使用 Web Service 实现系统的重用

柳 杨　杨贯中　（长沙 湖南大学软件学院 410082）

摘要：如何利用 Web Service 更好的实现系统的重用，文中给出了引入 Web Service 后系统的体系结构，并讨论了可行性与实现方法。

关键词：Web Service UDDI WDSL SOAP 松散耦合

1 采用 Web Service 实现系统重用的可行性

随着企业信息化的发展，应用系统的增多，由于应用系统在设计时并没有将它作为整个企业信息系统的一部分，造成了各个应用系统之间协同工作能力非常有限。另外，随着 Internet 的发展，一些业务流程将会跨越多个内部应用系统，甚至跨越不同的企业，很多资源需要共享。因此，在应用系统设计时采用一种共同的方式实现系统的重用、资源的共享是非常必要的。

对于系统的重用，有三个层次：

首先，是数据集的重用。但是由于数据库的复杂性和数量多，要在数据库之间移动数据，必须要明白哪个数据库中存放的是什么数据、采用的是什么格式，而且还必须知道何时以及怎样提取数据，更重要的是要熟悉目标数据库的类型和结构，也就是数据库的语义问题。

其次，是应用接口集的重用。这是由于数据集的重用存在着很大的弊端，所以产生了这个层次的系统重用，其关注于功能和数据的共享，而不仅是单纯的数据。应用接口集的重用通过使用 API 来实现。

最后，是业务方法集的重用。对于应用接口集的重用来说，使用 API 也产生了一些弊端，如异构系统不能调用 API。业务方法集的重用就是为了解决应用接口集的弊端而产生的，它通过注重于共享业务逻辑来使系统得到重用。通过开发虚拟组件，提供了基于业务需求的接口，只要接口保持不变，对系统的调用也不会发生改变，这样就使系统达到最大限度的重用。

传统业务方法集的系统重用一般是使用 CORBA、DCOM、RMI/IIOP 等分布式技术，点对点重用，实施代价昂贵，持续时间长。而且这些技术之间的相互操作性（与不同厂家和平台的软件之间共享数据和相互通信的能力）非常有限，客户端与服务端必须紧密耦合，一旦服务端的接口或执行方式发生变化，客户端将无法执行。这些问题影响了系统重用的能力。

与传统的方式比较，Web Service 的设计和应用更加简单。首先，由于 Web Service 的应用是松散耦合的，Web Service 的提供方和使用方相互独立；其次，Web Services 是完全的平台、语言无关，只要遵守 Web Services 的接口即可进行服务的请求与调用，而无须考虑系统的编程语言及部署的硬件平台；而且，由于所采用的协议都是通用的（HTTP、XML），Web Service 不只是能实现企业内部的重用，更重要的是实现跨企业的系统重用，并且能够被广泛支持与向前兼容，潜在消除了企业日后为提供新业务而需要进行二次投资的风险；再者，Web Service 是面向服务的，其服务接口可以动态改变，而服务使用者可以通过到 UDDI 注册中心自动重新查找、绑定服务，这样可以在不断扩大、变动的用户需求以及商业应用中实现最大程度的系统重用。

2 基于 Web Service 的系统体系架构

图 1 是 Web Service 体系架构的示意图。从示意图上可看出，在 Web Service 体系架构中有三种角色，它们分别是：UDDI 注册中心（也可以称为服务代理者）、服务使用者以及服务提供者。这三个角色之间的相互操作为：首先，服务提供者向 UDDI 注册中心发布他们所提供的服务，这个操作是通过 SOAP 消息进行传递的，发布的内容是使用 WSDL 进行描述的服务接口；然后，服务使用者也通过 SOAP 消息到 UDDI 注册中心查找服务提供者提供的服务，而 UDDI 注册中心将服务提供者提供的 WSDL 通过 SOAP 消息传递给服务使用者；接着，服务使用者按照接收到的 WSDL 中定义

