

第11章 使用XML数据

读者可能会觉得在一本 ASP 书中讲述本章内容不太合适，事实上扩展标记语言 (eXtensible Markup Language, XML) 正在越来越深入我们的生活，这是一件好事。XML 可以跨越所有边界，要获得真正独立的、跨平台的数据传送格式，XML 可能是唯一的选择。

这种说法可能会令人感到沮丧，但在计算机业内 XML 几乎得到了所有大型 (或小型) 开发厂商的认可。标准能达到如此的统一，过去也只有在 TCP/IP 协议被采纳为网络协议的标准时出现过这种现象。XML 是一种国际标准，受一个工业标准团体的控制，得到全世界的广泛支持，并成为为数不多的只有一个标准的技术之一。

读者可能会觉得很奇怪，当今世界各种标准的变化就像季节更替一样频繁，即使是单个标准也被多个公司分成数段以追求竞争优势。然而，XML 却是个例外，因为它确实获得了许多公司的广泛支持。令人惊奇的是所有的人似乎都在努力实现并遵循这种唯一的标准。如果回想一下过去给工业界造成那么多麻烦的有关标准问题的争论，XML 就更令人惊奇了。

既然 XML 是一个被普遍实现的标准，因此，在 ASP 中使用它也很简单。如果使用 ASP 创建一个 Web 站点，那么很有可能会使用某种形式的数据库来存储数据。而 XML 是另一种存储数据的格式，其应用也越来越广泛，因此必须掌握它。当然 XML 的功能不仅于此。

本章将讨论以下内容：

- XML 数据的含义。
- 文档对象模型 (Document Object Model, DOM) 是什么以及如何使用。
- Active X 数据对象 (ADO) 怎样使用 XML 以及如何提高性能。
- 如何才能使 XML 看上去更美观。

尽管立刻得到全面支持还不大可能，但我们已经看到 XML 支持确实已引入到 IE 和 ADO 中。困难在于 IE 和 ADO 的发展速度不一致，所以它们之间的交互还不够理想，故本章没有介绍人们所希望的有关数据传输的万能技术。在写这本书时，IE 和 ADO 的结合还不是紧密，但它们都在不断改进。所以尽管现在还没有新版本发布的具体消息，但 ADO 和 IE 将来肯定会更好地结合。

11.1 XML 的定义

在给 XML 一个明确的定义之前，最好了解什么是标记语言。首先这里有一个问题，因为“语言”这个术语用得并不恰当。实际上，XML 并不是编程语言，而 VB 或 C++ 才是真正的编程语言，XML 只是定义了如何标记文本或文档的一套规则。那么“标记”又是指什么？标记一个文档是指标识出文档的某些部分有特殊含义的过程。这可能难以理解，我们举一个有关超文本标记语言 (Hypertext Markup Language, HTML) 的例子，因为“HTML”中的“M”就代表了标记 (Markup)。

HTML 是一套规定文档布局的标记。HTML 包含了一些预先定义好的标记，每一种标记都有各自的含义，例如：

```
<BODY>
Here we have some text
<H1>This is a heading</H1>
This bit is normal text
<B>This is some bold text</B>
And finally some more normal text
</BODY>
```

这是一个含有少量标记的文本。文本以 `<BODY>` 标记开始，在 HTML 里，该标记表示文档主体的开始，主体部分以 `</BODY>` 标记结束。在这个文档的主体内有标题，放在 `<H1>` 和 `</H1>` 标记之间；另外还有一些粗体文本，放在 `` 和 `` 标记之间。这些文本标记为有特殊含义的文本。

你可能注意到上面的例子没有使用格式化这个词。这是经过仔细考虑的，因为标记和格式化并不是一回事。`<BODY>` 标记标定的是文档的一块区域，并没有定义任何格式。然而，`` 标记却标定文档的这一块区域用粗体显示。这是因为 HTML 中的 `` 标记是隐含了指定格式的标记。

所以请记住，标记语言只是一种规则，定义了如何给一篇文档中的特定部分增加特殊含义。这种定义可能会起到很好的格式化作用，但这并不是使用标记的唯一原因。

11.1.1 XML 和 HTML 的差别

尽管 XML 和 HTML 都使用标记，但是它们是不同的。其中最主要的区别是 XML 专门用来描述文本的结构，而不是用于描述如何显示文本。XML 并没有一套固定的标记，例如：

```
<BODY>
Here we have some text
<H1>This is a heading</H1>
This bit is normal text
<B>This is some bold text</B>
And finally some more normal text
</BODY>
```

上面的代码看起来与上一节的 HTML 代码不是完全一样吗？如果它是 HTML 文档，的确一样。如果把其加载到浏览器，以上内容就会显示如图 11-1 所示的结果，其作用好像是格式化文档。

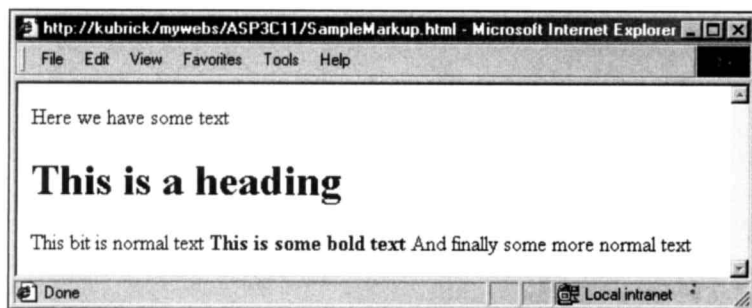


图 11-1 浏览器中显示的 HTML 文档

但是，如果上面的代码是 XML 文档，那么其中的标记就不具有任何含义，其内容只是说明：

- 有一个名为 BODY 的标记，在这个标记里面有一些文本。

- 有一个名为H1的标记，在这个标记里面有一些文本。
- 有一个名为B的标记，在这个标记里面有一些文本。

如果以上代码作为一个XML文档(文件的扩展名为.xml)加载到IE浏览器中，可以看得非常清楚，其结果如图11-2所示。

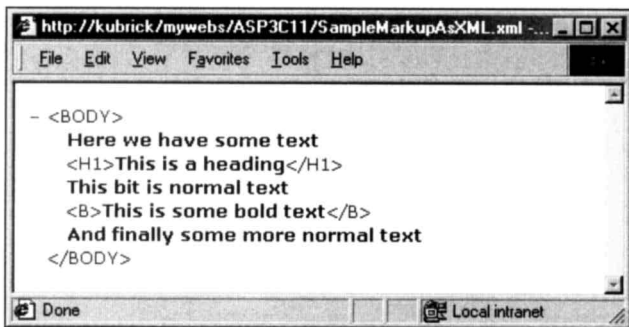


图11-2 浏览器中显示的XML文档

IE解释该XML文档并将其显示出来。请注意，IE并没对这个XML文档做任何处理，仅仅是显示出来而已。浏览器知道如何解释HTML文档，并且知道如何用标记所定义的格式来显示文档。同样，浏览器也知道如何解释XML文档，但由于XML标记不定义格式，所以文档不会得到格式化，于是原样显示这些标记。

但IE实际上还是做了一点格式编排，使XML更易读了。它把标记分为不同的层次，所以我们看到了一套结构化的标记，且IE没有解释这些标记。

迄今为止所学的XML文档由标记文档某些部分的标记组成。那么XML使用数据会有什么问题呢，再看另一个例子。这个例子在前面的章节中出现过，读者会发现XML在这里十分有意义。

```
<Authors>
  <Author>
    <au_id>172-32-1176</au_id>
    <au_lname>White</au_lname>
    <au_fname>Johnson</au_fname>
  </Author>
  <Author>
    <au_id>213-46-8915</au_id>
    <au_lname>Green</au_lname>
    <au_fname>Marjorie</au_fname>
  </Author>
  <Author>
    <au_id>238-95-7766</au_id>
    <au_lname>Carson</au_lname>
    <au_fname>Cheryl</au_fname>
  </Author>
</Authors>
```

上面的例子中使用了几对不同的标记。一开始，你可能会认为这些标记一定有其含义。他们都有一个有意义的名称，定义了Author的一个列表、单个Author以及一些与Author有关的值。在前面的章节中，这些内容出现了多次，当在一个浏览器中查看时，我们可以把它格式化成一个表来显示。但因为这是XML，XML中的标记不代表任何意义，如图11-3所示。



图11-3 IE 浏览器中显示的XML

可见，这里IE没有对其进行任何处理。所以即使这些标记对我们来说是有意义的，但它们对XML却没有。事实上，这段代码可以写成如下形式：

```
<HubbaHubbaHubbas>
  <HubbaHubbaHubba>
    <foo>172-32-1176</foo>
    <bar>White</bar>
    <qwerty>Johnson</qwerty>
  </HubbaHubbaHubba>
  <HubbaHubbaHubba>
    <foo>213-46-8915</foo>
    <bar>Green</bar>
    <qwerty>Marjorie</qwerty>
  </HubbaHubbaHubba>
  <HubbaHubbaHubba>
    <foo>238-95-7766</foo>
    <bar>Carson</bar>
    <qwerty>Cheryl</qwerty>
  </HubbaHubbaHubba>
</HubbaHubbaHubbas>
```

浏览器只是把这些标记原封不动地显示出来，如图 11-4所示。

标记可以是你所喜欢的任意符号。当然，一开始就给它一个有意义的名字是很直观的。XML的可读性十分强，所以一般使用能描述其内容的标记名字。

在此，已经看到XML由一系列能描述文档各部分的标记组成。在以上使用 Author信息的例子中，使用XML来描述数据，使用了代表数据字段名的标记名。这就是XML能作为一种数据交换格式的真正含义。它是标准的文本，所以可以很容易地从一台机器传送到另一台机器。但它并不是一种专用格式，所以任何人都可以读懂，并且如果标记名取得有意义的话，XML数据就具有“自我描述”的功能。



图11-4 浏览器显示的代码

现在你应该理解了XML数据的含义，下面再来看一些术语，以及XML的不同方法。

11.1.2 标记和元素

使用了“标记”这个名字来确定某些HTML的标记，比如或<H1>。元素是指利用这些标记而形成的一个整体。例如：

```
<B>Some bold text and <I>italic</I> text</B>
```

这一行由两个开始标记和两个结束标记组成，但只有两个元素。B元素由下面这一行组成：

```
<B>Some bold text and <I>italic</I> text</B>
```

而I元素由下面这一行组成：

```
<I>italic</I>
```

因此，一个元素由一个开始标记和一个结束标记组成，它们把文本包围在中间，其中也可以包括其他子元素。这一点很重要，因为这牵涉到一个“形式化的XML”的概念，其中每一个开始标记都必须有一个对应的结束标记。这一点与HTML 4.0及其以前的版本不同，在那些HTML版本中，某些标记没有结束标记(例如和
)和<P>标记。

如果使用XML来描述数据，有可能在一些域中不包含数据。在这种情况下，标记就为空。在XML中有两种方法可以定义空标记。第一种方法是使用一个开始标记和一个结束标记，但其中没有内容：

```
<TagName></TagName>
```

第二种方法只使用一个开始标记，但在标记后面加上斜杠：

```
<TagName/>
```

形式化的XML的另一层含义是XML的标记是大小写敏感的，所以在这种情况下开始标记和结束标记必须一致。这也意味着下面这一行是无效的XML：

```
<TAGName></tagName>
```

