**数据绑定**

到目前为止，请求已经能交给处理器进行处理了，接下来的事情是要进行收集数据啦，接下来我们看看能从请求中收集到哪些数据，如图

说明

1、@RequestParam绑定单个请求参数值；

2、@PathVariable绑定URI模板变量值；

3、@CookieValue绑定Cookie数据值

4、@RequestHeader绑定请求头数据；

5、@ModelValue绑定参数到命令对象；

6、@SessionAttributes绑定命令对象到session；

7、@RequestBody绑定请求的内容区数据并能进行自动类型转换等。

8、@RequestPart绑定“multipart/data”数据，除了能绑定@RequestParam能做到的请求参数外，还能绑定上传的文件等。

除了上边提到的注解，还可以通过如HttpServletRequest等API得到请求数据，但推荐使用注解方式，因为使用起来更简单

功能处理方法支持的参数类型 ，以及他们的具体含义

**一、ServletRequest/HttpServletRequest和 ServletResponse/HttpServletResponse**

public String requestOrResponse (

        ServletRequest servletRequest, HttpServletRequest httpServletRequest,

        ServletResponse servletResponse, HttpServletResponse httpServletResponse)   
Spring Web MVC框架会自动把相应的Servlet请求/响应作为参数传递过来。

**二、InputStream/OutputStream 和 Reader/Writer**

public void inputOrOutBody(InputStream requestBodyIn, OutputStream responseBodyOut)

        throws IOException {responseBodyOut.write("success".getBytes());   }   
requestBodyIn：获取请求的内容区字节流，等价于request.getInputStream();

responseBodyOut：获取相应的内容区字节流，等价于response.getOutputStream()。

public void readerOrWriteBody(Reader reader, Writer writer)

        throws IOException {   writer.write("hello");  }   
reader：获取请求的内容区字符流，等价于request.getReader();

writer：获取相应的内容区字符流，等价于response.getWriter()。

**注意：InputStream/OutputStream 和 Reader/Writer两组不能同时使用，只能使用其中的一组。**

**三、WebRequest/NativeWebRequest**

WebRequest是Spring Web MVC提供的统一请求访问接口，不仅仅可以访问请求相关数据（如参数区数据、请求头数据，但访问不到Cookie区数据），还可以访问会话和上下文中的数据；NativeWebRequest继承了WebRequest，并提供访问本地Servlet API的方法。

public String webRequest(WebRequest webRequest, NativeWebRequest nativeWebRequest) {

    System. *out*.println(webRequest.getParameter("test"));//①得到请求参数test的值

    webRequest.setAttribute("name", "value", WebRequest. *SCOPE\_REQUEST*);//②

    System. *out*.println(webRequest.getAttribute("name", WebRequest. *SCOPE\_REQUEST*));

    HttpServletRequest request =

        nativeWebRequest.getNativeRequest(HttpServletRequest.class);//③

    HttpServletResponse response =

        nativeWebRequest.getNativeResponse(HttpServletResponse.class);

        return "success";

    }

① webRequest.getParameter：访问请求参数区的数据，可以通过getHeader()访问请求头数据；

② webRequest.setAttribute/getAttribute：到指定的作用范围内取/放属性数据，Servlet定义的三个作用范围分别使用如下常量代表：

[SCOPE\_REQUEST](file:///F:\javass\s2s3h4\spring-framework-3.1.1.RELEASE-with-docs\spring-framework-3.1.1.RELEASE\docs\javadoc-api\org\springframework\web\context\request\RequestAttributes.html#SCOPE_REQUEST) ：代表请求作用范围；

[SCOPE\_SESSION](file:///F:\javass\s2s3h4\spring-framework-3.1.1.RELEASE-with-docs\spring-framework-3.1.1.RELEASE\docs\javadoc-api\org\springframework\web\context\request\RequestAttributes.html#SCOPE_SESSION) ：代表会话作用范围；

[SCOPE\_GLOBAL\_SESSION](file:///F:\javass\s2s3h4\spring-framework-3.1.1.RELEASE-with-docs\spring-framework-3.1.1.RELEASE\docs\javadoc-api\org\springframework\web\context\request\RequestAttributes.html#SCOPE_GLOBAL_SESSION) ：代表全局会话作用范围，即ServletContext上下文作用范围。

③ nativeWebRequest.getNativeRequest/nativeWebRequest.getNativeResponse：得到本地ServletAPI

**四、** **HttpSession**

public String session(HttpSession session) {

    System. *out*.println(session);

    return "success";

