# SpringMVC

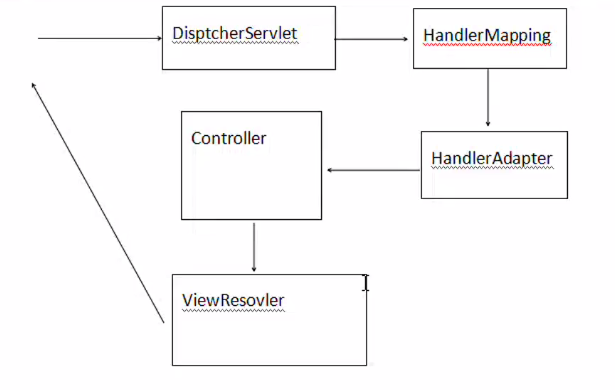
## DAY1

### SpringMVC简介

1. 重要组件

* **DispatcherServlet**：前端控制器，接收所有请求（如果配置 / 不包含jsp）
* **HandelerMapper**：解析请求格式，判断希望执行哪个具体的方法
* **HandleAdapter**：负责调用具体的方法
* **ViewResovler**：视图解析器。解析结果们准备跳转到具体的物理视图。

1. 运行原理图



### SpringMVC环境搭建

1. 导入jar
2. 在**web.xml**中配置前端控制器 **DispatchServlet**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app version=*"3.0"* xmlns=*"http://java.sun.com/xml/ns/javaee"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:web=*"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\_3\_0.xsd"* >  <servlet>  <servlet-name>abc</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>  <!-- 自启动 -->  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>abc</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  </web-app> |

1. 在src下新建springmvc.xml

|  |
| --- |
| xmlns:mvc=*"http://www.springframework.org/schema/mvc"*  *http://www.springframework.org/schema/mvc*  *http://www.springframework.org/schema/mvc/spring-mvc.xsd*  <!--扫描注解 -->  <context:component-scan base-package=*"com.jun.controller"/*>    <!-- 注解驱动 -->  <!-- org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping  org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter -->  <mvc:annotation-driven/>  <!--静态资源 -->  <mvc:resources location=*"/js/"* mapping=*"/js/\*\*"/*>  <mvc:resources location=*"/css/"* mapping=*"/css/\*\*"*>  <mvc:resources location=*"/images/"* mapping=*"/images/\*\*"/*>  </beans> |

1. Src下创建controller包，包下创建xxxController

|  |
| --- |
| @Controller  **public** **class** DemoController {  //地址栏输入路径  @RequestMapping("demo1")  **public** String demo1() {  System.***out***.println("执行控制器1");  **return** "demo1.jsp";  }  @RequestMapping("demo2")  **public** String demo2() {  System.***out***.println("执行控制器2");  **return** "demo2.jsp";  }  } |

### 字符编码过滤器

|  |
| --- |
| <!--字符编码过滤器配置 -->  <filter>  <filter-name>encoding</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>encoding</filter-name>  <url-pattern>\*</url-pattern>  </filter-mapping> |

### 传参

1. 把内容写到方法（HandleMethod）参数中，SpringMVC只要有这个内容，就会将其注入
   1. **基本数据类型**

默认保证参数名称和请求中传过来的名称相同；或者使用@RequestParam(value=”前端name名”)

@RequestParam(defaultValue=”2”):默认值为2

@RequestParam(require=true):传过来的参数为null时报错

* 1. **HandlerMathod中参数是对象类型**

1.2.1、请求参数名和对象中属性名对应

1.3、请求参数中包含多个同名参数（复选框）

1.3.1、使用@RequestParam(value=”前端name名”)List

1.4、请求参数中是 ： A.b 格式需要一个额外的类B，添 加A作为新建类B的属性，提供get/set方法。将B作为参数 传入

1.5、**restful传值方式**

在jsp中设置特定的格式

|  |
| --- |
| <a href=”demo/123/abc”>跳转</a> |

在控制器中

在@RequestMapping一定要和请求格式相同

{id1}中的id1是自定义的

@PathVariable：获取@RequestMapping中的内容，默认安装方法参数名称去寻找

|  |
| --- |
| @RequestMapping("demo3/{**id1**}/{**name**}")  **public** String demo3(@PathVariable String **name**, @PathVariable("**id1**") **int** age) |

