## Spring MVC框架

### 什么SpringMVC

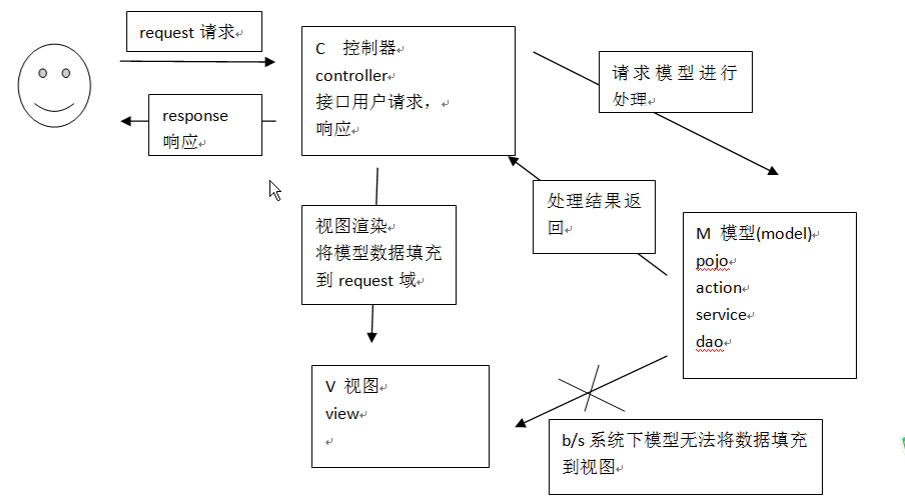
springmvc是spring框架下的一个模块,springmvc和spring无需通过中间整合层进行整合.



springmvc是一个基于mvc的web框架

### 1.2 mvc在b/s系统下的应用

mvc是一种设计模式, mvc在b/s下的应用:



1. 用户发起request请求至控制器(Controller)

控制接收用户请求的数据，委托给模型进行处理

1. 控制器通过模型(Model)处理数据并得到处理结果

模型通常是指业务逻辑

1. 模型处理结果返回给控制器
2. 控制器将模型数据在视图(View)中展示

web中模型无法将数据直接在视图上显示，需要通过控制器完成。如果在C/S应用中模型是可以将数据在视图中展示的。

1. 控制器将视图response响应给用户

通过视图展示给用户要的数据或处理结果。

### 1.3 springmvc框架



1. 发送请求到前端控制器(DispatcherServlet)
2. 前端控制器请求HandlerMapping查找Handler

可以根据xml配置,注解进行查找

1. 处理器映射器handlerMapping向前端控制器返回Handler
2. 前端控制器调用处理器适配器去(HandlerAdapter)执行Handler
3. 处理器适配器去执行handler
4. Handler执行完成给适配器返回ModelAndView
5. 处理器适配器向前端控制器返回ModelAndView

ModelAndView是Springmvc框架的一个底层对象, 包括Moder和view

1. 前端控制器请求视图解析器去进行视图解析

根据逻辑视图名解析成真正的视图(jsp)

1. 视图解析器向前端控制器返回view
2. 前端控制器进行视图渲染

视图渲染将模型数据(在ModelAndView对象中)填充到request中

11. 前端控制器向用户响应结果

组件:

1. 前端控制器 DispatcherServlet(不需要程序袁开发)

作用接收请求,响应结果,相当于转发器,中央处理器

有了DispatcherServlet减少了其它组件之间的耦合度

1. 处理器映射器 HandlerMapping(不需要程序员开发)

作用: 根据请求的 url查找Handler

1. 处理器适配器 HandlerAdapter

作用: 按照特定规则(HandlerAdapter要求规则)去执行Handler

注意: 编写Handler时按照handlerAdapter的要求去做,这样适配器才可以正确执行Handler

1. 视图解析器 View resolver

作用: 进行试图解析, 根据逻辑视图名解析成真正的视图

一般情况下需要通过页面标签或页面模版技术将模型数据通过页面展示给用户，需 要由程序员根据业务需求开发具体的页面。

1. 视图 View

View是一个接口, 实现类支持不同的View类型(jsp,freemaker,pdf,excel...)

