第 5 章
了解 Web 应用程序

随着互联网的快速发展，网站建设已是信息化建设中最基本的一项工作，因此掌握构建可靠、动态、稳定的 Web 站点的技能是开发人员必备的能力。本章着重介绍 Web 应用程序开发中所运用到的基本知识如 HTML、CSS、Javascript 等，再详述了应用程序的部署以及 IIS 服务器。

5.1 了解网页开发

在现在的网站建设中，了解网页的相关知识是必不可少的，下面先从 XML 了解开始，然后再依次了解网页开发中用到的相关语言及其基础知识。

5.1.1 XML

5.1.1.1 什么时候使用 XML

什么时候在应用程序中使用 XML？本节我们就来看看它的主要使用场景。

- 用户需要处理已经保存在 XML 中的数据时。当用户必须和一个现存的使用了特定 XML 的应用程序交换数据时，就会发生这种情况。
- 用户希望用 XML 保存数据并为未来可能的整合做好准备时。因为使用了 XML，用户知道将来可使第三方应用程序读取这些数据。
- 用户希望使用依赖于 XML 的技术时。例如，Web 服务使用各种建立在 XML 上的标准。

很多.NET 的功能在幕后使用 XML。例如，Web 服务使用一个建立在 XML 基础架构上的高层模型。使用 Web 服务时，用户不需要直接操作 XML，而是可以直接使用一个抽象对象。类似地，用户不需要直接操作 XML 来读取 ASP.NET 用户配置，或不需要将 DataSet 保存到一个文件或者依赖于其他背后有 XML 的.NET Framework 特性。在所有这些场景里，XML 在后台默默工作，用户不需要手工处理 XML 就可获得它的所有好处。

XML 在应用程序整合的场景中最有意义。不过，也没有理由说用户不可以使用 XML 格式保存用户的专有数据。这么做可以获得一些便利，如可以使用.NET 类从文件

中读取 XML 数据。当保存复杂、高度结构化的数据时，使用这些类带来的便利非常显著，它远高于用户自行设计文件格式并编写自己的文件解析逻辑所带来的便利。它还让其他开发人员更方便地理解、重用或改进用户创建的系统。

提示：开发人员必须理解的一个最重要的概念是，存储数据时必须决定两件事情：确定数据结构化的方式(逻辑格式)和确定数据保存的方式(数据的物理存储)。XML 是格式的选择而不是存储的选择。也就是说，如果用户决定使用 XML 格式保存数据，用户还要决定 XML 是要保存到数据库字段里，还是要播入到一个文件里，或者只是以字符串或其他对象的形式保存在内存中。

5.1.1.2 XML 简介

XML 规范是由 W3C(World Wide Web Consortium)定义的一组指南，用于以纯文本的形式描述结构化数据。和 HTML 类似，XML 是一种基于尖括号间标签的标记语言。和 HTML 一样，XML 的文本本质使得数据具有高度便携性并易于发布。此外，可以用任何标准的文本编辑器创建和编辑 XML 文档。

和 HTML 不同的是，XML 没有一组固定的标签。相反，XML 是一种可用于创建其他标记语言的元语言。换句话说，XML 建立用于命名和元素排列的简单规则，它允许用自定义的元素创建自己的数据格式。

下面是 XML 示例的代码。

```
<? xml version = "1.0" encoding = "utf - 8"? >
<PersonData>
  <People ID = "001">
  <Name>MyName</Name>
  <Age>30</Age>
  <Height>170</Height>
  </People>
<People ID = "002">
  <Name>MyName2</Name>
  <Age>32</Age>
  <Height>173</Height>
  </People>
<People ID = "003">
  <Name>MyName3</Name>
  <Age>31</Age>
  <Height>160</Height>
  </People>
<People ID = "004">
  <Name>MyName4</Name>
  <Age>34</Age>
  <Height>176</Height>
  </People>
<People ID = "005">
```

```
<Name>MyName5</Name>
<Age>35</Age>
<Height>175</Height>
</People>

</PersonData>
```

这个示例中使用<PersonData>、<People>、<Name>之类的元素声明文档的结构。不过可以自由使用最能描述数据的任意元素名称。

正是这种灵活性使 XML 非常成功。当然，灵活性也会有缺点。因为 XML 没有定义任何标准的数据格式，这就需要用户自行定义代表产品类别、发票、客户列表等的数据格式。不同的公司可以很容易地使用完全不同的标签名称和结构来保存相似的数据。尽管所有的应用程序都能够解析 XML 数据，但数据的写入者和读取者仍然需要对标签和结构达成共识，才能使读取者可以解释数据并抽取有意义的信息。

通常，第三方组织会为特定的问题域和行业定义标准。例如，如果用户需要在 XML 中保存数学等式，用户很可能会选择 MathML 格式。它是一种定义了具体标签以及具体结构的基于 XML 的格式。类似地，还存在数百个标准 XML 格式。它们用于房地产业、音符、法律文件、专利记录、矢量图形等。创建一个强健易用的 XML 格式需要具备一些经验，因此只要可能，最好使用标准的、广为接受的、基于 XML 的标记语言。

XML 最初诞生时，它的成功部分源于它的简单。XML 的规则远比它的前身 SGML (Standard Generalized Markup Language)要短小简单，并且简单的 XML 文档是可读的。不过在这几年中，许多其他支持标准不断地加入 XML，这样，在专业的应用程序里使用 XML 一点也不简单。

不过无论如何，现在 XML 比过去任何一天都更为有用。现代应用程序使用 XML 的好处有以下几点。

- 适应性。XML 无处不在。很多公司采用 XML 保存数据或正在积极考虑使用它。无论什么时候需要共享数据，XML 都会自动成为他们第一个(经常也是唯一一个)考查的对象。
- 扩展性和灵活性。和 EDI (电子数据交换)不同，XML 不会强加任何数据语义的规则，也不会把公司强行绑定到某个专用网络。这样，XML 适用于任意类型的数据并且实现的代价很低。
- 相关标准和工具。XML 成功的另一个原因在于帮助创建和处理 XML 文档的工具(如解析器)和相关标准(如 XML 架构、XPath 和 XSLT)。这样，几乎每种语言的开发人员都有现成的组件用于阅读 XML，验证 XML 是否有效，按某种规则(被称作架构)验证 XML 的有效性，搜索 XML 以及把 XML 从一种格式转换为另一种格式。

XML 像黏合剂一样让不同的系统可以共同协作。它帮助标准化业务进程和组织间事务。不过 XML 并不仅仅适用于公司间的数据交换。今天，很多的程序任务都是关于应用程序整合的——Web 应用程序整合众多 Web 服务，电子商务站点整合遗留的库存和定价系统，局域网应用程序整合现有的业务应用程序。所有这些应用程序都是通过交换 XML 文档从而组合在一起的。

5.1.1.3 XML 命名规范

XML 是一个非常严格的标准。这种严格性是用于保留广泛的兼容性的。如果 XML 的规则不严格，就会很难区分无害的不一致和严重的错误。更糟的是，不同的 XML 解析器对一些错误的处理方式可能不同，这就会导致处理的不一致性（或者甚至根本不能处理）。正是这些缺陷影响了一种不是基于 XML 的臭名昭著的语言——HTML。

为了避免这类问题，所有的 XML 解析器都会执行一些基本的质量检查。如果一个 XML 文档不能满足所有标谁，它就会被彻底拒绝。如果 XML 文档遵循这些规则，它就被认为是格式良好的。格式良好的 XML 不一定就是正确的 XML——例如，它还可能包含不正确的数据——不过，XML 解析器能够解析它。

XML 标签的命名规则如下：

- 区分大小写；
- 以字母或下划线开头；
- 可以包含字母、数字、下划线、连字符、句号。（不包含空格）

XML 标签必须有一个结束标签，有开始标签，必须有结束标签。开始标签和结束标签之间可以存放数据。通过字符和实体引用，可以通过引用将信息加入 XML 文档，而不必直接在文档中键入字符。在下列情况下，这样做很有用：

- 因为会被解释为标记，字符无法直接输入文档；
- 因为输入设备的限制，字符无法直接输入文档；
- 字符无法通过限于单字节字符的处理器可靠地传输；
- 字符串或文档片断反复出现，并且可以缩写。

为了显示内容，XML 提供了许多语法构造，以“and”符（&）开头，以分号（;）结尾。

通过字符引用，可以插入通过指向 Unicode 代码点的数字标识的 Unicode 字符。代码点可以使用十进制或十六进制表示法标识。

- &#value; 用于十进制引用的语法。
- &#xvalue; 用于十六进制引用的语法

例如，要插入欧元这个很多键盘仍然没有的符号，可以在文档中插入 &#x20AC 或 €。

表 5-1 为 XML 标记使用的字符的五种内置实体。

表 5-1 内置实体

实体	实体引用	含义
lt	&lt;	<（小于号）
gt	&gt;	>（大于号）
amp	&amp;	&（“and”符）
apos	&apos;	'（撇号或单引号）
quot	&quot;	"（双引号）

如果字符可能会使 XML 分析器错误地解释文档结构，请使用实体，而不要键入字符。' 和 " 实体引用在属性值中最常用。例如，要写 Me&You，请使用 Me&You。对于 a<b，请使用 a<b。对于 b>c，请使用 b>c。

5.1.1.4　XML 命名空间

随着 XML 标准的成长，已创建了数十种 XML 标记语言(通常叫做 XML 语法)，其中很多属于特定的行业、流程和信息类型。很多时候，能够使用某个公司特定的元素扩展某类标记，甚至还能够创建一个组合几个不同 XML 语法的 XML 文档变得很重要。这就带来一个问题。如果用户需要同时组合两个具有相同名称元素的 XML 语法，会发生什么呢？另一个相关但更典型的问题是如何区分它们。当应用程序需要区别文档中 XML 语法时，这个问题就会发生。例如，假设一个 XML 文档拥有订单以及用户相关的信息，它们分别使用名为 OrderML 和 ClientML 的标准。这个文档被送到一个订单实现应用程序，它只关心 OrderML 的内容。它如何快速区分哪些信息是需要的？哪些信息是需要被忽略的呢？

解决办法在于 XML 命名空间标准。这个标准背后的核心思想是所有的 XML 标记语言都拥有能够唯一区分相关元素的命名空间。从技术角度而言，命名空间通过明确元素使用的标记语言来消除元素的歧义。

所有的 XML 命名空间都使用 URI(Universal Resource Identifiers，统一资源标识符)。一般说来，这些 URI 看起来和网页的 URL 相似。例如，http://www.mycompany.com/mystandard 是一个典型的命名空间名称。尽管命名空间看起来是 Web 上某个有效的地址，但这不是必要的(也不应该被假设)。URI 被用于 XML 命名空间，因为它们大部分情况下是唯一的。通常，如果用户创建了一个新的 XML 语言，就会使用一个指向用户所控制的某个域名或站点的 URI。这样可以确保没有其他人使用这个 URI。不过，命名空间不一定是 URI——任意文本序列都是可以的。

要指定某个元素属于特定的命名空间，用户只需在开始标签中加入 xmlns 特性表明要使用的命名空间即可。例如，这里显示的元素是 http://mycompany/OrderML 命名空间的一部分。如果用户越过了这一步，这个元素将是所有命名空间的一部分：

```
<order xmlns = " http://mycompany/OrderML"></order>
```

用户可能会厌倦在 XML 文档中每加入一个元素时都要输入一遍完整的命名空间 URI。幸好，如果用户采用这种方式设置一个命名空间，它会成为所有子元素默认的命名空间。例如，在下面所示的 XML 文档中，<order>和<orderItem>都放在 http://mycompany/OrderML 命名空间里：

```
<? xml version = "1.0"? >
<order xmlns = " http://mycompany/OrderML">
    <orderItem>…</orderItem>
    <orderItem>…</orderItem>
    <orderItem>…</orderItem>
    <orderItem>…</orderItem>
</order>
```

用户可以为文档的单独部分定义一个新的命名空间。最简单的方法是使用命名空间前缀。命名空间前缀是可以插入到标签名称前面的一个短字符序列。定义前缀时，在

xmlns 特性中插入一个冒号和用户希望用作前缀的字符。

这个订单文档使用命名空间前缀把不用的元素映射到两个不同的命名空间中：

```
<? xml version = "1.0"? >
<ord:order xmlns:ord = " http://mycompany/OrderML"
        xmlns:ord = " http://mycompany/ClientML">
    <cli:client>
        <cli:firstName>…</cli:firstName>
<cli:lastName>…</cli:lastName>
    </cli:client>
    <ord:orderItem>…</ord:orderItem>
<ord:orderItem>…</ord:orderItem>
</ord:order>
```

命名空间前缀只用于把元素映射到某个命名空间。只要保持一致，用户所使用的实际的前缀并不重要。

5.1.1.5 XML 架构

XML 标准之所以成功，很大一部分要归功于它极端的灵活性。使用 XML，可以创建完全符合需求的标记语言。这种灵活性也带来了一些问题。世界各地的开发人员都使用用户的 XML 格式，怎样才能保证所有人都遵守规则？

解决办法是创建一个格式文档，它定义用户的自定义标记语言的规则，被称为架构。这些规则不会包括语法细节（如需要使用尖括号或需要正确的嵌套标签）。因为这些已经是 XML 标准要求的一部分。架构文档需要定义的是符合用户的数据类型的逻辑规则，它们包括以下几项。

