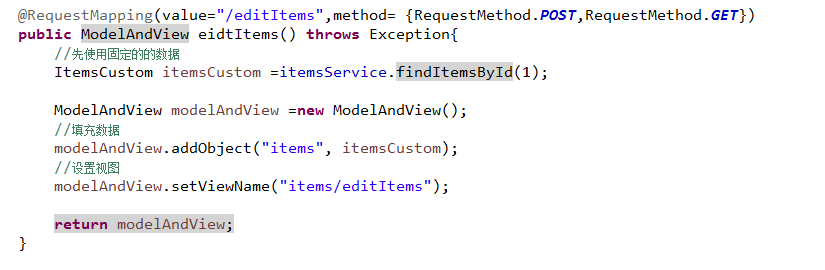


**GET和POST两种形式都可以**

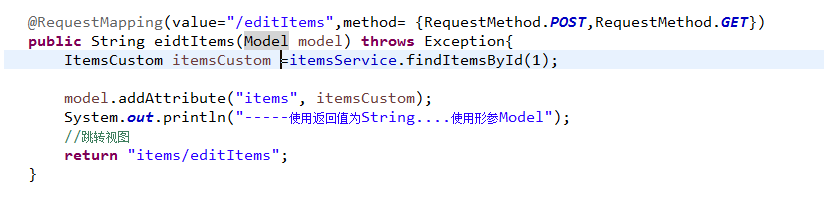


## Controller方法返回值

### 返回ModelAndView

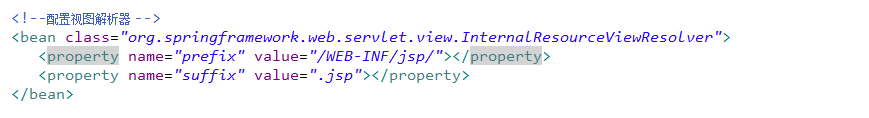


### 返回字符串



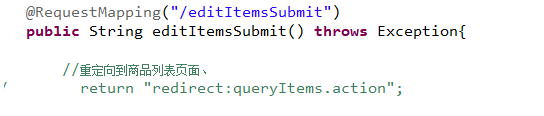
#### 逻辑视图





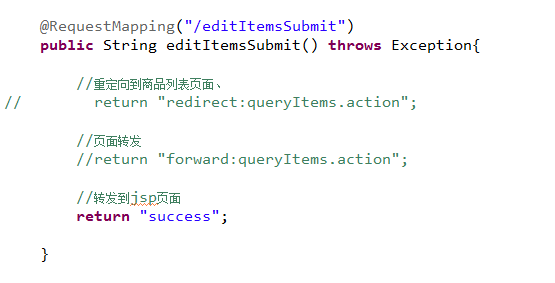
**实际转发的地址是/WEB-INF/jsp/items/editItens.jsp**

