第 3 章　文档类型定义

知识目标

◎ 了解 DTD 的概念及其作用
◎ 熟悉 DTD 的基本结构
◎ 掌握 DTD 元素的声明语法及对不同元素类型的声明方法
◎ 了解 DTD 中实体的声明
◎ 了解 DTD 属性的类型，掌握 DTD 属性的声明
◎ 掌握内部 DTD 和外部 DTD 的概念
◎ 能够用一般的文本编辑器编写内部 DTD 和外部 DTD 文档

技能目标

◎ 能够综合运用本章知识，编写内部 DTD 或外部 DTD 文档来约束和规范一个相应的 XML 文档
◎ 能够利用 XML 开发工具 XMLwriter 编写及调试 DTD 文档和 XML 文档，并验证 DTD 的有效性

XML 是 SGML 的子集，在 XML 文档元素、元素属性、实体等内容的声明和定义方面，沿用 SGML 的文档类型定义来描述和定义 XML 文档内容成了顺理成章的事情。

SGML 是把文档结构描述与内容本身分离的语言规范，它通过一个独立的描述性文本文件来定义 SGML 中出现的所有元素、类型、元素属性、字符集、实体、PCDATA（parser character data）和 CDATA 等，SGML 文档中的这些内容结构及其组合称为模式，这个描述性文本文件称为文档类型定义（document type definition，DTD）。DTD 明确地说明了 SGML 文档的元素、元素类型、内容模型，还为它们的结构和它们与其他元素的关系定义了规则。

3.1　DTD 概述

在 DTD 中，规定了要引用该 DTD 的 XML 文档能使用的标记元素，父元素包含的子元素，各元素出现的顺序，元素所具有的属性，元素及其属性值的数据类型，以及可使用的实体和符号规则等。

3.1.1 DTD 简介

DTD 是用来描述 XML 文档结构的一种常见方法，用来定义文档的逻辑结构。一个有效的 XML 文档应该是一个符合相应 DTD 验证的 XML 文档。有了统一制定的 DTD，XML 解析器就能够依据 DTD 来验证这类 XML 文档的结构和数据是否正确有效，并能正确解析。除此之外，DTD 还有其他方面的作用，主要可以归纳为如下几点：

(1)用来验证 XML 文档数据的有效性。

(2)统一某行业或组织联盟的 XML 文档的格式和结构。

(3)能够保证在一定范围内，交流和共享 XML 文档数据。

(4)为应用程序设计人员提供 XML 文档逻辑结构的参考。应用程序设计人员不必知道具体的数据就能编写出能够正确解析 XML 文档、显示数据样式和处理文档中的数据的应用程序。

DTD 的约定和声明语句可以包含在一个 XML 文档内部，称为内部 DTD；也可以保存为一个独立的文件，称为外部 DTD。外部 DTD 能方便地被多个 XML 文档引用，同时可用来对交换与共享数据的 XML 文档进行规范，因此很常见。例如，若由多家公司组成一个联盟，各公司联系非常紧密，经常需要通过 XML 文档相互交换数据，就可以编制统一的 DTD 文件存放在某个共享的服务器上，供外部使用的 XML 文档引用该 DTD。

XML 解析器打开一个不含 DTD 的 XML 文档时，只检查文档格式的正确性。但打开一个含有 DTD 的 XML 文档时，不但会检查文档格式的正确性，还会验证文档的有效性。也就是说，有效的 XML 文档不能包含在没有声明元素或属性的 DTD 中，且有效的 XML 文档所包含的每一元素与属性必须符合相关声明语句所定义的规则。

需要注意的是，只有通过 HTML 网页来载入或打开某个 XML 文档时，IE 浏览器中的 XML 解析器才会检查该文档的有效性。如果直接在 IE 浏览器中打开某个 XML 文档，XML 解析器只会检查文档格式的正确性，不会检查该文档是否符合有效性规则，即使该文档拥有文档类型定义也一样。

在 XML 文档中，所用到的绝大多数标记都是用户自己定义的，因此，有时会出现这样的情况：若 A 和 B 是属于同一行业的两个公司，需要利用 XML 文档相互交换数据，A 公司使用＜价格＞标记来表示其产品的价格信息，而 B 公司则可能使用＜售价＞标记来表示其产品的价格信息。当一个 XML 应用程序需要读取这两个公司的 XML 文档数据时，如果只知道用＜价格＞标记表示的是价格信息，或者只知道用＜售价＞标记表示的是价格信息，那么将会有一个公司的价格信息读不出来而产生错误的结果。显然，对于想利用 XML 文档交换数据的各个公司或合作伙伴来说，其间必须遵守共同的约定。只有这样，在用 XML 文档交换或共享数据时，才能够做到畅通无阻。这种对 XML 文档所进行的规范和约定就称为 DTD，即文档类型定义。

DTD 可以被看成是设计编制一类 XML 文档的模板。若能有一个被广泛接受的、统一的 DTD，将会为同行业之间的 XML 数据共享和交换带来诸多方便。例如，如果互联网上各大电子商场的 XML 文档数据都遵循统一约定的 DTD，那么依据这个 DTD 可以编写引用了此 DTD 的 XML 文档进行数据搜索的应用程序，而且可以快速有效地找到有用的信息。可以这样说，创建一个 DTD，依据这个 DTD 来检查 XML 文档的有效性，能够保证 XML 文档内容符合应用程序的结构，还可以被相应的软件很好地识别和处理。事实上，目前有许多行

业或企业联盟都制定了内部通用的 DTD，如 MathML、SMIL 和 WML 等。

3.1.2　DTD 的基本结构

DTD 用来对 XML 文档所使用的元素、元素间的关系、元素可用的属性以及可使用的实体等定义规则。实际上，DTD 是由若干条元素、属性、实体等的定义和声明语句组成的。

【例 3-1】　一个内部包含 DTD 的 XML 完整文档。

```
<?xml version="1.0" encoding="gb2312" standalone="yes"?>
<!--文件名为"例 3-1.xml"-->
<!DOCTYPE 书目 [
<!ELEMENT 书目 (书)+>
<!ELEMENT 书 (书名,作者,出版社,出版时间,内容简介)>
<!ATTLIST 书 类别 CDATA "计算机" ISBN CDATA #REQUIRED>
<!ENTITY introduction "C 语言是国内外广泛使用的计算机语言,学会使用 C 语言进行程序设计是计算机工作者的一项基本功。本书包含了 C 语言的基础知识……">
<!ELEMENT 书名 (#PCDATA)>
<!ELEMENT 作者 (#PCDATA)>
<!ELEMENT 出版社 (#PCDATA)>
<!ELEMENT 出版时间 (#PCDATA)>
<!ELEMENT 内容简介 (#PCDATA)>
]>
<书目>
<书 类别="计算机" ISBN="978-7-302-10853-5">
<书名>C 语言程序设计(第三版)</书名>
<作者>谭浩强</作者>
<出版社>清华大学出版社</出版社>
<出版时间>2005-07-03</出版时间>
<内容简介>&introduction;</内容简介>
</书>
<书 ISBN="978-7-5053-8786-3">
<书名>计算机网络(第四版)</书名>
<作者>谢希仁</作者>
<出版社>电子工业出版社</出版社>
<出版时间>2006-06-01</出版时间>
<内容简介>全书共 10 章,全面系统地介绍了计算机网络的发展和原理体系结构、物理层、数据链路层……</内容简介>
</书>
<!--其他书目信息-->
</书目>
```

用 IE 浏览器直接打开一个 XML 文档时，只能进行文档格式良好性检查，为了进行 DTD 有效性验证，使用 XMLwriter 来打开【例 3-1】的 XML 文档并验证后，会提示“文档是

格式良好的 XML 并符合 Schema/DTD”，如图 3-1 所示。

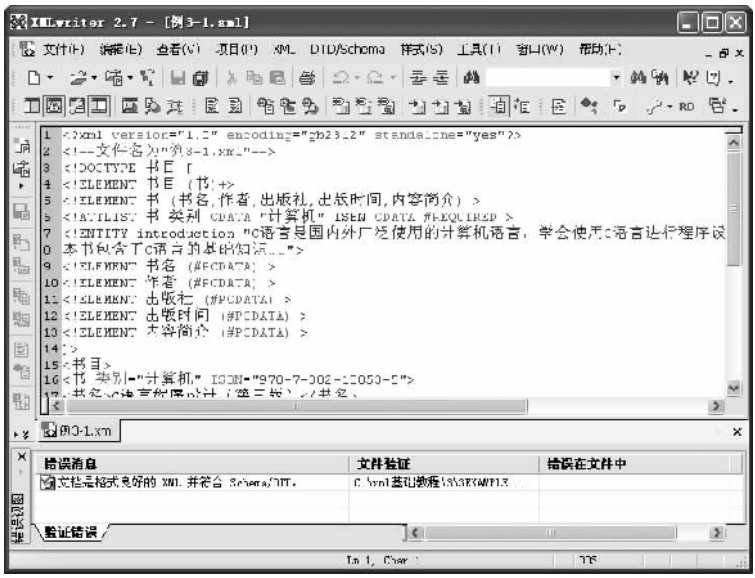

图 3-1 【例 3-1】的 XML 文档在 XMLwriter 中验证后的显示结果

XMLwriter 是一款专业的 XML 文档创建、编辑和调试工具软件，同时支持 XSL、DTD、Schema、CSS、HTML 和 TEXT 等文件的编辑，可以使用 CSS、XSL 等语言在集成的预览窗口中格式化一个 XML 文档。XMLwriter 用户界面直观，操作简单，具有自动查找、替换和书签的功能，还提供详尽的在线帮助，是 XML 初学者或专业开发者首选的开发工具，可以从 Internet 上搜索下载其试用版本。

从【例 3-1】XML 文档中可以看出，一个 DTD 的基本结构包括以下几部分：

(1)DTD 声明语句。DTD 声明语句是由“<!DOCTYPE”开始，以“]>”结束的语句。一个包含 DTD 的 XML 文档必须有这一条 DTD 声明语句。该语句同时还指定了 XML 文档的根元素名称。例如，在上述文档的 DTD 中，指定<书目>为这个 XML 文档的根元素。

(2)元素类型声明语句。元素类型声明语句是由“<!ELEMENT”开头的语句，用来定义一个 XML 文档中可以出现的元素，它定义了元素的名称、数据类型以及该元素所包含的子元素等。例如，上述文档的 DTD 声明了<书目>元素可以有任意多个<书>子元素，声明了<书>元素应有<书名>、<作者>、<出版社>、<出版时间>和<内容简介>5 个子元素。

(3)属性列表声明语句。属性列表声明语句是由“<!ATTLIST”开头的语句，定义了在指定元素内包含的属性名，以及这些属性的数据类型与默认值等。例如，上述文档的 DTD 中，声明了<书>元素有“类别”和“ISBN”两个属性，其属性值均为字符型数据，而且“类别”的默认值为“计算机”，“ISBN”属性是必须有的。

(4)实体声明语句。实体声明语句是由“<!ENTITY”开头的语句，用来存储常用的文字区块，或包括在文档中的非 XML 的数据。如上述文档的 DTD 中声明了一个名为 intro-

duction 的简单实体，包含了有关书的内容简介。

(5)注释。在一个 DTD 声明中，可以含有若干条注释语句，其格式与一般 XML 文档中的注释语句相同，这里不再特别说明。

3.2　元 素 声 明

一个合法有效的 XML 文档所使用的每一种元素，都必须在相应的 DTD 中明确声明，声明的内容包括元素的名称、可能包含的内容、元素的数据类型、元素允许具有的子元素及子元素出现的顺序以及元素所具有的属性等。在 DTD 中精确控制元素所包含的内容可以控制一个有效 XML 文档的逻辑结构。

3.2.1　元素声明的语法

在 DTD 中，元素声明的语法格式如下：

<!ELEMENT 元素名称 元素内容模型>

其中的参数说明如下：

- <!：表示元素声明语句的开始。
- ELEMENT：元素声明语句的关键字，用大写字母表示，指示该语句为元素声明语句。
- 元素名称：用来指定声明的元素名称，在使用时用具体的元素名称来代替。
- 元素内容模型(element content model，ECM)：用来描述元素可能包含的内容。
- >：表示一条元素声明语句的结束。

举例说明合法的元素声明，如：

- <!ELEMENT 标题 (#PCDATA)>
- <!ELEMENT 型号 (#PCDATA)>
- <!ELEMENT 价格 (#PCDATA)>
- <!ELEMENT 品牌 (#PCDATA)>
- <!ELEMENT 计算机 (品牌，型号，价格)>
- <!ELEMENT 计算机价格 (标题，计算机)>

对上述说明举例如下：

(1)例子共定义了标题、型号、价格和品牌 4 个基本元素，也就是不包含其他元素的元素，这些元素的内容均为字符数据，由“(#PCDATA)”指明；其中 PCDATA 为可解析数据。

(2)语句“<!ELEMENT 计算机 (品牌，型号，价格)>”定义了一个名为<计算机>的复合元素，包含<品牌>、<型号>和<价格>3 个子元素，而且子元素出现的顺序必须是<品牌>子元素在最前，接着是<型号>子元素，最后是<价格>子元素。

(3)语句“<!ELEMENT 计算机价格 (标题，计算机)>”定义了一个名为<计算机价格>的复合元素，其包含<标题>和<计算机>两个子元素，<标题>子元素必须出现在<计算机>子元素之前。

(4)若由上述语句组成的 DTD 所定义的元素都直接或间接地包含在元素<计算机价格>中，当然除了<计算机价格>元素自身，因此上述语句还隐含地定义了 DTD 的根元素，即<计算机价格>。

由上述语句组成的完整 DTD 见【例 3-2】。

【例 3-2】 DTD 元素声明示例。

```
<?xml version="1.0" encoding="gb2312"?>
<!ENTITY Title "计算机价格">
<!ELEMENT 标题 (#PCDATA)>
<!ELEMENT 型号 (#PCDATA)>
<!ELEMENT 价格 (#PCDATA)>
<!ELEMENT 品牌 (#PCDATA)>
<!ELEMENT 计算机 (品牌,型号,价格)>
<!ELEMENT 计算机价格 (标题,计算机)>
<!ATTLIST 价格 币种 CDATA "RMB">
```

【例 3-3】 依据【例 3-2】DTD 创建的一个正确的 XML 文档。

```
<?xml version="1.0" encoding="gb2312" standalone="no"?>
<!DOCTYPE 计算机价格 SYSTEM "例 3-2.dtd">
<计算机价格>
    <标题>&Title;</标题>
    <计算机>
        <品牌>IBM</品牌>
        <型号>启天 E4150</型号>
        <价格 币种="RMB">7600</价格>
    </计算机>
</计算机价格>
```

在 XMLwriter 中将【例 3-3】打开并进行有效性验证后的结果如图 3-2 所示。

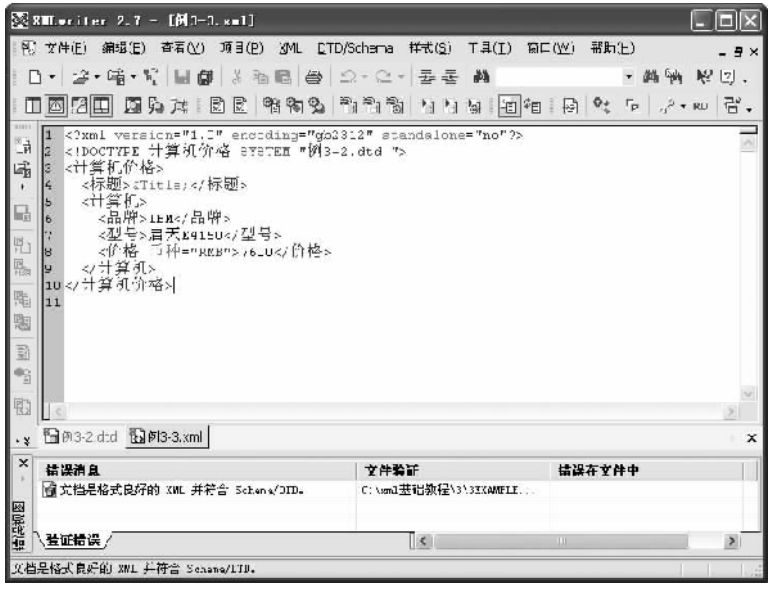

图 3-2 依据【例 3-2】编写的正确的 XML 例子

【例 3-4】　依据【例 3-2】编写的一个错误的 XML 文档。

```
<?xml version="1.0" encoding="GB2312" standalone="no"?>
<!DOCTYPE 计算机价格 SYSTEM "例 3-2.dtd">
<计算机价格>
    <计算机>
        <品牌>IBM</品牌>
        <价格 币种="RMB">7600</价格>
    <型号>
      <标题>&Title;</标题>
    </型号>
    </计算机>
</计算机价格>
```

由【例 3-4】可以看出，文档存在 3 处错误：

(1)<计算机>元素中的两个子元素<型号>和<价格>的出现顺序不符合所引用 DTD 中的规定。DTD 中规定<型号>元素必须出现在<价格>元素之前，而文档中正好相反。

(2)DTD 中规定<计算机价格>元素应包含<标题>和<计算机>两个子元素，而在文档中<计算机价格>元素中只包含了一个<计算机>子元素，而没有<标题>子元素。

(3)DTD 中规定<型号>元素为基本元素，即其内容必须为字符数据，但在文档中<型号>元素成了复合元素，包含了一个<标题>元素。

为了更清楚地看到 DTD 的有效性，先在 IE 浏览器中直接打开【例 3-4】的 XML 文档，由图 3-3 可以看出，该 XML 文档在 IE 浏览器中能正确打开，并不能检查出任何错误。

