[SpringMVC之controller篇](http://blog.csdn.net/liuyun1197628/article/details/22982749)

标签： [SpringMVC Contoller](http://www.csdn.net/tag/SpringMVC%20Contoller)

2014-04-05 12:38 11583人阅读 [评论](http://blog.csdn.net/liuyun1197628/article/details/22982749#comments)(1) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/liuyun1197628/article/details/22982749#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

SpringMVC（4） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

**概述**

继 Spring 2.0 对 Spring MVC 进行重大升级后，Spring 2.5 又为 Spring MVC 引入了注解驱动功能。现在你无须让 Controller 继承任何接口，无需在 XML 配置文件中定义请求和 Controller 的映射关系，仅仅使用注解就可以让一个 POJO 具有 Controller 的绝大部分功能 —— Spring MVC 框架的易用性得到了进一步的增强.在框架灵活性、易用性和扩展性上，Spring MVC 已经全面超越了其它的 MVC 框架，伴随着 Spring 一路高唱猛进，可以预见 Spring MVC 在 MVC 市场上的吸引力将越来越不可抗拒。

本文将介绍 Spring 2.5 新增的 Sping MVC 注解功能，讲述如何使用注解配置替换传统的基于 XML 的 Spring MVC 配置。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-spring25-mvc/#ibm-pcon)

**一个简单的基于注解的 Controller**

使用过低版本 Spring MVC 的读者都知道：当创建一个 Controller 时，我们需要直接或间接地实现 org.springframework.web.servlet.mvc.Controller 接口。一般情况下，我们是通过继承 SimpleFormController 或 MultiActionController 来定义自己的 Controller 的。在定义 Controller 后，一个重要的事件是在 Spring MVC 的配置文件中通过 HandlerMapping 定义请求和控制器的映射关系，以便将两者关联起来。

来看一下基于注解的 Controller 是如何定义做到这一点的，下面是使用注解的 BbtForumController：

**清单 1. BbtForumController.java**

|  |
| --- |
| package com.baobaotao.web;  import com.baobaotao.service.BbtForumService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.ModelAttribute;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import java.util.Collection;  @Controller //<——①  @RequestMapping("/forum.do")  public class BbtForumController {  @Autowired  private BbtForumService bbtForumService;  @RequestMapping //<——②  public String listAllBoard() {  bbtForumService.getAllBoard();  System.out.println("call listAllBoard method.");  return "listBoard";  }  } |

从上面代码中，我们可以看出 BbtForumController 和一般的类并没有区别，它没有实现任何特殊的接口，因而是一个地道的 POJO。让这个 POJO 与众不同的魔棒就是 Spring MVC 的注解！

在 ① 处使用了两个注解，分别是 @Controller 和 @RequestMapping。在“[使用 Spring 2.5 基于注解驱动的 IoC](http://www.ibm.com/developerworks/cn/java/j-lo-spring25-ioc/)”这篇文章里，笔者曾经指出过 @Controller、@Service 以及 @Repository 和 @Component 注解的作用是等价的：将一个类成为 Spring 容器的 Bean。由于 Spring MVC 的 Controller 必须事先是一个 Bean，所以 @Controller 注解是不可缺少的。

真正让 BbtForumController 具备 Spring MVC Controller 功能的是 @RequestMapping 这个注解。@RequestMapping 可以标注在类定义处，将 Controller 和特定请求关联起来；还可以标注在方法签名处，以便进一步对请求进行分流。在 ① 处，我们让 BbtForumController 关联“/forum.do”的请求，而 ② 处，我们具体地指定 listAllBoard() 方法来处理请求。所以在类声明处标注的 @RequestMapping 相当于让 POJO 实现了 Controller 接口，而在方法定义处的 @RequestMapping 相当于让 POJO 扩展 Spring 预定义的 Controller（如 SimpleFormController 等）。

为了让基于注解的 Spring MVC 真正工作起来，需要在 Spring MVC 对应的 xxx-servlet.xml 配置文件中做一些手脚。在此之前，还是先来看一下 web.xml 的配置吧：

