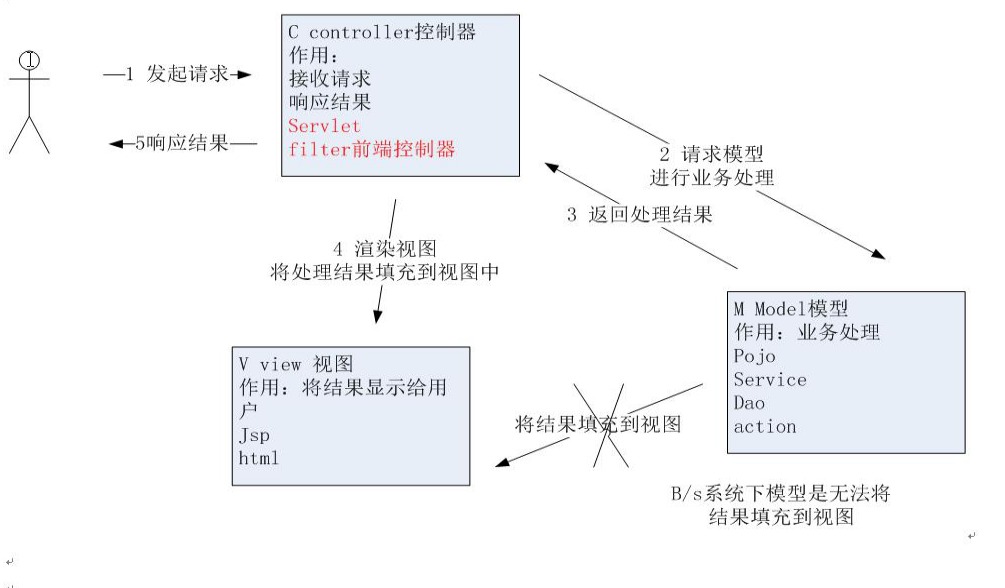
# SpringMVC

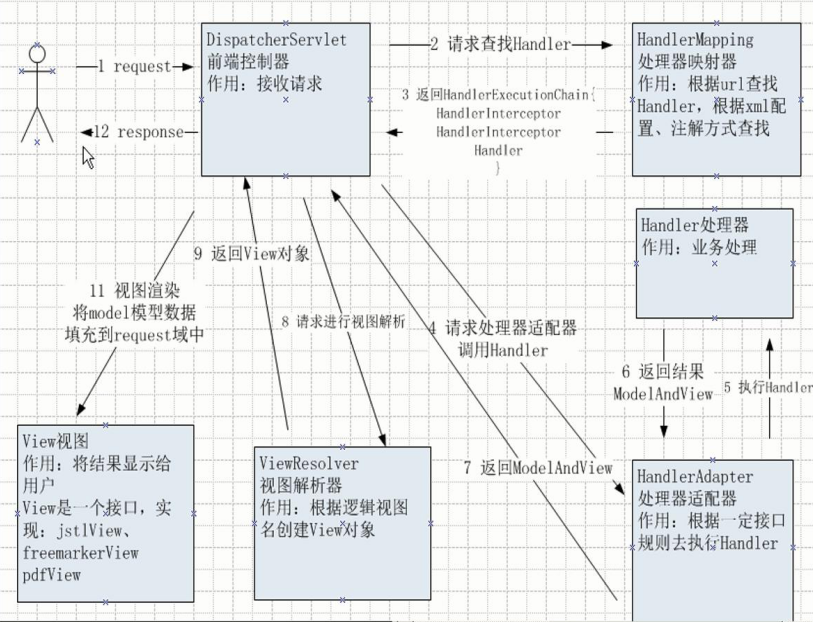
## 架构原理

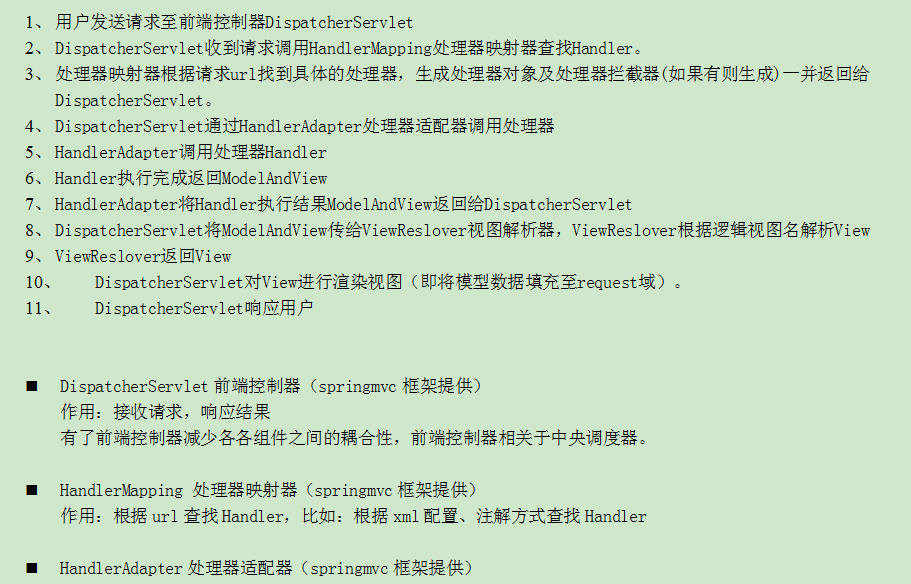
springmvc是spring的一部分

b/s架构



springmvc





## 入门

### 环境搭建

### 前端控制器配置



### 配置处理器适配器



### 开发Handler（自己写代码）

创建类实现Controller接口

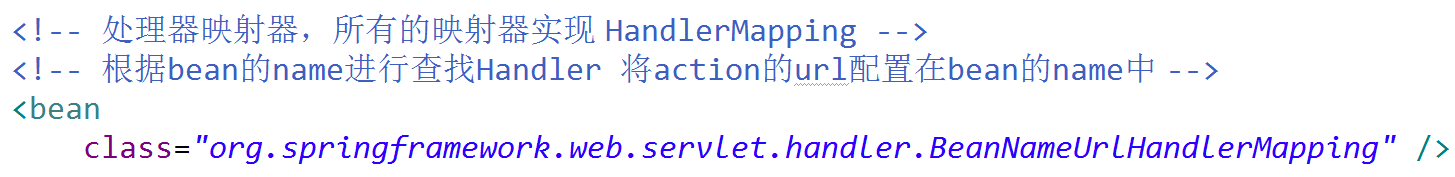




### 将Handler配置到spring



### 配置处理器映射器





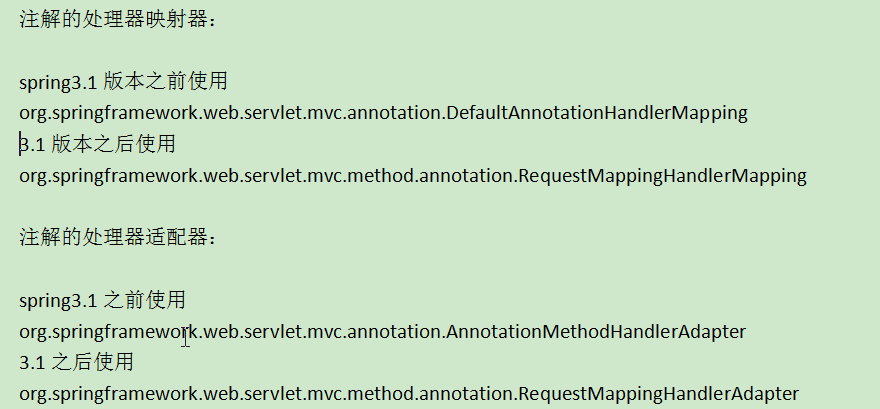


### 视图解析器

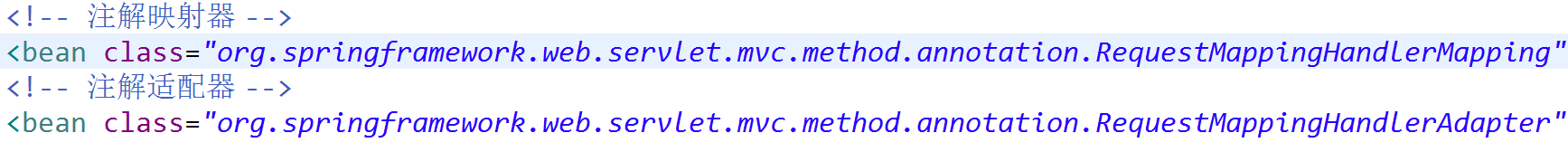


### 注解开发

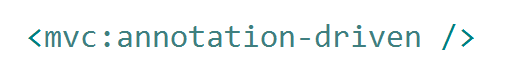
#### 注解处理器映射器和适配器



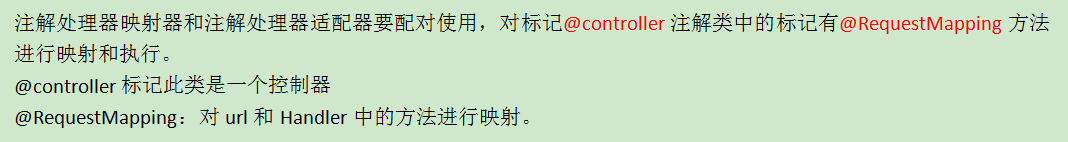
#### 配置映射器和适配器



上边两个配置可以用下面代替



