**使用 Spring 3 来创建 RESTful Web Services**

在 Java™ 中，您可以使用以下几种方法来创建 RESTful Web Service：使用 JSR 311（311）及其参考实现 Jersey、使用 Restlet 框架和从头开始开发。Spring 是流行的 Java EE 应用开发框架，现在它的 MVC 层也支持 REST 了。本文将介绍使用 Spring 开发 RESTful Web Services 的方法。读者将了解如何使用 Spring API 和注释来开发 RESTful Web Services，以及 Spring 是如何将这个新特性整合到它的框架中的。

**引言**

Roy Fielding 是 HTTP 1.0 和 1.1 标准的主要作者之一，2000 年，他在他的博士论文中首次提出了 REST。

通过 REST 风格体系架构，请求和响应都是基于资源表示的传输来构建的。资源是通过全局 ID 来标识的，这些 ID 一般使用的是一个统一资源标识符（URI）。客户端应用使用 HTTP 方法（如，GET、POST、PUT 或 DELETE）来操作一个或多个资源。通常，GET 是用于获取或列出一个或多个资源，POST 用于创建，PUT 用于更新或替换，而 DELETE 则用于删除资源。

例如，GET http://host/context/employees/12345 将获取 ID 为 12345 的员工的*表示*。这个响应表示可以是包含详细的员工信息的 XML 或 ATOM，或者是具有更好 UI 的 JSP/HTML 页面。您看到哪种表示方式取决于服务器端实现和您的客户端请求的 MIME 类型。

RESTful Web Service 是一个使用 HTTP 和 REST 原理实现的 Web Service。通常，一个 RESTful Web Service 将定义基本资源 URI、它所支持的表示/响应 MIME，以及它所支持的操作。

本文将介绍如何使用 Spring 创建 Java 实现的服务器端 RESTful Web Services。这个例子将使用浏览器、curl 和 Firefox 插件 *RESTClient* 作为发出请求的客户端。您可以 [下载](http://www.ibm.com/developerworks/cn/web/wa-spring3webserv/#download) 本文所使用的源代码。

本文假定您是熟悉 REST 基本知识的。[参考资料](http://www.ibm.com/developerworks/cn/web/wa-spring3webserv/#resources) 中有更多关于 REST 的信息。

[**回页首**](http://www.ibm.com/developerworks/cn/web/wa-spring3webserv/#ibm-pcon)

**Spring 3 的 REST 支持**

在 Spring 框架支持 REST 之前，人们会使用其他几种实现技术来创建 Java 的 RESTful Web Services，如 Restlet、RestEasy 和 Jersey。Jersey 是其中最值得注意的，它是 JAX-RS（JSR 311）的参考实现。[参考资料](http://www.ibm.com/developerworks/cn/web/wa-spring3webserv/#resources) 中有更多关于 JSR 311 和 Jersey 的信息。

Spring 是一个得到广泛应用的 Java EE 框架，它在版本 3 以后就增加了 RESTful Web Services 开发的支持。虽然，对 REST 的支持并不是 JAX-RS 的一种实现，但是它具有比标准定义更多的特性。REST 支持被无缝整合到 Spring 的 MVC 层，它可以很容易应用到使用 Spring 构建的应用中。

Spring REST 支持的主要特性包括：

* 注释，如 @RequestMapping 和 @PathVariable，支持资源标识和 URL 映射
* ContentNegotiatingViewResolver 支持为不同的 MIME/内容类型使用不同的表示方式
* 使用相似的编程模型无缝地整合到原始的 MVC 层

[**回页首**](http://www.ibm.com/developerworks/cn/web/wa-spring3webserv/#ibm-pcon)

**创建一个示例 RESTful Web Service**

本节中的例子将演示 Spring 3 环境的创建过程，并创建一个可以部署到 Tomcat 中的 “Hello World” 应用。然后我们再完成一个更复杂的应用来了解 Spring 3 REST 支持的重要概念，如多种 MIME 类型表示支持和 JAXB 支持。另外，本文还使用一些代码片断来帮助理解这些概念。您可以 [下载](http://www.ibm.com/developerworks/cn/web/wa-spring3webserv/#download) 本文的所有示例代码。

本文假定您已经熟悉 Spring 框架和 Spring MVC。

**Hello World：使用 Spring 3 REST 支持**

要创建这个例子所使用的开发环境，您需要：

* IDE：Eclipse IDE for JEE (v3.4+)
* Java SE5 以上
* Web 容器：Apache Tomcat 6.0（Jetty 或其他容器也可）
* Spring 3 框架（v3.0.3 是本文编写时的最新版本）
* 其他程序库：JAXB 2、JSTL、commons-logging

