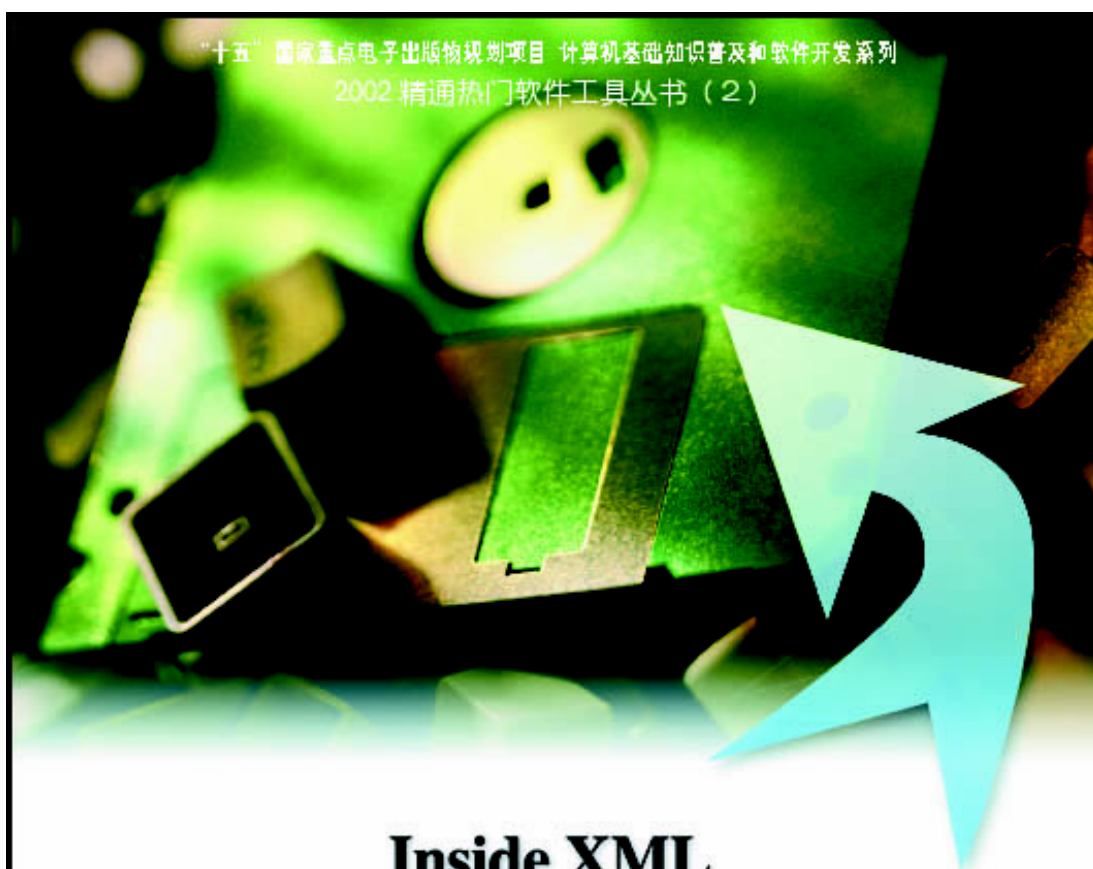




“十五”国家重点电子出版物规划项目 计算机基础知识普及和软件开发系列
2002 精通热门软件工具丛书（2）



Inside XML
XML 编程
从入门到精通

北京希望电子出版社 总策划
曾春平 王超 张鹏 编写

 中国科学技术出版集团
 北京希望电子出版社

“十五”国家重点电子出版物规划项目 计算机知识普及和软件开发系列

2002 精通热门软件工具丛书 (2)

XML 编程从入门到精通

北京希望电子出版社 总策划

曾春平 王超 张鹏 编写

本版特点

(1) 精心设计的 12 个实例详细描述 XML 编程的方法和技巧

读者对象

面向初、中级读者，对高级读者也有重要参考价值



北京希望电子出版社

Beijing Hope Electronic Press

www.bhp.com.cn

2002

内 容 简 介

这是一本通过 12 个典型范例介绍 XML 编程从入门到精通的专著。

全书由 3 篇, 12 章组成, 主要内容包括: 第一篇 XML 提高, 介绍数据建模、DTD 模式、XML 模式和 XML DR、名域机制。本篇附有 3 个实例; 第二篇 XML 与数据, 介绍 DOM 进阶、DOM 应用实例 (包括投票系统、留言本、网址及短消息管理器)、SAX 进阶、XML 与数据。并提供了 7 个实例; 第三篇 XML 工业应用, 主要内容是 WML (无线应用协议)、SMIL (同步多媒体集成语言)、XML 与电子商务、XML 扩展 (共 2 个实例)。

本书的特点是具体应用范例和软件功能相结合, 边讲边练, 由浅入深, 学习轻松, 上手容易。

本书面向初、中级读者, 对高级读者也有重要参考价值。

本版 CD 为配套书

系 列 盘 书 : “十五”国家重点电子出版物规划项目 计算机知识普及和软件开发系列
2002 精通热门软件工具丛书 (2)

盘 书 名 : XML 编程从入门到精通

文 本 著 作 者 : 曾春平 王超 张鹏

C D 制 作 者 : 希望多媒体开发中心

C D 测 试 者 : 希望多媒体测试部

责 任 编 辑 : 周 艳

出版、发行者 : 北京希望电子出版社

地 址 : 北京中关村大街 26 号, 100080

网址: www.bhp.com.cn E-mail: lwm@hope.com.cn

电话: 010-62562329, 62541992, 62637101, 62637102, 62633308, 62633309

(图书发行) 010-62613322-215 (门市) 010-62547735 (编辑部)

经 销 : 各地新华书店、软件连锁店

排 版 : 希望图书输出中心 周宇

C D 生 产 者 : 北京中新联光盘有限责任公司

文 本 印 刷 者 : 北京双青印刷厂

开 本 / 规 格 : 787 毫米×1092 毫米 16 开本 21.375 印张 498 千字

版 次 / 印 次 : 2002 年 2 月第 1 版 2002 年 2 月第 1 次印刷

印 数 : 0001—5000 册

本 版 号 : ISBN 7-900088-49-0

定 价 : 35.00 元 (本版 1CD)

说明: 凡我社产品如有残缺, 可执相关凭证与本社调换。

前 言

从 1998 年 2 月 W3C 正式推出 XML 后，在短短的三年间，XML 以惊人的速度在广大的设计人员中传播开来。随着 Internet 的飞速发展，HTML 开始对更多的网络设计要求显露出疲态。XML 也就是在这种大环境下孕育而生的。

一种技术的产生不可避免的会产生连带的效应，随着 XML 技术的成熟，XML 也越来越被人们重视。我个人也在研究 XML 的内容，尤其是比较深入的，象 XML 的接口、XML 通讯等内容。当我看了许多关于 XML 的书籍之后，我总感觉到好像欠缺了点什么。因为感觉总是每本书的内容差不多，不是从 XML 的语法讲起，就是从 XML 的历史开始，而内容中随处可见的总是一些浅层的内容，对于 XML 的一些内核和接口问题，往往什么都没有说或说的非常简略，这使我很失望。

事实上，我刚开始接触网络是从研究 ASP 和 JAVA 开始的，我想可能很多的程序员和我一样，这些技术在 HTML 网页上有很好的应用，我们总能找到很多可以实现的，技术。但是当我们涉及到 XML 的时候，我们有时候有点为难了，因为在大多数的书籍里，浅显的技术完全不需要我们已经掌握的技术。在这个方面，我们都需要更加完善的内容来介绍一些更深的内容，比如平台的互连，网络之间的通信等来发挥我们应有的能力。

对于绝大多数的 XML 入门者来说，XML 方面的可能只是一知半解，也就是停留在一个平台上，无法再提高了，这主要是国内现有的 XML 方面的书籍涉及到的内容面都比较狭窄，介绍的内容也都是很基本的一些内容，无法再更加深入；虽然有些好的国外书刊，但是往往也因为翻译不好而很难理解。读者可能有这样的感触，当自己花了很多的时间和金钱在书店购买了一大堆的 XML 的书籍，拿回家里，不得不从一本厚厚的书籍中寻觅出仅仅几十页的有用资源。这不能不说是一种浪费！因此，本书从开始就避开了一些浅显易懂的 XML 基础概念和 XML 基础语法。直接从 XML 的高级应用方面对这个很有前途的语言进行介绍。在这些介绍中，我们侧重在 SAX 接口和 DOM 接口的高级使用介绍中。对于 Windows 独行天下的今天，各个软件都充斥的接口、模块、对象等等诸如此类的东西。这种编程的趋势使得我们必须对新的语言的接口问题产生高度的重视。想要使用一个单独的语言完成一项工程是几乎不可能的，或者是非常繁琐的。使用接口，使得各个语言间有交流的通道。XML 在这一点上是成功的。

XML 的应用领域是极其广阔的，在本书的第 4 章之后，我们介绍了关于 XML 的在几个典型领域的应用，例如：电子商务、无线应用协议、多媒体 SMIL 等等。对一个设计者来说，一个语言最终的用处就是在实际的应用上，XML 的实际应用远远超出了本书中所介绍的这几方面。但是，可以通过本书的介绍对 XML 的应用有“质”的理解。对各个领域的应用的介绍是没有止境的，但是只有真正懂得了 XML 的应用的方法，才能在不断推出的 XML 扩展中，第一时间掌握它。

在写这本书的时候，我们假设了读者是有一定的 XML 的编程经验的设计人员，因此，一些 XML 的基础语法和 XML 的一些简单的概念将不会在本书中被解释。如果读者在阅读时感到有理解困难，请自行参照一些关于 XML 的基础读物。

本书着重于将 XML 作为一种开放技术的应用工具来介绍。提供了 XML 模式、数据交换、可视化风格等技术的研究。本书面向有一定基础的 XML 设计人员。其内容大多数涉及

的是一些 XML 应用方面的内容,比较技术性和专业性。本书的重点放在 XML 在 DOM、SAX 以及数据库方面的应用和 XML 的一些实际扩展。尤其是 XML 的扩展,本书介绍了最新的 WML 语言和电子商务包括象 BIZTALK 方面的研究。相信读完本书后读者对于 XML 的各种专业技术会有一个比较全面的了解。

本书由曾春平、王超和张鹏执笔编写。此外,张东、李晓、范智育、王宏生、李光龙、王瑾、吴浩、李炎、刘伟、刘华刚、朱峰、赵晓燕、李晓、马苍、郝春容、韦勇、成美华、萧峰、李菊、张浩然、李欣、张浩、李想、朱大成、周卫、赵中伟、杨竞锐、王贵新、张诚华、朱丽云、程松、李毅、赵郡超、孙名等同志在整理材料方面给予了作者很大的帮助。

由于时间仓促,加之编者的水平有限,缺点和错误在所难免,恳请专家和广大读者不吝赐教,批评指正。

编者

2001.11

声 明

本电子版不包含第二章内容, 请参阅配套图书。

目 录

第一篇 XML 提高

第 1 章 数据建模

第 2 章 DTD 模式

第 3 章 XML 模式和 XML DR

第 4 章 名域机制

第二篇 XML 与数据

第 5 章 DOM 进阶

第 6 章 DOM 应用实例

第 7 章 SAX 进阶

第 8 章 XML 与数据

第三篇 XML 工业应用

第 9 章 WML(无线应用协议)

第 10 章 SMIL(同步多媒体集成语言)

第 11 章 XML 与电子商务

第 12 章 XML 扩展

第一篇 XML 提高

- ☒ 第 1 章 数据建模
- ☒ 第 2 章 DTD 模式
- ☒ 第 3 章 XML 模式和 XML DR
- ☒ 第 4 章 名域机制

作为本书的第一篇，该篇内容是比较简单的。

首先**第 1 章**介绍了一些基本的信息建模原则。在介绍信息建模的时候，对静态模型和动态模型也分别作了必要的介绍。这两种模型都与 XML 的设计有关，而静态永久性数据和暂时性消息这两种类型也是很有意义的。

第 2 章和第 3 章对 XML 的模式做了一个深入的探讨，包括 DTD 和 XML Schema 机制，以及 XML DR 等相关知识。通过该章内容的学习，读者可以选择适合自己的验证方式，也可以对这几种方式进行不同程度的改进，总之力图满足使用的需求。

第 4 章则深入介绍了名域机制。可以毫不夸张的说，正是由于名域机制的引入，XML 才得到了广泛的应用。



第 1 章 数据建模

本章导读

任何文档的设计都需要考虑两点：一是要适合现在的需要；二是要能灵活扩展，适应未来的需要。XML 文档也不例外，因此需要一种标准的方法来设计 XML 文档，这就是本章要讨论的内容。本章将主要介绍 XML 文档设计时需考虑的若干因素，它们包括：

- ☒ 数据建模
- ☒ UML 方法
- ☒ 信息建模
- ☒ 设计 XML 文档

1.1 数据建模

1.1.1 UML 方法

UML (Unified Modeling Language) 是一种现今最流行的建模方法。本章的实例将会依照 UML 方法建立，下面先来介绍 UML 方法。

UML 是一种定义良好、易于表达、功能强大且普遍适用的建模语言。它融入了软件工程领域的新思想、新方法和新技术。它不仅支持面向对象的分析与设计，还支持从需求分析开始的软件开发的全过程。

由于这方面优势，1997 年 11 月 17 日，OMG 采纳了 UML 1.1 作为基于面向对象技术的标准建模语言。UML 代表了面向对象方法的软件开发技术的发展方向，具有巨大的市场前景，也具有重大的经济价值和国防价值。

标准建模语言 UML 的内容

首先，UML 融合了 Booch、OMT 和 OOSE 方法中的基本概念，而且这些基本概念与其他面向对象技术中的基本概念基本一致，所以 UML 必然成为商业界乐于采用的一种简单且一致的建模语言；其次，UML 不仅仅是上述方法的简单汇合，还是在这些方法的基础上广泛征求意见，集众家之长，几经修改而完成的，所以 UML 扩展了现有方法的应用范围；第三，UML 是标准的建模语言，而不是标准的开发过程。尽管 UML 的应用以系统的开发过程为背景，但由于不同的组织和不同的应用领域，所以开发过程也不尽相同。

作为一种建模语言，UML 的定义包括 UML 语义和 UML 表示法两个部分。

(1) UML 语义

描述基于 UML 的精确元模型定义。元模型为 UML 的所有元素在语法和语义上提供了一种简单、一致、通用的定义性说明，使开发者能在语义上取得一致，消除了因人而异的表达方法所造成的影响。此外 UML 还支持对元模型的扩展定义。

(2) UML 表示法

UML 符号表示法,为开发者或开发工具使用图形符号和文本语法来进行系统建模提供了标准。这些图形符号和文字表达的是应用级的模型,在语义上它是 UML 元模型的实例。

标准建模语言 UML 的重要内容可以由下列五类图(共 9 种图形)来定义:

- 第一类是用例图,从用户角度描述系统功能,并指出各功能的操作者。
- 第二类是静态图(Static diagram),包括类图、对象图和包图。其中类图描述系统中类的静态结构。不仅定义系统中的类,表示类之间的联系如关联、依赖、聚合等,还包括类的内部结构(类的属性和操作)。类图描述的是一种静态关系,在系统的整个生命周期都是有效的。对象图是类图的实例,几乎使用与类图完全相同的标识。他们的不同点在于对象图显示类的多个对象实例,而不是实际的类。一个对象图是类图的一个实例。由于对象存在生命周期,因此对象图只能在系统某一时间段存在。包由包或类组成,表示包与包之间的关系。包图用于描述系统的分层结构。

- 第三类是行为图(Behavior diagram),描述系统的动态模型和组成对象间的交互关系。其中状态图描述类的对象所有可能的状态以及事件发生时状态的转移条件。通常,状态图是对类图的补充。在实用上并不需要为所有的类画状态图,仅为那些有多个状态,其行为受外界环境的影响并且发生改变的类画状态图。而活动图描述满足用例要求所要进行的活动以及活动间的约束关系,有利于识别并行活动。

- 第四类是交互图(Interactive diagram),描述对象间的交互关系。其中顺序图显示对象之间的动态合作关系,它强调对象之间消息发送的顺序,同时显示对象之间的交互;合作图描述对象间的协作关系,合作图跟顺序图相似,显示对象间的动态合作关系。除显示信息交换外,合作图还显示对象以及它们之间的关系。如果强调时间和顺序,则使用顺序图;如果强调上下级关系,则选择合作图。这两种图合称为交互图。

- 第五类是实现图(Implementation diagram)。其中构件图描述代码部件的物理结构及各部件之间的依赖关系。一个部件可能是一个资源代码部件、一个二进制部件或一个可执行部件。它包含逻辑类或实现类的有关信息。部件图有助于分析和理解部件之间的相互影响程度。

配置图定义了系统中软硬件的物理体系结构。它可以显示实际的计算机和设备(用节点表示)以及它们之间的连接关系,也可显示连接的类型及部件之间的依赖性。在节点内部,放置可执行部件和对象以显示节点与可执行软件单元的对应关系。

建模语言的应用

从应用的角度看,当采用面向对象技术设计系统时,首先需要描述需求;其次根据需求建立系统的静态模型,以构造系统的结构;第三步是描述系统的行为。其中在第一步与第二步中所建立的模型都是静态的,包括用例图、类图(包含包)、对象图、组件图和配置图等五个图形,是标准建模语言 UML 的静态建模机制。而第三步中所建立的模型是可以执行的,或者表示执行时的时序状态或交互关系。它包括状态图、活动图、顺序图和合作图等四个图形,是标准建模语言 UML 的动态建模机制。因此,标准建模语言 UML 的主要内容也可以归纳为静态建模机制和动态建模机制两大类。

下面总结一下标准建模语言 UML 的主要特点:

☒ UML 统一了 Booch、OMT 和 OOSE 等方法中的基本概念。

- ☒ UML 吸取了面向对象技术领域其他流派的长处,其中也包括非 OO 方法的优点。UML 符号表示考虑了各种方法的图形表示,删除了大量易引起混乱的、多余的和极少使用的符号,也添加了一些新符号。因此,在 UML 中融入了面向对象领域的很多思想。这些思想并不是 UML 的开发者们发明的,而是他们依据最优秀的 OO 方法和丰富的计算机科学实践经验综合提炼而成的。
- ☒ UML 在演变过程中还提出了一些新的概念。在 UML 标准中新添加了模板(Stereotypes), 职责(Responsibilities), 扩展机制(Extensibility mechanisms), 线程(Threads), 过程(Processes), 分布式(Distribution), 并发(Concurrency), 模式(Patterns), 合作(Collaborations), 活动图(Activity diagram)等新概念,并清晰地地区分类型(Type)、类(Class)和实例(Instance), 细化(Refinement), 接口(Interfaces)和组件(Components)等概念。

UML 的目标是以面向对象图的方式来描述任何类型的系统,有广泛的应用领域。其中最常用的是建立软件系统的模型,还可以用于描述非软件领域的系统(如机械系统、企业机构或业务过程),也可以处理包含复杂数据的信息系统、具有实时要求的工业系统或工业过程等。总之,UML 是一种通用的标准建模语言,可以对任何具有静态结构和动态行为的系统进行建模。

此外,UML 适用于系统开发过程中从需求规格描述到系统完成后测试的不同阶段。在需求分析阶段,可以用用例来捕获用户需求。通过用例建模,可以描述对系统感兴趣的外部角色及其对系统(用例)的功能要求。分析阶段主要考虑问题领域中的主要概念(如抽象、类和对象等)和机制,需要识别这些类以及它们相互间的关系,并用 UML 类图来描述。

为实现用例,类之间需要协作,这可以用 UML 动态模型来描述。在分析阶段,只对问题领域的对象(现实世界的概念)建模,而不考虑定义软件系统中技术细节的类(如处理用户接口、数据库、通讯和并行性等问题的类)。这些技术细节将在设计阶段引入,因此设计阶段为构造阶段提供了更详细的规格说明。

编程(构造)是一个独立的阶段,其任务是用面向对象编程语言将设计阶段描述的类转换成实际的代码。需要指出的是,在用 UML 建立分析和设计模型时,应尽量避免考虑把模型转换成某种特定的编程语言。因为在早期,模型仅仅是理解和分析系统结构的工具,过早考虑编码问题十分不利于建立简单正确的模型。

UML 模型还可作为测试阶段的依据。系统通常需要经过单元测试、集成测试、系统测试和验收测试。不同的测试小组使用不同的 UML 图作为测试依据:单元测试使用类图和类规格说明;集成测试使用部件图和合作图;系统测试使用用例图来验证系统的行为;验收测试则由用户进行,以验证系统测试的结果是否满足分析阶段所确定的需求。

总之,标准建模语言 UML 适用于以面向对象技术来描述的任何类型的系统,而且适用于从需求规格描述直至系统完成后的测试和维护整个系统开发的不同阶段。

1.1.2 动态模型和静态模型

静态模型侧重于描述系统的状态。比如下面的语句:“一个客户有一个或多个账号”,“一本书有一个书名”,“每张音乐 CD 都有一个出厂公司”。这些语句主要是描述系统中对

象的类型、特性以及对象之间的关系。静态模型也可以定义词汇表，这些名称往往是比较简洁明了的，能得到大家的公认。

动态模型侧重于描述对信息的处理，例如：处理模型和工作流图表、数据流模型、以及对象生存周期历史。如下面的类型语句，“销售部门把产品报告发送给负责为顾客提供咨询的顾问”，“财务部提取公司某一部门的月盈余总结”。动态模型描述了信息的交换，即出于特定的目的将数据从一个地方发送到另一个地方。

一般的，静态模型与数据库的设计直接相关，信息被长期保存，并用于多种用途；而动态模型直接与信息的设计相关，信息存在的时间很短，而且用途常常随时间转移。但是在实际的设计中，我们并非单纯的把静态模型和动态模型分开来，而是同时考虑这两种模型。

静态信息模型

通常使用用例来建立静态模型。用例模型描述的是外部执行者（Actor）所理解的系统功能。用例模型用于需求分析阶段，它的建立是系统开发者和用户反复讨论的结果，表明了开发者和用户对需求规格达成的共识。首先，它描述了待开发系统的功能需求；其次，它将系统看作黑盒，从外部执行者的角度来理解系统；第三，它驱动了需求分析之后各阶段的开发工作，不仅在开发过程中保证了系统所有功能的实现，而且被用于验证和检测所开发的系统。

几乎在任何情况下都会使用用例来获取需求、规划和控制项目等信息。用例的获取是需求分析阶段的主要任务之一，而且是首先要做的工作。大部分用例将在项目的需求分析阶段产生，并且随着工作的深入会发现更多的用例，这些都应及时增添到已有的用例集中。用例集中的每个用例都是一个潜在的需求。

1. 获取执行者

获取用例首先要找出系统的执行者。可以通过让用户回答一些问题来识别执行者。以下问题可供参考：

- 谁使用系统的主要功能（主要使用者）。
- 谁需要系统支持他们的日常工作。
- 谁来维护、管理，使系统正常工作（辅助使用者）。
- 系统需要操纵哪些硬件。
- 系统需要与其它哪些系统交互，包括其它计算机系统和其它应用程序。
- 对系统产生的结果感兴趣的人或事物。

2. 获取用例

一旦获取的执行者，就可以对每个执行者提出问题以获取用例。以下问题可供参考：

- 执行者要求系统提供哪些功能（执行者需要做什么）。
- 执行者需要读、产生、删除、修改或存储的信息有哪些类型。
- 必须提醒执行者注意的系统事件有哪些，或者执行者必须提醒系统的事件有哪些。

怎样把这些事件表示成用例中的功能。

- 为了完整地描述用例，还需要知道执行者的某些典型功能能否被系统自动实现？

还有一些不针对具体执行者的问题（即针对整个系统的问题）：

- 系统需要何种输入输出? 输入从何处来? 输出到何处?
- 当前运行系统(也许是一些手工操作而不是计算机系统)的主要问题?

需要注意,最后两个问题并不是指没有执行者也可以有用例,只是获取用例时尚不知道执行者是什么。一个用例必须至少与一个执行者关联。还需要注意:不同的设计者对用例的利用程度也不同。例如, Ivar Jacobson 说, 对一个十来人的项目,他需要二十个用例。而在一个相同规模的项目中, Martin Fowler 则用了一百多个用例。我们认为:任何合适的用例都可使用,确定用例的过程是对获取的用例进行提炼和归纳的过程,对一个十来人的项目来说,二十个用例似乎太少,一百多个用例则嫌太多,需要保持二者间的相对均衡。

动态建模机制

动态建模机制主要包括下面几个概念,先介绍一下。

1. 消息

在面向对象技术中,对象间的交互是通过对象间消息的传递来完成的。在 UML 的四个动态模型中均用到消息这个概念。通常,当一个对象在调用另一个对象中时,即完成了一次消息传递。当操作执行后,控制便返回到调用者。对象通过相互间的通信(消息传递)进行合作,并在其生命周期中根据通信的结果不断改变自身的状态。

UML 定义的消息类型有三种:

- ☒ 简单消息(Simple Message) 表示简单的控制流。用于描述控制如何在对象间进行传递,而不考虑通信的细节。
- ☒ 同步消息(Synchronous Message) 表示嵌套的控制流。操作的调用是一种典型的同步消息。调用者发出消息后必须等待消息返回,只有当处理消息的操作执行完毕后,调用者才可继续执行自己的操作。
- ☒ 异步消息(Asynchronous Message) 表示异步控制流。当调用者发出消息后不用等待消息的返回即可继续执行自己的操作。异步消息主要用于描述实时系统中的并发行为。

2. 状态图

状态图(State Diagram)用来描述一个特定对象的所有可能状态及其引起状态转移的事件。大多数面向对象技术都用状态图表示单个对象在其生命周期中的行为。一个状态图包括一系列的状态以及状态之间的转移。

3. 顺序图

顺序图(Sequence Diagram)用来描述对象之间动态的交互关系,着重体现对象间消息传递的时间顺序。顺序图存在两个轴:水平轴表示不同的对象,垂直轴表示时间。顺序图中的对象用一个带有垂直虚线的矩形框表示,并且标有对象的名称和类名。垂直虚线是对象的生命线,用于表示在某段时间内对象是存在的。对象间的通信通过在对象的生命线间画消息来表示。消息的箭头指明消息的类型。

顺序图中的消息可以是信号(Signal)、操作调用或类似于 C++ 中的 RPC(Remote Procedure Calls)和 Java 中的 RMI(Remote Method Invocation)。当收到消息时,接收对象立即开始执行活动,即对象被激活了。通过在对象生命线上显示一个细长矩形框来表示激活。

消息可以用消息名及参数来标识。消息也可带有顺序号,但较少使用。消息还可带有条

件表达式,表示分支或决定是否发送消息。如果用于表示分支,则每个分支是相互排斥的,即在某一时刻仅可发送分支中的一个消息。

4. 合作图

合作图(Collaboration Diagram)用于描述相互合作的对象间的交互关系和链接关系。虽然顺序图和合作图都用来描述对象间的交互关系,但侧重点不一样。顺序图着重体现交互的时间顺序,合作图则着重体现交互对象间的静态链接关系。

合作图中对象的外观与顺序图中的一样。如果一个对象在消息的交互中被创建,则可在对象名称之后标以{new}。类似地,如果一个对象在交互期间被删除,则可在对象名称之后标以{destroy}。对象间的链接关系类似于类图中的联系(但无多重性标志)。通过在对象间的链接上标志带有消息串的消息(简单、异步或同步消息)来表达对象间的消息传递。

5. 活动图(Activity Diagram)

活动图的应用非常广泛,它既可用于描述操作(类的方法)的行为,也可以描述用例和对象内部的工作过程。活动图是由状态图变化而来的,它们各自用于不同的目的。活动图依据对象状态的变化来捕获动作(将要执行的工作或活动)与动作的结果。活动图中一个活动结束后将立即进入下一个活动(在状态图中状态的变迁可能需要事件的触发)。

6. 四种图的运用

上面对 UML 中用于描述系统动态行为的四个图(状态图、顺序图、合作图和活动图)做了简单的介绍。这四个图都可用于系统的动态建模,但它们各自的侧重点有所不同,分别用于不同的目的。下面总结一下如何正确使用这几个图,在实际的建模过程中要根据具体情况灵活运用这些建议。

首先,不要对系统中的每个类都画状态图。尽管这样做很完美,但太浪费精力,而且可能只关心某些类的行为。正确的做法是:为帮助理解类而画它的状态图。状态图描述跨越多个用例的单个对象的行为,而不适合描述多个对象间的行为合作。为此,常将状态图与其它技术(如顺序图、合作图和活动图)组合使用。

顺序图和合作图适合描述单个用例中几个对象的行为。其中顺序图突出对象间交互的顺序,而合作图的布局方法能更清楚地表示出对象之间静态的连接关系。当行为较为简单时,顺序图和合作图是最好的选择。但当行为比较复杂时,这两个图将失去其清晰度。因此,如果想显示跨越多用例或多线程的复杂行为,可考虑使用活动图。另外,顺序图和合作图仅适合描述对象之间的合作关系,而不适合对行为进行精确定义,如果想描述跨越多个用例的单个对象的行为,应当使用状态图。

1.2 信息建模

1.2.1 静态建模

下面开始建立模型了,事实上面对一大堆的数据,也许不知道该从何处下手。这就需要对一些问题进行透彻的了解,这些问题包括:存在哪些信息;他们为什么存在;商业目标是如何实现的;是否有更富创造性和想象力的方法等。这些问题对你的建模是非常有用的。理解了这些问题之后,就可以开始真正的建模了。开始建模的最好方法就是为系统中

的事物设置名称。不必顾虑第一步如何开始，只要从自己所处的位置开始就可以了。

命名事物

事物通常是指那些实体、对象、类或数据元素，我们将这些命名的事物称为对象类型。因此可以把与系统相关的所有事物列出来。下面以一个营业性质的网上音乐书店来说明静态建模和动态建模的建立，这些事物包括客户、预订、仓库、出品公司等。当然这个实例本身是比较简单的，对于比较复杂的信息模型，比如大型商业系统、化学系统等，就需要对系统进行文本描述，然后从中选出所有名词。

得到这些名词后，下一步的工作就是产生对象类型的定义。这可能比较麻烦，通常你经常会为某个名词的确定而思索很长的时间，比如说对于一个网上的预订，每个人对于“预订”的现实意义的理解是不尽相同的。看到预订两字，有人也许认为这是简单的电话预约，还有人认为是因为货还没到而作的预订，这些理解的偏差往往会使人们的交流产生误解。对于单个的网上预订，究竟怎么样才算是一个真正的预订呢？譬如一个人订购一件物品算是预订的话，那么一个人订购两件物品算是一个预订还是两个？同一件商品被预订了多次算是一个预订还是多次预订。预订的客户调整了预订对象仍算是一种预订吗？

事实上这些问题都需要在建立词汇表的时候明确的定义，否则就会引起麻烦，因此就一个网上音乐店来说，一个预订的含义就应该讲明白，即一个客户订购一件音乐商品。

经过以上的对象类型定义，就得到了一长串的对象类型列表，也许可能包含了一些较长的名字，这没关系，我们所需要努力达到的目标是如何对事物进行明确的标记，以此来使业界人士正确的理解和解释。而对于如何组织各种标记包括标记的包含关系，将在下面的几步中讨论。

分类

经过上面的步骤，我们已经列出了系统中所有的对象类型，并且给它们命名了，接下来就要为这些对象类型分类。分类是一个必要的过程，我们必须明确哪些对象应该是超类，它们的子类包含哪些内容，它们又是如何存在的，子类中应该包含哪些信息。比如音乐又可以分为三类，即 CD、MP3 和 TYPE，而从另一个角度来说，音乐也可以按照歌手来分类以方便客户进行查询。对于客户来说，也应该分为个人客户和公司客户。按照刚才所讲的例子，可以把各种分类用下表来表示：

表 1-1 各种分类信息

超类	子类
音乐	CD
	MP3
	TYPE（磁带）
客户	公司客户
	个人客户
歌手	大陆歌手
	港台歌手
	国外歌手

以上列出了本系统中的部分对象的分类，通常我们称上述方法为标志子类，标志子类非常有用，但更重要的事，它有助于理解对象的类型定义。通过子类的包含通常可以很明确的理解超类的真正含义而不致引起错误和混乱。

在 UML 方法中，常用图表来表示子类与超类之间的关系，利用箭头从子类指向超类表示它们之间的关系。

寻找关系

有了分类的对象类型定义，接下来做的就是确定对象类型之间的关系。下面举几个例子来说明这些关系，比如说一个客户可以预订一个或多个音乐 CD，每个预订可以包含一张或多张相同的音乐 CD，每张出品的音乐专辑是对应于一个出品公司的，每张音乐专辑的进货渠道可以有一个或多个等。

这种关系在 UML 中称之为关联，可以用 UML 表示法的示意图来表示如下：

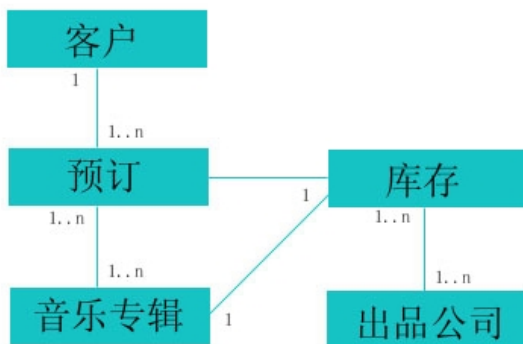


图 1-1 关联关系示意图

通常关系包含好多种，学过数据库或软件工程的读者应该很容易理解。在几类关系中，一对多关系在 XML 中最普遍：一章可以有多个段落，一个人可以预订多个房间，一本书可以有很多种版本等等。上图中我们将一端记为一个对象，另一端记为 1 到 n 个对象（1 意味至少有一个，至多有一个；1..n 表示至少一个，至多为 n 个）。有时候我们也会使用 0 到 n 个来代替 1..n 的关系。

另外还有多对多关系，例如一个读者可以读很多本书，同时一本书可以有很多个读者。在上面的示意图中也有这样的关系：一个预订可以包含很多张专辑，同时一张专辑可以承担好多个预订。对于多对多关系，通常我们要将每一对作为一个对象来命名：我们将有一个预订和一张音乐专辑组成的对称为一个有效的定购。

一对一关系并不常见，它标志一种唯一的关系，比如某个事物对应某个学名。但是在上面的图中也有这样的关系，即音乐专辑和库存的关系。我们之所以把它们的关系定位一对一的关系，是因为我们销售的音乐专辑一定要在库存中找到，同时每张库存中的音乐专辑由一一对应于在网站上公布的音乐专辑。

事实上在 UML 表示信息的过程中，有一种关系是最重要的，前面已经提到过，那就是包含关系，它总是一对多或一对一的。比如客户包含地址、收款单位（人）等。音乐专辑包含价格、种类、编号等等。UML 定义了两种形式的包含：聚合和构成。我们把系统中

将要用到的部分包含关系都用小表表示起来。在明确的书写系统文档，编写 XML 文档的时候需要用到它们之间的关系。如表 1-2 所示。

表 1-2 包含关系表示

根节点	包含
客户	地址
	收款单位（姓名）
	邮编
	客户编号
	类别
音乐专辑	音乐编号
	名称
	价格
	种类
库存	歌手
	编号
	出品公司
	库存量
	销售情况

定义特性

定义特性的主要目的就是为了明确表示静态模型，也就是说静态模型的各种数据类型该如何组织，是否要在固定的范围内取值，它是数字、日期还是文字，单位是什么等。

在 UML 中，可以将对象特性放在描述对象的框中，如图 1-2 所示。

事实上如果把整个系统都用这种方式来表示并且联系起来的话，整个系统就非常得清楚明白，我们也可以很容易完成系统的设计。

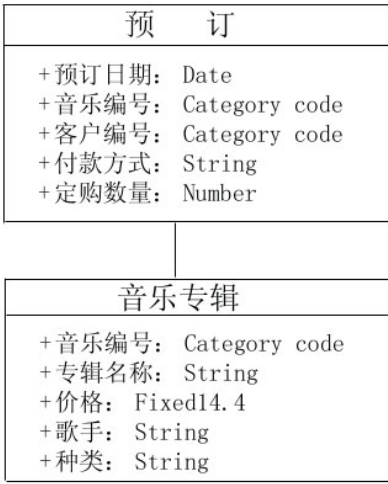


图 1-2 对象特性描述

1.2.2 动态建模

动态建模主要涉及的内容是系统中信息的流动和处理。下面来介绍几种动态建模的方法。

处理模型和工作流模型

处理模型和工作流模型侧重于任何企业在完成任务的过程中所扮演的角色，信息存储和处理阶段是相对次要的。例如，处理模型将描述用户如何来查询每个歌手的情况：它将会定义歌手的类型和分类，以及如何通过查询的方式来获取歌手的信息，客户遇到疑问的时候应该怎么办等。当然在整个过程顺序进行时，它可能还规定用户需要遵守的规则。处理模型通常集中于角色、责任，以及系统中各方参与者的任务，而工作流模型更多的关注于在参与者之间传递的文档。

数据流模型

数据流模型与处理模型非常的相似，但它更多的侧重于信息系统，而不是业务。数据流模型描述了数据存储、处理器和数据流。数据存储决定了将信息永久性的保留在何处（比如计算机数据库，或者仅仅放在档案柜中）；处理器将对数据进行操作；数据流是指将数据从一个处理器或数据存储传送到另一个处理器或数据存储。数据流模型极其依赖于静态信息模型。前面的例子中，静态模型通过客户、音乐专辑、库存等名词概念的定义来描述，但是它没有说明信息存储。相反的，数据流模型将说明有关音乐专辑的信息会一直保留在我们所给出的数据库中，直至数据库更新为止。

对象模型

对象模型包含了动态部分和静态部分。对象定义的动态部分（或称行为部分）侧重于每个对象能够做什么，它通过一组操作或方法定义了对象的行为。

对象模型比较适合作为设计工具。事实上在实际工作中，事件通常是和多个对象都有联系的，到底和那个对象相关联完全取决于设计的要求。

对象生存周期历史

对象生存周期历史（UML 称为对象生命线）也关注单个的对象，但是它是从整体角度考虑的：它描述了每个对象在生存周期内所执行的操作：如何创建对象，在对象的生存周期中能够发生哪些事情，对象如何响应这些事件，什么情况使得它最终被清除等。

对象生存周期历史对于测试模型的完整性很有价值。比如对于一个简单的预订，系统模型可能描述了如何预订相应的音乐专辑，但是忽略了如何取消预订。因此只有定义了每个对象如何进入系统，以及如何从系统中删除和退出以后，才会发现原来的设计中存在的纰漏。

使用案例

使用案例可以分析特定的用户任务是如何被执行的。例如：预订了音乐专辑的人如何取消订单。使用案例可以与处理模型非常相似，但它通常侧重于特定用户的活动。

在将业务模型化和描述 IT 系统的内部行为两个方面，用户案例都非常有用。虽然如何发挥使用案例的作用要根据实际的情况而确定，但是它所见过的最有价值的实例是将使用案例主要应用于用户与系统的交互：描述对话框。

UML 为描述使用案例提供了图形表示法，但是最好的方法是通过交互式原型，人们常常称它为 storyboard。通过使用案例描述的用户界面对话框将集中说明用户和系统之间的交互信息，而不是解释对话框在屏幕上如何显示。

对象交互图

对象交互图从一个比数据模型更加详细的层次来分析对象之间的消息交互。数据流模型仅仅说明在线预订，通过预订并向信用卡发送交易信息，然后收到确认信息作为应答。然而，交互图将详细描述构成这次会话的所有信息。

当需要描述两个独立系统之间的交互时，比如网上系统与一个音乐制作公司之间的交互，对象交互图就可以出色的完成任务。它能帮助将那些信息封装在消息中。由于这些消息通常是 XML 消息，因此对象交互图为设计每个消息的 XML 结构提供了所需的上下文。

下面要选择建立动态模型的方法了，事实上，并不一定要建立上述的模型，上面的方法只是作为有用的工具而已。我们必须综合各种方法的优点，还要和用户多交流，才能更好的定义各种系统之间的消息传送。

1.3 设计 XML 文档

接下来就回到本书的正题，讲述 XML 方面的知识了。事实上本部分的工作就是通过刚才建立的信息模型来建立 XML 文件系统。那么如何设计满足要求的 XML 文档呢？这时候有两类 XML 需要区分：一类是用于保存永久性信息的 XML 文档，这种文档用于数据存储，提供永久性数据，以便系统使用；而另一类则是用于消息的 XML 文档，称之为消息流，它将暂时存在的信息从一个子系统（或者人、或者数据存储）转移到另一个子系统中。下面介绍这两类 XML 文档的设计方法。

1.3.1 XML 文档

永久性 XML 文档的设计

永久性 XML 文档用于存储数据。从实用角度来说，它相当于一个数据库。但在设计时又与数据库不同，设计一个永久性的 XML 需要考虑两点：一是文档的大小，二是文档的类型定义。

1. 文档的大小

先来看文档的大小。一个 XML 文档不能过大也不能过小，对于有些应用系统来说，一个包含几千兆字节的 XML 文档是比较适宜的，而有些应用系统则宁愿使用小文档。由于 XML 不太适合于直接访问，所以目前要访问一个大文档的任何部分就必须解析整个文档，这就要耗费很长的时间。另一方面 XML 文档过小导致文档数量过多，会使系统繁杂而影响效率，而且它不能利用 XML 丰富的结构来表示信息模型中的关系。因此设计时要根据不同的系统设计要求来定。

当一些业务对象非常大而且各部分的结构非常复杂时，自然会将每个对象映射为一个 XML 文档。可以参看一些人才资源系统，它为每个职员记录使用一个 XML 文档，例如在病历系统中，每个病历都对应着一个 XML。再回到我们的例子，有几个方面是作为永久性数据存储的，包括用户、库存、音乐专辑等。当然由于这个系统相对较小，不可能也不必像人才资源系统或病历系统一样为每个客户或每个厂家建立一个 XML 文档，当然是组合成一个比较大的 XML 文档更方便一些。看下面的例子：

程序清单 1.1:

```
<?xml version="1.0" encoding="gb2312"?>
<库存>
  <音乐 种类="CD">
    <编号>CD-1023</编号>
    <名称>盛夏的果实</名称>
    <演唱者>莫文蔚</演唱者>
    <出品公司>上海音像</出品公司>
    <联系人>赵明明</联系人>
  </音乐>
  <音乐 种类="CD">
    <编号>CD-1024</编号>
    <名称>冷酷到底</名称>
    <演唱者>羽泉</演唱者>
    <出品公司>上海音像</出品公司>
    <联系人>赵明明</联系人>
  </音乐>
  <音乐 种类="MP3">
    <编号>MP3-0212</编号>
    <名称>男人哭吧哭吧不是罪</名称>
    <演唱者>刘德华</演唱者>
    <出品公司>河北音像</出品公司>
    <联系人>李香云</联系人>
  </音乐>
  ...
</库存>
```

利用上面的例子记录系统的库存：每张音乐专辑都设置了一个联系人，以便在出现订货或要求得到其他消息的时候，能够随时得到专辑的详细资料。联系人的信息则被写在另外一个 XML 文档中。如下面的例子所示：

程序清单 1.2:

```
<?xml version="1.0" encoding="gb2312"?>
<客户名单>
  <客户 种类="个人">
    <编号>KH-1365</编号>
```

```
<客户地址>DR-1876</客户地址>
<姓名>任建兴</姓名>
<订单>DD-345</订单>
</客户>
<客户 种类="个人">
  <编号>KH-1368</编号>
  <客户地址>DR-1886</客户地址>
  <姓名>马辛</姓名>
  <订单>DD-367</订单>
</客户>
<客户 种类="公司">
  <编号>KH-0023</编号>
  <客户地址>DR-123</客户地址>
  <单位>南北湖旅游局</单位>
  <订单>DD-033</订单>
</客户>
...
</客户名单>
```

上面的例子给出了客户的一些情况，可以看出，在客户中使用属性对它们进行了分类，还把客户的地址放在另一个 XML 文档中。这样做完全是为了设计的需要，不致使系统文件过于复杂化，使得在对文档进行访问时效率得到提高。

在使用 XML 文档保存永久性信息时，还要考虑如何查找系统中的信息。查找信息的过程可以分为两步：首先找到正确的文档，然后找到你感兴趣的内容。正确定位文档通常使用以下几种可选的方法：

- ☒ 利用操作系统文件存储中的目录结构通过名称定位文档。文档的文件名可能与系统中使用的对象标志符相关，人事文件可以用职员的人事编号命名。
- ☒ 利用文档之间的相互索引，这种方法类似于传统的 Web 站点。文档总是通过链接找到的，不过在此通常采用更加结构化的文档组织方式。举例来说，如果有一组足球比赛报道，每个报道都是由一个 XML 文档构成的，可以建立另一个文档作为它们的索引，列出所有比赛一日期、地点和参赛的球队。当然，不必手工维护这个索引，可以将它设置为自动更新：提交新的比赛报道后，系统自动对它进行分析，并将相关信息添加到索引文档中。这可以利用 XSLT 实现。
- ☒ 利用关系数据库索引文档。可以选择将 XML 文档保存为文件形式，并通过数据库引用，或者将 XML 文档也存储在数据库中，越来越多的数据库明确表明支持该功能。如果必要的话，还可以使用二进制大对象（Binary Large Object）字段。如果利用关系数据库索引 XML 文档，就可以通过任意 SQL 语句访问它们。
- ☒ 利用自由原文搜索引擎索引文档。越来越多的搜索引擎提供对 XML 的支持。通过这种方法，可以根据文档中任何位置出现的关键字搜索文档。虽然自由原文搜索通常被定位为支持用户对非结构化的查询，但是由于 XML 文档中的标记使得文档具有更强的结构，因此使得这种查询方法变得非常有效。

当然也可以利用 XML 的 DOM 结构来寻找信息，这通常都要借助于“XML 服务器”。利用 DOM 接口访问数据不必对整个文档进行读取和解析。它使数据访问更加平滑，避免使用一个 API 定位文档，使用另一个 API 在文档中搜索信息。这是因为 DOM 形式的文档存储使文档对象模型中的节点保存为数据库对象的。

从理论上来说，我们希望用户能了解所有的信息，但是实际上一个屏幕的信息是有限的，但是恰恰正是这一个屏幕的信息却是用户青睐的，对于大量的，能充满好几屏的信息数据，用户可能会觉得繁琐了。相反的，如果需要访问几百个 XML 文档才形成一屏信息，这样势必在访问每个页面时都会产生大量的开销，而解决这一问题的方法就是通过 DOM 建立文档的过滤来产生。关于 DOM 内容，将在以后章节详细介绍。

2. 文档的类型定义

再来看文档的类型定义。XML 并没有严格定义文档的类型概念，特别是独立的两个概念，它们并不一定要一一对应。可以用两种方法表示：一种是一个文档对应一个 DTD，如果不同的文档类型中有部分内容是公共的，可以定义 DTD 为能够共享的外部参数实体；另一种方法是所有的文档类型使用相同的 DTD，但是每一部分是用不同的顶级元素标记。DTD 没有规定那个元素应该作为文档元素，因此可以是同一 DTD 定义几个文档类型。通过这种方法，各个文档之间可以方便地进行交互。DTD 本身并不一定是一个孤立的实体，它可以通过外部实体表明其他 DTD 中的定义，或者通过一定条件部分进行参数化，因此它实际上能够变得非常灵活，但是这样一来往往导致 DTD 非常复杂而且又难以理解。

消息 XML 文档

消息可以是企业之间发送的用于表示物品预定交易的 EDI 式消息，也可以是一个子系统向另一个子系统提出传递数据的请求。例如在银行中，想要欺骗监测系统可以要求银行运作系统提供特定账户的交易历史，并接收 XML 消息形式的应答。除了这些，还可以为用户提供现实的信息，例如：当你通过浏览器咨询特定的音乐专辑的详细情况时，应答（response）以 XML 的消息形式从后台数据库系统返回前端，前端系统（或浏览器）利用适当的样式表进行展示，将应答消息变为可视化页面。有些消息可能来自人，例如：销售查询是由 Web 站点上的表单创建的。

以下通用的设计原则可以应用于所有的 XML 消息，无论这些消息具体的用途是什么：

- ☒ 设计应该反映信息内容，而不是体现预期的用途。随着时间的推移，信息的用途可能发生变化，但是信息的内容一般是稳定的。这条原则特别针对表示细节：如果信息是供人使用的，还应包含屏幕布局和字体大小等信息。
- ☒ 设计应该预计到可能的变化。当然，XML 本身的设计在这方面是领先的，它能够避免传统的缺陷，例如：固定长度的字段和固定长度的列顺序。但是，文档设计者在将信息结构化时有责任采用一种能够在一定程度上适应未来变化的方式。
- ☒ 尽量使用标准消息类型，少使用自己的“发明创造”。目前标准化的消息类型越来越多，例如：最初由微软发起的 Biztalk（该部分内容在后面将会详细介绍），以及独立的 OASIS 协会都提供了许多例子。除此之外，一些专业的网站比如 <http://www.ontology.org.com>、<http://www.rosettanet.org> 等，都提供了典型的消息实例。

- ☒ 在性能允许的范围内，数据编码应该与自然编码尽可能接近。避免过于简洁的编码，例如：将 W 作为 Withdraw 的编码，除非这种编码已经被业界广泛接受。使用该行业中已有的标识符和编码（比如信用卡号），尽量不要自己杜撰标识符，避免使用消息和现有数据形成密切关系的标志符。特别强调消息应该是独立的。

有一个设计决策是比较麻烦的，即我们是否要包含一些目前尚不需要的信息，以备将来使用。对于这个问题，没有一致的答案。如果包含对象的所有特征比从中选出接受者感兴趣的特性更加简单，可以将它们全都包含在内，由接受者决定舍弃哪些特性。另一方面，发送大量多余的数据是非常愚蠢的，当然也可以出于安全性或隐密性考虑保留一些特性。

在消息中包含常规信息是非常明智的，即使它重复了任何用于传递数据的消息系统。日期、时间、发送者标识和预期的接受者显然要包含消息；此外，还可以加入序号，它能够检查传输过程中是否有消息的丢失，或者是否有消息被传输了两次。之所以在 XML 消息中包含许多相关信息，是因为这样做能使得接收者更方便地从消息中提取信息，而不必使用单独的 API；更重要的是，如果消息是为了审计等用途而进行的归档，将所有内容放在一起非常便于存储。

下面的例子是一个消息形式的 XML 文档：

程序清单 1.3：

```
<?xml version="1.0" encoding="gb2312"?>
<今日预订 date="2001-09-23">
  <预订>
    <客户编号>KH-0786</客户编号>
    <音乐编号>CD-0987</音乐编号>
    <数量 单位="张">1</数量>
    <付款方式>信用卡</付款方式>
  </预订>
  <预订>
    <客户编号>KH-0568</客户编号>
    <音乐编号>CD-2365</音乐编号>
    <数量 单位="张">50</数量>
    <付款方式>信用卡</付款方式>
  </预订>
  <预订>
    <客户编号>KH-6987</客户编号>
    <音乐编号>TYPE-0256</音乐编号>
    <数量 单位="盘">5</数量>
    <付款方式>现金</付款方式>
  </预订>
  ...
</今日预订>
```

在上面的例子中，浏览器与用户发生信息的交换，然后直接由浏览器生成发送给系统的 XML 消息文档。该消息文档每天产生一个，在一天还没结束的时候，其他的信息保存

在浏览器端，并且不断地加入新的信息。关于文档中的一些细节的问题，下面将会讨论。

1.3.2 信息模型的映射

表示对象关系

表示对象关系与设计一个数据结构非常相似。可以把一个文档类型变成一个标记，通常的做法是选取简单的名词来表示，一方面是为了节省空间，另一方面也是突出内容，以免读者过多的注意标记本身而忽视了内容。当然，标记的选择分特殊和普通情况，譬如说实例中的一个“预订”，可以定义为“预订”，也可以表示为“音乐预订”。一个表示普遍的含义，在一定程度上简化了应用程序的代码的编写；另一个则表示特殊的类型，能更精确的定义与该元素相关的属性和子元素，下面的语句用音乐预订表示了这样的关系：

程序清单 1.4:

```
<?xml version="1.0" encoding="gb2312"?>
<预订>
  <音乐预订>
    <预订号>YD-0987</预订号>
    <客户>杨光</客户>
    <专辑>太委屈</专辑>
    <负责人>欣欣</负责人>
  </音乐预订>
  <音乐预订>
    <预订号>YD-0987</预订号>
    <客户>杨光</客户>
    <专辑>太委屈</专辑>
    <负责人>欣欣</负责人>
  </音乐预订>
  ...
</预订>
```

对于超类和子类的关系，不仅可以通过子元素包含来实现，还可以通过属性来实现，比如：

```
<预订 种类="音乐"/>
```

事实上在该实例中可以认为把种类属性作为可选的属性。

关系的表示

对于包含关系，可以通过子元素的包含来实现，这里不再细说。而对于元素之间的链接又该怎么办呢？通常的方法是利用属性 ID 和 IDREF 来实现。

ID 属性包含的值能够在文档中唯一的标志元素；IDREF 属性的值必须与文档中某个元素的 ID 相同。因此，ID 可以作为关系数据库中的主关键字，IDREF 则作为外部关键字。

另外,不要忘记 IDREFS 的数据类型(因为关系数据库中并没有直接与之对应的结构)。IDREFS 包含一组以空格分隔的 IDREF 值,因此它可以作为一对多关系的“一”端,或者多对多关系的任何一端。

使用 ID/IDREF 的主要优势在于 XML 解析器能够帮助进行文档的有效性检验。比如下面的例子中,对于预定同一张音乐 CD 的客户就用 IDREF 来连接起来,方便统计。

程序清单 1.5:

```
<?xml version="1.0" encoding="gb2312"?>
<客户名单>
  <客户 id="D01">
    <地址>KH-2012 </地址>
    <姓名>李建兴</姓名>
    <订单>DD-1576</订单>
  </客户>
  <客户 id="D02">
    <地址>KH-1023 </地址>
    <姓名>王频</姓名>
    <订单>DD-1577</订单>
  </客户>
  <客户 id="D03">
    <地址>KH-1024 </地址>
    <姓名>刘黎</姓名>
    <订单>DD-1578</订单>
  </客户>
  <客户 id="D04">
    <地址>KH-1026 </地址>
    <姓名>章新</姓名>
    <订单>DD-1579</订单>
  </客户>
  <预订 客户="D01 D02">
    <专辑>太委屈</专辑>
  <预订>
  <预订 客户="D03 D04">
    <专辑>年华</专辑>
  <预订>
</客户名单>
```

表示属性

在 XML 文档中,这样的问题也许会让你为难,就是选择使用 XML 属性来表示它,还是使用嵌套的(子)元素来表示它?先来看下面的两个例子:

程序清单 1.6:

```
<书 作者="路遥">
```



```
书名="平凡的世界"
出版社="北京出版社"
序列号="ISBN-346-10354-5"/>
```

这个例子把书的特性表示成为 XML 的属性。下面的例子则把书的特性表示成子元素的形式。

程序清单 1.7:

```
<书>
  <作者>路遥</作者>
  <书名>平凡的世界</书名>
  <出版社>北京出版社</出版社>
  <序列号>ISBN-346-10354-5</序列号>
</书>
```

早在 SGML 中就已经存在子元素和属性之间的区别了。SGML 最初是作为一种标记语言来发行、提供文本。后来，其他应用程序也将它作为通用的数据交换格式。从标记语言角度来考虑，SGML 中的内容（最终用户将在页面上看到的文本）和元数据（有关内容的信息，它是供执行各种处理操作的软件使用的）有着明显的区别。简单的来说，内容是用元素标记中的文本表示的，元数据是用属性表示的。这种差别一直存在于 HTML 中。

所以，SGML 中元素内容和属性之间的差别从本质上体现了信息在用途上的差别：元素内容是供文档的读者使用的，属性是供印刷商或他们的软件使用的。当然，一旦让 SGML 或 XML 用于软件系统之间的数据交换，这种差别就不存在了。因为在实际的应用中没法很明确的决定每部分的用途。比如说电子表格，决定哪些供软件使用，哪些供用户使用是很困难的。

因此从实用的角度来考虑，建议自由的选择使用元素和属性。下面的表 1-3 分析了使用元素和属性的各个弊端。

表 1-3 元素和属性的比较

	优点	缺点
XML 属性	<ul style="list-style-type: none">• DTD 能够对值进行约束：如果只允许有限的几个值，例如：“yes”和“no”，使用属性非常有效• DTD 能够定义缺省值验证 ID 和 IDREF 的有效性• 占用较少的空间• 对于某些数据类型，例如 NMTOKENS 能够进行空白的规格化，能够减轻应用程序解析的压力。• 便于使用 DOM 和 SAX 接口进行处理• 能够访问为解析的实体，例如二进制数据	<ul style="list-style-type: none">只支持简单的字符串值不支持元数据（即“属性的值”）无序
子元素	<ul style="list-style-type: none">• 支持任意复杂的值和重复的值• 有序• 支持“属性的属性”	<ul style="list-style-type: none">空间占有率略高编程比较复杂

优点

- 当数据模型改变时，子元素可扩展

缺点

如何权衡这些因素，取决于应用程序的实际情况。许多有经验的设计者认为最重要的因素是变化的潜力：随着应用程序的发展，是否能够扩展文档或隔绝消息。这一因素使得子元素优于属性，因为实际应用中最常见的是将一个简单的特性扩展为复杂的结构化特性。

编码

在 XML 文件中，无论是元素还是属性都表示对象的特征，都需要决定如何对他们的值进行编码，而所谓的编码通常涉及到如下的内容：特征的命名、各种单位以及一些需要统一化的格式等。

1. 特征的名称

先来看特征的名称。当使用元素或属性表示信息模型中的特征时，应该使用什么样的名称呢？比如说最简单的一个文档的创建日期，可以直接用“日期”来表示，也可以用“创建日期”来表示。还比如有一个客户的地址，可以用“地址”或“客户地址”来表示。那么究竟用哪个来表示比较好？

当然，在有许多不同的日期和地址存在的时，就只能用很长的名字来区分了，显然这样会比较明确，因为它消除了文档内容的含糊不清。但是，这种形式同时也删除了隐含的数据类型信息。也就是说我们原来使用的“客户地址”和“公司地址”有相同的有效性约束，但是当使用了较长名字后，系统软件又如何来区别呢？这是相当麻烦的事，唯一的办法就是修改 DTD，直接导致 DTD 的复杂化。

假如使用较短的名字，那么有可能使多个对象类型都包含相同的特征名称，可能导致的结果就是名字重载。举个简单的例子，对于某个 XML 文件系统，公司、音乐和客户都有自己的名字，如果都使用“名称”来表示是否合理呢？一般来说，XML 的元素名称都是全局的，而属性的名称则用于表示特定的元素类型，因此通常是局部的或说本地的。那么又如何来选择命名形式呢？一条通用的原则就是无论在哪种情况下，最好不要用相同的名称表示含义不相同的事物。可以考虑的选择有：

(1) 使用系统命名习惯，比如<Ship.Date>和<Receive.Date>，当然也可以用下划线代替。这不仅能够保证含义清晰，而且能够保持简洁性。甚至还可以以这种命名习惯扩展到各种应用软件中，但是这种结构化命名方式不能应用于 XSLT。

(2) 使用能说明含义的名称，数据类型用属性说明。比如对于上面的<Ship.Date>，可以表示为<Ship Date type=“Date”>。这意味着应用程序仍然可以获得数据类型。而对于文件中很多的 ShipDate 元素，我们可以使用 DTD 定义 FIXED 属性值。唯一的缺点就是不能同时说明所有的 type=“Date”的元素都符合相同的形式。

(3) 和上面的方法相反，数据类型作为元素，而属性表示元素扮演的角色。比如上面的发送日期可以表示为<Date type=“Ship”>。从编写 DTD 的角度来看，这种方法显然比上一种方法要好。

(4) 使用嵌套的元素层。这种方法不使用属性，而是用多个元素层次，如：

```

<Merchandise>
  <Ship>
    <Date>
    </Date>
  </Ship>
</Merchandise>

```

从理论上讲，这种方法是比较好的。但是产生了大量的标记，因此往往使文档比较复杂。

2. 二进制数据

在文档中可能用到的对象的所有属性并不是都可以用字符串来表示的，比如说对于一个图像文件“show.jpg”，或者说对于其他的多媒体数据。因此在 XML 中使用它们就需要用到其他的方法。其中一种方法就是使用 CDATA 来包含二进制数据，但是这样的包含方式往往使得数据没法解析，而一定要借助于特定的应用程序来解析。如果二进制数据直接包含在 XML 文档中，通常称为内部存储。对于内部存储，大多数人使用 Base64 编码。这种编码技术将每个二进制数据序列编码为一个 ASCII 字符。它非常适用于 XML，因为 ASCII 字符不会和 XML 中的特殊标记冲突，这样数据不会出现中止的界定符。但是字码的转换需要特定的应用程序来完成。

假如要在 XML 文档中使用外部独立的二进制对象，在 XML 的语法中曾规定了一些使用外部二进制数据的方法，如下面的形式：

```

<!NOTATION jpg SYSTEM "Acdsee.exe">

<!ENTITY pic1 SYSTEM "show.jpg" NDATA jpg>

```

其中 Acdsee.exe 表示能够处理这种特殊格式的数据的应用程序名。未解析实体的应用是作为 ENTITY 类型的属性值来出现的，对于上面的实体，在 DTD 中包含下面的声明：

```

<!ELEMENT pic EMPTY>
<!ATTLIST pic name ENTITY #REQUIRED>

```

这样在文档可以按下面的方法使用：

```
<pic name="pic1">
```

注意实体引用中并不包含“&”，因为在 DTD 的 ATTLIST 声明中已经进行了声明，也就是说解析器知道这是一个实体引用。

假如考虑到 XML 文件的显示的话，使用 HTML 中的方法会比较合适，即下面的格式：

```
<IMG SRC="show.jpg">
```

也可以使用应用程序把上面的语句解释为指向 jpg 文件的 URL。

当然对于二进制数据的处理还有其他的方法、其他的技术，也同样是可行的。事实上

假如用于 XML 文档的显示的话，基于 URL 的方法是目前最简单的一种方法；如果用于文件的传送的话，使用 BASE64 编码应该比较有效。

3. 标准化

标准化首先涉及到的一个问题就是度量单位的表示，比如一个数据的单位“元”，可以表示的形式有<价格 单位=“元”>9.80</价格>，或者直接把单位写到数据的后面<价格>9.80元</价格>。当然也可以用属性来表示整个的“价格”。显然各种形式并没有标准化，这种工作到目前为止还是由文档的设计者来完成的，也就是说文档的设计者会解决各种词法的编码问题，一旦建立了 XML 模式，只需要应用设计的标准数字数据类型就可以了。

对于日期和时间，目前国际上也有很多标准，许多人提倡根据 ISO8601 的日期形式来进行标准化，以避免出现混乱。这种标准的日期形式为 YYYY-MM-DD，ISO8601 还定义了时间的表示法，其中包括时区。

事实上国际标准化组织（ISO）发布了很多的标准，包括像 ISO2955 的国际公制度量单位，ISO6093 的数值等。在编写 XML 文档的时候我们推荐读者使用标准化的数据类型。这样避免很多麻烦。我们也期待在不久的将来更多的通用数据类型被标准化，它们能够成为 XML 模式活动的一部分。

实体

在 XML 中实体属于物理文档结构的一部分，它不属于逻辑结构。实体应用的结果是实体直接被内嵌在引用出现的位置。从这个意义来说，实体对于逻辑的作用并不是很大，但是相对于外部实体来说，由于外部实体与 XML 文档是完全脱离的，那么可以随时的更新实体而不必改动 XML 文档，这对 XML 的维护来说是一件很有效的事。

事实上这种方法是很有用的，常常在 XML 文档中用到一些随时需要更新的文件，而这种文档的更新与 XML 文档有完全不同的更新周期或更新权限，那么使用外部实体可以大大的节省空间和时间。

外部实体的声明在前面二进制数据中提到，用外部实体代替一个外部文件的内容要使用 SYSTEM 关键字。例如：

```
<!ENTITY Sign SYSTEM "2.doc">
```

在 XML 文档被解析的时候，XML 处理器将用指定文件的内容来代替实体指示。

1.4 小 结

本章主要介绍了一些基本的信息建模原则，对建模语言也作了一些介绍。在介绍信息建模的时候，对静态模型和动态模型分别作了必要的介绍，这两种模型都与 XML 的设计有关，而静态永久性数据和暂时性消息这两种类型也是很有意义的。

在本章后面几节，将通过一个音乐网站的建立来说明信息模型的建立过程，并且将信息模型转换为 XML 文档的设计。

下面一章将介绍 XML 模式中一个重要的部分：DTD 模式。

第2章 DTD 模式

HOPE



声 明

本电子图书不包括此章，请参阅配套图书。

北京希望电子出版社

2000

出版社

第 3 章 XML 模式和 XML DR

本章导读:

由于 DTD 的局限性,使得 W3C 致力于寻找解决的方案,从而花费了大量的时间,但是其中几个已经形成的重要建议似乎都不能完全解决问题。其中包括颇具影响的微软的专有 XML 数据规范。当然 W3C 自己的工作草案就是制定了 XML Schema。W3C XML Activity Page 声明:“尽管 XML1.0 提供了一种机制,文档类型定义(DTD)给标记的使用加了限制,但是对 XML 文档的自动处理需要更严格更全面的工具。其需要主要体现在对应用软件各部分的结合、文档结构、属性和数据类型等等的约束。”

XML Schema 的语法完全遵循 XML 的语法规范,因此用户不用像学习 DTD 一样再学习一遍其他的语法,而可以自己使用。关于 Schema 在前面的第 1 章中已经给大家介绍过,这里我们只是表述一下按照信息模型构件起来的 XML 文档怎样使用 XML Schema(使用简化的 XML DATA)。

本章另一个重要的内容是 XML DR。XML DATA 是在微软的大力倡导和支持下,才被越来越多的人使用并用来开发原型甚至产品。本章介绍的是简化的 XML Data,也可以说是 XML Data 的一个子集。它连接着 IE5 中的 XML 解析器。由于在简化版的 XML Data 兴起的时候,XML 模式只是一个推荐的标准,因此从应用方面来看,XML Data 的应用似乎比 XML 模式的应用更加广泛。

3.1 XML 模式

在 Schema 之前,另有一种定义 XML 结构的方式,即 DTD,两者的区别在于:

(1) XML Schemas 是 XML 文档,而 DTD 有自己的特殊语法。这样一来,你只需懂得 XML 的语法规则即可编写 Schema,无需学习其他语法规则;xml 文件与 xml schema 文件可以用相同的语法分析器来解析,而无须写两套分析器;xml schema 有强大、易用的扩展功能。

(2) XML Schema 利用名域将文档中特殊的节点与 schema 说明相联系,一个 xml 文件可以有多个对应的 schema,而用 DTD 的话,一个 xml 文件只能有一个相对应的 DTD。

(3) XML Schema 内容模型是开放的,可以随意扩充,而 DTD 将无法解析扩充的内容。

(4) DTD 只能把内容类型定义为一个字符串。而 XML Schemas 允许你把内容类型定义为整型、浮点型、数据类型、布尔型或者许多其他的简单数据类型,而无须重定义。

3.1.1 XML Schema 介绍

Schema 的产生

关于 DTD 和 XML SCHEMAS 的关系,前面已经介绍过一些内容。事实上当初编写

XML 的时候, DTD 只是简单的从 SGML 中发展而来的, DTD 中明显还深刻着 SGML 的烙印。与 XML 的巨变和成功相比, DTD 尚没有实质性的飞跃。其中关键的一点是, DTD 有自己的特殊语法, 它虽然可以用以规定限制 XML, 但本身并不是 XML。我们需要认真学习 DTD 才能熟练地创建和编写 XML 文档; 而且我们必须有两套不兼容的解析器: 一套用来分析 XML, 如判断其结构是否良好; 另一套用来分析 DTD, 再用分析的结果去检查 XML 文档, 判断 XML 是否有效。DTD 只提供了非常有限的几种数据类型。DTD 中规定的文档内容都是字符数据, 连整型、浮点型、数据型、布尔型这样简单的数据类型都不提供, 更别谈更复杂的数据类型了。DTD 不支持名域(Namespace)机制, 虽然 DTD 可以为不同的 XML 所引用, 但一个 XML 文档只能有一个相对应的 DTD。不同的 DTD 并不能同时对同一个 XML 文档进行规定, 哪怕只是对其中的部分元素。也就是说传统的 DTD 机制使得 XML 的继承性和重用性受到限制。DTD 有扩展的机制, 但这个机制太复杂而且很脆弱。DTD 扩展机制的最大毛病在于不能清楚的表达相互之间的关系。两个有着完全相同内容的元素怎么做也不能互相联系。

同样的, 被定义为参数实体的不同属性之间不能建立任何联系。另外 DTD 中的内容模型是不开放的, 它不能随意扩充内容, 否则将无法被解析。以上的这些缺点事实上已经在束缚 XML 的发展, 重新制定符合 XML 的大纲机制已经是大势所趋, 这样的新机制就是 XML Schema。

XML Schema 的新特点

基于 DTD 到 Schema 的变化, XML 大纲提供了一系列的新改进, 这些改进大大弥补了 DTD 的不足:

- ☒ 丰富的数据类型。XML 大纲支持的数据类型包括: 数字型、布尔型、整型、日期时间、URI、十进制数等等。而且它还支持由这些简单的类型生成更复杂的类型。
- ☒ 可以由用户自定义数据类型, 称为 Archetype(原型)。比方说, 可以先定义 "PostalAddress" 这个数据类型, 然后定义 "ShippingAddress" 和 "BillingAddress" 为这个类型的两个元素。
- ☒ 属性分组。属性的应用范围是多种多样的。有的是所有元素都有的, 有的是专门为图形元素设定的。
- ☒ 原型可以更新。这是最重要的特色。DTD 定义的内容模式是封闭的, 而 XML 大纲定义的内容模式是开放的, 是可以更新的。
- ☒ Namespace(名域)的支持。自从 XML 引入了名域, 有效性的检查就变得复杂了。而在 XML 完全开发之前, 这个问题很难有完满的解决。

XML Schema 的整体结构

XML Schema 结构规定了 XML Schema 的定义语言。它提供了规定 XML 文档的结构和内容约束的机制, 在 XML 1.0 DTD 的基础上更迈进了一步。这个机制和 XML Schema 的第二部分“数据类型”共同来对 XML 文档进行定义。

一个 XML Schema 由类型定义和元素声明等部件组成, 以衡量格式良好的元素信息的有效性。进一步还可以另外规定这些元素项目和其子节点, 比如属性和元素的缺省值、元

素类型和属性表等。从抽象数据模式上看，XML Schema 的通用组成项是 Schema 组件 (Schema component)。一个 XML Schema 是许多 Schema 组件的集合。共有 12 类组件，他们可以分为 3 组。最基本的一组组件有：

- ☒ 简单类型定义
- ☒ 复杂类型定义
- ☒ 属性声明
- ☒ 元素声明

次要一些的组件有：

- ☒ 属性组定义
- ☒ 标识限定定义
- ☒ 模型组定义
- ☒ 记法声明

以上两组的组件都有名称，可以单独地出现在 Schema 的内容中，而且可以被单独的访问。最后，“helper”组件提供了其它组件的细小部分。它们不能在 Schema 中单独出现而且不能独立访问，这些组件包括：

- ☒ 注释模型组
- ☒ 粒子
- ☒ 通配符

3.1.2 Schema 语法

现在我们已经知道 Schema 实际上是使用 XML 语法写成的，这意味着 XML 模式实际上只是 XML 的另一个应用而已（一个为了定义 XML 文档类的词汇表），正是如此，XML 才拥有了一个可以描述自己的模式。每个 Schema 都有自己定义模式的元素和属性的地方，我们称之为结构。在这里，元素的内容模型得到了描述，内容模型明确地描述了允许的元素的内容结构。可以说，结构是 XML 模式的核心。

