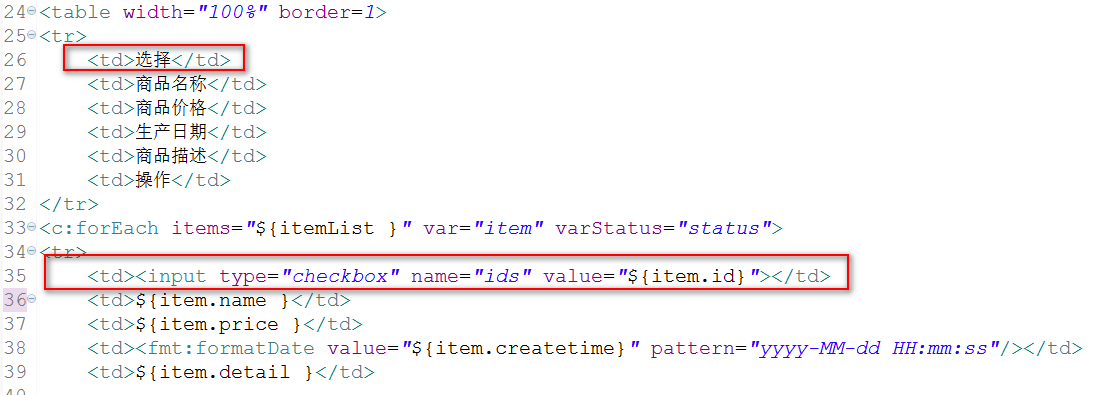
# SpringMVC第二天课堂笔记

# 高级参数绑定

## 数组类型的参数绑定

*基于批量删除商品操作完成参数传递。*

**Jsp修改**



**控制器修改**

/\*\*

\* 包装pojo传递演示

\* 第二天，演示数组参数的传递与List的传递

\* **@param** vo

\* **@return**

\*/

@RequestMapping("queryItem")

**public** String queryItem(QueryVo vo, Integer[] ids) {

System.***out***.println(vo);

//输出传入的数组

**if** (ids != **null** && ids.length > 0) {

**for** (Integer id : ids) {

System.***out***.println("传入的商品列表分别为：" + id);

}

}

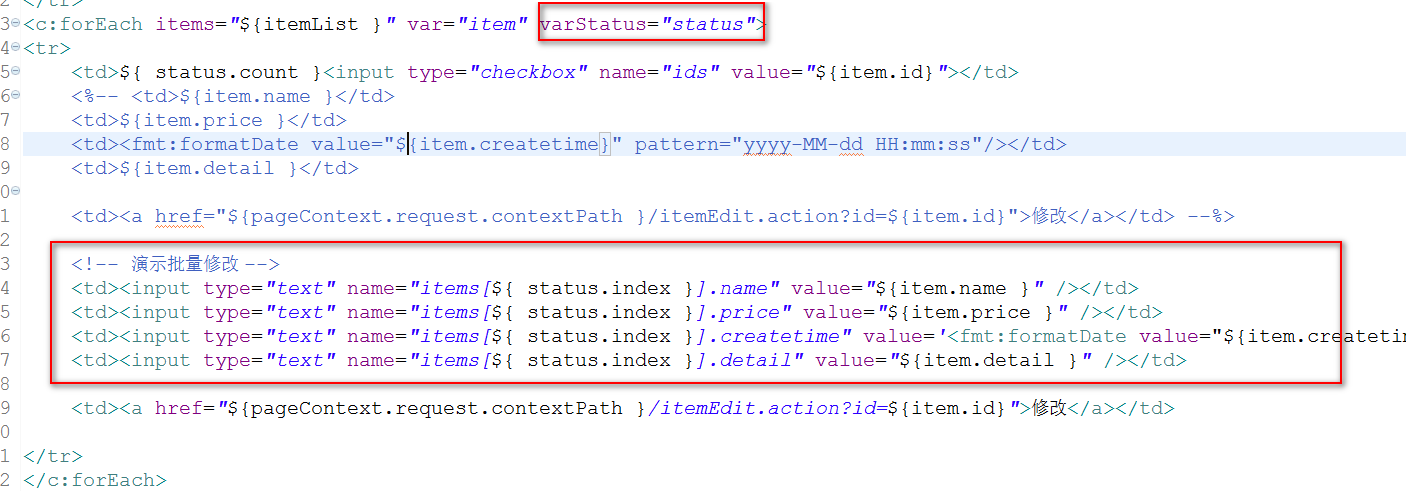
**return** "itemList";

