项目 4 个人网上图书馆网页设计

——用 XSL 实现个性化显示

一、知识目标

项目目标：通过个人网上图书馆网页设计实例的制作，展示 XSL 在 XML 网页设计中的作用，并较为详细地讲解 XSL 文件的结构、XSL 的常用标记的使用方法，以及使用 XSL 样式表对 XML 文档进行变换的方法。同时，对 XSL 模板的设计和使用也进行介绍。

教学目标：

- 理解 XSL 文件的基本结构。
- 掌握 XSL 文件中使用 CSS 样式表的方法。
- 掌握 XSL 中常用元素的含义和使用方法。
- 掌握 XSL 文件中 HTML 模板的设计方法。
- 理解 XSL 模板匹配和调用方法。

二、能力目标

- 培养设计 XSL 模板的能力。
- 培养阅读、书写 XSL 文件的能力。
- 培养利用 XSL 修饰、显示 XML 文档的能力。
- 培养利用 XSL 和 XML 进行网页设计的综合应用能力。

三、教学任务

- 创建利用 XSL 修饰的个人网上图书馆 XML 首页。
- 制作借阅书籍分类展示的 XML 网页。
- 制作借阅书籍介绍 XML 网页。

预备知识

一、XSL 概述

（一）XSL 简介

XSL（eXtensible Stylesheet Language，可扩展样式表语言）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 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 是包含了一套元素集的 XML 语法规范，而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 等其他格式。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，如数据的显示顺序的改变，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，或者经过折叠只显示外面的一层，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 Semantics and 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。

微软发布了两种 XSL 处理器：一个是可以从 XML 文档和 XSL 样式表产生 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 或其他 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，可将文档转换为相应的格式，供浏览器显示。

可扩展样式表语言（XSL）包括转换语言和格式化语言。每种语言都是一种 XML 应用。转换语言提供定义规则的元素如何将 XML 文档转换为另一个 XML 文档。转换的 XML 文档可能使用原文档的标记和 DTD，或者使用一组完全不同的标记。特别是可能会使用 XSL 第二部分（格式化对象）定义的标记。本章只讨论 XSL 转换语言中的部分内容。

（二）XSL 的编程思想

对于作为表现对象的 XML 文档，XSL 把它看成一棵由许多节点组成的树，这棵树称为源树。而组成 XML 文档的根元素和子元素都是源树的节点。当设计 XSL 文件来表现 XML 文档时，就是从源树的相应节点中取出需要的数据，而这些数据又形成一棵树，称为结果树。一旦

构造出结果树，则结果树与源树之间就是相互独立的，对结果树中数据的操作不会对源树中的相应数据产生任何影响。通过这样的方式，XSL 已实现了 XML 文档的内容与表现形式的彻底分离。

组成结果树的数据也不是随意放置的，而是存放在 XSL 的模板中。你可以用多种方式来构造这个模板，但通常是使用 HTML 来制作放置结果树数据的模板，并把这个模板称为 HTML 模板。

如果要让结果树中的数据更好地表现在页面中，那么就可以通过设置 HTML 标记的 STYLE 属性或是使用<STYLE></STYLE>标记来定义需要的样式。而页面最终的显示处理工作将交给浏览器去完成。

因此，当使用 XSL 来编写表现 XML 文档的样式文件时，就必须具备这种思想，即设计者编写 XSL 文件实际上是编写某种格式的模板，这个模板将用来存放从 XML 文档提取出的数据，完成由源树构造结果树的工作。设计者通过编写 XSL 代码来制作模板，在逻辑上实现结果树的构造，XSL 处理器将按照 XSL 代码中的各条指令，具体负责构造结果树的工作，而带有 XSL 处理器的浏览器则负责将得到的结果树转化为相应的文档（如 HTML 文档），并把转化后的文档在浏览器窗口中显示出来。

（三）结果树

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。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）。

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。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”。这是一个简单、典型的例子。XML 元素的内容被表现信息所包围。

（四）XSL 与 CSS 的比较

CSS 与 XSL 在某种程度上是重复的。XSL 的功能确实比 CSS 更强大，但是 XSL 的功能与其复杂性是分不开的。这一章仅仅涉及了 XSL 最基本的用途。实际上 XSL 更复杂，而且比 CSS 更难学习和使用，同时也带来了一个问题：“什么时候应该使用 CSS，什么时候应该使用 XSL？”

CSS 比 XSL 得到更广泛的支持。部分 CSS Level 1 被 Netscape 4 和 Internet Exploer 4 支持

作为 HTML 元素（尽管存在一些令人头疼的区别）；此外，Internet Exploer 5.0 和 Mozilla 5.0 能很好地支持可以同时用于 XML 和 HTML 的大部分 CSS Level 1 的内容和一些 CSS Level 2 的内容。因此，选择 CSS 会与更广泛的浏览器相互兼容。

另外，CSS 更成熟一些，CSS Level 1（包含目前为止大部分 CSS 内容）和 CSS Level 2 是 W3C 的推荐规范。早期的 XSL 采纳者曾经接受过考验，而且将在形式统一的标准之前接受再一次的考验。选择 CSS 意味着无须为了追随软件和标准的发展不停地重写自己的样式单。但是，XSL 将最终形成一个可用的标准。

因为 XSL 是一种新事物，不同的软件实现方式不同，实现的是草案标准的不同的子集。如果当前浏览器中不完善的 CSS 操作已经让人头疼的话，那么众多的 XSL 变种就会使人发疯。

但是，XSL 的功能很明显比 CSS 强大。CSS 仅允许格式化元素内容，不允许改变或重新安排这些内容，必须根据元素的内容或属性为元素选择不同的格式化方式或者增添诸如署名之类简单、额外的文本。XSL 非常适用于 XML 文档，它仅包含最少的数据，并且数据周围没有 HTML 装饰的情况。

使用 XSL 能够从页面上分离出关键数据，如刊头、向导栏和署名等。使用 CSS 不得不在数据文档中包含全部这些项目。XML+XSL 允许数据文档与 Web 页面文档分离单独存在，从而使得 XML+XSL 文档更容易维护和处理。

XSL 终将成为现实世界和大量数据应用的最佳选择，CSS 更适合于简单的页面，如祖母用于向她们孙子寄送图片的页面。但对于这些用途，HTML 已经足够。如果使用 HTML 行不通，XML+CSS 不会有多大的帮助。相较而言，XML+XSL 能够解决更多 HTML 不能解决的困难。对于传统的浏览器来说，仍然需要 CSS，但长远看来使用 XSL 才是发展方向。

XML 的样式表语言 XSL 比 CSS 要复杂得多。XSL 与 CSS 的比较如下：

1. CSS：HTML 的样式表语言

由于 HTML 使用预先确定的标记，因此这些标记的含义都很好理解：<p>元素定义一段，<h1>元素定义一个标题。浏览器知道如何显示这些元素。

使用 CSS 向 HTML 元素增加显示格式是一个简单的过程，很容易告诉浏览器用某种特殊字体或颜色来显示各个元素，浏览器也很容易理解。

2. XSL：XML 的样式表

由于 XML 不使用预先确定的标记（可以根据需要使用任意标记），因此标记的含义并不能被直接理解：<table>可以表示一个 HTML 表格，也可以表示一件家具。由于 XML 的特性，浏览器不知道如何显示一个 XML 文档。

为了显示 XML 文档，必须要有一个机制来描述如何显示文档。这些机制之一是 CSS，但是 XSL（可扩展的样式表语言）是 XML 的首选样式表语言，它要比 HTML 使用的 CSS 复杂得多。

3. XSL：不仅仅是一个样式表

XSL 包含 3 部分：XSLT、XPath 和 XSL-FO。

XSLT：一种用于转换 XML 文档的语言。

XPath：一种用于在 XML 文档中导航的语言。

XSL-FO：一种用于格式化 XML 文档的语言。

如果对此还不能理解，那么可以先将 XSL 理解成：一种将 XML 转换成 HTML 的语言，一种可以过滤和分类 XML 数据的语言，一种可以对一个 XML 文档的任意部分进行寻址的语

言，一种可以基于数据值格式化 XML 数据的语言（如用红色显示负数），一种向不同设备输出 XML 数据的语言（如屏幕、纸或音频设备）。

（五）在何处进行 XML 变换

使用 XSL 样式单可有三种主要方式将 XML 文档变换成其他格式（如 HTML）：

（1）XML 文档和相关的样式单都是用于客户端（Web 浏览器）的，然后客户端程序按照样式单中指定的格式变换文档，并将它呈现给用户。

（2）服务器将 XSL 样式单应用于 XML 文档，以便此文档能够变换成其他格式（通常为 HTML），并把变换后的文档发送到客户端程序（Web 浏览器）。

（3）第三种方式是将原 XML 文档变换成其他格式（常常为 HTML）后，才把此文档放置在服务器上。服务器和客户程序只处理变换后的文档。

这三种方法尽管都使用相同的 XML 文档和 XSL 样式，但每一种都使用不同的软件。将 XML 文档发送到 Internet Explorer 的普通 Web 服务器使用的就是第一种方法。使用 IBM alphaWork 的 XML 功能将文档应用于与 applet 兼容的 Web 服务器就是第二种方法的例证。使用命令行 XT 程序来将 XML 文档变换成 HTML 文档，然后将 HTML 文档放置在 Web 服务器上，采用的就是第三种方法。但是，这些方法都使用相同的 XSL 语言。

（六）支持 XSLT 的浏览器

几乎所有主要的浏览器均支持 XML 和 XSLT。

1. Mozilla Firefox

从 1.0.2 版本开始，Firefox 就已开始支持 XML 和 XSLT（以及 CSS）。

2. Mozilla

Mozilla 含有用于 XML 解析的 Expat，并支持 XML + CSS。Mozilla 同样支持命名空间。Mozilla 可执行 XSLT。

3. Netscape

从版本 8 开始，Netscape 就开始使用 Mozilla 引擎，所以它对 XML/XSLT 的支持与 Mozilla 是相同的。

4. Opera

从版本 9 开始，Opera 已开始支持 XML 和 XSLT（以及 CSS）。版本 8 仅支持 XML + CSS。

5. Internet Explorer

从版本 6 开始，Internet Explorer 已开始支持 XML、命名空间、CSS、XSLT 以及 XPath。版本 5 不兼容官方的 XSL 标准。

模块 1　个人网上图书馆首页制作

一、知识目标

通过本模块的制作和总结，大体上建立起利用 XSL 修饰 XML 文档的基本认识。通过学

习，将掌握下面这些内容：

- 理解 XSL 文件的基本结构；
- 在 XSL 文件中提取 XML 文档中数据的基本方法。
- 在 XSL 文件中设计 HTML 模板的基本方法。
- 在 XSL 文件中添加 CSS 样式代码的基本方法。

二、教学任务

通过设计读者俱乐部欢迎页面 XML 源文档以及表现该 XML 文档的 XSL 文件，制作结果在 IE 中的浏览效果如图 4-1 所示。

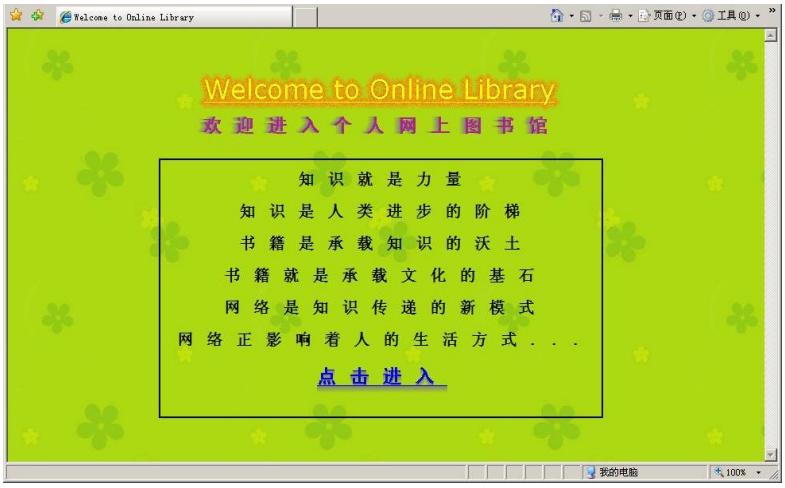

图 4-1　读者俱乐部欢迎网页效果

三、操作流程

（1）准备个人网上图书馆首页网页的内容。在制作 XML 网页之前，需明确网页上的内容，才能进一步规划或设计 XML 文档等。该网页包含的内容如下：

Welcome to Online Library
欢迎进入个人网上图书馆

知识就是力量
知识是人类进步的阶梯
书籍是承载知识的沃土
书籍就是承载文化的基石
网络是知识传递的新模式
网络正影响着人的生活方式...