#### Redirect 重定向

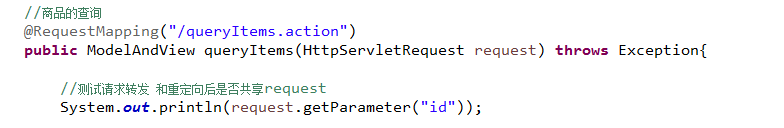


**通过重定向 request不是共享的 当修改提交以后再提交的Controller中无法获取相关的数据**

#### Forward转发



**通过转发 request是共享的 可以获取相关的数据**



## 参数绑定

注解适配器对RequestMapping标记的方法进行适配，对方法中的形参进行绑定

### 默认支持的参数类型

直接在方法的形参上定义类型对象，就可以使用这些对象。

#### HttpServletRequest

#### HttpServletResponse

#### HttpSession

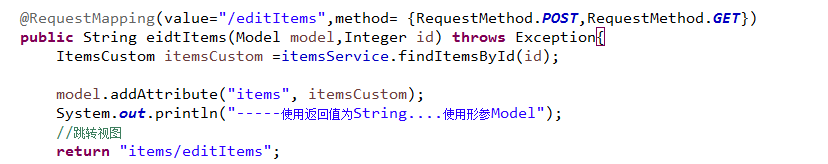
#### Model/ModelMap

ModelMap是Model接口的实现类 通过Model或者ModelMap可以直接向页面传递数据

### 简单类型

当前**请求的参数和处理器形参名称一致的时候 将会把请求参数和形参进行绑定**

#### 整形

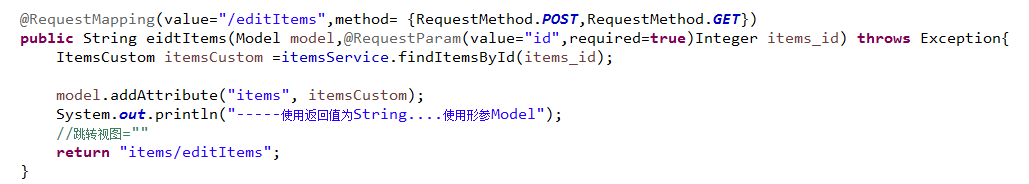


#### 字符串

#### 布尔型

#### 双精度/单精度

### @RequestParam



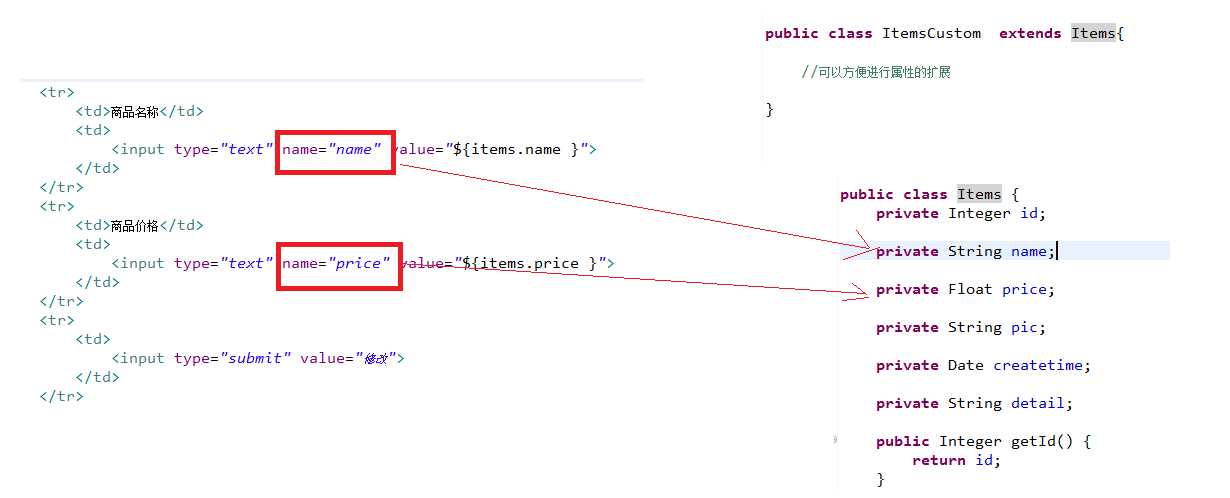
Value 是请求参数的名称 和当前的形参进行绑定

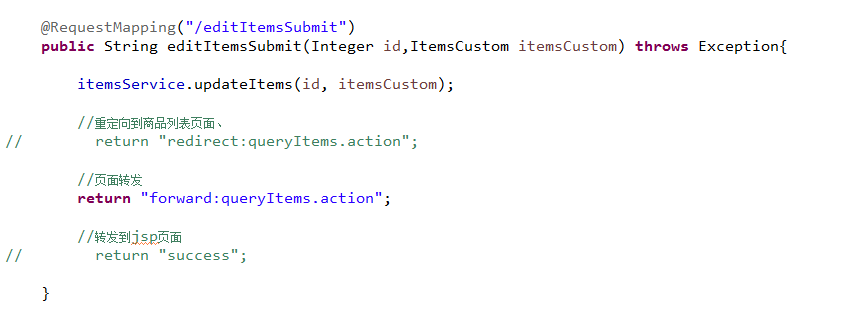
Required 是否必须要有 如果是true 将必须传递 否则程序报错

### POJO

#### 简单的POJO绑定

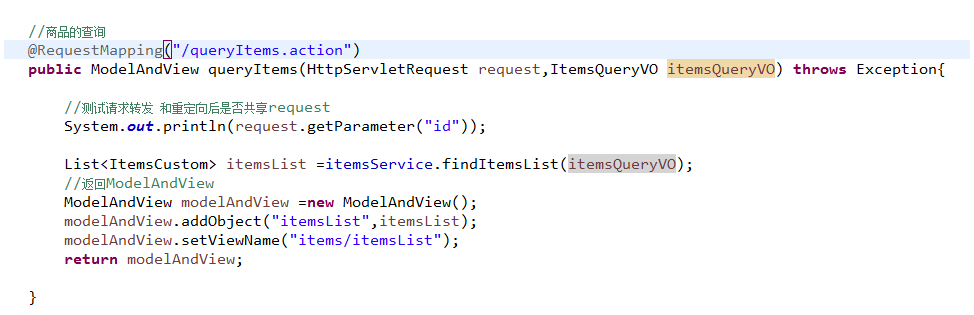
将POJO中的属性和对应的参数进行绑定





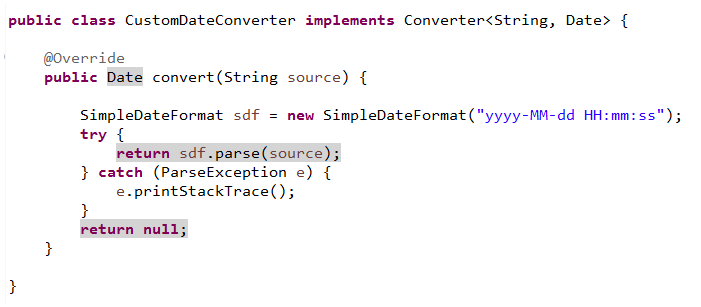
#### 包装POJO



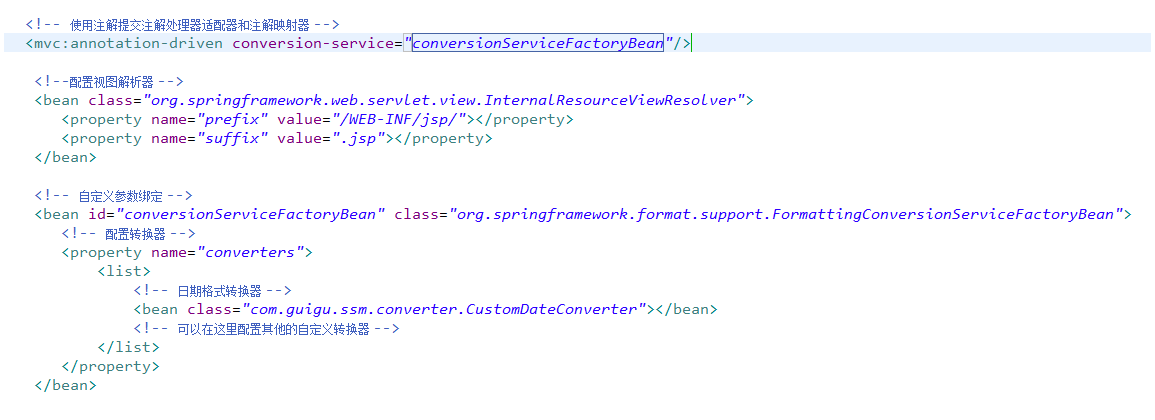


