返回总目录

目　　录

第二部分　XML 的语言基础

第五章　XML 语法

本章概括了 XML 的基本语法，对 XML 文档的语法构件进行了讨论。读者可以了解 XML 文档的编写方法和规范，并能够在此基础上开始模仿性地设计一些简单的结构化 XML 文档。

本章包括以下内容：

- XML 文档
- XML 声明
- 注释
- 属性与标记
- 实体参考
- CDATA 节
- 文档类型定义 DTD
- XML Schema
- 名字空间
- 正规有效的 XML 文档
- 数据岛
- XML 的相关标准

XML 是比现在我们正在使用的 HTML（Hyper Text Markup Language）更具有智能性的一种语言。如果读者了解 HTML 语法的话，那对制作 XML 的 Web 网页将很有帮助。在 HTML 4.0 中大约有 300 个不同的标记，大多数标记都有各自的属性。这样在应用时就会有数千种组合，为了精确地控制 HTML 的页面，使编写出的 HTML 页面尽量美观，必须熟练掌握这些标记和属性。XML 具有比 HTML 更强大的可扩展性，但它却不是靠繁多的标记和属性。相反的，XML 几乎没有预先定义好的标记和属性，XML 允许读者自己定义所需的标记和属性。

例如，我们可以用<pcbook>作为标记，其中的内容为计算机书籍的资源信息（title、author、publisher 等）。虽然 XML 中标记使用的自由度很大，但读者所使用的这些标记的创建也不是完全随意的，它们也要遵守 XML 中的特定的规则和语法，而且还要保证编写的 XML 文档具有良好结构。

下面介绍 XML 的语法，XML 基本语法的应用，学习如何编写正规的 XML 文档。通过本章的学习，相信读者将会对 XML 有一个初步的认识，会编写简单的 XML 文档，进而成为精通 XML 的高手。

5.1　XML 文档

XML 是一种标识语言。一个 XML 元素是由开始标签、结束标签以及标签之间的数据构成的。开始和结束标签用来描述标签之间的数据。标签之间的数据被认为是元素的值。例如，在下面一个 XML 元素的例子中，元素 title 的值是“Microsoft Windows 2000 网络基础结构”。

< title > Microsoft Windows 2000 网络基础结构</ title >

一个基本的 XML 文档就是一个 XML 元素，它可以嵌套 XML 元素。例如，下面的 XML 元素 pcbook 就是一个有效的 XML 文档。

```
<pcbook>
   <book>
     <dc_title> Microsoft Windows 2000 网络基础结构</dc_title>
          <dc_creator>[美]John Shum</dc_creator>
          <dc_subject>计算机技术</dc_subject>
   </book>
</pcbook>
```

XML 文档包含了 XML 标记和文本。XML 文档是一个序列的固定长度字节集，按照某些规则，可以是也可以不是一个文件。例如，某个 XML 文档可以：

- 存储在数据库中。
- 从网络流中读取。
- 由 CGI 程序在内存中创建。
- 是几个混合的不同文件，每个文件彼此镶嵌。
- 永远不以自身单个文件存在。

XML 文本由字符组成，包括字母、数字、标点符号、空格和制表符等。XML 使用 Unicode 字符集，它不仅包括英语和其他西欧字母中的常用字母和符号，而且还包括西里尔语，希伯来语、阿拉伯语以及汉语、日语等会意文字和朝鲜语的音节文字。

让我们先看一个 XML 文档的例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dlib SYSTEM "pcbook.dtd">
<pcbook>
          <dc_title> Microsoft Windows 2000 网络基础结构</dc_title>
          <dc_creator>[美]John Shum</dc_creator>
          <dc_subject>计算机技术</dc_subject>
```

<dc_description>书中全面介绍了如何实现Microsoft Windows 2000网络基础结构。全书共有16个单元、3个附录，内容包括：关于Microsoft Windows 2000网络基础结构的初步介绍、利用DHCP自动分配IP地址、利用DNS实现名称解析、利用WINS实现名称解析、利用公共密钥基础结构配置网络安全性、利用IPSec配置网络安全性、配置远程访问、支持到网络的远程访问、利用IAS扩展远程访问能力、将基于Windows 2000的服务器配置成路由器、针对网络配置Internet访问、配置Web服务器、利用RIS部署Windows 2000 Professional、管理Windows 2000网络、查找和解决Windows 2000网络服务中的问题、在操作系统之间配置网络连接等。</dc_description>

```
          <dc_publisher>北京希望电子出版社 </dc_publisher>
          <dc_contributor>北京希望电子出版社 </dc_contributor>
          <dc_date>2000-12-??</dc_date>
          <dc_type>外文翻译计算机书籍</dc_type>
          <dc_format>电子文档、源数据空间：6G</dc_format>
          <dc_identifier>7-900056-00-9/TP・00</dc_identifier>
```

```
    <dc_source>IDG</dc_source>
    <dc_language>chinese</dc_language>
    <dc_relation>http://www.bhp.com.cn</dc_relation>
    <dc_coverage>国外 计算机书籍 2000 年</dc_coverage>
    <dc_righits>北京希望电子出版社</dc_righits>
</pcbook>
```

从例子中可以看出，与HTML一样，XML也是一个基于文本的标记语言，用标记（一对尖括号）来表示数据。不同的是，XML的标记说明了数据的含义，而不是如何显示它。例如，HTML用标记（< b >...< /b >）表示“粗体”，而XML用标记（<pcbook>...< /pcbook >）表示信息的内容。

正如我们看到的，XML 文档首先由〈?Xml version= “1.0”?>开头，这一行是 XML 声明，它总是第一个出现在 XML 文档之中。接下来的就是这个 XML 文档的内容，它由一个根元素构成，这个根元素的名称是 pcbook，它由开始标记< pcbook >以及结束标记< / pcbook >括起来。开始标记与结束标记之间就是这个元素的内容。由于各个元素内容被各自的元素标记所包含，在 XML 中各种数据的分类查找和处理变得非常容易。图 5-1 是此文档用 Microsoft Internet Explorer 5.0 浏览时的效果。

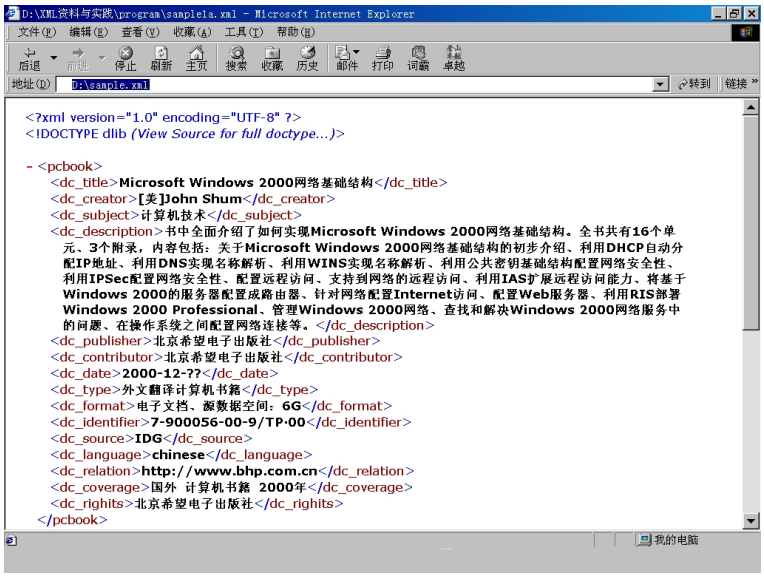

图 5-1

或许读者会觉得这个文档的浏览效果不太好，因为它把标记名称和 XML 声明都显示了出来。其实这是由于我们没有给这个文档引入 XSL 样式表。在后面的章节我们会对样式表进行详细的介绍，这里我们可以先为它加上 XSL 样式表，引用语句：

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

用 Microsoft Internet Explorer 5.0 浏览时的效果如图 5-2。

XML 的标记由一对尖括号组成（< tag >...< /tag >），在它们之间是 XML 数据的一个元素。一个元素可以完全包含在另一个元素之中，这样就可以表示层次结构。XML 与 HTML 的一个

重大区别就是 XML 文档必须是格式良好的，它必须满足几条规则，如标记不能交错嵌套等。如果没有 DTD（Document Type Definition，文档类型定义），文档可以包含任何类型的标记。但如果 XML 文档有相应的 DTD，那么它还需满足语义限制（上面的例子中引用了 pcbook.dtd，pcbook.dtd 的具体内容见第十四章）。DTD 规定了在 XML 文档中可以包含的标记种类和有效布置。只有其结构、数据类型和数据关联等均满足 DTD 要求的 XML 文档，才能被称为有效的 XML 文档。

XML 文档由一个个存储单元组成，这些单元称为实体，包括解析数据（parsed data）和未解析数据（unparsed data）。

解析数据由字符组成，其中一些形成字符数据，另一些形成标记。标记是对文档存储格式和逻辑结构的描述。在形式上，标记有以下各种可能项：注释、引用、字符数据段、起始标记、结束标记、空元素、文档类型声明（DTD）和序言。

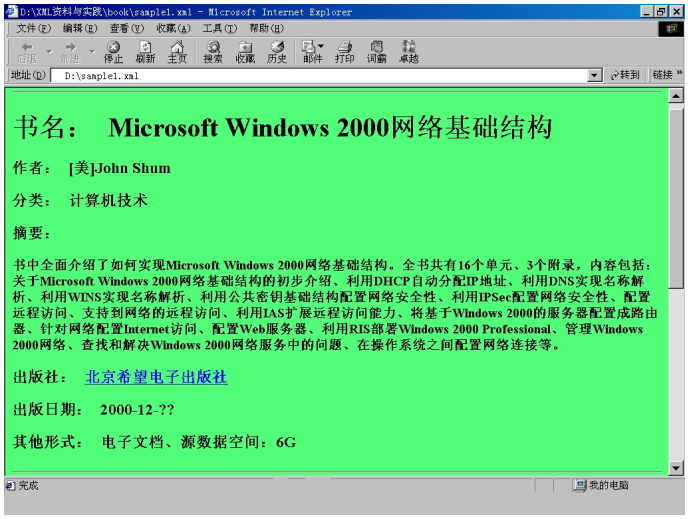

图 5-2

每个 XML 文档都有一个逻辑结构和物理结构。从物理角度来看，文档由实体单元组成，一个实体也可以在其他文档的实体中被引用。一个文档以一个根元素或文档实体来开始。从逻辑上讲，文档由声明（declaration）、元素（element）、注释（comment）、字符引用（character reference）和处理说明（processing instruction）组成。这些组成部分在文档的标记中必须明确规定。

物理结构从另一角度来规范 XML 文档。文档的起始标记和结束标记对数据进行结构化组织，并确定了元素的范围和相互之间的关系。

在 XML 文档中，除标记之外就是字符数据。一般的字符用其本身来表示，但这不适用于 XML 中的保留字符。例如，字符“&”和“<”只能作为标记定界符，或在注释、处理指令和 CDATA 字段中直接使用，其他情况下则需要用字符引用或特定的字符串来表示。

在上面的这个例子中，我们使用了 pcbook 作为根元素的标记名称，这并非是一定的。XML 拥有强大的扩展性，允许用户定义使用自己的标记。标记名称的选用完全是由用户的意愿决定的，我们在这里用 pcbook 来作它的标记名称就是自定的。用户当然也可以使用别的标记名称。

例如，下面的这两个根元素都是合法的。

例 5.1

```
<? Xml version="1.0"? >
<document>
    Hello XML!
</document>
```

例 5.2

```
<? Xml version="1.0" ?>
<welcome>
    Hello XML!
< /welcome>
```

了解 XML 文档以后，下面我们将介绍 XML 声明。

5.2 XML 声明

XML 中规定，每个 XML 文档都必须以 XML 声明开头。其中包括声明 XML 的版本以及所使用的字符集等信息。请注意，在 XML 文档的前面不允许再有任何其他的字符，甚至是空格，也就是说 XML 声明必须是 XML 文档中的第一个内容。下面的例子就包含了一个简单的 XML 文档：

```
<? Xml version="1.0" ?>
<pcbook>
    <book>
        <dc_title> Microsoft Windows 2000 网络基础结构</dc_title>
                <dc_creator>[美]John Shum</dc_creator>
                <dc_subject>计算机技术</dc_subject>
    </book>
</pcbook>
```

文档中第一行<? Xml version=“1.0”？>就是 XML 声明。XML 声明是一个 XML 进程说明，它由“<? Xml”开始，以“？>”结束。其中的 version=“1.0”声明了此 XML 文档所符合 XML 的版本。另外，XML 声明中还可以包含外围设备（standalone）属性。如下例所示：

```
<? Xml version="1.9" standalone="yes" ?>
```

此外，我们还要介绍 XML 声明的一个属性——encoding。Encoding 属性是用于指定 XML 文档的编码方式的。如果 XML 文档中包含中文，必须把 encoding 属性指定为 gb2312，否则浏览文档时只能看到乱码。该属性的具体使用方法如下所示：

```
<? Xml version="1.0" encoding="gb2312" ?>
<document>
    Hello XML!
</document>
```

5.3 注　　释

我们曾在许多编程语言中使用过注释。XML 中同样引入了注释语句。一些刚接触编程的读者可能会觉得编写注释完全是多余劳动，认为注释是不必要的。这是不对的，带有适当注释的 XML 文档不仅使其他人能方便地读懂、交流，更重要的是，它可以使用户自己将来对此文档方便地进行修改。在刚开始编写 XML 文档的时候，由于 XML 文档较小，可能感觉不到注释的重要性。但当用户长期从事 XML 文档编写工作的时候，面对着各式各样的 XML 文档就会感觉到注释的重要性。

XML 的注释与 HTML 类似：都是使用< !--.....-- >进行标记。注释可以在标记之外的任何地方增加。而在解析时，注释的内容将被忽略。为了保持兼容，不能在注释中加入双下横线（--）。XML 的注释以“< ！--”开始，以“ >”结束，处于注释标记“<! --”和“-- >”中的所有语句在处理时都被 XML 处理器所忽略。下面是一个简单的例子：

```
<? Xml version="1.0" ?>
<! -- A example for XML: extensible Markup Language -->
<document>
      Hello XML!
  <!-- please enjoy XML -- >
</document>
```

在这个 XML 文档中有两个注释声明：第一个是在 XML 声明之后，即<! -- A example for XML: extensible Markup Language -- >;而第二个则出现在<document>元素中，即<!-- please enjoy XML -- >。

基本上来说，注释声明的使用是非常自由的，它可以出现在大多数我们想让它出现的地方。但是也有一些地方是不能使用注释声明的。首先，注释声明绝对不能出现在 XML 声明的前面。我们已经在前面说明，XML 声明是 XML 文档的第一项内容，若注释出现在 XML 声明之前则会破坏 XML 文档的结构。例如，下面的这个例子在 XML 中是非法的：

