第 3 章　文档类型定义

应用 XML 的主要目的是为了存储和交换数据，但当一个应用程序中的 XML 文档数据来自不同的源时，保证所有的 XML 文档都遵循一个议定的 XML 结构（标记名称、属性名称、嵌套等）是非常重要的。那么如何保证每个提交的 XML 文档都遵循相同的 XML 结构呢？这就需要验证 XML 文档。文档类型定义（Document Type Definitions，DTD）的作用是定义 XML 文档的合法构建模块，它使用一系列合法的元素来定义文档的结构。通过 DTD，每一个 XML 文件均可携带一个有关其自身格式的描述。通过 DTD，独立的团体可基于某个标准的 DTD 来交换数据，而应用程序也可基于某个标准的 DTD 来验证从外部接收到的数据。用户还可以使用 DTD 来验证自身的数据。DTD 可在 XML 文档中声明，也可作为一个外部引用。本章主要内容：

- DTD 语法规则
- 创建 DTD
- 应用 DTD 验证 XML 文档

3.1　DTD 语法规则

通过前两章的学习，读者已经了解到，所有的 XML 文档均可分解为简单的构建模块，即元素、属性、实体以及 PCDATA 和 CDATA。元素是 XML 文档的主要构建模块，属性可提供有关元素的额外信息，实体是用来定义普通文本的变量，PCDATA（parsed character data）是被解析的字符数据。可把字符数据想象为 XML 元素起始标记与结束标记之间的文本。PCDATA 指会被解析器解析的文本。这些文本将被解析器检查实体以及标记，文本中的标记会被当作标记来处理，而实体则会被展开。不过，被解析的字符数据不应当包含“&”“<”或者“>”字符，而需要使用“&”“<”以及“>”，以实体引用的方式来分别替换它们。CDATA（character data）的意思是字符数据。CDATA 是不会被解析器解析的文本，在这些文本中标记不会被当作标记来对待，其中的实体也不会被展开。DTD 分别对这些构建模块进行定义，以此来验证 XML 文档。

3.1.1　DTD 元素

当使用一个 DTD 来定义 XML 文档的内容时，必须声明会在文档中出现的每一个元素。另外，DTD 声明中可以包括可选元素，即可能会也可能不会出现在 XML 文档中的元素。元素声明包括 3 个基本组成部分：ELEMENT 声明、元素名、元素内容模型，格式如下。

<!ELEMENT　element-name　category>

或者

<!ELEMENT　element-name (element-content)>

ELEMENT 表示要声明一个元素；元素名称（element-name）决定了该元素出现在 XML 文档中时必须使用的名称，包括命名空间前缀；元素的内容模型定义了元素中允许出现的内容。一个元素可以包含子元素、文本、子元素和文本的组合，也可以是空的。

1．声明包含子元素的元素（序列、选择、序列选择混合）

带有一个或多个子元素的元素通过圆括号中的子元素名进行声明，格式如下。

<!ELEMENT element-name
(child-element-name)>

或

<!ELEMENT element-name
(child-element-name,child-element-name,……)>

例如，文档中有一个<contact>元素并且该元素包含一个<name>子元素，可用下面的代码声名。

```
<!ELEMENT contact (name)>
```

有时，<contact>元素需要包含多个子元素，如包含<name>、<location>、<phone>、<knows>和<description>子元素，声明代码如下。

```
<!ELEMENT contact (name, location, phone, knows, description)>
```

内容模型中指定的每个元素在 DTD 中也必须有自己的定义。因此，上面例子中的元素<name>、<location>、<phone>、<knows>和<description>都必须在 DTD 中完成声明。处理器需要此信息，以便知道如何处理每一个遇到的元素。当然子元素也能有子元素。<contact>元素的完整声明如下。

```
<!ELEMENT contact (name,location,phone,knows,description)>
  <!ELEMENT name     (#PCDATA)>
  <!ELEMENT location   (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
  <!ELEMENT knows  (#PCDATA)>
  <!ELEMENT description (#PCDATA)>
```

当子元素按照由逗号分隔开的序列进行声明时，这些子元素必须按照相同的顺序出现在文档中。当具有 DTD 的 XML 文档发生以下 3 种情况时，XML 的解析将出现错误。

