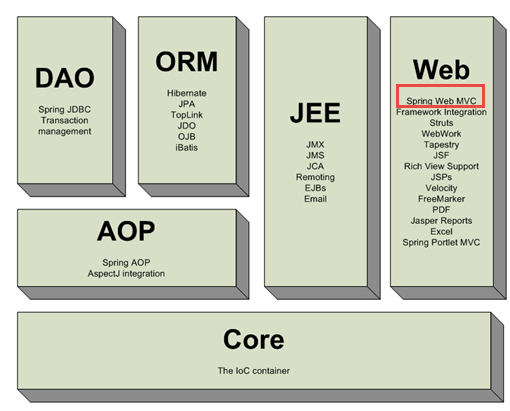
springmvc

# springmvc是什么

springmvc全称是spring web mvc，是spring框架一部分，是一个mvc的框架，和struts2一样是一个表现层框架。



# springmvc架构原理

## mvc在b/s系统中应用

mvc：是一种设计模式。设计模式理解为最佳实践。

b/s：早期大部分系统是c/s结构，是一种客户/服务器架构，和b/s（客户/服务器架构），b/s结构中客户端指的是浏览器。

web系统是一种b/s系统架构。



## springmvc框架原理



1. 用户发送请求至前端控制器DispatcherServlet
2. DispatcherServlet收到请求调用HandlerMapping处理器映射器查找Handler。
3. 处理器映射器根据请求url找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。
4. DispatcherServlet通过HandlerAdapter处理器适配器调用处理器
5. HandlerAdapter调用处理器Handler
6. Handler执行完成返回ModelAndView
7. HandlerAdapter将Handler执行结果ModelAndView返回给DispatcherServlet
8. DispatcherServlet将ModelAndView传给ViewReslover视图解析器，ViewReslover根据逻辑视图名解析View
9. ViewReslover返回View
10. DispatcherServlet对View进行渲染视图（即将模型数据填充至request域）。
11. DispatcherServlet响应用户

* DispatcherServlet前端控制器（springmvc框架提供）

作用：接收请求，响应结果

有了前端控制器减少各各组件之间的耦合性，前端控制器相关于中央调度器。

* HandlerMapping 处理器映射器（springmvc框架提供）

作用：根据url查找Handler，比如：根据xml配置、注解方式查找Handler

* HandlerAdapter处理器适配器（springmvc框架提供）

作用：执行Handler

不同类型的Handler有不同的HandlerAdapter，好处可以通过扩展HandlerAdapter支持更多类型的Handler

* Handler处理器（由程序员开发）

作用：业务处理

实现开发中又称为controller即后端控制器

Handler的开发按照HandlerAdapter的接口规则去开发。

Handler处理后的结果是ModelAndView，是springmvc的底层对象，包括 Model和view两个部分。

view中只包括一个逻辑视图名（为了方便开发起一个简单的视图名称）。

* ViewReslover视图解析（springmvc框架提供）

作用：根据逻辑视图名创建一个View对象（包括真实视图物理地址）

针对不同类型的view有不同类型的ViewReslover，常用的有jsp视图解析器即jstlView

* View视图（由程序员开发jsp页面）

作用：将模型数据填充进来(将model数据填充到request域)显示给用户

view是一个接口，实现类包括：jstlView、freemarkerView，pdfView...

# springmvc入门程序

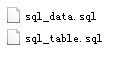
## 需求

查询商品信息列表

## 准备环境

mysql：mysql5.1

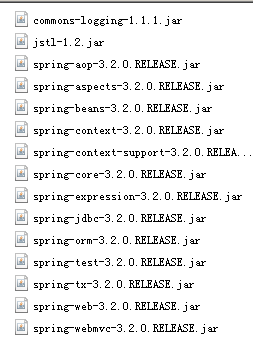
创建订单商品模型的数据库



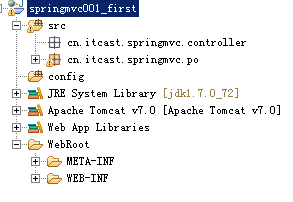


jar包：

spring3.2.0（包括springmvc的包spring-webmvc-3.2.0.RELEASE.jar）



创建web工程：



## 前端控制器配置

在web.xml配置springmvc的前端控制器



## 配置处理器适配器

开发Handler必须要按照处理器适配器接口规则去开发，先要确定使用哪个处理器适配器。

这里使用*SimpleControllerHandlerAdapter*

在springmvc.xml中配置适配器：

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

通过源码分析得出：

**public** **boolean** supports(Object handler) {

**return** (handler **instanceof** Controller);

}

只要实现了Controller接口的bean，*SimpleControllerHandlerAdapter都可以去执行。*

## 开发Handler

**public** **class** ItemsController1 **implements** Controller {

@Override

**public** ModelAndView handleRequest(HttpServletRequest request,

HttpServletResponse response) **throws** Exception {

// 调用service查询商品信息

// 这里不调用service使用静态数据模拟

// 商品列表

List<Items> itemsList = **new** ArrayList<Items>();

Items items\_1 = **new** Items();

items\_1.setName("联想笔记本");

items\_1.setPrice(6000f);

items\_1.setDetail("ThinkPad T430 联想笔记本电脑！");

Items items\_2 = **new** Items();

items\_2.setName("苹果手机");

items\_2.setPrice(5000f);

items\_2.setDetail("iphone6苹果手机！");

itemsList.add(items\_1);

itemsList.add(items\_2);

//构造 一个modelAndView

ModelAndView modelAndView = **new** ModelAndView();

//将itemsList商品信息设置到ModelAndView中，最终将itemsList填充到request域中

//此方法相当于request.setAttribute(arg0, arg1)

modelAndView.addObject("itemsList", itemsList);

//指定视图地址

modelAndView.setViewName("/WEB-INF/jsp/itemsList.jsp");

**return** modelAndView;

}

}

## 将Handler在spring容器中配置

在springmvc.xml中配置Handler



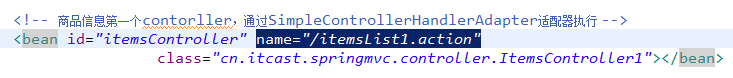
## 配置处理器映射器

在springmvc.xml中配置BeanNameUrlHandlerMapping：

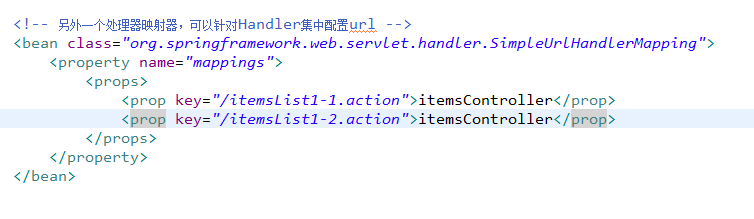


使用BeanNameUrlHandlerMapping，表示将定义的Bean（Handler）名字作为请求的url。

指定Handler在spring容器中配置时，指定name为Handler对应的url，如下：



simpleUrlHandlerMapping是BeanNameUrlHandlerMapping的增强版本，它可以将url和处理器bean的id进行统一映射配置。



## 视图解析器

使用jsp的视图解析器：

在springmvc.xml文件配置如下：

<!-- ViewResolver -->

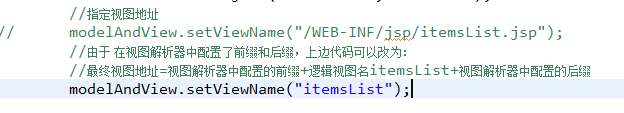
<bean class=*"**org.springframework.web.servlet.view.**InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/jsp/"*/>

<property name=*"suffix"* value=*".jsp"*/>

</bean>

修改controller的代码，改为使用逻辑视图名：



## jsp视图



## 调试

根据url找不到Handler，报以下错：



根据url找到了Handler，但是没有找到jsp视图报下边错误 ：



## 其它非注解的处理器映射器和适配器

目的，通过了解多个不同的处理器映射器及适配器工作过程。

### 非注解的处理器映射器

BeanNameUrlHandlerMapping：使用BeanNameUrlHandlerMapping，表示将定义的Bean（Handler）名字作为请求的url

simpleUrlHandlerMapping：是BeanNameUrlHandlerMapping的增强版本，它可以将url和处理器bean的id进行统一映射配置。

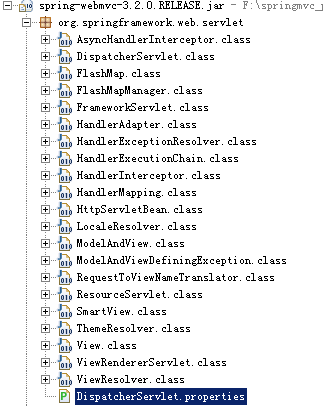
### 非注解的处理器适配器

SimpleControllerHandlerAdapter：只要实现了Controller接口的bean，SimpleControllerHandlerAdapter都可以去执行

HttpRequestHandlerAdapter：只要实现了HttpRequestHandler接口的bean，HttpRequestHandlerAdapter都可以去执行。