```
<! -- A example for XML: extensible Markup Language -->
<? Xml version="1.0" ?>
<document>
      Hello XML!
  <!-- please enjoy XML -- >
</document>
```

由于此文档的第一项内容不是 XML 声明，而是一个注释句，XML 处理器将拒绝处理这个文档。另外，注释声明也不允许出现在任何一个标记之中，下面的这个示例在 XML 中也是非法的：

```
<? Xml version="1.0" ?>
<document <! -- A example for XML: extensible Markup Language --> >
  Hello XML!
< /document>
```

此例中注释<! -- A example for XML: extensible Markup Language -->夹在标记<document>中，这是非法的。

另外，使用注释还要注意一点，注释声明的内容中不能包含“--”短横串。也就是说，在注释语句中，“--”只能作为开始和关闭标记的部分。例如，下面这句注释就是非法注释：

```
<! --The author -- who is a driver – go abroad! -- >
```

很显然，由于有这个限制，注释语句就不允许嵌套或是重叠使用，下面是一个错误的示例：

```
<? Xml version="1.0" ?>
<! -- A example for XML: extensible Markup Language -->
<document>
      Hello XML!
  <!--
       <note>
          <! -- please enjoy XML -- >
       </note>
        -- >
</document>
```

此例中注释<! -- please enjoy XML -- >嵌套在另一个注释之中，出现错误。

注释语句在 XML 文档中除了可以用来为文档附加说明之外，还可以用来隐藏那些暂时不用而又不是一定要删除的标记及内容。假设我们有一个关于书籍信息的 XML 文档。

```
<? Xml version= "1.0" ?>
<BOOKS>
   <pcbook>
       <book1> VB</ book1>
       < book2> VC++ </ book2>
       < book3> DELPHI </ book3>
    </pcbook>

    <sportsbook>
……
    </sportsbook>

    <poem>
……
    </poem>

</BOOKS>
```

上面这个文档的根元素是< BOOKS >。它下面又包含了一个< pcbook >元素、一个<sportsbook>元素和一个<poem>元素，而这几个元素中又各自包含了一些子元素。现在我们想暂时去掉< pcbook >中<book3>标记及其内容，但将来很可能还会再用到它。但如果把<book3>的标记与其内容从文档中删掉的话，以后再要用到它时又得重新进行 player3 的资料输入，这就很不方便。于是我们就可以用注释标记把它们括起来，那么当 XML 处理器处理到这里的时

候就会把它们当做注释内容而忽略掉。而在将来我们只需要把注释标记去掉就可以恢复使用这个标记了。这样既没有破坏 XML 文档的结构，又达到了目的。示例如下：

```
<? Xml version= "1.0" ?>
<BOOKS>
    <pcbook>
        <book1> VB</ book1>
        < book2> VC++ </ book2>
        <! -- < book3> DELPHI </ book3> -- >
     </pcbook>

     <sportsbook>
……
      </sportsbook>
……
```

在注释掉<book3>后，相应的资料仍然存在于 XML 文档中。只是这些资料会被 XML 处理器所忽略。

使用这种方法的时候，我们要注意被注释掉的部分不能影响剩余文档的结构性。例如，下面这样的注释就破坏了剩余文档的结构：

```
<? Xml version= "1.0" ?>
<BOOKS>
    <pcbook>
        <book1> VB</ book1>
        < book2> VC++ </ book2>
        <! -- < book3> DELPHI </ book3>
     </pcbook>
          -- >

     <sportsbook>
……
      </sportsbook>
……
```

由于注释标记把</football_team>也括了起来，那么，对于 XML 处理器，这个文档就变成下面的内容了。

```
<? Xml version= "1.0" ?>
<BOOKS>
    <pcbook>
        <book1> VB</ book1>
        < book2> VC++ </ book2>

   <sportsbook>
……
     </sportsbook>
……
```

与 HTML 不同，在 XML 中要求每一个标记都有效地被关闭。此时文档中的< pcbook >没有与之对应的</ pcbook >实现元素的关闭。这在 XML 中是不允许的。这个 XML 文档不是一个结构良好的 XML 文档了。

5.4 属性与标记

5.4.1 XML 属性

如果读者使用过 HTML，那么一定知道，在以往的 HTML 中要精确地控制网页的显示方式就需在 HTML 标记中使用大量的属性值。例如我们要显示一幅名为 welcome.gif，200x160 的图像，就要在<img>标记中使用以下的属性：

```
<img src="welcome.gif" width="200" height="160">
```

如果还要在图像周围安排文字，则还要加上 alt 属性；添加链接，则要加上 href 属性。而在 XML 中使用属性就自由得多了，用户可以自己定义所需要的标记属性。在开始标记与空标记中可以有选择地包含属性。属性会代表引入起始标记的数据。我们可以使用一个属性以存储关于该元素的多个数据。属性是由“=”分割开的名称数值对，如下例所示：

```
<? Xml version="1.0" encoding="gb2312" ?>
<document  language="English">
    Hello XML!
 <picture  src="welcome.gif"/>
</document>
```

在这个例子中，开始标记<document>有一个关于语言的属性，其属性名为 language，其值为 English。而在< picture >标记中也有一个属性，其属性名为 src，其值为 welcome.gif。

需要注意的是，所有的属性值都必须用引号括起来。

属性名称的命名规则与标记名称的命名规则一样，必须以字母或者下划线开始，后面的字符同样可以包括字母、数字、下划线、短横和圆点。名称中不能包含空格。而且，属性名称同样对大小写敏感。属性值始终是字符串，示例如下：

```
<book  price="200" />
```

此例中的 price 属性值是字符串 200，而不是数字 200。如果用户要对这个属性值进行算术运算，则需先将它转变成数值。

与属性名称不同，XML 对属性值的内容没有很严格的限制。属性值可包含空格或以数字开头，XML 属性值是由引号来界定的，所以属性值必须用引号括起来，一般使用双引号。如果属性值本身包含了双引号，那么就应该使用单引号；如果属性值同时包含单引号和双引号，那么就要使用实体参考“'”和“"”了。如下例：

```
<exam  time="120&amp;apos；30&amp;quot；"/>
```

还要注意的是，同一个标记不能有相同名称的两个属性，下面的这个例子就是不合规范的：

```
< exam  time="120"  time="30" />
```

当然，我们在使用属性之前首先要定义属性，这个工作将在 DTD 文档类型定义中进行，而有关 DTD 的介绍我们将在后面的章节中介绍。

5.4.2 XML 标记

标记是我们编写 XML 文档必须用到的。说到标记，读者可能会想到 HTML 中曾经使用过的一些标记，例如<font>、<form>、<body>等等。在 HTML 中所使用的标记都是已经定义好的，有各自固定的格式。而在 XML 中，没有一定的标记，我们可以按自己的需要来定义和使用标记。标记把 XML 文件与纯文本文件分开。在一个 XML 文档，标记就是以“<”开始并以“>”结束并且不在注释和 CDATA 节中（CDATA 节的内容我们后面介绍）的内容。我们先看看下面的 XML 文档实例：

```
<? Xml version="1.0 "?>
<FOOTBALL TEAM>
  <TEAM>
    <NAME> 国安 </NAME>
    <PROVINCE> 北京 </ PROVINCE >
  </ TEAM >

  <TEAM>
    <NAME> 实德 </NAME>
    <PROVINCE> 辽宁 </ PROVINCE >
  </ TEAM >

  <TEAM>
    <NAME> 申花 </NAME>
    <PROVINCE> 上海 </ PROVINCE >
  </ TEAM >

  <TEAM>
    <NAME> 力帆 </NAME>
    <PROVINCE> 重庆 </ PROVINCE >
  </ TEAM >
</ FOOTBALL TEAM >
```

在这个 XML 文档中共有三个标记，< TEAM >、<NAME>和< PROVINCE >，其中< TEAM >记录甲 A 足球队信息，<NAME>记录球队名称，<PROVINCE>记录所在省份。这些标记的名称都是可以自己设定的，读者可以用其他自己喜欢的标记名称来代替。

每一个标记都有一个独有的名称，如上面的 NAME、PROVINCE 等。标记名称必须以字母或者划线开头，后面的字符可以是字母、数字、下划线、短横和圆点。

需要注意的是，标记中不能包含空格。

下面的标记名称都是合法的：

```
<pcbook>
<Star-Craft >
<ProviNCE>
<book3.title>
<Michael_Jordan>
<_B2b>
```

非法的标记名称示例如下：

<Price%$56>
<9book>
<*Micro soft>

请注意 XML 文档对大小写是敏感的。也就是说，<name>和<Name>是两个不同的标记。

XML 不同于 HTML。HTML 中有些标记并不要求关闭，例如
、<img>等等；有些语法上要求有关闭标记的即使漏了关闭标记，浏览器也能照常处理，例如<B>。而在 XML 里，每个标记必须保证严格的关闭。关闭标记的名称与打开标记的名称相同，但是其名称前要加上“/”。例如，打开标记是<name>，那么关闭标记必须是</name>。由于 XML 对大小写敏感，在关闭标记时要注意不要使用错误的标记。如用</Name>是不能作为<name>的关闭标记的。下面是前面所介绍的合法标记的正确关闭标记：

</pcbook>
</Star-Craft >
</ProviNCE>
</book3.title>
</Michael_Jordan>
</_B2b>

XML 不像 HTML 那样允许标记没有对应的结束标记，它要求每一个标记都必须关闭，包括空标记。在 XML 中，空标记，即没有关闭标记的标记，需要用一个斜杠和一个三角括号/>来关闭。例如<img/>、
等。也可以用如下方法关闭：

<img> </img>

 </br>

在 HTML 中标记对是允许重叠的。但 XML 不然，它要求所有的标记对要么完全包含在另一对标记对之中，要么独立成对，之间不允许只出现一个开始标记或是结束标记。比如下面的例子就是非法的：

```
<? Xml version="1.0" ?>
<books>
  <b>
  <pcbook>
      book1
</b>
  </pcbook>
</books>
```

这个 XML 文档中标记<b>与标记<pcbook>重叠了，正确的文档如下：

```
<? Xml version="1.0" ?>
<books>
  <b>
    <pcbook>
      book1
 </pcbook>
</b>
</books>
```

5.5 实 体 参 考

从物理角度上说，XML 的文档可以看作是实体的组合。实体声明后就可以在其他地方引用。解析时解析器将用文本或二进制数据来代替该实体。实体声明有两大类：通用实体声明和参数实体声明。通用实体声明在引用时用“&”开始以“;”结束；参数实体声明引用时用“%”开始以“;”结束。

XML 文档中，符号“<”和“>”是被 XML 处理器解释成标记特定部分的。当我们需要在元素内容中出现这些特定符号时，就不能直接输入这些符号。不然 XML 处理器会把它们当作标记的一部分对待而导致出错。所以，在 XML 文档中要用实体参考来分别代表它们。XML 已经定义的实体参考有五个，如表 5-1 所示。

表 5-1

实体参考	字符
&amp;	&
&lt;	<
&gt;	>
&apos;	‘
&quot;	"

在使用它们的时候，注意要用“&”和“；”把它们括起来。例如，“&lt；”就代表符号“<”。在 HTML 中也有类似的用法，如“ ”、“©”等。

假如我们要在 XML 中显示如下内容：if　x<y and y>z　then x=y-z，相应有如下形式的 XML 文档：

```
<? Xml version="1.0" ?>
<document>
  if　x<y and y>z　then x=y-z
</document>
```

这样 XML 处理器在处理到 x<y&y>z 中的“>”和“<”符号时，就会把它当成是一个标记的一部分对待而导致出错。而显然在这里我们使用它是作为大于号和小于号而不是作为标记的一部分。正确的文档书写如下：

```
<? Xml version="1.0" ? >
<document>
    if x &lt;y and y &gt; z then x=y-z
</document>
```

在此 XML 文档中，我们用“>”取代了符号“>”，用“<”取代了符号“<”，XML 处理器在处理文档的时候会自动使用符号“>”来替代“>”，用“<”来替代符号“<”这两个实体参考，从而得到我们预期的显示效果。图 5-3 是此文档用 Microsoft Internet Explorer 5.0 浏览时的效果。

实体参考也可以用在标记内部，如下例所示：

```
 <PARAM name="class"value="The teacher asked,&amp;quot;Who want to answer
                          the boy&amp;apos;s question?&amp;quot;"
```

</PARAM>

在 XML 中除了可以使用这五个已定义的实体参考外，还允许用户定义自己的实体参考。实体参考的定义要在 DTD 中进行。DTD 将在后面的章节中进行详细介绍，目前我们只举简单的例子。以下的 DTD 定义了一个实体：

<!ENTITY name"repacement text">

其中 name 是实体参考的名称，实体参考的名称可以由字母、数字和下划线组成，但是不允许使用空格和其他任何标点符号。而且，实体参考的名称同样是大小写敏感的。“&Editor;”与“&editor;”是两个不同的实体参考。而 replacement text 则为实体参考将要替代的文本。例如：

<ENTITY motherland"The People's Republic of China">

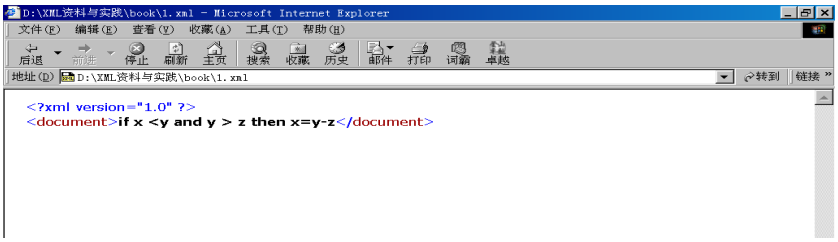

图 5-3

当我们在 XML 文档中使用“&motherland;”时，XML 处理器就会用 The People's Republic of China 来代替它。

XML 中实体参考的引入是非常有用的。它可以使 XML 文档管理起来更加方便。比如说假设我们有一些关于书籍销售的 XML 文档，其中包含一个价格标记：

<price>55.95</price>

它可能出现在各个文档的不同位置。当价格发生变化的时候我们要相应地改变<price>元素的值。但如果一个一个文档地去修改，不仅费时费力，还容易出现错漏。此时我们可以定义一个实体参考 price_value 来代替这个值。然后，在各个 XML 文档中使用以下的代码来表现这个产品的价格：

<price>& price_value; </price>

当产品价格发生变化后，我们只需要在这个实体参考定义的地方把它的值改过来，则所有引用这个实体参考的文本都会相应地发生变化，而不必把所有的文档都修改一次。实体参考带给我们的好处读者将会在自己的编程实践中体会得更深刻。

5.6 CDATA 节

通过前面的介绍，我们已经知道在 XML 中符号“>”、“<”、“&”等是被当作标记的特定部分来处理的。当我们要输入这样的文本的时候，不能直接输入。如果把这些符号直接输入，XML 处理器会把它们当成标记、实体参考或属性的一部分来处理而导致出错。然而，我们可以分别用实体参考“>”、“<”、“&”来代替它们。

例如，我们想用 XML 来显示下面的内容：