#### 开发Handler





#### 在spring配置Handler



## springmvc整合mybatis

### 整合dao

实现spring、mybatis整合

1. 创建web项目
2. 加入jar包
3. 配置......

## 参数绑定

### 什么是参数绑定

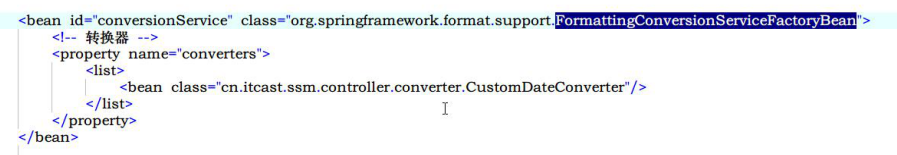
springmvc接收请求的key/value串（比如：id=2&type=101），经过类型转换，将转换后的值赋值给controller方法的形参，这个过程就叫参数绑定。

### 基本类型参数绑定

### pojo类型参数绑定

### 自定义类型转换器

默认提供的是年月日转换，如果需要时分秒就要自定义

1. 需要编写一个类让它实现Converter<S,T>接口，S表示输入类型，T表示转换后的类型
2. 实现convert方法，里面写具体转换逻辑
3. 配置该转换器  
   



### 包装类型的pojo绑定

包装类型pojo：属性也是一个pojo

绑定规则：页面 itemsCustom.name 形参 ItemsQueryVo

### 数组参数绑定

页面：name=”delete\_id” value=”${item.id}”

