

第8章 链接和查询

当XML 1.0规范刚刚被广泛而且稳定使用的时候，W3C很快又制定了一些其他标准，这些标准会在功能上超出XML，这在前面我们已经提到过，但是要想充分运用XML，我们需要一种方法在指向和查询XML文档之间运用链接。

当我们开始把越来越多的信息存入XML文档时，我们就需要构造一种方式，通过接口获取信息。我们需要一种方法来确定文档各个部分之间的关系，以及访问一个与其他资源有关的文档的内部各部分（或资源）。这些资源可以是同一文档的不同部分，不同文档的各个部分，甚至根本不是XML的项目。

在这一章，我们将会看到关于链接和查询的六个主要的方面：

- XML 信息集合——定义了各种信息的W3C文档，它们共同构成XML信息文档。充分了解信息集合是解决本章其他问题的关键。
- XLink——是W3C链接XML内部文档和其他资源的机制，大致类似一个超链接。XLink也可以把非XML文档链接起来。
- XPath——W3C关于查询部分XML文档的通用语言标准。
- XPointer——W3C指向一个XML文档的特殊区域或者特殊部分的机制。注意到XPointer包含XPath——XPath用来定义查询机制，而XPointer用来从非XML资源里区别出XML文档，比如HTML。
- XML文档片段交换——W3C关于传送部分XML文档的规范。它提供了一种详细说明文档上下文环境信息的途径，无须传送全部文档。
- 查询XML文档——我们将会看到XSLT（将在下一章详细说明）和怎样用它来查询XML文档。

8.1 XML 信息集合

XML信息集合即Infoset是一个W3C创造的工作草案，用于描述由大量信息共同构成的格式正规的XML文档。W3C文档关于Infoset的最新版本可在<http://www.w3.org/TR/xml-infoset>中找到。

不要被这些标准的术语所迷惑，我们的根本目的是提供一些通俗的词汇去描述一个XML文档的内容。任何反馈关于XML文档内容信息的XML处理器都能够按照这些信息的种类来分项描述内容。这些词汇构成了其他所有W3C标准升级的基础，它可以让程序访问XML文档的内部，而旧的标准则不得不依附于它们的下一次新版本。就像我们所看到的，文档对象模式（第5章讨论的）也按照这里所讲的信息类型控制着XML文档的内容。

8.1.1 信息类型

一个格式正规的XML文档由15种完全不同的信息共同构成。这些信息中的一部分要求必须按解析过的XML文档形式出现，以便与原始文档相适应（通过W3C标准），其他信息可以有选

择地放弃，那些说“必须”的只是在出现时必须（稍后我们将回到这个问题上）。这15种信息是：

- 一个确切的文档信息项目（必须的）
- 一个或者更多元素信息项目（必须的）
- 属性信息项目（必须的）
- 处理结构信息项目（必须的）
- 字符信息项目（必须的）
- 略过文档的引用信息项目（必须的）
- 注释信息项目（可选的）
- 一个文档类型声明信息项目（可选的）
- 实体信息项目（对于未分析的实体是必须的，对于已分析的实体是可选的）
- 符号信息项目（必须的）
- 实体开始标志信息项目（可选的）
- 实体结束标志信息项目（可选的）
- CDATA开始标志信息项目（可选的）
- CDATA结束标志信息项目（可选的）
- 命名空间声明项目（必须的）

每个格式正规的XML文档必须有一个确切的文档信息项目，和至少一个代表这个文档的根元素的元素信息项目。例如下面这个可能是最简单的格式正规的XML文档：

```
<Catalog/>
```

这个文档有一个文档信息项目，代表作为一个整体的文档（因为每一个XML文档都有一个文档信息项目），和一个代表<Catalog>元素的<Catalog>元素信息项目。

上面定义为可选项的项目，之所以是可选的是因为在一个分析过的文档表现中它们不是必须的，因为它要与W3C信息集合的原始定义完全相同。于是，假设你有下面的XML文档：

程序清单 8-1

```
<Catalog>
  <Book>
    <Title>Professional XML</Title>
    <!-- We are missing some information for this book! -->
  </Book>
</Catalog>
```

经过全面地分析，这个文档包括一个文档信息项目，三个元素信息项目，（Catalog，Book和Title）一个注释信息项目。根据信息集合的标准，注释信息项目可以不用，而且在原始文档中不会丢失信息——但是你必须保存这个文档信息项目和三个元素信息项目。

让我们浏览一下信息集合所详细说明的信息项目种类，以及它给每个项目所下的定义。

1. 文档

XML文档必须只有一个表明文档是个整体的文档信息项目，它有以下特点：

- 子信息项目按照它们在原始文档中出现的顺序排列。这个序列至少包括信息项目元素。另

外，它还必须包括在文档根元素内容之外定义的升级结构信息项目。由开发人员的决定，这个序列还应包括文档根元素之外的注释信息项目，如声明文档类型的信息项目（如果它在源文档中被详细说明）。

- 在一个文档中，服务于全部符号的无序的符号信息项目集合。
- 在一个文档中，服务于所有未分析实体的无序的实体信息项目集合。这个序列也可以包括分析过的实体信息项目、文档实体，如果开发人员选择的话，还包括外部的 DTD子集。
- 开发人员还可以选择包括所处理文档的 URI。

2. 元素

在XML文档中对于每一个元素，必须有一个元素信息项目。现在，对于一个元素的两种表示方法应该是一样的：

```
<Book/>
```

和

程序清单 8-2

```
<Book>
...
</Book>
```

元素信息项目有以下特征：

- 一系列有续的子元素、处理指令、略过实体的引用和字符信息项目，它们以在元素中出现的先后顺序排列。这个序列可以是空的。以开发人员的判断力，注释信息项目也可以被包括在内。这个序列还可以包括实体开始记号，实体结束记号，CDATA开始记号以及CDATA结束记号信息项目——如果包括这些，它们必须成对出现（没有开始记号就没有结束记号，反之亦然）。
- 一个属性信息项目的未排序集合是服务于这个元素的每一个属性的。这个集合也包括缺省属性。如果给这个元素一个命名空间属性，而分析器不懂命名空间，在这个序列里就需要有一个属性来描述命名空间；否则就不需要命名空间属性。这个装置可以是空的。
- 元素名称中相同资源识别器由命名空间处理器提供。如果分析器没有进行命名空间处理，或者没有命名空间详细说明这个元素，URI就是无效的。
- 元素名称的部分。如果分析器没有进行命名空间处理，这个名字就是确定的名称（如果一个元素名称包括一个命名空间标识符，它就包括命名空间和冒号）否则这只是名称的一部分（冒号后面的部分），或者如果命名空间被详细描述，这就是名称的全部。
- 一个关于命名空间声明信息项目参考的未排序集合。这些与被声明为这个元素的组成部分的命名空间相一致。
- 以开发者的判断力，在这个元素的范围内，一个关于命名空间声明信息项目说明的未排序集合与命名空间声明相对应（它们是在这个元素或以前的元素中被声明的）。

可能对于命名空间处理过程还有一些不清楚之处，下面举一些例子：

没有命名空间的XML：

程序清单 8-3

```
<Catalog>
  <Book>
    ...
  </Book>
</Catalog>
```

对于了解命名空间的分析器和不了解命名空间的分析器 <Book>元素可能是：

程序清单 8-4

```
URI = (null)
Local name = Book
```

反之，如果有命名空间的 XML：

程序清单 8-5

```
<Catalog xmlns:wrox='http://www.wrox.com/Catalog'>
  <wrox:Book>
    ...
  </wrox:Book>
</Catalog>
```

对于一个了解命名空间的分析器，<Book>元素有以下特性：

程序清单 8-6

```
URI = http://www.wrox.com/Catalog
Local name = Book
```

而对于一个不了解命名空间的分析器，它有以下特性：

程序清单 8-7

```
URI = (null)
Local name = wrox:Book
```

有关命名空间的更多信息请参见第 7 章。

3. 属性

对于文档中的每一个属性一定有一个属性信息项目。如果处理器了解命名空间，用于定义命名空间的属性就不会作为信息项目出现。例如：

程序清单 8-8

```
<Book color='red'>
  ...
</Book>
```

在这里，color 是一个属性。

属性信息项目有以下特点：

- 属性名称的 URI 部分。参看以前关于解释了解命名空间和不了解命名空间的分析器怎样对待 URI 元素的定义。

- 属性名称的本地部分。再参看元素的定义来了解如何分析命名空间。
- 一个已排序的关于每一个出现在（已格式化的）属性值里的特征信息项目序列。另外在属性值中，关于每一个实体引用的实体开始标记和实体结束标记信息项目也可以被包括在这个序列中。
- 由开发人员决定，一个标志显示出这个属性值在一个 DTD 或者模式中被详述或者被缺省。
- 由开发人员决定，这个属性的缺省值来自 DTD。
- 由开发人员决定，声明 DTD 中这个属性的类型——ID、IDREF、IDREFS、ENTITY、ENTITIES、NMTOKEN、NMTOKENS、NOTATION，CDATA 或 ENUMERATED 的属性类型。

4. 处理指令

在文档中，每一个处理指令都必须有一个处理指令信息项目。出于信息集合的目的，不把 XML 声明和外部已分析过的实体声明为处理指令。处理指令的语法是：

程序清单 8-9

```
<?operation foo?>
```

处理指令信息项目有以下特点：

- 处理指令的目标。这是在处理指令名称中紧随着“<?”的第一个符号。
- 处理指令的内容。这是结束词“>”前文本，带有去掉的前导空白空间。这也可以是空串。
- 开发人员还可以包括从开始就包括这个处理指令的实体 URI。（如果知道处理指令被逐行声明，这将是已处理文档的 URI）。

5. 对于忽略实体的引用

由于一种引用对应于一个实体，这里必有一种实体对应于忽略的实体信息项目，实体信息的忽略是由于未经验证的分析器不能解释，或者因为没有读到未知实体的定义（比如如果不可得到），或者因为分析器不包括扩展的已分析实体。

如下例所示，该实体引用：

程序清单 8-10

```
&Book;
```

由未经验证的分析器读取，但是这个分析器不去选择扩展外部的已分析实体。这个信息将由未知实体信息项目的引用来代替。

未知实体信息项目的引用有以下特征：

- 被说明过的实体的名称。
- 由开发人员决定，如果分析器已读过声明，那么就是未扩展的外部已分析实体信息项目的引用。

6. 字符

对于文档中的每一个未标记的字符都有一个字符信息项目。也就是每一个字符有一条信息

项目。

如下面的例子：

程序清单 8-11

```
<Book>
  <Title>ABC</Title>
</Book>
```

它有三个字符信息项目：一个是关于 A 的，一个是关于 B 的和一个关于 C 的。实际上，实现将相邻的字符组成一个文本或字符串，而不是单独地列出每个字符（在 W3C Infoset 规范中，这是可行的）。

字符信息项目有以下特性：

- 在 ISO10646（统一码）字符编码。
- 一个标明这个字符是在空白区内或不在空白区的标志。有效的分析器必须有这个标志，无效的分析器可以有选择地对一些不真实的信息作这个标志。
- 开发人员可以指明这个字符是否被包括在事先已确定的 XML 实体中。

7. 注释

对于文档中的每一个注释都有一个注释信息项目。下面就是一个注释：

程序清单 8-12

```
<!-- We need more information on this book! -->
```

注释信息项目包括以下内容：

- 注释的内容。

8. 文档类型声明

在文档中，如果开发人员选择了一个文档类型声明，就要有一个文档类型声明信息项目，例如：

程序清单 8-13

```
<!DOCTYPE catalog SYSTEM "http://www.wrox.com/Catalog/Catalog.dtd">
```

一个文档类型声明信息项目可以有以下内容：

- 一个关于外部 DTD 子集实体信息项目的引用。
- 一个关于出现在 DTD 里注释和处理指令信息项目的有序列表的引用。

9. 实体

未经分析的外部实体应该作为实体信息项目出现。在文档中，对于每一个实体都要有另外一个实体信息项目。如果一个实体不只一次被声明，只能用第一次声明来创建实体信息项目。

下面是一个内部实体声明的例子：

程序清单 8-14

```
<!Entity Version "1.0">
```

下面是一个外部实体声明的例子：

程序清单 8-15

```
<!Entity ProXMLBook SYSTEM "http://www.wrox.com/Catalog/ProXMLBook.xml">
```

下面例子是关于非XML数据类型的一个外部实体声明：

程序清单 8-16

```
<!Entity ProXMLCover SYSTEM "http://www.wrox.com/Catalog/ProXMLCover.gif"
  NDATA gif>
```

实体信息项目有以下特性：

- 实体种类（内部参数实体，外部参数实体，内部一般实体，外部一般实体，未分析的实体，文档实体或者外部DTD子集）。
- 实体名称。如果实体信息项目是这个文档或外部 DTD 子集，那么它是空。在上面的例子中，实体的名称分别是 Version、ProXMLBook 和 ProXMLCover。
- 实体的系统标识符。对于内部实体，这个内容是空；对于文档实体，它可能是空，也可能装有文档的系统标识符。系统标识符那个例子中是空。

`http://www.wrox.com/Catalog/ProXMLBook.xml`，和

`http://www.wrox.com/Catalog/ProXMLCover.gif`。

- 可能有实体的公共标识符。内部实体为空。
- 如果一个实体是未分析实体，那么还有和这个实体相关的符号信息项目。这对于其他实体信息类型是空。
- 实体的基础 URI。如果这个实体是个内部实体，这个值是空。
- 实体的内容，如果开发者决定这是一个内部实体。
- 开发人员还可以包括表示实体的字符编码集合的名称。
- 开发人员还可以包括一个表明实体独立状态的标识：有效值是 “ yes ” “ no ” 以及 “ not present ”。

10. 名称记号

对于 DTD 里声明的每一个名称记号都有一个名称记号信息项目，例如：

程序清单 8-17

```
<!NOTATION gif SYSTEM "gifviewer.exe">
```

名称记号信息项目有以下特性：

- 名称记号名称。
- 名称记号的系统标识符，或者如果没有指定系统标识符，这就是空。
- 名称记号的公共标识符，或者如果没有指定公共标识符，这就是空。
- 名称记号相应的基础 URI。

11. 实体开始标志

实体开始标志表明从一个已分析的普通实体插入文本的开始。在参数实体中不用这些标志。

实体开始标志信息项目有以下特性：

- 对于插入文本，有一个关于实体信息项目的引用。

12. 实体结束标志

实体结束标志表明从一个已分析的普通实体插入文本的结束。在参数实体中不用这些标志。

实体结束信息项目有以下内容：

- 对于插入文本，有一个关于实体信息项目的引用。

13. CDATA 开始标志

在CDATA部分，隐藏着一个表明所嵌入文本开始的实体开始标志。

CDATA开始标志信息项目没有特性。

14. CDATA 结束标志

在CDATA部分，隐藏着一个表明特性结束的实体结束标志。

CDATA结束标志信息项目没有特性。

15. 命名空间声明

对于每一个作为一个元素属性的命名空间来说，都有一个命名空间声明。

命名空间声明信息项目有以下内容：

- 要声明的命名空间。这是跟在以前确定的 `xmlns`：前缀后面的属性名称部分。
- 关于要声明命名空间的绝对 URI。必须全部或者单项列出这个特性及子特性（下面要详述的）。
- 构成属性值内容的一系列关于字符信息项目的有序字符序列引用。这还包括实体开始标志和实体结束标志以表明实体引用的区域。这个特性或者绝对 URI 特性（或者全部）必须存在。W3C已经加上这个特性，以便于以后能够运用非 URI命名空间。

8.1.2 信息集合的重要性

如果这些“信息项目”和“特性”听起来比较熟悉，实际上，它们应该是与 DOM里的项目完全相同的（在第5章详述）。事实上，W3C详述过的所有进入XML文档的多种技术——XML DOM，XLink，XPath，XPointer，以及XSLT——都是从XML信息集合描述过的基础结构得来的。为了充分利用这些技术提供的功能，你必须抛开认为XML文档是文本流的想法，而应该结合这些对象考虑它们。

最后，尽管信息集合为对象之间的连接提供了详细说明，它却没有详细说明任何特殊实现。我们将要讨论的技术用树结构来表示这些对象，还可以确切地把它们表示为带有指针的对象流——事实上，这也是大多数事件驱动处理程序的运作的方式。让我们看一个例子。假设有下面这个文档（你现在应该已经熟悉）——我将会放进color属性来修饰一下：

程序清单 8-18

```
<Catalog>
  <Book color="red">
    <Title>IE5 XML Programmer's Reference</Title>
    <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>
</Catalog>
```

图8-1是这个文档的一个树结构表示：

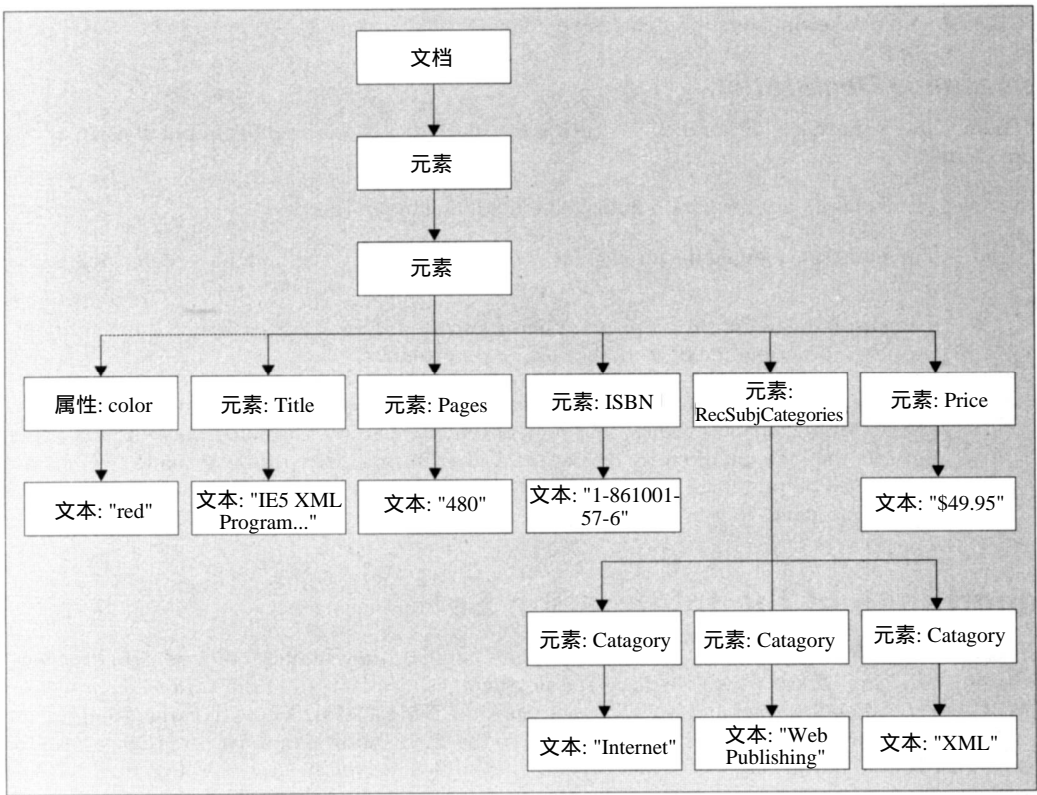


图 8-1

然而，Infoset还可以允许用表8-1中的形式表示文档：

表 8-1

信息项目 ID	信息项目类型	属 性
1	文档	子列表：2
2	元素	URI：空本地名：Catalog子列表：3属性集：空
3	元素	URI：空本地名：Book子列表：6,8,10,12,19属性集：4
4	属性	URI：空本地名：属性的彩色Text：5
5	文本	值：“red”

