## [17. Web MVC framework](http://docs.spring.io/spring-framework/docs/4.1.x/spring-framework-reference/htmlsingle/" \l "mvc)

### [17.1. Introduction to Spring Web MVC framework](http://docs.spring.io/spring-framework/docs/4.1.x/spring-framework-reference/htmlsingle/#mvc-introduction)（关于SpringMVC的介绍）

#### 17.1.1 Features of Spring Web MVC

Spring MVC框架主要围绕着DispatcherServlet设计而来，默认的请求句柄是通过@Controller和@RequestMapping注解，提供多元化的处理方法。通过Spring3.0介绍，@Controller原理允许你RESTful的WEB站点和应用，通过@PathVariable注解和其它的另外属性。

### 17.2 The DispatcherServlet

SpringMVC框架跟其它框架类似，是以请求为驱动而设计，分发请求至不同Controller。

接下来简单描述下SpringMVC DispatcherServlet请求工作流描述。DispatcherServlet的设计模式是以“Front Controller”为设计理念。



上述的请求工作流是站在高角度的。

DispatcherServlet是真正的Servlet，是继承于HttpServlet，被申明在WEB.XML文件中

<web-app>

<servlet>

<servlet-name>example</servlet-name>

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

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

</servlet>

<servlet-mapping>

<servlet-name>example</servlet-name>

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

</servlet-mapping>

</web-app>

上述的web.xml例子的配置，实际上DispatcherServlet会处理/example开始的请求访问，你也可以不同xml的配置，而是采纳动态方法。

**public** **class** MyWebApplicationInitializer **implements** WebApplicationInitializer {

*@Override*

**public** **void** onStartup(ServletContext container) {

ServletRegistration.Dynamic registration = container.addServlet("dispatcher", **new** DispatcherServlet());

registration.setLoadOnStartup(1);

registration.addMapping("/example/\*");

}

}



在上述的DispatcherServlet初始化中，Spring MVC为寻找一个【servlet-name】-servlet.xml在WEB-INF目录下

<web-app>

<servlet>

<servlet-name>**golfing**</servlet-name>

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

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

</servlet>

<servlet-mapping>

<servlet-name>**golfing**</servlet-name>

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

</servlet-mapping>

</web-app>

这个时候你就需要配置这么一个文件/WEB-INF/golfing-servlet.xml，能够包含所有的Spring WEB MVC的Bean，你也能在外面配置，以参数的形式加载进来。

<web-app>

<context-param>

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

<param-value>/WEB-INF/root-context.xml</param-value>

</context-param>

<servlet>

<servlet-name>dispatcher</servlet-name>

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

<init-param>

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

<param-value></param-value>

</init-param>

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

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

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

</servlet-mapping>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

</web-app>

#### 17.2.1 Special Bean Types In the WebApplicationContext

介绍下特殊的在WebApplicationContext的Bean

|  |  |
| --- | --- |
| Bean类型 | 说明 |
| [HandlerMapping](http://docs.spring.io/spring-framework/docs/4.1.x/spring-framework-reference/htmlsingle/#mvc-handlermapping) |  |
| HandlerAdapter |  |
| [HandlerExceptionResolver](http://docs.spring.io/spring-framework/docs/4.1.x/spring-framework-reference/htmlsingle/#mvc-exceptionhandlers) |  |
| [ViewResolver](http://docs.spring.io/spring-framework/docs/4.1.x/spring-framework-reference/htmlsingle/#mvc-viewresolver) |  |
| [LocaleResolver](http://docs.spring.io/spring-framework/docs/4.1.x/spring-framework-reference/htmlsingle/#mvc-localeresolver) &[LocaleContextResolver](http://docs.spring.io/spring-framework/docs/4.1.x/spring-framework-reference/htmlsingle/#mvc-timezone) |  |
| [ThemeResolver](http://docs.spring.io/spring-framework/docs/4.1.x/spring-framework-reference/htmlsingle/#mvc-themeresolver) |  |
| [MultipartResolver](http://docs.spring.io/spring-framework/docs/4.1.x/spring-framework-reference/htmlsingle/#mvc-multipart) |  |
| [FlashMapManager](http://docs.spring.io/spring-framework/docs/4.1.x/spring-framework-reference/htmlsingle/#mvc-flash-attributes) |  |
|  |  |

