CHAPTER 5

第 5 章 可扩展标记语言 XML

学习要点
（1）XML 文档的基本结构。 （2）如何用 CSS 在浏览器中控制 XML 文档的显示？ （3）如何用 XSL 控制 XML 文档在浏览器中的显示？ （4）什么是 XML 数据岛？ （5）XML DTD 和 XML Schema 的作用是什么？ （6）了解 XML DTD 的构成以及如何描述 XML 文档。 （7）如何通过 VS 2010 来定义和生成 XML Schema 文件？ （8）如何进行 XML DOM 的程序设计？ （9）XPath、XPointer、XLink 在 XML 中分别起什么作用？ （10）XML 文档和数据库系统之间的数据交换。 （11）XML 技术的应用和发展趋势。

于 1998 年问世的 XML（eXtensible Markup Language）是以 SGML（标准通用标记语言）为基础的。SGML 是一个国际标准，可以将其理解为是定义其他文档标记语言的语言。SGML 难于使用，而 XML 的目标就是要变得更加简单易用。HTML 也是基于 SGML 的。1999 年提出了 XHTML（eXtensible HTML），XHTML 使用 XML 语法构造规则对 HTML 进行了改写。XHTML 文档的构造规则比 HTML 要精确得多。这些规则的严格程度取决于在 XHTML 页面中所指定的文档类型声明（DOCTYPE）。

从 1998 年起，许多厂商（如 Adobe、IBM、微软、Netscape、Oracle 和 Sun）开始使用 XML 标准，且视 XML 为关键技术。目前许多工具和软件例如 Navigator、Internet Explorer 及 RealPlayer 等，都已经在软件内部使用 XML 技术。XML 文档使数据易于共享。一系列相关的 W3C 推荐标准解决了 XML 文档内进行转换、显示和导航的问题。

学习 XML 之前，必须明确 XML 的特点：XML 不是一个语言，它的规则用来构造其他语言；XML 创建了用来标记内容的基于标签的语言；XHTML

是由 XML 创建的语言的一种，也是对 HTML 的重新构造；XML 是基于 SGML 的。

XML 技术已成为 Web 开发中很重要的一项技术。很多读者可能存在“什么是 XML？XML 能做什么？使用 XML 能带来什么好处？XML 能不能替代数据库？XML 会取代什么？XML 的发展前景如何？如何利用 XML 来进行程序设计？”等等诸如此类的问题。通过本章的学习我们可从中找到全部答案。

5.1 XML 基础

5.1.1 XML 的概念

今天，XML（EXtensible Markup Language，可扩展标记语言）已成为 W3C 推荐使用的标准，是整个 Web 的基本结构和未来技术发展的基础。什么是 XML？

- XML 是一种类似于 HTML 的标记语言；
- XML 是用来描述数据的；
- XML 的标记不是在 XML 中预定义的，你必须定义自己的标记；
- XML 使用文档类型定义（DTD）或者模式（Schema）来描述数据；
- XML 使用 DTD 或者 Schema 后就是自描述的语言。

从上面 XML 定义来看，应清楚以下几点：

（1）可以用 XML 来定义标记，它和 HTML 是不一样的，XML 的用途比 HTML 广泛得多；XML 并不是 HTML 的替代。

（2）XML 不是 HTML 的升级，它只是 HTML 的补充，为 HTML 扩展更多功能，我们仍将在较长的一段时间里继续使用 HTML，但基于 XML 格式的 XHTML 将逐步取代 HTML。

（3）不能用 XML 来直接写网页。XML 文档存放自描述的数据，必须转换成 HTML 格式后才能在浏览器上显示。

一个简单的 XML 文档内容如下所示：

```
<?xml version="1.0"?>
<book>
  <title> XML 语言及应用</title>
  <author> 华铨平等 </author>
  <publisher> 清华大学出版社 </publisher>
  <publishdate> 200509</publishdate>
</book>
```

其中 book、title、author、publisher、publisherdate 都是自定义的标记（tag）。

5.1.2 XML 的特点

1．XML 的可扩展性

XML 具有的扩展性，正是体现了 XML 的强大功能和弹性。在 HTML 里，我们需熟悉许多固定标记后再使用这些标记。而 XML 中，可建立任何需要的标记。可充分发挥想

象力给文档起一些好记的标记名称。例如文档里包含一些游戏的攻略，可以建立一个名为<game>的标记，然后在<game>下再根据游戏类别建立<RPG>、<SLG>等标记。只要清晰，易于理解，可以建立任何数量的标记。

扩展性意味着更多的选择和强大的能力，但同时也产生了一个问题就是必须学会规划。应清楚文档由哪几部分组成、相互之间的关系和如何去识别它们。

2．XML 标记的描述性

标记（tag）又叫标识，也称元素名，用于描述数据，标识文档中的元素。不论是 HTML 还是 XML，标记的本质在于便于理解，如果没有标记，文档在计算机看来只是一个很长的字符串，每个字符串看起来都一样，没有重点之分。通过标记，文档才便于阅读和理解。XML 的扩展性允许为文档建立更合适的标记。不过，标记只是用来识别信息，它本身并不传达信息。

3．XML 语言的规则性

要遵循特定的 XML 语法来标识文档。虽然 XML 的扩展性允许创建新标识，但它仍必须遵循特定的结构、语法和明确的定义。XML 的标记有如下规则：

- 所有的标记都必须有一个相应的结束标记；
- 所有 XML 标记都必须合理嵌套；
- 所有 XML 标记都区分大小写；
- 所有标记的属性都必须用引号“""”括起来；
- 名字中可以包含字母、数字以及下划线；
- 名字不能以下划线开头，不能用诸如关键字 XML 来开头；
- 名字中不能包含空格。

在 XML 文档中的任何差错都会得到同一个结果：不能转换成 HTML，即网页不能被显示。各浏览器开发商已经达成协议，对 XML 实行严格而挑剔的解析，任何细小的错误都会被报告。

4．XML 文档的结构化

XML 促进文档结构化，所有的信息按某种关系排列。也就是说，结构化为 XML 文档建立了一个框架，就像写文章之前有一个提纲一样。结构化使文档看起来不会杂乱无章，每一个部分紧密联系，形成一个整体。结构化有下面两个原则：

- 每一部分（每一个元素）都和其他元素有关联。关联的级数形成了结构。
- 标记本身的含义与它描述的信息相分离。

5．允许 Meta 数据（元数据）

专业的 XML 使用者都会使用 meta 数据来工作。HTML 中我们知道可以使用 meta 标记来定义网页的关键字、简介等，这些标记不会显示在网页中，但可以被搜索引擎搜索到，并影响搜索结果的排列顺序。XML 对这一原理进行了深化和扩展，可以用 XML 描述信息在哪里，可以通过 meta 来验证信息、执行搜索、强制显示或者处理其他的数据。

下面是一些 XML metadata 在实际应用中的用途：可用于数字签名，使在线商务的提交动作有效；可以建立索引和进行更有效的搜索；可以在不同语言之间传输数据。W3C 组

织正在研究一种名为 RDF（Resource Description Framework）的 metadata 处理方法，可以自动交换信息，W3C 宣称，使用 RDF 配合数字签名，将使网络中存在“真实可信”的电子商务。

6．XML 的多样显示性

单独的 XML 文档使用格式化技术，比如 CSS 或者 XSL，才能在浏览器中显示。XML 将数据和格式分离。XML 文档本身并不知道如何显示数据，而必须由辅助文件来帮助实现。XML 中用来设定显示风格样式的文件类型有：

（1）XSL

XSL（Extensible Stylesheet Language，可扩展样式语言）是将来 XML 文档显示的主要文件类型。它本身也是基于 XML 格式的。使用 XSL，你可以灵活地设置文档显示的样式，文档将自动适应任何浏览器和 PDA（掌上电脑）。XSL 也可以将 XML 转化成 HTML 在浏览器中显示。

（2）CSS

CSS 是目前用来在浏览器上显示 XML 文档的主要方法。

（3）Behaviors

Behaviors 现在还没有成为标准。它是微软的 IE 浏览器特有的功能，用它可以对 XML 标记设定一些有趣的动作。

7．允许 XML DOM 操作

XML DOM 全称是 XML Document Object Model（文档对象模型），DOM 是用来干什么的呢？假设把 XML 文档看成一个单独的对象，DOM 就是如何用脚本语言对这个对象进行操作和控制的标准。XML 创建了标记，而 DOM 的作用就是告诉 Script 脚本语言如何在浏览窗口中操作和显示这些标记。

5.1.3 XML 与 HTML 的区别

1．传统的 HTML 存在的问题和不足

（1）HTML 的标记是固定的，有约 70 多个。Web 技术的飞速发展使新的数据格式不断产生并需要在网上展示，标准的 HTML 语法格式无法创建新的标记。也将无法支持那些专门的页面格式，例如数学公式、化学方程式、音乐乐谱、财务报表以及工程应用等。

（2）DHTML 带来的问题。在标准 HTML 无法满足用户需求情况下，人们在其基础上增加了动态的成分，如脚本程序等。但这些非标准技术制作的网页在不同的浏览器之间互不兼容。

（3）HTML 只是一种表现技术，它并不能揭示 HTML 标签所标记的信息的任何具体含义。例如，语句<h1>Peach</h1>是表示在 Web 浏览器中用标题 1 显示文本“Peach”，但 HTML 标记却没有表明“Peach”究竟代表什么意思，它可能是指一种水果，也可能是某公司的名字，或者是一个别的什么东西。HTML 当初在制定时并没有考虑到这方面的功能。

2．HTML 与 XML 的对比

XML 技术的发展可以大大弥补 HTML 的不足。表 5-1 列出了 HTML 与 XML 的对比。

表 5-1　HTML 与 XML 对比表

比 较 内 容	HTML	XML
可扩展性	不具有扩展性	可用于定义新的标记语言
侧重点	侧重于如何表现信息	侧重于如何结构化地描述信息
语法要求	不要求标记的嵌套、配对等，不要求标记之间具有一定的顺序	严格要求嵌套、配对和遵循树型结构
可读性及可维护性	难于阅读、维护	结构清晰，便于阅读和维护
数据和显示的关系	内容描述与显示方式整合为一体	仅为内容描述，它与显示方式相分离
保值性	不具有保值性	具有保值性
编辑及浏览工具	已有大量的编辑、浏览工具，例如 Frontpage、DreamWeaver 等	有较多编辑、浏览工具。例如 Vervet Logic 的 XML Pro V2、微软的免费软件 XML Notepad 2.2、ALTOVA 公司的 XML SPY 等

在学习 XML 技术过程中，我们会经常遇到很多技术名词，例如 XML DTD、XML Schema、XSL、CSS 等，图 5-1 列出了这些技术相互之间的关系。

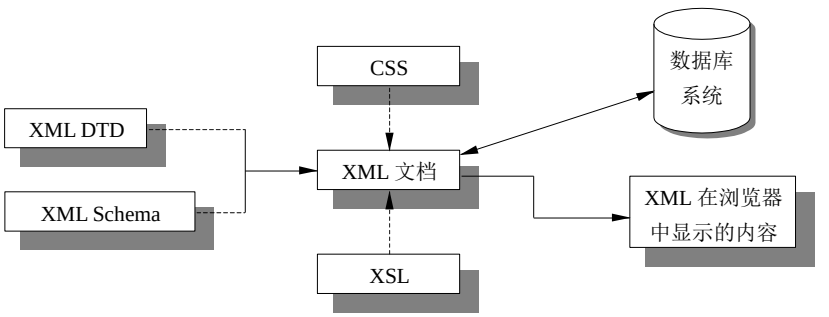


图 5-1　XML 相关技术关系图

图 5-1 中，XML DTD 是一种文本说明内容，既可以放在一个单独文档中，又可以直接放在某个 XML 文档中，用以说明 XML 文档中数据的类型和格式。不过由于 XML DTD 本身是非 XML 文档结构的，其对 XML 文档数据类型和格式的描述过于复杂，用户在使用时较难掌握，目前已逐步被 XML Schema 所替代。XML Schema 中对 XML 文档中数据类型和格式的描述采用了 XML 文档结构。CSS 和 XSL 分别用以说明 XML 文档在浏览器中以什么方式显示其中的数据。这些技术将会在后续章节中一一介绍。

5.2　XML 文档的基本结构

5.2.1　XML 文档的有关术语

1．Element（元素）和 Tag（标识）

元素在 HTML 中我们已经有所了解，它是组成 HTML 文档的最小单位，在 XML 中也一样。一个元素由一个标识来定义，包括开始和结束标识以及其中的内容，如：<author> book

</author>。唯一不同的是，在 HTML 中，标识是固定的，而在 XML 中，标识需要你自己来创建。一般来说我们可以混淆标识与元素的区别。

2．Attribute（属性）

属性是对标识的进一步描述和说明，一个标识可以有多个属性。XML 元素可以像 HTML 一样在开始标识（start tag）里书写属性。属性用来提供关于元素的附加信息。属性常用来提供数据部分以外的信息。如：

```
<file type="gif">computer.gif</file>
```

如上代码中可以看到，type 是属性，computer.gif 是数据。type 属性与数据并不相关，只是用来附加说明该元素用了 gif 格式的数据。

3．Declaration（声明）

每个 XML 文档的第一行都有一个 XML 声明<?xml version="1.0"?>。这个声明表示这个文档是一个 XML 文档，它遵循的是哪个 XML 版本的规范。

4．DTD（Document Type Definition，文档类型定义）

DTD 是用来定义 XML 文档中元素、属性以及元素之间关系的。通过 DTD 文档可以检测 XML 文档的结构是否正确。但建立 XML 文档并不一定必须需要 DTD 文档。

5．Well-formed XML（良好格式的 XML）

一个遵守 XML 语法规则，并遵守 XML 规范的文档称之为“良好格式”。如果所有的标识都严格遵守 XML 规范，那么 XML 文档就不一定需要 DTD 文档来定义它。

良好格式的文档必须以一个 XML 声明开始，如：<?xml version="1.0" standalone="yes" encoding="UTF-8"?>。其中必须说明文档遵循的 XML 版本，目前是 1.0；其次说明文档是“独立的”，它不需要 DTD 文档来验证其中的标识是否有效；最后要说明文档所使用的语言编码，默认的是 UTF-8。

6．Valid XML（有效的 XML）

一个遵守 XML 语法规则，并遵守相应的 DTD 文档规范的 XML 文档称为有效的 XML 文档。Well-formed XML 和 Valid XML 的最大区别就在于，前者完全遵循 XML 规范，后者有自己的文档类型定义（DTD）。

5.2.2 XML 文档基本结构

XML 文档是一个纯文本文件，可以用任意的文本编辑器编写，如记事本、Word 等。为了提高编写效率，也有一些专门的可视化 XML 创作及编辑工具，例如美国 Altova 公司的 XMLSpy 2006 企业版（http://www.xmlspy.com/）、Oxygen 公司的 XML Editor（http://www .oxygenxml.com/index.html）等，用户可从网上下载这些工具。

下面是一个典型的 XML 文档。

```
<?xml version="1.0" encoding="GB-2312" standalone="yes"?>
<?xml-stylesheet  type="text/css"  href="book.css"?>
<中国古典名著>
  <书>
```

```
    <书名>三国演义</书名>
    <作者>罗贯中</作者>
    <内容简介>略…</内容简介>
  </书>
  <书>
    <书名>西游记</书名>
    <作者>吴承恩</作者>
    <内容简介>略…</内容简介>
  </书>
</中国古典名著>
```

1．XML 文档结构由三个部分组成

（1）XML 文档声明

它位于文档的第一行，一般形式为：

```
<?xml version="versionNumber" [encoding="Value"]  [standalone="yes/no"] ?>
```

其中，versionNumber 为 XML 文档所遵循的 XML 规范的版本号：可选项 encoding 表示 XML 处理器使用的字符集，默认值为 UFT-8；可选参数 standalone 取值为 yes 或 no，默认值为 yes，表明该文档是否为一个独立文档。

（2）文档显示方式或文档类型定义等的声明部分

文档类型定义（DTD）部分，一般形式是<!DOCTYPE…>，如不需要可以省略。上例中第 2 行说明了此 XML 文档将由 book.css 定义的样式单来决定其显示方式。

（3）XML 标识的文档内容

2．XML 文档内容的结构

（1）声明根元素

每一个有效的 XML 文档有且仅有一个根元素。根元素是在一个 XML 文档中包含所有其他元素的元素，无论是在语法上还是逻辑上，根元素位于所有数据的顶层。根元素的声明和其他元素的声明方法一样，一般形式为：

```
<rootElementName>…</rootElementName>
```

rootElementName 是根元素的名称，必须成对出现，且区分大小写。根元素在逻辑上代表了数据的顶层，它必须位于 XML 声明结束后的下面一行。

（2）声明非根元素

在 XML 中，是通过在容器元素中嵌套被包含元素来描述数据对象的。一个被包含元素又可以包含自己的元素。包含其他元素的元素称为容器，所有的非根元素都包含在根元素中，根元素是最上层的容器元素。

```
<containedElement [attributesList=""]>
  <containedElement[attributesList=""]>
    …
```

```
  </containedElement>
  …
</containerElement>
```

（3）数据元素属性

一个数据元素可以有若干属性，属性必须在一个元素的起始标记中声明，一般形式为：

```
<elementName [属性名="属性值"] [属性名="属性值"] …="">
  elementValue
</elementName>
```

其中，元素名和属性名必须以字母或下划线开始，并且只能包含字母、数字、下划线、连字符和句点。例如，为汽车定义的三个属性为车牌号、车主和制造商，可以将<automobile>标记写为：

```
<automobile number="123456"  owner="Brion"  manufacture="Ford" >
</automobile>
```

由于其中<automobile>标记中只有属性描述而没有元素值，所以可以缩写成：

```
<automobile number="123456"  owner="Brion"  manufacture="Ford" />
```

可以将元素的属性名转换成元素名，例如上例中的转换结果是：

```
<automobile >
  <number>123456</number>
  <owner>Brion</owner>
  <manufacture>Ford</manufacture>
</automobile>
```

（4）定义名称空间

在 XML 中，用户可以自己定义标记和命名元素。 因此，如果把多个 XML 文件合并为一个，就很可能出现冲突，名称空间就是为此设计的。

XML namespaces 的严格定义是：namespaces 是用 URI 加以区别的、在 XML 文件的元素和属性中出现的所有名称的集合。URI 是 Uniform Resource Identifier（通用资源标志符）的缩写。在没有 namespaces 的 XML 1.0 文件里，元素和属性中出现的名称被称为“本地名称”（local names）。XML 名称空间定义的一般形式为：

```
<namespace:elementName xmlns:namespace="globalUniqueURI">
  <namespace:containedElement  namespace:attributeName="Vaue">
  </namespace:containedElement>
</namespace:elementName>
```

其中，namespace 是名称空间的唯一名称，elementName 是应用名称空间的 XML 文档

元素的名称。globalUniqueURI 是统一资源标识符，可根据实际情况设定一个来作为名称空间的 URI。attributeName 和 attribute Value 是和容器元素 containedElement 相关联的一个属性的名称和属性值。

定义名称空间的目的是唯一地标识一个元素或一组元素的属性。例如：

```
<r:customer xmlns:r="http://www.ABCStore.com/CustomerURI">
  <r:name r:Address="Beijing">Cherry</r:name>
</r:customer>
```

（5）包含非标准文本

通过预定义 XML 实体可以在 XML 文档中加入特殊符号。如果需要大量的特殊符号，可以使用 CDATA 段，CDATA 段可以使用户在一个 XML 文档中引用大量的特殊符号文本块。一般形式为：

```
<![CDATA[text]]>
```

Text 是包含特殊字符的文本串，该文本不被 XML 分析器检查。XML 处理器负责分析或者以一种有意义的方式使用该文本块。

【例 5.1】 Brion 给 Jane 的便条信息使用 XML 格式来说明。ex_5_1.xml 的文档内容为：

```
<?xml version="1.0" encoding="gb2312" ?>
<note>
  <to>Brion</to>
  <from>Jane</from>
  <heading>Reminder</heading>
  <body>Don’t forget me this weekend! </body>
  <![CDATA[This is an example of CDATA]]>
</note>
```

将该文档存盘后，双击该文档将其在浏览器中以树型方式打开，如图 5-2 所示。

图 5-2　一个简单的 XML 文档在浏览器中的显示

如果 XML 文档标记不配对或有其他不符合 XML 文档格式的要求，在浏览器上将显示具体的出错信息。在这里浏览器起着 XML 文档分析器的作用。XML 分析器有确认型和非确认型两种。确认型 XML 文档分析器检查 XML 文档的语法，将 XML 文档同文档类型定义 DTD 或模式文件作比较，还要判断 XML 数据是否和预定义的确认规则相符。非确认型 XML 分析器也进行 XML 文档语法的检查，但不进行 XML 文档和 DTD 及模式文件的比较。在微软的 Internet Explorer 浏览器中内置 XML 确认型分析器，即 MSXML。

5.3 用 CSS 在浏览器中控制 XML 文档的显示

5.3.1 XML 文档的四种 CSS 样式定义方式

本书第 3 章已经详细介绍了在 HTML 文档中如何用 CSS 来控制其页面显示。在这里，控制 XML 文档显示的样式表格式是和 HTML 中的样式表格式相似的，只不过在样式表中，将 HTML 元素名换成了 XML 元素名。下面简要介绍控制 XML 文档显示的 4 种 CSS 样式定义方式。

1．元素名称选择符。可同时为一个或多个元素定义样式。格式如下：

```
XML 元素名称 { 设置的样式规则 }
例如：
Name,company,price,unit,details,.myclass,adderss#a1
{ Display:block;
  Font-weight:bold;
  Font-size:0.8em
}
```

2．用户自定义类选择符。通过对类名的引用，不同的元素可使用同一样式，不同位置的同一元素可使用不同的类名。格式如下：

```
.类名称 { 设置的样式规则 }   或者              /*可以被 XML 文档中的任何元素使用 */
XML 元素名称.类名称 { 设置的样式规则 }         /*只能附加到指定名称的 XML 元素上*/
/*  类选择符定义的样式表应用到 XML 元素的方法是:   <XML 元素名称  class="类名称"> */
```

