

第13章 样式XML

当人们交流时，通常使用两个主要的感觉通道：听觉和视觉。我们前面已经看到，XML将数据与表示规则分隔开来。因此，XML的样式语言要将一个原始XML文件转换成适应视觉或听觉感应系统的一系列“解释对象”。由于我们现在开始考虑的是适用于移动产品的网络应用程序，而不是桌面计算机系统中使用的传统浏览器，因此我们需要有比HTML更灵活的样式机制。在这一章里，我们不仅讨论传统浏览器中的XML样式表，还要讨论如何生成其他格式的样式表，如打印样式表和语音浏览器的样式表。

样式语言是很奇怪的产品。它们的基础原型是描述编程，这种编程只需要说明“你想做什么”，而不需要告诉系统“如何得到你想要的结果”。根据需求，你可以选择一系列工具，其中一些是基于专用语言如Balise或Omnimark的，另外一些则是基于标准的说明如层叠样式表（Cascading Style Sheets, CSS）、扩展样式表语言（eXtensible Stylesheet Language, XSL）或文件样式语义和说明语言（Document Style Semantic and Specification Language, DSSSL）。

在这一章里，我们采用两种不同的样式语言来讨论XML解析或XML样式：

- CSS。
- XSL。

在本章结尾我们也简单地讨论一下DSSSL和Omnimark，它们都和XML一起使用。CSS和XSL是W3C产品，DSSSL是国际标准化组织（ISO）的产品。但是，虽然Omnimark可以从Web上免费下载，它实际上是一种专用语言。当讨论XSL时，我们主要讨论它如何将XML文件转换成HTML、VOXML格式或XSL格式对象（用于打印）。VOXML是VOXML集团的产品，主要用于生成语音浏览器。如果你对这个产品的需求市场有所疑问，可以想一想汽车广播或蜂窝电话，它们都是潜在的语音浏览器。但是，在开始讨论这些语言之前，我们先来看看一些基础理论。

13.1 解释的位置

Web的实际结构要基于客户-服务器系统。HTML浏览器在第一代网络客户中占主导地位，但现在第二代网络客户已经进入市场。这些新的浏览器就是XML浏览器：Internet Explorer（第5版或更高版本）是一个XML文档的浏览器。Mozilla项目（Open Source委员会正开发的新一代Netscape浏览器）也是一个XML浏览器。

对XML的解释可以在服务器端进行，也可以在客户端进行。如果客户端的浏览器是一个XML浏览器，那么HTTP服务器的任务就极为简单，它只要链接正确的样式表到文件上（或者文件本身也许已经有和样式表的链接），并将样式表发送到浏览器上。如果客户端的浏览器不是一个XML浏览器，那么文件在被传送出去以前必须转换成一个可以显示的文件。因此，将XML文件转换成可显示的文件这一任务既可以由XML浏览器完成，也可以由HTTP服务器完成。

服务器端的XML转换

HTTP 可以看作是文件服务器，但它除具有简单的文件服务器功能以外，还有一些其他功能。这些添加功能大部分都是脚本引擎格式，其中最普遍的是 ASP、JSP、PHP 或 Cold Fusion，它们都是基于模板的。

要想处理 XML，HTTP 服务器需要具有处理 XML 的附加功能。最简单的附加功能是将 XML 文件转换成 HTML 文档，并将 HTML 文档传送到浏览器上进行解释（如 XSLT 引擎）。更复杂一些的附加功能是 XML 大全库。我们在第 9 章已经看到，先对 XML 文件进行解析，接着转换成一种内部格式，然后样式引擎用模板将内部格式转换为 HTML 文件（参见图 13-1）。然而如果有详尽的 XML 库，解析过程就不必要了，因为在内部格式中已经对文件进行解释并存储，这样样式引擎就可以直接对内部格式文件进行操作了。通常有 XML 库的服务器性能要好于那些简单的文件系统。但文件系统可以通过对已经转换成 HTML 的文件进行缓存来提高它们的性能（参见图 13-1）。

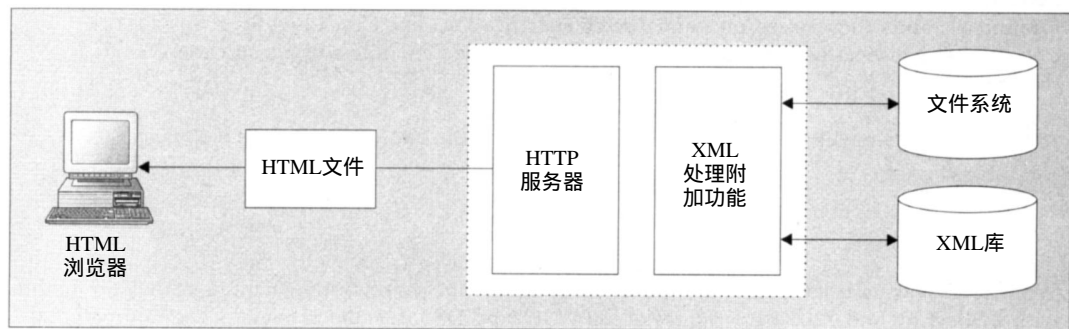


图 13-1

13.2 客户端的 XML 转换

如果客户端是一个 XML 浏览器，转换任务就容易多了。实际上，几乎所有的 HTTP 服务器都可以为 XML 浏览器提供 XML 文档。

过程很简单，HTTP 服务器传送 XML 文档给浏览器。然后浏览器用一个专用的结构“处理指令”浏览接收到的文档。我们在后面将会看到，处理指令允许浏览器对与 XML 文件相关联的样式表进行演绎并解释 XML 文档。

当然，你所选择的样式语言也取决于浏览器的功能。一些浏览器也许只能处理带 CSS 样式表的 XML 文档，而另外一些则可以处理带 XSL 样式表的 XML 文档。

13.3 解释模型

在本章中我们将要讨论的两个解释模型是语音解释模型和视觉解释模型。由于语音解释模型意味着一系列顺序的解释对象，这同时也意味着语音浏览器是一个时间相关的产品。而另一方面，视觉解释模型中的解释对象在空间放置，因此视觉浏览器是空间相关的产品。下面，我们看看这两种浏览器之间的差异。

13.3.1 视觉解释

现在，最普遍的可携带信息介质就是简单的一页纸。但是，如果考虑到每天要接收信息，则浏览器无疑正日益成为信息共享的通用介质。虽然浏览器是在屏幕上显示电子信息，但它和纸之间还有一个大的差异。纸为阅读者提供了一个固定尺寸的页面，而浏览器则通过观察端口或窗口提供了一个尺寸变化的页面。

可以传送XML文件的浏览器很容易得到，如 Microsoft Internet Explorer 5（免费下载）。IE 5 支持CSS和XSL。

IE 5不完全支持CSS，并且它对XSLT的实现也不符合W3C的建议。

此外，Netscape浏览器的下一代产品——当前正处于代码开发阶段，由一个开放资源项目进行（网址是<http://www.Mozilla.org/>）——将支持CSS和XSL的实现。

纸和浏览器都是视觉解释的介质基础。我们可以说这二者是视觉格式化对象的主要载体。样式语言的基本元素是区域。在标准样式语言中，区域是一个矩形地区。照这个观点来看，一个页面是包含其他区域的一个区域。这些包含矩形区域的矩形区域组成了一个树形结构。因此，视觉列表就是一个由格式对象（矩形区域）组成的树，页面位于树的顶部，底部是由格式对象组成的子体（参见图 13-2）。

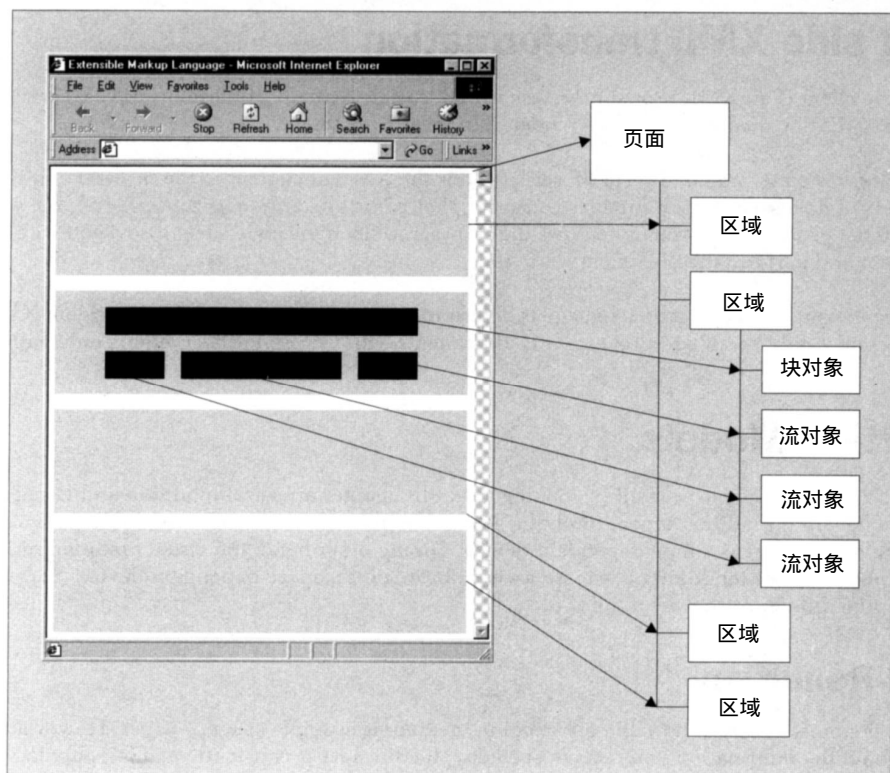


图 13-2

根据样式语言的复杂程度，有两种布局模型：

- 流式布局；
- 固定布局。

在流式布局中格式对象在介质（页面或滚动条）上一个接一个排列。如果书写顺序是从顶部到底和从左到右，那么对象的排列方式也同样是从顶部到底和从左到右（参见图 13-3）。例如，段落垂直排列，词语或句子则横向排列。大部分时间格式对象垂直和横向排列。但有时，CSS样式表语言允许块对象像段落一样竖直排列，而句子中的对象像词语或句子那样横向排列。但通常竖直排列的对象中包含横向排列的对象（适应西方的书写习惯）。

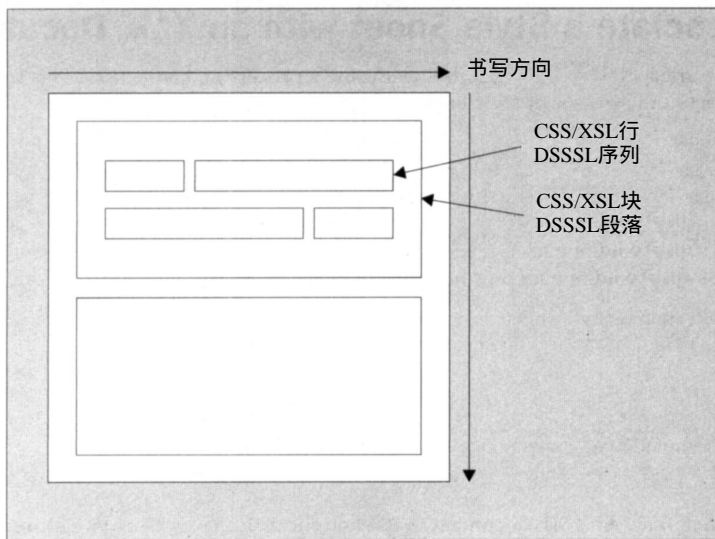


图 13-3

固定布局中格式对象在页面上有固定的位置。代表文件的区域就类似于一个卡尔坐标系，每个位置都可以由位置坐标唯一地确定。例如，CSS样式表语言允许某些格式对象的位置设定用left和top属性表示，left和top表示对象距载体左边界和上边界的距离。如果载体是一个文件，则对象的固定位置是相对于文件的上边界和左边界，如果载体是一个区域，则对象的位置是相对于区域的上边界和左边界的。

13.3.2 语音解释

语音浏览器相对于视觉浏览器而言，更不被人所知道，并且现在仍然主要处于实验室开发阶段。主要的语音模型同音乐模型很相似。声音以一定的时间间隔顺序组织。但同音乐不同的是，声音更复杂，因为在这种应用环境里声音是一个说出来的单词。因此，我们常常把语音浏览器中的元素叫做一段演讲，即由一系列停顿间隔开的词语组成的一段讲话。演讲的特征是语调、音质和节奏等。

最近在开发行业中已经出现了一些语音浏览器，比较著名的有：

- Hewlett Packard的SpeechML浏览器。

• IBM的VoiceXML浏览器。

但是，语音浏览器会总停留于实验阶段吗？我们可以用另外一个问句来回答这个问题：你会在开车时浏览吗？一个带语音命令的语音浏览器可以在汽车里传递 XML文档。因此，使用语音浏览器，可以在汽车中传递以 XML格式打包的新闻公告。

13.3.3 如何将样式表链接到XML文档上

可以使用处理规则来链接样式表到 XML文件上。处理规则规定浏览器或样式引擎：

- 怎样找到样式表。
- 样式表使用什么语言。
- 样式表是为何种媒介设计的。
- 当同时提供几种样式表时，菜单上显示的标题。
- 是否有可替换的样式表。

例如：

程序清单 13-1

```
<?xml-stylesheet href="myStyleSheet.css" type="text/css"
title="CSS style" media="screen"?>
```

处理指令 xml-stylesheet将XML文档和对应的样式表相链接。可以在 <http://www.w3.org/TR/xml-stylesheet/>上找到 W3C的建议标准 Associating Style Sheets with XML documents Version 1.0。

这个处理规则的属性如表 13-1所示。

表 13-1

属性	描 述
href	样式表地址。它的值是一个 URI，大部分情况下是一个 URL
type	样式表语言，表示为一种 MIME类型。例如，CSS语言的MIME类型是text/css，DSSSL语言的MIME类型是text/dsssl，XSL语言的MIME类型是text/xsl
media	样式表介质目标。可以是 screen、print、aural等
title	为样式表提供一个标题。这个标题可以被样式表引擎用于在菜单中显示
alternate	可以为yes或no。它告诉样式引擎同一介质是否有可替换的样式表

13.3.4 规则语言

大部分样式语言都是规则语言。规则意味着什么？一个规则由两部分组成：

- 模式匹配部分。
- 动作部分。

模式匹配部分是允许标记和行为之间相关联的一个表达式。动作部分是一个小程序或模板。如同我们在转换一章看到的，规则样式语言完成的功能包括将原始的 XML文件转换成一个