1. 处理器 handler

由于Handler涉及到具体的用户业务请求，所以一般情况需要程序员根据业务需求 开发Handler。

## Spring MVC 入门

Springmvc框架:

DispatcherServlet前端控制器: 接收request, 进行response

HanderMapping处理器映射器: 根据url查找Handler. (可以通过xml配置方式,注解方式)

HandlerAdapter处理器适配器: 根据特定规则去执行Handler,编写Handler时需要按照HandlerAdapter的要求去编写.

Handler处理器(后端控制器): 需要自己去编写.

Handler处理后结果是modelAndView,具体开发时Handler返回方法值类型包括: modelAndView, String(逻辑视图名),void(通过在handler形参中添加HttpServletRequest, HttpServletResponse, 类似原始Servlet开发方式,注意: 可以通过指定response响应的结果类型实现json数据输出)

View Resolver视图解析器: 根据逻辑视图名生成真正的视图 (在springmvc中使用view对象表示)

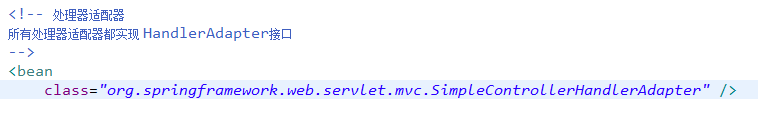
View视图: jsp页面, 仅是数据展示, 没有业务逻辑



### 2.1 配置前端控制器

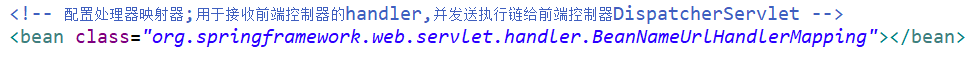


### 2.2 配置处理器适配器



### 2.3 配置处理器映射器

在classpath下的springmvc.xml中配置处理器映射器



### 2.4 配置视图解析器



## 3. 注解的处理器映射器和适配器

在spring3.1之前使用org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping注解映射器。

在spring3.1之后使用org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping注解映射器。

在spring3.1之前使用org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter注解适配器。

在spring3.1之后使用org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter注解适配器。

### 3.1 springmvc.xml配置映射器注解与适配器注解

|  |
| --- |
| <!-- 开启注解扫描 -->  <context:component-scan base-package=*"com.zsl.controller"*></context:component-scan>  <!-- 注解映射器 -->  <bean class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"*></bean>  <!-- 注解适配器 -->  <bean class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"*></bean>  <!-- 视图解析器 -->  <bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  </bean> |

#### 3.1.1 使用mvc:annotation-driven代替上边的注解

|  |
| --- |
| <!-- 开启注解扫描 -->  <context:component-scan base-package=*"com.zsl.controller"*></context:component-scan>    <!-- mvc:annotation-driven可以代替注解映射器和注解适配器配置,默认加载很多的参数绑定方法 -->  <mvc:annotation-driven></mvc:annotation-driven>    <!-- 视图解析器 -->  <bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  </bean> |

### 3.2 开发注解的Handler

#### 3.2.1 @Controller和@RequestMapping

@Controller():

标识这类是一个控制器

@RequestMapping(value=””, method=””):

实现对方法和url进行映射,一个方法对应一个url;

value: 填写url地址,相当于xml中的Handler的Bean

<bean name=”/\*\*\*\*” class=”全限定类名”>

method: 限制http请求方法,可以post和get

@Requestmapping(value=”/….”,method={RequestMethod.GET || RequestMethod.POST})

