.NET 基础拾遗（7）：Web Service 的开发与应用基础 - 文章 - 伯乐在线

一、SOAP和Web Service的基本概念

Web Service基于SOAP协议，而SOAP本身符合XML语法规范。虽然.NET为Web Service提供了强大的支持，但了解其基本机制对于程序员来说仍然是必需的。

1.1 神马是SOAP协议？

SOAP协议的全称是简单对象访问协议（Simple Object Access Protocol），SOAP致力于以XML形式提供一个简单、轻量的用于在分散或分布环境中交换结构化和类型信息的机制。SOAP只规范对象访问的方式，而不限制具体实现的技术环境，这意味着SOAP协议是一种跨平台的协议：一个.NET客户端程序可以按照SOAP协议访问一个基于JavaEE技术体系结构的Web Service。SOAP访问仍然基于HTTP协议，同时其内容又以XML形式展现。

SOAP规范由四部分组成：

① SOAP信封（SOAP envelop）

② SOAP编码规则（SOAP encoding rules）

③ SOAP RPC表示（SOAP RPC representation）

④ SOAP绑定（SOAP binding）

这里不对这四部分展开介绍，通过下面的一个小例子来直观地认识一下。

（1）在Web服务端，打算对外提供一个公共方法来供客户端调用，而客户端则需要提供这个方法需要的参数，并且最终得到返回值。假设这个方法被申明在MySimpleService.asmx文件中：

C#

```
[WebMethod]
        public string GetSumString(int para1, int para2)
        {
                int result = para1 + para2;
                return result.ToString();
        }
```

（2）当客户端试图使用这个Web Service方法时，就需要向服务器端发出这样的一个HTTP请求：

1

```
2
3
4
5
6
7
8
9
10
11
12
13
14
15
POST /MySimpleService.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: &quot;http://tempuri.org/GetSumString&quot;
&lt;?xml version=&quot;1.0&quot; encoding=&quot;utf-8&quot;?&gt;
&lt;soap:Envelope xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-instance&quot;
xmlns:xsd=&quot;http://www.w3.org/2001/XMLSchema&quot;
xmlns:soap=&quot;http://schemas.xmlsoap.org/soap/envelope/&quot;&gt;
    &lt;soap:Body&gt;
        &lt;GetSumString xmlns=&quot;http://tempuri.org/&quot;&gt;
            &lt;para1&gt;250&lt;/para1&gt;
            &lt;para2&gt;250&lt;/para2&gt;
        &lt;/GetSumString&gt;
    &lt;/soap:Body&gt;
&lt;/soap:Envelope&gt;
```

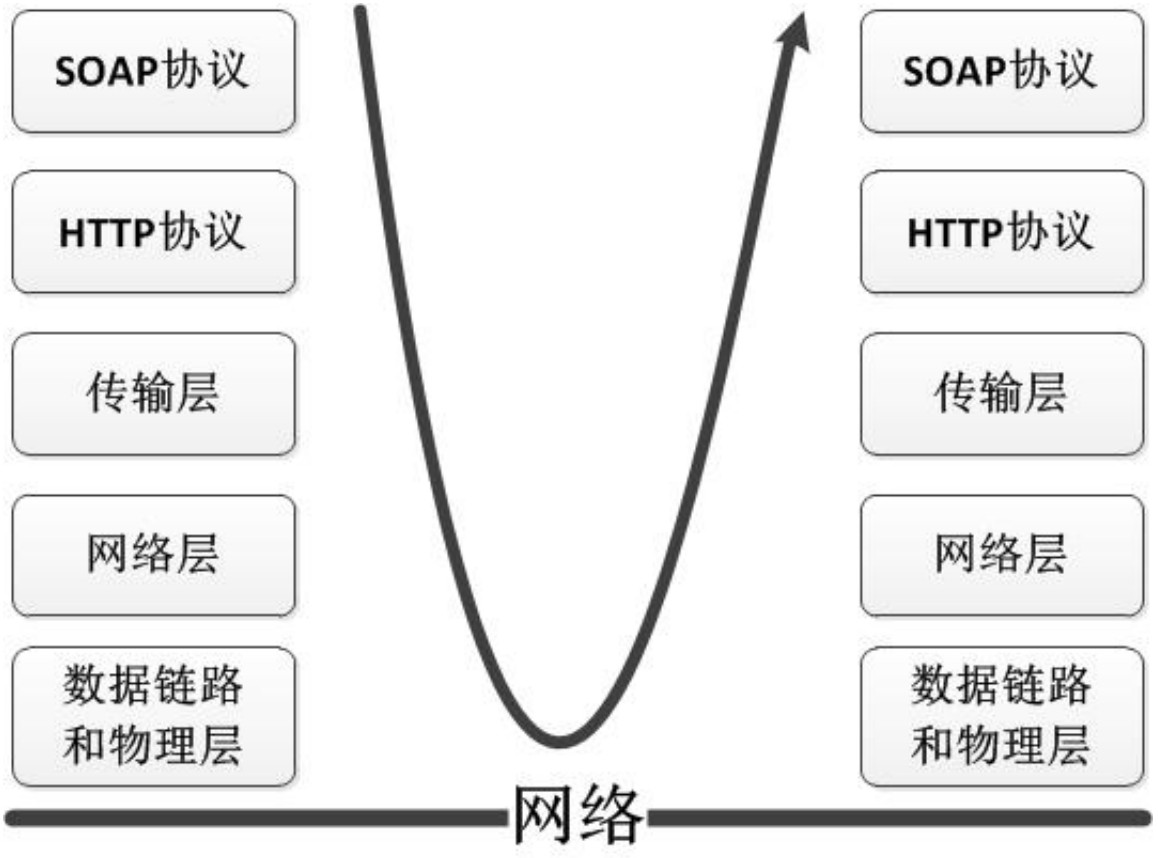
（3）等到Web Service服务器端接收到上面的请求之后，就可以进行相应的逻辑处理，并且返回结果。根据SOAP协议，HTTP响应如下形式：

```
1
2
3
4
5
6
7
8
9
```

```
10
11
12
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
&lt;?xml version=&quot;1.0&quot; encoding=&quot;utf-8&quot;?&gt;
&lt;soap:Envelope xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-instance&quot;
xmlns:xsd=&quot;http://www.w3.org/2001/XMLSchema&quot;
xmlns:soap=&quot;http://schemas.xmlsoap.org/soap/envelope/&quot;&gt;
    &lt;soap:Body&gt;
        &lt;GetSumStringResponse xmlns=&quot;http://tempuri.org/&quot;&gt;
            &lt;GetSumStringResult&gt;500&lt;/GetSumStringResult&gt;
        &lt;/GetSumStringResponse&gt;
    &lt;/soap:Body&gt;
&lt;/soap:Envelope&gt;
```

