

第17章 实例研究3——图书目录信息服务

在本章的实例研究中,我们将在SAX和XPath的基础上创建强大的图书目录信息服务(BCIS)。该系统允许出版者上载描述图书内容的XML目录文件,过滤其中的详细信息,然后以XML或HTML的格式通过电子邮件传递给系统的订阅者。两种传输格式都提供了URL,通过它们可以查看书目,并在Amazon.com进行在线购买。这些URL都是根据图书的ISBN和Amazon目前的Web站点结构动态生成的。它们并没有包含在原始的源文档中,这意味着BCIS系统具备包含到任何在线书店链接的潜力,这主要取决于订阅者的喜好。

BCIS必须具备扩展性,能够处理任意大小的XML目录文件,所以它使用SAX解析器来访问所有的XML数据文档。我们曾经在本书前面的一些章节中讨论过,SAX解析器在处理前并不把整个XML文档调入到内存中,而且在任何时候都只在内存中保留文档的一小部分——内存的确切使用数量取决于文档中的元素嵌套情况。这意味着对能够处理的文件大小并没有上限限制,也不必关心处理过程中使用的内存数量。这两点都非常重要。如果系统在DOM中载入文档,通常情况下它会受到可用内存数量的限制,或者因为分页而运行得非常缓慢。

BCIS是根据出版商/购买者之间的推模式规划的:出版商(服务器)直接把图书的细节传递给购买者(客户),而不是购买者向出版者索取相应的信息。BCIS的订阅者可以选择两种图书目录(例如与ASP、XML等等有关的图书),过滤出需要传递给自己的信息。所以,如果出版商上载的目录文件包含8 000本关于花园的图书,25 000本关于汽车的图书,2本关于XML的图书,那么注册时仅表示出对XML兴趣的订阅者只会收到关于这2本XML图书的信息。

出版商/购买者之间的推模式有几个关键的优势,使得它对于许多应用来说是一个极具吸引力的选择:

- 它节省了购买者的时间,因为不再需要翻阅出版商的信息,以了解最新变化。
- 它使得信息发布者能够节省宝贵的资源和带宽,因为服务器不再需要处理和服务由客户端可能发出的大量请求(因为没有发生什么变化)。
- 出版者知道订阅者更希望了解的是信息,所以不再依赖于读者记得回到源位置来查看是否有更新信息。

可能时下最流行的出版商/购买者服务的应用就是新闻邮件和Web站点的更新通知。在这两种情况下,出版商知道客户端感兴趣的是他们所提供的信息,或者提供的特定服务,所以在相关事件(例如Web站点中增加了新页面)发生时会把信息用电子邮件通知客户。当客户接收到信息时,会在空闲时阅读其中的内容,再决定是否采取进一步的动作,例如点击可能包含在电子邮件当中的超链接。这些类型的应用也可能根据过滤、传输、从服务器向客户端传递数据的原则进行构建。这意味着BCIS的编制风格使它很容易就能支持任何类型的信息和传输机制。正是出于这一原因,这类应用的设计看起来稍微有些复杂,但我们希望在本章结束时大家能够理解其中的原理。

到本章结束时，我们将了解到：

- 如何在 Visual Basic 中使用 SAX 解析器处理任意大小的 XML 文档，且无须使用太多的内存。注意，虽然我们使用的是 VB，我将保证大家能够掌握这类应用如何用任意语言实现，因此我们也将解释任何“VB 主义”的内容，虽然对于那些只进行过 C/C++ 或 Java 编程的人来说这有一些古怪。
- 一种有趣的 XSLT 替代品，通过将元代码用于模版，能够实现高性能的、编译过的样式表。
- 如何开发一个基于 XML 的推模式程序，它适用于许多不同的应用。

17.1 图书目录信息服务

在 BCIS 中有三种目标客户类型：

- 订阅者——根据所选择的两个目录，使用这类服务接收有关书籍的信息（过滤的数量可以是任意数量，之所以本例中为两个是为了减少代码的复杂性）。
- 出版商——希望把它们的书籍的信息传递给订阅者的公司或个人。
- 广告商——类似 Amazon.com 这样的公司，它们在线销售书籍，并向在传递给订阅者的信息中包含到它们的超链接的服务支付费用。

如果一名广告商既是出版商，又是订阅者也不足为奇。

在本章的实例研究中，我们不会特别讨论出版商。他们负责创建 XML 目录文件，其中包含将由系统处理的图书信息。正如我们稍后将看到的，在本实例研究中，BCIS 与订阅者之间的交互仅仅包括将图书目录文件拷贝到某个文件目录中。这种方法使得公司很容易将信息发布到 BCIS 服务器上，它们只需通过 FTP 将 XML 文档上传，但这也意味着你可能需要扩展系统，你应该决定对于推销给订阅者的每本书，是否向出版商收费。

本章的实例研究也很少涉及广告商，我们将实现的系统仅支持一个广告商，而且它是在转换代码中硬编码的。你当然可以对此进行扩展，使之包含多家公司。

17.1.1 系统概述

BCIS 的系统结构如图 17-1 所示。

出版商创建一个 XML 文档（显示为 Catalog.XML），其中包含要过滤的所有信息，他将这个文件推给订阅者。

文件 Users.XML 定义了订阅者及其过滤器。本章描述的系统并不提供用于管理该文件的 UI，因此为了增加更多的信息或者改变配置，你不得不（使用类似记事本的工具）手工修正和编辑这个文件。

SAX 用户个性化引擎加载用户信息，并根据每个用户的设置转换图书信息，为每个用户创建一个输出文件。根据我们已经讨论过的出版 / 订阅模型，每个转换后的文件可以通过电子邮件发送给用户。

1. XML 文档格式

订阅者信息和图书信息保存在不同的（独立的）XML 文档中。由于我们使用的 SAX 解析器不支持有效性验证，因此系统不使用 DTD 或模式验证文件的有效性。实际上，这一点对于我们

的实例研究没有很大的影响，但是它意味着如果你从本实例中衍生出的系统需要有效性验证，你必须亲自进行验证（例如：保证根元素名称正确等）。

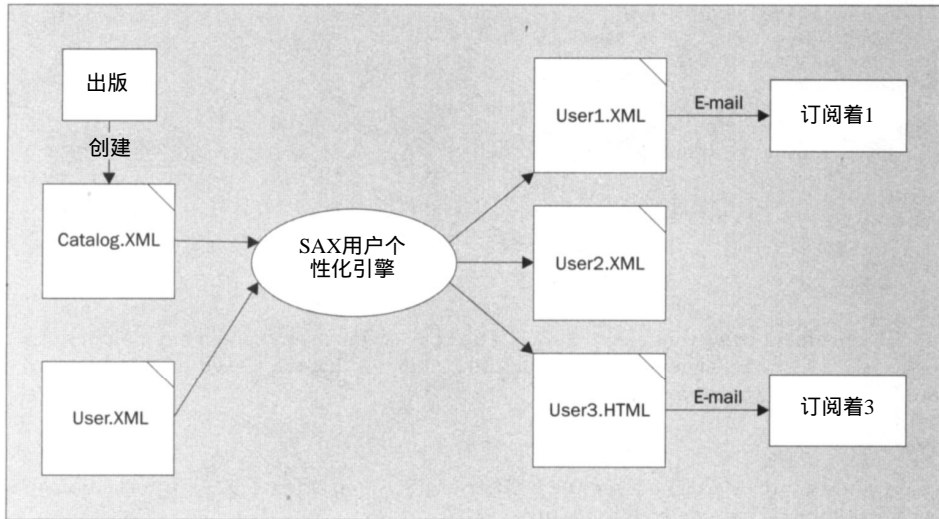


图 17-1

(1) Catalog.XML

本章的实例研究提供了一个 XML 目录样例，名为 Catalog.xml，该文件及本书的其余代码均可以从站点 <http://www.wrox.com/> 下载。该文件中包含的图书信息文件的格式是我们非常熟悉的，我们在本书中一直沿用这种格式。

程序清单 17-1

```
<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>
  <Price>$49.99</Price>
</Book>
<Book>
  ...
</Book>
</Catalog>
```

以上 XML 文档有根元素 <Catalog>，它能够包含任意数量的 <Book> 元素。每个 <Book> 元素

都有多个子元素，它们的含义不言自明，而且我们在以前的章节中曾详细介绍过。

(2) Users.XML

本章的实例研究中提供的 XML 用户文件样例名为 Users.XML。有关 BCIS 中订阅者的信息定义如下：

程序清单 17-2

```
<?xml version="1.0" ?>
<!-- This file defines the register users for the book notification service -->
<Users>

  <User>
    <Name>Richard Anderson</Name>
    <OutputFile>c:\proxml\RJA.HTML</OutputFile>
    <Email>rja@arpsolutions.demon.co.uk</Email>
    <DeliveryFormat>HTML</DeliveryFormat>
    <Category1>XML</Category1>
    <Category2>ATL</Category2>
  </User>

  <User>
    <Name>Jon Duckett</Name>
    <OutputFile>c:\proxml\JD.HTML</OutputFile>
    <Email>jond@wrox.com</Email>
    <DeliveryFormat>HTML</DeliveryFormat>
    <Category1>ASP</Category1>
    <Category2>XML</Category2>
  </User>

  <User>
    <Name>Karli Watson</Name>
    <OutputFile>c:\proxml\KW.XML</OutputFile>
    <DeliveryFormat>XML</DeliveryFormat>
    <Category1>Java</Category1>
    <Category2>Web Server</Category2>
  </User>

</Users>
```

以上 XML 的根元素为 <Users>，它能够包含任意数量的 <User> 元素。每个 <User> 元素有多个子元素（其中只能包含文本内容）：

表 17-1

子元素	说 明
<Name>	订阅者的姓名，用于记录日志
<OutputFile>	转换时输出文件的名称
<Email>（可选的）	如果存在，<OutputFile>将根据该地址通过电子邮件发送给用户
<DeliveryFormat>	指定应用于用户的转换样式表类。它的值可以是 XML 或 HTML
<Category1>	用户感兴趣的第一个图书目录。它与使用 XPath/Catalog/Book/RecSubjCategories/Category 的 XML 目录文件匹配
<Category2>	用户感兴趣的第二个图书目录。它与 <Category1> 匹配

对于本章介绍的实例研究，你可以进行扩展，例如：使之支持任意多个目录元素。为了简化代码，我仅仅实现了两个

(3) 投递格式

BCIS允许订阅者以HTML格式或XML格式接收图书信息。图 17-2显示了HTML投递格式，其中目录过滤器是使用ASP和ASP+编写的。

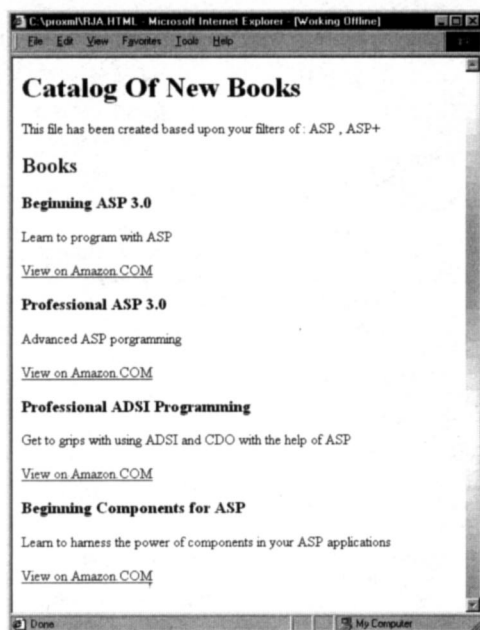


图 17-2

转换过程创建了一个非常简单的HTML输出文件，实际上，它只显示了两个原始的XML输入字段（书名和摘要）。它根据简单的算法计算到Amazon的链接，该算法首先删除ISBN号中的连字符，然后在前面增加前缀http://www.amazon.com/exec/obidos/ASIN/。

XML格式同样相当简洁，参见图 17-3。



图 17-3

以上两个转换过程都是由VB类模块执行的，我们稍后会介绍有关内容。

2. 订阅者/用户

如前所述，订阅者是BCIS实例研究中最重要实体，因为他们负责定义应用于源XML目录文档的过滤规则，投递格式（HTML或XML），以及可选的电子邮件地址。订阅者是使用类CUser模型化的，类CUserCollection用于以组为单位管理订阅者（参见图17-4）。

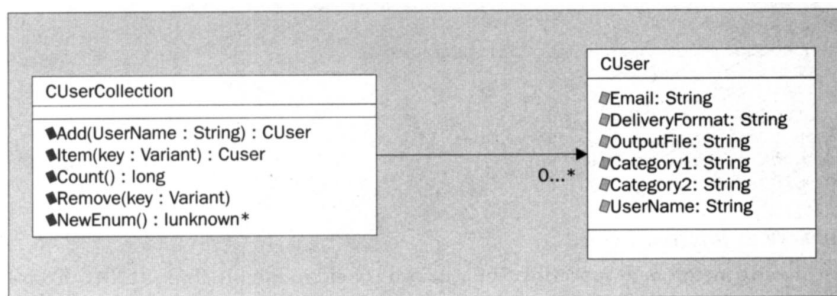


图 17-4

上图（及本章其他类似的图）采用UML表示法。我们描述一下这种表示法：它使用一个框表示类，框被分为三部分。顶部显示类名，中部显示属性（及类型），下部显示方法（以及参数和类型）。框之间的箭头表示拥有或包含等关系，箭头上的数字表示关系的元组数（一对一，一对多等）。在上图中，CUserCollection包含零个或多个CUser对象。要了解更多的信息，参见《Instant UML》（ISBN：1-861000-87-1）。

出版商和广告商并没有被模型化，因为系统并不介意谁创建了XML目录文件，而且我们只讨论到Amazon的链接。如果决定扩展系统，以每个用户/出版的每本书为单位向出版社收费，或者支持多个广告商，我们就需要将它们模型化，因为在那种情况下它们与订阅者同等重要。

(1) VB集合说明

当我们使用VB编写BCIS时，所有集合类都是使用类构造器附件创建的。

我虽然使用类构造器，但通常将变量名由mvar重命名为m_，并且使用匈牙利表示法表示变量的前缀。

在设计过程中，我不打算详细解释集合类，因为它们在现实世界中的角色非常简单：它们包含零个或多个相同类的对象。然而，考虑到有些人以前可能从未使用过VB，我将简要介绍CUserCollection类的GetEnumerator()方法。

GetEnumerator()方法允许我们使用如下的For...Each语法书写代码：

程序清单 17-3

```
Dim oUser as CUser
Dim oUserCollection as CUserCollection

For Each oUser in MyUserCollection
    MsgBox oUser.UserName
Next oUser
```


