第 5 章　XML 与 Web 服务

Web 服务是一种跨平台的共享服务，是当前较为流行的一种应用程序。而 Web 服务需要通过 XML 文档来描述，并利用 XML 的跨平台特性而被任何支持 XML 的技术所访问。

本章介绍 XML 与 Web 服务之间的联系，并重点介绍 Web 服务的描述语言 WSDL（Web Services Description Language）。

本章学习目标：

- ❑ 了解 Web 服务的作用及相关技术
- ❑ 了解 WSDL 文档结构
- ❑ 掌握 WSDL 各个元素的属性和用法
- ❑ 能够自定义 WSDL
- ❑ 了解 WSDL 使用原理
- ❑ 掌握如何通过 WSDL 调用 Web 服务

5.1 XML 与 Web 服务

Web 服务是一种应用程序，通过将应用程序定义为 Web 服务，可以在互联网的不同平台下实现共享。而 Web 服务使用开放的 XML 标准来描述、发布、发现、协调和配置，本节介绍 XML 与 Web 服务之间的联系，以及 WSDL 语言的概述。

5.1.1 Web 服务

Web 服务是 Web 服务器提供的一个应用程序，或者是执行代码的程序块，它通过标准的 XML 协议来展示它的功能，用于开发分布式互操作的应用程序。

一个 Web 服务就是一个应用程序，它提供了一个能够通过 Web 进行调用的 API。一个定义好的 Web 服务的应用程序可以被其他网页调用，直接发送一个 HTTP GET 请求即可获得需要的数据。

而 Web Service 使用 XML 标准来描述，是它能够在不同的平台上被直接调用的基础，具有以下几个特点。

- ❑ Web 服务是可互操作的、一种优秀的分布式应用程序。
- ❑ Web 服务具有普遍性，它使用 HTTP 和 XML 进行通信。任何支持这些技术的设备都可以拥有和访问 Web 服务。
- ❑ Web 服务可以轻松地穿越防火墙，真正实现自由通信。

Web 服务拥有一套完整的技术和标准，以实现其跨平台的可互操作性。这样的技术

包括对 Web 服务的描述、调用和实现等。只有对 Web 服务有了定义描述，才能够对其进行调用和实现。相关技术和标准如下所示。

- **Web 服务的描述** XML 和 XSD。
- **Web 服务的访问** SOAP。
- **Web 服务功能的描述文档** WSDL。
- **Web 服务的资源共享** UDDI。
- **Web 服务的远程调用** RPC。

1．XML 语言和 XSD

XML 是 Web 服务中表示和封装数据的基本格式。XML 易于建立和分析，最重要的是它既与平台无关、也与厂商无关。这是实现 Web 服务的互操作性的基础。

XML 解决了数据描述的问题，但它没有定义一套标准的数据类型，而标准的数据类型对于互操作性的实现是很重要的。W3C 制定了 XML Schema（XSD）解决这个问题，它定义了一套标准的数据类型，并给出一种语言来扩展这套数据类型。

创建 Web 服务所使用的数据类型都必须被转换为 XSD 类型，开发人员所使用的工具通常能够自动完成这个转换。

2．SOAP 协议与 RPC

SOAP（Simple Object Access Protocol）即简单对象访问协议，它可以运行在任何其他传输协议之上，是用于交换 XML 编码信息的轻量级协议。例如可以使用 SMTP（电子邮件协议）来传递 SOAP 消息。

SOAP 协议实现了 Web 服务的访问，但这种访问并不是直接的。Web 服务需要通过 SOAP 协议来封装，在封装后实现调用。

SOAP 规范定义了 SOAP 消息的格式，以及怎样通过 HTTP 协议来使用 SOAP。SOAP 也是基于 XML 和 XSD 的，XML 是 SOAP 的数据编码方式。因此 SOAP 协议同样是独立于任何编程语言、对象模型、操作系统和平台的协议。

3．WSDL 语言

WSDL（Web Service Description Language）即 Web 服务描述语言。当开发人员定义一个含有参数的方法时，需要有文档或注释对该方法的功能和参数做一个描述，以方便该方法的调用和维护。Web Service 是一种定义好供使用的程序，因此它同样需要有一种方式对其功能进行描述，而 WSDL 语言即对 Web 服务进行描述的语言。

WSDL 语言使用机器能够识别的方式提供一个正式描述文档，WSDL 语言同样基于 XML 语言。WSDL 用于描述 Web 服务及其函数、参数和返回值，由于它是基于 XML 的，所以既是机器可阅读的，也是人可阅读的。

一些最新的开发工具既能根据 Web Service 生成 WSDL 文档，又能导入 WSDL 文档，生成调用相应 Web Service 的代码。

定义 WSDL 语言来规范对 Web Service 的描述，其作用有以下几点。

- 开发工具可以自动处理通讯细节。

- 分布式系统的文档，通过 WSDL 语言的跨平台性使系统在交互中能够顺利进行交流；系统的复杂度越高，WSDL 语言的描述越重要。
- 利于标准化。

WSDL 语言作为对 Web Service 的描述，有标准的描述规范，如表 5-1 列举了 WSDL 语言在描述时的规范，表 5-2 列举了调用时的规范。

表 5-1 WSDL 语言描述规范

标签	说明
types	描述数据类型
message	定义传入传出的消息格式
portType	定义了一个入口的类型（使用了怎样的 request/response 消息对）：单请求、单响应、请求/响应、响应/请求
binding	确定 portType 将会使用何种传输协议（SOAP/HTTP-POST/…）
port	定义了一个关联某个 binding 的服务入口
service	一组 port 组成的 Web Service

表 5-2 WSDL 语言调用

标签	说明
Service	可以通过一个或多个 ports 调用
Port	获取服务的方法，绑定到 portType
Binding	如何获取 portType，每个 portType 可以有多个绑定，如 HTTP,JMS,SMTP 等
portType	服务的抽象定义，即接口的思想

4．UDDI

UDDI（Universal Description,Discovery and Integration）即通用描述、发现与集成服务，它是一套基于 Web 的、分布式的、为 Web 服务提供的信息注册中心实现标准规范。

UDDI 是一种集成服务，目的是为电子商务建立标准。UDDI 也包含一组使企业能够将自身提供的 Web Service 注册，以使别的企业能够发现访问协议的实现标准。

UDDI 服务最重要的是实现商业注册，任何企业都可以到注册中心去免费注册企业的信息和提供服务。

Web Service 相关技术间的相互作用如图 5-1 所示。

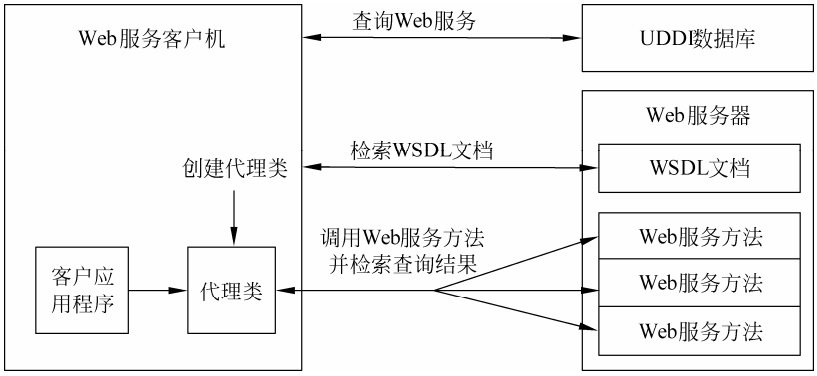

图 5-1 Web 服务的交互过程

5.1.2 WSDL 语言基础

WSDL 语言用于描述 Web 服务，它使用机器能阅读的方式提供一个正式的描述文档。Web Service 描述语言（WSDL）是基于 XML 的语言，描述 Web Service 及其函数、参数和返回值。

