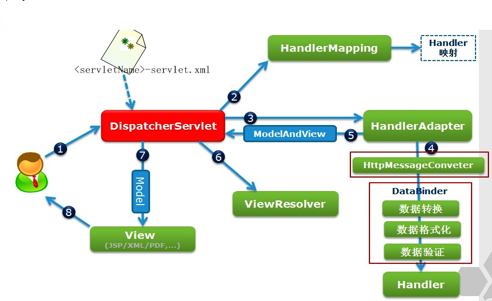
# 一 SpringMVC概述

## 1 SpringMVC架构

Spring Web MVC是一种基于Java的实现了Web MVC设计模式的请求驱动类型的轻量级Web框架，是Spring框架的一个模块。SpringMVC使用MVC架构模式的思想，将web层进行职责解耦，基于请求驱动指的就是使用请求-响应模型。

将其作为控制层使用

## 2 工作原理



（1） 用户发送请求到springmvc框架提供的DispatcherServlet 这个前端控制器

（2） 前端控制器会去找处理器映射器(HandlerMapping)，处理器映射器根据请求url找到具体的处理器，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 。

（3） 根据处理器映射器返回的处理器，DispatcherServlet 会找“合适”的处理器适配器(HandlerAdapter)

（4） 处理器适配器HandlerAdpater会去执行处理器(Handler开发的时候会被叫成Controller也叫后端控制器) 执行之前会有转换器、数据绑定、校验器等等完成上面这些才会去正在执行Handler

（5） 后端控制器Handler执行完成之后返回一个ModelAndView对象

（6） 处理器适配器HandlerAdpater会将这个ModelAndView返回前端控制器DispatcherServlet。前端控制器会将ModelAndView对象交给视图解析器ViewResolver。

（7） 视图解析器ViewResolver解析ModelAndView对象之后返回逻辑视图。

（8） 前端控制器DispatcherServlet对逻辑视图进行渲染(数据填充)之后返回真正的物理View并响应给浏览器。

请求如何给前端控制器？

这个应该在web.xml中进行部署描述

前端控制器如何根据请求信息选择页面控制器进行功能处理？

需要配置HandlerMapping进行映射

如何支持多种页面控制器呢？

配置HandlerAdapter从而支持多种类型的页面控制器

如何页面控制器如何使用业务对象？

利用Spring IoC容器的依赖注入功能

页面控制器如何返回模型数据？

使用ModelAndView返回

前端控制器如何根据页面控制器返回的逻辑视图名选择具体的视图进行渲染？

使用ViewResolver进行解析

不同的视图技术如何使用相应的模型数据？

因为Model是一个Map数据结构，很容易支持其他视图技术

## 3具体的核心开发步骤

1）DispatcherServlet在web.xml中的部署描述，从而拦截请求到的资源

2）HandlerMapping的配置，从而将请求映射到处理器

3）HandlerAdapter的配置，从而支持多种类型的处理器

4）ViewResolver的配置，从而将逻辑视图名解析为具体视图技术

5）处理器（页面控制器）的配置，从而进行功能处理，相当于控制层

# 二 SpringMVC入门

## 1 基本用法

1) 导入包

导入spring的基本包，

另外，还需要导入如下的jar包：

spring-aop-5.1.5.RELEASE.jar

spring-web-5.1.5.RELEASE.jar

spring-webmvc-5.1.5.RELEASE.jar

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>5.1.5.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.1.5.RELEASE</version>

</dependency>

如果需要输出日志，可以添加如下依赖：

<dependency>

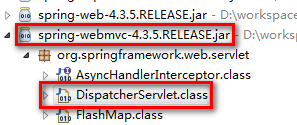
<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.2</version>