1. 根标记

另一个需要知道的术语是根标记。它表示最外层的标记，一个 XML 文档只能有一个根。

例如再来看看 Author 的例子：

```
<Authors>
  <Author>
    <au_id>172-32-1176</au_id>
    <au_lname>White</au_lname>
    <au_fname>Johnson</au_fname>
  </Author>
  <Author>
    <au_id>213-46-8915</au_id>
    <au_lname>Green</au_lname>
    <au_fname>Marjorie</au_fname>
  </Author>
  <Author>
    <au_id>238-95-7766</au_id>
    <au_lname>Carson</au_lname>
    <au_fname>Cheryl</au_fname>
  </Author>
</Authors>
```

这里的根标记是 <Authors>。因为这里只有一个根标记，故以上表述是合法的。但下面的代码是错误的：

```
<Authors>
  <Author>
    <au_id>172-32-1176</au_id>
    <au_lname>White</au_lname>
    <au_fname>Johnson</au_fname>
  </Author>
</Authors>
<Authors>
  <Author>
    <au_id>213-46-8915</au_id>
    <au_lname>Green</au_lname>
    <au_fname>Marjorie</au_fname>
  </Author>
  <Author>
    <au_id>238-95-7766</au_id>
    <au_lname>Carson</au_lname>
    <au_fname>Cheryl</au_fname>
  </Author>
</Authors>
```

这个例子里在顶层上有两个标记，因此这是无效的。

2. <?xml> 标记

这不是一个真正的 XML 标记，而是一个代表特殊处理指令的标记。<?xml> 标记是用于每个 XML 文档首行的特殊标记，可以确定版本号 and 语言信息。例如：

```
<?xml version="1.0"?>
```

这一行确定了 XML 的版本，即当前缺省的版本是 1.0 版。目前，1.0 版也是 XML 的唯一版本，但在 XML 文档中指定后，可在以后校对版本信息。

这个标记也可以定义 XML 数据中使用的语言种类。这很重要，因为在数据中包含的字符可能不是标准的英文 ASCII 码字符集。可以通过在“?xml”处理指令中增加 encoding 属性来

指定文档所用的编码方法。

```
<?xml version="1.0" encoding="iso-8859-1" ?>
```

表11-1是一些常用语言及其字符集的列表。

表11-1 常用语言及其字符集

语 言	字 符 集
Unicode(8位)	UTF-8
Latin 1(Western Europe, Latin America)	ISO-8859-1
Latin 2(Central/Eastern Europe)	ISO-8859-2
Latin 3(SE Europe)	ISO-8859-3
Latin 4(Scandinavia/Baltic)	ISO-8859-4
Latin/Cyrillic	ISO-8859-5
Latin/Arabic	ISO-8859-6
Latin/Greek	ISO-8859-7
Latin/Hebrew	ISO-8859-8
Latin/Turkish	ISO-8859-9
Latin/Lappish/Nordic/Eskimo	ISO-8859-10
Japanese	EUC-JP或Shift_JIS

如果了解更多的有关国际化的细节，可以参考 W3C中关于这方面主题的网页，其网址是www.w3c.org/International/。

3. 属性

与HTML一样，XML也用属性定义元素的特性，并且也必须是形式化的。属性必须用引号封闭起来，例如：

```
<BOOK ISBN="1-861002-61-0">Professional Active Server Pages 3.0</BOOK>
```

HTML允许不用引号，但在XML中这是非法的。

4. 特殊字符

有一些在标准的XML字符串中不能使用的特殊字符，如表 11-2所示。

表11-2 XML中的特殊字符及其替代字符

字 符	替代的字符
&	&
<	<
>	>
"	"
'	'

例如下面的XML是非法的：

```
<BOOK>Advanced Rocket Science Dave & Al</BOOK>
```

而以下XML是合法的：

```
<BOOK> Advanced Rocket Science by Dave &amp; Al</BOOK>
```

大多数能自动生成XML的工具把这些字符转换成特殊的对应字符，但在自己生成 XML时，要注意必须使用长格式。在 XML文档中使用这些单独的字符是非法的，如果文档中有这些字符，大多数XML处理程序将出错。

11.1.3 模式和文档类型定义

我们一开始就声明了 XML 标记实际上不代表任何意义，可以给标记取任何名字，但怎么才能知道什么类型的标记在文档中是允许使用的？因此必须使用文档类型定义（Document Type Definition, DTD）或模式（Schema）。模式和 DTD 的功能几乎是一样的，都规定了哪些元素在文档中可以使用，并可以把一个形式化的 XML 文档转变为一个有效的 XML 文档。也就是说它被正确标记（即有良好形式），并且只包含允许的元素和属性。

既用 DTD 也用模式的原因是因为微软觉得 DTD 在有些地方显得比较笨拙。DTD 是一种定义 XML 文档结构的文本文件，但 DTD 本身并不是 XML，有完全不同的语法规则。这就有点反常，所以在这一点上我们赞成微软的观点。如果处理 XML 文档，那么定义那些文档的结构也应该是 XML，这就是模式所做的，即模式是 DTD 的 XML 等价物。

下面看一个典型的 DTD，它用于 authors XML 文档，来自于 pubs 数据库：

```
<!ELEMENT DOCUMENT (AUTHOR+)>
<!ELEMENT AUTHOR (au_id, au_lname, au_fname, phone, address,
                    city, state, zip, contract)>
<!ELEMENT au_id (CDATA)>
<!ELEMENT au_lname (CDATA)>
<!ELEMENT au_fname (CDATA)>
<!ELEMENT phone (CDATA)>
<!ELEMENT address (CDATA)>
<!ELEMENT city (CDATA)>
<!ELEMENT state (CDATA)>
<!ELEMENT zip (CDATA)>
<!ELEMENT contract (CDATA)>
```

这实际上很简单。它说明了文档由零个或多个 AUTHOR 单元组成。AUTHOR 后面的加号指的是“一个或多个”。每个 AUTHOR 元素由九个其他元素构成。每个子元素都包含字符数据（CDATA）。

DTD 有两个缺点：

- 不是 XML。
- 不能为每一个元素指定数据类型，比如整数、日期等。CDATA 仅表示元素只包含字符数据，并不确定元素内容的实际类型。

因为这些原因，微软向 W3C 建议使用模式。如果将上面的 DTD 文档转换为模式，上面的内容将变成：

```
<Schema ID="Author">
  <Element name="au_id"/>
  <Element name="au_lname"/>
  <Element name="au_fname"/>
  <Element name="phone"/>
  <Element name="address"/>
  <Element name="city"/>
  <Element name="state"/>
  <Element name="zip"/>
  <Element name="contract"/>
</Schema>
```

再加上数据类型，将得到：

```
<Schema ID="Author">
  <Element name="au_id" type="string"/>
  <Element name="au_lname" type="string"/>
  <Element name="au_fname" type="string"/>
```



```
<Element name="phone" type="string"/>
<Element name="address" type="string"/>
<Element name="city" type="string"/>
<Element name="state" type="string"/>
<Element name="zip" type="string"/>
<Element name="contract" type="boolean"/>
</Schema>
```

现在模式不仅详细指出了可使用的元素，而且指明了数据类型。虽然 DTD的CDATA等价于一个字符串，但模式还允许其他数据类型，例如 contract元素就包含布尔型数据。

但有一点很重要，模式仍不是一个标准，所以这里没有更详细地叙述其结构和布局，以及如何使用它们。本章的内容实际上是关于如何在 ASP应用程序中使用XML数据，这可能也意味着将使用微软的XML工具。微软使用自己的模式格式，别的公司使用 DTD，现在还没有统一的标准。在本章后面的部分，将介绍当 ADO生成XML时所使用的模式。

因为以上内容是XML的一个核心部分，所以不得不提到它们，并且你在其他文档中也能看到模式和DTD。然而使用XML数据的方法对模式本身并没有很大影响，因此我们把有关这方面的详细讨论写到了其他书中。

如果感兴趣的话可以参考W3C Web站点(网址www.w3c.org/XML/)中的详细介绍。

11.1.4 名称空间

XML有一个问题：由于可以给一个元素取任意的名字，所以很有可能使用了别人用过的相同名字，或甚至在不同的XML文档中使用了相同的名字来表示不同的意思。例如，考虑下面的代码：

```
<contract>Yes</contract>
```

这是从pubs数据库中的authors表中取出来的，表示该作者是一个签过合同的作者。但是如果一个XML文档中包含以下内容，情况会是怎样？

```
<contract>F:/contacts/1999.doc</contract>
```

这Contract元素确定了包含合同的文档。

当这两个XML文档分开时，不会出现问题，但如果把这两个文档结合起来，如何分辨contract元素到底属于哪一个文档？这就是需要有名称空间(namespace)的原因，因为名称空间唯一地确定了哪个元素属于哪个模式。

通过在根标记中定义属性xmlns，将名称空间加入到XML文档，这需要一个统一资源标识符(Uniform Resource Identifier, URI)。这个URI只是一个能唯一地确定名称空间的名称。尽管任何唯一的名字都可以用作URI，但你会发现我们经常使用Web站点的URL。例如看一下contract的问题，它有两个不同类型的值。如果把来自两个不同源的XML数据组合起来，将得到以下结果：

```
<Authors>
  <Author>
    <au_id>172-32-1176</au_id>
    <au_lname>White</au_lname>
    <au_fname>Johnson</au_fname>
    <contract>Yes</contract>
    <contract>F:/contacts/1999.doc</contract>
  </Author>
</Authors>
```

如何才能分辨这两个 contract 元素？答案就是使用名称空间。于是，把它加入 XML 文档：

```
<Authors xmlns:pubs="http://www.wrox.co.uk/ms/PubsDB"
          xmlns:wrox="http://www.wrox.co.uk/authors">
  <Author>
    <au_id>172-32-1176</au_id>
    <au_lname>White</au_lname>
    <au_fname>Johnson</au_fname>
    <pubs:contract>Yes</pubs:contract>
    <wrox:contract>F:/contacts/Johnson1999.doc</wrox:contract>
  </Author>
</Authors>
```

这里通过加入 xmlns 属性来确定两个名称空间。其结构是：

```
xmlns:short_name=URI
```

short_name 就是在 XML 文档中用来联系元素和标记的名称。你很快将看到如何使用它。URI 只是一个名字，在 XML 文档中唯一地确定名称空间。这里需要着重注意的是，URI 只是纯粹的一个名字，实际上并不连接到 URL，也不隐含任何与 Web 服务器的连接。这只是一地确定每个名称空间的方法。只要它们在 XML 文档中是唯一的，可以用任何名称来命名。

在 XML 元素中，通过在标记前面加上适当的 *short_name* 来指定元素属于哪一个名称空间。

```
<pubs:contract>Yes</pubs:contract>
<wrox:contract>F:/contacts/1999.doc</wrox:contract>
```

这里，第一个元素属于名为 pubs 的名称空间，第二个元素则属于名为 wrox 的名称空间。也可以将名称空间应用于属性，例如：

```
<Authors xmlns:pubs="http://www.wrox.co.uk/ms/PubsDB"
          xmlns:wrox="http://www.wrox.co.uk/authors">
  <Author>
    <pubs:au_id pubs:type="UID">172-32-1176</pubs:au_id>
    <au_lname>White</au_lname>
    <au_fname>Johnson</au_fname>
    <pubs:contract>Yes</pubs:contract>
    <wrox:contract>F:/contacts/Johnson1999.doc</wrox:contract>
  </Author>
</Authors>
```