}   
此处的session永远不为null。

**注意：**session访问不是线程安全的，如果需要线程安全，需要设置AnnotationMethodHandlerAdapter或RequestMappingHandlerAdapter的synchronizeOnSession属性为true，即可线程安全的访问session。

**五、命令/表单对象**

Spring Web MVC能够自动将请求参数绑定到功能处理方法的命令/表单对象上。

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

public String toCreateUser(HttpServletRequest request, UserModel user) {

    return "customer/create";

}

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

public String createUser(HttpServletRequest request, UserModel user) {

    System. *out*.println(user);

    return "success";

}   
如果提交的表单（包含username和password文本域），将自动将请求参数绑定到命令对象user中去。

**六、** **Model、Map、ModelMap**

Spring Web MVC 提供Model、Map或ModelMap让我们能去暴露渲染视图需要的模型数据。

public String createUser(Model model, Map model2, ModelMap model3) {

    model.addAttribute("a", "a");

    model2.put("b", "b");

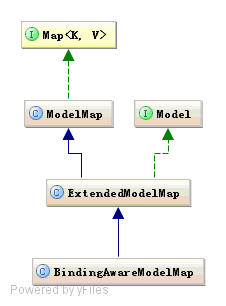
    model3.put("c", "c");

    System. *out*.println(model == model2);

    System. *out*.println(model2 == model3);

    return "success";}

**注意：**虽然此处注入的是三个不同的类型（Model model,Map model2, ModelMap model3），但三者是同一个对象



[AnnotationMethodHandlerAdapter](file:///F:\javass\s2s3h4\spring-framework-3.1.1.RELEASE-with-docs\spring-framework-3.1.1.RELEASE\docs\javadoc-api\org\springframework\web\servlet\mvc\annotation\AnnotationMethodHandlerAdapter.html)和RequestMappingHandlerAdapter将使用

BindingAwareModelMap作为模型对象的实现，即此处我们的形参（Model model, Map model2, ModelMap model3）都是同一个BindingAwareModelMap实例。

此处还有一点需要我们注意：

public ModelAndView mergeModel(Model model) {

    model.addAttribute("a", "a");//①添加模型数据

    ModelAndView mv = new ModelAndView("success");

    mv.addObject("a", "update");//②在视图渲染之前更新③处同名模型数据

    model.addAttribute("a", "new");//③修改①处同名模型数据

    //视图页面的a将显示为"update" 而不是"new"

    return mv;

}

从代码中我们可以总结出功能处理方法的返回值中的模型数据（如ModelAndView）会 合并 功能处理方法形式参数中的模型数据（如Model），但如果两者之间有同名的，返回值中的模型数据会覆盖形式参数中的模型数据。

**七、Errors/BindingResult**

public String error1(UserModel user, BindingResult result)

public String error2(UserModel user, BindingResult result, Model model)

public String error3(UserModel user, Errors errors)

以上代码都能获取错误对象。

Spring3.1之前（使用AnnotationMethodHandlerAdapter）错误对象必须紧跟在命令对象/表单对象之后，如下定义是错误的：

public String error4(UserModel user, Model model, Errors errors)

如上代码从Spring3.1开始（使用RequestMappingHandlerAdapter）将能正常工作，但还是推荐“错误对象紧跟在命令对象/表单对象之后”，这样是万无一失的。

**八、其他杂项**

public String other(Locale locale, Principal principal)

java.util.Locale：得到当前请求的本地化信息，默认等价于ServletRequest.getLocale()，如果配置LocaleResolver解析器则由它决定Locale，后续介绍；

java.security.Principal：该主体对象包含了验证通过的用户信息，等价于HttpServletRequest.getUserPrincipal()。

还有其他功能处理方法的形式参数类型（如HttpEntity、UriComponentsBuilder、SessionStatus、RedirectAttributes）等等。

n@RequestParam ：用于将请求参数区数据映射到功能处理方法的参数上。

**例子：**public String requestparam1(@RequestParam String username)

1：如果请求中包含username参数（如/requestparam1?username=zhang），则自动传入。

2：也可以使用@RequestParam("username")明确告诉Spring Web MVC使用username进行入参

n@RequestParam的主要参数

**value：**参数名字，即入参的请求参数名字，如username表示请求的参数区中的名字为username的参数的值将传入；

**required：**是否必须，默认是true，表示请求中一定要有相应的参数，否则将报400错误码；

**defaultValue：**默认值，表示如果请求中没有同名参数时的默认值，默认值可以是SpEL表达式，如“#{systemProperties[‘java.vm.version’]}”。

**例子1**：public String test(@RequestParam(value="username",required=false) String username)

表示请求中可以没有名字为username的参数，如果没有默认为null，需注意如下几点：

（1）：原子类型：必须有值，否则抛出异常，如果允许空值请使用包装类代替。

（2）：Boolean包装类型类型：默认Boolean.FALSE，其他引用类型默认为null。

**例子2**：public String requestparam5(

@RequestParam(value="username", required=true, defaultValue="zhang") String username) 表示如果请求中没有名字为username的参数，默认值为“zhang”。

n如果请求中有多个同名的应该如何接收呢？如给用户授权时，可能授予多个权限，先看如下代码：

public String test(@RequestParam(value="role") String roleList)