单击进入

（2）制作个人网上图书馆首页网页的 XML 源文档。打开记事本，输入如下代码，并以“Book4-1.xml”为文件名保存。

```
<?xml version="1.0" encoding="utf-8"?>
<FirstPage>
    <title1>Welcome to Online Library</title1>
    <title2>欢迎进入个人网上图书馆</title2>
    <text>
        <newline>知识就是力量</newline>
        <newline>知识是人类进步的阶梯</newline>
        <newline>书籍是承载知识的沃土</newline>
        <newline>书籍就是承载文化的基石</newline>
        <newline>网络是知识传递的新模式</newline>
        <newline>网络正影响着人的生活方式...</newline>
    </text>
    <bottom>
    单击进入
    </bottom>
</FirstPage>
```

（3）用 IE 浏览“Book4-1.xml”文件，结果如图 4-2 所示。

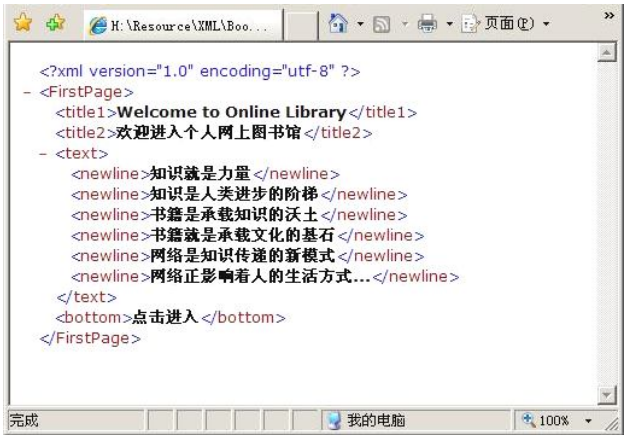

图 4-2　个人网上图书馆首页 XML 文档

（4）制作个人网上图书馆首页的 XSL 样式表文件。打开记事本，输入如下代码，以“Book4-1.xsl”作为文件名保存。

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>
<xsl:value-of select="FirstPage/title1" />
</title>
</head>
```

```
<body background="wz_bg.gif">
<center>
    <p id="title1">
        <xsl:value-of select="FirstPage/title1" />
    </p>
    <p id="title2">
        <xsl:value-of select="FirstPage/title2" />
    </p>
    <p id="text">
    <xsl:for-each select="FirstPage/text/newline" >
    <p>
        <xsl:value-of select="." />
    </p>
    </xsl:for-each>
    </p>
    <p id="bottom"><a href="BOOK4-2.xml">
    <xsl:value-of select="FirstPage/bottom" />
    </a>
    </p>
</center>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

（5）用 IE 浏览 Book4-1.xsl 文件，结果如图 4-3 所示。

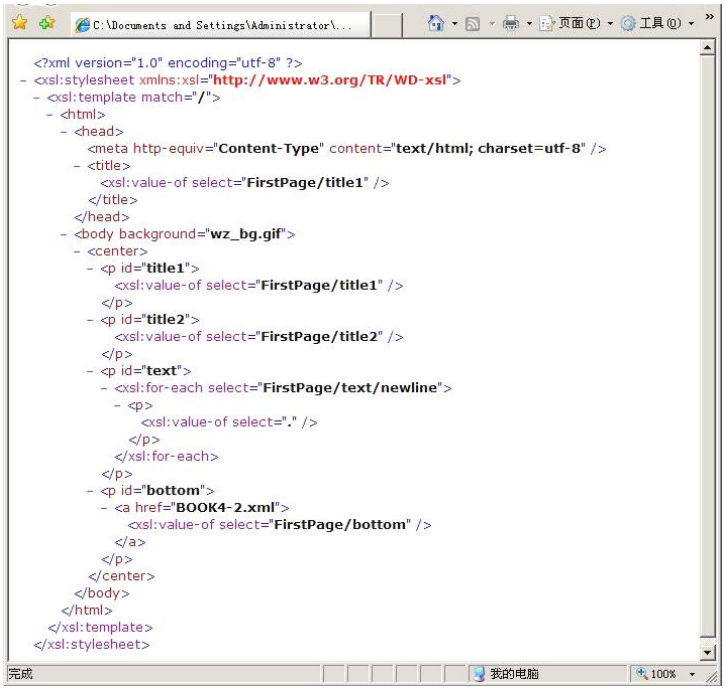

图 4-3　个人网上图书馆首页 XSL 文档

（6）添加关联。在“Book4-1.xml”文件的声明区中加入下面的处理指令并保存文档。

```
<?xml-stylesheet type="text/xsl" href="book4-1.xsl" ?>
```

这是关联样式表所设计的处理指令。该指令的作用是，在浏览这个 XML 文档时，通知浏览器去寻找一个名为 book4-1.xsl 的 xsl 文件，并用此文件来表现 XML 文档。添加关联后，浏览“Book4-1.xml”的效果如图 4-4 所示。

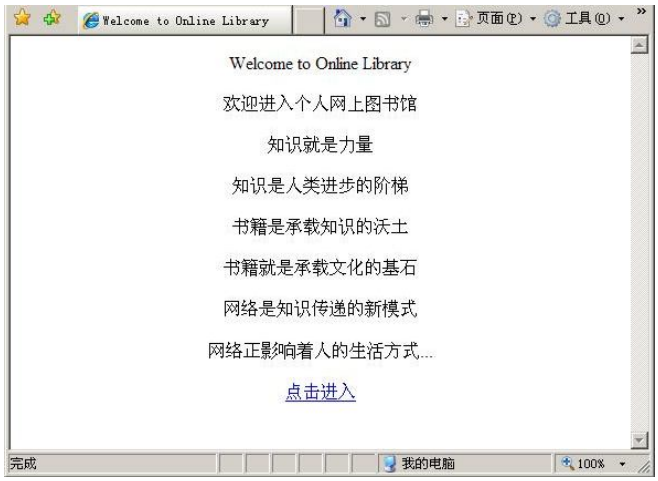

图 4-4　添加关联后浏览“Book4-1.xml”的效果

（7）在 XSL 样式表中添加 CSS 修饰。在 XSL 文件中使用 CSS 有两种方式：一种是可以直接设置文件中的 HTML 标记的 Style 属性；另外一种是在文件中使用<Style>、</Style>标记来单独定义 HTML 标记的样式。在“Book4-1.xsl”中，添加<Style>、</Style>标记，并输入如下 CSS 代码。

```
<style type="text/css">
#text {display:block;
        position:absolute;
        top:30%;
        left:20%;
        width:60%;
        height:60%;
        border:2px solid;}
#title1 {display:block;
        position:absolute;
        top:10%;
        left:10%;
        width:80%;
        height:30%;
        font-family:verdana;
        font-size:25pt;
        text-align:center;
        text-decoration:underline;
        color:yellow;
```

```
        filter:glow(color=#ff8800,strength=8);}
#title2 {display:block;
        position:absolute;
        top:20%;
        left:25%;
        width:50%;
        height:20%;
        font-family:"黑体";
        font-size:18pt;
        font-weight:bolder;
        letter-spacing:12pt;
        text-align:center;
        color:rgb(173,37,139);
        filter:shadow(color=#808080,direction=270);}
#bottom {display:block;
        position:absolute;
        top:78%;
        left:25%;
        width:50%;
        height:20%;
        font-family:"黑体";
        font-size:18pt;
        font-weight:bolder;
        letter-spacing:12pt;
        text-align:center;
        color:rgb(173,37,139);
        filter:shadow(color=#808080,direction=180);}
</style>
```

对下面所示的标记<p>添加 Style 属性。

```
   ...
  <p id="text">
       <xsl:for-each select="FirstPage/text/newline" >
       <p style="
       display:block;
       position:relative;
       top:30%;
       left:auto;
       font-family:'宋体';
        font-size:15pt;
        font-weight:bolder;
        letter-spacing:12pt;
        text-align:center;">
            <xsl:value-of select="." />
       </p>
       </xsl:for-each>
       ...
```

（8）再对“Book4-1.xml”文档用 IE 进行浏览，就得到如图 4-1 所示的最后效果。

四、知识要点分解

（一）XSL 变换的基本步骤

在 XSL 变换中，XSL 处理程序读取 XML 文档和 XSL 样式表。基于处理程序在 XSL 样式单中找到的指令，输出新的 XML 文档。

按照 XSL 变换语言的要求，为需要显示数据的 XML 文件编写 XSL 样式表。

把需要用 XSL 样式表显示数据的 XML 文件关联到该 XSL 样式表。

使用应用程序将 XSL 样式表变换为一个 HTML 文件。在网页设计中，该步骤由浏览器在浏览的时候完成。

XSL 变换的基本步骤如图 4-5 所示。

图 4-5　XSL 变换的基本步骤

（二）XSL 文件的结构

一般来说，XSL 文件具有如下的结构：

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <xsl:template match="/">
    ...
    </xsl:template>
    <xsl:template match="标记匹配模式">
    ...
    </xsl:template>
    <xsl:template match="标记匹配模式">
    ...
    </xsl:template>
</xsl:stylesheet>
```

其中，第一条语句<?xml version="1.0" encoding="utf-8"?>是 XML 声明。XSL 文件也是一个 XML 格式的文件，所以，也要包含 XML 文件的声明。第二条语句由 XSL 的 xsl:stylesheet 元素定义了一个 XSL 样式表，并给出 XSL 命名空间声明。因为在网页设计中需要让浏览器的 XSL 处理器来实现 XSL 变换，所以，根标记 xsl:stylesheet 必须有命名空间，而且命名空间必须是"http://www.w3.org/TR/WD-xsl"。

接着是模板的定义：

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

包围起来的部分是模板的定义，这是编写 XSL 文件的最主要的工作，在一个 XSL 文件中可以使用多个模板。需要指出的是，由于 XSL 文件是一个 XML 格式的文件，因此整个文件必须格式良好并符合 XML 规则，包括实现模板的 HTML 代码的书写格式都应该是格式良好的。

1. XSL 样式表中的模板

（1）模板标记。

XSL 样式表的基本结构是由若干个称为“模板”的标记组成，简称模板。模板都是根标记的子标记，模板标记的名称都是 template，例如：

```
<xsl:template match="标记匹配模式">
    模板内容...
</xsl:template>
```

一个模板的“模板内容”是由 HTML 标记和嵌入其中的 XSL 标记组成，XSL 变换处理器在做变换时，会将 XML 标记直接转化为 HTML 标记，并对 XSL 标记实施变换操作，将变换得到的内容嵌入到 HTML 文件中。

模板标记必须有 match 属性，该属性的值是一个特殊的字符串，称作“标记匹配模式”。根据标记匹配模式，可以找到该模板匹配的 XML 标记。如：

```
<xsl:template match="Book">
    模板内容...
</xsl:template>
```

这个模板匹配关联 XML 文件中的 Book 元素，也就是 XML 文件中的<Book>标记。

（2）根模板。

一个 XSL 样式表必须要有一个根模板。所谓根模板就是和 XML 文件中的根标记匹配的模板，但是，如果使用浏览器来处理 XSL 变换，根模板的“标记匹配模式”必须是“/”，如：

```
<xsl:template match="/">
    模板内容...
</xsl:template>
```

XSL 处理器必须找到根模板，然后开始 XSL 变换。XSL 处理器总是从根模板开始实施 XSL 变换。根模板的内容中可以包含其他模板的调用标记，如：