- XML 文档丢失序列中的子元素之一。
- 文档中包含的子元素多于序列中的子元素。
- 子元素出现在文档中的顺序不同于序列中的顺序。

考虑这样一种情形，假设上例中的<location>元素具有两个子元素<address>和<GPS>，而标明“location”只需要一个“address”或“GPS”，即二者只取其一，那么要如何在 DTD 中声明<location>元素呢？应该采用选择的声明方式，用管道分隔符（|）来分隔子元

素，声明代码如下。

```
<!ELEMENT location (address | GPS)>
```

这个声明使得<location>元素包含一个<address>或一个<GPS>元素。如果<location>元素是空的，或者它包含一个以上的这些元素，就会引发解析错误。

假如有如下的声明代码：

```
<!ELEMENT contact (name, location, phone, knows, (description | note))>
```

表明<contact> 元素必须包含 <name> 元素、<location> 元素、<phone> 元素、<knows> 元素，以及非 <description> 元素即<note>元素。这种声明方式称之为序列选择混合，在 DTD 元素声明中是较为常见的。

在 DTD 中声明元素时，有时需要考虑元素在文档中出现的频率，声明方式如下。

1）声明只出现 1 次的元素

格式如下。

```
<!ELEMENT element-name (child-name)>
```

例如：

```
<!ELEMENT contact (description)>
```

上面的实例声明：子元素<description>必须出现 1 次，并且在<contact>元素中只出现 1 次。

2）声明最少出现 1 次的元素

格式如下。

```
<!ELEMENT element-name (child-name+)>
```

例如：

```
<!ELEMENT contact (description+)>
```

上面实例中的“+”号声明：子元素<description>必须在<contact>元素中出现 1 次或几次。

3）声明出现 0 次或多次的元素

格式如下。

```
<!ELEMENT element-name (child-name*)>
```

例如：

```
<!ELEMENT contact (description*)>
```

上面实例中的“*”号声明：子元素<description>可以在<contact>元素中出现 0 或多次。

4）声明出现 0 次或 1 次的元素

格式如下。

```
<!ELEMENT element-name (child-name?)>
```

例如：

```
<!ELEMENT contact (description?)>
```

上面实例中的“？”号声明：子元素<description>可以在<contact>元素中出现 0 或 1 次。

表 3-1 是对上述符号的总结和说明。

表 3-1　声名元素出现次数的符号总结

符　　号	代表子元素出现的次数
?	不出现或只出现 1 次
*	不出现或可出现多次
+	必须出现 1 次以上
无符号	必须出现且只能出现 1 次

例如，<!ELEMENT reference (book*, newspaper+, magazine?, website)>这个声明中，<book>元素在 XML 文档中可以不出现或出现多次，<newspaper>元素必须出现 1 次以上，<magazine>元素可以不出现或只出现 1 次，而<website>必须出现而且只能出现 1 次。

2．声明空元素

XML 文档中的某些元素可能具有内容，也可能不具有内容，有的元素可能从不需要包含内容，这时需要定义一个空元素。空元素是用类别关键字 EMPTY 来声明的，格式如下。

```
<!ELEMENT element-name EMPTY>
```

例如：

```
<!ELEMENT br EMPTY>
```

在 XML 文档中，
元素不包含任何内容，其在 XML 文档中的表现形式为
。不要用 EMPTY 关键字来声明可能包含内容的元素。

3．声明纯字符数据的元素

纯字符数据的元素用带圆括号的#PCDATA 来声明，格式如下。

```
<!ELEMENT element-name (#PCDATA)>
```

例如：

```
<!ELEMENT from (#PCDATA)>
```

这个声明表示<from>元素的内容只能是文本。

4．声明带混合内容模型的元素

假设要设计一个管理客户信息的 XML 文档，希望在 XML 文档中为每个联系人添加描述，因此创建了一个元素<description>，代码如下。

```
<description>Joe is a developer and author for <title>Beginning XML</title>,
now in its <detail>5th  Edition</detail></description>
```

这个例子中，<description>元素包含的文本内容中穿插了子元素<title>和<detail>。在

DTD 中对<description>元素的定义应采用混合内容声明方式。声明 DTD 中混合内容模型只有一种方式——在添加元素时使用选择的方式。这意味着，内容模型中的每个元素之间必须由管道分隔符（|）来分隔，代码如下。

```
<!ELEMENT description (#PCDATA | title | detail)*>
```

上面的例子声明了<description>元素，并使用选择的方式来描述内容模型，即用管道分隔符来分隔每个元素。

当混合内容模型中包括元素时，关键字#PCDATA 必须首先出现在选项列表中。内容模型括号外的“*”符号标明了子元素<title>和<detail>可以在<contact>元素中出现 0 或多次，同时也允许有任意数量的文本在<contact>元素中出现。子元素和文本可以以任意顺序出现。

