SpringMVC

# Spring MVC概述

SpringMVC是一个WEB层、控制层框架，主要用来与客户端交互，业务逻辑的调用

SpringMVC是Spring大家族的一大组件，Spring整合SpringMVC可以做到无缝集成。

为了么有了Servlet还要学SpringMVC ？

Servlet的开发配置相对麻烦，servlet特别多的时候web.xml文件将会非常臃肿

每个Servlet都只能处理一个功能，如果需要多个功能就需要开发多个Servlet，项目中存在大量Servlet显得臃肿。

获取请求参数 进行类型转换 封装数据到bean的 过程比较繁琐。

其他开发中不方便的地方，例如，乱码问题..数据格式处理..表单校验。

# Spring MVC的组件

## 前端控制器(中央调度器)

本质上是一个Servlet，相当于一个中转站，所有的访问都会走到该Servlet，然后通过配置中转到相应的Handler(处理器)，获取到数据和视图后，再使用相应视图做出响应。

## 处理器映射器

本质上就是保存的处理器(Handler)和访问路径的对应关系，在需要时供前端控制器查阅

## 处理器适配器

本质上是一个适配器，可以根据要求找到对应的处理器来运行。前端控制器通过处理器映射器找到对应的处理器信息后，将请求响应对应的Handler信息交给处理器适配器处理，处理器适配器找到真正的Handler执行后，将结果Model和view返回给前端控制器。

## 处理器(Handler)

处理具体任务的类或方法。通常是通过@Controller注解修饰的类或方法。

## 视图解析器

本质还是一种映射关系，可以将视图名映射到真正的视图地址。前端控制器调用处理器适配完成后得到model和view，将view信息传给视图解析器得到真正的view

## 视图

本质上就是将handler处理器中返回的model数据嵌入到视图解析器解析后得到的jsp页面中，向客户端做出响应

|  |
| --- |
| mhtml:file://E:\JAVA\框架\课前资料-ssm\SpringMVC\课堂笔记\SpringMVC.mht!SpringMVC.files/image001.png |

# Spring MVC组件的配置

## 导入SpringMVC相关开发包

### Xml配置方式

|  |
| --- |
| 计算机生成了可选文字: 〔 ， WebR00t  META-INF  〔 ， WEB-INF  aopalliance-1.O.Jar  a ' pe 戲 」 一 17 刂 ar  aspecgweaver-1.7.4.Jar  的 ． 0 且 1 勾 ar  commons-fileupload-l„  commons-IO-2.O.1.Jar  commons-logging-  hibernate—validator-4.1.  」 acL ： ' on 叱 ore 一 1.  」 ackson-mapper-asl-l.g  」 pi 1.2jar  」 一 im 司 1.2.jar |

### 注解方式

|  |
| --- |
| 计算机生成了可选文字: ． SprlngMVCDayOI OI_Flrst E:XworkspaceXid  V WEB-INF  1  aopalliance-l .0.Jar  1  a ' pe 戲 」 一 17 刂 ar  1  aspecgweaver-17 刂 ar  1  c3pO ． 0 · 1.2.Jar  1  1  commons-fileupload-122j3r  1  commons-IO 2 £ · 1 Jar  1  commons-logging-10.4.Jar  1  £ 下 in 引 永  1  」 ac' ： ' on nnota 乜 on ' ． 2 jar  1  」 ackson-core-2.g.8.Jar  1  」 ackson-databind-2.g.8.Jar  1  」 pi 一 1 ajar  1  」 一 im 司 一 1.2jar  1  log4jjar  1  mysql-connector-Java-5.I.IO-bi  1  pi 一 1 5.8.jar  1  一 | 。 g 12j3r |

## 创建配置文件

在src目录下创建springMVC.xml

SpringMVC默认会自动在web应用的WEB-INF目录下去寻找[前端控制器ServletName]-servlet.xml作为当前SpringMVC的核心配置文件。