这里把名称空间应用于 au_id 元素的 type 属性，通过使用不同的名称空间或不使用名称空间把这个 type 属性与文档中其他元素的 type 属性唯一地区分开来。

以上主要讲述的意思是名称空间能确保正确地解释数据。如果 XML 文档含有某些需要进行特殊处理的信息，或者说包含一些特殊的信息，那么名称空间可以识别它们。使此 XML 与那些凑巧包含相同元素或属性的 XML 区分开。

同样，这是一个需要花费大量时间进行研究的领域，它已经超出了本章的范围。我们的真正目的只是想让读者理解什么是名称空间，在后面你会看到更多有关这方面的内容。在此并不想详细地研究它们，但现在你至少不会对它们是什么感到迷惑了。

11.1.5 文档对象模型

文档对象模型 (Document Object Model, DOM) 是针对 HTML 和 XML 文档的 API，定义了文档的逻辑结构以及访问它们的方法。这确实很重要，因为它定义了一个标准的访问和处理 XML 结构的方法。下面看一个简单的 XML 文档，然后再看看如何使用 DOM。

```
<Authors>
  <Author>
    <au_id>172-32-1176</au_id>
    <au_lname>White</au_lname>
    <au_fname>Johnson</au_fname>
  </Author>
  <Author>
    <au_id>213-46-8915</au_id>
    <au_lname>Green</au_lname>
    <au_fname>Marjorie</au_fname>
  </Author>
</Authors>
```

XML文档本质上是分层结构的文档，也就是说，总是有一个顶层或根元素，然后在下面有一些子元素。所以，上面的文档可以用图 11-5来表示。

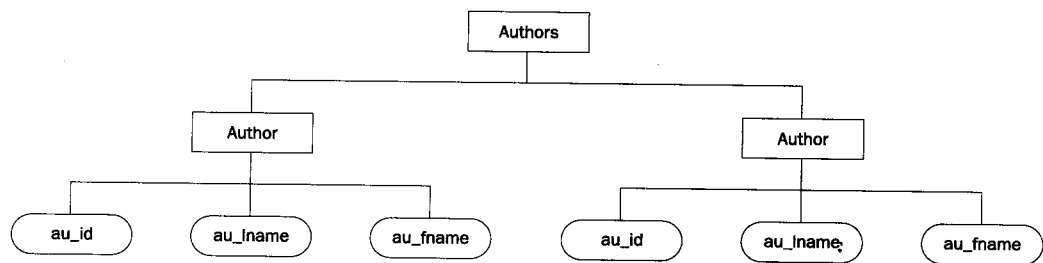


图11-5 XML文档的分层结构

如果子元素有很多，将会得到一个层次很深的树状图。用 DOM的术语来说，这些元素也称为节点(node)。一个节点代表树状结构中的一个普通元素。

1. 基本对象

为了表现这种分层结构的本质，DOM提供了整套的对象、方法及属性以使我们能处理DOM。在此不打算全部分析它们，但会讨论少数几个可能用到的对象和属性。首先介绍几个对象，如表 11-3所示。

表11-3 基本对象及说明

对 象	说 明
Node	在分层结构中的单个节点
NodeList	节点集合
NameNodeMap	允许用名字和索引号访问的节点集合

还有一些用来访问节点的属性，如表 11-4所示。

表11-4 属性及说明

属 性	说 明
childNodes	返回一个包含子节点的NodeList对象
firstChild	返回当前节点的第一个子节点
lastChild	返回当前节点的最后一个子节点
parentNode	返回当前节点的父节点
previousSibling	返回上一个兄弟节点，也就是在分层结构中相同层上的前一个节点
nextSibling	返回下一个兄弟节点，也就是在分层结构中相同层上的后一个节点
nodeName	节点的名字
nodeValue	节点的值

以上并没有列出全部的对象和属性，但能让我们明白使用它们可能做些什么。如果想得到全部的列表，可参考 MSDN帮助中有关XML的内容，也可在 msdn.microsoft.com/xml/ 中找到相关内容。

下面详细分析一下前面例子中所用的文档的节点结构，如图 11-6所示。

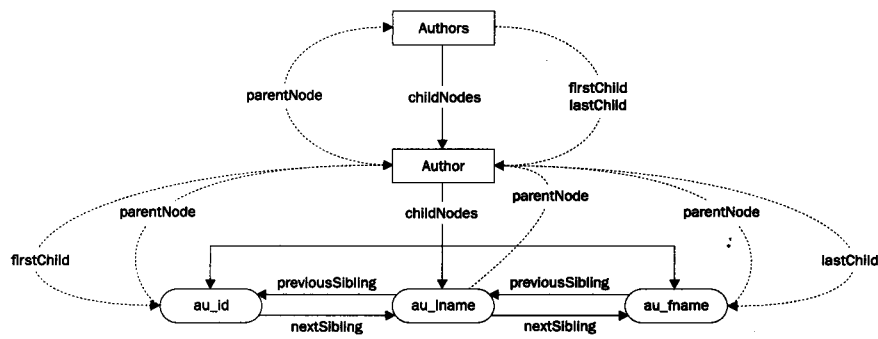


图11-6 例子中文档的节点结构

此时，可以很清楚了解如何使用这些属性在 XML DOM中进行定位。虚线箭头指出了该属性指向那个节点。根节点 Authors下的子节点存放于 childNodes集合中。在上面的例子中，由于根节点 Authors只有一个子节点，所以属性 firstChild和lastChild指向了同一个节点。在这种情况下，节点集合中唯一的节点就是 childNodes(0)。

节点 Author有三个子节点，存放于一个 childNodes集合中。属性 firstChild(也就是 childNodes(0))指向节点 au_id，属性 lastChild(也就是 childNodes(2))指向节点 au_fname。而属性 previousSibling和nextSibling则在同一层分别指向节点集合中上一个和下一个兄弟节点。

那么，假定有一个指向 Authors，名为 nodRoot的节点对象，则节点的代码如表 11-5所示。

表11-5 节点的代码表示法

代 码	指向的节点
nodRoot.childNodes(0)	Author
nodRoot.childNodes(0).firstChild	au_id
nodRoot.childNodes(0).firstChild.nextSibling	au_lname
nodRoot.childNodes(0).firstChild.parentNode	Author
nodRoot.childNodes(0).firstChild.nextSibling.parentNode	Author

在这一章后面，你会看到更多的有关这方面的例子。

2. 特殊的DOM对象

XML是可扩展的，主要用于满足文档的多样性，所以针对不同的节点类型有一些特殊的对象。这些对象继承了 Node对象的大部分方法和属性，同时也增加了一些特别的方法与属性用于处理那些特殊的节点类型。这些对象及说明如表 11-6所示。

表11-6 DOM的特殊对象及说明

对 象	说 明
Document	XML文档的根对象
DocumentType	与XML有关的DTD或模式的信息。等同于 DTD中的!DOCTYPE
DocumentFragment	Document的轻量拷贝，用于临时存储或文档的插入操作

(续)

对 象	说 明
Element	一个XML元素
Attribute或Attr	一个XML属性
Entity	一个经过分析或未经过分析的实体。等同于 DTD中的!ENTITY
EntityReference	一个实体的引用
Notation	一个符号, 等同于 DTD中的!NOTATION
CharacterData	文档中用于文本信息的基本对象
CDATASection	未经过分析的字符数据。等同于 DTD中的!CDATA
Text	元素或属性节点的文本内容
Comment	一个XML注释元素
ProcessingInstruction	一条处理指令, 存放于<? ?>段中
Implementation	应用程序特殊的实现细节

一般不会用到所有这些对象, 这里也不打算深入地解释它们以及它们与 XML的关系, 因为这些内容超出了本章的范围。我们主要关心的是作为数据的 XML的一些基础知识, 以及如何使用和处理它们。

如果想了解有关 XML和DOM的更多知识, 可以参考 Wrox出版的《XML IE5 Programmer's Reference》, ISBN 1-861001-57-6。也可以通过查看微软的网站 <http://msdn.microsoft.com/xml/> 或W3C的网站<http://www.w3.org/DOM/>来获得这方面的资料。

3. 使用DOM

因为XML是一个相对较新的领域, 许多浏览器都不支持 XML数据的处理。事实上, IE 5.0才是唯一对其支持较好的浏览器。所以, 作为一个 ASP开发者, 必须决定是否将 XML数据发送到浏览器, 还是在 ASP页面上处理XML数据并向浏览器发送纯粹的 HTML。当然, 这取决于接收数据的一方。但不论使用哪种方法, 懂得如何使用 DOM以及如何访问XML对象是有用的。

因此, 这一节通过使用一个名为 TraverseXML.html的网页, 将让我们了解, 如何利用 DOM来访问XML文档, 可在下载的代码中找到这个网页。这里计划采用 IE 5, 带有一个XML数据岛(XML Data Island)。在上一章节学习RDS时, 见过这个术语。数据岛其实只是一个与数据控件作用相似的HTML标记。

```
<XML ID="dsoData" SRC="authors.xml"></XML>
```

此时, 有了一个名为 dsoData的数据岛, 包含来自 XML文件authors.xml的数据。既然是数据岛, 就不会在屏幕上显示任何内容, 即它只是一个数据容器。所以, 需要一种从中提取并显示数据的方法。

为了做到这一点, 使用一个 SPAN元素, 当然也可以使用其他几种类型的元素。

```
<SPAN ID="txtData"></SPAN>
```

我们的目的是利用 DOM从数据岛中提取 XML信息, 并将数据在 SPAN元素中显示出来。先从根节点开始, 找到所有子节点并显示它们的详细信息。我们将显示节点的名称、类型及其值。同样, 既然子节点能包含子节点, 所以可以重复以上的过程, 遍历整个树状结构, 直到所有的节点都被访问到。因为这是一个有关树的遍历操作, 所以可以使用一个递归的函数

来实现。

节点的类型是要显示的信息之一。这可以用一个整数来识别，但鉴于数字的可读性差，所以将其转换为一个描述节点类型的字符串。为了做到这一点，事先需要声明一个全局数组，其中包含描述节点类型的文本，同时使用实际的节点类型数字作为索引。在 JScript 代码的最上端，放置如下代码：

```
var g_strNodeTypes = new Array('', 'ELEMENT (1)', 'ATTRIBUTE (2)',
    'TEXT (3)', 'CDATA SECTION (4)', 'ENTITY REFERENCE (5)',
    'ENTITY (6)', 'PROCESSING INSTRUCTION (7)', 'COMMENT (8)',
    'DOCUMENT (9)', 'DOCUMENT TYPE (10)', 'DOCUMENT FRAGMENT (11)',
    'NOTATION (12)');
```

递归函数 showChildNodes 接收一个 XML 节点，以及一个代表当前节点在分层结构中的层次的整数。这允许我们输出层次化内容，因此很容易读懂：

```
function showChildNodes(nodNode, intLevel)
{
    var strNodes = '';           // string containing the node information
    var intCount = 0;           // count of the nodes
    var intNode = 0;            // current node number
    var nodAttrList;            // node list of the attributes for a node
```

接下来可以开始构建字符串，首先是当前节点的名称、类型及其值。节点类型由一个整数来确定，这里通过使用前面定义的数组来获得有关节点类型的描述。getIndent 函数只返回一个空字符串，包含的空格数目最多等于树的层次：

```
// Get the values for this node
strNodes += getIndent(intLevel) + '<B>' + nodNode.nodeName
    + '</B> &nbsp; Type: <B>' + g_strNodeTypes[nodNode.nodeType]
    + '</B> &nbsp; Value: <B>' + nodNode.nodeValue + '</B><BR>';
```

