**Spring web mvc**

**框架课程**

# 注解开发-高级

## 上传图片

### 配置虚拟目录

1.enctype=*"multipart/form-data"*

2.post

### 配置解析器

<!-- 文件上传,这里的bean必须给id或name-->

<bean id=*"multipartResolver"*

class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>

<!-- 设置上传文件的最大尺寸为5MB -->

<property name=*"maxUploadSize"*>

<value>5242880</value>

</property>

</bean>

### jar包

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



### 图片上传

* controller：

//商品修改提交

@RequestMapping("/editItemSubmit")

**public** String editItemSubmit(Items items, MultipartFile pictureFile)**throws** Exception{

//原始文件名称

String pictureFile\_name = pictureFile.getOriginalFilename();

//新文件名称

String newFileName = UUID.*randomUUID*().toString()+pictureFile\_name.substring(pictureFile\_name.lastIndexOf("."));

//上传图片

File uploadPic = **new** java.io.File("F:/develop/upload/temp/"+newFileName);

**if**(!uploadPic.exists()){

uploadPic.mkdirs();

}

//向磁盘写文件

pictureFile.transferTo(uploadPic);

.....

* 页面：

form添加enctype="multipart/form-data"：

<form id=*"itemForm"*

action=*"*${pageContext.request.contextPath }*/item/editItemSubmit.action"*

method=*"post"* enctype=*"multipart/form-data"*>

<input type=*"hidden"* name=*"pic"* value=*"*${item.pic }*"* />

file的name与controller形参一致：

<tr>

<td>商品图片</td>

<td><c:if test=*"*${item.pic !=null}*"*>

<img src=*"/pic/*${item.pic}*"* width=*100* height=*100* />

<br />

</c:if> <input type=*"file"* name=*"pictureFile"* /></td>

</tr>

## json数据交互

### @RequestBody

作用：

@RequestBody注解用于读取http请求的内容(字符串)，通过springmvc提供的HttpMessageConverter接口将读到的内容转换为json、xml等格式的数据并绑定到controller方法的参数上。

本例子应用：

@RequestBody注解实现接收http请求的json数据，将json字符串（不是json对象）转换为java对象

### @ResponseBody

作用：

该注解用于将Controller的方法返回的对象，通过HttpMessageConverter接口转换为指定格式的数据如：json,xml等，通过Response响应给客户端

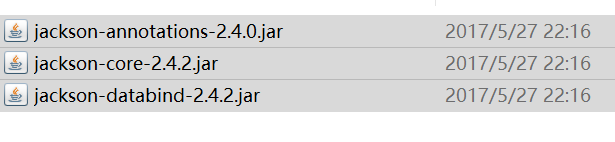
本例子应用：

@ResponseBody注解实现将controller方法返回对象转换为json响应给客户端

### 请求json，响应json实现：

#### 环境准备

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



#### 配置json转换器

在注解适配器中加入messageConverters

<!--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>

**注意：如果使用<mvc:annotation-driven /> 则不用定义上边的内容。**

#### controller编写

// 商品修改提交json信息，响应json信息

@RequestMapping("/editItemSubmit\_RequestJson")

**public** @ResponseBody Items editItemSubmit\_RequestJson(@RequestBody Items items) **throws** Exception {

System.*out*.println(items);

//itemService.saveItem(items);

**return** items;

}

#### 页面js方法编写：

引入 js：

<script type="text/javascript"

src="${pageContext.request.contextPath }/js/jquery-1.4.4.min.js"></script>

//请求json响应json

**function** request\_json(){

$.ajax({

type:"post",

url:"${pageContext.request.contextPath }/item/editItemSubmit\_RequestJson.action",

contentType:"application/json;charset=utf-8",

data:'{"name":"测试商品","price":99.9}',

success:**function**(data){

alert(data);

}

});

}

#### 测试结果：



从上图可以看出请求的数据是json格式

### 请key/value，响应json实现：

表单默认请求application/x-www-form-urlencoded格式的数据即key/value，通常有post和get两种方法，响应json数据是为了方便客户端处理，实现如下：

#### 环境准备

同第一个例子

#### controller编写

// 商品修改提交，提交普通form表单数据，响应json

@RequestMapping("/editItemSubmit\_ResponseJson")

**public** @ResponseBody Items editItemSubmit\_ResponseJson(Items items) **throws** Exception {

System.*out*.println(items);

// itemService.saveItem(items);

**return** items;

}

#### 页面js方法编写：

**function** formsubmit(){

**var** user = " name=测试商品&price=99.9";

alert(user);

$.ajax(

{

type:'post',//这里改为get也可以正常执行

url:'${pageContext.request.contextPath}/item/ editItemSubmit\_RequestJson.action',

//ContentType没指定将默认为：application/x-www-form-urlencoded

data:user,

success:**function**(data){

alert(data.name);

}

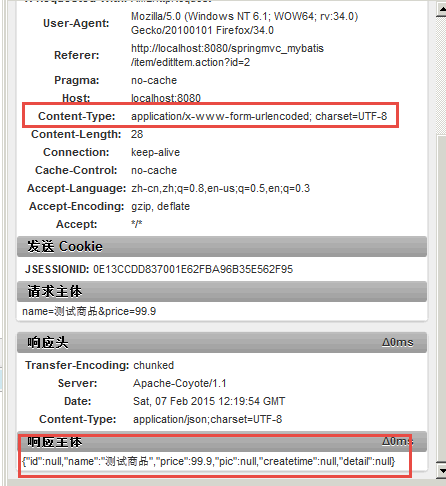
}

)

