第 3 章

文档类型定义(DTD)

本章目标

- 掌握 DTD 的基本结构;
- 掌握 DTD 在 XML 文档中的引用;
- 掌握 DTD 中元素和属性的声明。

第 2 章介绍了 XML 文档的基本语法,利用这些语法可以编写出格式良好的 XML 文档。然而,在实际开发中,经常会遇到这样一个问题:如何与他人交流自己设计的结构?

目前很多主流的浏览器已经提供了对 XML 的支持,但这种支持仅限于对 XML 内容的显示。如果所开发的程序还包含了新的词汇表,而通过这些新的词汇就可以明白作者的设计结构,那么作为 XML 词汇表的设计者,就必须通过某种通用的方式来说明词汇表的语法规则。为此,XML 1.0 提供了一种机制——文档类型定义(Document Type Definition,DTD),并将其作为规范的一部分。

DTD 将带来以下优越性:通过创建 DTD,能够准确地定义词汇表。所有词汇表规则都包含在 DTD 中。这样一来在 XML 中就不必像在 HTML 中一样,所有使用的标记都必须是预定义好的,标记的使用规则也是固定的。在 XML 中,就可以通过 DTD 定义自己的元素、属性及这些元素和属性的使用规则。凡是未在 DTD 中出现的规则都不属于词汇表的一部分。只要在文档实例中引入相关 DTD,解析器就可以利用 DTD 验证此文档实例的有效性,并将其中的内容与文档实例进行比较。另外,XML 创作工具也可以通过类似的方式使用 DTD。选择了 DTD 后,创作工具就能够实施该 DTD 中的规则,并根据 DTD 中说明的结构,仅允许用户在文档实例中添加 DTD 规则允许的元素和属性。

在本章,我们将创建有效的 XML 文档实例,它不仅遵守 XML 语法规则,而且符合 DTD 中的相关规则。

3.1 DTD 文档结构

DTD 可以存在于 XML 文档的内部,也可以独立地存在于 XML 文档的外部。不管是内部 DTD 还是外部 DTD,都能对 XML 文档中的词汇进行定义(这里的定义侧重于词

汇的使用规则)。简单地说,XML 文档主要由元素和相应的属性构成。所以 DTD 必须对 XML 文档中的元素和属性进行定义。而元素和属性的相关声明则构成了 DTD 文档的主要结构。当然,DTD 也可以对实体和其他项进行声明。

下面来看一个含有简单 DTD 的 XML 文档,如例 3-1 所示。

【例 3-1】 含有内部 DTD 的 XML 文档。

```
[1]  <?xml version="1.0" encoding="GB2312" standalone="yes"?>
[2]  <!DOCTYPE 联系人列表[
[3]  <!ENTITY content "某公司部分联系人信息">
[4]  <!ELEMENT 联系人列表(说明,联系人)>
     <!ELEMENT 说明(#PCDATA)>
     <!ELEMENT 联系人(姓名,ID,公司,EMAIL,电话,地址)>
     <!ELEMENT 姓名(#PCDATA)>
     <!ELEMENT ID(#PCDATA)>
     <!ELEMENT 公司(#PCDATA)>
     <!ELEMENT EMAIL(#PCDATA)>
     <!ELEMENT 电话(#PCDATA)>
     <!ELEMENT 地址(街道,城市,省份)>
     <!ELEMENT 街道(#PCDATA)>
     <!ELEMENT 城市(#PCDATA)>
     <!ELEMENT 省份(#PCDATA)>
[5]  <!ATTLIST 电话 类别 CDATA "固定电话">
[6]  ]>
[7]  <联系人列表>
      <说明>&content;</说明>
      <联系人>
          <姓名>张三</姓名>
          <ID>001</ID>
          <公司>A公司</公司>
          <EMAIL>zhang@ aaa.com</EMAIL>
          <电话 类别="固定电话">(010)62345678</电话>
          <地址>
              <街道>五街 1234 号</街道>
              <城市>北京市</城市>
              <省份>北京</省份>
          </地址>
      </联系人>
     </联系人列表>
```