图 3-3　依据【例 3-2】编写的不正确的 XML 例子

【例 3-4】可以通过结构完整性检验，但不能通过有效性检验。其在 XMLwriter 中的有效性验证的结果如图 3-4 所示。

图 3-4 中左下角的状态栏显示了检验结果：根据 DTD/Schema，元素内容无效。这表示 XML 解析器检查到<计算机>元素前缺少了<标题>元素。

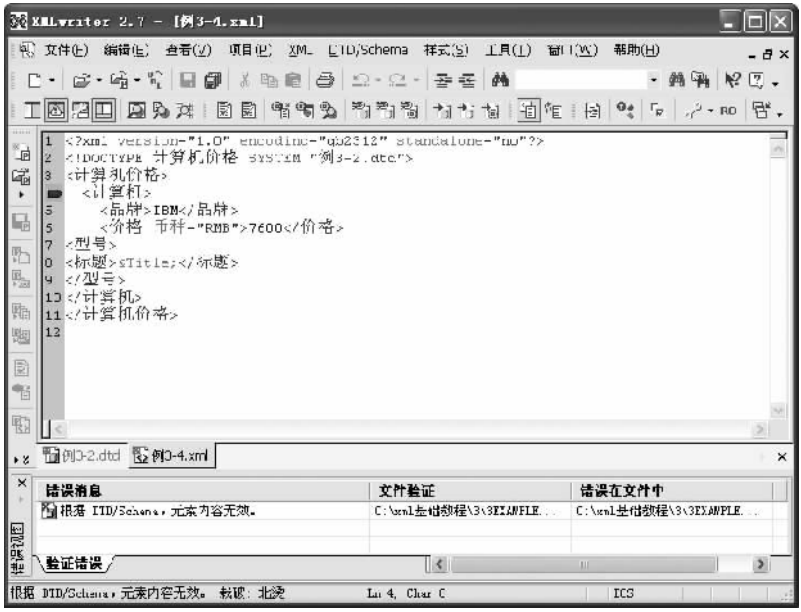

图 3-4 依据【例 3-2】编写的不正确 XML 例子的有效性验证

3.2.2 不同元素类型的声明

一个 DTD 不仅应向语法分析器说明它所关联的 XML 文档的根元素是什么，而且还要说明文档的内容和结构，从而规定文档结构中的每一个细节。为了定义这些细节，必须使用元素类型声明(element type declaration，ETD)来声明所有有效的文档元素，详细展开 DTD 中元素的声明部分。

ETD 不但说明了 XML 文档中可能存在的元素，给出了元素的名称，还给出了元素的具体类型。一个 XML 元素可以为空，也可以是一段纯文本，还可以有若干个子元素，而这些子元素同时又可以有各自的子元素。

1. 基本字符元素声明

XML 文档中的基本元素是指那些含有字符数据，而不含任何子元素的元素。在相应的 DTD 中，声明基本字符元素的具体格式如下：

<!ELEMENT 元素名称 (#PCDATA)>

这里的 PCDATA 是指可解析字符数据，小括号和 PCDATA 前面的“#”不能省略。

2. 含有子元素的元素声明

有时 XML 文档中的一个元素可以包含若干指定的子元素，也就是说，元素含有子元素。在相应的 DTD 中，声明含有子元素的元素的具体格式如下：

<!ELEMENT 父元素名 (子元素名 1,子元素名 2,子元素名 3,…)>

这种声明方式是十分严格的，也就是说，经上述声明的元素只能包含所指定的子元素，而不能直接包含其他任何字符数据；同时这些子元素在 XML 文档中必须以声明中的排列顺序依次出现在其父元素中，每个子元素必须出现且只能够出现一次。

例如，下面的 DTD 声明语句规定了<电子书>元素必须依次包含<书名>、<作者>、

<出版社>、<发行时间>和<定价>5 个子元素。这 5 个子元素只能含有字符数据，而且必须顺序出现在其父元素<电子书>元素中，不能省略，并且只能出现一次。

```
<!DOCTYPE 书目 [
<!ELEMENT 电子书 (书名,作者,出版社,发行时间,定价)>
<!ELEMENT 书名 (#PCDATA)>
<!ELEMENT 作者 (#PCDATA)>
<!ELEMENT 出版社 (#PCDATA)>
<!ELEMENT 发行时间 (#PCDATA)>
<!ELEMENT 定价 (#PCDATA)>
]>
```

因此，在相应的 XML 文档中下面的元素是有效的。

```
<电子书>
  <书名>Eclipse 3 高级编程</书名>
  <作者>李化、李政仪</作者>
  <出版社>清华大学出版社</出版社>
  <发行时间>2006-06-01</发行时间>
  <定价>58.00</定价>
</电子书>
```

而下面的<电子书>元素则不是有效的。首先元素<电子书>所包含的子元素顺序并非声明中所规定的顺序，同时丢掉了<发行时间>子元素，还有<定价>子元素在<电子书>元素中出现了两次。

```
<电子书>
  <书名>Eclipse 3 高级编程</书名>
  <定价>58.00</定价>
  <作者>李化、李政仪</作者>
  <出版社>清华大学出版社</出版社>
  <定价>68.00</定价>
</电子书>
```

3. 子元素出现次数的声明

在 XML 文档中，有的元素可能会出现多次或者一次也不出现。在对应的 DTD 中，对包含子元素的元素声明时，除了对父元素使用 ANY 关键字声明外，还可以在该元素的后面加上特定的符号来控制其可以出现的次数。

在对应的 DTD 中，控制子元素出现次数的声明格式为：

```
<!ELEMENT 元素名 控制字符>
```

表 3-1 列出了 DTD 中控制子元素出现次数的特定符号。

表 3-1　DTD 中控制子元素出现次数的特定符号

符　　号	允许元素出现的次数
无符号	没有基数操作符，表示必须出现且只能出现一次

续表

符　　号	允许元素出现的次数
?	可不出现或只出现一次
*	可不出现或可多次出现，即可出现任意次
+	出现一次或多次，即至少出现一次

例如：

<!ELEMENT 电子资源(电子书 *，电子报纸+，电子杂志?，网站)>

这个声明语句规定了：元素<电子资源>的子元素<电子书>在 XML 文档中可以一次也不出现，也可以出现多次；子元素<电子报纸>必须出现一次或一次以上，即至少出现一次；子元素<电子杂志>可以不出现或只出现一次；而子元素<网站>必须出现一次且只能出现一次。

4. 选择性子元素的声明

在 XML 文档中，有时一个父元素可以在指定的多个子元素中选择其中的一个作为子元素。

在对应的 DTD 中，选择性子元素的声明格式为：

<!ELEMENT 父元素名(子元素名 1|子元素名 2|子元素名 3|…)>

此声明语句规定了一个父元素可以包含的子元素必须是小括号内所指定的多个子元素中的一个。指定的可供选择的子元素之间用“|”作为分隔符。

下面先来看如下的 XML 文档：

```
<社会关系>
    <关系>
        <父亲>赵启宝</父亲>
    </关系>
        <职业>教师</职业>
        <政治面貌>党员</政治面貌>
    <关系>
        <女儿>赵媛媛</女儿>
    </关系>
        <职业>学生</职业>
        <政治面貌>团员</政治面貌>
</社会关系>
```

在上面的 XML 文档中，需要反映一个人的社会关系的关系元素。在只考虑至亲时，社会关系的关系只可能是父亲、母亲、丈夫、妻子、儿子、女儿这 6 种关系中的一种，所以可以定义一个元素<关系>，在 DTD 中定义<关系>元素的声明语句为：

<!ELEMENT 关系(父亲|母亲|丈夫|妻子|儿子|女儿)>

依据上述的 DTD 对元素的声明，在对应的 XML 文档中，下面列出的元素<关系>都是有效的，具体如下：

<关系>

```
    <父亲>赵启宝</父亲>
</关系>
<关系>
    <女儿>赵媛媛</女儿>
</关系>
```

但下面两个例子中的<关系>元素都是无效的，因为按照上面 DTD 的定义，元素<关系>不能包含多个子元素，也不允许包含任何其他子元素。

```
<关系>
    <父亲>赵启宝</父亲>
    <母亲>刘平</母亲>
</关系>
<关系>
    <叔叔>赵德宝</叔叔>
</关系>
```

另外，在选择性子元素的声明中还可以一起声明子元素出现的次数，从而实现对元素内容更为灵活的控制，例如：

```
<!ELEMENT 关系(父亲|母亲|丈夫|妻子|儿子|女儿)?>
```

上面的声明表示<关系>元素中的子元素可以是<父亲>、<母亲>、<丈夫>、<妻子>、<儿子>、<女儿>这 6 个中的任何一个，也可以一个都不出现。

5. EMPTY 元素的声明

例如，在 HTML 中，
、<HR>、<IMG>等标记都是没有内容的，称之为空元素。在 XML 文档中也可能会包含一些空元素，这些空元素只有一个独立标记，而没有任何内容。

在对应的 DTD 中，空元素的声明格式为：

```
<!ELEMENT 元素名 EMPTY>
```

下面的语句定义了一个空元素：

```
<!ELEMENT style EMPTY>
```

空元素在 XML 文档中的表示格式为<元素名/>，可以没有结束标记，但必须在标记名之后添加“/”。对于上面例子的空元素声明，在 XML 文档中使用此空元素的格式为：

```
<style/>
```

6. ANY 元素的声明

内容模型为 ANY 的元素实质上是没有结构的，若在 DTD 中声明了某一元素为 ANY 类型时，说明这一元素可以包含任意内容、任何数据、任何声明的子元素及其数据、子元素的组合。在对应的 DTD 中，ANY 元素的声明格式为：

```
<!ELEMENT 元素名 ANY>
```

下面的语句定义了一个包含任意内容的元素<音像>：

```
<!ELEMENT 音像 ANY>
```

这样在一个相对应的 XML 文档中，<音像>元素就可包含任意被声明过的元素，出现的次数和顺序也不受限制，并且该元素除了可以包含子元素以外，还可以直接包含字符数据。

7. 混合内容类型元素的声明

混合内容类型元素既允许包含字符数据，又允许包含以任意顺序出现的、指定的一个或多个子元素，还允许不包含任何内容。

混合内容类型元素与 ANY 型元素相比，最主要的区别在于，前者所能包含的子元素必须是在指定的若干个元素之中，而后者则没有任何限制。

在 DTD 中，要声明一个允许包含字符数据加上零个或多个指定元素的混合内容类型元素时，可以采用如下的元素声明格式：

<!ELEMENT 元素名(#PCDATA|子元素 1|子元素 2|…)*>

上述声明语句中，小括号内的每个子元素名应写在#PCDATA 之后，并用“|”符号分隔，再在右小括号之后加星号“*”。

试看下面关于一个客户的联系方式的元素定义的例子。

<!ELEMENT 联系方式(#PCDATA|联系电话|EMAIL|QQ 号|联系地址)*>

这个 DTD 中元素类型声明语句声明了一个<联系方式>元素，允许<联系方式>元素可以包含字符数据、任意个<联系电话>子元素、任意个<EMAIL>子元素、任意个<QQ 号>子元素、任意个<联系地址>子元素。

若 DTD 文档定义中包含了上述声明语句，则下面相应的 XML 文档示例中的几个“联系方式”元素都是有效的。

```
…
<联系方式>
    <联系电话>010-88888888</联系电话>
    <EMAIL>zhangxueyou@163.com</EMAIL>
</联系方式>
…
<联系方式>
    工作时间若想联系我，请发 E-mail，其他时间可打电话直接联系。
    <联系电话>010-88888888</联系电话>
    <EMAIL>zhangxueyou@163.com</EMAIL>
    <EMAIL>zhangxueyou@sina.com</EMAIL>
</联系方式>
…
<联系方式>
    科技改变你的梦想!
    <联系电话>010-66666666</联系电话>
    <联系电话>13888888888</联系电话>
    <QQ 号>88888888</QQ 号>
    <联系地址>北京市海淀区某路 8 号</联系地址>
</联系方式>
…
```

在上述 XML 文档的示例中，元素<联系方式>既可有字符数据，又可有任意个指定的子元素，每个子元素又可出现任意次。从表面上看，对元素内容的限制少了会使元素内容变

得丰富，但这样会扰乱文档数据的层次结构，所以实际上这样定义的 DTD 非常少。

8. 实体的声明

在 DTD 中，可以声明一个实体来将多种不同类型的数据并入到一个 XML 文档中，这样能够节省大量的时间，大大提高编制 XML 文档的效率。在 XML 文档中，可以把经常使用到的 XML 文字块定义成实体，以方便在 XML 文档任何需要的地方加入。此外，还可以将某个外部文档定义成外部实体，以方便将这个外部文档中的数据添加到一个 XML 文档中。

简单地说，实体是一个事先定义好的数据或数据集合，可以被方便地引用到任何需要这些数据或数据集合的地方。在 DTD 中，可以把实体看做是有一定内容的一个常量，可以是有效的 XML 文档本身，可以是外部的 DTD 子集，定义成 DTD 中外部实体的外部文档，还可以是在 DTD 中定义的用双引号引起来的字符串。

在 DTD 中，声明实体的一般格式为：

＜!ENTITY［％］实体名 实体值＞

其中，ENTITY 是关键字，必须大写，而且括在“＜!”和“＞”中。实体名是实体的名称，必须以字母或下划线开头，允许与文档中元素或属性的名称相同。另外，实体名称中字母的大小写是不同的，即是区分大小写的。例如，名为 style 的实体与名为 Style 的实体是两个不同的实体。实体值表示实体的具体内容，内部实体的内容是一串包含在单引号或双引号内的连续字符，其中不能包含“&”字符和“％”字符。

DTD 中定义各种实体的具体方法有：

(1)内部一般实体的定义和使用。在 DTD 中声明内部一般实体的格式为：

＜!ENTITY 实体名 实体值＞

在 XML 文档或者 DTD 中引用内部一般实体时，需在实体名前加“&”符号，在实体名后添加“;”符号。其对应的 XML 文档的使用格式为：

& 实体名；

(2)内部参数实体的定义和使用。与内部一般实体不同的是，参数实体的内容不仅可以包含文本，还可以包含元素类型声明、属性列表声明、一般实体声明、标签声明、处理指令和注释等。一般实体可在 XML 元素中引用，也可以在 DTD 中引用；但参数实体只能在 DTD 中引用，并且通常情况下只能在外部 DTD 文档中引用。

内部参数实体是指在外部 DTD 中定义的一段具体内容，只能在该 DTD 的其他声明语句中引用的已解析实体。声明内部参数实体的格式为：

＜!ENTITY ％ 实体名 实体值＞

在 DTD 中引用内部参数实体时，需要在实体名前加“％”，在实体名后添加“;”，其语法格式如下：

％实体名；

(3)外部一般实体的定义和使用。外部实体所对应的内容通常为一个独立存在的文件，在 DTD 中定义某个外部实体时需要指定该实体所对应文件的 URL。外部实体可以被多个文档引用，与只能在文档内部定义和引用的内部实体相比，外部实体具有更好的灵活性与共享性。

每个独立存在的 XML 文档都可以被定义为一个外部一般实体，在某个 XML 文档中通过对外部一般实体的引用，就可以嵌入另一个指定的 XML 文档，或者将多个 XML 文档合并成一个较大的 XML 文档。

在 DTD 中定义外部一般实体的格式为：

<!ENTITY 实体名 SYSTEM 实体 URL>

在 XML 文档中引用外部一般实体时，同样需在实体名前加“&”，在实体名后加“;”。其引用的语法格式为：

&实体名;

(4)外部参数实体的定义和使用。与一个独立的 XML 文档可以被定义为一个外部一般实体类似，一个独立的 DTD 文档可以被定义为一个外部参数实体，并可通过这个 DTD 文档的 URL 对其进行引用。这样，在某个内部 DTD 或者外部 DTD 中通过对外部参数实体的引用，就可以嵌入另一个指定的 DTD 文档，也可以将多个 DTD 文档合并成一个较大的 DTD 文档。与内部参数实体相比，外部参数实体具有更好的灵活性和共享性。

声明外部参数实体的语法格式为：

<!ENTITY % 实体名 SYSTEM 实体 URL>

在 DTD 中引用外部参数实体时，同样需在实体名前加“%”，在实体名后跟“;”。其语法格式为：

%实体名;

3.3 属性声明

DTD 中除了元素需要声明外，元素的属性也要进行相应的声明。属性声明规定了与给定的元素相联系的属性的名字、数据类型和默认值，还规定了属性是可选择的还是必需的，是否有默认值等。属性和元素的关系是隶属关系，属性隶属于元素。在书写 XML 文档时，属性写在元素结束标记“>”之前。

3.3.1 属性声明的语法

在 XML 文档中，属性是元素组成的可选部分，是由“=”分隔开的名称—键值对的组成，其作用是对元素及其内容的附加信息进行描述；DTD 中的属性声明用来定义其相应的 XML 文档中某些元素可接受的属性。属性定义相对于元素定义较为灵活，可以出现在 DTD 定义部分的任何位置，并不强制要求出现在对应元素的 DTD 定义之后，有时甚至可以为不存在的元素定义属性，从而扩大属性的适用范围。

DTD 中对属性定义的一般语法为：

```
<!ATTLIST 元素名 属性 1 名称 属性 1 类型 属性 1 的缺省取值
                 属性 2 名称 属性 2 类型 属性 2 的缺省取值
                 …
                 属性 n 名称 属性 n 类型 属性 n 的缺省取值
>
```

其中，ATTLIST 是 XML 系统关键字；元素名是要定义属性的元素的名称；属性名称是为元素定义的属性的名字，属性名称可以是以字母或汉字字符开头，包含字母、数字、连字符、下划线和英文句号的任何字符串；属性类型是用来定义元素属性值的数据类型，属性类型的相关内容，将在本章后面的内容中介绍。属性的缺省取值是为元素定义若其属性缺省

时默认的取值，即用来定义缺省值，是可选项。在一个属性声明语句中可以为一个元素声明多个属性。

下面的程序段为元素＜手机卡＞定义了“id”和“归属地”两个属性，其中“id”属性的属性类型为 ID，缺省值为＃REQUIRED 型；“归属地”属性的属性类型为 CDATA，缺省值为＃IMPLIED 型。