因为是基于 XML 的，所以 WSDL 既是机器可阅读的，又是人可阅读的。一些最新的开发工具既能根据 Web Service 生成 WSDL 文档，又能导入 WSDL 文档，生成调用相应 Web Service 的代码。

WSDL 将 Web 服务定义为服务访问点或端口的集合。客户端可以通过这些服务访问点对包含面向文档信息或面向过程调用的服务进行访问（类似远程过程调用）。

在 WSDL 中，由于服务访问点和消息的抽象定义，已经从具体的服务部署或数据格式绑定中分离，因此可以对抽象定义进行再次使用。

WSDL 首先对访问的操作和访问时使用的请求/响应消息进行抽象描述，然后将其绑定到具体的传输协议和消息格式上，以最终定义具体部署的服务访问点。相关的具体部署的服务访问点通过组合就成为抽象的 Web 服务。WSDL 有以下几个特点。

- WSDL 用于描述网络服务，是网络服务描述语言。
- WSDL 使用 XML 编写。
- WSDL 是一种 XML 文档。
- WSDL 也可用于定位网络服务，以及网络服务提供的操作（或方法）。
- WSDL 还不是 W3C 标准。

例如，在 ASP.NET 中，要向 Web 服务类添加有关的附加信息，则需要使用 WebService 属性来实现。该属性由 System.Web.Service.WebServiceAttribute 类实现，包含的内容有 NameSpace、Name 和 Description 等。

Description 是 WSDL 文档的一部分，它用于向 Web 服务添加描述信息。描述性消息将在 XMLWebService 的说明文件生成后显示给 XMLWebService 的潜在用户。

如向 Web 服务添加描述“返回 Hello World”，使用代码如下的：

```
[WebService(Namespace = "http://www.iWebServices.com", Name = "WebSer",
Description = "返回 Hello World")]
public class WebService1 : System.Web.Services.WebService
```

运行 Web 服务，效果如图 5-2 所示。图中首行 WebSer 字样下方即为对该 Web 服务的描述。

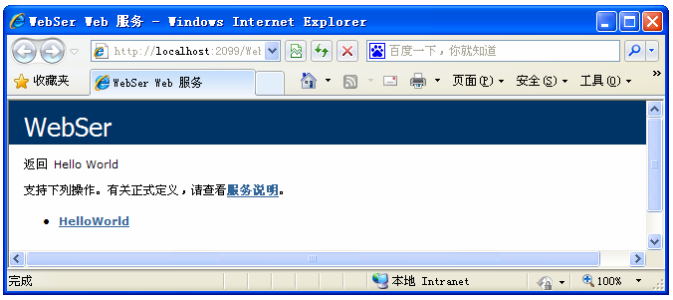

图 5-2 添加描述

5.1.3 WSDL 文档编写

本节将介绍如何编写 WSDL 文件，根据 WSDL 文件编写服务器端和客户端代码，并发布 Web Service 服务的过程。

WSDL 现在有两个版本，分别为 WSDL 1.1 和 WSDL 2.0。W3C 的官方文档地址如下所示。

- http://www.w3.org/TR/wsdlWeb Services Descrīption Language (WSDL) 1.1W3C Note 15 March 2001。
- http://www.w3.org/TR/2007/WD-wsdl20-primer-20070326/Web Services Descrīption Language (WSDL) Version 2.0 Part 0: PrimerW3C Working Draft 26 March 2007。

对于 WSDL 规范，可以参考以上两个官方文档。与 XSD 一样，WSDL 也是一个 XML 标准。而且 WSDL 是一个非常精确和复杂的标准，它详细描述了 Web 服务的方法和操作。

一个 WSDL 文档必须以 definitions 作为根元素，其中可以包含：types、message、portType、binding 和 service 元素。如下所示是一个 WSDL 文档基本结构（省略各元素的详细定义）：


```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime=http://schemas.xmlsoap.org/wsdl/mime/
xmlns:tns=http://tempuri.org/
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm=http://microsoft.com/wsdl/mime/textMatching/
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12=http://schemas.xmlsoap.org/wsdl/soap12/
targetNamespace="http://tempuri.org/" xmlns:wsdl="http://schemas.xmlsoap.
org/wsdl/">
  <wsdl:types>
    <s:schema > </s:schema>
  </wsdl:types>
  <wsdl:message >
    <wsdl:part />
  </wsdl:message>
  <wsdl:portType name="MathsSoap">
    <wsdl:operation name="numadd">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
      </wsdl:documentation>
      <wsdl:input name="addint" message="tns:addintSoapIn" />
      <wsdl:output name="addint" message="tns:addintSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
```

```
<wsdl:binding name="MathsSoap" type="tns:MathsSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="numadd">
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="Maths">
  <wsdl:port name="MathsSoap" binding="tns:MathsSoap">
    <soap:address location="http://localhost:2099/Maths.asmx" />
  </wsdl:port>
</wsdl:service>
```

以下内容为 5 个元素的简要说明。

- **types** 这部分定义了 Web 服务使用的所有数据类型集合，可以被元素的各消息组件所引用。通常使用 XML Schema 作为类型系统来描述。
- **message** 这部分定义了消息和消息传输的抽象类型化定义。这里将使用 types 所定义的类型来定义整个消息的数据结构。
- **portType** 这部分是一个抽象操作的列表，还定义了所有操作和返回的逻辑消息。
- **binding** 这部分为每个具体的端口类型都定义了消息格式和协议。
- **service** 这部分定义了 Web 服务的 URL 地址信息。

第一行声明该文档是 XML。尽管这并不是必需的，但它有助于 XML 解析器决定是否解析 WSDL 文件或只是报错。第二行是 WSDL 文档的根元素 definitions。

types 元素包含了 Web 服务中使用的所有数据类型集合。如果没有需要声明的数据类型，这一项可以默认。

message 元素定义消息和消息传输的抽象类型化。如果我们把操作看作函数，message 元素定义了函数的参数。message 元素中的每个 part 子元素都和某个参数相符。

输入参数在 message 元素中定义，与输出参数相隔离，输出参数有自己的 message 元素。

输出 message 元素以“Response”结尾，每个 part 元素都有名字和类型属性，就像函数的参数有参数名和参数类型。

part 元素的类型可以是 XSD 基类型，也可以是 SOAP 定义类型（soapenc）、WSDL 定义类型（wsdl）或是 Types 定义的类型。

用于交换文档时，WSDL 允许使用 message 元素来描述交换的文档。

一个 PortTypes 抽象操作列表中，可以有零个、单个或多个 portType 元素。由于抽象 PortType 定义可以放置在分开的文件中，在某个 WSDL 文件中没有 portType 元素是可能的。

一个 portType 元素可在 operation 元素中定义一个或是多个操作。operation 元素可以有一个、两个、三个子元素：input，output 和 fault 元素。