```
<? Xml version="1.0" ? >
<document>
   This is a example for XML document.
</document>
```

那么就应该编写如下 XML 文档：

```
<? Xml version = "1.0" ?>
<xmldoc>
&lt; ? xml version=&quot; 1.0&quot; ? &gt;
&lt; document&gt;
This is a example for XML document.
&lt; /document&gt;
</xmldoc>
```

在工作量较小的时候我们很轻易就能完成这些工作。但是我们有一段包含大量这些符号的文本需要输入的时候，我们就不得不花大力气来把全部的“<”符号用“<”来代替，把全部的“>”符号用“>”来代替，等等。这样的方法显然是非常费劲的，也是不实际的。

在这些时候，就应该使用 CDATA 节了。一般地，在一对三角括号“<>”中的内容都是标记，不在这些三角括号中的内容都是字符数据。然而，在 CDATA 节中的所有内容，包括“<、“>”、“”“”、“‘”、&等，全部被当做纯字符数据对待，而不再认为它们是标记、实体对考或属性的一部分。XML 处理器不会对它们进行解释。CDATA 节以“<![CDATA[”开始，以“]]>”关闭。

如果我们要编写一个包含以下文本的 XML 文档：

```
main()
{
   int a[10];
   int I;
   for (I=0; I <10; I++)
   scanf ("%d",&a[I]);
   printf (" \n");
for (I=0; I <10; i++)
print("%d", a[I]);
}
```

使用 CDATA 节可以大大减轻工作量，只要用“< ! [CDATA[”和“]]>”把这段文本括起来就可以了，该 XML 文档编写如下：

```
<? Xml version="1.0" ? >
<C_code>
   <! [CDATA [
main()
 {
    int a [10 ];
    int I;
    for (I=0; I < 10; i++)
    scanf ( "%d", &a [ i ]);
    printf ("\n");
    for (I=0; I < 10; i++)
```

```
    printf ("%d", a [ i ];
      }
    ] ] >
</C_code>
```

由于使用了 CDATA 节，工作变得简单方便，而且非常直观，不容易出错。作为比较，我们把前面 5.5 节的例子改用 CDATA 节来实现，写成下面的形式：

```
< ? xml version="1.0" ? >
<document>
   <! [CDATA[
        if   x<y and y>z   then x=y-z
    ]]>
  </document>
```

用 Microsoft Internet Explorer 5.0 浏览此文档的效果如图 5-4。

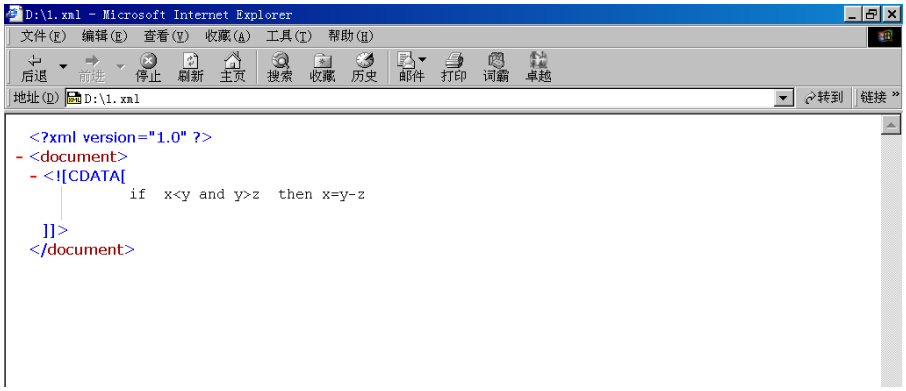

图 5-4

需要注意的是，在 CDATA 节中唯一不允许出现的文本是 CDATA 节的关闭标记“]]>”，由于 CDATA 节中不能包含“]]>”，故 CDATA 节是嵌套的。当要编写出现“]]>”的文本的时候，就只有使用>了。如下例：

```
<? Xml version="1.0" ? >
<document>
  &lt; ! [CDATA [   ]] &gt;
< / document >
```

5.7　文档类型定义 DTD

本节我们简单介绍一下 DTD（Document Type Declaration 文档类型定义，简称 DTD）的概念。DTD 在 XML 中占有非常重要的地位。这里只是先对它作一个大概介绍，在后面的章节中我们将会详细讨论 DTD 的应用。

规范的 XML 文档都必须是格式良好（well-formed）的。但可以规定其数据结构，也可以不被规定。如果被规定，那么该文档是有效的，否则是无效的。文档类型声明是最初应用的一种定义方法。DTD 一般包括标记声明或参数实体引用，有时还包括外部实体的 ID。标记声明

可以是元素类型声明、属性表声明、实体声明或符号声明。

XML 不同于 HTML，XML 并没有自己固定标准的标记以及属性。而用户要使用自己的标记、属性或者是实体参考等就要自行定义它们。而这些定义的工作要在 DTD 文档类型定义中进行。DTD 为 XML 文档定义了该文档中应该包含的或者是可以包含的元素、标记、属性和实体的一个清单以及它们之间的相互关系。也就是说 DTD 为 XML 文档规定了一套专用的规则。举个简单的例子来说，在一个有关甲 A 联赛的 XML 文档中，由 DTD 可以规定在此文档中一个<FOOTBALL TEAM>包含一个或者几个<TEAM>，而在<TEAM>中则必须只有一个<NAME>、多个<MEMBER>。另外还可能包含一个<MATCH TIME>等等。

DTD 文档类型定义位于 XML 声明与文档体之间，XML 声明与 DTD 合称为 XML 文档的序言。DTD 由“<！DOCTYPE name [”开始，其中 name 是文档名称，DTD 的结束标记是“]>”。在 DTD 中，可以定义元素类型、属性和实体参考等。

下面是一个简单例子：

```
<? Xml version= "1.0" ? >
<! DOCTYPE   LEAGUE [
<! ELEMENT TEAM (MEMBER*)>
<! ELEMENT   MEMBER (#PCDATA)>
]>

< TEAM >
  < MEMBER >Li Tie </ MEMBER >
  < MEMBER > Yang Chen </ MEMBER >
  <MEMBER > Zhang En Hua</ MEMBER >
</ TEAM >
```

在第一行的 XML 声明（<? Xml version=”1.0” ? >）之后就是 DTD 定义。其基本结构如下所示：

```
<! DOCTYPE   LEAGUE [
]>
```

这里定义了 LEAGUE 为文档名称，同时定义了根元素的标记名称为 TEAM。在这个 DTD 中部有两个元素类型定义。元素类型定义由“<！ELEMENT”开始，以“>”结束。本例中第一个元素类型定义是<! ELEMENT TEAM (MEMBER*)>。它定义了元素 TEAM 中可以包含多个 MEMBER 元素。MEMBER 后面的“*”号表示 MEMBER 元素可以在它的父元素中多次出现。而第二个元素类型定义是<! ELEMENT MEMBER（# PCDATA）>。它定义了 MEMBER 元素包含的内容是#PCDATA（已分析的字符数据）。元素类型定义在 DTD 中的具体使用将在后面的章节详细介绍。

下面我们再举几个例子来简单说明一下属性和实体参考的定义，至于详细的使用方法也将在以后详细介绍。

```
<? Xml version= "1.0" ? >
<! DOCTYPE   LEAGUE [
<! ELEMENT TEAM (MEMBER*)>
<! ELEMENT   MEMBER (#PCDATA)>
<! ATTLIST MEMBER CODE CDATA #REQUIRED>
```

```
]>

< TEAM >
  < MEMBER   CODE="5">Li Tie </ MEMBER >
  < MEMBER   CODE="20"> Yang Chen </ MEMBER >
  <MEMBER   CODE="2"> Zhang En Hua</ MEMBER >
</ TEAM >
```

在这个 XML 文档的 DTD 中除了两个元素类型定义外，还有一个属性定义<! ATTLIST MEMBER CODE CDATA #REQUIRED>。属性声明以“<! ATTLIST”开始，以“>”结束。这里声明了元素 MEMBER 的属性 CODE 的类型是 CDATA，使用#REQUIRED 来规定缺省值。

下面是一个定义实体参考的简单例子：

```
<? Xml version= "1.0" ? >
<! DOCTYPE   LEAGUE [
 <! ENTITY LEADER  “中国足协” >
 <! ELEMENT TEAM (MEMBER*)>
 <! ELEMENT   MEMBER (#PCDATA)>

]>

< TEAM >
  < MEMBER >Li Tie </ MEMBER >
  < MEMBER> Yang Chen </ MEMBER >
  <MEMBER > Zhang En Hua</ MEMBER >
</ TEAM >
```

实体参考的定义由“<! ENTITY”开始且以“>”结束。此例中的<! ENTITY LEADER “中国足协”>就是一个实体参考的定义。其中，LEADER 是实体名称，中国足协 r 是用来替代实体参考 LEADER 的内容。

5.8 XML Schema

XML 中的 DTD 明显带有 SGML 的烙印。关键的一点是，DTD 本身并不是 XML，而且它只提供了非常有限的数据类型，也不支持名字空间机制，另外 DTD 中的内容模型是不开放的，它不能随意扩充内容。重新制定符合 XML 的大纲机制已经是大势所趋。W3C XML Activity Page 声明：“尽管 XML1.0 提供了一种机制，文档类型定义（DTD）给标记的使用加了限制，但是对 XML 文档的自动处理需要更严格更全面的工具。需要主要体现在对应用软件各部分的结合、文档结构、属性和数据类型等的约束。W3C XML Schema 工作组正忙于定义 XML 文档的结构、内容和语义。”

1999 年 5 月 6 日，“XML Schema 工作草案”被接受为工作草案（Working Draft），同年 11 月 5 日，该草案被修改，但还是工作草案，它之所以还没有被确定为最终标准，是出于慎重考虑。不过经过多次的讨论和完善，目前的 XML Schema 工作草案已基本成熟。

- XML Schema 为一类文档建立了一个模式，规范了文档中的标签和文本可能的组合形式。它不仅包括了 DTD 能实现的所有功能，而且它本身就是规范的 XML 文档。XML

Schema 提供了一系列新特色，大大弥补了 DTD 的不足。

- 丰富的数据类型。XML Schema 支持的数据类型包括数字型、布尔型、整型、日期时间、URI、十进制数等。而且它还支持由这些简单的类型生成更复杂的类型。
- 可以由用户自定义数据类型。
- 支持属性分组。属性的应用范围是多种多样的，有的是针对所有元素，有的则专门针对图形元素。
- 原型可以更新。DTD 定义的内容模式是封闭的，而 XML Schema 定义的内容模式是开放的，可以随时更新。
- 支持名字空间。

微软 IE5 支持 XML Schema，这项预先展示的技术是建立在递交给 W3C 的 XML-Data 草案的基础上的。XML Schema 可被认为是 XML-Data 草案的子集，它符合文档内容描述（DCD）提议的特点。

IE5 中的 XML 解析器能够根据文档类型定义(DTD)或 XML Schema 解析 XML 文档。XML Schema 是用来声明内容模式的基于 XML 的语法。它有 DTD 所有的功能，并且还有其他的功能如数据类型定义。

下面让我们看一看如何建立 XML Schema。

我们先以下面的 XML 文档为例来看看每个节点的 schema 声明：

```
<class xmlns="x-schema:classSchema.xml">
    <student studentID="13429">
        <name>Jane Smith</name>
        <GPA>3.8</GPA>
    </student>
</class>
```

读者也许已经注意到在上面文档中默认的名字空间是“x-schema:classSchema.xml“。这告诉解析器根据 URL（classSchema.xml）上的 schema（x-schema）来解析整个文档。

下面是上面文档的完整的 schema。注意 schema 的根元素中的名字空间声明。第一个（xmlns=“urn:schemas-microsoft-com:xml-data”）表明这个 XML 文档是一个 XML Schema。第二个（xmlns:dt=“urn:schemas-microsoft-com:datatypes”）允许 schema 处理者在“ElementType”和“AttributeType”声明中的“type”属性前加“dt”前缀来说明元素的类型和内容的特征。

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
        xmlns:dt="urn:schemas-microsoft-com:datatypes">

    <AttributeType name="studentID" dt:type="string" required="yes"/>
    <ElementType name="name"content="textOnly">
    <ElementType name="GPA" content="textOnly" dt:type="float"/>
    <ElementType name="student" content="mixed">
        <attribute type="studentID"/>
        <element type="name"/>
        <element type="GPA"/>
    </ElementType>
    <ElementType name="class" content="eltOnly">
        <element type="student"/>
```

```
    </ElementType>
</Schema>
```

schema 用“Schema”元素开头，“Schema”元素包括 schema 名字空间的声明，在本例中还包括数据类型名字空间的声明。Schema 的内容以“AttributeType”和“ElementType”的声明开头。

```
<AttributeType name="studentID"dt:type="string" required="yes"/>
<ElementType name="name" content="extOnly">
<ElementType name="GPA" content="textOnly" dt:type="float"/>
```

这些声明接下来的是刚声明过元素的父元素的“ElementType”声明：

```
<ElementType name="student" content="mixed">
    <attribute type="studentID"/>
    <element type="name"/>
    <element type="GPA"/>
</ElementType>
```

这个过程继续下去，直到所有元素都已经声明了。

不同于 DTD，XML Schema 允许有一个开放的内容模式，可以进行定义数据类型、使用默认值等等操作而不必限定内容。

在下面的 schema 中，“GPA”元素的类型被定义并有一个默认值，但在“student”元素中没有其他节点被声明。

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
        xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <AttributeType name="scale" default="4.0"/>
    <ElementType name="GPA" content="textOnly" dt:type="float">
        <attribute type="scale"/>
    </ElementType>
    <AttributeType name="studentID"/>
    <ElementType name="student" content="eltOnly" model="open" order="many">
        <attribute type="studentID"/>
        <element type="GPA"/>
    </ElementType>
</Schema>
```

上面的 schema 允许用户只确认自己所关心的区域。这使用户对文档有更多的控制，并允许用户使用 schema 提供的一些特性而不必严格确认。

说明：

“ElementType”和“AttributeType”声明必须放在“attribute”和“element”内容声明之前。例如，在上面的 schema 中，“GPA”元素的“ElementType”声明必须放在“student”元素的“ElementType”声明之前。

“order”属性的默认值是建立在“content”属性的值上的。当 content 值为“eltOnly”时，order 默认值是“seq”。当 content 值为“mixed”时，order 默认值是“many”。

5.9 名 字 空 间

我们已经知道，在 XML 中允许使用自己的标记来编写 XML 文档。但是可以想象在 Internet 上，不同的人编写的 XML 文档很可能使用同样的标记名称来标识各自不同的元素。当我们需要把多个 XML 文档合并成一个时，就很可能发生标记名的冲突。另外，把多个 XML 文件合并为一个时，也很可能出现冲突。名字空间（Namespaces）就是为了解决这个问题的。对 XML 名字空间严格的定义是：名字空间是用 URI 加以区别的、在 XML 文件的元素和属性中出现的所有名称的集合。有了名字空间，用户就可以保证在他的文件中使用的名称是独一无二的。 在没有名字空间的 XML1.0 文件中，元素和属性中出现的名称无异于一族没有结构的字符。我们称它们为本地名称（local name）。本地名称在网络上是极不合适的。可以想象，网上会有成千上万的人使用同一个名称，而它们却代表了不同的意义。

名字空间通过 URI 区别同名的标识。我们已经完全可以相信不会出现冲突了，因为 URI 是独立的。

名字空间的声明要用到前缀 xmlns。声明的名字空间位于指定的 URI，同样的，它也有个名字，我们称这个名字为：名字空间名。名字空间名必须独有而一致。有它修饰过的元素就不要认为它由指定 URI 处的名字空间约束。

下面是名字空间冲突的实例。例如有一个关于部门员工的 XML 文档，用标记<name>来表示部门员工的名字：

