# @Controller和@RequestMapping用法和其他常用注解

# 一、简介

         在SpringMVC 中，控制器Controller 负责处理由DispatcherServlet 分发的请求，它把用户请求的数据经过业务处理层处理之后封装成一个Model ，然后再把该Model 返回给对应的View 进行展示。在SpringMVC 中提供了一个非常简便的定义Controller 的方法，你无需继承特定的类或实现特定的接口，只需使用@Controller 标记一个类是Controller ，然后使用@RequestMapping 和@RequestParam 等一些注解用以定义URL 请求和Controller 方法之间的映射，这样的Controller 就能被外界访问到。此外Controller 不会直接依赖于HttpServletRequest 和HttpServletResponse 等HttpServlet 对象，它们可以通过Controller 的方法参数灵活的获取到。为了先对Controller 有一个初步的印象，以下先定义一个简单的Controller ：

例1:

@Controller

public class MyController {

@RequestMapping ( "/showView" )

public ModelAndView showView() {

ModelAndView modelAndView = new ModelAndView();

modelAndView.setViewName( "viewName" );

modelAndView.addObject( " 需要放到 model 中的属性名称 " , " 对应的属性值，它是一个对象 " );

return modelAndView;

}

}

在上面的示例中，@Controller 是标记在类MyController 上面的，所以类MyController 就是一个SpringMVC Controller 对象了，然后使用@RequestMapping(“/showView”) 标记在Controller 方法上，表示当请求/showView.do 的时候访问的是MyController 的showView 方法，该方法返回了一个包括Model 和View 的ModelAndView 对象。这些在后续都将会详细介绍。

# 二、使用 @Controller 定义一个 Controller 控制器

例1所示@Controller 用于标记在一个类上，使用它标记的类就是一个SpringMVC Controller 对象。分发处理器将会扫描使用了该注解的类的方法，并检测该方法是否使用了@RequestMapping 注解。@Controller 只是定义了一个控制器类，而使用@RequestMapping 注解的方法才是真正处理请求的处理器，这个接下来就会讲到。

   单单使用@Controller 标记在一个类上还不能真正意义上的说它就是SpringMVC 的一个控制器类，因为这个时候Spring 还不认识它。那么要如何做Spring 才能认识它呢？这个时候就需要我们把这个控制器类交给Spring 来管理。

这个时候有两种方式可以把MyController 交给Spring 管理，好让它能够识别我们标记的@Controller 。

   第一种方式是在SpringMVC 的配置文件中定义MyController 的bean 对象。

<bean class="com.host.app.web.controller.MyController"/>

   第二种方式是在SpringMVC 的配置文件中告诉Spring 该到哪里去找标记为@Controller 的Controller 控制器。

< context:component-scan base-package = "com.host.app.web.controller" >

< context:exclude-filter type = "annotation"

expression = "org.springframework.stereotype.Service" />

</ context:component-scan >

注：

       上面 context:exclude-filter 标注的是不扫描 @Service 标注的类

# 三、使用 @RequestMapping 来映射 Request 请求与处理器

例1可以使用@RequestMapping 来映射URL 到控制器类，或者是到Controller 控制器的处理方法上。当@RequestMapping 标记在Controller 类上的时候，里面使用@RequestMapping 标记的方法的请求地址都是相对于类上的@RequestMapping 而言的；当Controller 类上没有标记@RequestMapping 注解时，方法上的@RequestMapping 都是绝对路径。这种绝对路径和相对路径所组合成的最终路径都是相对于根路径“/ ”而言的。

在这个控制器中，因为MyController 没有被@RequestMapping 标记，所以当需要访问到里面使用了@RequestMapping 标记的showView 方法时，就是使用的绝对路径/showView.do 请求就可以了。

例2

@Controller

@RequestMapping ( "/test" )

public class MyController {

@RequestMapping ( "/showView" )

public ModelAndView showView() {

ModelAndView modelAndView = new ModelAndView();

modelAndView.setViewName( "viewName" );

modelAndView.addObject( " 需要放到 model 中的属性名称 " , " 对应的属性值，它是一个对象 " );

return modelAndView;

}

}

例2是在控制器上加了@RequestMapping 注解，所以当需要访问到里面使用了@RequestMapping 标记的方法showView() 的时候就需要使用showView 方法上@RequestMapping 相对于控制器MyController上@RequestMapping 的地址，即/test/showView.do 。