形参：Integer[] delete\_id

### List<pojo>参数绑定

页面：name=”itemsList[${status.index}].name”

形参：ItemsQueryVo itemsQueryVo

类中：ItemsQueryVo {

private List<ItemsList> itemsList;

}

## RequestMapping

### URL路径映射

对controller方法进行映射

### 窄化请求映射

在类上加注解，类中的方法都会加上类上的路径，方便管理

### 请求方法限定

限制http的请求方法，提高安全性

@RequestMapping(value=”/itemsList”, method={RequestMethod.POST})

## controller返回值

### 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。

### void



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");

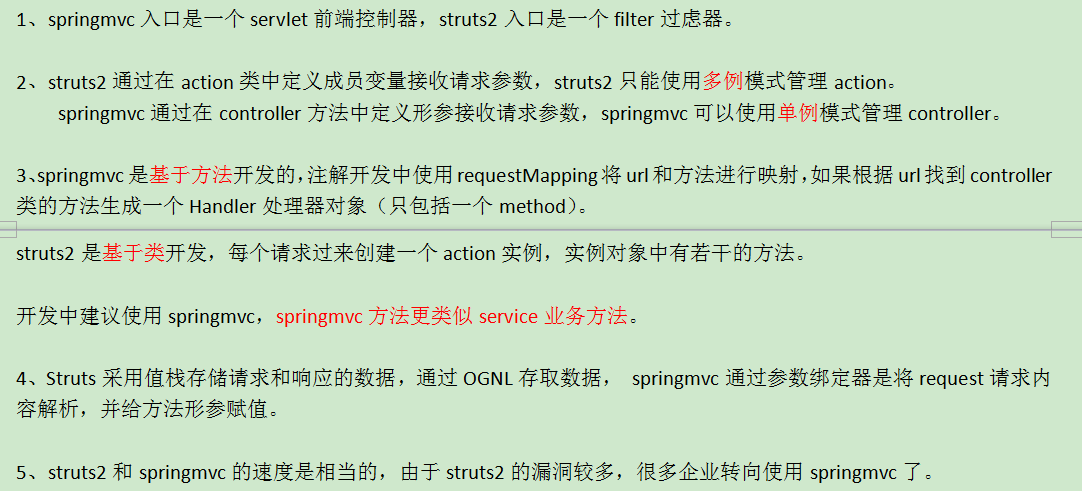
response.setContentType("application/json;charset=utf-8");