## DispatcherServlet.properties

每个系统启动后， springmvc的前端控制器会加载下边的配置文件。



## 注解开发

### 注解处理器映射器和适配器

注解的处理器映射器：

spring3.1版本之前使用

org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping，3.1版本之后使用

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

注解的处理器适配器：

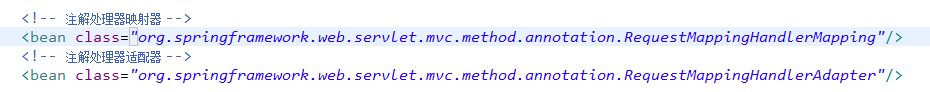
spring3.1之前使用

org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter

3.1之后使用

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

在springmvc.xml中配置注解处理器映射器和适配器：





### 开发Handler

注解处理器映射器和注解处理器适配器要配对使用，对标记@controller注解类中的标记有@RequestMapping方法进行映射和执行。

@controller标记此类是一个控制器

@RequestMapping：对url和Handler中的方法进行映射。

@Controller

**public** **class** ItemsController3 {

//查询商品的方法

@RequestMapping("/itemsList3")

**public** ModelAndView itemsList() **throws** Exception {

// 调用service查询商品信息

// 这里不调用service使用静态数据模拟

// 商品列表

List<Items> itemsList = **new** ArrayList<Items>();

Items items\_1 = **new** Items();

items\_1.setName("联想笔记本");

items\_1.setPrice(6000f);

items\_1.setCreatetime(**new** Date());

items\_1.setDetail("ThinkPad T430 联想笔记本电脑！");

Items items\_2 = **new** Items();

items\_2.setName("苹果手机");

items\_2.setPrice(5000f);

items\_2.setDetail("iphone6苹果手机！");

itemsList.add(items\_1);

itemsList.add(items\_2);

//构造 一个modelAndView

ModelAndView modelAndView = **new** ModelAndView();

//将itemsList商品信息设置到ModelAndView中，最终将itemsList填充到request域中

//此方法相当于request.setAttribute(arg0, arg1)

modelAndView.addObject("itemsList", itemsList);

//指定视图地址

// modelAndView.setViewName("/WEB-INF/jsp/itemsList.jsp");

//由于 在视图解析器中配置了前缀和后缀，上边代码可以改为：

//最终视图地址=视图解析器中配置的前缀+逻辑视图名itemsList+视图解析器中配置的后缀

modelAndView.setViewName("itemsList");

**return** modelAndView;

}

}

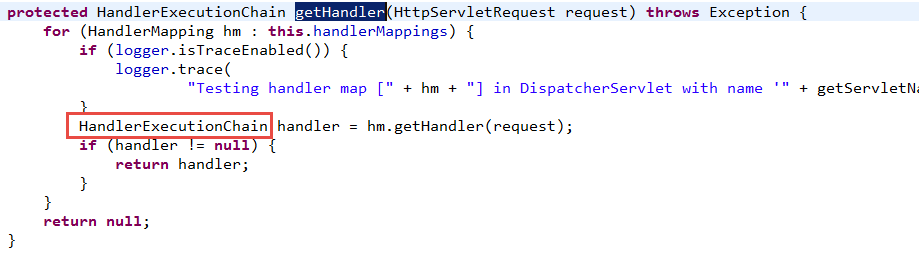
### 在spring容器中配置Handler

为了简单方便对Handler在spring容器中进行配置，采用组件扫描：



## 通过源码跟踪理解执行过程

1. 用户发送请求至前端控制器DispatcherServlet
2. DispatcherServlet收到请求调用HandlerMapping处理器映射器查找Handler。



1. 处理器映射器根据请求url找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。
2. DispatcherServlet通过HandlerAdapter处理器适配器执行处理器

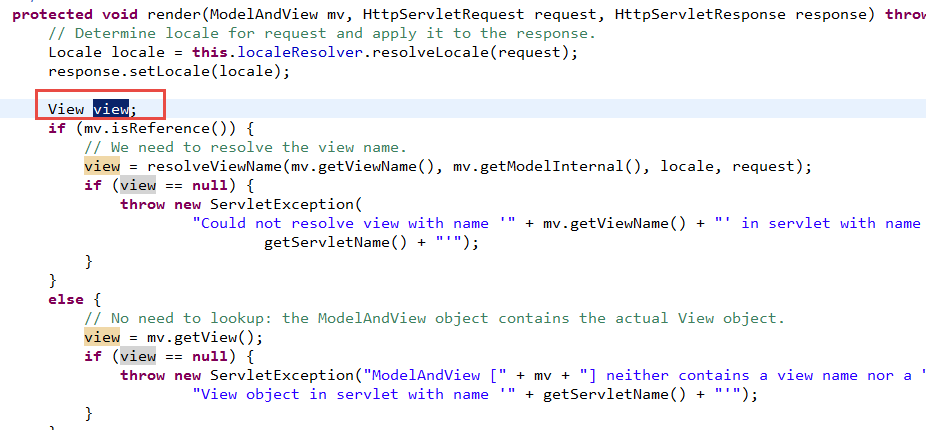
前端控制器查找能够去执行Handler的适配器：



1. HandlerAdapter调用处理器Handler
2. Handler执行完成返回ModelAndView



1. HandlerAdapter将Handler执行结果ModelAndView返回给DispatcherServlet
2. DispatcherServlet将ModelAndView传给ViewReslover视图解析器，ViewReslover根据逻辑视图名解析View
3. ViewReslover返回View

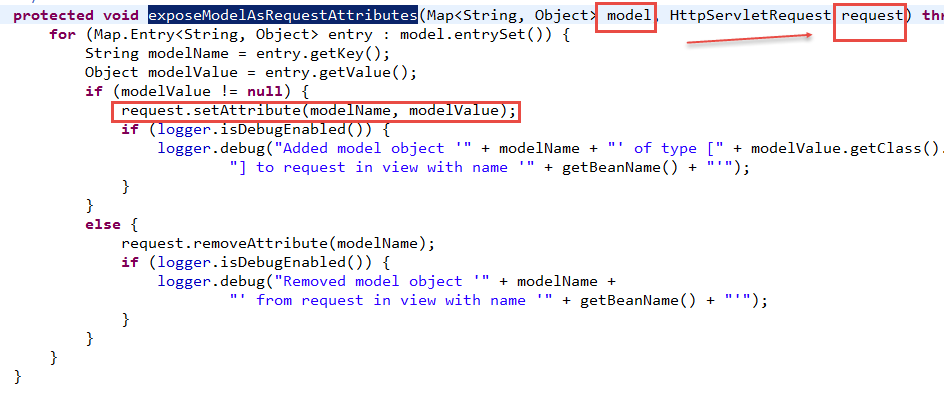


1. DispatcherServlet对View进行渲染视图（即将模型数据填充至request域）。

调用view对象的渲染方法：



将模型数据填充到request域：



1. DispatcherServlet响应用户

# springmvc和mybatis整合

## 需求

实现商品查询列表，从mysql数据库查询商品信息。

## 整合思路

spring容器

表现层springmvc（mvc框架）

业务层spring

持久层 mybatis（对jdbc封装）

mysql

spring的作用：

表现层：spring要将springmvc编写的controller控制器加入ioc容器中进行管理

业务层：spring要将service类加入ioc容器中进行管理

spring要对service方法执行进行事务控制

持久层：spring要将mapper接口代理对象在容器中进行管理

整合思路如下：

dao层：

mybatis+spring

目标：

使用spring对sqlSessionFactory进行管理

使用spring和mybatis整合包中的mapper扫描器对mapper接口进行扫描生成代理对象。

service层：

spring

目标：

让spring管理service类，将mapper代理对象注入到service对象中。

spring要对service方法执行进行事务控制

controller层：

springmvc+spring

目标：

使用注解处理器映射器和适配器进行开发Handler

Handler使用组件扫描方式在spring容器中进行注册

## 整合工程搭建

创建web工程。

加入jar包：

spring3.2.0所有包(包括springmvc的包)

mybatis及依赖包

mybatis和spring整合包

junit包

log4j....