每个 input 和 output 元素中的消息都引用 Message 中相关的 message 元素。

Bindings 可以有零个、一个或者多个 binding 元素。它的意图是制定每个 operation 通过网络调用和回应。

Services 同样可以有零个、一个、多个 service 元素。它还包含了 port 元素，每个 port 元素引用一个 Bindings 里的 binding 元素。Bindings 和 Services 元素都包含 WSDL 文档

以上 5 个元素构成了 Web 服务的完整定义，并且他们的关系非常紧密。其中，types、message 和 portType 元素共同组成了 WSDL 的抽象部分，称为服务接口定义（Service Interface Definition）。这些元素描述了 Web 服务的基本情况，他们独立于平台和具体的程序语言，也没有提供任何关于或者在何处查找 Web 服务实例的信息。

binding 和 service 元素组成了 Web 服务定义的具体部分，称为服务实现定义（Service Implementation Definition）。他们指定了访问特定 Web 服务实例所需的信息。例如，传输协议、Web 服务的地址等。

WSDL 文档的服务接口定义和服务实现定义两部分通过 binding 元素链接起来，如图 5-3 所示了这些元素间的关系。

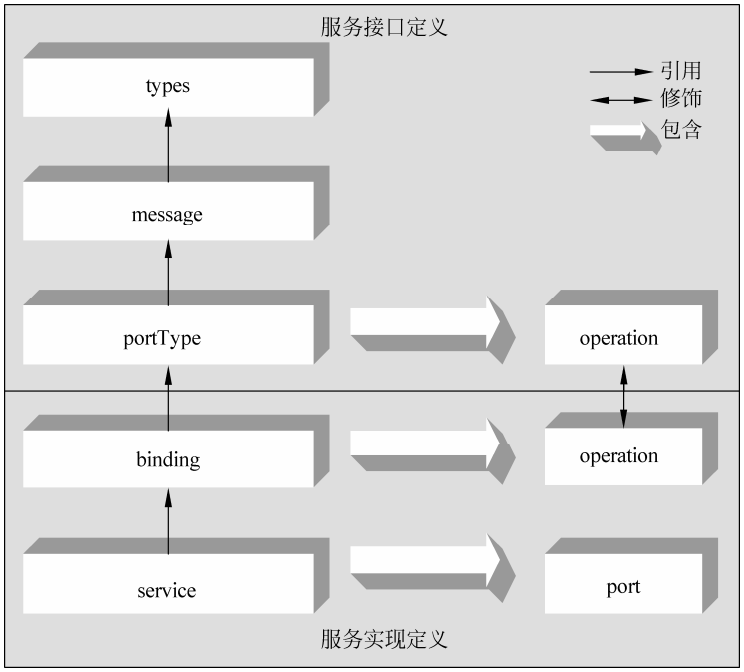

图 5–3 WSDL 文档结构

为了更好地说明 WSDL 文档中各个元素之间的关系，在图 5-3 中使用了三种连接符号。其中单向细线箭头表示了“引用”和“被引用”关系，双向细线箭头表示“修饰”关系，单向立体箭头表示了“包含”关系。

在图 5-3 中，message 元素使用 types 元素的定义，portType 元素使用 message 元素的定义，binding 元素引用 portType 元素，service 元素又引用了 binding 元素。由此可见，这五个元素是简单的引用关系。另外，portType 和 binding 元素都包含有 operation 元素，而且 binding 元素中的 operation 元素是对 portType 元素下 operation 元素的进一步描述。service 元素又包含 port 元素。

一个 WSDL 文档中可以没有 type 元素，如果有只能出现一次。其他四个元素可以

出现多次，也可以没有。最后，在 WSDL 文档中各个元素出现的顺序是有规定的，分别是：type、message、portType、binding 和 service。

在 WSDL 文档中还有两个不常用的元素，如下所示。

- documentation 元素用于提供一个可阅读的文档，它可以包含在任何其他元素中。
- import 元素用于导入其他 WSDL 文档或者 XSD，从而实现 WSDL 文档的模块化。

5.2 文档结构

在上述 WSDL 文档的编写中可以看出，WSDL 文档由多种元素构成。本节将详细介绍 WSDL 文档的构成元素。

5.2.1 definitions 根元素

在 WSDL 文档中 definitions 元素是根元素，也就是说该元素有且只能有一次。在 definitions 元素中定义了 WSDL 文档中用到的命名空间，像 XSD 命名空间和 SOAP 命名空间等。

如下所示为一个典型的 definitions 根元素的内容。

```
<wsdl:definitions
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:mime=http://schemas.xmlsoap.org/wsdl/mime/
    xmlns:tns=http://tempuri.org/
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:soap12=http://schemas.xmlsoap.org/wsdl/soap12/
    targetNamespace="http://tempuri.org/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
</wsdl:definitions>
```

命名空间对于如何区分 WSDL 文档元素非常重要，它通过 URL 惟一地标识了元素，避免元素的命名冲突。例如，“xmlns:s”就代表命名空间 http://www.w3.org/2001/XMLSchema，当使用这个命名空间时，只需要把“s:”作为前缀即可，例如“s:string”。

如表 5-3 中列出了在 WSDL 文档中经常出现的前缀、命名空间及其说明。

表 5-3 WSDL 常用命名空间

前缀	命名空间	说明
s:	http://www.w3.org/2001/XMLSchema	XSD 的命名空间，它是 WSDL 文档中默认的类型系统
soap:	http://schemas.xmlsoap.org/wsdl/soap	用于 SOAP 绑定的命名空间
tm:	http://microsoft.com/wsdl/mime/textMatching	由 Microsoft 支持的文本匹配所在的命名空间
soapenc:	http://schemas.xmlsoap.org/soap/encoding	SOAP 1.1 定义的 Encoding 命名空间

续表

前缀	命名空间	说明
mime:	http://schemas.xmlsoap.org/wsdl/mime	用于 WSDL MIME 绑定的命名空间
tns:	http://tempuri.org	当前 Web 服务所使用的命名空间
http:	http://schemas.xmlsoap.org/wsdl/http	用于 WSDL 中 GET 和 POST 绑定的命名空间
wsdl:	http://schemas.xmlsoap.org/wsdl	WSDL 文档的默认命名空间

definitions 元素还有一个 targetNameSpace 属性，它用于指定该元素内部声明元素所属的命名空间。例如，在前面的 WSDL 文档中 targetNameSpace 属性值为“http://tempuri.org”，就意味着该 WSDL 文档中所有声明的元素都属于 http://tempuri.org 命名空间。如果 WSDL 文档外部需要引用这些元素，就必须指明命名空间 http://tempuri.org。

5.2.2 types 元素

types 元素主要用于声明 Web 服务中用到的元素和类型，它实际上是一个嵌入的 XSD。在这个元素内所有的 XSD 数据类型都有效，而且允许用户通过扩展添加需要的其他类型系统。

在本书的第 4 章中已经介绍过 XSD 类型系统，这里从 WSDL 文档应用的角度介绍 types 元素使用到的 XSD。

假设，我们用 C#定义一个 Person 结构，代码如下所示：

```
public struct Person
{
    public string Name;
    public DateTime Birthday;
    public bool isMarry;
    public decimal Height;
    public decimal Weight;
}
```