树型结构，然后一次访问树上的一个结点，与一个特定的规则相匹配。当匹配成功时就执行一个程序。当行动部分是一个模板时，就要调用一个规则或程序来解释和输出模板的内容，因为模板可能包括处理器需要处理的其他元素。实际上，我们刚才所描述的整个过程就是 XML 解释引擎完成功能的通常过程。

讨论了样式语言的一些通用特性后，下面我们来看一看本章将要集中讨论的第一种语言：CSS。这里我们将对 CSS 进行概述，在附录 F 中有 CSS1 和 CSS2 特性的全部介绍。

13.3.5 CSS

如果使用过 HTML，你可能已经遇到过层叠样式表（Cascading Style Sheet, CSS）。一些浏览器中已经包括了 CSS1 实现的一部分或全部功能，但含有一部分 CSS2 功能的相对不多。CSS1 和 CSS2 都是 W3C 建议标准，可以分别在 <http://www.w3.org/TR/REC-CSS1> 和 <http://www.w3.org/TR/REC-CSS2> 上找到。CSS 语言和 HTML 的样式说明很相像，它的说明也很容易。

1. CSS 元素：盒子

如果你今天遇到 Democritus（公元前 460~370 年的希腊哲学家），跟他提到 CSS，他会问“什么是元素？”要回答他的问题，你只要说“CSS 中所有的东西都在盒子里 所以元素就是盒子。”这个概念在 CSS 建议标准文档中指的是盒子模型（参见图 13-4）。

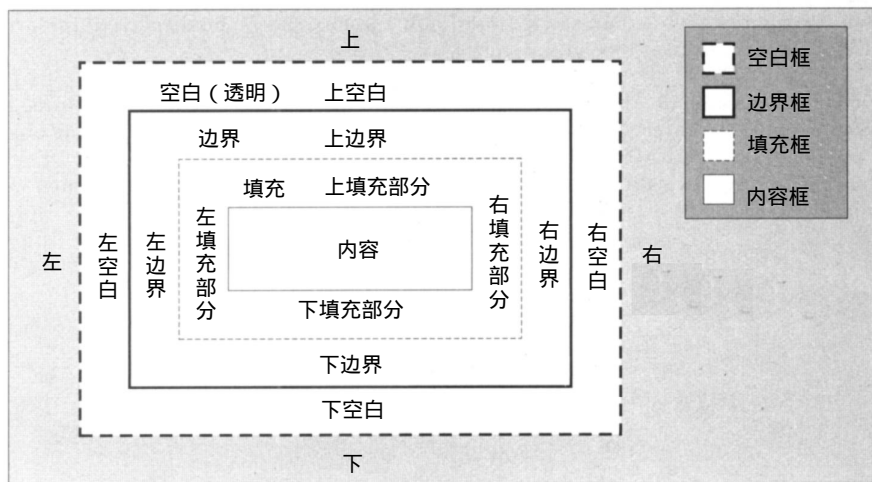


图 13-4

在 CSS 中一个盒子就是一个矩形区域，它有一些基本特性，如空白和边界特征。所有的盒子都能填充到其他盒子里，就像俄罗斯方块一样。一些盒子中包括其他盒子，另一些盒子则被别的盒子包含。这样的结构可以变成一棵由盒子组成的树或矩形区域。盒子树本身也包含在一个载体盒子中，这个载体盒子或者是连续的介质（如浏览器，它的屏幕可以滚动）或者是打印材料（区域的尺寸是固定的）。

2. XML文件的元素如何与CSS规则链接

知道了CSS是以盒子的方式解析各种事物的，我们要知道如何将 XML元素与这些盒子联系起来。答案是通过规则联结。这些规则包括一套与一个或几个元素类型相关联的特性。每个规则都由一个模式匹配部分和一个程序部分或行动部分组成。在CSS中，模式匹配部分也称作选择项，程序部分也称作特性集合。因此，CSS语言是一个规则语言。图 13-5是关于一个 CSS规则的例子。

左边是模式匹配部分（即选择项），它指定元素的名称。右边是对如何显示盒子的描述（特性）。稍后我们将更详细地讨论语法。

CSS在解释上很依赖于文件结构。格式对象树通常和文件树是一样的。基本上 CSS所做的就是将一个CSS格式对象和每个XML元素的特定属性相联结。例如，在图 13-6中的例子中，元素<ITEM>和一个块格式对象相连接，元素<DESCRIPTION>与一个块相连接等等。



图 13-5

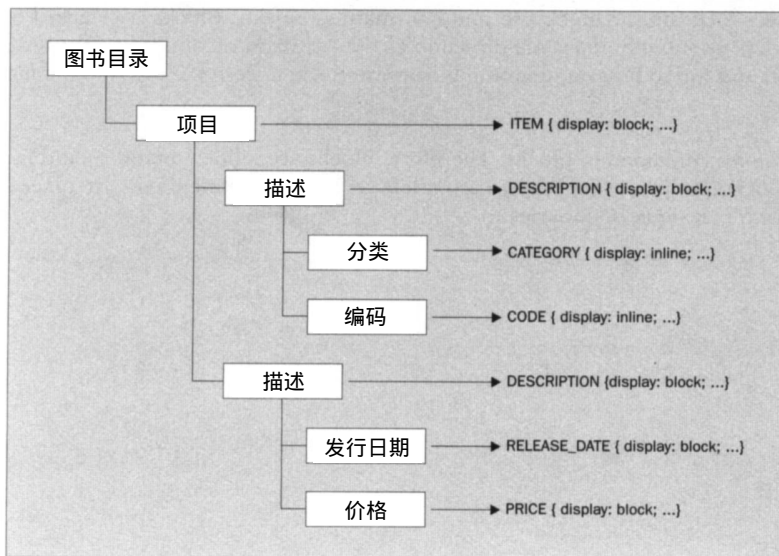


图 13-6

现在我们看看块和内嵌对象是什么。

3. CSS视觉模型

在CSS中有两种主要的格式对象：

- 块。
- 内嵌对象。

但是，如果从项目的角度考虑显示内容，则包括的项目有：

- 漂移对象。
- 列表。
- 表格。

我们也可以以表格为例考虑显示内容，因为表格包括一些特定的特征，它有表头、单元格、行和列。但是，在 CSS 中这些对象的模型并不一致，因为我们表述它们的方式是不同的。所以，块对象和内嵌流对象被看作是 display 属性的值，而漂移对象本身就是一个属性，它有不同的值。因此，CSS 中的语法与视觉模型中的不同。

- 块：值（显示属性）。
- 内嵌：值（显示属性）。
- 表格：值（显示属性）。
- 列表：属性。
- 漂移：属性。

现在我们集中讨论块对象和内嵌格式对象。块是一块块上下叠加起来的。堆栈方向与书写的方向一样。因此，如果书写的方向是从左到右，从上到下，则块也从上到下在载体矩形中堆叠。

内嵌对象是在块中包括的盒子。因此，块是内嵌载体，而内嵌对象包含在块中。如果书写方式是从左到右，内嵌盒子也在块的盒子边界里顺序堆放如图 13-7 所示。

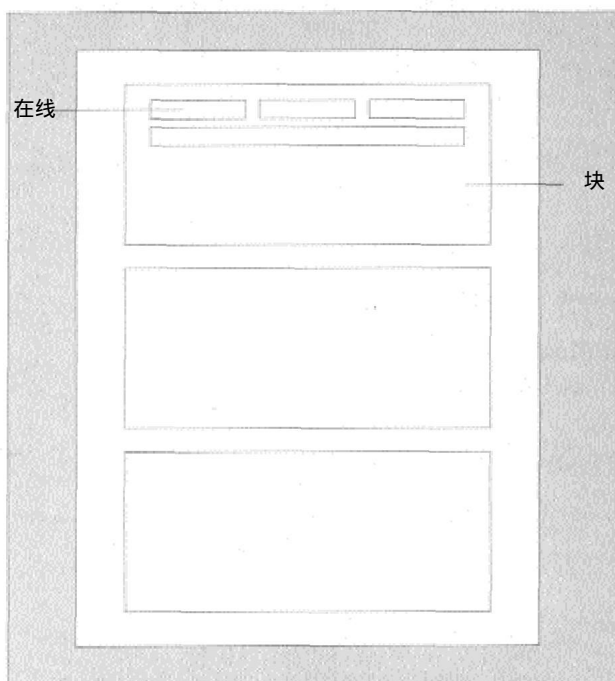


图 13-7

大部分样式语言都以一个流解释模型为基础。在这种情况下，格式对象只是一个接一个显示，按照书写方向进行解释。CSS 支持这种模型，但它也支持绝对位置。一个盒子可以在一个确

定位置，或者摆放在上一个显示的盒子后。

固定位置显示的文件通常用于显示表格，表格中的域以绝对的位置显示（参见图 13-8）。

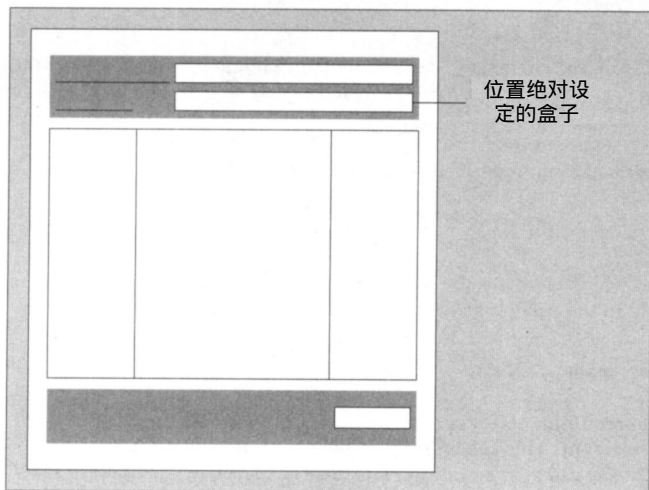


图 13-8

为了在传送页面中包括说明或图像，可以设定一个带漂移属性的元素，这个元素生成紧靠载体盒子的左边或右边的一个特定盒子。格式流将内容排列于飘移对象周围，如图 13-9所示。



图 13-9

4. CSS规则结构

如果你用过C或C++语言，CSS结构看起来会很熟悉，虽然相似的地方只包括两个方面：规则是用{ }符号括起来的，每个属性行用分号结束。

一个样式表包括许多规则。每个规则的语法如下：

程序清单 13-2

```
Selector { properties }
```

选项说明了属性要与哪个元素匹配并应用到这个元素上，这些属性括在花括号里，语法规式如下：

程序清单 13-3

```
Property = attribute: value;
```

注意即使解释对象也以一对属性和值表示，而不管这个对象是块、表格还是内嵌元素。因此对每个元素都要指定对象类别。综上所述，当使用CSS样式语言时，我们把属性和元素联系起来。选择项（CSS规则中的模式匹配部分）表示了与规则相匹配的元素或元素集，而与元素相关联的属性（通过选择项连接）却括在花括号里。

5. 模式匹配选项

任何一个CSS规则都以选项开始。选项是模式匹配的表达方式，它将一个特定的XML元素链接到某个特定的规则上。当CSS引擎遇到匹配选项的元素时，规则就失效了。规则失效仅仅意味着一个解释对象生成了，这个对象的属性由规则内容来设定。表 13-2列出了CSS2支持的选择项类型集：

表 13-2

模 式	含 义
*	同任意元素匹配
E	同任意E元素匹配
E, F	同任意E元素或F元素匹配
E F	同位于E元素下面的任意F元素匹配
E > F	如果F元素是E元素的子元素，则同F元素匹配
E: first - child	如果E元素是其父元素的第一个子元素，则同这个E元素匹配
E: link	如果E元素是一个超链接的源，则同E元素匹配，这个超链接的目标可能还没有被访问过（:link），或者已经被访问过（:visited）
E: visited	
E: active	在某些用户动作过程中同E匹配
E: hover	
E: focus	
E: lang(c)	如果元素E在（人类）语言c中，则同元素E匹配（文件语言说明如何确定语言）
E + F	同元素E后面紧跟着的元素F匹配
E[foo]	同带有foo属性集的元素E匹配（无论其值是什么）

(续)

模 式	含 义
E[foo = "warning"]	同foo属性值为“warning”的元素E匹配
E[foo~="warning"]	如果元素E的foo属性值为一系列给定值中的一个(其中一个值为“warning”),则同这个元素E匹配
E[lang = "en"]	如果元素E的属性lang值为以“en”开头的(从左边开始)用连字符分隔开的一系列值,则同元素E匹配
E#myid	同ID等于myid的元素E匹配

一个选项是由一串一个或多个简单选项组成的。CSS2说明中提到:

“一个简单选项是一个类型选项或通用选项,后面紧跟零个或多个属性选项、ID选项或伪等级,它们的顺序可任意排列。如果简单选项的所有成分都匹配,则简单选项匹配。”

一个类型选项就只是一个XML元素名称,只要XML文件树上出现这个元素,这个选项就和元素匹配了。通用选项是一个“*”字符,表明它与任何元素都匹配。

属性选项允许更细致的选择或方式匹配,它允许细到属性级的选择,甚至是属性值这一级。例如,如果一个元素<TITLE>的language属性设定为“English”,则下列规则适用于这个元素:

程序清单 13-4

```
TITLE {display: block;}
TITLE[language] {display: block;}
TITLE[language="English"] {display: block;}
```

第一个规则与所有的<TITLE>元素相匹配,甚至包括那些没有language属性的元素,并以块级元素显示它们。第二个规则与所有的具有language属性的元素相匹配,而不管其属性。最后,第三个规则只与language属性值为“English”的<TITLE>元素相匹配。

用ID选项可以实现更细致的匹配。ID选项允许规则与具有一个特定ID的某一特定元素相匹配。

通常情况下,CSS规则基于一个元素在文件树上的位置与其链接。但伪等级选项允许访问文件树上其他类型的结点。例如,伪等级first-child允许我们访问一个特定元素的第一个子元素,并将一个规则与其链接:

程序清单 13-5

```
TITLE: first-child {...}
```

我们可以将一个特定的CSS规则与文件树上的一个对象相关联,而不需要明确指定对象的名称。

伪元素是另一种类型的有趣选项。例如,有时很需要将段落中的第一行改为不同的格式,或将一章中的第一个字母设定为更大的字体。在这里,选定元素<DESCRIPTION>数据内容的第一行,将规则的一系列属性应用到第一行上:

程序清单 13-6

```
DESCRIPTION: first-line {...}
```

但在下面的代码中，元素<DESCRIPTION>数据内容的第一个字母与规则的属性集相关联：

程序清单 13-7

```
DESCRIPTION: first-letter {...}
```

只要每个简单选项都被空格、>或+分开，简单选项可以组成一个简单选项表达式。
下面的表达式：

程序清单 13-8

```
ITEM, TITLE
```

是一个与元素<ITEM>或<TITLE>相匹配的选项。因此，逗号可以看作是一个内含的或逻辑的表达式。

下面的表达式：

程序清单 13-9

```
ITEM > TITLE
```

只有元素<ITEM>的子元素<TITLE>才能与其匹配。因此，如果在文件类型或文件结构中的一个其他位置有元素<TITLE>，则不能满足规则。例如，如果在元素<DOCUMENT>的子元素中有元素<TITLE>，这个元素<TITLE>不会和具有这类选项的规则相匹配。

最后，选项：

程序清单 13-10

```
ITEM + TITLE
```

只与元素<ITEM>前面相邻的元素<TITLE>相匹配。例如，如果元素<CATEGORY>在元素<TITLE>前面，则规则没有匹配的元素。

我们也可以使用一组由逗号隔开的选项来匹配一些元素，如：

程序清单 13-11

```
TITLE [language="English"], ITEM, DOCUMENT > BOOK  
{  
    ...  
}
```

当CSS样式表用于HTML文档时，选项不区分大小写。但当CSS样式表用于传递XML文档时，CSS选项区分大小写 因为选项在一个区分大小写的语言环境中使用。

本章不能过多地讨论所有的技术细节，但这些内容在附录 F中都有。

6. 媒介类型和模块化样式表

XML文档的一个主要特性就是数据与显示方法的分离。一个 CSS样式表就是用于显示XML文档的一种方式。CSS允许对几种类型的媒介进行解释：

表 13-3

媒介类型	描 述
screen	一个屏幕设备，例如一个浏览器
print	一个打印产品如一本印刷的书
aural	一个语音设备，例如一个声音合成器
Braille	一个带触觉反馈的 Braille 设备
embossed	一个印花设备，像一个分页 Braille 打印机生成的一样
projection	一个投影设备
tty	一个电传打印机
tv	一台电视

一个 CSS 样式表可以包含几种媒介的样式说明，如一个样式说明可以为浏览器、打印机以及演讲设备提供服务。用 `@media` 命令指定每个目标产品。下例中将 CSS 规则同屏幕产品（通常是浏览器）相关联：

程序清单 13-12

```
@media screen { BOOKLIST {display :block;} }
```

如上所示，所有以某一特定类型的媒介为目标的规则都包括在 `@media` 语句中。同样选项也可以指定一些元素具有相同的规则，许多媒介类型都可以使用选项：

程序清单 13-13

```
@media screen, print { BOOKLIST {display: block;} }
```

但有时，有必要将一个样式表分成几个文件，使整体的解释说明更模块化，例如，将打印样式表存储在一个文件中，而将语音样式表存储在另一个文件中。

`@import` 命令允许你从其他样式表中引入规则。在一个样式表文件中命令 `@import` 应该位于其他所有的 CSS 命令之前。引入的样式表可以参照 URL：

程序清单 13-14

```
@import url(booklist_aural.css);
```

可以将命令 `@media` 和 `@import` 在同一表达式中合并，如下例所示：

程序清单 13-15

```
@import url(booklist1.css) aural;
```

在这个表达式中，`booklist1.css` 样式表是从外输入的，并包含在原有的命令 `@media` 中，在这里 `@media` 的类型是“aural”。媒介类型添加在输入命令引用上（参见图 13-10）：

7. 一个 CSS 样式脚本实例

前面论述的是理论，下面我们看一个实例。在这一章里，我们将使用同一个 XML 文档，分别用 CSS、DSSSL 和 XSL 实现。这个 XML 文档就是我们在本书其他部分里所使用的叫

booklist.xml的一个图书目录，只是略有改动：

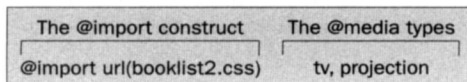


图 13-10

程序清单 13-16

```
<?xml version="1.0"?>
<BOOKLIST>
  <ITEM>
    <CODE>16-048</CODE>
    <CATEGORY>Scripting</CATEGORY>
    <RELEASE_DATE>1998-04-21</RELEASE_DATE>
    <TITLE>Instant JavaScript</TITLE>
    <PRICE>$49.34</PRICE>
  </ITEM>
  <ITEM>
    <CODE>16-105</CODE>
    <CATEGORY>ASP</CATEGORY>
    <RELEASE_DATE>1998-05-10</RELEASE_DATE>
    <TITLE>Instant Active Server Pages</TITLE>
    <PRICE>$23.45</PRICE>
  </ITEM>
  <ITEM>
    <CODE>16-041</CODE>
    <CATEGORY>HTML</CATEGORY>
    <RELEASE_DATE>1998-03-07</RELEASE_DATE>
    <TITLE>Instant HTML</TITLE>
    <PRICE>$34.23</PRICE>
  </ITEM>
</BOOKLIST>
```

在本章里我们将使用同一种解释，所有运行的脚本都应该显示同一结果，如图 13-11所示：



图 13-11

但是要想不对源文档做任何转换就从图书目录的例子得到上面的结果是不可能的，原因主要有两个：

- 我们应该对文档的元素进行重新排序
- 为了解释的需要，要加入一些信息

例如，书的标题应该位于每个分类项目的开始。此外，还应该添加 Category、Release date 和 Price 字符串来帮助读者明确每个信息类型。

我们不能使用 CSS 修改待解释的 XML 源文档，所以 XML 在按样式表显示以前要转换。这意味着我们要对元素重新排序，并在内容上添加一些新的字符串。XSLT 的转换功能十分胜任这项任务。

我们假设文档已经转换成下列 XML 文档（前面第 9 章中要求的一样），我们称这个文档为 trans_booklist.xml：

程序清单 13-17

```
<?xml version="1.0"?>
<?xml-stylesheet
  type="text/css"
  href="booklist.css"
  media="screen"?>
<BOOKLIST>
  <ITEM>
    <TITLE>Instant JavaScript</TITLE>
    <DESCRIPTION>
      <CATEGORY>Category: </CATEGORY>
      <CODE>(16-048)</CODE>
    </DESCRIPTION>
    <DESCRIPTION>
      <RELEASE_DATE>Release date: 1998-04-21</RELEASE_DATE>
      <PRICE>Price: $49.34</PRICE>
    </DESCRIPTION>
  </ITEM>
  <ITEM>
    <TITLE>Instant Active Server Pages</TITLE>
    <DESCRIPTION>
      <CATEGORY>Category: ASP</CATEGORY>
      <CODE>(16-105)</CODE>
    </DESCRIPTION>
    <DESCRIPTION>
      <RELEASE_DATE>release date: 1998-05-10</RELEASE_DATE>
      <PRICE>Price: $23.45</PRICE>
    </DESCRIPTION>
  </ITEM>
  <ITEM>
    <TITLE>Instant HTML</TITLE>
    <DESCRIPTION>
      <CATEGORY>Category: HTML</CATEGORY>
      <CODE>(16-041)</CODE>
    </DESCRIPTION>
    <DESCRIPTION>
      <RELEASE_DATE>release date: 1998-03-07</RELEASE_DATE>
```

```
<PRICE>Price: $34.23</PRICE>
</DESCRIPTION>
</ITEM>
</BOOKLIST>
```

首先，CSS样式表与XML文档的关联是通过xmlstylesheet处理规则来完成的：

程序清单 13-18

```
<?xml-stylesheet type="text/css" href="booklist.css" media="screen"?>
```

第一个属性type指定了样式表的类型，但如果是CSS样式表，应该设定为CSS MIME类型，即“text/css”。第二个属性href是指向样式表文档位置的链接。这个属性应该设定为一个URI（更多情况下是一个URL）。最后，末尾一个属性告诉CSS引擎解释产品是一个屏幕产品。对于一台打印机，我们应该把属性media设为“print”。

因为Microsoft Internet Explorer 5支持用CSS使XML文档样式化，因此可以实验将示范的XML文档与下面的CSS样式表相链接的结果，你将得到同上一个屏幕拷贝相同的结果。这就是我们要用来确定显示数据格式的CSS文件，叫做booklist.css，可以从<http://www.wrox.com/>上下载，还可以同时下载本书中的其他代码：

程序清单 13-19

```
@media screen, print
{
    ITEM
    {
        display: block;
        margin-left: 40pt;
        font-family: Times New Roman;
        font-size: 12pt;
        font-weight: 500;
        margin-bottom: 15pt;
        text-align: left;
        line-height: 12pt;
        text-indent: 0pt;
    }

    TITLE
    {
        display: block;
        font-family: Arial;
        font-weight: 700;
        font-size: 14pt;
    }

    DESCRIPTION
    {
        display: block;
    }

    CATEGORY
    {
        display: inline;
```

```

}

CODE
{
    display: inline;
}

RELEASE_DATE
{
    display: inline;
}

PRICE
{
    display: inline;
}
}

```

因为样式表处理语句与这个 CSS 相链接，并且 XML 已经转换成包括其他文本的文档，处理结果将会与我们在这一部分开始部分见到的完全相同。下面，我们再讨论一些其他 CSS 语法。

8. display 属性

display 属性告诉 CSS 解析器格式对象应该是一个块对象、内嵌对象还是一个表格。解释的元素被自动赋给一个缺省值。例如，一个具有 display 属性的元素将作为一个段落对待，格式流在这个格式对象后断开。但是，如果具有 display 属性的元素设定为 inline，则这个元素作为段落中一个语句或一个单词对待，并且解释后不断开格式流。最后，如果一个元素的 display 属性值为 table，它将设定一个表格所有行和列的排列信息。一个表格可能也会中断正常的格式流。所以，总结三个 display 的基本取值：

- block 作为段落，引起正常解释流的中断。
- inline 作为一个单词或一个语句，内嵌对象一个接一个排列（水平方向），不引起正常解释流的中断。
- table 设定一个表格区域，可能引起或不引起正常解释流的中断。

CSS 表格模型基于一个表格区域，可能引起或不引起正常解释流的中断；一个表格用行和列定义，但列是从行衍生而来的。一个行被分成单元格，当定义行时，单元格组成列，所以第一个单元格位于第一列，第二个单元格位于第二列，依此类推。

要生成表格或表格部分，显示属性可能会取表 13-4 中的值：

表 13-4

值	描 述
table	生成一个带块行为的盒子。将会产生一个格式流中断
table-inline	生成一个带内嵌行为的盒子。不会产生格式流中断
table-row	指明一个元素是一行单元
table-row-group	指明一个元素由一个或多个行组成
table-header-group	生成一个表头。一个表格标题组总是在其他行或组以前显示
table-footer-group	生成一个表格标注，一个表格标注总是在其他行或组之后显示
table-cell	指明一个元素代表一个表格的单元
table-caption	指明一个表格的标题

为了演示表格的显示格式，我们对前面范例 XML文档中的元素在转换前进行排列，然后以表格格式显示。

在写本书时，没有浏览器能显示用 CSS 2排列的表格，所以图 13-12仅仅是提供一个参考，看看文件被范例CSS样式表处理后的最终解释结果（实际上这个图是在 HTML 4.0中生成的，因为二者的模型很相似，在CSS样式表后我们将讨论HTML代码）。

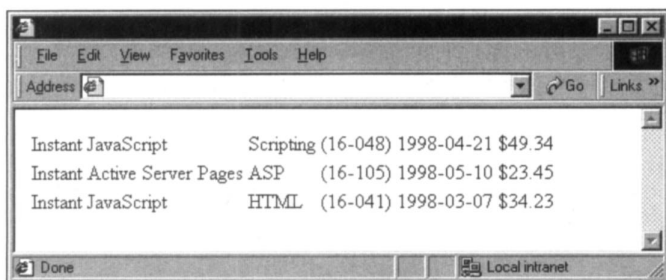


图 13-12

这是生成表格的样式表：

程序清单 13-20

```
@media screen
{
    BOOKLIST      {display: table;}
    ITEM          {display: table-row;}
    TITLE         {display: table-cell;}
    CATEGORY      {display: table-cell;}
    CODE          {display: table-cell;}
    RELEASE_DATE  {display: table-cell;}
    PRICE         {display: table-cell;}
}
```

上面的CSS样式表生成了一个包含 3行5列的表格。元素<BOOKLIST>生成一个table格式对象，其匹配的规则带有一对 display:table的属性/值。行由单元格填充，而单元格是由与元素<TITLE>、<CATEGORY>、<CODE>、<RELEASE_DATE>和<PRICE>相关的display:table-cell属性/值生成的。

你可能注意到CSS模型与HTML 4.0模型十分相似。下面用HTML生成相同的表格：

程序清单 13-21

```
<HTML>
<BODY>
  <TABLE>
    <TR>
      <TD>Instant JavaScript</TD>
      <TD>Scripting</TD>
      <TD>(16-048)</TD>
      <TD>1998-04-21</TD>
      <TD>$49.34</TD>
```



```
</TR>
<TR>
  <TD>Instant Active Server Pages</TD>
  <TD>ASP</TD>
  <TD>(16-105)</TD>
  <TD>1998-05-10</TD>
  <TD>$23.45</TD>
</TR>
<TR>
  <TD>Instant JavaScript</TD>
  <TD>HTML</TD>
  <TD>(16-041)</TD>
  <TD>1998-03-07</TD>
  <TD>$34.23</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

9. position属性

position属性确定盒子的固定位置。只要 position属性值不为static，元素的盒子就不随正常的解释流移动。它的位置将相对固定在一个盒子载体中，其位置和上一个显示的盒子位置无关。位置属性的优先级高于缺省块行为。

位置属性可以设定为一系列不同的值（参见表 13-5）。

表 13-5

值	描 述
static	缺省行为；正常流
relative	根据正常流计算盒子的位置。它的位置相对于正常流的距离用left、right、top和bottom属性来确定
absolute	盒子的位置用 left、right、top和bottom属性确定。用绝对值确定位置的盒子位于正常流之外。盒子相对于载体块的位置放置
fixed	同absolute的性质一样，只是盒子不能移动。例如，在一个屏幕中，当页面滚动时，盒子所处的位置相对于观察点是不动的，在视觉浏览器中，这个观察点就是浏览器窗口

值得注意的一点是绝对位置并不意味着载体块的文本要在用绝对值设定位置的对象周围分布。下面的例子 sample_position.xml将证明这一点：

程序清单 13-22