这是VB的速记特征，其结果是产生具有较强可读性的编码方法。它的底层实现是基于 COM 接口 IEnumVARIANT 的，它使我们能够摆脱传统的较为繁琐的通过索引访问集合项的方法：

程序清单 17-4

```
Dim oUser as CUser
Dim oUserCollection as CUserCollection
Dim lIndex as long

For lIndex = 0 to oUser.Count
    Set oUser = oUserCollection.Item(lIndex)
    MsgBox oUser.UserName
Next lIndex
```

(2) 加载订阅者信息

如前所述，订阅者信息是保存在文件 Users.XML 中的。为了将该文件转换为 CUser 类的实例，我们需要使用类 CLoadUserInfo（参见图 17-5）。

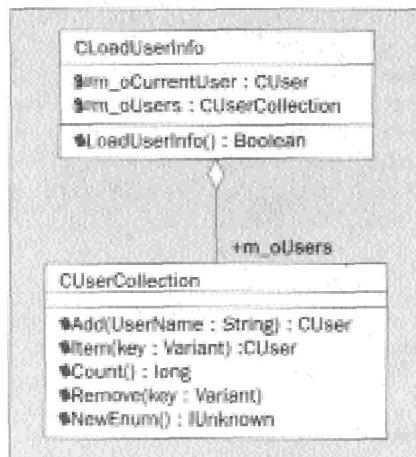


图 17-5

通过调用 CLoadUserInfo 的 LoadUserInfo() 方法，文件 Users.XML 将被转化为 CUser 对象，并放在 m_oUsers 的 CUserCollection 中。显然，XSLT 不能实现该转换过程，因此我们使用 SAX 解析器以及基于 SAX/XPath 的简单而聪明的转换引擎，这使得转换过程非常易于实现。转换引擎是由 CSAXTransformEngine 类实现的。

由于转换引擎是基于 SAX 的，因此我们不必将 XML 文档加载至 DOM。我们只需要响应 SAX 事件流中的某些事件，并且构建自己的 CUser 对象。这样就不必要为 XML 文档创建 DOM，在 DOM 中搜索节点，然后删除所有 DOM 节点。这种方式主要有以下两个优越性：

- 我们不必创建 DOM 节点，并在获取所需的数据后立即删除节点，因而也就不必为此浪费处理时间和内存。
- 我们能够处理任意大小的文件。无论用户的文件多大，我们只会受到 CUser 对象占用的内存的限制，而不会因 DOM 节点导致受限。如果曾经尝试着将 20MB 文件加载至 DOM 中，

你可能会发现它所需的内存是文件大小的 3~4 倍，即：大约 80MB。对于 SAX，解析过程通常只需要几 KB，确切的大小取决于文件中的元素嵌套和字符数据。这一特征使得 SAX 非常适于类似 BCIS 的系统，因为 BCIS 等系统对于需要处理的 XML 文档大小没有上限。

SAX 主要有以下两点不足：

- 无法随机访问 XML 文档中的任意数据，除非该数据是由当前事件提供的，或者该数据已经解析出来并且保存在变量中。
- 事件中不包含元素位置关系指示。举例来说，如果接收到 `startElement` 事件，而且元素名称为 `<Title>`，我们无法确定该元素是否包含在另一个元素中（比如：`<Book>`），除非我们在处理元素的过程中通过设置某些类型的标志表示嵌套。当然，`<Book>` 与 `<Title>` 之间可以是父/子关系或祖先/子孙关系。嵌套或关系越复杂，需要保留的标志越多。

要详细了解 DOM 与 SAX 的优缺点比较，参见本书前面的有关章节（第 5 章和第 6 章）。

对于 SAX 的第一个缺点，我们不太容易克服，唯一的方法是缓存信息。如果 XML 文档不十分复杂，这个缺点不会太显著。然而，假设你在处理 XML 文档的开始处之前就需要访问 XML 文档末尾的信息，你不得不解析两次文件，或者重新考虑 XML 文档的结构，并修改生成它的应用程序。

第二个有关关系位置的问题很容易解决，你只需要编写维护基于 `startElement` 和 `endElement` 调用的元素堆栈的 SAX 过滤器，例如：

程序清单 17-5

<code>startDocument</code>	<code>Stack = /</code>
<code>startElement("Books")</code>	<code>Stack = /Books</code>
<code>startElement("Book")</code>	<code>Stack = /Books/Book</code>
<code>startElement("Title")</code>	<code>Stack = /Books/Book/Title</code>
<code>characters("ProXML")</code>	<code>Stack = /Books/Book/Title</code>
<code>endElement("Title")</code>	<code>Stack = /Books/Book/Title</code>
<code>endElement("Book")</code>	<code>Stack = /Books/Book</code>
<code>endElement("Books")</code>	<code>Stack = /Books</code>
<code>endDocument</code>	<code>Stack = /</code>

产生 SAX 事件 `startElement` 时，我们将元素名称入栈，并有效地构造 XPath。产生 SAX 事件 `endElement` 时，我们删除元素名称。堆栈最初包含“/”，它是文档的根。随着 SAX 事件的产生，逐步建立堆栈，事件以及 XPath 将一起被传给下一个处理器。其结果是我们的类将处理类似如下的事件流：

程序清单 17-6

```
startDocument

startElement("/Books")
startElement("/Books/Book")
startElement("/Books/Book/Title")
characters("/Books/Book/Title", "ProXML")
endElement("/Books/Book/Title")
```

```
endElement("/Books/Book")
endElement("/Books")
```

```
endDocument
```

你可能感觉以上代码与DOM有些类似，从某种角度讲，你可以这样认为，但是你应该记住在以上堆栈中，我们的内存中只有一个图书实例，然而如果采用DOM，内存中可能会有上千个实例——因此空闲内存极少。

BCIS是建立在一种转换技术基础之上的，该技术已经发展成为通用的转换引擎，并应用于本实例研究。转换操作并不是根据XSLT样式表执行的，它使用一个或多个类，这些类作为SAX事件被调用，同时附加与SAX事件流匹配的某些样式（XPath）。这种方法使得BCIS具有很高的性能，因为伪样式表是经过编译的，而且根据SAX事件流转换文档更加简单，因为你不必跟踪事件嵌套。

这种方法比单纯的XSLT更加灵活，因为你可以利用常用的开发语言进行转换，这些语言可能允许数据库查询等操作（这在标准XSLT中是无法实现的）。另一个我个人非常推崇的优点是在执行非常复杂的转换时，你不必纠缠于相当繁琐的XSLT语法，它有助于提高程序的编写效率。

为了理解转换引擎的优点和工作原理，考虑图17-6所示的典型的SAX应用程序。

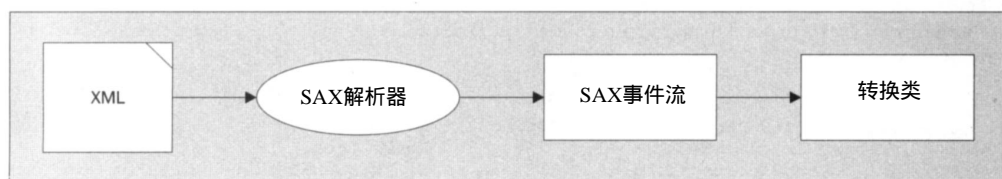


图 17-6

首先解析XML文档，然后由实现了SAX DocumentHandler类的类处理SAX事件。如果处理器代码要定位<Books>中的所有<Book>元素，它将有如下代码：

程序清单 17-7

```

Bool m_bInBooks as Boolean

Private Sub m_oSAXParser_startElement(ByVal sName As String, _
                                       ByVal pAttributeList As SAXLib.ISAXAttributeList)

    If sName = "Books" then
        m_bInBooks = true
    End if

    If sName = "Book" and m_bInBooks = true then
        MsgBox "We located a book"
    End if
End Sub

Private Sub m_oSAXParser_endElement(ByVal sName As String)
    If sName = "Books" then
        m_bInBooks = false
    End if
End Sub
  
```

SAX事件通过 sName 传递起始标记名称，通过 pAttributeList 传递起始标记中定义的属性。代码并不困难，但是定位嵌套多层的元素可能会导致大量状态标志，零乱的代码和错误。除此之外，如果我们寻找许多元素，事件处理器很快会变得相当庞大。正如我们前面所讨论的，通过引入附加的层/过滤器将 XPath 添加到事件中，将简化编程模型（参见图 17-7）。

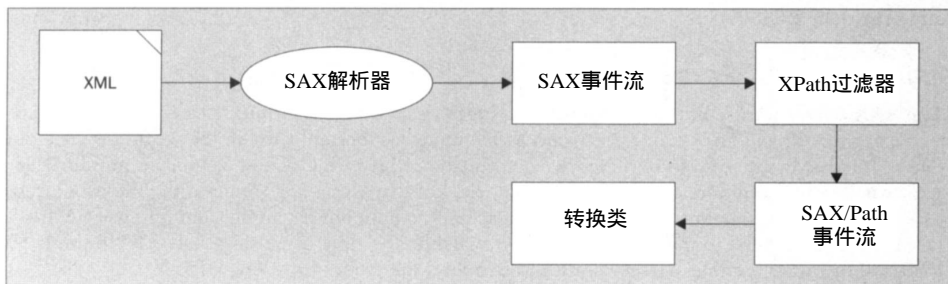


图 17-7

由于其他人负责维护元素嵌套（使用前面讨论的堆栈方法），我们就不需要任何标志，代码简化为如下形式：

程序清单 17-8

```
Private Sub m_oSAXParser_startElement(sXPath As String, _  
    sElementName As String, _  
    oAttribs As SAXLib.ISAXAttributeList)  
  
    If sXPath = "/Books/Book" then  
        MsgBox "We located a book"  
    End If  
End Sub
```

对于以上代码，可以用一个复杂的 XPath 进行测试，我们不必担心要在 startElement() 中设置标志，在 endElement() 中清除标志。代码变得极其简单，它是本章介绍的实例研究的基础。然而，如果我们需要搜索大量事件，使用 XPath 并不能解决事件处理器过大的问题。为此，我们将依据一个或多个 XPath 利用不同的类模块处理 SAX 事件（参见图 17-8）。

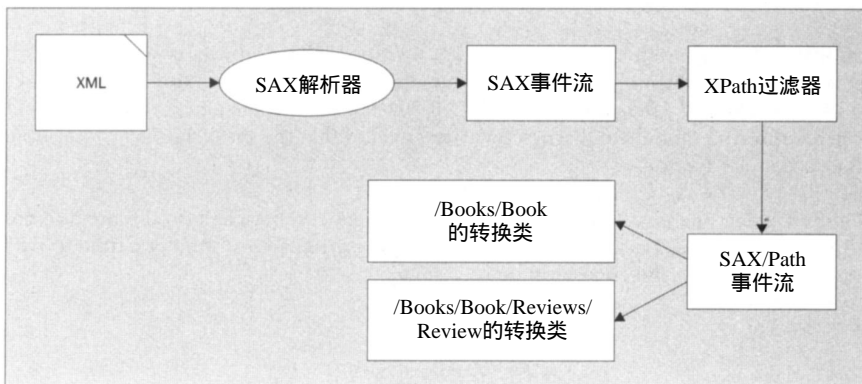


图 17-8

在上图中，一个类负责处理所有 XPath 为 /Books/Book 的 SAX 事件，另一个类负责处理 XPath 为 /Books/Book/Reviews/Review 的 SAX 事件。我们实现的引擎允许一个类处理任意多个 XPath，但是同一 XPath 至多只能由一个类处理。

此时，我们将略微偏离主题，讨论一下将在本章使用的 SAX 解析器，同时还要涉及一些 VB 代码。一旦了解了如何使用解析器，我们将继续创建 BCIS。

3. Visual Basic SAX 解析器

本章实例研究使用的 SAX 解析器是由 Vivid Creations (<http://www.vivid-creations.com>) 开发的，它称为 ActiveSAX。Vivid 公司在 XML 领域颇有名气，它是最初——1998 年年中——采用非 Java (C/C++，VB 等) 实现 SAX 的几家公司之一。对于商业/非商业用途，该解析器都是免费的，但是如果处理的文件超过 10KB，将显示一个不太讨人喜欢的小屏幕。就我们的实例研究而言，这个限制无所谓，但是如果你打算使用 SAX 处理大文件，你需要购买完整的拷贝（在编写本书时，它的价格是 99 英镑或 149 美元），或者采用其他解析器。你可以从以下地址下载解析器：
<http://www.vivid-creations.com/sax/index.htm>。

使用 ActiveSAX，我们将创建简单的应用程序，并通过列出 XML 文档中的所有元素演示如何使用 ActiveSAX。需要特别说明的是，本章并不打算详细介绍 ActiveSAX 解析器。

Vivid 公司的 Web 站点详细介绍了 ActiveSAX。站点上提供了许多可下载的 XML 样例 (SAX 和 DOM)，相信你会受益匪浅。

首先，你需要从前面列出的 URL 下载并安装 Vivid ActiveSAX 解析器，创建新的标准 EXE VB 项目，并添加对类型库 “Vivid Creations ActiveSAX” 的引用。

然后，在缺省的窗体上放置标准的 ListBox 控件，在窗体声明的代码模块的顶端声明 SAXParser 类型的变量 m_oSAXParser：

```
Dim WithEvents m_oSAXParser As SAXParser
```

你会注意到正在使用关键字 WithEvents，通过这种方式就能够使用连接点处理来自对象实例的事件：Gamma 等定义的 Observer 样式的 COM 实现（《Design Patterns》，ISBN 0-201-63361-2）。该样式与我们前面讨论的出版商/订阅者模型极其类似，除了现在出版商是一个对象，订阅者是一些 VB 代码。

定义之后，从对象组合框中选择 m_oSAXParser，点击方法组合框，依次选择事件 startElement，characters 和 endElement，将子例程原型添加到代码模块中（参见图 17-9）。

按照以下代码修改三个事件处理器的实现：

程序清单 17-9

```
Private Sub m_oSAXParser_startElement(ByVal sName As String, _  
                                       ByVal pAttributeList As SAXLib.ISAXAttributeList)  
  
    List1.AddItem "startElement:" & sName  
End Sub  
  
Private Sub m_oSAXParser_characters(ByVal sCharacters As String, _  
                                    ByVal iLength As Long)  
  
    List1.AddItem "characters:" & sCharacters
```

```
End Sub
```

```
Private Sub m_oSAXParser_endElement(ByVal sName As String)  
    List1.AddItem "endElement:" & sName  
End Sub
```

对于每个被处理的事件，我们将事件名称以及元素名称或字符数据写入列表框。
然后，按照以下代码添加窗体加载事件处理器：

程序清单 17-10

```
Private Sub Form_Load()  
    Set m_oSAXParser = New SAXParser  
    m_oSAXParser.parseString "<ProXML>Enjoy!</ProXML>"  
End Sub
```

