返回总目录

目　　录

第三部分　XML 实践

第八章　创建 XML 文档

本章我们将采用传统的介绍一种新语言的方法，为读者演示写出一段能显示出 Hello XML 的程序。虽然 XML 是一种标记语言，而非一种程序语言，但程序语言的基本规则和原理还是可以应用的。从一个完整、可用、容易了解的范例开始学习，会比从一些基本的片段但又没什么实用价值的东西开始学习更容易些。又假如在使用基本的工具碰到了问题时，这些问题会更容易地在这段简单的文档中加以解决。

本章包括以下内容：

- 建立一个简单的 XML 文档
- 文档的组成要素
- XML 文档的格式良好性与有效性
- 一个 XML 文档实例

8.1　建立一个简单的 XML 文档

8.1.1　Hello XML

在这一章节我们将学会如何写出一份真正的 XML 文档（虽然我们前面在介绍 XML 语法时已经有所涉及）。让我们从一个最简单的 XML 文档开始，这份文档如下：

```
<?xml version=”1.0” standalone=”yes”?>
<foo>
Hello XML
</foo>
```

这份文档不是非常复杂，但却是非常好的一份 XML 文档，更准确地说，这是一份格式正确（well-formed）的文档。这份文档可用任何常用的文字编辑器做出，比如 Notepad、BBEdit 或 EMACS 都可以。

8.1.2　保存 XML 文档

输入完以上的程序代码后，将其存成文档，可以叫做 hello.xml、MyFirstDocument.xml 或其他的名字，但其后都有相同的标准扩展名.xml。需要注意的是要将其存成普通的文字格式，而非一些像 WordPerfect 或 Microsoft Word 文字处理器的格式。

8.1.3　将 XML 文档载入浏览器

现在我们已经建好第一份 XML 文档，让我们看看它的结果。这份文档可以用任何支持 XML 的浏览器打开，比如 Internet Exploerer 5.0，我们所见到的结果会因不同的浏览器而不同。在这个例子中我们可以见到 XML 文档内的程序代码被显示出来，文字格式非常整齐，在不同文法处也有不同的颜色标出，在浏览器中显示如图 8-1 所示。然而这种显示

效果非常单调（如同我们在第五章中所见到的一样），产生这个问题的原因，是因为浏览器还不知道 FOO 元素到底是什么，我们必须利用样式表告诉浏览器每一项元素该做什么，我们将会大略进行介绍。但首先我们再更进一步来看看这份 XML 文档。

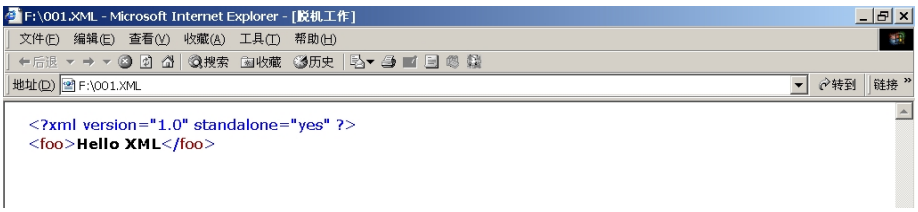

图 8-1

8.1.4 浏览简单的 XML 文档

为了更清楚地了解每一行程序的意义，我们来看上面的 XML 文档。第一行为 XML 文档的声明：<?xml version="1.0" standalone="yes"?>

这是一个 XML 标注的范例，标注从“<?”开始到“?>”介绍，“<?”后的第一个字也就是范例中的 xml 是标注的名称。XML 声明还有 version 和 standalone 两种属性，属性包括属性名与属性值，两者被一个等号分开，属性名称在等号左边，其值在等号右边，并且要用双引号标出。

每一份 XML 文档开头都要有 XML 的声明，并要特别标明所使用的版本。以上的范例中 version 属性正说明这份文档适用于 XML1.0 规范。XML 另外一个属性的声明 standalone 告诉我们这份文档在文档中是否完整，或是否还需要载入其它的文档。在以上的范例或是我们后面的例子中，所有的文档本身就是完整而不需要其他文档的。所以 standalone 属性值为“yes”。现在我们再来看看后三行：

```
<foo>
Hello XML!
</foo>
```

整体看这三行所形成的是一个 foo 元素，分开看的话“<foo>”为起始标记“</foo>”为结束标记，而“Hello XML!”是 foo 元素的内容。读者也许会问 foo 标记代表什么意思，而这个答案是我们想要它是什么它就是什么。除了上百个已定义好的标记，XML 可以让用户建立一个他所需要的标记，而<foo>标记的意义是根据用户的定义而来的。

8.1.5 建立 XML 标记的意义

标记可以有三种意义：结构、语言和样式。结构将文档分成树枝状的元素，语言将个别的元素与文档本身在真实世界的意义相互连结，样式可以定义出这些标记如何被表示出。

结构只是表达文档的组成，而并不关心不同的标记和元素。语言指的是撰写者、读者的心中或程序在产生或阅读文档时所了解的意义，例如一个了解 HTML 而不懂 XML 的网络浏览器，它会将段落文章的意义定义成标记“ ”和“ ”而不是“<GREETING>”和“</GREETING>”，然而对于一个了解英文的人来说“<GREETING></GREETING>”会比“ ”更容易了解。标记的第三种意义是样式，样式指的是如何在电脑屏幕或其

他输出装置上表现出标记的内容，样式可以定义出元素是字体、颜色、大小等等任何用户想要表现形式。对于计算机来说，样式会比语言上的意义更容易了解，而在 XML 内样式的意义可在样式表中定义出。

8.1.6 编写应用到 XML 文档上的样式表

XML 可以让用户建立任何想要的标记。当然，虽然用户可以完全自由地建立标记，但是浏览器却无法处理用户的标记，因此也无法去展示它们，因此，用户还需要撰写一份样式表来应用到 XML 上，告诉浏览器如何去显示特定的标记。如同标记一般，样式表也可以应用到不同的人或不同的文档上，而且用户建立的样式表也可以和别的样式表整合在一起。

我们可以使用的样式表语言不只一种。第一种是我们所用到的是层叠样式表（Cascading Style Sheet），简称为 CSS。CSS 的优点是建立在 W3C 的标准上，而且类似大家所熟悉的 HTML，以及支持 XML 的网络浏览器也都支持了 CSS。另外一种样式表为扩展样式表（Extensible Style Language），也就是 XSL，这是目前最强大的样式表语言，也是被设计来只能应用在 XML 上。然而，与 CSS 相比，XSL 太过复杂，并且也还没有完整的支持。在这里编写的应用样式表如下，并保存为 hello.css。

```
<STYLE TYPE="text/css">
<!--
foo{display:block ; font-size:40pt; font-weight:bold;    }
-->
</STYLE>
```

8.1.7 应用样式表到 XML 文档上

当我们写完一份 XML 文档和它的样式表，还必须要告诉浏览器把样式表应用到文档上。现在我们使用的方法是用一个标注来引用我们想要的样式表到相应的文档上。这个标注是“<?xml-stylesheet?>”，它有两个属性“type”和“href”。其中“type”属性定义样式表所用的语言，而“href”属性定义它取得的地址，或是相对路径。加入样式表的代码如下：

```
<?xml version="1.0" standalone="yes"?>
<?xml-stylesheet type="text/css" href="hello.css"?>
<foo>
Hello XML
</foo>
```

在浏览器中刷新代码后的显示如图 8-2 所示。

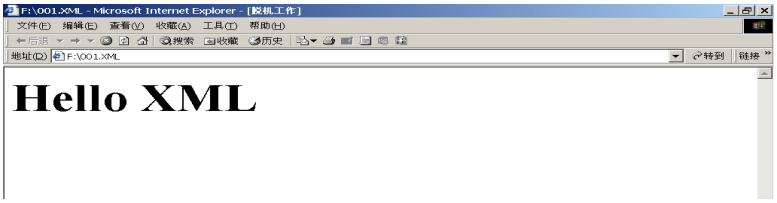

图 8-2

8.2 文档的组成要素

8.2.1 属性

在上面，所有的数据都以标记的名称或元素的内容来分类。这样是一个直接而且容易了解的方法，但却不是唯一的一种。XML 就跟 HTML 一样，各个元素都可以有属性。所谓的属性包含一对属性名称和属性值。名称和属性值都是字符串，而且在同一个元素中不能同时有二个名称相同的属性。以“<img>”标记来举个例子：

<IMG SRC= cup.gif WIDTH=89 HEIGHT=67 ALT=”Cup of coffee”>

上面 HTML 的例子中共有四个属性：SRC（值为 cup.gif），WIDTH（值为 89），HIGHT（值为 67），ALT（值为 Cup of coffee）。然而，在 XML 里，属性值必须以引号括起来，而且有起始标记就必须有结束标记。在 XML 中要如此表示：

<IMG SRC=”cup.gif” WIDTH=”89” HEIGHT=”67” ALT=”Cup of coffee”>

注意，XML 和 HTML 另一个不同点是 XML 并没有指定特定的意义给 IMG 标记和它的属性，而且并不保证 XML 浏览器会将 IMG 标记当作是将 cup.gif 这个图片载入并显示出来的指令。我们没有明确的规则来限定什么时候用子元素、什么时候用属性。一般来说，用户会用较符合自己应用的那一个。在使用的时候，用户会觉得属性会比子元素来的好用，或恰好相反。有一个推荐的做法是数据本身存在元素中，而描述数据的内容则存在属性中。当有疑问时，就把那些内容放在元素中。属性是一个存放 ID 号、URL、参考和一些并不直接与读者相关的内容的好地方。然而，有几条基本的原则需要注意：

（1）属性不能较好地表现结构。

（2）元素可以让用户加入元数据（形容数据的内容)。

（3）并不是所有人都会一致认为什么是元数据。

（4）元素在面对将来的改变时显得较有弹性。

8.2.2 结构化的元数据

有一个需要记住的原则是，元素能有子架构而属性不能。这使得元素更有弹性，而且可以更方便地让用户将元数据写成子元素。举例来说，假设用户正在写一个报告，而想要引用一段事实数据，那么也许就会像下面这样：

<FACT SOURSE=”The description of the fact ”>

The fact itselef

</FACT>

相当容易理解“The descrition of the fact ”是元数据。它并不是所要描述的事实本身，而是用来描述所引用的事实。然而，SOURCE 属性包含了许多暗示性的子架构。如果我们用元素代替属性的话，我们可以很方便地加入一些额外的内容，比如作者的 e-mail 地址、电子文档的地址、某一期期刊的标题或主题、或任何其它看起来重要的事。另一个一般的例子是日期。有关学术文档常用的元数据就是这份文档被收到的日期，这对一些发现和发明的发布优先权是很重要的。像下面在 ARTICLE 标记中加入一个 DATE 属性非常简单：

<article date=”06/28/1998”>

```
yours code here
</article>
```

然而，DATE 这个属性有一个由“/”来表明的子结构。要从属性值中存取这个结构远比从 DATE 元素中读取子元素困难得多，如下：

```
<date>
    <year>1998</year>
    <month>06</month>
    <day>28</day>
</date>
```

举例来说，对于 CSS 和 XSL，要让日期和月份看不见而只有年份显示出来是一件简单的事。以 CSS 来举个例子：

```
year{display:inline}
month{display:none}
day{display:none}
```

假如 DATA 是储存成一个属性，那么要读取它的一部分就没那么容易。必须另外用一些程序语言（如 ECMAScript 或 Java)写一个程序来分析数据格式。相对之下，使用 XML 工具和子元素比较容易。

此外，属性的语法含义并不是特别清晰。“10/11/1999”代表什么呢？尤其，它是指十月十一号还是指十一月十号？来自不同国家的读者可能对它有不同的理解。即使用户的解析器可以解读，也不能保证使用者会正确地输入。与此相反，XML 的子元素就不会产生此问题。

最后，使用 DATE 子元素来代替属性能让我们存放一个以上有关一个元素的日期。举例来说，学术文档常常会交回作者以便日后更新版本。这样一来，我们就必需注明新的版本的文档是什么时候收到的。

8.2.3 可扩展样式语言 XSL

在我们看 XML 的原始代码文档时，是可以看到属性的。然而，一旦将 CSS 样式表加上以后，就不能看到属性了。因为 CSS 只适用于元素的内容而不是属性，所以看到的将是一个空白的文档。假如我们使用的是 CSS 的话，那么只要是我们想显示出来的，都必须把它加入元素的内容中而不能放在属性里。然而，有另一种样式表语言可以让我们存取并显示属性中的数据，这个语言就是可扩展样式语言（eXtensible Style Language，XSL），而且 IE5.0 也支持 XSL 至少是部分支持。XSL 可以分成两部分：变换部分和格式化部分。

变化部分可以让用户定义标记。用户可以用标准的 HTML 标记来代替他的 XML 标记；也可以用 HTML 标记加上 CSS 属性来代替。用户还可以做很多工作，比如将用户文档中的元素重新排序，或是加入一些 XML 文档中从来没有出现过的内容。而格式化部分则提供了一个强力的功能，能将用户的文档当成一页页的网页。XSL 的格式化让用户可以设定网页的外观和版面配置，像是多专栏（多个文字区块））、文绕图、行隔控制复合字型属性等等。它是设计来当作有力的版面配置工具，包括从来源文档到网页和纸面印刷。举例来说，XSL 的格式化可以让一份包含播出时间和广告的 XML 文档变成一份本地报纸的电视节目列表，而且可以同时有印刷本和线上版本。但是，IE5.0 和一些其它的工具还未支持

XSL 格式。因此，接下来的章节会将重点放在 XSL 的转换上面。

1. XSL 样式表模板

一个 XSL 的样式表包含有 XML 文档所要灌入的模板。而一个模板，大概是下面这种形式：

```
<HTML>
<HEAD>
    <TITLE>
XSL INSTRUCTION TO GET THE TITLE
    </TITLE>
</HEAD>
<H1> XSL INSTRUCTION TO GET THE TITLE </H1>
<BODY>
XSL INSTRUCTION TO GET THE STATISTICS
<BODY>
</HTML>
```

用户可以将这个模板应用在不同的数据组中，这可能会使我们联想到 HTML 的一些服务器端嵌入。事实上，它们非常相似。然而，来源于 XML 文档与 XSL 样式表的格式转移是在客户端进行的，而不是在服务器端。再者，转移出来的文档不一定要是 HTML 格式。它可以是任何格式正确的 XML 文档。XSL 的指令可以取回任何储存在 XML 元素中的数据。这些数据包括了元素内容、元素名称、还有对我们的举例最为重要的一个——元素属性。这些指令模组会以元素的名称、值、属性名称、属性值，以及元素在 XML 文档的树状结构中的相对及绝对位置来判断、选择元素。只要数据被抽取出来后，我们就可以移动、拷贝、以各种方式来处理。但是在这里的简短介绍中，我们不打算说完所有 XML 格式转移能做的事。然而，稍后我们可以学到如何用 XSL 来立即写出一些效果惊人并且可以在网络上观看的文档。

8.2.4 CSS 与 XSL 之间的抉择

CSS 与 XSL 在某种程度上是重叠的，但是 XSL 的功能一定比 CSS 更强。然而，XSL 的功能强度却与它的复杂度成正比。这一章中只是浅浅的接触到我们能利用 XSL 做什么。XSL 实际上要复杂得多，而且比 CSS 更难学习。那么我们会问：什么时候我们该用 XSL，而什么时候该用 CSS 呢？CSS 的支持度比 XSL 高得多。一部分 CSS 的第一层级（CSS LEVEL1）的 HTML 标记已经被 Netscape 4 和 Internet Explorer 4 支持（虽然显示出来的效果不太一样）。再者，CSS Level1 的大部分和 CSS Level2 的一部分都很有可能在 Internet Explorer5.0 与 Mozilla5.0 中得到良好的支持（不管是在 HTML 或是 XML 的应用上）。因此，选择 CSS 的话，用户就能享有相当广的相容性。此外，CSS 比较稳定。CSSLevel1(涵盖了目前所见到的大部分)和 CSSLevel2 都是 W3C 的推荐标准。

然而，XSL 明显的比 CSS 更强，而且，终有一日，XSL 会成为一个可用的标准。CSS 只能让用户提供元素内容的样式，它并不能让用户改变或是重新排列这些元素内容，也不能让用户依据内容或是属性来选择不同的样式给不同的元素，也不能加入额外的文字内容（比如符号等等）。在数据很少且没有一些 HTML“花边”标记的情况下，XSL 也更为适

用。使用 XSL，用户能将重要的数据与其它网页上的东西分开。比如标题、导航列、和一些记号。假如使用 CSS 的话，就必须将所有出现在网页上的东西都加入文档内容中。XML+XSL 可以让数据文档与网页文档分开。因此，XML+XSL 文档是比较容易维护且容易使用的。将来，XSL 会变成真实世界及数据密集度高的应用程序的选择。CSS 较适用在简单的网页上，这种网页 HTML 就已经够用了。假如在用 HTML 时遇到瓶颈，XML+CSS 并不能突破多少，很快就会又遇到新的瓶颈。相反的，XML+XSL 能穿越 HTML 的限制。也许现在我们仍然需要 CSS 来配合现今不支持新标准的浏览器，但是从长久眼光来看，XSL 才是正确的路。然而，如果是用 ALT 子元素来说明的话，我们就可以加入一些内嵌式的标式。

8.2.5 超元数据

在我们使用子元素来存放元数据的情况下，要加入超元数据变得很容易。举例来说，如果我们把一首诗的作者当成是这首诗的元数据时，那么我们记录作者名字所使用的语言便是超元数据。如果 POET 是一个属性而不是元素的话，那么我们就会陷入一种笨重的结构中。而且如果我们想同时加入作者的英文和希腊名字的话，还会变的更庞大。什么才是超元数据？是我们要谨慎思考的事。哪些人会读我们的文档跟他们为什么读我们的文档会影响到他们认定超元数据与数据的标准。例如，当我们只是在阅读学术期刊时，作者是谁和内容是什么之间并没有直接的关系。然而，假如我们在进行学术推行委员会的工作，正在确定一份期刊的作者是谁时，作者的名字和他所发行的文档数量显然就比他的文档内容重要的多。实际上，我们对什么是超元数据什么是数据的定义标准也可能会改变。现在对我们而言无所谓的东西在日后可能对我们是非常重要的。我们可以使样式表来隐藏现在不重要的内容，然后在日后这些数据变重要时再改变样式表来显示出来。然而，如果一开始是用属性来存这些内容的，那么日后要再使它们显现出来是很困难的。所以，通常我们的作法是重写文档而不是改变样式表。

8.2.6 使用属性的最佳时机

前面我们罗列了一大堆使用元素而不要使用属性的理由，但有必要说明一下，其实也有一些时候使用属性是很自然的。首先，就像前面所提的，属性只适用于非常简单且非结构化、读者不会想看的数据。就像 IMG 标记中的 HEIGHT 和 WIDTH 属性一样。虽然随着图片的改变这两个属性也会改变，但是这些改变不外乎只是短短的文字串，很好处理。HEIGHT 和 WIDTH 都是一维的量，所以很符合属性应用的要求。再者，属性很适合用在记录有关文档的简单内容，这些内容通常跟文档的内容没什么关系。举例来说，我们有时会给每一个元素一个 ID 属性。ID 就是文档中的每一个元素所拥有的独特字串。可以利用这个 ID 做很多事，比如连结到文档中某一个元素（即使这个元素随着文档的改变而更换位置）。

ID 属性能够连结到特定的元素。这样一来，它们就可以提供像 HTML 的 A 元素中的 NAME 属性一样的功用。其它有关连结数据，像是 HREF 用来连结、SRC 用来读进图档或是二元数据等等，也都是以属性来进行，而且还做的很好。属性也常常用来储存一些有关特殊文档样式的内容。比如说，假设 TITLE 元素一般代表粗体字，但是想要让其中一个 TITLE

元素中的字是粗体加斜体的话。这样就能在不变动文档结构的条件下，来嵌入一些样式内容。这样的架构就能让一些不能加入元素到他们的所要标记里的文档编写者能对文档有更好的控制。例如，一个网管可能想要用某种特定的 DTD，但是不想让每个人都能设定这个 DTD。然而，他又想让他们可以对网页作一些微调。那么就可以利用这个架构来限制同时开放一些权限。如果不利用这个方法，我们就会发现又回到了 HTML 中。XML 是设计来帮助我们的，它的格式是可以有意义地自由混合，而它的文档不会再那么一成不变了。最后一个使用属性的理由是要维持与 HTML 的相容性。至少可以使用一些看起来很像 HTML 中的标记， 比如<IMG>、<P>和<TD>，也可以引用标准的 HTML 属性给这些标记用。这样有两个好处，一个是让一些较旧的浏览器至少可以部分解析并显示用户的文档，另一个是让写文档的人有较熟悉的语法可以用。

8.2.7 空标记

在上面我们完全没有使用属性，这是一种极端的情况。另一种情况，当然我们也可以完全把内容储存在属性中。但是一般来说，我们并不采用这种方法。将所有的内容储存在元素的内容中，也是一种极端的作法，但是在实作上却简单的多。然而，在下面的介绍中，我们为了解释空标记，将采用全为属性的作法。在某一种情况下，我们也许会发现元素中会没有东西（也就是没有什么好写的），这个时候，可以用所谓的空标记。空标记没有所谓的起始标记与结束标记，相反的，只有一个“空标记”。空标记与起始标记不同的是它以“/>”作结尾而不是“>”。举例来说，我们可以这样写“<PLAYER/>”来替代“<PLAYER></PLAYER>”。标记中也可以加入属性。XML 的解析器会将它当作一般的标记一样。下面的 PLAYER 元素跟上一个以空标记完成的 PLAYER 元素是一样的。“<PLAYER>”和“<PLAYER></PLAYER>”之间的差别只是在语法上的不同罢了。如果用户觉得他不喜欢空标记的语法，或是觉得很难阅读，那么也可以不用它。

8.2.8 构造良好的（well formed）XML 文档

在 HTML4.0 版本中，大约有一百个不同的标记。大部分的这些标记有六种属性，藉以产生数千种不同的变化。因为 XML 较 HTML 更为强大，因此读者可能会以为，必须认识更多的标记。其实不然。XML 由一种简约、延伸的方式而变得有力量，而不是藉由过多的标记。事实上，XML 几乎没有预先定义任何标记。取而代之的，XML 允许用户定义自己所需的标记。然而，这些用户所定义的标记及包含这些标记的文档并不完全是无限制的，他们必须遵守一些特定的规则。而我们将会在本章详述这些规则。遵循这些规则的文档，我们称之为构造良好的。使文档符合标准格式，是让文档能被 XML 处理器和浏览器读取的最基本要求。

1. XML 文档的组成

一份 XML 文档是由文字组成的，这些文字包含了 XML 标记和字元数据。它们是一串固定长度的位元位，遵守某种限制。它可以说是一个文档，也可以说不是。例如，一份 XML 文档可以是：

