# **[Web service是什么？](http://www.cnblogs.com/chenying99/archive/2012/03/30/2424508.html)**

## **一、Web service的概念**

想要理解Web service，必须先理解什么是Service（服务）。

传统上，我们把计算机后台程序（Daemon）提供的功能，称为"服务"（service）。比如，让一个杀毒软件在后台运行，它会自动监控系统，那么这种自动监控就是一个"服务"。**通俗地说，"服务"就是计算机可以提供的某一种功能。**

根据来源的不同，"服务"又可以分成两种：一种是**"本地服务"**（使用同一台机器提供的服务，不需要网络），另一种是"**网络服务"**（使用另一台计算机提供的服务，必须通过网络才能完成）。

举例来说，我现在有一批图片，需要把它们的大小缩小一半。那么，我们可以把"缩放图片"看成是一种服务。你可以使用"本地服务"，在自己计算机上用软件缩小图片，也可以使用"网络服务"，将图片上传到某个网站，让服务器替你缩小图片，完成后再通过网络送回给你。这就好比，一件事你可以自己做，也可以交给另一个人去做。肚子饿了，你可以自己做饭，也可以打电话去订一份比萨，让店家替你做好送上门。

**"网络服务"（Web Service）的本质，就是通过网络调用其他网站的资源。**

举例来说，去年我写过一个"[四川大地震图片墙](http://www.ruanyifeng.com/php/earthquake/)"，它能动态显示关于四川地震的最新图片。但是，所有的图片都不是储存在我的服务器上，而是来自[flickr.com](http://flickr.com/)。我只是发出一个动态请求，要求flickr.com向我提供图片。这种情况下，flickr.com提供的就是一种Web service。如果我把图片都存放在本地服务器，不调用flickr.com，那么我就是在使用"本地服务"。

所以，Web service让你的网站可以使用其他网站的资源，比如在网页上显示天气、地图、twitter上的最新动态等等。

## **二、Web Service架构和云**

如果一个软件的主要部分采用了"网络服务"，即它把存储或计算环节"外包"给其他网站了，那么我们就说这个软件属于Web Service架构。

**Web Service架构的基本思想，就是尽量把非核心功能交给其他人去做，自己全力开发核心功能。**比如，如果你要开发一个相册软件，完全可以使用Flickr的网络服务，把相片都储存到它上面，你只要全力做好相册本身就可以了。总体上看，凡是不属于你核心竞争力的功能，都应该把它"外包"出去。

**最近很红的"云计算"（cloud computing）或者"云服务"（cloud services），实际上就是Web Service的同义词，不过更形象一些罢了。它们不说你把事情交给其他计算机去做，而说你把事情交给"云"去做。**

## **三、本地服务的缺陷**

"网络服务"是未来软件开发和使用的趋势，本地服务将用得越来越少，主要因为以下三个原因：

**\*　本地资源不足。很多数据和资料，本地得不到，只有向其他网站要。**

**\*　成本因素。本地提供服务，往往是不经济的，使用专业网站的服务更便宜。这里面涉及硬件和人员两部分，即使你买得起硬件，专门找一个人管理系统，也是很麻烦的事。**

**\*　可移植性差。如果你想把本机的服务，移植到其他机器上，往往很困难，尤其是在跨平台的情况下。**

## **四、Web Service的优势**

除了本地服务的缺点以外，Web Service还有以下的优越性：

*\*　平台无关。不管你使用什么平台，都可以使用Web service。*

*\*　编程语言无关。只要遵守相关协议，就可以使用任意编程语言，向其他网站要求Web service。这大大增加了web service的适用性，降低了对程序员的要求。*

*\*　对于Web service提供者来说，部署、升级和维护Web service都非常单纯，不需要考虑客户端兼容问题，而且一次性就能完成。*

*\*　对于Web service使用者来说，可以轻易实现多种数据、多种服务的聚合（mashup），因此能够做出一些以前根本无法想像的事情。*

## **五、Web service的发展趋势**

根据我的观察，目前Web service有这样几种发展趋势。

*\*　在使用方式上，RPC和soap的使用在减少，Restful架构占到了主导地位。*

*\*　在数据格式上，XML格式的使用在减少，json等轻量级格式的使用在增多。*

*\*　在设计架构上，越来越多的第三方软件让用户在客户端（即浏览器），直接与云端对话，不再使用第三方的服务器进行中转或处理数据。*

# [**Web Service(SOAP)实现分布式服务的基本原理**](http://www.cnblogs.com/chenying99/p/3213517.html)

## Web Service(SOAP)基本概念

Web Service也叫XML Web Service WebService是一种可以接收从Internet或者Intranet上的其它系统中传递过来的请求，轻量级的独立的通讯技术。是:通过SOAP在Web上提供的软件服务，使用WSDL文件进行说明，并通过UDDI进行注册。

XML：(Extensible Markup Language)扩展型可标记语言。面向短期的临时数据处理、面向万维网络，是Soap的基础。

Soap：(Simple Object Access Protocol)简单对象存取协议。是XML Web Service 的通信协议。当用户通过UDDI找到你的WSDL描述文档后，他通过可以SOAP调用你建立的Web服务中的一个或多个操作。SOAP是XML文档形式的调用方法的规范，它可以支持不同的底层接口，像HTTP(S)或者SMTP。

WSDL：(Web Services Description Language) WSDL 文件是一个 XML 文档，用于说明一组 SOAP 消息以及如何交换这些消息。大多数情况下由软件自动生成和使用。

UDDI (Universal Description, Discovery, and Integration) 是一个主要针对Web服务供应商和使用者的新项目。在用户能够调用Web服务之前，必须确定这个服务内包含哪些商务方法，找到被调用的接口定义，还要在服务端来编制软件，UDDI是一种根据描述文档来引导系统查找相应服务的机制。UDDI利用SOAP消息机制（标准的XML/HTTP）来发布，编辑，浏览以及查找注册信息。它采用XML格式来封装各种不同类型的数据，并且发送到注册中心或者由注册中心来返回需要的数据。

## 调用原理

Web服务有两层含义：1、是指封装成单个实体并发布到网络上的功能集合体；2、是指功能集合体被调用后所提供的服务。简单地讲，Web服务是一个URL资源，客户端可以通过编程方式请求得到它的服务，而不需要知道所请求的服务是怎样实现的，这一点与传统的分布式组件对象模型不同。

Web服务的体系结构是基于Web服务提供者、Web服务请求者、Web服务中介者三个角色和发布、发现、绑定三个动作构建的。简单地说，Web服务提供者就是Web服务的拥有者，它耐心等待为其他服务和用户提供自己已有的功能；Web服务请求者就是Web服务功能的使用者，它利用SOAP消息向Web服务提供者发送请求以获得服务;Web服务中介者的作用是把一个Web服务请求者与合适的Web服务提供者联系在一起，它充当管理者的角色，一般是UDDI。这三个角色是根据逻辑关系划分的，在实际应用中，角色之间很可能有交叉：一个Web服务既可以是Web服务提供者，也可以是Web服务请求者，或者二者兼而有之。显示了Web服务角色之间的关系:其中，“发布”是为了让用户或其他服务知道某个Web服务的存在和相关信息;“查找（发现）”是为了找到合适的Web服务;“绑定”则是在提供者与请求者之间建立某种联系。