## （一）使用 URI 模板

URI 模板就是在URI 中给定一个变量，然后在映射的时候动态的给该变量赋值。如URI 模板http://localhost/app/{variable1}/index.html ，这个模板里面包含一个变量variable1 ，那么当我们请求http://localhost/app/hello/index.html 的时候，该URL 就跟模板相匹配，只是把模板中的variable1 用hello 来取代。在SpringMVC 中，这种取代模板中定义的变量的值也可以给处理器方法使用，这样我们就可以非常方便的实现URL 的RestFul 风格。这个变量在SpringMVC 中是使用@PathVariable 来标记的。在SpringMVC 中，我们可以使用@PathVariable 来标记一个Controller 的处理方法参数，表示该参数的值将使用URI 模板中对应的变量的值来赋值。

例3

[复制代码](javascript:void(0);)

复制代码

@Controller

@RequestMapping ( "/test/{variable1}" )

public class MyController {

@RequestMapping ( "/showView/{variable2}" )

public ModelAndView showView( @PathVariable String variable1, @PathVariable ( "variable2" ) int variable2) {

ModelAndView modelAndView = new ModelAndView();

modelAndView.setViewName( "viewName" );

modelAndView.addObject( " 需要放到 model 中的属性名称 " , " 对应的属性值，它是一个对象 " );

return modelAndView;

}

}

复制代码

[复制代码](javascript:void(0);)

在例3的代码中我们定义了两个URI 变量，一个是控制器类上的variable1 ，一个是showView 方法上的variable2 ，然后在showView 方法的参数里面使用@PathVariable 标记使用了这两个变量。所以当我们使用/test/hello/showView/2.do 来请求的时候就可以访问到MyController 的showView 方法，这个时候variable1 就被赋予值hello ，variable2 就被赋予值2 ，然后我们在showView 方法参数里面标注了参数variable1 和variable2 是来自访问路径的path 变量，这样方法参数variable1 和variable2 就被分别赋予hello 和2 。方法参数variable1 是定义为String 类型，variable2 是定义为int 类型，像这种简单类型在进行赋值的时候Spring 是会帮我们自动转换的，关于复杂类型该如何来转换在后续内容中将会讲到。

   在上面的代码中我们可以看到在标记variable1 为path 变量的时候我们使用的是@PathVariable ，而在标记variable2 的时候使用的是@PathVariable(“variable2”) 。这两者有什么区别呢？第一种情况就默认去URI 模板中找跟参数名相同的变量，但是这种情况只有在使用debug 模式进行编译的时候才可以，而第二种情况是明确规定使用的就是URI 模板中的variable2 变量。当不是使用debug 模式进行编译，或者是所需要使用的变量名跟参数名不相同的时候，就要使用第二种方式明确指出使用的是URI 模板中的哪个变量。

   除了在请求路径中使用URI 模板，定义变量之外，@RequestMapping 中还支持通配符“\* ”。如下面的代码我就可以使用/myTest/whatever/wildcard.do 访问到Controller 的testWildcard 方法。

[复制代码](javascript:void(0);)

复制代码

@Controller

@RequestMapping ( "/myTest" )

public class MyController {

@RequestMapping ( "\*/wildcard" )

public String testWildcard() {

System. out .println( "wildcard------------" );

return "wildcard" ;

}

}

复制代码

[复制代码](javascript:void(0);)

## （二）使用 @RequestParam 绑定 HttpServletRequest 请求参数到控制器方法参数

例4

@RequestMapping ( "requestParam" )

public String testRequestParam( @RequestParam(required=false) String name, @RequestParam ( "age" ) int age) {

return "requestParam" ;

}

在上面代码中利用@RequestParam 从HttpServletRequest 中绑定了参数name 到控制器方法参数name ，绑定了参数age 到控制器方法参数age 。值得注意的是和@PathVariable 一样，当你没有明确指定从request 中取哪个参数时，Spring 在代码是debug 编译的情况下会默认取更方法参数同名的参数，如果不是debug 编译的就会报错。此外，当需要从request 中绑定的参数和方法的参数名不相同的时候，也需要在@RequestParam 中明确指出是要绑定哪个参数。在上面的代码中如果我访问/requestParam.do?name=hello&age=1 则Spring 将会把request请求参数name 的值hello 赋给对应的处理方法参数name ，把参数age 的值1 赋给对应的处理方法参数age 。