#### 3.2.2 Controller返回类型:

ModelAndView返回类型:

|  |
| --- |
| // 使用Controller标识它是一个控制器  @Controller  **public** **class** FindController {    /\*  \* 注解控制器可以直接使用注解AutoWired获得对象,不需要通过  \* ServletContext.getAttribute(ROOT\_WEB\_...).getBean("")来获取对象了  \*/  @Autowired  **private** ServliceMethodImpl smi;  // @RequestMapping实现对方法和url进行映射,一个方法对应一个url  @RequestMapping(value="/FindStudentInfo",method={RequestMehtod.GET})  **public** ModelAndView FindStudentInfo(){  List<StudentInfo> list = smi.FindStudentInfo();  ModelAndView mav = **new** ModelAndView();  mav.addObject("student", list);  mav.setViewName("/index.jsp");  **return** mav;  }  } |

void返回类型:

|  |
| --- |
| /\*  \* 映射url  \*/  @RequestMapping(value="/deleteStudentInfo", method={RequestMehtod.POST})  **public** **void** deleteStudentInfo(HttpServletRequest req, HttpServletResponse resp) **throws** IOException{  /\*resp.setContentType("text/html;charset=utf-8");\*/  String id = req.getParameter("id");  String str = req.getParameter("string");  System.***out***.println(str);  **int** num = smi.deleteStudentInfo(Integer.*parseInt*(id));  **if**(0 != num){  PrintWriter pw = resp.getWriter();  pw.write("删除成功!!!");  }  } |

也可以通过response指定response指定响应结果, 例如响应json数据如下:

resp.setCharacterEncoding(“utf-8”);

resp.setContentType(“application/json;charset=utf-8”);

resp.getWriter().write(“json串”);

String 返回类型:

如果controller方法返回string, 表示返回逻辑视图名.

真正视图(jsp路劲)=前缀 + 逻辑视图名 + 后缀

|  |
| --- |
| @Controller  **public** **class** FindController {  /\*  \* 注解控制器可以直接使用注解AutoWired获得对象,不需要通过  \* ServletContext.getAttribute(ROOT\_WEB\_...).getBean("")来获取对象了  \*/  @Autowired  **private** ServliceMethodImpl smi;    // @RequestMapping实现对方法和url进行映射,一个方法对应一个url  @RequestMapping(value="/FindStudentInfo", method={RequestMethod.***GET***})  **public** String FIndStudentInfo(Model model){  List<StudentInfo> list = smi.FindStudentInfo();  ModelAndView mav = **new** ModelAndView();  // 通过形参中的model将model数据传到页面  // 相当于ModelAndView.addObject();  model.addAttribute("student", list);  **return** "/index.jsp";  }  } |

1. redirect重定向:

浏览器地址中的url会变化, 修改提交的request数据无法传到重定向的地址.因为重定向后进行request(request无法共享)

2. forward页面转发

通过forward进行页面转发, 浏览器地址栏url不变, request可以共享

|  |
| --- |
| **public** String FIndStudentInfo(Model model){  // 重定向:  // return "redirect:index.jsp";  // 转发:  **return** "forward:index.jsp";  } |

