# SpringMVC

**一、入门**

**配置文件**

<?**xml version="1.0" encoding="UTF-8"**?>  
<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"  
 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/mvc  
 http://www.springframework.org/schema/mvc/spring-mvc.xsd  
 http://www.springframework.org/schema/context  
 http://www.springframework.org/schema/context/spring-context.xsd"**>  
  
 <!--开启注解扫描-->  
 <**context:component-scan base-package="com.yuan"**/>  
  
 <!--配置视图解析器-->  
 <**bean id="internalResourceViewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>  
 <**property name="prefix" value="/WEB-INF/pages/"**/>  
 <**property name="suffix" value=".jsp"**/>  
 </**bean**>  
  
 <!--开启框架注解支持-->  
 <**mvc:annotation-driven**/>  
</**beans**>

**Controller类**

**package** com.yuan.controler;  
  
**import** org.springframework.stereotype.Controller;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
  
/\*\*  
 \* 控制器实例类  
 \*/  
@Controller  
**public class** HelloController {  
  
 @RequestMapping(**"/hello"**)  
 **public** String sayHello(){  
 System.**out**.println(**"你好 SpringMVC"**);  
 **return "success"**;  
 }  
}

**pom文件**

<**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 <**spring.version**>5.0.2.RELEASE</**spring.version**>  
</**properties**>  
  
<**dependencies**>  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.11</**version**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-context</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-web</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-webmvc</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>javax.servlet</**groupId**>  
 <**artifactId**>servlet-api</**artifactId**>  
 <**version**>2.5</**version**>  
 <**scope**>provided</**scope**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>javax.servlet.jsp</**groupId**>  
 <**artifactId**>jsp-api</**artifactId**>  
 <**version**>2.0</**version**>  
 <**scope**>provided</**scope**>  
 </**dependency**>  
  
</**dependencies**>

**入门案例中的组件分析**

1. 前端控制器（DispatcherServlet）

2. 处理器映射器（HandlerMapping）

3. 处理器（Handler）

4. 处理器适配器（HandlAdapter）

5. 视图解析器（View Resolver）

6. 视图（View）

**二、@RequestMapping详解**

1. RequestMapping注解的作用是建立请求URL和处理方法之间的对应关系

2. RequestMapping注解可以作用在方法和类上

1). 作用在类上：第一级的访问目录

2). 作用在方法上：第二级的访问目录

3). 细节：路径可以不编写 / 表示应用的根目录开始

4). 细节：${ pageContext.request.contextPath }也可以省略不写，但是路径上不能写 /

3. RequestMapping的属性

1). path 指定请求路径的url

2). value value属性和path属性是一样的

3). mthod 指定该方法的请求方式

4). params 指定限制请求参数的条件

5). headers 发送的请求中必须包含的请求

**三、请求参数绑定**

**1.普通参数**

1)方法添加参数

@RequestMapping(**"/param"**)  
**public** String paramUser(String username,String password){  
 System.**out**.println(**"username = "** + username);

System.**out**.println(**"password = "** + password);  
 **return "param"** ;  
}

2)页面属性=参数名

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

**2.JavaBean**

1)方法添加参数

@RequestMapping(**"/param"**)  
**public** String paramUser(User user){  
 System.**out**.println(**"user = "** + user);  
 **return "param"** ;  
}

2)页面属性=javaBean参数中的属性名

<**form action="first/param" method="post"**>  
 用户名:<**input type="text" name="username"**/><**br**/>  
 密码:<**input type="text" name="password"**/><**br**/>  
 年龄:<**input type="text" name="age"**/><**br**/>  
 papa:<**input type="text" name="js.papa"**/><**br**/>  
 mama:<**input type="text" name="js.mama"**/><**br**/>  
 <**input type="submit" name="提交"**/>  
</**form**>

**3.复杂参数**

1)方法添加参数

@RequestMapping(**"/param"**)  
**public** String paramUser(User user){  
 System.**out**.println(**"user = "** + user);  
 **return "param"** ;  
}