**清单 2. web.xml：启用 Spring 容器和 Spring MVC 框架**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns="http://java.sun.com/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" version="2.5">  <display-name>Spring Annotation MVC Sample</display-name>  <!-- Spring 服务层的配置文件 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:applicationContext.xml</param-value>  </context-param>    <!-- Spring 容器启动监听器 -->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener  </listener-class>  </listener>  <!-- Spring MVC 的Servlet，它将加载WEB-INF/annomvc-servlet.xml 的  配置文件，以启动Spring MVC模块-->  <servlet>  <servlet-name>annomvc</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet  </servlet-class>  <load-on-startup>2</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>annomvc</servlet-name>  <url-pattern>\*.do</url-pattern>  </servlet-mapping>  </web-app> |

web.xml 中定义了一个名为 annomvc 的 Spring MVC 模块，按照 Spring MVC 的契约，需要在 WEB-INF/annomvc-servlet.xml 配置文件中定义 Spring MVC 模块的具体配置。annomvc-servlet.xml 的配置内容如下所示：

**清单 3. annomvc-servlet.xml**

|  |
| --- |
| <?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-2.5.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-2.5.xsd">    <!-- ①：对web包中的所有类进行扫描，以完成Bean创建和自动依赖注入的功能 -->  <context:component-scan base-package="com.baobaotao.web"/>  <!-- ②：启动Spring MVC的注解功能，完成请求和注解POJO的映射 -->  <bean class="org.springframework.web.servlet.mvc.annotation.  AnnotationMethodHandlerAdapter"/>  <!-- ③：对模型视图名称的解析，即在模型视图名称添加前后缀 -->  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"  p:prefix="/WEB-INF/jsp/" p:suffix=".jsp"/>  </beans> |

因为 Spring 所有功能都在 Bean 的基础上演化而来，所以必须事先将 Controller 变成 Bean，这是通过在类中标注 @Controller 并在 annomvc-servlet.xml 中启用组件扫描机制来完成的，如 ① 所示。

在 ② 处，配置了一个 AnnotationMethodHandlerAdapter，它负责根据 Bean 中的 Spring MVC 注解对 Bean 进行加工处理，使这些 Bean 变成控制器并映射特定的 URL 请求。

而 ③ 处的工作是定义模型视图名称的解析规则，这里我们使用了 Spring 2.5 的特殊命名空间，即 p 命名空间，它将原先需要通过 <property> 元素配置的内容转化为 <bean> 属性配置，在一定程度上简化了 <bean> 的配置。

启动 Tomcat，发送 http://localhost/forum.do URL 请求，BbtForumController 的 listAllBoard() 方法将响应这个请求，并转向 WEB-INF/jsp/listBoard.jsp 的视图页面。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-spring25-mvc/#ibm-pcon)

**让一个 Controller 处理多个 URL 请求**

在低版本的 Spring MVC 中，我们可以通过继承 MultiActionController 让一个 Controller 处理多个 URL 请求。使用 @RequestMapping 注解后，这个功能更加容易实现了。请看下面的代码：

**清单 3. 每个请求处理参数对应一个 URL**

|  |
| --- |
| package com.baobaotao.web;  import com.baobaotao.service.BbtForumService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  @Controller  public class BbtForumController {  @Autowired  private BbtForumService bbtForumService;  @RequestMapping("/listAllBoard.do") // <—— ①  public String listAllBoard() {  bbtForumService.getAllBoard();  System.out.println("call listAllBoard method.");  return "listBoard";  }  @RequestMapping("/listBoardTopic.do") // <—— ②  public String listBoardTopic(int topicId) {  bbtForumService.getBoardTopics(topicId);  System.out.println("call listBoardTopic method.");  return "listTopic";  }  } |

在这里，我们分别在 ① 和 ② 处为 listAllBoard() 和 listBoardTopic() 方法标注了 @RequestMapping 注解，分别指定这两个方法处理的 URL 请求，这相当于将 BbtForumController 改造为 MultiActionController。这样 /listAllBoard.do 的 URL 请求将由 listAllBoard() 负责处理，而 /listBoardTopic.do?topicId=1 的 URL 请求则由 listBoardTopic() 方法处理。

对于处理多个 URL 请求的 Controller 来说，我们倾向于通过一个 URL 参数指定 Controller 处理方法的名称（如 method=listAllBoard），而非直接通过不同的 URL 指定 Controller 的处理方法。使用 @RequestMapping 注解很容易实现这个常用的需求。来看下面的代码：