(续)

信息项目 ID	信息项目类型	属 性
6	元素	URI : 空本地名 : Title子列表 : 7属性集空
7	文本	值 : " IE5 XML Programmer's Reference "
8	元素	URI : 空本地名 : Pages子列表 : 9属性集 : 空
9	文本	值 : 480
10	元素	URI : 空本地名 : ISBN子列表 : 11属性集 : 空
11	文本	值 : " 1-861001-57-6 "
12	元素	URI : 空本地名 : RecSubjCategories子列表 : 13, 15, 17属性集: 空
13	元素	URI : 空本地名 : Category子列表 : 14属性集 : 空
14	文本	值 : " Internet "
15	元素	URI : 空本地名 : Category子列表 : 16属性集 : 空
16	文本	值 : " Web Publishing "
17	元素	URI : 空本地名 : Categbry子列表 : 18属性集 : 空
18	文本	值 : " XML "
19	元素	URI : 空本地名 : Price子列表 : 20属性集 : 空
20	文本	值 : " \$49.99 "

8.1.3 小结

为了用W3C详细描述的各种技术访问和操作 XML文档，你首先需要了解 W3C在InfoSet规范中所定义的分区。一旦你能够从根据内容而考虑 XML文档转向根据链接的信息项目而考虑它们，就会发现通过链接和查询机制访问 XML文档变得非常直接和自然。这些技术使用在 InfoSet里描述的信息集合项目信息来控制 and 访问 XML文档的，它们依赖于这些用来浏览文档的项目里表示的父-子信息。

8.2 链接

让我们看一下怎样才能通过运用链接把 XML功能扩展到外部资源上，诸如其他 XML文档、HTML文档甚至图像。正如我们将会看到的，我们可以用链接去定义相似文档之间的关系，确定浏览文档的顺序，甚至在一个 XML文档中内嵌入非XML内容。

8.2.1 什么是链接

如果你使用过 HTML，一定熟悉超链接的概念。你能指定一个锚 (anchor)，使其起着链接到另一个文档的作用：

```
<A HREF="www.wrox.com/Catalog/Catalog.html">Book catalog</A>
```

这个声明告诉我们两件事：第一，它表明了文本Book catalog是两个资源之间连接的开始（因为它包括在一个含有HREF属性的A标记中）。第二，它表明URL www.wrox.com/Catalog/Catalog.html是相同连接的目标。这是关于链接的一个简单的例子——资源之间的一个连接。XML链接与

HTML链接相似，只是在功能上更强大、更具灵活性（我们将会在后面看到）。

1. 概念上的链接和显示的区别

注意上面的内容通常以某种方式突出显示连接的开始处（加下划线，改变它的颜色，等等），通过单击鼠标实现对目标内容的浏览。但是这种行为没有明确地定义在 HTML规范里——一个内容显示引擎能自由地以任何方式进行表现。如果你想理解 XML链接的工作方式，理解链接与显示的差别非常重要，因为XML链接规范只提供一个概念模型。

2. HTML链接问题

我们以前曾提到HTML链接与XLink相比非常不灵活。特别是，HTML锚链接有下面缺点：

(1) HTML链接内嵌在源文档里

这个限制防止我们在标记为不能编辑的文档之外建立链接，例如，一个文档不提供原始标记能力（比如图像文件）。如果在某种程度上能够让链接离开它们引用的文档，我们就能构造一个自己的链接库（在一个链接数据库或文件），它将我们的内容连接在一起，但受中心位置管理。稍后我们将要再回顾一下链接数据库这个主题。

(2) HTML链接只允许在一个方向浏览

如果我们有一系列希望用户能够浏览的页，我们需要明确地定义各页面之间的超链接。例如我们可以有一个文档叫 page1.htm，它包含超链接到 page2.htm的单词Next。如果我们想定义超链接（比如说单词 Previous）以使我们浏览到 page1.htm，我们需要明确地在这里定义指向其他方向的链接。如果一旦我们能够声明两个页是链接过，而不用担心浏览方向，这将是非常好的事情。

(3) HTML链接只链接两个资源

我们已经在Internet上看到站点，那里多个信息页被一系列的链接列表或被 Previous和Next连接：

程序清单 8-19

```
<H3>Joe's Grill - Menu</H3>
```

```
Breakfast:
```

```
Two eggs, any style.....$1.95  
Bacon.....$1.25  
Sausage.....$1.25  
Pancakes.....$3.00
```

```
<P><A HREF="http://www.joesgrill.com/beverages.htm">Beverages</A></P>  
<P><A HREF="http://www.joesgrill.com/appetizers.htm">Appetizers</A></P>  
<P><A HREF="http://www.joesgrill.com/sandwiches.htm">Sandwiches</A></P>  
<P><A HREF="http://www.joesgrill.com/grill.htm">From the Grill</A></P>  
<P><A HREF="http://www.joesgrill.com/dessert.htm">Desserts</A></P>  
<P><A HREF="http://www.joesgrill.com/main.htm">Return to main menu</A></P>
```

当然，菜单的每个页有近似的超链接列表，这就使菜单变得有魔力。如果 Joe's Grill决定加一个意大利面食菜单，每个页将需要有一个链接添加到意大利面食菜单上。如果我们能指定所有资源被链入一个地方并让浏览器负责在它们之间浏览，这将非常好。

(4) HTML链接不能指定显示引擎的行为

如果能指定引擎显示内容的一些附加概念行为的话将非常好显示引擎应该自动穿越链接，

还是应该在做此之前等待用户的交互？显示引擎应该创建一个新的上下文（对于浏览器是一个新的窗口）来表现链接内容，还是应嵌入当前内容？通过 HTML 链接，只有目标命名的窗口可以被指定，浏览器将根据它当前的状态做不同的反映（例如，如果一个带有明确名称的窗口已经打开，它将覆盖内容；否则，它将创建新窗口）。

W3C 的 XML 链接规范正像我们将要看到的，描述了所有这些问题。

8.2.2 W3C 规范：XLink

XML 链接规范被称作 XLink。它现在处在工作草案阶段，这意味着实现的细节可能在它变成 W3C 推荐书之前改动。这个文档最新的版本可在 <http://www.w3.org/TR/WD-xlink> 上找到（1999 年 12 月 20 日为止的最新版本）。

自从 XLink 规范还在工作草案阶段时起，到截稿时止，这里没有任何强大的功能上的变动。但是现在更好地理解这些概念，在它进入主流时将要使你能够充分利用 XLink 的优点。

1. XLink 声明

命名空间通过 W3C 声明了 XLink 的十二月工作草案：

<http://www.w3.org/1999/xlink/namespace/>

为了从一个 XML 文档声明一个链接，命名空间必须被定义成子树，链接在那里被声明。这里有两种方法声明一个链接：

- 你可以创建一个 XLink 元素。
- 你可以添加 XLink 属性到一个你自己的元素。

一个 XML 链接元素就是在 `<xlink:type>` 中的元素（注意在 `type` 位置有两种可能的值，我们将很快遇到它）：

程序清单 8-20

```
<xlink:simple href="authors.xml" role="author list" title="Author list"
              show="replace" actuate="onRequest">
  List of authors
</xlink:simple>
```

并且一个带有与它相关的 XLink 属性的元素，它带有一个 `xlink:type` 属性：

程序清单 8-21

```
<Authors xmlns:xlink="http://www.w3.org/XML/XLink/0.9" xlink:type="simple"
          xlink:href="authors.xml" xlink:role="author list"
          xlink:title="Author list" xlink:show="replace"
          xlink:actuate="onRequest"/>
```

当然命名空间应用的正常规则——如果你在上一级定义命名空间，则不用再一次声明链接元素。

注意如果你选择添加 XLink 属性到你自己的元素，并且你正使用一个 DTD，你将需要在 `<!ATTLIST>` 里为链接元素定义属性。否则，你的验证处理器将抱怨它不能识别 `xlink:*` 属性！对于上面的例子，在 DTD 里将需要这个元素定义：

```
<!ELEMENT Authors EMPTY>
<!ATTLIST Authors
  xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink/namespace/"
  xlink:type (simple|extended|locator|arc) #FIXED "simple"
  xlink:href CDATA #REQUIRED
  xlink:role CDATA #IMPLIED
  xlink:title CDATA #IMPLIED
  xlink:show (new|embedded|replace) "replace"
  xlink:actuate (onLoad|onRequest) "onRequest" >
```

2. 链接类型

正如我们刚刚说的，xlink:type元素和属性中type可以取两个值：simple或extended。simple链接类似于HTML超链接，而extended链接提供了更多的功能。正如我们后面将要看到的，简单链接是扩展链接的子集——尽管简单链接采用不同的语法。在本节中，我们将依次介绍每种类型的链接。首先从简单链接开始。

3. 简单链接

简单链接非常类似于HTML的链接，想必你对此应该非常熟悉。声明简单链接时，可以使用以下属性：

(1) xlink:type

对于简单链接，该属性总是 simple。如果你声明简单 XLink元素，元素名称应该是 xlink:simple。

(2) xlink:href

链接的目标URI。

(3) xlink:role

它是描述链接内容的功能的字符串。虽然 W3C没有指定role的用途，但是某些 XLink的实现使用role字符串控制文档的显示。

(4) xlink:title

这个用户可读的字符串描述了链接。同样，W3C没有指定如何将它应用于支持 XLink的显示程序，但是它能够为用户提供可视化的指示，说明元素是一个链接。

(5) xlink:show

该属性定义了如何向用户显示目标内容。它可以取以下三个值：

- new——目标内容应该显示在独立的环境中（对于浏览器，应该是新的浏览器窗口）。
- replace——目标内容应该替换原来环境中的源内容（对于浏览器，这是超链接的常规特征）。
- embedded——内容应该嵌入源文档的链接位置。

(6) xlink:actuate

该属性定义了何时触发链接。它可以取以下两个值：

- onRequest——用户必须采取某些操作才能够触发链接。它类似于HTML超链接的工作方式，用户必须点击链接的文本才能够激活链接。
- onLoad——加载源文档时，链接将自动激活。当 xlink:show属性为embedded时，该属性最

有用，但是当 `xlink:show` 为 `new` 时，也可以使用该属性（例如，打开源文档时，自动打开另一个环境窗口，并加载目的信息）。

简单链接的功能与 HTML 超链接基本相当——它以单方向链接两个位置，链接的开始总是链接本身的声明（参见图 8-2）。

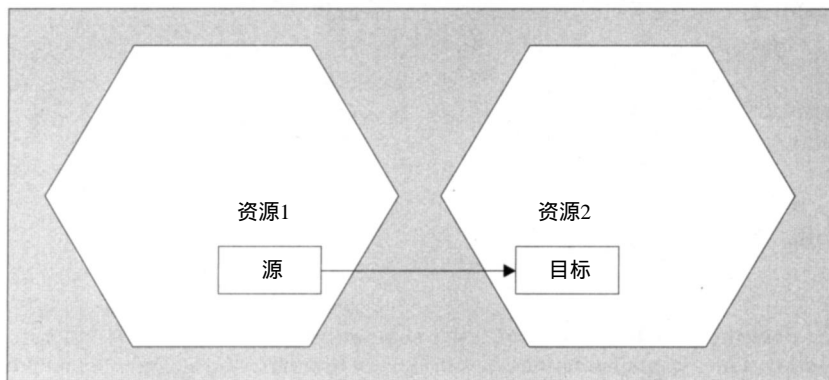


图 8-2

需要注意的是，即使我们将资源显示为不同的，对于两个相同的资源来说也是可以的（参见图 8-3）。

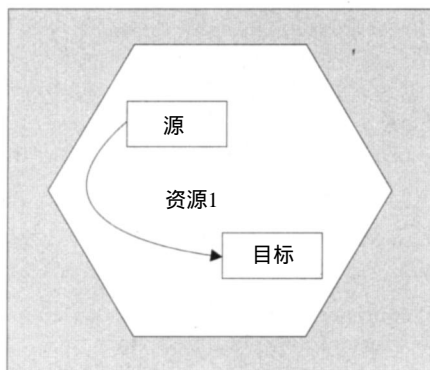


图 8-3

你应该牢记这一点，当我们讨论扩展链接时，会发现它的重要性。XLink 是对位置而不是资源进行操作的——资源的缺省位置为 “top”，但是资源类型也恰好采用这种定义。

4. 简单链接的例子

在我们继续讨论之前，先简要看两个例子：

程序清单 8-23

```
<xlink:simple xmlns:xlink="http://www.w3.org/1999/xlink/namespace/"
  xlink:href="authors.xml"
  xlink:role="author list"
  xlink:title="Author list"
```



```
xlink:show="new"  
xlink:actuate="onRequest"/>
```

这个例子创建了一个标题为 Author list 的链接，使用户知道有一个与之相关的链接信息（可以像HTML文档一样为之加上下划线）。当使用者击活这个链接，文档就会在新的上下文环境中打开。

程序清单 8-24

```
<authors xmlns:xlink="http://www.w3.org/1999/xlink/namespace/"  
  xlink:type="simple"  
  xlink:href="authors.xml"  
  xlink:role="author list"  
  xlink:title="Author list"  
  xlink:show="embedded"  
  xlink:actuate="onLoad"/>
```

这个例子说明，当源文档开始被提出来时，文档 authors.xml 应在源文档的链接位置被表示出来（记住这实际上应该由用户代理来决定如何对待链接，所以可能会有些变化）。

值得注意的是现在有些把 show 和 actuate 结合起来，而这并没有多大的意义，例如：xlink:show = "repalce" 和 xlink:actuate = "onLoad"。这样会潜在造成一个文档到另一个文档的重复定向，但是如果两个这样的链接同时出现在严格源文档中，应该会出现什么样的情形呢？对于所有特殊的显示行为，W3C 并没有试图去阐明在类似于这些的情景中，一个表达行为应该是怎样的，相信时间会告诉我们如何使用特定的工具去解决这个问题。

5. 扩展链接

对 XLink 来说，另外一种链接方式称为扩展链接。扩展链接允许把多个资源链接在一起，它们会被指定为（也就是说，在一个不是源文档的文档中）。让我们看一下扩展链接的语法。

这里定义了一个扩展链接：

程序清单 8-25

```
<!ELEMENT xlink:extended  
  (xlink:title*, xlink:arc*, xlink:locator, (xlink:arc | xlink:locator)*,  
  xlink:resource*)>  
<!ATTLIST xlink:extended  
  role          NMTOKEN          #IMPLIED  
  title         CDATA             #IMPLIED>
```

注意我们有四种类型的子元素：<xlink:title>、<xlink:arc>、<xlink:locator>和<xlink:resource>，稍后我们再讨论这些子元素。下面的属性可能与一个扩展链接相关：

(1) xlink:type

对于扩展链接来说，该属性总是属于扩展的。如果你声明了一个扩展 XLink 元素，那么这个扩展元素的名字应该是 <xlink:extended>。

(2) xlink:role

该属性和简单链接元素行使同样的功能。

(3) xlink:title

该属性和简单链接元素行使着同样的功能。

注意对于链接的目标元素来说并没有明确的标准——因为没有 href 属性。实际上，数据源也没有定义——不像简单链接，扩展链接并不意味着它们的数据源是链接所处的文档。为了指明参与链接的不同位置及链接之间的连接，我们有必要使用两个子元素 `<xlink:location>` 和 `<xlink:arc>`。

(4) `<xlink:title>` 元素

这个元素用来把扩展链接与语义信息相关联，例如：一个链接和菜单的不同页码相关联，可能具有值为“Menu”的 `<xlink:title>` 元素。这个信息的使用由处理器来完成——XLink 规范并没有规定该信息的使用。可以指定多个标题（例如，一个 XML 文档正在被国际化）。

这里是 `<xlink:title>` 元素的定义：

程序清单 8-26

```
<!ELEMENT xlink:title ANY>
<!ATTLIST xlink:title
  xml:lang      CDATA          #IMPLIED
>
```

其中一个属性 `xml:lang`，就是用来为国际化目的指定一种语言。

(5) `<xlink:locator>` 元素

这些总是作为扩展链接的子元素出现的，它们用来指定参与扩展链接的定位。例如，如果我们在五个不同的数据源之间采用链接（比如说，菜单的 5 个页码），那么参与链接的这五个子元素每个将有一个定位子元素。

这里是 `<xlink:locator>` 元素的定义：

程序清单 8-27

```
<!ELEMENT xlink:locator ANY>
<!ATTLIST xlink:locator
  href CDATA #REQUIRED
  role NMTOKEN #IMPLIED
  title CDATA #IMPLIED >
```

正如你所看到的，在这里你可以指定位置的 URI 以及它的文本名称和大致功能。这些属性发挥着它们在简单连接中同样的功能。

注意：一个定位并不是显式地指定一个链接——它仅仅指定一个参与链接的位置。为了定义定位之间显式的链接，我们需要使用 `<xlink:arc>` 元素。

(6) `<xlink:arc>` 元素

这些也总是作为扩展链接的子元素出现的，用来定义参与扩展链接的两个定位之间的连接。

这里是 `<xlink:arc>` 元素的定义：

程序清单 8-28

```
<!ELEMENT xlink:arc ANY>
<!ATTLIST xlink:arc
  from      NMTOKEN          #REQUIRED
  to        NMTOKEN          #REQUIRED
  show      (new|embedded|replace) "replace"
  actuate   (onRequest|onLoad)  "onRequest" >
```

show和actuate属性发挥着它们在简单连接中同样的功能：它们定义链接如何被初始化和显示。如果不指定这些属性，应该由部分实现来决定如何横贯这些 arc，这里有两个新属性：

(7) xlink:from

这个元素是定义链接起始点的 <xlink:locator>和<xlink:resource>元素的role属性的值。显式定义了连接的数据源能够使我们创建外联数据库，就如我们本章后面所见到的。

(8) xlink:to

这个元素是定义链接截止点的 <xlink:locator>和<xlink:resource>元素的role属性的值。

注意扩展链接中现在有不只一个的定位和资源具有相同的 role，一个定义于role上的arc会把所有具有role 的定位连接起来。例如，假如说我们有下面的文档：

程序清单 8-29

```
<xlink:extended xmlns:xlink="http://www.w3.org/1999/xlink/namespace/"
    role="family"
    title="John Smith's family">
  <xlink:locator href="johnsmith.xml"
    role="parent"
    title="John Smith"/>
  <xlink:locator href="marysmith.xml"
    role="parent"
    title="Mary Smith"/>
  <xlink:locator href="billysmith.xml"
    role="child"
    title="Billy Smith"/>
  <xlink:locator href="kateysmith.xml"
    role="child"
    title="Katey Smith"/>
  <xlink:locator href="johnsmithjr.xml"
    role="child"
    title="John Smith Jr."/>
  <xlink:arc from="parent"
    to="child"
    show="replace"
    actuate="onRequest"/>
</xlink:extended>
```