总之，声明带混合内容模型元素时，必须遵循以下 4 个原则。

（1）必须使用选择机制（管道分隔符（|））来分隔元素。

（2）关键字#PCDATA 必须首先出现在元素的列表中。

（3）必须没有内在的内容模型。

（4）如果有子元素，则“*”指示符必须出现在模型的末端。

5．声明带有任意内容的元素

使用关键字 ANY 声明带有任意内容的元素。例如：

```
<! ELEMENT description ANY >
```

ANY 关键字表示文本（PCDATA）和/或在 DTD 中声明的任何元素都可以在<description>元素的内容中使用，并且它们能够以任何顺序和任意次数出现。但是，ANY 关键字不允许在<description>元素中包含未在 DTD 内声明的元素。

3.1.2 DTD 属性

属性声明在很多方面与元素声明类似，除了要应用不同的关键字以外。在 DTD 中，属性通过关键字 ATTLIST 来进行声明。基本格式如下。

<!ATTLIST element-name attribute-name attribute-type default-value>

其中，ATTLIST 表明要声明属性，element-name 决定了与属性关联的元素的名称，attribute-name 声明了属性出现在 XML 文档中时必须使用的名称，attribute-type 定义属性的类型，default-value 定义属性的默认值。例如：

```
<!ATTLIST contacts source CDATA #IMPLIED>
```

这个例子中属性的名称为 source。这个 source 属性可以包含字符数据——CDATA 关键字用来定义属性的类型。最后，该声明表明该属性没有默认值。使用#IMPLIED 关键字表明这个属性可以不出现在<contact>元素中。该属性声明的第 3 部分是被称为属性值声明，它控制 XML 解析器如何处理属性值。

每个属性的声明都需要 3 个信息：属性名称、属性类型和属性值。

1．属性名称

除了符合基本的 XML 命名规则，用户必须确保给定的元素属性列表中没有重复的属性名称。属性名称决定了该属性出现在 XML 文档中时必须使用的名称，包括任何命名空间的前缀。

2．属性类型

当声明属性时，用户可以指定属性类型以便通知处理器该如何处理出现在值中的数据。表 3-2 为属性类型的选项及说明。

表 3-2　属性类型的选项及说明

属性类型	描　　述
CDATA	表示属性值是字符数据，默认类型。注意这与元素声明中的 PCDATA 关键字略有不同。在 CDATA 中解析器可以忽略某些保留字符
(en1|en2|…)	列出该属性的取值范围。一次只能有一个属性值能够赋予属性
ID	该属性在 XML 文档中是唯一的
IDREF	该属性值参考了另外一个 ID 属性（属性值中不允许有空格）
IDREFS	该属性值参考了多个 ID 属性，各 ID 属性的值用空格隔开
NMTOKEN	属性值只能由英文字母、数字、句号（.）、下划线（_）、冒号（:）、连字符（-）这些符号组成（不能是中文）
NMTOKENS	属性值能够由多个 NMTOKEN 组成，每个 NMTOKEN 之间用空格隔开
ENTITY	表示该属性的设定值是一个外部的实体，如一个图片文件
ENTITIES	属性值包含了多个外部 ENTITY，不同的 ENTITY 之间用空格隔开
NOTATION	属性值是在 DTD 中声明过的 NOTATION（如声明用什么应用软件解读某些二进制文件，比如图片）
xml:	属性值是一个预定义的 XML 值

属性类型的用法，见以下各示例。

1）CDATA

DTD 声明：

```
<!ATTLIST person name CDATA #REQUIRED>
```

XML：

```
<person name="Tom" />
```

2）(en1|en2|…)

DTD 声明：

```
<!ATTLIST person gender(male|female) #REQUIRED>
```

XML：

```
<person gender="male" />
```

3）ID

DTD 声明：

```
<!ATTLIST employee
    empID ID #REQUIRED
    name CADA #REQUIRED>
```

XML：

```
<employee empID="Z001" name="张三" />
employee empID="Z002" name="李四" />
```

4）IDREF/IDREFS

DTD 声明：

```
<!ELEMENT family (person+)>
<!ELEMENT person EMPTY>
<!ATTLIST person
relID ID #REQUIRED
parentID IDREFS #IMPLIED
name CDATA #REQUIRED>
```

