# 更新

2018年7月22日更新了json的使用和restful风格和拦截器

2018年7月21日更新了全局处理器、文件上传操作

2018年7月18日 创建此文本，添加了入门步骤：注解方式和xml方式，Controller（handler）类的写法，自定义参数绑定操作，MVC配置文件操作，其它中的工作原理，默认加载的组件的详细信息

# 前提

学习springMVC需掌握

**自定义绑定参数：** 虽然配置了3.1以后的适配器和映射器做了很多为很多类型自动转换绑定参数，但是由于日期数据有很多种格式，springmvc没办法把字符串转换成日期类型。所以需要自定义参数绑定。

自定义异常处理器：出现异常友好展示用户，实现HandlerExceptionResolver

json数据交互：@ResponseBody @RequestBody

自定义拦截器：拦截conroller类：进入方法前，方法中代码执行中，方法执行完

RESTful（了解）就是一个资源定位及资源操作的风格。不是标准也不是协议，只是一种风格

# 入门步骤

## 注解方式

第一步 导入jar包

注意：经测试后：还需要spring-aop才行

commons-logging-1.2.jar、log4j-1.2.17.jar、4.3.13版本的aop、beans、context、core、expression、tx、web、webmvc

第二步 web.xml中配置前端控制器DispatcherServlet

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

id=*"WebApp\_ID"* version=*"2.5"*>

<display-name>springmvc-first</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

<!-- 配置SpringMVC前端控制器 它是servlet-->

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param> 寻找核心配置文件的配置

<param-name>contextConfigLocation</param-name>

<param-value>classpath:springmvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup> 让此前端控制器它尽快加载

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<url-pattern>/</url-pattern> 默认配置方式

</servlet-mapping>

<url-pattern>的配置方式

第一种方式：/: 除了jsp，所有的请求的都由前端控制器拦截，所有静态文件需要配置不让前端控制器控制，这种方式配置的好处可以实现RESTFUL风格的url 核心文件记住需要配置不拦截静态资源文件

第二种方式：\*.action：只拦截请求以.action结尾的

第三种方式：/\*: 使用这个配置会连jsp都会由前端控制器解析jsp，不能根据jsp页面找到handler（自己写的类），会报错

第三步 创建核心配置： springmvc.xml

只配置一个Controller（handler）扫描就可以了，让Spring对自己写的类进行管理。其它的因为：有默认的加载

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd"*>

<!-- 配置controller\service\Repository\Component扫描包 -->

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

</beans>

第四步 创建Controller（handler）类：A

需要在类上添加@Controller注解，把Controller交由Spring管理

在方法上面添加@RequestMapping注解，里面指定请求的url。其中“.action”可以加也可以不加。

@Controller // 第一步 类上注解必须是controller

@RequestMapping(value="/wwl") 设置本类url的根路径

**public** **class** A { //第二步设置映射路径，就是url要访问本方法

@RequestMapping(value="/a",method= {RequestMethod.POST,RequestMethod.GET})

public String handleRequest(Model model){ //第三步 设置参数model（底层就是map）

List<Item> list = new ArrayList<Item>();

list.add(new Item(1, "1华为 荣耀6", "质量好"));

list.add(new Item(2,"2华为 荣耀7", "质量好！"));

list.add(new Item(3, "3华为 荣耀8","质量好！"));

model.addAttribute("c",list); //第四步 使用model把参数放入

return "redirecd:a.action"; //forward, //第五步 可以重定向到jsp或者conroller方法中

}

这样简单配置完就可以实现网页跳转并传送数据到jsp页面（springmvc默认加载了一些配置文件）

默认加载配件 ：spring-webmvc包下：org\springframework\web\servlet\DispatcherServlet.properties