扩展链接定义了下面的连接：

程序清单 8-30

```
John Smith to Billy Smith
John Smith to Katey Smith
John Smith to John Smith Jr.
Mary Smith to Billy Smith
Mary Smith to Katey Smith
Mary Smith to John Smith Jr.
```

(9) <xlink:resource>元素

这些是作为扩展链接的子元素出现的，是用来定义链接的本地（内联）部分（添加到<xlink:locator>元素，用来指定连接的外联部分）。这些元素具有role和title属性，具有一个ANY的内容类型，而这个内容的用途不是由XLink规范所定义的。

下面是<xlink:resource>元素的定义：

程序清单 8-31

```
<!ELEMENT xlink:resource ANY>
<!ATTLIST xlink:resource
    role          NMTOKEN          #IMPLIED
    title         CDATA            #IMPLIED
>
```

(10) 隐式与显式 Arcs

注意：查看一下参与扩展链接的位置列表可决定某些连接信息，假如我们声明了一个扩展链接：

程序清单 8-32

```
<xlink:extended xmlns:xlink="http://www.w3.org/1999/xlink/namespace/"
    role="menu pages"
    title="Joe's menu">
  <xlink:locator href="menu1.xml"
    role="menu page 1"
    title="Beverages"/>
  <xlink:locator href="menu2.xml"
    role="menu page 2"
    title="Appetizers"/>
  <xlink:locator href="menu3.xml"
    role="menu page 3"
    title="Sandwiches"/>
  <xlink:locator href="menu4.xml"
    role="menu page 4"
    title="Desserts"/>
  <xlink:arc from="menu page 1"
    to="menu page 2"
    show="replace"
    actuate="onRequest"/>
  <xlink:arc from="menu page 2"
    to="menu page 3"
    show="replace"
    actuate="onRequest"/>
  <xlink:arc from="menu page 3"
    to="menu page 4"
    show="replace"
    actuate="onRequest"/>
</xlink:extended>
```

声明中那些显式链接是由 <xlink:arc> 元素定义的（参见图 8-4）。

然而如果我们不包括 <xlink:arc> 元素，在扩展链接定义的一部分定义的每个定位之间会存在隐式 arc（参见图 8-5）。

W3C 并没有规定识别 XLink 解析器有必要处理隐式链接，或者是不是它们和显式链接的处理方式不同。另外，XLink 的实现如何处理这个问题还没有定论。

通过定义一个带有 locator 和 arc 的扩展链接，只要有必要可以按照任意复杂的式样，把任何数量的资源连接在一连。我们稍后看一下扩展链接的几种使用方法，不过首先我们看一下内联扩展链接和外联扩展链接的区别。

6. 内联扩展链接和外联扩展链接

扩展链接可以嵌入到参与扩展链接的一个资源中（如果这个资源恰好是 XML 文档）。这种链

接定义对于扩展链接中的任何定位并不意味着什么——它可以通过识别XLink处理器来解析，可以按照处理器显式定位之间的 arc 的任何合适的方式来使用它。

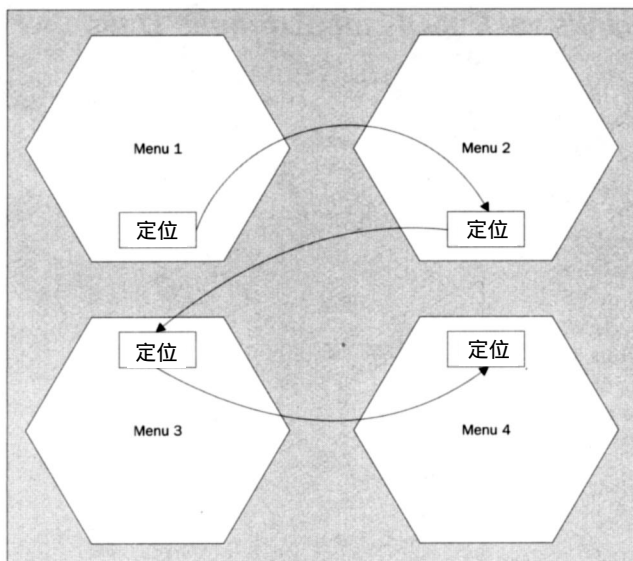


图 8-4

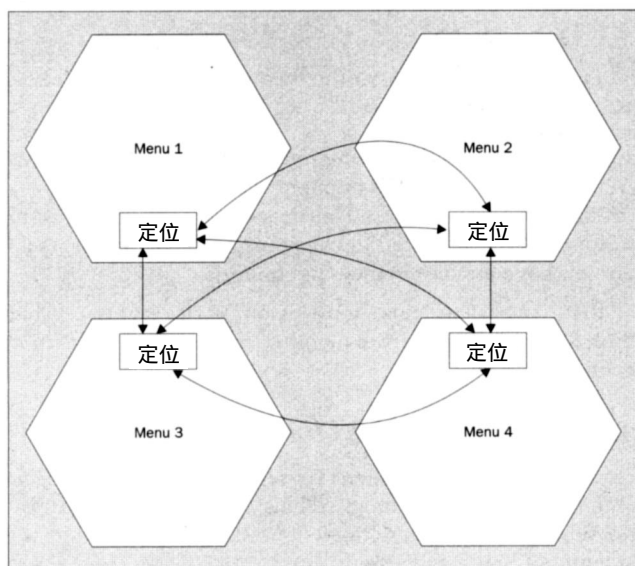


图 8-5

如果采用把扩展链接嵌入到链接中使用的一个位置，这种方法会产生两个问题：

- 如果包含了扩展链接的资源先被读取，那么一个识别 XLink 处理器能够浏览这个链接。但是如果资源 2 被首先读取了，将会发生什么呢？这样处理器就无法知道到资源 3 的合法链接

可以被浏览。这个链接信息可以在每个资源中重复，而这些资源在扩展链接中包含有定位，然而这样你就会陷入维护的梦魇。

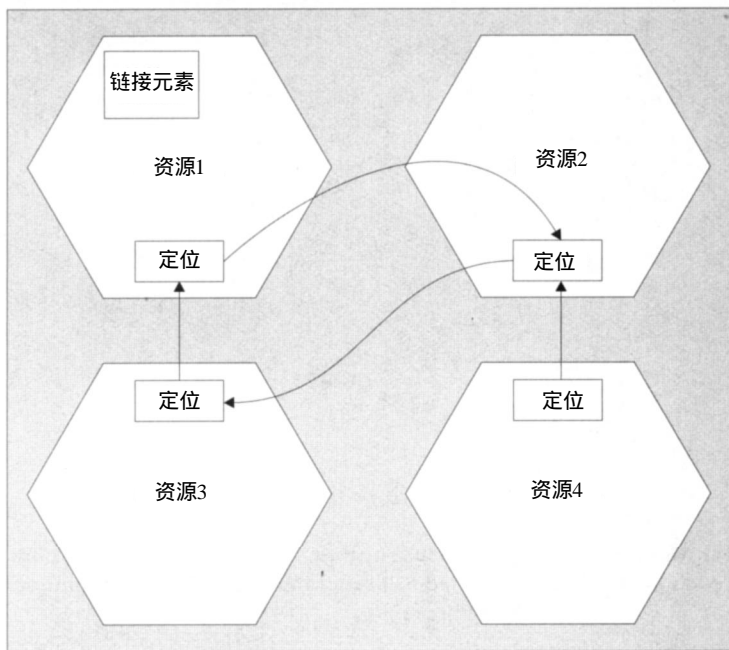


图 8-6

- 包含链接的资源必须是一个XML资源。例如，如果我们想在上面所显示的图案中把四个图像资源链接在一起，那该如何呢？我们在哪里放置链接信息呢？

7. 外联扩展链接——使用Linkbases

一类特殊的<xlink:extended>元素被用来向识别XLink处理器指明一个外联扩展链接存在于特定文档中。它们的role属性必须设置为xlink:external-linkset。locator子元素定义了外部XML文档，这个外部XML文档中包含了正被处理的文档的XLink信息：

程序清单 8-33

```
<xlink:extended
  xmlns:xlink="http://www.w3.org/1999/xlink/namespace/"
  role="xlink:external-linkset">
  <xlink:locator
    href="http://www.wrox.com/Catalog/linkdb.xml"
    role="linkdatabase"
    title="Out-of-line catalog links" />
</xlink:extended>
```

当一个识别XLink处理器遇到带有xlink:external-linkset角色的扩展链接，它读取了扩展链接元素的locator子元素指示的文档，来寻找引用了正被处理文档的外部链接。它然后就“记住”了那个信息，就好像它被包含在原始文档中。这将使维持项目间的链接变得更加容易——实际上，解决了前面的“Joe wants to add a pasta menu”问题。

图8-7是几个分离的 XML文档，它们包含有连接我们的资源外联链接信息。

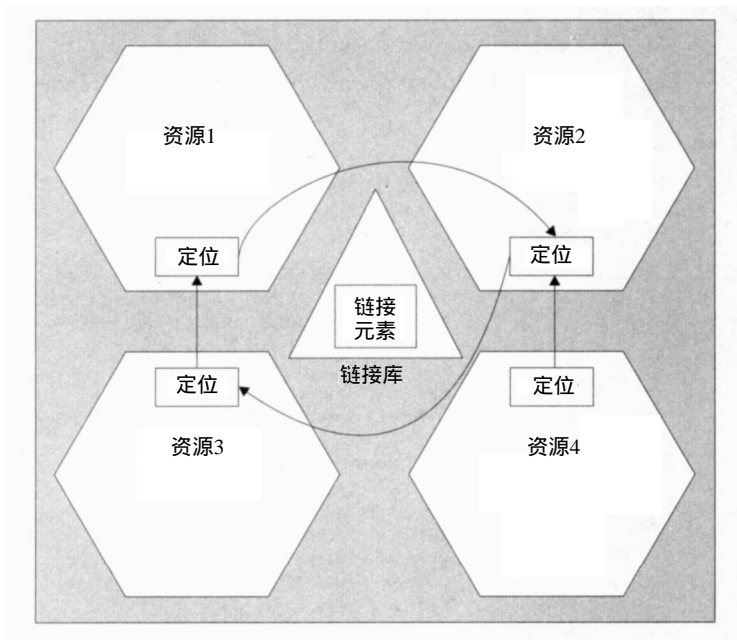


图 8-7

如果我们把扩展链接 `xlink:external-linkset` 加入到四个链接后的文档中，当其中任何一个文档变为当前文档时，该扩展链接信息仍可用（参见图 8-8）。

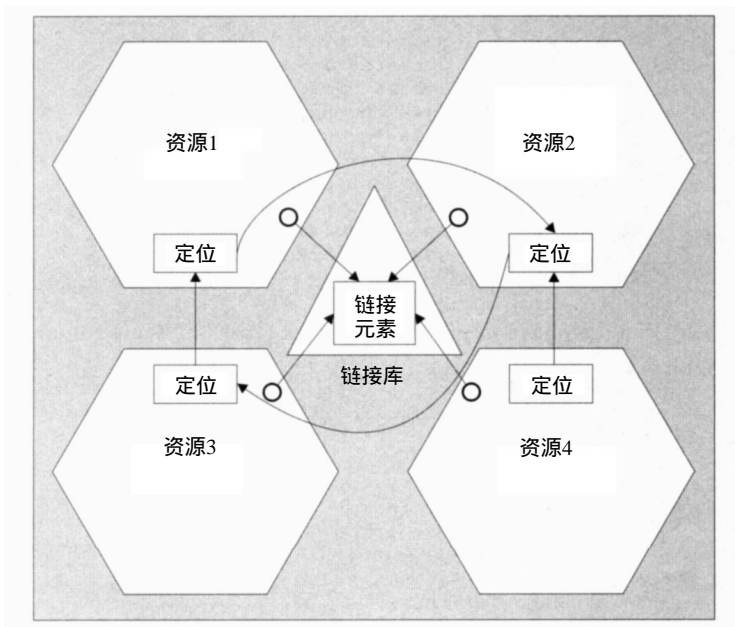


图 8-8

8. 扩展链接的几个例子

让我们看一下扩展链接如何解决我们前面提到的关于 HTML 链接的一些问题：

- HTML 链接必须嵌在源文档中。
- HTML 链接仅允许沿一个方向浏览。
- HTML 链接只链接两个数据源。
- HTML 链接并不指定显示引擎的行为。

(1) 链接数据库

利用扩展链接组，我们可以在文档之间维护链接列表。我们会有下面这个链接文档，menulink.xml：

程序清单 8-34

```
<xlink:extended xmlns:xlink="http://www.w3.org/1999/xlink/namespace/"
  role="menu links"
  title="See other parts of menu">
  <xlink:locator href="menu1.xml"
    role="menu page 1"
    title="Beverages"/>
  <xlink:locator href="menu2.xml"
    role="menu page 2"
    title="Appetizers"/>
  <xlink:locator href="menu3.xml"
    role="menu page 3"
    title="Sandwiches"/>
  <xlink:locator href="menu4.xml"
    role="menu page 4"
    title="Desserts"/>
  <xlink:arc from="menu page 1"
    to="menu page 2"
    show="replace"
    actuate="onRequest"/>
  <xlink:arc from="menu page 2"
    to="menu page 3"
    show="replace"
    actuate="onRequest"/>
  <xlink:arc from="menu page 3"
    to="menu page 4"
    show="replace"
    actuate="onRequest"/>
</xlink:extended>
```

然后我们将有下面的 menu1.xml（应用于所有菜单页相同的一般格式）：

程序清单 8-35

```
<menupage xmlns:xlink="http://www.w3.org/1999/xlink/namespace/">
  <xlink:extended role="xlink:external-linkset">
    <xlink:locator href="menulink.xml"/>
  </xlink:extended>
  <menuitem>
    <name>Coffee</name>
    <price>$0.99</price>
  </menuitem>
  <menuitem>
    <name>Tea</name>
    <price>$1.09</price>
```



```

</menuitem>
<menuitem>
  <name>Soda</name>
  <price>$1.25</price>
</menuitem>
</menupage>

```

当menu1.xml打开时，识别XLink处理器将从menulink.xml中读取扩展链接信息，然后将文档通过一种可以使该文档链接到 menu2.xml以更加清晰的方式显示出来。一个浏览器可以选择这种方式表达信息（当然，这仅仅是形式上的——目前情况下还没有浏览器直接支持 XLink，参见图8-9）。

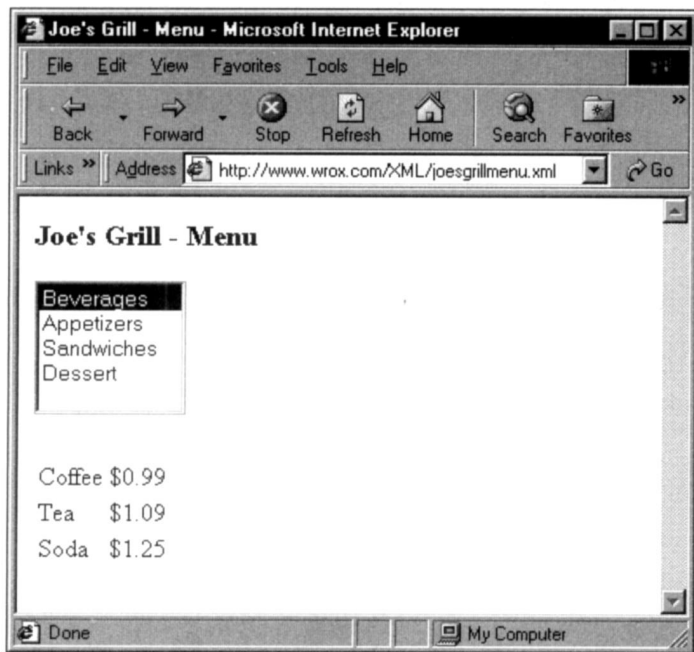


图 8-9

如果现在Joe想添加一个pasta菜单，我们所要作的仅仅是修改链接数据库文档：

程序清单 8-36

```

<xlink:extended xmlns:xlink="http://www.w3.org/1999/xlink/namespace/"
  role="menu links"
  title="See other parts of menu">
  <xlink:locator href="menu1.xml"
    role="menu page 1"
    title="Beverages"/>
  <xlink:locator href="menu2.xml"
    role="menu page 2"
    title="Appetizers"/>
  <xlink:locator href="menu3.xml"
    role="menu page 3"
    title="Sandwiches"/>

```

```
<xlink:locator href="menu4.xml"
    role="menu page 4"
    title="Desserts"/>
<xlink:locator href="menu5.xml"
    role="menu page 5"
    title="Pasta"/>
<xlink:arc from="menu page 1"
    to="menu page 2"
    show="replace"
    actuate="onRequest"/>
<xlink:arc from="menu page 2"
    to="menu page 3"
    show="replace"
    actuate="onRequest"/>
<xlink:arc from="menu page 3"
    to="menu page 4"
    show="replace"
    actuate="onRequest"/>
<xlink:arc from="menu page 3"
    to="menu page 5"
    show="replace"
    actuate="onRequest"/>
<xlink:arc from="menu page 5"
    to="menu page 4"
    show="replace"
    actuate="onRequest"/>
</xlink:extended>
```

当我们为这个 pasta 项目创建文档 menu5.xml 时，那么我们就无需返回去修改其他任何菜单页！我们的浏览器能够自动体现出这个变化（参见图 8-10）。

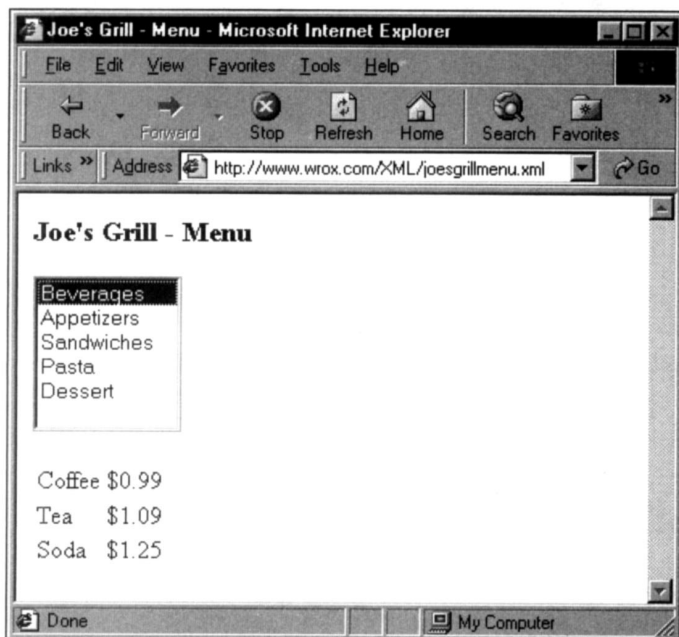


图 8-10

如果我们希望改变这些菜单页贯连的次序，可以通过修改链接数据库文档来实现这一点（不是去修改链接中出现的每一个参与文档，而这正是 HTML所需要的）。你可以看到，从内容中抽象出链接信息是用来控制和维持文档间链接的很强大的方法，这样就可以使其他没有控制你的内容的人员链接到该内容，或从该内容链接到其他地方——在下一部分你将会看到这一点。