下一步需要考虑这一节点是否有属性，如果有的话，必须再次遍历，并加到字符串中：

```
// Check there are some attributes
nodAttrList = nodNode.attributes;
if (nodAttrList != null)
{
    intCount = nodAttrList.length;
    if (intCount > 0)
    {
        // For each attribute, display the attribute information
        for (intAttr = 0; intAttr < intCount; intAttr++)
            strNodes += getIndent(intLevel + 1) + '<B>'
                + nodAttrList(intAttr).nodeName + '</B> &nbsp; Type: <B>'
                + g_strNodeTypes[nodAttrList(intAttr).nodeType]
                + '</B> &nbsp; Value: <B>'
                + nodAttrList(intAttr).nodeValue + '</B><BR>';
    }
}
```

最后，需要检查该节点是否还有子节点，对于其每一个子节点，调用相同的函数。对于碰到的每个节点重复构建以上字符串：

```
// Check for any child nodes
intCount = nodNode.childNodes.length;
if (intCount > 0)
    // For each child node, display the node,
    // attributes and its child node information
```



```

for (intNode = 0; intNode < intCount; intNode++)
    strNodes += showChildNodes(nodNode.childNodes(intNode), intLevel + 1);

return strNodes;
}

```

为了显示输出结果，可以使用如下代码：

```

domXMLData = dsoData
txtData.innerHTML = showChildNodes(domXMLData, 0);

```

这里只是调用了该函数，并将顶层节点传入函数中。结果如图 11-7所示。

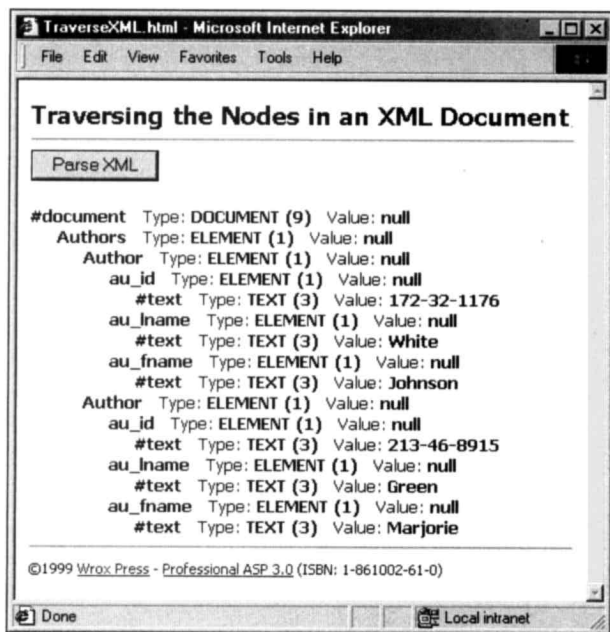


图11-7 显示的XML文档的节点

这里，你看到了XML的DOM的递归特性。图 11-7的最上面是一个#Document类型的节点，是所有继承者的父节点，即所有 XML文档的根节点。但注意，它并不是一个真正的元素，其类型为Document。所以XML数据的根是XML文档，但其下面是XML元素。

第一个元素，或者说根元素是 Authors元素。它包含每一个 Author元素，以及Author的每一个属性元素。同时，需要注意的是端节点（也就是没有子节点的节点）有另一个名为 #text的节点。它包含实际的节点文本，文本的内容令许多人觉得不解。为什么每个元素都有一个 null值，而它的子元素 #text却含有节点的文本值？为什么不让节点的值包含节点的文本？答案是很显然的，因为节点除了包含文本外，还可能包含子节点。如果一个节点既包含文本又包含节点，那么此节点的值应该是什么？是文本，还是一个子节点的指针？由于存在着这种多义性，W3C最终规定总将一个节点的文本放在一个类型为 TEXT的子节点中。

但你可能会奇怪：为什么访问节点的值时，并没有移到树的另一层来访问它的子节点？请看下一行代码：

```

+ '</B> &nbsp; Value: <B>' + nodNode.nodeValue + '</B><BR>';

```

这里只用了nodeValue属性，这是微软对DOM的一个扩展。在W3C的规范上，并没有提供

一种简单地访问节点值的方法，必须移到子节点来访问与之关联的 TEXT 节点。由于微软认为访问节点的值并获取其所含文本可能是最常见的操作，所以引入了 `nodeValue` 属性，它处理 TEXT 子节点。

这里，已经简单介绍了 DOM 是什么，并说明了它以树状结构保存 XML 数据。现在，你应该明白 XML 是什么以及如何访问它，接下来需要了解一下 XML 是如何与 ADO 和 IE 浏览器紧密结合的。

11.2 ADO 和 IE 5 中的 XML

ADO 2.5 的最大特点之一是：允许以 XML 文档的形式访问一个记录集。其优点在于能提供一种不同的处理数据的方法。你可能认为这并不是很重要，那么考虑一下下面的因素：

- 并不是每个处理数据的程序员都懂 ADO。许多程序员只是擅长于 HTML 或脚本语言，因此为了能通过 XML 访问数据，DOM 对他们来说应该是更可取的。
- 并不是每个程序员都既懂 XML 又懂 DOM，因此允许通过记录集来访问 XML 文档就意味着他们能充分利用自己所掌握的技术。
- 数据处理工作大多是关于数据的显示。如果用户不能访问和看见他们，这些数据的用处就不大，而 XML 提供了解决这一问题的方法。
- 用 XML 来编写的内容越来越多。
- 开发出来的 XML 工具越来越多。

以上这些观点加起来足可以说服我们学习 XML。

11.2.1 存储为 XML 形式的 ADO 记录集

一提到 XML 和 ADO，首先需要了解的是 ADO 是如何以 XML 的形式呈现数据的。一会儿将介绍从 ADO 中提取数据的方法，现在先来看一下 XML 的数据格式。

如果看一段直接由 ADO 转化来的 authors 表，你可能会感到一点惊讶：

```
<z:row au_id="172-32-1176" au_lname="White" au_fname="Johnson"
      phone="408 496-7223" address="10932 Bigge Rd."
      city="Menlo Park" state="CA" zip="94025" contract="True" />
<z:row au_id="213-46-8915" au_lname="Green" au_fname="Marjorie"
      phone="415 986-7020" address="309 63rd St. #411"
      city="Oakland" state="CA" zip="94618" contract="True" />
<z:row au_id="238-95-7766" au_lname="Carson" au_fname="Cheryl"
      phone="415 548-7723" address="589 Darwin Ln."
      city="Berkeley" state="CA" zip="94705" contract="True" />
```

这不像过去使用的 XML。在表中每一行对应一个元素，而不是每个字段对应一个元素，字段只是行元素的属性。为什么会这样？微软使用这种方法的原因是因为基于元素的 XML 文档相当繁琐。如果只用元素将一个大的记录集转换为 XML 文档，它将极为巨大。因为每一个字段都有一个开始标记和一个结束标记。因此，这种为每个数据项使用属性来定义数据的方式就减少了不必要的重复工作。

尽管微软正在寻找一种类似开关的技术来使用基于元素的方法，但以上方法是目前唯一能从 ADO 中提取 XML 数据的方法。在写这本书时，还不清楚在 ADO 2.5 版本中是否用到了这种技术，或者在今后的版本中是否会用到它。

11.2.2 ADO记录集名称空间

如果使用ADO发送和检索XML数据，会见到一个与XML数据文件关联的模式。这个模式嵌入到XML文档的顶端。这里有必要研究一下。首先要注意的是根元素，它确定了几个名称空间：

```
<xml xmlns:s="uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882"
      xmlns:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"
      xmlns:rs="urn:schemas-microsoft-com:rowset"
      xmlns:z="#RowsetSchema">
```

这些名称空间如表 11-7所示。

表11-7 名称空间及说明

名称空间	说 明
s	确定模式的 URI
dt	确定数据类型的 URI
rs	确定行集(记录集)
z	确定单独的行

记住，名称空间能唯一地确定 XML文档或其中的元素。使用名称空间保证了元素名应用于正确的模式。

11.2.3 ADO记录集模式

那么，可用一个名称空间来定义模式和记录集的独特属性，看一下模式本身：

```
<s:Schema id="RowsetSchema">
  <s:ElementType name="row" content="eltOnly">
    <s:AttributeType name="au_id" rs:number="1" rs:writeunknown="true">
      <s:datatype dt:type="string" dt:maxLength="11" rs:maybenull="false" />
    </s:AttributeType>
    <s:AttributeType name="au_lname" rs:number="2" rs:writeunknown="true">
      <s:datatype dt:type="string" dt:maxLength="40" rs:maybenull="false" />
    </s:AttributeType>
    <s:AttributeType name="au_fname" rs:number="3" rs:writeunknown="true">
      <s:datatype dt:type="string" dt:maxLength="20" rs:maybenull="false" />
    </s:AttributeType>
    <s:AttributeType name="phone" rs:number="4" rs:writeunknown="true">
      <s:datatype dt:type="string" dt:maxLength="12" rs:fixedlength="true"
        rs:maybenull="false" />
    </s:AttributeType>
    <s:AttributeType name="address" rs:number="5" rs:nullable="true"
      rs:writeunknown="true">
      <s:datatype dt:type="string" dt:maxLength="40" />
    </s:AttributeType>
    <s:AttributeType name="city" rs:number="6" rs:nullable="true"
      rs:writeunknown="true">
      <s:datatype dt:type="string" dt:maxLength="20" />
    </s:AttributeType>
    <s:AttributeType name="state" rs:number="7" rs:nullable="true"
      rs:writeunknown="true">
      <s:datatype dt:type="string" dt:maxLength="2" rs:fixedlength="true" />
    </s:AttributeType>
    <s:AttributeType name="zip" rs:number="8" rs:nullable="true"
      rs:writeunknown="true">
      <s:datatype dt:type="string" dt:maxLength="5" rs:fixedlength="true" />
  </s:ElementType>
</s:Schema>
```

```
</s:AttributeType>
<s:AttributeType name="contract" rs:number="9" rs:writeunknown="true">
  <s:datatype dt:type="boolean" dt:maxLength="2" rs:fixedlength="true"
    rs:maybenull="false" />
</s:AttributeType>
<s:extends type="rs:rowbase" />
</s:ElementType>
</s:Schema>
```

下面详细分析一下上述代码，你能对代码中不同的代码行有更深入的理解。

1. 行元素

先从确定模式的元素开始：

```
<s:Schema id="RowsetSchema">
```

这里有一个名为 Schema 的 XML 元素，这是代表模式的顶层元素。用名为 s 的名称空间来唯一地确定模式，并给模式一个 id 属性。这就像给任意类型的数据字段一个 ID 一样。在模式中，id 确定了模式本身。

接下来使用 ElementType 来定义元素：

```
<s:ElementType name="row" content="eltOnly">
```

这个元素有两个属性。第一个是 name 属性，给元素提供一个名字。上面这一行表明此元素的名字为 row，别忘了数据是如何为记录集中的每一行包含一行的。第二个属性为 content，确定了元素所保存的内容。其可能的值及说明如表 11-8 所示。

表11-8 content属性的值及说明

值	说 明
empty	此元素不能包含任何内容
textOnly	此元素只能包含文本，而不是其他元素
eltOnly	此元素只能包含其他元素，而不是文本
both	此元素既可以包含文本又可以包含其他元素