2)页面List下标从零开始，Map自己命名

<**form method="post" action="/first/param"**>  
 账户:<**input type="text" name="username"**/><**br**/>  
 密码:<**input type="text" name="password"**/><**br**/>  
 角色名1:<**input type="text" name="accounts[1].accountName"**/><**br**/>  
 职业1:<**input type="text" name="accounts[1].accountJob"**/><**br**/>  
 属性1:<**input type="text" name="accounts[1].accountPara"**/><**br**/>  
 角色名2:<**input type="text" name="accounts[3].accountName"**/><**br**/>  
 职业2:<**input type="text" name="accounts[3].accountJob"**/><**br**/>  
 属性2:<**input type="text" name="accounts[3].accountPara"**/><**br**/>  
 <**input type="submit" name="保存"**/>  
</**form**>

**4.中文乱码**

配置web.xml过滤器

<!--中文乱码过滤器-->  
<**filter**>  
 <**filter-name**>characterEncodingFilter</**filter-name**>  
 <**filter-class**>org.springframework.web.filter.CharacterEncodingFilter</**filter-class**>  
 <**init-param**>  
 <**param-name**>encoding</**param-name**>  
 <**param-value**>UTF-8</**param-value**>  
 </**init-param**>  
</**filter**>  
<**filter-mapping**>  
 <**filter-name**>characterEncodingFilter</**filter-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
</**filter-mapping**>

**5.自定义类型转换**

1)编写工具类

**public class** DateFmtUtil **implements** Converter<String, Date> {  
 @Override  
 **public** Date convert(String source) {  
 // 判断  
 **if** (source == **null**) {  
 **throw new** RuntimeException(**"请您传入数据"**);  
 }  
 **try** {  
 // 把字符串转换日期  
 **return new** SimpleDateFormat(**"yyyy-MM-dd"**).parse(source);  
 } **catch** (Exception e) {  
 **throw new** RuntimeException(**"数据类型转换出现错误"**);  
 }  
 }  
}

2)注册为组件

<!--注册日期工具类组件-->  
<**bean class="org.springframework.context.support.ConversionServiceFactoryBean" id="conversionService"**>  
 <**property name="converters"**>  
 <**bean class="com.yuan.util.DateFmtUtil"**/>  
 </**property**>  
</**bean**>  
  
<!--开启框架注解支持-->  
<**mvc:annotation-driven conversion-service="conversionService"**/>

**四、常用注解**

1.@RequestParam

范围：方法参数上(value = “表单属性名”)

作用：使 由于页面和传入参数的变量名不同导致 的 赋值失败情况变为成功

例子：

@RequestMapping(**"/param"**)  
**public** String paramUser(@RequestParam(**"uname"**) String username){  
 System.**out**.println(**"username = "** + username);  
 **return "param"** ;  
}

2.@RequestBody

范围：方法参数上

作用：用于获取请求体的内容（注意：get方法不可以）以键值对的形式aa=AA&bb=BB

例子：

@RequestMapping(**"/param"**)  
**public** String paramUser(@RequestBody String body){  
 System.**out**.println(**"body = "** + body);  
 **return "param"** ;  
}

3.@PathVaribale

范围：方法参数上(value = “restful路径上的占位符”)

作用： 把路径上的值取到赋值给方法上的形参

例子：

@RequestMapping(**"/param/{qid}"**)  
**public** String paramUser(@PathVariable(**"qid"**) Integer id){  
 System.**out**.println(**"id = "** + id);  
 **return "param"** ;  
}

4.@RequestHeader

范围：方法参数上value = “想要获取值的头名“

作用：获取指定请求头的值

例子：

@RequestMapping(**"/param"**)  
**public** String paramUser(@RequestHeader(**"请求头"**) Integer name){  
 System.**out**.println(**"name = "** + name);  
 **return "param"** ;  
}

5.@CookieValue

范围：方法参数上value = “想要获取值的Cookie名

作用：获取指定请求头的值

例子：

@RequestMapping(**"/param"**)  
**public** String paramUser(@CookieValue(**"JSESSIONID"**) String jsessionid){  
 System.**out**.println(**"jsessionid = "** + jsessionid);  
 **return "param"** ;  
}

6.@ModelAttribute

范围：方法上、方法参数上

作用：以页面传来的值为准，没传来的由前一个方法返回的返回值填补，**@ModelAttribute标注过的方法会在@RequestMapping标注过的方法前运行**