## XML方式

第一步 导入jar包

第二步 web.xml中配置前端控制器，（前两步都一样的）

第三步 创建核心配置： springmvc.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd">

<!-- 第一 先处理器映射器（默认加载了老的版本）：请求的url进入处理器映射器，都实现了HandlerMapping接口类 -->

<!-- 处理器映射器都会获得url最后的url路径去寻找相对应的bean名称-->

<bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"></bean>

<!--第二 然后配置Controller类：自己写的类实现Controller接口类-->

<!—在第一步匹配到了‘/add.action’：映射进入配置的类 -->

<bean name="/add.action" class="com.wwl.controller.UserController"></bean>

<!--第三 处理器适配器 判断此类是不是controller类-->

<!-- 所有的处理器适配器都是实现HandlerAdapter接口：来判断它是否要处理器适配器处理：

HandlerAdapter接口有一个方法：boolean supports(Object handler);实现类中：用instanceof判断类型

最后结论：类型匹配，才能由SimpleControllerHandlerAdapter适配器执行-->

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

<!--第四 视图解析器 解析jsp -->

<!-- 打开源码可以看到 ：默认使用jstl标签解析jsp，所以classpath下得有jstl得包 -->

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

总结 ：其实XMl的处理器映射器、处理器适配器、视图解析器都是默认就加载好的（老版本），你只需要配置：第二步的 controller类即可，老师还介绍了另外一种XML的映射器和适配器。

第四步 创建Controller类

实现controller接口让适配器判断此类属于controller类

//实现接口：Controller接口类

public class UserController implements Controller{

public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception {

List<Item> list = new ArrayList<Item>();

list.add(new Item(1, "1华为 荣耀6", "质量好"));

list.add(new Item(2,"2华为 荣耀7", "质量好！"));

ModelAndView n= new ModelAndView();

n.addObject("c",list);

n.setViewName("/jsp/a.jsp");

return n;

}

# Controller（handler）类

controller涉及到的配置情况

@Controller //需要配置：<context:component-scan base-package="com.wwl" />

@RequestMapping //需注解适配器和映射器（不配置也可以：默认已加载老版）配置新的：<mvc:annotation-driven/>

return "jsp"; //可以不配置（写全路径即可），配置InternalResourceViewResolver 视图解析器：（路径）和（逻辑视图名的后缀）

@Controller 第一

public class AController {

@RequestMapping("test") 第二

public String a(Model model){

model.addAttribute("key", "value");

return "jsp"; 第三

第一步 配置URL路径进入方法 @RequestMapping("test")

@Controller 第一步 注解，核心配置文件需配置 <context:component-scan>

public class AController {

@RequestMapping("test") 第二步 可不配置，默认加载2.5老板注解适配和映射，配置新的：<mvc:annotation-driven/>

public String a(Model model) {

return "a.jsp"; 第三步 没配置，就以工程上下文的全路径找资源视图，默认都是转发，重定向：”redirect:路径”

redirect:/itemEdit.action?itemId=" + item.getId() 换成重定向可以在后缀加上参数返回

}

最后url中显示路径是：localhost:8080/工程名/test

@RequestMapping()有六个属性

value：指定请求的实际地址，指定的地址可以是URI Template 模式

method：指定请求的method类型， GET、POST等；method={RequestMethod.GET, RequestMethod.POST }

consumes：指定处理请求的提交内容类型（Content-Type），例如application/json, text/html;

produces: 指定返回的内容类型，仅当request请求头中的(Accept)类型中包含该指定类型才返回

params： 指定request中必须包含某些参数值是，才让该方法处理。

headers： 指定request中必须包含某些指定的header值，才能让该方法处理请求

第二步 获取页面传递过来的数据

1、在controller类的方法形参上的名字和页面input表单的name一致即可自动绑定获取数据

推荐使用包装数据类型，因为基础数据类型不可以为null

public String a(String userName) { //页面表单input标签name="userName"这样就可以得到此值

return "login";

}

}

**使用 @RequestParam好处**：①required设置为true：必须有值②defaultValue可以设置默认值

public String a(@RequestParam(value="userName",required=true,defaultValue="呵呵") String a) { //在需要绑定参数的前面加注解

System.out.println(a); //因为设置了默认值即便没有值，默认值就是 呵呵

return "login";

}

entity（实体类）的接收：实体类中的属性名要和表单中的name一致

如遇到实体类中又维护着另一个实体类：标签中的name写法：User.name

public String a(Users user) { //页面表单input标签name要和user对象中的属性名都一致

String name=user.getName

return "login";

