

第9章 转换XML

有人将XML说成一种文档格式。有人将它说成一种存储数据的分级模式。按照其他的标准，一个XML文档可以被理解为通过一种网络化的处理机构来遍历数据。每个网络节点存储或处理数据并且将结果传输传给相邻的节点。在这个世界里，一个XML文档是一种流过或流在一个网络中的应用中的数据。然而，我们看到，任何XML文档都是元素的汇集。这些元素是按某种方案组织的（无论是显式的，即通过使用DTD或其他的标准，还是隐式的，即没有一个定义过的标准），并且也可能是一种分级命令。我们也可以说XML文档是一个序列化版本的分级命令——一种用于在处理机构之间交换信息的文本。然而，在内部，这些处理机构并不使用序列化的版本（XML文档）而是使用更便于工作的内部表示。

如果XML真能够帮助我们，来创建可相互跨越平台的与不同应用之间相互交流的复杂的应用，并且我们真的想重用使用XML标识出的数据，共享那些使用XML作为一种通用格式的数据的话，则需要为那些没有对他们的数据使用相同命令的人和应用做一些准备。在这一章，我们将考查一下有哪些方法可以将我们的数据转换为另一种XML词汇表的数据，或只是重新排列一下我们的数据。

在这一章，我们将重点放在处理XML的转换方面。转换XML有很多的原因，所以在这一章，先首先看一下为什么和什么时候要将XML转换为另一种格式。我们主要使用XSL转换方式，尽管将在本章的最后讨论其他转换的方法。

在XML领域中关于可用的不同的转换方法已经引起了激烈的争论。不同的程序员通常喜欢用不同的解决方法来处理转换XML，所以，我们将看一下下面的可选方法中的一些观点，这样你可以决定使用哪一种方法（我们假设在读过的第5章后你已经熟悉DOM，或者已经正在使用它）。

在看完了为什么需要转换文档的几个原因之后，我们将继续看一下使用XSLT作为转换语言。我们将向你介绍基本的语法，让你通过使用XSL来转换文档。这将包括一个将书的命令转换为一种新命令的例子。注意这个转换要求XPath的知识，它是一个在XSLT中使用的规格说明，用来指定一个XML文件的特别部分。XPath已经在前面的章节中详细地说明了。一旦我们看过了XSLT，我们将看一下如何使用DOM和脚本来修改同一本书列表的命令。在看过更多的动态文档之后，我们将使用所学的这章XSL部分和更早的DOM章节中的知识，根据用户的交互作用，创建一个重新排列过表格内容的文档。然后我们将比较两种转换方法。为了将转换包装起来，我们将看一下什么时候你可能会考虑使用不同的方法。总共这一章将涉及：

- 为什么XML转换是必要的。
- 关于XSLT语法的介绍。
- 使用XSLT来转换静态文档的一个例子。
- 使用XSLT来转换更多的动态XML文档。

这个将把你带到为所需要处理的 XML 文档选择转换类型的位置上来，并且教给你在不同转换类型背后的核心原则。

9.1 为什么转换 XML

如果我们使用 XML 来存储基于文本的文件，或者我们接收由其他类型的程序生成的 XML，它是一种混合格式。因为 XML 是一种平台独立的，并且可以在应用中的不同部分之间被转换，这就经常出现人们要求以不同的命令来处理信息。另外，我们可能经常需要根据一个变动的交互文档转换文档的命令。例如，根据一个用户的要求或选择进行文档的重构。转换一般属于下面三个方面之一：

- 命令的转换——从一种 XML 术语转换成另一种，就像翻译一样。例如两种金融标识语言 FPML 和 finML 的例子。
- 创建动态文档——允许用户对文档进行重组、过滤和部分排序，就像允许用户点击一个表格列的头对内容进行排序一样。
- 转换成另一种解释语言——准备提交给用户用在一些浏览器上，如 WAP（无线应用协议，Wireless Application Protocol）、HTML、VOXML 或 SVG（可变的向量图形，Scalable Vector Graphics）。

让我们按顺序看一下每一部分。

9.1.1 在不同词汇表之间转换

如果我们再考虑一下在第 2 章中用 XML 标识出的书的目录，可能对目录数据存在着许多潜在的应用。例如，Wrox 可能在它的站点上或内部网中使用这些书的一个目录，使用在第 3 章我们所创建的 DTD。与此同时，几家书店也几乎要求相同的信息。这个听上去像是一个关于 XML 的理想工作。然而，如果不同的书店使用不同的 DTD 来标识相同的数据，我们需要一种方法来将数据转换成同他们相兼容的版本。

例如，www.wrox.com 站点可能使用下面的 pubCatalog.dtd 来进行标识，这是在第 3 章中我们遇到过的：

程序清单 9-1

```
<Book ISBN="1-861003-11-0" level="Professional" pubdate="11-21-99"
  pageCount="500" authors="multi">
  <Title>Professional XML</Title>
  <Abstract>XML is an important area that you must learn about</Abstract>
  <RecSubjCategories>
    <Category>XML</Category>
    <Category>Programming</Category>
    <Category>Internet Programming</Category>
  </RecSubjCategories>
  <Price>$49.99</Price>
</Book>
```

然而，XYZBooks Inc. 可能要求一种不同格式的数据，就像：

```
<Book>
  <Title>Professional XML</Title>
  <ISBN>1-861003-11-0</ISBN>
  <Abstract>XML is an important area that you must learn about</Abstract>
  <Pubdate>11-21-99</Pubdate>
  <RecSubjCategories>
    <Category>XML</Category>
    <Category>Programming</Category>
    <Category>Internet Programming</Category>
  </RecSubjCategories>
  <Price>$49.99</Price>
</Book>
```

正如你所见，两个<Book>元素的属性有着自己的规则：<ISBN>和<Pubdate>。再强调一次，为了按两种格式准备和存储数据，我们只能将一种转换格式到另一种。

这仅仅是我们需要转换 XML 词汇表为另一种格式的例子中的一个，我想你可能想出更多的例子来。在电子商务中，这类的转换被认为是非常重要，不同的公司可能需要它们的数据的格式不同。另一方面，我们可能甚至决定更新某个已经存在的应用，需要转换旧的 XML（现在对你来说还只是一种想法）为新的命令。

转换在 XML 中担当着一个重要的角色，要记住，一旦使用 XML 做了标识，我们就可能重新使用数据。毕竟，如果我们仅是需要执行一个简单的转换工作，那就没有必要将数据保留两个版本了。XSL 的转换能力非常适合这类转换。

9.1.2 动态转换

上一节考查了一下以不同的方式提供相同的数据，两种方式都要求明确的，静态版本的 XML 文档。但是也意味着我们可能需要做更多的动态转换。如果你考虑过电子表格，在近二十年之前，它无庸置疑地改革了桌面 PC 的使用，用户可以要求点击表格列的头部使数据重排。这就要求一个动态转换。

任何要求用户交互的转换，或产生交互文档的转换，同产生一个静态文档相比是一个相当不同的工作。动态转换通常要求事件处理，它包括编程语言的使用。

因为脚本语言和 DOM 允许无 XSL 的转换，又因为文档对象模板（DOM，Document Object Model）可以被使用在浏览器上，通过与 JavaScript 和其他语言（如 Java，C++，Perl，Visual Basic 或 Python）的绑定，所以有些人宁愿通过 DOM 和脚本（无 XSL）来完成动态转换。在这一章的后面，我们将看一下使用两种方法的例子，还有为什么你优先想使用其中一种方法的原因。

9.1.3 不同的浏览器

许多的 Web 开发者都有着这样的经历，开发并行站点或为不兼容的浏览器开发部分站点是一件头痛的事。对于不兼容的浏览器来说存在着，XML 仅能够服务于 Web 浏览器，理解 XML 可能是另一回事了。然而，如果我们打算用 XML 开发站点，则能够将它转换为不同的标记语言，所以可以从核心的 XML 内容中创建出不同版本的 HTML。让我们看一下它是如何工作的（参见图 9-1）。

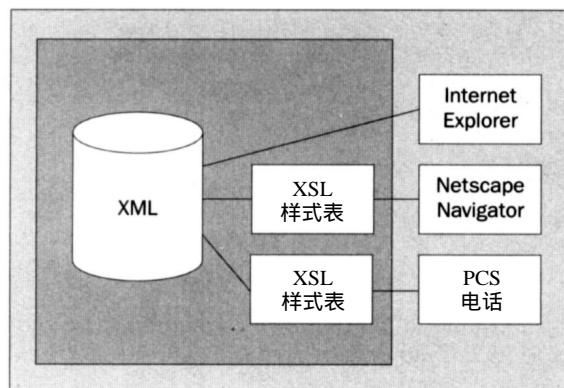


图 9-1

在这里我们使用了三种不同的样式表来创建 XML 内容的三种不同版本。IE5 的版本可以仍然使用 XML，另两种可能是两种不同的解释语言。这个方法避免了给不同的浏览器将内容复制三次。通过转换 XML 数据，几个页面可能使用相同的 XML 内容，可以提供给请求的浏览器按正确的格式翻译好的页面。在这个例子中，我们简单地使用 XSL 样式表作为一个模板来演示数据如何被显示。这些样式表作为下面数据的模板，所以可以使用这些样式表来转换几页的数据。

实际上，在需要显示在浏览器上的时候，将 XML 转换成 HTML 是非常流行的，因为 W3C 关于 XML 的详细说明很少。

由于新型的浏览器在 Internet 上的出现，这个方法开始变得日益重要。我们已经看到了数字电视，游戏控制，多种的移动设备，从手持个人数字助理（PDA，personal digital assistants）到移动电话，都提供 Internet 处理。随着这些不同的客户端的增加，它们分享着浏览器市场，它们将对为它们不同的需要而设计服务页面造成压力。可能包括，转换 XML 为另一种标识语言，就像无线标识语言（WML，Wireless Markup Language），它是一种应用于移动电话和 PDA 中的。所以，将内容转换为另一种版本的能力将变得逐渐普通起来。

9.2 XSL

可扩展样式语言（XSL，eXtensible Stylesheet Language）是一种基于 XML 的语言，它被设计用来转换 XML 文档到另一种 XML 文档或转换 XML 文档为可翻译对象。原始的 XSL 语言已经被分割成三种不同的语言：

- 转换工具（XSLT）。
- 翻译工具（XSLF——可以包括 XSLT 的使用）。
- XML 分级命令处理工具（XPath）。

XSL 有它自己的根，不管是在层叠样式表（CSS）中还是在一种叫 DSSSL（文档样式语义和规格语言（DSSSL——读为 'deessel'））的语言中。随着它的发展，XSL 的样式表现变得更接近于 CSS 和远离 DSSSL。样式化在第 13 章中有描述。

你可能已经猜到，在这一章中所看到的关键点是 XSL 的转换能力。XSLT 规范从 1999 年

11月16日起成为一种推荐的规范。由于它依赖于 XPath 规范，所以同一天 XPath 也成为推荐规范，在文档转换领域作为一种可选方法。

XSLT

这一部分考查了我们如何使用 XSLT 来转换 XML 文档，并且我们将看到 XPath 是如何应用到 XSLT 中的。在 XSLT 规范的第一行中明确指出：“[XSLT] 是一种转换 XML 文档到其他 XML 文档的语言”。如同我们在前面看到的一样，出于某种原因，我们可能需要将 XML 转换为另一种命令。为了实现这一点，需要一个 XSLT 处理器。一旦看过 XSLT 处理器实际的工作之后，我们将讨论两个常用的工具。

XSLT 是用 XML 编写的语言。这就意味着，一个用来转换 XML 的 XSLT 样式的表，实际是一个有着良好命令的 XML 文档。所以，在这一章，我们将学习 XSLT 的语法和了解它能为我们做些什么。

首先让我们先明确一个非常重要的一点：

XSLT 引擎不用来操作文档，而是用于操作命令。

为了让一个 XSLT 引擎能够转换 XML 文档，文档首先要被转换为一种命令 (structure) 或一种内部模型 (internal model)。内部模型是一棵树。这种模型是独立于任何一种处理它的 API 的。在 SGML 世界，这种抽象的模型叫做一个树林 (grove)。因为，XML 是 SGML 的子集，继承了 SGML 的一些基本的概念。所以，树林是一种简单的抽象树状命令，独立于任何一种处理这棵树入口的 API。例如，DOM 就是 W3C 推荐使用于处理树的 API。DOM 是 API，而树林则是抽象命令。所以，一个树林可能有多于一种的 API 或拥有不同语言的不同的 API。在整个章节中，在谈到抽象树状命令时我们使用树林。

看一下下面的 XML：

程序清单 9-3

```
<?xml version="1.0" ?>
<Book ISBN="1-861003-11-0" level="Professional" pubdate="11-21-99"
    pageCount="500" authors="multi">
  <Title>Professional XML</Title>
  <Abstract>XML is an important area that you must learn about</Abstract>
  <RecSubjCategories>
    <Category>XML</Category>
    <Category>Programming</Category>
    <Category>Internet Programming</Category>
  </RecSubjCategories>
  <Price>$49.99</Price>
</Book>
```

可能被表示为一种抽象树的型式，如图 9-2 所示。

它并不关心于我们将如何看待或处理文件，<Title>、<Abstract>、<RecSubjCategories>和<Price>都是<Book>的孩子，并且<Category>是<Book>的孙子，是<RecSubjCategories>的孩子。这就是为什么我们说，这个抽象的树状命令是一种独立于处理它的 API 的一种模型（正如 W3C 的 DOM），并且这种命令是 XSL 处理器所使用的，用于选择命令中的相应部分。XPath 是一种语言，用于处理树中的任何元素。