3．用户定义的 ID 选择符。可以先为某元素指定一个 ID 属性，再在 CSS 样式定义中通过“元素名#ID”的方式指定样式。注意：元素名和 ID 属性前面的#之间不能有空格。格式如下：

```
#ID 号 { 设置的样式规则 }   或者              /*可以被 XML 文档中的任何元素使用 */
XML 元素名称#ID 号 { 设置的样式规则 }         /*只能附加到指定名称的 XML 元素上*/
/*  ID 选择符定义的样式表应用到 XML 元素的方法是:   <XML 元素名称 id="id 号"> */
```

4．成组选择符。格式如下：

```
XML 元素名称 1, XML 元素名称 2, …, XML 元素名称 n { 设置的样式规则 }
.类名称 1, .类名称 2,…, .类名称 n { 设置的样式规则 }
#ID 号 1, #ID 号 2,…, #ID 号 n { 设置的样式规则 }
/*通过成组选择符定义样式表可集中定义多个XML文档元素的相同属性,减少了代码编写工作量。*/
```

5.3.2 CSS 样式和 XML 文档的联系方式

有三种方式可以将定义的 CSS 样式表和 XML 文档联系起来。

1．将定义的 CSS 样式表置于 XML 文档中，其格式为：

```
1  <?xml  version="1.0" encoding="utf-8"?>
2  <?xml-stylesheet  type="text/css"?>
3  <根元素 xmlns:HTML="http://www.w3.org/1999/xhtml">
4    <HTML:STYLE>
5      CSS 选择符 1 { 设置的样式规则 } …
6    </HTML:STYLE>
7    <其他元素> … </其他元素>
8  </根元素>
```

第 2 行不可缺少，告诉浏览器要用 CSS 来显示 XML 文档。第 3 行在根元素中定义了一个 HTML 名称空间，这里必须是 HTML 名称空间，名称空间的 URI 地址可以随意，即使是"abcd"也行，但要注意它的唯一性。在 XML 文档中插入样式单的 style 标志前，必须加上 HTML 名称空间。

2．将 CSS 样式表放在单独的扩展名为 CSS 的文件中，然后在 XML 文档声明部分通过声明语句引用 CSS 文件。

```
1  <?xml  version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/css"  href="css 文件" ?>
3  <根元素>
4    <其他元素> … </其他元素>
5  </根元素>
```

3．将上面两种方式结合起来：既在 XML 文档中定义 CSS 样式，又引用外部 CSS 文件。

第 3 章已经介绍过 CSS 样式规则，为便于读者的学习，下面列出了 XML 文档常用的 CSS 样式规则，如表 5-2 所示。有很多辅助工具可以帮助自动生成 CSS 样式。

表 5-2　XML 文档常用 CSS 样式规则

属　性　名	含　义	取　值	说　明
display	显示方式	Block / none	以块显示，前后换行 / 不显示
		inline	和前、后的元素在一行中显示（默认值）
font-size	字体大小	56%，0.5cm，0.2in，10pc，10pt 1em，1ex，1px	取值也可以是 xx-small，x-small，small，xx-large，x-large，medium，large，smaller，larger 等
font-style	字型	Italic/normal	斜体/正常字体
font-weight	字体粗细	normal	正常粗细
		bold	粗体
		Bolder/lighter	更粗/更细
		100，200，…，900	九种不同的灰度
color background-color	前景色 背景色	Red，green，blue 等	颜色的取值
		rgb(a,b,c)/ rgb(x,y,z)	0<=a,b,c<=255 / 0%<=x,y,z<=100%
		#0000FF	#000000---#FFFFFF
background-image	背景图	图像的 URL	

续表

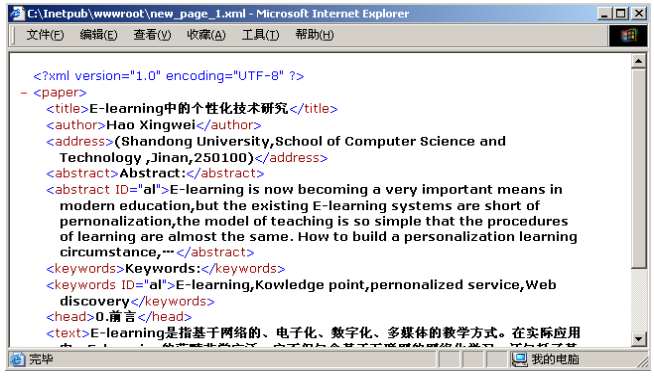
属 性 名	含 义	取 值	说 明
background-repeat	背景图重复	repeat	图像在水平和垂直两个方向上重复
		Repeat-x ； Repeat-y	图像在水平或垂直方向上重复
		No-repeat	图像不垂直
background-position	背景图位置	Top	垂直方向的顶端
		Center	垂直方向的中央
		Bottom	垂直方向的底端
		Left	垂直方向的左端
		right	垂直方向的右端
letter-spacing	文字间距	同 Font-size 取值	
text-align	文本对齐	Left / Center / right	左对齐 / 居中对齐 / 右对齐
text-decoration	文本划线	Underline / Overline	下划线 / 上划线
		Line-through / none	中划线 / 无划线
margin	页边距	同 Font-size 取值	上、下、左、右页边距
margin-top	上边距	同 Font-size 取值	也用作段落间距和左右缩进
margin-bottom margin-Left margin-right	下边距 左边距 右边距	同 Font-size 取值	
border-style	边框线样式	Dotted， dashed，solid，double，groove，ridge，inset，outset，none	边框线
border-weight	边框线粗	同 Font-size 取值	
border-color	边框线颜色	同 Font-size 取值	
text-indent	首行缩进	同 Font-size 取值	
line-height	行距	同 Font-size 取值	

【例 5.2】 CSS 样式表置于 XML 文档中示例。将下面代码保存为 ex_5_2.xml。

```
<?xml  version="1.0" encoding="UTF-8"?>
<?xml-stylesheet  type="text/css"?>
<paper xmlns:HTML="http://www.w3.org/1999/xhtml">
  <!-- 样式单定义 -->
  <HTML:STYLE>
    paper { Display:block;Font-size:20px;Line-height:160%;Text-align:
    center}
    author{ Display:block; Font-size:16px;Margin-top:5px;Margin-bottom:5px}
    Address{ Display:block; Font-size:16px; Text-align:center}
    abstract{
    Display:block;  Font-size:20px; Font-weight:bold; Font_style:italic;
    LText-align:left}
    abstract#al{
    Display:block; Font-size:14px; Font-weight:normal; Font_style: normal;
    Line-height:150%;}
    keywords{
    Display:block; Font-size:20px; Font-weight:bold;  Font_style: italic;}
```

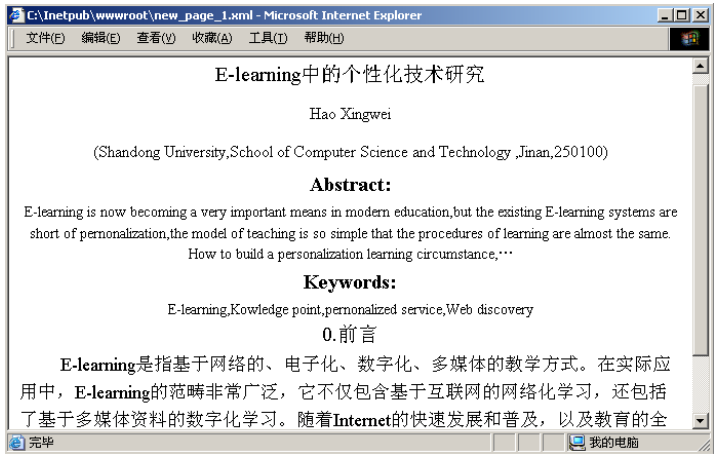
```
    keywords#al
    {Display:block;Font-size:14px; Font-weight:normal;Font_style: normal;
    Line-height:150%;}
    head{ Display:block;Font-size:20px; Line-height:150%}
    text{Display:block; Font-size:18px; text-align:left; text-indent:30pt;
    Line-height:150%;}
  </HTML:STYLE>
  <title>E-learning 中的个性化技术研究</title>
  <author>Hao Xingwei</author>
  <address>
    (Shandong University,School of Computer Science and Technology,Jinan,
    250100)
  </address>
  <abstract>Abstract: </abstract>
  <abstract ID="al">
    E-learning is now becoming a very important means in modern education,but
the existing E-learning systems are short of pernonalization,the model of
teaching is so simple that the procedures of learning are almost the same.
How to build a personalization learning circumstance,…
  </abstract>
  <keywords>Keywords:</keywords>
  <keywords ID="al">
    E-learning,Kowledge point,pernonalized service,Web discovery
  </keywords>
  <head>0.前言</head>
  <text>
    E-learning 是指基于网络的、电子化、数字化、多媒体的教学方式。在实际应用中,E-learning
的范畴非常广泛，它不仅包含基于互联网的网络化学习，还包括了基于多媒体资料的数字化学习。随
着 Internet 的快速发展和普及，以及教育的全球化和由此带来的教育竞争的加剧，基于 Web 的
E-learning 系统已经成为许多大学和教育机构实施现代化远程教育的重要手段。
  </text>
</paper>
```

在浏览器中运行效果如图 5-3 所示。



（a）没有 CSS 样式定义的 XML 文档

图 5-3　例 5.2 的运行效果



（b）用 CSS 控制 XML 文档的输出效果

图 5-3（续）

【例 5.3】 一个使用了 CSS 样式表的 XML 文档。样式文件 ex_5_3.css 内容如下：

```
Student{ Display: block}
Name.c1{ Font-size: 3em;color:red}
Name.c2 Font-size:2em;color:green}
Name.c3{ Font-size: 1em;color:blue}
```

引用 ex_5_3.css 样式的 XML 文档内容如下：

```
<?xml version="1.0" encoding="GB-2312" ?>
<?xml-stylesheet type="text/css"  href="ex_5_3.css" ?>
<student>
  <name class="c1">张三</name>
  <name class="c2">李四</name>
  <name class="c3">王五</name>
</student>
```

运行该 XML 文档，其显示效果如图 5-4 所示。

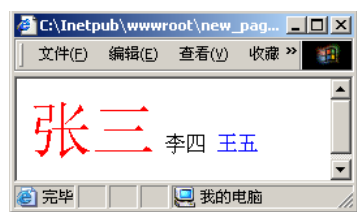


图 5-4　XML 使用 CSS

【例 5.4】 用表格来显示学生花名册，其中有两个学生的资料。从例中可以知道，通过 CSS 要用表格来表示 XML 文档，必须使用 HTML 名称空间，且每个 HTML 标记前必须指定 HTML 名称空间，例如“<html:tr>…</html:tr>”，如果标记不配对，则要用“/”表示结束，例如“<html:img src="bg.jpg" alt="it's a background images"/>”。在指定 HTML 元素的样式时，必须用“html\:”指定名称空间。ex_5_4.xml 文档内容如下：

```
<?xml version="1.0" encoding="gb2312" ?>
<?xml-stylesheet type="text/css" href="ex_5_4.css"?>
<roster xmlns:html="http://www.w3.org/1999/xhtml">
```

```
  <html:style type="text/css">
    html\:caption {font-weight:bold; text-decoration:underline} /*指定表格标
                                                          题的样式*/
    html\:table {background-image: url("images/bg.jpg")}  /* 指定表格的背景
                                                          图片*/
    .myclass {border-style:double;}
    #myid {border-style:groove;}
  </html:style>
  <html:table border="1" cellPadding="2">
    <html:caption> 学生花名册</html:caption>
    <html:tr>
      <student>
        <html:td> <name>李华</name> </html:td>
        <html:td> <origin id="myid">河北</origin> </html:td>
        <html:td> <age>15</age> </html:td>
        <html:td> <telephone>62875555</telephone> </html:td>
      </student>
    </html:tr>
    <html:tr>
      <student>
        <html:td> <name>张三</name> </html:td>
        <html:td> <origin class="myclass">北京</origin> </html:td>
        <html:td> <age>14</age> </html:td>
        <html:td> <telephone>82873425</telephone> </html:td>
      </student>
    </html:tr>
  </html:table>
</roster>
```

ex_5_4.css 文件内容如下：

```
roster,student {font-size:15pt;font-weight:bold; color:blue; display:block;
margin-bottom: 5pt;}
origin,age,telephone  {font-weight:bold;  font-size:12pt;  display:block;
color:block; margin-left: 20pt;}
name {font-weight:bold;font-size:14pt;display:block;color:red;margin-top:
5pt;
margin-left:8pt;}
```

此时，文件 ex_5_4.xml 在 IE 下的浏览效果如图 5-5 所示。

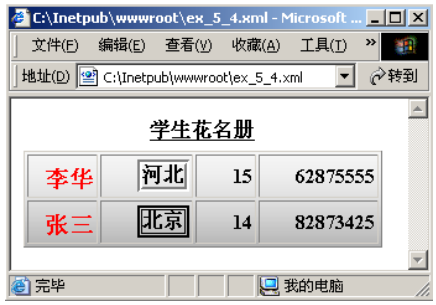


图 5-5　使用 CSS 将 XML 文档按表格输出

5.4　用 XSL 控制 XML 文档在浏览器中的显示

5.4.1　XSL 概述

可扩展样式单语言 XSL 是 eXtensible Stylesheet Language 的缩写，它是由 W3C 1999 年 11 月年制定的。XSL 自提出以来争议颇多，前后经过了几番大的修改。2007 年 1 月 W3C 发布了修改后的 XSLT 2.0 版本（http://www.w3.org/TR/xslt20/）。它仍然在进一步修改完善中。

CSS 通过创建 XML 元素的样式单来格式化 XML 文档，并且将其显示出来。而 XSL 采取的方式更加引人注目，它将 XML 文档转换成一个新的文档（包括 HTML 文档），通过浏览器或其他应用程序就可以显示出来。

XSL 本身也是遵循 XML 文档格式规范的一种标识语言，它提供的强大功能远远超过 CSS，如将元素再排序等。简单的 XML 文档可以通过 CSS 来转换输出，然而复杂的、高度结构化的 XML 文档则只能依赖于 XSL 极强的格式化能力展现给用户。

XSL 和 CSS 之间的异同如下：

① XSL 与 CSS 在很多功能上是重复的，但是它比 CSS 功能强大。不过 XSL 的强大功能与其复杂性是分不开的。

② CSS 只允许格式化元素内容，不允许改变或安排这些内容。但 XSL 没有这些限制，它可以提取元素、属性值、注释文本等几乎所有的文档内容。在 XML 领域，用 XSL 来格式化文档是未来发展的方向。

③ CSS 是一种静态的样式描述格式，其本身不遵从 XML 的语法规范。而 XSL 不同，它是通过 XML 进行定义的，遵守 XML 的语法规则，是 XML 的一种具体应用。也即 XSL 本身就是一个 XML 文档，系统可以使用同一个 XML 解释器对 XML 文档及其相关的 XSL 文档进行解释处理。

XSL 由两个关键部分组成：一个为转换引擎，将原始文档树（源树）转换为能够显示的文档树（结果树），这个过程称作树转换；另一个为格式化符号集，该符号集可以定义应用 XML 数据上的复杂的格式化规则，这个过程称作格式化。格式化符号集又称为格式对象 FO（Formatted Object）。

到目前为止，W3C 还未能出台一个得到多方认可的 FO，但是描述树转换的这一部分协议却日趋成熟，已从 XSL 中分离出来，另取名为 XSLT（XSL Transformations），正式推

荐标准于 2007 年 1 月问世，现在所说的 XSL 都是指 XSLT。与 XSLT 一同推出的还有其配套标准 XPath，这个标准用来描述如何识别、选择、匹配 XML 文档中的各个构成元件，包括元素、属性、文字内容等。

如前所述，XSLT 主要的功能就是转换，它将一个没有形式表现的 XML 文档作为一个源树，将其转换为一个有样式信息的结果树。在 XSLT 文档中定义了与 XML 文档中各个逻辑成分相匹配的模板，以及匹配转换方式。值得一提的是，尽管制定 XSLT 规范的初衷只是利用它来进行 XML 文档与可格式化对象之间的转换，但它的巨大潜力却表现在它可以很好地描述 XML 文档向任何一个其他格式的文档作转换的方法，例如转换为另一个逻辑结构的 XML 文档、HTML 文档、PDF 文档、XHTML 文档、VRML 文档、SVG 文档等。转换过程如图 5-6 所示。

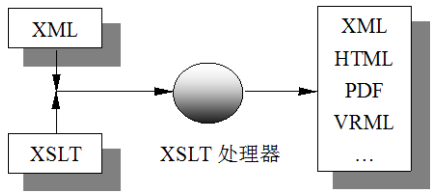


图 5-6　XSLT 转换过程

使用 XSL 定义 XML 文档显示方式的基本思想是：通过定义转换模板，将 XML 源文档转换为带样式信息的可浏览文档。最终的可浏览文档可以是 HTML 格式、FO 格式，或者其他面向显示方式描述的 XML 格式（如前面提到的 SVG 和 SMIL），限于目前浏览器的支持能力，大多数情况下是转换为一个 HTML 文档进行显示。

在 XML 中声明 XSL 样式单的方法与声明 CSS 的方法大同小异：

```
<? xml-stylesheet type="text/xsl" href="xsl 文件" ?>
```

下面先来看一个 XSLT 的简单例子。通过剖析这个例子，读者可以掌握一些 XSLT 的基本语法和功能，甚至可以照葫芦画瓢写出自己的 XSLT 文档。

【例 5.5】 下面是描述了包括三张 CD价目表的 XML 文件ex_5_5.xml：

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<?xml-stylesheet type="text/xsl" href="ex_5_5.xsl"?>
<CATALOG>
  <CD ID="001">
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD ID="002">
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tylor</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
```

```
    <YEAR>1988</YEAR>
  </CD>
  <CD ID="003">
    <TITLE>Greatest Hits</TITLE>
    <ARTIST>Dolly Parton</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>RCA</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1982</YEAR>
  </CD>
</CATALOG>
```

下面是控制价目表显示的 XSL 文档 ex_5_5.xsl：

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
  <xsl:template match="/">
    <html>
      <body>
        <table border="2" bgcolor="yellow">
          <tr><th>Title</th> <th>Artist</th></tr>
          <xsl:for-each select="CATALOG/CD"> <!--对 XML 文档树中根元素下的 CD 元素
          循环-->
        <tr>
        <td><xsl:value-of select="TITLE"/></td>  <!--提取 TITLE 元素的值-->
        <td><xsl:value-of select="ARTIST"/></td> <!--提取 ARTIST 元素的值-->
        </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

将上面的 XML 文件和 XSL 文件放在一个目录中，用 IE 打开 XML 文件，显示结果如图 5-7 所示。

图 5-7　使用 XSL 显示 XML 文档

5.4.2　XSL 模板元素

从例 5.5 可知，XSL 样式文档的基本结构如下：

① 以下面的指令作为文档开头（其中还可以包含其他属性，字符集也有许多选项）。

<?xml version="1.0" encoding="utf-8">

② 通过“xsl:stylesheet xmlns:xsl”来声明 XSL 名称空间。为使用新版本 XSLT，应将“http://www.w3.org/TR/WD-xsl”名称空间换成“http://www.w3.org/1999/XSL/Transform”。特别注意，采用不同的名称空间在浏览器中显示的结果可能会不一样。

③ 通过“xsl:template”定义模板来描述 XML 文档的显示格式。这是 XSL 的主要部分。

④ 通过 XML 数据的引用指明显示的数据。

⑤ 其中包含了大量 XHTML 语句的各种标记，标记必须配对。

⑥ 通过 xsl:for-each、xsl:if、xsl:choose 等语句进行数据的循环处理、条件处理、选择处理等工作。

⑦ 可以嵌入 JavaScript 或 VBScript 脚本语句或程序，使 XSL 具有更强大的运算功能。

在 XSL 中，数据的显示格式被设计细化成一个个模板，最后再将这些模板组合成一个完整的 XSL。这种方法可以使用户先从整体上考虑整个 XSL 的设计，然后将一些表现形式细化成不同的模板，再具体设计这些模板，最后将它们整合在一起。这样，宏观与微观设计的结合，更符合人们的条理化、规范化要求。由于 XML 的数据保存在具有严格层次结构的各个元素中，这种结构非常适合采用模板化的格式样式。图 5-8 为用模板格式化 XML 文档示意图。

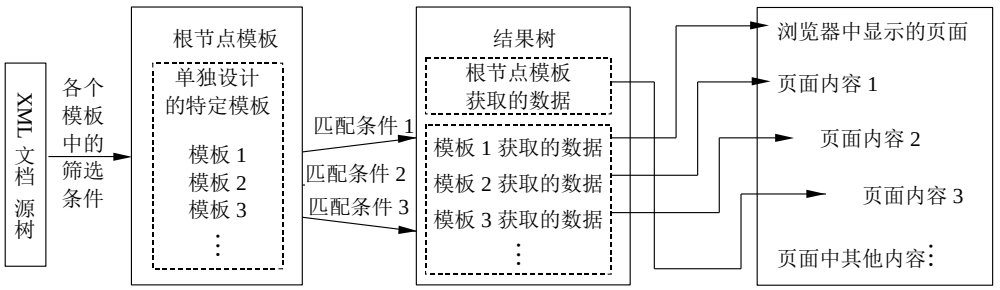


图 5-8　用模板格式化 XML 文档示意图

模板定义好了后，可通过 call-template 或 apply-templates 来调用模板，其过程就如同我们在 C 语言中定义了一个函数，就可在程序中需要的地方进行函数调用。

1．定义模板的语法结构

定义模板的语法结构为：

```
<xsl:template match="node-context" name="template name">  …
</xsl: template>
```

其中的属性含义如下：

① match 确定什么样的情况下执行此模板，也即源 XML 文档中哪些节点应该被相关的模板所处理，在此处使用 XML 文档中节点的名字；其中最上层的模板必须将 match 设为“/”。在一个 XSL 文档中必须有一个根模板，而且是唯一的。

② <xsl:template>元素用 match 属性从 XML 文档中选取满足条件的节点，针对这些特定的节点形成一个特定输出形式的模板。在一个 XSL 文档中一般要设计多个模板，在各个模板结构中描述了不同层次元素的数据显示格式、数据引用、数据处理等内容。

③ name 属性即是为定义的模板取一个用户自定义的名称。只能通过<xsl:call-template>元素来调用模板。

2．通过 name 调用模板

通过 name 属性调用模板的语法格式如下：