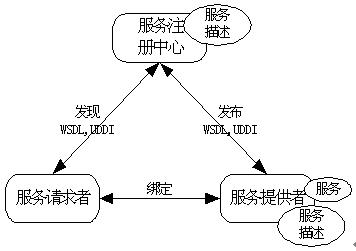


图2-1 Web service的体系结构

实现一个完整的Web服务包括以下步骤：

◆ Web服务提供者设计实现Web服务，并将调试正确后的Web服务通过Web服务中介者发布，并在UDDI注册中心注册； （发布）

◆ Web服务请求者向Web服务中介者请求特定的服务，中介者根据请求查询UDDI注册中心，为请求者寻找满足请求的服务； （发现）

◆ Web服务中介者向Web服务请求者返回满足条件的Web服务描述信息，该描述信息用WSDL写成，各种支持Web服务的机器都能阅读；（发现）

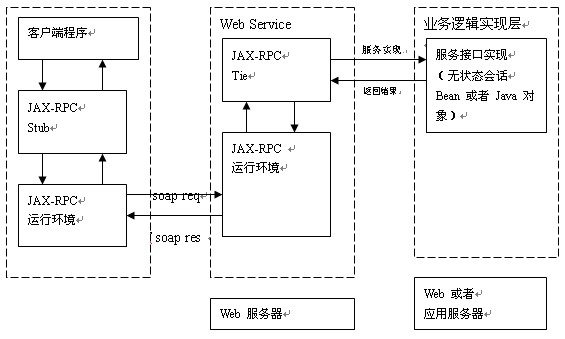
◆ 利用从Web服务中介者返回的描述信息生成相应的SOAP消息，发送给Web服务提供者，以实现Web服务的调用；（绑定）

◆ Web服务提供者按SOAP消息执行相应的Web服务，并将服务结果返回给Web服务请求者。（绑定）

简单的说， 就是客户端根据WSDL 生成 SOAP 的请求消息， 通过 HTTP 传输方式（也可以是其它传输方式， 如 FTP 或STMP 等，目前 HTTP 传输方式已经成为 J2EE Web Service 的标准）传给对方， 服务方实现服务请求， [Web Service（Web服务）](http://www.webxml.com.cn/)将结果以 SOAP 的消息格式返回给客户端。

如果人工去创建和解析基于 XML 格式的 SOAP 消息还是一个非常复杂的过程， 这样 JAX-RPC 应时而生， 他实现了J2EE Web Sercive  的远程分布式调用。

JAX - RPC ：Java APIs for XML-Based Remote Procedure Call. 它本质上是另一种RMI。 只是 JAX-RPC 以 SOAP 作为通信协议， RMI 以 RMI- IIOP或者 RMI - JRMP为通信协议。



客户端需要根据 WSDL 创建客户端 Java 程序， 其中包括 Stub 程序。 客户端调用相应的Stub 程序， 进一步调用JAX- RPC 运行环境创建 SOAP 请求消息， 通过 HTTP 传输给服务器端。

Web 服务器端的JAX-RPC 运行环境在收到 SOAP 请求消息后， 对 SOAP 的 XML 内容进行解析， 再通过 Tie 来调用服务接口实现类。（无状态会话 Bean 或者 Java  对象） ，得到结果后， 创建SOAP 响应消息返回给客户端。

**客户端基于JAX- RPC 实现远程分布式调用的基本原理**

（1）通过 JAX-RPC 创建 SOAP 请求消息  
（2）通过 JAX- RPC 将 SOAP 请求消息送到服务地址  
（3）通过 JAX- RPC 解析 SOAP 请求消息

**服务器端基于 JAX-RPC 实现远程分布式调用的基本原理**

服务器端的 JAX-RPC 的运行环境在收到了基于 XML 格式的SOAP 请求消息后， 会调用服务器端的 JAX-RPC Tie 对象的相应服务接口方法checkUserLogin, 将上面的基于 XML 格式的 SOAP 请求消息中的参数值映射为 Java 对象类传给 Tie 对象的接口方法， 将 loginName 和 password 都转化为 Java 的String 类型。 这是前述的 WSDL 中所定义的类型。

**利用设计模式**

设计模式在设计Webservice的时候显然可以起到相当大的作用。设计模式的主要目的就是为解决某些在类似环境下的相像问题提供已有的较为成熟的设计方案。在这里，只简单的提及一些很常用的模式，让我们了解到模式在Webservice中可以起到的作用。

Adapter ：为内部系统提供一个不同的接口

Façade：封装复杂的内部实现，提供一系列简单的接口

Proxy： 作为其他对象的代理，代替它提供服务  
   
Adapter 模式用于将一个组件的接口转化成客户所需要的样子，这里的客户就是Webservice。一个常见的情况就是将原有的老的系统包装成一个Webservice。比如现在使用的是J2EE的平台，而原来有一个C++的系统实现了某些功能，现在需要将它发布成Webservice，那么就需要利用JNI技术做一个Adapter，为原来的C++组件提供一个Java的接口，然后再转化为Webservice。  
   
Façade 模式用于构建粗粒度的服务，它包装了细粒度的服务，从而为复杂的系统提供了一个简单的接口。在J2EE中，Session Bean就象是一个Façade，而Entity Bean则是细粒度的服务。在Webservice中也一样，使用Façade模式可以将已有的组件的功能发挥殆尽。  
   
Proxy 模式用于充当其他对象的代理，类似于中间人的作用，将处理工作从一个对象传递到另一个对象。在Webservice中，它主要用于隐藏Soap消息构造的过程。也可以用于模拟对象（Mock Object）的创建。  
   
以上仅仅是一些可以用于Webservice开发的模式，如果你熟练的将这些模式应用于Webservice开发，你将会发现开发Webservice应用，将好像做一种特殊的面向对象设计。  
   
**安全**

Webservice为作为方便的服务被用广大领域使用的同时，也成为了黑客们的美食。在这里，本文将就目前对Webservice安全所能做的改进做简单介绍。

在Webservice中的安全主要分为以下三个方面。

传输      SSL/HTTPS 对连接加密，而不是传输数据

消息      数据加密(XML Encryption)   数字签名(XML-DSIG)

底层架构  利用应用服务安全机制  
   
传输时的安全是最容易被加入到你的Webservice应用中的，利用现有的SSL 和HTTPS协议，就可以很容易的获得连接过程中的安全。  
   
然而这种安全实现方法有两个弱点。一是它只能保证数据传输的安全，而不是数据本身的安全，数据一旦到达某地，那么就可以被任何人所查看。而在Webservice中，一份数据可能到达多个地方，而这份数据却不该被所有的接受者所查看。二是它提供的是要么全有要么全无的保护，你不能选择哪部分数据要被保护，而这种可选择性也是在Webservice中所常要用到的。  
   
第二层的保护是对于消息本身的保护。你可以使用已有的XML安全扩展标准，实现数字签名的功能，从而保证你的消息是来自特定方并没有被修改过。XML文件的加密技术从更大程度上加强了Webservice的安全，它能够定制数据传输到后，能否被接受者所查看，进一步完善了传输后的安全，业界也在不断的制定Webservice的安全标准，比如SAML 和 WS-Security。  
   
最后一层保护就是依靠底层架构的安全，这更多的来自于操作系统和某些中间件的保护。比如在J2EE中，主持Webservice的应用服务器。目前很多的J2EE应用服务器都支持Java Authentication and Authorization Service (JAAS)，这是最近被加入到J2SE 1.4当中的。利用主持Webservice的服务器，实现一些安全机制这是很自然的做法。另一种利用底层架构的安全方法就是，做一个独立的负责安全的服务器，Webservice的使用者和创建者都需要与之取得安全信任。