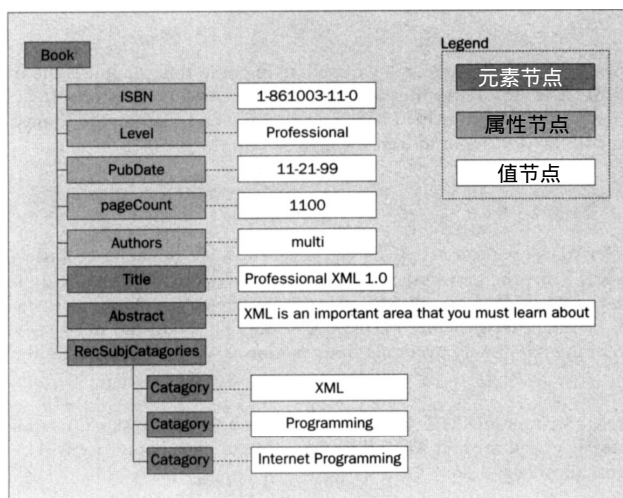


图 9-2

2. XSL 处理器如何转换源文档

正如我们所说，XSLT 操作的是文档的模型而不是语法。源和目标格式都是 XML 的应用，并且两个的分级命令都是一棵树。另外 XSL 样式表是一个 XML 文档，所以它也可以用一棵树来表示。所以，XSLT 处理器一共有三棵树。

XSLT 是一种公布的语言，意思是你来指定结果的显示，胜于说它如何被转换，并且这就是为什么我们使用 XSL 处理器来做这件工作。XSL 样式表是由模板组成，它指定了源树中的每个节点在结果中应如何显示。

图9-3说明了处理器是如何工作的。

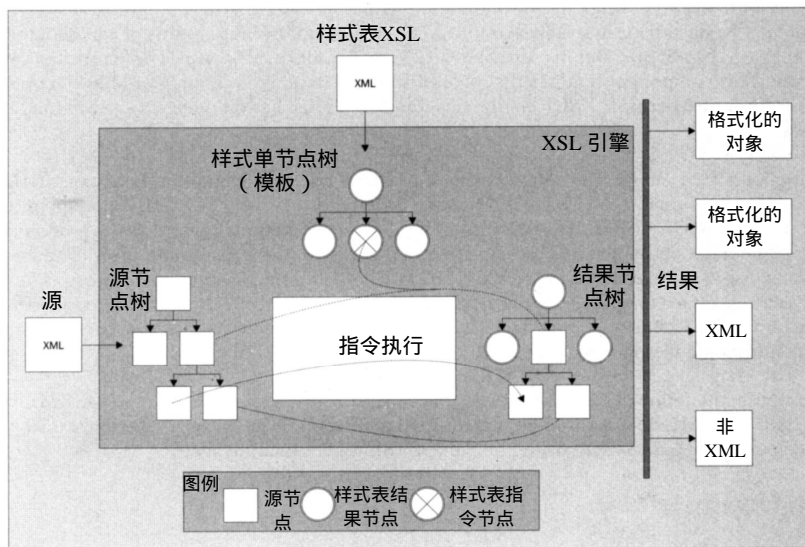


图 9-3

在这里面我们可以看到有三个命令。记住源和结果命令都是文档的抽象表现。处理器遍历源树林，从根开始，查找与在样式表中模板相匹配的节点。当找到一个，它使用在模板中的规则在结果树中写出结果的抽象表示。然后，在源文档中继续移动，一个节点接着一个节点，根据前导的XSLT指令<xsl:apply-template>，在样式表中查找相应的匹配。如果没有可匹配的模板，它就移动到下一个命令。我们可以说它执行一个缺省模板，不输出任何结果。然后，结果树被转换成一个XML文档、文本、HTML文档或希望的结果。

这个，在理论上可能会发生。但是也有不同的变化关于如何建立 XSLT引擎。XSLT引擎可能被优化，并且样式表可能不再需要被存储为树林或树状命令。然而，这给出了它们行为的一般的方法。

在浏览过一个XSL处理器为了执行它的转换是如何工作之后，你将需要确认在机器上已经安装了这个章节的例子。有不同的方法来实现一个 XSL处理器，让我们看一下其中的两个：

- MSXML——Microsoft的XML解析器，有一个DOM接口，以DOM组件的形式包括了一个XSLT引擎。包含在IE5中MSXML引擎与推荐的相比已经相当过时了。技术预览版则更新一些。
- XT——James Clark的致力研究的XSLT引擎。它是用Java编写的，所以可以跨越平台。从最新的XSLT规范的一致性来说XT要更新一些。

MSXML XSL处理器 MSXML不只是一个解析器，它还包括了一个XSL处理器。MSXML同IE5高度集成，但是也可以做为一个独立的COM组件，可从<http://msdn.microsoft.com/xml>得到，用于应用程序的集成。这个组件使用DOM来操作XML文档的抽象树状命令。因为这样，它可以被接口化用在像JavaScript、Delphi、Visual Basic、VisualCOBOL、VBScript、PerlScript、PythonScript、C++等语言中。这个组件要求至少系统中有IE4或以上版本存在，因为Microsoft的XSLT引擎需要一些它提供的其他的DLL。

最初的DOM接口被定义成使用CORBA接口定义语言（IDL，Interface Definition Language），但是Microsoft的组件技术COM使用不同的IDL，所以MSXML组件接口是用COM IDL来定义的。不过，Microsoft的DOM实现，考虑了推荐的精神，在对象接口中保留了同样的方法名。接口IXMLDocument是相当于W3C DOM一级document接口。W3C document接口从node接口派生，很像IXMLDocument接口从IXMLNode接口派生而来。IXMLDocument接口已经被扩展，包含了针对XML文档解析及转换的额外方法。

例如，下面的ASP脚本用于MSXML组件的IXMLDocument接口来解析要转换的XML文档和XSL样式表。然后，它将分析过的源XML文档进行转换，通过使用分析过的XSL转换表：

程序清单 9-4

```
<%@ LANGUAGE = VBScript %>
<%
    ' Set the source and style sheet locations
    sourceFile = Server.MapPath("catalog.xml")
    styleFile = Server.MapPath("catalog.xsl")

    ' Load the XML and get it parsed
    Set source = CreateObject("Microsoft.XMLDOM")
```

```
source.async = false
source.load(sourceFile)
' Load the XSLT and get it parsed
Set style = CreateObject("Microsoft.XMLDOM")
style.async = false
style.load(styleFile)

' Transform the XML document using the XSL transformation sheet.
Response.Write(source.transformNode(style))

%>
```

使用MSXML转换文档的一般机制是：

- 装入要转换的初始文档。load()方法同时会解析文档，以便文档作为一棵树状命令被保存（像我们在前面看到的一样）。
- 装入XSLT文档。load()方法再一次解析文档并将其转换成树。
- 使用transformNode()函数进行转换。这个函数返回一个字符串（一个BSTR）。在返回字符串中包含了转换后的文档。所以，如果XSLT转换表包含一个从XML到HTML转换的话，保存在结果串中的文档就是一个HTML文档。
- 集成在MSXML组件中的组件有。
 - 一个XML解析器。
 - 一个扩展的DOM一级树状接口。
 - 一个XSLT转换引擎。

XT XSL处理器 XT是另一种流行的XSLT处理器；由James Clark组写，易于使用并且可以从作者的主页上自由下载，网址是：<http://www.jclark.com/xml/xt.html>。它使用Java编写，并且已经在几种Java虚拟机上被成功地测试。对于Win32平台，可以下载一个单一的执行码，并且它要求在机器上安装了Microsoft的Java虚拟机。这个处理器将有助于实验这一章中所介绍的不同转换处理。

与MSXML不同，它包括了自己的XML解析器，XT引擎能够操作与任何SAX兼容的解析器（SAX在第6章中已经讨论过）。一旦SAX解析器用Java实现，它将与XT引擎相适应。软件包也包括了一个名为XP的快速Java解析器。

XT是通过命令行来使用的。在Windows下，运行XT比在其他平台上运行要容易。下面的命令行转换一个XML文档，通过XSLT样式表，输出一个HTML文档：

```
xt booklist.xml booklist.xsl booklist.htm
```

XT也可以接受XSLT参数，例如：

```
xt booklist.xml booklist.xsl booklist.htm result=HTML
```

在上面的例子中，result参数被包含在XSLT样式表中，作为一个XSLT变量。这个变量可以被用在XSLT模板中。

XT的一个很大的好处就是它可以在除了Windows以外的平台上运行。然而，不是所有的平台允许运行Java应用程序作为独立可执行的方式，大部分的平台要求通过Java JDK所提供的Java可执行应用程序来运行。例如，下面的命令行将在Linux下运行XT：

```
java -Dcom.jclark.xml.sax.parser=xp.jar com.jclark.xml.sax.Driver booklist.xml
booklist.xsl booklist.htm result=HTML
```


XT引擎的速度依赖于机器的处理能力，因为 Java 是一种解释语言。

3. 使用 XSLT

现在看一下 XSLT 是如何工作的，我们将直接跳到一个简单的例子上。让我们用一个用 XML 标记的书的细节开始，将它转换成 XHTML，用于在浏览器上显示。

XHTML 是 HTML 4.0 的扩展版本，被设计用在 XML 的应用上。更详细的信息，参考最新的 W3C 推荐标准 <http://www.w3.org/TR/xhtml1>。

这是一些用 XML 标记的书的目录信息，使用的是第 3 章的 DTD：

程序清单 9-5

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!--===== The Wrox Press Book Catalog Application =====>
<Catalog>
  <Book>
    <Title>Designing Distributed Applications</Title>
    <Authors>
      <Author>Stephen Mohr</Author>
    </Authors>
    <Publisher>Wrox Press, Ltd.</Publisher>
    <PubDate>May 1999</PubDate>
    <Abstract>5 principles that will make your web applications more flexible
      and live longer</Abstract>
    <Pages>460</Pages>
    <ISBN>1-861002-27-0</ISBN>
    <RecSubjCategories>
      <Category>Internet</Category>
      <Category>Programming</Category>
      <Category>XML</Category>
    </RecSubjCategories>
  </Book>
  <Book>
    <Title>Professional Java XML</Title>
    <Authors>
      <Author>Alexander Nakhimovsky</Author>
      <Author>Tom Myers</Author>
    </Authors>
    <Publisher>Wrox Press, Ltd.</Publisher>
    <PubDate>August 1999</PubDate>
    <Abstract>Learn to utilize the powerful combination of Java and
      XML</Abstract>
    <Pages>600</Pages>
    <ISBN>1-861002-85-8</ISBN>
    <RecSubjCategories>
      <Category>Java</Category>
      <Category>Programming</Category>
      <Category>XML</Category>
    </RecSubjCategories>
  </Book>
</Catalog>
```

让我们看一下简单的 XSLT 样式表，它将用来转换源文档到要求的结果文档。结果文档将是一个 XHTML 文档，用于显示分类中书的题目：

程序清单 9-6

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

```

<xsl:output method="html"/>

<xsl:template match="/">
  <html>
    <head>
      <title>The book catalog</title>
    </head>
    <body>
      <xsl:apply-templates select="//Book" />
    </body>
  </html>
</xsl:template>

<xsl:template match="Book">
  <DIV style="margin-left: 40pt;
margin-bottom: 15pt;
text-align: left;
line-height: 12pt;
text-indent: 0pt;" >
    <xsl:apply-templates select="Title" />
  </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>

```

最后，结果XHTML文档将看上去如程序清单 9-7所示。

根据XSLT推荐标准版本1，本例只能由XSLT引擎来执行。SAXON和XT属于该范围。IE 5.0不支持——因为它不支持XPath和一些XSLT命令。然而，在本例中可将以上文件改为IE 5.0支持的，将命名空间由http://www.w3.org/1999/XSL/Transform改为http://www.w3.org/TR/WD-xsl，并且删除<xsl:output method="html"/>语句。但要记住，即使做了上述修改，也不能适用于本章所有的例子，所以最好在出现最新的支持IE的解析器之后，再使用XT。

程序清单 9-7

```

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
  <title>The book catalog</title>
</head>
<body>
  <DIV style="margin-left: 40pt;
margin-bottom: 15pt;
text-align: left;
line-height: 12pt;
text-indent: 0pt;">
    <DIV style="margin-left: 40pt;

```