}

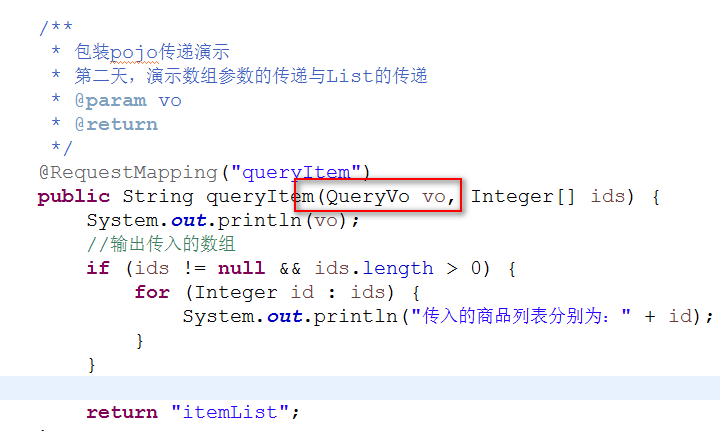
## List类型的绑定

*基于批量修改商品操作完成参数传递。*

**Jsp修改**



**控制器修改**



**POJO修改**



# @RequestMapping注解的使用

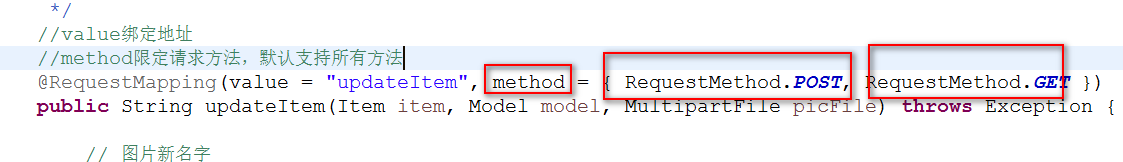
## 路径映射可以是数组



## @RequestMapping可以加在类头部，用于目录分级管理



## 限定请求方法



# Controller方法返回值

## 返回ModelAndView

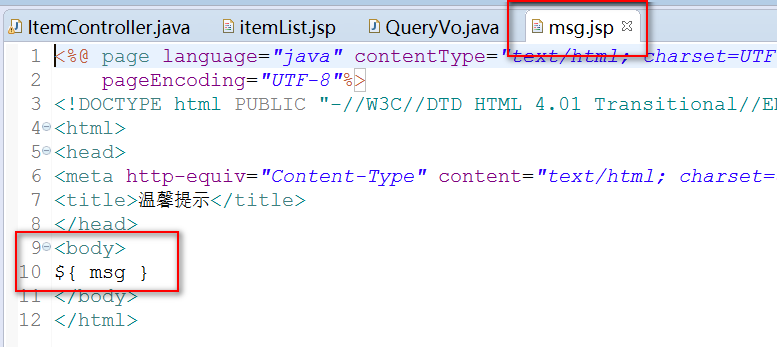
参考第一天内容。

## 返回void

### request



新建msg.jsp



### response



## 返回String

### 返回视图名字，参考第一天内容

### redirect与forward



# Springmvc中异常处理

思想：做一个全局异常处理器，处理所有没有处理过的运行时异常用于更友好地提示用户。

## 步骤

### 创建全局异常处理器

/\*\*

\* 全局异常处理器

\* **@author** Steven

\*

\*/

**public** **class** CustomerException **implements** HandlerExceptionResolver {

@Override

**public** ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object hanlder,

Exception e) {

//记录日志

e.printStackTrace();

//错误消息

String msg = "很抱歉，系统发生异常了，请联系管理员";

//响应用户错误提示

ModelAndView mav = **new** ModelAndView();

//返回错误消息

mav.addObject("msg", msg);