## 整合dao

实现mybatis+spring整合。

### 配置文件

mybatis的全局配置文件：SqlMapConfig.xml

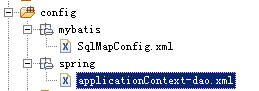


配置applicationContext-dao.xml

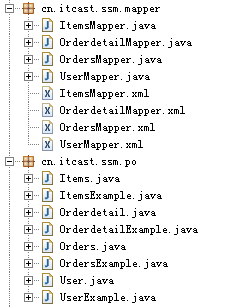
配置数据库连接池（dbcp）

配置SqlSessionFactory

配置mapper扫描器



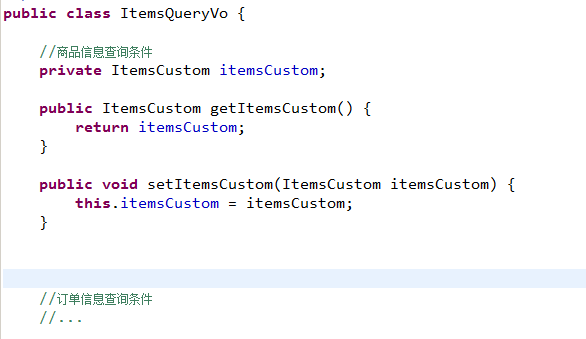
### 使用逆向工程生成po类



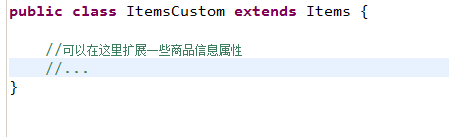
### mapper.xml

针对综合条件查询可能是多表的关系查询，建议自定义mapper。

查询条件要通过包装类型对象传入：



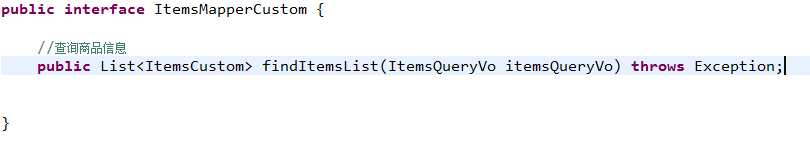
查询结果使用items的扩展类型映射输出：



mapper映射文件中statement的定义：

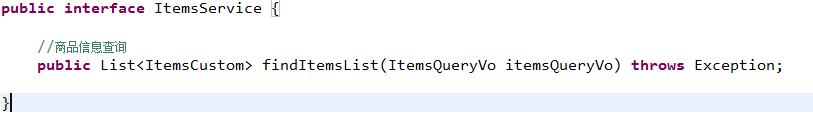


### mapper.java



## 整合service

### service接口





### 配置文件

创建applicationContext-service.xml：

配置service类



创建applicationContext-transacation.xml

配置声明式事务

<!-- 事务管理器 -->

<bean id=*"transactionManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

<property name=*"dataSource"* ref=*"dataSource"*/>

</bean>

<!-- 通知 -->

<tx:advice id=*"txAdivce"* transaction-manager=*"transactionManager"*>

<tx:attributes>

<tx:method name=*"save\*"* propagation=*"REQUIRED"*/>

<tx:method name=*"insert\*"* propagation=*"REQUIRED"*/>

<tx:method name=*"update\*"* propagation=*"REQUIRED"*/>

<tx:method name=*"delete\*"* propagation=*"REQUIRED"*/>

<tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"*/>

<tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"*/>

</tx:attributes>

</tx:advice>

<!-- aop -->

<aop:config>

<aop:advisor advice-ref=*"txAdivce"* pointcut=*"execution(\* cn.itcast.ssm.service.\*.\*(..))"*/>

</aop:config>

## 整合controller

### 配置文件

创建springmvc.xml：

配置组件扫描器，扫描@controller标记的类

配置注解方式的处理器映射器和适配器

配置视图解析

<!-- 使用组件扫描，将@controller注解的类扫描出来在容器中进行注册

组件扫描可以扫描标记有@controller、@service、@repository、@component的bean进行扫描

-->

<context:component-scan base-package=*"cn.itcast.ssm.controller"*/>

<!-- 注解处理器映射器 -->

<!-- <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/> -->

<!-- 注解处理器适配器 -->

<!-- <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/> -->

<!-- 上边的注解处理器映射器和适配器配置可以使用下边mvc:annotation-driven代替 -->

<mvc:annotation-driven></mvc:annotation-driven>

<!-- ViewResolver视图解析器，解析jsp视图 -->

<bean

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<!-- 指定视图路径前缀 -->

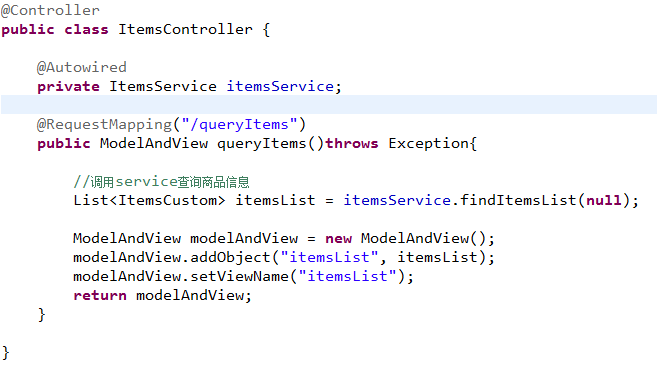
<property name=*"prefix"* value=*"/WEB-INF/jsp/"* />

<!-- 指定视图路径后缀 -->

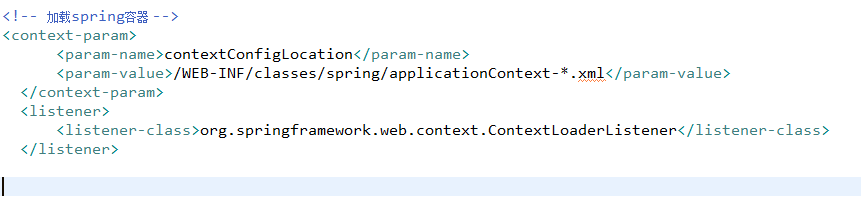
<property name=*"suffix"* value=*".jsp"* />

</bean>

### 开发controller



### 在web.xml配置加载spring容器



### 在web.xml配置前端控制器



# 注解开发-基础

## 需求

实现商品信息修改功能。

操作流程：

1、用户进入商品查询列表页面

2、用户点击“修改”进入商品修改页面

3、用户输入商品信息，提交修改信息

## 功能开发

### mapper

1、根据商品id查询商品信息

2、更新商品信息

以上两个接口使用逆向工程生成。

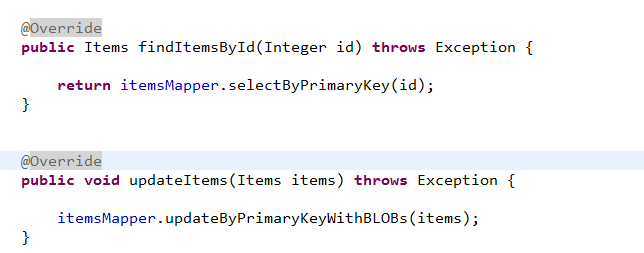
### service

接口1、根据商品id查询商品信息



接口2、更新商品信息





## 参数绑定

### 什么是参数绑定

springmvc接收请求的key/value串（比如：id=2&type=101），经过类型转换，将转换后的值赋值给controller方法的形参，这个过程就叫参数绑定。

### 默认支持的类型

在controller方法形参中如下类型是可以直接绑定成功，springmvc框架给以下类型的参数直接赋值：

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/ModelMap

ModelMap是Model接口的实现类，通过Model或ModelMap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。

使用Model和ModelMap的效果一样，如果直接使用Model，springmvc会实例化ModelMap。

### 基本类型参数绑定

基本类型类型：Integer、string、boolean、float。。。

绑定规则：

对于基本类型参数绑定，当请求的参数的名称和controller方法的形参名一致时可以绑定成功。

如果请求的参数的名称和controller方法的形参名不一致时，如何绑定？

就要使用@RequestParam进行绑定：

使用方法如下：

@RequestParam(value="ids") Integer id 将请求的参数名为ids的值绑定方法形参的id上。

required：是否必须，默认是true，表示请求中一定要有相应的参数，否则将报；

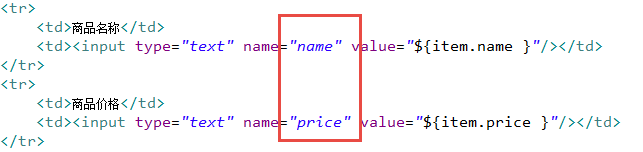
TTP Status 400 - Required Integer parameter 'XXXX' is not present

defaultValue：默认值，表示如果请求中没有同名参数时的默认值

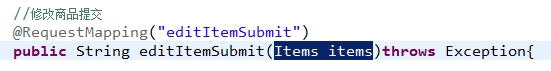
### 简单pojo参数绑定

简单pojo：pojo中都基本类型，没有包括自定义pojo属性。

页面参数名称的定义：

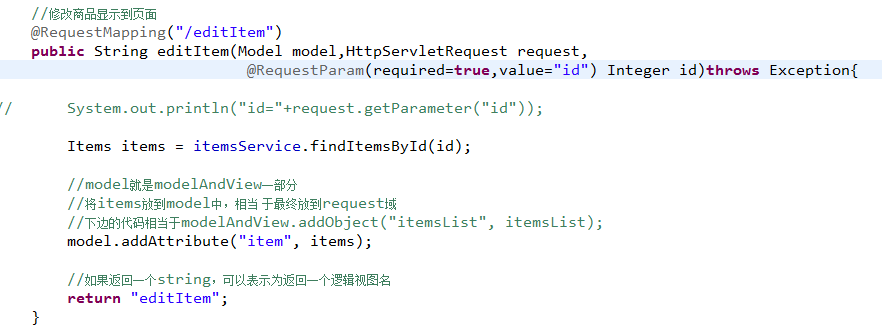


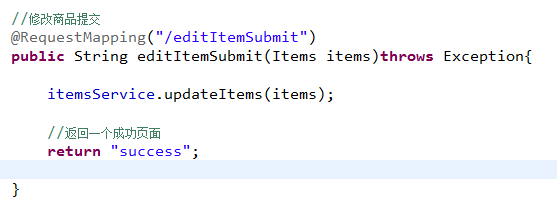
controller方法的参数定义：



当页面参数名称和controller方法的pojo的参数的属性名一致，可以将请求的参数绑定到pojo形参的属性中。

### controller开发





### 自定义参数绑定

问题：

页面输入 2020-12-12 12:12:12(年月日 时分秒)，springmvc默认提供年月日的绑定，没有对时分秒进行绑定。

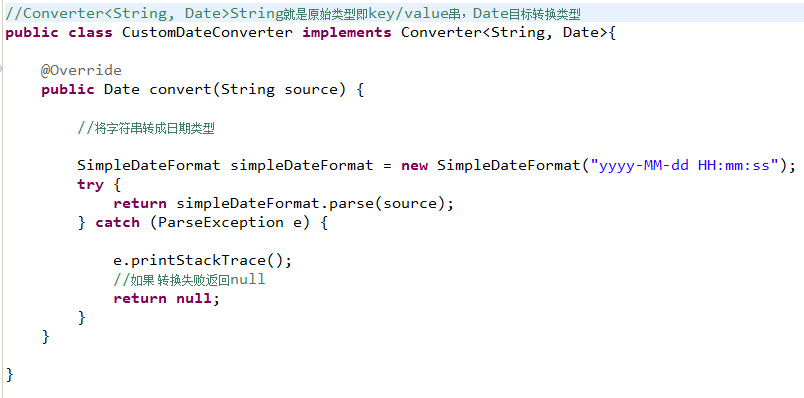
解决问题：

需要自定义参数绑定器，实现将页面输入年月日 时分秒全部转成日期类型数据。

自定义参数绑定器方法：

自定义参数绑定器实现Conver<S,T>，S表示源类型，T表示目标类型，S是页面请求的参数类型，即string，目标类型是要绑定到pojo的属性的类型，上边的例子T目标类型就是：Items中createtime属性类型即java.util.Date