```
        font-family: Arial;  
        font-weight: 700;  
        font-size: 14pt;">  
    <SPAN>Designing Distributed Applications</SPAN>  
  </DIV>  
</DIV>  
<DIV style="margin-left: 40pt;  
  margin-bottom: 15pt;  
  text-align: left;  
  line-height: 12pt;  
  text-indent: 0pt;">  
  <DIV style="margin-left: 40pt;  
    font-family: Arial;  
    font-weight: 700;  
    font-size: 14pt;">  
    <SPAN>Professional Java XML</SPAN>  
  </DIV>  
</DIV>  
</body>  
</html>
```

简单的结果看上去如图 9-4 所示。

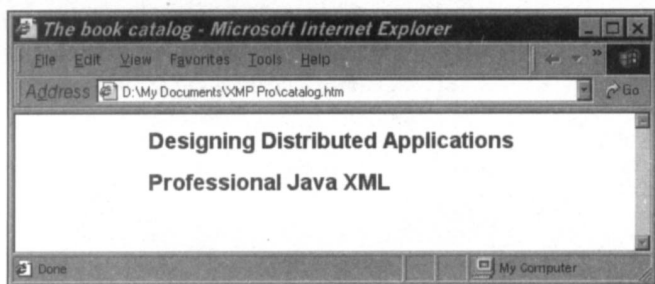


图 9-4

(1) 得到转换的帮助

为了更好的理解假定的例子，想象一下，你现在就是一个 XSLT 引擎，以它们的眼光来看这个世界（继续，没人看你）。首先，作为一个 XSLT 引擎，记住，你需要的是文档命令而不是文本本身。毕竟，你可以只处理命令而不是文本。所以，某人就不得不转换文本为要求的抽象树命令，树林。作为一个 XSLT 引擎，你可能有以下两个好朋友中的一个：

- 一个 DOM 接口的解析器。
- 一个只给出你每个元素事件的解析器。

如果你使用的是一个 DOM 接口的解析器的服务，这就意味着解析器封装了整个树，通过 DOM 接口来处理在树林上的任何对象。

如果你使用一个给出每个元素事件的解析器，将自己管理树林并且按自己的想法保存文档命令。这就是使用 SAX 接口的 Java 方法。

所以，可爱的 XSLT 引擎，你可以选择是通过 DOM 接口的解析器来得到帮助对树林进行管理，还是通过自己管理它。

内部命令可以用不同的方法实现；然而，尽管你使用关联数组或链表的链表，模型化的命令是一棵树。DOM是W3C的推荐，指出了如何定义命令的接口。

如果你记起前面的示意图，将有三棵树。一个包含了源文档的表示，一个是结果树命令的表示，但是第三棵树是什么？不是一个XSLT文档或是一个XML文档吗？是的。你已经开始进入角色了。如果是XSLT文档，转换成一种内部的树状命令并不表示将文本转换成一种分级命令。XSLT内部命令可能是一些其他内容，为XSLT处理优化过的东西。

所以，原始的XML文档首先被解析，然后被转换成一种抽象的树状命令，一种分级命令的内部表示。DOM是这个内部命令的接口。XSLT文档也被解析，转换成一种内部命令。可能是一种抽象的树状命令，但是也可能是另一种命令，被优化过用于模板的处理和模式匹配。

Catalog.xml文件：

程序清单 9-8

```
<Catalog>
<Book>
  <Title>IE5 XML Programmer's Reference</Title>
  <Authors>
    <Author>Alex Homer</Author>
  </Authors>
  <Publisher>Wrox Press, Ltd.</Publisher>
  <PubDate>August 1999</PubDate>
  <Abstract>Reference of XML capabilities in IE5</Abstract>
  <Pages>480</Pages>
  <ISBN>1-861001-57-6</ISBN>
  <RecSubjCategories>
    <Category>Internet</Category>
    <Category>Web Publishing</Category>
    <Category>XML</Category>
  </RecSubjCategories>
</Book>
</Catalog>
```

可能会被表示成为如下的抽象树，用在XSL处理器中（参见图9-5）。

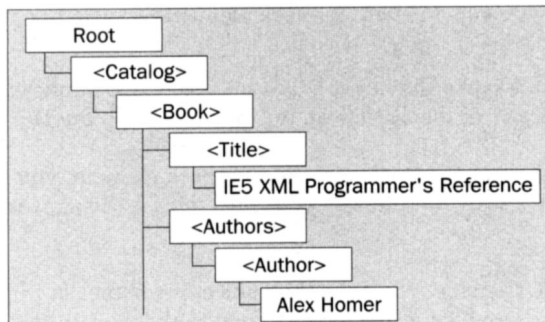


图 9-5

(2) 样式表如何转换文档

如我们所讲到的，XSL是一个XML的应用，所以样式表（如果你愿意也可以称之为转换表）

是一个真正的 XML 文档。因为它是一个 XML 文档，它可以使用 XML 的声明开始，指示了 XML 解析器这个文档编码的 XML 版本。

在我们的样式表中的根元素是 `<xsl:stylesheet>` 元素：

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

`<xsl:stylesheet>` 元素的第一个属性是 XSLT 的版本。第二个属性是 `xmlns:xsl`，用来控制 XSL 转换推荐的命名空间。

你可能还记得第 7 章，关于命名空间和模式，这个声明了 XSLT 的命名空间。你可以看到，命名空间的前缀为 `xsl`，所以根元素实际是 `<stylesheet>`，但是它被 `xsl:` 进行了限制，作为它的名字前缀。在已经声明了命名空间，任何一个以 `xsl:` 为前缀的元素都被看作 XSL 的词汇表。

`<stylesheet>` 元素包含三个模板，每一个都嵌在 `<template>` 元素中，在样式表中确切的是 `<xsl:template>`，是由于我们包括了命名空间。你会注意到，在 `<template>` 元素中有一个叫 `match` 的属性。这个属性的值是一个模式，按照 XPath 表达式的形式，用来匹配应该应用模板的树的节点。

首要的任务是告诉 XSLT 引擎所期望的输出。在这个例子中，希望是一个 HTML 的结果，指出使用：

```
<xsl:output method="html"/>
```

像一个处理器一样，了解期待的输出格式，你将从源文档树中的根节点开始。然后在样式表中匹配根节点，找到相对应的模板。注意，根节点是文档的节点，不是第一个元素。在这个例子中，根节点不是 `<Catalog>` 元素，而是 XML 文档本身。那么，我们有一个匹配文档根的模板吗？回答是肯定的。如果你还记得在上一章 XPath 部分，一个文档的根也可以表示为一个 `" / "` 符号。这一点我们看到在第一个模板中很明确：

程序清单 9-9

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

所以，你已经找到了匹配源文档中根元素的模板了。现在要做什么？为了更好地表示在 XSLT 处理器的头部发生了什么，想象一个游标航行于初始的 XML 文档节点树；它的位置是当前节点，现在当前节点是根元素。

第 1 步：你已经定位了当前游标在根节点处，并且在 XSLT 命令中发现了一个相匹配的模板。模板有一个 `" / "` 模式。所以，输出下面的结果。记住，我们工作于一个抽象的树状命令中，表示出来就是图 9-6 中左边的部分：

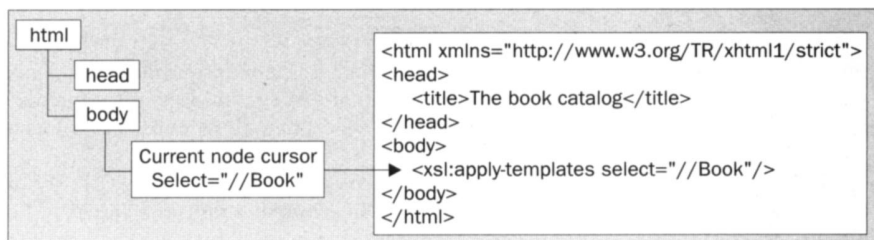


图 9-6

在第一个模板的中间，嵌套在 `<Body>` 元素间的是一个 `<xsl:apply-templates/>` 命令。这就是我们将书写内容的页面。它有一个叫做 `select` 的属性，它的值是一个 XPath 表达式。这个命令的意思是：

“从 XPath 的查询 `//Book`”中，得到一个节点列表。然后，对于这个节点列表中的每个节点，试着匹配一个模板。如果一个匹配找到了，则应用模板。”

但是 `//Book` 是什么意思？它的意思就是“选择从根节点派生出的 `<Book>` 元素”。

第2步：更深一步地进入我们的 XSL 文件中，找到一个匹配 `<Book>` 元素（`<xsl:template match="Book">`）的模板，所以下一步我们就应用这个模板给节点列表中的第一个 `<Book>` 元素。我们的当前游标现在在第一个 `<Book>` 节点上。

第3步：我们然后插入与 `<Book>` 匹配的模板内容到 `<apply-templates select="//Book"/>` 命令的位置处。首先我们加入缺省的 CSS 样式属性给书的数据——任何更深一层的元素不包含特定 CSS 样式属性将继承这些属性。接下来，我们找到另一个 `<apply-templates/>`，这次使用了一个 `"Title"` 的 `select` 属性。根据第一步的规则，我们知道这个要求命令一个 `<Title>` 节点的集合。然而，这一次当前节点是第一个 `<Book>` 节点，并且我们的 XPath 表达式指示了，新节点将只包含当前子节点中的 `<Title>` 节点。这就意味着我们的节点集将由第一个 `<Book>` 节点的 `<Title>` 子节点组成。

从这里我们开始看到了所使用的 XPath 表达式的多功能性。如果打算替换成 `<xsl:apply-templates select="//Title"/>` 用于存在的节点，节点集中将包含所有从根节点（`<Book>` 节点的根节点）继承下来的 `<Title>`——这将意味在树林中的所有 `<Title>` 节点（参见图 9-7）。

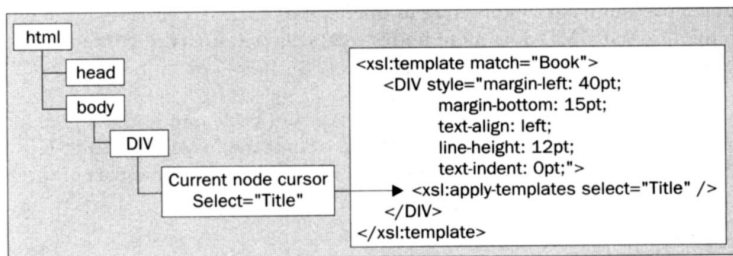


图 9-7

第4步：接着我们试图匹配一个模板针对于我们的节点集中 `<Title>` 节点。再一次我们找到了

一个匹配：`<xsl:template match=" Title ">`模板。然后在`<xsl:apply-templates select=" Title " />`命令的位置插入这个模板的内容。

第5步：`<Title>`匹配模板的内容由一些 CSS 样式属性和一个`<xsl:value-of select="."/>`元素组成。这个命令取出通过 `select` 属性指定的节点的值。在这个例子中，XPath 是“.”，意思是`<Title>`节点本身，所以我们输出`<Title>`节点的内容（参见图 9-8）。

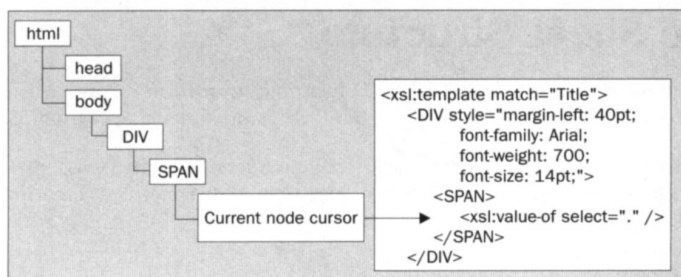


图 9-8

第6步：现在对于在`<Book>`中创建的节点集，对所有的节点使用了与之相匹配的模板。所以移动到最先创建的节点集中的下一个节点——是第二个`<Book>`节点。采用与第一个`<Book>`节点同样的方法，所以我们重复执行第四和第五步。循环进行处理，直到所有的 `<Book>` 元素都被处理完毕。

在这个处理过程中，我们不仅仅将一个 XML 文档从某种文档格式转换到另一种，而且在处理中也执行一些编辑——只有`<Book>`和`<Title>`元素被转换了。而且，转换不只是一到一的转换。对于初始文件中的每个元素，在结果文档中都可能有一个以上的元素。

我们应该在这里注意一些重要的事情。与 `<Title>`相匹配的模板没有在结果树中插入元素，它插入的是文本节点。如前所说，一个树林是一种内部分级命令。当 XML 文档转换成这种分级命令，我们转换文本文档为树形模型。在这棵树中，孩子元素也是树的子节点。数据内容也是一个子节点。例如，`<Book>`元素包含一个`<Title>`元素，这个`<Title>`元素不包含元素，但是包含数据内容，它变成一个子节点。

那么，当假装是一个 XSL 引擎时我们学到了什么？

- 首先是我们创建了一个树林，它是文档的一种内部树形表示。这个树林总是有一个根元素。根元素表示 XML 文档——它不是文档的顶层元素。然后，在根元素的下面是节点层次。每个节点都被打印出来。例如，一个节点可以表示一个 DTD、模式或一条处理指令。如果一个元素有属性，那么每一个元素也有一个属性节点的集合。如果一个节点有数据内容，那么数据内容被加到元素节点中。因此，一个元素节点可能有一个属性节点的集合和一个数据内容节点作为孩子。
- 第二，我们为 XSLT 文档创建了一个命令。这个可能也是一个树林，但是它可能是任何一种命令，而且为了模板处理和模式匹配作了优化。
- 然后，每一次我们遇到一个`<xsl:apply-templates>`元素，形成一个节点列表，使用这个列表继续处理。如果`<xsl:apply-templates>`元素包含一个 `select` 属性，从指定的 XPath 查询中获得

节点列表，另外这个节点列表将包括所有的子节点。

- 每一次遇到一个<xsl:value-of>命令，我们从源树中提取出一个值，根据在 XPath表达式中的select属性。
- 转换不限于一对一的翻译，它也允许其他新的信息内容，一对多的元素翻译，元素的增加，元素的删除。

9.3 XSLT 样式表命令

好了，我们已经看过了第一个例子，它是通过 XSLT文档来转换一个XML文档变成一种新的命令。现在，应该全面地看一下 XSLT的命令是怎样的。

从第一个例子中我们已经学了很多；不仅看到对于一个需要转换的文档，XSLT处理器是如何贯穿全部的——我们看到生成其他元素这一点非常有用——也使用了XSLT元素的四个关键字：

- <xsl:stylesheet>
- <xsl:template>
- <xsl:apply-templates>
- <xsl:value-of>

在开始的第一个例子中，我们使用了 XML的声明，因为样式表是一个 XML文档。记住 xsl: 作为限定前缀应用于属于 XSLT命名空间部分的元素，<stylesheet>元素是一个包含其他样式表元素的文档元素，这就是命名空间被声明的地方。在这个元素中我们有三个<template>元素，它被用于指定一个元素或其他节点中，对被用于指明 match的属性应如何转换。这个可以被看作大部分转换工作的主要的构建块。<xsl:apply-templates>元素被用于告诉处理器去处理当前元素的所有子元素，如果没有指出 select属性时。否则，只有匹配符合选择标准的节点被处理。最后，xsl:value-of元素被用于输出一个元素内容。

这个说明在 XSLT规范中定义两种元素。除了根元素之外，是模板和指令。很明显，<xsl:template>是一个模板，因为在抽象树状命令中，它将显示在根元素的下面。然而 <xsl:apply-templates>和<xsl:value-of>是指令，作为<template>元素的孩子显示出来。记住一个 XSLT文档是一个XML文档，因为如此就可以被转化为一个树状命令。

图9-9显示了顶级元素，<xsl:stylesheet>元素的孩子：

这个说明了为什么 <xsl:stylesheet>元素总是一个 XSLT样式表的根元素。在这个元素的下面，我们能够有很多的顶级元素。所以，XSLT处理器工作使用的抽象命令就是像这样的，有一个根和很多的顶级元素。让我们浏览其中一些。

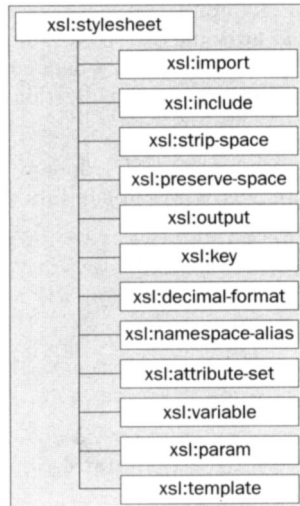


图 9-9

9.3.1 创建模板

在任何一个 XSLT样式表中的关键命令是 <template>元素，它使用一个 match属性，它的值是

一个模式——或XPath表达式——表示应该使用哪一个节点的模板。更明确地说，任何 XPath表达式返回一个节点列表，与 match属性值相匹配的节点候选。然而，容易记忆的是模式是一个 XPath表达式，它指出对于一个节点应该应用哪一个模板。在模板里面我们可以包含元素和元素的内容。

9.3.2 处理空白

为了帮助我们使用空白，XSL指供了两种命令，被用作顶级元素：

- xsl:strip-space 用来除掉被选中的节点中的空白节点。
- xsl:preserve-space 保持内容中的任何空白。

1. xsl:strip-space。

当元素的名字包含一个 elements属性时，<xsl:strip-space>元素从树中删除由空白组成的文本节点。例如，下面的 <xsl:strip-space>元素将删除任何 <BOOKLIST>或<ITEM>元素中的仅由空白组成的文本节点。