XML：

```
<family>
   <person relID="P_1" name="father">
   <person relID="P_2" name="mother>
   <person relID="P_1 P_2" name="son">
</family>
```

5）NMTOKEN/NMTOKENS

DTD 声明：

```
<!ELEMENT poems (title, content)>
<!ELEMENT title (#PCDATA)>
<!ATTLIST title author NMTOKEN #REQUIRED>
<!ELEENT content (#PCDATA)>
```

XML:

示例 1

```
<pomes>
   <title author="李白"></title>
   <content>
      床前明月光，疑是地上霜。
   </content>
</pomes>
```

示例 2

```
<pomes>
   <title author="libai"></title>
   <content>
      床前明月光，疑是地上霜。
   </content>
</pomes>
```

示例 1 在解析时会发生错误，因为属性 author 的值是中文。示例 2 是正确的。

NMTOKENS 与 NMTOKEN 属性类型类似，只是它包含多个由空格分隔的字符。

6）ENTITY/ENTITIES

DTD 声明：

```
<!ELEMENT library (number, img)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT img EMPTY>
<!ATTLIST img src ENTITY #REQUIRED>
<!NOTATION gif SYSTEM "gifprocess.exe">
<!ELEMENT pic SYSTEM "pic1.gif" NDATA gif>
```

XML：

```
<library>
  <number>A001</number>
  <img src="pic" />
</library>
```

NOTATION 标识要链接到 XML 文档的外部数据项的格式。NOTATION 声明能够说明格式的名称以及相关的外部处理器。解析器可以根据声明将自己不能识别的数据交给外部处理器处理。

语句<!NOTATION gif SYSTEM “gifprocess.exe”>表明 GIF 格式文件由外部的 gifprocess.exe 程序处理。

语句<!ELEMENT pic SYSTEM “pic1.gif” NDATA gif>用于声明实体，NDATA 关键字说明实体的数据有相应的 NOTATION 类型。

以上代码将 GIF 文件 pic1.gif 与 img 元素相关联。对于经常要重用的实体，这种方法非常值得推荐。但是，假如实体的值需要频繁修改，这种方法就不可取了。

为了将 ENTITY 作为属性类型使用，需要执行 4 个步骤。前 3 个步骤都是在 DTD（外部 DTD 或内部子集)中进行声明。第 4 个步骤涉及特定的文档实例。下面将这 4 个步骤总结如下。

（1）声明元素和 NOTATION。

（2）为元素声明类型为 ENTITY 的属性。

（3）声明一个或多个实体，以便在属性中使用。

（4）在文档中创建元素类型实例，将实体名称作为属性值。

ENTITIES 类型的属性值与 ENTITY 类似，不同的是它可以包含多个由空格分开的实体。

DTD 声明：

```
<!ELEMENT library (number, img)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT img EMPTY>
<!ATTLIST img src ENTITIES #REQUIRED>
<!NOTATION gif SYSTEM "gifprocess.exe">
```

```
<!ELEMENT pic SYSTEM "pic1.gif" NDATA gif>
<!ELEMENT report SYSTEM "report1.gif" NDATA gif>
```

XML：

```
<library>
  <number>A001</number>
  <img src="pic report" />
</library>
```

7）NOTATION

DTD 声明：

```
<!ELEMENT Image EMPTY>
<!NOTATION jpg SYSTEM "1.exe">
<!NOTATION gif SYSTEM "2.exe">
<!ATTLIST Image type NOTATION (gif|jpg) "gif">
```

XML：

```
<Image type="jpg" />
```

在以上声明中，Image 元素可以有一个名为 type 的属性，它是 NOTATION 类型的。该属性可选的值有 gif 和 jpg。如果元素实例没有定义 type 属性，解析器会假设该属性设置为缺省值 gif。然而，在上述实例中，值 jpg 覆盖了缺省值。

在上面的 DTD 声明示例中，关键字 NOTATION 声明了 jpg 和 gif。NOTATION 主要用来表明文档中需要来自外部源的数据，而该数据 XML 本身是不能进行解析的，比如各种格式的二进制文件（图形文件、声音文件等），需要外部的应用程序来进行处理。