开发自定义参数绑定器：



在springmvc.xml中在处理器适配器上配置参数绑定器：

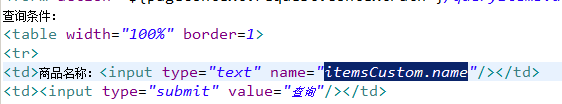


思考：如果自定义参数绑定器将提交的空字符串""转成null。

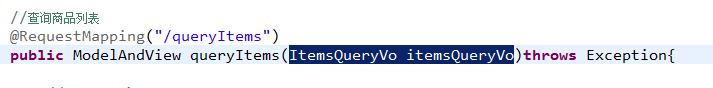
### 包装类型pojo绑定

包装类型pojo：pojo中属性也是一个pojo。

页面请求参数名称定义：



controller方法形参的定义：



包装类型的pojo绑定规则：

itemsCustom是包装pojo的属性名。

itemsCustom.name中name就是包装pojo中属性itemsCustom的一个属性。

### 集合类型参数绑定-数组

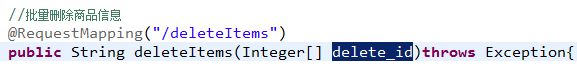
需求： 在商品查询列表页面批量选择商品进行删除。

页面要提交多个商品id，controller方法使用数组接收页面提交的参数。

页面参数名称定义：



controller方法形参数定义：



绑定规则：

如果请求多个参数且名称一致，可以绑定到和请求参数名一致的数组中。

### 集合类型参数绑定-List<pojo>

需求：

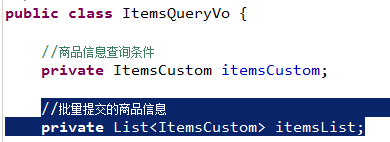
实现商品信息批量修改功能，类似功能有成绩录入，一次录入好多门成绩，执行批量提交。

页面批量修改效果：



controller方法形参定义：

使用包装类型接收页面提交的批量商品信息：



在包装类型中创建List<ItemsCustom> itemsList属性。

页面参名称定义：

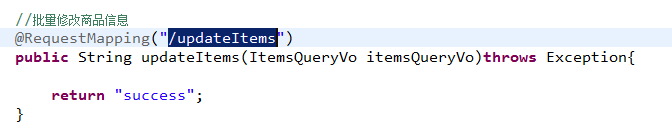


## requestMapping

requestMapping注解的作用：对controller方法进行映射。

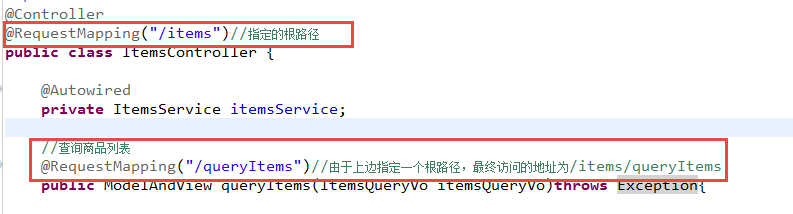
* URL路径映射

requestMapping指定url，对controller方法进行映射。



* 窄化请求映射

为了更好的管理url，为了避免url冲突，可以在class上使用requestMapping指定根url。



**在开发时候，需要提前进行url规划，以避免后期修改url后，需要大量修改页面上的url地址。**

* 请求方法限定

通过requestMapping限制http的请求方法，可以提高系统安全性。



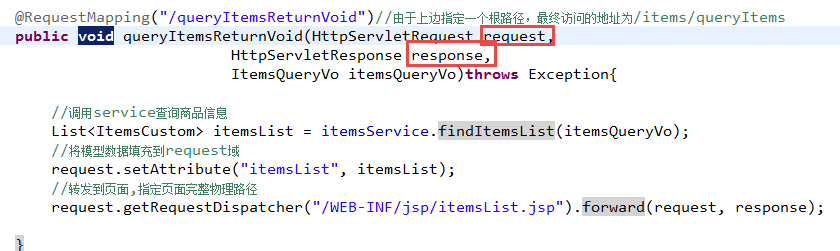
## controller方法的返回值

* 返回ModelAndView

controller方法中定义ModelAndView对象并返回，对象中可添加model数据、指定逻辑视图名。

* 返回Void

类似原始serlvet 的开发。



响应结果的三种方法：

1、使用request转发页面，如下：

request.getRequestDispatcher("页面路径").forward(request, response);

2、也可以通过response页面重定向：

response.sendRedirect("url")

3、也可以通过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");

* 返回String(推荐使用)

1. 页面转发方式

格式是：forward:转发地址（不能写http://，只能写action的地址）

特点：转发的上一个请求request和要转发的地址共用request，转发后浏览器的地址是不变化。

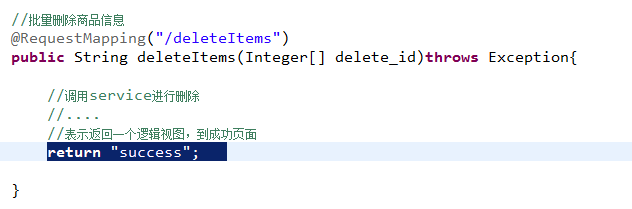
1. 页面重定向方式

格式是：redirect:重定向地址（比如：http://.....）

特点：重定的上一个请求request和要重定的地址不公用request，重定后浏览器的地址是变化的。

1. 表示逻辑视图名

返回一个string如果即不是转发格式，也不是重定向的格式，就表示一个逻辑视图名。



## springmvc与struts2本质区别

1、springmvc入口是一个servlet前端控制器，struts2入口是一个filter过虑器。

2、struts2通过在action类中定义成员变量接收请求参数，struts2只能使用多例模式管理action。

springmvc通过在controller方法中定义形参接收请求参数，springmvc可以使用单例模式管理controller。

3、springmvc是基于方法开发的，注解开发中使用requestMapping将url和方法进行映射，如果根据url找到controller类的方法生成一个Handler处理器对象（只包括一个method）。

struts2是基于类开发，每个请求过来创建一个action实例，实例对象中有若干的方法。

开发中建议使用springmvc，springmvc方法更类似service业务方法。

4、Struts采用值栈存储请求和响应的数据，通过OGNL存取数据， springmvc通过参数绑定器是将request请求内容解析，并给方法形参赋值。

5、struts2和springmvc的速度是相当的，由于struts2的漏洞较多，很多企业转向使用springmvc了。

## springmvc实现校验（了解）

### springmvc的校验方法

在页面中通过js进行校验属于前端校验，从用户体验性出发要前端页面添加一些校验，不用等到用户提交到后台系统在前端可以校验，给出校验结果，体验性上比较好。

出于系统安全的考虑，就需要在服务端也添加校验。

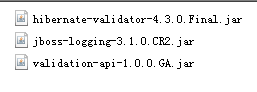
使用springmvc提供Validation校验框架完成在controller中校验。

Spring3支持JSR-303验证框架，JSR-303 是JAVA EE 6 中的一项子规范，叫做Bean Validation，官方参考实现是Hibernate Validator（与Hibernate ORM 没有关系），JSR 303 用于对Java Bean 中的字段的值进行验证。

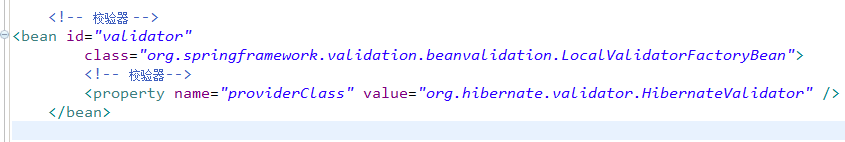
### 需求

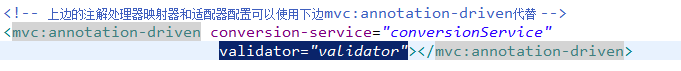
商品信息在修改时进行校验，商品名称长度不能超过30字符长度，商品生产日期为必须填项。