```
<xsl:strip-space elements="BOOKLIST ITEM" />
```

所以，上面的元素告诉 XSL引擎，如果元素<BOOKLIST>和<ITEM>由空白组成，它们的文本节点应该从树（然而元素节点被保留）中被删除。

2. xsl:preserve-space

同样，如果我们想保留某种元素内容的空白，那么包含 <xsl:preserve-space>元素。再一次指出，elements属性用于指示所有所想保留空白的元素列表。下面的例子，对于 <CATALOG>和<PRICE>元素空白被保留。

```
<xsl:preserve-space elements="CATALOG PRICE"/>
```

9.3.3 输出格式

<xsl:output />元素能够被用于指示一个结果树（尽管并不要求一个XSL处理器实现这个功能）的输出格式。

再一次强调，这是一个顶级元素，在正常情况下，应立即跟在 <xsl:stylesheet>元素后面。这不是一个强制性的元素，在很多情况下，XSL引擎可能对于 HTML在某些条件下有缺省设置：

- 结果树的根节点应该有一个子节点。
- 根的第一个子节点应该是一个html元素。
- 任何在第一个子节点前的节点应该只包含空白字符。

也可以将结果树设成不同的格式，如 xml、html或text。<xsl:output>元素的一个有趣的属性是encoding属性。这个在后面可以允许我们将某种编码转换成另一种 XSLT引擎所支持的一种目标编码。例如，一个 ASCII编码的XML文档可以被转换成统一码编码文档。所以，为了转换一个XML文档到一个新的使用不同编码的XML文档，应该在 <xsl:stylesheet>元素后面立即包括 <xsl:output>元素：

程序清单 9-10

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" encoding="UTF-16"/>
```

9.3.4 合并样式表

一个重用代码的方便方法是创建模块。然后，这些模块可以被用在其他的模块中——XSLT可以包含或导入外部的样式表。有两种命令可以达到这个效果：

- <xsl:include>元素
- <xsl:import>元素

1. xsl:include

<xsl:include>元素只是简单地允许我们包含一个外部的样式表，在定义 <xsl:include>元素的位置处。通过URI所指出的XSLT文档首先被解析，然后被包括文档的<stylesheet>元素的子元素替换需要替换文档的<xsl:include>元素。有必要将<xsl:include>命令定位成<stylesheet>元素的子元素：

程序清单 9-11

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

2. xsl:import

<xsl:import>与<xsl:include>有很大不同——<xsl:include>只是表示执行一个文件包含，但是<xsl:import>修改文档树。实际上，<xsl:import>命令修改模板的顺序和优先处理。

最重要的是，这个元素应该放在其他顶级元素的前面——它应该是<xsl:stylesheet>元素的第一个子元素。

首先，所有被输入的样式表被当做文本被包含。一但它们都被收集完毕，将被用于生成一棵输入树。所以，每一个被输入的样式表都被包含在宿主样式表的输入树中。这就可以让被输入的样式表自己再输入其他的样式表。

例如，booklist.xml样式表可能输入一个像这样的样式表：

程序清单 9-12

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

现在，让newBooks.xml输入其他的样式表：

程序清单 9-13

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
```



```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:import href="recentXML.xsl">
```

然后，合成的输入树看上去应该如图 9-10所示。

这就形成了一种命令，它指出一个样式表可以优先于另一个，Booklist.xsl的优先权要高于其他两个。当模板与元素相匹配时，Booklist.xsl首先被处理，然后是 NewBooks.xsl，最后是 recent.xsl。<xsl:import>命令直接影响了样式表的处理。XSL文档树通过这个元素被修改，并且样式表被组装进了单一单元——输入树。

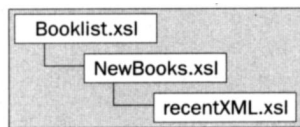


图 9-10

9.3.5 嵌入样式表

一个样式表不必是一个分离的文档。它能够被嵌入到另一个 XML文档中。例如，一个动态命令的XML文档可能在转换给用户机构之前包括它的样式表，一个 XSL样式表被嵌入到XML文档中：

程序清单 9-14

```
<?xml version="1.0"?>
<?xsl:stylesheet href="#BooklistStyle" type="text/xsl" media="screen"?>
<xsl:stylesheet version="1.0"
    id="BooklistStyle"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    ...
</stylesheet>
<BOOKLIST>
    ...
</BOOKLIST>
```

这里，样式表被包含在XML文档中。通过一个XML（用#号所指）id指出，所以XSL处理器知道样式表是在文档中的特殊片段，通过一个元素中的id属性定义的。XSL处理器将然后从文档中提取出样式表片段，分析它，并且为XSL处理器命令内部命令。XML文档本身被解析，但是结果的文档树并不包含样式表。所以，从一个单一的XML文档，用户机构得到两个文档命令：

- 一个用于XML文档的树林，不包括<xsl:stylesheet>元素（它包括了整个样式表）。这个命令可以用DOM处理。
- 一个XSL文档，只包括<xsl:stylesheet>元素和它的内容。这个命令可能是也可能不是一个树林，可能也可能不被DOM处理。

9.4 使用XSLT的例子

如你所见，XSLT是一个有力的工具，通过一种命令或文档类型，来转换XML文档为一个新的文档——如将一上XML文档翻译成XHTML。我们已经看过前面教给XSLT处理器如何工作的例子，看过了一个引用部分，解释了对于在XSLT中最常使用的元素，这一章的第二部分将看一些例子，关于在不同的环境下使用XSLT。这些将包括：

- 命令的转换，从一个XML词汇表到另一个。
- 使用循环重复处理元素——xsl:for-each。
- 对需要处理的元素排序。
- 使用xsl:if和xsl:choose进行条件处理。
- 创建动态文档。

9.4.1 命令的转换

让我们看一个例子，是将一个XML文件改成另一种XML命令，不单是XHTML。比方说，我们需要重新排序一个同事给你的XML文档的元素，像下面：

程序清单 9-15

```
<?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>
```

情况良好，但是他也给你增加了一些调料，给你提出了一些需求：

- 文档应该输出到能够通过CSS样式表来表现XML的浏览器上。
- 每一项（指书）必须作为一个块显示。
- 每个标题要首先显示（在块中）。
- 分类和代号应该显示在同一行上，但是分类要显示在前。
- 每块的最后一行应该包含发行日期，接着是价格。

并且，如果不够，他建议<CATEGORY>内容应该用“Category:”字符串显示，并且应该用圆括号括起来。好，发行日期和价格也应该用一个“-”分隔开。并且在圣代冰淇淋上加点樱桃，你只能使用CSS1样式表。这时候你可能会想这不是你过的日子。但是XSLT正等着帮助你。

那么，你需要做的第一件事情是，为了能够用CSS来格式化文档，要将存在的文档命令转化，看上去像这样：

程序清单 9-16

```

<?xml version="1.0"?>
<?xml-stylesheet
  type="text/css"
  href="catalog.css"
  media="screen"?>
<BOOKLIST>
  <ITEM>
    <TITLE>Instant JavaScript</TITLE>
    <DESCRIPTION>
      <CATEGORY>Category: Scripting</CATEGORY>
      <CODE>(16-048)</CODE>
    </DESCRIPTION>
    <LISTING>
      <RELEASE_DATE>Release date: 1998-04-21</RELEASE_DATE>
      <PRICE>Price: $49.34</PRICE>
    </LISTING>
  </ITEM>
  <ITEM>
    <TITLE>Instant Active Server Pages</TITLE>
    <DESCRIPTION>
      <CATEGORY>Category: ASP</CATEGORY>
      <CODE>(16-105)</CODE>
    </DESCRIPTION>
    <LISTING>
      <RELEASE_DATE>release date: 1998-05-10</RELEASE_DATE>
      <PRICE>Price: $23.45</PRICE>
    </LISTING>
  </ITEM>
  <ITEM>
    <TITLE>Instant HTML</TITLE>
    <DESCRIPTION>
      <CATEGORY>Category: HTML</CATEGORY>
      <CODE>(16-041)</CODE>
    </DESCRIPTION>
    <LISTING>
      <RELEASE_DATE>release date: 1998-03-07</RELEASE_DATE>
      <PRICE>Price: $34.23</PRICE>
    </LISTING>
  </ITEM>
</BOOKLIST>

```

为了进行转换，我们将使用下面的样式表，稍后将学到很多的细节。它只包括了两个模板。这个例子可以用XT和SAXON来执行。为了在Microsoft的IE上运行下面的模板，你需要一个比在IE5中提供的MSXML组件更新的版本。

程序清单 9-17

```

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

  <xsl:output method="xml"/>

  <xsl:template match="/">
    <xsl:processing-instruction name="xml-stylesheet"
      href="catalog.css" type="text/css" media="screen"
    </xsl:processing-instruction>

```

```

<BOOKLIST>
  <xsl:apply-templates/>
</BOOKLIST>
</xsl:template>

<xsl:template match="ITEM">
  <ITEM>
    <TITLE>
      <xsl:apply-templates select="TITLE/text()" />
    </TITLE>
    <DESCRIPTION>
      <CATEGORY>
        Category:
        <xsl:apply-templates select="CATEGORY/text()" />
      </CATEGORY>
      <CODE>
        (<xsl:apply-templates select="CODE/text()" />)
      </CODE>
    </DESCRIPTION>
    <LISTING>
      <RELEASE_DATE >
        Release date:
        <xsl:apply-templates select="RELEASE_DATE/text()" />
      </RELEASE_DATE>
      <PRICE>
        - Price:
        <xsl:apply-templates select="PRICE/text()" />
      </PRICE>
    </LISTING>
  </ITEM>
</xsl:template>

</xsl:stylesheet>

```

在以前的例子中我们看到，源文档首先被转换成一个树林（一个内部分级命令）。在为了处理将XSLT文档被转换成一种同部命令之后，接着元素进行匹配（也可能不，看例子而定）模板。

第一个节点同文档的根相匹配：

程序清单 9-18