- 被储存在数据库中。

- 被 CGI 程序在记忆体中创造出来。
- 几个不同文档的连结，这些文档彼此互相嵌入。
- 不存在于自身的文档中。

然而，如果用户将一份 XML 文档视为一个文档，并不会有什么重大的错误。只要在心里记得它并非真的是硬盘上的一个文档就可以了。

XML 文档是由一种我们称为实体的储存单位所构成。每一个实体是由文字或二元数据所构成，二者仅能择其一。文字数据是由字元所组成的，二元数据是用于影像和 applet 等等……举一个具体的例子，一份包含了<IMG>标记的 HTML 文档仅能称为实体而不是文档。一份 HTML 文档加上所有以<IMG>嵌入的图片，这样才能算是一份完整的文档。

我们将只使用一种简单的 XML 文档来做范例。这种 XML 文档只包含一个实体，也就是文档本身。除此之外，这些文档将只采用文字数据，而不用二元数据，如影像或 applets。这些文档不须依靠其他的文档，本身即可完全被了解。也就是说，这些文档是独立的。这类文档在它的 XML 声明中，有一个 standalone 属性，属性值为 yes，如下：

```
<? xml version=”1.0”  standalone=”yes”?>
```

外部实体和实体引用可被用来将多个文档和其他数据来源结合起来，组成一个 XML 文档。这些文档若不参考其他文档，无法被解读。故这些文档的 standalone 属性值是 no。

```
<? xml version=”1.0”  standalone=”no”?>
```

8.2.9 实体引用

实体引用是一种标记。这种标记在文档被解析之时，会被替换为字元数据。XML 预先定义了五个实体引用。实体引用用来替换一些特殊的字元。若不以实体引用替换，这些特殊字元会被翻译为标记的一部分。例如，实体引用“&lt；”代表的是小于号（<）。小于号直接使用的话，将被视为起始标记。在 XML 中，实体引用必须以分号作结，这点和 HTML 是不同的。因此，“&gt；”是一个正确的实体引用，而“&gt”不是。在一般的 XML 的文档，小于号会被译为起始标记，而“&”号会被翻译为实体引用。因此，若要表示小于号和“&”符号，则必须写成“&lt；”和“&amp；”。例如：我们把“Ben & Jerry’s New York Super Fudge Chunk Ice Cream”这段话写为

```
Ben &amp; Jerry’s New York Super Fudge Chunk Ice Cream.
```

大于号、双引号、单引号，都必须以此方式改写，否则便会被视为标记的一部分。然而，比较简单的方式是，我们仅需养成这种改写的习惯，而不要理会标记是如何翻译的。此外实体引用也可以被用于属性值中。

8.3 XML 文档的格式良好性与有效性

如果一个 XML 文档是格式良好的，则说明它符合 XML 的基本规则。如：

- 每个元素必须有起始和结束标志。
- 它必须有一个，而且只有一个根元素。
- 正确地格式化空元素。
- 开始和结束标注可以大写或小写，但它们必须匹配。

- 元素必须正确地嵌套。
- 属性值必须用引号括起来。

一个有效的 XML 文档是格式良好的，并且已经证实对一个 DTD 是有效的，这意味着 XML 解析器已经判断文档符合与该文档关联模式的规则。

8.4 一个 XML 文档实例

下面的实例代码描述了中国甲 A 足球联赛的信息，包括各参赛队的队名、主要队员、战绩等相关资料（部分资料数据不准确）。

```
<?xml version="1.0" encoding="gb2312" standalone="yes"?>
<SEASON YEAR="2000">
  <LEAGUE NAME="甲 A 足球联赛">
      <TEAM NAME="大连实德" CITY="大连">
          <PLAYER NAME="张恩华" POSITION="后卫" GAME="26">
          </PLAYER>
          <PLAYER NAME="韩文海" POSITION="守门员" GAME="26">
          </PLAYER>
          <PLAYER NAME="李明" POSITION="前卫" GAME="23">
          </PLAYER>
          <POSITION>1</POSITION>
      </TEAM>

      <TEAM NAME="上海申花" CITY="上海">
          <PLAYER NAME="祁宏" POSITION="前锋" GAME="18">
          </PLAYER>
          <PLAYER NAME="吴承瑛" POSITION="后卫" GAME="20">
          </PLAYER>
          <PLAYER NAME="申思" POSITION="前卫" GAME="25">
          </PLAYER>
          <POSITION>2</POSITION>
      </TEAM>

      <TEAM NAME="四川全兴" CITY="成都">
          <PLAYER NAME="黎兵" POSITION="前锋" GAME="23">
          </PLAYER>
          <PLAYER NAME="姚夏" POSITION="前锋" GAME="21">
          </PLAYER>
          <PLAYER NAME="高健斌" POSITION="守门员" GAME="26">
          </PLAYER>
          <POSITION>3</POSITION>
      </TEAM>

      <TEAM NAME="重庆隆鑫" CITY="重庆">
          <PLAYER NAME="姜峰" POSITION="前卫" GAME="22">
```

```
    </PLAYER>
    <PLAYER NAME="魏新" POSITION="后卫" GAME="21">
    </PLAYER>
    <PLAYER NAME="比坎尼奇" POSITION="前卫" GAME="26">
    </PLAYER>
    <POSITION>4</POSITION>
</TEAM>

<TEAM NAME="鲁能泰山" CITY="济南">
    <PLAYER NAME="宿茂臻" POSITION="前锋" GAME="26">
    </PLAYER>
    <PLAYER NAME="宋黎辉" POSITION="前卫" GAME="15">
    </PLAYER>
    <PLAYER NAME="卡西亚诺" POSITION="前锋" GAME="26">
    </PLAYER>
    <POSITION>5</POSITION>
</TEAM>

<TEAM NAME="北京国安" CITY="北京">
    <PLAYER NAME="商毅" POSITION="前锋" GAME="24">
    </PLAYER>
    <PLAYER NAME="邵佳一" POSITION="前卫" GAME="26">
    </PLAYER>
    <PLAYER NAME="李红军" POSITION="后卫" GAME="20">
    </PLAYER>
    <POSITION>6</POSITION>
</TEAM>

<TEAM NAME="沈阳海狮" CITY="沈阳">
    <PLAYER NAME="谢育新" POSITION="前卫" GAME="16">
    </PLAYER>
    <PLAYER NAME="闵劲" POSITION="前卫" GAME="23">
    </PLAYER>
    <PLAYER NAME="徐冀宁" POSITION="前卫" GAME="26">
    </PLAYER>
    <POSITION>7</POSITION>
</TEAM>

<TEAM NAME="辽宁抚顺" CITY="抚顺">
    <PLAYER NAME="曲圣卿" POSITION="前锋" GAME="20">
    </PLAYER>
    <PLAYER NAME="李铁" POSITION="前卫" GAME="26">
    </PLAYER>
    <PLAYER NAME="李金羽" POSITION="前锋" GAME="22">
    </PLAYER>
    <POSITION>8</POSITION>
```

```
</TEAM>

<TEAM NAME="深圳平安" CITY="深圳">
    <PLAYER NAME="哈吉" POSITION="后卫" GAME="20">
    </PLAYER>
    <PLAYER NAME="彭伟国" POSITION="前卫" GAME="25">
    </PLAYER>
    <PLAYER NAME="谢峰" POSITION="后卫" GAME="26">
    </PLAYER>
    <POSITION>9</POSITION>
</TEAM>

<TEAM NAME="天津泰达" CITY="天津">
    <PLAYER NAME="于根伟" POSITION="前锋" GAME="24">
    </PLAYER>
    <PLAYER NAME="江津" POSITION="守门员" GAME="26">
    </PLAYER>
    <PLAYER NAME="张效瑞" POSITION="前卫" GAME="23">
    </PLAYER>
    <POSITION>10</POSITION>
</TEAM>

<TEAM NAME="青岛颐中" CITY="青岛">
    <PLAYER NAME="曲波" POSITION="前卫" GAME="22">
    </PLAYER>
    <PLAYER NAME="张卫华" POSITION="后卫" GAME="26">
    </PLAYER>
    <PLAYER NAME="彭伟军" POSITION="前卫" GAME="21">
    </PLAYER>
    <POSITION>11</POSITION>
</TEAM>

<TEAM NAME="云南红塔" CITY="昆明">
    <PLAYER NAME="福迪" POSITION="前锋" GAME="20">
    </PLAYER>
    <PLAYER NAME="常辉" POSITION="前卫" GAME="23">
    </PLAYER>
    <PLAYER NAME="马庆" POSITION="后卫" GAME="26">
    </PLAYER>
    <POSITION>12</POSITION>
</TEAM>

<TEAM NAME="厦门厦新" CITY="厦门">
    <PLAYER NAME="唐晓程" POSITION="前锋" GAME="20">
    </PLAYER>
    <PLAYER NAME="于远伟" POSITION="前卫" GAME="22">
```

```
            </PLAYER>
            <PLAYER NAME="朱广辉" POSITION="后卫" GAME="23">
            </PLAYER>
            <POSITION>13</POSITION>
        </TEAM>

        <TEAM NAME="延边敖东" CITY="延吉">
            <PLAYER NAME="黄东春" POSITION="前锋" GAME="24">
            </PLAYER>
            <PLAYER NAME="高钟勋" POSITION="前卫" GAME="12">
            </PLAYER>
            <PLAYER NAME="佐拉" POSITION="前锋" GAME="24">
            </PLAYER>
            <POSITION>14</POSITION>
        </TEAM>
    </LEAGUE>
</SEASON>
```

在下一章里，我们还会基于这个例子进行分析和处理，读者将会学到更多的关于创建 DTD 文档的知识。

8.5 小　　结

本章引导读者学会如何创建一个简单的 XML 文档：建立 XML 标记，将样式表附加到 XML 文档获得特定的显示效果，将 XML 载入浏览器等等。在此前提下，我们将进一步深入研究 XML 文档的组成结构和相关规范，为编写大型的复杂的 XML 文档打下良好的基础。

第九章　创建属于自己的 DTD

XML 是一种标记语言。在这一章里，读者将开始学习将设计出来的标记描述出来，像这样的标记语言是由文档类型定义（DTD）所定义出来的，这就是本章所要介绍的部分。个别的文档以 DTD 为标准来比较的过程称作有效确认。如果该文档能符合 DTD 的限制，那这份文档即可称为有效。反之，则称之为无效。

本章包括以下内容：

- 什么是文档类型定义
- 文档类型声明
- 判定 DTD 是否有效
- 元素声明
- DTD 中的注释
- 文档共享相同的 DTD
- MATHML 的 DTD 文档

9.1　什么是文档类型定义

文档类型定义（Document Type Definition）缩写为 DTD。它提供了一连串的元素、属性、表示法和文档中的各种实体以及其相互间的关系。DTD 详细地叙述出一组文档结构的规则，举例而言，DTD 可能会指定一个 BOOK 元素只能有一个 ISBN 的子元素，也恰好有一个 TITLE 的子元素，但是可以有一个或是一个以上的 AUTHOR 子元素及可有可无的 SUBTITLE 子元素。DTD 以一连串对标签的定义，来完成特有的元素、实体、属性及表示法的定义。

DTD 可以被引入到所描述的文档的档案中，也可以用外部的 URL 来连结，其中以外部连结的 DTD 还可以让网站上各个不同的文档共享。DTD 提供了一种让网络上的组织团体都同意的文档类型，也使得新的准则转移到这种标记的形式上。比如说，一个出版商为了便于出书而希望作家都能依循某种特定的格式，然而某个作家可能喜欢把字全都写在一行而不去注意将小标题对准标准线。如果他依照 XML 的规格来写的话，那出版商就能很容易的检查出作者是否遵照原本 DTD 所定好的格式，如果作者没有照格式来的话，甚至可以很快找出有哪几项是和标准不同的，对编辑来说，这可比从作家的风格来找出那少数几个不合规定的地方来得容易多了。

DTD 让程序能够理解从别的地方来的文档，举例来说，如果化学家们经由适当的组织，如 American Chemical Social，协商通用一种符合 DTD 的化学表式法，那他们就能看得懂世界各地其他化学家所提出的报告了。DTD 定义出了什么应该、什么不应该出现在一份文档之中，同时也建立出了编辑器必须支持浏览或编辑其元素所需要的准则。更重要的是它还建立出延伸的部分、那些 DTD 判为无效的部分。如此一来才能避免软件供应商把持这些原

本应该要开放的协议而垄断市场。

此外，DTD 还能在不提供其原始资料的情况下，表示出一个网页或文档的架构元素，也就意味着用户能先建立出很好的风格架构的基础上，很安全、不破坏结构地把用户想要放的资料放上去。就好像在主体建筑结构做好了之后，再慢慢开始粉刷一样。在读资料的人不会看到，甚至不会知道文档的结构，但只要这样的架构在，无论是作者、JavaScript、CGI、资料库或是其它的程序都能毫无困难地使用这个文档。

用户还可以利用 DTD 来做更多的事情，可以用它来定义一个常用文档的集汇，例如签名或地址，可以确保输入资料的人可以遵照用户想要的格式，也可以把资料从相关的资料库转移过去或是传送过去，甚至能将 XML 当成转换不同格式的文档到 DTD 的一个媒体，所以我们不再多说，直接看看 DTD 到底是什么样子的吧！

9.2 文档类型声明

文档类型声明详细记载了一个文档所用的 DTD。文档类型声明出现在文档的前端，在 XML 的声明后，而在根元素之前。它可能包含了文档类型声明本身或是一个可以找到声明的 URL 的地址。当声明分成外部和内部声明时，也有可能两者皆有。

文档类型声明和文档类型定义并不相同。只有文档类型定义才简写为 DTD，文档类型声明要包含或是提及文档类型定义，但不会有文档类型定义包含了文档类型声明这样的说法，这一点不应该混淆。

下面看一下 hello.xml：

```
<?xml version="1.0" standalone="yes"?>
<GREETING>
Hello XML!
</GREETING>
```

这份文档包含了一个元素 GREETING（记得，<?xml version="1.0" standalone="yes"是一个程序指令，而不是一个元素），下面的程序将这份文档加入了文档类型声明而重写了一次，其中声明了根元素为 GREETING。而这个文档类型声明也包含了声明 GREETING 元素所包含的是合法字符的文档类型定义。

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE GREETING [
  <!ELEMENT GREETING (#PCDATA)>
]>
<GREETING>
Hello XML!
</GREETING>
```

它和 hello.xml 不同的地方在加进去的那三行：

```
<!DOCTYPE GREETING
<!ELEMENT GREETING (#PCDATA)>
]>
```

这几行是文档类型声明，文档类型声明把 XML 声明和文档本身分成了两部分，而 XML

声明和这个文档类型声明的部分则合称为这个文档的前端，在这个简短的例子当中，<?xml verson="1.0" standalone="yes"?> 是 XML 的声明部分；而 <DOCTYPEGREETING [<!ELEMENT GREETING (#PCDATA)>]> 则是文档类型声明的部分；<!ELEMENT GREETING （#PCDATA)>是文档类型定义，而<GREETING> Hello XML! </greeting>则是文档本身，或称之为根元素。

一个文档类型声明都是以“<!DOCTYPE”为开头，而以“]>”为结尾，虽然换行或空白字符是不影响文档类型声明的，但通常我们还是会习惯把开头跟结尾的部分放在不同行，当然也可以把范例中的文档类型声明写在同一行，像下面这样：

```
<!DOCTYPE GREETING [ <!ELEMENT GREETING (#PCDATA)> ]>
```

例中这个根元素的名字“－GREETING”直接接在“<!DOCTYPE”之后，这不只代表了名字，更代表了一个要求，任何一个包含文档类型声明的有效文档都必须要有这个根元素“GREETING”。而在对应的“[”和“]”之中则是文档的文档类型定义。

DTD 包含了一连串定义出个别的元素、实体和属性的标签定义，其中的一个就定义出了根元素。整个 DTD 很简洁地就列在下面这一行：

```
<!ELEMENT GREETING (#PCDATA)>
```

但是一般而言，DTD 是会比这复杂而冗长许多的。

这一行中“<!ELEMENT GREETING （#PCDATA）>”是一个元素类型声明（在 XML 中，两个字即使只有大小写不同也是视为不同的）。在这个例子里唯一被声明的元素是“GREETING”。这个元素可以包含了合法字符（或 #PCDATA）。整份文档除了标签外都得是合法字符，这也包括了当文档被分析时一些被取代的实体转换，如&。

9.3 判定 DTD 是否有效

一份有效的文档必须符合 DTD 要求的限制。此外其根元素要在文档中的文档类型声明里进行声明才行，上面程序的文档类型声明和 DTD 告诉我们一个有效的文档要像这样：

```
<GREETING>
  Welcome to the XML's World!
</GREETING>
```

但不能像下面这样：

```
<GREETING>
<tag1> Welcome to the XML's World! </tag1>
<tag2/>
</GREETING>
```

也不能像下面这样：

```
<GREETING>
<GREETING> Welcome to the XML's World! </GREETING>
</GREETING>
```

这份文档除了把合法字符放在起始标签“<GREETING>”和结束标签“</GREETING>”之中外，不能多加也不能少掉任何的东西。有效的文档不像是格式良好的文档那样，还能出现一些随便加进去的标签，任何一个用到的标签都得在 DTD 里声明过才行。在上面程序

中的“<GREETING>”只能拿来当作起始的根元素。并不是所有的文档都必须是有效的，也不是所有的解析器都会检查文件是否有效，事实上大部分的浏览器，包括 IE5 和 Mozilla 都不检查文档是否有效。

一个能判断文档有效性的解析器读进 DTD 并且会检查文档是不是都遵循了 DTD 的限制，如果是的话，解析器会将资料送到处理 XML 文件的地方（如浏览器或是资料库），如果解析器发现有错误，则会显示出这个错误。如果用户想直接写 XML，则必须先检查文档是否有效，这样才能确定不会让读者遇到错误。

有些程序库还提供了可以独立用指令列的解析器，也就是能读进一份文档，然后显示出是否有错误而不将文档展示出来的程序。例如，samples.XJParse 是一个放在 IBM's XML for Java1.1.16 类程序库的包，而 XJParse 则是包里的一个 Java 程序。如果用户想使用这个程序的话，必须先把 XMLfor Java 的档案加到用户的 Java 类路径里。然后就可以通过开启一个 DOS 的视窗或是通过命令行方式，传给这个程序用户要解析其有效性的文档的本地名称或是 URL 地址，就像下面这样：

```
C:\xml4j>java samples.XJParse.XJParse -d D:\XML\08\Invalid.xml
```

也可以用 URL 地址来取代档案名称，像下面这样：

```
C:\xml4j>java samples.XJParse.XJParse -d
http://metalab.unc.edu/books/bible/examples/08/in 有效.xml
```

此外，XJParse 还会把树状的文档格式放在它所对应的错误之后，这并不是太好的输出方式。不过一个像 XJParse 这样的判断文档是否有效的解析器并不是用来显示出文档的结果的，它的工作应该是把文档分成树状的结构，然后将树中的每一个结点的资料传给用来显示这些资料的程序，比如一些浏览器，如 Netscape Navigator 或是 IE；或者也可能是个资料库；当然，更有可能是用户自己所写的程序。用户通过使用 XJParse 或其它的解析器来判断自己是不是写了一个别的程序都看得懂的好的 XML 文档，实际上用户做的不是输出，而是在测试自己的文档。

因为 XML for Java 和大多数的有效解析器都是以 Java 写成的，所以它们也继承了所有跨平台的 Java 程序的缺点。首先，在使用解析器之前必须先安装 Java 开发包（JDK）或是 Java。接下来，要把 XML for Java 的档案加到用户的类的路径里。不过这两项工作都并非容易的事，而这些工具十分难以使用。

如果用户是要写网络上的文档，检验这些文档最简单的方法就是用浏览器来读取这些文档，看看有没有错误出现。不过并不是所有的浏览器都会对文档的有效与否加以检查，有些浏览器甚至只检查文档是否是格式良好的。IE5.0 beta 2 会检查文档是否有效，但其正式发行版则没有这项功能。

如果文档是放在网络上，而且又不是特别秘密内容的话，应用网上的有效解析器也是一个很好的选择，这些解析器只要用户输入一个 URL 即可。这和 Java 的运行时软件、类别路径和环境变数混乱比起来，具有很明显的优点。

9.4 元 素 声 明

创造出适合一份指定文档的 DTD 的第一步，是要先了解所要写的那些资料的结构在

DTD 中要定义的元素。有时候资料是很有组织的，如通讯录；有的则格式比较自由，如有插图的小故事或是杂志里的文章。

在 XML 中，元素有两种基本的形式。简单元素包含了文字，也就是所谓的合法字符，在文中的#PCDATA 或 PCDATA。复合元素则包含了其他的元素或是较少见的文字，在标准的 XML 中是没有整数、浮点数、日期或其它的文档类型的。早期曾经对用 XML 语法来定义一些原本以 DTD 来描述的文档类型的方案做过一些努力。如果用户已经制定好了手上的资料，哪些是可有可无的、哪些之间是要有什么关系的，都已经了若指掌，那么就可以准备开始做一个简明的文档的 DTD 了，如果觉得有一点复杂的话，那就先把这些关系简化点。用剪切粘贴的动作来将一个 DTD 引入另一个 DTD 里是个方便的方法，很多元素都能在别的文档里面再被重复地使用。

每一个用在有效的 XML 文档里的标签都要在 DTD 的元素声明里声明过，一个元素声明详述出它的名字和可能有的内容成份。这些列出来的内容则被称为内容详述。所谓的内容详述是用一个简单的文法来精确地说明什么内容应该或是不应该出现在一份文档之内。听起来好像很复杂，其实你只要加一些像*、?或+之类的标点符号在元素的名字后面，就能指出这些元素可能出现一次以上、可有可无或是一定得出现超过一次。

DTD 是很严格的：没有被明确允许的都被视为禁止的。然而 DTD 的语法却能让用户用几个式子很紧密地描述出一些复杂累赘的关系。用从上层到下层、从外层到里层的方法来做 DTD 是个不错而简单的方法。这让用户能在建立起一个 DTD 时顺便也做出一份范例文档，一方面可以看看 DTD 是不是正确的，另一方面可以确定写出来的格式正是自己想要的。

9.4.1 ANY 关键字

用户要做的第一件要务就是要先定出一个根元素来。所有的元素类型定义都是以“<!ELEMENT”做为开头（一定要大写），以“>”做为结尾。其中声明了元素的名字，然后是内容的叙述。而关键字 ANY（也是都要大写）则说明了所有可能的元素或是合法字符都可以是根元素的子元素。

用 ANY 的根元素是相当普遍的，特别是一些本身没什么结构的文档，但是仍有一些情况还是避免用 ANY 比较好。原则上还是将每一个标签描述得越精确越好，DTD 通常在开始发展时都是非常精确的，到后来为了适应使用上的需要或是较难想象的内容才减低它的限制。因此，一开始定订严格的格式再慢慢放松是比较好的。

9.4.2 #PCDATA

既然任何标签都有可能在文档中出现，那么任何可能出现的标签都要先声明好，以下是 YEAR 这个元素的元素声明：