- 文档词汇。它定义了哪些元素或特性的名字可以在用户的 XML 文档中使用。
- 文档结构。它定义了标签放在哪里，还包括一些指定某些标签必须放在其他标签之前、之后或之中的规则。用户还可以指定某个元素可以出现的次数。
- 支持的数据类型。这允许用户定义数据是普通文本，或者必须是可解析的数值数据、日期信息等。
- 允许的数据范围。这允许用户将数值限制在某个范围内，将文本限制在某个特定的长度内，强迫正则表达式模式匹配，或者限制仅可以是某些特定的值。

XML 架构标准定义了用户在创建架构文档时必须遵守的规则。下面这个 XML 架构定义了前面所示的产品类别规则：

```
<? xml version = "1.0"? >
<xsd:schema xmlns:xsd = "http://www/w3/org/2001/XMLSchema">
  <xsd:element name = "productCatalog">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name = "catalogName" type = "xsd:string"/>
        <xsd:element name = "expiryDate" type = "xsd:date"/>
```

```
        <xsd:element name = "products">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name = "product" type = "productType" maxOccurs = "unbounded" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name = "productType">
    <xsd:sequence>
      <xsd:element name = "productName" type = "xsd:string" />
      <xsd:element name = "productPrice" type = "xsd:decimal" />
      <xsd:element name = "inStock" type = "xsd:boolean" />
    </xsd:sequence>
    <xsd:attribute name = "id" type = "xsd:integer" use = "required"/>
  </xsd:complexType>
</xsd:schema>
```

所有的架构文档都是以跟元素<schema>开头的 XML 文档。这些元素在 XML 架构命名空间(http://www/w3/org/2001/XMLSchema)中定义。用户的架构文档必须使用正确的命名空间名。虽然一般使用 xsd(本例中使用)和 xs,但是用户可以将它自由映射到用户喜欢的命名空间前缀,以便在架构文档中使用它。

在<schema>元素内,有两种类型的定义。一种类型是<schema>元素,它定义目标文档必须遵循的结构;还有一种类型是<complexType>元素,它用于定义文档结构的较小的数据结构。<element>标签是架构的核心,同时它还是所有验证的起点。在这个示例中,<element>标签确定产品类别必须以一个叫做<productCatalog>的跟元素开始。<productCatalog>元素内部是一个由三个元素组成的序列。第一个元素是<catalogName>,它包含普通文本;另一个元素是<expiryDate>,它包含符合日期呈现规则的文本;最后一个元素是<products>,它包含<product>元素的列表。每个<product>元素都是一个复杂的类型,这个类型由<complexType>在文档最后定义。这个复杂类型(名为 productType)由含有产品信息的三个元素所组成的序列构成。这些元素必须分别保存为文本(<productName>)、十进制数(<productPrice>)和布尔值(<inStock>)。这个复杂类型包含一个必需的特性 id。

5.1.2　HTML

HTML 即超文本标记语言,是标准通用标记语言下的一个应用。其中“超文本”就是指页面内可以包含除文字以外的其他非文字元素,如图片、链接、音乐、程序等。表5-2是

HTML 元素汇总表。

表 5-2 HTML 元素汇总

文档类型宣告	<! DOCTYPE html>
根元素元素	html
META 元素	head、title、base、link、meta、style
部件元素	body、section、nav、article、aside、h1、h2、h3、h4、h5、h6、hgroup、header、footer、address
分组内容元素	p、hr、br、pre、blockquote、ol、ul、li、dl、dt、dd、figure、figcaption、div
文本层次语义元素	a、em、strong、small、cite、q、dfn、abbr、time、code、var、samp、kbd、sub、sups、i、b、mark、ruby、rt、rp、bdo、span
编辑元素	ins、del
嵌入内容元素	img、iframe、embed、object、param、video、audio、source、canvas、map、area
表格元素	table、caption、colgroup、col、tbody、thread、tfoot、tr、td、th
表单元素	form、fieldset、legend、label、button、select、datalist、optgroup、option、textarea、keygen、output、progress、meter
互动元素	details、summary、command、menu
脚本元素	script、noscript

1. 文档应该是结构良好的

结构良好(Well-formed)是由 XML 引入的一个新概念。也就是说所有的元素都必须是结束标签或者以特殊的方式书写,所有的标签必须合理地嵌套。

尽管如此,嵌套使用在 HTML 中仍然是合法的,而且在现有的浏览器中都能够被广泛接受。

- 正确:元素嵌套

```
<p><em>这是段落一。</em></p>
```

- 错误:元素交叉

```
<em><p>这是段落一。</em></p>
```

2. 对于非空标签也需要结束标签

在 HTML 中每一个标签都会有结束标记,有关结束标签要注意:

(1)基于 SGML 的 HTML4 里面,允许特定的标签省略结束标签;这些元素暗含有结束标记。

(2)HTML 中不允许省略结束标记。所有元素必须有结束标签。

(3)在 DTD 中声明为空的元素,可以用结束标签或者使用空元素速记法。

- 正确:结束的元素

```
<p>明天会更好。</p><p>后天会更好。</p>
```

- 错误:没有结束的元素

```
<p>明天会更好。<p>后天会更好。
```

3. 属性值必须总是使用引号

所有的属性值都必须使用引号包含,包括那些以数值类型出现的。

- 正确:属性值使用引号

```
<td rowspan = "3"/>
```

- 错误:属性值没有使用引号

```
<td rowspan = 3/>
```

4. 禁止属性简化

(1)HTML 不支持属性简化,属性值对必须书写完整。

(2)属性名像 compact 和 checked 在没有指定具体值班的情况下,属性名不能够使用。

• 正确:没有简化属性

```
<dl compact = "compact" />
```

• 错误:简化属性

```
<dl compact />
```

5. 空元素

空元素必须有一个结束标签,或者用 />来结束开始标签。例如,
或者<hr ></hr>。

• 正确:结束空标签

```
<br /><hr />
```

• 错误:没有结束空标签

```
<br><hr>
```

6. 了解 HTML5

下列各表介绍了 HTML5 的基本标签内容,如表 5-3 列出了 HTML5 的主体结构标签,表 5-4 列出了 HTML5 各元素标签释义,表 5-5 列出了 HTML 5 元素通用属性和事件句柄,表 5-6 汇总了 HTML5 元素标记。

表 5-3　　HTML5 的主体结构标签

header	页面头部,不同于<head></head>
aside	边栏
nav	外部链接集合
section	章节或段落
article	类似文章、摘要或留言 POST 等形式的记录(一般搭配内嵌头部、尾部、底部结构使用)
hggroup	类似于标题,标题信息、可选标题、TAG 标签这样的数据,还是英文更好理解一些,英文是 heading of a section
address	联系信息,一般用在 article 或 body 锚元素周围
footer	页脚

表 5-4　　HTML5 各元素标记释义

标记	类型	意义	介绍
文件标记			
<html>	•	根文件标记	让浏览器知道这是 HTML 文件
META 标记			
<head>	•	开头	提供文件整体信息
<title>	•	标题	定义文件标题,显示于浏览器顶端
<base>	o	基准标记	可将相对 URL 转绝对及指定链接
<link>	o	外部资源连接	必须带 rel 属性描述
<meta>	o	其他 META 数据	不能被 title, base, link, style, 和 script 元素描述的 META 数据

（续表）

标记	类型	意义	介绍
＜style＞	•	嵌入文档风格信息	
部件标记			
＜body＞	•	文档主体开始	文档内容容器
＜section＞	•	代表通用文档或应用部件	
＜nav＞	•	导航链接	外部链接或文档内部链接
＜article＞	•	页面模块	类似文章、摘要或留言 POST 等形式的记录
＜aside＞	•	孤立模块	一般作为边栏广告、说明、引用、导航等，aside 围堵部分一般与正文耦合较小
＜h1＞	•	标题标记	此外还有 h2，h3，h4，h5，h6
＜hgroup＞	•	群组标题	用在一组 h1～h6 这样的元素集合时使用，用来区分主副标题
＜header＞	•	组说明或组导航	也可叫页头标题
＜footer＞	•	页脚标题	作用范围跟最近部件元素有关
＜address＞	•	地址或联系信息	
分组内容标记			
＜p＞	•	段落标记	
＜hr＞	o	水平分割线	
＜br＞	o	换行	
＜pre＞	•	预格式化分本块	
＜blockquote＞	•	块引用	
＜ol＞	•	编号列表	
＜ul＞	•	项目列表	
＜li＞	•	列表项	
＜dl＞	•	定义列表	
＜dt＞	•	定义名称	
＜dd＞	•	定义说明	
＜figure＞	•	流内容区块说明	多结合 figcaption 使用
＜figcaption＞	•	figure 内容属性	
＜div＞	•	定位标记	无实际意义
文本层次语义标记			
＜a＞	•	链接标记	
＜em＞	•	强调标记	
＜strong＞	•	加重标记	
＜small＞	•	字体缩小	
＜cite＞	•	斜体标记	
＜q＞	•	引用标记内容	
＜dfn＞	•	术语定义	
＜abbr＞	•	缩略语	
＜time＞	•	日期时间	
＜code＞	•	程序代码	

（续表）

标记	类型	意义	介绍
＜var＞	•	变量	
＜samp＞	•	范例	
＜kbd＞	•	键盘字	
＜sub＞＜sups＞	•	上标字/下标字	
＜i＞	•	斜体标记	
＜b＞	•	粗体标记	
＜mark＞	•	标记或高亮	
＜ruby＞	•	注解标记	
＜rt＞	•	ruby 子元素	结合 ruby 使用，比如：＜ruby＞天＜rt＞Tian＜/rt＞缘＜rt＞Yuan＜/rt＞＜/ruby＞
＜rp＞	•	ruby 子元素	一般做 rt 元素注释使用
＜bdo＞	•		
＜span＞	•	自定义标记	
编辑标记			
＜ins＞	•		
＜del＞	•		
嵌入内容标记			
＜img＞	•	图片标记	
＜iframe＞	•	框架标记	
＜embed＞	•	嵌入标记	
＜object＞	•	对象标记	
＜param＞	•	参数标记	
＜video＞	•	视频标记	
＜audio＞	•	音频标记	
＜source＞	•	来源标记	
＜canvas＞	•	制图标记	
＜map＞	•	地图标记	
＜area＞	•	区域标记	
表格标记			
＜table＞	•	表格标记	设定该表格的各项参数
＜caption＞	•	表格标题	做成一打通列以填入表格标题
＜colgroup＞	•		
＜col＞	•		
＜tbody＞	•		
＜thread＞	•		
＜tfoot＞	•		
＜tr＞	•	表格列	设定该表格的列
＜td＞	•	表格栏	设定该表格的栏
＜th＞	•	表格标头	相等于＜TD＞，但其内文字字体会变粗

（续表）

标记	类型	意义	介绍
互动元素			
＜details＞	•		
＜summary＞	•		
＜command＞	•		
＜menu＞	•		
其他标记			
＜script＞	•		
＜noscript＞	•		
表单标记			
＜form＞	•	表单标记	决定该表单的运作模式
＜fieldset＞	•		
＜legend＞	•		
＜input＞	•	输入标记	
＜label＞	•		
＜button＞	•	按钮	
＜select＞	•	选择标记	
＜datalist＞	•		
＜optgroup＞	•		
＜option＞	•	选项	
＜textarea＞	•		
＜keygen＞	•		
＜output＞	•		
＜progress＞	•		
＜meter＞	•		

备注：• 表示该标记属于围堵标记，需要结束标记＜/标记＞。

o 表示该标记属空标记，不需要结束标记。

表 5-5　　HTML5 元素通用属性和事件句柄

属性/事件	句柄
HTML5 元素通用属性表	accesskey、class、contenteditable、contextmenu、dir、draggable、hidden、id、lang、spellcheck、style、tabindex、title
HTML5 元素事件句柄属性	onabort、onblur ＊、oncanplay、oncanplaythrough、onchange、onclick、oncontextmenu、ondblclick、ondrag、ondragend、ondragenter、ondragleave、ondragover、ondragstart、ondrop、ondurationchange、onemptied、onended、onerror ＊、onfocus ＊、onformchange、onforminput、oninput、oninvalid、onkeydown、onkeypress、onkeyup、onload ＊、onloadeddata、onloadedmetadata、onloadstart、onmousedown、onmousemove、onmouseout、onmouseover、onmouseup、onmousewheel、onpause、onplay、onplaying、onprogress、onratechange、onreadystatechange、onscroll、onseeked、onseeking、onselect、onshow、onstalled、onsubmit、onsuspend、ontimeupdate、onvolumechange、onwaiting

表 5-6　　HTML5 元素标记汇总表

文档类型宣告	＜! DOCTYPE html＞
根元素元素	html
META 元素	head、title、base、link、meta、style
部件元素	body、section、nav、article、aside、h1、h2、h3、h4、h5、h6、hgroup、header、footer、address
分组内容元素	p、hr、br、pre、blockquote、ol、ul、li、dl、dt、dd、figure、figcaption、div
文本层次语义元素	a、em、strong、small、cite、q、dfn、abbr、time、code、var、samp、kbd、sub、sups、i、b、mark、ruby、rt、rp、bdo、span
编辑元素	ins、del
嵌入内容元素	img、iframe、embed、object、param、video、audio、source、canvas、map、area
表格元素	table、caption、colgroup、col、tbody、thread、tfoot、tr、td、th
表单元素	form、fieldset、legend、label、button、select、datalist、optgroup、option、textarea、keygen、output、progress、meter
互动元素	details、summary、command、menu
脚本元素	script、noscript

