第 5 章

XML 基础

学习要点

(1) XML 文档的基本结构。
(2) 用 CSS 在浏览器中控制 XML 文档显示的方法。
(3) 用 XSL 控制 XML 文档在浏览器中显示的方法。
(4) 进行 XML DOM 程序设计的方法。
(5) XML 文档和数据库系统之间的数据交换。
(6) XML 技术的应用和发展趋势。

于 1998 年问世的 XML 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 难以使用,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。

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。


视频讲解

5.1 XML 文档

5.1.1 XML 的概念

今天,XML 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，必须定义自己的标记。
- XML 使用文档类型定义(DTD)或者 Schema(模式)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 的定义来看，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，必须转换为 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、publishdate 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 中，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，文档中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 标记的规则

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、数字以及下画线。
- 名字不能以下画线开头,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：不能转换为 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,所有的信息都按某种关系排列。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,每一部分都紧密联系,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 meta 数据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 meta 数据处理方法,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,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如下。

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,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 转换为 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(XML 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 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 无法满足用户需求的情况下，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这方面的功能。

2. HTML 与 XML 的对比

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

表 5-1 HTML 与 XML 的对比

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、配对和遵循树形结构
可读性及可维护性	难以阅读、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 技术的过程中，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。

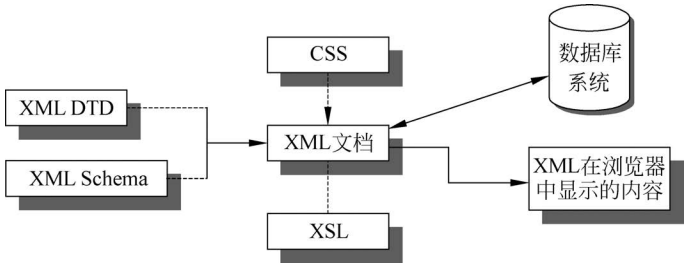


图 5-1 XML 相关技术之间的关系

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,用于说明 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 分别用于说明 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。

5.1.4 XML 文档术语以及基本结构

1. XML 文档术语

(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,标识需要自己来创建。一般来说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)中书写属性。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。例如:

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

从以上代码中可以看到,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、属性以及元素之间的关系的。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 DTD。

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>
```

XML 文档结构由三部分组成。

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

文档类型定义部分，一般形式为<! 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

XML 文档内容有以下几种结构。

(1) 声明根元素。

每个有效的 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"] …[属性名="属性值"]>
  elementValue
</elementName>
```

其中,元素名和属性名必须以字母或下画线开始,并且只能包含字母、数字、下画线、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,因此可以缩写成:

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

可以将元素的属性名转换为元素名,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 中名称空间(namespace)的严格定义是:名称空间是用 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 (统一资源标识符)的缩写。在没有 namespace 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 name)。XML 中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]]>
```

其中，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形方式打开，如图 5-2 所示。

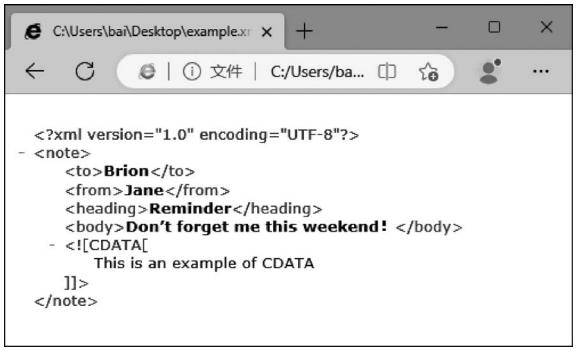


图 5-2　一个简单的 XML 文档在浏览器中的显示

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 或模式文件进行比较，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。

5.2　用 CSS 控制 XML 文档在浏览器中的显示


视频讲解

5.2.1　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 文档显示的四种 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,address# 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 }                    /* 可以被 XML 文档中的任何元素使用 */
```

或者

```
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 }                    /* 可以被 XML 文档中的任何元素使用 */
```