```

<xsl:template match="/">
  <xsl:processing-instruction name="xml-stylesheet">
    href="catalog.css" type="text/css" media="screen"
  </xsl:processing-instruction>
  <BOOKLIST>
    <xsl:apply-templates/>
  </BOOKLIST>
</xsl:template>

```

XML文档可以与样式表相关联，使用一个<?xml-stylesheet ...?>处理指令。我们想让文档与一个CSS样式表相关联，所以不得不写在模板中以便让命令文档中包含处理指令。

为了在结果树中创建处理指令，我们使用一个特别的XSL命令，<xsl:processing-instruction>元素。name属性提供了处理指令的名字，数据内容为其他所有的属性。所以下面的XSL元素：

```
<xsl:processing-instruction name="xml-stylesheet">
  href="catalog.css" type="text/css" media="screen"
</xsl:processing-instruction>
```

被转化为命令树为：

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

包含在模板中的其他<BOOKLIST>元素将被插入到结果树中。现在熟悉的<apply-templates>命令指示XSLT处理器应该处理所有的子元素，由于没有一个选择标准——子元素将被处理成当前节点的子节点——然后，这些节点将同模板进行匹配。如果一个包含数据的孩子没有一个匹配的模板，它被插入到结果树中。否则，如果同一个模板相匹配，模板被处理，并且它的内容将包含在结果树中。

你可能已经注意到，没有模板匹配<BOOKLIST>元素。XSLT引擎有一个隐含的模板用于匹配任何一个没有明确指定模板的元素。这个模板允许递归处理，用于处理与模式相匹配的没有明确定义模板的元素。这个隐含的模板被定义为：

```
<xsl:template match="*|/">
  <xsl:apply-templates/>
</xsl:template>
```

所以<BOOKLIST>元素，它没有一个明确定义的模板规则，与隐含的模板相匹配。这个隐含模板也可以叫做缺省模板。

<BOOKLIST>元素包含<ITEM>元素，对于它存在一个模板。实际上，这是一个我们想重新组织的元素。重新组织<ITEM>元素相当容易，我们只是包含了按照需要排列的元素。如果不得不加入新的元素，只需简单地在模板中包含它们。

我们使用<xsl:apply-templates>命令与用在第一个模板的颇为不同。在前面我们使用 select属性来指示XSLT引擎，只有与选择标准匹配的元素使用明确的模板，否则使用缺省的模板。

下面的表达式在输出树中包含了初始XML文档的<TITLE>的数据内容。

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

select属性的XPath表达式用来指示XSLT引擎，<TITLE>元素的文本节点的内容将插在<xsl:apply-templates>命令所在的相同地方（参见图9-11）。

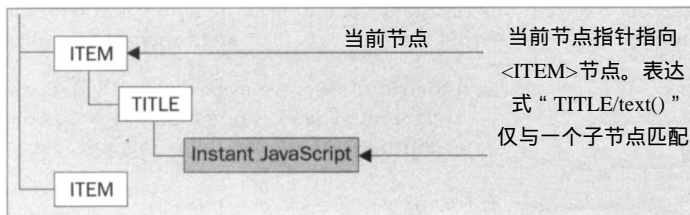


图 9-11

注意这里是与<ITEM>元素相匹配的模板，因为<TITLE>元素的节点是它的孩子，正确的表达式应该是“TITLE/text()”。如果我们使用“//TITLE/text()”来代替，所有<TITLE>元素的内容将会如下被插入到结果树中：

```
<TITLE>Instant JavaScriptInstant Active Server PagesInstant HTML</TITLE>
```


这是因为加入“//TITLE”意味着“处理所有从根节点继承的节点（对于元素类型），名字为<TITLE>。”注意，XPath表达式“./TITLE/text()”意思是“处理所有从当前选中节点继承的名字<TITLE>的节点”。在//前加.则意思就完全不同了（参见图9-12）。

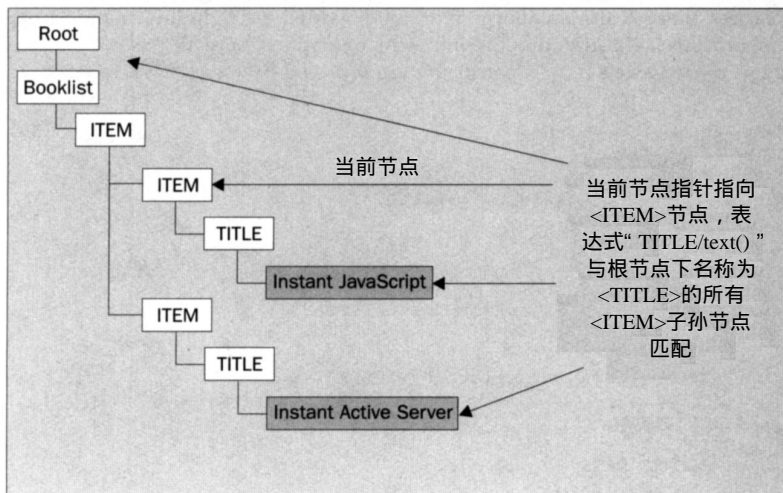


图 9-12

所以，包含在模板中的匹配<ITEM>节点的所有<xsl:apply-template select ...>命令都是相对于当前选中节点的。在我们的例子中，这就是<ITEM>节点。当前节点的游标通过模板的匹配属性移动到<ITEM>元素：

程序清单 9-19

```
<xsl:template match="ITEM">
  <ITEM>
    <TITLE>
      <xsl:apply-templates select="TITLE/text()" />
    </TITLE>
    <DESCRIPTION>
      <CATEGORY>
        Category:
        <xsl:apply-templates select="CATEGORY/text()" />
      </CATEGORY>
      <CODE>
        (<xsl:apply-templates select="CODE/text()" />)
      </CODE>
    </DESCRIPTION>
    <LISTING>
      <RELEASE_DATE >
        Release date:
        <xsl:apply-templates select="RELEASE_DATE/text()" />
      </RELEASE_DATE>
      <PRICE>
        - Price:
        <xsl:apply-templates select="PRICE/text()" />
      </PRICE>
    </LISTING>
  </ITEM>
</template>
```

```

    </LISTING>
  </ITEM>
</xsl:template>

```

像前面我们看到的，有一个可替换的方法来从初始 XML 文档中提取正确的信息。如下面举例所示，所有 `<xsl:apply-templates.../>` 命令的地方被 `<xsl:value-of .../>` 命令所替换：

程序清单 9-20

```

<xsl:template match="ITEM">
  <ITEM>
    <TITLE>
      <xsl:value-of select="."/>
```

我们看到了有两种方法在正确的地方插入正确的元素：

- 使用 `<xsl:apply-templates>` 命令。
- 使用 `<xsl:value-of>` 命令。

我推荐使用第二个命令——`<xsl:value-of>`——它明确地告诉我们是被选节点的值插入到输出树中去。你可能已经注意到，我们在选择表达式中没有包括 “ `text()` ” 指令，因为元素的值就是它的数据内容。

新的数据内容也可被加入到输出内容中。例如，我们想在结果数据的开始处包含 `Category:` 表达式，这样我们可以得到像这样的代码：

```
<CATEGORY>Category: Scripting</CATEGORY>
```

再一次，我们使用 `<xsl:value-of.../>` 命令，它被 `<CATEGORY>` 元素的内容所替换，但是我们也加入了文本 `Category`（参见图 9-13）。

使用 XSLT 转换文档格式在大多数场合是非常重要的。为了以适当的方式显示 XML 文档，我们使用 XSLT 转换成另外一种格式。当在结果文档里使用相同的标签时，我们插入内容，从模板中使用 `<value-of>` 或者 `<apply-templates>` 命令获得源文档数据元素，自己写进元素中去。我们很容易地创建新的标签，文档能被转换成一个完全新的词汇表。我们能够使用以前的例子（在第 2 章

曾经介绍过)转换这个文档成词汇表。

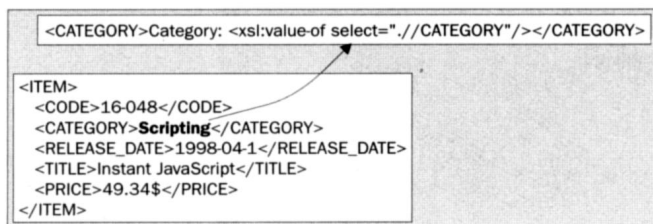


图 9-13

这种技术能完美地将文档转换成表示语言,例如HTML, WML(Wireless Markup Languages)。当我们以不同的XML命令交换信息时,这对于转换不同的内容是非常有用的。例如,当我们转换金融数据的时候,一个公司使用FPML,另一个公司使用FinXML,我们能在两者之间转换。

9.4.2 循环

循环是我们在过程语言中经常使用的命令。XSLT也支持循环命令,使用<xsl:for-each.../>方式。它的内容被重复执行直到最初XML文档中的元素符合指定的属性。举一个例子,我们能够使用for-each命令把XML书列表文档转换成表项在列表文档中列出的XHTML文档。下面是一个XSLT程序完整的清单,你将看到使用上的一些有趣的事情:

程序清单 9-21

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

<html xmlns="http://www.w3.org/TR/xhtml1/strict"
      xsl:version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <head>
    <title>The book catalog listed in a table</title>
  </head>
  <body>
    <table border="1" cellspacing="0" cellpadding="5">
      <tbody>
        <xsl:for-each select="BOOKLIST/ITEM">
          <tr>
            <th align="left"><xsl:value-of select="\"/>
```

你可能注意到的第一件事情是它使用了与以往例子不同的格式。该程序仅仅包含了简单的

模板固定匹配根元素。事实上，在这个二选一的格式中包括 `<xsl:template>` 这样的命令是没有必要的。这个固定的模板是 `<xsl:template match=“ / ”>`。

我们创建的表每个 `<ITEM>` 包含一行。在这个例子中，我们不能使用模板匹配机制，我们使用 `for-each` 命令循环始终贯穿程序，使用 `value-of` 命令获取元素内容：

程序清单 9-22

```
<xsl:for-each select="BOOKLIST/ITEM">
  <tr>
    <th align="left"><xsl:value-of select="./TITLE"/></th>
    <td><xsl:value-of select="./CATEGORY" /></td>
    <td><xsl:value-of select="./RELEASE_DATE" /></td>
    <td><xsl:value-of select="./PRICE" /></td>
  </tr>
</xsl:for-each>
```

我们告诉处理器，“对根 `<BOOKLIST>` 元素中的每一个 `<ITEM>` 元素，写 `<TITLE>`、`<CATEGORY>`、`<RELEASE_DATE>` 和 `<PRICE>` 元素的内容进入表中”。循环命令的标准是从根开始的一个 XPath 表达式；因此，我们不得不显式地包含在文档树的分支中每一个 `<ITEM>` 的元素。当不再有元素满足条件时循环才能结束。

为了包含表单元格中的值，我们使用 `<xsl:value-of.../>` 元素。与你所设想的一样，它获取了匹配 `select` 属性的 XPath 表达式的节点的数据内容。

结果输出显示在图 9-14 中。

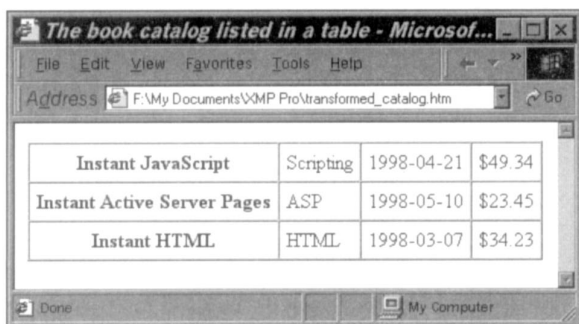


图 9-14

9.4.3 排序

在我们的数据列表中移动数据，加入新的数据，连接合成文档到 CSS 样式表，之后，让我们加入排序。目标是先按照种类排序，再按照题目排序。

用到这个方法的 XSL 命令是 `<xsl:sort>`。为了让 XSLT 引擎排序，我们包括 `select` 属性设置到 XPath 值。举个例子，按题目排序，我们使用下面的命令：

```
<xsl:sort select="./TITLE"/>
```

问题是：我们怎么找到包含这种用法的例子。它是仅仅被使用在 `<xsl:apply-templates.../>` 或

者<xsl:for-each...>元素中的一个指令元素，下面这个示例是对“循环”一节中示例的修改：

程序清单 9-23

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

<html xmlns="http://www.w3.org/TR/xhtml1/strict"
      xsl:version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <head>
    <title>The book catalog listed in a table</title>
  </head>
  <body>
    <table border="1" cellspacing="0" cellpadding="5">
      <tbody>
        <xsl:for-each select="BOOKLIST/ITEM">
          <xsl:sort select="."/CATEGORY"/>
          <xsl:sort select="."/TITLE"/>
          <tr>
            <th align="left"><xsl:value-of select="."/TITLE"/></th>
            <td><xsl:value-of select="."/CATEGORY" /></td>
            <td><xsl:value-of select="."/RELEASE_DATE" /></td>
            <td><xsl:value-of select="."/PRICE" /></td>
          </tr>
        </xsl:for-each>
      </tbody>
    </table>
  </body>
</html>
```

现在，<xsl:for-each...>元素在怎样处理被选择的节点上采用了新的指令。引擎在用模板匹配节点之前对它们进行排序。在上面的例子中，节点首先按照种类排序，然后按照题目排序，像图9-15显示的一样。

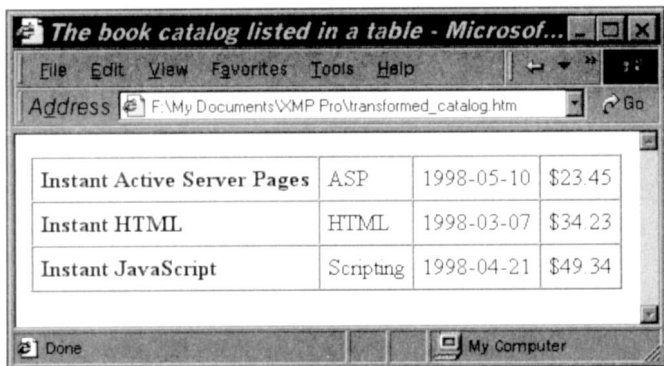


图 9-15

排序的次序按照 <xsl:sort.../>中指定的元素次序排列。举个例子，下面的命令将替代<TITLE>和<RELEASE_DATE>对节点排序。