NOTATION 声明的格式如下。

```
<!NOTATION NAME ExternalID>
```

需要注意的是，NAME 必须由字母、数字、句号（。）、下划线（_）、连字符（-）或冒号（：）组成，并且第一个字符必须为字母或者下划线。

下面的例子显示了定义 GIF 图像作为不解析的外部内容。

```
<!NOTATION gif SYSTEM "iexplore">
    <!ENTITY logo SYSTEM "http://www.wrm.com/picturecategory/picwrm.gif"
    NDATA gif>
<!-- 此处的 NDATA 表示 XML 不解析该数据 -->
<!ELEMENT pic EMPTY>
<!ATTLIST pic loc ENTITY #REQUIRED>
```

然后，在具体的文档中编写代码：<pic loc= “&logo;” />。根据 DTD 的定义，loc 属性值是一个不解析的实体。解析器根据 DTD 定义知道这一点，便不对 loc 属性值进行解析，也不会像解析实体一样把它包括到 XML 文档里面，同时 XML 解析器将通知 iexplore.exe 该引用的存在。

3．属性值

在每个属性声明中必须指定属性值将以怎样的状态出现在文档中。表 3-3 为属性值的选项及说明。

表 3-3　属性值选项及说明

值	说　明
Value	属性的默认值
#REQUIRED	属性值是必需的
#IMPLIED	属性值是可选的
#FIXED value	属性值是固定的

属性值选项的用法，见以下各示例。

1）默认值

DTD 声明：

```
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
```

XML：

```
<square width="100" />
```

在上面的例子中，square 元素定义为含有 CDATA 类型的 width 属性的空元素。如果没有指定 width 值，那它默认为 0。

2）#REQUIRED

属性声明格式如下。

<!ATTLIST element-name attribute_name

attribute-type #REQUIRED>

DTD 声明：

```
<!ATTLIST person number CDATA #REQUIRED>
```

正确的 XML 表述：

```
<person number="5677" />
```

错误的 XML 表述：

```
<person />
```

3）#IMPLIED

属性声明格式如下。

<!ATTLIST element-name attribute-name

attribute-type #IMPLIED>

DTD 声明：

```
<!ATTLIST contact fax CDATA #IMPLIED>
```

XML：

```
<contact fax="555-667788" />
```

或

```
<contact />
```

4）#FIXED

属性声明格式如下。

<!ATTLIST element-name attribute-name
attribute-type #FIXED "value">

DTD 声明：

```
<!ATTLIST sender company CDATA #FIXED "Microsoft">
```

正确的 XML：

```
<sender company="Microsoft" />
```

不正确的 XML：

```
<sender company="W3Schools" />
```

3.1.3 DTD 实体

有时候可能需要在多个文档中调用同样的内容，比如公司名称、版权声明等，为了避免重复输入这些内容，可以声明一个实体来表示这些内容，这样在文档中只需要引用这个实体即可。当 XML 处理器对这个文档进行分析处理后，引用实体的位置会被实体的内容所替换。就实体的引用方式来说，可分为内部实体和外部实体。就实体的类型来分，可分为一般实体（general entity）和参数实体（parameter entity）。一般实体是在文档内容中使用的实体，而参数实体则是在 DTD 中使用的已分析实体。不管是一般实体还是参数实体，都使用 ENTITY 关键字来声明。

1．内部实体

内部实体在 XML 文档内部定义，实体的内容在声明中指定。内部实体都是已分析的实体，它们没有单独的物理存储对象。

语法格式：

```
<!ENTITY 实体名 "实体值">
```

内部实体示例如下。

DTD 声明：

```
<!ENTITY writer "Donald Duck">
<!ENTITY copyright "Copyright WRM.">
```

XML：

```
<author>&writer; &copyright;</author>
```

运行结果：

```
<author>Donald Duck Copyright WRM.</author>
```

2．外部实体

外部实体在单独的（外部）文件中定义，外部实体可以是已分析实体，也可以是未分

析实体。

语法格式：

```
<!ENTITY 实体名 SYSTEM "URI/URL">。
```

关键字 SYSTEM 表明这是一个私有的外部实体，后面的 URI/URL 称为该实体的系统标识符，用于指定外部文件的位置。除了关键字 SYSTEM 外，也可以使用 PUBLIC 关键字来声明公共的外部实体，其语法格式与应用关键字 SYSTEM 的语法格式类似，为<!ENTITY 实体名 PUBLIC“URI/URL”>。外部实体示例如下。

DTD 声明：

```
<!ENTITY writer SYSTEM "http://www.w3schools.com/entities/entities.xml">
<!ENTITY copyright SYSTEM
  "http://www.w3schools.com/entities/entities.dtd">
```

