**使用 Jersey 和 Apache Tomcat 构建 RESTful Web 服务**

Representational state transfer（REST）在 2000 年由 Roy Fielding 在博士论文中提出。但是，在 Java™ 社区中，直到 2008 年 JSR 311(JAX-RS) 规范定稿后才将其标准化。第一个版本的参考实现甚至更晚。在本文中，我将介绍 Jersey —— JSR 311 的参考实现，描述其必要的 API 和注释。我还将展示如何通过在 Apache Tomcat 中集成 Jersey 从 servlet 风格的服务转型到 RESTful 服务。

**RESTful Web 服务简介**

REST 在 2000 年由 Roy Fielding 在博士论文中提出，他是 HTTP 规范 1.0 和 1.1 版的首席作者之一。

REST 中最重要的概念是*资源（resources）*，使用全球 ID（通常使用 URI）标识。客户端应用程序使用 HTTP 方法（GET/ POST/ PUT/ DELETE）操作资源或资源集。RESTful Web 服务是使用 HTTP 和 REST 原理实现的 Web 服务。通常，RESTful Web 服务应该定义以下方面：

* Web 服务的基/根 URI，比如 http://host/<appcontext>/resources。
* 支持 MIME 类型的响应数据，包括 JSON/XML/ATOM 等等。
* 服务支持的操作集合（例如 POST、GET、PUT 或 DELETE）。

表 1 演示了典型 RESTful Web 服务中使用的资源 URI 和 HTTP 方法。（[参考资料](http://www.ibm.com/developerworks/cn/web/wa-aj-tomcat/#resources) 提供了有关 RESTful Web 服务的更多介绍和设计考虑事项。）

**表 1. RESTful Web 服务示例**

| **方法/资源** | **资源集合， URI 如： http://host/<appctx>/resources** | **成员资源，URI 如： http://host/<appctx>/resources/1234** |
| --- | --- | --- |
| **GET** | 列出资源集合的所有成员。 | 检索标识为 1234 的资源的表示形式。 |
| **PUT** | 使用一个集合更新（替换）另一个集合。 | 更新标记为 1234 的数字资源。 |
| **POST** | 在集合中创建数字资源，其 ID 是自动分配的。 | 在下面创建一个子资源。 |
| **DELETE** | 删除整个资源集合。 | 删除标记为 1234 的数字资源。 |

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

**JSR 311 (JAX-RS) 和 Jersey**

JSR 311 或 JAX-RS（用于 RESTful Web Services 的 Java API）的提议开始于 2007 年，1.0 版本到 2008 年 10 月定稿。目前，JSR 311 版本 1.1 还处于草案阶段。该 JSR 的目的是提供一组 API 以简化 REST 样式的 Web 服务的开发。

在 JAX-RS 规范之前，已经有 Restlet 和 RestEasy 之类的框架，可以帮助您实现 RESTful Web 服务，但是它们不够直观。Jersey 是 JAX-RS 的参考实现，它包含三个主要部分。

* 核心服务器（Core Server）：通过提供 JSR 311 中标准化的注释和 API 标准化，您可以用直观的方式开发 RESTful Web 服务。
* 核心客户端（Core Client）：Jersey 客户端 API 帮助您与 REST 服务轻松通信。
* 集成（Integration）：Jersey 还提供可以轻松集成 Spring、Guice、Apache Abdera 的库。

在本文的以下部分，我介绍了所有这些组件，但是更关注核心服务器。

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

**构建 RESTful Web 服务**

我将从可以集成到 Tomcat 的 “hello world” 应用程序开始。该应用程序将带领您完成设置环境的过程，并涉及 Jersey 和 JAX-RS 的基础知识。

然后，我将介绍更加复杂的应用程序，深入探讨 JAX-RS 的本质和特性，比如多个 MIME 类型表示形式支持、JAXB 支持等。我将从样例中摘取一些代码片段来介绍重要的概念。

**Hello World：第一个 Jersey Web 项目**

要设置开发环境，您需要以下内容（见 [参考资料](http://www.ibm.com/developerworks/cn/web/wa-aj-tomcat/#resources) 中的下载）：

* IDE：Eclipse IDE for JEE (v3.4+) 或 IBM Rational Application Developer 7.5
* Java SE5 或更高版本
* Web 容器：Apache Tomcat 6.0（Jetty 和其他也可以）
* Jersey 库：Jersey 1.0.3 归档，包含所有必需的库