它在 types 元素中的 XSD 定义代码如下所示：

```
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://
    tempuri.org/">
      <s:element name="Person">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="Name" type=
            "xs:string" />
            <s:element minOccurs="0" maxOccurs="1" name="Birthday" type=
            "xs:date" />
            <s:element minOccurs="0" maxOccurs="1" name="isMarry" type=
            "xs:boolean" />
            <s:element minOccurs="0" maxOccurs="1" name="Height" type=
```

```
                "xs:decimal" />
                <s:element minOccurs="0" maxOccurs="1" name="Weight" type=
                "xs:decimal" />
              </s:sequence>
            </s:complexType>
          </s:element>
        </s:schema>
      </wsdl:types>
```

可以看到在这里使用 complexType 元素定义了一个复杂类型来实现。在 types 元素中复杂类型也是使用最多的，这是因为在 SOAP 协议中总是以复杂类型来“理解”参数。

types 元素的另一个作用是为 message 元素定义参数，而且输入参数与方法的名称相同，输出参数则是在方法的名称后添加“Response”后缀。

下面再看一个例子，假设有如下代码所示的 Web 服务方法：

```
public User check(int id,string name, string pass)
{
   //此处省略方法的实现
}
```

可以看到，定义了一个名为 check 的方法，该方法有三个参数：整型 id 参数、字符串类型 name 参数和字符串类型 pass 参数。check 方法的返回值是一个 User 类型。

根据前面内容中对 types 元素的介绍，在最终的 WSDL 文档中应该有一个名为 check 的元素来定义方法所需的三个参数，另一个名为 checkResponse 的元素来定义方法的返回值。而且这些元素都应该定义为复杂类型，最终代码如下所示：

```
    <wsdl:types>
       <s:schema elementFormDefault="qualified" targetNamespace="http:
       //tempuri.org/">
          <s:element name="check">
             <s:complexType>
                <s:sequence>
                   <s:element minOccurs="1" maxOccurs="1" name="id" type=
                   "s:int" />
                   <s:element minOccurs="0" maxOccurs="1" name="name" type=
                   "s:string" />
                   <s:element minOccurs="0" maxOccurs="1" name="pass" type=
                   "s:string" />
                </s:sequence>
             </s:complexType>
          </s:element>
          <s:element name="checkResponse">
            <s:complexType>
              <s:sequence>
                <s:element minOccurs="0" maxOccurs="1" name="checkResult"
                type="tns:User" />
              </s:sequence>
```

```
            </s:complexType>
          </s:element>
          <s:complexType name="User">
              <s:sequence>
                  <s:element minOccurs="1" maxOccurs="1" name="id" type=
                  "s:int" />
                  <s:element minOccurs="0" maxOccurs="1" name="name" type=
                  "s:string" />
                  <s:element minOccurs="0" maxOccurs="1" name="pass" type=
                   "s:string" />
                  <s:element minOccurs="1" maxOccurs="1" name="lasttime" type=
                  "s:dateTime" />
              </s:sequence>
          </s:complexType>
      </s:schema>
  </wsdl:types>
```

注 意

由于 XSD 存在很多个版本，而 WSDL 规范推荐使用 2001 版本。因此为了使用它，需要指定命名空间为“http://www.w3.org/2001/XMLSchema”。

5.2.3 message 元素

message 元素定义了 Web 服务中包含的方法及参数。该元素主要有两种用法：一种是在 type 元素中定义方法和参数，然后在 message 元素中引用；另一种则是直接在 message 元素中定义方法和参数。

在 message 元素中可以包含如下三个部分。

1．name 属性

如果该属性的是由“方法名+SoapIn”组成的，说明该方法用于调用消息；而如果是由“方法名+SoapOut”组成的，则说明该方法是用于处理结果消息的。另外，如果用户正在使用 HTTP 的 GET 方式，还会在方法名后追加“HttpGetIn”或者“HttpGetOut”字符；而对于 HTTP 的 POST 方法，则会追加“HttpPostIn”或者“HttpPostOut”字符。

2．part 元素的 name 属性

对 part 元素设置 name 属性是因为这些消息的参数是在以上 types 元素中定义的。

3．part 元素的 element 属性

该属性通过定义在每一个输入和输出消息中的名称来引用复杂类型。

如对于求数据乘方的方法 numpow(int num, int pow)，该方法返回 num 的 pow 次方（整型），定义其 types 元素和 message 元素如下所示：

```
<wsdl:types>
  <s:schema elementFormDefault="qualified" targetNamespace="http://
  tempuri.org/">
    <s:element name="power">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="num" type="s:int" />
          <s:element minOccurs="1" maxOccurs="1" name="pow" type="s:int" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="powerResponse">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="powerResult" type=
          "s:int" />
        </s:sequence>
      </s:complexType>
    </s:element>
  </s:schema>
</wsdl:types>
<wsdl:message name="powerSoapIn">
  <wsdl:part name="parameters" element="tns:power" />
</wsdl:message>
<wsdl:message name="powerSoapOut">
  <wsdl:part name="parameters" element="tns:powerResponse" />
</wsdl:message>
```

上述代码使用的是在 type 元素中定义的方法和参数，在 message 元素中引用的方式。

5.2.4 portType 元素

portType 元素是 WSDL 文档中最重要的 WSDL 元素之一，它描述一个 Web 服务可被执行的操作，以及相关的消息。

一个 portType 表示一组操作（operation），而这组操作元素定义该 portType 中所有方法的语法。每个 operation 元素包括方法的名称、参数（message 元素）和参数数据类型（message 下的 part 元素）。

在一个 WSDL 文档中可以包含多个 portType 元素，而每个 portType 元素都包含一组相关的操作。多个 portType 元素之间使用 name 属性来进行区分，所以他们的值必须是惟一的。同样，operation 元素的 name 属性也必须是惟一的，它区分同一个 portType 下的多个操作。

operation 元素使用 input、output 和 fault 来描述一个操作。他们不是必须的，但如果有，也只能出现一次。各个元素的含义如下：

- **input 元素** 用于描述操作的输入，包括输入的名称和类型。

- **output 元素**　用于描述操作的输出，包括输出的名称和类型。
- **fault 元素**　当操作出现错误时，可以用这个元素指定格式报告。

例如，对于 numpow(int num, int pow)方法，该方法返回 num 的 pow 次方（整型），其 operation 元素内应该同时具有 input 元素和 output 元素，代码如下所示：

```
<wsdl:portType name="MathsSoap">
  <wsdl:operation name="numpow">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    整型数字乘方,参数 int num, int pow,返回 num 的 pow 次方</wsdl:documentation>
    <wsdl:input name="power" message="tns:powerSoapIn" />
    <wsdl:output name="power" message="tns:powerSoapOut" />
  </wsdl:operation>
</wsdl:portType>
```

以上代码中的 input 元素和 output 元素都只有 message 属性，而没有 name 属性。这是因为它们的 name 属性主要是用于区分名称相同、但输入和输出不同的操作。但是在这里的 portType 元素中，只有一个操作，所以没有必要在 input 和 output 元素中使用 name 属性。

上述 numpow()方法没有重载，较为简单。对于有着重载的方法，需要分别声明 message 元素和 operation 元素。如现有两个重载的 numadd()方法如下所示：


```
public int numadd(int a,int b)
public string numadd(string a, string b)
```

