SpringMVC的常用注解

1、@Controller  
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。

@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。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。有两种方式：

（1）在SpringMVC 的配置文件中定义MyController 的bean 对象。

（2）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" />//路径写到controller的上一层(扫描包详解见下面浅析) 2、@RequestMapping RequestMapping是一个用来处理请求地址映射的注解，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。

RequestMapping注解有六个属性，下面我们把她分成三类进行说明（下面有相应示例）。

1、 value， method；

value： 指定请求的实际地址，指定的地址可以是URI Template 模式（后面将会说明）；

method： 指定请求的method类型， GET、POST、PUT、DELETE等；

2、consumes，produces

consumes： 指定处理请求的提交内容类型（Content-Type），例如application/json, text/html;

produces: 指定返回的内容类型，仅当request请求头中的(Accept)类型中包含该指定类型才返回；

3、params，headers

params： 指定request中必须包含某些参数值是，才让该方法处理。

headers： 指定request中必须包含某些指定的header值，才能让该方法处理请求。

3、@Resource和@Autowired  
@Resource和@Autowired都是做bean的注入时使用，其实@Resource并不是Spring的注解，它的包是javax.annotation.Resource，需要导入，但是Spring支持该注解的注入。

1、共同点

两者都可以写在字段和setter方法上。两者如果都写在字段上，那么就不需要再写setter方法。

2、不同点

（1）@Autowired

@Autowired为Spring提供的注解，需要导入包org.springframework.beans.factory.annotation.Autowired;只按照byType注入。

复制代码  
public class TestServiceImpl {  
// 下面两种@Autowired只要使用一种即可  
@Autowired  
private UserDao userDao; // 用于字段上

@Autowired

public void setUserDao(UserDao userDao) { // 用于属性的方法上

this.userDao = userDao;

}

* 1
* 2
* 3
* 4

}  
复制代码  
@Autowired注解是按照类型（byType）装配依赖对象，默认情况下它要求依赖对象必须存在，如果允许null值，可以设置它的required属性为false。如果我们想使用按照名称（byName）来装配，可以结合@Qualifier注解一起使用。如下：

public class TestServiceImpl {  
@Autowired  
@Qualifier(“userDao”)  
private UserDao userDao;  
}  
（2）@Resource

@Resource默认按照ByName自动注入，由J2EE提供，需要导入包javax.annotation.Resource。@Resource有两个重要的属性：name和type，而Spring将@Resource注解的name属性解析为bean的名字，而type属性则解析为bean的类型。所以，如果使用name属性，则使用byName的自动注入策略，而使用type属性时则使用byType自动注入策略。如果既不制定name也不制定type属性，这时将通过反射机制使用byName自动注入策略。

复制代码  
public class TestServiceImpl {  
// 下面两种@Resource只要使用一种即可  
@Resource(name=“userDao”)  
private UserDao userDao; // 用于字段上

@Resource(name="userDao")

public void setUserDao(UserDao userDao) { // 用于属性的setter方法上

this.userDao = userDao;

}

* 1
* 2
* 3
* 4

}  
复制代码  
注：最好是将@Resource放在setter方法上，因为这样更符合面向对象的思想，通过set、get去操作属性，而不是直接去操作属性。

@Resource装配顺序：

①如果同时指定了name和type，则从Spring上下文中找到唯一匹配的bean进行装配，找不到则抛出异常。

②如果指定了name，则从上下文中查找名称（id）匹配的bean进行装配，找不到则抛出异常。

③如果指定了type，则从上下文中找到类似匹配的唯一bean进行装配，找不到或是找到多个，都会抛出异常。

④如果既没有指定name，又没有指定type，则自动按照byName方式进行装配；如果没有匹配，则回退为一个原始类型进行匹配，如果匹配则自动装配。

@Resource的作用相当于@Autowired，只不过@Autowired按照byType自动注入。

4、@ModelAttribute和 @SessionAttributes  
代表的是：该Controller的所有方法在调用前，先执行此@ModelAttribute方法，可用于注解和方法参数中，可以把这个@ModelAttribute特性，应用在BaseController当中，所有的Controller继承BaseController，即可实现在调用Controller时，先执行@ModelAttribute方法。

@SessionAttributes即将值放到session作用域中，写在class上面。

具体示例参见下面：使用 @ModelAttribute 和 @SessionAttributes 传递和保存数据

5、@PathVariable  
用于将请求URL中的模板变量映射到功能处理方法的参数上，即取出uri模板中的变量作为参数。如：

复制代码  
@Controller  
public class TestController {  
@RequestMapping(value="/user/{userId}/roles/{roleId}",method = RequestMethod.GET)  
public String getLogin(@PathVariable(“userId”) String userId,  
@PathVariable(“roleId”) String roleId){  
System.out.println(“User Id : " + userId);  
System.out.println(“Role Id : " + roleId);  
return “hello”;  
}  
@RequestMapping(value=”/product/{productId}”,method = RequestMethod.GET)  
public String getProduct(@PathVariable(“productId”) String productId){  
System.out.println(“Product Id : " + productId);  
return “hello”;  
}  
@RequestMapping(value=”/javabeat/{regexp1:[a-z-]+}",  
method = RequestMethod.GET)  
public String getRegExp(@PathVariable(“regexp1”) String regexp1){  
System.out.println("URI Part 1 : " + regexp1);  
return “hello”;  
}  
}  
复制代码  
6、@requestParam  
@requestParam主要用于在SpringMVC后台控制层获取参数，类似一种是request.getParameter(“name”)，它有三个常用参数：defaultValue = “0”, required = false, value = “isApp”；defaultValue 表示设置默认值，required 铜过boolean设置是否是必须要传入的参数，value 值表示接受的传入的参数类型。