## 4. SpringMVC参数绑定过程

### 4.1 SpringMVC参数绑定过程

从客户端请求key/value数据，经过参数绑定，将key/value数据绑定到controller方法的形参上。

springmvc中，接收页面提交的数据是通过方法形参来接收。而不是在controller类定义成员变更接收！！！！

客户端请求

key/value

处理器适配器调用springmvc提供参数绑定组件将key/value数据转成controller方法的形参

参数绑定组件：在spirngmvc早期版本使用PropertyEditor（只能将字符串传成java对象）

**后期使用converter**（进行任意类型的传换）

spirngmvc提供了很多**converter（转换器）**

**在特殊情况下需要自定义converter**

**对日期数据绑定需要自定义converter**

controller方法（形参）

### 4.2 默认支持的类型

直接在controller方法形参上定义下边类型的对象，就可以使用这些对象。在参数绑定过程中，如果遇到下边类型直接进行绑定。

#### 4.2.1 HttpServletRequest

通过request对象获取请求信息

#### 4.2.2 HttpServletResponse

通过response处理响应信息

#### 4.2.3 HttpSession

通过session对象得到session中存放的对象

#### 4.2.4 Model/ModelMap

model是一个接口，modelMap是一个接口实现 。

作用：将model数据填充到request域。

### 4.3 简单类型

通过@RequestParam对简单类型的参数进行绑定。

如果不使用@RequestParam，要求request传入参数名称和controller方法的形参名称一致，方可绑定成功。

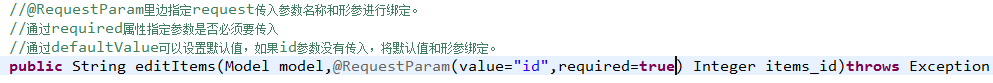
public String editItem(Model model,Integer id) throws Exception｛

｝

如果使用@RequestParam，不用限制request传入参数名称和controller方法的形参名称一致。

通过required属性指定参数是否必须要传入，如果设置为true，没有传入参数，报下边错误：





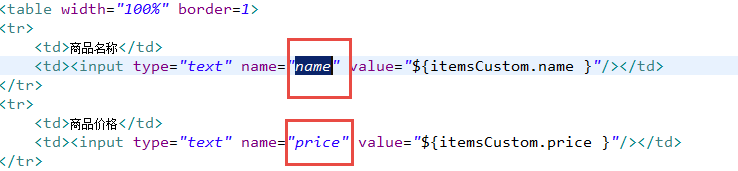
### 4.4 pojo绑定(JavaBean绑定)

页面中input的name和controller的pojo形参中的属性名称一致，将页面中数据绑定到pojo。

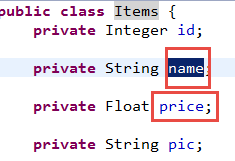
将pojo对象中的属性名于传递进来的属性名对应，如果传进来的参数名称和对象中的属性名称一致则将参数值设置在pojo对象中.

请求的参数名称和pojo的属性名称一致，会自动将请求参数赋值给pojo的属性。

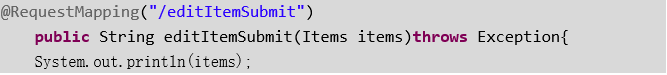
页面定义：



controller的pojo形参的定义：



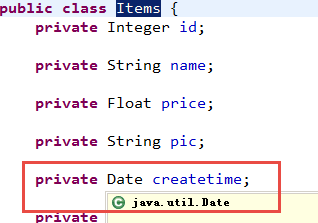
Controller方法定义如下:



### 4.5自定义参数绑定实现日期类型绑定

对于controller形参中pojo对象，如果属性中有日期类型，需要自定义参数绑定。

将请求日期数据串传成 日期类型，要转换的日期类型和pojo中日期属性的类型保持一致。



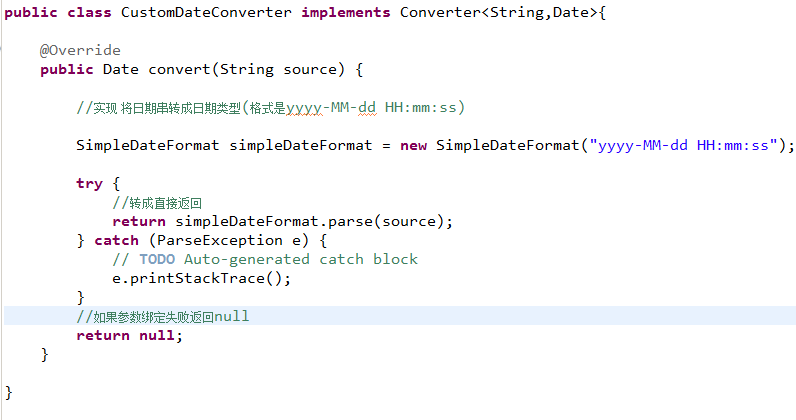
所以自定义参数绑定将日期串转成java.util.Date类型。

需要向处理器适配器中注入自定义的参数绑定组件。

#### springmvc. 配置方式:



#### 自定义日期类型绑定(编写java代码):



java类需要实现Converter接口;