上述代码中的两个方法分别返回两个整型数据的和及两个字符串的连接，分别有着 int 类型和 string 类型的参数，定义其 WSDL 如下所示：

```
<wsdl:message name="addintSoapIn">
  <wsdl:part name="parameters" element="tns:addint" />
</wsdl:message>
<wsdl:message name="addintSoapOut">
  <wsdl:part name="parameters" element="tns:addintResponse" />
</wsdl:message>
<wsdl:message name="addstringSoapIn">
  <wsdl:part name="parameters" element="tns:addstring" />
</wsdl:message>
<wsdl:message name="addstringSoapOut">
  <wsdl:part name="parameters" element="tns:addstringResponse" />
</wsdl:message>

<wsdl:portType name="MathsSoap">
  <wsdl:operation name="numadd">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    两个整型数的和</wsdl:documentation>
    <wsdl:input name="addint" message="tns:addintSoapIn" />
    <wsdl:output name="addint" message="tns:addintSoapOut" />
  </wsdl:operation>
```

```
    <wsdl:operation name="numadd">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
      两个字符串合并</wsdl:documentation>
      <wsdl:input name="addstring" message="tns:addstringSoapIn" />
      <wsdl:output name="addstring" message="tns:addstringSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
```

上述代码中，portType 元素中包含两个 operation 元素，但它们的 name 属性相同，都是 numadd。为了区分，在 input 元素和 output 元素中使用 name 属性把它们分别命名为 addint 和 addstring。

portType 元素在 binding 元素中引用，把一组操作绑定到协议。而这些操作又映射到 binding 元素的 SOAP 操作进行调用。实际上，portType 依附于协议，将操作映射到 SOAP 方法，这样就可以通过 WSDL 文档的服务接口定义部分与服务实现定义部分进行连接。

5.2.5 binding 元素

binding 元素用于指定操作使用的传输协议、序列化方式以及编码格式。在 WSDL 文档中的前三个部分（types、message 和 portType）都是从抽象层面上描述 Web 服务，而 binding 元素将着重描述数据传输的物理细节，也就是说 binding 元素是对前三个部分所做的具体化描述。

在同一个 WSDL 文档中抽象定义和具体说明不仅是分开的，而且 WSDL 还允许把抽象定义放到一个单独的文件中，然后使用 import 元素导入到最终的 WSDL 文档中。这在 Web 服务的实际开发中是非常有用的。当不同的开发商开发同一种类型的商业应用程序时，它们可以定义一套标准的操作，并把它分发给各个开发商，而各个开发商实现各自不同的绑定。在发布 Web 服务时，它们可以把自己的绑定描述与抽象定义综合起来，生成自己的 WSDL 文档。

例如，可以为学校系统定制一套标准操作，这实际上是一个 WSDL 抽象文档，而每个学校可以定制自己的传输协议、序列化方式和编码格式。

为了更好地说明 binding 元素的内容，绑定以 5.2.4 小节中的 numadd()方法为例。定义 binding 代码如下所示：

```
  <wsdl:binding name="MathsSoap" type="tns:MathsSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="numadd">
      <soap:operation soapAction="http://tempuri.org/addint" style=
      "document"/>
      <wsdl:input name="addint">
        <soap:body use="literal" />
      </wsdl:input>
```

```
      <wsdl:output name="addint">
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="numadd">
      <soap:operation soapAction="http://tempuri.org/addstring" style=
      "document"/>
      <wsdl:input name="addstring">
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output name="addstring">
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
```

如上述代码所示，这里 binding 元素中的 name 属性指定绑定的名称为 MathsSoap。如果有多个 binding 元素，name 属性必须是惟一的。binding 元素的 type 属性引用一个 portType 元素，在这里为“tns:MathsSoap”。

接下来的“<soap:binding>”元素指定 SOAP 消息的网络协议，这里是 HTTP。另外，由于描述的 Web 服务带有重载的 numadd()方法，所以在绑定部分必须有两个 operation 元素，分别指定他们的传输协议和消息格式。区分他们的标识是 input 元素中的 name 属性，分别为 addint 和 addstring。

两个 operation 元素中都有一个 soapAction 属性，它的值是一个 URL。当使用 HTTP 传递 SOAP 消息时就会使用 soapAction HTTP 头表明该 SOAP 请求的时机，服务器可以根据这个 HTTP 头信息传递 SOAP 消息。operation 的 soapAction 属性就是指明调用这个操作时 SoapAction 头应该设置的值。

```
public int numadd(int a,int b)
public string numadd(string a, string b)
```

这里的两个 operation 元素中 soapAction 的值不同，如果要调用“public int numadd(int a,int b)”则需要使用下面的 HTTP 头：

```
<soap:operation soapAction="http://tempuri.org/addint" style="document" />
```

而如果要调用“public string numadd(string a, string b)”则使用的 HTTP 应该为：

```
<soap:operation soapAction="http://tempuri.org/addstring" style="document"/>
```

- binding 中的 operation 元素和 portType 中的 operation 元素类似，可以包含 input、output 和 fault 元素。但这三种元素不指定操作的消息，而是指定消息的格式。
- 在这个例子中，operation 元素中即有 input 元素（输入）又有 output 元素（输出），他们通过各自的 name 属性区分是哪个操作。

- 由于使用的是 SOAP 绑定，所以在 input 元素中使用 “<soap:body>” 元素来指定 SOAP 消息中 body 元素内容的格式。body 元素主要有三个属性，如下所示。
  - **use** 此属性用来指定数据是 encoded 还是 literal。literal 表示 SOAP 消息中包含的数据只是简单的按照抽象定义部分的要求进行格式化；而 encoded 则表示使用 encodingStyle 属性决定数据的编码格式。
  - **namespace** 每个 SOAP 消息的 body 部分都可以有自己的命名空间。
  - **encodingStyle** 这是一个 URL 值，用于指定 SOAP 消息的编码规则。如果使用 SOAP 协议中定义的编码规则，则应该是 http://schemas.xmlsoap.org/soap/encoding。

5.2.6 service 元素

一个 service 元素可以由多个 port 元素组成，每个 port 元素都为一个 binding 元素指定一个访问地址。如果多个 port 元素同时为一个 binding 元素指定不同的地址，那这些地址中的任意一个都可以使用。

在一个 WSDL 文档中通常只使用一个 service 元素。当然也可以有多个 service 元素。例如，可以根据传输协议把所有的 HTTP POST 放到一个 service 元素中；而所有的 SOAP 放到另一个中，客户也可以使用自己支持的协议来搜索 service。

下面给出一个名为 WebService 的 service 元素的内容。该元素中包含一个名为 WebServiceSoap 的 port 元素，相应的绑定为 WebServiceSoap，SOAP 绑定的地址为 http://localhost/WebSite/WebService.asmx，如下所示：

```
<wsdl:service name="WebService">
  <wsdl:port name="WebServiceSoap" binding="tns:WebServiceSoap">
    <soap:address location="http://localhost/WebSite/WebService.asmx" />
  </wsdl:port>
</wsdl:service>
```

5.3 WSDL 技术

在了解到 WSDL 文档结构和编写方法之后，需要知道 WSDL 的相关技术，包括 WSDL 文档的使用原理、使用自定义的 WSDL 来替换 Web 服务自动生成的 WSDL，以及使用 WSDL 文档调用 Web 服务等。

5.3.1 WSDL 端口

端口描述是由 Web Service 提供的界面（合法操作），介绍一个 Web Service 可被执行的操作，以及相关的消息，通过 message 和 portType 来共同描述。

端口定义了指向某个 Web Service 的连接点。可以把该元素比作传统编程语言中的一个函数库（或一个模块、或一个类），而把每个操作比作传统编程语言中的一个函数。WSDL 定义了四种类型操作类型，如表 5-4 所示。