例子1：方法上

@Controller  
@RequestMapping(**"/first"**)  
**public class** HelloController {  
 @RequestMapping(**"/modelAttribute"**)  
 **public** String second(User userFirst){  
 System.**out**.println(**"userFirst = "** + userFirst);  
 **return "param"** ;  
 }  
  
 @ModelAttribute  
 **public** User first(){  
 User userFirst = **new** User();  
 List<Account> accounts = **new** ArrayList<Account>();  
 Account account1 = **new** Account();  
 account1.setAccountName(**"setAccountName1"**);  
 account1.setAccountJob(**"setAccountJob1"**);  
 account1.setAccountPara(**"setAccountPara1"**);  
 Account account2 = **new** Account();  
 account2.setAccountName(**"setAccountName2"**);  
 account2.setAccountJob(**"setAccountJob2"**);  
 account2.setAccountPara(**"setAccountPara2"**);  
 accounts.add(account1);  
 accounts.add(account2);  
 userFirst.setAccounts(accounts);  
 **return** userFirst ;  
 }  
}

例子2：方法参数上

/\*\*  
 \* 控制器实例类  
 \*/  
@Controller  
@RequestMapping(**"/first"**)  
**public class** HelloController {  
 @RequestMapping(**"/modelAttribute"**)  
 **public** String second(@ModelAttribute(**"user1"**)User userFirst) {  
 System.**out**.println(**"userFirst = "** + userFirst);  
 **return "param"**;  
 }  
  
 @ModelAttribute  
 **public void** first(Map<String,User> userMap) {  
  
 User userFirst = **new** User();  
 List<Account> accounts = **new** ArrayList<Account>();  
 Account account1 = **new** Account();  
 account1.setAccountName(**"setAccountName1"**);  
 account1.setAccountJob(**"setAccountJob1"**);  
 account1.setAccountPara(**"setAccountPara1"**);  
 accounts.add(account1);  
 userFirst.setAccounts(accounts);  
 userMap.put(**"user1"**,userFirst);  
 **return**;  
 }  
}

7.@SessionAttribute

范围：

作用：用于多次执行控制器方法间的参数共享

例子：

/\*\*  
 \* 控制器实例类  
 \*/  
@Controller  
@RequestMapping(**"/first"**)  
  
/\*\*  
 \* 把数据存入到session域对象中  
 \*/  
@SessionAttributes(value = {**"username"**, **"password"**, **"age"**}, types = {String.**class**, Integer.**class**})  
**public class** HelloController {  
  
 /\*\*  
 \* 向session中存入值  
 \*  
 \* **@return** \*/  
 @RequestMapping(path = **"/save"**)  
 **public** String save(Model model) {  
 System.**out**.println(**"向session域中保存数据"**);  
 model.addAttribute(**"username"**, **"root"**);  
 model.addAttribute(**"password"**, **"123"**);  
 model.addAttribute(**"age"**, 20);  
 **return "success"**;  
 }  
  
 /\*\*  
 \* 从session中获取值  
 \*  
 \* **@return** \*/  
 @RequestMapping(path = **"/find"**)  
 **public** String find(ModelMap modelMap) {  
 String username = (String) modelMap.get(**"username"**);  
 String password = (String) modelMap.get(**"password"**);  
 Integer age = (Integer) modelMap.get(**"age"**);  
 System.**out**.println(username + **" : "** + password + **" : "** + age);  
 **return "success"**;  
 }  
  
 /\*\*  
 \* 清除值  
 \*  
 \* **@return** \*/  
 @RequestMapping(path = **"/delete"**)  
 **public** String delete(SessionStatus status) {  
 status.setComplete();  
 **return "success"**;  
 }  
}

**五、响应结果和数据视图**

**1.响应之返回值是String**

@RequestMapping(**"/stringTest"**)  
**public** String stringTest(Model mode) {  
 System.**out**.println(**"stringTest..............."**);  
 User user = **new** User();  
 user.setUsername(**"妹妹"**);  
 user.setPassworld(**"meimei"**);  
 user.setAge(21);  
 mode.addAttribute(**"user"**, user);  
 System.**out**.println(**"user = "** + user);  
 System.**out**.println(mode);  
 **return "success"**;  
}