在@RequestParam 中除了指定绑定哪个参数的属性value 之外，还有一个属性required ，它表示所指定的参数是否必须在request 属性中存在，默认是true ，表示必须存在，当不存在时就会报错。在上面代码中我们指定了参数name 的required 的属性为false ，而没有指定age 的required 属性，这时候如果我们访问/requestParam.do而没有传递参数的时候，系统就会抛出异常，因为age 参数是必须存在的，而我们没有指定。而如果我们访问/requestParam.do?age=1 的时候就可以正常访问，因为我们传递了必须的参数age ，而参数name 是非必须的，不传递也可以。

## （三）使用 @CookieValue 绑定 cookie 的值到 Controller 方法参数

例5

@RequestMapping ( "cookieValue" )

public String testCookieValue( @CookieValue ( "hello" ) String cookieValue, @CookieValue String hello) {

System. out .println(cookieValue + "-----------" + hello);

return "cookieValue" ;

}

在上面的代码中我们使用@CookieValue 绑定了cookie 的值到方法参数上。上面一共绑定了两个参数，一个是明确指定要绑定的是名称为hello 的cookie 的值，一个是没有指定。使用没有指定的形式的规则和@PathVariable、@RequestParam 的规则是一样的，即在debug 编译模式下将自动获取跟方法参数名同名的cookie 值。

## （四）使用 @RequestHeader 注解绑定 HttpServletRequest 头信息到Controller 方法参数

例6

@RequestMapping ( "testRequestHeader" )

public String testRequestHeader( @RequestHeader ( "Host" ) String hostAddr, @RequestHeader String Host, @RequestHeader String host ) {

System. out .println(hostAddr + "-----" + Host + "-----" + host );

return "requestHeader" ;

}

在上面的代码中我们使用了 @RequestHeader 绑定了 HttpServletRequest 请求头 host 到Controller 的方法参数。上面方法的三个参数都将会赋予同一个值，由此我们可以知道在绑定请求头参数到方法参数的时候规则和 @PathVariable 、 @RequestParam 以及 @CookieValue 是一样的，即没有指定绑定哪个参数到方法参数的时候，在 debug 编译模式下将使用方法参数名作为需要绑定的参数。但是有一点 @RequestHeader 跟另外三种绑定方式是不一样的，那就是在使用 @RequestHeader 的时候是大小写不敏感的，即 @RequestHeader(“Host”) 和 @RequestHeader(“host”) 绑定的都是 Host 头信息。记住在 @PathVariable 、 @RequestParam 和 @CookieValue 中都是大小写敏感的。

## （五） @RequestMapping 的一些高级应用

在RequestMapping 中除了指定请求路径value 属性外，还有其他的属性可以指定，如params 、method 和headers 。这样属性都可以用于缩小请求的映射范围。

### 1.params属性

例7

@RequestMapping (value= "testParams" , params={ "param1=value1" , "param2" , "!param3" })

public String testParams() {

System. out .println( "test Params..........." );

return "testParams" ;

}

在上面的代码中我们用@RequestMapping 的params 属性指定了三个参数，这些参数都是针对请求参数而言的，它们分别表示参数param1 的值必须等于value1 ，参数param2 必须存在，值无所谓，参数param3 必须不存在，只有当请求/testParams.do 并且满足指定的三个参数条件的时候才能访问到该方法。所以当请求/testParams.do?param1=value1&param2=value2 的时候能够正确访问到该testParams 方法，当请求/testParams.do?param1=value1&param2=value2&param3=value3 的时候就不能够正常的访问到该方法，因为在@RequestMapping 的params 参数里面指定了参数param3 是不能存在的。

### 2.method属性

例8

@RequestMapping (value= "testMethod" , method={RequestMethod. GET , RequestMethod. DELETE })

public String testMethod() {

return "method" ;

}

在上面的代码中就使用method 参数限制了以GET 或DELETE 方法请求/testMethod.do 的时候才能访问到该Controller 的testMethod 方法。

### 3.headers属性

例9

@RequestMapping (value= "testHeaders" , headers={ "host=localhost" , "Accept" })

public String testHeaders() {

return "headers" ;