以上代码创建SAX解析器实例，然后请求解析器解析文本“<ProXML>Enjoy!</ProXML>”。
运行代码，虽然我增加了列表框字体的大小，但是你所看到的输出应该与图 17-10非常相似。

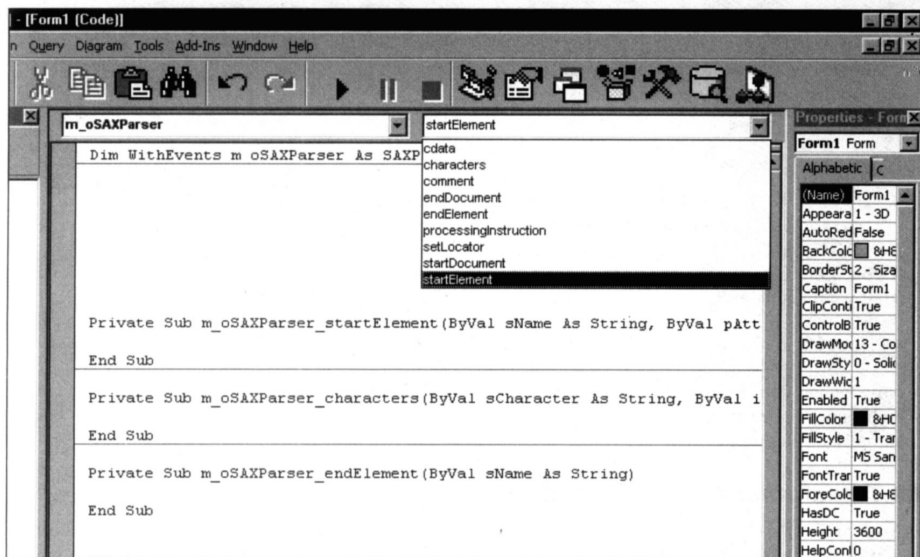


图 17-9

LoadUserInfo假设文件Users.XML位于当前工作目录中。

如果愿意，还可以通过parseFile()方法要求解析器处理文件，
我们稍后将用到该代码：

```
m_oSAXParser.parseFile "c:\proxml\users.xml"
```

既然我们已经了解了即将使用的解析器，讨论了执行转换
所需的SAX过滤器/转换技术，下面让我们分析一下真正所要求
的功能的类。

4. 转换引擎

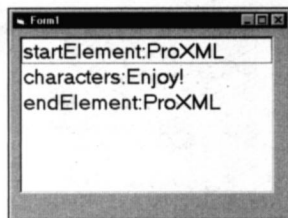


图 17-10

XSLT样式表是由模板和样式构成的。每个模板有效地描述了要放置在目标文件中的标记和元素内容，其中的样式定义的 XPath 说明如何将源文档拖动到目标位置。我们可以根据模板和样式建立转换引擎，模板是一个类中包含的一些代码，当特定的样式（XPath）与前面讨论的 XPath/SAX 事件流匹配时，将调用模板代码。例如：

程序清单 17-11

```
Private Sub m_oSAXParser_startElement(sXPath As String, _
                                     sElementName As String, _
                                     oAttribs As SAXLib.ISAXAttributeList)

    If sXPath = "/Books/Book" then
        SomeClass.startElement sXPath, sElementName, oAttribs
    End if

    If sXPath = "/Books/Book/Reviews/Review" then
        AnotherClass.startElement sXPath, sElementName, oAttribs
    End if
End Sub
```

类模块的事件映射可以应用于所有 SAX 事件，例如：endElement 和 characters：

程序清单 17-12

```
Private Sub m_oSAXParser_characters(sXPath as String, _
                                    ByVal sCharacters As String, _
                                    ByVal iLength As Long)

    If sXPath = "/Books/Book" then
        SomeClass.characters sXPath, sCharacters
    End if

    If sXPath = "/Books/Book/Reviews/Review" then
        AnotherClass.startElement sXPath, sElementName, oAttribs
    End if
End Sub
```

显然，对于每个应用程序，我们不希望手工编写以上重复匹配和事件转发代码，因此将该功能内置于 BCIS 转换引擎。

既然已经讨论了 BCIS 的功能，下面开始编写实现系统的代码。

17.1.2 创建 BCIS

创建新的标准 EXE 项目，添加对 Vivid ActiveSAX type library 的引用。

1. SAX XPath 过滤器接口——ITemplate

为了使转换引擎能够以通用的方式调用我们的模板处理器，需要定义描述所有事件的接口（仅仅定义了方法而无实现的 VB 类模块）。转换引擎将使用该接口调用模板处理器的方法，而不必知道真正实现该接口的确切的类。

添加新的类模块，将它的名称改为 ITemplate，并添加以下代码：

程序清单 17-13

```
Option Explicit
```

```
' start of the XPath

Sub startElement(sXPath As String, _
                 sElementName As String, _
                 oAttribs As SAXAttributeList)

End Sub

' characters within the XPath

Sub characters(sXPath As String, _
              sData As String)

End Sub

' end of the XPath

Sub endElement(sXPath As String, _
               sElementName As String)

End Sub

' start of document

Sub startDocument()

End Sub

' end of document

Sub endDocument()

End Sub
```

在Java或C/C++中，在此你可以只定义一个抽象类。

然后，添加名为CXMLCatalogTransform的类模块，并在代码模块中添加以下行，告知VB它实现了ITemplate接口：

```
Implements ITemplate
```

既然已经声明了实现接口的类模块，对象组合框将包含 ITemplate项。选择它，你将看到事件组合框列出了它的所有方法（参见图 17-11）。

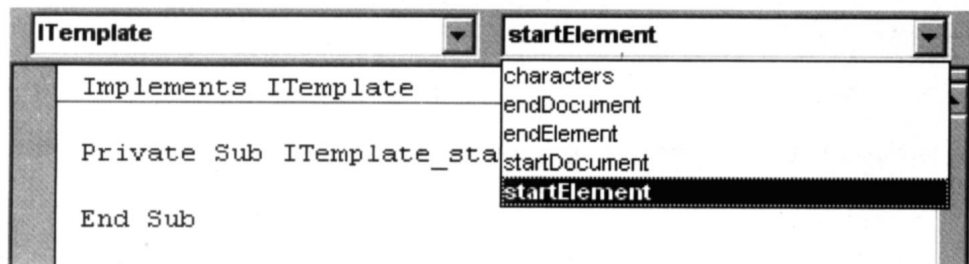


图 17-11

选择事件组合框中的每个方法，VB将向类模块中添加子例程原型：

程序清单 17-14

```

Private Sub ITemplate_characters(sXPath As String, _
                                sData As String)

End Sub

Private Sub ITemplate_endDocument()

End Sub

Private Sub ITemplate_endElement(sXPath As String, _
                                sElementName As String)

End Sub

Private Sub ITemplate_startDocument()

End Sub

Private Sub ITemplate_startElement(sXPath As String, _
                                sElementName As String, _
                                oAttribs As SAXLib.ISAXAttributeList)

End Sub

```

修改characters()方法，使之包含显示事件名称和传递的 XPath的消息框：

程序清单 17-15

```

Private Sub ITemplate_characters(sXPath As String, _
                                sData As String)
    MsgBox "CXMLCatalogTransform: xpath=" & sXPath & " data=" & sData
End Sub

```

然后，为了演示在VB中如何实现多态性和接口，在Form1的代码模块中添加以下子例程：

```

Sub DummyCharactersEvent(oTemplate As ITemplate)
    oTemplate.characters "/Catalog/Book/Title", "ProXML"
End Sub

```

该例程向接口传递引用，然后调用 characters事件，同时传递 XPath “/Catalog/Book/Title” 和数据 “ProXML”。在实际的BCIS中，这些是根据XML源文档创建的，但是现在，我们将值硬编码在程序中，以便演示引擎是如何工作的。调用 characters方法的代码不知道哪个对象类实现了接口，这意味着它能够以多态方式调用任何对象类的 characters方法。这种方法非常有效，因为你可以不修改代码的情况下，在任何时刻添加新的 XPath的类模块。为了说明这一点，在窗体中增加一个按钮，双击它，将事件处理器改为如下形式：

程序清单 17-16

```

Private Sub Command1_Click()
    Dim oXMLCatalogTransform As New CXMLCatalogTransform
    DummyCharactersEvent oXMLCatalogTransform
End Sub

```

代码创建 CXMLCatalogTransform类的实例，然后通过调用子例程引用 ITemplate接口的

characters方法。如果你运行项目，并点击按钮，将看到图 17-12所示的消息框，它是由 CXMLCatalogTransform类的实例创建的。

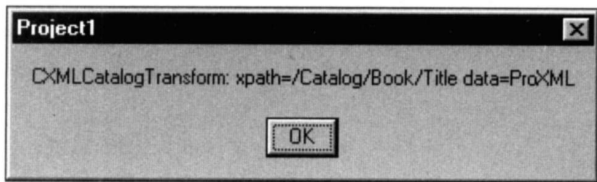


图 17-12

关键在于即使 DummyCharactersEvent()方法接受了 ITemplate类型的参数，VB仍然会向 CXMLCatalogTransform对象询问 ITemplate接口，然后调用相应的函数。为了说明这一点，添加一个名为 CHTMLCatalogTransform的类模块，从 CXMLCatalogTransform模块拷贝所有代码，但是将 ITemplate_characters()中的消息框调用改为如下形式：

程序清单 17-17

```
Private Sub ITemplate_characters(sXPath As String, _  
                                sData As String)  
    MsgBox "CHTMLCatalogTransform: xpath=" & sXPath & " data=" & sData  
End Sub
```

回到按钮的事件处理程序，修改代码，创建该类的实例，调用 DummyCharacterEvents()，并将新创建的对象作为参数传递：

程序清单 17-18

```
Private Sub Command1_Click()  
    Dim oXMLCatalogTransform As New CXMLCatalogTransform  
    DummyCharactersEvent oXMLCatalogTransform  
  
    Dim oHTMLCatalogTransform As New CHTMLCatalogTransform  
    DummyCharactersEvent oHTMLCatalogTransform  
End Sub
```

运行项目，点击按钮，除了原来的消息框，你还将看到由 CHTMLCatalogTransform类创建的消息框（参见图 17-13）。

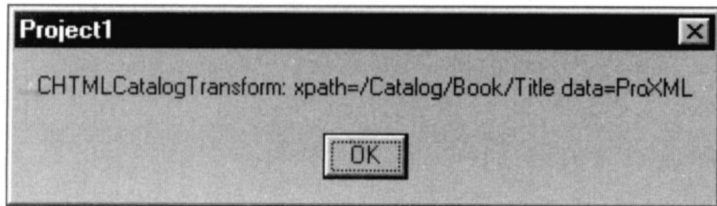


图 17-13

以上例子说明我们这个简单的函数能够调用两个不同对象类的相同函数，它同时解释了转

换引擎扩展模块应遵循的基本原则。当然，转换引擎不应该是硬编码的，它应该允许你注册 XPath，以及在 SAX 事件流中检测到 XPath 时要调用的相关模板处理器。

虽然本章的实例研究不涉及这项内容，但是很容易将每个转换类模块移至它自己的 ActiveX DLL 中，通过这种方式能够创建真正的可扩展系统。

2. 样式匹配

为了将 XPath 与模板处理器相关联，我们使用 CPattern 和 CPatternCollection。CPattern 用于存放字符串，除此之外，它还包含对模板处理器的引用（参见图 17-14）。

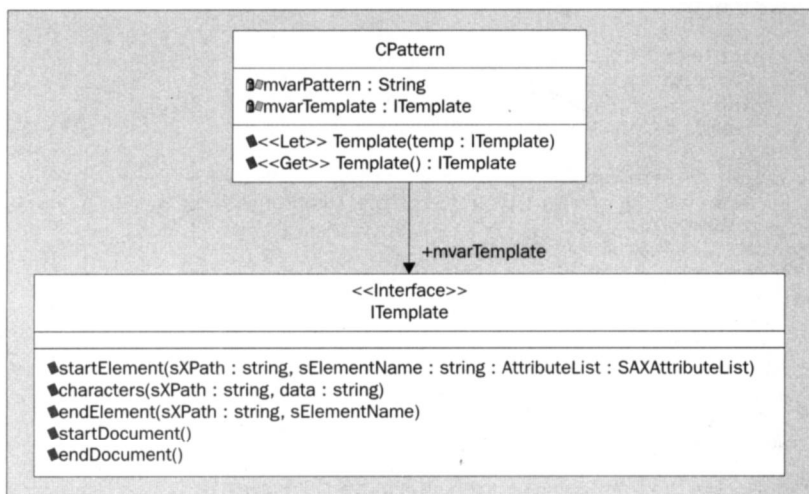


图 17-14

实际上，一个模板处理器能够负责处理一个或多个样式，反之亦然，但是在本章的实例研究中，一个样式只与一个模板相关。简而言之，CPattern 是与接口 ITemplate 相关的，这样就不需要具体的类——它为系统提供了扩展性。

与 CUser 类相似，CPatternCollection 类是由零个或多个 CPattern 类构成的（参见图 17-15）。

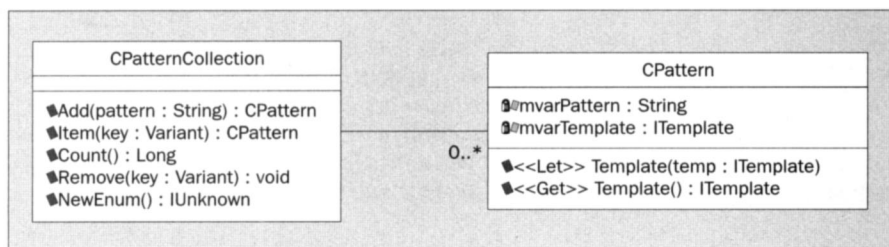


图 17-15

(1) 创建 CPattern 和 CPatternCollection

为了创建这些类，我们首先讨论类构造器。

如果你以前从未遇到过这个工具，可以通过附件菜单调用它——通过附件管理器菜单项

能够安装它。类构造器的含义非常明确，你能够利用它迅速创建具有特定属性和方法的类——不必过多担心输入错误。它还提供了许多其他有用的工具，例如：我们后面将要用到的迅速创建集合，但是其中的大多数工具都超出了本章的讨论范围。

为CPattern添加类模块，使用类构造器添加属性和方法。通过按键 Ctrl-S更新属性，你的CPattern类模块将类似于以下代码（注意，以下代码的格式已经重排，但是仅限于缩进）：