以上 XML 文档包括一个完整的内部 DTD 文档,可以看出[2]~[6]部分是在例 2-1 的基础上明显增加的部分,此部分构成了一个内部 DTD。

- [1]是 XML 声明。
- [2]是内部 DTD 的开始,也是在 XML 文档内部引用 DTD 的方式。
- [3]是实体的声明,实体名为“content”,所代表的内容是“某公司部分联系人信息”。
- [4]是对 XML 文档中要用到的元素的声明。

- [5]是对元素的属性的声明。
- [6]是内部 DTD 的结束标记。
- [7]以后是按照内部 DTD 的规则所写出的一个 XML 文档实例的主体部分。

从例子中不难看出,DTD 的文档结构不是一般的 XML 文档结构,它有自己独立的语法。有了 DTD 我们就可以定义自己的标记的语法规则,从而构建自己特有的置标语言(置标语言就是标记按照一定的规则合在一起而构成的)。

3.2　DTD 中的元素声明

所有有效的 XML 文档中使用的元素都必须在 DTD 中先进行声明。声明的内容包括元素的名称、元素可能包含的内容和所具有的属性,以及元素在 XML 文档中出现的先后顺序、元素与元素之间的关系等。DTD 对 XML 文档中的元素的这种声明称为元素类型声明(ETD)。这样就可以在 DTD 中对元素所能包含的内容进行较为精确的限制,从而控制文档的逻辑结构。

3.2.1　元素声明的语法

从例 3-1 中可以看出,元素类型声明最基本的是要声明元素的名称和元素的内容类型。元素类型声明的基本语法如下:

```
<!ELEMENT elementName elementContentModel>
```

其中,"<!"表示一条指令的开始。ELEMENT 是关键字,表明是在声明一个元素,作为关键字 ELEMENT 一定要大写。elementName 用来指定元素的名称,在使用时用具体的元素名称替代。而 elementContentModel 用来描述元素可能包含的内容,即指定元素内容模型(Element Content Model,ECM)。元素的内容可能是只包含文本的简单内容,也可能包含其他的子元素,还可能既有文本又有子元素。通过元素内容模型(ECM),可以指定元素的内容类型及其他信息,例如子元素出现的顺序及可以出现的次数等。

3.2.2　控制元素的内容

根据元素所包含的内容,即通过元素内容模型(ECM)可以将元素内容类型归纳为以下 6 种。

- 简单类型:元素内容只能是文本字符内容,且没有属性。
- 包含简单内容的复杂类型:元素内容只能是文本字符内容,但可以有属性。
- 包含复杂内容的复杂类型:元素内容可以包含子元素,也可以有属性。
- 混合内容类型:元素内容既可以有文本字符内容,也可以包含子元素,同时还可以有属性。
- 空内容类型:元素内容为空,但可以有属性,此类元素一般都带有属性。
- 任何内容类型:元素内容不受限制,也可以有属性。

1. 简单类型声明

简单类型表示元素只能含有文本字符,声明语法如下:

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

elementName 是要声明的元素的名称，在具体应用时，用实际要定义的元素名称取代。“(#PCDATA)”部分是对元素内容模型的描述，“#PCDATA”表示元素只能包含字符数据。PCDATA 是 Parser Character DATA 的缩写，意思是可解析字符数据，实际上就是字符数据。

例如对例 3-1 中的“说明”元素的声明：

```
<!ELEMENT 说明 (#PCDATA)>
```

对于这个声明，以下的“说明”元素的使用都是合法的：

```
<说明>&content;</说明>
<说明>某公司部分联系人信息</说明>
```

2. 包含简单内容的复杂类型声明

简单内容表示元素只能包含合法的 XML 文本字符。而复杂类型表示该元素还可以有属性。带有简单内容的复杂类型的元素类型声明(ETD)采用的结构与简单类型声明的结构一样。但是在该元素上还有属性定义，即还要在 DTD 中给该元素声明属性。

如例 3-1 中的“电话”元素的声明：

```
<!ELEMENT 电话 (#PCDATA)>
<!ATTLIST 电话 类别 CDATA "固定电话">
```