}

headers 属性的用法和功能与params 属性相似。在上面的代码中当请求/testHeaders.do 的时候只有当请求头包含Accept 信息，且请求的host 为localhost 的时候才能正确的访问到testHeaders 方法。

## （六）以 @RequestMapping 标记的处理器方法支持的方法参数和返回类型

### 1. 支持的方法参数类型

         （1 ）HttpServlet 对象，主要包括HttpServletRequest 、HttpServletResponse 和HttpSession 对象。 这些参数Spring 在调用处理器方法的时候会自动给它们赋值，所以当在处理器方法中需要使用到这些对象的时候，可以直接在方法上给定一个方法参数的申明，然后在方法体里面直接用就可以了。但是有一点需要注意的是在使用HttpSession 对象的时候，如果此时HttpSession 对象还没有建立起来的话就会有问题。

   （2 ）Spring 自己的WebRequest 对象。 使用该对象可以访问到存放在HttpServletRequest 和HttpSession 中的属性值。

   （3 ）InputStream 、OutputStream 、Reader 和Writer 。 InputStream 和Reader 是针对HttpServletRequest 而言的，可以从里面取数据；OutputStream 和Writer 是针对HttpServletResponse 而言的，可以往里面写数据。

   （4 ）使用@PathVariable 、@RequestParam 、@CookieValue 和@RequestHeader 标记的参数。

   （5 ）使用@ModelAttribute 标记的参数。

   （6 ）java.util.Map 、Spring 封装的Model 和ModelMap 。 这些都可以用来封装模型数据，用来给视图做展示。

   （7 ）实体类。 可以用来接收上传的参数。

   （8 ）Spring 封装的MultipartFile 。 用来接收上传文件的。

   （9 ）Spring 封装的Errors 和BindingResult 对象。 这两个对象参数必须紧接在需要验证的实体对象参数之后，它里面包含了实体对象的验证结果。

### 2. 支持的返回类型

         （1 ）一个包含模型和视图的ModelAndView 对象。

   （2 ）一个模型对象，这主要包括Spring 封装好的Model 和ModelMap ，以及java.util.Map ，当没有视图返回的时候视图名称将由RequestToViewNameTranslator 来决定。

   （3 ）一个View 对象。这个时候如果在渲染视图的过程中模型的话就可以给处理器方法定义一个模型参数，然后在方法体里面往模型中添加值。

   （4 ）一个String 字符串。这往往代表的是一个视图名称。这个时候如果需要在渲染视图的过程中需要模型的话就可以给处理器方法一个模型参数，然后在方法体里面往模型中添加值就可以了。

   （5 ）返回值是void 。这种情况一般是我们直接把返回结果写到HttpServletResponse 中了，如果没有写的话，那么Spring 将会利用RequestToViewNameTranslator 来返回一个对应的视图名称。如果视图中需要模型的话，处理方法与返回字符串的情况相同。

   （6 ）如果处理器方法被注解@ResponseBody 标记的话，那么处理器方法的任何返回类型都会通过HttpMessageConverters 转换之后写到HttpServletResponse 中，而不会像上面的那些情况一样当做视图或者模型来处理。

   （7 ）除以上几种情况之外的其他任何返回类型都会被当做模型中的一个属性来处理，而返回的视图还是由RequestToViewNameTranslator 来决定，添加到模型中的属性名称可以在该方法上用@ModelAttribute(“attributeName”) 来定义，否则将使用返回类型的类名称的首字母小写形式来表示。使用@ModelAttribute 标记的方法会在@RequestMapping 标记的方法执行之前执行。

## （七）使用 @ModelAttribute 和 @SessionAttributes 传递和保存数据

SpringMVC 支持使用 @ModelAttribute 和 @SessionAttributes 在不同的模型和控制器之间共享数据。 @ModelAttribute 主要有两种使用方式，一种是标注在方法上，一种是标注在 Controller 方法参数上。

当 @ModelAttribute 标记在方法上的时候，该方法将在处理器方法执行之前执行，然后把返回的对象存放在 session 或模型属性中，属性名称可以使用 @ModelAttribute(“attributeName”) 在标记方法的时候指定，若未指定，则使用返回类型的类名称（首字母小写）作为属性名称。关于 @ModelAttribute 标记在方法上时对应的属性是存放在 session 中还是存放在模型中，我们来做一个实验，看下面一段代码。

[复制代码](javascript:void(0);)