**2.响应之返回值是void**

@RequestMapping(**"/voidTest"**)  
 **public void** voidTest(HttpServletRequest request, HttpServletResponse response) **throws** Exception {  
 System.**out**.println(**"voidTest................"**);  
// 转发  
 request.getRequestDispatcher(**"/success.jsp"**).forward(request, response);  
// 重定向  
 response.sendRedirect(request.getContextPath()+**"/response.jsp"**);  
 }

**3.响应之返回值是ModeAndView**

@RequestMapping(**"/ModeAndViewTest"**)  
**public** ModelAndView ModeAndViewTest(Model mode) {  
 ModelAndView mv = **new** ModelAndView();  
 User user = **new** User();  
   
 user.setUsername(**"妹妹"**);  
 user.setPassworld(**"meimei"**);  
 user.setAge(21);  
   
 // 把user对象存储到mv对象中，也会把user对象存入到request对象  
 mv.addObject(**"user"**,user);  
  
 // 跳转到哪个页面  
 mv.setViewName(**"success"**);  
   
 **return** mv;  
}

**4.响应之返回值是利用forward和redirect进行页面跳转**

/\*\*  
 \* 使用关键字的方式进行转发或者重定向  
 \* **@return** \*/  
@RequestMapping(**"/testForwardOrRedirect"**)  
**public** String testForwardOrRedirect(){  
 System.**out**.println(**"testForwardOrRedirect方法执行了..."**);  
  
 // 请求的转发  
 // return "forward:/WEB-INF/pages/success.jsp";  
  
 // 重定向  
 **return "redirect:/index.jsp"**;  
}

**5.响应json数据之过滤静态资源**

Spring MVC配置文件

<!--前端控制器，哪些静态资源不拦截-->  
<**mvc:resources location="/css/" mapping="/css/\*\*"**/>  
<**mvc:resources location="/images/" mapping="/images/\*\*"**/>  
<**mvc:resources location="/js/" mapping="/js/\*\*"**/>

**6.响应json数据之发送ajax的请求**

<**script src="js/jquery.min.js"**></**script**>  
  
<**script**>  
 // 页面加载，绑定单击事件  
 **$**(**function**(){  
 **$**(**"#btn"**).click(**function**(){  
 // alert("hello btn");  
 // 发送ajax请求  
 **$**.**ajax**({  
 // 编写json格式，设置属性和值  
 **url**:**"user/testAjax"**,  
 **contentType**:**"application/json;charset=UTF-8"**,  
 **data**:**'{"username":"hehe","password":"123","age":30}'**,  
 **dataType**:**"json"**,  
 **type**:**"post"**,  
 success:**function**(data){  
 // data服务器端响应的json的数据，进行解析  
 alert(data);  
 alert(data.username);  
 alert(data.password);  
 alert(data.age);  
 }  
 });  
 });  
 });  
</**script**>

**7.响应json数据之响应json格式数据**

/\*\*  
 \* 模拟异步请求响应  
 \*/  
@RequestMapping(**"/testAjax"**)  
@ResponseBody  
**public** User testAjax(@RequestBody User user){  
 System.**out**.println(**"testAjax方法执行了..."**);  
 // 客户端发送ajax的请求，传的是json字符串，后端把json字符串封装到user对象中  
 System.**out**.println(user);  
 // 做响应，模拟查询数据库  
 user.setUsername(**"hiahia"**);  
 user.setAge(40);  
 // 做响应  
 **return** user;  
}

**六、SpringMVC实现文件上传**

**1. 文件上传的回顾**

**1).Controller**

/\*\*  
 \* 普通文件上传  
 \* **@return** \*/  
@RequestMapping(**"/fileupload1"**)  
**public** String fileupload1(HttpServletRequest request) **throws** Exception {  
 System.**out**.println(**"文件上传..."**);  
  
 // 使用fileupload组件完成文件上传  
 // 上传的位置  
 String path = request.getSession().getServletContext().getRealPath(**"/uploads/"**);  
 // 判断，该路径是否存在  
 File file = **new** File(path);  
 **if**(!file.exists()){  
 // 创建该文件夹  
 file.mkdirs();  
 }  
  