}

}

注意：

提交的表单中不要有日期类型的数据，否则会报400错误。如果想提交日期类型的数据需要用到自定义参数绑定。

2、形参中添加如下类型的参数处理适配器会默认识别并进行赋值

1）HttpServletRequest 2）HttpServletResponse 3）HttpSession

public String a(HttpServletRequest request,HttpServletResponse requests) throws ServletException, IOException {

String username=request.getParameter("userName");

request.getRequestDispatcher("a.jsp").forward(request, requests);

return "login";

}

第三步 把方法中的数据返回页面，页面通过el表达式进行获取

第一种方式：方法中的返回值为String操作

第一步 核心配置文件SpringMVC.xml需要配置视图解析器：

要优先解析jsp，找不到再解析html的效果：https://blog.csdn.net/wwzuizz/article/details/78268007

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

<property name="order" value="44" /> //优先级

<property name="contentType" value="text/html"/>//向浏览器打印内容格式

<property name="prefix" value="/WEB-INF/"/> //工程中的真实位置

<property name="suffix" value=".jsp"/> //逻辑视图名的后缀

</bean>

第二步 在形参中添加接口：Model，使用方法model.addAttribute("key", "value");

@Controller

public class AController {

@RequestMapping("test")

public String a(Model model) {

model.addAttribute("key", "value"); //相当于request.setAttribute方法

return "a"; //在 第一步配置了逻辑视图名：通过http://localhost:8080/springMVC/test访问就会跳转到a.jsp

“redirect:a.jsp” 可以进行重定向，默认什么也不写都是转发

注意 “redirect:a.jsp”:需要全路径，还有WEB—INF下的是访问不到的，因为WEB—INF目录是受保护的，只能转发访问

第二种方式：返回值为ModelAndView的操作

requestMapping("test2")

public ModelAndView b() {

ModelAndView n = new ModelAndView();

n.addObject("key","你好"); //相当于request.setAttribute

n.setViewName("/jsp/a.jsp"); //要跳转的页面

return n;

第三种方式：返回值为void 的操作：这个就是原始的方式了形参放入httpsetvletRequest即可

# 自定义参数绑定

虽然配置了3.1以后的适配器和映射器做了很多为很多类型自动转换绑定参数

由于日期数据有很多种格式，springmvc没办法把字符串转换成日期类型。所以需要自定义参数绑定。

步骤有两步，两步配置完后，形参中有相同类型就会自动去匹配转换器进行转换

第一步 自定义类实现Converter接口

//泛型：key：需要转换的源的类型，值：需要转换的目标类型

public class DateConverter implements Converter<String, Date>{

public Date convert(String resource) { //实现此方法，字符串：yyyy-MM-dd HH:mm:ss从这里传进

try { //进行字符串转换时间类操作

SimpleDateFormat sdf= new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

return sdf.parse(resource); //把转换的时间值返回

} catch (ParseException e) {

e.printStackTrace();

}

return null;

}

第二步 springMVC核心配置文件进行配置

<mvc:annotation-driven conversion-service="c"/> 注解射配器和映射器配置的标签中加入属性指定转换器类

<bean id="c" class="org.springframework.format.support.FormattingConversionServiceFactoryBean"> spring的转换器类

<property name="converters">

<list>

<bean class="com.wwl.converter.DateConverter"/> //这里配置第一步的类的全路径

</list>

</property>

</bean>

# 自定义异常处理器

最简单的异常配置

自定义一个类实现HandlerExceptionResolver接口

@Component 第一步：把此类给spring管理

public class CustomerException implements HandlerExceptionResolver{

第二步 实现接口中的方法

public ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object handler,

Exception ex) {

ModelAndView mav = new ModelAndView();

mav.setViewName("msg");

mav.addObject("msg","系统发生异常！请联系管理员");

return mav;//第三步 跳转页面至msg.jsp页面，用el标签进行打印消息

}

这三步就配置好全局异常：一旦有任何异常都会转发页面至msg.jsp 友好提示用户

结合之前学的自定义异常

自定义异常类继承Exception类，不用给spring管理，

测试使用 throw new MyException("这是我自定义的异常") ：因为是new异常对象的操作

public class MyException extends Exception {

private String msg;