复制代码

@Controller

@RequestMapping ( "/myTest" )

public class MyController {

@ModelAttribute ( "hello" )

public String getModel() {

System. out .println( "-------------Hello---------" );

return "world" ;

}

@ModelAttribute ( "intValue" )

public int getInteger() {

System. out .println( "-------------intValue---------------" );

return 10;

}

@RequestMapping ( "sayHello" )

public void sayHello( @ModelAttribute ( "hello" ) String hello, @ModelAttribute ( "intValue" ) int num, @ModelAttribute ( "user2" ) User user, Writer writer, HttpSession session) throws IOException {

writer.write( "Hello " + hello + " , Hello " + user.getUsername() + num);

writer.write( "\r" );

Enumeration enume = session.getAttributeNames();

while (enume.hasMoreElements())

writer.write(enume.nextElement() + "\r" );

}

@ModelAttribute ( "user2" )

public User getUser() {

System. out .println( "---------getUser-------------" );

return new User(3, "user2" );

}

}

复制代码

[复制代码](javascript:void(0);)

当我们请求 /myTest/sayHello.do 的时候使用 @ModelAttribute 标记的方法会先执行，然后把它们返回的对象存放到模型中。最终访问到 sayHello 方法的时候，使用 @ModelAttribute 标记的方法参数都能被正确的注入值。执行结果如下所示：

 Hello world,Hello user210

       由执行结果我们可以看出来，此时 session 中没有包含任何属性，也就是说上面的那些对象都是存放在模型属性中，而不是存放在 session 属性中。那要如何才能存放在 session 属性中呢？这个时候我们先引入一个新的概念 @SessionAttributes ，它的用法会在讲完 @ModelAttribute 之后介绍，这里我们就先拿来用一下。我们在 MyController 类上加上 @SessionAttributes 属性标记哪些是需要存放到 session 中的。看下面的代码：

[复制代码](javascript:void(0);)

复制代码

@Controller

@RequestMapping ( "/myTest" )

@SessionAttributes (value={ "intValue" , "stringValue" }, types={User. class })

public class MyController {

@ModelAttribute ( "hello" )

public String getModel() {

System. out .println( "-------------Hello---------" );

return "world" ;

}

@ModelAttribute ( "intValue" )

public int getInteger() {

System. out .println( "-------------intValue---------------" );

return 10;

}

@RequestMapping ( "sayHello" )

public void sayHello(Map<String, Object> map, @ModelAttribute ( "hello" ) String hello, @ModelAttribute ( "intValue" ) int num, @ModelAttribute ( "user2" ) User user, Writer writer, HttpServletRequest request) throws IOException {

map.put( "stringValue" , "String" );

writer.write( "Hello " + hello + " , Hello " + user.getUsername() + num);

writer.write( "\r" );

HttpSession session = request.getSession();

Enumeration enume = session.getAttributeNames();

while (enume.hasMoreElements())

writer.write(enume.nextElement() + "\r" );

System. out .println(session);

}

@ModelAttribute ( "user2" )

public User getUser() {

System. out .println( "---------getUser-------------" );

return new User(3, "user2" );

}

}

复制代码

[复制代码](javascript:void(0);)

在上面代码中我们指定了属性为 intValue 或 stringValue 或者类型为 User 的都会放到 Session中，利用上面的代码当我们访问 /myTest/sayHello.do 的时候，结果如下：

 Hello world,Hello user210

仍然没有打印出任何 session 属性，这是怎么回事呢？怎么定义了把模型中属性名为 intValue 的对象和类型为 User 的对象存到 session 中，而实际上没有加进去呢？难道我们错啦？我们当然没有错，只是在第一次访问 /myTest/sayHello.do 的时候 @SessionAttributes 定义了需要存放到 session 中的属性，而且这个模型中也有对应的属性，但是这个时候还没有加到 session 中，所以 session 中不会有任何属性，等处理器方法执行完成后 Spring 才会把模型中对应的属性添加到 session 中。所以当请求第二次的时候就会出现如下结果：

 Hello world,Hello user210

user2

intValue

stringValue

当 @ModelAttribute 标记在处理器方法参数上的时候，表示该参数的值将从模型或者 Session 中取对应名称的属性值，该名称可以通过 @ModelAttribute(“attributeName”) 来指定，若未指定，则使用参数类型的类名称（首字母小写）作为属性名称。