如此一来，客户端就得到了服务端的处理结果，换句话说，客户端已经得到了Web Service提供的服务。

> PS：最后，再说一下SOAP协议和HTTP协议，它们的关系非常类似于网络分层中的上下层协议，使用SOAP协议的双方将SOAP数据包放入HTTP报文之中，并且通过HTTP协议完成实际的传输，换句话说，SOAP是对HTTP的一个封装，下图说明了这一过程：



1.2 WSDL又是什么鬼，它有啥作用？

（1）WSDL介绍

WSDL（Web Service Description Language）是Web服务描述语言，它是一种由微软、IBM、Intel等大型供应商提出的语言规范，目的就是为了描述Web服务器所提供的服务，以供使用者参考。WSDL是一种复合XML语法规范的语言，它的设计完全基于SOAP协议，当一个Web Service服务器期望为使用者提供服务说明时，WSDL是最好的选择之一。

这里仍以上面的实例来说明，在Web服务端提供了这样一个方法：

C#

```
1
string GetSumString(int para1, int para2)
```

当服务端视图利用WSDL告诉客户端如何使用该方法时，就会提供下面的这样一个WSDL文件（仍然是一个XML）：

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
&lt;?xml version=&quot;1.0&quot; encoding=&quot;utf-8&quot;?&gt;
```

```
&lt;wsdl:definitions xmlns:tm=&quot;http://microsoft.com/wsdl/mime/textMatching/&quot;
xmlns:soapenc=&quot;http://schemas.xmlsoap.org/soap/encoding/&quot;
xmlns:mime=&quot;http://schemas.xmlsoap.org/wsdl/mime/&quot;
xmlns:tns=&quot;http://tempuri.org/&quot;
xmlns:soap=&quot;http://schemas.xmlsoap.org/wsdl/soap/&quot;
xmlns:s=&quot;http://www.w3.org/2001/XMLSchema&quot;
xmlns:soap12=&quot;http://schemas.xmlsoap.org/wsdl/soap12/&quot;
xmlns:http=&quot;http://schemas.xmlsoap.org/wsdl/http/&quot;
targetNamespace=&quot;http://tempuri.org/&quot;
xmlns:wsdl=&quot;http://schemas.xmlsoap.org/wsdl/&quot;&gt;
    &lt;wsdl:types&gt;
        &lt;s:schema elementFormDefault=&quot;qualified&quot;
targetNamespace=&quot;http://tempuri.org/&quot;&gt;
            &lt;s:element name=&quot;GetSumString&quot;&gt;
                &lt;s:complexType&gt;
                    &lt;s:sequence&gt;
                        &lt;s:element minOccurs=&quot;1&quot; maxOccurs=&quot;1&quot;
name=&quot;para1&quot; type=&quot;s:int&quot; /&gt;
                        &lt;s:element minOccurs=&quot;1&quot; maxOccurs=&quot;1&quot;
name=&quot;para2&quot; type=&quot;s:int&quot; /&gt;
                    &lt;/s:sequence&gt;
                &lt;/s:complexType&gt;
            &lt;/s:element&gt;
            &lt;s:element name=&quot;GetSumStringResponse&quot;&gt;
                &lt;s:complexType&gt;
                    &lt;s:sequence&gt;
                        &lt;s:element minOccurs=&quot;0&quot; maxOccurs=&quot;1&quot;
name=&quot;GetSumStringResult&quot; type=&quot;s:string&quot; /&gt;
                    &lt;/s:sequence&gt;
                &lt;/s:complexType&gt;
            &lt;/s:element&gt;
        &lt;/s:schema&gt;
    &lt;/wsdl:types&gt;
    &lt;wsdl:message name=&quot;GetSumStringSoapIn&quot;&gt;
        &lt;wsdl:part name=&quot;parameters&quot; element=&quot;tns:GetSumString&quot;
/&gt;
    &lt;/wsdl:message&gt;
    &lt;wsdl:message name=&quot;GetSumStringSoapOut&quot;&gt;
        &lt;wsdl:part name=&quot;parameters&quot;
element=&quot;tns:GetSumStringResponse&quot; /&gt;
    &lt;/wsdl:message&gt;
    &lt;!-- 这里省略其他定义 --&gt;
```

```
&lt;/wsdl:definitions&gt;
```

如上xml所示，在<wsdl:types>节点下，WSDL定义了GetSumString方法的名字：