因此，这一行代码的作用是确定一个只能包含其他元素的行元素。如果查看一下 XML 数据，会看到下面的数据：

```
<z:row au_id="172-32-1176" au_lname="White" au_fname="Johnson"
  phone="408 496-7223" address="10932 Bigge Rd."
  city="Menlo Park" state="CA" zip="94025" contract="True" />
```

既然这本来就是一个空标记，你可能想把类型设为 empty；但是对模式来说，empty 意味着根本没有内容，包括没有属性。

2. 字段的属性

如果回顾一下 XML 数据，会发现字段的值其实是以属性来表示的。所以在模式中必须定义这些属性。这包括两部分内容，首先是定义属性本身，其次是定义属性的数据类型。

为了定义一个属性，首先必须定义属性的名字，它应该映射到数据字段的名字。另外定义两个其他的属性。第一个是为每个属性定义的唯一的一个数字，这是第一个属性，所以给其取值为 1，剩下的属性依次编号。另一个属性是 writeunknown，它定义了属性是否能被更新：

```
<s:AttributeType name="au_id" rs:number="1" rs:writeunknown="true">
AttributeType 还用 一个子元素定义属性的数据类型：
<s:datatype dt:type="string" dt:maxLength="11" rs:maybenull="false" />
```

这包含了数据的类型、最大长度以及属性是否允许为空值。

其他数据类型可能会包含一些不同的信息。例如，contract字段包含下面的内容：

```
<s:datatype dt:type="boolean" dt:maxLength="2" rs:fixedlength="true"
            rs:maybenull="false" />
```

这里多了一个属性，它用来确定数据类型是否包含固定长度的数据。

3. 数据类型

当从ADO中生成XML数据时，数据类型会自动产生，但如果想自己编写 XML，或为已经存在的XML创建模式，可能希望增加对数据类型的支持。毕竟，这是模式优于 DTD的一个方面。表11-9列出了XML数据模式支持的数据类型。

表11-9 XML数据模式支持的数据类型及说明

类 型	说 明
bin.base64	一个二进制对象
bin.hex	十六进制的八位组
boolean	0或1(0代表false，1代表true)
char	一个字符长度的字符串
date	一个ISO 8601日期类型，不带时间。格式为 yyyy-mm-dd
dateTime	一个ISO 8601日期类型，可以带时间。格式为 yyyy-mm-ddThh:mm:ss
dateTime.tz	一个ISO 8601日期类型，可以带时间和时区。格式为yyyy-mm-ddThh:mm:ss-hh:mm。时区用超过(+)或落后(-)于格林尼治标准时间(GMT)的小时数来表示
fixed.14.4	固定的浮点数，左边最多有14位整数位，右边最多有4位小数位
float	浮点数
int	整数类型
number	浮点数
time	一个ISO 8601时间。格式为 hh:mm:ss
time.tz	一个可带有时区的ISO 8601时间。格式为 hh:mm:ss-hh:mm
i1	一个八位整数(一个字节)
i2	一个十六位整数(二个字节)
i4	一个三十二位整数(四个字节)
r4	一个四位实数
r8	一个八位实数
ui1	一个八位无符号整数(一个字节)
ui2	一个十六位无符号整数(二个字节)
ui4	一个三十二位无符号整数(四个字节)
uri	统一资源标识符
uuid	一系列十六进制数字，代表一个唯一的标识符。GUID就是一个例子

W3C也规定了一些原始类型，如表 11-10所示。

表11-10 W3C规定的一些原始类型及说明

类 型	说 明
entity	代表XML的ENTITY类型
entities	代表XML的ENTITIES类型
enumeration	代表一个枚举类型
id	代表XML的ID类型
idref	代表XML的IDREF类型
idrefs	代表XML的IDREFS类型

(续)

类 型	说 明
nmtoken	代表XML的NMTOKEN类型
nmtokens	代表XML的NMTOKENS类型
notation	代表XML的NOTATION类型
string	代表字符串类型

11.2.4 IE数据岛和绑定

我们已经介绍了几个 XML数据岛的例子，回忆一下当时它们是如何创建的和如何将HTML元素绑定到XML数据岛。可以在一个HTML页面中用<XML>标记创建一个数据岛，例如：

```
<XML ID="dsoData" SRC="authors.xml"></XML>
```

这里引用了一个外部XML文件作为数据源。另外可以在XML标记内嵌入XML数据：

```
<XML ID="dsoData">
  <Authors>
    <Author>
      <au_id>172-32-1176</au_id>
      <au_lname>White</au_lname>
      <au_fname>Johnson</au_fname>
    </Author>
  </Authors>
</XML>
```

在前一章中详细讨论过绑定。绑定可以采用两种形式。第一种是绑定单个元素，例如：

```
<INPUT TYPE="TEXT" DATASRC="#dsoData" DATAFLD="au_id"></INPUT>
```

以上代码将单个元素绑定到数据源的一个字段。另一种方法是使用表绑定，做法是先把表绑定到数据源，然后再将表中的元素绑定到数据字段：

```
<TABLE DATASRC="#dsoData">
  <TR>
    <TD><INPUT TYPE="TEXT" DATAFLD="au_id"></INPUT></TD>
    <TD><INPUT TYPE="TEXT" DATAFLD="au_fname"></INPUT></TD>
  </TR>
</TABLE>
```

这个表将给出图 11-8所示的结果。

这里不再对此进行详细讨论，因为在前一章中已经分析过这个简单的例子。

1. 绑定分层的数据

迄今为止，所有分析的数据都相当简单，但是如果绑定一组由 ADO产生的XML数据的话，就会发现一个问题。这主要是因为 IE没有将模式当作数据的定义，也就是说 IE所理解的是下面的内容：

```
<XML>
  <s:Schema>
    . . . schema data
  </s:Schema>
  <rs:data>
```

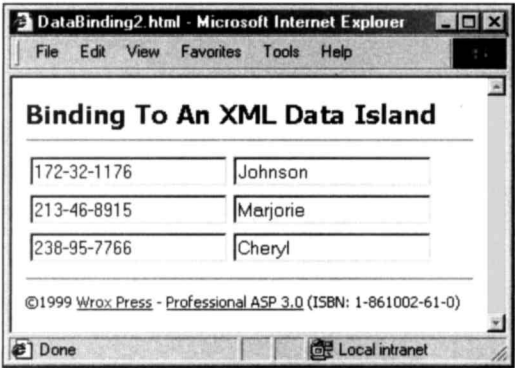


图11-8 显示绑定的数据

```

    <z:row . . ./>
    <z:row . . ./>
  </rs:data>
</XML>

```

即IE看到了两组数据，即模式和真实的数据。但这并不是问题的全部，别忘了我们需要的数据是放在行内的，所以数据有两个层次，这也意味着必须做一点额外的工作。

对这种分层的数据进行绑定的解决方法是(在databinding.html中有示范)使用两层绑定：

```

<TABLE ID="tblData" DATASRC="#dsoData" DATAFLD="rs:data">
<TR><TD>
  <TABLE ID="tblData" BORDER="1" DATASRC="#dsoData" DATAFLD="z:row">
    <TR>
      <TD><SPAN DATAFLD="au_id"></SPAN></TD>
      <TD><SPAN DATAFLD="au_fname"></SPAN></TD>

```

这里，先把外部的表与含有所有数据的元素进行绑定，然后把一个内部的表绑定到数据行。这样，各个字段就同以前一样全部绑定完毕。

如果有多层的分层结构，那么只需增加更多的层来进行绑定。例如，看一下下面的 XML 数据：

```

<XML ID="dsoPublishers">
<Publishers>
  <Publisher>
    <pub_id>0736</pub_id>
    <pub_name>New Moon Books</pub_name>
    <city>Boston</city>
    <state>MA</state>
    <country>USA</country>
    <title>
      <title_id>BU2075</title_id>
      <title>You Can Combat Computer Stress!</title>
      <price>$2.99</price>
      <pubdate>6/30/91</pubdate>
      <sale>
        <stor_id>7896</stor_id>
        <ord_num>X999</ord_num>
        <ord_date>2/21/93</ord_date>
        <qty>35</qty>
      </sale>
    </title>
    <employee>
      <emp_id>PDI47470M</emp_id>
      <fname>Palle</fname>
      <minit>D</minit>
      <lname>Ibsen</lname>
    </employee>
    <employee>
      <emp_id>KFJ64308F</emp_id>
      <fname>Karin</fname>
      <minit>F</minit>
      <lname>Josephs</lname>
    </employee>
  </Publisher>
</Publishers>
</XML>

```

以上数据包含了几个不同的数据层。首先是出版商 (publishers)，然后每个出版商对应一个雇员 (employees) 的列表和一个销售情况 (sales) 的列表。接下来，每个销售情况有一个销售书目的列表。图 11-9 展示了这种结构。

假定在 XML 中有这样的层次数据，并且想在浏览器中很快看见这些数据，那么需要做的就是获得包含出版商的顶层表。对于每个出版商，包含两个子表，一个包含雇员，一个包

含销售情况。而对于每个销售情况又有其他表来详细描述销售书目。

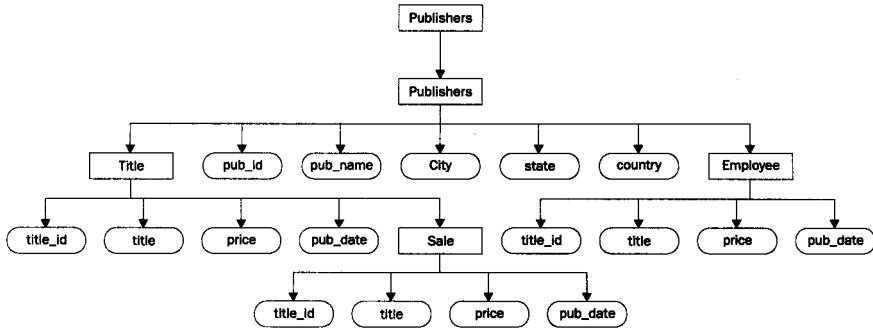


图11-9 多层分层结构

绑定这种多层数据的唯一方法是使用分层绑定。对于上面的 XML，可以按照下面的代码来实现嵌套的表：

```
<TABLE ID="tblPublishers" DATASRC="#dsoPublishers" BORDER="1">
  <TBODY>
    <TR>
      <TD><DIV DATAFLD="pub_name"></DIV></TD>
      <TD><DIV DATAFLD="city"></DIV></TD>
      <TD><DIV DATAFLD="state"></DIV></TD>
    </TR>
    <TR>
      <TD COLSPAN=4>
        <TABLE>
          <TR>
            <TD>Titles</TD>
          </TR>
          <TR>
            <TD COLSPAN=4>
              <TABLE DATASRC="#dsoPublishers" DATAFLD="Title" BORDER="1">
                <TR>
                  <TD><DIV DATAFLD="title"></DIV></TD>
                  <TD><DIV DATAFLD="price"></DIV></TD>
                </TR>
                <TR>
                  <TD COLSPAN=3>
                    <TABLE DATASRC="#dsoPublishers" DATAFLD="sale" BORDER="1">
                      <TR>
                        <TD><DIV DATAFLD="ord_date"></DIV></TD>
                        <TD><DIV DATAFLD="qty"></DIV></TD>
                      </TR>
                    </TABLE>
                  </TD>
                </TR>
              </TABLE>
            </TD>
            <TD>Employees</TD>
          </TR>
          <TR>
            <TD COLSPAN=4>
              <TABLE DATASRC="#dsoPublishers" DATAFLD="Employee" BORDER="1">
                <TR>
                  <TD><DIV DATAFLD="fname"></DIV></TD>
                  <TD><DIV DATAFLD="lname"></DIV></TD>
                </TR>
              </TABLE>
            </TD>
          </TR>
        </TABLE>
      </TD>
    </TR>
  </TBODY>
</TABLE>
```



```
</TR>
</TABLE>
</TD>
</TR>
</TBODY>
</TABLE>
```