public MyException(String msg) {this.msg = msg; } //构造方法

public String getMsg() {return msg; } //get方法

}

上面的CustomerException类中的方法就可以这样写

public ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object handler,

Exception ex) {

String e="系统发生异常！请联系管理员"; //全局信息

if(ex instanceof MyException) {

e=((MyException) ex).getMsg(); //自己定义的异常信息

}

ModelAndView mav = new ModelAndView();

mav.setViewName("msg");

mav.addObject("msg",e);

return mav;//跳转页面至msg.jsp页面，用el标签进行打印消息

}

# JSON使用

第一步 加入jar包 注意版本问题，spring4.0的需要jackson-core-2.7.0.jar

jackson-annotations-2.4.0.jar、jackson-core-2.4.2.jar、jackson-databind-2.4.2.jar

第二步 conroller类的编写

json数据响应给浏览器

1、实体类：@ResponseBody 把实体类pojo转换为json串响应给浏览器

@RequestMapping("jsonTest")

@ResponseBody 第一步

public Users u() {

user.setAge(5); user.setId("33"); user.setName("你好");

return user; 返回值直接返回实体类直接打印给浏览器：效果和servlet一样，浏览器直接打印：json格式的内容

}

浏览器显示页面：{id "33"name "你好" age 5}

2、Map：@ResponseBody把map封装的数据转换为json串响应给浏览器

@RequestMapping("/area")

@ResponseBody

private Map<String, Object> queryArea() {

Map<String, Object> map= new HashMap<String, Object>(); //创建map<String,Object>

List<Area> list = areaService.query();

map.put("total",list.size()); //存储总数

map.put("key", list); //存储list的实体类

return map;

}

浏览器显示：{"total":4,"key":[{"areaId":3,"areaName":"东苑","areaDesc":"东苑描述","priority":12},{"areaId":4,"areaName":"南苑","areaDesc":"南苑描述","priority":10},{"areaId":5,"areaName":"西苑","areaDesc":"西苑描述","priority":9}]}

浏览器json数据请求服务器

@RequestBody 页面传来的数据是json格式进行转换为实体类属性中

public void u(@RequestBody Users user) {

System.out.println(user);//打印：{Users ：[{id "33"name "你好" age 5}]}

}

两个注解可以配合使用，把传来的页面数据json串封装到实体类属性中，修改完后，又打印回浏览器

@ResponseBody

public User u(@RequestBody Users user) { //第一步 把json数据封装到user中

System.out.println(user);//类中的属性值： id "33",name "你好" ,age 5

user.setName(“我不好”); //第二步 修改name的值，又转换为json格式响应回浏览器

return user;

}

# RESTFUL风格

第一步 web中配置前端控制器DispatcherServlet必须为 ”/”

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:springmvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<url-pattern>/</url-pattern> //必须为: /

</servlet-mapping>

</web-app>

第二步 controller类中使用注解接收参数@PathVariable

@RequestMapping("/ert/{id}/{name}") //第一步 需要这里设置参数名称：占位符{}，

@ResponseBody

public void userQuery(@PathVariable("I d") Integer ids,@PathVariable("name") String nae) { 第二步 使用@PathVariable注解

System.out.println("ID："+ids);

System.out.println("name:"+nae);

}

第三步 页面访问此方法url：ip地址+端口号+工程名+地址+参数

http://localhost:8080/springmvc/ert/1/你好

# 拦截器

第一步 自定义一个类实现HandlerInterceptor类，它有三个方法

public class HandlerInterceptor1 implements HandlerInterceptor {

//springmvc拦截器只针对HandlerMapping进行拦截

第一 进入handler(Controller)的方法之前执行:用于身份认证、身份授权

public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)

throws Exception {

System.out.println("1");

return true; //false：表示拦截，不向下执行

}

第二 进入Handler方法之后，返回modelAndView之前执行：（菜单导航）将公用的模型数据传入视图

public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,

ModelAndView modelAndView) throws Exception {

System.out.println("2");

}

第三 执行完handler方法后执行此方法：统一的异常处理，统一的日志处理

public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex)

throws Exception {

System.out.println("3");