重写Convert()方法;

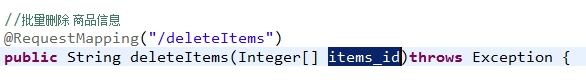
Converter(原类型, 目标类型);

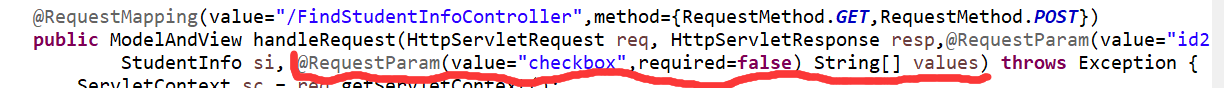
### 4.6 数组绑定

商品批量删除，用户在页面选择多个商品，批量删除。

关键：将页面选择(多选)的商品id，传到controller方法的形参，方法形参使用数组接收页面请求的多个商品id。

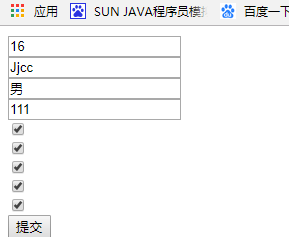
controller方法定义：





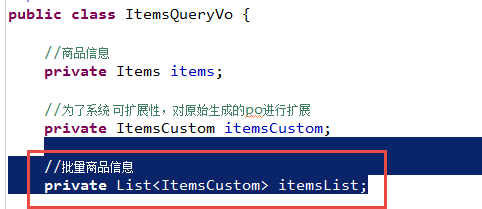
页面定义:

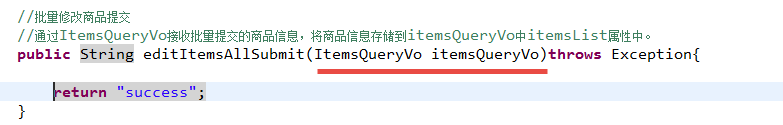




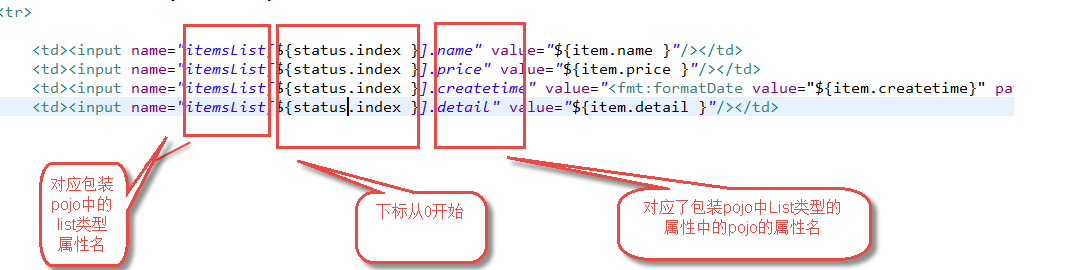
### 4.7 list集合绑定

使用List接收页面提交的批量数据，通过包装pojo接收，在包装pojo中定义list<pojo>属性





页面定义：



### 4.8 map绑定

也通过在包装pojo中定义map类型属性。

在包装类中定义Map对象，并添加get/set方法，action使用包装对象接收。

包装类中定义Map对象如下：

**Public class** QueryVo {

private Map<String, Object> itemInfo = new HashMap<String, Object>();

//get/set方法..

}

页面定义如下：

<tr>

<td>学生信息：</td>

<td>

姓名：<inputtype=*"text"*name=*"itemInfo['name']"*/>

年龄：<inputtype=*"text"*name=*"itemInfo['price']"*/>

.. .. ..