或者在前端控制器中配置目录

创建这个文件，这个文件本身其实就是Spring的配置文件，所以导入Spring相关的约束信息，包括 beans、context、mvc

|  |
| --- |
| 计算机生成了可选文字: K?xml version:"1.9" encoding:"U 丆 F-8"b  < bea n s xmlns:"http://www. springframework. org/schema/beans  xmlns ： context: " h 亡 亡 ://www. springframework ． org/schema/context ”  xmlns ： aop: "h 亡 亡 ://www. springframework ． org/schema/aop  xmlns ： m v c = " h 亡 亡 ://www. springframework. org/schema/mvc '  xmlns ： xsi: " h 亡 亡 ://www. 射 3 ． org/201/XMLSchema- instance  xsi ： schemaLocation: " h 亡 ： //www. springframework ． org/schema/beans  org/schema/beans/spring-beans-3.2 ．  org/schema/context  org/schema/context/spring-context-  org/schema/aop  org/schema/aop/spring-aop-3.2.xsd  org/schema/mvc  org/schema/mvc/spring-mvc-3.2.xsd  http://www  http://www  http://www  http://www  http://www  http://www  http://www  ． springframework ． |

## 配置前端控制器(WEB.xml中配置)

前端控制器本质为Servlet，需要在web.xml中配置Servet。

|  |
| --- |
| <servlet>  <servlet-name>springMvc</servlet-name>  <!--如果报错DispatcherServlet' is not assignable to javax.Servlet 则需要导入 tomcat-->  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <!--如果不配置 会到默认目录找默认文件-->  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:springMVC.xml</param-value>  </init-param> </servlet> <servlet-mapping>  <servlet-name>springMvc</servlet-name>  <url-pattern>\*.action</url-pattern><!--随便写个路径就可以--> </servlet-mapping> |

## 配置视图解析器(Spring MVC配置文件中)

在SpringMVC配置文件中配置

|  |
| --- |
| <!--配置视图解析器--> <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  <!--前缀和后缀，处理器适配器返回视图后会默认在视图名加上前缀后缀-->  <!--前缀和后缀，处理器如果返回的时资源跳转的字符串，则不会匹配前缀后缀-->  <property name="prefix" value="/jsp"/>  <property name="suffix" value=".jsp"/> </bean> |

## 创建处理器(类中、方法中配置注解)

### 配置方式创建

### 创建处理器

想要开发一个处理器，写一个类实现Controller接口，重写handlerRequest方法，编写代码处理请求，并将处理好的数据和目标视图封装到ModelAndView中返回

|  |
| --- |
| 计算机生成了可选文字: public class He110 implements Controller  实  public ModelAndView handleRequest(HttpServ1etRequest request,  HttpServ1etResponse response) throws Exception {  / / 1 ． BIJWv10de1AndView  ode1AndView mav = new ModelAndView();  / / 2 ． 封 装 据  mav ． （ "msgl  " he110 ， worldN" ） ；  mav ． addObject( "msg2  "hello,springmvc 、  / / 3 ． 封 装 视 图  mav ． setViewName （ " he110  封 装 视 图 名 称  / / 4 ． 返 回 Mode IAndVe IW  return mav;  E@mav  创 建 mav  封 装 若 干 键  值 对 数 据 |

#### 配置处理器映射器中的路径和处理器的映射关系

在SpringMVC配置文件中配置

|  |
| --- |
|  |

#### 处理流程

|  |
| --- |
|  |

### 注解方式创建

#### 在SpringMVC配置文件中配置

|  |
| --- |
| <!--开启Spring MVC的包扫描 存放处理器的包--> <context:component-scan base-package="cn.shu.blog.web"/> <!--开启注解方式 MVC--> <mvc:annotation-driven/> |

#### 创建处理器

在SpringMVC包扫描目录下开发控制器，并配置为Spring的bean，在其中开发控制器方法，并通过@RequestMapping注解进行路径映射

|  |
| --- |
| 计算机生成了可选文字: @Controller  public classl FirstControIIer {  @RequestMapping （ /test01. action")  public Mode IAndVi ew test010 {  ， ， 夕 留 厂 对 熨 型 意  Mode IAndV i ew mav = new ModelAndView()  mav ． addObject （ attributeName: "kl" attributeValue:  mav ． addObject （ attributeName: "k2" attributeValue:  刀 身 初 图 意  mav. setViewName （ " tl 勹  云 回 翮 艹  re turn maV ；  " VI 勹 ，  " v2 勹 ， |