### 17.3 Implementing Controllers

Controllers提供了访问应用的接口，Controllers拦截用户的输入，然后转换为model，通过view展示。Spring2.5介绍了MVC Controller的注解方式， @RequestMapping,@RequestParam, @ModelAttribute。

*@Controller*

**public** **class** HelloWorldController {

*@RequestMapping("/helloWorld")*

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

model.addAttribute("message", "Hello World!");

**return** "helloWorld";

}

}

就像你看到的样子，通过@Controller and @RequestMapping能够很灵活地进行方法名的命令。

#### 17.3.1 Defining a controller with @Controller

为了能够自动检测到注解的controller，你需要配置如下信息

<?xml version="1.0" encoding="UTF-8"?>

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

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

xmlns:p="http://www.springframework.org/schema/p"

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

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">

<context:component-scan base-package="org.springframework.samples.petclinic.web"/>

*<!-- ... -->*

</beans>

#### 17.3.2 Mapping Requests With @RequestMapping

你可以通过RequestMapping 注解，映射到希望的句柄的方法上，通过如下信息配置

*@Controller*

**@RequestMapping("/appointments")**

**public** **class** AppointmentsController {

**private** **final** AppointmentBook appointmentBook;

*@Autowired*

**public** AppointmentsController(AppointmentBook appointmentBook) {

**this**.appointmentBook = appointmentBook;

}

**@RequestMapping(method = RequestMethod.GET)**

**public** Map<String, Appointment> get() {

**return** appointmentBook.getAppointmentsForToday();

}

**@RequestMapping(value="/{day}", method = RequestMethod.GET)**

**public** Map<String, Appointment> getForDay(*@PathVariable* *@DateTimeFormat(iso=ISO.DATE)* Date day, Model model) {

**return** appointmentBook.getAppointmentsForDay(day);

}

**@RequestMapping(value="/new", method = RequestMethod.GET)**

**public** AppointmentForm getNewForm() {

**return** **new** AppointmentForm();

}

**@RequestMapping(method = RequestMethod.POST)**

**public** String add(*@Valid* AppointmentForm appointment, BindingResult result) {

**if** (result.hasErrors()) {

**return** "appointments/new";

}

appointmentBook.addAppointment(appointment);

**return** "redirect:/appointments";

}

}

##### Matrix Variables

矩阵参数出现在path的变量上，每一个矩阵参数都是通过“;”分割开来，比如"/cars;color=red;year=2012"，多个参数可能被分割成"color=red,green,blue"或者重复"color=red;color=green;color=blue"

比如

*// GET /pets/42;q=11;r=22*

*@RequestMapping(value = "/pets/{petId}", method = RequestMethod.GET)*

**public** **void** findPet(*@PathVariable* String petId, *@MatrixVariable* **int** q) {

*// petId == 42*

*// q == 11*

}

*// GET /owners/42;q=11/pets/21;q=22*

*@RequestMapping(value = "/owners/{ownerId}/pets/{petId}", method = RequestMethod.GET)*

**public** **void** findPet(

*@MatrixVariable(value="q", pathVar="ownerId")* **int** q1,

*@MatrixVariable(value="q", pathVar="petId")* **int** q2) {

*// q1 == 11*

*// q2 == 22*

}

默认值

*// GET /pets/42*

*@RequestMapping(value = "/pets/{petId}", method = RequestMethod.GET)*

**public** **void** findPet(*@MatrixVariable(required=false, defaultValue="1")* **int** q) {

*// q == 1*

}

所有矩阵参数可能存放在map中

*// GET /owners/42;q=11;r=12/pets/21;q=22;s=23*

*@RequestMapping(value = "/owners/{ownerId}/pets/{petId}", method = RequestMethod.GET)*

**public** **void** findPet(

*@MatrixVariable* Map<String, String> matrixVars,

*@MatrixVariable(pathVar="petId"")* Map<String, String> petMatrixVars) {

*// matrixVars: ["q" : [11,22], "r" : 12, "s" : 23]*

*// petMatrixVars: ["q" : 11, "s" : 23]*

}