### 自定义参数绑定

#### 自定义转换器



#### 配置在springmvc.xml



### 集合类型的参数绑定

**需求**

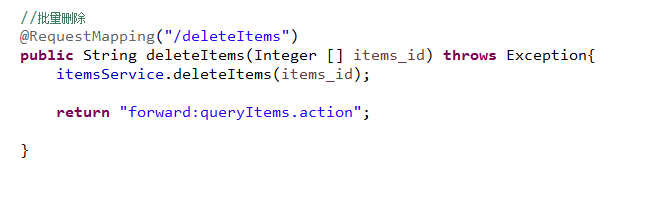
批量删除商品信息

**Jsp页面的修改**

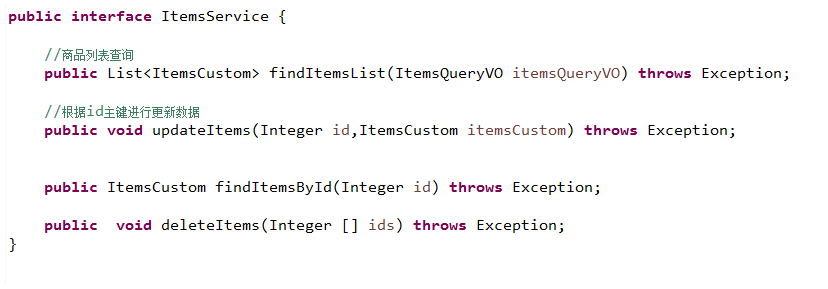




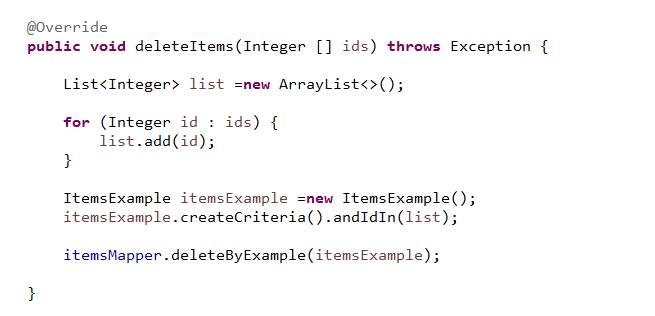
**Controller**



**Service**



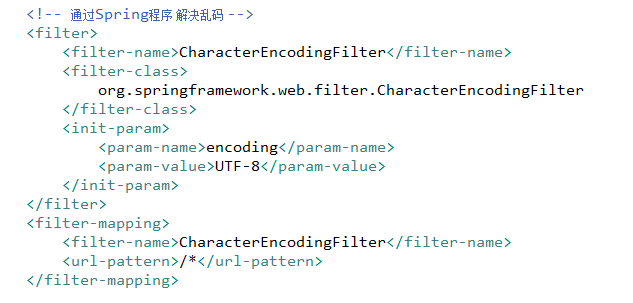
**ItemsServiceImpl**



## 常见问题

### 乱码

在web.xml中配置



# 注解开发—高级

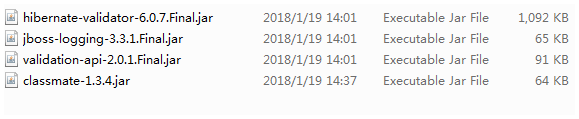
## 数据校验Validation

### 需求

在商品信息修改提交的过程中对数据进行校验，比如商品名称必须输入，价格必须是合法的数字 ，生成日期不能为空

### 加入jar包

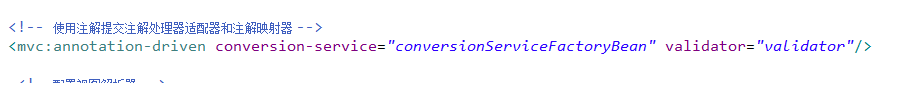
SpringMVC使用的是Hibernate校验框架



### 配置校验器

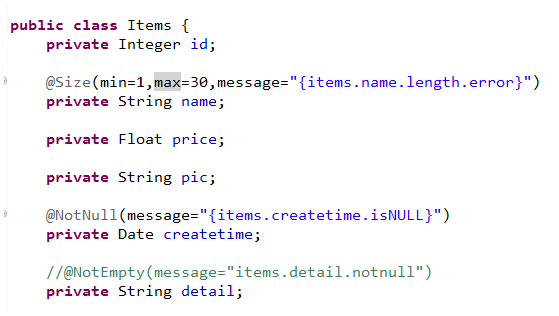


### 将校验器加到处理器适配器中

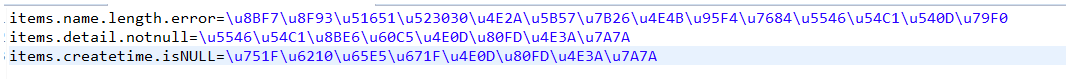


### 添加校验规则

使用Hiberante校验框架



### 错误信息的设置



### 捕获错误信息



添加@Validated 注意这是对后面的参数进行校验， 必须在这个后面有BindingResult 用于存储校验结果的

执行相关的操作

## 分组校验

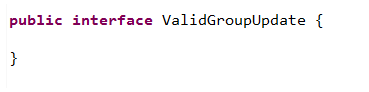
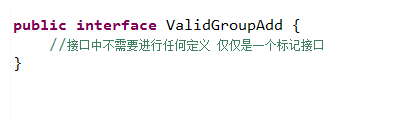
需求

对Items进行添加只对name属性和createtime进行校验

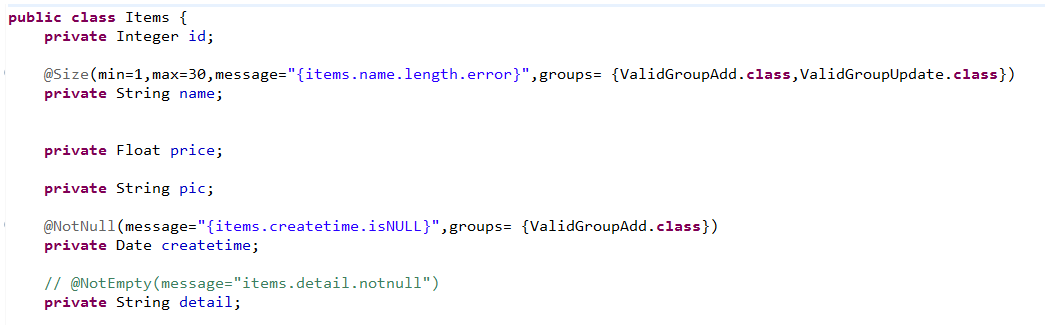
