# 5.SpringMVC

## 5.1.SpringMVC概述

springMVC介绍：是前端控制框架，主要用来负责与页面的交互。是spring提供的，基于MVC设计思想的框架

SpringMVC流程图：

计算机生成了可选文字:
价嵘
医
‘、
医
V1ew
视图渲染
10将model填充到view中展现
1．发送request尸－
2．请求执行
冬目巨
前端控制器
客户端
D1spatherservle
t
3．返回
处理器吸肠尹器
HandlerMappin
g
根据JRL查找
handler
hand!er
请求执行
9返回view
8．解析view
7返
回
mod
e}an
d
Vie
视图解析器
V1eWFe50IVeF
5执行
HandlerAdapter
处理器适配器
H。ndler处理器
，一．「J．一
b卫连巴」
mode!AndVie
W

组件介绍：

1．DispatcherServlet（前端控制器）

作用：接收请求，响应结果，相当于转发器

有了DispatcherServlet就减少了其他组件直接的耦合度。

2.HandlerMapping（处理器映射器）

作用：根据请求的url来查找handler。

3.HandlerAdapter（处理器适配器）

作用：调用适当的处理器去执行请求。

4.Handler（处理器）

作用：执行请求。

5.ViewResolver（视图解析器）

作用：进行解析，根据逻辑视图名解析成真正的视图（view）

6.view（视图）

作用：实现真正的视图

步骤：

1．客户端发送请求，到前端控制器

2．前端控制器请求处理器映射器进行解析，根据URL来匹配我们的处理器

3．处理器映射器向前端控制器返回匹配到的处理器。

4．前端控制器调用处理器适配器去执行处理器。

5．处理器适配器调用适当的处理器去执行请求。

6．处理器执行完成，向适配器返回ModelAndView（包含了我们需要的对象信息和页面名称）。

7．处理器适配器向前端控制器返回ModelAndView。

8．前端控制器调用视图解析器进行视图解析，根据逻辑视图名解析成真正的视图（jsp）。

9．视图解析器向前端控制器返回view（真实路径）。

10．前端控制器将model的数据填充到页面中（将model数据填充到request域中）

11．向用户响应结果

**springmvc其实主要就学两个东西：1.配置文件。2.如何传值**

## 5.2.SpringMVC入门案例

1. 导包（课前资料）
2. 在web-inf下创建一个hello.jsp。
3. **在web.xml中**配置前端控制器（dispatcherServlet，其实就是配置一个servlet）

<servlet>

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

<servlet-class>（按ctrl+shift+t，搜dispatcherServlet，可以快速定位我们需要的类，不用看源码，在下面第二行public中可以找到类）org.springframework.web.servlet.DispatcherServlet</servlet-class>

</servlet>

<servlet-mapping>

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

<url-pattern>/</url-pattern>

</servlet-mapping>

说明：\*.do和\*.action是拦截后缀名为.do和.action结尾的请求。

**4．在web-inf下编写springMVC的核心配置文件**，默认名称为springmvc-servlet.xml（为什么是这个名字呢？因为是servlet-name名称+-servlet，为什么要+servlet呢？找dispatcherServlet源码，找到父类frameWorkServlet）

记得取资料

（创建完以后，我们是不是要导入头文件，在我们的课前资料里面已经准备好了，大家直接拷贝过来就ok了）

先编写一个controller，新建cotroller包（web层），建一个项目，比如HelloController

（Springmvc是spring提供的组件，spring如何管理对象？bean标签）

<!-- 配置映射关系 -->（从url映射到controller）

<bean name=*"/hello.action"* class=*"controller.HelloController"*></bean>

（但是这样非常麻烦，每一个url请求，就要写这样的一行配置，所以，如果我们可以做到一劳永逸是最好的，对吧，所以，怎么做才能一劳永逸呢？如果我们spring将所有的controller都管理起来，在controller上写上对应的url的注解，是不是就可以搞定了，所以，我们这里配置，就两行，一行是管理controller，一行是开启注解。）

<!--开启包扫描 -->

<context:component-scan base-package="controller"/>

<!--开启SpringMVC注解 -->

<mvc:annotation-driven/>

5．配置视图解析器，在springmvc-servlet.xml中添加:

<!-- 配置视图解析器

内部资源视图解析器

view对象中的值：hello

拼接页面的真实路径/WEB-INF/hello.jsp

-->

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

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

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

</bean>

（配置完毕，但是controller和jsp都是空的吧，无法展示页面）

6．在HelloController中，添加代码，转向jsp

@Controller

public class HelloController{

@RequestMapping(value="toHello ")

model.addAttribute("msg", "Hello SpringMVC ");

return "hello"; //返回的是页面的名称view

}

7．在hello.jsp中获取数据

<body>

<h1>我来了</h1>

<h3>${msg}</h3>

</body>

8．测试

9．将springmvc-servlet.xml放在src下

（为什么要放到src下？方便java代码的调用。为什么不把web.xml也放web-inf下？因为程序默认在web-inf下寻找配置文件，需要一个入口。）

再次测试，发现报500，找不到配置文件，因为默认程序扫描的是web-inf目录，所以需要在web.xml中配置：

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

<init-param>

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

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

</init-param>

</servlet>

总结：只有三处配置，而且只需要第一次配置好就可以了！！！

## 5.3.如何传值

**1.普通传值：（直接在参数列表中传入。）**

1.1.将课前资料中的addUser.jsp拷贝到web-inf下。

1.2.在controller包下创建RegistController，并加上@Controller注解

1.3.在RegistController中添加转向代码和传值代码

//转向页面

@RequestMapping("toRegist")

**public** String toRegist(){

**return** "addUser";

}

/\*

\* 获取参数，并将参数放入model中，转向页面

\*/

@RequestMapping("addUser")

**public** String regist(String username,String password,**int** age,String email,Model model){

model.addAttribute("msg","用户名："+username+"密码："+password+"年龄："+age+"email:"+email);

**return** "userShow";

}

1.4.在web-inf下创建userShow.jsp，并获取数据

<body>

<h1>${msg }</h1>

</body>

**注意，参数中的名字必须与页面中name属性的名字相同。**

**2.为对象赋值。（直接在参数列表中传入对象）**

2.1.创建User类，并添加属性，添加get、set和toString方法，

**public** **class** User {

**private** String username;

**private** String password;

**private** Integer age;

**private** String email;

2.2. 在controller中直接在参数中传入User对象

@RequestMapping("/addUser")

**public** String addUser(User user,Model model){

2.3.对象中有引用

**private** Dog dog;

依然可以直接赋值，但是在jsp中需要：

<tr>

<td>狗名:</td>

<td><input name=*"dog.name"* type=*"text"*/></td>

</tr>

注意：页面中的name名称必须与实体类中set方法的名称相同。

**3.复选框（直接在参数列表中传入。）**

在addUser.jsp中添加如下复选框代码

<tr>

<td>爱好:</td>

<td><input name=*"like"* type=*"checkbox"* value=*"敲代码"*/>敲代码

<input name=*"like"* type=*"checkbox"* value=*"调bug"*/>调bug

<input name=*"like"* type=*"checkbox"* value=*"砸键盘"*/>砸键盘

<input name=*"like"* type=*"checkbox"* value=*"打自己"*/>打自己

<input name=*"like"* type=*"checkbox"* value=*"灵感来了"*/>灵感来了

</td>

</tr>

直接在controller参数列表中获取String，发现是用“，”隔开的字符串。

**public** String regist(User user,String like,Model model){

直接在controller参数列表中写String[]，也可以匹配，按需选择。

**public** String regist(User user,String[] like,Model model){

**参数乱码问题：**

只需在web.xml中配置即可：

<!-- 定义过滤器 -->

<filter>

<filter-name>encodingFilter</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>encodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

< url-pattern > / </ url-pattern >   不会匹配到\*.jsp，即：\*.jsp不会进入spring的 DispatcherServlet类 。  
< url-pattern > /\* </ url-pattern > 会匹配\*.jsp，会出现返回jsp视图时再次进入spring的DispatcherServlet 类，导致找不到对应的controller所以报404错。

提交参数名称与接收参数名称不同：

需要在接收参数前加@RequestParam注解，如下代码：

/\*

\*@RequestParam(value="like",defaultValue="放弃治疗",required=false)

\* 此注解解决提交参数与接收参数名称不一致问题，其中：

\* value：是提交参数名称

\* defaultValue：提交为空时，默认值

\* required（true/false）：是否必须提交参数

\*/

//获取参数，并将参数放入model中，转向页面

@RequestMapping("addUser")