}

}

第二步 核心配置文件进行配置 <mvc:interceptors>

<!-- 拦截器配置 -->

<mvc:interceptors>

<!-- 可以配置多个自定义拦截器，他们是顺序执行的-->

<mvc:interceptor>

<mvc:mapping path="/\*\*"/> <!-- /\*\*表示拦截所有的url包括子类以后的 -->

<bean class="com.wwl.interceptor.HandlerInterceptor1"/><!--自定义类实现了HandlerInterceptor接口 -->

</mvc:interceptor>

<mvc:interceptor>

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

<bean class="com.wwl.interceptor.HandlerInterceptor2"/>

</mvc:interceptor>

</mvc:interceptors>

第三步 测试

conroller类

@RequestMapping("/ert/{id}/{name}")

@ResponseBody

public void userQuery(@PathVariable("id") Integer ids,@PathVariable("name") String nae) {

System.out.println("ID："+ids);

System.out.println("name:"+nae);

}

测试 ：两个拦截器都放行

http://localhost:8080/springmvc/ert/1/你好；测试进入方法到结束方法返回浏览器打印情况：

打印情况：进方法前1，进方法前11，ID：1，name:你好，中22，中2，执行完33，执行完3

测试：第一个拦截器放行，第二个不放行

打印情况：进方法前1，进方法前11，执行完3

执行代码原理

public void method() { System.out.println("最后要进入的方法"); } //要访问的方法

public static void main(String[] args) {

boolean flag =true; //可以说成它就是拦截器1

boolean flag2 =true; //可以说成它就是拦截器2

UserController a = new UserController();

if(flag) {

System.out.println("进入方法前拦截1");

if(flag2) {

System.out.println("进入方法前拦截2");

a.method(); //最后进入的方法

System.out.println("进入方法后拦截22");

}

System.out.println("进入方法后拦截2");}}

总结

日志处理：要把日志处理放在第一个拦截器中的第三个方法，而且拦截器必须放行

登陆认证、权限认证都放在第一个拦截器的第一个方法

# 文件上传

第一步 还需要两个配件包：commons-fileupload-1.3.2和commons-io-2.2.jar

<!-- 文件上传需要的配件:maven配置上这个即可两个上传文件的包就有了 -->

<dependency>

<groupId>commons-fileupload</groupId>

<artifactId>commons-fileupload</artifactId>

<version>1.3.2</version>

</dependency>

第二步 核心配置文件需配置web.jar下的CommonsMultipartResolver的bean

注意：此 id名字必须为：multipartResolver，不可以改别的名字

<!-- id名称必须是：multipartResolver，此类在 web的jar包中 -->

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

<property name="defaultEncoding" value="utf-8"></property>

<!-- 最大上传文件大小8m -->

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

<property name="maxInMemorySize" value="10960"></property>

</bean>

第三步 浏览器页面的写法

<form action="${pageContext.request.contextPath}/upload" method="post" enctype="multipart/form-data">//第一

<input type="text" name="name">

<input type="file" name="m"> 第二

<input type="submit" value="提交">

</form>

第四步 conroller类的编写

第一种：需要MultipartFile类直接在请求中写名称要和页面上传文件的name一致

conroller类编写：注意：jsp的 <input type="file" name="m">这里名称’m’要和MultipartFile定义的对象名称一致

public class UploadTest {

@RequestMapping("upload") //这里的m需要和页面中的file中的name一致

public String upload(Model model,String name,MultipartFile m,HttpServletRequest request) throws Exception{

String newName=UUID.randomUUID().toString();//图片新名字

String oldName=m.getOriginalFilename(); //1、获取图片原来的名字

String sux= oldName.substring(oldName.lastIndexOf(".")); //获取后缀

File file2= new File(request.getServletContext().getRealPath("img")+"\\"+newName+sux); //2、新名字再加后缀

m.transferTo(file2); //3、此方法写入本地磁盘

model.addAttribute("newName", newName+sux); //把名字传入

return "pig"; //转发到pig.jsp页面：显示图片

}

第二种：这是2019年1月11日看了o2o工程的写法

//第一 设置参数的类CommonsMultipartResolver：如上传文件限制大小，(这里创建此类用来第二步的判断)

CommonsMultipartResolver multipartResolver = new CommonsMultipartResolver(request.getSession().getServletContext());