```
<!ELEMENT YEAR (#PCDATA)>
```

这个声明说明了 YEAR 只能包含合法字符，也就是不含标签的内容，它也不能包含它自己作为子元素。下面的 YEAR 元素是有效的：

```
<YEAR>2000</YEAR>
```

下面这一个 YEAR 元素也是有效，因为 XML 并不检查 PCDATA 的内容是什么，只要

是合法文字而且没有标签就都可以。

```
<YEAR>football and basketball</YEAR>
```

然而下面这个 YEAR 标签就无效了，因为它包含了子元素。

```
<YEAR>
<MONTH>January</MONTH>
</YEAR>
```

在文档类型声明中的 SEASON 和 YEAR 的元素声明可以写成如下的形式：

```
<!DOCTYPE SEASON [
<!ELEMENT SEASON ANY>
<!ELEMENT YEAR (#PCDATA)>
]>
```

通常，空白和首段空格都不影响程序代表的意义，元素声明的顺序不同都视为相同。下一个文档类型声明和前一个所描述的是同样的内容：

```
<!DOCTYPE SEASON [
<!ELEMENT YEAR (#PCDATA)>
<!ELEMENT SEASON ANY>
]>
```

这两者的 SEASON 元素都只能包含合法文字及任何数量、任何顺序声明过的元素。唯一有声明的元素是只能包含有合法文字的 YEAR 元素，因为元素 SEASON 也可以包含合法文字，因此还可以在 YEAR 之外加一点文字。到最后我们还是不会使用这样的文档，但无论如何，现在它是合法的，因为 SEASON 声明为能接受 ANY（任何）内容的。大多数的情况下将某个元素的子元素定为 ANY 是很方便的，而后再将真正要用到的子元素取代掉这些。

9.4.3 子元素列表

因为根元素元素声明为接受任何型态的子元素，所以用户放什么东西进去都无所谓。编辑一些没什么组织结构的文档，比如一些杂志上的文章，一些段落、图表、照片、小标题到处都是的时候，这样自由的设计将会十的方便。不过有时候用户还是得多练习如何控制资料的位置，使用更多的限制。

即使一个元素会以子元素的身份出现在别的元素之后，它仍只需在它的“<!ELEMENT>”声明中声明一次就可以了，改变这些声明的顺序，只要确定把它们都放到了 DTD 里，它们都将被视为相同的。

9.4.4 序列设定

现在让我们为 SEASON 元素也加点限制。因为一个 SEASON 包含了一个 YEAR 和一个 LEAGUE 元素，所以现在我们将原本 SEASON 能包含 ANY（任何）的定义，改为将这两个子元素加入 SEASON 的元素声明中，并且放在括弧里用逗点格开，就像下面这样：

```
<!ELEMENT SEASON (YEAR, LEAGUE)>
```

这样的一排子元素称为一个序列，在这样的声明之下，每一个有效的 SEASON 元素都只能有一个 YEAR 元素和一个 LEAGUE 元素，其它就什么都不能有了。现在整个文档类型声明看起来就像这样：

```
<!DOCTYPE SEASON [
<!ELEMENT YEAR (#PCDATA)>
<!ELEMENT LEAGUE (LEAGUE_NAME)>
<!ELEMENT LEAGUE_NAME (#PCDATA)>
<!ELEMENT SEASON (YEAR, LEAGUE)]>
```

9.4.5 不确定数目的子元素

1. 一个或一个以上

假设每一个 LEAGUE 元素有一个 NAME 子元素，同时可能有很多个 TEAM 子元素，要描述 NAME 十分简单，就如下面所述：

```
<!ELEMENT LEAGUE(NAME)>
<!ELEMENT NAME (#PCDATA)>
```

然而要声明出 TEAM 子元素则要有一点技巧。如果在 DIVISION 中要有四个 TEAM 的子元素并不难，如下所示：

```
<!ELEMET LEAGUE(NAME, TEAM, TEAM, TEAM, TEAM)>
```

五个或六个的情况也都并不困难。但如果要如何描述出是要四到六个这样的情况呢？事实上 XML 没有提供这么做的方法。但可以简化要求，变成要一个或是一个以上的子元素，并且只要在那个给定的元素之后加上一个加号（+），然后放在待定义的母元素的子元素栏，就如下面这样：

```
<!ELEMENT LEAGUE (NAME, TEAM+)>
```

这也就说明了 LEAGUE 必定要有一个 NAME，并且后面要跟着一个或是一个以上的 TEAM 子元素。

2. 零个或多个子元素

每一个 TEAM 元素都包含着一个 CITY 元素、一个 NAME 元素和没有给定数量的 PLAYER 元素。我们想要描述出一个 TEAM 元素可能没有，也可能有一个以上的 PLAYER 元素的状态，只要在子元素栏中个元素后加上星号（*）就可以了。例如：

```
<!ELEMENT TEAM (CITY, NAME, PLAYER*)>
<!ELEMENT CITY (#PCDATA)>
< !ELEMENT NAME (#PCDATA)>
```

如果我们要描述一个可能包含一个或者零个子元素的情况，我们可以在子元素后面加一个问号“?“来表达。

一般说来，每一个母元素都会有不少子元素。如果要指出这些元素成为一个序列，则要把它们用逗号分格开来。然而，每个子元素也都可能以星号、问号、加号来表示出他们在序列中应该出现的次数。目前我们只能要求到某个元素会或是不会在某个特定的次序中出现，但可以想办法让 DTD 更灵活一点，比如在指定的位置上让作者自由选择使用哪一种的元素。举例来说，以一个描述顾客付费状况的 DTD 来说，每个 PAYMENT 元素可能会有一个 CREDIT_CARD 或者是一个 CASH 的子元素提供有关付费方式的信息。然而，这两种是不会一起出现的。

当我们让文档的作者能在几个元素中选择某一个的时候，只要把原来用逗号隔开改成

用铅直线（“|“）分格母元素的子元素栏里的元素就行了。这种描述内容的方法就叫作选择。如果我们只想要在一堆元素中选一个出来，就可以把这些元素都用铅直线分隔开来。当我们学会用括号把全部的元素组成一部分、一部分时，我们会发现铅直线更加有用了。我们也可以用括号将这些组合起来的整体再加进原本那些星号、加号、问号的用法，这样可以表现出更多种的出现方式来。

9.4.6 括号中的元素

最后一个在处理元素上该学会的，就是学会如何用括号来将元素组合起来。每一组由括号结合数个元素所组合起来的被视为另一个新的元素。像这样在括号中的元素还能再放进一层以括号组合而成的元素中，一如原本应该是要放一般元素的位置。此外也可以再加进一些星号、问号、加号一起使用，同时可以一直不断地层层组织，这样可以创造出各种复杂结构。

举例来说，考虑一个两个元素所对应组成的组合。这个例子也就同于 HTML 定义的用法。每一个“<dt>”标签都要对到一个“<dd>”的标签。如果想要在 XML 中表现出这样的结构出来，那么这个“dl”元素的声明就会如下面这种形式：

```
<!ELEMENT dl (dt, dd)*>
```

这个括号表示出它是成对出现的“<dt><dd>”标签，而非单独出现的“<dd>”标签。

通常元素或多或少都会以随机的方式出现，新闻杂志的文章普遍都有一个标题，后面大概还会跟着数段的文字，然而还会有一些散置文章各处的图表、照片、小标题和引言，说不定最后还会有一条结束线。我们可以在母元素的声明中列出这些所有的可能的子元素，并且运用括号、铅直线来组合，同时也可以在括号外放一个星号，说明这个括号内的元素出现的次数允许是零次或更多次，就如下例：

```
<!ELEMENT ARTICLE (TITLE, (P | PHOTO | GRAPH | SIDEBAR
        | PULLQUOTE | SUBHEAD)*, BYLINE?)>
```

另一个例子里，假设我们想有一个 DOCUMENT 元素，其中包含了一个 TITLE 元素、若干个任意混杂的文字和图片及最后的一个可有可无的签名，可以声明如下：

```
<!ELEMENT DOCUMENT ( TITLE, (PARAGRAGH | IMAGE)*, SIGNATURE?)>
```

上面这个例子并不是描述这样结构唯一的方法，事实上它也不是最好的方法。另一种选择则是声明一个包含 PARAGRAPH 和 IMAGE 元素的 BODY 元素，然后再把它加进 TITLE 元素和 SIGNATURE 元素之间，例如：

```
<!ELEMENT DOCUMENT (TITLE, BODY, SIGNATURE?)>
    <!ELEMENT BODY ((PARAGRAPH | IMAGE)*)>
```

这两种用法的不同之处是第二种用法在文档中需要一个额外的元素 BODY，这个元素提供了一个新的层级，或许这不一定对使用这份文档的应用程序有用。但真正的要考虑的是读者（或是读这份文章的程序）是不是想要把 BODY 视为一个整体和 TITLE 及 IMAGE 分开，并用其中元素的个数来分辨出内容来。

另一方面而言，可能有一组由许多不同类且排列不规则的元素，例如 CD、唱片、录音带的记录。用以区分各种不同分类的元素声明可以如下面这种形式：

```
<!ELEMENT MUSIC_LIST (CD | ALBUM | TAPE)*>
```

有一些情况是元素声明所难以掌握的，例如找不到什么好方法能规定一份文档必须以

元素 TITLE 作为开头、以 SIGNATURE 作为结尾而中间可以放任意种类的元素。这是因为元素 ANY 是不能跟其它的元素结合的。

另外，一般而言，对一个元素出现位置所做的限制越少，那么对该元素个数的控制就会相对越少。例如我们不能要求一份文档里只有一个某元素，但那个元素可以出现在文档当中的任何一个地方。

然而用括号来组成一个个元素的区块，无论是排成一排用逗号分开，或是用铅直线分开，都能制定出可以决定细部规则的复杂结构。但也不要因此而过分追求复杂化，简洁的处理方法还是比较理想的。DTD 定得越复杂，要写出一份有效的文档也会变得越不容易，更不用说要维护这个 DTD 会变得有多难了。

9.4.7 使用混合内容

读者可能已经注意到，目前范例中所出现过的元素都只包含了元素或合法字符之一，却没有两者兼备的。唯一的例外大概就是稍早出现的根元素范例，其标签组都还没发展完全。在这些例子中，因为根元素能包含 ANY 的资料，因此同时有子元素和文字是允许的。

我们可以声明一个同时包含文字和子元素的标签，称为混合内容。可以用这样的方法来决定每个 TEAM 里任意的文字区块。例如：

```
<!ELEMENT TEAM (#PCDATA | CITY | NAME | PLAYER)*>
```

把子元素和文字资料混合在一起的动作，对文档的结构产生严格的限制。特别是子元素只能出现它们的名字。不能强迫它们出现的顺序、它们出现的次数，甚至是它们究竟要不要出现。必须把下面这个 DTD 中的子元素视为 DTD 中的必然现象：

```
<!ELEMENT PARENT (#PCDATA | CHILD1 | CHILD2 | CHILD3)*>
```

更改子元素出现的次数是错误的，也不能在有文字资料的元素声明中使用逗号、问号或是加号。文字资料只能用铅垂线分开，其它的都不行。像下面这种情况就是错误的：

```
<!ELEMENT TEAM (CITY , NAME , PLAER* , #PCDATA)>
```

混合内容的主要理由是：当我们想要把旧的文字资料转成为 XML，并且在加入一些标签却还没完成就测试其是否有效时。这是一个不错的技巧，毕竟在一写完程序代码就发现错误是比几个小时之后才发现要好得多了，不过这样却是进一步发展的阻碍。而这部分最好不要让远端的用户知道。在完成一个 DTD 时，不应该把子元素和文字都混在一起，我们总是可以做出一个存文字资料的标签。

举例来说，我们可以声明一个只包含“#PCDATA”的新元素 RLURB，放在每个 TEAM 元素的末端，用以引入一个文字区块。就像下面这样：

```
<!ELEMENT TEAM (CITY , NAME , PLAYER* , BLURB)>
    <!ELEMENT BLURB (#PCDATA)>
```

这不会使得文档有什么明显的改变，所需要做的不过是在元素 TEAM 中加进开始和结束的标签。无论如何，这都使文档变得更加稳固。此外，要从 XML 处理器得到资料的树状结构的 XML 应用程序能在混合资料最少的情况之下，最有效率地处理资料。

9.4.8 空元素

在前面的讨论中，曾经提到过定义出没有内容的元素有时候会很有用。在 HTML 中的例子有图形、横线、换行（ <IMG>、<HR>、
 ），在 XML 中，我们以“/>”当作

标示空元素的结尾。例如前面的几个标签就变为“<IMG/>”、“<HR/>”和“
”。

一份有效文档必须要声明出所有用到的空元素及非空元素。因为空元素没有子元素，所以它们的声明比较简单，用一个<!ELEMENT>声明，加入该元素的名字，然后把原本放子元素的地方换成一个关键字 EMPTY（在 XML 中所有的关键字都是要依照所规定的大小写来使用的），例如：

```
<!ELEMENT BR EMPTY>
<!ELEMENT IMG EMPTY>
<!ELEMENT HR EMPTY>
```

9.5 DTD 中的注释

DTD 和就 XML 文档一样都包含了注释，这些注释不能在声明中出现。注释通常可以把 DTD 组织成几个不同的部分，可以写出一个特定的元素能有什么内容，还可以进一步解释这个元素是什么。举例来说，YEAR 元素的声明可能有以下的注释：

```
<!-  A year should have four digit such as  1999、2000  -->
  <!ELEMENT YEAR (#PCDATA)>
```

注释的多少并没有限制，加进多的注释会使 DTD 程序变长（同时也会变得比较难搜寻、比较不好下载）。

9.6 文档共享相同的 DTD

有效的例子将 DTD 包含在文档的前端，然而 XML 真正的作用是能让不同人写的文档共享同一个 DTD。如果 DTD 不直接包含在文档中，而是从外部连结，那对 DTD 的任何改变将也得转移到每个使用这个 DTD 的文档上。换句话说，被更改过的 DTD 是不保证和之前文档的相容性的。相容性的改变将会破坏文档。

使用外部连结的 DTD 时，文档类型声明也跟着改变。关键字 SYSTEM 能找到该 DTD 绝对或是相对 URL 的路径，取代了原本在方括号内的 DTD，举例如下：

```
<!DOCTYPE root_element_name SYSTEM "DTD_URL" >
```

这里“root_element_name”同样代表根元素的名字，“SYSTEM”是 XML 的关键字，而“DTD_URL”则代表可以找到 DTD 的相对或是绝对的 URL 路径，例如：

```
<!DOCTYPE SEASON SYSTEM "aaa.dtd">
```

9.6.1 远端 URL 的 DTD

如果 DTD 是要给很多文档共同使用的，就不能总是把 DTD 和文档放在同一个目录下，取而代之的，必须要详细地叙述出 DTD 可以在哪里找到。例如：我们假设数学标记语言的 DTD 可以在 http://www.w3.org/TR/NOTE-sgml-xml .里找到，则可以用文档前端的<!DOCTYPE>标签来连结：

```
<!DOCTYPE SEASON SYSTEM
        " http://www.w3.org/TR/NOTE-sgml-xml/mathml.DTD . ">
```

上面的例子使用的是不管在哪里都可以使用的 URL 地址，把 DTD 设成以现在所在目录或是根目录为标准的相对路径也是可以的。例如下面这几个都是有效的文档类型声明：

```
<!DOCTYPE SEASON SYSTEM " TR/NOTE-sgml-xml / mathml.dtd"
    <!DOCTYPE SEASON SYSTEM " /NOTE-sgml-xml / mathml.dtd"
    <!DOCTYPE SEASON SYSTEM "../ mathml.dtd"
```

需要注意的是，一份文档不能有一个以上的文档类型声明，也就是只能有一个<!DOCTYPE>标签。

9.6.2 公共 DTD

SYSTEM 这个关键字是当私有的 DTD 被一个作者或是团体使用时所用的。然而 XML 有一部分的承诺是要让能涵盖工业界的机构，如 ISO 或 IEEE 能够将公共 DTD 标准化。这样的过程能够帮助人们不用一直为同一个东西创造标签，也可以让使用者便于互阅文档。

为了组织外的人所创造的 DTD 使用关键字 PUBLIC 来取代原本的关键字 SYSTEM。此外，这个 DTD 还有一个名字，语法如下：

```
<!DOCTYPE root_element_name PUBLIC "DTD_name" "DTD_URL">
```

同样的，“root_element_name”是根元素的名字，“PUBLIC”是 XML 的关键字，用来表示这个 DTD 是要给多数的人使用的，而且还有个名字。“DTD_name”则是和 DTD 有关的名字，有一些 XML 处理器会用这个名字把 DTD 从中央的储存器找出来。最后的“DTD_URL”则是当一个中央储存器无法用名字找到这个 DTD 时，还可以用来找到它的相对或是绝对路径。

9.6.3 内部和外部的子集

虽然大部分的文档都很容易由部分定义好的东西组合起来，但并非所有的文档都用一样的样板。很多使用所有者自己所制作的元素的文档可能需要用到像 HTML4.0 DTD 那样的标准 DTD，其它的可能还需要重新整理后的标准元素。例如有一个 HTML 网页可能有一个 BODY 元素，它必须要包含一个 H1 的标头，然后跟着一个 DL 的定义列；而另一个网页可能有好几个标头、段落和图片任意散置。如果某个特定的文档有一些特殊的结构是其他网站都没有的，则这份文档比较适合把 DTD 的部分和资料内容放在一起。这也使得文档更便于编辑。

为了这个目的，一份文档可以同时使用内部和外部的 DTD。内部的声明放在<!DOCTYPE>标签结尾的方括号里。

当外部连结和内部连结因为元素名称相同而发生冲突时，内部所定义的那个元素有优先权，这个优先权提供了一个一般、部分继承的手法。

9.7 MATHML 的 DTD 文档

这是一个比较复杂的 DTD 文档，它是规范数学标记语言的 DTD。数学标记语言是为了在 Web 上运用数学公式而开发的一种标记语言。其核心就是 DTD 文档的创建。下面就是该标记语言的 DTD。

```
<?xml version = "1.0"    ?>
<!-- Content model for content and presentation                -->
<!--    and browser interface tags in MathML                    -->
<!-- initial draft 9.May.1997                  syntax = XML         -->
<!-- author = s.buswell sb@stilo.demon.co.uk                     -->
<!-- revised 14.May.1997 by Robert Miner                          -->
<!-- revised 29.June.1997 and 2.July.1997 by s.buswell    -->
<!-- revised 15.December.1997    by s.buswell                     -->
<!-- revised 8.February.1998       by s.buswell                   -->
<!-- revised 4.april.1998               by s.buswell               -->
<!-- W3C Recommendation           7 April 1998                          -->
<!-- ************************************************    -->
<!-- general attribute definitions for class & style & id & other -->
<!-- : attributes shared by all mathml elements                -->
<!ENTITY % att-globalatts          'class CDATA #IMPLIED
                                    style CDATA #IMPLIED
                                    id       ID       #IMPLIED
                                    other CDATA #IMPLIED'   >
<!-- ************************************************    -->
<!-- Presentation element set                                    -->
<!-- presentation attribute definitions -->
<!ENTITY % att-fontsize             'fontsize CDATA #IMPLIED'                    >
<!ENTITY % att-fontweight          'fontweight (normal | bold) #IMPLIED'   >
<!ENTITY % att-fontstyle           'fontstyle (normal | italic) #IMPLIED' >
<!ENTITY % att-fontfamily          'fontfamily CDATA      #IMPLIED'              >
<!ENTITY % att-color                'color CDATA      #IMPLIED'                      >
<!ENTITY % att-fontinfo             '%att-fontsize;
                                    %att-fontweight;
                                    %att-fontstyle;
                                    %att-fontfamily;
                                    %att-color;'   >
<!ENTITY % att-form          'form (prefix | infix | postfix) #IMPLIED' >
<!ENTITY % att-fence          'fence (true | false ) #IMPLIED'       >
<!ENTITY % att-separator     'separator (true | false ) #IMPLIED'       >
<!ENTITY % att-lspace          'lspace CDATA    #IMPLIED'             >
<!ENTITY % att-rspace          'rspace CDATA    #IMPLIED'             >
<!ENTITY % att-stretchy       'stretchy (true | false ) #IMPLIED'         >
<!ENTITY % att-symmetric     'symmetric (true | false ) #IMPLIED'       >
<!ENTITY % att-maxsize         'maxsize CDATA #IMPLIED'            >
<!ENTITY % att-minsize         'minsize CDATA #IMPLIED'            >
<!ENTITY % att-largeop                 'largeop (true | false ) #IMPLIED' >
<!ENTITY % att-movablelimits         'movablelimits (true | false )
#IMPLIED' >
<!ENTITY % att-accent                        'accent (true | false)    #IMPLIED'>
<!ENTITY % att-opinfo '%att-form;
                                %att-fence;
```

```
                        %att-separator;
                        %att-lspace;
                        %att-rspace;
                        %att-stretchy;
                        %att-symmetric;
                        %att-maxsize;
                        %att-minsize;
                        %att-largeop;
                        %att-movablelimits;
                        %att-accent;'              >
<!ENTITY % att-width                    'width CDATA #IMPLIED'     >
<!ENTITY % att-height                   'height CDATA #IMPLIED' >
<!ENTITY % att-depth                    'depth CDATA #IMPLIED'     >
<!ENTITY % att-sizeinfo                 '%att-width;
                                         %att-height;
                                         %att-depth;'               >
<!ENTITY % att-lquote                    'lquote CDATA #IMPLIED'     >
<!ENTITY % att-rquote                    'rquote CDATA #IMPLIED'     >
<!ENTITY % att-linethickness            'linethickness CDATA #IMPLIED' >
<!ENTITY % att-scriptlevel              'scriptlevel CDATA #IMPLIED'>
<!ENTITY % att-displaystyle             'displaystyle (true | false)
#IMPLIED'>
<!ENTITY % att-scriptsizemultiplier 'scriptsizemultiplier CDATA
#IMPLIED'   >
<!ENTITY % att-scriptminsize            'scriptminsize CDATA #IMPLIED'>
<!ENTITY % att-background               'background CDATA #IMPLIED' >
<!ENTITY % att-open                      'open CDATA #IMPLIED' >
<!ENTITY % att-close                     'close CDATA #IMPLIED' >
<!ENTITY % att-separators                'separators CDATA #IMPLIED' >
<!ENTITY % att-subscriptshift           'subscriptshift CDATA #IMPLIED'>
<!ENTITY % att-superscriptshift          'superscriptshift CDATA #IMPLIED'      >
<!ENTITY % att-accentunder               'accentunder (true | false)#IMPLIED'>
<!ENTITY % att-align                    'align CDATA #IMPLIED'                  >
<!ENTITY % att-rowalign                 'rowalign CDATA #IMPLIED'      >
<!ENTITY % att-columnalign              'columnalign CDATA #IMPLIED'     >
<!ENTITY % att-groupalign                'groupalign CDATA #IMPLIED'      >
<!ENTITY % att-alignmentscope           'alignmentscope CDATA #IMPLIED'   >
<!ENTITY % att-rowspacing                'rowspacing CDATA #IMPLIED'          >
<!ENTITY % att-columnspacing             'columnspacing CDATA #IMPLIED'     >
<!ENTITY % att-rowlines                  'rowlines CDATA #IMPLIED'                 >
<!ENTITY % att-columnlines               'columnlines CDATA #IMPLIED'               >
<!ENTITY % att-frame                      'frame (none | solid | dashed)#IMPLIED'   >
<!ENTITY % att-framespacing             'framespacing CDATA #IMPLIED'           >
<!ENTITY % att-equalrows                 'equalrows CDATA #IMPLIED'              >
<!ENTITY % att-equalcolumns              'equalcolumns CDATA #IMPLIED'        >
<!ENTITY % att-tableinfo                 '%att-align;
```