7、@ResponseBody  
作用： 该注解用于将Controller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body数据区。

使用时机：返回的数据不是html标签的页面，而是其他某种格式的数据时（如json、xml等）使用；

8、@Component  
相当于通用的注解，当不知道一些类归到哪个层时使用，但是不建议。

9、@Repository  
用于注解dao层，在daoImpl类上面注解。

注：  
1、使用 @RequestMapping 来映射 Request 请求与处理器  
方式一、通过常见的类路径和方法路径结合访问controller方法

方式二、使用uri模板

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;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7

}  
复制代码  
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。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。

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。

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。如：

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  
当@RequestParam中没有指定参数名称时，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。

2、使用 @RequestMapping 的一些高级用法  
（1）params属性

@RequestMapping (value= “testParams” , params={ “param1=value1” , “param2” , “!param3” })  
public String testParams() {  
System. out .println( “test Params…” );  
return “testParams” ;  
}  
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

@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 的时候才能访问到该Controller 的testMethod 方法。

（3）headers属性

@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。

3、 @RequestMapping 标记的处理器方法支持的方法参数和返回类型

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。

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。

4、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（model）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  
@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" );

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26

}  
复制代码  
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：

* 1

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" );

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29

}  
复制代码  
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。

5、@PathVariable和@RequestParam的区别  
请求路径上有个id的变量值，可以通过@PathVariable来获取 @RequestMapping(value = “/page/{id}”, method = RequestMethod.GET)  
@RequestParam用来获得静态的URL请求入参 spring注解时action里用到。

简介：  
handler method 参数绑定常用的注解,我们根据他们处理的Request的不同内容部分分为四类：（主要讲解常用类型）

A、处理requet uri 部分（这里指uri template中variable，不含queryString部分）的注解： @PathVariable;

B、处理request header部分的注解： @RequestHeader, @CookieValue;

C、处理request body部分的注解：@RequestParam, @RequestBody;

D、处理attribute类型是注解： @SessionAttributes, @ModelAttribute;

（1）、@PathVariable  
当使用@RequestMapping URI template 样式映射时， 即 someUrl/{paramId}, 这时的paramId可通过 @Pathvariable注解绑定它传过来的值到方法的参数上。

示例代码：

复制代码  
@Controller  
@RequestMapping("/owners/{ownerId}")  
public class RelativePathUriTemplateController {

@RequestMapping("/pets/{petId}")  
public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {  
// implementation omitted  
}  
}  
复制代码  
上面代码把URI template 中变量 ownerId的值和petId的值，绑定到方法的参数上。若方法参数名称和需要绑定的uri template中变量名称不一致，需要在@PathVariable(“name”)指定uri template中的名称。

（2）、 @RequestHeader、@CookieValue  
@RequestHeader 注解，可以把Request请求header部分的值绑定到方法的参数上。

示例代码：

这是一个Request 的header部分：

Host localhost:8080  
Accept text/html,application/xhtml+xml,application/xml;q=0.9  
Accept-Language fr,en-gb;q=0.7,en;q=0.3  
Accept-Encoding gzip,deflate  
Accept-Charset ISO-8859-1,utf-8;q=0.7,\*;q=0.7  
Keep-Alive 300  
@RequestMapping("/displayHeaderInfo.do")  
public void displayHeaderInfo(@RequestHeader(“Accept-Encoding”) String encoding,  
@RequestHeader(“Keep-Alive”) long keepAlive) {  
}  
上面的代码，把request header部分的 Accept-Encoding的值，绑定到参数encoding上了， Keep-Alive header的值绑定到参数keepAlive上。

@CookieValue 可以把Request header中关于cookie的值绑定到方法的参数上。

例如有如下Cookie值：

JSESSIONID=415A4AC178C59DACE0B2C9CA727CDD84

@RequestMapping("/displayHeaderInfo.do")  
public void displayHeaderInfo(@CookieValue(“JSESSIONID”) String cookie) {  
}  
即把JSESSIONID的值绑定到参数cookie上。

（3）、@RequestParam, @RequestBody  
@RequestParam

A） 常用来处理简单类型的绑定，通过Request.getParameter() 获取的String可直接转换为简单类型的情况（ String–> 简单类型的转换操作由ConversionService配置的转换器来完成）；因为使用request.getParameter()方式获取参数，所以可以处理get 方式中queryString的值，也可以处理post方式中 body data的值；