```
1
&lt;s:element name=&quot;GetSumString&quot;&gt;
```

参数数量、每个参数的类型：

```
1
2
3
4
5
6
&lt;s:complexType&gt;
     &lt;s:sequence&gt;
            &lt;s:element minOccurs=&quot;1&quot; maxOccurs=&quot;1&quot;
name=&quot;para1&quot; type=&quot;s:int&quot; /&gt;
            &lt;s:element minOccurs=&quot;1&quot; maxOccurs=&quot;1&quot;
name=&quot;para2&quot; type=&quot;s:int&quot; /&gt;
     &lt;/s:sequence&gt;
&lt;/s:complexType&gt;
1
2
3
4
5
6
7
&lt;s:element name=&quot;GetSumStringResponse&quot;&gt;
     &lt;s:complexType&gt;
            &lt;s:sequence&gt;
                 &lt;s:element minOccurs=&quot;0&quot; maxOccurs=&quot;1&quot;
name=&quot;GetSumStringResult&quot; type=&quot;s:string&quot; /&gt;
            &lt;/s:sequence&gt;
     &lt;/s:complexType&gt;
&lt;/s:element&gt;
```

通过完整的描述，使用者就能够了解如何使用该Web服务了。

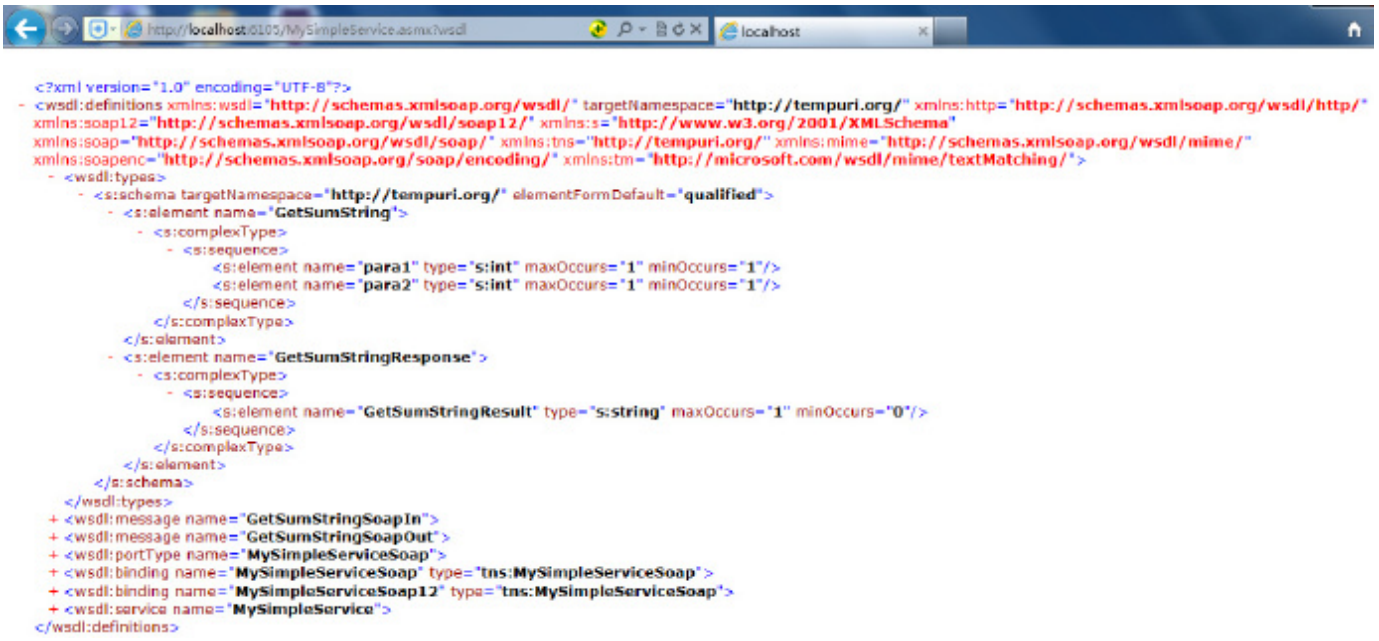
（2）获取和使用WSDL

当Web Service服务器提供WSDL时，就可以通过特定的工具获得WSDL文件。最直接的方式就是在URL中直接添加WSDL参数，来发送得到WSDL文件的请求，如下所示：

```
1
```

http://localhost:6105/MySimpleService.asmx?wsdl

这时点击回车就可以得到如下图所示的WSDL结果：



```
<?xml version="1.0" encoding="UTF-8"?>
- <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://tempuri.org/" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://tempuri.org/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/">
  - <wsdl:types>
    - <s:schema targetNamespace="http://tempuri.org/" elementFormDefault="qualified">
      - <s:element name="GetSumString">
        - <s:complexType>
          - <s:sequence>
              <s:element name="para1" type="s:int" maxOccurs="1" minOccurs="1"/>
              <s:element name="para2" type="s:int" maxOccurs="1" minOccurs="1"/>
            </s:sequence>
          </s:complexType>
        </s:element>
      - <s:element name="GetSumStringResponse">
        - <s:complexType>
          - <s:sequence>
              <s:element name="GetSumStringResult" type="s:string" maxOccurs="1" minOccurs="0"/>
            </s:sequence>
          </s:complexType>
        </s:element>
      </s:schema>
    </wsdl:types>
  + <wsdl:message name="GetSumStringSoapIn">
  + <wsdl:message name="GetSumStringSoapOut">
  + <wsdl:portType name="MySimpleServiceSoap">
  + <wsdl:binding name="MySimpleServiceSoap" type="tns:MySimpleServiceSoap">
  + <wsdl:binding name="MySimpleServiceSoap12" type="tns:MySimpleServiceSoap">
  + <wsdl:service name="MySimpleService">
</wsdl:definitions>
```

1.3 Web Service中如何处理附件？

尽管Web Service提供的方法的参数类型没有任何限制，也就意味着所有的附件可以通过字节数组来进行传递，但是把字节流直接内嵌在SOAP消息的做法有很多问题，这也曾经成为XML语法和SOAP协议被诟病的原因。这里主要介绍一下XOP的概念。

在XOP出现之前，SOAP处理二进制数据的方式都很简单，比如当一个Web Service服务端提供了如下的方法时：

```
1
void UploadSmallAttach(Byte[] attachment)
```

客户端调用该Web Service，只需要发出下面这样的SOAP请求即可：