```
                                        %att-rowalign;
                                        %att-columnalign;
                                        %att-groupalign;
                                        %att-alignmentscope;
                                        %att-rowspacing;
                                        %att-columnspacing;
                                        %att-rowlines;
                                        %att-columnlines;
                                        %att-frame;
                                        %att-framespacing;
                                        %att-equalrows;
                                        %att-equalcolumns;
                                        %att-displaystyle;'              >
<!ENTITY % att-rowspan                  'rowspan CDATA #IMPLIED'              >
<!ENTITY % att-columnspan               'columnspan CDATA #IMPLIED'           >
<!ENTITY % att-edge                      'edge (left | right)    #IMPLIED ' >
<!ENTITY % att-actiontype               'actiontype CDATA #IMPLIED' >
<!ENTITY % att-selection                'selection CDATA #IMPLIED ' >
<!-- presentation token schemata with content-->
<!ENTITY % ptoken "mi | mn | mo | mtext | ms" >
<!ATTLIST mi            %att-fontinfo;
                        %att-globalatts;       >
<!ATTLIST mn            %att-fontinfo;
                        %att-globalatts;       >
<!ATTLIST mo            %att-fontinfo;
                        %att-opinfo;
                        %att-globalatts;        >
<!ATTLIST mtext         %att-fontinfo;
                        %att-globalatts;        >
<!ATTLIST ms            %att-fontinfo;
                        %att-lquote;
                        %att-rquote;
                        %att-globalatts;                >
<!-- empty presentation token schemata -->
<!ENTITY % petoken    "mspace" >
<!ELEMENT mspace      EMPTY    >
<!ATTLIST mspace      %att-sizeinfo;
                      %att-globalatts;            >
<!-- presentation general layout schemata -->
<!ENTITY % pgenschema "mrow|mfrac|msqrt|mroot|
                       mstyle|merror|mpadded| mphantom|mfenced" >
<!ATTLIST mrow        %att-globalatts;     >
<!ATTLIST mfrac       %att-linethickness;
                      %att-globalatts; >
<!ATTLIST msqrt       %att-globalatts;     >
<!ATTLIST mroot       %att-globalatts;     >
```

```
<!ATTLIST mstyle       %att-fontinfo;
                       %att-opinfo;
                       %att-lquote;
                       %att-rquote;
                       %att-linethickness;
                       %att-scriptlevel;
                       %att-displaystyle;
                       %att-scriptsizemultiplier;
                       %att-scriptminsize;
                                        %att-background;
                                        %att-open;
                                        %att-close;
                                        %att-separators;
                       %att-subscriptshift;
                       %att-superscriptshift;
                       %att-accentunder;
                       %att-tableinfo;
                       %att-rowspan;
                       %att-columnspan;
                                        %att-edge;
                                        %att-actiontype;
                                        %att-selection;
                       %att-globalatts;              >
<!ATTLIST merror       %att-globalatts;     >
<!ATTLIST mpadded      %att-sizeinfo;
                       %att-lspace;
                       %att-globalatts;        >
<!ATTLIST mphantom     %att-globalatts;        >
<!ATTLIST mfenced      %att-open;
                       %att-close;
                       %att-separators;
                       %att-globalatts;        >
<!-- presentation    layout schemata : scripts and limits -->
<!ENTITY % pscrschema    "msub|msup|msubsup|
                       munder|mover|munderover|mmultiscripts" >
<!ATTLIST msub         %att-subscriptshift;
                       %att-globalatts;        >
<!ATTLIST msup         %att-superscriptshift;
                       %att-globalatts;              >
<!ATTLIST msubsup      %att-subscriptshift;
                       %att-superscriptshift;
                       %att-globalatts;              >
<!ATTLIST munder       %att-accentunder;
                       %att-globalatts;              >
<!ATTLIST mover        %att-accent;
                       %att-globalatts;               >
```

```
<!ATTLIST munderover %att-accent;
                     %att-accentunder;
                     %att-globalatts;          >
<!ATTLIST mmultiscripts
                     %att-subscriptshift;
                     %att-superscriptshift;
                     %att-globalatts;                >
<!-- presentation layout schemata: script empty elements -->
<!ENTITY % pscreschema "mprescripts|none" >
<!ELEMENT mprescripts       EMPTY                >
<!ATTLIST mprescripts       %att-globalatts;          >
<!ELEMENT none              EMPTY                >
<!ATTLIST none              %att-globalatts;          >
<!-- presentation layout schemata: tables -->
<!ENTITY % ptabschema "mtable|mtr|mtd" >
<!ATTLIST mtable      %att-tableinfo;
                     %att-globalatts;          >
<!ATTLIST mtr          %att-rowalign;
                     %att-columnalign;
                     %att-groupalign;
                     %att-globalatts; >
<!ATTLIST mtd         %att-rowalign;
                     %att-columnalign;
                     %att-groupalign;
                     %att-rowspan;
                     %att-columnspan;
                     %att-globalatts;     >
<!ENTITY % plschema    "%pgenschema;|%pscrschema;|%ptabschema;" >
<!-- empty presentation layout schemata -->
<!ENTITY % peschema "maligngroup | malignmark" >
<!ELEMENT    malignmark    EMPTY                >
<!ATTLIST malignmark       %att-edge;
                           %att-globalatts;          >
<!ELEMENT maligngroup    EMPTY                >
<!ATTLIST maligngroup     %att-groupalign;
                           %att-globalatts;          >
<!-- presentation action schemata -->
<!ENTITY % pactions "maction" >
<!ATTLIST maction     %att-actiontype;
                     %att-selection;
                     %att-globalatts;        >
<!-- Presentation entity for substitution into content tag constructs -->
<!-- excludes elements which are not valid as expressions                    -->
<!ENTITY % PresInCont       "%ptoken; | %petoken; |
                             %plschema; | %peschema; | %pactions;">
<!-- Presentation entity - all presentation constructs -->
```

```
<!ENTITY % Presentation       "%ptoken; | %petoken; | %pscreschema; |
                              %plschema; | %peschema; | %pactions;">
<!-- ********************************************** -->
<!-- Content element set                                  -->
<!-- attribute definitions                                -->
<!ENTITY % att-base           'base CDATA "10"'          >
<!ENTITY % att-closure        'closure CDATA "closed"'   >
<!ENTITY % att-definition     'definitionURL CDATA """   >
<!ENTITY % att-encoding       'encoding CDATA """        >
<!ENTITY % att-nargs          'nargs CDATA "1"'          >
<!ENTITY % att-occurence      'occurence CDATA "function-model"' >
<!ENTITY % att-order          'order CDATA "numeric"'    >
<!ENTITY % att-scope          'scope CDATA "local"'      >
<!ENTITY % att-type           'type CDATA #IMPLIED'      >
<!-- content leaf token elements -->
<!ENTITY % ctoken "ci | cn" >
<!ATTLIST ci     %att-type;
                 %att-globalatts;      >
<!ATTLIST cn     %att-type;
                 %att-base;
                 %att-globalatts;      >
<!-- content elements - specials -->
<!ENTITY % cspecial "apply | reln | lambda" >
<!ATTLIST apply     %att-globalatts;      >
<!ATTLIST reln      %att-globalatts;      >
<!ATTLIST lambda    %att-globalatts;      >
<!-- content elements - others -->
<!ENTITY % cother "condition | declare | sep" >
<!ATTLIST condition          %att-globalatts;      >
<!ATTLIST declare            %att-type;
                             %att-scope;
                             %att-nargs;
                             %att-occurence;
                             %att-definition;
                             %att-globalatts;      >
<!ELEMENT sep          EMPTY >
<!ATTLIST sep          %att-globalatts;      >
<!-- content elements - semantic mapping -->
<!ENTITY % csemantics "semantics | annotation | annotation-xml" >
<!ATTLIST semantics          %att-definition;
                             %att-globalatts;      >
<!ATTLIST annotation         %att-encoding;
                             %att-globalatts;      >
<!ATTLIST annotation-xml     %att-encoding;
                             %att-globalatts;      >
<!-- content elements - constructors -->
```

```
<!ENTITY % cconstructor "interval | list | matrix | matrixrow | set |
vector" >
<!ATTLIST interval          %att-closure;
                            %att-globalatts;        >
<!ATTLIST set               %att-globalatts;        >
<!ATTLIST list              %att-order;
                            %att-globalatts;        >
<!ATTLIST vector            %att-globalatts;        >
<!ATTLIST matrix            %att-globalatts;        >
<!ATTLIST matrixrow         %att-globalatts;        >
<!-- content elements - operators -->
<!ENTITY % cfuncop1ary "inverse | ident " >
<!ELEMENT inverse               EMPTY           >
<!ATTLIST inverse           %att-definition;
                            %att-globalatts;        >
<!ENTITY % cfuncopnary "fn |    compose" >
<!ATTLIST fn                %att-definition;
                            %att-globalatts;        >
<!ELEMENT ident             EMPTY           >
<!ATTLIST ident             %att-definition;
                            %att-globalatts;        >
<!ELEMENT compose           EMPTY           >
<!ATTLIST compose           %att-definition;
                            %att-globalatts;        >
<!ENTITY % carithop1ary "abs | conjugate | exp | factorial" >
<!ELEMENT exp               EMPTY           >
<!ATTLIST exp               %att-definition;
                            %att-globalatts;        >
<!ELEMENT abs               EMPTY           >
<!ATTLIST abs               %att-definition;
                            %att-globalatts;        >
<!ELEMENT conjugate         EMPTY           >
<!ATTLIST conjugate         %att-definition;
                            %att-globalatts;        >
<!ELEMENT factorial         EMPTY           >
<!ATTLIST factorial         %att-definition;
                            %att-globalatts;        >
<!ENTITY % carithop1or2ary "minus" >
<!ELEMENT minus             EMPTY           >
<!ATTLIST minus             %att-definition;
                            %att-globalatts;        >
<!ENTITY % carithop2ary "quotient | divide | power | rem" >
<!ELEMENT quotient          EMPTY           >
<!ATTLIST quotient          %att-definition;
                            %att-globalatts;        >
<!ELEMENT divide            EMPTY           >
```

```
<!ATTLIST divide          %att-definition;
                          %att-globalatts;      >
<!ELEMENT power           EMPTY                 >
<!ATTLIST power           %att-definition;
                          %att-globalatts;      >
<!ELEMENT rem             EMPTY                 >
<!ATTLIST rem             %att-definition;
                          %att-globalatts;      >
<!ENTITY % carithopnary "plus | times | max | min | gcd" >
<!ELEMENT plus            EMPTY                 >
<!ATTLIST plus            %att-definition;
                          %att-globalatts;      >
<!ELEMENT max             EMPTY                 >
<!ATTLIST max             %att-definition;
                          %att-globalatts;      >
<!ELEMENT min             EMPTY                 >
<!ATTLIST min             %att-definition;
                          %att-globalatts;      >
<!ELEMENT times           EMPTY                 >
<!ATTLIST times           %att-definition;
                          %att-globalatts;      >
<!ELEMENT gcd             EMPTY                 >
<!ATTLIST gcd             %att-definition;
                          %att-globalatts;      >
<!ENTITY % carithoproot "root" >
<!ELEMENT root            EMPTY                 >
<!ATTLIST root            %att-definition;
                          %att-globalatts;      >
<!ENTITY % clogicopquant "exists | forall" >
<!ELEMENT exists          EMPTY                 >
<!ATTLIST exists          %att-definition;
                          %att-globalatts;      >
<!ELEMENT forall          EMPTY                 >
<!ATTLIST forall          %att-definition;
                          %att-globalatts;      >
<!ENTITY % clogicopnary "and | or | xor" >
<!ELEMENT and             EMPTY                 >
<!ATTLIST and             %att-definition;
                          %att-globalatts;      >
<!ELEMENT or              EMPTY                 >
<!ATTLIST or              %att-definition;
                          %att-globalatts;      >
<!ELEMENT xor             EMPTY                 >
<!ATTLIST xor             %att-definition;
                          %att-globalatts;      >
<!ENTITY % clogicop1ary "not" >
```

```
<!ELEMENT not                EMPTY          >
<!ATTLIST not                %att-definition;
                             %att-globalatts;       >
<!ENTITY % clogicop2ary "implies" >
<!ELEMENT implies            EMPTY          >
<!ATTLIST implies        %att-definition;
                             %att-globalatts;       >
<!ENTITY % ccalcop "log | int | diff | partialdiff" >
<!ELEMENT log                EMPTY          >
<!ATTLIST log                %att-definition;
                             %att-globalatts;       >
<!ELEMENT int                EMPTY          >
<!ATTLIST int                %att-definition;
                             %att-globalatts;       >
<!ELEMENT diff               EMPTY          >
<!ATTLIST diff               %att-definition;
                             %att-globalatts;       >
<!ELEMENT partialdiff            EMPTY          >
<!ATTLIST partialdiff        %att-definition;
                              %att-globalatts;      >
<!ENTITY % ccalcop1ary "ln" >
<!ELEMENT ln                 EMPTY          >
<!ATTLIST ln                 %att-definition;
                             %att-globalatts;       >
<!ENTITY % csetop2ary "setdiff" >
<!ELEMENT setdiff            EMPTY          >
<!ATTLIST setdiff        %att-definition;
                             %att-globalatts;       >
<!ENTITY % csetopnary "union | intersect" >
<!ELEMENT union              EMPTY          >
<!ATTLIST union              %att-definition;
                             %att-globalatts;       >
<!ELEMENT intersect              EMPTY          >
<!ATTLIST intersect          %att-definition;
                             %att-globalatts;       >
<!ENTITY % cseqop "sum | product | limit" >
<!ELEMENT sum                EMPTY          >
<!ATTLIST sum                %att-definition;
                             %att-globalatts;       >
<!ELEMENT product            EMPTY          >
<!ATTLIST product        %att-definition;
                             %att-globalatts;       >
<!ELEMENT limit              EMPTY          >
<!ATTLIST limit              %att-definition;
                             %att-globalatts;       >
<!ENTITY % ctrigop "sin | cos | tan | sec | csc | cot | sinh
```

```
                    | cosh | tanh | sech | csch | coth
                    | arcsin | arccos | arctan" >
<!ELEMENT sin              EMPTY          >
<!ATTLIST sin              %att-definition;
                           %att-globalatts;       >
<!ELEMENT cos              EMPTY          >
<!ATTLIST cos              %att-definition;
                           %att-globalatts;       >
<!ELEMENT tan              EMPTY          >
<!ATTLIST tan              %att-definition;
                           %att-globalatts;       >
<!ELEMENT sec              EMPTY          >
<!ATTLIST sec              %att-definition;
                           %att-globalatts;       >
<!ELEMENT csc              EMPTY          >
<!ATTLIST csc              %att-definition;
                           %att-globalatts;       >
<!ELEMENT cot              EMPTY          >
<!ATTLIST cot              %att-definition;
                           %att-globalatts;       >
<!ELEMENT sinh             EMPTY          >
<!ATTLIST sinh             %att-definition;
                           %att-globalatts;       >
<!ELEMENT cosh             EMPTY          >
<!ATTLIST cosh             %att-definition;
                           %att-globalatts;       >
<!ELEMENT tanh             EMPTY          >
<!ATTLIST tanh             %att-definition;
                           %att-globalatts;       >
<!ELEMENT sech             EMPTY          >
<!ATTLIST sech             %att-definition;
                           %att-globalatts;       >
<!ELEMENT csch             EMPTY          >
<!ATTLIST csch             %att-definition;
                           %att-globalatts;       >
<!ELEMENT coth             EMPTY          >
<!ATTLIST coth             %att-definition;
                           %att-globalatts;       >
<!ELEMENT arcsin           EMPTY          >
<!ATTLIST arcsin           %att-definition;
                           %att-globalatts;       >
<!ELEMENT arccos           EMPTY          >
<!ATTLIST arccos           %att-definition;
                           %att-globalatts;       >
<!ELEMENT arctan           EMPTY          >
<!ATTLIST arctan           %att-definition;
```

```
                                %att-globalatts;        >
<!ENTITY % cstatopnary "mean | sdev | var | median | mode" >
<!ELEMENT mean                  EMPTY           >
<!ATTLIST mean          %att-definition;
                                %att-globalatts;        >
<!ELEMENT sdev                  EMPTY           >
<!ATTLIST sdev          %att-definition;
                                %att-globalatts;        >
<!ELEMENT var                   EMPTY           >
<!ATTLIST var           %att-definition;
                                %att-globalatts;        >
<!ELEMENT median                EMPTY           >
<!ATTLIST median        %att-definition;
                                %att-globalatts;        >
<!ELEMENT mode                  EMPTY           >
<!ATTLIST mode          %att-definition;
                                %att-globalatts;        >
<!ENTITY % cstatopmoment "moment" >
<!ELEMENT moment                EMPTY           >
<!ATTLIST moment        %att-definition;
                                %att-globalatts;        >
<!ENTITY % clalgop1ary "determinant | transpose" >
<!ELEMENT determinant           EMPTY           >
<!ATTLIST determinant   %att-definition;
                                %att-globalatts;        >
<!ELEMENT transpose             EMPTY           >
<!ATTLIST transpose     %att-definition;
                                %att-globalatts;        >
<!ENTITY % clalgopnary "select" >
<!ELEMENT select                EMPTY           >
<!ATTLIST select        %att-definition;
                                %att-globalatts;        >
<!-- content elements -    relations    -->
<!ENTITY % cgenrel2ary "neq" >
<!ELEMENT neq                   EMPTY           >
<!ATTLIST neq           %att-definition;
                                %att-globalatts;        >
<!ENTITY % cgenrelnary "eq | leq | lt | geq | gt" >
<!ELEMENT eq                    EMPTY           >
<!ATTLIST eq            %att-definition;
                                %att-globalatts;        >
<!ELEMENT gt                    EMPTY           >
<!ATTLIST gt            %att-definition;
                                %att-globalatts;        >
<!ELEMENT lt                    EMPTY           >
<!ATTLIST lt            %att-definition;
```

```
                                %att-globalatts;        >
<!ELEMENT geq                   EMPTY           >
<!ATTLIST geq               %att-definition;
                                %att-globalatts;        >
<!ELEMENT leq                   EMPTY           >
<!ATTLIST leq               %att-definition;
                                %att-globalatts;        >
<!ENTITY % csetrel2ary "in | notin | notsubset | notprsubset" >
<!ELEMENT in                    EMPTY           >
<!ATTLIST in                %att-definition;
                                %att-globalatts;        >
<!ELEMENT notin                 EMPTY           >
<!ATTLIST notin             %att-definition;
                                %att-globalatts;        >
<!ELEMENT notsubset                 EMPTY           >
<!ATTLIST notsubset         %att-definition;
                                %att-globalatts;        >
<!ELEMENT notprsubset               EMPTY           >
<!ATTLIST notprsubset       %att-definition;    %att-globalatts;        >
<!ENTITY % csetrelnary "subset | prsubset" >
<!ELEMENT subset                EMPTY           >
<!ATTLIST subset            %att-definition;
                                %att-globalatts;        >
<!ELEMENT prsubset              EMPTY           >
<!ATTLIST prsubset          %att-definition;
                                %att-globalatts;        >
<!ENTITY % cseqrel2ary "tendsto" >
<!ELEMENT tendsto               EMPTY           >
<!ATTLIST tendsto           %att-definition;
                                %att-type;
                                %att-globalatts;        >
<!ENTITY % cquantifier "lowlimit | uplimit | bvar | degree | logbase" >
<!ATTLIST lowlimit      %att-globalatts;        >
<!ATTLIST uplimit       %att-globalatts;        >
<!ATTLIST bvar          %att-globalatts;        >
<!ATTLIST degree        %att-globalatts;        >
<!ATTLIST logbase       %att-globalatts;        >
<!ENTITY % cop1ary "%cfuncop1ary; | %carithop1ary; | %clogicop1ary;
                            | %ccalcop1ary; | %ctrigop; | %clalgop1ary; " >
<!ENTITY % cop2ary "%carithop2ary; | %clogicop2ary;| %csetop2ary; " >
<!ENTITY % copnary "%cfuncopnary; | %carithopnary; | %clogicopnary;
                            | %csetopnary; | %cstatopnary;    | %clalgopnary; " >
<!ENTITY % copmisc "%carithoproot; | %carithop1or2ary; | %ccalcop;
                            | %cseqop; | %cstatopmoment; | %clogicopquant;" >
<!ENTITY % crel2ary "%cgenrel2ary;    | %csetrel2ary; | %cseqrel2ary;      " >
<!ENTITY % crelnary "%cgenrelnary; | %csetrelnary;" >
```

```
<!ENTITY % Content "%ctoken; | %cspecial;    | %cother;    | %csemantics;
                    |%cconstructor; | %cquantifier;
                    |%cop1ary; |%cop2ary; |%copnary; |%copmisc;
                    |%crel2ary; |%crelnary;" >
<!ENTITY % ContInPres "ci | cn    | apply    | fn | lambda | reln
                    | interval | list | matrix |matrixrow
                    | set | vector" >
<!ENTITY % ContentExpression      "(%Content; | %PresInCont;)* "          >
<!ENTITY % PresExpression         "(%Presentation; | %ContInPres;)* "       >
<!ENTITY % MathExpression         "(%PresInCont; | %ContInPres;)* "        >
<!ELEMENT ci            (#PCDATA | %PresInCont;)*     >
<!ELEMENT cn            (#PCDATA | sep | %PresInCont;)*     >
<!ELEMENT apply              (%ContentExpression;)     >
<!ELEMENT reln               (%ContentExpression;)     >
<!ELEMENT lambda             (%ContentExpression;)     >
<!ELEMENT condition          (%ContentExpression;)     >
<!ELEMENT declare            (%ContentExpression;)     >
<!ELEMENT semantics                  (%ContentExpression;)     >
<!ELEMENT annotation                 (#PCDATA)                 >
<!ELEMENT annotation-xml             (%ContentExpression;)     >
<!ELEMENT interval           (%ContentExpression;)     >
<!ELEMENT set                (%ContentExpression;)     >
<!ELEMENT list               (%ContentExpression;)     >
<!ELEMENT vector             (%ContentExpression;)     >
<!ELEMENT matrix             (%ContentExpression;)     >
<!ELEMENT matrixrow          (%ContentExpression;)     >
<!ELEMENT fn                 (%ContentExpression;)     >
<!ELEMENT lowlimit           (%ContentExpression;)     >
<!ELEMENT uplimit            (%ContentExpression;)     >
<!ELEMENT bvar               (%ContentExpression;)     >
<!ELEMENT degree             (%ContentExpression;)     >
<!ELEMENT logbase            (%ContentExpression;)     >
<!ELEMENT mstyle             (%PresExpression;)                >
<!ELEMENT merror             (%PresExpression;)                >
<!ELEMENT mphantom           (%PresExpression;)                >
<!ELEMENT mrow               (%PresExpression;)                >
<!ELEMENT mfrac              (%PresExpression;)                >
<!ELEMENT msqrt              (%PresExpression;)                >
<!ELEMENT mroot              (%PresExpression;)                >
<!ELEMENT msub               (%PresExpression;)                 >
<!ELEMENT msup               (%PresExpression;)                 >
<!ELEMENT msubsup            (%PresExpression;)                 >
<!ELEMENT mmultiscripts      (%PresExpression;)                 >
<!ELEMENT munder             (%PresExpression;)                 >
<!ELEMENT mover              (%PresExpression;)                 >
<!ELEMENT munderover         (%PresExpression;)                 >
```