【例 3-5】 DTD 属性声明示例。

```
＜?xml version＝"1.0" encoding＝"gb2312"?＞
＜!--DTD 中为元素声明属性的例子--＞
＜!DOCTYPE 手机卡品［
    ＜!ELEMENT 手机卡品（手机卡＊）＞
    ＜!ELEMENT 手机卡（＃PCDATA）＞
    ＜!ATTLIST 手机卡 id ID ＃REQUIRED
                    归属地 CDATA ＃IMPLIED
＞
］＞
＜手机卡品＞
    ＜手机卡 id＝"BJ410735687" 归属地＝"北京"＞移动动感地带卡＜/手机卡＞
    ＜手机卡 id＝"SH410733762" 归属地＝"上海"＞移动动感地带卡＜/手机卡＞
＜/手机卡品＞
```

3.3.2　属性默认值的设置

在 DTD 中，定义属性的缺省值时，可以对其取值作出许多规定，包括文档是否需要为一个属性提供取值，是否在未定义取值时使用它的缺省值，这个缺省值是否可以修改等。根据这些规定，在定义属性时可以指定属性的默认值，可以用 REQUIRED、IMPLIED、FIXED 三个关键字分别来指定在对应的 XML 文档中必须提供元素属性的属性值，忽略元素属性的属性值，使用固定值作为元素属性的属性值。

1. 必须赋值的属性

关键字 REQUIRED 用来声明 XML 文档中必须给出属性的属性值。例如，在版权图书 XML 文档的 DTD 中，需要为＜图书＞元素定义属性 ISBN，为了确保图书是正规出版社出版，在＜图书＞元素中必须给出图书的 ISBN，一般正版图书都有 ISBN，没有 ISBN 的可能是非正规的出版物。对于这种情况，在 DTD 中为＜图书＞元素定义属性时就可以使用关键字 REQUIRED 来指定属性所属元素，在书写时必须给出属性值，不用提供属性的缺省值。

在 DTD 中，为元素声明属性必须赋值的格式为：

```
＜!ATTLIST 元素名 属性名 属性类型 ＃REQUIRED＞
```

因此，在 DTD 中，为＜图书＞元素声明 ISBN 属性的语句为：

```
＜!ATTLIST 图书 ISBN CDATA ＃REQUIRED＞
```

2. 属性值可有可无的属性

当使用 IMPLIED 关键字时，语法分析器不再强行要求在 XML 文档中给该属性赋值，而且也不必在 DTD 中为该属性提供缺省值。

在 DTD 中，为元素声明一个属性值可有可无的属性的格式为：

<!ATTLIST 元素名 属性名 属性类型 #IMPLIED>

上述对<图书>元素的属性声明语句扩充为：

<!ATTLIST 图书 ISBN CDATA #REQUIRED CIP CDATA #IMPLIED>

3. 有固定取值的属性

有一种特殊情况，需要为一个特定的属性提供一个缺省值，且不希望 XML 文档的程序员用别的值替代这个缺省值。这时，可使用 FIXED 关键字来声明属性，并为该属性提供一个固定取值的缺省值。

在 DTD 中，为元素声明一个属性值固定的属性的格式为：

<!ATTLIST 元素名 属性名 属性类型 #FIXED "缺省值">

上述对<图书>元素的属性声明语句继续扩充如下：

```
<!ATTLIST 图书 ISBN CDATA #REQUIRED
    CIP CDATA #IMPLIED
    经销 CDATA #FIXED "各地新华书店">
```

4. 事先定义了缺省值的属性

在 DTD 中，在属性定义时若没有使用任何关键字，就必须为该属性提供一个缺省值。如果在对应的 XML 文档中为该属性提供一个新的属性值，这个新属性值就覆盖了事先定义的缺省值，若不给出属性值，就默认采用 DTD 中给出的缺省值。

在 DTD 中，为元素定义一个具有默认属性值的属性的格式为：

<!ATTLIST 元素名 属性名 属性类型 "缺省值">

此时为<图书>元素声明属性的例子可扩充为：

```
<!ATTLIST 图书 ISBN CDATA #REQUIRED
    CIP CDATA #IMPLIED
    经销 CDATA #FIXED "各地新华书店"
    开本 CDATA "16 开">
```

3.3.3 属性的类型

属性类型规定了属性的值是哪种类型的数据，常见的属性类型为 CDATA，即字符数据类型的属性。事实上，XML 规范允许为元素的属性指定 10 种不同的类型，各种属性类型的名称及其说明见表 3-2。

表 3-2 DTD 中属性的数据类型

属性类型	说　明
CDATA	属性值为普通的字符数据
ID	属性值在 XML 文档中必须是唯一的
IDREF	表示该属性值参考了 XML 文档中另一个 ID 属性
IDREFS	表示该属性值参考了用空格分隔的 XML 文档中的多个 ID 属性
ENTITY	表示该属性的设定值是一个外部实体，例如，一个声音文件

续表

属性类型	说　　明
ENTITIES	表示该属性包含了多个外部实体,实体间用空格分隔
NMTOKEN	属性值只能由字母、数字、下划线、连字符、圆点、冒号等组成
NMTOKENS	属性值能由多个 NMTOKEN 组成,NMTOKEN 间用空格分隔
NOTATION	属性值是在 DTD 中声明过的注释名称
Enumerated(枚举)	属性值是从给定值的列表中选定的一个

这 10 种类型总体上可分为三大类。第一类是字符串类型,指的是 CDATA,其值可以是任何合法的字符串。第二类属于枚举类型,包括 Enumerated 和 NOTATION,需要在 DTD 中为它们声明可取值的列表。剩下的 7 个属于第三类,称为记法类型,含有不同的词法及语义限定。

1. CDATA 类型

CDATA 指的是纯文本,是由多个字符组成的字符串,其中,字符可以是除了“<”、“'”和“"”的任意字符。

如果属性值中既包含双引号又包含单引号,则定界符最好使用双引号,属性值中的引号用预定义实体引用代替。例如:

```
<显示器 Size='22"'>…</显示器>
```

如果属性值中包含了其他特殊字符,也要使用预定义实体引用替代,如“&”代替“&”,“<”代替“<”,“"”代替“"”,“'”代替“'”。用经纬度作为属性来表示地理位置的 XML 语句如下:

```
<地理位置 经度="135&amp;quot;18&amp;apos;" 纬度="36&amp;quot;26&amp;apos;">
```

需要特别注意的是,在 DTD 中声明 XML 元素的内容为字符串时,用的是“#PCDATA”,而声明属性值为字符数据时,用的是“#CDATA”。

2. ID 类型

在一个 XML 文档中,所有声明为 ID 属性类型的元素,其对应属性的值不能重复。

【例 3-6】 一个 ID 属性类型示例。

```
<?xml version="1.0" encoding="gb2312"?>
<!DOCTYPE 移动通信卡 [
    <!ELEMENT 卡品 (#PCDATA)>
    <!ELEMENT 号码 (#PCDATA)>
    <!ELEMENT 生效日期 (#PCDATA)>
    <!ELEMENT 亲情卡 (卡品,号码,生效日期)>
    <!ATTLIST 亲情卡 id 号 ID #REQUIRED>
    <!ATTLIST 亲情卡 品牌 (中国移动|中国联通) "中国移动">
    <!ATTLIST 亲情卡 属地 CDATA #IMPLIED>
    <!ELEMENT 手机卡 (卡品,号码,生效日期,亲情卡)>
    <!ATTLIST 手机卡 id 号 ID #REQUIRED>
    <!ATTLIST 手机卡 品牌 (中国移动|中国联通) "中国移动">
```

```
    <!ATTLIST 手机卡 属地 CDATA #IMPLIED>
    <!ELEMENT 移动通信卡 (手机卡*)>]>
<移动通信卡>
    <手机卡 id号="S001" 品牌="中国移动">
        <卡品>1+1 亲情卡</卡品>
        <号码>15801056876</号码>
        <生效日期>2010-1-8</生效日期>
        <亲情卡 id号="S001" 品牌="中国移动" 属地="北京">
            <卡品>动感地带</卡品>
            <号码>15836897766</号码>
            <生效日期>2010-3-11</生效日期>
        </亲情卡>
    </手机卡>
    <手机卡 id号="S003" 品牌="中国移动">
        <卡品>家庭特别亲情卡</卡品>
        <号码>15823658667</号码>
        <生效日期>2010-2-18</生效日期>
        <亲情卡 id号="S004" 品牌="中国移动" 属地="北京">
            <卡品>超级市话卡</卡品>
            <号码>15877779988</号码>
            <生效日期>2010-3-16</生效日期>
        </亲情卡>
    </手机卡>
</移动通信卡>
```

在 XMLwriter 中进行有效性验证的结果如图 3-5 所示。

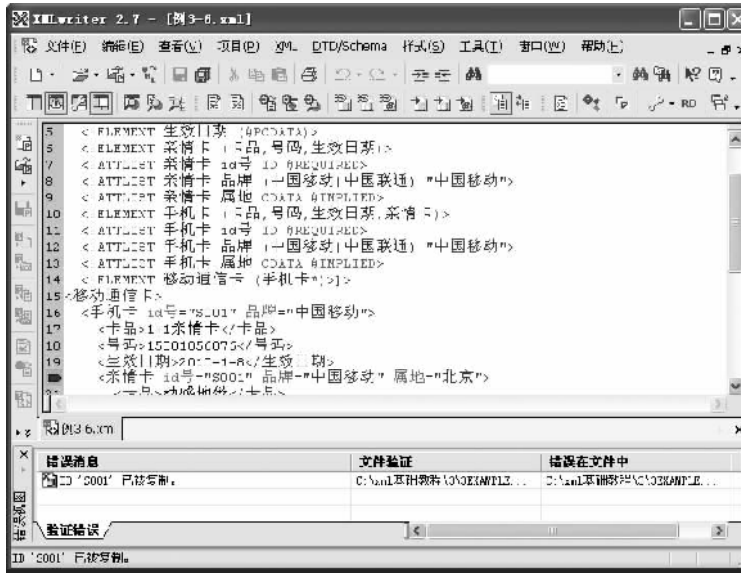

图 3-5 【例 3-6】有效性验证结果

XMLwriter 有效性检查结果显示，第 20 行的错误信息是“ID 'S001' 已被复制”。说明＜亲情卡＞元素的“id 号”属性的值与其他元素的 ID 类型的属性值重复了，是不符合 DTD 规范的 XML 文档要求的。

使用 ID 类型的属性需要注意的是：

(1)一个元素只能声明一个 ID 类型的属性。

(2)因为 ID 类型的属性要求属性值在一个 XML 文档内唯一，而 FIXED 类型要求属性的属性值都是声明时的值，在一个 XML 文档内都是一样的，因此 ID 类型的属性与 FIXED 类型缺省值不能同时出现，否则是非法的。

例如，下面的属性声明中属性“id 号”和“品牌”都是 ID 类型，因此是非法的。属性“品牌”既声明为 ID 类型，同时其缺省值又声明为 FIXED，因此也是错误的。

```
＜!ATTLIST 手机卡 id 号 ID ＃REQUIRED
          品牌 ID ＃FIXED
          属地 CDATA ＃IMPLIED＞
```

3. IDREF 类型

ID 是 identifier 的缩写，IDREF 是 identifier reference 的缩写。IDREF 类型属性的属性值必须是其他元素的 ID 类型属性的属性值，而且此 ID 类型的属性值必须在文档的其他位置已经设定过。

【例 3-7】 IDREF 属性类型示例。

```
＜?xml version＝"1.0" encoding＝"gb2312"?＞
＜!DOCTYPE 移动通信卡 [
    ＜!ELEMENT 卡品 (＃PCDATA)＞
    ＜!ELEMENT 号码 (＃PCDATA)＞
    ＜!ELEMENT 生效日期 (＃PCDATA)＞
    ＜!ELEMENT 亲情卡 (卡品,号码,生效日期)＞
    ＜!ATTLIST 亲情卡 id 号 ID ＃REQUIRED＞
    ＜!ATTLIST 亲情卡 亲情卡 id 号 IDREF ＃REQUIRED＞
    ＜!ATTLIST 亲情卡 品牌 (中国移动|中国联通) ＃REQUIRED＞
    ＜!ATTLIST 亲情卡 属地 CDATA ＃REQUIRED＞
    ＜!ELEMENT 手机卡 (卡品,号码,生效日期,亲情卡)＞
    ＜!ATTLIST 手机卡 id 号 ID ＃REQUIRED＞
    ＜!ATTLIST 手机卡 亲情卡 id 号 IDREF ＃REQUIRED＞
    ＜!ATTLIST 手机卡 品牌 (中国移动|中国联通) "中国移动"＞
    ＜!ATTLIST 手机卡 属地 CDATA ＃IMPLIED＞
    ＜!ELEMENT 移动通信卡 (手机卡＊)＞]＞
＜移动通信卡＞
    ＜手机卡 id 号＝"S001" 亲情卡 id 号＝"S002" 品牌＝"中国移动"＞
        ＜卡品＞1＋1 亲情卡＜/卡品＞
        ＜号码＞15801056876＜/号码＞
        ＜生效日期＞2010-1-8＜/生效日期＞
        ＜亲情卡 id 号＝"S002" 亲情卡 id 号＝"S001" 品牌＝"中国移动" 属地
```

```
="北京">
            <卡品>动感地带</卡品>
            <号码>15836897766</号码>
            <生效日期>2010-3-11</生效日期>
        </亲情卡>
    </手机卡>
    <手机卡 id 号="S003" 亲情卡 id 号="S004" 品牌="中国移动">
        <卡品>家庭特别亲情卡</卡品>
        <号码>15823658667</号码>
        <生效日期>2010-2-18</生效日期>
        <亲情卡 id 号="S004" 亲情卡 id 号="S003" 品牌="中国移动" 属地
="北京">
            <卡品>超级市话卡</卡品>
            <号码>15877779988</号码>
            <生效日期>2010-3-16</生效日期>
        </亲情卡>
    </手机卡>
</移动通信卡>
```

在 XMLwriter 中进行有效性验证结果如图 3-6 所示。

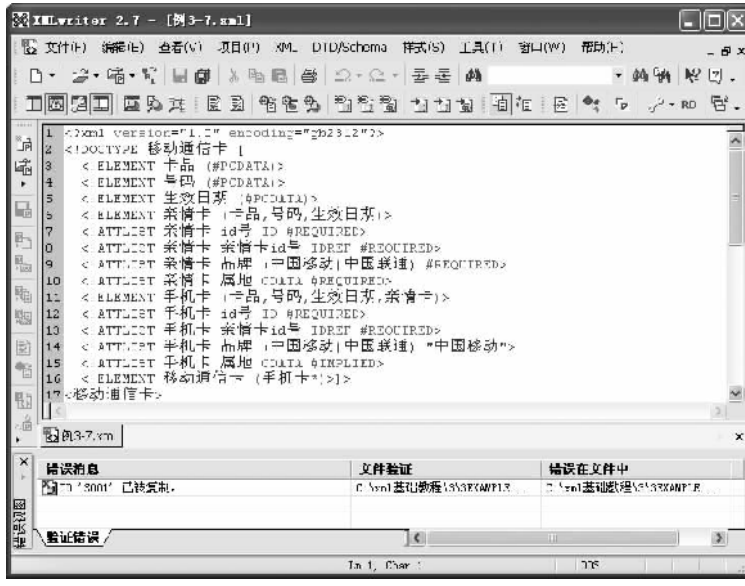

图 3-6 【例 3-7】有效性验证结果

若把【例 3-7】中的语句

<亲情卡 id 号="S004" 亲情卡 id 号="S003" 品牌="中国移动" 属地="北京">

改为

<亲情卡 id 号="S004" 亲情卡 id 号="S005" 品牌="中国移动" 属地="北京">

更改后的程序再通过 XMLwriter 检查 XML 文档的有效性时，就会在其状态栏提示“带

有名称空间"的属性'亲情卡 id 号' 引用文档中没有定义的 ID 'S005'"。此时【例 3-7】的 XML 文档是无效的文档，如图 3-7 所示。

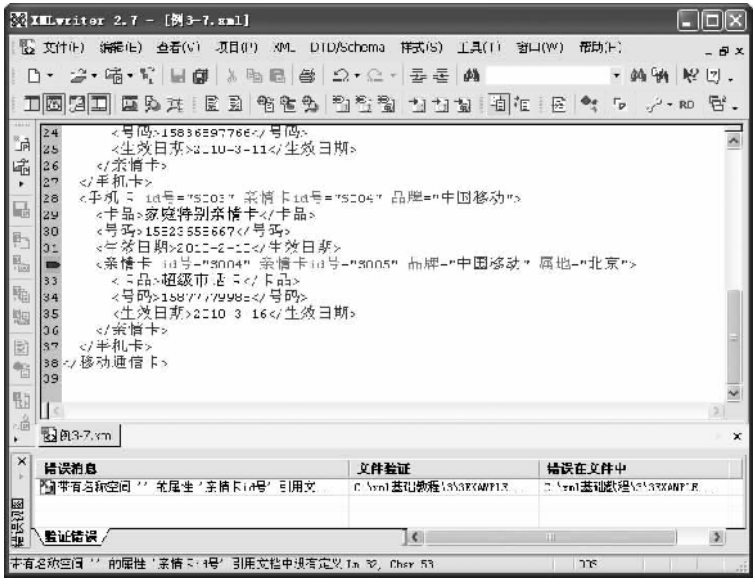

图 3-7　修改文档后的验证结果

4. IDREFS 类型

IDREFS 类型的属性与 IDREF 类型的属性是有关系的。IDREFS 类型属性的属性值可以有多个，每一个都是在文档中其他位置设定过 ID 类型的属性值，多个属性值之间用空格分隔。