#### 处理流程

|  |
| --- |
| 计算机生成了可选文字: <％@ page contentType:'text/html .charset:UTF-8" language:  "Java" ％ >  Chead:-  etitle:-Titlec:/t1tle:-  ebodp  first-jsp---- $1k1 )$fk2)$fk3)  c/bodp  Vi eW  Model  1 spat cherServI  c:servlet»  cservlet-clasvorg-springframework-web-servlet-DispatcherServletc/servlet-clasv  c:/servlet»  curl-patterrv*-action«/url-patterrv  Handl ermapping  处 理 器 映 射 器  Model&View  @Controller  public class FirstController  Re uestMa i n "/first.action"  pu I C  ModelAndView mav new ModelAndView();  mav-addObject("k1 ","vxl 勹 ；  mav-addObject("k2","vx2");  mav-addObject("k3","vx3");  mav-setViewName("first"),  return mav-  " 端 控 制 器  Vi “  Vi  视 图 解 折 器  4 一 翻 图 一 一 >  Model Vi ew  Handl erAdapt er  处 理 器 适 配 器  5  an  处 理 器  handl er  处 理 器  handl er  处 理 器  handl er  处 理 器  handl er  处 理 器  ebean class:.org/springframework web.servlet.view.lnternalResourceViewResolver5  cproperty name:"prefix" value:  "IWEB-lNF/jsp/'S«/propertp  cproperty name:"suffix" value: '.Jsp'Sc/propertp  c:/bearv |

#### SpringMVC注解方式工作原理

1、服务器启动时，先加载WEB.xml，创建前端控制器Servlet，前端控制器加载SpringMVC配置文件。

2、当解析到包扫描时，扫描指定包，将有@Controler注解的类解析为处理器。

3、如果配置了注解方式，就会解析Spring-MVC注解。

4、解析@RequestMapping(value="/hello.action")，将指定的地址和当前方法的映射关系进行保存。

5、当用户发出请求访问一个地址时，SpringMVC寻找该地址映射关系，如果存在，则找到响应处理器相应方法执行，如果找不到，则报404。

## 配置处理器映射器、处理器适配器

处理器映射器相当于在方法注解上 就已经配置了，处理器适配器不需要配置

# @RequestMapping注解详解

通过注解实现访问路径到处理器方法的映射

## @RequestMapping基本使用

* 可以用在方法上和类上
* 用在方法上表示将该方法变为一个处理器，且和指定路径做映射。
* 用在类上则配置的路径会作为这个类中所有处理器的路径的父路径使用。
* 在访问时要加上父路径和方法上的路径才能由该处理器处理

## @RequestMapping注解属性

|  |
| --- |
|  |

### String name() default "";

### String[] value() default {};

指定访问的路径 KEY为value，使用时如果注解只有一个参数，”value=“ 可省略，若KEY对应的只有一个值，则方括号 { } 可省略

|  |
| --- |
|  |

访问：localhost/addComment.action

value属性支持通配符匹配：

|  |
| --- |
|  |

访问：localhost/login/后接任意字符都可访问

### String[] path() default {};

与value同义，path(value)（path 和 value 互相引用，参见RequestMapping接口源码）path属性，和B中的value属性使用一致，两者都是用来作为映射使用的。也可使用通配符。

### RequestMethod[] method() default {};

指定该处理器接收哪种方式的请求，以下表示只接收POST请求

|  |
| --- |
|  |

### String[] params() default {};

* 用来限定当前请求中必须包含指定名称的请求参数才会被当前处理器处理
* 通过params属性指定只处理请求参数符合指定要求的请求

|  |
| --- |
| 格式1:只指定名称，要求必须具有该名称的请求参数  格式2:以"名称=值"或"名称!=值"的方式指定必须具有某个请求参数，且值必须等于或不等于给定值  格式3:以"!名称"的方式指定必须不包含指定名称的请求参数 |

以下表示接收有name参数，且gender为male，且没有!age...的请求

|  |
| --- |
|  |

### String[] headers() default {};

* 用来限定当前请求中必须包含指定名称的请求头才会被当前处理器处理

|  |
| --- |
| 格式1:只指定名称，要求必须具有该名称的请求头  格式2:以"名称=值"或"名称!=值"的方式指定必须具有某个请求头，且值必须等于或不等于给定值  格式3:以"!名称"的方式指定必须不包含指定名称的请求头 |