```
<? Xml version="1.0" ?>
<employee>
  <name> tom </anme>
</employee>
```

而在另一个有关财务的 XML 文档中用标记<name>来表示财务报表的表头。

```
<? Xml version="1.0" ?>
<list>
   <name> head </name>
</list>
```

当需要合并这两个 XML 文档时，就会出现标记名称的冲突。如果简单地把它们合并，则会把员工姓名与财务报表的表头混淆起来。如何解决这个问题呢，把其中一个 XML 文档中冲突的标记改成没有冲突的档记当然是一种办法，而更好的解决方法是引入名字空间。

名字空间的使用提供了一个简单的方法来解决这种可能存在的冲突。具体来说，名字空间的作用是指明元素是属于那个文档的。

名字空间声明和属性一样，都是存在于元素的开始标记之中。我们也可以把它看成一个包含前缀 xmlns 的属性。该属性名就是名字空间的名称，而属性的值就是该名字空间对应的 URI。这个 URI 就是使用名字空间的元素所属的文档。

下面是一个简单名字空间的声明：

```
<x xmlns: edi="http: // ecommerce.org/schema">
   <! – the "edi" prefix is bound to http: //ecommerce.org/schema
   for the "x" element and contents - - >
</x>
```

x 是元素的名称。xmlns 前缀指出一个名字空间在这一元素中被用到。HTTP 的地址表明了名字空间 xchema 的位置。Edi 是这个名字空间的名称。

我们看看该名字空间的具体应用方法：

```
<? Xml version="1.0" ?>
<x xmlns: edi="http: //ecommerce.org/schema">
  <! - - the "price" element's namespace is http: //ecommerce.org/schema - ->
  <dei: price units="Euro"> 32.18 </edi: price>
</x>
```

在这个例子中，我们首先声明了名字空间 edi。该名字空间对应的约束内容在 URI“http：//ecommerce.org/schema”中。而 price 标记名称是受名字空间 edi 所限制的，即该元素的定义存在于“http：//ecommerce.org/schema”中。注意我们把 price 的标记写为<edi: price>。这样，即使别人在这个文档中使用了<price>标记来表示其他的元素，也不会与之混淆。

下面是名字空间在属性名中的应用：

```
<? Xml version="1.0" ?>
<x xmlns: eid="http: //ecommerce.org/schema'>
  <! - - the "taxClass" attribute's namespace is http: //ecommerce.org/schema - ->
  <lineItem edi: taxClass="exempt"> Baby food </lineItem>
  <x>
```

此例中标记<lineItem>的 taxClass 属性名受名字空间 edi 的限制。

多重的名字空间前缀的声明类似一个元素的多个属性，如下例：

```
<?  xml version="1.0" ?>
<! - - both namespace prefixed are available throughout - ->
<bk: book xmlns: bk="urn: loc.gov: books"
  xmlns: isbn="urn:ISBN: 0-395-36341-6">
  <bk: title> Cheaper by the Dozen </bk:title>
<isbn: number> 1568491379 </isbn: number?
</bk: book>
```

名字空间不仅约束那些指明了名字空间的元素，如果不特殊说明，元素内的子元素也要受它的约束，除非它们受到另一个名字空间定义的限制。一个缺省的名字空间只修饰声明了这个名字空间的那个元素以及该元素的子元素。如果一个缺省名字空间声明的 URI 参数值是空的，那么这个元素就不在任何一个名字空间中。

注意，属性是不能直接用缺省名字空间的。请看下面的例子：

```
<? Xml version="1.0" ?>
<! – initially, the default namespace is "books" →
<book xmlns="urn: loc.gov:books"
  xmlns: isbn="urn: ISBN: 0-395-36341-6">
  <TITLE> cheaper by the Dozen </title>
  <isbn: numbef> 1568491379 </isbn: number>
<notes>
  <! –make HTML the default namespace for some commentary →
  <p xmlns="urn: w3-org-ns: HTML">
   This is a <I> funny </I> book!
  </p>
 </notes>
```

```
</book>
```

此例中有两个缺省名字空间。这是第三行的 book xmlns='urn: loc.gov: books'语名所说明的。如果没有在之前加上名字空间，那么该元素的名字空间是 urn: loc.gov: books。而有特别说明的元素 number 的名字空间是 isbn。

5.10 正规有效的 XML 文档

一般地，一个 XML 文档包含一个序言。虽然我们可以根据自己的需要构造出许多标记，但是 XML 文档也必须遵循结构性规则。如果一个 XML 文档是非格式良好的，非结构性的，那么 XML 处理器对这个 XML 文档的解释就会出错。在 XML 中，一个 XML 文档应该是有效而且是格式良好的。

5.10.1 格式良好的 XML 文档

什么是格式良好的 XML 文档呢？XML 虽然没有固定的标记规范，但它还是有一定的语法规范的。如果一个 XML 文档包含一个或多个元素，各元素都有正确嵌套，并且正确地使用了属性和实体参考，符合 XML 的基本语法规范，那么就认为这个 XML 文档是格式良好的。要保证编写出的 XML 文档是格式良好的，我们就必须遵循前面几节所介绍的 XML 的基本规则。这里有必要再重申一下。

1. XML 文档必须以一个 XML 声明开始

在一个 XML 文档中，XML 声明绝对是文档的第一项内容。XML 处理器处理 XML 文档时首先读取 XML 声明来确定一些初始信息。所以必须保证在我们的 XML 文档中的第一项内容是 XML 声明。

下面这个例子由于错误地使注释出现在 XML 声明之前，而成为非法的：

```
<! --XML document -- >
< ? xml version="1.0" ? >
<document>
  Hello XML!
</document>
```

2. 保证每一个开始标记有对应的结束标记

XML 文档要求每个标记都严格关闭。XML 文档中若是有标记没被关闭，或是结束标记与开始标记不匹配都会导致 XML 处理器解释 XML 文档时出错。请注意，XML 文档对大小写敏感，也就是说，<Document>与</document>不是一对匹配的标记。而这往往是造成开始标记与结束标记不匹配的原因。另外在使用注释的时候也要注意不能破坏 XML 文档的结构性。

XML 文档中的空标记是不包含数据的。但是它也同样要求严格关闭。我们在空标记后面用“/>”来保证其关闭。

```
<? Xml version="1.0" ?>
<document>
  <p> On some fields  <! - - <p> - - >
  <p> This has clearly not happened. </p>
```

```
</document>
```

由于注释标记把一个结束标记注释掉了，破坏了 XML 文档的结构性。

下面的例子错误地使用了</title>标记来关闭<Title>，也破坏了文档的结构性。

```
<? Xml version="1.0" ?>
<document>
  <Title>book1 </title>
  <price> 55.99 </price>
</document>
```

3. 各元素之间正确地嵌套

XML 文档不同于 HTML，它是不允许各标记互相重叠的。所以元素间只能互相嵌套。也就是说，每个元素的标记对都要成对地被另一对标记对包含或不包含，不能有单个开始标记或是单个结束标记出现在另一个标记对中。

```
<? Xml version="1.0" ? >
<document>
  <p> <b> Welcome </P> </b>
</document>
```

上例的元素 p 与 b 的标记对没有正确地嵌套，而是互相重叠。从而破坏了此文档的结构性。

4. 正确使用实体参考

由于 XML 处理器认为符号“<”是一个标记的开始，符号“&”是一个实体参考的开始，所以我们在元素内容中要使用这类符号时，就要使用实体参考，以免 XML 处理器解释文档时出错。在使用实体参考时，要用“&”与“；”把实体参考括起来。这在使用自定义的实体参考时也要注意。

下面是一个错误的示例。user’s 中有一个单引号没有用实体参考代替：

```
<? Xml version="1.0" ? >
<document>
  Please input the user’s password.
</document>
```

5.10.2 有效的 XML 文档

如果一个 XML 文档与一个文档类型定义（DTD）相关联，而该 XML 文档符合该 DTD 的各种规则，那么，我们称这个 XML 文档是有效的。注意 DTD 对于 XML 文档来说并不是必须的。但 XML 文档要由 DTD 来保证其有效性。所以要保证 XML 文档的有效性就必须在 XML 文档中引入 DTD。而且有 DTD 的 XML 文档会使 XML 文档会人读起来易懂，也令人容易地找出文档中的错误。

下面是一个正式的 XML 文档：

```
<? Xml version="1.0" >
<! DOCTYPE DOCUMENT [
<! ELEMENT CUSTOMER (NAME, DATE, ORDERS)>
<! ELEMENT NAME (LASTNAME, FIRSTNAME)>
<!   ELEMENT LASTNAME (#PCDATA)>
```

```
(! ELEMENT FIRSTNAME (#PCDATA)>
<! ELEMENT DATE (#PCDATA)>
<ELEMENT ORDERS (ITEM)*)
<!ELEMENT ITEM(PRODUCT, NUMBER, PRICE)>
<! ELEMENT PRODUCT (#PCDATA)>
<! ELEMENT NUMBER(#PCDATA)>
<!ELEMENT PRICE(#PCDATA)>
]>
<DOCUMENT>
   <CUSTOMER>
    <NAME>
<LASTNAME> Edwards </LASTNAME>
<FIRSTNAME> Britta <FIRSTNAME>
<NAME>
<DATE> April 17, 1998 </DATE>
<ORDERS>
   <ITEM>
      <PRODUCT>Cucumber </PRODUCT>
       <NUMBER> 5 </NUMBER>
       <PRICE> $ 1.25 </PRICE>
   </ITEM>
   <ITEM>
       <PRODUCT? Lettuce </PRODUCT>
       <NUMBER> 2 </NUMBER>
       <PRICE> $ .98 </PRICE>
   <ITEM>
 </ORDERS>
<CUSTOMER>
    <CUSTOMER>
   <NAME>
<LASTNAME> Thompson </LASTNAME>
<FIRSTNAME> Phoebe </FIRSTNAME>
 </NAME>
 <DATE> May 27,1998 </DATE>
<ORDERS>
   <ITEM>
    <PRODUCT> Banana </PRODUCT>
    <NUMBER> 12 </NUMBER>
   <PRICE> $2.95 </PRICE>
 </ITEM>
<ITEM>
   <PRODUCT> Apple </PRODUCT>
   <NUMBER> 6 </NUMBER>
   <PRICE> $1.50 </PRICE>
      </ITEM>
   </ORDERS>
</DOCUMENT>
```

这个 XML 文档清楚地记录了一些顾客的情况，XML 强大的可扩展性使得数据的存储和分类更加容易。上面的这个 XML 文档既是有效的，又是格式良好的。第一行的<? Xml version=”1.0” ? >是 XML 声明。从第二行之后是以“< ! DOCTYPE DOCUMENT [”开始的文档类型定义。定义了<DOCUMENT>元素中可以有任意个<CUSTOMER>元素；每个<CUSTOMER>元素中又包含一个<NAME>、<DATE>和<ORDERS>元素，而<NAME>中包含一个<FIRSTNAME>和一个<LASTNAME>元素；<ORDERS>中包含任意个<ITEM>元素；<ITEM>中包含一个<PROCUCT>元素、一个<NUMBER>元素和一个<PRICE>元素；最后 DTD 以“]>”结束。

第十四行的<DOCUMENT>标记就是这个 XML 文档的根元素的开始标记，注意它在文档的末尾由</DOCUMENT>标记关闭。而在根元素<DOCUMENT>中则按 DTD 的定义正确地包含了各个子元素。

一个没有关联 DTD 的 XML 文档虽然可以是格式良好的，但却不是有效的：

```
<? Xml version="1.0" ?>
<document>
  Welcome to XMLworld.
</document>
```

当然，一个关联了 DTD 的 XML 文档若是不符合 DTD 的所有定义，那么它也不是有效的。

5.11 数 据 岛

本节我们将介绍数据岛的概念。数据岛是指存在于 HTML 页面中的 XML 代码。数据岛允许用户在 HTML 页面中集成 XML、对 XML 编写脚本，而不需要像 HTML 那样通过脚本或<OBJECT>标签来读取 XML。几乎所有能够存在于一个结构完整的 XML 文档中的内容都能存在于一个数据岛中。包括处理指示、DOCTYPE 声明和内部子集等（注意，编码串不能放在数据岛中。）。

下面就是一个简单的数据岛：

```
<XML ID ="XMLID">
  < customer >
   < name >Herbert Hanley </name>
   <custid> 81422 </custid>
  </customer>
</XML>
```

数据岛由标记<XML>开始。在开始标记中要有一个 ID 属性。最后以</XML>结束。元素中的内容就是 XML 代码。上面这个例子是内嵌的数据岛，也就是说，XML 代码包含在<XML>标记中。然而，我们也可以通过在 XML 标记中指定 SRC 属性来引用外部的 XML 代码：

```
<XML ID "XMLID" SRC= "customer.xml"> </XML>
```

这个数据岛的 XML 代码放在文件 customer.xml 中，通过 SRC 属性来指定。

访问数据岛类似于访问文档对象。XML 文档对象是指一个拥有属性和方法的对象，用户可以利用这些属性和方法访问和处理 XML 文档。当一个 XML 数据岛被读取和解析时，就会

创建一个 XML 文档对象。下面是一个带有数据岛的 HTML 页面，数据岛在<XML>元素中：

```
<HTML>
  <HEAD>
   <TITLE> HTML   with XML Data Island </TITLE>
  </HEAD>
  <BODY>
   <p> Within this document is an XML data island. </P>
   <XML ID="resort XML">
 <resorts>
  <resort> Cal;inda Cabo Baja </resort>
  <resort> Na Balam Resort </resort>
 </resorts>
</XML>
</BODY>
</HTML>
```

我们能通过 ID 属性访问数据岛，resortXML 成为文档对象的名称。我们能利用这个对象的方法和属性访问它的根节点和子节点。在上面的例子中，根节点是<resorts>，子节点是<resort>。下面列出了一些属性和方法，我们可以使用这些属性和方法来访问 XML 文档的节点。

- XMLDocument:返回对 XML 文档对象模式的引用。
- documentElement:返回 XML 文档的根节点。
- childNodes:返回节点的子节点目录。
- item：通过索引访问目录中的个别节点，索引值从 0 开始，所以 item（0）返回第 1 个节点。
- text:返回节点的内容。

下面的代码访问第 2 个子节点（<resort>），并返回它的内容（“Na Balam Resort”）：