【例 3-8】 IDREFS 属性类型示例。

```
<?xml version="1.0" encoding="gb2312"?>
<!DOCTYPE 移动通信卡 [
    <!ELEMENT 卡品 (#PCDATA)>
    <!ELEMENT 号码 (#PCDATA)>
    <!ELEMENT 生效日期 (#PCDATA)>
    <!ELEMENT 手机卡 (卡品,号码,生效日期)>
    <!ATTLIST 手机卡 id号 ID #REQUIRED>
    <!ATTLIST 手机卡 亲情卡 id号 IDREFS #IMPLIED>
    <!ATTLIST 手机卡 品牌 (中国移动|中国联通) "中国移动">
    <!ATTLIST 手机卡 属地 CDATA #IMPLIED>
    <!ELEMENT 移动通信卡 (手机卡*)>
]>
<移动通信卡>
    <手机卡 id号="S001" 亲情卡 id号="S002 S003 S004" 品牌="中国移动">
        <卡品>1+1 亲情卡</卡品>
        <号码>15801056876</号码>
        <生效日期>2010-1-8</生效日期>
```

```
        </手机卡>
        <手机卡 id 号="S002" 亲情卡 id 号="S001 S004" 品牌="中国移动" 属地
="北京">
            <卡品>动感地带</卡品>
            <号码>15836897766</号码>
            <生效日期>2010-3-11</生效日期>
        </手机卡>
        <手机卡 id 号="S003" 亲情卡 id 号="S002 S004" 品牌="中国移动">
            <卡品>家庭特别亲情卡</卡品>
            <号码>15823658667</号码>
            <生效日期>2010-2-18</生效日期>
        </手机卡>
        <手机卡 id 号="S004" 亲情卡 id 号="S001 S003" 品牌="中国移动" 属地="
北京">
            <卡品>超级市话卡</卡品>
            <号码>15877779988</号码>
            <生效日期>2010-3-16</生效日期>
        </手机卡>
    </移动通信卡>
```

在 XMLwriter 中进行有效性验证，结果如图 3-8 所示。

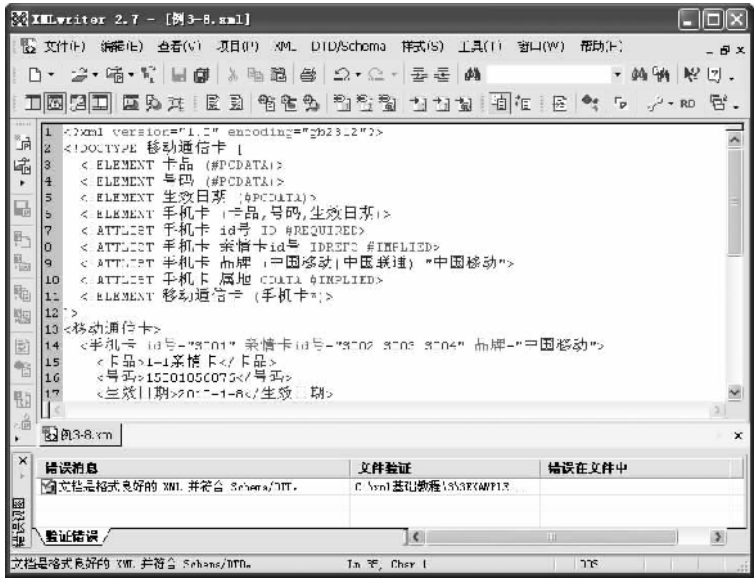

图 3-8 IDREFS 属性类型示例

5. ENTITY 类型

ENTITY 类型的属性值必须是一个外部实体，而这个外部实体为二进制形式的文件，如 GIF、TIF、JPEG、MP3、AVI、RMVB 等格式的文件。ENTITY 类型的属性提供了将外部的不可析实体链接到 XML 文档的方式。

【例 3-9】 ENTITY 属性类型示例。

```
<?xml version="1.0" encoding="gb2312"?>
<!DOCTYPE 移动通信卡 [
    <!ELEMENT 卡品 (#PCDATA)>
    <!ELEMENT 号码 (#PCDATA)>
    <!ELEMENT 图片 EMPTY>
    <!ELEMENT 生效日期 (#PCDATA)>
    <!ELEMENT 手机卡 (卡品,号码,图片,生效日期)>
    <!ATTLIST 手机卡 id 号 ID #REQUIRED>
    <!ATTLIST 手机卡 品牌 (中国移动|中国联通) "中国移动">
    <!ATTLIST 手机卡 属地 CDATA #IMPLIED>
    <!ATTLIST 图片 图片文件 ENTITY #IMPLIED>
    <!ENTITY pic01 "picture01.jpg">
    <!ENTITY pic02 "picture02.jpg">
    <!ENTITY pic03 "picture03.jpg">
    <!ELEMENT 移动通信卡 (手机卡*)>]>
<移动通信卡>
    <手机卡 id 号="S001" 品牌="中国移动">
        <卡品>1+1 亲情卡</卡品>
        <号码>15801056876</号码>
        <图片 图片文件="&pic01;"/>
        <生效日期>2010-1-8</生效日期>
    </手机卡>
    <手机卡 id 号="S002" 属地="北京">
        <卡品>动感地带</卡品>
        <号码>15836897766</号码>
        <图片 图片文件="&pic02;"/>
        <生效日期>2010-3-11</生效日期>
    </手机卡>
    <手机卡 id 号="S003" 品牌="中国移动">
        <卡品>家庭特别亲情卡</卡品>
        <号码>15823658667</号码>
        <图片 图片文件="&pic03;"/>
        <生效日期>2010-2-18</生效日期>
    </手机卡>
</移动通信卡>
```

由于目前的大多数解析器都不支持这种非 XML 数据的引用，所以无法提供此例的截图，这种情况还包括下面的 ENTITIES 属性类型和 NOTATION 属性类型。

6. ENTITIES 类型

ENTITIES 类型和 ENTITY 类型的关系，IDREFS 类型和 IDREF 类型的关系，以及与后面要介绍的 NMTOKENS 类型和 NMTOKEN 类型的关系是一样的，实际上，前者就是后者的复数形式。ENTITIES 类型的属性值由多个外部的不可解析实体组成，名称之间用空格分隔。

【例 3-10】 ENTITIES 属性类型示例。

```
<?xml version="1.0" encoding="gb2312"?>
<!DOCTYPE 移动通信卡 [
    <!ELEMENT 卡品 (#PCDATA)>
    <!ELEMENT 号码 (#PCDATA)>
    <!ELEMENT 图片 EMPTY>
    <!ELEMENT 生效日期 (#PCDATA)>
    <!ELEMENT 手机卡 (卡品,号码,图片,生效日期)>
    <!ATTLIST 手机卡 id 号 ID #REQUIRED>
    <!ATTLIST 手机卡 品牌 (中国移动|中国联通) "中国移动">
    <!ATTLIST 手机卡 属地 CDATA #IMPLIED>
    <!ATTLIST 图片 图片文件 ENTITIES #IMPLIED>
    <!ENTITY pic01 "picture01.jpg">
    <!ENTITY pic02 "picture02.jpg">
    <!ENTITY pic03 "picture03.jpg">
    <!ELEMENT 移动通信卡 (手机卡*)>
]>
<移动通信卡>
    <手机卡 id 号="S001" 品牌="中国移动">
        <卡品>1+1 亲情卡</卡品>
        <号码>15801056876</号码>
        <图片 图片文件="&pic01; &pic03;"/>
        <生效日期>2010-1-8</生效日期>
    </手机卡>
    <手机卡 id 号="S002" 属地="北京">
        <卡品>动感地带</卡品>
        <号码>15836897766</号码>
        <图片 图片文件="&pic02;"/>
        <生效日期>2010-3-11</生效日期>
    </手机卡>
    <手机卡 id 号="S003" 品牌="中国移动">
        <卡品>家庭特别亲情卡</卡品>
        <号码>15823658667</号码>
        <图片 图片文件="&pic02; &pic03;"/>
```

```
        <生效日期>2010-2-18</生效日期>
    </手机卡>
</移动通信卡>
```

7. NOTATION 类型

在 XML 文档中引入外部的不可解析实体后，XML 解析器是无法解析这些二进制文件的。因此 XML 规定，通过 NOTATION 类型的属性可以指定一个其他的应用程序来识别和处理这些二进制文件，但是要具有这种能力需要和 NOTATION 声明结合起来。具体来讲，NOTATION 声明定义了一个外部应用程序，NOTATION 类型的属性引用了这个外部应用程序的定义，从而可达到为不可解析的数据指定一个应用程序的目的。

【例 3-11】 NOTATION 属性类型示例。

```
<?xml version="1.0" encoding="gb2312"?>
<!DOCTYPE 移动通信卡[
    <!ELEMENT 卡品 (#PCDATA)>
    <!ELEMENT 号码 (#PCDATA)>
    <!NOTATION ACDSee SYSTEM "Program Files\ACD\ACDSee\ACDSee.exe">
    <!NOTATION Photoshop SYSTEM "Program Files\Photoshop 8.0\photoshop.exe">
    <!NOTATION Fireworks SYSTEM "Program Files\Macromedia\Fireworks.exe">
    <!ELEMENT 图片 EMPTY>
    <!ELEMENT 生效日期 (#PCDATA)>
    <!ELEMENT 手机卡 (卡品,号码,图片,生效日期)>
    <!ATTLIST 手机卡 id号 ID #REQUIRED>
    <!ATTLIST 手机卡 品牌 (中国移动|中国联通) "中国移动">
    <!ATTLIST 手机卡 属地 CDATA #IMPLIED>
    <!ATTLIST 图片 图片文件 ENTITIES #IMPLIED>
    <!ENTITY pic01 "picture01.jpg">
    <!ENTITY pic02 "picture02.jpg">
    <!ENTITY pic03 "picture03.jpg">
    <!ELEMENT 移动通信卡 (手机卡*)>
]>
<移动通信卡>
    <手机卡 id号="S001" 品牌="中国移动">
        <卡品>1+1 亲情卡</卡品>
        <号码>15801056876</号码>
        <图片 图片文件="&pic01; &pic03;" 显示程序="ACDSee"/>
        <生效日期>2010-1-8</生效日期>
    </手机卡>
    <手机卡 id号="S002" 属地="北京">
        <卡品>动感地带</卡品>
        <号码>15836897766</号码>
        <图片 图片文件="&pic02;" 显示程序="Photoshop"/>
```

```
        <生效日期>2010-3-11</生效日期>
    </手机卡>
    <手机卡 id 号="S003" 品牌="中国移动">
        <卡品>家庭特别亲情卡</卡品>
        <号码>15823658667</号码>
        <图片 图片文件="&pic02; &pic03;" 显示程序="Fireworks"/>
        <生效日期>2010-2-18</生效日期>
    </手机卡>
</移动通信卡>
```

8. NMTOKEN 类型

NMTOKEN 类型的属性值只能是英文字母、数字、“.”、“:”、“-”、“_”等字符。若设置了字符集，也可由该字符集的特定字符构成，如设定的字符集为“GB2312”，就可使用汉字。这与标记的命名规则是相同的，但不同的是，此类型属性值不一定非得以英文字母、“_”和特定字符集的特定字符开头，其开头的字符可以是除了“:”的所有字符。冒号“:”可以出现在字符串的中间，但为了避免与命名域混淆，一般不提倡这样使用。

在设计 XML 文档的处理程序时，NMTOKEN 类型将会发挥出它的作用。由于 NMTOKEN 类型对合法字符的要求非常严格，使得它在 Java、JavaScript、VBScript 等许多编程语言中，获得的都是合法数据，所以将会减少处理程序的许多麻烦。

9. NMTOKENS 类型

NMTOKENS 类型是 NMTOKEN 类型的复数形式，但是在 NMTOKENS 类型中，空格也属于合法字符。

NMTOKENS 类型允许属性有多个属性值，属性值之间用空格分隔，并且要求它们在同一对引号内，例如：

```
<!ELEMENT 手机卡(卡品,号码,生效日期)>
<!ATTLIST 手机卡 id 号 ID #REQUIRED>
<!ATTLIST 手机卡 品牌(中国移动|中国联通) "中国移动">
<!ATTLIST 手机卡 属地 NMTOKENS "北京">
```

下面的 XML 文档片段中，手机卡元素的“属地”属性的属性值是合法的。

```
<手机卡 id 号="S001" 品牌="中国移动" 属地="北京 上海">
    <卡品>1+1 亲情卡</卡品>
    <号码>15801056876</号码>
    <生效日期>2010-1-8</生效日期>
</手机卡>
```

10. Enumerated(枚举)类型

如果属性值不是一个任意的字符串，而只可能是几个指定的值中的一个，例如，一般的“性别”属性只可能是“男”和“女”中的一个，不可能是其他值，就可以将这个属性声明为 Enumerated(枚举)类型。

这里的 Enumerated 不是一个关键字。实际上，将一个属性设置为 Enumerated 类型，就

是在属性声明语句的“类型”位置列出用“|”分隔的可选值。例如：

```
<!ATTLIST 成员 性别 (男|女) #REQUIRED>
```

Enumerated 类型的属性也可以设置缺省值，例如：

```
<!ATTLIST 成员 性别 (男|女) "男">
```

【例 3-12】 Enumerated 属性类型示例。

```
<?xml version="1.0" encoding="GB2312"?>
<!DOCTYPE 移动通信卡[
    <!ELEMENT 卡品 (#PCDATA)>
    <!ELEMENT 号码 (#PCDATA)>
    <!ELEMENT 生效日期 (#PCDATA)>
    <!ELEMENT 手机卡 (卡品,号码,生效日期)>
    <!ATTLIST 手机卡 id 号 ID #REQUIRED>
    <!ATTLIST 手机卡 品牌 (中国移动|中国联通) "中国移动">
    <!ATTLIST 手机卡 属地 CDATA #IMPLIED>
    <!ELEMENT 移动通信卡 (手机卡*)>
]>
<移动通信卡>
    <手机卡 id 号="S001" 品牌="中国移动">
        <卡品>1+1 亲情卡</卡品>
        <号码>15801056876</号码>
        <生效日期>2010-1-8</生效日期>
    </手机卡>
    <手机卡 id 号="S002" 品牌="中国移动">
        <卡品>家庭特别亲情卡</卡品>
        <号码>15823658667</号码>
        <生效日期>2010-2-18</生效日期>
    </手机卡>
    <手机卡 id 号="S003" 品牌="中国电信">
        <卡品>超级市话卡</卡品>
        <号码>15823333667</号码>
        <生效日期>2010-2-18</生效日期>
    </手机卡>
</移动通信卡>
```

在 XMLwriter 中进行有效性验证，结果如图 3-9 所示。

XMLwriter 有效性验证结果显示：“根据 DTD/Schema，属性‘品牌’有一个无效值。”本例中，元素“手机卡”的属性“品牌”是 Enumerated 类型，说明此属性值只能取“中国移动”和“中国联通”两者之一，而不能取其他值，若没有指出此属性，则取值为“中国移动”。例中有一“品牌”的属性值取为“中国电信”，所以是非法的。

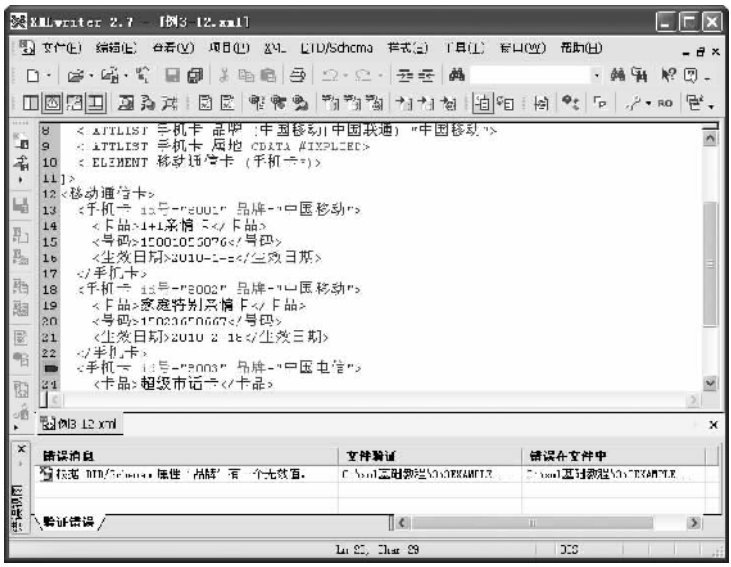

图 3-9 【例 3-12】在 XMLwriter 中进行有效性验证结果

3.4 内部 DTD 和外部 DTD

DTD 分为内部 DTD 和外部 DTD,使用它们可以使 XML 文档更加完善。

3.4.1 内部 DTD

所有的文档都是由前导说明和文档体构成的。前导说明中包含了 XML 声明,而文档体中则是具体的数据信息,文档中还可以包含一些处理指示,如在前导说明中包含 DTD。

使用 DTD 的最简单的方法是在 XML 文档的前导说明部分加入一个 DTD 描述,加入的位置是紧接在 XML 处理指示之后。

一个包含 DTD 的 XML 文档的结构为:

```
<?xml version="1.0" encoding="GB2312" standalone="yes"?>
<!DOCTYPE 根元素名[
        元素描述
]>
文档体
```

这样就定义了一个文档,它以 DOCTYPE 中规定的根元素名作为其根元素的名称。

【例 3-13】 内部 DTD 应用示例。

```
<?xml version="1.0" encoding="GB2312" standalone="yes"?>
<!DOCTYPE 联系人列表[
        <!ELEMENT 联系人列表 (联系人*)>
        <!ELEMENT 联系人 (姓名,ID,公司,EMAIL,电话,地址)>
        <!ELEMENT 地址 (街道,城市,省份)>
```

```
        <!ELEMENT 姓名 (#PCDATA)>
        <!ELEMENT ID (#PCDATA)>
        <!ELEMENT 公司 (#PCDATA)>
        <!ELEMENT EMAIL (#PCDATA)>
        <!ELEMENT 电话 (#PCDATA)>
        <!ELEMENT 街道 (#PCDATA)>
        <!ELEMENT 城市 (#PCDATA)>
        <!ELEMENT 省份 (#PCDATA)>]>
<联系人列表>
    <联系人>
        <姓名>王明</姓名>
        <ID>0001</ID>
        <公司>华硕股份有限公司北京分公司</公司>
        <EMAIL>wang@163.com</EMAIL>
        <电话>(010)86437688</电话>
        <地址>
            <街道>五里河 7689 号</街道>
            <城市>北京市</城市>
            <省份>北京</省份>
        </地址>
    </联系人>
    <联系人>
        <姓名>赵晓</姓名>
        <ID>0002</ID>
        <公司>联想股份有限公司上海分公司</公司>
        <EMAIL>zhaoxiao@sina.com</EMAIL>
        <电话>(021)64370998</电话>
        <地址>
            <街道>南京路 675 号</街道>
            <城市>上海市</城市>
            <省份>上海</省份>
        </地址>
    </联系人>
</联系人列表>
```