以下注解表示必须有请求头host，且等于localhost

|  |
| --- |
|  |

### String[] consumes() default {};

* 指定处理请求的提交内容类型（Content-Type），例如：application/json、text/html时，才能够让该方法处理请求

例：只接收json数据类型的请求

|  |
| --- |
|  |

### String[] produces() default {};

* 指定返回的内容类型，返回的内容类型必须是request请求头（Accept）中所包含的类型

例：返回json数据

|  |
| --- |
|  |

* 此外，produces属性还可以指定返回值的编码

例：指定utf-8编码

|  |
| --- |
|  |

好像只对@ResponseBody修饰的方法有效？**测试了，好像是**

# 获取请求参数

## 通过request对象获取

在处理器方法上传递request对象，SpringMVC会自动传递

然后通过request的getParameter获取

|  |
| --- |
|  |

## 形参直接接收参数

|  |
| --- |
|  |

### 如果请求参数名和方法形参名不同（@RequestParam）

可通过@RequestParam来指定该形参接收哪个请求参数

|  |  |
| --- | --- |
| value | 参数名字，即入参的请求参数名字，如value=“delId”表示请求的参数区中的名字为delId的参数的值将传入 |
| required | 是否必须，默认是true，表示请求中一定要有相应的参数，否则将报400错误码； |
| defaultValue | 默认值，表示如果请求中没有同名参数时的默认值 |

例：使用形参myName接收请求参数中的name参数

|  |
| --- |
|  |

### 请求参数封装到bean

#### 普通属性处理

前提是bean中必须提供相应的set方法，且传递的参数名需与bean中set方法对应

Bean:

|  |
| --- |
|  |

处理器：可以把请求参数封装到User bean中的userName属性

|  |
| --- |
|  |

参数:

|  |
| --- |
|  |

#### Bean中属性有Bean的处理

Comment Bean：此时Comment中有User类型

|  |
| --- |
|  |
|  |

处理器：

|  |
| --- |
|  |

前端请求：user.userName方式，可以把参数赋值给user bean中的userName属性

|  |
| --- |
|  |

### 多个同名参数

当传递的参数有多个同名参数时

|  |
| --- |
| 计算机生成了可选文字: < t r 〉  (td) 爱 F'ltd)  "checkbox"  "checkbox"  "checkbox"  "checkbox"  name:  name:  name:  name:  "Like  "Like  "Like  "Like  ' value  value  ' value  ' value  zq / 〉 足 琳  "Lq' / 〉 篮 琳  "qq ' / 〉 铅 琳  < t d 〉  (input  (input  (input  (input  Kltd)  K/tr)  type  type  type  type |

* 如果形参中直接指定String类型，则SpringMVC会以逗号分隔保存到该形参中。
* 如果形参中指定String[] 数组类型，则SpringMVC会保存到该数组中。

### 请求参数中文乱码问题

Tomcat默认使用ISO-8859-1编码，使用编码应该和Tomcat版本有关。

#### SpringMVC提供了过滤器用来解决全站POST提交乱码

这种方式只能解决POST提交的乱码，对GET方式提交的乱码无效！

|  |
| --- |
| 计算机生成了可选文字: 配 置 Spr1ngMVC 乱 码 解 决 过 滤 器 一 一 〉  (filter)  (filter-nam»characterEncodingFi1terK/fi1ter-narne)  (filter-clas»org ． springframework ． web ． filter ． CharacterEncodingFi1ter  (init—param)  'param—name)encoding'/pararn—name)  (param-valu»utf-8  4/param—value)  K/init—param)  K/filter)  (filter-mappino  (filter-nam»characterEncodingFi1terK/fi1ter-narne)  Kurl-pattern)/*(/url-pattern)  K/filter-mappino |

#### Get提交乱码(手动编解码)

Tomcat默认使用ISO-8859-1编码，使用编码应该和Tomcat版本有关。

先获取字节，再转utf-8

|  |
| --- |
| mhtml:file://E:\JAVA\框架\SpringMVCDay02_下午\笔记\SpringMVC.mht!SpringMVC.files/image037.png |