**清单 4. 一个 Controller 对应一个 URL，由请求参数决定请求处理方法**

|  |
| --- |
| package com.baobaotao.web;  import com.baobaotao.service.BbtForumService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  @Controller  @RequestMapping("/bbtForum.do") // <—— ① 指定控制器对应URL请求  public class BbtForumController {  @Autowired  private BbtForumService bbtForumService;  // <—— ② 如果URL请求中包括"method=listAllBoard"的参数，由本方法进行处理  @RequestMapping(params = "method=listAllBoard")  public String listAllBoard() {  bbtForumService.getAllBoard();  System.out.println("call listAllBoard method.");  return "listBoard";  }  // <—— ③ 如果URL请求中包括"method=listBoardTopic"的参数，由本方法进行处理  @RequestMapping(params = "method=listBoardTopic")  public String listBoardTopic(int topicId) {  bbtForumService.getBoardTopics(topicId);  System.out.println("call listBoardTopic method.");  return "listTopic";  }  } |

在类定义处标注的 @RequestMapping 让 BbtForumController 处理所有包含 /bbtForum.do 的 URL 请求，而 BbtForumController 中的请求处理方法对 URL 请求的分流规则在 ② 和 ③ 处定义分流规则按照 URL 的 method 请求参数确定。所以分别在类定义处和方法定义处使用 @RequestMapping 注解，就可以很容易通过 URL 参数指定 Controller 的处理方法了。

@RequestMapping 注解中除了 params 属性外，还有一个常用的属性是 method，它可以让 Controller 方法处理特定 HTTP 请求方式的请求，如让一个方法处理 HTTP GET 请求，而另一个方法处理 HTTP POST 请求，如下所示：

**清单 4. 让请求处理方法处理特定的 HTTP 请求方法**

|  |
| --- |
| package com.baobaotao.web;  import com.baobaotao.service.BbtForumService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  @Controller  @RequestMapping("/bbtForum.do")  public class BbtForumController {  @RequestMapping(params = "method=createTopic",method = RequestMethod.POST)  public String createTopic(){  System.out.println("call createTopic method.");  return "createTopic";  }  } |

这样只有当 /bbtForum.do?method=createTopic 请求以 HTTP POST 方式提交时，createTopic() 方法才会进行处理。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-spring25-mvc/#ibm-pcon)

**处理方法入参如何绑定 URL 参数**

**按契约绑定**

Controller 的方法标注了 @RequestMapping 注解后，它就能处理特定的 URL 请求。我们不禁要问：请求处理方法入参是如何绑定 URL 参数的呢？在回答这个问题之前先来看下面的代码：

**清单 5. 按参数名匹配进行绑定**

|  |
| --- |
| @RequestMapping(params = "method=listBoardTopic")  //<—— ① topicId入参是如何绑定URL请求参数的？  public String listBoardTopic(int topicId) {  bbtForumService.getBoardTopics(topicId);  System.out.println("call listBoardTopic method.");  return "listTopic";  } |

当我们发送 http://localhost//bbtForum.do?method=listBoardTopic&topicId=10 的 URL 请求时，Spring 不但让 listBoardTopic() 方法处理这个请求，而且还将 topicId 请求参数在类型转换后绑定到 listBoardTopic() 方法的 topicId 入参上。而 listBoardTopic() 方法的返回类型是 String，它将被解析为逻辑视图的名称。也就是说 Spring 在如何给处理方法入参自动赋值以及如何将处理方法返回值转化为 ModelAndView 中的过程中存在一套潜在的规则，不熟悉这个规则就不可能很好地开发基于注解的请求处理方法，因此了解这个潜在规则无疑成为理解 Spring MVC 框架基于注解功能的核心问题。

我们不妨从最常见的开始说起：请求处理方法入参的类型可以是 Java 基本数据类型或 String 类型，这时方法入参按参数名匹配的原则绑定到 URL 请求参数，同时还自动完成 String 类型的 URL 请求参数到请求处理方法参数类型的转换。下面给出几个例子：

* listBoardTopic(int topicId)：和 topicId URL 请求参数绑定；
* listBoardTopic(int topicId,String boardName)：分别和 topicId、boardName URL 请求参数绑定；

λ

特别的，如果入参是基本数据类型（如 int、long、float 等），URL 请求参数中一定要有对应的参数，否则将抛出 TypeMismatchException 异常，提示无法将 null 转换为基本数据类型。

另外，请求处理方法的入参也可以一个 JavaBean，如下面的 User 对象就可以作为一个入参：

**清单 6. User.java：一个 JavaBean**

|  |
| --- |
| package com.baobaotao.web;  public class User {  private int userId;  private String userName;  //省略get/setter方法  public String toString(){  return this.userName +","+this.userId;  }  } |