B）用来处理Content-Type: 为 application/x-www-form-urlencoded编码的内容，提交方式GET、POST；

C) 该注解有两个属性： value、required； value用来指定要传入值的id名称，required用来指示参数是否必须绑定；

示例代码：

复制代码  
@Controller  
@RequestMapping("/pets")  
@SessionAttributes(“pet”)  
public class EditPetForm {  
@RequestMapping(method = RequestMethod.GET)  
public String setupForm(@RequestParam(“petId”) int petId, ModelMap model) {  
Pet pet = this.clinic.loadPet(petId);  
model.addAttribute(“pet”, pet);  
return “petForm”;  
}  
}  
复制代码  
@RequestBody

该注解常用来处理Content-Type: 不是application/x-www-form-urlencoded编码的内容，例如application/json, application/xml等；

它是通过使用HandlerAdapter 配置的HttpMessageConverters来解析post data body，然后绑定到相应的bean上的。

因为配置有FormHttpMessageConverter，所以也可以用来处理 application/x-www-form-urlencoded的内容，处理完的结果放在一个MultiValueMap<String, String>里，这种情况在某些特殊需求下使用，详情查看FormHttpMessageConverter api;

示例代码：

@RequestMapping(value = “/something”, method = RequestMethod.PUT)  
public void handle(@RequestBody String body, Writer writer) throws IOException {  
writer.write(body);  
}  
（4）、@SessionAttributes, @ModelAttribute  
@SessionAttributes:

该注解用来绑定HttpSession中的attribute对象的值，便于在方法中的参数里使用。

该注解有value、types两个属性，可以通过名字和类型指定要使用的attribute 对象；

示例代码：

@Controller  
@RequestMapping("/editPet.do")  
@SessionAttributes(“pet”)  
public class EditPetForm {  
// …  
}  
@ModelAttribute

该注解有两个用法，一个是用于方法上，一个是用于参数上；

用于方法上时： 通常用来在处理@RequestMapping之前，为请求绑定需要从后台查询的model；

用于参数上时： 用来通过名称对应，把相应名称的值绑定到注解的参数bean上；要绑定的值来源于：

A） @SessionAttributes 启用的attribute 对象上；

B） @ModelAttribute 用于方法上时指定的model对象；

C） 上述两种情况都没有时，new一个需要绑定的bean对象，然后把request中按名称对应的方式把值绑定到bean中。

用到方法上@ModelAttribute的示例代码：

@ModelAttribute  
public Account addAccount(@RequestParam String number) {  
return accountManager.findAccount(number);  
}  
这种方式实际的效果就是在调用@RequestMapping的方法之前，为request对象的model里put（“account”， Account）。

用在参数上的@ModelAttribute示例代码：

@RequestMapping(value="/owners/{ownerId}/pets/{petId}/edit", method = RequestMethod.POST)  
public String processSubmit(@ModelAttribute Pet pet) {

}  
首先查询 @SessionAttributes有无绑定的Pet对象，若没有则查询@ModelAttribute方法层面上是否绑定了Pet对象，若没有则将URI template中的值按对应的名称绑定到Pet对象的各属性上。

6、< context:component-scan base-package = “” />浅析  
component-scan 默认扫描的注解类型是 @Component，不过，在 @Component 语义基础上细化后的 @Repository, @Service 和 @Controller 也同样可以获得 component-scan 的青睐

有了context:component-scan，另一个context:annotation-config/标签根本可以移除掉，因为已经被包含进去了

另外context:annotation-config/还提供了两个子标签

<context:include-filter> //指定扫描的路径

* + 1

<context:exclude-filter> //排除扫描的路径

* + 1

context:component-scan有一个use-default-filters属性，属性默认为true,表示会扫描指定包下的全部的标有@Component的类，并注册成bean.也就是@Component的子注解@Service,@Reposity等。

这种扫描的粒度有点太大，如果你只想扫描指定包下面的Controller或其他内容则设置use-default-filters属性为false，表示不再按照scan指定的包扫描，而是按照context:include-filter指定的包扫描，示例：

<context:component-scan base-package=“com.tan” use-default-filters=“false”>  
<context:include-filter type=“regex” expression=“com.tan.*"/>//注意后面要写.*  
</context:component-scan>  
当没有设置use-default-filters属性或者属性为true时，表示基于base-packge包下指定扫描的具体路径  
<context:component-scan base-package=“com.tan” >  
<context:include-filter type=“regex” expression=”.controller.*"/>  
<context:include-filter type=“regex” expression=".service.*"/>  
<context:include-filter type=“regex” expression=".dao.\*"/>  
</context:component-scan>

效果相当于：  
<context:component-scan base-package=“com.tan” >  
<context:exclude-filter type=“regex” expression=".model.\*"/>  
</context:component-scan>

注意：本人尝试时无论哪种情况context:include-filter和context:exclude-filter都不能同时存在