```
<xsl:call-template  name="template name"/>
```

其中 name 属性代表的模板名称和模板定义的名称相同。

3．通过 select 调用模板

通过 select 属性调用模板的语法格式如下：

```
<xsl:apply-templates  select="pattern">
```

其中 select 属性确定应调用的模板，即选取用<xsl:template>标记建立的模板。

对于设计好的模板，是通过<xsl:apply-templates>元素调用的，这样，即使以后要对这些模板做相应的修改与扩充也很方便，不致于出现互相干扰、混杂不清的情况。这种从上至下、逐层细化的设计方法，极大地减少了工作复杂程度，也大大减少了差错的产生，可以实现多人协作设计。在学习 XSL 模板时，为便于理解，可将定义模板看成定义一个函数，调用模板看成调用函数。

【例 5.6】 下面是一个对例 5.5 中 CD 价目表的 XML 文件采用 call 模板处理的例子。第 3～9 行定义了一个根模板，它是必需的。第 10～17 行定义了一个模板，模板名称为"myTemplate"，第 6 行按名称通过<xsl:call-template>标记进行模板调用。这种方式类似于函数调用。最终在浏览器中输出“CD 001: Empire Burlesque CD 002: Hide your heart CD 003: Greatest Hits”。Ex_5_6.xsl 文件内容如下：

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <xsl:stylesheet  version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/
   Transform">
3    <xsl:template match="/"> <!—根模板是必需的-->
4      <HTML><HEAD> <TITLE>模板的调用</TITLE></HEAD>
5        <BODY>
6          <xsl:call-template name ="myTemplate"/>   <!-- 调用模板-->
7        </BODY>
8      </HTML>
9    </xsl:template>
10   <xsl:template name="myTemplate" match ="CATALOG/CD" >
11    <p align="center" style="font-weight:bold;">
12       <!--对所有 CD 循环处理-->
13       <xsl:for-each select="CATALOG/CD"> <!--CATALOG/CD 大小写要和 XML 文
                                              档中一致-->
14         CD <xsl:value-of select="@ID"/>: <!--提取 CD 的 ID 属性的值-->
15       <xsl:value-of select="TITLE"/>     <!--提取每个 CD 的 title-->
16     </xsl:for-each> </p>
17   </xsl:template>
18 </xsl:stylesheet>
```

【例 5.7】 下面是一个对例 5.5 中 CD 价目表的 XML 文档采用 apply 模板处理的例子。第 10～17 行定义了一个模板，第 6 行 select 通过<xsl: apply-templates>标记进行模板调用。最终输出结果和例 5.6 相同。Ex_5_7.xsl 文件内容如下：

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <xsl:stylesheet  version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/
   Transform">
3    <xsl:template match="/">
4      <HTML><HEAD><TITLE>模板的调用</TITLE></HEAD>
5        <BODY>
6          <xsl:apply-templates select ="CATALOG"/>
7        </BODY>
8      </HTML>
9    </xsl:template>
10   <xsl:template match ="CATALOG">
11     <p align="center" style="font-weight:bold;">
12       <xsl:for-each select="./CD">
13         CD <xsl:value-of select="@ID"/>:
14         <xsl:value-of select="TITLE"/>
15       </xsl:for-each>
16     </p>
17   </xsl:template>
18 </xsl:stylesheet>
```

上面例子中，只用了一个根模板和一个自定义的模板，可根据实际情况自定义多个模板，而且可在模板中再定义模板。此过程如同函数中再调用函数，可以嵌套很多层。

5.4.3　XSL 选择元素和测试元素

XSL 选择元素的作用是：用选择的方式将数据从 XML 文档中提取出来。这是一种在 XSL 中广泛应用且操作简单的获得数据的方法。XSL 测试元素的过程是：先对选择的对象进行测试，然后对符合条件的记录进行预定的处理。

1．XSL 选择元素

选择元素有两种不同的方式，各自的语法格式描述如下。

（1）xsl:value-of

```
语法：  <xsl:value-of select="模式"/>
功能：  该语法的作用是从 XML 文档中提取指定节点的数据输出到结果树中。select 属性用来指定
XML 文档数据节点名称。
```

（2）xsl:for-each

```
语法：  <xsl:for-each  select="模式"> … </xsl:for-each >
功能：  循环处理指定节点的相同数据。通过 select 属性选择 XML 文档中的某些元素进行循环处理。
```

2．XSL 测试元素

测试语法有以下两种方式，各自的语法格式描述如下。

（1）xsl:if

```
语法：<xsl:if test="测试条件"> …内容…  </xsl:if>
功能：  测试 test 属性给定的条件，当条件的值为 True 时执行相关内容，否则不执行。
```

对于测试条件，情况比较复杂。测试条件可以为一个关系表达式或逻辑表达式，其书写规则如下面的例子所示：

```
姓名="张三"                表示：节点元素“姓名”的值为“张三”
成绩 &gt;= 90              表示：节点元素“成绩”的值大于或等于 90
成绩 &gt;= 90 or 成绩 &lt; 60]  表示：节点元素“成绩”的值大于等于 90 或小于 60
@性别 ="女"                表示：当前节点元素的“性别”属性值为“女”时
```

（2）xsl:choose

```
语法：  <xsl:choose>
          <xsl:when test="测试条件"> …内容… </xsl:when>
          <xsl:when test="测试条件"> …内容… </xsl:when>
          …
          <xsl:otherwise>  …内容… </xsl:otherwise>
      </xsl:choose>
功能：   在有多个测试条件的情况下执行满足条件（由 test 指定测试条件）的内容。当所有条件都不满足时执行<xsl:otherwise>指定的内容。
```

5.4.4　XSL 常用运算符

XSL 中的运算符包括选择运算符和特殊字符、逻辑运算符、关系运算符以及集合运算符，这些运算符构成的表达式可以使用在测试条件中，常用的运算符如表 5-3 所示。

表 5-3　常用的运算符

运算符	含　义	运算符	含　义
/	选择子元素，返回左侧元素的直接子元素；位于最左侧的/表示选择根节点的直接子元素	//	引用任意级别的后代元素
		.	当前元素
*	通配符，选择任意元素，不考虑名字	@	属性名的前缀
！*	在相关节点上应用指定方法	@*	通配符，选择任意属性
（）*	分组，明确指定优先顺序	[]	应用过滤样式
:	名字作用范围分隔符，将名字作用范围前缀与元素或属性名分隔开来	[]*	下标运算符，在集合中指示元素
|	集合运算符，返回两个集合的联合	or	逻辑或
>	大于。在 XSL 中，要用> 表示	and	逻辑与
>=	大于等于。在 XSL 中，要用>= 表示	not()	逻辑非
<	小于。在 XSL 中，要用< 表示	=	相等
<=	小于等于。在 XSL 中，要用<= 表示	!=	不等
mod	取模	+, –, *, div	加减乘除

【例 5.8】 该例针对存放简历的 XML 文档 Ex_5_8.xml，在 Ex_5_8.xsl 中采用了几种不同的运算符。在应用时，只需要将 ex_5_8.xsl 文档中的第 8 行进行替换。Ex_5_8.xsl 中第 13 行演示了如何直接提取 birthday 的值。读者可仿照此例举一反三地练习。

```
(1) <xsl:for-each select="*/resume">
    说明：此处用通配符*代替了 document 元素名称。对每个 resume 循环处理。
(2) <xsl:for-each select="/document/resume [grade &lt; 60]">
    说明：[ ]表示选择条件，仅循环处理成绩<60 分的学生
(3) <xsl:for-each select="//resume [@id='008']">
    说明：对简历中具有 Id 属性编号为“0008”的人进行处理
(4) <xsl:for-each select="*/resume [cellphone]">
    说明：对简历中具有"cellphone"元素的人进行处理，也即对简历中提供了手机号码的人进行
    处理。
(5) <xsl:for-each select="*/resume [skill='Web 开发']">
    说明：对简历中具有"Web 开发"技能的所有人进行处理
```

ex_5_8.xml 文档：

```
<?xml version="1.0" encoding="gb2312" ?>
<?xml-stylesheet type="text/xsl" href="ex_5_8.xsl"?>
<document>
  <resume id="007">
    <name>李敏</name>
    <sex>男</sex>
    <birthday>1971-12-30</birthday>
    <skill>Web 开发</skill>
    <skill>游戏编程</skill>
    <cellphone>13983911111</cellphone>
    <grade>50</grade>
  </resume>
  <resume id="008">
    <name>王甜珠</name>
    <sex>女</sex>
    <birthday>1981-01-30</birthday>
    <skill>舞蹈</skill>
    <grade>80</grade>
  </resume>
</document>
```

ex_5_8.xsl 文档：

```
1  <?xml version="1.0" encoding="gb2312"?>
2  <xsl:stylesheet  version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/
   Transform">
3    <xsl:template match="/">
```

```
    <HTML><HEAD><TITLE>XML 技术</TITLE></HEAD>
      <BODY BGCOLOR="#00CC66">
        <TABLE border="1" cellspacing="0">
          <TH>姓名</TH><TH>性别</TH>
          <xsl:for-each select="/document/resume [grade &lt; 60]">
            <TR><TD><xsl:value-of select="name"/></TD>
                <TD><xsl:value-of select="sex"/> </TD>
            </TR>
          </xsl:for-each>
        </TABLE> 第个人的生日是<xsl:value-of select="/*/*/birthday"/>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

5.4.5 XSL 内置函数

XSL 提供了 100 多个内置函数，这些函数大大方便了开发者的使用，例如用 current()可获得当前节点，用 current-date()、current-time()分别用来返回当前日期和时间。可参阅 http://www.w3schools.com/xpath/xpath_functions.asp 获取所有内置函数列表。这里主要介绍几个常用的 XSLT 内置函数。

1．position()函数、last()函数

```
作用：分别表示确定当前和最后一个节点元素的位置。
举例：<xsl:if test="position()!=last()">
说明：判断当前节点元素是否为最后一个。
```

2．count()函数

```
作用：统计计数，返回符合条件的节点的个数。
举例：<p><xsl:value-of select="count(PERSON[name=tom])"/></p>
说明：显示 PERSON 元素中姓名属性值为 tom 有几个。
```

3．number()函数

```
作用：将属性值中的文本转换为数值。
举例：<p> The number is: <xsl:value-of select="number(book/price)"/></p>
说明：显示书的价格。
```

4．substring()

```
语法：substring(value, start, length)
作用：截取字符串。
举例：<p><xsl:value-of select="substring(name, 1, 3)"/></p>
说明：截取 name 元素的值，从第一个字母开始直到第三个。
```

5．string-length(string)函数

```
语法：string-length(string)
作用：获取字符串的长度。
举例：<p><xsl:value-of select="string-length(name)"/></p>
说明：获取 name 元素的字符串长度值。
```

6．concat()和 string-join()串连接函数

```
语法：concat(string,string,...) 或者 string-join((string,string,...),sep)
作用：将多个字符串连接在一起。后者在连接子串时可带分割符号。
举例：concat('XPath ','is ','FUN!')
      string-join(('We', 'are', 'having', 'fun!'), ' ')  用空格连接字串
说明：前者返回 'XPath is FUN!', 后者返回 'We are having fun! '
```

7．sum()、max()、min()、avg()函数

```
作用：分别表示求和、求最大值、求最小值、求平均值。
举例：<p>Total Price = <xsl:value-of select="sum(//price)"/></p>
说明：计算所有价格的和。其中//表示引用任意级别的后代元素。
```

5.4.6　XSL 中的其他常用元素

XSL 中除了前面讨论的元素外还提供了一些其他处理元素。表 5-4 列出了几个常用的 XSL 处理元素。为了方便读者理解和掌握使用这些元素，请阅读例 5.9。

表 5-4　XSL 常用其他处理元素

xsl 元素	说明及例子
xsl:comment	向结果树中写入一个注释，例如<xsl:comment>这是一个例子</xsl:comment>
xsl:copy	将源文档中的当前节点复制到结果树中，当前节点的子节点不复制
xsl:output	指定在结果树中创建什么类型的格式。例如 <xsl:output method="text" indent="yes"/>指定生成一个文本型结果文档，该元素应放在根模板前。method 可以取值为 xml、html 和 text。Indent 指定生成的结果文档是否要缩进
xsl:sort	与 xsl:for-each 或者 xsl:apply-templates 一起使用，为选定的节点列表排序。例如 <xsl:sort order="descending" select="TITLE"/>中，order 有升序 ascending 和降序 descending 选项；select 属性指定要排序的节点元素
xsl:text	输出指定文本到结果树中。例如 <xsl:text>我输出来了</xsl:text>
xsl:variable	用来声明和分配局部或者全局变量
xsl:element	在结果树中以指定的名称创建一个元素
xsl:attribute	为结果树中的某元素创建一个属性，已有同名属性则被它替换

【例 5.9】 下面是一个对例 5.5 中 CD 价目表的 XML 文档采用相关 XSL 元素处理的例子。第 7 行表示了一个注释；第 11 行表示按 CD 的 title 降序排序；第 14 行用于向结果树中输出文本；第 24、25 行演示了如何定义变量和访问变量；第 27～30 行演示了如何在结果树中定义新元素和属性（这里动态创建了 HTML 的 IMG 元素）；第 31、32 行演示了 sum

函数和 substring 函数的使用。最终浏览器效果如图 5-9 所示。ex_5_9.xsl 文件内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet  version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" indent="yes"/>
<xsl:template match="/">
<html><head><title>XSL 其他元素应用示例</title></head>
     <body>
       <xsl:comment>这是一个例子</xsl:comment>  <!--注释-->
       <h2>My CD Collection</h2>
       <p> Titles:
       <xsl:for-each select="CATALOG/CD">
          <xsl:sort order="descending" select="TITLE"/>
          <xsl:value-of select="TITLE"/>
          <xsl:if test="position()!=last()">
            <xsl:text>, </xsl:text>
          </xsl:if>
          <xsl:if test="position()=last()-1">
            <xsl:text> and </xsl:text>
          </xsl:if>
          <xsl:if test="position()=last()">
            <xsl:text>!</xsl:text>
          </xsl:if>
        </xsl:for-each>

         <xsl:variable name="myStr">I'm OK!</xsl:variable> <!--创建了一个变量 myStr-->
         <xsl:value-of select="$myStr"/> <!--输出变量 myStr 的值-->

  <xsl:element name="img">   <!--创建了一个 HTML 的 img 元素-->
  <!--img 元素的 src 属性-->
  <xsl:attribute name="src">mmc.gif</xsl:attribute>
  <!—img 元素的 alt 属性-->
  <xsl:attribute name="alt">This is a picture</xsl:attribute>
  </xsl:element>
         <p> Total Price = <xsl:value-of select="sum(CATALOG/CD/PRICE)"/>
         </p>
         <p> <xsl:value-of select="substring(CATALOG/CD/TITLE, 1, 3)"/>
         </p>
       </p></body>
</html>  </xsl:template>
</xsl:stylesheet>
```

5.4.7　XSL 应用实例

图 5-9　浏览器显示效果

XSL 语言包含的内容非常丰富，很难在一个章节中覆盖所有细节。我们在数据库中，可通过 SQL 语言编程对数据进行各种操作，以实现数据的查询、修改或更新。在学习 XSLT 技术时可以这样来理解 XSL 的应用，就是将 XML 源文档看成是数据库中的数据表，将 XSL 语言看成针对数据库操作的 SQL 语言，可以用 XSL 语言对 XML 源文档进行查询、修改或更新操作，最后将处理的结果在浏览器中显示出来。下面的几个例子可以满足读者入门之需要。

【例 5.10】 下面是一个对例 5.5 中价目表的 XML 文档采用 xsl:if 控制的 XSL 文件 ex_5_10.xsl。

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
  <xsl:template match="/">
    <html>
      <body>
        <table border="2" bgcolor="yellow">
          <tr> <th>Title</th><th>Artist</th></tr>
          <xsl:for-each select="CATALOG/CD">
            <xsl:if test="ARTIST='Bob Dylan'">
              <tr>
                <td><xsl:value-of select="TITLE"/></td>
                <td><xsl:value-of select="ARTIST"/></td>
              </tr>
            </xsl:if>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

将上面的 XML 文件和 XSL 文件放在一个目录中，用 IE 打开 XML 文件，显示效果如图 5-10 所示。

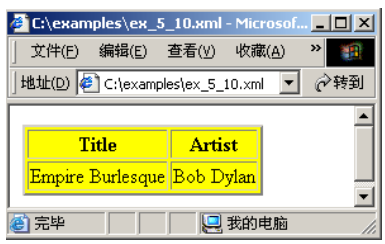


图 5-10　采用 xsl:if 的 XSL 文件显示效果

【例 5.11】 下面是一个对例 5.5 中价目表的 XML 文档采用 xsl:choose 控制的 XSL 文件 ex_5_11.xsl，其内容为：

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
  <xsl:template match="/">
    <html>
      <body>
        <table border="2" bgcolor="yellow">
          <tr><th>Title</th><th>Artist</th></tr>
          <xsl:for-each select="CATALOG/CD">
            <tr>
              <td><xsl:value-of select="TITLE"/></td>
              <xsl:choose>
                <xsl:when test="ARTIST='Bob Dylan'">
                  <td bgcolor="#ff0000">
                    <xsl:value-of select="ARTIST"/>
                  </td>
                </xsl:when>
                <xsl:otherwise>
                  <td><xsl:value-of select="ARTIST"/> </td>
                </xsl:otherwise>
              </xsl:choose>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

将上面的 XML 文档和 XSL 文件放在一个目录中，用 IE 打开 XML 文件，显示效果如图 5-11 所示。

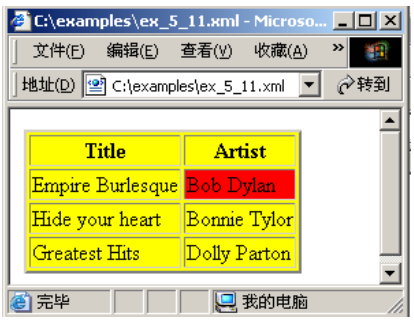


图 5-11 采用 xsl:choose 的 XSL 显示效果

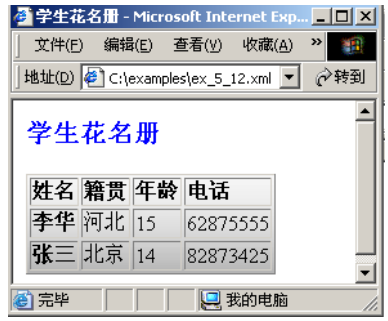


图 5-12 采用模板的 XSL 显示效果

【例 5.12】 下面是一个和例 5.4 相比较的 XSL 控制 XML 文档显示的例子（见图 5-12）。将该例和例 5.4 中采用 CSS 方式格式化 XML 文档的方式相比，用 XSL 要灵活方便得多。ex_5_12.xml 文档内容如下：

```
<?xml version="1.0" encoding="gb2312"?>
<?xml-stylesheet type="text/xsl" href="ex_5_12.xsl"?>
<roster>
  <student>
    <name>李华</name>
    <origin>河北</origin>
    <age>15</age>
    <telephone>62875555</telephone>
  </student>
  <student>
    <name>张三</name>
    <origin>北京</origin>
    <age>14</age>
    <telephone>82873425</telephone>
  </student>
</roster>
```

对应的 ex_5_12.xsl 文件内容如下：

```
1  <?xml version="1.0" encoding="gb2312"?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
   Transform">
3    <xsl:template match="/">
4      <HTML>
5        <HEAD><TITLE>学生花名册</TITLE>
6          <STYLE>
7            .title{font-size:15pt; font-weight:bold; color:blue }
8            .name{color:red}
9            table {background-image: url("images/bg.jpg")}
10         </STYLE>
11       </HEAD>
12       <BODY>
13         <P class="title" >学生花名册</P>
14         <xsl:apply-templates select="roster"/>
15       </BODY>
16     </HTML>
17   </xsl:template>
18
19   <xsl:template match="roster">
20     <TABLE BORDER="1">
21       <THEAD>
```

```
        <TD><B>姓名</B></TD>
        <TD><B>籍贯</B></TD>
        <TD><B>年龄</B></TD>
        <TD><B>电话</B></TD>
      </THEAD>
      <xsl:for-each select="student">
        <xsl:sort order="ascending" select="name"/>
        <TR>
          <TD> <B><xsl:value-of select="name"/></B>  </TD>
          <TD>   <xsl:value-of select="origin"/>    </TD>
          <TD>   <xsl:value-of select="age"/>       </TD>
          <TD>   <xsl:value-of select = "telephone"/></TD>
        </TR>
      </xsl:for-each>
    </TABLE>
  </xsl:template>
</xsl:stylesheet>
```

5.5 XML 数据岛及其应用

所谓 XML 数据岛就是在 HTML 页面文档中采用 HTML 中的专门标签“<XML></XML>”定义的一块数据。可用 HTML 标记的属性 datasrc 指定 XML 数据岛的 ID，用 datafld 属性绑定 XML 数据源中的节点元素名称来获取数据。

XML 数据岛在 HTML 页面中使用有两种方式，即内嵌式和连接式。内嵌式就是直接将 XML 文档的片段数据插入到 HTML 页面中“<XML>”和“</XML>”之间，语法形式如下：

```
<XML id="数据岛 ID">     元素列表     </XML>
```

而连接式则是将整个外部的 XML 文档链接到 HTML 页面中，其语法形式如下：

```
<XML ID="数据岛 ID"  SRC="XML 文档 URI"> </XML>
```

XML 数据岛的应用更多的是与 XML DOM 结合使用，读者可在学完 DOM 部分后强化这方面的应用。下面举两个简单的例子来说明 XML 数据岛在 HTML 中的应用。

【例 5.13】 XML 数据岛在 HTML 中的应用示例一。XML 文档 ex_5_13.xml 的内容如下：

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<studentList>
  <stu>
    <sid>200202051001</sid>
```

```
    <name>黎　佳</name>
    <gender>男</gender>
    <cName>02 级财务(1)班</cName>
  </stu>
  <stu>
    <sid>200202051002</sid>
    <name>樊　娟</name>
    <gender>女</gender>
    <cName>02 级财务(1)班</cName>
  </stu>
  <stu>
    <sid>200202051003</sid>
    <name>韦　威</name>
    <gender>男</gender>
    <cName>02 级财务(1)班</cName>
  </stu>
</studentList>
```