或者

```
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。*/
```

5.2.2 CSS 样式和 XML 文档联系

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>
```

第二行不可缺少,告诉浏览器要用 CSS 来显示 XML 文档。第三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	
background-repeat	背景图重复	repeat	图像在水平和垂直两个方向上重复
		rfepeat-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	文本画线	underline/overline	下画线/上画线
		line-through/none	中画线/无画线
margin	页边距	同 font-size 取值	上、下、左、右页边距

续表

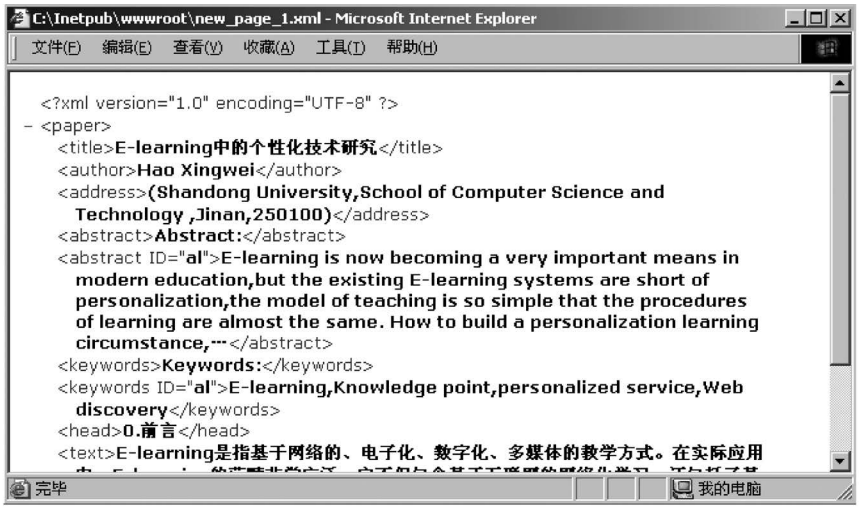
属 性 名	含 义	取 值	说 明
margin-top	上边距	同 font-size 取值	也用作段落间距和左右缩进
margin-bottom margin-Left margin-right	下边距 左边距 右边距	同 font-size 取值	
border-style	边框线样式	dotted, dashed, solid, double, groove, ridge, inset, outset, none	边框线
border-weight	边框线粗细	同 font-size 取值	
border-color	边框线颜色	同 font-size 取值	
text-indent	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。将下面的代码保存为文件 ex_5_2.xml。

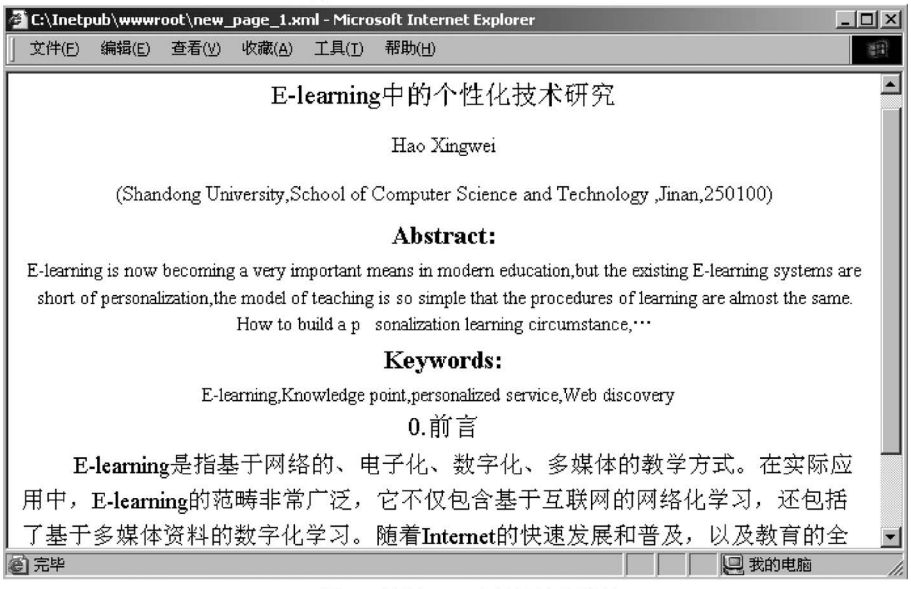
```
<?xml  version="1.0" encoding="UTF-8"?>
<?xml-stylesheet  type="text/css"?>
<paper xmlns:html="http://www.w3.org/1999/xhtml">
  <!-- 样式单定义 -->
  <html:style>
    paper { display:block;font-size:20px;line-height:160%;text-align:
center}
    author{ display:block; font-size:16px;margin-top:5px;margin-bottom:5px}
    address{ display:block; font-size:16px; text-align:center}
    abstract{
    display:block;  font-size:20px; font-weight:bold; font_style:italic;
text-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 personalization,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="a1">
    E-learning,Knowledge point,personalized 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>
```

上述代码在浏览器中的运行结果如图 5-3 所示。



(a) 没有CSS样式定义的XML文档



(b) 用CSS控制XML文档的输出效果

图 5-3　例 5-2 的运行效果

【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>
```