对Items进行修改仅仅对name一个属性进行校验即可

解决办法： 定义多个校验分组 其实就是一个空接口

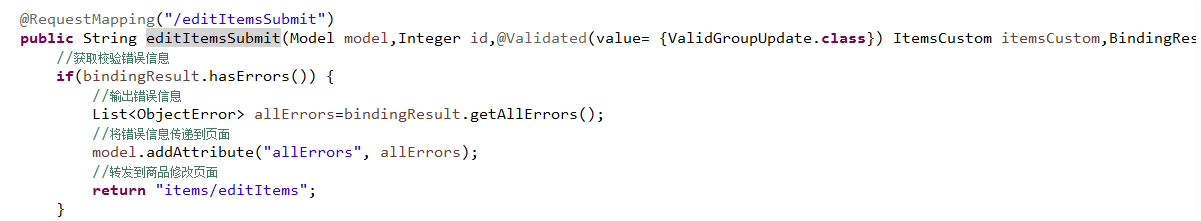
### 定义校验分组

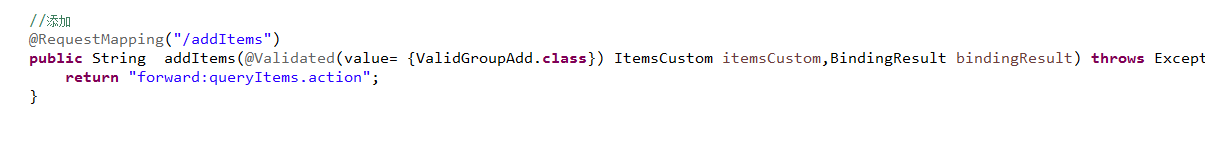


### 在校验规则上添加分组



### 在Controller方法上指定分组





### 校验注解

@Null 被注释的元素必须是空

@NotNull 注释的元素不能为空

@Min(value=10) 被注释的元素必须是一个数字 其值必须大于等于指定的最小值

@Min(value=10,message="{items.price.min.error}",groups= {ValidGroupUpdate.class})

@Max(value=) 被注释的元素必须是一个数字 其值必须小于等于指定的最大值

@Past(message="{items.createtime.past}",groups= {ValidGroupUpdate.class})

要求日期必须是一个过去的日期

@Future 要求日期必须是一个将来的日期

@Size(min=1,max=30,message="{items.name.length.error}",groups= {ValidGroupAdd.class,ValidGroupUpdate.class})

设置长度和大小

@NotEmpty 被注释的字符串不为空

@Email要求输入的邮箱必须是一个合法的邮箱

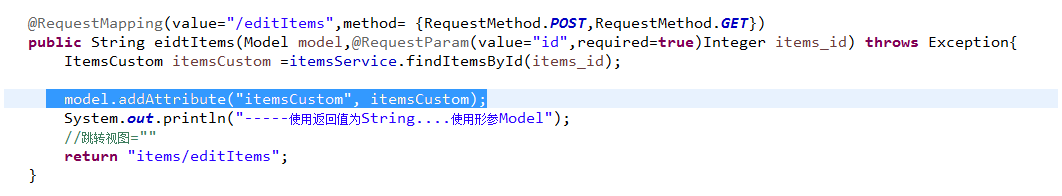
## 数据回显

### 需求

表单提交失败 需要再次回到表单重写填写数据 原来提交的数据需要在页面显示

### POJO类型

页面中的属性是itemsCustom.name 这种形式 那么将自动回显数据



SpringMVC默认支持POJO数据回显，SpringMVC自动将形参中的POJO自动重写放回到request域对象中 request中的key值就是POJO的类名(首字母小写)

