基于 ASP. NET 的 Web 服务代理服务器的设计与实现

陈　旌　（中国科技大学研究生院 北京 100039）
周小暄　（北京科技大学管理学院 100081）

摘要：Web 服务代理服务器为外部网客户访问部署在内部网上的 Web 服务提供了一个统一的访问端点。本文以微软的 ASP. NET 为基础，设计并实现了一个 Web 服务代理服务器。

关键词：Web 服务　代理服务器　WSDL　UDDI　ASP. NET

1　引言

对于部署在企业内部网上的 Web 服务，如果直接从外部网上进行调用却存在着一些困难。这是因为部署在企业内部网上的 Web 服务一般情况下都位于防火墙之后，对于外部网是不可见的，也是无法直接访问的；另外，因为企业对外部网是不可控的，因此从外部网对 Web 服务进行访问，通常需要更高等级的安全性保障。

为了解决这些问题，在内部网和外部网之间可以部署一个作为中介的 Web 服务代理服务器。如果需要将在内部网上部署的 Web 服务暴露给外部网上的客户，服务提供者可以将此 Web 服务的端口和绑定信息导入到的这个 Web 服务代理服务器上，这样来自于外部网上的 Web 服务请求会被首先定向到 Web 服务代理服务器上，而 Web 服务代理服务器又会将这些请求重定向到内部网中的实际 Web 服务上。此外，Web 服务代理服务器还能够为客户提供 Web 服务请求的单点控制、访问和验证。本文接下来着重描述了一种 Web 服务代理服务器的设计与实现。

2　Web 服务代理服务器的结构和工作流程

本文介绍的 Web 服务代理服务器其结构和工作流程如图 1 所示，它是建立在微软 ASP. NET 基础上的，由管理控制台和代理服务器两部分组成。其工作流程为：

（1）服务提供者通过管理控制台在 Web 网关上部

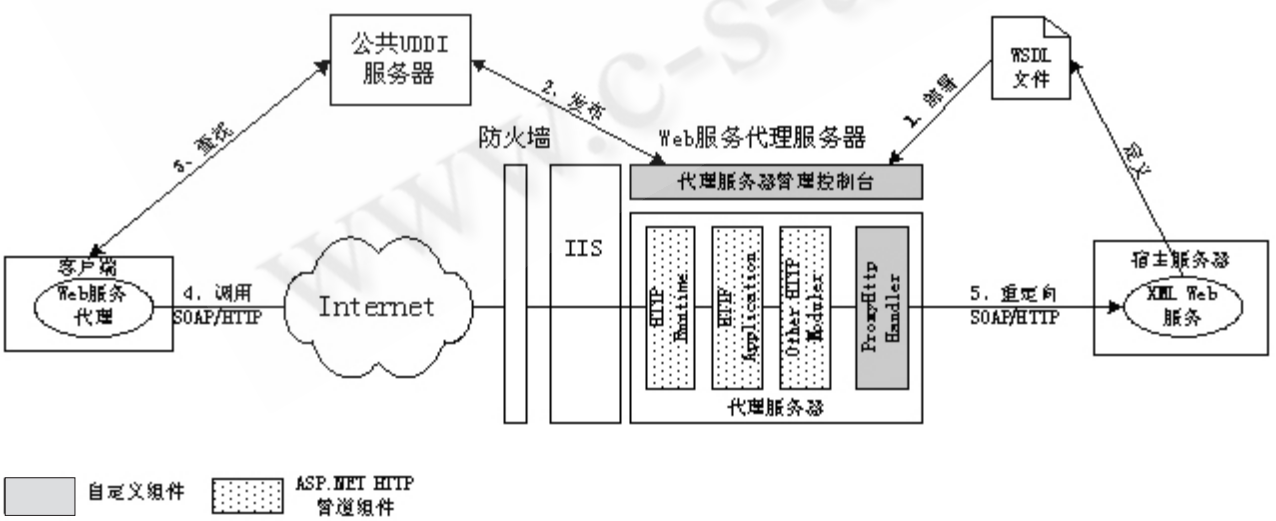

图 1　Web 服务代理服务器的结构和工作流程

署 Web 服务,即将在内部网中部署的用 WSDL 定义的 Web 服务映射成由代理服务器提供的新服务,并且为新的服务指定安全性措施,以及将新服务的访问许可授权给外部网上的客户。