一个模式由导言、不定数量的定义和声明组成。下面先从导言开始讨论。

导言

导言放在根元素里面，它至少包含属性的三个方面，我们来看它的一般形式：

```
<schema targetNS="http://10.12.13.6/122_1.xsd" version="1.0"
xmlns="http://www.w3.org/1999/XMLSchema">
```

其中 targetNS 用来说明正在使用的模式的命名空间和 URI；version 用来指明模式的版本号；xmlns 为 XML 模式规范提供命名空间。

在这里我们假定模式驻留在 10.12.13.6 上，并且叫做 122_1.xsd，这是 XML 模式的文件扩展。它保留在第一个版本中，缺省的命名空间声明是 XML 模式：结构的应用，并且是一个关闭的模型模式。这意味着所有与这个模式一致的文档将要完全由模式来定义，而完全不含有任何外部内容。

类型定义

类型定义包括简单类型定义和复杂类型定义两类。需要强调的是，XML 模式定义的结构仅仅依赖于类型定义。这允许一个模式设计者去声明能在模式中使用的扩展类型。它们被用于说明元素和属性的内容和类型。

下面来看简单的类型定义是如何来实现的，在 XML 模式中通过 `datatype` 来实现。可以这样来定义一个简单类型定义：

```
<datatype name="smallInt" source="integer">
  <minExclusive value="1"/>
  <maxExclusive value="99"/>
</datatype>
```

复杂的数据类型定义主要是针对一些简单类型定义无法完成的定义而产生的，相对来说，它的定义可能比较麻烦，它通过 `<type>` 来实现。如下所示为一简单程序：

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

复杂类型定义在 XML 模式中很重要，它组成了很多重要的元素内容模型，像上面的例子就是把元素和属性组成了一个新的元素内容模型。

元素声明

在 DTD 中我们使用 `<!ELEMENT` 语法来声明一个元素，在 Schema 中我们同样要使用 `element` 元素，不过一般使用小写，如下面的形式：

```
<element name="音乐"/>
```

在这里 `name` 属性简单的拥有一个我们正在生成的元素的值，也就是我们在 XML 中需要声明的元素名。简单元素由数据类型和一系列属性声明的引用或一个属性组的引用组成。这与 DTD 的这种声明有点类似：除了内容被赋予类型，元素只包含 PCDATA。比如：

```
<element name="music" type="string"/>
<element name="price" type="float"/>
```

注意，DTD 中没有字符串和浮点数值类型的概念。

属性和属性组

在 Schema 中，属性声明由 `<attribute>` 元素组成，`attribute` 元素至少包含一个 `name` 属性。当然也有可选的 `cardinality` 属性：`minOccurs` 和 `maxOccurs`，它用来指出属性是否显示，以及如何显示。一个属性还需要属性的数据类型，比如字符串类型或整型。属性的声明可能

是 default 和 fixed 属性。我们看下面的例子：

```
<attribute name="价格" type="float" default="9.8"/>
```

有时候我们需要建立一个属性组，XML 模式则提供了定义这些属性组的方法，如下面的例子：

```
<attributeGroup name="music">
  <attribute name="price" type="float" default="9.8"/>
  <attribute name="singer" type="string"/ >
  <attribute name="company" type="string"/>
</attributeGroup>
<type name="musictype">
  <attributeGroup ref="music"/>
</type>
```

这个例子声明了一个 music 的属性组，并且在 musictype 里使用它。

内容模式

对于内容模式，XML 模式定义了另外一个属性来表示，这个属性就是 content 属性，content 属性说明了哪些元素能够被包含，关于 content 的属性值和含义可以参看下面的表 3-1。

表 3-1 Content 的属性值和含义

Content 的属性值	含义
Unconstrained	任何类型的内容
Empty	空元素
Mixed	元素和字符类型

在使用的的时候可以这样来使用，如：

```
<type name="musictype" content="unconstrained"/>
<type name="music" content="empty"/>
<type content="mixed">
  <element .../>
</type>
```

事实上我们经常遇到拥有很多只含有元素的内容，这样有时候就要为这些元素排序，也就是要定义一个规则把这些元素按照某种规则排列起来。相对于元素组<group>来说，它有一个 order 属性的值用来表示排序。它的值是可选的两个关键字，下面把它们与 DTD 进行一下比较，如下表 3-2 所示。

表 3-2 排序关键字的含义及与之等价的 DTD

排序关键字	含义	DTD 等价物
seq	元素必须按准确的顺序排序	, (逗号)
choice	模型元素之一准确的出现	

模型组的建立

模型组的建立在 XML 模式中使用<group>语句。一个模型组可以包含两个或更多的元素，元素的排序方式可以是上面介绍的两种方式之一。当然也同样可以利用 minOccurs 和 maxOccurs 规定元素出现的次数。比如下面的一段程序：

程序清单 3.1:

```
<type minOccurs="1" maxOccurs="2">
  <group order="seq">
    <element type="A"/>
    <element type="B"/>
  </group>
  <group order="choice" minOccurs="0" maxOccurs="4">
    <element type="C"/>
    <element type="D"/>
  </group>
</type>
```

在这个模型中，文档以 AB 序列开始，至少出现一次，但是最多也只能出现两次。而且 AB 是以顺序出现的。相对于 CD 来说，它们出现的次数可以有大的差异，而且出现的次序也没有特别的规定。

实例

事实上了解了上述的一些 XML 模式语法，我们已经可以编写任意的 XML 模式了，毕竟 XML 模式（Schema）本身就是用 XML 语法来编写的。从这个意义上来说，XML 模式比 DTD 来的简便多了。下面给出一段元素的定义，在结构上它是完全符合 XML 结构的，但它的声明方式比 DTD 结构更加清楚。

程序清单 3.2:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/1999/XMLSchema">
  <element name="个人物品">
    <type>
      <group order="seq">
        <element name="个人信息" type="个人信息Type"/>
        <element name="书籍" type="书籍Type" minOccurs="1"
maxOccurs="unconstrained"/>
        <element name="音乐" type="音乐Type" minOccurs="1"
```

```

maxOccurs="unconstrained"/>
    </group>
  </type>
</element>
<element name="个人信息">
  <type name="个人信息Type" content="elementOnly" >
    <group order="seq">
      <element name="出生日期" type="decimal"/>
      <element name="职业" type="string"/>
      <element name="姓名" type="string"/>
    </group>
  </type>
</element>
<element name="书籍">
  <type name="书籍Type" content="elementOnly" >
    <group order="seq">
      <element name="编号" type="decimal"/>
      <element name="书名" type="string"/>
      <element name="作者" type="string"/>
      <element name="价格" type="decimal"/>
      <element name="类别" type="string"/>
    </group>
  </type>
</element>
<element name="音乐">
  <type name="音乐Type" content="elementOnly" >
    <group order="seq">
      <element name="编号" type="decimal"/>
      <element name="专辑名" type="string"/>
      <element name="演唱者" type="string"/>
      <element name="价格" type="decimal"/>
      <element name="类别" type="string"/>
    </group>
  </type>
</element>
</schema>

```

上面这段程序代码是针对于下面的这段 XML 代码写的。

程序清单 3.3:

```

<?xml version="1.0" encoding="GB2312"?>
<个人物品>
  <个人信息>
    <出生日期>1978.2</出生日期>
  </个人信息>
</个人物品>

```

```

    <职业>学生</职业>
    <姓名>洋洋</姓名>
  </个人信息>
  <书籍>
    <编号>B-001</编号>
    <书名>中文版WINDOWS98详解</书名>
    <作者>程月</作者>
    <价格>35.00</价格>
    <类别>计算机</类别>
  </书籍>
  <音乐>
    <编号>Y-001</编号>
    <专辑名>海浪</专辑名>
    <价格>69.00</价格>
    <类别>CD</类别>
    <演唱者>黄品源</演唱者>
  </音乐>
</个人物品>

```

3.1.3 XML 模式的数据类型

真实世界依靠字符串、数值、集合等概念组合各种类型的数据。相对于 XML 模式来说它有两类数据，一类称之为原始数据，原始数据是组合成其他数据的基础，也是应用最广泛的数据类型；相对于原始数据，还有一类就是派生类的数据，这些数据可以是在不断应用过程中逐渐形成的，还有一些可能是用户自己定义的类型，下面列两张表格来表示它们。

表 3-3 XML 模式的原始数据类型

元素数据类型	定义（含义）
string	字符串类型
boolean	集合（true,false）
float	实数的标准数学概念，双精度 32 为浮点
double	实数的标准数学概念，双精度 64 为浮点
decimal	实数的标准数学概念，覆盖范围比 double 小
timeInstant	日期和时间的联合，用来定义一个准确的时间，模板为 YYYY-MM-DDThh:mm:ss.sss 的形式
Uri	URI 引用
Binary	任意长的二进制数据体
recurringInstant	一种带有固定频率再现的时间实例
timeDuration	日期和时间的联合，用来定义一段时间、间隔或持续时间。词汇模板为 PnYnMnDTnHnMnS

上表 3-3 是原始的类型，下面的表 3-4 给出了 XML 模式中的一些派生的类型。

表 3-4 XML 模式的派生数据类型

生成的类型	基础类型	含义
language	string	自然语言标识符
NMTOKEN	NMTOKENS	XML1.0NMTOKEN
NMTOKENS	String	XML1.0NMTOKENS
Name	NMTOKEN	XML1.0 名称
Qname	Name	XML1.0 限定名
NCNAME	Name	“未开拓”名称
ID	NCNAME	属性类型的 ID
IDREF	IDREFS	属性类型
IDREFS	string	属性类型
ENTITY	ENTITIES	ENTITY
ENTITIES	string	ENTITIES
NOTATION	NCNAME	NOTATION
Integer	Decimal	离散数值类型
positive-integer	Integer	正整数
non-positive-integer	Integer	非负整数
non-negative-integer	Integer	负整数
date	RecurringInstant	标准日期
time	RecurringInstant	时间

在这里还要说明的是，在 XML 模式中用户自定义类型是使用 datatype 来表述的，使用 source 属性可以引用基本的类型，比如下面的例子就定义了一个新的类型。

```
<datatype name="price" source="decimal">
```

3.1.4 属性声明

下面该说 Schema 的属性定义了，Schema 中用来定义属性的元素有两个，AttributeType 和 attribute。前一个是声明属性的，后一个说明一个元素中究竟包含了哪些属性，也既是属性的引用。

属性的语法表达如下：

```
<AttributeType
  name="属性名"
  dt:type="属性类型"
  dt:value="枚举值列表"
  default="缺省值"
  required="{yes|no}" />
```

属性名和属性的类型

在属性定义中，属性名是必需的，它声明该属性的类型的名称。在使用属性时我们要用属性名来引用属性。

Schema 中属性的类型与 DTD 的大同小异，同时 Schema 还支持一些扩展属性，在前面讲述元素的类别中我们曾列了两个表来说明数据的类别，这些数据类别同样适合属性的数据类型使用，读者可以参照前面的内容。

属性类型的声明语句为 `dt:type="属性类型"`，比如我们需要声明一个标号的属性，类型为“Id”，就可以这样写：

```
<AttributeType
  标号
  dt:type="Id"
  required="yes">
</Attribute>
```

1. dt:value:

只有当 `dt:type` 取值为“enumeration”时才有效，此时，`dt:value` 值需列出所有可能的值。

Default:

`default` 是指定该元素的缺省取值。`default` 取值必须是有效的，例如，当 `dt:type` 取值为“enumeration”时，`default` 的取值必须来自 `dt:value` 所列出的值。

2. required:

`required` 指定该属性对于引用它的元素是否是必需的。`required` 只有两个可选值“yes”和“no”，“yes”表示该属性是必须有的，“no”表示该属性是非必须的。

当声明完属性后我们要在元素中引用它们，但是在引用之前，我们还需要使用 `attribute` 把元素和属性联系起来。`attribute` 的语法表达如下：

```
<attribute
  type="attribute-type"
  default="default-value"
  [required="{yes|no}"]/>
```

`attribute` 实际上只是对 Schema 中 `AttributeType` 声明的引用，而具体引用什么属性类型就要看 `attribute` 中的 `type` 属性了，`type` 属性指明了要引用的属性类型，所以它必须取某个定义过的属性名即属性声明中的 `name`。至于 `default` 和 `required` 属性和 `<AttributeType>` 是一样的，这里不再赘述。下面举几个元素属性的例子：

```
<AttributeType
  name="等级"
  dt:type="enumeration"
  dt:value="A B C D"
```

```
required="yes"/>
```

```
<AttributeType
  name="编号"
  dt:type="Id"
  required="yes"/>
```

```
<ElementType name="商品" content="elonly">
  <attribute type="等级" default="A">
  <attribute type="编号">
</ElementType>
```

属性分组

在介绍 XML Schema 优点时，我们曾提到 Schema 的一大优点就是属性分组，属性分组的好处是可以把一个元素的所有属性包含在一起，这样容易看懂也容易理解，使用起来也很方便，只要在元素中直接写入属性的组名就可以引用组中的所有属性了。按照属性分组的观点，我们把商品的一些属性分为一组，如下所示：

```
<attrGroup name="商品属性">
  <AttributeType name="编号" dt:type="Id" required="true"/>
  <AttributeType name="等级" dt:type="enumeration" dt:value="A B C D"
    required="true"/>
  <AttributeType name="厂家" dt:type="string" required="true"/>
  <AttributeType name="价格" dt:type="fixed.14.4"/>
  <AttributeType name="备注" dt:type="string"/>
</attrGroup>
```

可以这样来使用：

```
<ElementType name="商品" content="empty">
  <AttrGroupRef name="商品属性"/>
</ElementType>
```

3.1.5 Schema 实例

下面举两个简单的 Schema 实例。源文件如下所示。

程序清单 3.4:

```
<?xml version="1.0" encoding="GB2312"?>
<Schema name="Untitled-schema" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="编号" model="closed" content="textOnly" dt:type="string"/>
  <ElementType name="出品公司" model="closed" content="textOnly" dt:type="string"/>
  <ElementType name="库存" model="closed" content="eltOnly" order="seq">
```



```

    <AttributeType name="xmlns" dt:type="string"/>
    <attribute type="xmlns"/>
    <element type="音乐" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="联系人" model="closed" content="textOnly" dt:type="string"/>
  <ElementType name="名称" model="closed" content="textOnly" dt:type="string"/>
  <ElementType name="演唱者" model="closed" content="textOnly" dt:type="string"/>
  <ElementType name="音乐" model="closed" content="eltOnly" order="seq">
    <AttributeType name="种类" dt:type="enumeration" dt:values="CD MP3"
required="yes"/>
    <attribute type="种类"/>
    <element type="编号" minOccurs="1" maxOccurs="1"/>
    <element type="名称" minOccurs="1" maxOccurs="1"/>
    <element type="演唱者" minOccurs="1" maxOccurs="1"/>
    <element type="出品公司" minOccurs="1" maxOccurs="1"/>
    <element type="联系人" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>

```

程序清单 3.5:

```

<?xml version="1.0" encoding="GB2312"?>
<Schema name="Untitled-schema" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="编号" model="closed" content="textOnly" dt:type="string"/>
  <ElementType name="单位" model="closed" content="textOnly" dt:type="string"/>
  <ElementType name="订单" model="closed" content="textOnly" dt:type="string"/>
  <ElementType name="客户" model="closed" content="eltOnly" order="seq">
    <AttributeType name="种类" dt:type="string" required="yes"/>
    <attribute type="种类"/>
    <element type="编号" minOccurs="1" maxOccurs="1"/>
    <element type="客户地址" minOccurs="1" maxOccurs="1"/>
    <element type="姓名" minOccurs="0" maxOccurs="1"/>
    <element type="单位" minOccurs="0" maxOccurs="1"/>
    <element type="订单" minOccurs="1" maxOccurs="1"/>
  </ElementType>
  <ElementType name="客户地址" model="closed" content="textOnly" dt:type="string"/>
  <ElementType name="客户名单" model="closed" content="eltOnly" order="seq">
    <AttributeType name="xmlns" dt:type="string"/>
    <attribute type="xmlns"/>
    <element type="客户" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="姓名" model="closed" content="textOnly" dt:type="string"/>
</Schema>

```

程序清单 3.6:

```

<?xml version="1.0" encoding="GB2312"?>
<Schema name="Untitled-schema" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="付款方式" model="closed" content="textOnly" dt:type="string"/>
  <ElementType name="今日预订" model="closed" content="eltOnly" order="seq">
    <AttributeType name="date" dt:type="date" required="yes"/>
    <attribute type="date"/>
    <AttributeType name="xmlns" dt:type="string"/>
    <attribute type="xmlns"/>
    <element type="预订" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="客户编号" model="closed" content="textOnly" dt:type="string"/>
  <ElementType name="数量" model="closed" content="textOnly" dt:type="i1">
    <AttributeType name="单位" dt:type="enumeration" dt:values="盘 张" required="yes"/>
    <attribute type="单位"/>
  </ElementType>
  <ElementType name="音乐编号" model="closed" content="textOnly" dt:type="string"/>
  <ElementType name="预订" model="closed" content="eltOnly" order="seq">
    <element type="客户编号" minOccurs="1" maxOccurs="1"/>
    <element type="音乐编号" minOccurs="1" maxOccurs="1"/>
    <element type="数量" minOccurs="1" maxOccurs="1"/>
    <element type="付款方式" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>

```

Schema 的工作还需要不断的完善,就目前的 xml 模式来看,仍然有很多完整性约束是无法用模式表达的,必须有程序来限制,例如目前还无法定义属性之间的约束(死亡时期应该晚于出生日期),它只能定义对直接子元素的约束。实际上,有些有效性约束只能在应用程序升级时完成,正如关系数据库中的更加成熟的约束定义。但是总的说来,Schema 机制确实比 DTD 机制有明显的进步。

3.2 XML DR

3.2.1 XML DATA 简介

当初由于 DTD 的局限性,只推荐使用下面几种模式:文档内容描述(DCD)、面向对象的 XML(SOX)模式以及文档定义标记语言(DDML)等,还包括 XML DATA。相比较而言,XML DATA 比 DTD 有很大的优势,这些优势主要体现在:

用 XML 写

XML Data 为了模式结构使用 XML 词汇表,它允许用户不必去学一种新的语法而去读

写模式。从这个意义上来说，它比 DTD 来的简单。

数据分类

XML Data 添加了强大的元素和属性分类，而且回答了我们最初对于 DTD 的异议。这些可能由数据类型命名空间定义的基础数据类型，或者复杂的，由设计者提供的模式所提供的用户自定义类型。

允许数值上的约束

XML Data 要求在元素和属性数值范围上的约束被定义，比如最小值和最大值等。而在 DTD 中我们显然无法做到这一点。

类型继承

XML Data 支持类型继承，这些使我们在试着用 XML 解决的问题里描述实体时发展和扩展了元素。我们能够写一些具有普遍意义的超类型声明。

开放和关闭的内容模式

XML Data 另一个强大的特性就是开放和关闭的内容模式概念。经典的 DTD 是一个关闭的模式。但是在 XML Data 中可以使用 content 属性来决定是开放还是关闭内容模式。而开放的内容模式下我们可以混合命名空间。

由于微软的大力倡导和支持，使得 XML DATA 开始被越来越多的人使用并用来开发原型甚至产品。本节讲述的是简化的 XML Data，它可以说是 XML Data 的一个子集。它连接着 IE5 中的 XML 解析器。由于在简化版的 XML Data 兴起的时候，XML 模式只是一个推荐的标准，因此从应用方面来看，XML Data 的应用似乎比 XML 模式的应用更加广泛。对于模式的完整参考，可以参看以下地址：<http://msdn.microsoft.com/xml/reference/schema.start.asp>。XML Data 在有些命令方面还是和 XML 模式很相像的，下面表 3-5 把一些常用的命令行表示如下。

表 3-5 XML 模式和 XML DR 命令

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

3.2.2 XML DR 语法详解

XML DR 的一般结构

下面讲述这些 XML DR 的具体结构。

在讲述元素定义之前，首先要声明的是 XML DR 的基本结构，XML DR 是由一组元素构成的，其根元素是<Schema>，<Schema>元素是 XML DR 中第一个出现的元素，用于表明该 XML 文档是一个 XML DR 文档。相应的，XML DR 的结束标记</Schema>一般在文档的末尾。这样，一个 Schema 文档的结构如下：

```
<Schema name="Schema名" xmlns="命名空间">
    元素和属性定义等内容
</Schema>
```

Schema 具有两个属性，name 指定该 Schema 的名称，可以省略。而 xmlns 则指定 Schema 文档中包含的 namespace（命名空间）。在 Schema 中，一个 XML 文档已包含多个命名空间，一般的在编写 Schema 文档的时候，这两句是不能少的：

```
xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schema-microsoft-com:datatypes"
```

第一个命名空间 xmlns="urn:schemas-microsoft-com:xml-data"，它说明的是引用微软 Schema 类型定义，指定本文档是一个 XML Schema 文档；第二个是 xmlns:dt="urn:schema-microsoft-com:datatypes"，它表示引用微软 Schema 数据类型定义，这样在文档中就可以使用 Schema 中定义过的数据类型。

当然假如我们需要应用到其他文档中定义过的元素或属性等内容的话，可以再加入其它的命名空间，比如：

```
<Schema
    xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schema-microsoft-com:datatypes"
    xmlns:use="http://www.xml_study.edu.cn/132-2.xml">
    元素和属性定义等内容
</Schema>
```

元素定义

在<Schema>元素下可以加入各元素的定义语句。

元素声明的语法表达如下：

```
<ElementType name="元素名"
    content="{empty|textonly|eltonly|mixed}"
    dt:type="元素类型"
    order="{one|seq|many}"
```

```

    model="{open|closed}"
    maxOccurs="{0|1}" minOccurs="{1|*}" >
</ElementType>

```

内容 (content)

一般来说, 一个元素声明的最简单 schema 声明如下:

```
<elementType 元素名A/>
```

该句声明没有指明元素的内容和数据类型, 都取默认值, 假如要指明元素的内容的话, 就要用到 content(内容)。content 规定内容的属性, 用于描述元素中的内容类型。在可选的内容中, empty 表示元素的内容为空, 即空元素; textonly 顾名思义, 指元素内容中只能出现字符串; eltonly 则表示元素中只能包含子元素; mixed 是涉及的范围最广, 它可以包含元素和已分析的字符数据, 这很像 DTD 中的 ANY。下面例子是使用 content 来规定元素的内容:

```

<elementType name="A" content="empty"/>
<elementType name="B" content="textonly"/>
<elementType name="C" content="mixed"/>
<elementType name="D" content="eltonly">
    <element type="E">
    <element type="F">
</elementType>

```

minOccurs 和 maxOccurs

minOccurs 和 maxOccurs 表示元素在该项中最少出现的次数和最多出现的次数, 我们可以取 minOccurs 为 0 或 1, 表示该元素可以出现的最少次数为 0 次或 1 次; maxOccurs 属性可以取 “1” 和 “*”, 表示该元素出现的最多次数为 1 次或是任意多次。假如省略任意的 minOccurs 或 maxOccurs, 则系统都会默认为 1。

在元素定义中, 有一些处理器支持 occurs 属性:

```

<ElementType name="A">
    <element type="B" occurs="REQUIRED"/>
    <element type="C" occurs="ONEORMORE"/>
</ElementType>

```

occurs 的值有 REQUIRED, OPTIOANL, ONEORMAORE, ZEROORMORE 分别对应必须出现一次、可选出现 (0 或 1 次)、一次以上、任意次数。缺省值是 REQUIRED。

数据类型

dt:type 这个属性用于指定元素文本的数据类型, 当然它们也用来指明属性的数据类型。在 DTD 中, 元素的数据类型只有字符串一种 (#PCDATA), 而在 XML DATA 中, 内建的

数据类型达到了二十多种，如下表 3-6 所示。

表 3-6 XML DATA 的数据类型

Schema 中的数据类型	表示的含义
String	字符串形式，相当于 DTD 中的 #PCDATA
Enumeration	相当于 DTD 中的 ENUMERATION
Id	相当于 DTD 中的 ID
Idref	相当于 DTD 中的 IDREFS
Entity	相当于 DTD 中的 ENTITY
Entities	相当于 DTD 中的 ENTITIES
Notation	相当于 DTD 中的 NOTATION
Nmtoken	相当于 DTD 中的 NMTOKEN

扩展数据类型共有 23 中，包括 bin.base64, bin.hex, char, data, dateTime.tz, fixed.14.4, float, int, number, time.tz, i1, i2, i3, i4, r4, r8, ui1, ui2, ui4, uri, uuid，已经可以覆盖相当广泛的应用。下面表 3-7 把常用的一些扩展数据类型说明如下：

表 3-7 常用扩展数据类型

数据类型	表示的含义
Boolean	布尔型：0 或 1，表示 false 和 true
Char	单字符如 “a”，“b” 等
Time	时间类型，符合 ISO8601 格式，无日期和时区部分。如 “09:34:28”
Data	日期类型，符合 ISO8601 格式且无时间部分，如 “2000-12-04”
DateTime	日期类型，符合 ISO8601 格式，带可选时间部分但无时区部分，如 “2000-12-04T09:34:28”
Fixed.14.4	数值类型，同 “number” 类似，精度上小数点前不超过 14 位，小数后不超过 4 位，如 3.1415
Float	实数类型，位数不受限制，可以包含符号位和小数点以及指数。有一定取值类型
Int	数值类型，可以包含符号位，但不含小数位和指数位。如 1, -10, 4568 等
Number	数值类型，位数不限，可以包含符号和小数位以及指数，有一定取值范围

使用时只需在 dt:type 的等号后面写上类型就好了，例如希望商品的编号不是字符串形式，而是数值形式，就可以这样定义：

```
<ElementType name="编号" dt:type="number">
```

也可以把价格定为整数 int 类型：

```
<ElementType name="编号" dt:type="int">
```

读者可以按照自己的需要使用。

再深入一点的说,用户可以根据需要定义自己的数据类型,当然,必须以一种基本类型为基类来派生。一般的,在已定义过的数据类型中基本上都可以找到自己需要的类型,但是有时候这些类型还是不够的,比如说我们设计一个电话号码的元素,我们希望的是电话号码必须为 7 或 8 位,或者是四位区号再加上电话号码,这时我们需要定义一个数据类型 (data type) 来验证它的合法性,事实上,在原有的数据类型中是不存在这样的类型的,我们需要自己定义一个数据类型,可以这样来定义:

```
<datatype name="telCode">
  <basetype name="integer"/>
  <lexicalRepresentation>
    <lexical>9999999</lexical>
    <lexical>9999-9999999</lexical>
    <lexical>99999999</lexical>
    <lexical>9999-99999999</lexical>
  </lexicalRepresentation>
</datatype>
```

这样我们定义了一种新的数据类型 telCode,这种数据类型的基类是整数类型(integer),从第三行开始定义给出了该类型所允许内容的详细情况。定义指出,该元素可以是一个 7 位的数字字符;也可以是一个带 4 位区号的 7 位数字字符,也可以是 8 位的数字字符。

定义了数据类型后,我们希望在 XML Schema 中引用它们,我们可以这样来使用定义好的数据类型:

```
<ElementType name="电话号码">
  <datatype name="telCode">
</ElementType>
```

当然我们也可以引用别的文件中定义过的数据类型,这是我们需要用到名域(或称为命名空间)。

元素的顺序 (order) 和元素分组 (group)

order 的取值有三个,分别为 seq, one 和 many, seq 代表 Schema 中定义元素在 XML 文档中出现的顺序必须和定义时的顺序是一样的。例如希望元素 A 下面的元素按顺序出现,则:

```
<elementType name="A" order="seq">
  <element type="B"/>
  <element type="C"/>
  <element type="D"/>
  <element type="E"/>
</elementType>
```

这样定义后，子元素必须按照 B, C, D, E 的顺序出现，假如在使用该 Schema 的 XML 文本中出现顺序不对了，那么该文档就不是有效的 XML 文档了。

one 代表的是“选一”的结果，假如把该元素的 order 声明为 one 的话，那么它下面的子目录子能出现一个，比如：

```
<elementType name="A" order="one">
  <element type="B"/>
  <element type="C"/>
  <element type="D"/>
  <element type="E"/>
</elementType>
```

这时在 A 目录下面就只能出现 B, C, D, E 元素中的一个。

many 类似于 one，假如一个元素的 order 声明为 many 的话，那么它的子元素可以以任意顺序和数量出现。

有时候可以省略 content 的值，这时候系统会取默认值，假如一个元素声明为 eltonly 的话，则该元素的 order 默认值为“seq”；假如为“mixed”，则为“many”。

当试图改变子元素出现的数量和顺序时，不可能仅靠 order 属性就能完成。事实上，我们有时候希望在一大堆的子元素中的两个元素是“二选一”的关系，有时候希望出现的子元素必须同时出现的，那么就要用到 group 属性。比如：

```
<ElementType name="地址">
  <group order="one">
    <element type="省">
      <element type="直辖市">
    </group>
    <element type="城市">
      <element type="街道">
    </ElementType>
```

以上的语句有点类似于 DTD 中的括号：

```
<!ELEMENT 地址((省|直辖市),城市,街道)>
```

//group 的一般表达式如下：

```
<group maxOccurs="{0|1}"
  minOccurs="{1|*}"
  order="{one|seq|many}">
  内容
</group>
```

当一些子元素被定义为 group 之后，group 下的子元素相当于隶属于 group 这个元素，我们可以定义该 group 出现的次数，也可以定义该 group 中元素出现的数量和顺序。比如：

```

<ElementType name="A">
  <group maxOccurs="*" order="many">
    <element type="B"/>
    <element type="C"/>
    <element type="D"/>
  </group>
  <element type="E"/>
</ElementType>

```

这样定义后该 group 在元素 A 中至少出现一次，而且 group 中的元素是任意可选的。

model 属性

model 属性可以选 open 和 close，open 表示该元素类型是开放的，当一个元素设置为 open 后，该元素就可以从别的地方继承子元素，也可以被别的元素继承，即该元素下面还可以出现其他的子元素，比如：

```

<ElementType name="被继承">
  <element type="C"/>
</ElementType>
<ElementType name="继承">
  <element type="A"/>
  <element type="B"/>
  <superType type="被继承"/>
  <element type="C"/>
</ElementType>

```

上面的例子中我们从“被继承处”继承了一个“C”的子元素。

假如一个 model 声明为 open 的话，也可以利用命名空间在 XML 的文档中使用没有在该元素下定义过的元素。比如我们假定在商品元素下已经定义了名称、价格、等级等三个子元素并且商品的 model 属性规定为 open，这时在文档中我们可以写下面的代码：

```

<商品>
  <名称>华硕光驱40X</名称>
  <厂家>华硕电脑公司</厂家>
  <等级>A</等级>
  <详细资料 xmlns="http://10.12.41.6/message.xml">
    <批量价>386.00</批量价>
    <零售价>400.00</进价>
  </详细资料>
</商品>

```

名域问题会在后面的章节中详细介绍，读者暂时可不必细究。
默认的情况下，在 XML-DR 中的内容模式是开放的。

相反的情况是“closed”。当一个元素的 model 属性设置为 closed 之后，该元素下面就只能包含该元素下面直接的子元素，而且别的元素也不能引用它里面的元素了。

当然 model 属性可以选另外的一个值 refinable，这涉及到我们在前面讲 Schema 优点时提到的原型定义 (Archetype)，原型定义是用户真正自己定义的一种数据类型，假如在 XML 文档中多处需要用到同样的一些元素，我们可以定义一个原型把这些元素包含进去，然后在别的多处使用这个原型，这显然大大节省了空间和时间，比如：

```
<archetype name="地址">
  <element type="国家" minOccurs=0 maxOccurs=1/>
  <element type="省"/>
  <element type="城市"/>
  <element type="街道"/>
  <element type=""" 邮政编码"/>
</archetype>
```

这样定义后，使用时只需把原型包含进要加入该原型的元素下面就可以了：

```
<elementType name="目的地">
  <archetypeRef name="地址"/>
</elementType>
```

定义了原型后，假如我们想用原型来定义另一个原型，就要使用 model="refinable"，具体的做法如下：

```
<archetype name="地址" model="refinable">
  <element type="街道">
  <element type="城市">
</archetype>
<archetype name="美国地址">
  <refines>
    <archetypeRef name="地址"/>
  <refines>
    <element type="州 (state)"/>
    <element type="zip"/>
</archetype>
```

其中前面一段是定义一个可以被重新定义的原型，后面的就是沿用原型来定义另一个原型了。

在定义中使用了<refines>标签，该标签在定义原型时是用来引用另一原型的。

3.2.3 XML DR 中的名域空间

讲完了 XML DR 的元素和属性定义，有必要提一下 DR 中的命名空间的问题。关于名域的深入探讨将放在后面的章节中讨论，这里我们先简单介绍一下 XML DR 中的命名空间

问题。Schema 支持名域这一点在 XML Schema 的优点中写得很明确。DTD 是不支持名域的，而 XML Schema 很完美的支持名域，这表现在利用 Schema 机制我们可以引用文件中定义过的元素、属性和数据类型等，当然也包括原型。在很多情况下都需要用到外部的 Schema 大纲，事实上在前面讲 XML Schema 的一般结构时就提到了命名空间的问题，我们使用的是如下的形式：

```
<Schema
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schema-microsoft-com:datatypes">
  元素和属性定义等内容
</Schema>
```

前一句声明指明本文档是一个 XML DR 文档；后一句声明很明显引用了该命名空间（urn:schema-microsoft-com:datatypes）下的数据类型定义，我们在 Schema 中用 dt:type 来指明元素和属性的类型。

下面来说明用命名空间来引用其他的大纲中的数据。要引用在另一个大纲里定义过的元素，必须做两件事。首先，使用名域声明来引出另一个大纲，该声明包括一个以“xmlns:”开头的属性，其后是可选择的前缀名，最后是另一个大纲的统一资源标识符（URI）。比如：

```
<ElementType name="订单" xmlns:A="http://bme.telorg.edu.cn/134_1.xml">
```

上面的话指明了一个 URI 为“http://bme.telorg.edu.cn/134_1.xml”的大纲，并且给出它的前缀名为 A，前缀是在本地引用该名域的局部名称，当知道了该名域的具体位置后，就可以在 Schema 中使用该名域下的元素或其他资源了。使用是非常简单的，比如：

```
<ElementType name="收件人" xmlns:A="http://bme.telorg.edu.cn/134_1.xml">
  <element type="A:地址"/>
</ElementType>
```

前缀“A”在这里与嵌套名域声明相匹配，这样“A:送货地址”表示的就是由 http://bme.telorg.edu.cn/134_1.xml 中的大纲定义的“送货地址”元素了。如上定义之后，XML 文档相应的也产生了影响，在文档中使用“送货地址”这个元素必须也要指明域名 A 了。

```
<订单>
  <A:收件人>
    <A:姓名>马艳飞</A:姓名>
    <A:街道>王府井46#</A:街道>
    <A:城市>北京</A:城市>
    <A:直辖市>北京</A:直辖市>
    <A:邮编>100001</A:邮编>
  </A:收件人>
  <订货日期>2000-12-2<订货日期>
```

```
<发货日期>2000-12-12</发货日期>
<备注>护肤系列</备注>
<货物单>
  <商品>
    <名称>珊拉那止痘系列</名称>
    <等级>A</等级>
    <价格>56.88</价格>
  </商品>
</货物单>
</订单>
```

从上例中可以很清楚的理解“收件人”元素是来自于标号为“A”的大纲的，至于大纲的具体位置，是在 Schema 中定义时声明过的。对于前缀，可以任意改变，读者可以使用自己喜欢或容易理解的前缀，比如我们把“A”改为“发货地址”，这时你只需要把文档中“A”全改为“发货地址”就可以了。

3.2.4 实体声明和记法声明

实体声明

XML DR 中的实体声明和记法声明与 DTD 不同，但是基本的原理是一样的。在 Schema 中实体声明分为三类，内部实体、外部解释实体和非解释实体。

内部实体指的是定义一般的实体，也是最简单的形式，一般形式为：

```
<textEntity name="实体名">
  需要设为实体的文本内容
</textEntity>
```

外部解释实体是指一个格式良好的 XML 文档，声明外部实体的一般方法是：

```
<externalEntity name="实体名" system="外部的xml文档"/>
```

以上两种实体在文档中引用的方法和 DTD 中定义实体后引用的方法是一样的，即在实体名称的前面加上“&”，在后面加上“;”。下面举几个例子来说明：

```
<textEntity name="诗">
  墙里秋千墙外道，墙外行人墙内佳人笑，笑渐不闻声渐消，多情却被无情恼。
</textEntity>
<externalEntity name="Chapter135" system="135_1.xml"/>
```

以上定义的两个实体参数在 XML 文档中就可以简单的用&诗;和&Chapter15;来引用。

非解释实体的使用是为了让用户可以在 XML 文档中引用二进制文件，它的一般形式为：


```
<unsparsedEntity name="图像说明" system=http://10.12.41.6/135_1.bmp notation="BMP"/>
```

非解释性实体的引用与前两个实体的引用不同，下面的代码演示了如何引用刚才定义的非解释实体：

```
<picture location="图像说明"/>
```

记法声明（NOTATION）

Schema 中的记法声明和 DTD 中的相似，在 Schema 中，NOTATION 的定义方法如下：

```
<notation name="BMP" system="acdsee.exe">
```

3.2.5 注释（description）

有时候需要在 Schema 中添加注解，用于说明 Schema 中定义的内容，这时我们需要用到 description。Description 可以当作一个标签来使用，标签中的内容就是注解的内容，使用起来很简单，如：

```
<ElementType name="商品">
```

```
  <description>
```

```
    这是注释的一个简单例子
```

```
  </description>
```

```
  <element type="名称">
```

```
</ElementType>
```

3.2.6 具体实例

按照以上的思想，把本部分第一个 XML 实例的文档类型定义用 XML Schema 给出如下：

程序清单 3.7：

```
<?xml version="1.0" encoding="gb2312"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schema-microsoft-com:datatypes">
<ElementType name="销售商品" content="mixed"/>
<ElementType name="商品" content="elonly">
  <element type="编号"/>
  <element type="生产商"/>
  <element type="等级"/>
  <element type="定价"/>
</ElementType>
<ElementType name="编号" content="textonly"/>
<ElementType name="生产商" content="textonly"/>
<ElementType name="等级" content="textonly"/>
<ElementType name="定价" dt:type="fixed.14.4">
```

```

        <attribute type="货币单位"/>
    </ElementType>
    <AttributeType name="货币单位" dt:type="enumetation" dt:value="人民币 港币 美元"
required="true"/>
</Schema>

```

下面再举一个订单的例子，大家可以比较一下两个程序，源文件如下：

程序清单 3.8:

```

<?xml version = "1.0" encoding="GB2312"?>
<订货信息 xmlns="137_3.xml">
  <商品>
    <编号>239</编号>
    <名称>教科书</名称>
    <单价>24.00</单价>
    <数量 单位="本">120</数量>
  </商品>
  <订货人信息>
    <姓名>张正</姓名>
    <详细地址>
      <街道>浙大路20#</街道>
      <城市>杭州</城市>
      <省>浙江省</省>
      <国家>中国</国家>
      <邮政编码>310027</邮政编码>
    </详细地址>
  </订货人信息>
  <备注 付款发式="现金" 订货日期="2000-12-07">
    请赶快发货，等着用！
  </备注>
  <商品>
    <编号>368</编号>
    <名称>家用电脑</名称>
    <单价>6700.00</单价>
    <数量 单位="台">1</数量>
  </商品>
  <订货人信息>
    <姓名>翁翔</姓名>
    <详细地址>
      <街道>南京路路55#</街道>
      <城市>上海</城市>
      <直辖市>上海市</直辖市>
      <国家>中国</国家>
      <邮政编码>210002</邮政编码>
    </详细地址>
  </订货人信息>
</订货信息>

```

```

    </详细地址>
  </订货人信息>
  <备注 付款发式="信用卡" 订货日期="2000-12-08">
    请在15天内寄到!
  </备注>
</订货信息>

```

它的 xml schema 文件如下所示:

程序清单 3.9:

```

<?xml version = "1.0" encoding="GB2312"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schema-microsoft-com:datatypes">
  <ElementType name="订货信息" content="mixed"/>
  <ElementType name="商品" content="elonly" minOccurs="1" maxOccurs="*">
    <element type="编号"/>
    <element type="名称"/>
    <element type="单价"/>
    <element type="数量"/>
  </ElementType>
  <ElementType name="编号" dt:type="ID"/>
  <ElementType name="名称"/>
  <ElementType name="单价" dt:type="fixed.14.4"/>
  <ElementType name="数量" dt:type="number">
    <attribute type="单位"/>
  </ElementType>
  <AttributeType name="单位" required="true"/>
  <ElementType name="订货人信息" content="elonly">
    <element type="姓名"/>
    <element type="详细地址"/>
  </ElementType>
  <ElementType name="姓名"/>
  <ElementType name="详细地址" content="elonly">
    <element type="街道"/>
    <element type="城市"/>
  <group order="one">
    <element type="省"/>
    <element type="直辖市"/>
  </group>
  <element type="国家"/>
  <element type="邮政编码"/>
</ElementType>
<ElementType name="街道"/>
<ElementType name="城市"/>

```

```
<ElementType name="省"/>
<ElementType name="直辖市"/>
<ElementType name="国家"/>
<ElementType name="邮政编码" dt:type="ficed.14.4"/>
<ElementType name="备注" content="mixed">
  <attribute type="付款方式" default="现金"/>
  <attribute type="订货日期"/>
</ElementType>
<AttributeType name="付款方式" dt:type="enumeration" dt:value="现金 支票 信用卡"
required="true" />
<AttributeType name="订货日期" dt:type="date" required="true"/>
</Schema>
```

3.3 小 结

本章在第 2 章介绍 DTD 模式的基础上，主要介绍了 XML 的 Schema 机制，而且介绍了它的语法，同时对 XML-DR 也作了不少的介绍。

事实上通过比较，用户可以选择自己的验证方式，也可以对这几种方式进行不同程度的改进，总之最大限度地满足需要。

下一章将介绍 XML 机制中十分重要的名域机制。

第 4 章 名域机制

本章导读

前面两章对 XML 的几种模式进行了详细的介绍，现在就来看一下名域机制。在 XML 机制中，名域（namespace，也称为名字空间、名称空间）是一个很重要的概念，如果没有名域机制，那么 XML 的应用范围将受到很大的限制，很难想象 XML 的应用像现在这样广泛。在 XML1.0 的标准中找不到名域的规定，名域的规定是通过后来的一份单独标准来增补的，这份标准叫做“Namespaces in XML”。

本章分三节来介绍名域，前一节介绍使用名域的原因以及一些基本概念，后两节分别介绍名域的语法和应用以及 Schema 中的名域问题。

4.1 名域

4.1.1 使用名域的原因

名域在 XML 中的应用是很普遍的。简单地说，使用名域的目的就是为了使用其他 XML 文档中的词汇表。事实上我们在写一个项目或计划的时候，往往会分解问题，一个程序结构语言按照这种意图提供了模块、类、组件、包、函数等。设计词汇表往往类同于编写程序，你往往要将一个大的问题分解成不同的词汇表。而且分解词汇表的意义也是很容易理解的，DTD 本身就是难写难懂的，一个庞大的 DTD 可能导致的结果就是可读性明显的降低，而且很难维护和更新。这样看来分解 DTD 是我们编写项目必须要使用的一个手段了。但是当我们分解了 DTD 之后，新的问题就出现了。也就是各个分解的 DTD 文件又如何有效的整合起来工作呢？如何将定义分解的词汇表整合起来，这就是我们要真正解决的问题。

重用

要有效的整合各个分解的 DTD，我们可以利用的方法是重用。也就是我们在实现编写项目前要商量好各种词汇如何定义，然后每个人使用一部分词汇。这当然是一件极其复杂的事，而且往往因为词汇的扩充使得 DTD 文档明显的复杂化。

在描述真实世界概念的时候，我们将要不断的发现存在的普遍结构。因为任何复杂的单元都是从简单的元素开始的，一个简单的普遍的结构往往能够长时间的存在并且使用。当你正努力为一个公司做计划的时候，你可能会发现有其他的 DTD 存在，实际上，借鉴现有的 DTD 往往使你的工作变得更加简单，而忽视它们会令每个人的工作很难办。

如果你正在编写一个应用程序并和其他的合作伙伴的程序进行连接，除了重用现有的概念之外，没有别的选择。在使用中的 DTD 已经形成了一种普遍并且易理解的通用形式。这是每个编程人员都应该借鉴的。

但是除了重用外，以往的 DTD 往往并不能满足你现在的要求，唯一的办法就是扩充，而扩充的词汇需要自己定义，而实际上这也是非常正常的，每个人都会扩充自己的词汇，

然后劝说别人去适应他自己的词汇。但是每个人的词汇的不同往往会引起一些纷争和误解，导致整个系统不能正常的工作，或者说每个人形成了不同的词汇后，他很难再改变自己去适应别人的词汇（除非对方的词汇有非常明显的优势）。这势必需要我们想办法来解决。重用只是一个推荐的方法，以尽可能的减少词汇混乱。但在解决实际问题时，还需要其他的方法。

名称冲突

我们谈到的实际问题就是名称冲突。事实上在把各个分解的 DTD 整合起来的时候，不可避免的会导致不同的文档使用相同的词汇，这是非常普遍的。可以举个例子来说明相同词汇引起的混乱情况。

假设某一个商店的各种销售商品存放在一个 XML 文档中，里面介绍了某个阶段商店商品的销售情况及其业务员的信息。其 XML 文档如下：

程序清单 4.1:

```
<商品信息>
<商品>
  <基本信息>
    <代号>...</代号>
    <种类>...</种类>
    <品牌>...</品牌>
    <单价 单位="元">...</单价>
  </基本信息>
  <销售情况>
    <季度一>...</季度一>
    <季度二>...</季度二>
    <季度三>...</季度三>
    <季度四>...</季度四>
  </销售情况>
  <数量>
    <库存量>...</库存量>
    <预订量>...</预订量>
  </数量>
  <进货单位>...</进货单位>
  <业务员>
    <姓名>...</姓名>
    <电话>
      <公司电话>...</公司电话>
      <住宅电话>...</住宅电话>
      <手机>...</手机>
    </电话>
    <电子信箱>...</电子信箱>
    <传真>...</传真>
```



```

    </业务员>
  </商品>
  ....
</商品信息>

```

为了某种需要，我们希望经常和该商品进货单位的业务员联系，而进货单位的业务员信息也保存在一个 XML 文档中：

程序清单 4.2:

```

<业务员名单>
  <业务员>    <!--业务员1-->
    <姓名>...</姓名>
    <部门>...</部门>
    <职位>...</职位>
    <电话>
      <公司电话>...</公司电话>
      <住宅电话>...</住宅电话>
      <手机>...</手机>
    </电话>
    <电子信箱>...</电子信箱>
    <传真>...</传真>
  </业务员>
  ...
</业务员名单>

```

需要把上面的几个相关的元素连接起来。比如要和进货单位的业务员联系，需要在“商品信息”中增加一个进货单位相应的“联系人”的信息，以便在需要进货时能够很容易的找到他。我们把两个 XML 文档中的部分信息结合起来，修改后的文档将是下面的形式：

程序清单 4.3:

```

<商品信息>
<商品>
  <基本信息>
    <代号>...</代号>
    <种类>...</种类>
    <品牌>...</品牌>
    <单价 单位="元">...</单价>
  </基本信息>
  <销售情况>
    <季度一>...</季度一>
    <季度二>...</季度二>
    <季度三>...</季度三>
    <季度四>...</季度四>
  </销售情况>

```

```

<数量>
  <库存量>...</库存量>
  <预定量>...</预定量>
</数量>
<进货单位>...</进货单位>
<业务员>
  <姓名>...</姓名>
  <电话>
    <公司电话>...</公司电话>
    <住宅电话>...</住宅电话>
    <手机>...</手机>
  </电话>
  <电子信箱>...</电子信箱>
  <传真>...</传真>
</业务员>
<联系人>
  <姓名>...</姓名>
  <部门>...</部门>
  <职位>...</职位>
  <电话>
    <公司电话>...</公司电话>
    <住宅电话>...</住宅电话>
    <手机>...</手机>
  </电话>
  <电子信箱>...</电子信箱>
  <传真>...</传真>
</联系人>
</商品>
....
</商品信息>

```

由于这两套信息中同时包含了诸如“电子信箱”、“电话”、“姓名”等这样的元素，明显造成了混乱。虽然对人来说，通过上下文将来自两个 XML 文档的信息分清，但是对于计算机程序来说，这是一件不容易的事。因此，我们有必要设计一种简单明了的方法，使计算机能很容易的处理这来自两个 XML 文档中的相同元素。即使没有任何名称上的冲突，我们仍然有必要设计这样一种方法让计算机能知道哪些元素是来自第一个 XML 文档的，哪些元素是来自另一个 XML 文档的，以便使用对应的 XML 文档的 DTD 来确认文件结构的有效性。

解决方法

一个可行的方法是改变我们所知道的所有“联系人”的信息元素，我们可以在“联系人”的子元素前面都加上“进货单位”以示区别。这虽然可行，但是假如我们改动了元素

名后还有必要改变 DTD 的结构，而且在现在的网络世界中，并非所有的 DTD 都是自己设计的，假如我们使用了别人的 DTD，那么修改元素名的方法变得也不现实了。

名域的概念正是在这样的背景下提出来的。它的概念非常直接，如果每套 XML 词汇都有一个独一无二的标志来代表的话，并且在使用的时候这个标志和词汇中的元素、属性名连在一起，那么就不会和其它词汇中的元素造成混乱了。这是因为，每个词汇中的元素、属性都已经事先被该词汇的独特标志码给限定了。其实这个解决办法和前面提到的方法有共同的地方，那就是在含糊的元素或属性前加一个限定，使这个词汇中的元素或属性名独一无二，避免了和其他词汇相冲突。只不过名域的做法是让编写 XML 文件的程序员为文件中所用到的语汇自行指定标志码。

一个独特的标志码代表一套词汇，各词汇中的名称因而能各得其所，有它们自己的活动领域，这就是名域这个名称的由来。

4.1.2 名域空间的表示

上一节已经讲了 XML 中使用的名域标志和词汇中的元素、属性名的联合来表示该词汇表中的内容，这样势必要求标志码是独一无二的。那么究竟它是怎么表示的呢？

XML 机制采用了一种聪明、简便的方法来指定名字空间即直接用网址代替名称空间，由于一般的域名（DNS）都是独一无二的，这样就不用担心 XML 文档会发生混乱了。

利用名域空间，可以把上面的例子改变一下：

程序清单 4.4:

```
<k:商品信息 xmlns:k="http://10.12.41.6/shop.dtd"
  xmlns:m="http://10.12.48.8/company.dtd">
<k:商品>
  <k:基本信息>
    <k:代号>...</k:代号>
    <k:种类>...</k:种类>
    <k:品牌>...</k:品牌>
    <k:单价 单位="元">...</k:单价>
  </k:基本信息>
  <k:销售情况>
    <k:季度一>...</k:季度一>
    <k:季度二>...</k:季度二>
    <k:季度三>...</k:季度三>
    <k:季度四>...</k:季度四>
  </k:销售情况>
  <k:数量>
    <k:库存量>...</k:库存量>
    <k:预定量>...</k:预定量>
  </k:数量>
  <k:进货单位>...</k:进货单位>
  <k:业务员>
    <k:姓名></k:姓名>
```

```
<k:电话>
  <k:公司电话>...</k:公司电话>
  <k:住宅电话>...</k:住宅电话>
  <k:手机>...</k:手机>
</k:电话>
<k:电子信箱>...</k:电子信箱>
<k:传真>...</k:传真>
</k:业务员>
<k:联系人>
  <m:姓名>...</m:姓名>
  <m:部门>...</m:部门>
  <m:职位>...</m:职位>
  <m:电话>
    <m:公司电话>...</m:公司电话>
    <m:住宅电话>...</m:住宅电话>
    <m:手机>...</m:手机>
  </m:电话>
  <m:电子信箱>...</m:电子信箱>
  <m:传真>...</m:传真>
</k:联系人>
</k:商品>
....
</k:商品信息>
```

从上面的例子可以看出，名域是通过“独特标志码+名称”的方式，使每个 XML 词汇中的元素、属性名都能有它们自己的领域，而不会和其他词汇中的名称发生冲突。也就是说在名称前面加上特殊的标志码，这个名称就变成了独一无二的了。

虽然 XML 选用 URI 来做独特的标志码，但是一般的 URI 都很长，如果直接拿来放在元素和属性名称前，不但书写不便，而且阅读也不方便。此外，URI 里面常常包含一些 XML 语法规则不准用作元素、属性名的字符，因此通常使用一个简短的代号来代替 URI。这个简短的代号在名域的标准中称为“前置字串”（Namespace prefix），一般称为前缀，它由编辑 XML 文档的人自由决定的，例如上例中的“k”和“m”。前缀只能包含 XML 标准中规定的允许用作元素和属性名的字符组成，这包括了英文字母和所有收录在 Unicode 中的汉字。于是名字空间就有了它独特的形式，一般的在一个使用名域的 XML 文档中，总需要包括两部分，一是名域的宣告，另一个就是名域的使用。名域的声明如下：

```
<元素 xmlns:前缀="URI">
```

使用时可以这样来使用：

```
<前缀:元素>
```

它代表的意思就是<URI:元素>，即该 URI 下词汇中定义的元素。

其实，在命名空间声明中，等号右边的命名空间名虽说要求是一个 URI，但其目的并不是要直接获取一个 Schema 或 DTD 文件，而在于标识特定的命名空间。也就是说，xml 语法分析器看到一个命名空间声明后，就把等号左边的命名空间前缀和右边的命名空间名绑定在一起，对于后面使用了该前缀的合法名称，都看作是这个命名空间中的。但是，等到语法分析器进行有效性检测时，它不是把这个命名空间映射到 URI 所指的 Schema 文件或 DTD 文件，而是去找所有在 DOCTYPE 中声明的内部和外部的 DTD 或 Schema，看哪一个所定义的合法元素、属性名与文件中用到的合法元素、属性名相同。

在前面的例子 4.4 中，“k”是给文档中的“商品信息”这个 XML 词汇指定的名域标识码，而“m”是给“业务员”指定的标识码。为了书写方便，以“k”前缀代替商品信息的标志网址（xmlns:k="http://10.12.41.6/shop.dtd"），用“m”代替进货公司“业务员”的标志网址（xmlns:m="http://10.12.48.8/company.dtd"）。可以看出，一个元素里面可以放置多个 xmlns 的属性声明。

注意，前缀名不能使用“xml”这三个字母开头的字符串，因为在 XML 中这个字符串是保留作特殊用途的。还要指出的是，URI 全称为 Uniform Resource Identifier（统一资源标志码），它不同于一般所讲的网址（URL，Uniform Resource Locator），也叫统一资源定位码。相比较而言，URI 比较广义，它包含所有的 URL 和 URN（Uniform Resource Name，统一资源名）。

4.2 名域的范畴和应用

4.2.1 名域的范畴

在 XML 中，名域涉及到一个范畴的概念，范畴即名域的覆盖范围，它指的是哪些元素和属性在该名域里面的，哪些又不在的问题。从前一节的例子看出，我们在文档的前面都使用了名域，那么很明显，在元素前面加“k”的元素就是属于商店的，前面加“m”的元素是属于进货公司的。但是如果我们在文档中使用预设名域，情况就不同了。

预设和限定

在 XML 中预设名域是指没有指定前置字串的名域，所有未加上前置字串和冒号的元素、属性名都是在预设的名域范围中的。预设名域和一般的名域一样，可以通过声明将预设的名域标识出来。预设名域的宣告和前面介绍的宣告一般名域的方法是一样的，只不过少了冒号和前置字串部分，形式如下面所示：

<元素 xmlns="URI">

对于预设的名域，它的覆盖区域是从开始声明的元素开始，到该元素结束中间的所有内容，当然还要除去有前缀的元素和被包含在另外一个预设名域中的元素。如下的形式：

```
<元素 xmlns="URI">
```

```
...
```

```
</元素>
```

假如该元素中没有其它的预设名域，那么该元素中所有没有前缀的元素将全适用该名域的词汇规则。现把前一节的例子改变一下：

程序清单 4.5:

```
<商品信息 xmlns=http://10.12.41.6/shop.dtd xmlns:m="http://10.12.48.8/company.dtd">
```

```
<商品>
```

```
<基本信息>
```

```
<代号>...</代号>
```

```
<种类>...</种类>
```

```
<品牌>...</品牌>
```

```
<单价 单位="元">...</单价>
```

```
</基本信息>
```

```
<销售情况>
```

```
<季度一>...</季度一>
```

```
<季度二>...</季度二>
```

```
<季度三>...</季度三>
```

```
<季度四>...</季度四>
```

```
</销售情况>
```

```
<数量>
```

```
<库存量>...</库存量>
```

```
<预定量>...</预定量>
```

```
</数量>
```

```
<进货单位>...</进货单位>
```

```
<业务员>
```

```
<姓名></姓名>
```

```
<电话>
```

```
<公司电话>...</公司电话>
```

```
<住宅电话>...</住宅电话>
```

```
<手机>...</手机>
```

```
</电话>
```

```
<电子信箱>...</电子信箱>
```

```
<传真>...</传真>
```

```
</业务员>
```

```
<联系人>
```

```
<m:姓名>...</m:姓名>
```

```
<m:部门>...</m:部门>
```

```
<m:职位>...</m:职位>
```

```
<m:电话>
```

```
<m:公司电话>...</m:公司电话>
```

```
<m:住宅电话>...</m:住宅电话>
```



```

    <m:手机>...</m:手机>
    </m:电话>
    <m:电子信箱>...</m:电子信箱>
    <m:传真>...</m:传真>
  </联系人>
</商品>
....
</商品信息>

```

从这个例子可以看出，“商品信息”这套词汇被指定为预设的名域，又由于商品信息为根元素，并且里面没有包含其它预设名域，因此任何没有前缀和冒号的元素名都会被认为属于这个域的。

虽然文档里面有很多默认的名域，但是在其中有一段以“m”标记出来的段落，该部分我们称之为限定。限定的出现基于设计，当不能使用名域区分必要的各种元素，需要一个更精细的划分尺度时，我们就需要使用限定。在上面的例子中使用到前缀“m”的内容都属于限定的内容。

优先级

还有一种情况是在 XML 文档中包含了两个预设名域。它们的范畴关系就涉及到优先级的问题。在 XML 中内层的名域要高于外层，可参看下面的形式：

```

<元素1 xmlns="URI1">
  ...
  <元素2 xmlns="URI2">
    ...
  </元素2>
  ...
</元素1>

```

在这个例子中，元素 2 中包含的内容将会适用 URI2 标志码定义的词汇，而“元素 1”中的除“元素 2”外的其他元素将适用 URI1 标识码定义的词汇。再看下面的例子：

程序清单 4.6:

```

<商品信息 xmlns="http://10.12.41.6/shop.dtd">
  <商品>
    <基本信息>
      <代号>...</代号>
      <种类>...</种类>
      <品牌>...</品牌>
      <单价 单位="元">...</单价>
    </基本信息>
    <销售情况>
      <季度一>...</季度一>
    </销售情况>
  </商品>
</商品信息>

```

```

        <季度二>...</季度二>
        <季度三>...</季度三>
        <季度四>...</季度四>
    </销售情况>
    <数量>
        <库存量>...</库存量>
        <预定量>...</预定量>
    </数量>
    <进货单位>...</进货单位>
    <业务员>
        <姓名></姓名>
        <电话>
            <公司电话>...</公司电话>
            <住宅电话>...</住宅电话>
            <手机>...</手机>
        </电话>
        <电子信箱>...</电子信箱>
        <传真>...</传真>
    </业务员>
<!--进入名域2，改用company.dtd-->
    <联系人 xmlns:m="http://10.12.48.8/company.dtd">
        <姓名>...</姓名>
        <部门>...</部门>
        <职位>...</职位>
        <电话>
            <公司电话>...</公司电话>
            <住宅电话>...</住宅电话>
            <手机>...</手机>
        </电话>
        <电子信箱>...</电子信箱>
        <传真>...</传真>
    </联系人> <!--离开名域2-->
</商品>
....
</商品信息>

```

在这个例子中，就不用担心 XML 中的内容会发生混乱，因为在预设名域的约定下，系统会分清那个是来自“shop.dtd”，那个是来自“company.dtd”。

4.2.2 实际应用

利用 XML 的名域机制，我们可以使用任何在 HTML 定义过的标签，而且显示的时候也和 HTML 中的一样。下面的例子就通过名域机制引入 HTML 的标签，当然使用 HTML 标签的前提是通过名字空间的声明。

通过 HTML 方法可以在 XML 文档中加入一些 XML 中没有定义过的标签。比如想在 XML 中加入表单, 就可以通过 HTML 的方法向文档中加入诸如<form>、<input>等标签。因为目前在 XML 中还没有一个通行的“FormML”标准, 所以在 XML 中加入表单还是很有用的。另外, 也可以通过 HTML 方法在 XML 文档中加入 HTML 中的 Maquee, Layers, 脚本等内容。

当然也可以通过 HTML 的<A>标签来为 XML 中的内容加入超链接, 虽然 XML 有自己的连接方式 (XLink/XPointer)。可参看下面实例的源代码:

程序清单 4.7:

```
<?xml version = "1.0" encoding="GB2312"?>
<?xml-stylesheet type="text/css" href="213_2.css"?>
<book xmlns:html="http://www.w3.org/TR/REC-html40">
<html:h2 align="center">图像处理</html:h2>
<html:hr width="600" align="center"/>
  <author> 何小艇主编 </author>
  <preface>本书是面向21世纪课程教材, 同时也是浙江省高等教育重点建设教材之一。本书以电子系统设计方法为主线, 以数字系统、模拟系统、智能系统(以单片机为核心的数模混合系统)三大系统的设计原理、方法并结合实例为主题展开。全书特点是理论与实际相结合, 并注意了实用性。可供高等学校工科电子工程类、信息工程类、电子技术类、电气工程类、自动控制类以及机电工程专业作为本科生教材, 也可供有关工程技术人员作为学习电子系统设计的参考书, 同时可作为全国大学生电子设计竞赛的培训教材及参考书。
  </preface>
  <html:hr width="600" align="center"/>
  <connect>
    <html:p>
      <html:div align="center"><html:a href="mailto:hexiaot@sina.com">和作者联系
    </html:a></html:div>
    </html:p>
    <html:p><html:div align="center">
      <html:form name="form1" method="get" action="111.html">
        <html:input type="submit" name="Submit" value="其他书目介绍"/>
      </html:form>
    </html:div></html:p>
  </connect>
</book>
```

它的 CSS 文档代码如下:

程序清单 4.8:

```
book{
  display:block;
  color:#0000a0; }
author{
```

XML 高级应用

```
display:block;
font-family:黑体;
font-size:120%;
font-weight:bold;
text-align:center;}
preface{
display:block;
margin-left:10%;
margin-right:10%;
font-family:宋体;
font-size:100%;
line-height:15pt;
text-indent:2em;}
connect{
display:block;
color:#99ccff; }
```

上例的显示如图 4-1 所示。

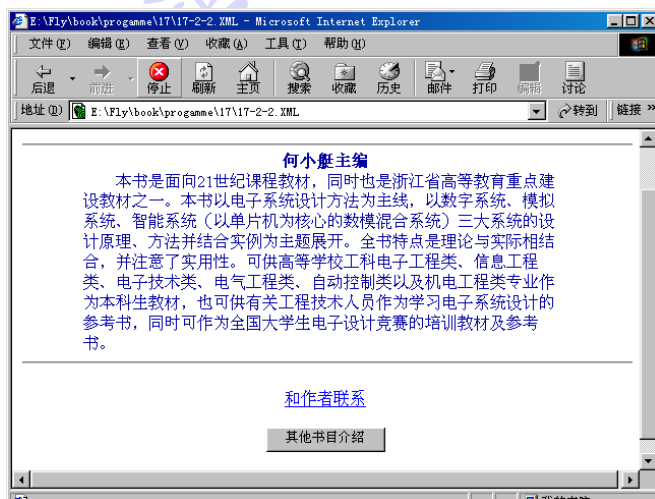


图 4-1 XML 使用标签实例

下面看另外一个例子，该例使用了 ActiveX 和 Javascript:

程序清单 4.9:

```
<?xml version="1.0" encoding="gb2312"?>
<?xml-stylesheet type="text/css" ?>
<link xmlns:html="http://www.w3.org/TR/REC-html40">
<html:head>
<html:script language="JavaScript">
<!--//
var version = "other"
```

```

browserName = navigator.appName;
browserVer = parseInt(navigator.appVersion);
if (browserName == "Netscape" && browserVer >= 3) version = "n3";
else if (browserName == "Netscape" && browserVer < 3) version = "n2";
else if (browserName == "Microsoft Internet Explorer" && browserVer >= 4) version = "e4";
else if (browserName == "Microsoft Internet Explorer" && browserVer < 4) version = "e3";
function marquee1()
{
    if (version == "e4")
    {
        document.write("<marquee behavior=scroll direction=up width=368 height=230
scrollamount=1 scrolldelay=60 onmouseover='this.stop()' onmouseout='this.start()' align='center'>")
    }
}
function marquee2()
{
    if (version == "e4")
    {
        document.write("</marquee>")
    }
}
function eng()
{
    CHANGE.innerHTML="<font size='6' color='#3333CC' face='华文彩云, 楷体_GB2312, 宋体
'>XML STUDY BOOK</font>";
}
function chn()
{
    CHANGE.innerHTML="<font size='6' color='#3333CC' face='华文彩云, 楷体_GB2312, 宋体
'>XML学习教程</font>";
}

//-->
</html:script>
</html:head>
<html:body bgcolor="#FFFFFF">
<html:table border="1" cellpadding="0" width="370" height="43" align="center"
bordercolor="#CC6633" cellpadding="0" cellspacing="0">
<html:td>
    <html:div onmouseover="eng()" onmouseout="chn()" align="center"
ID="CHANGE"><html:font size="6" color="#3333CC" face="华文彩云, 楷体_GB2312, 宋体">XML学习
教程</html:font></html:div>
</html:td>

```

```

</html:table>
<html:br>
<html:table border="1" cellpadding="0" width="370" height="43" align="center" cellspacing="0">
  <html:tr>
    <html:td bgcolor="99ccff">
      <html:div align="center"><html:font color="#3333FF"><html:b><html:font face="华文行楷"
color="#CC6600">本书目录简介</html:font></html:b></html:font></html:div>
    </html:td></html:tr>
  <html:tr>
    <html:td>
      <html:div align="left">
        <html:script language="JavaScript">marquee1();</html:script>
      </html:div>
      <html:p align="left"><html:img src='image/blue_dot.gif' width='10' height='10'/>第一章
        XML简介<html:br>
          本章主要向读者介绍XML的产生的一些背景和其他的一些XML规范<html:br>
          <html:br>
          <html:img src='image/blue_dot.gif' width='10' height='10'/>第二章 XML的语法简介
        <html:br>
          本章向大家介绍的是XML的语法规则，包括一些常用的XML文件格式以及涉及到的一些语法准则。<html:br>
          <html:br>
          <html:img src='image/blue_dot.gif' width='10' height='10'/>第三章 XML的语法检查
        <html:br>
          本章大家会了解XML文档的有效性准则，它是通过DTD文件和SHEMA文件实现的。
        <html:br>
          <html:br>
          <html:img src='image/blue_dot.gif' width='10' height='10'/>第四章 表现XML<html:br>
          我们可以通过CSS或XSL来表现XML文档，通过本章的学习，相信大家会很清楚的了解XML的表现，同时也会对XML的前景看好。<html:br>
          <html:br>
          <html:img src='image/blue_dot.gif' width='10' height='10'/>第五章 XML的链结语言
        <html:br>
          本章给大家介绍的将是XML的链接语言XLink和XPointer，XML的链接语言提供了比HTML更强大的功能。<html:br>
          <html:br>
          <html:img src='image/blue_dot.gif' width='10' height='10'/>第六章 DOM<html:br>
          本章向大家介绍的是XML的DOM规范，使用DOM我们可以很容易的控制XML文档。
        </html:p>
        <html:p align="left">。 </html:p>
        <html:p align="left">。 </html:p>
        <html:p align="left">。 <html:br>

```

```

<html:br/>
<html:script language="JavaScript">marquee2(); </html:script>
</html:p>
</html:td></html:tr></html:table>
</html:body>
</link>

```

文档的显示效果如图 4-2 所示:

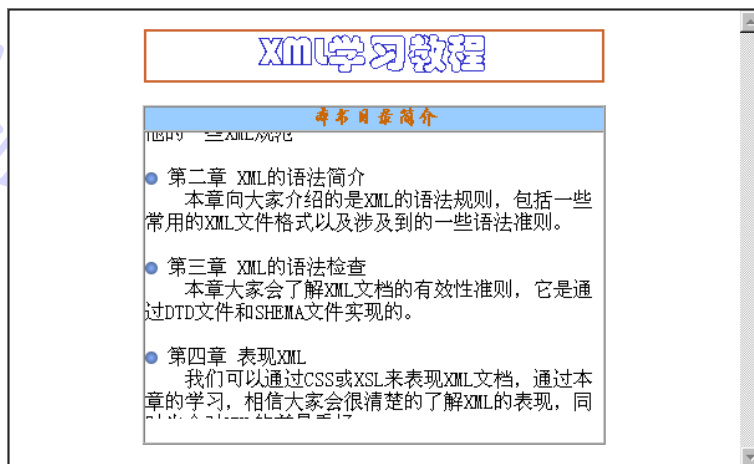


图 4-2 XML 使用 ActiveX 实例

4.3 Schema 中的名域

关于 Schema 中的名域问题,前面第 2 章在介绍 Schema 机制的时候曾经提到过,这里有必要再说明一下。因为 Schema 文件本身就是一个 XML 文件,因此能够通过包含别的 Schema 文件来把两个文件中的词汇定义联合起来。Schema 中名字空间的用法跟一般的 XML 文件中的用法是一样的。下面先给出两个 DTD 的例子,然后将它们编写成 XML Schema 的形式,先看两个 DTD 文档的源文件。

商店的 Schema 文件, shop.xml。

程序清单 4.10:

```

<?xml version = "1.0" encoding="GB2312"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schema-microsoft-com:datatypes">
  <ElementType name="商品信息" content="elonly">
    <element type="商品"/>
  </ElementType>
  <ElementType name="商品" content="elonly" minOccurs="1" maxOccurs="*">
    <element type="基本信息"/>
    <element type="销售情况"/>
    <element type="数量"/>
  </ElementType>

```



```
<element type="进货单位"/>
<element type="业务员"/>
</ElementType>
<ElementType name="基本信息" content="elonly" minOccurs="1" maxOccurs="*">
  <element type="代号"/>
  <element type="种类"/>
  <element type="单价"/>
  <element type="品牌"/>
</ElementType>
<ElementType name="代号" dt:type="ID"/>
<ElementType name="种类"/>
<ElementType name="品牌"/>
<ElementType name="单价" dt:type="fixed.14.4">
  <attribute type="单位"/>
</ElementType>
<Attribute type="单位" required="true"/>
<ElementType name="销售情况" content="elonly">
  <element type="一季度"/>
  <element type="二季度"/>
  <element type="三季度"/>
  <element type="四季度"/>
</ElementType>
<ElementType name="一季度" dt:type="number"/>
<ElementType name="二季度" dt:type="number"/>
<ElementType name="三季度" dt:type="number"/>
<ElementType name="四季度" dt:type="number"/>
<ElementType name="数量" content="elonly">
  <element type="库存量"/>
  <element type="预定量"/>
</ElementType>
<ElementType name="库存量" dt:type="number"/>
<ElementType name="预定量" dt:type="number"/>
<ElementType name="进货单位" content="textonly"/>
<ElementType name="业务员" content="elonly">
  <element type="姓名"/>
  <element type="电话"/>
  <element type="电子信箱"/>
  <element type="传真"/>
</ElementType>
<ElementType name="姓名" content="textonly"/>
<ElementType name="电话" content="elonly">
  <element type="公司电话"/>
  <element type="住宅电话"/>
```