```
<xsl:sort select="."/TITLE"/>
<xsl:sort select="."/RELEASE_DATE"/>
```


记住，<sort>命令在对节点进行处理前，先对节点进行重新排序（参见图 9-16）。

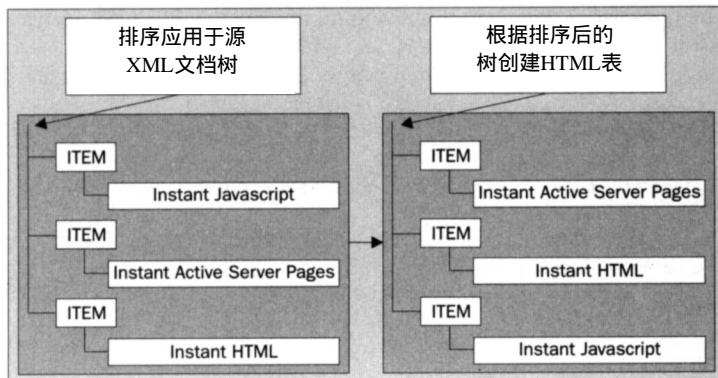


图 9-16

9.4.4 条件处理

在过程语言中经常使用的另外一种命令是：

- if命令，在XSLT中以<xsl:if>表示。
- if/elseif命令，在XSLT中以<xsl:choose>表示。

在这一点上，你可以想象声明语言 XSLT 包含许多程序命令。事实上，产生声明语言是让你不必显式地告诉 XSLT 引擎输出特定的内容。若对其指定内容或者模板，都将包含在输出结果中。尽管如此，XSLT 仍然有一定的过程化特性。

现在看来前一个例子，仅仅想要在结果树中包含 <ITEMS>的部分 Scripting 种类。为了这样做，我们需要一个过滤器，或一个 if 命令，指出引擎“如果碰到这种情况就这样做”。为了达到这种结果，我们在模板中包括 <xsl:if>指令元素，像下面的例子这样：

程序清单 9-24

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

<html xmlns="http://www.w3.org/TR/xhtml1/strict"
      xsl:version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <head>
    <title>The book catalog listed in a table</title>
  </head>
  <body>
    <table border="1" cellspacing="0" cellpadding="5">
      <tbody>
        <xsl:for-each select="BOOKLIST/ITEM">
          <xsl:if test="contains(CATEGORY/text(), 'Scripting')">
            <tr>
              <th align="left"><xsl:value-of select="//TITLE"/></th>
              <td><xsl:value-of select="//CATEGORY" /></td>
              <td><xsl:value-of select="//RELEASE_DATE" /></td>
              <td><xsl:value-of select="//PRICE" /></td>
            </tr>
          </xsl:if>
        </xsl:for-each>
      </tbody>
    </table>
  </body>
</html>
```

```

        </xsl:if>
      </xsl:for-each>
    </tbody>
  </table>
</body>
</html>

```

现在，在循环中我们包含了一个被满足的条件。如果条件是真，模板中包含在 `<xsl:if>` 中的元素被插入结果树中。否则，模板仅仅被跳过了。

在 `test` 属性中，我们比较字符串 `Scripting` 和 `<CATEGORY>` 元素的数据内容。实际上，我们使用 `contains()` 函数检验是否 `<CATEGORY>` 元素文本节点包含 `Scripting` 字符串。首先，如果字符串 `string1` 包含字符串 `string2`，`contains(string1,string2)` 函数返回布尔类型的 `TRUE`，就像我们在上一章看到的判断部分。`string1` 被使用 `<CATEGORY>` 数据内容的 `CATEGORY/text()` XPath 表达式包含。一个元素数据内容总是一个字符串。

你可能注意到在属性的值中，我们对字符串使用表达式 `'Scripting'` 替代了 `"Scripting"`，使用单引号而不是双引号。这是因为仅仅整个属性的值能被双引号括起来。因此，任何需要被括起来的表达式，像例子中的字符串那样，仅能使用单引号，像下面的表达式这样：

```
<xsl:if test="contains(CATEGORY/text(), 'Scripting')">
```

图9-17的HTML文档的模式显示了使用转换样式表。

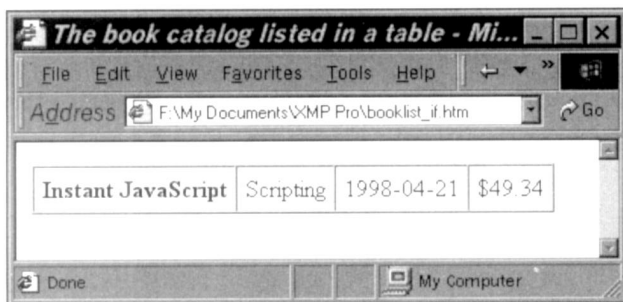


图 9-17

有时我们需要一个命令显示依赖于项目匹配的某种动作。例如，在前一个例子中，我们用过滤基本树的方法去转换 `Scripting` 类 `<ITEM>` 的一部分。在下一个例子中，我们将使用基于种类类型的不同的转换。

在这个例子中，我们想要表的每一行使用不同的颜色，每一个种类类型使用不同的颜色。我们可以使用 `if/elseif` 方法，以 `<xsl:choose>` 元素的形式。这种形式总是和 `<xsl:when>` 元素同时使用的。每一个条件被 `<xsl:when>` 命令判断，更独特的这种方式的判断属性：

程序清单 9-25

```

<?xml version="1.0"?>

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

  <xsl:template name="DoTableBody">

```

```

<th align="left"><xsl:apply-templates select="."/></th>
<td><xsl:apply-templates select="."/></td>
<td><xsl:apply-templates select="."/></td>
<td><xsl:apply-templates select="."/></td>
</xsl:template>

<xsl:template match="/*">
  <html xmlns="http://www.w3.org/1999/xhtml">
    <head>
      <title>The book catalog listed in a table</title>
    </head>
    <body>
      <table border="1" cellpadding="0" cellspacing="3">
        <tbody>
          <xsl:for-each select="/BOOKLIST/ITEM">
            <xsl:choose>
              <xsl:when test="contains(CATEGORY/text(),'HTML')">
                <tr style="color:red">
                  <xsl:call-template name="DoTableBody"/>
                </tr>
              </xsl:when>
              <xsl:when test="contains(CATEGORY/text(),'Scripting')">
                <tr style="color:green">
                  <xsl:call-template name="DoTableBody"/>
                </tr>
              </xsl:when>
              <xsl:when test="contains(CATEGORY/text(),'ASP')">
                <tr style="color:blue">
                  <xsl:call-template name="DoTableBody"/>
                </tr>
              </xsl:when>
              <xsl:when test="contains(CATEGORY/text(),'JavaScript')">
                <tr style="color:yellow">
                  <xsl:call-template name="DoTableBody"/>
                </tr>
              </xsl:when>
            </xsl:choose>
          </xsl:for-each>
        </tbody>
      </table>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>

```

在上面的例子中，我们用 `<xsl:choose>` 方式建立 if/elseif 命令。每一个条件用 `<xsl:when>` 元素判断。你可能注意到我们使用和上一个例子一样的表达式，但是这一次使用它检验每一个单独的分支。既然这样，我们检验是否 `<CATEGORY>` 元素数据内容包含字符串 'HTML'。如果是真的，我们把这行的颜色属性设为红色。我们继续这种方式，检验另外一个 category 类型。和上一个例子不同的是，我们检查许多条件，而上一个只是检查简单的条件。图 9-18 显示了输出。

需要说明的一个重要情况是，我们使用了一种子程序作为模板，简单的表格不能使用这种方式。因为这样，我们使用通常的 `<xsl:stylesheet>` 命令替代它。

Instant JavaScript	Scripting	1998-04-21	\$49.34
Instant Active Server Pages	ASP	1998-05-10	\$23.45
Instant HTML	HTML	1998-03-07	\$34.23

图 9-18

9.4.5 名称模板

在前一个例子中我们使用没有参数的名称模板：

程序清单 9-26

```
<xsl:template name="DoTableBody">
  <th align="left"><xsl:apply-templates select="./TITLE"/></th>
  <td><xsl:apply-templates select="./CATEGORY" /></td>
  <td><xsl:apply-templates select="./RELEASE_DATE" /></td>
  <td><xsl:apply-templates select="./PRICE" /></td>
</xsl:template>
```

一个名称模板能够接收参数。在我们的例子中，想要行头部的对齐方式作为参数（左、右、居中）。为了这样做，我们加入<xsl:param...>元素命名模板，像下面的片段中显示的那样：

程序清单 9-27

```
<xsl:template name="DoTableBody">
  <xsl:param name="alignment">left</xsl:param>
  <xsl:param name="color">green</xsl:param>
  <tr style="color:{ $color }">
    <th align="{ $alignment }">
      <xsl:apply-templates select="./TITLE"/>
    </th>
    <td><xsl:apply-templates select="./CATEGORY" /></td>
    <td><xsl:apply-templates select="./RELEASE_DATE" /></td>
    <td><xsl:apply-templates select="./PRICE" /></td>
  </tr>
</xsl:template>
```

一个参数被命名为 alignment，它的默认值是 left。另外一个参数是 color，它的默认值是 green。这些默认值能被 call-template 重载为命名模板的参数值：

程序清单 9-28

```
<xsl:call-template name="DoTableBody">
  <xsl:with-param name="alignment">
```

```

        center
    </xsl:with-param>
    <xsl:with-param name="color">
        red
    </xsl:with-param>
</xsl:call-template>

```

下面显示了用 call-template 命令创建的两个不同的 HTML 文档（参见图 9-19）。第一个图是用 call-template 创建，无参数，两个参数值是默认的，alignment 的值为 left，color 的值为 green。另外一个模式是带参数的 call-template 命令创建的，在 ASP 种类中，两个参数被设为 center 和 red。

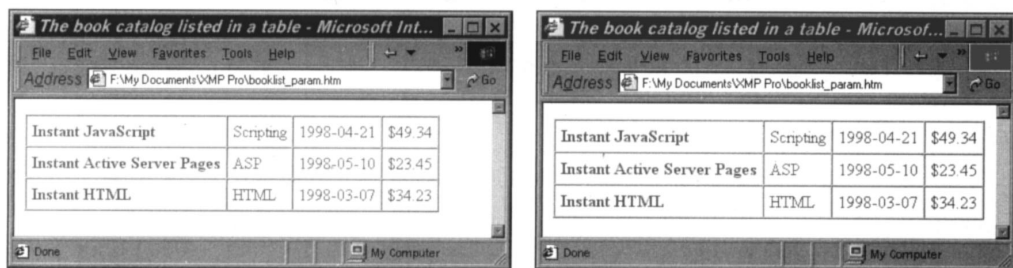


图 9-19

9.4.6 编号方式

现在，让我们给 XSLT 风格表加入行编号，其结果将如图 9-20 所示。

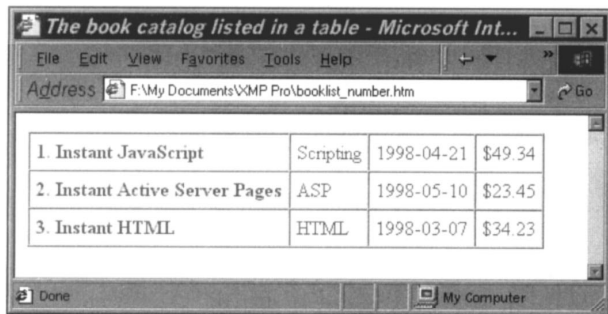


图 9-20

为了获得这个结果，我们使用和前一个例子相同的样式表，但是这一次我们加入 <xsl:number...> 命令。命令的属性是 value 和 format，value 指明了插入到输出树中的值，format 指明了输出的格式。我们指定 value 属性为元素集（<ITEM> 元素）中的当前指针位置。下面是代码段的模式：

程序清单 9-29

```

<xsl:template name="DoTableBody">
  <xsl:param name="alignment">left</xsl:param>
  <xsl:param name="color">green</xsl:param>

```

```

<tr style="color:{ $color} ">
  <th align="{ $alignment} ">
    <xsl:number value="position()" format="1. " />
    <xsl:apply-templates select="./TITLE"/>
  </th>
  <td><xsl:apply-templates select="./CATEGORY" /></td>
  <td><xsl:apply-templates select="./RELEASE_DATE" /></td>
  <td><xsl:apply-templates select="./PRICE" /></td>
</tr>
</xsl:template>

```

<xsl:number...>元素能够使用一些另外的属性，增加它的灵活性。要得到更多的细节，查看XSLT在网上<http://www.w3.org/TR/xslt>中的声明。

