

第7章 命名空间和模式

在这本书里我们所了解到的定义 XML 词汇的工具——就像 DTD 那样组织得非常好的 XML 基本规则——在 W3C XML 1.0 推荐书里作了介绍。它们是 XML 世界的基础部分，为应用开发者定义了核心功能，允许我们（以及其他的人）生成标记词汇以用来描述我们目前所处的困难领域。但是在我们准备掌握这些技术的同时，特别是在比较大的实际应用里，通常是希望我们在计划里多一些功能性的东西。

在进入这一章之前，让我们先看几个我们可以利用的问题。不过，只要我们在头两页看到了什么是我们正在错过的，就不会让这些问题干扰你，我们将在本章剩余的篇幅里去阐述我们提到的所有问题。一些问题你可能已经遇到过了，鉴别其他问题可以防止你花大量毫无结果的时间去钻牛角尖。

首先，在第 3 章里我们已经提到了 DTD 的一些缺陷。这些缺陷主要围绕在它们是用语法而不是用 XML 来描述，以及它们表达得不够充分。于是我们要做的第一件事将是再回顾一下 DTD 的这些问题，并尝试一下解决它们的几种办法。这样做将帮助我们力量集中在一些其他与定义我们自己标记有关的问题上。

其他你可能考虑的问题是关于每个人是否有能力去创建他们自己的标记。你可以大致想象一下每个人用相同的元素名称去定义不同的东西。比如说，如果你考虑使用“monitor”这样的元素，那么它在不同的环境将有几种不同的意思。如果你在计算机外围设备使用一个 DTD，“monitor”可能指的是计算机屏幕，同时在音乐制作间里扬声器通常也叫做“monitor”。如果这里有一个学校 DTD，“monitor”可能指的是一个被赋予几种职责的学生，然而在原子核电站，“monitor”可能放在报警的地方。即便意思相同，在两种不同的定义中，其内容也会发生改变。面对元素的这些潜在的不同用途，我们需要一种方法去区分元素的特定用途，特别是在我们在同一个 XML 文档里混用不同的词汇。为了解决这个问题，这里有个 W3C 组织提出的称为 XML 命名空间的规范，它允许我们在一个命名空间定义元素的前后联系。

而且，我们还需要把由于遵照不同的 DTD 而将来自不同地方的 XML 文档结合起来。这时，我们就可以描述大量的信息，而单一的 DTD 不合适并很难让读者去理解，或者说它可用于电子商务，在那里我们需要把商业伙伴的数据和我们自己的数据联系起来。然而，XML 推荐书没提供不修改或新建 DTD 而在单一文档里混合 DTD 的方法（通过外部参考文档）。

进一步考虑，随着越来越多的行业标准 DTD 不断产生，已经存在与你涉足的领域相关的 DTD 这种可能性就越大。如果现有的 DTD 对于你眼前的应用不太完美，还不如重新创建一个新版本，这可能对于在各个分离的 DTD 中添加你自己的定制信息非常有帮助，它还可以允许你以标准格式交换确定的子集信息。不过我们不能轻易用 DTD 做到这一点。

这些问题变得越来越重要，特别是考虑到 XML 提供商在电子商务领域的承诺，在这个领域不同的公司与用户以固定的格式交换数据非常有意义。从代码读取 DTD 并与文档一致是可能的，

但这不是一件简单的事情。于是我们需要几种方法去研究不同的和相近的比较词汇，以便我们能建立一种联系。到此，W3C组织正在致力于被写入XML，称作XML 模式具有可选性DTD 模式的工作。

这种可选计划语言将标明这些问题以及我们将要在本章剩下的篇幅里看到的一系列 DTD的其它缺点。但我们首先通过从多源建立单一 XML文档的方式，大概看一下一部分问题。

使用相同的元素名称，依据不同的 DTD建立起属于不同的模式来源，依此来源建立单一XML文档的问题涉及到词汇的数据（但它们已经被建立起来）以及这些规则的来源。XML组织和它的支持者们已经开始这些问题方面的工作，其结果对于正在出现的基于XML的电子商务很有帮助。如果你对利用XML关联有分离的团体建立起来的不同类型的系统很感兴趣，你需要理解这些XML世界新的扩充。

这一章将要揭示一些XML组织致力于解决这些问题的成果。它将给你提供两个方面的知识：命名空间和XML模式。命名空间帮助XML词汇表设计者去将复杂的问题分解成细小的问题，以及根据需要混合多意词来描述单一XML文档里的问题。模式允许词汇表设计者去建立更多而准确的词汇定义，而在过去要靠DTD和XML语法才能做到的。

这两个工具回答一些当利用XML去处理模棱两可的问题时出现的问题。特别是命名空间和XML 模式允许XML设计者和程序员能够做到：

- 更好地组织围绕一个复杂问题的词汇表。
- 当转入和转出数据时，提供一种方法去保留强大的录入功能。
- 比起DTD许可，更加准确、灵活地描述词汇表。
- XML里的“读”词汇表规则，允许访问词汇表定义，而不用增加解析器的难度。

1999年1月14日，XML 命名空间达到了W3C推荐的程度。模式正在通过标准的方法开展工作，但很快就需要一种推荐说明书。这种对模式的需求在应用开发团体里非常强大，以至于模式支持的技术调研小组开始安装解析器。这是由于为了应付推荐书出来之后模式快速的转换，模式草案准备得足够值得推敲。

7.1 混合词汇表

回忆一下我们在第3章见过的图书目录 DTD。在建立一个站点之后，用XML写成PubCatalog.dtd词汇表，发布作品目录，你可以决定在线出售作品。这意味着需要能够为目录里的书籍编排顺序。因此，需要一个DTD来研究书籍的顺序。

如果继续按照DTD章节提到的去做，你可能会添加到PubCatalog.dtd文件，这是因为这两个范围标明了同一个问题——共享书籍数据的不同部分。但它们也可以看作不同的问题领域，这是因为一个标明了目录的整体，而其他的标明了目录里的销售款项。当被这两部分传递的信息里有一些重迭，而你又想通过一个DTD去研究两个领域，就会以被一大堆复杂的DTD所迷惑而告终。

庞大并且包含很多主题信息的DTD很难让程序员阅读和理解。更重要的是，如果你已经在产品里使用了目录DTD，DTD的改变可能会影响应用程序。但是这里有更好的解决方案，即融合利用命名空间的单一文档中与各个目录、表单DTD保持一致的数据，因此我们将研究这种可

能性。但先让我们从近处看一下你面临的问题。

7.1.1 分解问题

首先，你为什么愿意用目录 DTD 混合表单的细节呢？至少有两个问题值得讨论，一个是所有书籍的目录，另一个是每个题目的出售情况。当你考虑正在写一个大的程序，会把整个问题分解成细小的问题。一些结构程序语言按照这种意图提供了模块、类、组件、包、函数等。设计词汇表可以看作与编程类似的问题。你总是要将一个大的问题分解成多个词汇表。但是我们必须要克服的问题其实并不是写每个 DTD 去描述许多词汇表，我们在第 3 章已经看到怎样才能做到这一点。如果我们将定义分解成目录和索引 DTD (order DTD) 的话，真正的问题存在于整合 DTD 进一个文档的实体。

7.1.2 重用

在 PubCatalog.dtd 里我们使用了 BOOK 元素。某种程度上，在标记描述元素内容的数据的过程中，这些做得相当完美。但是由于我们想要实现在线定书，当涉及到顾客想要订购的书籍时，很可能还想使用相同的元素名称。的确，两者很有可能用不同的 DTD 来描述。毕竟表单里的 BOOK 元素可能是表单元素的子级，因此，在 PubCatalog.dtd 里这是一个子 CATALOG。

正如我们已经建议的，这是一个在创建 XML 词汇表时反复出现的问题。在描述真实世界概念时，我们将要不断地发现存在着的普遍结构。毕竟，复杂的创作是从简单的建筑元素——比如颜色、形状、价格和尺寸，简单的事物不能长时间不定义，于是这里将有很多元素名称的实例，他们已经有了定义和内容模型。

无论是你还是别人，用这些元素创建了一个 DTD，借鉴已经被证明的 DTD，你的任务将变得更加容易（的确，对于处理词汇表里已经定义的结构代码是可以利用的）。这就是重用的含义。

即便你在为一个公司做计划，可能被现有的 DTD 所困扰。实际上，借鉴它们可以使你的工作变得更简单，而忽视它们会另每个人的工作很难办，这是由于因为程序员的参与，DTD 代表了一种特定集合内的智力投资。正如其他人知道的，这些 DTD 描述了业务问题。在真实的生活里，建立于与我们的这些例子里的书籍相关的 DTD 意味着你的任务是去扩展它，这在某种程度上是当前已经知道和定义了的概念的延续。

的确，如果你正在编写一个应用程序并和其他合作伙伴的程序进行连接，除了重用现有的概念之外，没有别的选择。使用中的 DTD 已经形成了一种你为了理解而去讲的通用语言。无论什么时候概念存在，你都应该努力按照概念去理解。

已有定义的使用者正在努力地扩展它们并进行初始化。劝说别人去适应你的关于这个问题的观点可能非常困难。即便你能够完成这项壮举，也应该认识到建立新的定义和代码和从现有的 DTD 规划你的新内容相比会付出额外的代价。重用节省时间、人力和资金。

7.1.3 多义性名称冲突

不论你是重用其他设计者的 DTD 的定义，还是将离散的 DTD 连接起来去生成一个描述符合

问题的文档，如果正在使用的文档里采用相同名称的元素，你都会冒多义性名称冲突这一问题的风险。比如书是一个非常好的概念。你可以确定这里有几个 DTD 声明了“BOOK”这个元素，至少有出版商、印刷商、零售商和图书馆等。在文档里单一使用 BOOK 名称需要一个约定，即要与合适的 BOOK 元素声明相匹配。在我们的例子里，BOOK 是一个对目录和表单都通用的名称。

一个用 PubCatalog.dtd 做标记的文档可能包括了下面这些对 <BOOK> 元素的使用：

程序清单 7-1

```
<Book>
  <Title>Professional XML</Title>
  <Abstract>Compendium book containing everything you need to learn to use
    XML in your programming solutions today.</Abstract>
  <RecSubjCategories>
    <Category>XML</Category>
    <Category>Programming</Category>
    <Category>Internet</Category>
  </RecSubjCategories>
</Book>
```

因此一个关于书的表单可能需要像下面那样使用 <BOOK> 元素：

程序清单 7-2

```
<Order>
...
  payment and shipping information
...
  <Item>
    <Book>
      <Title>Professional XML</Title>
      <ISBN>1-861003-11-0</ISBN>
    </Book>
    <Quantity>3</Quantity>
    <Price US$="49.99" />
    <Discount US$="10.00" />
    <SubTotal US$="119.97" />
  </Item>
</Order>
```

如果我在读一个 XML 文档中包含来自两种词汇表的数据，那么怎么知道它指的是哪一种定义呢？

当你从多个 DTD 把名称实例拿过来使用的话，问题变得严重了。假如我们有一个土木工程师参与市政规划的应用程序。当谈到照明，我们为了交通灯和街灯而需要利用已有的 DTD。各自独立工作，每个词汇表设计者都会选择 <Light> 这个词来作为元素的名称。如果他们知道了最终应使用他们自己的 DTD，他们就会选择 <TrafficSignal> 和 <StreetLamp>，但在 DTD 写入时，未来的使用是未知的。现在我们面临着一种风险，即文档里有一个模棱两可的“Light”元素。

给 <Light> 声明两种用途非常困难。第一个声明需有交通信号灯并枚举它的颜色属性。枚举非常重要，因为对于交通信号灯来说只有三个有效的颜色。一个应用可以基于这些属性的值用来做一些错误检查：

```
<!ELEMENT Light EMPTY>
<!ATTLIST Light color (red | yellow | green) #REQUIRED>
```

第二种声明在它的颜色属性上没有特别严格的限制。的确，路灯的选择常常基于价格，而不是颜色。但颜色还是作了如下描述：

```
<!ELEMENT Light EMPTY>
<!ATTLIST Light color CDATA #REQUIRED>
```

下面看看这段混有两种DTD的XML应用文档：

程序清单 7-3

```
<Inventory>
  <Light color="red"/>
  .
  .
  <Light color="white"/>
  .
  .
</Inventory>
```

从这一点，我们不能区分 Light元素指的是交通灯还是街灯（没有检查暗含在 DTD中的颜色的限制）。那么一个正在接收的应用程序怎样知道颜色这个属性是否可接收呢？我们不知道哪一个元素查找哪一个 DTD，以及第二个 Light元素的颜色属性值对于用于交通灯是无效的。这个问题对格式正规的文档就存在多义性。而且如果 Light和Color名字需要确认，我们可能会给应用制造一大堆混乱，这就是提到的名字冲突问题。

7.2 命名空间

XML 命名空间是解决多义性和名字冲突问题的方案。根据 W3C组织的推荐书“XML中的命名空间”（1999年1月14日）：

.....一种名称的集合，通过一种 URI引用来标识，作为元素类型和属性名称，它应用于XML文档。

命名空间是一组具有结构的名称的集合；这听起来像一个 DTD，的确，一个 DTD可以是一种命名空间。在这种情况下，URI可以是在你的服务器上的地址，比如：

<http://www.wrox.com/xmltdtds/PubCatalog.dtd>

尽管URI不需要是一个 URL（如果你不能明确二者的差别，我们简单描述一下它们）。在这种情况下，命名空间是指在 PubCatalog.dtd里用到的名称。因此如果我们通过某种方式把使用 BOOK元素与命名空间联系起来，将会知道在已连接的文档里任何关于 BOOK的引用将要在我们的 PubCatalog.dtd里涉及到它的用法。

DTD规定了一个文档的整体结构（并且是那么的准确），我们正好以一个命名空间为资源，规划所需要的定义。说到这里，一个命名空间不需要是一个像 DTD那样的有固定结构的定义，而这个有限的定义领域使命命名空间广泛应用于 XML。如果命名空间是 DTD或者模式，我们使用的定义必须在所描述的结构和语法上保持连续性。但是我们可以自由地使用需要的名称，并且使用命名空间来区分元素的使用。

于是，为了在文档里有效地使用命名空间，而文档中连接着来自不同地方的元素，我们需要两部分：

- URI 引用，定义了元素的使用方法。
- 一个别名，我们可以用此来标识元素来自哪个命名空间，这将采用元素前缀的形式（例如在<catalog:Book>那里，catalog是模糊的BOOK元素的别名）。

7.3 定义和声明命名空间

看到了命名空间在XML里所带来的优点，我们需要仔细看一下如何真正地使用它们。首先看一下在文档里怎样声明一个命名空间，然后看一下在文档里怎样使用命名空间，最后再给出几个例子。

通常，简单描述的特性通常作为属性来建模，并且这就是命名空间是怎样在 XML 声明的。但这里有几个变形与转化，于是我们将要一步一步地去学习当在一个 XML 文档里声明一个命名空间时能描述什么。

7.3.1 声明一个命名空间

如果每个人在他们打算去认识一个命名空间声明，我们需要一个保留的词汇给他们。命名空间推荐标准给了我们 xmlns。属性值就是 URI，其唯一地定义了正在用的命名空间。URI 经常是一个指向 DTD 的 URL，但它并不必须是。用这种方式管理一个 URI，以唯一区分命名空间已经足够了。这里有几个简单的命名空间声明：

```
xmlns="http://www.wrox.com/bookdefs/book.dtd"  
xmlns="urn:wrox-publishing-orderdefs"
```