response.getWriter().write("json串");

### String

* 页面转发方式  
  格式：forword:转发地址  
  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。
* 页面重定向方式  
  格式：redirect:重定向地址  
  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。
* 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的区别



## validation校验

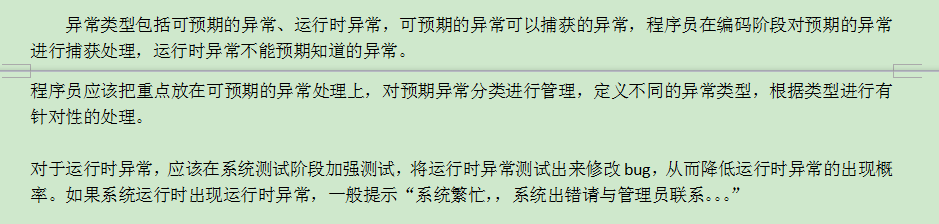


## 数据回显

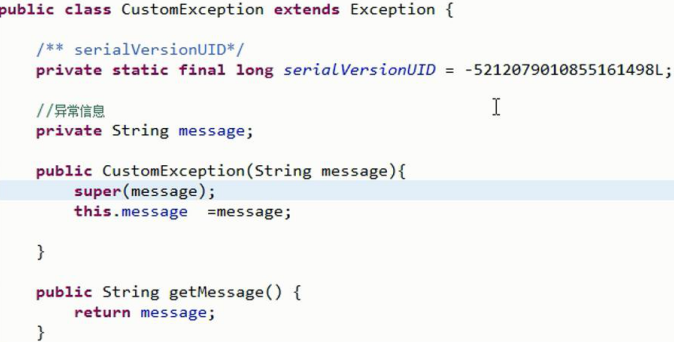
页面校验错误，页面应该把刚才提交的数据显示到页面

## 异常处理

### 业务系统中异常处理方法



### 自定义异常



### 异常处理器

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");

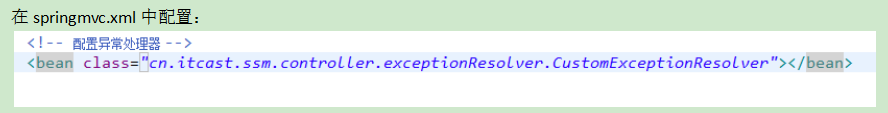
modelAndView.addObject("message", message);

return modelAndView;