```
resortXML.XMLDocument.documentElement.childNodes.item(1).text
```

5.12　XML 的相关标准

XML 尚在发展之中，相关标准和术语很多：

（1）W3C 建议（Recommendations）：W3C 产生的规范的最终形式。之所以称为“建议”是因为它并不强加于任何人，但已不再进一步讨论和复查了。

- SAX（Simple API for XML，XML 简单应用程序接口）：这实际上是在 XML-DEV 邮件列表上协作产生的，并不是 W3C 的标准，但事实上已和 W3C 建议有着同等地位。这一 API 是事件驱动的，又称“顺序访问”协议。每当它看到一个新的 XML 标记（或遇到一个错误，或想通知用户什么事情时）就用一个 SAX 解析器注册句柄，激活回调方法。
- DOM（Document Object Model，文档对象模型）：DOM 将一个 XML 文档转换成程序中的一个对象集合，然后可以任意处理对象模型。这一机制也称为“随机访问”协议，因为用户可以在任何时间访问数据的任何一部分，然后修改、删除或插入新数据。

- DTD（Document Type Definition，文档类型定义）：DTD 规范实际是 XML 规范的一部分，同时又是可选的（可以写一个没有 DTD 的 XML 文档）。另外还有一个更加灵活的 Schema 提案可以替代它。DTD 规定在 XML 文档中可以包含的标记种类和有效布置，因此可保证用户不会创建一个无效的 XML 结构或者保证用户看到的 XML 结构是有效的。然而，对于一个复杂的文档来说创建一个排除所有无效组合并允许所有有效组合的 DTD 是很困难的。DTD 可以作为 Prolog 的一部分放在文档前面，也可以作为一个独立实体存在，或者分散在文档 prolog 和一个或几个实体中。
- RDF（Resource Description Framework，资源描述框架）：RDF 是定义关于数据的数据的标准。例如，和 XHTML 规范或 HTML 标记一起使用，RDF 可用于描述页面的内容。举例来说，如果用户的浏览器将用户的个人信息存为名字、Email 地址，一个 RDF 描述就可以将数据传输给需要名字和 Email 地址的应用。要进一步了解 RDF，可查看http://www.w3.org/TR/PR-rdf-syntax。
- XSLT（XSL Transformations，XSL 转换）：XSLT 是作为 XSL 的一部分使用的，用于将 XML 文档转换为其它 XML 文档。在 XSLT 之外，XSL 还包括一个用于定义格式化的 XML 词汇表，它用 XSLT 来描述文档如何用格式化词汇表转换为另一个文档。目前 XSLT 版本为 1.0，可查看 http:/ /www.w3.org/TR/1999/REC-xslt-19991116。

（2）Xpath（XML Path Language，XML 路径语言）：Xpath 是寻址一个 XML 文档内的组成部分的一种语言，被设计成可为 XSLT 和 XPointer 共同使用。Xpath 使用一个简练的、非 XML 的句法以方便在 URL 和 XML 属性之中的使用。Xpath 在一个 XML 文档的抽象逻辑结构上操作，其名字来源于它用路径记法表达 XML 文档中的层次结构。Xpath 的设计使它具有一个子集可用于匹配（测试一个节点是否满足一个模式），这一使用在 XSLT 中有介绍。目前 XPath 版本为 1.0，可查看 http://www.w3.org/TR/1999/REC-xpath-19991116。2．W3C Proposed Recommendations（W3C 提出的建议）。

W3C “proposed recommendation”是一个非常接近于结束的 W3C recommendation 提案。它仍可复查，但已有许多人作了大量工作，因此这类标准一般不会再有多大改动。

- RDF Schema（大纲）：RDF Schema 规定了描述如何翻译一个 RDF 中的声明所需的一致性规范和附加信息。要进一步了解 RDF Schema，可查看 http://www.w3.org/TR/PR-rdf-schema.。
- XHTML（Extensible HyperText Markup Language，扩展超文本标记语言）：XHTML 规范是使 XML 文档看起来和操作类似于 HTML 文档的一种方式。既然 XML 可以包含任何用户愿意定义的标记，为什么不定义一套看起来象 HTML 的标记呢？这一规范的结果就是一个文档，可以在浏览器中显示，也可以作为 XML 数据处理。数据可能不是“纯粹的”XML，但也比标准的 HTML 容易处理的多。例如，一个格式良好的 XML 文档中每一个标记都必须有一个对应的结束标记，否则必须用（</ >）结束。所以我们可以看到（< p >...< /p >）或（< p/ >），但决不会只看到< p >。而 HTML 中的一个（< dt >）标记可以由（< /dt >）、另一个（< dt >）、（< dl >）（< /dl >）结束。XHTML 规范是将 HTML4.0 再形成为 XML。XHTML 目前版本为 1.0，可查看 http://www.w3.org/TR/1999/PR-xhtml1-19991210。

（3）W3C Working Drafts（工作草案）。W3C Working Draft 是概念性的，可供人们开

始实现。在将标准化为实践的过程中产生的反馈很可能影响内部细节，但不会影响规范的整体轮廓。

- XSL（Extensible Stylesheet Language，扩展样式表语言）：XML 标准规定了如何标识数据，而不是如何显示。XSL 标准本质上是一个让用户指定如何显示一个 XML 标记的翻译机制（如在 HTML 中）。可使用不同的 XSL 格式，为不同用途用不同方式显示同一数据。XSL 的翻译部分已非常完善，并有许多实现。然而，XSL 的第二部分即格式化对象，也称为对象流，可在一个页面上定义不同区域并将它们联结起来。当一个文本流被定向到集合时，它填充第一个区域，然后当第一个区域填满时“流”到第二个区域。要进一步了解 XSL，可查看http://www.w3.org/TR/WD-xsl。
- XLL（XML Link Language，XML 链接语言）：XLL 协议包括两个规范，XLink 和 XPointer，用于处理 XML 文档间的链接。这些规范仍处于初级阶段，但肯定将对 XML 文档的使用产生巨大影响。

① XLink：XLink 处理 XML 文档间的链接。它允许一些非常复杂的链接，包括双向链接、链接到多个文档、“扩展”链接（将链接的信息插入到页面中，而不是用一个新页面来取代它）、在一个独立文档中创建的两个文档间的链接，以及间接链接（可指向一个“地址簿”而不是直接指向目标文档，这样当更新地址簿时，任何使用它的文档都将自动更新）。要进一步了解 XLink，可查看http://www.w3.org/TR/WD-xml-link。

② XPointer：XPointer 使用一个文档或文档段的 ID（标识符）指向它。XPointer 定义了“在 XML 文档内部寻址”的机制，而不需要文档作者预先为那一段定义一个标识符。它提供了“元素、符号串和 XML 文档其它部分的引用，无论它们是否有一个明确的标识符属性。”要进一步了解 XPointer，可查看 http：//www.w3.org/TR/WD-xptr。

- XML Schema（大纲）：XML Schema 定义一个文档可以包含的元素类型，它们的关系和它们可包含的数据，远远超出了现有 DTD 规范的方式。尽管 DTD 能够校验 XML 文档，但它有许多缺点。一个原因在于 DTD 规范不是分层的。例如对一个包含几个“已析符号数据”（PCDATA）元素的通信地址，它的 DTD 可能是这样：

```
<!ELEMENT 通信地址 （名字，地址，邮政编码） >
    <!ELEMENT 名字(#PCDATA)>
    <!ELEMENT 地址 (#PCDATA)>
    <!ELEMENT 邮政编码(#PCDATA)>
```

可以看到，它是线性的，没有包含，可能会影响名字空间，使用户不得不在不同设置中为相似的元素使用新的名字。另一个问题是不清楚注释解释的范围。另外，它不能严格指定域的有效条件，如邮政编码必须是 6 位数字。因此目前已经提出了许多提案，构造一个更像数据库、指定校验标准的层次化大纲。可查看：http：//www.w3.org/TR/xmlschema-1 和 http：//www.w3.org/TR/xmlschema-2。

（4）W3C“Notes”（注释）。“Notes”根本不是 W3C 的标准。实际上，它是由不同个人和组织提出的提案。W3C 发布它们以供为标准工作的人参考。

- DDML / Xschema（Document Definition Markup Language / XSchema，文档定义标记语言/XSchema）：像 DTD 一样的文档定义是很好的，但 DTD 的语法有些奇怪。DDML 是旧 XSchema 提案的新名称，指定了一个 XML 文档的有效性约束。它是 DTD 的可

能后继者，但最终合法标准是哪个还不清楚。要进一步了解 DDML，可查看 http://www.w3.org/TR/NOTE-ddml。

- DCD（Document Content Description，文档内容描述）：DCD 提案是一个定义标准 XML 数据库前段的机制。要进一步了解 DCD，可查看http://www.w3.org/TR/NOTE-dcd。
- SOX（Schema for Object-oriented XML，面向对象 XML 大纲）：SOX 是一个 schema 提案，包括可扩展数据类型、名字空间和嵌入文档。要进一步了解 SOX，可查看 http://www.w3.org/TR/NOTE-SOX。

5.13 小　　结

在本章中，我们学习了 XML 文档必须符合的规则和基本语法规范，掌握了 XML 声明、如何进行注释、使用实体参考和 CDATA 节、XML Schema 以及名字空间的应用等重要知识，认识了数据岛、DTD 和 XML 文档的正规有效性。此外，对 XML 的相关标准作了初步的介绍。

第六章　XML 链接语言

本章将介绍令人兴奋的可扩展链接语言，它是一种在文档之间进行链接的崭新技术。通过与原有的 HTML 链接的比较，我们将更深入地理解简单链接、扩展链接、定位器和分段等技术并应用于程序设计的具体实践中。

本章包括以下内容：

- XLink 概述
- 深入 XLink

Web 是一种“超文本”，其数十亿的页面由超链接连接在一起，只要点击页面上带下划线的字，就能迅速从一个页面链接到另一个页面。当用 XML 驱动后，超链接将具有更多的功能。被命名为 XLink 的基于 XML 的超文本标准不久将由 W3C 发布。XLink 使用户能够从目标页面列表中选择要链接的页面。其他类型的超链接可以在用户点击的地方插入文字或图片，而不是迫使用户离开页面。

也许最有用的事实是，XLink 将使作者能够使用间接链接，该链接指向中央数据库的条目，而不是链接到页面本身。当页面地址改变时，作者只需编辑一条数据库记录就能修改所有指向该记录的超链接。这将有助于消除由于超链接断开而产生的类似于“404 File Not Found”的错误。XLink 所具有的更有效的处理、更准确的搜索、更灵活的链接等功能，将使 Web 结构产生革命性的变化，有可能产生全新的信息访问方式。用户将发现这个新的 Web 比今天的 Web 更快、更强大、更有用。

6.1　概　　述

当前，网络应用和开发者对 HTML 有限的基本链接功能很不满足。一些 HTML 开发者看到了 SGML 的更完善的 HyTime 定义和它更强的功能，但又限于 SGML 的复杂性，XML 的出现为他们提供了一个很好的机会。

与现有 HTML 链接系统由于只能提供最简单的功能而受到批评相反，许多开发者似乎对当前的链接语法非常满意，小规模的开发工具和 Web 映射工具围绕这一主要标准发展起来。HTML 的 HREF 属性令大多数开发者满意。XML 没有违反以前的标准，它只是添加进去很多东西，但是基本的 HTML 链接结构仍然保留，HTML 的某些方面引入了 XML 的标准，便于 XML 文档链接到 HTML。

用户点击一个 HTML 超链接时，当前的网页被连接到的文件替代，而 XLink 令 Web 建立者给链接增加行为。例如，现在我们可以用一些 JavaScript，使在链接处弹出一个独立的窗口，但是 XLink 让 Web 建立者对链接进行编码来执行一系列动作，包括弹出一个链接选择的菜单。

另一个应用可以是弹出一个对话框，可能是一个提醒用户它们正要更新数据库的警告。链接弹出菜单可能需要用户点击一个框来表示在进一步处理前他们接受任务。

下面我们首先看一下 HTML 文档的链接和 XLink 的对比。

6.1.1 XLink 与 HTML 中的链接

1. HTML 中的链接

在 HTML 中，A 元素几乎是所有链接的钥匙；而 LINK 元素也能起一些作用，提供多种的连结功能：比如连结到样式表（新的浏览器会提供使用者多一点选择，让其自行选择使用哪一个样式表），或是连结到有音乐的地方（当下载完毕后可自动地播放），抑或是连结到另一个网页（可使浏览器预先载入该网页，以省去等待的时间）等。

A 元素有几个属性，有一个属性在开发者那里得到频繁使用，它就是具有多功能的 HREF。HREF 通常采用 URL（统一资源定位器）作为它的值，它代表了链接目标。URL 可以是绝对地址也可以是相对地址。绝对 URL 以一个模式开始，它用来描述解释 URL 的协议。常用的模式包括 http、ftp、gopher、mailto、nntp、file 和 JavaScript。在多数情况下，它将作为到服务器或者服务器上的文件的索引出现，它的前面带有两个斜线。例如一个使用 HTTP 协议的绝对 URL 执行以下语法：

http: //hostcomputer: port /path? Query

端口和查询是可选项。主机必须包含有效 DNS 名字和 IP 地址，端口可选择指定主计算机上的端口号（80 是 http 的缺省值）。路径是指定到主计算机上的特殊文件的路径。只有数字、字母和 $、-、+、！、*、’、（、）及句号和逗号才能在路径中出现。其它的字符可以通过%标记换码，后面跟着它们的十六进制值。查询选项提供服务器的额外信息，使服务器能以适当的方式响应有关方式和信息。查询的内容限于和路径相同的字符。

相对 URL 使用当前页面（如果它存在，URL 由页面的 HEAD 元素中的 BASE 元素设置）的 URL 作为到它们信息的前缀。相对 URL 不包括模式。

绝对 URL 和相对 URL 都可以在查询的末尾包含分段识别器。HTML 中的分段识别器包括一个括号和一个值，值是链接目标文档中 A 元素的 NAME 属性。例如：

<A HREF="# mylink"> Click here to go ! < /A>

…

…

<A NAME="mylink"> Please enjoy the link!

点击“Click here to go !”将屏幕移动到带有 NAME 属性 mylink 的 A 元素指定的位置。当然也可以使用与 URL 联合的分段识别器。

<A HREF="welcome. htm # mysite">

点击它，将把用户带到包含<A NAME =“mysite”>的 welcome. htm 文件行。

A 元素也允许 REV 和 REL 标签用来表示锚点 URL 和目标 URL 之间的关系。REL 表示 URL 到锚点的关系（沿着链接向前移动），而 REV 表示锚点到 URL 的关系（沿着链接反向移动）。它们都没有在锚点标签中广泛使用。出现在 HTML 文档 HEAD 元素中的 LINK 元素，也支持 HREF、REL 和 REV 属性。与 A 元素不同的是，LINK 元素只把外边的资源连接到文档，并且使用方法与 IMP、APPLET、SCRIPT 和 OBJECT 元素使用它们的 SRC 属性的方法相同。

2. XLink 中的简单链接

XLink 定义了几种常用的连结方法：Simple、Extended、Group 和 Document。Simple 的用法比较接近在 HTML 内<a>标志的用法；Extended 的用法包含 arc 和 locator 的元素，并允许各种种类的扩充链接；Group 和 document 的用法，是让群组链接到一些特别的文件。

以下我们将说明 Simple 的语法，有两种方法可以知道一个链接是否是 XLink：

- 直接使用 simple。
- 以 XLink：type 表示。

先让我们来看看一个应用 Simple 的例子：