```
<!ELEMENT mtable          (%PresExpression;)                    >
<!ELEMENT mtr             (%PresExpression;)                    >
<!ELEMENT mtd             (%PresExpression;)                    >
<!ELEMENT maction         (%PresExpression;)                    >
<!ELEMENT mfenced         (%PresExpression;)                    >
<!ELEMENT mpadded         (%PresExpression;)                    >
<!-- presentation tokens contain PCDATA or malignmark constructs -->
<!ELEMENT mi      (#PCDATA | malignmark )*          >
<!ELEMENT mn      (#PCDATA | malignmark )*          >
<!ELEMENT mo      (#PCDATA | malignmark )*          >
<!ELEMENT mtext   (#PCDATA | malignmark )*          >
<!ELEMENT ms      (#PCDATA | malignmark )*          >
<!ENTITY  %   att-macros      'macros CDATA #IMPLIED' >
<!ENTITY  %   att-mode        'mode   CDATA #IMPLIED' >
<!ENTITY  %   att-topinfo     '%att-globalatts;
                              %att-macros;
                              %att-mode;'      >
<!ENTITY  %   att-name        'name CDATA #IMPLIED' >
<!ENTITY  %   att-height      'height CDATA #IMPLIED' >
<!ENTITY  %   att-width       'width CDATA #IMPLIED' >
<!ENTITY  %   att-baseline    'baseline CDATA #IMPLIED' >
<!ENTITY  %   att-overflow    'overflow
(scroll|elide|truncate|scale) "scroll"' >
<!ENTITY  %   att-altimg      'altimg CDATA #IMPLIED' >
<!ENTITY  %   att-alttext     'alttext CDATA #IMPLIED' >
<!ENTITY  %   att-browif      '%att-type;
                              %att-name;
                              %att-height;
                              %att-width;
                              %att-baseline;
                              %att-overflow;
                              %att-altimg;
                              %att-alttext; '      >
<!ELEMENT math          (%MathExpression;)    >
<!ATTLIST  math           %att-topinfo;
                          %att-browif;    >
<!-- end of DTD fragment -->
```

9.8 小　　结

通过本章的学习，现在我们可以描述 XML DTD 和文档的结构，可以在 XML 文档中调用内部或外部的 DTD 子集；同时，对 DTD 的元素组成和相互关联有了更直观的理解。要想真正掌握更多的 DTD 知识，还需要阅读大量的文献和资料。

第十章　内容与形式的结合——XSL 的应用

可扩展的样式语言 XSL 是一个 XML 应用程序，它提供定义规则的元素来转换和显示 XML 文档，从而实现内容与形式的分离。下面我们将学习 XSL 的具体规范和程序编写，感受 XSL 的巨大威力。

本章包括以下内容：

- XSL 概述
- 理解 XSL
- 构造结果树
- 样式表结构
- 模板规则与模式
- 模板
- 联合样式表
- XSLT 概述
- 对象格式化

10.1　XSL 概述

XSL（eXtensible Stylesheet Language，可扩展样式语言）是为 XML 文档定义的一种标识语言，它将提供远远超过 CSS 的强大功能，如将元素再排序等。实际上简单的 XML 已可被 CSS 所解释，然而复杂的高度结构化的 XML 数据或 XML 文档则只能依赖于 XSL 极强的格式化的能力而展现给用户。XSL 以包含了一套元素集的 XML 语法规范而定义，该语法规范将被用来把 XML 文档转换成 HTML 文档。一个 XSL 样式表集合了一系列设计规则以用于将信息从 XML 文档中提取出来，并将其转换成 HTML 等其它格式。这种转换将采用一种公开的方式，使其更加容易地被程序员描述。而且 XSL 还将提供多种脚本语言的通道以满足更为复杂的应用需求，因此尽管 XSL 是一项新的标识语言，但程序员完全可以继续充分发挥其所熟悉的 HTML 或脚本语言的优势。XSL 凭借其可扩展性能够控制无穷无尽的标签，而控制每个标签的方式也是无穷无尽的。这就给 Web 提供了高级的布局特性。例如旋转的文本、多列和独立区域。它支持国际书写格式，可以在一页上混合使用从左至右、从右至左和从上至下的书写格式。

XSL 能使 Web 浏览器直接根据用户的不同需求改变文档的表示法例如数据的显示顺序改变，从而不需要再与服务器进行交互通信。通过变换样式表，同一个文档可以显示得更大，或者经过叠折只显示外面的一层，或者变为打印格式。可以设想一个适合用户学习特点的技术手册，它为初学者和更高一级的用户提供不同的样式，而所有的样式都是根据同样的文本产生的。

正如 XML 介于 HTML 和 SGML 之间一样，XSL 标准介于 CSS 和 SGML 的 DSSSL

（Document Style Semanticsand Specification Language，文档样式语义和规范语言）之间。DSSSL 定义格式化对象的全特征模式。由于 DSSSL 使用框架语法，而且是很复杂的，所以 DSSSL 未能得到推广应用。XSL 支持 DSSSL 流对象和 CSS 对象，并对复杂的任务提供进入脚本语言的通道，而且允许扩展。实现从 CSS 到 XSL 的映射是可能的，因而内容开发商无需学习这种语言的全部。

作为一种技术展望，微软发布了两种 XSL 处理器：一个是可以从 XML 文档和 XSL 样式层产生 HTML 输出的命令行应用程序，另一个是一种 ActiveX 控件，用于在浏览器中显示 XML。微软的这种 XSL 处理器适合在 Windows 95 和 Windows NT 环境下通过 Internet Explorer 4.0 浏览器使用。

IBM 公司及其 Lotus 子公司发布了 XSL 的原型，这是一个能将 XML 格式转换成 HTML 或其它 Web 格式的转换引擎，现在已可在 www.alphaworks.ibm.com 免费下载。这个转换引擎称为 LotusXSL，基于 WWW 联合会最新的 XSL 工作草案完成。除了能将 XML 文档转换成 HTML 外，XSL 还能将 XML 转换为 PGML（Precision Graphics Markup Language，精确图形描述语言）。如果电子商务中用 XML 表示产品数据，用户可以使用 XSL 定义网站中数据的格式以及信息图形显示方式等。LotusXSL 打包成一个 JavaBean。用户可用 LotusXSL 创建样单，定义转换方式，以可将文档转换为相应的格式，供浏览器显示。

10.2 理解 XSL

10.2.1 结果树

下面我们将看到的最重要的一点，即 XSL 不仅仅是应用样式。当使用 XML 处理器时，XSL 源文档中的信息将被评价、重新安排，然后重新组装。我们最终得到的不只是 XML 数据版本，而且还是可以被容易地添加、修改和重新排序的灵活的源信息。这个最终产品叫做结果树（Result Tree）。

这可以通过一系列测试产生。下面是一个简单的例子：

```
<xsl:template match="recipe_name">
  <P>
    <xsl:process-children/>
  </P>
</xsl:template>
```

最先要解释的是以“/” 结束的标记符是空的。即此种类型的标记符的起始和结束标记符之前什么也不发生。在 HTML 中类似的例子是“<img>”标记符。因为一个图像所需的所有信息都包含在一个标记符中，所以就没有必要存在结束标记符“</img>”。组织良好的 XML 文档可以接受空标记符，同时 XSL 样式表必须是组织良好的 XML。

让我们再回到例子，它告诉 XSL 处理器如果发现一套“<recipe_name>”标记符，就应该分离出内容，然后用“ ”和“ ”包围起来。或者，可以用 XSL 的术语说“添加到结果树中”。这是一个相当简单的测试，而且很典型。XML 元素的内容被表现信息所包围。

10.2.2 一个 XSL 样式表

下面是一个完整的样式表：

```
<xsl:stylesheet>
    <xsl:template match = "/">
      <HTML>
      <BODY>
          <xsl:process-children/>
      </BODY>
      </HTML>
    </xsl:template>
    <xsl:template match = "author">
          <H1>
          <xsl:process-children/>'s fabulous
          </H1>
    </xsl:template>
    <xsl:template match = "recipe_name">
        <H2>
        <xsl:process-children/>
        </H2>
    </xsl:template>
    <xsl:template match = "meal">
        <TABLE><TR><TD><H3>EAT FOR:</H3></TD>
        <TD><H3><xsl:process-children/></H3></TD>
        </TR></TABLE>
    </xsl:template>
    <xsl:template match = "directions">
        <H4>DIRECTIONS</H4>
        <P>
        <xsl:process-children/>
        </P>
    </xsl:template>
    <xsl:template match = "ingredients">
         <B>INGREDIENTS</B><BR></BR>
         <xsl:process-children/>
    </xsl:template>
    <xsl:template match = "item">
         <BR>
        <xsl:process-children/>
        </BR>
    </xsl:template>
</xsl:stylesheet>
```

它是一个能起作用的 XSL。这里可能只有下面的命令需要解释：

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

```
        <xsl:process-children/>
      </BODY>
      </HTML>
  </xsl:template>
```

第一行的“/”告诉处理器这个节点应用到 XML 文档的根上。于是，这部分中的命令是结果树的基础。处理器被告之把<HTML>和<BODY>标记符放在文档的开始和结尾处，然后处理或打印所有的子元素。因为它是根元素，所以意味着“打印所有的东西”。

现在，如果我们再认真考虑一下，就会觉得有点奇怪。如果根层的 process-children 命令把源代码传递给结果树，那么所有与模板匹配的节点都可以与已经经过处理的源码一起工作。

然而，出现的问题是：XSL 有一套确定哪些内容被传递给结果树的规则，其中最重要的规则是：最特定的匹配将会赢。显然，元素名的模板匹配比根层的匹配更特定。因此，所有与模板匹配的节点将超越根层的规则。

注意用 XML 数据添加 HTML 标记符是很容易的。当 XSL 处理器看到那些不在 XSL 词汇表中的标记符时，就会把它们传递给结果树。如果我们花些时间，就可能发现其中巨大的潜力。XSL 可以被用做一种转换语言。存储在一个 XML 文档中的数据可以用完全不同的标记符转换到另一个文档中。还有，信息可以被修改成与可以对应一套不同的标记符集的 XML 应用程序一起工作的形式。

而且还不只这些，如人们所期望的，样式表可以用匹配的标记符打开和关闭，其中是一套组织良好的单元。

10.2.3 选择 XSL

简单的<xsl:template match>还不能完全满足我们的要求。比如，如果希望当<course>标记符出现时取消<meal>标记符的内容。这样的话就不用担心界面上同时显示 dinner 和 appetizer。我们可能还希望通过在最后的 ingredient 后面插入大量的空白来调整版面。

SL 有一套用来把元素与其父成员或子成员匹配的工具。它也允许位置上的匹配。例如，可以在第一个和最后一个某个特定元素上应用特定的格式，等等。

现在，如果网页与数据库没有联系，那么我们就不得不写一个查询语句。如果对 SQL 不太熟，所以必须要得到 DBA 的帮助。但是如果懂得利用 XSL，就可以解决这些问题了。

```
<xsl:style sheet>
   <xsl:template match = "/">
      <xsl:for-each select ="list/recipe">
         <TABLE>
         <TR><TD>
         <xsl:process select = "recipe_name"/></TD>
         <TD>
            <xsl:for-each select = "ingredients/item">
               <BR><xsl:process-children/></BR>
            </xsl:for-each>
         </TD></TR>
         </TABLE>
      </xsl:for-each>
```

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

结果并不神奇。但是在 table 标记符中加些花样将没问题。显然，在 XML 中存储了信息，下面让我们仔细看看这段代码。

```
    <xsl:template match = "/">
        <xsl:for-each select ="list/recipe">
```

第一行很熟悉，只是简单地与模板相匹配。但是第二行却有些不同：在元素清单中出现的每个元素做每件事，直到</xsl:for-each>标记符。

然后开始 HTML 表格，用<xsl:process select="recipe_name"/>标记符把“recipe_name”元素中的内容输出到表格单元中。在关闭第一个表格单元后，事情开始变得有趣了。下一行（<xsl:for-each select="ingredient/item">）开始一个附加的嵌套循环，允许我们把全部“ingredient”输出到合适的显示信息中。样式表的其余部分应该很好理解。

“for-each”函数是 XSL 的几个程序化的特征之一。还有“if-then”和选择函数。这些特征允许任何人都可以以任何能想到的方式（或至少是可行的方式）容易地操纵 XML 内容。

10.3 构造结果树

XSL 是表达样式表的语言，每一个样式表描述了呈现一类 XML 源文档的规则。呈现的过程包括两部分：第一，由源树建立结果树；第二，结果树被解释并在显示器、纸张或语音等其他媒体的格式化形式输出。

第一步，构造结果树,是将模式与模板相结合实现的。模式与源树中的元素相匹配，模板被实例化产生部分结果树。结果树与源树是分离的，结果树的结构可以和源树截然不同。在结果树的构造中，源树可以被过滤和重新排序，还可以增加任意的结构。

第二步，格式化是用该 XSL 文档规定的格式化词表实现结果树的构造。正规来说，这个词表是一个 XML 的名字空间。词汇表中的每一种元素类型对应一个格式化对象类，一种格式化对象类表达一种特定的格式化表现方式。例如，块格式化对象类表示将一段的内容拆成一行一行，词汇表的每个属性对应一种格式化特性。格式化对象类有一特殊的格式化特性集合，这样能够更好地控制格式化对象类的表现方式。例如，在集合各行之前或之后控制行的缩进、行间距。一个格式化对象能拥有内容，而它的格式化表现应用于其内容。

XSL 可以不需要结果树使用格式化词库，这样能够被用作通用的 XML 传输。例如，XSL 能被用来将 XML 转化为结构良好的 HTML，即为采用 HTML 定义的元素类型和属性的 XML。当结果树采用了格式化词库，相遵循的 XSL 实现必须能够根据在该文档中定义的格式化词库的语义解释结果树；它也能将结果树具体化为 XML，但没有必要这样做。

样式表包含了一套模板的规则集合。一个模板规则有两个部分：匹配源树中节点的模式以及实例化后组成部分结果树的模板。它允许一个样式表可用于有类似源树结构的一大类文档。一个模板包含一些元素，它们规定了文字结果的元素结构。一个模板还可以包含作为产生结果树片段的指令元素。当一个模板实例化后，执行每一个指令并置换为其产生的结果树片段。指令能够选择并处理后代的元素。通过查找可应用的模板规则后实例化其

模板，后代的元素处理后产生了结果树片段。元素只有在被执行的指令选中时才作处理。在搜索可用模板规则过程中，不止一个模板规则可能匹配给定元素的模式。然而，仅应用一个模板的规则，决定采用哪一规则的方法在“模板规则的冲突决定”中说明。XSL 用 XML 的名字空间来区别属于 XSL 处理器指令的元素和规定文字结果树结构的元素。指令元素属于 XSL 名字空间。在文档中采用前缀“xsl：”表示 XSL 名字空间中的元素。一个 XSL 样式表包含了一个“xsl：stylesheet”稳当元素。该元素可以包含“xsl：template”元素来规定模板的规则。下面的例子是一个简单的 XSL 样式表，它为包含 emphasis 元素的 para 元素序列构造结果树。“result-ns=“fo””属性表明正在构造使用格式化对象词库的树。“para”元素成为了块格式化对象，字体大小为 10pt，之前的空格为 12pt。

```
<xsl:stylesheet xmlns:xsl=http://www.w3.org/TR/WD-xsl
                xmlns:fo="http://www.w3.org/TR/WD-xsl/FO" result-ns="fo">
<xsl:template match="/">
<fo:page-sequence font-family="serif">
<xsl:process-children/>
</fo:page-sequence>
</xsl:template>
<xsl:template match="para">
<fo:block font-size="10pt" space-before="12pt">
<xsl:process-children/>
</fo:block>
</xsl:template>
</xsl:stylesheet>
```

“xsl：stylesheet”元素也能包含由其他 XSL 的样式表导入的元素、定义宏的元素、定义全局常量的元素、以及识别源属性为个别元素标记的元素。

10.4　样式表结构

在 XML 文档中样式表用元素“xsl：stylesheet”来表示。XSL 处理器处理源文档和样式表时都必须采用 XML 的名字空间机制（W3C XML Names）。所有 XSL 定义的元素（在文档中带有前缀“xsl”）只有是属于 URI 为 http://www.w3.org/TR/WD-xsl 中的某一个名字空间时才会被 XSL 识别；XSL 定义的元素只是在样式表中才认得，而并不是在源文档中。

“xsl：stylesheet”元素有一项可选的属性“result-ns”；它的值需要有一个名字空间前缀。如果规定了这项属性，所有的结果元素必须属于前缀所确定的名字空间。如果有名字空间被确定为缺省名字空间，那么属于该名字空间的结果元素可以用一个空字符串来赋值。“result-ns”属性规定了 XSL 格式化对象的名字空间，那么除了构造 XML 结果树之外，XSL 处理器还根据必须文档中定义的语义来解释它。XSL 格式化对象的名字空间的 URI 为 http://www.w3.org/TR/WD-xsl/FO。例中用前缀“fo：”代表该名字空间。“xsl：stylesheet”元素可以包含以下类型的元素：

xsl:importxsl:includexsl:idxsl:strip-spacexsl:preserve-spacexsl:define-macroxsl:define-attribute-setxsl:define-constantxsl:template。下例表现了一个样式表的结构，省略号表示那里

的属性值或内容可以省略。其中的元素可以在样式表中出现多次或者不出现。

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:import href="..."/>
<xsl:include href="..."/>
<xsl:id attribute="..."/>
<xsl:strip-space element="..."/>
<xsl:preserve-space element="..."/>
<xsl:define-macro name="...">
</xsl:define-macro>
<xsl:define-attribute-set name="...">
</xsl:define-attribute-set>
<xsl:define-constant name="..." value="..."/>
<xsl:template match="...">
</xsl:template>
</xsl:stylesheet>
```

元素出现的顺序没有特殊的规定，除了 xsl：import 元素和错误恢复外，用户可以根据自己的需要自由排序，样式表的产生工具也不需要去控制元素出现的次序。

10.5 模板规则与模式

10.5.1 模板规则

模板规则由 xsl：template 元素来规定。“match”属性标识了规则应用的源节点（集）。例如：一个 XML 文档可能包含下面的内容：

```
This is an <emph>important</emph> point.
```

下列的模板规则匹配 emph 类型的元素，另有一个模板产生一个 fo：sequence 格式化对象，其 font-weight 属性为粗体。

```
<xsl:template match="emph">
<fo:sequence font-weight="bold">
<xsl:process-children/>
</fo:sequence>
</xsl:template>
```

xsl：process-children 元素会递归处理“父”元素的“子”节点。

10.5.2 模式

模式是一个字符串，它用于匹配 XML 源文档中的一个元素。最通常的模式规定了匹配元素的类型名称。 例如，emph 模式匹配类型为 emph 的元素。更复杂的模式规定了相匹配元素的“祖先”节点的元素类型。例如，olist/item 模式匹配类型为 item 并且父元素为 olist 类型的元素。“祖先”序列中的每个元素类型之后可以跟着一系列由逗号分隔的限定词。如，list[attribute（ordered）="yes"]]/item[first-of-type（）]匹配 item 类型的元素，位于相同层次的元素的第一个，而且父节点类型为 list，属性名“ordered”的值为“yes”。

这部分详细描述了模式的语法和语义。匹配元素的模式被称为是匹配模式。 xsl：template 的模式就是匹配模式。一个模式也可以用来选择一系列的节点；这样的模式被称为是选择模式。在一个选择模式中，存在一个当前节点来提供选择的前后关系。该模式会选择相匹配的一列源节点。这些节点是以文档来排序的。xsl：process、xsl：for-each 和 xsl：value-of 中的模式都是选择模式。

10.6 模　　板

当应用于源元素的规则被确定后，就要具体实现该规则的模板。一个模板能包含文字结果的元素，字符数据和产生结果树部分的指令。指令由 XSL 名字空间中的元素来表示，可以选择后代元素来处理。有两类这样的指令，xsl：process-children 和 xsl：process；xsl：process-children 指令处理源元素的邻近子元素，而 xsl：process 指令处理由指定模式来选择的元素。见下例：

```
<xsl:template match="chapter/title">
<fo:rule-graphic/>
<fo:block space-before="2pt">
<xsl:text>Chapter </xsl:text>
<xsl:number/>
<xsl:text>: </xsl:text>
<xsl:process-children/>
</fo:block>
<fo:rule-graphic/>
</xsl:template>
```

10.6.1 文字结果元素

在一个模板中，样式表中不属于 XSL 名字空间的元素具体化将产生相同类型的节点；生成的元素节点会有已经对在模板树中的元素规定的属性。文字结果元素的一个属性的值被认为是一个属性值模板：它能包含在花括号（{}）中的字符串。结果元素节点的名字空间前缀映射是在样式表中移去映射到 XSL 名字空间的 URI 后的映射。因为 XSL 处理器只作用于属于 XSL 名字空间的元素，所以就有这样的问题：如何新建属于 XSL 名字空间的元素？URI 是 http://www.w3.org/TR/WD-xsl 的名字空间如果紧接出现一个或多个的/quote 成为被引用的名字空间时，应用名字空间将作特殊处理。

10.6.2 命名属性集

xsl：define-attribute-set 元素定义了一个自命名的属性集合。name 属性规定了属性集的名称。xsl：define-attribute-set 元素的内容是一个规定属性的 xsl：attribute-set 元素。一个文字结果元素或者一个 xsl：attribute-set 元素能指定一属性集名称为 xsl：use 属性的值。下面的例子产生了一个称为 title-style 的属性集并在模板规则中使用它。

```
<xsl:define-attribute-set name="title-style">
<xsl:attribute-set font-size="12pt" font-weight="bold"/>
</xsl:define-attribute-set>
```

```
<xsl:template match="chapter/heading">
<fo:block xsl:use="title-style" quadding="start">
<xsl:process-children/>
</fo:block>
</xsl:template>
```