```
1
2
3
4
5
6
7
8
&lt;?xml version=&quot;1.0&quot; encoding=&quot;utf-8&quot;?&gt;
&lt;soap:Envelope xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-instance&quot;
xmlns:xsd=&quot;http://www.w3.org/2001/XMLSchema&quot;
xmlns:soap=&quot;http://schemas.xmlsoap.org/soap/envelope/&quot;&gt;
    &lt;soap:Body&gt;
        &lt;UploadSmallAttach xmlns=&quot;http://tempuri.org/&quot;&gt;
            &lt;attachment&gt;D0CF11E0A1B11AE10000000000000000000000003E0000300FEFF0900060
```

000000000000000000000060000000000000000000DE0200000000000000000000010000000000000000FEFFFFFFFF0000000000000000000000D80200000000000D9020000DA02000DB02000000DC020000DD0200000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF</attachment>

</UploadSmallAttach>

</soap:Body>

</soap:Envelope>

如上所示，其中<attachment>节点下的一大堆字符，就是某个文件的字节流。通过这种方式，确实是可以实现传送二进制附件的功能的，但这样的处理过于粗略，且传输没有任何优化。W3C为此特别指定了XOP规范。

XOP（XML-binary Optimized Packages）意为XML二进制打包，它把二进制数据流从SOAP消息中分离出来，进行单独打包。上述的客户端请求如果使用XOP规范的话，将转变为如下结果：

C#

Content-Type: image/png

Content-Transfer-Encoding: binary

Content-ID: <sample@book.com>

D0CF11E0A1B11AE1000000000000000000000000003E0000300FEFF0900060000000000000000000000060000000000000000000DE0200000000000000000000010000000000000000FEFFFFFFFF0000000000000000000000D80200000000000D9020000DA02000DB02000000DC020000DD0200000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF二、使用.NET开发Web Service

.NET为Web Service提供了全面的支持，无论是创建Web Service还是访问Web Service，使用.NET都能快速有效地完成需求。

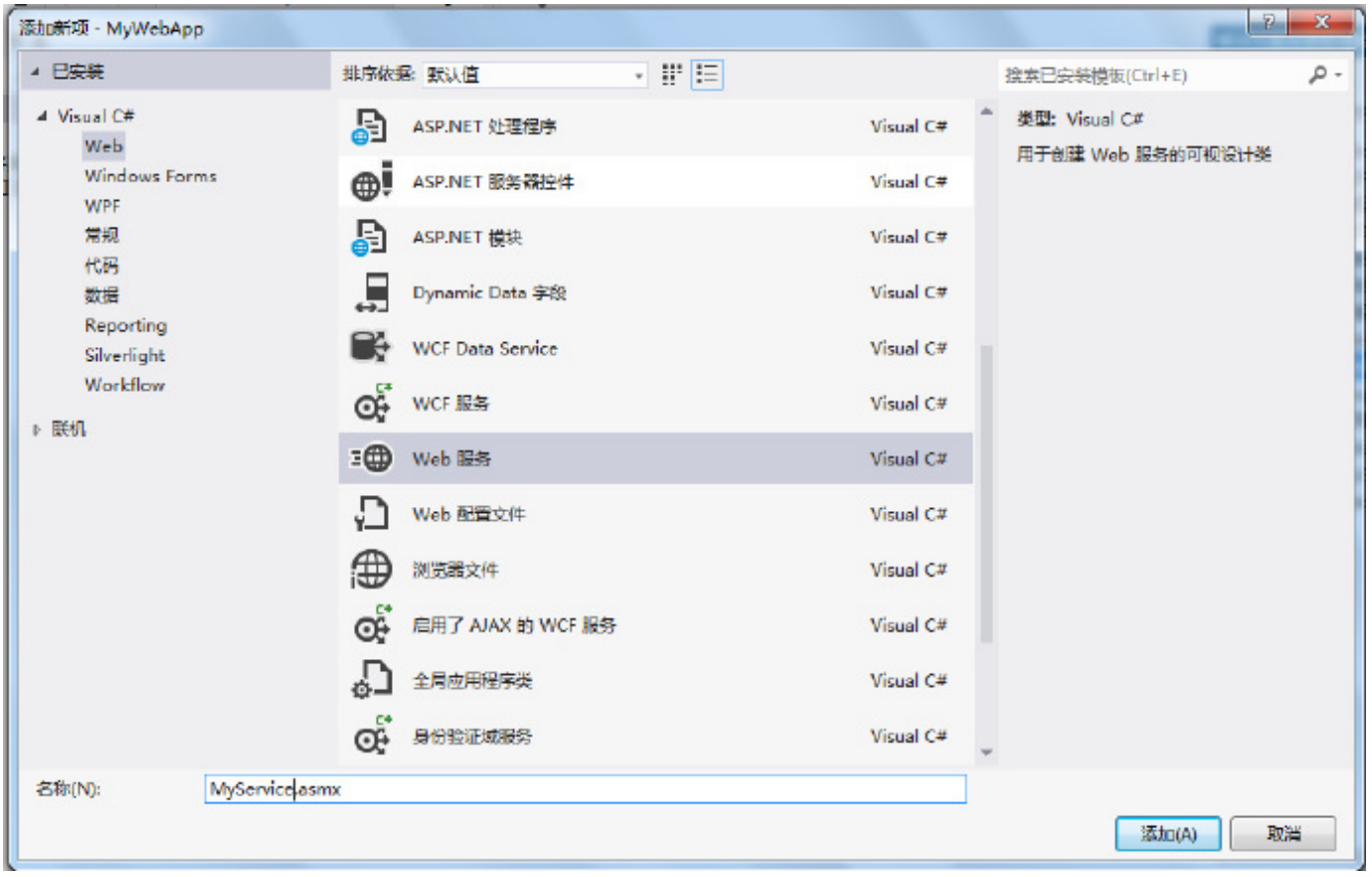
2.1 如何在.NET中创建Web Service?

（1）使用WebMethod特性创建Web Service

在.NET中，所有Web Service的资源被定义为asmx文件，而在ASP.NET被安装时，asmx文件也会在IIS中被注册成由ASP.NET组件来处理。也就是说，一个asmx文件和其后台代码asmx.cs组成了一个Web Service资源。

为了让我们能够把注意力集中在逻辑的处理上，而忽略SOAP通信的工作，.NET提供了Web Service类型和WebMethod特性。在继承自Web Service类型的公共方法上添加WebMethod特性，就可以申明为一个Web Service方法。

① 创建一个Web服务



② asmx文件只是简单地声明了后台代码的位置，而不包含任何工作代码。后台代码都在asmx.cs中：

```
C#/// <summary>
        /// MySimpleService 的摘要说明
        /// </summary>
        [WebService(Namespace = "http://tempuri.org/")]
        [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
        [System.ComponentModel.ToolboxItem(false)]
        // 若要允许使用 ASP.NET AJAX 从脚本中调用此 Web 服务，请取消注释以下行。
        // [System.Web.Script.Services.ScriptService]
        public class MySimpleService : System.Web.Services.WebService
        {
                [WebMethod]
                public string GetSumString(int para1, int para2)
                {
                        int result = para1 + para2;
                        return result.ToString();
                }
        }
```

> Luckily，WebService和WebMethod为我们提供了完全包装好的SOAP处理功能，而在大多数情况下，我们所要做的就是继承和使用它们。

（2）创建自定义的类型来处理对asmx文件的请求

在ASP.NET的处理机制中，所有的HTTP请求通道都通过管道来寻找处理程序。我们所熟悉的WebForm和

WebService，都是实现了IHttpHandler接口的Http处理程序，这导致了它们有能力处理特定的Http请求。事实上，我们可以通过配置Web.config来自定义Http处理程序和资源的映射匹配关系，如同下面的配置所展示的一样：

```
1
2
3
4
5
&lt;httpHandlers&gt;
            &lt;add verb=&quot;*&quot; path=&quot;*.ashx&quot;
type=&quot;System.Web.UI.SimpleHandlerFactory&quot;/&gt;
            &lt;add verb=&quot;*&quot; path=&quot;*.aspx&quot;
type=&quot;System.Web.UI.PageHandlerFactory&quot;/&gt;
            &lt;add verb=&quot;*&quot; path=&quot;*.asmx&quot;
type=&quot;System.Web.Services.Protoclos.WebServiceHandlerFactory&quot;/&gt;
&lt;/httpHandlers&gt;
```

实现IHttpHandler接口很简单，必须的工作就仅仅是实现一个ProcessRequest方法和一个只读属性：