5.1.3　级联样式表

级联样式表(CSS)是一种用来表现 HTML 或者 XML 等文件样式的计算机语言，目前最新版本为 CSS3。它是能够真正做到内容与网页表现相分离的一种样式设计语言。相对于传统 HTML 的表现而言，CSS 能够对网页中的对象的位置排版进行像素级的精确控制，支持几乎所有的字体、字号样式，拥有对网页对象和模型样式编辑的能力，并能够进行初步交互设计，是目前基于文本展示最优秀的表现设计语言。目前也有不少框架针对 CSS 和 JQuery(轻量级的 JavaScript 库、兼容 CSS3 和各种浏览器)进行了一定的包装和优化，如 Twitter 推出的一个开源的用于前端开发的工具包 Bootstrap，有兴趣的同学可以深入了解一下。

5.1.4　JavaScript

一种直译式脚本语言 JavaScript，是一种动态类型、弱类型、基于原型的语言，内置支持类型。它的解释器被称为 JavaScript 引擎，为浏览器的一部分，广泛用于客户端的脚本语言。最早是在 HTML 网页上使用，用来给 HTML 网页增加动态功能。然而现在 JavaScript 也可被用于网络服务器，如 Node.js。

1995 年，JavaScript 由网景公司的布兰登·艾克，在网景导航者浏览器上首次设计实现而成。因为网景公司与升阳公司合作，网景公司管理层次结构希望它外观看起来像 Java，因此取名为 JavaScript。但实际上它的语法风格与 Self 及 Scheme 较为接近。

为了取得技术优势，微软推出了 JScript，CEnvi 推出 ScriptEase，与 JavaScript 同样可在浏览器上运行。为了统一规格，1997 年，在 ECMA(欧洲计算机制造商协会)的协调下，由 Netscape、Sun、微软、Borland 组成的工作组确定统一标准：ECMA-262。因为 JavaScript 兼容于 ECMA 标准，因此也称为 ECMAScript。

5.1.4.1　Javascript 的引用方式

1. Script 标签

如需在 HTML 页面中插入 JavaScript，请使用 ＜script＞ 标签。＜script＞ 和 ＜/

script> 会告诉 JavaScript 在何处开始和结束。<script> 和 </script> 之间的代码行包含了 JavaScript：

例如

```
<script>
alert("My First JavaScript");
</script>
```

用户无需理解上面的代码。只需明白，浏览器会解释并执行位于 <script> 和 </script> 之间的 JavaScript。那些老旧的实例可能会在 <script> 标签中使用 type="text/javascript"。现在已经不必这样做了。JavaScript 是所有现代浏览器以及 HTML5 中的默认脚本语言。

2. <body> 中的 JavaScript 示例

在本例中，JavaScript 会在页面加载时向 HTML 的 <body> 写文本：

```
<! DOCTYPE html>
<html>
<body>
<script>
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph</p>");
</script>
</body>
</html>
```

3. <head> 或 <body> 中的 JavaScript

用户可以在 HTML 文档中放入不限数量的脚本。脚本可位于 HTML 的 <body> 或 <head> 部分中，或者同时存在于两个部分中。

通常的做法是把函数放入 <head> 部分中，或者放在页面底部。这样就可以把它们安置到同一处位置，不会干扰页面的内容。<head> 中可以加入 JavaScript 函数。

在本例中，我们把一个 JavaScript 函数放置到 HTML 页面的 <head> 部分。该函数会在点击按钮时被调用：

```
<! DOCTYPE html>
<html>

<head>
<script>
function myFunction()
{
document.getElementById("demo").innerHTML = "My First JavaScript Function";
}
</script>
</head>
```

```
<body>

<h1>My Web Page</h1>

<p id = "demo">A Paragraph</p>

<button type = "button" onclick = "myFunction()">Try it</button>

</body>
</html>
<body> 中的 JavaScript 函数示例,在点击按钮时被调用。
<! DOCTYPE html>
<html>
<body>

<h1>My Web Page</h1>
<p id = "demo">A Paragraph</p>
<button type = "button" onclick = "myFunction()">Try it</button>
<script>
function myFunction()
{
document.getElementById("demo").innerHTML = "My First JavaScript Function";
}
</script>

</body>
</html>
```

4. 引用外部 Javascript

也可以把脚本保存到外部文件中。外部文件通常包含被多个网页使用的代码。外部 JavaScript 文件的文件扩展名是 .js。如需使用外部文件,请在 <script> 标签的“src”属性中设置该 .js 文件:

```
<! DOCTYPE html>
<html>
<body>
<script src = "myScript.js"></script>
</body>
</html>
```

5.1.4.2　Javascript 的变量

Javascript 在定义变量时,统一使用“var 变量名”表示,例如:var str;甚至可以省略

var 这个关键字。Javascript 变量的类型是由 JS 引擎决定的。与代数一样，JavaScript 变量可用于存放值(比如 x=2)和表达式(比如 z=x+y)。

- 变量可以使用短名称(比如 x 和 y)，也可以使用描述性更好的名称(比如 age，sum，totalvolume)。
- 变量必须以字母开头
- 变量也能以 $ 和 _ 符号开头(不过我们不推荐这么做)
- 变量名称对大小写敏感(y 和 Y 是不同的变量)

提示：JavaScript 语句和 JavaScript 变量都对大小写敏感。

例如：

- var name="这是一个 string 类型";
- //name 是一个 string 类型；
- var age=24;//age 是 number 类型；
- var flag=true;//flag 是 boolean 类型；
- var email;//email 只是声明，没有赋值，因此代表的类型是"undefined"，也就是无法确定；
- name=10;//name 自动变成了 number 类型。

5.2 了解 Microsoft ASP. NET Web 应用程序开发

本节主要讲解了. NET Web 应用开发的生命周期、事件模型等等，这非常有助于读者建立 Web 应用开发的初步思想，为后来的学习指出了方向。

5.2.1 页面生命周期

在创建 ASP. NET 网站之前，了解页面生命周期非常重要。在服务器端，对 ASP. NET 页面的处理是按阶段发生的，在各个阶段都会产生各种事件。这允许页面能够在任何阶段插入到处理流并按照所期望的那样去回应。

在这里简单地了解一下 ASP. NET 页面处理流程的主要阶段。下面列出了一个 ASP. NET 页面处理流程的主要阶段：

- 初始化
- 加载
- 回发事件处理
- 呈现
- 卸载

重要的是，这些阶段都是在每一次 Web 请求时独立发生的。图 5-1 显示了这些阶段执行的顺序。当然还有许多阶段并没有出现在

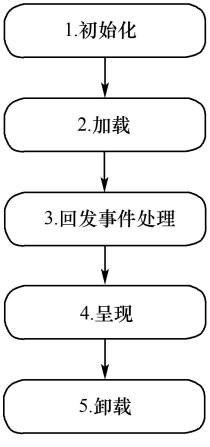

图 5-1 ASP. NET 页面生命周期

图 5-1 中，这是因为它们通常只是在编写 ASP.NET 自定义控件时才被使用，并且不会由页面直接处理。

在页面生命周期的每个阶段，页面将引发可运行自定义代码进行处理的事件。对于控件事件，通过以声明方式使用特性（如 onclick）或以使用代码的方式，均可将事件处理程序绑定到事件。下面列出了最常用的页面生命周期事件：PreInit、Init、InitComplete、Preload、Load、控件事件、LoadComplete、PreRender、PreRenderComplete、SaveStateComplete、Render（这不是事件，但 Page 类会调用控件上的这个方法）、Unload 等。

5.2.2　事件模型

传统的 ASP 使用线性处理模型，这就意味着页面上的代码是从开始到结束来进行处理的，并按照顺序执行。由于是线性处理模型，传统的 ASP 开发人员即使写一个简单的页面，也需要写一定数量的代码。一个典型的例子是一个具有 3 个“提交”按钮（它们的作用各不相同）的页面。在这个例子中，用户的脚本代码需要区别提交页面时单击的是哪个按钮，并根据条件逻辑执行正确的动作。

ASP.NET 提供了一个全新的变化——事件驱动模型。在这个模型中，自定义在 Web 窗体中添加控件并决定响应的事件。每个事件处理程序封装在单独的方法中，这就保持了页面的整洁性和组织性。事件模型并不是新概念，但直到 ASP.NET 出现以前，它只应用于富客户端应用的 Window 用户界面的编程。下面列出“提交”按钮点击事件的代码：

Default.aspx 文件：

```
<asp:Button ID = "btn" runat = "server" Text = "提交" Width = "120" Height = "30" OnClick = "
btn_Click" />
```

Default.aspx.cs 文件：

```
protected void btn_Click(object sender, EventArgs e)
    {
        StringBuilder strb = new StringBuilder();
        if (cb_csharp.Checked)
        {
            strb.Append("C#");
        }
        if(cb_vbnet.Checked)
        {
            strb.Append("VB.NET");
        }
        if (strb ! = null)
        {
            String s = "我的名字是:" + tb_username.Text.Trim() + ",我的邮箱是" + tb_
email.Text.Trim() + ",我使用的的编程语言是" + strb.ToString();
            lbl_Welcome.Text = s.ToString();
```

```
                lbl_Welcome.ForeColor = Color.Red;
            }
            else
            {
                String s = "我的名字是:" + tb_username.Text.Trim() + ",我的邮箱是" + tb_
email.Text.Trim();
                lbl_Welcome.Text = s.ToString();
                lbl_Welcome.ForeColor = Color.Red;
            }
        }
```

Default.aspx.vb 文件:

```
    Protected Sub btn_Click(sender As Object, e As EventArgs)
            Dim str As String = tb_username.Text

            Dim message As String = "<span style=""color:Red"">欢迎用户," + tb_username.
Text + "</span>"
            Response.Write(message)

            lbl_Welcome.Text = "用户好," + tb_username.Text
            lbl_Welcome.ForeColor = Drawing.Color.Red

            Dim strb As New StringBuilder
            If (cb_csharp.Checked) Then
                strb.Append("C# ")
            End If
            If (cb_vbnet.Checked) Then
                strb.Append("VB.NET")
            End If
            If (String.IsNullOrEmpty(strb.ToString())) Then

                Dim s As String = "我的名字是:" + tb_username.Text.Trim() + ",我的邮箱是"
+ tb_email.Text.Trim()
                lbl_Welcome.Text = s.ToString()
                lbl_Welcome.ForeColor = Drawing.Color.Red

            Else
                Dim s As String = "我的名字是:" + tb_username.Text.Trim() + ",我的邮箱是"
+ tb_email.Text.Trim() + ",我使用的的编程语言是" + strb.ToString()
                lbl_Welcome.Text = s.ToString()
                lbl_Welcome.ForeColor = Drawing.Color.Red
            End If
        End Sub
```

图 5-2 显示了“提交”按钮点击事件处理程序运行结果。

图 5-2　“提交”按钮点击事件程序运行结果

大多数 ASP.NET 网页的代码都被放在用来处理 Web 控件事件的事件处理程序里面。使用 Visual Studio,用户有以下 3 种方法来添加一个事件处理程序。

• 手工键入。在这种情况下,需要在页面类中直接添加方法,并且必须制定适当的参数以便事件处理程序的签名能精确地匹配想要处理的事件的签名。也需要通过添加 OnEventName 特性,编辑控件标签来让它关联到相应的事件处理程序上。(另外,可以使用委托通过编程来做到这一点。)

• 在设计视图中双击控件。在这种情况下,Visual Studio 会为这个控件的默认事件创建一个事件处理程序(并相应的调整控件标签)。比如,如果双击页面,Visual Studio 将会创建一个 Page_Load 事件处理程序。如果双击按钮控件,Visual Studio 将为 Click 事件创建一个事件处理程序。

• 从“属性”窗口中选择事件。选中控件并单击“属性”窗口上方的闪电形图标,用户会看到这个控件所提供的所有事件列表。双击要处理的事件旁边的空白区域,Visual Studio 会自动在页面类中生成事件处理程序并调整控件标签。

第二个和第三个办法最为方便。最三个方法最灵活,因为它允许选择已在页面类中创建的方法。在“属性”窗口中选中事件并单击右边的下拉箭头,用户就会看到页面类中所有和事件签名匹配的方法列表,用户可以从该列表中选择一个方法,并把它和事件相关联。这项技术的唯一局限是它只对 Web 控件有效,而对服务器端 HTML 控件无效。

5.2.3　客户端与服务器端编程的对比

服务器端语言指的是在服务器运行的动态语言,如对 CSS、Javascript、HTML(客户端部分)来说,服务器端运行的动态语言主要是对数据库的操作和访问,当然还有其他的一些功能。但主要是对数据库进行操作。

举例来说,要判断用户输入的用户名是中文还是英文,有没有带数字,这样的功能可以用客户端脚本语言来完成,但要判断这个用户有没有在网站进行过注册。由于需要将用户输入的用户名与数据库中的信息进行比对,因此一定需要服务器端运行的动态语言才行。另外客户端上处理大多数的工作,就减少通过网络发送数据的需求量。发送给服务器的大量数据可能会导致延迟。对于在本地计算机上运行的应用程序,一般事件都会得到迅速处理。还有在本地网络上传输数据的速度比通常的 Internet 连接快很多。这样一来,数据在网络上的传输速度更快,产生的瓶颈问题也会更少。以 VB. NET 为例,先看以下这段代码:

```
<%@ Page Title = "Home Page" Language = "VB"
    MasterPageFile = "~/Site.Master"  AutoEventWireup = "true"
    CodeFile = "Default.aspx.vb" Inherits = "_Default" %>
<asp:Content runat = "server" ID = "HeadContent" ContentPlaceHolderID = "HeadContent">
    <script lang = "javascript" type = "text/javascript">
        function onClikc()
        {
            document.getElementById("lblClient").innerText = "Changed";
            document.getElementById("lblServer").innerText = "Server";
        }
    </script>
</asp:Content>
<asp:Content runat = "server" ContentPlaceHolderID = "MainContent">
    <asp:Button ID = "btnServer" runat = "server" Text = "Server" />
    <asp:Label ID = "lblServer" runat = "server" Text = "Server" ClientIDMode = "Static" />
    <br />
    <input id = "btnClient" type = "button" value = "Client" onclick = "return onClikc()" />
    <asp:Label ID = "lblClient" runat = "server" Text = "Client" ClientIDMode = "Static" />
</asp:Content>
```