#### 也可以直接修改Tomcat中连接器的配置来使tomcat默认采用指定编码处理请求参数

但这种方式不建议大家使用，因为生产环境下不一定允许修改此项

|  |
| --- |
| 计算机生成了可选文字: -Connector pr0t000 工 = " HT 了 p ／ 1 ． 1 "  connection 二 = " 20000 "  UR 工 " |

### 日期数据处理

在处理器方法上，如果指定Date类型的形参时，SpringMVC无法自动封装，需要手动注册适配器来指定转换方式。

|  |
| --- |
|  |

### SpringMVC文件上传

#### 准备文件上传表单

文件上传表单必须满足如下三个条件

* 文件上传项必须有name属性
* 表单必须是Post提交
* 表单必须是enctype="multipart/form-data"

表单：

|  |
| --- |
| 计算机生成了可选文字: action  $fpageContext. request.contextPath } / heLL08 ． 00 亡 讠 on "  < t r 〉  (td) 文 件 Kltd)  (tdxinput type  K/tr)  'file  name:  "fx"/x/td) |

#### SpringMVC配置文件中配置文件上传工具

|  |
| --- |
| 计算机生成了可选文字: 配 置 文  、 1 。 bean  必 须 是 这 个 i d 必 须 是 这 个 类  'bean id: 'muLtipartResoLver  class:"or · S 亡 ． commons. artResoLV  property name:"maxUpLoadStze" value:"1924"X/property)  K/bean)  配 罱 上 传 文  件 时 的 黲 敵 |

#### 在处理器中实现文件上传

处理器可以接收一个MutilpartFile形参，形参名与表单字段name一致，或者通过@RequestParam指定。

该参数可以获取文件的输入流、相关信息，或者直接调用transferTo(File file)保存到

本地。

|  |
| --- |
|  |

### RESTFul风格的请求参数处理

#### 可以将参数放到请求的路径中，然后处理器获取路径的值作为参数

* 普通get请求：

|  |
| --- |
| Url：localhost/XXXX/addUser.action?name=tom |

* RESTFul风格的请求：

|  |
| --- |
| Url：localhost/XXXX/addUser/tom/18.action |

#### 在路径中用{ }指定参数名 处理器形参用@PathVariable指定

|  |
| --- |
|  |

#### 访问：最终获取到的就是值shuxinsheng

|  |
| --- |
|  |

# SpringMVC资源跳转

SpringMVC的资源跳转不会匹配视图解析器的前缀和后缀

## 请求转发

### 传统方式实现请求转发

|  |
| --- |
| @RequestMapping("/test01.action") public void test01(*HttpServletRequest* request, *HttpServletResponse* response) throws ServletException, IOException {  request.getRequestDispatcher("/index.jsp").forward(request,response); } |

### springmvc方式实现请求转发

可以通过返回forward:/xxxx.xxx格式的字符串表明要转发到指定地址

|  |
| --- |
| @RequestMapping(**"/test02.action"**)  **public** String test02(){  **return "forward:/index.jsp"**;  } |

## 请求重定向

### 传统方式实现请求重定向

|  |
| --- |
| @RequestMapping(**"/test01.action"**)  **public void** test01(HttpServletRequest request,HttpServletResponse response) **throws** IOException {  response.sendRedirect(request.getContextPath()+**"/index.jsp"**);  } |

### SpringMVC方式实现请求重定向

可以通过返回redirect:/xxxx.xxx格式的字符串表明要重定向到指定地址

通过这种方式实现请求重定向时不用在路径前写应用名，SpringMVC会自动拼接应用名

|  |
| --- |
| @RequestMapping(**"/test02.action"**)  **public** String test02(){  **return "redirect:/index.jsp"**;  } |

## 定时刷新

SpringMVC中没有提供实现定时刷新的便捷方式，只能用传统方式实现定时刷新。

|  |
| --- |
| @RequestMapping(**"/test01.action"**)  **public void** test01(HttpServletRequest request,HttpServletResponse response) **throws** IOException {  response.setContentType(**"text/html;charset=utf-8"**);  response.getWriter().write(**"注册成功，3秒后回到主页.."**);  response.setHeader(**"refresh"**,**"3;url="**+request.getContextPath()+**"/index.jsp"**);  } |