```
<xsl:template match="/">
    模板内容...
    <xsl:apply-templates select="Book" />
    模板内容...
</xsl:template>
<xsl:template match="Book">
    模板内容...
</xsl:template>
```

在根模板的模板内容中，通过<xsl:apply-templates select="Book" />调用了下面的一个模板。XSL 样式表可以定义很多模板，但如果没有根模板，这个样式表就没有任何作用，其他模板如果在根模板中没有直接或间接的调用，也实现不出任何效果。XSL 样式表组成结构如图 4-6 所示。

2. XSL 模板的调用

XSL 变换的时候，XSL 处理器首先找到根模板，并在对根模板的变换过程当中，使用 XSL 模板调用标记来调用其他模板。

图 4-6　XSL 样式表组成结构

（1）模板调用。

模板调用可以使用两种类型的标记。

第一种，带 select 属性的标记：

```
<xsl:apply-templates select="标记匹配模式" />
```

称作有条件的 XSL 模板调用标记。可以在一个根模板中调用其他的非根模板，例如：

```
<xsl:template match="/">
    模板内容...
    <xsl:apply-templates select="Books/author" />
    模板内容...
</xsl:template>
<xsl:template match="Books/author">
    模板内容...
</xsl:template>
```

第二种是不带 select 属性的标记：

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

由于该模板调用不带 select 属性，没有“标记匹配模式”，所以需要作为“xsl:for-each”标记的子标记来使用。

```
<xsl:for-each select="标记匹配模式" >
    <xsl:apply-templates />
< xsl:for-each >
```

（2）模板调用的执行过程。

对于带 select 属性的模板调用标记，XSL 处理器首先根据<xsl:apply-templates select="标记匹配模式" />中的“标记匹配模式”，到 XML 文件中寻找所有和“标记匹配模式”匹配的 XML 标记，然后，逐个地为这些标记到 XSL 样式表中寻找匹配的模板，如果找到，就对该模板的内容实施 XSL 变换，将变换后的文本嵌入到 HTML 文件中。模板调用标记的执行过程如图 4-7 所示。

对于不带 select 属性的模板调用标记，需要作为“xsl:for-each”标记的子标记来使用：

```
<xsl:for-each select="标记匹配模式" >
    <xsl:apply-templates />
< xsl:for-each >
```

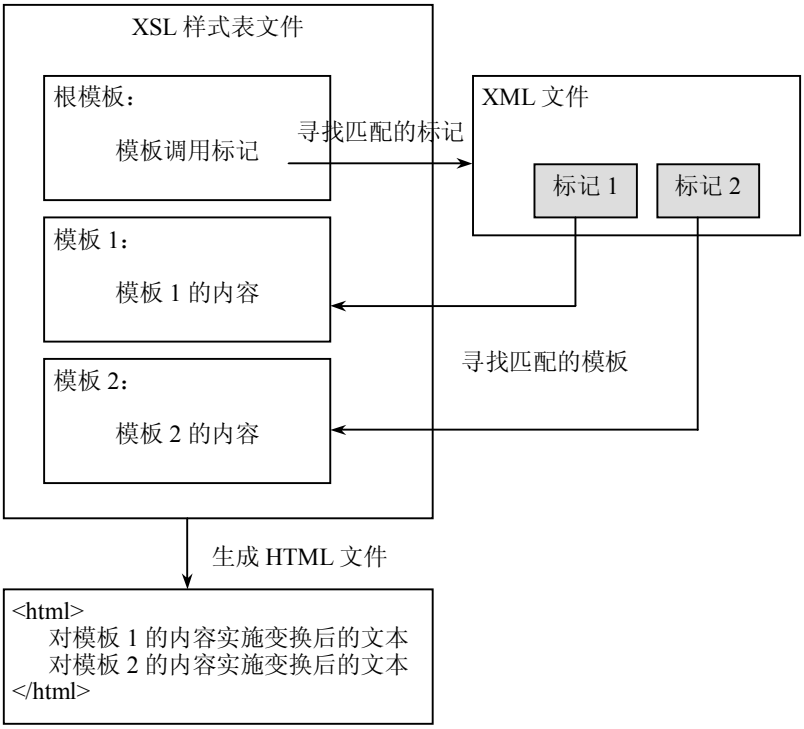

图 4-7　模板调用的执行过程

XSL 处理器首先根据<xsl:for-each select="标记匹配模式" >中的“标记匹配模式”，到 XML 文件中寻找所有和“标记匹配模式”匹配的 XML 标记，然后按照带 select 属性的模板调用标记一样处理，select 属性匹配的 XML 标记跟<xsl:for-each select="标记匹配模式">中匹配的 XML 标记相同。

（三）进行 XSL 变换的方法

1. 关联 XSL

关联 XSL 文件的方法很简单，只在需要使用 XSL 文件来定义样式的 XML 文档中添加如下格式的处理指令就行了：

```
<?xml-stylesheet href="Book4-1.xsl" type="text/xsl" ?>
```

其中：

- type 属性：指明将引用一个 XSL 样式文件来表现文档。
- href 属性：指明被引用的 XSL 文件的位置和名称。

如果 XSL 文件与引用它的 XML 文档在同一个文件夹中，那么只需给出 XSL 文件名。否则将提供相应的 XSL 文件的完整路径。

为这个处理指令提供的 XSL 文件的路径和文件名都必须正确，否则用浏览器来浏览文档时会出现错误提示。这也是将 CSS 样式单与文档链接的方法。是使用 CSS 还是使用 XSL 的唯一区别是 type 属性具有 text/xsl 值，而不是 text/css 值。

2. XSL 文件中使用 HTML 标记

XSL 文件可以直接使用 HTML 标记。XSL 模板的“模板内容”是由 HTML 标记和嵌入其中的 XSL 标记组成，XSL 变换处理器在做变换时，会将 XML 标记直接转化为 HTML 标记，并对 XSL 标记实施变换操作，将变换得到的内容嵌入到 HTML 文件中。如：

```
<xsl:template match="/">
  <html>
  <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <title>
    <xsl:value-of select="FirstPage/title1" />
  </title>
  </head>
  <body background="wz_bg.gif">
  </body>
  </html>
</xsl:template>
```

为了能够更容易地使用 XSL 又不会增加太多新语法，解决方法就是在 XSL 文件中插入已经得到广泛应用的 HTML 标记。可以在 XSL 文件中直接使用 HTML 的标记，同时还可以利用 XSL 增加标记和标记属性的 XSL 元素，即 xsl:element 和 xsl:attribute 来动态地调整 HTML 的格式。

在 XSL 中使用 HTML 标记的另一个好处在于 HTML 超链接的使用。XML 虽然在设计时已经将适用于网络作为目标之一，但是在超链接的部分却是由 XLink 和 XPoint 来完成的。但 XLink 和 XPoint 相关标准的制定比 XML 还慢。因此，就 XML 本身而言，目前还不具备提供完善的超链接的功能，所以，利用将 HTML 的标记嵌入到 XSL 文件中的方式，将使得显示在浏览器中的 XML 文件具备超链接功能。

除了使用 HTML 提供的超链接机制外，将 XSL 与 HTML 组合起来还有一个好处，就是可以将不应该属于 XML 文件资料的部分单独列出来，从而使 XML 文档是一个完全结构化的数据资料。以此实现 XML 文档中的数据能在网络中实现统一传输和交换。

3. HTML 的格式要求

嵌入到 XSL 文件中的 HTML 标记是 XSL 文件的一部分，这就意味着样式表中的 HTML 必须是格式良好的。因为 XSL 文件本身也是一个 XML 文件。虽然在编写 HTML 网页文件的时候，可以按照自己的喜好来书写 HTML 标记，只要浏览器解释通过就可以了。但作为一种增强 Web 健壮性的方法，信息产业企业正在规范使用格式良好的文档数据，并且同时简化和加速文档数据的处理过程。业界期望将来的 HTML 标准应该更符合 XML 规范。所以必须使用一种更加严格的语法。

书写格式良好的HTML代码是很容易的，只要在书写或者转换HTML时注意下面几个方面：

- 所有标记必须封闭。HTML 允许确定的结束标记可以随意取舍，如<P>、
、<IMG>等，但是 XML 要求所有的标记都必须是封闭的。
- 不允许有交叉标记。XML 不允许开始标记和结束标记相互交叉。
- 字符匹配问题。必须为开始标记和结束标记选择一致的大小写格式。在这一点上，并没有指定用大写或小写，只要开始标记和结束标记一致。通常，在 XSL 文件中使用

HTML 标记时使用大写。

- 设置属性。在 XSL 文件中设置 HTML 标记的属性时，属性的值必须用双引号或单引号括起来。这可能是在把 HTML 标记嵌入到 XSL 文件中时最容易犯的错误，因为在 HTML 文档中，可以不为属性值加上引号。但在 XSL 文件中这样做，就不能通过浏览器的语法检查。
- 使用单根。XML 不允许把作为唯一顶层元素的<HTML>标记省略掉。
- 更少的内建实体。XML 仅仅定义了一个内建字符实体的最小集。
- 避免脚本块。HTML 中的脚本块可以包含不可解析的字符，如保留的“<”和“&”，这些字符在 XML 中需要用一个字符实体来代替，或者把脚本块封装在一个 CDATA 节中。另外，JavaScript 的注释是结束于一行的末尾，因此在包含注释的脚本块中保存空白符是很重要的。但在 XML 中，空白将被规格化。这样，终止 JavaScript 注释的新行就看不见了，任何跟在注释后面的 JavaScript 代码都会被认为是注释的一部分而被忽略掉，这种情况经常会导致脚本错误。使用 CDATA 节就能保证空白符号被保留。

4. 在 XSL 样式表中使用 CSS

在 XSL 文件中使用 CSS 有两种方式：一种是可以直接设置在文件中使用的 HTML 标记的 Style 属性；另一种是在文件中使用<Style>、</Style>标记来单独定义 HTML 标记的样式。使用<Style>、</Style>标记的方式如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
…
<style type="text/css">
#text {display:block;
       position:absolute;
       top:30%;
       left:20%;
       width:60%;
       height:60%;
       border:2px solid;}
</style>
…
</xsl:template>
</xsl:stylesheet>
```

直接设置在文件中使用的 HTML 标记的 Style 属性的方式，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
…
  <p style="
     display:block;
     position:relative;
     top:30%;
     left:auto;
     font-family:'宋体';
```

```
        font-size:15pt;
        font-weight:bolder;
        letter-spacing:12pt;
        text-align:center;">
            <xsl:value-of select="." />
        </p>
…
</xsl:template>
</xsl:stylesheet>
```

模块 2　借阅书籍分类展示页面制作

一、知识目标

通过本模块的制作和总结，进一步建立对 XML 文档实施 XSL 变换来制作网页的基本认识。通过学习，将掌握下面这些内容：

- 常用的 XSL 标记的意义及使用方法。
- 利用 XSL 变换来修饰 XML 文档。
- 进一步掌握用 XML 和 XSL 进行网页设计的知识。

二、教学任务

制作借阅书籍分类展示页面，通过在 XML 文档中使用 HTML 标记和在 XML 文档中运用脚本技术，制作借阅书籍分类展示页面。制作结果在 IE 中浏览效果如图 4-8 所示。

图 4-8　借阅书籍分类展示页面效果

三、操作流程

（1）准备借阅书籍分类展示页面的内容。在制作 XML 文档之前，需明确文档要包含的数据内容，才能进一步规划或设计 XML 文档等。该页面包含的内容如下：

借阅书籍分类展示

书名	作者	索取号	出版社	页码	价格	详细信息
二十世纪中国社会科学. 教育学卷	叶澜主编	C092/6	上海人民出版社	403 页	CNY45.00	详细信息
二十世纪中国社会科学. 社会学卷	卢汉龙，彭希哲主编	C092/14	上海人民出版社	538 页	CNY59.00	详细信息
我们时代的文化症候	王光明，胡越主编	C53/84	社会科学文献出版社	428 页	CNY30.00	详细信息
苏格拉底问题	刘小枫，陈少明主编	C53/91	华夏出版社	296 页	CNY35.00	详细信息
中国统计年．200	国家统计局编	C832-54/5	中国统计出版社	38,1011 页	CNY298.00	详细信息
最经典的交际训练手册	郑悦素编著	C912.1/359	蓝天出版社	260 页	CNY19.80	详细信息
生理卫生与疾病防篇	郭晋武主编	C912.1-43/3	武汉大学出版社	412 页	CNY18.00	详细信息
让别人喜欢你的 66 个理由	李先明主编	C912.1-49/111	天津科学技术出版社	195 页	CNY12.80	详细信息
办公室通关密码	王耀明著	C912.1-49/119	中国社会科学出版社	316 页	CNY49.00	详细信息
职场人际 24 戒律	（美）雷克•布林克曼	C912.1-49/122	中信出版社	99 页	CNY16.80	详细信息

（2）编写 XML 文档。根据上面给出的数据内容，编写 XML 文档。以下的代码就是借阅书籍分类展示页面的 XML 源文档，以“Book4-2.xml”文件名保存。