# [**设计与开发 JAX-WS 2.0 Web 服务**](http://www.cnblogs.com/chenying99/archive/2012/11/22/2782980.html)

## JAX-WS 简介

为何使用 JAX-WS？

JAX-WS 是用于简化使用 Java 构造 Web 服务和 Web 服务客户机的工作的技术。该技术提供了完整的 Web 服务堆栈，可减少开发和部署 Web 服务的任务。JAX-WS 支持 WS-I Basic Profile 1.1，后者可确保使用 JAX-WS 堆栈开发的 Web 服务能够供采用 WS-I Basic Profile 标准使用任意语言开发的任意客户机使用。JAX-WS 还包括了 Java Architecture for XML Binding (JAXB) 和 SOAP with Attachments API for Java (SAAJ)。

JAXB 提供了一种非常方便的方法来将 XML 模式映射到 Java 代码的表示形式，从而支持数据绑定功能。JAXB 消除了将 SOAP 消息中的 XML 模式消息转换为 Java 代码的工作，因而不必全面了解 XML 和 SOAP 解析。JAXB 规范定义 Java 和 XML 模式之间的绑定。SAAJ 提供了标准的方法来处理 SOAP 消息中包含的 XML 附件。

而 且，JAX-WS 提供了用于将传统 Java 对象（Plain Old Java Object，POJO）类转换为 Web 服务的 Annotation 库，从而加速了 Web 服务的开发工作。另外，它还指定了从采用 Web 服务描述语言（Web Services Description Language，WSDL）定义的服务到实现该服务的 Java 类之间的详细映射。采用 WSDL 定义的任意复杂类型都通过遵循 JAXB 规范定义的映射来映射为 Java 类。JAX-WS 之前与 Java Platform, Enterprise Edition (Java EE) 5 绑定。而 JAX-WS 2.0 规范是作为 Java Community Process (JCP) 的 JSR 224 开发的。

## 开发 Web 服务

契约优先方法与代码优先方法

进入 JAX-WS 时代的最好方法莫过于首先开发一个 Web 服务。可以采用以下两种方法之一开发 Web 服务：

* **契约优先：** 从 WSDL 契约着手，生成 Java 类来实现服务。
* **代码优先：** 从 Java 类着手，使用 Annotation 来生成 WSDL 文件和 Java 接口。

契约优先 WSDL 方法需要对用于定义消息格式的 WSDL 和 XML 模式定义（XML Schema Definition，XSD）有良好的理解。如果您对 Web 服务相当陌生，最好从代码优先方法着手，本教程中将使用此方法开发 Web 服务。

代码优先 Web 服务开发

使用代码优先方法时，将从实现希望作为服务公开的功能的 Java 类或类入手。在已经提供了 Java 实现且需要将实现作为服务公开的情况下，代码优先方法尤为有用。

## 开发订单处理 Web 服务

让我们首先创建一个订单处理 Web 服务，用于接受订单信息、配送信息和订购物品并最终生成确认 ID 作为响应。订单处理服务的代码如清单 1 中所示。这是一个虚拟实现，将在控制台输出客户 ID 和物品数量，然后输出虚拟订单 ID *A1234* 。

**清单 1. 订单处理 Web 服务实现**

|  |
| --- |
| package com.ibm.jaxws.tutorial.service;    import javax.jws.WebMethod;  import javax.jws.WebService;  import javax.jws.soap.SOAPBinding;  import com.ibm.jaxws.tutorial.service.bean.OrderBean;      //JWS annotation that specifies that the portType name of the   //Web service is "OrderProcessPort," the service name   //is "OrderProcess," and the targetNamespace used in the generated  //WSDL is "http://jawxs.ibm.tutorial/jaxws/orderprocess."    @WebService(serviceName = "OrderProcess",  portName = "OrderProcessPort",   targetNamespace = "http://jawxs.ibm.tutorial/jaxws/orderprocess")    //JWS annotation that specifies the mapping of the service onto the  // SOAP message protocol. In particular, it specifies that the SOAP messages   //are document literal.    @SOAPBinding(style=SOAPBinding.Style.DOCUMENT,use=SOAPBinding.Use.LITERAL,  parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)    public class OrderProcessService {    @WebMethod   public OrderBean processOrder(OrderBean orderBean) {    // Do processing...  System.out.println("processOrder called for customer"  + orderBean.getCustomer().getCustomerId());    // Items ordered are  if (orderBean.getOrderItems() != null) {  System.out.println("Number of items is "  + orderBean.getOrderItems().length);  }    //Process order.    //Set the order ID.  orderBean.setOrderId("A1234");    return orderBean;  }  } |

OrderBean 中包含订单信息，如清单 2 中所示。具体来说，其中包含对客户、订单项和配送地址对象的引用。

**清单 2. 包含订单信息的 OrderBean 类**

|  |
| --- |
| package com.ibm.jaxws.tutorial.service.bean;    public class OrderBean {    private Customer customer;    private Address shippingAddress;    private OrderItem[] orderItems;    private String orderId;    public Customer getCustomer() {  return customer;  }    public void setCustomer(Customer customer) {  this.customer = customer;  }    public String getOrderId() {  return orderId;  }    public void setOrderId(String orderId) {  this.orderId = orderId;  }    public Address getShippingAddress() {  return shippingAddress;  }    public void setShippingAddress(Address shippingAddress) {  this.shippingAddress = shippingAddress;  }    public OrderItem[] getOrderItems() {  return orderItems;  }    public void setOrderItems(OrderItem[] orderItems) {  this.orderItems = orderItems;  }    } |

开发 JAX-WS Web 服务的起点是一个使用 javax.jws.WebService Annotation 进行了标注的 Java 类。所使用的 JAX-WS Annotation 属于 Web Services Metadata for the Java Platform 规范 (JSR-181) 的一部分。您可能已经注意到了，OrderProcessService 使用 WebService Annotation 进行了标注，而后者将类定义为了 Web 服务端点。

OrderProcessService 类（带有 @javax.jws.WebService Annotation 的类）隐式地定义了服务端点接口（Service Endpoint Interface，SEI），用于声明客户机可以对服务调用的方法。除了使用 @WebMethodAnnotation 标注且 exclude 元素设置为 true 的方法外，类中定义的所有公共方法都会映射到 WSDL 操作。@WebMethod Annotation 是可选的，用于对 Web 服务操作进行自定义。除了 exclude 元素外，javax.jws.WebMethod Annotation 还提供 operation name 和 action 元素，用于在 WSDL 文档中自定义操作的 name 属性和 SOAP action 元素。这些属性是可选的；如果未定义，会从类名称派生缺省值。

实 现 Web 服务后，需要生成部署服务所需的所有构件，然后将 Web 服务打包为部署构件（通常为 WAR 文件），并将 WAR 文件部署到任何支持 JAX-WS 2.0 规范的兼容服务器上。通常生成的构件是提供基于服务接口将 Java 对象转换为 XML、WSDL 文件和 XSD 模式的功能的类。

出于测试目的，Java 6 绑定了一个轻量级 Web 服务器，可以通过调用简单的 API 调用将 Web 服务发布到该服务器上。接下来我们将了解如何使用此方法测试 Web 服务。

发布服务

生成 JAX-WS 构件

运行 wsgen 工具，以生成订单处理 Web 服务的 JAX-WS 可移植构件。此工具将读取 Web SEI 类，并生成 Web 服务部署和调用所需的所有构件。wsgen 工具生成需要发布的 Web 服务的 WSDL 文件和 XSD 模式。