程序清单 17-19

```
Option Explicit

'local variable(s) to hold property value(s)
Private m_Pattern As String 'local copy
'local variable(s) to hold property value(s)
Private m_Template As ITemplate 'local copy

Public Property Set Template(ByVal vData As ITemplate)
    ' used when assigning an Object to the property, on the left side of a
    ' Set statement.
    ' Syntax: Set x.Template = Form1
    Set m_Template = vData
End Property

Public Property Get Template() As ITemplate
    ' used when retrieving value of a property, on the right side of an assignment.
    ' Syntax: Debug.Print X.Template
    Set Template = m_Template
End Property

Public Property Let Pattern(ByVal vData As String)
    ' used when assigning a value to the property, on the left side of
    ' an assignment.
    ' Syntax: X.Pattern = 5
    m_Pattern = vData
End Property

Public Property Get Pattern() As String
    ' used when retrieving value of a property, on the right side of an assignment.
    ' Syntax: Debug.Print X.Pattern
    Pattern = m_Pattern
End Property
```

需要注意的是，我在所有VB类模块中手工添加了“Option Explicit”行。通过添加该行，VB能够保证所有变量都是在引用/使用之前定义的。这是一种非常值得借鉴的方式，因为否则VB会在第一次使用时动态声明，这可能导致某些错误只能在运行时才能检测出来，如果你像我一样是个拼写水平一般的人，这简直就是一场噩梦。

转换引擎类将包含样式集合，而且将提供生成集合的AddPattern()方法（参见图17-16）。

添加类模块CSAXTransformEngine，并添加CPatternCollection类型的成员变量m_Patterns，以及间接生成集合的方法AddPattern：

程序清单 17-20

```
Private m_Patterns As New CPatternCollection

Function AddPattern(sXPath As String, oTemplate As ITemplate) As CPattern
```

```
Set AddPattern = m_Patterns.Add(sXPath, oTemplate)
End Function
```

转换引擎将使用样式集合判断是否要将处理源 XML 文档时产生的各种 SAX 事件转发给模板处理器。

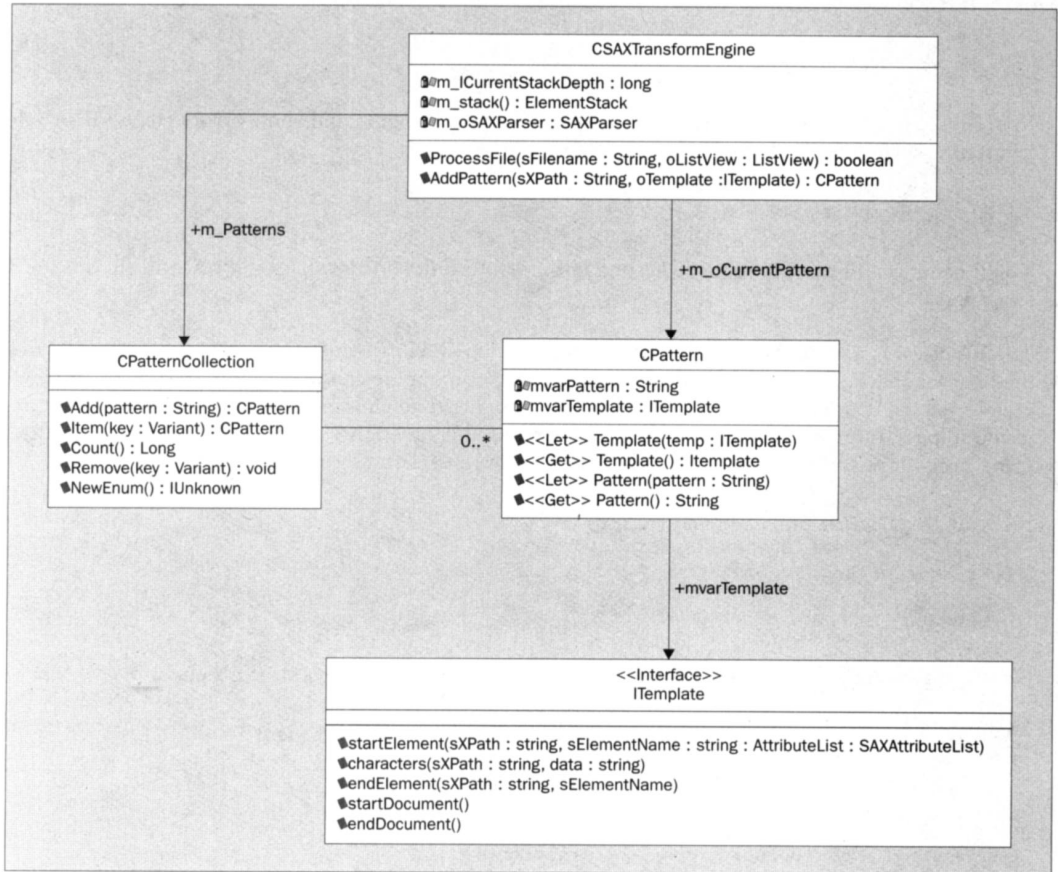


图 17-16

(2) 添加SAX解析器

下面，我们需要将SAX解析器集成在转换引擎类中，因此声明SAX解析器实例：

```
Dim WithEvents m_oSAXParser As SAXParser
```

为了使用户从转换引擎得到反馈，并显示错误信息，我们还要声明保存列表视图引用的变量：

```
Dim m_oListView As ListView
```

对于ListView，你需要添加对Microsoft Windows Common Controls 6.0组件的引用。

日志例程将访问ListView，引擎代码将访问该日志例程为用户提供反馈。使用转换引擎的代码能够传递它创建的ListView的引用，而后转换引擎将使用该引用。从视觉角度考虑，我为本章的实例研究选择了ListView，但是利用该函数封装日志，你很容易将它改为文件或其他更适当的

形式。下面显示了LogLine()例程的实现：

程序清单 17-21

```
Sub LogLine(sLineToLog As String)
    If Not (m_oListView Is Nothing) Then
        m_oListView.ListItems.Add , , sLineToLog
    End If
End Sub
```

下面，我们在CSAXTransformEngine类中添加ProcessFile()方法：

程序清单 17-22

```
Function ProcessFile(sFilename As String, oListView As ListView) As Boolean
    Dim bRC As Boolean
    Set m_oListView = oListView
    Set m_oSAXParser = New SAXParser

    bRC = m_oSAXParser.parseFile(sFilename)

    If bRC = False Then
        LogLine " File is not well formed"
        ProcessFile = False
    Else
        ProcessFile = True
    End If
End Function
```

ProcessFile()方法要求转换引擎解析指定的文件，并根据通过 AddPattern()方法添加在 CPatternCollection中的样式调用适当的模板处理器。该函数将 ListView保存在变量m_oListView中，以便LogFile()子例程访问（该子例程的作用是为用户提供反馈），然后，函数创建SAX解析器的实例，并要求它解析文件。如果出于某种原因XML解析失败，我们将通过调用LogLine()例程通知用户，该例程会在由oListView参数指定的ListView中添加项目。

为了检查代码的效果，我们更新窗体，使之包含 ListView。此时，我们将添加构成最终 GUI的所有控件，因此我们需要添加对以下控件的引用：

- Windows Tabbed Dialog Control 6.0。
- Microsoft Internet Controls。

删除我们以前在窗体中添加的命令按钮以及事件处理器代码。添加标签控件（使用 VB提供的缺省名称），它的高度是窗体高度的 2/3，在标签之下，添加两个命令按钮和两个文本编辑框（参见图17-17）。

对于窗体中的大多数控件，我都使用了VB提供的缺省名称，你完全可以更改这些名称，但是如果文本中用到这些名称，应该格外谨慎。

标有文本Split File的按钮的名称应该为SplitFile，旁边的编辑框的名称应该是FileToProcess。

标有文本Scan Directory的按钮的名称应该为ScanDirectory，旁边的编辑框的名称应该是ScanPath。

调出标签控件的属性，将第一个标签命名为 Web View，将第二个标签命名为 Log Messages，通过将标签计数改为2删除第三个标签。选择第一个标签，在其上放置 WebBrowser控件。在该控

件下，放置样式为 Dropdown List 的组合框（参见图 17-18）。

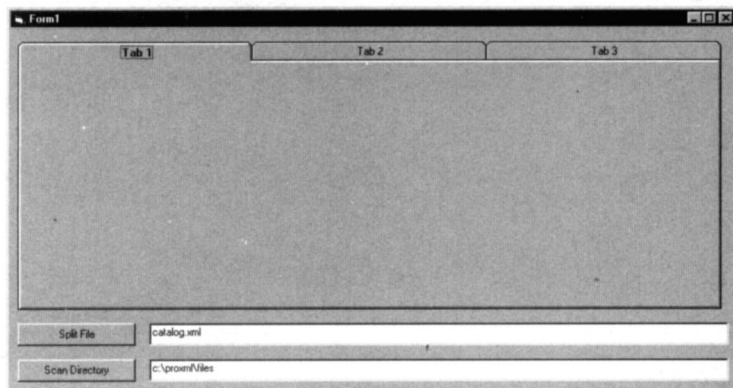


图 17-17

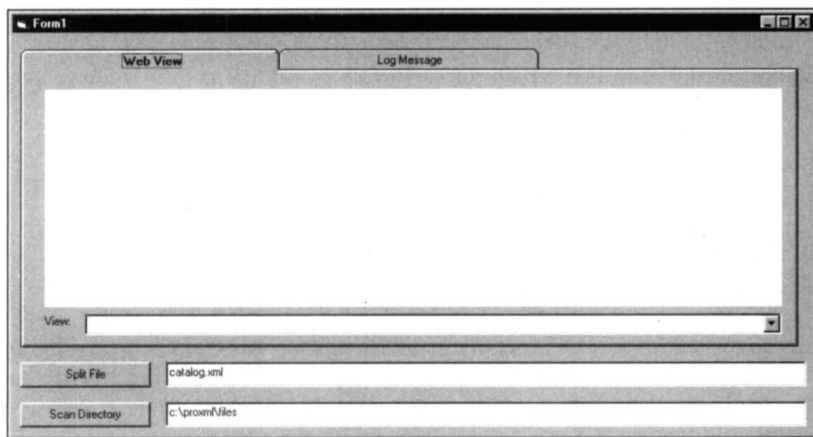


图 17-18

Web浏览器控件将用于查看我们创建的转换信息，组合框显示了转换引擎创建的所有文件的名称。

选择第二个标签，在其上放置 ListView 控件（不是 listbox），将它命名为 LogWindow（参见图 17-19）。

这是我们完整的 GUI。按钮 Split File 用于要求转换引擎处理单个 XML 文档。按钮 Scan Directory 用于在目录中扫描所有 XML 文档，并要求处理每个文件。

(3) 记录元素嵌套，创建 XPath

SAX 转换引擎使用 ElementStack 结构数组记录 SAX 事件流中的事件嵌套：

```
Private Type ElementStack
    sName As String
End Type
```

这个堆栈用于为每个转发给转换类模块处理器的 SAX 事件建立 XPath。

将该定义添加到 CSAXTransformEngine 的类模块中，在其中设置两个变量：

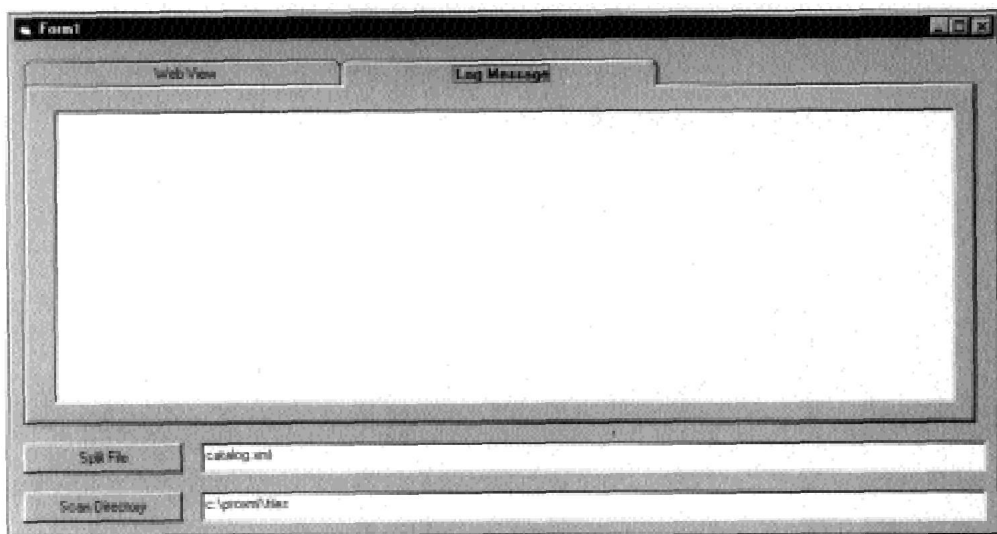


图 17-19

```
Dim m_lCurrentStackDepth As Long
Dim m_stack(100) As ElementStack
```

数组大小100是我随意选择的值，你可以根据自己的需求使用 ReDim 增加、删除或更改数组的大小。对于大多数XML文档来说，100应该是足够的。

当引擎首次启动时，堆栈的当前深度（ m_lCurrentStackDepth ）被设置为零，当产生 startElement 事件时，深度值增1：

程序清单 17-23

```
Private Sub m_oSAXParser_startElement(ByVal sName As String, _
                                     ByVal pAttributeList As SAXLib.ISAXAttributeList)
    ' Add this element to the stack

    m_stack(m_lCurrentStackDepth).sName = sName
    m_lCurrentStackDepth = m_lCurrentStackDepth + 1
End Sub
```

在增加堆栈深度之前，将元素的名称放入堆栈。

产生 endElement 事件时，堆栈深度减1：

```
Private Sub m_oSAXParser_endElement(ByVal sName As String)
    m_lCurrentStackDepth = m_lCurrentStackDepth - 1
End Sub
```

在事件流的任何时刻，当前 XPath 是通过列举堆栈中的内容获得的。我们使用 BuildXPath 函数执行这项操作，将 XPath 存入变量 m_sCurXPath：

程序清单 17-24

```
Dim m_sCurXPath as String
```

```

' Builds an XPath based upon the current element nesting

Sub BuildXPath()
    Dim l As Long

    m_sCurXPath = "/"

    For l = 0 To m_lCurrentStackDepth - 1
        m_sCurXPath = m_sCurXPath + m_stack(l).sName
        If l <> m_lCurrentStackDepth - 1 Then
            m_sCurXPath = m_sCurXPath + "/"
        End If
    Next l
End Sub