```

    <element type="手机"/>
</ElementType>
<ElementType name="电子信箱" content="textonly"/>
<ElementType name="传真" dt:type="fixed.14.4"/>
<ElementType name="公司电话" dt:type="fixed.14.4"/>
<ElementType name="住宅电话" dt:type="fixed.14.4"/>
<ElementType name="手机" dt:type="fixed.14.4"/>
</Schema>

```

公司的 Schema 文件, company.xml:

程序清单 4.11:

```

<?xml version = "1.0" encoding="GB2312"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schema-microsoft-com:datatypes">
  <ElementType name="业务员名单" content="elonly">
    <element type="业务员"/>
  </ElementType>
  <ElementType name="业务员" content="elonly">
    <element type="姓名"/>
    <element type="部门"/>
    <element type="职位"/>
    <element type="电话"/>
    <element type="电子信箱"/>
    <element type="传真"/>
  </ElementType>
  <ElementType name="姓名" content="textonly"/>
  <ElementType name="部门" content="textonly"/>
  <ElementType name="职位" content="textonly"/>
  <ElementType name="电话" content="elonly">
    <element type="公司电话"/>
    <element type="住宅电话"/>
    <element type="手机"/>
  </ElementType>
  <ElementType name="电子信箱" content="textonly"/>
  <ElementType name="传真" dt:type="fixed.14.4"/>
  <ElementType name="公司电话" dt:type="fixed.14.4"/>
  <ElementType name="住宅电话" dt:type="fixed.14.4"/>
  <ElementType name="手机" dt:type="fixed.14.4"/>
</Schema>

```

假如使用 Schema, 就不需要在文档中使用两个 DTD 了, 可以利用 Schema 支持名域的方法来把我们需要的联系人的信息元素包含进来, 这样可写成下面的形式:

程序清单 4.12:

```
<?xml version = "1.0" encoding="GB2312"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schema-microsoft-com:datatypes">
  <ElementType name="商品信息" content="elonly">
    <element type="商品"/>
  </ElementType>
  <ElementType name="商品" content="elonly" minOccurs="1" maxOccurs="*">
    <element type="基本信息"/>
    <element type="销售情况"/>
    <element type="数量"/>
    <element type="进货单位"/>
    <element type="业务员"/>
  </ElementType>
  <ElementType name="基本信息" content="elonly" minOccurs="1" maxOccurs="*">
    <element type="代号"/>
    <element type="种类"/>
    <element type="单价"/>
    <element type="品牌"/>
  </ElementType>
  <ElementType name="代号" dt:type="ID"/>
  <ElementType name="种类"/>
  <ElementType name="品牌"/>
  <ElementType name="单价" dt:type="fixed.14.4">
    <attribute type="单位"/>
  </ElementType>
  <AttributeType name="单位" required="true"/>
  <ElementType name="销售情况" content="elonly">
    <element type="一季度"/>
    <element type="二季度"/>
    <element type="三季度"/>
    <element type="四季度"/>
  </ElementType>
  <ElementType name="一季度" dt:type="number"/>
  <ElementType name="二季度" dt:type="number"/>
  <ElementType name="三季度" dt:type="number"/>
  <ElementType name="四季度" dt:type="number"/>
  <ElementType name="数量" content="elonly">
    <element type="库存量"/>
    <element type="预定量"/>
  </ElementType>
  <ElementType name="库存量" dt:type="number"/>
  <ElementType name="预定量" dt:type="number"/>
</Schema>
```

```

<ElementType name="进货单位" content="textonly"/>
<ElementType name="业务员" content="eltonly">
  <element type="姓名"/>
  <element type="电话"/>
  <element type="电子信箱"/>
  <element type="传真"/>
</ElementType>
<ElementType name="姓名" content="textonly"/>
<ElementType name="电话" content="eltonly">
  <element type="公司电话"/>
  <element type="住宅电话"/>
  <element type="手机"/>
</ElementType>
<ElementType name="电子信箱" content="textonly"/>
<ElementType name="传真" dt:type="fixed.14.4"/>
<ElementType name="公司电话" dt:type="fixed.14.4"/>
<ElementType name="住宅电话" dt:type="fixed.14.4"/>
<ElementType name="手机" dt:type="fixed.14.4"/>
<!-- 引用另外一套词汇 -->
<ElementType name="联系人" xmlns:link="company.xml">
  <element type="link:姓名"/>
  <element type="link:电话"/>
  <element type="link:电子信箱"/>
  <element type="link:传真"/>
</ElementType>
</Schema>

```

有了这种方法，就可以在 XML 文档中使用一个外部词汇定义了。

但是要注意，在 Schema 中引用的是其它文件的词汇定义，在说明它的时候我们把元素包含为：

```
<element type="link:姓名"/>
```

这就要求我们在使用这些元素的时候也要像在 Schema 中写的那样，不能省略 link 前缀。如：

程序清单 4.13:

```

<商品信息>
  <商品>
    <基本信息>
      <代号>...</代号>
      <种类>...</种类>
      <品牌>...</品牌>
    
```

```

        <单价 单位="元">...</单价>
    </基本信息>
    <销售情况>
        <季度一>...</季度一>
        <季度二>...</季度二>
        <季度三>...</季度三>
        <季度四>...</季度四>
    </销售情况>
    <数量>
        <库存量>...</库存量>
        <预定量>...</预定量>
    </数量>
    <进货单位>...</进货单位>
    <业务员>
        <姓名>...</姓名>
        <电话>
            <公司电话>...</公司电话>
            <住宅电话>...</住宅电话>
            <手机>...</手机>
        </电话>
        <电子信箱>...</电子信箱>
        <传真>...</传真>
    </业务员>
    <联系人>
        <link:姓名>...</link:姓名>
        <link:部门>...</link:部门>
        <link:职位>...</link:职位>
        <link:电话>
            <link:公司电话>...</link:公司电话>
            <link:住宅电话>...</link:住宅电话>
            <link:手机>...</link:手机>
        </link:电话>
        <link:电子信箱>...</link:电子信箱>
        <link:传真>...</link:传真>
    </联系人>
</商品>
....
</商品信息>

```

4.4 小 结

本章首先讨论了在 XML 文档中使用名域的原因,谈到了词汇表混合时因为多义性词汇引起的词汇表混乱,从而指出了名域的必要性。然后展开讨论,利用实例来说明名域的

各种机制，对域名域的表示和名域的几种表示方式作了很明确的说明，同时对名域的优先级问题也作了实例性的讲解。

本章承接前面一章，因此在本章最后，特意加上了 XML Schema 中的名域机制。希望读者对 Schema 和名域都有更好的了解。同时考虑到程序运行时的显示效果，利用名域机制引用了很多 HTML 的标签。这也正是 XML 的用处之一了。

北京拓源电子书出版社

第二篇 XML 与数据

- ☒ 第 5 章 DOM 进阶
- ☒ 第 6 章 DOM 应用实例
- ☒ 第 7 章 SAX 进阶
- ☒ 第 8 章 XML 与数据

本篇导读：

第 5 章和第 6 章介绍了编程接口 DOM。DOM 全称是 Document Object Model，就是一个文档对象组成的模型，它不光用于 XML，最先是用于 HTML 的。DOM 对 XML 开发者是相当重要的，对于 XML 应用开发来说就是一个对象化的 XML 数据接口。最基本的 XML 开发通常都要使用它。

第 7 章介绍另一编程接口 SAX。SAX 其实就是 Simple Application interface for XML，这个接口规范是 XML 分析器和 XML 处理器提供的较 XML 更底层的接口。它能提供给应用以较大的灵活性。

第 8 章则深入探讨了 XML 的存储、数据的交换以及 XML 和数据库的问题。XML 存储部分主要说明文件系统的局限性以及借助于数据库的必要性和优点。数据的交换主要讲述如何在异质通信之间交换数据。此外，还介绍了如何在复杂的情况下处理 XML 模式与数据库之间的关系。



第 5 章 DOM 进阶

本章导读

本章及第 6 章将深入探讨一个重要的编程接口：DOM 接口。本章先介绍 DOM 的相关知识，第 6 章则给出几个具体的实例。

对于要从事 XML 高级开发的技术人员来说，DOM 是必须掌握的一种技术。DOM 提供了良好的机制来对 XML 文档中的节点进行访问和操作，而且在很大程度上保证了各个平台之间的互操作性。

5.1 XML 文档的加载和遍历

5.1.1 DOM 回顾

首先对 DOM 进行一个简单的回顾。

对于一个网页设计人员来说，文档对象模型一定不会陌生，因为在 HTML 中早就知道了窗口、文档以及历史等对象都被认为是浏览器模型的一部分，但是各种浏览器实现这些对象的方式都是有所不同的，为了创建标准化的方法，W3C 提出了 DOM 规范：它定义了构成 DOM 的不同对象，编程人员可以通过任何编程语言来实现它的具体应用。

DOM 对 XML 开发者来说是相当重要的，对于 XML 应用开发来说就是一个对象化的 XML 数据接口。最基本的 XML 开发通常都要使用它。简单说，DOM 就是一组对象的集合，通过操纵这些对象，程序员能操纵 XML 和 HTML 数据。利用 DOM 中的对象，可以对文档进行读取、遍历、修改、添加以及删除等操作。

总体来说，使用 DOM 有着如下的几点好处：

(1) 能保证正确的语法和格式

由于用 DOM 处理 XML 文档的时候要加载 XML 文档并在内存中生成一棵 DOM 节点树，因此可以避免无结束标记或者是不正确的嵌套等语法错误，这就可以让开发人员把精力集中到主要问题上，而不必顾虑文档的语法和格式问题。

(2) 简化了文档的操作

使用 DOM 对 XML 文档中的节点进行访问和操作比较简单，我们只需要掌握几种常用的接口就可以轻松地进行开发。

(3) 与数据库可以良好的结合转换

由于 DOM 在表示 XML 文档中的各个节点的关系时非常类似于我们常用的关系数据库的处理方法，所以可以轻松地在数据库和 XML 文件之间转换。

5.1.2 节点树

DOM 中的最基本对象应该就是 Node 了。从它又派生出许多类型的 node。所有这些 node 组成一棵文件树，它包含了 XML 或 HTML 文档的几乎全部信息。

在结构化的 XML 文档中,信息是按层次化的树形结构组织的。所以 XML 文档的模型的组织也必然是树形的。由 DOM 创建的节点树是 XML 文件内容的逻辑表示,它显示了文件提供的信息,以及它们之间的关系,而不受 XML 的语法限制。

先来看一个例子。

程序清单 5.1:

```
<?xml version="1.0" encoding="gb2312"?>
<books>
  <book>
    <name>XML基础</name>
    <author>翁宇翔</author>
    <author>芮云</author>
    <author>张建飞</author>
    <price unit="RMB">88.88</price>
  </book>
  <book>
    <name>XML进阶</name>
    <author>翁宇翔</author>
    <author>芮云</author>
    <author>张建飞</author>
    <price unit="RMB">188.88</price>
  </book>
</books>
```

这个文档被加载之后就在内存之中存储了一个 DOM 树,如图 5-1 所示。

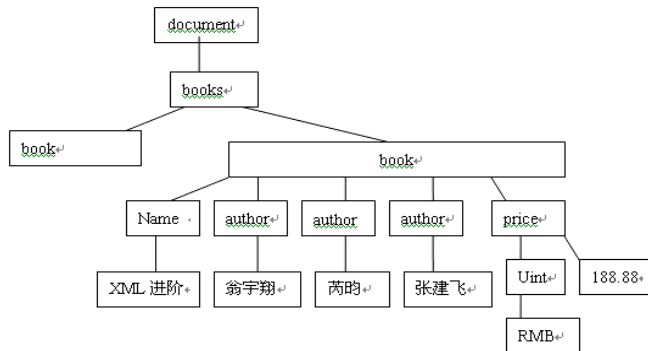


图 5-1 DOM 节点树

从上图看到,在 DOM 中,文档的逻辑结构类似一棵树。文档、文档中的根、元素、元素内容、属性、属性值等都是对象模型的形式表示的。文档对象模型利用对象来把文档模型化,这些模型不仅描述了文档的结构,还定义了模型中的对象的行为。换句话说,在上面给出的例子里,图中的节点不是数据结构,而是对象,对象中包含方法和属性。下面来看一下 DOM 中定义的对 DOM 树的节点进行各种操作的常用对象和方法:

☑ Document 对象:即文档对象,作为树的最高节点,Document 对象是对整个文档

进行操作的入口。

- ☒ **Element 和 Attr 对象：**这些节点对象都是文档某一部分的映射，节点的定级层次恰好反映了文档的结构。如图 5-1 中的 Name 就是 Element 对象，而 unit 则是 Attr 对象。
- ☒ **Text 对象：**作为 Element 和 Attr 对象的子节点，Text 对象表达了元素或属性的文本内容。Text 节点不再包含任何子节点。在上图中 Element 对象 name 的 Text 对象就是“XML 进阶”；Attr 对象 unit 的 Text 对象就是“RMB”。
- ☒ **集合索引：**DOM 提供了几种集合索引方式，可以对节点按指定方式进行遍历。索引参数都是从 0 开始记数的。

DOM 树中的所有节点都是从 Node 对象继承而来的。Node 对象定义了一些最基本的属性和方法，利用这些方法可以实现对树的遍历，同时，根据属性还可以得知节点的名称、取值并判断其类型。事实上 DOM 执行了更进一步的操作，它将文档中的每个项目看作节点——元素，属性，注释，处理命令，甚至构成属性的文本。

5.1.3 文档的创建和加载

在对节点树有了一定的了解之后，下面介绍如何应用 DOM 接口。

首先来看如何创建一个 Document 对象。通过创建 Document 对象，应用程序就得到了对 XML 文档进行操作的入口。我们可以用各种不同的语言创建 Document 对象，在实际应用中开发者可以根据各自的喜好来选择开发语言。

这里介绍一下如何用几种常用的语言来创建 Document 对象。

VB Script:

```
Dim doc
Set doc = CreateObject("Microsoft.XMLDOM")
```

JScript:

```
var doc = new ActiveXObject("Microsoft.XMLDOM")
```

VB:

```
Dim doc
Set doc = CreateObject("Microsoft.XMLDOM")
```

VC:

```
HRESULT hr = CoCreateInstance
(CLSID_DOMDocument, NULL,
CLSCTX_INPROC_SERVER,
IID_IXMLDocument,
(LPVOID*)&m_pXMLDocument);
```

现在我们已经创建了 Document 对象，也就是得到了对文档进行操作的入口，为了把创建的这个文档对象同实际的 XML 文档关联在一起，在微软的 msxml 中，有两种实现方

式。一种是通过 load 方法直接加载 XML 文档，另一种是通过 loadXML 方法加载 XML 文档片断。下面就分别用 VB 和 VC 两种语言来看一下这两种方法的具体应用。

1. 通过 load 方法来加载一个 XML 文档，如下所示：

(1) VB 例子

```
Dim doc
Set doc = CreateObject("Microsoft.XMLDOM ")
doc.async = False
doc.load("books.xml")
```

在这段代码中，CreateObject("microsoft.xmlDOM")用来创建分析器实例；async 属性用来指明是否允许异步下载，当 async 的值为 true 时（缺省值），load 方法不等待 XML 文档下载完毕，就把控制返回给调用进程，在这种情况下，如果想获知文档的下载状态，可以通过 readyState 属性来实现。在上面的例子中，把 async 赋值为 false，表明只有当文档下载完毕，控制才返回给调用进程。load("books.xml")方法告诉分析器加载名字为 books.xml 的 XML 文档。

(2) VC 例子

```
IXMLDOMDocument* pDoc = NULL;
if (SUCCEEDED(CoInitialize(NULL)))
{
    if(SUCCEEDED(CoCreateInstance(CLSID_DOMDocument, NULL,
                                  CLSCTX_INPROC_SERVER,
                                  IID_IXMLDOMDocument,
                                  (void**)&pDoc)))
    {
        CString strFilePath = "c:\\books.xml";
        CComVariant vFile(strFilePath);
        VARIANT_BOOL vBool;
        pDoc->load(vFile,&vBool);
        if(vBool==VARIANT_TRUE)
        {
            AfxMessageBox("load success");
        }
        pDoc->Release();
    }
    CoUninitialize();
}
```

在这段代码中，由于 MSXML 中的 DOM 所有的接口都是 COM 接口，所以先用 CoInitialize(NULL)来初始化 COM 环境，与之对应的最后用 CoUninitialize()来释放所有被该线程占用的资源和与之相关的 dll 库等。CoCreateInstance 创建 COM 接口 pDoc。注意，用完 COM 接口后用 Release()函数释放，这时 COM 对象内的计数器自动减一，当该计数器为零时自动释放 COM 对象，这是一个很好的编程习惯。

HRESULT load(VARIANT xmlSouse, VARIANT_BOOL* isSuccessful)函数用来将 xml 文档加载到内存中,即 DOM 树中。参数 xmlSouse 为 xml 加载文件的路径,参数 isSuccessful 为返回参数,标识是否成功。

2. 通过 loadXML 方法来加载 XML 文档片断,如下所示。

load 方法加载的是一个完整的 XML 文档,loadXML 方法加载的是字符串,这个字符串是符合 XML 语法的 XML 片断,当然,如果字符串的内容是一个完整的 XML 文档的全部内容也是可以的。如下:

```
book = "<book>"
book= book & "<name>XML基础</name>"
book = book & "<title>XML应用</title>"
book= book & "<author>翁宇翔</author>"
book = book & "<author>芮云</author>"
book = book & "<author>张建业</author>"
book = book & "<price unit='RMB'>88.88</price>"
book = book & "</book>"
```

还是来看 VB 和 VC 是如何实现的。

(1) VB 例子

```
Dim doc
Set doc = CreateObject("Microsoft.XMLDOM")
doc.async = False
doc.loadXML(book)
```

函数 loadXML(string strxml)的参数 strxml 为要加载的字符串。

(2) VC 例子

```
CString strxml = book //假设这样将book字符串的内容赋给strxml
IXMLDOMDocument* pDoc = NULL;
if (SUCCEEDED(CoInitialize(NULL)))
{
    if(SUCCEEDED(CoCreateInstance(CLSID_DOMDocument, NULL,
                                  CLSCTX_INPROC_SERVER,
                                  IID_IXMLDOMDocument,
                                  (void**)&pDoc)))
    {
        VARIANT_BOOL isSuccessful;
        pDoc->loadXML(strxml.AllocSysString(),&isSuccessful);
        if(isSuccessful==VARIANT_TRUE)
        {
            AfxMessageBox("loadxml success");
        }
    }
}
```

```

        pDoc->Release();
    }
    CoUninitialize();
}

```

函数 HRESULT loadXML(BSTR xmlString, VARIANT_BOOL* isSuccessful) 将一个符合 xml 语法的文档片断或者整个文档加载到文档树中。参数 xmlString 为要加载的字符串，注意类型，参数 isSuccessful 为返回参数，标识是否成功。

5.1.4 文档的遍历

前面已经学习了如何创建和加载 XML 文档对象，那么下一个问题就是如何对文档进行操作或者是从文档中获取所需要的信息，这就要求能够通过 DOM 树来访问树中的任何一个节点，也就是对 DOM 树的遍历。

首先应该了解文档树结构，然后就可以操作树中的每一个节点。

下面我们给出一个新的 xml 文档 users.xml。

程序清单 5.2:

```

<?xml version="1.0" encoding="gb2312" ?>
<Site Address="www.communiage.com">
  <Pages>
    <Page Address="/index.htm">
      <Users>
        <User>
          <ID>234132</ID>
          <NickName>肖</NickName>
          <State>在线</State>
          <LandTime>12:23:32</LandTime>
          <IP>166.111.168.6</IP>
          <System>Win98</System>
          <Browser>IE</Browser>
        </User>
        <User>
          <ID>23132</ID>
          <NickName>张</NickName>
          <State>下线</State>
          <LandTime>12:23:32</LandTime>
          <IP>166.111.168.6</IP>
          <System>Win98</System>
          <Browser>IE</Browser>
        </User>
      </Users>
    </Page>
    <Page Address="/product/index.htm">

```

```

    <Users>
      <User>
        <ID>234132</ID>
        <NickName>王</NickName>
        <State>隐身</State>
        <LandTime>12:23:32</LandTime>
        <IP>166.111.168.6</IP>
        <System>Win98</System>
        <Browser>IE</Browser>
      </User>
      <User>
        <ID>23132</ID>
        <NickName>李</NickName>
        <State>在线</State>
        <LandTime>12:23:32</LandTime>
        <IP>166.111.168.6</IP>
        <System>Win98</System>
        <Browser>IE</Browser>
      </User>
    </Users>
  </Page>
</Pages>
</Site>

```

获得节点

一般通过两个方法得到树中的节点，分别为 `nodeFromID` 和 `getElementsByTagName`。下面我们分别用 VB 和 VC 两种语言来看一下它们的具体应用。

(1) VB 例子

程序清单 5.3:

```

Dim dom
  Dim nodelist As IXMLDOMNodeList
  Dim node As IXMLDOMNode
Dim i As Integer, length As Integer
  Set dom = CreateObject("microsoft.xmlDOM")
  doc.async = False
  dom.Load "c:\users.xml"
  Set nodelist = dom.getElementsByTagName("Page")
  length = nodelist.length
  For i = 0 To length - 1
    Set node = nodelist.Item(i)
    MsgBox node.xml
  Next

```


函数 `getElementsByTagName` 的参数是要获得的元素集合的元素名, 返回的是元素名为该参数的元素集合。

(2) VC 例子

程序清单 5.3

```

IXMLDOMDocument *pIXMLDOMDocument = NULL;
CString strFindText (_T("Page"));
IXMLDOMNodeList *pIDOMNodeList = NULL;
IXMLDOMNode *pIDOMNode = NULL;
long value;
BSTR bstrItemText;
HRESULT hr;
try
{
    /* 此处省略创建一个DOMDocument
    文档对象并装载具体文档的代码 */
    /* 下面的代码用来得到一个和元素名称Page相关的所有节点的集合 */
    //是否正确地得到了指向IDOMNodeList的指针
    hr = pIXMLDOMDocument->getElementsByTagName
    (strFindText.AllocSysString(), &pIDOMNodeList);
    SUCCEEDED(hr) ? 0 : throw hr;
    //得到所包含的节点的个数
    hr = pIDOMNodeList->get_length(&value);
    if(SUCCEEDED(hr))
    {
        for(int ii = 0; ii < value; ii++)
        {
            //得到具体的一个节点
            pIDOMNodeList->get_item(ii, &pIDOMNode);
            if(pIDOMNode )
            {
                //得到该节点相关的信息
                BSTR bsxml;
                pIDOMNode->get_xml(&bsxml);
                CString strxml = CString(bsxml);
                pIDOMNode->Release();
                pIDOMNode = NULL;
            }
        }
    }
}

```

这段代码的意思是在文档中获得元素名为“Page”的所有元素节点, 然后用 `get_xml`

函数将这些节点的 xml 展现出来。IXMLDOMNodeList 是一个节点集合，符合条件的所有节点通过 `getElementsByTagName` 返回，并保存在该集合中。要得到某一节点可以通过 IXMLDOMNodeList 的函数 `get_item` 来获得。

函数 `HRESULT get_item(long index, IXMLDOMNode** nodeItem)` 用来获得节点集合中的某一个节点指针。参数 `index` 为节点在该节点集合中的索引位置，`nodeItem` 为返回的节点指针。

VB 中的用法也差不多，而且更简单，此处就不再叙述了。

如果你要访问的是文档中的根节点，如 `users.xml` 中的 `Pages` 节点，用文档的 `documentElement` 属性就更简单了。如：

```
Set rootNode = doc.documentElement.
```

节点包括元素、属性等，是一个完整的树型结构。如果要获得节点指针对应的元素指针，如前面获得的某个 `Page` 节点，它包括元素 `<Page>`，属性 `Address`。在 VC 中可以通过接口的 `QueryInterface` 函数获得元素指针。反过来，通过该函数可以获得某个元素对应的节点指针。

例子：

```
IXMLDOMNode* pNode = NULL;
//获得节点的代码，如前所述的方法。
IXMLDOMElement* pElement = NULL;
Assert(pNode);
pNode->QueryInterface(IID_IXMLDOMElement, (void**)&pElement);
```

反之获得节点指针一样。

函数 `HRESULT QueryInterface (REFIID iid, void** ppvObject)`，参数 `iid` 为要获得的接口指针接口 ID，参数 `ppvObject` 为要返回的接口指针。

获得了元素接口指针后，同样可以通过 `getElementsByTagName` 函数来获得该元素下满足条件的节点了，而不是整个文档满足条件的节点。现在我们可以根据节点间的父子关系引用适当的方法对任何节点进行遍历。

文档的遍历

下面我们就通过一个例子来说明如何引用这些方法来对 XML 文档进行遍历，分别用 VB 和 VC 两种语言写出来。我们选用的 XML 文档还是在前面介绍过的 `books.xml`。

1. VB 例子

如果要访问第二个 `book` 节点，就可以用如下的代码：

```
Set book1=rootNode.childNodes(1)
```

也可以以第二个 `book` 节点为基准来访问其他的节点：

☒ `book1.ownerDocument` 返回 `Document` 节点，指向 XML 文档本身

- ☒ book1.previousSibling 返回第 1 个 book 节点
- ☒ book1.parentNode 返回 books 节点
- ☒ book1.firstChild 返回 name 节点
- ☒ book1.lastChild 返回 price 节点
- ☒ book1.childNodes 返回子节点集合。

由于节点记数从 0 开始, 所以 book1.childNodes(0)的结果与 book1.firstChild 的结果是一样的。

下面看一段比较复杂的代码:

```
Dim myDoc
Set myDoc = CreateObject("microsoft.xmlDOM")
myDoc.async = False
myDoc.load("books.xml")
Set rootNode = myDoc.documentElement
Set bookNode = rootNode.childNodes.item(0)
Set authorNode = bookNode.childNodes.item(1)
Set textNode = authorNode.childNodes.item(0)
Set theName = textNode.nodeValue
```

这段代码返回的值是“翁宇翔”, 来详细分析如下:

在上面的代码中, rootNode 是文档的根元素 books 节点, bookNode 和 authorNode 分别是元素类型的节点 book(第一个)和 author, textNode 是 TEXT 类型的节点, theName 则是字符串“翁宇翔”。

2. VC 例子

```
IXMLDOMDocument* pDoc = NULL;
//此处省略创建一个DOMDocument文档对象并装载具体文档的代码
IXMLDOMElement pRootElement = NULL;
IXMLDOMNode* pRootNode = NULL;
IXMLDOMNode* pbookNode = NULL;
IXMLDOMNode* pnameNode = NULL;
IXMLDOMNodeList* pbookList = NULL;
IXMLDOMNodeList* pnameList = NULL;

HRESULT hr = pDoc->get_documentElement(&pRootElement);
If( SUCCEEDED( hr ) )
{
    hr = pRootElement-> QueryInterface(IID_IXMLDOMNode,(void**)&pRootNode);
    if( SUCCEEDED( hr ) )
    {
        pRootNode->get_childNodes(&pbookList);
        hr = pbookList->get_item( 0,&pbookNode)
        if( SUCCEEDED( hr ) )
```

```

{
    pbookNode->get_childNodes(&pnameList);
    hr = pnameList->get_item(0,&pnameNode);
    if( SUCCEEDED( hr ) )
    {
        BSTR bsText;
        PnameNode->get_text(&bsText);
        CString strText = (CString)bsText;
        AfxMessageBox(strText);//strText = "翁宇翔"
        pnameNode->Release();
    }
    pnameList->Release();
    pbookNode->Release();
}
pbookList->Release();
pRootNode->Release();
}
pRootElement->Release();
}

```

函数 `HRESULT get_childNodes(IXMLDOMNodeList** childList)` 用来获得该节点下的第一层节点链表指针。参数 `childList` 是返回的节点集合指针。通过该集合可以获得这些子节点。

XMLParser

下面给出一个遍历文档的比较完整的例子，是用 VC 写的。

本程序是一个基于对话框的 MFC 应用程序，用的是微软的 `msxml2.0` 解析器，这需要安装 IE5 以上。在工程的设置中加上 `msxml2.lib` 库，头文件要包括 `msxml.h`。

只要在对话框的编辑框中输入 xml 文档的完整路径，本程序就可以将它完全解析，并在树中展示出来，包括解析元素、属性、属性值、元素值等。

如果将前面的 `usres.xml` 解析，结果如图 5-2。

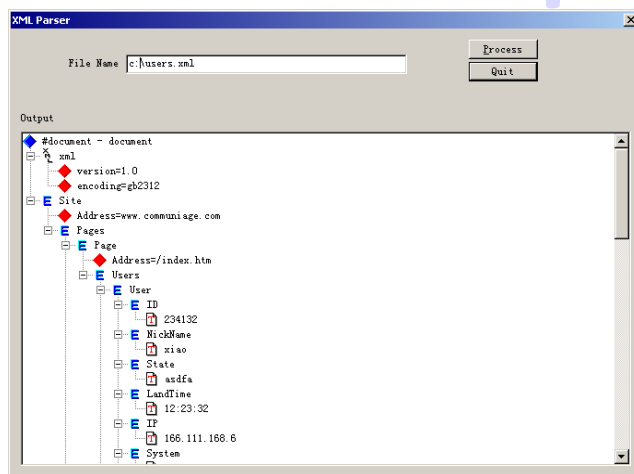


图 5-2 遍历文档实例

程序的流程图如图 5-3。

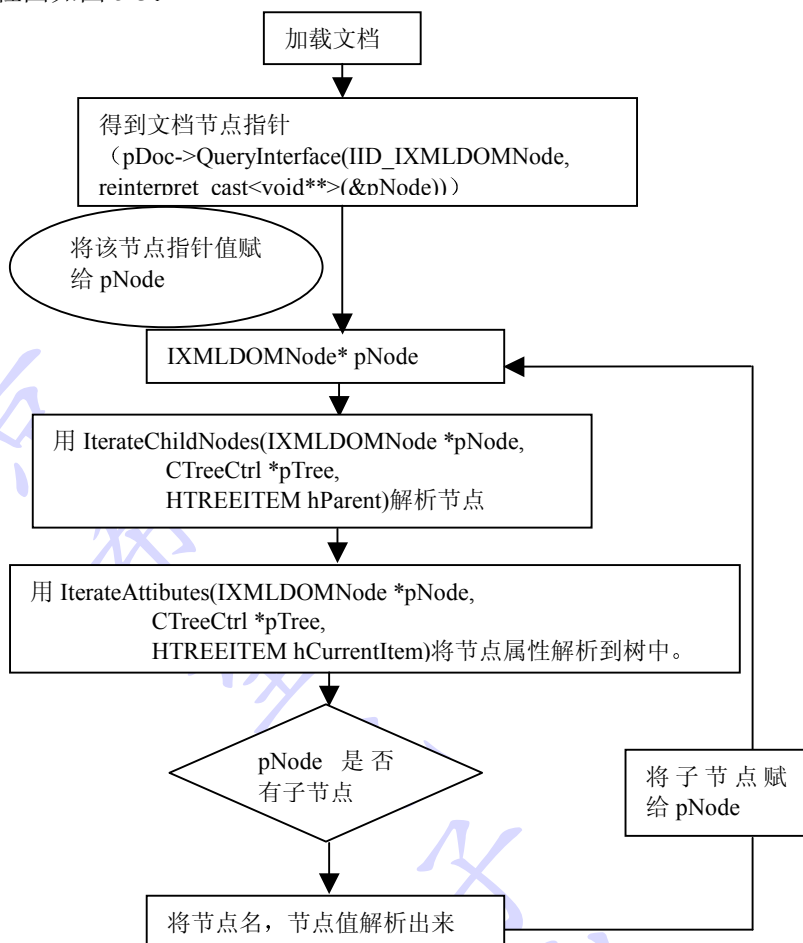


图 5-3 程序流程

解析代码都是在对话框的.cpp 文件中写的，当用户点击 Process 按钮的时候开始解析，程序是从头到尾将所有节点都遍历一遍。

下面将对话框类的头文件和.cpp 文件都显示出来，读者可以慢慢研究。

程序清单 5.4:

```

// XmlParserDlg.h : header file
//
///////////////////////////////////////////////////////////////////
// Include the MSXML stuff.
//
#include "msxml.h"

/////////////////////////////////////////////////////////////////
// CXmlParserDlg dialog

```

```

class CXmlParserDlg : public CDialog
{
    DECLARE_DYNAMIC(CXmlParserDlg);

// Construction
public:
    CXmlParserDlg(CWnd* pParent = NULL); // standard constructor
    virtual ~CXmlParserDlg();

// Dialog Data
//{{AFX_DATA(CXmlParserDlg)
enum { IDD = IDD_XMLPARSER_DIALOG };
    // NOTE: the ClassWizard will add data members here
//}}AFX_DATA

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CXmlParserDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
    HICON m_hIcon;

    BOOL CanExit();

// Generated message map functions
//{{AFX_MSG(CXmlParserDlg)
virtual BOOL OnInitDialog();
afx_msg void OnGo();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()

//
// XML support code.
//
    HRESULT CheckLoad(IXMLDOMDocument *pDoc);
    HRESULT ReportError(IXMLDOMParseError *pXMLError);
    bool IterateChildNodes(IXMLDOMNode *pNode,
        CTreeCtrl *pTree, HTREEITEM hParent=NULL);
    bool IterateAttributes(IXMLDOMNode *pNode,
        CTreeCtrl *pTree, HTREEITEM hCurrentItem);

```

};

程序清单 5.5:

```

/*****
// XmlParserDlg.cpp : implementation file
//

#include "stdafx.h"
#include "xmlparser.h"
#include "XmlParserDlg.h"

#include <atlbase.h>

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// CXmlParserDlg dialog

IMPLEMENT_DYNAMIC(CXmlParserDlg, CDialog);

CXmlParserDlg::CXmlParserDlg(CWnd* pParent /*=NULL*/)
: CDialog(CXmlParserDlg::IDD, pParent)
{
//{{AFX_DATA_INIT(CXmlParserDlg)
// NOTE: the ClassWizard will add member initialization here
//}}AFX_DATA_INIT
// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
}

CXmlParserDlg::~CXmlParserDlg()
{
}

void CXmlParserDlg::DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CXmlParserDlg)
// NOTE: the ClassWizard will add DDX and DDV calls here
//}}AFX_DATA_MAP
}

```



```

BEGIN_MESSAGE_MAP(CXmlParserDlg, CDialog)
//{{AFX_MSG_MAP(CXmlParserDlg)
    ON_BN_CLICKED(IDC_GOBTN, OnGo)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CXmlParserDlg message handlers

BOOL CXmlParserDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    //
    // Load the bitmap IDB_TREE into the tree control.
    //
    CImageList imgTree;
    imgTree.Create(IDB_TREE, 16, 0, RGB(255, 51, 255));
    CTreeCtrl *pTree = static_cast<CTreeCtrl*>(GetDlgItem(IDC_XMLTREE));
    if ( pTree )
    {
        pTree->SetImageList(&imgTree, TVSIL_NORMAL);
        imgTree.Detach();
    }

    return TRUE; // return TRUE unless you set the focus to a control
}

////////////////////////////////////
// OnGo
//
// 当用户按Process按钮时触发该事件
//
void CXmlParserDlg::OnGo()
{
    //
    // Check the file specified exists.
    //
    CString strFileName;
    CEdit *pEdit = static_cast<CEdit*>(GetDlgItem(IDC_EDIT1));
    if ( pEdit )
        pEdit->GetWindowText(strFileName);
}

```

```
//
// Assuming it's not an empty string, try to process.
//
if ( !strFileName.IsEmpty() )
{
    if ( SUCCEEDED(CoInitialize(NULL)) )
    {
        IXMLDOMDocument *pDoc;

        if ( SUCCEEDED (CoCreateInstance(CLSID_DOMDocument,
            NULL,
            CLSCTX_INPROC_SERVER,
            IID_IXMLDOMDocument,
            reinterpret_cast<void*>(&pDoc))))
        {
            //
            // Tell the "doc" that we're not going to load asynchronously.
            //
            if ( SUCCEEDED(pDoc->put_async(VARIANT_FALSE)) )
            {
                CComVariant vFile(strFileName);
                VARIANT_BOOL vBool;
                pDoc->load(vFile,&vBool);
                if ( vBool == VARIANT_TRUE )
                {
                    IXMLDOMNode *pNode;
                    if ( SUCCEEDED(pDoc->QueryInterface
                        (IID_IXMLDOMNode,
                        reinterpret_cast<void*>(&pNode))))
                    {
                        CtreeCtrl* pTree =
                            static_cast<CtreeCtrl*>
                                (GetDlgItem(IDC_XMLTREE));
                        HTREEITEM hItem =
                            pTree->GetChildItem(TVI_ROOT);
                        while( hItem != NULL )
                        {
                            HTREEITEM hNext =
                                pTree->GetNextSiblingItem(hItem);
                            pTree->DeleteItem(hItem);
                            hItem = hNext;
                        }
                    }
                }
            }
        }
    }
}
```

```

        }

        IterateChildNodes(pNode,pTree);
        pNode->Release();
        pNode = NULL;
    }
}
else
    CheckLoad(pDoc);
}
pDoc->Release();
pDoc = NULL;
}
CoUninitialize();
}
}
}

////////////////////////////////////
// CheckLoad
//
// Ensure an XML document loaded correctly.
//
HRESULT CXmlParserDlg::CheckLoad(IXMLDOMDocument *pDoc)
{
    HRESULT hResult = E_FAIL;
    long IErrorCode = E_FAIL;
    IXMLDOMParseError *pXMLError = NULL;

    if (SUCCEEDED(pDoc->get_parseError(&pXMLError))
        && SUCCEEDED(pXMLError->get_errorCode(&IErrorCode))
        && ( IErrorCode != 0 ) )
        hResult = ReportError(pXMLError);

    //
    // Clean-up pointers used.
    //
    if ( pXMLError )
    {
        pXMLError->Release();
        pXMLError = NULL;
    }
}

```

```

//
// Pass back the return code.
//
return IErrorCode;
}

////////////////////////////////////
// ReportError
//
// Standardised reporting mechanism for COM errors to do with XML.
//
HRESULT CXmlParserDlg::ReportError(IXMLDOMParseError *pXMLError)
{
    long ILine;
    long ILinePos;
    long IErrorCode;
    BSTR bstrFile;
    BSTR bstrReason;

    //
    // Whilst these could all return errors, we'll assume that since the
    // COM was awake enough to report an error (and return the structure)
    // that it can work with these calls.
    //
    pXMLError->get_line(&ILine);
    pXMLError->get_linepos(&ILinePos);
    pXMLError->get_errorCode(&IErrorCode);
    pXMLError->get_reason(&bstrReason);
    pXMLError->get_url(&bstrFile);

    //
    // Format this message into a suitable output.
    //
    if ( ILine > 0 )
    {
        CString strError;
        strError.Format(_T("Error processing XML file:
            %S\nError on line %d, position %d\n")
            _T("Error (%x) text: %S"),
            bstrFile,
            ILine, ILinePos, IErrorCode,
            bstrReason);
        MessageBox(strError, _T("Parse Error"), MB_OK|MB_ICONERROR);
    }
}

```

```

        TRACE(_T("%s\n"),strError);
    }

    SysFreeString(bstrFile);
    SysFreeString(bstrReason);

    return NOERROR;
}

////////////////////////////////////
// WalkTopLevel
//
bool CXmlParserDlg::IterateChildNodes(IXMLDOMNode *pNode,
                                       CTreeCtrl *pTree,
                                       HTREEITEM hParent)
{
    BSTR bstrNodeName;
    HTREEITEM hCurrentItem;

    if ( pTree )
    {
        if ( pNode )
        {
            CString strOutput;
            pNode->get_nodeName(&bstrNodeName);

            //
            // Find out the node type (as a string).
            //
            BSTR bstrNodeType;
            pNode->get_nodeTypeString(&bstrNodeType);
            CString strType;
            strType.Format(_T("%S"),bstrNodeType);
            SysFreeString(bstrNodeType);

            DOMNodeType eEnum;
            pNode->get_nodeType(&eEnum);

            int nImg = 0;

            CString strValue;
            if ( eEnum == NODE_TEXT )
            {

```

```
BSTR bstrValue;
pNode->get_text(&bstrValue);
strOutput.Format(_T("%S"),bstrValue);
SysFreeString(bstrValue);

nImg = 2;
}
else if ( eEnum == NODE_COMMENT )
{
    VARIANT vValue;
    pNode->get_nodeValue(&vValue);

    CString strValue;
    if ( vValue.vt == VT_BSTR )
        strOutput.Format(_T("%S"),V_BSTR(&vValue));
    else
        strOutput.Format(_T("Unknown comment type"));
    VariantClear(&vValue);

    nImg = 3;
}
else if ( eEnum == NODE_PROCESSING_INSTRUCTION )
{
    strOutput.Format(_T("%S"), bstrNodeName);
    nImg = 4;
}
else if ( eEnum == NODE_ELEMENT )
{
    strOutput.Format(_T("%S"), bstrNodeName);
    nImg = 5;
}
else
{
    //
    // Other types, include the type name.
    //
    strOutput.Format(_T("%S - %s"), bstrNodeName,strType);
}
SysFreeString(bstrNodeName);

//
// Add it to the tree view.
//
```

```

        TVINSERTSTRUCT tvItem;
        tvItem.hParent = hParent ? hParent : TVI_ROOT;
        tvItem.hInsertAfter = TVI_LAST;
        tvItem.item.ilImage = nImg;
        tvItem.item.iSelectedImage = nImg;
        tvItem.item.pszText = strOutput.GetBuffer(MAX_PATH);
        tvItem.item.cchTextMax = MAX_PATH;
        tvItem.item.mask = TVIF_IMAGE
            | TVIF_SELECTEDIMAGE | TVIF_TEXT;
        hCurrentItem = pTree->InsertItem(&tvItem);
        strOutput.ReleaseBuffer();

        IterateAttributes(pNode, pTree, hCurrentItem);
    }
}

//
// Any child nodes of this node need displaying too.
//
IXMLDOMNode *pNext = NULL;
IXMLDOMNode *pChild;
pNode->get_firstChild(&pChild);
while( pChild )
{
    IterateChildNodes(pChild, pTree, hCurrentItem);
    pChild->get_nextSibling(&pNext);
    pChild->Release();
    pChild = pNext;
}

//
// Ensure after all of that, the item is expanded!
//
pTree->Expand(hCurrentItem, TVE_EXPAND);
return true;
}

```

```

////////////////////////////////////
// IterateAttributes
//
bool CXmlParserDlg::IterateAttributes(IXMLDOMNode *pNode,
                                       CTreeCtrl *pTree,

```



```

        HTREEITEM hCurrentItem)
{
    IXMLDOMNamedNodeMap *pAttrs;

    if ( SUCCEEDED(pNode->get_attributes(&pAttrs)) && (pAttrs != NULL) )
    {
        IXMLDOMNode *pChild;
        pAttrs->nextNode(&pChild);
        while(pChild)
        {
            BSTR bstrName;
            VARIANT vValue;

            pChild->get_nodeName(&bstrName);
            pChild->get_nodeValue(&vValue);

            CString strValue;
            switch ( vValue.vt )
            {
            case VT_BSTR:
                strValue.Format(_T("%S"),V_BSTR(&vValue));
                break;
            default:
                strValue = _T("Unsupport type");
                break;
            }

            CString strAttrib;
            strAttrib.Format(_T("%S=%s"),bstrName,strValue);
            SysFreeString(bstrName);
            VariantClear(&vValue);

            pChild->Release();
            pAttrs->nextNode(&pChild);

            //
            // Add the attributes to the tree too.
            //
            TVINSERTSTRUCT tvAttribItem;
            tvAttribItem.hParent = hCurrentItem;
            tvAttribItem.hInsertAfter = TVI_LAST;
            tvAttribItem.item.ilImage = 1;
            tvAttribItem.item.iSelectedImage = 1;

```

```

        tvAttrItem.item.pszText = strAttrib.GetBuffer(MAX_PATH);
        tvAttrItem.item.cchTextMax = MAX_PATH;
        tvAttrItem.item.mask = TVIF_IMAGE |
            TVIF_SELECTEDIMAGE | TVIF_TEXT;
        pTree->InsertItem(&tvAttrItem);
        strAttrib.ReleaseBuffer();
    }
    pAttrs->Release();
}
return true;
}

```

本程序有两个函数比较重要，用来解析节点和元素属性。

函数 `IterateChildNodes(IXMLDOMNode *pNode, CTreeCtrl *pTree, HTREEITEM hParent)` 用来将节点 `pNode` 解析到树控件中。参数 `pNode` 是节点指针，`pTree` 是树控件指针，`hParent` 是该节点指针要加入到树控件中的位置。

函数 `IterateAttributes(IXMLDOMNode *pNode, CTreeCtrl *pTree, TREEITEM hCurrentItem)` 将节点的属性都解析到树控件中。参数 `pNode` 是节点指针，`pTree` 是树控件指针，`hCurrentItem` 是要加入到树控件中的位置。

5.2 文档元素的添加、删除和内容修改

目前，我们已经能够通过 DOM 获取 XML 文档中的信息了。如前所述，通过 DOM 还可以动态地更改 XML 文档中的内容。下面仍旧使用上面的 `books.xml` 文档，通过实例来说明如何更改 XML 文档中的内容。

5.2.1 添加元素

假如希望在 `books.xml` 中，给第一个 `book` 元素增添一个字符串为“人民邮电出版社”的 `publisher` 元素，实现这一添加元素操作的语句如下，分别用 VB 和 VC 两种语言写：

1. VB 例子

```

Set node = rootNode.childNodes.item(0)
Set newNode = myDocument.createElement("publisher")
Set rtnNode = node.insertBefore(newNode, node.lastChild)
Set textNode = myDocument.createTextNode("人民邮电出版社")
node.childNodes.item(1).appendChild(textNode)

```

2. VC 例子

```

IXMLDOMDocument* pDoc = NULL;
//此处省略创建一个DOMDocument文档对象并装载具体文档的代码
IXMLDOMElement pRootElement = NULL;
IXMLDOMNode* pRootNode = NULL;

```

```

IXMLDOMNode* pbookNode = NULL;
IXMLDOMNode* pnameNode = NULL;
IXMLDOMNodeList* pbookList = NULL;
IXMLDOMNodeList* pnameList = NULL;

HRESULT hr = pDoc->get_documentElement(&pRootElement);
If( SUCCEEDED( hr ) )
{
    hr = pRootElement-> QueryInterface(IID_IXMLDOMNode,(void**)&pRootNode);
    if( SUCCEEDED( hr ) )
    {
        pRootNode->get_childNodes(&pbookList);
        hr = pbookList->get_item( 0,&pbookNode)
        if( SUCCEEDED( hr ) )
        {
            CComVariant varType(NODE_ELEMENT);
            CString strNodeName = _T("publisher");
            IXMLDOMNode* pPublisherNode = NULL;
            pDoc->createNode(varType,strNodeName.AllocSysString(),NULL,& pPublisherNode);
            CString strValue = "人民邮电出版社";
            pPublisherNode->put_text(strValue.AllocSysString());
            IXMLDOMNode* pOutNode = NULL;
            pbookNode ->appendChild(pPublisherNode,& pOutNode);
            CString strFilePath = "c:\\books.xml";
            CComVariant comFile(strFilePath);
            pDoc->save(comFile);
            pOutNode->Release();
            pPublisherNode->Release();
            pbookNode->Release();
        }
        pbookList->Release();
        pRootNode->Release();
    }
    pRootElement->Release();
}

```

HRESULT createNode(VARIANT Type,BSTR name,BSTR nameSpaceURI,IXMLDOMNode** node)

函数用来创建一个元素节点

参数 Type 为创建的元素节点类型, 参数 nameSpaceURI 为 URI 命名空间, 一般为 NULL, 参数 node 为返回的创建成功的元素节点指针。

函数 HRESULT put_text(BSTR* text)用来给一个元素赋值。参数 text 为要给该元素赋的

值。

函数 `HRESULT appendChild(IXMLDOMNode* newChild,IXMLDOMNode* outChild)`用来将一个节点挂在该父节点的子节点集合的最后。参数 `newChild` 为要挂的新子节点，参数 `outChild` 为挂后返回的在该子节点集合中的指针。

在添加操作之后的结果如下所示：

程序清单 5.6:

```
<?xml version="1.0" encoding="gb2312"?>
<books>
  <book>
    <name>XML基础</name>
    <author>翁宇翔</author>
    <author>芮云</author>
    <author>张建飞</author>
    <publisher>人民邮电出版社</publisher>
    <price unit="RMB">88.88</price>
  </book>
  <book>
    <name>XML进阶</name>
    <author>翁宇翔</author>
    <author>芮云</author>
    <author>张建飞</author>
    <price unit="RMB">188.88</price>
  </book>
</books>
```

5.2.2 删除元素

如果要把刚才添加的元素删除，可通过下面的代码实现，分别用 VB 和 VC 表达。

1. VB 例子

```
Set node = root.childNodes.Item(0)
Set oldNode = node.removeChild(node.childNodes.Item(4))
```

其中，`oldNode` 中存放的是已被删除的节点。在删除某个节点时，以该节点为根的子树将整个被删除。

2. VC 例子

```
IXMLDOMNode* pParentNode = NULL;
IXMLDOMNode* pchildNode = NULL;
//获得pParentNode和pchildNode的代码
ASSERT(pParentNode);
ASSERT(pchildNode);
IXMLDOMNode* pOutNode = NULL;
```

```
pParentNode->removeChild(pchildNode,&pOutNode);
```

先得到要删除的节点 `pChildNode`，然后用父节点指针 `pParentNode` 调用 `removeChild` 函数将 `pchildChild` 从其子节点集合中删除，而参数 `pOutNode` 是将 `pChildNode` 集合中去除后的指针值。

5.2.3 改变元素内容

如果要把上面源程序中第一本书的定价改成 158 元的话，可以用如下的代码实现：

1. VB 例子

```
Set node = root.childNodes.Item(0)
Set priceNode = node.childNodes.Item(4)
priceNode.childNodes.Item(4).nodeValue = " 158"
```

2. VC 例子

先获得要更改的 `price` 元素的节点指针，然后调用 `put_text` 函数就可以，例子看前面的添加元素的例子。

更改操作之后的结果如下所示：

程序清单 5.7:

```
<?xml version="1.0" encoding="gb2312"?>
<books>
  <book>
    <name>XML基础</name>
    <author>翁宇翔</author>
    <author>芮云</author>
    <author>张建飞</author>
    <price unit="RMB">158</price>
  </book>
  <book>
    <name>XML进阶</name>
    <author>翁宇翔</author>
    <author>芮云</author>
    <author>张建飞</author>
    <price unit="RMB">188.88</price>
  </book>
</books>
```

前面对 DOM 树的常用操作做了简单的介绍。一般说来，支持 DOM 的 XML 分析器通常会会对 DOM 做一些扩展，这些扩展不属于 DOM 规范中的标准，但却给 DOM 树的操作带来了方便，不同的分析器所做的扩展也不尽相同，可以通过查询相关技术支持资料或者帮助来获取更多的信息。

5.2.4 错误处理

DOM 在解析 XML 文档的时候可能会产生各式各样的错误，读者可以根据 `ParseError` 对象中的属性得知出错的可能原因及相关信息。

可以调用 `ParseError` 对象，通过建立一个指向 `IXMLDOMParseError` 的接口就可以和 `ParseError` 的属性相交互。`IXMLDOMParseError` 一共提供了七个属性，可以通过它来获取详细的信息。这些属性及其含义如表 5-1 所示。

表 5-1 错误处理

属性	含义
<code>ErrorCode</code>	错误代码
<code>Filepos</code>	错误在文档中的绝对字符位置
<code>Line</code>	错误所在行的行号
<code>Linepos</code>	错误所在行的字符位置
<code>Reason</code>	错误产生原因
<code>SrcText</code>	错误所在行的源代码
<code>url</code>	最近一份含有解析错误的 XML 文档的 URL 地址

在实际调试中可以套用如下的代码段：

```
Dim xDoc MSXML.DOMDocument
Set xDoc=New MSXML.DOMDocument
If xDoc.Load("你的XML文档") then
    '成功装载文档
    '处理代码
Else
    '不能装载文档
    Dim strError as string
    dim ERROR as IXMLDOMParseError
    set ERROR as xDoc.parseError
    With ERROR
        strError="不能装载文件"&_
            "因为下面的原因"&vbCrLf&_
            "Error#:"&.errorCode&":"&ERROR.reason&_
            "Line#:"&.Line&vbCrLf&_
            "Line Position:"&.linepos&vbCrLf&_
            "Position In File:"&.filepos&vbCrLf&_
            "Source Text:"&.srcText&vbCrLf&_
            "Docement URL:"&.url
    end with
    MsgBox strErrText,vbExclamation
End If
```

Set ERROR=Nothing

上面的例子当 XML 文档解析出错的时候,将把来自 `ParseError` 的出错信息显示给用户, 这些信息包括出错的原因、位置等。后面的章节将依照规范对几个主要的接口作一详细的介绍。

5.3 DOM 接口

在对 DOM 有了总体上的了解之后,现在来看一下 DOM 规范中的几个主要接口。这里简单的以表格的形式来回顾一下 DOM 的几个重要接口。

5.3.1 Document 接口

`Document` 接口代表了整个 XML 文档,提供了对文档中的数据进行访问和操作的入口。

由于元素、文本节点、注释、处理指令等都不能脱离文档的上下文关系而独立存在,所以在 `Document` 接口提供了创建其他节点对象的方法,通过该方法创建的节点对象都有一个 `ownerDocument` 属性,用来表明当前节点是由谁所创建的以及节点同 `Document` 之间的联系。

`Document` 节点是 DOM 树中的根节点,也即对 XML 文档进行操作的入口节点。通过 `Document` 节点,可以访问到文档中的其他节点,如处理指令、注释、文档类型以及 XML 文档的根元素节点等等。

表 5-2 给出了 `Document` 接口的常用属性。

表 5-2 Document 接口的常用属性

属性名	性质	含义
Doctype	DocumentType 类型的只读属性	记录文档的文档类型声明
documentElement	Element 类型的只读属性	该属性为访问文档的根元素提供了一种简单的方法

表 5-3 是 `Document` 接口的常用方法。

表 5-3 Document 接口的常用方法

方法名	含义	参数	返回值	异常 DOMException
CreateAttribute	创建一个具有给定的名称的属性节点,然后可以利用 <code>setAttributeNode</code> 方法把该属性设置为某个元素的属性	DOMString name (属性的名字)	Attr 类型的属性节点	INVALID_CHARACTER_ERR
createCDATASection	创建一个 <code>CDATASection</code> 节点,节点的值就是参数 <code>data</code> 中所传递的内容	DOMString data (<code>CDATASection</code> 的内容)	<code>CDATASection</code> 类型的节点	NOT_SUPPORTED_ERR

方法名	含义	参数	返回值	异常 DOMException
CreateMessage	创建一个注释节点, 节点的值就是参数 data 中所传递的内容	DOMString data (Message 节点的内容)	Message 类 型的节点	无
createElement	创建一个指定类型的元素节点	DOMString tagName	Element 类 型的节点	INVALID_CHAR ACTER_ERR
createEntityReference	创建一个实体引用节点。如果被引用的实体是已知的, 那么该实体应用节点跟相应的实体节点具有同样的子节点集	DOMString name (要创建的引用实体的名字)	EntityReference 类型 的节点	INVALID_CHAR ACTER_ERR; NOT_SUPPORTED_ERR
createProcessingInstruction	创建一个具有指定的名字和数据的处理指令节点	DOMString target (处理指令的目标部分) DOMString data: 处理指令的数据部分	ProcessingInstruction 类型的节点	INVALID_CHAR ACTER_ERR; NOT_SUPPORTED_ERR
createTextNode	创建一个具有指定的字符串内容的文本节点	DOMString data (文本节点的内容)	Text 类型 的节点	无
getElementById	返回一个具有给定 ID 的元素节点。如果没有这样的元素存在, 就返回 NULL	DOMString elementId (要匹配的元素 id 值)	相匹配的 元素节点	无
getElementsByTagName	返回一个节点的集合 (NodeList), 该集合中所有的元素都具有参数中所给定的标记名, 集合中的所有元素按照在 DOM 树中的前序排列进行排序	DOMString tagname (要匹配的元素标记名)	相匹配的 元素节点的集合	无

5.3.2 Node 接口

现在来介绍另一个重要的接口 Node, DOM 规范中有很大部分接口是从 Node 接口继承过来的, 例如, Element, Attr, CDATASection 等接口, 都是从 Node 继承过来的。

在 DOM 树中, 一个 Node 接口实例代表了树中的一个节点。DOM 树中包含很多各种不同类型的节点, 这些节点基本上都是从 Node 继承过来的。Node 接口中定义了所有不同类型的节点中都具有的属性和方法。

如前所述, DOM 规范中定义了很多不同类型的节点。在这些节点类型定义中, 采用不

同的整数来代表不同的节点类型。为了保证日后能够很容易的对节点类型进行扩展，W3C 保留了 1~200 之间的整数，作为不同的节点类型的定义。

节点类型中的常量定义如表 5-4 所示：

表 5-4 节点类型中的常量定义

常量	含义
ATTRIBUTE_NODE	属性节点 (Attr)
CDATA_SECTION_NODE	CDATA 节点 (CDATASection)
MESSAGE_NODE	注释节点 (Message)
DOCUMENT_FRAGMENT_NODE	文档片断节点 (DocumentFragment)
DOCUMENT_NODE	Document 节点 (Document)
DOCUMENT_TYPE_NODE	文档类型节点 (DocumentType)
ELEMENT_NODE	元素节点 (Element)
ENTITY_NODE	实体节点 (Entity)
ENTITY_REFERENCE_NODE	实体引用节点 (EntityReference)
NOTATION_NODE	Notation 节点 (Notation)
PROCESSING_INSTRUCTION_NODE	处理指令节点 (ProcessingInstruction)
TEXT_NODE	Text 节点 (Text)

下面来介绍 Node 接口的常用属性，如表 5-5。

表 5-5 Node 接口的常用属性

属性名	性质	含义
attributes	NamedNodeMap 类型的只读属性	如果当前节点是 ELEMENT_NODE 类型的节点，则 NamedNodeMap 中包含了当前元素的所有属性信息。如果当前节点的节点类型不是 ELEMENT_NODE，则属性值为 null
childNodes	NodeList 类型的只读属性	一个 NodeList 类型的实例中，包含了当前节点的所有子节点。如果当前节点没有子节点，NodeList 中就不包含任何节点
firstChild	Node 类型的只读属性	当前节点的第一个子节点，如果没有这样的节点，就返回 null
lastChild	Node 类型的只读属性	当前节点的最后一个子节点，如果没有这样的节点，就返回 null
nextSibling	Node 类型的只读属性	当前节点的直接后继节点，如果没有这样的节点，就返回 null
nodeName	DOMString 类型的只读属性	当前节点的名字，对于不同的节点类型，有不同的取值
nodeType	unsigned short 类型的只读属性	当前节点的类型

属性名	性质	含义
nodeValue	DOMString 类型的属性	当前节点的值, 对于不同的节点类型, 该属性具有不同的取值
parentNode	Node 类型的只读属性	当前节点的父节点
previousSibling	Node 类型的只读属性	当前节点的直接前驱节点。如果没有这样的节点, 就返回 null。

表 5-6 给出了 Node 接口几种重要的方法。

表 5-6 Node 接口几种重要的方法

方法名	含义	参数	返回值	异常 DOMException
appendChild	把参数中传递过来的 newChild 添加到当前节点的所有的子节点列表最后	Node newChild, 要添加的节点	添加的节点	HIERARCHY_REQUEST_ERR; WRONG_DOCUMENT_ERR; NO_MODIFICATION_ALLOWED_ERR;
cloneNode	复制当前节点。复制产生的节点没有父节点 (parentNode 是 null)。包括复制该元素本身的所有的属性和属性值, 以及由 XML 处理器生成的缺省的属性和属性值	boolean deep	复制的节点	无
hasChildNodes	判断当前节点是否有子节点	DOMString data, Message 节点的内容	Message 类型的节点	无
createElement	创建一个指定类型的元素节点	无	boolean true 如果当前节点包含有子节点, 返回值为真, 否则, 返回值为假	无
insertBefore	把节点 newChild 插入到当前节点的子节点 refChild 之前	Node newChild, 要插入的节点。 Node refChild,	插入的节点	HIERARCHY_REQUEST_ERR; WRONG_DOCUMENT_ERR ;

方法名	含义	参数	返回值	异常
				DOMException
		用来指明要把新添加的节点插入到哪个节点之前		NO_MODIFICATION_ALLOWED_ERR ; NOT_FOUND_ERR
removeChild	从当前节点的子节点中删除 oldChild 节点, 并返回 oldChild 节点	Node oldChild: 要删除的节点	被删除的节点	NO_MODIFICATION_ALLOWED_ERR ; NOT_FOUND_ERR
replaceChild	用 newChild 节点代替当前节点中的子节点 oldChild , 并返回 oldChild 节点	Node newChild : 要插入到当前节点子节点列表中的节点。 Node oldChild: 将要被替代的节点	被删除的节点	HIERARCHY_REQUEST_ERR; WRONG_DOCUMENT_ERR; NO_MODIFICATION_ALLOWED_ERR NOT_FOUND_ERR

5.3.3 NodeList 接口

NodeList 接口提供了对节点集合的抽象定义。NodeList 用于表示有顺序关系的一组节点, 比如某个节点的子节点序列; 另外, 它还出现在一些方法的返回值中, 例如 GetNodeByName 方法的返回值就是一个 NodeList 类型的节点集合。

在 DOM 中, NodeList 的对象是动态的, 也就是说对文档的改变, 会直接反映到相关的 NodeList 对象中。例如, 如果通过 DOM 获得一个 NodeList 对象, 该对象中包含了某个 Element 节点的所有子节点的集合, 那么, 当再通过 DOM 对 Element 节点进行操作(添加、删除、改动节点中的子节点)时, 这些改变将会自动地反映到 NodeList 对象中, 而不需 DOM 应用程序再做其他额外的操作。

NodeList 中的每个 item 都可以通过一个索引来访问, 该索引值从 0 开始。

下面就对 NodeList 接口的属性 length 作一个介绍。

- ☒ length: unsigned long 类型的只读属性, 该节点集合中的节点个数。访问集合中的节点的合法的索引值为 0 到 length-1 之间的数(包括 0 和 length-1)。

NodeList 还有一个没有异常的方法 item。

- ☒ Item: 返回节点集合中的第 index 个节点。index 的取值从 0 开始。如果 index 的值大于等于集合中的节点个数, 就返回 null。

这个方法的参数是 unsigned long index, 表示要获取的节点在集合中的索引值。它返回节点集合中的第 index 个节点。如果 index 的值不在合法范围内, 就返回 null。

5.3.4 NamedNodeMap 接口

实现了 NamedNodeMap 接口的对象中包含了可以通过名字来访问节点的集合。

NamedNodeMap 并不是从 NodeList 继承过来的, NamedNodeMap 所包含的节点集中的节点是无序的。实现了 NamedNodeMap 接口对象中所包含的节点也可以通过索引来进行访问, 但是, 这只是提供了一种枚举 NamedNodeMap 中所包含节点的一个简单方法, 并不表明在 DOM 规范中为 NamedNodeMap 中的节点规定一种排列顺序。

NamedNodeMap 表示的是一组节点和其唯一名字的对应关系, 这个接口主要用在属性节点的表示上。

下面就对 NamedNodeMap 接口的属性 length 作一个介绍。

☒ length : unsigned long 类型的只读属性, 该节点集中的节点个数。访问集合中的节点的合法的索引值为 0 到 length-1 之间的数 (包括 0 和 length-1)。

表 5-7 给出了 NamedNodeMap 接口的方法。

表 5-7 NamedNodeMap 接口的方法

方法名	含义	参数	返回值	异常 DOMException
getNamedItem	返回给定名称的节点	DOMString name: 要获取的节点的节点名	返回具有给定的节点名称的节点, 如果在当前的节点集中没有这样的节点, 就返回 null	无
item	返回节点集中的第 index 个节点。index 的取值从 0 开始。如果 index 的值大于等于集合中的节点个数, 就返回 null	unsigned long index: 要获取的节点在集合中的索引值 (从 0 开始)	节点集中的第 index 个节点。如果 index 的值不在合法范围内, 就返回 null	无
removeNamedItem	删除给定名称的节点	DOMString name: 要删除的节点的节点名	如果在当前节点集中存在给定名称的节点 (即存在要删除的节点), 就返回该节点	NO_MODIFICATION_ALLOWED_ERR; NOT_FOUND_ERR
setNamedItem	把给定节点名的节点添加到当前节点集中。如果在当前节点集中已经存在相同节点名的节点了, 那么就用现在的节点替换掉已经存在的节点	Node arg: 要添加到当前节点集中的节点。该节点在此之后可以通过节点的 nodeName 属性而被访问	如果在参数中传递过来的节点替换了当前节点集中已经存在的一个节点, 则返回值为被替换的节点, 否则, 返回值为 null	NO_MODIFICATION_ALLOWED_ERR; NOT_FOUND_ERR; WRONG_DOCUMENT_ERR; INUSE_ATTRIBUTE_ERR

上面已经介绍了 DOM 规范中的 Document, Node, NodeList 以及 NamedNodeMap 四个接口。这四个接口是 DOM 规范中最主要的四个接口。通过这四个接口, 已经可以完成对 XML 文档的绝大多数常用操作。

5.4 数据岛

这里先对数据岛的概念作一个简单的回顾。

数据岛指的是存在于 HTML 网页中的 XML 代码段, 它在 HTML 中形成了一个数据的集合。数据岛允许我们在 HTML 网页中集成 XML、对 XML 编写脚本, 而不需要象 HTML 那样通过脚本或<object>标签来读取 XML。

数据岛有它特有的形式, 由标记<XML>开始, 在开始标记中要有一个 ID 属性, 用于指定该数据岛的名称, 最后我们还要以</XML>结束。元素<XML>包含的内容就是 XML 代码。数据岛也分为两种, 一种为内嵌的数据岛形式, 另一种为外嵌的形式。

关于数据岛我们就简单的这么介绍这么多了。

我们在这一节所要关注的问题是如何对数据岛的节点进行访问和操作。

由于数据岛的属性结构, 访问数据岛类似于访问文档对象。XML 文档对象是指一个拥有属性和方法的对象, 我们利用这些属性和方法访问和处理 XML 文档。当一个 XML 数据岛被读取和解析的时候, 就会创建一个 XML 文档对象。

5.4.1 数据岛对象

一个数据岛对象主要有 Tagname, Text 和 Url 三种属性, 可以通过 ID 属性来访问它们。

☑ 通过 Tagname 方法得到标签的名称, 也就是"XML"这个字符串。

☑ 通过 Text 方法得到数据岛中的所有数据, 但不包括标签的名称。

下面来看一个例子, 这个例子使用了脚本语言来访问数据岛的 Tagname 和 Text 属性值。

程序清单 5.8:

```
<html>
  <head>
    <title>
      数据岛对象
    </title>
    <script language="vbs">
      sub show_onclick()
        t1.value=student.tagname
        t2.value=student.text
      end sub
    </script>
  </head>
  <body>
    <xml id="student">
      <学生>
```

```

        <学号>0001</学号>
        <姓名>张三</姓名>
        <系别>电子商务</系别>
    </学生>
</xml>
'内嵌的数据岛
<p>tagname的值<input type="text" size="10" name="t1"></p>
<p>text的值<input type="text" size="50" name="t2"></p>
<p><input type="button" value="show" name="show"></p>
</body>
</html>

```

它的执行结果如图 5-4 所示。

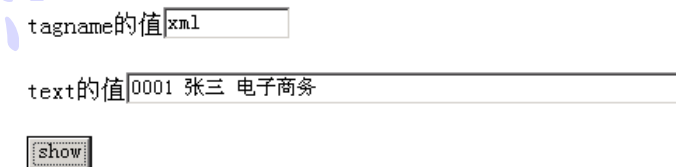


图 5-4 脚本语言访问数据岛

对于外嵌的 XML 数据岛,则可以利用 URL 方法来读取 XML 文件的 URL 路径和名称。还是用一个例子来说明。

先创建一个 XML 文件 student.xml, 清单如下。

程序清单 5.9:

```

<?xml version="1.0" encoding="gb2312"?>
<学生>
    <学号>0001</学号>
    <姓名>张三</姓名>
    <系别>电子商务</系别>
</学生>

```

然后创建一个外嵌的数据岛, 用脚本语言读取各个属性值。

程序清单 5.10:

```

<html>
<head>
    <title>
        数据岛对象
    </title>
    <script language="vbs">
        sub show_onclick()
            t1.value=student.tagname
            t2.value=student.text
            t3.value=student.url

```



```

        end sub
    </script>
</head>
<body>
    <xml id="student" src="student.xml"></xml>
    <p>tagname的值<input type="text" size="30" name="t1"></p>
    <p>text的值<input type="text" size="30" name="t2"></p>
    <p>url的值<input type="text" size="80" name="t3"></p>
    <p><input type="button" value="show" name="show"></p>
</body>
</html>

```

这段代码执行后的结果如图 5-5 所示。

图 5-5 URL 方法读取数据岛

5.4.2 节点的操作

根结点

可以使用 `DocumentElement` 属性来访问数据岛的根元素，下面先介绍一下几个常用的属性 `TagName`、`Text` 和 `GetAttribute`。

- ☑ `TagName` 方法得到标签的名称。
- ☑ `Text` 方法得到数据段落中除了标签名之外的所有数据。
- ☑ `GetAttribute` 方法得到节点的属性值，如果在节点上有多个属性，那么我们可以用 `Attributes.Item(index)` 方法来得到所需要的属性，当然编号从 0 开始。

下面就举个例子来说明这些方法的具体应用。

首先创建一个 XML 文件 `student1.xml`，清单如下。

程序清单 5.11:

```

<?xml version="1.0" encoding="gb2312"?>
<学生 性别="男" 籍贯="浙江">
    <学号>0001</学号>
    <姓名>张三</姓名>
    <系别>电子商务</系别>
</学生>

```

接下来创建一个外部 XML 数据岛，用上面介绍的方法来访问各个属性值。

程序清单 5.12:

```
<html>
  <head>
    <title>
      数据岛对象
    </title>
    <script language="vbs">
      sub show_onclick()
        t1.value=student.tagName
        t2.value=student.text
        t3.value=student.url
        t4.value=student.documentElement.Attributes.item(0).nodename
        t5.value=student.DocumentElement.Attributes.item(0).Text
        t6.value=student.documentElement.Attributes.item(1).nodename
        t7.value=student.DocumentElement.Attributes.item(1).Text
      end sub
    </script>
  </head>
  <body>
    <xml id="student" src="student.xml"></xml>
    <p>根元素tagName的值<input type="text" size="30" name="t1"></p>
    <p>根元素text的值<input type="text" size="30" name="t2"></p>
    <p>根元素url的值<input type="text" size="80" name="t3"></p>
    <p>根元素第一个属性名<input type="text" size="8" name="t4">
      值为<input type="text" size="8" name="t5"></p>
    <p>根元素第二个属性名<input type="text" size="8" name="t6">
      值为<input type="text" size="8" name="t7"></p>
    <p><input type="button" value="show" name="show"></p>
  </body>
</html>
```

这段代码的执行结果如图 5-6 所示。

根元素tagName的值

根元素text的值

根元素url的值

根元素第一个属性名 值为

根元素第二个属性名 值为

图 5-6 根节点访问实例

子节点

下面来看一下如何访问各个子节点。

可以用 `DocumentElement.ChildNodes.Item(Index)` 来访问各个子节点，子节点也有 `Tagname` 和 `Text` 两个属性可供访问。此外我们还可以用 `Length` 属性来获得 `Item` 集合的长度，也就是说子节点的数目。

还是用前面介绍过的 XML 文件 `student.xml`。

程序清单 5.13:

```
<?xml version="1.0" encoding="gb2312"?>
<学生>
  <学号>0001</学号>
  <姓名>张三</姓名>
  <系别>电子商务</系别>
</学生>
```

现在要把各个节点的名字和内容分别显示出来。

程序清单 5.14:

```
<html>
  <head>
    <title>
      数据岛对象
    </title>
    <script language="vbs">
      sub show_onclick()
        t1.value=student.documentElement.childNodes.item(0).Text
        t2.value=student.documentElement.childNodes.item(1).Text
        t3.value=student.documentElement.childNodes.item(2).Text
        t4.value=student.documentElement.childNodes.item(0).tagname
        t5.value=student.documentElement.childNodes.item(1).tagname
        t6.value=student.documentElement.childNodes.item(2).tagname
      end sub
    </script>
  </head>
  <body>
    <xml id="student" src="student.xml"></xml>
    <p><input type="text" size="30" name="t4">
      <input type="text" size="30" name="t1"></p>
    <p><input type="text" size="30" name="t5">
      <input type="text" size="30" name="t2"></p>
    <p><input type="text" size="30" name="t6">
      <input type="text" size="30" name="t3"></p>
    <p><input type="button" value="show" name="show"></p>
```

```
</body>
</html>
```

这段代码的显示结果如图 5-7 所示。

学号	0001
姓名	张三
系别	电子商务
<input type="button" value="show"/>	

图 5-7 子节点访问实例

5.4.3 其它显示方法

XSL 回顾

XSL 全名为 eXtensible StyleSheet Language，中文译为“可扩展样本语言”，主要是用来搭配 XML 文件使用的。因为 XML 文件中并不描述其内容如何被显示，所以 XML 文件可以调用一 XSL 文件，如此应用软件如浏览器就能根据 XSL 文件中的描述，将 XML 文件的内容显示在浏览器中。需要说明一点的是，目前只有 IE 5.5（以及更高的版本）支持对 XSL 的解析。

不管程序本身内含的技术是什么，对于用户而言，重要的是程序能够做什么。用户甚至不关心你采用的什么技术，所以在许多情况下，用户希望软件能够向他们显示用 XML 编码的信息。但是，XML 标记并没有提示如何在显示屏或页面上显示信息。

XML 实际上是通过内容做标记来描述其意义，从而使显示与内容相分离。目前，针对 XML 样式表开发的标准被称为可扩展样式表语言（Extensible Stylesheet Language），即 XSL。XSL 定义了 XML 的语法规则，该语法规则将被用来把 XML 文件转换成 HTML、XML 或其他格式的文档。一个 XSL 样式表集合了一系列设计规则，用于从 XML 文件中抽取信息，并将其转换成 HTML 等其他格式。这种转换采用了公开的方式，使其能够更加方便地被程序员描述。而且 XSL 还将提供多种脚本语言的通道，以满足更为复杂的应用需求。

XSL 能使 Web 浏览器直接根据用户的不同需求改变文档的显示法。例如，不需要与服务器进行交互通信，就可以改变数据的显示顺序。通过变换样式表，同一个文档可以显示得更大，或者经过折叠，只显示外面的一层，或者变为打印格式。比如为了方便残疾人阅读，样式表可以将 XML 译为盲文或可听见的语音，这项功能还给想在汽车里冲浪的人带来了好处：因为将页面转换为语音是非常方便。有几种 Web 浏览器的最新版本都能阅读。

XML 文档，并采用合适的样式表，在屏幕上对信息进行分类和格式化。基于 XML 的网站除了运行速度更快、更易使用外，读者可能根本不知道他所看到的是 XML 文档而不是 HTML 文档。

XSL 本身是一项 XML 应用。它直接架构在 XML 语法之上，分为两个部分：第一部分负责将 XML 的源代码转换为另一种 XSLT 格式；第二部分提供大量的格式化命令，可用

来配合印刷或屏幕显示，精确地设定外观样式格式 XSL-FO，这是一种独立于设备的格式。第一部分的转换语法可以用来服务于第二部分。

事实上，XSL 的转换语法并不限于将 XML 转换成 FO 命令，XSL 可以输出任何格式正确的 XML 文档。因为这个特性，可以用它来进行以下几种格式的转换：XML->HTML、XML->XML、XML->SVG 以及 XML->VRML 等。

同时需要注意的是，XML 样式语言还正在发展之中。正如 XML 介于 HTML 和 SGML 之间一样，XML 的样式表语言 XSL 标准也介于 CSS 和 DSSSL 之间。

XSL 在网络中的应用大体分为两种模式：

1. 服务器端转换模式

在这种模式下，XML 文件在被下载到浏览器前先转换成 HTML，然后再将 HTML 文件送往客户端进行浏览。有两种方式：

- ☒ 动态方式；即当服务器接到转换请求时再进行实时转换，这种方式对服务器要求较高。如果你的服务器性能较差的话，很可能造成系统拥挤现象。
- ☒ 批量方式；事先用 XSL 将一批 XML 转换成 HTML 文件，然后服务器接收到请求后调用转换好的 HTML 文件即可。实际上就是每天定时或系统空闲的时候在后台做转换，就象饭店每天事先作好的冷盘一样。

2. 客户端转换模式

这种方式将 XML 和 XSL 文件都传送到客户端，由浏览器实时转换。前提是浏览器必须支持 XML+XSL。当前流行的浏览器 IE 和 Netscape 都支持 XML+XSL。

前面已经提到，XSL 是由两种方式构成，一种方式用来转换 XML 文本内容，另一种则是格式化 XML 文本内容。我们可以通过 XSL 首先转换(过滤和整理)XML 数据内容，然后通过 XSL 的格式化显示方法定义数据内容显示方式(比如字体、大小、颜色等)。当然实际上也可以定义一个可以被浏览器支持的用来显示转换后的 XML 数据的方式，目前来讲这种方式往往是转换 XML 元素为 HTML 元素。

同时，XSL 也可以加入或者移动一些元素到输出文件，也可以重新整理排列这些元素，并且决定哪些元素将要被显示和如何被显示。

写程序的人都知道，理论说的再多，不如一个实例有效，好，现在就来举一个例子说明 XSL 在 XML 中的应用。

首先生成一个 XML 文档，把它命名为 nickmean.xml。

程序清单 5.15：

```
<?xml version="1.0" encoding="GB2312"?>
<?xml-stylesheet href="nickmean.xsl" type="text/xsl"?>
<网络用语集合>
<词语>
<名称>美眉</名称>
<含义>表示在网上横行的漂亮女孩子</含义>
</词语>
<词语>
<名称>恐龙</名称>
```

<含义>表示在网络上寻找另一个自我的表面张力和比例结构不友好的女孩子

</含义>

</词语>

</网络用语集合>

下面编写一个 XSL 文档来决定该 XML 文档的显示格式，该文档必须命名为 nickmean.xsl，也就是上面例子中<?xml-stylesheet href="nickmean.xsl" type="text/xsl"?>这一行指定的 href 的值。如下所示。

程序清单 5.16:

```
<?xml version="1.0" encoding="GB2312"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<xsl:apply-templates select="//网络用语集合/词语"/>
</body>
</html>
</xsl:template>
<xsl:template match="//网络用语集合/词语">
<DIV STYLE="color:blue">
<xsl:value-of select="名称"/>
</DIV>
<DIV STYLE="color:black">
<xsl:value-of select="含义"/>
</DIV>
</xsl:template>
</xsl:stylesheet>
```

如果希望以表格的形式显示结果的话，其 XSL 文档代码如下（当然该文档也需要命名为 nickmean.xsl）：

程序清单 5.17:

```
<?xml version="1.0" encoding="GB2312"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<xsl:apply-templates select="网络用语集合"/>
</body>
</html>
</xsl:template>
<xsl:template match="网络用语集合">
<table Border="1">
```

```

<xsl:for-each select="词语">
<tr>
<td><xsl:value-of select="名称"/></td>
<td><xsl:value-of select="含义"/></td>
</tr>
</xsl:for-each>
</table>
</xsl:template>
</xsl:stylesheet>

```

加载样式表

上一小节介绍了 XSL 的知识，可以看出，XSL 自己也是一种 XML 语法，因此数据岛也提供了一种方便的方法来加载样式表。可以在网页中同时加载 XML 源数据和 XSL 样式表。来看一个实例。

下面是要显示的 XML 源数据 stu.xml。

程序清单 5.18:

```

<?xml version = "1.0" encoding="GB2312"?>
<?xml-stylesheet type="text/xsl" href="stu.xsl"?>
<学生管理>
  <学生>
    <学号>97001</学号>
    <姓名>张三</姓名>
    <性别>男</性别>
    <籍贯>浙江</籍贯>
  </学生>
  <学生>
    <学号>97005</学号>
    <姓名>周俊</姓名>
    <性别>男</性别>
    <籍贯>浙江</籍贯>
  </学生>
  <学生>
    <学号>97007</学号>
    <姓名>王人水</姓名>
    <性别>女</性别>
    <籍贯>山东</籍贯>
  </学生>
  <学生>
    <学号>97008</学号>
    <姓名>欧阳紫荆</姓名>
    <性别>女</性别>
    <籍贯>广东</籍贯>
  </学生>

```



```
</学生>
</学生管理>
```

下面是用来显示该 XML 文件的 XSL 文件 stu.xsl。

程序清单 5.19:

```
<?xml version = "1.0" encoding="GB2312"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <HTML>
      <H1>学生信息管理</H1>
      <BODY COLOR="#0000A0">
        <TABLE BORDER="1">
          <TR>
            <TH>学号</TH>
            <TH>姓名</TH>
            <TH>性别</TH>
            <TH>籍贯</TH>
          </TR>
          <xsl:for-each select="学生管理/学生">
            <TR>
              <TD><xsl:value-of select="学号"/></TD>
              <TD><xsl:value-of select="姓名"/></TD>
              <TD><xsl:value-of select="性别"/></TD>
              <TD><xsl:value-of select="籍贯"/></TD>
            </TR>
          </xsl:for-each>
        </TABLE>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

使用如下的 HTML 文件同时加载这两个文件就可以显示该 XML 数据源了。

程序清单 5.20:

```
<HTML>
  <HEAD>
    <TITLE>数据岛的显示和转换</TITLE>
  </HEAD>
  <XML ID="source" SRC="stu.XML"></XML>
  <XML ID="style" SRC="stu.XSL"></XML>
  <SCRIPT FOR="window" EVENT="onload">
    xslTarget.innerHTML=source.transformNode(style.XMLDocument);
  </SCRIPT>
```

```
<BODY>
<div id="xslTarget"></div>
</BODY>
</HTML>
```

当网页网全下载以后，onload 事件被触发，transformNode 方法在基于 XML 的源数据上执行。transformNode 带有一个参数，它是一个描述 XSL 样式表的 DOM 节点，这个调用的返回值是一个 XML 文本。而 XSL 样式表产生了一个结构良好的 HTML 文档。

整个实例的显示结果如图 5-8 所示。

数据绑定

除此之外我们还可以用数据绑定的方法来显示数据岛文件。

还是显示上面介绍过的 XML 数据源 stu.xml，下面就是我们的程序清单 student.html。

程序清单 5.21:

```
<html>
<head>
<title>
数据岛的显示
</title>
</head>
<body>
<xml id="stu" src="stu.xml"></xml>
<h2>学生信息管理</h2>
<table DATASRC=#stu border="1">
<tr>
<td>
<div DATAFLD="学号">
</td>
<td>
<div DATAFLD="姓名">
</td>
<td>
<div DATAFLD="性别">
</td>
<td>
<div DATAFLD="籍贯">
</td>
</tr>
</table>
</body>
```

学生信息管理

学号	姓名	性别	籍贯
97001	张三	男	浙江
97005	周俊	男	浙江
97007	王人水	女	山东
97008	欧阳紫荆	女	广东

图 5-8 加载 XSL 的学生管理系统

```
</html>
```

显示结果如图 5-9 所示。

5.4.4 数据岛实例

在这里我们要应用前面介绍的数据岛知识来创建一个火车站的查询系统。

因为火车的信息可能时刻改变，比如国庆会添加专列，所以我们要创建的数据岛应该是外嵌的，我们把有关的火车信息存储在一个 XML 文件 train.xml 中，清单如下所示。

程序清单 5.22:

```
<?xml version="1.0" encoding="gb2312"?>
<trains>
  <train>
    <id>K222</id>
    <from>杭州</from>
    <to>金华</to>
    <lefttime>12: 23</lefttime>
  </train>
  <train>
    <id>k244</id>
    <from>杭州</from>
    <to>苏州</to>
    <lefttime>1: 20</lefttime>
  </train>
  <train>
    <id>k001</id>
    <from>杭州</from>
    <to>北京</to>
    <lefttime>23: 23</lefttime>
  </train>
  <train>
    <id>k345</id>
    <from>杭州</from>
    <to>广州</to>
    <lefttime>23: 12</lefttime>
  </train>
</trains>
```

首先在一个文本框中输入车次，然后用一个循环语句来判断各列火车的车次是否与之相符，如果是的话就跳出循环，用如下的代码显示该次列车的信息。

学生信息管理

97001	张三	男	浙江
97005	周俊	男	浙江
97007	王人水	女	山东
97008	欧阳紫荆	女	广东

图 5-9 数据绑定的学生管理系统

```
t1.value=train.documentElement.childNodes.item(i).childNodes.item(1).text
t2.value=train.documentElement.childNodes.item(i).childNodes.item(2).text
t3.value=train.documentElement.childNodes.item(i).childNodes.item(3).text
```

下面给出完整的程序清单 seek.html。

程序清单 5.23:

```
<html>
  <head>
    <title>
      查询系统
    </title>
    <style type="text/css">
<!--
p { font-family: "宋体"; font-size: 9pt;color: #0066cc; text-decoration: none}
h3 { font-family: "宋体"; font-size: 12pt; color: #0066cc; text-decoration: none}
-->
    </style>
    <script language="JavaScript">
      function loadform(){
        len=train.documentElement.childNodes.length
        ‘取得总的火车列数
        for (i=0;i<=len;i++)
          {
            if (train.documentElement.childNodes.item(i).childNodes.item(0).text==t.value)
              break;
          }
        t4.value=i;
        t1.value=train.documentElement.childNodes.item(i).childNodes.item(1).text
        t2.value=train.documentElement.childNodes.item(i).childNodes.item(2).text
        t3.value=train.documentElement.childNodes.item(i).childNodes.item(3).text
      }
    </script>
  </head>
  <body>
    <xml id="train" src="train.xml"></xml>
    <h3>杭州火车站查询系统</h3>
    <p>车次: <input type="text" size="10" name="t">
      <input type="submit" value="提交" name="ok" onclick="loadform()">
      <input type="reset" value="清空"></p>
    <hr>
    <p>起点站: <input type="text" size="30" name="t1"></p>
    <p>终点站: <input type="text" size="30" name="t2"></p>
    <p>发车时间: <input type="text" size="30" name="t3"></p>
```

```
<input name="t4" type="hidden">
</body>
</html>
```

该程序的运行结果如图 5-10 所示。

杭州火车站查询系统

车次:

起点站:

终点站:

发车时间:

图 5.10 火车站查询系统

5.5 小 结

通过上面的学习,可以看到 DOM 为访问和操作 XML 文档的各个节点提供了良好的机制,所以我们应该掌握 DOM 的几个主要接口的属性和方法,可以用 DOM 熟练地对 XML 文档进行访问和操作。此外我们还介绍了数据岛技术,并且因为本书同时介绍了 XSL 样式表的相关知识。

下一章将给出三个详细的实例。

第 6 章 DOM 应用实例

本章导读

第 5 章深入介绍了 DOM 接口的知识，同时还详细介绍了 XML 数据岛的有关概念。对于要从事 XML 高级开发的技术人员来说，DOM 是必须掌握的一种技术。DOM 提供了良好的机制来对 XML 文档中的节点进行访问和操作，而且在很大的程度上保证了各个平台之间的互操作性。

本章就在所学知识的基础上，给出三个较大的实例。

6.1 投票系统

现在在每个网站都能看到投票系统。这些网站基本上都是用 Access 之类的数据库来存储投票后的数据，然后再显示的。这里我们要用 XML 文件来存储数据，然后再用 DOM 对数据进行操作和显示。

下面介绍的这个例子是网站的投票系统，有四个选项，分别是相当好、还可以、很一般和太烂了。网站的访问者可以选择自己的看法，投票之后，结果就显示在当前的网页上。

首先，要把信息存储在一个 XML 文件 list.xml 之中，清单如下，其中 a 表示相当好、b 表示还可以、c 表示很一般而 d 则表示太烂了。

程序清单 6.1:

```
<?xml version="1.0" encoding="gb2312"?>
<newlist>
  <list>
    <a>101</a>
    <b>190</b>
    <c>66</c>
    <d>17</d>
  </list>
</newlist>
```

下一步，我们就要对这个 XML 文件进行操作了。如下的代码加载了该文件。

```
strSourceFile = "c:\new\list.xml"
Set objXML = Server.CreateObject("Microsoft.XMLDOM")
objXML.load(strSourceFile)
```

然后需要读取节点：

```
Set rootNode = objXML.documentElement
Set objRootsite = objXML.getElementsByTagName("list")
```

‘得到标签名为“list”的节点

```
a=objRootsite.item(0).childnodes.item(1).text
```

```
b= objRootsite.item(0).childnodes.item(3).text
```

```
c=objRootsite.item(0).childnodes.item(5).text
```

```
d=objRootsite.item(0).childnodes.item(7).text
```

‘分别读取a、b、c、d各个节点的数据，得到的结果是以字符串表示的，存储在四个变量a、b、c、d中。

接下来要做的就是结合 asp 来对访问者的投票来进行处理。

如下的代码根据客户端返回的信息，对投票者的选项进行了判断。在相应的变量上加 1，然后在 DOM 树中修改相应的节点信息。

```
<%
vote = Request("vote")
If vote <> Empty then
    Select case vote
    Case "相当好"
        a=a+1
        objRootsite.item(0).childnodes.item(1).text=a
    Case "还可以"
        b=b+1
        objRootsite.item(0).childnodes.item(3).text=b
    Case "很一般"
        c=c+1
        objRootsite.item(0).childnodes.item(5).text=c
    Case "太烂了"
        d=d+1
        objRootsite.item(0).childnodes.item(7).text=d
    End Select
End if
%>
```

由于 DOM 树是保存在内存中的，所以我们需要将修改过的 XML 文件保存在硬盘上。

```
objXML.save(strSourceFile)
```

接下来统计投票的总数，并且对各个选项的得票率进行统计。

```
<%
Total = CInt(a)+CInt(b)+CInt(c)+CInt(d)
If Total > 0 Then
    ' 防止出现除以 0 的错误
    per1 = a / Total
    per2 = b / Total
    per3 = c / Total
```

```

    per4 = d / Total
End If
%>

```

这里需要注意的一点是统计投票总数时不能用如下的代码：

```
Total = a+b+c+d
```

因为这样就成了字符串之间的连接运算了，也就是说，当 a, b, c, d 分别为 101, 190, 66 和 17 的时候，得到的结果就成了 1011906617 了。所以先要用 CInt 方法来把字符串转换为整数类型数据。

再用如下的代码把得到的投票率在表格中显示。

```

</TR>
  <TD width="68">得票率</TD>
  <TD> <%=FormatNumber(per1*100, 2, True)%>%</TD>
  <TD> <%=FormatNumber(per2*100, 2, True)%>%</TD>
  <TD> <%=FormatNumber(per3*100, 2, True)%>%</TD>
  <TD> <%=FormatNumber(per4*100, 2, True)%>%</TD>
</TR>

```

为了更形象地显示各个选项的得票率，所以引进了如下的代码，用图形来表示。

```

<TR BgColor=Yellow HEIGHT=250 ALIGN=BOTTOM valign="bottom">
  <TD >图示</TD>
  <TD width="68" align="CENTER" >
    <IMG SRC= pic.gif width=50 height=<%=per1*400%> >
  </TD>
  <TD width="68" align="CENTER" >
    <IMG SRC=pic.gif width=50 height=<%=per2*400%> >
  </TD>
  <TD width="68" align="CENTER" >
    <IMG SRC=PIC.gif width=50 height=<%=per3*400%> >
  </TD>
  <TD width="68" align="CENTER" >
    <IMG SRC=PIC.gif width=50 height=<%=per4*400%> >
  </TD>
</TR>

```

在这里根据得票率对一张图片的显示高度分别进行了缩放，以达到所求的显示效果。到此为止，我们就完成了这个投票系统的设计。当在服务端运行它的时候就可以进行工作了，它的运行结果，如图 6-1 所示。

您对本站的想法如何

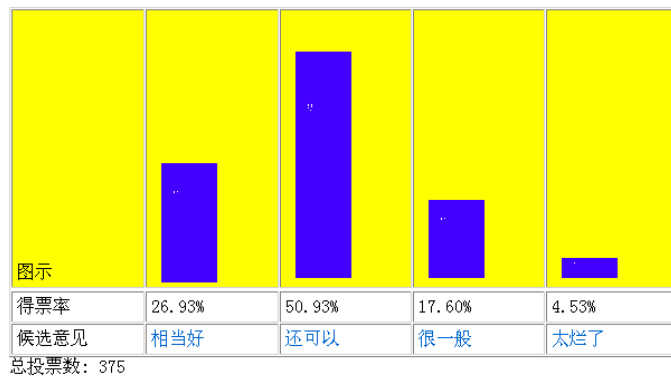


图 6-1 网站投票系统

下面给出整个系统的程序清单 vote.asp，希望大家仔细琢磨。

程序清单 6.2:

```
<HTML>
<head>
<title>
    网站投票
</title>
<style type="text/css">
    <!--
    td { font-family: "宋体"; font-size: 12pt; text-decoration: none}
    a { font-family: "宋体"; font-size: 12pt; color: #0066CC; text-decoration: none}
    a:hover { color: #FF6600; text-decoration: underline}
    -->
</style>

<%
strSourceFile = "c:\new\list.xml"
Set objXML = Server.CreateObject("Microsoft.XMLDOM")
objXML.load(strSourceFile)
Set rootNode = objXML.documentElement
Set objRootsite = objXML.getElementsByTagName("list")

a=objRootsite.item(0).childNodes.item(1).text
b = objRootsite.item(0).childNodes.item(3).text
c=objRootsite.item(0).childNodes.item(5).text
d=objRootsite.item(0).childNodes.item(7).text
%>

<%
```

```

vote = Request("vote")
If vote <> Empty then
    Select case vote
        Case "相当好"
            a=a+1
            objRootsite.item(0).childnodes.item(1).text=a
        Case "还可以"
            b=b+1
            objRootsite.item(0).childnodes.item(3).text=b
        Case "很一般"
            c=c+1
            objRootsite.item(0).childnodes.item(5).text=c
        Case "太烂了"
            d=d+1
            objRootsite.item(0).childnodes.item(7).text=d
    End Select
    objXML.save(strSourceFile)
End if
Total = CInt(a)+CInt(b)+CInt(c)+CInt(d)
If Total > 0 Then
    per1 = a / Total
    per2 = b / Total
    per3 = c / Total
    per4 = d / Total
End If
%>

<h2> 您对本站的想法如何 </h2>
<BLOCKQUOTE>
    <TABLE Border=1 Cellpadding=4 Cellspacing=1 width="600">
        <TR BgColor=Yellow HEIGHT=250 ALIGN=BOTTOM valign="bottom">
            <TD >图示</TD>
            <TD width="68" align="CENTER" >
                <IMG SRC= pic.gif width=50 height=<%=per1*400%> >
            </TD>
            <TD width="68" align="CENTER" >
                <IMG SRC=pic.gif width=50 height=<%=per2*400%> >
            </TD>
            <TD width="68" align="CENTER" >
                <IMG SRC=pic.gif width=50 height=<%=per3*400%> >
            </TD>
            <TD width="68" align="CENTER" >
                <IMG SRC=pic.gif width=50 height=<%=per4*400%> >
            </TD>
        </TR>
    </TABLE>
</BLOCKQUOTE>

```

```

</TD>
</TR>
<TR>
  <TD width="68">得票率</TD>
  <TD> <%=FormatNumber(per1*100, 2, True)%>%</TD>
  <TD> <%=FormatNumber(per2*100, 2, True)%>%</TD>
  <TD> <%=FormatNumber(per3*100, 2, True)%>%</TD>
  <TD> <%=FormatNumber(per4*100, 2, True)%>%</TD>
</TR>
<TR>
  <TD width="68">候选意见</TD>
  <TD> <A HREF="<%=Myself%>?vote=相当好">相当好</A> </TD>
  <TD> <A HREF="<%=Myself%>?vote=还可以">还可以</A> </TD>
  <TD> <A HREF="<%=Myself%>?vote=很一般">很一般</A> </TD>
  <TD> <A HREF="<%=Myself%>?vote=太烂了">太烂了</A> </TD>
</TR>
</TABLE>
总投票数: <%=Total%>
</BLOCKQUOTE>
</BODY>
</HTML>

```

6.2 留言本

留言本对大家来说一定不陌生。下面我们就用 ASP 和 DOM 来编写一个留言本，基本思想是用 ASP 和 DOM 来读取和存储 XML 数据，并利用 XML 数据来存储留言信息，达到和使用数据库存储数据一样的功能。与前面介绍的投票系统有所区别，这个例子使用了 XSL 样式表来对 XML 文件进行显示，而不是 HTML 的表格。

由于 XML 数据的跨平台性，只要把这些数据存储为 XML，那么这些数据就能被任何其它语言或系统所识别，而不用做任何改动。这显然是别的数据库所不能实现的功能。比如我们在网上的留言信息可以直接被转换成 wap 格式在手机上显示。

和前面介绍的投票系统一样，首先需要创建一个 XML 文件 book.xml 来存储信息。我们需要保存的数据有时间、姓名、E-mail、主页、地址和留言内容，因此使用如下的 XML 文件来存储信息。

程序清单 6.3:

```

<guestbook>
  <book date="2001-9-20 13:49:19">
    <name>翁宇翔</name>
    <email>wengyux_cn@sina.com</email>
    <homepage>http://night507.myetang.com</homepage>
    <adress>浙江瑞安</adress>
    <message>

```

```
<![CDATA[这个留言本用ASP和DOM来读取和存储XML数据，  
并利用XML数据来存储留言信息，  
达到和用数据库存储数据一样的功能，  
欢迎大家使用!!!]]>
```

```
</message>
```

```
</book>
```

```
<book date="2001-9-22 20:30:44">
```

```
<name>芮云</name>
```

```
<email>nani@sina.com</email>
```

```
<homepage>http://</homepage>
```

```
<adress>浙江杭州</adress>
```

```
<message>
```

```
<![CDATA[恩，  
不错，我喜欢这个留言本  
而且速度还很快啊!! ]]>
```

```
</message>
```

```
</book>
```

```
</guestbook>
```

为了保持留言的格式，这里使用了<![CDATA[]]>来保存留言信息。
下一步要做的就是加载该 XML 文件，代码如下所示。

```
Dim doc
```

```
Set doc = Server.CreateObject("Microsoft.XMLDOM")
```

```
doc.async = False
```

```
doc.Load Server.MapPath("book.xml")
```

在这个例子中我们尝试用 XSL 来显示该 XML 文件。

```
Dim xslDoc
```

```
Set xslDoc = Server.CreateObject("Microsoft.XMLDOM")
```

```
xslDoc.async = False
```

```
xslDoc.load Server.MapPath("book.xml")
```

```
Response.Write doc.TransformNode(xslDoc)
```

因此需要建立一个 xsl 文件，在这个文件中显示了 XML 文件的各条信息。

程序清单 6.4:

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

```
<xsl:template match="/">
```

```
<table border = "1" width="100%" bgcolor="#bbCCCC">
```

```
<tr bgcolor="#0099CC">
```

```
<td>
```

```
最近留言||<a href="index.html">我要留言</a>
```

```
</td>
```

```

</tr>
<tr>
<table width="100%" border="1">
<xsl:for-each select="guestbook/book" order-by="-book[@date]">
<tr bgcolor="#bbCC00">
    <td rowspan="2" width="20%">
        留言人: <xsl:value-of select="name"/><p/>
        来自: <xsl:value-of select="adress"/>
    </td>
    <td width="80%" >
        留言时间: <xsl:value-of select="@date"/>||
        Email: <xsl:value-of select="email"/>||
        个人主页: <xsl:value-of select="homepage"/>
    </td>
</tr>
<tr>
    <td width="72%">
        <pre><xsl:value-of select="message"/></pre>
    </td>
</tr>
</xsl:for-each>
</table>
</tr>
</table>
</xsl:template>
</xsl:stylesheet>

```

到这里我们就完成了 XML 文件的显示, 也就是说完成了留言本显示的功能, 显示的结果如图 6-2 所示, 接下来探讨如何添加新留言。

留言本	
最近留言 我要留言	
留言人: 芮云 来自: 浙江杭州	留言时间: 2001-9-22 20:30:44 Email: nani@sina.com 个人主页: http:// 恩, 不错, 我喜欢这个留言本 而且速度还很快啊!!
留言人: 翁宇翔 来自: 浙江瑞安	留言时间: 2001-9-20 13:49:19 Email: wengyux_cr@sina.com 个人主页: http://night507.myetang.com 这个留言本用ASP和DOM来读取和存储XML数据, 并利用XML数据库来存储留言信息, 达到和用数据库存储数据一样的功能, 欢迎大家使用!!!

图 6-2 留言本

首先需要创建一个留言的界面, 下面是它的部分 HTML 代码。

```

<form method=post action="opreate.asp">
<table border = 1>
<tr>
  <td>
    <table width=500  cellspacing =3 cellpadding=3 bgcolor="#bbcccc">
      <tr>
        <td align=right>姓名:</td>
        <td><input type=text name="name" maxlength=30></td>
      </tr>
      <tr>
        <td align=right>email:</td>
        <td><input type=text name="email" maxlength=50></td>
      </tr>
      <tr>
        <td align=right>主页:</td>
        <td>
          <input type=text name="homepage" value="http://" maxlength=75>
        </td>
      </tr>
      <tr>
        <td align=right>地址:</td>
        <td><input type=text name="adress" maxlength=15></td>
      </tr>
      <tr>
        <td align=right valign=top>留言</td>
        <td><textarea name="message" cols=50 rows=5></textarea></td>
      <tr align="center">
        <td><input type=submit value="提交">
        <input type=reset value="清空"></td>
      </tr>
    </table>
  </td>
</tr>
</table>

```

显示效果如图 6-3 所示。

图 6-3 添加留言页面

完成了留言本的界面之后，就可以通过对 DOM 树的节点的操作来添加新的留言。首先读取客户端传送过来的各个参数，并记录在各个变量之中。

```
theName=Request("name")
theEmail=Request("email")
theHomePage=Request("homepage")
theadress=Request("adress")
themessage=Request("message")
```

然后加载该 XML 文件

```
Set theDOM = Server.CreateObject("Microsoft.XMLDOM")
theDOM.async = false
theDOM.load server.mappath("book.xml")
```

在根节点上添加子节点 book，并且设置属性 date，其值默认为系统时间。

```
Set thebookNode =
    theDOM.documentElement.AppendChild(theDOM.createElement("book"))
thebookNode.setAttribute "date", now()
```

依次创建 book 节点的各个子节点，它们的值为客户端返回的参数，并把他们作为 book 节点的子节点用 appendChild 方法添加到文档中。

```
Set thechildNode = thebookNode.appendChild(theDOM.createElement("name"))
thechildNode.Text = theName
Set thechildNode = thebookNode.appendChild(theDOM.createElement("email"))
thechildNode.Text = theEmail
Set thechildNode = thebookNode.appendChild(theDOM.createElement("date"))
thechildNode.Text = Now()
```

```

Set thechildNode = thebookNode.appendChild(theDOM.createElement("homepage"))
thechildNode.Text = theHomePage
Set thechildNode = thebookNode.appendChild(theDOM.createElement("adress"))
thechildNode.Text = theadress
Set thechildNode = thebookNode.appendChild(theDOM.createElement("message"))
thechildNode.appendChild theDOM.createCDATASection(themessage)

```

最后我们保存修改后的 XML 文档。

```
theDOM.Save Server.MapPath("book.xml")
```

至此，我们就完成了留言本的设计，下面给出所有的程序清单。

程序清单 6.5: index.html

```

<html>
<head>
<style type="text/css">
<!--
td { font-family: "宋体"; font-size: 9pt;color: #0066cc; text-decoration: none}
a { font-family: "宋体"; font-size: 9pt; color: #0066cc; text-decoration: none}
a:hover { color: #ff6600; text-decoration: underline}
-->
</style>

<title>留言本</title>
</head>
<body bgcolor="#cccc66" text="#000000" link="#0000ff" alink="#0000ff" vlink="#0000ff">

<center>

<h1>留言本</h1>
<a href="viewguestbook.asp">观看留言</a>
<p>
<form method=post action="opreate.asp">
    <table border = 1>
        <tr>
            <td>
                <table width=500   cellspacing =3 cellpadding=3 bgcolor="#bbcccc">
                    <tr>
                        <td align=right>姓名:</td>
                        <td ><input type=text name="name" maxlength=30></td>
                    </tr>
                    <tr>
                        <td align=right>email:</td>

```



```

        <td><input type=text name="email" maxlength=50></td>
    </tr>
    <tr>
        <td align=right>主页:</td>
        <td><input type=text name="homepage" value="http://" maxlength=75></td>
    </tr>
    <tr>
        <td align=right>地址:</td>
        <td><input type=text name="adress" maxlength=15></td>
    </tr>
    <tr>
        <td align=right valign=top>留言</td>
        <td><textarea name="message" cols=50 rows=5></textarea></td>
    <tr align="center">
        <td></td><td><input type=submit value="提交">
            <input type=reset value="清空"></td>
    </tr>
</table>
</tr>
</table>
</form>
</center>
</body>
</html>

```

程序清单 6.6: operate.asp

```

<html>
<%
    Dim theName
    Dim theEmail
    Dim theHomePage
    Dim theadress
    Dim themessage
    Dim theDOM
    Dim theRootNode
    Dim thebookNode
    Dim thechildNode

    theName=Request("NAME")
    theEmail=Request("EMAIL")
    theHomePage=Request("HOMEPAGE")
    theadress=Request("adress")
    themessage=Request("MESSAGE")

```

```
Set theDOM = Server.CreateObject("Microsoft.XMLDOM")
theDOM.async = false
theDOM.load server.mappath("book.xml")

If theDOM.parseError.ErrorCode <> 0 Then ' not found! create an empty document
    theDOM.loadXML "<guestbook/>"
End If

Set thebookNode = theDOM.documentElement.AppendChild(theDOM.createElement("book"))
thebookNode.setAttribute "date", now()

Set thechildNode = thebookNode.appendChild(theDOM.createElement("name"))
thechildNode.Text = theName

Set thechildNode = thebookNode.appendChild(theDOM.createElement("email"))
thechildNode.Text = theEmail

Set thechildNode = thebookNode.appendChild(theDOM.createElement("homepage"))
thechildNode.Text = theHomePage

Set thechildNode = thebookNode.appendChild(theDOM.createElement("adress"))
thechildNode.Text = theadress

Set thechildNode = thebookNode.appendChild(theDOM.createElement("message"))
thechildNode.appendChild theDOM.createCDATASection(themessage)
%>
<h1> 感谢你的留言</h1>
<pre>
<%= themessage %>
</pre>
<p> <a href="viewguestbook.asp">回到留言本</a> </p>
</html>
```

程序清单 6.7: book.xml

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body bgcolor="#CCCC66">
<center>
<h1>留言本</h1>
</center>
```

```

<table border = "1" width="100%" bgcolor="#bbCCCC">
  <tr bgcolor="#0099CC">
    <td>最近留言||
      <a href="index.html">我要留言</a>
    </td>
  </tr>
  <tr>
    <table width="100%" border="1">
      <xsl:for-each select="guestbook/book" order-by="-book[@date]">
        <tr bgcolor="#bbCC00">
          <td rowspan="2" width="20%">
            留言人: <xsl:value-of select="name"/><p/>
            来自: <xsl:value-of select="adress"/></td>
          <td width="80%">
            留言时间: <xsl:value-of select="@date"/>||
            Email: <xsl:value-of select="email"/>||
            个人主页: <xsl:value-of select="homepage"/>
          </td>
        </tr>
        <tr>
          <td width="72%"><pre><xsl:value-of select="message"/></pre></td>
        </tr>
      </xsl:for-each>
    </table>
  </tr>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

程序清单 6.8: viewguestbook.asp

```

<%
  Dim doc
  Dim xsl doc

  Set doc = Server.CreateObject("Microsoft.XMLDOM")
  doc.async = False
  doc.Load Server.MapPath("book.xml")
  Set xsl doc = Server.CreateObject("Microsoft.XMLDOM")
  xsl doc.async = False
  xsl doc.load Server.MapPath("book.xsl")
  Response.Write doc.TransformNode(xsl doc)

```

%>

6.3 网址及短消息管理器

这个管理器大家肯定比较陌生，它主要是配合网站管理员管理网站工作用的，例如编辑一些常用的网址和常用语，以便引导浏览者和与之交流。

程序是基于 MFC 的文档和视图开发的，程序中用到了微软的 msxml 的几乎所有接口函数，包括遍历、修改、增加文档的元素、属性、元素值、属性值等。

最后程序运行的效果图如图 6-4:

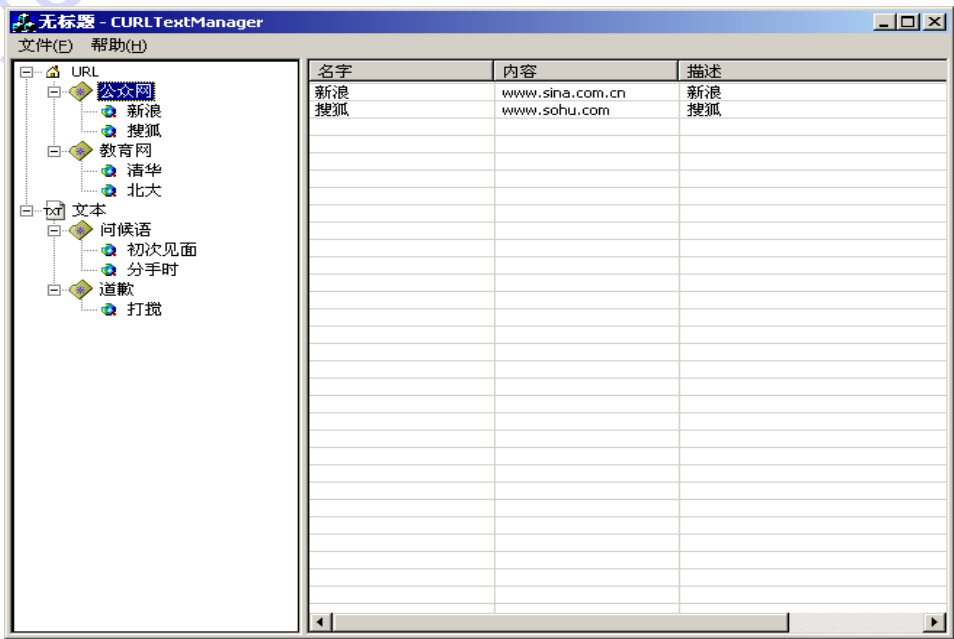


图 6-4 网址及短消息管理器效果

在左边的树视图中通过点击鼠标右键来激活右键菜单可以添加删除组（Group），添加删除某一条信息（Item）。选中组这一级时，右边的列表视就会列出该组下所有的信息条目。在右边的视图中也可以通过右键菜单删除选中的条目。

其实本程序的思路很简单，先定义xml文档保存信息。本程序的xml文档结构清晰，主要包括网址URLs和常用语Texts两部分，而每一部分又可以分成不同的组（Group）。从主窗口也可以比较清晰的看见文档的结构。源代码如下所示：

程序清单 6.9:

```
<?xml version="1.0" encoding="gb2312"?>
<Root>
  <URLs>
    <Group Name="公众网">
      <Item>
        <Name>新浪</Name>
        <Body>www.sina.com.cn</Body>
```

```
<Comment>新浪</Comment>
</Item>
<Item>
  <Name>搜狐</Name>
  <Body>www.sohu.com</Body>
  <Comment>搜狐</Comment>
</Item>
</Group>
<Group Name="教育网">
  <Item>
    <Name>清华</Name>
    <Body>www.tsinghua.edu.cn</Body>
    <Comment>清华</Comment>
  </Item>
  <Item>
    <Name>北大</Name>
    <Body>www.pku.edu.cn</Body>
    <Comment>北大</Comment>
  </Item>
</Group>
</URLs>
<Texts>
  <Group Name="问候语">
    <Item>
      <Name>初次见面</Name>
      <Body>gald to meet to you</Body>
      <Comment>初次见面</Comment>
    </Item>
    <Item>
      <Name>分手时</Name>
      <Body>good bye</Body>
      <Comment>分手时</Comment>
    </Item>
  </Group>
  <Group Name="道歉">
    <Item>
      <Name>打搅</Name>
      <Body>execuse me</Body>
      <Comment>打搅</Comment>
    </Item>
  </Group>
</Texts>
</Root>
```

然后就是将该 xml 文档解析到左边的树视图中，这些工作主要在 CdirectoryView 中完成的。程序流程图如下所示：

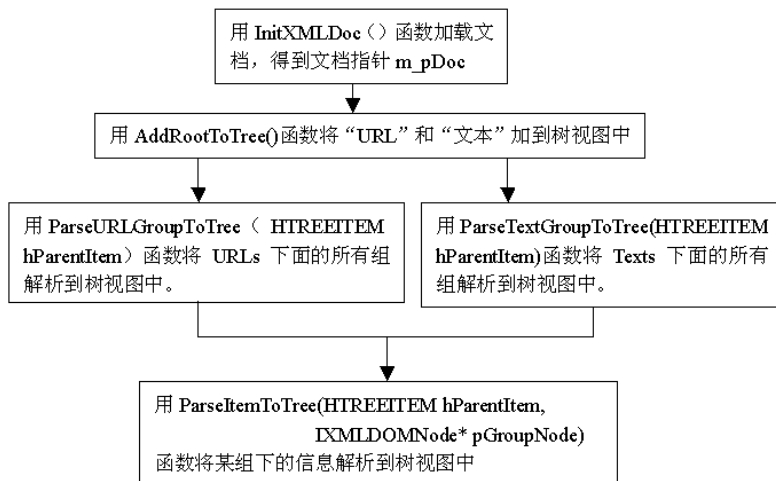


图 6-5 程序流程图

本程序中主要的类及其主要的函数解释如下：

6.3.1 ParseXMLDOM 类

下面是编者自己写的解析文档的一些函数，方便在其他类中调用。

(1) 函数：GetNodeAttributeValue(CString strAttrName, IXMLDOMNode *pNode)

功能：得到节点的某个属性的值

参数：strAttrName 属性名

pNode 节点指针

代码如下：

```

CString GetNodeAttributeValue(CString strAttrName, IXMLDOMNode *pNode)
{
    IXMLDOMNamedNodeMap* pAttrMap;//节点的属性链表
    pNode->get_attributes(&pAttrMap);
    IXMLDOMNode* pAttrNode;//节点的属性节点
    BSTR bsIDText = strAttrName.AllocSysString();
    pAttrMap->getNamedItem(bsIDText, &pAttrNode);
    pAttrNode->get_text(&bsIDText);
    return CString(bsIDText);
}
  
```

先用 get_attribute 得到元素节点的属性集合，然后用 getNamedItem 根据属性名得到该集合中某一属性指针，最后用 get_text 就得到该属性的属性值了。

(2) 函数：GetNodeByElement(IXMLDOMELEMENT *pElement)

功能：根据元素指针得到相应的节点指针

参数: pElement 元素指针

代码如下:

```
IXMLDOMNode* GetNodeByElement(
    IXMLDOMElement *pElement)
{
    IXMLDOMNode* pNode = NULL;
    pElement->QueryInterface(IID_IXMLDOMNode,(void**)&pNode);
    return pNode;
}
```

(3) 函数: GetElementByNode(IXMLDOMNode *pNode)

功能: 根据节点指针得到相应的元素指针

参数: pNode 元素指针

代码如下:

```
IXMLDOMElement* GetElementByNode(IXMLDOMNode *pNode)
{
    IXMLDOMElement* pElement = NULL;
    pNode->QueryInterface(IID_IXMLDOMElement,(void**)&pElement);
    return pElement;
}
```

(4) 函数: GetSingleChildNodeText(CString strChildBaseName,IXMLDOMNode* pParentNode)

功能: 得到某父节点下的唯一子节点的内容

参数: strChildBaseName 唯一子节点名

pParentNode 父节点

代码如下:

```
CString GetSingleChildNodeText(
    CString strChildBaseName,IXMLDOMNode* pParentNode)
{
    IXMLDOMNode* pSingleChildNode;
    BSTR bsChildBaseName = strChildBaseName.AllocSysString();
    pParentNode->selectSingleNode(bsChildBaseName,&pSingleChildNode);
    BSTR bsText;
    pSingleChildNode->get_text(&bsText);
    return CString(bsText);
}
```

先用 selectSingleNode 函数得到这个唯一子节点,然后用 get_text 得到这个子节点的值。

(5) 函数: GetSingleNodeBaseName(IXMLDOMNode* pNode)

功能: 得到节点名

参数: pNode 节点指针

代码如下:

```
CString GetSingleNodeBaseName(IXMLDOMNode* pNode)
```

```

{
    BSTR bsBaseName;
    pNode->get_baseName(&bsBaseName);
    return CString(bsBaseName);
}

```

(6) 函数: GetSingleNodeText(IXMLDOMNode* pNode)

功能: 得到节点的内容

参数: pNode 节点指针

代码如下:

```

CString GetSingleNodeText(IXMLDOMNode* pNode)
{
    BSTR bsText;
    pNode->get_text(&bsText);
    return CString(bsText);
}

```

(7) 函数: SetNodeAttributeValue(CString strAttrName, CString strAttrValue, IXMLDOMNode* pNode)

功能: 给节点的某个属性赋值

参数: strAttrName 属性名
 strAttrValue 属性值
 pNode 节点指针

代码如下:

```

void SetNodeAttributeValue(
    CString strAttrName, CString strAttrValue, IXMLDOMNode* pNode)
{
    IXMLDOMNamedNodeMap* pAttrMap; // 节点的属性集合
    pNode->get_attributes(&pAttrMap);
    IXMLDOMNode* pAttrNode; // 节点的属性节点
    pAttrMap->getNamedItem(strAttrName.AllocSysString(), &pAttrNode);
    pAttrNode->put_text(strAttrValue.AllocSysString());
}

```

先用 get_attributes 和 getNamedItem 得到要赋值的属性指针, 然后用 put_text 为该属性赋值。

该类的完整代码如下:

程序清单 6.10:

```

// ParseXMLDOM.h: interface for the CParseXMLDOM class.
class CParseXMLDOM
{
public:

    IXMLDOMElement* GetElementByNode(IXMLDOMNode* pNode);

```



```

IXMLDOMNode* GetNodeByElement(IXMLDOMElement* pElement);
CString GetElementAttributeValue(CString strAttrName,
                                IXMLDOMElement* pElement);
CString GetNodeAttributeValue(CString strAttrName,
                              IXMLDOMNode* pNode);
CString GetSingleChildNodeText(CString strChildBaseName,
                              IXMLDOMNode* pParentNode);
CString GetSingleNodeBaseName(IXMLDOMNode* pNode);
CString GetSingleNodeText(IXMLDOMNode* pNode);
void SetNodeAttributeValue(CString strAttrName,
                           CString strAttrValue,
                           IXMLDOMNode* pNode);
void SetSingleChildNodeText(CString strChildBaseName,
                            CString strValue,
                            IXMLDOMNode* pParentNode);

CParseXMLDOM();
virtual ~CParseXMLDOM();

};

```

// ParseXMLDOM.cpp: implementation of the CParseXMLDOM class.

```

//
////////////////////////////////////

```

```

#include "stdafx.h"
#include "CURLTextManager.h"
#include "ParseXMLDOM.h"

```

```

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

```

```

////////////////////////////////////
// Construction/Destruction
////////////////////////////////////

```

```

CParseXMLDOM::CParseXMLDOM()
{

```

```
}

CParseXMLDOM::~CParseXMLDOM()
{

}

CString CParseXMLDOM::GetNodeAttributeValue(CString strAttrName,
                                             IXMLDOMNode *pNode)
{
    IXMLDOMNamedNodeMap* pAttrMap;//节点的属性链表
    pNode->get_attributes(&pAttrMap);
    IXMLDOMNode* pAttrNode;//节点的属性节点
    BSTR bsIDText = strAttrName.AllocSysString();
    pAttrMap->getNamedItem(bsIDText,&pAttrNode);
    pAttrNode->get_text(&bsIDText);
    return CString(bsIDText);
}

CString CParseXMLDOM::GetElementAttributeValue(CString strAttrName,
                                             IXMLDOMElement *pElement)
{
    IXMLDOMNode* pNode = GetNodeByElement(pElement);
    return GetNodeAttributeValue(strAttrName,pNode);
}

XMLDOMNode* CParseXMLDOM::GetNodeByElement(IXMLDOMElement *pElement)
{
    IXMLDOMNode* pNode = NULL;
    pElement->QueryInterface(IID_XMLDOMNode,(void**)&pNode);
    return pNode;
}

XMLDOMElement* CParseXMLDOM::GetElementByNode(IXMLDOMNode *pNode)
{
    IXMLDOMElement* pElement = NULL;
    pNode->QueryInterface(IID_XMLDOMElement,(void**)&pElement);
    return pElement;
}

CString CParseXMLDOM::GetSingleChildNodeText(CString strChildBaseName,
                                             IXMLDOMNode* pParentNode)
{

```

```

        IXMLDOMNode* pSingleChildNode;
        BSTR bsChildBaseName = strChildBaseName.AllocSysString();
        pParentNode->selectSingleNode(bsChildBaseName,&pSingleChildNode);
        BSTR bsText;
        pSingleChildNode->get_text(&bsText);
        return CString(bsText);
    }
    CString CParseXMLDOM::GetSingleNodeBaseName(IXMLDOMNode* pNode)
    {
        BSTR bsBaseName;
        pNode->get_baseName(&bsBaseName);
        return CString(bsBaseName);
    }
    CString CParseXMLDOM::GetSingleNodeText(IXMLDOMNode* pNode)
    {
        BSTR bsText;
        pNode->get_text(&bsText);
        return CString(bsText);
    }

    void CParseXMLDOM::SetNodeAttributeValue(CString strAttrName,
        CString strAttrValue,
        IXMLDOMNode* pNode)
    {
        IXMLDOMNamedNodeMap* pAttrMap;//节点的属性链表
        pNode->get_attributes(&pAttrMap);
        IXMLDOMNode* pAttrNode;//节点的属性节点
        pAttrMap->getNamedItem(strAttrName.AllocSysString(),&pAttrNode);
        pAttrNode->put_text(strAttrValue.AllocSysString());
    }

    void CParseXMLDOM::SetSingleChildNodeText(CString strChildBaseName,
        CString strValue,
        IXMLDOMNode* pParentNode)
    {
        IXMLDOMNode* pSingleChildNode;
        BSTR bsChildBaseName = strChildBaseName.AllocSysString();
        pParentNode->selectSingleNode(bsChildBaseName,&pSingleChildNode);
        pSingleChildNode->put_text(strValue.AllocSysString());
    }
}

```

6.3.2 CDirectoryView 类

该类显示了左边的树视图，包括了整个文档的结构。

(1) 函数: InitXMLDoc ()

功能: 加载文档, 得到文档指针 m_pDoc

参数: void

代码如下:

```
void InitXMLDoc()
{
    if (SUCCEEDED(CoInitialize(NULL)))
    {
        if(SUCCEEDED(CoCreateInstance(CLSID_DOMDocument, NULL,
                                       CLSCTX_INPROC_SERVER,
                                       IID_IXMLDOMDocument,
                                       (void**)&m_pDoc)))
        {
            CString strParth = ".\\doc\\URLText.xml";
            CComVariant vParth(strParth);
            VARIANT_BOOL isSuccessful;
            m_pDoc->load(vParth,&isSuccessful);
            if(isSuccessful==VARIANT_TRUE)
            {
                AddRootToTree();
            }
        }
    }
}
```

(2) 函数: AddRootToTree()

功能: 将“URL”和“文本”加到树视图中

参数: void

代码如下:

```
void AddRootToTree()
{
    CTreeCtrl* pTreeCtrl = & GetTreeCtrl();
    HTREEITEM hURLItem = AddStrToTree(TVI_ROOT,0,0,"URL");
    m_hURLsItem = hURLItem;
    ParseURLGroupToTree(hURLItem);
    HTREEITEM hTextItem = AddStrToTree(TVI_ROOT,1,1,"文本");
    m_hTextsItem = hTextItem;
    ParseTextGroupToTree(hTextItem);
}
```

(3) 函数: ParseURLGroupToTree(HTREEITEM hParentItem)

功能: 函数将 URLs 下面的所有组解析到树视图中

参数: hParentItem 在树视图中的位置

代码如下:

```
void ParseURLGroupToTree(HTREEITEM hParentItem)
{
    CString strURLs = _T("URLs");
    BSTR bsURLs = strURLs.AllocSysString();
    IXMLDOMNodeList* pList;
    IXMLDOMNode* pURLsNode;
    m_pDoc->getElementsByTagName(bsURLs,&pList);
    pList->get_item(0,&pURLsNode);
    m_pURLsNode = pURLsNode;
    VARIANT_BOOL hasChild;
    pURLsNode->hasChildNodes(&hasChild);
    if(hasChild == VARIANT_TRUE)
    {
        IXMLDOMNodeList* pGroupList;
        IXMLDOMNode* pGroupNode;
        long lngGroups;
        pURLsNode->get_childNodes(&pGroupList);
        pGroupList->get_length(&lngGroups);
        for(long i=0;i<lngGroups;i++)
        {
            pGroupList->get_item(i,&pGroupNode);
            CString strName = m_xmlParse..
            GetNodeAttributeValue("Name",pGroupNode);
            HTREEITEM hGroupItem = AddStrToTree(hParentItem,2,2,strName);
            pGroupNode->hasChildNodes(&hasChild);
            if(hasChild == VARIANT_TRUE)
            {
                ParseItemToTree(hGroupItem,pGroupNode);
            }
            struct TreeGroupInfo* pGroupInfo = new struct TreeGroupInfo;
            pGroupInfo->pGroupNode = pGroupNode;
            pGroupInfo->hGroupItem = hGroupItem;
            m_GroupList.AddTail(pGroupInfo);
        }
        // ParseURLToTree(pURLsNode);
    }
}
```

先用 `getElementsByTagName` 和 `get_item` 得到组节点, 然后用 `CParseXMLDOM` 类的 `GetNodeAttributeValue` 函数得到组名, 并加到树视图中。

(4) 函数: `ParseTextGroupToTree (HTREEITEM hParentItem)`

功能: 函数将 `Texts` 下面的所有组解析到树视图中

参数: hParentItem 在树视图中的位置

代码如下:

```
void ParseTextGroupToTree(HTREEITEM hParentItem)
{
    CString strTexts = _T("Texts");
    BSTR bsTexts = strTexts.AllocSysString();
    IXMLDOMNodeList* pList;
    IXMLDOMNode* pTextsNode;
    m_pDoc->getElementsByTagName(bsTexts,&pList);
    pList->get_item(0,&pTextsNode);
    m_pTextsNode = pTextsNode;
    VARIANT_BOOL hasChild;
    pTextsNode->hasChildNodes(&hasChild);
    if(hasChild == VARIANT_TRUE)
    {
        IXMLDOMNodeList* pGroupList;
        IXMLDOMNode* pGroupNode;
        long lngGroups;
        pTextsNode->get_childNodes(&pGroupList);
        pGroupList->get_length(&lngGroups);
        for(long i=0;i<lngGroups;i++)
        {
            pGroupList->get_item(i,&pGroupNode);
            CString strName = m_xmlParse.
            GetNodeAttributeValue("Name",pGroupNode);
            HTREEITEM hGroupItem = AddStrToTree(hParentItem,2,2,strName);
            pGroupNode->hasChildNodes(&hasChild);
            if(hasChild == VARIANT_TRUE)
            {
                ParseItemToTree(hGroupItem,pGroupNode);
            }
            struct TreeGroupInfo* pGroupInfo = new struct TreeGroupInfo;
            pGroupInfo->pGroupNode = pGroupNode;
            pGroupInfo->hGroupItem = hGroupItem;
            m_GroupList.AddTail(pGroupInfo);
        }
    }
}
```

和 ParseURLGroupToTree 函数差不多。

(5) 函数: ParseItemToTree(HTREEITEM hParentItem,IXMLDOMNode* pGroupNode)

功能: 将某组下的信息解析到树视图中

参数: hParentItem 在树视图中的位置

hGroupNode 组节点指针

代码如下:

```
void ParseItemToTree(HTREEITEM hParentItem,IXMLDOMNode* pGroupNode)
{
    IXMLDOMNodeList* pList;
    IXMLDOMNode* pItemNode;
    // CString strURL = _T("URL");
    // BSTR bsURL = strURL.AllocSysString();
    pGroupNode->get_childNodes(&pList);
    long lngItems;
    pList->get_length(&lngItems);
    for(long i=0;i<lngItems;i++)
    {
        pList->get_item(i,&pItemNode);
        CString strName = m_xmlParse.GetSingleChildNodeText("Name",pItemNode);
        AddStrToTree(hParentItem,3,3,strName);
    }
}
```

先用 get_childNodes 和 get_item 得到信息节点指针,然后用 CParseXMLDOM 类的 GetSingleChildNodeText 函数得到信息名。

(6) 函数: AddStrToTree(HTREEITEM hItem,int ilmage,int iSeledImage,CString str)

功能: 将一个字符串加到树视图的特定位置。

参数: hItem 在树视图中的位置
 ilmage 在树视图中的图标序号
 iSelectedImage 在树视图中选中时的图标序号
 str 字符串

代码如下:

```
HTREEITEM AddStrToTree(HTREEITEM hItem,int ilmage, int iSeledImage,CString str)
{
    CTreeCtrl* pTreeCtrl = &GetTreeCtrl();
    TV_INSERTSTRUCT TCItem;//插入数据项数据结构
    TCItem.hParent = hItem;
    TCItem.hInsertAfter = TVI_LAST;
    TCItem.item.mask=TVIF_TEXT|TVIF_PARAM|TVIF_IMAGE|TVIF_SELECTEDIMAGE;//设屏
    TCItem.item.pszText=(LPTSTR)(LPCTSTR)str;
    TCItem.item.lParam=0;//序号
    TCItem.item.ilImage=ilmage;//正常图标
    TCItem.item.iSelectedImage=iSeledImage;//选中图标
    HTREEITEM hNextItem=pTreeCtrl->InsertItem(&TCItem);//插入ID节点
    return hNextItem;
```

蔽

```
}
```

该类的完整代码如下：

程序清单 6.11：

```
// DirectoryView.h : header file
//

/////////////////////////////////////////////////////////////////
// CDirectoryView view

#include "ParseXMLDOM.h"
#include "ItemsInfo.h"
struct TreeGroupInfo
{
    HTREEITEM hGroupItem;
    IXMLDOMNode* pNode;
};
class CDirectoryView : public CTreeView
{
protected:
    CDirectoryView();          // protected constructor used by dynamic creation
    DECLARE_DYNCREATE(CDirectoryView)

// Attributes
protected:
    CList<struct TreeGroupInfo*,struct TreeGroupInfo*>m_GroupList;//整个树型数组
    IXMLDOMNode* m_pURLsNode;
    IXMLDOMNode* m_pTextsNode;
    HTREEITEM m_hURLsItem;
    HTREEITEM m_hTextsItem;
    HTREEITEM m_hHitItem;
    CImageList m_ImageList;
    CParseXMLDOM m_xmlParse;
    IXMLDOMDocument* m_pDoc;
    CItemsInfo m_ItemsInfo;
    void InitXMLDoc();
    void AddRootToTree();
    void ParseURLGroupToTree(HTREEITEM hParentItem);
    void ParseTextGroupToTree(HTREEITEM hParentItem);

// void ParseTextToTree(HTREEITEM hParentItem, IXMLDOMNode* pNode);
    HTREEITEM AddStrToTree(HTREEITEM hItem, int iImage,
        int iSelectedImage, CString str);
```



```

        BOOL IsExistTheItem(CString strName, HTREEITEM hCurrentItem,
                           HTREEITEM hParentItem);

public:
    void ParseItemToTree(HTREEITEM hParentItem, IXMLDOMNode* pGroupNode);
    void DeleteChildItems(HTREEITEM hParentItem);

// Operations
public:

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CDirectoryView)
public:
    virtual void OnInitialUpdate();
protected:
    virtual void OnDraw(CDC* pDC);          // overridden to draw this view
//}}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CDirectoryView();

#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

    // Generated message map functions
protected:
    {{{AFX_MSG(CDirectoryView)
    afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnEditCopy();
    afx_msg void OnAddGroup();
    afx_msg void OnAddItem();
    afx_msg void OnRclick(NMHDR* pNMHDR, LRESULT* pResult);
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnMenuEdit();
    afx_msg void OnMenuDelete();
    }}}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

// DirectoryView.cpp : implementation file
//

```

```

#include "stdafx.h"
#include "CURLTextManager.h"
#include "DirectoryView.h"
#include "AddDlg.h"
#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CDirectoryView

IMPLEMENT_DYNCREATE(CDirectoryView, CTreeView)

CDirectoryView::CDirectoryView()
{
    m_hHitItem = NULL;
    m_pURLsNode = NULL;
    m_pTextsNode = NULL;
    m_hURLsItem = NULL;
    m_hTextsItem = NULL;
}

CDirectoryView::~CDirectoryView()
{
}

BEGIN_MESSAGE_MAP(CDirectoryView, CTreeView)
//{{AFX_MSG_MAP(CDirectoryView)
    ON_WM_RBUTTONDOWN()
    ON_COMMAND(ID_EDIT_COPY, OnEditCopy)
    ON_COMMAND(ID_MENU_ADDGROUP, OnAddGroup)
    ON_COMMAND(ID_MENU_ADDITEM, OnAddItem)
    ON_NOTIFY_REFLECT(NM_RCLICK, OnRclick)
    ON_WM_LBUTTONDOWN()
    ON_COMMAND(ID_MENU_EDIT, OnMenuEdit)
    ON_COMMAND(ID_MENU_DELETE, OnMenuDelete)
//}}AFX_MSG_MAP

```

```

END_MESSAGE_MAP()

////////////////////////////////////
// CDirectoryView drawing

void CDirectoryView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
}

////////////////////////////////////
// CDirectoryView diagnostics

#ifdef _DEBUG
void CDirectoryView::AssertValid() const
{
    CTreeView::AssertValid();
}

void CDirectoryView::Dump(CDumpContext& dc) const
{
    CTreeView::Dump(dc);
}
#endif // _DEBUG

////////////////////////////////////
// CDirectoryView message handlers
void CDirectoryView::OnInitialUpdate()
{
    CTreeView::OnInitialUpdate();

    CTreeCtrl* pTreeCtrl = &GetTreeCtrl();
    ::SetWindowLong(m_hWnd, GWL_STYLE, WS_VISIBLE | WS_TABSTOP
        | WS_CHILD | WS_BORDER | TVS_HASBUTTONS
        | TVS_LINESATROOT | TVS_HASLINES
        | TVS_DISABLEDRAHDROP);
    CCURLTextManagerApp *pApp=
        (CCURLTextManagerApp*)AfxGetApp();//创建图象列表
    m_ImageList.Create(16,16,ILC_COLOR16,5,5);
    m_ImageList.Add(pApp->LoadIcon(IDI_ICON_URL_HOME));
    m_ImageList.Add(pApp->LoadIcon(IDI_ICON_TEXT_HOME));
    m_ImageList.Add(pApp->LoadIcon(IDI_ICON_GROUP));

```

```

        m_ImageList.Add(pApp->LoadIcon(IDI_ICON_URL_URL));
        m_ImageList.Add(pApp->LoadIcon(IDI_ICON_TEXT_TEXT));
        pTreeCtrl->SetImageList(&m_ImageList,TVSIL_NORMAL);
        InitXMLDoc();
    }
    void CDirectoryView::InitXMLDoc()
    {
        if (SUCCEEDED(CoInitialize(NULL)))
        {
            if(SUCCEEDED(CoCreateInstance(CLSID_DOMDocument, NULL,
                CLSCTX_INPROC_SERVER,
                IID_IXMLDOMDocument,
                (void**)&m_pDoc)))
            {
                CString strParth = ".\\doc\\URLText.xml";
                CComVariant vParth(strParth);
                VARIANT_BOOL isSuccessful;
                m_pDoc->load(vParth,&isSuccessful);
                if(isSuccessful==VARIANT_TRUE)
                {
                    AddRootToTree();
                }
            }
        }
    }
    void CDirectoryView::AddRootToTree()
    {
        CTreeCtrl* pTreeCtrl = & GetTreeCtrl();
        HTREEITEM hURLItem = AddStrToTree(TVI_ROOT,0,0,"URL");
        m_hURLsItem = hURLItem;
        ParseURLGroupToTree(hURLItem);
        HTREEITEM hTextItem = AddStrToTree(TVI_ROOT,1,1,"文本");
        m_hTextsItem = hTextItem;
        ParseTextGroupToTree(hTextItem);
    }
    void CDirectoryView::ParseURLGroupToTree(HTREEITEM hParentItem)
    {
        CString strURLs = _T("URLs");
        BSTR bsURLs = strURLs.AllocSysString();
        IXMLDOMNodeList* pList;
        IXMLDOMNode* pURLsNode;
        m_pDoc->getElementsByTagName(bsURLs,&pList);
        pList->get_item(0,&pURLsNode);
    }

```

```

m_pURLsNode = pURLsNode;
VARIANT_BOOL hasChild;
pURLsNode->hasChildNodes(&hasChild);
if(hasChild == VARIANT_TRUE)
{
    IXMLDOMNodeList* pGroupList;
    IXMLDOMNode* pGroupNode;
    long lngGroups;
    pURLsNode->get_childNodes(&pGroupList);
    pGroupList->get_length(&lngGroups);
    for(long i=0;i<lngGroups;i++)
    {
        pGroupList->get_item(i,&pGroupNode);
        CString strName = m_xmlParse.GetNodeAttributeValue
            ("Name",pGroupNode);
        HTREEITEM hGroupItem = AddStrToTree
            (hParentItem,2,2,strName);
        pGroupNode->hasChildNodes(&hasChild);
        if(hasChild == VARIANT_TRUE)
        {
            ParseItemToTree(hGroupItem,pGroupNode);
        }
        struct TreeGroupInfo* pGroupInfo =
            new struct TreeGroupInfo;
        pGroupInfo->pGroupNode = pGroupNode;
        pGroupInfo->hGroupItem = hGroupItem;
        m_GroupList.AddTail(pGroupInfo);
    }
}
}

void CDirectoryView::ParseTextGroupToTree(HTREEITEM hParentItem)
{
    CString strTexts = _T("Texts");
    BSTR bsTexts = strTexts.AllocSysString();
    IXMLDOMNodeList* pList;
    IXMLDOMNode* pTextsNode;
    m_pDoc->getElementsByTagName(bsTexts,&pList);
    pList->get_item(0,&pTextsNode);
    m_pTextsNode = pTextsNode;
    VARIANT_BOOL hasChild;
    pTextsNode->hasChildNodes(&hasChild);
    if(hasChild == VARIANT_TRUE)
    {

```

```

        IXMLDOMNodeList* pGroupList;
        IXMLDOMNode* pGroupNode;
        long lngGroups;
        pTextsNode->get_childNodes(&pGroupList);
        pGroupList->get_length(&lngGroups);
        for(long i=0;i<lngGroups;i++)
        {
            pGroupList->get_item(i,&pGroupNode);
            CString strName = m_xmlParse.
                GetNodeAttributeValue("Name",pGroupNode);
            HTREEITEM hGroupItem =
                AddStrToTree(hParentItem,2,2,strName);
            pGroupNode->hasChildNodes(&hasChild);
            if(hasChild == VARIANT_TRUE)
            {
                ParseItemToTree(hGroupItem,pGroupNode);
            }
            struct TreeGroupInfo* pGroupInfo =
                new struct TreeGroupInfo;
            pGroupInfo->pGroupNode = pGroupNode;
            pGroupInfo->hGroupItem = hGroupItem;
            m_GroupList.AddTail(pGroupInfo);
        }
    }
}

void CDirectoryView::ParseItemToTree(HTREEITEM hParentItem,
    IXMLDOMNode* pGroupNode)
{
    IXMLDOMNodeList* pList;
    IXMLDOMNode* pItemNode;
    pGroupNode->get_childNodes(&pList);
    long lngItems;
    pList->get_length(&lngItems);
    for(long i=0;i<lngItems;i++)
    {
        pList->get_item(i,&pItemNode);
        CString strName = m_xmlParse.
            GetSingleChildNodeText("Name",pItemNode);
        AddStrToTree(hParentItem,3,3,strName);
    }
}

HTREEITEM CDirectoryView::AddStrToTree(HTREEITEM hItem,int ilmage,
    int iSeledImage,CString str)

```

```

{
    CTreeCtrl* pTreeCtrl = &GetTreeCtrl();
    TV_INSERTSTRUCT TCItem; //插入数据项数据结构
    TCItem.hParent = hItem;
    TCItem.hInsertAfter = TVI_LAST;
    TCItem.item.mask=TVIF_TEXT|TVIF_PARAM
        |TVIF_IMAGE|TVIF_SELECTEDIMAGE; //设屏蔽
    TCItem.item.pszText=(LPTSTR)(LPCTSTR)str;
    TCItem.item.lParam=0; //序号
    TCItem.item.ilImage=ilImage; //正常图标
    TCItem.item.iSelectedImage=iSeledImage; //选中时图标
    HTREEITEM hNextItem=pTreeCtrl->InsertItem(&TCItem); //插入ID节点
    return hNextItem;
}

void CDirectoryView::OnRButtonDown(UINT nFlags, CPoint point)
{
    CTreeView::OnRButtonDown(nFlags, point);
    CTreeCtrl* pTreeCtrl = & GetTreeCtrl();
    HTREEITEM hHitItem = pTreeCtrl->HitTest(point,&nFlags);
    m_hHitItem = hHitItem;
    if(hHitItem == NULL)
        return;
    //弹出菜单
    CRect rc;
    this->GetWindowRect(&rc);
    CMenu* pAddMenu = new CMenu;
    pAddMenu->LoadMenu(IDR_MENU_ADD);
    CMenu* pFileMenu = pAddMenu->GetSubMenu(0);
    ASSERT(pFileMenu);
    point.x = rc.left + point.x;
    point.y = rc.top + point.y;

    //验证点击的项
    HTREEITEM hHitParentItem = pTreeCtrl->GetParentItem(hHitItem);
    if(hHitParentItem == NULL && hHitItem != NULL) //点击URL或者Text
    {
        pFileMenu->EnableMenuItem(ID_MENU_ADDITEM,
            MF_GRAYED|MF_DISABLED|MF_BYCOMMAND);
        pFileMenu->EnableMenuItem(ID_MENU_EDIT,
            MF_GRAYED|MF_DISABLED|MF_BYCOMMAND);
        pFileMenu->EnableMenuItem(ID_MENU_DELETE,
            MF_GRAYED|MF_DISABLED|MF_BYCOMMAND);
    }
}