为了生成 JAX-WS 构件，首先需要编译服务和 Bean 源文件：

1. 打开命令提示符，并进入到 c:/JAXWS-Tutorial目录。
2. 运行以下命令，以编译 Java 文件，并将类文件放入其各自文件夹中：   
     
   javac com/ibm/jaxws/tutorial/service/\*.java com/ibm/jaxws/tutorial/service/bean/\*.java
3. 运行以下命令，以生成 JAX-WS 构件：   
     
   wsgen -cp . com.ibm.jaxws.tutorial.service.OrderProcessService -wsdl

wsgen 工具提供了大量的选项，例如，其中提供了 -wsdl 选项，用于生成服务的 WSDL 和模式构件。运行此命令后，应该在 JAXWS-Tutorial 文件夹中看到生成的 OrderProcess.wsdl 和 OrderProcess\_schema1.xsd，而且会看到在 com/ibm/jaxws/tutorial/service/jaxws 文件夹中创建了 JAX-WS 构件。

生成了构件后，运行以下 Web 服务发布器客户机，以发布订单处理 Web 服务。

1. 从 c:/JAXWS-Tutorial 文件夹运行以下命令，以编译 OrderWebServicePublisher ：  
     
   javac com/ibm/jaxws/tutorial/service/publish/OrderWebServicePublisher.java
2. 然后运行以下命令：  
     
   java com.ibm.jaxws.tutorial.service.publish.OrderWebServicePublisher

运行 Java 程序后，应该看到以下消息： The Web service is published at http://localhost:8080/OrderProcessWeb/orderprocess. To stop running the Web service, terminate this Java process.

这会将订单 Web 服务发布到 http://localhost:8080/OrderProcessWeb/orderprocess。可以通过显示订单处理 Web 服务生成的 WSDL 来验证 Web 服务是否在运行：

1. 打开浏览器，并导航到 http://localhost:8080/OrderProcessWeb/orderprocess?wsdl。

|  |
| --- |
| https://www.ibm.com/i/v14/rules/blue_rule.gif https://www.ibm.com/i/c.gif |

|  |  |  |
| --- | --- | --- |
| https://www.ibm.com/i/c.gif   |  |  | | --- | --- | | https://www.ibm.com/i/v14/icons/u_bold.gif | [**回页首**](https://www6.software.ibm.com/developerworks/cn/education/webservices/ws-jax/section4.html#main) | |

分析 OrderWebServicePublisher

在分析 WSDL 和模式构件前，让我们分析一下 OrderWebServicePublisher 的代码。清单 3 提供了 OrderWebServicePublisher 客户机的源代码。

**清单 3. 用于发布订单处理 Web 服务的代码**

|  |
| --- |
| package com.ibm.jaxws.tutorial.service.publish;    import javax.xml.ws.Endpoint;    import com.ibm.jaxws.tutorial.service.OrderProcessService;    public class OrderWebServicePublisher {    public static void main(String[] args) {    Endpoint.publish("http://localhost:8080/OrderProcessWeb/orderprocess",  new OrderProcessService());    }    } |

通过 Endpoint.publish() 方法，可以方便地发布和测试 JAX-WS Web 服务。publish() 接受两个参数：Web 服务的位置和 JAX-WS Web 服务实现类。publish() 方法在指定的 URL（本例中为本地主机，端口为 8080）创建轻量级 Web 服务器，并将 Web 服务部署到该位置。此轻量级 Web 服务器在 Java 虚拟机（Java Virtual Machine，JVM）中运行，可通过调用 endpoint.stop() 方法以有条件的方式终止，或终止 OrderWebServicePublisher 客户机。

分析生成的 WSDL

要查看生成的订单处理 Web 服务 WSDL，在浏览器中键入以下 URL 位置： http://localhost:8080/OrderProcessWeb/orderprocess?wsdl .

让我们分析 WSDL 一些重要方面的内容，并了解如何基于 JAX-WS 元数据生成 WSDL 和模式构件，首先要分析的是生成的 XSD。此内容使用 xsd:import 标记导入到 WSDL 文件中（请参见清单 4）；schemaLocation 指定 XSD 的位置。

**清单 4. 包含订单处理模式定义的 WSDL 文件**

|  |
| --- |
| <types>    <xsd:schema>  <xsd:import namespace="http://jawxs.ibm.tutorial/jaxws/orderprocess"   schemaLocation="OrderProcess\_schema1.xsd"/>  </xsd:schema>  </types> |

在浏览器中打开 schemaLocation (http://localhost:8080/OrderProcessWeb/orderprocess?xsd=1)，以查看模式定义在浏览器中呈现的情况。让我们分析一下其中的情况：模式定义最开始是 targetNamspace 和tns 声明，映射到在 OrderProcessService 的 @WebService Annotation 中定义的 targetNamespace http://jawxs.ibm.tutorial/jaxws/orderprocess。清单 5 给出了对应的代码。

**清单 5. 模式（Schema）命名空间声明**

|  |
| --- |
| <xs:schema version="1.0"   targetNamespace="http://jawxs.ibm.tutorial/jaxws/orderprocess"  xmlns:tns="http://jawxs.ibm.tutorial/jaxws/orderprocess"   xmlns:xs="http://www.w3.org/2001/XMLSchema"> |

前面指定的 wsgen 工具命令生成两个包装 Bean 类，ProcessOrder 和 ProcessOrderResponse ，分别包含订单处理 Web 服务的输入和输出消息。将基于这些包装 Bean 类生成以下模式元素：

* processOrder 属于 processOrder 类型，表示其中包含一个元素，且此元素的名称为 arg0 ，类型为 orderBean 。可以看到，在 ProcessOrder 类和 processOrder 复杂类型之间存在一对一映射。
* processOrderResponse 与 processOrderResponse 类型类似，后者的定义映射到 ProcessOrderResponse 类。

让我们仔细分析一下清单 6 中的代码。

**清单 6. processOrder 的模式声明**

|  |
| --- |
| <xs:element name="processOrder" type="tns:processOrder"/>  <xs:element name="processOrderResponse" type="tns:processOrderResponse"/>  <xs:complexType name="processOrder">  <xs:sequence>  <xs:element name="arg0" type="tns:orderBean" minOccurs="0"/>  </xs:sequence>  </xs:complexType> |

清单 7 中所示的 orderBean 类型定义映射到 OrderBean 类。orderBean 类型定义包括：

* 一个 customer 元素，其类型为 customer 。
* 一个 orderId ，其类型为 string 。
* orderItems （它为数组类型，因为其将 maxOccurs 属性指定为 unbounded ），其类型为 orderItem 。
* shippingAddress ，其类型为 address 。

**清单 7. processOrder 的模式声明**

|  |
| --- |
| <xs:complexType name="orderBean">  <xs:sequence>  <xs:element name="customer" type="tns:customer" minOccurs="0" />   <xs:element name="orderId" type="xs:string" minOccurs="0" />   <xs:element nillable="true" maxOccurs="unbounded" name="orderItems"   type="tns:orderItem" minOccurs="0" />   <xs:element name="shippingAddress" type="tns:address"   minOccurs="0" />   </xs:sequence>  </xs:complexType |

类似地，模式的其余部分 customer 、orderItems 和 address 分别映射到 Customer 、OrderItem 和 Address Java Bean。

分析了模式定义后，接下来让我们回头来看看 WSDL 中的消息定义，如清单 8 中所示。WSDL 指定消息 processOrder 和 processOrderResponse ，其所属的元素为 processOrder 和 processOrderResponse （我们已经讨论了其模式定义）。portType 指定操作 processOrder ，其输入消息为 processOrder ，而输出消息为 processOrderResponse 。

**清单 8. WSDL 文档中的 processOrder 消息元素**

|  |
| --- |
| <message name="processOrder">  <part element="tns:processOrder" name="parameters" />   </message>  <message name="processOrderResponse">  <part element="tns:processOrderResponse" name="parameters" />   </message>  <portType name="OrderProcessService">  <operation name="processOrder">  <input message="tns:processOrder" />   <output message="tns:processOrderResponse" />   </operation>  </portType> |