如果请求参数类似于url?role=admin&role=user，则实际roleList参数入参的数据为“admin，user”，即多个数据之间使用“，”分割；我们应该使用如下方式来接收多个请求参数：

public String test(@RequestParam(value="role") String[] roleList)

或

public String test(@RequestParam(value="list") List list)

n@PathVariable

功能：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

**例子：**

@RequestMapping(value="/users/{userId}/topics/{topicId}")

public String test(

       @PathVariable(value="userId") int userId,

       @PathVariable(value="topicId") int topicId)

如请求的URL为“控制器URL/users/123/topics/456”，则自动将URL中模板变量{userId}和{topicId}绑定到通过@PathVariable注解的同名参数上，即入参后userId=123、topicId=456

n@CookieValue

功能：用于将请求的Cookie数据映射到功能处理方法的参数上

**例子1：**

public String test(@CookieValue(value="JSESSIONID", defaultValue="") String sessionId)

如上配置将自动将JSESSIONID值入参到sessionId参数上，defaultValue表示Cookie中没有JSESSIONID时默认为空。

**例子2：**

public String test2(@CookieValue(value="JSESSIONID", defaultValue="") Cookie sessionId)

传入参数类型也可以是javax.servlet.http.Cookie类型。

@CookieValue也拥有和@RequestParam相同的三个参数，含义一样。

n@RequestHeader

功能：用于将请求的头信息区数据映射到功能处理方法的参数上

**例子：**

@RequestMapping(value="/header")

public String test(

       @RequestHeader("User-Agent") String userAgent,

       @RequestHeader(value="Accept") String[] accepts)

如上配置将自动将请求头“User-Agent”值入参到userAgent参数上，并将“Accept”请求头值入参到accepts参数上。

@RequestHeader也拥有和@RequestParam相同的三个参数，含义一样。

n@ModelAttribute

@ModelAttribute一般具有如下三个作用：

1：绑定请求参数到命令对象：放在功能处理方法的入参上时，用于将多个请求参数绑定到一个命令对象，从而简化绑定流程，而且自动暴露为模型数据用于视图页面展示时使用；

2：暴露表单引用对象为模型数据：放在处理器的一般方法（非功能处理方法）上时，是为表单准备要展示的表单引用对象，如注册时需要选择的所在城市等，而且在执行功能处理方法（@RequestMapping注解的方法）之前，自动添加到模型对象中，用于视图页面展示时使用；

3：暴露@RequestMapping方法返回值为模型数据：放在功能处理方法的返回值上时，是暴露功能处理方法的返回值为模型数据，用于视图页面展示时使用。

n **一、绑定请求参数到命令对象**

如实现用户登录，需要捕获用户s登录的请求参数（用户名、密码）并封装为用户对象，此时可以使用@ModelAttribute绑定多个请求参数到我们的命令对象。

**例子：**public String test1(@ModelAttribute("user") UserModel user)

**说明：1：**和前面命令/表单对象一样，只是此处多了一个注解@ModelAttribute(“user”)，它的作用是将该绑定的命令对象以“user”为名称添加到模型对象中供视图页面展示使用。我们此时可以在视图页面使用${user.username}来获取绑定的命令对象的属性。

2：绑定请求参数到命令对象支持对象图导航式的绑定，如请求参数包含“?username=zhang&password=123&workInfo.city=bj”自动绑定到user中的workInfo属性的city属性中。

3：@RequestMapping(value="/model2/{username}")

public String test2(@ModelAttribute("model") UserModel model) {

URI模板变量也能自动绑定到命令对象中，当你请求的URL中包含“&username=zhang”会自动绑定到命令对象上。

**当URI模板变量和请求参数同名时，** **请求参数** **具有高优先权。**

n **二、暴露表单引用对象为模型数据**

**例子1：**

@ModelAttribute("cityList")

public List cityList() {

    return Arrays. *asList*("北京", "山东");

}

如上代码会在执行功能处理方法之前执行，并将其自动添加到模型对象中，在功能处理方法中可以使用Model入参，则可以在处理方法中使用citylist了，如：

public ModelAndView handleRequest(Model m) {

List list = (List)m.asMap().get("cityList");

for(String s : list){

System.out.println("s==="+s);

}

......