(2) 标出只读文档

利用外链接注释来自其他数据源只读文档也是一个很好的方法。从下面的简短的例子可以看出它是如何做的。

假如我们有下面称为quotelist.xml的只读文档，它包含着引用，这可以通过注释来表明：

程序清单 8-37

```
<quotelist>
  <quote>
    Now is the time for all good men to come to the aid of their country.
  </quote>
</quotelist>
```

我们希望将这个文档加到另一个文档 comments.xml 中：

程序清单 8-38

```
<comment>
  By men, the author obviously meant all people, not the male gender.
</comment>
```

我们可以指定一个链接文档，commentlink.xml，它在文档 comments.xml 和只读文档 quotelist.xml 之间添加了链接：

程序清单 8-39

```
<xlink:extended xmlns:xlink="http://www.w3.org/1999/xlink/namespace/"
  role="quote comments"
  title="Comments">
  <xlink:locator href="quotelist.xml"
    role="quotes"
    title="Famous quotes"/>
  <xlink:locator href="comments.xml"
    role="comments"
    title="Commentary"/>
  <xlink:arc from="quotes"
    to="comments"
    show="new"
    actuate="onRequest"/>
</xlink:extended>
```

当识别XLink处理器打开链接文档时，读取扩展链接，可以看到 quotolist文档和comments文档之间存在一个链接。可以提出 quotelist内容，并允许链接回注释文档（参见图 8-11）。

点击链接元素就可得到 comment信息（参见图 8-12）。

如果我们能链接到 quotelist.xml 文档中精确的位置，该位置对应着我们正在评论的单词，“men”，那就更好了——不过这是 XPointer的工作，我们将在下面讨论。

9. Xlink总结

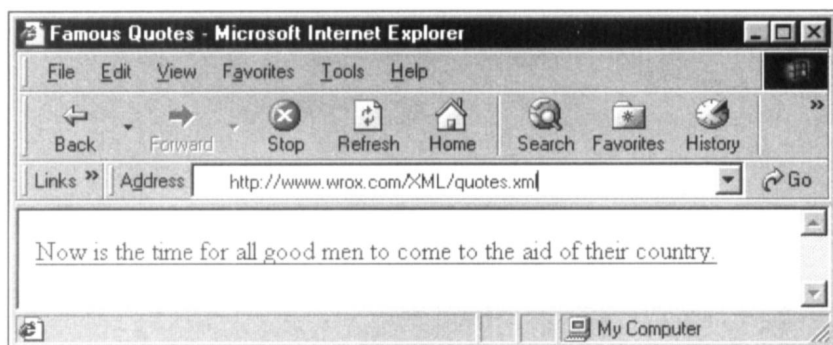


图 8-11

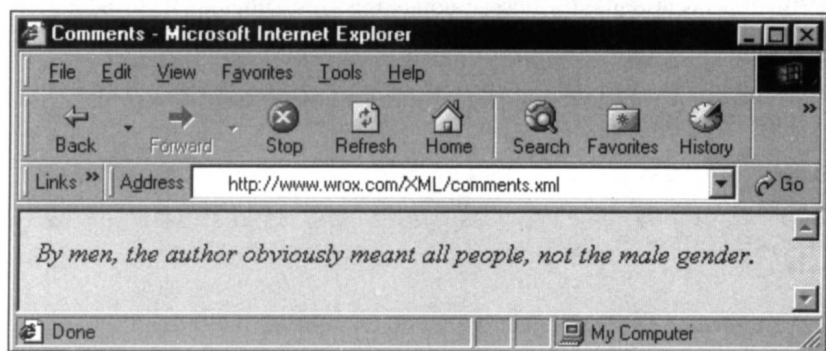


图 8-12

XLink为在XML文档中定义链接提供一种灵活的机制，使不同的资源连接到一起。这些资源甚至可以是通常并不包含链接的资源，像图像文件。XLink可以用于把一个文档链接到另一个文档，可以用于比HTML超链接更多的文档，或者它可以把许多不同资源链接在一起。它也可以用于从内容中抽象出链接信息，以便于链接信息的更新。然而，XLink还没有在任何一种目前最常使用工具包中被实现并流行——虽然我们毫无疑问地相信XLink的一些形式很快被用于XML的开发中。

8.3 XPointer

我们前面提到如果我们能够指向一个XML文档的一些组成部分，而不是指向整体文档——单独子树、属性或者甚至文本内容的一部分单独字符。W3C提出一个机制可以使我们这样做。

8.3.1 HTML指针

XML指针的概念在HTML中有一个类似物：``标识。这个标识指明利用HTML指针语法可以链接到被`<A>`元素标识的位置。例如：我们有下面的文档，`content.htm`：

程序清单 8-40

```
<HTML>
  <BODY>
    <A NAME="sentence1">This is the first sentence.</BR></A>
    <A NAME="sentence2">This is the second sentence.</BR></A>
    <A NAME="sentence3">This is the third sentence.</BR></A>
  </BODY>
</HTML>
```

我们从另一个文档可以链接到该文档，index.htm，利用下面的语法：

程序清单 8-41

```
<HTML>
  <BODY>
    <A HREF="content.htm#sentence1">Go to the first sentence</BR></A>
    <A HREF="content.htm#sentence2">Go to the second sentence</BR></A>
    <A HREF="content.htm#sentence3">Go to the third sentence</BR></A>
  </BODY>
</HTML>
```

HTML利用符号 # 来指明其后面的文字指向了目标文档中的一个命名的锚点。浏览器可以通过把显示器调整到目标文档的标识位置，象征性地映射出内容来。注意：像 HTML锚点一样，目标文档不需要直接指定；如果不是这样，所有的 XPointer位置将和文档的基础 URI相关连（通常该文档包含XPointer）。XPointer表达式可以用于URI适用的任何地方。一个识别 XPointer处理器正确地对它们进行处理。例如，XPointer 可以对扩展XLink的定位的URI进一步精炼。

HTML指针的问题

像HTML链接一样，HTML指针存在缺点：

(1) HTML锚点必须先被声明

为了指出HTML文档中一个特定的位置，文档必须含有 锚引用。如果没有这个声明，HTML指针将不能指向HTML文档的一个位置，也就不可能指入一个不具有锚点声明的只读文档。

(2) HTML锚点必须链接到整个文档

没有办法使得一个HTML指针指向一个目标文档的一个部分——而只能指向整个文。如果我们能够定义一个指针使之仅仅指向目标文档的一个部分那就好了。这样处理器就可以显示出该部分而不是整个目标文档。

后面我们将会看到，XML指针机制将能够解决这两个问题。

8.3.2 XPointer 规范

W3C关于XML 指针的规范被称为XPointer，它可以在下面的网址中查到：

<http://www.w3.org/TR/xptr>。

在本书编写的时候，XPointer 还是一个处于最后召集状态的运行草案，这意味着下两个月内它可能成为推荐产品，而该规范内出现的信息将不可能发生重大改变。

注意：XPointer基本上是另一个规范，XPath的延伸。XPath是用于表达XML文档内独立信息项目的W3C通用机制，也是XSLT的一个主要组成部分。XPointer提供了语法，用于说明一个

到达XML文档的链接的地址信息。稍后我们将简要讨论一下 XPath。

1. 在URI中指定XPath

XPointer 可以按照和HTML指针类似的方法被引用。当使用识别 XPointer处理器进行处理的时候,通过给XML文档自身的URI附加一个XPointer程序段标识符,URI可以包含该XML文档内一个位置的引用。和HTML锚点引用不同,应用于URI的指针机制可以通过放置指针机制名称及定位的括弧来识别——对于XPointer来说,总是采用XPointer()的形式。

例如: `http://www.wrox.com/Catalog/catalog.xml#xpointer(book1)` 会指向 `catalog.xml` 文档内具有ID为 `book1` 的元素。我们稍后将从一定的深度讨论程序段标识符的指定方法。

上个例子按照和HTML指针同样的方法指向文档,程序段标识符指明了在映射内容时,文档移动的位置。请求一个需要显示的指定程序段而忽略文档的其他部分也是可行的,这通过使用程序段指示符“|”,而不是“#”实现:

```
http://www.wrox.com/Catalog/catalog.xml|xpointer(book1)
```

这个URI仅仅映射具有ID为 `book1` 的元素(以及任何包含在该元素内的子元素及属性),而忽略了文档的其他部分,这就为筛选大的XML文档,返回和附近位置相关的信息提供一个很好的方法。

2. 程序段标识符可以如何指定

在XPointer中有三种方法指定程序段标识符。其中之一为完全指定,非常复杂,并允许以一定的灵活性来指向一个XML文档。它是建立在W3C XPath推荐标准基础之上,这一点我们在讨论完整规范机制时将会对其进行探讨。首先,我们讨论在XPointer中指定程序段标识符的其他两种方法。

(1) Bare Name程序段标识

为提供我们在HTML中使用的类似功能,一种速记表示法被提出,用来指向具有特定ID的元素。例如,假如我们有下列的文档, `catalog.xml`:

程序清单 8-42

```
<Catalog>
  <Book color="red" ID="book1">
    <Title>IE5 XML Programmer's Reference</Title>
    <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>
</Catalog>
```

我们可以使用下面的句法指向具有ID为 `book1` 的Book元素。

```
#book1
```

如果程序段仅仅是一个ID值,该指针指向具有该ID的元素。注意为了使这个速记表示正常工作,正被指向的文档必须具有指定该元素的ID属性的方案。

(2) 子序列程序段标识

子序列程序段标识符，或者称为 `tumbler` 程序段标识符，允许文档通过遍历子元素树来指向。一些例子可能是有用的。在我们的例子目录文档，`catalog.xml` 中：

程序清单 8-43

```
<Catalog>
  <Book color="red" ID="book1">
    <Title>IE5 XML Programmer's Reference</Title>
    <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>
</Catalog>
```

我们可以利用 URI 指向第二个 `Category` 元素：

`#/1/1/4/2`

该语法可以按照下列方式理解：

- 转到文档中第一个元素（`Catalog` 元素）
- 然后转向该元素的第一个子元素（`book1<Book>` 元素）
- 然后转向该元素的第四个子元素（`<RecSubjCategories>` 元素）
- 然后转向该元素的第二个子元素（`Web Publishing <Category>` 元素）

`Tumblers` 也可以从一个命名节点开始，该节点在 `Bare Name` 程序段标识方法中被指定。下面的 URI 和上面的等价：

`http://www.wrox.com/catalog/catalog.xml#xpointer(book1/4/2)`

(3) 完整 XPointer 规范

完整 XPointer 规范是建立在 XPath 推荐标准基础上的。XPath 提供一种通用方法用于指定文档的某一部分。对于 XPointer 和 XSLT 来说，它是一个基本技术，在后面的章节我们将涉及到。下一部分我们看一下 XPath 表达式的构建方法，以及它们如何用于指向 XML 文档。

8.4 W3C XPath 推荐标准

XPath 是 XSL 和 XPath 工作组成员协同工作的规范，他们意识到两者都需要一种选择一部分 XML 文档的方法。两个工作组都使用并依赖于 XPath 提供的功能。XPath 具有一个推荐状态，意思是它准备生效，并在这个版本中它不会从当前状态发生变化。它可以在 `http://www.W3.org/TR/XPath` 中找到。在这一部分中，我们进一步探讨一下 XPath，它将使你具备更加充分的信息来实现 XPointer 指针。

8.4.1 Location Step

Location Step 将是我们构建 XPointers 时使用最多的构造。它们提供了从 XML 文档中选择节

点的方法。它们都通过上下文节点进行操作，它仅仅是在评估 location step 时作为XML文档的当前节点（如果一个节点没有通过其他某种方法被指定，那么当前节点是文档的根元素）。注意，如果我们在 XPointer 中具有一个以上的 location step，那么将会有有一个以上的当前节点被评估。我们在后面举出几个例子时这一点将更加清楚。

位置节点由三种类型信息构建：轴、节点测试及零个或多个谓词。让我们看一下它们中的每一个及在目标文档的位置节点中的角色。

1. 轴

轴基于上下文节点来分割文档。在评估表达式的时候，它用来定义一个初始区域来应用节点测试和谓词。可能有表 8-2 中所列的这些轴：

表 8-2

轴	定 义
child	包含上下文节点的所有子节点
descendant	包含所有上下文节点的子节点、孙子节点等等
parent	上下文节点的父节点
ancestor	上下文节点的父节点、祖父节点
following-sibling	上下文节点的下面同属节点
preceding-sibling	上下文节点的前面同属节点
following	文档顺序中跟随上下文节点的所有节点，该轴并不包括上下文节点、或者属性及命名空间节点的后代节点
preceding	文档顺序中位于上下文节点之前的所有节点，该轴并不包括上下文节点、或者属性及命名空间节点的祖先节点
attribute	上下文节点的属性节点
namespace	上下文节点的命名空间节点
self	上下文节点
descendant-or-self	后代节点及自身节点的联合
ancestor-or-self	祖先节点及自身节点的联合

一旦我们指定一个文档中进行分析的某个部分，我们将可以使用节点测试进行更加深入细致的研究。

2. 节点测试

节点测试允许从指定的轴中选择特定的元素或者节点类型。下面是几种节点测试：

- 指定一个元素名称，该元素名称仅和具有该名字的节点相匹配。一个 Book 节点测试仅和指定轴中称为 <Book> 的元素匹配。
- 指定通配符，*，来匹配指定轴中所有元素。
- node() 节点测试匹配指定轴中所有节点。
- text() 节点测试匹配指定轴中所有文本元素。
- comment() 节点测试指定轴中所有注释元素。
- processing-instruction() 节点测试匹配指定轴中所有的处理指令元素，而且在括号中给出名字；该测试仅仅匹配具有指定名字的那些处理指令元素。

3. 谓词

谓词对通过轴和节点测试得到的节点集合进行更深入的过滤。一个谓词是一个布尔表达式，用来对通过使用轴和节点测试过滤后得到的结果节点集合中每一个节点进行评估。

你可以使用XPath提供的许多函数对你所需要的节点进行测试。这些函数返回不同形状的结果，如字符串和数字等你可以使用一些比较运算符 `=`，`!=`，`<=`，`<`，`>=`和`>`进行相互之间的比较或者和你所提供的常量进行比较。大一些的表达式可以通过布尔运算符 `and`和`or`进行分离。这样，正被讨论的表达式将传递给 `Boolean()`函数，它将对表达式按照下面方法进行处理：

- 数字-当且仅当它们既不为零，正零，也不为 NaN（看下面）时为 true。
- 节点集合-当且仅当它们为非空时为 true。
- 字符串-当且仅当它们的长度为非零时为 true。
- 对象-如果这些是四种基本类型（数字、节点集合、布尔值和字符串）之外的一种类型，它们将通过一种依据该类型的方法转换成布尔型。

数字是双精度64位IEEE754值，并分为下面几类：

- 正数
- 负数
- 正零
- 负零
- 正无穷大
- 负无穷大
- 不是一个数（“NaN”）

运算符`+`、`-`、`*`、`div`和`mod`可以用于数值运算，括号将会影响到数学运算顺序。

由XPath提供的最简单的函数是 `Position()`，它可用来对正被讨论的元素位置进行简要的评估。例如，我们可以利用它仅选择上下文元素的第一个 `<Book>`子元素，使用下面的谓词：

```
position() = 1
```

我们稍后将会看到谓语如何适用于 XPath表达式。

由于该函数经常被使用，它被隐式地用来表示数值中或者数字常量中函数结果的位置，所以上面谓词可以简写为：

```
1
```

在XPath标准中谓词函数的全集被分为下面几类：

- 节点集合函数
- 字符串函数
- 布尔函数
- 数值函数

我们将依次看一下这几种类型，不过在列出这些函数之前，我们现复习一下它们所需的一些基本定义。

(1) 定义

许多函数在当前的表达式评估上下文环境中以某种方式执行。这个上下文包括下面几种上下文节点（我们已经知道，它是正被评估的节点，或者如果没有指定节点的话是根节点）。

- 上下文大小——上下文中节点的全部合计，使用上下文节点及轴来决定。
- 上下文位置——上下文中节点的当前位置，它小于或者等于上下文大小。
- 变量绑定——在变量名字和价值之间的映射，这里值为对象。
- 函数库——函数名字和函数之间的映射。
- 命名空间声明——从命名空间前缀到URI之间的映射。

任何给定节点有一个字符串-值，每种类型节点的字符串-值都存在，可以来自正被讨论的节点的一部分，或者也可以由它的下一级的字符串-值生成。

一些类型的节点有一个扩展名，形式为：MyNamespace:MyName。其中MyNamespace部分称为命名空间-URI，Myname部分称为本地部分。

现在让我们继续看一下函数。

(2) 节点集合函数

这些函数与多个节点有关，如表 8-3所示。

表 8-3

函数(返回类型，名和参数)	说 明
number last()	返回表达式评估上下文的上下文大小
number position()	返回表达式评估上下文的上下文位置
number count(Node-set)	返回讨论节点集合的节点数目
node-set id(object)	返回一个节点集合，该集合中节点的 ID属性和object参数匹配。object 可以是ID的空格分开的字符串，或者节点集自身，这种情况下，ID的集合由那些节点的字符串-值构成
string local-name(node-set?)	返回提供（按文档顺序）的节点集合中第一个节点（或者上下文节点，如果参数被忽略的话）的扩展名的本地部分。空节点集合或者不具备命名空间的节点将返回空字符串
string-namespace-uri(node-set?)	返回提供（按文档顺序）的节点集合中第一个节点（或者上下文节点，如果参数被忽略的话）的扩展名的命名空间部分。空节点集合或者不具备命名空间的节点将返回空字符串
string name(node-set?)	返回 QName，表示提供（按文档顺序）的节点集合中第一个节点（或者上下文节点，如果参数被忽略的话）的扩展名

(3) 字符串函数

字符串函数将对字符串进行操作或者返回字符串（参见表 8-4）。

表 8-4

函数（返回类型，名和参数）	说 明
string string(Object?)	返回指定对象的stringified 版本，或者当没有指定节点的时候，上下文节点的字符串-值。如果对象是一个节点集合，那么将返回该集合中（按文档顺序）第一个节点的字符串-值。数值返回其字符串表示，布尔值返回 true或false，其他类型的对象返回它们本身支持的值
string concat(string,string, string*)	返回参数的连接结果

(续)