可以想象，若每个 XML 文档都如【例 3-13】那样在 XML 文档中加入一段 DTD 定义，是比较烦琐的，且对同一个 DTD 适合于多个 XML 文档的情况，重复添加 DTD 代码不仅浪费精力而且降低效率，一旦需要对 DTD 进行修改，就需要重复地去修改每个包含这一 DTD 的 XML 文档。因此，在实际的 XML 应用中，更多的情况是许多个 XML 文档需要定义一个相同的 DTD，这时可将 DTD 单独生成一个 DTD 文档文件，若某一个 XML 文档要使用该 DTD 时，只需在文档中引用已建立好的 DTD 文档文件就可以了，这个 DTD 文档文件就是

外部 DTD。

3.4.2 外部 DTD

可以在“格式良好的”XML 文档中定义使用 DTD,也就是使用内部 DTD;也可以将外部 DTD 作为一个外部文档被引用,而且需在 XML 文档中说明,如 standalone="no"。

使用外部 DTD 的最大好处是 DTD 可以方便高效地被多个 XML 文档共享。这样,只要写一个 DTD 文档,就可以供多个 XML 文档引用。事实上,许多国际组织就是通过外部 DTD 来统一它们的数据交换格式的。这样不仅大大减少了输入代码的工作量,而且还能保证修改 DTD 时,不用去一一修改每个引用了它的 XML 文档,只需修改一个公用的 DTD 文档即可。不过这样做需要 DTD 的改动必须向后兼容。

为了引用一个外部 DTD,必须修改 XML 声明和 DOCTYPE 声明。

首先,XML 声明中必须把 standalone 属性的属性值设置为 no,用以说明这个 XML 文档不是自成一体的。代码如下:

```
<?xml version="1.0" encoding="GB2312" standalone="no"?>
```

其次,在 DOCTYPE 声明中,应该加入 SYSTEM 属性,代码格式如下:

```
<!DOCTYPE 根元素名 SYSTEM "外部 DTD 文件的 URL">
```

在【例 3-13】中,若使用外部 DTD,DOCTYPE 声明语句应该为:

```
<!DOCTYPE 联系人列表 SYSTEM "http://www.mydomain.com/dtds/mydtd.dtd">
```

在此代码中,URL 使用了绝对路径,当然也可以使用相对路径,例如:

```
<!DOCTYPE 联系人列表 SYSTEM "mydtd.dtd">
```

这段代码说明了 mydtd.dtd 这个 DTD 文件和引用它的 XML 文件放在了同一个目录下。或者这个 DTD 文件还可能放在 XML 文件的父目录下的其他子目录中,如 dtds 中,可表示为:

```
<!DOCTYPE 联系人列表 SYSTEM ".../dtds/mydtd.dtd">
```

使用外部 DTD,可以方便地把 DTD 文档从 XML 文档中剥离出来,粘贴到另一个文件 mydtd.dtd 中。因此,要使用一个 XML 文档除了需要一个有效的 XML 文件外,还需要一个 DTD 文件。

【例 3-14】 外部 DTD 应用示例。

先建立一个 DTD 文件“例 3-14.dtd”:

```
<?xml version="1.0" encoding="GB2312"?>
<!ELEMENT 联系人列表 (联系人*)>
<!ELEMENT 联系人 (姓名,ID,公司,EMAIL,电话,地址)>
<!ELEMENT 地址 (街道,城市,省份)>
<!ELEMENT 姓名 (#PCDATA)>
<!ELEMENT ID (#PCDATA)>
<!ELEMENT 公司 (#PCDATA)>
<!ELEMENT EMAIL (#PCDATA)>
<!ELEMENT 电话 (#PCDATA)>
<!ELEMENT 街道 (#PCDATA)>
<!ELEMENT 城市 (#PCDATA)>
```

```
<!ELEMENT 省份 (#PCDATA)>
```

再建立一个 XML 文档文件“例 3-14.xml”：

```
<?xml version="1.0" encoding="GB2312" standalone="no"?>
<!DOCTYPE 联系人列表 SYSTEM "例 3-14.dtd">
<联系人列表>
    <联系人>
        <姓名>王明</姓名>
        <ID>0001</ID>
        <公司>华硕股份有限公司北京分公司</公司>
        <EMAIL>wang@163.com</EMAIL>
        <电话>(010)86437688</电话>
        <地址>
            <街道>五里河 7689 号</街道>
            <城市>北京市</城市>
            <省份>北京</省份>
        </地址>
    </联系人>
    <联系人>
        <姓名>赵晓</姓名>
        <ID>0002</ID>
        <公司>联想股份有限公司上海分公司</公司>
        <EMAIL>zhaoxiao@sina.com</EMAIL>
        <电话>(021)64370998</电话>
        <地址>
            <街道>南京路 675 号</街道>
            <城市>上海市</城市>
            <省份>上海</省份>
        </地址>
    </联系人>
</联系人列表>
```

需要说明的是，在 XML 文档的前导说明中，如果既包括内部 DTD，又包括外部 DTD，则优先使用内部 DTD，也就是说内部 DTD 具有更高的优先级，内部 DTD 中含有的实体说明、属性声明等优先于外部 DTD 中相对应的声明。另外，为每个 XML 文档单独定义的 DTD 可以推广成为一个系统内共享的 DTD，系统内共享的 DTD 还可以进一步推广到系统或组织外部，扩大为行业内甚至公众使用的 DTD，称之为公共 DTD。

使用外部 DTD 时，要在 DOCTYPE 中使用关键字 SYSTEM。实际上，SYSTEM 不是引用外部 DTD 的唯一方法，这个关键字主要用于引用一个作者或组织所编写的众多 XML 文档中通用的 DTD。还存在另一种外部 DTD，一个由权威机构制定的、提供给特定行业或公众使用的 DTD，引用外部 DTD 的另一个方法是使用 PUBLIC 关键字来引用公开给公众使用的 DTD。

当使用 PUBLIC 关键字引用一个 DTD 时，这个外部 DTD 还需要得到公共 DTD，其一般形式为：

<!DOCTYPE 根元素 PUBLIC "外部 DTD 名称" "外部 DTD 的 URL">

因此，在联系人列表的例子中，若使用公共 DTD 时，DOCTYPE 声明如下：

<!DOCTYPE 根元素 PUBLIC "联系人 DTD" "http://www.mydomain.com/dtds/mydtd.dtd">

公共 DTD 标识名的命名规则和 XML 文档的命名规则稍有不同。具体地说，DTD 名称只能包含字母、数字、空格和下面的符号：_、%、$、#、@、(、)、+、:、=、/、!、*、;、?。同时，DTD 名称还必须符合其他一些标准规定。例如，ISO 标准的 DTD 以 ISO 开头；被改进的非 ISO 标准的 DTD 以加号“+”开头；未被改进的非 ISO 标准的 DTD 以减号“-”开头。无论是哪一种情况，开始部分后面都跟着两个斜杠“//”及 DTD 所有者的名称，在这个名称之后又是两个斜杠“//”，然后是 DTD 所描述的文档的类型。最后，一对斜杠之后是语言的种类（参见 ISO 639）。

在联系人列表的例子中，若引用一个公共 DTD 时，DOCTYPE 声明如下：

<!DOCTYPE 联系人列表 PUBLIC "-//My DTD//Contact Data//CN" "http://www.mydomain.com/dtds/mydtd.dtd">

这看上去很复杂，不过对于 DTD 的命名通常不是其引用者的任务，XML 文档的编写者只要在自己的文档中把事先定义好的 DTD 名称放在相应的位置中就可以了。

实　　训

一、实训目的

掌握 DTD 的基本语法。

二、实训要求

本实训几乎涉及 DTD 的所有基本语法知识，因此在动手实践时，应先复习本章的所有内容，并掌握使用 DTD 验证 XML 文档有效性的方法。

三、实训内容

本实训是在第 2 章实训的基础上，为 XML 文档增加 DTD，用以建立一个有关单位职工考勤的 XML 文档，并进行 DTD 有效性验证。

四、实训设计

在某单位职工考勤系统的设计中，要设计一个根元素为<职工考勤表>的 XML 文档，用到了“人员信息表”、“单位信息表”和“考勤信息表”3 个表单，如表 2-1 至表 2-3 所示。

在本实训中，根元素为<职工考勤表>，下面有 3 个子元素分别是<人员信息表>、<单位信息表>和<考勤信息表>，将所有的信息进行分类归档，以使 XML 文档的数据结构化，并使用内部 DTD 或外部 DTD 为所有元素定义数据类型和其他约束来规范 XML 文档数据。

【例 3-15】 该单位职工考勤系统 XML 文档是在第 2 章实训的基础上稍加修改，并采用了内部 DTD，增加了 DTD 代码。若使用外部 DTD 的话，读者可参考本章 3.4.2 的内容。

```
<?xml version="1.0" encoding="gb2312"?>
<!--文件名:例 3-15.xml-->
<!DOCTYPE 职工考勤表[
<!ENTITY male "男">
<!ENTITY female "女">
<!ELEMENT  姓名  (#PCDATA)>
<!ELEMENT  性别  (#PCDATA)>
<!ELEMENT  年龄  (#PCDATA)>
<!ELEMENT  籍贯  (#PCDATA)>
<!ELEMENT  人员信息(姓名,性别,年龄,籍贯)>
<!ELEMENT  人员信息表(人员信息*)>
<!ATTLIST  人员信息  工号  ID  #REQUIRED>
<!ELEMENT  单位名称(#PCDATA)>
<!ELEMENT  类别(#PCDATA)>
<!ELEMENT  位置(#PCDATA)>
<!ELEMENT  人员(#PCDATA)>
<!ELEMENT  电话(#PCDATA)>
<!ELEMENT  单位信息(单位名称,类别,位置,人员,电话)>
<!ATTLIST  单位信息  单位编号  ID  #REQUIRED>
<!ELEMENT  单位信息表(单位信息*)>
<!ELEMENT  考勤日期(#PCDATA)>
<!ELEMENT  是否在岗(#PCDATA)>
<!ELEMENT  考勤信息  (考勤日期,是否在岗)>
<!ATTLIST  考勤信息  工号  IDREF  #REQUIRED
                     部门编号  IDREF  #REQUIRED>
<!ELEMENT  考勤信息表  (考勤信息*)>
<!ELEMENT  职工考勤表  (人员信息表,单位信息表,考勤信息表)>
]>
<职工考勤表>
<!--这是单位职工考勤系统的数据-->
    <!--这是人员信息表-->
    <人员信息表>
        <人员信息 工号="001">
            <姓名>张三</姓名>
            <性别>&male;</性别>
            <年龄>20</年龄>
            <籍贯>南阳</籍贯>
        </人员信息>
        <人员信息 工号="002">
            <姓名>李四</姓名>
```

```
            <性别>&female;</性别>
            <年龄>22</年龄>
            <籍贯>信阳</籍贯>
        </人员信息>
    </人员信息表>
    <!--这是单位信息表-->
    <单位信息表>
        <单位信息 单位编号="P001">
            <单位名称>办公室</单位名称>
            <类别>行政</类别>
            <位置>C201</位置>
            <人员>5</人员>
            <电话>1111111</电话>
        </单位信息>
        <单位信息 单位编号="P002">
            <单位名称>计科系</单位名称>
            <类别>院系</类别>
            <位置>B202</位置>
            <人员>24</人员>
            <电话>2222222</电话>
        </单位信息>
        <单位信息 单位编号="P003">
            <部门名称>图书馆</部门名称>
            <类别>教辅</类别>
            <位置>2 号楼</位置>
            <人员>50</人员>
            <电话>3333333</电话>
        </单位信息>
    </单位信息表>
    <!--这是考勤信息表-->
    <考勤信息表>
        <考勤信息 工号="001" 单位编号="P001">
            <考勤日期>2010-4-29</考勤日期>
            <是否在岗>在</是否在岗>
        </考勤信息>
        <考勤信息 工号="002" 单位编号="P002">
            <考勤日期>2010-4-29</考勤日期>
            <是否在岗>否</是否在岗>
        </考勤信息>
    </考勤信息表>
</职工考勤表>
```

【例 3-15】在 XMLwriter 中验证 DTD 有效性的结果如图 3-10 所示。

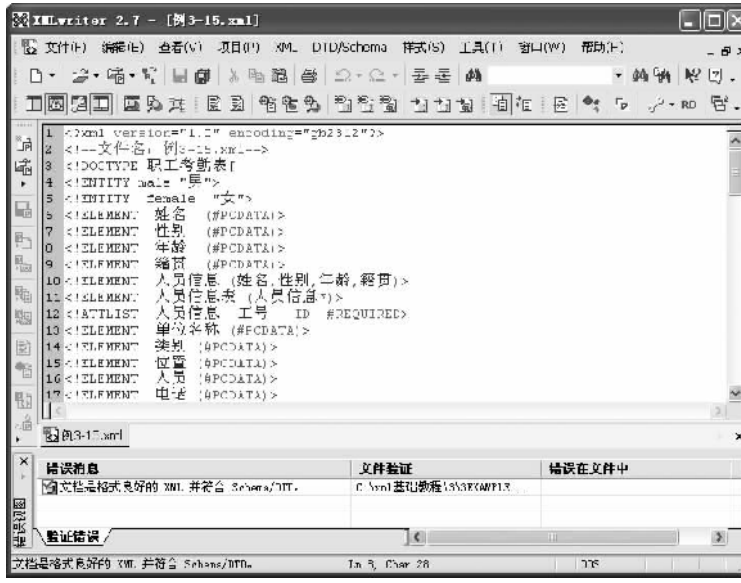

图 3-10　【例 3-15】在 XMLwriter 中的显示结果

习　题　3

1. 什么是 XML 文档的有效性？什么是文档类型定义？
2. 简述 XML 文档的有效性与 DTD 的关系以及 DTD 在其相应的 XML 文档中的作用。
3. 在 DTD 中，如何定义元素类型？简述元素的内容模式及每一种类型的表示方法。
4. 简述在 DTD 中控制子元素的出现次数的方法。
5. 在 DTD 中，如何定义元素属性的类型？常用的属性类型有哪些？
6. 什么是内部 DTD 和外部 DTD?
7. 如何在 XML 文档中引用一个外部 DTD?

第 7 章　使用 XSL 转换 XML 文档

知识目标

◎ 掌握 XSL 的基本概念及其组成
◎ 了解 CSS 和 XSL 的区别
◎ 掌握 XSL 样式表元素及命名空间的适用范围
◎ 掌握 XSL 模板规则和调用方式
◎ 熟练掌握用不同的方式选择结点的方法
◎ 掌握 XSL 常见的控制指令及其使用方法
◎ 熟练掌握 XSL 与 CSS 的结合使用

技能目标

◎ 熟练运用 XSL 模板
◎ 根据不同情况选择合适的结点选择方法
◎ 运用 XSL 控制指令设计复杂的显示规则
◎ XSL 与 CSS 的结合使用

可扩展样式表语言(extensible stylesheet language,XSL)是专门针对 XML 文档的,并为之提供一个功能强大而又容易使用的语法样式。XSL 实际上包含 3 种语言:XSLT,一种用于转换 XML 文档的语言,它将一个 XML 文档(输入或源文档)转换为另一个 XML 文档(输出或结果文档);XPath,一种用于在 XML 文档中导航的语言;XSL-FO,一种用于格式化 XML 文档的语言。

XSL 的出现使得 XML 格式的文档不会出现 HTML 文档结构混杂、内容繁乱等问题,XML 的编写者也可以集中精力于数据本身,而不受显示方式等细枝末节的影响。定义不同的样式表可以使相同的数据呈现出不同的显示外观,从而适合于不同的应用,甚至能够在不同的显示设备上显示。

7.1　XSL 概述

XSL 最早由 W3C 于 1999 年正式提出,它定义了如何转换和表示 XML 文档,是 XML 文档的首选显示样式语言。XSL 比 CSS 样式表功能强大得多,当然也要复杂得多。XSL 文档是依据 XML 规定的样式语言,实际上是 XML 文档的一种延伸。XSL 能够向输出文件里添加新的元素,或者移动元素。XSL 也能够重新排列或者查找数据,它可以检测并决定哪些

元素被显示，显示多少等。

CSS 同样可以格式化 XML 文档，有了 CSS 为什么还需要 XSL 呢？因为 CSS 虽然能够很好地控制输出的样式，如色彩、字体和大小等，但是它也有严重的局限性：

(1)CSS 不能重新排列文档中的元素。

(2)CSS 不能判断和控制哪个元素被显示，哪个不被显示。

(3)CSS 不能统计计算元素中的数据。

CSS 只适合用于输出比较固定的最终文档。CSS 的优点是简洁，消耗系统资源少。XSL 的另一个特点就是支持中文，而 CSS 不支持中文。

1. XSL 的组成

XSL 主要由两部分组成，第一部分是 XSLT(XSL transformation)，可以把 XML 文档从一种格式转换为另一种格式。它使用 XPath 匹配结点，把一个 XML 文档转换为另一个不同的文档，得到的文档可以是 XML、HTML、无格式文本或任何其他基于文本的文档。XSL 的第二部分是 XSL 格式化对象(formatting object)。格式化对象提供了 CSS 的另一种方式来格式化 XML 文档，以及把样式应用到 XML 文档上。因此，XSL 在转换 XML 文档时分为明显的两个过程，首先转换文档结构，其次将文档格式化输出。这两步可以分离开来并单独处理，XSL 在发展过程中由此也逐渐分裂为 XSLT(结构转换)和 XSL-FO(格式化输出)两种分支语言，其中 XSL-FO 的作用就类似 CSS 在 HTML 中的作用。而下面重点介绍的是第一步的转换过程，也就是 XSLT。