的接口调用方式通过 SOAP 消息将服务绑定到服务提供者。

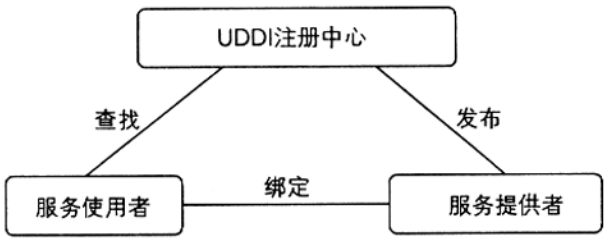

图 1

3 将应用系统功能封装为 Web Service

假设目前已存在一个学习对象管理系统，该系统是为了使学习对象得到更好的管理和重用而使用 J2EE 构架来设计和实现的，其中对学习对象采用 IMS 的基于 XML 的 LOM(Learning Object Metadata，学习对象元数据)来描述。在学习对象管理系统中有一个将学习对象保存到数据库中(即资源入库)的功能，可以按照如下方法将此功能封装为 Web Service，以便可以使用异构的.net 客户端来调用。

首先，在 JBuilder 中建立一个项目，将资源入库功能所需要调用的类包导入，新建一个类 BaseService，在类 BaseService 中建立一个方法 putNewResource(String name, String tokeID, String bytes, int mark)实现资源入库功能，然后通过使用 JBuilder 的功能直接由类 BaseService 生成相应的 Web Service，编译后自动生成相应的基于 XML 的 WSDL 文件和调用此 Web Service 的客户端类(这些客户端类只能在使用 Java 语言进行编程的时候使用)，而此 WSDL 文件描述了如何调用此 Web Service，由上述过程生成的 WSDL 文件如下：

```
<definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns="http://webservices.urms"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    targetNamespace="http://webservices.urms">
  <message name="putNewResource">
  <!—定义消息中包含的数据类型，type 属性中的值是标准 XML Schema 中定义的数据类型以及 types 元素中自定义的数据类型，由于此 Web Service 中所有的数据类型都是标准 XML Schema 中定义的数据类型，故 WSDL 设计文档中未包含 types 元素 - - >
  <! - -资源入库服务请求消息中第一个参数为 string 类型，此参数表示资源文件的名称 - - >
  <part name="string" type="partns:string"
    xml:partns="http://www.w3g.org/2001/XMLSchema" />
  <! - -资源入库服务请求消息中第二个参数为 string 类型，此参数表示上传的令牌号 - - >
  <part name="string0" type="partns:string"
    xml:partns="http://www.w3g.org/2001/XMLSchema" />
  <! - -资源入库服务请求消息中第三个参数为 base64Binary 类型，此参数表示具体上传的资源数据的字节数组 - - >
  <part name="bytes" type="partns:base64Binary"
    xml:partns="http://www.w3g.org/2001/XMLSchema" />
  <! - -资源入库服务请求消息中第四个参数为 int 类型，此参数表示上传的标志，其中 0 为上传开始，1 为上传继续，2 为上传结束，3 为一次操作完毕，4 为上传的是资源压缩包 - - >
  <part name="inVal" type="partns:int"
    xml:partns="http://www.w3g.org/2001/XMLSchema" />
  </message>
  <! —定义服务的消息类型，此处定义的是回应的类型 - - >
  <message name="putNewResourceResponse">
  <! - -资源入库服务回应消息数据类型为 boolean 类型，true 表示成功，false 表示失败 - - >
```

```
<part name = " result"  type = " partns : boolean"
  xml : partns = " http : //www. w3g. org/2001/
XMLSchema"  / >
</message >
<portType name = " BaseServicePort" >
<! - - 资源入库服务的名称 - - >
<operation name = " putNewResource" >
<! - - 资源入库服务的请求消息(可以看作是对服务调用的输入参数),为上面定义的 message 元素 - - >
<input message = " tns : putNewResource"  / >
<! - - 资源入库服务回应消息(可以看作是对服务调用的输出参数),为上面定义的 message 元素 - - >
< output  message = " tns : putNewResourceResponse"  / >
</operation >
</portType >
< binding  name = " BaseServicePortSoapBinding"
type = " tns : BaseServicePort" >
<! —soap : binding 的格式为 rpc,即所有送到服务的参数都要封包到一个 XML 元素中,该元素名字为下面的 operation 元素的 name 属性值,transport 元素指定了传输协议为 http - - >
<soap : binding style = " rpc"  transport = " http : //
schema. xmlsoap. org/soap/http"  / >
<operation name = " … " >
<! —定义了通过 http 传输时 soapAction 的值 - - >
<soap : operation soapAction = " "  style = " rpc"  /
>
<! —定义请求的编码方式 - - >
<input >
<soap : body use = " encoded"  namespace = " http : //webservices. urms"
  encodingStyle = " http : //schema. xmlsoap. org/
soap/encoding"  / >
</input >
<! —定义回应的编码方式 - - >
  <output >
    < soap : body use = " encoded"  namespace
= " http : //webservices. urms"
      encodingStyle = " http : //schema. xmlsoap.
org/soap/encoding"  / >
  </output >
</operation >
</binding >
<service name = " BaseService" >
< port  name = " BaseServicePort"  binding = "
tns : BaseServicePortSoapBinding" >
< soap : address  location = " http : //localhost :
7001/urmsws/urms. webservices? WSDL" / >
</port >
</service >
</definitons >
```