```
<?xml version="1.0" encoding="utf-8"?>
<BookItems>
    <Item>
        <CName>
        二十世纪中国社会科学．教育学卷
        </CName>
        <author>
        叶澜主编
        </author>
        <ItemNum>
        C092/6
        </ItemNum>
```

```
        <publisher>
        上海人民出版社
        </publisher>
        <PageCount>
        403 页
        </PageCount>
        <price>
        CNY45.00
        </price>
    </Item>
    <Item>
        <CName>
        二十世纪中国社会科学．社会学卷
        </CName>
        <author>
        卢汉龙，彭希哲主编
        </author>
        <ItemNum>
        C092/14
        </ItemNum>
        <publisher>
        上海人民出版社
        </publisher>
        <PageCount>
        538 页
        </PageCount>
        <price>
        CNY59.00
        </price>
    </Item>
……
</BookItems>
```

书籍的相关信息是利用 Item 元素来定义的，在 Item 元素下面又分别用 CName、author、ItemNum、publisher、PageCount、price 这几个子元素来定义书名、作者、索取号、出版社页数、价格等信息。在上面的代码里，给出了 Item 元素定义的示例，下面继续在 XML 代码里添加其他的 Item 元素，各子元素的值参见图 4-8。

（3）用 IE 浏览 Book4-2.xml 文件，结果如图 4-9 所示。

（4）编写借阅书籍介绍页面的 XSL 样式表，编写的代码如下，并以“Book4-2.xsl”为文件名进行保存。

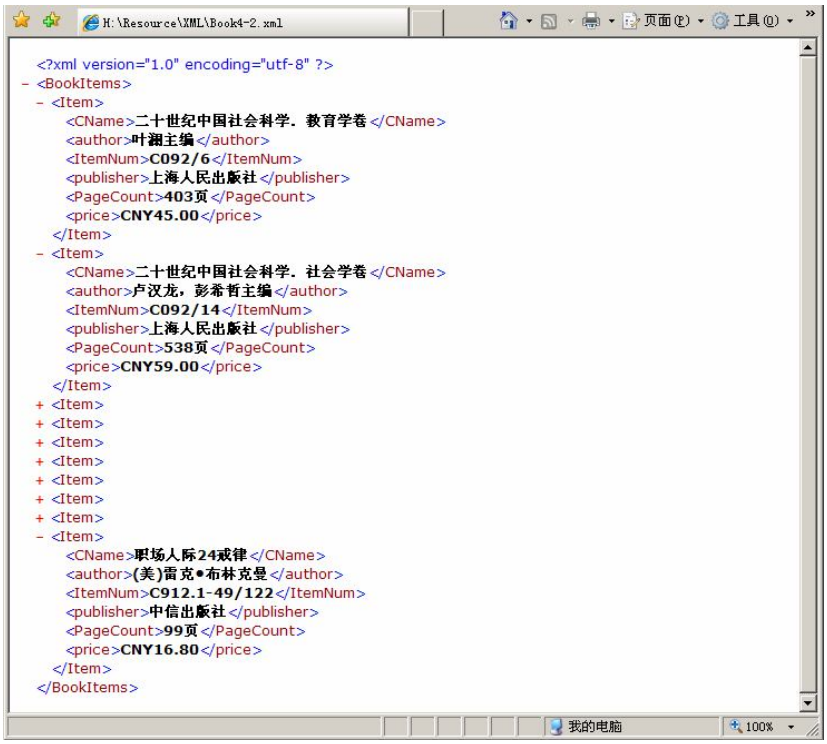

图 4-9 “Book4-2.xml”文件的浏览效果

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>借阅书籍分类展示</title>
</head>
<body>
<h2 align="center"><strong>借阅书籍分类展示</strong></h2>
<hr></hr>
<table width="815" border="1" align="center" cellpadding="0" cellspacing="1">
  <tr height="42" bgcolor="#6699FF">
    <td width="150"><p align="center"><strong>书名 </strong></p></td>
    <td width="95"><p align="center"><strong>作者 </strong></p></td>
    <td width="100"><p align="center"><strong>索取号 </strong></p></td>
    <td width="159"><p align="center"><strong>出版社 </strong></p></td>
    <td width="108"><p align="center"><strong>页码 </strong></p></td>
    <td width="97"><p align="center"><strong>价格</strong></p></td>
    <td width="97"><p align="center"><strong>详细信息</strong></p></td>
  </tr>
  <xsl:for-each select="BookItems/Item">
  <tr height="42">
```

```
      <td><p align="center"><xsl:value-of select="./Cname" /></p></td>
      <td><p align="center"><xsl:value-of select="./author" /></p></td>
      <td><p align="center"><xsl:value-of select="./ItemNum" /></p></td>
      <td><p align="center"><xsl:value-of select="./publisher" /></p></td>
      <td><p align="center"><xsl:value-of select="./PageCount" /></p></td>
      <td><p align="center"><xsl:value-of select="./price" /></p></td>
      <td><p align="center">详细信息</p></td>
    </tr>
    </xsl:for-each>
</table>
<p></p>
</body>
</html>

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

（5）在 XML 文件中添加 XSL 样式表链接。在“Book4-2.xml”的第二行添加如下代码：

```
<?xml-stylesheet type="text/xsl" href="Book4-2.xsl" ?>
```

保存后用 IE 浏览得到如图 4-8 所示的最终效果。

四、知识要点分解

（一）常用的 XSL 标记

样式表的基本结构就是模板，模板也是一种语言，模板中有一个重要的参数就是“match”，该参数的值决定了哪些 XML 标记和该模板相匹配。

模板变换过程中，还可以使用许多重要的子孙标记，比如，前面用过的“xsl:value-of”标记，“xsl:apply-templates”标记和“xsl:for-each”标记等。它们都可以作为模板标记的子标记，而且这些 XSL 标记还可以形成父子关系。下面将陆续介绍这些 XSL 标记。

W3C 推荐（XSLT Version 1.0）的 XSLT 元素如表 4-1 所示。

注意，在表中：

- N：表示最早支持此标签的 Netscape 版本。
- IE：表示最早支持此标签的 Internet Explorer 版本。

在 IE5 中所支持的元素可能出现非标准的行为，这是由于 IE5 发布于 XSLT 被确立为正式的 W3C 标准之前。

表 4-1　XSLT Version 1.0 的 XSLT 元素

元素	描述	IE	N
apply-imports	应用来自导入样式表中的模板规则	6.0	
apply-templates	向当前元素或当前元素的子元素应用模板	5.0	6.0
attribute	向元素添加属性	5.0	6.0
attribute-set	创建命名的属性集	6.0	6.0

续表

元素	描述	IE	N
call-template	调用一个指定的模板	6.0	6.0
choose	与<when>以及<otherwise>协同使用，来表达多重条件测试	5.0	6.0
comment	在结果树中创建注释节点	5.0	6.0
copy	创建当前节点的一个备份（无子节点及属性）	5.0	6.0
copy-of	创建当前节点的一个备份（带有子节点及属性）	6.0	6.0
decimal-format	定义当通过 format-number() 函数把数字转换为字符串时，所要使用的字符和符号	6.0	
element	在输出文档中创建一个元素节点	5.0	6.0
fallback	假如处理器不支持某个 XSLT 元素，规定一段备用代码来运行	6.0	
for-each	遍历指定的节点集中的每个节点	5.0	6.0
if	包含一个模板，仅当某个指定的条件成立时应用此模板	5.0	6.0
import	用于把一个样式表中的内容导入另一个样式表中	6.0	6.0
include	把一个样式表中的内容包含到另一个样式表中	6.0	6.0
key	声明一个命名的键	6.0	6.0
message	向输出写一条消息（用于错误报告）	6.0	6.0
namespace-alias	把样式表中的命名空间替换为输出中不同的命名空间	6.0	
number	测定当前节点的整数位置，并对数字进行格式化	6.0	6.0
otherwise	规定 <choose> 元素的默认动作	5.0	6.0
output	定义输出文档的格式	6.0	6.0
param	声明一个局部或全局参数	6.0	6.0
preserve-space	用于定义保留空白的元素	6.0	6.0
processing-instruction	生成处理指令节点	5.0	6.0
sort	对结果进行排序	6.0	6.0
strip-space	定义应当删除空白字符的元素	6.0	6.0
stylesheet	定义样式表的根元素	5.0	6.0
template	当指定的节点被匹配时所应用的规则	5.0	6.0
text	通过样式表生成文本节点	5.0	6.0
transform	定义样式表的根元素	6.0	6.0
value-of	提取选定节点的值	5.0	6.0
variable	声明局部或者全局的变量	6.0	6.0
when	规定 <choose> 元素的动作	5.0	6.0
with-param	规定需被传入某个模板的参数的值	6.0	6.0

对于常见的标记的使用，将在下面分别举例说明。为了举例的方便，在没有特别指出的情况下，将只给出 XSL 样式表文件的代码，而 XML 文件的代码，则基于下面的清单 4-1 中的

XML 文件代码清单。

清单 4-1　Atom.xml 文件的代码

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="14-2.xsl"?>

<PERIODIC TABLE>
    <ATOM STATE="GAS">
        <NAME>Hydrogen</NAME>
        <SYMBOL>H</SYMBOL>
        <ATOMIC_NUMBER>l</ATOMIC_NUMBER>
        <ATOMIC_WLIGHT>1.00794</ATOMIC_WEIGHT>
        <BOILING_POINT UNITS="Kelvin">20.28</BOILING_POINT>
        <MELTING_POINT UNITS="Kelvin">13.81</MELTING_POINT>
        <DENSITY UNITS="grdMS/cubic centimeter"><!- At 300K ->
        0.0899
        </DENSITY>
    </ATOM>
    <ATOM STATE="GAS">
        <NAME>Helium</NAME>
        <SYMBOL>He</SYMBOL>
        <ATOMIC_NUMBER>2</ATOMIC_NUMBER>
        <ATOMIC_WEIGHT>4.0026</ATOMIC_WEIGHT>
        <BOILING_POINT UNITS="Kelvin">4.216</BOILING_POINT>
        <MELTING_POINT UNITS="Kelvin">0.95</MELTING_POINT>
        <DENSITY UNITS="grams/cubic centimeter"><!- At 300K ->
        0.1785
        </DENSITY>
    </ATOM>
</PERIODIC_TABLE>
```

其中，样式表连接的语句：<?xml-stylesheet type="text/xsl" href="14-2.xsl"?>将根据不同的例子而有所改变。

1. 使用 xsl:value-of 标记

定义和用法：

<xsl:value-of> 元素可提取选定节点的值。

<xsl:value-of> 元素可用于选取某个 XML 元素的值，并把它输出。

注释：select 属性（必选）的值是一个 XPath 表达式。它的工作原理类似对文件系统的定位，比如用一个斜杠来选择子目录。

语法：

```
<xsl:value-of select="expression" disable-output-escaping="yes|no"/>
```

属性：

属性	值	描述
select	expression	必需。XPath 表达式，规定了从哪个节点/属性来提取值
disable-output-escaping	yes no	默认值为“no”。如果值为“yes”，通过实例化<xsl:text>元素生成的文本节点在输出时将不进行任何转义。如果设置为“yes”，则“<”将不进行转换。如果设置为“no”，则被输出为“<”