在 Eclipse 中创建一个 Web 应用，然后设置 Tomcat 6 作为它的运行环境。然后，您需要设置 web.xml 文件来激活 Spring WebApplicationContext。这个例子将 Spring bean 配置分成两个文件：rest-servlet.xml 包含与 MVC/REST 有关的配置，rest-context.xml 包含服务级别的配置（如数据源 beans）。清单 1 显示了 web.xml 中的 Spring 配置的部分。

**清单 1. 在 web.xml 中激活 Spring WebApplicationContext**

<context-param>

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

<param-value>

/WEB-INF/rest-context.xml

</param-value>

</context-param>

<!-- This listener will load other application context file in addition to

rest-servlet.xml -->

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

<servlet>

<servlet-name>rest</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

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

</servlet>

<servlet-mapping>

<servlet-name>rest</servlet-name>

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

</servlet-mapping>

在 rest-servlet.xml 文件中创建 Spring MVC 的相关配置（Controller、View、View Resolver）。清单 2 显示了其中最重要的部分。

**清单 2. 在 rest-servlet.xml 文件中创建 Spring MVC 配置**

<context:component-scan base-package=*"dw.spring3.rest.controller"* />

<!--To enable @RequestMapping process on type level and method level-->

<bean class=*"org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping"* />

<bean class=*"org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter"* />

<!--Use JAXB OXM marshaller to marshall/unmarshall following class-->

<bean id=*"jaxbMarshaller"*

class=*"org.springframework.oxm.jaxb.Jaxb2Marshaller"*>

<property name=*"classesToBeBound"*>

<list>

<value>dw.spring3.rest.bean.Employee</value>

<value>dw.spring3.rest.bean.EmployeeList</value>

</list>

</property>

</bean>

<bean id=*"employees"* class=

*"org.springframework.web.servlet.view.xml.MarshallingView"*>

<constructor-arg ref=*"jaxbMarshaller"* />

</bean>

<bean id=*"viewResolver"* class=

*"org.springframework.web.servlet.view.BeanNameViewResolver"* />

上面的代码中：

**Component-scan**

启用对带有 Spring 注释的类进行自动扫描  
在实践中，它将检查控制器类中所定义的 @Controller 注释。

**DefaultAnnotationHanlderMappings 和 AnnotationMethodHandlerAdapter**

使用 @ReqeustMapping 注释的类或函数的 beans 由 Spring 处理  
这个注释将在下一节进行详细介绍。

**Jaxb2Mashaller**

定义使用 JAXB 2 进行对象 XML 映射（OXM）的编组器（marshaller）和解组器（unmarshaller

**MashallingView**

定义一个使用 Jaxb2Mashaller 的 XML 表示 view

**BeanNameViewResolver**

使用用户指定的 bean 名称定义一个视图解析器   
本例将使用名为 “employees” 的 MarshallingView。

这样就完成了 Spring 的相关配置。下一步是编写一个控制器来处理用户请求。清单 3 显示的是控制器类。

**清单 3. dw.spring3.rest.controller.EmployeeController**

@Controller

**publicclass** EmployeeController {

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

**public** ModelAndView getEmployee(@PathVariable String id) {

Employee e = employeeDS.get(Long.*parseLong*(id));

**returnnew** ModelAndView(*XML\_VIEW\_NAME*, "object", e);

}

}

@RequestMapping 注释是 Spring REST 特性的关键所在。它指定所注释的方法将处理哪个 HTTP 方法（RequestMethod.GET）和哪个 URI（/employee/{id}）。注意：

* 对于 {id} 占位符，使用 @PathVariable 注释可以将 {} 内的值注入到函数的参数。
* XML\_VIEW\_NAME 为 employees，这是 rest-servlet.xml 中定义的视图名称。
* employeeDS 是一个基于内存的数据源，它的实现已经超出本文写作范围。

将 Web 应用发布到您的 Tomcat 上。这时，您可以打开浏览器，然后输入 http://<host>:<port>/<appcontext>/service/employee/1。浏览器就会显示一个 ID 为 1 的员工信息的 XML 视图。

请继续阅读以了解更多关于 Spring REST 支持的特性。

**方法**

资源操作是通过 HTTP 方法实现的，如 GET、POST、PUT 和 DELETE。您在前面已经了解了如何使用 GET 方法查询员工信息。现在我们将介绍 POST、PUT 和 DELETE。

通过使用 @RequestMapping 注释的功能，处理不同方法的代码是非常相似的。清单 4 显示了 EmployeeController 的代码片断。

**清单 4. dw.spring3.rest.controller.EmployeeController**

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

**public** ModelAndView addEmployee(@RequestBody String body) {

Source source = **new** StreamSource(**new** StringReader(body));

Employee e = (Employee) jaxb2Mashaller.unmarshal(source);

employeeDS.add(e);