引用该 XML 文档的 HTML 文件 ex_5_13.htm 的代码如下：

```
<html><head><title>XML 数据岛在 HTML 中的应用</title> </head>
<body style="background-color:#fffff0">
<xml id="xmldata" src="ex_5_13.xml"></xml>
   <h2>财务管理专业学生名单</h2>
   <table datasrc="#xmldata" style="border-style:inset;border-color:Red"
   border="1">
      <thead> <th>学号</th> <th>姓名</th><th>性别</th> <th>班级</th> </thead>
       <tr>
          <td><span datafld="sid"></span></td>
          <td><span datafld="name"></span></td>
          <td><span datafld="gender"></span></td>
          <td><span datafld="cName"></span></td>
       </tr>
   </table>
</body></html>
```

运行后的效果如图 5-13 所示。

图 5-13　XML 数据岛在 HTML 页面的简单应用

【例 5.14】 XML 数据岛在 HTML 中的应用示例二。

```
<HTML><HEAD><Title>HTML 中的数据岛中的记录集</Title></HEAD>
<body bgcolor="#00FFFF">
<font color="#FF0000"><b>HTML 中的 XML 数据岛记录编辑与添加</b></font>
<HR>酒店名称: <input type="text" datasrc="#islet" DataFLD="NAME"><BR>
    地址: <input type="text" datasrc="#islet" DataFLD="Address"><BR>
    主页: <input type="text" datasrc="#islet" DataFLD="HomePage"><BR>
电子邮件: <input type="text" datasrc="#islet" DataFLD="E-Mail"><BR>
    电话: <input type="text" datasrc="#islet" DataFLD="TelePhone"><BR>
    级别: <input type="text" datasrc="#islet" DataFLD="Grade"><HR>
<input id="first" type="button" value="首条" onclick="islet.recordset.
moveFirst()">
<input id="prev" type="button" value="上条" onclick="islet.recordset.
movePrevious()">
<input id="next" type="button" value="下条" onclick="islet.recordset.
moveNext()">
<input id="last" type="button" value="末条" onclick="islet.recordset.
moveLast()">
<input id="Add" type="button" value="添加" onclick="islet.recordset.
addNew()">
<XML ID="islet">
  <HotelList>
    <Hotel>
      <Name>四海大酒店</Name>
      <Address>海魂路号</Address>
      <HomePage>www.sihaohotel.com.cn</HomePage>
      <E-Mail>master@sihaohotel.com.cn</E-Mail>
      <TelePhone>(0989)8888888</TelePhone>
      <Grade>五星级</Grade>
    </Hotel>
    <Hotel>
      <Name>五湖宾馆</Name>
      <Address>东平路号</Address>
      <HomePage>www.wuhu.com.cn</HomePage>
      <E-Mail>web@wuhu.com.cn</E-Mail>
      <TelePhone>(0979)1111666</TelePhone>
      <Grade>四星级</Grade>
    </Hotel>
    <Hotel>
      <Name>"大沙漠"宾馆</Name>
      <Address>留香路号</Address>
      <HomePage>www.dashamohotel.com.cn</HomePage>
      <E-Mail>master@dashamohotel.com.cn</E-Mail>
      <TelePhone>(0989)87878788</TelePhone>
      <Grade>五星级</Grade>
    </Hotel>
  </HotelList>
</XML>
</body></HTML>
```

程序运行后的效果如图 5-14 所示。

图 5-14　XML 数据岛在浏览器中的显示效果

5.6　XML DTD 及 XML Schema

在数据库中，数据表用来存放数据，数据表的结构定义（包括主键、外键、列名、列的数据类型和长度、列之间的约束关系等）保存在数据库系统中。如果把 XML 文档看成数据表，那么我们还需要一种文件来定义 XML 文档中数据的结构类型及其相互关系等，这种文件就是我们本节将要讨论的 DTD（Document Type Definition，文档类型定义）和 XSD（XML Schema Definition）文件。因为 XML 文档允许自建标记元素，随意创建的 XML 文档如不施加某种规则使其结构化，将很难用于相互之间的数据交换。因此 DTD 和 XML Schema 想要解决的问题是如何实现 XML 文档的结构化。

DTD 文件本身不是用 XML 标记元素来组织的，其描述文档类型的方式过于复杂，而 XSD 文件是用 XML 标记元素来组织的；在 DTD 中只有一个数据类型，就是用在元素中的 PCDATA 和用在属性中的 CDATA，在里面写日期也行，数字可以写，字符更是没问题，而 XML Schema 正是针对这些 DTD 的缺点而设计的，具有很强的描述能力、扩展能力和处理维护能力。因此一种趋势是 XML Schema 技术将成为文档类型定义方面的主流技术。由于 DTD 和 Schema 用来描述 XML 数据的底层组织结构，所以我们一般称作 XML DTD 和 XML Schema。一般将 XML Schema 称作 XML 模式，但微软称其为 XML 架构，在使用过程中可混淆这两种叫法。下面分别介绍 XML DTD 和 XML Schema。

5.6.1　XML DTD

DTD 用来定义了 XML 文档中可以使用的元素符号、元素的属性、元素的排列方式/顺序、元素能够包含的内容、可使用的实体或符号规则等。一个 XML 文档可以用多个不同的 DTD 来进行描述。在 XML 文档中使用 DTD 有两种方式：可以在 XML 文档中直接设置（内部 DTD），也可以保存为一个完全独立的文件中（外部 DTD）。此外也可两者兼而有之。

1．定义内部 DTD

下面先来看一个例子，以便对 DTD 作一个大致了解。

【例 5.15】 一个包含了 DTD 定义的 XML 文档。

```
<?xml version = "1.0" encoding="utf-8" standalone = "yes"?>
<!DOCTYPE 联系人列表[
  <!ELEMENT 联系人列表 (联系人)*>
  <!ELEMENT 联系人 (姓名,ID,公司,EMAIL,电话,地址)>
  <!ELEMENT 地址 (街道,城市,省份)>
  <!ELEMENT 姓名 (#PCDATA)>
  <!ELEMENT ID (#PCDATA)>
  <!ELEMENT 公司 (#PCDATA)>
  <!ELEMENT EMAIL (#PCDATA)>
  <!ELEMENT 电话 (#PCDATA)>
  <!ELEMENT 街道 (#PCDATA)>
  <!ELEMENT 城市 (#PCDATA)>
  <!ELEMENT 省份 (#PCDATA)>
]>
<联系人列表>
  <联系人>
    <姓名>张三</姓名>
    <ID>001</ID>
    <公司>A 公司</公司>
    <EMAIL>zhang@aaa.com</EMAIL>
    <电话>(010)62345678</电话>
    <地址>
      <街道>五街 1234 号</街道>
      <城市>北京市</城市>
      <省份>北京</省份>
    </地址>
  </联系人>
</联系人列表>
```

通过浏览器或其他一些工具（称作 XML 解析器）可以检查上述 XML 文档的有效性。如果联系人列表中的数据内容格式和<!DOCTYPE>语句中 DTD 定义部分相吻合，那么浏览器就会显示如图 5-15 所示效果，否则就会出现出错信息。

图 5-15　包含 DTD 定义的 XML 文档

从例 5.15 可以看出，为了将 DTD 声明与 XML 文档相关联，XML 提供了特殊的 DOCTYPE 声明。DOCTYPE 声明必须位于 XML 声明之后，且在任何文档元素之前，但可在这两者之间插入注释和处理指令，其语法如下：

```
<! DOCTYPE 文档根元素 [
   文档类型定义
]>
```

例 5.15 同时还给出了如何定义 DTD 中的元素，其通用形式如下：

```
<! ELEMENT 元素名 规则>
```

有效的 XML 文档中用到的每个元素都必须在 DTD 中预先定义，而元素名的命名规则也必须遵循 XML 文档的基本语法。定义 XML 文档中的元素所遵循的规则是多样的，主要的规则如表 5-5 所示。

表 5-5　元素声明规则

规　则	描　述	形　式
EMPTY 规则	XML 元素必须不包含任何数据	<! ELEMENT 元素名 EMPTY>
例：DTD 中的<! ELEMENT banner EMPTY>，在 XML 文档中符合规则的是<banner /> 或者<banner></banner>		
#PCDATA 规则	元素不含子元素，仅包含字符数据	<! ELEMENT 元素名(#PCDATA)>
例：DTD 中的 <!ELEMENT 公司 (#PCDATA)>，在 XML 文档中符合规则的是 <公司>A 公司</公司> 注：关键字 PCDATA 表示已编译的字符数据(Parsed Character Data)		
多声明	允许元素按一定顺序进行嵌套	<! ELEMENT 元素名(元素名 A,元素名 B)>
例：<!ELEMENT 地址 (街道,城市,省份)> 表示 地址元素由街道,城市,省份三个子元素组成		
混合声明	允许元素既含子元素又含#PCDATA	<! ELEMENT 元素名(#PCDATA|元素名 A)>
ANY 规则	元素中可包含 DTD 认可的任何内容	<! ELEMENT 元素名 ANY>

如例 5.15 所示，在定义元素时用到了上面的规则，还使用了诸如“*”的符号，这些符号是告诉 XML 解析器在 XML 文档中该元素的出现次数，具体含义见表 5-6。

表 5-6　元素出现次数控制符

符号	描　述	示　例	符号	描　述	示　例
?	不出现或只出现一次	元素名?	+	出现一次或多次	元素名+
*	不出现或出现多次	元素名*	（无）	只出现一次	元素名

XML 文档中的元素可以包含各种属性，在 DTD 中如何去定义这些属性呢？只需要通过关键字 ATTLIST 来定义即可，其通用形式如下：

```
<! ATTLIST 元素名 属性名 类型 默认值>
```

属性是不能单独存在的，必然属于某一元素，因此在上面的定义中，“元素名”是指包含该“属性名”的元素。其中的元素名是必需的，因为在 DTD 中，属性可以放在任何位置，如没有与之相对应的元素名，则 XML 解析器就很难辨别该属性的归属，导致解析错误。属性的默认值类型采用表 5-7 中的四种之一。

表 5-7　属性默认值类型

序　号	类　型	描　述
1	#REQUIRED	必须提供属性值
2	#IMPLIED	属性值是可选的
3	#FIXED VALUE	属性值必须提供，且不能更改，例：　#FIXED "center"
4	Default	提供默认的属性值

属性的类型表明了属性值的级别，属性类型如表 5-8 所示。

表 5-8　属性类型

序　号	类　　型	描　　述
1	CDATA	属性值只能为文本串
例: <! ATTLIST anElement myAttr CDATA #REQUIRED>　anElement 元素的 myAttr 属性必须为字串 在 XML 文档中表示：　<anElement myAttr="I love you!">		
2	Enumerated	列出该属性值的取值范围，一次只能有一个属性值能够赋予属性
例: <! ATTLIST anElement myAttr (left|center|right) "center">　默认为 center，可三选一 在 XML 文档中表示：　<anElement myAttr="left">		
3	NMTOKEN	表示属性值必须以字母或下划线开头，之后是字母、数字、下划线、圆点、和短横线，不能有空格
例: <! ATTLIST anElement myAttr NMTOKEN　#REQUIRED > 在 XML 文档中表示：　<anElement myAttr="_A123.555-1">		
4	NMTOKENS	表示属性值能够由多个 NMTOKEN 组成，每个 NMTOKEN 之间用空格隔开
例: <! ATTLIST anElement myAttr NMTOKENS　#REQUIRED > 在 XML 文档中表示：　<anElement myAttr="_A123.555-1 _A778.AC-1 ABCDEF">		
5	ID	该属性在 XML 文件中是唯一的，常用来表示人的身份证号码
例: <! ATTLIST anElement myAttr ID　#REQUIRED >		
6	IDREF	表示该属性值是参考了另一个 ID 属性
例: <! ATTLIST book publisher IDREF #REQUIRED>　表示元素 book 的属性 publisher 的值来自于元素 publisher 的 ID 属性值 在 XML 文档中表示：　<book publisher="p009"> <publisher publishID="p009">		
7	IDREFS	表示该属性值是参考了多个 ID 属性，这些 ID 属性值用空格隔开
8	ENTITY	表示该属性的设置值是一个外部的实体 ENTITY，如一个图片文件
9	ENTITIES	表示该属性值包含多个外部的 ENTITY，不同的 ENTITY 之间用空格隔开
10	NOTATION	将属性值和 DTD 中的<! NOTATION>声明联系起来

其实，一个复杂的 XML 文档除了包含元素（元素中可能含有属性）外，还包含其他一些类型，诸如实体（ENTITY）、标记（NOTATION）等。本书在此不作介绍，读者可参阅相关资料。

2．定义外部 DTD

一个良好的 XML 文档是：文档只存放相关的数据，而将有关该文档的一切定义放在外部文件中，在文档中只是使用简单的语句使之建立联系，从而提高文档的可读性和可维护性。外部 DTD 就是基于此而被人们大量使用。

【例 5.16】 采用外部 DTD 的 XML 文档示例。DTD 文件 ex_5_16.dtd 的内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!ELEMENT 联系人列表 (联系人)*>
<!ELEMENT 联系人 (姓名,EMAIL,地址)>
<!ELEMENT 地址 (街道,城市,省份)>
<!ELEMENT 姓名 (#PCDATA)>
```

```
<!ELEMENT EMAIL (#PCDATA)>
<!ELEMENT 街道 (#PCDATA)>
<!ELEMENT 城市 (#PCDATA)>
<!ELEMENT 省份 (#PCDATA)>
```

XML 文件 ex_5_16.xml 的内容如下：

```
<?xml version = "1.0" encoding="utf-8" standalone = "no"?>
<!DOCTYPE 联系人列表 SYSTEM "ex_5_16.dtd">
<联系人列表>
  <联系人>
    <姓名>张三</姓名>
    <EMAIL>zhang@aaa.com</EMAIL>
    <地址>
      <街道>五街 1234 号</街道> <城市>北京市</城市> <省份>北京</省份>
    </地址>
  </联系人>
</联系人列表>
```

从该例可看出，定义外部 DTD 与定义内部 DTD 差不多，只是将 DTD 作为一个后缀名为.dtd 的独立文档来存放。在 XML 文档中引用该文件时，首先将 XML 声明中的属性“standalone”设置为“no”，然后在 DOCTYPE 声明中采用如下说明即可：

```
<! DOCTYPE 根元素名 SYSTEM "DTD 文件名">
```

其中，如果采用了关键字 SYSTEM 则必须要求该 DTD 文档是 XML 文档私有的，如果系统中存在不止一个文档使用该 DTD 作为模板，则可采用另一关键字 PUBLIC，则 DOCTYPE 声明就变为如下形式：

```
<! DOCTYPE 根元素名 PUBLIC "DTD 文件名" 或 "URI of DTD">  DTD 文件名可以是 DTD 文件的 URI。
```

5.6.2　XML Schema

前面介绍的 DTD 本身是存在许多缺陷的，主要如下：

（1）DTD 是用不同于 XML 语言编写的，需要不同的解析器技术。这增加了工具开发商的负担，他们不仅要开发用于 XML 文档自身的解析器，还要专门开发用于 DTD 的解析器。此外，对于那些学习 XML 技术的人来说，还得专门去学习 DTD 语法规则。

（2）DTD 不支持名称空间（又叫名域空间或名字空间）。XML 的优点是能够很好地支持数据之间的跨平台或跨系统，如果没有名称空间，则各种名称就会发生冲突，致使数据之间无法正确识别。

（3）DTD 不支持继承和子类。

（4）DTD 中没有数据类型的概念，就无法验证某些具有特殊含义的元素是否符合预先定义的要求。

正是意识到 DTD 存在上述薄弱环节，W3C 自 1999 年初开始，以推荐的方式提出一种新的解决方案来定义 XML 文档的结构、内容和语法。这就是我们通常所说的 XML 模式（Schema）或架构（微软称之为架构）。

XML 模式发展到现在已经有很多版本，微软在 VS.NET 2002 中最先采用 XML-DR（XML Data-reduced Schema），2004 年 10 月发布 W3C XML Schema 1.1 建议标准。微软 VS 2010 完全支持上述多种版本。初学 XML Schema 的读者在阅读有关方面的资料时，会发现介绍的语法规则各不相同，会让你产生云里雾里的感觉。因此本节的例子将完全采用 W3C XML Schema 来验证 XML 文档，全部在 VS 2010 环境下调试通过。

XML Schema 的一个应用就是异构数据交换。例如可以将 SQL Server 数据库中的数据取出来，形成 XSD 和 XML 文件，XSD 存放数据结构描述，XML 存放数据内容，然后可以将这两个文件穿过防火墙传送到 Linux 操作系统的 Sybase 数据库中，实现异构操作平台和异构数据库的数据交换。如果没有 XSD 文件，只有 XML 文件，那么数据内容因缺乏数据结构描述而无法自动放入 Sybase 数据库中。XML Schema 的另外一个应用就是电子数据交换。例如国内某公司需要向美国订购一批原材料，把这些订购数据按既定规则自动生成结构描述 XSD 文件和内容描述 XML 文件，传送给美方公司，美方公司依据这些数据开据发货单。

与 DTD 类似，模式也是用来描述 XML 文档的内容和结构，同时在运行 XML 文档时也是解析器验证 XML 文档有效性的依据。在使用模式时，常使用两种文档类型，即实例文档（也就是我们通常所说的 XML 文档）和模式文档（扩展名为 XSD）。

首先来看一个使用模式的 XML 文档，了解其大致构成。

【例 5.17】 采用模式验证的 XML 文档。XML 文档 ex_5_17.xml 的内容如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<booklist xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ex_5_17.xsd">
  <book classify ="自然科学">
    <ISBN>7-302-12066-8</ISBN>
    <title>JAVA 实用教程</title>
    <authorlist>
      <author>Herbert Schildt</author>
      <author>马海军</author>
    </authorlist>
    <price>64.00</price>
  </book>
  <book classify ="社会科学">
    <ISBN>7-5037-1978</ISBN>
    <title>投资学</title>
    <authorlist>
      <author>张中华</author>
      <author>谢进城</author>
    </authorlist>
    <price>19.00</price>
```

```
  </book>
</booklist>
```

验证上述 XML 文档的模式文档 ex_5_17.xsd 的内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="ex_5_17"  elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="booklist">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="book" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ISBN" type="xs:string" />
              <xs:element name="title" type="xs:string" />
              <xs:element name="authorlist" minOccurs="1" >
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="author" type="xs:string"  maxOccurs=
                    "5"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="price" type="xs:decimal" />
            </xs:sequence>
            <xs:attribute name="classify" use="required">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="社会科学" />
                  <xs:enumeration value="自然科学" />
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

ex_5_17.xml 文档通过第 3 行的 xsi:noNamespaceSchemaLocation="ex_5_17.xsd" 调用模式文档 ex_5_17.xsd。运行 XML 文档，就会根据模式文档验证其结构有效性。

上面的 XSD 文件在 VS 2010 中生成的过程如下：在 VS 2010 中，用鼠标右击“解决方案资源管理器”中的项目名或子文件夹，选择“添加新项”，在弹出的对话框中选择“XML 架构”，输入文件名后单击“添加”，就可形成一个空的 XSD 文件，并出现 XSD 设计器界

面。选择“使用 XML 编辑器查看和编辑基础 XML 架构文件”，手工输入上述代码。完成后，可将自动生成的对应节点从 XML 架构资源管理器拖到设计图面上，以查看各个节点的详细信息和节点之间的关系。在各节点上，右击鼠标，选择“查看代码”，就可看到 XSD 文件内容。在 VS 2010 环境下，在编辑对应的 XML 文档过程中会自动按照 XSD 来验证其内容。

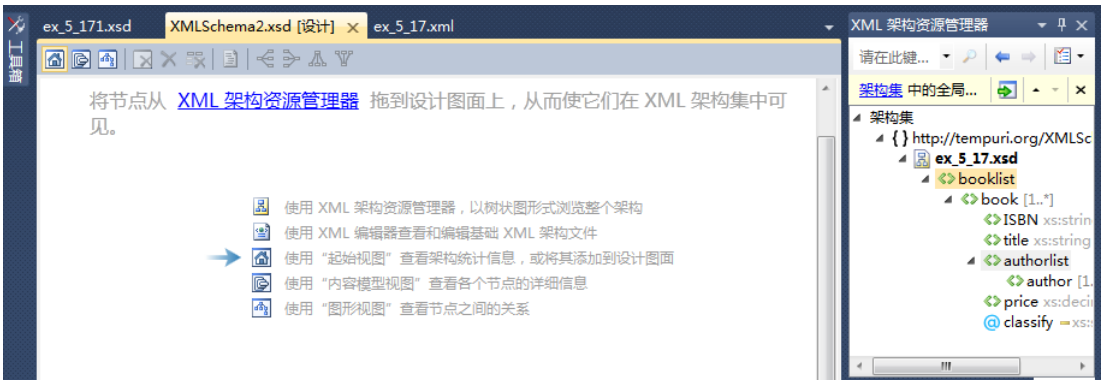


图 5-16　VS 2010 中的 XSD 设计器

从例 5.17 可以看出，模式文档是完全依照 XML 的基本语法和结构来组织的。与 XML 文档相似，模式文档中也存在一个唯一的根元素<Schema>…</Schema>，该元素放在文档的最开头，表明此文档是模式文档，其结构如下：

```
<Schema name="schema-name" xmlns="namespace" >
   其他子元素
</Schema>
```

在例 5.17 定义 XSD 模式文档中，会用到一些重要概念：实例文档、名称空间、元素、复合类型、简单类型和属性。下面分别对这些概念进行介绍。

1．实例文档

被 XML 模式文件来验证的 XML 文档就称作实例文档。

2．名称空间

名称空间是读者在学习 XML Schema 技术过程中最难掌握和应用的一个重要概念。在学习 XML Schema 时应着重掌握以下几点：

（1）由于 XML 文档允许自己定义标志元素，在处理 XML 文档时，例如合并两个 XML 文档时，相同的标志元素可能有着完全不同的含义，因此采用名称空间来解决 XML 文档标志元素的冲突问题。在 XML Schema 技术中，名称空间区分为两种，一种是一些大公司或机构所提供的基础名称空间，用来规定 XSD 和 XML 文档的结构形式以及相关数据类型等；另一种是用户定义的名称空间，可用来防止 XML 文档中元素的重名。