“xsl:value-of”XSL 标记的格式如下：

<xsl:value-of select="标记匹配模式"/>

该 XSL 标记必须在模板中使用，作为模板的子标记。当“标记匹配模式”是特殊的“.”号时，可以将<xsl:value-of select="."/>简写为<xsl:value-of />

xsl:value-of 元素把输入文档中的节点值复制到输出文档中。xsl:value-of 元素的 select 特性指定正在获取的是哪个节点值。

例如，假设要将文字“An Atom”替换为由“NAME”子元素内容给出的“ATOM”元素的名称，可用<xsl:value-of select="NAME"/>替换“An Atom”，如下所示：

```
<xsl:template match="ATOM">
<xsl:value of select="NAME"/>
</xsl:template>
```

然后，当将样式单应用于清单 4-1 时，产生如下文本。

```
<html><head/><body>
Hydrogen
Helium
</body></html>
```

选择其值的项目（本例中的 NAME 元素）是与源节点有关的。源节点是由模板来匹配的项目（本例中特指“ATOM”元素）。因此，当氢（Hydrogen）的 ATOM 与<xsl:template match= "ATOM">相匹配，氢（Hydrogen）的 ATOM 的 NAME 元素就由 xsl:value-of 选定了。当氦（Helium）的 ATOM 与<xsl:template match= "ATOM">相匹配时，氦（Helium）的 ATOM 的 NAME 元素就由 xsl:value-of 选定了。

节点的值总是字符串，有时可能为空字符串。此字符串的精确内容由节点的类型而定。最普通的节点类型为元素，元素节点的值特别简单，就是在元素的开始标记和结束标记之间的所有可析字符数据（但不是标记！）。例如，清单 4-1 中的第一个 ATOM 元素如下所示.

```
<ATOM STATE="GAS">
<NAME>Hydrogen</NAME>
<SYMBOL>H</SYMBOL>
<ATOMIC_NUMBER>l</ATOMIC_NUMBER>
<ATOMIC_WEIGHT>1.00794</ATOMIC_WEIGHT>
<OXIDATION_STATES>1</OXIDATION_STATES>
<BOILING_POINT UNITS="Kelvin">20.28</BOILING_POINT>
<MELTING_POINT UNITS="Kelvin">13.81</MELTING_POINT>
<DENSITY UNITS="grams/cubic centimeter"><!- At 300K ->
0.0899
</DENSITY>
</ATOM>
```

元素的值显示如下：

```
Hydrogen
H
1
1.00794
1
20.28
```

13.81
0.0899

通过删除所有的标记和注释后计算出了这些值。包括空格在内的其他一切内容都完整无缺地保留着。其他六个节点类型的值也可以类似的非常明显的方式加以计算。总结了这些值的结果填入下表。

节点类型	值
根节点	根元素的值
元素	包括在元素中的所有可析的字符数据（包括元素的任何后代中的字符数据）
文本	节点的文本，实际上为节点本身
属性	标准化的属性值（详细说明见 XML 1.0 推荐的 3.3.3 节）；主要为实体还原后的属性值，截去前导和后继的空格；不包括属性名、等号或引号
命命名空间	用于命命名空间的 URL
处理指令	处理指令的值，不包括<?或?>以及处理指令名
注释	注释文本，不包括<!--和—>

2. 使用 xsl:for-each 标记

定义和用法：

<xsl:for-each> 元素可遍历指定的节点集中的每个节点。

语法：

```
<xsl:for-each select="expression">
  <!-- Content:(xsl:sort*,template) -->
</xsl:for-each>
```

属性：

属性	值	描述
select	expression	必需。被处理的节点集

xsl:value-of 元素只用于能够不含糊地确定要获取哪个节点值的上下文中。如果有多个可能项可供选择，那么只选择第一项。例如，由于普通的“PERIODIC_TABLE”元素包含一个以上的“ATOM”，所以下列的规则较差。

```
<xsl:template match="PERIODIC_TABLE">
<xsl:value-of select="ATOM"/>
</xsl:template>
```

有两种方法可依次处理多个元素。第一种方法已经看到了，只需要按下列方式与 select 属性（它选择想要包括的特定元素）一起使用 xsl:apply-templates。

```
<xsl:template match="PERIODIC_TABLE">
<xsl:apply-templates select="ATOM"/>
</xsl:template>
<xsl:template match="ATOM">
<xsl:value-of select="."/>
</xsl:template>
```

第二个模板中的 select="."告诉格式化程序取匹配的元素（本例中的“ATOM”）的值。

第二种方法是使用 xsl:for-each。xsl:for-each 元素依次处理由 select 属性选择的每个元素。不过，无需任何附加的模板。例如：

```
<xsl:template match="PERIODIC_TABLE">
<xsl:for-each select="ATOM">
<xsl:value-of select="."/>
</xsl:for-each>
</xsl:template>
```

如果省略 select 特性，那么处理源节点（本例中的“PERIODIC_TABLE”）的所有子节点。

```
<xsl:template match="PERIODIC_TABLE">
<xsl:for-each>
<xsl:value-of select="ATOM"/>
</xsl:for-each>
</xsl:template>
```

3. 使用 xsl:element 标记

通常，只使用文字元素本身就可以将元素插入到输出文档中。例如，要插入 P 元素，只需要在样式单的适当位置键入<P>和</P>。但是，偶尔也需要使用输入文档的详细内容，来确定将哪个元素放在输出文档中。例如，当将使用特性来提供信息的源符号集变换成使用元素来提供相同信息的输出符号集时，就是这种情况。

xsl:element 元素将元素插入到输出文档中。元素名由 xsl:element 元素的 name 属性中的属性值模板给出。元素的内容来自于 xsl:element 元素的内容，此元素可能包括要插入这些项的 xsl:attribute、xsl:pi 和 xsl:comment 指令（下面讨论所有的指令）。

假设根据 STATE 特性的值，要用“GAS”、“LIQUID”和“SOLID”元素来代替“ATOM”元素。使用 xsl:element 将“STATE”属性值转换为某个元素名，从而只需要一条规则就可以做到这一点。具体作法如下所示。

```
<xsl:template match="ATOM">
<xsl:element name="{@STATE}">
<NAME><xsl:value-of select=”NAME”/></NAME>
<!- rules for other children ->
</xsl:element>
</xsl:template>
```

使用更为复杂的属性值模板，就可以实现所需的大多数运算。

4. 使用 xsl:attribute 标记

只使用文字属性，就可以将属性包括在输出文档中。例如，要插入带有 ALIGN 属性（其值为 CENTER）的 DIV 元素，只需在样式单的适当位置处键入<DIV ALIGN="CENTER">和</DIV>即可。但是，为了确定属性值，有时甚至是为了确定属性名，常常不得不依赖于从输入文档中读取的数据。

例如，假设要获得一样式单，可选择原子（ATOM）名，并把这些原子（ATOM）名格式化为与 H.html、He.html、Li.html 等文件的链接：

```
<LI><A HREF="H.html">Hydrogen</A></LI>
<LI><A HREF="He.html">Helium</A></LI>
<LI><A HREF="Li.html">Lithium</A></LI>
```

在输入文档中，每个不同的元素都有一个不同的 HREF 属性值。xsl:attribute 元素计算属

性名和值，并将它插入到输出文档中。每个 xsl:attribute 元素要么是 xs1:element 元素的子元素，要么是文字元素。在输出中，xsl:attribute 计算出来的属性关联到与父元素计算出来的元素上。属性名是由 xsl:attribute 元素的 name 属性指定的。属性值是由 xsl:attribute 元素的内容给出的。例如，下面的模板规则将产生上面显示的输出结果。

```
<xsl:template match="ATOM">
<LI><A>
<xsl:attribute name="HREF">
<xsl:value-of select="SYMBOL"/>.html
</xsl:attribute>
<xsl:value-of select="NAME"/>
</A></LI>
</xsl:template>
```

所有的 xsl:attribute 元素都必须放在其父元素的其他任何内容之前。在已经开始写出元素内容之后，就不能将属性加到元素中。例如，下面的模板是非法的。

```
<xsl:template match="ATOM">
<LI><A>
<xsl:value-of select="NAME"/>
<xsl:attribute name="HREF">
<xsl:value-of select="SYMBOL"/>.html
</xsl:attribute>
</A></LI>
</xsl:template>
```

5. 使用 xsl:pi 生成处理指令标记

xsl:pi 元素将指令放在输出文档中。处理指令的目标由所需的 name 属性指定。xsl:pi 元素的内容成为处理指令的内容。例如，下面的规则将“PROGRAM”元素用 gcc 处理指令代替。

```
<xsl:template select="PROGRAM">
<xsl:pi name="gcc"> -04</xsl:pi>
</xsl:template>
```

输入文档中的“PROGRAM”元素由输出文档中的下面的处理指令所代替。

```
<?gcc -04?>
```

若这些指令的结果为纯文本，那么 xsl:pi 元素的内容可包括 xsl:value-of 元素和 xsl:apply-templates 元素。例如，

```
<xsl:template select="PROGRAM">
<xsl:pi name="gcc">-04 <xsl:value-of select="NAME"/></xsl:pi>
</xsl:template>
```

xsl:pi 的最常用的用途之一，就是当从 XML 生成 XML 时，用来插入 XML 声明（尽管 XML 声明在技术上并不是处理指令）。例如：

```
<xsl:pi name="xml">version="1.0" standalone="yes"</xsl:pi>
```

xsl:pi 元素不能包括 xsl:element 和在结果中产生元素和属性的其他指令。此外，它还不能包括在输出文档中插入“?>”的任何指令和文字文本，因为这会使处理指令提前结束。

6. 使用 xsl:comment 生成注释标记

使用 xsl:comment 元素可以在输出文档中插入注释。它没有属性。其内容为注释文本。例如，

```
<xsl:template select="ATOM">
```

<xsl:comment>There was an atom here once.</xsl:comment>
</xsl:template>

此规则使用下面的输出代替“ATOM”节点。

<!-There was an atom here once.->

如果 xsl:value-of 元素和 xsl:apply-templates 元素指令的结果是纯文本的话，那么 xsl:comment 元素的内容可包括这些元素。它不能包括 xsl:element 以及在结果中产生元素和属性的其他指令。此外，xsl:comment 还不能包括在注释中插入双连字符的任何指令或文字文本。这样在输出文档中会使注释很难看，所以这种情况是不允许的。

7. 使用 xsl:text 生成文本

xsl:text 元素将其内容作为文字文本插入到输出文档中。例如，下面的规则将每个 ATOM 元素用字符串“There was an atom here once.”代替。

<xsl:template select="ATOM">
<xsl:text>There was an atom here once.</xsl:text>
</xsl:template>

xsl:text 元素用得不多，这是因为在多数情况下，键入文本更容易。但是，xsl:text 的确有一个优点，它可以准确地保留空白。当处理诗句、计算机源代码或空白显示具有重要意义的其他信息时，使用 xsl:text 是很有用的。

8. 使用 xsl:copy 复制当前节点

xsl:copy 元素将源代码复制到输出文档中。子元素、属性和其他内容不会自动复制。但是，xsl:copy 元素的内容也是选择要复制这些内容的 xsl:template 元素。当将文档从某个标记符号集转换成相同的或相近的相关标记符号集时，这种方法通常是有用的。例如，下面的模板规则删除原子（ATOM）的属性和子元素，并用其内容值来代替：

<xsl:template match="ATOM">
<xsl:copy>
<xsl:apply-templates/>
</xsl:copy>
</xsl:template>

xsl:copy 使模板具有的用途之一就是恒等转换，也就是说，可将一文档转换成本身。这种转换与下面类似：

<xsl:templdte match="*|@*|comment()|pi()|text()">
<xsl:copy>
<xsl:apply-templates select="*|@*|comment()|pi()|text()"/>
</xsl:copy>
</xsl:template>

可对恒等转换进行稍微调节，以产生相似的文档。例如，清单 4-2 是一样式单，它可去掉文档中的注释而文档的其他部分不受影响。在恒等转换中，去掉 comment()节点的 match 和 select 属性值，而保留此节点的其他部分就可以产生这种结果。