```

m_sCurXPath声明为全局变量，向模板处理器转发characters等事件时，将使用该变量。

为了在建立 XPath 的过程中，能够在日志窗口中看到 SAX 事件流中的 XPath，我们需要在 m_oSAXParser_startElement() 方法中调用 BuildPath() 函数，并使用 LogLine() 函数将 m_sCurXPath 写入日志窗口：

程序清单 17-25

```

Private Sub m_oSAXParser_startElement(ByVal sName As String, _
                                       ByVal pAttributeList As SAXLib.ISAXAttributeList)
    ' Add this element to the stack

    m_stack(m_lCurrentStackDepth).sName = sName
    m_lCurrentStackDepth = m_lCurrentStackDepth + 1

    BuildXPath
    LogLine "Start: " & m_sCurXPath

End Sub

```

为了看到代码的效果，双击 Split File 按钮，在事件处理器中添加以下代码：

程序清单 17-26

```

Private Sub SplitFile_Click()
    Dim oFP As CSAXTransformEngine

    ' Create the Transform Engine
    Set oFP = New CSAXTransformEngine

    oFP.ProcessFile Me.FileToProcess.Text, Me.LogWindow
End Sub

```

它创建转换类的实例，调用 ProcessFile() 方法，方法的参数包括：第一个文本编辑框中指定文件名，以及引擎记录 XPath 所用的 ListView 的引用。为了使转换引擎添加的日志消息可读，我们需要修改 Form_Load() 事件，将 ListView 视图设置为 lvwReport，并添加栏目：

程序清单 17-27

```

Private Sub Form_Load()
    LogWindow.View = lvwReport
    LogWindow.ColumnHeaders.Add , , "Message", Me.ScaleWidth
End Sub

```

如果我们不执行这项操作，窗体将使用缺省的图标视图，使得文本难以理解。

(4) SAX+XPath

运行窗体，指定 catalog.xml 文档的路径（你可以从站点 <http://www.wrox.com/> 获得本章的可下载代码，其中包含 catalog.xml 文档），然后点击 Split File 按钮。应该看到列出的所有 XPath（参见图 17-20）。

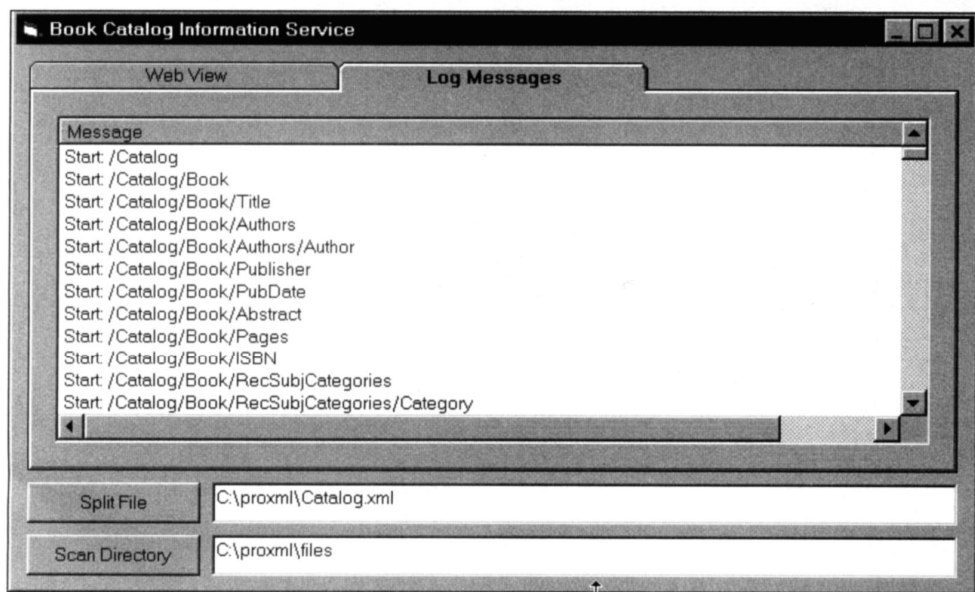


图 17-20

为了对事件流有更完整的认识，我们将修改 `m_oSAXParser_endElement()`，使之在遇到结束标记时显示元素的 XPath：

程序清单 17-28

```
Private Sub m_oSAXParser_endElement(ByVal sName As String)
    BuildXPath
    LogLine "End: " & m_sCurXPath

    m_lCurrentStackDepth = m_lCurrentStackDepth - 1
End Sub
```

我们还要添加字符事件处理器，用以记录 XPath 和字符数据：

程序清单 17-29

```
Private Sub m_oSAXParser_characters(ByVal sCharacters As String, _
    ByVal iLength As Long)
    LogLine "characters: " & m_sCurXPath & " : " & sCharacters
End Sub
```

再次运行窗体，并解析相同的文件，我们将对经 XPath 扩展的 SAX 事件流有更完整的认识

(参见图 17-21)。

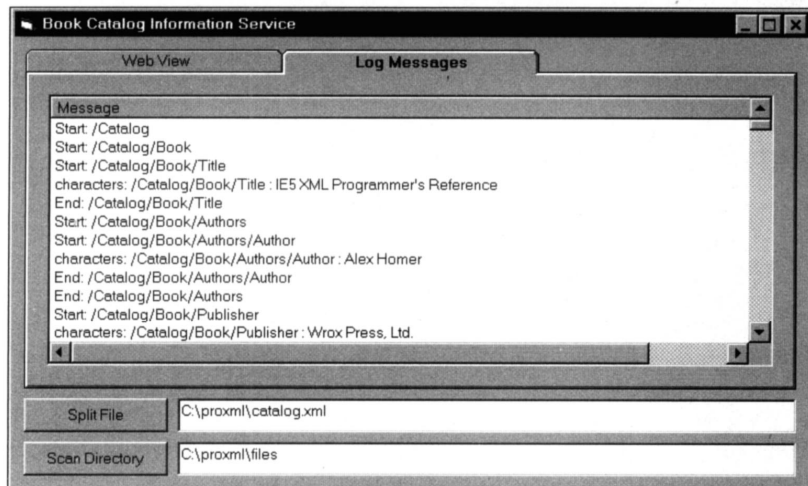


图 17-21

(5) 阶段总结

到目前为止，我们已经介绍了：

- 如何使用 ITemplate 接口在不知道具体的类的情况下调用模板处理器类模块函数。
- 如何使用 CPattern 和 CPatternCollection 将 XPath 与模板处理器相关联。
- 如何用 XPath 扩展 SAX 事件流。

下面我们要将所有内容组合在一起。

(6) MatchPatterns MatchPatterns() 方法通过搜索 CSAXTransformEngine 的 m_Patterns 变量中的 CPatternCollection 对象将 XPath 与模板处理器匹配。如果找到匹配的样式，就将 CPattern 对象放在成员变量 m_oCurrentPattern 中。每个 SAX 事件处理器都将检查该变量，除非它的值被设置为 Nothing，事件处理器会将事件转发给相关的模板处理器。

样式匹配代码将当前的 XPath 与 CPattern 对象的 Pattern 属性进行比较。如果它们精确匹配，或者 Pattern 属性为 “*”，表示匹配所有 XPath，则搜索成功。如果使用 “*” 样式，你应该确保它是列表中的最后一项，因为一旦遇到匹配项后，搜索立即停止。下面显示了该方法的完整代码，你应该将它添加到 CSAXTransformEngine 类中：

程序清单 17-30

```
Sub MatchPatterns()
    Dim Pattern As CPattern

    ' Build the current XPath
    BuildXPath

    ' Match current XPath against configured template handlers
```

```

For Each Pattern In m_Patterns
    If m_sCurXPath = Pattern.Pattern Or Pattern.Pattern = "*" Then
        Set m_oCurrentPattern = Pattern
        Exit Sub ' Searching ends!
    End If
Next Pattern

' Clear the current pattern

Set m_oCurrentPattern = Nothing
End Sub

```

代码调用我们前面看到的 BuildXPath() 函数，然后对于 m_Patterns 中的每个 CPattern 对象，判断它是否与当前的 XPath 匹配。如果匹配，记录样式，并退出例程。如果找不到匹配的样式，清除当前的样式变量。既然有了该函数，我们就可以修改 startElement 和 endElement 事件处理器中现有的对 BuildXPath() 的调用：

程序清单 17-31

```

Private Sub m_oSAXParser_startElement(ByVal sName As String, _
                                       ByVal pAttributeList As SAXLib.ISAXAttributeList)
    ' Add this element to the stack

    m_stack(m_lCurrentStackDepth).sName = sName
    m_lCurrentStackDepth = m_lCurrentStackDepth + 1

    MatchPatterns

    If Not (m_oCurrentPattern Is Nothing) Then
        m_oCurrentPattern.Template.startElement m_sCurXPath, sName, pAttributeList
    End If
End Sub

Private Sub m_oSAXParser_endElement(ByVal sName As String)
    MatchPatterns

    If Not (m_oCurrentPattern Is Nothing) Then
        m_oCurrentPattern.Template.endElement m_sCurXPath, sName
    End If
    m_lCurrentStackDepth = m_lCurrentStackDepth - 1
End Sub

```

这两个数据处理器不再直接记录 XPath，如果设置了 m_oCurrentPattern 对象引用，它们会将事件转发给与当前样式相关的模板处理器。如果未设置当前样式，则不采取任何操作。字符事件处理器中也要进行类似的修改，但是不需要调用 MatchPatterns() 方法，因为我们知道只有前两个事件中的元素嵌套发生了变化，因此已经设置了当前路径：

程序清单 17-32

```

Private Sub m_oSAXParser_characters(ByVal sCharacters As String, _
                                     ByVal iLength As Long)
    If Not (m_oCurrentPattern Is Nothing) Then
        m_oCurrentPattern.Template.characters m_sCurXPath, sCharacters
    End If
End Sub

```


在编译新代码之前，我们还需要在 CSAXTransformClass 类模块中添加 m_oCurrentPattern 变量定义：

程序清单 17-33

```
Dim m_stack(100) As ElementStack
Dim m_oListView As ListView
Dim m_oCurrentPattern As CPattern
Dim m_sCurXPath As String
```

从理论角度讲，我们已经实现了基本的转换引擎。为了查看它的效果，修改 SplitFile_Click()，创建 CXMLCatalogTransform 类的实例，使用 AddPattern() 方法在转换引擎中添加 XPath 样式“/Catalog/Book/Title”，并将 CXMLCatalogTransform 对象的引用作为模板处理器传递：

程序清单 17-34

```
Private Sub SplitFile_Click()
    Dim oFP As CSAXTransformEngine

    ' Create the Transform Engine
    Set oFP = New CSAXTransformEngine

    ' Create the XML catalog transform
    Dim oTransform As New CXMLCatalogTransform

    oFP.AddPattern "/Catalog/Book/Title", oTransform
    oFP.ProcessFile Me.FileToProcess.Text, Me.LogWindow
End Sub
```

然后，修改 CXMLCatalogTransform ITemplate_characters() 事件处理器，使之仅显示数据，而不显示 XPath，因为我们知道 XPath 总是“/Catalog/Book/Title”——它是在调用 AddPattern() 时指定的：

```
Private Sub ITemplate_characters(sXPath As String, sData As String)
    MsgBox sData
End Sub
```

运行项目，将在消息框中看到 XML 文档中每本书的书名。转换引擎提供了过滤 XML 原文档的功能，只有具有特定 XPath 的事件才转发给我们定义的模板处理器。正如我们通过 CXMLCatalogTransform 类中的字符处理器看到的，它能够简化代码，并且使你对这种技术的灵活性有所认识。

(7) startDocument 和 endDocument

为了完成 CSAXTransformEngine，我们将实现 startDocument 和 endDocument SAX 事件，并采用类似于 startElement 的方法将它们转发给模板处理器。

处理到 XML 文档的开始处时，SAX 解析器将调用 startDocument 事件处理器。如果找到匹配项，代码调用 MatchPatterns() 方法设置 m_oCurrentPattern 变量。我们知道匹配项总是“/”（文档的根），因为此时堆栈不包含任何项。如果找到样式，将调用相关的模板处理器的 startDocument 处理器：

程序清单 17-35

```
Private Sub m_oSAXParser_startDocument()
    MatchPatterns

    If Not (m_oCurrentPattern Is Nothing) Then
        If Not (m_oCurrentPattern.Template Is Nothing) Then
            m_oCurrentPattern.Template.startDocument
        End If
    End If
End Sub
```

endDocument方法也大同小异，但是它调用模板处理器的 endDocument方法：

程序清单 17-36

```
Private Sub m_oSAXParser_endDocument()
    MatchPatterns

    If Not (m_oCurrentPattern Is Nothing) Then
        If Not (m_oCurrentPattern.Template Is Nothing) Then
            m_oCurrentPattern.Template.endDocument
        End If
    End If
End Sub
```

3. 从Users.XML加载订阅者信息

到目前为止，我们介绍了 BCIS 的基本设计，以及转换引擎的实现。既然已经编写了转换引擎，只需要少量的编码，我们就能够使用它迅速地创建 BCIS。

第一个需求是将订阅者信息加载至内存。该信息用于通知 BCIS 必须将图书目录信息发送给谁，以及每个订阅者对哪类图书信息感兴趣。下面显示了 XML 用户文件（users.xml 文档同样可以从前面列出的站点中下载）：

程序清单 17-37

```
<?xml version="1.0" ?>
<!-- This file defines the register users for the book notification service -->
<Users>

    <User>
        <Name>Richard Anderson</Name>
        <OutputFile>c:\proxml\RJA.HTML</OutputFile>
        <Email>rja@arpsolutions.demon.co.uk</Email>
        <DeliveryFormat>HTML</DeliveryFormat>
        <Category1>XML</Category1>
        <Category2>ATL</Category2>
    </User>
    ...

</Users>
```

正如前面所讨论的，CLoadUserInfo 会将该文件加载至 CUserCollection，使得每个用户都由一个 Cuser 对象表示（参见图 17-22）。

与前面对于 CPattern 和 CPatternCollection 的处理类似，使用类构造器添加 CUser 和 CUserCollection 类模块。如上图所示，对于用户类模块，你需要添加 String 类型的属性 Email、

DeliveryFormat、OutputFile、Category1、Category2和UserName。对于CUserCollection类模块，修改CUserCollection的Add()方法，从函数原型中删除 Category1、Category2、OutputFile、DeliveryFormat和Email参数，以及函数中的任何引用，修改后的代码将如下所示：