接下来定义了 WSDL 绑定。此绑定将 soap:binding 样式定义为 document ，soap:body 使用 literal 标记指定操作 processOrder 的输入和输出消息格式。生成的 WSDL 定义映射到 @SOAPBindingAnnotation（已在 OrderProcessService 类上定义，请参见清单 9）。

**清单 9. WSDL 文档的绑定信息**

|  |
| --- |
| <binding name="OrderProcessPortBinding" type="tns:OrderProcessService">  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />   <operation name="processOrder">  <soap:operation soapAction="" />   <input>  <soap:body use="literal" />   </input>  <output>  <soap:body use="literal" />   </output>  </operation>  </binding> |

接下来定义 WSDL 服务。这将指定端口和对应的绑定类型，以及服务的实际位置。此位置通常为 HTTP 位置，在本例中为 http://localhost:8080/OrderProcessWeb/orderprocess。可以在清单 10 中了解到具体的情况。

**清单 10. WSDL 文档的服务信息**

|  |
| --- |
| <service name="OrderProcess">  <port name="OrderProcessPort" binding="tns:OrderProcessPortBinding">  <soap:address location="http://localhost:8080/OrderProcessWeb/orderprocess" />   </port> |

我们已经对生成的 WSDL 和模式构件进行了分析。清单 11 给出了一个示例 SOAP 请求消息，此消息是在 Web 服务客户机调用 processOrder 操作时发送的。

**清单 11. processOrder 操作的示例 SOAP 消息**

<?xml version="1.0"?>  
 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"   
 xmlns:ns1=" http://jawxs.ibm.tutorial/jaxws/orderprocess"   
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
 <soapenv:Body>  
 <ns1:processOrder>  
 <arg0>  
 <customer><customerId>A123</customerId>  
 <firstName>John</firstName><lastName>Smith</lastName></customer>  
 <orderItems><itemId>11</itemId><qty>11</qty></orderItems>  
 </arg0>  
 </ns1:processOrder>  
 </soapenv:Body>  
 </soapenv:Envelope>

创建 Web 服务客户机

从 WSDL 创建 Web 服务客户机

在本部分，我们将了解如何从 WSDL 创建 Web 服务客户机。JAX-WS 提供了名为 wsimport 的工具，用于从 WSDL 生成 JAX-WS 可移植构件。生成的可移植构件通常包括以下内容：

* SEI
* 服务（需要实现的服务实现类）
* 从模式类型生成的 JAXB 生成类
* 从 wsdl:fault 映射的异常类（如果有）

客 户机使用生成的构件调用 Web 服务。Web 服务客户机并不需要处理任何 SOAP 格式（如创建或解析 SOAP 消息）。这将由 JAX-WS 运行时予以处理，此运行时将使用生成的构件代码（JAXB 生成类）。Web 服务将处理 Java 代码（JAXB 生成类），从而减少了开发 Web 服务客户机和对 Web 服务调用操作的工作。

先使用 wsimport 工具从 OrderProcess WSDL 生成 JAX-WS 构件。然后要创建 Web 服务客户机，后者使用生成的构件代码调用订单处理 Web 服务。要生成 JAX-WS 构件，贤进入到 JAXWS-Tutorial 目录，并运行清单 12 中所示的 wsimport 命令。不过，进行操作前，请确保已经按照[生成 JAX-WS 构件部分](https://www6.software.ibm.com/developerworks/cn/education/webservices/ws-jax/section4.html#genjax) 中的步骤 5 所述的方法，通过运行 OrderWebServicePublisher 发布了 Web 服务。

**清单 12. 用于生成供 Web 服务客户机使用的 JAX-WS 构件的 wsimport 命令**

|  |
| --- |
| wsimport -keep -p com.ibm.jaxws.tutorial.service.client   http://localhost:8080/OrderProcessWeb/orderprocess?wsdl |

-keep 选项指示保留生成的文件，-p 选项指定需要在其中生成构件的包名称。http://localhost:8080/OrderProcessWeb/orderprocess?wsdl 指定 WSDL 文件的位置。以下构件是从 OrderProcessServiceWSDL 生成的：

* **JAXB 类（Address 、Customer , OrderBean 和 OrderItem ）：** 通过读取 OrderProcessService WSDL 中定义的模式定义生成
* **RequestWrapper 和 ResponseWrapper 类（ProcessOrder 和 ProcessOrderResponse ）：** 包装 document literal-wrapped 样式类型的输入和输出
* **服务类 (OrderProcess )：** 客户机用于请求 Web 服务的类
* **服务接口 (OrderProcessService )：** 包含着用于服务实现接口的类

接下来了解一下如何使用上面生成的构件创建 Web 服务客户机。com/ibm/jaxws/tutorial/service/client 文件夹中提供了一个示例参考代码。Web 服务客户机的代码如清单 13 中所示。

**清单 13. 订单处理 Web 服务客户机的代码清单**

|  |
| --- |
| package com.ibm.jaxws.tutorial.service.client;    import java.net.MalformedURLException;  import java.net.URL;    import javax.xml.namespace.QName;    public class OrderClient {    final QName qName = new QName(  "http://jawxs.ibm.tutorial/jaxws/orderprocess", "OrderProcess");    public static void main(String[] args) {  if (args.length != 1) {  System.out  .println("Specify the URL of the OrderProcess Web Service");  System.exit(-1);  }  URL url = getWSDLURL(args[0]);  OrderClient client = new OrderClient();  client.processOrder(url);  }    private static URL getWSDLURL(String urlStr) {  URL url = null;  try {  url = new URL(urlStr);  } catch (MalformedURLException e) {  e.printStackTrace();  throw new RuntimeException(e);  }  return url;  }    public void processOrder(URL url) {    OrderProcess orderProcessingService = new OrderProcess(url, qName);    System.out.println("Service is" + orderProcessingService);    OrderBean order = populateOrder();    OrderProcessService port = orderProcessingService.getOrderProcessPort();  OrderBean orderResponse = port.processOrder(order);    System.out.println("Order id is " + orderResponse.getOrderId());    }    private OrderBean populateOrder() {    OrderBean order = new OrderBean();  Customer customer = new Customer();  customer.setCustomerId("A123");  customer.setFirstName("John");  customer.setLastName("Smith");  order.setCustomer(customer);    // Populate Order Item.  OrderItem item = new OrderItem();  item.setItemId("11");  item.setQty(11);    order.getOrderItems().add(item);  return order;  }  } |
|  |

上面列出的 Web 服务客户机代码执行以下任务：

* 通过传入 OrderProcess Web 服务的 WSDL URL 和服务的 QName 创建 OrderProcess 类的实例。
* 创建 OrderBean 的实例，并使用 populateOrder() 方法填充订单信息。
* 对服务调用 getOrderProcessPort() ，以检索到服务的代理（也称为端口）。端口实现服务所定义的接口。
* 调用端口的 processOrder 方法，并同时传入在上面的第二个列表项目中创建的 OrderBean 实例。
* 从服务获得 OrderBean 响应并输出订单 ID。

运行 Web 服务客户机

要运行 Web 服务客户机，请首先从 JAXWS-Tutorial 文件夹运行以下命令来编译 Web 服务客户机：

javac com/ibm/jaxws/tutorial/service/client/OrderClient.java

通过使用以下命令提供订单处理 Web 服务的 WSDL URL 来运行 Web 服务客户机：

java com.ibm.jaxws.tutorial.service.client.OrderClient http://localhost:8080/OrderProcessWeb/orderprocess?wsdl