 // 解析request对象，获取上传文件项  
 DiskFileItemFactory factory = **new** DiskFileItemFactory();  
 ServletFileUpload upload = **new** ServletFileUpload(factory);  
 // 解析request  
 List<FileItem> items = upload.parseRequest(request);  
 // 遍历  
 **for**(FileItem item:items){  
 // 进行判断，当前item对象是否是上传文件项  
 System.**out**.println(**"item = "** + item);  
 **if**(item.isFormField()){  
 // 说明普通表单向  
 }**else**{  
 // 说明上传文件项  
 // 获取上传文件的名称  
 String filename = item.getName();  
 // 把文件的名称设置唯一值，uuid  
 String uuid = UUID.randomUUID().toString().replace(**"-"**, **""**);  
 filename = uuid+**"\_"**+filename;  
 // 完成文件上传  
 item.write(**new** File(path,filename));  
 // 删除临时文件  
 item.delete();  
 }  
 }  
  
 **return "success"**;  
}

**2).pom**

<**dependency**>  
 <**groupId**>commons-fileupload</**groupId**>  
 <**artifactId**>commons-fileupload</**artifactId**>  
 <**version**>1.3.1</**version**>  
</**dependency**>  
  
<**dependency**>  
 <**groupId**>commons-io</**groupId**>  
 <**artifactId**>commons-io</**artifactId**>  
 <**version**>2.4</**version**>  
</**dependency**>

**3).xml**

<**form action="/user/fileupload1" method="post" enctype="multipart/form-data"**>  
 选择文件：<**input type="file" name="upload"** /><**br**/>  
 <**input type="submit" value="上传"** />  
</**form**>

**2.SpringMVC传统方式文件上传**

**1).Controller**

/\*\*  
 \* SpringMVC文件上传  
 \* **@return** \*/  
@RequestMapping(**"/fileupload2"**)  
**public** String fileuoload2(HttpServletRequest request, MultipartFile upload) **throws** Exception {  
 System.**out**.println(**"fileuoload2文件上传..."**);  
  
 // 上传的位置  
 String path = request.getSession().getServletContext().getRealPath(**"/uploads/"**);  
 // 判断，该路径是否存在  
 File file = **new** File(path);  
 **if**(!file.exists()){  
 // 创建该文件夹  
 file.mkdirs();  
 }  
  
  
 // 获取上传文件的名称  
 String filename = upload.getOriginalFilename();  
 // 把文件的名称设置唯一值，uuid  
 String uuid = UUID.randomUUID().toString().replace(**"-"**, **""**);  
 filename = uuid+**"\_"**+filename;  
 // 完成文件上传  
 upload.transferTo(**new** File(path,filename));  
 **return "success"**;  
}

**2).pom**

**3).xml**

<**h3**>Springmvc文件上传</**h3**>  
  
<**form action="/user/fileupload2" method="post" enctype="multipart/form-data"**>  
 选择文件：<**input type="file" name="upload"** /><**br**/>  
 <**input type="submit" value="上传"** />  
</**form**>

**3.SpringMVC跨服务器方式文件上传**

**1).Controller**

/\*\*  
 \* 跨服务器文件上传  
 \* **@return** \*/  
 @RequestMapping(**"/fileupload3"**)  
 **public** String fileuoload3(MultipartFile upload) **throws** Exception {  
 System.**out**.println(**"跨服务器文件上传..."**);  
  
 // 定义上传文件服务器路径  
 String path = **"http://localhost:9090/uploads/"**;  
  
 // 说明上传文件项  
 // 获取上传文件的名称  
 String filename = upload.getOriginalFilename();  
 // 把文件的名称设置唯一值，uuid  
 String uuid = UUID.randomUUID().toString().replace(**"-"**, **""**);  
 filename = uuid+**"\_"**+filename;  
  
 // 创建客户端的对象  
 Client client = Client.create();  
  
 // 和图片服务器进行连接  
 WebResource webResource = client.resource(path + filename);  
  
 // 上传文件  
 webResource.put(upload.getBytes());  
  
 **return "success"**;  
 }  
}

**2).pom**

<**dependency**>  
 <**groupId**>com.sun.jersey</**groupId**>  
 <**artifactId**>jersey-core</**artifactId**>  
 <**version**>1.18.1</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>com.sun.jersey</**groupId**>  
 <**artifactId**>jersey-client</**artifactId**>  
 <**version**>1.18.1</**version**>  
</**dependency**>