**设置 Jersey 的环境**

首先，为 Eclipse 上的 Tomcat 6.0 创建服务器运行时。这是用于 RESTful Web 应用程序的 Web 容器。然后创建一个名为 “Jersey” 应用程序，并将目标运行时指定为 Tomcat 6.0。

最后，从 Jersey 开发包中将以下库复制到 WEB-INF 下的库目录：

* 核心服务器：jersey-core.jar，jersey-server.jar，jsr311-api.jar，asm.jar
* 核心客户端：（用于测试）jersey-client.jar
* JAXB 支持：（在高级样例中使用）jaxb-impl.jar，jaxb-api.jar，activation.jar，stax-api.jar，wstx-asl.jar
* JSON 支持：（在高级样例中使用）jersey-json.jar

**开发 REST 服务**

现在，您已经设置好了开发第一个 REST 服务的环境，该服务对客户端发出 “Hello”。

要 做到这一点，您需要将所有的 REST 请求发送到 Jersey 容器 —— 在应用程序的 web.xml 文件中定义 servlet 调度程序（参见清单 1）。除了声明 Jersey servlet 外，它还定义一个初始化参数，指示包含资源的 Java 包。

**清单 1. 在 web.xml 文件中定义 Jersey servlet 调度程度**

<servlet>

<servlet-name>Jersey REST Service</servlet-name>

<servlet-class>

com.sun.jersey.spi.container.servlet.ServletContainer

</servlet-class>

<init-param>

<param-name>com.sun.jersey.config.property.packages</param-name>

<param-value>sample.hello.resources</param-value>

</init-param>

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

</servlet>

<servlet-mapping>

<servlet-name>Jersey REST Service</servlet-name>

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

</servlet-mapping>

现在您将编写一个名为 HelloResource 的资源，它接受 HTTP GET 并响应 “Hello Jersey”。

**清单 2. sample.hello.resources 包中的 HelloResource**

@Path("/hello")

public class HelloResource {

@GET

@Produces(MediaType.TEXT\_PLAIN)

public String sayHello() {

return "Hello Jersey";

}

}

该代码中有几个地方需要强调：

* 资源类（Resource Class）：注意，资源类是一个简单的 Java 对象 (POJO)，可以实现任何接口。这增加了许多好处，比如可重用性和简单。
* 注释（Annotation）：在 javax.ws.rs.\* 中定义，是 JAX-RS (JSR 311) 规范的一部分。
* @Path：定义资源基 URI。由上下文根和主机名组成，资源标识符类似于 http://localhost:8080/Jersey/rest/hello。
* @GET：这意味着以下方法可以响应 HTTP GET 方法。
* @Produces：以纯文本方式定义响应内容 MIME 类型。

**测试 Hello 应用程序**

要测试应用程序，可以打开您的浏览器并输入 URL http://<host>:<port>/<appctx>/rest/hello。您将看到响应 “Hello Jersey”。这非常简单，使用注释处理请求、响应和方法。

以下部分将涉及 JAX-RS 规范的必要部分，使用 Contacts 示例应用程序中的代码片段进行介绍。您可以在源代码包中找到这个高级样例的所有代码（参见 [下载](http://www.ibm.com/developerworks/cn/web/wa-aj-tomcat/#download)）。

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

**资源**

资源是组成 RESTful Web 服务的关键部分。您可以使用 HTTP 方法（如 GET、POST、PUT 和 DELETE）操作资源。应用程序中的所有内容都是资源：员工、联系人、组织等。在 JAX-RX 中，资源通过 POJO 实现，使用 @Path 注释组成其*标识符*。资源可以有子资源。在这种情况下，父资源是资源集合，子资源是成员资源。

在样例 Contacts 应用程序中，您将操作个人联系人和联系人集合。ContactsResource 是 /contacts URI 组成的集合资源，ContactResource 是 /contacts/{contactId} URI 组成的成员资源。下划线 JavaBean 是一个简单的 Contact 类，使用 id、名称和地址作为成员字段。参见清单 3 和清单 4 了解详情。您还可以从本文最后下载完整的代码包（参见 [下载](http://www.ibm.com/developerworks/cn/web/wa-aj-tomcat/#download)）。

**清单 3. ContactsResource**

@Path("/contacts")

public class ContactsResource {

@Context

UriInfo uriInfo;