图 5-4 XML 使用 CSS

运行该 XML 文档，其显示结果如图 5-4 所示。

【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 5-3 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 浏览器显示的结果如图 5-5 所示。

图 5-5 使用 CSS 将 XML 文档按表格输出

5.3 用 XSL 控制 XML 文档在浏览器中的显示


视频讲解

5.3.1 XSL 概述

可扩展样式表语言(eXtensible Stylesheet Language，XSL)是由 W3C 于 1999 年 11 月制定的。XSL 自提出以来争议颇多，前后经过了几番大的修改。2017 年 6 月，W3C 发布了 XSLT 的 3.0 版本。2021 年 3 月，W3C 发布了 XSLT 2.0 版本的第二版。

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 文档转换为一个新的文档(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),这个过程称为树转换;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,这个过程称为格式化。格式化符号集又称为格式对象(Formatted Object,FO)。

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 Transformation),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进行转换的方法,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。

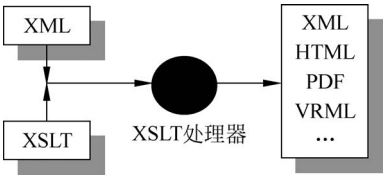


图 5-6 XSLT 转换过程

使用 XSL 定义 XML 文档显示方式的基本思想: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 文件 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 浏览器打开 XML 文件，显示结果如图 5-7 所示。

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tylor
Greatest Hits	Dolly Parton

图 5-7　使用 XSL 显示 XML 文档

5.3.2　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。

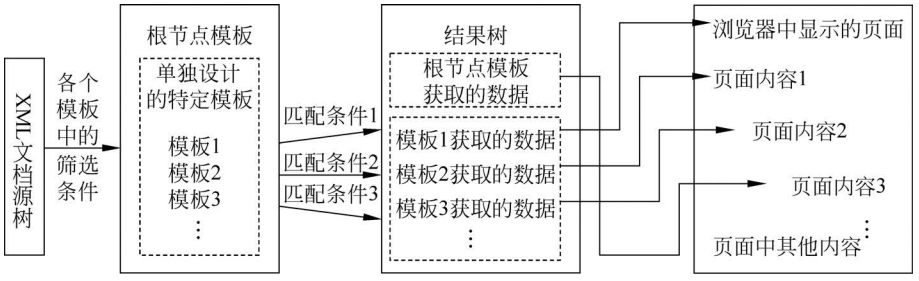


图 5-8　用模板格式化 XML 文档示意

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 属性调用模板

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 属性调用模板

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，不至于出现互相干扰、混杂不清的情况。这种从上至下、逐层细化的设计方法，极大地减少了工作的复杂程度，也大幅减少了差错的产生，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"/>?b<!-- 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>
```

上面的例子中,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。

5.3.3 XSL 选择和测试元素

XSL 选择元素的作用: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: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"的值大于或等于 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。

5.3.4 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
>=	大于或等于。在 XSL 中，要用 > =表示	not	逻辑非
<	小于。在 XSL 中，要用 < 表示	=	相等
<=	小于或等于。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 < 60]">。

说明：[ ]表示选择条件，仅循环处理成绩小于 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 文档内容如下：

```
<?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><head><title>XML 技术</title></head>
5          <body bgcolor="#00CC66">
6            <table border="1" cellspacing="0">
7              <th>姓名</th><th>性别</th>
8              <xsl:for-each select="/document/resume[grade &lt; 60]">
9                <tr><td><xsl:value-of select="name"/></td>
10                   <td><xsl:value- of select="sex"/></td>
11               </tr>
12             </xsl:for-each>
13           </table>第个人的生日是<xsl:value-of select="/ * / * /birthday"/>
14         </body>
15       </html>
16     </xsl:template>
17   </xsl:stylesheet>
```

5.3.5 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 函数与 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。例如：