这里有几个绑定的表，使用 XML 数据中的连续的层作为数据源，最后的结果如图 11-10 所示。

这里所做的只是在 HTML 中增加了一些绑定的表，而实际的数据仍由 XML 数据岛来提供。现在的显示结果并不是太美观，但只要稍加格式化，就能得到一个令人满意的显示结果，如图 11-11 所示。

如果觉得 HTML 绑定有一些混乱，也可以在 DATAFLD 属性中包含分层结构的层，这样可以看得更清楚一些。例如：

```
<TABLE ID="tblPublishers" DATASRC="#dsoPublishers">
  <TABLE DATASRC="#dsoPublishers" DATAFLD="Title">
    <TABLE DATASRC="#dsoPublishers" DATAFLD="Title.sale">
      <TABLE DATASRC="#dsoPublishers" DATAFLD="Employee">
```

这并没有改变什么，只是可以更清楚地看到了 sale 字段在分层结构中的位置。



图11-10 多层绑定显示的结果

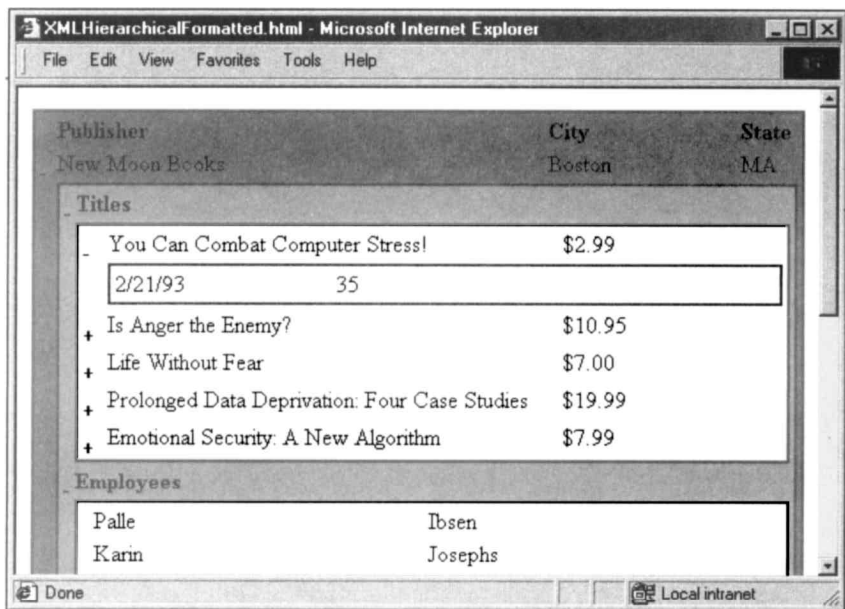


图11-11 多层绑定格式化后显示的结果

11.2.5 以XML数据格式保存记录集

通过使用 Recordset 对象的 Save 方法并指定存储格式为 adPersistXML，ADO 可以很容易地生成 XML 格式的记录集。这在 ADO 2.5 中并不是新内容，新颖之处在于能直接将分层结构的记录集转化成别的对象，如 Stream 和 DOM 对象。

由于我们不再局限于创建包含 XML 数据的文本文件，因此这一特性为数据的处理和传输提供了各种可能性。这又有什么好处？作为一个程序员可能会希望有：

- 一种简单的生成数据的方法。
- 一种能快速传输数据的方法。

创建文本文件是容易的，但比较慢。因为它涉及向磁盘中写文件，然后从磁盘中读文件并处理。应该尽可能地排除这种速度缓慢的操作，因为需要的是一种快速处理数据的方法。容易做的事未必是一件好事。

1. 将ADO记录集转换为XML文件

将ADO记录集转换为XML文件的最简单的方法是使用ADO以前版本中的Save方法：

```
rsAuthors.Save "C:\temp\authors.xml"
```

这里没有什么新内容，把它包括进来是为了便于和其他方法做比较。

2. 将ADO记录集转换为一个流

流只是在内存中的一块数据，没有进行任何方式的处理。ADO 2.5版本引入了新的Stream对象，它可以作为记录集保存的目标，例如：

```
Set rsAuthors = Server.CreateObject("ADODB.Recordset");  
Set stmAuthors = Server.CreateObject("ADODB.Stream");
```

```
rsAuthors.Open "authors", strConn
```

```
rsAuthors.Save stmAuthors, adPersistXML
```

现在获得了一个包含XML记录集的Stream对象，可以用Stream对象的方法和属性来处理数据。例如，可以使用ReadText方法将XML数据提取到字符串中：

```
strXMLAuthors = stmAuthors.ReadText
```

此时，strXMLAuthors包含完整的XML记录集，其中也包括模式。

3. 把ADO记录集转换为Response对象

其他关于流的观点更为直接。其中有一种把它定义为“流动或移动的连续体”。因此，我们不妨把数据流看成是从一个地方移动到另一个地方的数据流。现在，考虑一下从ASP页面向浏览器发送数据的过程，实际上这也是一种数据流。

要这样做可以使用Response对象的一个新特征，它提供对流的支持。因此，可以把数据输出到流的一个端，并让它达到另一端。我们使用的一端是Response对象，另一端是浏览器。因为在Response对象中集成了对流的支持，所以可以使用下面的代码：

```
rsAuthors.Save Response, adPersistXML
```

上面的代码直接将记录集以XML数据格式存入Response对象。然后Response对象负责向浏览器发送数据。这种方式非常好。

4. 创建XML数据岛

看起来创建数据岛似乎是一种理想的做法，但并不像把XML数据保存到Response对象那么简单。事实上这样做往往是非常困难的。

这主要是因为XML数据岛是一个带有两个属性的HTML标记：

- ID：一般的HTML ID属性，它唯一地确定元素。
- SRC：它确定XML源文件。

如果手工创建数据岛，可以使用下面的代码：

```
<XML ID="dsoData" SRC="authors.xml"></XML>
```

也可以不使用SRC属性，而直接把XML数据放在XML标记内，例如：

```
<XML ID="dsoData">
  <Authors>
    . . .
  </Authors>
</XML>
```

后一种方法似乎更理想。例如，假定有一个含有以下代码的ASP页面：

```
<%
Dim rsAuthors
Set rsAuthors = Server.CreateObject("ADODB.Recordset")

rsAuthors.Open "authors", strConn

rsAuthors.Save Response, adPersistXML

rsAuthors.Close
Set rsAuthors = Nothing
%>
```

在HTML页面中是以下内容：

```
<xml xmlns:s="uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882"
      xmlns:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"
      xmlns:rs="urn:schemas-microsoft-com:rowset"
      xmlns:z="#RowsetSchema">
  <s:Schema id="RowsetSchema">
    <s:ElementType name="row" content="eltOnly">
      <s:AttributeType name="au_id" rs:number="1" rs:writeunknown="true">
        . . .
      </s:AttributeType>
    </s:ElementType>
  </s:Schema>
  <rs:data>
    <z:row au_id="172-32-1176" au_lname="White" au_fname="Johnson"
           phone="408 496-7223" address="10932 Bigge Rd."
           city="Menlo Park" state="CA" zip="94025" contract="True" />
    . . .
  </rs:data>
</xml>
```

我们删去了一些数据以便能看得更清楚一些。首先应该承认，这看起来象一个数据岛，但又不具有ID属性，所以不能给它绑定任何字段。这不是一个太大的问题，因为我们可以给HTML元素增加属性。这可以在页面加载时在浏览器中或在客户端脚本中实现，或者在XML数据发送给客户之前在ASP代码中实现。这里不想介绍如何做，因为实际上这不能真正解决什么问题。

必须记住数据岛是一个HTML标记，这会带来两个问题：

- 在XML中的数据是无效的，因为有两个根元素：s:Schema和rs:data。
- 所有非XML的属性将被忽略。因此所有的名称空间信息（xmlns属性）将被忽略，致使XML包含的数据无效。

接下来将一个XML标记放入ASP代码中，例如：

```
<XML ID="dsoAuthors">
<%
Dim rsAuthors
Set rsAuthors = Server.CreateObject("ADODB.Recordset")
```

```
rsAuthors.Open "authors", strConn

rsAuthors.Save Response, adPersistXML

rsAuthors.Close
Set rsAuthors = Nothing
%>
</XML>
```

现在，在浏览器中显示出下面的内容：

```
<XML ID="dsoAuthors">
  <xml xmlns:s="uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882"
        xmlns:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"
        xmlns:rs="urn:schemas-microsoft-com:rowset"
        xmlns:z="#RowsetSchema">
    <s:Schema id="RowsetSchema">
      <s:ElementType name="row" content="eltOnly">
        <s:AttributeType name="au_id" rs:number="1" rs:writeunknown="true">
          . . .
        </s:AttributeType>
      </s:ElementType>
    </s:Schema>
    <rs:data>
      <z:row au_id="172-32-1176" au_lname="White" au_fname="Johnson"
            phone="408 496-7223" address="10932 Bigge Rd."
            city="Menlo Park" state="CA" zip="94025" contract="True" />
      . . .
    </rs:data>
  </xml>
</XML>
```

从逻辑上来说，上面的代码应该可以正常工作，但是，在一个数据岛内不能有名为 XML 的标记。

既然不能拥有一个名为 XML 的内部标记，为什么不能在 ASP 代码中重新命名它？也许可以将 XML 加载到 DOM 中，并且改变 nodeName 属性。但是这仍行不通，因为在 DOM 中，顶层元素的名称是不允许改变的。

那么，现在该怎么办？看上去直接将一个记录集保存为 Response 对象无法创建一个数据岛。这可能是因为 ADO 和 IE 发展的方式不同，这将来也许会得到改善。

然而，不要认为这不可能，因为有很多方式可创建一个数据岛，但并不只是涉及 Recordset 对象。其中一种方法就是使用 DOM。

5. 将 ADO 记录集转换为一个 DOM 对象

迄今为止，我们已经知道可以将记录集保存为一个文件，或者是 Response 对象。另外，还可以将一个记录集保存为 DOM 对象。在本章的前面部分，曾经提到 DOM 对象以及如何使用该对象处理 XML。因此，在 ASP 代码中可以这样做：

```
<%
Dim rsAuthors
Dim xmlDOM

Set rsAuthors = Server.CreateObject("ADODB.Recordset")
Set xmlDOM = Server.CreateObject("Microsoft.XMLDOM")

rsAuthors.Open "authors", strConn

rsAuthors.Save xmlDOM, adPersistXML
```

此时，DOM对象xmlDOM包含了记录集中的数据。这允许在将数据发送到浏览器之前，根据需要对数据进行处理。在本章后面，你会看到有关更多的例子。

借助DOM来创建数据岛

DOM比较有用的地方之一是可以创建数据岛。虽然不是一种十分优秀的解决方法（许多人更愿意称之为一个恐怖的黑客），但是处理速度很快，工作起来也很正常。我们可以用一个字符串从DOM中获取XML。例如：

```
rsAuthors.Save xmlDOM, adPersistXML
```

```
strXML = xmlDOM.xml
```

