

第5章 文档对象模型

现在，应该开始研究 XML 文档的结构，以及如何利用它描述层次化信息。下面，我们将说明如何通过程序访问 XML 文档。其中一种方法是通过文档对象模型（Document Object Model，DOM）。在本章中，我们将介绍文档对象模型，并借助几个程序实例解释它的功能。

5.1 什么是文档对象模型

文档对象模型一词在 Web 浏览器领域并不陌生。窗口、文档和历史等对象都被认为是浏览器对象模型的一部分。然而，任何做过 Web 开发的人都知道各种浏览器实现这些对象的方式不尽相同。对于如何通过 Web 访问和操作文档结构这个问题，为了创建更加标准化的方法，W3C 提出了目前的 W3C DOM 规范。

W3C DOM 是一种独立于语言和平台的定义，即：它定义了构成 DOM 的不同对象的定义，却没有提供特定的实现，实际上，它能够用任何编程语言实现。例如，为了通过 DOM 访问传统的数据存储，可以将 DOM 实现为传统数据访问功能之外的一层包装。利用 DOM 中的对象，开发人员可以对文档进行读取、搜索、修改、添加和删除等操作。DOM 为文档导航以及操作 HTML 和 XML 文档的内容和结构提供了标准函数。

5.1.1 XML 文档结构

刚刚接触 XML 的开发人员常常会认为 XML 的主要目的是为文件中的信息片段命名，使之易于被其他人理解。结果，这些新手开发的文档简直如同“标记汤”——无序的数据元素列表与有意义的标记名称组合在一起，但是它与普通的文件一样都将信息置于同一层：

程序清单 5-1

```
<INVOICE>
  <CUSTOMER>Homer J. Simpson</CUSTOMER>
  <ADDRESS>142 Evergreen Terrace</ADDRESS>
  <CITY>Springfield</CITY>
  <STATE>VA</STATE>
  <ZIP>00000</ZIP>
  <PRODUCT1>Plutonium</PRODUCT1>
  <UNITS1>10</UNITS1>
  <PRODUCT2>Donuts</PRODUCT2>
  <UNITS2>937</UNITS2>
  <PRODUCT3>Beer</PRODUCT3>
  <UNITS3>1028</UNITS3>
  <PRODUCT4>Peanuts</PRODUCT4>
  <UNITS4>1</UNITS4>
</INVOICE>
```

许多开发人员都忽略了 XML 能够显示元素之间的关系这一特性——特别是表示两个元素的

父子关系。如果将上述文件改写为以下形式，将产生更好的效果：

程序清单 5-2

```
<INVOICE>
  <CUSTOMER NAME="Homer J. Simpson"
    ADDRESS="142 Evergreen Terrace"
    CITY="Springfield"
    STATE="??"
    ZIP="00000">
  <LINEITEM PRODUCT="Plutonium"
    UNITS="10"/>
  <LINEITEM PRODUCT="Donuts"
    UNITS="937"/>
  <LINEITEM PRODUCT="Beer"
    UNITS="1028"/>
  <LINEITEM PRODUCT="Peanuts"
    UNITS="1"/>
</INVOICE>
```

在这种形式的文档中，发票元素显然包括四个行式项目子元素。它还简化了文档的搜索——如果我们要寻找钚的所有订单，可以查询PRODUCT属性值为“Plutonium”的LINEITEM元素——而不必依次查看每个PRODUCT元素。

以上文档结构可以用图5-1中的节点树表示，它显示了所有元素以及它们之间的相互关系。

如果要给文本文件中的发票增加行式项目，必须读取文件直至发票的最后一个行式项目的末尾，插入新的行式项目文本，然后继续处理文档的后续部分。正如你所料，这种技术很快会变得非常棘手，特别是当节点树变得越来越深时。然而，如果你能够根据树结构以节点形式对文档进行操作，添加行式项目就轻而易举了——只需创建新的LINEITEM节点，并将它作为INVOICE节点的子节点。

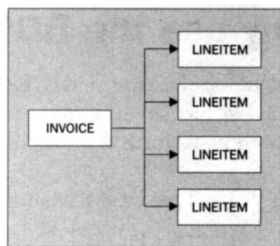


图 5-1

这就是DOM的工作原理。

当你使用DOM对XML文本文件进行操作时，它首先要解析文件，将文件分解为独立的元素、属性和注释等。然后，它以节点树的形式（在内存中）创建XML文件的表示。此后，开发人员可以通过节点树访问文档的内容，并根据需要修改文档。

事实上，DOM执行了更进一步的操作，它将文档中的每个项目看作节点——元素、属性、注释、处理指令，甚至构成属性的文本。因此，对于我们上面的例子，DOM实际上会将文档表示为图5-2所示的形式。

DOM提供了强大的接口集合，以简化对 DOM节点树的操作。

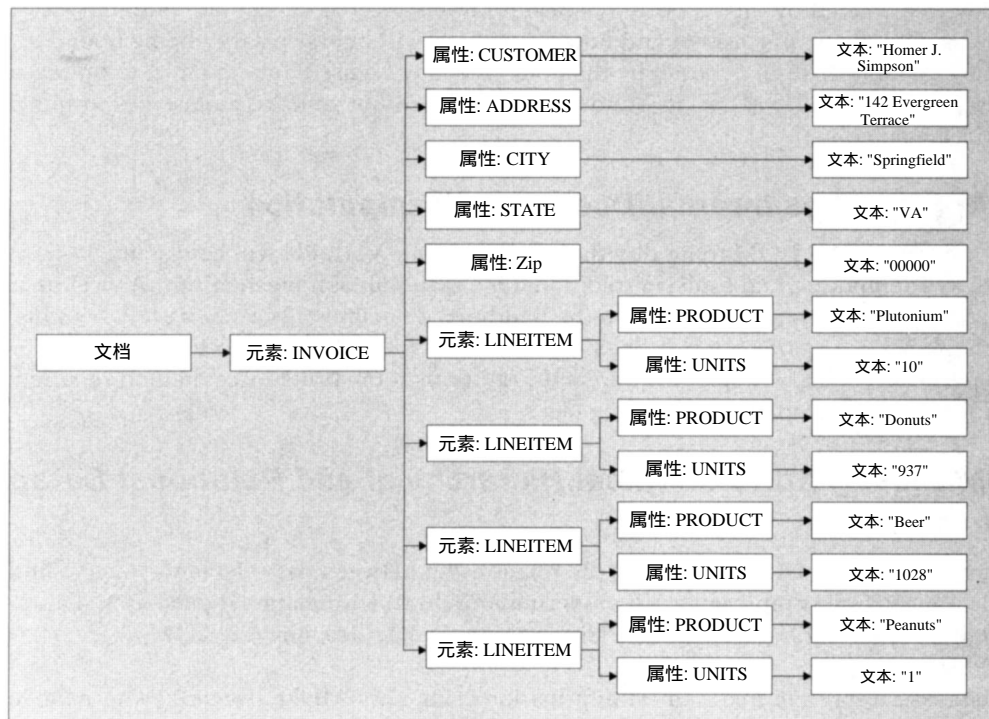


图 5-2

5.1.2 为何使用DOM

提到访问XML文件，DOM永远只能算作可选的访问方法之一。与其他用于产生XML文档的机制相比，例如：直接向一个流写入，使用DOM有以下几点优越性：

- DOM能够保证正确的语法和格式正规性。
- DOM能够从语法中提取内容。
- DOM能够简化内部文档操作。
- DOM能够贴切地反映典型的层次数据库和关系数据库的结构。

让我们依次分析这几个优点。

1. DOM能够保证正确的语法和格式正规性

由于DOM将文本文件转化为抽象的节点树表示，因此能够完全避免无结束标记和不正确的标记嵌套等问题。使用DOM操作XML文档时，开发人员不必担心文档的文本表示——只需要关注父子关系和相关的信息。另外，DOM能够避免文档中不正确的父子关系。例如，一个Attr对象永远也不能成为另一个Attr对象的父对象。

2. DOM能够从语法中提取内容

由DOM创建的节点树是XML文件内容的逻辑表示——它显示了文件提供的信息，以及它们之间的关系，而不受限于XML语法。例如，节点树蕴含的信息可以用于更新关系数据库，或者

创建HTML页面——开发人员不必纠缠于XML语法规范。

3. DOM能够简化内部文档操作

就修改XML文件的结构而言，使用DOM比使用传统的文件操作机制更加简单。正如我们在前面的例子中所描述的，利用DOM在文档中增加元素非常简便。另外，你可以通过几条命令执行全局性操作（例如：从文档中删除具有特定标记名称的所有元素），而不必采用繁琐的方法——首先对文件进行扫描，然后删除相关的标记。

4. DOM能够贴切地反映典型的层次数据库和关系数据库的结构

DOM表示数据元素关系的方式非常类似于现代层次型和关系型数据库表示信息的方法。这使得利用DOM在数据库和XML文件之间移动信息变得相当简单。

大部分数据库都使用“雪花”结构表示层次型信息，数据库中的信息从中心“顶级”表向外辐射，类似于车轮的车条（参见图5-3）。

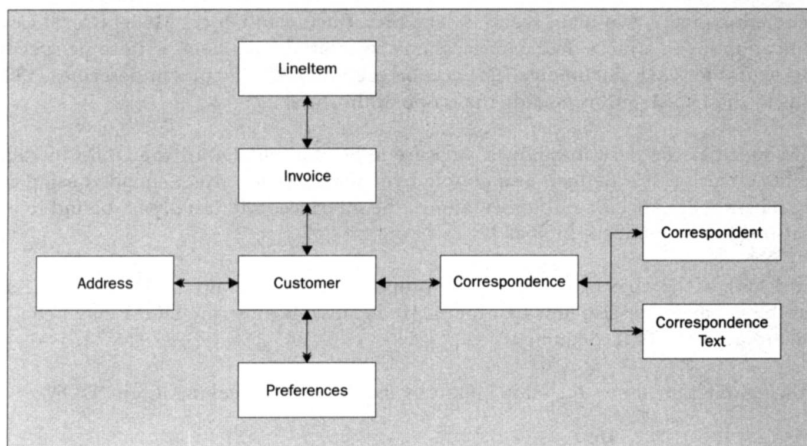


图 5-3

注意，每个客户可能有多张发票，而且每张发票可能包含多个行式项目。为了支持这种行为，XML允许我们包含一个子元素的多个拷贝。上图中的每个元素基本上都对应着一张表，每张表都包含多列（这些列在XML中作为属性出现）。