10.6.3 模板中的文字

模板也能包含 PCDATA（Parsed Character Data）。在模板中去除空格后的每个数据字符将在结果树中产生一个数据字符。文字的数据字符也可以包装在一个 xsl：text 元素中。这样的包装处理可能改变空格的去除但不影响 XSL 处理器对字符的处理。

10.6.4 xsl:process-children 的处理

下例新建用于 chapter 元素的块并处理它的相邻子元素。

```
<xsl:template match="chapter">
<fo:block>
<xsl:process-children/>
</fo:block>
</xsl:template>
```

xsl：process-children 指令处理当前节点的所有子节点，包括字符。处理源树中的字符是将字符添加到结果树。因此，其中的“&lt；”标记在源树中表示“<”字符，该源树将由内置的模板规则在结果树中转换为“<”字符，而当结果树具体化为一个 XML 文档时，“<”字符又将表示为“&lt；”。

10.6.5 xsl：process 的处理

xsl：process 元素处理由一个模式选择的元素。xsl：process 元素的模式是一个选择模式，因而它被间接地定位到当前节点。下面的例子对 author-group 的所有 author 子节点进行处理：

```
<xsl:template match="author-group">
<fo:sequence>
<xsl:process select="author"/>
</fo:sequence>
</xsl:template>
```

xsl：process 元素处理所有匹配规定模式的元素。字符数据不被 xsl：process 元素匹配。模式不能包含属性模式，除非它作为属性限定的一部分。模式控制了发生匹配的深度。下例处理所有 author 节点中 first-name 元素：

```
<xsl:template match="author-group">
<fo:sequence>
<xsl:process select="author/first-name"/>
</fo:sequence>
</xsl:template>
```

在模式中使用“//”操作符可以匹配任意的深度。下例处理在 book 元素中的所有 heading 元素。

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

```
<fo:block>
<xsl:process select=".//heading"/>
</fo:block>
</xsl:template>
```

10.6.6 直接处理

当结果是已知的规则结构，能够直接确定选择元素的模板是很有益的。xsl：for-each 元素包括一个模板，它具体实现由 select 属性规定的每个选择元素。比如对下面的 XML 文档：

```
<customers>
<customer>
<name>...</name>
<order>...</order>
<order>...</order>
</customer>
<customer>
<name>...</name>
<order>...</order>
<order>...</order>
</customer>
</customers>
```

下面的 XSL 将生成一个 HTML 文档，包括一个表格，其中的一行就为一个 custom 元素：

```
<xsl:template match="/">
<HTML>
<HEAD>
<TITLE>Customers</TITLE>
</HEAD>
<BODY>
<TABLE>
<TBODY>
<xsl:for-each select="customers/customer">
<TR>
<TH>
<xsl:process select="name"/>
</TH>
<xsl:for-each select="order">
<TD>
<xsl:process-children/>
</TD>
</xsl:for-each>
</TR>
</xsl:for-each>
</TBODY>
</TABLE>
```

```
</BODY>
</HTML>
</xsl:template>
```

10.6.7 模板中的条件

XSL 中有两个指令来支持条件处理：xsl：if 和 xsl：choose。xsl：if 指令提供简单的 if-then 的条件选择；xsl：choose 支持多条件的选择。

10.6.8 计算产生的文本

在模板中，xsl：value-of 元素能用于计算产生的文本，比如通过从源树中提取文本或插入字符常数的值。它由 xsl：value-of 元素通过一个规定为 expr 属性值的字串表达式来实现。字串表达式也能在文字结果元素的属性值中使用，只要将该字串表达式套入{}中。

10.6.9 宏

宏能产生结果集合还能被引用，就像一个单独的对象。在下例中，为一封装的段落定义了一个宏，在其内容之前增加“Warning!”语句。在匹配 warning 元素的规则中该宏被引用。

```
<xsl:define-macro name="warning-para">
<fo:box>
<fo:block>
<xsl:text>Warning! </xsl:text>
<xsl:contents/>
</fo:block>
</fo:box>
</xsl:define-macro>
<xsl:template match="warning">
<xsl:invoke macro="warning-para">
<xsl:process-children/>
</xsl-invoke>
</xsl:template>
```

10.7 联合样式表

XSL 提供两种机制来联合样式表：

- 样式表导入，允许样式表之间相互引用。
- 样式表包含，允许样式表被原文组合。

10.7.1 样式表导入

一个 XSL 样式表可以包含 xsl：import 元素。所有 xsl：import 元素必须出现在样式表的开头。xsl：import 元素有一个 href 属性，它的值就表示要导入的样式表的 URI。相对 URI 是指相对于 xsl：import 元素的基 URI。

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

```
<xsl:import href="article.xsl"/>
<xsl:import href="bigfont.xsl"/>
<xsl:define-attribute-set name="note-style">
<xsl:attribute-set font-posture="italic"/>
</xsl:define-attribute-set>
</xsl:stylesheet>
```

导向的样式表中的规则和定义比任何被导入样式表中的规则和定义都要重要。同样，一个被导入的样式表中的规则和定义比之前导入的样式表中的规则和定义都要重要。一般来说，更重要的规则或定义比次重要的规则或定义要优先。每一类的规则和定义都会详细规定它。

10.7.2 样式表包含

一个样式表中可以用 xsl：include 元素来包含另一个 XSL 样式表。xsl：include 也有 href 属性，它的值就表示被包含的样式表的 URI。相对 URI 是指相对于 xsl：include 元素的基 URI。xsl：include 元素可以作为 xsl：stylesheet 元素的子元素，出现在任何 xsl：import 之后。若 XML 树的层次在上，则该包含生效。由 href 属性值定位的资源内容作为一个 XML 文档解析，在该文档中的 xsl：stylesheet 元素的子元素替代包含文档的 xsl：include 元素。同时在被包含的文档的 xsl：import 元素在包含文档中移上至任一存在的 xsl：import 元素之后。不像 xsl：import，被包含的规则或定义不影响他们被处理的方式。

10.7.3 嵌入样式表

通常一个样式表就是一个完整的 XML 文档，xsl：stylesheet 元素作为文档的元素。然而一个 XSL 样式表也可以嵌入在其它文档内容之中。内嵌的方式可能有两种：XSL 样式表可以原文嵌入在一个非 XML 文档中或者 xsl：stylesheet 不作为文档元素出现在一个 XML 文档中。在第二种情况增加了出现内嵌样式，即自己规定样式的文档的可能。XSL 还没有为之定义相应的机制。这是由于可以采用把样式表结合文档的通用方式来实现，只要满足：

该方式允许一部分内容可以规定为样式表，例如使用有片段标识符 URI。

该方式本身能被嵌入在文档中，比如作为一个处理指令。定义这样的方式不在 XSL 的范围之内。

下例表明了怎样用 xml：stylesheet 处理指令将样式表和文档结合来实现内嵌样式。其中的 URI 在片段标识符中使用了一个 Xpointer 来确定 xsl：stylesheet 元素的位置。

```
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="#id(style1)"?>
  <!DOCTYPE doc SYSTEM "doc.dtd">
<doc>
<head>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl" id="style1">
<xsl:import href="doc.xsl"/>
<xsl:template match="id(foo)">
  <fo:block font-weight="bold"><xsl:process-children/></fo:block>
</xsl:template>
</xsl:stylesheet>
```

```
</head>
<body>
<para id="foo">
</para>
</body>
</doc>
```

10.8 XSLT 概述

XSLT 也可以脱离 XSL 使用。但是，XSLT 并不是一种完全通用的 XML 转化语言。它主要被用来转化 XSL 需要的信息。

10.8.1 为什么需要 XSLT

XML 文档具有树型的嵌套结构。但是信息本身可以有多种组织结构。一个开发者选择的结构可能对其它开发者并不适用，面向一个应用设计的结构可能不适合其它应用。有时，还需要向不同的用户呈现不同的重点，或者对某些用户隐藏掉一些信息。

另外，开发者设计信息结构时，有时考虑不很周全，以至设计出的结构并不是最优的。当开发者集中精力考虑如何展现数据时，很可能会顾此失彼，不能很好地建立数据。

引入 XSLT 就是为了给开发者提供一个良好的工具，解决以上问题。

10.8.2 XSLT 的特点

使用 XSLT，开发者可以描述一种从现有的 XML 文档建立新的结构化文档的方法，由 XSLT 引擎来实现。

XSLT 并不是一种编程语言。XSLT 的实现方法是给出实例，而不是描述执行过程。开发者要做的是把模板提供给引擎，并指明在进行转换过程中何时何地使用模板。在模板中可以加入指令，告诉引擎从一个或多个输入文档中自动搜索信息，并插入模板中的空位。

假如使用 XSL 来展现信息，那么只要通过上述方法，把信息转化为由 XSL 词汇组成的树状结构。展现引擎键会自动根据他们的语义来展现信息。这大大简化了应用开发的难度。

使用 XSLT 的应用是很灵活的。用户可以打乱信息的出现次序，挑选有用的信息，甚至是同样信息在一个结果中出现多次。生成的结果可以是 XML、包括 CSS 的 HTML、不包括 CSS 的 HTML 或简单文本。

XSLT 的用途也很广泛。用户可以使用自己的 XML 信息来为他的同事或用户合成新的数据实例；可以直接从数据源生成 HTML/CSS 网页；可以将数据以简单文本方式传递给其它系统；可以建立操作系统的 Script 语言。

也就是说，用户可以在开始时按自己喜欢的方式来组织数据，以最好的方式实现业务目标；同时，却仍然可以用无数种方式来应用这些数据。这本身就是 XML 的目标，现在 XSLT 使它成为现实。

10.8.3 W3C 提供的非标准化实例

1. 关于文档

这个例子是把符合一个简单的 DTD 的文档转化为 XHTML 的样式表。

DTD 是：

```
<!ELEMENT doc (title, chapter*)>
<!ELEMENT chapter (title, (para|note)*, section*)>
<!ELEMENT section (title, (para|note)*)>
<!ELEMENT title (#PCDATA|emph)*>
<!ELEMENT para (#PCDATA|emph)*>
<!ELEMENT note (#PCDATA|emph)*>
<!ELEMENT emph (#PCDATA|emph)*>
```

样式表是：

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/TR/xhtml1/strict">
<xsl:strip-space elements="doc chapter section"/>
<xsl:output
method="xml"
indent="yes"
encoding="iso-8859-1"
/>
<xsl:template match="doc">
<html>
<head>
<title>
<xsl:value-of select="title"/>
</title>
</head>
<body>
<xsl:apply-templates/>
</body>
</html>
</xsl:template>
<xsl:template match="doc/title">
<h1>
<xsl:apply-templates/>
</h1>
</xsl:template>
<xsl:template match="chapter/title">
<h2>
<xsl:apply-templates/>
</h2>
</xsl:template>
<xsl:template match="section/title">
```

```
<h3>
<xsl:apply-templates/>
</h3>
</xsl:template>
<xsl:template match="para">
<p>
<xsl:apply-templates/>
</p>
</xsl:template>
<xsl:template match="note">
<p class="note">
<b>NOTE: </b>
<xsl:apply-templates/>
</p>
</xsl:template>
<xsl:template match="emph">
<em>
<xsl:apply-templates/>
</em>
</xsl:template>
</xsl:stylesheet>
```

假如输入文档如下：

```
<!DOCTYPE doc SYSTEM "doc.dtd">
<doc>
<title>Document Title</title>
<chapter>
<title>Chapter Title</title>
<section>
<title>Section Title</title>
<para>This is a test.</para>
<note>This is a note.</note>
</section>
<section>
<title>Another Section Title</title>
<para>This is <emph>another</emph> test.</para>
<note>This is another note.</note>
</section>
</chapter>
</doc>
```

XHTML 结果将是这样：

```
<?xml version="1.0" encoding="iso-8859-1"?>
<html xmlns="http://www.w3.org/TR/xhtml1/strict">
<head>
<title>Document Title</title>
</head>
<body>
```

```
<h1>Document Title</h1>
<h2>Chapter Title</h2>
<h3>Section Title</h3>
<p>This is a test.</p>
<p class="note">
<b>NOTE: </b>This is a note.</p>
<h3>Another Section Title</h3>
<p>This is <em>another</em> test.</p>
<p class="note">
<b>NOTE: </b>This is another note.</p>
</body>
</html>
```

2. 关于数据

在这个例子中，三个不同的 XSLT 样式表从同一个 XML 中提取数据，生成三种不同的表现：HTML、SVG 和 VRML。

输入数据是：

```
<sales>
<division id="North">
<revenue>10</revenue>
<growth>9</growth>
<bonus>7</bonus>
</division>
<division id="South">
<revenue>4</revenue>
<growth>3</growth>
<bonus>4</bonus>
</division>
<division id="West">
<revenue>6</revenue>
<growth>-1.5</growth>
<bonus>2</bonus>
</division>
</sales>
```

这个样式表把数据转化为 HTML：

```
<html xsl:version="1.0" xmlns:xsl=http://www.w3.org/1999/XSL/Transform    lang="en">
<head>
<title>Sales Results By Division</title>
</head>
<body>
<table border="1">
<tr>
<th>Division</th>
<th>Revenue</th>
<th>Growth</th>
```

```
<th>Bonus</th>
</tr>
<xsl:for-each select="sales/division">
<!-- order the result by revenue -->
<xsl:sort select="revenue" data-type="number" order="descending"/>
<tr>
<td>
<em><xsl:value-of select="@id"/></em>
</td>
<td>
<xsl:value-of select="revenue"/>
</td>
<td>
<!-- highlight negative growth in red -->
<xsl:if test="growth &lt; 0">
<xsl:attribute name="style">
<xsl:text>color:red</xsl:text>
</xsl:attribute>
</xsl:if>
<xsl:value-of select="growth"/>
</td>
<td>
<xsl:value-of select="bonus"/>
</td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
```

输出的 HTML 是：

```
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Sales Results By Division</title>
</head>
<body>
<table border="1">
<tr>
<th>Division</th><th>Revenue</th><th>Growth</th><th>Bonus</th>
</tr>
<tr>
<td><em>North</em></td><td>10</td><td>9</td><td>7</td>
</tr>
<tr>
<td><em>West</em></td><td>6</td><td style="color:red">-1.5</td><td>2</td>
</tr>
```

```
<tr>
<td><em>South</em></td><td>4</td><td>3</td><td>4</td>
</tr>
</table>
</body>
</html>
```

这个样式表把数据转化为 SVG：

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/Graphics/SVG/SVG-19990812.dtd">
<xsl:output method="xml" indent="yes" media-type="image/svg"/>
<xsl:template match="/">
<svg width = "3in" height="3in">
<g style = "stroke: #000000">
<!-- draw the axes -->
<line x1="0" x2="150" y1="150" y2="150"/>
<line x1="0" x2="0" y1="0" y2="150"/>
<text x="0" y="10">Revenue</text>
<text x="150" y="165">Division</text>
<xsl:for-each select="sales/division">
<!-- define some useful variables -->
<!-- the bar's x position -->
<xsl:variable name="pos"
select="(position()*40)-30"/>
<!-- the bar's height -->
<xsl:variable name="height"
select="revenue*10"/>
<!-- the rectangle -->
<rect x="{$pos}" y="{150-$height}"
width="20" height="{$height}"/>
<!-- the text label -->
<text x="{$pos}" y="165">
<xsl:value-of select="@id"/>
</text>
<!-- the bar value -->
<text x="{$pos}" y="{135-$height}">
<xsl:value-of select="revenue"/>
</text>
</xsl:for-each>
</g>
</svg>
</xsl:template>
</xsl:stylesheet>
```

输出的 SVG 是：

```
<svg width="3in" height="3in"
xmlns="http://www.w3.org/Graphics/SVG/svg-19990412.dtd">
```

```
<g style="stroke: #000000">
<line x1="0" x2="150" y1="150" y2="150"/>
<line x1="0" x2="0" y1="0" y2="150"/>
<text x="0" y="10">Revenue</text>
<text x="150" y="165">Division</text>
<rect x="10" y="50" width="20" height="100"/>
<text x="10" y="165">North</text>
<text x="10" y="45">10</text>
<rect x="50" y="110" width="20" height="40"/>
<text x="50" y="165">South</text>
<text x="50" y="105">4</text>
<rect x="90" y="90" width="20" height="60"/>
<text x="90" y="165">West</text>
<text x="90" y="85">6</text>
</g>
</svg>
```

这个样式表把数据转化为 VRML：

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!-- generate text output as mime type model/vrml, using default charset -->
<xsl:output method="text" encoding="UTF-8" media-type="model/vrml"/>
<xsl:template match="/">#VRML V2.0 utf8
# externproto definition of a single bar element
EXTERNPROTO bar
field SFInt32 x
field SFInt32 y
field SFInt32 z
field SFString name
]
"http://www.vrml.org/WorkingGroups/dbwork/barProto.wrl"
# inline containing the graph axes
Inline
url "http://www.vrml.org/WorkingGroups/dbwork/barAxes.wrl"
}
<xsl:for-each select="sales/division">
bar {
x <xsl:value-of select="revenue"/>
y <xsl:value-of select="growth"/>
z <xsl:value-of select="bonus"/>
name "<xsl:value-of select="@id"/>"
}
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

输出的 VRML 是：

```
#VRML V2.0 utf8
```

```
# externproto definition of a single bar element
EXTERNPROTO bar
field SFInt32 x
field SFInt32 y
field SFInt32 z
field SFString name
]
"http://www.vrml.org/WorkingGroups/dbwork/barProto.wrl"
# inline containing the graph axes
Inline
url "http://www.vrml.org/WorkingGroups/dbwork/barAxes.wrl"
}
bar {
x 10
y 9
z 7
name "North"
}
bar {
x 4
y 3
z 4
name "South"
}
bar {
x 6
y -1.5
z 2
name "West"
}
```

10.9 对象格式化

10.9.1 简介

在构造结果树的时候可以采用格式化词库中定义的对象来生成结果。我们已经在 XSL 概述中了解了格式化对象的基本概念，通常格式化词库是一个 XML 的名字空间。其中的每一个元素的类型对应一类格式化对象。最新的 XSL 草案已经定义了一些这样的格式化对象。虽然它还在完善之中，但我们不妨把它的基础和精髓介绍给大家。

10.9.2 格式化对象及其属性

表明为 non-core 的对象不必在 XSL 中实现。已经定义的格式化对象有下列这些：

1. 布局格式化对象（Layout Formatting Objects）

- page-sequence：提供这样的机制，可以定义主序列(sequences)，然后将内容和这些主序列相结合。
- simple-page-master：描述网页的普通布局或布局序列（打印或在线状态）。

2. 内容流对象（Content Flow Objects）

- queue：集中在页序列中被替代的内容。
- sequence：将内容分组并允许分配共享继承属性。
- list：将所有项组成一个列表。
- list-item：为每个列表中项目组合其列表项标签和主体。
- list-item-label：保存一个列表项的数字或标签。
- list-item-body：保存列表项的主体内容。
- block：用以表示段落、标题、说明等。
- character：格式化程序的原子单位，在需要明确重载有具体表现字体的字符或字符串时使用它。
- rule-graphic：可用于绘制一图形线将网页划分为几个区域。
- graphic：保存一个图片或矢量图像，在 XSL 可能替代为行内或块层次。graphic 的内容可能是内部流或外部的连接。
- score：突出的文本。产生下划线、横透线、上标线等。
- Boxes：设置底色和边框。
- Building Blocks：指示格式化程序在内容流的当前位置构造文本对象。
- page-number：使得格式化程序产生页码。
- link：网页浏览器链接。
- link-end-locator：链接的目标或对象。

10.10　XSL 完整实例

下面是一个简单但完整的样式表例子：

```
<?xml version='1.0'?>
  <xsl:stylesheet xmlns:xsl=http://www.w3.org/TR/WD-xsl
xmlns:fo="http://www.w3.org/TR/WD-xsl/FO" result-ns="fo" indent-result="yes">
<xsl:template match='/'>
<fo:page-sequence font-family="serif">
<fo:simple-page-master name='scrolling'/>
<fo:queue queue-name='body'>
<xsl:process-children/>
</fo:queue>
</fo:page-sequence>
</xsl:template>
<xsl:template match="title">
```

```
<fo:block font-weight="bold">
<xsl:process-children/>
</fo:block>
</xsl:template>
<xsl:template match="p">
<fo:block>
<xsl:process-children/>
</fo:block>
</xsl:template>
<xsl:template match="emph">
<fo:sequence font-style="italic">
<xsl:process-children/>
</fo:sequence>
</xsl:template>
</xsl:stylesheet>
```

XML 的源文档是：

```
<doc>
<title>An example</title>
<p>This is a test.</p>
<p>This is <emph>another</emph> test.</p>
</doc>
```

它将生成下面的结果：

```
<fo:page-sequence xmlns:fo="http://www.w3.org/TR/WD-xsl/FO" font-family="serif">
<fo:simple-page-master name="scrolling"/>
<fo:queue queue-name="body">
<fo:block font-weight="bold">An example</fo:block>
<fo:block>This is a test.</fo:block>
<fo:block>This is <fo:sequence font-style="italic">another</fo:sequence> test.</fo:block>
</fo:queue>
</fo:page-sequence>
```

10.11 小　　结

XSL 作为一种正在发展中的样式机制，将随着 XML 的发展逐渐使 Web 文档的显示产生变革。XSL 最为突出的特性就是可以将用某种标记语言描述的文档转换成另外一种标记语言描述的文档，或者是得到个性化的文档显示设置。

第十一章　XML DOM 技术

DOM（文档对象模型）是 HTML 和 XML 文档的编程基础，它定义了处理执行文档的途径。程序员可以使用 DOM 增加文档、定位文档结构、填加修改删除文档元素。W3C 的重要目标是把利用 DOM 提供一个使用于多个平台的编程接口。W3C DOM 被设计成适合多个平台，可使用任意编程语言实现的方法。

本章包括以下内容：

- DOM 规范简单介绍
- DOM 的核心结构
- 节点接口
- 使用 XML 解析器
- 装载一个 XML 文档到解析器中
- XML 错误
- ParseError 对象与属性
- 节点树
- 装载 XML 进入解析器
- 遍历 XML 节点树

11.1　DOM 规范简单介绍

1998 年 10 月，W3C 发布了 DOM 的正式规范，可以在 http：//www.w3.org/TR/REC-DOM-Level-1/ 查阅该规范。下一代 DOM，W3C 工作草案可以在 http://www.w3.org/TR/REC-DOM-Level-2/查阅，但是尚未成熟，不要以它为基础进行开发。DOM Level-1 是一个稳定的文档，可用于生产环境。它描述了一组节点界面，使得脚本语言和其他编程语言能够存取文档的节点。它描述了几种节点类型，按节点的不同，能以不同的方式存取它们。下面描述不同节点类型及它们的一些性质和方法。

11.2　DOM 的核心结构

11.2.1　节点

这是基础类，所有节点都由它导出（NodeList 和 NamedNodeMap 节点除外）。该节点用于描述以类称为节点的对象，所有节点都属于此类。该节点包括若干重要性质。每一节点类型都继承这些性质（NodeList 和 NamedNodeMap 节点除外）：