```
<NOTE XLink:form="simple " href=="footnote7.xml ">7</NOTE>
<COMPOSER XLink:form="simple "   inline=="true "   href="http://www.mywebsite.com">
  Tom White
</COMPOSER>
<IMAGE XLink:form="simple " href=="logo.gif "/>
```

在 XML 中，我们调用典型的 HTML 链接：嵌入式链接。链接可以连接资源（用链接元素中的定位器可以到达的任何信息），根据 XLink 工作草稿，资源包括文档，但是它也可以包含图形和其它文件。定位器是 URL，定义的方法和 HTML 定义 URL 的方法一样，并也可以同样地使用分段识别器。对于嵌入链接，定义链接的元素要作为这些资源之一来计数，并作为其它的目标。标准 HTML href 链接是非直接的链接，因为它只指出一个方向：从提供链接的元素到目标定位。对于嵌入链接，涉及的元素作为一个资源（在这个例子中，Back 按钮不计数，作为提供两个方法链接）。当链接启动，遍历描述采取的支作，即使链接不在任何新地方采取激励者（可以是用户或程序）。

在 XML 中的简单链接携带所有的在链接元素中的链接信息。在对应用程序的简单链接中，用户不需要寻找携带有关定位器信息的其它元素，所有的定位器信息放在 bref 属性中。创建一个真实的 XML 链接不是简单的；每一个链接的元素需要不只一个属性。使用简单链接元素的初步 DTD 如下：

```
<! ELEMENT simple ANY>
<! ATTLIST simiple
        Xml:link          CDATA                  #FIXED"simple"
        Href              CDATA                  #REQUIRED
        Role              CDATA                  #IMPLIED
        Title             CDATA                  #IMPLIED
        Inline            (true | false)         "true"
        Content-role      CDATA                  #IMPLIED
        Content-title     CDATA                  #IMPLIED
        Show              (embed |replace |new)  #IMPLIED
        Actuate           (auto | user)          #IMPLIED
        Behavior          CDATA                  #IMPLIED
>
```

用户不必简单地调用链接元素，因为任何元素都是一个链接元素。用在这里和工作草稿中的简单元素声明只是为了说明。事实上，任何元素都可以是一个链接——不需要创建像 HTML 中 A 那样的单独元素，它存在的唯一性是为了执行简单链接。

href 属性仍然和它在 HTML 中工作的方法一样，虽然在后面我们将会看到它也扩展了，而其余的属性是新的。第一个属性，xml:link，用来对处理应用程序宣布，这个元素表示的是一个简单链接的元素。所有的链接需要 xml：link 属性；在 DTD 中用一个固定的值定义属性，通常可以证明它是一个比在每一个单独的 instance 元素中说明属性的更好方法。避免创建文档和使用没有声明 xml:：ink 的 href 属性的 DTD。即使它们可以在 HTML 浏览器中工作，xml：link 服从处理应用程序，如果失去 xml：link 属性，将忽略 href 属性。Role 属性是可选项（像所有标记#IMPLIED 的属性，或者在这里提供一个缺省值）。Role 属性是给机器处理器使用的，给人阅读的信息存在 title 属性中。Href 属性给链接提供定位器，URL 已经做过描述。Title 也是选择项，它包括弹出的信息，类似 ALT 标签。

Inline 属性，当设置为“真”时，声明所有建立在这个元素上的链接以嵌入方式链接。在简单链接中，inline 属性按缺省变为“真”。简单链接的工作草图状态通常是嵌入方式，并且总是单方向的，但是考虑到简单外部链接，它是困难的，而这种外部链接是有用的（一种可能是链接到突出显示信息，而不期望真的浏览）。简单链接向前指向一个单一的定位器，不需要更复杂的用于多方向的工具，或外部链接。即使简单链接可以指向其它链接，但 XLink 对取得那样的结果有更好的结构。

另外两个属性是 content-role 和 content-title，它们和属性 role 和 title 完成的功能类似；然而它们描述的是定位器指向的内容，而不是链接的内容。Role 和 title 属性把链接元素作为资源描述，而 content-role 和 content-title 描述目标资源。

Show 属性表现了在 HTML 链接上，是 XML 最大的改进之一。Show 属性接受 replace、new 和 embed 作为它的值。替换是 HTML 中的标准操作，链接用目标资源替换处理应用程序中（例如浏览器窗口）当前的资源。New 属性通过 HTML’s TARGET 属性提供类似的作用，即打开新的上下文中的目标资源。在浏览器内，新的上下文可能是一个新窗口；在处理应用程序内，新的上下文可以是额外的并行操作的过程。Embed 属性加入一完整的新维数度。当链接穿过时，由 href 属性指定的资源将嵌入到源资源的体中。换句话说，它将嵌入到作为链接的元素中。Embed 属性对于开发者来说可以创建链接，它所起的更大作用就像 SRC 属性对比 ref 属性所起的作用。实际上，需要对 HTML’s SRC 创建对等物的 XML 文档，有可能使用嵌入和由启动属性产生的自动穿过结合。

启动属性接受 auto 和 user 两个值。当链接穿过之前，Usr 需要外部作用（例如用户点击）。Usr 也可以是人查看文档，或者另一个处理应用程序在使用它。Auto 需要它一遇到资源，链接就被处理应用程序穿过。当 auto 和 show 属性的 embed 值结合，auto 的作用像客户端，需要处理应用程序寻找资源和链接元素本身。由于这些资源的不同内容，处理应用程序需要多功能来支持它。

最后一个简单链接的属性是 behavior，它提供开发者另一个地方存放指导程序如何穿过这个链接的信息。不像 role 和 content-role 属性，它不是链接到特定目标的连结，并可以把链接描述为单位。XLink 规范说明不能为 behavior 提供例子的值，解释这个属性是文档作者和应用程序开发者的事。另外，behavior 属性可用来把任意格式的任意数据传递给读入此数据的应用程序中，应用程序使用这些数据来对如何进行链接作出附加说明。下面的代码指定在切断链接时，播放声音文件 music.wav：

```
<COMPOSER XLink:form="simple "
href="http://www.mywebsite.com">

behavior="sound: music.wav ">
   Tom White
</COMPOSER>
```

对于出现链接的链接元素，其 behavior 属性必须在 DTD 中声明。例如：COMPOSER 元素可以这样声明：

```
<!ELEMENT COMPOSER (#PCDATA)>
<!ATTLIST COMPOSER
XLink:form CDATA #FIXED "simple "
href CDATA #REQUIRED
behavior CDATA #IMPLIED
>
```

现在描述了这些属性，我们将用以前的声明元素来创建一个简单元素，它使用所有的这些属性并描述每一部分做什么。

```
<simple role="userlist"
href=http: //www.mywebsite.com /welcome.htm
title="welcome"content-role="hello" content-title="welcome to my site"
show="replace" actuate="user"
behavior="visit"/>
```

href 属性识别目标资源。Role 属性给我们的处理应用程序提供有关在链接中我们目标的 role 属性的信息，在这个例子中是“userlist”。Content-role 属性给同一个处理应用程序提供有关这个资源的信息，在这个例子中是“hello”。Behavior 属性把链接描述为 visit。Title 属性把目标描述为“welcome”，而 content-title 给用户提供这个资源的描述是“welcome to my site”（欢迎访问我的站点）。因为启动属性要设置“usr”，处理应用程序在指向它动作之前，不需要做任何链接，典型的是通过用户点击链接。当链接穿过，目标文档将代替链接资源（技术上看是元素，但是很象整个文档），因为 show 属性设置为“replace”。

另一个实例：

```
<my:crossReference
  xmlns:my="http://example.com/"
  xmlns:XLink="http://www.w3.org/1999/XLink"
  XLink:type="simple"
  XLink:href="students.xml"
  XLink:role="studentlist"
  XLink:title="Student List"
  XLink:show="new"
  XLink:actuate="onRequest">
Current List of Students
</my:crossReference>
```

由于属性名和类型都是标准化的，这样，如果一篇文档中有多个链接元素，可将属性声明变成参数实体引用，并在每个链接元素的声明中重复引用，非常方便。实例如下：

```
<!ENTITY %link-attributes
        "XLink:form CDATA #FIXED ‘simple ’
        href            CDATA       #REQUIRED
        behavior        CDATA       #IMPLIED
        content-role    CDATA       #IMPLIED
        content-title   CDATA       #IMPLIED
        role            CDATA       #IMPLIED
        title           CDATA       #IMPLIED
        show            (new |replace |embed)    ‘new ’
        actuate         (user |auto)             ‘user ’
        behavior        CDATA       #IMPLIED ">
<!ELEMENT COMPOSER (#PCDATA)>
<!ATTLIST COMPOSER
          %link-attributes;>
<!ELEMENT AUTHOR (#PCDATA)>
<!ATTLIST AUTHOR   %link-attributes;>
<!ELEMENT WEBSITE (#PCDATA)>
<!ATTLIST WEBSITE   %link-attributes;>
```

6.1.2 XLink 实现 HTML 链接

了解了两者的异同后，现在可以在 XML 中重构 A 元素了。我们从一个简单的文本开始，并加入一些特性，以便可以清晰地看出 XML 与 HTML 有何不同。这里 A 元素属性声明为：

```
<! ELEMENT A ANY>
<! ATTLIST A
        xml:link          CDATA          #FIXED"simple"
        href              CDATA          #REQUIRED
        title             CDATA          #IMPLIED
        inline            (turel| flase)  "replace"
        actuate           (auto| user)    "user"
>
```

现在，XML 中的 A 元素将像它在 HTML 中一样以下面的形式工作：<A href=“target.html”>或<A href=“target.html” title=“My Weblink”>。在 XML 中，xml：link、inline、show 和 actuate 值已经指定缺省值，不需要明确声明。A 元素表示用户启动的标准例子，简单的嵌入链接，按缺省方式，将用目标资源的内容代替源资源的内容。

REV 和 REL 属性的大部分分别被 role 和 content-role 代替。虽然我们也可以为 A 元素只产生 REV 和 REL 属性来完成任务，但最好还是把它们移到 XML 中。XML 提供把 xml：link 属性重新映射到其它属性名的结构。在元素已经有名为 role 和 title（与链接毫无关系）的属性地方，以及产生向后兼容属性的情况，我们就会发现经常要使用重新映射。使用重新映射需要加入 xml：attribute 属性。这个属性的值是属性名成对地列表。这对成员中的第一个成员必须是 xml：link 属性。这个属性的值是属性名成对地列表。这对成员中的第一个成员必须是 xml：link 属性的标准名；第二个成员的属性名是它的映射。在我们的例子中，我们需要为 A 元素加入以下属性列表到属性声明：

```
Xml:attributes      CDATA #FIXED "role REL content-role REW"
REV     CDATA    #IMPLIED
REL     CDATA    #IMPLIED
```

Xml：attributes 声明需要把 REL 属性值当作通常 XML role 属性链接的值来处理，而 content-role 属性的值等于 REV 属性的值。

HTML 中的 A 元素的有一个特点：它不能直接把下一页转到 XML 中。TARGET 属性虽然不可使用，但可以通过给 show 一个“new”值产生同样的结果。target 属性与 show 属性结合就可以映射到 behavior 属性，但是成功与否完全依靠处理应用程序解释 behavior 属性的能力。Web 浏览器可以处理这个问题，但是 XML 解析器不能。

HTML 中的 A 元素在 XML 中需要编址的最后一个属性是 NAME 属性，它在 HTML 中用来创建分段识别器。Xml：link 采用不同的方法创建分段识别器。现在为可以为类型 ID 的 A 元素创建 NAME 属性：

```
NAME      ID      #IMPLIED
```

通常，调用 ID-type 要归于 ID。这更便于使用层叠样式表和其它期望发现 ID 属性的结构。从长远看，需要重新命名 NAME 属性 ID。

Link 遇到一个简单的#fragmentidentifier，它将检查文档中 ID 值列表。ID 值包括所有以 ID 定义的属性，不只是名为 ID 的那些（注意：要记住声明 ID 属性作为类型 ID；否则，XLink 将忽略它）。在这个例子中，XML 可以使用 NAME 属性，因为它是类型 ID，作为分段识别器。这就产生了平衡提供

XML 功能的 HTML 语法。我们的（在多数）完整 A 元素属性声明如下所示：

```
<! ELEMENT A ANY>
<! ATTLISTA
xml:link                CDATA              #FIXED   "simple"
xml:attributes          CDATA              #FIXED "role REL content-role REV"
href                    CDATA              #REQUIRED
title                   CDATA              #IMPLIED
inline                  (true|fa|se)       "replace"
show                    (embed|replace|mew)  "user"
REV                     CDATA              #IMPLIED
REL                     CDATA              #IMPLIED
NAME                    ID                 #IMPLIED
TARGET                  CDATA              #IMPLIED
```

6.2 深入 XLink

6.2.1 定位器和分段

即使 XLink 可以使用简单的 HTML 分段识别器符号，但它使用 XPointer 将是有趣的事情。许多 XPointer 完成的任务是十分有用的，即使在简单的链接中，使作者和开发者能够处理元素，而不是文档，作为涉及链接的主要单位。因为它可以使用 XPointer；XLink 定位器语法比 HTML 的强大得多，它提供一些可以通过结构，给文档的部分、ID、HTML 锚点甚至文本内容编址的工具（XPointer 将在下一章中详细介绍）。

HTML 使分段识别器能够跟踪使用以下 href 属性语法的 URL：

```
Href="url#fragmentidentifier"
```

XML 也有类似的语法功能，但是使用更高级的 XPointer，而不是 HTML 分段识别器：