**returnnew** ModelAndView(*XML\_VIEW\_NAME*, "object", e);

}

@RequestMapping(method=RequestMethod.*PUT*, value="/employee/{id}")

**public** ModelAndView updateEmployee(@RequestBody String body) {

Source source = **new** StreamSource(**new** StringReader(body));

Employee e = (Employee) jaxb2Mashaller.unmarshal(source);

employeeDS.update(e);

**returnnew** ModelAndView(*XML\_VIEW\_NAME*, "object", e);

}

@RequestMapping(method=RequestMethod.*DELETE*, value="/employee/{id}")

**public** ModelAndView removeEmployee(@PathVariable String id) {

employeeDS.remove(Long.*parseLong*(id));

List<Employee> employees = employeeDS.getAll();

EmployeeList list = **new** EmployeeList(employees);

**returnnew** ModelAndView(*XML\_VIEW\_NAME*, "employees", list);

}

在上面的代码中：

* RequestMethod.<Method> 的值确定所注释的函数应该处理哪个 HTTP 方法。
* 通过 @RequestBody，HTTP 请求的主体内容可以作为一个参数注入。

在本例中，主体内容是表示员工信息的 XML 数据。我们使用 JAXB 2 来将 XML 解组为 Java Bean，然后将它存储。一个示例请求主体可以是：

<employee><id>3</id><name>guest</name></employee>

* 其他可以注入到函数参数的有用的注释有 @PathVariable、@RequestParm 等等。Spring 文档有完整列表的注释（见 [参考资料](http://www.ibm.com/developerworks/cn/web/wa-spring3webserv/#resources)）。

**资源集合**

通常，您还需要操作批量的资源。例如，您可能希望获取所有员工的信息而不只是一个员工的信息。您可以采取类似于之前情况的方法实现；您所需要做的修改就是将 URI 从 /employee 修改成 /employees。员工的复数形式能够正确反映批量的语义。清单 5 显示了这种实现方法。

**清单 5. EmployeeController 的 getAllEmployees**

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

**public** ModelAndView getEmployees() {

List<Employee> employees = employeeDS.getAll();

EmployeeList list = **new** EmployeeList(employees);

**returnnew** ModelAndView(*XML\_VIEW\_NAME*, "employees", list);

}

您需要为 Employee 集合声明一个包装类。这个包装类是 JAXB 2 所需要的，因为它无法正确地编组 java.util.List 类。清单 6 显示了EmployeeList 类。

**清单 6. dw.spring3.rest.bean.EmployeeList**

@XmlRootElement(name="employees")

**publicclass** EmployeeList {

**privateint** count;

**private** List<Employee> employees;

**public** EmployeeList() {}

**public** EmployeeList(List<Employee> employees) {

**this**.employees = employees;

**this**.count = employees.size();

}

**publicint** getCount() {

**return** count;

}

**publicvoid** setCount(**int** count) {

**this**.count = count;

}

@XmlElement(name="employee")

**public** List<Employee> getEmployees() {

**return** employees;

}

**publicvoid** setEmployees(List<Employee> employees) {

**this**.employees = employees;

}

}

**内容协商**

REST 服务的另一个常用特性是它们能够根据请求产生不同的表示。例如，如果客户端请求所有员工的 HTML/text 表示方式，那么服务器就应该为用户产生一个符合语法规则的 HTML 页面。如果客户端请求的是员工的 application/XML 表示方式，那么服务器就应该产生一个 XML 结果。其他受欢迎的表示方式还有 ATOM 和 PDF。

Spring 3 引入了一个名为 ContentNegotiatingViewResolver 的新视图解析器。它可以根据请求的内容类型（请求头中的 Accept 属性）或 URI 后缀来切换视图解析器。下面的例子使用 ContentNegotiatingViewResolver 来实现多种表示方式的支持。

在 rest-servlet.xml 文件中，用注释去掉原来定义的 viewResolver。而使用 ContentNegotiatingViewResolver 来替代它，如 清单 7 所示。

**清单 7. 定义内容协商**

<bean class=*"org.springframework.web.servlet.view.ContentNegotiatingViewResolver"*>

<property name=*"mediaTypes"*>

<map>

<entry key=*"xml"* value=*"application/xml"*/>

<entry key=*"html"* value=*"text/html"*/>

</map>

</property>

<property name=*"viewResolvers"*>

<list>

<bean class=*"org.springframework.web.servlet.view.BeanNameViewResolver"*/>

<bean class=*"org.springframework.web.servlet.view.UrlBasedViewResolver"*>

<property name=*"viewClass"* value=

*"org.springframework.web.servlet.view.JstlView"*/>

<property name=*"prefix"* value=*"/WEB-INF/jsp/"*/>

<property name=*"suffix"* value=*".jsp"*/>

</bean>

</list>

</property>

</bean>

这个定义显示了处理两种请求内容的支持：application/xml 和 text/html。这段代码也定义了两个视图解析器：其中BeanNameViewResolver 是负责处理 application/xml 的，而另一个 UrlBasedViewResolver 则负责处理 text/html。