表 5-4 操作类型

类型	定义
One-way	此操作可接受消息，但不会返回响应
Request-response	此操作可接受一个请求并会返回一个响应
Solicit-response	此操作可发送一个请求，并会等待一个响应
Notification	此操作可发送一条消息，但不会等待响应

如下代码中，端口“glossaryTerms”定义了一个名为 setTerm 的 one-way 操作，代码如下所示：

```
<message name="newTermValues">
   <part name="term" type="xs:string"/>
   <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
   <operation name="setTerm">
      <input name="newTerm" message="newTermValues"/>
   </operation>
</portType >
```

这个 setTerm 操作可接受新术语表项目消息的输入，这些消息使用一条名为 newTermValues 的消息，此消息带有输入参数 term 和 value。不过，没有为这个操作定义任何输出。

以下代码中，端口 glossaryTerms 定义了一个名为 getTerm 的 request-response 操作，代码如下所示：

```
<message name="getTermRequest">
   <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
   <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

getTerm 操作会请求一个名为 getTermRequest 的输入消息，此消息带有一个名为 term 的参数，并将返回一个名为 getTermResponse 的输出消息，此消息带有一个名为 value 的参数。

5.3.2 使用自定义 WSDL

用 Visual Studio 创建 Web Service 会自动生成 WSDL。若要使用指定的 WSDL，则需要将 WSDL 文件放到 Web 服务文件中所在的目录，并使用一系列的属性来定义，如下所示。

如 Web Service 名字是 MyWebService，则需要依次执行如下步骤。

（1）将自定义 WSDL 文件放到 MyWebService.asmx 文件所在的目录，并在 MyWebService 类上加上[WebServiceBinding]属性标签，如下所示：

```
[WebServiceBinding(Name = "MyBinding", Location = "MyCustom.wsdl")]
public class MyWebService : WebService
{
 // ...
}
```


MyBinding 是 WSDL 文件中定义的 binding 的名字，MyCustom.wsdl 是自定义的 WSDL 文件。

（2）在 WebService 方法上加上[SoapDocumentMethod]属性标签，如下所示：

```
[WebMethod]
[SoapDocumentMethod(Action="http://tempuri.org/HelloWorld",   Binding=
"MyBinding")]
public string HelloWorld() {
 // …
}
```

以上代码中的 Action 是 WSDL 文件中定义的 operation 元素的 soapAction 地址，Binding 参考第（1）步里的定义。这样就可以使用自定义的 WSDL 了。

5.3.3 WSDL 文档使用原理

我们在取得 Web 服务的 WSDL 文档之后，通常有两种使用方式，这两种方式都是程序级的利用，开发人员没有必要阅读或者理解它。

第一种方式发生在开发时，开发人员根据 WSDL 文档生成调用 Web 服务的客户端代码。使用 wsdl 命令生成的客户端代码，通常也被称为 Web 服务代理类。然后，在编写客户端应用程序的过程中，我们就直接使用 Web 服务代理类，就像使用本地类一样，而实际的 Web 服务调用发生在代理类与 Web 服务之间。这种方式的执行过程如图 5-4 所示。

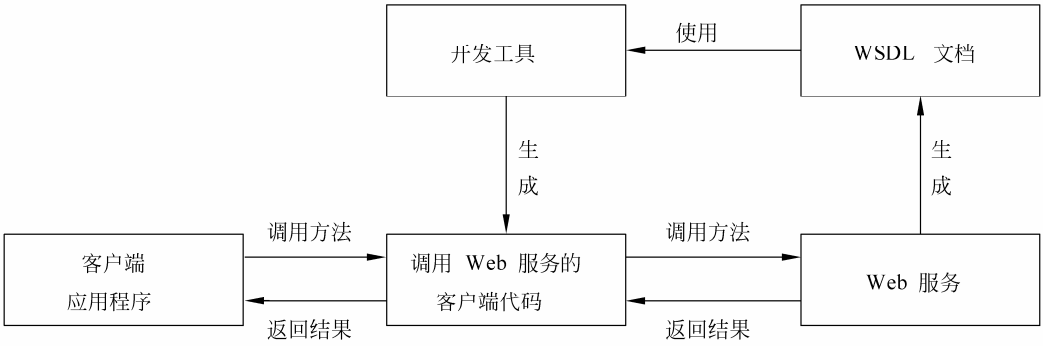

图 5-4 开发时使用 WSDL 文档过程

第二种方式是指在程序运行时使用 WSDL 文档。此时，客户端应用程序通过它所处的运行环境发出对 Web 服务的调用（例如，使用 SOAP Toolkit 的客户端组件），运行环境根据 WSDL 文档生成正确的 Web 服务调用请求，并接受 Web 服务返回的结果，然后把结果传递给客户端应用程序。整个过程如图 5-5 所示。

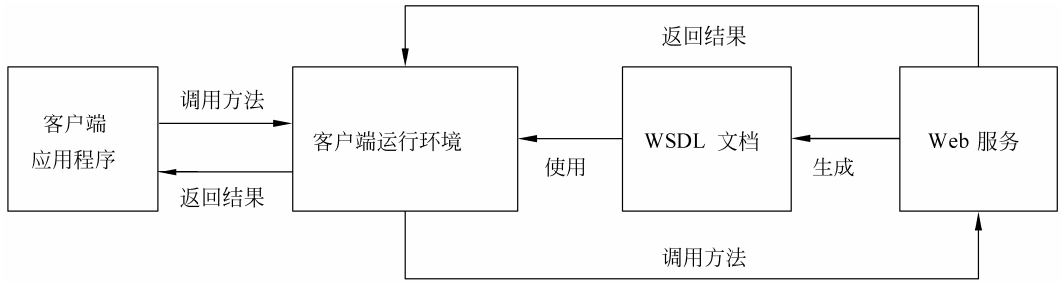

图 5-5 运行时使用 WSDL 文档过程

5.3.4 WSDL 调用 Web 服务

无论采用哪一种方式使用 WSDL 文档，首先有一点必须保证，就是能够通过一个 URL 来访问 Web 服务。从而才有可能获取或者生成 WSDL 文档，接下来则是在开发或者运行时调用 Web 服务。

在本节中创建一个 Web 服务，并生成一个 WSDL 文档，再通过 WSDL 文档来调用 Web 服务，使用 VS2010 和 C#语言，步骤如下。

（1）创建一个解决方案，添加一个 Web 应用程序。Web 服务的创建需要在项目名称上右击，选择【添加】|【新建项】选项打开如图 5-6 所示的窗口，选择【Web 服务】，为 Web 服务命名。

由图 5-6 可以看出，Web 服务的后缀名为.asmx，单击【添加】按钮完成创建，并打开刚创建的 Web 服务。

此时，在 Web 服务中有代码如下所示：

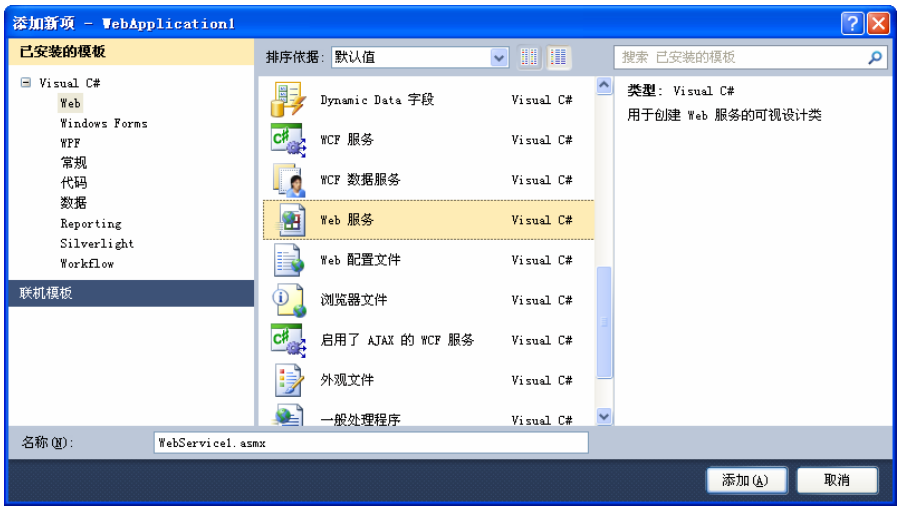

图 5-6 添加 Web 服务