清单 4-2　从文档中删除注释的 XSL 样式单

<?xml version="1.0"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
<xsl:template match="* | @* | pi() | text()">

```
< xsl:copy>
<xsl:apply-templates select="* | @* | pi() | text()"/>
</xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

xsl:copy 只复制源节点。使用 xsl:copy-of，可以复制其他节点，可能不止一个。xsl:copy-of 的 select 属性选择要复制的节点。例如，清单 4-3 是一样式单，它使用 xsl:copy-of，只复制有 MELTING_POINT（熔点）子元素的 ATOM 元素，从而将没有 MELTING_POINT（熔点）的元素从周期表（PKRIODIC_TABLE）中去掉。

清单 4-3　只复制有“MELTING_POINT”子元素的“ATOM”元素的样式单

```
<?xml version="1.0"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
<xsl:template match="/PERIODIC_TABLE">
<PERIODIC_TABLE>
<xsl:apply-templates select="ATOM"/>
</PERIODIC_TABLE>
</xsl:template>
<xsl:template match="ATOM">
<xsl:apply-templates
select="MELTING_POINT"/>
</xsl:template>
<xsl:template match="MELTING_POINT">
<xsl:copy-of select="..">
<xsl:apply-templates select="*|@*|pi()|text()"/>
</xsl:copy-of>
</xsl:template>
<xsl:template match="* | @* | pi() | text()">
<xsl:copy>
<xsl:apply-templates select="* | @* | pi() | text()"/>
</xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

这是一个从源符号集到同一个符号集的 XSL 转换的例子。不像本章中的大多数例子那样，此例不转换成结构整洁的 HTML。

9. 使用 xsl:number 为节点计数

xsl:number 在输出文档中插入格式化整数。由“expr”属性计算出来的数值，四舍五入成最接近的整数，然后根据“format”属性值，对此整数进行格式化，从而获得整数值。为这两个属性提供了恰当的缺省值。例如，考查清单 4-4 中的“ATOM”元素的样式单。

清单 4-4　为原子计数的 XSL 样式单

```
<?xml version="1.0"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
<xsl:template match="PERIODIC_TABLE">
```

```
<html>
<head><title>The Elements</title></head>
<body>
<table>
<xsl:apply-templates select="ATOM"/>
</table>
</body>
</html>
</xsl:template>
<xsl:template match="ATOM">
<tr>
<td><xsl:number expr="position()"/></td>
<td><xsl:value-of select="NAME"/></td>
</tr>
</xsl:template>
</xsl:stylesheet>
```

将清单 4-4 中的样式单应用于清单 4-1 时，输出类似如下：

```
<html><head><title>The
Elements</title></head><body><table><tr><td>l</td><td>Hydrogen
</td></tr>
<tr><td>2</td><td>Helium</td></tr>
</table></body></html>
```

由于氢（Hydrogen）是其父元素的第一个“ATOM”元素，所以其号码为 1。由于氦（Helium）是其父元素的第二个 ATOM 元素，所以其号码为 2（这些号码对应于氢和氦的原子序数，这种对应关系是清单 4-1 的副产品，而清单 4-1 正是以原子序数的顺序进行排列的）。

10. 使用 xsl:if 标记

xsl:if 元素提供了根据模式来改变输出文档的简单途径。xsl:if 的 test 属性含有选择表达式，用来计算布尔值。如果此表达式为 true，即输出 xsl:if 元素的内容；否则，不输出 xsl:if 元素的内容。例如下面的模板取消所有“ATOM”元素的名称。除列表中的最后一个元素外，在所有的元素后加入一个逗号和一个空格。

```
<xsl:template match="ATOM">
<xsl:value-of select="NAME"/>
<xsl:if test="not(position()=last())">, </xsl:if>
</xsl:template>
```

本模板确保列表类似于“Hydrogen, Helium”的样子，而不是“HydrogenHelium”的样子。

不存在 xsl:else 或 xsl:else-if 元素。由 xsl:choose 元素提供了类似功能。

11. 使用 xsl:choose 标记

根据几个可能的条件，xsl:choose 元素从几个输出结果中选择一个。xsl:when 子元素提供各种条件及其相关的输出模板。xsl:when 元素 test 属性为布尔值的选择表达式。如果多个条件都为真，那么只显示第一个为真的元素的内容。如果 xsl:when 元素都不为真，那么显示 xsl:otherwise 子元素的内容。例如，下面的规则根据“ATOM”元素的“STATE”属性是为

“SOLID”、“LIQUID”还是“GAS”，来改变输出文档的颜色：

```
<xsl:template match="ATOM">
<xsl:choose>
<xsl:when test="@STATE='SOLID'">
<P style="color:black">
<xsl:value-of select="."/>
</P>
</xsl:when>
<xsl:when test="@STATE='LIQUID">
<P style="color:blue">
<xsl:value-of select="."/>
</P>
</xsl:when>
<xsl:when test="@STATE='GAS">
<P style="color:red">
<xsl:value-of select="."/>
</P>
</xsl:when>
<xsl:other>
<P style="color:green">
<xsl:value-of select="."/>
</P>
</xsl:other>
</xsl:choose>
</xsl:template>
```

模块 3　借阅书籍介绍页面制作

一、知识目标

通过本模块的制作和总结，进一步掌握 XSL 样式表的使用，以及合并多个样式表的使用方法。通过学习，主要将掌握下面这些内容：

- 掌握如何匹配节点的模式。
- 掌握几种匹配符号的使用方法。
- 熟练运用结合 XML 和 XSL 进行网页设计的技能。

二、教学任务

在本模块中主要制作一个新书介绍页，运用在 HTML 中使用 XML 标记的方法。制作结

果在 IE 中浏览效果如图 4-10 所示。

图 4-10　新书介绍页网页效果

三、操作流程

（1）准备借阅书籍介绍页面的内容。在制作 XML 文档之前，需明确文档要包含的数据内容，才能进一步规划或设计 XML 文档等。该页面包含的内容如下：

借阅书籍介绍

	书名：　Ajax 基础教程 作者：　（美）阿斯利森 • 舒塔 出版社：　人民邮电出版社 出版时间：　2006 年 02 月 ISBN：　ISBN：7-115-14481-8/TP • 5211 定价：　35.00 元
内容简介：	Ajax 技术可以提供高度交互的 Web 应用，给予用户更丰富的页面浏览体验。本书重点介绍 Ajax 及相关的工具和技术，主要内容包括 XMLHttpRequest 对象及其属性和方法、发送请求和处理响应、构建完备的 Ajax 开发工具、使用 JsUnit 测试 JavaScript、分析 JavaScript 调试工具和技术，以及 Ajax 开发模式和框架等。本书中所有例子的代码都可以从 Apress 网站本书主页的源代码（Source Code）免费得到。本书适合各层次 Web 应用开发人员和网页设计人员阅读

精彩导读：	几年前开始构建 Web 应用时，我们感觉这简直就是软件开发的“圣杯”。以前，我们一直开发的都是胖客户应用，公司每次发布这种公司应用的新版本时，总是需要将这种应用部署到分散在全国各地的数百个用户那里去，让我们沮丧的是，这种复杂的安装过程不仅冗长而且很容易出错，不仅让开发人员很头疼，用户也非常不满。通过浏览器来部署应用，这看上去相当不错，因为这样，就不再需要在客户端上安装软件了。所以，与许多其他公司一样，我们公司也很快转型，开始在 Web 上部署应用。尽管部署起来相对容易，但 Web 应用也有自己的问题。在用户看来，最突出的问题是用户界面没有了以往丰富的交互性。Web 应用仅限于使用 HTML 提供的一组基本部件，而这是很有限的。更糟糕的是，与服务器交互需要完全刷新页面，很多用户已经熟悉了功能强大的客户—服务器应用，对他们来说，这一点很让人不快。我们曾经一直认为，在 Web 应用中只要刷新页面就必须完全刷新，好像这是在所难免的，所以往往想方设法地避免页面刷新。我们甚至还考虑过编写一个 Java applet，由它处理浏览器和服务器之间的通信。不过，随着越来越多 Web 应用的部署，我们很快发现，用户已经习惯了这种完全页面刷新的方式，这么一来，我们也不再那么强烈地想要另辟蹊径了。……
本书目录：	……

（2）编写 XML 文档。根据上面给出的数据内容，编写 XML 文档。以下的代码就是借阅书籍介绍页面的 XML 源文档，以“Book4-3.xml”文件名保存。

```
<?xml version="1.0" encoding="gb2312"?>
<Book>
    <title>
        Ajax 基础教程
    </title>
    <cover>
        ajax.jpg
    </cover>
    <author>
        （美)阿斯利森  舒塔
    </author>
    <publisher>
        人民邮电出版社
    </publisher>
    <pubtime>
        2006 年 02 月
    </pubtime>
    <price>
        35.00 元
    </price>
    <number>
    ISBN:7-115-14481-8/TP·5211
    </number>
    <abstract> Ajax 技术可以提供高度交互的 Web 应用，给予用户更丰富的页面浏览体验。本书重点介绍 Ajax 及相关的工具和技术，主要内容包括 XMLHttpRequest 对象及其属性和方法、发送请求和处理响应、构建完备的 Ajax 开发工具、使用 JsUnit 测试 JavaScript、分析 JavaScript 调试工具和技术，以及 Ajax 开发模式和框架等。本书中所有例子的代码都可以从 Apress 网站本书主页的源代码（Source Code）免费得到。本书适合各层次 Web 应用开发人员和网页设计人员阅读。
```

</abstract>

<comment>　　几年前开始构建 Web 应用时，我们感觉这简直就是软件开发的“圣杯”。以前，我们一直开发的都是胖客户应用，公司每次发布这种公司应用的新版本时，总是需要将这种应用部署到分散在全国各地的数百个用户那里去，让我们沮丧的是，这种复杂的安装过程不仅冗长而且很容易出错，不仅让开发人员很头疼，用户也非常不满。通过浏览器来部署应用，这看上去相当不错，因为这样，就不再需要在客户端上安装软件了。所以，与许多其他公司一样，我们公司也很快转型，开始在 Web 上部署应用。 尽管部署起来相对容易，但 Web 应用也有自己的问题。在用户看来，最突出的问题是用户界面没有了以往丰富的交互性。Web 应用仅限于使用 HTML 提供的一组基本部件，而这是很有限的。更糟糕的是，与服务器交互需要完全刷新页面，很多用户已经熟悉了功能强大的客户—服务器应用，对他们来说，这一点很让人不快。我们曾经一直认为，在 Web 应用中只要刷新页面就必须完全刷新，好像这是在所难免的，所以往往想方设法地避免页面刷新。我们甚至还考虑过编写一个 Java applet，由它处理浏览器和服务器之间的通信。不过，随着越来越多 Web 应用的部署，我们很快发现，用户已经习惯了这种完全页面刷新的方式，这么一来，我们也不再那么强烈地想要另辟蹊径了。……

</comment>

</Book>

（3）用 IE 浏览“Book4-3.xml”文件，结果如图 4-11 所示。

图 4-11　“Book4-3.xml”文件的浏览效果

（4）编写借阅书籍介绍页面的 XSL 样式表，编写的代码如下，并以“Book4-3.xsl”为文件名进行保存。

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

```
<title>借阅书籍介绍</title>
</head>
<body>
<h2 align="center"><strong>借阅书籍介绍</strong></h2>
<table border="1" align="center">
  <tr>
    <td width="128">
    <xsl:element name="img">
      <xsl:attribute name="src">
          <xsl:value-of select="Book/cover" />
      </xsl:attribute>
      <xsl:attribute name="width">
          127
      </xsl:attribute>
      <xsl:attribute name="height">
          155
      </xsl:attribute>
    </xsl:element>
    </td>
    <td width="720">
      <table border="0" align="left">
          <tr>
            <th scope="row" align="left">书名：</th>
            <td><xsl:value-of select="Book/title" /> </td>
          </tr>
          <tr>
            <th scope="row" align="left">作者：</th>
            <td><xsl:value-of select="Book/author" /></td>
          </tr>
          <tr>
            <th scope="row" align="left">出版社：</th>
            <td><xsl:value-of select="Book/publisher" /></td>
          </tr>
          <tr>
            <th scope="row" align="left">出版时间：</th>
            <td><xsl:value-of select="Book/pubtime" /></td>
          </tr>
          <tr>
            <th scope="row" align="left">ISBN：</th>
            <td><xsl:value-of select="Book/number" /></td>
          </tr>
          <tr>
            <th scope="row" align="left">定价：</th>
            <td><xsl:value-of select="Book/price" /></td>
          </tr>
    </table>
```

```
    </td>
    </tr>
    <tr>
      <td >内容简介：</td>
      <td>
      <xsl:value-of select="Book/abstract"/>
      </td>
    </tr>
    <tr>
      <td >精彩导读：</td>
      <td>
      <xsl:value-of select="Book/comment"/>
      </td>
    </tr>
    <tr>
      <td >本书目录：</td>
      <td>…… </td>
    </tr>
  </table>

  </body>
  </html>

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

（5）在 XML 文件中添加 XSL 样式表链接。在“Book4-3.xml”的第二行添加如下代码：