既然 WSDL 是 XML 文档,那么其也具有了 XML 文档跨平台的特性,所以,在异构系统中可以通过 WSDL 来获得调用 Web Service 的细节,从而使系统功能的重用得到最大限度的实现。在本文中,为了简化 Web Service 的调用,将 Web Service 的 URL 作为已知的链接传递给 Web Service 调用者,而不需将 Web Service 注册到 UDDI 注册中心,调用者也无需到 UDDI 注册中心搜寻所需要的 Web 服务。

4　在异构系统中调用 Web Service

要调用 Web Service,首先要获得 Web Service 的 URL,在本文中此 URL 已经由 Web Service 发布方传递给 Web Service 的调用方了。在此客户端使用异构的 Visual Studio . NET 来调用,调用的流程如下:首先,客户端根据 Web Service 传递过来的 URL,获得 Web Service 的 WSDL 文档,使用 Visual Studio . NET 根据 WSDL 生成 Web Service 的 SOAP 代理类;其次,在客户端使用生成的 SOAP 代理类象访问本地对象一样来访问这个 Web Service。

下面就是使用 Visual Studio . NET 调用 Web Service 的例子:

首先,在 Visual Studio . NET 的项目中添加 Web 引用,在 URL 中输入 Web Service 发布方传递过来的 URL,然后点击添加引用,Visual Studio . NET 就会根据输入的 URL 自动生成相应的 SOAP 代理类。

接着,使用代理类来调用 Web Service,其中 BaseService 是自动生成的代理类:

```
BaseService bs = new BaseService();
String tokeID = "00000000";
int mark = 0;
String name = "index.pdf";
FileInfo fi = new FileInfo("c:\temp\index.pdf");
byte[] a = new byte[fi.Length];
BinaryReader br = new BinaryReader(File.Open
("c:\index.pdf", FileMode.Open)); for(int i = 0; i
< fi.Length; i++){ a[i] = br.ReadByte(); }
Boolean res = bs.putNewResource(name, to-
keID, a, mark);
```

从上面可以看出，调用 Web Service 非常简单，就好像调用本地对象一样，非常简单。对于异构系统来说，使用 Web Service 可以使系统得到最大限度的重用，而无需在系统构架变化时又重新进行异构环境下的重复系统开发。

5 使用 Web Service 所需要考虑的问题

(1) 由于是基于 XML 的应用，Web Service 与生俱来地在拥有 XML 带来的一切优势的同时，不可避免地继承了 XML 所带来的一些限制。Web Service 通常需要大量的 CPU 资源。因为 XML 数据要经过多步处理才能被系统使用。首先是校验(validate)，检查它的格式是否符合 XML 的规范，以及根据应用程序定义(DTD 或 Schema)检查是否符合语义上的规范；然后还要进行解析(parse)，从 XML 文档分解出单个的元素；最后还要转换成应用程序所需要的二进制表达。同时，Web service 还意味着占用较多的内存资源。在进行 XML 解析的时候会产生大量的临时内存对象，特别是在处理 DOM 对象的时候。

(2) SOAP 的安全性。为了保证 SOAP 协议的简单性，SOAP 本身并没有提供安全性的保证。可以利用 SOAP 的 Header 块来进行一些身份认证的功能，或者利用 WS－Sercurity 来加强 SOAP 安全性。

参考文献

1 柴晓路著，《Web 服务架构与开放互操作技术》，清华大学出版社。

2 H. Kreger. Web Services Conceptual Architecture (WSCA 1.0). http://www－3. ibm. com/software/solutions/webservices/pdf/WSCA. pdf. May,2001.

3 B. Lublinsky, M. Farrell . Web Services—The Implementation Iceberg, EAI Journal, 6, 2002.

4 JCP. org. JSR 109, Implementing Enterprise Web Services. 2002－11－15.

5 开发 .NET Web 服务[CP/DK]. Microsoft 2002－10