```
<?xml version="1.0"?>
<?xml-stylesheet href="sample_position.css" type="text/css" media="screen"?>
<SAMPLE_POSITION>
  <TEXT>The position property imposes a fixed position to a box.
  When the position property is set to anything except static,
  the element's box doesn't follow the normal rendition flow. Its
  position will be fixed relatively to a box container, and not
  relative to the last displayed box - the default block behavior
  is overridden.
</TEXT>
```

```
<OVERLAP>
  box with position: absolute
</OVERLAP>
</SAMPLE_POSITION>
```

这是与其相关联的样式表 sample_position.css :

程序清单 13-23

```
TEXT
{
  display: block;
  font-family: Arial;
  font-size: 10pt;
}

OVERLAP
{
  position: absolute;
  left: 60pt;
  top: 20pt;
  background-color: white;
  font-weight: 700;
  font-size: 18 pt;
}
```

上面的例子并不吸引人，但下面的图 13-13 将会清楚地显示 CSS 中绝对位置的含义。元素 `<OVERLAP>` 的数据内容用固定位置（相对于浏览器的上边界和左边界）显示。

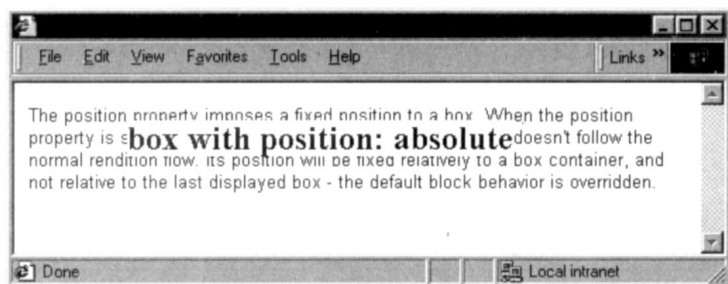


图 13-13

即使在文件 sample_position.xml 中元素 `<TEXT>` 是位于元素 `<OVERLAP>` 之前，元素 `<OVERLAP>` 的内容也在元素 `<TEXT>` 的数据内容上面显示。与缺省格式流无关，和元素 `<OVERLAP>` 相关联的盒子在绝对位置显示。在一个浏览器中，绝对位置相对于文件边界的上部和左部设定。

如果我们使用的是相对位置属性，则元素 `<OVERLAP>` 的数据内容的显示位置将相对于元素 `<TEXT>` 的数据内容（用一个 CSS 规则转换为一个块对象）的下边界和左边界。这样，`<TEXT>` 的数据内容就不在 `<OVERLAP>` 数据内容的中间显示，而是在 `<OVERLAP>` 数据内容之后，用 `left` 和 `top` 属性确定相对距离。

在下面的程序中我们对前面的 XML 文档样式表处理规则进行了改写，并将改后的程序叫做

sample_position2.xml :

程序清单 13-24

```
<?xml version="1.0"?>
<?xml-stylesheet href="sample_position2.css" type="text/css" media="screen"?>
<SAMPLE_POSITION>
  <TEXT>The position property imposes a fixed position to a box.
    When the position property is set to anything except static,
    the element's box doesn't follow the normal rendition flow. Its
    position will be fixed relatively to a box container, and not
    relative to the last displayed box - the default block behavior
    is overridden.
  </TEXT>
  <OVERLAP>
    box with position: relative
  </OVERLAP>
</SAMPLE_POSITION>
```

下面是与上述文档相关联的样式表 sample_position2.css :

程序清单 13-25

```
TEXT
{
  display: block;
  font-family: Arial;
  font-size: 10pt;
}

OVERLAP
{
  position: relative;
  left: 60pt;
  top: 20pt;
  background-color: white;
  font-weight: 700;
  font-size: 18 pt;
}
```

它们的结果参见图 13-14。

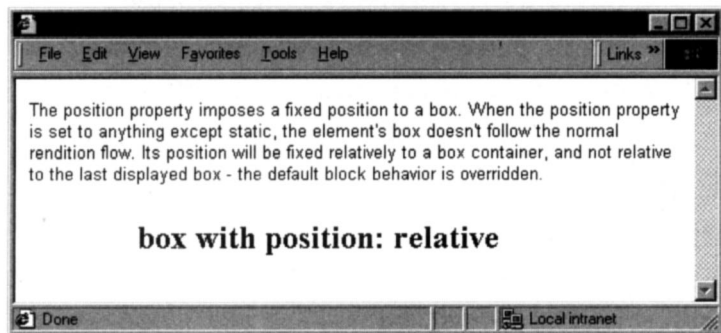


图 13-14

图13-15显示了三种类型的位置。

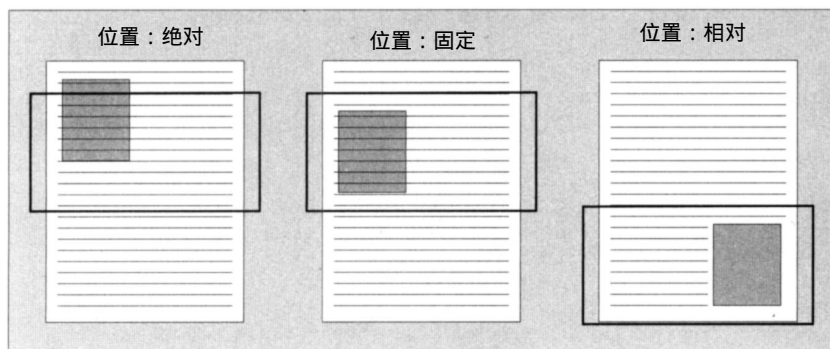


图 13-15

在第一张图（左边）中你可以看到，灰色区域是一个转换为格式对象的元素，位置设定为绝对位置。因此，它的位置与文档起始位置的距离是绝对的。窗口（图中为画在上面的矩形）只显示了元素的一部分，即包含在窗口（观察点）中的部分。因为文档在观察区域是滚动的，所以显示内容也会随着内容滚动而变化。

在第二张图（中间）中，元素的位置固定，因此距离不是相对于文档的左边界和上边界设定的，而是相对于观察窗口的左边界和上边界设定的。当我们滚动文档时，灰色区域总是在同一个地方显示。

最后，最末一张图（右边）表明了灰色区域是处于一个相对位置，其位置相对于上一块区域的左边界和下边界设定。

10. z 顺序

元素不仅可以在一个二维空间中绝对定位，而且可以在三维空间中定位。相信在这里讨论一下层是大有好处的。

想象一个带有几个透明层的传输文件。将你的想象力延伸，想象每个透明层上的盒子。第三维就是透明层的叠加。z轴就是一种叠加的表现形式，盒子在其中可以互相层叠。每个盒子都位于一个堆叠层里，每个盒子都有一个代表堆栈级别的整数（有可能是负数）；这个整数表明它在一堆栈中相对于其他盒子的位置。堆栈级别高的盒子放在级别低的盒子上面。例如，一个堆栈级别为15的盒子比级别为2的盒子距离顶部更近一些。

层可以用来生成动画对象。脚本可以管理 DOM和元素属性，因此要想生成一个带有移动对象的动画文件是完全可能的。例如，一个 XML格式的幻灯片演示可以和一个 CSS样式表一起传送，其中的动画段落可以使演示更精彩。第三维或 z的位置可以用属性 z-index确定，如：

程序清单 13-26

```
OVERLAP
{
    display: block;
    z-index: -1;
    font-size: 10pt;
}
```

如果没有指定的背景颜色，重叠的盒子是透明的。但如果对重叠的盒子指定颜色，则用属性background-color设定显示颜色。

11. float属性

另一个有趣的属性是float属性，它将盒子定位于载体块的左边或右边的某个相对位置上。当元素用float属性设定时，它就贴在载体块的左边或右边。在下图中，两个元素被解释为移动盒子。它们附着在块盒子上，并在相对于块盒子的位置上显示，而不是在包含文件的盒子中显示。所以，不像属性/值为position:absolut的元素，属性/值为float:right或float:left的元素位于当前载体盒子的某个相对位置上，如图13-16所示。

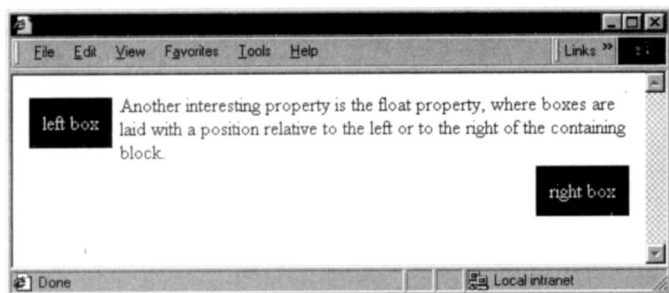


图 13-16

盒子有3个基本属性：margin、border和padding（参见图13-17）。

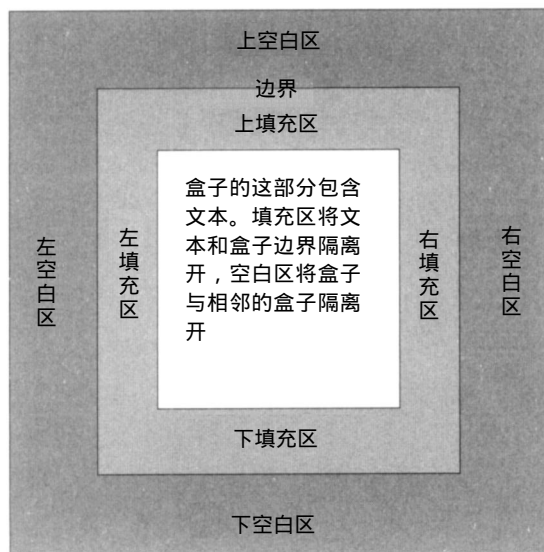


图 13-17

每个盒子都有边界，即使边界是不可见的。边界将盒子与相邻的盒子或画面分开。边界与相邻盒子外边缘的距离，或者边界与载体盒子的距离叫做margin。盒子和盒子边界的距离叫padding。

关于这方面的例子，参考下面的 XML 代码 float_sample.xml：

程序清单 13-27

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="float_sample.css" media="screen"?>
<FLOAT_SAMPLE>
  <PARAGRAPH>
    <LEFT_BOX>
      left box
    </LEFT_BOX>
    Another interesting property is the float property, where
    boxes are laid with a position relative to the left or to the
    right of the containing block.
    <RIGHT_BOX>
      right box
    </RIGHT_BOX>
  </PARAGRAPH>
</FLOAT_SAMPLE>
```

下面是相应的样式表 float_sample.css：

程序清单 13-28

```
LEFT_BOX
{
  float: left;
  margin-left: 2pt;
  margin-top: 2pt;
  margin-right: 2pt;
  margin-bottom: 2pt;
  padding-top: 8pt;
  padding-left: 8pt;
  padding-right: 8pt;
  padding-bottom: 8pt;
  background-color: black;
  color: white;
}

RIGHT_BOX
{
  float: right;
  margin-left: 2pt;
  margin-top: 2pt;
  margin-right: 2pt;
  margin-bottom: 2pt;
  padding-top: 8pt;
  padding-left: 8pt;
  padding-right: 8pt;
  padding-bottom: 8pt;
  background-color: black;
  color: white;
}

PARAGRAPH
{
  display: block;
}
```

解释事件的顺序如下：

段落元素遇到CSS引擎并与带PARAGRAPH选项的规则相匹配。段落内容被设定为一个四周带边框的块对象，但它的内容并不直接显示。

CSS引擎在元素 <PARAGRAPH> 中遇到的第一个元素就是元素 <LEFT_BOX>。元素 <LEFT_BOX> 与带LEFT_BOX选项的规则相匹配。设定图像为一个带边框的移动盒子。移动盒子按float属性的指定方向向左边移动。用属性 margin-top、margin-left、margin-right和margin-bottom设定包围移动对象的边框。用属性 padding-top、padding-left、padding-right和padding-bottom设定显示对象的周围区域。移动盒子的显示位置相对于载体盒子（元素 <PARAGRAPH> 以前生成的块段落）的左边界和上边界确定。

然后CSS引擎得到元素<PARAGRAPH>的数据内容。内容在移动盒子的四周显示。移动盒子和文本之间的距离用属性 margin-right和margin-bottom设定。

12. 交互式行为

Microsoft在Internet Explorer 5中引入了行为扩展。这对CSS来说是一个添加的功能，添加了如高亮度显示文本等交互式功能。在写本书的时候，在 <http://www.w3.org/TR/becss>上提供一个工作草案。我们可以在任何XML元素上添加一个行为。和DHTML文件不同，通常XML文件不包含脚本，因为脚本不符合将数据和表示方式（或交互方式）分隔的概念。为了增加解释文件的交互性可以对任何XML元素加上行为。例如，当鼠标点在解释后的元素 <TITLE>上时要想高亮度显示它，必须像下例一样向元素上添加一个行为：

程序清单 13-29

```
@media screen { TITLE {behavior: url(highlight.htc); }
```

在这种情况下，行为可以采用包含一些 JavaScript的HTML组件（或HTC）的形式。

在最新的行为扩展工作草案中一个有趣的特征是 @script命令。它允许在一个样式表中包括脚本。@script是一个功能强大的命令，它对于那些已经对程序语言如 JavaScript、PerlScript、PythonScript或VBScript有经验的开发者来说十分有用。脚本有什么好处呢？脚本可以完成许多事情，如管理 DOM，以及添加事件处理器到 DOM结点生成的事件上。这些功能在支持DOM Level 1的浏览器上不可能实现，但在 DOM Level 2的工作草案中有支持事件这一部分。可以在<http://www.w3.org/TR/WD-DOM-Level-2/events.html>上找到DOM事件界面的解释文件。通过DOM Level 2界面增加事件处理可能很可笑，但 @script CSS命令将会使事情变得十分容易。

@script命令中包含的或指向的脚本同HTML里<script>元素中的脚本相似。在函数或过程（或对象）中没有包括的代码被放在一个内部“主要”函数中，当第一次装载样式表并解释时执行这些代码。事件处理器可以是函数或程序，它们的名字通常用 On_开头。例如，一个添加到元素<TITLE>上的事件处理器可以命名为 On_TitleClick()。那么，我们如何将事件和事件处理器连接起来呢？可以通过对与元素相关联的规则添加一个新属性来实现。要在元素 <TITLE>上添加一个事件处理器，我们可以对规则添加下列内容：

```
TITLE
{
    display: block;
    color: red;
    OnClick: "On_TitleClick(event)";
}
```