下面是将 User 作为 listBoardTopic() 请求处理方法的入参：

**清单 7. 使用 JavaBean 作为请求处理方法的入参**

|  |
| --- |
| @RequestMapping(params = "method=listBoardTopic")  public String listBoardTopic(int topicId,User user) {  bbtForumService.getBoardTopics(topicId);  System.out.println("topicId:"+topicId);  System.out.println("user:"+user);  System.out.println("call listBoardTopic method.");  return "listTopic";  } |

这时，如果我们使用以下的 URL 请求：http://localhost/bbtForum.do?method=listBoardTopic&topicId=1&userId=10&userName=tom

topicId URL 参数将绑定到 topicId 入参上，而 userId 和 userName URL 参数将绑定到 user 对象的 userId 和 userName 属性中。和 URL 请求中不允许没有 topicId 参数不同，虽然 User 的 userId 属性的类型是基本数据类型，但如果 URL 中不存在 userId 参数，Spring 也不会报错，此时 user.userId 值为 0。如果 User 对象拥有一个 dept.deptId 的级联属性，那么它将和 dept.deptId URL 参数绑定。

**通过注解指定绑定的 URL 参数**

如果我们想改变这种默认的按名称匹配的策略，比如让 listBoardTopic(int topicId,User user) 中的 topicId 绑定到 id 这个 URL 参数，那么可以通过对入参使用 @RequestParam 注解来达到目的：

**清单 8. 通过 @RequestParam 注解指定**

|  |
| --- |
| package com.baobaotao.web;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;  …  @Controller  @RequestMapping("/bbtForum.do")  public class BbtForumController {    @RequestMapping(params = "method=listBoardTopic")  public String listBoardTopic(@RequestParam("id") int topicId,User user) {  bbtForumService.getBoardTopics(topicId);  System.out.println("topicId:"+topicId);  System.out.println("user:"+user);  System.out.println("call listBoardTopic method.");  return "listTopic";  }  …  } |

这里，对 listBoardTopic() 请求处理方法的 topicId 入参标注了 @RequestParam("id") 注解，所以它将和 id 的 URL 参数绑定。

**绑定模型对象中某个属性**

Spring 2.0 定义了一个 org.springframework.ui.ModelMap 类，它作为通用的模型数据承载对象，传递数据供视图所用。我们可以在请求处理方法中声明一个 ModelMap 类型的入参，Spring 会将本次请求模型对象引用通过该入参传递进来，这样就可以在请求处理方法内部访问模型对象了。来看下面的例子：

**清单 9. 使用 ModelMap 访问请示对应的隐含模型对象**

|  |
| --- |
| @RequestMapping(params = "method=listBoardTopic")  public String listBoardTopic(@RequestParam("id")int topicId,  User user,ModelMap model) {  bbtForumService.getBoardTopics(topicId);  System.out.println("topicId:" + topicId);  System.out.println("user:" + user);  //① 将user对象以currUser为键放入到model中  model.addAttribute("currUser",user);  return "listTopic";  } |

对于当次请求所对应的模型对象来说，其所有属性都将存放到 request 的属性列表中。象上面的例子，ModelMap 中的 currUser 属性将放到 request 的属性列表中，所以可以在 JSP 视图页面中通过 request.getAttribute(“currUser”) 或者通过 ${currUser} EL 表达式访问模型对象中的 user 对象。从这个角度上看， ModelMap 相当于是一个向 request 属性列表中添加对象的一条管道，借由 ModelMap 对象的支持，我们可以在一个不依赖 Servlet API 的 Controller 中向 request 中添加属性。

在默认情况下，ModelMap 中的属性作用域是 request 级别是，也就是说，当本次请求结束后，ModelMap 中的属性将销毁。如果希望在多个请求中共享 ModelMap 中的属性，必须将其属性转存到 session 中，这样 ModelMap 的属性才可以被跨请求访问。

Spring 允许我们有选择地指定 ModelMap 中的哪些属性需要转存到 session 中，以便下一个请求属对应的 ModelMap 的属性列表中还能访问到这些属性。这一功能是通过类定义处标注 @SessionAttributes 注解来实现的。请看下面的代码：