关于 Web 资源的术语可能令人混淆。统一资源标识符 (URI) 是一些资源的唯一名称。统一资源定位器 (URL) 根据协议和网络位置定位资源。第一个例子是 URL，因为它允许一个浏览器利用 HTTP 从一个特定的位置得到资源。第二个例子给资源命名但没提供位置。字面上的 urn 来自于 URI。

最初使用命名空间动机之一是能够从不同的来源混合名称，从那以来，这对于能够提供别名非常有用，而你能在一个涉及到声明的文档里通篇使用这个别名。可以靠加个冒号和你的别名到 xmlns 属性而实现该功能。因此上面的例子就变成了：

```
xmlns:catalog="http://www.wrox.com/bookdefs/PubCatalog.dtd"  
xmlns:order="urn:wrox-publishing-sales-orderdefs"
```

在这里前缀 catalog 将要涉及到来自 PubCatalog.dtd 的元素，而其他的将要涉及在 order.dtd 里声明的元素。在这些声明出现之后，我们能只使用 BOOK 去提及最初的命名空间声明，以及用 ORDER 去涉及其他的（不用 URI）。使用这些声明和它们的别名让我们提供了更多的信息。

图 7-1 是组成命名空间声明的部分。

7.3.2 限定名

如果不能和一个我们想要使用的特定的名称绑定在一起，声明一个命名空间是没有什么用处的。这些已经通过利用限定名做到了。这就可能是你希望的——一个从命名空间勾画出来并

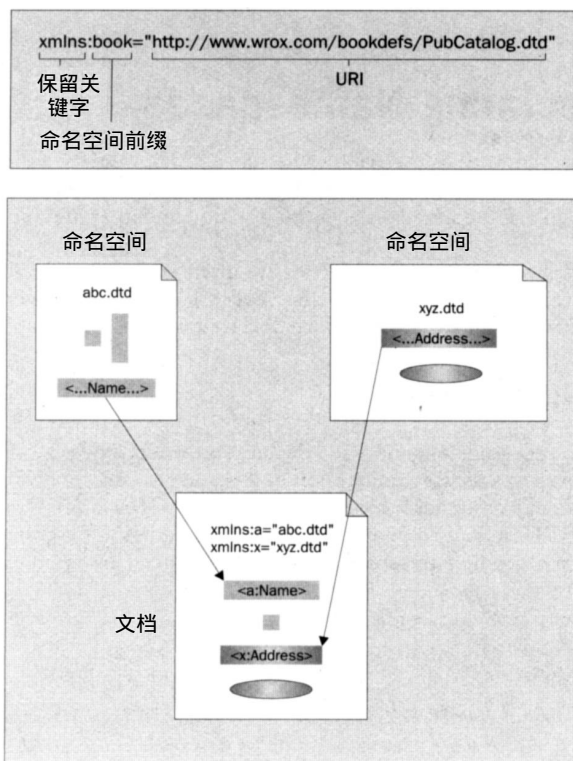


图 7-1

经其限定了的名称。通过别名创建一个确认过的名称，确切地说称作命名空间前缀，并把它放在名称的开始。回到在目录和表单 DTD里包括BOOK元素这个问题，假如我们像下面那样用 catalog前缀声明了一个目录命名空间：

```
xmlns:catalog="http://www.wrox.com/bookdefs/PubCatalog.dtd"
```

我们能够使用前缀 catalog，使元素来自哪个命名空间更加明确。于是 <catalog:Book />将要告诉我们BOOK名称来自 catalog 命名空间声明。同样 Order 命名空间也有 Book名称，但限定过的名称避免了多义性和冲突的可能性。名称 Title作为来自一个特殊的命名空间被清楚地验证过。命名空间前缀经常被提及为前缀，而名称本身是基本名。

限定名可被应用于元素和属性名称。这里有一个混合一些命名空间的例子：

```
<catalog:Book order:ISBN="1-861003-11-0">
```

这个元素<Book>从我们在上面看到的第一个命名空间那里产生，而属性 ISBN从order 命名空间产生。

7.3.3 范围

命名空间声明就像变量在程序语言里那样有它的作用范围。这非常重要，这是因为命名空间并不是总是定义在XML文档开始，它们能够被包含在文档的较后部分。一个命名空间声明因此而应用于有声明出现的元素，尽管与此同时子元素并没有清清楚楚地描述出来。只要被用在

命名空间声明的范围之内，就能够访问到命名空间。

但是我們也需要去混合命名空间，在那里元素另外地继承命名空间的作用域，于是这里有两种可以声明作用域的办法：缺省和限定。

1. 缺省

如你想象的，在一个文档里在每一个名称前加一个前缀非常令人厌烦。实际上，通过在的工具集里引入名称作用域的概念，能够分配很多前缀。如果定义了缺省的命名空间，在声明作用域里所有没经验证的名称被假定属于缺省的。于是如果你在根元素声明了一个缺省的命名空间，它将被看作整个文档将缺省的命名空间，并只能在文档里声明过的更多的命名空间所覆盖。

通常省略前缀可以将一个命名空间声明为某范围内缺省的。

这就是如何在 XML 文档里使用这些去内嵌入一些 HTML，这些文档根据一种为书的内容所设计的称作 BookContent.dtd 的 DTD 来标记：

程序清单 7-4

```
<Chapter xmlns="http://www.wrox.com/bookdefs/BookContent.dtd">
  <Title number="7">Namespaces and Schemas</Title>
  <Author>I. M. Named</Author>
  <Content>
    <Paragraph>
      Let's have a table:
      <table xmlns="http://www.w3.org/TR/REC/REC-html40">
        <tr>
          <td>A tisket</td><td>A tasket</td>
        </tr>
        <tr>
          <td>One fish</td><td>Two fish</td>
        </tr>
      </table>
    </Paragraph>
    <Paragraph>This is a very short paragraph</Paragraph>
  </Content>
</Chapter>
```

<Title>、<Author>、<Content>和<Paragraph>这些元素以及来自缺省命名空间的属性 number 在 <Chapter> 元素定义。但是在 Chapter 元素里，你能看到 table 元素和它的子元素——tr 和 td。这些属于用 table 元素声明的 HTML 命名空间。应注意到这个例子里当 table 元素关闭时，HTML 命名空间声明作用域随之结束。下面 Paragraph 的出现并不是来自 HTML 命名空间。

当一个前缀被定义并被一个名称利用时，明确地声明了命名空间。由于一个没有限定而被命名空间接受的名称，一个缺省的命名空间必须被声明为带有一个作用域，这个作用域包括没限定的名称（不包括前缀）。

2. 限定

如果你能够清楚地区分命名空间当然非常好。但有些时候可能想要在一篇文档里从外面的命名空间来浏览名称。你需要一个更精细的划分尺度。除了在整个空间声明命名空间，还可以利用限定过的名称。在文档开头声明你将需要的命名空间，然后在使用地点限定它们。

程序清单 7-5

```
<Measurements xmlns="urn:mydecs-science-measurements"
  xmlns:units="urn:mydecs-science-unitsofmeasure"
  xmlns:prop="urn:mydecs-science-thingsmeasured">
  <OutsideAir units:units="Fahrenheit">86</OutsideAir>
  <FuelTank>
    <prop:Volume units:units="liters">120</prop:Volume>
    <prop:Temperature units:units="Celsius">20</prop:Temperature>
  </FuelTank>
</Measurements>
```

在根元素Measurements，我们声明了命名空间。缺省值涉及到了元素<OutsideAir>、<FuelTank>和<Measurements>。但是我需要用测量单位验证一部分我们曾用命名空间units和属性units做过的内容。当这些属性在文档里突然出现时，能够验证那些名称非常有用。最终，我需要区分一些测量方法的类型，即prop:Volume和prop:Temperature。尽管我已经在<FuelTank>元素里声明这些prop命名空间，我还是可以依靠在开始声明命名空间及使用限定名，自由地反复使用这个命名空间（或许在一个更长的文档里）

让我们更仔细地看一下命名空间声明并将它与在接下来的<Chapter>元素里出现的命名空间作一个对比。那个声明被绑定在DTD上，这样一来就可能使用与DTD冲突的名称。在这个例子里，我们有唯一的名称，但没有DTD URL。命名空间的存在主要是用来将名称组织成特有的集合以及回避名称冲突。W3C命名空间推荐标准没有描述任何有关验证的使用方法。确实，XML 1.0 Recommendation没有说任何有关命名空间的东西。XML模式的成就（后面我们将看到）做得更多，但当前命名空间用于验证的任何东西将要严格地保留一件人造物品——个人分析器工具，直到XML模式成为正式的W3C推荐标准。

7.4 在格式正规的书籍里使用命名空间实例

让我们试着标记这本书的内容，并看一下是否能以一种有用的方式利用我们的工具命名空间。假设像第3章那样DTD内容已经建立。我们将要从现有的目录DTD借用名称，而不是再创建存在于HTML里的标记，同样将借用命名空间。现在将把验证问题放在一边，并假设这个文档只需要被格式正规化。更多注意一下作用域问题，这里将要开始标记这本书，显示这一章的开始部分：

程序清单 7-6

```
<Book xmlns="urn:wrox-pubdecs-content"
  xmlns:cat="urn:wrox-pubdecs-catalog"
  cat:ISBN="1-861003-11-0"
  cat:level="Professional"
  cat:pubdate="1999-11-01"
  cat:thread="WebDev"
  cat:pagecount="450">
  <cat:Title>Professional XML</cat:Title>
  <cat:Abstract>The W3C positions on namespaces and schemas are
    presented, together with a review of commercial support.</cat:Abstract>
  <Author>
    <FirstName>Iye</FirstName>
    <MI>M</MI>
```

```

<LastName>Named</LastName>
<Biographical>
  Iye M. Named is a researcher with the Adaptive Content
  division of Wrox Press. He has many good ideas, which he
  is too shy to mention.
</Biographical>
<Portrait piclink="inamed.jpg"/>
</Author>
<Chapter>
  <Title>Namespaces and Schemas</Title>
  <Section SectionAuthor="inamed">
    <Paragraph>The tools for defining XML vocabularies that you've seen so far
    in this book - the basic rules of well-formed XML as well as DTDs - are the ones
    provided in the W3C XML 1.0 Recommendation...
    </Paragraph>
    <Paragraph>Both problems ...</Paragraph>
    <Paragraph>This chapter ...</Paragraph>
    <Paragraph>The two ...
    <UL xmlns="http://www.w3.org/TR/REC/REC-html40">
      <LI>Better organize...</LI>
      <LI>Provide...</LI>
      <LI>Describe vocabularies...</LI>
      <LI>"Read" vocabulary rules...</LI>
    </UL>
    </Paragraph>
    <Paragraph>XML Namespaces...</Paragraph>
  </Section>
  <Section SectionAuthor="inamed">
    <Title>Mixing Vocabularies</Title>
    <Paragraph>Recall the Book Catalog DTD...</Paragraph>
    ...
  </Section>
  ...
</Chapter>
...
</Book>

```

我在根元素定义了两个命名空间。内容命名空间是缺省值，因我需要严重依靠那个命名空间并且想要限定尽可能少的名称。我发现从目录命名空间借用几个名称非常有用，于是用前缀 cat 声明了那个命名空间。这允许我从目录命名空间引进一些属性，并在根元素包括它们，这些属性是从内容命名空间得到的。然后，我需要包含一个列表。这些在 HTML 里已经建好，于是我声明了另一个命名空间：

```
<UL xmlns="http://www.w3.org/TR/REC/REC-html40">
```

我没提供一个前缀，于是 HTML 变成了缺省的命名空间，但这只是对于 UL 元素和它的子级，列表项目（LI）。只要将那个区域合并，用与 UL 元素相近的标记，我们以缺省值回复内容命名空间。

开始这个例子之前我要告诉你，这是一个格式正规的例子。的确，如果我在指向 DTD 的命名空间声明里提供了 URL 并且要求你通过一个限定的解析器去运行它，你将会因为几件事而震惊。XML 1.0 推荐标准在每个文档里并没有提供一个以上的 DTD。在这里，尽管 DTD 被用作唯一的名称，它们并没有因验证而读出来，原始的 DTD 没有来自 HTML 命名空间名称的概念。只要你一试着引进外面的名称，解析器将会指出错误，这是由于你引进的元素或属性在第一个 DTD 下不被允许。我希望已经告诉你了命名空间是有用的。验证也是有用的。使二者一致就是

XML 模式的好处之一。

7.5 模式

第一件要弄清楚的事是 DTD 实际上是一种模式。但是当 XML 领域里的人们提及模式时，他们通常是指用 XML 语法写成的 DTD 替代物，是一个我们在这一章用过的术语。这里对于 DTD 有很多可供选择的提议，并且 W3C 组织当前正致力于从这些工作中建立一个标准的可以选择的绘图灵感。在某种意义上，我们可以把模式想象为一种强制机制，其中定义了允许的元素、属性等等，我们正在约束使用者选择标记和他们的内容模型。

通常，我们能够把模式说成元数据或关于数据的数据，并且当我们快要明白一些模式的效果不止参与定义词汇表时，它们早就开始去解释特定类型数据的关系。

如果你想要替换 DTD，需要具备至少与他们相同的能力。你需要描述 XML 文档的自然状态和结构。就像 DTD，一个模式是 XML 词汇表组建和规则的描述。模式靠许可在表达一些词汇表概念有更高的精度细化了 DTD。并且模式做了一些大幅度的改变。它们使用了完全不同于 DTD 的语法。它们允许我们去借用其他的模式，而且解决了你在最后的命名空间例子里遇到的问题。它们提供了元素和属性的数据类型。整个模式实际上对于描述词汇表问题是个更好的答案。

XML 与 DTD 结合得很好。与此同时，在改进它们的问题上有很大的兴趣。这种兴趣包括多种形式，其中有各种已被采纳的建议（其中几种可从注解中的 W3C 站点得到）。当这些成为工作中重要的一部分时，这也推迟了一个涉及多个最普遍的模式急需特征的推荐标准的采纳。特别是，许多开发者想要很强的分类能力去验证多个命名空间，并在一些时候使用 XML 语法。幸运的是，这种状况已经被解决。在写这本书的时候（2000 年 1 月），关于模式的 W3C 工作组在使许多对模式语言有贡献的建议合成一个有用的描述上做得很好。改进的模式给 XML 文档自动交换数据带来很大的益处。

7.5.1 与 DTD 有关的问题

你可能在学习 DTD 语法规则上投入了很大的精力，并且缺少模式描述不能阻止你探索许多 XML 的好处和研究许多有趣的例子。因此你可能奇怪 DTD 出了什么问题，以至于你必须学习一种新的方法。首先，学习 DTD 非常值得，这是因为（在写的时候）它们对于描述你自己的标记只提供了一种标准。并且，有许多用 DTD 定义过的标志语言，读懂它们对于采纳标志非常有帮助。

但是，正如我们在第 3 章里建议的，DTD 有几个缺点，并且当我们试着用 XML 做更多事情时，这些缺点会变得更加明显。

- 它们很难写及理解。
- 它们的元数据在程序处理时非常困难。
- 它们不可扩展。
- 它们不对命名空间提供支持。
- 它们不支持数据类型。
- 它们不支持继承。

让我们按顺序看一下每个问题。

1. DTD很难写及理解

比起XML，DTD使用一种语法，即Extended Backus Naur Form(EBNF)，并且许多人发现它很难阅读和使用。而被提议的XML模式实际上使用了XML去描述它们定义的语言，在学会读、写它们之前去掉了学习EBNF的困难。