运行 Web 服务客户机时，会在控制台看到以下输出（OrderWebServicePublisher 在控制台中运行）：

processOrder called for customer A123   
Number of items is 1

在运行 Web 服务客户机的控制台中，会得到以下输出：

Order id is A1234

如上面的客户机代码中所示，并不会处理调用 Web 服务操作时使用的任何基于 SOAP 或 XML 的格式；相反，需要处理的是输入和输出消息的 JAXB 生成类，并使用服务接口和服务类对象（充当 Web 服务调用的存根）。存根负责从 JAXB Annotation 创建 SOAP 请求，并将 SOAP 响应转换回 Java 对象。

您现在已经成功地创建和发布了 Web 服务，并通过 Web 服务客户机成功地执行了此服务。

总结

在本教程中，我们了解了如何使用代码优先的开发方法和 JAX-WS 技术设计和开发 Web 服务。JAX-WS 是一个非常不错的选择，因为其中提供了完整的 Web 服务堆栈，以简化 Web 服务的开发和部署。

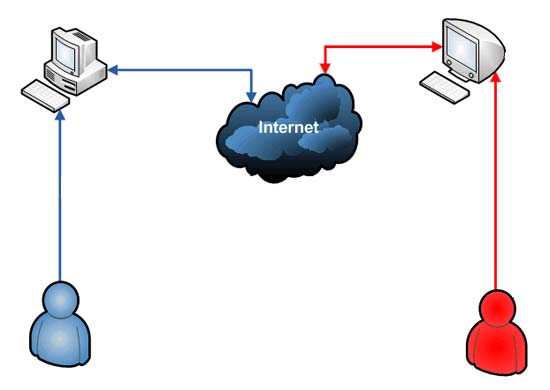
本 教程中开发的订单处理 Web 服务使用 Document 样式的 Web 服务，可确保服务使用者和服务提供者使用 XML 文档进行通信。XML 文档遵循定义良好的契约，而此类契约通常都是使用 XML 模式定义创建的。XML 模式格式指定服务使用者能够调用和遵循的业务消息的契约。Document 样式的 Web 服务应该是开发企业 Web 服务的首选方法。

# [**理解RESTful架构**](http://www.cnblogs.com/chenying99/archive/2012/03/30/2424507.html)

越来越多的人开始意识到，**网站即软件**，而且是一种新型的软件。

这种"互联网软件"采用客户端/服务器模式，建立在分布式体系上，通过互联网通信，具有高延时（high latency）、高并发等特点。

网站开发，完全可以采用软件开发的模式。但是传统上，软件和网络是两个不同的领域，很少有交集；软件开发主要针对单机环境，网络则主要研究系统之间的通信。互联网的兴起，使得这两个领域开始融合，**现在我们必须考虑，如何开发在互联网环境中使用的软件。**



RESTful架构，就是目前最流行的一种互联网软件架构。它结构清晰、符合标准、易于理解、扩展方便，所以正得到越来越多网站的采用。

但是，到底什么是RESTful架构，并不是一个容易说清楚的问题。下面，我就谈谈我理解的RESTful架构。

**一、起源**

REST这个词，是[Roy Thomas Fielding](http://en.wikipedia.org/wiki/Roy_Fielding)在他2000年的[博士论文](http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm)中提出的。



Fielding是一个非常重要的人，他是HTTP协议（1.0版和1.1版）的主要设计者、Apache服务器软件的作者之一、Apache基金会的第一任主席。所以，他的这篇论文一经发表，就引起了关注，并且立即对互联网开发产生了深远的影响。

他这样介绍论文的写作目的：

*"本文研究计算机科学两大前沿----软件和网络----的交叉点。长期以来，软件研究主要关注软件设计的分类、设计方法的演化，很少客观地评估不同的设计选择对系统行为的影响。而相反地，网络研究主要关注系统之间通信行为的细节、如何改进特定通信机制的表现，常常忽视了一个事实，那就是改变应用程序的互动风格比改变互动协议，对整体表现有更大的影响。****我这篇文章的写作目的，就是想在符合架构原理的前提下，理解和评估以网络为基础的应用软件的架构设计，得到一个功能强、性能好、适宜通信的架构。****"*

*(This dissertation explores a junction on the frontiers of two research disciplines in computer science: software and networking. Software research has long been concerned with the categorization of software designs and the development of design methodologies, but has rarely been able to objectively evaluate the impact of various design choices on system behavior. Networking research, in contrast, is focused on the details of generic communication behavior between systems and improving the performance of particular communication techniques, often ignoring the fact that changing the interaction style of an application can have more impact on performance than the communication protocols used for that interaction. My work is motivated by the desire to understand and evaluate the architectural design of network-based application software through principled use of architectural constraints, thereby obtaining the functional, performance, and social properties desired of an architecture. )*

**二、名称**

Fielding将他对互联网软件的架构原则，定名为REST，即Representational State Transfer的缩写。我对这个词组的翻译是"表现层状态转化"。

如果一个架构符合REST原则，就称它为RESTful架构。

**要理解RESTful架构，最好的方法就是去理解Representational State Transfer这个词组到底是什么意思，它的每一个词代表了什么涵义。**如果你把这个名称搞懂了，也就不难体会REST是一种什么样的设计。

**三、资源（Resources）**

REST的名称"表现层状态转化"中，省略了主语。"表现层"其实指的是"资源"（Resources）的"表现层"。

**所谓"资源"，就是网络上的一个实体，或者说是网络上的一个具体信息。**它可以是一段文本、一张图片、一首歌曲、一种服务，总之就是一个具体的实在。你可以用一个URI（统一资源定位符）指向它，每种资源对应一个特定的URI。要获取这个资源，访问它的URI就可以，因此URI就成了每一个资源的地址或独一无二的识别符。

所谓"上网"，就是与互联网上一系列的"资源"互动，调用它的URI。

**四、表现层（Representation）**

"资源"是一种信息实体，它可以有多种外在表现形式。**我们把"资源"具体呈现出来的形式，叫做它的"表现层"（Representation）。**

比如，文本可以用txt格式表现，也可以用HTML格式、XML格式、JSON格式表现，甚至可以采用二进制格式；图片可以用JPG格式表现，也可以用PNG格式表现。

URI只代表资源的实体，不代表它的形式。严格地说，有些网址最后的".html"后缀名是不必要的，因为这个后缀名表示格式，属于"表现层"范畴，而URI应该只代表"资源"的位置。它的具体表现形式，应该在HTTP请求的头信息中用Accept和Content-Type字段指定，这两个字段才是对"表现层"的描述。

**五、状态转化（State Transfer）**

访问一个网站，就代表了客户端和服务器的一个互动过程。在这个过程中，势必涉及到数据和状态的变化。

互联网通信协议HTTP协议，是一个无状态协议。这意味着，所有的状态都保存在服务器端。因此，**如果客户端想要操作服务器，必须通过某种手段，让服务器端发生"状态转化"（State Transfer）。而这种转化是建立在表现层之上的，所以就是"表现层状态转化"。**

客户端用到的手段，只能是HTTP协议。具体来说，就是HTTP协议里面，四个表示操作方式的动词：GET、POST、PUT、DELETE。它们分别对应四种基本操作：**GET用来获取资源，POST用来新建资源（也可以用于更新资源），PUT用来更新资源，DELETE用来删除资源。**