```

```

    }
    else if(pTreeCtrl->GetParentItem(hHitParentItem) == NULL
           && hHitParentItem != NULL)//点击组
    {
        pFileMenu->EnableMenuItem(ID_MENU_ADDGROUP,
                                   MF_GRAYED|MF_DISABLED|MF_BYCOMMAND);
    }
    else if(pTreeCtrl->GetChildItem(hHitItem) == NULL
           && hHitItem != NULL)//点击项
    {
        pFileMenu->EnableMenuItem(ID_MENU_ADDGROUP,
                                   MF_GRAYED|MF_DISABLED|MF_BYCOMMAND);
        pFileMenu->EnableMenuItem(ID_MENU_ADDITEM,
                                   MF_GRAYED|MF_DISABLED|MF_BYCOMMAND);
    }

    pFileMenu->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON, point.x,
                              point.y, this);

    delete pAddMenu;
    pAddMenu = NULL;
}

void CDirectoryView::OnEditCopy()
{
    // TODO: Add your command handler code here
}

void CDirectoryView::OnAddGroup()
{
    CAddGroupDlg dlgGroup;
    if(dlgGroup.DoModal() == IDOK)
    {
        if(IsExistTheItem(dlgGroup.m_strName,
                          NULL, m_hHitItem) == TRUE)
        {
            AfxMessageBox("已经存在该组了,请另取一个名字!");
            return;
        }
        HTREEITEM hNewGroupItem = AddStrToTree
            (m_hHitItem, 2, 2, dlgGroup.m_strName);
        IXMLDOMNode* pNewGroupNode;
        CComVariant varType(NODE_ELEMENT);
        CString strGroup = _T("Group");
    }
}

```



```

        m_pDoc->createNode(varType,strGroup.AllocSysString(),
                           NULL,&pNewGroupNode);
        IXMLDOMNode* pNew;
        if(m_hHitItem==m_hURLsItem)
        {
            m_pURLsNode->appendChild(pNewGroupNode,&pNew);
        }
        else if(m_hHitItem==m_hTextsItem)
        {
            m_pTextsNode->appendChild(pNewGroupNode,&pNew);
        }
        BSTR bsName = CString("Name").AllocSysString();
        CComVariant varValue(dlgGroup.m_strName);
        IXMLDOMElement* pNewGroupEl =
            m_xmlParse.GetElementByNode(pNew);
        pNewGroupEl->setAttribute(bsName,varValue);
        struct TreeGroupInfo* pGroupInfo =
            new struct TreeGroupInfo;
        pGroupInfo->pGroupNode = pNew;
        pGroupInfo->hGroupItem = hNewGroupItem;
        m_GroupList.AddTail(pGroupInfo);

        CComVariant varParth(CString(".\\doc\\URLText.xml"));
        m_pDoc->save(varParth);
    }
}

void CDirectoryView::OnAddItem()
{
    CAddItemDlg dlgItem;
    dlgItem.m_strStaticBody = "内容: ";
    if(dlgItem.DoModal()==IDOK)
    {
        if(IsExistTheItem(dlgItem.m_strName,
                          NULL,m_hHitItem)==TRUE)
        {
            AfxMessageBox("已经存在该项了,请另取一个名字!");
            return;
        }
        AddStrToTree(m_hHitItem,3,3,dlgItem.m_strName);
        struct TreeGroupInfo* pGroupInfo;
        POSITION pos = m_GroupList.GetHeadPosition();
        BOOL bHit = FALSE;
        while(bHit != TRUE)

```

```

{
    pGroupInfo = m_GroupList.GetNext(pos);
    if(pGroupInfo->hGroupItem == m_hHitItem)
        bHit = TRUE;
}
IXMLDOMNode* pItemNode;
CComVariant varType(NODE_ELEMENT);
CString strItem = _T("Item");
m_pDoc->createNode(varType,strItem.AllocSysString(),
    NULL,&pItemNode);
IXMLDOMNode* pNewItemNode;

IXMLDOMNode* pChildNode;
IXMLDOMNode* pNewChildNode;
CComVariant varNodeValue(dlgItem.m_strName);
CString strNode = _T("Name");
m_pDoc->createNode(varType,strNode.AllocSysString(),
    NULL,&pChildNode);
pChildNode->put_text(dlgItem.m_strName.AllocSysString());
pItemNode->appendChild(pChildNode,&pNewChildNode);

strNode = _T("Body");
varNodeValue = CComVariant(dlgItem.m_strBody);
m_pDoc->createNode(varType,strNode.AllocSysString(),
    NULL,&pChildNode);
pChildNode->put_text(dlgItem.m_strBody.AllocSysString());
pItemNode->appendChild(pChildNode,&pNewChildNode);

strNode = _T("Comment");
varNodeValue = CComVariant(dlgItem.m_strComment);
m_pDoc->createNode(varType,strNode.AllocSysString(),
    NULL,&pChildNode);
pChildNode->put_text(dlgItem.m_strComment.AllocSysString());
pItemNode->appendChild(pChildNode,&pNewChildNode);

pGroupInfo->pGroupNode->appendChild(pItemNode,&pNewItemNode);

CComVariant varParth(CString(".\\doc\\URLText.xml"));
m_pDoc->save(varParth);
}
}

```

```

void CDirectoryView::OnRclick(NMHDR* pNMHDR, LRESULT* pResult)
{
    *pResult = 0;
}

void CDirectoryView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CTreeView::OnLButtonDown(nFlags, point);
    CTreeCtrl* pTreeCtrl = & GetTreeCtrl();
    HTREEITEM hHitItem = pTreeCtrl->HitTest(point, &nFlags);
    //验证点击的项
    HTREEITEM hHitParentItem = pTreeCtrl->GetParentItem(hHitItem);
    if(pTreeCtrl->GetParentItem(hHitParentItem) == NULL
        && hHitParentItem != NULL)//点击组
    {
        struct TreeGroupInfo* pGroupInfo;
        POSITION pos = m_GroupList.GetHeadPosition();
        BOOL bHit = FALSE;
        while(bHit != TRUE)
        {
            pGroupInfo = m_GroupList.GetNext(pos);
            if(pGroupInfo->hGroupItem == hHitItem)
                bHit = TRUE;
        }
        CCURLTextManagerApp* pApp =
            (CCURLTextManagerApp*)AfxGetApp();
        CMainFrame* pMainFrm = (CMainFrame*)pApp->m_pMainWnd;
        pMainFrm->m_pContentView->SetGroupInfo(pGroupInfo);
    }
}

void CDirectoryView::OnMenuEdit()
{
    CTreeCtrl* pTreeCtrl = & GetTreeCtrl();
    //验证点击的项
    HTREEITEM hHitParentItem =
        pTreeCtrl->GetParentItem(m_hHitItem);
    if(hHitParentItem == NULL
        && m_hHitItem != NULL)//点击URL或者Text
    {
        return;
    }
    else if(pTreeCtrl->GetParentItem(hHitParentItem) == NULL

```

```

        && hHitParentItem != NULL)//点击组
    {
        CAddGroupDlg dlgGroup;
        struct TreeGroupInfo* pGroupInfo;
        POSITION pos = m_GroupList.GetHeadPosition();
        BOOL bHit = FALSE;
        while(bHit != TRUE)
        {
            pGroupInfo = m_GroupList.GetNext(pos);
            if(pGroupInfo->hGroupItem == m_hHitItem)
                bHit = TRUE;
        }
        dlgGroup.m_strName = m_xmlParse.GetNodeAttributeValue
            ("Name",pGroupInfo->pGroupNode);
        if(dlgGroup.DoModal()==IDOK)
        {
            if(IsExistTheItem(dlgGroup.m_strName,
                pGroupInfo->hGroupItem ,
                hHitParentItem)==TRUE)
            {
                AfxMessageBox("已经存在该组了,请另取一个名字!");
                return;
            }
            m_xmlParse.SetNodeAttributeValue("Name",
                dlgGroup.m_strName,pGroupInfo->pGroupNode);
            pTreeCtrl->SetItemText(m_hHitItem,dlgGroup.m_strName);
            CComVariant varParth(CString(".\\doc\\URLText.xml"));
            m_pDoc->save(varParth);
        }
    }
else if(pTreeCtrl->GetChildItem(m_hHitItem) == NULL
    && m_hHitItem != NULL)//点击项
{
    struct TreeGroupInfo* pGroupInfo;
    POSITION pos = m_GroupList.GetHeadPosition();
    BOOL bHit = FALSE;
    while(bHit != TRUE)
    {
        pGroupInfo = m_GroupList.GetNext(pos);
        if(pGroupInfo->hGroupItem == hHitParentItem)
            bHit = TRUE;
    }
    IXMLDOMNodeList* pItemList;

```

```

IXMLDOMNode* pItemNode;
pGroupInfo->pGroupNode->get_childNodes(&pItemList);
long lngItems;
CString strHitItemText =
    pTreeCtrl->GetItemText(m_hHitItem);
pItemList->get_length(&lngItems);
for(long i=0;i<lngItems;i++)
{
    pItemList->get_item(i,pItemNode);
    if(m_xmlParse.GetSingleChildNodeText
        ("Name",pItemNode)==strHitItemText)
    {
        CAddItemDlg dlgItem;
        dlgItem.m_strStaticBody = "内容: ";
        dlgItem.m_strName = m_xmlParse
            .GetSingleChildNodeText("Name",pItemNode);
        dlgItem.m_strBody = m_xmlParse.
            GetSingleChildNodeText("Body",pItemNode);
        dlgItem.m_strComment = m_xmlParse.
            GetSingleChildNodeText("Comment",pItemNode);
        if(dlgItem.DoModal() == IDOK)
        {
            if(IsExistTheItem(dlgItem.m_strName,m_hHitItem,
                hHitParentItem)==TRUE)
            {
                AfxMessageBox("已经存在该组了,请另取一个名字!");
                return;
            }
            m_xmlParse.SetSingleChildNodeText("Name",
                dlgItem.m_strName,pItemNode);
            m_xmlParse.SetSingleChildNodeText("Body",
                dlgItem.m_strBody,pItemNode);
            m_xmlParse.SetSingleChildNodeText("Comment",
                dlgItem.m_strComment,pItemNode);
            pTreeCtrl->SetItemText(m_hHitItem,dlgItem.m_strName);
            CComVariant varParth(CString(".\\doc\\URLText.xml"));
            m_pDoc->save(varParth);
        }
        break;
    }
}
}
}
}

```

```

void CDirectoryView::OnMenuDelete()
{
    CTreeCtrl* pTreeCtrl = & GetTreeCtrl();
    //验证点击的项
    HTREEITEM hHitParentItem = pTreeCtrl->GetParentItem(m_hHitItem);
    if(hHitParentItem == NULL && m_hHitItem != NULL)//点击URL或者Text
    {
        return;
    }
    else if(pTreeCtrl->GetParentItem(hHitParentItem) == NULL
        && hHitParentItem != NULL)//点击组
    {
        struct TreeGroupInfo* pGroupInfo;
        POSITION pos = m_GroupList.GetHeadPosition();
        BOOL bHit = FALSE;
        while(bHit != TRUE)
        {
            pGroupInfo = m_GroupList.GetNext(pos);
            if(pGroupInfo->hGroupItem == m_hHitItem)
                bHit = TRUE;
        }
        IXMLDOMNode* pGroupParentNode;
        pGroupInfo->pGroupNode->get_parentNode(&pGroupParentNode);
        IXMLDOMNode* pGroupOutNode;
        pGroupParentNode->removeChild(pGroupInfo->pGroupNode,
            &pGroupOutNode);
        pTreeCtrl->DeleteItem(pGroupInfo->hGroupItem);
        CComVariant varParth(CString(".\\doc\\URLText.xml"));
        m_pDoc->save(varParth);
    }
    else if(pTreeCtrl->GetChildItem(m_hHitItem) == NULL
        && m_hHitItem != NULL)//点击项
    {
        struct TreeGroupInfo* pGroupInfo;
        POSITION pos = m_GroupList.GetHeadPosition();
        BOOL bHit = FALSE;
        while(bHit != TRUE)
        {
            pGroupInfo = m_GroupList.GetNext(pos);
            if(pGroupInfo->hGroupItem == hHitParentItem)
                bHit = TRUE;
        }
    }
}

```

```

IXMLDOMNodeList* pItemList;
IXMLDOMNode* pItemNode;
pGroupInfo->pGroupNode->get_childNodes(&pItemList);
long lngItems;
CString strHitItemText =
    pTreeCtrl->GetItemText(m_hHitItem);
pItemList->get_length(&lngItems);
for(long i=0;i<lngItems;i++)
{
    pItemList->get_item(i,&pItemNode);
    if(m_xmlParse.GetSingleChildNodeText
        ("Name",pItemNode)==strHitItemText)
    {
        IXMLDOMNode* pOutItemNode;
        pGroupInfo->pGroupNode->removeChild
            (pItemNode,&pOutItemNode);
        pTreeCtrl->DeleteItem(m_hHitItem);
        CComVariant varParth(
            CString(".\\doc\\URLText.xml"));
        m_pDoc->save(varParth);
        break;
    }
}
}
}

BOOL CDirectoryView::IsExistTheItem(CString strName,HTREEITEM hCurrentItem,
    HTREEITEM hParentItem)
{
    CTreeCtrl* pTreeCtrl = &GetTreeCtrl();
    if (pTreeCtrl->ItemHasChildren(hParentItem))
    {
        HTREEITEM hNextItem;
        HTREEITEM hChildItem =
            pTreeCtrl->GetChildItem(hParentItem);
        if(strName==pTreeCtrl->GetItemText(hChildItem)
            && hChildItem != hCurrentItem)
            return TRUE;
        while (hChildItem != NULL)
        {
            hNextItem = pTreeCtrl->GetNextItem
                (hChildItem, TVGN_NEXT);
            hChildItem = hNextItem;
            if(strName==pTreeCtrl->GetItemText(hChildItem)

```

```

        && hChildItem != hCurrentItem)
        return TRUE;
    }
    return FALSE;
}
else
{
    return FALSE;
}
}
void CDirectoryView::DeleteChildItems(HTREEITEM hParentItem)
{
    CTreeCtrl* pCtrl = &GetTreeCtrl();
    while(pCtrl->ItemHasChildren(hParentItem))
    {
        HTREEITEM hItem = pCtrl->GetChildItem(hParentItem);
        pCtrl->DeleteItem(hItem);
    }
}

```

6.3.3 CContentView 类

该类显示右边的列表视图，显示具体的每一个条目，根据左边树视图的情况作出响应。

(1) 函数：AddDataToList()

功能： 将对应组节点的内容解析到该列表视中。

参数： void

代码如下：

```

void CContentView::AddDataToList()
{
    CListCtrl* pCtrl = &GetListCtrl();
    pCtrl->DeleteAllItems();
    VARIANT_BOOL hasChild;
    m_pGroupInfo->pGroupNode->hasChildNodes(&hasChild);
    if(hasChild == VARIANT_TRUE)
    {
        IXMLDOMNodeList* pltemList;
        IXMLDOMNode* pltemNode;
        long lngItems;
        m_pGroupInfo->pGroupNode->get_childNodes(&pltemList);
        pltemList->get_length(&lngItems);
        for(long i=0;i<lngItems;i++)
        {
            pltemList->get_item(i,&pltemNode);

```



```

        AddItemToList(i, pItemNode);
    }
}
}

```

(2) 函数: AddItemToList(int iItem, IXMLDOMNode* pItemNode)

功能: 将某个条目节点加到列表视图中的相应位置

参数: iItem 列表视中的位置

pItemNode 项目节点

代码如下:

```

void CContentView::AddItemToList(int iItem, IXMLDOMNode* pItemNode)
{
    CListCtrl* pCtrl = &GetListCtrl();
    LV_ITEM lvitem;
    int iActualItem;
    CString strName = _T("");
    CString strBody = _T("");
    CString strComment = _T("");
    for(int iSubItem=0; iSubItem<3; iSubItem++)
    {
        lvitem.mask = LVIF_TEXT|(iSubItem == 0? LVIF_IMAGE : 0);
        lvitem.iItem = (iSubItem == 0)? iItem : iActualItem;
        lvitem.iSubItem = iSubItem;
        lvitem.ilImage = (iItem%2)?0:2;
        switch(iSubItem)
        {
            case 0:
                strName = m_xmlParse.GetSingleChildNodeText
                    ("Name", pItemNode);
                lvitem.pszText = (LPTSTR)(LPCTSTR) strName;
                break;
            case 1:
                strBody = m_xmlParse.GetSingleChildNodeText
                    ("Body", pItemNode);
                lvitem.pszText = (LPTSTR)(LPCTSTR) strBody;
                break;
            case 2:
                strComment = m_xmlParse.GetSingleChildNodeText
                    ("Comment", pItemNode);
                lvitem.pszText = (LPTSTR)(LPCTSTR) strComment;
                break;
        }
        if (iSubItem == 0)
    }
}

```

```

        iActualItem = pCtrl->InsertItem(&lvitem);
    else
        pCtrl->SetItem(&lvitem);
    }
}

```

(3) 函数: OnEditDelete()

功能: 将该列表视图中选中条目对应的文档条目删掉

参数: void

代码如下:

```

void CContentView::OnEditDelete()
{
    CListCtrl* pCtrl = &GetListCtrl();
    IXMLDOMNodeList* pltemList;
    IXMLDOMNode* pltemNode;
    m_pGroupInfo->pGroupNode->get_childNodes(&pltemList);
    POSITION pos = pCtrl->GetFirstSelectedItemPosition();
    CList<IXMLDOMNode*, IXMLDOMNode*> DeleteltemList;
    while(pos!=NULL)
    {
        int nSelItem = pCtrl->GetNextSelectedItem(pos);
        pltemList->get_item(nSelItem, &pltemNode);
        DeleteltemList.AddTail(pltemNode);
    }
    pos = DeleteltemList.GetHeadPosition();
    while(pos!=NULL)
    {
        pltemNode = DeleteltemList.GetNext(pos);
        IXMLDOMNode* pOutItemNode;
        m_pGroupInfo->pGroupNode->removeChild(pltemNode, &pOutItemNode);
    }
    AddDataToList();
    CCURLTextManagerApp* pApp = (CCURLTextManagerApp*)AfxGetApp();
    CMainFrame* pFrame = (CMainFrame*)pApp->m_pMainWnd;
    pFrame->m_pDirectoryView->DeleteChildItems(m_pGroupInfo->hGroupItem);
    pFrame->m_pDirectoryView->ParseItemToTree(m_pGroupInfo->hGroupItem,
        m_pGroupInfo->pGroupNode);
    IXMLDOMDocument* pDoc;
    m_pGroupInfo->pGroupNode->get_ownerDocument(&pDoc);
    CComVariant varParth(CString(".\\doc\\URLText.xml"));
    pDoc->save(varParth);
}

```

该类的完整代码如下:

程序清单 6.12:

```

// ContentView.h : header file
//

////////////////////////////////////

// CContentView view
#include "ParseXMLDOM.h"
#include "Directoryview.h"
class CContentView : public CListView
{
protected:
    CContentView();           // protected constructor used by dynamic creation
    DECLARE_DYNCREATE(CContentView)

// Attributes
protected:
    struct TreeGroupInfo* m_pGroupInfo;
    CParseXMLDOM m_xmlParse;
    void SetListCtrlStyle();
    void SetTitle();

public:
    void SetGroupInfo(struct TreeGroupInfo* pGroupInfo);
    void AddDataToList();
    void AddItemToList(int item, IXMLDOMNode* pItemNode);

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CContentView)
public:
    virtual void OnInitialUpdate();
protected:
    virtual void OnDraw(CDC* pDC);      // overridden to draw this view
   //}}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CContentView();

#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;

```

```

#endif

    // Generated message map functions
protected:
   //{{AFX_MSG(CContentView)
    afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnEditDelete();
   //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif

// !defined(AFX_CONTENTVIEW_H__E610A8AE_F077_4CCA_94C3_1F0066416F2F__INCLUDED_)

// ContentView.cpp : implementation file
//

#include "stdafx.h"
#include "CURLTextManager.h"
#include "ContentView.h"
#include "MainFrm.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////

// CContentView

IMPLEMENT_DYNCREATE(CContentView, CListView)

CContentView::CContentView()
{
}

CContentView::~CContentView()
{
}

```

```

}

BEGIN_MESSAGE_MAP(CContentView, CListView)
//{{AFX_MSG_MAP(CContentView)
    ON_WM_RBUTTONDOWN()
    ON_COMMAND(ID_EDIT_DELETE, OnEditDelete)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CContentView drawing

void CContentView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
}

////////////////////////////////////
// CContentView diagnostics

#ifdef _DEBUG
void CContentView::AssertValid() const
{
    CListView::AssertValid();
}

void CContentView::Dump(CDumpContext& dc) const
{
    CListView::Dump(dc);
}
#endif // _DEBUG

////////////////////////////////////
// CContentView message handlers

void CContentView::OnInitialUpdate()
{
    CListView::OnInitialUpdate();

    SetListCtrlStyle();
    SetTitle();
}

```

```

}
void CContentView::SetListCtrlStyle()
{
    DWORD dwStyle = GetWindowLong(m_hWnd, GWL_STYLE);
    dwStyle &= ~(LVS_TYPMASK);
    dwStyle &= ~(LVS_EDITLABELS);
    // Make sure we have report view and send edit label messages.
    SetWindowLong( m_hWnd, GWL_STYLE, dwStyle |
LVS_REPORT|LVS_NOLABELWRAP );

    // Enable the full row selection and the drag drop of headers.
    DWORD styles = LVS_EX_FULLROWSELECT|LVS_EX_GRIDLINES;
    // Use macro since this is new and not in MFC.
    ListView_SetExtendedListViewStyleEx(m_hWnd, styles, styles );
}

void CContentView::SetTitle()
{
    CListCtrl* pListCtrl=&GetListCtrl();

    TCHAR rgtsz[3][10] = { _T("名字"),
        _T("内容"), _T("描述")};

    LV_COLUMN lvcolumn;
    CRect rect;
    pListCtrl->GetWindowRect(&rect);
    for(int i=0;i<3;i++)
    {
        lvcolumn.mask = LVCF_FMT | LVCF_SUBITEM | LVCF_TEXT
            | LVCF_WIDTH | LVCF_ORDER;
        lvcolumn.fmt = LVCFMT_LEFT;
        lvcolumn.pszText = rgtsz[i];
        lvcolumn.iSubItem = i;
        lvcolumn.iOrder = i;
        if(i==0)
        {
            lvcolumn.cx = rect.Height()/3;
        }
        else if(i==1)
            lvcolumn.cx = rect.Height()/2;
        else
    }

```

```

        lvcolumn.cx = rect.Height()-70;
        pListCtrl->InsertColumn(i, &lvcolumn);
    }
}

void CContentView::SetGroupInfo(struct TreeGroupInfo* pGroupInfo)
{
    m_pGroupInfo = pGroupInfo;
    AddDataToList();
}

void CContentView::AddDataToList()
{
    CListCtrl* pCtrl = &GetListCtrl();
    pCtrl->DeleteAllItems();
    VARIANT_BOOL hasChild;
    m_pGroupInfo->pGroupNode->hasChildNodes(&hasChild);
    if(hasChild == VARIANT_TRUE)
    {
        IXMLDOMNodeList* pItemList;
        IXMLDOMNode* pItemNode;
        long lngItems;
        m_pGroupInfo->pGroupNode->get_childNodes(&pItemList);
        pItemList->get_length(&lngItems);
        for(long i=0;i<lngItems;i++)
        {
            pItemList->get_item(i,&pItemNode);
            AddItemToList(i,pItemNode);
        }
    }
}

void CContentView::AddItemToList(int iltem, IXMLDOMNode* pItemNode)
{
    CListCtrl* pCtrl = &GetListCtrl();
    LV_ITEM lvitem;
    int iActualItem;
    CString strName = _T("");
    CString strBody = _T("");
    CString strComment = _T("");
    for(int iSubItem=0;iSubItem<3;iSubItem++)
    {
        lvitem.mask = LVIF_TEXT|(iSubItem == 0? LVIF_IMAGE : 0);
        lvitem.iltem = (iSubItem == 0)? iltem : iActualItem;
        lvitem.iSubItem = iSubItem;
        lvitem.ilImage = (iltem%2)?0:2;
    }
}

```

```

        switch(iSubItem)
        {
        case 0:
            strName = m_xmlParse.GetSingleChildNodeText
                ("Name",pItemNode);
            lvitem.pszText = (LPTSTR)(LPCTSTR) strName;
            break;
        case 1:
            strBody = m_xmlParse.GetSingleChildNodeText
                ("Body",pItemNode);
            lvitem.pszText = (LPTSTR)(LPCTSTR) strBody;
            break;
        case 2:
            strComment = m_xmlParse.GetSingleChildNodeText
                ("Comment",pItemNode);
            lvitem.pszText = (LPTSTR)(LPCTSTR) strComment;
            break;
        }
        if (iSubItem == 0)
            iActualItem = pCtrl->InsertItem(&lvitem);
        else
            pCtrl->SetItem(&lvitem);
    }
}

```

```

void CContentView::OnRButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default

    CListView::OnRButtonDown(nFlags, point);
    CListCtrl* pListCtrl = & GetListCtrl();
    int nHitItem = pListCtrl->HitTest(point,&nFlags);
    if(nHitItem == -1)
        return;
    //弹出菜单
    CRect rc;
    this->GetWindowRect(&rc);
    CMenu* pAddMenu = new CMenu;
    pAddMenu->LoadMenu(IDR_MENU_EDIT);
    CMenu* pFileMenu = pAddMenu->GetSubMenu(0);
    ASSERT(pFileMenu);
    point.x = rc.left + point.x;
    point.y = rc.top + point.y;
}

```



```

        pFileMenu->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON, point.x,
            point.y, this);
        delete pAddMenu;
        pAddMenu = NULL;
    }

    void CContentView::OnEditDelete()
    {
        CListCtrl* pCtrl = &GetListCtrl();
        IXMLDOMNodeList* pltemList;
        IXMLDOMNode* pltemNode;
        m_pGroupInfo->pGroupNode->get_childNodes(&pltemList);
        POSITION pos = pCtrl->GetFirstSelectedItemPosition();
        CList<IXMLDOMNode*, IXMLDOMNode*> DeleteltemList;
        while(pos!=NULL)
        {
            int nSelItem = pCtrl->GetNextSelectedItem(pos);
            pltemList->get_item(nSelItem, &pltemNode);
            DeleteltemList.AddTail(pltemNode);
        }
        pos = DeleteltemList.GetHeadPosition();
        while(pos!=NULL)
        {
            pltemNode = DeleteltemList.GetNext(pos);
            IXMLDOMNode* pOutItemNode;
            m_pGroupInfo->pGroupNode->removeChild(pltemNode, &pOutItemNode);
        }
        AddDataToList();
        CCURLTextManagerApp* pApp = (CCURLTextManagerApp*)AfxGetApp();
        CMainFrame* pFrame = (CMainFrame*)pApp->m_pMainWnd;
        pFrame->m_pDirectoryView->DeleteChildItems(m_pGroupInfo->hGroupItem);
        pFrame->m_pDirectoryView->ParseItemToTree(m_pGroupInfo->hGroupItem,
            m_pGroupInfo->pGroupNode);
        IXMLDOMDocument* pDoc;
        m_pGroupInfo->pGroupNode->get_ownerDocument(&pDoc);
        CComVariant varParth(CString(".\\doc\\URLText.xml"));
        pDoc->save(varParth);
    }
}

```

6.3.4 其他信息

除了上面介绍的三个主要类以外，本管理器还需要用到下面一些东西，由于本书篇幅有限，不再详细给出，只是介绍给大家。

- ☒ CAddGroupDlg 类：添加和编辑组(Group)节点的对话框。

☒ CAddItemDlg 类：添加和编辑 Item 节点的对话框。

本例中解析文档就是对文档的遍历。增加删除修改文档元素时，必须记住每次都要调用 IXMLDOMDocument 接口的 Save 函数，以把这些改动保存到文档中，存盘。做类似程序时，主要是要清楚了解文档的结构，然后按照树的特性访问该文档。

6.4 小 结

可以说，本章是第 5 章的继续，虽然本章没有介绍多少新的知识，但给出了三个比较详细的例子，包括投票系统，留言本，网址及短消息管理等。它们综合了 XML、DOM 以及 XSL 样式表等许多知识，读者需要仔细理解。

但是，某些情况下如读取相当大的 XML 文档时，使用 DOM 技术并不是最好的选择。因此我们还需要掌握另一种技术 SAX，所以下一章将介绍 SAX。

第 7 章 SAX 进阶

本章导读

由于 DOM 的某些局限，SAX 在某种环境下成了开发者的首选。SAX 的风格完全不同于 DOM，它是一种事件驱动接口，具有良好的兼容性。虽然在开始的时候，SAX 规范是用 Java 接口书写的，但是现在已经有许多语言支持 SAX 接口了。由于 Java 平台提供了网络上安全而方便地传播的基础，XML 技术也为数据提供了同样的能力，所以本章还是选用 Java 语言来编写应用程序。

7.1 SAX

7.1.1 SAX

SAX 即 XML 简易应用程序编程接口（Simple API for XML），它的接口风格完全不同于 DOM 接口，采用了基于事件的方式来处理 XML 文档。基于事件是指 SAX 为应用开发者提供了处理感兴趣的特定元素的方法，而不必要求在应用层次处理之前预先建立元素。这样做带来的好处是可以不用创建没有用的结构；作为替代，可以在任何感兴趣的事件发生时调用应用程序的源代码。

SAX 这个接口规范是 XML 分析器和 XML 处理器提供的较 XML 更底层的接口。从本质上说，SAX 是一种 Java 接口，它能给应用程序提供较大的灵活性。

SAX 提供了 `AttributeList`，`DocumentHandler`，`DTDHandler`，`EntityResolver`，`ErrorHandler`，`Locator`，`Parser` 等七个接口以及 `HandlerBase`，`InputSource`，`SAXException` 和 `SAXParseException` 等四个类，下一节将详细介绍这些接口和类。

解析器在对 XML 文档进行解析时，遇到不同的分析事件就会触发在上述的接口中定义的不同方法。可由程序员决定如何处理这些事件；当然也可以忽略它们，但会丢弃事件中的信息。如图 7-1 如下。

为了对 SAX 编程有一个总体的了解，先介绍最重要的一个接口：`DocumentHandler` 接口。

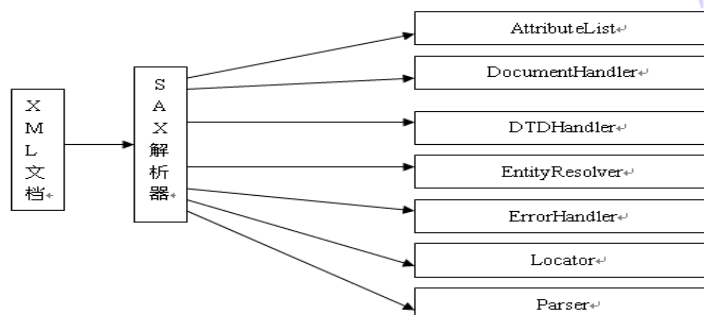


图 7-1 SAX 接口图解

该接口中，定义了一系列的方法，这些方法是 SAX 分析器在对 XML 文档进行分析时遇到不同的分析事件所激发的相应处理。现在回顾一下最常用的五个方法：startDocument, endDocument, startElement, endElement 以及 characters。

- ☒ StartDocument(): 表示文档开始。
- ☒ EndDocument(): 表示文档结束。
- ☒ startElement(String name, AttributeList attrs): 表示元素开始。当一对标记中的起始标记中的所有内容被处理后，解析器激发此事件。包括了标记名和其属性。标记名就作为一个字符串变量传递给 startElement 方法。起始标记所代表的元素的所有属性都以 AttributeList 的形式传递给 startElement 方法。
- ☒ endElement(String name): 表示元素结束。标记名就作为一个字符串变量传递给 endElement 方法。
- ☒ characters(char ch[],int start, int length): 包含字符数据，类似于 DOM 的一个 Text 节点。在一个元素内的字符以一个字符数组的形式传递给 characters 方法。characters 方法还有另外两个参数，其中 start 代表元素内的第一个字符在字符数组中的偏移，另一个参数 length 代表了字符数组的长度，也即字符的个数。

如果要对如下的 XML 文件用 SAX 进行处理。

程序清单 7.1:

```
<?xml version="1.0"?>
<guests>
  <guest>
    <name>yuxiang weng</name>
    <agegrade>a</agegrade>
    <money>1832.3</money>
  </guest>
</guests>
```

那么 SAX 解析器在解析时就调用如下的一系列方法。

```
startDocument()
  startElement("guests")
    startElement("guest")

      startElement("name")
      characters("yuxiang weng")
      endElement("name")

      startElement("agegrade")
      characters("a")
      endElement("agegrade")

      startElement("money")
```

```
characters("1832.3")
endElement("money")

endElement("guest")
endElement("guests")
endDocument()
```

这样一来, 开发者只要在应用程序中提供当上述事件发生时需要调用的方法就可以了。

7.1.2 DOM 和 SAX

前面一章已经介绍过了 DOM 接口, 现在来看一下 DOM 和 SAX 的一些显著的区别。

DOM 是基于树形结构的 W3C 推荐的 API 标准, 而 SAX 是事件驱动的有广泛支持的 API 标准。DOM 适合于结构化编辑 XML 文档, 例如排序、记录移动和其他应用、共享 XML 文档。而 SAX 效率高 (不创建显式数据结构), 适合大文档, 可完成内存不足与文档结构无关的任务, 如计算 XML 文档结点数、提取特定结点内容等。

应该说凡是 SAX 可以做的工作 DOM 都可以做, 但是由于 DOM 是整体装入和处理 XML 文档, 因此对于系统资源的占用很大, 效率低, 速度慢。这在处理大文档时尤其明显。采用 DOM 模型往往给服务器端带来很繁重的任务。而对于 SAX 来说, DOM 的很多工作它很难完成, 例如排序、移动等。因为它缺少对于 XML 文档的整体视图。这就要求在采用 SAX 之前必须确定自己的应用有没有不适合 SAX 的内容。

更具体的说, SAX 有着如下的优点:

(1) SAX 可以解析任意大小的文件

因为 SAX 不像 DOM 那样需要把整个 XML 文档加载到内存中, 所以它对内存的占用不会随着文档的大小而有所变化。

(2) 适合于只处理某些特定数据的场合

由于在某些场合下, 只需要处理我们所关心的数据, 所以把大量的无关数据和有用数据一起读入内存是一种浪费, 而 SAX 是一种事件驱动的接口, 适合于这种场合的应用。

(3) 简单快速

正如 SAX 的全名 “XML 简易应用程序编程接口” 所反映的, SAX 是一种快速而简单的接口, 绝大多数的操作解析器都可以由开发者自己编程实现。

但是, SAX 也存在着一些不足的地方:

(1) 不能对文档作随机的存取

因为 SAX 不会把整个 XML 文档加载到内存中, 所以它不能像 DOM 一样对 XML 文档作随机的存取。SAX 提供的是一种顺序访问机制, 对于已经处理过的数据, 不能再返回作处理。

(2) 目前还没有主流浏览器支持 SAX

虽然有许多 SAX 接口的解析器, 但是目前还没有主流浏览器支持 SAX 接口。

下图 7-2 表示了 DOM 和 SAX 接口在开发程序中所扮演的角色。

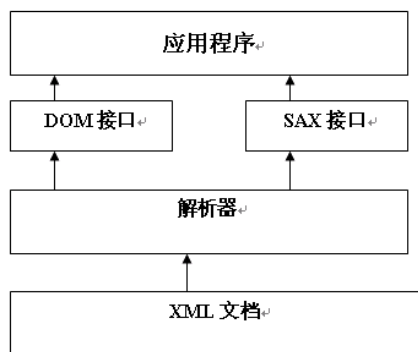


图 7-2 SAX 和 DOM 工作原理图

7.2 SAX 接口和类

SAX 是由许多 Java 接口构建的，有些接口是解析器中的类实现的，而有些接口则是由应用程序中的类来实现。

下面就分别介绍 SAX 的接口和类。

7.2.1 SAX 接口

AttributeList 接口

AttributeList 接口是出现在特定的起始标记中的一组属性，SAX 解析器实现了这个接口，并将接口实例作为每个 startElement 事件的第二个参数传递给 SAX 应用程序。只有在 startElement 的调用范围内，解析器提供的实例才能返回有效的结果。

一般而言，可以使用如下的两种方法从 AttributeList 中获取信息。

(1) 对整个列表作循环操作

用如下的一段代码来说明这个方法：

```

public void startElement (String name,AttributeList atts)
{
    for (int i=0;i<atts.getLength();i++)
    {
        String name=atts.getName(i);
        String type=atts.getType(i);
        String value=atts.getValue(i);
        其他操作
    }
}
  
```

这段代码中用到的方法可以参考下面给出的 AttributeList 接口方法表。

(2) 对特定属性值或类型的操作

```

public void startElement (String name,AttributeList atts)
  
```

```

{
    String id=atts.getValue("id");
    String publisher=atts.getValue("publisher");
    其他操作
}

```

这段代码使用了 `Public String getValue(String name)` 方法，根据名称返回了列表中指定属性的值。

下表 7-1 列出了该接口的各种方法。

表 7-1 AttributeList 接口方法

方法	说明
<code>Public int getLength()</code>	返回列表中所含的属性个数
<code>Public String getName(int index)</code>	返回列表中被索引的属性名称
<code>Public String getType(int index)</code>	根据位置返回列表中指定属性的类型
<code>Public String getType(String name)</code>	根据名称返回列表中指定属性的类型
<code>Public String getValue(int index)</code>	根据位置返回列表中指定属性的值
<code>Public String getValue(String name)</code>	根据名称返回列表中指定属性的值

DocumentHandler 接口

这个接口是 SAX 中最常用的，前面已经比较详细地介绍过了。

`DocumentHandler` 接口接收了 XML 文档事件的通知，是大部分 SAX 应用程序所实现的主要接口，如果应用程序需要被告知基本解析事件，那么它就实现这个接口，并通过 `setDocumentHandler` 方法向 SAX 解析器注册实例。然后解析器使用该实例报告与文档相关的诸如元素开始、结束之类的基本事件。

在这个接口中，事件的顺序反映了文档中信息的顺序，比如像前面所介绍的一样，元素的所有内容将依次出现在 `startElement` 事件和相应的 `endElement` 事件之间。

如果开发者不希望继承整个接口，那么可以继承 `HandlerBase` 类，这个类实现了缺省的功能，通过对该类的实例化就可以获得缺省的处理程序了。

下面表 7-2 给出这个最常用的 SAX 接口中所定义的方法。

表 7-2 SAX 接口的方法

方法	说明
<code>Public void characters(char ch[],int start, int length) throws SAXException</code>	接收字符数据的通知
<code>Public void endDocument() throws SAXException</code>	接收文档结束的通知
<code>Public void endElement(String name) throws SAXException</code>	接收元素结束的通知
<code>Public void ignorable Whitespace(char ch[],int start, int length) throws SAXException</code>	接收元素内容中可忽略的空白通知
<code>Public void processingInstruction(String target, String data)</code>	接收处理指令的通知

方法	说明
throws SAXException	
Public void setDocumentLocator(Locator locator) throws SAXException	接收用于定义 SAX 文档事件源的对象
Public void startDocument() throws SAXException	接收文档开始的通知
Public startElement(String name, AttributeList atts) throws SAXException	接收元素开始的通知

DTDHandler 接口

这个接口接收和 DTD 有关的基本事件的通知。如果应用程序希望接收和 DTD 相关的事件就应该继承该接口。SAX 并没有提供 DTD 的完整的细节，但是这个接口具备了这种功能，所以这个接口可以访问文档中所引用的表示法和未解析的实体。但是绝大多数 XML 文档并不会引用表示法和未解析的实体，所以大多数的 SAX 应用程序不需要使用 DTDHandler 这个接口。

如果 SAX 应用程序需要有关表示法和未解析的实体的信息，那么就要实现 DTDHandler 接口了，并用解析器的 setDTDHandler 方法向 SAX 解析器注册实例，解析器就可以使用该实例向应用程序报告表示法和未解析实体的声明了。

SAX 解析器可以以任何顺序来报告这些事件，无论表示法和未解析实体是按照什么顺序声明的，然而 DTD 事件却必须在文档处理器的 startDocument 事件之后，第一个 startElement 事件之前报告。

下面表 7-3 给出 DTDHandler 接口所定义的方法。

表 7-3 DTDHandler 接口的方法

名称	说明
Public void notationDecl(String name, String publicId, String systemId) throws SAXException	接收表示法声明事件的通知
Public void unparsedEntityDecl(String name, String publicId, String systemId, String notationName) throws SAXException	接收未解析实体事件声明事件的通知

EntityResolver 接口

EntityResolver 接口用于识别实体，当 XML 文档包含外部实体引用时，解析器通常会自动分析 URL，定位并解析相关的文件。

如果 SAX 应用程序需要对外部实体进行处理时，它必须实现这个接口，并且通过解析器的 setEntityResolver 方法向 SAX 解析器注册实例。做了这些处理之后，应用程序才可以在解析器包含外部实体之前截获它们并进行处理。

大多数的 SAX 应用程序都没有必要实现 EntityResolver 接口，但是对于一些使用特殊的输入源构建 XML 文档的应用程序和使用 URI（系统标志符）代替 URL 的应用程序而言，这个接口是非常有用的。

EntityResolver 接口只有一个方法。如表 7-4 所示。

表 7-4 EntityResolver 接口的方法

方法	说明
Public InputSource resolveEntity(String publicId,string systemId) throws SAXException,IOException	允许应用程序解析外部实体

下面用一个实例来说明 EntityResolver 接口是如何识别一个 URI 为 “http://www.somesite.com/entity”的实体。

```

import org.xml.sax.EntityResolver;
import org.xml.sax.InputSource;
public class TheResolver implements EntityResolver
{
    public InputSource resolveEntity(String publicId,string systemId)
    {
        if (systemId.equals("http://www.somesite.com/entity"))
        {
            TheReader reader=new TheReader();
            Return new InputSource(reader);
            '返回一个InputSource对象
        }
        else {return null;}
    }
}

```

ErrorHandler 接口

ErrorHandler 接口是对 SAX 错误进行处理的。如果 SAX 应用程序需要定制错误处理，那么它就要继承这个接口，并且通过解析器的 setErrorHandler 方法向 SAX 解析器注册实例。在实现了这个接口后，解析器就可以通过这个接口报告所有的错误和警告信息了。

这个接口把 SAX 错误分为三类：警告、错误和致命错误。解析器可以使用 ErrorHandler 接口取代抛出异常，应用程序将决定是否针对不同类型的错误和警告抛出异常。但是在调用了 fatalError 方法之后，解析器不再继续提供有价值的信息。

下面来看一下 ErrorHandler 接口的三个方法。

表 7-5 ErrorHandler 接口的方法

方法	说明
Public void error(SAXParseException exception) throws SAXException	接收可恢复的错误的通知
Public void fatalError(SAXParseException exception) throws SAXException	接收致命错误的通知

方法	说明
Public void warning(SAXParseException exception)	接收警告的通知
throws SAXException	

Locator 接口

Locator 接口用来确定源 XML 文档的当前位置，将 SAX 事件和文档位置关联起来。如果 SAX 解析器希望为 SAX 应用程序提供位置信息，那么就要继承这个接口，并且通过文档处理器的 setDocumentLocator 方法将实例传送给应用程序。应用程序可以使用该对象获得 XML 源文档中的其他文档处理器事件的位置。

但是在使用 Locator 接口时要注意，只有在每个文档处理器方法的范围之内，该对象返回的结果才有效。如果试图在其他情况下使用 Locator 接口将会得到不可预计的结果。

下面看一下 Locator 接口的几个方法。

表 7-6 Locator 接口的方法

方法	说明
Public String getPublicId()	返回当前文档事件的公共标志符
Public String getSystemId()	返回当前文档事件的系统标志符
Public int getLineNumber()	返回当前文档事件结束位置的行号
Public int getColumnNumber()	返回当前文档事件结束位置的列号

Parser 接口

Parser 接口是所有 SAX 解析器必须实现的基本接口，它允许应用程序为不同类型的事件注册处理器，并从 URI 或者字符流初始化一次解析。应用程序通过创建解析器实例并调用 parse() 方法解析 XML 文档。

注意，SAX 解析器是可以重用的，但是不可重载。也就是说，一旦一次解析完成，应用程序可以重用解析器对象；但是在一次解析过程中它不能循环调用 parser() 方法。

下面给出了 Parser 接口的所有方法。

表 7-7 Parser 接口的方法

方法	说明
Public void parse(InputSource source) throws SAXException, IOException	解析 XML 文档
Public void parse(String systemId) throws SAXException, IOException	根据 URI 解析 XML 文档
Public void setDocumentHandler(DocumentHandler handler)	允许应用程序注册文档事件处理器
Public void setDTDHandler(DTDHandler handler)	允许应用程序注册 DTD 事件处理器
Public void setEntityResolver(EntityResolver resolver)	允许应用程序注册定制的实体分析器
Public void setErrorHandler(ErrorHandler handler)	允许应用程序注册错误事件处理器

方法	说明
handler)	
Public void setLocale(Locale locale)throws SAXException	允许应用程序为错误和警告请求适当地区域

到这里为止，已经介绍了 SAX 的所有接口。下面介绍 SAX 中定义的类。

7.2.2 SAX 的类

HandlerBase 类

HandlerBase 类是由 SAX 本身提供的缺省基类，这个类实现了 DocumentHandler, DTDHandler, EntityResolver, ErrorHandler 这四个接口的缺省行为。如果开发者只需要继承部分接口时，可以扩展本类。至于它提供的方法可以参见 DocumentHandler, DTDHandler, EntityResolver, ErrorHandler 等接口的方法介绍，在这里不再重复。

InputSource 类

利用 InputSource 类，SAX 应用程序可以将有关的输入源的信息封装在一个对象之中，也就是说，InputSource 对象代表了 XML 文档或它所引用的外部实体的容器。通常情况下，应用程序将 InputSource 类实例化，并在其中保存输入信息的位置，解析器会询问它从何处获取输入信息。

InputSource 对象通过如下的三种方式为解析器提供输入：URI（系统标志符）、Reader（传递统一码字符流）以及 InputStream（传递未解析的字节流）。SAX 解析器将利用 InputSource 对象确定如何读取 XML 输入信息。如果有字符流，解析器将直接读取该字符流；如果没有字符流，而存在字节流，解析器将使用该字节流；如果两者都不存在，解析器则试图打开 URI 资源。

应用程序可以通过两种方式将输入源传递给解析器：作为 Parser.parse 方法的参数或者作为 EntityResolver.resolveEntity 方法的返回值。

InputSource 类提供了四种构造器，如下表 7-8 所示。

表 7-8 InputSource 类的构造器

名称	说明
Public InputSource()	零参数缺省构造器
Public InputSource(String systemId)	使用系统标志符创建新的输入源
Public InputSource(InputStream byteStream)	使用字节流创建新的输入流
Public InputSource(Reader characterStream)	使用字符流创建新的输入流

InputSource 类提供了如下的方法。如表 7-9 所示。

表 7-9 InputSource 类的方法

名称	说明
Public void setPublicId(String publicId)	设置本输入源的公共标志符

名称	说明
Public String getPublicId()	获取本输入源的公共标志符
Public void setSystemId(String systemId)	设置本输入源的系统标志符
Public String getSystemId()	获取本输入源的系统标志符
Public void setByteStream(InputStream byteStream)	设置本输入源的字节流
Public InputStream getByteStream()	获取本输入源的字节流
Public void setEncoding(string encoding)	如果编码已知, 设置字符编码
Public String getEncoding()	获取字节流或者 URI 的字符编号
Public void setCharaterStream(Reader charaterStream)	设置本输入源的字符流
Public Reader getCharaterStream()	获取本输入源的字符流

SAXException 类

SAXException 类用于表示解析器或者应用程序在处理过程中检测到的错误, 可以封装普通的 SAX 错误或者警告。应用程序的开发者能够通过创建该类的子类扩展它的功能, SAX 处理器可以抛出该异常或者作为它的子类的异常。

如果应用程序需要传递其他类型的异常, 则必须将这些异常包含在 SAXException 或者从 SAXException 派生出来的异常中。

SAXException 类有三个方法, 如表 7-10。

表 7-10 SAXException 的方法

名称	说明
Public String getMessage()	返回本异常的详细信息
Public Exception getException()	如果存在内嵌的异常, 则返回该异常
Public String toString()	将本异常转换为字符串

对于普通的错误或者警告 SAXException 类已经可以处理, 但是要包含有关 XML 文档中特定位置的信息的话, 我们应该使用 SAXParseException 子类。

SAXParseException 类

正如上面提到的, SAXParseException 类扩展了 SAXException 类, 除了具备 SAXException 的基本功能之外, SAXParseException 类能够获取 XML 源文档中出现错误的位置的参数, 对于应用程序检测到的错误, 可以通过 Locator 对象获得该信息。

实际上, 应用程序不需要抛出异常, 它可以简单地读取其中的信息, 采取操作。

SAXParseException 类提供了四个构造器, 分别如下表 7-11。

表 7-11 SAXParseException 类的构造器

名称	说明
Public SAXParseException (String message,Locator locator)	从消息和 Locator 中创建新的 SAXParseException
Public SAXParseException (String message,Locator locator , Exception e)	在 SAXParseException 中包含当前的异常

名称	说明
Public SAXParseException (String message,String publicId, String systemId,int lineNumber, int columnNumber)	创建新的 SAXParseException
SAXParseException (String message,String publicId, String systemId,int lineNumber, int columnNumber, Exception e)	创建新的包含异常的 SAXParseException

SAXParseException 类所提供的方法如下表 7-12。

表 7-12 SAXParseException 类的方法

名称	说明
Public String getPublicId()	获取产生异常的实体的公共标志符
Public String getSystemId()	获取产生异常的实体的系统标志符
Public int getLineNumber()	返回产生异常的文本的结束位置的行号
Public int getColumnNumber()	返回产生异常的文本的结束位置的列号

此外 SAX 还包含了三个辅助类：

- ☒ AttributeListImpl 类是 AttributeList 类的实现
- ☒ LocatorImpl 类是 Locator 接口的实现
- ☒ ParserFactory 类允许开发者在允许时加载以参数标志的解析器

限于篇幅，这些类就不再详细介绍了。下一节中，我们将用实例来说明以上接口和类的用法。

7.3 SAX 实例

下面就用实例来说明 SAX 接口在应用程序中的用途。

在这里选用了 SUN 公司的 XML 解析器 JAXP (Java API for XML Parsing)，我们已经知道 SAX 解析器会在以下情况下产生事件：在文档开始和结束时，在一个元素开始和结束时，或者它在一个元素中找到字符时，以及其他若干点。所以程序员要做的就是编写代码来处理每个事件，以及处理从解析器获得的解析信息。

7.3.1 元素属性处理

第一个例子是这样的，在年终的时候统计一个商店的月平均营业额。这个商店的每个月的营业额都存储在一个 XML 文件 money.xml 中。

程序清单 7.2:

```
<?xml version = "1.0"?>
  <money>
```

```
<month id="1" shop="myshop" price="8000.00"></month>
<month id="2" shop=" myshop " price="9000.00"></month>
<month id="3" shop=" myshop " price="7000.00"></month>
<month id="4" shop=" myshop " price="8000.00"></month>
<month id="5" shop=" myshop " price="7800.00"></month>
<month id="6" shop=" myshop " price="7800.00"></month>
<month id="7" shop=" myshop " price="7900.00"></month>
<month id="8" shop=" myshop " price="7900.00"></month>
<month id="9" shop=" myshop " price="7000.00"></month>
<month id="10" shop=" myshop " price="7800.00"></month>
<month id="11" shop=" myshop " price="11000.00"></month>
<month id="12" shop=" myshop " price="4500.00"></month>
</money>
```

首先引入所需的类包：

```
import java.io.*;
import org.xml.sax.*;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
```

这些类包的作用在前一节都已经详尽介绍过了，不再赘述。

接下来创建一个解析器实例：

```
SAXParserFactory factory = SAXParserFactory.newInstance();
'创建了一个SAXParserFactory的实例factory
try {
    out = new OutputStreamWriter (System.out, "UTF8");
    '建立输出流
    SAXParser saxParser = factory.newSAXParser();
    '由这个类工厂创建了一个分析器实例
    saxParser.parse( new File(argv [0]), new search() );
    '通过该分析器实例处理分析事件
}
catch (Throwable t) {
    t.printStackTrace ();
}
```

在 month 元素开始的时候，就将变量 count 加 1，并且将 price 属性值累加到变量 totalPrice 中，那么到了最后一个元素就可以计算出该年度的总的营业额。

```
public void startElement (String name, AttributeList attrs)
    throws SAXException {
```

```

        if(name.equals("month"))
        {
            count=count+1;
            ‘月数累加
            double price = new Double (attrs.getValue("price")).doubleValue() ;
            ‘把字符串转换为double类型
            totalPrice+=price;
            ‘总的营业额相加
        }
    }
}

```

最后在文档结束的时候计算并输出月平均营业额。

```

public void endDocument()
{
    averagePrice=totalPrice/count;
    ‘月平均营业额
    System.out.println("the avarage money ervry month is "+averagePrice);}
}

```

下面就看一下该程序的运行结果，如图 7-3 所示。

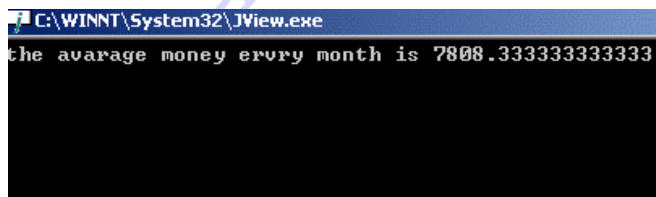


图 7-3 程序运行结果

最后给出该程序的清单。

程序清单 7.3:

```

import java.io.*;
import org.xml.sax.*;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
public class search extends HandlerBase
{
    private double totalPrice=0.0;
    private double averagePrice=0.0;
    private int count=0;
    public static void main (String argv [])throws Exception
    {
        if (argv.length != 1) {
            System.err.println ("Usage: cmd filename");

```

```

        System.exit (1);
    }
    SAXParserFactory factory = SAXParserFactory.newInstance();
    try {
        out = new OutputStreamWriter (System.out, "UTF8");
        SAXParser saxParser = factory.newSAXParser();
        saxParser.parse( new File(argv [0]), new search() );
    }
    catch (Throwable t)
    {
        t.printStackTrace ();
    }
}

static private Writer out;
public void startElement (String name, AttributeList attrs)
    throws SAXException {
    if(name.equals("month"))
    { count=count+1;
      double price = new Double (attrs.getValue("price")).doubleValue() ;
      totalPrice+=price;
    }
}

public void endDocument()
{
    averagePrice=totalPrice/count;

    System.out.println("the avarage money ervry month is "+averagePrice);}
}
}

```

7.3.2 SAX 字符处理

上面这个例子用到了 `startElement()` 和 `endDocument()` 这两个重要的方法，但是总体上来说比较简单。

下面就让我们再看一个例子。这个例子要统计一个顾客在一个商店花费的总额以及每次的平均消费。我们把顾客的每次消费信息都保存在一个 XML 文档 `guest.xml` 中。

程序清单 7.4:

```

<?xml version="1.0"?>
<guest>
    <spent name="weng">1213</spent>
    <spent name="li">3213</spent>
    <spent name="rui">213</spent>
    <spent name="weng">6213</spent>

```



```

    <spent name="weng">2088</spent>
    <spent name="zhang">1013</spent>
</guest>

```

如果要统计的是顾客 weng 的如上的参数，那么首先要在 spent 元素开始的时候判断 name 属性值是否为 weng，如果是的话就令变量 flag 为 1，并且将表示该顾客光临的次数的变量 count 加 1。代码如下所示。

```

public void startElement (String name, AttributeList attrs)
    throws SAXException {
    if(name.equals("spent"))
    {
        String guest = attrs.getValue("name");
        if(guest.equals("weng"))
        {
            flag=1;
            ‘表示为我们要处理的信息
            count=count+1;
            ‘次数加1
        }
    }
}

```

然后在事件 characters () 发生以后，判断标志变量 flag 是否为 1，如果是的话就表示目前处理的是顾客 weng 的数据，读取该数据并且转换为 double 类型，保存在 price 变量中，并且累加到表示总消费额的 totalPrice 变量上，最后将标志变量 flag 重新置为 0。代码如下。

```

public void characters (char buf [], int offset, int len)
    throws SAXException
{
    if (flag==1)
    {
        String s = new String(buf, offset, len);
        ‘读取字符数据块
        double price = new Double(s).doubleValue();
        ‘转换为double类型
        totalPrice+=price;
        ‘总的消费额累加
        flag=0;
        ‘重置flag为0
    }
}

```

最后在文档结束事件发生的时候，输出总的消费额和每次平均消费。

```
public void endDocument()
{
    System.out.println("guest weng spent money in our shop is "+totalPrice);
    System.out.println("guest weng spent money in our shop every time
                        in average is "+totalPrice/count);}
}
```

来看一下该程序运行的结果，如图 7-4 所示。

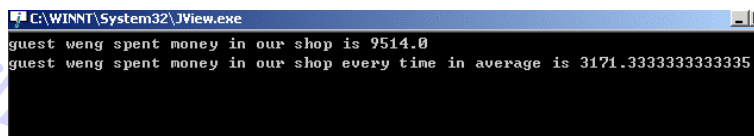


图 7-4 程序运行结果

下面给出整个程序的清单。

程序清单 7.5:

```
import java.io.*;
import org.xml.sax.*;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class cal extends HandlerBase
{
    int flag=0;
    private double totalPrice=0.0;
    private int count=0;
    public static void main (String argv [])throws Exception
    {
        if (argv.length != 1)
        {
            System.err.println ("Usage: cmd filename");
            System.exit (1);
        }
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            out = new OutputStreamWriter (System.out, "UTF8");
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( new File(argv [0]), new cal() );
        }
        catch (Throwable t)
        {
        }
```

```

        t.printStackTrace ();
    }
}
static private Writer out;
public void startElement (String name, AttributeList attrs)
    throws SAXException
{
    if(name.equals("spent"))
    {
        String guest = attrs.getValue("name");
        if(guest.equals("weng"))
        {
            flag=1;
            count=count+1;
        }
    }
}
public void characters (char buf [], int offset, int len)
    throws SAXException
{
    if (flag==1)
    {
        String s = new String(buf, offset, len);
        double price = new Double(s).doubleValue();
        totalPrice+=price;
        flag=0;
    }
}
public void endDocument()
{
    System.out.println("guest weng spent money in our shop is "+totalPrice);
    System.out.println("guest weng spent money in our shop every time
        in average is "+totalPrice/count);
}
}

```

到此为止对 SAX 的介绍就结束了，大家可以动手编写几个实例来熟悉这种技术。

7.4 小 结

本章讨论了 DOM 和 SAX 的区别，读者可以根据实际情况和编程要求来作出选择。SAX 接口的最大特性是基于事件，因此本章详细地介绍了 SAX 的各个类和接口。读者应该熟悉诸如 startDocument(), endDocument(), startElement(), endElement()和 characters()等重要的

方法。最后本章还给出了两个实例，希望读者详细读解。

后面一章将介绍 XML 与数据，该章将用到前面介绍过的很多知识点，希望大家用心解读。

北京和信电子出版社

第 8 章 XML 与数据

本章导读

很难想象离开了数据库后，动态 XML 该如何表现。事实上，XML 与数据库的关系越来越密切。从存储在不同介质中的数据自动的生成 XML 文档以及从不同的数据存储交换信息，将成为未来面向信息的 Internet 的主要特点。信息世界的多极化必将使动态 XML 文档不断的增加：它们用于加载任何东西，包括图像和声音；还用于越来越多的信息交换。这就必然需要多人的操作，从这点上看，最好的方法就是利用数据库。

本章介绍给大家的内容包括 XML 的存储、数据的交换以及一个较大的实例。

XML 存储部分主要说明文件系统的局限性以及借助于数据库的必要性和这样做的优势。

数据的交换主要讲述如何在异质通信之间交换数据，这一点是很重要的。不管是服务器到服务器之间还是客户到服务器之间都需要一种更有效的手段来进行联系，这种方法就是 XML。在后面会说明为什么 XML 更加有效。

在实例中，仍旧是用信息建模一章中的音乐网站例子来建立一个数据库系统，并且通过 XML 来传送信息。下面就来开始本章的学习吧。

8.1 XML 的存储

XML 的存储是以文件形式存在的，能否直接利用文件形式来存储信息呢？能不能找到一种更加有效的方法呢？这些将是本小节讨论的内容。

8.1.1 文件系统的局限性

可行性研究主要是针对于保存在系统中的 XML 文件来说的，这些文件因为以后要用到，或者还有其他的用途，所以必须把它们保存起来，而这些存储的介质通常就是文件系统。到目前为止，我们涉及的 XML 文档都是普通文件系统中的文件，它们的保存方式和保存一般的字处理文件比如象 Word、Excel 等的方法没什么不同，而且 XML 甚至可以用任何的编辑器来编辑它们。从普遍性来考虑这当然是 XML 的一个优点，但是单一的文件系统往往给应用带来不便。事实上在存储为文件系统的 XML 中，内容的更新存在问题，当然最好的办法就是使用特殊的处理软件，但是这样的软件目前还比较少，所以目前这个阶段 XML 的更新需要花费不少的精力。假如在一个文件中不断的加入新的内容（对于一个大的系统来说这通常是不可避免的事），对于一个单一文件的处理信息量的不断增加将会给我们带来很大的麻烦；还有对于文件，假如它的内容足够小并且已经不经常使用的时候，往往会将它丢在一边甚至一不小心就删除了，这就造成了信息的丢失。

通常称信息的保存为持续性，也就是说这些信息在退出系统以后仍然是存在的，以后还可以被使用。假如在以后的使用中由于文件的特殊性，不得不需要多个人同时对一个 XML 文档进行操作的时候，例如对于一个简单的订单处理文档，是否可以很顺利的实现呢？在这里就需要说明文件系统的局限性了。

对于一个文件系统来说，某些方面可能使用的很好，但在一些关键性的应用方面，我们不得不面对很多的尴尬。下面把文件系统的局限性总结一下：

☒ 文档大小

前面提到过文档的大小问题。程序员可能对小的 XML 文档有很好的处理能力，但是一旦 XML 的信息变大，那么传递 XML 文件就会变得非常的不实用。这不仅仅是因为它大，更主要的是很难维护这样一个文档。

☒ 并发性

我们可能想在文档的不同部分同时进行修改，或者有时候需要不同的人对同一文档的不同部分在不同时间更新。对一个单一的文档来说，可能很难实现这一点，因为在通常的系统中，一个文件在一个时间只能有一个人可以进行处理。假如两个人需要对同一个文档进行操作的话，那么不可避免的就要排队等候。如果试图在同一时刻对同一文档进行不同的操作，通常会导致信息的丢失。

☒ 编辑工具

通常需要处理文档的不同部分，比如对数据的维护和更新等。当然这些操作不能对其他的部分产生影响，这通常需要一个合适的处理数据的工具。目前常用的处理 XML 文档的编辑器很难做到这一点。

☒ 安全性

对信息系统来说，有时候不得不考虑安全性问题。某些信息可能只对某些人开放的，而某些信息不让某些人修改等，这些问题的实现对于文件来说恐怕很难实现（文件系统的安全性只到达文件级）。

☒ 数据的集成

有时候需要集成来自不同部分的数据或者信息，这些数据存在于不同的介质甚至服务器上，对于文件系统来说，如何提取数据又如何集成是比较困难的，因为文件的存储以二进制形式，很难有特征来提取必须的信息。

通过上面的分析，我们基本上了解了需要加强的一些功能，这些功能包括：要能够很轻松的处理大的 XML 文档；能够允许许多人在同一时刻处理同一文档；需要一个更合适的编辑工具来处理数据；需要设置文档的安全性问题；能够很容易的集成不同数据源中的数据等。

8.1.2 解决方案

XML 服务器

最先想到的解决方案是通过特殊的 XML 服务器来分离文档和节点。事实上，对于一个文档，当需要不同的人对它进行操作时，系统会根据要求来分离文档，把不同需要的节点提取出来，组成新的文档然后把它显示出来由不同的人来更新和维护。这确实是一个可行的方法，比如对于下面的一段结构：

程序清单 8.1：

```
<?xml version="1.0" encoding="gb2312"?>
<客户名单>
```

```
<客户 种类="个人">
  <编号>KH-1365</编号>
  <客户地址>DR-1876</客户地址>
  <姓名>任建兴</姓名>
  <订单>DD-345</订单>
</客户>
<客户 种类="公司">
  <编号>KH-0023</编号>
  <客户地址>DR-123</客户地址>
  <单位>南北湖旅游局</单位>
  <订单>DD-033</订单>
</客户>
</客户名单>
```

有了这段程序，完全有理由把两类不同的用户分离开来，并且可以创建两个更完美的 XML 文档。可用下面的结构来说明个人用户的信息：

```
<客户 种类="个人">
  <编号>KH-1365</编号>
  <客户地址>DR-1876</客户地址>
  <姓名>任建兴</姓名>
  <订单>DD-345</订单>
</客户>
```

并且公司客户的信息可以这样来表示：

```
<客户 种类="公司">
  <编号>KH-0023</编号>
  <客户地址>DR-123</客户地址>
  <单位>南北湖旅游局</单位>
  <订单>DD-033</订单>
</客户>
```

上述这两段代码都是以一个完整的 XML 文档的形式在内存中存储的。它们可以在更新的时候被提取出来并且在内存中生成，在更新结束后又重新写回到原来的文件中。这样通过处理节点，至少已经为两个人同时创建了对一个 XML 文档进行操作的可能性——通过允许它们处理分离的节点。用户的不同级别也可以通过系统来辨别，当然还需要一个验证的系统来确认用户的身份，是游客还是管理员，从而赋予不同的权力。

至于系统的实现，最好的方法就是利用 DOM 和脚本语言（ASP，JSP，PHP 等）来控制节点，这样做需要一定的时间，因为对于不同的用户可以指定不同的节点集来进行处理：以一种有效的 XML 文档的形式，利用 DOM 来做需要很多细节上的问题。而且对于大型的 XML 文档来说，可能在对节点进行处理的时候需花费很大的计算时间。

数据库

或许可以考虑一些中间的技术来实现在数据库中存储文档。前面已经确定，使用文件系统来处理文档不是一个用来解决安全性和用户数量的有效方法，而且我们也看到，在节点级进行操作时，可能需要某种方法来解决。而数据库恰恰能够很好利用该方法，而且在存储方面它也有比其他系统更多的优点。

下面将数据库和文件系统进行比较。

首先，从大小来说，数据库通常可以处理海量的数据信息。因为信息可以以一种很小的单位来访问，例如可以把客户编号或音乐 CD 的编号当作索引用来导航，这样就很容易控制。

其次，从并发性来说，数据库一点都不担心这个问题。数据库本来就被设计为允许多个用户同时处理信息。例如，大部分的产品都同时允许一个或多个用户订购。

再次，从安全性来说，数据库有一个很好的控制，通常允许不同级别的处理。例如在数据库系统中，可以很容易实现这样一个功能：一个用户可以看到某些内容，但是不能修改它们；而另一个用户却有权力，能够修改它们。

最后，从集成性来说，数据库对于可重用信息的共享非常适合，而且还很容易利用 SQL 语句来访问、提取不同的信息。

看来数据库技术确实很适合我们的需要。而谈到数据库，就要了解两类基本的数据库技术：一类是关系数据库；另一类是面向对象的数据库技术。关系数据库和面向对象数据库两个都允许控制小数量的信息。下面对它们分别作一下介绍。

1. 关系数据库

关系数据库被称为 DBMS。把一个 XML 文档作为关系型数据库管理系统的一个字段来进行存储是很直接的，并且任何数据库产品都可以实现这个功能。然而一旦存储完成，文件就不再是一个文本字符串：XML 的存储与在数据库中保存图像和文本文档采用相同的方法。

特别强调的是，Oracle 在它的 8i 产品中增加了一个特性，允许 XML 文档被存在一个字段中，也就是允许按 XML 进行查询。这个功能是通过扩展普通的 SELECT 语句来完成的。这点对于一个强大的文档存储非常有利：一个表中的一条记录可以有一个字段来保存 XML 文档，而其他的字段可以用来保存文档的信息，例如是否被加锁、谁最后修改它等。

与一般的开发项目相比，这些特性最可能被用于建立一个自己的全 XML 服务器。程序员可以通过控制关系数据库中的信息，生成 XML 文档用于传送，或者说把 XML 的内容写入数据库。而对于系统的更新，关系数据库也有一套完整的方法来实现，这些功能可以通过不同的 SQL 语句查询或通过其他的途径来实现。这里不再赘述。

2. 面向对象的数据库

面向对象的数据库被称为 OODBMS。面向对象的数据库在某些方面确实有很大的优势，而且现在大部分的商家也开始将它们的数据库产品用于非常强大的 XML 文档仓库。其中的一些产品包括象 DynaBase 等。与关系数据库相比，可能它的速度比不上关系数据库，但在使用面向对象编程技术的时候，可以让信息更易于管理，更直观一点的说，可以更好的模拟节点分级。

下面再来看看 OO 方法和 XML 的关系。事实上在创建一个 XML 文档的时候，往往使用 DOM 的方法，而使用 DOM 要求大量的调用 createElement()和 setAttribute()方法。这样做可能会发生很多错误。使用 OO 方法，就可以创建更加稳定的对象。例如下面的一段 XML 代码：

程序清单 8.2:

```
<?xml version="1.0" encoding="gb2312"?>
<今日预订 date="2001-09-23" xmlns:dt="urn:schema-microsoft-com:datatypes">
  <预订>
    <客户编号>KH-0786</客户编号>
    <音乐编号>CD-0987</音乐编号>
    <数量 单位="张" dt:type="i8">1</数量>
    <付款方式>信用卡</付款方式>
  </预订>
</今日预订>
```

使用 OO 方法来创建上面的对象可以使用 JavaScript，代码如下：

程序清单 8.3:

```
root=oParser.createElement("今日预订");
root.setAttribute("data", "2001-09-23");
root.setAttribute("xmlns:dt", "urn:schema-microsoft-com:datatypes");

temp=oParser.createElement("预订");
child1=oParser.createElement("客户编号");
child1.text="KH-0786";
temp.appendChild(child1);
child1=oParser.createElement("音乐编号");
child1.text="CD-0987";
temp.appendChild(child1);
child1=oParser.createElement("数量");
child1.text="1";
root.setAttribute("单位", "张");
root.setAttribute("dt:type", "i8");
temp.appendChild(child1);
child1=oParser.createElement("付款方式");
child1.text="信用卡";
temp.appendChild(child1);
root.appendChild(temp);
```

可以使用函数的方法来简化上面的操作，但是面向对象的好处是处理数据的模型而不仅仅是节点。目前可以参考的面向对象数据库有 eXcelon1.1 和 POET，这里就不再细讲。读者有兴趣的话可以参考相关的一些网站。

8.1.3 关系数据库

实际上，目前广泛使用的仍然是关系数据库，所以下面详细介绍有关知识。因为后面介绍的内容中涉及到的数据库都是关系数据库。

关系数据库使用我们熟悉的行和列的方法来存储数据，因为这样可以表示许多现实世界的问题，并且对很多这样的问题能快速做出响应。

在关系数据库中最熟悉的就是表，典型的表的形式就如图 8-1 所示。

音乐编号	专辑名	演唱者	价格	说明
CD-1023	盛夏的果实	莫文蔚	¥ 49.99	莫文蔚的新专辑
CD-1024	冷酷到底	羽泉	¥ 45.99	羽泉真情演绎
CD-0212	男人哭吧哭吧不是罪	刘德华	¥ 12.00	低沉的音调、迷人的魅力