2. 程序处理元数据非常困难

使用EBNF也使在DTD里自动处理元数据非常困难。这里当然是指DTD的解析器。你可能已经有了一个；它是你偏爱的经过限定的解析器。经过限定的解析器在它们能够验证一个适合的文档之前必须装载和读入DTD。但是不可能从一个程序使用DOM去探究DTD。DOM不能提供访问用EBNF写成的词汇表元数据。你的验证过的解析器读了DTD并保护它的信息的完整。如果DTD用XML写成，而我们能够像研究根据规则写成的文档那样研究它们，这不是很好吗？这个特性将允许我们使用DOM去调查新遇到的词汇表结构或者甚至为了验证依据运行时的条件修改一个词汇表的规则。

3. DTD不可扩展并且不对命名空间提供支持

正如我们在命名空间检测看到的，一个DTD是it。一个词汇表里所有的规则必须存在于DTD。你将需要的每一件事放入DTD，并且你与它们同在。你不能不建立扩展实体而从其他源借用。

既然写了catalog.dtd文件，你应该想要在代码上加一个新的部分，为了新的<releaseDate>元素，整个DTD将要必须重写。即便你通过拷贝和粘贴其中的大部分来实现这些，你必须小心地确定现有的文档仍然有效。

进一步讲，创建并维护你自己的标记声明的子集并不像简单地引用现有的定义那样灵活。你不能允许文档作者在后面包含一些有趣的而在DTD里没有发现的东西。当然，我们并不是经常给文档作者这么大的自由度，但当设计一个新词汇表时给出一个选项，即可以利用现有模式的一部分，那将变得非常好。

还有，因为一个词汇表里所有的规则必须存在于DTD，如我们所看到的，你不能混合命名空间。虽然你能使用一个命名空间去将元素类型引入一个文档，你不能使用命名空间去查阅一个DTD里的元素声明。如果一个命名空间使用了这个命名空间的所有元素，它必须在DTD里声明。

4. DTD不支持数据类型

XML最强劲的地方是各个文档完全用一个通用的数据类型-文本写成。当我们写程序时，经常需要谈论类型而不是文本。比起文本，DTD没提供几种数据类型，当我们在特定类型的应用程序里使用XML时，这是一个非常严重的缺陷。

因为DTD提供无标准的机制去包含无文本类型的我们标记的数据，这意味着我们必须暗中共享有关数据类型的信息，在解析文档时为我们自己执行转换。比如我们想要完成一些在数字元素内容上的循环，在应用能够被希望处理这些数据之前，必须要将文本转换成合适的数据类型。

5. DTD不支持数据继承

采用DTD没有办法表达继承，于是你设想一下，我们有一个称为“books”的类，这里没有

办法说books是一个publications的子类，并且没有办法使books从publications继承下来。

假如我们将书籍分成三类：专家水平、程序员参考手册和初学者指导手册，我们不能说它们是books的子类，因此也就不能继承books类的属性。

总的来说，DTD对于定义文档结构非常好，并且当我们考虑到XML是源于SGML时，很容易去理解在XML 1.0描述里选择DTD，而SGML也使用了DTD。但是当我们看到XML用于更多的程序环境，而不只是文档标记，这些限制变得越来越重要。

之后，模式搜索地址是基本原则。在查看当前的XML模式草案状态之前，应该回顾一下其他一些XML组织内有关元数据方面的工作，于是我们就能评价它们的发展方向。

7.5.2 一个对创建模式的帮助

学术界在开展元数据的论题之前不能坐下来等待XML的发明。元数据——关于数据的数据——是用来描述信息的。这可能简单得像建立一个模式数据库，也可能模糊得像在这样的一个模式里，没有定义地讨论含义。

学术团体——以及一些与XML相关的元数据提议——趋向于更加雄伟的计划。一个例子是资源描述框架（RDF），W3C组织为描述资源准备了力量，于是它们被自动解决。其他计划瞄准了更多的代替DTD或以相关的数据库模式方式描述数据。

因为渴望着一种基于XML的模式语言来替代和扩展DTD，大量的提议被提了出来。它们包括：

- XML-Data。
- 文档内容描述(DCD)。
- 面向对象的XML（SOX）模式。
- 文档定义标记语言（DDML最初被人称作X模式）。

上面这些没有一个直接得到W3C支持的正式的工作，但是每一个都在W3C关于XML模式的工作中考虑过了。

我们的希望落到了RDF以及一些简单的DTD的XML版本上。我们需要一种方法去在一个简单但富有表现力的表里表达结构和内容。当我们的确想要去欣赏尽可能多的表现力时，我们不能忘了一个事实，即在得到一个软件方面实行并给组织接纳的提议，简单化也是一个强大的因素。毕竟XML本身是一个SGML简单化的版本。对核心强劲但简单的属性，通过减少属性集合，XML作者建立了一个简单的标准，它很快得到了广泛的认可。

于是在有关XML模式部分，我们将要看一些基于XML的元数据提议。首先我们将要看一下前景看好的RDF的努力，然后其他两种模式提议，即XML-Data和DCD。这将给予我们一个环境致力于来自W3C的模式方面的工作。在指向这些时，我们将要指出基于XML模式的一些主要的题目。W3C模式工作组观察了其中的每一个，然后他们对于该领域非常有兴趣，这一领域是XML模式成功建立的基础，以及将灵感及有用的概念引入到最新的XML元数据定义版本。

在看过这些领域之后，我们将要看看W3C组织是如何进展正在形成的XML模式工作的，并且在本章的最后看一下如何利用MSXML里的早期的命名空间和模式支持。

这一章我们回顾了三个提议，决不是只影响当前W3C XML模式成就，也不仅在XML领

域的元数据的成就。你可以在<http://www.w3.org/Metadata/>和<http://www.w3.org/TR/>里看到这些成果。一些其他W3C以外的成果可以在Robin Cover的XML站点上阅读到，它的索引在<http://www.oasis-open.org/cover/siteindex.html>上找到。我在下面有限篇幅里涉及到的三个提议都在XML 模式成果主流中，并且可以建议一些XML 模式的贡献。其他一些注释包括面向对象的模式（SOX）和文档定义标志语言（DDML，前面提到的X模式）。

注意我们并没有试着去讲解每一个提议而是去引进一些在其中一部分元数据提议里表明的关键概念。由于W3C XML 模式成果还没有被完全认可，除了演示的目的外，这里还没有应用程序来支持它。但是我们将要着眼于详细的语法，它已被Microsoft的MSXML解析器（它作为单机组件搭载在IE5上）在技术上得以试运行。MSXML使用了叫做XML Data简化版的XML Data提议子集。这些例子将在本章结尾出现。现在让我们继续看看将要引入的第一个提议。

1. 资源描述框架

资源描述框架（RDF）在元数据成就方面有很好的前景。它允许一个设计者去描述对象，添加属性来定义和描述它们，以及生成复杂的关于对象的语句，比如关于资源关系的语句。其中被提议的使用包括站点图、内容级别、数据流通道定义、搜索引擎数据集合（网络行进）、数字库集合和分发权限。这些描述分为两部分：

- 模型和语法。
- RDF 模式。

基础的RDF模型是一个完整的推荐标准（1999年2月22日）。它像其他语法一样，遍及了可用XML表达的描述性数据模型。RDF 模式是一个已被建议的推荐标准（1999年3月3日），它覆盖了用来表达RDF数据模型的XML词汇表。RDF利用了发展用来定义网络内容和操作级别的系统——Internet Content Selection平台（PICS）的经验以及早期在元数据方面的学术工作。

靠RDF发展了的模式不止能够定义名称和结构，还能做决断，比如正在争论之中的事物的关系。RDF能被复杂化，但它提供了如此强大的富有表现力的功能，它的复杂性需要很好地进行描述。

RDF适应于三个概念：资源(resources)、属性(properties)和语句(statements)。

(1) 资源

资源可以是任何东西——概念域中任何切实的从URI可引用到的实体，从整个网络站点到HTML或XML页中的单一元素。它甚至包含一些在站点上不能利用的东西，比如一本打印出来的书。

资源被写入。一个类体系通常用来定义类别，而具体的资源实例就来自那里。支持类继承，于是一个设计者能够指定定义水平的范围，从广义普遍的到狭义具体的。这里有两个简单的类定义，第一个定义了一个普遍意义的 Rocket类，第二个通过继承将 Rocket类细化成 ChemicalRocket类。rdfs和rdf 命名空间是RDF推荐标准的一部分：

程序清单 7-7

```
<rdfs:Class rdfs:ID="Rocket">
  <rdfs:subClassOf
    rdf:resource="http://www.w3.org/TR/WD-rdf-schema#Resource"/>
</rdfs:Class>
```

```
<rdfs:Class rdf:ID="ChemicalRocket">
  rdfs:ClassOf rdf:resource="#Rocket" />
</rdfs:Class>
```

(2) 属性

资源用属性来定义和描述它们。约束使它们具有了外形。这些约束限制了可赋给属性数值的类型，以及可选类型数值的范围。给我们的 chemical rocket一些fuel：

程序清单 7-8

```
<rdfs:Class rdf:ID="Fuels">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/TR/
    WD-rdf-schema#Resource"/>
</rdfs:Class>

<rdf:Property ID="fuel">
  <rdfs:range rdf:resource="#Fuels" />
  <rdfs:domain rdf:resource="#ChemicalRocket" />
</rdf:Property>
```

我们的fuel属性作为Fuels类被写入，属性能够具有范围内的数值。为了做这些，与上面某些别的地方近似，我们将必须在模式里做一个类声明来实现，或者讨论的火箭燃料提供字面上的数值。这个燃料属性应用于ChemicalRocket类，它的属性域（参见图7-2）。

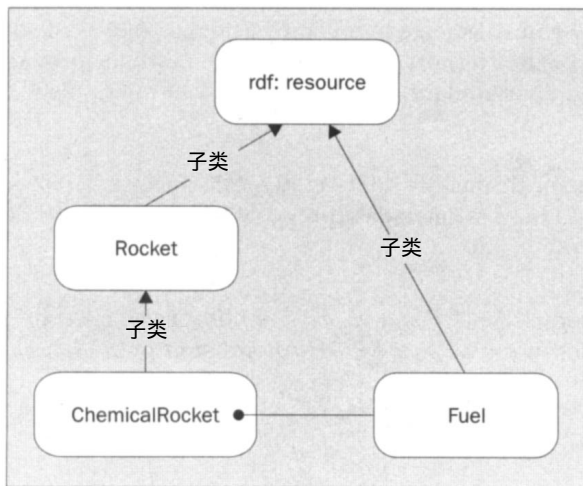


图 7-2

(3) 语句

一旦通过资源和属性定义了名称和结构，关于概念域的语句就可以生成了。这可以通过主语资源、属性谓语和数值宾语来实现。这些值可能是描述具体语句的文字，或者是有关整个类的强大有力的语句的资源。让我们先做一个在文档里与 RDF 模式保持一致的简单、详细的有关火箭的语句。第一，你开始声明一个 ChemicalRocket 类的实例，并给它一个名称：

程序清单 7-9

```
<?xml version="1.0" ?>
<rdf:RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  <ChemicalRocket ID="Moonship" xmlns="urn:my-rdf-rocket-schema"/>
    <rdf:Description about="Moonship">
      <fuel>hydrogen</fuel>
    </rdf:Description>
  </rdf:RDF>
```

一旦声明了火箭的实例——Moonship，利用资源的ID属性，进展到RDF Description元素。对于燃料特性已经提供了一个特别的值——hydrogen（记住，Moonship是ChemicalRocket的一个实例，并且这个类用fuel特性）。这可能使许多事情看起来非常简单，但我们能用同样的办法去做有关类的语句。若你在模式里做了更多的语句，你将要生成很多有关问题域的明确的知识。

RDF是强大的，允许富有表现力、功能强大的语句。它解决了DTD很强的分类限制。确实是，强大的分类是RDF模式的核心。遗憾的是，设计一个RDF模式是一个很费力的过程，它包括很多类和属性的声明。能够生成有意义的语句比起我们需要定义XML词汇表的目的，或许是更强大的特性。

这并不是说，它们在其他环境没有用。RDF语句让我们用一种机器可以读懂的格式规范地描述事实。通常说，我们写的XML词汇表潜在地依赖于对真实世界概念普遍意义上的理解。通过RDF语句，至少在理论上能够提供足够的信息，有关一个应用可以发现附加的词汇表的事实。这将可以使我们更好地使用新词汇表，并决定什么时候它对于手边的问题可用。简而言之，RDF提供了一个提供描述词汇表环境的工具，其中一个工具可用将来将词汇表放在文章合适的位置。

尽管一个设计者努力地致力于定义名称、结构和关系的工作中，这也可能是一项非常繁重的重担。下面的元数据提议采取了几个步骤，以到达富有表现力并有普遍性的程度。

更多的有关RDF信息可以从站点<http://www.w3.org/TR/REC-rdf-syntax>（基础模型）和<http://www.w3.org/TR/PR-rdf-模式>（RDF模式）。

2. XML Data

比起RDF，XML Data瞄准了更为现代的领域。这个提议已被ArborText、DataChannel、Inso和Microsoft提交到W3C组织，很显然将重点集中到了自动化文档和处理，但它比DTD更有前景。

XML-Data在语法和概念模式之间有着显著的区别。二者使用相同的语言，但它们却为我们提供不同的方式去考虑正在标记的语言。

语法模型是一个描述怎样用标记写文档的规则集合，比如DTD是一个语法模式的例子。在一个按照我们的目录DTD标记的XML文档里，<Book>元素能够合法地包含<Title>、<Abstract>、<RecSubjCategories>和<Price>元素。这个世界的一个语法XML Data模式在词汇表结构上描述了近似的约束规则。

概念模型却描述了概念或对象之间的关系，同时它们对于创建关系数据库模型非常理想。我们能够使用XML Data模式用一种脱离任何XML文档的方式假定一种关系——书包含标题和价格。在这个意义上，XML Data是用来扩展XML从关系数据库里很容易包含信息的能力。在关系数据库里通过关键字获得的主要的关系能正式地在XML Data模式里得到。我们能够用命名空间得到比如说特别的连接里的特别关系，这要借助对连接的表声明命名空间，根据所属表验证

查询字段。

我们在第10章讨论带数据库的模式用途。

XML Data提供了一些有趣的工具，它们使XML Data比DTD更加强大。这些工具解决了几十个在DTD里发现的问题，于是让我们看一下其中几个问题以及它们是怎样用的。

(1) 用XML写

XML Data 为了模式结构使用XML词汇表，它允许用户不用必须去学一种新的语法而去读写模式。这也意味着我们能够使用DOM和现有的解析器去细读一个模式或者动态创建一个新的。

继续概念上的比喻，我们能够动态创建一个模式给特别的基于查询本身的SQL查询。数据容器将拥有数据和正式结构，并且将永远不会知道这是动态创建的。

(2) 数据分类

XML Data添加了强大的元素和属性分类，而且回答了我们最初的对于DTD的异议。这些可能是有数据类型命名空间定义的基础数据类型，或者复杂的，由设计者提供的模式所提供的用户自定义类型。这里不再需要应用程序去理解一些元素或属性的数据类型并在使用数据前把文本串转换成合适的格式。这个信息能隐式地在模式里被描述，并且解析器能按照自己的需要进行应用的转换。

(3) 允许数值上的约束

XML Data要求在元素和属性数值范围上的约束被定义，比如最小值和最大值。这在很多你验证XML文档的环境里特别有所帮助。想象一下订购系统，在那里你只接受了超过最小值100美元的定单，最大值为1000美元，你可以强加这些限制到用XML Data写成的定单模式里。换一种想法，如果他们的帐户没有钱，你能够使用约束去防止人们消费，或防止他们输入无效的数值。