**六、综述**

综合上面的解释，我们总结一下什么是RESTful架构：

　　（1）每一个URI代表一种资源；

　　（2）客户端和服务器之间，传递这种资源的某种表现层；

　　（3）客户端通过四个HTTP动词，对服务器端资源进行操作，实现"表现层状态转化"。

**七、误区**

RESTful架构有一些典型的设计误区。

**最常见的一种设计错误，就是URI包含动词。**因为"资源"表示一种实体，所以应该是名词，URI不应该有动词，动词应该放在HTTP协议中。

举例来说，某个URI是/posts/show/1，其中show是动词，这个URI就设计错了，正确的写法应该是/posts/1，然后用GET方法表示show。

如果某些动作是HTTP动词表示不了的，你就应该把动作做成一种资源。比如网上汇款，从账户1向账户2汇款500元，错误的URI是：

*POST /accounts/1/transfer/500/to/2*

正确的写法是把动词transfer改成名词transaction，资源不能是动词，但是可以是一种服务：

*POST /transaction HTTP/1.1  
　　Host: 127.0.0.1  
　　  
　　from=1&to=2&amount=500.00*

**另一个设计误区，就是在URI中加入版本号**：

*http://www.example.com/app/1.0/foo*

*http://www.example.com/app/1.1/foo*

*http://www.example.com/app/2.0/foo*

因为不同的版本，可以理解成同一种资源的不同表现形式，所以应该采用同一个URI。版本号可以在HTTP请求头信息的Accept字段中进行区分（参见[Versioning REST Services](http://www.informit.com/articles/article.aspx?p=1566460)）：

*Accept: vnd.example-com.foo+json; version=1.0*

*Accept: vnd.example-com.foo+json; version=1.1*

*Accept: vnd.example-com.foo+json; version=2.0*

[**构建REST风格的Web Service**](http://www.cnblogs.com/chenying99/archive/2012/06/12/2546089.html)

1．什么是REST?

REST 是由 Roy Fielding 在他的论文[《Architectural Styles and the Design of Network-based Software Architectures》](http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm)中提出的一个术语。

REST 是英文 Representational State Transfer 的缩写，有中文翻译为“具象状态传输”（参考：[《SIP/IMS网络中的Representational State Transfer (REST)和数据分布》](http://dev2dev.bea.com.cn/bbsdoc/20060529259.html)）。

可以将REST归纳如下：

1.1首先REST只是一种风格，不是一种标准

|  |
| --- |
| You will not see the W3C putting out a REST specification. You will not see IBM or Microsoft or Sun selling a REST developer's toolkit. Why? Because REST is just an architectural style. You can't bottle up that style. You can only understand it, and design your Web services in that style.  *-****Roger L. Costello***[*《Building Web Services the REST Way》*](http://www.xfront.com/REST-Web-Services.html)*。* |

1.2  REST是以资源为中心的

在REST中，认为Web是由一系列的抽象资源（Abstract Resource）组成，这些抽象的资源具有不同的具体表现形式((Representational State)。譬如，定义一个资源为photo，含义是照片，它的表现形式可以是一个图片，也可以是一个.xml的文件，其中包含一些描述该照片的元素。或是一个html文件。并且这些具体的表现可以分布在不同的物理位置上。

1.3  REST的目的是决定如何使一个定义良好的 Web程序向前推进

一个程序可以通过选择一个Web页面上的超链(这些链接代表资源)，使得另一个Web页面 (Representational State of an abstract resource) 返回(transfer)到用户，使程序进一步运行。

1.4  REST充分利用或者说极端依赖HTTP协议

**它通过逻辑URI定位资源**

在 REST 的定义中，一个 Web 应用总是使用固定的 URI 向外部世界呈现（或者说暴露）一个资源。

这里必须注意

1，一个URI对应一个资源。

2，这里的URI是一个逻辑URI，

一个逻辑 URI： [url]http://www.example.com/photo/logo[/url]

一个物理 URI： [url]http://www.example.com/photo/logo.html[/url]

**通过分辨 HTTP Request Header 信息来分辨客户端是想要取得资源的哪一种表现形式的数据。**

我们来看一个实际例子：

[url]http://www.example.com/photo/logo[/url] 指向 example.com 网站（可以视为一个 Web 应用）中类型为 photo，名字为 logo 的资源。我们用浏览器访问这个 URI，看到的将可能是一个 html 文档，其中用 <img src=”……” /> 来显示实际的照片。

[url]http://www.example.com/photo/logo[/url]这个地址很可能会在服务器内部处理为 [url]http://www.example.com/photo.jsp?name=logo[/url] 这样的地址。photo.jsp 是服务器端的一个动态脚本文件，根据 name 参数生成 html 文档返回给浏览器。

现在假设我们要获取这张照片的 XML 文档。XML 文档中包含照片的文件名、文件大小、拍摄日期等等信息。也就是说我们要获取“**同一个资源的不同表现形式的数据**”。对于这个要求，我们可以很容易的用另一个 URL地址达到：[url]http://www.example.com/xml/logo[/url]。

但是，这就违背了“**URI 唯一标识一个资源**”的定义。如果我们要获取同一个资源的多种表现形式，那么就要使用更多的 URL，从而给一个资源指定了多个不同的 URI。

而在REST 中，不管是获取照片的 xhtml 文档还是 XML 文档，或者照片文件本身，都是用同一个 URI，就是 [url]http://www.example.com/photo/logo[/url]。

那这是怎么办到的呢？是通过分辨 HTTP Request Header 信息来分辨客户端是想要取得资源的哪一种表现形式的数据。

当我们用浏览器访问一个网址时，浏览器会构造一个 HTTP 请求。这个请求有一个头信息，其中包括了本次请求接受何种类型的数据。通常浏览器发送的 HTTP 请求头中，Accept 的值都是 \*/\*，也就说接受服务器返回的任何类型的数据。

只要我们指定一个特定的 Accept 参数，那么服务器就可以通过判断该参数来决定返回什么类型的数据。所以在一个采用 REST 架构的应用中，要获取同一个资源的不同表现形式的数据，只需要使用不同的 HTTP 请求头信息就行了。

如果考虑为 Web 应用增加 Web Services，这种技术的价值就体现出来了。比如写了一个程序，现在只需要构造一个包含 Accept: text/xml 的 HTTP 请求头，然后将请求发送到[url]http://www.example.com/photo/logo[/url] 就可以了。返回的结果就是一个 XML 文档，而不是别的资源形式的数据。

**在 REST 架构中，用不同的 HTTP 请求方法来处理对资源的 CRUD（创建、读取、更新和删除）操作**

我们在 Web 应用中处理来自客户端的请求时，通常只考虑 GET 和 POST 这两种 HTTP 请求方法。实际上，HTTP 还有 HEAD、PUT、DELETE 等请求方法。而在 REST 架构中，用不同的 HTTP 请求方法来处理对资源的 CRUD（创建、读取、更新和删除）操作：

POST: 创建

GET: 读取

PUT: 更新

DELETE: 删除

经过这样的一番扩展，我们对一个资源的 CRUD 操作就可以通过同一个 URI 完成了：

[url]http://www.example.com/photo/logo[/url]（读取）  
仍然保持为 [GET] [url]http://www.example.com/photo/logo[/url]

[url]http://www.example.com/photo/logo/create[/url]（创建）  
改为 [POST] [url]http://www.example.com/photo/logo[/url]

[url]http://www.example.com/photo/logo/update[/url]（更新）  
改为 [PUT] [url]http://www.example.com/photo/logo[/url]

[url]http://www.example.com/photo/logo/delete[/url]（删除）  
改为 [DELETE] [url]http://www.example.com/photo/logo[/url]

从而进一步规范了资源标识的使用。

通过 REST 架构，Web 应用程序可以用一致的接口（URI）暴露资源给外部世界，并对资源提供语义一致的操作服务。这对于以资源为中心的 Web 应用来说非常重要。