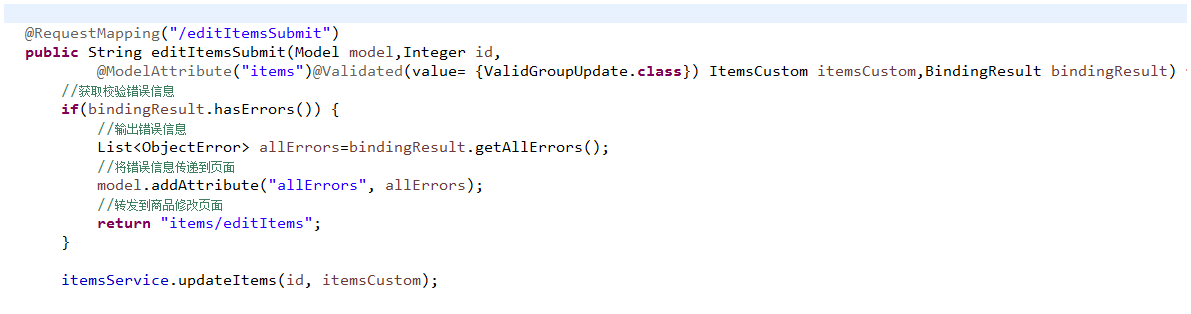


### @ModelAttribute()

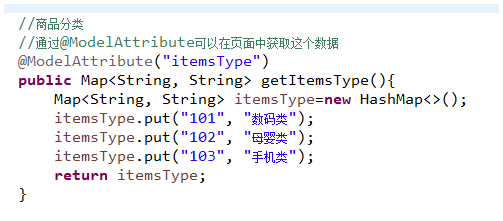
如果存储的数据key值不少POJO类名(首字母小写)，可以使用@ModelAttribute完成数据的回显 。

@ModelAttribute 作用如下:

1. 绑定请求参数到POJO并且暴露为模型数据传递到页面中 可以实现数据的回显



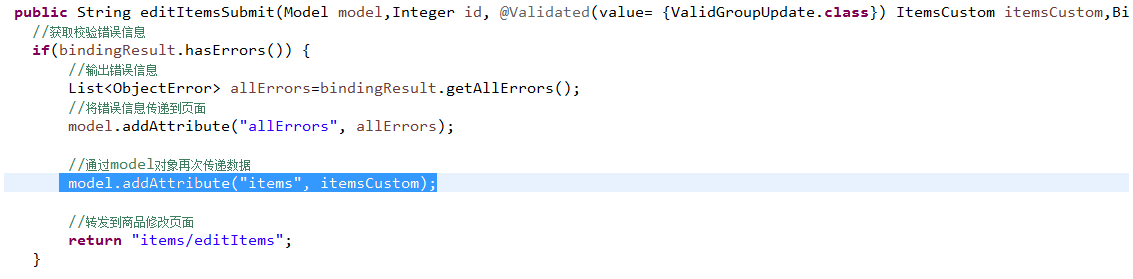
2、将方法返回值传递到模型数据 传递到页面



### 使用Model对象传递

使用Model对象直接传递 可以不使用@ModelAttribute

在校验失败的验证中 再次传递数据



Model对象可以传递数据到页面中

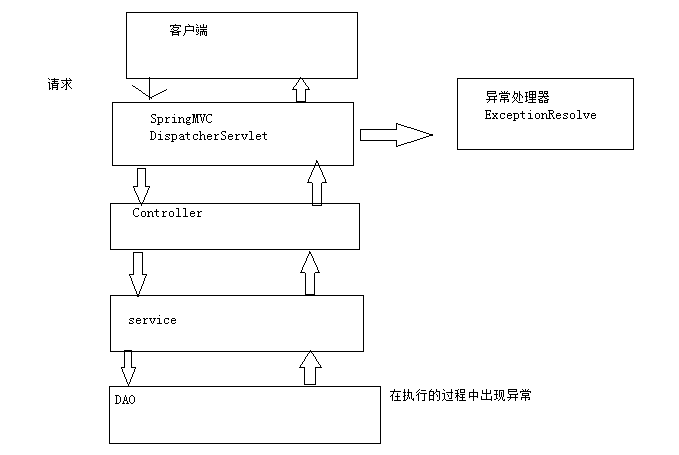
## 异常处理

SpringMVC在处理请求的过程中出现异常。对异常进行处理，自定义异常处理器。

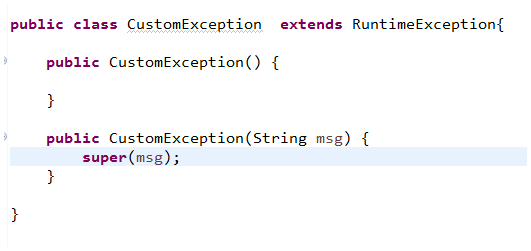
### 异常处理思路

系统异常包含两类，运行时异常和检查期异常。 而检查期异常可以通过代码进行减少出错。

系统中DAO、Service、Controller层 这些都会抛出异常。最后都由SpringMVC进行处理

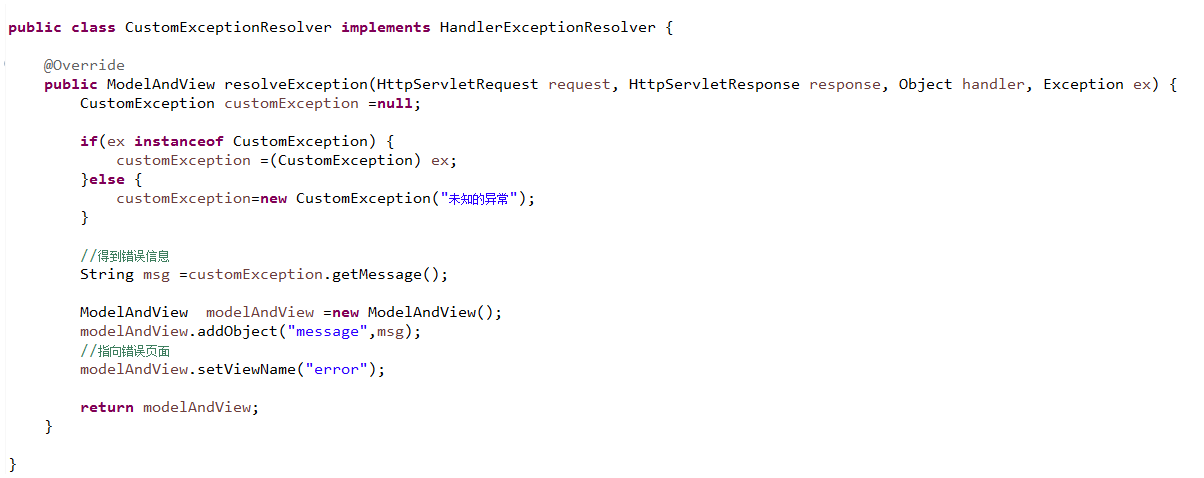


### 自定义异常

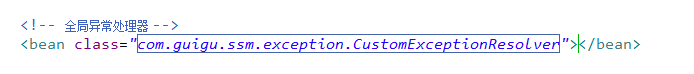


### 自定义异常处理器

SpringMVC提供了一个HandlerExceptionResolver接口