(4) 类型继承

一个有趣的重用机制是XML Data支持类型继承。这些使我们在试着用XML解决的问题里描述实体时发展和扩展了元素。我们能够写一些具有普遍意义的超类型声明。实体可能按照这种方式用于DTD，但是类型继承标准化了这个过程。没有一个正式的语义集合，实体可能被误用到不是使用户清晰，而是使他们迷惑的地方。一个正式的继承机制给了我们一个工具，在解决一些控件如何使用时去提高重用。

(5) 开放和关闭的内容模型

XML Data另一个强大的特性是开放和关闭的内容模型的概念。经典的DTD是一个关闭的模型。文档转换成它一定坚持这个原则，并且可能不包含任何不遵循这个规则的内容，因为词汇表里的所有规则一定存在于DTD。

如果一个模式是开放的，文档转换成它可能包含没有在DTD里声明的其他信息。与模式一致的部分必须服从存在于模式里的规则，但我们能够插入其他条目而不受当前模式的限制。这些条目可能在另一个模式里定义过，或者可能完全没有限制。我们可能插入一些特殊的值。更为重要的是，从我们当前讨论的立场出发，开放模型文档是我们混合命名空间的方式。我们能够大量完全遵照一个模式的信息嵌入文档中去以与另外一个模式保持一致。更为正式地，各个元素可能明确地被声明为含有开放或关闭的内容模型。这些通过content属性来实现。这个属

性缺省的值是open。这里有个例子：

程序清单 7-10

```
<elementType id="Person" content="closed">
  <element type="#name"/>
  <element type="#address"/>
</elementType>

<!-- This document fragment is invalid due to the added Telephone element -->
<Person>
  <Name>John Doe</Name>
  <Address>123 Anywhere Street Blasted Rock, NV</Address>
  <Telephone>555-1212</Telephone>
</Person>
```

若是上面例子中的content属性被赋成open，这个片段就是有效的。

(6) 扩展的ID和IDREF结构

XML Data通过关系扩展ID和IDREF结构。在关系里，一个元素担当着对另一个元素内容的关键字或索引。这可直接应用于关系型数据库的主键和外键。它对于有两种语言的文档也特别有用。别名和关联非常有用。

别名用于定义一个同等意义的元素，于是在我们的例子里，可以在英文文档里有 <Book>，并且想要用相应的法语元素将这个标记翻译成 <Livre>。

其他时候，我们想要假设两种标记说明特定的事物，这需通过关联来实现。

换一个角度来考虑，我们可能有一个购物文档，其中有一个 <Purchaser>元素，它引用其他地方的<Customer>元素。<Purchaser>的关联是<Customer>是指<Purchaser>是<Customer>的别名。这些对于从事实体关系图表工作的数据库设计者非常熟悉。

正如你看到的，XML Data直接回答了我们对于DTD提出的问题。我们将不会在XML Data的实用信息上走得太远，这是因为Microsoft IE 5.0带来的XML解析器里的模式支持已经出现了简化形式的提议。我们将要在本章后面深入研究这些支持。

更多的关于XML Data信息可能在<http://www.w3.org/TR/1998/NOTE-XML-data/>找到。

3. 文档内容描述

文档内容描述（DCD）提议是与XML Data提议密切相关的。它已被IBM、Microsoft和Textuality公司提交。一个RDF词汇表很明确地是为声明XML词汇表而设计的。它的支持者利用富有表现力的元数据标准——RDF——去创建被提议的更多适合领域的标准。这个地方XML的创建与简化的SGML子集有相同的特性。

尽管XML Data一些更为先进的特性已经不见了，DCD在语法上近似于XML Data。DCD没有提及关系和关联。它主要集中在定义XML词汇表。然而像元素继承一样，它保留了强大的对XML Data数据类型的支持。像XML Data一样，DCD允许词汇表设计者去声明一个模式模型，或者开放的、或者关闭的。不像XML Data，在用来定义元素时，DCD使用相同的机制声明开放或关闭的模式。像XML Data一样，DCD允许对元素内容值上的约束说明。例如，一个名为<SmallInvestment>的元素可能利用对它允许数值的约束来声明一个固定数量的类型，比如说比

零大并且小于等于一万。

```
<ElementDef Type="SmallInvestment" Datatype="fixed14.4" MinExclusive="0.00"  
Max="10000.00">
```

DCD在吸收RDF的主体同时，是一个对DTD问题的直接攻击。它将强大的功能转向简单化。由于它与XML Data和IE上的支持的模式如此相同，我们就不会在DCD上研究得很深。但是为了理解W3C模式的功能，应记住DCD是元数据的简单结束。DTD将重点放在精度上，并且为了给XML模式提供简单可使用的标准而放弃了深度。

W3C声明中有关文档内容的说明可在站点<http://www.w3c.org/TR/NOTE-dcd/>。

4. 寻找正确的平衡点

这些提议只代表了一系列元数据能力的精华部分。它们决不只是影响在XML模式上。参考这本书的上下文考虑它们。问问你自己“什么是便于XML在网络应用中真正需要的？”问答前面提出的问题只是一个必备条件的子集。事实上，对于企业网应用，我们可能甚至没有能力去用XML解析器读模式。我将要讨论另一个问题：简单化。应用集成，特别是覆盖公众Internet，更需要简单、可靠的方案。复杂性只能招致失败，延误传输。就像简单的XML在知名度和接纳程度上快速地超过复杂的SGML，我相信一个简单但有效的元数据提议将会才是真正能满足需要的。

RDF在自己的领域非常完美。它强大的表现能力可被用于特殊的舞台。然而期望一个复杂、标准的RDF任何时候很快变成网络应用开发工具包的整体部分是不合乎道理的。XML Data和DCD与这个标志比较紧密；它们去掉了复杂性以有利于它们的促进者想要作为本质的东西。这是一条要被画的复杂的线。XML Data关系重要吗？更多地依赖于基于今后几年内XML应用的本质。

一个致力于XML模式的工作组正努力工作并希望在2000年内达到推荐标准的要求。XML模式有很多方面归功于RDF、XML Data、DCD以及几个其他的提议。当前的成就看起来正倾向于简单化，尽管它可能在后期很好地被扩展。由于这希望在这本书发行之后很快成为改进的W3C建议书，我们将要深入研究一下草案。

7.6 W3C在XML模式方面的工作

W3C XML模式工作组在1999年12月17日约定了两部分关于XML模式的工作草案。像任何工作草案一样，特别的属性和语法受到后来的版本的影响。这些模式回答了我们在本章前面提出的有关DTD的问题。它们用XML语法写成，允许使用多个命名空间，它提供强大的内容分类。而且它们是XML 1.0 DTD功能的超集。它丰富的表现力超过DCD，但远比RDF精练。简而言之，这是一个有前途的元数据成就。

1999年12月17日的工作草案被分成两部分：结构和数据类型。

结构部分，XML模式部分：结构，处理元素和属性的描述和声明。那里提供的材料允许XML设计者去指定复杂的元素结构及设定这些元素内容数值的约束。这些描述部分可以从<http://www.w3.org/TR/xmlschema-1/>上找到。

第二部分，XML模式部分2：数据类型，提出了标准的数据类型内容集合，就像从它们生

成新类型的规则一样。这些描述部分可以从 <http://www.w3.org/TR/xmlschema-2/> 上找到。

7.6.1 DTD与XML 模式的比较

你目前正充满希望地急于去学习 XML 模式的正式语法。只是为了证实一下，在这里让我提供一个非常简单的 DTD 和它的翻成 XML 模式的形式。对于我谈到过的有关模式和它们的特性，我还没让你看过一个例子。当前的做法——DTD——和我们希望成为未来做法的模式对比，将会让你看到事物是怎样急剧地变化的。它将让你对目前为止所讨论过内容有所领悟。不要过多担心模式的语法。我们将要在下面几部分详细研究。试着勾画一幅情景，并且用它来作为今后的参考。

考虑下面给一个人命名的 DTD：

程序清单 7-11

```
<!ELEMENT Name (Honorific?, First, MI?, Last, Suffix?)>
<!ELEMENT Honorific (#PCDATA)>
<!ELEMENT First (#PCDATA)>
<!ELEMENT MI (#PCDATA)>
<!ELEMENT Last (#PCDATA)>
<!ELEMENT Suffix (#PCDATA)>
```

我们最少必须拥有姓和名两部分，但可能随意地拥有一个中间大写字母，敬语（Mr., Ms., Dr. 等）和一个后缀（Jr., III 等）。在模式里这些看起来就像这样：

程序清单 7-12

```
<Schema ...>
  <element name="Name">
    <type>
      <element name="Honorific"
        type="string" minOccurs="0" maxOccurs="1"/>
      <element name="First" type="string"/>
      <element name="MI"
        type="string" minOccurs="0" maxOccurs="1"/>
      <element name="last" type="string"/>
      <element name="suffix"
        type="string" minOccurs="0" maxOccurs="1"/>
    </type>
  </element>
</Schema>
```

模式表单有些长，但你将会注意到我们描述了更多的信息。开始，我们有一个 <Schema> 元素作为模式的根结点。然后有一个元素叫做“Name”，它的名字在 <element> 标记“name”属性里被赋值，于是声明一个 <name> 元素：

```
<element name="Name">
```

这为了什么？我曾经在最简单的表单里使用它，但是你应该知道它能被赋予一个名字，并被给予一个声明。在这样的表单里，它适用于在别的地方重用，以及描述 <Name> 元素的内容模型。注意包含在 <Name> 里的元素是怎样被声明的。因为它们是简单的类型（比如字符串或

PCDATA), 我们能在<Name>声明实体里去声明它们, 而用再做其他处理。你将要看到 XML 模式提供了一个比我们现在 DTD 里有的更长的基础类型列表。

注意任意的元素是怎样描述的。通过模式, 我们能够描述元素出现次数的最小值和最大值。这能导致比我们能在 DTD 里描述的更复杂的内容模型。

但是终上所述, 很明显的事实是——模式是 XML。你在前几章学到的 DOM 操作能被用来应对程序里的这个模式, 并能将它拆开。这些在 DTD 表单里不能实现。

7.6.2 结构

我们能用 DTD 定义的每一件事在 XML 模式的结构部分得到了解释。由于 XML 模式是由 XML 语法写成, 结构是指我们能用来定义标记的 XML 命令。当然, 这意味着 XML 模式实际上只是 XML 的另一个应用 (一个为了定义 XML 文档类的词汇表), 并且正是如此, 拥有了一个模式可以来描述它自己 (事实上, 附录里为 XML 模式结构部分提供了模式和 DTD 去描述模式词汇表)。

于是规范的结构部分是定义模式的元素和属性出现的地方。更重要的是, 元素的内容模型在这里得到了描述。内容模型明确地描述了允许的元素内部结构。结构是 XML 模式的核心。于是让我们详细考虑一下它们。

1. 编写模式

一个模式由导言、不定数量 (或没有) 的定义和声明组成。下面几部分讨论这些定义, 于是让我们从导言开始。

(1) 导言

在根元素模式可找到导言。这一定至少包含属性的三部分信息:

- targetNS, 它是正在使用的模式的命名空间和 URI
- version, 用来指定模式的版本
- xmlns, 为 XML 模式规范提供命名空间
- 可选, finalDefault 和/或者 exactDefault, 为两种后面将要常涉及的扩展名提供缺省值

它可能包括转出和转入, 包括结构, 这些我们将在后面讨论它们。这里是一个演示导言的模式例子:

程序清单 7-13

```
<?xml version="1.0"?>
<schema targetNS="http://myserver/myschema.xsd"
        version="1.0"
        xmlns="http://www.w3.org/1999/XMLSchema">
    ...
</schema>
```

这里, 我们假定的模式驻留在 myserver, 并且叫做 myschema.xsd, .xsd 是 XML 模式的文件扩展名。它保留在第一个版本里。缺省的命名空间声明是 XML 模式: 结构的引用, 并且它是一个关闭的模型模式, 这意味着所有与这个模式一致的文档将要完全由模式来定义, 完全不含有任何外部内容。

(2) 简单类型定义

为XML 模式定义的结构紧紧依赖于类型定义。这允许一个模式设计者去声明能在模式里使用的扩展类型。它们将要被用于说明元素和属性的内容和类型。一个简单类型定义用于限制不包括元素的信息。它由名称和说明组成，其中的说明既是另一个类型定义的引用，又是一系列面的集合。面将在本章后部的数据类型部分被详尽描绘。在数据类型元素里可找到独立简单的类型定义：

程序清单 7-14

```
<datatype name="smallInt" source="integer"/>
  <minExclusive value="0"/>
  <maxExclusive value="10"/>
</datatype>
```

我们将要详细讨论数据类型的结构。也能在其他的声明里有一个简单的类型定义，比如属性。这由type属性完成，例如type="smallInt"，它告诉我们声明项的类型。

(3) 复合类型定义

这些是XML 模式里重要的结构。没有它们，将不能组成重要的元素内容模型。<type>元素包含在一个复合的类型定义里。嵌套在里面，我们声明了元素和属性，或者对模型组的引用。例如：

程序清单 7-15

```
<type name="someContent">
  <element .../>
  <attribute .../>
</type>
```

复合类型定义可能变得非常棘手。在学习了怎样去声明属性和元素之前，这些非常难以理解。在我们向前走时你把注意力放在看到的<type>元素上，然后你将会明白我的意思。

(4) 属性与属性组

属性声明由<attribute>元素组成，而<attribute>元素至少包含一个name属性。<attribute>元素也有可选的cardinality属性：minOccurs和maxOccurs，它可用来指出属性是否显示，以及如果是，经常怎样显示。一个type属性指明了属性的数据类型，比如说字符串或整型。一个属性声明可能是default和fixed属性。这些更像DTD里的关键字IMPLIED和FIXED。fixed属性的值是必须经常有的值。default属性的值是假如属性没有明确地在XML文档元素里出现时的值。这里有属性声明的几个例子：

```
<attribute name="simpleAttr"/>

<attribute name="sequenceNo" type="integer" default="0"/>
```

我们经常遇到一组相关的属性，它们在一个模式里应用于多元素声明。XML 模式结构为这些提供属性组的想法。这是一个命名的属性声明的集合：

程序清单 7-16

```
<attributeGroup name="troopParameters">
  <attribute name="serialNum" type="string"/>
  <attribute name="rank" type="string"/>
```

```
</attributeGroup>

<type name="officerParms">
  <attributeGroup ref="troopParameters"/>
</type>
```

这里我们声明了troopParameters属性组，然后在officerParms类型定义里使用了它。

(5) 内容模型

我们不想在没有内容模型时进展得太远，并且XML 模式提供了比DTD更准确的描述内容模型的机制。它们使用复合类型定义和一个新的结构，<group>元素，去建立元素声明的内部内容。

我们现在需要类型元素的另一个属性——content属性（参见表 7-1）。content属性告诉我们哪种元素能被包含（尽管它没指出有关许可的属性的任何内容）：例如：

表 7-1

内容属性值	含 义
unconstrained	任何类型的内容
empty	空元素
mixed	元素和字符数据

程序清单 7-17

```
<type name="WideOpen" content="unconstrained"/>

<type name="NothingHere" content="empty"/>

<type content="mixed">
  <element ... />
</type>
```