2．RESTful Web Services 特点

通过将HTTP Accept类型设置为text/xml，我们便可以将资源返回的“具象状态”表示为可跨平台识别的数据。这是应用于Web Services的基础。

RESTful Web Service具有以下特点：

2.1无状态的

每一个来自客户端的request必须包含所有必要的信息，即不能从服务器端获得任何保存的上下文信息。

REST 的 “客户机－无状态－服务器” 约束禁止在服务器上保存会话状态。符合这一约束进行设计

1，可以提高系统的可靠性和可伸缩性。它不需要昂贵的维护和支持费用，因为状态并不在服务器上维护。

2，可以进行资源缓存。Web的高速缓存既可以驻留在客户主机中，也可以驻留在中间网络高速缓存服务器主机中。

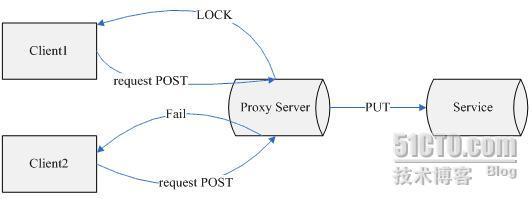
2.2  HTTP头中可见的统一接口和资源地址

通过对于HTTP Head 的解析，我们便可以了解到当前所请求的资源和请求的方式。相对于SOAP RPC风格来说，我们必须解析HTTP体。

这样做对于一些代理服务器的设置，将带来很高的处理效率。

REST 系统中所有的动作和要访问的资源都可以从H TTP和URI 中得到，这使得代理服务器、缓存服务器和网关很好地协调工作。而RPC模 型的SOAP 要访问的资源仅从 URI无法得知，要调用的方法也无法从 HTTP中得知，它们都隐藏在 SOAP 消息中。

同样的，在REST系统中的代理服务器还可以通过 HTTP 的动作 (GET 、 POST)来进行控制。



2.3  返回一般的XML格式内容

一般情况下，一个RESTful Web Service将比一个SOAP RPC Web Service占用更少的传输带宽。

|  |
| --- |
| POST/Order HTTP/1.1  Host:[url]www.northwindtraders.com[/url]  Content-Type:text/xml  Content-Length:nnnn    <UpdatePO>        <orderID>098</orderID>        <customerNumber>999</customerNumber>        <item>89</item>        <quantity>3000</quantity>  </UpdatePO> |

|  |
| --- |
| POST/Order HTTP/1.1  Host:[url]www.northwindtraders.com[/url]  Content-Type:text/xml  Content-Length:nnnn  SOAPAction:”urn:northwindtraders.com:PO#UpdatePO”    <SOAP-ENV:Envelope  xmlns:xsi=”[url]http://www.3w.org/1999/XMLSchema/instance[/url]”  xmlns:SOAP-ENV=”[url]http://schemas.xmlsoap.org/soap/envelope[/url]”  xsi:schemaLocation=”[url]http://www.northwindtraders.com/schema/NPOSchema.xsd[/url]”>  <SOAP-ENV:Body xsi:type=”NorthwindBody”>      <UpdatePO>        <orderID>098</orderID>        <customerNumber>999</customerNumber>        <item>89</item>        <quantity>3000</quantity>      </UpdatePO>  </SOAP-ENV:Body>  </SOAP-ENV:Envelope> |

2.4  安全机制

REST使用了简单有效的安全模型。REST中很容易隐藏某个资源，只需不发布它的U RI；而在资源上也很容易使用一些安全策略，比如可以在每个 URI 针对 4个通用接口设置权限；再者，以资源为中心的Web服务是防火墙友好的，因为 GET的 意思就是GET， P UT 的意思就是PUT，管理员可以通过堵塞非GET请求把资源设置为只读的，而现在的基于RPC 模型的 SOAP 一律工作在 HTTP 的 POST上。

使用 SOAP RPC模型，要访问的对象名称藏在方法的参数中，因此需要创建新的安全模型。

2.5  无法用于事务型的服务

对于事务型的服务，一个简单的例子就是银行事务，在那里用户可以把钱从一个账户转移到另一个账户上。用户不想直接操作资源（钱、银行账户等等），他们只想告诉银行他们想要达到的目的，并且让银行根据他们的利益对资源进行处理。

所以从这一条，我们应该明白，选择基于REST或SOAP RPC风格的Web 服务，我们应该首先考虑这个服务是针对资源的还是针对活动的。

*- James Snell，面向资源与面向活动的 Web 服务，*[*英文原文*](http://www.ibm.com/developerworks/webservices/library/ws-restvsoap/?S_TACT=105AGX52&S_CMP=cn-a-ws)*。*

3．JAX-WS

是Java Architecture for XML Web Services的缩写,简单说就是一种用Java和XML开发Web Services应用程序的框架, 目前版本是2.0, 它是JAX-RPC 1.1的后续版本。

[JAX-WS2.0](https://jax-ws.dev.java.net/)用JAXB2来处理Java Object与XML之间的映射。

JAX-WS通过javax.xml.ws.Provider接口来构建REST风格的终端。

4．WADL

[Web Application Description Language (WADL)](http://research.sun.com/spotlight/2006/2006-04-24-TR153.html)是由SUN公司提出的，旨在提供一种Web 应用的描述语言。WADL主要描述一个Web 应用的

* 资源列表-站点所有的资源
* 资源之间的关系-说明资源之间的链接关系
* 所有应用于每个资源的特定方法-应用于每个资源的HTTP方法，指定的输入和输出以及它们支持的数据格式
* 资源表现的形式-支持的MIME类型和所用到的XML schemas



下面是一个WADL示例，它描述了Yahoo新闻搜索的应用。

* Lines 2 开始一个应用描述，定义所使用的namespaces。
* Lines 3–6 定义XML 语法，这个示例中是两个W3C XML Schema。
* Lines 7–11描述了Yahoo新闻搜索Web 资源以及它所支持的HTTP方法。
* Lines 12–26 描述了‘search’ GET方法。
* Lines 13–21 描述输入。
* Lines 22–25 描述所有可能的输出。

5．关于WADL的项目介绍

[***GlassFish***](https://glassfish.dev.java.net/)***»***[***Web Application Description Language***](https://wadl.dev.java.net/)

***wadl2java***

***A tool that generates client side stubs from WADL files. May be used from the command line or as an Apache Ant plug-in, see the***[***wadl2java documentation***](https://wadl.dev.java.net/wadl2java.html)***for full details.***

***wadl2java\_yahoo***

***A sample project that uses the wadl2java tool to create stubs for the***[***Yahoo News Search Service***](http://developer.yahoo.com/search/news/V1/newsSearch.html)***. Includes a simple main method that uses the generated stubs to query for the latest Java news.***

6．建议阅读的资料

[1]Roy Thomas Fielding，[Architectural Styles and the Design of Network-based Software Architectures](http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm)

[2][RESTwiki](http://rest.blueoxen.net/cgi-bin/wiki.pl)

[3] [Rest Architectural Style](http://c2.com/cgi/fullSearch)

[4] [REST mailing list](http://groups.yahoo.com/group/rest-discuss/)

[5] [Building Web Services the REST Way](http://www.xfront.com/REST-Web-Services.html)

[6] [面向资源与面向活动的 Web 服务](http://www-128.ibm.com/developerworks/cn/webservices/ws-restvsoap/)

[7] [Some thoughts about SOAP versus REST on Security](http://www.prescod.net/rest/security.html)

[8]WADL语言作者[Marc Hadley's Blog](http://weblogs.java.net/blog/mhadley/)

[9][Java Web Service Technologies @Sun](http://java.sun.com/webservices/technologies/index.jsp)