函数 (返回类型, 名和参数)	说 明
boolean starts-with(string, string)	如果第一个字符串以第二个字符串开始, 返回 true, 否则返回 false
boolean contains(string, string)	如果第一个字符串包含第二个字符串, 返回 true, 否则返回 false
string Substring-before(string, string)	如果第一个字符串包含第二个字符串, 返回第一个字符串出现在第二个字符串之前的部分, 否则返回一个空字符串
string substring-after(string, string)	如果第一个字符串包含第二个字符串, 返回第一个字符串出现在第二个字符串之后的部分, 否则返回一个空字符串
string substring(string, number, number)	返回字符串的一部分, 它以第一个数字索引的字符开始, 并由第二个数字指定长度。如果第二个数字缺省, 返回的字符串将包含从第一个数字索引的字符开始, 到原始字符串结束的部分。字符将从开始被索引, 这里, 第一个字符是字符 1
number string-length(string?)	返回指定字符串的字符数目, 如果没有提供字符串, 则返回上下文节点的字符串-值的字符数目
string normalize-space(string?)	去除前后空白空间, 或者使用单个字符来替代空白, 如果没有指定对象, 则对上下文节点的字符串-值进行操作
string translate(string, string, string)	根据第二个字符串和第三个字符串来替代所提供的第一个字符串中的字符。第一个字符串中的每个字符和第二个字符串中的字符相比较, 如果匹配, 将其替换, 替换字符位于第三个字符串中, 它的位置和第二个字符串中匹配字符的位置相匹配。如果第三个字符串中该位置没有字符 (它将会发生在第三个字符串短于第二个字符串的时候), 则把字符去掉即可

(4) 布尔函数

布尔函数均返回布尔值 (参见表 8-5)。

表 8-5

函数 (返回类型, 名和参数)	说 明
boolean boolean(Object)	基于提供的对象返回布尔值, 支配这种转换的规则我们前面讨论过
boolean not(boolean)	如果命题是错误的, 返回 true, 否则, 返回 false
boolean true()	返回 true
boolean false()	返回 false
boolean lang(string)	如果上下文节点的语言 (可以通过其 xml:lang 属性得到) 和指定的语言相同, 返回 true。如果语言不同或者该属性没有出现在该节点中, 返回 false

(5) 数值函数

这些函数基于参数返回数值 (参见表 8-6)。

表 8-6

函数（返回类型，名和参数）	说 明
number number(object?)	将字符串转换成与它们等价的数字（根据 IEEE754 “截取到最相近的值”规则）或者转换成 NaN，布尔值 1（true）或 0（false），使用节点集合的第一个节点（按照文档顺序）转成等价的数值（和字符串中相同的方法），或者其他对象基于它们的类型转为数值
number sum(node-set)	返回由节点集中节点的字符串-值表示的数值总和
number floor(number)	返回不大于命题的最大数值（接近于正无穷大）且是整数
number ceiling(number)	返回不小于命题的最小数值（接近于负无穷大）且是整数
number round(number)	返回和命题值最为接近的一个整数，如果两个整数符合这个要求，则返回接近于正无穷大的值

我们稍后将看一些例子，这样会使命题的使用更为清楚。

4. 把它放在一起

一个位置集合规范采用下面的形式：

程序清单 8-44

```
axis::node test[predicate]
```

这样为了选择上下文元素的前三个 <Book>子元素，我们采用：

程序清单 8-45

```
child::Book[position() <= 3]
```

注意如果这个表达式嵌入在一个标签（例如一个 XLink指向一个 XPointer）中，则避免使用字符<和&:

程序清单 8-46

```
child::Book[position() &lt;= 3]
```

一个XPointer程序段可以由一系列被前向斜杠分割的位置集合定义。它由一些表达式开始，这些表达式建立上下文节点——或者仅一个前向斜杠，表示上下文节点从文档元素开始，或者一个缩写绝对位置（在我们讨论缩写时，将会提到），一个ID定位可以用来建立上下文节点。

这样，为了浏览我们的例子中第一个 <Book>元素中<RecSubjCategories>元素内第二个<Category>元素，我们使用如下的定位：

程序清单 8-47

```
Catalog.xml#/  
  child::Book[position() = 1]/  
    child::RecSubjCategories/  
      child::Category[position() = 2]
```

哟，太容易了。让我们看一些缩写，它们可以用来缩短程序段标识符表达式。

5. 缩写

有些简单的方法用于指定 XPath 常使用的结构，这样你可以在定义 XPointer 程序段标识符时利用它。

如果没有轴被指定，则假设一个子轴。下面两个程序段选择文档元素的第一个 <Book>子元素：

程序清单 8-48

```
Catalog.xml#/child::Book[position() = 1]
Catalog.xml#/Book[position() = 1]
```

属性轴同样可以缩写——我们可以用 @ 来代替 attribute::，由此，这两个程序段均选择第一个 <Book>元素的 color 属性：

程序清单 8-49

```
Catalog.xml#Book[position() = 1]/attribute::color
Catalog.xml#Book[position() = 1]/@color
```

一个通常的结构是 /descendant-or-self::node() 它选择上下文节点的后代节点用于更深层次的处理。不过，这种结构可以缩写成为 //。下面两个例子选择文档中每个 <Title> 节点：

程序清单 8-50

```
Catalog.xml#/descendant-or-self::node()/Title
Catalog.xml#//Title
```

“.” 缩写和 self::node() 相同，下面两个定位指定了上下文节点的 <Title> 后代：

程序清单 8-51

```
self::node()//Title
../Title
```

同样，“..” 缩写和 parent::node() 相同。这样，下面两个定位指定了上下文父节点的所有 <Title> 子节点：

程序清单 8-52

```
parent::node()/Title
../Title
```

最后，在一个谓词中，短语 position()=X 可以使用 X 代替。下面两个定位指定了上下文节点的第二个 <Title> 子节点：

程序清单 8-53

```
Title[position() = 2]
Title[2]
```

如果我们将缩写应用于我们上面的例子，将是：

程序清单 8-54

```
Catalog.xml#/  
  child::Book[position() = 1]/  
  child::RecSubjCategories/  
  child::Category[position() = 2]
```

我们知道可以把它表示成：

程序清单 8-55

```
Catalog.xml#/Book[1]/RecSubjCategories/Category[2]
```

XPath提供了一种灵活的机制，用于将XML文档的独立部分表示成所喜欢的间隔尺寸，甚至减至文本元素的独立字符。XPointer作为该技术的杠杆允许XML文档作为URI定位的部分被指向。并且，XPointer在两个重要方面扩展了XPath的功能。

8.4.2 XPointer对XPath的扩展

XPointer对XPath进行一些扩展，使之允许一些附加的功能。它引入了点和范围的概念作为文档内对位置的描述（除XPath内节点结构之外），并提供了一些函数用来处理这些新的位置索引。

1. 点

XPointer定义一个点位置概念，作为和节点的区别。不过，一个点位置可以是一个节点。它也可以是字符内容内一个特殊的位置（例如，在<Title>元素的文本值的第三个字符）。点对于范围的定义是有用的，我们下面将会看到这一点。

2. 范围

XPointer定义了范围位置的概念，它被定义成为XML结构和两个点间的内容。注意，这样会导致没有格式正规的部分，因为仅有一些元素的部分被包括在范围之内。例如，这种指定范围的能力将会允许一个指针指向目标文档内一个特殊单词的所有事件（例如，作为一个搜索引擎的输出）。一个范围可以这样声明：

程序清单 8-56

```
#xpointer(<locator> to <locator>)
```

这样，一个用来选择始于ID为book1的元素终于ID为book3的元素的范围，包含了落入其间的按照文档顺序的所有内容，可以这样指定：

程序清单 8-57

```
#xpointer(book1 to book3)
```

3. 附加函数

XPointer规范中有一些附加的函数，允许点和范围位置的生成和处理。让我们看一下其中三个最重要的函数：string-range()，here()和unique()。

函数 `string-range()` 搜索目标文档内的文本，并返回所发现的目标字符串事件的范围位置。例如，下面这个定位将返回 `catalog.xml` 文档内字符串 `XML` 的所有事件：

程序清单 8-58

```
catalog.xml#xpointer(string-range(/, "XML"))
```

例如，这个函数可以用于搜索引擎中——每个单词的事件在 `XPointer` 内将是一个范围，允许一个应用程序显示这些事件周围的文本，并可能允许通过 `XLink` 跳至文档内那个位置。

函数 `here()` 返回包含 `XPointer` 自身的元素。该函数允许 `XPointer` 指向其他位置，这些位置和指向文档自身内 `XPointer` 的位置相关。例如，下面的 `XPointer` 将指向 `XPointer` 元素自身的上一级：

程序清单 8-59

```
#xpointer(here()/..)
```

函数 `unique()` 是一个布尔函数，用来决定一个位置集合是否仅仅包含一项。在 `XPointer` 中确认一个单个位置由 `XPointer` 表达式所指定是很重要的（例如，若自动处理正在翻译的文件）。下面两个谓词意义相同：

程序清单 8-60

```
[unique()]  
[count() = 1]
```

8.4.3 XPointer错误

不合适的 `XPointer` 可能会导致三个错误。任何识别 `XPointer` 解析器需要以某种方式处理这些错误（`XPointer` 规范将精确的机制留给开发者设计识别 `XPointer` 处理器）。

1. 语法错误

一个和 `XPointer` 规范的语法限制不相对应的程序段标识符将会导致语法错误。

2. 资源错误

语法正确的程序段标识符，但是附加了一个不合适的资源（例如一个文档不是一个格式正规 XML 文档），将会导致资源错误。

3. 子资源错误

语法正确的程序段标识符，附加了一个格式正规的 XML 文档，但是不能产生一个有效的位置会导致一个子资源错误。

任何识别 `XPointer` 的处理应用程序应该能捕捉这些错误，并按照顺序进行处理。

8.4.4 小结

`XPointer` 为 URI 提供一个机制来指向一个 XML 文档的某一个部分。然而目前 `XPointer` 还没有严格地安装启用，当前的规范正处于最后的集成状态，使用 `XPointer` 的处理器安装启用不久将会成为现实。

8.5 XML程序段交换

随着XML文档存储容量扩充,以及文档平均规模的增加,文档操作变得越来越难处理。如果我们在任何时候能够方便地利用文档的一小部分而不必把整个文档转换并加载进来,这样就很好了。为达到这个目的,W3C为XML程序段交换制定了一个标准(称为XFL,不过现有的工作草案并没有使用这个缩写),定义了XML文档块产生和转换的一些机制。然而,这个标准仅仅停留在工作草案的状态——虽然这个研究组已经结束了这项工作,然而决定推迟执行和推广这个规范,直到其他一些XML规范发展成熟一些以后。

这个关于XML程序段交换的W3C规范可以在如下的网址查到:

<http://www.w3.org/TR/WD-xml-fragment>。

8.5.1 什么是文档程序段

文档程序段在这个规范中被定义为原始文档的正规匹配子集。文档的正确匹配子集包含整个信息项——不过和XML文档不同,不必具有类似意义上的完整形状。必须明确的是正确匹配子集必须包含整体标签(不允许部分标签),即一个正确匹配子集如果具有初始标签,它必须有一个对应的结束标签。后面我们将会看到一些例子。

由服务于程序段的应用程序来决定哪一个信息块对使用者是有用的。一个最普遍的例子就是仅仅传输使用者的程序段所必需的元素,不过其他类型的程序段也是可能的。为了帮助我们了解哪一个文档块是一个程序段,让我们看一下本章前面的一个示意图(参见图 8-13)。

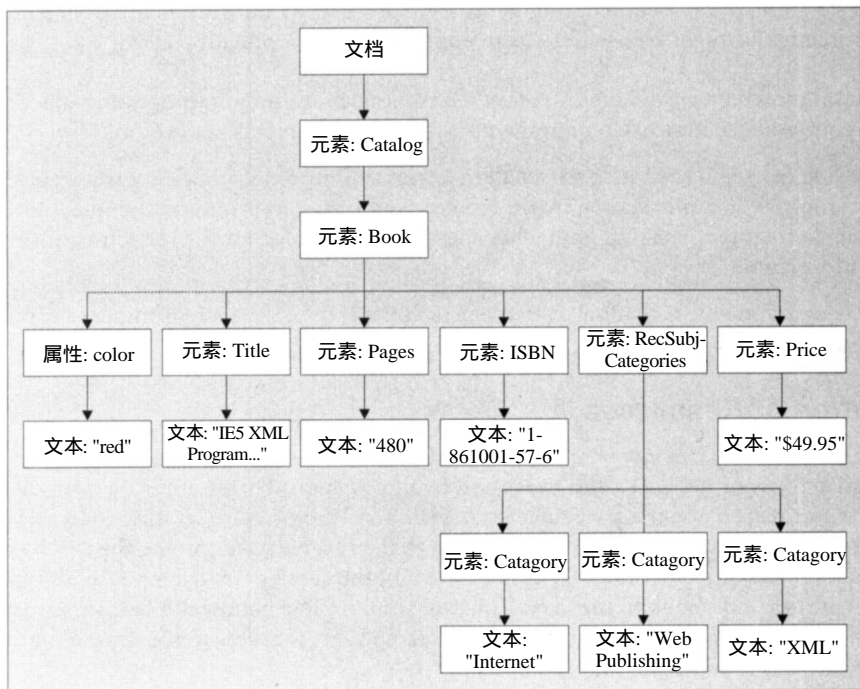


图 8-13

在上面的树状示意图上，通过直线和主树干相连的每个分支被认为可能是程序段，因为每个分支在原始文档范围内连续形式被定义。它们被嵌入在标识内而不能保证是正确匹配，从而属性被定义为程序段。并且邻近的兄弟（和它们的子节点）可以一同被定义为一个程序段，这是由于它们一起同时出现在原始文档中，还可以按照原始（序列化文本）文档：一个有效的程序段作为一个邻近的文本块出现原始文档中，相反的是非邻近的文本决不会是有效的程序段。不过必须指出的是可能有些邻近的文本块不是有效的程序段——上面的规则仍然适用。

上面的示意图中一些有效的程序段可能是：

- <RecSubjCategories>元素和它们的子节点
- 文本项“480”
- 前两个 <Category>元素和它们的子节点

然而，下面这些可能就不是有效的程序段：

- 第一个<Category>元素和第三个<Category>元素
- <color>属性
- <Price>开始标识，但没有</Price>结束标识
- 不具有文本子节点的<Book>元素

8.5.2 文档程序段的一些用途

我们知道了文档程序段的概念，那么我们用它们做些什么呢？让我们看一下程序段能够在应用程序开发中发挥作用的几个途径：

- 使用程序段可以有助于我们节省资源消耗（处理时间、内存需求、网络需求、存储需求等）。
- 使用程序段有助于我们从源文档中隔离出相关的信息子集，只要这些信息是相邻近的。非邻近信息不能以一个单一程序段来发送，避免使程序段对特定信息块的校订来说不是最优的选择。
- 使用程序段可以使我们能够为庞大的XML文档产生一个实时编辑环境。

1. 节省资源

程序段对我们发挥作用最明显的表现就是能够减少网络传输和系统处理的信息量。举例来说，一个使用者需要从Wrox目录中了解某一本书的有关信息。这时发送给接收方的不是Wrox的整个目录（关于所有书籍的详细资料），而这会迫使使用者从整个目录中搜索自己感兴趣的一本书的信息，相反发送方仅发送这一本书的程序段给使用者即可。这样就会减少传输消耗的带宽，进而使接收段的剖析更为快捷，直接得到需求的信息，而根本不用去考虑滤除不必要的信息。

2. 收集信息子集

让我们扩展一下本章前面的例子。假定Wrox在XML目录文档中包含了关于特权，分布数量等的私人信息。不过，如果信息不是邻近的，XFI不允许我们将其作为一个段来传送信息，XFI这里显然不是一个好的解决办法。让我们看下面的例子：

程序清单 8-61

```
<Catalog>
  <Book color="red">
    <Title>IE5 XML Programmer's Reference</Title>
    <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>
    <Royalties>...</Royalties>
    <DistributedCopies>...<DistributedCopies>
  </Book>
</Catalog>
```

如果一个人索要关于《IE5 XML Programmer's Reference》这本书的信息，就会产生一个仅包含兄弟元素作为该书的公有信息的程序段，但不包括私有信息：

程序清单 8-62

```
<Title>IE5 XML Programmer's Reference</Title>
<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>
```

然而，程序段对信息编辑来说不是个好的选择（编辑待打印的信息子集），因为只有邻近的章节被指定——如果我们想发送除去私有信息的整个目录，利用程序段的方法将不能达到这个目的。对这类操作一个更好的办法是XPath和XSLT，在我们讨论那些技术的时候将会看到这一点。

3. 实时编辑和版本控制

仍然看上面的例子，我们假设目录文档内容更深入一点，实际包含了正被讨论的每本书的文本：

程序清单 8-63

```
<Catalog>
  <Book color="red">
    <Title>IE5 XML Programmer's Reference</Title>
    <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>
    <Chapter id="chap1">...</Chapter>
    <Chapter id="chap2">...</Chapter>
    <Chapter id="chap3">...</Chapter>
  </Book>
</Catalog>
```

```
<Chapter id="chap4">...</Chapter>
...
</Book>
</Catalog>
```

下面让我们看一下如何为该书每个章节实现一个实时编辑和版本控制的简单表单。
我们需要某一类数据库来记录跟踪树中每章的当前状态（参见表 8-7）。

表 8-7

书	章	状 态
IE5 XML Programmer's Reference	1	Not checked out
IE5 XML Programmer's Reference	2	Checked out to "jond"
IE5 XML Programmer's Reference	3	Checked out to "kevinw"
IE5 XML Programmer's Reference	4	Not checked out

当然，如果我们对每章实施控制，就可以把它作为 <Chapter>元素上的一个状态属性达到这个目的。

然后我们编写一段应用程序允许不同的作者和编辑检验并对每一章进行检查。当对一作者来说某一章被检查过了，仅把那一个程序段发送给作者：

```
<Chapter id="chap3">...</Chapter>
```

这就允许一个作者对一个章节进行操作，同时另外一个作者可操作另外一章。而不会相互改写和覆盖对方的工作。当一个作者对某一章的修改完成之后，他就可以把修改程序段发送回 Wrox，然后集成到整个原始文档。在数据库中指定更新一个版本表单，指明何人何时编辑过文档将会是一个较简单的事情，现在一些商业 XML 服务器已经可以实现这类功能。

8.5.3 问题：Bare文档程序段并不总是充分的

前面提到我们可利用 XFI 检索到 XML 文档的细节，但是经常接收方需要比此文档程序段所包含的更多的信息来完成工作；它经常需要一些上下文的信息。下面我们要考查一些文档程序段不足以提供充分信息的情形，然后再看一下 W3C 对这个问题的解决办法。

1. 描述什么

经常在一个设计好的 XML 文档中，元素根据它们在文档结构中的不同位置以不同的含义被重复使用，看下面的例子：

程序清单 8-64

```
<Bookstore>
  <Book>
    <Title>IE5 XML Programmer's Reference</Title>
    <Price>$49.95</Price>
  </Book>
  <Coffee>
    <CoffeeType>Double mocha latte</CoffeeType>
    <Price>$2.99</Price>
  </Coffee>
</Bookstore>
```

现在，假如收到了如下的文档程序段：

```
<Price>$2.99</Price>
```

这是一本编程书的价格还是一对 mocha latte 的价格？由于没有充分的信息，所以很难讲。发送方可以发送完整的父元素：

程序清单 8-65

```
<Coffee>
  <CoffeeType>Double mocha latte</CoffeeType>
  <Price>$2.99</Price>
</Coffee>
```