图 8-1 典型的关系表格

作为关系数据库，最主要的就是如何来表述事物模型中的关系。事实上，关系数据库在处理二维数据方面有很好的效果，尤其在表述一对一、一对多的关系上，关系数据库可以很好的控制它们。对于多对多关系，关系数据库也可以通过建立不同的表并建立联系来达到控制的目的。从这点上看，文件系统是显然无法相比的。

要了解关系数据库，就有必要了解一些基本的概念，包括表和查询。当然最主要的是关系数据库的查询机制，也就是通常所说的 SQL 句法。

先来看表的结构，本书的数据库建立在 Access 数据库系统上。打开 Access，建立一个叫 Music 的数据库，然后就在数据库中建立我们需要的表格，上面的图 8-1 所示就是一个典型的表。

表的建立是很容易的事，因为数据的录入并不是一件很费力的事，关键是如何定义表中各个字段的限制。很多人习惯于使用设计器建立表，因为这样建立的表格可以把原先设计的字段限制表现在表中。事实上，XML 文件中的各个文档标记内容都是有规定的，不管是字符串的还是数值的，当我们把 XML 文件与数据库中的内容联系起来时，就不得不考虑它们之间的兼容性，否则就会引起混乱。比如以表 8-1 为例。

表 8-1 一张关系数据表

列名	数据类型	是否允许空值
音乐编号	字符串	否
专辑名	字符串	否
价格	货币	否
演唱者	字符串	否

以后的讨论中将会很详细的讨论 XML 模式与数据库中字段之间的关系。下面来详细的说明一下数据库中的查询语句，也就是 SQL 语句。关于 SQL 语句，它的核心就是 SELECT 查询语句。一个简单的查询语句如下面的形式：

SELECT * FROM 音乐

在这里*表示检索所有的行以及列，即整个数据库。当系统运行了该语句后，返回的是一个记录集，里面记录了所有符合要求的数据列。当要把它作为结果输出时，就可以象数据库中的记录一样一条一条的输出。

也可以选择特定的列，比如下面的形式：

```
SELECT 专辑名 FROM 音乐
```

一般的查询是有条件的，也就是说要求查询特定的记录，这就要使用 WHERE 子句。WHERE 子句是很有用的查询子句，比如说要寻找所有由莫文蔚演唱的专辑的名称，可以这样来表示：

```
SELECT 专辑名 FROM 音乐 WHERE 演唱者='莫文蔚'
```

使用数据库的另一大优势就是可以集成不同表中的数据生成 XML 文档，可以说数据库提高效率的最主要的途径就是使用连接，而这一点正是依靠 SQL 语句来实现的。通常的一个数据库中都有很多有关系的表，这些表的设计按照范巴斯范式来建立（理论上的数据库表都应该满足第三范式）。它们之间的关系通常就是靠主键和外键的关系来连接的。

下面来看看在 XML 中经常出现的几种关系，这几种关系中最常见的当属一对一和一对多关系，比如说一张音乐专辑有一个价格，每个价格需要一个货币单位。前面介绍的内容中只涉及到一个表，那就是音乐表，事实上在整个 Music 数据库中还有很多张其他的表，这些表包括销售情况表、定购表、定购人表、处理人表等等。而这几张表就是通过主键和外键联系起来的。为了说明连接，这里给出几个表的设计。下面的一个表 8-2 用于保存订单（定购表）。

表 8-2 定购表的结构

列名	数据类型	是否允许空值
定购号	字符串	否
音乐编号	字符串	否
数量	数字	否
日期	日期/时间	否
备注	备注	是

再看下面的表 8-3，这个表的用于说明定购人的情况：

表 8-3 定购人表的结构

列名	数据类型	是否允许空值
定购号	字符串	否
定购人	字符串	否
详细地址	字符串	否
邮政编码	数字	否
备注	备注	是

对于一对一或一对多关系，可能只需要建立一张表就可以了。但是对于多对多关系，就不能把所有的多对多关系中的内容全部写到同一张表中去了，这就违背了前面讲到的范式要求。多对多关系简单来说，就是一个订购人可以订购很多张专辑，同时同一张专辑又可以被很多个订购人订购。上面的关系需要建立三个表来表示这种多对多的关系。在使用中就要用到连接，也就是有时候将需要的数据分散在两个表，或者将给定限制条件的内容存于另外一张表中。

比如说需要寻找所有订购了《男人哭吧哭吧不是罪》专辑的订购人的信息，就可以这样来表示：

```
SELECT PER.订购人, PER.详细地址, PER.邮政编码 FROM 音乐, 订购, 订购人 AS PER
WHERE 订购.订购号=订购人.订购号 AND
音乐.音乐编号=订购.音乐编号 AND
音乐.专辑名='男人哭吧哭吧不是罪'
```

再比如要得到订购人翁宇翔订购的音乐专辑名和数量，可以用下面的语句来说明：

```
SELECT MUS.专辑名, BOK.数量 FROM 音乐 AS MUS, 订购 AS BOK, 订购人
WHERE 订购.订购号=订购人.订购号 AND
音乐.音乐编号=订购.音乐编号 AND
订购人.订购人='翁宇翔'
```

可以看出利用了 SQL 语句，完全可以在输入条件的情况下生成我们需要的 XML 文件。而这个输出的形式可以利用 ASP 来实现。后面一节将会给大家介绍数据库与 XML 文档之间的信息交换问题。

当然关系数据库也有它的局限性，这主要表现在关系数据库在输出成 XML 文档时对于父子关系的控制。在 XML 文档中，父子关系是相当严格的，而且它们的关系一般都是由 XML 模式来确定的，相对于数据库，数据库可能无法模拟在 XML 中所有的父子关系，或者说数据库模拟父子关系的能力是有限的。

在上面列出的几个表中，我们定制的数据库都是相当于 XML 文档的第三级标记来说的，也就是说它们是最底层的节点。但是可能会遇到更加复杂的情况，不得不要求我们在数据库中列出一些父节点，而这些父节点和子节点的共同存在必将影响 XML 文档的输出。举个例子，比如说在 XML 文档中我们可能严格规定在 C 元素只能包含在 B 元素和 A 元素下，在数据库中我们是通过设置主键和外键来联系父子关系的，但是在一个表中我们只能设置一个主键和一个外键，这相当于我们不能模拟有多个父亲的节点的父子关系，比如下面的情况：

```
<A>
  <B>
    <C/>
  </B>
<C/>
```


这样看来关系数据库虽然能够很好的模拟节点，但是它不能很好的处理节点的关系。但是我们还不能放弃使用关系数据库，因为在大多数的情况下，我们遇到的都是单一的父子关系，在这些情况下，关系数据库都表现得很出色。

事实上在模拟节点的时候我们可以指定一个这样的标准或规则，在这个规则下，我们按照 XML 严格规定了一些数据库中的要求，也就是说我们要按照 XML 的模式来实现数据库的存储。在后面的一些内容中，我们将会给大家介绍这些规则。

8.2 数据交换

本节介绍的内容主要是 XML 与数据库之间的信息交换。下面先介绍为什么在服务器之间使用 XML 传送以及实现 XML 的可能性；然后我们继续利用 DOM 和 XSL 方法来深入的讨论数据库和 XML 文档之间的信息传送；最后我们给大家介绍查询。

8.2.1 数据传送

利用 XML 传送

也许大家都有这样的疑问，为什么费力的用 XML 来传送文件，难道不能直接使用各种数据库文件来传送数据吗？或者说直接在客户端向服务器发送消息以获取服务器上的数据库中的信息；我们可以直接发送数据库文件；也可以直接利用 Internet 通信处理数据库。在以上的条件下，又何必费力的把数据库文件转换为 XML 文档的形式。事实上，我们必须了解一个概念，就是现在的数据库系统之间是通过什么联系起来的，这就是通常所说的 ODBC。

ODBC (Open Database Connectivity, 开放数据库连接) 是微软公司开放服务结构 (WOSA, Windows Open Services Architecture) 中有关数据库的一个组成部分，它建立了一组规范，并提供了一组对数据库访问的标准 API (应用程序编程接口)。这些 API 利用 SQL 来完成其大部分任务。ODBC 本身也提供了对 SQL 语言的支持，用户可以直接将 SQL 语句送给 ODBC。

一个基于 ODBC 的应用程序对数据库的操作不依赖任何 DBMS，不直接与 DBMS 打交道，所有的数据库操作由对应的 DBMS 的 ODBC 驱动程序完成。也就是说，不论是 FoxPro、Access 还是 Oracle 数据库，均可用 ODBC API 进行访问。由此可见，ODBC 的最大优点是能以统一的方式处理所有的数据库。

那么利用 ODBC 来通信有什么问题呢？这些问题可以用下面的话来概括：

- ☒ 只有安装或理解 ODBC 的系统才能够接收消息
- ☒ 很多防火墙不允许 ODBC 的信息交换
- ☒ ODBC 很容易被黑客攻击。这些黑客可能发送一些需要被服务器验证的未经授权的交易。

再来考虑利用 XML 来传送信息。首先就是可以去除客户端对于 ODBC 的依赖。这一点可以在客户端的通信管道两端加上 XML 接口来实现。同样，如果以某种方式封装 XML，并且通过 80 端口 (正常的 HTTP 端口) 传送它，则可以解决防火墙的阻挡问题。实际上，

微软的一个新的技术，SOAP（简单对象访问协议）就是这么做的。

这样看来利用 XML 来传送数据确实有它的好处，那么用 XML 来传送数据是否有技术可以实现呢？事实上，现在的很多技术都支持 XML 与数据库的连接，这些技术包括像 ASP，DOM，SOAP，XML-RPC 等，下面就来看看利用 ASP 来实现数据库与 XML 的信息传送。

利用 ASP 实现

利用 ASP 来实现 XML 与数据库的联系，必须首先建立一个数据库。实现起来则要从两个方面来做，一方面是如何把数据库中的信息提取出来生成所需的 XML 文档，另一方面就是怎样把一些重要的信息写入到数据库中。下面就用程序来实现这两点。先来看数据库到 XML 文档的信息传送，这个实现应该还是比较简单的。下面就是一个最简单的 XML 和数据库联系的例子。

程序清单 8.4:

```
<%Language=VBScript%>
<%
    response.ContentType="text/xml"
    dim rs
    set rs=GetMdbRecordset("music.mdb", "音乐")
    if rs is nothing then
        response.write "GetMdbRecordset 调用失败"
        response.end
    end if
%>
<?xml version="1.0" encoding="gb2312"?>
<库存>
    <%
        Do While Not rs.EOF
        %>
        <音乐 种类="<%=rs("种类")%>">
        <音乐编号><%=rs("音乐编号")%></音乐编号>
        <专辑名><%=rs("专辑名")%></专辑名>
        <演唱者><%=rs("演唱者")%></演唱者>
        <价格><%=rs("价格")%></价格>
        <% if rs("说明") <> Empty then %>
            <简介><%=rs("说明")%></简介>
        <% end if %>
        <% if rs("备注") <> Empty then %>
            <备注><%=rs("备注")%></备注>
        <% end if %>
        </音乐>
    <%
        rs.MoveNext
    %>
```



```

Wend
%>
<库存>
<%
rs.close
%>

```

上面显示的 XML 文档可能看上去像下面的形式：

程序清单 8.5:

```

<?xml version="1.0" encoding="gb2312"?>
<库存>
  <音乐 种类="CD">
    <音乐编号>CD-1023</音乐编号>
    <专辑名>盛夏的果实</专辑名>
    <演唱者>莫文蔚</演唱者>
    <价格>¥12.00</价格>
    <简介>莫文蔚的新专辑</简介>
  </音乐>
  <音乐 种类="CD">
    <音乐编号>CD-1024</音乐编号>
    <专辑名>冷酷到底</专辑名>
    <演唱者>羽泉</演唱者>
    <价格>¥49.99</价格>
    <简介>羽泉真情演绎</简介>
  </音乐>
  <音乐 种类="MP3">
    <音乐编号>MP3-0212</音乐编号>
    <专辑名>男人哭吧哭吧不是罪</专辑名>
    <演唱者>刘德华</演唱者>
    <价格>¥45.99</价格>
    <简介>低沉的音调、迷人的魅力</简介>
  </音乐>
</库存>

```

下面该来看如何把 XML 信息写入数据库中去了，简单的方法就是利用节点的控制来做。在前面给出的音乐网站中，每天都有人填写订单，对于这些订单，当然不可能全部都保存在 XML 文档中。而只需要隔一段时间把这些信息写入数据库中，以便做存根用。下面来看如何把这些已经处理过的信息写入数据库中。XML 文件的形式如下所示。

程序清单 8.6:

```

<?xml version="1.0" encoding="gb2312"?>
<今日预订>
  <预订 date="2001-09-23">

```

```

    <客户 种类="个人">
      <客户编号>KH-0786</客户编号>
      <客户名>任建新</客户名>
      <详细地址>
        浙江大学1198信箱
      </详细地址>
      <邮政编码>310027</邮政编码>
    </客户>
    <音乐编号>CD-0987</音乐编号>
    <数量 单位="张">560</数量>
    <付款方式>信用卡</付款方式>
  </预订>
  <预订 date="2001-09-23">
    <客户 种类="单位">
      <客户编号>KH-0787</客户编号>
      <客户名>西湖区旅游局</客户名>
      <详细地址>
        浙江省杭州市西湖区旅游局
      </详细地址>
      <邮政编码>310026</邮政编码>
    </客户>
    <音乐编号>CD-2365</音乐编号>
    <数量 单位="张">50</数量>
    <付款方式>信用卡</付款方式>
  </预订>
  <预订 date="2001-09-23">
    <客户 种类="个人">
      <客户编号>KH-0788</客户编号>
      <客户名>马辛</客户名>
      <详细地址>
        天津市和平区112号
      </详细地址>
      <邮政编码>3000051</邮政编码>
    </客户>
    <音乐编号>TYPE-0256</音乐编号>
    <数量 单位="盘">100</数量>
    <付款方式>现金</付款方式>
  </预订>
</今日预订>

```

上面的单个预订信息是在一个面向客户的网页上记录下来的，这个网页自动记录每天的用户订单并且生成所需要的 XML 文档。对于这个 XML 文档，我们需要不断的维护，否则可能因为时间的缘故会变得非常的庞大。但是上面的信息量很大，所以分别保存在两个

不同的表中，这两个表分别是订购表和订购人表。下面来看如何把这些信息分别写入这两张表中。先来看看这两张表是如何设计的。订购表包含了 6 个字段，在数据库中可以这样来定义这些字段。如表 8-4 所示。

表 8-4 订购表的结构定义

字段名	数据类型	大小	必填字段
订购号	自动编号	/	/
客户号	文本	50	是
音乐编号	文本	50	是
数量	数字	整型	是
日期	日期/时间	常规日期	是
备注	备注	255	否

实际上，很难用有效性规则来定义客户号、音乐编号等。相对于订购表来说，可以把客户的种类设成为查阅向导，在向导中只设置两个值，即“个人”和“客户”。这个规则同样用在音乐库存表中的音乐种类字段中。

表 8-5 改进后的订购表的结构

字段名	数据类型	大小	必填字段
客户号	文本	50	是
订购人	文本	50	是
种类	查阅向导（文本）	50	是
详细地址	文本	255	是
邮政编码	文本	50	是
备注	备注	255	否

ASP 程序将会是下面的形式。

程序清单 8.7:

```
<%
strSourceFile = Server.MapPath("/") & "\Per-bok.xml"
Set objXML = Server.CreateObject("Microsoft.FreeThreadedXMLDOM")
objXML.load(strSourceFile)
Set objRootsite = objXML.documentElement.selectSingleNode("今日预订")
%>
```

假如把上面提到的 XML 文件保存为 Per-bok.xml 文档，上面代码将首先获取 XML 文件的路径（根据虚拟目录不同而不同），然后以自由线程创建一个 XML 对象并且把 XML 文件读入内存，选取根节点今日预订。

程序清单 8.8:

```
<%
set rs1=GetMdbRecordset("music.mdb", "订购")
```

```

        if rs1 is nothing then
            response.write "GetMdbRecordset 调用失败"
            response.end
        end if
        set rs1=GetMdbRecordset("music.mdb", "订购人")
        if rs1 is nothing then
            response.write "GetMdbRecordset 调用失败"
            response.end
        end if
    %>

```

打开两个数据库表，这两个数据库表示用来存储生成的 XML 信息的，然后我们通过 DOM 方法来读取节点的信息，这段代码如下：

程序清单 8.9:

```

<%
AllNodesNum=objRootsite.childNodes.length-1
i=0
while AllNodesNum>0
Set attr1=objRootsite.childNodes.item(i).attributes.getNamedItem("date")
attrDate=attr1.nodeValue
kh=objRootsite.childNodes.item(i).childNodes.item(0)
Set attr2=kh.attributes.getNamedItem("种类")
attrType=attr2.nodeValue
khcode=kh.childNodes.item(0).text
khname=kh.childNodes.item(1).text
khaddress=kh.childNodes.item(2).text
khzip=kh.childNodes.item(3).text
yncode=objRootsite.childNodes.item(i).childNodes.item(1).text
ynnum=objRootsite.childNodes.item(i).childNodes.item(2).text
paytype=objRootsite.childNodes.item(i).childNodes.item(3).text
AllNodesNum=AllNodesNum-1
i=i+1
%>

```

上面的代码主要是利用 DOM 方法提取 XML 文档中的各个节点。显然 objRootsite 对象所对应的节点为<预订>，objRootsite.childNodes.item(i)所对应的节点就是<预订>节点，因为不止一个<预订>节点用了 item()来识别当前节点数据，i 用于记录预订的节点，随着 i 的递减，<预订>节点跟着往下移一个读取<预订>节点数据。

objRootsite.childNodes.item(i).childNodes.item(1).text 所对应的节点为具体的每个节点的文本值，这里(1)所对应的就是音乐编号（因为<预订>节点的子节点才是真正所需要的）。

接下来就是把信息写入数据库中了，可以这样来实现：

程序清单 8.10:

```

<%
    rs1.AddNew
    rs1("客户号")=khcode
    rs1("音乐编号")=yncode
    rs1("数量")=ynnum
    rs1("日期")=attrDate
    rs1.update
    rs2.AddNew
    rs2("客户号")=khcode
    rs2("订购人")=khname
    rs2("种类")=attrType
    rs2("详细地址")=khaddress
    rs2("邮政编码")=khzip
    rs2.update
wend
set objXML=nothing
rs1.close
rs2.close
%>

```

这样看来确实可以很好的处理数据库与 XML 文档之间的联系,也就是说完全可以利用 XML 来传送信息,而且现在很多的软件都支持数据库到 XML 的格式转变。

8.2.2 DOM 方法新建 XML

前面已经介绍了一种直接使用数据库和 ASP 生成 XML 文档的方法,事实上我们需要更成熟的技术,这种技术允许我们很容易的创建需要的 XML 档,包括文档节点和属性的添加,删除,修改等等,目前可以用 DOM 方法来实现它。而通常的 DOM 方法可以结合一些 SQL 语句或 XSL 语法限定查询。

需要做的第一件事是修改脚本,把从数据库取回的信息存储到一个 XML DOM。使用 DOM 也需要注意一些问题:这些问题包括确保所有的标记都是匹配的;所有的属性都用双引号扩起来;并且所有的命名空间都是正确的。在传输结果之前,甚至可以用解析器来验证生成节点树。

来看下面的一个程序。

程序清单 8.11:

```

<%Language=VBScript%>
<%
    Response.ContentType="text/xml"
    sql="select * from 订购人 where 种类='个人'"
    set rs=GetMdbRecordset("music.mdb", sql)
    if rs is nothing then

```

```
response.write "GetMdbRecordset 调用失败"
response.end
end if
dim oParser
set oParser=Server.CreateObject("Microsoft.XMLDOM")
```

这段代码首先打开数据库中的订购表,选取符合要求的列(这个条件的限制是通过 SQL 语句来实现的),然后创建一个 DOM 用来保存结果。

程序清单 8.12:

```
dim perboks
set perboks=oParser.createElement("客户列表")
do while not rs.eof
    dim perbok
    set perbok=oParser.createElement("客户")
    perbok.setAttribute "种类", rs("种类")
    FieldToElement perbok, rs, "客户号"
    FieldToElement perbok, rs, "订购人"
    FieldToElement perbok, rs, "详细地址"
    FieldToElement perbok, rs, "邮政编码"
    perboks.appendChild(perbok)
    rs.MoveNext
    Set perbok=nothing
loop
rs.close
set oParser.documentElement=perboks
%>
```

读者应该很好理解上面的代码。事实上,前面已经通过数据库建立了结果集,而在这里要做的是如何把这个结果集写入 DOM 树中。这样做通常需要建立一个根目录,建立根目录的语句可通过下面的语句来实现:

```
dim perboks
set perboks=oParser.createElement("客户列表")
```

看来在 DOM 中建立节点比起在开始和结尾处写<客户列表>和</客户列表>要容易,DOM 能够保证这个层次被正确维护。在创建了根目录之后,就可以开始循环处理数据了。每一次从数据库中得到一条记录,然后在 DOM 中创建一个新的元素,这个新元素命名为<客户>。

```
dim perbok
set perbok=oParser.createElement("客户")
```

在我们还没有建立节点互连之前，各个节点是完全自由浮动的，并不是按照创建的先后顺序来安排的。可以看到在后面我们把客户节点附接给客户列表节点。

```
perboks.appendChild(perbok)
```

在循环中，可以为客户节点添加了一个属性“种类”，虽然这可能有点多余，因为在打开数据库提取结果集的时候就只提取了种类为个人的用户，但是为了说明 DOM 属性的创建，这里还是说明一下：

```
perbok.setAttribute "种类", rs("种类")
```

在后面的循环中，通过一个 FiledToElement() 函数向客户节点加入元素内容，这样相对来说容易一些（可以省去很多代码的量）。在后面定义了这个函数。

创建了 DOM 后，最后要做的就是把它发送给浏览器。把必要的信息放在一个 XML 文档中，然后使用 DOM 的 XML 属性来编号它的内部结构的文本化信息。

对数据库中表“定购人”来说，事实上上面的程序没有处理一个字段，那就是“备注”。因为备注通常用来记录一些说明性的内容，而且设置为可以为 null，所以并没有在上面说明。假如读者想把它加入到程序中的话，可以这样来表示：

```
If not IsNull(rs("备注")) then
    FieldToElement perbok, rs, "备注"
End if
```

8.2.3 查询和优化

XSL 查询

通常使用 XSL 方法来实现 XML 的查询机制。上面一段代码只是简单的以 XML 的形式输出所有的作者，但是使用 DOM 要好于自己写标记。换句话说，不应该增加额外的功能到已有的东西上。然而，既然数据在 DOM 中，那么就可以用它来处理任何我们想处理 XML 的事情，特别是可以增加自己的 XSL 查询语句。

XSL 查询语句必须用于将数据库中的数据转换为 DOM 的代码之后，因为只有那时才可以使用 XSL 语句。

程序清单 8.13:

```
<?xml version="1.0" encoding="gb2312"?>
<%
    dim myxsl
    myxsl=Request.QueryString("xsl")
    if IsEmpty(myxsl) then
        Response.write oParser.xml
```

```
else
    set oNodeList=oParser.documentElement.selectNodes(myxsl)
    dim l
    for l=0 to oNodeList.length-1
        Response.Write oNodeList.item(i).xml
    next
end if
%>
```

这里使用 Request 命令来读取 XSL 指令，实际上完全可以在程序中简单的写入我们需要的 XSL 语句，比如象下面的几种形式：

```
客户[邮政编码="310027"]
/客户/订购人
/客户/客户号
```

当然相对于创建整体 DOM 树来说，这样做还是很简单的。因此相对的 XSL 查询语句也比较简单。在上面的程序段中，如果没有查询被传给调用者的话，将返回整个结果集。如果存在一个查询，就可以使用 selectNodes 语句来进行过滤。SelectNodes 语句可以用在任意的节点上，可以这样写：

```
set oNodeList=客户.selectNodes(myxsl)
```

输出结果的时候使用了循环，因为可能有一些顶级的节点需要在结果节点列表中循环。为每一个节点输出 XML 要好于简单的把整棵树输出为 XML。

事实上，用 XSL 语句创建的结果树可能不是一个有效的 XML 文档，比如下面的查询：

```
客户列表/客户/客户号
```

它返回的节点集可能就是如下的样子：

```
<客户号>KH-1023</客户号>
<客户号>KH-1035</客户号>
<客户号>KH-1038</客户号>
```

作为一个 XML 文档，它的结构就不是完整的，因为它没有根节点。但是可以像前面一样简单的用一个<客户编号>节点包装每件东西，这或许有点重复的味道，因为根节点与它的子节点代表着同一个意思。这样做得到的结果就是如下的样子：

```
<客户编号>
<客户号>KH-1023</客户号>
<客户号>KH-1035</客户号>
```

```
<客户号>KH-1038</客户号>
</客户编号>
```

从结构上看,虽然得到了一个有效的 XML 文档,但是这势必破坏了原来的 DTD 模式。因为我们新添加了一个叫客户编号的元素,而在原来的模式中并不存在这样的节点。

也可以新建节点来包容结果集,这样的节点常常称为容器,而专业人士对于容器的解决方式就是通过名域来实现的,也就是说可以利用另外的 DTD 模式中定义过的标签来包容结果集。这是一个完全可行的方法。比如说对于上面的例子,完全可以像下面这样来表示客户编号这个容器:

```
<WR:客户编号 xmlns:wr="http://10.12.13.66/xml/sjk21-3.dtd">
<客户号>KH-1023</客户号>
<客户号>KH-1035</客户号>
<客户号>KH-1038</客户号>
</WR:客户编号>
```

甚至可以加入任意的信息,包括像日期等。

优化

事实上,使用 XSL 的查询机制效率是相当低的。因为在提取我们需要的信息前,必须保证所有的记录都提取出来。对于上面的简单程序来说,这可能不是一个很难的问题。但是对于一个有着成千上万的记录的数据库来说,这显然会极大的影响效率,所以需要不同程度的优化。优化,就要使用到 SQL 语句,因为利用了 SQL 语句,在提取数据集的时候就对它们进行了过滤,也就是说不必再提取所有的记录,而是直接提取我们需要的记录。前面已经介绍 SQL 语句,所以这里只想介绍如何把 Xpath 语句转化为需要的 SQL 语句。

SQL 提供了功能强大的查询机制,所以大多数的 XSL 查询都能很好的用 SQL 语句表示出来。事实上,前面讲到的 DOM 例子就已经使用了 SQL 语句了:把种类为“个人”的记录全部提取出来。

```
sql="select * from 定购人 where 种类='个人'"
```

相对于 XSL 来说,它的语句可能就是如下的形式:

```
//客户[@种类="个人"]
```

这样生成的 XML 还是比较简单的。对于比较复杂的 XML 文档来说,XSL 的查询机制还是很丰富的。比如要得到所有预订数量在 50 以上(不包括 50)的预订客户的信息,用 XSL 表现的形式如下所示:

```
/预订[数量 >= 50]/客户
```

注意：在 XSL 语法中，使用各种符号都有它特殊的规定，不能使用“>”、“<”等保留符号。

下面用 SQL 语句表示出来，这里需要打开两个表：

```
select B.客户号, B.订购人, B.详细地址, B.邮政编码 from 订购 as A, 订购人 as B where A.
客户号=B.客户号 and A.数量 >50
```

如果要提取所有在 2000 年 9 月 23 号填写的订单，并且订购人的种类是单位的所有信息，可以这样来写：

```
/预订[@date="00-09-23" $and$ 客户[@种类="单位"]]
```

也可以很容易的用 sql 语句来表示出来：

```
select * from 订购 as A, 订购人 as B
where A.日期="00-29-23" and B.种类="单位" and A.客户号=B.客户号
```

到这里本节内容就快结束了，上面给大家介绍了 XML 与数据库之间信息的转换，DOM 方法与数据库以及 XSL 查询等技术。可能我们对查询的认识还不够，但查询的使用范围确实相当广，甚至可以在 ASP 生成 XML 文件后使用 XSL 查询，比如说下面的形式：

```
http://10.12.13.66/asp-xmlchang.asp?xsl=//预订[订购人="任建新"]
```

如果使用 SQL 查询，完全有可能构建更加复杂化的形式包括象包含和子查询等方式：

```
Select * from 订购人 where
    客户号 in ( select * from 订购
where 数量>50)
```

上面经常用到一个函数 GetMdbRecordset()，这里把这个函数的源代码写出如下：

```
Function GetMdbRecordset( FileName, Source )
    Set GetMdbRecordset = GetMdbRs( FileName, Source, 2, "" )
End Function
```

8.3 XML 模式与数据库

前面提到过如何处理 XML 模式与数据库之间的关系，应该说给出的实例还是比较简单的。事实上我们还会遇到关于复杂类型的很多节点、数据类型等问题。下面介绍一些可能对你比较适用的标准，当然还是建议读者具体情况具体分析，找出最好的办法来。

8.3.1 数据库到 XML 模式

很多软件都提供数据库到 XML 文档或 XML 模式之间的转换，总体来说这些转换还是

不够令人满意的，因为在很多的节点处理上，它们往往会使用一些自己定义的通用标签，而这些标签对于表明 XML 内容是没有任何意义的。比如说使用 XML SPY 来导入一个数据库 MUSIC 中的音乐表，那么就得到一个全部是节点来表示的 XML 文档。而它的 XML 模式可能就象下面表示的那样。

程序清单 8.14:

```
<?xml version="1.0" encoding="GB2312"?>
<Schema name="Untitled-schema" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="Import" model="closed" content="eltOnly" order="seq">
    <AttributeType name="xmlns" dt:type="string"/>
    <attribute type="xmlns"/>
    <element type="Row" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="Row" model="closed" content="eltOnly" order="seq">
    <element type="音乐编号" minOccurs="1" maxOccurs="1" />
    <element type="种类" minOccurs="1" maxOccurs="1" />
    <element type="专辑名" minOccurs="1" maxOccurs="1" />
    <element type="演唱者" minOccurs="1" maxOccurs="1" />
    <element type="价格" minOccurs="1" maxOccurs="1" />
    <element type="说明" minOccurs="1" maxOccurs="1" />
  </ElementType>
  <ElementType name="价格" model="closed" content="textOnly" dt:type="fixed.14.4" />
  <ElementType name="说明" model="closed" content="textOnly" dt:type="string" />
  <ElementType name="演唱者" model="closed" content="textOnly" dt:type="string" />
  <ElementType name="音乐编号" model="closed" content="textOnly" dt:type="string" />
  <ElementType name="种类" model="closed" content="textOnly" dt:type="string" />
  <ElementType name="专辑名" model="closed" content="textOnly" dt:type="string" />
</Schema>
```

很明显，它使用 Import 元素和 Row 元素代替了前面的库存和音乐的标签。而数据库的导出不能设置属性和标签。一般的数据库导出，要么使用节点输出，要么是用属性输出，而不能是属性和节点同时输出。

为了与使用结构输出的 XML 相区分，我们把从数据库中输出的 XML 叫做 XML 记录集。它对于我们非常有用，主要是因为它用的是 XML。一旦用 XML 记录数据，就可以使用以前介绍过关于 XML 文档的所有工具：可以使用 XSLT 来将信息转换成一系列的使用 HTML 的表格，或者可以改变列的名称，这样数据就可以插入到别的数据库中了。

下面程序用于转换上面的信息：

代码清单 8.15:

```
<?xml version = "1.0" encoding="GB2312"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
```

```
<HTML>
  <H1>音乐库存信息</H1>
<BODY COLOR="#0000A0">
  <TABLE BORDER="1">
    <TR>
      <TH>音乐编号</TH>
      <TH>专辑名</TH>
      <TH>演唱者</TH>
      <TH>价格</TH>
      <TH>说明</TH>
    </TR>
    <xsl:for-each select="import/Row">
      <TR>
        <TD><xsl:value-of select="音乐编号"/></TD>
        <TD><xsl:value-of select="专辑名"/></TD>
        <TD><xsl:value-of select="演唱者"/></TD>
        <TD><xsl:value-of select="价格"/></TD>
        <TD><xsl:value-of select="说明"/></TD>
      </TR>
    </xsl:for-each>
  </TABLE>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

当然还可以使用脚本来转换 XML 数据，这就是前面提到的通过 ASP 或 DOM 实现的数据库到 XML 文档的输出。究竟使用那种方法较好，没有人知道。但目前许多人还是倾向于使用 ASP 和 DOM，毕竟对于标签来说我们还是需要使用 XML 模式中已经定义过的。而一般的数据库都是通过 XML 模式来建立的，但数据库在自动导出的过程中不可避免要使用我们在模式中没有定义过的通用的标签格式，这就破坏了 XML 模式。所以如果希望建立高效的数据转换的话，就必须使用 ASP 和 DOM。

8.3.2 规则

这里给大家介绍一些建立关系数据库的关键知识点，也就是如何使得创建的关系数据库中的数据与模式相一致，下面就给出一些规则。

在 XML 模式中经常遇到的情况可能是下面的几种：

- ☒ 元素的分层
- ☒ 属性
- ☒ 纯文本元素
- ☒ 枚举元素或属性

- ☒ ID 数据类型的属性
 - ☒ MinOccurs 和 MaxOccurs 属性
 - ☒ 拥有指定数据类型的元素或属性
 - ☒ 拥有 idref, idrefs 数据类型的属性
- 分别来对它们进行讨论。

元素的分层

数据库中的表通常用来保存有文本内容或其他内容的标记，这样就可以形成一个列，但是对一些父节点的控制可能不尽人意。就像上面建立的数据库表，都记录了节点的内容，注意这些父节点都是“空”的，这里的空代表该节点下不存在文本内容或数字内容等具体的东西，而只有其他节点（eltOnly）。

但是在实际中可能遇到 mixed 类型的父节点，也就是说它里面可以包含子节点，也可以包含文本内容。这种情况下，就不得不在数据库中建立这样的一列。在用户知道某个节点是父节点的情况下，可以不用特殊的标记来表示它，但是通常情况下都需要标记它，因为它是一个父节点，比其他的同类节点可能大一级或两级。标记父节点的方法大家可以自己来定义规则。比如说使用 F-1-、F-2-等来标记。

当然对于有些严格的系统来说，可能不得不记录下所有的标记。这些标记有严格的父子关系，所以这样就比较麻烦，但我们可以试着把父节点和子节点的关系通过数据库表表示出来，比如说给父节点新建一个与该父节点同名的表，然后在表里建立子节点字段，并且写入它的子节点名称（对于大的 XML 模式来说，可能需要建立很多张表）。比如说对于下面的这段 XML 模式代码：

程序清单 8.16:

```
<ElementType name="员工" model="closed" content="eltOnly" order="seq">
  <element type="编号" minOccurs="1" maxOccurs="1"/>
  <element type="姓名" minOccurs="1" maxOccurs="1"/>
  <element type="职务" minOccurs="1" maxOccurs="1"/>
</ElementType>
```

就可以建立这样的一个图 8-2:

ID	节点名	类型	最小出现	最大出现
1	编号	内容	1	1
2	姓名	内容	1	1
3	职务	内容	1	1

图 8-2 建立员工表

注意这个表的名称是员工，而对于它的设计样式可以如下面表 8-6 所示：

表 8-6 XML 表示的设计样式

字段名	数据类型	大小	必填字段
节点名	文本	50	是
类型	查阅向导（文本）	50	是
最小出现	文本	50	否
最大出现	文本	50	否

经过上面的设计后，就不用担心如何处理 minOccurs 和 maxOccurs 这两个属性了，因为在父节点的表中已经定义了最大出现和最小出现了。

这可能不是最好的方法，读者完全可以根据自己的需要改变数据库的设计。

属性

常常把属性放在表中的一个列中表示，而且通常用特殊的名字来表示。比如说可以在名字前面加上“attr”。而且通常属性都是不允许为空的，也就不能把属性的值定义为空。这样就可以得到属性的规则：

对于每个属性节点，创建一个与属性节点同名的列；如果要求属性，那么列不允许为空。

下面举个例子来说明。比如说对于下面的这一段程序：

程序清单 8.17：

```
<ElementType name="商品" model="closed" content="eltOnly" order="seq">
  <AttributeType name="编号" dt:type="enumeration" required="yes"/>
  <AttributeType name="连接" dt:type="enumeration" required="yes"/>
  <attribute type="编号"/>
  <attribute type="连接"/>
  <element type="等级" minOccurs="1" maxOccurs="1"/>
  <element type="定价" minOccurs="1" maxOccurs="1"/>
</ElementType>
```

显然按照分层规则可以建立一张商品的表格，用来说明包含子节点和属性，然后再建立一张后续的表用来保存这些子节点和属性的内容。如表 8-7 所示。

表 8-7 保存节点和属性的表

字段名	数据类型	大小	必填字段
节点名	文本	50	是
类型	查阅向导（文本）	50	是
最小出现	文本	50	否
最大出现	文本	50	否

这样看来关于查阅向导也需要扩充了。因为在原来的查阅向导中我们只定义了 3 个值，它们是 eltOnly，Mixed 和叶节点，这里就需要加入属性，以便系统能够很好的识别。

对于纯文本的节点，这里不再细说，因为纯文本的节点一定属于叶节点，只需要用一系列来表示就可以了。前面讲到的几个数据中，我们就为每一个纯文本的节点建立了一个字段。

枚举

关于枚举类型的元素或属性，通常要求单独建立一个表来保存枚举的各个值，而且使用不同于一般属性的名称来表示该属性或节点。比如对于属性，用 `attr` 作为前缀，而枚举的属性，就需要用 `attr-enum` 作为前缀。

这种命名方式和定义方法很有用。事实上，这种用表值来记录枚举类型的值的方法是很严格的。当枚举类型较多时，这种方法也确实很好。比如说对货币单位，可选的类型可能包含了数十种，这样的枚举在某方面来说不得不考虑用表来记录，因为货币是有严格的名字的，因为在一些公用场合错用货币单位长度是不可原谅的。

因此可以说，该规则就是对一个枚举类型的属性或节点创建一个与属性或节点有相同名字的表，使用前缀 `attr-enum` 或 `node-enum`。

但是对有些枚举类型来说，完全没有必要这么麻烦的去建立一个表，比如说对于 `yes` 和 `no` 类型的枚举。我们知道在 `access` 数据库中有这样的类型，完全可以用数据类型“是/否”来表示。而且假如枚举的属性不是很严格的时候，完全可以使用 `access` 数据库中的查阅向导来建立枚举的各个值。这样，就不必为每一个枚举的类型建立一个数据库表了。来看下面的例子。

程序清单 8.18:

```
<?xml version="1.0" encoding="GB2312"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="处理人" model="closed" content="textOnly" dt:type="enumeration"/>
  <ElementType name="订单" model="closed" content="eltOnly" order="seq">
    <AttributeType name="发货" dt:type="boolean" required="yes"/>
    <attribute type="发货"/>
    <AttributeType name="xmlns" dt:type="string"/>
    <attribute type="xmlns"/>
    <element type="专辑名" minOccurs="1" maxOccurs="1"/>
    <element type="数量" minOccurs="1" maxOccurs="1"/>
    <element type="处理人" minOccurs="1" maxOccurs="1"/>
  </ElementType>
  <ElementType name="数量" model="closed" content="textOnly" dt:type="i1"/>
  <ElementType name="专辑名" model="closed" content="textOnly" dt:type="string"/>
</Schema>
```

上面的模式把处理人表示为枚举类型，并且把发货属性表示成 `Boolean` 型，即可选是或否。那么在建立数据库的时候可以这样来定义数据库中的一些规则。如表 8-8 所示。

表 8-8 订购表的结构

字段名	数据类型	大小	必填字段
ID	自动编号	/	是
发货	是/否	/	是
专辑名	文本	50	是
数量	文本	50	是
处理人	查阅向导（文本）	50	否

注意：这里，我们使用是/否来表示“发货属性”，对于处理人则使用查阅向导来表示枚举类型。因为就一个单位来说，处理人总是固定的几个，不需要特别的严格。

ID、idref 和 idrefs

ID 属性通常用来指代其他元素的某个特别的标志，也就是说可能要引用到其他的一个元素，假如把该属性保存为一列的话，最简单的方法就是通过指定特殊名称来表示。比如说使用前缀 attr-id-来表示 id 类型的数据。

和 id 相比，idrefs 的存储可能就比较复杂了，不得不借助另外的一张表。因为在 idrefs 中记录的数值可能包含很多个，而且这些数字都代表着不同的各个节点或其他数据类型（不能再使用查阅向导来表示）。所以对于 idrefs，不得不要建立一个表，这个表的名称通常取为和 idrefs 名称一样，里面有一列，可以取用 idrefs-的名称，在里面记录下 idrefs 的各个数值。

相对于 id 来说，idref 的存储方式和 id 类型差不多。往往用一个列来表示 idref 的数据类型，但是要使用前缀 attr-idref。以便很好的区别其他属性。

下面举个例子来说明 idrefs 这个模式的用法，比如说下面的形式：

程序清单 8.19:

```
<Book isbn="1-86909-56-3"
level="Experienced"
pubdate="00-3-25"
pagecount=""
authors="1 3 5"
threads="2 4 6">
```

很显然，上面的代码使用了两个 idrefs 类型的属性，这两个属性是“authors”和“threads”。按照前面的规则就需要创建两张表，一张为 authors，另外一张为 threads，而且需要在这两张表中分别建立一个字段，命名为 attr-idrefs-authors 和 attr-idrefs-threads，并在里面存储必要的信息。

数据类型

在本节的最后，让我们来看看 XML 模式和我们使用的 Access 数据库中的数据类型。关于 XML 模式中的数据类型，大家可以参考 XML 模式这一章。在 Access 数据库中，可

以使用的几类大的数据类型包括象文本、数字、日期/时间、货币等。下表 8-9 把这两个数据类型做一下比较。

表 8-9 XML 模式数据类型和 Access 数据类型的比较

XML 模式中的数据类型	Access 数据库中的类型
String	文本
Int, float, double, number...	数字
Date, Datetime, Time	日期/时间
/	货币

本节讲述的主要是 XML 模式到 Access 数据库的一些规则，这些规则只是作为参考，实际的情况可能是不同的。所以必须按照实际的情况来确定如何来存储 XML 信息。

8.3.3 ADO 简介

自若干年前推出开放式数据库连接(ODBC)应用程序编程接口 (API) 以来，出现了各种各样的数据库访问技术，而 ADO.NET 是其中最新的一种。在这过程中，发生了许多有趣的事。例如，COM 闯入数据库领域，开始培植 OLE DB 的殖民进程。然后，大致相当于 OLE DB 自动化版本的 ActiveX® Data Objects (ADO) 被选来统治 Windows® 数据库开发者的 Visual Basic® 和 ASP 社区。

通过 .NET，Microsoft 正在提供通用框架（即 Framework Class Library），其中将包括所有现有的 Windows API 甚至更多的内容。特别值得一提的是，它包括大量常用的库，而这些库现在需要通过各个 COM 对象分别获得。在这些库中，你会发现 XML 和 ADO 对象模型，它们被集成到了叫做 ADO.NET 的类子树中。

ADO.NET 事实上成为构建 .NET 应用程序的基础。和 ADO 不同的是，ADO.NET 遵循更通用的原则，不专门面向数据库。ADO.NET 集合了所有允许数据处理的类。这些类表示具有典型数据库功能（如索引、排序和视图）的数据容器对象。尽管 ADO.NET 是 .NET 数据库应用程序的权威解决方案，但从总体设计上来看，它不象 ADO 模型那样以数据库为中心，这是 ADO.NET 的一大特点。

同时 ADO.NET 又保持着与以前的 ADO 模型有关的一些主要概念，它已经被极大的完善，并从不同的信息来源提供途径去获得结构化的数据——一个平台文本文件，从数据库管理系统获得的相关数据，或者是分级的 XML 数据——然而，所有都按照一个相容的、标准化的设计模型来执行。

快速浏览 ADO.NET

SQL Server 7.0（及更新版本）以及可以通过 OLE DB 提供者进行访问的任何数据源，这些又称为被管理的提供者（Managed Provider）。ADO.NET 框架的数据存取 API 提供了两种方式分别识别并处理两种类型的数据源：SQL Server 7.0（及更新版本）和可以通过 OLE DB 提供者进行访问的任何数据源。SQL(System.Data.SQL)库可以直接连接到 SQL Server 的数据，而 ADO (System.Data.ADO)库可用于其他通过 OLE DB 提供者进行访问的任何数据源。

SQL Server 被管理的提供者在 MS SQL Server 7.0 或以后的版本中使用叫做“tabulardata stream”的专用协议，而没有使用 OLE DB, ADO 或 ODBC。

ADO.NET 被管理的提供者能够在这些 OLE DB 提供者下工作，详细情况请参看表 8-10 数据库引擎和对应的数据库提供者。

表 8-10 访问数据库

驱动程序 Driver	提供者 Provider
SQLOLEDB	SQL OLE DB Provider
MSDAORA	Oracle OLE DB Provider
JOLT	Jet OLE DB Provider
MSDASQL/SQLServer ODBC	SQL Server ODBC Driver via OLE DB for ODBC Provider
MSDASQL/Jet ODBC	Jet ODBC Driver via OLE DB Provider for ODBC Provider

现在 ADO.NET 还不支持 MSDASQL/Oracle ODBC Driver (Oracle DB Driver For ODBC)。

下面将介绍每个被管理的提供者都可用的 ADO.NET 的核心组件

- ☒ Connections——连接和管理数据库事务。
- ☒ Commands——向数据库发送的命令。
- ☒ DataReaders——直接读取流数据。
- ☒ DataSets 和 DataSetCommands——对驻留内存中的数据进行存储和操作。

核心的 ADO.NET 功能基本上可以被概括为如下内容：

Connection 对象在 Web 页面和数据库间建立连接。Commands 对象向数据库提供者发出命令，返回的结果以一种流的方式贯穿于这些连接中。结果集可以用 DataReaders 快速的读取，也可以储存到驻留内存的 DataSets 对象中，然后通过 DataSetCommands 对象让用户在数据集中访问和操作记录。开发者可以用 DataSet 内置的方法在基础的数据源上去处理数据集。

为了使用 .NET 框架中的被管理提供者，需要把下面的名空间 (namespaces) 包括到 VB 页面中，如下所示。

‘SQL 被管理的提供者：

Import System.Data.SQL

‘ADO 被管理的提供者：

Import System.Data.ADO

XML 扩展支持

在 ADO 中，XML 只不过是输入和输出格式。然而在 ADO.NET 中，XML 是一种数据格式，提供了操作、组织、共享和传递数据的手段。任何带入 DataSet 的数据，无论其来源，都能通过双面编程模型进行处理。你可以顺序交替访问信息，或者按行访问，也可以按照 XML 文档对象模型驱动的非顺序、层次结构路径进行访问。

DataSet 将数据和架构作为 XML 文档进行读写。数据和架构都可以通过 HTTP 传

输,并且能在所有支持 XML 的平台上使用。相同的数据在不同的时候可以通过不同的架构来呈现,这是通过 XSLT 实现的。你可以使用 ReadXmlSchema 方法编写架构。XML 架构包括数据集中的表的说明,以及表的关系和约束。在调用 ReadXmlData 方法填充 DataSet 之前,应该先完成这个步骤。

8.4 应用实例

接下来给读者介绍一个网上音乐预订的网页。这里要说明其中的一部分内容,这部分内容包括音乐网站显示、用户信息输入以及用户信息的保存等 3 个部分。下面就来详细介绍这 3 个部分。

8.4.1 音乐显示

对于音乐 XML 文档的生成,我们使用上面介绍过的内容,也就是使用 ASP 方法联系数据和 XML 文档,这段代码类似于前面出现的代码。

程序清单 8.20:

```
<%Language=VBScript%>
<%
    response.ContentType="text/xml"
    dim rs
    set rs=GetMdbRecordset("music.mdb", "音乐")
    if rs is nothing then
        response.write "GetMdbRecordset 调用失败"
        response.end
    end if
%>
<?xml version="1.0" encoding="gb2312"?>
<音乐库存>
<%
    Do While Not rs.EOF
%>
<音乐>
    <音乐编号><%=rs("音乐编号")%></音乐编号>
    <专辑名><%=rs("专辑名")%></专辑名>
    <演唱者><%=rs("演唱者")%></演唱者>
    <价格 单位="元"><%=rs("价格")%></价格>
    <% if rs("说明") <> Empty then %>
        <说明><%=rs("说明")%></说明>
    <% end if %>
    <% if rs("备注") <> Empty then %>
        <备注><%=rs("备注")%></备注>
    <% end if %>
```

```

</音乐>
<%
    rs.Movenetxt
Wend
%>
</音乐库存>
<%
    rs.close
%>

```

上面的这段代码，这里不再说明。当然也可以使用 DOM 的方法来建立 XML 文档。下面来看看如何显示这个 XML 文档，有可选的两个形式：一个是直接在生成的 ASP 文件中加入 XSL 文件转换；还有就是使用数据岛的方法来实现。不管使用那种方法，得到的结果都是一样的，这里先把 XSL 文件的内容表示出来。

程序清单 8.21:

```

<?xml version="1.0" encoding="gb2312"?>
<!-- 本例基于xmldocument和xsl:document -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl" language="VBScript">
<xsl:template match="/">
<HTML>
    <HEAD>
    <STYLE>
        BODY {margin:0}
        .bg {font:8pt Verdana; background-color:0000a0; color:white}
        H1 {font:bold 14pt Verdana; width:100%; margin-top:1em}
        .row {font:8pt Verdana; border-bottom:1px solid #CC88CC}
        .header {font:bold 9pt Verdana; cursor:hand; padding:2px; border:2px outset gray}
    </STYLE>
    </HEAD>
</HTML>

```

上面的这段代码是我们使用的 XSL 文件的头。其实它也没有实质内容，仅仅定义了 XSL 文件中要用到的 CSS 的格式，而且指明在 XSL 文档中用到的脚本语言将会是 VB 脚本语言。

程序清单 8.22:

```

<SCRIPT LANGUAGE="VBScript"><xsl:comment>
<![CDATA[
    Dim stylesheet, source, sortField
    Function sort(field)
        sortField.value = field
        listing.innerHTML = source.documentElement.transformNode(stylesheet)
    
```

```

End Function
]]></xsl:comment>
</SCRIPT>
<SCRIPT LANGUAGE="VBScript" for="window" event="onload"><xsl:comment>
<![CDATA[
    Set stylesheet = document.XSLDocument
    Set source = document.XMLDocument
    Set sortField = document.XSLDocument.selectSingleNode("//@order-by")
]]>
</xsl:comment>
</SCRIPT>

```

上面的程序代码定义了两个函数，主要的作用是用来排序的。在我们的 XSL 文件中，需要使用到上面的两个程序来对显示的信息排序。

程序清单 8.23:

```

<HTML>
<BODY>
    <TABLE width="100%" cellpadding="0">
    <TR>
        <TD class="bg"/>
        <TD class="bg">
            <H1>音乐新干线</H1>
            <DIV>本期推荐: <B>难得一见</B></DIV>
            <DIV>影人介绍: <B>孙燕姿</B></DIV>
        </TD>
    </TR>
    <TR>
        <TD class="bg" width="120" valign="top">
            <P>我们将会努力的做好每一件工作~! </P>
            <P>希望各位音乐朋友多提意见! </P>
        </TD>
        <TD class="bg" valign="top">
            <DIV id="listing"><xsl:apply-templates select="库存"/></DIV>
        </TD>
    </TR>
    </TABLE>
    <table width="100%" cellpadding="0">
    <tr>
        <td ALING="CENTER" class="BG"><CENTER>
        <A HREF="MUSIC-BOOK.ASP" class="BG">订购</A></CENTER></td>
        <td ALING="CENTER" class="BG"><CENTER>
        <A HREF="#" ONCLICK="self.close()">关闭</A></CENTER></td>
    </tr>
    </table>

```

```

</tr>
</table>
</div>
<table width="750" border="0" cellspacing="2" cellpadding="1" align="center">
  <tr>
    <td align="center">**网版权所有&copy;2001 RESEARCH VER1.0</td>
  </tr>
</table>
<p>&nbsp;</p>
</BODY>
</HTML>
</xsl:template>
<xsl:template match="库存">
  <TABLE STYLE="background-color:white">
    <THEAD>
      <TD width="80"><DIV class="header" onClick="sort('音乐编号')">音乐编号</DIV></TD>
      <TD width="80"><DIV class="header" onClick="sort('专辑名')">专辑名</DIV></TD>
      <TD width="80"><DIV class="header" onClick="sort('演唱者')">演唱者</DIV></TD>
      <TD width="80"><DIV class="header" onClick="sort('价格')">价格</DIV></TD>
      <TD width="80"><DIV class="header" onClick="sort('种类')">种类</DIV></TD>
      <TD width="80"><DIV class="header" onClick="sort('说明')">说明</DIV></TD>
    </THEAD>
    <xsl:for-each select="音乐" order-by="/音乐编号">
      <TR>
        <TD><DIV class="row"><xsl:value-of select="/音乐编号"/></DIV></TD>
        <TD><DIV class="row"><xsl:value-of select="专辑名"/></DIV></TD>
        <TD><DIV class="row"><xsl:value-of select="演唱者"/></DIV></TD>
        <TD><DIV class="row"><xsl:value-of select="价格"/></DIV></TD>
        <TD><DIV class="row"><xsl:value-of select="@种类"/></DIV></TD>
        <TD><DIV class="row"><xsl:value-of select="说明"/></DIV></TD>
      </TR>
    </xsl:for-each>
  </TABLE>
</xsl:template>
</xsl:stylesheet>

```

剩下的代码都写在上面了。可以看出，这里使用 HTML 中的表格形式来表现 XML 文档的，而且表的每一个列都使用了 SORT 函数。也就是说可以通过单击列头来对这些数据进行排序，当然这些排序是按照字符串的音序来排的。在音乐的数量很多的时候，这样做对客户是非常有利的。

如果使用直接的转换，则可以直接在上面的 ASP 文件中写入语句。

```
<?xml version="1.0" encoding="gb2312"?>
```

```
<?xml:stylesheet type="text/xsl" href="music-sort.xsl"?>
```

注意：引用 XSL 转换的语句必须写在 XML 声明之后。

当然也可以使用数据岛的方法来表现该 XML 文档。这需要另外再建立一个新的文件，该文件的代码如下所示。

程序清单 8.24:

```
<HTML>
<HEAD>
  <TITLE>XSL</TITLE>
</HEAD>
<XML id="source" src="music-2.xml"></XML>
<XML id="stylesheet" src="music-sort.xsl"></XML>
<SCRIPT FOR="window" EVENT="onload">
  result = source.transformNode(stylesheet.XMLDocument);
  xmlSrc.innerHTML = result;
</SCRIPT>
<BODY>
<SPAN id=xmlSrc></SPAN>
</BODY>
</HTML>
```

到目前为止，这个音乐信息的网站可以显示出来了，它的显示如图 8-3 所示。



音乐编号	专辑名	演唱者	价格	种类	说明
CD-1023	盛夏的果实	莫文蔚	¥49.99	CD	莫文蔚的新专辑
CD-1024	熟悉到底	蔡康	¥45.99	CD	羽泉真真演绎
CD-1025	年华	刘若英	¥58.99	CD	刘若英的情歌永远是那么的入口
CD-1026	难得一见	孙燕姿	¥29.99	CD	迷人的音色
MP3-0212	男人哭吧哭吧不是罪	刘德华	¥12.00	MP3	低沉的音调、迷人的魅力
MP3-0213	那么新嫩	金海心	¥12.00	MP3	充满青春活力
MP3-0214	白鸽	左和	¥12.00	MP3	震撼人心
MP3-0215	NOW 5	蔡见	¥14.00	MP3	经典西方上榜歌曲精选
TY-0034	至少还有你	林忆莲	¥9.00	TYPE	还是这么动听
MP3-0217	阴天	莫文蔚	¥12.00	MP3	
MP3-1027	胡思乱想	无印良品	¥36.00	CD	清新的年轻感觉
MP3-0218	原野	周惠精选	¥12.00	MP3	不想让你知道~这个约定
TY-0036	最爱	蔡康	¥12.00	TYPE	最爱最爱~
CD-1028	海原	蔡品源	¥68.00	CD	海浪的冲击力
MP3-1029	套小狼	刘德华	¥26.00	CD	轻松的心情，轻松的感觉！
MP3-0219	就要爱了吗	苏慧伦	¥14.00	MP3	爱~
MP3-0220	情人	杜德伟	¥12.00	MP3	情人的感觉
MP3-1030	下砂	游鸿明	¥56.99	CD	单身贵族式的情调

图 8-3 音乐网站的显示效果

8.4.2 客户信息

下面使用表单形式来处理客户信息，这需要一个 ASP 页面，并使用 post 或 get 方法来发送表单信息。这个网页的名称叫做 music-book.asp，也就是上面的网页中“订购”连接的网页。下面来看它的源程序。

程序清单 8.25:

```

<html>
<head>
<title>音乐预订</title>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<link rel="stylesheet" href="style/style.css" type="text/css">
</head>
<body bgcolor="#FFFFFF" text="#000000">
<table width="750" border="0" cellspacing="2" cellpadding="1" align="center">
<tr>
<td align="center">音乐预订</td>
</tr>
</table>
<table width="500" border="0" cellspacing="2" cellpadding="1" align="center">
<tr>
<td align="right">
<div align="left">[<a href="music.asp">返回前页</a>] [<a
href="mailto:coldstar.sina.com">给我发信</a>]</div>
</td>
</tr>
</table>
<br>

```

上面的程序段是我们制作的网页的头，它说明了网页的 CSS 文档的形式，设置了标题，同时还给网页做了必要的链接。上面的“返回前页”就是为了到达音乐显示的页面。另外还设置了一个信箱，方便客户在必要的时候给我们发消息。

程序代码 8.26:

```

<form name="save" method="post" action="savebook.asp">
<table width="500" border="0" cellspacing="1" cellpadding="3" align="center">
<tr>
<td width="1" bgcolor="#ffffff" rowspan="9"></td>
<td width="100" bgcolor="#3399FF">
<div align="center"><font color="#FFFFFF">音乐编号(ID)</font><font
color="#FFFFFF">:</font></div>
</td>
<td width="400" bgcolor="eeeeee">
<input type="text" name="txtid" maxlength="30" size="14" class="f1">
</td>
<td width="1" bgcolor="#ffffff" rowspan="9"></td>
</tr>
<tr>
<td colspan="2"></td>
</tr>
<tr>
<td colspan="2"></td>
</tr>
<tr>
<td colspan="2"></td>
</tr>
<tr>
<td colspan="2"></td>
</tr>
<tr>
<td colspan="2"></td>
</tr>
<tr>
<td colspan="2"></td>
</tr>
<tr>
<td colspan="2"></td>
</tr>
<tr>
<td colspan="2"></td>
</tr>

```

```

        <div align="center"><font color="#FFFFFF">数量:*</font></div>
    </td>
    <td width="400" height="2" bgcolor="eeeeee">
        <input type="text" name="txtnum" size="20" maxlength="10" class="f1">
    </td>
</tr>
<tr>
    <td nowrap width="100" bgcolor="#3399FF">
        <div align="center"><font color="#FFFFFF">客户种类:*</font></div>
    </td>
    <td width="400" bgcolor="eeeeee">
        <select size="1" name="txtkhtype">
            <option>个人</option>
            <option>单位</option>
        </select></td>
</tr>
<tr>
    <td nowrap width="100" bgcolor="#3399FF">
        <div align="center"><font color="#FFFFFF">客户名:*</font></div>
    </td>
    <td width="400" height="2" bgcolor="eeeeee">
        <input type="password" name="txtname" size="20" maxlength="40" class="f1">
    </td>
</tr>
<tr>
    <td nowrap width="100" bgcolor="#3399FF">
        <div align="center"><font color="#FFFFFF">详细地址:*</font></div>
    </td>
    <td width="400" bgcolor="eeeeee">
        <input type="text" name="txtaddress" size="40" maxlength="50" class="f1">
    </td>
</tr>
<tr>
    <td nowrap width="100" bgcolor="#3399FF">
        <div align="center"><font color="#FFFFFF">邮政编码:*</font></div>
    </td>
    <td width="800" bgcolor="eeeeee">
        <input type="text" name="txtezip" size="10" maxlength="30" class="f1" style="width: 130;
height: 19">
    </td>
</tr>
<tr>
    <td nowrap width="100" bgcolor="#3399FF">

```

```

        <div align="center"><font color="#FFFFFF">Email信箱:</font></div>
    </td>
    <td width="800" bgcolor="eeeeee">
        <input type="text" name="txtebox" size="20" maxlength="30" class="f1" style="width:
130; height: 19">
    </td>
</tr>
<tr>
    <td nowrap width="100" bgcolor="#3399FF">
        <div align="center"><font color="#FFFFFF">付款方式:</font></div>
    </td>
    <td width="400" bgcolor="eeeeee">
        <select size="1" name="txttype">
            <option>信用卡</option>
            <option>现金</option>
            <option>其他</option>
        </select></td>
</tr>
<tr>
    <td nowrap width="100" bgcolor="#3399FF" rowspan="2">
        <div align="center"><font color="#FFFFFF">需要说明的内容:</font></div>
    </td>
    <td width="400" bgcolor="eeeeee">
        <div align="center">
            <p align="left">
                <textarea name="txtcontent" cols="40" rows="6" ></textarea>
            </p>
        </div>
    </td>
</tr>
<tr>
    <td width="400" bgcolor="eeeeee">
        <input type="submit" name="Submit" value="订购" class="f2">
        <input type="reset" name="Submit2" value="重置" class="f2">
    </td>
</tr>
</table>
</form>

```

上面的代码全都是表单的内容，里面一共设置了 9 个用户需要填写的内容，其中包括必填项目音乐编号、客户名、客户种类、详细地址、邮政编码和付款方式，以及不是必须填写的项目如客户的 E-mail 信箱和客户需要说明的内容，这些对象都被赋予了不同的名称象 txtid 等。这些信息都会发送到一个叫“savebook.asp”的网页上去。还需要两个按钮：

一个用于发送信息；一个用来重置各个 input 文本框的内容。

上面的 asp 文件，可以使用 Dreamweaver4 来编写，主要的工作是表格中各个项目的对齐问题，还有就是 input 文本框的大小设置。另外还需要一个 CSS 文档，用来让我们的网页更美观。上面的程序段还没有结束，需要加上一个结束部分。

程序清单 8.27:

```
<div align="center"><font color="#990000">*</font>为必填项，支持中文显示<br>
</div>
<table width="750" border="0" cellspacing="2" cellpadding="1" align="center">
<tr>
<td align="center">**网版权所有&copy;2001 RESEARCH VER1.0</td>
</tr>
</table>
<p>&nbsp;</p>
</body>
</html>
```

8.4.3 保存用户信息

保存用户信息，需要做的只有两件事：一件是有效性检查；另外一件就是把我们需要信息写入 XML 文档中去。事实上，有效性检查是必须的，假如没有有效性检查的话，我们可能会得到很多无用的表单，而处理这些表单就会花费很多无用的劳动力，导致工作的效率降低了。来看下面的程序。

程序清单 8.28:

```
<%
txtid=left(request("txtid"), 30)
txtname=left(request("txtname"), 30)
txtaddress=request("txtaddress")
txtzip=left(request("txtzip"), 10)
txtbox=request("txtbox")
txttype=request("txttype")
txtcontent=left(request("txtcontent"), 500)
txtkhtype=request("txtkhtype")
txtnum=request("txtnum")
```

上面程序段的工作是获取 (Request) 对象，也就是说读取 music-book.asp 表单发送来的消息。在文档里面用到了 left() 函数，该函数是读取文本框中靠左边的指定长度内不是空格的字节内容。可以看出共读取了 9 个对象并且把他们保存为和该对象同名的字符串变量。

程序清单 8.29:

```
if txtid="" and IsValidId(txtid) then
response.write "<script language='javascript'>" & VbCrLf
response.write "alert('请填入你预订的音乐编号!');" & VbCrLf
```

```
response.write "history.go(-1);" & vbCrLf
response.write "</script>" & VbCRLF
response.end
end if

if txtname="" then
response.write "<script language='javascript'" & VbCrLf
response.write "alert('请填入订购人姓名或单位!');" & VbCrLf
response.write "history.go(-1);" & vbCrLf
response.write "</script>" & VbCRLF
response.end
end if

if txtaddress="" then
response.write "<script language='javascript'" & VbCrLf
response.write "alert('请填入详细地址!');" & VbCrLf
response.write "history.go(-1);" & vbCrLf
response.write "</script>" & VbCRLF
response.end
end if

if txtzip="" and IsValidZip(zip) then
response.write "<script language='javascript'" & VbCrLf
response.write "alert('请输入你的邮政编码!');" & VbCrLf
response.write "history.go(-1);" & vbCrLf
response.write "</script>" & VbCRLF
response.end
end if

if txttype="" then
response.write "<script language='javascript'" & VbCrLf
response.write "alert('请输入你的付款方式!');" & VbCrLf
response.write "history.go(-1);" & vbCrLf
response.write "</script>" & VbCRLF
response.end
end if

if txtnum="" and IsValidnum(txtnum) then
response.write "<script language='javascript'" & VbCrLf
response.write "alert('请输入订购的音乐专辑数量!');" & VbCrLf
response.write "history.go(-1);" & vbCrLf
response.write "</script>" & VbCRLF
response.end
```

end if

上面这段程序的作用就是有效性检查。对于客户名、客户地址以及付款方式、客户种类等，需要检查它们的内容是否为空，假如为空，就需要用户重新填写，因为这些内容是必填的对象。对于音乐编号、邮政编码以及订购数量等，则不但要判断它们是否为空，还需要判断它们填入的内容是否有效。因为音乐编号是有规定的，也就是必须为“KH-*”的式样，而邮政编码必须符合国家规定，还有订购数量对象必须要填的内容为数字类型的字符串。这些功能通过三个函数来实现，它们分别是 IsValidId(), IsValidZip()和 IsValidNum()。

程序清单 8.30:

```
function IsValidId(id)
dim names, i, c
IsValidEmail = true
names = Split(id, "-")
if UBound(names) <> 1 then
    IsValidid = false
    exit function
end if
if names(0) <> "KH"
    IsValidid = false
    exit function
end if
for i = 1 to Len(name(1))
c = Lcase(Mid(name(1), i, 1))
if not IsNumeric(c) then
    IsValidEmail = false
    exit function
end if
next
next
end function
```

这里就不再介绍其他两个函数了。相对而言，IsValidId()这个函数是最难的，因为我们必须保证它的前缀为“KH-”，并且后面部分是数字，否则它就是一个无效的音乐编号输入。这里使用 Split()函数分离字符串，然后分别对前后两部分进行检查。当然可能还有其他更好的检查方法，读者可以自己去写。下面来看整个程序的最后部分。

程序清单 8.31:

```
txtid=server.htmlencode(txtid)
txtname=server.htmlencode(txtname)
txtaddress=server.htmlencode(txtaddress)
txtzip=server.htmlencode(txtzip)
txttype=server.htmlencode(txttype)
```

```
txtbox=server.htmlencode(txtbox)
txtcontent=server.htmlencode(txtcontent)
txtkhtype=server.htmlencode(txtkhtype)
txtnum=server.htmlencode(txtnum)
strSourceFile = Server.MapPath("/") & "music-book.xml"

Set objXML = Server.CreateObject("Microsoft.XMLDOM")
objXML.load(strSourceFile)
Set objRootsite = objXML.documentElement
txtkhid=NextCode objRootsite, "客户编号"
dim perboks
set perboks=objXML.createElement("预订")
set perbok=oParser.createElement("客户")
perbok.setAttribute "种类", txtkhtype
AddElement perbok, txtkhid, "客户编号"
AddElement perbok, txtname, "客户名"
AddElement perbok, txtaddress, "详细地址"
AddElement perbok, txtzip, "邮政编码"
perboks.appendChild(perbok)
AddElement perboks, txtid, "音乐编号"
AddElement perboks, txtnum, "数量"
AddElement perboks, txttype, "付款方式"

objRootsite.appendChild(perboks)
Set perbok=nothing
```

本段程序的主要目的就是把收集到的信息写入 XML 文档中。首先打开已经存在的 XML 文档，读取我们需要的客户编号形成下一个客户编号，然后把必要的信息写入 XML 文档。

关于文档的写入说明不再细说，读者可以自己体会。这里主要介绍两个函数，一个就是形成下一个客户编号函数 NextCode ()，还有一个是 AddElement () 函数。NextCode 函数比较麻烦，因为系统可能不停的运作，有可能发生 XML 文档中没有客户编号（刚刚被存入数据库中并且内容全部清空了）的情况，如果这样，我们就需要到数据库中取读取客户编号来形成下一个客户编号。AddElement 函数相对比较简单，不再赘述。

程序清单 8.32:

```
Sub AddElement(root, txt, node)
If Not IsNull(txt) then
    Dim oNode
    Set oNode=objXMLObject.createElement(node)
    ONode.text=txt
    Root.appendChild(oNode)
    Set oNode=nothing
```

```

End if
End sub
sub NextCode(root, node)
    dim txt, i
    i=root.childNodes.length()-1
    if i<> 0 then
        txt=root.childNodes.item(i)
        names=split(txt, "-")
        txt=IntToStr(StrToInt(names(1))+1)
        NextCode="KH-"+txt
    else
        dim rs
        rs=GetMdbRecorSet("music.mdb", "订购")
        rs.moveLast
        txt=rs("客户编号")
        names=split(txt, "-")
        txt=IntToStr(StrToInt(names(1))+1)
        NextCode="KH-"+txt
    end if
end sub

```

最后，把程序的后续部分写出来如下所示。这段代码的作用要给出信息已经成功写入文档的提示。

程序清单 8.33:

```

showmessage "你预订的资料已经记录下来了！"
%>
<%
Sub ShowMessage( msg )
%>
<html>
<head><meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<link rel="stylesheet" href="style/style.css" type="text/css"></head>
<CENTER>
<%=msg%><HR width="400"/>
<FORM action="music.xml" method="get">
    <INPUT Type="submit" Value="退出" class="f1">
</FORM>
</CENTER>
</html>
<%
Response.End
End Sub %>

```

8.4.4 后续优化工作

一个程序写好以后都需要优化的。对该程序，可以从下面几方面来考虑。

对于音乐显示，可以为每个音乐专辑做一个链接，用来表明每张专辑的详细介绍。当然也可以对歌手做链接。

对于音乐预订的 asp 页面，更多的考虑是如何使客户填写更加容易。因此可以把它和音乐显示链接起来（这是通常网站都采用的方法），也就是说在每张音乐专辑的后面都加上定购的链接，当用户点击了该定购后，系统会自动记录下该音乐专辑的音乐编号，用户只需要填写自己的必要信息就可以了，不必再填写音乐的编号信息了。

上面的几个网页都用到了一个 css 文档，这里也把这个文档写出来以供读者参考。该文档如下。

程序清单 8.34:

```
a { font-size: 9pt; color: #0000aa; font-family:"宋体"; text-decoration:none }
A:link { font-size:9pt;font-family:"宋体"}
A:visited { font-size: 9pt;font-family:"宋体"}
A:active { font-size: 9pt;font-family:"宋体"}
a:hover { font-size: 9pt;font-family:"宋体"; color: #ff0000; text-decoration:underline}
body { font-size: 9pt;font-family:"宋体"}
table { font-size: 9pt;font-family:"宋体"}
input { font-size: 9pt; font-family: 宋体 }
select { font-size: 9pt; font-family: 宋体 }
.f1 {BORDER-BOTTOM: 1px inset; BORDER-LEFT: 1px inset; BORDER-RIGHT: 1px inset;
BORDER-TOP: 1px inset; FONT-SIZE: 12px; HEIGHT: 14pt; VERTICAL-ALIGN: text-bottom}
.f2 { BACKGROUND-COLOR: #cccccc; BORDER-BOTTOM: 1px ridge; BORDER-LEFT: #ffffff
1px ridge; BORDER-RIGHT: 1px ridge; BORDER-TOP: #ffffff 1px ridge; FONT-SIZE: 12px; HEIGHT:
14pt}
```

8.5 小 结

虽然已经建立了大量的方法去完成任务，XML 和数据库之间仍然有很大的距离。但是我们也应该看到，许多数据库厂商的执行官都表示，将给数据库加入 XML 功能，也把数据库的功能带给 XML。因此可能在不久，我们就能够很容易的控制数据库和 XML 之间的信息交换了，而不再需要本章介绍的这些麻烦的通过控制节点来对 XML 文档进行操作的方法了。

第三篇 XML 工业应用

- ☒ 第 9 章 WML(无线应用协议)
- ☒ 第 10 章 SMIL(同步多媒体集成语言)
- ☒ 第 11 章 XML 与电子商务
- ☒ 第 12 章 XML 扩展

第 9 章对 WML(无线应用协议)做了一个详细的介绍。无线通信协议在将来的应用会越来越广泛和深入生活, WML 则为这种通信提供了介质。通过编辑 WML, 可以用手机像上网般浏览各类网站。可以说不久的将来, WML 就像台式机所浏览的 HTML 一样。

第 10 章对 XML 语言的另一个高级应用 SMIL(同步多媒体集成语言技术)做了一个详细介绍。SMIL 采用 RTSP 协议(Real Time Streaming Protocol, 实时流传输协议), 用类似 HTML 的标记方式对 Web 上多媒体展示的时间和布局控制进行了规定, 安排了网页上的影像、声音及文字各部分的时间流程, 从而开创了 Web 上多媒体的创作和展示的新阶段。

第 11 章对电子商务以及 XML 在电子商务中的应用做了详细介绍, 重点阐述了 XML 在现代电子商务中的应用。同时还介绍了 XML BizTalk 电子商务框架, 以及如何运用 BizTalk 框架来开发特定的电子商务。

第 12 章是本书的最后一章, 主要是补充并总结整本书的内容。先讲述了 XML 的标准体系, 并且对于 XML 中的数据也作了一些必要的说明。此外还承接前面几章, 对 XML 的扩展和应用进行进一步的说明。



第 9 章 WML(无线应用协议)

本章导读

WAP(Wireless Application Protocol)就是“无线应用协议”的意思。它由一系列协议组成,用来标准化无线通信设备,例如蜂窝电话,无线电收发机,也可用于 Internet 访问,包括 E-mail, WWW, Newsgroups 和 IRC (Internet Relay Chat)。

WAP 的协议包括以下四个层次:

- ☒ Wireless Application Environment (WAE)
- ☒ Wireless Session Layer (WSL)
- ☒ Wireless Transport Layer (WTP)
- ☒ Wireless Transport Layer Security (WTLS)

WAP 由 Erisson, Motorola, Nokia, Unwired Planet 四家公司发起,现在其会员已包括像 Alcatel, AT&T, 西门子, 英国电信, 法国电信, 贝尔大西洋, 贝尔南方等。目前 WAP 已经开始投入了实际的应用。最典型的如 SmartPhone, 可以用移动电话来收发 E-mail, 查询信息, 付账, 实现 PIM 功能等, 多么美妙! 当然要提醒你, 不要忘了要想在移动电话和掌上设备的狭小 LCD 上看到漂亮的 Web 画面在目前还是不现实的, 所谓的上网功能是指收发文字 E-mail。而查询信息则要靠特定的 ISP 来实现, 不是单靠移动电话厂商能办到的, 所以在 WAP Forum 中有不少电信公司也参与了 WAP 的实现。

移动商务被人们看好的一个重要原因是: 手机的普及速度远远超过了 PC。单从数量上看, 世界范围内的手机拥有量确实在经历一个飞速增长时期。但经济学家也指出, 如果单单考虑数量而不顾及其他的影响服务质量的因素, 那么, 对于 WAP 的探索无异于瞎子摸象。现阶段对于移动商务来说最急需解决的是信息获得成本高、效率低的问题。相对于计算机来说, 手机的显示屏幕太小, 这使得用户在单位时间获得单位信息与所需支付的费用比率达到了令人难以接受的程度。虽然, 以 IBM 为代表的语音技术商提出了以“听”代“读”的解决方案, 但是以目前的语音识别和合成技术水平, 尚不能说这条道路一定可行。

目前仅限于简单语句和标准发音水平上的识别技术, 对于一个将市场定位于普通消费群体的产品来说, 所实现的功能还是太过于简单了。大众的文化教育层次和对 IT 技术的掌握程度之间的巨大差异造成了语音合成技术的先天不足。对于 IBM 提出的语音合成技术来说, 当前尚不存在既满足小尺寸要求、又满足大计算量要求的高性能芯片。

此外, 出于原始市场划分上的冲突, 目前移动商务还存在着严重的兼容性问题, 考虑到当年微软用了将近 5 年的时间才将 Windows 树立为标准的事实, 我们没有任何理由对该问题的迅速解决抱有乐观的态度。

当然, 对于商务来说, 安全保障应当是最先考虑和始终保证的一个问题。但正如有线网络中存在着恼人的黑客一样, 无线商务的“不掉线、免拨号”承诺, 虽然给了用户一个方便的信息交互环境, 但同时也为恶意的技术闯入打开了方便之门, 原因不言而喻: 你在网上的时间越多, 受到攻击的概率也就越大, 而就以前发生的手机黑客非法闯入一事来看, 要作到常在河边走而又不湿鞋, 恐怕不是一个轻而易举就能解决的问题。

总之，移动商务如果运转起来的话，首先要保证安全，其次要降低信息成本，再有就是要在技术上可行。

本章讲述的是 WML (Wireless Markup Language) 无线标签语言。WML 是 WAP 中用来标签传输给移动电话、PDA 等设备的数据的语言服务。关于 WAP 讨论的权威网站可以参考 <http://www.wapforum.org>。

9.1 WAP Server 的建立

建立一个 WAP 的站点，首先要有一个 WAP 的开发环境。这一节将以 Windows2000 的 IIS5.0 以及 Windows98 的 PWS 和 Unix 平台中使用最为普遍的 Apache 等三种 Web Server 为基础，来介绍如何利用它们来建立自己的 WAP Server。

9.1.1 预备知识

WAP 进行信息传输的部分是使用 HTTP 来进行的，也就是说，与现有的 WWW 的信息平台是一样的。正因为这样，现有的 Web Server 都可以经过一定的设置，改装成 WAP Server，以提供对无线装置的服务。

在 Web 服务器的设置中，读者一定不会对 PHP,ASP,CGI 等后缀名感到陌生吧。在 WAP 服务器中，提供服务的文件类型目前有五种，他们的后缀名分别是 wml, wmlc, wmls, wmlsc, wbmp。依次代表的文件类型是：WML, 原始文档；WML, 原始文档的二进制码；WML Script, 原始程序码；WML Script 二进制码；单色的 Wireless BMP 文档。这五个后缀名必须填加到 Web Server 的 MIME Type (多用途网际邮件扩充) 设定中，Web Server 才能正确处理 WAP 文档。这样，Web Server 就顺利的过渡到 WAP Server 了。

从 IIS, PWS, Netscape Enterprise Server 到 Apache，无论使用哪种 Web Server 软件，只要添加上面提到的五种 MIME Type，就能提供 WAP 服务了。这五种后缀名及其 MIME Type 的列表如表 9-1 所示：

表 9-1 WAP 的五种 MIME Type

后缀名	MIME Type
Wml	text/vnd.wap.wml
wmlc	application/vnd.wap.wmlc
Wmls	text/vnd.wap.wmlscript
Wmlsc	application/vnd.wap.wmlscriptc
Wbmp	image/vnd.wap.wbmp

有了这五个后缀名和它们的 MIME Type 后，可以开始一步步的设置我们的 Web Server，接下来我就将从经典的 Microsoft IIS 开始讲述 WAP Server 的设置过程。

9.1.2 从 Microsoft IIS5.0 到 WAP Server

目前大多数的读者使用的操作平台都是 Windows 2000 Professional 了。IIS5.0 作为 Windows 2000 的内部组件，可以在“控制面板”的“添加/删除程序”中的“添加/删除 Windows

组件”中选择安装或是删除 IIS5.0。安装的步骤非常的简易，只需要在 IIS5.0 前的方框打上勾，然后按回车键就行了，这里就不再赘述了。

安装完成后，我们先启动 IIS，进入它的管理界面，如图 9-1 所示。

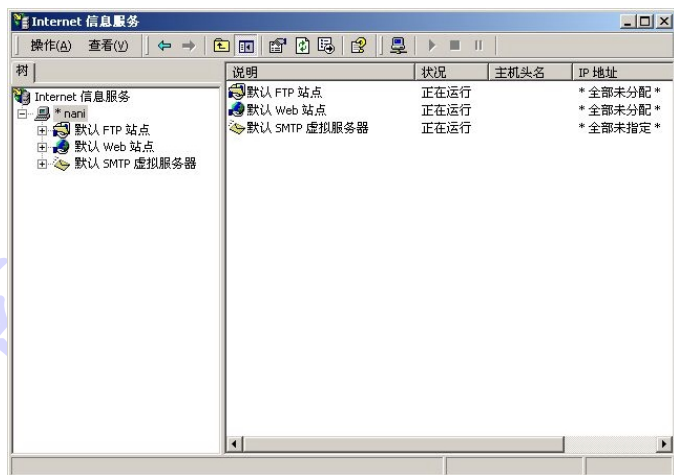


图 9-1 IIS5.0 启动界面

从图中可以看到默认 FTP 站点，默认 Web 站点和默认 SMTP 虚拟服务器三个选择项目。在默认 Web 站点处右键单击鼠标，在弹出的 POP 菜单中选择属性菜单。然后将会出现默认 Web 站点的属性设定界面，如图 9-2 所示。

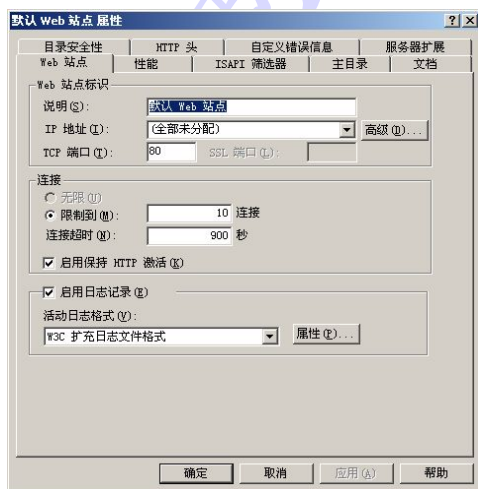


图 9-2 默认 Web 站点的属性设定界面

默认 Web 站点的属性设定界面内的内容相当的复杂，从设置 Web 站点的 IP 和端口号码到自定义服务器的出错信息。这里需要设定的 MIME Type 是在“HTTP 头”选项卡中的。点击“HTTP 头”选项卡，可以看见四大模块的设置。如图 9-3 所示，分别是：

启用内容失效

选择该复选框以包括对时间敏感的文档的失效信息。例如，股票的报价，新闻的公告

栏和某些赛事的现场报分等，浏览器将当前日期与失效日期进行比较，以便确定是否向服务器请求更新网页。

自定义 HTTP 头

该属性将自定义 HTTP 头从 Web 服务器发送到客户浏览器。自定义 HTTP 头可以用来发送当前 HTML 规范中尚不支持的指令。例如，产品发布时，IIS 尚不支持的更新的 HTML 标签。

内容分级

使用内容分级，将在 Web 页的 HTTP 头中嵌入描述性的标签。从 Internet Explorer 3.0 开始，浏览器都会检测内容分级以帮助用户识别可能有危险的 Web 内容。

MIME 映射

这就是工作的区域。选择“文件类型”按钮配置 MIME 映射。这些映射设置各种 Web 服务返回到浏览器的文件类型。

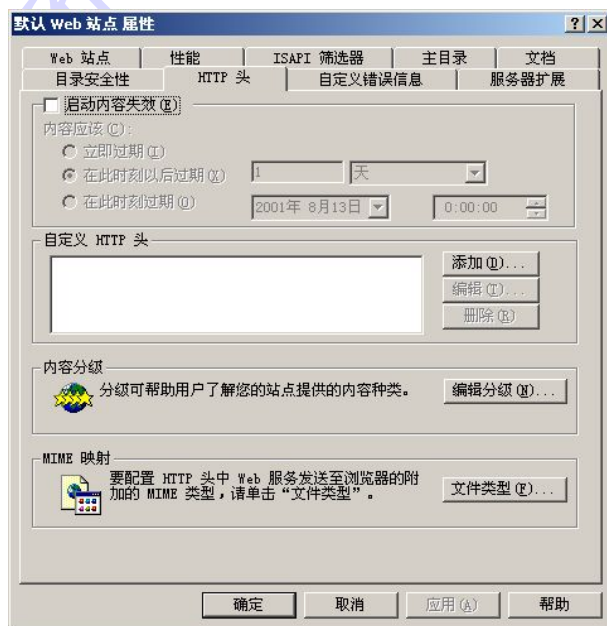


图 9-3 “HTTP 头”选项卡

点击“文件类型”按钮，出现了已注册的文件类型的列表。单击“新类型”，然后按照表 9-1 的数据类型，添加五个 MIME Type 映射，如图 9-4 所示。

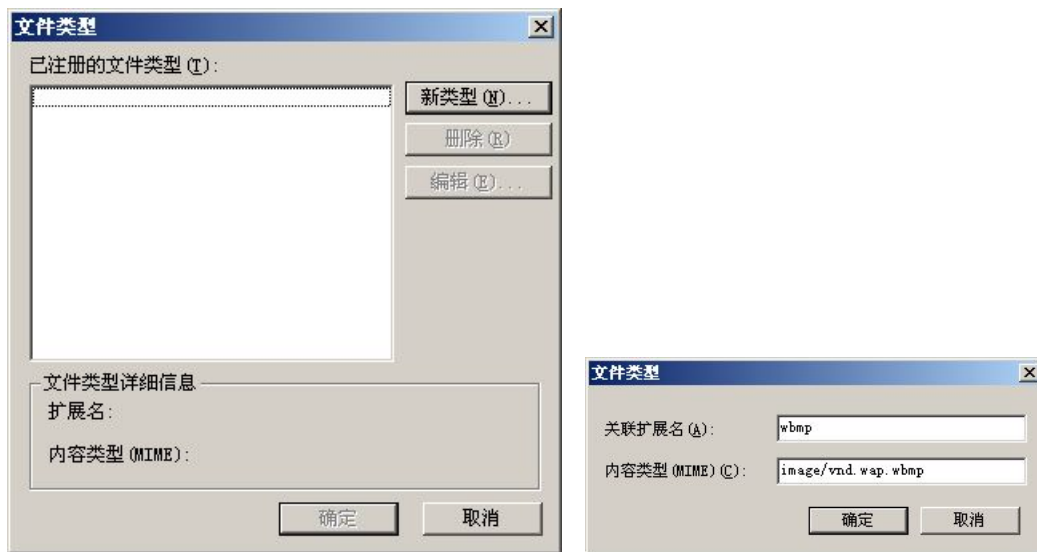


图 9-4 添加 MIME Type 映射

至此，完成了从 Microsoft IIS5.0 到 WAP Server 的设置工作，就可以在 IIS 中直接使用这些 WAP 文档了。

9.1.3 从 PWS 到 WAP Server

与 Microsoft IIS5.0 不同，PWS 中没有专门用来设置 MIME Type 映射的设定界面。PWS 的 MIME Type 映射是完全依靠 Windows 操作系统内部的 MIME Type 映射的。所以，我们与其说是修改 PWS，不如说是修改 Windows 操作系统。

打开一个浏览器窗口，选择文件夹选项，再选择“文件类型”选项卡。然后就和设置 IIS5.0 的方法一样，添加五个 MIME Type 映射就完成了 PWS 到 WAP Server 的进化。值得注意的是，Windows98 或者 Windows NT 在添加完 MIME Type 映射后必须重新启动，才能让所做的修改生效。

9.1.4 从 Apache 到 WAP Server

之所以在介绍了 IIS5.0 和 PWS 后还要介绍 Apache，是因为 Apache 是目前所有 Web Server 软件中支持平台数目最多的一个。Apache 支持 Windows NT，Linux 以及 Unix 操作系统。根据应用的广泛性来看，Linux 操作系统的使用者较多，因此，我们将主要以 Linux 配合 Apache 作为升级 WAP Server 的例子。

Apache 的前身是 NCSA 的 httpd。但是相对于 NCSA 的 httpd，Apache 新增加了许多功能，最显著的就是 Apache 提供了多线程处理程序的功能，这使得 Apache 的 Web Server 的工作效率明显优于它的早期版本 NCSA 的 httpd。

如果读者使用的是 Linux 操作系统，在安装了 Apache 以后，Linux 系统的 /etc 目录下就会新增一个 httpd 目录。这是用来存放所有的 Apache 的配置文件的目录。/usr/bin 目录将被用来存放执行文件。Apache 还有一部分的设置文件保存在 /etc/httpd/conf 目录下。

下面将要修改的设置保存在 /etc/mime.types 这个文件中。修改的方法十分的简便，打开

文件 mime.types, 读者将会发现它的格式比较简单, 一行就是一个 MIME Type 映射。要增加 MIME Type 映射, 只需要在 mime.types 文件中新增一行, 先输入 MIME Type, 然后空一个输入后缀名。以 wbmp 文件为例, 在文件中加入的代码如下:

```
image/vnd.wap.wbmp wbmp //MIME Type与后缀名之间必须有一个空格
```

输入完五项 MIME Type 映射后, 重启 Apache, 就完成设定了。

各种 Web Server 升级到 WAP Server 的方法大同小异, 如果读者使用的是比较特别的 Web Server 软件, 若不希望更换的话, 可以从新增 MIME Type 映射为目标, 对现有的 Web Server 进行改造。

当 Web Server 升级到 WAP Server 后, 就可以开展 WML 的撰写工作了。

9.2 WML 的软件使用

前几节中介绍了 WML 文档的编写, 那么使用什么样的软件来编写 WML 文档呢? 显然任何一款文本编辑器都可以用来编写 WML 文档, 甚至包括 Windows 自带的记事本程序。但是, 没有一个设计人员能永远保证自己的程序是完全正确的。设计者们需要一个能编写 WML 文档并且能检验 WML 文档的正确性的软件, 本书在 9.2.2 中将介绍 Nokia WAP Toolkit, 这是一个集合 WML 编辑器、WAP 模拟器和 WBmp 编辑器于一体的高效软件。有了 WML 文档编辑器还是不够的, 还需要一个 WML Gateway, 很遗憾 Nokia WAP Toolkit 并没有提供 WML Gateway 功能。虽然 Nokia 公司也提供了一个 Nokia WAP Server 程序, 但是其 10M 多的大小实在是让人望而却步。因此, 将在 9.2.1 中介绍 Infinite 公司提供的 Infinite WAP Server。

9.2.1 Infinite WAP Server

Infinite 公司同 Nokia 公司一样, 很早就参加了 WAP 技术的开发。他们的 WAP Gateway 产品名称叫做 WAPlite。下载的网址如下:

<http://www.infinite.com/captaris/download.asp?KEYWORD=WAPlite>

该网页如图 9-5 所示, 网页中有一些表格, 是 Infinite 公司对软件使用用户的信息调查表格。这些信息是完全保密的, 你可以放心的填写。WAPlite 有 30 天免费试用期, 在试用期后, 使用者可以去 Infinite 的网页重新下载 WAPlite 升级原来的版本。

WAPlite 最大的优点是在不失去 WAP Gateway 的主要功能的情况下, 整个软件的大小仅仅只有 655K, 与 Nokia WAP Server 的 10M 相比, 简直是个人设计编程人员的福音。目前的 WAPlite 版本号是 2.0。Infinite 公司对软件的更新速度较快, 如果读者在下载时的版本已经超过了 2.0, 只要不是测试版, 那么功能上是不会有任何差错的。

Captaris
Business within your reach.

Solutions | **Products** | **Services**
Calixpress
Infinite
MediaLing Services
OmniBridge
RightFax

Infinite
Products & Solutions
Demo
• **Downloads**
Training
Sales
Support
Market Overviews
Case Studies
FAQ
Press
Tradeshows & Events
Strategic Alliances
Careers
About Infinite
Contact Infinite
Original Infinite Site

DOWNLOAD FORM

To download a fully operational evaluation version of the product, please submit the form below. The product download version is not a demo, but it does have a 30-Day time limit.

If you have product questions, need assistance with an evaluation, or would like to serialize the product please contact the [Sales Department](#) and they will gladly assist you.

Fields marked with a * are required.

* Full Name:
E-Mail Address:
Is this download for: ☒ Evaluation ☐ Upgrade
How did you hear about us?
My organization is: ☒ End User ☐ Reseller ☐ Distributor
* Phone:
Fax:
Company Name:
* Address 1:
Address 2:
* City, State, Zip:
* Country: [China]

Thank you for providing your information. This information is used only for customer services purposes. Our lists are not rented, leased or sold, and they are never used for broadcast e-mails. Thank you for your interest!

[Home](#) | [Products & Solutions](#) | [Demo](#) | [Downloads](#) | [Discussion](#) | [Training](#) | [Sales](#) | [Support](#) | [Market Overviews](#) | [Case Studies](#) | [FAQ](#) | [Press](#) | [Tradeshows & Events](#) | [Partners](#) | [Strategic Alliances](#) | [Careers](#) | [About Infinite](#) | [Contact Infinite](#) | [Original Infinite Site](#)

Copyright © 2001 Captaris, Inc.

图 9-5 WAPLite 的下载网页

WAPLite 的安装跟一般的应用程序没有什么差别，只是在安装进度条到达 100% 的时候会弹出一个对话框，如图 9-6。

License Authorization

Serial Number:
User Count:
Authorization Code:

Select "Cancel" to install 30-day trial version

图 9-6 输入 WAPLite 软件的注册信息

这是 Infinite 公司为了确定你是否是公司的正式用户还是普通的试用用户。如果你只是想试用这个软件，那么按下 **Cancel** 键就可以顺利的完成安装，并且得到 30 天的免费试用期了。注意，在安装完成后必须重新启动计算机才能使软件生效。

启动 WAPLite2.0，并且按下 **Install Service** 按钮就可以启动 WAP Gateway 的服务了。具体的界面如下图 9-7。

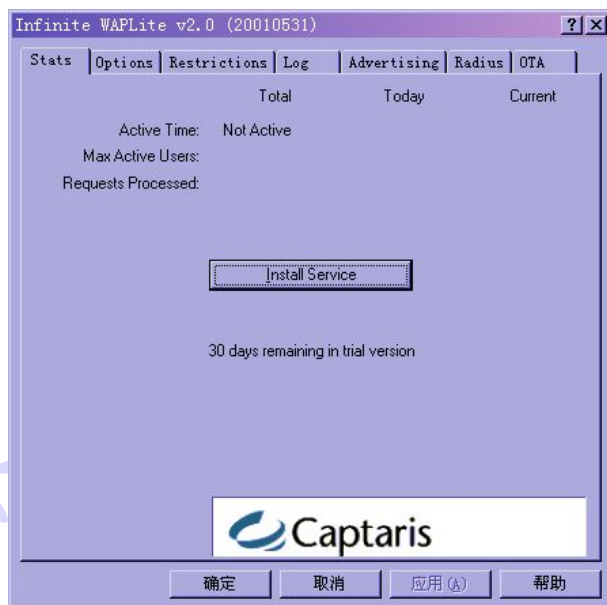


图 9-7 启动和设置 WAPLite2.0

开启服务后，WAP Server 就正式启动了。其他的设置使用软件的默认设置就可以了。

介绍了 WAP Server 软件，接下来我们将介绍模拟手机环境并且集成 WML 编辑器的软件：Nokia WAP Toolkit。

9.2.2 Nokia WAP Toolkit

Nokia WAP Toolkit 目前的版本是 3.0，并且已经改名为 Nokia Mobile Internet Toolkit 3.0。我们将以经典的 Nokia WAP Toolkit 2.1 为说明的理由。至于 Nokia Mobile Internet Toolkit 3.0，它的使用与 2.1 版本没有太大的区别。但是在文件的大小上由以前的 2.1 版本的 18M 上升到了 3.0 版本的 21.5M。3.0 的运行环境也是与 2.1 一样，注意需要 Windows 操作系统与 Java 2.0 SDK 的运行环境。

Nokia 公司提供了 Toolkit 的免费下载路径：

http://forum.nokia.com/wapforum/main/1,,1_1_50_30,00.html

下载的连接页面如图 9-8 所示，在进入这个界面时，你需要在 Nokia WAP 论坛先注册一个账号。这个账号是完全免费的。

Toolkit 的软件大小比较大，而且不能使用 FlashGet 等下载软件下载，因此如果读者是拨号上网的用户，那么在下载时请预先准备好大约 5 小时的下载时间。

Toolkit 的安装十分简易，这儿就不赘述了。

在确定安装了 Java 2.0 SDK 运行环境后，启动 Toolkit，Toolkit 的启动需要较长的时间。启动后，读者将会看到一个精美的 LOGO，如下图 9-9：

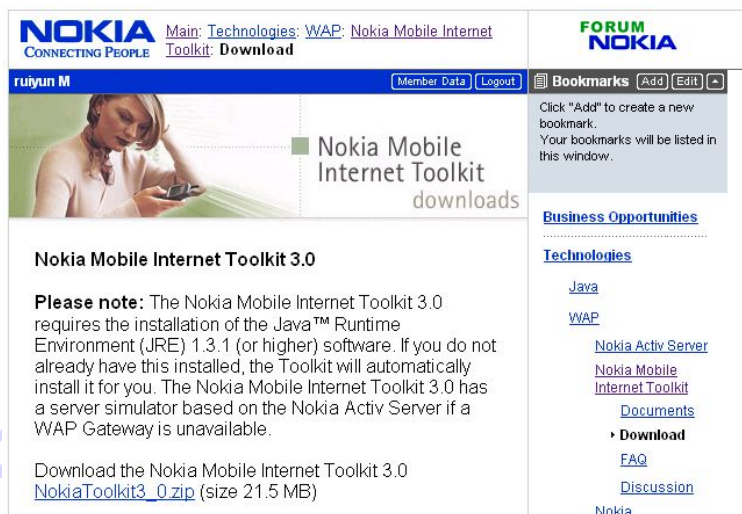


图 9-8 Toolkit 的下载网址

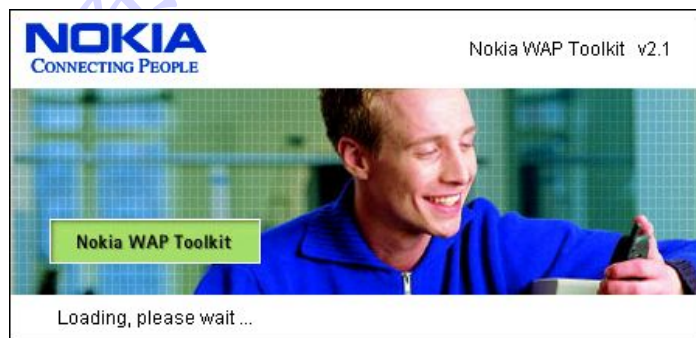


图 9-9 Toolkit 的启动运行界面

经过一段启动时间后，将会出现两个窗口。如图 9-10。

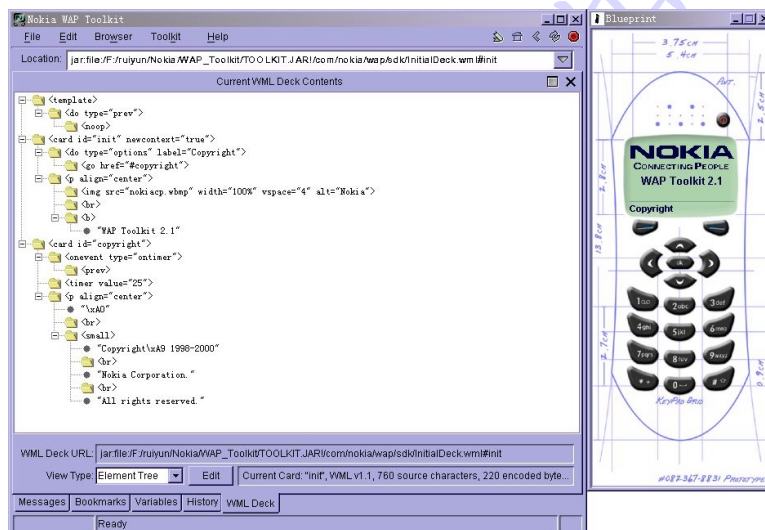


图 9-10 Toolkit 的使用界面

左边的窗口是 WML 编辑器，右边的那个就是 Blueprint 手机模拟界面。本书的实例分析都是在 Blueprint 界面下载取的效果图。关于手机模拟界面就不多描述了，读者可以在上面使用像手机一样的按键，浏览自己编写的 WML 文档。值得一提的是模拟器与手机还是有一定微弱差异的，读者如果想得到很精确的屏幕效果，那么还是要将自己的作品发表到 Internet 上，然后通过手机浏览，以手机的现实效果为准。

下文将简单的介绍一下 Toolkit 的使用方法。Toolkit 分为以下几个功能菜单：

- ☑ 文件 (File)：有关新建、保存文件以及推出系统等功能。
- ☑ 编辑 (Edit)：有关撤销、剪切、粘贴、查找等操作的实现。大多数软件都有以上两个菜单的功能。
- ☑ 浏览 (Browse)：主要有浏览主页、历史中的前一页、增加书签、装载 WML 文档等功能。这部分内容与 Internet Explorer 的使用类似，就不做什么介绍了。
- ☑ 工具箱 (Toolkit)：这是重点介绍的部分
- ☑ 帮助 (Help)：这个部分有 Toolkit 的帮助文件，是 PDF 格式的，读者需要有 Acrobat Reader 软件才能打开。

以上的功能中，读者比较关心的应该是工具箱部分。工具箱部分的子菜单有显示 (Show)，选择模拟设备 (Select Device) 和模拟设备设置选项 (Device Settings) 三个部分。

- ☑ 显示 (Show)

显示部分的子菜单很多，如图 9-11。

每个选项都对应这一个主窗口底部的选项卡。从上到下代表的选项卡是 Toolkit 信息记录，浏览器标签，WML 页变量内容，WML 页浏览历史记录，当前 WML 页的树形结构，浏览器 Session 和缓冲信息，主动推送信息窗口，手机模拟设备窗口。

在这些选项卡中，当前 WML 页的树形结构是使用频率最高的。它显示了当前浏览的 WML 页的树形结构，这一点对设计者来说十分重要。能很轻松的看到自己的 WML 框架的问题所在。以 Toolkit 自带的 welcome1.wml 为例，在树形结构显示的结果如图 9-12：

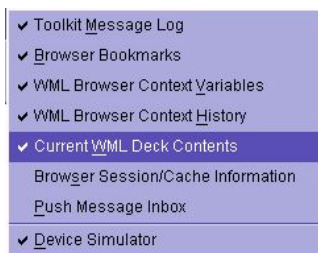


图 9-11 显示菜单介绍

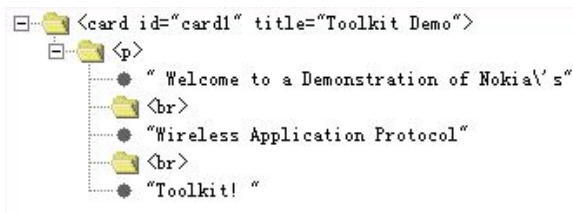


图 9-12 welcome1.wml 的树形结构

每打开一个 WML 文档系统都会自动增加一个选项卡，其中就是该文档详细代码。建议使用树型结构来编辑 WML 文档，这种编辑方法的优点是错误率小，系统自动性高。

- ☑ 选择模拟设备 (Select Device)

2.1 版本只提供了 Blueprint 模拟设备，读者如果喜欢别的显示界面可以去 Nokia Wap

论坛，在下载 Toolkit 的下方有模拟设备界面的下载。模拟设备之间在功能上是没有任何区别的，推荐使用默认的 Blueprint 界面即可。

☒ 模拟设备设置选项（Device Settings）

打开模拟设备设置选项菜单，可以看见如图 9-13 的选项设置区。

Communication 选项卡是用来设定连线状态的。如果不需要通过 WAP Gateway 浏览的话，选择默认的 Use HTTP Server Connection 就可以了。如果需要通过 WAP Gateway 连接，则需要选择 Use WAP Gateway Connection，并且设定 WAP Gateway 的 IP 地址。

Push 选项卡是用来设置主动的推送信息的方法和几项参数设置。其中包括 Push 使用的连接状态，当信息到达时自动设定 Push 选项，等待 Push 信息三个可选的参数。关于 Push，在主窗口的 Push 选项卡中有很明了的界面解释。

Cache 选项卡可以用来设置系统缓冲的大小。

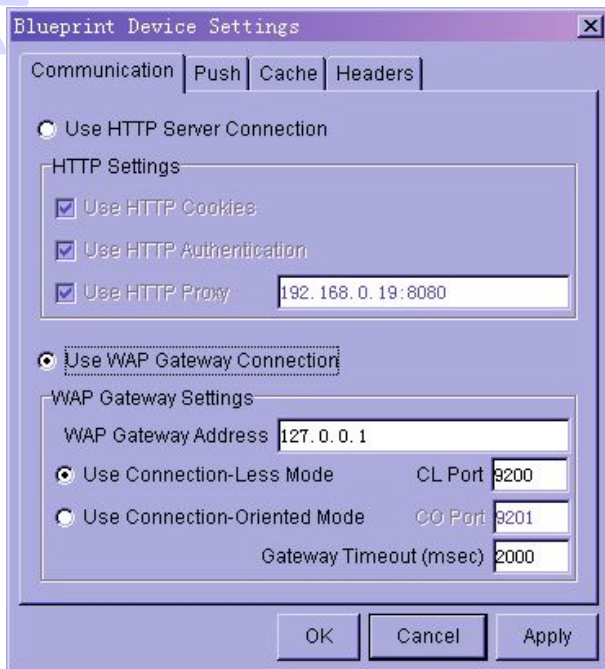


图 9-13 配置 Toolkit 属性

9.3 WML 语法规则

开始介绍 WML 的语法规则之前，先介绍一下 WML 文档的特性。

首先，WML 是为具有以下特点的设备而设计的：

- ☒ 体积小（相对于 PC 机）。
- ☒ 极其有限的内存和 CPU 大小。
- ☒ 通讯带宽较窄和时延长。

因此，对于一个 DECK（见 9.4.3 中的 WML 文档的基本框架），其文件大小最好不要超过 1.2K。

其次,读者应该注意到,WML 的语法跟 XML 一样。WML 文档和 HTML 与 XML 的文档看起来有很多相似之处。WML 的这一优点在于使得网页开发者们在过去 10 年中所学的东西今天依然适用。WML 本身就是一种 XML (可扩展标签语言),比如说要在屏幕上显示“大于”和“小于”,就必须用实体标签符号即 `lg` 和 `gt` 来表示(关于标签符号将在 9.4 中介绍)。

下面将对 WML 语法规则做详细的讲述。读者在编写 WML 文档的时候,必须遵循这些规则。

9.3.1 字符集

WML 使用 XML 的字符集,也就是当前使用的通用字符集 ISO/IEC-10646(Unicode 2.0),同时也支持其他系列的子集,例如,US-ASCII, ISO-8859-1 或者 UTF-8。这样就不必使用整个 Unicode(UCS-4)编码,除非你正在使用的不是 UTF-8 编码。

9.3.2 大小写敏感

与 HTML 不同的是,WML 是一种大小写敏感的语言。所有的标签、属性和枚举属性都必须使用小写。所以读者在编写 WML 页面的时候,必须注意到大小写问题。甚至包括参数的名字和参数的数值都是大小写敏感的。例如: `variable1`, `Variable1` 和 `vaRiable1` 就是不同的三个参数。

9.3.3 WML 文档的基本框架

WML 不支持嵌套标签注释。WML 语法采用 `card` 和 `deck` 的结构,一个 WML 文件称为一个 `deck`,一个 `deck` 可包含多张 `card`。查看 `deck` 内不同的 `card` 可得出 WML `deck` 的页面结构导航。WML 通过一次性把 `deck` 的所有 `card` 发送给浏览器便可实现对网络内容的各种处理了。用户无需再次进入网络即可从各个方向实现对这些内容的浏览。

WML 文件的一般格式如下:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM/DTD WML 1.1//EN"
    "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
    <head>
        <access/>
        <meta.../>
    </head>

    <card>
        .....
    </card>
    <card>
        .....
    </card>
```

.....
<wml>

WML 文档的结构看上去和 HTML 文档的结构很类似，对于每一个 DECK，在其文档开头必须指明以下的类型声明。

```
<?xml version="1.0"?>  
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"  
"http://www.wapforum.org/DTD/wml_1.1.xml">
```

在前面已经提到过了字母的大小写敏感问题，所以读者千万要注意字母的大小写。

9.3.4 WML 文档中的保留符号

在 WML 中，一共有八个保留符号，如表 9-2 所示。

表 9-2 WML 的保留符号

保留符号	意义
'	单引号
"	双引号
<	小于号
>	大于号
 	不断行空白字符
­	连字符
&	符号 “&”
$	符号 “\$”

应该注意的是保留符号的末尾都有一个 “;”，这是保留符号的一部分，读者在编写文档时千万不要遗忘。除了这八个保留符号外，还可以通过
SCII 字码或者是通过&#ASCII 字码两种形式来引用 ASCII 字符。其中，第一种形式使用的是十六进制数，第二种形式使用的是十进制数。例如，“B”和“B”代表的都是字符“B”。但是以这种形式编写的文档可读性相当的低。

9.3.5 标签的属性

许多 WML 标签有一个或多个属性，有些是必须的，有些则是可选择的。属性让你指定附加的信息，用来确定设备如何解释这些标签。WML 标签有以下两种使用方式：

```
<element a1="value1" a2="value2" ...> content </element>  
  
<element a1="value1" a2="value2" .../>
```

必须使用成对的单引号或者双引号将属性值包含在内，而且每个属性之间要使用空格

分开。

9.3.6 引用变量

在 WML 中引用变量，很像在 Unix 下引用 Shell 变量，可以在格式化的文本、URLs、选择文本等中使用变量。WML 可以在 DECK 的某张 card 内定义变量，然后通过把一个值存储在该变量中，让别的 card 可以直接显示这个变量值，而无需对整个 deck 进行重复的存取和显示。变量结构如下显示：

```
$变量名
$(变量名)
```

注意，在变量名中存在空格时，就使用第二种引用方案，即在变量名外加上圆括号。

9.4 WML 中的标签

受到外部接受设备的性能限制，所以 WML 中的标签只保留了 HTML 和 XML 标签中的一小部分。根据功能上的划分，整理为以下的表 9-3。

表 9-3 WML 的标签的功能分类

页面与卡片		文本格式	
<wml>			
<card>		<big>	
<template>		 	
<head>			
<access>		<i>	
<meta>		<p>	
	用户输入	<small>	
<input>			
<select>		<table>	
<option>		<td>	
<optgroup>		<tr>	
<fieldset>		<u>	
	定时		事件
<timer>		<do>	
	参数	<onevent>	
<setvar>			
	锚		任务
<anchor>		<go>	
<a>		<prev>	
	图片	<noop>	

页面与卡片	文本格式
	<refresh>

下面将所有标签的语法格式逐个详细介绍。

9.4.1 页面与卡片

<wml>标签

<wml>标签的语法格式:

```
<wml xml:lang="lang">
  content
</wml>
```

<wml>标签和 HTML 中的<html>标签一样, 用来表明这是一个 WML 的 DECK, 它有一个可选的 xml:lang 属性, 用来制定文档的语言, 比如<wml xml:lang="zh">表示文档语言为中文。

<card>标签

<card>标签的语法格式:

```
<card id="name"
  title="label"
  newcontext="boolean"
  ordered="true/false"
  onenterforward="url"
  onenterbackward="url"
  ontimer="url">
  content
</card>
```

<card>标签的各个属性如下介绍。

- ☒ id: 用来指定 Card 的名字, 可用来在 Card 间跳转, 相当于在 HTML 中在页内跳转时用。
- ☒ title: 用来作为标签的标记, 该属性一般不会显示在屏幕上。
- ☒ newcontext: 默认值为 false, 用来指示当跳转到该 Card 时, 手机是不是要清除以前保留的信息, 包括变量、堆栈里的历史记录和重新设置手机状态等。
- ☒ ordered: 默认值是 true, 用来设定该 Card 里的内容是按固定的顺序显示, 还是按用户的选择来显示, 这点和 HTML 不同。Card 页面里的内容可以按一定的顺序显示, 默认的是按线性顺序显示, 即按代码的顺序。

但是, 要注意的是, 下面三个标签必须按依次顺序来写<onevent>, <timer>, <do>(这

和以后要讲的“事件”有关)。这样做是为了方便填表单，当 `ordered` 设置为 `true` 时，如果一个表单的内容不能在一屏里显示完，就分成多屏显示；当 `ordered` 设置为 `false` 时，手机可以显示一个概要 Card 来总结有效的选项，用户可以从中学取表单选项来填写。

`<template>` 标签

`<template>` 标签的语法格式：

```
<template onenterforward="url"
  onenterbackward="url"
  ontimer="url">
  content
</template>
```

`<template>` 标签的属性分析如下：

`template`：模板元素，用于定义所有 `card` 共享的元素。例如

```
<template>
  <do label="Back" name="prev" type="prev">
    <prev/>
  </do>
</template>
```

以上的代码就是用来定义所有的 `Card` 中的 `Back` 键。`template` 标签的使用是相当广泛的，几乎每个 WML 文档都会使用 `template` 以减少工作量。有关于 `template` 标签的具体使用方法在 9.4.2 中会有详细的介绍。

`<head>` 标签

`<head>` 标签的语法格式：

```
<head>
  content
</head>
```

和 HTML 一样，`<head>` 标签包含了该 `DECK` 的相关信息。`<head>` 标签之间可以包含一个 `<access>` 标签和多个 `<meta>` 标签。

`<access>` 标签

`<access>` 标签的语法格式：

```
<access domain="domain"
  path="path"
/>
```


<access>标签相当于 HTML 中的<BASE>标签，用于指定该 DECK 的访问控制信息。它有两个可选的属性：domain 用来指定域，默认值为当前域；path 用来指定路径，默认值为"/"，即根目录。由于<access>单独使用，所以要用 "/" 结尾，以后对于类似的情况不再赘述。

<meta>标签

<meta>标签的语法格式：

```
<meta name="name"|http-equiv="name"  
      content="value"  
      forua="true|false"  
>
```

<meta>标签和 HTML 中的类似，提供了该 DECK 的 meta 信息。

name 属性是必选的，包括以下三种情况：

- ☒ name="name" UP.Link Server 忽略 meta 数据
- ☒ http-equiv="name" UP.Link Server 将 meta 数据转为 HTTP 响应头(同 HTML)
- ☒ user-agent="agent" UP.Link Server 直接将 meta 数据传给手机设备

content 属性也是必选的，其内容根据属性而定。scheme 属性目前尚不支持。

forua 属性为可选属性，指定在该 wml 文件传到客户端之前，<meta>标签可不可以被中间代理删除(因为传输的协议可能改变)，默认值为 false。

目前支持的 meta 数据：

```
<meta http-equiv="Cache-Control" content="max-age=3600"/>
```

指定 Deck 在手机内存缓存中的存储时间周期，默认值为 30 天（除非内存耗尽）。在这个期间，手机对于访问过的 Deck 直接从缓存里调用。如果信息是对时间敏感的，可以用 max-age 指定 Deck 在缓存里的生存期，最小单位是秒，如果指定为 0，则每次都需通过连接服务器来调用该 Deck。

```
<meta user-agent="vnd.up.markable" content="false"/>  
<meta user-agent="vnd.up.bookmark" content="指定的URL"/>
```

以上的两个代码也类似于普通浏览器的书签功能。当用户将一个 Card 做了书签后，手机浏览器首先用一个标记记录该 Card，这个标记默认的是<card>标签中的 title 属性（在后面马上会提到的），然后当用户选择了该书签以后，浏览器就会打开被记录的 URL。但是因为在默认的情况下，手机会记录所有的 Deck，所以，一般<meta>被用来使手机不要记录当前的 URL，即：

```
<meta user-agent="vnd.up.markable" content="false"/>
```

此外，如果要为书签指定不同于当前 DECK 的 URL，可用以下代码：

<meta user-agent="vnd.up.bookmark" content="指定的URL"/>。

9.4.2 用户输入

<input>标签

<input>标签的语法格式：

```
<input name="变量名"
      title="标题"
      type="text/password"
      value="value"
      format="文字控制格式"
      emptyok="true/false"
      size="n"
      maxlength="n"
      tabindex="n"
/>
```

<input>标签的各个属性如下简要分析：

- ☒ type: <input>标签中只有两个元素（type 和 name）是必须的，其他的都是可选的。Type 属性定义的是输入栏的显示模式，可填的选择项是 text 或者 password。当设置为 text 的时候，输入的字符将以纯文字显示；而设置成 password 的时候，输入过程中以纯文字方式显示，输入完成后以星号代替纯文字。
- ☒ name: 在 name 后面被赋值的变量，将在整个 WML 的 Deck 中传递数据。
- ☒ title: “title”就是要显示的选项的名称。因为在手机的显示界面是以选单的方式进行输入的。如果 title 没有被设定的话，手机就会使用默认值来代替 title 的位置显示。
- ☒ value: 定义输入栏的初始化数值。
- ☒ format: 该属性定义的是文字的控制格式，用来要求使用者所输入的文字格式符合特定的文字格式。下面将文字格式的写法与其所代表的格式意义列成表格，如下 9-4 所示：

表 9-4 format 的文字格式

格式写法	意义
A	必须是大写的英文字母或标点符号，不能是数字
a	必须是小写的英文字母或标点符号，不能是数字
N	必须是数字，不能是其他的字符
X	必须是除了小写英文字母以外的字符
x	必须是除了大写英文字母以外的字符
M	输入任何字符，为了输入简单，用户可以设定所有的字符都是大写的，

格式写法	意义
	但输入时允许输入任何字符
m	输入任何字符，为了输入简单，用户可以设定所有的字符都是小写的， 但输入时允许输入任何字符
\指定的字符	在指定的位置自动显示\后的指定的字符

具体的 format 的写法比较难懂，这里举几个例子帮助读者的理解。

(1)学生证号：一个 8 位的数字，表示符号如下：

```
format="NNNNNNNN"
format="8N"
```

以上的两个表达方式是等价的，但是数字位数超过 10 就不能用第二种简化的写法了。

(2)带区号的电话号码：一个 4 位的区号，一个“-”连字符号，8 位数字的电话号码，表示符号如下：

```
format="NNNN/-NNNNNNNN"
format="NNNN/-8N"
```

读者一定注意到第二个表示法中没有使用 format=“4N/-8N”，因为数字加格式定义符的这种简化写法只能在格式字符串的末尾出现，并且只能使用一次。

- ☒ emptyok: 根据元素的名称可知，该元素定义输入栏是否允许是空。默认值是 false，也就是说，默认为必须输入字符。
- ☒ size: 定义输入栏的大小。
- ☒ maxlength: 定义输入栏可以容纳的字符的最大值。
- ☒ tabindex: 定义在手机光标在各个项目栏之间移动时的顺序，数字越大的项目栏就越晚被选定。

<select>标签

<select>标签的语法格式：

```
<select title="标题"
  multiple="true/false"
  name="variable"
  iname="index_var"
  ivalue="default"
  tabindex="n">
  content
</select>
```

<select>标签的属性分析如下所示：

- ☒ **title:** 与 input 的 title 元素相同。
- ☒ **name:** 这是 select 标签唯一的必须的元素。它定义的是指定给 select 这个标签的变量名。
- ☒ **value:** 定义为指定给 select 标签的变量的默认值。当 multiple 的属性值为“true”时，可以直接在 value 处定义多项预设值。定义的语法如下：

value="第一个预设值;第二个预设值"

- ☒ **multiple:** 可选的属性值为 true 或 false。默认值为 false，当 multiple 的属性值被赋值为 true 时，select 元素可以重复选择不同的选项。
- ☒ **iname:** 指定给 select 元素的索引变量名。
- ☒ **ivalue:** 指定给 select 元素的索引预设值。
- ☒ **iname** 和 **ivalue** 属性，将在 9.4 的范例中再做详细的介绍。

<option>标签

<option>标签的语法格式：

```
<option title="标题"
      value="value"
      onpick="url">
  content
</option>
```

<option>标签的属性分析如下：

- ☒ **option:** 用来显示项目的字符串。
- ☒ **value:** option 元素中唯一必须的属性，是项目所对应的信息。是直接显示在手机上的信息。被选择的时候，value 的属性值被赋值给 select 中 name 定义的变量。
- ☒ **onpick:** 手机在选取选项后被重定向到 URL。

<optgroup>标签

<optgroup>标签的语法格式：

```
<optgroup title="标题">
  content
</optgroup>
```

- ☒ **optgroup:** 选项群组元素，与 option 元素混合使用，将几个类似的 option 编制成一组。类似与 VB 中的 frame 的功能。

9.4.3 文本格式

这部分与 HTML 中的含义大致相同。因为比较简单，相同部分只给出基本的语法格式。

读者完全可以从语法格式中理解各个元素的使用方法。在不同部分将给出必要的注解，读者可以特别注意一下特别注解的部分。

标签

标签的语法格式：

```
<b>
  text
</b>
```

text 内容被显示为粗体。

<big>标签

<big>标签的语法格式：

```
<big>
  text
</big>
```

☒ big：将文字变大，因为手机上只有一种大尺寸的字体，所以无需指定字体的具体大小，而直接书写想要显示为大字体的文字就可以了。

**
标签**

**
标签的语法格式：**

```
<br/>
```

标签

标签的语法格式：

```
<em>
  text
</em>
```

<i>标签

<i>标签的语法格式：

```
<i>
  text
</i>
```

<p>标签

<p>标签的语法格式:

```
<p align="alignment" mode="wrapmode"/>
```

<small>标签

<small>标签的语法格式:

```
<small>
```

```
text
```

```
</small>
```

☒ small: 与 big 相反, small 是将文字变小。手机只有一种小字体。

标签

标签的语法格式:

```
<strong>
```

```
text
```

```
</strong>
```

<table>标签的语法格式

```
<table align="alignment" title="标题" columns="n"/>
```

<td>标签的语法格式

```
<td>
```

```
content
```

```
</td>
```

<tr>标签的语法格式

```
<tr>
```

```
<td>
```

```
content
```

```
</td>
```

```
</tr>
```

<u>标签的语法格式:

```
<u>
    text
</u>
```

9.4.4 图片

标签的语法格式:

```

```

标签的属性分析如下所示:

- ☒ **src**: 要手机显示的图形的 URL, 必须是 **wbmp** 这种单色图形格式的文件才能被手机显示。
- ☒ **localsrc**: 如果 **src** 所指针的图形文件不存在的话, 手机将自动显示 **localsrc** 所指针的图标。
- ☒ **alt**: 如果 **src** 所指针的图形文件不存在的话将显示在图形位置的文字注释。
- ☒ **align**: 图形的对齐模式, 有三个模式可选 **left**, **center**, **right**。默认值为 **left**。
- ☒ **height**: 强制设定图形的显示高度。
- ☒ **width**: 强制设定图形的显示宽度。
- ☒ **vspace**: 在图形的上下添加几个空白字符, 如果手机的显示高度不足的话, 该设定将不生效。
- ☒ **hspace**: 在图形的左右添加几个空白字符, 如果手机的显示高度不足的话, 该设定将不生效。

9.4.5 锚

<anchor>标签的语法格式

```
<anchor title="label">
    task
    text
```

</anchor>

<a>标签的语法格式

```
<a title="label" href="url">
    task
    text
</a>
```

<a>标签的属性:

- ☒ title: 超链接显示的名称。
- ☒ href: 指定超链接的 URL。

9.4.6 事件

<do>标签的语法格式

```
<do type="type"
    label="label"
    name="name"
    optional="boolean">
    task
</do>
```

do 标签必须包括的属性只有一个 type, 另外还有三个可选的属性 label, name, optional。do 标签指定与卡片连接的菜单。菜单的显示情况根据浏览器的不同, 也各不相同。有些浏览器产生一个带有菜单和选项的连接将标签和内容连接起来, 而另一些浏览器则将用弹出框和按钮列表的形式显示菜单。

<do>标签的属性如下:

- ☒ type: 定义 do 标签的动作形式。这个属性被赋值时比较混乱, 它可能的属性值有 accept, delete, help, options, prev, reset, unkown 等。这些动作形式的含义如下:
 - ☐ accept: 接受 do 标签定义的动作。
 - ☐ delete: 删除目录或已经选定的选项。
 - ☐ help: 通过浏览器请求帮助。
 - ☐ options: 要求某些选择性的动作。
 - ☐ prev: 回到浏览器的上一层网页。
 - ☐ reset: 清楚所有输入或选定的项目。
 - ☐ unkown: 执行某些未被定义的动作。
- ☒ label: 定义了手机上显示的动作的提示字符串。但是有三种特殊的情况, 读者

有必要特别的注意一下：

- ❑ 在 `do` 标签内定义的动作如果是 `<prev/>` 的时候，无论在 `label` 内进行任何的设置都将被大部分手机浏览器忽略。而在手机上默认的显示“Back”选项。

- ❑ 在 `do` 标签内定义的动作如果是 `<noop/>` 的时候，`label` 内一切设置都不会显示在手机浏览器上。

- ❑ `label` 不被设置的时候，系统将默认显示为“OK”

- ☒ `optional`：定义该标签是否可以被浏览器忽略。属性值等于 `true` 时可以被忽略，等于 `false` 时不能被忽略。这个属性一般不太会去使用，因为设计者既然编写了代码，应该不会有故意想把它忽略的做法。

`<onevent>` 标签的语法格式

```
<onevent type="type">
  task
</onevent>
```

`<onevent>` 标签只有一个属性。

- ☒ `type`：内部事件的触发条件，当前浏览器状态满足触发条件时，浏览器就会触发这个条件下设置的 `Task`，内部事件总共有 4 种触发条件：

- ❑ `ontimer`

满足时钟设置的条件时，该条件成立。关于时钟设置问题，后面还有专门的说明。

- ❑ `onenterbackward`

通过 `Prev` 或其他外部命令返回到当前 `Card`，该条件成立。

- ❑ `onenterforward`

当浏览器通过链接进入当前 `Card` 时，该条件成立。

- ❑ `onpick`

在使用 `Option` 控件列表的时候，任何点击控件的行为都会触发本事件，包括选择和去掉选择。

9.4.7 任务

`<go>` 标签的语法格式

```
<go href="url"
  sendreferer="boolean"
  method="method"
  accept-charset="charset"
  content
</go>
```

`<go>` 标签用来使浏览器访问其他资源或将数据传送给网络服务器。`go` 标签的功能类似

于 HTML 中的 form 标签。它的属性分析如下：

- ☒ href: 可以写某一个完整的 URL, 如果在同一个 Deck 文件中, 可以使用“#CARD 的名称”来定位。
- ☒ method: 有两个可选的方法值 Get 和 Post。Get 方法可以直接在 URL 后面加一个问号再加“变量名=变量值”的方式向服务器传输数据, 而 Post 方法需要在 go 标签内添加 postfield 元素, 然后将需要向服务器传输的数据的变量数据写在其中。具体的语法如下:

```
<postfield name= "变量名" value= "变量值" />
```

accept-charset: 字符串内码设置元素, 默认值是 UTF-8。

<prev>标签的语法格式

```
<prev>
    content
</prev>
```

<prev>标签可以不写任何属性, 直接使用<prev/>回到浏览器的前一个浏览过的 Card。如果没有浏览过的 Card 浏览器仍然会显示 Back 按钮, 但是没有任何的效果。

<noop>标签的语法格式

```
<noop/>
```

noop 标签用来屏蔽 template 标签中的同名的设定。

<refresh>标签的语法格式

```
<refresh>
    content
</refresh>
```

refresh 标签用于刷新浏览器, 重新读取 Card 中的变量, 这一功能与 Web 页中的刷新功能完全相同。

9.4.8 定时

<timer>标签的语法格式

```
<timer name="变量名" value="value" />
```

timer 标签是一个计时器, 可以用来计算浏览者在 Card 上的停留时间。当浏览者在 Card 上的停留时间超过设定值时浏览器将获得一个 ontimer 事件。

它的属性如下。

- ☒ **name**: 计时器指定的变量名, 当 **name** 被赋值后, 计时器将被设定为指定的变量的值。时间的单位为十分之一秒。
- ☒ **value**: 直接设置计时器的时间, 单位也是十分之一秒。

9.4.9 变量

<setvar>标签的语法格式:

```
<setvar name="变量名" value="变量值" />
```

setvar 标签用变量赋值。等价与其他高级编程语言中的“变量名=变量值”。

9.5 WML 的实例分析

前面两节介绍了 WML 的语法规则和各种功能的标签。纯文字的描述只是个感性认识, 为了增加读者的理性认识, 这一节中将提供多个实例。

9.5.1 单页的含有图片的 WML

单页的 WML 文档太过于简单, 所以这里将图像的显示也放在一起介绍。这个 WML 文档只包含一个 Card, 在 Card 中显示了一张图片和一个链接。

程序清单 9.1:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM/DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <template>
    <do type="prev" label="back">
      <prev/> //提供了一个template级的Back按钮
    </do>
  </template>
  <card id="CoCo" title="Super POP Star">
    <p>
      <a href="http://wap.cocolee.com.cn/">CoCoLee WAP</a><br/>
       //图像logo.wbmp
    </p>
  </card>
</wml>
```

以上的这段代码在 Nokia WAP Toolkit 2.1 的 Blueprint 界面下的效果如图 9-14。

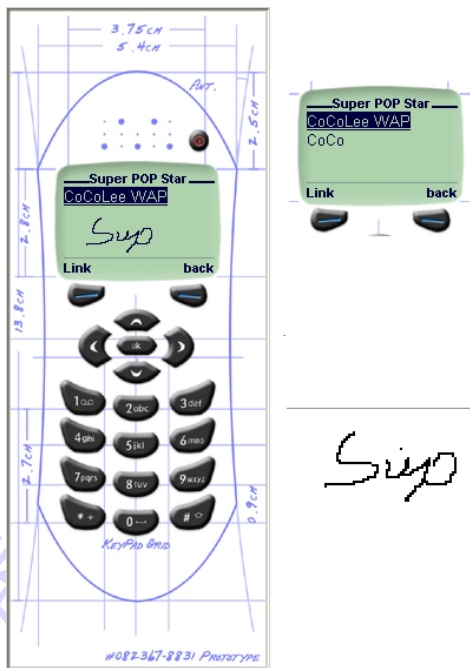


图 9-14 代码在 Blueprint 界面下的效果

关于 Nokia WAP Toolkit 2.1 的使用方法,在 9.2 节中已经详细介绍了。读者可以把 Nokia WAP Toolkit 当成一个 Internet Explorer 来看待。图 9-14 的左图给出了 Blueprint 界面的全貌,在后面的介绍里将不再给出全图,而只给出浏览的结果部分。右上图为 logo.wbmp 文件不存在时 Blueprint 的显示情况。右下图是作者手工编辑的 logo.wbmp 文件。

从图 9-14 中可以发现,在 wbmp 图不存在时,浏览器自动将标签中的 alt 属性值代替显示在 wbmp 图的出现位置。

在光标默认停留在 CoCoLee WAP 上,浏览器自动显示 Link 按钮,因为没有连接所指定的“http://wap.cocolee.com.cn/”,所以在点击浏览器左边的按钮以后 Blueprint 界面不会有任何的变化。

9.5.2 多页的 WML 的定位

本程序基于 9.1.wml,只是扩充 Card 的数量。

程序清单 9.2.wml

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <template>
    <do type="prev" label="back">
      <prev/>
    </do>
  </template>
```

```

<card id="CoCo" title="Super POP Star">
  <p>
    
    <a href="#card1">Next Card</a> 使用<a>跳转到下一个CARD
  </p>
</card>

<card id="card1" title="Super CoCo">
  <p>
    My favourite artist.
  </p>
  <do type="accept" label="go">
    <go href="#Card2"/> 使用<go>跳转到下一个CARD
  </do>
  <p>Do you want more?</p>
</card>
<card id="Card2" title="Favourite Song">
  <do type="prev" label="before">
    <noop/> 屏蔽了back按钮
  </do>
  <p>She has many songs.</p>
  <p>"A Love Before Time"</p>
  <p align="center">is the best hit.</p>
</card>
</wml>

```

图 9-15 是 Blueprint 界面所得到的预览效果图。依次的顺序是左上图 为 Card CoCo，右上图 为 Card1，下图 为 Card2。

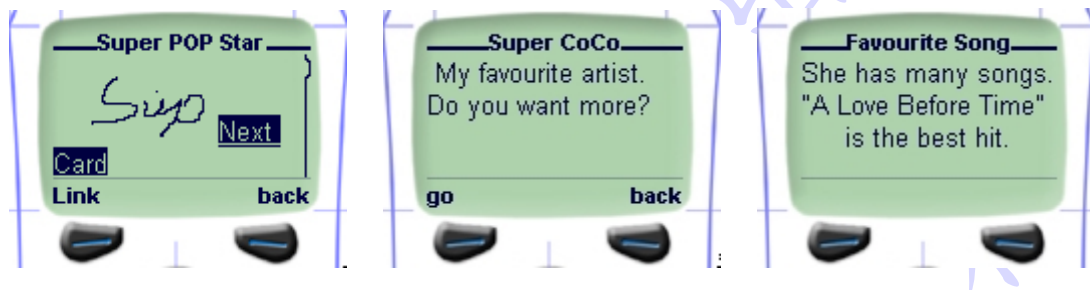


图 9-15 Blueprint 界面所得到的预览效果图

从 Card CoCo 的预览结果发现因为空间的不足，“Next Card”被浏览器自动的分隔在两行中显示。这是因为<p>标签并不能产生段落标记。如果将第 13 行的代码改为

```
<br/>
```

那么将会得到图 9-16 左图的效果。还应该注意，
标签不能在外部直接使用。一般

都嵌套在<p>等文本标签中使用，否则编译器将显示出错信息。

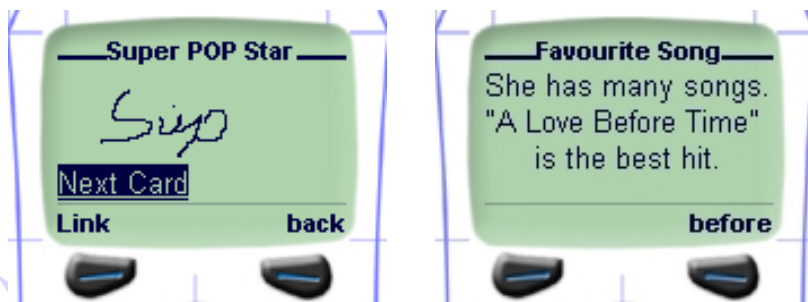


图 9-16 修改后的界面

第 29 行的 “<do type=“prev” label=“before”>” 屏蔽了 template 级的<prev/>标签。第 30 行的 “<noop/>” 使得浏览器不显示 back 按钮。如果将 “<noop/>” 更换为 “<prev/>”，浏览器的显示结果是图 9-16 的右图。因为更换后的<prev/>是 Card 级的，因此，浏览器显示的按钮名称为 “before”。

9.5.3 使用计时器

在这一小节，使用计时器标签编写一个简单的测试反应能力的小游戏。整个游戏程序由三个 Card 组成。第一个 Card 给浏览者一个准备游戏的时间，第二个 Card 给浏览者一秒钟的时间按下 Link 键，到达第三个 Card，也就是游戏的结束场景了。

程序清单 9.3.wml

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="first" ontimer="#second">
<timer value="50"/>
<p>This is a five-second rest.</p>
</card>

<card id="second">
<onevent type="ontimer">
<go href="#first"/>
</onevent>
<timer value="10"/>
<p>You have only one second or you can go to the third CARD.</p>
<p>
<a href="#third">CLICK HERE!</a>
</p>
</card>

<card id="third" >
```

```

<p>You get the point.</p>
<p>The End:</p>
</card>
</wml>

```

图 9-17 是 Blueprint 界面所得到的预览效果图。依次的顺序是左上图为 first，右上图为 second，下图为 third。



图 9-17 Blueprint 界面所得到的预览效果图

看了效果图以后，分析一下这个程序。程序中有两个等待的时间。

Card first 中的代码如下：

```

<card id="first" ontimer="#second">
  <timer value="50"/>

```

而 Card second 中使用了另一种计时器的调用方法，代码如下：

```

<onevent type="ontimer">
  <go href="#first"/>
</onevent>
<timer value="10"/>

```

前一段代码直接调用 ontimer 计时器编写计时程序，ontimer="#second"与后面的<timer value="50"/>决定了跳转到 Card second 的等待时间。Value 后面的赋值是以 0.1 秒为单位的。50 也就相当于 5 秒钟。

请仔细阅读后一段代码，value="10"也就意味着在一秒钟后，onevent 事件被激活。激活后将执行<go href="#first"/>，并且跳转到 Card first。这是一种使用 onevent 标签的 ontimer 触发类型达到计时器目的的编写方法。

程序 9.3 因为没有使用 template 标签定义 do 事件，因此每一个 Card 的显示界面中都没有类似“go”和“back”的按钮。这种情况下编辑出来的 wml 页，因为没有彼此的联系所以并没有多大的使用价值。

9.5.4 赋值与数据交换

前几个小节中都是介绍单机的 WML 文档的编写和显示工作。但在现实生活中，手机

往往需要与服务器有数据交换，例如使用手机订购机票。这就是本小节将要论述的内容：WML 的赋值与数据交换。

程序 9.4.wml 和程序 9.5.asp 构成了一个客户端到服务器的 WML 数据传输范例。在程序 9.4.wml 中共有两个数据变量需要传回服务器（arv1，arv2），为了程序的简练，预先设置他们的值为 1 和 2。服务器端的程序 9.5.asp 将 arv1 和 arv2 相加然后将结果传回客户端。

以下是两个程序的程序清单。

程序清单 9.4.wml

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.com/DTD/wml_1.1.xml">
<wml>
<template>
  <onevent type="onenterforward">
    <refresh>
      <setvar name="MyASP" value="countsev.asp"/>
      //定义变量MyASP
    </refresh>
  </onevent>
</template>

<card id="Start" title="Count Page">
<do type="accept">
  <go href="$MyASP" method="post">
    //使用变量MyASP
    <postfield name="arv1" value="1"/>
    <postfield name="arv2" value="2"/>
    //两个变量的递交
  </go>
</do>
<p>Transfer Data...</p>
</card>
</wml>
```

以上程序的浏览结果如图 9-18。



图 9-18 客户端的程序结果图

在文档的 `template` 部分使用了代码 `<setvar name="MyASP" value="countsev.asp"/>` 定义一个变量 `MyASP` 用于指定服务器的 ASP 文件的地址。因为变量的值为 `countsev.asp`，并没有指定路径，所以文件 `countsev.asp` 必须放在默认的 WAP 服务器的根目录下。而客户端程序 `9.4.wml` 可以放置在磁盘的任何位置。调用 `MyASP` 的语句在 `Card` 标签内部，代码是 `<go href="$MyASP" method="post">`，应该注意到的是在调用变量时必须在变量名前必须加上 `$` 符号表示使用的是变量的值。

程序中的以下两行代码是将变量的数据传回到服务器的。

```
<postfield name="arv1" value="1"/>
<postfield name="arv2" value="2"/>
```

这两行代码使用了 `postfield` 标签。`postfield` 标签的使用方法参见本书的 9.3.7 的 `go` 标签部分。

当手机浏览者按下 `ok` 键后，数据 `arv1` 和 `arv2` 开始传输到服务器，由程序 `9.5.asp` 处理。服务器程序的代码如下：

程序清单 9.5.asp

```
<script language="vbscript" Runat="Server">
<% Response.ContentType="text/vnd.wap.wml" %>

Response.write("<?xml version='1.0'?>")
Response.write("<!DOCTYPE wml PUBLIC '-//WAPFORUM//DTD WML 1.1//EN'"
"http://www.wapforum.com/DTD/wml_1.1.xml">")
Response.write("<wml>")
Response.write("<card id='Result' title='Result Page'>")
Response.write("<p>")
Response.write("The result is "</p>")
Response.write("<p>")
Response.write("</card>")
Response.write("</wml>")
</script>
```


服务器端程序的运行效果如图 9-19。



图 9-19 服务器端程序的运行效果

服务器使用 VBScript 编写。大部分的代码是用来将 WML 文档结构传送到客户端。计算部分使用的语句是`<%arv1%>+<%arv2%>`，非常的简单。但是也可以看出这种 ASP 程序将会把大量的数据传输浪费在构建 WML 文档结构上，这对于网络带宽来说是一种相当大的浪费。但是就目前来说，并没有合适的解决对策。

服务器端的运行需要一个 WML Server，在 9.1 节中已经介绍了自己如何改装 WML Server。请注意代码`<% Response.ContentType="text/vnd.wap.wml" %>`，如果 WML Server 没有正确的设置，那么手机是无法正确现实结果的。

9.6 小 结

无线通信协议在将来的应用会越来越广泛和深入生活。随着手机与 PDA 的普及，台式机笨重，不利于搬动等特点就越来越明显了。虽然台式机的处理能力能大大的超越手机和 PDA，但是日常生活中的大部分工作都不需要很大的信息处理量，并且随着生活节奏的加快，人们不可能总是固定在同一个地方，更不可能随时都有一个台式机来操作。台式机和手机就像固定电话与移动电话一样，如果有一天，移动电话的功能、成本和收费能与固定电话一样，那么固定电话就再也没有存在的必要了。

WML 文档的结构是比较简单的，但是因为它所服务的对象是手机而不是 PC，所以在编写 WML 文档的时候要尽可能的标准。在调试时首先使用 PC 手机模拟器，当发现显示结果与预期效果有出入的时候，先检查代码的正确性，然后再尝试使用手机直接接收 WML 页。

WML 为无线通信提供了介质。通过编辑 WML 可以在手机上如上网般浏览各类网站。可以说 WML 就像台式机所浏览的 HTML 一样，将会有异常飞速的发展。

第 10 章 SMIL(同步多媒体集成语言)

本章导读

同步多媒体合成语言 SMIL, 即 Synchronized Multimedia Integration Language, 它是一种用于描述多媒体演示文档的语言, 允许将一批独立的多媒体对象, 包括视频、音频、图像、文字等等, 在时间和空间轴上集成为一个同步的多媒体演示文档。这个规范继承了 XML 跨平台的优点, 而且将所有资源都以 URI 的形式存在网络中, 无须编译即可使用, 所以它推出后立刻得到了众多厂商的支持。

本章就对 XML 语言的另一个高级应用 SMIL 同步多媒体集成语言技术做了一个详细介绍。

10.1 SMIL 简介

互联网络发展至今, 实现 Web 上动态交互功能的页面效果的技术日趋增多。从静态的 HTML 发展到动态的 HTML(DHTML), 从 CGI 编程到 JAVA 虚拟机的实现。Web 呈现给我们的已经不再是静止不动的文本信息, 而是一个生动活泼的现实世界。WWW 虽然一向擅长于多媒体的应用, 可是在动态的多媒体方面一直存在带宽限制及资料同步等问题。多媒体网页制作新技术 SMIL 正是在这种情况下应运而生。

SMIL 是英文 Synchronous Multimedia Interchange Language 的缩写, 中文译为同步多媒体集成语言。它是基于 XML 的一个应用, 将静、动态媒体合为一体的标志语言。XML 是 SGML 的子集, 它继承了 SGML 的通用性, 去掉了 SGML 的复杂性, 承担着 HTML (面向静态媒体的置标语言) 不堪重负的使命。SMIL 则采用 RTSP 协议 (Real Time Streaming Protocol, 实时流传输协议), 用类似原有 HTML 的标记方式对 Web 上多媒体展示的时间和布局控制进行了规定, 安排了网页上的影像、声音及文字各部分的时间流程, 从而开创了 Web 上多媒体的创作和展示的新阶段。