(2) 在完成服务的部署后,管理控制台将新的服务发布到公共 UDDI 目录上。

(3) 外部网客户通过公共 UDDI 查找到部署到代理服务器上的新服务,并生成此服务的客户端代理。

(4) 外部网客户通过服务的客户端代理调用此服务,进而将调用消息发送到 Web 服务代理服务器上。

(5) IIS 服务器通过监听网络端口获得客户端代理发出的调用请求,接着将这个请求依次传递给代理服务器,由代理服务器完成重定向工作。同时,代理服务器还必须对访问请求进行访问验证。

3 Web 服务代理服务器的实现

3.1 管理控制台的实现

管理控制台是一个 ASP. NET 应用程序,由两个模块组成:Web 服务部署模块、Web 服务发布模块。

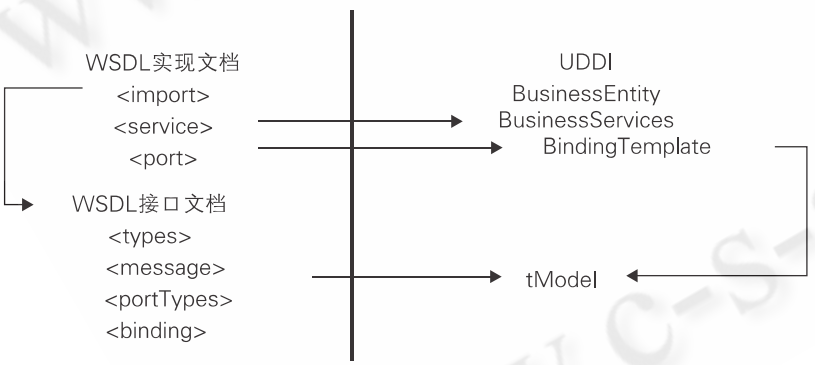

图 2　从 WSDL 到 UDDI 的映射

(1) Web 服务部署模块。将内部网上的 Web 服务部署到 Web 服务网关上,首先需要将定义此 Web 服务的 WSDL 文件上载到管理控制台上,接着为它指定在 Web 服务代理服务器上的名称(例如 UserIdentify)。确认后,服务部署模块首先在 IIS 服务器上为这个新的服务创建一个虚拟目录(目录名称即服务名称),并在这个目录下创建一个后缀为. asmpx 的文件(例如"UserIdentify. asmpx"以及一个 Web. config 文件。. asmpx 文件是一个 XML 格式的文件,只包括两行内容,其格式为:

<? XML version = "1.0"? >

< ServiceAddress > Service Location < \ServiceAddress >

Web. config 文件的格式与常规 ASP. NET Web 应用程序的 Web. config 文件相同。

接下来,调用. NET 框架中 System. Web. Services. Description 命名空间下的 ServiceDescription 类读取上载的 WSDL 文档:首先从此文件中读取位于 port 项下的服务端点地址,将这个地址(例如,http://ServicesHost/UserIdentify/UserIdentify. asmx)添加到. asmpx 文件的 ServiceAddress 项下;接着,将此文件分成两个新的 WSDL 文件,WSDL 实现文件和 WSDL 接口文件。WSDL 实现文件包含了原 WSDL 文件的 services 部分,只是将 Web 服务代理服务器作为了服务的端点地址(从 http://ServicesHost/UserIdentify/UserIdentify. asmx 改为 http://MyProxy/UserIdentify/UserIdentify. asmpx)。WSDL 接口文件则包含了原 WSDL 文档的 types、message、binding 和 portType 部分。代码略。