**清单 10. 使模型对象的特定属性具有 Session 范围的作用域**

|  |
| --- |
| package com.baobaotao.web;  …  import org.springframework.ui.ModelMap;  import org.springframework.web.bind.annotation.SessionAttributes;  @Controller  @RequestMapping("/bbtForum.do")  @SessionAttributes("currUser") //①将ModelMap中属性名为currUser的属性  //放到Session属性列表中，以便这个属性可以跨请求访问  public class BbtForumController {  …  @RequestMapping(params = "method=listBoardTopic")  public String listBoardTopic(@RequestParam("id")int topicId, User user,  ModelMap model) {  bbtForumService.getBoardTopics(topicId);  System.out.println("topicId:" + topicId);  System.out.println("user:" + user);  model.addAttribute("currUser",user); //②向ModelMap中添加一个属性  return "listTopic";  }  } |

我们在 ② 处添加了一个 ModelMap 属性，其属性名为 currUser，而 ① 处通过 @SessionAttributes 注解将 ModelMap 中名为 currUser 的属性放置到 Session 中，所以我们不但可以在 listBoardTopic() 请求所对应的 JSP 视图页面中通过 request.getAttribute(“currUser”) 和 session.getAttribute(“currUser”) 获取 user 对象，还可以在下一个请求所对应的 JSP 视图页面中通过 session.getAttribute(“currUser”) 或 ModelMap#get(“currUser”) 访问到这个属性。

这里我们仅将一个 ModelMap 的属性放入 Session 中，其实 @SessionAttributes 允许指定多个属性。你可以通过字符串数组的方式指定多个属性，如 @SessionAttributes({“attr1”,”attr2”})。此外，@SessionAttributes 还可以通过属性类型指定要 session 化的 ModelMap 属性，如 @SessionAttributes(types = User.class)，当然也可以指定多个类，如 @SessionAttributes(types = {User.class,Dept.class})，还可以联合使用属性名和属性类型指定：@SessionAttributes(types = {User.class,Dept.class},value={“attr1”,”attr2”})。

上面讲述了如何往ModelMap中放置属性以及如何使ModelMap中的属性拥有Session域的作用范围。除了在JSP视图页面中通过传统的方法访问ModelMap中的属性外，读者朋友可能会问：是否可以将ModelMap中的属性绑定到请求处理方法的入参中呢？答案是肯定的。Spring为此提供了一个@ModelAttribute的注解，下面是使用@ModelAttribute注解的例子：

**清单 11. 使模型对象的特定属性具有 Session 范围的作用域**

|  |
| --- |
| package com.baobaotao.web;  import com.baobaotao.service.BbtForumService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.ui.ModelMap;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;  import org.springframework.web.bind.annotation.SessionAttributes;  import org.springframework.web.bind.annotation.ModelAttribute;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpSession;  @Controller  @RequestMapping("/bbtForum.do")  @SessionAttributes("currUser") //①让ModelMap的currUser属性拥有session级作用域  public class BbtForumController {  @Autowired  private BbtForumService bbtForumService;  @RequestMapping(params = "method=listBoardTopic")  public String listBoardTopic(@RequestParam("id")int topicId, User user,  ModelMap model) {  bbtForumService.getBoardTopics(topicId);  System.out.println("topicId:" + topicId);  System.out.println("user:" + user);  model.addAttribute("currUser",user); //②向ModelMap中添加一个属性  return "listTopic";  }  @RequestMapping(params = "method=listAllBoard")  //③将ModelMap中的  public String listAllBoard(@ModelAttribute("currUser") User user) {  //currUser属性绑定到user入参中。  bbtForumService.getAllBoard();  System.out.println("user:"+user);  return "listBoard";  }  } |

在 ② 处，我们向 ModelMap 中添加一个名为 currUser 的属性，而 ① 外的注解使这个 currUser 属性拥有了 session 级的作用域。所以，我们可以在 ③ 处通过 @ModelAttribute 注解将 ModelMap 中的 currUser 属性绑定以请求处理方法的 user 入参中。

所以当我们先调用以下 URL 请求： http://localhost/bbtForum.do?method=listBoardTopic&id=1&userName=tom&dept.deptId=12

以执行listBoardTopic()请求处理方法，然后再访问以下URL： http://localhost/sample/bbtForum.do?method=listAllBoard

你将可以看到 listAllBoard() 的 user 入参已经成功绑定到 listBoardTopic() 中注册的 session 级的 currUser 属性上了。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-spring25-mvc/#ibm-pcon)