其中 Title 是浏览器中显示的页面标题;Language 是设置服务器代码编译为哪种语言;MasterPageFile 是 Web 页面使用的主页面文件(或者称母版页);AutoEventWireup 默认值为 True,如果设置为 False,则必须手动连接事件(比如页面加载事件),从而代码更复杂;CodeFile 当使用独立代码文件或代码隐藏页面时,它就是一个包含代码的页面;Inherits 仅指页面所继承的类名。单击 Client 按钮,使用脚本语言处理页面事件,不会再回调 Web 服务器,因为浏览器自身会处理,ClientIDMode 可以设置在脚本语言中使用的 ID 的行为。使用 Static 会迫使控件的 ID 成为客户 ID,使用 Static 时控件名称不要重复。单击 Server 按钮会将窗体提交给服务器。

5.3 了解 Web 托管

开发完网站的主要目的是要将网站发布出去,供用户浏览使用。本节主要介绍了网站怎么发布,怎么进行托管等一系列的步骤。

5.3.1　创建网站

要创建一个网站，可以执行如下操作步骤：

(1)先安装 IIS。请进入“控制面板”，依次选“程序和功能”→“启用或关闭 Windows 功能”，将“Internet Information Services”前的小钩去掉(如有)，重新勾选后按提示操作即可完成 IIS 的添加。这里由于系统的不同可能有少许的差异，这里的步骤都是以 Windows7 或者以上系统为例。

(2)进入“开始”|“控制面板”|“管理工具”|“Internet Information Services (IIS)管理器”，以打开 IIS 管理器，如图 5-3 所示。右击网站选择“添加网站”，出现图 5-4 的对话框，主要修改三个地方：“网站名称”“物理路径”“端口”。如果是局域网，还要修改“IP 地址”。例如：192.168.　.　，然后修改端口号，点击确定即可。

图 5-3　添加网站

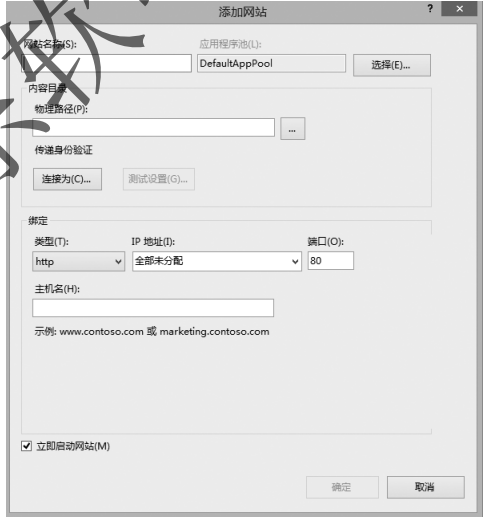

图 5-4　添加网站对话框

5.3.2 部署 Web 应用程序

部署是将应用程序的副本发布到其他计算机上的行为，使之能在新环境中运行。这是一个非常大的体系，也称为安装。但安装和部署存在细微的差别。部署是一个发布过程，也就是发布软件的方法，而安装则是加载、配置和安装软件的过程。换言之，安装就是如何配置软件，而部署是如何在需要的地方获取软件。

5.3.2.1 Web 应用程序部署的困扰

开发人员多半是在自己的实验与工作环境中开发与测试完成，再经过 Staging Server（转移服务器），就会发布到企业内实际上线使用。

如果开发人员会使用 Windows Installer，则会将流程封装为 MSI 文件，但这种方法仍然偏少，多半是在 Web 应用程序本身是产品时才会这样做。在企业内部若要部署以及转移 Web 应用程序不是件容易的工作。

Web 应用部署另一个重要工作是多台服务器之间的转移。散布在多台服务器上的 Web 应用那么多，而 Web 应用程序又有各自的设置要处理，仅是这部分工作就可能要耗费许多时间。

5.3.2.2 Web 应用程序部署的考量

静态的 Web 网页大多只有文件夹以及 HTML 文件和图片等，但大多数的 Web 应用需要再加上程序连接的数据库，同时应用的性质不同也会需要额外的设定，像是存取控制表、XML 或是专属的文件存储格式等。

以往的作法，大多数是由开发人员撰写安装说明（Installation Instruction），详述了应用所使用的文件夹结构，设定值（XML 或系统登录数据库）、引用的组件类型（如 COM）以及其他所需要额外要安装的工具或者组件等等。

微软为了解决系统管理人员的负担，在 IIS7.0 中研发了一个以规则为主（rule-based），以提供者来处理服务（provider-specific）的 Web 应用程序发布工具，并且在 IIS.NET 中发布，及在 IIS7.5 中内置，此工具称为 Web Deployment Tool。

5.3.2.3 Web Deploy 简介与架构

Web Deployment Tool 工具是一个可以让系统管理人员直接在 IIS7.0 中进行应用程序部署的工具集。它分为图形界面工具以及命令行工具两部分，图形界面工具与 IIS 管理员完全整合，而命令行工具则是由 msdeploy.exe 提供。它的功能有：

- 封装（Packaging）在 IIS 中现有的站点设置或 Web 应用程序，不论是网站或者是虚拟目录都可以。在封装的同时，还可以连带封装应用程序所属的各种设置，包含数据库和组件等等。
- 部署已封装好的应用程序，不但可以不需要管理员权限，同时也会自动与 IIS 的管理服务（Management Service）结合，以进一步处理安装。
- Web Deploy 可以在不同的服务器（目前支持 IIS6.0 和 7.X）间进行 Web 应用程序的同步化（Synchronization），同步的部分包含了应用程序本身、数据库以及各种设置（包括 SSL 以及 IIS 本身的配置），而且只有变更的部分会被处理。

Web Deploy 工具的原理是将 Web 应用程序的部署划分为数个不同的设置，由各自

的设置提供者(Configuration Provider)来执行自己的配置工作,而且配置设定间还可以套用相互关联,来控制条件以及处理必要控件的设置。

已安装 Web Deploy 工具的 IIS7 管理员可以在应用程序的管理中找到两个新的功能:

• 导出:导出功能会弹出一个导出功能对话框,系统管理人员(或开发人员)可以在这里加入 Web 应用程序所需要的设置值,包含数据库和组件。Web Deploy 工具内置了 27 种组件的配置功能,它允许开发人员自主配置组件,以支持其他有特殊需求的应用程序开发人员或厂商处理特殊的配置要求。

• 导入:每一个应用程序封装成一个 ZIP 文件。在这个文件中,包含了应用程序所有封装的必要文件,也包含了一些系统配置,这些设置会在封装时写入到封装文件中。由于必要的配置资料都已经包含在封装的 ZIP 文件里,所以系统管理人员只要将这个 ZIP 文件拿到另一台服务器上执行导入功能即可将网站部署起来。

5.3.3　状态管理

无论 Web 应用程序框架多么先进,它都不能改变这样一个事实——HTTP 是一种无状态协议。每次 Web 请求后,客户端和服务器端断开,同时 ASP.NET 引擎释放页面对象,这种框架结构保证了 Web 应用程序能够同时响应数千个并发请求而不会导致服务器内存崩溃。其负面效应是用户必须通过其他技术存储 Web 请求之间的信息,并在需要的时候获取它们。

ASP.NET 为状态管理提供了多种选择。可以根据要存储的数据、存储数据要花费的时间、数据的范围(不管它是由单个用户控制还是多个请求共享)以及其他安全性和性能条件选择适合的选项。ASP.NET 种不同的状态管理选项是互补的,即用户可以在同一个 Web 应用程序(同一个页面)中结合使用它们。

表 5-7、表 5-8 和表 5-9 对可用的状态管理作了简单对比。

表 5-7　状态管理选择对比(第一部分)

	视图状态	查询字符串	自定义 cookie
支持的数据类型	任何可序列化的.NET 数据类型	长度受限的字符串数据	字符串数据
存储位置	当前网页中的一个隐藏字段	浏览器的 URL 字符串	客户计算机(根据其生命周期的设置,它可以存在于内存中或一个小的文本文件中)
生命周期	在单个页面回发的过程中永久保持	在用户输入新的 URL 或关闭浏览器时丢失,但可以在多次访问中保持	由程序开发人员设置。可在多个页面中使用并可在多次访问中保持
范围	仅限于当前页面	仅限于目标页面	整个 ASO.NET 应用程序
安全性	默认为不安全,但用户可以通过 page 指令强制加密	明文且容易被用户修改	不安全且可被用户修改
对性能的影响	存储但不会影响服务器性能	无影响,因为数据量非常小	无影响,因为数据量非常小
典型应用	页面特定设置	将产品 ID 从类别页传送到细节页面	网站的个性化设置

表 5-8 状态管理选项对比(第二部分)

	配置文件	应用程序状态
支持的数据类型	所有可序列化的.NET 数据类型,如果使用默认的进程内状态服务,不可序列化的数据类型也被支持	所有.NET 数据类型
存储位置	依据所选择的模式,可以是服务器内存(默认值)或者专用的数据库	服务器内存
生命周期	一段时间后超时(通常是 20 分钟,可通过全局设置或代码修改)	应用程序的生命周期(通常是服务器重新启动)
范围	整个 ASP.NET 应用程序	整个 ASP.NET 应用程序。和大多数其他类型的方法不同的是,应用程序数据对所有用户通用
安全性	安全,因为数据从来没有被传到客户端。不过,如果不使用 SSL,会话可能会被劫持	非常安全,因为数据从来没有被传送到客户端
对性能的影响	储存大量数据会导致服务器性能下降,特别是在同时有大量用户时,因为每个用户都有独立的绘画数据副本	因为数据从来不会被移除或过期,存储大量数据会导致服务器性能下降
典型应用	在购物车中保存项目	储存全局数据

表 5-9 状态管理选项对比(第三部分)

	配置文件	缓存
支持的数据类型	所有可序列化的.NET 数据类型	所有.NET 数据类型。如果创建自定义的配置文件,那么支持不可序列化类型
储存位置	后端数据库	服务器内存
生命周期	永久	依赖于用户设置的过期策略。但在服务器内存紧张的情况下可能会提前释放
范围	整个 ASP.NET 应用程序,不可被其他应用程序访问	和应用程序的状态一致(对所有用户和所有页面可用)
安全性	相对安全,虽然数据从来没有被传送,但它存储在数据库中,而数据库安全可能会受到威胁	非常安全,因为数据从来没有被传送到客户端
对性能的影响	易于存储大量数据,但每次请求读写数据时会带来微量影响	存储大量的数据可能会挤占其他有用的缓存信息的空间。不过,ASP.NET 能够尽早移除项目以优化性能
典型应用	保存客户账户信息	保存从数据库获取的数据

显而易见,ASP.NET 中并不缺少管理状态的方法。幸好,大部分状态管理系统提供了类似的基于集合的编程接口。一个例外是配置文件特征,它为用户提供更高层次的数据模型。

5.3.4　认识 IIS 的角色

互联网信息服务(IIS,Internet Information Services),是由微软公司提供的基于运行 Microsoft Windows 的互联网基本服务。适用于 Windows Server 的 IIS 具有弹性、安全且容易管理的网页服务器,能够装载网络上的任何项目。无论是媒体串流处理或 Web 应用程式装载,IIS 的可扩充与开放式架构足以应付最高难度的工作。IIS 是以开放的模块化架构为基础,可让用户通过免费的 IIS 扩展功能自订和新增功能,灵活的弹性足以满足各种客户安全需求。

5.3.4.1　建立和设定 IIS 中的虚拟目录

要建立和设定 IIS 中的虚拟目录,可以执行如下操作步骤:

(1)用户可以使用 IIS 管理器建立 ASP.NET Web 应用程序的虚拟目录。进入“开始→控制面板→管理工具→Internet Information Services (IIS)管理器”以打开 IIS 管理器,对于有“已停止”字样的服务,均在其上单击右键,选“启动”来开启。

(2)对客户端浏览器而言,虚拟目录就好像包含在 Web 服务器的根目录中,即使实际上可能位于其他地方。这个方法能够让用户发布不在 Web 服务器根文件夹中的 Web 内容,例如位于远程电脑上的内容。对设定本机 Web 开发工作的站点来说,这个方法也很方便,因为不需要唯一的站点识别,所需要的步骤比建立唯一的站点要少。

(3)若要建立虚拟目录,Web 服务器上必须已建立网站。IIS 在安装期间会在电脑上建立预设的网站。如果用户尚未建立自己的网站,可以在预设的网站中建立虚拟目录。图 5-5 就是在默认网站(Default Web Site)上右击选择“添加虚拟目录”。