上图的结构等价于以下XML文件：

程序清单 5-3

```
<CUSTOMER>
  <INVOICE>
    <LINEITEM/>
  </INVOICE>
  <ADDRESS/>
  <CORRESPONDENCE>
    <CORRESPONDENT/>
    <CORRESPONDENCETEXT/>
  </CORRESPONDENCE>
  <PREFERENCES/>
</CUSTOMER>
```

利用DOM建立文档的树结构简化了系统之间的信息传递。

5.1.3 DOM规范

与其他Internet标准一样，DOM规范也是由W3C维护的。在编写本书时，W3C提出了两个DOM文档——Level 1和Level 2文档。

1. DOM Level 1

W3C DOM Level 1文档处于建议的状态。这意味着W3C已经审阅过它，接受了成员对它的注释，并且经过修订，正在将它提升为WWW的标准。<http://www.w3.org/TR/REC-DOM-Level-1/>提供了建议的完整文本。

Level 1文档包含两个主要部分。第一部分，文档对象模型（核心）Level 1定义了用于访问任何结构化文档的接口，以及用于访问XML文档的特殊扩展。文档的第二部分描述了DOM针对HTML的扩展，它超出了本书的讨论范围。

DOM规范通过定义数据类型DOMString描述了DOM如何操作字符串。该数据类型定义为双字节字符集，采用UTF-16编码机制进行编码。对于特定的实现，接口通常被绑定到也采用UTF-16编码的系统数据类型，例如：Java的String类型。

下面让我们来看看构成DOM Level 1规范的对象、方法和属性。需要注意的是，其中描述的行为仅仅适用于XML文档；当用于访问HTML文档时，DOM将有不同的行为。

图5-4显示了构成DOM的对象类层次：

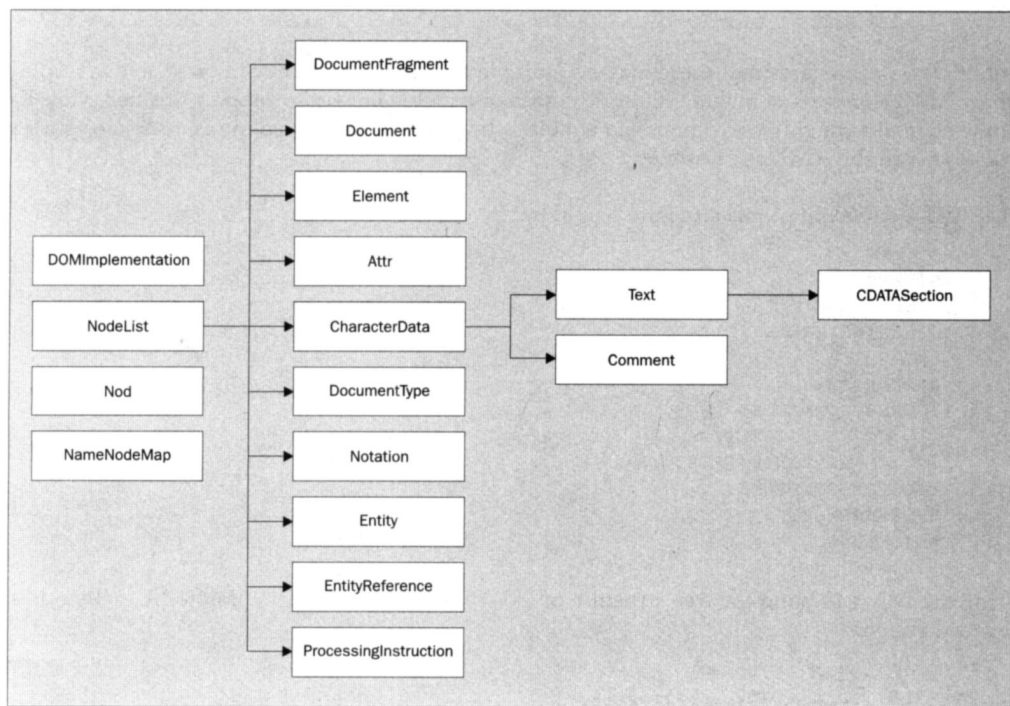


图 5-4

附录B提供了DOM对象的完整文档。

2. DOM Level 2

在编写本书时，W3C DOM Level 2规范正处于候选建议的状态。这意味着直接负责该规范的技术团体已经对它进行了重要的审阅，现在它需要的是真正的实现，以及来自 W3C之外各方面的技术反馈。Level 2规范不仅包含上述所有对象，而且新增了以下特征：

- 支持命名空间——正如我们将在第7章中看到的，命名空间用于区分 XML中具有相同名称的离散数据元素。它们通常提供返回原始的 XML结构文件的链接，该文件包含某种格式的元素信息。DOM Level 2将提供查询和修改文档命名空间的机制。
- 样式表——DOM Level 2包含样式表的对象模型，以及用于查询和操作特定文档的样式表的方法。
- 过滤——DOM Level 2新增了用于过滤XML文档内容的方法。
- 事件模型——DOM Level 2计划提供XML的事件模型。
- 范围（Range）——DOM Level 2包含用于操作大块文本的函数，它有助于在 XML中处理传统的文档。

<http://www.w3.org/TR/DOM-Level-2/>提供了DOM Level 2规范的W3C候选建议的完整文本。

3. 理解IDL和绑定

阅读DOM规范时，许多开发人员有可能对接口定义的方式感到陌生。记住，W3C将DOM定位为独立于平台的，即：W3C指定了特定系统的实现需要提供哪些方法和属性，但没有详细说明如何获得这些实现。为此，W3C选择通过以下几种方式表达与DOM的接口：通过OMG IDL（它是CORBA 2.2规范的一部分），或者通过Java和ECMAScript相结合。当开发人员使用非W3C规范的DOM实现应用程序时，应该参考随类库一起提供的特定实现的文档。例如，Microsoft在<http://msdn.microsoft.com/xml/reference/xmldom/start.asp>提供了他们实现的XML DOM的文档。

5.1.4 现实世界中的DOM

到目前为止，我们已经从纯理论角度介绍了DOM——我们从W3C通用建议的角度讨论了DOM。下面，看看DOM在现实世界是如何实现的。

当前浏览器中的DOM

在编写本书时，只有Internet Explorer 5内置了DOM类库，并且支持XSL。客户端代码可以引用页面中的数据岛——内嵌在HTML文档中的XML文档——通过文档中数据岛对象的XMLDocument属性能够获得该引用，同时通过该引用能够调用Microsoft类库支持的所有功能。这些类库支持DOM Level 1规范定义的全部功能，另外，它提供的扩展函数能够简化XML数据访问，并操作XSL样式表。

Netscape尚未提供对DOM的内置访问（至少到4.7版为止是这种情况），但是如果你有一组ActiveX或Java DOM类库，就能够在客户端利用Java或JavaScript对XML文档进行访问和操作。需要注意的是，在访问XML文档之前，你的客户端需要下载并安装这些类库。现在正在开发的Netscape的下一版本将内置对XML和XSL的支持。

Microsoft实现的ActiveX和Java类库形式的DOM可以从地址<http://msdn.microsoft.com/>

downloads/tools/xmlparser/xmlparser.asp免费下载。

判断浏览器类型

除非你能够控制应用程序的目标浏览器（例如：你正在为公司开发 Intranet应用程序，并且公司要求使用Internet Explorer），否则保证代码在各种浏览器上的通用性是至关重要的。为了实现这一目标，同时不以牺牲程序的功能为代价，你的代码应该在要执行特定浏览器才具备的功能之前查询浏览器的类型。

让我们看下面的例子，它是用 JavaScript编写的。你可以将这个代码片段包含在你的 Web 站点的缺省页面中：

程序清单 5-4

```
<SCRIPT>
var agent = window.navigator.userAgent;
var explorer = agent.indexOf ("MSIE 5");
if (explorer > 0)
{
    window.location.href = "http://www.mycompanywebsite.com/defaultie.htm";
}
else
{
    window.location.href = "http://www.mycompanywebsite.com/defaultns.htm";
}
</SCRIPT>
```

以上代码通过检查导航（navigator）对象的userAgent属性，判断用于访问页面的浏览器的类型。只有Microsoft的Internet Explorer 5.X在userAgent属性中内置了字符串“MSIE 5”。通过检查属性中是否包含该字符串，脚本能够确定所用的浏览器是否是 Internet Explorer 5，并根据判断的结果进行适当的重定向。例如，页面 defaultie.htm（专用于IE 5浏览器）中可能包含要使用IE 5客户端XML DOM 特征的JScript代码，而页面defaultns.htm（专用于非IE 5浏览器）可能依赖于服务器端的表单处理实现相同的目的。注意，随着新版本的出现，以上代码可能要根据各种浏览器客户端传递的userAgent字符串进行适当调整。

5.1.5 特殊的XML DOM实例——HTML DOM

要理解HTML和XML之间的差异，最重要的是理解它们共同的发展根源：SGML。

1. HTML不是XML

SGML是通用标记语言标准（Standard Generalized Markup Language）的缩写。SGML是第一个被广泛使用的标记语言系统，如今在许多商业领域仍然广泛使用，特别是需要频繁处理文档的领域（例如：出版公司）。与HTML这种纯粹的标记语言不同，SGML实际上是一种定义标记语言的方式。SGML的这一特征与XML非常类似，但是SGML嵌套标记和定义信息的方式更加灵活。HTML是一种特殊的SGML实现，它遵守SGML的语法规则，但是它包含允许在HTML文档中出现的特定的元素和属性定义。实际上，有一个SGML的DTD（文档类型定义）描述了HTML的词汇表和规则。地址<http://www.w3.org/TR/html40/sgml/dtd.html>提供了这个DTD的完整内容。

另一方面，XML是SGML的直接子集。XML设计的主要目标是创建一种能够保持 SGML 灵活性但更加易于解析的标记定义语言。因此，许多在 SGML中有效的结构对于XML文档来说是无效的（因此HTML也不是有效的XML文档）。