XML：

```
<author>&writer; &copyright;</author>
```

这里表示用文档 http://www.w3schools.com/entities/entities.xml 来表示实体 writer 的具体内容，用文档 http://www.w3schools.com/entities/entities.dtd 来表示实体 copyright 的具体内容。需要指出的是，这里的 entities.xml 文档必须是一个格式良好的 XML 文档。

3．实体类型

有两种类型的实体：一般实体（又称普通实体）和参数实体。它们都可以定义为内部实体，也可以用关键字 SYSTEM 或 PUBLIC 定义为外部实体。实体的定义必须出现在引用之前，而且要注意嵌套正确，不能出现循环引用的情况。在 DTD 中，这两种类型的实体都得到了广泛的应用。表 3-4 展示了实体的类型和用法。

表 3-4　实体类型及用法

类　型		一 般 实 体	参 数 实 体
使用场合		用在 XML 文档中	只用在 DTD 中元素和属性的声明中
声明方式	内部	<!ENTITY 实体名"文本内容">	<!ENTITY % 实体名"文本内容">
	外部	<!ENTITY 实体名 SYSTEM"外部文件 URL 地址">	<!ENTITY % 实体名 SYSTEM"外部文件 URL 地址">
引用方式		&实体名;	%实体名;

① 一般实体

所谓一般实体，就是该实体在具体的文档中使用，示例如下。

```
<?xml version = "1.0" encoding="UTF-8" standalong="yes"?>
<!DOCTYPE file[
   <!ELEMENT file ANY>
   <!ELEMENT movie EMPTY>
   <!ATTLIST movie source ENTITY #REQUIRED>
   <!ENTITY BladeRunner SYSTEM "dvds/BR/br.move">
]>
<file>
```

```
    <movie source="&BladeRunner;" />
</file>
```

这里用关键字 SYSTEM 定义了一个外部一般实体，用文档 dvds/BR/br.move 来表示实体 BladeRunner 的具体内容。如果将<!ENTITY BladeRunner SYSTEM “dvds/BR/br.move”>改为

<!ENTITY BladeRunner “blue house”>，则为定义一个内部一般实体。

② 参数实体

所谓参数实体，就是该实体实际上不在具体文档中使用，而是在 DTD 中使用。比如可以定义一个如下的参数实体：

```
<!ENTITY % address "street,city,zip,country">
```

然后在 DTD 内部通过%address;来引用它。具体例子如下。

```
<!ELEMENT contact (name,phone,%address;)>
```

这里定义了一个内部参数实体。在 XML 处理器分析时，解析器将用“street,city,zip,country”来替代%address。需要指出的是，如果想在 DTD 标记声明中引入参数实体，那么就必须用外部 DTD，这是因为在内部 DTD 中引入参数实体时不能写在标记声明内部。要解决这个问题，可以把这个内部 DTD 写成一个外部 DTD 文档——可建一个 DTD 文档，把原 XML 文档中的内部 DTD 复制到外部 DTD 文档中去。

外部参数实体和内部参数实体的关系与外部一般实体和内部一般实体的关系一样，也就是说，实体的内容不使用两个引号来标明，而使用一个外部的 URL 来表示，比如：

```
<!ENTITY % address SYSTEM "http://somewebsite/somecategory/something.xml">
```

然后就可以在 DTD 内部通过%address;来引用它。

3.2 应用 DTD

在 XML 文档中，通过包含文档类型声明来建立当前文档和 DTD 的关联。当进行有效性验证的 XML 处理器读到该声明时，它获取 DTD 并根据其中定义的规则对文档进行检验。文档类型声明必须位于 XML 声明之后，且在根元素之前，不过在 XML 声明和文档类型声明之间可以插入注释和处理指令。用户可以直接在 XML 文档中定义 DTD（内部 DTD），也可以通过 URI 引用外部的 DTD 文件（外部 DTD），或同时采用这两种方式。

1．内部DTD

最简单的使用 DTD 的方法是在 XML 文档的序言部分加入一个 DTD 描述，加入的位置是在 XML 声明之后。一个包含 DTD 的 XML 文档的结构如下。

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!DOCTYPE 根元素名[
    元素描述
]>
```