```
Href="url # XPointer"
  或  href="url | XPointer"
  或  href="url? XML_XPointerTR=XPointer"
```

在第一个语句中，使用#，由 URL 参考定位，作为整个文档的定位被处理器取走（如果 show 属性设置为“replace”——代替当前的文档），然后被 XPointer 参考的定位由客户定位。在第二个语句中，使用“|”连结符，根据 XLink 的标准，把没有意向的信号发到处理模型，是用于访问指定的资源。使用标准查询语法的最后文本给服务器提供处理 XPointer，减小传输所需带宽，因为服务器可以只返回它所需要的那些文档。

XPointer 给 show 属性提供大量更强功能的 embed 值，因为它可以使链接访问到文档的部分，用当前文档的上下文显示，而不是完全代替当前文档。ID 值的使用，使作者容易地把文档细分为有很好的结构元素、便于管理的组块。用户可以显示一个很长的文件，这个文件包含许多较小的块分解为更小的块，而不是作为一个很大的文件显示它。这就使利用其它系统的已知链接特性来获取其它文档变得容易起来。

6.2.2 扩展链接

扩展链接给链接世界一个全新的几何结构，使建立新的体系结构和界面成为可能。扩展链接是多方向的链接，从长远来看，外部链接使开发者可以创建更复杂的结构，更容易对链接进行管理。

扩展链接使开发者可以创建链接组，有效地为用户（或处理应用程序）提供一套不只是单个目标链接的选择。虽然应用程序处理扩展链接在工作草图中仍不很明确，但我们可以容易地描绘出作为一套选择的扩展链接，它将出现在弹出的菜单（或其它界面）中，让用户选择方向。用在这里的标准应用程序是一个辞典。当用户点击单词，一套同义词将出现在弹出的菜单中。用户可以从这些单词中选择，或者在所选择的单词中查看更进一步的信息。

扩展链接的“指向多个目标”特性实例如下：

```
<WEBSITE XLink:form="extended ">Cafe au Lait
        <locator href="http://metalab.unc.edu/javafaq/">
        North Carolina
        </locator>
        <locator
        href="http://sunsite.univie.ac.at/jcca/mirrors/javafaq/">
        Austria
        </locator>
        <locator href="http://sunsite.icm.edu.pl/java-corner/faq/">
        Poland
        </locator>
        <locator href="http://sunsite.uakom.sk/javafaq/">
        Slovakia
        </locator>
        <locator href="http://sunsite.cnlab-switch.ch/javafaq/">
        Switzerland
        </locator>
</WEBSITE>
```

执行这些链接有一点复杂。当我们做简单链接时，我们开始检查一些样本声明，在这个例子中，这个元素能表现扩展外部链接：

```
<!ELEMENT extended ANY>
<! ATTLIST extended
      xml:link        CDATA          #FIXED"extended"
      imline          (trrel | flase)      "ture"
      content-role    CDATA          #IMPLIED
content-title         CDATA          #IMPLIED
>
```

在以前的简单例子中，执行扩展链接的元素不需要给扩展命名。它们只需要简单地把 xml：link 属性设置为“extended”。

扩展元素和简单元素基本类似，但有两点变化。

（1）Xml：link 属性值由原来的“simple”改变“extended”。

（2）扩展元素没有任何 href 属性和任何目标描述。扩展元素必须包括一套包含定位器的子元素。定位器元素如下所示：

```
<! ELEMENT locator ANY>
```

```
<! ATTLIST locator
  xml:link      CDATA                #FIXEX"locator"
  role          CDATA                #IMPLIED
  href          CDATA                #REQUIRED
  title         CDATA                #IMPLIED
  show          (embedf | replace | new)   "rep;ace"
  actuate       (auto | user)                "user"
  behavior      CDATA                  #IMPLIED
>
```

定位器元素携带关键链接信息。Href 携带用于启动链接的定位器。Title 提供用于给人阅读的信息，而 role 携带用于提供给处理应用程序的信息。每一个定位器都有自己的 show、actuate 和 behavior 属性。如果定位器不定义这些属性，它将用在扩展元素中，按缺省指定的属性。这就使创建指向不同位置但共享相同的 behavior 属性的链接组变得容易。同时保留了当需要时，单个链接到不同 behavior 的权利。

我们将会发现扩展链接，不论是嵌入还是外部，在数量很大的情况下都有用。嵌入链接可以使用如下所示的 DTD：

```
<extended> History Texts
<locator title="African" href = "African.xml" />
<locator title="Asian" href= "asian.xml" />
<locator titlo="European" href= "European.xml" />
<locator title="North American" href= "namerican.xml" />
<locator title="Pacific" href= "pacific.xml" />
<locator title="South American" href="samerican.xml" />
</extended>
```

这个链接完成了几件事情。第一，如果用户在文档中遇到它，这个元素表示实时嵌入链接。点击单词“History Texts”将出现一个菜单，它提供在定位器元素中的选择标题的列表。选择这些标题之一，就会把用户带到索引文档。第二，因为这是扩展链接，它建立的链接可以连结到所有这些文档。在定位器中列表的文档，如果它们在某处遇到这个声明，所有相互的链接在文档级。右击它们中任何一个的页边空白，可以调出其它文档的菜单，这个文档点击了链接。当我们到达 xml：link 属性文档值，我们还会重复它。

扩展外部链接类似以前的链接，除了扩展链接元素的内容不是它自己链接部分外。点击扩展链接元素的内容，如果它有任何信息出现，就不要点击菜单。扩展外部链接只建立其它元素之间的连结。例如：

```
<extended inline="false">
<locator title="Overview" href="#overview" />
<locator title="Architecture" href="#architecture" />
<locator title="Detailed Design" href="#details" />
<locator title="Parts List" href="#parts" />
</extended>
<SECTION ID="overview"> <title> Overview </title>
…
</SECTION>
<SECTION ID="architecture"> <title> Architecture </title>
…
```

```
</SECTION>
<SECTION ID ="details"> <title> Detailed Design </title>
…
</SECTION />
<SECTION ID="parts"> <title> Parts List </title>
…
</SECTION>
```

在这个例子中的扩展元素除了链接外，没有其它内容。它在跟随的 SECTION 元素之间建立链接。不关心 Overview SECTION 的读者可以点击它并得到一个更详细描述的菜单。类似地，闯进部件列表的人也可以使用同样的菜单返回到 Overview 或 Architecture SECTION 元素。这些链接真正具有多方向的功能（如果应用程序支持它），因为用户可以在多个定位之间导航，不用关心向前或向后移动，在这个意义上，正是客户所期望的。这些链接可以向两个方向穿行——点击目标文档（或使用它的链接界面）将出现链接列表，它包括用户要访问的文档。增加的灵活性无疑会搞乱许多在 HTML 链接的 Web 中已经迷惑的用户，但是对有经验的用户是一个功能强大的导航工具。

下面我们再看一个实例，以加深对扩展链接的理解。首先是一个非标准的扩展链接元素及其子元素的声明：

```
<!ELEMENT courseload ((tooltip|person|course|gpa|go)*)>
<!ATTLIST courseload
   xmlns:XLink      CDATA              #FIXED "http://www.w3.org/1999/XLink"
   XLink:type       (extended)         #FIXED "extended"
   XLink:role       CDATA              #IMPLIED
   XLink:title      CDATA              #IMPLIED>

<!ELEMENT tooltip ANY>
<!ATTLIST tooltip
   XLink:type       (title)            #FIXED "title"
   xml:lang         CDATA              #IMPLIED>

<!ELEMENT person EMPTY>
<!ATTLIST person
   XLink:type       (locator)          #FIXED "locator"
   XLink:href       CDATA              #REQUIRED
   XLink:role       CDATA              #IMPLIED
   XLink:title      CDATA              #IMPLIED
   XLink:label      NMTOKEN            #IMPLIED>

<!ELEMENT course EMPTY>
<!ATTLIST course
   XLink:type       (locator)          #FIXED "locator"
   XLink:href       CDATA              #REQUIRED
   XLink:role       CDATA              #FIXED "http://www.example.com/linkprops/course"
   XLink:title      CDATA              #IMPLIED
   XLink:label      NMTOKEN            #IMPLIED>

<!ELEMENT gpa ANY>
```

```
<!ATTLIST gpa
   XLink:type        (resource)          #FIXED "resource"
   XLink:role        CDATA               #FIXED "http://www.example.com/linkprops/gpa"
   XLink:title       CDATA               #IMPLIED
   XLink:label       NMTOKEN             #IMPLIED>

<!ELEMENT go EMPTY>
<!ATTLIST go
   XLink:type        (arc)               #FIXED "arc"
   XLink:arcrole     CDATA               #IMPLIED
   XLink:title       CDATA               #IMPLIED
   XLink:show        (new
                     |replace
                     |embed
                     |other
                     |none)              #IMPLIED
   XLink:actuate     (onLoad
                     |onRequest
                     |other
                     |none)              #IMPLIED
   XLink:from        NMTOKEN             #IMPLIED
   XLink:to          NMTOKEN             #IMPLIED>
```

完成元素定义声明后，我们就可以利用这些声明来建立扩展链接，代码如下：

```
<courseload XLink:title="Course Load for Pat Jones">

   <person
      XLink:href="students/patjones62.xml"
      XLink:label="student62"
      XLink:role="http://www.example.com/linkprops/student"
      XLink:title="Pat Jones" />

   <person
      XLink:href="profs/jaysmith7.xml"
      XLink:label="prof7"
      XLink:role="http://www.example.com/linkprops/professor"
      XLink:title="Dr. Jay Smith" />
   <!-- more remote resources for professors, TAs, etc. -->

   <course
      XLink:href="courses/cs101.xml"
      XLink:label="CS-101"
      XLink:title="Computer Science 101" />
   <!-- more remote resources for courses, seminars, etc. -->

   <gpa XLink:label="PatJonesGPA">3.5</gpa>
```

```
    <go
      XLink:from="student62"
      XLink:to="PatJonesGPA"
      XLink:show="new"
      XLink:actuate="onRequest"
      XLink:title="Pat Jones's GPA" />
    <go
      XLink:from="CS-101"
      XLink:arcrole="http://www.example.com/linkprops/auditor"
      XLink:to="student62"
      XLink:show="replace"
      XLink:actuate="onRequest"
      XLink:title="Pat Jones, auditing the course" />
    <go
      XLink:from="student62"
      XLink:arcrole="http://www.example.com/linkprops/advisor"
      XLink:to="prof7"
      XLink:show="replace"
      XLink:actuate="onRequest"
      XLink:title="Dr. Jay Smith, advisor" />

</courseload>
```

6.2.3 扩展链接组

上面的方法看上去很有吸引力，但它们使管理链接更加困难。XML 文档不能总知道和它共享链接的其它文档，特别是如果那些链接存储在属于其他人的内容上。管理错综复杂的链接是对链接信息中心化的挑战。幸运的是，xml：link 提供一些基本工具，专门处理这类问题。用于 xml：link 属性的组和文档值可以创建扩展链接组元素，它帮助管理链接，告诉文档检查彼此相关链接，并为有关数据组建立票据交换所。使用扩展链接组需要创建两个元素：一个用来定义组，另一个识别组中的文档。对这些元素的声明类似以下形式：

```
<!ELEMENT group (document * )>
<! ATTLIST group
                    xml:link          CDATA        #FIXED "group"
                    step              CDATA        #IMPLIED
>
<! ELEMENT document                   EMPTY>
<! ATTLIST   document
                    xml:link          CDATA        #FIXED "document"
                    href              CDATA        #REQUIRED
>
```

这些声明比跟在它们后面的简单得多。在组元素中，所有在元素实例中需要声明的是 step 属性，它告诉处理应用程序，在停止搜索相关链接之前，有多少链接层。文档元素只包含 href，它把处理应用程序带到用户想要搜索链接的文档。当处理应用程序遇到这些元素中的任何一个，它将装入由在组中文档元素 href 属性指定的文档。然后它检查这些文档，并连接到源文档，创建一个链接表格。处理应用

程序将装入所有的文档，并处理在它们中的链接信息。如果 step 属性大于 1，处理应用程序把文档装入到源文档被第一个装入圆括号文档链接的地方。

扩展链接组的优点，是它们可以使处理应用程序容易地获取有关文档的完全信息，这些最初的文档是链接到的，而不是在打开新文档时发现链接。用户可以在组里安排一套文档，使相关的链接容易发现：

```
<group step=1>
<document href="linkgroup2.xml" />
<document href="linkgroup3.xml" />
<document href="linkgroup4.xml" />
<document href="linkgroup5.xml" />
</group>
```

当处理应用程序到达在文件 linkgroup1.xml 中的元素组，它将打开文档 linkgroup2.xm1、linkgroup3.xml、linkgroup4.xml 和 linkgroup5.xml。在解析它们后，它将决定它们是否有任何到 linkgroup1.xml 的链接。如果它们有，处理应用程序将把这些链接加到 linkgroup1.xml 的链接列表中，并使它们可以被用户使用。没有这些链接，linkgroup1.xml 用户只能看到这些链接，它们来源于 linkgroup1.xml 文件本身，是一套非常受限制的链接。

扩展链接组使中心化链接信息成为可能，取代了错综复杂的链接，而使开发者能够检查和管理链接，而不必阅读没完没了的文档。它也减少了带宽成本，使开发者能够要求 XML 文档只下载一个（或几个）额外文档创建完整的链接列表。

扩展链接组也使把链接有效地从其它文档加到用户自己的文档成为可能。只要其它文档认可用户的文档，从用户的文档到目标文档的链接将被目标文档认可，并可以作为输出链接出现。即使目标文档不认可用户的文档，当浏览文档时，处理应用程序仍能记住来自用户的文档的链接，使文档相互之间可以方便地参考。如果处理应用程序记住从文档到文档之间的链接，目标文档可以链接到用户的文档，虽然只在处理应用程序的上下文内。这使得注释文档比过去更简单得多。

XML 链接似乎很复杂，但是进一步研究和使用将使它们更友好。如果我们对它进行深入地应用，将会发现这些链接设计的含义就像 XML 的含义一样生动而灵活。

6.3 小　　结

通过本章的学习，我们可以描述 XML 的链接机制，创建简单的或扩展的 XML 链接，决定在应用程序中如何选取不同的链接实现方式。随着愈来愈多的浏览器和相关技术的支持，XLink 的前景将是一片光明。

第七章　XML 指针语言

本章将介绍与 XLink 关系紧密的 XML 指针语言，它将使得 XLink 更为精确。XPointer 详细描述了链接如何在文档里面进行寻址和引用特定的部分。

本章包括以下内容：

- “第一次亲密接触”XPointer
- XPointer 规范及应用

XML 整洁定义的结构使它有可能容易地描述文档部分，而不用丢掉带有标识符的文档。利用嵌套结构和元素名字潜在的重要意义，XPointer 可以指定用户提取文档中一部分内容的路径。当 XPointer 与 XLink 说明结合在一起时，它就会成为在文档间精确指定链接的功能强大的工具。

（注：我们这里使用的是 XML Pointer Language——XPointer Version 1.0，7 June 2000 草案文档，详情请查阅 http://www.w3.org/TR/2000/CR-xptr-20000607）

7.1　“第一次亲密接触”XPointer

XPointer（XML Pointer Language，XML 指针语言）是定义 XML 文档各部分的寻址方案。XLink 指向一个特定资源的 URI（实际上是 URL），URI 可以包含 XPointer 部分，以便更明确地标识目标资源或文档所要求的部分和元素。在 HTML，要链接到一个页面的中间，页面作者必须在那儿加上定位标识符。使用 XPointer，用户可以“取址到”(不是“连接到”)其他人的文本的任何部分。显而易见，这样将有助于工作于法律文件，科学和学术论文，甚至 W3C 规范。

XPointer 在搜索文档，并把文档分段的住处子集返回上有强大的功能。然而，XPointer 不是真正的查义语言（它们更像一套公路图的指南），它们有很多相同的功能，即优化文档导航。XPointer 把着眼点放在位置而不是内容上，然而它不是真的指向 XPointer 看起来像的结果。典型情况下，它们以容易处理的树型结构出现（像树的子集，由 W3C 文档对象模型或 DOM 创建），但是在有些情况下，它们可以包含元素部分，甚至属性。

XPointer 起源于文本编码主动性（TEI）标准。XPointer 用组成定位器的定位项指定资源，它可以包含单一的定位器（虽然一些定位器可以包含其它的定位器）。我们先建立简单的定位，然后再把它们联合起来。定位器可以包含绝对、相对、范围和字符串匹配定位项。字符串匹配项有最多的受限制的词表，但是它们提供的精确程序是其它项不能相比的。属性项使得它可以检索属性值以及元素内容。范围项参考了两个定位器之间的信息。相对项使链接能够参考文档的内容，绝对定位项用更传统的 ID 和 NAME 编址方案以及其它基本定位来识别元素。

通过使用 XPointer，XLink 允许文档之间的更为复杂的链接：用 XPointer 引用文档的特定元素；引用第一个、第十个元素；引用特定元素的子元素或者其他操作。XPointer 不但指向文档中的某一点，还可以指向一个范围。下面是 XPointer 的几个简单实例：