### 跳转方式

1. 默认的跳转方式是 请求转发
2. 设置重定向： return “ redirect：…(资源路径) “

重定向： **地址栏地址改变**

1. 设置转发：添加 forword：…(资源路径) 或省略forward

### 视图解析器

1. springMVC会提供默认的视图解析器，程序员也可以自定义视图解析器

|  |
| --- |
| <!-- 自定义视图解析器 -->  <bean id=*"viewResolver"* class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"prefix"* value=*"/"*/>  <property name=*"suffix"* value=*".jsp"*/>  </bean> |

1. 如果不希望执行试图解析器，则在方法返回值前面添加

forward: 或者redirect:

### @ResponseBody

1. 在方法上，只有@RequestMapping时，无论方法返回值是什么，都认为需要跳转
2. 在方法上添加@ResponseBody后，如果返回值**满足key-value形式（对象或map）**会把响应头设置为：application/json;charser=utf-8，之后会以流的方式响应给客户端
3. 如果不满足key-value（返回值为**string**），会自动把响应头设置为text/html，此时可能出现中文乱码。解决：在**@ResponseMapping(value=”...”,produces=”text/html”)**

## DAY2

## 文件下载

浏览器访问资源时，相应的响应头如果没有设置Content-disposition，浏览器默认按照inline（能显示就显示，不能显示就下载）处理。只需要修改相应头中 **Context-Disposition=“attachment;filename=文件名”。**

**attachment：**以附件形式下载

下载实现步骤：

1. 导入jar：commons-fileupload/io
2. Jsp中添加下载超链接，并在springmvc.xml中放行静态资源

|  |
| --- |
| <a href=*"download?fileName=notes.txt"*>下载</a>  <mvc:resources location=*"/files/"* mapping=*"/files/\*\*"*/> |

1. 编写控制器方法

|  |
| --- |
| **public** **void** download(String fileName, HttpServletRequest req, HttpServletResponse res) **throws** IOException {  //设置响应流中文件进行下载  **res.setHeader("Content-Disposition", "attachment;filename=" + fileName);**  //把二进制流放入到响应体中  ServletOutputStream os = res.getOutputStream();  String realPath = req.getServletContext().getRealPath("files");  File file = **new** File(realPath, fileName);  **byte**[] bs = FileUtils.*readFileToByteArray*(file);  os.write(bs);  os.flush();  os.close(); |

## 文件上传

1. 基于commons-fileupload.jar
2. MultipartResovler：把客户端上传的文件流转换成MultipartFile封装类；通过MultipleFile封装类获取到文件流
3. 表单数据类型分类

在form的enctype属性控制表单类型：

a)默认值：**application/x-www-form-urlencoded***（少了文字信息）*

*b) text/plain （大量文字信息：邮件、论文）*

*c) multipart/form-data(表单中包含二进制文件)*

实现步骤：

1. Jsp页面：**enctype**=*"multipart/form-data"* **method**=*"post"*

|  |
| --- |
| <form action=*"upload"* enctype=*"multipart/form-data"* method=*"post"*>  姓名：<input type=*"text"* name=*"name"*>  文件：<input type=*"file"* name=*"file"*>  <input type=*"submit"* value=*"提交"*>  </form> |

1. 配置springmvc.xml

Multipart解析器

异常解析器:**运行过程中出现key中的异常，则跳转到指定界面**

|  |
| --- |
| <!--Multipart解析器 -->  <bean id=*"multipartResolver"* class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>  <!-- 设置上传最大为50kb -->  <!-- <property name="maxUploadSize" value="50"/> -->  </bean>  <!--异常解析器 -->  <bean id=*"exceptionResolver"* class=*"org.springframework.web.servlet.handler.SimpleMappingExceptionResolver"*>  <property name=*"exceptionMappings"*>  <props>  <prop key=*"org.springframework.web.multipart.MaxUploadSizeExceededException"*>/error.jsp</prop>  </props>  </property>  </bean> |

1. 控制器类： MultipartFile 的对象名要和表单中file的name相同

|  |
| --- |
| @RequestMapping("upload")  **public** String upload(MultipartFile file, String name) **throws** IOException {  System.***out***.println(name);  //获取文件名  String filename = file.getOriginalFilename();  String suffix = filename.substring(filename.lastIndexOf("."));  String uuid = UUID.*randomUUID*().toString();  FileUtils.*copyInputStreamToFile*(file.getInputStream(), **new** File("E:/" + uuid + suffix));  **return** "demo2";  } |