```
C#
public void ProcessRequest(HttpContext context)
public bool IsReusable
```

在HttpContext类型的上下文对象中包含了Http请求（HttpRequest），也包含了Http返回类型对象（HttpResponse），并且允许程序员往里面写入希望的返回内容。IsReusable属性则返回当前对象是否可被重用来应对所有类似的Http请求。

鉴于此，我们可以自己实现一个实现IHttpHandler接口的处理程序，在配置文件中将其绑定到.asmx文件上，就可以实现Web Service方法了。当然，为了符合SOAP规范，我们需要在ProcessRequest方法中解析SOAP请求，并且把返回值放入一个SOAP包中。

下面的代码示例展示了如何自定义asmx处理程序（这里只展示了如何编写实现IHttpHandler接口的类型并使其工作，省略了繁琐的SOAP解析和组织工作）

① 新建一个ashx程序，实现IHttpHandler接口

```
C#/// <summary>
        /// MySimpleHandler 的摘要说明
        /// </summary>
        public class MySimpleHandler : IHttpHandler
        {
                public void ProcessRequest(HttpContext context)
                {
                        context.Response.Write("<h1>Hello Web Service!</h1>");
                }
```

```
        public bool IsReusable
        {
                get
                {
                        return true;
                }
        }
}
```

② 修改Web.config文件，加入自定义HttpHandler类型

```
1
&lt;add verb=&quot;*&quot; path=&quot;*.asmx&quot;
type=&quot;MyWebApp.MySimpleHandler,MyWebApp&quot;/&gt;
```

（3）自定义Web Service资源文件和处理程序

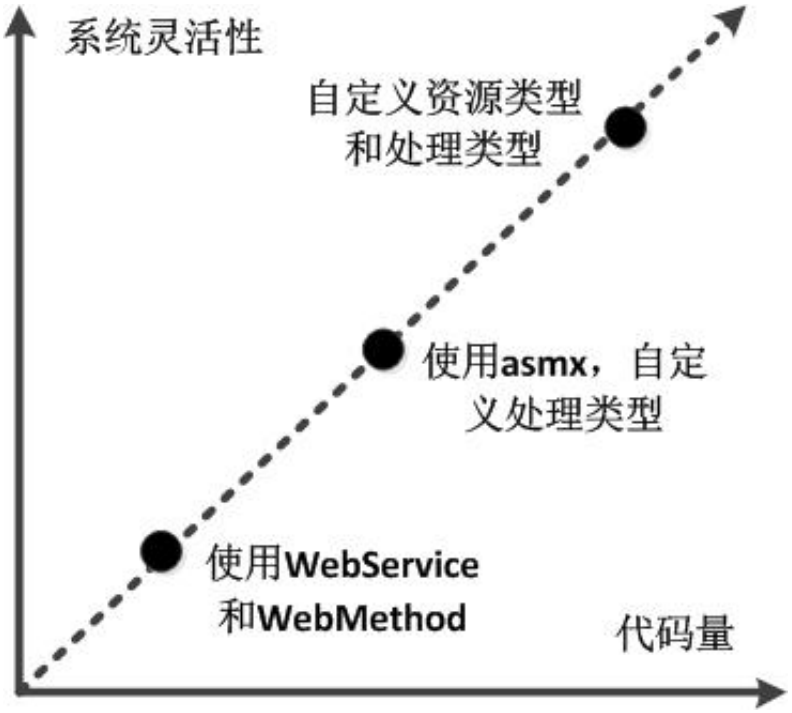
事实上，我们完全可以定义一种新的资源文件而不是采用asmx文件名，只要我们为其定制HttpHandler程序。需要做的工作为：

① 定义一个新的文件格式如asnew，在IIS中匹配asnew和aspnet_isapi.dll处理程序；

② 自定义一个实现了IHttpHandler接口的类型，在这个类型中实现SOAP；

③ 在需要使用asnew文件的站点配置文件中绑定asnew和新的Http处理类型；

这样的方案显然可以为系统带来更大的灵活性，但同时也意味着更大的代码量。下图展示了三种实现Web Service的方法之间的关系：



2.2 WebMethod特性包含哪些属性，各有神马用处？

WebMethod特性在Web Service中被用来申明一个公开方法，了解其使用方法是在正确编写Web Service的基础。在WebMethod特性中，一共包含了6个属性，这6个属性对WebMethod的使用非常重要。

（1）BufferResponse属性

该属性表明是否启用对Web Service方法响应的缓冲。当设置为true时，Web Service方法将响应序列化到内存缓冲区中，直到缓存区被用满或者响应结束后，响应才会被发送给客户端。相反，设置为false时，.NET默认以16KB的块区缓冲响应，响应在被序列化的同时将会被不断发送给客户端，无论该响应是否已经完全结束。

> PS：默认BufferResponse被设置为true。当Web Service要发送大量数据流给客户端时，设置BufferResponse为false时可以防止大规模数据一次性刷新到内存，而对于小量数据，设置为true则可以有效地提高性能。

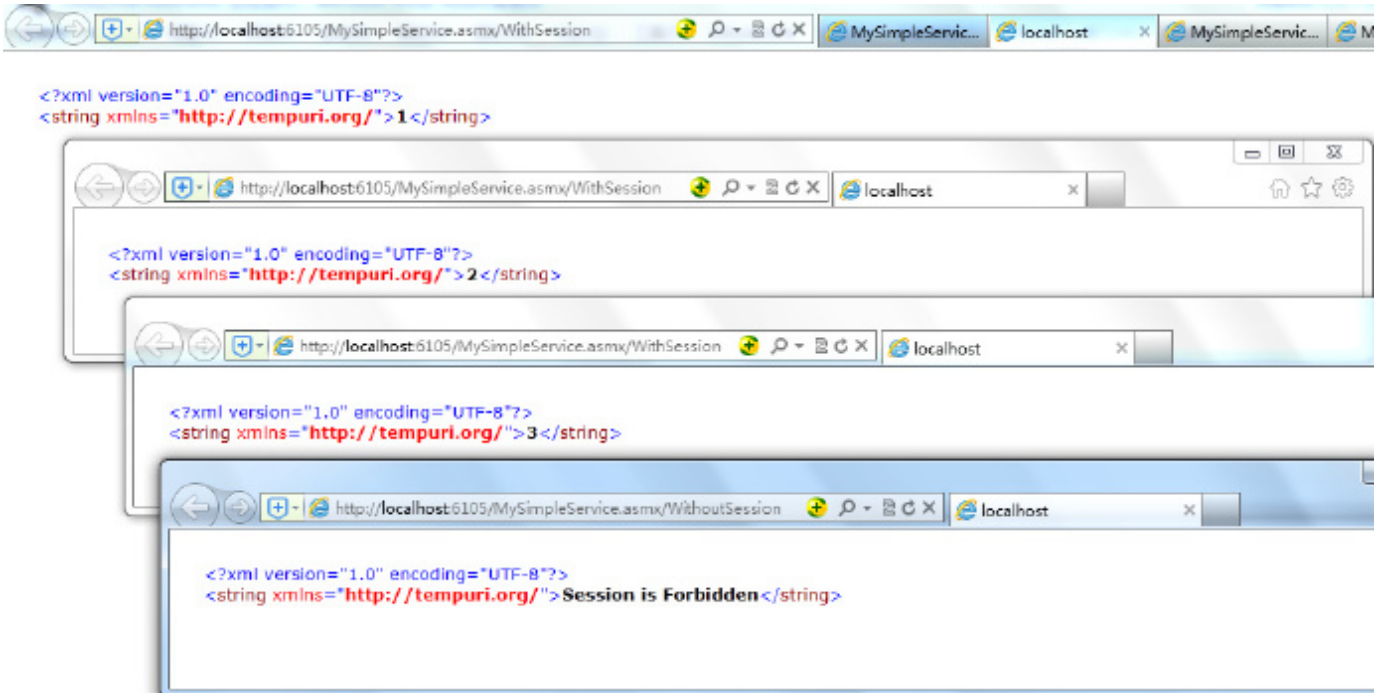
（2）EnableSession属性

该属性指定是否启用会话状态。如果为true，则启用，为fasle则禁用。默认被设置为false。

C#