- nodename：节点的名字。

- nodevalue：节点值。
- attributes：属性集合。
- parentNode：给定节点的父节点。
- firstChild：子节点集合中第一个子节点。
- lastChild：子节点集合中最后一个字节点。
- childNode：包含一个节点的所有子节点的集合。
- previousSibling：该特性表示的是在文档次序中，给定节点在另一与其共有父节点的节点之后出现时，前者相对于后者在节点表中的位置。
- nextSibling：该特性表示的是在文档次序中，给定节点在另一与其共有父节点的节点之前出现时，前者相对于后者在节点表中的位置。
- owerDocument：代表 DOM 树结构内的文档实例。
- nodeType：返回无符号整型，代表节点类型。
- InsertBefore（newChild，oldChild）：在现有子节点之前插入一个新节点的方法，子节点必须确实存在，同时新节点的插入不会导致非结构良好 XML 标记。
- ReplaceChild（newChild ，oldChild）：以另一节点替代一个节点表的子节点表中的一个旧节点的方法。
- RemoveChild（oldChild）：删除一个节点表的子节点表中的一个旧节点的方法。
- AppendChild（newChild）：将一个字节点插入一个子节点表的方法。
- HasChildNode（）：如果一个节点有子节点则返回“真”，否则返回“假”。
- CloneNode（）：生成给定节点的一个拷贝的方法。

11.2.2 字符数据

该节点类不能直接访问，而是作为文本、CDATA 段和内容节点的性质和方法的基础类。它的性质包括：

- Data：描述包含在任何能够从字符数据类继承属性的节点的文本。
- Length：给定文本串的字符个数。
- SubstringData （ offsetIntegerValue ， CountIntegerValue ）：该方法计算由 offsetIntegerValue 开始到 CountIntegerValue 结束的字符个数。
- AppendData（appendstringValue）：该方法将一个字符串添加到节点的现有字符串中。
- InsertData（offsetUnsignedLongInteger，newStringValue）：该方法在给定索引位置将新字符串插入文本的一个现有字符串中。
- DeleteData（offsetValue， CountValue）：该方法从字符串中删除数据。
- ReplaceData（offsetValue ，CountValue，dataValue）：该方法替换字符串数据。

11.2.3 文档

文档节点是一个 XML 文档的实例。由文档节点可以创建其他节点并可使用一个程序将它们插入树中。该节点的主要属性是：DocumentElement，它代表文档实例。该节点可用的最重要的方法是 createElement（）和 createDocumentFragment（）方法。createElement（）

方法创建一个元素节点，它有个参数，即应给予该元素的元素标注名。createDocumentFragment()方法创建一个文档段节点。

11.2.4 元素

该节点表示一个元素。与其它节点一样，它继承节点类的所有性质和方法。它仅有的特性是 tagName 属性，这是一个描述元素的标注名的字符串。该节点包括下列方法：getAttribute（name ）、setAttribute（name value）、removeAttribute（name）、getAttributeNode（ name ） 、 removeAttributeNode （ oldAttribute ） 、 setAttributeNode （ newAttribute ） 和 getElementsByTagName（name）。

11.2.5 其他节点

DocumentType

当前，该节点代表 DTD 的 ENTITY 和 NOTATION 说明。DOM 规范的未来版本将会扩展该节点。

NodeList

该节点是子节点的集合，包含关于其子节点的个数的信息。

NamedNodMap

该节点是包含属性节点的信息的字节点集合。

Entity

该节点由 DTD 的 ENTITY 说明导出，它的属性依次产生属性节点。

EntityReference

该节点是文档实例中引用的实体的结果。

Notation

该节点是 DTD 中 Notation 说明的结果。

Text

该节点是包含在元素中的字符数据。它继承字符数据节点类的方法。

Attribute

该节点代表元素中或实体或记号节点说明中定义的属性。大多数与属性节点相关的操作实际上都由元素节点的方法完成。该节点包括：Name Property，该字符串表示给定属性的名字；Value Property，表示节点的值；Specified property，如果属性装入 xml 分析器之前在文档中出现，其值为真。

CDATASection

该节点代表 CDATA 数据段。

ProcessingInstruction

该节点代表处理指令。

Comment

该节点代表注释。

Document Fragment

该节点表示一个文档实例的部分树，它可以用于插入附加节点。其中每个节点都可通过所有节点共有的 nodeType 属性访问。

11.3 节 点 接 口

XML 解析器用来装载 XML 文档到缓存中。文档装载时，可以使用 DOM 进行检索和处理。DOM 采用树形结构表示 XML 文档，文档元素是树的最高阶层，该元素有一个或多个子节点用来表示树的分枝。

节点接口程序通常用来读和写 XML 节点树中的个别元素，文档元素的子节点属性可以用来构造个别元素节点。XML 解析器用来证明 Web 中的 DOM 支持遍历节点树的所有函数，并可通过它们访问节点及其属性、插入删除节点、转换节点树到 XML 中。

所有 Microsoft XML 解析器函数得到 W3C XML DOM 的正式推荐，除了 load 和 loadXML 函数（正式的 DOM 不包括标准函数 loading XML 文档）。有 13 个节点类型被 Microsoft XML parser 支持，下面列出几个常用节点：

节点类型	例子
Document type	<!DOCTYPE food SYSTEM "food.dtd">
Processing instruction	<?xml version="1.0"?>
Element	<drink type="beer">Carlsberg</drink>
Attribute	type="beer"
Text	Carlsberg

11.4 使用 XML 解析器

为了更加熟练地处理 XML 文档，必须使用 XML 解析器。Microsoft XML 解析器是 IIS 5.0 所带的一个 COM 组件，一旦安装了 IIS 5.0，解析器可以利用 HTML 文档和 ASP 文档中的脚本。

Microsoft XMLDOM 解析器支持以下编程模式：

- 支持 JavaScript、VBScript、Perl、VB、Java、C++等等。
- 支持 W3C XML1.0 和 XML DOM。
- 支持 DTD 和 validation。

如果使用 IE 5.0 中的 JavaScript，可以使用下面的 XML 文档对象：

```
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")
```

如果使用 VBScript，可以使用下面的 XML 文档对象：

```
set xmlDoc = CreateObject("Microsoft.XMLDOM")
```

如果使用 ASP，可以使用下面的 XML 文档对象：

```
set xmlDoc = Server.CreateObject("Microsoft.XMLDOM")
```

11.5　装载一个 XML 文档到解析器中

下面的代码装载存在的 XML 文档进入 XML 解析器：

```
<script language="JavaScript">
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")
xmlDoc.async="false"
xmlDoc.load("note.xml")
// ..... processing the document goes here
</script>
```

第一行脚本增加了一个 Microsoft XML 解析器实例，第三行装载名为“note.xml”的 XML 文档进入解析器中。第二行保证文档装载完成以后解析器进行下一步工作。

11.6　XML 错误

下面是使用不正确的格式装载 XML 文档的实例：

```
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")
xmlDoc.async="false"
xmlDoc.load("note_error.xml")

document.write("<br>Error Code: ")
document.write(xmlDoc.parseError.errorCode)
document.write("<br>Error Reason: ")
document.write(xmlDoc.parseError.reason)
document.write("<br>Error Line: ")
document.write(xmlDoc.parseError.line)
```

11.7　ParseError 对象与属性

打开 XML 文档时，XML 解析器产生错误代码，并存在 parseError 对象中，包括错误代码、错误文本和错误行号等信息。

parseError 属性描述：

- errorCode：返回长整型错误代码。
- Reason：返回字符串型错误原因 Line：返回长整型错误行号。
- LinePos：返回长整型错误行号位置。
- SrcText：返回字符串型产生错误原因。
- url：返回 url 装载文档指针。
- filePos：返回长整型错误文档位置。

11.8 节 点 树

一种通用的析取 XML 文档的方法是遍历节点树和它的元素值。下面是使用 VBScript 写的遍历节点树的程序代码：

```
set xmlDoc=CreateObject("Microsoft.XMLDOM")
xmlDoc.async="false"
xmlDoc.load("note.xml")

for each x in xmlDoc.documentElement.childNodes
  document.write(x.nodename)
  document.write(": ")
  document.write(x.text)
next
```

11.9 装载 XML 进入解析器

```
<html>
<body>
<script language="javascript">
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")
xmlDoc.async="false"
xmlDoc.load("note.xml")
document.write("The first XML element in the file contains: ")
document.write(xmlDoc.documentElement.childNodes.item(0).text)
</script>
</body>
</html>
```

11.10 遍历 XML 节点树

```
<html>
<body>
<script language="VBScript">
```

```
txt="<h1>Traversing the node tree</h1>"
document.write(txt)
set xmlDoc=CreateObject("Microsoft.XMLDOM")
xmlDoc.async="false"
xmlDoc.load("note.xml")
for each x in xmlDoc.documentElement.childNodes
   document.write("<b>" & x.nodename & "</b>")
   document.write(": ")
   document.write(x.text)
   document.write("<br>")
next
</script>
</body>
</html>
```

11.11 小　　结

在本章中，我们对 DOM 从总体框架上有了基本的了解，掌握了关于 DOM 的一些知识，以及如何在程序中实际地去应用 DOM。利用 DOM，我们可以实现动态创建文档，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强大的功能。

第十二章　同步多媒体合成语言 SMIL

在基于文本的 HTML 的 Web 页面出现不久，开发者们开始寻找一种能在 Web 页面中包含更丰富媒体的方法——包括音频和视频。由于 Web 出现以前，存在多种多媒体的文档格式，而且存在不同文档格式的播放器，不同的播放器播放相应的文档格式的多媒体数据。当 Web 成为一种重要的信息发布工具时，为了在 Web 上发布多媒体内容的信息，各开发商制造了各种相应的插件。但不同的插件只能播放相应文档格式的内容，为了播放许多不同种类的媒体内容，用户必须安装多个播放器插件。为了统一格式，使用户安装了一个播放器插件以后，就能播放多种格式的多媒体内容。这就是 SMIL 语言的设计目标。

本章包括以下内容：

- SMIL 是什么
- SMIL1.0 规范简介
- SMIL DTD
- SMIL 主要结构细节
- SMIL 支持工具
- SMIL 实例

12.1　SMIL 是什么

12.1.1　SMIL 定义

SMIL 即 W3C 的同步多媒体集成语言（Synchronized Multimedia Integration Language），是一种与 XML 1 完全兼容的标记语言，其宗旨是将多媒体对象集成到同步表现中。可把文本、静止图像、音频、视频等媒体内容组合在一起。作者可以描述下面这些要点：

- 多媒体形式的临时行为。
- 多媒体形式在屏幕上的版式。
- 与媒体对象相关的联接。

12.1.2　SMIL 的特点

1. 向电视靠拢

电视节目使用了大量多媒体元素，其中图像的显示、伴音、文字等必须是同步的。而 Web 虽然已经是一个多媒体环境，但缺少一种时间上的同步控制机制。例如：“打开音频文档 A 的同时打开视频文档 B”，或是“在音频文档播放完之后显示图像 C”等，目前 Web 还缺乏这种同步控制机制。而 SMIL 可以表述这类信息，因而可以在 Web 上创建类似于电视节目的内容。

2. 提高带宽利用率

在电视新闻播放时，大部分屏幕都显示全动态的视频信息，只有一小部分屏幕区域是静态的。而 SMIL 的一个显著优点就是它在显示电视内容时，可以尽量避免将低带宽的文本、图片转换为高带宽的视频，从而降低了对带宽的需求。

3. 简化创作工具

目前，很少有人开发 Web 上的同步多媒体信息，因为这需要专门的创作工具或者要进行编程。而创建 SMIL 文档和 HTML 文档类似，只需一个简单的文本编辑器就可完成。开发者可以使用一些简单的 XML 元素，不用学习复杂的脚本语言。

4. 促进信息的国际化

SMIL 可以满足对多语言的需求。例如：SMIL 文档可以在同一页面中包括汉语和英语的音频文档，然后根据用户的参数设置自动选择下载中文或英文版本。

5. 与 Web 体系结构紧密集成

SMIL 中包括所有 Web 用户熟悉的部件，如 URL、基于 CSS 的页面布局、基于 HTML 的超链接以及基于 XML 的语法。SMIL 是 W3C 推荐的第一个使用 XML 名字空间集成新元素的语言，SMIL 元素也可以加入到其他需要同步功能的 XML 应用中去。

12.2 SMIL1.0 规范简介

1998 年万维网联盟（W3C）正式推荐了同步多媒体综合语言 SMIL。1999 年 8 月 3 日，在第一个草案的基础上，W3C 推出了 SMIL Boston 版本。SMIL Boston 有了许多重要的扩展：包括可重复使用的模块、通用的动画设计、改良的交互功能以及电视综合功能等。

SMIL Boston 规范包括以下内容：

（1）SMIL 的模块，这些模块为其它基于 XML 的语言（如 XHTML）提供多媒体特色。SMIL 有九个模块，包括：动画模块；内容控制模块、事件模块、层次模块、链接模块、媒体对象模块、元信息模块、结构模块和时序和同步模块。目前对于动画、链接、媒体对象以及时序和同步模块有详细的定义和说明，其他尚在进行之中。

（2）将 SMIL 的时序信息集成到到基于 XML 的语言。

（3）SMIL 文档对象模块。

SMIL Boston 设计目标：

（1）定义一基于 XML 的简单语言，可以允许用户编写交互性的多媒体内容。利用 SMIL Boston，作者可以描述多媒体内容的临时表现方式，结合媒体对象的超链接以及描述在屏幕上表现的层次。

（2）SMIL 的语法和语义可以在其它基于 XML 的语言中重用，尤其是需要表达定时和同步的情况。例如，SMIL Boston 部件应该能将定时集成到 XHTML 中。

SMIL Boston 被定义为一标记模块的集合，它规定了语义和 SMIL 特定应用领域的 XML

语法。所有的模块都有相应的文档对象模块。

SMIL Boston 对 SMIL1.0 语法的修正更有利于友好的 DOM 语法支持。最显著的是从用连字符连接的属性名改变为大小写混合的属性名，例如：原来的属性名 clip-begin 变为了 clipBegin.SMIL Boston，模块没有包含 SMIL 1.0 的属性，这样为支持的集成应用减轻了负担。SMIL 支持重放“application/smil”文档（或<smil></smil>文档）的应用——文档播放器必须支持 SMIL1.0 的属性名。

12.3 SMIL DTD

```
<!-- Generally useful entities -->
<!ENTITY % id-attr "id ID #IMPLIED">
<!ENTITY % title-attr "title CDATA #IMPLIED">
<!ENTITY % skip-attr "skip-content (true|false) 'true'">
<!ENTITY % desc-attr "
        %title-attr;
        abstract        CDATA     #IMPLIED
        author          CDATA     #IMPLIED
        copyright       CDATA     #IMPLIED
">
<!ELEMENT smil (head?,body?)>
<!ATTLIST smil
        %id-attr; >
<!ENTITY % layout-section "layout|switch">
<!ENTITY % head-element "(meta*,((%layout-section;), meta*))?">
<!ELEMENT head %head-element;>
<!ATTLIST head %id-attr;>
<!ELEMENT layout ANY>
<!ATTLIST layout
        %id-attr;
        type CDATA      "text/smil-basic-layout">
<!ENTITY % layout-attrs "
        height          CDATA       #IMPLIED
        width           CDATA       #IMPLIED
        background-color    CDATA       #IMPLIED
">
<!ELEMENT region EMPTY>
<!ATTLIST region
        %id-attr;
        %title-attr;
        %layout-attrs;
        left            CDATA       "0"
        top             CDATA       "0"
        z-index         CDATA       "0"
        fit             (hidden|meet|scroll|slice|fit)      "hidden">
```

```
<!ELEMENT root-layout EMPTY>
<!ATTLIST root-layout
        %id-attr;
        %title-attr;
        %layout-attrs;
        overflow        (visble|hidden|scroll|auto|inherit)     "hidden"
        %skip-attr; >
<!ELEMENT meta EMPTY>
<!ATTLIST meta
        name    NMTOKEN #REQUIRED
        content CDATA   #REQUIRED
        %skip-attr; >
<!ENTITY % media-object "audio|video|text|img|animation|textstream|ref">
<!ENTITY % schedule "par|seq|(%media-object;)">
<!ENTITY % inline-link "a">
<!ENTITY % assoc-link "anchor">
<!ENTITY % link "%inline-link;">
<!ENTITY % container-content "(%schedule;)|switch|(%link;)">
<!ENTITY % body-content "(%container-content;)*">
<!ELEMENT body (%body-content;)*>
<!ATTLIST body %id-attr;>
<!ENTITY % sync-attributes "
        begin   CDATA   #IMPLIED
        end     CDATA   #IMPLIED
">
<!ENTITY % system-attribute "
        system-bitrate          CDATA           #IMPLIED
        system-language         CDATA           #IMPLIED
        system-required         NMTOKEN         #IMPLIED
        system-screen-size      CDATA           #IMPLIED
        system-screen-depth     CDATA           #IMPLIED
        system-captions         (on|off)        #IMPLIED
        system-overdub-or-caption (caption|overdub)  #IMPLIED">
<!ENTITY % fill-attribute "   fill    (remove|freeze)    'remove'  ">
<!ENTITY % par-content "%container-content;">
<!ELEMENT par    (%par-content;)*>
<!ATTLIST par
        %id-attr;
        %desc-attr;
        endsync CDATA           "last"
        dur     CDATA           #IMPLIED
        repeat  CDATA           "1"
        region  IDREF           #IMPLIED
        %sync-attributes;
        %system-attribute; >
<!ENTITY % seq-content "%container-content;">
```

```
<!ELEMENT seq       (%seq-content;)*>
<!ATTLIST seq
            %id-attr;
            %desc-attr;
            dur         CDATA                  #IMPLIED
            repeat    CDATA                   "1"
            region    IDREF                  #IMPLIED
            %sync-attributes;
            %system-attribute; >
<!ENTITY % switch-content "layout|(%container-content;)">
<!ELEMENT switch (%switch-content;)*>
<!ATTLIST switch
            %id-attr;
            %title-attr; >
<!ENTITY % mo-attributes "
            %id-attr;
            %desc-attr;
            region        IDREF                 #IMPLIED
            alt             CDATA                 #IMPLIED
            longdesc     CDATA                 #IMPLIED
            src             CDATA                 #IMPLIED
            type            CDATA                 #IMPLIED
            dur             CDATA                 #IMPLIED
            repeat        CDATA                 '1'
            %fill-attribute;
            %sync-attributes;
            %system-attribute;
">
<!ENTITY % mo-content "(#PCDATA|%assoc-link;)*">
<!ENTITY % clip-attrs "
            clip-begin          CDATA     #IMPLIED
            clip-end             CDATA     #IMPLIED
">
<!ELEMENT ref                %mo-content;>
<!ELEMENT audio             %mo-content;>
<!ELEMENT img                %mo-content;>
<!ELEMENT video             %mo-content;>
<!ELEMENT text               %mo-content;>
<!ELEMENT textstream      %mo-content;>
<!ELEMENT animation       %mo-content;>
<!ATTLIST ref                %mo-attributes; %clip-attrs;>
<!ATTLIST audio             %mo-attributes; %clip-attrs;>
<!ATTLIST img                %mo-attributes; >
<!ATTLIST video             %mo-attributes; %clip-attrs;>
<!ATTLIST animation       %mo-attributes; %clip-attrs;>
<!ATTLIST text               %mo-attributes; >
```

```
<!ATTLIST textstream      %mo-attributes; %clip-attrs;>
<!ENTITY % smil-link-attributes "
        %id-attr;
        %title-attr;
        href              CDATA                    #REQUIRED
        show              (replace|new|pause)      'replace'
">
<!ELEMENT a (%schedule;|switch)*>
<!ATTLIST a
        %smil-link-attributes;
>
<!ELEMENT anchor EMPTY>
<!ATTLIST anchor
        %skip-attr;
        %smil-link-attributes;
        %sync-attributes;
        coords            CDATA                    #IMPLIED
        fragment-id       CDATA                    #IMPLIED
        z-index           CDATA                    "0"
>
```

12.4 SMIL 主要结构细节

SMIL 元素把文档划分为文档头和主体部分，然后描述作为表现部分的各种各样的多媒体元素。任何 SMIL 文档的文档元素是 SMIL，其他任何文档元素必须嵌套在<SMIL></SIML>标记之内。

像 HTML 中一样，SMIL 文档的文档头部分是用来包含那些提供文档重要信息的元素的，但是这些元素不会影响文档的最后显示。下面的元素将包含在文档头中：

HEAD 元素

内容：LAYOUT、META、SWITCH。

属性：ID。

LAYOUT 元素 指明文档使用的是什么样式的版式语言。

内容：REGION、ROOT-LAYOUT。

属性：ID、TYPE。

REGION 元素

内容：SKIP-CONTENT。

属性：BACHGROUND-COLOR、FIT=（FILL|HIDDEN|MEET|SCROLL|SLICE）、HEIGHT、ID、LEFT、TITLE、TOP、WIDTH、ZINDEX。

多数属性都是作为一种工具，用来描述便捷的物理属性。BACHGROUND-COLOR 属性用来描述制定区域的背景颜色；HEIGHT 和 WIDTH 属性设置区域的尺寸；LEFT 和 HIGH 用来制定区域的位置。FIT 属性指定如果内容超出区域能显示的空间时应该如何改变区域

的大小以及显示多少区域的内容，该属性的默认值是隐藏。ID 属性提供了一种方法，用来给区域分配一个唯一的标识符。

ROOT-LAYOUT 元素

内容：空。

属性：BACKGROUND-COLOR、HEIGHT、ID、TITLE、WIDTH。

BACKGROUND-COLOR、HEIGHT、ID、TITLE 和 WIDTH 属性所起的作用与它们在 REGION 元素中起的作用一样。

BODY 元素

内容：A、ANIMATION、AUDIO、IMG、PAR、REF、SEQ、SWITCH、TEXT、TEXTSTREAM、VIDEO。

属性：ID。

PAR 元素

内容：A、ANIMATION、AUDIO、IMG、PAR、REF、SEQ、SWITCH、TEXT、TEXTSTREAM、VIDEO。

属性：包含大量的属性来控制媒体显示。

“<PAR></PAR>”标志对描述了元素的一个集合，这个集合的元素必须同时播放，而不是按顺序播放。使用该元素，可以在播放背景音乐的同时，使一系列的幻灯片一个接一个地出现。