**请求处理方法的签名规约**

**方法入参**

我们知道标注了 @RequestMapping 注解的 Controller 方法就成为了请求处理方法，Spring MVC 允许极其灵活的请求处理方法签名方式。对于方法入参来说，它允许多种类型的入参，通过下表进行说明：

|  |  |
| --- | --- |
| **请求处理方法入参的可选类型** | **说明** |
| Java 基本数据类型和 String | 默认情况下将按名称匹配的方式绑定到 URL 参数上，可以通过 @RequestParam 注解改变默认的绑定规则 |
| request/response/session | 既可以是 Servlet API 的也可以是 Portlet API 对应的对象，Spring 会将它们绑定到 Servlet 和 Portlet 容器的相应对象上 |
| org.springframework.web.context.request.WebRequest | 内部包含了 request 对象 |
| java.util.Locale | 绑定到 request 对应的 Locale 对象上 |
| java.io.InputStream/java.io.Reader | 可以借此访问 request 的内容 |
| java.io.OutputStream / java.io.Writer | 可以借此操作 response 的内容 |
| 任何标注了 @RequestParam 注解的入参 | 被标注 @RequestParam 注解的入参将绑定到特定的 request 参数上。 |
| java.util.Map / org.springframework.ui.ModelMap | 它绑定 Spring MVC 框架中每个请求所创建的潜在的模型对象，它们可以被 Web 视图对象访问（如 JSP） |
| 命令/表单对象（注：一般称绑定使用 HTTP GET 发送的 URL 参数的对象为命令对象，而称绑定使用 HTTP POST 发送的 URL 参数的对象为表单对象） | 它们的属性将以名称匹配的规则绑定到 URL 参数上，同时完成类型的转换。而类型转换的规则可以通过 @InitBinder 注解或通过 HandlerAdapter 的配置进行调整 |
| org.springframework.validation.Errors / org.springframework.validation.BindingResult | 为属性列表中的命令/表单对象的校验结果，注意检验结果参数必须紧跟在命令/表单对象的后面 |
| rg.springframework.web.bind.support.SessionStatus | 可以通过该类型 status 对象显式结束表单的处理，这相当于触发 session 清除其中的通过 @SessionAttributes 定义的属性 |

Spring MVC 框架的易用之处在于，你可以按任意顺序定义请求处理方法的入参（除了 Errors 和 BindingResult 必须紧跟在命令对象/表单参数后面以外），Spring MVC 会根据反射机制自动将对应的对象通过入参传递给请求处理方法。这种机制让开发者完全可以不依赖 Servlet API 开发控制层的程序，当请求处理方法需要特定的对象时，仅仅需要在参数列表中声明入参即可，不需要考虑如何获取这些对象，Spring MVC 框架就象一个大管家一样“不辞辛苦”地为我们准备好了所需的一切。下面演示一下使用 SessionStatus 的例子：

**清单 12. 使用 SessionStatus 控制 Session 级别的模型属性**

|  |
| --- |
| @RequestMapping(method = RequestMethod.POST)  public String processSubmit(@ModelAttribute Owner owner,  BindingResult result, SessionStatus status) {//<——①  new OwnerValidator().validate(owner, result);  if (result.hasErrors()) {  return "ownerForm";  }  else {  this.clinic.storeOwner(owner);  status.setComplete();//<——②  return "redirect:owner.do?ownerId=" + owner.getId();  }  } |

processSubmit() 方法中的 owner 表单对象将绑定到 ModelMap 的“owner”属性中，result 参数用于存放检验 owner 结果的对象，而 status 用于控制表单处理的状态。在 ② 处，我们通过调用 status.setComplete() 方法，该 Controller 所有放在 session 级别的模型属性数据将从 session 中清空。