@Context

Request request;

@GET

@Produces({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

public List<Contact> getContacts() {

List<Contact> contacts = >new ArrayList<Contact>();

contacts.addAll( ContactStore.getStore().values() );

return contacts;

}

@Path("{contact}")

public ContactResource getContact(

@PathParam("contact") String contact) {

return new ContactResource(uriInfo, request, contact);

}

}

有几个有趣的地方需要注意。

* @Context： 使用该注释注入上下文对象，比如 Request、Response、UriInfo、ServletContext 等。
* @Path("{contact}")：这是 @Path 注释，与根路径 “/contacts” 结合形成子资源的 URI。
* @PathParam("contact")：该注释将参数注入方法参数的路径，在本例中就是联系人 id。其他可用的注释有 @FormParam、@QueryParam 等。
* @Produces：响应支持多个 MIME 类型。在本例和上一个示例中，APPLICATION/XML 将是默认的 MIME 类型。

您也许还注意到了，GET 方法返回定制 Java 对象而不是 String（纯文本），正如上一个 Hello World 示例所示。 JAX-RS 规范要求实现支持多个表示形式类型，比如 InputStream、byte[]、JAXB 元素、JAXB 元素集合等等，以及将其序列化为 XML、JSON 或纯文本作为响应的能力。下文我将提供更多有关表示形式技术的信息，尤其是 JAXB 元素表示形式。

**清单 4. ContactResource**

public class ContactResource {

@Context

UriInfo uriInfo;

@Context

Request request;

String contact;

public ContactResource(UriInfo uriInfo, Request request,

String contact) {

this.uriInfo = uriInfo;

this.request = request;

this.contact = contact;

}

@GET

@Produces({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

public Contact getContact() {

Contact cont = ContactStore.getStore().get(contact);

if(cont==null)

throw new NotFoundException("No such Contact.");

return cont;

}

}

ContactResource 的代码简单明了。注意以下内容：

* Representation Type Contact：Contact 是一个简单的 JavaBean，由 @XmlRootElement 注释，这使它可以表示为 XML 或 JSON。
* ContactStore：这是基于 HashMap 的内存数据存储库，其实现对于本文不重要。

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

**方法**

HTTP 方法映射到资源的 CRUD（创建、读取、更新和删除） 操作。尽管您可以做一些小修改，比如让 PUT 方法变成创建或更新，但基本的模式如下：

* HTTP GET：获取/列出/检索单个资源或资源集合。
* HTTP POST：新建资源。
* HTTP PUT：更新现有资源或资源集合。
* HTTP DELETE：删除资源或资源集合。

因为我已经介绍过 GET 方法，我将从 POST 开始说明。就像其他方法一样，我仍然使用 Contact 示例进行说明。

**POST**

通常通过填写表单创建新联系人。也就是说，HTML 表单将 POST 到服务器，服务器创建并维护新创建的联系人。清单 5 演示了该操作的服务器端逻辑。

**清单 5. 接受表单提交（POST）并新建一个联系人**

@POST

@Produces(MediaType.TEXT\_HTML)

@Consumes(MediaType.APPLICATION\_FORM\_URLENCODED)

public void newContact(

@FormParam("id") String id,

@FormParam("name") String name,

@Context HttpServletResponse servletResponse

) throws IOException {

Contact c = new Contact(id,name,new ArrayList<Address>());

ContactStore.getStore().put(id, c);

URI uri = uriInfo.getAbsolutePathBuilder().path(id).build();

Response.created(uri).build();

servletResponse.sendRedirect("../pages/new\_contact.html");

}

注意该示例的以下部分：

* @Consumes：声明该方法使用 HTML FORM。
* @FormParam：注入该方法的 HTML 属性确定的表单输入。
* @Response.created(uri).build()： 构建新的 URI 用于新创建的联系人（/contacts/{id}）并设置响应代码（201/created）。您可以使用 http://localhost:8080/Jersey/rest/contacts/<id> 访问新联系人。

**PUT**

我使用 PUT 方法更新现有资源。但是，也可以通过更新实现，或者像清单 6 中的代码片段展示的那样创建一个资源。

**清单 6. 接受 PUT 请求并创建或更新联系人**

@PUT

@Consumes(MediaType.APPLICATION\_XML)

public Response putContact(JAXBElement<Contact> jaxbContact) {

Contact c = jaxbContact.getValue();

return putAndGetResponse(c);