但是现在我们收到一些不想或者不必知道的信息。如果我们知道一些程序段的环境信息，而不必接收上下文所有的内容，这样就非常好了。

2. 利用IDREF和IDREFS

前面的例子中已经提到了 Wrox 已经发布了它的版本控制管理软件。一位作者正在写作第四章，这时她意识到应该为第一章写的内容引入一个索引，她记得为那一段加了一个 ID，但记不起来是什么了。由于仅仅有第四章程序段可以利用，而不能利用 IDREF 映射到第一章的 ID，如果她能有些关于第一章的某类信息就好了——例如利用 IDREFS 标识的各部分的标题——而不必要下载整本书。

3. 验证处理器

假设我们用 DTD 来指定目录例子中的内容：

程序清单 8-66

```
<!DOCTYPE catalog SYSTEM "www.wrox.com/XML/Catalog.dtd">
<Catalog>
  <Book color="red">
    <Title>IE5 XML Programmer's Reference</Title>
    <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>
    <Chapter id="chap1">...</Chapter>
    <Chapter id="chap2">...</Chapter>
    <Chapter id="chap3">...</Chapter>
    <Chapter id="chap4">...</Chapter>
    ...
  </Book>
</Catalog>
```

而且，一位作者想对一个章节进行检验。如果该作者利用验证处理器来查证章节结构，利用原始的 DTD 去做会产生一个问题——原始 DTD 期盼的是一个 <Catalog> 元素、<Book> 元素等等。一个理想的文档版本会包含占位符来满足 DTD，但不包含占位符的内容，从而使带宽消耗和处理时间达到最小化。

8.5.4 解决办法：环境信息

幸运的是 W3C 在制定 XFI 时已经预计到会产生这些问题，所以提供了向接收方传输程序段的同时附加传输环境信息的机制。

1. 什么是环境信息

环境信息是发送到接收方用于帮助描述原始文档中程序段结构位置的信息。XFI 规范在允许为使用者提供什么样的环境信息方面有很大灵活性；发送的信息应该基于使用者的需求。是和直达文档的程序段祖先一样少，或者和原始文档中所有不同的元素标识一样多，这些由 XML 服务器来决定什么样的环境信息对用户处理器有用。

然而规范没有为环境信息如何传输作出明确的解释，它提出了两条建议。第一个建议是向需求方发送两个文件：

- 程序段上下文规范文件，它包含了有关这个元素的所有必要的信息，和包含该程序段的文件索引。
- 程序段本身作为一个独立的文件。

这样就需要产生两个文件，其中一个以某种方式隐藏起来（通过存储在磁盘的文件或者其他机制），它意味着接收处理器能够提供围绕该程序段的信息。

另外一个建议是仅传输一个文件，利用命名空间从真正的程序段内容中分离出环境信息。为了举例说明程序段交换，我们采用前一条建议（稍后我们再来看后一条建议）。

2. 什么可以是环境信息的组成部分

W3C 指定下面一些信息可组成程序段的环境信息：

- 用于原始文档的 DTD 的 URI
- 文档的内部子集的 URI
- 从中提取出程序段的原始文档的 URI
- 原始文档中程序段位置的详细说明书
- 程序段体的祖先信息
- 程序段体的兄弟信息
- 每个祖先的兄弟信息
- 每个祖先或者兄弟的后代信息
- 上面所指的这些元素的属性信息

要注意的是这覆盖了节点树中除包含特定的程序段以外的所有节点；XML 服务器的作者可设计出程序段发生器，用于涵盖与接收方解释程序段这个目的相关的信息的每个部分。

3. 如何描述程序段

W3C 已经为申明程序段产生如下的命名空间：

<http://www.w3.org/XML/Fragment/1.0>

<fcs> 元素（Fragment Context Specifier 的缩写）是程序段上下文详细说明的外包元素。所有的程序段上下文都用 <fcs> 元素括起来，它具有下面四个属性：

- extref——原始文档 DTD 的 URI。

- intref——“扩展”内部子集的 URI。
- parentref——原始文档本身的 URI。
- sourcelocn——原始文档内程序段位置的规范。

值得注意的是此时 W3C 没有为定位指定编码方案，虽然我们希望 W3C 利用 XPointer 来指定位置。

<fcs>元素的子结点应该是原始文档元素树的某一部分（可能会包含属性）。并且，要描述的元素数的部分由发送应用程序决定，而且是基于接收应用程序的需求。在子树中程序段体应该归属的位置，应该包含一个 <fragbody>元素。这个元素具有属性——fragbodyref——应该是正确程序段的 URI 索引。

让我们看下面的例子。假如说我们有如下的目录文件：

程序清单 8-67

```
<Catalog>
  <Book color="red">
    <Title>IE5 XML Programmer's Reference</Title>
    <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>
    <Chapter id="chap1">...</Chapter>
    <Chapter id="chap2">...</Chapter>
    <Chapter id="chap3">...</Chapter>
    <Chapter id="chap4">...</Chapter>
    ...
  </Book>
</Catalog>
```

我们仅把书的 ISBN 发送给接收方，我们会产生一个文档块上下文的详细说明，如下所示：

程序清单 8-68

```
<f:fcs xmlns:f="http://www.w3.org/XML/Fragment/1.0"
  parentref="http://www.wrox.com/Catalog/Catalog.XML"
  xmlns="http://www.wrox.com/Catalog/">
  <Catalog>
    <Book>
      <f:fragbody fragbodyref="http://www.wrox.com/Catalog/ISBN.XML"/>
    </Book>
  </Catalog>
</f:fcs>
```

这里你可以看到程序段上下文详细说明文件包含 <Catalog>元素和<Book>元素，提供了程序段的上下文和第二文件的索引，它准确包含了所需求的程序段。在以 <fragbody>元素引用“http://www.wrox.com/Catalog/ISBN.XML”的程序段自身文件中，我们可得到下面的程序段：

程序清单 8-69

```
<ISBN>1-861001-57-6</ISBN>
```

8.5.5 回顾实例

下面再看一下我们的三个例子，了解如何利用程序段和上下文将信息发送给接收方的。

1. 我描述什么

前面我们试图解决如何断定发送到接收方的价格是和咖啡相关而不是书。如果我们传输下面的程序段和上下文的详细说明：

程序清单 8-70

```
<f:fcs xmlns:f="http://www.w3.org/XML/Fragment/1.0"
  parentref="http://www.wrox.com/Bookstore/Bookstore.XML"
  xmlns="http://www.wrox.com/Bookstore/">
  <Bookstore>
    <Coffee>
      <f:fragbody fragbodyref="http://www.wrox.com/Bookstore/Price.XML"/>
    </Coffee>
  </Bookstore>
</f:fcs>
```

使用者将会知道下面的程序段是 <Coffee>元素的孩子，<Bookstore>元素的孙子：

程序清单 8-71

```
<Price>$5.99</Price>
```

所以对于接收方来说 Price.xml 文件中 <Price> 元素表示什么就变得很明显了。

2. IDREF 和 IDREFS

如果作者所编辑的内容全部在第四章的话，那么她如何为第一章的一个段落添加一个索引 ID 呢？如果我们发送出下面的程序段上下文详细说明：

程序清单 8-72

```
<f:fcs xmlns:f="http://www.w3.org/XML/Fragment/1.0"
  parentref="http://www.wrox.com/FullText/FullText.XML"
  xmlns="http://www.wrox.com/FullText/">
<Catalog>
  <Book>
    <Chapter id="chap1">
      <para ID="IntroXML">
      <para ID="IntroChap1">
      <para ID="IntroChap2">
      <para ID="IntroChap3">
      <para ID="IntroChap4">
      ...
    </Chapter>
    <Chapter id="chap2">
      <para ID="XMLKeywords">
      ...
    </Chapter>
    <Chapter id="chap3">
      <para ID="XMLDTDs">
```

```
...
</Chapter>
<f:fragbody fragbodyref="http://www.wrox.com/FullText/IE5XMLChapter4.xml"
</Book>
</Catalog>
</f:fcs>
```

作者会有一个到本章上下文的引用，它是 IE5XMLChapter4.xml 文件的一个片段：

程序清单 8-73

```
<Chapter id="chap4">
...
</Chapter>
```

现在，作者在程序段上下文详细说明文件中拥有可以反馈到每一章各位置的 ID，可以发送——也就是说可以为 IntroChap4 添加一个 IDREF 用于回指第一章中给出的关于第四章的信息。

3. 验证处理器

为了保证验证处理器运行良好，我们引入一个除 character 数据外的所有必备元素的副本——最优元素将被剔除。这样对这个例子来说，我们可以传输下面的程序段上下文详细说明：

程序清单 8-74

```
<f:fcs xmlns:f="http://www.w3.org/XML/Fragment/1.0"
extref="http://www.wrox.com/XML/Catalog.dtd"
parentref="http://www.wrox.com/FullText/FullText.XML"
xmlns="http://www.wrox.com/FullText/">
<Catalog>
  <Book>
    <Title></Title>
    <Pages></Pages>
    <ISBN></ISBN>
    <RecSubjCategories>
    </RecSubjCategories>
    <Price></Price>
    <f:fragbody fragbodyref="http://www.wrox.com/FullText/IE5XMLChapter4.XML"/>
  </Book>
</Catalog>
</f:fcs>
```

它提供了具有使下面程序段有效的所需元素的处理器：

程序清单 8-75

```
<Chapter id="chap4">
...
</Chapter>
```

需要指出的是我们去掉了 <Category> 元素（这是由于我们在 DTD 中声明 <Category> 是可选的）和附加的 <Chapter> 元素（这是由于在 DTD 中 <Chapter> 被指定具有一个或多个事件）。收到这两个文件时，处理器会在适当的位置分析程序段体，就好像它是一个外部解析实体，去掉上下文块的外包，并根据 DTD 来验证文档。

8.5.6 如何传输程序段

现在我们已经给出了一些程序段和程序段上下文详细说明，那么如何将它们发送给接收方呢？W3C没有在XFI规范中强制程序段的传输方式，但它提供了两种不同的方法使识别片段接收处理器可以收到传输。

1. 分离文件机制

我们已经看了程序段处理两个文件的方法。一个文件包含程序段本身，另外一个程序段上下文详细说明文件。为了发送信息给接收方，发送了程序段上下文详细说明文件。识别片段接收处理器剖析了程序段上下文详细说明文件，并到 fragbodyref属性中的位置读取程序段本身信息（参见图8-14）。

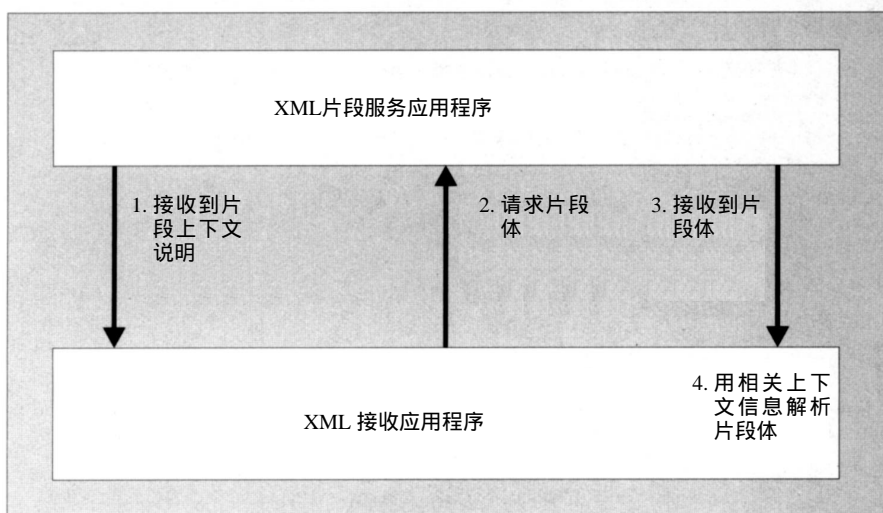


图 8-14

不过这种方式需程序段服务器应用程序产生两个文件，其中一个文件被隐藏起来（通过将文件存于硬盘或者其他机制）。而且，这需要在网络上多一个来回，而这是我们尽可能要避免的。

2. 计划包机制

W3C声明了程序段传输在XFI规范范畴之外，不过提供了非正式（建议采用但并不必须的W3C码）方法，令程序段和程序段体一起打包作为一个文件。最主要的一点就是建议采用一个新的命名空间来提供包含程序段体的元素。就像如下形式来取代含有程序段上下文详细说明的两个文件：

程序清单 8-76

```
<f:fcs xmlns:f="http://www.w3.org/XML/Fragment/1.0"
  parentref="http://www.wrox.com/Bookstore/Bookstore.XML"
  xmlns="http://www.wrox.com/Bookstore/">
  <Bookstore>
```

```
<Coffee>
  <f:fragbody fragbodyref="http://www.wrox.com/Bookstore/Price.XML"/>
</Coffee>
</Bookstore>
</f:fcs>
```

程序段体形式如下：

程序清单 8-77

```
<Price>$5.99</Price>
```

这样我们就仅采用一个文件：

程序清单 8-78

```
<p:package xmlns:p=http://www.w3.org/XML/Package/1.0
  xmlns:f="http://www.w3.org/XML/Fragment/1.0"
  xmlns="http://www.wrox.com/Bookstore/">
  <f:fcs parentref="http://www.wrox.com/Bookstore/Bookstore.XML">
    <Bookstore>
      <Coffee>
        <f:fragbody/>
      </Coffee>
    </Bookstore>
  </f:fcs>
<p:body>
  <Price>$5.99</Price>
</p:body>
</p:package>
```

值得注意的是我们不再为程序段体指明位置——它被认为假定在 `<p:body>` 元素。这个方案仅需要一个来回即可到达服务器（见图 8-15）。

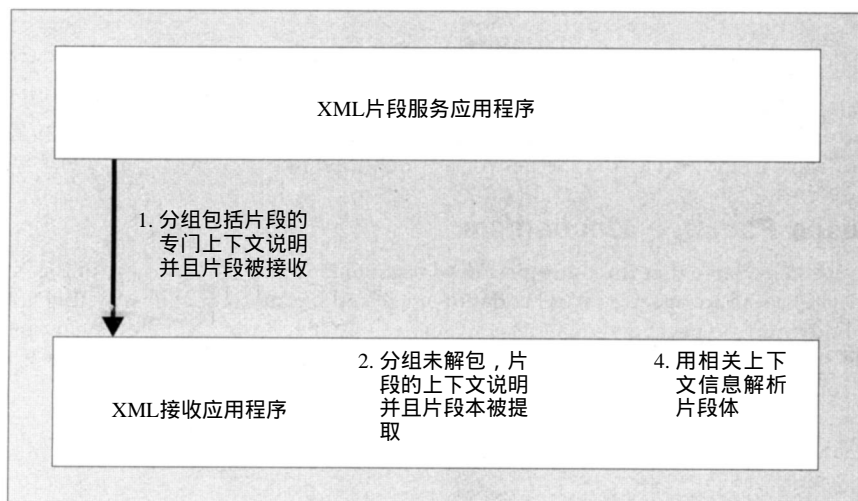


图 8-15

这个技术使我们减少了达到服务器的循环次数，并且减轻了服务器应用程序本身的复杂性。

8.5.7 小结

XML程序段交换提供了一种方法，将一部分 XML文档和足够的环境信息叠加在一起发送到接收方，使得这些文档对接收方真正有用。让我们看一下程序段发挥作用的集中方式——例如允许多个作者同时操作文档，或者减少通过网络传输并经由 XML处理器处理的无用信息。不过返回的程序段必须对应于原始文档的邻近元素，这就可能使对原始文档的操作变得复杂。不过目前还没有适用性广的识别片段处理器，熟悉程序段和上下文的概念当会使你为程序段做准备当它们开始出现的时候。将各种 XML技术集中起来，XFI或许可以被用于传输通过查询或者通过利用XLink生成的程序段连结所产生的部分程序段文档。

8.6 查询

我们已经了解了如何从 XML文档链接并指向它们的不同部位。如果能够利用一种查询语言来获取某一部分的XML文档并对之进行操作，就非常方便了。这下面的这一部分里，我们将讨论一下查询XML文件并从中提取内容的几种方法。

8.6.1 什么是查询语言

如果你使用过关系数据库，可能熟悉一种形式的查询语言：SQL(结构化查询语言)，允许使用者获得数据库中包含的信息，并可以以几种方法对数据进行操作。在你深入了解查询 XML文档和原始资料这个主题之前，我们将看一下查询语言是如何工作的。那么现在让我们看一下一些可能利用SQL的操作，那么对XML来说使用一些假定的查询语言应该是可能的。

1. 返回信息的行方式约束

当我们从数据库中返回信息时最基本的一件事情是仅返回与特定规则相匹配的行来对行记录进行过虑，从而实现对信息进行限制。这可以利用 SQL语言中的SELECT语句的WHERE子语句来实现这个目标：

程序清单 8-79

```
SELECT *  
  FROM Book  
 WHERE Title = "Professional XML"
```

2. 返回信息的列方式约束

我们也可以通过从数据库中仅返回感兴趣的列来对列进行限制，达到对返回信息进行约束的目的。可以在SELECT语句中使用下面方式指定：

程序清单 8-80

```
SELECT Title, Price  
  FROM Book
```

3. 总结返回信息

我们也有必要对返回的信息进行归纳总结，把几个不同行的一些信息块集成在一起形成一

条信息。SQL提供了一些可用的函数来实现在这个功能：

程序清单 8-81

```
SELECT AVG(Price)
FROM Book
```

4. 对返回信息进行排序

对一个查询返回的信息进行重新排序的能力是非常重要的，SQL提供了ORDER BY语句做为实现排序的方法：

程序清单 8-82

```
SELECT Book
ORDER BY Price
```

5. 内部连接

由于SQL表的本身性质决定，我们常从多于一个的表中返回信息来得到一个有意义的结果。在SQL中这个操作通过inner join 和equijoin来实现：

程序清单 8-83

```
SELECT Book.Title, RecSubjCategories.Category
FROM Book INNER JOIN RecSubjCategories ON Book.BookID =
RecSubjCategories.BookID
WHERE Book.Author = "Kevin Williams"
```

6. 外部连接

在SQL中，当我们从多于一个的地方提取信息的时候，常希望从一个表中返回信息，而与此相关的信息在其他表中并不存在。例如，我们希望返回所有书籍和它们所属的类别，但是我们也希望能够得到那些没有所属类别的所有书籍，这在SQL中可以通过outer join操作来实现：

程序清单 8-84

```
SELECT Book.Title, RecSubjCategories.Category
FROM Book
LEFT OUTER JOIN RecSubjCategories
ON Book.BookID = RecSubjCategories.BookID
WHERE Book.Author = "Kevin Williams"
```

7. 表内容操作

利用SQL命令对数据库中信息进行操作是可行的，可以在逐行的基础上对信息进行添加、修改、删除等操作，这些命令分别是：INSERT、UPDATE和DELETE：

程序清单 8-85

```
INSERT RecSubjCategories (BookID, Category)
VALUES (1, "XML")

UPDATE Book
SET Title = "Pro XML"
WHERE BookID = 1
```



```
DELETE Book
WHERE BookID = 1
```