当我们接触到只含有元素的内容时，事情变得更加有趣。现在需要一些内容操作者——模式草图里术语称为排序，去演示怎样排列内容。这些排序是<group>元素order属性的值。这个新元素给了我们一种方法去提供声明里有序的元素实体。排序显示在表 7-2中。

表 7-2

排序关键字	含 义	DTD等价物
seq	元素必须按准确的顺序排序	, (逗号)
choice	模型元素之一准确地出现	(管道符号)

(6) 元素声明

这里我们能立即看到XML怎样用来在XML应用里生成模式语法，在那里我们必须使用<!ELEMENT语法去在一个DTD里声明一个<Book>元素，现在将元素声明放在XML元素里，于是我们使用：

```
<element name="Book" />
```

这里<element/>元素用来声明一个元素（这个元素描述它的内容为了保持与自我描述数据的一致性）。name属性简单拥有一个我们正在生成的元素的值。

简单元素由数据类型和一系列属性声明的引用或一个属性组的引用组成。这与 DTD的这种

声明类似：除了内容被赋了类型，元素只包含 PCDATA。比如：

```
<element name="ZIP" type="string"/>
<element name="windspeed" type="float"/>
```

这些将符合：

```
<!ELEMENT ZIP #PCDATA>
<!ELEMENT windspeed #PCDATA>
```

当然，这里将没有 DTD 那里的字符串和浮点数值类型的概念。当我们想要用结构定义一个元素，用一个内容模型替代这个数据类型引用。让我们将这个放在一旁，然后看怎样通过在其他声明添加引用来建立一个元素声明。为这个简单的 XML 片段说明一下模式：

程序清单 7-18

```
<Name>
  <First>John</First>
  <MI>A.</MI>
  <Last>Doe</Last>
</Name>
```

这里是需要的元素声明：

程序清单 7-19

```
<element name="First" type="string"/>
<element name="MI" type="string"/>
<element name="Last" type="string"/>
<element name="Name">
  <type>
    <group order="seq">
      <element type="First" type="string" minOccurs="1"/>
      <element type="MI" type="string" minOccurs="0"/>
      <element type="Last" type="string" minOccurs="1"/>
    </group>
  </type>
</element>
```

这些已经足够简单了。First, MI 和 Last 是字符串。注意，我已经将 MI 字符串转化成了合适的长中间大写字母，比如像 O'M 或 A.G。现在我们将要把它们包裹在一起成复合元素 <Name>。

例子通常是学习的最好方法，这里有一些例子以及它们的 DTD 等价物：

程序清单 7-20

```
<element name="ListOfNames">
  <type>
    <group order="seq">
      <element type="CustomerName"/>
      <element type="SalesName"/>
      <element type="ProductName"/>
    </group>
  </type>
</element>
```

```

    </type>
  </element>

  <!ELEMENT (CustomerName, SalesName, ProductName)>

  <element name="PickOne">
    <type order="choice">
      <group order="choice">
        <element type="ColumnOne"/>
        <element type="ColumnTwo"/>
      </group>
    </type>
  </element>

  <!ELEMENT PickOne (ColumnOne | ColumnTwo)>

```

现在，我们希望能够描述元素内容的多发性问题。为了做到这个，我们使用元素引用上的 minOccurs 和 maxOccurs 属性。当接触了模型组一段时间之后，将看到我们能同样在那里应用这些属性，以建立更多复杂的内容模型。

(7) 模型组

一些其他的模式结构允许建立定义块和声明块。正如我们已经看到的，可以在特殊的类型里拥有一个下面我们能够给它命名的模型组。这个结构使我们能够建立复杂内容模型，与此同时，我们能够引用命名模型组去建立元素内容模型一些部分，以通过将一个名字放入模型组再利用类型和元素声明，而且允许我们在别的地方引用它。这里是一些例子：

程序清单 7-21

```

<type minOccurs="1" maxOccurs="2">
  <group order="seq">
    <element type="A"/>
    <element type="B"/>
  </group>
  <group order="choice" minOccurs="3" maxOccurs="7">
    <element type="C"/>
    <element type="D"/>
  </group>
</type>

```

在这个模型里，每一个文档将以 AB 序列开始。这至少出现一次，或许两次，我们能选择 C 或 D，并做三至七次这种选择。最后，我们将所有元素以任意的顺序恢复过来。下面将是一个合法的与这个内容模型一致的文档片段。

```

<A/><B/><A/><B/>    <!-- sequence -->
<C/><C/><D/><C/>    <!-- choice -->

```

也可以将组套入组成复合内容模型，例如：

程序清单 7-22

```

<group order="seq">
  <group order="choice">
    <element type="A"/>
    <element type="B"/>
  </group>

```

```
--  
</group>  
<group order="choice">  
  <group order="choice">  
    <element type="A"/>  
    <element type="B"/>  
  </group>  
  <group order="seq">  
    <element type="B"/>  
    <element type="C"/>  
    <element type="D"/>  
  </group>  
</group>  
</group>
```

相应地对于一些元素<foo>的DTD内容模型是：

```
<!ELEMENT foo ((A | B), ((A | B) | (B, C, D)))>
```

现在考虑如果我们能通过名称查阅内容模型组，怎样能使用它们：

程序清单 7-23

```
<group name="partsGroup" order="seq">  
  <element type="BigParts"/>  
  <element type="LittleParts"/>  
</group>  
  
<element name="PartsAndTheirMeasures">  
  <type>  
    <group ref="partsGroup"/>  
    <attribute name="count" type="integer"/>  
    <attribute name="size" type="integer"/>  
  </type>  
</element>
```

在下面的例子里，定义了一个内容模型，然后将它合并到一个元素声明里。连接这些结构使模式设计者灵活、有效地重用和指定词汇表规范。

程序清单 7-24

```
<attributeGroup name="partMeasures">  
  <attribute name="count" type="integer"/>  
  <attribute name="size" type="integer"/>  
</attributeGroup>  
  
<element name="PartsAndTheirMeasures">  
  <type>  
    <group ref="partsGroup"/>  
    <attributeGroup ref="partMeasures"/>  
  </type>  
</element>
```

这是第一个例子的变体。我没有在<element>里建立属性声明，而是创建了一个包含声明的属性组，然后利用元素组和属性组的规范创建元素声明。这是使用属性组的另一个方法。

程序清单 7-25

```
<element name="PairedFasteners">
```

```
<type>
  <group order="seq">
    <element type="Nut"/>
    <element type="Bolt"/>
  </group>
</type>
<attributeGroup ref="partMeasures"/>
</element>
```

这时，我想以不同的元素内容重用属性组。我已经能够在元素声明里通过明确地描述内容模型做到这些。注意，我的内容模型包含 Nut和Bolt类型的元素。我将必须在模式中别的地方声明这些类型。

(8) 通配符

XML模式提供 any元素允许我们引进一个通配符到一个模式的任何特殊点。模式提供了下面四种中任意一种写模式方法开始工作：

- 任何格式正规的XML元素结构
- 任何格式正规的元素结构，只要它在任何一个通配符不出现的命名空间
- 任何格式正规的元素结构，它是从一个明确的命名空间提供出来的
- 任何格式正规的元素结构，它是从一个当前的命名空间提供出来的

通配符也可以用在与属性的关联，在那里我们能使用 anyAttribute元素。这是四种情况的一个例子：

程序清单 7-26

```
<any/>

<any namespace="##other"/>

<any namespace=http://www.myserver.com/OtherSchema/>

<any namespace="##targetNamespace"/>
```

注意，other和targetNamespace关键字的使用方法。现在有一个在与元素声明里的属性建立关联里使用通配符的例子：

程序清单 7-27

```
<element name="someElement">
  <type>
    <anyAttribute namespace=http://www.w3.org/1999?XMLSchema/>
    <element name="someNum" type="integer"/>
  </type>
</element>
```

这里我们声明了一个元素，它有一个单一子元素， <someNum>，并且可以有任意的在 W3C 模式为XML 模式声明的属性。

2. 派生类型定义

当在类型上使用源属性时，我们非常有效地从现有的类型派生新的。XML 模式提供一些正

式的规则用来类型派生，我们现在就要检查这些规则。特别是我们能扩展一个类型或限制它。derivedBy属性的值说明了使用了哪种方法。

派生

当一个新的类型添加附加的内容到它原来的类型时，它扩展了另一个类型。这种情况下，所有在源类型里声明的内容将会出现在派生的类型里。例如，我们通过在现有内容添加一个敬语元素来扩展了一个PersonName类型声明：

程序清单 7-28

```
<type name="PersonName">
  <element name="FirstName" type="string"/>
  <element name="MI" type="string"/>
  <element name="LastName" type="string"/>
</type>

<type name="FormalPersonName" source="PersonName" derivedBy="extension">
  <element name="honorific" type="string"/>
</type>
```

但如果我们想因为某种原因限制一个类型而派生一个新类型时，我们能给 derivedBy属性一个限制值，并且添加<restrictions>元素：

程序清单 7-29

```
<type name="ShortName" source="PersonName" derivedBy="restriction">
  <restrictions>
    <element name="MI" maxOccurs="0"/>
  </restrictions>
</type>
```

在这里我们限制了类型，于是<MI>元素不再出现。当派生类型时，要确认关于元素和属性的限制比起那些关于相同源类型的声明限制更加有限制性。

类型可以控制从它们自己派生，以及通过使用三个属性：abstract,exact和final控制它们在实例文档里出现。如果abstract的值是true，没有声明的类型实例可以出现在实例文档里。正如希望的其隐含的缺省值是false。如果exact的值是true，在它那里没有派生的类型出现在实例文档里。只有被声明过的类型可被使用。如果final的值被赋成true，今后不再允许类型派生。

3. 撰写

我们能将模式和命名空间联合在一起允许使用者从多个模式建立文档实例。模式也允许设计者使用其他的模式建立他们自己的模式文档。这在术语上称为撰写 (Composition)。

(1) 引入

你能引入另一个模式的一部分用在提供了的其他模式的命名空间，<import>元素涉及了这些。这个元素拥有命名空间属性，它的值是你想要使用的模式的URI。你也可以提供一个模式Location属性去指向想要的模式文件。一旦你引入了一个命名空间，就能够在你的模式里使用一些从其他模式来得结构：

程序清单 7-30

```
<schema name="SomeOtherSchema.xsd"
  xmlns:other=" http://www.OtherOrg.org/schemas/Useful.xsd" >
  <import namespace="http://www.OtherOrg.org/SomeUsefulSchema"
    schemaLocation="http://www.OtherOrg.org/schemas/Useful.xsd"/>
  ...
  <element ref="other:stuff" name="someName"/>
</schema>
```

当一个结构被引入到一个模式里，它保留着外部的资源。我们正在有效地组建一个新的模式，通过连入部分另一个模式而不是将它们整个包含进新的模式。当一个验证解析器按照模式验证一个文档时，它必须重新得到其他模式去验证与外部资源不一致的文档里的组成部分。

(2) 包含

`<include>`元素说明了包含。它在一个模式里出现在 `<import>`元素之后，`<export>`元素之前。`<include>`元素和需要的属性模式 Location 一起是个空的元素，它的值是一个指向包含模式的 URI。当这个元素出现在模式里时，这个模式被理解成包含它声明过的类型，同时所有包含的模式里声明的类型提供了几个遇到的标准：URI 必须融入另一个模式，而这样设计的模式必须有一个 target 命名空间属性并与包含模式的 target 命名空间属性值一致。

4. 注释模式

没有计算定义和代码是不需要提供附加的注释或处理信息的机制来完成的。模式为这个提供了 `<annotation>` 元素。这个元素可能包含由字符数据而不是人类的假想构成的 `<info>` 元素，或者为模式做同样事情的 `<appinfo>` 元素。无论哪个元素可能有一个 infoSource 属性，它提供一个有更多信息的 URI 引用。

程序清单 7-31

```
<element name="HardToRemember">
  <annotation>
    <info>
      I want to remember the following about this element declaration...
    </info>
  </annotation>
  ...
</element>
```

7.6.3 数据类型

真实世界依赖于数字、字符串和集合概念，于是用现代程序语言写成的程序支持为了定义新类型而内嵌类型和过程的详细体系。而且对于 XML 模式附加的数据类型对程序员为他们应用程序里的数据而使用 XML 将是一笔很大的财产。这种对数据类型的支持包含检查文档里数值的有效性，以及在处理 XML 文档时帮助从文本到原始类型的合适的转换。于是如果我们打算用 XML 文档作为集成程序和系统的基础，我们需要捕捉标记的信息的数据类型。

这就是 XML 模式规范的第二部分——XML 模式：Datatype 瞄准要做的。它不仅提供了捕捉基础数据类型的方法，同时给了我们一个记录加在属于我们问题里的数据上的方法。它将让我们记录数值范围，设置和列出顺序。它也将让我们为数据允许的字符串表示法指定掩码。

模式数据类型被说成拥有一套独特的值，称做它们的数值空间。这是类型能够具有数值的抽象集合。例如，整数集是整数类型的数值空间。限制这个空间里的数值的属性和操作描述了这个空间的特性。当给用户去表现一个数据类型时，需要一个词汇表示——这个类型的字面字符串表示。一个实数可能被表示为一个数字字符，一个小数点和小数点后一个描述的数字。日期被表示成 YYYY-MM-DD。这是 ISO 8610 格式，它为 XML 表示数据时间而采用。

XML 模式：Datatype 全部描述了数值空间，列出了类型属性的约束。它提供了一系列的原始数据类型，然后详述了从那些原始类型生成新类型的机制。这个草案包含了大量生成的高度有效的类型，但模式设计者对于生成他们自己希望的为特定应用使用的类型非常欢迎。

一些属性，术语称为 facet，被提供用来说明数据类型。facet 细化数值空间以给我们新类型的允许数值。facet 是基本的或约束的。基本 facet 定义数据类型的一些基本属性。约束 facet 在数值空间放置一些约束而不是定义它的属性。比如说，字符串有长度。长度并不告诉你有关字符串的属性，但它们定义了什么样的字符串的值被允许。XML 模式里提供的每一个类型列出了它详细的 facet。一个非常重要的 facet 是词汇表示。既然我们根据 XML 在讲一个基于文本的系统，则必须说明无文本类型的文本表示法。facet 的特别含义依赖于数据类型。更为重要的列在下面的表格里。

1. 原始类型

原始数据类型是那些没被定义成与其他类型有关的类型。它们是自明的。我们从所描述类型的直觉观念出发。XML 模式提议包含典型的 XML 1.0 类型是自然的，但它也添加一些它自己的类型。

表 7-3 中是由 XML 模式引入的原始类型。

表 7-3

模式原始类型	定 义
string	ISO 10646 或无编码字符的有限序列，比如 “thisisastring”
boolean	集合 {true, false}
float	实数的标准数学概念，对应一个双精度 32 位浮点类型
double	实数的标准数学概念，对应一个双精度 64 位浮点类型，有一系列小数尾数，后面可能接着字母 E 和一个整数指数，例如 6.02E23
decimal	实数的标准数学概念，它覆盖比 double 更小的范围，并由一系列被句点分开的数字组成，比如 9.06
timeInstant	日期和时间的联合，用来定义一个明确的时间实例，编码为字符串，2000-01-01T08:12:00.000 代表 2000 年 1 月 1 日 8:12，用秒和小数的秒来表达。这个类型经常表达为 YYYY-MM-DDThh:mm:ss.sss，但能直接后续一个 Z 去指明这个时间是 CUT 时间。可以选择地将时区通过使用后续 hh:mm 的一个 + 或 -，提供与 CUT 的差来说明。例如上面的日期和时间可以后续 -04:00。
timeDuration	日期和时间的联合，用来定义一段时间、间隔或持续时间。例如一个月被表达成 P0Y1M0DT0H0M0S，词汇模板为 PnYnMnDTnHnMnS，并且可在前边加 + 或 - 符号。当不需要精细的时间间隔时，这种描述的右侧可被修剪。例如 P2Y3M 代表 2 年零 3 个月。注意数字放在代表时间间隔的字符前面。秒用一个用数字来表达，它可以包括一个代表小数秒的小数。词汇表示法前面的减号表示负时间段