//响应错误提示页面

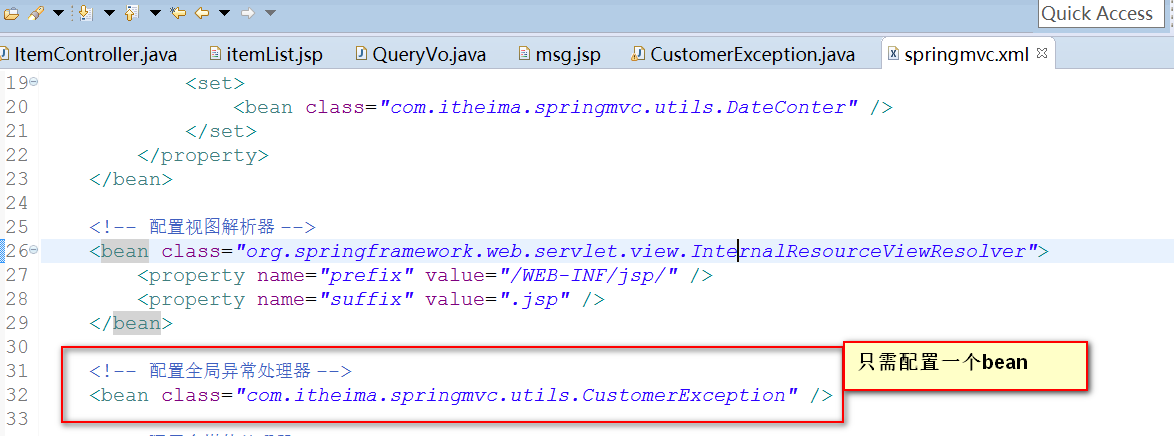
mav.setViewName("msg");