这些功能确实超出我们假定 XML 查询引擎的范围，XML 可以通过其他的方式来达到这个目的，例如 DOM。这个功能对一个强大的查询语言来说不是基本要求，而可以很方便地实现。

8. 从不止一个的源获得信息

SQL 可以实现经常从不止一个的地方返回信息的功能——有代表性的就是同一台机器上存储有不同数据库的信息。例如，在 SQL 服务器的 SQL 执行中，这可以通过为表名加上所属数据库名称的前缀来实现：

程序清单 8-86

```
SELECT WorkDB.Book.*
FROM WorkDB.Book, PublishedDB.Book
WHERE WorkDB.Book.BookID = PublishedDB.Book.BookID
```

9. 程序处理

在大部分 SQL 实现中，会产生一些形式的存储查询和存储过程，SQL 语法虽还坚持这一点，但是处理是通过程序的方式。看一下下面存储程序的例子：

程序清单 8-87

```
CREATE PROC DefaultCategories (@BookID integer)
AS
BEGIN
    DELETE RecSubjCategories
    FROM RecSubjCategories INNER JOIN Book ON RecSubjCategories.BookID =
        Book.BookID
    WHERE Book.BookID = @BookID

    INSERT RecSubjCategories (BookID, Category)
    VALUES (@BookID, "No category specified")
END
```

这个存储程序实现两个功能：首先，删除指定书籍的所有目录；其次，它为这本书添加一个缺省目录。有很多结构可以使用，例如临时表、变量和指针，它们可以使存储程序成为操作关系数据的强大工具。

下面我们将会看到对于 XML 可行的这些技术是如何实现这些相同的功能。不过，在我们了解那些技术之前，我们有必要知道和关系数据库不同的 XML 文档是如何构造的。

8.6.2 关系型数据库和 XML 文档之间的区别

在我们对 XML 文档进行查询的时候，为了解即将面临的问题，有必要简要地回顾一下 XML 文档的结构。首先为了对比的需要，我们来看一下 SQL 的结构。

SQL 数据结构包含一系列表。每个表具有一列或多个列。例如，下面是我们正要讨论的 Book 表和 RecSubjCategories 表的 SQL 表生成脚本：

程序清单 8-88

```
CREATE TABLE Book (
    BookID int,
```

```
Title varchar(100),
Pages integer,
ISBN varchar(15),
Price varchar(15) )

CREATE TABLE RecSubjCategories (
  RecSubjCategoriesID int,
  BookID int,
  Category varchar(100) )
```

表8-8和表8-9显示了这些表的样本数据，第一个是 Book表。

表 8-8

BookID	Title	Pages	ISBN	Price
1	IE5 XML Programmer's Reference	480	1-861001-57-6	\$49.99

第二个是 RecSubjCategories表。

表 8-9

RecSubjCategoriesID	BookID	Category
1	1	XML
2	1	Web Publishing
3	1	Internet

关于这些数据结构，下面几问题很重要。

1. SQL行具有单一的标识符

在我们的样本表中第一个元素是该表的主关键字。一个设计完整的 SQL数据库总是为每个表定义一个主关键字，它唯一地定义了表中的每个记录。通常主关键字是任意的，一般为系统设定的数字（通过一个序列或其他机制）。相比而言，XML元素（类似于数据库中的表格）不是用关键字来定义（不过，XML提供了一个机制：ID属性，来控制信息）的。例如，下面 XML中的文档是完全可以被接受的：

程序清单 8-89

```
<NutritionHistory>
  <SingleDay>
    <Date>January 7, 2000</Date>
    <Beverage>Jolt Cola</Beverage>
    <Beverage>Jolt Cola</Beverage>
    <Beverage>Jolt Cola</Beverage>
    ...
  </SingleDay>
</NutritionHistory>
```

值得注意的是三个 <Beverage>元素正好具有相同的内容。在文档被解析的时候，XML可以根据它们被遇见的次序区别开这些元素。这就带来数据库和 XML之间的第二个区别。

2. SQL行并不反映顺序

关系型数据库并不反映出表中出现的信息的任何顺序。例如，在我们检验这些样本结构之前，为 Book设定了三个“RecSubjCategories”：Internet、Web Publishing和XML。这些类项的主

题被用在每本书的背面，用来向书商说明这些书应该摆放在书架的哪个地方，在计划订货时可以被使用。在XML文档中，它们被遇见时的顺序可以反映一种顺序——这本书首先和XML书籍最相符，然后作为选择的是适用关于网络出版的一些信息，如果前面两种都不适用，则适用于网络书籍。在数据库中对信息进行排序的时候，这些顺序信息将会丢失。如果这些信息得以保持，赋予RecSubjCategories记录的一些列排序信息将被用来揭示类别的优先顺序。

我们的查询语言应该了解XML文档出现的信息的顺序特征，并根据使用这种语言查询的任何结果来保存信息。而且元素的位置适用于查询引擎，这样可以利用那个信息来对结果进行过滤——例如，对于给定的<Book>元素，查询引擎能够返回第二个<Category>元素。

3. SQL结构并不提供层次封装

从概念上来说，关系型结构并不是层次性的——在前面的例子中，它并没有提供给我们对<Book>元素内部<RecSubjCategories>元素封装的方法。相反，<RecSubjCategories>元素必须含有指回它们所属元素的指针——在我们的例子中，表<RecSubjCategories>中BookID列就回指到“含有”RecSubjCategories信息的书记录。XML这种指针机制可以让我们根据需要定义一对一或者一对多的引用，甚至超出允许我们选择指向数据库结构中任意其他的元素。

这些关于指针的话题会使你想起XML中的等价物——利用IDREF或者IDREFS属性来回指具有给定ID的元素。我们可以设想一个<Book>和<RecSubjCategories>的DTD程序段看起来如下所示：

程序清单 8-90

```
<!ELEMENT Catalog (Book*, RecSubjCategories*)>

<!ELEMENT Book (ID, Title, Pages, ISBN, Price)>
<!ELEMENT ID ID>
<!ELEMENT Title #PCDATA>
<!ELEMENT Pages #PCDATA>
<!ELEMENT ISBN #PCDATA>
<!ELEMENT PRICE #PCDATA>

<!ELEMENT RecSubjCategories (ID, BookID, Category)>
<!ELEMENT BookID IDREFS>
<!ELEMENT Category #PCDATA>
```

在这种结构形式下，上面的例子将成为如下所示：

程序清单 8-91

```
<Catalog>
  <Book ID="B1">
    <Title>IE5 XML Programmer's Reference</Title>
    <Pages>480</Pages>
    <ISBN>1-861001-57-6</ISBN>
    <Price>$49.99</Price>
  </Book>
  <RecSubjCategories ID="R1" BookIDs="B1">
    <Category>XML</Category>
    <Category>Web Publishing</Category>
    <Category>Internet</Category>
  </RecSubjCategories>
</Catalog>
```

为了找出一本书的类别序列，我们有必要寻找具有回指该书的 BookID 的 <RecSubjCategories> 元素。相反，为了找出包含特定类别序列的书籍，我们有必要搜索拥有被正讨论的 <RecSubjCategories> 元素的 BookIDs 所指向 ID（或 IDs）的书（或多本书）。关系型数据库天生就是利用连接（join）来解决这类问题，能够使元素之间连接的界限很清晰，而标准的 XML 工具（如 DOM 或者 SAX）的横向连接需要额外的工作。

如我们所知，XML 允许信息封装为父亲的孩子。这仅典型地用于表达一对一或者一对多的关系（把同一个孩子嵌入不止一个的父母中，会造成信息的重复，在 XML 文档中利用 ID-IDREFS 机制进行描述将会好一些）。我们例子的最初版本利用封装显示了类别信息是书籍信息的组成部分。XML 技术，如：DOM、SAX、XLink、XPointer、XPath 和 XSLT 利用节点树的形式表达父母 - 孩子的关系更加能够发挥作用。我们的 XML 查询语言能够浏览父母 - 孩子关系和 ID-IDREF 关系——以任一方向——能够保存查询得到的信息。

4. XML 混淆了属性和纯文本内容

在 XML 文档中，将属性和带有自己本身的文本内容和纯文本内容的特定元素相关联是可能的。下面是两种可能的情形：

程序清单 8-92

```
<Catalog>
  <Book Title="IE5 XML Programmer's Reference"
    Pages="480"
    ISBN="1-861001-57-6"
    Price="$49.99"/>
</Catalog>

<Catalog>
  <Book ID="B1">
    <Title>IE5 XML Programmer's Reference</Title>
    <Pages>480</Pages>
    <ISBN>1-861001-57-6</ISBN>
    <Price>$49.99</Price>
  </Book>
</Catalog>
```

这两种形式在句法构成上是不同的，但从语义上来说是完全相等的——书籍有标题，无论它们是作为书籍的属性还是作为书籍的纯文本子元素。每个样本的这个信息可以被存储在以前定义的 Book 表中：

程序清单 8-93

```
CREATE TABLE Book (
  BookID int,
  Title varchar(100),
  Pages integer,
  ISBN varchar(15),
  Price varchar(15) )
```

“文本元素与属性”的辩论仍在继续——有人尝试为 XML 内容添加标准的形式（例如 Microsoft 的 BizTalk 对象的规范形式可以在网址 <http://www.biztalk.org/resources/canonical.asp> 中查到），但是 W3C 标准并没有把 XML 文档限制为一种形式或其他形式。我们的查询引擎应该对

两种情形都能够进行操作。

5. XML允许元素混合模型

XML允许元素被定义成具有混合内容模型——它们可以包含文本信息和子元素。这里有个例子：

程序清单 8-94

```
<!ELEMENT Book(#PCDATA, Title, Pages, #PCDATA, ISBN, #PCDATA)>
```

符合上面规范的<Book>元素应该是下面这种形式：

程序清单 8-95

```
<Book>
  Here's some random text.
  <Title>Professional XML</Title>
  <Pages>480</Pages>
  Some other text appears here.
  <ISBN>1-861001-57-6</ISBN>
  And finally another fragment of text.
</Book>
```

如果考虑对段落进行标记，这些需要就变得很明显了：一些单词或短语可能是粗体、斜体，但是段落中大多数文本并没有额外的标注。如果你过去习惯于利用数据对象工作，这多少会造成错觉——为了在关系型数据库中描述上述信息，一些语义构造有必要添加进关系型数据库中（如Book表中value1、value2、value3列）。如果XML有更加灵活的结构，问题就更深入了：

程序清单 8-96

```
<!ELEMENT Book((#PCDATA | Title | Pages | ISBN)*)>
```

在这种情形下，我们在<Book>元素中可以有任何数量的文本块。

我们用于访问XML的任意一种查询语言必须能够处理像这样的内容规范。因此，了解了什么是基本的查询语言所必须解决的问题之后，让我们看一下XML查询能力进展到什么程度。

8.6.3 XML查询语言的发展历史

早期的XML使用者很快就认识到查询语言的必要性。为了达到这个目的，两个研究小组在1998年开始研究并对XML文档查询机制，并向W3C提出方案。在我们讨论查询语言目前的状况之前，让我们看一下XML查询语言的发展历史。

1. XML-QL

W3C提交了一种称为XML-QL的查询语言。这个方案的作者按照数据库的思路来解决问题，采用了用于访问层次型数据库和关系型数据库的相同技术。XML-QL指定了一个XML文档的样本程序段，提供了构建机制允许输出同类信息，而不用考虑开发者的愿望。在继续讨论之前，让我们先看一个例子。

设想要创建一个文档，在样本XML目录中列出所有书籍的标题，输出结果按如下形式：

程序清单 8-97

```
<Titles>
  <Title>IE5 XML Programmer's Reference</Title>
  <Title>Designing Distributed Applications</Title>
  ...
</Titles>
```

在XML-QL，我们将使用下面的查询：

程序清单 8-98

```
CONSTRUCT <Titles> {
  WHERE
    <Book>
      <Title>$t</Title>
    </Book> IN "http://www.wrox.com/XML/catalog.xml"
  CONSTRUCT
    <Title>$t</Title>
} </Titles>
```

这里你可以看到和SQL的相似之处。利用WHERE子句我们可以对正在操作的信息进行限制，CONSTRUCT子句可以用来创建产生的输出。这些子句是可以嵌套的，为我们的信息提取和信息描述提供很大的灵活性。

尽管XML-QL是灵活的，但它有两个缺点：

- XML没有规定保持次序——在XML-QL处理器中结果输出并不保证和原始文档具有相同的顺序。这对于XML数据文档来说可能不是一个大问题，然而对于XML文本标识文档来说就相当严重了（我们可以想象阅读一本段落顺序混乱的书！）
- XML没有规定保持结构——你可以注意在前面的例子中，必须通过指定使用的标识来产生结果文档的整体结构——实际上，在查询中重新产生结构。如果我们尝试将大的文档减至为我们所感兴趣的那一部分（例如，一个给定作者的所有著作），将结构信息保持下来就非常好了——我们就不用建造书籍的整体子结构，仅仅把原始文档的结构镜像下来就可以了。

在XML-QL方案提交到W3C的同时，另外一种建议也被提出来了，这是一种采用更多方向标识的办法：XQL。

2. XQL

XQL从结构化文档的前景考虑来解决问题。它在设计时考虑尽可能在提供方法减少内容的同时保持原始文档的结构和顺序。让我们回顾一下前面一部分的例子。（若熟悉Microsoft的XSL的安装使用，请记住该讨论仅针对原始XQL版本——我们会逐渐过渡到流行状态。）

利用XQL，通过查询从目录中生成一系列标题将是下面这种形式：

```
//Title
```

输出结果如下：

程序清单 8-99

```
<xql:result>
  <Title>IE5 XML Programmer's Reference</Title>
  <Title>Designing Distributed Applications</Title>
```

```
...
</xql:result>
```

注意，利用 XQL 原始文档的顺序能够保证得以维持。上面所示的文本文档由容器 `<xql:result>` 包起来。用于返回节点集的 XQL 处理器简单地返回一列有序的 `<Title>` 元素节点。

XQL 同时保持了层次信息。例如，如果我们选择抽取称为 “IE5 XML Programmer's Reference” 的书，我们可通过如下查询实现：

```
//Book[Book.Title="IE5 XML Programmer's Reference"]
```

查询输出如下：

程序清单 8-100

```
<xql:result>
<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>
</xql:result>
```

注意所有被选 Book 元素的嵌套内容得以保持下来，然而如果利用 XML-QL 我们必须在子句中指定所有的子元素。

XQL 能够解决利用 XML-QL 所产生的一些问题，不过它在下面两个方面存在缺陷。

- XQL 不能提供信息区分——XQL 的设计是以节点操作，而不是依靠信息，当两个元素具有相同的内容时，它就没有自身解决办法进行鉴别。例如，利用 XQL 不能在 XML 目录中产生作者的不同列——XQL 会对原始文档中的每个 `<Author>` 节点产生一个 `<Author>` 节点，而不理会节点的内容。
- XML 不能关系——处理关系型信息一个普通的技术就是 pivot 关系。例如，我们有一列书籍和它们各自的作者，但是我们希望产生一列由所有作者和他们出版的书的列表。由于 XQL 能够对层次保持，利用 XQL 进行查询是不可能的，它最初设计时就是这样。

虽然对于访问层次型信息，XML 是最自然的方法，它不能像 XML-QL 一样灵活地处理数据。综合的灵活性和结构化访问机制将是较理想的查询语言。

3. XSLT 和 XPath

尽管缺少查询标准，利用 W3C 现有的两种推荐方案：XSLT 和 XPath，来产生解决问题的办法还是可能的。你无疑认识了 XQL 的查询模式——XQL 是 XPath 的最直接的祖先。XPath 维持了原始文档的层次性和结构，适用于访问 XML 文档的独立节点。XSLT（下一章我们将会详细学习）

适用于对查询结果进行处理，构建新的元素，在有必要的时候（采用和 XML-QL 很类似的方法）从新组织结果元素。我们将会明白，综合 XSLT-XPath 将允许开发者随意而方便地对源文档的信息进行操作和重组。

8.6.4 使用 XPath 和 XSLT 查询 XML 文档

目前可以使用的用于查询 XML 文档最好的技术是 XSLT，XSLT 使用 XPath 从文档中滤除结点，不过 XSLT 用来以任何数目的形式来描述数据。如果你熟悉利用 XSLT 进行 XML 到 HTML 的转换（将会在第 9 章学到更多关于 XSLT 和转换的知识）的话，XSLT 也可以用来对任何结构进行转换——包括 XML。这一部分我们将看一下如何利用 XSLT 将一个 XML 文档转换成另一个，表达了一些我们在本章前面讨论的查询需求。

利用 XT 来测试 XSLT 和 XPath 查询

随着 XSLT 和 XPath 的兴起，然而常用的 XML 库如 Microsoft 和 SUN 的安装使用都不完全支持 XSLT 和 XPath，这些工作草案只能称为建议性的。Microsoft 已经允诺在下一个 MSXML 版本中完全支持 XSLT 和 XPath，预计可以在 2000 年 1 月份可以出来。到那时，James Clark（XSLT 规范的编者，XPath 规范的编者之一）会提供出 XSLT 可安装在 Java 类库或者包的集合，并在 Win32 环境下可以运行。这些可从下面网址下载：

<http://www.jclark.com/xml/xt.html>

本章这一部分的所有例子都用 XT 进行了测试。XT 的下载和使用须知可以在附录 G 中找到。本章所有例子都是在我们标准的 catalog.xml 文件上操作的：

程序清单 8-101

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!--===== The Wrox Press Book Catalog Application =====>

<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>
```

(1) 信息的行方式约束

在XML中，信息的行方式约束等价于基于元素内容的元素约束。如我们所知，XPath表达式就是被用来寻找具有给定值的字段。

因此，如果我们想返回目录中作者为 Alex Homer的书籍，我们可以使用下面的格式：

程序清单 8-102

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/Catalog">
    <xsl:copy>
      <xsl:for-each select="//Book[Authors/Author='Alex Homer']">
        <xsl:copy>
          <xsl:apply-templates name="childnodes"/>
        </xsl:copy>
      </xsl:for-each>
    </xsl:copy>
  </xsl:template>
  <xsl:template name="childnodes" match="*">
    <xsl:copy>
      <xsl:apply-templates name="childnodes"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

注意上面的格式中有两个模板。名字为“childnodes”的模板被用来递归地写出节点的后代元素，而这些节点正被写入输出XML文档——<xsl:copy>用来复制当前节点，同样childnodes模板把遇见的所有子节点复制到文件中。我们会在下面的几个例子中使用这种技术。

用来执行行选择的这部分模板是位于第一个模板中的块 <xsl:for-each>。这个块仅选择文档中待输出的那些书——位于方括号内的，和XPath表达式相匹配的那些（如我们所知，选择带有文本Alex Homer的<Author>元素）。在遇见我们的catalog.xml文档时该脚本输出会返回 Alex 编著的三本书：

程序清单 8-103

```
<?xml version="1.0" encoding="utf-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>
  <Price>$49.99</Price>
</Book>
<Book>
  <Title>Professional ASP 3.0</Title>
  ...
</Book>
```