**方法返回参数**

在低版本的 Spring MVC 中，请求处理方法的返回值类型都必须是 ModelAndView。而在 Spring 2.5 中，你拥有多种灵活的选择。通过下表进行说明：

|  |  |
| --- | --- |
| **请求处理方法入参的可选类型** | **说明** |
| void | 此时逻辑视图名由请求处理方法对应的 URL 确定，如以下的方法：   |  | | --- | | @RequestMapping("/welcome.do")  public void welcomeHandler() {  } |   对应的逻辑视图名为“welcome” |
| String | 此时逻辑视图名为返回的字符，如以下的方法：   |  | | --- | | @RequestMapping(method = RequestMethod.GET)  public String setupForm(@RequestParam("ownerId") int ownerId, ModelMap model) {  Owner owner = this.clinic.loadOwner(ownerId);  model.addAttribute(owner);  return "ownerForm";  } |   对应的逻辑视图名为“ownerForm” |
| org.springframework.ui.ModelMap | 和返回类型为 void 一样，逻辑视图名取决于对应请求的 URL，如下面的例子：   |  | | --- | | @RequestMapping("/vets.do")  public ModelMap vetsHandler() {  return new ModelMap(this.clinic.getVets());  } |   对应的逻辑视图名为“vets”，返回的 ModelMap 将被作为请求对应的模型对象，可以在 JSP 视图页面中访问到。 |
| ModelAndView | 当然还可以是传统的 ModelAndView。 |

应该说使用 String 作为请求处理方法的返回值类型是比较通用的方法，这样返回的逻辑视图名不会和请求 URL 绑定，具有很大的灵活性，而模型数据又可以通过 ModelMap 控制。当然直接使用传统的 ModelAndView 也不失为一个好的选择。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-spring25-mvc/#ibm-pcon)

**注册自己的属性编辑器**

Spring MVC 有一套常用的属性编辑器，这包括基本数据类型及其包裹类的属性编辑器、String 属性编辑器、JavaBean 的属性编辑器等。但有时我们还需要向 Spring MVC 框架注册一些自定义的属性编辑器，如特定时间格式的属性编辑器就是其中一例。

Spring MVC 允许向整个 Spring 框架注册属性编辑器，它们对所有 Controller 都有影响。当然 Spring MVC 也允许仅向某个 Controller 注册属性编辑器，对其它的 Controller 没有影响。前者可以通过 AnnotationMethodHandlerAdapter 的配置做到，而后者则可以通过 @InitBinder 注解实现。

下面先看向整个 Spring MVC 框架注册的自定义编辑器：

**清单 13. 注册框架级的自定义属性编辑器**

|  |
| --- |
| >bean class="org.springframework.web.servlet.mvc.annotation.  AnnotationMethodHandlerAdapter"<  >property name="webBindingInitializer"<  >bean class="com.baobaotao.web.MyBindingInitializer"/<  >/property<  >/bean< |

MyBindingInitializer 实现了 WebBindingInitializer 接口，在接口方法中通过 binder 注册多个自定义的属性编辑器，其代码如下所示：

**清单 14.自定义属性编辑器**

|  |
| --- |
| package org.springframework.samples.petclinic.web;  import java.text.SimpleDateFormat;  import java.util.Date;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.beans.propertyeditors.CustomDateEditor;  import org.springframework.beans.propertyeditors.StringTrimmerEditor;  import org.springframework.samples.petclinic.Clinic;  import org.springframework.samples.petclinic.PetType;  import org.springframework.web.bind.WebDataBinder;  import org.springframework.web.bind.support.WebBindingInitializer;  import org.springframework.web.context.request.WebRequest;  public class MyBindingInitializer implements WebBindingInitializer {  public void initBinder(WebDataBinder binder, WebRequest request) {  SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");  dateFormat.setLenient(false);  binder.registerCustomEditor(Date.class,  new CustomDateEditor(dateFormat, false));  binder.registerCustomEditor(String.class, new StringTrimmerEditor(false));  }  } |

如果希望某个属性编辑器仅作用于特定的 Controller，可以在 Controller 中定义一个标注 @InitBinder 注解的方法，可以在该方法中向 Controller 了注册若干个属性编辑器，来看下面的代码：

**清单 15. 注册 Controller 级的自定义属性编辑器**

|  |
| --- |
| @Controller  public class MyFormController {  @InitBinder  public void initBinder(WebDataBinder binder) {  SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");  dateFormat.setLenient(false);  binder.registerCustomEditor(Date.class, new CustomDateEditor(dateFormat, false));  }  …  } |

注意被标注 @InitBinder 注解的方法必须拥有一个 WebDataBinder 类型的入参，以便 Spring MVC 框架将注册属性编辑器的 WebDataBinder 对象传递进来。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-spring25-mvc/#ibm-pcon)