### 加入validator的jar包



### 在适配器配置校验器

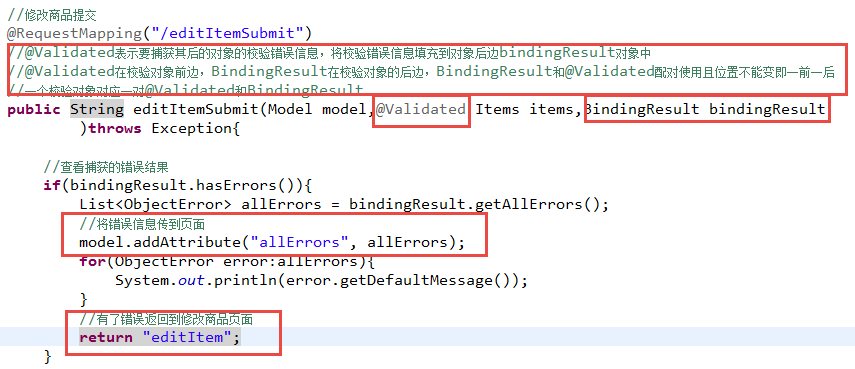




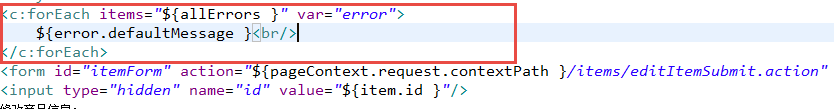
### 在pojo中配置校验规则



### 在controller方法捕获校验结果



在页面显示错误信息：



### 分组校验

问题：

当两个controller方法公用一个pojo进行校验，此时如果不同的方法想只校验一部分pojo的属性，如何实现？

比如：商品修改时只校验商品名称的长度而不校验生产日期是否为空。

实现：

可以将不同的校验需求以分组的方式进行区分。

分组就是一个标识，标识哪些校验规则属性哪一个分组。

#### 定义分组

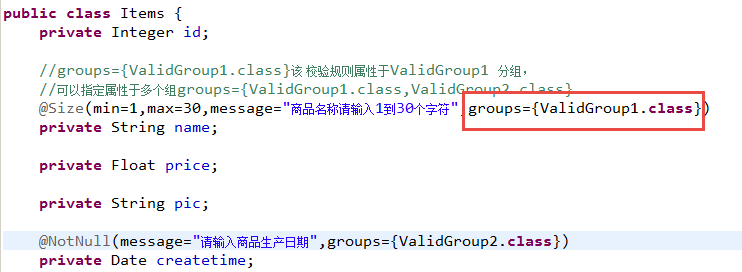
分组定义在实现时就是定义一个java接口。

如下定义了两个分组：

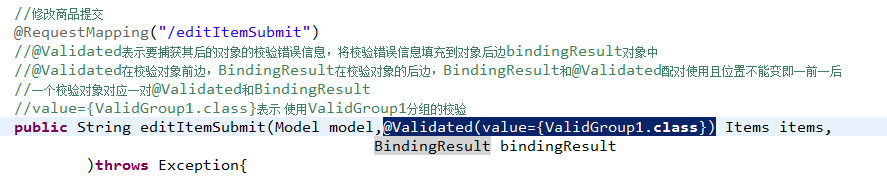


#### 对校验规则进行分组

在pojo类中指定每个校验规则属性于哪个分组。



#### 在controller方法指定使用哪个分组



### 校验错误信息配置文件

将校验错误信息不能直接硬编码到pojo类中，不利于系统维护。

#### 将错误信息配置在一个配置文件中

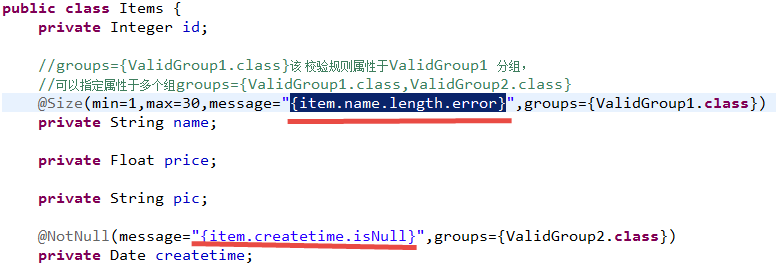
在classpath 定义文件：



文件内容是错误信息：



#### 在pojo中引用错误信息配置文件中的key



#### 在错误信息配置文件注入到校验器



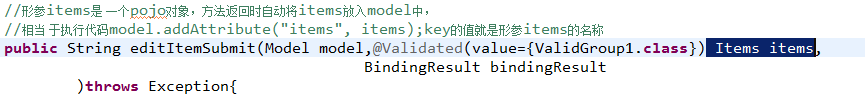
## springmvc实现数据回显

### 什么是数据回显

当提交请求到controller，如果controller进行校验错误重新返回到页面，页面中应该显示刚刚提交的信息，这叫数据回显。

### pojo的数据回显

针对controller形参是pojo对象，默认支持数据回显。



1、ModelAttribute可以绑定请求参数到pojo并且暴露为模型数据传到视图页面

如果不满足上边的描述，形参名称和页面上request的key值不一致，使用@ModelAttribute指定回显的数据设置到request域的key值。

@ModelAttribute("item")，相当 于执行model.addAttribute("item", items);

2、将方法返回值暴露为模型数据传到视图页面

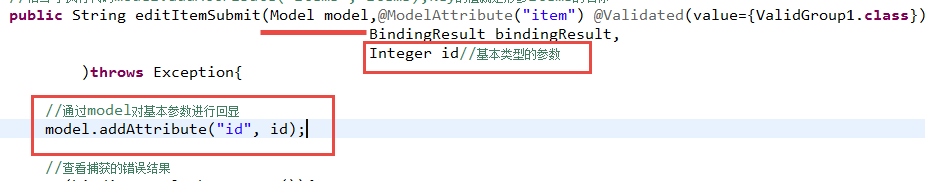


### 基本类型的数据回显

如果形参是java基本类型的，默认不支持数据回显。

数据回显方法：

使用model对象进行回显：



## springmvc异常处理

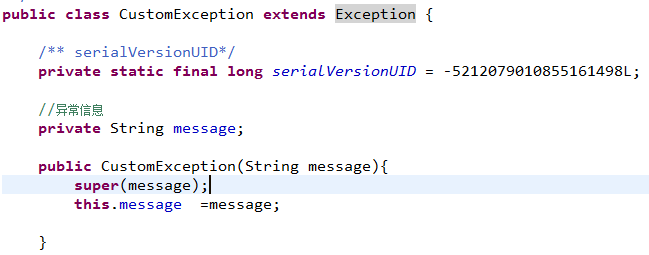
### 业务系统中异常处理方法

异常类型包括可预期的异常、运行时异常，可预期的异常可以捕获的异常，程序员在编码阶段对预期的异常进行捕获处理，运行时异常不能预期知道的异常。

程序员应该把重点放在可预期的异常处理上，对预期异常分类进行管理，定义不同的异常类型，根据类型进行有针对性的处理。

对于运行时异常，应该在系统测试阶段加强测试，将运行时异常测试出来修改bug，从而降低运行时异常的出现概率。如果系统运行时出现运行时异常，一般提示“系统繁忙，，系统出错请与管理员联系。。。”

### 自定义异常类型



### springmvc进行异常处理

springmvc提供异常处理器接口HandlerExceptionResolver，自定义异常处理器对系统中异常进行处理。

异常处理流程：

1、异常发生（在controller、service、dao）

2、在异常处理器中捕获异常

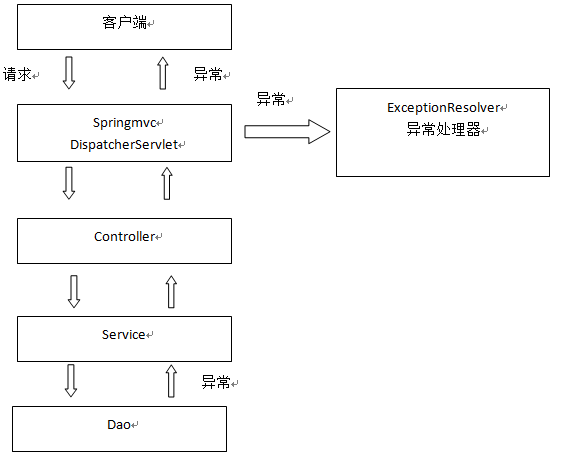
3、判断异常类型，根据异常类型有针对性去处理

如果异常类型是系统自定义的异常说明程序员在编码中有目的抛出预期的异常

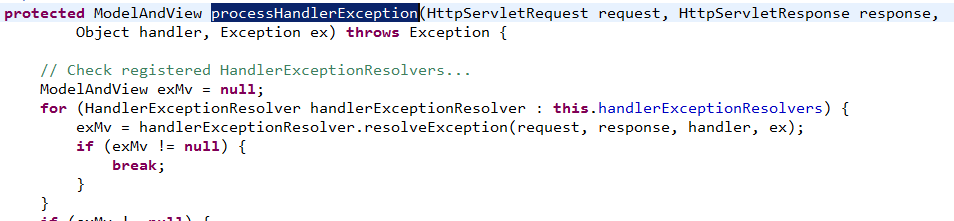
调用异常类型的getMessage方法获取异常信息。

如果不是系统自定义的异常说明是运行时异常，提示用户“系统执行中出现错误请与管理员联系”

springmvc异常处理流程：



在dao中预期异常抛给service，service又抛给controller，controller将异常抛给前端控制器，由前端控制器去调用异常处理器执行异常，参考下边的源代码：



### 异常处理器开发

**package** cn.itcast.ssm.controller.exceptionResolver;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.springframework.web.method.HandlerMethod;

**import** org.springframework.web.servlet.HandlerExceptionResolver;

**import** org.springframework.web.servlet.ModelAndView;

/\*\*

\*

\* <p>Title: CustomExceptionResolver</p>

\* <p>Description: 自定义异常处理器</p>

\* <p>Company: www.itcast.com</p>

\* **@author** 传智播客

\* **@date** 2020-12-12

\* **@version** 1.0

\*/

**public** **class** CustomExceptionResolver **implements** HandlerExceptionResolver {

/\*\*

\* （非 Javadoc）

\* <p>Title: resolveException</p>

\* <p>Description: </p>

\* **@param** request

\* **@param** response

\* **@param** handler handler就是HandlerMethod，根据url映射的Handler实例 对象

\* **@param** ex 抛出的异常

\* **@return**

\* **@see** org.springframework.web.servlet.HandlerExceptionResolver#resolveException(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse, java.lang.Object, java.lang.Exception)

\*/

@Override

**public** ModelAndView resolveException(HttpServletRequest request,

HttpServletResponse response, Object handler, Exception ex) {

//执行到此方法说明将异常信息已经捕获，ex就是异常信息

CustomException customException = **null**;

//如果是系统自定义的异常有针对性的进行业务处理

**if**(ex **instanceof** CustomException){

//获取异常信息

//比如还可以记录异常日志...