**return** mav;

}

}

### 配置异常处理器



### 测试访问queryVoid.action



### 更智能、更友好的提示，解决方案

#### 新建自定义异常类

/\*\*

\* 自定义异常

\* **@author** Steven

\*

\*/

**public** **class** MyException **extends** Exception {

//错误消息

**private** String msg;

**public** MyException() {

**super**();

}

**public** MyException(String msg) {

**super**();

**this**.msg = msg;

}

**public** String getMsg() {

**return** msg;

}

**public** **void** setMsg(String msg) {

**this**.msg = msg;

}

}

#### 修改异常处理器，加上异常判断

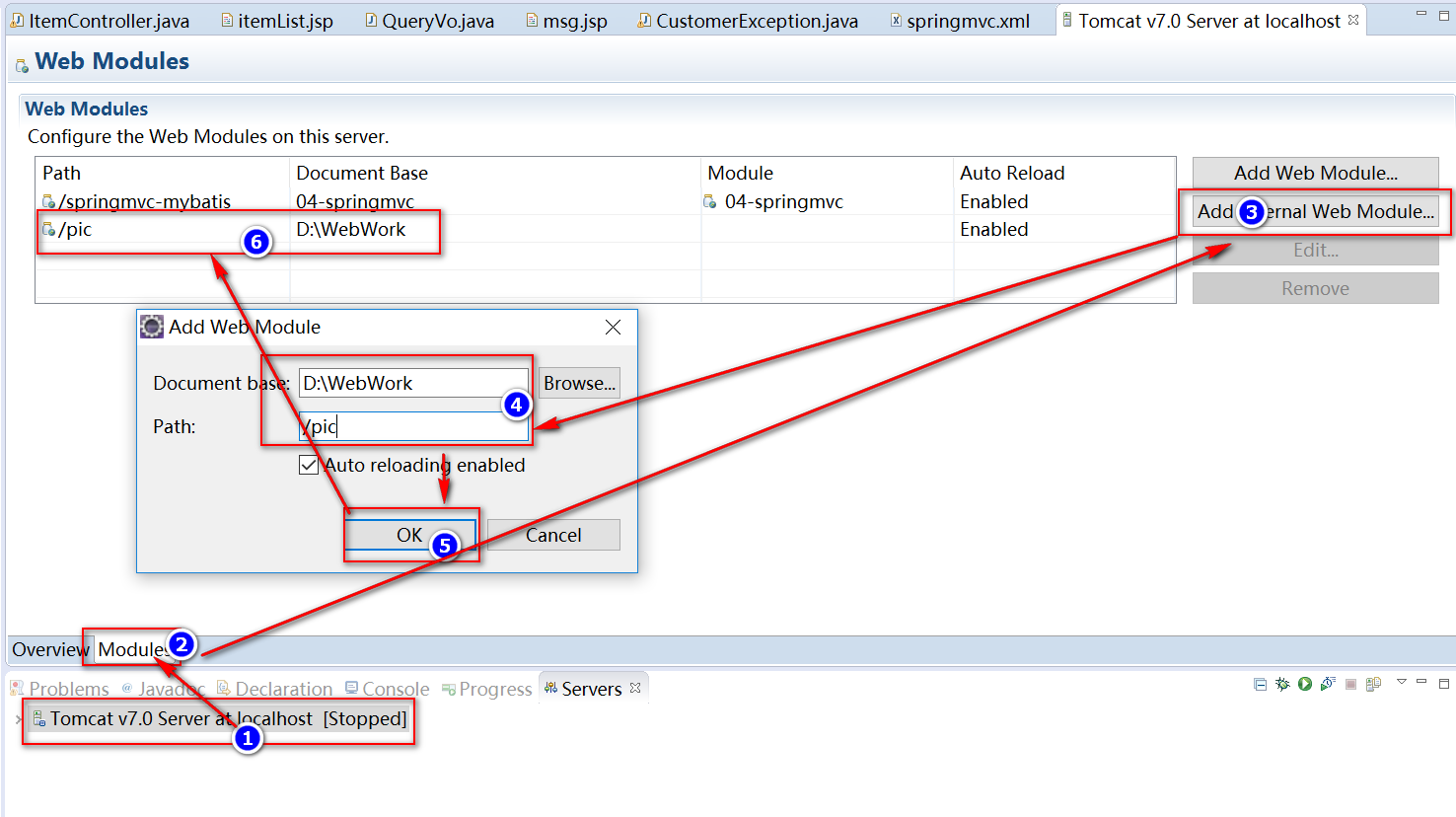


#### 测试访问queryVoid.action

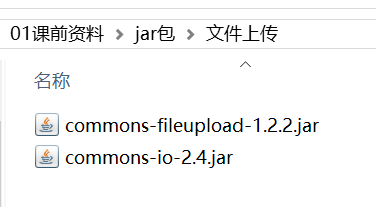


# 图片上传处理

## 配置虚拟目录



## 加入上传功能需要的jar包



## 配置多媒体解析器

<!-- 配置多媒体处理器 -->

<!-- 注意：这里id必须填写：multipartResolver -->

<bean id=*"multipartResolver"* class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>

<!-- 最大上传文件大小 -->

<property name=*"maxUploadSize"* value=*"8388608"* />

</bean>

## jsp修改



## 编写图片上传处理代码

//图片上传用MultipartFile接收文件

@RequestMapping(value = "updateItem", method = { RequestMethod.***POST***, RequestMethod.***GET*** })

**public** String updateItem(Item item, Model model, MultipartFile picFile) **throws** Exception {

// 图片新名字

String name = UUID.*randomUUID*().toString();

// 图片原名字

String oldName = picFile.getOriginalFilename();

// 后缀名

String exeName = oldName.substring(oldName.lastIndexOf("."));

File pic = **new** File("D:\\WebWork\\" + name + exeName);

// 保存图片到本地磁盘

picFile.transferTo(pic);

// 更新商品图片信息

item.setPic(name + exeName);

itemServices.update(item);

model.addAttribute("item", item);

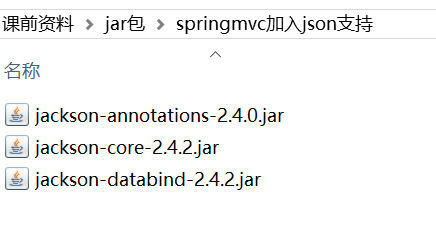
model.addAttribute("msg", "修改商品成功");

**return** "itemEdit";

}

# json数据交互

## 加入jar包



## 编码

/\*\*

\* json数据交互演示

\*

\* **@param** item2

\* **@return**

\*/

@RequestMapping("getItem")

//@ResponseBody把pojo转成json串响应用户

@ResponseBody

//@RequestBody用于接收用户传入json串转成pojo

**public** Item getItem(@RequestBody Item item2) {

System.***out***.println("接收到的json商品数据为：" + item2);

Item item = itemServices.getItemById(3);

**return** item;

}

## 测试，安装google浏览器测试工具

*详情可查看教案上的安装谷歌浏览器工具节点*

# Springmvc实现Restful

## 编码

/\*\*

\* RESTful风格演示

\*

\* **@param** ids

\* **@param** model

\* **@return**

\*/

//RESTful风格url上的参数通过{}点位符绑定

//点位符参数名与方法参数名不一致时，通过@PathVariable绑定

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

**public** String testRest(@PathVariable("id") Integer ids, Model model) {

Item item = itemServices.getItemById(ids);

model.addAttribute("item", item);

**return** "itemEdit";

}

## 测试

测试直接访问url{**http://localhost:8080/项目名/item/1.action**}即可。

# 拦截器

## 拦截器开发流程

### 创建拦截器

#### 一号拦截器

/\*\*

\* 自定义拦截器

\* **@author** Steven

\*

\*/

**public** **class** MyInterceptor1 **implements** HandlerInterceptor {

//在Controller方法执行后被执行

//处理异常、记录日志

@Override

**public** **void** afterCompletion(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, Exception arg3)

**throws** Exception {

System.***out***.println("MyInterceptor1.afterCompletion.....");