在这种情况下元素 `<TITLE>` 的属性 `OnClick` 指的是 `On_TitleClick()` 函数，这个函数应该在 `@script` 命令中包括。

另一个要点是虽然我们说 CSS 不能更改 XML 文档的结构，但如果执行了 `@script` 命令这一点就不成立了。为什么？因为浏览器可能允许脚本管理 DOM，这就有可能改变文件结构。下面是一个带行为扩展的浏览器所遵循的事件顺序：

- 解释 XML 文档并建立层次结构（这个结构就是用 DOM 访问的内容）。
- 装载和解释 CSS 样式表。
- 提取 `@script` 命令的内容并交给一个脚本处理器（例如一个 JavaScript 解释器）进行解释和执行。
- 脚本解释器首先执行“主要”函数（或内部的“主要”函数）。这个代码可以访问 DOM，还允许 DOM 管理文件结构的每个元素。
- 向结构中的每个元素添加一套属性。通过选项匹配功能完成匹配。
- 当用户与解释文件交流时，中止并调用添加到元素上的事件处理器——即调用在 `@script` 命令中包括的一个程序或函数。元素和事件处理器之间的连接是通过一个事件属性（如 `OnClick`）来实现的。

现在，有了这些新的附加功能，CSS 变得更有意思，其特征有：

- 行为扩展在文档交互和文档结构管理上的方式与 XSL 十分相似，但它使用的是程序语言如 JavaScript、VBScript、PerlScript、PythonScript 或者浏览器所支持的任何语言。
- 简单的原型——将属性添加到元素上。
- 简单的语法。

13. CSS 小结

CSS 是一种极其简单的样式语言，很容易记忆和使用。它的主要优点就是简单。如果你仅仅想解释一个 XML 文档，不需要做任何元素转换，那么 CSS 是最适合的。CSS 的主要缺点是没有复杂的解释模型，并且其应用依赖于要处理的 XML 文档结构。这也意味着处理程序必须明白 XML

而许多浏览器都不是这样的。如果你需要一种不太依赖于文档结果和解释格式的语言，则 XSL 和 DSSSL 很合适。不过，目前正在进行的工作将使 CSS 语言增加一些诱人的功能和附件。

下面我们将集中讨论本章中要用到的另一种主要样式语言：XSL。

13.3.6 XSL

同 CSS 相比，扩展样式表语言使用一种完全不同的方法解释 XML 文件，使 XML 文件变成可显示的对象。XSL 更灵活，可以将 XML 转换成适用于不同媒体的语言。在第 9 章中我们讨论

XML转换时曾提到过 XSLT。在本章里我们将讲述如何用 XSLT将XML文件转换成HTML格式以及其他的XML文件结构。

将XML转换成HTML是在Web上显示XML的一种有效方法，因为这意味着我们不只为使用最新浏览器的客户提供页面。这个转换过程在服务器上完成。而且，像我们在本章前面所说的一样，有时我们需要生成一个XML文档的其他文档类型或文档结构，在这里XSL的转换功能就十分适合完成这项任务。实际上我们用XSLT对第9章中的样本XML文档进行元素重排，这样就可以用一个CSS样式表表示这个文档。所以在本章的其余部分里，我们将讨论如何使用XSL生成不同的表示格式。

如同我们在本章开始时说的一样，样式不仅仅是关于如何在Web上显示一个文档。如果我们要适应不同的应用需求而改写XML内容，那么必须能在同一个源文档的基础上生成不同的格式文档。所以，下面我们看看如何用XSL显示样本XML文档，使其分别用于在线显示、打印以及语音浏览器。

XSL现在分为三种不同的规范：

- XPath <http://www.w3.org/TR/xpath>。
- XSLT <http://www.w3.org/TR/xslt>。
- XSLF <http://www.w3.org/TR/xsl>。

XSLT说明着重于XML文档转换，而XSLF说明着重于格式对象，XPath则着重于从XML层次结构上访问结点。极少数情况下，XSL引擎会支持转换部分(XSLT)和XPath。在浏览器中(如IE 5)XSLT也许比XSLF更普遍，因为它们的语言是基于XML的。XSL的格式对象在打印格式化器中生成，如FOP(<http://www.jtauber.com/fop/>)或RenderX(<http://www.renderx.com/>)，这两者都能将XML文档转换成PDF。当需要将文档传输到打印介质上时，这些格式转换是十分有用的。

我们在本章开始时提到过，解释XML不仅仅是准备将XML文档使其能在Web上浏览。现在，XSL可以为打印和在线解释提供服务，也可以将XML文档转换成其他的标记语言，如用于语音浏览的VOXML。所以下面我们看看如何使用这些应用功能。

1. XSL转换

在第9章中我们讨论了XML文档如何转换成HTML，或其他的XML文档结构(如XHTML，其规范可以在<http://www.w3.org/TR/xhtml1/>上找到)。用XSLT解释不依赖于文档结构。再举一个例子，如果一个关系数据库服务器对一个查询的返回结果是一个XML文档，则在解释前要用一个样式表生成页面对象或重新排列元素的顺序。

此外，我们可以用生成HTML以及其他XML文档结构的转换规则来生成其他语言的文件，如VOXML(<http://www.voxml.com>) 用于语音解释的一个XML应用程序。这些转换在XSL规范中的XSLT部分可以找到(<http://www.w3.org/TR/xslt>)。

2. XSL格式化对象

当我们处理打印对象时，XML文档可以转换成XSL格式化对象。在实际中XSL格式化对象的部分再转换成另一种格式，如PDF，这个过程是单独进行的，并且采用了XSL的一个不同部分：XSLF。我们必须指出的是XSL格式对象不仅可以转换成PDF，也可以转换成一些其他的新

格式 如TeX、RTF、MIF等。XSLF格式对象同DSSSL格式对象一样，不依赖于某一特定的解释或格式模型。因此，也同DSSSL一样，一个XSLF引擎可以仅用一个样式表就将XML文档转换成不同的输出格式。

XSL是可扩展的，它的名字已经表示了它的扩展性。这表明可以很容易地将格式对象加入到已经存在的语言中，而没有CSS的限制 固定的格式化对象集。XSL格式化对象方面的技术还需要进行很多工作，在写本章的时候，还没有形成一个W3C建议标准。最新的XSL工作草案可以在<http://www.w3.org/TR/WD-xsl/>上得到。

3. XSLF如何工作

同其他的规则语言一样，XSL文档（无论是XSLT还是XSLF）是由模式匹配部分和行动部分组成的规则集合。在XSL中这些规则叫做模板，所以一个XSL样式表的基本元素是一个模板。XSL模板的模式匹配部分被称为一个XPath表达式（见第8章和第9章）。

一旦XPath已经选定了一个文件结点，它就与模板的内容相关联。因为要将XML文档转换成一个XSL格式化对象的结果树，所以需要将原始XML文档中的每个元素匹配到一个特定的模板上。并且，当要求转换原始文件时，可以用几个XSLT命令和XPath表达式一起访问转换成层次结构的XML文档的任何一个结点。

一个XPath表达式允许我们访问任何文件结点。XSLT命令允许我们对结点和含XSL格式对象的模板进行匹配。

为了更好地理解这个过程，我们讨论一下一个XSL引擎的内部组成；

如同第9章中提到的，一个使用DOM的XSL处理器生成三个树：

- 一个源树，包含有待解释的XML源文档。
- 一个包含XSL样式表的树。
- 一个结果树，用来操作结果文档。

当遇到一个显式或隐式的apply-template命令时，处理器将为树上包含样式表的每一个模板在源文件树上找到一个匹配的结点。当进行这项匹配时，模板的内容作为写往输出树的结果的基本内容。例如，如果目标是解释浏览器中的文档，那么模板中有可能包含HTML命令。对于打印设备，输出将是XSL格式对象，XSL格式对象可以用XSL的打印格式转换成图形对象，其中打印格式化器前面已经提到过，如FOP，能够生成PDF对象。

13.3.7 用XSL解释XML

在这一部分里我们将讨论如何把作为范例的XML图书目录转换成HTML格式、用VOXML标记的语音格式以及用PDF的视觉表示格式。我们在第9章中详细讨论了用XSLT转换XML文档的过程，所以这一部分将集中于如何将XML文档显示给用户，而不是转换过程。

在下面3个例子中，我们使用下面的XML文档：

程序清单 13-31

```
<?xml version="1.0"?>
<BOOKLIST>
  <ITEM>
    <CODE>16-048</CODE>
```

```

<CATEGORY>Scripting</CATEGORY>
<RELEASE_DATE>1998-04-21</RELEASE_DATE>
<TITLE>Instant JavaScript</TITLE>
<PRICE>$49.34</PRICE>
</ITEM>
<ITEM>
  <CODE>16-105</CODE>
  <CATEGORY>ASP</CATEGORY>
  <RELEASE_DATE>1998-05-10</RELEASE_DATE>
  <TITLE>Instant Active Server Pages</TITLE>
  <PRICE>$23.45</PRICE>
</ITEM>
<ITEM>
  <CODE>16-041</CODE>
  <CATEGORY>HTML</CATEGORY>
  <RELEASE_DATE>1998-03-07</RELEASE_DATE>
  <TITLE>Instant HTML</TITLE>
  <PRICE>$34.23</PRICE>
</ITEM>
</BOOKLIST>

```

那么我们以这个XML文档的HTML版本开始，这个HTML样式是用于浏览器的。

1. 一个带HTML可视解对象的XSLT样式范例

我们在第9章中已经讨论过了用XSLT生成XHTML文档，这个过程主要是格式转换。下面我们将再简单讨论一下HTML文档的生成。用XSL将XML转换成HTML很简单，这项技术可以用在服务器上，使服务器支持那些不能读XML文档的客户端，从而解决了CSS的固有问题之一——只能在支持XML的浏览器上解释XML。

XSL的一个有趣特征是它能以不同的顺序解释文档的元素（我们在转换一章中已经讨论过）。例如，在下面的例子里，即使元素<TITLE>不是元素<ITEM>的第一个子元素，通过XSLT的转换功能它也可以最先显示。下面是一个叫booklist.xsl的例子：

程序清单 13-32

```

<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

  <xsl:template match="/">
    <html xmlns="http://www.w3.org/TR/xhtml1/strict">
      <head>
        <title>The book catalog</title>
      </head>
      <body>
        <xsl:apply-templates select="BOOKLIST/ITEM" />
      </body>
    </html>
  </xsl:template>

  <xsl:template match="ITEM">
    <xsl:apply-templates select="TITLE" />
    <DIV style="margin-left: 40pt;
      font-family: Times New Roman;
      font-size: 12pt;
      font-weight: 500;

```



```

margin-bottom: 15pt;
text-align: left;
line-height: 12pt;
text-indent: 0pt;" >
<DIV>
  <SPAN>Category: </SPAN>
  <SPAN>
    <xsl:value-of select="CATEGORY"/>
  </SPAN>
  <SPAN> (</SPAN>
  <SPAN>
    <xsl:value-of select="CODE"/>
  </SPAN>
  <SPAN>)</SPAN>
</DIV>
<DIV>
  <SPAN>Release date: </SPAN>
  <SPAN>
    <xsl:value-of select="RELEASE_DATE"/>
  </SPAN>
  <SPAN> - Price: </SPAN>
  <SPAN>
    <xsl:value-of select="PRICE"/>
  </SPAN>
</DIV>
</DIV>
</xsl:template>

<xsl:template match="TITLE">
  <DIV style="margin-left: 40pt;
    font-family: Arial;
    font-weight: 700;
    font-size: 14pt;" >
    <SPAN>
      <xsl:value-of select="." />
    </SPAN>
  </DIV>
</xsl:template>

</xsl:stylesheet>

```

上面的XSLT样式表可以用IE 5测试，在样本XML文档中<?xml version = "1.0"?>后面加一个处理命令。如果你在文件booklist.xml中输入如下处理命令：

程序清单 13-33

```
<?xml-stylesheet type="text/xsl" href="booklist.xsl" media="screen"?>
```

IE 5将会显示如图13-18所示结果。

注意虽然这个样式表在IE 5中有效，但它所支持的XSL版本同现在讨论的工作草案并不一样，上例也是根据这个工作草案编写的。下面，我们来看看这个样式表的作用。样式表文档由一系列模板构成，每个模板用标志<xsl:template>开头，以标志</xsl:template>结束。xsl:部分是XSL的命名空间，用于防止应用处理程序将<template>标记和特定XML词汇表中使用的具有相同名字的其他元素相混淆。所有的XSL元素都使用这个命名空间标识。

第一个模板元素使用了match属性，match属性指定模板适用的元素。这是一个可选项。在

这种情况下，选项的值是XML文档的根元素，用正斜杠表示：

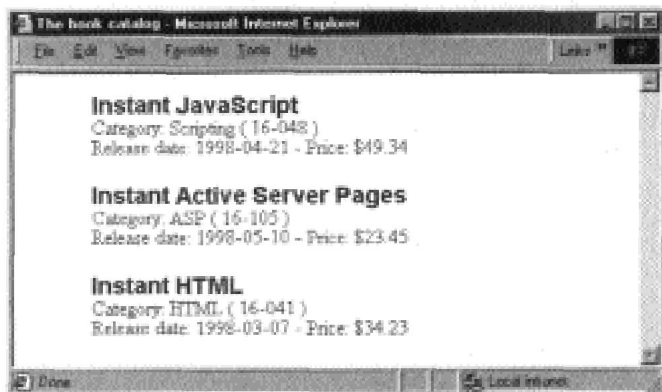


图 13-18

程序清单 13-34

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

这实际上是一个XPath表达式，同XPath工作草案中的一样。

在这个模板中，我们开始生成结果HTML文档，然后跳到xsl:apply-templates命令：

程序清单 13-35

```
<html xmlns="http://www.w3.org/TR/xhtml1/strict">
  <head>
    <title>The book catalog</title>
  </head>
  <body>
    <xsl:apply-templates select="BOOKLIST/ITEM" />
```

上面代码段落中最后一行的命令用来执行在此指定的任何其他模板。XPath表达式“BOOKLIST/ITEM”在这里表示元素<BOOKLIST>的子元素<ITEM>的任何模板都会被匹配。如果你看看第一个模板的剩余部分：

程序清单 13-36

```
  </body>
</html>
</xsl:template>
```

可以看到我们希望在关闭HTML文档以前写下HTML元素的剩余部分。这样，当处理器执行到<xsl:apply-templates/>元素时，就知道要将所有其他模板的内容插入到这个位置上，并用模板的内容替换XSL命令。