下面是前 5 个例子的总结，在原始 BooklistXML 文档中应用的 XSLT 命令：

- 循环——使用<xsl:for-each...>元素。
- 排序——使用<xsl:sort...>元素。
- 条件处理——使用<xsl:if...>或<xsl:choose...>元素。
- 名称模板——使用<xsl:template name...>，<xsl:param...>，<xsl:with-param...>和<xsl:call-template...>元素。
- 编号方式——使用<xsl:number...>元素。

程序清单 9-30

```

<?xml version="1.0"?>

<xsl:stylesheet xsl:version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/TR/xhtml1/strict">

  <xsl:output method="html"/>

  <xsl:template name="DoTableBody">
    <xsl:param name="alignment">left</xsl:param>
    <xsl:param name="color">green</xsl:param>
    <tr style="color:{ $color} ">
      <th align="{ $alignment} ">
        <xsl:number value="position()" format="1. " />
        <xsl:apply-templates select="./TITLE"/>
      </th>
      <td><xsl:apply-templates select="./CATEGORY" /></td>
      <td><xsl:apply-templates select="./RELEASE_DATE" /></td>
      <td><xsl:apply-templates select="./PRICE" /></td>
    </tr>
  </xsl:template>

  <xsl:template match="/">
    <html xmlns="http://www.w3.org/TR/xhtml1/strict">
      <head>
        <title>The book catalog listed in a table</title>
      </head>
      <body>
        <table border="1" cellspacing="0" cellpadding="5">
          <tbody>
            <xsl:for-each select="/BOOKLIST/ITEM">
              <xsl:sort select="./CATEGORY"/>

```



```

<xsl:sort select="//TITLE"/>
<xsl:choose>
  <xsl:when test="contains(CATEGORY/text(),'HTML')">
    <xsl:call-template name="DoTableBody"/>
  </xsl:when>
  <xsl:when test="contains(CATEGORY/text(),'Scripting')">
    <xsl:call-template name="DoTableBody">
      <xsl:with-param name="color">
        blue
      </xsl:with-param>
    </xsl:call-template>
  </xsl:when>
  <xsl:when test="contains(CATEGORY/text(),'ASP')">
    <xsl:call-template name="DoTableBody">
      <xsl:with-param name="alignment">
        center
      </xsl:with-param>
      <xsl:with-param name="color">
        red
      </xsl:with-param>
    </xsl:call-template>
  </xsl:when>
</xsl:choose>
</xsl:for-each>
</tbody>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>

```

图9-21是通过以下步骤实现的。

- 用XT转换原始的XML文档。
- 使用Microsoft IE显示转换的结果（HTML文档）。

任何一个与W3C规范完全兼容的XML浏览器将转换和显示文档，如果初始的XML文档包含一个<xsl:stylesheet...>元素。

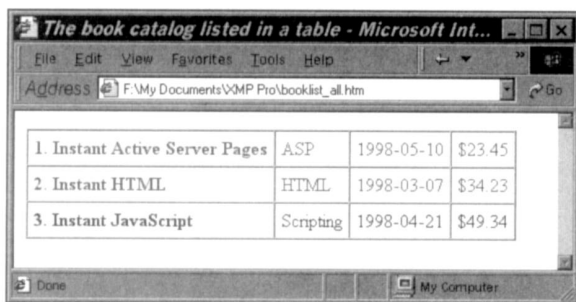


图 9-21

9.4.7 拷贝

我们现在可以对 Booklist XML 文档执行一个不同的操作。我们将在转换后的文档中保留与

初始的XML文档一样的命令，并且结果仍然为XML。我们将对<ITEM>元素进行简单的排序，根据它的<CODE>值（数据内容）。下面的列表将完成这项工作：

程序清单 9-31

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

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

  <xsl:output method="xml"/>

  <xsl:template match="*">
    <xsl:copy>
      <xsl:apply-templates>
        <xsl:sort select="//CODE"/>
      </xsl:apply-templates>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

首先，只有一个模板用于匹配任意的元素节点。然后，<xsl:copy>元素指明XSLT引擎拷贝元素节点到结果树。我们也告诉XSLT引擎通过<CODE>元素的值进行排序。结果通过下面的XML文档进行了演示——初始的文档保持了它的命令，但是<ITEM>元素根据<CODE>的值进行了排序：

程序清单 9-32

```
<BOOKLIST>
  <ITEM>
    <CODE>16-041</CODE>
    <CATEGORY>HTML</CATEGORY>
    <RELEASE_DATE>1998-03-07</RELEASE_DATE>
    <TITLE>Instant HTML</TITLE>
    <PRICE>$34.23</PRICE>
  </ITEM>
  <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>
</BOOKLIST>
```

9.5 使用DOM进行XML文档的转换

一个XML文档也可以使用DOM来转换。DOM是一个树林的接口，用于文档的表示。然而，

使用DOM来转换XML文档可能会步入危险的境地，因为大部分的 DOM实现可能有大量的私有的命令。然而，作为这些操作命令的一种解释，你应该把它归咎于 DOM规范的不足。例如，甚至在DOM2规范中也没有指出如何装入或存储一个 XML文档，所以一个DOM实现器将不得不发明出对于文档适合或不适合的方法，这样在很多的情况下，他们包含了一些私有的命令。

9.5.1 用DOM进行命令的转换

为了比较在XSLT和DOM转换XML文档命令的方法，让我们使用这一章中前面的命令化转换部分的同一个例子。

为了更新，我们想进行一个XML文档转换，具有这样的形式：

程序清单 9-33

```
<?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>
```

成为XML文档具有这样的形式：

程序清单 9-34

```
<?xml version="1.0"?>
<BOOKLIST>
  <ITEM>
    <TITLE>Instant JavaScript</TITLE>
    <DESCRIPTION>
      <CATEGORY>Category: Scripting</CATEGORY>
      <CODE>(16-048)</CODE>
    </DESCRIPTION>
    <LISTING>
      <RELEASE_DATE>Release date: 1998-04-21</RELEASE_DATE>
      <PRICE>Price: $49.34</PRICE>
    </LISTING>
  </ITEM>
```

```

<ITEM>
  <TITLE>Instant Active Server Pages</TITLE>
  <DESCRIPTION>
    <CATEGORY>Category: ASP</CATEGORY>
    <CODE>(16-105)</CODE>
  </DESCRIPTION>
  <LISTING>
    <RELEASE_DATE>release date: 1998-05-10</RELEASE_DATE>
    <PRICE>Price: $23.45</PRICE>
  </LISTING>
</ITEM>
<ITEM>
  <TITLE>Instant HTML</TITLE>
  <DESCRIPTION>
    <CATEGORY>Category: HTML</CATEGORY>
    <CODE>(16-041)</CODE>
  </DESCRIPTION>
  <LISTING>
    <RELEASE_DATE>release date: 1998-03-07</RELEASE_DATE>
    <PRICE>Price: $34.23</PRICE>
  </LISTING>
</ITEM>
</BOOKLIST>

```

下一部分将展示，VBScript可以修改Booklist XML文档的命令。

VBScript例子

VBScript可以同我们使用XSLT一样获取同样的结果。这一部分中的脚本不可移植（原因将在下一部分描述），并且只能运行在Windows平台上。这就是在使用XSLT与使用带DOM的脚本语言的主要不同。为什么选择VBScript？原因很简单——因为三百万的开发者可以读懂它。

下面的脚本可以通过Windows脚本主机（WSH，Windows Script Host）来运行。假设你已经安装了WSH，很简单，将文件保存为Transform.vbs，并且双击它：

程序清单 9-35

```

Set xmlDoc = CreateObject("Microsoft.XMLDOM")
XMLDoc.load("C:\My Documents\XML Pro\Booklist.xml")

Set oItem = xmlDoc.getElementsByTagName("ITEM")
Set oTitle = xmlDoc.getElementsByTagName("TITLE")
Set oCode = xmlDoc.getElementsByTagName("CODE")
Set oPrice = xmlDoc.getElementsByTagName("PRICE")
Set oRelease_Date = xmlDoc.getElementsByTagName("RELEASE_DATE")
Set oCategory = xmlDoc.getElementsByTagName("CATEGORY")

For i=0 To (oItem.length -1)
  Set oDescription = XMLDoc.createElement("DESCRIPTION")
  Set oTempDescription = oItem.item(i).appendChild(oDescription)
  oTempDescription.appendChild(oTitle.item(i))
  oTempDescription.appendChild(oCode.item(i))
  oTempDescription.appendChild(oCategory.item(i))
  Set oListing= XMLDoc.createElement("LISTING")
  Set oTempListing = oItem.item(i).appendChild(oListing)
  oTempListing.appendChild(oRelease_Date.item(i))
  oTempListing.appendChild(oPrice.item(i))
Next

XMLDoc.Save("sample.xml")

```

让我们看一下这个脚本的更多的细节。第一个任务是使用 `CreateObject()` 方法创建一个 DOM 对象。然后装入源文档到 DOM 中，分析它，并且填入内部的树状命令——所有这些使用 `load()` 方法：

```
Set xmlDoc = CreateObject("Microsoft.XMLDOM")
XMLDoc.load("C:\My Documents\XML Pro\Booklist.xml")
```

上面的两行不是 DOM 规范的一部分，它们只是特别用在 VBScript 环境中。

很幸运，我们没有得到一个复杂的文档命令，使用 `getElementsByTagName()` 方法可以很容易地得到需要的元素。如果文档命令包含同样的元素，但是位于不同的级别下，可能就会复杂得多。那么，下一步就是得到需要转换的所有的元素对象：

程序清单 9-36

```
Set oItem = xmlDoc.getElementsByTagName("ITEM")
Set oTitle = xmlDoc.getElementsByTagName("TITLE")
Set oCode = xmlDoc.getElementsByTagName("CODE")
Set oPrice = xmlDoc.getElementsByTagName("PRICE")
Set oRelease_Date = xmlDoc.getElementsByTagName("RELEASE_DATE")
Set oCategory = xmlDoc.getElementsByTagName("CATEGORY")
```

现在我们需要处理包含在 `booklist` 中的任何的 `<ITEM>` 元素对象。为了实现它，在 DOM 中我们获得了 `<ITEM>` 元素节点的对象实例的个数，使用 `length()` 方法。你可能希望在这里是 `count()` 方法，但是 W3C 规范使用 `length()` 来表示一个计数实例。注意，有效子节点的范围是从 0 到 `length-1`。

```
For i=0 To (oItem.length -1)
```

因为我们不得不增加一个新元素节点作为 `<ITEM>` 元素的子节点，我们创建了一个对象工厂保存在 DOM 对象中：

```
Set oDescription = XMLDoc.createElement("DESCRIPTION")
```

然后我们在当前处理的 `<ITEM>` 节点的后面追加新的元素，作为一个子节点。一个元素对象被返回，代表了 `<DESCRIPTION>` 元素节点。

```
Set oTempDescription = oItem.item(i).appendChild(oDescription)
```

到这个阶段，我们已经修改了内部命令，通过增加 `<DESCRIPTION>` 节点为 `<ITEM>` 元素的子节点，如图 9-22 所示。

然后我们包含了 `<TITLE>`、`<CODE>` 和 `<CATEGORY>` 作为 `<DESCRIPTION>` 元素的孩子：

```
oTempDescription.appendChild(oTitle.item(i))
oTempDescription.appendChild(oCode.item(i))
oTempDescription.appendChild(oCategory.item(i))
```

实际上，我们移动了这些节点的实际位置到一个新的位置，如图 9-23 所示。

然后我们使用同样的方法来创建一个新的 `<LISTING>` 元素节点。我们生成它，并把它作为子节点插入到 `<ITEM>` 元素节点中，我们将 `<RELEASE_DATE>` 和 `<PRICE>` 元素节点从原来作为 `<ITEM>` 元素节点的子节点的位置，移动到作为 `<LISTING>` 元素节点的子节点的新的位置：

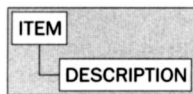


图 9-22

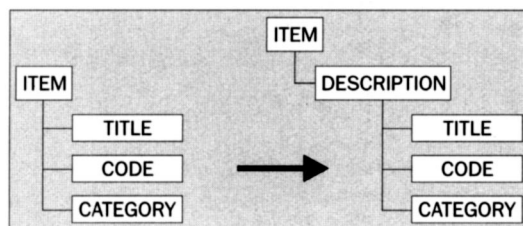


图 9-23

程序清单 9-37

```

Set oListing= XMLDoc.createElement("LISTING")
Set oTempListing = oItem.item(i).appendChild(oListing)
oTempListing.appendChild(oRelease_Date.item(i))
oTempListing.appendChild(oPrice.item(i))
Next

```

最后，我们将转换后的节点存为一个XML文档：

```
XMLDoc.Save("sample.xml")
```

9.5.2 在运行时修改一个XSLT文档

直到现在，我们一直停留在标准的界限里，并且以前的例子可以在任何一个遵循 W3C推荐标准的XSLT处理器上运行。然而，在这一部分，我们将使用一些 Microsoft的专有扩展来演示XSLT是如何使用在用户交互上的。

一个XSL样式表有两种激活方式。一种是在要进行处理XML文档中包含一个<xsl:stylesheet>处理指令，另一种是通过DOM的私有扩展进行交互。在所有前面的例子中，我们使用了处理指令作为一个与样式表的链接，但是在这部分，我们将使用微软的专有的DOM扩展来同XSLT处理器进行交互，用来显示如何用XSLT排序图书列表。