//第二 用第一步获取的类来判断有无：multipart/form-data，若没有直接跳过

if (multipartResolver.isMultipart(request)) {

//第三 如果是附带上传文件的请求直接把request强制转换为MultipartHttpServletRequest

MultipartHttpServletRequest multipartRequest = (MultipartHttpServletRequest) request;

//第四 最后获取到上传文件中的<input type="file" name="m">强制转换为CommonsMultipartFile

//注意CommonsMultipartFile类:可以获取文件名字、IO流等

CommonsMultipartFile shopImg = (CommonsMultipartFile) multipartRequest.getFile("m");

shopImg.getOriginalFilename();

}

第五步 显示图片：保存文件在文件夹下（可以把此文件夹配置为另一个服务器）

图片如果是保存在同一服务器中 ：

<body><img src="${pageContext.request.contextPath}/img/${requestScope.newName}"></body>

需要注意的问题：springMvc的静态文件默认都是拦截的，核心文件配置：<mvc:resources location="/img" mapping="/\*\*"/>

图片是保存在另一个服务器中：（tomcat可以配置很多服务器，同时开启）： 拼接url形式完成

图片操作：<img src="http://localhost:8080/tupian/01%20(8).jpg"> 这里就不用配置静态文件的问题了，进的是另一个服务器

配置两个服务器同时开启第一种：找到tomcat下的server.xml的和工程同一个host标签内加入：

<Host appBase="webapps" autoDeploy="true" name="localhost" unpackWARs="true">

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs" pattern="%h %l %u %t &quot;%r&quot; %s %b" prefix="localhost\_access\_log." suffix=".txt"/>

<Context docBase="springMVC" path="/springMVC" reloadable="true" source="org.eclipse.jst.jee.server:springMVC"/>

<Context docBase="C:\Users\wenwenliang\Pictures\壁纸" path="/tupian" reloadable="true"/></Host>

第二种 下面的配置等同于在上面配置的标签内容：

eclipse下双击打开tomcat配置页面，左下角选择Modules—》add external Web Module,选择真实文件夹和url要访问的路径

# MVC配置文件

1、配置注册扫描bean

<!-- 配置controller\service\Repository\Component扫描包 -->

<context:component-scan base-package="com.wwl" />

2、注解映射器和适配器：不配置默认加载spring2.5的两个，下面介绍3.1版本

使用<mvc:annotation-driven />标签其实就是配置了3.1以后新版本注解映射器和注解适配器的配置，此标签还加载了很多的参数绑定方法：比如json转换解析器就默认加载了

<!-- 注解驱动 -->

<mvc:annotation-driven />//等同于配置下面的两个3.1以后的注解映射器和适配器

配置处理器映射器：RequestMappingHandlerMapping

注解式处理器映射器：对类中标记了@ResquestMapping的方法进行映射。

根据@ResquestMapping定义的url匹配@ResquestMapping标记的方法，匹配成功返回HandlerMethod对象给前端控制器。

HandlerMethod对象中封装url对应的方法Method。

从spring3.1版本开始：不再建议使用DefaultAnnotationHandlerMapping（它是默认加载的）

在springmvc.xml配置文件中配置如下：

<!-- 配置处理器映射器 -->

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

配置处理器适配器 RequestMappingHandlerAdapter

配置注解式处理器适配器，对标记@ResquestMapping的方法进行适配。

从spring3.1版本开始，不再建议AnnotationMethodHandlerAdapter的使用（默认加载），

在springmvc.xml配置文件中配置如下：

<!-- 配置处理器适配器 -->

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

3、视图解析器：InternalResourceViewResolver

视图解析器他也是默认就加载的，但可以自己再次实例化注册beng提供优化：前缀和后缀，这个视图解析器支持JSP视图解析

<!-- 配置视图解析器 -->

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

<!-- 配置逻辑视图的前缀 -->

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

<!-- 配置逻辑视图的后缀 -->

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

逻辑视图名需要在controller中返回ModelAndView指定，比如逻辑视图名为ItemList，则最终返回的jsp视图地址:“WEB-INF/jsp/itemList.jsp”