//..

customException = (CustomException)ex;

}**else**{

//如果抛出的异常不是系统自定义的异常

//提示信息为“系统执行出错，请与管理员联系。。”

//重新构造 一个自定义的异常对象

customException = **new** CustomException("系统执行出错，请与管理员联系");

//比如还可以记录异常日志...

//..

}

//获取异常信息

String message =customException.getMessage();

//将请求转发错误信息页面，在错误页面上显示错误信息

ModelAndView modelAndView = **new** ModelAndView();

modelAndView.setViewName("error");

modelAndView.addObject("message", message);

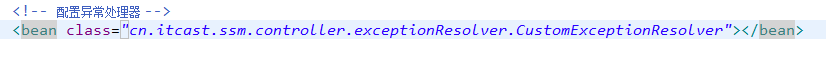
**return** modelAndView;

}

}

### 在springmvc中配置异常处理器

在springmvc.xml中配置：

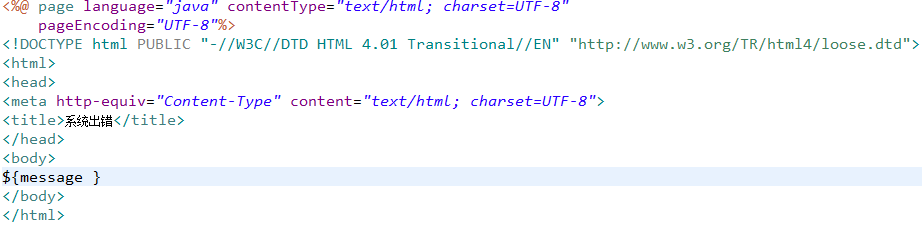


### 异常处理测试

需求：

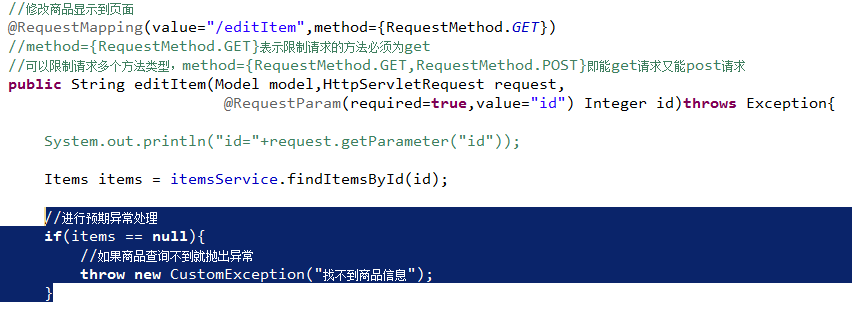
当商品信息修改时，如果根据商品id查询不到商品信息，抛出异常，异常信息：找不到商品信息。

编写error.jsp页面：



测试方法：

在controller、service中抛出异常都可以测试。

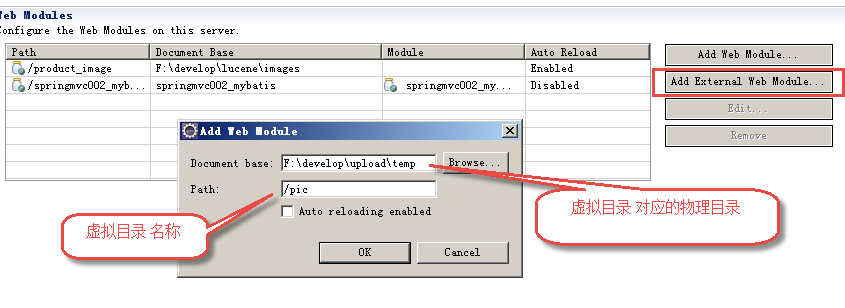


## springmvc上传图片

### 图片虚拟目录

将图片上传到固定的物理目录下，通过虚拟目录 访问物理目录 下的图片。

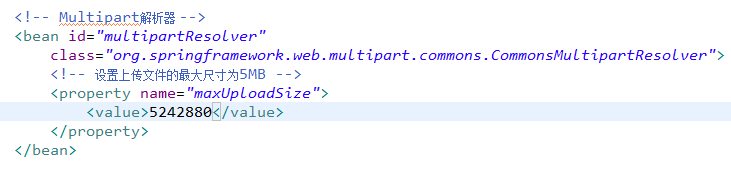
在tomcat上配置虚拟目录，访问虚拟目录可以访问虚拟目录对应物理目录



上边配置方式通过直接修改tomcat 下conf/server.xml文件配置虚拟目录：

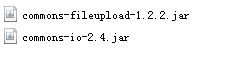


### 配置解析器



### 加入CommonsMultipartResolver解析器依赖的jar包

*CommonsMultipartResolver*解析器依赖commons-fileupload和commons-io，加入如下jar包：



### springmvc实现上传

在controller方法形参上添加MultipartFile类型的参数，参数名称和上传页面上file的名称一致，才可以绑定成功。

**public** String editItemSubmit(Model model,@ModelAttribute("item") @Validated(value={ValidGroup1.**class**}) Items items,

BindingResult bindingResult,

Integer id,//基本类型的参数,

MultipartFile picture

)**throws** Exception{

//通过model对基本参数进行回显

model.addAttribute("id", id);

//查看捕获的错误结果

**if**(bindingResult.hasErrors()){

List<ObjectError> allErrors = bindingResult.getAllErrors();

//将错误信息传到页面

model.addAttribute("allErrors", allErrors);

**for**(ObjectError error:allErrors){

System.*out*.println(error.getDefaultMessage());

}

//有了错误返回到修改商品页面

**return** "editItem";

}

//实现上传图片

**if** (picture != **null** && picture.getOriginalFilename() != **null**

&& !picture.getOriginalFilename().equals("")) {

//获取图片原始名称，目标要从原始名称中获取文件的扩展名

String originalFilename = picture.getOriginalFilename();

//新文件名称

String fileName\_new = UUID.*randomUUID*()

+ originalFilename.substring(originalFilename

.lastIndexOf("."));

//新文件

File newFile = **new** File("F:\\develop\\upload\\temp\\"+fileName\_new);

//将内存中的文件内容写入磁盘上

picture.transferTo(newFile);

//更新新文件名到数据库中

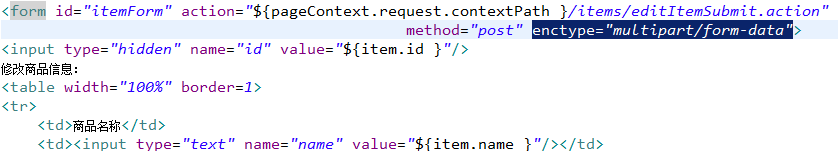
items.setPic(fileName\_new);

}

### 修改页面提交的编码enctype

在form中指定multipart编码格式：

enctype="multipart/form-data"



如果form表单上添加enctype="multipart/form-data"，而springmvc.xml中没有配置multipart解析器，此时其它类型参数（pojo，基本类型）都不能绑定成功。

所以，主要只要form中有enctype="multipart/form-data"配置，必须要springmvc.xml中配置multipart解析器

## springmvc对json数据处理

json数据为什么流行，因为json格式数据比较简单，更符合面向对象，在页面使用js解析json比较方便的。

### springmvc对json支持

请求json数据

指定contentType="applicatoin/json"

controller方法

形参：java 对象(pojo)

返回值类型：pojo类型

1、将json转成java对象

springmvc使用jackson包将json转成java对象，使用@requestBody实现

2、参数绑定

请求key/value数据

指定contentType="application/x-www-form-urlencoded "

controller方法

形参：java 对象(pojo)

返回值类型：pojo类型

1、参数绑定

将controller方法返回值转成json输出

使用@responseBody实现

将controller方法返回值转成json输出

使用jackson包将java对象转成json，通过response输出为contentType="applicatoin/json"

使用jackson包将java对象转成json，通过response输出为contentType="applicatoin/json"

输入json数据好处：为了适应众多客户端请求，可以将请求的内容类型统一为json数据格式。

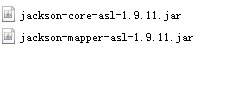
将请求结果以json格式输入好处：在页面使用js解析json比较方便

springmvc中通过@requestBody注解实现将json数据转成java对象。内部使用jackson包将json转成java对象

springmvc中通过@responseBody注解实现将java对象转成json输出。

### 准备环境

#### 加入jackson包到工程



#### 向处理器适配器配置json转换器

**如果使用<mvc:annotation-driven />不用单独配置json转换器。**

**如果不使用mvc注解驱动：**

<!--注解适配器 -->

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

<property name=*"messageConverters"*>

<list>

<bean class=*"org.springframework.http.converter.json.MappingJacksonHttpMessageConverter"*></bean>