```
public class MySimpleService : System.Web.Services.WebService
        {
                [WebMethod(EnableSession = true)]
                public string WithSession()
                {
                        return TryGetSession();
                }
                [WebMethod(EnableSession = false)]
                public string WithoutSession()
                {
                        return TryGetSession();
                }
                private string TryGetSession()
                {
                        if (Session == null)
                        {
                                return "Session is Forbidden";
                        }
                        if (Session["number"] == null)
                        {
                                Session["number"] = 0;
                        }
                        Session["number"] = (int)Session["number"] + 1;
                        return Session["number"].ToString();
                }
        }
```

分别访问WithSession和WithoutSession方法，结果如下图所示：



（3）CacheDuration属性

该属性指示启用对Web Service方法结果的缓存。服务端将会缓存每个唯一参数集的结果，该属性的值指定服务器端应该对结果进行多少秒的缓存处理。如果该值为0，则禁用对结果进行缓存；如果不为零，则启用缓存，单位为秒，意为设置多少秒的缓存时间。默认该值被设为0。

C#

```
[WebMethod(CacheDuration = 10, EnableSession = true)]
        public string WithCache()
        {
                if (Session["number"] == null)
                {
                        Session["number"] = 0;
                }
                Session["number"] = (int)Session["number"] + 1;
                return Session["number"].ToString();
        }
```

上面的WithCache方法设置了10秒的缓存时间，即10秒内的访问都会得到一样的结果。

（4）Description属性

该属性很简单，提供了对某个Web Service方法的说明，并且会显示在服务帮助页上面。

（5）MessageName属性

该属性是Web Service能够唯一确定使用别名的重载方法。除非另外指定，默认值是方法名称。当指定MessageName时，结果SOAP消息将反映该名称，而不是实际的方法名称。

当Web Service提供了两个同名的方法时，MessageName属性会很有用，这一点将会体现在WSDL中：

C#