（2）W3C XML Schema 建议规范中有两个基础名称空间：http://www.w3.org/2001/XMLSchema 和 http://www.w3.org/2001/XMLSchema-instance。前者用于 XSD 文档，后者用

于 XML 文档。使用方法见上例中 ex_5_17.xml 和 ex_5_17.xsd 中的第 2 行和第 3 行。它们的作用是用这两个基础名称空间来约束 XSD 和 XML 文档所遵循的规范。例如在这样的名称空间中，可以使用的数据类型有：

```
Anytype、anyURI、base64Binary、boolean、byte、date、datetime、decimal、double、
duration、ENTITY、ENTITIES、float、gDay、gMonth、gMonthDay、gYear、gYearMonth、
hexBinary、ID、IDREF、IDREFS、int、integer、language、long、NAME、NCNAME、
negativeInteger、NMTOKEN、NMTOKENS、nonNegativeInteger、nonPositiveInteger、
normalizedString、NOTATION、positiveInteger、QNAME、short、string、time、token、
unsignedByte、unsignedInt、unsignedLong、unsignedshort
```

如果用其他的名称空间，则数据类型可能就不是或不完全是上面所列的内容。

（3）因 URI 具有唯一性，所以名称空间采用 URI 方式，并采用前缀的方式来表示。这样，在名称空间中声明的标志元素就可以保证其唯一性，相互之间就不会有冲突。上例中采用了 W3C 推荐使用的公共前缀：XML 模式 URI 前缀为"xs:"；XML 文档实例 URI 的前缀为"xsi:"。若你想在 XSD 文件中使用自己的前缀"myxs:"来代替"xs:"，则修改 ex_5_17.xsd 文件的所有"xs"为" myxs"。XSD 中的名称空间前缀"xs:"可以自定义，但随后的基础名称空间的 URI 不能更换。当然也可将"xs"改为"xsd"或不用前缀。微软早期采用了"xsd"前缀，名称空间用 schemas-microsoft-com:xml-data，不过现在已采用 W3C 推荐的名称空间。例 5.17 中，将 ex_5_17.xsd 文档中的前缀改成 mxs 后的内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<myxs:schema id="ex_5_17"  elementFormDefault="qualified"
xmlns:myxs="http://www.w3.org/2001/XMLSchema">
  <myxs:element name="booklist">
    …
  </myxs:element>
</myxs:schema>
```

（4）一个 XML 文档可以使用多个 XSD 文件来验证，需在 XSD 文件中使用 target Namespace 属性指定目标名称空间。例 5.17 中 ex_5_17.xsd 文件的第 2、3 行修改为（其他不变）：

```
<xs:schema  id="ex_5_17"  xmlns:txs="http://local"  targetNamespace="http:
//local"elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/
XMLSchema">
```

其中声明了一个前缀为 txs 的名称空间 http://local，通过 targetNamespace 属性指定 XML 文档将会采用目标名称空间 http://local 来验证。在 ex_5_17.xml 文档中通过 schemaLocation 指定 XSD 中定义的目标名称空间和 XSD 文件路径（注意：XSD 文件名和目标名称空间之间有一个空格），这样就可通过 XSD 来验证该 XML 文档。

```
<?xml version="1.0" encoding="utf-8" ?>
<txs:booklist  xmlns:txs="http://local"  xsi:schemaLocation="http://local
ex_5_17.xsd"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <txs:book classify ="自然科学">
  …
  </txs:book>
```

```
  <txs:book classify ="社会科学">
…
   </txs:book>
</txs:booklist>
```

上例介绍的是采用一个目标名称空间的 XSD 文件来验证 XML 文档，若有多个 XSD 文件来验证 XML 文档，则需按上述方式分别指定每个 XSD 文件的目标名称空间，然后将具有目标名称空间的 XSD 文件与 XML 实例文档相关联。在 XML 文档中通过 schemaLocation 可以列出多对 URI 引用，每一对都有不同的目标名称空间部分，例如：

```
<p:Person  xmlns:p="http://localhost/People" xmlns:v="http://localhost/
Vehicles"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://localhost/People
http://localhost/schemas/people.xsd
http://localhost/schemas/Vehicles http://localhost/schemas/vehicles.xsd">
  <name>John</name>
  <age>28</age>
  <height>59</height>
  <v:Vehicle>
    <color>Red</color>
    <wheels>4</wheels>
    <seats>2</seats>
  </v:Vehicle>
</p:Person>
```

3．简单类型 simpleType

使用简单类型 simpleType 元素可自定义用户数据类型，例如：

```
<xs:simpleType name="orderQuantity">
   <xs:restriction base="xs:integer">
     <xs:minInclusive value="1" />
     <xs:maxInclusive value="50" />
   </xs:restriction>
  </xs:simpleType>
```

说明：自定义了一个 orderQuantity 数据类型，它是介于 1 和 50 之间的整数值。其中 orderQuantity 为自定义数据类型名称，restriction 元素中通过 base 属性指定使用哪种数据类型，minInclusive 和 maxInclusive 元素被称作 facet 元素，指定取值范围。facet 元素除了 minInclusive 和 maxInclusive 外，还有 minExclusive 和 maxExclusive 元素用来指定不在此

取值范围的值；length 元素用来限制字符串的长度、minLength 和 maxLength 元素用来指定元素内容的长度范围；pattern 元素可用于以正则表达式限制元素的数据内容；fractionDigits 和 totalDigits 元素用于表示小数位数和十进制最大数；enumeration 元素用来表示枚举值。例如下面定义了一个 IDType 数据类型，用来表示限制身份证号码为 18 位的数字字符串：

```
<xs:simpleType name="IdType">
   <xs:restriction base="xs:string">
     <xs:length value="18" />
     <xs:pattern value="[0-9]*" />
   </xs:restriction>
 </xs:simpleType>
```

4．元素 element

在模式中定义元素的格式为“<xs:element name="元素名">”。元素又可区分为复合类型（complexType）元素和值类型元素。值类型元素不包含子元素，其语法格式为：

```
<xs:element name ="元素名" type ="数据类型" default ="默认值" fixed ="固定值"
    maxOccurs ="整数"  minOccurs ="整数">
```

其中 maxOccurs 和 minOccurs 表示该元素出现的最大和最小次数，不限次数时取值 unbounded。Default 和 Fixed 分别用来指定该元素的默认值和元素所具有的固定取值，它们不可同时出现。Type 用来指定数据类型，例如 xs:string、xs:integer 等。若该元素使用自定义用户数据类型，则在 type 属性中输入自定义数据类型名称，例如针对上面 simpleType 中定义的 orderQuantity 数据类型，输入 type = "orderQuantity"。

5．组 sequence

sequence 元素用于指定一个组，要求组中的元素以指定的先后顺序出现在容器元素中。在要求若干子元素依次出现的地方就必须加上“<xs:sequence>…</xs:sequence>”。

除了 sequence 元素外，还可用 choice 元素来定义一个组，但组中的元素只能选择其中之一；用 group 来建立一个组，其中又可包括 sequence 和 choice 元素等。

6．复合类型 complexType

复合类型可以看做是模式中元素的基本容器。这种类型可能有多个子元素或多个属性，采用如下形式定义：

```
<xs:element name="元素名">
  <xs:complexType>
    <xs:sequence>
       子元素……
    </xs:sequence>
       属性定义……
  </xs:complexType>
</xs:element>
```

下面定义的复合类型说明在 XML 文档中元素 zooAnimals 元素包含 elephant、bear、giraffe 三个子元素，这三个子元素出现的次数不限，但在实例文档中必须依次出现。

```
<xs:element name="zooAnimals">
  <xs:complexType>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element name="elephant"/>
      <xs:element name="bear"/>
      <xs:element name="giraffe"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

7．属性

与 DTD 类似，模式在定义元素时同样需要考虑哪些元素有属性，都有些什么属性，怎样去定义这些属性。在前面介绍复合类型的元素时，我们可以看到属性定义的放置位置。其一般格式为：

```
<xs:attribute name="属性名称" type="数据类型" default="默认值" fixed="固定值"
         use="required/prohibited/optional" >
</xs:attribute>
```

其中 type 用来指定属性数据类型，例如 xs:string、xs:Boolean 等，也可用来指定 simpleType 自定义数据类型名称。Use 用来表示该属性的使用方式，有三个选项 required/prohibited/optional，分别表示该属性必须输入一个值、禁止使用此属性、属性可有可无（缺省值）。例如，在 XML 实例文档中存在这样的代码片段：

```
<weight units="1bs">24.50</weight>
```

在模式中则必须采用如下的定义方式，才能说明这一代码是有效的。

```
<xs:element name="weight">
  <xs:complexType>
    <xs:attribute name="units" type="xs:float" />
  </xs:complexType>
</xs:element>
```

若换用用户自定义数据类型，则属性定义过程为：

```
<xs:element name="weight">
  <xs:complexType>
    <xs:attribute name="units">
      <xs:simpleType>
        <xs:restriction base="xs:decimal">
          <xs:fractionDigits value="2" />
```

```
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

如果你已经创建了 XML 文档，可以在 VS 2010 中基于现有 XML 文档创建对应的架构文件。方法是在 VS 2010 中首先打开 XML 文档，然后鼠标单击主菜单“XML”|“创建架构”后，即可马上生成同样文件名的 XSD 文件。当然若需将两个文件联系起来，还需要手工在 XML 文档根元素中加入实例名称空间，可参见例 5.17 中的 ex_5_17.xml 文档的第 2、3 行。

以上介绍了如何利用 XMLSchema 来验证一个 XML 文档的有效性。如读者对这方面感兴趣可以查询相关资料进行深入探讨和研究。

5.7　XML DOM 及其编程实例

前面讨论的都是针对一个已有的 XML 文档，如何通过 CSS、XSLT 利用浏览器解析和浏览它，如何通过 DTD 和 XML Schema 来验证它的数据结构。这一节要学习的内容是如何动态生成一个 XML 文档，如何对 XML 文档进行数据的添加、删除、查询等操作。这可以在 VB/VC/Delphi/JavaScript/ASP.NET 等应用程序开发中通过 XML 接口程序来访问 XML 文档。访问 XML 文档的标准应用程序接口有两种：W3C 提出的 DOM（Document Object Model）和 XML_DEV 邮件列表成员提出的 SAX（Simple API for XML）。

5.7.1　XML DOM 与 SAX 简介

XML DOM 是 W3C 提出的针对 XML 的文档对象模型，它独立于平台和语言，定义了一套标准的用于 XML 的对象和一种标准的访问与处理 XML 文档的方法。

对于 XML 应用开发来说，DOM 就是一个对象化的 XML 数据接口，一个与语言无关、与平台无关的标准接口规范。它定义了 XML 文档的逻辑结构，给出了一种访问和处理 XML 文档的方法。利用 DOM，程序开发人员可以动态地创建 XML 文档，遍历文档结构，添加、修改、删除文档内容，改变文档的显示方式，等等。可以这样说，文档代表的是数据，而 DOM 则代表了如何去处理这些数据。无论是在浏览器里还是在浏览器外，无论是在服务器还是在客户端，只要有用到 XML 的地方，都可利用 DOM 接口进行编程应用。

DOM 接口中的 XML 分析器，在对 XML 文档进行分析之后，不管这个文档有多简单或者多复杂，其中的信息都会被转化成一棵对象节点树——DOM 树，也即 DOM 将 XML 文档作为树结构来看待。在这棵节点树中，有一个 Document 根节点，所有其他的节点都是根节点的后代节点。节点树生成之后，就可以通过 XML DOM 接口访问、修改、添加、删除树中的节点和属性以及文本内容，等等。

DOM 将 XML 文档中的每个成分看作一个节点。例如整个文档是一个文档节点；每个 XML 标签是一个元素节点；包含在 XML 元素中的文本是文本节点；每一个 XML 属性是

一个属性节点；注释属于注释节点。

【例 5.18】 DOM 把 XML 文件当作一种树型结构。每个元素、属性以及 XML 文档中的文本都可以看成树上的节点。一个节点树可以把一个 XML 文档展示为一个节点集，以及它们之间的连接。在一个节点树中，最顶端的节点被称为根；每一个节点，除根之外，都拥有父节点；一个节点可以有无限的子节点；叶是无子节点的节点；同级节点指拥有相同的父节点。下面的 ex_5_18.xml 文档，其 DOM 节点树如图 5-17 所示。

```
<?xml version="1.0" encoding="UTF-8" ?>
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="Web">
    <title lang="en">XQuery Kick Start</title>
    <author>James McGovern</author>
    <author>Per Bothner</author>
    <author>Kurt Cagle</author>
    <author>James Linn</author>
    <author>Vaidyanathan Nagarajan</author>
    <year>2003</year>
    <price>49.99</price>
  </book>
  <book category="Web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

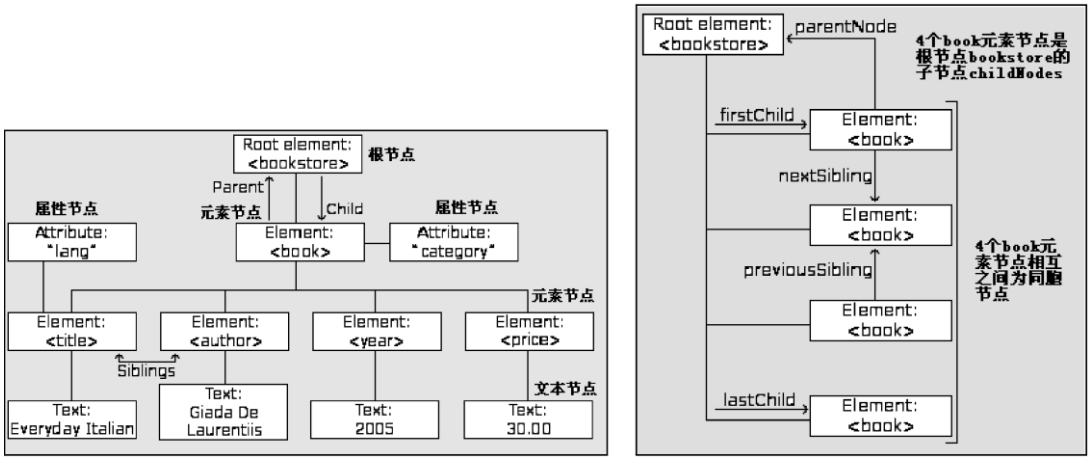


图 5-17　XML 文档的 DOM 节点树

Mozilla 和 IE 浏览器都支持 W3C DOM 规范。但 IE 和 Mozilla 在 DOM 的处理方式上存在某些差异。微软的 XML 解析器是和 IE 5 以及更高版本浏览器合在一起的 COM 组件。一旦安装了 IE，就可使用 XML DOM 了。

通过 XML DOM 接口，应用程序可在任何时候访问 XML 文档中的任何一部分数据，因此，这种利用 XML DOM 接口的机制也被称作随机访问机制。

与 XML DOM 不同，SAX 提供的访问模式是一种顺序模式，这是一种快速读写 XML 数据的方式。当使用 SAX 分析器对 XML 文档进行分析时，会触发一系列事件，并激活相应的事件处理函数，应用程序通过这些事件处理函数实现对 XML 文档的访问，因而 SAX 接口也被称作事件驱动接口。

XML DOM 和 SAX 接口实现过程中分别侧重于不同的方面，二者各有长短，分别满足了不同的应用需求。

DOM 树所提供的随机访问方式给应用程序的开发带来了很大的灵活性，它可以任意地控制整个 XML 文档中的内容。然而，由于 DOM 分析器把整个 XML 文档转化成 DOM 树放在了内存中，因此，当文档比较大或者结构比较复杂时，对内存的需求就比较高。而且，对于结构复杂的树的遍历也是一项耗时的操作。所以，XML 分析器对机器性能的要求比较高，实现效率不十分理想。不过，由于 XML 分析器所采用的树结构的思想与 XML 文档的结构相吻合，同时鉴于随机访问所带来的方便，因此，DOM 分析器得到了广泛的应用。

SAX 分析器在对 XML 文档进行分析时，触发了一系列的事件，由于事件触发本身是有时序性的，因此，SAX 提供的是一种顺序访问机制，对于已经分析过的部分，不能再倒回去重新处理。SAX 之所以被叫做简单应用程序接口，是因为 SAX 分析器只做了一些简单的工作，大部分工作还要由应用程序自己去做。也就是说，SAX 分析器在实现时，它只是顺序地检查 XML 文档中的字节流，判断当前字节是 XML 语法中的哪一部分、是否符合 XML 语法，然后再触发相应的事件，而事件处理函数本身则要由应用程序自己来实现。同 DOM 分析器相比，SAX 分析器缺乏灵活性。然而，由于 SAX 分析器实现简单，对内存要求比较低，实现效率比较高，对于那些只需要访问 XML 文档中的数据而不对文档进行更

改的应用程序来说，SAX 分析器更为合适。

无论是 DOM 接口还是 SAX 接口，都各有其优缺点。也正是如此，它们将长期并存下去，在不同的应用中发挥不同的作用。限于篇幅，本节仅介绍广泛使用的 DOM 接口的编程应用。有兴趣的读者可参阅 SAX 接口相关资料。

5.7.2 XML DOM 对象

在 XML DOM 接口规范中，有四个基本的 XML DOM 对象即 Document（文档）、Node（节点）、NodeList（节点列表）和 NamedNodeMap（有名节点映射）。

1．Document 对象

Document 对象代表了整个 XML 文档，因此，它是整棵 DOM 树的根，提供了对文档中的数据进行访问和操作的入口。通过 Document 节点，可以访问到文档中的其他节点，如处理指令、注释、文档类型以及 XML 文档的根元素节点，等等。

创建 Document 对象的语法格式为：

```
JavaScript / JScript:
//MSXML2.0 版本,支持 DTD 的处理
var doc = new ActiveXObject("Microsoft.XMLDOM")
//MSXML3.0 版本, 支持 DTD、XSD 的处理
var doc = new ActiveXObject("Msxml2.DOMDocument.3.0")
VB Script:
Dim docSet doc = CreateObject("Microsoft.XMLDOM")
VB：在 VB 中通过[工程]|[引用]选择 Microsoft XML 6.0(msxml6.dll)
Dim doc As New MSXML2.DOMDocument60  '采用 XML 接口 6.0 版本
xmlDoc.async = False
xmlDoc.Load App.Path & "\books.xml"  '将 XML 文档加载到 DOC 对象中
If (xmlDoc.parseError.errorCode <> 0) Then
   MsgBox("You have error " & xmlDoc.parseError.reason)
Else
   MsgBox doc.childNodes(0).nodeName    '显示文档中第一个节点的名称
End If
Set doc = Nothing
C#:
using System.Xml;
…
XmlDocument doc=new XmlDocument();
```

由此可见，可以在各种应用程序中使用 DOM 接口，无论是客户端还是服务器端。在 ASP.NET 中可以使用 XmlDocument 类实现对 Document 节点的各种操作。在这里仅讨论通过 JavaScript 进行 XML DOM 编程。

Document 对象的主要属性和方法分别如表 5-9 和表 5-10 所示。表格中 IE、F、O、W3C 分别代表浏览器 Internet Explorer、Firefox、Opera 和 W3C 标准，表示是否支持该属性或方法、从什么版本开始支持。

表 5-9　Document 对象主要属性

属　性	描　述	IE	F	O	W3C
async	async 属性可规定 XML 文件的下载是否应当被同步处理	5	1.5	9	No
childNodes	返回属于文档的子节点的节点列表	5	1	9	Yes
doctype	返回与文档相关的文档类型声明（DTD）	6	1	9	Yes
documentElement	返回文档的根节点	5	1	9	Yes
documentURI	设置或返回文档的位置	No	1	9	Yes
firstChild	返回文档的首个子节点	5	1	9	Yes
lastChild	返回文档的最后一个子节点	5	1	9	Yes
nodeName	依据节点的类型返回其名称	5	1	9	Yes
nodeType	返回某个节点的节点类型	5	1	9	Yes
nodeValue	根据节点的类型来设置或返回某个节点的值	5	1	9	Yes
text	返回某个节点及其后代的文本（仅用于 IE）	5	No	No	No
xml	返回某个节点及其后代的 XML 文档内容（仅用于 IE）	5	No	No	No

表 5-10　Document 对象主要方法

方　法	描　述	IE	F	O	W3C
createAttribute (name)	创建一个拥有指定名称的属性节点，并返回新的 Attr 对象	6	1	9	Yes
createCDATA-Section()	创建一个 CDATA 区段节点	5	1	9	Yes
createComment()	创建一个注释节点	6	1	9	Yes
createElement()	创建一个元素节点	5	1	9	Yes
createElementNS()	创建一个带有指定名称空间的元素节点	No	1	9	Yes
createTextNode()	创建一个文本节点	5	1	9	Yes
getElementById(id)	返回带有给定 ID 属性值的元素。如果不存在，则返回 null	5	1	9	Yes
getElementsBy-TagName()	返回一个带有指定名称的所有元素的节点列表	5	1	9	Yes
renameNode()	重命名一个元素或者属性节点			No	Yes
load(文件名)	加载一个 XML 文档				
Loadxml("XML 文档内容")	将存储在字符串中的 XML 文档内容加载为 document 对象				
save(文件名)	保存到一个 XML 文档				

2．Node 对象

Node 对象在整个 DOM 树中具有举足轻重的地位。在 DOM 树中，Node 对象代表了树中的一个节点，如图 5-18 所示。

XML 文档中的每个成分都是一个 Node 对象。从 Node 对象继承过来的对象有 Document、Element、Attribute、Text、Comment 等，这些从 Node 对象继承过来的子对象类型如表 5-11 所示。

注意：尽管所有的对象都继承了用以处理子类节点或父类节点的节点属性或节点对象，但是并不是所有的对象有包含子类或是父类。例如文本节点中可能不包含子类，所以将子类节点添加到文本节点中可能会导致一个 DOM 错误。