# 域的使用

## ServletRequest(request)域

### 传统方式

处理器方法上传递*HttpServletRequest*类型的形参，然后通过该参数写入或读取数据

### SpringMVC方式

* **写入**：通过在处理器方法上声明形参Model，通过该参数，写入数据，则默认写入reques 域中。注意：model不能读取。

|  |
| --- |
|  |

* **读取**：形参添加注解@RequestAttribute，自动注入

|  |
| --- |
|  |

## HttpSession(session)域

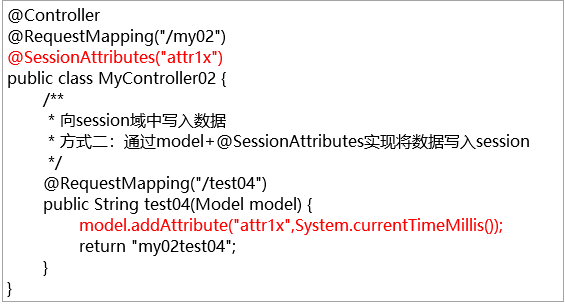
### 传统方式

形参获取HttpSession参数，SpringMVC自动注入，通过该参数操作Session

### SpringMVC方式

* **写入**：传递Model参数，然后在类上使用@SessionAttributes注解，可将写入到Model的参数自动写入到session，这个注解有个require参数，要求是否必须拿到，如果为true，则请求中没有该参数会报404。

注意：后面有个s，并且该注解只能用于类上



* **读取**：可在方法参数上添加注解，可从Session域中获取数据

|  |
| --- |
|  |

## ServletContext域

只能通过传统方式，在参数中传递request域，然后获取该域

|  |
| --- |
|  |

# 其它注解

## @ModelAttribute

### 使用在方法上

则被修饰的方法将会在当前类的任意handler(处理器)方法执行之前执行，该方法的返回值会自动存入mode中供后续使用。

|  |
| --- |
| **计算机生成了可选文字: @Mode1Attribute("uname") publicStringmaol(){ System.out return''25 .println("maol"); "'; } 自动将uname二25 存入model中** |

### 修饰方法形参

则会从model中获取属性值赋值到被修饰的方法参数上

|  |
| --- |
| 计算机生成了可选文字: @RequestMapping("/test02.action") publicvoidtestOZ(@Mode1Attribute("uname")Stringuname){ SVStem.OUt.D广intln(Un日me、： ．尸．、工日尸 自动从model中获取uname对应的值 赋值到当前方法参数上 飞J |

## @CookieValue

修饰处理器形参，用来从Cookie中获取指定名称的Cookie值

|  |
| --- |
| 计算机生成了可选文字: @RequestMaI)I)1ng （ /test01. action")  public void test01 (@CookieVaIue ("JSESSIONID") String v) {  System. 0 厶 printIn (v) |

## @RequestHeader

修饰处理器形参，用来获取指定名称请求头的值

|  |
| --- |
| 计算机生成了可选文字: @RenuestMapping （ /test01. action")  public void test01 (@RequestHeader ("User—Agent") String v) {  System. 0 厶 printIn (v) |

# 异常处理

## 为当前Controller配置错误处理

只需在当前Controller方法上加上@ExceptionHandler 注解,方法可接收Exception参 数

|  |
| --- |
| 计算机生成了可选文字: @ExceptionHand1er  public ModelAndView exceptionHand1er(Exception e)f  System. 0 亡 出 常 了 。  ModelAndView mav = new ModelAndView （ ） ；  mav ． addObject(' exception"  mav. setViewName ("err"),  return mav; |

## 注解方式配置全局的错误处理

专门定义一个类处理全局异常，类上需加上@ControllerAdvice注解

方法上加上@ExceptionHandler 注解,方法可接收Exception参数

|  |
| --- |
| 计算机生成了可选文字: @Contr011erAdvice  public class GlobalExceptionHand1er {  @ExceptionHand1er  public ModelAndView exceptionHand1er(Exception e)f  System. 0 亡 出 常 了 。  ModelAndView mav = new ModelAndView （ ） ；  mav ． addObject(' exception"  mav ． setViewName (' err2F') ；  return mav; |