```
文件体……
```

这样，用户就定义了一个带有内部 DTD 的 XML 文档。下面是一个带有内部 DTD 的 XML 文档实例。

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!DocType customers[
<!Element customers(customer*)>
<!Element customer(firstname,lastname,homephone,notes)>
<!Element firstname(#PCDATA)>
<!Element lastname(#PCDATA)>
<!Element homephone(#PCDATA)>
<!Element notes(#PCDATA)>
<!ATTLIST customer customerid CDATA#Required>]>
<customers>
  <customer customerid="1">
    <firstname>John</firstname>
    <lastname>Cranston</lastname>
    <homephone>(445) 269-9857</homephone>
    <notes>
      <![CDATA[He registed as our member since 1990. John has nice credity.
      He is a member of Custom International.]]>
   </notes>
  </customer>
  <customer customerid="2">
    <firstname>Annie</firstname>
    <lastname>Loskar</lastname>
    <homephone>(445) 269-9482</homephone>
    <notes>
      <![CDATA[Annie registed as our member since 1984. He became our VIP
      customer in 1996.]]>
    </notes>
  </customer>
  <customer customerid="3">
    <firstname>Bernie</firstname>
    <lastname>Christo</lastname>
    <homephone>(445) 269-3412</homephone>
    <notes>
      <![CDATA[Bernie registed as our member since June 2010. He is a new
     member.]]>
    </notes>
  </customer>
<customer customerid="4">
    <firstname>Ernestine</firstname>
    <lastname>Borrison</lastname>
    <homephone>(445) 269-7742</homephone>
    <notes>
      <![CDATA[Ernestine registed as our member since Junl 2010. She is a
     new member.]]>
    </notes>
  </customer>
</customers>
```

上面的例子中，文档类型声明由“<!”开始，后面紧跟一个关键字 DocType，然后是文档根元素的名字，接下来是标记声明块，标记声明放在中括号“[”“]”之间，由一个

或多个标记声明构成，最后由“>”结束。

在 XML 文档中定义 DTD 的方式比较直观，修改较方便，而且不用担心 XML 处理器找不到 DTD。但是它也有一些缺点，如导致文档本身的长度增加，若多个 XML 文档要共用一个 DTD，就需要在每个文档中加入相同的 DTD 等。为克服内部 DTD 的缺点，需要引入外部 DTD。

2．外部DTD

一个 DTD 既可以是内部的，包含在一个“形式良好”的 XML 文档中（standalone=“yes”），也可以是外部的，作为一个外部文件被引用（standalone=“no”）。外部 DTD 的好处是，只要写一个 DTD 文件，就可以方便高效地被多个 XML 文档所引用。这样做不仅简化了输入工作，还保证当需要对 DTD 做出改动时，不用一一去修改每个引用它的 XML 文档，而只要修改一个公用的 DTD 文件就足够了。不过需要注意，如果 DTD 的改动不是“向后兼容”的，那么解析器解释原先写的那些 XML 文档就可能会出现问题。

要引用一个外部 DTD，必须修改 XML 声明和 DOCTYPE 声明。XML 声明中必须说明这个文件不是自成一体的，即 standalone 属性的属性值是 no。代码如下。

```
<?xml version="1.0" encoding="UFT-8" standalone="no"?>
```

在 DOCTYPE 声明中，应该加入 SYSTEM 属性。语法格式如下。

<!DOCTYPE 根元素名 SYSTEM “外部 DTD 文件的 URL”>

例如：

```
<!DOCTYPE contect SYSTEM "http://somewebsite/somecategory/something.dtd">
```

这里的 URL 是一个绝对路径。它也可以是一个相对路径，如：

```
<!DOCTYPE contect SYSTEM "something.dtd">
```

说明这个 DTD 文件和引用它的 XML 文件在同一个目录下。如果 DTD 文件在 XML 文件的父目录的子目录下，表示为：

```
<!DOCTYPE contect SYSTEM "../dtds/something.dtd">
```

使用这种方法，用户可以方便地把 DTD 文件从 XML 文档中分离出来。

仍然回到前面那个包含客户信息的 XML 文档，如果使用外部 DTD，其内容变化如下。

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE customers SYSTEM "Customers.dtd">
<customers>
  <customer customerid="1">
    <firstname>John</firstname>
    <lastname>Cranston</lastname>
    <homephone>(445) 269-9857</homephone>
    <notes>
      <![CDATA[He registed as our member since 1990. John has nice credity.
      He is a member of Custom International.]]>
   </notes>
  </customer>
  <customer customerid="2">
```