```
[WebMethod(MessageName="HelloWorld1")]
        public string HelloWorld(int num)
        {
                return num.ToString();
        }
        [WebMethod(MessageName = "HelloWorld2")]
        public string HelloWorld()
        {
                return "Hello World!";
        }
```

（6）TransactionOption属性

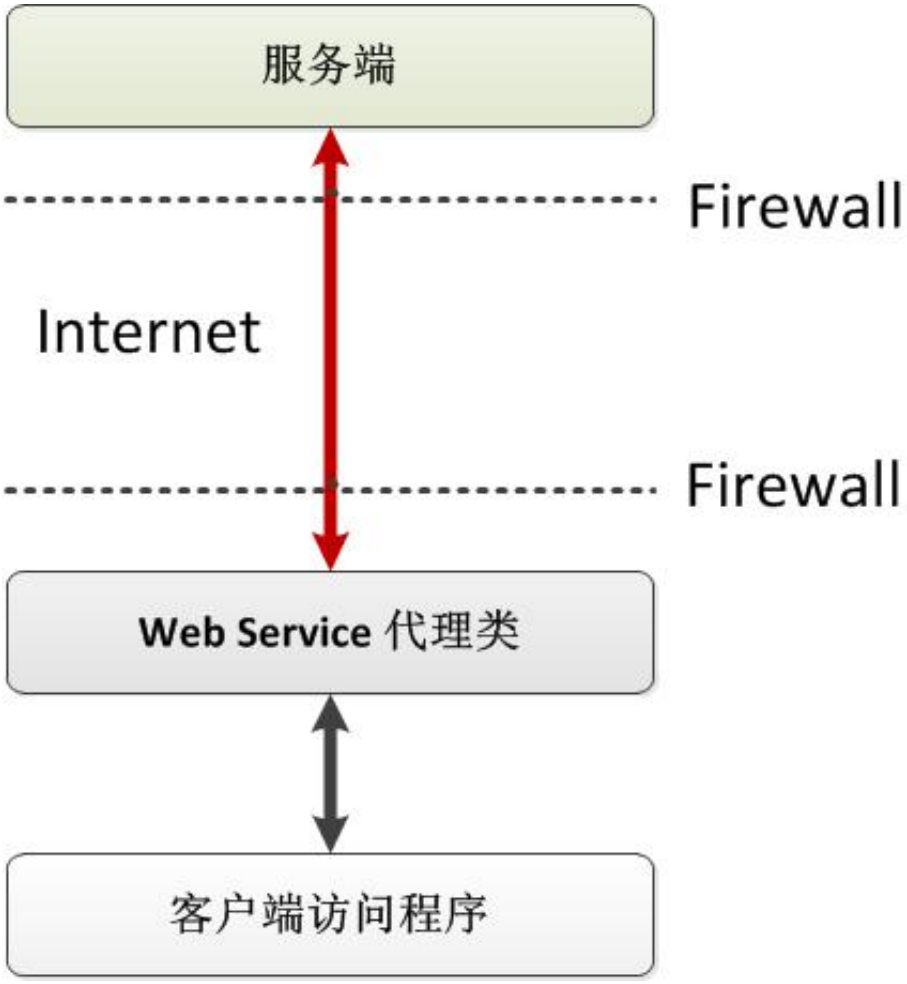
该属性用以设置Web Service方法的事务特性，在.NET中事务模型是基于声明性的，而不是编写特定的代码来处理提交和回滚事务。在Web Service中，可以通过TransactionOption属性来设置该方法是否需要被放入一个事务之中。如果申明了事务属性，执行Web Service方法时引发异常会自动终止事务，相反如果未发生任何异常，则自动提交事务。

事务最常用的一个场景就是数据库访问，所以该属性在利用Web Service实现的分布式数据库访问中就特别有用。

2.2 如何生成Web Service代理类型？

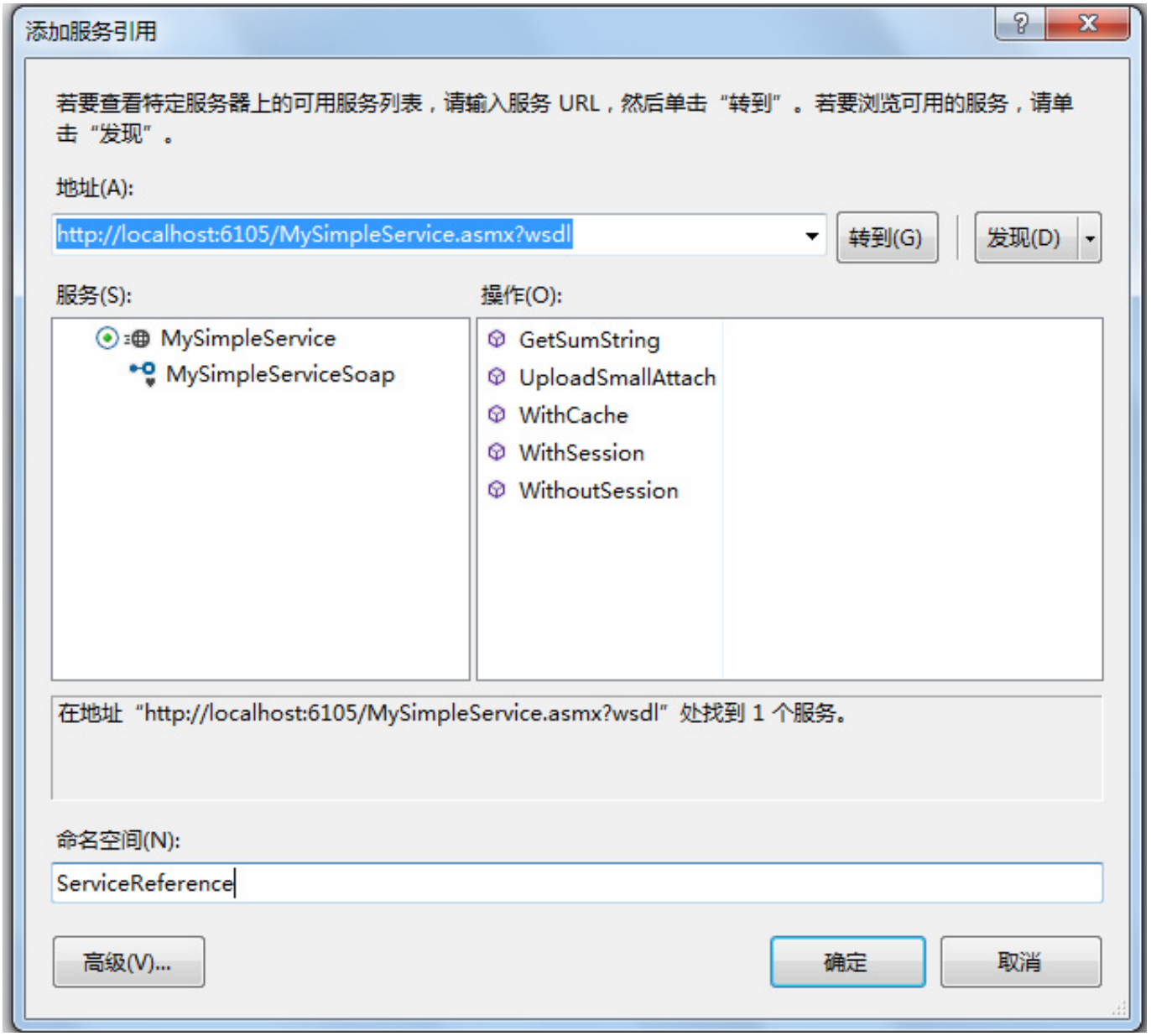
（1）Web Service代理类的概念

所谓的代理类，就是SOAP协议的代理类型，它使得我们可以通过调用本地的类型方法（代理类），来达到访问Web Service方法的目的。代理类的最终目的就是将程序员从繁琐的SOAP消息处理和XML解析中解放出来，而专注于逻辑工作。下图说明了代理类的作用：

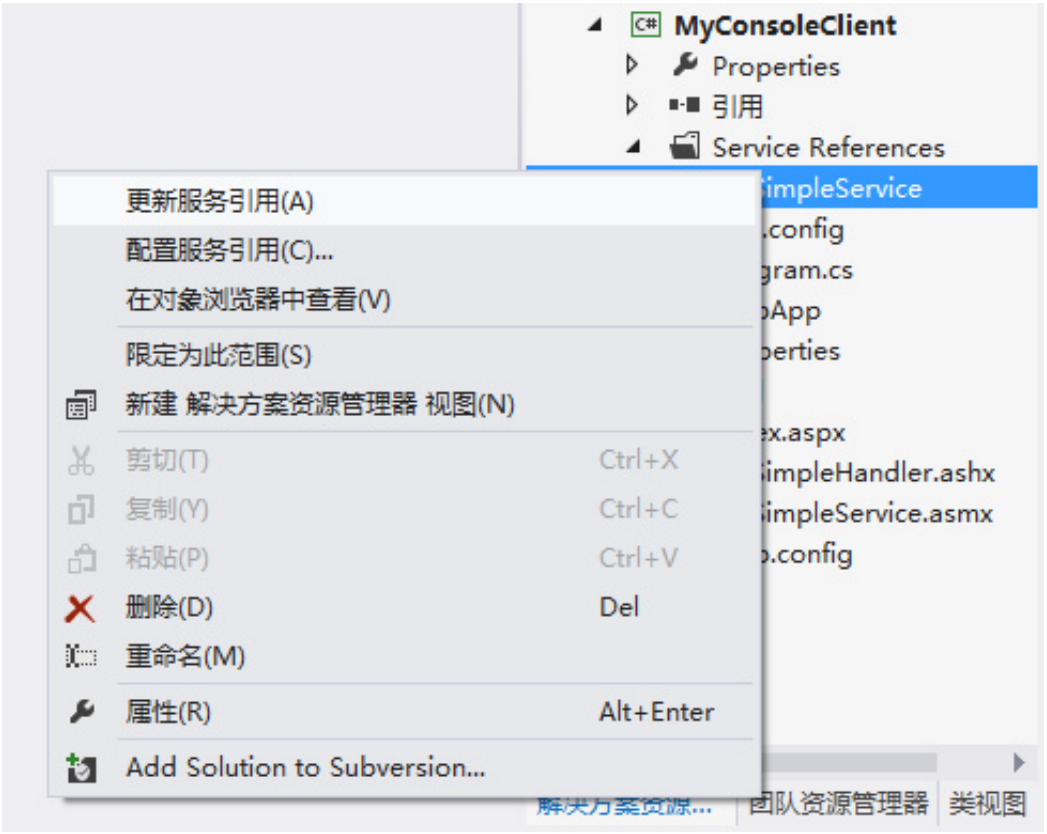


（2）如何生成Web Service代理类

在Visual Studio中提供了一个很简单的生成Web Service代理类的方法就是Web引用，如下图所示：



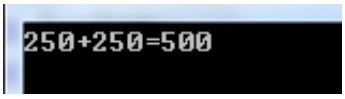
当Web引用被添加后，一个代理类型也会自动生成。并且，当服务端的Web Service更新后，我们只需要简单地更新一下Web引用，就可以方便地更新代理类型。



在客户端逻辑中，只需要调用代理类的对应接口就OK，十分简单：