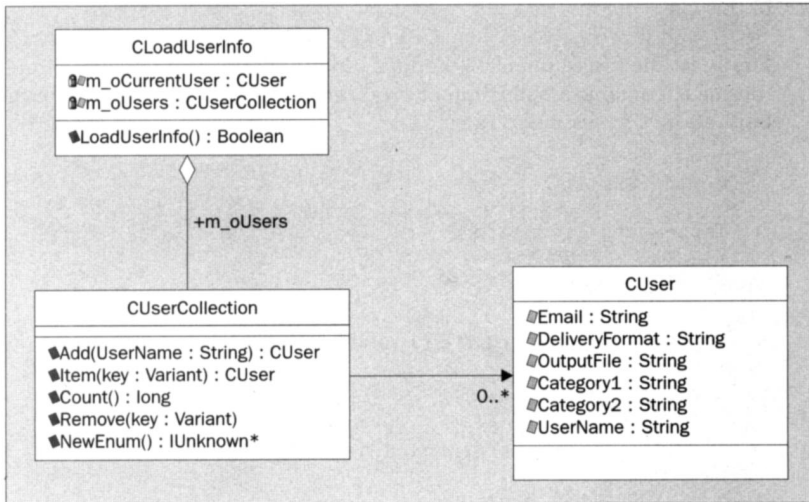


图 17-22

程序清单 17-38

```

Public Function Add(UserName As String, Optional sKey As String) As CUser
    'create a new object
    Dim objNewMember As CUser
    Set objNewMember = New CUser

    'set the properties passed into the method
    objNewMember.UserName = UserName
    If Len(sKey) = 0 Then
        mCol.Add objNewMember
    Else
        mCol.Add objNewMember, sKey
    End If

    'return the object created
    Set Add = objNewMember
    Set objNewMember = Nothing
End Function

```

我们之所以删除所有参数，是因为最初创建 CUser对象时，这些参数并非都是可用的，由于它们是CUser的属性，因此通过类构造器添加它们。当 XPath与ITemplate_characters()方法中的输入流匹配时，将设置相应的属性。例如，当匹配 XPath “/Users/User/Email” 时，我们将设置Email属性。当XPath为 “/Users/User/Name” 时，我们将创建CUser对象，同时设置UserName属性。

添加类模块CLoadUserInfo，但是不要使用类构造器，因为我们已经创建了该类。

首先，在类模块的顶端添加以下行，声明它实现了 ITemplate接口，它包含用户集合，以及

跟踪m_oCurrentUser中构造的当前用户（订阅者）配置文件：

程序清单 17-39

```
Option Explicit

Implements ITemplate
Dim m_oCurrentUser As CUser
Private m_oUsers As New CUserCollection
```

然后，添加LoadUserInfo()方法：

程序清单 17-40

```
Function LoadUserInfo() As Boolean
    Dim oFP As New CSAXTransformEngine
    Dim oPattern As CPattern

    ' Match all elements
    Set oPattern = oFP.AddPattern("*", Me)
    LoadUserInfo = oFP.ProcessFile("users.xml", Nothing)
End Function
```

在创建转换引擎之前，注册样式和模板处理器（模板处理器就是 CLoadUserInfo类本身），而后请求处理 Users.XML。与前面的模板处理器不同，我们曾使用了通配符“*”样式。这意味着每个SAX事件都会添加XPath，并转发给模板处理器。由于要处理的XML文档非常简单，因此我们采用这种方法，否则要求多个类模块加载用户信息就矫枉过正了。

下面，按照前面介绍的方法为 ITemplate添加空的事件处理器，然后参照以下代码修改 ITemplate_characters()方法：

程序清单 17-41

```
Private Sub ITemplate_characters(sXPath As String, sData As String)
    Select Case sXPath
        Case "/Users/User/Name"
            Set m_oCurrentUser = m_oUsers.Add(sData)

        Case "/Users/User/OutputFile"
            m_oCurrentUser.OutputFile = sData

        Case "/Users/User/Category1"
            m_oCurrentUser.Category1 = sData

        Case "/Users/User/Category2"
            m_oCurrentUser.Category2 = sData

        Case "/Users/User/Email"
            m_oCurrentUser.Email = sData

        Case "/Users/User/DeliveryFormat"
            m_oCurrentUser.DeliveryFormat = sData
    End Select
End Sub
```

当匹配XPath “/Users/User/Name”时，ITemplate_characters()方法通过调用CUserCollection的Add()方法创建新的CUser对象，将返回对象设置为m_oCurrentUser。如果匹配其他XPath，则

将字符数据拷贝至当前用户对象的 `m_oCurrentUser` 属性。

为了使其他对象能够访问加载的用户信息，我们添加名为 `Users` 的属性：

```
Property Get Users() As CUserCollection
    Set Users = m_oUsers
End Property
```

为了测试 `CLoadUserInfo` 类，添加 `Scan Directory` 按钮的事件处理器，并增加以下代码：

程序清单 17-42

```
Private Sub ScanDirectory_Click()
    Dim oUser As CUser
    Dim oLoadUsers As New CLoadUserInfo
    oLoadUsers.LoadUserInfo

    For Each oUser In oLoadUsers.Users
        Me.LogWindow.ListItems.Add , , "User=" & oUser.UserName
    Next oUser
End Sub
```

运行项目并点击按钮，你将看到日志窗口中列出了所有用户名（参见图 17-23）。

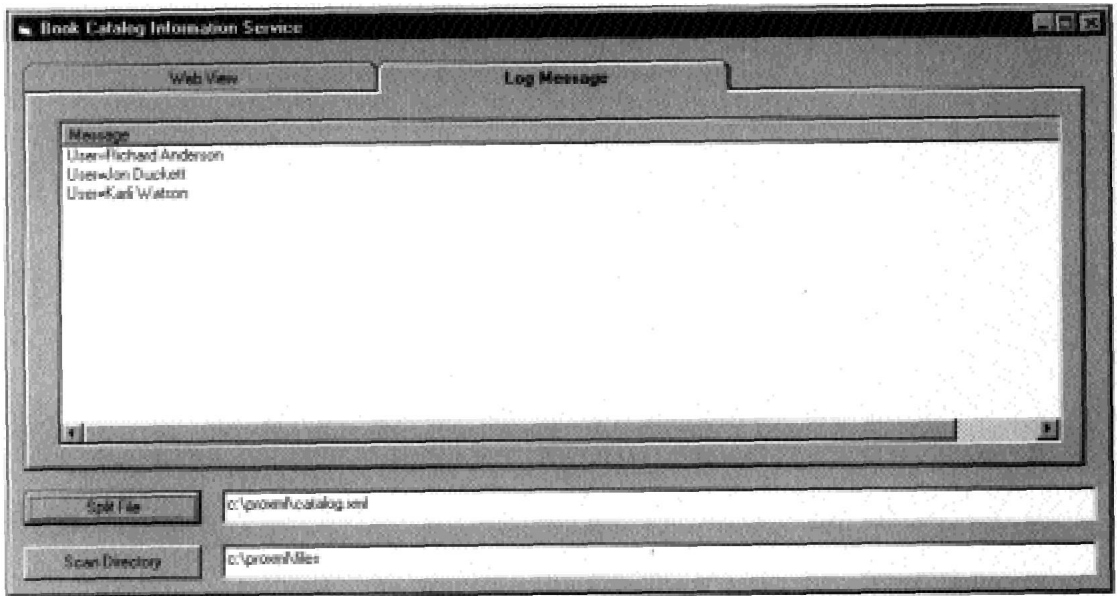


图 17-23

如果你非但没有看到用户名列表，反而遇到了错误，你需要再次检查 `CLoadUserInfo` 解析的文件 `Users.xml` 是否位于当前目录下，因为代码采用的是相对路径：

```
LoadUserInfo = oFP.ProcessFile("users.xml", Nothing)
```

另外，你也可以通过如下方式指定路径：

```
LoadUserInfo = oFP.ProcessFile("c:\proxml\users.xml", Nothing)
```

现在，你已经完成了用户信息的加载，下一步是根据每个用户选择的分类和喜欢的输出格

式——XML或HTML——转换XML源文档。为此，我们需要回到前面实现的 CXMLCatalogTransform和CHTMLCatalogTransform类，编写真正执行转换过程的代码。

4. XML 到XML转换

图17-24显示了XML到XML转换类CXMLCatalogTransform。

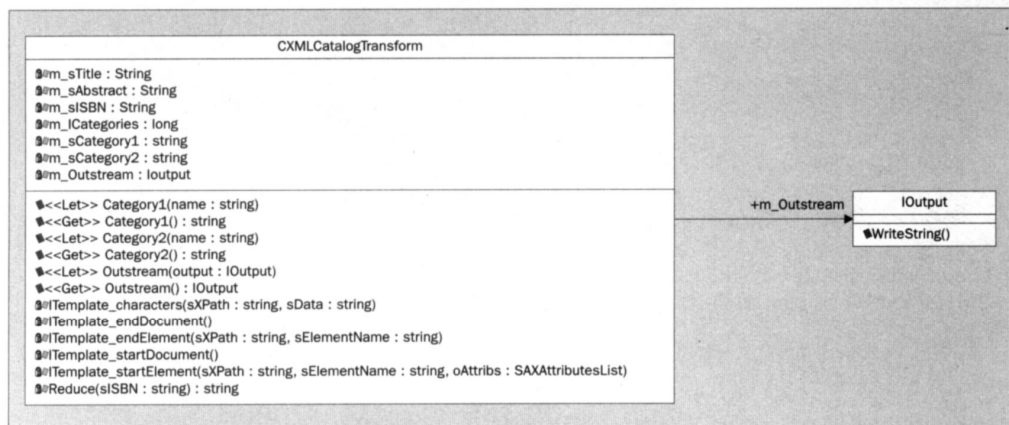


图 17-24

属性Category1和Category2用于告知转换类应该匹配源XML文档中的哪两个图书分类。任何与这些分类不符的图书都将被忽略。与这些分类匹配的图书将写入输出流。

修改我们刚才创建的 CXMLCatalogTransform模块的代码，如下所示声明成员变量和属性访问方法（我们还可以使用类构造工具完成该任务）：

程序清单 17-43

```
Option Explicit

Implements ITemplate

' Define temporary storage for book info

Dim m_sTitle As String
Dim m_sAbstract As String
Dim m_sISBN As String
Dim m_lCategories As Long
Private m_sCategory1 As String
Private m_sCategory2 As String
Private m_Outstream As IOutput

Public Property Set Outstream(ByVal vData As IOutput)
    Set m_Outstream = vData
End Property

Public Property Get Outstream() As IOutput
    Set Outstream = m_Outstream
End Property
```



```

Public Property Let Category2(ByVal vData As String)
    m_sCategory2 = vData
End Property

Public Property Get Category2() As String
    Category2 = m_sCategory2
End Property

Public Property Let Category1(ByVal vData As String)
    m_sCategory1 = vData
End Property

Public Property Get Category1() As String
    Category1 = m_sCategory1
End Property

```

下一页显示了ITemplate方法新的实现方式。

调用ITemplate_characters事件处理器时，私有变量 m_sTitle、m_sAbstract和m_sISBN用于保存来自XML源文档的详细信息：

程序清单 17-44

```

Private Sub ITemplate_characters(sXPath As String, sData As String)
    ' Check filters
    If sXPath = "/Catalog/Book/RecSubjCategories/Category" Then
        If sData = m_sCategory1 Or sData = m_sCategory2 Then
            m_lCategories = m_lCategories + 1
        End If
    End If

    If sXPath = "/Catalog/Book/Title" Then
        m_sTitle = sData
    End If

    If sXPath = "/Catalog/Book/ISBN" Then
        m_sISBN = sData
    End If

    If sXPath = "/Catalog/Book/Abstract" Then
        m_sAbstract = sData
    End If
End Sub

```

例程检查的第一个XPath是“/Catalog/Book/RecSubjCategories/Category”。如果找到相应的内容，例程将使用当前的过滤器检查图书的推荐分类。如果找到匹配项，则变量 m_lCategories增一。每本书可能有多个推荐的分类，因此变量 m_lCategories用于确定针对当前的图书找到多少个匹配项。当ITemplate_endElement()方法检测到图书的结束时，如果变量 m_lCategories不等于零，将图书信息写入输出流：

程序清单 17-45

```

Private Sub ITemplate_endElement(sXPath As String, sElementName As String)
    If sXPath = "/Catalog/Book" Then
        If m_lCategories <> 0 Then
            m_Outstream.WriteString "<Book>"
            m_Outstream.WriteString "<Title>" & m_sTitle & "</Title>"

```

```

m_Outstream.WriteString "<AmazonURL>" & _
    "http://www.amazon.com/exec/obidos/ASIN/" & Reduce(m_sISBN) & _
    "</AmazonURL>"
m_Outstream.WriteString "<Abstract>" & m_sAbstract & "</Abstract>"
m_Outstream.WriteString "</Book>"
End If
End If
End Sub

```

对于以上代码，特别需要说明的是我们使用简单的 WriteString() 语句构建输出的 XML。这是由于我们使用 SAX 解析器处理源文档，这一点非常有意义。如果我们选择通过 DOM 对象创建输出文档，对于大型的输出文件，我们可能会遇到内存问题，而且由于创建 DOM 对象会产生相关的额外开销，因此要花费更多的处理器时间。总之，这种方法非常适于 BCIS，而且它比使用 DOM 对象快大约 3 倍。

该代码不检查 “&” 等 XML 的保留字符；对于需要实际应用的代码，你应该进行这项检查。

对于图书分类匹配，我们本来可以使用 Boolean 值，而不是 long，但是我之所以使用 long 是因为可以允许用户为匹配操作定义分类的最小值。输出 Amazon 的 URL 时调用的 Reduce() 函数用于保证 ISBN 不包含连字符——输入流的 ISBN 中包括连字符，但是这与 Amazon 网站点的 URL 不一致。以下代码用于删除输入串中的连字符：

程序清单 17-46

```

Function Reduce(ISBN As String) As String
    Dim l As Long
    Dim s As String

    For l = 1 To Len(ISBN)
        If Mid(ISBN, l, 1) <> "-" Then
            s = s + Mid(ISBN, l, 1)
        End If
    Next l

    Reduce = s
End Function

```

为 Amazon 添加的链接可以移至子例程中。如果我们决定支持多广告商，就应该这样做，但是对于单一的项，内嵌的代码也是可行的。

处理新的图书时，我们需要重置 m_lCategories 变量，因此当我们在 ITemplate_startElement 方法中检测到 XPath “/Catalog/Book/RecSubjCategories” 时，我们将重新设置 m_lCategories 变量。如果我们忘记重置计数器，第一次匹配成功之后的所有图书都将出现在输出流中——这可不是我们的订阅者所期望的！由于所有的图书分类都定义为 <RecSubjCategories> 元素的子元素，因此在 ITemplate_startElement 方法中重新设置值是一种很好的方式，虽然我们也可以在 endElement 事件处理器中将值重新设置为零。下面显示了将值归零的方法：

程序清单 17-47

```

Private Sub ITemplate_startElement(sXPath As String, _
    sElementName As String, _