</td>

</tr>

Contrller方法定义如下：

public String useraddsubmit(Model model,QueryVo queryVo)throws Exception{

System.out.println(queryVo.getStudentinfo());

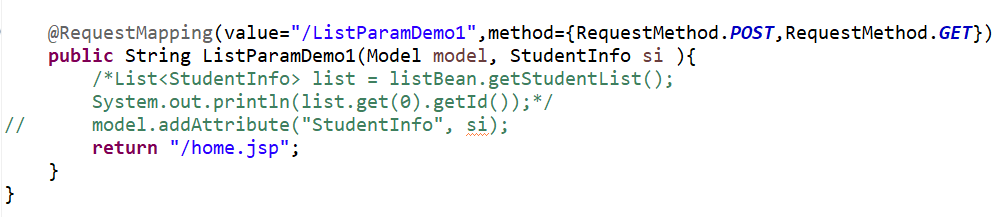
}

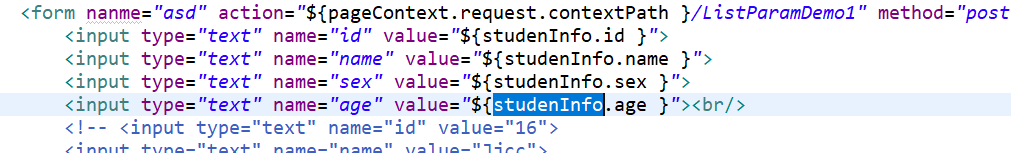
## 5. 数据回显

表单提交失败需要再回到表单页面重新填写，原来提交的数据需要重新在页面上显示。

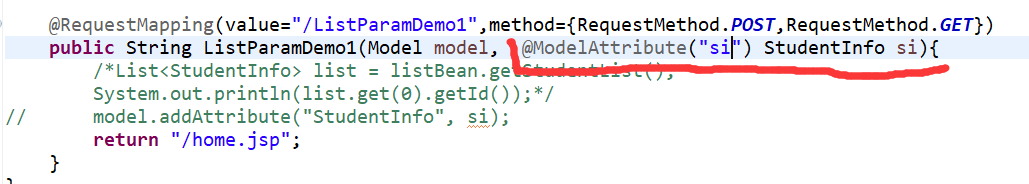
### 1、springmvc默认对pojo数据进行回显。

pojo数据传入controller方法后，springmvc自动将pojo数据放到request域，key等于pojo类型（首字母小写）

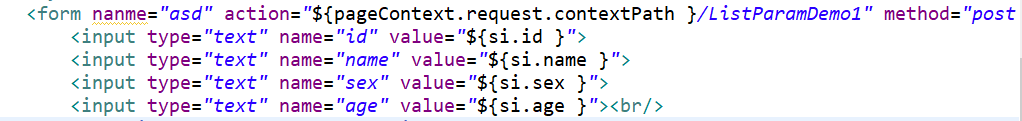




如果key不是pojo的类名(首字母小写),可以使用@ModelAttribute指定pojo回显到页面在request中的key

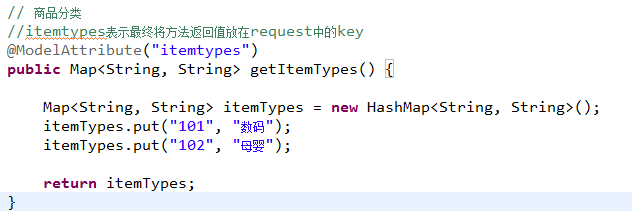


页面:

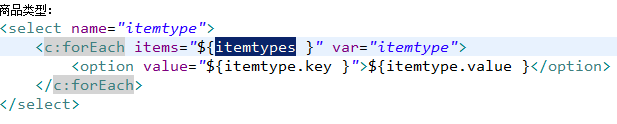


### 2、@ModelAttribute还可以将方法的返回值传到页面

使用@ModelAttribute(“....”)注解在方法上,方法不需要提供url,因为值已经保存在了request中.调用其它controller转发的链接后,值也可以获取