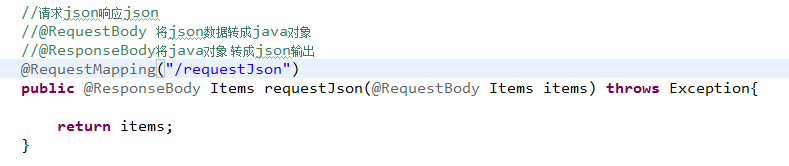
</list>

</property>

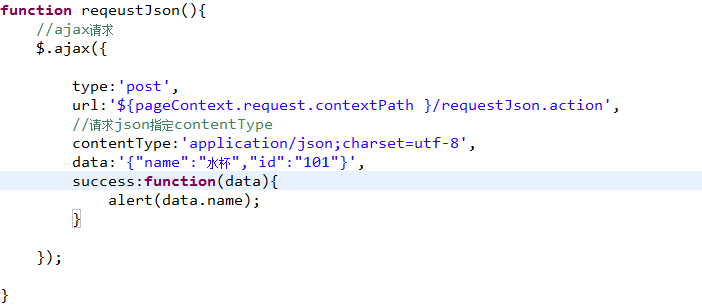
</bean>

### 请求json响应json

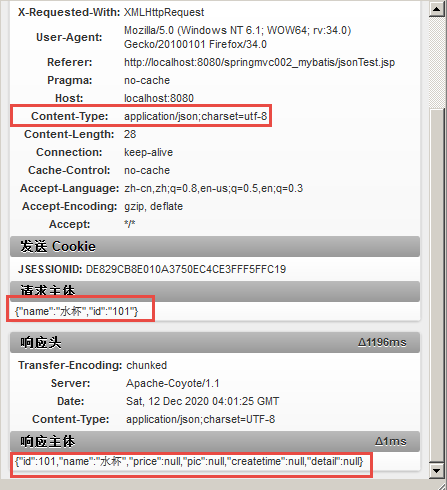
#### controller



#### jsp页面请求json解析响应的json

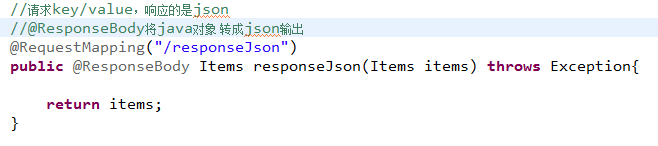


测试结果：

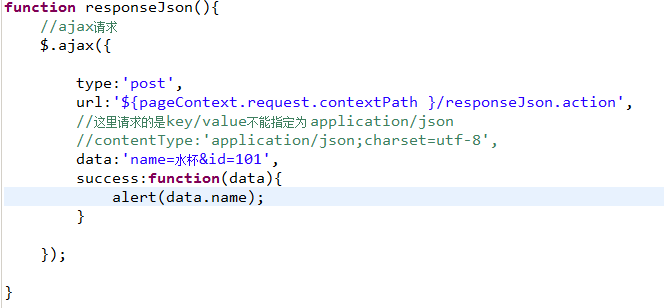


### 请求key/value响应json

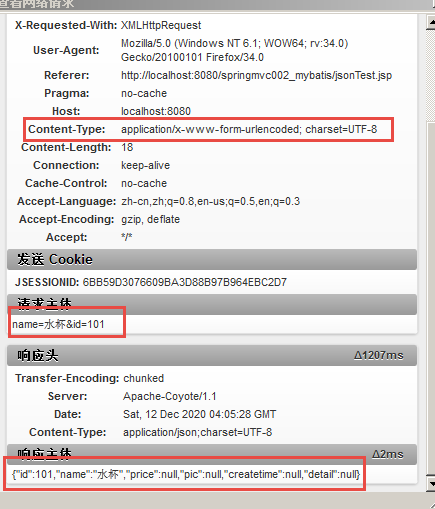
#### controller



#### jsp页面请求key/value、解析响应的数据



测试结果：



## springmvc对rest的支持

### RESTful的理解

REST，即Representational State Transfer的缩写。我对这个词组的翻译是"表现层状态转化"。

REST的名称"表现层状态转化"中，省略了主语。"表现层"其实指的是"资源"（Resources）的"表现层"。**所谓"资源"，就是网络上的一个实体，或者说是网络上的一个具体信息**

rest方式url要求简洁，一个url对应互联网上的一个资源：

http://www.itcast.cn/user/001 001用户资源 特点：将原来url请求参数放在url中传入服务。

。。。

"资源"是一种信息实体，它可以有多种外在表现形式。**我们把"资源"具体呈现出来的形式，叫做它的"表现层"（Representation）。**

一个资源可以使用html展示、xml展示、json展示，，这此展示方式都不同的表现形式

通过url输出json、xml等格式的数据。

就是HTTP协议里面，四个表示操作方式的动词：GET、POST、PUT、DELETE。它们分别对应四种基本操作：**GET用来获取资源，POST用来新建资源（也可以用于更新资源），PUT用来更新资源，DELETE用来删除资源。**

比如：

请求url：http://www.itcast.cn/user/001

当请求的方法为get时，表示获取001的用户资源，服务端程序应该根据http的方法类型判断应该查询001用户

请求url：http://www.itcast.cn/user/001

当请求的方法为post时，表示新建一个001的用户资源。

### 需求

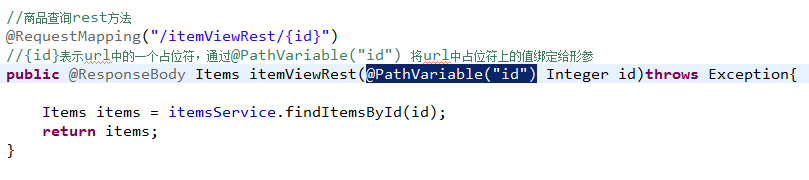
查询商品信息，将商品信息输出为json。

分析：

url：springmvc采用rest风格的url，将参数通过url传入 controller方法。

输出json：springmvc采用@responseBody将java对象输出json数据。

### controller方法



#### url模板映射

@RequestMapping(value="/ viewItems/{id}")：{×××}占位符，请求的URL可以是“/viewItems/1”或“/viewItems/2”，通过在方法中使用@PathVariable获取{×××}中的×××变量。

@PathVariable用于将请求URL中的模板变量映射到功能处理方法的参数上。

@RequestMapping("/viewItems/{id}")

**public** @ResponseBody viewItems(@PathVariable("id") String id,Model model) **throws** Exception{

//方法中使用@PathVariable获取useried的值，使用model传回页面

//调用 service查询商品信息

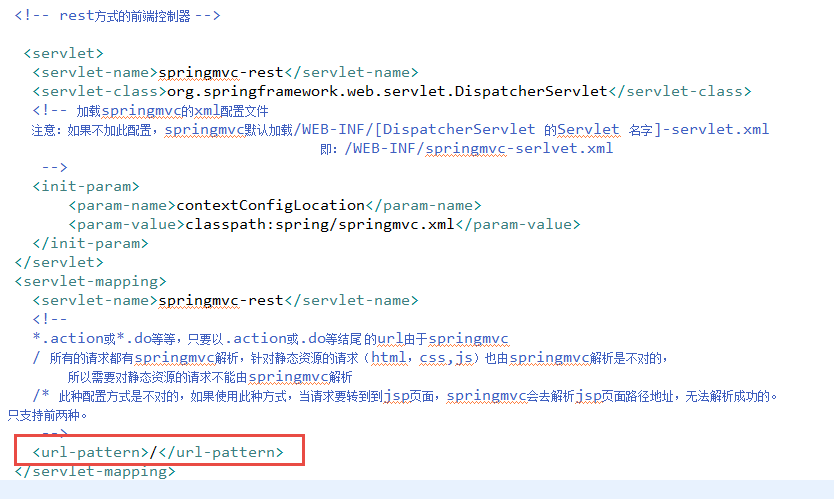
ItemsCustom itemsCustom = itemsService.findItemsById(id);

**return** itemsCustom;

}

如果RequestMapping中表示为"/viewItems/{id}"，id和形参名称一致，@PathVariable不用指定名称。

### 配置前端控制器支持rest



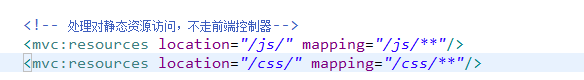
### 处理静态资源请求

如果在DispatcherServlet中设置url-pattern为 /则必须对静态资源进行访问处理。

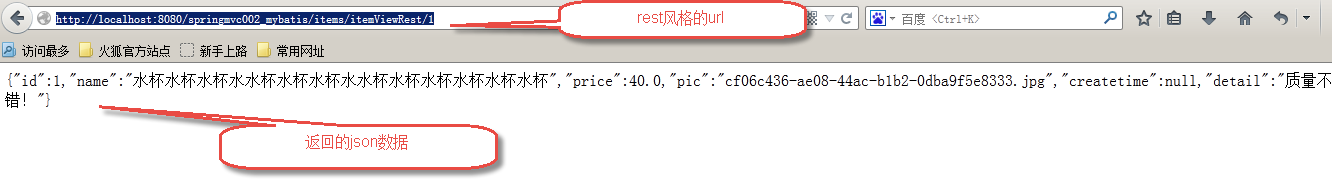
spring mvc 的<mvc:resources mapping="" location="">实现对静态资源进行映射访问。

如下是对js文件访问配置：

<mvc:resources location="/js/" mapping="/js/\*\*"/>



### 测试结果



## springmvc拦截器

### 拦截器的作用

springmvc提供拦截器实现对Handler进行面向切面编程，可以Handler执行之前、之后、之中添加代码，这种方式就是切面编程。

拦截器相当 于一个过虑器，可以预处理、后处理等。

使用拦截器实现：

1、在Handler执行之前拦截校验用户权限，如果用户没有操作权限就不能继续执行Handler.