</dependency>

2) web.xml配置



<!—引入spring mvc的核心控制器 配置 -->

<servlet>

<servlet-name>DispatchServlet</servlet-name>

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

<!-- 指定加载哪个配置文件 -->

<init-param>

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

<param-value>classpath:spring-mvc.xml</param-value>

</init-param>

<!-- tomcat服务器启动时，创建servlet对象 -->

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

</servlet>

<servlet-mapping>

<servlet-name>DispatchServlet</servlet-name>

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

</servlet-mapping>

3) 控制类

实现Controller接口，重写handleRequest方法

**public** **class** HelloController **implements** Controller {

@Override

**public** ModelAndView handleRequest(HttpServletRequest arg0, HttpServletResponse arg1) **throws** Exception {

// **TODO** Auto-generated method stub

ModelAndView mv = **new** ModelAndView();

//向ModelAndView中写入数据

mv.addObject("name", "zhangsan");

//设置要跳转到的资源的名称

//mv.setViewName("/index.jsp");

//逻辑名称,需要配置视图解析器

mv.setViewName("index");

**return** mv;

}

}

注意：当视图解释器解析ModelAndVIew是，其中model本生就是一个Map的实现类的子类。视图解析器将model中的每个元素都通过request.setAttribute(name, value);添加request请求域中。这样就可以在JSP页面中通过EL表达式来获取对应的值

1. 配置文件

*引入命名空间*

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

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

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

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

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

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

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*

*http://www.springframework.org/schema/aop*

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

*http://www.springframework.org/schema/tx*

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

*http://www.springframework.org/schema/mvc*

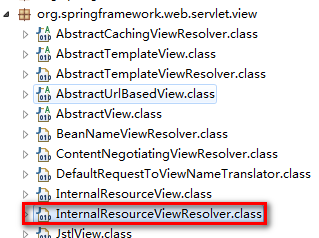
*<http://www.springframework.org/schema/mvc/spring-mvc.xsd>"*>

<!-- id 表示访问的url,使用id或者name都可以 -->

<bean id=*"/hello.action"* class=*"com.rr.controller.HelloController"*></bean>

## 2 视图解析器 ViewResolver

spring-webmvc的jar包下



<!-- 视图解析器 -->

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

<!-- 前缀 -->

<property name=*"prefix"* value=*"/"*></property>

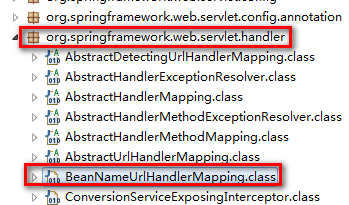
<!-- 后缀 -->

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

</bean>

## 3 映射器HandlerMapping

spring-webmvc的jar包下



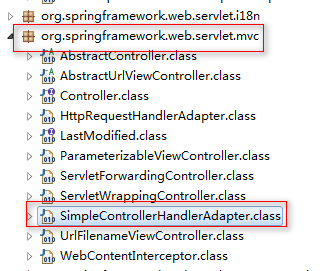
<!-- 默认的映射器，可以不写 -->

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

BeanNameUrlHandlerMapping将请求的资源（路径）直接映射到对应的Bean进行处理，即某个Controller

## 4 适配器 HandlerAdapter

spring-webmvc的jar包下



<!-- 默认的适配器，可以不写 -->

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

 通过该适配器，实现处理器（Controller）某个功能方法的调用，SimpleControllerHandlerAdapter将会调用处理器的handleRequest方法进行功能处理，该处理方法返回一个ModelAndView给DispatcherServlet

# 三 注解方式

## 常用注解：

@RequestMapping

用来处理请求地址映射的注解（将请求映射到对应的控制器方法中），可用于类或方法上。用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。类似的注解还有@GetMapping、@PostMapping、@PutMapping、@DeleteMapping、@PatchMapping

@RequestParam

用于将指定请求参数映射到方法的参数上

@SessionAttributes

在默认情况下，ModelMap中的属性作用域是request级别，如果希望在多个请求中共享ModelMap中的属性，可以将属性转存到session 中。通过该注解，允许我们有选择地指定 ModelMap 中的哪些属性需要转存到 session 中。

@ModelAttribute

可以应用在方法参数上或方法上。当修饰在方法参数上时，会将注解的参数对象添加到Model中；当用在请求处理方法上时会将该方法变成一个非请求处理的方法，但其它请求处理方法被调用时会首先调用该方法。