第二个模板同所有的<ITEM>元素匹配（像template元素的match属性指定的一样），这些元素在原始XML文档中用apply-templates命令选定：

程序清单 13-37

```
<xsl:template match="ITEM">
  <xsl:apply-templates select="TITLE" />
```

首先应该显示<TITLE>元素，即使在原始文档中<TITLE>元素是<ITEM>元素的第四个子元素。为了显示<TITLE>元素，要执行与这个元素匹配的模板。<TITLE>的模板实际上位于样式表的底部：

程序清单 13-38

```
<xsl:template match="TITLE">
  <DIV style="margin-left: 40pt;
    font-family: Arial;
    font-weight: 700;
    font-size: 14pt;" >
    <SPAN>
      <xsl:value-of select="." />
    </SPAN>
  </DIV>
</xsl:template>
```

这个模板包括了用 CSS 特性建立起来的一个<DIV>元素。这个<DIV>元素包含一个单独的，的内容用元素<xsl: value-of/>设定为同元素<TITLE>的内容一样。<xsl: value-of select = "."/>可以有效地提取匹配 select 属性的元素的值。当 "." 作为 select 属性的一个值时，它表明我们想选择当前匹配结点的值（这里是元素<TITLE>的内容）。因此，整个模板生成了我们显示模块的第一行（同本章中的其他例子采用完全相同的样式）。

执行过这个模板后，处理器将继续执行最后一个模板的剩余部分，从最后一个模板中<apply-templates/>被调用的地方开始执行。

程序清单 13-39

```
<xsl:apply-templates select="TITLE" />
<DIV style="margin-left: 40pt;
  font-family: Times New Roman;
  font-size: 12pt;
  font-weight: 500;
  margin-bottom: 15pt;
  text-align: left;
  line-height: 12pt;
  text-indent: 0pt;" >
  <DIV>
    <SPAN>Category: </SPAN>
    <SPAN>
      <xsl:value-of select="CATEGORY"/>
    </SPAN>
```

这个模板的其余部分生成一个<DIV>元素 不是在另一个模板中，而是在同<ITEM>元素相匹配的模板中。这个最后的<DIV>元素包含内嵌 HTML 元素（如）。一些 HTML 元素只是插入输出结果的字符串，另外一些则是由<xsl: value-of />命令生成的：

程序清单 13-40

```
<SPAN>
  <xsl:value-of select="CATEGORY"/>
</SPAN>
```

这里是提取元素<CATEGORY>中的数据内容（值）而生成的。在这之前，我们对元素的内容进行说明，在本例中是 Category：

程序清单 13-41

```
<SPAN>Category: </SPAN>
```

对代码完成了同样的操作后（只是这次将其放入括号），我们采取和前一个同样的方式添加一个<DIV>元素。这个<DIV>元素中含有元素<RELEASE_DATE>和<PRICE>的内容，并通过<xsl:value-of>命令分别同元素<RELEASE_DATE>和<PRICE>相匹配。

不像CSS脚本，我们对源文档中XML文档元素的显示进行了重新排序，而不必先重新组织XML文档本身。因此，XSL允许更多的解释自由度并且内容和显示也具有更高的独立性。

下面，我们来看看如何生成这个文档的语音解释版本。

2. 一个带VOXML语音解释对象的XSLT样式表示例

由Motorola牵头的VOXML协会提供了一个语音浏览器原型及一个开发者可以进行实验的VOXML SDK。SDK和VOXML项目的相关信息可以在<http://www.voxml.com>上找到。VOXML项目的建立是想为语音应用提供一个通用的和广泛支持的平台，就像HTML为基于Web的应用提供了平台一样。VOXML应用采用了对话的形式。浏览和输入都通过终端用户的声音识别来进行，输出通过文字-语音转换技术或录下来的语音来完成。

VOXML语言基于XML格式，因此一个XSLT文档可以用来将一个XML源文档转换成一个VOXML应用程序。开发一个VOXML应用程序同开发一个基于Web的应用程序一样简单。HTTP服务器可以将VOXML文档传给语音用户。VOXML应用程序在本地进行解译，用户通过一个麦克风和耳机或PC相连的话筒与应用程序进行交互。因此，一个VOXML应用服务器可以就是一个处理XML文档的HTTP服务器。语音浏览器进行语音识别与文字-语音的转换过程。

VOXML应用程序不仅仅局限于语音用户上，它还可以应用到与旧的普通电话系统相连接的VOXML浏览器上。在这种情况下，输入和输出设备是电话（参见图13-19）。

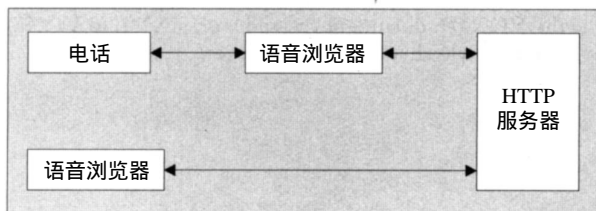


图 13-19

在<http://www.voxml.com/downloads/VOXMLwp.pdf>上有描写这个结构的白皮书和一个简单的例子。

现在，应该在我们的潜在用户和语音浏览器之间建立一个对话了。我们建立的语音应用程序可以用 VOXML SDK 来测试和改进，VOXML SDK 在 VOXML 协会组建的网站 <http://www.voxml.com/login.asp> 上免费提供下载。

源文档可以用 XT 转换成 VOXML，XT 是由 James Clark 建立的一个 XSLT 处理器，可以从 <http://www.jclark.com/xml/xt.html> 上下载（我们在第 9 章中也使用了 XT，附录 G 讲述了如何安装 XT）。生成的 VOXML 文档可以用 VOXML 模拟器来解译。

下图是 VOXML 模拟计算机显示的对话。当将 XML 文档转换为 VOXML 后，你要键入转换文档的 URL。这个文档应该有 .xml 扩展名。你点击进入按钮，然后突然看到 Merlin 在与你对话并等待你的回答。你可以通过麦克风或对话框来回答 Merlin。屏幕上提供了一个文字区域，如果你没有麦克风，则可以用文字来回答（参见图 13-20）。

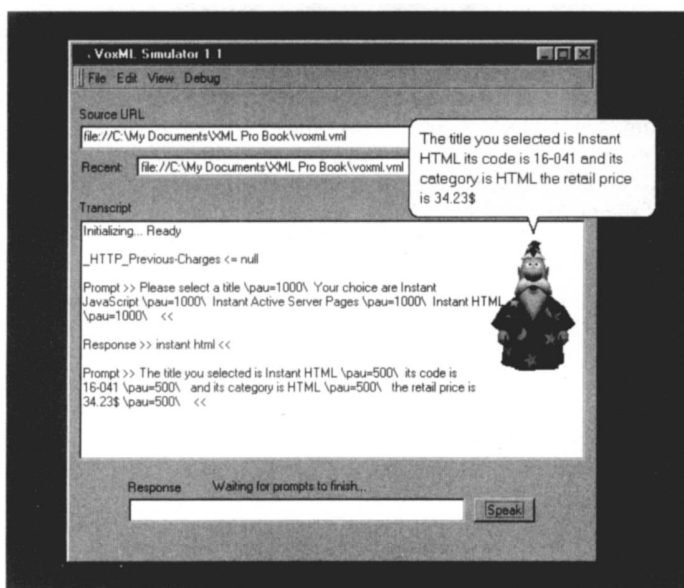


图 13-20

下面是用 XT 处理器将 XML 转换成 VOXML 生成的 VOXML 文档。稍后再讨论样式表。

程序清单 13-42

```
<?xml version="1.0"?>
<DIALOG>
  <STEP NAME="init">
    <PROMPT>
      Please select a title<BREAK MSECS="1000"/>
    </PROMPT>
    <PROMPT>
      Your choices are: Instant JavaScript<BREAK MSECS="1000"/>
      Instant Active Server Pages<BREAK MSECS="1000"/>
      Instant HTML<BREAK MSECS="1000"/>
    </PROMPT>
    <INPUT TYPE="OPTIONLIST">
      <OPTION NEXT="#code:16-048">
```

```

        Instant JavaScript
    </OPTION>
    <OPTION NEXT="#code:16-105">
        Instant Active Server Pages
    </OPTION>
    <OPTION NEXT="#code:16-041">
        Instant HTML
    </OPTION>
</INPUT>
</STEP>
<STEP NAME="code:16-048">
    <PROMPT>
        The title you selected is Instant JavaScript
    <BREAK MSECs="500"/>
    </PROMPT>
    <PROMPT>
        its code is 16-048<BREAK MSECs="500"/>
    </PROMPT>
    <PROMPT>
        its category is Scripting<BREAK MSECs="500"/>
    </PROMPT>
    <PROMPT>
        and the retail price is $49.34<BREAK MSECs="500"/>
    </PROMPT>
</STEP>
<STEP NAME="code:16-105">
    <PROMPT>
        The title you selected is Instant Active Server Pages
    <BREAK MSECs="500"/>
    </PROMPT>
    <PROMPT>
        its code is 16-105<BREAK MSECs="500"/>
    </PROMPT>
    <PROMPT>
        its category is ASP<BREAK MSECs="500"/>
    </PROMPT>
    <PROMPT>
        and the retail price is $23.45<BREAK MSECs="500"/>
    </PROMPT>
</STEP>
<STEP NAME="code:16-041">
    <PROMPT>
        The title you selected is Instant HTML
    <BREAK MSECs="500"/>
    </PROMPT>
    <PROMPT>
        its code is 16-041<BREAK MSECs="500"/>
    </PROMPT>
    <PROMPT>
        its category is HTML<BREAK MSECs="500"/>
    </PROMPT>
    <PROMPT>
        and the retail price is $34.23<BREAK MSECs="500"/>
    </PROMPT>
</STEP>
</DIALOG>

```

这个文档给用户关于图书的三种选择；如果用户选择了一个，它就会告诉用户他们所选择的题目名。

我们准备讨论 VOXML 语言的细节。这些细节在 VOXML SDK 上都有。但是我们将集中讨论叫 booklist_aural.xsl 的 XSLT 样式表：

程序清单 13-43

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml"/>
<xsl:template match="/">
  <DIALOG>
    <xsl:apply-templates/>
  </DIALOG>
</xsl:template>

<xsl:template match="BOOKLIST">
  <STEP NAME="init">
    <PROMPT>Please select a title<BREAK MSECS="1000" /></PROMPT>
    <PROMPT>
      Your choices are
      <xsl:for-each select="ITEM">
        <xsl:value-of select="TITLE"/><BREAK MSECS="1000" />
      </xsl:for-each>
    </PROMPT>
    <INPUT TYPE="OPTIONLIST">
      <xsl:for-each select="ITEM">
        <xsl:element name="OPTION">
          <xsl:attribute name="NEXT">
            #code:<xsl:value-of select="CODE"/>
          </xsl:attribute>
          <xsl:value-of select="TITLE"/>
        </xsl:element>
      </xsl:for-each>
    </INPUT>
  </STEP>
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="ITEM">
  <xsl:element name="STEP">
    <xsl:attribute name="NAME">code:<xsl:value-of select="CODE"/>
  </xsl:attribute>
  <PROMPT>
    The title you selected is
    <xsl:value-of select="TITLE"/>
    <BREAK MSECS="500" />
  </PROMPT>
  <PROMPT>
    its code is
    <xsl:value-of select="CODE"/>
    <BREAK MSECS="500" />
  </PROMPT>
  <PROMPT>
    its category is
```



```

        <xsl:value-of select="CATEGORY" />
        <BREAK MSECs="500" />
    </PROMPT>
    <PROMPT>
        and the retail price is
        <xsl:value-of select="PRICE" />
        <BREAK MSECs="500" />
    </PROMPT>
</xsl:element>
</xsl:template>

</xsl:stylesheet>

```

首先我们用 xsl: output 命令告诉 XSLT 引擎想要的输出类型是 XML。当用 XT 引擎时，可以选择 XML 或 HTML 引擎。

下面我们利用根结点 文档结点对第一个模板进行匹配。由于不能引入另一个 XML 处理指令，我们用元素 xsl: processing-instruction 来包括 <?xml version = "1.0"?> 处理指令。

注意：根据 XSLT 的建议，XML 声明不能用 processing-instruction 命令来完成。用一个 xsl: output 命令并将其 method 属性设为 “xml”，就完全可以生成相应的 XML 声明。由于当前 XT 的局限性，我们在样本模板中包括了 processing-instruction 命令。

我们在 VOXML 引擎调用的 <DIALOG> 元素中包括了 xsl: apply-templates 命令（在上一个例子中我们用 <html> 元素包含了这个命令）。这就要求 XSLT 引擎处理根结点的所有子结点。

程序清单 13-44

```

<DIALOG>
    <xsl:apply-templates/>
</DIALOG>

```

第二个模板同 <BOOKLIST> 元素相匹配。模板中包含一个为 VOXML 引擎生成对话步骤的 <STEP> 元素。<PROMPT> 元素的内容被作为演讲来解析。它对图书目录中提供的标题进行计数。计数是用一个与 <ITEM> 元素相匹配的 xsl: for-each 循环来完成的。每出现一个 <ITEM> 元素，都用 xsl: value-of 命令把 <TITLE> 元素的内容插入 <PROMPT> 元素，并在 </PROMPT> 结尾标识前添加一个 <BREAK> 元素：

程序清单 13-45

```

<PROMPT>
    Your choices are
    <xsl:for-each select="ITEM">
        <xsl:value-of select="TITLE"/><BREAK MSECs="1000" />
    </xsl:for-each>
</PROMPT>

```

<INPUT> 元素也使用了同样的技术。一些 <OPTION> 元素包含在一个与任意 <ITEM> 元素相匹配的 xsl: for-each 循环中。由于要用从 XML 文档中提取出来的数据生成 <OPTION> 元素，我们一起使用了 xsl: element 命令和 xsl: attribute 命令：

程序清单 13-46

```
<xsl:for-each select="ITEM">
  <xsl:element name="OPTION">
    <xsl:attribute name="NEXT">
      #code:
      <xsl:value-of select="CODE"/>
    </xsl:attribute>
    <xsl:value-of select="TITLE"/>
  </xsl:element>
</xsl:for-each>
```

这个循环生成下列输出内容：

程序清单 13-47