2. XSLT 简介

XSLT 提供一套规则，用于将一组元素描述的 XML 数据转换为另一组元素描述的文档，或者是将该数据转换为一种自定义的文本格式。XSLT 把 XML 文档转换为完全不同的输出，这样可以把数据内容存储在 XML 文档中，然后通过各种媒体将其输出到各种介质中，如无线电、打印、语音等介质。当数据发生变化时，只需要修改 XML 源文档，不需要在多处重复相同的修改工作。通常情况下，XSLT 用于将 XML 文档转换为 HTML，目的是可以在浏览器中进行显示。

最简单的 XSLT 应用情况首先涉及两个文档：包含原始数据的 XML 文档和用来转换该文档的 XSLT 转换文档。将 XML 原始文档输入，处理器将 XSLT 文档作为模板进行转换，最终输出需要的文档。处理过程如图 7-1 所示。

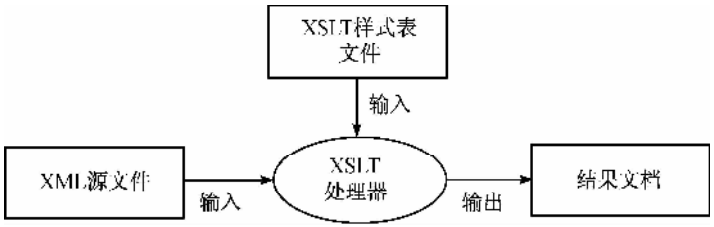

图 7-1　XSLT 处理器工作示意图

XSLT 的一些应用包括：

(1)以查看为目的添加元素。如向 XML 格式的订单中添加公司标志或发送者的地址。

(2)向输出文件中增加新的元素或删除一些元素。如创建图书目录。

(3)对 XML 文档进行分类，测试并确定显示哪些元素。

(4)在不同的 XML 数据字典或模型之间进行转换。如将公司专用的文档转换为符合业界标准的文档。

(5)将 XML 文档转换为 HTML 文档,用以实现与现有浏览器之间的兼容。

3. XPath 简介

XPath 是 XSL 的重要组成部分,它是一种专门用来在 XML 文档中查找信息的语言。可以这样来解释:如果将 XML 文档看做一个数据库,XPath 就是 SQL;如果将 XML 文档看成 DOS 目录结构,XPath 就是 cd、dir、tree 等目录操作命令的集合。通常会将 XSLT 语法和 XPath 语法混在一起阐述。

在 XSLT 中,XPath 表达式返回 4 种类型值:结点集合(node-set)、布尔值、数字和字符串。XSLT 元素通常把 XPath 表达式作为属性值,采用计算表达式的结果。

基本上 XSLT 的最常规用法是返回结点集合或者字符串,具体取决于有关的元素定义。例如,<xsl:template match="chapter"> 定义了当前结点上下文环境内针对<chapter>结点的模板。在这种情况之下,XPath 表达式 chapter 即可返回结点集合作为以后 XSL 函数可用的新的上下文。而在模板内的<xsl:value-of select="title"/>代码中,XPath 表达式把当前上下文中任何<title>结点的原始内容以字符串的形式返回。

表 7-1 列举出了 XPath 访问结点数据的特殊字符用法。

表 7-1 XPath 访问结点数据的特殊字符

特殊符号	含 义	范 例	说 明
/	子结点操作符	item/itemName	匹配<item>结点下的<itemName>子结点
//	与任何一类结点匹配	//itemName	匹配所有的<itemName>子结点
*	选择任何元素的通配符	user/*	<user>结点下的所有子结点
@	属性前缀	user/@name	<user>结点的 name 属性
@*	属性通配符(选择所有的属性)	user/@*	<user>结点下的所有属性
[ ]	可以在其内指定元素或属性,也可加上额外的测试条件	user[@name]	<user>结点下有属性 name 的子结点
|	多个结点匹配	user|server	与<user>或<server>匹配
!	应用一个信息方法至引用结点	!nodeName()	返回结点的确定名称
.	当前结点	.	取得当前结点
..	父结点	../itemName	父结点下的 itemName 结点

4. XSL-FO 格式化对象

XSL-FO 全称是 XSL 格式化对象(XSL formatting object)。XSLT 只是一种转换机制,需要使用目标语言来描述如何对文档进行格式化。就 XSL-FO 格式本身来说,可以看做是一种排版格式。一个完整的 XSL-FO 文档包含了信息内容及控制信息显示方式的版式,其

对信息描述方法的多样性可完全媲美于目前常用的文档格式，如 PDF、WPS、DOC 等。

XSL-FO 更重要的功能是与 XSLT 共同控制 XML 数据的显示方式。这里，XSLT 通常用于描述怎样转换 XML 元素，而 XSL-FO 通常用来描述怎样表示 XML 文档内容。设计良好的 XSL(这里指 XSLT 和 XSL-FO)可以作为模板，修饰功能相近的多个 XML 文档，这给重复性数据，如数据库等的排版显示带来了操作上的方便。

把 XSL-FO 看做一种排版格式，并不是说要把它与传统印刷业的排版格式相提并论，虽然它也可以表达这些排版格式所要表达的内容。确切地说，它是一种“基于 Web 的排版格式”。XSL-FO 的目的之一就是在网络上进行复杂文档的分页处理、大文档和复杂排版格式的处理，以及网络打印等。

从目前的使用情况来看，XSL-FO 可以应用于下列领域：政府公文传输排版系统、政府公文集中打印系统、保险行销系统、市场调查分析用户报告生成系统、PDF 自动发稿系统、报表管理系统、周刊杂志的页排版、技术手册的制作等。

7.2　XSL 样式表

XSL 样式表可以控制 XML 文件中数据元素的输出，样式表元素就是控制这些输出的“执行官”，这些样式表元素各有不同的功能，在使用的时候用恰当的样式表元素组合可以达到事半功倍的效果。

7.2.1　样式表元素

常用的 XSL 的样式表元素如表 7-2 所示。

表 7-2　常用的 XSL 样式表元素

XSL 元素	含　义
xsl:apply-templates	当前 XML 元素结点匹配模板规则
xsl:value-of	输出被选择结点的值
xsl:for-each	对 select 属性指定的相同结点的子结点执行循环处理
xsl:attribute	在输出中新增一个属性
xsl:comment	在输出中创建一个注释结点
xsl:choose	与 xsl:when 和 xsl:otherwise 元素一起，提供多分支条件判断
xsl:when	在一个 xsl:choose 元素内提供单一的条件判断
xsl:otherwise	为一个 xsl:choose 元素提供默认的处理
xsl:if	简单的条件判断，单分支
xsl:stylesheet	一个多模板样式表的文档元素
xsl:template	XML 元素定义特定的模板
xsl:element	在输出中新增一个元素

其中某些样式表元素用于匹配 XML 文档中的特定结点，按照模板内容的指示进行格式化输出。此类元素主要有 xsl:template、xsl:apply-templates。

还有一些样式表元素用于将数据从 XML 文档中提取出来。这是一种在 XML 中应用广泛且操作简单的获得数据的方法。此类元素主要有 xsl:for-each、xsl:value-of。

另外，用于测试模式的样式表元素应先对选择对象进行测试，然后对符合条件的对象进行规定的处理。此类元素主要有 xsl:if、xsl:choose、xsl:when、xsl:otherwise。

7.2.2 样式表命名空间

XML 样式表命名空间经历了多次改版，所使用的命名空间分别为：

```
<!--第一种命名空间-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<!--第二种命名空间-->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!--第三种命名空间-->
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

第一种命名空间是草案，即非正式版，FireFox 不支持此草案版，第二种和第三种命名空间由 W3C 分别发布于 1999 年和 2007 年，后面两种是正式版本。第一种和后两种的语句有部分不同，函数也有较大的区别。浏览器上的区别是，第一种 IE 5.5 以上支持，第二种 IE 6 以上支持。

可不要小看这几种命名空间的不同写法，很多样式表元素在不同的命名空间中所使用的效果是不一样的，通俗地说就是很多样式表元素在第一种命名空间里可以正常使用，到了第二种命名空间里可能就不能使用，或者被其他样式表元素替代。例如，第二种命名空间将 order-by 属性更改为<xsl:sort>元素，并相应地更改 DHTML 脚本代码，以便可以动态地重新对页面排序。对于具体的一些改变，读者可以去 http://www.w3.org/1999/XSL/Transform 详细查看。

<xsl:stylesheet> 根元素中 version="1.0"或 version="2.0"是必须有的。

7.2.3 样式表模块组合

在样式表元素中，常用的模块组合有：

(1)<xsl:template>和<xsl:apply-templates>。

(2)<xsl:choose>和<xsl:when>、<xsl:otherwise>。

(3)<xsl:template>和<xsl:for-each>、<xsl:sort>、<xsl:value-of>。

(4)<xsl:apply-templates>和<xsl:sort>等。

它们的用法将在下面的学习中逐步描述。

7.3　XSL 模板

XSL 样式表文件由一个或多个被称为“模板”的规则集组成，每个模板都包含了与指定结点相匹配的应用规则。模板规则包含了两个部分：模式(pattern)和模板(template)。模式用于在原始文档中匹配结点，模板用于定义结点的处理规则。

7.3.1　XSL 模板的定义

模板是 XSLT 中最重要的概念之一，即使用模板将同样的格式应用于一个 XML 文档的重复元素。在某些元素下，模板就是要应用的规则。可以将模板看做一个模块，不同的模块完成不同的文档格式转换功能。

<xsl:template>元素定义了用于进行转换了的结点内容，其基本语法格式如下：

```
<xsl:template match="标记匹配模式" priority="n" mode="mode" language=" ">
    <!--模板内容-->
</xsl:template>
```

<xsl:template>可以有两个父标记<xsl:stylesheet>和<xsl:apply-templates>，而子标记可以有很多，如<xsl:value-of>等。一个 XML 元素对应一个特定的模板，该 XML 的显示样式由模板内容决定。

match 为必须存在的属性，该属性是一个特殊的字符串，称为模板的标记匹配模式。match 属性的作用是使模板和 XML 元素相匹配，根模板的标记 match 必须是"/"；language 属性确定在此模板中执行什么脚本语言，其取值与 HTML 中的 script 标记的 language 属性的取值相同，缺省值为 JavaScript。

XSL 处理器必须先找到根模板，才能开始 XSL 转换，即 XSL 处理器总是从根模板开始实施 XSL 转换。根模板的内容可以包括其他的模板调用标记，如果 XSL 样式表文件中没有根模板，那么这个样式表文件就没有任何用途。

当 XSL 处理器使用 XSL 样式表文件转换 XML 文档时，处理器将遍历 XML 文档的树状结构，一次浏览一个结点，并将浏览的结点与 XSL 样式表中的每个模板规则的模式进行对比：如果相匹配，处理器就会输出此规则的模板。模板通常包含一些元素指令、新的数据或者从 XML 源文档中复制的数据。

xsl:template 元素有一个 mode 属性，可以根据需要去匹配不同模式的模板。

```
<xsl:template match="/" mode="blue">
    …
<title>学生花名册</title>
<style>.title{font-size:25pt; font-weight:bold; color:blue }
    …
<xsl:template match="/" mode="yellow">
    …
<title>学生花名册</title>
```

<style>.title{font-size:25pt; font-weight:bold; color:yellow }

如果要将 title 输出为蓝色，则用下面语句匹配：

<xsl:apply-templates select="/" mode="blue"/>

如果要将 title 输出为黄色，则写为：

<xsl:apply-templates select="/" mode="yellow"/>

模板总是与结点相对应，一个结点可能对应于不同的模板，XSL 中可以为 xsl:template 设置优先级，其语法格式如下：

<xsl:template match="标记匹配模式" priority="n">模板内容</xsl:template>//n 为优先级

使用 XSL 模板，可以实现模块化的设计。而在模块设计完成以后就需要模板调用元素<xsl:apply-templates>来实现了。

7.3.2 XSL 模板的调用

<xsl:apply-templates>用于告诉 XSL 处理器处理当前结点的所有子结点，基本语法格式如下：

<xsl:apply-templates select="标记匹配模式" />

其中，select 表示选择结点环境，并对其施加相应的<xsl:template>标记所建立的模板。

下面创建一个实例，结合该实例说明模板调用的过程。

【例 7-1】 创建模板和模板调用。

首先创建一个 XML 文件 7-1.xml，代码如下：

```
<?xml version="1.0" encoding="gb2312"?>
<!--file name:7-1.xml-->
<?xml-stylesheet type="text/xsl" href="7-1.xsl"?>
<学生列表>
    <学生 编号=" 201002100001">
        <姓名>张三</姓名>
        <年龄>18</年龄>
        <总分 数值类型="十进制">
            450.00
        </总分>
    </学生>
    <学生 编号="201002100006">
        <姓名>John</姓名>
        <年龄>19</年龄>
        <总分 数值类型="八进制">
            550.00
        </总分>
    </学生>
```

```
        <学生 编号="201003220001">
            <姓名>王五</姓名>
            <年龄>17</年龄>
            <总分 数值类型="十进制">
                435.00
            </总分>
        </学生>
        <学生 编号="201003220004">
            <姓名>李和谐</姓名>
            <年龄>20</年龄>
            <总分 数值类型="十进制">
                465.00
            </总分>
        </学生>
        <学生 编号="201003220007">
            <姓名>赵四熙</姓名>
            <年龄>18</年龄>
            <总分 数值类型="十六进制">
                123.00
            </总分>
        </学生>
    </学生列表>
```

然后创建 XSL 样式表文件 7-1.xsl,代码如下:

```
    <?xml version="1.0" encoding="gb2312"?>
    <!--file name:7-1.xsl-->
    <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
        <xsl:template match="/">
            <xsl:apply-templates/>
        </xsl:template>
        <xsl:template match="学生列表">
            <xsl:apply-templates/>
        </xsl:template>
        <xsl:template match="学生">
            <xsl:apply-templates select="姓名"/>
        </xsl:template>
    </xsl:stylesheet>
```

运行 7-1.xml 文件,结果如图 7-2 所示。

图 7-2 模板调用结果

执行 7-1.xml 文件时，会调用与之关联的 7-1.xsl 样式表文件。XSL 处理器首先读取 XML 文档的根结点，并与第一个模板规则相匹配。

```
<xsl:template match="/">
    <xsl:apply-templates/>
</xsl:template>
```

其中<xsl:apply-templates/>元素表示处理根结点的所有子结点。当 XSL 处理器在读到<学生列表>结点时(7-1.xml 文档的根元素)，与第二个模板规则相匹配。

```
<xsl:template match="学生列表">
    <xsl:apply-templates/>
</xsl:template>
```

同样，在该模板规则中也使用了<xsl:apply-templates/>元素，规定了 XSL 处理器处理<学生列表>结点的所有子结点。当 XSL 处理器读到<学生>结点时，与第三个模板规则相匹配。

```
<xsl:template match="学生">
    <xsl:apply-templates select="姓名"/>
</xsl:template>
```

在该模板规则中使用的<xsl:apply-templates/>元素带有 select 属性，此属性告诉 XSL 处理器只处理<学生>结点下的<姓名>结点。由于在<学生列表>下有 5 个<学生>结点，所以第三个模板规则将匹配 5 次。

7.4 XSL 结点的访问

XSL 样式表文件中必须要有一个根模板，XSL 处理器总是从根模板开始匹配 XML 文档，在处理的过程中，再调用其他模板，这些模板是用来显示指定标记的。如何确定某个模板匹配哪个标记，是哪个标记的显示样式，这就涉及到了“标记匹配模式”。

“标记匹配模式”就是描述该模板适用于哪个(或哪些)标记。“标记匹配模式”就是满足一定条件的一组 XML 标记。实际上，“标记匹配模式”就是告诉模板如何在 XML 文档中选择结点的。

7.4.1　使用元素名访问结点

XSL 中使用元素名访问结点需要使用<xsl:template>的 match 属性，XSL 处理器会根据 match 提供的元素名称去 XML 文档中提取该结点的相关数据。

下面创建一个实例，来演示如何利用元素名称来访问结点。在这里不再创建新的 XML 文档，继续使用 7-1.xml 文件。现在创建 XSL 样式表文件，代码见【例 7-2】。

【例 7-2】　使用元素名访问结点。

```
<?xml version="1.0" encoding="gb2312"?>
<!--file name:7-2.xsl-->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
            <head>
                <title>使用元素名访问结点</title>
            </head>
            <xsl:apply-templates/>
        </html>
    </xsl:template>
    <xsl:template match="学生列表">
        <body>
            <table border="1" bordercolor="teal" width="80%" align="center" bgcolor="orange">
                <th>姓名</th>
                <th>年龄</th>
                <th>总分</th>
                <xsl:apply-templates/>
            </table>
        </body>
    </xsl:template>
    <xsl:template match="学生">
        <tr>
            <td align="center"><xsl:value-of select="姓名"/></td>
            <td align="center"><xsl:value-of select="年龄"/></td>
            <td align="center"><xsl:value-of select="总分"/></td>
        </tr>
    </xsl:template>
</xsl:stylesheet>
```

将上述代码保存成名为 7-2.xsl 的文件，打开 7-1.xml 文件，修改该文件关联的 XSL 文件为：

```
<?xml-stylesheet href="7-2.xsl" type="text/xsl"?>
```

运行 7-1.xml,结果如图 7-3 所示。

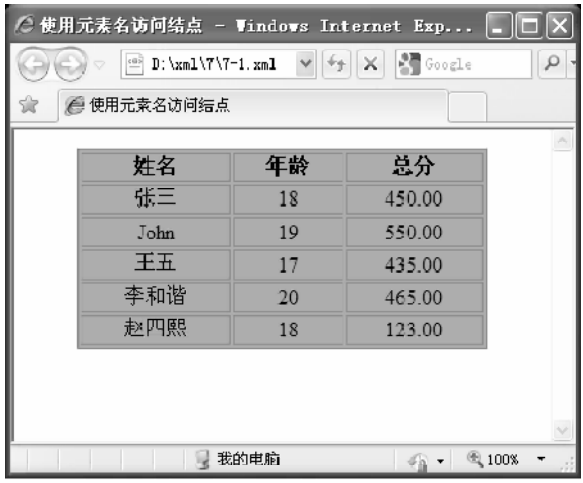

图 7-3 使用元素名访问结点

上述模板在匹配结点时是按元素名称进行匹配的,如<学生列表>和<学生>结点。这里最主要的是要注意"<xsl:template match="学生">"这个模板,需要重复执行 5 次(因为有 5 个<学生>结点)。

另外,还可使用元素属性来访问结点,其基本语法格式如下:

标记[@属性] 或 标记[@属性="属性值"]

它可以建立匹配具有指定属性的标记的模板,起到筛选的作用。如果只需要为带有某个属性的标记创建模板,就可以使用上面的形式指定。

【例 7-3】 使用元素属性访问结点。