在实践中，当您在浏览器上输入 http://<host>:<port>/<appcontext>/service/employees，那么它就会请求 text/html 形式的所有员工信息。然后 UrlBasedViewResolver 会进行解析，而 Spring 将会选择 /WEB-INF/jsp/employees.jsp 作为返回的视图。当您添加请求头Accept:application/xml 并再发起请求时，那么 BeanNameViewResolver 就会进行解析。如 [清单 5](http://www.ibm.com/developerworks/cn/web/wa-spring3webserv/#listing5) 中的代码，它将使用一个名为employees 的视图来表示，它就是所定义的 JAXB 2 编组器视图。

getAllEmployees() 的控制器代码不需要修改。employees.jsp 将会使用名为 employees 的模型对象来渲染。清单 8 显示了 employees.jsp 的代码片断。

**清单 8. /WEB-INF/jsp 中的 employees.jsp**

<table border=*1*>

<thead><tr>

<th>ID</th>

<th>Name</th>

<th>Email</th>

</tr></thead>

<c:forEach var=*"employee"* items="${employees.employees}">

<tr>

<td>${employee.id}</td>

<td>${employee.name}</td>

<td>${employee.email}</td>

</tr>

</c:forEach>

</table>

[**回页首**](http://www.ibm.com/developerworks/cn/web/wa-spring3webserv/#ibm-pcon)

**与 REST 服务通信的客户端**

现目前为止，您已经开发一个简单的支持对员工信息的 CRUD（增删查改）操作的 RESTful Web Service。接下来这一节将介绍如何与这个服务进行通信。您将使用 curl 来测试这个 REST 服务。

您也可以使用名为 RESTClient 的 Firefox 插件来测试 REST 服务。它很容易使用且带有良好的 UI。[参考资料](http://www.ibm.com/developerworks/cn/web/wa-spring3webserv/#resources) 有关于它的下载信息。

**使用 curl**

Curl 是一个流行的能以 HTTP 和 HTTPS 协议向服务器发送请求的命令行工具。Curl 是 Linux® 和 Mac® 上的一个内置工具。对于 Windows® 平台，您可以另外下载这个工具（见 [参考资料](http://www.ibm.com/developerworks/cn/web/wa-spring3webserv/#resources)）。

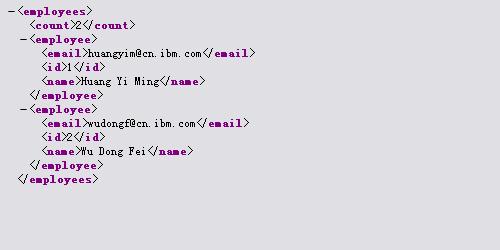
要初始化查询所有员工信息的第一个 curl 命令，您可以输入：

curl –HAccept:application/xml

http://localhost:8080/rest/service/employees

它的响应将是 XML 格式的，并且包含所有的员工信息，如图 1 所示。

**图 1. XML 方式表示的所有员工信息**



您也可以在浏览器上尝试访问相同的 URL。这时，头信息中的 Accept 为 text/html，所以浏览器会显示 employees.jsp 中定义的一个表格。图 2 显示的是这种情况。

**图 2. HTML 方式表示的所有员工信息**



要将一个新的员工信息 POST 到服务器上，我们可以使用下面的代码。[清单 4](http://www.ibm.com/developerworks/cn/web/wa-spring3webserv/#listing4) 中的 addEmployee() 代码将会使用请求体并将它解组为Employee 对象。

curl -X POST -HContent-type:application/xml --data

"<employee><id>3</id><name>guest</name><email>guest@ibm.com</employee>"

http://localhost:8080/rest/service/employee

这样就添加了一个新的员工信息。您可以使用第一个例子来验证员工列表。

PUT 方法类似于 POST。

curl -X PUT -HContent-type:application/xml --data

"<employee><id>3</id><name>guest3</name><email>guest3@ibm.com</employee>"

http://localhost:8080/rest/service/employee/3

上面的代码修改了 ID 为 3 的员工数据。

[**回页首**](http://www.ibm.com/developerworks/cn/web/wa-spring3webserv/#ibm-pcon)

**结束语**

现在 Spring 3 已经在它的 MVC 层上支持 REST 了，您可以使用 Spring API 和注释来开发 RESTful Web Services 了。本文的示例向您介绍了如何使用一些能够帮您简化开发 Java 服务器端 RESTful Web Services 的 Spring 3 的新特性。