Node 对象的主要属性和方法如表 5-12 和表 5-13 所示。

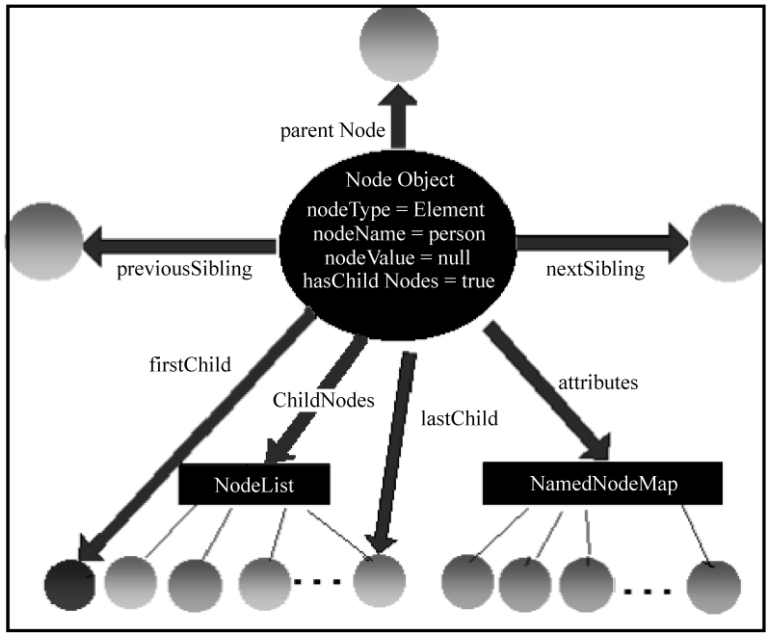


图 5-18　典型的 Node 对象及其相互关系

表 5-11　节点对象的类型

节点对象类型	描　　述	子　对　象
Document	代表整个文件对象（DOM 树根节点）	Element（至多 1 个），ProcessingInstruction，Comment，DocumentType
DocumentFragment	轻型文件对象（允许嵌入另一部分文档）	Element，ProcessingInstruction，Comment，Text，CDATASection，EntityReference
DocumentType	为文档定义的实体列表对象	无
EntityReference	一个实体参数对象	Element，ProcessingInstruction，Comment，Text，CDATASection，EntityReference
Element	代表元素节点对象	Element，Text，Comment，Processing-Instruction，CDATASection，EntityReference
Attr	代表一个属性对象(属性与其他节点类型不同，它们没有父节点)	Text，EntityReference
ProcessingInstruction	处理指令对象	无
Comment	代表注释对象	无
Text	元素中的或属性中的文本内容（字符数据）	无
CDATASection	一块包含字符的文本区，这里的字符也可以是标记	无
Entity	代表实体对象	Element，ProcessingInstruction，Comment，Text，CDATASection，EntityReference
Notation	在 DTD 中声明的 notation 符号对象	无

表 5-12　节点对象的主要属性

属　性	描　述	IE	F	O	W3C
baseURI	返回一个节点的绝对基准 URL	No	1	No	Yes
childNodes	返回一个节点的子节点的节点列表	5	1	9	Yes
firstChild	返回一个节点的第一个子节点	5	1	9	Yes
lastChild	返回一个节点的最后一个子节点	5	1	9	Yes
localName	返回一个节点的本地名称	No	1	9	Yes
namespaceURI	返回一个节点的名称空间的 URI	No	1	9	Yes
nextSibling	返回上一个同胞节点	5	1	9	Yes
nodeName	返回指定节点类型的节点名称	5	1	9	Yes
nodeType	返回节点类型	5	1	9	Yes
nodeValue	设置或返回指定节点类型的节点值	5	1	9	Yes
ownerDocument	返回节点的根元素（文档对象）	5	1	9	Yes
parentNode	返回一个节点的父节点	5	1	9	Yes
prefix	设置或返回一个节点的名称空间前缀	No	1	9	Yes
previousSibling	返回上一个同胞节点	5	1	9	Yes
textContent	设置或返回当前节点及其子节点的文本内容	No	1	No	Yes
text	返回当前节点及其子节点的文本。仅 IE 支持	5	No	No	No
xml	返回当前节点及其子节点的 XML 文档内容。仅 IE 支持	5	No	No	

表 5-13　节点对象主要方法

方　法	描　述	IE	F	O	W3C
appendChild()	将一个新的子节点添加到一个节点中的子类节点列表末尾	5	1	9	Yes
cloneNode()	克隆一个节点	5	1	9	Yes
compareDocumentPosition()	比较两个节点的文档位置	No	1	No	Yes
hasAttributes()	如果节点包含属性，则返回 true；否则返回 false	5	1	9	Yes
hasChildNodes()	如果一个节点包含子节点则返回 true，否则返回 false	5	1	9	Yes
insertBefore()	在当前子节点之前插入一个新的子节点	5	1	9	Yes
isEqualNode()	检验两个节点是否相等	No	No	No	Yes
isSameNode()	检验两个节点是否相同	No	1	No	Yes
removeChild()	删除一个子节点	5	1	9	Yes
replaceChild()	替换一个子节点	5	1	9	Yes

下面列出了从 Node 对象继承过来的对象的相关属性和方法。

（1）Element 对象

Element 对象表示一个 XML 文档中的某个元素。元素可包含属性和文本。假如某个元素含有文本，则此文本由一个文本节点来代表。文本永远被存储于文本节点中。例如在<year>2005</year>中，其中 year 为一个元素节点，此节点之下存在一个文本节点，其中含有文本 2005。由于元素对象也是一种 Node，因此它继承了 Node 对象的属性和方法。表 5-14 列出了 element 对象除了继承 Node 对象的属性和方法外新增的属性和方法。

表 5-14　Element 对象部分属性和方法

属　　性	描　　述	IE	F	O	W3C
attributes	可返回元素的属性的一个 NAMEDNODEMAP	5	1	9	Yes
schemaTypeinfo	可返回与元素相关联的类型信息			No	Yes
tagName	可返回元素的名称	5	1	9	Yes
方　　法	**描　　述**	**IE**	**F**	**O**	**W3C**
getAttribute()	返回某个属性的值	5	1	9	Yes
getAttributeNS()	通过名称空间返回某个属性的值	No	1	9	Yes
getAttributeNode()	以一个 Attribute 对象返回一个属性节点	5	1	9	Yes
getAttributeNodeNS()	通过名称空间返回一个属性节点对象	No		9	Yes
getElementsByTagName()	返回匹配元素节点以及它们的子节点的 NodeList	5	1	9	Yes
getElementsByTagNameNS()	通过名称空间返回匹配元素节点以及它们的子节点的 NodeList	No	1	9	Yes
hasAttribute()	返回某元素是否拥有匹配某个指定名称的属性	5	1	9	Yes
hasAttributeNS()	返回某元素是否拥有匹配某个指定名称和名称空间的属性	No	1	9	Yes
hasChildNodes()	返回元素是否拥有任何子节点	5	1	9	Yes
removeAttribute()	删除某个指定的属性	5	1	9	Yes
removeAttributeNS()	删除某个指定的带有某个名称空间的属性	No	1	9	Yes
removeAttributeNode()	删除某个指定的属性节点	5	1	9	Yes
setUserData(key,data,handler)	把某个对象关联到元素上的某个键			No	Yes
setAttribute()	添加一个新属性	5	1	9	Yes
setAttributeNS()	通过名称空间添加一个新属性		1	9	Yes
setAttributeNode()	添加一个新的属性节点	5	1	9	Yes

（2）Attr 对象

Attr 对象表示某个 Element 对象的一个属性。属性的容许值通常被定义在某个 DTD 中。

由于 Attr 对象也是一种节点，因此它可继承 Node 对象的属性和方法。不过属性无法拥有父节点，同时属性也不被认为是元素的子节点，对于许多 Node 对象的属性来说都将返回 null。表 5-15 列出了 Attr 对象除了继承 Node 对象的属性外新增的属性。

表 5-15　Attr 对象的部分属性

属　　性	描　　述	IE	F	O	W3C
name	返回属性的名称	5	1	9	YES
specified	如果属性值被设置在文档中，则返回 TRUE，如果其默认值被设置在某个 DTD/SCHEMA 中，则返回 FALSE	5	1	9	YES
value	设置或返回属性的值	5	1	9	YES

（3）Text 对象

Text 对象表示元素或属性的文本内容。Text 对象的部分属性和方法见表 5-16 所示。

表 5-16　Text 对象的部分属性和方法

属　性	描　述	IE	F	O	W3C
data	设置或返回元素或属性的文本	6	1	9	Yes
isElementContentWhitespace	如果文本节点包含空白字符内容，则返回 true，否则返回 false	No	No	No	Yes
length	返回元素或属性的文本长度	6	1	9	Yes
wholeText	以文档中的顺序从此节点返回相邻文本节点的所有文本	No	No	No	Yes
方　法	**描　述**	**IE**	**F**	**O**	**W3C**
appendData()	向节点追加数据	6	1	9	Yes
deleteData()	从节点删除数据	6	1	9	Yes
insertData()	向节点中插入数据	6	1	9	Yes
replaceData()	替换节点中的数据	6	1	9	Yes
replaceWholeText(text)	使用指定的文本来替换此节点以及所有相邻的文本节点	No	No	No	Yes
splitText()	在指定的偏移处将此节点拆分为两个节点，同时返回包含偏移处之后的文本的新节点	6	1	9	Yes
substringData()	从节点提取数据	6	1	9	Yes

（4）CDATASection 对象和 Comment 对象

CDATASection 对象表示某个文档中的 CDATA 区段。CDATA 区段包含了不会被解析器解析的文本。一个 CDATA 区段中的标签不会被视为标记，同时实体也不会被展开。主要的目的是为了包含诸如 XML 片段之类的材料，而无须转义所有的分隔符。在一个 CDATA 中唯一被识别的分隔符是"]]>"，它可标示 CDATA 区段的结束。CDATA 区段不能进行嵌套。Comment 对象表示文档中注释节点的内容。CDATASection 对象和 Comment 对象的属性和方法相同，如表 5-17 所示。

表 5-17　CDATASection 对象的部分属性和方法

属　性	描　述	IE	F	O	W3C
data	设置或返回此节点的文本	6	1	No	Yes
length	返回 CDATA 区段的长度/ 可返回此节点的文本的长度	6	1	No	Yes
方　法	**描　述**	**IE**	**F**	**O**	**W3C**
appendData()	向节点追加数据	6	1	No	Yes
deleteData()	从节点删除数据	6	1	No	Yes
insertData()	向节点中插入数据	6	1	No	Yes
replaceData()	替换节点中的数据	6	1	No	Yes
splitText()	把 CDATA 分拆为两个节点	6	1	No	
substringData()	从节点提取数据	6	1	No	Yes

（5）DocumentType 文档类型对象

每个文档都包含一个 DOCTYPE 属性，该属性值可以是一个空值或是一个文档类型 DocumentType 对象。DocumentType 对象提供了一个用于定义 XML 文档的实体对象。

DocumentType 对象属性如表 5-18 所示。

表 5-18　DocumentType 对象属性

属　性	描　述	IE	F	O	W3C
entities	返回一个包含 DTD 声明实体的 NamedNodeMap[指定节点映射]	6	No	9	Yes
internalSubset	以字符串的形式返回内部 DTD	No	No	No	Yes
name	返回 DTD 名称	6	1	9	Yes
notations	返回一个包含 DTD 声明符号的 NamedNodeMap[指定节点映射]	6	No	9	Yes
systemId	返回用于确认外部 DTD 的系统	No	1	9	Yes

（6）parseError 对象

微软的 parseError 对象可被用来从微软的 XML 解析器中取回错误信息。在试图打开一个 XML 文档时，就可能发生一个解析器错误（parser-error）。通过 parseError 对象可取回错误代码、引起错误的行，等等。parseError 对象不属于 W3C DOM 标准，其属性如表 5-19 所示。

表 5-19　parseError 对象的属性

属　性	描　述
errorCode	返回一个长整数（LONG INTEGER）形错误代码
reason	返回包含错误原因的字符串
line	返回一个指明错误行数的长整数（LONG INTEGER）
linepos	返回一个指明错误位于哪个行位置的一个长整数（LONG INTEGER）
srcText	返回错误所在行的一个字符串
url	返回指向已加载文件的 URI
filepos	返回指明错误所在文件中的位置的一个长整数（LONG INTEGER）

ProcessingInstruction、DocumentImplementation 以及其他对象由于不常用，此处略。

3．NodeList 对象

NodeList 对象是一个节点的集合，它包含了某个节点中的所有子节点对象，可用于表示有顺序关系的一组节点（例如某个节点的子节点序列）。可用 GetNodeByName 方法返回节点的值。我们可通过节点列表中的节点索引号来访问列表中的节点（索引号由 0 开始）。节点列表可保持其自身的更新。如果节点列表或 XML 文档中的某个元素被删除或添加，列表也会被自动更新。在一个节点列表中，节点被返回的顺序与它们在 XML 被规定的顺序相同。NodeList 对象的属性 length 可返回某个节点列表中的节点数目，NodeList 对象的方法 item 可返回节点列表中处于某个指定的索引号的节点。

4．NamedNodeMap 对象

NamedNodeMap 对象也是一个节点的集合，利用该对象可建立节点名和节点之间的一一映射关系，从而利用节点名可以直接访问特定的节点。

NamedNodeMap 对象类似于 NodeList 对象，主要区别是：NamedNodeMap 通过名称来描述节点，而不是通过序数索引。另一个显著区别正如图 5-18 所示，NamedNodeList 只应用

于属性。NamedNodeList 也只有一个返回节点数量的 length 属性，而方法如表 5-20 所示。

表 5-20　NamedNodeMap 对象的方法

方　法	描　述	IE	F	O	W3C
getNamedItem()	可返回指定的节点（通过名称）	5	1	9	Yes
getNamedItemNS()	可返回指定的节点（通过名称和名称空间）			9	Yes
item()	可返回处于指定索引号的节点	5	1	9	Yes
removeNamedItem()	可删除指定的节点（根据名称）	6	1	9	Yes
removeNamedItemNS()	可删除指定的节点（根据名称和名称空间）			9	Yes
setNamedItem()	设置指定的节点（根据名称）			9	Yes
setNamedItemNS()	设置指定的节点（通过名称和名称空间）			9	Yes

5.7.3　XML DOM 编程实例

XML DOM 所包含的内容很多，有兴趣的读者可参阅 http://www.w3school.com.cn/xmldom/index.asp 和 http://msdn.microsoft.com/library/chs/default.asp?url=/ library/CHS/cpguide/html/cpconXMLDocumentObjectModelDOM.asp 获取详细资料。为引导读者迅速入门，下面我们通过具体的带有详细注解的例子来说明怎样使用 XML DOM 进行对 XML 文档的 操作。

【例 5.19】 首先建立一个 JavaScript 文件 ex_5_19.js 用于创建 Document 对象。内容如下：

```
function loadXMLDoc(dname) {  //创建 XML DOM Document 对象 xmlDoc
var xmlDoc;
if (window.ActiveXObject)  // 针对 IE 浏览器
   xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
// 针对 Mozilla, Firefox, Opera 等浏览器
else if (document.implementation && document.implementation.createDocument)
     xmlDoc=document.implementation.createDocument("","",null);
else
     alert('Your browser cannot handle this script');
xmlDoc.async=false;
xmlDoc.load(dname); //加载 XML 文档到 Document 对象
If (xmlDoc.parseError.errorCode <> 0) alert(("You have error:" & xmlDoc.
parseError.reason))
return(xmlDoc);
}
```

针对例 5.18 中的 ex_5_18.xml 文档，下列代码用 XML DOM 实现了对它的各种操作，请仔细阅读并理解其代码含义。

```
<html><head>
<script type="text/javascript" src="loadxmldoc.js"></script>
```

```
</head><body>
<script type="text/javascript">
//确认最后一个子节点为元素节点
function get_lastchild(n){
var x=n.lastChild;
  while (x.nodeType!=1){
    x=x.previousSibling;  }
return x;
}

 xmlDoc=loadXMLDoc("books.xml");       //装入 books.xml 文档内容到内存
 var root=xmlDoc.documentElement;      //选择 XML 文档根节点

 document.write("<h2>根节点: " + root.nodeName+"</h2>");

//1.返回第二个 book 元素节点中的 price 的文本值
var bookNode = root.childNodes[1];  //得到根节点下的第二个 book 元素节点
var priceNode = bookNode.childNodes[3]; //得到 book 节点下的 price 子节点,处于第
                                          //4 个位置。从 0 开始计数
var textNode = priceNode.childNodes[0]; //得到 price 元素的文本节点
var textValue = textNode.nodeValue;     //得到 price 元素的文本节点值
document.write(textValue+"<br/>") ;     //输出文本节点值

//2.如何遍历 XML 文档
var x=xmlDoc.getElementsByTagName('book'); //返回所有 book 元素的节点集合
for(i=0;i<x.length;i++)  { //对每一个 book 元素循环
   var bookChildNodeList = x[i].childNodes;
          document.write("第"+i+"个 book 节点下包含的元素: ");
        for(j=0;j<bookChildNodeList.length;j++){//对 book 节点下的各个子元素循环
           var elNode = bookChildNodeList.item(j);
          var textNode = elNode.childNodes[0]; //得到 title 元素的文本节点
          var textValue = textNode.nodeValue;  //得到 title 元素的文本节点值
          document.write(elNode.nodeName +":"+textValue+" ; " );
        }
        document.write("<br/>");
}

//3.获取 XML 文档中的所有书名
var x=xmlDoc.getElementsByTagName('title'); //得到所有 title 元素对象
for (i=0;i<x.length;i++)  {
  document.write(x[i].childNodes[0].nodeValue +"<br/>")//对每个 title 输出内容
  }
```

```
//4.获取 book 节点的属性 category 的值的第一种方法
var x=xmlDoc.getElementsByTagName('book');
for (i=0;i<x.length;i++)  {
  document.write(x[i].getAttribute('category')+"<br/>"); }

//5.获取 book 节点的属性 category 的值的第二种方法
var x=xmlDoc.getElementsByTagName('book');
for(i=0;i<x.length;i++)  {
  var att=x.item(i).attributes.getNamedItem("category");
  document.write(att.value + "<br />")
  }

//6.为 book 节点设置一个新的属性 edition 和属性值
var x=xmlDoc.getElementsByTagName('book');
for(i=0;i<x.length;i++)  {
  x.item(i).setAttribute("edition","FIRST");
  }
var x=xmlDoc.getElementsByTagName("title");
for (i=0;i<x.length;i++)  {
  //输出 book 中所有 title 和 edition 值
  document.write(x[i].childNodes[0].nodeValue);
  document.write(" - Edition:"+x[i].parentNode.getAttribute('edition')+"<br
  />");
  }

//7.修改 book 节点中 category 属性的值，即设置成新值,然后删除属性值
var x=xmlDoc.getElementsByTagName('book');
for(i=0;i<x.length;i++)  {
  x.item(i).setAttribute("category","BESTSELLER");
  }
for (i=0;i<x.length;i++)  {
  document.write(x[i].getAttribute('category')+"<br/>"); //输出所有属性值
  }
for(i=0;i<x.length;i++)  {
  y = x.item(i);
  y.removeAttribute('category');
  }
for (i=0;i<x.length;i++)  {
  document.write(x[i].getAttribute('category')+"<br/>"); //输出所有 category
  属性值为 null
  }

//8.删除末尾的<book>元素
document.write("book 节点的数量为:"+ xmlDoc.getElementsByTagName('book').
length +"<br />");
```

```
var lastNode=get_lastchild(root);
var delNode=root.removeChild(lastNode);
document.write("执行 removeChild 后的 book 节点数量:");
document.write(xmlDoc.getElementsByTagName('book').length);

//9.编辑文本节点内容
//得到 title 元素的文本节点
var x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
document.write("<br/>"+ x.nodeValue);
//从文本节点中删除从 0 开始的 9 个字符   Everyday Italian ==〉 Italian
x.deleteData(0,9);
//在文本节点中 0 位置开始添加字符 Italian ==〉 Easy Italian
x.insertData(0,"Easy ");
//替换文本内容中从 0 开始的 8 个字符为 Lovable Easy
x.replaceData(0,8,"Lovable Easy");
document.write(" ==> "+x.nodeValue +"<br/>");  // ==〉 Lovable Easy Italian

//10.在节点列表的末端添加一个节点(生成 book 的一个子节点)
var x=xmlDoc.getElementsByTagName('book');
var newel,newtext
for (i=0;i<x.length;i++)  {
  newel=xmlDoc.createElement('edition');
  newtext=xmlDoc.createTextNode('First');
  newel.appendChild(newtext);
  x[i].appendChild(newel);
  }
 document.write(root.xml+"<br/>");

//11.在指定的现存节点前添加一个新的子元素。增加一个 book 节点
var newNode=xmlDoc.createElement("book");  //新建元素节点 book
var newTitle=xmlDoc.createElement("title"); //新建元素节点 title
var newText=xmlDoc.createTextNode("A Notebook"); //新建文本节点
newTitle.appendChild(newText); //形成 title 元素的文本节点
newNode.appendChild(newTitle); //形成 book 节点的子节点
root.insertBefore(newNode,get_lastchild(root)); //放到 DOM 树末节点之前
document.write(root.xml+"<br/>");

//12.替换节点列表中的一个节点
root.replaceChild(newNode,get_lastchild(root));
</script>
</body></html>
```

5.8 XML 文档的安全性

XML 文档作为 ASCII 文件，已成为 Internet 上的信息交换格式，而与信息交换有关的一个重要问题是安全。XML 文档安全包括：

（1）机密性。用于保证只有预期的接收者阅读 XML 中预期的部分。要保证机密性，重要的是加密 XML。XML Encryption 是加密 XML 的标准。

（2）完整性。用于保证 XML 在从源到最终目的地的传输过程中不会被改变。XML Signature 标准允许发送者在要保证完整性的内容后面附加数字签名。

（3）真实性。用于保证 XML 确实是由声明发送它的人发送的。随内容一起发送的 XML 签名可以帮助确保发送者的身份。有效载明的接收者可以使用发送者的公开密钥验证数字签名。如果数字签名是有效的，身份就得到了确认，否则就不能确认。

