用于创建REST风格系统的框架

文/Eric J. Bruno 译/靳黎明

表述性状态转移，简称“REST”，是比Web服务限制更少的SOA形式。

面向服务架构（SOA）和开发是一个示例，采用SOA软件，组件通过简明的接口创建，而且每一个组件执行一系列包含相关功能的独立集合。通过定义良好的接口和使用合同，每一个组件都为软件组件提供服务。这很类似于为商业服务的会计师，他们所提供的服务包括很多相关功能——记账、报税、投资管理等等。

SOA没有任何技术需求和限制。你可以通过使用任何一种标准的语言（如CORBA、远程过程调用（RCP）、或者XML）来构建一个服务。

一个Web服务就是SOA的实例，而且是实现选择的定义良好的集合。大体上说，技术选择是简单对象访问协议（SOAP）和Web服务定义语言（WSDL）。它们都是基于XML的。WSDL用于描述接口（“契约”），而SOAP则是描述所传输的数据。由于XML、SOAP和WSDL的平台无关特性，Java所具备操作系统无关性，因此Java成为web服务实现的一个很流行的选择。

由于Web服务系统很标准而且没有很多的平台限制，因此，它们是客户端/服务器系统以及专有对象模型如COBRA或者COM的改进版本。除此之外，典型的用于实现Web服务的标准、语言以及协议都在帮助所构建的系统来进行更好的衡量。

表述性状态转移(REST)

然而，现实中已经存在着甚至比Web服务限制更少的一种SOA形式——表述性状态转移（REST）。Roy Fielding在他的博士学位论文中描述到，REST是一系列独立技术特征的集合，除了需要基于HTTP的需求。

与下列一系列特征相符合的系统就可以称之为“REST风格”：

- 系统中所有的组件通过接口进行通讯，这些接口的方法定义和动态代码非常清晰。
- 每一个组件都是唯一的，它们通过超媒体链接（URL）定义。
- 遵循客户端/服务器架构（Web浏览器和Web服务器）。
- 所有的通讯都是无状态的。
- 架构是分层的，数据可以在任何层进行高速缓存。

这些特征直接与Web开发中所使用的技术相关联，而且根据Fielding在他文章中所描述的，这些特征正是大多数Web开发成功的原因。HTTP协议，它的方法接口（(GET、POST、HEAD等等），URL、HTML以及JavaScript的使用，也包括什么是web服务器和什么是web浏览器之间的明确区别，所有的这些技术都是直接与前四条原则相关的。在大多数网站实现中：负载平衡、内存高速缓存、防火墙、路由器等等，也都是满足前四条原则（包括层）的。这些设备早已被人们所接受，因为它们并不影响组件之间的接口；它们完全用于提高整体性能和通讯效率。

REST与其他的软件架构有着很大的区别，REST把软件架构（接口、组件、连接器以及模式等等）的概念与网络架构（可携带性、带宽管理、吞吐量测量以及协议延时等等）的概念相互结合起来。这种结合使得REST成为可测量的分布式系统的理想状态，就处理能力和通讯效率而言这一点是非常关键的。

图1展示了REST的架构，它是逻辑软件架构和物理网络元素的相互结合。如图中所示，通讯是在HTTP上执行的，客户端包含可选服务器高速缓存以满足整体的高效性，服务为后端数据库中部署高速缓存，没有对于最大客户端数量的任何限制，也没有对每个客户端可用拥有最多服务数的限制，服务可以调用服务，负载平衡硬件用于可测量性，防火墙用于安全。

在数据高速缓存方面，仍然有一些很值得注意的关键点。首先，需要标记数据，无论是采用隐型的方式还是明确的标记，无论是作为高速缓存或者不采用高速缓存的方式，都要进行数据标记。第二，虽然可能使用特殊化的高速缓存，如web浏览器（custom、内存数据结构），但是，通用高速缓存、如web浏览器高速缓存或者是第三方web高速缓存（如Akamai），也都是可以被使用的。