```
<?xml version="1.0" encoding="GB2312"?>
<!--file name:7-3.xsl-->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
            <head>
                <title>使用元素属性访问结点</title>
            </head>
            <xsl:apply-templates/>
        </html>
    </xsl:template>
    <xsl:template match="学生列表">
        <body>
            <xsl:apply-templates/>
        </body>
    </xsl:template>
    <xsl:template match="学生">
```

```
            <p>
                <xsl:apply-templates select="姓名"/>
                <xsl:apply-templates select="总分"/>
            </p>
        </xsl:template>
            <xsl:template match="学生列表/学生/姓名">
                <font color="teal" size="6"><xsl:value-of select="."/>
</font>
        </xsl:template>
            <xsl:template match="总分[@数值类型='八进制']">
                <font color="red" size="7"><xsl:value-of select="."/>
                                                                    //(八进制)
                </font>
        </xsl:template>
    </xsl:stylesheet>
```

将上述代码保存为名为 7-3.xsl 的文件，打开 7-1.xml 文件，修改该文件关联的 XSL 文件为：

```
<?xml-stylesheet href="7-3.xsl" type="text/xsl"?>
```

运行 7-1.xml，结果如图 7-4 所示。

图 7-4　使用元素属性访问结点

在上述代码中，“<xsl:value-of select="."/>”表示获取当前结点值；“<xsl:template match="总分[@数值类型='八进制']">”表示该模板只匹配<总分>的“数值类型”属性是八进制的<总分>元素结点，而<总分>的<数值类型>是十进制或十六进制的<总分>元素结点将被排除在外，只能以普通方式显示。

7.4.2　使用匹配符访问结点

match 的属性值其实是一个路径信息，通过使用匹配符“*”可以建立匹配任何标记的

模板，无论该标记是根标记的哪级子标记。例如，<xsl:template match="*">表示匹配任何元素，<xsl:template match="学生列表/*/*">表示匹配<学生列表>根元素结点的任意子结点的子结点。

下面创建一个实例来演示如何使用匹配符访问结点。在这里不再创建新的 XML 文档，继续使用 7-1.xml 文件。现在创建 XSL 样式表文件，代码见【例 7-4】。

【例 7-4】 使用匹配符访问结点。

```
<?xml version="1.0" encoding="GB2312"?>
<!--file name:7-4.xsl-->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
            <head>
                <title>使用匹配符访问结点</title>
            </head>
        <xsl:apply-templates/>
        </html>
    </xsl:template>
    <xsl:template match="学生列表">
        <body>
            <xsl:apply-templates/>
        </body>
    </xsl:template>
    <xsl:template match="学生">
        <p>
            <xsl:apply-templates/>
        </p>
    </xsl:template>
        <xsl:template match="学生列表/*/姓名">
            <font color="blue" size="4"><xsl:value-of select="."/></font>

    </xsl:template>
        <xsl:template match="学生列表/*/总分">
            <font color="red" size="6"><xsl:value-of select="."/></font>
        </xsl:template>
</xsl:stylesheet>
```

将上述代码保存成名为 7-4.xsl 的文件，打开 7-1.xml 文件，修改该文件关联的 XSL 文件为：

```
<?xml-stylesheet href="7-4.xsl" type="text/xsl"?>
```

运行 7-1.xml，结果如图 7-5 所示。

图 7-5　使用匹配符访问结点

7.4.3　使用路径访问结点

模板中的“标记匹配模式(XML 标记的路径信息)”由根标记名称、子标记名称、斜杠或双斜杠共同组成。

斜杠可以用来匹配子结点。一般来说,XPath 中路径开始于当前文中,但在文件系统中,路径可以采用绝对定位而非相对定位的方式。XPath 中也可以有这种方式,使用开始斜杠指向文档的根而不是文档的第一个结点,所以＜xsl:template match＝"/book/title"＞能匹配元素＜book＞结点内的＜title＞结点,而＜xsl:template match＝"学生列表/学生"＞表示匹配＜学生列表＞元素结点下的任意＜学生＞子结点。

双斜杠也可以用来匹配子结点。有时希望对某一类元素结点进行处理,而不管该类结点在整个 XML 文档中所处的位置,这就可以使用双斜杠来匹配结点。例如,＜xsl:template match＝"//姓名"＞会匹配文档内各个位置的＜姓名＞结点而不论其是位于“/学生/姓名”还是位于“/学生/团小组/姓名”。双斜杠还可以位于路径的中间,例如,＜xsl:template match＝"/学生//姓名"＞表示匹配＜学生＞元素结点下所有的＜姓名＞元素结点,＜xsl:template match＝"//＊"＞表示匹配文档中的所有结点。

下面创建一个实例来演示如何使用路径访问结点。这里继续使用 7-1.xml 文件。现在创建 XSL 样式表文件,代码见【例 7-5】。

【例 7-5】　使用路径访问结点。

```
＜?xml version＝"1.0" encoding＝"GB2312"?＞
＜!--file name:7-5.xsl--＞
＜xsl:stylesheet version＝"1.0" xmlns:xsl＝"http://www.w3.org/1999/XSL/Transform"＞
    ＜xsl:template match＝"/"＞
        ＜html＞
            ＜head＞
```

```
            <title>使用路径访问结点</title>
        </head>
        <xsl:apply-templates/>
        </html>
    </xsl:template>
    <xsl:template match="学生列表">
      <body>
        <xsl:apply-templates/>
      </body>
    </xsl:template>
    <xsl:template match="学生">
      <p>
        <xsl:apply-templates/>
      </p>
    </xsl:template>
      <xsl:template match="学生列表/学生/姓名">
          <font color="teal" size="6"><xsl:value-of select="."/></font>
      </xsl:template>
      <xsl:template match="//年龄">
          <font color="red" size="5"><xsl:value-of select="."/></font>
      </xsl:template>
</xsl:stylesheet>
```

将上述代码保存成名为 7-5.xsl 的文件，打开 7-1.xml 文件，修改该文件关联的 XSL 文件为：

```
<?xml-stylesheet href="7-5.xsl" type="text/xsl"?>
```

运行 7-1.xml，结果如图 7-6 所示。

图 7-6 使用路径访问结点

7.4.4　使用条件访问结点

使用方括号“[]”可以提供一个标记需要满足的条件，即可以更精确地匹配某一个结点。为了只选出第二个<学生>结点，可以用一个形如<xsl:template match="//学生[2]">的 XPath 表达式来实现。

使用“|”可以给出几个可以选择的标记，例如：

<xsl:template match="//book|chapter|title">

表示该模板是标记名称为 book、chapter 和 title 标记的匹配模式，不管这些标记是根标记的哪一级子标记。

下面创建一个实例来演示如何使用条件访问结点。这里继续使用 7-1.xml 文件。现在创建 XSL 样式表文件，代码见【例 7-6】。

【例 7-6】　使用条件访问结点。

```
<?xml version="1.0" encoding="GB2312"?>
<!--file name:7-6.xsl-->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
            <head>
                <title>使用条件访问结点</title>
            </head>
            <xsl:apply-templates/>
        </html>
    </xsl:template>
    <xsl:template match="学生列表">
        <body>
            <xsl:apply-templates/>
        </body>
    </xsl:template>
    <xsl:template match="学生">
        <p>
            <xsl:apply-templates/>
        </p>
    </xsl:template>
      <xsl:template match="//姓名|年龄">
            <font color="teal" size="6"><xsl:value-of select="."/></font>
      </xsl:template>
      <xsl:template match="学生[4]">
            <font color="red" size="5"><xsl:value-of select="."/></font>
      </xsl:template>
```

```
</xsl:stylesheet>
```

将上述代码保存成名为 7-6.xsl 的文件，打开 7-1.xml 文件，修改该文件关联的 XSL 文件为：

```
<?xml-stylesheet href="7-6.xsl" type="text/xsl"?>
```

运行 7-1.xml，结果如图 7-7 所示。

图 7-7 使用条件访问结点

在上面的代码中，“<xsl:template match="//姓名|年龄">”表示模板匹配任一路径下的<姓名>和<年龄>结点，而“<xsl:template match="学生[4]">”表示匹配第四个<学生>的所有子元素结点，此模板和“<xsl:template match="//姓名|年龄">”有部分重复，由于它位置靠后，最后的呈现形式以此模板的样式输出；集合中的第一个结点的编号是 1。

7.5 XSL 控制指令

XSL 文件实质上是 XML 文档，由 XSL 标记和 HTML 标记组成，这些标记能被 XSL 处理器识别。XSL 标记可以对输出的数据进行筛选和判断，从而达到过滤数据的目的。

7.5.1 判断指令

<xsl:if>标记主要用来设定结点满足某个条件时才被模板处理，可实现单分支。其基本语法格式如下：

```
<xsl:if test="条件" script=" " language=" ">标记内容</xsl:if>
```

test 用来设置标记过滤的条件，script 表示是否使用脚本程序，language 表示脚本程序使用的语言的种类。只有当 test 设置的条件为 true 时，XSL 处理器才会执行<xsl:if>标记下面的指令，否则不执行下面的指令。

1. test 条件

如果一个 XSL 标记有标记匹配模式，就可以将＜xsl:if＞标记作为子标记来使用。条件表达式的第一项必须是标记匹配模式匹配的 XML 标记。如果标记匹配模式匹配的 XML 标记不是根标记，条件表达式的第一项必须使用“.”来表示标记匹配模式匹配的 XML 标记，这时可以使用下列条件。

1)属性条件

如果想判断和“.”匹配的 XML 标记是否具有某个属性，就可以使用＜xsl:if＞标记的下列语法格式：

```
＜xsl:if test=".[@属性名称]"＞
＜!--标记内容--＞
＜/xsl:if＞
```

如果和“.”匹配的标记是＜学生＞，并有属性“职务”，那么＜学生＞标记就满足下列＜xsl:if＞标记中 test 所要求的条件。

```
＜xsl:if test=".[@职务]"＞
＜!--标记内容--＞
＜/xsl:if＞
```

2)属性值条件

如果想判断和“.”匹配的 XML 标记是否具有某个属性，并判断该属性值和某个特定属性值进行关系比较后的结果是否为 true，就可以使用＜xsl:if＞标记的下列两种语法格式。

第一种语法格式如下：

```
＜xsl:if test=".[@属性名称 关系操作符 '特定属性值']"＞
＜!--标记内容--＞
＜/xsl:if＞
```

对于这种格式，关系元素将按字母顺序比较大小。如果和“.”匹配的标记是＜学生＞，并有属性职务，而且该属性的值是 banzhang，那么＜学生＞标记就满足下列＜xsl:if＞标记中 test 所要求的条件。

```
＜xsl:if test=".[@职务 $gt$ 'banzhang']"＞
＜!--标记内容--＞
＜/xsl:if＞
```

第二种语法格式如下：

```
＜xsl:if test=".[@属性名称 关系操作符 特定属性值]"＞
＜!--标记内容--＞
＜/xsl:if＞
```

对于这种格式，要求属性值以及与其比较的特定值的内容都是数字字符，关系运算符将按数字大小比较。如果和“.”匹配的标记是＜学生＞，并有属性“体重”，而且该属性的值是 106，那么＜学生＞标记就满足下列＜xsl:if＞标记中 test 所要求的条件。

```
＜xsl:if test=".[@体重 $ge$ 106]"＞
＜!--标记内容--＞
＜/xsl:if＞
```

由于 XML 中“＜”和“＞”都有特殊的意义，所以 XSL 中的关系运算符必须使用其他的

符号来代替，如表 7-3 所示。

表 7-3 XSL 的关系运算符

比较运算符	替代符号	功能说明
$ eq $	=	等于
$ ne $	!=	不等于
$ lt $	<	小于
$ le $	<=	小于等于
$ gt $	>	大于
$ ge $	>=	大于等于

3)子标记条件

如果需要判断和“.”匹配的 XML 标记是否有某个子标记，则可以使用<xsl:if>标记的下列语法格式：

```
<xsl:if test="./子标记名称">
<!--标记内容-->
</xsl:if>
```

如果和“.”匹配的标记是<学生>，该标记有子标记<姓名>，那么<学生>标记就满足下列<xsl:if>标记中 test 所要求的条件。

```
<xsl:if test="./姓名">
<!--标记内容-->
</xsl:if>
```

4)子标记及其属性条件

如果需要判断和“.”匹配的 XML 标记是否有特定属性的子标记，则可以使用<xsl:if>标记的下列语法格式：

```
<xsl:if test="./子标记名称[@属性名称]">
<!--标记内容-->
</xsl:if>
```

如果和“.”匹配的标记是<学生>，该标记有子标记<姓名>，而且<姓名>有属性“姓”，那么<学生>标记就满足下列<xsl:if>标记中 test 所要求的条件。

```
<xsl:if test="./姓名[@姓]">
<!--标记内容-->
</xsl:if>
```

5)子标记及其属性、属性值条件

如果需要判断和“.”匹配的 XML 标记是否有特定属性的子标记，并且需要判断子标记的属性值和某个特定属性值进行关系比较的结果是否为 true，就可以使用<xsl:if>标记的下列两种语法格式。

第一种语法格式如下：

```
<xsl:if test="./子标记名称[@属性名称 关系操作符 '特定属性值']">
<!--标记内容-->
</xsl:if>
```

对于这种格式，关系元素将按字母顺序比较大小。

第二种语法格式如下：

```
<xsl:if test="./子标记名称[@属性名称 关系操作符 特定属性值]">
<!--标记内容-->
</xsl:if>
```

对于这种格式，要求属性值以及与其比较的特定值的内容都是数字字符，关系运算符将按数字大小进行比较。

上述 test 条件设置，都必须声明命名空间为：

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

2. xsl:if　实例

下面创建一个实例来演示＜xsl:if＞标记的使用。首先创建 XML 文件，代码见【例 7-7】。

【例 7-7】　＜xsl:if＞实例。

```
<?xml version="1.0" encoding="gb2312" ?>
<!--file name:7-7.xml-->
<?xml-stylesheet type="text/xsl" href="7-7.xsl"?>
<学生信息>
    <学生>
        <姓名>李三封</姓名>
        <总分>580</总分>
        <排名 升="1">第 10 名</排名>
    </学生>
    <学生>
        <姓名>章四喜</姓名>
        <总分>543</总分>
        <排名 降="5">第 13 名</排名>
    </学生>
    <学生>
        <姓名>汪韩浩</姓名>
        <总分>520</总分>
        <排名 升="3">第 18 名</排名>
    </学生>
    <学生>
        <姓名>平河西</姓名>
        <总分>589</总分>
        <排名 降="4">第 7 名</排名>
    </学生>
```

```
    <学生>
        <姓名>黎米</姓名>
        <总分>598</总分>
        <排名 升="2">第 5 名</排名>
    </学生>
</学生信息>
```

然后创建 XSL 样式表文件 7-7.xsl,代码如下:

```
<?xml version="1.0" encoding="GB2312"?>
<!--file name:7-7.xsl-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
      <html>
        <head>
            <title>xsl:if 实例</title>
        </head>
        <body>
            <xsl:apply-templates select="学生信息/学生/*"/>
        </body>
      </html>
  </xsl:template>
      <xsl:template match="学生信息/学生/姓名">
            <h2><font color="red">
                <xsl:value-of select="."/>
            </font></h2>
      </xsl:template>
      <xsl:template match="学生信息/学生/总分">
            总分:
          <font size="4">
                <xsl:value-of select="."/>
          </font>
      </xsl:template>
      <xsl:template match="学生信息/学生/排名">
          <xsl:if test=".[@升]">
            排名:
            <font color="blue">
              <xsl:value-of select="."/>
          </font><sup>升<xsl:value-of select="./@升"/>名</sup>
          <xsl:if test=".[@升 $ge$ 3]">
          <font color="teal" size="4">排名上升至少 3 位,请继续努力!
          </font>
```

```
            </xsl:if>
            </xsl:if>
            <xsl:if test=".[@降]">
              排名:
              <font color="green">
              <xsl:value-of select="."/>
            </font><sub>降<xsl:value-of select="./@降"/>名</sub>
            </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

将上述代码保存为 7-7.xsl,运行 7-7.xml,结果如图 7-8 所示。

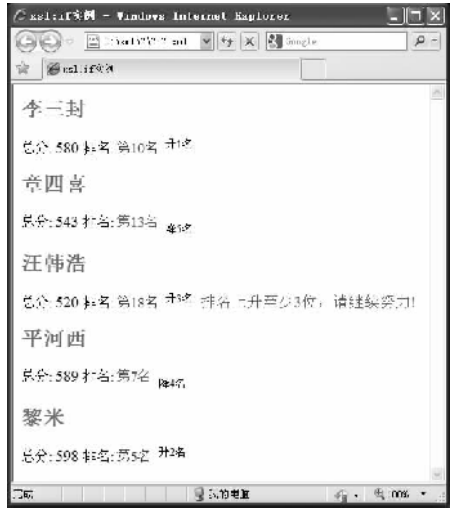

图 7-8　【例 7-7】最终显示结果

7.5.2　多条件判断指令

<xsl:choose>可以对数据同时测试多个条件,根据不同条件输出不同的结果,该元素没有属性设置,表示一个多选测试的开始。<xsl:choose>包含了一组<xsl:when>元素,在 test 属性中可以设置多种条件。测试时将从上至下一次匹配直到找到满足的条件。如果所有的<xsl:when>元素都不满足条件,则应用<xsl:otherwise>元素,该元素无属性设置。基本语法格式如下:

```
<xsl:choose>
    <xsl:when test="条件 1">
        <!--样式定义-->
    </xsl:when>
        …
    <xsl:when test="条件 n">
        <!--样式定义-->
```

```
    </xsl:when>
    <xsl:otherwise>
        <!--样式定义-->
    </xsl:otherwise>
</xsl:choose>
```