(4)用户可能想要建立使用唯一识别的新网站,当做建立 Web 应用程序虚拟目录的替代方案。

图 5-5　IIS 管理器上建立虚拟目录

5.3.4.2 IIS Database Manager

IIS Database Manager 具有如下作用：

• IIS Database Manager 可让用户从“IIS 管理员”轻松管理本机和远端数据库。

• IIS Database Manager 会根据网页服务器或应用程序设定，自动搜索数据库，并提供连线至网络上连接任一数据库的能力。

5.3.4.3 设定连级超时

连线等候超时，有助于减少因闲置连线、耗用处理资源而导致的损失。开启连线等候超时的时候，IIS 会在连线时强制等候超时。

5.4 了解 Web 服务和 WCF

本节会对 Web Service 和 WCF 做简单的介绍，并演示一个简单的 Web Service 的创建和运行调试的过程，以及对 Web Service 和 WCF 的引用方式做简单的介绍。

5.4.1 Web Service

Web Service 是一个平台独立的、低耦合的、自包含的、基于可编程的 Web 的应用程序，可使用开放的 XML(标准通用标记语言下的一个子集)标准来描述、发布、发现、协调和配置这些应用程序，用于开发分布式的互操作的应用程序。

它是一种构建应用程序的普遍模型，可以在任何支持网络通信的操作系统中实施运行；它是一种新的 Web 应用程序分支，是自包含、自描述、模块化的应用，可以发布、定位，通过 Web 调用。Web Service 是一个应用组件，它逻辑性地为其他应用程序提供数据与服务。各应用程序通过网络协议和规定的一些标准数据格式(Http，XML，Soap)来访问 Web Service，通过 Web Service 内部执行得到所需结果。Web Service 可以执行从简单的请求到复杂商务处理的任何功能。一旦部署以后，其他 Web Service 应用程序可以发现并调用它部署的服务。

Web 服务平台的元素有 XML、SOAP、WSDL、UDDI。

1. XML (可扩展标记语言)

我们在前面的章节中对 XML 做了详细的介绍，这里不再重复介绍。

2. SOAP

SOAP(Simple Object Access Protocol)即简单对象访问协议，它是用于交换 XML(标准通用标记语言下的一个子集)编码信息的轻量级协议。它有三个主要方面：XML-envelope 为描述信息内容和如何处理内容定义了框架，将程序对象编码成为 XML 对象的规则，执行远程过程调用(RPC)的约定。SOAP 可以运行在任何其他传输协议上。例如，用户可以使用 SMTP，即因特网电子邮件协议来传递 SOAP 消息，这可是很有诱惑力的。在传输层之间的报头是不同的，但 XML 有效负载保持相同。

Web Service 希望实现不同的系统之间能够用“软件-软件对话”的方式相互调用，打破了软件应用、网站和各种设备之间的格格不入的状态，实现“基于 Web 无缝集成”的目

标。

3. WSDL(Web Services Description Language)

Web 服务描述语言 WSDL 就是用机器能阅读的方式提供的一个正式描述文档而基于 XML(标准通用标记语言下的一个子集)的语言,用于描述 Web 服务及其函数、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。

4. UDDI(Universal Description, Discovery and Integration)

UDDI 的目的是为电子商务建立标准;UDDI 是一套基于 Web 的、分布式的、为 Web 服务提供的、信息注册中心的实现标准规范,同时也包含一组使企业能将自身提供的 Web 服务注册,以使别的企业能够发现的访问协议的实现标准。

实际上,Web 服务的主要目标是跨平台的可互操作性。为了达到这一目标,Web 服务完全基于 XML(可扩展标记语言)、XSD(XMLSchema)等独立于平台、独立于软件供应商的标准,是创建可互操作的、分布式应用程序的新平台。由此可以看出,使用 Web 服务会带来极大的好处。

Web 服务的主要目标是跨平台的可互操作性。为了实现这一目标,Web 服务完全基于 XML(可扩展标记语言)、XSD(XML Schema)等独立于平台、独立于软件供应商的标准,是创建可互操作的、分布式应用程序的新平台。

因此使用 Web 服务有以下优点:

1. 跨防火墙的通信

如果应用程序有成千上万的用户,而且分布在世界各地,那么客户端和服务器之间的通信将是一个棘手的问题。因为客户端和服务器之间通常会有防火墙或者代理服务器。传统的做法是选择用浏览器作为客户端,写下一大堆 ASP 页面,把应用程序的中间层暴露给最终用户。这样做的结果是开发难度大,程序很难维护。要是客户端代码不再如此依赖于 HTML 表单,客户端的编程就简单多了。如果中间层组件换成 Web 服务,就可以从用户界面直接调用中间层组件,从而省掉建立 ASP 页面的那一步。要调用 Web 服务,可以直接使用 Microsoft SOAP Toolkit 或. NET 这样的 SOAP 客户端,也可以使用自己开发的 SOAP 客户端,然后把它和应用程序连接起来。这不仅缩短了开发周期,还减少了代码复杂度,并能够增强应用程序的可维护性。同时,应用程序也不再需要在每次调用中间层组件时,都跳转到相应的"结果页"。

2. 应用程序集成

企业级的应用程序开发者都知道,企业经常都要把用不同语言写成的、在不同平台上运行的各种程序集成起来,而这种集成将花费很大的开发力量。应用程序经常需要从运行的一台主机上的程序中获取数据;或者把数据发送到主机或其他平台应用程序中去。即使在同一个平台上,不同软件厂商生产的各种软件也常常需要集成起来。通过 Web 服务,应用程序可以用标准的方法把功能和数据"暴露"出来,供其他应用程序使用。

XML Web 服务提供了在松耦合环境中使用标准协议(HTTP、XML、SOAP 和 WSDL)交换消息的能力。消息可以是结构化的、带类型的,也可以是松散定义的。

3. B2B 的集成

B2B(Business to Business)指的是商家(泛指企业)对商家的电子商务,即企业与企

业之间通过互联网进行产品、服务及信息的交换。通俗的说法是指进行电子商务交易的供需双方都是商家(或企业、公司),他们使用了 Internet 的技术或各种商务网络平台,完成商务交易的过程。

Web 服务是 B2B 集成成功的关键。通过 Web 服务,公司可以只需把关键的商务应用"暴露"给指定的供应商和客户,就可以了。Web 服务运行在 Internet 上,在世界任何地方都可轻易实现,其运行成本就相对较低。Web 服务只是 B2B 集成的一个关键部分,还需要许多其他的部分才能实现集成。用 Web 服务来实现 B2B 集成的最大好处在于可以轻易实现互操作性。只要把商务逻辑"暴露"出来,成为 Web 服务,就可以让任何指定的合作伙伴调用这些商务逻辑,而不管他们的系统在什么平台上运行,使用什么开发语言。这样就大大减少了花在 B2B 集成上的时间和成本。

4. 软件和数据重用

Web 服务在允许重用代码的同时,可以重用代码背后的数据。使用 Web 服务,再也不必像以前那样,要先从第三方购买、安装软件组件,再从应用程序中调用这些组件;只需要直接调用远端的 Web Service 就可以了。另一种软件重用的情况是,把好几个应用程序的功能集成起来,通过 Web Service "暴露"出来,就可以非常容易地把所有这些功能都集成到用户的门户站点中,为用户提供一个统一的、友好的界面。可以在应用程序中使用第三方的 Web 服务 提供的功能,也可以把自己的应用程序功能通过 Web 服务提供给别人。两种情况下,都可以重用代码和代码背后的数据。

从以上论述可以看出,Web 服务在通过 Web 进行互操作或远程调用的时候是最有用的。不过,也有一些情况,Web 服务根本不能带来任何好处,Web 服务有以下缺点:

1. 单机应用程序

目前,企业和个人还使用着很多桌面应用程序。其中一些只需要与本机上的其他程序通信。在这种情况下,最好就不要用 Web 服务,只要用本地的 API 就可以了。COM 非常适合于在这种情况下工作,因为它既小又快。运行在同一台服务器上的服务器软件也是这样。当然 Web 服务也能用在这些场合,但那样不仅消耗太大,而且不会带来任何好处。

2. 局域网的一些应用程序

在许多应用中,所有的程序都是在 Windows 平台下使用 COM,都运行在同一个局域网上。在这些程序里,使用 DCOM 会比 SOAP/HTTP 有效得多。与此相类似,如果一个.NET 程序要连接到局域网上的另一个.NET 程序,应该使用.NET Remoting。其实在.NET Remoting 中,也可以指定使用 SOAP/HTTP 来进行 Web 服务调用。不过最好还是直接通过 TCP 进行 RPC 调用,那样会有效得多。

5.4.2 客户端应用程序如何调用 Web 服务

在这个示例中,选择创建一个新的 Web 项目(在以前章节中,创建的是无项目文件的网站),可以按照下列步骤操作。

(1)选择"文件"|"新建"|"项目"。

(2)在弹出的"新建项目"对话框中,请选择"Web" 项目类型。

(3)在“.NET Framework”列表里,选择“.NET Framework 3.5”。

(4)在项目模版中选择“ASP.NET Web 服务应用程序”(“.NET Framework 4”和“.NET Framework 4.5”不提供这个模版)。

(5)在“名称”中输入“FirstService”。

(6)单击“确定”按钮,FirstService 项目添加完成。

打开“Service1.asmx”,可以知道这个服务的名称是“Service1”,它已经提供了一个 HelloWorld 的 Web 服务方法。

```
public class Service1 : System.Web.Services.WebService
{

    [WebMethod]
    public string HelloWorld()
    {
        return "Hello World";
    }
}
```

```
VB.NET:
Public Class Service1
    Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function HelloWorld() As String
       Return "Hello World"
    End Function
End Class
```

```
VB.NET:
Public Class Service1
    Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function SayHello(stringName As String) As String
        Return Getting(stringName)
    End Function
    Public Function Getting(stringName As String) As String
        Return "Hello," + stringName
    End Function
End Class
```

下面添加 SayHello 和 Greeting 方法并且在网页中测试。

```
public class Service1 : System.Web.Services.WebService
{
    [WebMethod]
```

```
    public string SayHello(string stringName)
    {
        return Geeting(stringName);
    }
     public string Geeting(string stringName)
     {
         return "Hello," + stringName;
     }
}
```

将 HelloWorld 这个 Web 服务方法删除，新增 SayHello 和 Greeting 这两个方法。

• SayHello 会调用 Greeting，传递 stringName 参数，并且返回执行结果。

• 由于 Greeting 没有 WebMethod 属性，因此无法在 Web 服务中被使用，如果想让 Greeting 方法能够在 Web 服务中被使用，只要在 Greeting 方法前面加上 WebMethod 属性即可。

右键单击选择 Service1. asmx，在弹出的右键菜单中选择“在浏览器中查看”，可以看到服务名称 Service1 提供的方法 SayHello。如图 5-6 所示。

图 5-6　Service1. asmx

这个网址最主要的目的是给出服务引用，提供其他程序调用，待会将知道如何使用这个服务，接下来测试刚刚写的程序是否正确。

在 SayHello 方法中需要传递参数给 Web Service，可以在“值”的文本框中，输入 stringName 的参数值。在这里输入 Test，并且单击“调用”按钮。如图 5-7 所示。

此时会回传字符串类型的参数“Hello，Test”，符合当初预期的执行结果。如图 5-8 所示。

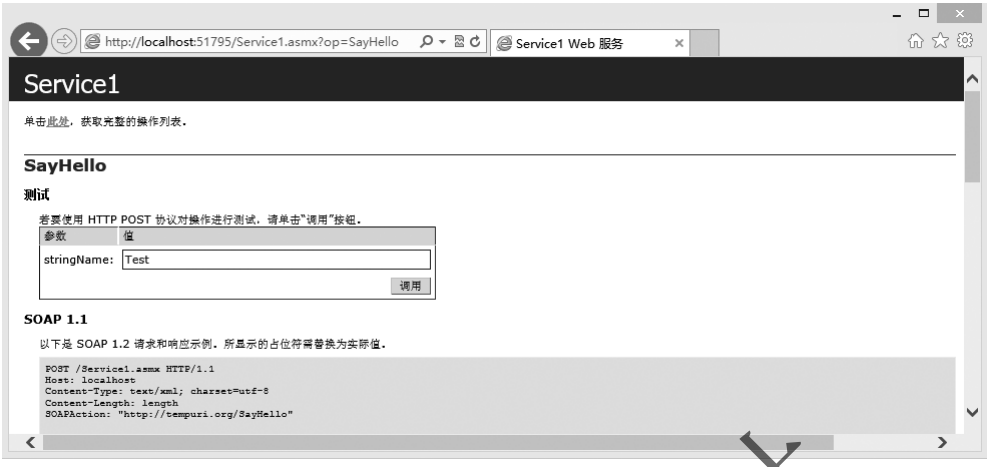

图 5-7　测试

图 5-8　执行结果

在现有的解决方案中，添加 CallHelloService 项目，可以按照下面步骤操作：

(1)选择“文件”→“新建”→“项目”。

(2)在弹出的“新建项目”对话框中，选择“Web”项目类型。

(3)在“.NET Framework”列表里，选择“.NET Framework 4.5”。

(4)在项目模版中选择“ASP.NET 空 Web 应用程序”。

(5)在“解决方案”下拉框中，选择“添加到解决方案”。

(6)在“名称”文本框中输入“CallFirstService”。

(7)单击“确定”按钮，CallHelloService 项目添加完成。

选择 CallFirstSercie 项目名称，右键单击，在弹出的右键菜单中选择“添加服务引用”。在弹出的对话框中单击“发现”按钮，如图 5-9 所示。

图 5-9　添加服务引用

此时在服务窗口中会列出这个解决方案中包含哪些服务，目前的解决方案中只包含 Service1 这个服务。设置服务引用名称为 ServiceReference1，最后单击"确定"按钮。

这时会发现，在 CallFirstService 项目中新增了 Web References 文件夹，文件夹内可以看到刚刚添加的服务引用"ServiceReference1"。如图 5-10 所示。