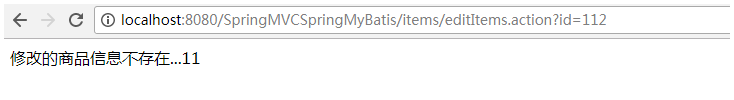
### 异常处理器的配置



### 测试

Service层抛出异常





## 上传图片

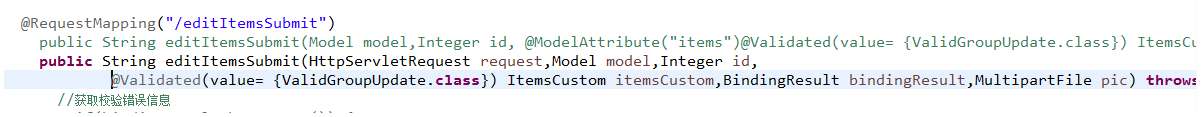
### 添加jar包



### 配置解析器



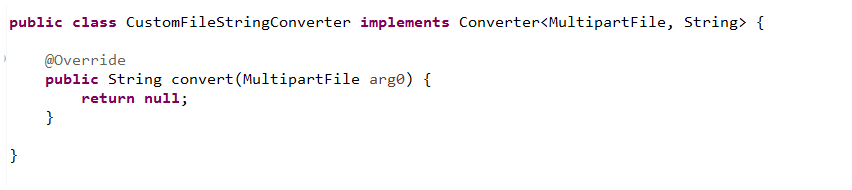
### 图片上传

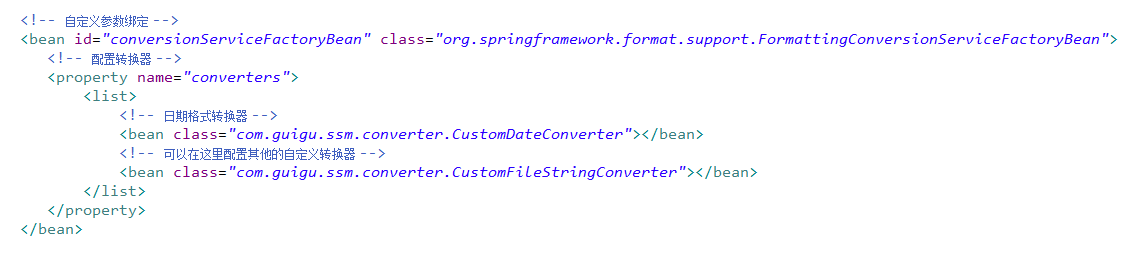




图片上传和数据校验 一起会报错  
自定义类型转换器

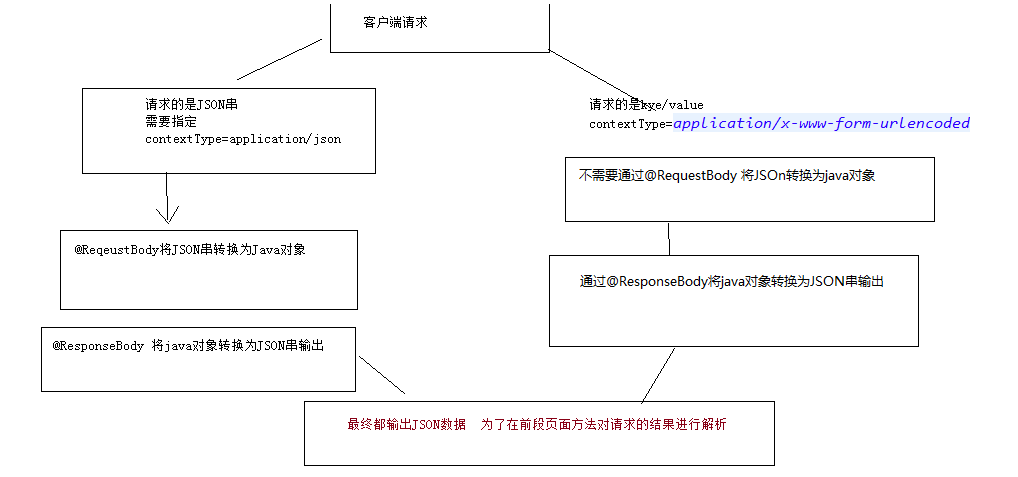
把MultipartFile类型转换为String类型





## JSON数据交互

JSON数据格式 key:value 经常在html页面中使用 JSON的格式比较简单，解析也十分方便。



1、请求JSON 输出JSON。

2、请求是key/value 输出JSON

### @RequestBody

作用： @RequestBody 注解用于读取HTTP请求的内容，通过SpringMVC的HttpMessageConverter接口将读取的到的内容转换为JSON或xml等格式数据并绑定到Controller方法的形参上、