2. HTML DOM存在的问题

就HTML DOM为开发人员提供的访问HTML文档内容的方式而言，它不是十分灵活。特别是由于它要支持被非正式地称为“DOM Level 0”的一组特征——Internet Explorer 3.0和Netscape 3.0中实现的功能，它是在标准化用于定义HTML页面行为的对象模型之前提出的。仅仅为了向后兼容，HTML DOM必须实现一些功能。另外，HTML DOM支持SGML的特殊实现（依赖于是否存在用于说明文档可能布局的预定义 DTD）。

3. Internet的未来：XHTML

W3C正在研究一个关于XML实现的建议——XHTML。它将遵守XML的所有语法规则（正确嵌套的元素，引号包含的属性，等等），同时它也符合HTML的词汇表（可用的元素和属性，以及它们之间的关系）。虽然目前尚未出现专门支持XHTML的浏览器，但是HTML解析器和XML解析器都能够解析正确的XHTML文档。当你编写HTML时，最好遵守以下规则，使得文档也能够被XHTML的规则正确解析：

- 文档必须是格式正规的，所有元素必须正确结束（每个 <TAG>必须有对应的</TAG>，或者对于空元素，必须在起始标记的末尾增加“/”以表示结束）。
- 元素必须正确嵌套——即：每个结束标记必须作为最近的处于打开状态的标记的结束符（不允许交错）。
- 元素和属性名称必须是小写的。
- 空元素必须有结束标记，或者以“/”结尾的起始标记。
- 属性-值对必须显式定义。
- 脚本元素和样式元素应该包含在CDATA部分，以免被错误地解析。
- id属性应该用于保存元素标识符。

下面我们详细说明这些规则。

(1) 文档必须是格式正规的，所有元素必须正确结束，元素必须正确嵌套

HTML的某些元素不需要显式结束，例如：段落元素 <P>。解析器通过检查后面的元素判断段落元素应该在何处结束。例如，以下HTML片段在XHTML中是无效的：

```
<p>This is the first paragraph.  
<p>This is the second paragraph.
```

在XHTML中，应该使用以下代码：

```
<p>This is the first paragraph.</p>  
<p>This is the second paragraph.</p>
```

(2) 元素和属性名称必须是小写的

XML是区分大小写的，它将仅仅大小写不同的标记视为不同的标记。XHTML DTD规定所有元素和属性都应该使用小写标记，XHTML文档必须遵守这条规则。

因此，对于以下代码：


```
<INPUT TYPE="button" ID="button1" VALUE="Click me!" />
```

应该改为：

```
<input type="button" id="button1" value="Click Me!" />
```

注意，我们在标记结尾增加了斜杠。这就是下一条规则。

(3) 空元素必须有结束标记，或者在起始标记末尾增加“ / ”

在XML中，所有元素都必须是闭合的；如果某个元素定义为 EMPTY（它只可能包含属性，而不会有其他相关的信息），它可以以独立的结束标记结束，或者在起始标记的末尾增加斜杠。当然，在HTML中允许某些元素不指定结束标记，例如 <P>——遇到的下一个起始标记意味着前一个标记块的结束。为了尽可能与以前的浏览器兼容，应该使用独立的结束标记表示整个标记的结束，而不应该使用带斜杠的起始标记——有些浏览器不能正确地解析末尾的斜杠。

(4) 属性-值对必须显式定义

如果元素有缺省的属性，指定该属性的值时必须包含属性的名称。

以下代码：

```
<dl compact>
```

应该改为：

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

(5) 脚本元素和样式元素应该包含在 CDATA部分中，以免被错误地解析

由于script和style元素在XHTML DTD中被声明为 #PCDATA元素，因此解析器会将<和&视作标记的开始。为了避免这种误会，script和style元素应该包含在CDATA部分中，使它们能够直接包括非转义文本。

以下代码：

```
<script>
```

```
...
```

```
</script>
```

应该改为：

```
<script>
```

```
<![CDATA[
```

```
...
```

```
]]>
```

```
</script>
```

(6) id属性应该用于保存元素标识符

在XHTML DTD中，id属性的类型是ID；因此，应该使用id属性而不是name属性标识元素。在将来的XHTML实现中，name属性将被逐步淘汰。

由于XHTML是一种正在形成的标准，目前许多可视化的HTML编辑工具都不支持这些规则，因此如果使用它们编辑文档，很可能无意中破坏XHTML文档的正确性。在你创建XHTML文档时，要牢记这一事实。当你对编辑工具表示怀疑时，最好使用文本编辑器修改文档，而不要使用FrontPage或Microsoft Visual InterDev等工具。

谨遵这些建议，你将保证你的文档与未来转向XHTML的浏览器最大限度的兼容。另外，你所创建的XHTML文档可能会以MIME类型的HTML或XML的形式发送给客户端，使得客户端在

文档的显示和操作上拥有最大程度的灵活性。客户端可以直接使用 XML DOM 操作 XHTML 文档的内容。

5.2 使用DOM

我们已经讨论过 DOM 是如何结构化的，它将 XML 文档转化为可以通过程序访问的节点树。我们也说明 DOM 规范仅仅描述了访问机制，而不涉及特定的实现。但是，我们如何利用这些信息，将它们应用于特定的问题？为此，我们需要使用 DOM API。

5.2.1 DOM API

编写通过 DOM 访问 XML 文件的软件时，必须使用特定的 DOM 实现。实现是某种形式的类库，它设计为运行在特定的硬件和软件平台上，并访问特定的数据存储（例如：文本文件，关系数据库，等等）。

1. 什么是API

应用程序编程接口（Application Programming Interface, API）。不要受接口一词的误导——API 实际上是一组类库，一个组件利用它指示另一个组件执行更底层的服务。同样地，API 必须是一个接口的实现，它包含适当的代码用于连接其他组件，并指示它们执行相应的功能。

DOM并不是API

如前所述，W3C DOM 仅仅提供了 DOM 类库的接口定义，而没有提供特定的实现。为程序员提供 DOM 实现的任务要由第三方完成。当你打算使用 DOM 操作应用程序中的 XML 结构时，对于应用程序的每个目标平台，都要获取相应平台的 DOM 实现。在大多数情况下，这些类库要与你的应用程序绑定，并与应用程序的二进制代码一起分发。

需要注意的是，与 HTML 解析器的实现类似，DOM 的实现应该声明它所遵守的 W3C 规范。考虑到在本书出版之际 W3C 规范的状态（DOM Level 1 是一个建议，DOM Level 2 是一个候选建议），DOM 的所有实现至少要提供 Level 1 文档中描述的功能。许多 DOM 实现还提供附加的功能——Level 2 文档中描述的行为，或者特定实现的开发者认为有用的附加行为。例如，Microsoft DOM 支持 Level 1 规范的所有内容，除此之外还提供其他导航方法，以及用于支持样式表的方法和属性，等等。与其他开发工作类似，在决定是否利用特定的 DOM 实现提供的附加功能之前，必须考虑你的目标平台。

2. XML数据结构编程

使用 DOM 访问 XML 节点树中的信息时，最好围绕着 DOM 提供的访问机制设计系统。举例来说，如果你使用面向对象的数据库，可以根据 XML 元素定制相应的对象。如果你知道要使用哪些元素，可以创建封装其他对象的对象，并以尽可能最有效的方式在内存中复制 XML 树——我们在前面介绍了一个有关“雪花型”数据库的例子。

另外，你应该记住 XML 文件常常相当大，因此内存管理变得至关重要。良好的 DOM 实现能够在需要时提供即时的元素提取（读取元素信息时，这是以更长的搜索和获取时间为代价的），同时它应该提供良好的内存管理，避免超过内存容量或者交换失败。程序应该监视系统资源，并设定阈值，当超过阈值时，不允许 DOM 访问更多的文件，直至系统资源被释放。如果仅仅由

于文件过大而导致采用 DOM方式的实现无法处理，则只能转向 SAX等事件驱动的解析器（第6章将讨论SAX）。

5.2.2 客户端和服务端

虽然有许多 DOM和XML应用程序，但是它们基本上可以分为两种类型：用于服务器端的（或者在一个可控的环境中，例如：客户机-服务器系统）和用于客户端的。针对每种类型，我们将讨论DOM的一些潜在应用。

1. 服务器端的DOM

由于Internet开发人员能够对服务器上的软件进行更多的控制，因此 DOM的应用通常首先是在服务器端的。DOM能够极大地简化不同商业系统之间的数据交换，并且为数据的存档和获取提供了理想的机制。

(1) 文档交换

XML在企业中的首要应用将是简化过程之间以及业务之间的通信。与普通的文件或数据库等传输格式相比，XML有许多优势：

- XML文件是独立于平台的——与Access数据库或SQL Server不同，XML文件实际上能够被任何系统读取和理解，系统只要读取文档，并利用系统上的 DOM实现将文档解析为节点树。
- XML文件是自描述的——与普通的文件（它需要程序员协商文件的格式描述，并进行适当的转换）不同，设计良好的XML文件只需要很少的外部解释文档——每个描述作者的元素被明确地标记为<author>，依此类推。
- XML文件显示层次化信息——普通的文件可能包含重复的子元素组（例如：某个作者写的书），而XML文件设计为以一种自然的方式表达层次信息——通过节点树。举例来说，如果XML文件包含7位作者和22本书，只需要遍历由DOM创建的节点树，就立即能够了解哪本书是与哪位作者相关的。

目前，软件界有一种强大的趋势——将用于业务之间信息传递的XML格式标准化。BizTalk（www.biztalk.org）和XML Mortgage Partners（www.xmlmortgage.org）等组织正在创建DTD、模式和数据字典，它们将有助于提高不同业务之间信息交流的效率。

(2) 存档

如果要将信息存档，XML是一种理想的存储方式——特别是当这些信息是面向对象的或者来自于层次型数据库。关系型数据库通常也（但不总是）比较容易表示为 XML节点树。由于XML文件是基于文本的，而且可能包含许多重复的文本（标记），因此常常有较高的压缩率。一个典型的相当大的XML文件能够压缩到原来大小的十分之一至二十分之一。通过遍历数据库的层次树或关系树，并使用 DOM构造适当的节点树，很容易将整个信息集存档到一个 XML文件中。

以一个发票系统为例，假设它要删除超过一年的发票。系统可以在每天晚上运行一个自动过程，它扫描数据库，寻找要存档的发票。如果使用 DOM，该过程可以在与发票相关的层次型信息树或关系型信息树中搜索。客户信息、运输信息和行式项目信息等等都将封装在一个文件