```
<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。例如：

```
<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 的有几个。

3. number 函数

作用：将属性值中的文本转换为数值。例如：

```
<p>The number is: <xsl:value-of select="number(book/price)"/></p>
```

说明：显示书的价格。

4. substring 函数

语法：

```
substring(value,start,length)
```

作用：截取字符串。例如：

```
<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。例如：

```
<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隔符号。例如：

```
concat('XPath ','is ','FUN! ')
string-join(('We', 'are', 'having', 'fun! '), ' ')  //用空格连接字符串
```

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。例如：

```
<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 XML DOM 编程基础

视频讲解

5.4.1 XML DOM 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。无论是在浏览器中还是在浏览器外,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换为一棵对象节点树——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。

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;每个 XML 属性是一个属性节点;注释属于注释节点。

【例 5-9】 DOM 把 XML 文件当作一种树形结构。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,最上端的节点被称为根;每个节点,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_9.xml 文档,其 DOM 节点树如图 5-9 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>
```

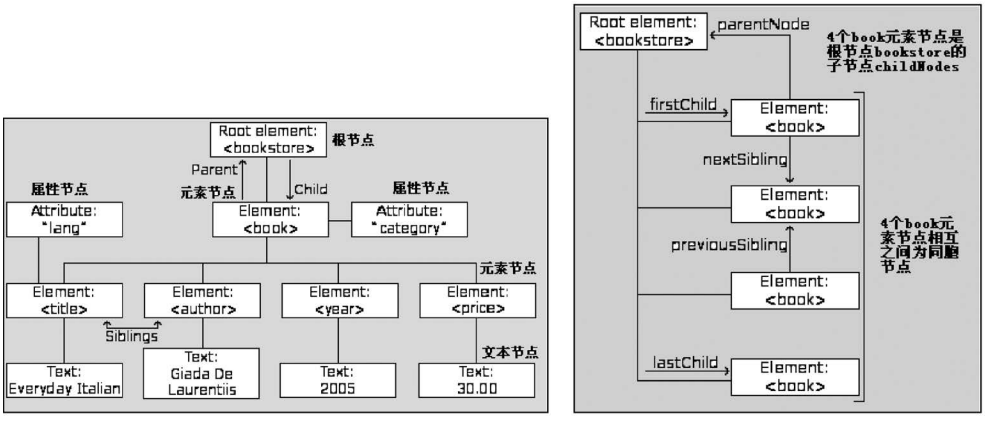


图 5-9　XML 文档的 DOM 节点树

5.4.2　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。

创建 Document 对象的语法格式为：

JavaScript/JScript：

```
//MSXML 2.0 版本，支持 DTD 的处理
var doc=new ActiveXObject("Microsoft.XMLDOM")
//MSXML 3.0 版本，支持 DTD、XSD 的处理
var doc=new ActiveXObject("Msxml2.DOMDocument.3.0")
```

VBScript：

```
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 5-4 和表 5-5 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 5-4　Document 对象的主要属性

属　性	描　述	IE	F	O	W3C
async	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 5-5　Document 对象的主要方法