}

**例子2：**

@ModelAttribute("user")  //①

public UserModel getUser(@RequestParam(value=“username", defaultValue="") String username) {  //TODO 去数据库根据用户名查找用户对象  }   
如你要修改用户资料时一般需要根据用户的编号/用户名查找用户来进行编辑，此时可以通过如上代码查找要编辑的用户。也可以进行一些默认值的处理。

@RequestMapping(value="/model1") //②

public String test1(@ModelAttribute("user") UserModel user, Model model)

**说明：**   
此处我们看到①和②有同名的命令对象，那Spring Web MVC内部如何处理的呢：

1、首先执行@ModelAttribute注解的方法，准备视图展示时所需要的模型数据；@ModelAttribute注解方法形式参数规则和@RequestMapping规则一样，如可以有@RequestParam等；

2、执行@RequestMapping注解方法，进行模型绑定时首先查找模型数据中是否含有同名对象，如果有直接使用，如果没有通过反射创建一个，因此②处的user将使用①处返回的命令对象。即②处的user等于①处的user。

n **三、暴露@RequestMapping方法返回值为模型数据**

**例子：**

public @ModelAttribute("user2") UserModel test3(@ModelAttribute("user2") UserModel user)

大家可以看到返回值类型是命令对象类型，而且通过@ModelAttribute(“user2”)注解，此时会暴露返回值到模型数据（名字为user2）中供视图展示使用。

可能有同学会注意到，此时@RequestMapping注解方法的入参user暴露到模型数据中的名字也是user2，那么到底user2代表哪一个呢？

规则是：@ModelAttribute注解的返回值会覆盖@RequestMapping注解方法中的@ModelAttribute注解的同名命令对象

n **四、匿名绑定命令参数**

**例子1：**

    public String test4(@ModelAttribute UserModel user, Model model)

    或   public String test5(UserModel user, Model model)

**说明：**   
    此时我们没有为命令对象提供暴露到模型数据中的名字，此时的名字是什么呢？Spring Web MVC自动将简单类名（首字母小写）作为名字暴露，如“cn.javass.springmvc.model.UserModel”暴露的名字为“userModel”。

**例子2：**

    public @ModelAttribute List test6()

    或   public @ModelAttribute List test7()

**说明：**   
    对于集合类型（Collection接口的实现者们，包括数组），生成的模型对象属性名为“简单类名（首字母小写）”+“List”，如List生成的模型对象属性名为“stringList”，List生成的模型对象属性名为“userModelList”。

其他情况一律都是使用简单类名（首字母小写）作为模型对象属性名，如Map类型的模型对象属性名为“map”。

n **@SessionAttributes** ：绑定命令对象到session

//1、在控制器类头上添加@SessionAttributes注解

@SessionAttributes(value = {"user"})    //①

public class SessionAttributeController

//2、@ModelAttribute注解的方法进行表单引用对象的创建

@ModelAttribute("user")    //②

public UserModel initUser()

//3、@RequestMapping注解方法的@ModelAttribute注解的参数进行命令对象的绑定

@RequestMapping("/session1")   //③

public String session1(@ModelAttribute("user") UserModel user)

//4、通过SessionStatus的setComplete()方法清除@SessionAttributes指定的会话数据

@RequestMapping("/session2")   //③

public String session(@ModelAttribute("user") UserModel user, SessionStatus status) {

    if(true) { //④

        status.setComplete();   }

    return "success";

}

n@SessionAttributes(value = {“user”})含义：

@SessionAttributes(value = {“user”}) 标识将模型数据中的名字为“user” 的对象存储到会话中（默认HttpSession），此处value指定将模型数据中的哪些数据（名字进行匹配）存储到会话中，此外还有一个types属性表示模型数据中的哪些类型的对象存储到会话范围内，如果同时指定value和types属性则那些名字和类型都匹配的对象才能存储到会话范围内。

n包含@SessionAttributes的执行流程如下所示：

① 首先根据类上的@SessionAttributes注解信息，查找会话内的对象放入到模型数据中；

② 执行@ModelAttribute注解的方法：如果模型数据中包含同名的数据，则不执行@ModelAttribute注解方法进行准备表单引用数据，而是使用①步骤中的会话数据；如果模型数据中不包含同名的数据，执行@ModelAttribute注解的方法并将返回值添加到模型数据中；