}

private Response putAndGetResponse(Contact c) {

Response res;

if(ContactStore.getStore().containsKey(c.getId())) {

res = Response.noContent().build();

} else {

res = Response.created(uriInfo.getAbsolutePath()).build();

}

ContactStore.getStore().put(c.getId(), c);

return res;

}

我还在本示例中包含了许多不同的概念，重点强调以下概念：

* Consume XML：putContact() 方法接受 APPLICATION/XML 请求类型，而这种输入 XML 将使用 JAXB 绑定到 Contact 对象。您将在下一节中找到客户端代码。
* 空响应带有不同的状态码：PUT 请求的响应没有任何内容，但是有不同的状态码。如果数据存储库中存在联系人，我将更新该联系人并返回 204/no content。如果没有新联系人，我将创建一个并返回 201/created。

**DELETE**

实现 DELETE 方法非常简单。示例请查看清单 7。

**清单 7. 删除其 ID 确定的联系人**

@DELETE

public void deleteContact() {

Contact c = ContactStore.getStore().remove(contact);

if(c==null)

throw new NotFoundException("No such Contact.");

}

**表示形式**

在上一节中，我介绍了几个表示形式类型。现在我将简要浏览一遍并深入探讨 JAXB 表示形式。其他受支持的表示形式有 byte[]、InputStream、File 等。

* String：纯文本。
* Response：一般 HTTP 响应，包含带有不同响应代码的定制内容。
* Void：带有 204/no content 状态码的空响应。
* Resource Class：将流程委托给该资源类。
* POJO：使用 @XmlRootElement 注释的 JavaBean，这让它成为一个 JAXB bean，可以绑定到 XML。
* POJO 集合：JAXB bean 集合。

JAX-RS 支持使用 JAXB (Java API for XML Binding) 将 JavaBean 绑定到 XML 或 JSON，反之亦然。JavaBean 必须使用 @XmlRootElement 注释。清单 8 使用 Contact bean 作为示例。没有明确 @XmlElement 注释的字段将包含一个名称与之相同的 XML 元素。清单 9 显示了用于一个 Contact bean 的序列化 XML 和 JSON 表示形式。联系人集合的表示形式与此相同，默认使用 <Contacts> 作为包装器元素。

**清单 8. Contact bean**

@XmlRootElement

public class Contact {

private String id;

private String name;

private List<Address> addresses;

public Contact() {}

public Contact(String id, String name, List<Address> addresses) {

this.id = id;

this.name = name;

this.addresses = addresses;

}

@XmlElement(name="address")

public List<Address> getAddresses() {

return addresses;

}

public void setAddresses(List<Address> addresses) {

this.addresses = addresses;

}

// Omit other getters and setters

}

**清单 9. 一个 Contact 的表示形式**

**XML representation:**

<contact>

<address>

<city>Shanghai</city>

<street>Long Hua Street</street>

</address>

<address>

<city>Shanghai</city>

<street>Dong Quan Street</street>

</address>

<id>huangyim</id>

<name>Huang Yi Ming</name>

</contact>

**JSON representation:**

{"contact":[{"address":[{"city":"Shanghai","street":"Long

Hua Street"},{"city":"Shanghai","street":"Dong Quan

Street"}],"id":"huangyim","name":"Huang Yi Ming"}]}

对于使用 JAXB 的更高主题，请查看 [参考资料](http://www.ibm.com/developerworks/cn/web/wa-aj-tomcat/#resources) 中的项目主页。

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

**与 REST 服务通讯的客户端**

在目前为止的示例中，我开发了一个支持 CRUD 的 RESTful Web 服务。现在我开始解释如何使用 curl 和 Jersey 客户端 API 与该 REST 服务通讯。这样一来，我可以测试服务器端代码，并介绍更多有关客户端技术的信息。

**使用 curl 与 REST 服务通讯**

Curl 是一个流行的命令行工具，可以向使用 HTTP 和 HTTPS 协议的服务器发送请求。这是一个与 RESTful Web 服务通讯的好工具，因为它可以通过任何 HTTP 方法发送内容。Curl 已经在 Linux 和 Mac 中自带了，并且有一个实用工具，可以在 Windows® 平台上进行安装（见 [参考资料](http://www.ibm.com/developerworks/cn/web/wa-aj-tomcat/#resources)）。