```
namespace WebApplication1
{
    /// <summary>
    /// WebService1 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，请取消对下行的注释。
   // [System.Web.Script.Services.ScriptService]
    public class WebService1 : System.Web.Services.WebService
    {
        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }
    }
}
```

由上述代码可知，Web 服务默认的命名空间为项目名称，有类 WebService1 和一个方法 HelloWorld()。方法中的代码与 C#中的方法定义格式一样，但在 Web 服务中，每一个方法前面都要有[WebMethod]语句。上述代码已经是一个完整的 Web 服务了，开发人员可根据需要，在文件中定义所需要的方法。

（2）删除原有的方法，添加新的方法，获取当前时间，代码如下所示：

```
public class WebService1 : System.Web.Services.WebService
{
    [WebMethod(MessageName = "NowTime", Description = "当前时间")]
    public string NowTime()
```

```
        {
            DateTime nowtime = DateTime.Now;
            return nowtime.ToString();
        }
    }
```

（3）Web 服务修改之后，最好对该项目重新生成，以保存对 Web 服务的修改。打开新添加的项目并为该项目添加 Web 引用。在项目名称处右击，选择【添加 Web 引用】选项，打开如图 5-7 所示的对话框。

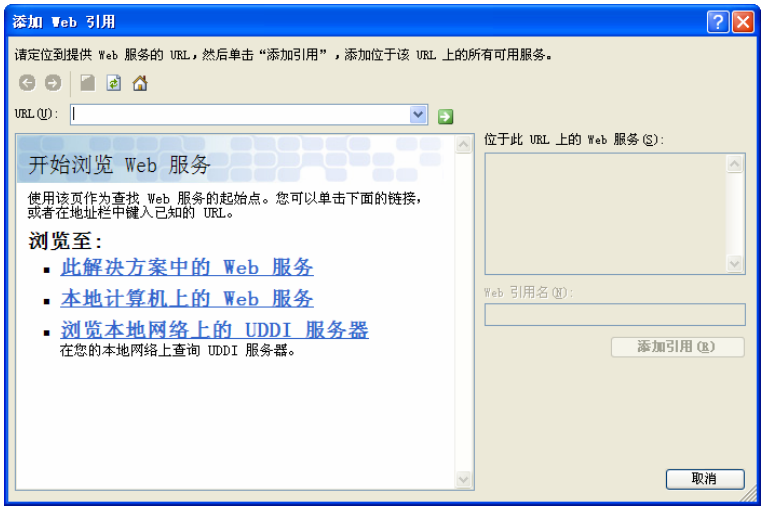

图 5-7　添加 Web 引用

（4）在图 5-7 所示的对话框中，有可以选择的 Web 服务地址，这里选择此解决方案中的 Web 服务，因此解决方案中的 Web 服务列表如图 5-8 所示。

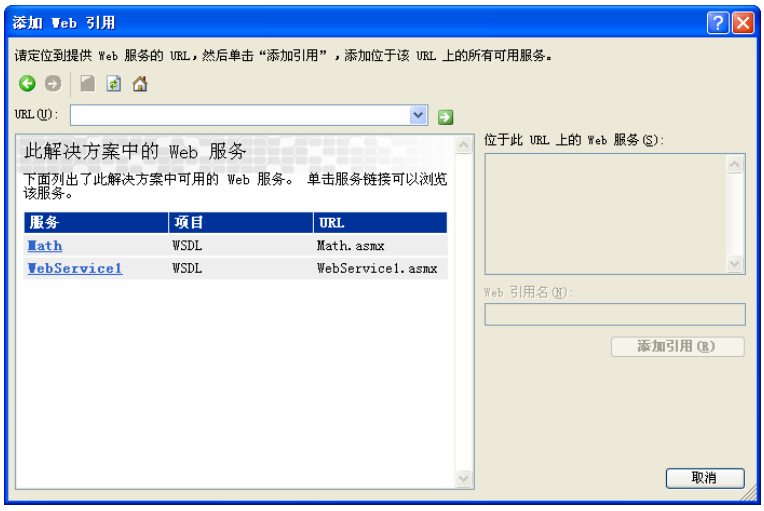

图 5-8　Web 服务列表

（5）选择步骤（1）中创建的 WebService1，则该窗体显示出对 WebService1 的描述，如图 5-9 所示。

图 5-9 WebService1 描述

在图 5-9 的右侧，需要为该 Web 服务定义一个命名空间，以便在程序中引用。之后单击【添加引用】按钮完成添加。

（6）对 Web 服务的引用添加之后，即可对该 Web 服务进行直接调用，如创建一个 Web 页面，添加一个标签控件 Label1，主要代码如下所示：

```
<form id="form1" runat="server">
<div>
    <h1 style="color:Red">当前时间：</h1>
    <h1>
        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
    </h1>
</div>
</form>
```

（7）为页面添加后台代码如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    NowTimes.WebService1 daytime = new NowTimes.WebService1();
    Label1.Text = daytime.NowTime();
}
```