}

//在Controller方法执行后，返回ModelAndView之前被执行

//设置或者清理页面共用参数等等

@Override

**public** **void** postHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, ModelAndView arg3)

**throws** Exception {

System.***out***.println("MyInterceptor1.postHandle.....");

}

//在Controller方法执行前被执行

//登录拦截、权限认证等等

@Override

**public** **boolean** preHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2) **throws** Exception {

System.***out***.println("MyInterceptor1.preHandle.....");

//返回true放行，false拦截

**return** **true**;

}

}

#### 二号拦截器，复制一号，修改一下系统输出消息

### 配置拦截器

<!-- 拦截器定义 -->

<mvc:interceptors>

<!-- 定义一个拦截器 -->

<mvc:interceptor>

<!-- path配置</\*\*>拦截所有请求，包括二级以上目录，</\*>拦截所有请求，不包括二级以上目录 -->

<mvc:mapping path=*"/\*\*"*/>

<bean class=*"com.itheima.springmvc.interceptor.MyInterceptor1"* />

</mvc:interceptor>

<!-- 定义一个拦截器 -->

<mvc:interceptor>

<!-- path配置</\*\*>拦截所有请求，包括二级以上目录，</\*>拦截所有请求，不包括二级以上目录 -->

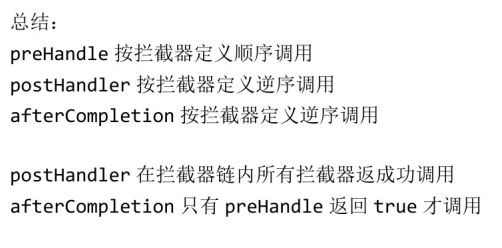
<mvc:mapping path=*"/\*\*"*/>

<bean class=*"com.itheima.springmvc.interceptor.MyInterceptor2"* />

</mvc:interceptor>

</mvc:interceptors>

### 测试流程，可参考教案



## 拦截器案例应用，登录拦截器

### 思路

1. 有一个登录页面，需要写一个controller访问页面
2. 登录页面有一提交表单的动作。需要在controller中处理。
   1. 判断用户名密码是否正确
   2. 如果正确 想session中写入用户信息
   3. 返回登录成功，或者跳转到商品列表
3. 拦截器。
   1. 拦截用户请求，判断用户是否登录
   2. 如果用户已经登录。放行
   3. 如果用户未登录，跳转到登录页面。

### login.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<title>用户登录</title>

</head>

<body>

<form action=*"*${pageContext.request.contextPath }*/user/login.action"*>

用户名：<input type=*"text"* name=*"username"* /><br>

密码：<input type=*"password"* name=*"password"* /><br>

<input type=*"submit"*>

</form>

</body>

</html>

### UserController

/\*\*

\* 登录拦截器

\* **@author** Steven

\*

\*/

@Controller

@RequestMapping("user")

**public** **class** UserController {

/\*\*

\* 跳转登录页面

\* **@return**

\*/

@RequestMapping("toLogin")

**public** String toLogin(){

**return** "login";

}

/\*\*

\* 用户登录

\* **@return**

\*/

@RequestMapping("login")

**public** String login(String username,String password,HttpSession session){

String result = "login";

//断定用户是否允许登录

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

session.setAttribute("username", username);

//登录成功，跳转商品列表

result = "redirect:/itemList.action";

}

**return** result;

}

}

### LoginInterceptor拦截器编码

*这里省略类声名和其它两个方法。*

//处理执行前被执行

//登录拦截、权限验证

@Override

**public** **boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object arg2) **throws** Exception {

Object attribute = request.getSession().getAttribute("username");

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

//放行，执行处理器

**return** **true**;

}**else**{

response.sendRedirect(request.getContextPath() + "/user/tologin.action");

//放行，执行处理器

**return** **false**;

}

}

### 拦截器配置

<!-- 登录拦截器 -->

<mvc:interceptor>

<!-- path配置</\*\*>拦截所有请求，包括二级以上目录，</\*>拦截所有请求，不包括二级以上目录 -->

<mvc:mapping path=*"/\*\*"*/>

<!-- 配置不拦截请求的地址 -->

<!-- <mvc:exclude-mapping path="/user/toLogin.action"/>

<mvc:exclude-mapping path="/user/login.action"/> -->

<mvc:exclude-mapping path=*"/user/\*"*/>

<bean class=*"com.itheima.springmvc.interceptor.LoginInterceptor"* />

</mvc:interceptor>