中，这个文件用于表示发票。然后，系统可以压缩该文件，并将它保存到磁带或其他存档介质上。如果你对于特定的发票有疑问，可以获取并解压相应的 XML 文件。而后通过读取文件寻找所需的信息，或者利用 DOM 再次将信息加载到数据库中。

2. 客户端的 DOM

在编写本书时，只有 Microsoft Internet Explorer 5.0 客户端内置了 DOM 功能，Netscape 及其他浏览器的开发人员正在为它们的系统增加 DOM Level 1 支持。一旦支持 DOM 的浏览器被广泛使用，Internet 开发人员就能够在客户端利用 DOM 改善信息展示的方式，并减少与服务器的交互。

(1) 灵活的客户端展示

使文档对于不同的客户端可视变得越来越重要。根据客户端的类型以及文档的用途，客户端可能要通过多种方式展示文档。例如，蜂窝电话提供商开始尝试通过电话本身的微型 LCD 屏幕提供有限形式的浏览。就这种目的而言，HTML 并不理想，因为它不包含说明标记中内容含义的信息，它仅仅说明了如何展示这些内容。因此，电话的展示引擎不知道告诉用户蛇的颜色以及它是否有毒是不是非常重要。XML 通过将有关内容的信息作为标记的一部分解决了这个问题。对于设备接收到的文档，定制的浏览器可以使用 DOM 遍历它的节点树，并且有选择地鉴别可以忽略的信息。

(2) 客户端数据输入

随着 DOM 与主要浏览器的集成，可以使用客户端 DOM 对 XML 文档进行操作，为用户提供更高级的交互。系统可以从客户端收集结构化信息，并将它一次性传回服务器，而不必通过跨越几个页面的一系列表单获取这些信息。

5.2.3 DOM 在出版过程中的应用

下面让我们看一下如何在企业环境中利用 DOM 产生和操作 XML 文档。

1. DOM 和数据库

XML 为在不同的数据库之间传递信息提供了理想的机制。从本质上讲，数据库是专有的——每个数据库有不同的元素命名结构，不同的规格化级别，甚至描述枚举信息的方法也不尽相同。利用 DOM 能够简化各种数据库之间信息传递的方式。

通常，在数据库之间传递数据时，必须为每种类型的传输构建定制的翻译程序（参见图 5-5）。

通过将 DOM 作为公共的传输机制，能够大幅度减少需要编写的翻译程序的数量——每个数据库只要从公共的经过协商的 XML 结构导入和导出即可（参见图 5-6）。

DOM 能够用于创建这些转换机制。

2. 使用 DOM 创建复杂的 XML 文档

通过 DOM 操作 XML 文档的优点之一在于 DOM 是随机访问的，即：任何时刻都可以在 XML 树的任何位置创建和附加节点。当你根据层次型或关系型数据库中的信息构建 XML 文档时，这一特征非常 valuable。下面的例子有助于解释这一优点。

假设我们有以下数据库：

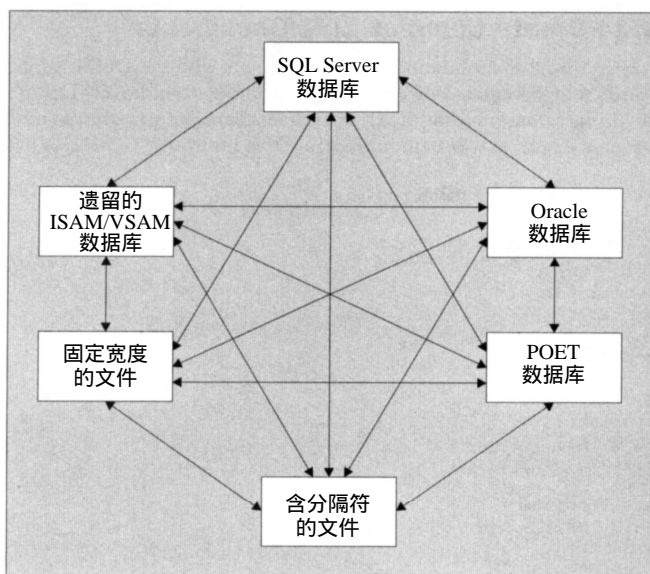


图 5-5

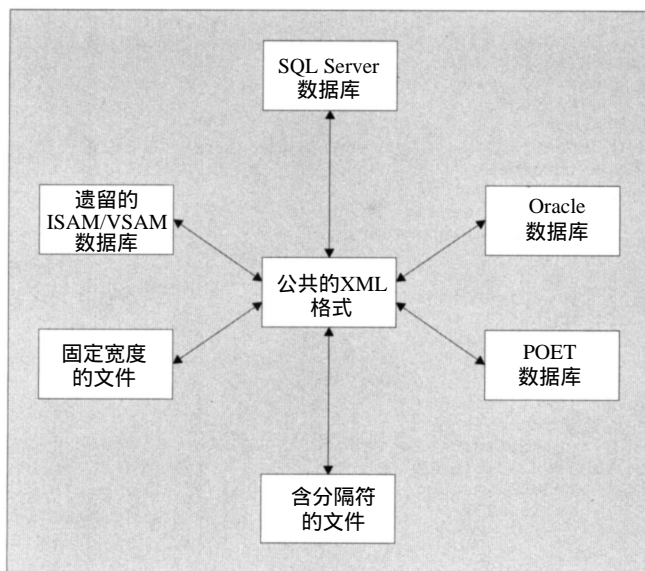


图 5-6

程序清单 5-5

```
CREATE TABLE customer(  
  customerid int,  
  customername varchar(100),  
  city varchar(50),  
  state char(2),  
  zip varchar(10))
```



```
CREATE TABLE invoice(  
    invoiceid int,  
    customerid int,  
    invoicedate datetime)
```

```
CREATE TABLE lineitem(  
    lineitemid int,  
    invoiceid int,  
    product varchar(50),  
    units int)
```

我们希望根据这些表中存储的信息为特定的客户ID创建一个XML文件，它应该具有以下形式：

程序清单 5-6

```
<customer id="customer1"  
    customername="Homer J. Simpson"  
    address="142 Evergreen Terrace"  
    city="Springfield"  
    state="VA"  
    zip="00000">  
    <invoice id="invoice1"  
        invoicedate="11/7/1999">  
        <lineitem id="lineitem1"  
            product="Plutonium"  
            units="17"/>  
        <lineitem id="lineitem2"  
            product="Donuts"  
            units="8726"/>  
        </invoice>  
        <invoice id="invoice2"  
            invoicedate="11/9/1999">  
            <lineitem id="lineitem3"  
                product="Beer"  
                units="37816"/>  
            <lineitem id="lineitem4"  
                product="Peanuts"  
                units="1"/>  
            </invoice>  
        </customer>
```

如果打算手工编写XML文件，我们必须执行以下步骤：

- 从customer表中获取客户信息。
- 将客户信息写入XML文件。
- 从invoice表中获得该客户的所有发票。
- 对于每个发票，执行以下操作：
- 将发票的信息写入XML文件。
- 从lineitem表中获得该发票的所有行式项目。
- 对于每个行式项目，执行以下操作。
- 将行式项目的信息写入XML文件。
- 写入发票对象的结束标记。
- 写入客户对象的结束标记。

然而，如果使用DOM，我们将通过以下方法产生节点树。

- 产生客户根节点。

- 获取客户的所有发票。
- 为每个发票创建一个节点，并将它附加到客户节点。
- 获取客户的所有行式项目。
- 为每个行式项目创建一个节点，并将它附加到适当的发票节点。

这是一个简单的例子，但是它足以说明通过 DOM 创建 XML 文档比将信息写入文本文件更加简单。你不必为了获得所需的信息在各个表之间来回跳跃，每个表中的所有信息可以同时写入文件。随着节点树深度的增加，第一种方法将变得越来越繁琐，而第二种方法具有很好的扩展性。另外，使用 DOM 产生文档能够保证文档是格式正规的。在我们的第一个例子中，假如我们忘记了发票对象的结束标记——就无法解析 XML 文档。

5.3 使用 DOM 和 XML 的应用实例

下面我们将通过几个实例说明如何在实际的应用中使用 DOM。

5.3.1 简单的客户端实例

在本节中，我们将在客户端使用 JScript 和 DOM 对象创建代表图书的 XML 文档。由于 DOM 对象是用于客户端的，因此这个例子必须使用 Internet Explorer 5 运行。它允许用户输入有关图书的信息，书的作者和类别。它使用 DOM 及时产生 XML，并使用 XSL 样式表显示输入的信息。应用程序将提供图 5-7 所示的用户界面。