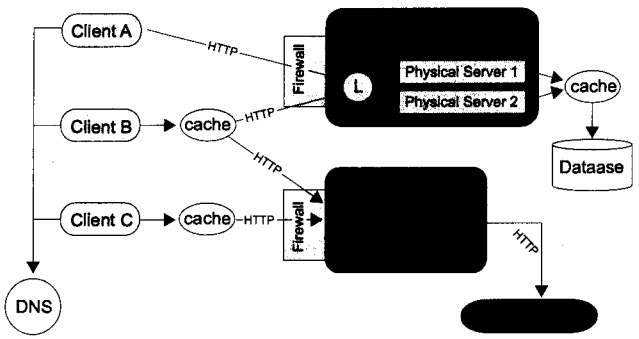

图1 该架构图提供一个REST特征的可视化总览

构建REST风格的系统

如果你排除典型的web服务协议((XML-RPC SOAP和WSDL等等)，那么，你将如何构建一个基于SOA的REST风格的系统呢？通过使用REST，你可以使用与用于请求一个web页面——HTTP查询URL的相同的机制。举个例子，在下面的示例1中，范例SOAP调用，就是从人力资源web服务中请求雇员的利益信息。

通过使用REST，你可以通过很简单的方法来替代SOAP调用，正如上面的示例1中所示，使用URL链接http://humanresources.com/benefits?user=<USER_SSID>&type=full_time_employee。

只要你知道了HTTP查询的URL链接定义，那么，你就可以通过该链接调用REST风格的服务。最终得响应可以是HTML、逗点分格文件数据、XML或者更为成熟的文件类型（如电子制表）。但是，有些人抱怨，在响应的所有这些形式中，除了基于超媒体内容的形式之外，其他的都不是真正的REST风格。然而，只要系统对于请求和通讯协议是满足REST风格的，那么，响应类型就不是那么重要了。

当你通过使用一个Java Servlet来构建web应用程序时，举个例子，它直接通过URL查询参数来读出数据，然后将某种基于文本的响应返回到调用端（请参考代码1）。

REST服务框架

虽然，与REST服务相比，web服务正变得越来越复杂，但是，web服务拥有一些web服务工具和库能够帮助程序开发人员避免重复的编码和测试。可是对于REST而言，这样的工具和库并不是很多，起码对于Java程序开发人员来说是这样的。为了弥补这一点，我已经构建了REST服务框架（电子版可用，请参见“资源中心”第五页），如此一来，我不再为每一个我需要的服务进行重复编码。这个基于Java的框架，使得我能够集中于我所要实现的接口和功能的编码，而不是在最起码的实现Java Servlet的代码及对于数据的映射URL查询参数上浪费时间。

在REST框架中有两个非常重要的组件：

- REST服务器，是把HTTP URL查询映射到应用程序代码的Java Servlet；
- REST工作器，是当请求到来时被框架动态调用的Java类，然后生成响应；

REST工作对象是你开发的用于提供服务功能的对象。每一个工作器都必须执行RestWorker接口，如示例2所示。

RestWorker接口定义如下方法：

- 当用于工作器对象的HTTP请求到达时调用onRequest，这是由对象的类名称所决定的
- REST服务器调用cacheReference来决定该工作器对象的关联是否应该被高速缓存。如果是被高速缓存的，那么，对象关联被存储并用于所有的用于该对象URL关键值的HTTP请求的子序列；另外，对于每个请求都会实例化一个新的对象。

REST服务器定义一个通用的HTTP URL查询参数request，它用于确定哪一个REST工作器对象应该被调用。该参数的值与被调用的类名相匹配。举个例子，当请求到达时：

```
http://<rest_server_name>   /res
tserver?request=EmpBenefits
```

REST服务器尝试加载一个名为EmpBenefits的类，它实现RestWorker接口，调用onRequest方法。如果有的

```
<SOAP-ENV:Envelope xmlns:SOAP
 ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header>
        some data here...
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
    <GetBenefits>
        <user>123-45-6789</user>
        <type>full_time_employee</type>
    </GetBenefits>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

例1：范例SOAP调用为雇员请求利益信息

```
protected void doPost( HttpServletRequest req, HttpServletResponse resp)
  throws ServletException, IOException{
    ServletOutputStream out = resp.getOutputStream();
    String response;

    String userSSID = req.getParameter("user");
    String userType = req.getParameter("type");
    if ( userType.equals("full_time_employee")) {
        Employee emp = lookupUser(userSSID);
        String medPlan = emp.getMedicalPlan();
        String dntPlan = emp.getDentalPlan();
        String retPlan = emp.getRetirementPlan();
        Response = "User " + emp.getFullName() +
                   " has medical plan: " + medPlan +
                   ", and dental plan: " + dntPlan +
                   ", and retirement plan: " + retPlan;
    }
    else {              // ...
    }
    out.println(response);// Output the response from the worker