图 5-10　新增的 Service References 文件夹

在 CallFirstService 项目中的 Default. aspx 文件中加入以下内容。

```
<asp:TextBox ID = "tb_name" runat = "server" Width = "200"></asp:TextBox>
<asp:Button ID = "btn_hello" runat = "server" Width = "60" Text = "Hello" OnClick = "btn_hello_Click" />
<br /><br />
<asp:Label ID = "lbl_hello" runat = "server"></asp:Label>
```

给按钮控件添加 Click 事件，并且添加以下事件处理程序。

C#：

```
protected void btn_hello_Click(object sender, EventArgs e)
{
        ServiceReference1. Service1SoapClient firstservice = new ServiceReference1.
```

```
Service1SoapClient();
            string name = tb_name.Text.ToString();
            lbl_hello.Text = firstservice.SayHello(name);
        }
```

VB.NET：

```
Protected Sub btn_hello_Click(sender As Object, e As EventArgs)
        Dim firstservice As ServiceReference1.Service1SoapClient
        firstservice = New ServiceReference1.Service1SoapClient()
        Dim name As String = tb_name.Text.ToString()
        lbl_hello.Text = firstservice.SayHello (name)
End Sub
```

在单击按钮后，将文本框的值作为 SayHello 的参数值，并且将执行的结果赋值给 lbl_hello 上。如图 5-11 所示。

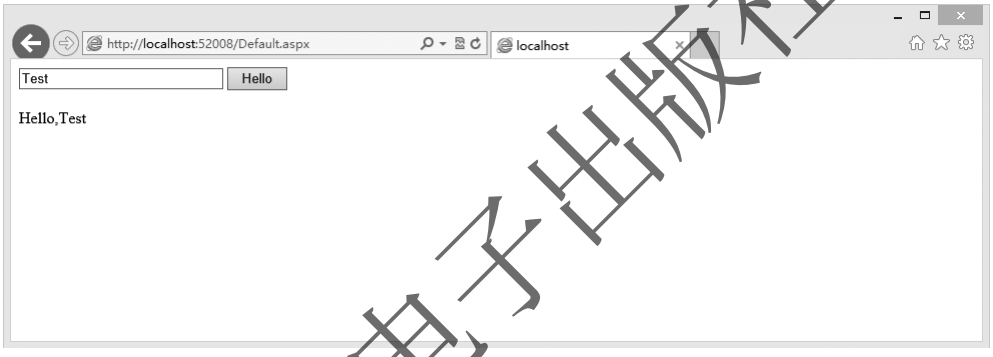

图 5-11　Web Service 示例

5.4.3　WCF 简介

WCF(Windows Communication Foundation)是由微软发展的一组数据通信的应用程序开发接口，可以翻译为 Windows 通讯接口。

它是.NET 框架的一部分，由 .NET Framework 3.0 开始引入，与 Windows Presentation Foundation 及 Windows Workflow Foundation 并行为新一代 Windows 操作系统以及 WinFX 的三个重大应用程序开发类库。在 .NET Framework 2.0 及以前版本中，微软发展了 Web Service(SOAP with HTTP communication)，.NET Remoting (TCP/HTTP/Pipeline communication) 以及基础的 Winsock 等通信支持。由于各个通信方法的设计方法不同，而且彼此之间也有相互的重叠性(例如 .NET Remoting 可以开发 SOAP，HTTP 通信)，对于开发人员来说，不同的选择会有不同的程序设计模型，而且必须要重新学习，让开发人员在使用中有许多不便。同时，面向服务架构(Service-Oriented Architecture) 也开始盛行于软件工业中，因此微软重新查看了这些通信方法，并设计了一个统一的程序开发模型，对于数据通信提供了最基本、最有弹性的支持，这就是 WCF。

WCF 由于集合了几乎由.NET Framework 所提供的通信方法，因此学习曲线比较

陡峭,开发人员必须要针对各个部份的内涵做深入的了解,才能够操控 WCF 来开发应用程序。

通信双方的沟通方式由合约来订定。通信双方所遵循的通信方法,由协议绑定来订定。通信期间的安全性,由双方约定的安全性层次来订定。

WCF 的基本概念是以契约(Contract)来定义双方沟通的协议,合约必须要以接口的方式来体现,而实际的服务代码必须要由这些合约接口派生并实现。合约分成了四种:

- 数据契约(Data Contract),订定双方沟通时的数据格式。
- 服务契约(Service Contract),订定服务的定义。
- 操作契约(Operation Contract),订定服务提供的方法。
- 消息契约(Message Contract),订定在通信期间改写消息内容的规范。

一个 WCF 中的契约,就如同下列代码所示:

C#:

```
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;namespace WcfService1
{
    [ServiceContract]
    public interface IService1
    {
        [OperationContract]
        string GetData(int value);

        [OperationContract]
        CompositeType GetDataUsingDataContract(CompositeType composite);
}
}
```

VB.NET:

```
Imports System
Imports System.Collections.Generic
Imports System.Runtime.Serialization
Imports System.ServiceModel
Imports System.ServiceModel.Web
Namespace WcfService1
    <ServiceContract> _
     Public Interface IService1

    <OperationContract> _
    Function GetData(value As Integer) As String
```

```
    <OperationContract> _
    Function GetDataUsingDataContract(composite As CompositeType) As CompositeType

  End Interface
End Namespace
```

由于 WCF 支持了 HTTP，TCP，Named Pipe，MSMQ，Peer-To-Peer TCP 等协议，而 HTTP 又分为基本 HTTP 支持（BasicHttpBinding）以及 WS-HTTP 支持（WsHttpBinding），而 TCP 亦支持 NetTcpBinding，NetPeerTcpBinding 等通信方式，因此，双方必须要统一通信的协议，并且也要在编码以及格式上要有所一致。

一个设置通信协议绑定的示例如下：

```
<? xml version = "1.0" encoding = "utf-8" ? >
<configuration>
<system.serviceModel>
<! - - 设定服务系结的资讯 - - >
<services>
<service name = " CalculatorService" >
<endpoint address = "" binding = " wsHttpBinding" bindingConfiguration = " Binding1"
contract = "ICalculator" />
</service>
</services>
<! - - 设定通讯协定系结的资讯 - - >
<bindings>
<wsHttpBinding>
<binding name = "Binding1">
</binding>
</wsHttpBinding>
</bindings>
</system.serviceModel>
</configuration>
```

虽然 WCF 也可以使用 SOAP 做通信格式，但它和以往的 ASP.NETXML Web Services 不同，因此有部分技术文章中，会将 ASP.NET 的 XML Web Services 称为 ASMX Service。

WCF 的服务可以挂载于 Console Application，Windows Application，IIS（ASP.NET）Application，Windows Service 以及 Windows Activation Services 中，但大多都会挂在 Windows Service。

从功能的角度来看，WCF 完全可以看作是 ASMX、.Net Remoting、Enterprise Service、WSE，MSMQ 等技术的并集。（注：这种说法仅仅是从功能的角度。事实上 WCF 远非简单的并集这样简单，它是真正面向服务的产品，它已经改变了通常的开发模式。）因此，对于上述汽车预约服务系统的例子，利用 WCF，就可以解决包括安全、可信赖、互操作、跨平台通信等需求。开发者不用再去分别了解.Net Remoting，ASMX 等各种技

术了。

概括地说，WCF 具有如下的优势：

1. 统一性

前面已经叙述，WCF 是对于 ASMX、.Net Remoting、Enterprise Service、WSE、MSMQ 等技术的整合。由于 WCF 完全是由托管代码编写，因此开发 WCF 的应用程序与开发其他的.NET 应用程序没有太大的区别，我们仍然可以像创建面向对象的应用程序那样，利用 WCF 来创建面向服务的应用程序。

2. 互操作性

由于 WCF 最基本的通信机制是 SOAP(Simple Object Access Protocol，简易对象访问协议)，这就保证了系统之间的互操作性，即使是运行不同的上下文中。这种通信可以是基于.NET 到.NET 间的通信。可以跨进程、跨机器甚至于跨平台的通信，只要支持标准的 Web 服务，例如 J2EE 应用服务器(如 WebSphere，WebLogic)。应用程序可以运行在 Windows 操作系统下，也可以运行在其他的操作系统下，如 Sun Solaris、HP Unix、Linux 等。

3. 安全与可信赖

WS-Security，WS-Trust 和 WS-SecureConversation 均被添加到 SOAP 消息中，以用于用户认证、数据完整性验证、数据隐私等多种安全因素。

在 SOAP 的 header 中增加了 WS-ReliableMessaging 允许可信赖的端对端通信。而建立在 WS-Coordination 和 WS-AtomicTransaction 之上的基于 SOAP 格式交换的信息，则支持两阶段的事务提交(two-phase commit transactions)。

上述的多种 WS-Policy 在 WCF 中都给与了支持。对于 Messaging 而言，SOAP 是 Web 服务的基本协议，它包含了消息头(header)和消息体(body)。在消息头中，定义了 WS-Addressing 用于定位 SOAP 消息的地址信息，同时还包含了 MTOM(消息传输优化机制，Message Transmission Optimization Mechanism)。

4. 兼容性

WCF 充分地考虑到了与旧有系统的兼容性。安装 WCF 并不会影响原有的技术如 ASMX 和.NET Remoting。即使对于 WCF 和 ASMX 而言，虽然两者都使用了 SOAP，但基于 WCF 开发的应用程序，仍然可以直接与 ASMX 进行交互。

下面我们来讨论一下 Web Service 和 WCF 到底有什么区别。

WCF 其实一定程度上就是 ASP.NET Web Service，因为它支持 Web 服务的行业标准和核心协议，因此 ASP.NET Web Service 和 WSE 能做的事情，它几乎都能胜任，跨平台和语言更不是问题(数据也支持 XML 格式化，而且提供了自己的格式化器)。

但是 WCF 作为微软主推的一个通讯组件或者平台，它的目标不仅仅是在支持和集成 Web 服务，因为它还兼容和具备了微软早期很多技术的特性。

根据微软官方的解释，WCF(之前的版本名为“Indigo”)是使用托管代码建立和运行面向服务(Service Oriented)应用程序的统一框架。它使得开发者能够建立一个跨平台的安全、可信赖、事务性的解决方案，且能与已有系统兼容协作。WCF 是微软分布式应用程序开发的集大成者，它整合了.NET 平台下所有的和分布式系统有关的技术，如

Enterprise Sevices(COM+). NET Remoting、Web Service(ASMX)、WSE3.0 和 MSMQ 消息队列。以通信(Communiation)范围而论,它可以跨进程、跨机器、跨子网、企业网乃至于 Internet;以宿主程序而论,可以以 ASP. NET,EXE,WPF,Windows Forms,NT Service,COM+作为宿主(Host)。WCF 可以支持的协议包括 TCP,HTTP,跨进程以及自定义,安全模式则包括 SAML, Kerberos,X509,用户/密码,自定义等多种标准与模式。也就是说,在 WCF 框架下,开发基于 SOA 的分布式系统变得容易了,微软将所有与此相关的技术要素都包含在内,掌握了 WCF,就相当于掌握了叩开 SOA 大门的钥匙。如果用户想对 WCF 的学习系统化一些,必须向对如 Enterprise Sevices(COM+). NET Remoting、Web Service(ASMX)、WSE3.0 和 MSMQ 消息队列也有个详细的认识。

5.4.4　WCF 引用

下面在上节的解决方案中,添加 WcfService1 项目,如图 5-12 所示:

(1)选择“文件”|“新建”|“项目”。

(2)在弹出的“新建项目”对话框中,选择“WCF” 项目类型。

(3)在“. NET Framework”列表里,选择“. NET Framework 4.5”。

(4)在项目模版中选择“WCF 服务应用程序”。

(5)在“名称”文本框中请输入“WcfService1”。

(6)单击“确定”按钮,WcfService1 项目添加完成。

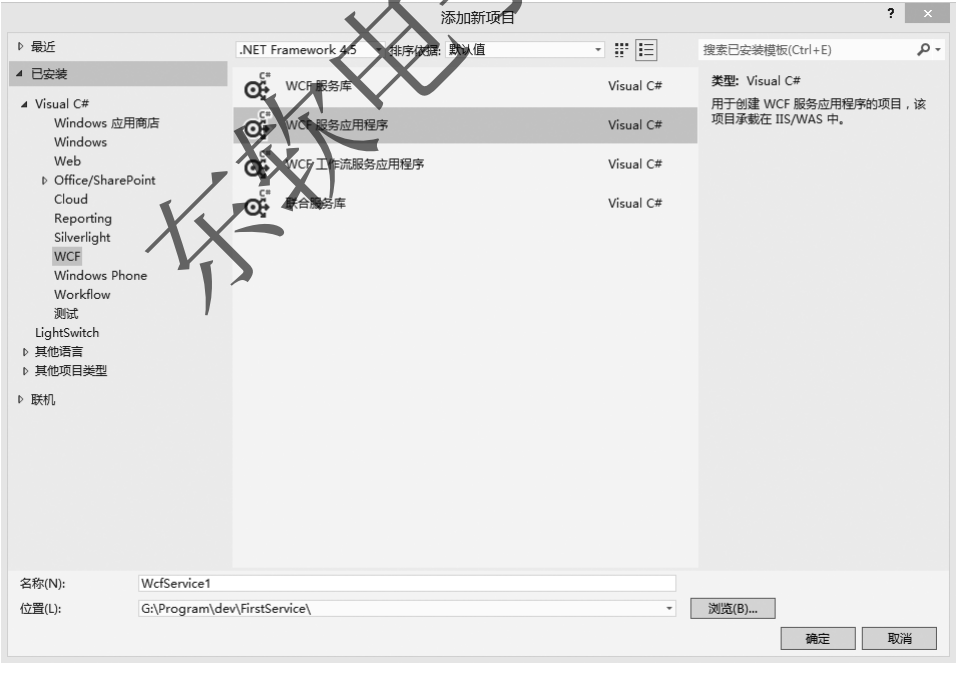

图 5-12　添加 WCF 服务应用程序

下面做一个简单的示例，在 WcfService1 项目中可以看到，有 IService1.cs 和 Service1.svc 这两个文件，分别在这两个文件中添加一些内容，代码如下所示：