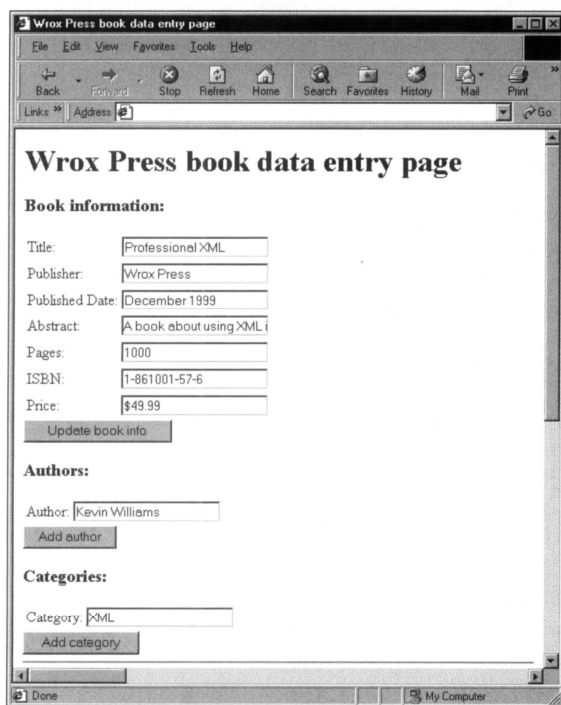


图 5-7

当用户通过按钮添加新项目时，将看到它们以如图 5-8所示的形式显示。

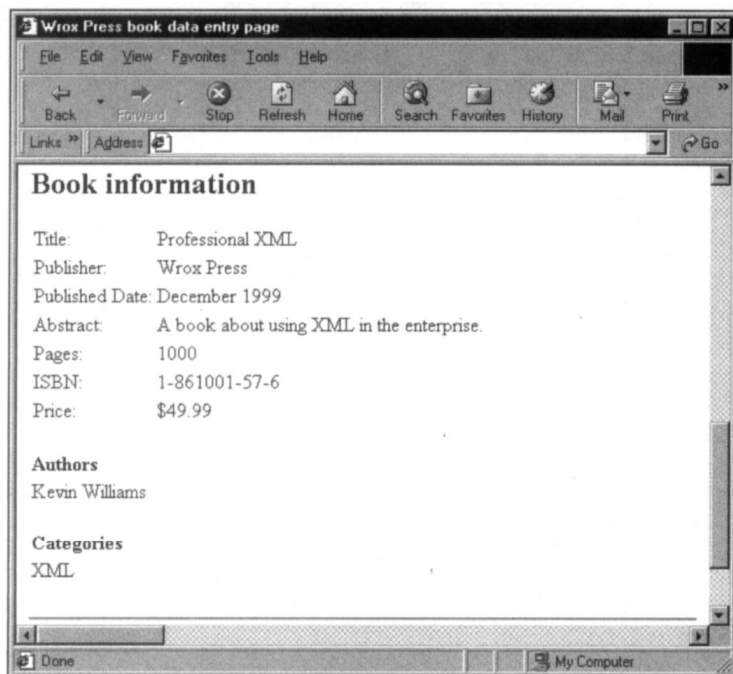


图 5-8

表单下方和页面底部之间将出现图书信息的 HTML 表示和所创建的 XML（如图 5-8 所示）。该演示利用 DOM 通过程序创建 XML 文档。

利用 DOM 允许用户修改页面

我们的 Web 站点 <http://www.wrox.com> 提供了本书的可下载代码，你可以从中获得本节实例程序的文件。

首先，我们来创建提供图书数据输入的 HTML 页面——BookClient.htm。页面中应该包含表单，以便用户输入图书的一般性信息：

程序清单 5-7

```
<HTML>
<HEAD>
  <TITLE>Wrox Press book data entry page</TITLE>
</HEAD>
<BODY onload="initializeBook()">
  <H1>Wrox Press book data entry page</H1>
  <H3>Book information:</H3>
  <TABLE>
    <TR>
      <TD>Title:</TD><TD><INPUT id=txtTitle></TD>
    </TR>
    <TR>
      <TD>Publisher:</TD><TD><INPUT id=txtPublisher></TD>
```

```

</TR>
<TR>
  <TD>Published Date:</TD><TD><INPUT id=txtPubDate></TD>
</TR>
<TR>
  <TD>Abstract:</TD><TD><INPUT id=txtAbstract></TD>
</TR>
<TR>
  <TD>Pages:</TD><TD><INPUT id=txtPages></TD>
</TR>
<TR>
  <TD>ISBN:</TD><TD><INPUT id=txtISBN></TD>
</TR>
<TR>
  <TD>Price:</TD><TD><INPUT id=txtPrice></TD>
</TR>
</TABLE>
<INPUT id=btnUpdate type=button value="Update book info"
  onclick="updateBookInfo()">

<H3>Authors:</H3>
<TABLE>
  <TR>
    <TD>Author:</TD><TD><INPUT id=txtAuthor></TD>
  </TR>
</TABLE>
<INPUT id=btnAddAuthor type=button value="Add author"
  onclick="addAuthor()">
<H3>Categories:</H3>
<TABLE>
  <TR>
    <TD>Category:</TD><TD><INPUT id=txtCategory></TD>
  </TR>
</TABLE>
<INPUT id=btnAddCategory type=button value="Add category"
  onclick="addCategory()">

```

在上面的代码中，我们调用了 4 个 JScript 函数：initializeBook()，updateBookInfo()，addAuthor() 和 addCategory()。这些函数利用 DOM 初始化和修改文档信息。表单由三组控件构成。第一组是图书常规信息的输入框和更新按钮（由于在我们的图书 XML 中图书常规信息只能出现一次）。第二组是用于添加图书作者的表单（因为作者可以出现多次），最后一组是用于添加图书类别的表单。

下面我们将创建 XML 数据岛，它将包含已经完成的图书：

```

<XML id=docBook>
  <Book>
  </Book>
</XML>

```

在本例中，使用数据岛作为通知 Internet Explorer 5 的信号，说明要在代码中操作名为 docBook 的 XML 文档。注意，我们已经指定了 XML 文档的根对象，即：名为 Book 的空元素。当然，也可以选择将数据岛发送至客户端之前不向其中写入任何内容，或者仅仅添加一些初始信息（如：图书的 ID）。

代码的下一部分包含用于操作图书的脚本：

```
<SCRIPT>
```

```
var docBook;
```

我们将 `docBook` 指定为全局变量，这样访问 XML 文档时就不必使用全名 `document.all("docBook").XMLDocument`。这个语法是 IE5 中新增的，它允许客户端 JavaScript 使用 XML DOM 操作——XMLDOMDocument 对象的所有功能。我们将在 `initializeBook()` 函数中初始化 `docBook` 变量的值，你应该记得这个函数是在 `<BODY>` 标记的 `onload` 事件中调用的：

```
function initializeBook()
{
    docBook = document.all("docBook").XMLDocument;
    docBook.async = false;
    renderElements();
}
```

在以上代码中，我们首先初始化 `docBook` 变量，使之指向 `docBook` 文档。然后，将文档的 `async` 属性设置为 `false`。它使得对文档的所有操作都必须以同步方式执行——仅当操作完成后才返回函数继续执行。它能够避免在文档更新过程中访问文档（另外，你可以编写一个事件处理器等待 `ondataavailable` 事件的触发——但是如果不需要执行特别的事件处理操作，最好让代码处于等待状态）。最后，我们调用显示 DOM 内容的函数——以普通格式显示以及使用 XSL 样式表显示。

我们还需要一个辅助函数。这个辅助函数根据元素名称创建或替换指定父元素的相应子元素。这样，当我们的用户决定将书名由“XML for Professionals”改为“Professional XML”时，就不会导致 `<Book>` 元素中突然出现了两个 `<Title>` 子元素。

该函数从 DOM 中引入了一些新的属性和方法。在继续讨论之前，先看看这些新属性和新方法。

`createElement()` 方法作用于文档，它用于实例化 `Element` 对象。它唯一的参数是要创建的新元素的名称。方法将返回所创建的 `Element` 对象。通过这种方式创建的 `Element` 对象是孤立的；即：它不与任何特定的父元素相关联。需要通过 `appendChild()` 等方法将该元素链接到 XML 节点树的适当位置。

`createText()` 方法作用于文档，它用于创建 `Text` 对象。它唯一的参数是构成 `Text` 对象内容的文本，方法返回所创建的 `Text` 对象。与 `createElement()` 方法类似，`createText()` 创建的节点也是孤立的，它需要依靠特定的方法添加到节点树中。`Text` 对象代表文档中无格式的文本；通常，它们作为 `Element` 对象的子对象。例如，在下面这个简短的代码片段中：

```
<Book>
  <Title>Professional XML</Title>
</Book>
```

代码中的 `<Book>` 有子元素 `<Title>`；元素 `<Title>` 有子对象 `<Text>`，它包含字符串“Professional XML”。

`appendChild()` 方法作用于任何节点，它用于在两个节点之间建立父子关系。被添加的元素作为方法的参数，方法将返回被添加的节点。

getElementsByTagName()方法可以作用于任何节点，它用于在本节点的子元素中定位标记与参数字符串匹配的元素。它返回 NodeList 对象，其中包含与查询条件匹配的节点的无序集合。

replaceChild()方法可以作用于任何节点，它将一个节点中的特定子节点替换为另一个子节点。新旧节点都将作为方法的参数，方法将返回旧的（被删除的）节点。

程序清单 5-8

```
function createOrReplaceElement(sElementName, sElementValue, elementParent)
{
    var elementItem;
    var textValue;
    var nodelistOldItem;

    elementItem = docBook.createElement(sElementName);
    textValue = docBook.createTextNode(sElementValue);
    elementItem.appendChild(textValue);

    nodelistOldItem = elementParent.getElementsByTagName(sElementName);
    if (nodelistOldItem.length > 0)
    {
        elementParent.replaceChild(elementItem, nodelistOldItem.item(0));
    }
    else
    {
        elementParent.appendChild(elementItem);
    }
}
```

在以上代码中，我们创建新的元素节点和文本节点，它们代表要添加的元素。然后，将文本节点附加为元素节点的子节点。最后，检查父节点中是否存在与新创建的元素节点名称相同的子节点。如果存在，将用新的子节点替换旧的子节点；否则，将新的子节点添加至父节点中。

下面，我们将实现用于更新文档中图书常规信息的函数：

程序清单 5-9

```
function updateBookInfo()
{
    createOrReplaceElement("Title",
                           txtTitle.value,
                           docBook.documentElement);
    createOrReplaceElement("Publisher",
                           txtPublisher.value,
                           docBook.documentElement);
    createOrReplaceElement("PubDate",
                           txtPubDate.value,
                           docBook.documentElement);
    createOrReplaceElement("Abstract",
                           txtAbstract.value,
                           docBook.documentElement);
    createOrReplaceElement("Pages",
                           txtPages.value,
                           docBook.documentElement);
    createOrReplaceElement("ISBN",
                           txtISBN.value,
                           docBook.documentElement);
    createOrReplaceElement("Price",
```

```
        txtPrice.value,  
        docBook.documentElement);  
  
    renderElements();  
}
```

以上代码获取表单输入控件中指定的值，并利用这些值创建新元素，然后将新元素添加到文档中，或者替换原来的子元素。

下面的函数用于添加图书的作者：

程序清单 5-10

```
function addAuthor()  
{  
    var elementAuthor;  
    var textAuthor;  
    var nodelistAuthors;  
    var elementAuthors;  
  
    elementAuthor = docBook.createElement("Author");  
    textAuthor = docBook.createTextNode(txtAuthor.value);  
    elementAuthor.appendChild(textAuthor);  
    nodelistAuthors = docBook.getElementsByTagName("Authors");  
    if (nodelistAuthors.length == 0)  
    {  
        elementAuthors = docBook.createElement("Authors");  
        docBook.documentElement.appendChild(elementAuthors);  
    }  
    else  
    {  
        elementAuthors = nodelistAuthors.item(0);  
    }  
  
    elementAuthors.appendChild(elementAuthor);  
  
    renderElements();  
}
```