```
root()
id(dt-xmldecl)
```

descendant(2,termref)

following(,termdef,term,CDATA Section)

html(recent)

id(NT-extSubsetDecl)

上面的每个语句都选定文档中的特定元素，XPointer 中没有指定文档，但 XLink 指定 XPointer 作用的文档。将 XPointer 加入 XLink 并不难，只需要把 XPointer 追加到 URI 中，以#分开。我们将上面的 XPointer 作为 URL 的后缀示例如下：

http://www.w3.org/TR/1998/REC-xml-19980210.xml#root()

http://www.w3.org/TR/1998/REC-xml-19980210.xml#id(dt-xmldecl)

http://www.w3.org/TR/1998/REC-xml-19980210.xml#descendant(2,termref)

http://www.w3.org/TR/1998/REC-xml-19980210.xml#following(,termdef,term,CDATA Section)

http://www.w3.org/TR/1998/REC-xml-19980210.xml#id(NT-extSubsetDecl)

通常，它们作为 locator 元素的 href 特性值使用：

<locator

href="http://www.w3.org/TR/1998/REC-xml-19980210.xml#root()">

Extensible Markup Language (XML)1.0

</locator>

我们还可以使用竖线（|）来代替#，表示不链接整个文档，只获取 XPointer 引用的文档的一部分，具体如下：

http://www.w3.org/TR/1998/REC-xml-19980210.xml|root()

http://www.w3.org/TR/1998/REC-xml-19980210.xml|id(dt-xmldecl)

http://www.w3.org/TR/1998/REC-xml-19980210.xml|descendant(2,termref)

http://www.w3.org/TR/1998/REC-xml-19980210.xml|following(,termdef,term,CDATA Section)

http://www.w3.org/TR/1998/REC-xml-19980210.xml|id(NT-extSubsetDecl)

7.2 XPointer 规范及应用

7.2.1 绝对定位项

绝对定位项提供对少数关键 XML 文档块的访问。root（）、origin（）等关键词使它很容易为一些元素编址，这些元素是：文档的根元素、使用关键词的元素、带有 ID 的元素和标准 HTML 中 A 带有 NAME 属性值的元素。绝对定位项必须出现在定位器的开始部分；如果不指定绝对定位项，root（）将被指定为缺省绝对定位项。

Root（）项指定文档的根元素，它是文档树最外面的元素。例如http://www.w3.org/TR/REC-xml文档的根元素是spec，要选择它，可以用下面的URI：

http://www.w3.org/TR/REC-xml #root()

Root（）有效地访问整个文档的内容；Origin（）关键词访问链接元素本身，并经常为后来的相对项提供绝对位置。这就使得 XPointer 能够指定内容，例如像“第二节文档放到链接元素下面”。这样便需要跟在 root（）和 origin（）关键词后的空插入语。

绝对定位项应用实例：

```
AbsTerm ::=   'root()' | 'origin()' | IdLoc | HTMLAddr
IdLoc ::=   'id(' Name ')'
HTMLAddr ::=   'html(' SkipLit ')'
```

7.2.2 相对定位项

相对定位项很灵活，但更复杂。相对定位项的关键词是导航文档树的工具，它的列表如下。

Child	选择定位资源子段元素（必须是直接资源下的嵌套的元素）
Descendant	选择在定位资源内容中出现的元素（可以嵌套多个级）
Ancestor	选择它的内容在定位资源中被发现的元素（父元素）
Preceding	选择在定位资源前出现的元素
Psibling	选择在定位资源以前出现的同属元素（同属元素有相同的父元素）
Following	选择出现在定位资源后的元素
Fsibling	选择在定位资源后出现的同属元素（同属元素有相同的父元素）

所有的相关关键词都使用一套相同的参数，用圆括号封装：

```
(instance, Node Type, Attribute, Value)
```

必须包括 Instance（它是数值，或指“所有”)和 Node Type,Attribute 和 Value 是可选择的，只有当需要用属性值来识别时，才使用这两个元素。Instance 使开发者能够在相对定位资源的位置中指定元素。Node Type 定义定位项的候补类型。

Instance 和 Node Type 使开发者可以指定，用相对定位关键词描述的结构中某一节点，第 n 个的外观。这可以使它容易地用在文档中的位置指定相对位置。Instance 的值即可以是正整数，也可以为负整数（例如：3、-126、+88），或者是关键词 all。使用 all 做 instance 的值，它将返回到所有满足标准的节点，这个标准是由剩余的定位项指定的。而数字表示沿着文档树移动的方式，这个方式由定位项指定。正整数使计数从符合定位项的第一个结构外观算起，而负整数表示计数从最后一个符合定位项的结构外观算起。例如：

CHILD（3，QUOTE）表示的是定位资源元素中的 QUOTE 元素的第三个外观。

CHILD（5，EXPLANATION）表示定位资源元素中的 EXPointerLANATION 元素的第五个外观。

使用负数做为 instance 的值，计数的方向相反，例如，CHILD（-1，PRICE）代表在定位资源元素中的最后一个 PRICE 元素，而 CHILD（-3，PRICE）代表在定位资源元素中，从最后一个往前算起的第三个元素。

Node Type 典型地以元素名字出现。然而，开发者可能不会总知道，或者需要知道它们指定目标的名字。为了适应这种情况，XML 可以使 Node Type 采用不是元素名的三个值。值#element 使得定位项接受所有的元素作为匹配侯选者，CHILD（2，#CLEMENT）表定位资源元素中第二个元素。#text 值告诉定位项接受文本节点作为匹配侯选者。文本节点是由字符数据构成，它和混合声明中的标记合并。例如：

```
<CASE> Name: <NAME> Jim </NAME>
Fish bonker: <BONKER> 01234 </BONKER>
```

Crime: <CRIME> Fishing without mowing the lawn first </CRIME>

Status: <STATUS> Dismissed < /STATUS> < /CASE>

第一个虚拟元素（child(1,#text)）包含文本“NAME：”，第二个元素（child(2,#text)）包含新行字符和“Fish bonket：”，等等。

XPointer 也可以索引注释、处理指令和 CDATA 部分。#command 值索相命令，而#pi 索引注释，#cdata 索引 CDATA 部分。最后，#al 表示 Nade Type 最后一个可以得到的值。如果应用到以前的例子，使用 XASE 元素作为定位资源，CHILD（1，#all）将索引“NAME：”，而 CHILD（2，#all）将索引 NAME 元素后面的元素。

在 Attribute 和 Value 中加入参数，执行属性匹配，可以进行另一级别的精确指定定位。Attribute 和 Value 都是可选择的。但是如果 Attribute 参数出现，Value 参数也必须跟随出现。CHILD（1，*，TARGET，ME）索引包含带有 ME 值的属性为 TARGET 的定位资源中的第一个元素。CHILD（2，QUOTE，SPEAKER，ROOSEVELT）索引把 SPEAKER 属性设置为 ROOSEVELT 的定位资源中的第二个 QUOTE 元素。这两个参数都接受*值代替指定参数。如果你用*代替 Attribute 参数，任何属性的参数值都将和它匹配。如果你用*作为 Value 的参数，由属性参数命名的属性可以是任何值。CHILD（1，QUOTE，SPEAKER，*）索引有 SPEAKER 属性声明的第一个 QUOTE 元素，不管它的值是什么。CHILD（3，QUOTE，*，ROOSEVELT）索相，值设置为 ROOSEVELT 的任何属性的第三个 QUOTE 元素。属性可以命名为 PRESIDENT、SPEAKER 或 Q2FD，这没有关系。

Value 参数也可以接受值#IMOLIED。类似属性列表声明的缺省类型，它是索引已声明的，但没有分配缺省值的属性。在这种情况下，它只索引文档中已声明但还没有指定的属性。如果元素给属性提供了一个值，即使只有一个声明#IMPLIED，使用#IMPLIED 作为值的 XPointer 将不再认可它。

7.2.3 属性定位项

属性定位项看起来可能比大多数其它项简单，用来和识别元素的项结合时，它只采用一个属性名作为选择器，并返回属性值。这在某种意义上是唯一的，因为其它的 XPointer 指定定位而不是对每一个服务进口返回值，但是读者很快会认识到这种项的使用。它使用的语法如下：

attr（Attribute Name)

例如，att（id）将把一个来自当前识别的元素的名为 id 的属性值返回（或者根元素，如果在它之前没有其它声明）。

7.2.4 范围定位项

Span（）关键词和 XPointer 合并可以收集从第一个 XPointer 开始到第二个 XPointer 结束的所有信息。但是，如果第一个 XPointer 的索引定位出现在第二个参数之后，将会出现一些无法预料的情况。

Spanning 使它可以容易地产生与 XPointer 合并，简化产生用于各种任务的链接。Spon（）的语法看起象以下形式：

Span(XPointer1, XPointer2)

这两个 XPointer 可以分开单独解释；第一个 XPointer 的选择对第二 XPointer 的选择没有任何影响。Span（）项然后选择所有的在第一个 XPointer 开始和第二个 XPointer 结束之间的内容。

<PERSON>

Name: <NAME>Tom < /NAME>

News editor: < AGE > 36 < /AGE>

ADDRESS: <STATE> NEW YORK < / STATE >

ID: <PID> 64589508 < /PID>

< / PERSON >

索引以上的文档，XPointer（span(child(NAME, 1), child(AGE,1)识别以下分段：

<NAME> Tom < /NAME>

News editor: < AGE > 36 < /AGE>

类似，XPointer（span(child(NAME,1) ， child(#text,2))识别以下分段：

<NAME> Tom < /NAME>

News editor:

7.2.5 字符串定位项

字符串定位项是最后一套定位项。字符串定位关键词使用以下语法：

string (Instance, String, Position, Length)

Instance 参数和它在相对定位项中的工作方式相同。String 参数只是匹配字符串（在搜索匹配字符串中，处理器将忽略所有的标记字符，以便字符串值可以通过元素边界）。Position 提供从识别定位返回的精确字符位置偏移量，从发现的字符串的第一个字符算起。如果 Position 是零（或出现差错），那么定位将是所发现的字符串的第一个字符；如果它是 2，它是开始后的第三个字符。Length 不在适当的位置，定位以所发现的字符串为参考）。示例如下：

<LINE> The worms crawl in and the worms crawl out </LINE>

<LINE> The ones that crawl in are lean and thin < /LINE>

<LINE> The ones that crawl out are fat and stout </ LINE>

定位项 string（1,worms,4,1）返回定位字母 s，它是第一行第一个单词的外观“worms”的字母 s。String（2,worms,2,1)返回定位 m，它是单词第二个外观“worms”中的字母 m。String（1,crawl,1,1）、string（2,crawl,1,2）、string（3,crawl,1,3）和 string（4,craw1,1,1,4）返回定位字母 c,它是单词“crawl”第一个外观字母 c，字母“cr”在第二个，“cra”在第三个，“craw”在第四个。

7.2.6 合并定位项

当定位项在合并中使用时，定位项便产生很多功能。绝对定位项可以方便地指定新缺省定位器元素，代替缺省 root（）。相对定位项可以组合，使用户可以在第五个 FRAB 元素（它的 FLIPGELLY 属性设置为“postmaster”）外观上直接指定元素。合并项是简单的，只是指导它们按顺序列表。项将从左到右对它们进行估算。例如，给出以下文档：

```
< PUBLISHERSHELF ID="Music Publishers">
    <publisher>
            <name> ACA - American Composers Alliance</name>
            <email>info@composers.com</email>
            <homepage>http://www.composers.com/</homepage>
            <address>
                <street>170 West 74th St.</street>
                <city>NY</city>
                <state>NY</state>
                <zip>10023</zip>
            </address>
            <voice>212-362-8900</voice>
```

```
        <fax>212-874-8605</fax>
</publisher>

<publisher>
        <name>Alfred Publishing</name>
        <address>
                <street>15535 Morrison</street>
                <city>South Oaks</city>
                <state>CA</state>
                <zip>91403</zip>
        </address>
</publisher>

<publisher>
        <name>Music 70</name>
        <address>
                <street>250 West 57th Street</street>
                <street># 232</street>
                <city>NY</city>
                <state>NY</state>
                <zip>10107</zip>
        </address>
</publisher>
```

定位器 id（Music Publishers）选择 PUBLISHERSHELF 元素。

Id（Music Publishers）.child（1, publisher）选择第一个 publisher 元素。

name 元素，被 id（Music Publishers）. child. （1, publisher）. Child （1, name）选择。

Id（Music Publishers ）.child. （3, publisher）. PSIBLING（1, publisher）选择 publisher 元素。

第二和第三个 publisher 元素和它们的内容被 span（id (Music Publishers). Child（2, publisher）; id（Music Publishers）. Child（3, publisher）选择。甚至当程序块不能整齐地匹配单个元素时，也能给开发者提供用于产生程序块的所有功能。

现在我们可以创建一些功能强大的索引，它超过允许开发者通过 HTML A 元素的命名功能。我们的索引可以通过文档结构精确地定位他们的目标。下一步把这些索引用新链接技术合并，进一步扩大工具箱，把链接提高到一个全新的水平。

7.3 小　　结

XPointer 与 XLink 相结合，为 XML 文档的链接创造了一个强有力的环境，将超链接带入了一个前所未有的高度并会用一种不可预见的方式改变 Web 和我们的生活。