@PathVariable

用于将请求的URL中的模板变量映射到处理方法的参数上。

@ResponseBody

用于将控制层中的方法返回的对象，通过适当的HttpMessageConverter转换为指定格式后，写入到Response对象的数据区。一般转为json格式

@CookieValue

从Http请求头中获取指定的Cookie的值，

value：cookie的名称；required：是否必须

## 1 基本用法





<!-- 扫描注解 -->

<context:component-scan base-package=*"com.rr.anno"*></context:component-scan>

<!-- 处理@RequestMapping 的映射器 -->

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

<!-- 处理 @RequestMapping 的适配器-->

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

<!-- 使用下面的配置，可以不用写RequestMappingHandlerMapping和 RequestMappingHandlerAdapter-->

<mvc:annotation-driven />

关于<mvc:annotation-driven>:

该配置可以注册RequestMappingHandlerMapping和RequestMappingHandlerAdapter。如果不写，也不会影响资源的访问，这是因为spring4中，springmvc容器启动时会默认的加载DefaultAnnotationHandlerMapping和AnnotationMethodHandlerAdapter，但是这两个类已经不建议使用了。Spring5中默认就是使用RequestMappingHandlerMapping。另外，该配置还提供数据绑定支持，@NumberFormatannotation支持，@DateTimeFormat支持，@Valid支持，读写XML的支持（JAXB），读写JSON的支持（Jackson）等。

## 2 向界面返回数据

借助Model/ModelMap等对象

//http://localhost:8080/F15\_SpringMvc/user/world.action

@Controller //表示控制层

@RequestMapping("/user")

@SessionAttributes(value={"tid", "tname"})//可以向session中存数据

**public** **class** WorldController {

//Model是一个接口， 其实现类为ExtendedModelMap，该类继承了ModelMap类

//请求映射

@RequestMapping("/world.action")

**public** String annotest(Model mm){

//向Model对象中写数据

mm.addAttribute("name", "lisi");

//返回跳转的url

**return** "/index.jsp";

}

@RequestMapping("/session.action")

**public** String test(Model m){

//tid是@SessionAttributes中出现的key值，配合@SessionAttributes，

//然后通过向Model中写入数据，会将该值也写到session中

m.addAttribute("tid", "xxxxxxx");

**return** "redirect:/session.jsp";

}

@RequestMapping("/session2.action")

**public** String testSession(HttpSession session){

//通过HttpSession对象向session域中写入数据

session.setAttribute("tage", 20);

**return** "redirect:/session.jsp";

}

@RequestMapping("/context.action")

**public** String testContext(HttpServletRequest request){

//通过request获取servletcontext

request.getSession().getServletContext().setAttribute("context", "hahah");

**return** "redirect:/context.jsp";

}

}

## 3 转发/重定向

spring mvc 默认跳转方式为转发

使用forward和redirect前缀后，视图解析器不起作用

支持写法：

"forward:/index.jsp";

"redirect:/index.jsp";

其中，"/"相对于应用来说，另外，通过这两个前缀，还可以转发或重定向到其他的controller资源

//默认转发

@RequestMapping("/world1.action")

**public** String annotest(ModelMap mm){

//通过ModelMap向域中写数据

mm.addAttribute("name", "lisi1234");

//返回跳转的url

**return** "/index.jsp";

}

//转发

@RequestMapping("/forword.action")

**public** String forwordtest(ModelMap mm){

//通过ModelMap向域中写数据

mm.addAttribute("name", "lisi1234");

//返回跳转的url

**return** "forward:/index.jsp";

}

//重定向

@RequestMapping("/redirect.action")

**public** String redirecttest(ModelMap mm){

//通过ModelMap向域中写数据，request域

mm.addAttribute("name", "lisi1234");

//返回跳转的url

**return** "redirect:/index.jsp";

}

## 4 传递表单参数

1)直接接收

//直接传参，String username参数名与<input type="text" name="username"/>的name值必须一致

@RequestMapping("/register")

**public** String register(String username, String pwd, Model m){

System.***out***.println(username);