```
<Book>
  <Title>Beginning Components for ASP</Title>
  ...
</Book>
</Catalog>
```

(2) 返回信息的列方式约束

对于XML来说，通过对文本水平的元素或者由一个样式表的返回属性进行约束可以达到对返回信息进行列方式约束的目的。假如说我们想从整个书籍清单中提取一些书的标题，而 Alex Homer是这些书的有贡献作者，我们可以使用下面的样式表：

程序清单 8-104

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/Catalog">
    <xsl:copy>
      <xsl:for-each select="//Book[./Authors/Author='Alex Homer']">
        <xsl:copy>
          <xsl:apply-templates select="Title" name="childnodes"/>
        </xsl:copy>
      </xsl:for-each>
    </xsl:copy>
  </xsl:template>
  <xsl:template name="childnodes" match="*">
    <xsl:copy>
      <xsl:apply-templates name="childnodes"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

这里，我们把另外一个XPath表达式作为孩子来从被选书籍中选择标题——仅返回被选书的标题。很明显，在第二个select指令中可以修改XPath来改变文本元素——或者列——正在被返回的。上面的查询返回下面的输出：

程序清单 8-105

```
<?xml version="1.0" encoding="utf-8"?>
<Catalog>
  <Book>
    <Title>IE5 XML Programmer's Reference</Title>
  </Book>
  <Book>
    <Title>Professional ASP 3.0</Title>
  </Book>
  <Book>
    <Title>Beginning Components for ASP</Title>
  </Book>
</Catalog>
```

注意我们在中间增加了空白以使XT的输出更具可读性。

(3) 返回信息概括

XML信息可利用XPath的内置集成函数来进行归纳总结。例如，我们如果想获取部分由 Alex Homer完成的每本书的总页数，可以采用下面的样式表：

程序清单 8-106

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/Catalog">
    <xsl:copy>
      <xsl:element name="totalPages">
        <xsl:value-of select=
          "sum(//Book[./Authors/Author='Alex Homer']/Pages)"/>
      </xsl:element>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

这里我们把使用<xsl:value-of>元素中XPath表达式选择出的<Pages>元素的数量值用sum()函数累加起来就可以了。注意我们利用 XPath表达式按照同样的方式对价格进行操作，因为在目录中定义<Price>元素时，加了一个前缀\$。由于XPath提供了一些用于处理字符操作的函数，它不能对按照SQL命令相同的方式对集合进行操作，那样会使嵌套操作（例如，去掉前导美元符，并把结果数值累加起来）变得很困难。

把这个格式页面应用于catalog.xml文档之后。我们有：

程序清单 8-107

```
<?xml version="1.0" encoding="utf-8"?>
<Catalog>
  <totalPages>3036</totalPages>
</Catalog>
```

(4) 排序

现在我们把查询延伸一下，来查询是 Alex Homer是作者之一的那些书籍。我们也对返回的书按照标题进行排序（按字母顺序）：

程序清单 8-108

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/Catalog">
    <xsl:copy>
      <xsl:for-each select="//Book[./Authors/Author='Alex Homer']">
        <xsl:sort select="Title"/>
        <xsl:copy>
          <xsl:apply-templates select="Title" name="childnodes"/>
        </xsl:copy>
      </xsl:for-each>
    </xsl:copy>
  </xsl:template>
  <xsl:template name="childnodes" match="*">
    <xsl:copy>
      <xsl:apply-templates name="childnodes"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

我们只需把<xsl:sort>元素添加到<xsl:for-each>元素。它指明了<xsl:for-each>元素的Select属性返回的节点集按照Title排序。

这是我们的转换输出：

程序清单 8-109

```
<?xml version="1.0" encoding="utf-8"?>
<Catalog>
  <Book>
    <Title>Beginning Components for ASP</Title>
  </Book>
  <Book>
    <Title>IE5 XML Programmer's Reference</Title>
  </Book>
  <Book>
    <Title>Professional ASP 3.0</Title>
  </Book>
</Catalog>
```

(5) 内部连接

由于XML文档具有像指针一样的包含方式，将内部连接的概念应用于它们就不太必要了——如果信息块已经包含在你所具备的那一部分，就没有必要再去查找了。另一方面，如果我们利用ID和IDREF属性把元素之间连接起来，我们则需要这个功能。XPath允许我们沿着节点树来将一个文档贯穿起来，允许我们产生一些类似于SQL内部连接机制提供的选择书-作者对，而这些书的作者之一为Alex Homer：

程序清单 8-110

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/Catalog">
    <xsl:copy>
      <xsl:for-each select="//Book[Authors/Author='Alex Homer']">
        <xsl:for-each select="Authors/Author">
          <xsl:element name="BookAuthor">
            <xsl:element name="Title">
              <xsl:value-of select="../../Title"/>
            </xsl:element>
            <xsl:element name="Author">
              <xsl:value-of select="."/>
            </xsl:element>
          </xsl:element>
        </xsl:for-each>
      </xsl:for-each>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

这里，我们使用了两个<xsl:for-each>块。第一个块处理问题的过滤方面：哪些书是你感兴趣的？第二个块，沿着节点树向下反复浏览来得到我们感兴趣的東西：<Author>元素。然后再沿着节点树向上浏览——去获得我们感兴趣的另一个信息块，也就是书的标题。下面是转换输出：

程序清单 8-111

```
<?xml version="1.0" encoding="utf-8"?>
<Catalog>
  <BookAuthor>
    <Title>IE5 XML Programmer's Reference</Title>
```

```

    <Author>Alex Homer</Author>
  </BookAuthor>
<BookAuthor>
  <Title>Professional ASP 3.0</Title>
  <Author>Alex Homer</Author>
</BookAuthor>
<BookAuthor>
  <Title>Professional ASP 3.0</Title>
  <Author>Brian Francis</Author>
</BookAuthor>
<BookAuthor>
  <Title>Professional ASP 3.0</Title>
  <Author>David Sussman</Author>
</BookAuthor>
<BookAuthor>
  <Title>Beginning Components for ASP</Title>
  <Author>Alex Homer</Author>
</BookAuthor>
<BookAuthor>
  <Title>Beginning Components for ASP</Title>
  <Author>Richard Anderson</Author>
</BookAuthor>
<BookAuthor>
  <Title>Beginning Components for ASP</Title>
  <Author>Simon Robinson</Author>
</BookAuthor>
</Catalog>

```

(6) 外部连接

由于还是由于XML文档提供了包含功能，外部连接也向关系数据库那样重要——处理器只需查看下一元素即可弄清楚信息块存在还是不存在。不过，通过浏览文档树，来提供一些类似外部连接所提供的功能还是可行的。假如我们提取一系列书籍，指明这些书是不是由 Alex Homer完成了一部分。在SQL中，我们可以使用外部连接，如果一本书 Alex Homer并没有写其中的一部分，则返回NULL。在XSLT中，我们可以利用count()函数和<xsl:choose>分支元素来提供类似功能：

程序清单 8-112

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/Catalog">
    <xsl:copy>
      <xsl:for-each select="Book">
        <xsl:choose>
          <xsl:when test="count(/Authors[Author='Alex Homer']) > 0">
            <xsl:element name="BookAuthor">
              <xsl:element name="Title">
                <xsl:value-of select="Title"/>
              </xsl:element>
              <xsl:element name="CowrittenByAlexHomer">Yes</xsl:element>
            </xsl:element>
          </xsl:when>
          <xsl:otherwise>
            <xsl:element name="BookAuthor">
              <xsl:element name="Title">
                <xsl:value-of select="Title"/>
              </xsl:element>
              <xsl:element name="CowrittenByAlexHomer">No</xsl:element>
            </xsl:element>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:for-each>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>

```

```

        </xsl:element>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:for-each>
</xsl:copy>
</xsl:template>
</xsl:stylesheet>

```

这里，我们来验证一下 Alex Homer 是不是每本书的有贡献作者之一（函数 count（）行使检查功能）；然后我们根据条件元素 <xsl:choose> 代表的路径采用一种方法来创建元素。变换结果输出如下：

程序清单 8-113

```

<?xml version="1.0" encoding="utf-8"?>
<Catalog>
  <BookAuthor>
    <Title>IE5 XML Programmer's Reference</Title>
    <CowrittenByAlexHomer>Yes</CowrittenByAlexHomer>
  </BookAuthor>
  <BookAuthor>
    <Title>Designing Distributed Applications</Title>
    <CowrittenByAlexHomer>No</CowrittenByAlexHomer>
  </BookAuthor>
  <BookAuthor>
    <Title>Professional Java XML</Title>
    <CowrittenByAlexHomer>No</CowrittenByAlexHomer>
  </BookAuthor>
  <BookAuthor>
    <Title>XML Design and Implementation</Title>
    <CowrittenByAlexHomer>No</CowrittenByAlexHomer>
  </BookAuthor>
  <BookAuthor>
    <Title>Beginning ASP 3.0</Title>
    <CowrittenByAlexHomer>No</CowrittenByAlexHomer>
  </BookAuthor>
  <BookAuthor>
    <Title>Professional ASP 3.0</Title>
    <CowrittenByAlexHomer>Yes</CowrittenByAlexHomer>
  </BookAuthor>
  <BookAuthor>
    <Title>Professional Site Server 3.0</Title>
    <CowrittenByAlexHomer>No</CowrittenByAlexHomer>
  </BookAuthor>
  <BookAuthor>
    <Title>Professional ADSI Programming</Title>
    <CowrittenByAlexHomer>No</CowrittenByAlexHomer>
  </BookAuthor>
  <BookAuthor>
    <Title>Beginning Components for ASP</Title>
    <CowrittenByAlexHomer>Yes</CowrittenByAlexHomer>
  </BookAuthor>
</Catalog>

```

(7) 表内容操作

由于 XSLT 是在源文档的变换后版本，而不是在源文档上操作的，所以不会修改原始文档。正如前面我们所讨论的，有一些其他工具更适合于 XML 文档的这类操作。

(8) 从不止一个数据源返回信息

XSLT提供一个函数 `xsl:document()` 来使扩展文档在基础文档被转换的同时被处理。这就保证了可以从多个数据源中抽取信息并集成到一个结果中。例如,假如说我们有一个文档位于和页面 `status.xml` 相同的目录,而这个页面描述了 `catalog.xml` 所显示的信息的状态:

程序清单 8-114

```
<CatalogStatus>
  <GeneratedDate>December 17, 1999</GeneratedDate>
  <LastModifiedDate>December 12, 1999</LastModifiedDate>
  <LastUpdatedBy>Jon Duckett</LastUpdatedBy>
</CatalogStatus>
```

现在,让我们看一下当我们请求 Alex Homer 所著书籍的标题时,如何把 `<GeneratedDate>` 元素从文档添加到结果的结构中去:

程序清单 8-115

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/Catalog">
    <xsl:copy>
      <xsl:copy-of select="document('status.xml')//GeneratedDate"/>
      <xsl:for-each select="//Book[./Authors/Author='Alex Homer']">
        <xsl:copy>
          <xsl:apply-templates select="Title" name="childnodes"/>
        </xsl:copy>
      </xsl:for-each>
    </xsl:copy>
  </xsl:template>
  <xsl:template name="childnodes" match="*">
    <xsl:copy>
      <xsl:apply-templates name="childnodes"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

我们已经引导 XSLT 处理器打开文档 `status.xml`, 并把元素 `<GeneratedDate>` 添加到了变换文档的根元素 `<Catalog>` 中。注意,普通的定位步骤可用来对函数 `document()` 产生的节点集进行过滤。

产生的 XML 文档为:

程序清单 8-116

```
<?xml version="1.0" encoding="utf-8"?>
<Catalog>
  <GeneratedDate>December 17, 1999</GeneratedDate>
  <Book>
    <Title>IE5 XML Programmer's Reference</Title>
  </Book>
  <Book>
    <Title>Professional ASP 3.0</Title>
  </Book>
  <Book>
    <Title>Beginning Components for ASP</Title>
  </Book>
</Catalog>
```

(9) 程序处理

虽然利用 XSLT 进行某种程度的程序处理是可能的，但是它却不能提供和 SQL 服务器或者 Oracle 相近的东西。例如，元素 `<xsl:for-each>` 支持限制形式的游标（允许每次一个地对节点集中的独立节点进行操作），却不允许编程时游标向前或者向后“移动”。让我们看最后一个例子——产生一列 catalog.xml 包含的作者及每个作者参与写作的书的标题。

在 SQL 中可以采用下列指令进行刷新：

程序清单 8-117

```
SELECT a.author, b.title
FROM author INNER JOIN book ON author.bookID = book.bookID
ORDER BY a.author
```

这样可以产生表 8-10 所示结构类型的结果。

表 8-10

作 者	标 题
Alex Homer	IE 5 XML Programmer's Reference
Alex Homer	Professional ASP 3.0
Alex Homer	Beginning Components for ASP
Brian Francis	Professional ASP 3.0
...	

系统接受这种单调的结构，然后提供一个消除重复作者信息的方法。另一部分利用在 SELECT 指令中采用 DISTINCT 关键字首先提取作者，然后提取每位作者所著的书。

我们希望完美的 XSLT 转换输出看起来该是下面这个样子：

程序清单 8-118

```
<?xml version="1.0" encoding="utf-8"?>
<Catalog>
  <Author>
    <Name>Alexander Nakhimovsky</Name>
    <Title>Professional Java XML</Title>
  </Author>
  <Author>
    <Name>Alex Homer</Name>
    <Title>IE5 XML Programmer's Reference</Title>
    <Title>Professional ASP 3.0</Title>
    <Title>Beginning Components for ASP</Title>
  </Author>
  ...
</Catalog>
```

为了达到这个目的，我们准备使用一种新的 XSLT 元素类型——而且是可行的——利用它连回到文档中另一个位置。让我们看一下如何实现，完整的页面如下所示：

程序清单 8-119

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/Catalog">
    <xsl:copy>
      <xsl:for-each select="//Author/text()">
```

```
<xsl:sort select="."/>
<xsl:variable name="thisAuthor" select="."/>
<xsl:if test="count(preceding::Author[text()=$thisAuthor]) = 0">
  <xsl:element name="Author">
    <xsl:element name="Name">
      <xsl:value-of select="$thisAuthor"/>
    </xsl:element>
    <xsl:for-each select="//Book[./Authors/Author=$thisAuthor]">
      <xsl:copy-of select="Title"/>
    </xsl:for-each>
  </xsl:element>
</xsl:if>
</xsl:for-each>
</xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

我们把上面拆开，逐个看每一部分，首先：

程序清单 8-120

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/Catalog">
    <xsl:copy>
      <xsl:for-each select="//Author/text()">
```

这里，我们先利用 `<xsl:for-each>` 产生一个关于文档中出现的作者名字的游标。注意，这里会产生一个副本——每次一本书的作者将会被提到，作者的名字会出现在列表中。我们来看一下如何尽快将这些名字区分开来：

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

我们将按字母顺序对名字排序（这里对第一个名字排序）。

程序清单 8-121

```
<xsl:variable name="thisAuthor" select="."/>
```

这个指令存储了正在变量中出现的作者的值。在环境信息发生改变的时候，这将是回指信息的一个很好的方法，后面我们将会看到这一点。

程序清单 8-122

```
<xsl:if test="count(preceding::Author[text()=$thisAuthor]) = 0">
```

这是我们处理区分问题的一种不太好的方法。对于一位作者的每本书来说，通过 `<xsl:for-each>` 元素的 `select` 属性选择节点会发生重复——例如，Alex Homer 的名字会出现三次。为保证每位作者仅操作一次，我们将作下面的测试。把 XPath 翻译成英语，这个测试是这样的：“Only proceed if there's no `<Author>` element that appears earlier in the (original) document with the same text.” 很自然地，这将保证每位作者真正出现一次，这样我们就可以得到正被寻找的这种区别。（另外，由于 XSLT 变量范围规则的原因，在变量中保存上一个作者名，并和当前的作者相比较，来看看是否发生了变化，“传统的”方法将不起作用）。

程序清单 8-123

```
<xsl:element name="Author">
  <xsl:element name="Name">
    <xsl:value-of select="$thisAuthor"/>
  </xsl:element>
```

我们创建了带有作者名字（从变量中得到）的 <Author>元素和<Name>元素。

程序清单 8-124

```
<xsl:for-each select="//Book[./Authors/Author=$thisAuthor]">
```

这里我们为文档中作者和当前作者相匹配的所有书的示例了另外一个游标。注意在这里我们没有使用“.”来代替\$thisauthor——因为方括号内索引的上下文正好是正被核对的 <Book>元素的上下文，而不是第一个 <xsl:for-each>的上下文。在使用 XSLT 时了解上下文对避免产生不希望的结果是很关键的。

程序清单 8-125

```
<xsl:copy-of select="Title"/>
```

我们将书的 <Title>元素写到输出中，并继续迭代：

程序清单 8-126

```
    </xsl:for-each>
  </xsl:element>
</xsl:if>
</xsl:for-each>
</xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

该页面的输出为：

程序清单 8-127

```
<?xml version="1.0" encoding="utf-8"?>
<Catalog>
  <Author>
    <Name>Alexander Nakhimovsky</Name>
    <Title>Professional Java XML</Title>
  </Author>
  <Author>
    <Name>Alex Homer</Name>
    <Title>IE5 XML Programmer's Reference</Title>
    <Title>Professional ASP 3.0</Title>
    <Title>Beginning Components for ASP</Title>
  </Author>
  <Author>
    <Name>Brian Francis</Name>
    <Title>Beginning ASP 3.0</Title>
    <Title>Professional ASP 3.0</Title>
  </Author>
</Catalog>
```

```
<Name>Chris Ullman</Name>
<Title>Beginning ASP 3.0</Title>
</Author>
...
</Catalog>
```

8.6.5 查询语言展望

虽然XSLT和XPath可以使开发者在相当大程度上对查询和XML文档内容表达进行操纵，然而W3C表示这两种技术不是查询的最终解决方案。一些查询（像上面的 books-per-author 查询）如果使用XSLT和XPath将相当困难，而且查询语言中通常出现的一些附加功能（如元素的添加或更新）是没有用的。W3C已经成立一个XML查询研究小组（XML Query Working Group）来研制从真实和虚拟的文档中提取数据的灵活的网络查询工具。不过，在本书写作的时候，该小组还没有任何成果出来，我们期待不久一种更新的查询技术草案能够出现。

8.7 小结

这一章我们讨论一些目前最新的访问和操作XML文档的最新技术：

- 回顾了W3C Inforset，W3C用它来描述组成XML文档的信息块。
- 回顾了XLink规范，它定义了数据源之间的文档中创建链接的机制。
- 回顾了XPointer，它定义了指向XML文档中特定位置或者位置范围的机制。
- 探讨了XML程序段交换的工作草案规范，以及如何传输带有环境信息的XML程序段，从而有助于解释XML程序段。
- 我们探讨了一些现有的XML查询语言，并通过几个例子验证了两个最新的查询技术：XSLT和XPath

由于时间仓促，这些技术很多还没有被广泛应用，相信不久的将来它们肯定会成为被广泛使用的XML技术集合的一个组成部分。如果你已经了解了这些技术，当它们变得实用的时候，将会使你在未来的竞争中更具优势。