在这个例子中使用的几个命令对于Microsoft的IE5是特殊的。一些命令不过是过时的XSLT命令，其他的在这个实现中是流行的，但不是W3C标准的一部分。所以，举出的例子只能在Microsoft的IE5上工作。其他的XSLT引擎，像XT，将会报错。

下面的XSLT脚本将用于翻译Booklist XML文档：

程序清单 9-38

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
  <html>
    <head>
      <title>The book catalog</title>
      <script>
        <xsl:comment>
          <![CDATA[
            var xslStylesheet = null;
            var xmlSource = null;
            var attribNode = null;
            function sort(field)

```

```

        {
            attribNode.value = field;
            Booklist.innerHTML =
                xmlSource.documentElement.transformNode(xslStylesheet);
        }
    ]]>
</xsl:comment>
</script>
<script for="window" event="onload">
    <xsl:comment>
        <![CDATA[
            xslStylesheet = document.XSLDocument;
            xmlSource = document.XMLDocument;
            attribNode =
                document.XSLDocument.selectSingleNode("//@order-by");
            sort('TITLE');
        ]]>
    </xsl:comment>
</script>
</head>
<body>
    <div id="Booklist"></div>
</body>
</html>
</xsl:template>

<xsl:template match="BOOKLIST">
    <table border="0" frame="border" cellpadding="0" width="100%">
        <thead title="Alt-click sorts in descending order.">
            <tr style="background: brown; color: white; font-family: MS Sans Serif; font-size: 10pt">
                <th onclick="sort('TITLE')" width="33%" style="cursor: hand;">
                    <div>Title</div>
                </th>
                <th onclick="sort('CATEGORY')" width="20%" style="cursor: hand;">
                    <div>Category</div>
                </th>
                <th onclick="sort('CODE')" width="10%" style="cursor: hand;">
                    <div>Code</div>
                </th>
                <th onclick="sort('RELEASE_DATE')" style="cursor: hand;">
                    <div id="RELEASE_DATE">Release Date</div>
                </th>
                <th onclick="sort('PRICE')" width="10%" style="cursor: hand;">
                    <div>Price</div>
                </th>
            </tr>
        </thead>
        <tbody id="BOOKLIST_TABLE_BODY">
            <xsl:for-each select="//ITEM" order-by="//TITLE">
                <tr>
                    <td><xsl:value-of select="TITLE"/></td>
                    <td><xsl:value-of select="CATEGORY"/></td>
                    <td><xsl:value-of select="CODE"/></td>
                    <td align="center"><xsl:value-of select="RELEASE_DATE"/></td>
                    <td><xsl:value-of select="PRICE"/></td>
                </tr>
            </xsl:for-each>
        </tbody>
    </table>
</xsl:template>

</xsl:stylesheet>

```

Microsoft的IE浏览器要做的第一件事是分析 XML和XSL文档。Microsoft的分析器为两个文

档创建了一个文档模型。这些文档中的每个都可以被 DOM的扩展版本进行处理。Microsoft用 COM接口实现了DOM，并且加入了新的函数。我们可以说，在某些方面 Microsoft的扩展接口继承了W3C的标准接口，并且加入了新的属性和方法。就是这些被加入的方法，不是 W3C的方法，我们将在例子中使用。

执行运行时动态行为的元素是 `<script>` 元素。在运行时被分析和执行的第一个脚本没有与任何特别对象关联。这个脚本即没有 `for` 属性也没有 `event` 属性。

程序清单 9-39

```
<script>
  <xsl:comment>
    <![CDATA[
      var xslStylesheet = null;
      var xmlSource = null;
      var attribNode = null;
```

脚本本身是用 `<xsl:comment>` 元素括起来的。`<xsl:comment>` 被翻译（在输出树中）成 XML 文档注释元素。这个脚本是用 JavaScript 所写的，你可能注意到，我们声明了三个对象，并且赋了一个 `null` 值。

在运行时将被激活的第一个事件是 `window.onload` 事件。我们在这个事件上附上一段脚本：

程序清单 9-40

```
<script for="window" event="onload">
  <xsl:comment>
    <![CDATA[
      xslStylesheet = document.XSLDocument;
      xmlSource = document.XMLDocument;
      attribNode =
        document.XSLDocument.selectSingleNode("//@order-by");
      sort('TITLE');
    ]]>
  </xsl:comment>
</script>
```

首先，我们从文档对象中得到 XSL 样式表文档。然后，我们从文档对象中得到 XML 文档。在 XML 对象分级中，文档对象包括了两个扩展的 DOM：XML 扩展 DOM 和 XSL 扩展 DOM（参见图 9-24）。

两个对象都是有用的，对排序、转换和在 IE 上显示 XML 文档。

我们也可以从 XSL 树上得到排序域对象。为了得到这个对象，我们请求扩展 DOM 返回找到的包含 `order-by` 属性的第一个对象。这里有一个单个的元素包含这样的属性：`<xsl:for-each>` 结构。

程序清单 9-41

```
<xsl:for-each select="//ITEM" order-by="//TITLE" >
  <tr>
    <td><xsl:value-of select="TITLE"/></td>
```

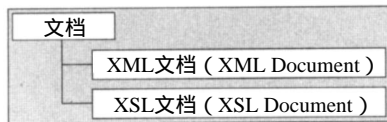


图 9-24

```

        <td><xsl:value-of select="CATEGORY"/></td>
        <td><xsl:value-of select="CODE"/></td>
        <td><xsl:value-of select="RELEASE_DATE"/></td>
        <td><xsl:value-of select="PRICE"/></td>
    </tr>
</xsl:for-each>

```

应该提醒大家的是，在写出这个文档的时候，最新的规范已经改变了排序机制，这个命令不再是一个标准的了。所以，当 IE 变得与这个规范兼容时，排序应该被定义成：

程序清单 9-42

```

<xsl:for-each select="//ITEM">
    <xsl:sort select="TITLE"/>
    <tr>
        <td><xsl:value-of select="TITLE"/></td>
        <td><xsl:value-of select="CATEGORY"/></td>
        <td><xsl:value-of select="CODE"/></td>
        <td><xsl:value-of select="RELEASE_DATE"/></td>
        <td><xsl:value-of select="PRICE"/></td>
    </tr>
</xsl:for-each>

```

所以，JavaScript 中包含排序对象的行变成：

```

attribNode =
    document.XSLDocument.selectSingleNode("xsl:sort/@select");

```

selectSingleNode() 返回的是属性节点，不是元素节点。每一个有一个或多个属性的元素，在文档树中，被转化成有一个或多个属性子节点的元素节点。所以，attribNode 变量包含的是属性节点对象。

接下来，我们通过标题的顺序来排序和显示条目：

```
sort('TITLE');
```

使用 sort() 函数：

程序清单 9-43

```

function sort(field)
{
    attribNode.value = field;
    Booklist.innerHTML =
        xmlSource.documentElement.transformNode(xslStylesheet);
}

```

首先，属性节点对象的值设成了 'TITLE' 字符串。实际上，这个有效地修改了扩展 DOM。所以这里代替用 XSL 修改 XML 文档，而是使用扩展 DOM——在运行时，XSL 样式表可能被扩展 DOM 和一段相应的脚本修改。

第二行需要更多的解释。首先，你可能会自问，Booklist 对象是从哪里来的。这个对象是在 XSL 脚本中使用下面的命令创建的：

```

<body>
    <div id="Booklist"></div>
</body>

```

如你所见，当创建一个<DIV>元素时，我们就创建了一个名为 Booklist的唯一标识对象。这是一个HTML元素，用来接收XML到HTML转换的结果。所以，当下面的表达式被 JavaScript解释器执行时：

```
Booklist.innerHTML =  
xmlSource.documentElement.transformNode(xslStylesheet);
```

我们对初始的XML文档中的documentElement对象调用了transformNode()方法。这是一个XSL的扩展DOM，用来排序 xslStylesheet变量。然后，结果被排序放在 Booklist对象的innerHTML属性中。这个会引起HTML文档的刷新，从而造成显示刷新。然后排序后的表显示在屏幕上（见图9-25）。

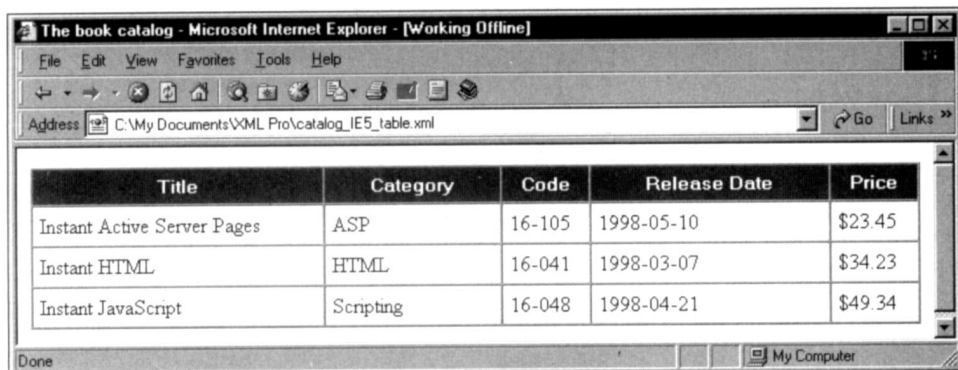


图 9-25

表头被设置了CSS样式，用于指示每次光标在表头时，浏览器显示出一只小手。用户习惯于在可被点击的东西上定义一只小手。每一列的表头都同一个 sort()函数相联，在这个例子中，行为就像onClick事件的事件处理器。例如，如果用户点击了 Price列，那么排序函数被调用，使用 <onclick="sort('PRICE')"...>命令。sort()函数接收 'PRICE' 字符串作为一个参数，设置为 attribNode变量的值。实际上，这个就改变了XSL样式表。改动等于将下面的命令：

```
<xsl:for-each select="//ITEM" order-by="//PRICE" >
```

替换成：

```
<xsl:for-each select="//ITEM" order-by="//TITLE" >
```

同最初的XSL样式表。在XSL扩展DOM已经被修改完毕，我们再一次地转换了初始的XML文档，使用改过的XSL样式表，并且将这个转换的结果作为HTML Booklist对象（如此命名是因为<DIV>的id）的innerHTML属性的输入。

所以，因为XSL文档被转换成一个树状命令（因为扩展DOM是对这棵树提供的接口），就可能使用这个接口修改XSL脚本。被修改的XSL脚本然后可以对原始文档执行一个不同的转换。所以，为了得到基于用户交互的不同结果，XSL脚本可以在脚本语言中被修改。

9.6 XSL转换与DOM转换的比较

XSL与DOM转换处理的一个最主要的不同之处是，XSL是一种公布语言，而不是过程语言。

因为这样，XSL描述的是转换后的文档状态与初始文档的关系。DOM则是一种API，它允许对树状命令进行操作。

我们看到前面 WSH VBScript使用DOM来达到与我们使用XSLT样式表同样的转换结果。但是我们可以说，更精细的转换引擎（由DOM1或DOM2规范所命令的）比XSLT来说限制更多。这主要是因为，DOM1和DOM2规范没有集成XPath表达式合并为到达一个特别的树状命令节点的能力。所以，在某种条件下，用DOM来转换XML文档要比使用XSLT要困难得多。如果未来的DOM规范版本包括了通过XPath表达式到达一个特别节点的能力，那么使用DOM可能像使用XSLT一样容易和有效。

如同我们在其他的DOM使用的例子中看到的，一个XSLT样式表可以转换一个XML文档为HTML。结果的HTML文档可以包含脚本，用于在后面操纵内容的树状命令。脚本过程可以通过用户的动作被触发，并且这些脚本可能包含使用DOM API的代码来操纵与初始的XML文档相关的XSLT文档。这就是我们所做的，当用户在一个表头点击时，我们改变一些XSLT元素的值（存储的值），来对这个列进行排序。在这个例子中，DOM用于改变XML元素的属性值，并且由于XSLT本身也是一个XML文档，它也可以使用DOM API进行修改。这时，DOM通过提供排序向XSLT转换加入值，而不必在脚本中包括排序代码。

总之，我们可以说在实际的技术状态中，XSLT样式表可以做得比使用DOM API脚本更易移值。就像在开始看到的——现实的DOM1和DOM2规范不包含任何装入和保存XML文档的命令。因为所有这些原因，最好使用XSLT样式表来进行转换，而不是使用包含DOM命令的脚本。

9.7 小结

在这一章里，我们已经看过了转换XML文档命令。特别是，我们花费了大量时间关注于XSLT（XSL转换）上。这需要XPath和XPointer的知识，在第8章已经学过了。

我们看到有几个为什么需要转换XML文档的原因。这些包括：

- 将XML转换为一种表式语言。
- 在XML的不同词汇表中进行转换。
- 生成动态文档。

XSLT实际上是一个巨大的专题，希望这一章能够让你习惯于这个特别规范的语法。而且关于这个题目很可能将被写成一本书（的确，留意一个Wrox出版社的《XSLT Programmer's Reference》，ISBN 1-861003-12-9），这个将让你习惯可用的全部功能，并且在写你自己的转换样式表时提供一个坚实的基础。

由于在XSLT规范完成之前，在IE5中可用的实现就被介绍了，所以它们有一些不同，包括一些扩展。然而，也有一些可以用在你的应用程序中的XSLT处理器：

- XT - <http://www.jclark.com/xml/xt.html>。
- SAXON - <http://users.iclway.co.uk/mhkay/saxon>。