先声明“电话”元素所包含的内容是只含字符数据的简单内容，再声明该元素上使用的属性，属性名为“类别”，属性值是字符数据，默认取值为“固定电话”。对于这个声明，“电话”元素的以下使用是合法的：

```
<电话 类别="固定电话">(010)62345678</电话>
<电话 类别="移动电话">13880443013</电话>
```

3. 包含复杂内容的复杂类型声明

复杂内容表示元素内容可以包含其他元素作为该元素的子元素。复杂类型表示该元素可以有属性。对复杂内容的声明，只需将含有简单内容的复杂类型声明中的内容模型(ECM)部分改为相应的子元素即可，语法如下：

```
<!ELEMENT elementName(element1,element2,...)>
```

“element1”与“element2”表示所声明的元素含有的子元素，在具体应用中，以具体子元素名称替代即可。如例 3-1 中的“联系人列表”和“联系人”元素的声明：

```
<!ELEMENT 联系人列表(说明,联系人)>
<!ELEMENT 联系人(姓名,ID,公司,EMAIL,电话,地址)>
```

对于所包含的子元素，可以控制其出现的先后顺序、出现的次数，还可以对子元素进行分组。

(1) 控制子元素出现的先后顺序

在这种格式下,元素拥有哪些子元素,每个子元素出现的次数和位置都有明确的规定,在具体的文档实例中,必须严格执行。这就是子元素列表的设置方式。语法如下:

```
<!ELEMENT Element_name(child_element,child_element,...)>
```

在此语法中,(child_element,child_element,...)部分为元素 Element_name 所拥有的子元素列表。子元素按照设想的某种次序(这种次序以逗号隔开)依次出现,如例 3-1 中元素“联系人”所包含的子元素,在 XML 实例文档中必须按照“姓名”、“ID”、“公司”、“EMAIL”、“电话”、“地址”的顺序依次出现,而且每个元素必须出现一次。如果同一元素需要多次出现,就需要将该元素重复地包含在子元素中。如:

```
<!ELEMENT 个人信息(姓名,性别,喜好,喜好)>
```

这个声明中,“个人信息”就可以包含两个“喜好”子元素,以下实例是合法的:

```
<个人信息>
  <姓名>张三</姓名>
  <性别>男</性别>
  <喜好>书法</喜好>
  <喜好>音乐</喜好>
</个人信息>
```

在实际应用中,这种情况会经常遇到。个人喜好要根据个人的不同情况而定,可能很少,也可能很多。也就是说,不同的实例文档中的“喜好”可能出现 1 次、2 次,也可能出现多次,对于同一元素可以出现的次数在 DTD 声明中可以进行较为严格的控制。

(2) 控制元素出现的次数

在子元素列表的设置方式中,除了设置各个子元素的出现顺序,也隐含地规定了各元素的出现次数为 1 次。同一元素要多次出现,可以将此元素在子元素列表中重复相应的次数。但出现的次数太多,或出现的次数不能确定时,这种方法就显得比较笨拙了。可以利用简单的符号来控制元素出现的次数。

DTD 支持的可以控制元素出现次数的符号,如表 3-1 所示。

表 3-1　控制元素次数的符号

支持的符号	确定的次数	支持的符号	确定的次数
?	0 次或 1 次	+	1 次或多次
*	0 次或多次,即任意次		

在使用时将具体的符号加在要控制的元素后就可以实现对该元素出现次数的控制了。具体使用方法如下。

设定一个元素可以出现 1 次,也可能不出现。这可以通过在元素名后面追加一个“?”符号来实现。如:

```
<!ELEMENT 个人信息(姓名,性别,喜好?)>
```

则以下实例都是合法的:

```
<个人信息>
  <姓名>张三</姓名>
  <性别>男</性别>
  <喜好>书法</喜好>
</个人信息>
```

或

```
<个人信息>
  <姓名>张三</姓名>
  <性别>男</性别>
</个人信息>
```

设定一个元素可能不出现,也可能出现 1 次或多次,这可以通过在元素名后面追加一个“*”符号来实现。如:

```
<!ELEMENT 个人信息(姓名,性别,喜好*)>
```