该函数创建元素和文本对象，并在它们之间建立父子关系。然后，它检查文档中是否存在容器元素<Authors>。如果不存在，函数创建该容器元素。最后，它将在函数开始处创建的元素附加到新的<Authors>对象或已有的<Authors>对象中。而后，函数再次刷新文档，更新样式表形式的图书信息以及纯XML部分。

下面这个函数用于添加图书的类别：

程序清单 5-11

```
function addCategory()  
{  
    var elementCategory;  
    var textCategory;  
    var nodelistRecSubjCategories;  
    var elementRecSubjCategories;  
  
    elementCategory = docBook.createElement("Category");  
    textCategory = docBook.createTextNode(txtCategory.value);  
    elementCategory.appendChild(textCategory);  
    nodelistRecSubjCategories =  
        docBook.getElementsByTagName("RecSubjCategories");  
    if (nodelistRecSubjCategories.length == 0)
```

```

{
    elementRecSubjCategories = docBook.createElement("RecSubjCategories");
    docBook.documentElement.appendChild(elementRecSubjCategories);
}
else
{
    elementRecSubjCategories = nodelistRecSubjCategories.item(0);
}

elementRecSubjCategories.appendChild(elementCategory);

renderElements();
}

```

此时，XML文档仅仅以XML DOM节点树的形式存在于内存中。为了让用户看到信息的变化，我们需要展示文档。下面这个函数正是完成这项功能，它使用样式表和普通格式展示文档的内容：

程序清单 5-12

```

function renderElements()
{
    document.all("divRawXML").innerText = docBook.xml;
    bookInfo.innerHTML = docBook.transformNode(bookXSL.documentElement);
    authorTable.innerHTML = docBook.transformNode(authorXSL.documentElement);
    categoryTable.innerHTML =
        docBook.transformNode(categoryXSL.documentElement);
}

</SCRIPT>

```

使用文档的xml属性，能够得到整个文档的纯文本XML输出。这是大多数人对于XML“文件”的认识。需要注意的是，xml属性是Microsoft对DOM的扩展—W3C计划在DOM Level 3中包含输入和输出功能。当讨论DOM在服务器端的应用时，我们将看到如何使用纯粹的DOM Level 1功能产生XML文件。

我们将divRawXML元素的文本（位于文件底部）设置为普通的文本输出。然后，我们使用样式表将XML文档转化为可读性更强的输出。第9章将详细介绍XSL。

最后，我们列出了三个内嵌的样式表，以及HTML页面的其余部分，其中包含renderElement()函数涉及的分页元素：

程序清单 5-13

```

<XML id=bookXSL>
  <DIV xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <xsl:choose>
      <xsl:when test="/Book/Title[. $ne$ '']">
        <TABLE BORDER="0" CELLPADDING="1">
          <TR>
            <TD>Title:</TD>
            <TD><xsl:value-of select="/Book/Title"/></TD>
          </TR>
          <TR>

```

```

        <TD>Publisher:</TD>
        <TD><xsl:value-of select="/Book/Publisher"/></TD>
    </TR>
    <TR>
        <TD>Published Date:</TD>
        <TD><xsl:value-of select="/Book/PubDate"/></TD>
    </TR>
    <TR>
        <TD>Abstract:</TD>
        <TD><xsl:value-of select="/Book/Abstract"/></TD>
    </TR>
    <TR>
        <TD>Pages:</TD>
        <TD><xsl:value-of select="/Book/Pages"/></TD>
    </TR>
    <TR>
        <TD>ISBN:</TD>
        <TD><xsl:value-of select="/Book/ISBN"/></TD>
    </TR>
    <TR>
        <TD>Price:</TD>
        <TD><xsl:value-of select="/Book/Price"/></TD>
    </TR>
</TABLE>
</xsl:when>
<xsl:otherwise>
    Book information not yet specified.
</xsl:otherwise>
</xsl:choose>
</DIV>
</XML>

<XML id=authorXSL>
    <DIV xmlns:xsl="http://www.w3.org/TR/WD-xsl">
        <TABLE BORDER="0" CELSPACING="1">
            <TR>
                <TD><STRONG>Authors</STRONG></TD>
            </TR>
            <xsl:for-each select="/Book/Authors/Author">
                <TR>
                    <TD><xsl:value-of select="text()"/></TD>
                </TR>
            </xsl:for-each>
        </TABLE>
    </DIV>
</XML>

<XML id=categoryXSL>
    <DIV xmlns:xsl="http://www.w3.org/TR/WD-xsl">
        <TABLE BORDER="0" CELSPACING="1">
            <TR>
                <TD><STRONG>Categories</STRONG></TD>
            </TR>
            <xsl:for-each select="/Book/RecSubjCategories/Category">
                <TR>
                    <TD><xsl:value-of select="text()"/></TD>
                </TR>
            </xsl:for-each>
        </TABLE>
    </DIV>
</XML>

```

```
<HR>
<H2>Book information</H2>
<P><DIV id=bookInfo></DIV></P>
<P><DIV id=authorTable></DIV></P>
<P><DIV id=categoryTable></DIV></P>
<HR>
The text expression of the current contents of the DOM tree is:
<PRE><DIV id=divRawXML></DIV></PRE>
</BODY>
</HTML>
```

通过这种方式在客户端创建的 XML 文档能够传回服务器端进行操作，虽然这超出了本例的讨论范围。最简单的方法是设置输入元素，它通常是隐含的，它的值是文档的 `xml` 属性的值；当表单被提交时，这些信息将成为提交元素的一部分。如果客户端的 XML 文档过大，不适于作为表单的一部分提交，可以考虑其他方式，例如：FTP，使用 Posting Acceptor 的 HTTP，其他 HTTP 文件提交程序，`xmlhttp` 事务（Microsoft 特有的结构），或者 Microsoft 的 SOAP 协议（第 1 章讨论了 SOAP 协议）等。

5.3.2 更复杂的编程实例

下面让我们看看如何将 DOM 应用于服务器端。我们将构建一个 ASP 页面，它通过 Posting Acceptor 接收客户端提交的固定宽度的文本文件，然后解析文件并使用 DOM 建立数据岛，最后将结果发送回客户端。

1. 使用 DOM 根据文本创建 XML 文档

在这个例子中，用户不必通过 HTML 页面输入图书的数据，我们已经有代表图书的固定宽度的文本文件。假设它已经通过某种机制（FTP、Posting Acceptor 或其他任何方法）上载到服务器的 `/uploads` 目录。ASP 页面将解析上载的文件，在 HTML 页面中产生数据岛，并将页面发送回客户端供用户查看。

你可以通过 Wrox 的 Web 站点 <http://www.wrox.com> 获得本节实例的代码文件。

我们将提交的图书文件如下：

程序清单 5-14

Professional XML	Wrox Press	December 1999
800	1861001576	49.99
This book is a definitive, practical guide to XML.		
AMichael Kay		
ASSteven Livingston		
ABrian Loesgen		
ADidier Martin		
ASStephen Mohr		
ANikola Ozu		
AMark Seabourne		
APeter Stark		
AKevin Williams		
CXML		
CInternet Development		
CClient-server Development		

这是一个固定宽度的文件，许多遗留的系统仍然使用这类文件导出数据。文件的第一行包含书名、出版商和出版日期；第二行包含书的价格和 ISBN 号；第三行是书的摘要。文件的其余部分列出了书的作者（以字母 A 开头的行）和类别（以字母 C 开头的行）。在实际应用中，产生该文件的系统还应该提供用于描述文件格式和内容的文档。

我们将使用下面的表单 BookForm.htm 请求解析该文件：

程序清单 5-15

```
<HTML>
  <BODY>
    <FORM action="/DisplayBook.asp"
      method="post">
      <INPUT type="submit" value="Parse the uploaded file">
    </FORM>
  </BODY>
</HTML>
```

DisplayBook.asp 文件如下：

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
  <TITLE>Thank you for submitting your book information!</TITLE>
```

为了能够在客户端看到利用样式表显示的结果，我们将使用前一节介绍的客户端实例中的 renderElement() JScript 函数；然而，在本例中最重要的是如何解析文件。一旦解析了文件，就很容易将它存储在关系数据库中，或者以其他形式保存。

程序清单 5-16

```
<SCRIPT>
function renderElements()
{
  bookInfo.innerHTML =
    docBook.transformNode(bookXSL.documentElement);
  authorTable.innerHTML =
    docBook.transformNode(authorXSL.documentElement);
  categoryTable.innerHTML =
    docBook.transformNode(categoryXSL.documentElement);
}
</SCRIPT>
```

下面，我们将实现几个辅助性子例程。第一个是 AddElementToParent()，它将指定的元素和值添加到指定的 DOM 中特定的父元素中：

程序清单 5-17

```
<%
Sub AddElementToParent (domBook, elemParent, sChild, sValue)
  Dim elemSubelement
  Dim textSubelement

  Set elemSubelement = domBook.CreateElement(sChild)
  Set textSubelement = domBook.CreateTextNode(sValue)
  elemSubelement.appendChild(textSubelement)
```

```

        elemParent.appendChild(elemSubelement)

        Set elemSubelement = Nothing
        Set textSubelement = Nothing
    End Sub

```

第二个辅助子例程用于从 DOM 产生 XML 流。需要说明的是，DOM 的某些实现，例如：Microsoft 提供的实现，包含了用于产生这个文本的属性，然而由于这并不是 Level 1 要求的功能，因此你使用的 DOM 实现可能不提供该属性，为此，必须使用如下子例程。另外，大多数节点类型，例如：属性、注释等，是我们的例子无法处理的。我们将在后面重新分析这段代码，说明如何导出其他节点类型。但是现在，这个子例程只能处理 element 和 text 节点类型：

程序清单 5-18