此时，变量strXML包含XML数据。那么，这对我们有什么帮助？前面已经说了，在数据岛内不能有一个名为<XML>的标签，也无法重新命名顶层元素。那么可以这样处理这个字符串：

```
strXML = "<xxx" & Mid(strXML, 5, Len(strXML) - 10) & "xxx>"
```

因为XML必须始于“<XML”，并且终于“XML>”，所以只需替换掉那部分字符串。至于用什么来替换它们无关紧要，只要不是XML就可以。

这样，现在的ASP代码应该是这样：

```
<XML ID="dsoAuthors">
<%
    Dim rsAuthors
    Dim xmlDOM

    Set rsAuthors = Server.CreateObject("ADODB.Recordset")
    Set xmlDOM = Server.CreateObject("Microsoft.XMLDOM")

    ' Open the recordset
    rsAuthors.Open "authors", strConn

    ' Save it to a DOM object
    rsAuthors.Save xmlDOM, adPersistXML

    ' Close the recordset
    rsAuthors.Close
    Set rsAuthors = Nothing

    ' Extract the XML as a string
    strXML = xmlDOM.xml

    ' Replace the xml tags with something else
    strXML = "<xxx" & Mid(strXML, 5, Len(strXML) - 10) & "xxx>"

    ' and send it back to the browser
    Response.Write strXML
%>
</XML>
```

当然，这并不是什么伟大的技术解决方案，但确实可以运行，而且工作情况良好。使我们能够轻而易举地从ASP创建一个数据岛，而且在ADO与IE之间仍存在障碍的时候，这还不失为一个好途径。

11.2.6 打开记录集

既然有许多方法可以将记录集保存为XML，那么反过来，自然也就有多种用记录集提取

XML数据的方法。然而需要注意的是，ADO记录集只能识别微软的XML数据模式，如果你所用的XML与这种模式不一致，就不能将它加载到一个 Recordset对象中。这也是另一个令人苦恼的由于ADO中的XML支持缺乏灵活性而引起的问题。

1. 从XML文件打开记录集

从一个XML文件加载记录集与保存这个记录集同样简单，例如：

```
rsAuthors.Open "authors.xml"
```

这将打开XML文件，再将其解析为记录集。所用的文件名必须是一个物理路径名。

2. 从Request对象打开记录集

前面已经提到流已被添加进 Response对象中，其实流也被添加到了 Request对象中。这就意味着可以执行如下操作：

```
rsData.Open Request, , , , adCmdFile
```

这样做的好处是可以与 IE 5.0中引入的XMLHTTP对象协同工作。XMLHTTP对象能利用HTTP协议向服务器发送XML数据或从服务器发送XML数据给客户。微软将一些HTTP功能封装进XMLHTTP对象是为了便于使用。在此不打算对它做进一步的阐述，如果需要更多的信息，请查阅IE 5的文档或MSDN。

这个对象使我们能够从客户应用程序将XML发送到ASP网页。下面的代码是一个实现该功能的客户页面的例子：

```
<XML ID="dsoData" SRC="authors.xml"></XML>

<BUTTON ID="cmdUpdateAll" TITLE="Save All Changes"
  ONCLICK="postXML()">Save</BUTTON>

<SCRIPT LANGUAGE="JScript">

function postXML()
{
  var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
  xmlhttp.open("POST", "XMLReadFromRequest.asp", false);
  xmlhttp.send(dsoData.XMLDocument);
}

</SCRIPT>
```

本例利用Open方法，并指定使用的HTTP方法以及要连接到的URL来初始化一个请求。最后的false表明不使用异步通信。然后用send方法来将XML数据岛的内容传送到指定的URL。

使用POST方法的结果是一个被传送到指定的URL的XML流。在上面的例子中，指定了一个ASP网页，并且XML以Request对象的形式出现在这个ASP网页中，因为这是一个普通的HTTP请求。这意味着可以使用下面的代码：

```
rsData.Open Request, , , , adCmdFile
```

3. 从DOM对象打开记录集

虽然能直接将记录集保存为XML的DOM对象，但却不能直接从DOM对象打开记录集。这对于那些本身支持流的对象来说显得比较奇怪，不过，对于这个问题有一个解决方法。只是这种方法稍有一点复杂，牵涉到一个Stream对象。

需要将DOM中的XML数据保存到一个Stream对象中，然后从这个Stream对象打开记录集。

结果与其他方法是相同的，只是需要额外的一个步骤。下面是一个例子：

```
Set rsData = Server.CreateObject("ADODB.Recordset")
Set stmData = Server.CreateObject("ADODB.Stream")
Set domXML = Server.CreateObject("MSXML.DOMDocument")

' Open a normal recordset
rsData.Open "authors", strConn

' Save it into an XML DOM object
rsData.Save domXML, adPersistXML
rsData.Close

' Now start the process of opening the recordset based on the DOM
' First open an empty stream
stmData.Open

' Write the XML into the stream
stmData.WriteText domXML.xml

' Mark the end of the stream
stmData.SetEOS

' Reposition back to the start of the stream
stmData.Position = 0

' Open the recordset based upon the stream
rsData.Open stmData
```

这种解决方法稍有点繁琐，但运行效果良好。同样，这个领域一定会随着时间的推移而日趋完善，当ADO实现了更好的方法后，上述方法可能就会没用了。

对上述代码加以修改，能得到一种复制记录集的简便方法。下面是一个 Visual Basic 例子，可用于一个组件中：

```
Function DuplicateRecordset(rsData As ADODB.Recordset) As ADODB.Recordset

    Dim stmData As ADODB.Stream
    Dim rstNewData As ADODB.Recordset

    Set stmData = New ADODB.Stream
    Set rstNewData = New ADODB.Recordset

    rsData.Save stmData

    rstNewData.Open stmData

    Set DuplicateRecordset = rstNewData

End Function
```

这段代码不很长，但假如需要记录集的一个拷贝，这还是很有用的。这不同于克隆，克隆将产生单个记录集和两个对它的引用，而这方法则创建了一个全新的记录集的拷贝。

11.3 用XSL来设计XML

XSL即可扩展样式语言(eXtensible Styling Language)，是一种用于转换XML数据的基于XML的语言。这种转换可以在XML的一种格式与另一种格式间进行，或者可以从XML转换到HTML格式，也可以从XML转换到任意类型的文本输出。

XSL产生的原因之一是因为XML纯粹用于数据。我们已经说过XML标记只用于识别数据，不像HTML标记那样可以用来指定数据格式。因此，需要使用某种方式来格式化XML以显示XML数据。

XSL由两部分构成，即转换语言和格式化语言，这里讨论的核心是转换部分。为什么这样说？这只是因为XSL现在还不是一种规范的标准，也就是说大多数浏览器对它的支持是不完全的。几个主要的浏览器基本上都不支持XSL的格式化部分，许多浏览器甚至连它的转换部分也不支持。

微软发布IE 5时，按照当时建立的标准加入了对XSL转换部分的支持。但自那以后由于标准一直在发展，微软决定不再更新其对XSL的支持，除非这种标准得到认可。这意味着IE 5并不支持XSL的格式化部分，但部分地支持其转换语言。

如果需要使用XSL的最新版本，可以使用符合最新标准的XSL解析器。W3C的网站(www.w3c.org)详细说明了XSL标准的现状，www.oasis-open.org/cover/xsl.html站点也在追踪着这个标准的发展方向，并且还有一些关于XSL的软件和文章。

XSL样式表

XSL是根据一套与XML中的元素或属性相匹配的规则设计的。这些规则被称为模板，在一个模板内你可以循环访问元素和属性，并可以应用其他模板执行其他类型的处理。凡是不属于XSL处理指令的组成部分的文本都被输出，这就是通过匹配元素与输出文本和元素的值来转换XML的过程。

下面来考虑一下本章前面用过的由ADO生成的关于authors的XML文件，这里介绍如何将这个XML文件转换成HTML表。但我们并不打算详细讨论XSL，一是因为该主题太大，二是因为这也超出了本书的范围。我们只是想说明使用XSL能做些什么。XSL的一个功能是将XML转换成HTML。

首先需要理解XSL是XML的一种形式，所以它也由一系列标记组成。一个样式表(styleshet)的顶层标记如下所示：

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

以上代码确定了样式表和应用于它的名称空间。既然是XML，那么样式表的最后就应该有结束标记：

```
</xsl:stylesheet>
```

在样式表内，标记采用模板的形式，匹配部分的XML文档。也就是说只有先匹配XML标记，然后才能输出标记的值，以及其他文本。下面分析一个简单的样式表，了解其工作方式：

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

```
<xsl:template match="/">
```

```
<HTML>  
<BODY>
```

```
<xsl:apply-templates select="//rs:data" />
```

```
</BODY>  
</HTML>
```

```
</xsl:template>
```

```
<xsl:template match="//rs:data">
```

```
<TABLE BORDER="1">
<xsl:for-each select="z:row">
  <TR>
    <xsl:for-each select="(@*)">
      <TD>
        <xsl:value-of/>
      </TD>
    </xsl:for-each>
  </TR>
</xsl:for-each>
</TABLE>
```

```
</xsl:template>
```

```
</xsl:stylesheet>
```

这个样式表将XML格式的ADO记录集转换为一个HTML表。下面进行详细地分析。首先是样式表的声明：

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

接下来是template命令，指明了要匹配的元素。在这里匹配的是根元素，由符号 "/"表示：

```
<xsl:template match="/">
```

此时，开始处理XML树。如果要把XML转换为HTML，需要输出开始一个HTML文档的HTML标记：

```
<HTML>
```

```
<BODY>
```

现在的任务是查找实际的数据。如果回想一下对XML格式的记录集的研究，会记得记录集的数据是存放在rs:data元素里的，因此用apply-template来寻找另一个模板。一旦应用该模板并且在此模板内的所有处理都完成后，处理过程就回到当前的模板：

```
<xsl:apply-templates select="//rs:data" />
```

在处理完内部模板后，就可以结束HTML文档了：

```
</BODY>
```

```
</HTML>
```

```
</xsl:template>
```

现在到了样式表的核心部分，在这里处理数据：

```
<xsl:template match="//rs:data">
```

所有的数据都保存在rs:data元素内，而每一行存放在z:row元素内。字段存放在行的属性中。首先输出<HTML>标记以开始创建表：

```
<TABLE BORDER="1">
```

需要处理每一行数据，因为这将作为HTML表的一行。为此，使用for-each语句，它与VBScript中的for...each循环语句是一样的。语句的select部分告诉XSL循环访问的是哪些元素——在这里是每个z:row元素：

```
<xsl:for-each select="z:row">
```

此时已有一个数据行，因此可以用下面的代码开始输出HTML行：

```
<TR>
```

现在需要处理属性。这些属性是与记录集的字段相对应的，同样使用for-each结构。但此时select不同，需要匹配每个属性，不管这些属性叫什么名字。实际上，如果这个程序用来处理由ADO生成的XML文档，那么就不可能知道任何行、元素和属性等的名称，因此，使用

“*”号作为通配符，它什么都可以匹配。比如为了匹配一个属性，必须使用“@”符号，所以“@*”就意味着可以匹配每一种属性：

```
<xsl:for-each select="@*">
```