}

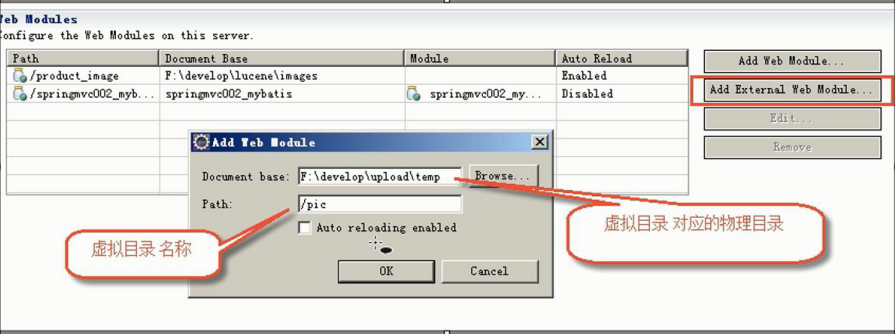
}

### 在springmvc中配置异常处理器



## 图片上传

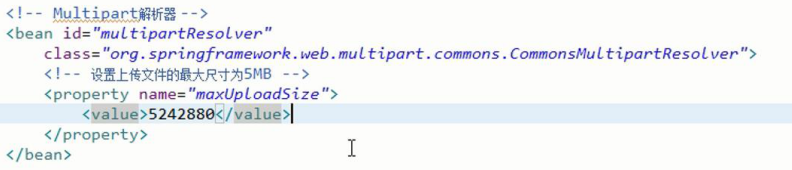
一般需要上传到单独的服务器，这里模拟配置一个虚拟目录



也可在conf/server.xml配置

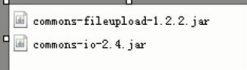


### 配置文件上传解析器



id名称是固定的

### 加入依赖jar包



### 实现上传

1. 形参加入MultipartFile picture
2. 代码  
   

## Json交互

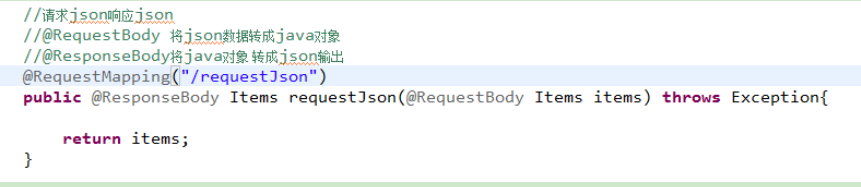
json数据简单，更符合面向对象，前端js解析也方便

### 加入jackson依赖

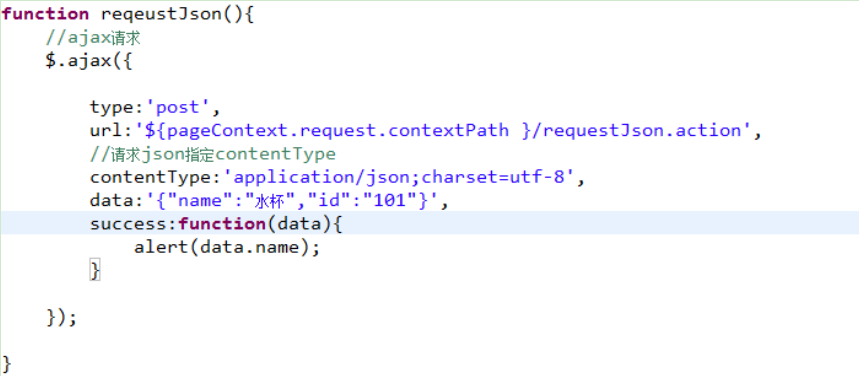


### 请求Json响应Json

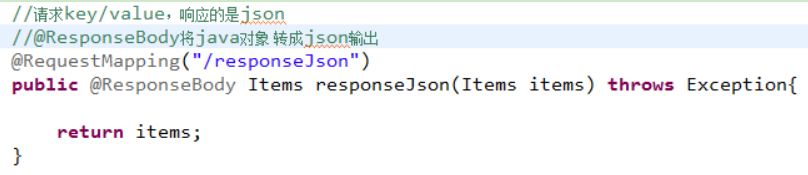


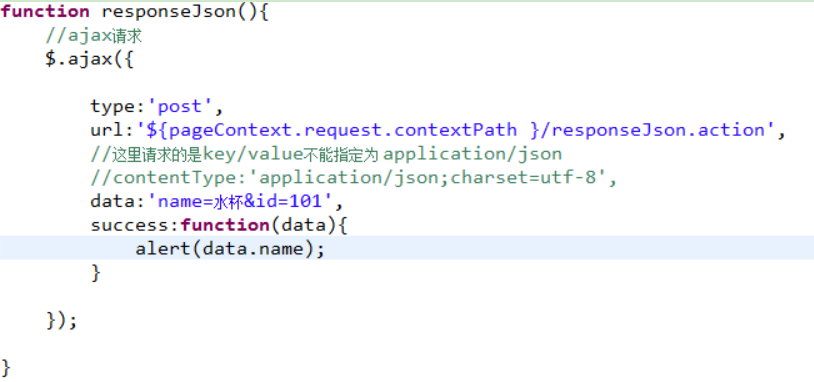


页面：



### 请求key/value响应Json





## REST的支持

## 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，可以在Handler执行之前、之后、之中添加代码，这种方式就是面向切面编程

拦截器相当于一个过滤器

### 自定义拦截器

1. 定义类实现HandlerInterceptor

实现三个方法



1. 配置拦截规则  
   