2、Handler执行完成后统一在拦截器中记录操作日志。

### springmvc拦截器定义

springmvc提供HanlderInterceptor接口，实现该 接口的就是拦截器。

**public** **class** HandlerInterceptor1 **implements** HandlerInterceptor {

//执行时机：Handler执行之前去执行

//使用场景：用于用户权限拦截校验

@Override

**public** **boolean** preHandle(HttpServletRequest request,

HttpServletResponse response, Object handler) **throws** Exception {

//返回 true表示不拦截即放行

//返回false表示拦截，不再向后执行

**return** **false**;

}

//执行时机：Handler执行未返回modelAndView之前 去执行

//使用场景 ：将页面需要通用的信息通过modelAndView传到页面，比如：菜单 导航通用在此方法取出来传到页面

//比如：统一定义视图决定视图转向

@Override

**public** **void** postHandle(HttpServletRequest request,

HttpServletResponse response, Object handler,

ModelAndView modelAndView) **throws** Exception {

}

//执行时机：Handler执行完成并且 将modelAndView返回

//使用场景 ：统一进行异常处理，统一记录操作日志。。。

@Override

**public** **void** afterCompletion(HttpServletRequest request,

HttpServletResponse response, Object handler, Exception ex)

**throws** Exception {

}

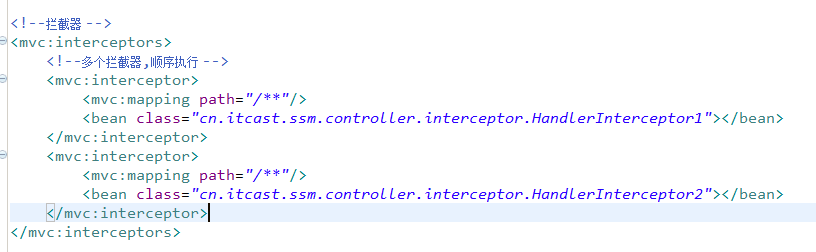
}

### 拦截器配置

springmvc的拦截器针对HandlerMapping处理器映射器配置的。

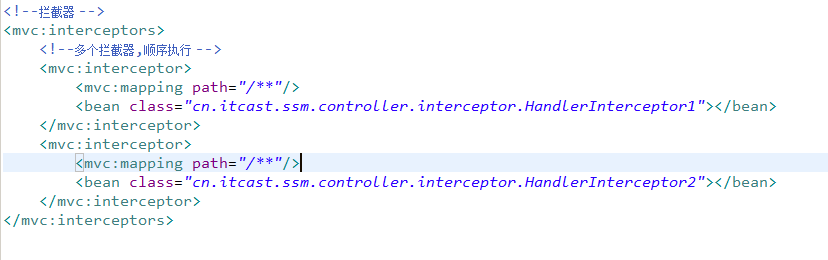
针对所有mapping配置全局拦截器：

springmvc中没有全局拦截器概念，使用这种配置，springmvc将配置的拦截器分别注入到多个处理器映射器。



### springmvc拦截器测试

使用两个拦截器进行测试，两个拦截器配置为顺序执行。



#### 全都放行

两个拦截器打印日志如下：

HandlerInterceptor1...preHandle

HandlerInterceptor2...preHandle

HandlerInterceptor2...postHandle

HandlerInterceptor1...postHandle

HandlerInterceptor2...afterCompletion

HandlerInterceptor1...afterCompletion

preHandle是按照拦截器配置顺序去执行

postHandle和afterCompletion按照拦截器配置相反顺序去执行。

#### 第一个放行，第二个不放行

两个拦截器打印日志如下：

HandlerInterceptor1...preHandle

HandlerInterceptor2...preHandle

HandlerInterceptor1...afterCompletion

preHandle不放行，postHandle和afterCompletion不会执行

只要preHandle放行，afterCompletion就执行。

只要有一个拦截器不放行，所有拦截器postHandle都不执行。

#### 都不放行

两个拦截器打印日志如下：

HandlerInterceptor1...preHandle

前边的拦截器不放行，后边拦截器不执行。

#### 思考

实现统一异常处理拦截器如何实现？

实现如下：

定义一个拦截器，在afterCompletion实现异常处理。

必须要将此拦截器配置第一个位置，且preHandle要放行。

### springmvc拦截器练习

#### 需求

通过springmvc拦截器实现用户权限校验，当用户请求方法时校验该 用户是否登陆，如果没有登陆跳转到登陆页面让用户登陆。

#### 分析

拦截器实现：

在preHandle方法中实现拦截的业务逻辑

业务逻辑执行流程：

1、进入preHandler方法表示已经拦截了用户的请求

2、判断用户访问的url是否公开地址（不登陆即可访问的址），比如：首页、登陆url

如果是公开 地址就放行，如果不是公开地址向下走流程

3、判断用户当前是否登陆，当前session中是否存在用户身份信息

如果存在就放行，如果不存在跳转到登陆页面

#### 拦截器实现

**public** **class** LoginHandlerInterceptor **implements** HandlerInterceptor {

//执行时机：Handler执行之前去执行

//使用场景：用于用户权限拦截校验

@Override

**public** **boolean** preHandle(HttpServletRequest request,

HttpServletResponse response, Object handler) **throws** Exception {

//获取用户的url

String url = request.getRequestURI();

//判断用户访问的url是否公开地址,如果是则放行,实际开发中要将公开 地址配置在配置文件中

**if**(url.indexOf("/login.action")>=0){//如果用户请求的登陆url是公开地址

//放行

**return** **true**;

}

//判断用户当前是否登陆，从当前session中取出用户身份信息

//session中身份信息在用户登陆成功后设置进去的，这里用户身份信息key为activeUser

HttpSession session = request.getSession();

String activeUser = (String) session.getAttribute("activeUser");

**if**(activeUser!=**null**){

//如果session中存在用户身份信息，则放行

**return** **true**;

}

//用户身份信息不合法，跳转登陆页面

request.getRequestDispatcher("/WEB-INF/jsp/login.jsp").forward(request, response);

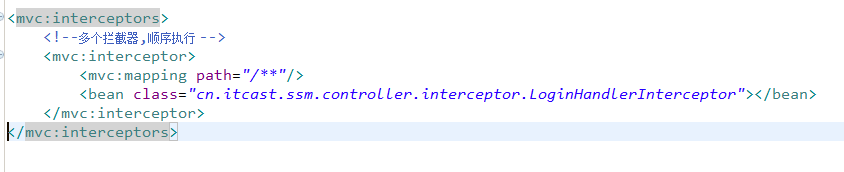
//返回 true表示不拦截即放行

//返回false表示拦截，不再向后执行

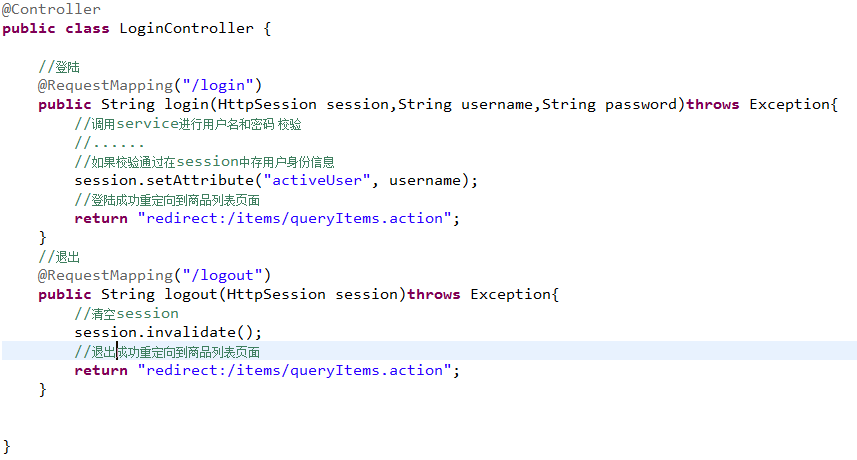
**return** **false**;

}

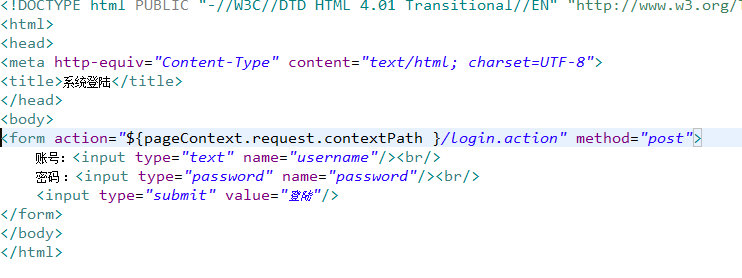
#### 拦截器配置



#### 登陆的controller方法



#### 登陆页面



## 问题

### 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：

修改tomcat目录 的conf/server.xml文件，添加URIEncoding="utf-8"

<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