```
<OPTION NEXT="#code:16-048">Instant JavaScript</OPTION>
<OPTION NEXT="#code:16-105">Instant Active Server Pages</OPTION>
<OPTION NEXT="#code:16-041">Instant HTML</OPTION>
```

同样，模板中包含 xsl: apply-templates使XSLT引擎处理所有的子结点。

第三个模板与<ITEM>元素相匹配。这个模板用 xsl: element命令生成一个<STEP>元素，用 xsl: attribute生成NAME属性。它用 xsl-value-of从XML元素中取得数值，而 xsl-value-of将<CODE>数据内容添加到“code:”内容中：

程序清单 13-48

```
<xsl:element name="STEP">
  <xsl:attribute name="NAME">
    code:
    <xsl:value-of select="CODE"/>
  </xsl:attribute>
</xsl:element>
```

后面的<PROMPT>元素是用 xsl: value-of命令得到的数据内容建立的：

程序清单 13-49

```
<PROMPT>
  The title you selected is
  <xsl:value-of select="TITLE"/>
  <BREAK MSECs="500" />
</PROMPT>
```

上述语句生成下面的输出结果：

程序清单 13-50

```
<PROMPT>
  The title you selected is Instant JavaScript
  <BREAK MSECs="500"/>
</PROMPT>
```

可以先用XT将XML文档转换成VOXML来检验VOXML例子，然后再用VOXML SDK中的模拟器来解释VOXML文档。

3. 一个带格式化对象的 XSLT 样式表示例

James Tauber 用 XSL 格式对象生成了可以将 XML 转换到 PDF 的 FOP。FOP 软件包免费下载，站点是 <http://www.jtauber.com/fop/>。在 <http://www.renderx.com> 上有一个很快要发布的 XSLF 商业软件包。

PDF 格式是基于一个页面模型的。可以用 Adobe PDF 浏览器或 PDF 插入组件打印一个页面或在屏幕上显示页面。

XSLF 文档的通常结构（可以用一个 XSLT 转换引擎生成）以一个 `<root>` 根元素开始，`<root>` 有两个子元素，一个 `<fo:layout-master-set>` 元素和一个或多个 `<fo:page-sequence>` 元素（参见图 13-21）。

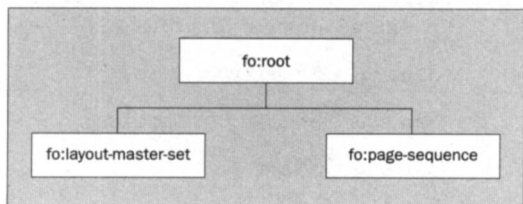


图 13-21

`fo:layout-master-set` 命令包含一个或多个：`<fo: simple-page-master>` 元素，这些元素的作用是页面布局的模板，而 `<fo: page-sequence>` 元素则保存文档的内容。这些元素将一个页面分成 5 个独立区域（参见图 13-22）：

- 主体区（`xsl-body`）。
- 前区，或头部（`xsl-before`）。
- 后区，或脚部（`xsl-after`）。
- 开始区，或左滚动条（`xsl-start`）。
- 结束区，或右滚动条（`xsl-end`）。

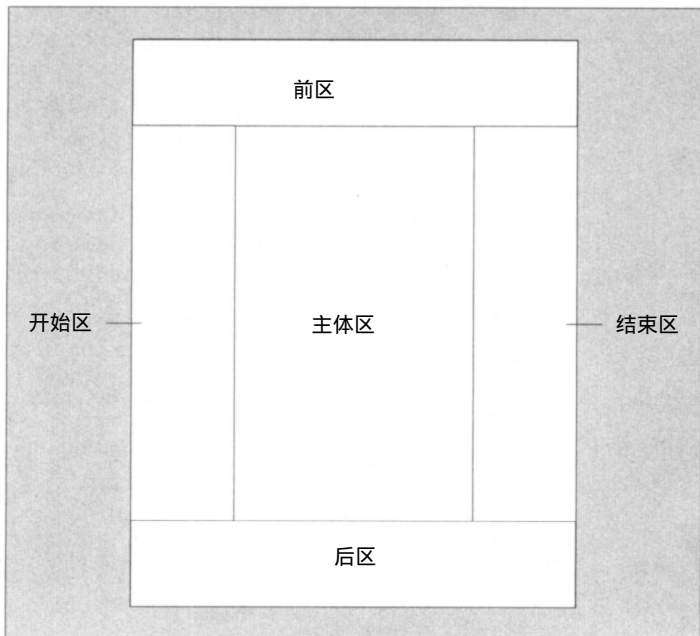


图 13-22

fo:simple-page-master有的属性 (参见表 13-6)

表 13-6

属 性	描 述
id	一个唯一的元素标识符
margin-top	同CSS的margin-top一样
margin-bottom	同CSS的margin-bottom一样
margin-left	同CSS的margin-left一样
margin-right	同CSS的margin-right一样
margin	对块区域或内嵌区域的 margin-top, margin-right, margin-bottom, margin-left 属性进行设定的速记属性
page-master-name	页面管理器的一个标识性名字。这个名字连续被 fo: sequence-specifier-single, fo: sequence-specifier-repeating, fo: sequence-specifier-alternating使用来生成一个页面实例
page-height	决定页面的高度。例如，一个格式为 8 ¹ / ₂ × 11 的页面可以设定为 11 英寸
page-width	决定页面的宽度。例如，一个格式为 8 ¹ / ₂ × 11 的页面可以设定为 8 ¹ / ₂ 英寸
reference-orientation	内容的显示方向可以和页面一样，也可以旋转。例如，给定一个 90 的值，可以使内容在包含区域里从方向 refrence-orientation 开始，逆时针方向旋转 90°。
size	这个属性指定了一个页面盒子的尺寸和方向。例如，如果值为 “ landscape ”，则页面盒子和目标区域的大小一样，并且较长的边为水平方向
writing-mode	表明书写的方向。例如，可以设定为远东书写模式，即从右到左

这是将目录转换成 XSL 格式对象的一个样式表的例子：

程序清单 13-51

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <fo:root xmlns:fo=http://www.w3.org/XSL/Format/1.0>

    <fo:layout-master-set>
      <fo:simple-page-master page-master-name="pagemaster-1">
        <fo:region-body
          column-count="1"
          reference-orientation="0"
          margin="1in" />
      </fo:simple-page-master>
    </fo:layout-master set>

    <fo:page-sequence>
      <fo:sequence-specification>
        <fo:sequence-specifier-single
          page-master-name="pagemaster-1"/>
      </fo:sequence-specification>

      <fo:static-content flow-name="xsl-before">
        <fo:block text-align="centered"
          font=36pt Times"
          font-weight="bold">
```

```

        space-after.optimum="6pt"
        color="red">Book Catalog</fo:block>
</fo:static-content>

<fo:static-content flow-name="xsl-after">
    <fo:block>
        <fo:block text-align="centered" font="10pt Times">
            <fo:page-number/>
        </fo:block>
    </fo:block>
</fo:static-content>

<fo:flow flow-name="xsl-body">
    <fo:block>
        <fo:display-sequence
            text-align="justified"
            font="10pt Times">
            <xsl:apply-templates/>
        </fo:display-sequence>
    </fo:block>
</fo:flow>
</fo:page-sequence>
</fo:root>
</xsl:template>

<xsl:template match="ITEM">
    <fo:block space-before.optimum="9pt">
        <xsl:apply-templates select="TITLE" />
        <fo:block>
            <fo:inline-sequence>Category: </fo:inline-sequence>
            <fo:inline-sequence>
                <xsl:value-of select="CATEGORY"/>
            </fo:inline-sequence>
            <fo:inline-sequence> (</fo:inline-sequence>
            <fo:inline-sequence>
                <xsl:value-of select="CODE"/>
            </fo:inline-sequence>
            <fo:inline-sequence> )</fo:inline-sequence>
        </fo:block>
        <fo:inline-sequence>Release date: </fo:inline-sequence>
        <fo:inline-sequence>
            <xsl:value-of select="RELEASE_DATE"/>
        </fo:inline-sequence>
        <fo:inline-sequence>- Price: </fo:inline-sequence>
        <fo:inline-sequence>
            <xsl:value-of select="PRICE"/>
        </fo:inline-sequence>
    </fo:block>
</fo:block>
</xsl:template>

<xsl:template match="TITLE">
    <fo:block
        font="14pt Arial"
        font-weight="bold">
        <fo:inline-sequence>
            <xsl:value-of select="." />

```

```
</fo:inline-sequence>
</fo:block>
</xsl:template>
```

与根结点匹配的模板生成了一个页面控制，其主题区域包含一个单列。

reference-orientation的方向与页面（portrait）的一样，主体区域的周围边框都是 1 英寸：

程序清单 13-52

```
<fo:layout-master-set>
  <fo:simple-page-master page-master-name="pagemaster-1">
    <fo:region-body
      column-count="1"
      reference-orientation="0"
      margin="1in" />
    </fo:simple-page-master>
  </fo:layout-master set>
```

生成了一个页面控制布局后，下一步是生成页面序列。元素 `<fo:page-sequence>` 包含一个 `<fo:sequence-specification>` 元素，零个或多个 `<fo:static-content>` 元素以及一个 `<fo:flow>` 元素（参见图 13-23）。

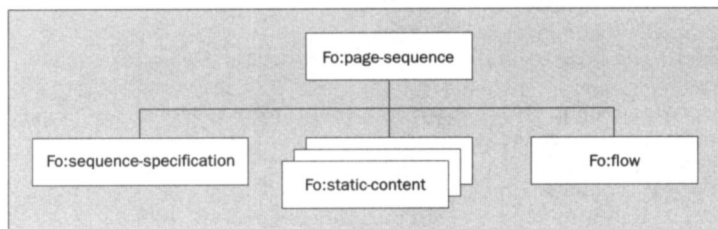


图 13-23

基本上元素 `<fo:page-sequence>` 告诉 XSL 格式生成器页面将如何显示。可以想象这个元素将缺省配置条件应用于后面所有被解释的页面。

`<fo:sequence-specification>` 元素包括一个或多个 `<fo:sequence-specifier-single>`、`<fo:sequence-specifier-repeating>` 和 `<fo:sequence-specifier-alternating>` 元素。

在例子中我们用 `<fo:sequence-specifier-single>` 元素告诉 XSL 格式生成器我们准备用 “pagemaster-1” 页面控制，这个页面控制是在元素 `<fo:simple-page-master>` 中定义的。这就为要解释的页面提供了一个模板。在这里页面都将会会有一个 1 英寸的边框，这同 `page-master` 格式对象中提到的一样。缺省状态下，所有的页面都用同一种格式：

程序清单 13-53

```
<fo:sequence-specification>
  <fo:sequence-specifier-single
    page-master-name="pagemaster-1"/>
</fo:sequence-specification>
```

`<fo:sequence-specifier-alternative>` 元素允许我们对文件设定不同的页面布局，如图书的页面

就具有不同的布局。例如，可以将一个 `page-master` 用于偶数页，另一个用于奇数页。`<fo:sequence-specifier-repeating>` 元素允许我们为封页设定一个页面布局，为后面的页面设定一个循环系列。

在我们的页面中包含了两个静态内容格式对象：题目和页编号。这个静态内容将含在所有的页面中，而与XML文档内容没有任何联系。这有点象在程序中定义常数。

首先，我们为题目建立一个静态内容。题目在文档体之前显示，设定 `flow-name` 属性值为“`xsl-before`”来实现这个操作。题目本身用 `<fo: block>` 元素生成，这个元素居中，黑体，红色，用36pt Times字体显示，并在其下面插入尺寸为6pt的空间。XSL block同CSS block对象十分相似，因此它们有许多共同的属性：

程序清单 13-54

```
<fo:static-content flow-name="xsl-before">
  <fo:block text-align="centered"
    font=36pt Times"
    font-weight="bold"
    space-after.optimum="6pt"
    color="red">Book Catalog</fo:block>
</fo:static-content>
```

在示例中，我们定义了一个在主体区域后显示的页编号，通过设定 `flow-name` 属性为“`xsl-after`”来完成。用另一个块格式对象包含页编号，页编号居中显示，并用 10pt Times字体：

程序清单 13-55

```
<fo:static-content flow-name="xsl-after">
  <fo:block>
    <fo:block text-align="centered" font="10pt Times">
      <fo:page-number/>
    </fo:block>
  </fo:block>
</fo:static-content>
```

最后，我们定义 `<fo: flow>` 元素。这个格式对象中含有 XML 文档内容。我们将 `flow-name` 属性设定为“`xsl-body`”：

程序清单 13-56

```
<fo:flow flow-name="xsl-body">
```

这个元素包括了所有与XML文档相关联的格式对象以及文档主体。

这样，我们在一个页面控制中定义每个页面的基本整体布局。页面控制在 `<fo: sequence-specification>` 元素中作为一个模板使用。然后，每个与元素 `<fo: static-content>` 相连的页面底部都加一个页编号。最后，我们定义文档主体，所有与XML文档相关的流对象都在文档主体上分布（参见图13-24）。

主体中的所有元素都从 `<fo:display-sequence>` 元素中继承了它们的属性。它指定了所有包含元素要继承的一套属性。因为在 `xsl: apply-template` 命令中包含有 `<xsl: apply-templates>` 元素，并

且这个命令是在根模板中生成的，所以随后的模板都要替代这个命令并从 `display-sequence` 元素中继承了所有的缺省属性。

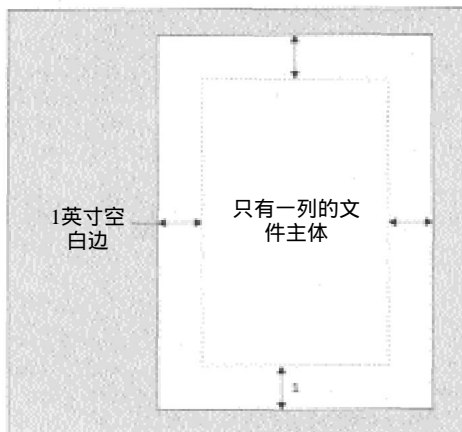


图 13-24

程序清单 13-57

```
<fo:block>
  <fo:display-sequence
    text-align="justified"
    font="lpica Times">
    <xsl:apply-templates/>
  </fo:display-sequence>
</fo:block>
```

每个 `<ITEM>` 元素同一个模板相匹配，这个模板为这个元素的每个实例生成一个块。在块显示前有一部分空间（9 pt）。我们加入 “optimum” 关键字来告诉 XSL 格式化器这块空间的最佳值是 9pt。我们应该已经提到过最大或最小值：

程序清单 13-58

```
<fo:block space-before.optimum="9pt">
```

这个主体块经常包含三个其他块：

- 题目。
- 书的目录和编码。
- 书的发行日期和价格。

通过激活与 `<TITLE>` 元素相关联的模板可以生成标题。这个模板包含一个 `<fo: block>` 元素，这个元素会覆盖掉一些 `display-sequence` 属性：

程序清单 13-59

```
<xsl:template match="ITEM">
  <fo:block space-before.optimum="9pt">
```

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

...