```
    <firstname>Annie</firstname>
    <lastname>Loskar</lastname>
    <homephone>(445) 269-9482</homephone>
    <notes>
      <![CDATA[Annie registed as our member since 1984. He became our VIP
      customer in 1996.]]>
    </notes>
  </customer>
  <customer customerid="3">
    <firstname>Bernie</firstname>
    <lastname>Christo</lastname>
    <homephone>(445) 269-3412</homephone>
    <notes>
      <![CDATA[Bernie registed as our member since June 2010. He is a new
      member.]]>
    </notes>
  </customer>
<customer customerid="4">
    <firstname>Ernestine</firstname>
    <lastname>Borrison</lastname>
    <homephone>(445) 269-7742</homephone>
    <notes>
      <![CDATA[Ernestine registed as our member since Junl 2010. She is a
      new member.]]>
    </notes>
  </customer>
<customer customerid="5">
    <firstname>Ernestine</firstname>
    <lastname>Borrison</lastname>
    <notes>
      <![CDATA[Ernestine registed as our member since Junl 2010. She is a
      new member.]]>
    </notes>
    <homephone>(445) 269-7742</homephone>
  </customer>
</customers>
```

对应的 DTD 内容如下。

```
<?xml version="1.0" encoding="utf-8"?>
<!ELEMENT customers (customer*)>
<!ELEMENT customer (firstname,lastname,homephone,notes)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT homephone (#PCDATA)>
<!ELEMENT notes (#PCDATA)>
<!ATTLIST customer customerid CDATA #REQUIRED>
```

在声明文档类型时，用关键字 SYSTEM 来指出外部 DTD 文件的位置。使用 SYSTEM 关键字来声明的语法格式如下。

<!DOCTYPE 根元素的名字 SYSTEM“外部 DTD 文件的 URI”>

外部 DTD 文件的 URI 可以是相对 URI，也可以是绝对 URI，本示例中使用的是相对 URI。

在上面的 DTD 示例中，所有的关键字都是大写的。在 DTD 中定义的元素和属性名大

小写可以是任意指定的，但因为 XML 文档是大小写敏感的，所以一旦给一个元素或属性命名，那么在整个文档中要使用相同的大小写。

3.3 DTD 的局限性

DTD 有效地推动了 XML 的发展。然而，由于它是一个较古老的技术，其局限性也是较为明显的，主要表现在以下几个方面。

- 语法结构

DTD 的语法与 XML 语法完全不同，使用 DOM，XPath 和 XSL 无法处理，为自动化文档处理带来不便。

- 数据类型

DTD 提供的数据类型有限，有时候无法满足行业的需要。DTD 数据类型不能自由扩充，不利于 XML 数据交换场合验证。

- 文档结构

DTD 中，所有元素、属性都是全局的， 无法声明仅与上下文位置相关的元素或属性。

- 名称空间

DTD 中没有名称空间的概念，不直接支持名称空间。

尽管 DTD 存在着局限性，但 DTD 是非常基础性的概念，它有助于理解其他模式，以及如何利用 XML 通过标准的方式交换文档。

习题

1．请根据如下的 DTD 声明，编写一个带有内部 DTD 的 XML 文档。

```
<!DOCTYPE BookList [
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
<!ELEMENT PubDate (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Book (Title,Author,Publisher,PubDate,ISBN)>
<!ELEMENT BookList (Book)*>
<!ATTLIST Book Category CDATA "计算机"]>
```

2．为上题中实现的 XML 文档的<Author>元素添加一个 Gender（性别）属性声明，该属性应该允许有两个可能的值：男性和女性，并确保属性是必需的。

3．请根据如下的 DTD 声明，编写一个带有内部 DTD 或外部 DTD 的 XML 文档。

```
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
<!ELEMENT PubDate (#PCDATA)>
```

```
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Book (Title,Author,Publisher,PubDate,ISBN)>
<!ELEMENT BookList (Book)*>
<!NOTATION jpg SYSTEM "image/jpeg">
<!ENTITY Photo1 SYSTEM "photo1.jpg" NDATA jpg>
<!ENTITY Photo2 SYSTEM "photo2.jpg" NDATA jpg>
<!ATTLIST Book Category CDATA "计算机"
          BookID ID #REQUIRED
          Photo ENTITY #IMPLIED>
```

4．根据读者自己的实际情况，应用 DTD 和 XML 创建一个通讯录。要求：使得每个联系人（contact）可具有一个或多个电话号码，可具有 0 个或多个电子邮件，可具有 0 个或 1 个的网站。