页面上可以得到itemTypes数据。



### 3. 使用最简单的方法就是使用Model或Request将值保存到请求域中



request.setAttribute(“”, “”);

## 6. 文件上传

### 6.1 SpringMVC中对多部件类型解析

#### 1. 上传需要的Jar包



#### 2. 在springmvc.xml中配置multipart类型解析器。

在 页面form中提交enctype="multipart/form-data"的数据时，需要springmvc对multipart类型的数据进行解析。



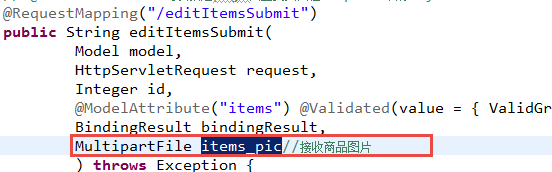
#### 3. 上传文件HTML代码

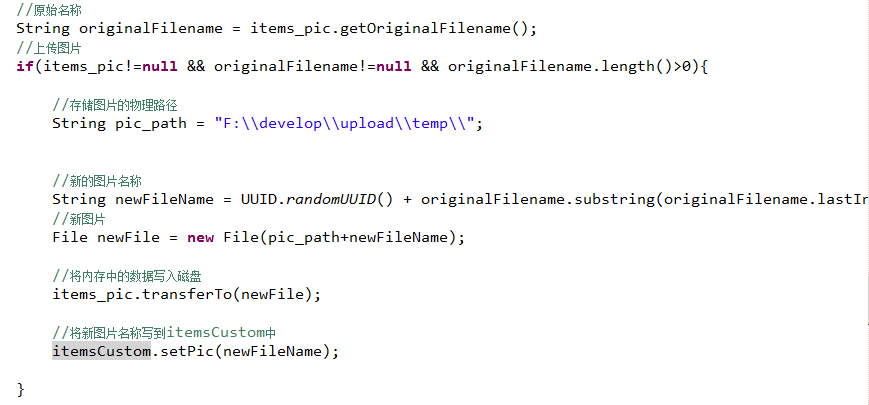
enctype=”multipart/form-data”必须添加

|  |
| --- |
| <form action=*"*${pageContext.request.contextPath }*/uploadFile"* method=*"post"* enctype=*"multipart/form-data"*>  <input type=*"file"* name=*"choose"*>  <input type=*"submit"* value=*"提交"*>  </form> |