<xsl:template match="TITLE">
  <fo:block
    font=14pt Arial"
    font-weight="bold">
    <fo:inline-sequence>
      <xsl:value-of select = "." />
    </fo:inline-sequence>
  </fo:block>
</xsl:template>
```

其他两个块格式对象在包含 <fo: inline-sequence> 格式对象的题目后显示内容，内容是通过 <xsl:value-of> 元素从XML文档中提取出来的。一个内部序列格式对象同一个 CSS 内部对象非常相似。

一旦你已经生成了转换文档，可以用 FOP 这样的应用程序来生成文档的 PDF 版本。

4. 关于XSL的结论

XSLT 很明显成功了，IE 5 加上 XSLT 实现 HTML+CSS 的转换功能，在写本书时 Mozilla 对 XSLT 的支持正处于开发阶段。所以如果浏览器支持一种样式语言，而 XSLT 是一个功能强大的语言，我们就有望取得成功。

我不能说 XSLF（XSL 的格式部分）也同样如此。它的未来取决于 XML 内置的编辑工具的支持。但即使是这样，如果浏览器不支持这种语言，它就不能像 XSLT 一样普及。但是，XSLF 可能同 DSSSL（我们下面将会提到）的功能一样 作为一种打印样式语言。

XSLT 在服务器端的转换功能很有潜力，因为一个 XML 文档在桌面浏览器上可以转换成 HTML+CSS 格式，在语音浏览器上可以转换成 VOXML 格式，在移动电话和 PDA 上可以转换成 WML 格式。

讨论过 CSS 和 XSL 后，我们现在简单讨论一下另外两种语言 DSSSL 和 Omnimark，你也有可能会学会更多关于它们的知识。我们简单地讨论一下这两种语言，然后给出一个用于图书目录文件的代码示例，这样你对其语法就会有一个概念 我们没有篇幅来讨论它们的细节问题，但你应该对是否要花更多的时间来学习它们有个想法。

13.3.8 DSSSL

DSSSL 自从 1996 年起就是一个 ISO 标准。这个语言主要由两部分组成：

- 一个样式语言。
- 一个转换语言。

样式语言的伸缩性很强，可以像 CSS 一样简单，也可以是一个完整的表达语言。表达语言是一个允许你在 XML 文档层次模型上进行繁琐处理的完整编程语言。它的可视化模型基于不依赖任何特定条件的流对象。例如，DSSSL 段落流对象可以同同一个 HTML <p> 元素和一个 RTF/par 对象相映射。它是一个可以处理 SGML 和 XML 文档的规则语言。XSL 中包含了许多 DSSSL 特性。

DSSSL 通常很难学，很复杂。这是因为 DSSSL 包含功能强大的表达语言。表达语言是 LISP

的一个子集 Scheme。熟悉模块程序语言如 C、C++、Basic、Pascal 等的程序员可能学起来会有困难。DSSSL 中可以不包含表达语言，这样，它的难度就和 CSS 一样了。

DSSSL 表达语言很适合符号管理。可以用 DSSSL 样式格式化对象和基于方案的表达语言来处理很复杂的样式表。实际上，DSSSL 常用于印刷刊物而不是在线解释。但是，也有一些服务器端的 DSSSL 被用于将 SGML 或 XML 文档转换成 HTML。

一个 DSSSL 脚本在处理 XML 文档时先将其转换成一个“GROVE”。GROVE 相当于 ISO 中的 W3C DOM。但是，它更多的是一个抽象模型，虽然 DOM 是一个 API。我们说 GROVE 可能更接近于 W3C Information Set 而不是 DOM。GROVE 是一个数据模型，而 DOM 是这个数据模型的一个接口。所以，当处理 GROVE 或数据结构时，DSSSL 规则同 XML 文档元素模式匹配。DSSSL 规则模型比 XSL 中的更复杂，它包含几种规则类型 而 XSL 有一种类型就足够了。

可以用 DSSSL 转换 XML 文档。它的解释模型同 XSL 一样功能强大，并且 XML 文档结构和解释之间互相独立。DSSSL 也可以加入新元素和字符串，并对最终输出结果进行重新排序。

写本书时 XSLF 还处于开发阶段。但同 XSLF 不一样的是 DSSSL 在每天的文档处理中已经开始使用了。第一个应用 DSSSL 的是 James Clark，他写了一个叫 Jade 的程序。现在这个程序由一个国际集团维护，Jade 改为 OpenJade，是开放式的。OpenJade 可以免费下载，能将 XML 或 SGML 文件转换为 HTML、SGML、XML、PDF、MIF、TeX、RTF 或 Braille，其站点为 <http://www.netfolder.com/DSSSL>。

1. 一个样式表示例

在下面的例子中，我们可以生成同 XSL 应用一样的结果，而不必像 CSS 应用那样在样式处理以前要转换文件。我们不会详细讨论脚本，但会很快地看一下 DSSSL 的一些基本概念：

程序清单 13-60

```
<!doctype style-sheet
PUBLIC "-//Netfolder//DTD DSSSL library//EN" >
(root
  (make scroll
    (process-children)
  )
)

(element ITEM
  (make paragraph
    start-indent: 40pt
    font-family-name: "Times New Roman"
    font-size: 12pt
    font-weight: 'medium
    space-after: 15pt
    (process-matching-children "TITLE")
    (make paragraph
      font-weight: 'medium
      (literal "Category: ")
      (process-matching-children "CATEGORY")
      (literal " (")
      (process-matching-children "CODE")
      (literal ")")
    )
  )
)
```

```

        (make paragraph
          (literal "Release date: ")
          (process-matching-children "RELEASE_DATE")
          (literal " - Price: ")
          (process-matching-children "PRICE")
        )
      )
    )

    (element TITLE
      (make paragraph
        font-family-name: "Arial"
        font-size: 14pt
        font-weight: 'bold
      )
    )

    (element CATEGORY
      (make sequence)
    )

    (element CODE
      (make sequence)
    )

    (element RELEASE_DATE
      (make sequence)
    )

    (element PRICE
      (make sequence)
    )
  )

```

第一个规则生成一个 scroll 格式对象。它代表着浏览器的滚动浏览。样式表里这一部分中的 root 元素指向 XML 文档而不是根元素。如果我们要将文件翻译到打印介质上，那么应该采用 simple-page-sequence 格式对象。

同 <ITEM> 元素相匹配的规则完成了大部分工作。首先，生成一个 paragraph 对象。paragraph 是一个包含 sequence 对象的矩形区域。paragraph 同 CSS 中的 box 对象相似，sequence 对象同 inline 对象相似。

段落属性叫做 characteristics，可能已经注意到它们同 CSS 属性名称相似。DSSSL 命令：

程序清单 13-61

```
(process-matching-children "TITLE")
```

让样式引擎取消与 <TITLE> 元素相关联的规则。后者生成了前者中的一个 paragraph 对象。一旦已经生成了这个段落，就会有另外两个段落添加到这个段落上。

同我们在 XSLT 到 HTML 的转换中做的一样，在源文档中没有新内容被添加进去。这个操作是通过同字符串等价的 literal 格式对象完成的。内容在生成同 <CATEGORY> 元素相关的格式对象以前加入：

程序清单 13-62

```
(literal "Category: ")  
(process-matching-children "CATEGORY")
```

如果一系列元素共享一个规则体，用下面的语句更方便：

程序清单 13-63

```
(element CATEGORY CODE RELEASE_DATE PRICE  
  (make sequence)  
)
```

这类命令不是 DSSSL ISO 标准的一部分呢，但在 OpenJade 软件包中用来简化应用。

DSSSL 说明不是静态的，这个 ISO 样式语言现在正进入第二个编辑阶段。它现在叫 DSSSL-2 项目，是对现有说明的改进和扩展。很明显，维持当前的说明是为了保证现存样式表的稳定性。

2. DSSSL 讨论小结

你可能注意到了 DSSSL 并不一定比 XSLF 难学，实际上在某些方面它甚至更简单。当然，它的表示语言“模式”对于习惯用程序语言如 JavaScript 的人来说可能会不太容易理解。但是，它的格式对象结构体系十分简单易读，甚至比 XSL 还简单。但在访问 GROVE 结点方面比 XPath 费劲得多。

总的来说，DSSSL 是一个功能强大的语言，可能会适合于打印资料的文档处理，而不是在线解释，目前打印领域已经开始使用 DSSSL 了。

13.3.9 Omnimark

最近，Omnimark 语言（由 Omnimark Technologies 开发）正以免费软件包的形式提供。它可以在 <http://www.omnimark.com> 上下载。因为可以免费下载并把 XML 文档转换成 HTML，我们粗略地看看这种简单语言。

同前面讨论过的语言一样，Omnimark 是一个规则语言。规则通过 Omnimark 的 element 指令与 XML 元素相匹配。同 DSSSL、CSS 或 XSL 不一样，Omnimark 不使用一套直观的格式对象，而是用一个叫 output() 的程序。output() 程序用一个字符串作为它的参数。这个字符串可以包括任何表达式，因此可以用来对基于文本的格式语言如 VOXML、HTML、TeX 等输出文档。

同 DSSSL 或 XSL 一样，Omnimark 有一个用于包括其他规则生成内容的命令。%c 表达式代表 content，用规则的输出替换，这些规则同当前被取消规则的子元素相匹配。例如，<ITEM> 元素是 <BOOKLIST> 元素的一个子元素，如果同 <BOOKLIST> 元素相匹配的规则中包含一个带 %c 的参数的输出命令，那么就由和 <ITEM> 元素匹配的规则生成的内容替换结果文件中的表达式。

1. 一个 Omnimark 脚本的范例

下面的脚本同其他样式表语言有点不一样，虽然它也是基于规则的。XML 文档的元素同 Omnimark 规则相匹配。一个最简单的规则中包括一个或多个 output 命令。生成的结果采用了字符串的形式：

程序清单 13-64

```

down-translate with xml

element BOOKLIST
  output "<html>%n<head>%n" ||
    "<title>Transform with Omnimark<title>%n" ||
    "</head>%n<body>%c</body>%n</html>"

element ITEM
  output "<DIV style='margin-left: 40pt;%n' ||
    'font-family: Times New Roman;%n' ||
    'font-size: 12pt;%n' ||
    'font-weight: 500;%n' ||
    'margin-bottom: 15pt;%n' ||
    'text-align: left;%n' ||
    'line-height: 12pt;%n' ||
    'text-indent: 0pt;' >%n"

element TITLE
  output "<DIV style='margin-left: 40pt;%n' ||
    'font-family: Arial;%n' ||
    'font-weight: 700;%n' ||
    'font-size: 14pt;' >%n > ||
    "<SPAN>%c</SPAN>%n</DIV>%n"

element DESCRIPTION
  output "<DIV>%n%c</DIV>%n"

element ( CATEGORY | CODE | RELEASE_DATE | PRICE )
  output "<SPAN>%c</SPAN>"

```

你可能注意到输出表达式中有一些用 % 分开的字符。在输出字符串中有一些命令。例如，规则中的 %c 命令：

程序清单 13-65

```

element BOOKLIST
  output "<html>%n<head>%n" ||
    "<title>Transform with Omnimark<title>%n" ||
    "</head>%n<body>%c</body>%n</html>"

```

同 XSLT 的 apply-templates 命令是等价的。它告诉 Omnimark 引擎同 <BOOKLIST> 元素的子元素相匹配的其他规则将会插入到生成字符串的相应位置上。

另一个命令 %n 表示插入行的一个结尾（很像 CR/LF 一个回车/换行符），这样输出字符串的可读性更好。

2. Omnimark 讨论小结

如果不谈转换功能，Omnimark 是一种易用的语言，但在转换功能方面 XSLT 更好一些。不过对于通常的 XML 到 HTML 转换，以及对于那些不习惯声明语言而喜欢程序语言的人来说，Omnimark 是一个值得推荐的工具。

13.4 小结

本章中我们详细地讨论了两种样式语言——CSS和XSL它们可以向用户提供 XML，然后简单讨论了可以和XML一起使用的DSSSL和Omnimark。

我们看到CSS1和CSS2建议标准十分明确并已经在IE 5中有相应的部分，将来Mozilla也要加入这些内容。另外，CSS3正在开发中，它将对现有的CSS模型进行改进。它们十分适合在支持XML和CSS的浏览器中浏览XML。此外，这种语言简单易学，使用方便，将会在当前的HTML用户中普及。

然后我们讨论XSL。在写本书时XSLT和XPath刚刚成为W3C的建议标准，所以要等到主要浏览器生产厂家都采用新标准还要一段时间（虽然他们最终一定会采用这个标准的）。同时，也有其他工具支持XSLT。此外，格式对象也正在研究中，它将增强对打印的支持。XSLT是一个强大的通用转换工具。它使我们生成一系列用于显示的格式，而不必要求一个支持XML的用户代理。

我们也讨论了功能强大的DSSSL语言，它可以处理SGML或XML文件。DSSSL基于一个很快就要改版的国际标准，很快我们将见到第二代的DSSSL，它将会包括一些根据我们在过去几年中积累的经验所添加的新特性（我们希望）。最后，我们简单讨论了Omnimark语言。虽然它是一个专用语言，但每天在许多场合下都使用它并被证明这是一个有用的工具。同大多数专用语言相反的是，Omnimark有一个免费下载的版本，习惯于程序语言的人们使用它会得心应手。

大部分工具都可以免费下载，因此你可以亲自检验一下它们的局限性和应用潜力：

- CSS: IE 5 (<http://www.microsoft.ie>)和Mozilla (<http://www.mizilla.org>)。
- XSLT: IE 5、James Clark的XT (<http://www.jclark.com/xml/xt.html>)和 SAXON (<http://users.iclway.co.uk/mhkay/saxon>)。
- XSLF: James Tauber的FOP (<http://www.jtauber.com/fop/>)。
- DSSSL: OpenJade (<http://www.netfolder.com/DSSSL>)。
- Omnimark: Omnimark Corp (<http://www.omnimark.com>)。