（8）运行该页面，其效果如图 5-10 所示。

（9）Web 服务的顺利使用，编译器在该项目的目录下已经创建了 WSDL 文档，在浏览器中查看，其效果如图 5-11 所示。

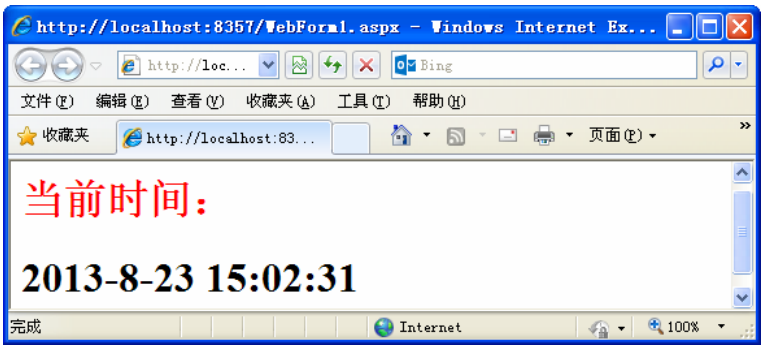

图 5-10 Web 服务使用效果

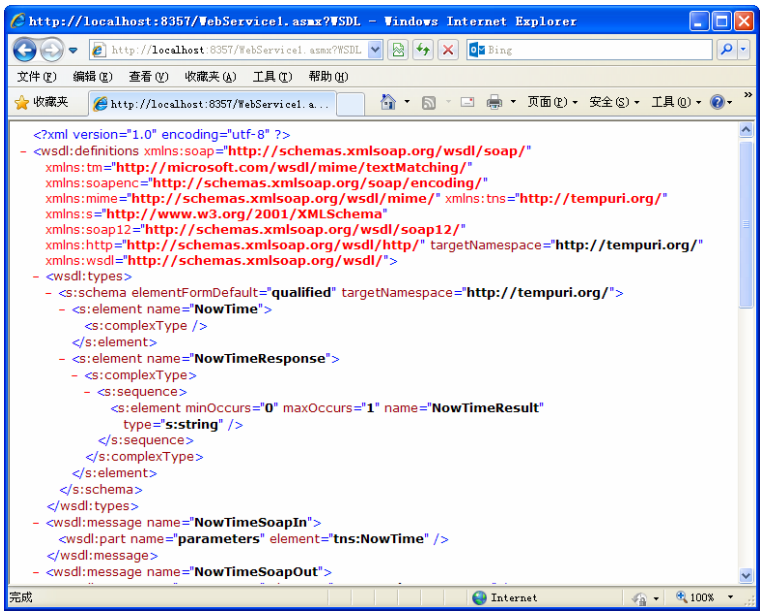

图 5-11 WSDL 文档

5.4 实验指导 5-1：两数相加 Web 服务

本章主要介绍的是 WSDL 与 Web 服务，现使用 C#语言定义 Web 服务，并针对该 Web 服务来定义 WSDL 文档，步骤如下。

（1）定义 Web 服务，添加两个方法，一个是无参数、返回一个字符串类型的方法 HelloWorld()，一个是有两个参数，返回两个参数和的方法 AddNum(int a,int b)，代码如下所示：

```
using System;
using System.Web;
using System.Web.Services;
namespace WSDL
```

```
{
    /// <summary>
    /// Math 的摘要说明
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    [System.Web.Script.Services.ScriptService]
    public class Math : System.Web.Services.WebService
    {
        [WebMethod(MessageName = "AddNumber", Description = "整型数字相加，
         参数 int a, int b，返回 a 与 b 的和")]
       public int AddNum(int a,int b)
       {
          return a+b;
       }
    }
}
```

（2）接下来定义 WSDL 文档，首先定义根元素 wsdl:definitions，代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://tempuri.org/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
targetNamespace="http://tempuri.org/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
```

（3）接下来定义 types 元素，声明 Web 服务中用到的元素和类型。该 Web 服务中有两个方法，其名称分别被定义为 HelloWorld 和 AddNumber。其中，HelloWorld 无参数有字符串类型返回值，AddNumber 有整型返回值和两个整型参数 a、b，定义代码如下所示：

```
<wsdl:types>
  <s:schema elementFormDefault="qualified" targetNamespace="http:
  //tempuri.org/">
    <s:element name="HelloWorld">
      <s:complexType />
    </s:element>
    <s:element name="HelloWorldResponse">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="HelloWorldResult"
          type="s:string" />
        </s:sequence>
```

```
      </s:complexType>
    </s:element>
    <s:element name="AddNumber">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="a" type="s:int" />
          <s:element minOccurs="1" maxOccurs="1" name="b" type="s:int" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="AddNumberResponse">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="AddNumberResult"
          type="s:int" />
        </s:sequence>
      </s:complexType>
    </s:element>
  </s:schema>
</wsdl:types>
```


（4）message 元素定义 Web 服务中包含的方法及参数，与 types 元素不同的是，types 元素重点定义 Web 服务中元素的类型，而 message 元素重点定义方法和参数的操作类型，如下所示：

```
<wsdl:message name="HelloWorldSoapIn">
  <wsdl:part name="parameters" element="tns:HelloWorld" />
</wsdl:message>
<wsdl:message name="HelloWorldSoapOut">
  <wsdl:part name="parameters" element="tns:HelloWorldResponse" />
</wsdl:message>
<wsdl:message name="AddNumberSoapIn">
  <wsdl:part name="parameters" element="tns:AddNumber" />
</wsdl:message>
<wsdl:message name="AddNumberSoapOut">
  <wsdl:part name="parameters" element="tns:AddNumberResponse" />
</wsdl:message>
```

（5）portType 元素描述 Web 服务可被执行的操作，以及相关的消息，即 Web 服务中，展示给用户的解释性内容，代码如下所示：

```
<wsdl:portType name="MathSoap">
  <wsdl:operation name="HelloWorld">
    <wsdl:input message="tns:HelloWorldSoapIn" />
    <wsdl:output message="tns:HelloWorldSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="AddNum">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    整型数字相加，参数 int a, int b，返回 a 与 b 的和</wsdl:documentation>
```

```
      <wsdl:input name="AddNumber" message="tns:AddNumberSoapIn" />
      <wsdl:output name="AddNumber" message="tns:AddNumberSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
```

（6）binding 元素用于指定操作使用的传输协议、序列化方式以及编码格式，两个方法都是简单方法，代码如下所示：

```
    <wsdl:binding name="MathSoap" type="tns:MathSoap">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
      <wsdl:operation name="HelloWorld">
        <soap:operation soapAction="http://tempuri.org/HelloWorld" style=
        "document" />
        <wsdl:input>
          <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
          <soap:body use="literal" />
        </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="AddNum">
        <soap:operation soapAction="http://tempuri.org/AddNumber" style=
        "document" />
        <wsdl:input name="AddNumber">
          <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output name="AddNumber">
          <soap:body use="literal" />
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
```

（7）service 元素定义 Web 服务的 URL 地址信息，该 Web 服务是本地服务，地址 http://localhost:8357/Math.asmx，代码如下所示：

```
    <wsdl:service name="Math">
      <wsdl:port name="MathSoap" binding="tns:MathSoap">
        <soap:address location="http://localhost:8357/Math.asmx" />
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

5.5 思考与练习

一、填空题

1．Web 服务描述语言是_____（Web Service Description Language）语言。

2．Web 服务描述语言的根元素为_____元素。

3．定义函数参数的元素是_____元素。

4．一个 WSDL 文档中可以没有_____

元素，但是如果有只能出现一次。

5．Web 服务具有普遍性，它使用 HTTP 和_____进行通信。

二、选择题

1．WSDL 是一个基于_____的文档，它描述了 Web 服务各个方面的元素。

A．XML　　B．XSD

C．SOAP　　D．HTTP

2．下面给出的元素中不属于 WSDL 文档的是_____。

A．types　　B．description

C．message　　D．binding

3．关于各个元素在 WSDL 文档出现的先后顺序，下面正确的是_____。

A．binding、message、portType、service、type

B．message、binding、type、portType、service

C．service、type、portType、binding、message

D．type、message、portType、binding、service

4．下面命名空间中不属于 definitions 元素的是_____。

A．http://www.w3.org/2001/XMLSchema

B．http://schemas.xmlsoap.org/wsdl/soap

C．http://microsoft.com/wsdl/mime/textMatching

D．http://www.wsdl.com/schema

5．以下不属于 operation 元素子元素的是_____。

A．input　　B．output

C．port　　D．fault

6．下列元素，如果有、则只能出现一次的是_____。

A．message　　B．binding

C．fault　　D．service

三、简答题

1．简述 Web 服务的作用及相关技术。

2．简单说明 WSDL 文档结构。

3．简单说明 WSDL 使用原理。

4．总结 WSDL 各个元素的用法

5．简单介绍如何通过 WSDL 调用 Web 服务。