（4）抗否认性。用于保证发送者不能否认发送了 XML。由发送者的私钥生成的附在 XML 上的 XML 签名保证了发送者的不可否认性。

1．XML 加密（Xenc）

除了在传送 XML 文件时采用标准进行加密，W3C 和 IETF 还制定了一项标准来对一个 XML 文档中的数据和部分内容进行加密。这样，如果一个文档只是某些敏感部分需要进行保护，你就可对它们单独进行加密。对于同一个文档中的不同部分用不同的密钥进行加密，你就可以把同一个 XML 文件发给不同的接受者，而接受者只能看见和它相关的部分。一旦采用这个方法对一个 XML 文件进行加密，在加密部分的首尾就会出现两个标记，表示该文件是以 W3C 公布的标准进行加密的。真实的标识名被替代；数据本身显示为一连串的密码。这个标准使 XML 数据提供者可以根据用户的不同对内容进行颗粒化的控制。而且由于只加密重要敏感数据而不加密整个文件，整个文件仍可以被 XML 处理器识别和处理。

2．XML 签名（XML-SIG）

XML 签名和 XML 加密紧密相关。和安全认证签名相似，XML 也是用于确保 XML 文件内容没有被篡改。为了适应各种文件系统和处理器在版式上的不同，XML 签名采用了标准化模式。这就使得 XML 签名可以适应 XML 文件可能遇到的各种环境。当对内容进行签名时，标准化模式使用文件里的数据和标识产生一个独一无二的签名，忽略了一些诸如段落结束或者制表符之类的次要信息。收到一个文件后，客户系统就开始进行“XML 签名解密转换”，它通过辨认信息是在标识前还是标识后来区分内容和签名：内容在标识后，而签名在标识前。通过比较运算结果和文件中的签名，可以确认数据的完整性。 XML 签名和 XML 加密结合在一起，可以确保数据发送和接收的一致性。

3．XML 安全相关开发工具

除了采用加密算法外，可以借助一些相关安全工具来确保 XML 文档的安全性。

（1）Apache security 提供了一个免费的符合 W3C 的 XML 加密标准的 Java 实施工具。

（2）IBM XML Security Suite 包括三种 XML安全工具：认证、数据加密、XML 访问控制。

（3）XML security library 是一个免费的 XML安全工具，用于 C 程序员。

（4）商业工具如 KeyTools 和 JCSI 等，为 XML 文档提供安全性。

（5）可利用微软.NET 的 System.Security.Cryptography.Xml 命名空间所提供的 EncryptedXml 和 SignedXml 类直接对 XML 文档进行加密或签名（需添加引用 System.Xml 命名空间）。有兴趣的读者可参阅 http://msdn2.microsoft.com/zh-cn/library/system.security. Cryptography.xml(VS.80).aspx 网站。

5.9 XPATH、XLINK 和 XPOINTER 简介

5.9.1 XPath

当我们采用 XSL 转换 XML 文档时需要处理其中的一部分数据，那么应该如何查找和定位 XML 文档中的信息呢，也就是说在模板声明语句 xsl:template match = ""和模板应用语句 xsl:apply-templates select = ""中怎样确定这两个引号中的内容，使得我们能够精确地找到需要的节点？XPath 就是一种专门用来在 XML 文档中查找信息的语言，换言之，如果将 XML 文档看做一个数据库，则 XPath 就是 SQL 查询语言。

1．XPath 节点的种类

XPath 的数据类型中包括七种可能的节点形式：根（root）、元素（element）、属性（attribute）、名称空间（namespace）、处理指令（processing instruction）、注释（comment）和文字（text）。

在 XML 文档中每个元素都有一个对应的元素节点。一个元素节点可以包含其他的元素节点、注释节点、处理指令节点与文字节点等。每一个元素节点都有一个相关的名称空间节点集。在 XML 文档中，名称空间是通过保留属性声明的，因此，在 XPath 中，该类节点与属性节点极为相似，它们与父元素之间的关系是单向的，并且不具有共享性。文本节点包含了一组字符数据，即 CDATA 中包含的字符。任何一个文本节点都不会有紧邻的兄弟文本节点，而且文本节点没有扩展名，只能在整体上对文本节点进行操作。

2．XPath 轴线

轴线是指上下文节点之间的关系。XPath 轴线定义了多种形式的节点轴线，如表 5-21 所示。

表 5-21　XPath 的节点轴线

序　号	节 点 轴 线	描　　述
1	Ancestor	当前节点的祖先节点，包括了当前节点的父节点，直至根节点
2	Ancestor-or-self	当前节点和当前节点的祖先节点
3	Attribute	当前节点的属性
4	Child	当前节点的子节点
5	descendant	当前节点的后代（即子节点，子节点的子节点，……）
6	descendant-or-self	当前节点及其后代
7	following	当前节点之后的所有节点
8	following-sibling	当前节点的所有兄弟节点
9	namespace	当前节点的名称空间节点
10	parent	当前节点的父节点
11	preceding	当前节点的所有前节点
12	preceding-sibling	当前节点的所有前兄弟节点
13	self	当前节点

3．基本的 XPath 表达式及其缩写形式

XPath 中最基本的表达式及其缩写形式如表 5-22 所示。在实际应用中就可使用缩写形式。

表 5-22　XPath 的基本表达式及其缩写形式实例

表　达　式	缩 写 形 式	描　　述
/	.	返回当前节点
Child::SUBJECT	SUBJECT	返回当前节点的子节点的 SUBJECT 元素
Child::*	*	返回当前节点的子节点的所有元素
Descend::TITLE	//TITLE	返回当前节点的后代的所有 TITLE 元素
Child::text()	text()	返回当前节点的子节点的文本节点
Attribute::COUNTRY	@COUNTRY	返回当前节点的 COUNTRY 属性
Child::SUBJECT/ child::TITLE	SUBJECT/TITLE	返回当前节点的 SUBJECT 元素的子元素 TITLE
/decend::STUDENT/ child::Name	//STUDENT/NAME	返回拥有父节点 STUDENT 的所有 NAME 属性
Child::STUDENT [position()=3]	STUDENT[3]	返回当前节点的子节点的第三个 STUDENT 元素
Child::STUDENT [Attribute::COUNTRY="CN"]	STUDENT[(@COUNTRY= "CN")]	返回当前节点的子节点属性 COUNTRY 的值为“CN”的 STUDENT 元素

4．XPath 函数

XPath 函数（又称谓词表达式）主要分为四组，即节点集型、字符串型、数值型和布尔型。

（1）节点集类型

所谓节点集就是节点的集合。在 XSL 中，一个表达式返回的结果往往是一系列的节点。要从节点集中选择一个或多个节点时，就需要使用关于节点集的函数，如表 5-23 所示。

表 5-23　节点集函数

序　号	函数名（参数）	描　　述
1	count(node-set)	返回节点集中的节点数
2	id(object)	使用 id 属性来选出符合要求的节点
3	last()	返回 XPath 节点或上下文大小数目
4	local-name(node-set)	返回节点集中第一个节点的本地名
5	name(node-set)	返回节点集中第一个节点的完全名称
6	namespace-uri(node-set)	返回节点集中第一个节点的名称空间的 URI
7	position()	返回当前节点的位置

（2）字符串类型

在 XPath 中，字符串使用的是 Unicode 字符，为了能方便地操作字符串，XPath 提供了大量的字符串函数，如表 5-24 所示。

表 5-24　字符串函数

序　号	函数名(参数)	描　　述
1	Concat(string,string,string*)	返回所有字符串连接起来的字符串
2	Contains(string, string)	判断第二个字符串是否包含在第一个字符串中
3	Normalize-space(string)	去掉字符串首尾空白部分，并用一个空格代替中间连续的空格

续表

序　号	函数名(参数)	描　　述
4	Starts-with(string, string)	判断第一个字符串是否以第二个字符串开始
5	String(value)	将数值转换为字符串
6	String-length(string)	返回字符串的长度
7	Substring(string,start,length)	返回字符串从 start 处开始长度为 length 的子串
8	Substring-after(string,substr)	返回从第一次出现 substr 之后到 string 最后一个字符的部分
9	Substring-before(string,substr)	返回从 string 的第一个字符开始到第一次出现 substr 的部分
10	Translate(value, str1, str2)	将 value 字符串中的 str1 用 str2 来代替

（3）数值类型

XPath 中的数值函数如表 5-25 所示。

表 5-25　数值型函数

序　号	函数名（参数）	描　　述
1	Ceiling(value)	返回比 value 值大的最小整数
2	Floor(value)	返回比 value 值小的最大整数
3	Number(string)	将字符串转化为数字
4	Round(value)	对参数进行四舍五入
5	Sum(node-set)	返回节点集中所有数值的和

（4）布尔类型

布尔值只可能为两个值，即 true 或 false。在 XSL 中，可以将任何类型的数据转变成布尔值，转换一般按下面的规则进行：

- 如果数值为 0 则布尔值为 false，其余的为 true；
- 空节点集为 false，反之为 true；
- 长度为 0 的字符串布尔值为 false，其余的为 true。

XPath 为我们提供了以下布尔型函数，如表 5-26 所示。

表 5-26　布尔型函数

序　号	函数名（参数）	描　　述
1	Boolean(object)	依据提供的对象返回一布尔型值
2	Not(object)	求反
3	True()	返回 true
4	False()	返回 false
5	Lang(string)	判断上下文节点语言

【例 5.20】 使用 Xpath 的简单例子。其中 XML 文档 ex_5_20.xml 文件内容为：

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="ex_5_20.xsl"?>
<root>
  <x>green</x>
  <y>
```

```
    <x>blue1</x> <x>blue2</x>
  </y>
  <z>
    <x>red1</x>  <x>red2</x>
  </z>
  <x>green</x>
</root>
```

ex_5_20.xsl 文件内容为：

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet version="1.0"
3       xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4    <xsl:template match="root">
5      <xsl:for-each select="//y" >
6        <xsl:value-of select="." />,
7        <xsl:if test="not(position()=last())">,</xsl:if>
8      </xsl:for-each>
9    </xsl:template>
10 </xsl:stylesheet>
```

该例中第 5 行、第 6 行和第 7 行均采用了 XPath 表达式，浏览器输出结果为"blue1 blue2 ,"。另外在 XML DOM 编程中也可使用 XPath 表达式，例如：

```
//演示了如何选取 price 值大于 35 的 price 节点
xmlDoc.selectNodes("/bookstore/book[price>35]/price") ;
```

5.9.2　XLink

XLink 可用来解决 XML 文档之间的链接问题。可以这样来理解：一个 XML 文档非常大，将它拆分成为多个 XML 文档，分散放在 Web 上不同的网络位置，那么如何处理这种情况？先看下面这个例子。

【例 5.21】 建有超链接的 XML 文档 ex_5_21.xml：

```
<?xml version="1.0"?>
<bookstore xmlns:xlink="http://www.w3.org.sixxs.org/1999/xlink">
  <book title="Harry Potter">
    <description
    xlink:type="simple"
    xlink:href="http://book.com.sixxs.org/images/HPotter.gif"
    xlink:show="new">
      As his fifth year at Hogwarts School of Witchcraft and
      Wizardry approaches, 15-year-old Harry Potter is...
    </description>
  </book>
```

```
  <book title="XQuery Kick Start">
    <description
    xlink:type="simple"
    xlink:href="http://book.com.sixxs.org/images/XQuery.gif"
    xlink:show="new">
      XQuery Kick Start delivers a concise introduction to the Xquery
      standard...
    </description>
  </book>
</bookstore>
```

从例 5.21 可以看出，ex_5_20.xml 文档中的两本书的封面图已放到由 xlink:href 指定的链接位置，当然这里也可改为另一个 XML 文档的名称。

在进行 XLink 设计前必须先声明一个名称空间。XLink 的名称空间通常采用如下的 URI 定义：xmlns:xlink="http://www.w3.org/1999/xlink"。

定义好名称空间后，就可以开始着手设计 XLink 了。XLink 包含了许多属性，常用的属性如表 5-27 所示。

表 5-27　XLink 常用属性

序　号	属　　性	描　　述
1	type	指定链接类型，值可为 simple 或者 extended
2	href	URI 链接目标
3	role	链接目标的功能
4	title	可人为读取的链接功能描述符
5	actuate	定义链接触发方式，值可为 onQuest 或 onLoad
6	show	定义链接目标资源被检索到时其显示方式，值可为 new, replace, embedded
7	from	定义 locator 或 resource 元素的 role 属性的取值，链接始点
8	to	定义 locator 或 resource 元素的 role 属性的取值，链接终点

5.9.3　XPointer

XPointer 用于在资源内定位片断（fragement），它支持在 XML 文档中定位元素、属性、字符串等内部结构。例如，我们可以定位到根元素或者当前元素的第 5 个子元素，也可以定位到文档中的某一个位置或两个位置之间的区域。

XPointer 采用基于 XSL 转换中的 XPath 语言，并在其基础上进行了扩展。XPointer 不再使用 XPath 中的节点这个概念，而改用位置（location），每一个位置可以是节点、点（point）或范围（range）。点就是 XML 文件内的某个位置，而范围则是两个点之间的所有文档内容。

XPath 只能定位一个节点，而 XPointer 除了定位一个节点外，还可定位点和范围；通过字符串匹配定位资源片段；在 URI 引用中定位资源片断。例如：

```
http://www.xxx.edu.cn/index.html/#xpointer(/child::*[position()=3])
```

表示在 URI 引用中定位资源片断。通过这种方式，可以在某个网站中引用另一个网站的某城市的天气预报，而不需要访问那个网站的关于天气预报的数据库。又如：

```
xlink:href=http://dog.com.org/dogbreeds.xml#XPointer(id('Rottweiler'))
```

通过 XLink 精确定位到 ogbreeds.xml 文档中的"Rottweiler"的 ID 值。

XPointer 中的范围是由起始点和结束点定义的，形式如下：

```
#Xpointer(<locatorElement>to<locatorElament>)   例如：
#Xpointer(/dir1/text()/start-point()[position()=1] to /dir1/text()/
start-point()[position()=5])
```

XPointer 提供的语法则是通过一系列扩展函数来实现的。XPointer 的扩展函数如表 5-28 所示。

表 5-28　XPointer 的扩展函数

序　号	函　　数	描　　述
1	End-point()	返回包含位置终点的位置集
2	Here()	返回包含指向当前元素的 XPointer 的元素
3	Start-point()	返回包含位置起点的位置集。例如 Start-point()[position()=12]
4	String-range()	查询目标文档。例如：String-range(//title,"hello",2,3)[6] 表示选择 title 元素中第 6 次出现的 hello 中的从第二个字符起的连续 3 个字符，即 ell
5	Unique()	相当于 count()=1。如果位置集包含在一个条目中，则返回 true
6	range-to()	查询某一范围。例如 XPointer(id("chap1")range-to(id("chap2"))) 表示定位范围是从 ID 属性值为 chap1 到 ID 属性值为 chap2 的范围

5.10　XML 与数据库

随着 XML 的广泛应用，各大数据库公司都将自己的产品重新设计，使之能够最大限度地支持 XML，如 DB2、Sybase、Oracle、Informix 数据库以及由 The Connection Factory 开发的 XHive、由 ozone-db.org 开发的 XML Repository 和 eXcelon 公司的 eXcelon，还有微软公司的 SQL Server 2005 和 SQL Server 2008 等。

微软于 2000 年 1 月宣布其 SQL Server 数据库对 XML 提供支持，就一直致力于将 XML 技术同其 SQL Server 相集成，以帮助建立下一代高效的基于 Web 的企业应用。该公司于 2008 年推出的新版本 SQL Server 2008，是一个完全支持 XML 的产品。

5.10.1　SQL Server 对 XML 的支持

SQL Server 2000 对 XML 的支持：

1．使用 SELECT 语句中的 FOR XML 子句提取 XML 数据。

例如：

```
select * from Northwind.dbo.customers  for xml auto
```

在一个 SELECT 语句中运用 FOR XML 子句，它有三种模式可以以不同的格式来返回 XML：RAW、AUTO 和 EXPLICIT。RAW 模式将结果中的每个记录作为一个普通的行元素来返回，它被包含在一个标签中，并将每个列的值作为一个属性。AUTO 模式将每个记录作为行元素返回，根据源表或视图名称对它的元素进行命名。如果查询从一个表返回多个列，那么每个列的值就会被作为表元素的属性来返回。但如果 SELECT 语句执行了合并操作，那么 AUTO 模式就代表的是子行，它们作为元素嵌套在父行下。EXPLICIT 模式有几个参数，可以通过这些参数来定义返回的 XML 的样式。可以为每个元素定义标签，明确确定数据是如何嵌套的。FOR XML 语句使我们不必再返回一个行记录集，再在客户端或中间层将它转换成 XML 了。

2．简单的 HTTP URL 请求。

例如：

```
http://localhost/web?sql=select * from Northwind.dbo.customers  for xml auto
```

实现此功能需要在“开始”|“程序”| Microsoft SQL Server |“在 IIS 中配置 SQLXML”菜单所打开的“用于 SQL Server 的 IIS 虚拟目录管理”对话框中进行配置。

要通过 HTTP 访问一个 SQL Server 数据库，首先在上述对话框中设置一个虚拟目录，这个虚拟目录在 HTTP 协议和一个特定的数据库之间提供了一个链接。在对话框的虚拟目录设置中指定虚拟目录的名称、物理路径、服务器名称、数据库名称和注册信息。一旦创建了一个虚拟目录，就可通过一个 URL 将查询发送到数据库了。假如设置了一个叫做 Northwind 的虚拟目录，并在浏览器中输入了查询 http://localhost/Northwind?sql= SELECT * FROM+Shippers+FOR+XML+AUTO+&root=Shippers，它就会返回相应的 XML 数据。与运用 ADO 或其他任何技术相比，HTTP 查询会让我们更容易地来访问网站或 Web 应用程序的数据。对于一个简单的查询语句来说，HTTP 查询会很好，但对于一个更复杂的查询来说，这种格式就会变得难以理解并很难管理了。此外这种方法也不安全，因为查询源代码是暴露给用户的。另外一种可选方法是在 HTTP 上调用一个模板查询。一个模板查询就是一个包含 SQL 查询的 XML 文件。模板作为文件保存在服务器上。因此，如果你在一个叫做 GetShippers.xml 的模板中封装了 Shippers SELECT 查询，那么 URL 查询的形式就会是：http://localhost/Northwind/templates/GetShippers.xml。模板也可以带有参数，当模板调用一个存储过程时，该功能会很有用。在 URL 查询和模板查询中，如果想从查询返回一个 HTML 页面，可以指定一个 XSLT 样式表，将它用于 XML。模板查询是读取数据的一个更安全的方法，它可以被缓存以得到更好的性能。

3．OPENXML。

OPENXML 函数可以让你像操作一个表那样来运用 XML 数据，可以将它们转换成内存中的一个行记录集。要运用 OPENXML，首先要调用 sp_xml_ preparedocument 存储过程，实际上，它将 XML 解析成一个数据树，并将那个数据的句柄传递到 OPENXML 函数。然后就可操作那个数据了，可以进行查询、将它插入到表中等操作。OPENXML 函数带三个

参数：用于 XML 文档内部显示的句柄、一个 rowpattern 参数和一个 flags 参数。Rowpattern 参数指定了应该返回原始的 XML 文档中的哪些节点。Flags 参数指定了以属性为中心的映射（结果集中列名符合属性名）或以元素为中心的映射（结果集中列名符合元素名）。在处理完 XML 数据后，我们可以调用 sp_xml_removedocument 将 XML 数据从内存中删除。

4．通过 SQLXML 得到更多的支持（需要安装 XML for SQL Server 2000 Web Release1 方可使用）。

通过发布 SQLXML，微软在 SQL Server 中提供了更多的 XML 支持。SQLXML 包含有 updategram 和 XML BulkLoad 功能。你可以在线下载最新版本的 SQLXML。可以通过基于 XML 的模板，运用 updategram 来插入、更新或删除表中的数据。updategram 提供了一个方法，使我们可以直接从 XML 更新 SQL Server 数据，这样就不用从 XML 文档得到数据，然后再用一个记录集或调用一个存储过程来处理了。updategram 只是可以简单地插入、更新或删除数据，如果需要查看一个值是否存在或在更新前查看一些商业规则，那么就应该用 OPENXML。

虽然可用 OPENXML 函数和 updategram 来插入数据，但对于加载大量的 XML 数据来说，这两种方法都不实用。可用 XML BulkLoad 将大量的 XML 数据插入到 SQL Server 表中。实际上可用 SQLXML BulkLoad 组件来加载数据，可从一个客户端应用程序来调用这个组件。在 BulkLoad 组件中，可以指定是否执行数据表检查约束（check constraint）、当插入数据时，是否应该锁定数据表，等等。

默认情况下，BulkLoad 不进行事务处理，但可指定所有加载的数据都是在一个单独的事务处理过程中，这样就可实现要么提交成功，要么回滚。如果用了事务处理，所有的数据在插入前都会被写进一个临时的文件。这就意味着，你需要足够的磁盘空间来保存临时文件，而且加载数据可能会很慢。但 XML BulkLoad 给我们提供了一个很好的方法，使我们可以将大量的数据写到 SQL Server 中；否则，你就必须提取数据，然后用另外的方法将它加载到数据库中。

根据具体实现情况，可以在 Web 应用程序中通过 HTTP 和 XSLT 的 XML 查询来替代标准的 ASP/ADO 数据访问，从而得到 HTML 输出结果，这种方法可以极大地提高性能。内容感兴趣的读者可进一步参阅参考文献[25]。

SQL Server 2005 对 XML 的支持：

在 SQL Server2005 中，FOR XML 功能新增了根元素和元素名称的新选项，使用 FOR XML 调用以便你可以建立复杂的层次关系的能力，和一个新的使得你可以定义将要使用 Xpath 语法来提取 XML 结构的 Path 模式。例如：

```
SELECT ProductID AS '@ProductID',
ProductName AS 'ProductName'
FROM Products
FOR XML PATH ('Product'), ROOT ('Products')
```