SEQ 元素

内容：A、ANIMATION、AUDIO、IMG、PAR、REF、SEQ、SWITCH、TEXT、TEXTSTREAM、VIDEO。

属性：大量属性控制多媒体如何显示。

“<SEQ></SEQ>”元素对描述的是必须按顺序播放的元素，该元素允许将一些媒体部件组织成单一的形式。

ANIMATION AUDIO IMG TEXT TEXTSTREAM 和 VIDEO 元素

内容：ANCHOP。

属性：大量属性可参见 SMIL 规范。

这一组标志对在 SMIL 规范里被集体称为媒体对象元素，它们的作用是描述多媒体场景的所有不同部分。

<ANIMATION></ANIMATION>描述的是动画。

<AUDIO></AUDIO>描述的是音频。

<VIDEO></VIDEO>描述的是视频文档。

<IMG></IMG>描述的是图像。

<TEXT></TEXT>描述的是文本。

<TEXTSTREAM></TEXTSTREAM>描述的是必须在场景中从一边移到另一边的文本块（文本流信息）。

要想看看所有这些不同的元素如何组合在一起，创造出真实的 SMIL 文档，可访问 RealNetworks 的技术站点 http：//www.real.com/showcase/realplayer/tech/index.htm。

12.5 SMIL 支持工具

RealNetworks 公司的 RealPlayer G2 是第一个支持 SMIL 的商业软件，目前它已能支持大部分 SMIL1.0 规范。其语言和 HTML 文档类似，SMIL 文档由一个标记开始，并且包含两部分内容。一部分包括对页面布局和外观的描述信息，另一部分包括页面内容及时序信息。成对出现的标记写法与 HTML 文档中的一样，而单独出现的标记，必须以“/>”结束，这是 XML 语言所作的语法定义。

基本标记表示同步播放多媒体元素，表示要依次播放，和可以任意相互嵌套以产生特殊效果。时序控制是自动完成的，也就是说，在一个段中，第一个文档播放结束时第二个文件会立即接着播放。如果希望对文档进行单独控制，可使用 begin、end、dur 等特性，以决定何时开始播放及播放多久。

12.6 SMIL 实例

例 12.1 “幻灯片”，使用行内时序：

考虑一种基于 XML 的图像列表语言。每一个文档都包括一列指向 JPEG 图像的索引。在行内说明了图像相互间的时序关系。以下就是这样的一个文档，在表中的每一个图像存在规定的持续时间，然后由下一个图像所代替。但最后一个图像只能持续 10 秒中的 8 秒时间，因为已经达到了其父节点规定的总时间。

```
<imagelist timeLine="seq" end="28s">
   <image dur="5s" src="image1.jpg" />
   <image dur="3s" src="image2.jpg" />
   <image dur="12s" src="image3.jpg" />
   <image dur="10s" src="image4.jpg" />
</imagelist>
```

例 12.2 “增长的列表”，使用 CSS 时序：

```
/* style sheet document "growlist.css": */
.seqtimecontainer { timeLine: seq; dur: 30s}
LI { dur: 10s; }

<!-- HTML document (which happens to be well-formed XML): -->
<HTML>
   <HEAD>
    <LINK rel="stylesheet" type="text/css" href="growlist.css" />>
   </HEAD>
   <BODY>
     <UL class="seqtimecontainer">
          <LI>This is item 1.   It appears from 0 to 30 seconds.
          </LI>
          <LI>This is item 2.   It appears from 10 to 30 seconds.
          </LI>
```

```
        <LI>This is item 3.  It appears from 20 to 30 seconds.
        </LI>
      </UL>
  </BODY>
</HTML>
```

例 12.3 “正方形”，使用时间表：

考虑一个用图形语言写成的文档，三个大的正方形排列在一个长方形中，每一个正方形中包含有一个更小的正方形。我们要能产生一个时间表，安排这些正方形出现不同次序。

```
<rectangle id="window" geometry="..." fill="...">
  <square id="b1" ... >
    <square id="s1" ... / >
  </square>
  <square id="b2" ... >
    <square id="s2" ... / >
  </square>
  <square id="b3" ... >
    <square id="s3" ... / >
  </square>
</rectangle>
```

要使大的正方形逐个出现，每个正方形出现的同时出现小的正方形，时间表可以这样表示：

```
<time>
  <seq>
    <par>
      #b1 { dur: 2s }
      #b2 { dur: 2s; begin: 2s; }
      #b3 { dur: 2s; begin: 4s; }
    </par>
    <par>
      #s1 { }
      #s2 { }
      #s3 { }
    </seq>
  </seq>
</time>
```

注释：外部“窗口”——矩形没有给出明确的时序。在该例中，我们默认意味着不需要起始时间，如果没有明确的持续时间那就无限制延续。

注意，目前 W3C 的建议还没有法律上的约束力，但 SMIL 已经引起了业界的广泛重视。W3C 同步多媒体工作组负责人表示，SMIL 一定会获得成功。SMIL 是一种正在完善中的语言，但它顺应了家庭娱乐与计算机技术相融合的趋势，拓展了 Web 的使用范围，必将为今后 WWW 技术的发展带来深远的影响。

12.7 小　　结

多媒体是 Internet 和 Web 产业中增长最快且非常活跃的领域，SMIL 通过规范多媒体 Web 分发规则而推动各种包含多媒体的信息能够共享。本章介绍了 SMIL 技术及其相关的思想，使读者明确 SMIL 将在未来用作 Web 技术中的多媒体集成方案。

第十三章　ASP 与 XML 的联合开发

本章我们通过一个具体的 Web 站点开发实例来说明 XML 和 ASP 在开发三层体系结构应用程序时的初步应用。

本章包括以下内容：

- 三层 Web 应用程序简介
- Server-Side XML in ASP

13.1　三层 Web 应用程序简介

企业信息网络以 Web 为中心，采用 TCP/IP 技术，以 HTTP 为传输协议，客户端通过 Browser 访问 Web 以及与 Web 相连的后台数据库，这种网络信息存取技术及其体系结构，被称之为三层体系结构。如图 13-1。

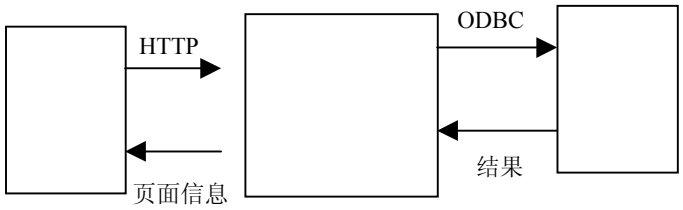

图 13-1

我们通过微软公司的一个在线拍卖程序 Auction Demo 来说明利用 XML 和 ASP 进行开发的过程。在网站 http://www.microsoft.com/xml 中可以看到程序演示。它主要有三个 Web 页面来完成功能实现：

UserInterface.htm：该页使用动态 HTML 来允许 Web 浏览器将拍卖信息提交给客户。通过 ASP 在中间层上收集和更新数据。

Auction.asp：该页是一个 asp 页面。当 UserInterface.htm 请求该页的时候，其中的脚本就会在服务器上执行。脚本生成 auction.xml 文档，xml 文档包含最新的拍卖数据，并发送给客户。

Makebid.asp：当用户想要进行拍卖的时候，UserInterface.htm 请求该页，它在中间层上执行，用新的拍卖信息对数据元进行更新。

13.1.1　建立项目数据库

本项目是建立了一个名叫 Auction 的关系数据库，其中有两个表，一个是 Item 表，另一个是 Bids 表，Item 表包含要拍卖的每个项目的数据。

13.1.2　定义 XML 文档结构

创建有用 XML 文档的关键是正确的数据结构。对于 Auction Demo 来说，这意味着在

XML 中 Item 表中的记录是如何用 ITEM 元素来表示的。下面是一个 ITEM 元素模板：

```
<ITEM>
    <TITLE></TITLE>
    <ARTIST></ARTIST>
    <DIMENSIONS></DIMENSION>
    <MATERIALS></MATERIALS>
    <YEAR></YEAR>
</ITEM>
```

对于 Item 表中的每一个字段，ITEM 元素都有相应的子元素。

13.1.3 使用 ASP 生成 XML 文档

可以使用 ASP 在中间层生成 XML 文档。ASP 提供了一个环境，开发者可以在其中将标记语言和嵌入脚本相结合来动态地生成文档。脚本可以用很多脚本语言来编写，如 VBScript、JScript 等，你可以调用服务端的组件来访问数据库、执行应用程序和处理信息。在中间层上生成 XML 的能力，使得 Web 应用程序提供能够在客户上进行操作和刷新的内容，而无需刷新整个用户界面。下面是 Auction.asp 程序代码从数据库中获取数据并且生成 XML 文档。

```
<%@ LANGUAGE= VBScript %>
<?xml version= "1.0"?>
<AUCTIONBLOCK>
<%Set conn=Server.CreateObject("ADODB.Connection")
Conn.open    "Auction", "Auction", "Auction"
Set ItemRS=Conn.Excute("Select * From item")
DO While Not ItemRS.EOF
%>
<ITEM>
    <TITLE><%=ItemRS("Artist")%></TITLE>
    <ARTIST><%=ItemRS("Artist")%></ARTIST>
    <DIMENSIONS><%=ItemRS("Dimension")%></DIMENSION>
    <MATERIALS><%=ItemRS("Materrials")%></MATERIALS>
    <YEAR><%=ItemRS("Year")%></YEAR>
</ITEM>
<%
ItemRS.MoveNext
Loop
%>
< /AUCTIONBLOCK >
```

13.1.4 从多数据库生成 XML

在中间层上从多数据库中生成 XML 文档的方法和从单一数据库中生成没有多大区别，唯一的区别是需要建立多个数据库联接。下面是一个从多数据库中生成 XML 文档的例子：

```
<%@ LANGUAGE= VBScript %>
<?xml version="1.0"?>
<AUCTIONBLOCK>
```

```
<%Set conn=Server.CreateObject("ADODB.Connection")
Conn.open    "Gallery1", " Gallery1", " Gallery1"
Set ItemRS=Conn.Excute("Select * From item")
DO While Not ItemRS.EOF
%>
<ITEM>
        <TITLE><%=ItemRS("Artist")%></TITLE>
        <ARTIST><%=ItemRS("Artist")%></ARTIST>
        <DIMENSIONS><%=ItemRS("Dimension")%></DIMENSION>
        <MATERIALS><%=ItemRS("Materrials")%></MATERIALS>
        <YEAR><%=ItemRS("Year")%></YEAR>
</ITEM>
<%
ItemRS.MoveNext
Loop
%>
<%Set conn=Server.CreateObject("ADODB.Connection")
Conn.open    "Gallery2", " Gallery2", " Gallery2"
Set ItemRS=Conn.Excute("Select * From item")
DO While Not ItemRS.EOF
%>
<ITEM>
        <TITLE><%=ItemRS("title")%></TITLE>
        <ARTIST><%=ItemRS("Artist")%></ARTIST>
        <DIMENSIONS><%=ItemRS("size")%></DIMENSION>
        <MATERIALS><%=ItemRS("medium")%></MATERIALS>
        <YEAR><%=ItemRS("Date")%></YEAR>
</ITEM>
<%
ItemRS.MoveNext
Loop
%>
< /AUCTIONBLOCK >
```

13.1.5 从数据库和 XML 数据源中生成 XML

在系统中，数据源往往不止一种。我们可以从多个不同的数据源生成 XML 文档。下面是一个从数据库和 XML 文档生成 XML 文档的例子：

```
<%@ LANGUAGE= VBScript %>
<?xml version="1.0"?>
<AUCTIONBLOCK>
<%Set conn=Server.CreateObject("ADODB.Connection")
Conn.open    "Gallery1", " Gallery1", " Gallery1"
Set ItemRS=Conn.Excute("Select * From item")
DO While Not ItemRS.EOF
%>
<ITEM>
```

```
        <TITLE><%=ItemRS("Artist")%></TITLE>
        <ARTIST><%=ItemRS("Artist")%></ARTIST>
        <DIMENSIONS><%=ItemRS("Dimension")%></DIMENSION>
        <MATERIALS><%=ItemRS("Materrials")%></MATERIALS>
        <YEAR><%=ItemRS("Year")%></YEAR>
</ITEM>
<%
ItemRS.MoveNext
Loop
%>
<%
Set XML=Server.CreateObject("msxml")
XML.URL=http://datasourse3/Gallery3.xml
Set Items=XML.root.children
For I=0 TO Items.length-1
%>
<ITEM>
        <TITLE><%=Items.item(I).children.item("TITLE").text%></TITLE>
        <ARTIST><%=Items.item(I).children.item("ARTIST").text%></ARTIST>
        <DIMENSIONS><%=Items.item(I).children.item("DIMENSIONS").text%></DIMENSION>
        <MATERIALS><%=Items.item(I).children.item("MATERIALS").text%></MATERIALS>
        <YEAR><%=Items.item(I).children.item("DATE").text%></YEAR>
</ITEM>
<%Next%>
</ AUCTIONBLOCK >
```

13.1.6 创建用户界面

用户界面对于任何应用程序都很重要，必须允许用户以有效的方式与程序进行交互。下面的程序是通过程序脚本和描述绑定来实现动态交互的。

1. 创建拍卖文档对象

```
var auction= new ActiveXObject("msxml");
auction.URL=HTTP://Webserver/auction.asp;
```

2. 从拍卖文档中提取数据

```
var root=auction.root;
var item0=root.children.item("ITEM",0);
var title=item0.children.item("TITLE").text;
document.all("item_title").innerText=title;
<Div ID="item_title"></DIV>
```

3. 绑定在 XML DSO 上的数据

数据绑定用于填充用户界面的部分来显示图画和下面的标注。

```
<APPLET ID=auction CODE=com.ms.xml.dso.XMLDSO.class MAYSCRIPT    WIDTH=0 HEIGHT=0 >
<PARAM NAME="URL" VALUES="auction.asp">
```

```
</APPLET>
<TD>
    <DIV STYLE="margin-left:16px; margin-top:16px; margin-right:16px ">
        <DIV ID=pict>
            <DIV class="details”>
            <span    DATASRC=#auction DATAFLD=MATERIALS></span>,
            <span    DATASRC=#auction DATAFLD=YEAR></span>,
            <span    DATASRC=#auction DATAFLD=DIMENSIONS></span>,
            </DIV>
        </DIV>
    </DIV>
</TD>
```

用 XML DSO 显示 XML 的好处之一是异步处理 XML 文档来重现该页。因此如果图画很多，就能使文档的开始元素先于最后的元素显示。

13.1.7 更新数据源

从客户页面上获得数据后，需要更新数据库信息。下面是更新数据库信息的代码：

```
<%@ LANGUAGE=VBScript%>
<%
title=Request.QueryString("title")
price= Request.QueryString("price")
bidder= Request.QueryString("bidder")

set BidRS=Server.CreateObject("ADODB.Recordset")
Connect="data sourse=Auction;user id =sa ;password=";"
BidRS.CursorType=2
BidRS.Open "Bids", Connect

BidRS.AddNew
BidRS("item")=title
BidRS("price")=price
BidRS("bidder")=bidder
BidRS.Update
BidRS.Close
%>
<STATUS>OK</STATUS>
```

13.1.8 结语

XML 通过提供动态的、可以在客户上进行寻址和操作的可访问的内容来启动 Web 应用程序。另外，它无需一定要刷新整个用户界面就可以更新内容。通过减少到服务器的存取数据的次数来节省时间，提高效率。使用 XML，用户可以通过 Internet 来管理数据，就像目前在本地所做的那样。结果是 Web 成为一个具有交互性能共同使用的媒介。

13.2 Server-Side XML in ASP

13.2.1 简介

随着免费的 IE5 的发布，在 Web 应用中使用 XML 将更加容易。这里介绍一些有关在 ASP 应用程序中使用升级后的 XML 的文档对象模型所产生的新的功能。

13.2.2 需求

在服务器端使用 XML 让开发者有了一个全新的开发功能的天地。当我们不断地提高使用 XML 的无限扩展能力的时候，我们也需要一些基础知识才能熟练地在服务器端操作 XML。

13.2.3 文档对象模型 DOM

在 IE5 中，升级后的 DOM 完全支持和 W3C 所推荐的文档对象模型内核（Level 1）中所表达的调试界面。而且它还包含了很多方法来支持相关的 XML 技术，如 XSL、XSL 匹配模式、名字空间、数据类型以及模式等等。

W3C 推荐了两组的 DOM 编程接口。首先是定义了编写使用 XML 的应用程序的界面。其次是定义了使开发者更容易地使用 XML 的界面。第二组界面是用于方便开发者的，但并不是必需的。

在 ASP 应用程序中使用 DOM 是一件非常简单的事，但是需要在服务器端本身也安装了 IE5。IE5 的安装过程将决定其所能支持的 XML 的功能强弱。只要安装了 IE5 后，用户只要用下面的语句就可以在 ASP 中使用 XML DOM。

```
<% Set objXML = Server.CreateObject("Microsoft.XMLDOM") %>
```

13.2.4 server 端的 XML

一旦创建了 DOM 对象实例后,用户就可以建立起自己的 XML 文档或装入现有的文档。在装入文档时，可以选择其中的 XML 文本中的字符串、打开 XML 文档或者是装载其中一部分内容。在下面的例子中，假设我们的服务器上存有一份最近的 Scripting News XML 文档的代码。在装载文档之前，假设 DOM 对象的异步属性的值为 FALSE。这意味着要 DOM 不要以异步的方式来下载 XML 文档。这点是很重要的，下载了文档后，我们开始使用它的内容。要是没有下载到它的内容，当我们访问它的时候就会报错。

```
<%
Set objXML = Server.CreateObject("Microsoft.XMLDOM")
objXML.async = False
objXML.Load (Server.MapPath("mostRecentScriptingNews.xml"))
%>
Let Us look at the actual XML document that we are loading:
下面是我们要下载的文档：
<?xml version="1.0"?>
<!DOCTYPE scriptingNews SYSTEM
        "http://www.scripting.com/dtd/scriptingNews.dtd">
```

```
<scriptingNews>
    <header>
      <copyright>Copyright 1997-1999 UserLand Software, Inc.
      </copyright>
      <scriptingNewsVersion>1.0</scriptingNewsVersion>
      <pubDate>Wed, 03 Mar 1999 08:00:00 GMT</pubDate>
      <lastBuildDate>Thu, 04 Mar 1999 03:37:03 GMT</lastBuildDate>
    </header>
    <item>
      <text>Wired: A Linux Car Stereo! Wow.</text>
      <link>
        <url>http://www.wired.com/news/news/technology/ ?
              story/18236.html
        </url>
        <linetext>A Linux Car Stereo</linetext>
      </link>
    </item>
    ?
    <item>
      <text>According to News.com, Hewlett-Packard will offer
            customers storage and computing on a rental basis.
      </text>
      <link>
        <url>http://www.news.com/News/Item/ ?
              0,4,33202,00.html?st.ne.fd.mdh
        </url>
        <linetext>According to News.com</linetext>
      </link>
    </item>
</scriptingNews>
```

DOM 包含了错误对象，使用它可以访问到最近的错误信息。这个对象对于在 ASP 中调试和错误陷阱非常有帮助。

```
<%
If objXML.parseError.errorCode <> 0 Then
      handle the error
End If
%>
```

我们可以从中得到许多有价值的错误信息，以下一些有关的属性的表达式：

Property　Description

ErrorCode：　The error code

filepos：　The absolute file position in the XML document containing the error

Line：　The line number in the XML document where the error occurred

Linepos：　The character position in the line containing the error

Reason：　The cause of the error

SrcText：　The data where the error occurred

URL：　The URL of the XML document containing the error

在我们的例子中，这个错误对象向我们展示了 XML 文档提及的 DTD 文档。在装载 XML 文档后检查错误对象是一个好的方法。

既然我们建立了一个有效的使用 DOM 对象的文档，让我们看一下文档里有什么。DOM 揭示了许多有用的方法来确定在 XML 文档中的内容。因为 DOM 解释了文档中的嵌套节点的树型结构，我们可以使用 TagName 方法来获取文档中的节点和其他元素。

我们的第一个目标是发现我们的 Scripting News 拷贝的发行数据。从上面的 DTD 的例子中，我们知道这个信息是存储在 pubDate 节点中的。一个获取这个节点内容的简单方法就是先创建一个包含所有 XML 文档中的节点的对象集合，然后循环找出这个节点。由于 DTD 表明了 pubDate 节点不包含任何的子节点，所以我们可以用文本属性很快地得到这个节点的内容。

```
<%
Set objXML = Server.CreateObject("Microsoft.XMLDOM")
Set objLst = Server.CreateObject("Microsoft.XMLDOM")
objXML.async = False
objXML.Load (Server.MapPath("mostRecentScriptingNews.xml"))
If objXML.parseError.errorCode <> 0 Then
      handle the error
End If

Set objLst = objXML.getElementsByTagName("*")

For i = 0 to (objLst.length ?1)

    If objLst.item(i).nodeName = "pubDate" Then
      StrDate = objLst.item(i).text
      Exit For
    End If

Next
%>
```

注意，以上例子我们使用了 getElementsByTagName 方法，这个方法返回一个包含 XML 文档中所有元素或节点的集合。我们有了 DTD 并能获得 pubDate 节点的准确位置，因此我们就能用它在集合中的序列号直接访问它。以上文档中使用的循环对于节点集合的确是一种行之有效的方法。

既然我们已经获得了发布数据，就让我们看看怎样得到文档中的大标题的数目。我们要从 DTD 中获得有关的知识，还要从储存在 pubDate 节点中获取大标题的数目。我们可以用另一个循环，就像刚才做的那样。然后在每次循环对计数器加一来获取数目。然而有个更好的方法来得到这个信息，就是使用 DOM 中的其他方法。

在下面的例子中，我们所做的工作仅仅是创建一个包含所有的节点的对象集合。然后使用 LENGTH 属性就可以知道这个节点的数目了，也就是说，获得这个文档中的大标题数目。程序如下：

```
<%
```

```
Set objLst = objXML.getElementsByTagName("item")
strNoOfHeadlines = objLst.Length
%>
```

多数情况下，我们将信息显示在 ASP 页面上，下面这个例子就表明了怎样用循环在 ASP 页面上列出大标题和他们的 URL 地址：

```
<%
Set objXML = Server.CreateObject("Microsoft.XMLDOM")
Set objLst = Server.CreateObject("Microsoft.XMLDOM")
Set objHdl = Server.CreateObject("Microsoft.XMLDOM")

objXML.async = False
objXML.Load (Server.MapPath("mostRecentScriptingNews.xml"))

If objXML.parseError.errorCode <> 0 Then
        handle the error
End If

Set objLst = objXML.getElementsByTagName("item")

noOfHeadlines = objLst.length
%>

<HTML><BODY>
<H1>Scripting News Headlines</H1>

<%
For i = 0 To (noOfHeadlines ?1)

   Set objHdl = objLst.item(i)

   Response.Write("<a href=""" & _
                        objHdl.childNodes(1).childNodes(0).text & _
                        """>" & objHdl.childNodes(0).text & _
                        "</a><br>")
Next
%>

</BODY></HTML>
```

这里我们了解了一些有关 XML 文档的结构以及 DOM 对象的强大功能的信息，从中可以知道：在 ASP 文档中很容易使用 XML，并且能够随心所欲地发送信息到客户端。

13.3 小　　结

Web 应用系统借助 ASP 和 XML 可以实现动态的访问和远程管理数据，使 Web 成为具有巨大交互性的信息媒介。ASP 与 XML 的结合在实际应用有着非常大的潜力，有兴趣的读者可以关注相应的书籍和资料。