接下来，开始创建HTML窗体元：

```
<TD>
```

在窗体元内，需要获得属性的值。指令 Value-of告诉XSL输出一个节点的值，由于并没有规定任何需要特殊输出的内容，所以便输出当前节点的值：

```
<xsl:value-of/>
```

下面是循环的结尾，关闭窗体元，关闭表的行，最后关闭这个表：

```
</TD>
</xsl:for-each>
</TR>
</xsl:for-each>
</TABLE>
```

```
</xsl:template>
```

```
</xsl:stylesheet>
```

虽然这看起来不是很简单，但是并没有脱离常规的编程规则。它看上去显得比较古怪，这是因为所有的操作都由XML指定的，并且其语法也都是围绕着匹配XML中的节点，不过像循环这类的操作已经很接近标准的编程习惯了。

1. 嵌入的样式

应用一个XML样式表的最简单的方法是在XML文件中指定样式表。这只需将下面这行加到XML文件的顶端：

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

对一个ASP程序员来说，这显得不是很灵活，因为这意味着物理地创建一个XML文件，然后编辑它并放入样式表的引用。

2. 动态样式

可以动态地应用样式表，而不是把它嵌入到XML中。可以在发送XML数据到浏览器之前，在ASP页面内动态地应用样式表。例如：

```
<%
Dim rsAuthors
Dim domXML
Dim domStyle

' Create the objects
Set rsAuthors = Server.CreateObject("ADODB.Recordset")
Set domXML = Server.CreateObject("Microsoft.XMLDOM")
Set domStyle = Server.CreateObject("Microsoft.XMLDOM")

' Open the recordset
rsAuthors.Open "authors", strConn

' Save the recordset to a DOM
rsAuthors.Save domXML, adPersistXML
rsAuthors.Close

' Open the stylesheet
domStyle.Load Server.MapPath("RecordsetToTable.xsl")
```

```

' Transform the XML
domXML.TransformNodeToObject domStyle, Response
:

' Tidy up
Set rsAuthors = Nothing
Set domXML = Nothing
Set domStyle = Nothing
%>

```

这与在本章和前一章中所见的代码没有多大差别。现在已经创建一个记录集并保存为 XML DOM 对象。不同之处在于接下来将样式表加载到另一个 XML DOM 对象中，并使用了 TransformNodeToObject 方法来应用这个样式表。

上述方法的优点是对客户只发送纯粹的 HTML，因此需要使用 XML 数据时，就无须考虑浏览器是否可以处理 XML，这一点非常有用。如果用户使用的浏览器支持 XM，也可以在客户端执行这种转换操作。

下面的例子使用了三个数据岛。第一个用来存储 XML 数据，后面二个用来存储样式表，记住它们是 XML，所以这样做是很合法的。一个样式表把 XML 转换成一个表，另一个把 XML 转换成一个列表。

```

<XML ID="dsoAuthors" SRC="authors.xml" HEIGHT="0" WIDTH="0"></XML>
<XML ID="styleTable" SRC="RecordsetToTable.xsl"></XML>
<XML ID="styleList" SRC="RecordsetToList.xsl"></XML>

```

然后创建两个按钮，以调用执行样式化的函数：

```

<BUTTON ID="cmdStyleOne" onClick="applyStyle('styleTable')">Recordset</BUTTON>
<BUTTON ID="cmdStyleTwo" onClick="applyStyle('styleList')">List</BUTTON>

```

接下来创建一个 DIV 元素，这是表或列表将要显示的地方：

```

<P>
<DIV ID="txtResult"></DIV>

```

最后，创建执行转换的函数。它使用数据源的 TransformNode 方法将选择的样式表应用到 XML 数据。其结果显示在 DIV 元素中：

```

<SCRIPT LANGUAGE=JScript>

function applyStyle(sStyle)
{
    var strStyle = document.all(sStyle).XMLDocument;
    txtResult.innerHTML = dsoAuthors.transformNode(strStyle);
}

</SCRIPT>

```

如果想在客户端完成数据转换，以上方法是非常有用的。尽管这个方法依赖于 IE，但确实减轻了 Web 服务器的负担，并且给用户带来了更多的灵活性，因为他们可以选择自己所看到的数据格式。

11.4 相关的领域

围绕着 XML 还有许多没有定义标准的领域，但仍然被各种各样的委员会使用着：

- XSL 仍然在发展，目前还不是一个标准。一旦它成为一个标准，将来的浏览器肯定会加入对 XSL 的支持。
- 可扩展的超文本标记语言 (Extensible HyperText Markup Language, XHTML) 应该是一种

HTML。XML与HTML最大的区别是严格程度不一样。HTML中的某些元素不一定需要一个关闭元素(比如, LI和BR元素), 并且某些属性也并不需要引号引起来。而 XML在这方面要求非常严格, 所有的元素都必须有关闭标记, 同时所有的属性都要用引号引起来。XHTML只是将这些字符串规则应用到 HTML中。这样做的目的, 首先是提高了标准化的程度, 另外也就意味着 XHTML文件可以很容易地被XML解析器处理, 因为它遵循了XML的语法规则。XHTML主要依据HTML 4.01版本, 4.01版本增加了许多功能。

- XLink和XPointer是允许链接XML文档的XML扩展。XLink与HTML中的<A>标记的功能基本相同。XPointer能够链接一个目标资源的单个或多个位置, 每一个位置可以是文档的任何部分。这扩展了现有的链接模式的功能, 同时也提供了一个更灵活的、有效的解决方案。
- 格式化对象是XSL样式规范的第二个组成部分, 它引入了一套标准标记用来格式化对象。它们提供了一个与显示无关的格式化描述, 这与 HTML中描述和布局相结合的元素正好相反。
- BizTalk是微软设计的用来统一使用XML的框架。它定义了一套标准的专门用于解决一般的业务逻辑方案的XML标记。如需要更多的有关 BizTalk的信息, 请查询网址 <http://www.biztalk.org>。

11.5 用于SQL Server的XML技术预览

就在本书送去印刷之前, 微软发布了一个关于如何从 SQL Server中检索XML数据的技术预览, 称为用于SQL Server的XML技术预览, 它采用一种位于 Web服务器和SQL Server之间的ISAPI DLL的形式。这个DLL可以将SQL查询转换为XML, 也能将XML复原回SQL数据。这个DLL的灵活性相当高, 功能也极为强大。

该技术的一些特征包括以下几个方面:

- 以行格式(由ADO使用)和元素格式提取XML。
- 可以包含一个带有XML数据的DTD。
- 可以包含一个带有XML数据的模式。
- 可以通过XML插入、删除和更新数据。

既然还只是一个预览, 这里不打算做详细的分析, 但这确实很令人兴奋, 这给未来 XML的发展带来了极大的希望, 同时也使我们能更容易地使用 XML数据。只需要使用这个工具, 就可以在浏览器中看到图 11-12所示的结果。

它依赖于一个由 Technology Preview注册工具建立的名为 Northwind的虚拟目录。它创建这个虚拟目录, 确保ISAPI DLL处理所有进入该目录的请求。它能够处理 SQL命令, 并将其发送给SQL Server, 然后返回格式化为XML的SQL数据。如果不喜欢这种XML格式, 而希望使用一种更传统的基于元素的布局, 那么只需在查询的最后加上“elements”, 显示结果如图 11-13所示。

正如在本章前面所见的那样, 我们已经尝试了在 ASP中使用ADO来创建数据岛, 但是却碰到了许多困难。微软开发的这个新工具使我们可以做到这一点:

```
<XML ID="dsoData"
  SRC="http://Kanga/pubs?sql=SELECT * FROM authors FOR xml AUTO">
</XML>
```

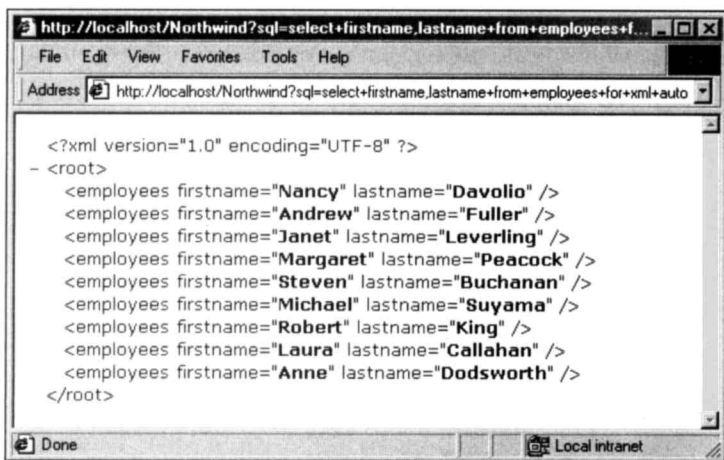


图11-12 在浏览器中显示的XML数据结果1

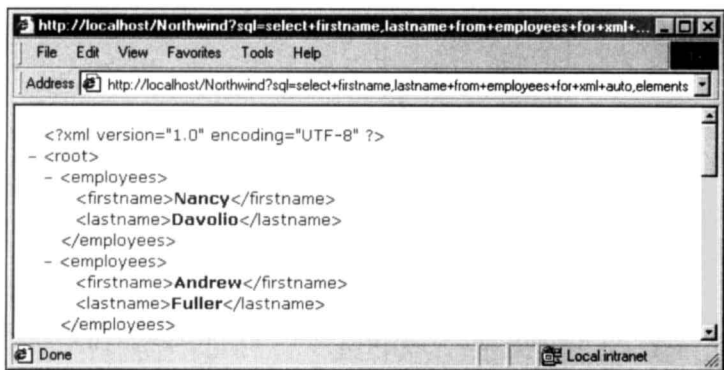


图11-13 在浏览器中显示的XML数据结果2

这里不打算详细叙述这个问题。这有两个理由：首先，这是一个很新的内容；其次，相关内容可能会发生变化。然而，从某些未来的微软技术中我们看到了所期望的内容。

如果想下载关于该技术预览的拷贝，可以从 MSDN的站点下载，其网址是：msdn.microsoft.com/workshop/xml/articles/xmlsql/default.asp。该文件的大小不超过 250KB，因此不用很长时间就可以得到这个文件。

11.6 小结

本章以很快的速度研究了实际上非常大的一个领域的内容，而且该领域正在日新月异地发展。XML为数据交互提供了巨大的潜力，XML可能将出现在许多新的领域里。

毫无疑问你将越来越多地看到 XML。因此，本章主要讨论了 XML的一些基本问题，以及对于我们来说XML意味着什么。本章主要讨论了以下的内容：

- XML是什么及它与数据之间如何联系。
- ADO与XML之间如何交互。
- 在ADO和XML之间如何使用Stream对象传输记录集。
- 如何使用XSL将XML数据转换成HTML。

我们知道一些微软技术相互之间结合得不是太好。这也是可以预料的，因为这些技术是

由不同的人员花费不同的时间开发的。但这给我们的编程带来了一定的困难。本章的任务之一就是向读者展示应用相应的技术能够做那些工作，这可以使读者花费较少的时间了解这方面的内容。虽然听起来让人有些沮丧，但是微软在其各种技术之间的结合方面不断进行着改进。相信不久的将来，其各种产品会以统一的方式工作，而不是同时提供多种解决方案。

我们已经花费了一些时间对一些领域的内容进行简单的探讨。这种探讨之所以肤浅的，部分原因是这实在是一个很大的题目，另外的原因是该领域还在发展之中。但是也不要产生误解，XML还是有必要掌握的。