```

Sub WriteNodeXML (nodeTarget)
    Dim i

    If nodeTarget.NodeType = 1 Then
        ' Element
        Response.Write "<" & nodeTarget.tagName & ">"
        For i = 0 to nodeTarget.childNodes.Length - 1
            WriteNodeXML nodeTarget.childNodes.item(i)
        Next
        Response.Write "</" & nodeTarget.tagName & ">"
    ElseIf nodeTarget.NodeType = 3 Then
        ' Text Node
        Response.Write nodeTarget.data
    End If

End Sub
%>

```

程序中的递归用于产生文件中的嵌套标记——当使用递归时，你应该特别注意堆栈问题。然而，对于大多数 XML 文档，这种技术是恰当的。

实际上，真正的文件解析操作是在将包含 XML 的数据岛中执行的。一旦解析完毕，就可以利用 Response.Write 生成 XML（在 WriteNodeXML() 过程中），它将创建 HTML 中的数据岛，这个 HTML 文件将返回给客户端：

程序清单 5-19

```

<XML id=docBook>
<%
    Dim fileInvoice
    Dim tsInvoice
    Dim domInvoice
    Dim elemInvoice
    Dim elemLineItem

    Dim sFilename
    Dim sPath
    Dim sLine
    Dim sWork

    Const ForReading = 1

```

在本例中，我们将文件名硬编码——但是在实际环境中，文件应该来自 Posting Acceptor 或其他源：

```
sFilename = "e:\web\book.txt"
```

然后，我们创建 Microsoft XML DOM 对象的实例，并将它设置为同步操作：

```
' create the instance of the DOM and the root Book element
Set domBook = CreateObject("Microsoft.XMLDOM")
domBook.async = false
```

下面，我们创建 Book 元素，并将它添加到新文档中：

```
Set elemBook = domBook.CreateElement("Book")
domBook.appendChild elemBook
```

现在，我们将打开文件，并开始解析：

```
' open the file
Set fileBook = CreateObject("Scripting.FileSystemObject")
Set tsBook = fileBook.OpenTextFile(sFilename, ForReading)
```

对于与图书相关的每条常规信息，我们使用 AddElementToParent() 子例程将适当的子元素添加到 Book 元素中。该子例程将根据标记名称和数据创建元素节点和文本节点，并将它们链接到一起，添加到指定的父节点中：

程序清单 5-20

```
' process the title and publisher line
sLine = tsBook.ReadLine

sWork = Trim(Mid(sLine, 1, 30)) ' Title
AddElementToParent domBook, elemBook, "Title", sWork

sWork = Trim(Mid(sLine, 31, 20)) ' Publisher
AddElementToParent domBook, elemBook, "Publisher", sWork

sWork = Trim(Mid(sLine, 51, 20)) ' PubDate
AddElementToParent domBook, elemBook, "PubDate", sWork

' process the number of pages, ISBN, and price line
sLine = tsBook.ReadLine

sWork = Trim(Mid(sLine, 1, 10)) ' Number of pages
AddElementToParent domBook, elemBook, "Pages", sWork

sWork = Trim(Mid(sLine, 11, 13)) ' ISBN
AddElementToParent domBook, elemBook, "ISBN", sWork

sWork = Trim(Mid(sLine, 24, 10)) ' Price
AddElementToParent domBook, elemBook, "Price", sWork

' process the abstract line
sLine = tsBook.ReadLine

AddElementToParent domBook, elemBook, "Abstract", sLine
```

现在，文件中未处理的行分别对应于 Author元素和Category元素。文件的第一列指示了哪些代表作者，哪些代表类别——A表示作者，C表示类别。首先，我们需要产生 RecSubjCategories和Authors容器元素，并将它们添加为 Book元素的子节点：

程序清单 5-21

```
Set elemRecSubjCategories = domBook.CreateElement("RecSubjCategories")
Set elemAuthors = domBook.CreateElement("Authors")

elemBook.appendChild(elemRecSubjCategories)
elemBook.appendChild(elemAuthors)
```

下面，我们将读取文件的剩余行。对于每一行，先确定它是类别还是作者，然后将它添加为适当容器元素的子节点：

程序清单 5-22

```
While Not tsBook.AtEndOfStream
    sLine = tsBook.ReadLine
    If Left(sLine, 1) = "A" Then
        AddElementToParent domBook, elemAuthors, "Author", Mid(sLine, 2)
    Else
        AddElementToParent domBook, elemRecSubjCategories, "Category", _
            Mid(sLine, 2)
    End If
Wend
tsBook.Close
```

下面，我们将向应答串中写入纯 XML。由于我们的代码是在数据岛中执行的，因此写入的数据将嵌在数据岛中：

程序清单 5-23

```
WriteNodeXML elemBook

' and clear our objects
Set fileBook = Nothing
Set tsBook = Nothing
Set domBook = Nothing
Set elemBook = Nothing
Set elemAuthor = Nothing
Set elemRecSubjCategories = Nothing
Set elemAuthors = Nothing

%>
</XML>
```

下面的三个样式表和 HTML代码片段选自第一个例子。唯一的变化是 BODY元素现在包含 onload事件，它将使用三个样式表展示 XML数据岛：

程序清单 5-24

```
<XML id=bookXSL>
  <DIV xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

```

<xsl:choose>
  <xsl:when test="/Book/Title[. $ne$ '']">
    <TABLE BORDER="0" CELLPADDING="1">
      <TR>
        <TD>Title:</TD>
        <TD><xsl:value-of select="/Book/Title"/></TD>
      </TR>
      <TR>
        <TD>Publisher:</TD>
        <TD><xsl:value-of select="/Book/Publisher"/></TD>
      </TR>
      <TR>
        <TD>Published Date:</TD>
        <TD><xsl:value-of select="/Book/PubDate"/></TD>
      </TR>
      <TR>
        <TD>Abstract:</TD>
        <TD><xsl:value-of select="/Book/Abstract"/></TD>
      </TR>
      <TR>
        <TD>Pages:</TD>
        <TD><xsl:value-of select="/Book/Pages"/></TD>
      </TR>
      <TR>
        <TD>ISBN:</TD>
        <TD><xsl:value-of select="/Book/ISBN"/></TD>
      </TR>
      <TR>
        <TD>Price:</TD>
        <TD><xsl:value-of select="/Book/Price"/></TD>
      </TR>
    </TABLE>
  </xsl:when>
  <xsl:otherwise>
    Book information not yet specified.
  </xsl:otherwise>
</xsl:choose>
</DIV>
</XML>

<XML id=authorXSL>
  <DIV xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <TABLE BORDER="0" CELLSPACING="1">
      <TR>
        <TD><STRONG>Authors</STRONG></TD>
      </TR>
      <xsl:for-each select="/Book/Authors/Author">
        <TR>
          <TD><xsl:value-of select="text()"/></TD>
        </TR>
      </xsl:for-each>
    </TABLE>
  </DIV>
</XML>

<XML id=categoryXSL>
  <DIV xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <TABLE BORDER="0" CELLSPACING="1">

```



```
<TR>
  <TD><STRONG>Categories</STRONG></TD>
</TR>
<xsl:for-each select="/Book/RecSubjCategories/Category">
  <TR>
    <TD><xsl:value-of select="text()" /></TD>
  </TR>
</xsl:for-each>
</TABLE>
</DIV>
</XML>

</HEAD>
<BODY onload="renderElements()">
  <H2>Book information</H2>
  <P><DIV id=bookInfo></DIV></P>
  <P><DIV id=authorTable></DIV></P>
  <P><DIV id=categoryTable></DIV></P>

</BODY>
</HTML>
```

DisplayBook.asp文件将产生如图5-9所示的输出。

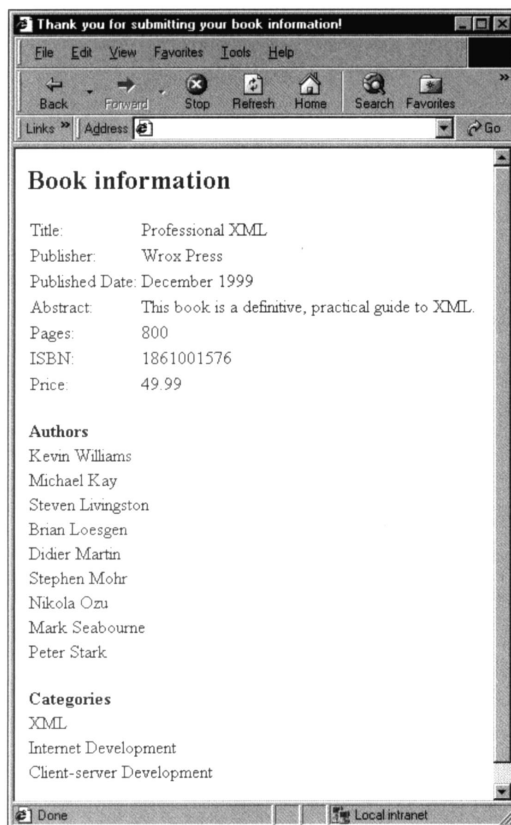


图 5-9

这个简单的例子说明了如何在服务器端利用 DOM 处理信息。例如，在线发票系统可以通过 FTP 将发票文件放置到服务器上的某个目录；每天早晨可以运行预定的程序解析目录中的所有文件，将它们转化为 XML，并存档以备后用。或者，某个需要动态查询发票信息的程序可以从数据库中读取适当的信息，使用 DOM 将它转化为 XML，并将生成的 XML 返回给客户端进行处理。

2. 使用 DOM 修改信息

你可能已经注意到在上面的例子中，ISBN 的格式非常不好。通常，ISBN 是以特定位置的连字符分隔的，例如：

1-861001-57-6

然而，我们在文件中得到的是不含任何格式的纯数字：

1861001576

我们能够通过在本脚本中增加代码解决这个问题。下面将在前面的例子中增加用于设定 ISBN 格式的代码。

当文件关闭，已经构建 XML DOM 树之后，添加以下代码：

程序清单 5-25

```
Dim nodelistISBN
Dim i

Set nodelistISBN = elemBook.getElementsByTagName("ISBN")
For i = 0 to nodelistISBN.length - 1
    sWork = nodelistISBN.item(i).childNodes(0).data
    sWork = Left(sWork, 1) & "-" & Mid(sWork, 2, 6) & "-" & _
        Mid(sWork, 8, 2) & "-" & Mid(sWork, 10, 1)
    nodelistISBN.item(i).childNodes(0).data = sWork
Next

Set nodelistISBN = Nothing
```