```
<?xml-stylesheet type="text/xsl" href="Book4-3.xsl" ?>
```

保存后用 IE 浏览得到如图 4-10 所示的最终效果。

四、知识要点分解

（一）匹配节点的模式

xsl:template 元素的 match 特性支持复杂的语法，允许人们精确地表达想要或不想要与哪个节点匹配。xsl:apply-templates、xsl:value-of、xsl:for-each、xsl:copy-of 和 xsl:sort 的 select 属性支持功能更加强大的语法的超集，允许人们精确地表达想要或不想要选择哪个节点。下面讨论匹配和选择节点的各种模式。

1. 匹配根节点

为了使输出的文档结构整洁。从 XSL 变换的第一个输出内容应为输出文档的根元素。因此，XSL 样式单一般以应用于根节点的规则开始。要在规则中指定根节点，可将其 match 属性设置为合适的值。例如：

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

```
<xsl:apply-templates/>
</html>
</xsl:template>
```

此规则应用于根节点，并且只应用于输入树形结构的根节点。当读取到此根节点时，就输出<html>标记，处理根节点的子节点，然后输出</html>标记。本规则推翻了根节点的缺省规则。清单 4-5 显示了应用于根节点的带有单一规则的样式单。

清单 4-5　用于根节点的带有单一规则的 XSL 样式单

```
<?xml version="1.0"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
<xsl:template match="/">
<html>
<head>
<title>Atomic Number vs. Atomic Weight</title>
</head>
<body>
<table>
Atom data will go here
</table>
</body>
</html >
</xsl:template>
</xsl:stylesheet>
```

由于本样式单只为根节点提供一条规则，并且由于规则的模板未指明对子节点进行进一步的处理，因而只是按原样输出，所以在模板中所看到的所有内容都将插入到结果文档中。换句话说，将清单 4-5 中的样式单应用于清单 4-1（或其他任何结构整洁的 XML 文档）中，所获得的结果如下：

```
<html><head><title>Atomic Number vs. Atomic
Weight</title></head><body><table>
Atom data will go here
</table></body></html>
```

2. 匹配元素名

正如前面介绍的那样，最基本的模式只包含一个元素名，用来匹配所有带有该名的元素。例如，下面的模板与“ATOM”元素相匹配，并将“ATOM”元素的“ATOMIC_NUMBER”子元素标成粗体。

```
<xsl:template match="ATOM">
<b><xsl:value-of select="ATOMIC_NUMBER"/><b>
</xsl:template>
```

清单 4-6 显示的是扩充了清单 4-5 的样式单。首先，在根节点的规则模板中包括了 xsl:apply-templates 元素。此规则使用 select 属性来确保只有“PERIODIC_TABLE”元素获得处理。

其次，使用 match=“PERIODIC_TABLE”语句创建了只适用于“PERIODIC_TABLE”元素的规则。本规则设置周期表（PERIODIC_TABLE）的标题，然后应用模板来从“ATOM”

元素中生成周期表（PERIODIC_TABLE）的主体。

最后，“ATOM”使用<xsl:apply-templates select="NAME"/>、<xsl:apply-templates select="ATOMIC_NUMBER"/>和<xsl:apply templates select="ATOMIC_WEIGHT"/>规则，明确地选择“ATOM”元素的“NAME”、“ATOMIC_NUMBER”和“ATOMIC_WEIGHT”子元素。它们都包装在 HTML 的 tr 和 td 元素中，以便最终的结果是与原子量（ATOMIC_NUMBER）相匹配的原子（ATOM）序数表。图 4-12 显示将清单 4-6 中的样式单应用于清单 4-1 的输出结果。

对本样式单需要注意的是：输入文档中的 NAME、ATOMIC_NUMBER 和 ATOMIC_WEIGHT 元素的精确顺序是不重要的。它们在输出文档中以选择它们的顺序出现，也就是说首先为原子序数，然后是原子量。相反，在输入文档中，各个原子依字母顺序排序。以后，将会看到如何使用 xsl:sort 元素来改变这个顺序，以便使用更常规的原子序数的顺序来排列原子。

清单 4-6　select 用于元素的特定类的模板

```
<?xml version="1.0"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
<xsl:template match="/">
<html>
<head>
<title>Atomic Number vs. Atomic Weight</title>
</head>
<body>
<xsl:apply-templates select="PERIODIC_TABLE"/>
</body>
</html>
</xsl:template>
<xsl:template match="PERIODIC_TABLE">
<hl>Atomic Number vs. Atomic Weight</hl>
<table>
<th>Element</th>
<th>Atomic Number</th>
<th>Atomic Weight</th>
<xsl:apply-templates select="ATOM"/>
</table>
</xsl:template>
<xsl:template match="ATOM">
<tr>
<td><xsl:value-of select="NAME"/></td>
<td><xsl:value-of select="ATOMIC_NUMBER"/></td>
<td><xsl:value-of select="ATOMIC_WEIGHT"/></td>
</tr>
</xsl:template>
</xsl:stylesheet>
```

3. 使用“/”符号匹配子节点

在 match 属性中并不局限于当前节点的子节点，可使用“/”符号来匹配指定的元素后代。

当单独使用“/”符号时，它表示引用根节点。但是，在两个名称之间使用此符号时，表示第二个是第一个的子代。例如，“ATOM/NAME”引用“NAME”元素，“NAME”元素为“ATOM”元素的子元素。

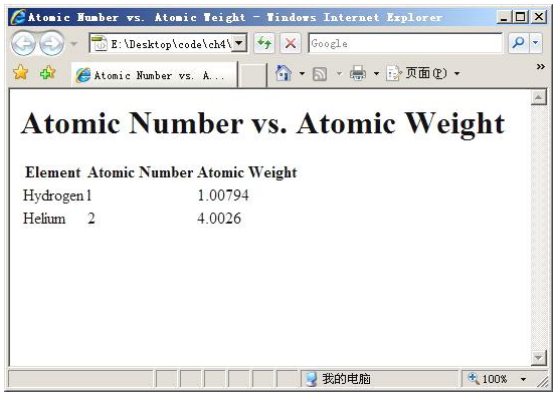

图 4-12　显示的原子序数与原子量的关系表

在 xsl:template 元素中，这种方法能够用来只与某些给定类型的元素进行匹配。例如，下面的模板规则将“ATOM”子元素的“SYMBOL”元素标记为“strong”。此规则与非 ATOM 元素的直系子元素的“SYMBOL”元素无关。

```
<xsl:template match="ATOM/SYMBOL">
<strong><xsl:value-of select="."/></strong>
</xsl:template>
```

请记住，本规则选择的是作为“ATOM”元素子元素的“SYMBOL”元素，而不是选择拥有 SYMBOL 子元素的 ATOM 元素。换句话说，在<xsl:value-of select="."/>中的“.”符号引用的是“SYMBOL”，而不是“ATOM”。

将模式写成一行的形式，就可以指定更深层的匹配。例如，PERIODIC_TABLE / ATOM / NAME 选择的是其父为“ATOM”元素（其父为 PERIODIC_TABLE 元素）的 NAME 元素。

还可以使用“*”通配符来代替层次结构中的任意元素名。例如，下面的模板规则应用于“PERIODIC_TABLE”孙元素的所有 SYMBOL 元素。

```
<xsl:template match="PERIODIC_TABLE/*/SYMBOL">
<strong><xsl:value-of select="."/></strong>
</xsl:template>
```

最后一点，就如上面所看到的那样，单独使用“/”本身，表示选择文档的根节点。例如，下面的规则应用于文档根元素的所有“PERIODIC_TABLE”元素。

```
<xsl:template match="/PERIODIC_TABLE">
<html><xsl:apply templates/></html>
</xsl:template>
```

虽然“/”引用根节点，但“/*”则引用任意根元素。例如，

```
<xsl:template match="/*">
<html>
<head>
<title>Atomic Number vs. Atomic Weight</title>
```

```
</head>
<body>
<xsl:apply-templates/>
</body>
</html>
</xsl:template>
```

4. 使用“//”符号匹配子代

有时候，尤其是使用不规则的层次时，更容易的方法就是越过中间节点，只选择给定类型的所有元素而不管这些元素是不是直系子、孙、重孙或其他所有的元素。双斜杠（//）引用任意级别的后代元素。例如，下面的模板规则应用于“PERIODIC_TABLE”的所有“NAME”子代，而不管它们具有何种层次的关系。

```
<xsl:template match=" PERIODIC_TABLE //NAME">
<i><xsl:value-of select="."/></i>
</xsl:template>
```

周期表实例相当简单，一看就懂，但这种技巧在更深层次，尤其当元素包含该类的其他元素时（例如 ATOM 包含 ATOM），就显得更加重要。

模式开头的操作符选择根节点的任何子节点。例如，下面的模板规则处理所有的“ATOMIC_NUMBER”元素，而同时完全忽略其位置。

```
<xsl:template match="// ATOMIC_NUMBER ">
<i><xsl:value-of select="."/></i>
</xsl:template>
```

5. 通过 ID 匹配

有人或许想把某一特定的样式应用于特定的单一元素中，而不改变该类型的所有其他元素。在 XSL 中实现此目的的最简单的方法是，将样式与元素的 ID 选择符（其中包括以单引号括起来的 ID 值）做到这一点。例如，下面的规则使带有 ID 值为 e47 的元素变为粗体。

```
<xsl:template match="id('e47')">
<b><xsl:value-of select="."/></b>
</xsl:template>
```

当然，上面假设以此方式选择的元素具有在源文档的 DTD 中声明为 ID 类型的特性。但是，通常情况并非如此。首先，许多文档没有 DTD，只不过结构整洁，但不合法。即使有 DTD，也无法确保任何元素都有 ID 类型的属性。可以在样式单中使用 xsl:key 元素，用来把输入文档中的特定特性声明作为 ID 来看待。

6. 使用“@”符号来匹配特性

@符号根据属性名与属性相匹配，并选择节点。方法很简单，只需在要选择的属性前加上@符号。例如，清单 4-7 显示一样式单，用它来输出一张原子序数和熔点对照的表格。不仅写出了 MELTING_POINT 的值，而且也写出了 UNITS 属性的值。这是由于<xsl:value-of select ="@UNITS"/>所获得的结果。

清单 4-7　使用@来选择 UNITS 特性的 XSL 样式单

```
<?xml version="1.0"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
<xsl:template match="/PERIODIC_TABLE">
```

```
<html>
<body>
<hl>Atomic Number vs. Melting Point</hl>
<table>
<th>Element</th>
<th>Atomic Number</th>
<th>Melting Point</th>
<xsl:apply-templates/>
</table>
</body>
</html>
</xsl:template>
<xsl:template match="ATOM">
<tr>
<td><xsl:value-of select="NAME"/></td>
<td><xsl:value-of select="ATOMIC_NUMBER"/></td>
<td><xsl:apply-templates select="MELTING_POINT"/></td>
</tr>
</xsl:template>
<xsl:template match="MELTING_POINT">
<xsl:value-of select="." />
<xsl:value-of select="@UNITS"/>
</xsl:template>
</xsl:stylesheet>
```

回想一下，属性节点的值只是此属性的字符串值。一旦应用清单 4-7 中的样式单，“ATOM”元素就会格式化成如下形式：