(续)

模式原始类型	定 义
recurringInstant	一种带有固定频率再现的时间实例，比如每一天，用一个破折号代替任何没有在timeInstant词汇模板提供的时间段。例如一个在每天 08:00出现的实例将被表达成——T08:00:00.000。
binary	任意长的二进制数据体
uri	URI引用

2. 生成和用户自定义的类型

正如名称所示，一个生成的数据类型是从现有的类型建立的，这个类型叫做基础类型。

XML 模式指定了一些广泛使用的生成类型。这些被列在表 7-4中。

表 7-4

生成的类型	基础类型	含 义
language	string	自然语言标识符；一个在 XML里与LanguageID相遇的记号，例如“en”
NMTOKEN	NMTOKENS	XML 1.0 NMTOKEN
NMTOKENS	string	XML 1.0 NMTOKENS
Name	NMTOKEN	XML 1.0 名称
Qname	Name	XML 1.0 限定名
NCNAME	Name	XML 1.0 “未开拓的”名称
ID	NCNAME	XML 1.0 属性类型ID
IDREF	IDREFS	XML 1.0 属性类型IDREF
IDREFS	string	XML 1.0 属性类型IDREFS
ENTITY	ENTITIES	XML 1.0 ENTITY
ENTITIES	string	XML 1.0 ENTITIES
NOTATION	NCName	XML 1.0 NOTATION
Integer	decimal	离散数字类型的标准数学概念（这里离散使它从数字定义分离出来）
non-negative-integer	integer	非负整数的标准数学概念
positive-integer	integer	正整数的标准数学概念
non-positive-integer	integer	负整数或零的标准数学概念
negative-integer	integer	严格的负整数的标准数学概念
date	recurringInstant	标准日期概念，是指一段时间间隔，从午夜开始，持续24小时
time	recurringInstant	与timeInstant左侧截去后相同，形如 hh:mm:ss.sss

我们用datatype元素声明了一个新类型。这个元素有 name和source属性。source属性值指明了新的类型来自的类型。这是一个最小的例子：

```
<datatype name="height" source="decimal"/>
```

我们通过添加 facets 进而说明一个新类型。这些必须适合基础类型，这就是说，只有顺序的 facets 可以被用于从顺序的基础类型产生的数据类型。典型情况下将要为新类型指定有限制的 facets，这些通过为限制的基础类型 facets 提供明确的数值来实现。例如，让我们声明一些产生的类型，表示大的和小的产品定单：

程序清单 7-32

```
<datatype name="largeOrder" source="integer">
  <minExclusive value="1000"/>
</datatype>

<datatype name="smallOrder" source="integer">
  <minExclusive value="0"/>
  <maxInclusive value="1000"/>
</datatype>
```

integer 类型具有有约束的 facets 表示范围，它们名为 minInclusive、minExclusive、maxInclusive 和 maxExclusive。上面的例子利用了这些可以实现：一个小定单可以包括在 1 和 1000 单位之间的所有订单。在我们的类型系统里一个大的定单是超过 1000 单位的所有订单。

7.7 简化了的 XML Data

在写本书时 XML 模式还不是一个推荐标准（2000 年 1 月），于是我们不能在这里提供一个正在使用的例子。但是，为了看到将怎样利用 XML 模式的功能，看一下用 XML 语法写成的模式不同的实现方式，我们把它称为简化的 XML Data，一个在 Microsoft 的 MSXML 解析器里实现的 XML Data 子集，能够在 IE5 或作为一个独立的组件使用它。在开始写的时候，简化的 XML Data 语法不同于可利用的 XML 模式工作草案，这样将告诉我们能够怎样使用 XML 模式在应用程序里带来的好处。

广泛使用起来的解析器不是只有 MSXML，但 Microsoft 由于他们的主动，正在积极地使用简化的 XML Data，特别是 BizTalk。这包括在分享电子商务市场词汇表所做出的努力。当 Microsoft 许诺在草案变成推荐书时采纳 XML 模式，结果立刻导致了很多人正在利用简化的 XML Data 建立原型甚至产品，这在 W3C 模式推荐书之前作为一个中间的方法。

由于这是一个我们现在能够工作的工具，并且它正被用在几个领域的原型，在本章倒数第二部分我们将要看一下简化的 XML Data 语法。由于已经看了语法，下面我们将建立几个例子，让你看到这几个新的模式功能。

IBM 在他们的 XML4J 解析器的 beta 版引进了对 XML 模式的部分支持。但是，自从 MSXML 拥有更丰富的支持并且是发行工具，我们将集中力量在简化的 XML Data 上。

什么是简化的 XML Data

正如我们说过的，简化的 XML Data（XML-DR）是整个 XML Data 提议的一个子集，根据子集包括的多少，它就像文档内容描述说明书包含的那些需要完成 DTD 任务的结构，大概提供了相同的功能。对于 DTD 提供的功能，它也提供了一个新的扩展。它是由一个搭载 Internet Explore

5.0的XML解析器里的叫做技术预览的工具实现。它也被一些商业工具支持，特别是 DTD/模式编辑器比如可扩展的XML Authority。这些正在做明确的调查，因为它对于实验可以利用并且正被大量的发明者使用。

模式支持

简化的XML Data在概念上与XML 模式的核心结构类似，只是语法有些微妙的不同。更为复杂的结构比如类型，不能被复制，但你需要的定义XML里词汇表的每个内容经常使用非常简单的语法。这是在简化的XML Data里说明的元素和它们的XML 模式等价物：

小心注意名字的使用，因为在XML模式和XML-DR模式之间有些微妙的不同（参见表7-5）。

XML-DR 模式的完整参考可以在 <http://msdn.microsoft.com/xml/reference/schema.start.asp>上在线搜寻。

(1) 模式

XML-DR里的Schema元素非常近似于XML 模式里的模式元素。这个元素完成下面的功能：

- 包含元素和属性声明。
- 命名模式。
- 声明在模式里使用的命名空间。

不像XML模式，XML-DR里的模式不使用包含 import、export和include元素的导言。相反，它们使用命名空间声明。每一个 XML-DR 模式必须声明XML Data和Microsoft的数据类型命名空间。如果遵守固定的命名习惯（下面我们讨论支持XML-DR的解析器时将要介绍这些），来自另一个命名空间的外部内容可能被解析器使用和验证。这里是一个忽略内容的模式例子：

程序清单 7-33

```
<Schema name="ShortSchema.xml" xmlns="urn:schemas-microsoft-com:xml-data"
        xmlns:dt="urn:schemas-microsoft-com:datatypes">
    ... <!-- Declarations here -->
</Schema>
```

(2) 元素和属性

元素和属性分别被声明在ElementType和AttributeType元素。

```
<elementType name="myElement" />

<ElementType>元素有五个重要属性（参见表7-6）。
```

表 7-5

XML 模式命令	XML-DR命令
schema	Schema
element	ElementType
elementRef	element
attribute	AttributeType
none	attribute
datatype	datatype
None	description
ModelGroup.group	group

表 7-6

ElementType属性	含 义
name	元素名称
content	描述可能包含在元素里的内容 :empty,textOnly(只对PCDATA), eltOnly (只对元素内容), mixed (PCDATA和元素)
dt:type	表示元素类型。这个属性与XML 模式里的<datatype>元素一致。有效值从XML数据类型预览工具得到
model	开放或关闭的内容模型
order	子元素的基本顺序：one (从一系列元素选出的一个) seq (特定的元素序列) many (以任意顺序可能出现或不出现的特定元素)

元素还能包含下面四种内容类型之一，这些内容类型用 <ElementType>元素的content属性来描述：

- 没有内容：empty。
- 只有文本：textOnly。
- 只有子元素：eltOnly。
- 混合文本和子元素：mixed。

我们能用<element>和<attribute>元素去约束已经声明的元素的内容。这些元素声明子元素和可以用于一个元素的属性。

<element>元素能有三个属性（参见表 7-7）。

表 7-7

属 性	描 述
type	与在模式里定义的<ElementType>的name属性值相一致
minOccurs	引用元素类型在元素里能出现的最小次数，当出现次数为 0，即元素是可选的，此值为0；当最少出现一次时，此值为1（缺省值为1）
maxOccurs	引用元素类型在元素里能出现的最大次数，当出现次数最多为 1次时，此值为1；当出现次数不限制时，此值为*（缺省值为1）

<attribute>元素也能有三个属性（参见表 7-8）。

表 7-8

属 性	描 述
default	属性的缺省值，不考虑它所重载的<Attribute>元素里提供的任何缺省值
type	与在模式里定义的< AttributeType>元素的name属性值相一致
required	指明属性是否必须出现在元素里，如果需要取值yes。如果在< AttributeType>元素里指明了就不需要了

让我们看一些简单的元素声明和它们的 DTD等价物。首先我们有一个父元素叫做 <Fex>，它包含一个子元素叫做 <Tex>。

```
<ElementType name="Fex" content="mixed" order="many">
  <element type="Tex"/>
</ElementType>
```


这里，一些为<Tex>声明的<ElementType>将在模式的别的地方被包含。在 DTD里，我们刚刚看到的声明将是：

```
<!ELEMENT Fex (#PCDATA | Tex)*>
```

接下来，有一个<Person>元素，它有子元素<FirstName>、<MI>和<LastName>:

```
<!ELEMENT Person (FirstName, MI, LastName)>
```

使用XML-DR，这将看起来像下面那样：

程序清单 7-34

```
<ElementType name="Person" content="eltOnly" order="seq">
  <element type="FirstName"/>
  <element type="MI"/>
  <element type="LastName"/>
</ElementType>
```

内容模型通过使用 order属性和group元素组成。因此，如果想要变得更复杂，我们需要看一下<group>元素的属性以及使用已经看过的 <element>元素的属性（参见表 7-9）。

表 7-9

<group>属性	含 义
maxOccurs	group可能出现的最大次数。可取值0或*（很多）
minOccurs	group可能出现的最小次数。可取值0（可选）或1（至少出现1次）
order	包含的元素和组的顺序。可能是 one（从group里选出一个元素）、seq（序列里的每个元素）或 many（group里可能或不可能，按一定顺序出现的任意元素）

于是，如果我们想根据 XML-DR声明下面的内容模型：

```
<!ELEMENT Foo ((X | Y) | (A, B?, C))>
```

它将像下面那样：

程序清单 7-35

```
<ElementType name="Foo" content="eltOnly" order="one">
  <group order="one">
    <element type="X"/>
    <element type="Y"/>
  </group>
  <group order="seq">
    <element type="A"/>
    <element type="B" minOccurs="0"/>
    <element type="C"/>
  </group>
</ElementType>
```

为了声明属性，我们需要<AttributeType>元素，它有一个必选的和四个可选的属性。通过包含一个 <attribute>元素将属性与元素结合， <attribute>元素作为用来声明元素的<ElementType>的子元素，就像处理元素一样。 <attribute>元素的type属性参阅了声明属性的<AttributeType>（参见表7-10）。

表 7-10

AttributeType属性	含 义
name	必须的；属性名称
default	属性缺省值；必须与 Dt:type一致
dt:type	属性定义的数据类型
dt:values	当 dt:type是枚举时一系列可能的值
required	指明属性是否出现在元素的所有的实例中； True或False

在Windows 2000之前的IE5解析器版本，dt:type可以具有XML 1.0原始类型：entity、entities、enumeration、id、idref、idrefs、nmtoken、nmtokens、notation和string。下面讨论的数据类型的整个范围均被搭载在Windows 2000上的解析器支持。

这里是我们怎样去添加一个必须的 age属性到事先在DTD里声明的<Person>元素里：

```
<!ELEMENT Person (FirstName, MI, LastName)>
<!ATTLIST Person age CDATA #REQUIRED>
```

在XML-DR里这些变成了：

程序清单 7-36

```
<AttributeType name="age" required="yes"/>

<ElementType name="Person" content="eltOnly" order="seq">
  <attribute type="age">
    <element type="FirstName"/>
    <element type="MI"/>
    <element type="LastName"/>
  </ElementType>
```

<attribute>元素也可用来声明一个缺省值或指明这个属性是否为必须。如果相关联的<AttributeType>提供了required属性，我们不需要在<attribute>元素里重复那个属性。

另一个有趣的变化是一个<AttributeType>元素可以出现在<ElementType>实例范围里。那种情况下，声明的属性类型只是在<ElementType>声明范围里有效，并且在模式的别的地方不能引用。这里是用age声明的同一个<Person>和age，age被定义成单独用于<Person>：

程序清单 7-37

```
<ElementType name="Person" content="eltOnly" order="seq">
  <AttributeType name="age" required="yes"/>
  <element type="FirstName"/>
  <element type="MI"/>
  <element type="LastName"/>
</ElementType>
```

(3) 组

我们看到了<group>元素用于<ElementType>元素声明。它的使用非常简单，但这个元素的属性所允许的数值不同于我们在XML 模式看到的那些。首先，它最重要的功能不灵活。all按顺序枚举了类型，但也不被支持。<Group>具有maxOccurs、MinOccurs和Order属性，这些我们在前面部分看过了。

(4) 注释

XML-DR为内嵌文档提供了描述性元素。

```
<description>This is how you use the description element</description>
```

当然，XML类型注释<!--Some comment here-->同样工作得很好。但是我们能用描述性元素捕捉与模式相关的注释，比如使用方法注释，它可用于一种特殊工具的使用说明。XML-DR自己不区分两种格式的注释，而只提供二者任意一种。

(5) 数据类型

XML-DR使我们能在IE 5.0里使用数据类型技术预览来提供强大的元素和属性录入功能。除了dt:type属性之外这些使用了<AttributeType>元素。这个元素——<datatype>拥有单一属性dt:type，以指明父元素的类型。

```
<ElementType name="Age">
  <datatype dt:type="int"/>
</ElementType>
```

IE 5.0里强大的录入支持不仅提供XML 1.0原始类型，还支持大量对PC应用普遍的派生类型。表7-11是支持的附加的类型：

表 7-11

数据类型	含 义
bin.base64	基于64二进制编码的MIME类型
bin.hex	由16进制表示的字节
boolean	0 (false) 或1(true)
char	单字符
date	ISO 8601日期（不含时刻）
datetime	ISO 8601日期时刻，带有可选的时间和分数秒，可达到十亿分之一秒的分辨率
datetime.tz	datetime，带有可选的时区
fixed.14.4	数字类型，小数点前不超过14位，后面不超过4位
float	实数，带有可选的符号、小数位和指数
int	整数
number	通用数字类型，在位数上没有限制；可能带有可选的符号、小数位和指数
time	ISO 8601格式的时间组
time.tz	同Time，带有时区
i1	1字节有符号整数
i2	2字节有符号整数
i4	4字节有符号整数
r4	4字节有符号实数
r8	8字节有符号实数
ui1	1字节无符号整数
ui2	2字节无符号整数
ui4	4字节无符号整数
uri	URI字符串
uuid	16位表示字节并组成一个COM类型的UUID；连字号可选，如果有就忽略