(2) 服务发布模块。当完成了服务部署后,管理控制台接着会调用服务发布模块。服务发布模块根据服务的新 WSDL 文档生成 UDDI 所需的信息并且将这些信息发布到公共的 UDDI 目录上,图 2 描述了从 WSDL 到 UDDI 之间的映射关系。其主要步骤为:

设置需要访问的 UDDI 节点访问属性,并且在这个 UDDI 节点上选择或者新注册一个用于保存服务详细信息的商业实体 businessEntity。

```
Publish. Url = "https://test. uddi. microsoft. com/publish";
Publish. User = "MyProxyPublisher";
Publish. Password = "12345678";
SaveBusiness sb = new SaveBusiness();
sb. BusinessEntities. Add();
sb. BusinessEntities[0]. Name = "MyGateWay";
//添加其他的 businessEntity 属性...
```

根据 WSDL 接口文件中的内容发布 tModel:通过 WSDL 接口文档中的 targetNamespace 设置 tModel 的

name 项;如果 WSDL 文件中有一个 documentation 项,那么就使用它来创建 tModel 的 Descriptions;将 WSDL 服务接口文档的网络可访问位置写入到 tModel 的 overviewDoc 中;将 tModel 发送到 UDDI 服务器上并返回一个 tModel key。

```
//读取 WSDL 接口文件
ServiceDescription interfaceDescription = ServiceDescription.Read("UserIdentify-interface.wsdl");
//根据接口文件的内容发布 tModel
TModel tMod = new TModel();
tMod.Name = interfaceDescription.TargetNamespace;
tMod.Descriptions.Add(interfaceDescription.Documention);
tMod.OverviewDoc.OverviewURL = "http://MyProxy/UserIdentify/UserIdentify-interface.wsdl";
...
SaveTModel stm = new SaveTModel(tModel);
TModelDetail savedTMod = stm.Send(myConn);
//返回 tModel key.
String tmodkey = savedTMod.TModels[0].TModelKey;
```

根据 WSDL 服务实现文件的内容发布 businessService。要完成这个操作,必须分别创建 businessService 和 bindingTemplate。首先创建 businessService:通过服务 name 项设置 businessService 的名称;通过 WSDL 的 documentation 项设置 businessService 的 Descriptions。接着创建 bindingTemplate:如果文件的 port 项包含有 documentation,那么使用它的内容来创建 bindingTemplate 的 Descriptions;使用 port 项内的 Location 的内容来设置 accessPoint;向 bindingTemplate 添加一个 tModelInstanceInfo,该元素包含 tModelKey;在 instanceDetails 中创建 overviewDoc,overviewDoc 包含 WSDL 服务实现文档的位置;在完成了所有的信息设置后,发送发布请求(代码略)。

3.2 代理服务器的实现

(1) 服务重定向的实现。如图 1 所示,代理服务器是建立在 ASP.NET 管道模型基础上的。ASP.NET 管道模型包括 HttpApplication 对象、各种 HTTP 模块对象和 HTTP 处理程序对象及其相关工厂对象。客户端的请求首先到达 IIS 服务器,IIS 服务器通过命名管道将文件类型请求转发给 ASP.NET 辅助进程中的 HttpRuntime 对象,HttpRuntime 对象检查请求,并将其转发给 HttpApplication 对象的实例。接着 HTTP 请求会被依次传递给各个 HTTP 模块对象(HttpModuler),这些 HTTP 模块对象通常并不对 HTTP 请求本身进行处理,只是完成访问认证,会话状态处理等一些工作。最后,ASP.NET 调用 HTTP 处理程序(HttpHandler)完成对请求的实际的处理并返回响应信息。

为了完成代理服务器的重定向功能,本文自定义了一个 HTTP 处理程序:ProxyHtttpHandler。自定义 HTTP 处理程序的方法很简单,只需要实现一个如下的接口:

```
interface IHttpHandler
{
  void ProcessRequest(HttpContext ctx);
  bool IsReuseable { get; }
}
```