```
<tr><td>Hydrogen</td><td>l</td><td>13.8lKelvin</td></tr>
<tr><td>Helium</td><td>2</td><td>0.95Kelvin</td></tr>
```

可以使用各种层次操作符将属性与元素组合起来。例如，“BOILING_POINT/@UNITS”引用“BOILING_POINT”元素的“UNITS”属性。“ATOM/*/@UNITS”就能匹配“ATOM”子元素的任何“UNITS”元素。当与模板规则中的属性匹配时，这种做法是特别有用的。必须记住，要匹配的是属性节点，而不是包含它的元素。最常见的错误是，不知不觉地将属性节点与包含它的元素节点搞混淆。例如，请看下面的规则，它试图将模板应用于具有“UNITS”属性的所有子元素。

```
<xsl:template match="ATOM">
<xsl:apply-templates select="@UNITS"/>
</xsl:template>
```

上面语句实际上做的是，将模板应用于“ATOM”元素中并不存在的“UNITS”属性。

也可以使用*来选择元素的所有属性，例如，BOILING_POINT/@*可选择 BOILING_POINT 元素的所有属性。

7. 使用 comments()来匹配注释

大多数时候，可能应该完全忽略 XML 文档中的注释。要使注释成为文档的必不可少的部分，确实不是好主意。但是，当不得不选择注释时，XSL 确实提供了选择注释的手段。

为了选择注释，可使用 comment()模式。尽管此模式有类似函数的圆括号，但实际上决不带任何参数。要区分不同的注释不太容易。例如，回想一下“DENSITY”元素具有如下的形式：

```
<DENSITY UNITS=”grams/cubic centimeter”><!- At 300K ->
6.51
</DENSITY>
```

此模板规则不仅输出密度的值和单位，而且还打印测量密度的条件。

```
<xsl:template match="DENSITY">
<xsl:value-of select="."/>
<xsl:value-of select="@UNITS"/>
<xsl:apply-templates select="comment()"/>
</xsl:template>
```

清单 4-1 使用注释而不是特性或元素来指定条件，就是为了用于本例。实际应用时，决不要将重要信息放在注释中。XSL 允许人们选择注释的唯一的理由是，为了用样式单把一种标记语言变换成另一种标记语言，同时又能使注释保持不变。选择注释的任何其他方面的用途都意味着原文档设计得不好。下面的规则匹配所有的注释，并使用 xsl:comment 元素将它们再次复制出来。

```
<xsl:template match="comment()">
<xsl:comment><xsl:value-of select="."/></xsl:comment>
</xsl:template>
```

可是，要注意，用于施加模板的缺省规则对注释无效。因此，遇到注释时，如果要使缺省规则起作用，需要包括 xsl:apply-templates 元素，无论注释放在何处，此元素都能选择注释。

使用层次操作符可以选择特定的注释。例如，下面的规则匹配“DENSITY”元素内部的注释：

```
<xsl:template match="DENSITY/comment()">
<xsl:comment><xsl:value-of select="." /></xsl:comment>
</xsl:template>
```

8. 使用 pi()来匹配处理指令

谈到编写结构化的、智能化的、可维护的 XML 时，处理指令并不比注释好。但是都有一些必需的应用，其中包括将样式单附加到文档上。

pi()函数选择处理指令。pi()的参数是放在引号内的字符串，表示要选择的处理指令的名称。如果没有参数，则匹配当前节点的第一个处理指令子节点。但是，可以使用层次操作符。例如，下面的规则匹配根节点的第一个处理指令子节点（很可能是 xml-stylesheet 处理指令）。xsl:pi 元素使用指定的名称和输出文档中的值来插入一个处理指令。

```
<xsl:template match="/pi()">
<xsl:pi name="xml-stylesheet">
type="text/xsl" value="auto.xsl"
</xsl:pi>
</xsl:template/>
```

下列规则也匹配 xml-stylesheet 处理指令，但是通过其名称来匹配。

```
<xsl:template match="pi( xml-stylesheet )">
<xsl:pi name="xml-stylesheet">
<xsl:value-of select="."/>
```

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

事实上，区分根元素和根节点的主要原因之一就是，为了读取和处理序言中的处理指令。尽管 xml-stylesheet 处理指令使用“名称=值”这样的句法，但 XSL 并不把它们当做属性看待，这是因为处理指令不是元素。处理指令的值只是跟在其名称后面的空格和结束符“?>”之间的所有内容。

用来施加模板的缺省规则并不匹配处理指令。因此，遇到 xml-stylesheet 处理指令时，如果要使缺省规则起作用，需要包括“xsl:apply-templates”元素，此元素在适当的地方匹配缺省规则。例如，下面这个用于根节点的模板确实将模板应用于处理指令。

```
<xsl:template match="/">
<xsl:apply-templates select="pi()"/>
<xsl:apply-templates select="*"/>
</xsl:template>
```

9. 用 text()来匹配文本节点

尽管文本节点的值包括在选择的元素值部分中，但它们作为节点通常被忽视。但是，text()操作符确实能够明确选择一个元素的文本子元素。尽管这种操作符有圆括号，但不需要任何参数。例如：

```
<xsl:template match="SYMBOL">
<xsl:value-of select="text()"/>
</xsl:template>
```

此操作符存在的主要原因是为了用于缺省规则。无论作者是否指定缺省规则，XSL 处理程序必须提供下列的缺省规则。

```
<xsl:template match="text()">
<xsl:value-of select="."/>
</xsl:template>
```

这意味着无论何时将模板应用于文本节点，都会输出此节点的文本。如果并不需要这种缺省行为，可以将其推翻。例如，在样式单中，包括下列空模板规则，将会阻止输出文本节点，除非另外的规则明确地匹配。

```
<xsl:template rnatch="text()">
</xsl:template>
```

10. 使用“或”操作符|

竖线（|）允许一条模板规则匹配多种模式。如果节点与某种模式相匹配，则此节点将激活该模板。例如，下面模板规则与“ATOMIC_NUMBER”和“ATOMIC_WEIGHT”元素都匹配。

```
<xsl:template match="ATOMIC_NUMBER|ATOMIC_WEIGHT">
<B><xsl:apply-templates/></B>
</xsl:template>
```

也可以在“|”两边加入空格，这样使代码更清晰。例如：

```
<xsl:template match="ATOMIC_NUMBER | ATOMIC_WEIGHT">
<B><xsl:apply-templates/></B>
</xsl:template>
```

还可以顺次使用两个以上的模式。

（二）合并多个样式单

单一 XML 文档可以使用在许多不同的 DTD 中描述的不同的标记符号集。有时希望将不同的标准样式单用于那些不同的符号集。但是，也可能还要将样式规则用于特定的文档。xsl:import 和 xsl:include 元素可用来合并多个样式单，以便组织和重新将样式单用于不同的符号集和目的。

1. 样式表导入

样式表导入可以使用 xsl:import 进行。xsl:import 元素为顶级元素，其 href 特性提供导入的样式单的 URI。所有的 xsl:import 元素都必须放在 xsl:stylesheet 根元素中的顶级元素中。例如，下面的这些 xsl:import 元素导入 genealogy.xsl 和 standards.xsl 样式单。

```
<xsl:stylesheet  xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:import href="genealogy.xsl"/>
<xsl:import href="standards.xsl"/>
……
</xsl:stylesheet>
```

导入的样式单中的规则可能与执行导入的样式单中的规则发生冲突。如果真是这样，那么执行导入的样式单中的规则优先。如果不同的被导入样式单中的两个规则发生冲突，那么最后那个被导入的（如上面例子中的 standards.xsl）优先。

xsl:apply-imports 元素与 xsl:apply-templates 有点差别，后者只使用被导入的规则。xsl:apply-imports 元素不使用执行导入的样式单中的任何规则。这样就可以访问被导入的规则，否则被导入的规则就会被执行导入的样式单中的规则所覆盖。除了名称不同外，xsl:apply-imports 与 xsl:apply-templates 有一样的句法，唯一的差别是它只与被导入样式单中的模板规则匹配。

2. 样式表包含

样式表的包含可以使用 xsl:include 进行。xsl:include 元素也是顶级元素，它将另一个样式单复制到当前样式单中它所出现的位置处（更确切是说，它将远程文档中 xsl-stylesheet 元素的内容复制到当前文档中）。它的 href 特性提供要包括的样式单的 URI。xsl:include 元素可放在顶级处于最后那个 xsl:import 元素之后的任何地方。

不像 xsl: imporlt 元素所包括的规则那样，xsl:include 元素所包括的规则与执行包括的样式单中的规则具有同样的优先级，利用这种优先级关系来决定是否从一个样式单到另一个样式单的复制和粘贴。

3. 在文档中嵌入样式单

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 样式表也可以嵌入到其他文档内容之中。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 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。

可使用 xsl:stylesheet 直接将 XSL 样式单包括在使用它的 XML 文档中。为达此目的，xsl:stylesheet 元素必须以文档元素的子元素而不是根元素本身的形式出现。它可能有一个 id 特性，用来为其取唯一的名称，此 id 特性是作为 xsl:stylesheet 处理指令中的 href 特性值的形式出现的，紧跟在 anchor（锚）标识符（#）之后。清单 4-8 演示此过程。

清单 4-8　在 XML 文档中嵌入的 XSL 样式单

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="#id(mystyle)"?>
<PERIODIC_TABLE>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
id= "mystyle ">
<xsl:template match="/">
<html>
<xsl:apply-templates/>
</html>
</xsl:template>
<xsl:template match="PERIODIC_TABLE">
<xsl:apply-templates/>
</xsl:template>
<xsl:template match="ATOM">
<P>
<xsl:value-of select="."/>
</P>
</xsl:template>
</xsl:stylesheet>
<ATOM>
<NAME>Actinium</NAME>
<ATOMIC_WEIGHT>227</ATOMIC_WEIGHT>
<ATOMIC_NUMBER>89</ATOMIC_NUMBER>
<OXIDATION_STATES>3</OXIDATION_STATES>
<BOILING_POINT UNITS="Kelvin">3470</BOILING_POINT>
<MELTING_POINT UNITS="Kelvin">1324</MELTING_POINT>
<SYMBOL>Ac</SYMBOL>
<DENSITY_UNITS="grams/cubic centimeter"><!- At 300K ->
10.07
</DENSITY>
<ELECTRONEGATIVITY>1.1</ELECTRONEGATIVITY>
<ATOMIC_RADIUS UNITS="Angstroms">1.88</ATOMIC_RADIUS>
</ATOM>
</PERIODIC_TABLE>
```

知识与能力拓展

4-1　XSL 变换的目的是什么？

4-2　XSL 样式表的基本结构是怎样的？

4-3　XSL 样式表必须有根模板吗？请举例说明根模板的格式。

4-4　假设 XML 文件的根标记是“books”，那么 XML 文件中哪些标记和下列模板匹配？

```
<xsl:template match="books/*/*">
    模板内容...
</xsl:template>
```

4-5　请写出下列 XSL 样式表“Lianxi4-1.xsl”变换后得到的 html 文件。

Lianxi4-1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="lianxi4-1.xsl" ?>
<交通工具>
    <小汽车>
        <价格> 23 万元 </价格>
        <产地> 上海 </产地>
    </小汽车>
    <货车>
        <价格> 8 万元 </价格>
        <产地> 北京 </产地>
    </货车>
</交通工具>
```

Lianxi4-1.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <xsl:template match="/">
    <html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>交通工具</title>
    </head>
    <body>
    <table   border="1" align="center">
      <xsl:for-each select="交通工具/*" >
      <tr>
        <th > <xsl:copy/> </th>
        <xsl:apply-templates />
      </tr>
      </xsl:for-each>
    </table>
    </body>
```

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

        <xsl:template match="//价格">
            <td>
            <xsl:value-of/>
            </td>
        </xsl:template>
        <xsl:template match="//产地">
            <td>
            <xsl:value-of/>
            </td>
        </xsl:template>
    </xsl:stylesheet>
```