```

```

                                oAttribs As SAXLib.ISAXAttributeList)
' If recommended subject categories start element:
' reset matched categories count

If sXPath = "/Catalog/Book/RecSubjCategories" Then
    m_lCategories = 0
End If
End Sub

```

如果采用基于 SAX 的方式过滤 XML 数据，只能从输入流中缓存各种数据元素，然后将它们按照原有的顺序写入输出流。它不可能在输入流中向回搜索（访问以前发生的事件），如果我们希望在检查与过滤器匹配的图书分类之前将元素写入输出流，就不得不删除一些已经写入的内容——这不是一种可取的方法。你也可以为每本书创建一个输出，但是首先将它写入临时变量（或者很小的 DOM），一旦与当前的过滤器匹配，再将它写入真正的文件，但是我们采用的方法对于本章的实例研究来说非常恰当。

前面显示的 `m_Outstream` 变量是指向接口 `IOutput` 的引用，`IOutput` 只有一个方法 `WriteString()`。通过引入类似于 `ITemplate` 中使用的重构特性，我们的转换类能够将输出写入任何介质：文件、内存、套接字等。

添加名为 `IOutput` 的类模块，按照以下代码修改类模块：

```

Option Explicit

Sub WriteString(sData As String)

End Sub

```

稍后我们将实现该接口。

任何 XML 文档都只能有一个根元素。到目前为止，我们只讨论了写入 `<Books>` 元素的代码，现在我们需要创建根元素。为此，我们将实现 `startDocument` 和 `endDocument` 事件处理器。

下面显示了 `CXMLCatalogTransform` 的 `startDocument` 事件处理器：

程序清单 17-48

```

Private Sub ITemplate_startDocument()
    m_Outstream.WriteString "<?xml version='1.0' ?>"
    m_Outstream.WriteString "<Catalog>"
    m_Outstream.WriteString "<Notes>"
    m_Outstream.WriteString "Filters: " & m_sCategory1 & " , " & m_sCategory2
    m_Outstream.WriteString "</Notes>"
    m_Outstream.WriteString "<Books>"
End Sub

```

首先，写入 XML 声明和根起始标记，以及详细描述用于创建文件的过滤器的 `<Notes>` 元素。最后一行写入 `<Books>` 元素的起始标记，它包含匹配的所有图书。

到达源 XML 文档结束时，激活 `endDocument` 事件处理器，因此代码将写入 `<Books>` 元素的结束标记以及根元素 `<Catalog>` 的结束标记：

程序清单 17-49

```

Private Sub ITemplate_endDocument()
    m_Outstream.WriteString "</Books>"

```

```
m_Outstream.WriteString "</Catalog>"
End Sub
```

完成XML转换类之后，我们将在主窗体中添加方法 PerformXMLToXMLTransform ()，它将针对特定的用户调用XML到XML的转换：

程序清单 17-50

```
Sub PerformXMLToXMLTransform(oUser As CUser, sFilename As String)
    Dim oFP As CSAXTransformEngine
    Dim oPattern As CPattern
    Dim oTemplate As CXMLCatalogTransform
    Dim oFSO As New FileSystemObject

    ' Create the Transform Engine
    Set oFP = New CSAXTransformEngine

    ' Create the Template Class
    Set oTemplate = New CXMLCatalogTransform
    Set oTemplate.Outstream = Me

    ' Set the filters for this users transformation
    oTemplate.Category1 = oUser.Category1
    oTemplate.Category2 = oUser.Category2

    ' Match all elements
    Set oPattern = oFP.AddPattern("*", oTemplate)

    Set m_oOutStream = oFSO.CreateTextFile(oUser.OutputFile)
    oFP.ProcessFile sFilename, Me.LogWindow

    Me.Combo1.AddItem oUser.OutputFile

    m_oOutstream.Close
End Sub
```

方法以CUser对象和要处理的XML输入文件作为参数。CUser对象提供转换的详细信息（例如：格式和输出文件名）。以上代码创建转换引擎，创建模板处理器，然后将模板处理器传递给前面讨论的输出接口（IOutput），所有输出都将写入该接口：

```
Set oTemplate.Outstream = Me
```

正如以上代码所示，IOutput接口是由窗体实现的，因此我们必须在主窗体的顶端添加以下行：

```
Implements IOutput
```

而且IOutput要实现写入字符串的方法：

```
Private Sub IOutput_WriteString(sData As String)
    m_oOutstream.Write sData
End Sub
```

变量m_oOutstream定义为TextStream对象，它是Microsoft Scripting Runtime类型库的一部分，因此你还需要在项目中添加对该类型库的引用。添加之后，在窗体代码模块的顶端声明该对象：

```
Dim m_oOutstream As Scripting.TextStream
```

在代码调用转换引擎处理函数之前要设置该对象：

```
Set m_oOutStream = oFSO.CreateTextFile(oUser.OutputFile)
oFP.ProcessFile sFilename, Me.ListView1
```

输出文件名是由用户对象指定的，因此 XML 文档中的每个用户要定义一个用户对象。

PerformXMLToXMLTransform() 中的最后一部分代码是构造 XML 到 XML 转换类的分类过滤器，这是通过从用户对象中拷贝分类实现的：

```
oTemplate.Category1 = oUser.Category1
oTemplate.Category2 = oUser.Category2
```

由于我们能够以每个用户为基础通过调用 PerformXMLToXMLTransform() 方法执行 XML 转换，对于每个希望以 XML 格式接收数据的订阅者，我们可以修改 SplitFile 点击处理器执行转换过程：

程序清单 17-51

```
Private Sub SplitFile_Click()
    Dim oUser As CUser

    Dim oLoadUsers As New CLoadUserInfo
    oLoadUsers.LoadUserInfo

    For Each oUser In oLoadUsers.Users
        Me.LogWindow.ListItems.Add , , "User=" & oUser.UserName
        If oUser.DeliveryFormat = "XML" Then
            PerformXMLToXMLTransform oUser, Me.FileToProcess.Text
        End If
    Next oUser
End Sub
```

如果运行代码，你会发现它只为将分发格式设置为 XML 的订阅者执行 XML 转换过程，而且 Web 视图标签中的组合框将列出已经创建的所有文件（参见图 17-25）。

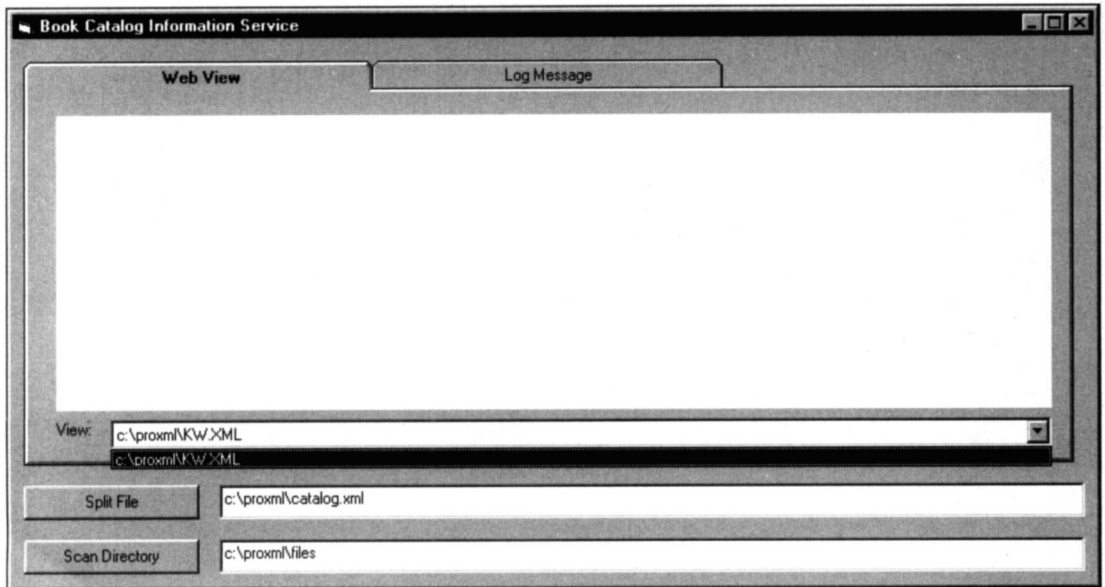


图 17-25

该组合框是通过 PerformXMLToXMLTransform() 函数添加的：

```
Me.Combo1.AddItem oUser.OutputFile
```

组合框中显示的输出文件名是由 Users.XML 中的 OutputFile 元素定义的：

程序清单 17-52

```
<?xml version="1.0" ?>
<!-- This file defines the register users for the book notification service -->
<Users>

  <User>
    <Name>Richard Anderson</Name>
    <OutputFile>c:\proxml\RJA.HTML</OutputFile>
    <Email>rja@arpsolutions.demon.co.uk</Email>
    <DeliveryFormat>HTML</DeliveryFormat>
    <Category1>ASP</Category1>
    <Category2>ASP+</Category2>
  </User>
```

然而，如果你从组合框中选择文件名，什么也不会发生，因为我们尚未实现改变处理器。为此，通过在表单中添加以下代码实现 combobox 的 click 处理器，它用于告知 webbrowser 控件定位至指定的文件：

```
Private Sub Combo1_Click()
    Me.WebBrowser1.Navigate Me.Combo1.Text
End Sub
```

如果我们通过点击 Split File 按钮并选择 KW.XML 项执行转换，我们将看到 Web 浏览器控件中显示 XML 文档的内容（参见图 17-26）。

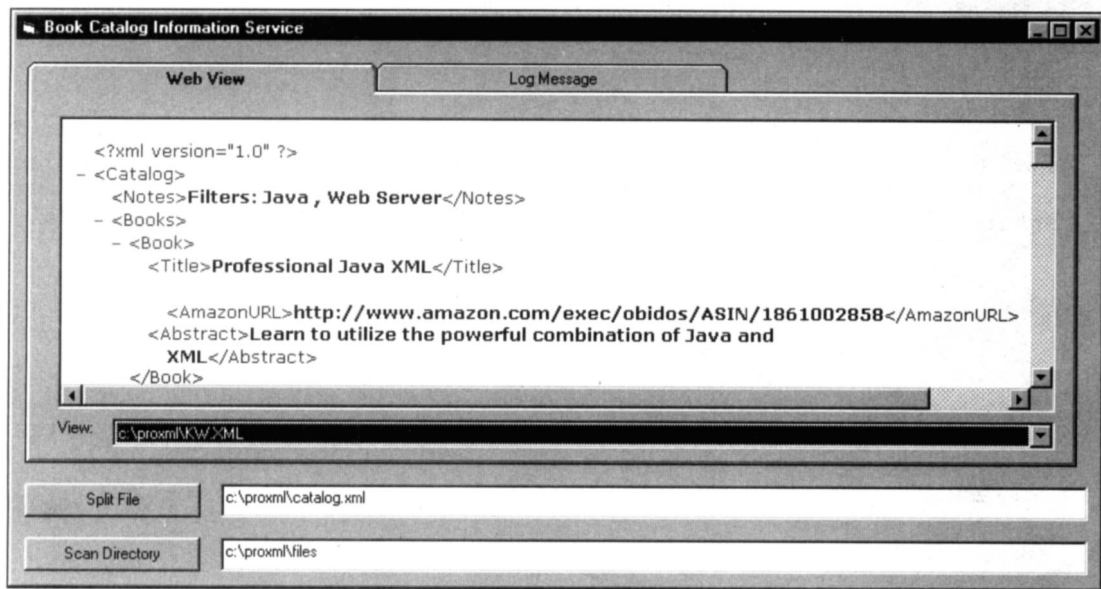


图 17-26

5. XML到HTML的转换

XML到HTML的转换的实现方式与XML到XML的转换基本相同，它们唯一的差别在于这次的输出格式是HTML。因此，我只是简单地列出代码，而不做详细的解释。

删除CHTMLCatalogTransform类中现有的代码，并用CXMLCatalogTransform中的代码进行替换。事件处理器startDocument、endDocument和endElement中需要修改的行已经突出显示：

程序清单 17-53

```
Private Sub ITemplate_endElement(sXPath As String, sElementName As String)
    If sXPath = "/Catalog/Book" Then
        If m_lCategories <> 0 Then
            m_Outstream.WriteString "<H3>" & m_sTitle & "</H3>"
            m_Outstream.WriteString "<P>" & m_sAbstract & "</P>"
            m_Outstream.WriteString "<A HREF=" & "" & _
                "http://www.amazon.com/exec/obidos/ASIN/" & _
                Reduce(m_sISBN) & "" & ">View on Amazon.com</A>"
        End If
    End If
End Sub

Private Sub ITemplate_startDocument()
    m_Outstream.WriteString "<HTML>"
    m_Outstream.WriteString "<H1>Catalog Of New Books</H1>"
    m_Outstream.WriteString "<P>"
    m_Outstream.WriteString "This file has been created based upon your " & _
        "filters of : " & m_sCategory1 & " , " & m_sCategory2
    m_Outstream.WriteString "</P>"

    m_Outstream.WriteString "<H2>Books</H2>"
End Sub

Private Sub ITemplate_endDocument()
    m_Outstream.WriteString "</HTML>"
End Sub
```

为了将XML转换为HTML，我们将在窗体中添加用于创建HTML输出文件的辅助函数。为此，拷贝PerformXMLToXMLTransform()函数，将它的名称改为PerformXMLToHTMLTransform，并将所有对CXMLCatalogTransform的引用改为对CHTMLCatalogTransform的引用：

程序清单 17-54

```
Sub PerformXMLToHTMLTransform(oUser As CUser, sFilename As String)
    Dim oFP As CSAXTransformEngine
    Dim oPattern As CPattern
    Dim oTemplate As CHTMLCatalogTransform
    Dim oFSO As New FileSystemObject

    ' Create the Transform Engine
    Set oFP = New CSAXTransformEngine

    ' Create the Template Class
    Set oTemplate = New CHTMLCatalogTransform
    Set oTemplate.OutStream = Me
    ' Set the filters for this users transformation
```

```

oTemplate.Category1 = oUser.Category1
oTemplate.Category2 = oUser.Category2

' Match all elements
Set oPattern = oFP.AddPattern("*", oTemplate)

Set m_oOutStream = oFSO.CreateTextFile(oUser.OutputFile)
oFP.ProcessFile sFilename, Me.LogWindow

Me.Combo1.AddItem oUser.OutputFile

m_oOutStream.Close
End Sub