现在，我们初始化获取所有联系人的第一个 curl 命令。您可以参考 [清单 3](http://www.ibm.com/developerworks/cn/web/wa-aj-tomcat/#list3) 获取服务器端代码。

curl http://localhost:8080/Jersey/rest/contacts

响应将使用 XML 并包含所有联系人。

注意，getContacts() 方法还生成一个 application/json MIME 类型响应。您还可以请求该类型的内容。

curl –HAccept:application/json http://localhost:8080/Jersey/rest/contacts

响应将是一个包含所有联系人的 JSON 字符串。

现在，我将 PUT 一个新的联系人。注意，[清单 6](http://www.ibm.com/developerworks/cn/web/wa-aj-tomcat/#list6) 中的 putContact() 方法接受 XML 并使用 JAXB 将 XML 绑定到 Contact 对象。

curl -X PUT -HContent-type:application/xml --data "<contact><id>foo</id>

<name>bar</name></contact>" http://localhost:8080/Jersey/rest/contacts/foo

一个通过 “foo” 识别的新联系人将添加到联系人存储库。您可以使用 URI /contacts 或 /contacts/foo 验证联系人集合或单个联系人。

**使用 Jersey Client 与 REST 服务通讯**

Jersey 还提供了一个客户端库，帮助您与服务器通讯并对 RESTful 服务进行单元测试。该库是一个一般实现，可以整合任何 HTTP/HTTPS-based Web 服务。

客户端的核心类是 WebResource 类。您可以使用该类根据根 URI 构建一个请求 URL，然后发送请求并获取响应。清单 10 展示了如何创建 WebResource 实例。注意 WebResource 是一个大对象，因此只创建一次。

**清单 10. 创建 WebResource 实例**

Client c = Client.*create*();

WebResource r=c.resource("http://localhost:8080/Jersey/rest/contacts");

第一个 Jersey 客户端示例将发送 GET 请求获取所有联系人并打印响应状态码和响应内容，参见清单 11。

**清单 11. GET 所有联系人并打印响应**

ClientResponse response = r.get(ClientResponse.class);

System.out.println( response.getStatus() );

System.out.println( response.getHeaders().get("Content-Type") );

String entity = response.getEntity(String.class);

System.out.println(entity);

清单 12 展示了另一个创建通过 “foo” 识别的新联系人的示例。

**清单 12. 创建一个联系人**

Address[] addrs = {

new Address("Shanghai", "Ke Yuan Street")

};

Contact c = new Contact("foo", "Foo Bar", Arrays.asList(addrs));

ClientResponse response = r

.path(c.getId())

.accept(MediaType.APPLICATION\_XML)

.put(ClientResponse.class, c);

System.out.println(response.getStatus());

注意 WebResource 实例的 API。它构建 URI，设置请求头，并在一行代码中调用请求。内容（Contact 对象）将自动绑定到 XML。

清单 13 展示了检索通过 “foo” 识别的联系人（已上一个示例中创建）的最后一个示例然后删除该联系人。

**清单 13. 检索 “foo” 联系人并删除**

GenericType<JAXBElement<Contact>> generic = new GenericType<JAXBElement<Contact>>() {};

JAXBElement<Contact> jaxbContact = r

.path("foo")

.type(MediaType.APPLICATION\_XML)

.get(generic);

Contact contact = jaxbContact.getValue();

System.out.println(contact.getId() + ": " + contact.getName());

ClientResponse response = r.path("foo").delete(ClientResponse.class);

System.out.println(response.getStatus());

注意，当您想获取 JAXB bean 响应时，您需要使用 Java 2 Platform, Standard Edition (J2SE) 中引入的范型特性。

使用 Jersey 客户端练习这些示例。您可以在资源包中找到更多样例代码（见 [下载](http://www.ibm.com/developerworks/cn/web/wa-aj-tomcat/#download)）。还可以参考 Jersey 网站查看更多信息（见 [参考资料](http://www.ibm.com/developerworks/cn/web/wa-aj-tomcat/#resources)）。

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

**结束语**

Jersey 可以使用 Jersey 集成库与其他框架或实用工具库集成。目前，Jersey 可以集成 Spring、Guice，还支持 ATOM 表示形式与 apache-adbera 的集成。在 Jersey 项目主页可以找到 API 和入门指南。