4、配置静态资源不拦截

<mvc:resources location="/img" mapping="/\*\*"/> //配置了img包下的所有不进行拦截

5、配置响应浏览器编码

<!-- 在spring-mvc.xml文件中加入这段配置后，spring返回给页面的都是utf-8编码了 -->

<bean

class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">

<property name="messageConverters">

<list>

<bean

class="org.springframework.http.converter.StringHttpMessageConverter">

<property name="supportedMediaTypes">

<list>

<value>text/html;charset=UTF-8</value>

</list>

</property>

</bean>

</list>

</property>

</bean>

# 其它

默认加载配件 有些默认加载的映射器是废弃的（还能用），所以还需要配置代替它们

spring-webmvc包下：org\springframework\web\servlet\DispatcherServlet.properties

AcceptHeaderLocaleResolver: 国际化（i18n）, 当Accept-Language==null时，使用系统默认Locale

FixedThemeResolver：默认的主题解析器。springmvc主题静态资源的集合，包括样式及图片，用来控制应用的视觉风格

BeanNameUrlHandlerMapping：xml的默认映射器：通过BeanName与Handler产生关系。即将BeanName映射到Handler.

DefaultAnnotationHandlerMapping：2.5注解默认映射器，配合适配器开启支持@Controller 和 @RequestMapping

AnnotationMethodHandlerAdapter：2.5处理器适配器和处理器映射器 3.1后换了（需配置）

HttpRequestHandlerAdapter：其类似于一个简单的Servlet，只有一个handlerRequest方法

AnnotationMethodHandlerExceptionResolver：注解异常类

ResponseStatusExceptionResolver：异常类

DefaultHandlerExceptionResolver：还是异常类

DefaultRequestToViewNameTranslator：将请求URL转换为逻辑视图名

http://localhost:9080/web上下文/list -------> 逻辑视图名为list

http://localhost:9080/web上下文/list.html -------> 逻辑视图名为list(默认删除扩展名)

http://localhost:9080/web上下文/user/list.html -------> 逻辑视图名为user/list

InternalResourceViewResolver：这个最重要的需要配置的视图解析器

SessionFlashMapManager：用于管理FlashMap

springMVC工作流程

1、用户发送请求至前端控制器DispatcherServlet

https://loaclhost:8080/springmvc/UserController/add.do

2、DispatcherServlet收到请求调用HandlerMapping处理器映射器。

@Controller

public class UserController

3、处理器映射器根据请求url找到具体的处理器（自己写的类），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适配器调用处理器（自己写的类方法代码）

执行处理器(Controller，也叫后端控制器)。

5、Controller执行完成返回ModelAndView

6、HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet

7、DispatcherServlet将ModelAndView传给ViewReslover视图解析器

组件的说明

**DispatcherServlet：前端控制器（相当于老大指挥小弟做事）**

用户请求到达前端控制器，它就相当于mvc模式中的c，dispatcherServlet是整个流程控制的中心，由它调用其它组件处理用户的请求，dispatcherServlet的存在降低了组件之间的耦合性。

HandlerMapping：处理器映射器：全部映射器都实现了此接口类

HandlerMapping负责根据用户请求url找到Handler即处理器，springmvc提供了不同的映射器实现不同的映射方式，例如：配置文件方式，实现接口方式，注解方式等。

Handler：处理器 自己写的类有注解@controller或XML实现了controller接口类

Handler 是继DispatcherServlet前端控制器的后端控制器，在DispatcherServlet的控制下Handler对具体的用户请求进行处理。

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。

HandlAdapter：处理器适配器：全部适配器都实现了此接口类

通过HandlerAdapter对处理器进行执行，这是适配器模式的应用，通过扩展适配器可以对更多类型的处理器进行执行。

ViewResolver：视图解析器

View Resolver负责将处理结果生成View视图，View Resolver首先根据逻辑视图名解析成物理视图名即具体的页面地址，再生成View视图对象，最后对View进行渲染将处理结果通过页面展示给用户。

View：视图

springmvc框架提供了很多的View视图类型的支持，包括：jstlView、freemarkerView、pdfView等。我们最常用的视图就是jsp。

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。

说明：在springmvc的各个组件中，处理器映射器、处理器适配器、视图解析器称为springmvc的三大组件。

需要用户开发的组件有handler、view