C#：

IService1.cs：

```
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;

namespace WcfService1
{
    [ServiceContract]
    public interface IService1
    {
        [OperationContract]
        string GetData(int value);

        [OperationContract]
        CompositeType GetDataUsingDataContract(CompositeType composite);

        [OperationContract]
        int GetInt(int a, int b);
    }

    [DataContract]
    public class CompositeType
    {
        bool boolValue = true;
        string stringValue = "Hello ";

        [DataMember]
        public bool BoolValue
        {
            get { return boolValue; }
            set { boolValue = value; }
        }

        [DataMember]
        public string StringValue
        {
```

```
            get { return stringValue; }
            set { stringValue = value; }
        }
    }
}
```

Service1.svc.cs：

```
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;

namespace WcfService1
{
    public class Service1 : IService1
    {
        public string GetData(int value)
        {
            return string.Format("You entered: {0}", value);
        }

        public int GetInt(int a, int b)
        {
            return a + b;
        }

        public CompositeType GetDataUsingDataContract(CompositeType composite)
        {
            if (composite == null)
            {
                throw new ArgumentNullException("composite");
            }
            if (composite.BoolValue)
            {
                composite.StringValue += "Suffix";
            }
            return composite;
        }
    }
}
```

VB.NET：

IService1.vb：

```
Imports System.ServiceModel
'注意：使用上下文菜单上的“重命名”命令可以同时更改代码和配置文件中的接口名“IWcfService1”。
<ServiceContract()>
Public Interface IWcfService1
    <OperationContract()>
    Sub DoWork()

    <OperationContract()>
    Function GetInt(a As Integer, b As Integer) As Integer
End Interface
Service1.svc.vb：
Public Class WcfService1
    Implements IWcfService1
    Public Sub DoWork() Implements IWcfService1.DoWork
    End Sub
    Public Function GetInt(a As Integer, b As Integer) As Integer Implements IWcfService1.GetInt
        Return a + b
    End Function
End Class
```

选择 Service1. svc，右键单击，在弹出的右键菜单中选择“在浏览器中查看”，可以看到已创建服务。如图 5-13 所示。

图 5-13　Service1. svc

选择 CallFirstSercie 项目名称，右键单击，在弹出的右键菜单中选择“添加服务引

用”。在弹出的对话框中单击“发现”按钮，结果如图 5-14 所示。

图 5-14　添加服务引用

选择 Service1.svc 这个服务，设置服务引用名称为 wcfservice，最后单击“确定”按钮。

下面在 CallHelloService 项目中添加一个 Web 窗体，并命名为 wcftest。并添加以下内容：

wcftest.aspx：

```
<asp:TextBox ID = "tb_a" runat = "server"></asp:TextBox>
<label> + </label>
<asp:TextBox ID = "tb_b" runat = "server"></asp:TextBox>
<label> = </label>
<asp:TextBox ID = "tb_add" runat = "server"></asp:TextBox>
<asp:Button ID = "btn_add" runat = "server" Text = "ADD" OnClick = "btn_add_Click"/>
```

C#：

wcftest.aspx.cs：

```
protected void btn_add_Click(object sender, EventArgs e)
{
      wcfservice.Service1Client wcfs = new wcfservice.Service1Client();
      int a = int.Parse(tb_a.Text.ToString());
      int b = int.Parse(tb_b.Text.ToString());
      tb_add.Text = wcfs.GetInt(a,b).ToString();
}
```

VB.NET：

wcftest.aspx.vb：

```
Protected Sub btn_add_Click(sender As Object, e As EventArgs)
      Dim wcfs As ServiceReference2.WcfService1Client
      wcfs = New ServiceReference2.WcfService1Client
      Dim a As Integer = CType(tb_a.Text.ToString(), Integer)
      Dim b As Integer = CType(tb_b.Text.ToString(), Integer)
```

```
        tb_add.Text = wcfs.GetInt(a, b).ToString()
    End Sub
```

在浏览器中打开 wcftest.aspx,在 tb_a 和 tb_b 文本框中输入值,单击 ADD 按钮,tb_add 文本框中显示出刚输入的数值的和。如图 5-15 所示。

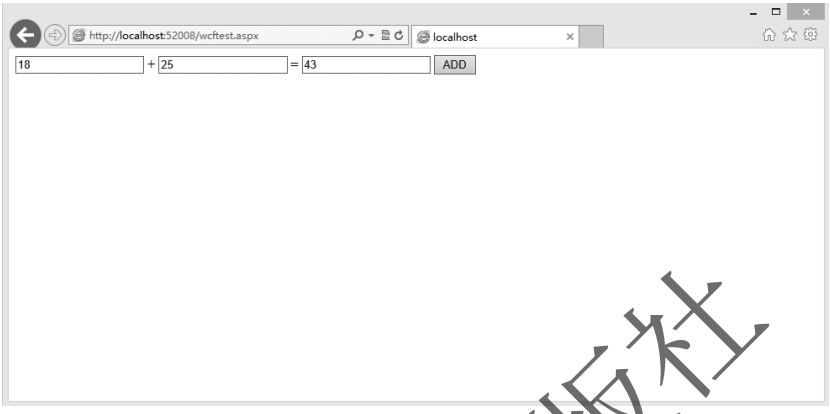

图 5-15 WCF 服务示例

5.4.5 SOAP 和 Web 服务定义语言(WSDL)

本小节将介绍关于 SOAP 和 Web 服务定义语言(WSDL)的相关知识。

5.4.5.1 SOAP

简单对象访问协议(SOAP)是交换数据的一种协议规范,是一种轻量的、简单的、基于 XML(标准通用标记语言下的一个子集)的协议,它被设计成在 WEB 上交换结构化的和固化的信息。SOAP 可以和现存的许多因特网协议和格式结合使用,包括超文本传输协议(HTTP)、简单邮件传输协议(SMTP)、多用途网际邮件扩充协议(MIME)。它还支持从消息系统到远程过程调用(RPC)等大量的应用程序。

(1)简单对象访问协议(SOAP,Simple Object Access Protocol)是一种标准化的通讯规范,主要用于 Web 服务(Web Service)中。

(2)SOAP 的出现是为了简化网页服务器在从 XML 资料库中提取资料时,无需花时间去格式化页面,并能够让不同应用程序通过 HTTP 通讯协定,以 XML 格式互相交互彼此的资料,使其与程序语言、平台和硬件无关。

(3)SOAP 讯息可以发送到一个具有 Web 服务功能的 Web 站点。例如一个含有价格资讯的数据库,参数中标明这是一个查询讯息,将返回一个 XML 格式的资讯,其中包括查询结果。

(4)资料标准化后用可分析的结构来传递,所以可以直接被第三方的使用者利用。

5.4.5.2 SOAP 消息格式

SOAP 在标准化消息格式环境中,可以做所有它能完成的工作。消息的主体部分是"text/xml"形式的 MIME 类型,并且包含一个 SOAP 封套。该封套是一个 XML 文档。封套包含了报头(可选的)和报文(必须有的)。封套的报文部分总是用于最终接收的消息,而报头项目可以确定执行中间处理的目标节点。附件、二进制数字及其他项目可以附

加到报文上。

SOAP 提供了一种让客户端指定哪个中间处理节点必须处理报头项目的方法。由于报头与 SOAP 消息的主体内容是互不相关的，所以可用它们给消息添加信息，而不会影响对消息报文的处理。

例如，报头可用于为报文中包含的请求提供数字签名。在这种情形下，身份验证/授权服务器可以使报头项目独立于报文，可以剥离信息以验证签名。一旦通过验证，封套的其余部分将被传递给 SOAP 服务器，它将对消息的报文进行处理。深入研究一下 SOAP 封套，有助于明了 SOAP 报头和报文元素的位置和用途。

SOAP 的消息格式：

```
<SOAP-ENV:Envelope 各种属性 = "">
  <! - - 示例 - - >
  <SOAP:HEADER>

  </SOAP:HEADER>
  <SOAP:Body>

  </SOAP:Body>
</SOAP - ENV:Envelope>
```

5.4.5.3　SOAP 的语法规则

1. 构建模块

一条 SOAP 消息就是一个普通的 XML 文档，包含下列元素：

- 必需的 Envelope 元素，可把此 XML 文档标识为一条 SOAP 消息。
- 可选的 Header 元素，包含头部信息。
- 必需的 Body 元素，包含所有的调用和响应信息。
- 可选的 Fault 元素，提供有关在处理此消息所发生错误的信息。

2. 语法规则

这里是一些重要的语法规则：

- SOAP 消息必须用 XML 来编码
- SOAP 消息必须使用 SOAP Envelope 命名空间
- SOAP 消息必须使用 SOAP Encoding 命名空间
- SOAP 消息不能包含 DTD 引用
- SOAP 消息不能包含 XML 处理指令

3. SOAP 的基本消息结构

```
<? xml version = "1.0" encoding = "utf - 8" ? >
< soap: Envelope xmlns: soap = " http://www. w3. org/2001/12/soap - envelope" soap:
encodingStyle = "http://www.w3.org/2001/12/soap - encoding">
  <soap:Header>
    <! - - 示例 - - >
  </soap:Header>
```

```
<soap:Body>
  <! --示例-->
  <soap:Fault>
    <! --示例-->
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

5.4.5.4 SOAP 用例

Internet 上某些地方的客户端应用程序使用 Web 服务，就是通过 SOAP 显示对象。对象无法访问 Web 上任意位置的远程数据。对这些网络命题应用传递逻辑，我们可以为 Web 服务和 SOAP 下一个总的结论：某些位置的客户端可以使用 Web 上任意位置的数据。这就是所要证明的。

下面是更加详细一点的用例。

(1)SOAP 客户端使用 UDDI 注册来查找 Web 服务。不用直接操作 WSDL，大多数情况下 SOAP 应用程序将硬连接到使用特定类型的端口和特定样式的绑定，并且它将通过 UDDI 动态配置要调用的、与发现的 Web 服务匹配的服务地址。

(2)客户端应用程序创建 SOAP 消息，它是一个可执行想要的请求/响应操作的 XML 文档。

(3)客户端把 SOAP 消息传送给监听 SOAP 请求的 Web 服务器上的 JSP 或 ASP 页面。

(4)SOAP 服务器解析 SOAP 包并在其领域调用合适的对象方法，在 SOAP 文档中包 含的参数中传递。在 SOAP 服务器接收消息之前，中间处理节点可以执行 SOAP 报头指示的特殊功能，可视情况确定是否执行这步操作。

(5)请求对象执行指示的功能，并返回数据给 SOAP 服务器，它把响应打包到 SOAP 封套中。服务器把 SOAP 封套包裹在要发送回请求机器的响应对象中，如 servlet 或 COM 对象。

(6)客户端接收对象，剥离出 SOAP 封套并把响应文档发送给最初发出请求的程序，完成请求/响应循环。

总之，SOAP 是一种基于 XML 的协议，它用于在分布式环境中发送消息，并执行远程过程调用。使用 SOAP，不用考虑任何特定的传输协议(尽管通常选用 HTTP 协议)，就能使数据序列化。用 SOAP 来构建平台与语言中性的互操作系统是一个好的选择。总之，SOAP 和 Web 服务已为在 XML 上构建分布式应用程序基础结构所需的一切都考虑好了。

通过解决 COM 和 Java 组件对象模型之间的冲突，SOAP 把多个平台在访问数据时所出现的不兼容性问题减至最少。先把这些讨论放在一边，SOAP 是一种适用于所有类型的对象实体的理想的媒介，即使对于像 Brad Pitt 和 Edward Norton 之类的好莱坞电影角色，也可用作一种通信媒介。就像在电影中一样，期待着这种新技术带来震撼世界的效果。

5.4.6　WSDL

WSDL(Web 服务描述语言,Web Services Description Language)是为描述 Web 服务发布的 XML 格式。W3C 组织(World Wide Web Consortium)没有批准 1.1 版的 WSDL,当前的 WSDL 版本是 2.0,是 W3C 的推荐标准(recommendation)(一种官方标准),并将被 W3C 组织批准为正式标准。在诸多技术文献中通常将 Web 服务描述语言简写为 WSDL。

WSDL 描述 Web 服务的公共接口。这是一个基于 XML 的关于如何与 Web 服务通讯和使用的服务描述;就是描述与目录中列出的 Web 服务进行交互时需要绑定的协议和信息格式。通常采用抽象语言描述该服务支持的操作和信息,使用的时候再将实际的网络协议和信息格式绑定给该服务。

5.4.6.1　WSDL 的基本元素

WSDL 元素采用 XML 语法来描述与服务进行交互的基本元素:

- Type(消息类型):数据类型定义的容器,它使用某种类型系统(如 XSD)。
- Message(消息):通信数据的抽象类型化定义,它由一个或者多个 part 组成。
- Part:消息参数。
- Operation(操作):对服务所支持的操作进行抽象描述,WSDL 定义了四种操作:

(1)单向(one-way):端点接受信息;

(2)请求-响应(request-response):端点接受消息,然后发送相关消息;

(3)要求-响应(solicit-response):端点发送消息,然后接受相关消息;

(4)通知(notification):端点发送消息。

- Port Type (端口类型):特定端口类型的具体协议和数据格式规范。
- Binding:特定端口类型的具体协议和数据格式规范。
- Port:定义为绑定和网络地址组合的单个端点。
- Service:相关端口的集合,包括其关联的接口、操作、消息等。

5.4.6.2　WSDL 文档结构

WSDL 文档是利用这些主要的元素来描述某个 Web 服务,表 5-10 列出了组成 WSDL 的各个元素。

表 5-10　　WSDL 文档的元素

元素	定义
definitions	WSDL 文档的根元素,该元素的属性指明了 WSDL 文档的名称、文档的目标名字空间以及 WSDL 文档应用的名字空间的速记定义
types	Web 服务使用的数据类型
portType	Web 服务执行的操作
operation	一个服务包含的操作的描述,当操作被调用时,操作被定义为两个 endpoint 之间的消息传递
binding	Web 服务使用的通信协议
service	相关 port 元素的集合,这些元素被存储,用户组织 endpoint 定义
port	通过 binding 和物理地址定义的 endpoint,这个元素将所有抽象定义聚集在一起

一个 WSDL 文档的主要结构是类似这样的：