System.***out***.println(pwd);

m.addAttribute("name", username);

**return** "index";

}

//@RequestParam 表示请求参数的注解

//本例中，jsp页面的请求参数为user，将user的值赋值给username

@RequestMapping("/register1")

**public** String register1(@RequestParam("user") String username, String pwd, Model m){

System.***out***.println(username);

System.***out***.println(pwd);

m.addAttribute("name", username);

**return** "index";

}

//通过HttpServletRequest

@RequestMapping("/register2")

**public** String register2(HttpServletRequest request){

System.***out***.println(request.getParameter("user"));

System.***out***.println(request.getParameter("pwd"));

request.setAttribute("name", request.getParameter("user"));

**return** "index";

}

2)封装为对象

//通过对象传参

@RequestMapping("/register3")

**public** String register3(User user, ModelMap mm){

System.***out***.println(user.getUsername());

System.***out***.println(user.getPwd());

mm.addAttribute("name", user.getUsername());

**return** "index";

}

3)日期格式转换

a. 使用@DateTimeFormat注解

b. 使用@InitBinder注解

//日期类型转换器

@InitBinder

**public** **void** initBinder(WebDataBinder binder) {

binder.registerCustomEditor(

Date.**class**,

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

);

}

//method的值表示参数的提交方式

@RequestMapping(value="/register4", method=RequestMethod.***POST***)

**public** String register4(String name, Date date, ModelMap mm){

System.***out***.println(name);

System.***out***.println(date);

mm.addAttribute("name", name);

**return** "index";

}

//对象中有日期类型

@RequestMapping(value="/register5", method=RequestMethod.***POST***)

**public** String register5(User user, ModelMap mm){

System.***out***.println(user.getUsername());

System.***out***.println(user.getBirth());

mm.addAttribute("name", user.getUsername());

**return** "index";

}

4）post提交方式参数的中文乱码处理

<!-- post 方式的中文乱码解决 -->

<filter>

<filter-name>characterEncoding</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>utf8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>characterEncoding</filter-name>

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

</filter-mapping>

## 5 servlet中url-pattern配置

1）url-pattern中 / 与 /\* 区别

可以匹配的资源基本相同，比如 .js、.html、不带后缀的资源；

但是 / 对jsp资源不起作用，会直接跳转到jsp页面；/\* 会过滤jsp资源，跳转到servlet

如果使用的restful风格的url，访问的资源没有后缀，而且jsp需要直接访问，这个时候就需要配置为 /

2）静态资源配置

web.xml中：

<servlet>

<servlet-name>DispatchServlet</servlet-name>

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

<init-param>

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

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

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>DispatchServlet</servlet-name>

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

</servlet-mapping>

/的情况下，如果控制器的资源没有写后缀，则访问该资源时，可以匹配任意的后缀

@RequestMapping("/test2")

如下写法的路径都可以访问到该资源：

<http://localhost:8080/mvc01/user/test2.html>

<http://localhost:8080/mvc01/user/test2.do>

http://localhost:8080/mvc01/user/test2

spring的配置文件中：

<mvc:annotation-driven/>

<!--

方案1

Spring 会把一些找不到 controller 的请求，

放给默认的 servlet 容器来处理(比如tomcat交给名称为default的servlet)；从而能正常访问静态资源;

该方案下<mvc:annotation-driven/>必须写

将静态资源的处理经由Spring MVC框架交回Web应用服务器处理

<mvc:default-servlet-handler/>

-->

<!--

方案2

mapping是写在页面上的静态资源前缀，location 是指向的真正文件夹位置，用来定位静态资源;

<mvc:annotation-driven/>必须写

由Spring MVC框架自己处理静态资源

针对静态资源设置映射路径,通过映射的虚拟路径访问静态资源

-->

<mvc:resources location=*"/js/"* mapping=*"/js/\*\*"*/>

<mvc:resources location=*"/"* mapping=*"/\*\*"*/>

<mvc:resources location="/WEB-INF/static/" mapping="/static/\*\*"></mvc:resources>