## DAY3

## 自定义拦截器

发送**请求**时被拦截器拦截，在**控制器**前后添加额外功能。

1. SpringMVC拦截器和过滤器Filter的区别

* 拦截器只能拦截Controller
* Filter可以拦截任何请求

1. 自定义拦截器实现：
   1. 新建实现HandleInterceptor的类

（1）：**preHandle：**在进入控制器之前执行，如果返回值是false则阻止进入控制器

（2）：**postHandle：**控制器执行完成，进入jsp之前执行具备视图和传值功能，敏感词过滤，日志记录

（3）：**afterCompletion：**jsp执行完成之后执行，记录执行过程中出现的异常到日志中

|  |
| --- |
| **public** **class** LoginInterceptor **implements** HandlerInterceptor {  //在进入控制器之前执行，如果返回值是false则阻止进入控制器  //控制代码  @Override  **public** **boolean** preHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2) **throws** Exception {  **return** **false**;  }  //控制器执行完成，进入jsp之前执行  //具备视图和传值功能  //敏感词过滤，日志记录  @Override  **public** **void** postHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, ModelAndView arg3)  **throws** Exception {  }  //jsp执行完成之后执行  //记录执行过程中出现的异常到日志中  @Override  **public** **void** afterCompletion(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, Exception arg3)  **throws** Exception { } |

* 1. 在springmvc.xml配置拦截器
     + - 拦截所有控制器

|  |
| --- |
| <!-- 所有的controller都被拦截 -->  <mvc:interceptors>  <bean class=*"com.jun.interceptor.LoginInterceptor"*/>  </mvc:interceptors> |

* + - * 拦截指定控制器

|  |
| --- |
| <!-- 特定的controller被拦截 -->  <mvc:interceptors>  <mvc:interceptor>  <mvc:mapping path=*"demo1"*/>  <mvc:mapping path=*"demo2"*/>  <mvc:mapping path=*"demo3"*/>  <bean class=*"com.jun.interceptor.LoginInterceptor"*/>  </mvc:interceptor>  </mvc:interceptors> |

## 拦截器栈

当多个拦截器同时生效时，组成了拦截器栈；**先进后出，执行顺序和配置顺序有关**

**谁在前先执行谁**

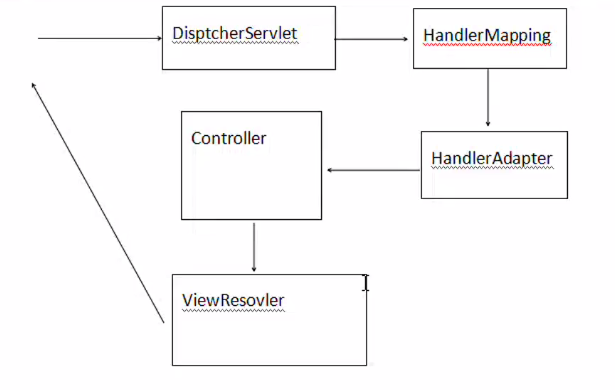
**preHandle(A)-->preHandle(B)-->控制器-->**

**postHandle(B) -->postHandle(A)-->Jsp-->**

**afterCompletion(B) -->afterCompletion(A)**

|  |
| --- |
| <mvc:interceptors>  <bean class=*"com.jun.interceptor.LoginInterceptorA"*/>  <bean class=*"com.jun.interceptor.LoginInterceptor****B****"*/>  </mvc:interceptors> |

## SpringMVC运行原理



如果在web.xml中设置DispatcherServlet的<url-parrtern>为 / 时，用户发起请求到控制器，首先会执行DispatcherServlet。由DispatcherServlet调用HandleMapping的DefaultAnnotationHandlerAdapt解析URL，解析后调用HandleAdapter组件的AnnotationMethodHandlerAdapter调用Controller中的HandlerMethod。当HandlerMethod执行完成之后会返回View，会被ViewResovler进行视图解析，解析后调用jsp对应的 .class文件并运行，最终把运行 .class文件的结果响应给客户端