下面创建一个实例来说明<xsl:choose>标记的使用方法。在这里不再创建新的 XML 文档,继续使用 7-7.xml 文件。

【例 7-8】 <xsl:choose>实例。

```
<?xml version="1.0" encoding="GB2312"?>
<!--file name:7-8.xsl-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
    <html>
        <head>
            <title>xsl:choose 实例</title>
        </head>
        <body>
            <ul type="1">
              <xsl:apply-templates select="学生信息/学生/*"/>
            </ul>
        </body>
    </html>
  </xsl:template>
    <xsl:template match="学生/*">
    <xsl:choose>
    <xsl:when test=".[@升 $ge$ 2]">
    <font style="color:red;font-size:30">
        <xsl:value-of select="../姓名"/>
        <xsl:value-of select="../总分"/>
        <xsl:value-of select="../排名"/>
        排名上升了 <xsl:value-of select="@升"/>名
    </font>
    </xsl:when>
    <xsl:when test=".[@降 $gt$ 4]">
    <font style="color:green;font-size:20">
        <xsl:value-of select="../姓名"/>
        <xsl:value-of select="../总分"/>
        <xsl:value-of select="../排名"/>
        排名下降了<xsl:value-of select="@降"/>名
    </font>
```

```
        </xsl:when>
      <xsl:otherwise>
        <p></p>
      </xsl:otherwise>
      </xsl:choose>
    </xsl:template>
</xsl:stylesheet>
```

将上述代码保存成名为 7-8.xsl 的文件，打开 7-7.xml 文件，修改该文件关联的 XSL 文件为：

```
<?xml-stylesheet href="7-8.xsl" type="text/xsl"?>
```

运行 7-7.xml，结果如图 7-9 所示。

上面的代码中要注意<xsl:when>条件满足后输出子标记的语句：

```
<xsl:value-of select="../姓名"/>
<xsl:value-of select="../总分"/>
<xsl:value-of select="../排名"/>
排名上升了 <xsl:value-of select="@升"/>名
```

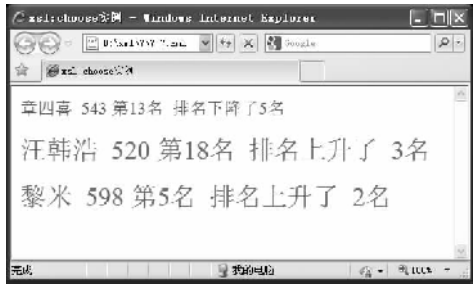

图 7-9　<xsl:choose>实例

其中“..”表示上一级标记，由于<xsl:template match="学生/*">把模板定位到了<学生>的子标记，所以要想输出<姓名><总分><排名>，这样写是必须的。如果用<xsl:value-of select="."/>代替这 3 行，那么只能输出<排名>，因为<xsl:when test=".[@升 ge 2]">和<xsl:when test=".[@降 gt 4]">把匹配模板定位到了标记<排名>，由于升和降是排名的属性，所以<xsl:value-of select="."/>只能输出当前的标记<排名>；而在<xsl:otherwise>中只使用了段落标记<p>，那么不满足条件的标记就会输出一个空段落。如果在此加入<xsl:value-of select="."/>，会出现什么输出结果，读者可以尝试一下，并思考其原因。

7.5.3　循环处理指令

如果需要对 XML 文档中多个相同结点的数据进行同样的处理和输出，就可以使用特定的<xsl:for-each>标记作为循环处理指令，然后通过<xsl:value-of>来具体指定输出的子结点。这种循环处理的指令能够遍历整个 XML 文档结构树，其基本语法格式如下：

```
<xsl:for-each select="标记匹配模式" order-by: "标记名称">
    <xsl:value-of …/>
```

```
        …
    </xsl:for-each>
```

order-by 是一个可选项，用于将显示的元素按一定的顺序排列。以分号分隔其属性值的排列列表，如果属性值前面有“+”，则按升序排列；如果有“-”，则按降序排列。默认情况下按升序排列。例如：

```
<xsl:for-each select="book/author" order-by="+price">
```

下面创建一个实例说明<xsl:for-each>标记的使用方法。在这里继续使用 7-7.xml 文件。现在创建 XSL 样式表文件，代码见【例 7-9】。

【例 7-9】 <xsl:for-each>实例。

```
<?xml version="1.0" encoding="GB2312"?>
<!--file name:7-9.xsl-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
    <html>
        <head>
            <title>xsl:for-each 实例</title>
        </head>
        <body>
            <xsl:apply-templates/>
        </body>
    </html>
    <h3 align="center"> ××班级学生排名变化信息</h3>
    <hr/>
    <xsl:for-each select="学生信息/学生" order-by="-总分">
        <span style="font-style:italic">姓名:</span>
        <xsl:value-of select="姓名"/><br/>
        <span style="font-style:italic">总分:</span>
        <xsl:value-of select="总分"/><br/>
        <span style="font-style:italic">排名:</span>
        <xsl:value-of select="排名"/><br/>
        <span style="font-style:italic">变化:</span>
        <xsl:if test="./排名[@升]">
            <font color="red">上升<xsl:value-of select="*/@升"/>名
            </font>
            <hr/>
        </xsl:if>
            <xsl:if test="./排名[@降]">
                    <font color="teal">下降<xsl:value-of select=
"*/@降"/>名</font>
                <hr/>
```

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

将上述代码保存为 7-9.xsl，打开 7-7.xml 文件，修改该文件关联的 XSL 文件为：

```
<?xml-stylesheet href="7-9.xsl" type="text/xsl"?>
```

运行 7-7.xml，结果如图 7-10 所示。

图 7-10　<xsl:for-each>实例

【例 7-9】的代码按学生总分降序排列，并按不同格式输出排名变化。

<xsl:for-each>排序功能在<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">命名空间内起作用。

7.5.4　输出内容排序

【例 7-9】中给出了 order-by 的简单排序方法，下面创建一个案例来介绍<xsl:sort>的排序方法。在这里不再创建新的 XML 文档，继续使用 7-7.xml 文件。现在创建 XSL 样式表文件，代码见【例 7-10】。

【例 7-10】　<xsl:sort>实例。

```
<?xml version="1.0" encoding="GB2312"?>
<!--file name:7-10.xsl-->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
        <head>
```

```
            <title>xsl:sort 实例</title>
        </head>
        <body>
            <xsl:apply-templates/>
        </body>
    </html>
</xsl:template>
<xsl:template match="学生信息">
    <ul>
        <xsl:apply-templates select="学生">
        <xsl:sort order="descending" select="总分"/>
        <xsl:sort select="姓名"/>
        </xsl:apply-templates>
    </ul>
</xsl:template>
<xsl:template match="学生">
  <xsl:if test="./排名[@升]">
    <font color="red"><dt>↑</dt>
        <xsl:value-of select="*/@升"/>名
    </font>
    <xsl:value-of select="姓名"/>
    <xsl:value-of select="总分"/>
  </xsl:if>
  <xsl:if test="./排名[@降]">
    <font color="teal"><dd>↓</dd>
        <xsl:value-of select="*/@降"/>名
    </font>
    <xsl:value-of select="姓名"/>
    <xsl:value-of select="总分"/>
  </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

将上述代码保存成名为 7-10.xsl 的文件，打开 7-7.xml 文件，修改该文件关联的 XSL 文件为：

```
<?xml-stylesheet href="7-10.xsl" type="text/xsl"?>
```

运行 7-7.xml，效果如图 7-11 所示。

图 7-11　＜xsl:sort＞实例

小说明

＜xsl:sort＞排序功能要在命名空间＜xsl:stylesheet version＝"1.0" xmlns:xsl＝"http://www.w3.org/1999/XSL/Transform"＞中才能起作用。

实　　训

一、实训目的

1. 掌握 XSL 文档的创建和使用方法。
2. 了解 CSS 和 XSL 的区别。
3. 掌握 XSL 模板规则和调用方式。
4. 熟练掌握用不同的方式选择结点的方法。
5. 掌握 XSL 常见的控制指令。
6. 熟练掌握 XSL 与 CSS 的结合使用。

二、实训要求

1. 运用 XSL 结点选择呈现 XML 文档。
2. 运用 XSL 控制指令呈现 XML 文档。
3. 运用 CSS 与 XSL 的结合呈现 XML 文档。

三、实训内容

先创建一个包含图书基本信息的 XML 文档，然后再创建一个 XSL 文档来呈现此 XML 文档。

四、实训设计

下面通过一个 XML 实例，综合运用 XSL 和 CSS 的知识，来达到美观地呈现样式的目的。

首先创建一个 XML 文档 7-8.xml，代码如下：

```
＜?xml version＝"1.0" encoding＝"gb2312"?＞
＜!--file name:7-8.xml--＞
＜?xml-stylesheet type＝"text/xsl" href＝"7-11.xsl"?＞
```

```
<图书信息>
    <图书 编号="0001" 类别="文艺" 数量="150" 库存="80" 打折="8.5">
        <书名>三国演义</书名>
        <作者>罗贯中</作者>
        <出版社>文艺出版社</出版社>
        <书号>0-764-58007-8</书号>
        <定价>80</定价>
    </图书>
    <图书 编号="0002" 类别="文艺" 数量="300" 库存="180" 打折="8.7">
        <书名>红楼梦</书名>
        <作者>曹雪芹</作者>
        <出版社>三秦出版社</出版社>
        <书号>7-80-546839-7</书号>
        <定价>22</定价>
    </图书>
    <图书 编号="0003" 类别="文艺" 数量="200" 库存="175" 打折="8.5">
        <书名>西游记(上下册)</书名>
        <作者>吴承恩</作者>
        <出版社>人民文学出版社</出版社>
        <书号>7-02-000873-9</书号>
        <定价>40.10</定价>
    </图书>
    <图书 编号="0004" 类别="文艺" 数量="300" 库存="192" 打折="8.7">
        <书名>水浒传(上下册)</书名>
        <作者>施耐庵</作者>
        <出版社>时代文艺出版社</出版社>
        <书号>7-53-871401-4</书号>
        <定价>40.00</定价>
    </图书>
    <图书 编号="0181" 类别="计算机" 数量="250" 库存="156" 打折="8.0">
        <书名>XML Web Service 开发</书名>
        <作者>微软公司著</作者>
        <出版社>高等教育出版社</出版社>
        <书号>7-04-015825-6</书号>
        <定价>65.00</定价>
    </图书>
    <图书 编号="0189" 类别="计算机" 数量="230" 库存="160" 打折="8.0">
        <书名>VB.NET 程序设计语言</书名>
        <作者>微软公司著</作者>
        <出版社>高等教育出版社</出版社>
```

```
        <书号>7-04-013188-9</书号>
        <定价>86.00</定价>
    </图书>
    <图书 编号="0028" 类别="计算机" 数量="100" 库存="90" 打折="7.5">
        <书名>JavaScript 速成教程</书名>
        <作者>Michael Moncur 著</作者>
        <出版社>机械工业出版社</出版社>
        <书号>7-111-09070-5</书号>
        <定价>28.00</定价>
    </图书>
    <图书 编号="0074" 类别="计算机" 数量="130" 库存="78" 打折="7.8">
        <书名>软件工程 Java 语言实现</书名>
        <作者>Stephen R. Schach</作者>
        <出版社>机械工业出版社</出版社>
        <书号>7-111-06714-2</书号>
        <定价>51.00</定价>
    </图书>
    <图书 编号="0109" 类别="计算机" 数量="210" 库存="160" 打折="8.0">
        <书名>人工智能</书名>
        <作者>Nils J. Nilsson 著</作者>
        <出版社>机械工业出版社</出版社>
        <书号>7-111-04738-6</书号>
        <定价>45.00</定价>
    </图书>
    <图书 编号="0305" 类别="计算机" 数量="115" 库存="26" 打折="7.5">
        <书名>ASP 开发实例</书名>
        <作者>清汉计算机工作室编著</作者>
        <出版社>机械工业出版社</出版社>
        <书号>7-980039-74-2</书号>
        <定价>53.00</定价>
    </图书>
</图书信息>
```

然后创建 XSL 样式表文件 7-11.xsl，代码如下：

```
<?xml version="1.0" encoding="gb2312" ?>
<!--file name:7-11.xsl-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <xsl:template match="/">
        <xsl:apply-templates/>
    </xsl:template>
    <xsl:template match="图书信息">
```

```
            <html>
                <head>
                    <title>实训 为 XML 文件设计 XSL 文档</title>
                    <style>
                        .style{font-size:36;color:red}
                          # mn{background-color:lightyellow;font-size:16;font-
weight:bold}
                    </style>
                </head>
                <body>
                    <p align="center" class="style">同时使用 XSL 与 CSS 修饰
XML 文档</p>
                     <table border="1" bordercolor="teal" width="95%" id=
"mn" align="center">
                        <th>编号</th><th>书名</th><th>作者</th>
<th>出版社</th><th>书号</th>
                        <th>定价</th><th>类别</th><th>数量</th>
<th>库存</th><th>打折</th>
                        <xsl:apply-templates select="图书"/>
                    </table>
                </body>
            </html>
        </xsl:template>
        <xsl:template match="图书">
            <tr>
                <td><xsl:value-of select="@编号"/></td>
                <td><xsl:value-of select="书名"/></td>
                <td><xsl:value-of select="作者"/></td>
                <td><xsl:value-of select="出版社"/></td>
                <td><xsl:value-of select="书号"/></td>
                <td><xsl:value-of select="定价"/></td>
                <td><xsl:value-of select="@类别"/></td>
                <td><xsl:value-of select="@数量"/></td>
                <td><xsl:value-of select="@库存"/></td>
                <td><xsl:value-of select="@打折"/></td>
            </tr>
        </xsl:template>
    </xsl:stylesheet>
```

运行 7-8.xml,结果如图 7-12 所示。

编号	书名	作者	出版社	书号	定价	类别	数量	库存	打折
0001	三国演义	罗贯中	文艺出版社	0-764-58007-8	80	文艺	150	80	8.5
0002	红楼梦	曹雪芹	三秦出版社	7-80-546839-7	22	文艺	300	180	8.7
0003	西游记(上下册)	吴承恩	人民文学出版社	7-02-000873-9	40.10	文艺	200	175	8.5
0004	水浒传(上下册)	施耐庵	时代文艺出版社	7-53-871401-4	40.00	文艺	300	192	8.7
0181	XML Web Service开发	微软公司著	高等教育出版社	7-04-015825-6	65.00	计算机	250	156	8.0
0189	VB.NET程序设计语言	微软公司著	高等教育出版社	7-04-013188-9	86.00	计算机	230	160	8.0
0028	JavaScript速成教程	Michael Moncur著	机械工业出版社	7-111-09070-5	28.00	计算机	100	90	7.5
0074	软件工程Java语言实现	Stephen R. Schach	机械工业出版社	7 111 06714 2	51.00	计算机	130	78	7.8
0109	人工智能	Nils J. Nilsson著	机械工业出版社	7-111-04738-6	45.00	计算机	210	160	8.0
0305	ASP开发实例	清汉计算机工作室编著	机械工业出版社	7-980039-74-2	53.00	计算机	115	26	7.5

图 7-12　实训最终显示结果

习　题　7

1. XSL 的作用是什么？它主要由哪几部分组成？

2. 请概括出 XSL 与 CSS 的区别。

3. 简述<xsl:template>和<xsl:apply-templates>元素的用途以及二者之间的关系。

4. XPath 的匹配符有哪些？各有什么作用？

5. 如何建立 XML 文档与 XSLT 的关联？

6. 下面是有关学生的 students.xml 文档，请编写出相应的 XSL 文档并将其转换成 HTML 格式，以表格的方式显示数据。

```
<?xml version="1.0" encoding="gb2312"?>
<学生>
    <信息 姓名="张三">
        <学号>A1001</学号>
        <籍贯>河南</籍贯>
        <年级>2</年级>
    </信息>
    <信息 姓名="Xiao Ming">
        <学号>A1002</学号>
        <籍贯>北京</籍贯>
        <年级>2</年级>
    </信息>
    <信息 姓名="ChenKai">
```

```
        <学号>A1003</学号>
        <籍贯>河北</籍贯>
        <年级>2</年级>
    </信息>
</学生>
```

7. 下面是一个使用 XSL 在输出 XML 文档时创建超链接和图片标记的例子，使用了<xsl:element>和<xsl:attribute>标记，题中只给出了 XSL 文件的部分代码，根据执行结果补全该代码(使用<xsl:value-of>从 XML 文档中匹配标记来分别指定<xsl:element>的元素值和<xsl:attribute>的属性值)。

XSL 文档代码如下：

```
<?xml version="1.0" encoding="gb2312"?>
<!--file name:Excise7.xml-->
<?xml-stylesheet type="text/xsl" href="Excise7.xsl"?>
<人物>
    <姓名>孔子</姓名>
    <肖像>kongzi.jpg</肖像>
    <网址>http://baike.baidu.com/view/17990.htm</网址>
</人物>
```

XSL 文档代码如下：

```
<?xml version="1.0" encoding="gb2312"?>
<!--file name:Excise7.xsl-->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
<xsl:template match="/">
    <html><body>
    <title>创建超链接和图片标记</title>
      <xsl:element name="A">
        <xsl:attribute name="href">
          ____________________
        </xsl:attribute>
        <xsl:value-of select="人物/姓名"/>
      </xsl:element>
      <xsl:element name="IMG">
        <xsl:attribute name="src">
        ____________________
      </xsl:attribute>
      </xsl:element>
      </body>
  </html>
</xsl:template>
```

</xsl:stylesheet>

补全代码后，显示结果如图 7-13 所示。

图 7-13　第 7 题最终显示结果