ProxHttpHandler 的实现代码如下面所示。当用户对 Web 服务的调用(以 HttpContext 对象表示)被传递给 ProxyHttpHandler 后,ProxyHttpHandler 会首先从用户的调用请求中获取其中包含的 SOAP 消息;接着从 default.asmpx 文件中读取 Web 服务在内部网中的实际地址,并且以此地址为访问端点,转发得到的 SOAP 消息,这样外部网客户对代理服务器的调用请求就被重定向到了内部网中的实际 Web 服务上。最后,ProxyHttpHandler 还会将内部网 Web 服务返回的消息发送给外部网的客户。

```
using System;
using System.Configuration;
using System.Web;
using System.Net;
using System.Text;
using System.IO;
namespace ProxyHttpHandler
{
//定义 ProxyHttpHandler 处理类
public class ProxyHttpHandler: IHttpHandler
{
 public void ProcessRequest(HttpContext context)
 {
  //得到请求中包含的 SOAP 消息
```

```
Stream st = context.request.inputstream;
//读取 Default.asmpx 文件,获得内部网上 Web 服务的实际地址
XmlTextReader rdr = new XmlTextReader("default.asmpx");
rdr.MoveToNextAttribute();
string serviceuri = rdr.value;
Uri WebServiceURI = new Uri(serviceuri);V //将 SOAP 消息转发给内部网上的 Web 服务,完成请求消息的重定向
HttpWebRequest request = (HttpWebRequest)WebRequest.Create(WebServiceURI);
reques.Method = "POST";
request.ContentType = "text/xml";
Stream requestStream = request.GetRequestStream();
requestStream = st;
HttpWebResponse response = (HttpWebResponse)request.GetResponse();
//收到响应消息
Stream receiveStream = response.GetResponseStream();
StreamReader readStream = new StreamReader(receiveStream, Encoding.Default);
string returnmessage = readStream.ReadToEnd();
//返回收到的响应消息
context.Response.Write(returnmessage);
//关闭流
requestStream.Close();
readStream.Close();
response.Close();
context.Response.End();
}
public bool IsReusable
{
get
{
return true;
}
}
```

由于代理服务器上部署的 Web 服务的端点地址都是相应虚拟目录下的.asmpx 文件,因此必须将 ProxyHttpHandler 与.asmpx 文件类型关联起来,ASP.NET 辅助进程才会调用 ProxyHttpHandler 来处理外部网用户对 Web 服务的调用请求。要完成这项工作就必须在 Web.config 文件中加入如下的项:

```
<configuration>
<system.web>
<httpHandlers>
<add verb="*" path="*.asmpx" type="ProxyHttpHandler.ProxyHttpHandler,
ProxyHttpHandler" />
</httpHandlers>
</system.web>
</configuration>
```

另外,要想使 IIS 服务器将对.asmpx 文件的调用请求发送给 ASP.NET 辅助进程,还需要通过 IIS 服务器管理工具中的应用程序配置页面,将.asmpx 文件类型映射到 Aspnet_isapi.dll。

(2) 服务访问的安全性。由于本文介绍的代理服务器是建立在 IIS 服务器上的,因此可以充分利用 IIS 服务器提供的安全套接字 SSL、Windows 集成身份验证以及 PassPort 身份验证等用于确保安全性的方法。

4 结束语

本文介绍的 Web 服务代理服务器解决了外部网用户在访问部署在内部网中的 Web 服务时遇到的地址转换问题,并且提供了服务访问认证功能。内部网中的服务提供者可以将自己的 Web 服务发布在 Web 服务代理服务器上,并将 Web 服务代理服务器作为外部网客户的访问端点。

参考文献

1 Greg Flurry, IBM WebSphere 开发者技术期刊:探索 WebSphere Web 服务网关的新特征, www.ibm.com.cn.

2 Ashish Baner jee 等,康博译, C# Web 服务高级编程——使用.NET Remoting 和 ASP.NET 创建 Web 服务,清华大学出版社。