```
<definitions>

<types>
  definition of types.....
</types>

<message>
  definition of a message....
</message>

<portType>
  definition of a port.....
</portType>

<binding>
  definition of a binding....
</binding>

</definitions>
```

WSDL 文档可包含其他的元素，比如 extension 元素以及一个 service 元素，此元素可把若干个 Web 服务的定义组合在一个单一的 WSDL 文档中。

设计一个 WSDL 的步骤如下：

- 定义服务用到的 data types；
- 定义服务用到的消息；
- 定义服务接口；
- 定义消息与接口之间的 bindings 和线上数据的具体呈现方式；
- 定义每个服务的传输细节。

5.4.6.3 设计 WSDL

1. 定义逻辑数据单元

(1)将数据拆分成逻辑单元

这些单元能被映射为数据类型，并被服务的物理实现所引用。如果用户定义了一个服务接口，并且该接口已经实现，用户必须将实现操作的数据类型转换成 XML 元素，用于组装成消息。如果用户从头开始，用户必须定义用户的消息构建时用到的构建块，这样从实施角度看才有意义。

定义服务数据单元可用的类型体系。根据 WSDL 规范，用户可以使用任何类型体系。然而在 W3C 规范中定义的 XML schema 是首选的规范的类型体系。因此，XML 架构是在 Apache CXF 的内在类型系统。

XML schema 被用来定义一个 XML 文档如何构建，用于定义一个文档由哪些元素组成。这些元素可以使用 XML schema 类型，比如 xsd:int 的或者他可以使用用户定义

的类型，用户定义类型也是使用 XML 元素的组合来构建的，或者他们是用严格的已存在类型构建。通过结合类型定义和元素定义，用户可以创建复杂的 XML 文档，可以包含复杂的数据。当定义服务使用的数据单元时，用户可以定义它们作为类型，这些类型指明消息组成部分的结构，用户也能够定义用户的消息单元作为组成消息结构的元素。

生成数据单元的考虑。用户可以考虑简单地生成逻辑数据单元，这些单元直接映射用户在服务中使用的数据类型。当以这个方式工作时，必须紧密遵循构建 RPC 类型应用程序的构建模型，这不是构建面向服务架构程序必需的策略。Web 服务组织提供了一定数量的手册来定义数据单元，另外 W3C 也提供了下 main 的手册来教用户如何使用 XML schema 展现数据类型。

(2)将数据单元(data units)添加到文档

依赖于用户如何选择去生成 WSDL 文档，生成新的数据定义要求有大量的知识，CXF GUI 工具提供一定数量的帮助来描述数据类型。其他的 XML 编辑器提供不同级别的帮助。不论用户选择哪种编辑器，拥有一些与文档相关的知识是很重要的。定义 WSDL 中用到的数据，需以下步骤：

- 确定接口要用到的所有数据单元；
- 在文档中生成一个 types 元素；
- 创建一个 schema 元素，作为 types 元素的子元素；
- complex 类型是元素的集合，使用 complexType 元素来定义数据类型；
- 对于每个数组来说，定义它的数据类型也使用 complexType 元素；
- 对于每个复杂类型来说，都可以从简单类型衍生出来，定义数据类型可以通过 simpleType 元素；
- 对于每个枚举类型，定义数据类型使用 simpleType 元素；
- 对于每个元素来说，定义它们使用 element 元素。

(3)XML 模式简单类型

如果一个消息组件是简单类型，那么就不需要给它定义一个类型。接口使用的复杂类型也是通过简单类型定义的。

注 1：输入简单类型。XML 简单类型是主要的放置元素。在 element 元素中。简单类型也被使用在 restriction 元素和 extension 元素的 base 属性中。简单类型总是使用 xsd 作为前缀。例如，为了指明类型 int，用户将输入 xsd:int 在 type 属性中。

```
<element name = "simpleInt" type = "xsd:int" />
```

CXF 支持下列 XML schema 简单类型：xsd:string，xsd:normalizedString，xsd:int，xsd:base64Binary 等。

(4)定义复杂数据类型

XML schema 提供灵活和强大的机制来构建负责数据结构。用户可以通过创建一个元素和属性的序列采创建数据结构。用户看可以扩展已有类型来创建更复杂的类型。

另外，为了构建复杂数据结构，用户可以描述特定的类型，如枚举类型。数据类型中的数据有一个特定的取值范围，或者数据类型的数值必须遵循某种特定的模式，通过扩展或者限制原始类型。

(5)定义数据结构

在 XML schema 中,数据单元是数据域的集合,这些数据域是通过负责类型元素定义的。指明一个复杂的类型需要三个信息:

- 复杂元素的名称需要被指定。
- 当它被放入线上时,复杂类型的第一个子元素用来描述该结构的域的行为。
- 每个结构中的域都被用 element 元素定义。

下面的例子是一个复杂类型有两个子元素,结构如下:

```
struct personalInfo
{
  string name;
  int age;
};
```

对应的 WSDL 文档:

```
<complexType name = "personalInfo">
  <sequence>
    <element name = "name" type = "xsd:string" />
    <element name = "age" type = "xsd:int" />
  </sequence>
</complexType>
```

注 1:复杂类型的种类:XML schema 有三种方式描述当 XML 文档被展现并通过线上展示时,其中的域如何被组织。第一个子元素确定哪种复杂类型被引用。下面展示了用来定义复杂类型行为的三种方式:

- sequence:所有的复杂类型域必须被显示,并且它们必须有一个确定的次序,该次序与类型定义的次序相同。
- all:所有的复杂类型域都需要有,但是次序无所谓。
- choice:仅仅是元素中一个可以出现在消息中。

注 2:定义结构的部分:用户定义数据域,这些数据域是由一个结构体组成。每个复杂类型元素应该包括至少一个 element 元素。每个 element 元素对应已经定义的数据结构中的一个域。为了充分描述数据结构中的域,element 必须有两个属性:

- name 属性:指明数据域的名称并且是唯一的
- type 属性:指明该域存储的数据的类型,可以是简单类型,也可以是复杂类型。

除此而外还有两个重要属性:minOccurs 和 maxOccurs。这两个属性用来设置该域在结构中发生的次数上下限。缺省情况下每个字段值发生一次。使用这些属性,用户可以改变结构体中一个域发生的次数。下面的例子中,previousJobs 最少发生 1 次,最多 7 次。

```
<complexType name = "personalInfo>
  <all>
    <element name = "name" type = "xsd:string"/>
```

```
    <element name = "age" type = "xsd:int"/>
    <element name = "previousJobs" type = "xsd:string:
            minOccurs = "3" maxOccurs = "7"/>
  </all>
</complexType>
```

注 3：定义属性：在 XML 文档中，属性被包含在 element 内部。例如，在 complexType 元素中 name 就是属性，它通常跟在<sequence>、<all>等元素的后面。例如：

```
<complexType name = "personalInfo>
  <all>
    <element name = "name" type = "xsd:string"/>
    <element name = "previousJobs" type = "xsd:string"
            minOccurs = "3" maxOccurs = "7"/>
  </all>
  <attribute name = "age" type = "xsd:int" use = "optional" />
</complexType>
```

(6)定义数组

CXF 支持两种方式来定义数组，第一种是定义一个复杂类型，采用简单元素，它的最大发生属性是个大于 1 的值，第二种方式是使用 SOAP 数组。SOAP 数组提供增加的功能，比如 ability 来简化定义多维数组并且发送分离后的数组。

复合类型数组：复合类型数组是特殊的 sequence 复合类型。用户可以通过简单元素简单定义一个复合类型通过，并且制定该元素的最大发生 maxOccurs 属性。例如，定义一个数组，拥有二个浮点数字，用户可以使用复合类型如下：

```
<complexType name = "personalInfo>
  <element name = "averages" type = "xsd:float" maxOccurs = "20"/>
</complexType>
```

用户也可以定义一个最小发生属性。

SOAP 数组：SOAP 数组定义通过 SOAP-ENC:Array，该类型基于 wsdl:arrayType 元素。

```
<complexType name = "TypeName">
  <complexContent>
    <restriction base = "SOAP-ENC:Array">
      <attribute ref = "SOAP-ENC:arrayType"
              wsdl:arrayType = "ElementType<ArrayBounds>"/>
    </restriction>
  </complexContent>
</complexType>
```

上述语法中，typename 指定新定义的数组名，ElementType 指定数组中元素的数据

类型。ArrayBounds 指定数组的维度及元素数量，单位数组通过[]，多维通过[ ][ ]。

```
<complexType name = "SOAPStrings">
  <complexContent>
    <restriction base = "SOAP-ENC:Array">
      <attribute ref = "SOAP-ENC:arrayType"
                 wsdl:arrayType = "xsd:string[]"/>
    </restriction>
  </complexContent>
</complexType>
```

(7)通过 extension 定义类型

类似主要的编码语言，XML schema 允许用户创建数据类型，这些数据类型集成已有的数据类型，这种机制被称之为 extension。例如，用户可以创建一个名叫 alienInfo 的新类型，它集成了 PersonalInfo 结构，通过增加了新的元素 planet。extension 的类型定义包括四个部分：

- 通过 name 属性来定义类型名
- 通过 complexContent 元素来指定新类型将拥有多余一个的元素。
- 被继承的那个类，被称之为 base 类型，通过 base 属性来指定。
- 新类型和属性被定义在 extension 元素中。例子如下：

```
<complexType name = "alienInfo">
  <complexContent>
    <extension base = "personalInfo">
      <sequence>
        <element name = "planet" type = "xsd:string"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

(8)通过 restriction 来定义类型(简单例子)

```
<simpleType name = "SSN">
  <restriction base = "xsd:string">
    <pattern value = "\d{3} - \d{2} - \d{4}"/>
  </restriction>
</simpleType>
```

(9)定义枚举类型

```
<simpleType name = "widgetSize">
  <restriction base = "xsd:string">
    <enumeration value = "big"/>
    <enumeration value = "large"/>
```

```
    <enumeration value = "mungo"/>
  </restriction>
</simpleType>
```

2. 定义元素

在 XML schema 中的元素体现为一个元素的实例。大多数最基本的元素由简单元素组成。像 element 元素，它是由一定数量的复杂类型定义的。它有三个属性：

• 名称：一个需要的属性来指明元素的名字。

• type：指明元素的类型，这个类型可以是任何 XML schema 的原类型或者任何已经命名的复合类型。如果类型已经内置定义，这个属性可以被省略。

• nillable：指定是否元素可以从一个文档中忽略，如果是 ture，则元素可以被任何根据该 SCHEMA 生成的数据文件忽略。

一个元素可以有一个内置类型定义，通过复合类型元素或者简单类型元素，内置类型被指明。一旦用户说明了数据类型是否是复合或者简单的，用户可以定义任何用户需要的数据类型。内置类型定义建议不被使用，因为不支持重用。

5.4.6.4　WSDL 和 UDDI

1. UDDI 简介

UDDI 是 OASIS 发起的一个开放项目，它使企业在互联网上可以互相发现并且定义业务之间的交互。UDDI 业务注册包括三个元件：

• 白页：有关企业的基本信息，如地址、联系方式以及已知的标识；

• 黄页：基于标准分类的目录；

• 绿页：与服务相关联的绑定信息，及指向这些服务所实现的技术规范的引用。

UDDI 是核心的 Web 服务标准之一。它通过简单对象存取协议进行消息传输，UDDI 规范利用了 W3C 和 Internet 工程任务组织(IETF)的很多标准作为其实现基础，比如扩展标注语言(XML)、HTTP 和域名服务(DNS)这些协议。另外，在跨平台的设计特性中，UDDI 主要采用了已经被提议给 W3C 的 SOAP(Simple Object Access Protocol，简单对象访问协议)规范的早期版本。

2. UDDI 数据类型

在 UDDI 注册中心有 4 种主要的数据类型：businessEntity、businessService、bindingTemplate 和 tModel。businessEntity 提供关于商家的信息，可以包含一个或多个 businessService。这个商家是服务提供者。Web 服务的技术和业务描述在 businessService 和其 bindingTemplate 中被定义。每个 bindingTemplate 包含一个对一个或多个 tModel 的引用。tModel 被用于定义服务的技术规范。