#### 4. Controller方法

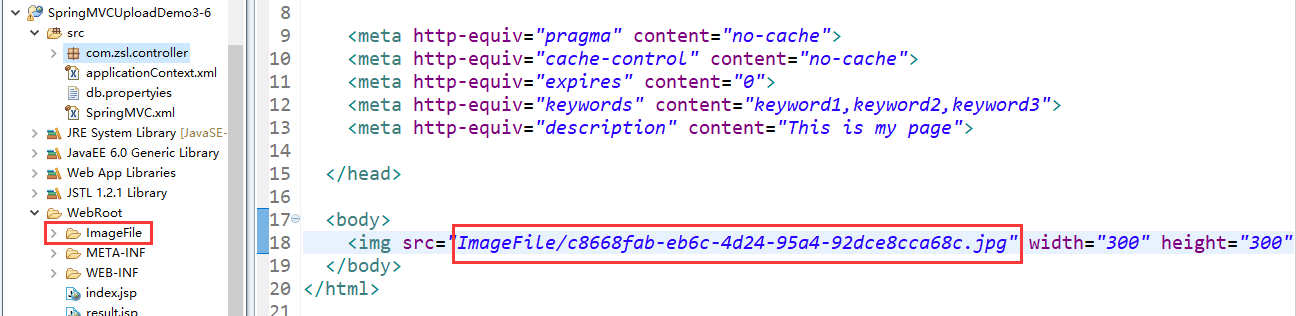
MuLtipartFile 对象是文件上传的对象



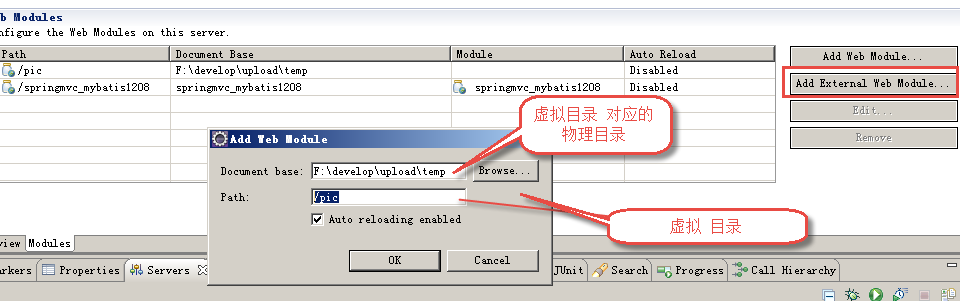


#### 注意: img的src路径填写(推荐设置虚拟目录,不设置直接src访问,会有延迟,且还需要设置workspace的刷新)

如果没有设置Tomcat的虚拟路径, jsp界面的img标签的src需要写成:



如果要设置Tomcat的虚拟路径:



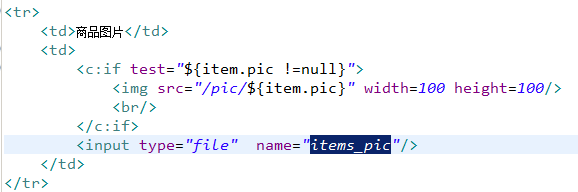
也可以直接修改tomcat的配置：

在conf/server.xml文件，添加虚拟 目录 ：

<Context path="/img" docBase="F:\SpringMVCDemo\SpringMVCUploadDemo3-6\WebRoot\Image" reloadable="true"></Context>

注意：在图片虚拟目录 中，一定将图片目录分级创建（提高i/o性能），一般我们采用按日期(年、月、日)进行分级创建。

然后img标签的src可以写成:



## 7. RESTful

RESTful架构，就是目前最流行的一种互联网软件架构。它结构清晰、符合标准、易于理解、扩展方便，所以正得到越来越多网站的采用。

RESTful（即Representational State Transfer的缩写）其实是一个开发理念，是对http的很好的诠释。

1、对url进行规范，写RESTful格式的url

非REST的url：http://...../queryItems.action?id=001&type=T01

REST的url风格：http://..../items/001

特点：url简洁，将参数通过url传到服务端

2、http的方法规范

不管是删除、添加、更新。。使用url是一致的，如果进行删除，需要设置http的方法为delete，同理添加。。。

后台controller方法：判断http方法，如果是delete执行删除，如果是post执行添加。

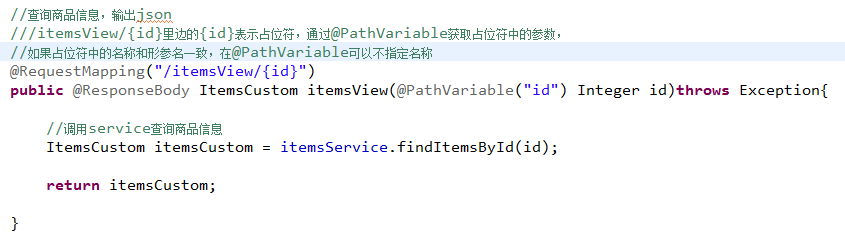
3、对http的contentType规范

请求时指定contentType，要json数据，设置成json格式的type。

### 7.1 编写Controller

定义方法，进行url映射使用REST风格的url，将查询商品信息的id传入controller .

输出json使用@ResponseBody将java对象输出json。



@RequestMapping(value="/ itemsView/{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。

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

### 7.2 RESTful方法的前端控制器

在web.xml配置：



### 7.3 对静态资源的解析

配置前端控制器的url-parttern中指定/，对静态资源的解析出现问题：



在springmvc.xml中添加静态资源解析方法。