```

现在，我们已经生成了转换类和辅助函数，可以修改 SplitFile_Click()方法，使之能够根据订阅者的配置信息选择适当的转换引擎：

程序清单 17-55

```

Private Sub SplitFile_Click()
    Dim oUser As CUser
    Dim oLoadUsers As New CLoadUserInfo
    oLoadUsers.LoadUserInfo

    For Each oUser In oLoadUsers.Users
        Me.LogWindow.ListItems.Add , , "User=" & oUser.UserName
        If oUser.DeliveryFormat = "XML" Then
            PerformXMLToXMLTransform oUser, Me.FileToProcess.Text
        Else
            PerformXMLToHTMLTransform oUser, Me.FileToProcess.Text
        End If
    Next oUser
End Sub

```

如果我们再次运行程序，你会发现现在能够查看订阅者的 HTML输出（参见图 17-27）。

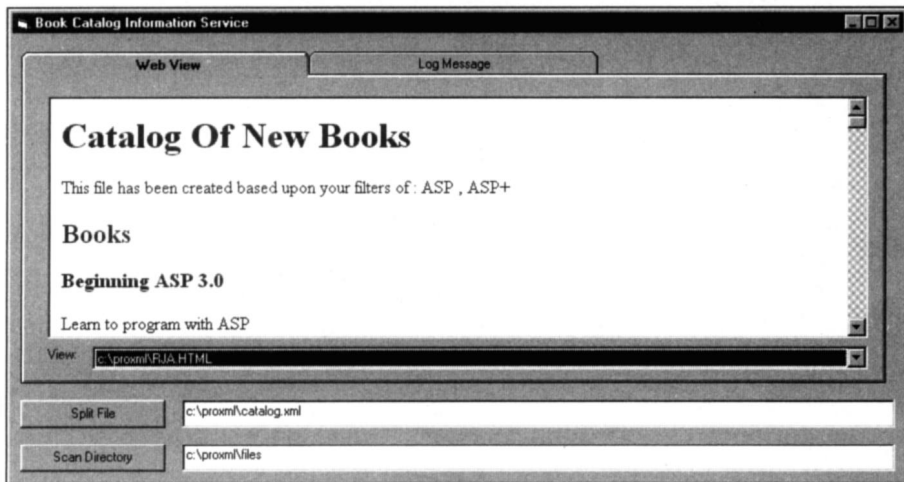


图 17-27

既然转换引擎已经就绪，BCIS中尚未实现的部分只剩下直接邮件分发组件以及自动扫描目录，并检测和处理多个XML文档。

6. 扫描目录功能

为了实现扫描目录功能，我们将使用 Microsoft Scripting Runtime提供的组件。

双击Scan Directory按钮，按照以下代码修改点击处理器：

程序清单 17-56

```
Private Sub ScanDirectory_Click()  
    Dim oFSO As New FileSystemObject  
    Dim oFolder As Folder  
  
    Set oFolder = oFSO.GetFolder(Me.ScanPath)  
  
    Dim oFile As File  
    Dim bRC As Boolean  
    Dim sDestFile As String  
  
    ' Process each file in the directory  
  
    Me.LogWindow.ListItems.Add , , "Scanning directory " & Me.ScanPath  
  
    For Each oFile In oFolder.Files  
        If InStr(oFile.Name, ".xml") <> 0 Then  
            Me.LogWindow.ListItems.Add , , oFile.Name  
        End If  
    Next oFile  
End Sub
```

代码利用FileSystemObject的GetFolder()方法根据当前扫描目录获取 Folder对象。然后，使用For...Each语句枚举文件夹中的所有文件。如果文件名中包含文本“ .xml ”(扩展名)，我们将在ListView中列出文件名（参见图 17-28）。

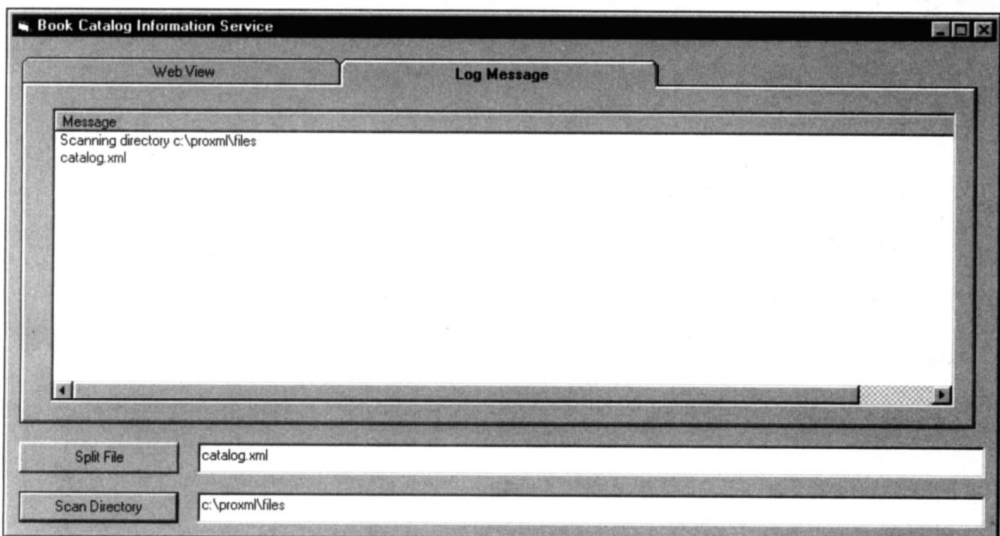


图 17-28

成功检测到文件之后，我们需要依次处理每个文件。为此，我们在窗体中添加函数 ProcessFile()。该函数改写了 Split File 点击处理器中的代码，它用于接受文件名，我们没有使用编辑框中的值：

程序清单 17-57

```
Private Sub ProcessFile(sFilename As String)
    Dim oUser As CUser
    Dim oLoadUsers As New CLoadUserInfo
    oLoadUsers.LoadUserInfo

    For Each oUser In oLoadUsers.Users
        Me.LogWindow.ListItems.Add , , "User=" & oUser.UserName
        If oUser.DeliveryFormat = "XML" Then
            PerformXMLToXMLTransform oUser, sFilename
        Else
            PerformXMLToHTMLTransform oUser, sFilename
        End If
    Next oUser
End Sub
```

我们可以从扫描代码中调用该函数，为每位订阅者处理每个文件：

程序清单 17-58

```
For Each oFile In oFolder.Files
    If InStr(oFile.Name, ".xml") <> 0 Then
        Me.LogWindow.ListItems.Add , , oFile.Path
        ProcessFile oFile.Path
    End If
Next oFile
```

扫描目录时，你应该保证目录中只包含有效的 XML 分类文件，否则你的订阅者将接收空的 XML 或 HTML 文件。在实际应用系统中，你必须改进转换类模块接口，使之能够产生某种反馈，说明是否创建了输出，以及是否值得发送。

如果运行项目，并指定包含 10 个 XML 分类文件的文件夹，而且我们的 Users.xml 文档定义了三个订阅者，程序总共将执行 30 次转换。我们当然可以改写代码，使之仅执行三次转换，如果你打算采用这种方式，就把它作为练习吧。然而，还有一个问题有待解决。对于 3 个用户 10 个文件的情况，每次转换都使用相同的输出文件名。在执行下一次转换之前，我们需要对前一次转换的结果进行一定的处理，以免丢失。对于 BCIS，我们采取的操作是将转换结果通过电子邮件发送给用户。

7. 直接邮件分发组件

为了将图书信息分发给订阅者，我们使用 Microsoft Collaboration Data Object (CDO) 库。在编译以下代码之前，需要在项目中添加 CDO 支持，因此从 Project | References 对话框中选择类型库 “Microsoft CDO 1.21 Library”。如果找不到 CDO 库，说明你需要从 Office CD 中安装 Outlook 中的组件。参照以下代码添加类 CSimpleMailer 和 SendFile() 子例程：

程序清单 17-59

```

Option Explicit

' Send a file to the specified email address

Function SendFile(sTo As String, sFilename As String) As Boolean
    Dim oSession As New Session
    Dim oMessage As Message
    Dim oRecipient As Recipient
    Dim oAttachments As Attachments

    ' Simple catch all error
    On Error GoTo failed

    ' Establish mapi session. You need to adjust this to match your mail profile
    oSession.Logon "Microsoft Outlook"

    ' Create a new message setting subject, text and attachment
    Set oMessage = oSession.Outbox.Messages.Add
    oMessage.Subject = "Automated Catalog Update Delivery"
    oMessage.Text = "Please find attached the latest catalog filtered " & _
        "especially for you!"

    ' Set the attachment
    oMessage.Attachments.Add "Catalog", , CdoFileData, sFilename

    ' Set the recipient
    Set oRecipient = oMessage.Recipients.Add(Name:=sTo, Type:=CdoTo)
    oRecipient.Resolve False

    ' Send the message and log off
    oMessage.Send
    oSession.Logoff

    SendFile = True
    Exit Function

failed:
    SendFile = False
End Function

```

你应该将以下行：

```
oSession.Logon "Microsoft Outlook"
```

替换为表示电子邮件帐号的字符串，例如：

```
oSession.Logon "Karli Watson", "", True, True
```

该命令的详细语法超出了本章的讨论范围。

要了解有关CDO的更多信息，参见《Professional CDO Programming》(ISBN：1-861002-06-8) 或《ADSI CDO Programming with ASP》(ISBN：1-861001-90-8)。

为订阅者创建了发送电子邮件文件的类之后，让我们回到主窗体的 ProcessFile()函数并修改它，如果为订阅者配置了电子邮件地址，它将使用邮件类发送文件：

程序清单 17-60

```

Private Sub ProcessFile(sFilename As String)
    Dim oUser As CUser
    Dim oMailer As New CSimpleMailer

```

```
Dim bRC As Boolean
Dim oLoadUsers As New CLoadUserInfo
oLoadUsers.LoadUserInfo

For Each oUser In oLoadUsers.Users
    Me.LogWindow.ListItems.Add , , "User=" & oUser.UserName
    If oUser.DeliveryFormat = "XML" Then
        PerformXMLToXMLTransform oUser, sFilename
    Else
        PerformXMLToHTMLTransform oUser, sFilename
    End If

    ' If an email address is present e-mail the file to the user
    If oUser.Email <> "" Then
        bRC = oMailer.SendFile(oUser.Email, oUser.OutputFile)
        If bRC = True Then
            Me.LogWindow.ListItems.Add , , "E-mailed file to " & oUser.Email
        Else
            Me.LogWindow.ListItems.Add , , "Failed to e-mail file to " & _
                oUser.Email
        End If
    End If
End If
Next oUser
End Sub
```

对代码的修改包括：声明邮件组件实例，并在转换完成之后检查订阅者的 CUser对象的 Email属性。如果存在电子邮件地址，我们通过邮件将输出的结果发送给订阅者，并根据 CSimpleMailer类返回的代码写入成功/失败日志。

再次运行项目，点击扫描目录按钮请求处理数据时，你将在日志窗口中看到有关邮件发送的消息（参见图 17-29）。

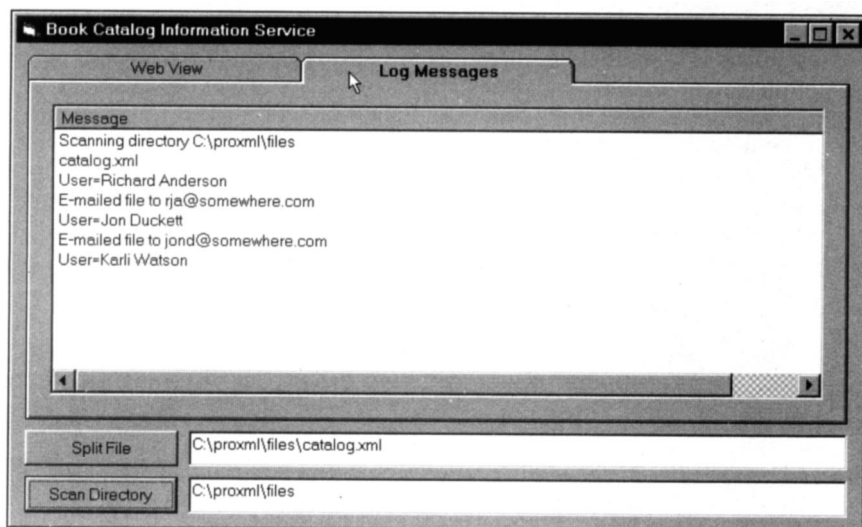


图 17-29

每位订阅者都将在他们的收件箱中看到输出结果（参见图 17-30）。

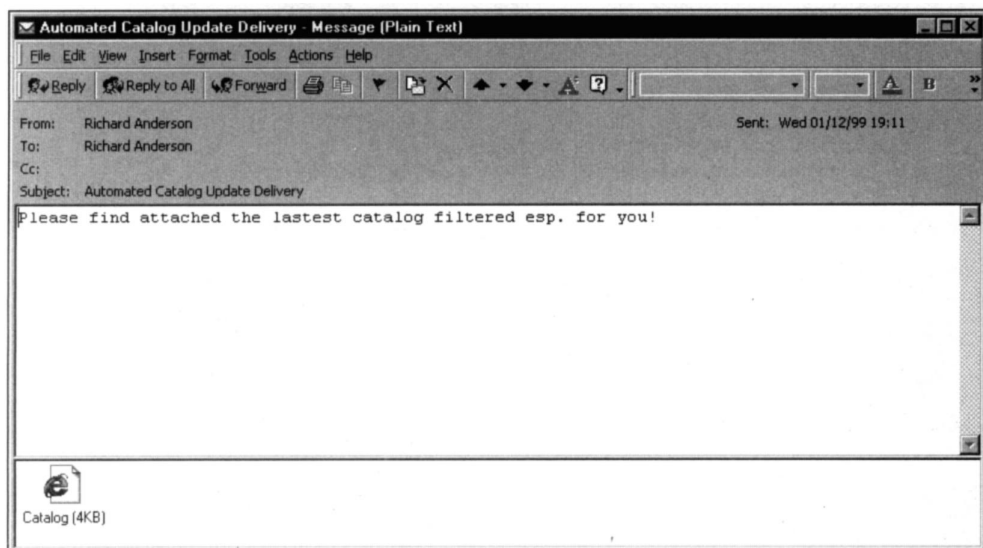


图 17-30

Outlook等大多数电子邮件客户都能够自动启动相关的查看程序，使得你很容易看到 HTML 内容。

17.2 小结

在本章中，我们介绍了如何使用 VB 类模块而不是 XSLT 转换 XML 数据。该方法是建立在 XSLT 的原理基础之上的，它并不是 VB 所特有的，你也可以用其他语言来实现。

选择本章介绍的方法或者 XSLT 主要取决于你编写的应用程序。XSLT 是一种功能相当强大的语言，但是它受到当前规范的限制，而且不能简便地执行数据库查询，以及高级的单文件入多文件出的转换。