则以下实例都是合法的:

```
<个人信息>
  <姓名>张三</姓名>
  <性别>男</性别>
</个人信息>
```

或

```
<个人信息>
  <姓名>张三</姓名>
  <性别>男</性别>
  <喜好>书法</喜好>
</个人信息>
```

或

```
<个人信息>
  <姓名>张三</姓名>
  <性别>男</性别>
  <喜好>书法</喜好>
  <喜好>音乐</喜好>
  <喜好>运动</喜好>
</个人信息>
```

设定一个元素可能出现 1 次,也可能出现多次。这可以通过在元素名后追加一个“+”符号实现。如:

```
<!ELEMENT 个人信息(姓名,性别,喜好+)>
```

则以下实例是合法的:

```
<个人信息>
  <姓名>张三</姓名>
```

```
    <性别>男</性别>
    <喜好>书法</喜好>
</个人信息>
```

或

```
<个人信息>
    <姓名>张三</姓名>
    <性别>男</性别>
    <喜好>书法</喜好>
    <喜好>音乐</喜好>
    <喜好>运动</喜好>
</个人信息>
```

(3) 从元素中进行选择

有时需要在两个或多个互斥的元素中选择其中一个。从多个元素中进行选择使用“或”符号,即“|”。语法如下:

```
<!ELEMENT element_a (element_b|element_c|...)>
```

此声明表示从 element_b、element_c 等元素中选择一个元素作为元素 element_a 的子元素。

例如对于个人信息,其配偶一项,当被描述者是男性时,表示配偶的元素应该是“妻子”;若描述者是女性,则表示配偶的元素应该为“丈夫”。此时需要根据被描述者的性别来确定表示配偶的子元素是“妻子”还是“丈夫”,这两个子元素不能同时出现在个人信息元素中,如例 3-2 所示。

【例 3-2】 从元素中进行选择。

```
<?xml version="1.0" encoding="GB2312" standalone="yes"?>
<!DOCTYPE 员工信息 [
    <!ELEMENT 员工信息(员工+)>
    <!ELEMENT 员工(姓名,性别, (妻子|丈夫))>
    <!ELEMENT 姓名(#PCDATA)>
    <!ELEMENT 性别(#PCDATA)>
    <!ELEMENT 妻子(姓名,性别)>
    <!ELEMENT 丈夫(姓名,性别)>
    ]>
<员工信息>
    <员工>
        <姓名>李虎</姓名>
        <性别>男</性别>
        <妻子>
            <姓名>吴小花</姓名>
            <性别>女</性别>
        </妻子>
    </员工>
    <员工>
```

```
            <姓名>王梅</姓名>
            <性别>女</性别>
            <丈夫>
                <姓名>李海</姓名>
                <性别>男</性别>
            </丈夫>
        </员工>
    </员工信息>
```

选择性元素还可以与其他控制元素次数的方法组合使用，这样可以实现对元素内容更为灵活的控制。如在找工作时通常希望留下自己的联系方式，联系方式可以只留一种，为了方便及时联系，往往需要留下多种联系方式。可以按如下方法实现：

```
<!ELEMENT 联系方式(固定电话|手机|EMAIL|QQ)*>
```

此声明中“*”表示可以在“固定电话”、“手机”、“EMAIL”、“QQ”中进行任意次地选择，也就是说“联系方式”可能没有子元素，也可能有一个子元素，还可能有多个子元素，相同子元素也可以重复出现。

以下实例都是合法的：

```
<联系方式>
    <固定电话>87989653</固定电话>
</联系方式>
```

或

```
<联系方式>
    <固定电话>87989653</固定电话>
    <QQ>441838906</QQ>
</联系方式>
```

等。

需要说明的是，对于“联系方式”中出现多个子元素的情况，是多次选择后的结果，是“*”在发挥作用，而不是意味一次可以选择多个元素。对于在选择后追加“?”和“+”也可以按照同样的方法理解。

(4) 对子元素进行分组

在声明包含复杂内容的复杂类型元素时，可以使用括号将其部分子元素组合为一个“元素组”，该元素组在特性上与普通元素没有什么区别。在元素组内部，元素要按规定的次序出现，而且可以对其应用控制元素出现次数的“*”、“?”、“+”等控制符，这就进一步增加了元素内容设定的灵活性。