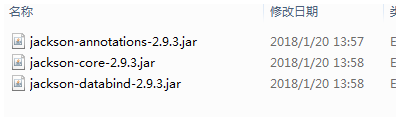
### @ResponseBody

作用：@ResponseBody 用于 将方法的返回对象 通过HttpMessageConverter转换为指定的JSON格式的数据

### 请求JSON，响应JSON实现

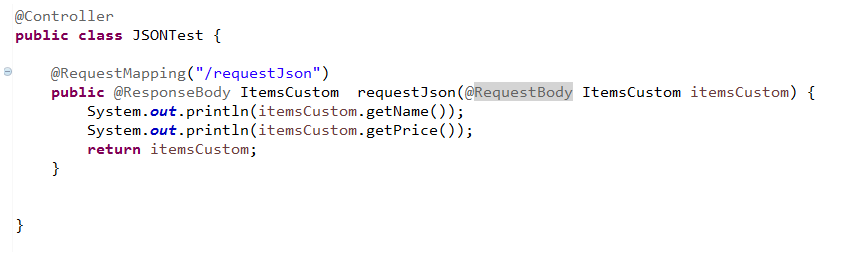
#### 环境准备

SpringMVC默认使用MappingJackSonHttpMessageConverter对JSOn数据进行准换

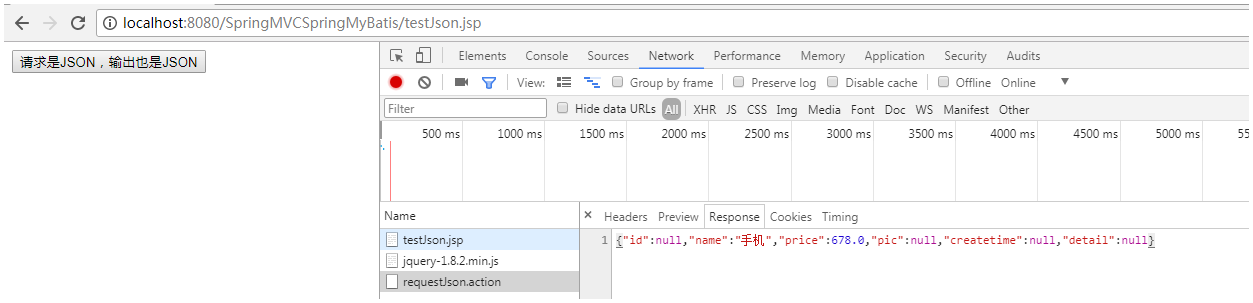


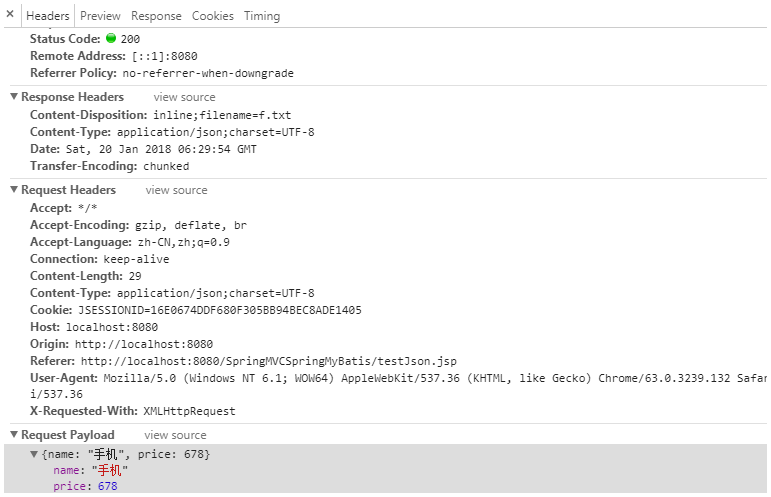
#### 配置JSON转换器

如果是使用mvc:annotation-driven 配置则不需要定义任何内容



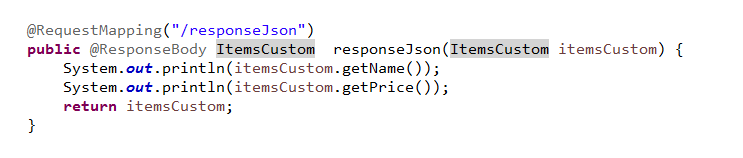
#### 测试代码





### 请求key/value 响应JSON的实现







# 拦截器

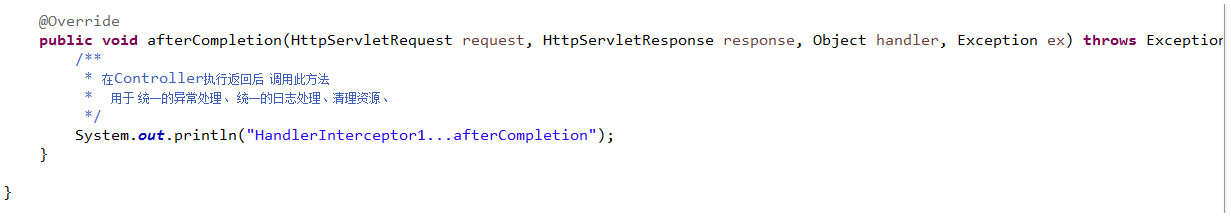
## 定义

SpringMVC的拦截器就是类似于Servlet的过滤器，用于预处理和后处理。

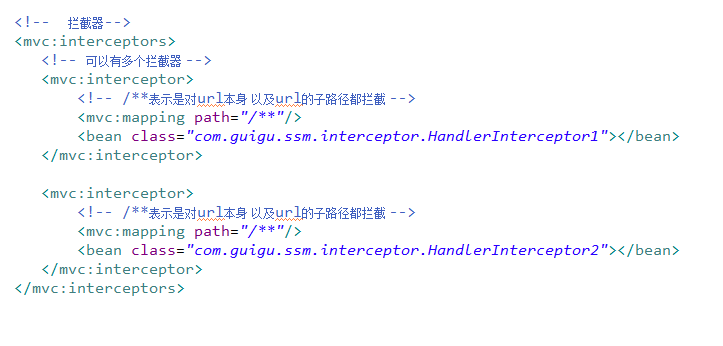
## 入门基础

### 拦截器的定义

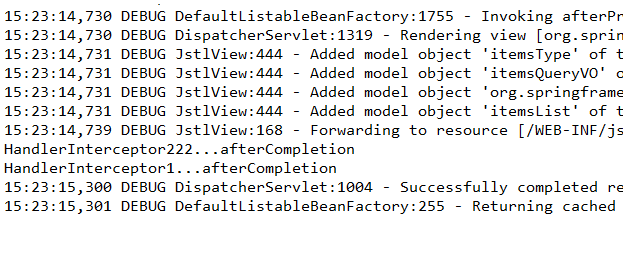
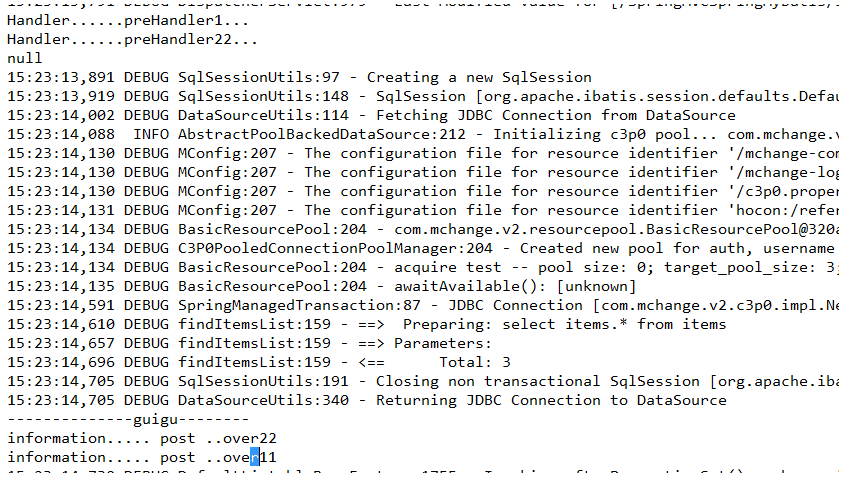




### 拦截器的配置



### 测试代码



## 拦截器的应用

### 需求

用户请求的url进行拦截

如果是公开的url 无需登录直接可以访问 直接放行

如果用户没有登录 访问数据 是不能直接访问 跳转到登录页面

如果用户已经登录 访问数据 继续执行相关的操作

### 用户登录的Controller



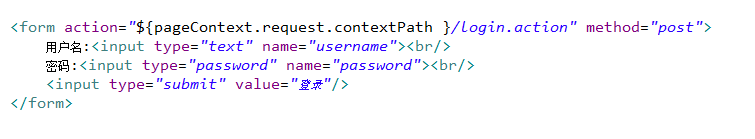
### 用户自定义拦截器



### 配置拦截器



### JSP



## 练习

商品查询 添加查询条件：价格范围查询10-20 选做日期

添加商品功能

删除商品

关于用户或者订单的增删改查。 要求按照学习的内容 每个模块都要涉及