返回结果为：

<Products>

```
        <Product ProductID="1">
            <ProductName>Widget</ProductName>
            </Product>
            <Product ProductID="2">
                <ProductName>Sprocket</ProductName>
    </Product>
</Products>
```

除了增强了 SQL Server 2000 中的现有 XML 功能，SQL Server 2005 还添加了一种新的本地 xml 数据类型，此数据类型能够用于为 XML 数据创建变量和列，例如：

```
CREATE TABLE SalesOrders
 (OrderID integer PRIMARY KEY,
  OrderDate datetime,
  CustomerID integer,
 OrderNotes xml)
```

可以使用 xml 数据类型存储数据库中的标记文档或半结构化数据。列和变量可以用于非类型 XML 和类型 XML，其中后者是由 XML 架构定义 （XSD） 架构验证的。开发人员可以使用 CREATE XML SCHEMA COLLECTION 语句为数据验证定义架构，例如。

```
CREATE XML SCHEMA COLLECTION ProductSchema AS
'<?xml version="1.0" encoding="UTF-16"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- schema declarations go here -->
</xs:schema>
```

创建架构集合后，可以通过引用该架构集合并使用其包含的架构声明关联 xml 变量或列，如：

```
CREATE TABLE SalesOrders
(OrderID integer PRIMARY KEY,
OrderDate datetime,
CustomerID integer,
OrderNotes xml(ProductSchema))
```

在插入或更新值时，相关架构集合中的声明将验证类型 XML，出于符合或兼容性原因，有可能强制实施 XML 数据结构的业务规则。

xml 数据类型也提供了一些方法，这些方法可以用于在实例中查询和操纵 XML 数据。例如，可以在 xml 数据类型的实例中使用 query 方法查询 XML，如下面的示例所示。

```
declare @x xml
set @x=
'<Invoices>
<Invoice>
```

```
  <Customer>Kim Abercrombie</Customer>
  <Items>
   <Item ProductID="2" Price="1.99" Quantity="1" />
   <Item ProductID="3" Price="2.99" Quantity="2" />
   <Item ProductID="5" Price="1.99" Quantity="1" />
  </Items>
</Invoice>
<Invoice>
  <Customer>Margaret Smith</Customer>
  <Items>
   <Item ProductID="2" Price="1.99" Quantity="1"/>
  </Items>
</Invoice>
</Invoices>'
SELECT @x.query(
'<CustomerList>
{
for $invoice in /Invoices/Invoice
return $invoice/Customer
}
</CustomerList>')
```

这个例子中的查询使用了用于在文档中查找每个 Invoice 元素的 XQuery 表达式，并且从每个 Invoice 元素返回包含 Customer 元素的 XML 文档，其返回结果如下所示：

<CustomerList>

<Customer>Kim Abercrombie</Customer>

<Customer>Margaret Smith</Customer>

</CustomerList>

另一个在 SQL Server 2005 中引入的与 XML 相关的显著功能是支持 XML 索引。为了增强 XML 的查询功能，可以为类型 xml 列创建主 XML 索引和辅助 XML 索引。主 XML 索引是 XML 实例中所有节点的细化表示，查询处理器可以使用它快速查找 XML 值中的节点。创建主 XML 索引后，可以创建辅助 XML 索引改善特定查询类型的性能。下面的例子就是创建主 XML 索引和类型 PATH 的辅助 XML 索引，这可以改善使用 XPath 表达式识别 XML 实例中节点的查询性能。

```
CREATE PRIMARY XML INDEX idx_xml_Notes
ON SalesOrders (OrderNotes)

 CREATE XML INDEX idx_xml_Path_Notes
ON SalesOrders (OrderNotes)
USING XML INDEX idx_xml_Notes
FOR PATH
```

SQL Server 2008 对 XML 的支持：

SQL Server 2008 中与 XML 相关的主要增强功能包括：

（1）XML 架构验证增强功能

可以通过强制实施与一个或几个 XSD 架构符合的方法验证 XML 数据。架构为特定 XML 数据结构定义许可的 XML 元素和属性，并通常用于确保包括所有所需数据元素的 XML 文档使用正确的结构。

SQL Server 2005 通过使用 XML 架构集合引入了 XML 数据验证。一般的方法是通过使用 CREATE XML SCHEMA COLLECTION 语句创建一个包含 XML 数据架构规则的架构集合，然后在定义 xml 列或变量时，引用架构集合的名称，这些 xml 列或变量必须符合架构集合中的架构规则。这样，SQL Server 就会验证在架构集合的列或变量中插入或更新的、违反架构声明的任何数据。

SQL Server 2005 中的 XML 架构支持实现完整的 XML 规范的大子集，并且包含了大多数通用的 XML 验证场景。SQL Server 2008 扩展了该支持，使其包括以下已经由用户标识的附加架构验证要求：支持 lax 验证。即完全支持 dateTime、 time 和 date 验证，包括时区信息保护。改进了对 union 和 list 类型的支持。

XML 架构通过 any、anyAttribute 和 anyType 声明支持 XML 文档中的通配符部分。

```
<xs:complexType name="Order" mixed="true">
  <xs:sequence>
<xs:element name="CustomerName"/>
  <xs:element name="OrderTotal"/>
    <xs:any  namespace="##other"  processContents="skip"  minOccurs="0"
maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
```

此架构声明定义了一个命名为 Order 的 XML 元素，该元素必须包括命名为 CustomerName 和 OrderTotal 的子元素。此外，该元素还可以包含不限数量的其他元素，但这些元素应与 Order 类型属于不同的命名空间。下面的 XML 显示了 一个包含使用此架构声明定义的 Order 元素实例的 XML 文档。注意：Order 中还包含一个没有在架构中显式定义的 shp:Delivery 元素。

```
<Invoice xmlns=http://adventure-works.com/order
xmlns:shp="http://adventure-works.com/shipping">
  <Order>
    <CustomerName>Graeme Malcolm</CustomerName>
<OrderTotal>299.99</OrderTotal>
  <shp:Delivery>Express</shp:Delivery>
</Order>
</Invoice>
```

验证通配符部分依赖于架构定义中通配符部分的 processContents 属性。在 SQL Server 2005 中，架构可以对 any 和 anyAttribute 声明使用 skip 和 strict 的

processContents 值。在前面的例子中，通配符元素的 processContents 属性已经设置为 skip，因此没有尝试验证元素的内容。尽管架构集合包括对 shp:Delivery 元素的声明（例如，定义一个有效传递方法列表），但该元素仍然是未验证的，除非在 Order 元素的通配符声明中将 processContents 属性设置为 strict。

SQL Server 2008 添加了对第三个验证选项的支持。通过将通配符部分的 processContents 属性设置为 lax，可以对任何含有与它们相关架构声明的元素强制实施验证，但是忽略任何在架构中未定义的元素。继续前面的例子，如果将架构中通配符元素声明的 declaration 属性设置为 lax，并为 shp:Delivery 元素添加一个声明，则在 XML 文档中的 shp:Delivery 元素将被验证。然而，如果替换 shp:Delivery 元素，则文档就包含一个在架构中未定义的元素，此元素将被忽略。

此外，XML 架构规范定义了 anyType 声明，该声明包含 anyType 内容模式的 lax 处理。SQL Server 2005 不支持 lax 处理，因此 anyType 内容会被严格验证。SQL Server 2008 支持 anyType 内容的 lax 处理，因此该内容会被正确验证。

（2）XQuery 增强功能

SQL Server 2005 引入了 xml 数据类型，提供了用于对存储在列或变量中的 XML 数据执行操作的大量方法。可执行的大多数操作都使用 XQuery 语法导航和操纵 XML 数据。SQL Server 2005 支持的 XQuery 语法包括 FLWOR 表达式中的 for、where、order by 和 return 语句，这些语句可用于循环访问 XML 文档中的节点，也可用于返回值。

SQL Server 2008 添加了对 let 语句的支持，该语句用于向 XQuery 表达式中的变量赋值，如下面的示例所示：

```
declare @x xml
set @x=
'<Invoices>
<Invoice>
  <Customer>Kim Abercrombie</Customer>
  <Items>
   <Item ProductID="2" Price="1.99" Quantity="1" />
   <Item ProductID="3" Price="2.99" Quantity="2" />
   <Item ProductID="5" Price="1.99" Quantity="1" />
  </Items>
</Invoice>
<Invoice>
  <Customer>Margaret Smith</Customer>
  <Items>
   <Item ProductID="2" Price="1.99" Quantity="1"/>
  </Items>
</Invoice>
</Invoices>'

SELECT @x.query(
'<Orders>
```

```
{
for $invoice in /Invoices/Invoice
let $count :=count($invoice/Items/Item)
order by $count
return
<Order>
{$invoice/Customer}
<ItemCount>{$count}</ItemCount>
</Order>
}
</Orders>')
```

这个例子返回以下 XML。

```
<Orders>
    <Order>
        <Customer>Margaret Smith</Customer>
        <ItemCount>1</ItemCount>
        </Order>
        <Order>
        <Customer>Kim Abercrombie</Customer>
        <ItemCount>3</ItemCount>
    </Order>
</Orders>
```

（3）XML DML 增强功能

与使用 XQuery 表达式对 XML 数据执行操作一样，xml 数据类型通过其 modify 方法支持 insert、replace value of 和 delete 这些 XML DML 表达式。可以使用这些 XML DML 表达式操纵 xml 列或变量中的 XML 数据。

SQL Server 2008 添加了对使用 insert 表达式中的 xml 变量向现有 XML 结构插入 XML 数据的支持。例如，假定一个名称为 @productList 的 xml 变量包括以下 XML：

```
<Products>
  <Bike>Mountain Bike</Bike>
  <Bike>Road Bike</Bike>
</Products>
```

可以使用以下代码向产品列表中插入一个新自行车：

```
DECLARE @newBike xml
SET @newBike = '<Bike>Racing Bike</Bike>'
SET @productList.modify
('insert sql:variable("@newBike") as last into (/Products)[1]')
```

运行这段代码后，@productList 变量中会包括以下 XML。

```
<Products>
  <Bike>Mountain Bike</Bike>
```

```
    <Bike>Road Bike</Bike>
     <Bike> Racing Bike </Bike>
</Products>
```

5.10.2　XML 与数据库的互操作过程

首先我们应该明确 XML 不是数据库，数据库系统有它自己的一套管理模式，而 XML 仅仅是用来存放结构化数据的文件，在这一点相当于 XML 文档仅仅代表着数据库中的某一个表。因此 XML 不可能取代数据库，但将数据库和 XML 结合起来，能够完成很多以前无法完成的工作，例如异构数据交换、应用系统集成等。

开发一个访问数据库的 XML 应用系统需要同时借助 XML 编程接口和数据库编程接口，前者用于对 XML 文档的解析、定位和查询，所需技术包括 DOM 和 SAX；后者则是用于访问数据库，如数据库中数据的更新和检索等等，需要利用的技术有 ODBC、JDBC、ADO/ADO.NET 等。

XML 与数据库的互操作过程如图 5-19 所示。

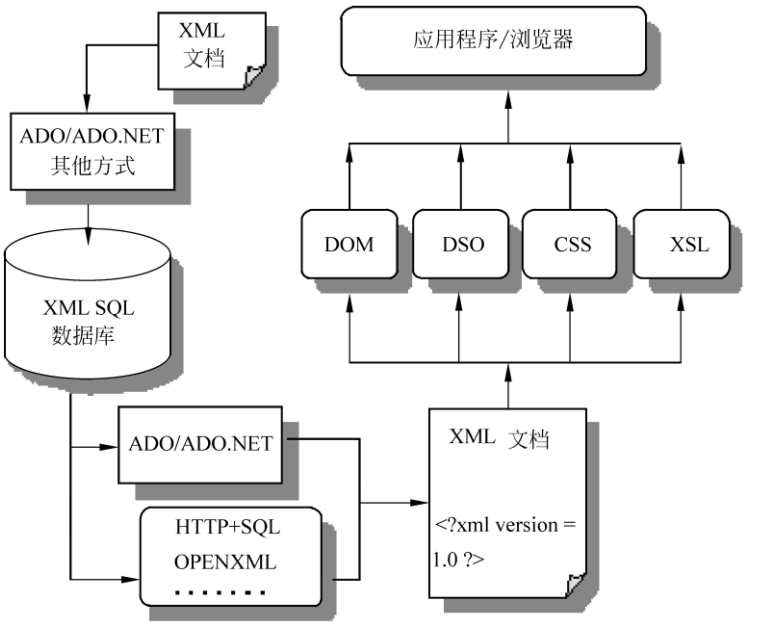


图 5-19　XML 与数据库的互操作过程

对于 XML 文档，可以通过 DOM 读取 XML 文档中的节点。如本章前面所述，DOM 是 W3C 的一种技术标准，实际上是提供一组 API 来处理 XML 数据，可以通过 JavaScript、JScript、VBScript 等脚本程序来调用，也可通过 C++、Java 等高级语言来实现。

其次，通过 DSO（Data Source Object）进行 XML 的数据绑定可以方便地将 XML 节点同 HTML 标记捆绑，从 XML 文档中读取或写入数据。DSO 的工作方式有几种，一种是同 DOM 类似，通过对 XML 节点树进行遍历来搜索节点，每次仅将节点数据同 HTML 的

一个元素（如 SPAN 元素）相关联；第二种同第一种的不同之处在于将节点数据同一个 HTML 多值元素（如 TR 元素）相关联。

样式单 CSS 和 XSL 实际上是通过给 XML 数据赋予一定的样式信息以使得其能够在浏览器中显示。CSS 技术早在 HTML 3.2 中就得以实现，其关键是将 HTML 中的元素同预先定义好的一组样式类相关联以达到样式化的目的，而 XML 同样也支持这种技术。XSL 同 CSS 有些类似，不同之处在于它是通过定义一组样式模板将 XML 源节点转换成 HTML 文档或其他 XML 文档。XSL 实际上也同样符合 XML 规范，它提供了一套完整的类似控制语言的元素和属性，最终可完成丰富多彩的样式描述。

在 ASP/ASP.NET 页面文档中嵌入 ADO/ADO.NET 对象，将数据库中的数据写入 XML 文档或将 XML 数据写入数据库是微软对数据访问技术的一种扩展。读者可参阅本书第 6 章 6.9 节“ASP.NET 中 XML 编程基础”部分。

5.11 XML 的应用和发展前景

虽然人们对 XML 的某些技术标准尚有争议，但是人们已经普遍认识到 XML 的作用和巨大潜力。下面对 XML 的应用和发展前景进行介绍。

5.11.1 XML 的应用

1．数据集成

企业在信息化过程中已建立了若干个独立的应用系统，例如考勤管理系统、人事管理系统、财务管理系统、库存管理系统，等等。各个系统可能是由不同的软件公司开发的，可能采用了不同的技术、运行在不同的平台上。但企业运作是一个整体，需要各个系统相互配合，因此应用系统间的数据交换成为困扰信息主管的一大难题。于是，可能会出现这样的尴尬局面：月初，考勤系统管理员将上月的员工考勤数据打包后复制或发送邮件传送给人事部门，财务部门也将员工所在部门的销售业绩统计打包传送给人事部门，而后人事部门运行一个批处理程序合并考勤数据和业绩统计，最后计算出员工工资。类似的情况在目前企业中比比皆是。企业缺乏一个顺畅的业务管理平台，不能将各部门的信息有机的集成在一起，势必造成管理上的混乱。

其实造成上述这种效率低下局面的原因就是各个系统没有统一的数据结构约定。其后果不但是效率低下，而且信息冗余，造成资源的巨大浪费。在这种情况下，XML 是解决这一问题的强大法宝。XML 将起到粘合剂的作用，通过它，使得各业务模块有机结合，数据交换畅通无阻，从整体达到理顺业务操作的目的。

数据集成的主要步骤为：

（1）要对整个业务进行调整，摒弃不合理部分。基于 XML 的数据集成不仅仅是要进行系统开发，对旧有系统的合理改造也是很重要的。

（2）对业务模式归纳总结，并从中抽象出 XML 数据交换模型，也即制定数据交换的 DTD 或 Schema。这是最基本的，但同时也是最为困难的一步。XML 消息流要符合企业的信息流。不要将 XML 看做是用来代替对象或者开发软件的新方法，它应该是一种表达层次结构信息并且在不同的应用系统间传输这种信息的有效途径。在制定 XML 数据交换模

型中，一个易犯的错误是直接照搬原来的数据格式而仅仅将其逐字逐句地“翻译”成 XML，毕竟这是一个改造旧系统的“工程”，去粗存精方是上策。

（3）结合制定好的 XML 数据交换模型，运用 XML DOM 和 SAX（Simple API for XML）等技术编写应用程序，也可直接在原系统上进行改造。

2．交易自动化

目前全球电子商务的发展非常迅速，各种行业甚至跨行业的 XML 电子商务规范与框架层出不穷，其中比较有代表性的是：Ariba 的 cXML、IBM 的 tpaML、CommerceOne 的 xCBL 2.0、Microsoft 的 BizTalk 框架、CommerceNet 的 eCo 计划、RosettaNet 的 eConcert 计划与 PIP 规范集以及联合国 UN/CEFACT 小组和 OASIS 发起的 ebXML 计划。

一个典型的应用是，开发这样一个智能代理程序：首先，该程序向某电子商务交易系统发出一个供货商资料查询请求，在得到应答后，自动连接答复中提供的所有供货商站点；然后，搜索预定商品的信息，并对获取到的不同商家针对该商品的价格、质量、服务等信息按一定的商业规则进行比较；最后，得出理想的结果，并自动向该站点下订单。所有这一切都离不开 XML 技术的应用。

3．设计标记语言

作为元标记语言，XML 为用户提供了定义本行业本领域标记语言的最好工具。目前这一应用的成功例子比比皆是，例如化学领域的 CML、数学领域的 MathML、移动通信领域的 WML 等。

4．文件保值

XML 良好的保值性和自描述性使它成为保存历史档案，如政府文件、公文、科学研究报告等的最佳选择。如放在数据库中保存，若干年后该数据库系统已不复存在或不能在新的计算下运行，文件只能成为垃圾了。

5．集成不同数据源

XML 文档可以用来描述包含在不同应用中的数据，从 Web 页面到数据库记录等，Web 应用的中间层服务程序将这些用 XML 表示的数据组合起来，然后提交给客户端或者下一步的应用。XML 还可以将多个来源的数据集成在一个文档内显示。

6．本地计算

XML 数据传输到客户端后，客户端可以利用 XML 分析器对数据进行解析和操作，在完成系统所需功能的同时，合理分配客户端和服务器的负荷。比如：数据库记录可以直接传输到客户端，然后再进行排序。

7．数据的多种显示

XML 将内容与表现分离，XML 只描述数据的结构和语义，显示外观则通过样式单文件（CSS 或 XSL）进行描述。因此，只需在显示时配置不同的样式单，即可实现多种显示效果。

8．网络出版

随着互联网的发展，网络已经成为一种新的媒体，人们在网络上发布各种信息，信息的发布形式和发布语言也多种多样，其中基于 XML 的显示技术和显示语言发挥了重要作用。比如 eBook、eNewspaper 等，就利用了 XML 的显示语言。

9．支持 Web 应用的互操作和集成

Web 界面定义语言（Web Interface Definition Language，WIDL）是 webMethods 定义的一个 XML 应用，它是一个能够用于 Web 的资源和企业应用接口的语言标准。通过它，Web 应用可以自动存取 Web 资源和企业应用。

除了上述应用外，XML 的应用还期待你有更多的发掘。

5.11.2 XML 的发展前景

XML 自推出以来，尤其是在 1998 年 2 月成为 W3C 推荐标准以来，受到了广泛的支持。各大软件厂商如 IBM、Microsoft、Oracle、Sun 等都积极支持并参与 XML 的研究和产品化工作，先后推出了支持 XML 的产品或者改造原有的产品以支持 XML，W3C 也一直致力于完善 XML 的整个理论体系。

XML 虽然获得了极大的支持，但是它还有一段路要走。首先，XML 的规则还有许多技术细节没有解决。其次，现在虽然出现了一些 XML 工具和应用，但是其市场反应还有待进一步观察。另外如何让更多的人迅速学会使用 XML，并利用它进行开发，进而促进 XML 的应用也是一个问题。XML 的出现和迅猛发展并不意味着 HTML 即将退出互联网舞台，由于 HTML 的易学易用和非常多的工具支持，HTML 将在较长的时间里继续在 Web 舞台上充当主角。但如果用户想超越 HTML 的范围，XML 将是最佳的选择。

另外，由于 XML 是用于自定义标记的元语言，任何个人、公司和组织都可以利用它来定义自己的标记语言，这虽然是 XML 的魅力和灵活性之所在，但同时也是 XML 的最大问题之所在。如果每个人、公司和组织都定义了自己的标记语言，它们之间的通信就会出现困难。因此在一些领域先后出现了一些标准化组织，它们的任务就是规范本领域的标记语言，形成统一的标准，使得在本领域内的通信成为可能。但在标准推出并得到广泛认可之前，各自为政的局面将继续下去。更糟糕的是，由于对应用的理解不一致和商业利益等原因，同一个领域也许还有多个标准化组织，它们形成的标记语言并不完全兼容，使得采取不同标准的计算机仍然难以通信。

无论如何，XML 的出现使互联网跨入了一个新的阶段，它已成为因特网领域中一个重要的数据交换标准和开发平台。没有 XML 就没有 Web 服务，也就没有今天构建应用软件的轰轰烈烈的 SOA（Service Oriented Architecture）。XML 的诞生已经而且将继续促使全新种类的基础架构和应用程序的产生，而这些新的基础架构和应用程序又将需要新的软件和硬件工具。可以预测，无论是在软件还是硬件上，XML 都将开辟一系列的新市场，促成互联网上新的革命。

思考练习题

1．简述什么是 XML，XML 与 HTML 有什么区别。
2．CSS 与 XSL 有什么区别？
3．什么是 XML 数据岛？如何使用？
4．XML DTD 与 XML Schema 有什么区别？
5．怎么理解 XML 安全性？XML 有哪几种安全标准？

6．XPath 和 XPointer 有什么区别？XLink 与 HTML 中的超链接有什么区别？
7．对本章的所有例子进行上机验证。