对子元素进行分组的语法如下：

```
<!ELEMENT element(child_element,...(child_element,...),...)>
```

例如，在找工作时为了让公司更加相信自己的实践工作能力，往往需要在简历上注明以往工作经历，每一次工作经历应该提供起始时间和工作单位等信息。可以通过以下方

法实现。

```
<!ELEMENT 个人简历(姓名,性别,出生年月,(工作单位,起始时间,结束时间)*,联系方式*)>
```

则以下实例是合法的：

```
<个人简历>
    <姓名>李海</姓名>
    <性别>男</性别>
    <出生年月>1984-12-25</出生年月>
    <工作单位>北方电子联合公司</工作单位>
    <起始时间>2007-3-4</起始时间>
    <结束时间>2007-5-1</结束时间>
    <工作单位>新东方电子有限公司</工作单位>
    <起始时间>2008-1-5</起始时间>
    <结束时间>2008-3-1</结束时间>
    <联系方式>lihai@ 163.com</联系方式>
</个人简历>
```

在这个实例中由“工作单位”、“起始时间”和“结束时间”构成的组共出现了两次。

4. 混合内容类型声明

混合内容类型的元素允许其内容可以既包含字符数据又含有子元素。声明此类元素的基本语法如下：

```
<!ELEMENT elementName(#PCDATA |element1|element2|...)*>
```

例如：

```
<!ELEMENT 联系人(#PCDATA|姓名|电话|EMAIL)*>
```

则以下实例都是合法的：

```
<联系人>
    <姓名>李海</姓名>
    <电话>(028)87346570</电话>
    <EMAIL>lihai@ 163.com</EMAIL>
</联系人>
<联系人>
    <姓名>李小明</姓名>
    <电话>(028)87949581</电话>
    <EMAIL>zxiaom@ 163.com</EMAIL>
    该人是公司销售员
</联系人>
<联系人>
    <姓名>王丽丽</姓名>
    该人是公司经理
    <EMAIL>lili@ 163.com</EMAIL>
</联系人>
```

元素既有字符数据又可包含子元素，从表面上看元素内容的限制少了，但这样会扰乱

文档的层次结构，一般在完成的文档中是不应该出现这种混合元素的。从技术上说，可以轻易地建立一个元素来包含这些字符数据。包含混合内容的元素在实际应用中用得较少。

5. 空内容类型声明

在 XML 实例文档中，还可能有这样的元素，元素本身不包含任何内容，但可以有属性。这种元素的声明语法如下：

```
<!ELEMENT elementName EMPTY>
```

关键字 EMPTY 必须大写，表示元素 elementName 不能包含任何内容，包括文本字符及元素，但是它可以有属性，即在 DTD 中可以有属性声明。

例如：

```
<!ELEMENT 图片 EMPTY>
```

在 XML 实例文档中，使用如下：

```
<图片/>
```

一般情况下，空元素都包含属性，否则该元素的出现没有多大的意义。

6. 任何内容类型声明

这是对于元素内容的最为宽松的限定，实际对元素内容几乎没有任何要求，语法如下：

```
<!ELEMENT Element_name ANY>
```

关键字 ANY 必须大写，表示元素的内容可以是任意子元素，也可以是合法的字符数据。

例如：

```
<!ELEMENT 说明 ANY>
```

则以下的实例都是合法的：

```
<说明>2008 年 2 月修改后的联系人信息</说明>
<说明>&content;</说明>
<说明><联系人>李海</联系人></说明>
```

实际应用中，除非文档明确要求使用这样的元素，否则最好避免使用这种设定。过分的滥用将导致文档结构的不明确，这与使用 DTD 的初衷背道而驰。应该尽可能准确地描述每个元素的内容。

3.3 DTD 中的属性声明

同元素一样，所有有效的 XML 文档中使用到的属性也必须先在 DTD 中进行声明。声明的内容包括属性在哪个元素上使用，属性的名称，属性值的类型，属性默认值，以及元