## 配置文件方式配置全局错误处理 (了解)

在SpringMVC配置文件中配置

|  |
| --- |
| <!--配置错误处理--> <bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">  <property name="exceptionMappings">  <props>  <prop key="java.lang.Throwable">404</prop>  </props>  </property> </bean> |

# 返回数据

若方法返回值为String，则默认返回的是视图名并匹配前缀后缀。

若想返回一段数据给客户端，则需要用注解@ResponseBody修饰,或者在类上用@RestController替代@ResponseBody和@Controller

## 返回字符串数据

### 方式一：通过HttpServletResponse对象返回

|  |
| --- |
| @RequestMapping("/test01.action") public void test01(*HttpServletResponse* resp) throws Exception {  resp.setContentType("text/html;charset=utf-8");  resp.getWriter().write("abcde");  resp.getWriter().write("中国"); } |

### 方式二：参数直接获取PrintWriter对象返回

|  |
| --- |
| @RequestMapping("/test.action") public void test02(PrintWriter writer) throws Exception {  writer.write("shuxinsheng"); } |

### 方式三：通过@ResponseBody返回

|  |
| --- |
| /\*\*   向客户端直接返回数据 - 字符串  \*     方式二：通过@ResponseBody将返回值直接写入响应  \* 其中可以通过@RequestMapping的produces属性控制输出时的类型编码\*/  @ResponseBody @RequestMapping(value="/test.action",produces="text/html;charset=utf-8") public String test () {  return "舒新胜"; } |

## 返回JSON数据

### 方式一：手动拼接json 字符串

|  |
| --- |
| @ResponseBody @RequestMapping(value="/test.action") public String test04() {  String json = "{id:'123',name:'shuxinsheng'}";  return json; } |

### 方式二：通过配置@ResponseBody利用内置的jackson将对象处理为json返回

需要导入处理JSON的jar包

|  |
| --- |
| @ResponseBody @RequestMapping(value="/test.action",produces="application/json;charset=utf-8") public User test05() {  User user = new User();  return user; } |

# 处理器方法支持的参数类型和返回值类型总结

|  |
| --- |
|  |

|  |
| --- |
|  |

## 支持的方法参数类型

### HttpServletRequest

代表当前请求的对象

### HttpServletResponse

代表当前响应的对象

### HttpSession

代表当前会话的对象

### WebRequest

SpringMVC提供的对象，相当于是request和session的合体，可以操作这两个域中的属性。

### InputStream OutputStream Reader Writer

代表request中获取的输入流和response中获取的输出流

### 通过@PathVariable @RequestParam声明的方法参数

@PathVariable可以将请求路径的指定部分获取赋值给指定方法参数

@RequestParam可以将指定请求参数赋值给指定方法参数

如果不写此注解，则默认会将同名的请求参数赋值给方法参数

### 通过@RequestAttribute、@SessionAttribute和@ModelAttribute声明的方法参数

@RequestAttribute从request域中获取数据

@SessionAttribute从Session域中获取数据

@ModelAttribute从Model中获取数据

### 通过@CookieValue和@RequestHeader声明的方法参数

@CookieValue可以将请求中的指定名称的cookie赋值给指定方法参数

@RequestHeader可以将请求参数中的指定名称的头赋值给指定方法参数

### Model和ModelMap和java.util.Map

向这些Model ModelMap Map中存入属性，相当于向模型中存入数据

### Bean类

SpringMVC自动将请求参数封装到bean

### MultipartFile

实现文件上传功能时，接收上传的文件对象

### Errors  BindingResult

实现数据验证的参数

## 支持的返回值类型

### ModelAndView

可以返回一个ModelAndView对象，在其中封装Model和View信息

### \*View

可以直接返回一个代表视图的View对象

### 字符串

直接返回视图的名称

### void

如果返回值类型是void，则会自动返回和当前处理器路径名相同的视图名

### 方法被@ResponseBody修饰

当方法被@ResponseBody修饰时，默认将返回的对象转为json写入输出

### 除以上之外返回的任何内容都会被当做模型中的数据来处理，值为返回的数据，键为返回类型名首字母转小写，而返回的视图名等同于返回值为void的时的视图名