**3).xml**

<**h3**>跨服务器文件上传</**h3**>  
  
<**form action="/user/fileupload3" method="post" enctype="multipart/form-data"**>  
 选择文件：<**input type="file" name="upload"** /><**br**/>  
 <**input type="submit" value="上传"** />

**七、SpringMVC框架中的拦截器**

**1.拦截器的概述**

1. SpringMVC框架中的拦截器用于对处理器进行预处理和后处理的技术。

2. 可以定义拦截器链，连接器链就是将拦截器按着一定的顺序结成一条链，在访问被拦截的方法时，拦截器链 中的拦截器会按着定义的顺序执行。

3. 拦截器和过滤器的功能比较类似，有区别

1). 过滤器是Servlet规范的一部分，任何框架都可以使用过滤器技术。

2). 拦截器是SpringMVC框架独有的。

3). 过滤器配置了/\*，可以拦截任何资源。

4). 拦截器只会对控制器中的方法进行拦截。

4. 拦截器也是AOP思想的一种实现方式

5. 想要自定义拦截器，需要实现HandlerInterceptor接口。

**2.自定义拦截器步骤**

1. 创建类，实现HandlerInterceptor接口，重写需要的方法

/\*\*  
 \* **@author** Yuan-9826  
 \*/  
**public class** MyInterceptor1 **implements** HandlerInterceptor {  
  
 /\*\*  
 \* 预处理，controller方法执行前  
 \* return true 放行，执行下一个拦截器，如果没有，执行controller中的方法  
 \* return false不放行  
 \* **@param request** \* **@param response** \* **@param handler** \* **@return** \* **@throws** Exception  
 \*/  
 @Override  
 **public boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) **throws** Exception {  
 System.**out**.println(**"MyInterceptor1拦截器成功了"**);  
 **return true**;  
 }  
  
 /\*\*  
 \* 后处理方法，controller方法执行后，success.jsp执行之前  
 \* **@param request** \* **@param response** \* **@param handler** \* **@param modelAndView** \* **@throws** Exception  
 \*/  
 @Override  
 **public void** postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) **throws** Exception {  
 System.**out**.println(**"MyInterceptor1执行了...后2222"**);  
 // request.getRequestDispatcher("/WEB-INF/pages/error.jsp").forward(request,response);  
 }  
  
 /\*\*  
 \* success.jsp页面执行后，该方法会执行  
 \* **@param request** \* **@param response** \* **@param handler** \* **@param ex** \* **@throws** Exception  
 \*/  
 @Override  
 **public void** afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) **throws** Exception {  
 System.**out**.println(**"MyInterceptor1执行了...最后2222"**);  
 }  
}

2.在springmvc.xml中配置拦截器类

<!--配置拦截器-->  
<**mvc:interceptors**>  
 <!--配置拦截器-->  
 <**mvc:interceptor**>  
 <!--要拦截的具体的方法-->  
 <**mvc:mapping path="/interceptor/\*"**/>  
 <!--不要拦截的方法  
 <mvc:exclude-mapping path=""/>  
 -->  
 <!--配置拦截器对象-->  
 <**bean class="com.yuan.interceptor.MyInterceptor1"**/>  
 </**mvc:interceptor**>  
</**mvc:interceptors**>

**3.配置多个拦截器**

多个拦截器是按照配置的顺序决定的

<!--配置拦截器-->  
<**mvc:interceptors**>  
 <!--配置拦截器1-->  
 <**mvc:interceptor**>  
 <!--要拦截的具体的方法-->  
 <**mvc:mapping path="/interceptor/\*"**/>  
 <!--不要拦截的方法  
 <mvc:exclude-mapping path=""/>  
 -->  
 <!--配置拦截器对象-->  
 <**bean class="com.yuan.interceptor.MyInterceptor1"**/>  
 </**mvc:interceptor**>  
</**mvc:interceptors**>

<!--配置拦截器2-->  
<**mvc:interceptors**>  
 <!--配置拦截器-->  
 <**mvc:interceptor**>  
 <!--要拦截的具体的方法-->  
 <**mvc:mapping path="/interceptor/\*"**/>  
 <!--不要拦截的方法  
 <mvc:exclude-mapping path=""/>  
 -->  
 <!--配置拦截器对象-->  
 <**bean class="com.yuan.interceptor.MyInterceptor2"**/>  
 </**mvc:interceptor**>  
</**mvc:interceptors**>