**public** String regist(User user,@RequestParam(value="like",defaultValue="放弃治疗",required=**false**) String[] likes,Model model){

**4.日期**

* 1. 在addUser.jsp页面中加入生日输入框：

<tr>

<td>生日:</td>

<td><input name="birthday" type="*datetime*"/></td>

</tr>

5.2．在User类中添加属性：**private** Date birthday; 并添加get and set 方法与toString。

测试，发现如果输入2017/1/12没问题，如果输入2017-1-12则报400，说明提交日期格式时，默认格式为“/”分隔

5.3．在RegistController中加入日期格式工具：

//初始化绑定

@InitBinder

public void InitBinder (ServletRequestDataBinder binder){

binder.registerCustomEditor(

java.util.Date.class,

new CustomDateEditor(new SimpleDateFormat("yyyy-MM-dd"), true));

}

**5.文件上传**

文件上传总共需要三步：

6.1.配置文件上传解析器，在springmvc-servlet.xml中加入如下代码：

<!--配置文件上传解析器,id的名称必须为:multipartResolver -->

<bean id="multipartResolver"

class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

<property name="maxUploadSize" value="10485760"></property>

</bean>

6.2.在addUser.jsp中添加上传文件代码，并在form表单中加入文件上传解析器：

<form action=*"*${pageContext.request.contextPath}*/addUser"* method=*"POST"* **enctype=*"multipart/form-data"*>**

<tr>

<td>头像上传:</td>

<td><input name=*"photo"* type=*"file"*/></td>

</tr>

6.3.在RegistController中接收文件并输出到本地文件：

@RequestMapping("addUser")

**public** String regist(User user,@RequestParam(value="like",defaultValue="放弃治疗",required=**false**) String[] likes,MultipartFile photo,Model model) **throws** IOException{

model.addAttribute("msg",user+",爱好很多哦:"+Arrays.*toString*(likes));

//原始方式传输

/\*FileOutputStream out = new FileOutputStream(new File("D:\\photo\\"+photo.getOriginalFilename()));

out.write(photo.getBytes());

out.close();\*/

//简便方式传输

FileUtils.*writeByteArrayToFile*(**new** File("D:\\photo\\"+photo.getOriginalFilename()), photo.getBytes());

**return** "userShow";

**6.RESTFul**

我们以前学的get提交方式：

URL: [http://localhost/SpringMVC1708/addUser?username=abc&age=22&email=abc@tedu.cn](http://localhost/SpringMVC1708/addUser?username=abc&password=123&age=22&email=abc@tedu.cn)

而RESTFul提交方式为：

URL:

<http://localhost/SpringMVC1708/addUser/abc/22/abc@tedu.cn>

RESTFul：传值时只传递具体的值，多个值之间用“/”分隔。

现在大部分网站使用的都是RESTFul形式， 此形式传递数据量小。

RESTFul实例：

6.1.创建一个controller：

@Controller

**public** **class** RestFulController {

//restFul方式接收参数。

@RequestMapping("restFul/{username}/{email}/{age}")

**public** String restFul(@PathVariable String username,@PathVariable **int** age,@PathVariable String email,Model model){

model.addAttribute("msg","restFul传过来的参数，用户名："+username+",邮箱："+email+",年龄:"+age);

**return** "hello";

}

}

注意：参数位置必须与url中的位置相对应

6.2.在浏览器中输入url：

<http://localhost/SpringMVC1707/restFul/abc/abc@tedu.cn/22>

7.SpringMVC中的重定向和转发

重定向redirect：1.浏览器地址栏发生改变

2.多次请求多次响应

3.不能传递参数

转发forward：1.浏览器地址栏不发生改变

2.一次请求一次响应

3.可以传递参数

以HelloController为例，添加代码：

//转发

/\*@RequestMapping("helloxxx")

public String sayhelloxxx(Model model){

model.addAttribute("msg","hello springMVC");

return "forward:toHello";

}\*/

//重定向

@RequestMapping("helloxxx")

**public** String sayhelloxxx(Model model){

model.addAttribute("msg","hello springMVC");

**return** "redirect:toHello";

}

删除hello页面转向的model

@RequestMapping("toHello")

**public** String sayhello(Model model){

**return** "hello";

}

其实我们发现我们之前直接return页面，就是转发。

转发可以直接访问wen-inf下的资源，如return "forward:/hello";

而重定向不可以。