③ 执行@RequestMapping方法，绑定@ModelAttribute注解的参数：查找模型数据中是否有@ModelAttribute注解的同名对象，如果有直接使用，否则通过反射创建一个；并将请求参数绑定到该命令对象；

此处需要注意：如果使用@SessionAttributes注解控制器类之后，③步骤一定是从模型对象中取得同名的命令对象，如果模型数据中不存在将抛出HttpSessionRequiredException Expected session attribute ‘user’(Spring3.1)

或HttpSessionRequiredException Session attribute ‘user’ required - not found in session(Spring3.0)异常。

④ 如果会话可以销毁了，如多步骤提交表单的最后一步，此时可以调用SessionStatus对象的setComplete()标识当前会话的@SessionAttributes指定的数据可以清理了，此时当@RequestMapping功能处理方法执行完毕会进行清理会话数据。

n **@Value**

功能：用于将一个SpEL表达式结果映射到到功能处理方法的参数上

例子：

public String test(@Value("#{systemProperties['java.vm.version']}") String jvmVersion)

n **@MatrixVariable**

功能：用于接收URL的path中的矩阵参数

语法格式：XXX/XXX/path;name=value;name=value

开启功能：

（1）如果是xml配置的RequestMappingHandlerMapping，那么需要设置removeSemicolonContent属性为false

（2）如果是注解的方式，直接设置的enableMatrixVariables=“true“ 就可以了

例子代码见下页：

  测试的URL为

<http://localhost:9080/mvcexample/users/42;q=11;r=12/others/21;q=22;s=23>

@RequestMapping(value = "/users/{userId}/others/{otherUserId}",method = RequestMethod.GET)

public void hello(

//如果只有一个地方有q，也可以这么取，但如果有多个q，这样就错了，必须像第二个那样去指定取谁的q值

// @MatrixVariable int q,

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

@MatrixVariable(value="q", pathVar="otherUserId") int q2,

@MatrixVariable Map matrixVars,

@MatrixVariable(pathVar="userId") Map userIdMatrixVars

) {

// System.out.println("q=="+q);

System.out.println("q1="+q1+",q2="+q2+",matrixVars="+matrixVars+",userIdMatrixVars="+userIdMatrixVars);

}

n **运行结果为：**

q1=11,q2=22,matrixVars={q=[11, 22], r=[12], s=[23]},userIdMatrixVars={q=[11], r=[12]}

nSpring3引入一个mvc XML的命名空间用于支持mvc配置，包括如：

：

1：自动注册基于注解风格的处理器需要的DefaultAnnotationHandlerMapping、AnnotationMethodHandlerAdapter

2：支持Spring3的ConversionService自动注册

3：支持JSR-303验证框架的自动探测并注册（只需把JSR-303实现放置到classpath）

4：自动注册相应的HttpMessageConverter（用于支持@RequestBody  和 @ResponseBody）（如XML输入输出转换器（只需将JAXP实现放置到classpath）、JSON输入输出转换器（只需将Jackson实现放置到classpath））等。

：注册自定义的处理器拦截器；

：收到相应请求后直接选择相应的视图；

：逻辑静态资源路径到物理静态资源路径的支持；

：当在web.xml 中DispatcherServlet使用/ 映射时，能映射静态资源（当Spring Web MVC框架没有处理请求对应的控制器时（如一些静态资源），转交给默认的Servlet来响应静态文件，否则报404找不到资源错误，）。

n参考示例

*"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"*

    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/mvc*

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

    default-lazy-init= *"true"*>

*"cn.javass"* use-default-filters= *"false"*>

*"annotation"*expression= *"org.springframework.stereotype.Controller"*/>

      validator= *"validator"*

      conversion-service= *"conversionService"*

      >

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

*"viewClass"*value= *"org.springframework.web.servlet.view.JstlView"*/>

*"prefix"* value= *"/WEB-INF/jsp/"*/>

*"suffix"* value= *".jsp"*/>

*"conversionService"*class= *"org.springframework.format.support.FormattingConversionServiceFactoryBean"*>

*"converters"*>

*"cn.javass.springmvc.convert.StringToPhoneNumberConverter"*/>

*"validator"*

class= *"org.springframework.validation.beanvalidation.LocalValidatorFactoryBean"*>

*"providerClass"*  value= *"org.hibernate.validator.HibernateValidator"*/>

*"validationMessageSource"*ref= *"messageSource"*/>

*"messageSource"*

class= *"org.springframework.context.support.ReloadableResourceBundleMessageSource"*>

*"basename"* value= *"classpath:messages"*/>

*"fileEncodings"* value= *"utf-8"*/>

*"cacheSeconds"* value= *"120"*/>