以上代码在 Book 元素的子孙中搜索名为 ISBN 的元素。现在，我们知道每个 ISBN 元素有一个子节点——包含格式不恰当的字符串的文本节点。我们读取该元素的值，在适当位置插入连字符，用正确的格式更新元素值。需要注意的是，我们遍历了 NodeList 中的所有元素，因此如果存在多个 ISBN 元素，以上代码块不会遗漏任何元素——这使得它非常适于修改大型 XML 文件中深层嵌套的元素。

现在，DisplayBook.asp 文件的输出将如图 5-10 所示。

3. 使用 DOM 删除元素

DOM 也可以用于从节点树中删除元素。比如，Kevin Williams 不再是书的作者——但是所有文件都仍然认为他是书的作者。我们可以使用 DOM 删除名为 Kevin Williams 的作者。

同样，我们是以前面的例子为基础的。

在我们添加的 ISBN 代码之后，继续插入以下代码块：

程序清单 5-26

```
Dim nodelistAuthors
```

```
Set nodelistAuthors = elemBook.getElementsByTagName("Author")
For i = 0 to nodelistAuthors.length - 1
    If nodelistAuthors.item(i).childNodes(0).data = "Kevin Williams" Then
        ' we'll delete this node
        nodelistAuthors.item(i).parentNode.removeChild( _
                                                    nodelistAuthors.item(i))
    End If
Next

Set nodelistAuthors = Nothing
```

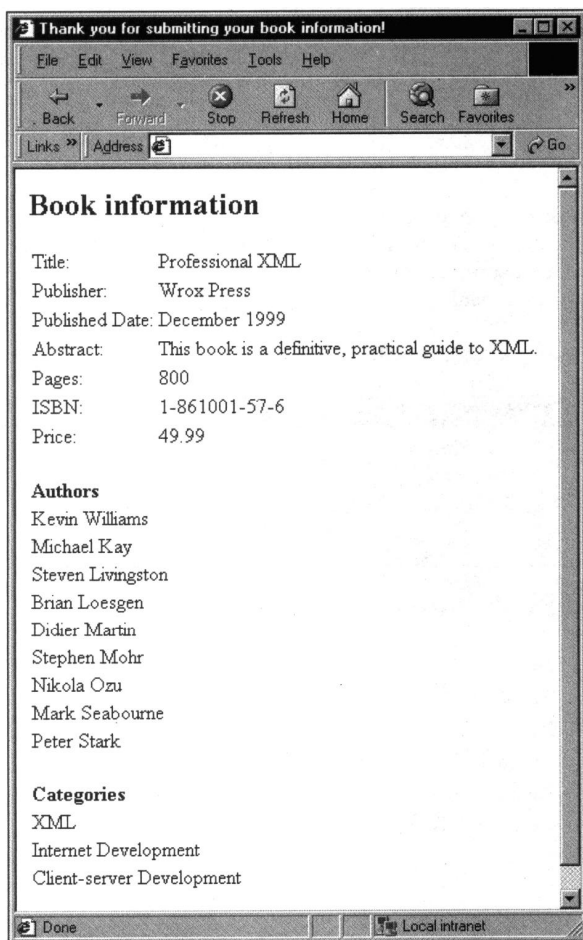


图 5-10

以上代码在 Book 元素的子孙节点中扫描所有 Author 元素；并从父节点中删除含文本 Kevin Williams 的子节点。值得注意的是，我们不需要更多的清除工作；甚至连节点本身仍然是有效的，但是它不与其他任何节点相关联，因此它不会出现在生成的 XML 中。

现在，DisplayBook.asp 文件的输出将如图 5-11 所示。

4. 再谈 XML 流的产生

让我们回忆一下用于生成节点树的 XML 的递归子例程 WriteNodeXML()：

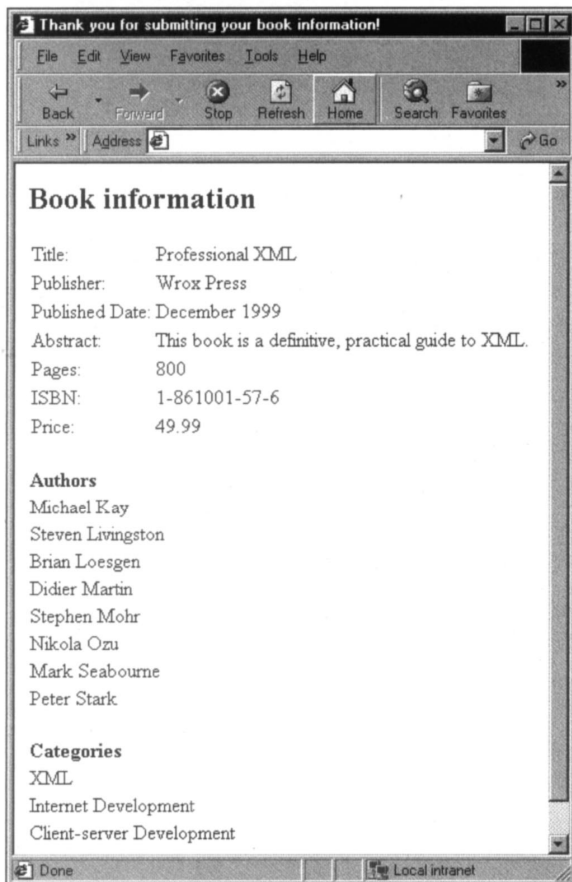


图 5-11

程序清单 5-27

```
Sub WriteNodeXML (nodeTarget)
    Dim i

    If nodeTarget.NodeType = 1 Then
        ' Element
        Response.Write "<" & nodeTarget.tagName & ">"
        For i = 0 to nodeTarget.childNodes.Length - 1
            WriteNodeXML nodeTarget.childNodes.item(i)
        Next
        Response.Write "</" & nodeTarget.tagName & ">"
    ElseIf nodeTarget.NodeType = 3 Then
        ' Text Node
        Response.Write nodeTarget.data
    End If

End Sub
```

虽然这段代码完全能够应付我们的例子，但是它几乎不能处理其他可能出现的 DOM变换形式。尽管介绍能够正确表达各种节点类型的函数超出了本书的讨论范围，但是我们不妨看看如何表达属性（这也是一个相当重要的节点类型）。

由于属性是特殊的节点，它并不是与之关联的元素节点的子节点；而是位于元素的 `attributes` 属性中。该属性返回的 `NamedNodeMap` 能够表达特定元素的所有属性信息。我们将在原来的 `WriteNodeXML()` 函数中增加处理属性的代码：

程序清单 5-28

```
Select Case nodeTarget.NodeType
Case 1
    ' Element node
    Response.Write "<" & nodeTarget.tagName
    For i = 0 to nodeTarget.attributes.Length - 1
        Response.Write " " & nodeTarget.attributes.item(i).Name & "="
        Response.Write chr(34) & _
            nodeTarget.attributes.item(i).nodeValue & chr(34)
    Next
    Response.Write ">"
    For i = 0 to nodeTarget.childNodes.Length - 1
        WriteNodeXML nodeTarget.childNodes.item(i)
    Next
    Response.Write "</" & nodeTarget.tagName & ">"
Case 3
    ' Text node
    Response.Write nodeTarget.data
Case Else
End Select
```

现在，属性的名称-值对将嵌入元素的起始标记中。为了将 DOM的内容完全表示为文件，我们还需要处理注释、处理指令和 CDATA 部分等。

毫无疑问，DOM 为访问 XML 文档的内容提供了简单灵活的方式。DOM 可以用在客户端或服务端，它能够对 XML 结构进行操作、添加或删除。

5.4 DOM和XML的未来

XML 仍然处在早期开发阶段。如果你在 HTML 1.0 规范发布时曾经做过 HTML 开发，你就知道迄今为止这种语言已经发生了巨大的变化——XML 也可能出现同样的情况。在本节中，我们将展望 DOM 和 XML 的发展趋势，以及它们将对我们产生的影响。

5.4.1 W3C的工作

W3C 正在定义 XML 和 DOM。在编写本书时，DOM Level 1 规范处于建议状态，DOM Level 2 规范处于候选建议状态。另外，W3C 已经开始讨论 DOM Level 3 规范。在 Level 3 中，W3C 准备将用于加载文档和将 XML 结构保存为文件的机制标准化（许多特定的 DOM 实现已经非正式地提供了这些功能），另外它还将解决文档有效性验证的问题。该版本还将标准化文档的查看和格式化机制。除了 Level 3，W3C 计划增加用户交互机制，例如：提示和查询语言。

5.4.2 应用

致力于提供能够访问和修改 XML 文件的工具的第三方开发商不计其数。这些开发商都希望能够充分利用 DOM 的功能。在许多实例中，这些工具构成了 DOM 的外包装，它使得开发人员能够在更高层访问信息（而不是手工遍历节点树）。其中一个应用是 XPath——这是一种查询语言，它用于控制节点树的遍历和数据的获取。DOM 在今后一两年内可能有所变化，因而这些用于访问 XML 文档的工具也将随之改变。

5.4.3 数据库、模式和 DOM

数据库和 XML 文档之间的界限越来越窄。目前，已经能够通过查询语言访问 XML 文档，通过 XML 模式控制文档内容的类型，随着这些技术的发展，数据库和 DOM 之间的信息传递出现差错的可能性越来越小。例如，Microsoft 和 Oracle 已经在他们的数据库服务器应用程序中内置了 XML 支持。

5.5 小结

我们已经看到 DOM 为遍历构成 XML 文档的节点树以及获取其中存储的信息提供了自然的面向对象的机制。特别是：

- DOM 为处理 XML 文档提供了可编程的方法。
- DOM 允许我们在客户端和服务端修改 XML 数据结构。
- DOM 为数据库之间的信息传递提供了理想的机制。
- DOM 在不同的平台上可以有不同的实现方式。
- DOM 是内存密集型操作，它不适于处理大型 XML 文件。

简而言之，读取和操作 XML 文档时，使用 DOM 将保证各种平台之间获得最大程度的互操作性。然而，使用 DOM 并不一定是最佳策略，特别是对于非常大的文件。为了避免将整个文档加载到内存中而造成的开销，可以使用 SAX 等事件驱动的解析器处理大型 XML 文件，我们将在下一章介绍 SAX。