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 5-10 所示。

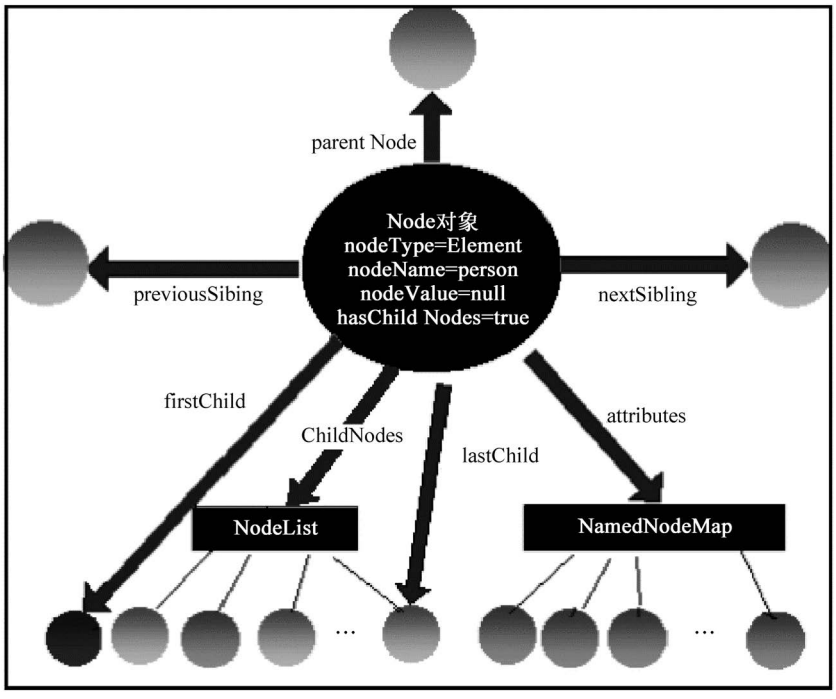


图 5-10　典型的 Node 对象及其相互关系

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 5-6 所示。

表 5-6　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

续表

节点对象类型	描　　述	子　对　象
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 中声明的符号对象	无

注意,尽管所有的对象都继承了用于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都包含子类或是父类。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 5-7 和表 5-8 所示。

表 5-7　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 URI	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	返回下一个同胞节点	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 5-8　节点对象的主要方法

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

续表

方　法	描　述	IE	F	O	W3C
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 5-9 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 5-9　Element 对象的部分属性和方法

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 5-10 列出了 Attr 对象除了继承 Node 对象的属性外新增的属性。

表 5-10 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如表 5-11 所示。

表 5-11 Text 对象的部分属性和方法

属 性		描 述	IE	F	O	W3C
属性	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	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。其主要目的是包含诸如 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]]>,它可标识 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 5-12 所示。

表 5-12 CDATASection 对象的部分属性和方法

属性		描述	IE	F	O	W3C
属性	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	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 5-13 所示。

表 5-13 DocumentType 对象的属性

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、引起错误的行为等。parseError 对象不属于 W3C DOM 标准,其属性如表 5-14 所示。

表 5-14 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
uri	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。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,则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 5-10 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 5-15 所示。

表 5-15 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

5.4.3 XML DOM 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 5-10】 建立一个 JavaScript 文件 ex_5_10.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 5-9 中的 ex_5_9.xml 文档,下列代码用 XML DOM 实现了对它的各种操作,可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>
```

5.5 XML 与数据库


视频讲解

5.5.1 SQL Server 对 XML 的支持

1. 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为 XML 了。

(2) 简单的 HTTP URL 请求。

例如，实现此功能需要在“开始”|“程序”|Microsoft SQL Server|“在 IIS 中配置 SQLXML”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 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。假如设置了一个叫作 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,如果你在一个叫作 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,可以将它们转换为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,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。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 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,SQLXML 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 提供了一个很好的方法,使用户可以将大量的数据写到 SQL Server 中;否则用户必须提取数据,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。

2. SQL Server 2005 对 XML 的支持

在 SQL Server 2005 中,FOR XML 功能新增了根元素和元素名称的新选项,使用 FOR XML 调用以便可以建立拥有复杂层次关系的能力的、使得你可以定义 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 中现有的 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。例如:

```
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
```

3. 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以下几方面。

(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>
```

5.5.2 XML 与数据库的互操作过程

首先应该明确 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，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 5-11 所示。

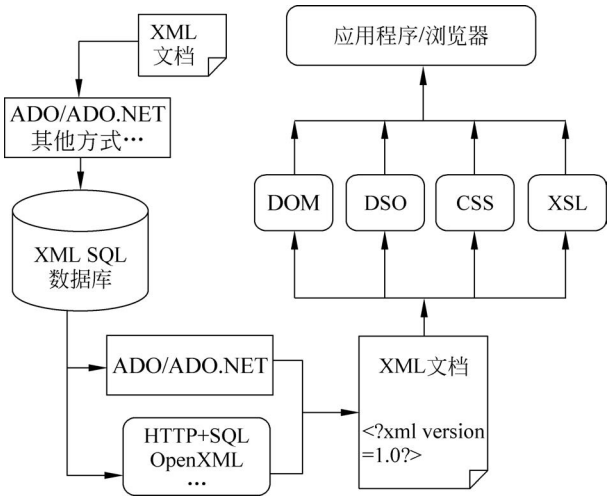


图 5-11　XML 与数据库的互操作过程

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 的工作方式有两种: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 源节点转换为 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。

思考练习题

1. 简述什么是 XML 及 XML 与 HTML 的区别。
2. CSS 与 XSL 有什么区别?
3. 对本章的所有例子进行上机验证。