}

从上边的js代码看出，已去掉ContentType的定义，ContentType默认为：application/x-www-form-urlencoded格式。

#### 测试结果



从上图可以看出请求的数据是标准的key/value格式。

### 小结

实际开发中常用第二种方法，请求key/value数据，响应json结果，方便客户端对结果进行解析。

# 拦截器

## 定义

Spring Web MVC 的处理器拦截器类似于Servlet 开发中的过滤器Filter，用于对处理器进行预处理和后处理。

## 拦截器定义

实现HandlerInterceptor接口，如下：

**Public class** HandlerInterceptor1 **implements** HandlerInterceptor{

/\*\*

\* controller执行前调用此方法

\* 返回true表示继续执行，返回false中止执行

\* 这里可以加入登录校验、权限拦截等

\*/

@Override

**Public boolean** preHandle(HttpServletRequest request,

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

// **TODO** Auto-generated method stub

**Return false**;

}

/\*\*

\* controller执行后但未返回视图前调用此方法

\* 这里可在返回用户前对模型数据进行加工处理，比如这里加入公用信息以便页面显示

\*/

@Override

**Public void** postHandle(HttpServletRequest request,

HttpServletResponse response, Object handler,

ModelAndView modelAndView) **throws** Exception {

// **TODO** Auto-generated method stub

}

/\*\*

\* controller执行后且视图返回后调用此方法

\* 这里可得到执行controller时的异常信息

\* 这里可记录操作日志，资源清理等

\*/

@Override

**Public void** afterCompletion(HttpServletRequest request,

HttpServletResponse response, Object handler, Exception ex)

**throws** Exception {

// **TODO** Auto-generated method stub

}

}

## 拦截器配置

### 针对某种mapping配置拦截器

<bean

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

<property name=*"interceptors"*>

<list>

<ref bean=*"handlerInterceptor1"*/>

<ref bean=*"handlerInterceptor2"*/>

</list>

</property>

</bean>

<bean id=*"handlerInterceptor1"* class=*"springmvc.intercapter.HandlerInterceptor1"*/>

<bean id=*"handlerInterceptor2"* class=*"springmvc.intercapter.HandlerInterceptor2"*/>

### 针对所有mapping配置全局拦截器

<!--拦截器 -->

<mvc:interceptors>

<!--多个拦截器,顺序执行 -->

<mvc:interceptor>

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

<bean class=*"com.qf.springmvc.filter.HandlerInterceptor1"*></bean>

</mvc:interceptor>

<mvc:interceptor>

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

<bean class=*"com.qf.springmvc.filter.HandlerInterceptor2"*></bean>

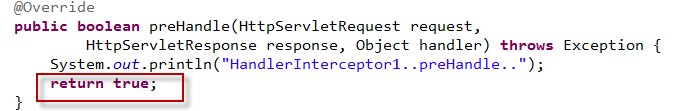
</mvc:interceptor>

</mvc:interceptors>

## 正常流程测试

### 代码：

定义两个拦截器分别为：HandlerInterceptor1和HandlerInteptor2，每个拦截器的preHandler方法都返回true。



### 运行流程

HandlerInterceptor1..preHandle..

HandlerInterceptor2..preHandle..

HandlerInterceptor2..postHandle..

HandlerInterceptor1..postHandle..

HandlerInterceptor2..afterCompletion..

HandlerInterceptor1..afterCompletion..

## 中断流程测试

### 代码：

定义两个拦截器分别为：HandlerInterceptor1和HandlerInteptor2。

### 运行流程

HandlerInterceptor1的preHandler方法返回false，HandlerInterceptor2返回true，运行流程如下：

HandlerInterceptor1..preHandle..

从日志看出第一个拦截器的preHandler方法返回false后第一个拦截器只执行了preHandler方法，其它两个方法没有执行，第二个拦截器的所有方法不执行，且controller也不执行了。

HandlerInterceptor1的preHandler方法返回true，HandlerInterceptor2返回false，运行流程如下：

HandlerInterceptor1..preHandle..

HandlerInterceptor2..preHandle..

HandlerInterceptor1..afterCompletion..

从日志看出第二个拦截器的preHandler方法返回false后第一个拦截器的postHandler没有执行，第二个拦截器的postHandler和afterCompletion没有执行，且controller也不执行了。

总结：

preHandle按拦截器定义顺序调用

postHandler按拦截器定义逆序调用

afterCompletion按拦截器定义逆序调用

postHandler在拦截器链内所有拦截器返成功调用

afterCompletion只有preHandle返回true才调用

## 拦截器应用

### 用户身份认证

**Public class** LoginInterceptor **implements** HandlerInterceptor{

@Override

**Public boolean** preHandle(HttpServletRequest request,

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

//如果是登录页面则放行

**if**(request.getRequestURI().indexOf("login.action")>=0){

**return true**;

}

HttpSession session = request.getSession();

//如果用户已登录也放行

**if**(session.getAttribute("user")!=**null**){

**return true**;

}

//用户没有登录跳转到登录页面

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

**return false**;

}

}

### 用户登陆controller

//登陆提交

//userid：用户账号，pwd：密码

@RequestMapping("/login")

**public** String loginsubmit(HttpSession session,String userid,String pwd)**throws** Exception{

//向session记录用户身份信息

session.setAttribute("activeUser", userid);

**return** "redirect:item/queryItem.action";

}

//退出

@RequestMapping("/logout")

**public** String logout(HttpSession session)**throws** Exception{

//session过期

session.invalidate();

**return** "redirect:item/queryItem.action";

}

# 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