在 Windows 2000 之前的解析器版本,上面表格里描述的派生数据类型被限制用于格式正规的 XML 文档。这些版本的 XML-DR 模式不支持验证。

7.7.1 MSXML 对命名空间和模式的支持

MSXML 具有命名空间和简化的 XML Data 模式预览的功能。你必须使用 Microsoft 的基于 COM 的 XML 解析器——MSXML,它与 Internet Explorer 5.0 搭载或者如果你想在代码里支持 XML-DR,可以从他们的站点下载。这里反复强调 XML-DR 是 Microsoft 的技术预览,而不是 W3C 的成就。它严格地被 Microsoft 所有并只由它提供支持。在早期对模式做很多支持尝试的同时,MSXML 的 XML-DR 支持潜在地对于那些想要加入到 XML 元数据前线的程序员来说非常有用。

MSXML 通过 XML DOM 扩展支持 XML-DR 和命名空间。

早期 IBM 的 XML4J 解析器的尝试版本 (EA2) 部分支持 1999 年 9 月的 XML 模式草案。IBM 指出这种支持今后可能在解析器的发行版里出现。鉴于目前 XML 模式的变化形式,提供正在运行的例子确实非常困难,当你读完这本书时会明白这一点。

1. 命名空间

MSXML 里对命名空间的支持是非常强大的。当访问 DOM 里一个已经被命名空间验证过的结点时,你能使用 `basename` 或者 `prefix` 以及命名空间 URI 属性去获得与命名空间相关的信息 (参见表 7-12)。

表 7-12

属 性	含 义
<code>basename</code>	返回字符串,它的值是未经验证的结点基本名
<code>prefix</code>	返回字符串,带着命名空间前缀
<code>namespace URI</code>	返回字符串,带着命名空间的 URI,与结点的命名空间前缀一致

这里有两种对于程序员有吸引力的办法处理命名空间。如果你想要使用扩展了的 Microsoft XML DOM 创建一个经命名空间验证过的结点,不能使用 `createElement()` 去直接创建。相反,必须用文档对象的 `createNode()` 方法。它用枚举的数值去指明创建的结点类型,验证了的名称以及相关的 URI:

```
qualifiedNode = doc.createNode(1, "pub:Book", "urn:myschemas-pub");
```

另外,你能通过使用后续设置了 `xmlns` 属性的 `createElement()` 创建一个缺省的命名空间:

```
qualifiedNode = doc.createElement("Book");  
qualifiedNode.setAttribute("xmlns", "urn:myschemas-pub");
```

但是应注意到,在后面的情况,失去了用命名空间验证其他名称的能力。

上面的 JavaScript 程序行用验证过的名称 `<pub:Book>` 在 `myschemas-pub` 命名空间里创建了一个元素。表 7-13 中有部分结点类型枚举的列表,它们被 MSXML 支持,是对命名空间的兴趣所在。

表 7-13

枚 举	值
NODE_ELEMENT	1
NODE_ATTRIBUTE	2
NODE_ENTITY_REFERENCE	5
NODE_ENTITY	6
NODE_NOTATION	12

一个结点的 attribute 集合支持 getQualifiedItem() 方法，允许你根据属性验证过的名字进行搜索。这种方法接收所需要属性的基本名称和前缀，如果属性被发现则返回一个结点对象。为了找到一个属性 pub:isbn，将要进行如下调用：

```
FoundAttr = nodeAttrs.getQualifiedItem("isbn", "pub");
```

2. 验证

像任何其他的解析器一样，MSXML 将要验证一个 DTD 文档。并且如果一个模式由前缀 x-schema 来提供，MSXML 假定接在冒号后面的名称是一个 XML-DR 类型的模式并且要试着将它装入并用它验证文档。

```
<Book xmlns="x-schema:PubCatalog.xml">  
  <!-- PubCatalog is assumed to be a schema -->
```

如果你验证了你的 XML-DR 模式文档，就有机会获得权利使用模式结点，它在你文档里定义了特殊的结点。这些由 definition 属性完成。如果调用了对应于一个元素或属性的结点，你将得到一个包含可应用的 <ElementType> 或 <AttributeType> 元素的结点：

```
SchemaNode = node.definition;
```

7.7.2 数据类型

MSXML 里的数据类型支持对 XML-DR 预览是独立的但又是补充。即便你不用模式，也能在格式正规的 XML 里使用强行分类的元素和属性。为了做到这一点，必须要在你的文档里声明 Microsoft 数据类型命名空间：

```
<MyRootElement xmlns:dt="urn:schemas-microsoft-com:datatypes">
```

一旦做了这些，你能在结点里使用支持的类型。可以预先使用 nodeValue() 去得到一个元素或属性的值的地方，能调用 nodeTypedValue() 去获得强行分类的值。假想你有一些包含元素的 XML：

```
<PageCount dt:dt="int">350</PageCount>
```

你可以用 JavaScript 接下来实现下面的内容：

```
sCount = node.nodeValue;      <!-- returns the string "350" -->  
Count = node.nodeTypedValue;  <!-- returns the numeric value 350 -->
```

如果有两个结点 <node1> 和 <node2>，代表来自两个不同文档的 <PageCount> 元素，你能够像下面那样得到整个页数：

```
TotalCount = node1.nodeTypedValue + node2.nodeTypedValue;
```

属性 `nodeTypeString` 返回一个固定字符串，表示数据类型。例如我们的 `<PageCount>`，这个属性应该返回 “int”。

7.8 图书目录中的变化

目前为止，我希望能渴望应用命名空间和模式信息到本书目录的实例里。尽管我们不愿在本书剩下的篇幅里继续这个模式，但我还是打算去演示怎样使用我在这一章里表示过的内容来改进我们对本书出版信息的理解和组织。

7.8.1 为什么烦恼

我们的 Book Catalog DTD 出了什么问题？实际上，它正开始有起色。如果我们合适地验证了目录文档，有关 Book Catalog 的每一样东西必须进入一个 DTD。本章前面普遍应用我们特有的 DTD 时，所有的批评都瞄准了 DTD。

我们要做的第一件事是将出版目录范围分成两个分离的模式，一个反映处理作者的命名空间，另一个处理目录信息。另外，我们能够对一些的属性和元素提供很强的分类，因此到了书写处理目录的应用程序时，我们的工作变得更加轻松。因为 XML 模式是一种流的状态，它接近推荐书里的状态，我们将要在 MSXML 里使用 XML-DR 版本的模式。

1. 分割

在第3章见到的 `catalog.dtd` 给了我们许多概念。一个目录一定需要 `books`，但如果作者存在目录外面不是更好吗？毕竟，如果我们需要为标记每本书的真实内容写一个模式，或许想同时在这里包含作者信息。这就是将 Book Catalog DTD 分成两个独立的模式的主要动机：Catalog 和 Author。当我们想要创建一个目录文档，能够为 Catalog 模式声明一个缺省的命名空间，然后使用一个验证过的命名空间去引进 Author 模式。

2. 附加的语法

在 `catalog.dtd` 里有几个属性，能够提供强大的分类功能。如果我们包含数据类型，它将非常容易地用来计算总页数，我们将肯定喜欢能够来自目录的一个定单计算定单总数。因此需要仔细查阅 Catalog 模式，看什么样的属性应该用类型信息检验。

3. 元数据发现

用 XML 语法创建一个模式对于程序员非常有用，在那里为他们提供一些对编写程序的支持，对此操作手册的目录文档根据我们的模式做了标记。我们提供的最大支持是简单地使 DTD 带有模式表单形式。一旦它采用 XML 语法，程序员能够使用相同的解析器，他们曾用它与 XML 文档实例一起来发现元数据背后的含义。

假设你不熟悉我们的模式。你能够用 `<definition>` 元素调查单个元素。这在文档浏览器里非常有用。一个用户可能单击为得到特定条目的附加信息，查看与之相连的元数据。我们不愿意表现 XML 定义，当然如果一个条目是数值类型，我们可能会显示。在枚举的情况下，当然显示条目能取的数值的范围。主要信息能够看到很重要，比如说一个属性是否是必须的。只要我们用 XML 语法提供一个模式，所有这些能够在文档实例被读到时发现。在我们将 `catalog DTD` 转成

一个模式之后，将显示能够怎样去使用 DOM在模式里生成相互协调的元素。这些将采用一个模式，同时提供给你一些参照元素以及它们是怎样使用的。

7.8.2 重铸DTD

仔细看一下我们的 DTD。我们将做完一个到 XML-DR 格式的转换，正像我们所做的，显示对定义不断的改进。

对于XML-DR 模式文件扩展还没有明确的一致意见。Microsoft发起者趋向使用xml，与此同时，一个商业化可用的工具使用xdr。如我们看到的，W3C 模式工作组倾向为他们的模式使用xsd。我将在下面的例子里使用xml。无论什么情况，模式是XML，于是它的MIME 类型保留text/xml。

目录分成三个部分（参见图 7-3）：

- 有关出版者的出版商信息（Publisher ），
- 包含描述信息的线索（Threads ），
- 包含有关书籍的信息（Books ），

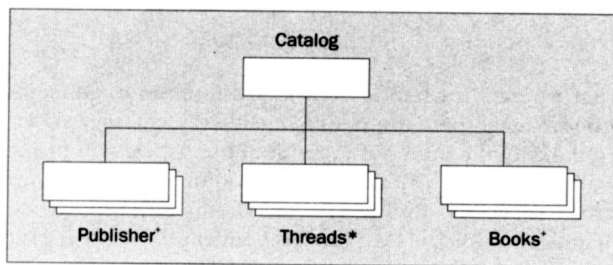


图 7-3

出版商信息部分也包括作者信息细节，但是我们正打算删除作者信息，将它们放在独立的作者模式里，于是我们能在目录模式里借用，并在其他地方使用它们。因此，在回到目录的其他部分之前，让我们从作者模式开始。

1. 作者模式

我们应该首先看一下作者模式，这是因为接下来建立的目录模式将从它那里借鉴些内容。第一件要做的事是去掉 <Author> 元素声明和一切从属于它的东西，建立新的模式文件——authors.xml。文件开始应该声明与 XML 1.0 的一致性，命名模式以及声明 XML-DR 和数据类型命名空间：

程序清单 7-38

```
<?xml version="1.0"?>
<Schema name="authors.xml"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
```

缺省的命名空间是 XML-DR 以及数据类型命名空间将用前缀 dt 加别名。Author 元素是我们的

起点。它只是依次包含与名称相关的、<Biographical>和<Portrait>元素等元素内容：

```
<!ELEMENT Author ((FirstName, MI?, LastName, Biographical, Portrait)>
<!ATTLIST Author authorCiteID ID #REQUIRED>
```

在XML-DR里，这些变成了：

程序清单 7-39

```
<AttributeType name = "authorCiteID" dt:type = "ID" required = "yes"/>
<ElementType name = "Author" content = "eltOnly" order = "seq">
  <attribute type = "authorCiteID"/>
  <element type = "FirstName"/>
  <element type = "MI" minOccurs = "0" maxOccurs = "1"/>
  <element type = "LastName"/>
  <element type = "Biographical"/>
  <element type = "Portrait"/>
</ElementType>
```

我们为authorCiteID保留了XML ID类型以用来保存作者和书籍之间的连接。注意特别是在MI上的重要部分。它可能出现零次或者一次，这就是说，它是可选的。现在声明 <Author>的子元素：

程序清单 7-40

```
<ElementType name = "FirstName" content = "textOnly"/>
<ElementType name = "MI" content = "textOnly"/>
<ElementType name = "LastName" content = "textOnly"/>
<ElementType name = "Biographical" content = "textOnly"/>
<AttributeType name = "picLink"/>
<ElementType name = "Portrait" content = "empty">
  <attribute type = "picLink"/>
</ElementType>
```

关闭最高级<Schema>元素和你已经做了。现在你有一个可以重用的模式，它能被引进我们标记作者元素信息的任何地方。

2. 目录模式

既然从目录DTD移走了作者信息，将它们放在单独的模式里，我们将注意力转向重建 XML 里的目录数据。我们将称这个模式为 PubCatalog.xml。当需要包含作者细节时，这些将从作者模式借用过来。这里是开放的信息：

程序清单 7-41

```
<?xml version = "1.0"?>
<Schema name = "PubCatalog.xml"
  xmlns = "urn:schemas-microsoft-com:xml-data"
  xmlns:dt = "urn:schemas-microsoft-com:datatypes"
  xmlns:athr = "x-schema:authors.xml">
```

注意怎样为我们新建立的作者模式——authors.xml用别名前缀athr添加一个命名空间。

让我们深入研究一下：我们以<Catalog>元素开始，它包含其他信息。这里包括<Publisher>、<Thread>和<Book>元素，就像我们在早期的 catalog.dtd里有过的那样，其中的每一个可能出现

很多次：

程序清单 7-42

```
<ElementType name = "Catalog" content = "eltOnly" order = "seq">
  <element type = "Publisher" minOccurs = "1" maxOccurs = "**"/>
  <element type = "Thread" minOccurs = "0" maxOccurs = "**"/>
  <element type = "Book" minOccurs = "1" maxOccurs = "**"/>
</ElementType>
```

下面我们需要声明 isbn 属性，它将用在我们刚刚声明的 <Publisher> 和 <Book> 元素里：

```
<AttributeType name = "isbn" required = "yes"/>
```

(1) 出版商

我们需要着手的下一部分是刚刚声明的 <Publisher> 元素的内容。它还包含在 DTD 里看到的开始三个子元素，但是已经为作者详细信息创建了一个独立的模式，于是需要去参阅那个命名空间并借用它（参见图 7-4）。

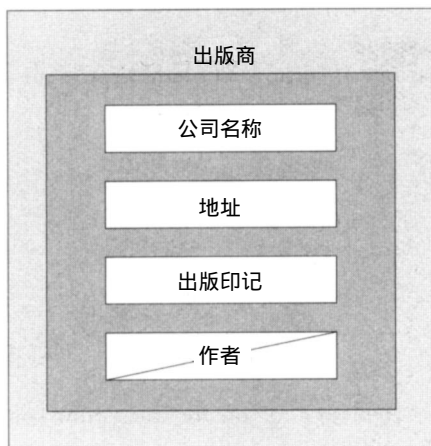


图 7-4

正如提到的，我们能够利用 <description> 元素去生成有关对于处理应用程序可利用的 DTD 信息，这就是我们要做的，这里我们正使用它来指定 <Publisher> 元素可用于出版商信息。

程序清单 7-43

```
<ElementType name = "Publisher" content = "eltOnly" order = "seq">
  <description> Publisher section </description>
  <attribute type = "isbn"/>
  <element type = "CorporateName"/>
  <element type = "Address" minOccurs = "1" maxOccurs = "**"/>
  <element type = "Imprints"/>
  <element type = "athr:Author" minOccurs = "0" maxOccurs = "**"/>
</ElementType>
```

深入研究模式，<CorporateName> 元素，它非常简单，包含 DTD 里的 PCData，于是我们指定它的内容只是文本：

```
<ElementType name = "CorporateName" content = "textOnly"/>
```

下面我们具有地址信息，你可以想起包含一个 yes/no 的 headquarters 属性枚举，首先定义：

程序清单 7-44

```
<AttributeType name = "headquarters"
    dt:type = "enumeration" dt:values = "yes no"/>
<ElementType name = "Address" content = "eltOnly" order = "seq">
    <attribute type = "headquarters"/>
    <element type = "Street" minOccurs = "1" maxOccurs = "*" />
    <element type = "City" />
    <element type = "PoliticalDivision" />
    <element type = "Country" />
    <element type = "PostalCode" />
</ElementType>
```