```
C#
class Program
        {
                static void Main(string[] args)
                {
                        using (MySimpleServiceSoapClient proxy = new
MySimpleServiceSoapClient())
                        {
                                string result = proxy.GetSumString(250, 250);
                                Console.WriteLine("250+250={0}", result);
                        }
                        Console.ReadKey();
                }
        }
```

执行结果为：



2.3 简述.NET中Web Service的异常机制

即使有了本地的代理类，调用Web Service方法还是调用本地方法有所区别，如果Web Service出现了异常，那么这些异常信息就需要被封装在SOAP信息中发送回客户端。

（1）SOAP中对异常的规定

SOAP规定了异常信息的携带方式，那就是全被放入fault节点中。fault节点必须是Body节点的子节点，而且一个SOAP消息只能出现一个fault节点。

子节点	描述
<faultcode>	识别故障的代码
<faultstring>	供认阅读的有关故障的说明
<faultactor>	是谁引发异常
<detail>	存留设计Body元素的应用程序专用错误信息

其中faultcode是一个错误码，其取值和每个值所代表的含义都在SOAP中有所定义，下表列出了所有faultcode及其含义：

faultcode节点值	描述
VersionMismatch	SOAP Envelop元素的无效命名空间被发现
MustUnderstand	Header元素的一个直接子元素无法被理解
Client	消息被不正确地构成，或包含了不正确的信息
Server	服务器有问题，因此无法处理进行下去

（2）服务端对未捕获异常的处理

在使用WebService类型和WebMethod特性创建Web Service的情况下，服务器端的异常都会被捕捉，并且所有异常都会被放入到SoapException类型中，并且返回给客户端。我们可以在服务端代码中直接使用SoapException异常，通过设置其属性来告知客户端：

① Message：原始异常的Message属性

② Code：服务器异常码

③ Actor：Web Service方法的URL

④ Detail：空引用，但有一个空的详细信息元素存在于故障元素之中

服务端会把SoapException放入Fault节点之中并返回给客户端，以此来告知服务端发生的异常。

（3）客户端代理类对fault节点的处理

如果使用.NET自动生成的Web Service代理类，那么它将能够自动地识别fault节点，并且还原为SoapException异常。这里可以通过下面的一段代码示例来直观地了解这一点：

① 首先在Web Service方法中直接抛出一个异常，如下代码所示：

C#

```
class ServiceClient
        {
                static void Main(string[] args)
                {
                        using (MySimpleServiceSoapClient proxy = new
MySimpleServiceSoapClient())
                        {
                                try
                                {
                                        // 这里异常将会被代理类抛出
                                        proxy.HelloException();
                                }
                                catch (SoapException ex)
                                {
                                        // 打印异常信息内容
                                        Console.WriteLine("Actor:{0}", ex.Actor);
                                        Console.WriteLine("CodeName:{0}", ex.Code.Name);
                                        Console.WriteLine("Detail:{0}",
ex.Detail.InnerText);
                                        Console.WriteLine("Message:{0}", ex.Message);
                                }
                        }
                        Console.ReadKey();
                }
        }
```

参考资料

（1）朱毅，《进入IT企业必读的200个.NET面试题》

（2）张子阳，《.NET之美：.NET关键技术深入解析》

（3）王涛，《你必须知道的.NET》

加入伯乐在线专栏作者。扩大知名度，还能得赞赏！详见《招募专栏作者》

1 赞 2 收藏 评论

合作联系

Email：bd@Jobbole.com

QQ： 2302462408 （加好友请注明来意）

更多频道

小组 - 好的话题、有启发的回复、值得信赖的圈子
头条 - 分享和发现有价值的内容与观点
相亲 - 为IT单身男女服务的征婚传播平台
资源 - 优秀的工具资源导航
翻译 - 翻译传播优秀的外文文章
文章 - 国内外的精选文章
设计 - UI,网页，交互和用户体验
iOS - 专注iOS技术分享
安卓 - 专注Android技术分享
前端 - JavaScript, HTML5, CSS
Java - 专注Java技术分享
Python - 专注Python技术分享