}
```

代码1

```
public interface RestWorker {
    public String onRequest(Map paramMap);
    public boolean cacheReference();
}
```

例2：RestWorker接口定义

话，所有其他的URL查询参数，都作为named-value pairs java.util.Map通过。

代码2展示了用于Rest服务器Java Servlet的doPost方法，doPost方式实现该功能。当追踪了request参数之后，整个URL查询参数集都通过对HTTPServletRequest.getParameterMap的调用进行追踪。请求参数从映射中清楚，因为这些参数仅仅对于REST服务器有用，对工作器对象是没有任何意义的。另外，调用传给getWorker类，第一次尝试定位于高速缓存中的该请求工作器对象的关联。如果没有发现提前高速缓存关联的话，那么调用将被发送给creatWorker（请参考代码3）。

方法creatWorker使用Class.forName库调用来为制定的类名Java.lang.Class对象加载一个关联，使用Java映像。

```
protected void doPost( HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException{
    ServletOutputStream out = resp.getOutputStream();
    String response;
    // Determine which worker this request is for
    String urlKey = req.getParameter("request");
    if ( urlKey == null ) {
        out.println("REST Server Ready");
        return;
    }
    // Get the URL query parameters (remove param "request")
    Map paramMap = req.getParameterMap();
    HashMap params = new HashMap(paramMap);
    params.remove("request");
    // Lookup the correct worker for this request and call it
    RestWorker worker = getWorker(urlKey);
    if ( worker != null )
        response = worker.onRequest(params);
    else
        response = "No REST worker for " + urlKey;
    // Output the response from the worker
    out.println(response);
}
```

```
public RestWorker createWorker(String className){
    try {
        Class compClass = Class.forName( className );
        if ( compClass != null ){
            Object obj = compClass.newInstance();
            return (RestWorker)obj;
        }
    }
    catch ( Exception e ){
        log(e.getMessage(), e);
    }
    return null;
}
```

```
public String onRequest(Map params)
{
    String resp = "Thank you for calling EchoWorker. ";
    if ( params.size() == 0 )
        return resp;
    resp += "\n\nHere are the parameters you passed: ";
    Set keys = params.keySet();
    Iterator keyIter = keys.iterator();
    while ( keyIter.hasNext() )
    {
        String key = (String)keyIter.next();
        String[] val = (String[])params.get(key);
        resp += "\n " + key + "=" + val[0];
    }
    return resp;
}
```

通过标准Java Servlet容器，如Apache Tomcat、只有在Web应用程序WEB-INF或者类路径种的类才能够在缺省情况下被定位。因此，对于worker的算法，你必须把你的RestWorker对象放在该目录下。一旦类被定位，Java.lang.Class对象就会被加载，实际的RestWorker对象就会通过调用Class.newinstance来进行实例化。

下一步，调用工作对象的cache方法。如果返回值为真，那么，该对象的关联将存储在内存高速缓存中来使得子序列请求能够执行更快一些（描述的Java映像代码并不是一直需要被调用的），如果调用该方法的返回值为假，那么对象关联将仅仅用于该调用。这使得你能够运行的服务器上通过将类对象简单复制WEB-INF或者类路径中来动态更新你的工作器对象。子序列请求需要使用这一新类。

最终，通过来自原始HTTP请求的Map参数调用工作对象的onRequest方法。这一方法的返回值被用作响应，直接返回到web客户端。因此，你的对象可以返回HTML、逗点分格文件数据、XML或者更为可读的文本。正如从web浏览器中所观察到的简单“echo”工作对象的输出（参见代码4）。对RestWorker对象的HTTP请求以分段的形式返回，该请求中包括所有的请求参数和值。

结论

我已经在不同产品系统中成功的构建了很多的REST风格的服务，也构建了很多的Web服务。根据我的经验，构建、配置和使用REST服务比使用发展成熟的web服务要更快、更简单。如果你是因为担心Web服务开发的复杂性还没有跳跃到基于SOA开发这一领域的话，那么，不要担心，请尝试一下REST和这个框架。■

本文原文：http://www.ddj.com/web-development/199902676，略有删节。