**如何准备数据**

在编写 Controller 时，常常需要在真正进入请求处理方法前准备一些数据，以便请求处理或视图渲染时使用。在传统的 SimpleFormController 里，是通过复写其 referenceData() 方法来准备引用数据的。在 Spring 2.5 时，可以将任何一个拥有返回值的方法标注上 @ModelAttribute，使其返回值将会进入到模型对象的属性列表中。来看下面的例子：

**清单 16. 定义为处理请求准备数据的方法**

|  |
| --- |
| package com.baobaotao.web;  import com.baobaotao.service.BbtForumService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.ui.ModelMap;  import org.springframework.web.bind.annotation.ModelAttribute;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;  import org.springframework.web.bind.annotation.SessionAttributes;  import java.util.ArrayList;  import java.util.List;  import java.util.Set;  @Controller  @RequestMapping("/bbtForum.do")  public class BbtForumController {  @Autowired  private BbtForumService bbtForumService;  @ModelAttribute("items")//<——①向模型对象中添加一个名为items的属性  public List<String> populateItems() {  List<String> lists = new ArrayList<String>();  lists.add("item1");  lists.add("item2");  return lists;  }  @RequestMapping(params = "method=listAllBoard")  public String listAllBoard(@ModelAttribute("currUser")User user, ModelMap model) {  bbtForumService.getAllBoard();  //<——②在此访问模型中的items属性  System.out.println("model.items:" + ((List<String>)model.get("items")).size());  return "listBoard";  }  } |

在 ① 处，通过使用 @ModelAttribute 注解，populateItem() 方法将在任何请求处理方法执行前调用，Spring MVC 会将该方法返回值以“items”为名放入到隐含的模型对象属性列表中。

所以在 ② 处，我们就可以通过 ModelMap 入参访问到 items 属性，当执行 listAllBoard() 请求处理方法时，② 处将在控制台打印出“model.items:2”的信息。当然我们也可以在请求的视图中访问到模型对象中的 items 属性。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-spring25-mvc/#ibm-pcon)

**小结**

Spring 2.5 对 Spring MVC 进行了很大增强，现在我们几乎完全可以使用基于注解的 Spring MVC 完全替换掉原来基于接口 Spring MVC 程序。基于注解的 Spring MVC 比之于基于接口的 Spring MVC 拥有以下几点好处：

* 方便请求和控制器的映射；
* 方便请求处理方法入参绑定URL参数；
* Controller 不必继承任何接口，它仅是一个简单的 POJO。

但是基于注解的 Spring MVC 并不完美，还存在优化的空间，因为在某些配置上它比基于 XML 的配置更繁琐。比如对于处理多个请求的 Controller 来说，假设我们使用一个 URL 参数指定调用的处理方法（如 xxx.do?method=listBoardTopic），当使用注解时，每个请求处理方法都必须使用 @RequestMapping() 注解指定对应的 URL 参数（如 @RequestMapping(params = "method=listBoardTopic")），而在 XML 配置中我们仅需要配置一个 ParameterMethodNameResolver 就可以了。

**参考资料**

**学习**

* [Spring 系列：Spring 框架简介](http://www.ibm.com/developerworks/cn/java/wa-spring1/)：优秀的 Spring 框架入门系列，了解 Spring 框架的基本概念。
* [轻量级开发的成功秘诀，第 3 部分: Spring 露出水面](http://www.ibm.com/developerworks/cn/opensource/os-lightweight3/)：介绍了在 Spring 框架的轻量级 Ioc 容器。
* [Spring Framework 和 IBM WebSphere Application Server](http://www.ibm.com/developerworks/cn/websphere/library/techarticles/0706_johnsonbuck/0706_johnsonbuck.html)：Interface21 的首席执行官 Rod Johnson 和 IBM 的 WebSphere Open Source 主管 Paul Buck 讨论了 Spring Framework 通过 IBM WebSphere Application Server 认证对 Spring 和 WebSphere 产品系列的开发人员和客户有何重要意义。
* [Tiger 中的注释，第 1 部分: 向 Java 代码中添加元数据](http://www.ibm.com/developerworks/cn/java/j-annotate1/)：解释了元数据如此有用的原因，向您介绍了 Java 语言中的注释，并研究了 Tiger 的内置注释。
* [Tiger 中的注释，第 2 部分: 定制注释](http://www.ibm.com/developerworks/cn/java/j-annotate2.html)：说明了如何创建定制注释，如何用自己的注释注解文档，并进一步定制代码。

**获得产品和技术**

* [Springframework 网站](http://www.springframework.org/download)：下载 Spring 框架。