注意 XML-DR 里的 enumeration 数据类型表单。接下来，我们声明用在地址元素里的元素：

程序清单 7-45

```
<ElementType name = "Street" content = "textOnly"/>
<ElementType name = "City" content = "textOnly"/>
<ElementType name = "PoliticalDivision" content = "textOnly">
    <description>State, province, canton, etc.</description>
</ElementType>
<ElementType name = "Country" content = "textOnly"/>
<ElementType name = "PostalCode" content = "textOnly"/>
```

<Publisher> 元素的第三个子元素打算留下出版商的特征：

程序清单 7-46

```
<ElementType name = "Imprints" content = "eltOnly" order = "seq">
    <element type = "Imprint" minOccurs = "1" maxOccurs = "*" />
</ElementType>

<AttributeType name = "shortImprintName" dt:type = "ID"/>
<ElementType name = "Imprint" content = "textOnly">
    <attribute type = "shortImprintName" />
</ElementType>
```

<Publisher> 元素的第四个子元素具有 DTD 里的作者信息细节，但是鉴于我们已经将它删除，接下来转向讨论 <Thread>。

(2) 线索

<Thread> 用来指明书的分类区域。通过书封底上的代码，你能够看到三个不同的线索，它们用来给书籍分类，例如在书店，当决定将书放到哪部分时它们将被使用。

程序清单 7-47

```
<AttributeType name = "threadID" dt:type = "ID"/>
<ElementType name = "Thread" content = "textOnly">
    <description>
        Subject threads consist of one or more books
        related by some thread of study
    </description>
    <attribute type = "threadID" />
</ElementType>
```

我们再一次使用<description>元素去解释什么样的线索被使用。

(3) 书

最后一部分来处理书自身的内容。就像我们在 DTD 章节里说明的，书一定包含标题、摘要、介绍的主题种类和价格（参见图 7-5）。

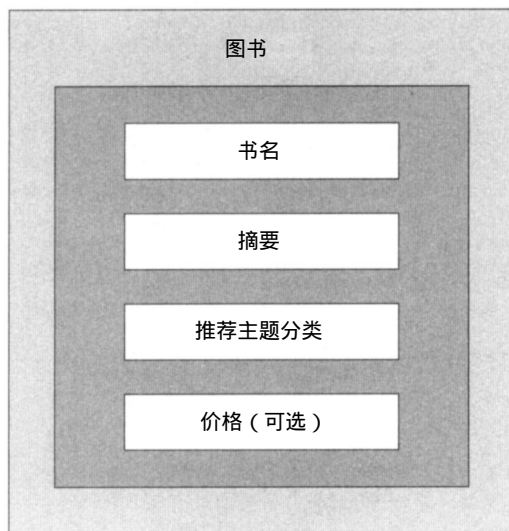


图 7-5

在我们定义这些元素之前，必须定义几个属性：

```
<AttributeType name = "ISBN" dt:type = "ID" required = "yes"/>
<AttributeType name = "level"/>
<AttributeType name = "pubdate" required = "yes"/>
```

下面看看 pageCount 属性。这里我们将真正使用强大的数据类型功能。我们将给这个属性一个整数类型：

```
<AttributeType name = "pageCount" dt:type="int" required = "yes"/>
```

然后我们继续各种引用：

程序清单 7-48

```
<AttributeType name = "authors" dt:type = "IDREFS"/>
<AttributeType name = "threads" dt:type = "IDREFS"/>
<AttributeType name = "imprint" dt:type = "IDREF"/>

<AttributeType name = "shortImprintName" dt:type = "ID"/>
```

既然设置了将要使用的属性，声明<Book>的内容，它使用了刚刚声明的属性和几个子元素：

程序清单 7-49

```
<ElementType name = "Book" content = "eltOnly" order = "seq">
  <description> Book summary information (no content) </description>
  <attribute type = "ISBN"/>
  <attribute type = "level"/>
```

```

<attribute type = "pubdate"/>
<attribute type = "pageCount"/>
<attribute type = "authors"/>
<attribute type = "threads"/>
<attribute type = "imprint"/>
<element type = "Title"/>
<element type = "Abstract"/>
<element type = "RecSubjCategories"/>
<element type = "Price" minOccurs = "0" maxOccurs = "1"/>
</ElementType>

```

然后，描述这些子元素的内容：

程序清单 7-50

```

<ElementType name = "Title" content = "textOnly"/>
<ElementType name = "Abstract" content = "textOnly"/>
<ElementType name = "RecSubjCategories" content = "eltOnly" order = "seq">
  <element type = "Category"/>
  <element type = "Category"/>
  <element type = "Category"/>
</ElementType>
<ElementType name = "Category" content = "textOnly"/>

```

<Price>元素声明又将我们带到数据类型支持。货币属性需要一个枚举，同时元素的文本值本身应该是一个数值类型以适于描述货币：

程序清单 7-51

```

<AttributeType name = "currency" dt:type = "enumeration"
  dt:values = "USD GBF CD" required = "yes"/>
<ElementType name = "Price" dt:type="fixed.14.4" content = "textOnly">
  <attribute type = "currency"/>
</ElementType>
</Schema>

```

这就是说，通过一些从 DTD语法到 XML-DR 的转换，以及一些附加的强大数据类型，我们创建了一个新的目录模式，它通过命名空间的支持，重用了作者模式。这给了我们与 DTD 所提供的相同种类的验证支持，即我们改变里子 catalog.xml 文件的根元素以反映使用了模式：

```

<?xml version = "1.0"?>
<Catalog xmlns = "x-schema:PubCatalog.xml">

```

注意命名空间声明排除了对 DOCTYPE 的声明。

7.8.3 模式协调

拥有模式里的所有元素和它们的内容的简单列表将是一件非常好的事情。这就是说，对于每一个元素声明，我们将有一个许可的子元素和用于它属性的列表。这样，我们应该能够测量改变任何特别的元素和属性带来的影响。因为 XML-DR 模式使用 XML 语法，所以能够使用 MSXML 和一些 JavaScript 去产生这种效果。图 7-6 就是当它完成并在 PubCatalog.xml 模式文件里指出来时看起来的样子：

SchemaConcordance.html 源代码可从我们的站点 <http://www.wrox.com> 得到。不同于提供一个恰当的 URL 用于你需要的参考索引文件，这里没有配置需求。

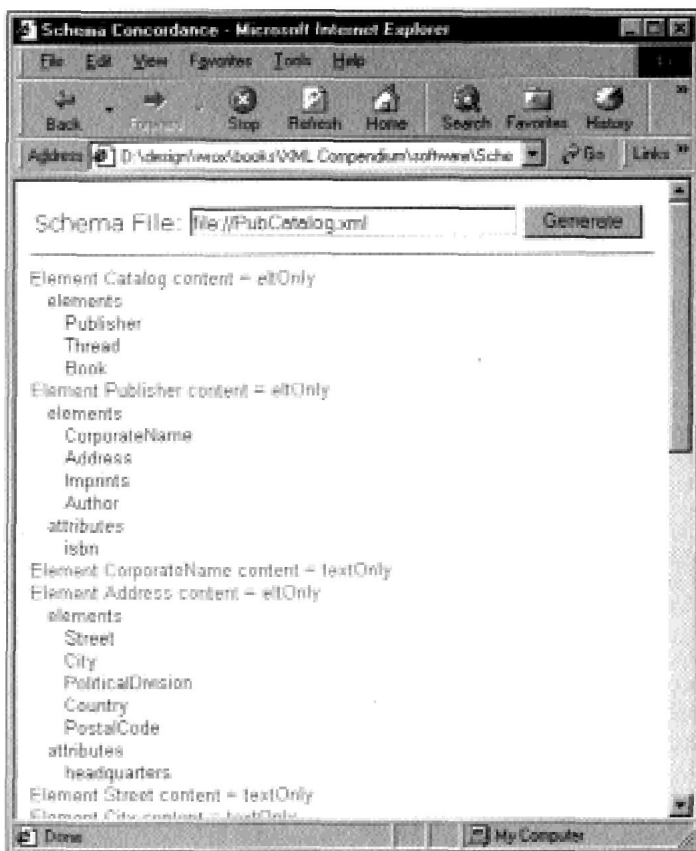


图 7-6

1. 搜索元素

我们知道，一个模式文档以 `<Schema>` 根元素开始。它的子元素将是 `<ElementType>` 和 `<AttributeType>` 元素。每个用 `<ElementType>` 元素声明的元素包含一系列元素和属性。这在某种程度上简化了我们的工作。所有我们要做的是遍历这个 `<Schema>` 元素的子结点列表，并处理每一个搜索的 `<ElementType>` 元素。这里是所需要的代码的核心部分：

程序清单 7-52

```
if (parser.documentElement.nodeName == "Schema")
{
    for (var ni=0; ni < parser.documentElement.childNodes.length; ni++)
    {
        if (parser.documentElement.childNodes(ni).nodeName == "ElementType")
            CrossRefElement(parser.documentElement.childNodes(ni));
    }
}
```

我们知道子元素的序号，于是通过一个简单的循环遍历整个文档。元素结点的 `nodeName` 属性可以让我们通过查找 `<ElementType>` 名称搜索元素声明。

2. 处理一个元素声明

函数 `CrossRefElement()` 接收一个 `<ElementType>` 元素结点并列出它的内容。这就是一个比较困难的地方。这里并不担保 `<element>` 和 `<attribute>` 元素将被筛选。模式能以 `ElementType` 为序在元素前列出属性，然后以另外的顺序反转它们，或者甚至混合这两者。我们需要一个连续的顺序，这样能够在输出里加入适当的标题。我们将必须建立两个数组，一个用作元素名称，一个用作属性名称，然后在结束元素声明时显示结果。这里是函数 `CrossRefElement()` 的一部分，用来提取元素声明信息：

程序清单 7-53

```
var rChildElements = new Array();
var rAttributes = new Array();
var WorkNode;
var nEltCount = 0;
var nAttrCount = 0;

for (ni = 0; ni < eltNode.childNodes.length; ni++)
{
    WorkNode = eltNode.childNodes(ni);
    switch (WorkNode.nodeName)
    {
        case "element":
            rChildElements[nEltCount++] =
                WorkNode.attributes.getNamedItem("type").text;
            break;

        case "attribute":
            rAttributes[nAttrCount++] =
                WorkNode.attributes.getNamedItem("type").text;
            break;

        case "group":
            SqueezeGroup(WorkNode, rChildElements, rAttributes);
            nEltCount = rChildElements.length;
            nAttrCount = rAttributes.length;
            break;
    }
}
...
```

当遇到一个 `<element>` 和 `<attribute>` 模式元素时，得到 `type` 属性的值，我们知道它是相关的 `<ElementType>` 和 `<AttributeType>` 元素的名称。通过使用 `getNameItem()` 函数做到这一点，Microsoft 在 MSXML 里使用的 DOM 扩展很明确地通过名称得到属性。如果模式不包含组，我们的工作就已经完成。因为组涉及我们需要的特定的元素和属性信息，所以必须调用另一个函数 `SqueezeGroup()`。这个函数看起来几乎同在上面看到的一样：

程序清单 7-54

```
function SqueezeGroup(node, rElts, rAttrs)
{
    var nEltCt = rElts.length;
    var nAttrCt = rAttrs.length;
    var childNode;

    // Fix up indices for empty arrays
```

```

if (nEltCt < 0)
    nEltCt = 0;
if (nAttrCt < 0)
    nAttrCt = 0;

for (var nj = 0; nj < node.childNodes.length; nj++)
{
    childNode = node.childNodes(nj);

    switch (childNode.nodeName)
    {
        case "element":
            rElts[nEltCt++] = childNode.attributes.getNamedItem("type").text;
            break;

        case "attribute":
            rAttrs[nAttrCt++] =
                childNode.attributes.getNamedItem("type").text;
            break;

        case "group":
            SqueezeGroup(childNode, rElts, rAttrs);
            nEltCt = rElts.length;
            nAttrCt = rAttrs.length;
            break;
    }
}
}

```

SqueezeGroup()忽略了group结点和包含元素和属性名称的数组。到目前为止，数组里可能有了一些名称，所以我们必须基于当前数组的长度设置数组的索引：

```

var nEltCt = rElts.length;
var nAttrCt = rAttrs.length;

```

由于SqueezeGroup()能添加数量，CrossRefElement()在当控制从SqueezeGroup()返回到它那里时必须重置它的索引：

程序清单 7-55

```

case "group":
    SqueezeGroup(WorkNode, rChildElements, rAttributes);
    nEltCount = rChildElements.length;
    nAttrCount = rAttributes.length;
    break;

```

最后，因为组可能包含其他的组，我们回归调用SqueezeGroup()去确认得到了组的所有信息：

程序清单 7-56

```

case "group":
    SqueezeGroup(childNode, rElts, rAttrs);
    nEltCt = rElts.length;
    nAttrCt = rAttrs.length;
    break;

```

3. 显示结果

一旦一个<ElementType>元素被完全处理后，我们能使用 DHTML用叫做DIV的名字显示结果。CrossRefElement()的最后一部分完成了该功能：

程序清单 7-57

```
sEltHeader = "Element " + eltNode.attributes.getNamedItem("n"
    " content = " + eltNode.attributes.getNamedItem("n"
ListLine(sEltHeader, "green");
tabsize += 12;
// List all child elements
if (rChildElements.length > 0)
{
    ListLine("elements", "blue");
    tabsize += 12;
    for (ni = 0; ni < rChildElements.length; ni++)
        ListLine(rChildElements[ni], "black");
    tabsize -= 12;
}
// List all attributes
if (rAttributes.length > 0)
{
    ListLine("attributes", "blue");
    tabsize += 12;
    for (ni = 0; ni < rAttributes.length; ni++)
        ListLine(rAttributes[ni], "black");
    tabsize -= 12;
}
tabsize -= 12;
```

ListLine()是一个实用的函数，它带有一些文本和颜色字符串，并用合适的颜色将文本插入DIV。变量tabsize和listline是全局变量，用来控制相对应的文本区域。

7.9 小结

我们看到写在XML里的命名空间和模式提供一些强有力的用来表现的新工具。它们帮助我们在建立与几个模式有关的单一文档实例时克服困难。这个能力意味着我们在必要或将大而复杂的模式分成小而更加易管理的模式时，能扩展现有的模式。同时，XML里的模式提供了许多其他好处，这些好处正表明了DTD相应的缺点。

RDF非常强大，尽管强大的功能可能使它对于日常应用太过于复杂。W3C模式就是为XML程序员将元数据引入中心，但它还不是一个标准。同时，在W3C XML模式标准化之前，XML-DR和它相关的数据类型给我们一个与XML模式在结构上非常接近的工具，并允许我们立刻写一些代码。当XML-DR没有提供在XML模式里许诺的能力的全部范围，我们能够对这些例子做些改进。特别是给Book Catalog例子带来下面的好处：

- 通过词汇表分割，更好地组织和再利用复杂域。
- 当转自或转出到XML时强大的类型化数据。
- 保证DTD精确（模式在重要地方允许更大的灵活性）。
- 用XML语法表现我们的问题，允许我们使用传统的XML解析器去读和处理模式。

元数据正从高级的学院团体移向XML程序员日常的工具箱。元数据方面丰富的研究成果在W3C的控制下正在产生集中而实际的标准。确实，XML模式在它们像DOM那样被支持后，将要为程序员做同样多的工作。MSXML里的XML-DR支持暗示了这种可能性。或许在你读到这些的时候，W3C可能已出版了模式推荐书。