1998 年 6 月 15 日, 环球网协会 (W3C) 将同步多媒体集成语言 SMIL 作为 W3C 推荐文本正式发布, 其最强有力的支持来自于 Real Networks 公司。1999 年 8 月 3 日, 在第一个草案的基础上, W3C 推出了 SMIL Boston 版本。2000 年 6 月 22 日, SMIL Boston 最新版本推出。目前 SMIL Boston 有了许多重要的扩展, 包括可重复使用的模块、通用的动画设计、改良的交互功能以及电视综合功能等。

SMIL 语言能够处理的多媒体信息包括文字、图像、声音、数据等, 利用 SMIL 制作多媒体网页的关键是如何实现这些信息的同步。其中, Realtext 是纯文本信息的存储格式, 文件扩展名为 .rt; RealPix 是图像信息的存储格式, 文件扩展名为 .rp; Realaudio 是音频信息的存储格式, 文件扩展名为 .ra; Realvideo 是视频信息的存储格式, 它的文件扩展名为 .rv; RealMedia 的文件扩展名为 .rm; RealSmil 格式的文件扩展名为 .smi。其中 .rt 文件、.rp 文件和 .smi 文件均可以用任何一种文本编辑器, 或可扩展标记语言的专用编辑器来编辑。而音频视频信息则需要用 RealEncoder 工具来编辑。

要使用 smil, 需要把内容都用标记<smil>包含起来, 它的整体形式如下所示:

```
<smil>
  <head>
    <!--屏幕布局描述-->
  </head>
  <body>
    <!--时间行为描述-->
  </body>
</smil>
```

先来看 smil 屏幕布局的形式, 使用的元素包括 layout, root-layout, region 等。layout 用于定义窗口布局, 其子元素 root-layout 用于描述主浏览窗口, 而 region 用来把窗口分为若干个区域, 确定各个区域的位置、背景等等。比如下面的形式定义了一个主窗口, 并在其中定义了两个子窗口。

```
<layout>
<root-layout background-color="white" height="300" width="450"/>
  <region id="scene" top="0" left="0" height="300" width="350"/>
  <region id="comment" top="20" left="370" height="260" width="430">
</layout>
```

对于演示文档时间行为的描述, 可以使用 pal 标记, seq 标记等来描述, pal 用于把包含动画(animation), 视频(video), 音频(audio), 文本流(textstream), 文本(text), 图片(image) 组织成并发形式, 而 seq 属性则把它们组织成串行的形式, 还有一些可以使用的属性 begin, end, dur, endsync, repeat 等用来具体规定它的起止时间。看下面的形式:

```
<par>
<seq id="places">
  
  
  
</seq>
<textstream id="commenttext" region="comment" src="comment.rt" endsync=id(places)/>
<audio id="bgmusic" src="audio/background.rm" repeat="15"/>
</par>
```

上面的代码的含义是: 在窗口左边的区域顺次播放三幅图片, 第一幅图片播放 15 秒, 播放完毕后等待 10 秒后播放第二幅, 第一幅图片播放完毕后的第 25 秒停止第二幅图片的播放, 第 35 秒后开始播放第三幅图片, 播放 15 秒。在播放图片的同时, 右边的区域播放解说文字, 直到左边的图片播放完毕后才停止播放。除此之外, 还有背景音乐, 将这段音乐播放 15 次。

smil 在描述它的资源链接时使用的也是<a>属性，用 show 属性来表达超链接激活时浏览器的状况，而<anchor>标记用于使同一个对象的不同区域指向不同的链接，或者在不同的时间指向不同的链接。比如：

```

<anchor href="paris1" begin="0s" end="5s" show="new">
<anchor href="paris2" begin="5s" end="15s" show="new">
</img>
```

最后在合成一个完整的文档的时候，由于 smil 本身还是一个 xml 文档，因此在前面还要进行 xml 声明，同时需要引入名域声明，加上这两句话：

```
<?xml version="1.0" encoding="GB2312"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL1.0//EN"
"http://www.w3.org/TR/REC-smil/SMIL10.dtd">
```

10.2 SIML 的元素

10.2.1 文件头元素

smil 元素

smil 元素是根元素，它有一个属性 id，这个属性在一个文件中唯一地标识一个元素。该元素值是一个 XML 标识符。

在 smil 元素中可以包含 body 和 head 这两个子元素。

head 元素

head 元素包含与演示的时序行为无关的信息，它有一个属性 id，这个属性在一个文件中唯一地标识一个元素。该元素值是一个 XML 标识符。

在 head 元素中可以包含任意多个 meta 元素，并可以包含一个 layout 元素或一个 switch 元素。

layout 元素

layout 元素规定了文件体中的元素是如何在一个视频的或音频的抽象表示平面上定位的。如果一个文件中没有 layout 元素，则文件体中元素的定位与实现相关。

可以通过 switch 元素将多个 layout 元素封装在一个 SMIL 文件中来达到包含多个可选布局的效果。这样我们就可以用于使用不同的布局描述语言来描述文件的布局。

下面的例子说明了如何用 CSS 代替 SMIL 基本布局语言：

```
<smil>
<head>
<switch>
```

```

<layout type="text/css">
    [region="r">{ top: 20px; left: 20px }
</layout>
<layout>
<region id="r" top="20" left="20" />
</layout>
</switch>
</head>
<body>
<seq>

</seq>
</body>
</smil>

```

layout 元素有两个属性 id 和 type。

id 属性在一个文件中唯一地标识一个元素。该元素值是一个 XML 标识符。

type 属性定义了 layout 元素中使用的布局语言。如果播放器不能理解这种语言，就跳过标签<layout>和</layout>之间的所有内容。type 属性的缺省值是“text/smil-basic-layout”。

如果 layout 元素的 type 属性值为缺省值“text/smil-basic-layout”，它可以包含 region 和 root-layout 这两个元素。

如果 layout 元素的 type 属性为其他值，则在 layout 元素中包含字符数据。

meta 元素

meta 元素用来定义文件的属性(如，作者，截止日期，关键字表，等等)并给这些属性赋值。每一个 meta 元素规定一个属性/属性值对。

meta 元素的主要属性有：

- ☒ content 属性规定 meta 元素中定义的属性的属性值，是不可缺少的。
- ☒ id 属性在一个文件中唯一地标识一个元素。该元素值是一个 XML 标识符。
- ☒ name 属性标识了 meta 元素中定义的属性，同样该属性也是 meta 元素的不可缺少的属性。
- ☒ skip-content 属性的引入是为从 SMIL 以后的可扩展性考虑的。它在下述两种情况下被解释：

(1) 如果在 SMIL 的未来版本中引入了一个新元素，而此元素能以 SMIL 1.0 元素作为其元素内容，“skip-content”属性控制一个 SMIL 1.0 播放器是否处理其内容。

(2) 如果 SMIL 版本 1.0 中的一个空元素在未来的 SMIL 版本中不再为空，skip-content 控制一个 SMIL 1.0 播放器是否忽略其内容，或是导致一个语法错误。

如果“skip-content”属性的值是“true”，且以上的两种情况之一适用，那么元素内容被忽略。如果值为“false”，元素内容被处理。“skip-content”的缺省值是“true”。

meta 元素是一个空元素，不包含任何内容或者是子元素。

10.2.2 基本布局元素

下面介绍 SMIL 的基本布局元素。SMIL 基本布局与 CSS2 中定义的视觉表示模型相一致，它重用了 CSS2 规范中定义的格式化属性，同时引入了新的 `fit` 属性。我们需要注意的是 SMIL 基本布局只控制媒体对象元素的布局，如果要用于其他 SMIL 元素则是非法的。

SMIL 基本布局的类型标识符是 “text/smil-basic-layout”。

下面的样式表(styleshet)定义了 SMIL 基本布局中有效的 CSS2 属性 “display” 和 “position” 的取值。这些属性值是固定的：

```
a{display:block}
anchor{display:block}
animation{display: block; position: absolute}
body{display: block}
head{display: none}
img{display: block; position: absolute}
layout{display: none}
meta{display: none}
par{display: block}
region{display: none}
ref {display: block; position: absolute}
root-layout{display: none}
seq{display: block}
smil{display: block}
switch{display:block}
text{display: block; position: absolute}
textstream{display: block; position: absolute}
video{display: block; position: absolute}
```

注意：作为这些定义的结果之一，所有以绝对方式定位的元素(animation, img, ref, text, textstream 和 video)被包含在以根元素(smil)内容边界定义的单个包含块中。

缺省值

SMIL 基本布局定义了所有与布局相关的属性的缺省值。它们与 CCS2 中相应属性的初始值一致。

如果程序员想给文件中的所有媒体对象元素选择缺省的布局值，文件中必须包含一个类型为 “text/smil-basic-layout” 的空 layout 元素，如：

```
<layout type="text/smil-basic-layout"></layout>
```

region 元素

region 元素是一个空元素，它控制媒体对象元素的位置，大小和缩放。

在下面作为例子的片断中，一个 text 元素被定位于距离显示窗口上边界 5 个象素远的位置：


```

<smil>
<head>
  <layout>
    <region id="a" top="5" />
  </layout>
</head>
<body>
  <text region="a" src="text.html" dur="10s" />
</body>
</smil>

```

region 元素可以有如下属性:

- ☒ background-color: 除了 SMIL 基本布局不要求支持“系统颜色”外, background-color 属性的定义和使用与 CSS2 规范中的“background-color”属性相同。如果没有 background-color 属性, 则采用透明的背景。
- ☒ fit : 属性规定了可视媒体对象的真正高度和宽度, 我们要注意它不同于 region 元素中 height 和 width 属性所规定的值时的行为。它可以取下列值:

(1) fill

独立地缩放对象的高度和宽度, 使之正好与方框的各边相接。

(2) hidden

如果媒体对象元素的真正高度(宽度)小于“region”元素中定义的高度(宽度), 从上边界(左边界)开始显示, 余下的高度(宽度)用背景色填充。

如果媒体对象元素的真正高度(宽度)大于“region”元素中定义的高度(宽度), 从上边界(左边界)开始显示, 直到“region”元素中定义的值为止, 切去在此高度(宽度)之下(之右)的对象的部分。

(3) meet

保持长宽比缩放可视媒体对象, 直到它的高度或宽度等于规定的高度或宽度, 不切去其内容。对象的左上角置于方框的左上角, 左边或下边的空余空间用背景色填充。

(4) scroll

当元素要显示的内容超出边界时, 提供滚动机制。

(5) slice

保持长宽比缩放可视媒体对象, 直到它的高度或宽度等于规定的高度或宽度, 一部分内容可能会被切去。取决于实际情况, 可以显示可视媒体对象的水平或垂直的一部分。切去媒体对象右边超出的宽度。切去媒体对象底部超出的高度。

fit 属性的缺省值是“hidden”。

☒ height

此属性的定义和使用与 CSS2 规范中的“height”属性相同。属性值可以是“百分数”值, 也可以是 CSS2 中“length”值的一种变化形式。对于“length”值, SMIL 基本布局仅支持以 CSS2 中定义的像素为单位。

☒ id

此属性通过将有位置概念的元素 `region` 属性设为一个 `region` 元素的 `id` 值, 可以将此 `region` 元素作用于此有位置概念的元素。“`id`”属性是 `region` 元素的必需属性。

☒ left

此属性的定义和使用与 CSS2 规范中的“`left`”属性相同。属性值有与“`height`”属性相同的限制。它的缺省值是 0。

☒ skip-content

这个属性在介绍 `meat` 的属性时已经详细的描述过, 不再重复说明。

☒ title

此属性提供了其所在元素的参考信息。`title` 属性的值可以被客户代理以各种不同的方式解释。例如, 可视浏览器经常将其作为“工具提示”(当定点设备停在一个对象之上时出现的一个简短的消息)。

☒ top

此属性的定义和使用与 CSS2 规范中的“`top`”属性相同。属性值有与“`height`”属性相同的限制。它的缺省值是 0。

☒ width

此属性的定义和使用与 CSS2 规范中的“`width`”属性相同。属性值有与“`height`”属性相同的限制。

☒ z-index

此属性的定义和使用与 CSS2 规范中的“`z-index`”属性相同, 下列情况例外:

- (1) 如果元素 A 和元素 B 产生的两个方框有相同的堆栈层次。
- (2) 如果元素 A 的显示迟于元素 B 的显示, A 的方框迭加于 B 的方框之上。
- (3) 如果元素的显示在同一时间开始, 而元素 A 在 SMIL 文件文本中迟于元素 B 出现, A 的方框迭加于 B 的方框之上。

`region` 也是一个空元素。

root-layout 元素

`root-layout` 元素是一个空元素, 它决定根元素的布局属性值, 根元素的这些值又决定了视窗的大小。

`root-layout` 可以有如下属性:

☒ background-color

除了 SMIL 基本布局不要求支持“系统颜色”外, `background-color` 属性的定义和使用与 CSS2 规范中的“`background-color`”属性相同。如果没有 `background-color` 属性, 则采用透明的背景。

☒ height

此属性的定义和使用与 CSS2 规范中的“`height`”属性相同。属性值可以是“百分数”值, 也可以是 CSS2 中“`length`”值的一种变化形式。对于“`length`”值, SMIL 基本布局仅支持以 CSS2 中定义的象素为单位, 它设定了根元素的高度。只允许表示长度的值。

☒ id

在一个文件中唯一地标识一个元素。该元素值是一个 XML 标识符。

☒ skip-content

这个属性在介绍 meat 的属性时已经详细的描述过，不再重复说明。

☒ title

此属性提供了其所在元素的参考信息。title 属性的值可以被用户代理(user agent)以各种不同的方式解释。

☒ width

此属性的定义和使用与 CSS2 规范中的“width”属性相同。属性值有与“height”属性相同的限制。它设定了根元素的宽度。只允许表示长度的值。

10.2.3 文件体元素

body 元素

body 元素包含与此文件的时序和链接行为有关的信息。它隐含地定义了一个 seq 元素，body 元素有一个属性 id，在一个文件中唯一地标识一个元素。该元素值是一个 XML 标识符。

在 body 中可以包含下列 a, animation, audio, img, par, ref, seq, switch, text, textstream, video 等元素。

par 元素

par 元素是一个同步元素，它的子元素可以在时间上重叠。par 中子元素的文本次序对它们演示的次序没有对应的意义。

下面介绍 par 元素的一些常用的属性。

☒ abstract

对元素所包含内容做简短描述。

☒ author

元素内容的作者。

☒ begin

此属性规定了元素显式开始的时间。

此属性可以包含下列两类值：

1. 延时值

延时值是用于计量演示时间的时钟值。演示时间以演示播放的速度增加。它的行为如同录像播放设备上计数器的时间数码。它可以被用户或播放器停止，减少或增加。

延时值的语义取决于此元素的第一个为同步元素(即当祖元素中的“a”元素和“switch”元素被忽略)的祖元素：

如果此祖元素也是一个 par 元素，此属性值定义了从元素的实际开始起的一个延时，我们来看一个例子来说明。

<par>

<audio id="myaudio" begin="10s" src="audio" />

</par>

下面用图 10-1 来说明它的延时效果。

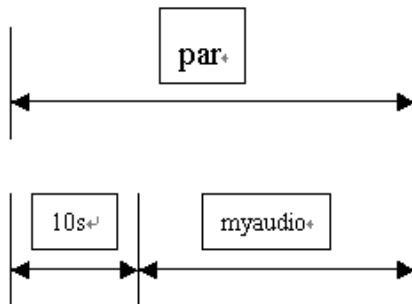


图 10-1 par 元素延时例子一

如果此祖元素是一个 seq 元素(也是一个同步元素,下面将作介绍),此属性值定义了从第一个为同步元素的词法前趋的实际结束起的一个延时。看下面的例子。

<seq>

<audio id="1" src="audio1" />

<audio id="2" begin="10s" src="audio2" />

</seq>

下面用图 10-2 来说明它的延时效果。

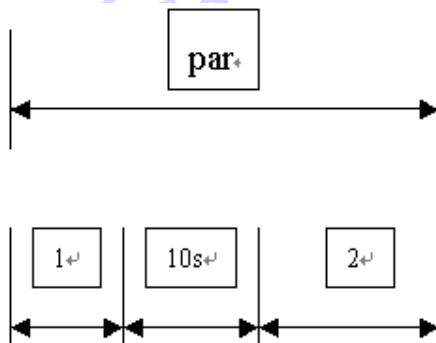


图 10-2 par 元素延时例子二

2. 事件值

当某些特定事件发生时此元素开始。属性的值是一个元素事件。产生事件的元素必须在作用域内。“在作用域内”的元素的集合 S 如下决定:

将此元素的第一个为同步元素的祖元素的所有子元素加入 S 中。

从 S 中去掉所有“a”元素和“switch”元素。将所有“a”元素的子元素加入 S 中,子元素是“switch”的情况例外。得到的集合 S 即“在作用域内”的元素的集合。

再来看一个例子。

<par>

<audio id="a" begin="10s" src="audio" />

</par>

用图 10-3 来说明它的延时效果。

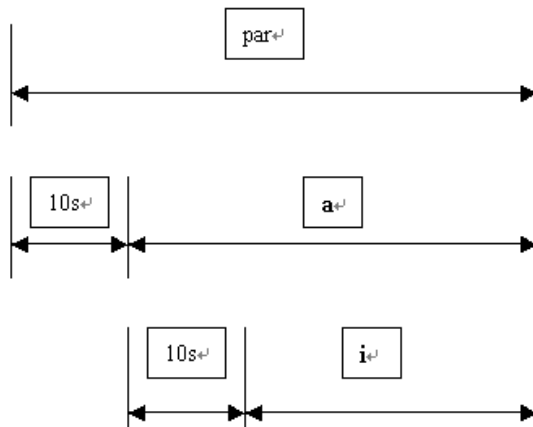


图 10-3 par 元素延时例子三

☒ copyright

元素所包含内容的版权声明。

☒ dur

此属性规定了元素的显式时长，此属性可以是一个时钟值或字符串“indefinite”。

☒ end

此属性规定了元素的显式结束，此属性可以包含与“begin”属性相同的属性值类型。

☒ id

此属性在一个文件中唯一地标识一个元素。

☒ repeat

此属性定义媒体对象重复的次数。缺省值是 1。

☒ system-bitrate

此属性定义连接速度。它以位/秒为单位规定了系统可用的近似带宽。带宽的度量方法与应用相关，这意味着应用可以使用端到端连接的复杂度量方法，也可以使用由用户控制的简单静态设置。例如，第二种情况可用于根据用户与网络的连接状况作出选择。使用调制解调器的用户的典型值会是 14400, 28800 和 56000 等。如果可用的系统比特率大于或等于给出的值，结果为“true”。如果可用的系统比特率小于给出的值，结果为“false”。这个属性可以取任何大于 0 的整数值。如果其值超出了实现定义的最大带宽值，那么它的属性值总是为“false”。

☒ system - captions

此属性定义是否使用音轨的基于文本的等价物。它允许创作者区分对应于演示的音频部分的额外文字(面向有听力障碍或想学习阅读而想要或需要这些信息的观众)和面向大部分观众的文字。如果用户指明希望看到闭路字幕信息，其值为“on”。如果用户指明不希望看到这些信息，其值为“off”。

☒ system-language

此属性定义表现的语言。它是一个以逗号分隔的语言名的列表，此属性是一个以逗号分隔的语言名的列表。

☒ system-overdub-or-caption

此属性规定了文件使用子标题还是配音对话。它可以取值“caption”和“overdub”。

☒ system-required

此属性指定了一个扩展的名字。如果具体实现支持此扩展，结果为“true”，否则结果为“false”。

☒ system-screen-size

此属性定义了特定的屏幕大小。

☒ system-screen-depth

此属性定义了表现的像素深度。

☒ region

此属性指定了在文件布局部分定义一个抽象表示面(视频的或音频的)。它的值必须是一个 XML 标识符。如果在布局部分没有定义表示面，此元素的格式化属性的值由缺省布局决定。

seq 元素

seq 也是一个同步元素。该元素的子元素构成一个时间序列。

它所包含的属性与 par 元素的属性基本一致，我们就不再做介绍了。

在对 par 和 seq 元素有了一个整体的了解之后我们再对子元素同步做一个介绍。

在一个同步组中，子元素的同步精确度与实现相关。比如当 par 元素包含两种或多种连续媒体类型如视频或音频时，其中的一个会出现时延。

播放器可以有两种同步行为：

(1) 硬同步

播放器将 par 元素的所有子元素与一个共同时钟同步。

(2) 软同步

par 元素的每个子元素有自己的时钟，独立于 par 元素中其他子元素的时钟运行时钟值。

先来看一下时钟值的表示方法：

Clock-val::= Full-clock-val | Partial-clock-val | Timecount-val

Full-clock-val::= Hours ":" Minutes ":" Seconds ("." Fraction) ?

Partial-clock-val::= Minutes ":" Seconds ("." Fraction) ?

Timecount-val::= Timecount ("." Fraction) ?

("h" | "min" | "s" | "ms") ? ; 缺省为 “s”

Hours::= 2DIGIT; 任何正整数

Minutes::= 2DIGIT; 从00到59

Seconds::= 2DIGIT; 从00到59

Fraction::= DIGIT+

Timecount::= DIGIT+

2DIGIT::= DIGIT DIGIT

DIGIT::= [0-9]

下面是一些合法的时钟值的例子：

完整的时钟值：12:30:33 = 12 小时 30 分 33 秒

部分的时钟值：12:33 = 12 分 33 秒

计时值：7h = 7 小时

45min = 45 分

30s = 30 秒

5ms = 5 毫秒

有 n 位的一个小数表示下列值：

$x * 1/10^{**n}$

比如：

00.5s = $5 * 1/10$ 秒 = 500 毫秒

00:00.008 = $8 * 1/1000$ 秒 = 8 毫秒

下面介绍元素事件值

元素事件值指出了同步元素中的特定事件。

元素事件有下列语法：

Element-event::= "id(" Event-source ")(" Event ")"

Event-source::= Id-value

Event::= "begin" | Clock-val | "end"

在规范中定义了下列事件：

1.begin

在元素的实际开始处产生此事件。

比如：begin="id(x)(begin)"

2.clock-val

当与某元素相关联的时钟到达某个特定值时产生此事件。这个时钟在元素实际开始时从 0 开始。对于 par 和 seq 元素，此时钟给出了从元素实际开始计起的演示时间。对于媒体对象元素，它的语义与实现相关。可以给出从元素实际开始时计起的演示时间，也可以给出对象的媒体时间。后者可因显示时延或网络时延等原因不同于对象开始显示后所经过的演示时间。

比如：begin="id(x)(10s)"

3.end

在元素的实际结束处产生此事件。

比如：begin="id(x)(end)"

媒体对象元素：ref, animation, audio, img, video, text 和 textstream 元素

媒体对象元素允许在 SMIL 演示中包括媒体对象。有两类媒体对象：有视频、音频等时长概念的媒体对象和文本、图像等没有时长概念的媒体对象。

播放器在回放媒体对象时，不是从媒体对象元素的名字推断其实际类型。它必须完全依赖于其他有关类型的信息，如 `type` 属性中包含的类型信息或由服务器或操作系统给出的类型信息。

当然我们在实际应用中从可读性出发应该确保元素名字反映出了媒体对象所属的类型是 `animation`, `audio`, `img`, `video`, `text` 还是 `textstream`。当我们不确定该媒体应属于哪一类型时，我们可以使用通用的“`ref`”元素。

媒体对象元素可以有如下属性：

☒ `abstract`

元素所包含内容的简短描述，提供附加信息，或者内容的概述。

☒ `alt`

对于不能显示一个特定媒体对象的用户代理，此属性指明了替代文本。我们建议所有的媒体对象元素都有“`alt`”属性，且其属性值是一个有意义的描述。

☒ `author`

该属性定义了内容的整理者。

☒ `begin`

决定媒体对象在它的父包容器中何时出现。

☒ `clip-begin`

这个属性规定了连续媒体对象的子片断的开始，偏移量从媒体对象的头部计起。

此属性的值包括一个计量单位说明符和其后的时间值，时间值的语法和语义取决于计量单位说明符。允许下列格式：

SMPTE 时戳

SMPTE 时间码[SMPTE]可以用于得到帧一级的存取精确度。计量单位说明符可以是下列值：

`smpte`

`smpte-30-drop`

这两个值指明使用每秒 29.97 帧的“SMPTE 30 drop”格式。时间值的“`frames`”域可以取值 0 到 29。对每秒 30 帧和每秒 29.97 帧的差别的处理是跳掉每分钟的头两帧(索引为 00 和 01)，为 10 整数倍的分钟除外。

`smpte-25`

时间值的“`frames`”域可以取值 0 到 24。

时间值的格式为时:分:秒:帧:子帧。如果帧的值为 0，可以被省略。子帧以帧的百分之一为单位。

例子：

`clip-begin="smpte=10:12:33:20"`

正常播放时间

正常播放时间用 SMIL 时钟值表示时间。计量单位说明符为“npt”，时间值的语法与 SMIL 时钟值的语法相同。

例子：

```
clip-begin="npt=123.45s"  
clip-begin="npt=12:05:35.3"
```

☒ clip-end

clip-end 属性规定了应播放的连续媒体对象(如声频, 视频或其他演示)的子片断的结束。它使用与 clip-begin 属性值相同的语法。

如果“clip-end”属性的值超过了媒体对象的时长, 它的值被忽略, 片断的结束被设为媒体对象的实际结束。

☒ copyright

该属性定义了文档的版权信息。

☒ dur

此属性规定了元素的显式时长, 此属性可以是一个时钟值或字符串“indefinite”。

☒ end

此属性规定了元素的显式结束, 此属性可以包含与“begin”属性相同的属性值类型。

☒ id

此属性在一个文件中唯一地标识一个元素。

☒ longdesc

此属性指出了指向媒体对象详细描述的一个链接(URI)。此描述应补充说明 alt 属性提供的简短说明。当媒体对象被关联为链接终点时, 此属性应提供与此链接终点相关的信息。

☒ region

此属性指定了在文件布局部分定义一个抽象表示面(视频的或音频的)。

☒ src

此属性的值是媒体对象的 URI。

☒ title

此属性提供了其所在元素的参考信息, 我们建议所有的媒体对象元素都有“title”属性, 且其属性值是一个有意义的描述。

☒ type

用“src”属性引用的媒体对象的 MIME 类型。

switch 元素

switch 元素允许我们规定一个可选的元素集合, 但却只能从中选取一个可接受的元素。当一个元素是一个 SMIL 1.0 元素, 它的媒体类型可以被解码, 并且它所检测的所有属性都得出值“true”时, 称此元素是可接受的。播放器依它们在 switch 元素中出现的次序逐个检测这些元素, 选择第一个可接受的元素, 其余元素被摒弃。

因此，我们应该按期望值从高到低排序这些可选元素，而且还应该在 switch 元素的最后放置一个相对保险的元素以保证<switch>中至少有一个元素会被选中。

switch 元素可以有 id、title 属性。

如果 switch 元素被直接或间接地用作 body 元素的子元素，它可以包含 a, animation, audio, img, par, ref, seq, switch, text, textstream, video 等子元素任意多次。

如果 switch 元素被用于 head 元素中，它可以包含多个 layout 元素。

下面来看两个例子来说明如何用 switch 元素来检测属性。

```
<par>
  <text .../>
  <switch>
    <par system-bitrate="40000">
      .....
    </par>
    <par system-bitrate="24000">
      .....
    </par>
    <par system-bitrate="10000">
      .....
    </par>
  </switch>
</par>
```

在这段代码中，媒体播放器每次查看一个选项，根据媒体播放器和媒体服务器间连接的已知特性查找一个可接受的比特率。

```
<par>
  <text .../>
  <switch>
    <par system-screen-size="1280x1024" system-screen-depth="16">
      .....
    </par>
    <par system-screen-size="640x480" system-screen-depth="32">
      .....
    </par>
  </switch>
</par>
```

在这段代码中播放器根据屏幕的特性来选择选项之一。

10.2.4 超链接元素

超链接元素使得描述对象间的链接访问关系成为可能。SMIL 只提供内嵌的链接元素。

链接仅限于单向单端链接，也就是说所有的链接有且仅有一个源和一个终点。SMIL 中的所有链接由用户激活。SMIL 浏览器可能会使用一个 HTML 插件去显示内嵌的 HTML 页。相应的，HTML 浏览器可能会用一个 SMIL 插件显示 HTML 页中内嵌的 SMIL 文件。

在这样的演示中，可以在文件不同的层次定义链接，有可能会产生冲突。在这种情况下，包含文件中定义的链接优先于内嵌对象中定义的链接。

如果链接在内嵌的 SMIL 文件中定义，此链接只影响这个内嵌的 SMIL 文件。如果链接在 SMIL 文件中内嵌的非 SMIL 文件中定义，链接只影响内嵌文件的播放而不影响 SMIL 包含文件的播放。此限制在以后的 SMIL 版本中可能会取消。

下面我们介绍两个超链接元素 a 和 anchor。

a 元素

对于学过 HTML 的读者而言，a 元素的功能与 HTML a 元素的功能非常相似。SMIL 只不过增加了一个属性“show”用于在激活链接时控制源的时序行为。出于同步的考虑，“a”元素是透明的，即：它不影响它的子元素的同步。“a”元素不应该嵌套。“a”元素必须有 href 属性。

a 元素可以有如下属性：

☒ id

此属性在一个文件中唯一地标识一个元素。

☒ href

此属性包含了链接终点的 URI，它是 a 元素必需的。

☒ show

此元素在激活链接时，控制包含此链接的源文件的行为。它可以有下列值：

- **replace**：当前的演示暂停在当前状态，被链接终点的资源所代替。如果播放器提供了一种历史记录机制，当用户返回到源文件时，源演示从暂停的状态恢复播放。
- **new**：目的资源的播放在一个新的上下文中开始，不影响源资源。
- **pause**：当前的演示暂停在当前状态，目的资源在一个新的上下文中开始。当目的资源的显示结束后，源演示从暂停的状态恢复播放。

Show 属性的缺省值是“replace”。

☒ title

此属性提供了其所在元素的参考信息，我们建议所有的“a”元素都有“title”属性，且其属性值是一个有意义的描述。创作工具应该保证 SMIL 文件中引入的每一个元素都具有此属性。

下面我们来看一个例子来说明 a 元素的实际应用。

程序清单 10.1：

```
<smil>
<head>
<meta name="Title" content="Welcome to RealPlayer 8"/>
<meta name="Author" content="Roy Wilkie"/>
```

```

<meta name="Copyright" content="2000, RealNetworks"/>

<layout>
<root-layout height="300" width="350" background-color="black"/>
<region id="full_screen" left="0" top="0" height="300" width="350" z-index="1"/>
</layout>
</head>

<body>
<seq>
  <par>
    <a href="test.rt">
      <animation src="firstrun.swf" region="full_screen" fill="freeze" />
      <audio src="firstrun.rm"/>
    </a>
  </par>

</seq>
</body>
</smil>

```

这个文件是改造 realplayer 的欢迎界面，我们在其中加入了...语句，当链接被激活时，正在播放的 firstrun 资源文件会被 test 文件所代替。

程序清单 10.2:

```

<window type="marquee" bgcolor="yellow" height="80" width="380" >
  <clear/>
  <font size="+5" color="red">
    <b>"WELCOME TO THE REALPLAYER' WORLD"</b>
  </font>
</window>

```

运行这个例子时，首先播放器播放的是 firstrun.smi 文件，如图 10-4 所示，当把鼠标移到播放区域上的时候鼠标就变成了手形，就像 html 中的链接一样，单击鼠标播放器就播放 test .rt 来代替 firstrun.smi 文件，如图 10-5 所示。



图 10-4 播放 firstrun.smi 文件



图 10-5 播放 test.rt

下面再对这个例子做几个扩展。

(1) 如果把<par></par>中的代码改成：

```
<a href="test.rt" show="new">
  <animation src="firstrun.swf" region="full_screen" fill="freeze" />
  <audio src="firstrun.rm"/>
</a>
```

那么激活链接后，播放器会在当前播放的文件之外开始新的目标文件的播放。

(2) 如果把<par></par>中的代码改成：

```
<a href="test.rt" show="pause">
  <animation src="firstrun.swf" region="full_screen" fill="freeze" />
  <audio src="firstrun.rm"/>
</a>
```

那么激活链接后，播放器暂停正在播放的文件而开始新的文件播放。

anchor 元素

a 元素的功能是有限的，它只允许将链接和完整的媒体对象相关联。anchor 元素为 SMIL 实现了如下的功能：

(1) anchor 使用的“href”属性允许以媒体对象的空间和时间局部作为链接的终点。(相比之下，“a”元素只允许将链接和完整的媒体对象相关联)

(2) anchor 元素用“id”属性将媒体对象的局部作为链接的终点。

(3) anchor 元素允许用“coords”将对象分成空间局部。

(4) anchor 元素允许用“begin”和“end”属性将对象分成时间局部。begin 和 end 属性的值相对于媒体对象的开始。

下面介绍一下 anchor 元素的一些常用属性

☒ begin

在元素的实际开始处产生此事件。

☒ coords

此属性在一个可视媒体对象的演示区域定义了一个矩形。如果链接与一个矩形区域相

关联，此属性的语法和语义类似于 HTML 中 image map 的 coords 属性。矩形用四个长度值来定义：头两个值说明了矩形左上角的坐标。后两个值说明了矩形右下角的坐标。坐标值相对于可视媒体对象的左上角。如果坐标以百分数的形式给出，则是相对于媒体对象显示区域的整个宽和高而言。

包含错误坐标的属性将被忽略，比如 right-x 小于或等于 left-x，bottom-y 小于或等于 top-y 等。此外如果坐标值定义的矩形超出了媒体对象覆盖的范围，超出的宽度和高度在媒体对象的边界被切去。

coords 属性值的语法如下：

coords-value::= left-x "," top-y "," right-x "," bottom-y

☒ end

在元素的实际结束处产生此事件

☒ show

此元素在激活链接时，控制包含此链接的源文件的行为，在 a 元素中详细介绍过不再重复介绍。

☒ skip-content

引入此属性是为了 SMIL 以后的可扩展性，我们在前面介绍过，不再重复。

□ title

此属性提供了其所在元素的参考信息，我们建议所有的“anchor”元素都有“title”属性，且其属性值是一个有意义的描述。

看下面的例子，一个视频片断所占的屏幕空间被分为两个部分。两个部分有各自关联的链接。

```
<video src="http://site.com/MyVideo">
  <anchor href="http://site1.com/Video1" coords="0%,0%,60%,60%"/>
  <anchor href="http://site2.com/Video2" coords="60%,60%,100%,100%"/>
</video>
```

10.3 应用实例

现在对 SMIL 已经有了一个比较全面的认识，下面就举一个例子来做一个回顾。

首先创建 3 个 .rt 文件，分别命名为 test.rt，test2.rt 和 test3.rt。

程序清单 10.3: test.rt

```
<window type="marquee" bgcolor="yellow" height="50" width="200" >
  <clear/>
  <font size="+2" color="red">
    <b>"IT IS MY EXAMPLE!!"</b>
  </font>
</window>
```

程序清单 10.4: test2.rt

```

<window type="marquee" bgcolor="yellow" height="50" width="200" >
  <clear/>
  <font size="+2" color="red">
    <b>"it is the test's link!!"</b>
  </font>
</window>

```

程序清单 10.5: test3.rt

```

<window type="marquee" bgcolor="yellow" height="50" width="200" >
  <clear/>
  <a href="readme.html">
    <font size="+2" color="red">
      <b>"it is the video's link!!"</b>
    </font>
  </a>
</window>

```

程序清单 10.6: example.smi

```

<smil>
<head>
  <meta name="title" content="SMIL 示例"/>
  <meta name="author" content="翁宇翔"/>
  <meta name="copyright" content="寒星工作室"/>
<layout>
  <root-layout width="200" height="200" background-color="yellow" />
  <region id="test" left="0" top="150" width="200" height="50" />
  <region id="videotest" left="0" top="0" width="200" height="150"/>
</layout>
</head>
<body>
  <par>
    <a href="test2.rt">
      <text src="test.rt" region="test" />
    </a>
    <a href="test3.rt">
      <video src="videotest.rm" region="videotest"/>
    </a>
  </par>
</body>
</smil>

```

在这个实例中，用基本布局元素来对显示区域进行了上下两个部分的分割，分别用 `a` 元素链接到各自的目标文件。如图 10-6 所示。

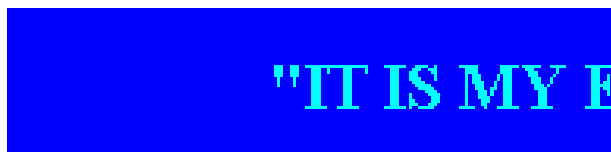


图 10-6 XML 使用 HTML 标签实例

当激活不同的区域时，会链接到不同的目标文件上。比如激活了上半部分的链接，那么播放器就开始播放 test3.rt 这个文件，如图 10-7 所示。



图 10-7 播放 test3.rt

在 test3.rt 文件中有这么一段代码：

```
<a href="readme.html">  
  <font size="+2" color="red">  
    <b>"it is the video's link!!!"</b>  
  </font>  
</a>
```

这里做了一个链接，这一点从图中的下划线也可以看出，而且链接的目标文件是一个 HTML 文档，当我们激活它的时候，播放器将调用浏览器打开目标 HTML 文件，就与 HTML 文件中的链接如出一辙。

10.4 小 结

同步多媒体集成语言 SMIL 开创了 Web 上多媒体的创作和展示的新阶段。在这一章里我们对 SMIL 的各个元素做了详尽而全面的介绍，最后使用一个综合实例来对整章作了一个回顾。读者在学习完本章之后应该能够根据需要编写自己的 SMIL 文件。

在全面了解了 WML 与 SMIL 之后，第 11 章将开始一个新的话题，即 XML 与电子商务。

第 11 章 XML 与电子商务

本章导读

随着网络的日益普及，人们不再满足于静态的欣赏网页和仅仅阅读网络上的文章，更迫切需要相互交换数据（交互）、进行消费、通信、资讯交流或者是客户服务一类的交互式的信息交流。电子商务正是为了满足这些要求而诞生的。

因为电子商务是网络的产物，所以它受到网络发展的制约。因此，虽然电子商务起源于七十年代，但真正的发展却是最近的十几年。在网络发展的初期，带宽一直是制约网络软件的瓶颈，那时电子商务的语言是 EDI。随着网络技术的发展，网络带宽已经不再是设计者所考虑的首要问题，软件的开发效率、使用效率和文档的可读性成为设计者需要考虑的第一要素。在这种背景下，XML 在电子商务中获得了越来越广泛的应用。

本章将会对 XML 在电子商务中的应用进行详细的介绍。

11.1 什么是电子商务

11.1.1 电子商务的定义

随着电子技术和因特网（Internet，网络）的发展，信息技术作为工具被引入到商贸活动中，产生了电子商务（Electronic Commerce，简写 EC 或 Electronic Business，简写 EB）。通俗的说，电子商务就是在计算机网络（主要指 Internet 网络）的平台上，按照一定的标准开展的商务活动。当企业将它的主要业务通过内联网(Intranet)、外联网(Extranet)以及 Internet 与企业的职员、客户、供销商以及合作伙伴直接相连时，其中发生的各种活动就称作电子商务。

到目前为止，人们尚未对电子商务有一个统一明确的认识，甚至有人认为电子商务可以追溯到以莫尔斯码点和线的形式在电线中传输的商贸活动。七十年代的电子数据交换（EDI—Electronic Data Interchange）技术的发展可以被认为是真正意义上的电子商务，但由于它的复杂性与非通用性，严重约束了其全面推广的可能。真正使电子商务迅猛发展，是在互联网通讯标准和 HTML 标准得到 IT 行业的支持，成为电子商务的主流之后，这也开辟了运用电子手段进行商务活动的新纪元。

下面给出 IBM 公司和美国政府分别为电子商务下的定义。

IBM 公司为电子商务（E-Business）所定义的概念包括三个部分：

- ☒ 内联网（Intranet）
- ☒ 外联网（Extranet）
- ☒ 电子商务（E-commerce）

它所强调的是在网络计算环境下的商业化应用，不仅仅是硬件和软件的结合，也不仅仅是通常意义下强调交易的狭义的电子商务（E-commerce），而是把买方、卖方、厂商及其合作伙伴在因特网（Internet），内联网（Intranet），和外联网（Extranet）结合起来的应用。

它同时强调这三部分是有层次的。只有先建立良好的 Intranet，建立好比较完善的标准和各种信息基础设施，才能顺利扩展到 Extranet，最后扩展到 E-commerce。

美国政府为电子商务定义的概念是通过 Internet 进行的各项商务活动，包括：广告、交易、支付、服务等活动，全球电子商务将会涉及全球各国。

总体来说，电子商务是两方或多方通过电子渠道交换一些种类的商业信息。电子商务涉及的范围十分广泛，或者说，电子商务并不在意卖些什么，它的目的只是一方按照双方预先规定的格式从另一方获得其所需要的某些类型的信息。

11.1.2 电子商务的三种业务模式

电子商务只是一个笼统的概念。具体实现网上交易时，通常有以下三种业务模式。

对客户直接销售

对客户直接销售，即 B2C (Business to Customer) 是最简单的一种业务模式。这种模式是由客户端直接浏览公司主页，当遇到想购买的商品时，就使用信用卡或是邮汇等方法付款。公司收到货款后，直接发货给消费者。

这种模式的优点是省略了中间商，使公司拥有最高的回报率。

企业对企业

即 B2B (Business to Business)，是与 B2C 类似的一种业务模式。与公司直接交易的不是消费者而是分销商，公司不再处理由消费者发出的信用卡和邮汇业务，而是直接从分销商账户上的转账。与消费者接触的是分销商，由他们完成收取消费者的付款和为消费者送货的服务。

这种模式有利于提高公司运作的自动化，

信息共享和内容耦合

信息共享和内容耦合模式有许多种形式，它可能是通过互联网由各个子部门向上一级部门上报的销售报告，也可能是智力产品的多部门平等的信息共享，这种例子在音乐、新闻等信息量大，更新速度快的产业部门相当常见。

90 年代以前，很少有公司采用这样的设计模式。但是，进入 90 年代后，随着网络的普及与高速发展，信息量以惊人的速度猛增。因此，许多新兴行业需要更多的有效信息。信息共享和内容耦合模式就是在这种环境下产生的。

这种耦合模式最突出的特点是产生了一种新的行业：信息中间商。信息中间商是专门收集信息供应商的有效信息，经过选择提供给客户浏览器（一般都是从多个信息供应商处获得信息，或者是从一个信息供应商的特定信息）。

以上的三种电子商务模式具有各自的优点，也为我们展示了在设计电子商务工程时使用的几种最主要、最流行的开发模式。需要根据公司的实际情况选用，比如公司的信息流通量和公司软硬件的实际开发实力。

11.1.3 在电子商务中使用 XML 的理由

在过去的 30 年里，电子商务所主要使用的语言是 EDI (电子数据交换)。EDI 是电子

商务的铺路人，它在过去的 30 年间更换了几种格式。各种公司通过使用各种行业交易标准进行电子数据交换，通过使用 EDI 节省了成本，提高了商业处理能力。在 80 年代早期，ANSI 就发现了制定电子商务标准的优势，并成立了一个正式的 EDI 标准委员会，它的代号为 X12。

EDI 技术初步预言了 Internet 的商业引用，EDI 不仅包括了特定的信息格式，还定义了通信协议，甚至一些硬件要求。实现这个系统的费用相当巨大，也正是这一点限制了 EDI 技术发展的市场。从实用角度来说，只有政府机构和大规模的公司才有能力完全实现 EDI 的应用。所以几乎所有现在的 EDI 标准都被修订过，使 XML 作为信息和传输的基础成为可能。

目前使用最多的 EDI 标准是 X12，它的特点是表达式非常紧凑。这样在信息交换时所需要的时间减少了。但是，为此付出的代价就是 X12 的可读性相当的差。如今，我们拥有着越来越大的带宽，因此，信息传输量的大小已经不再是影响公司信息传输时所花费的资金的主要矛盾了，而可读性的重要性就体现了出来。从可读性强的文档（即 XML）中提取有效信息所需要消耗的时间比从那些可读性差的文档中提取所消耗的时间要短的多。另一方面，使用 XML 语言编写文档时，可以使用比 X12 更简单、更易于阅读和理解的词汇表，这将大幅度的提高编写效率，缩短开发时间。

如何在电子商务中使用 XML？

XML 提供了一种标记数据的标准语法，并允许在消息中加入任何附加的信息。这种优势使得我们能够将脚本同商业规则相联系，使得表达式和消息结构完全可以直接得到使用。正是这种优势，我们才将 XML 作为在客户端和服务端之间传送的介质。

在电子商务的 XML 解决方案中，我们已经看到了一些行业所创立的具体的标记。下面的介绍里，读者将会看到一个较为普遍，并且越来越为人们所熟知的方案：BizTalk。BizTalk 的官方网站是 <http://www.BizTalk.com>，那儿有关于 BizTalk 的最新动态以及最全面的介绍。

11.2 XML BizTalk 框架

XML 为我们提供了一个电子商务的解决方案，BizTalk。有了这种通用的标准，就能解决前面谈到的问题了。XML 是一个国际通用的标准，它从一开始就得到了广泛的支持。不光是微软，还有微软的竞争对手都支持它。认证它的不仅仅是一家公司，而是 W3C。各家公司互相竞争的是它的框架，即它的 Schema。

XML 文件能实现数据和计算的分离，也就会让模板中的 Business logic 一层一层地分离出来，减轻模板的编码负担。XML 框架是驾驭 XML 文件的结构，它是一种高层次的结构控制。通过 XML 框架，Business logic 被转移到模板之外，真正实现了数据与计算的分离。只有通过框架，XML 文件才能实现通信的自由与高效，否则 XML 文件只是小范围内的自娱自乐工具。

现在已经有多少 XML 框架呢？可能在几个月之前，你还可以说“好像就只有 BizTalk 一家”。但现在你会发现有出了许许多多。BizTalk 是微软支持的，现在已被运用到了微软的整个解决方案框架中。微软把 XML 看成它的未来核心战略。它把 BizTalk 一手扶植起来

就是为了这个战略服务。除此之外，联合国（UN/CEFACT）和 OASIS 联合于 99 年底推出了 EBXML 动议，它已吸引了全球包括 XMLSolutions 的许多组织的支持，发展前途不可小视。除此之外，一些范围较小的框架还有 HR-XML。我相信在短期内，各种局部的框架会涌现出许多。从现在的情况来看，BizTalk 是最成功的。但现在的成功决不代表未来的成功，BizTalk 已经受到了挑战。

11.2.1 什么是 BizTalk

BizTalk 是什么？其实它包括的范围不只是框架。BizTalk 包含有框架部分，有 BizTalk Server，还有网上的接口，BizTalk 还为业界提供了一个免费 Schema 库。BizTalk 的范围是广泛的，它提供的服务是全方位的，它已经被微软加入到了它的整个解决方案架构当中。从某种意义上说，微软在冒险。因为 W3C 并没有完全推出 XML，XML Schema 的说明，微软是在用它已存在的影响力推销它自己的理念。

BizTalk 的免费 Schema 库是由支持 BizTalk 的业界企业制定的。它们把制定好的 Schema 提供给 BizTalk 认证，经过认证后的 Schema 被加入到 Schema 库中，供在线免费下载。这实质上是给一个特定的产业树立了一个目标。这个业界中的其它企业可以直接引用这些 Schema。换句话说，微软想要用它来扩充尽可能多的支持者，扩充微软的阵容。

下面把重点放在介绍 BizTalk 框架上。

信息是在软件发送的一种文件。信息区别文件的地方在于它期望得到回应，或是含有操作的意图。每当一个特定的应用发出一个信息，都是产生一个加头的 XML 文件。这是一个符合特定 Schema 的 XML 文件，是一个符合 BizTalk 框架的 XML 文件。当然 BizTalk 希望这个 Schema 是 BizTalk Schema 库中的，而且在它所加的头中，信息定义结构必须是符合 BizTalk 框架定义的。这个加头的 XML 文件被应用送到发送 BizTalk Server，由发送 BizTalk Server 加上一些传输特定参数后，被打包发送到接收 BizTalk Server。接收 BizTalk Server 收到信息包后会做一些相应的处理，将信息送到目的应用程序。

11.2.2 符合 BizTalk 框架的文件结构

☒ 文件根标记（ROOT）

```
<biztalk_1>
  xmlns="urn:schemas-biztalk-org:biztalk/biztalk_1.xml">
</biztalk_1>
```

文件根标记是必须的。这个标记以及它内部的名域(namespace)说明文件是符合 BizTalk 框架 1.0 版本的，这个标记名不能改动。

☒ 文件头标记（HEADER）

```
<header>
  <!-- Header and processing information is contained here -->
</header>
```

文件头标记是可选的。头信息被包含在这个标记中，它其实就是 Business logic 中被提取出来的一部分。

☒ 文件主体标记 (BODY)

```
<body>
  <!-- Business transaction information is contained here -->
</body>
```

文件主体是必须的。

所以总体看，一个完整的符合 BizTalk 的文件是这样的：

```
<biztalk_1 xmlns="urn:schemas-biztalk-org:biztalk/biztalk_1.xml">
<header>
  <!-- Header and processing information is contained here -->
</header>
<body>
  <!-- Business transaction information is contained here -->
</body>
</biztalk_1>
```

11.2.3 深入讨论 BizTalk 框架

文件根标记没什么复杂的内容，只是机械的声明而已。

下面来仔细看看文件头标记，然后理解一下文件头标记里元素及其属性和子元素的内容。下面是一个完整的文件头标记：

程序清单 11.1:

```
<header>
<delivery>
<message>
  <messageID>11111</messageID>
  <sent>2001-07-02T19:00:01+08:00</sent>
  <subject>New Purchase Order</subject>
</message>
<to>
<address>http://www.reciever.com/reciever.asp</address>
<state>
  <referenceID>1</referenceID>
  <handle>1</handle>
  <process>recieveprocess</process>
</state>
</to>
<from>
  <address>http://www.sender.com/send.asp</address>
```

```

    <state>
      <referenceID>2</referenceID>
      <handle>2</handle>
      <process>sendprocess</process>
    </state>
  </from>
</delivery>

<manifest>
<document>
  <name>product</name>
  <description>Your Order Goes Here</description>
</document>
<attachment>
  <index>1</index>
  <filename>produc_t.bmp</filename>
  <description>Product Image</description>
  <type>bitmap</type>
</attachment>
</manifest>
</header>

```

以上的代码分为两段是希望读者能更容易理解文件头标记的结构。

很明显，文件头标记包含了 **delivery** 元素和 **manifest** 元素。并且它们又有各自的子元素。下面将以程序清单 11.1 为例，对这两个元素进行详细的说明。因为子元素的层套关系比较多，读者阅读的时候要留意元素与元素之间的层次关系。

☐ delivery 元素

delivery 元素允许出现的次数最多是一次，它的作用是描述信息，以及信息从哪儿来，发到哪儿去。

delivery 元素包含了三个元素：**message** 元素，**to** 元素，**from** 元素。这三个元素都是必须出现的。

message 元素提供描述符和时间以及信息标签。该元素中包含的信息对框架本身是没有用处的，但是对跟踪 BizTalk 信息队列却是非常有用的。**message** 元素又包括了：**messageID** 元素；**sent** 元素；**subject** 元素等三个元素。其中只有 **subject** 元素是可选的，前两个都是必须出现的。

以下的这段代码是从程序清单 11.1 中节选出来的，**message** 元素部分。

```

<message>
  <messageID>11111</messageID>
  <sent>2001-07-02T19:00:01+08:00</sent>
  <subject>New Purchase Order</subject>

```

```
</message>
```

其中, messageID 元素是应用程序中的 GUID 设定的顺序信息编号。sent 元素的格式是遵循 XML-DR 规定的 datetime.tz 数据类型 (既 ISO8601 的子集)。例子中的时间, 是指信息被发送的时间是 2001 年 7 月 2 日 19 点正, 发送的时区比格林威治时间早 8 小时 (北京时间)。subject 元素是在调试程序时, 用于描述信息的类型。

从结构上看, to 元素和 from 元素十分相像, 它们都包含了一个 address 元素和一个 state 元素。address 元素是必须的, 而 state 元素是可选的。以下面的这段节选的代码为例, 分析一下 to 元素和 from 元素的结构。

```
<to>
<address>http://www.reciever.com/reciever.asp</address>
<state>
  <referenceID>1</referenceID>
  <handle>1</handle>
  <process>recieveprocess</process>
</state>
</to>
```

address 元素是一个 URI, 它表明的是消息的逻辑地址。关于 URI 和 URL 需要简单的介绍一下。XML 文档可用于 Web, 正如 HTML 和其他文档一样。使用时, 也如 HTML 文档一样, 被统一资源定位符 (Uniform Resource Locator, 简称为 URL) 所引用。虽然 URL 已被人们广泛理解并被广泛支持, 但 XML 规范使用的是更为通用的统一资源标识符 (Uniform Resource Identifier, 简称为 URI)。URI 对于定位 Internet 上的资源是更为通用的架构, 更为注重资源而不太注重位置。理论上说, URI 可找出镜像文档的最为近似的副本或是找出已经从一个站点移动到另一站点的文档。实际上, URI 仍然处于进一步的研究之中, 被当前的软件所唯一支持的一种 URI 正是 URL。

state 元素包含 referenceID 元素, handle 元素和 process 元素。referenceID 元素必须出现, 是唯一的商务交换时的描述符。handle 元素是可选的, 提炼 referenceID 以指定交换的一个特定部分, 例如在一个工作流程中的一步。process 元素, 也是可选的, 提炼 handle 以处理内容。

为了方便读者的查找和使用, 表 11-1 给出了 delivery 元素的结构:

表 11-1 delivery 元素结构

delivery						
message			to & from			
messageID	sent	subject	address	state		
				referenceID	handle	process

☒ manifest 元素

manifest 如果译成中文为载货单,那么读者很快就明白了 manifest 元素的主要功能:从一个应用程序接收传输的内容。有两种类型的文档可以被传输, BizTalk 消息和非 XML 附件。相对应这两种类型的传输文档, manifest 元素有两个子元素,即 document 子元素和 attachment 子元素。这两个子元素均可多次出现在文件头标记中。下面依然使用程序清单 11.1 的 manifest 部分为例,进行介绍。

```
<manifest>
<document>
<name>product</name>
<description>Your Order Goes Here</description>
</document>
<attachment>
  <index>1</index>
  <filename>produc_t.bmp</filename>
  <description>Product Image</description>
  <type>bitmap</type>
</attachment>
</manifest>
```

在这段节选中, document 元素和 attachment 元素都仅出现了一次。

document 元素是用来描述包含在消息传递中的一个 BizTalk 消息。document 元素含有两个子元素: name 子元素和 description 子元素。

- ☒ name 元素是发送文档的根元素的名称,如例子中的“<name>product</name>”,就是表示发送的文档的根元素是 product。
- ☒ description 元素并不是一个必须的元素,它为接收者提供一个关于发送的文档的类似于注释的一个可读的描述。

attachment 元素是用来描述包含在消息传输中的非 XML 文档的附件的信息的。attachment 元素包含了 4 个子元素: index 子元素, filename 子元素, description 子元素, type 子元素。其中, index 子元素和 filename 子元素是必须的,而 description 子元素和 type 子元素是可选的。

- ☒ index 元素指定的是附件定义符。
- ☒ filename 元素是传输的附件的文件名。该文件名不含路径,但是要有文件名的后缀。
- ☒ attachment 元素的 description 子元素和 document 元素的 description 子元素作用相似,只不过 attachment 元素的 description 子元素描述的是附件的信息。
- ☒ type 元素定义了附件文档的文件类型。

manifest 元素的介绍结束了,为了方便读者的查找和使用,表 11-2 给出了 manifest 元素的结构:

表 11-2 manifest 元素结构

manifest					
document		attachment			
name	description	index	filename	description	type

前面详细论述了 BizTalk 框架中文件头标签的结构, 以及各个元素的功能和定义。下面对文档主体在格式上的要求做一个介绍。

文档主体是设计者参与设计的部分, 也是 BizTalk 不可缺少的一部分。文档主体的根元素一般都取值为文件头标签中 document 元素的 name 元素的取值。继续以 11.1 为例, 设计的文档主体的结构如下:

```
<body>
  <product
    xmlns = "urn: schemas-biztalk.org:.....">
    你的XML文档以及数据
  </product>
</body>
```

文档的主体是必须的, 它是模式设计者参与设计的部分。没有现成的模式时, 你就需要为你的商务处理流程开发一种新的模式。所写的 XML 文档应该遵循这些模式并作为 body 元素的子元素出现。文件头元素需要与一些相关的数据描述一起到达发送端。Body 元素是发送消息的原因, 它有一个或多个子元素。每个子元素都是根据 BizTalk 模式书写的文档。

以上就是 BizTalk 框架的详细讨论, 关于 BizTalk 的具体使用实例, 将在下面的 11.3.1 中做出介绍和讨论。

11.3 BizTalk 模式的实现

前面小节一直围绕着 BizTalk 的理论、技术、框架来讨论, 本节将向读者展示 BizTalk 的具体实现。

11.3.1 BizTalk 的使用

为了便于读者的理解, 下面以一个 CD 音像店为例, 使用 BizTalk 框架构建一个简易的电子商务。

程序清单 11.2:

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



```

<!-- CD的价格和货币的种类 -->
<ElementType name="Currency" content="textonly" model="open" dt:type="string"/>
<ElementType name="Price" content="textonly" model="open" dt:type="number"/>
<ElementType name="Cost" content="eltonly" model="close">
  <element type="Currency"/>
  <element type="Price"/>
</ElementType>

<!-- CD的出版信息 -->
<ElementType name="Title" content="textonly" model="open" dt:type="string"/>
<ElementType name="Singer" content="textonly" model="open" dt:type="string"/>
<ElementType name="Subject" content="textonly" model="open" dt:type="string"/>
<ElementType name="PublishDate" content="textonly" model="open" dt:type="date"/>
<ElementType name="Reference" content="textonly" model="open" dt:type="string"/>
<ElementType name="RefURL" content="textonly" model="open" dt:type="uri"/>
<ElementType name="TitleData" content="eltonly" model="open">
  <element type="Title"/>
  <element type="Singer"/>
  <element type="ISBN"/>
  <element type="Subject"/>
  <element type="PublishDate"/>
  <element type="Reference"/>
  <element type="RefURL"/>
</ElementType>

<!-- CD的简要说明 -->
<ElementType name="CDNumber" content="textonly" model="open" dt:type="string"/>
<ElementType name="Summary" content="textonly" model="open" dt:type="string"/>
<ElementType name="CDinf" content="eltonly" model="open">
  <element type="CDNumber"/>
  <element type="Summary"/>
  <element type="TitleData"/>
  <element type="Cost"/>
</ElementType>

</Schema>

```

这个实例并不复杂，下面将对它进行一步步的详细解释。读者可以对照 11.2 中对 BizTalk 的理论描述来深入的理解 BizTalk。所有的 BizTalk 模式文档必须使用 XML-DR 模式，所以必须将命名空间设定为 Microsoft-XML-DR。一旦使用了这个命名空间，我们就可以使用 XML-DR 模式的各种数据类型了。命名空间的定义，在程序 11.2 中仅包括了第一到第四行，这就构成了该程序的第一部分。

在程序的第二部分，定义了一个 Cost 元素，用来表征 CD 的销售价格。Cost 包含了 Currency 和 Price 两个子元素。Price 定义了 CD 的价格，Currency 定义了该价格所使用的货币种类。由这两部分共同体现出正确的 CD 的销售价格。

```
<ElementType name="Currency" content="textonly" model="open" dt:type="string"/>
<ElementType name="Price" content="textonly" model="open" dt:type="number"/>
<ElementType name="Cost" content="elonly" model="close">
  <element type="Currency"/>
  <element type="Price"/>
</ElementType>
```

程序的第三部分定义了 CD 的各项出版信息。例如 Title 定义了 CD 的 CD 名，Singer 定义了 CD 的演唱者。其他的几个元素代表的含义就不一一列举了。

```
<ElementType name="Title" content="textonly" model="open" dt:type="string"/>
.....
<ElementType name="PublishDate" content="textonly" model="open" dt:type="date"/>
<ElementType name="RefURL" content="textonly" model="open" dt:type="uri"/>
```

第三部分中的 PublishDate 元素应用了 XML-DR 模式中的时间数据类型，采用的是 ISO 8601 格式。也就是按照 YYYY-MM-DD 的格式。

RefURL 元素与 Reference 元素不同，RefURL 为 CD 的参考链接，它应用的是统一资源定义符数据类型。而 Reference 元素是 CD 的参考曲目，它应用的是字符串数据类型。读者可以注意一下 URI 数据类型的定义方法。这种数据类型在电子商务的设计中出现非常频繁。

在多个 CD 的出版信息元素后，还定义了一个 TitleData 元素来统一包含这些元素。这是一个模块化的设计思路。需要注意的是 CD 的演唱者的数量问题，一张 CD 至少有一个演唱者，也可能有多个演唱者，因此，在 TitleData 的定义中使用如下的语句才能使这个元素变得合理，也避免了模式在使用中的局限性。

```
<element type="Singer" minOccurs="1" maxOccurs="*">
```

当然这样的情况在很多元素中都会出现，我们只是做一个 minOccurs 和 maxOccurs 的使用说明而已。

程序的第四部分，完成了整个程序的设计，结束了根元素 CDinf 的定义。CDinf 中包含了前几部分的所有元素，并且新定义了两个元素 CDNumber 和 Summary。

整个程序的设计类似于 C/C++ 或者 VB 等高级语言的模块化设计思想。读者可以结合自己以前的编程经验来对照这个模式的编程思路。

基于这个模式文档，编写了下面的 XML 文档。

程序清单 11.3:

```
<?xml version="1.0">
<CDinf xmlns="x-schema:CD.xml">
  <CDNumber>
    10001
  </CDNumber>

  <Summary>
    It's a very CD about String's best songs.
  </Summary>

  <TitleData>
    <Title>The Best of String</Title>
    <Singer>String</Singer>
    <Subject>Hot Hits of String</Subject>
    <PublishDate>2001-07-01</PublishDate>
    <Reference>String's Gold 10 Years</Reference>
    <RefURL>http://www.music_String.com.cn</RefURL>
  </TitleData>

  <Cost>
    <Currency>Chinese $</Currency>
    <Price>120</Price>
  </Cost>
</CDinf>
```

上面的 XML 是纯粹的根据程序 11.2 的结构一一对应而编写的。从中可以看到 XML 的可读性非常强。

实现这个 XML 的传输的 BizTalk 的实例如下:

程序清单 11.4:

```
<biztalk_1 xmlns="urn:schemas-biztalk-org:biztalk/biztalk-1.0.xml">
<header>
  <header>
    <delivery>
      <message>
        <messageID>97</messageID>
        <sent>2001-07-02T19:00:01+08:00</sent>
        <subject>CD Information</subject>
      </message>
    <to>
      <address>http://localhost/buy.asp</address>
    </to>
```

```
<from>
  <address>http://localhost/in/index.asp</address>
</from>
</delivery>
<manifest>
  <document>
    <name>CD Information</name>
  </document>
</manifest>
</header>
</header>

<body>
  <CDinf xmlns="urn:schemas-biztalk.org:CD.xml">
    <CDNumber>
      10001
    </CDNumber>

    <Summary>
      It's a very CD about String's best songs.
    </Summary>

    <TitleData>
      <Title>The Best of String</Title>
      <Singer>String</Singer>
      <Subject>Hot Hits of String</Subject>
      <PublishDate>2001-07-01</PublishDate>
      <Reference>String's Gold 10 Years</Reference>
      <RefURL>http://www.music_String.com.cn</RefURL>
    </TitleData>

    <Cost>
      <Currency>Chinese $</Currency>
      <Price>120</Price>
    </Cost>
  </CDinf>
</body>
</biztalk_1>
```

为了读者阅读时能突出重点，在程序 11.4 中一些可选的元素都被隐去了。虽然程序如此简单，但仍然能对发出的交易或者浏览请求做出信息的反馈。

11.3.2 BizTalk Jumpstart 工具包简介

要编写一个电子商务程序，就需要一个好的环境。类似于 PWS 和 IIS 提供的 Web 服务器一样，开发电子商务程序也需要提供一个虚拟的网络服务器。Microsoft 公司开发了这样一个基于 Windows 平台的 BizTalk 服务器，即 BizTalk Jumpstart 工具包。这是一个免费软件，下载的网址是 <http://www.biztalk.org/resources/tools.asp>。

因为 Jumpstart 是一个图形化的编辑工具。初学者十分容易掌握它的操作，并且已经有相当的读者正在使用或者早已使用过 Jumpstart，因此对于 Jumpstart 工具包的使用在下文将做一个简要的介绍。

注意：BizTalk Jumpstart 工具包要求 Windows 2000 做为它的运行环境。

BizTalk Jumpstart 工具包包含以下的组件：

- ☒ 选择器
- ☒ 计时器
- ☒ 一致性检测器
- ☒ 插入产生器
- ☒ 信息组件
- ☒ 应用程序适配器
- ☒ 命名空间服务器
- ☒ 属性管理器

选择器其实是一个 COM 组件，它负责接受客户端所发出的信息，并转达到相应的应用程序适配器。

计时器是用来测试对程序进行不对称信息提交时程序的运行状况。

一致性检测器是用来维护程序信息的工具。

其他的组件比较简单就不再赘述了。

11.4 小 结

电子商务是数字时代公司运作的必然产物。电子商务的最大特点是低成本，高效率。虽然电子商务相对于其他商务模式来说多了一个软件开发期，但是，一旦电子商务的模式确立，公司的营业范围将会有几倍到几百倍的扩大。

电子商务是依存于网络的发展的。在今天，没有人还会怀疑网络的生存必要。越来越多的网民在不同年龄阶层和不同生活层次的人们中诞生。网络社会与现实社会的人群构成也越来越相似。网络的带宽与速度在近 10 年里也有了飞速的发展。从 14.4K 的调制解调器到 56K 的调制解调器到 ISDN、ADSL 到宽带网，发展的迅速是其他行业望尘莫及的。这些都为电子商务创造了良好的发展土壤。可以说电子商务已经渐渐的与网络结合，并且密不可分。

本章中介绍的电子商务以其在实际应用中与 XML 的结合为核心，重点阐述了 XML 在现代电子商务中的应用。并且介绍了 XML BizTalk 电子商务框架，以及运用 BizTalk 框架如何开发特定的电子商务。

到此为止，我们已经对 XML 的高级技术以及其应用有了一个全面深入的了解。最后一章将系统地讨论一下 XML 的体系结构以及几种其他扩展与应用。

北京和信电子出版社

第 12 章 XML 扩展

本章导读

本章的内容主要是补充并总结整本书的内容，开始先讲述 XML 的标准体系，并且对于 XML 中的数据也作一些必要的说明，在后面主要是承接上面几章对 XML 的扩展和应用进行进一步的说明。本章介绍的主要内容有：

- ☒ XML 的体系结构
- ☒ XML 与数据
- ☒ CDF, MathML, SVG

12.1 全面理解 XML

12.1.1 XML 的标准体系

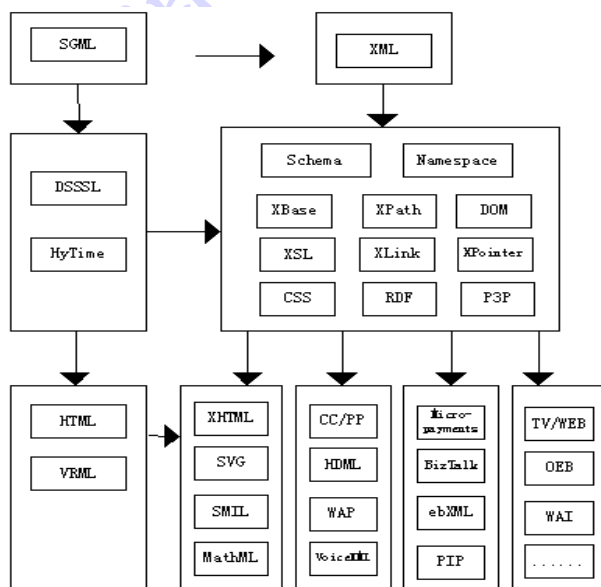


图 12-1 XML 标准体系

与 SGML 相关标准的体系相似，XML 相关标准也可分为元语言标准、基础标准、应用标准三个层次，如图 12-1 所示。

- ☒ 元语言标准（meta-Language）：定义的是用来描述标准的元语言，在 XML 标准体系中就是 XML 标准。XML 标准在 XML 标准体系中与 SGML 标准在 SGML 标准体系中的地位相似，是整个体系的核心，其他 XML 相关标准都是用它制定的或为其服务的。
- ☒ 基础标准（Foundation Standards）：这一层次的标准是为 XML 的进一步实用化制

定的标准,规定了采用 XML 制定标准时的一些公用特征、方法或规则。在第一小节中讨论的标准基本都属于这一层次。

- ☒ 应用标准 (Application Standards): XML 已开始被广泛接受,大量的应用标准,特别是针对 internet 的应用标准,纷纷采用 XML 进行制定。有人甚至认为,XML 标准是 Internet 时代的 ASCII 标准。在这个 Internet 时代,几乎所有的行业领域都与 Internet 有关。而这些行业一旦与 Internet 发生关系,都必然要有其行业标准,这些标准往往是采用 XML 来制定的。当前较为重要的应用标准主要包括:
- ☒ 用于 XML 显示的标准: XHTML (采用 XML 对 HTML 的重新定义), SVG (有关矢量图形的), SMIL (有关多媒体同步显示的), MathML (有关数学公式符号的);
- ☒ 用于移动设备的标准: CC/PP (移动设备的内容协商与信息交换)、HDML (手持设备)、WAP (无线应用设备)、VoiceXML (通过语音进行 web 访问);
- ☒ 用于电子商务领域的标准: Micropayments (W3C 制定的), BizTalk (Microsoft 发起的电子商务的 schema 库), ebXML (联合国 UN/CEFACT 小组和 OASIS 共同发起的), PIP (由诸多 IT 业的巨子组成的一个标准化组织 RosettaNet 的应用网络标准。), cXML, xCBL, tpaML 等等;
- ☒ 其他领域的标准: TV/web (web 电视), OEB (电子图书), WAI(方便残障人进行 Web 访问)。

12.1.2 XML 的驱动

HTML 概念是一种包含相对小标准标记集的标记语言,与一些多少标准化的行为相关。XML 概念是一个无限大的可能标记的集合,相关的行为根本没有标准。行为的规范必须来自其它某个地方。在发布时,这通常是一个样式表,但是在其它领域可以像 JavaBeans 一样柔性,或者像一种行业标准协议一样专业,程序员在其上编写标准化应用。

XML 支持者将这一点总结为 XML 定义句法而不是语义。一些理论家则反对说这一简单化的表述忽视了 XML 句法对象和构成它们所表示的 XML 数据(如元素和属性)的语义联系。然而,"句法而非语义"这一口号试图强调的更多,更简单:不像 HTML 标记,XML 标记没有预定义的含义。意义或行为必须由以程序的运行条件或者脚本或者用样式表的声明条件甚至古老而好用的普通文本提供。

当预期的 XML 用户可怜地询问 XML 如何在他们的 Web 浏览器上显示时,关于这一点的混淆就很明显了。回答是它并不显示--至少不是靠它自己。

要在一个浏览器内仿效现在对 HTML 所做的,你必须单独提供 HTML 作为一个整体但是难以管理的整体所提供的:你必须提供一个文档的内容(在 XML 中表示)和对它的处理,而这必须用程序定义(用脚本)或者声明它(用样式表)。

当前缺乏一种即使对于 XML 也足够强大同时又易于使用的样式表语言,这阻止了 XML 作为 Web 文档的普遍使用。为 HTML 开发的样式表语言层叠样式表(CSS)可用于为 XML 文档应用样式,但它不具备基于 XML 发布通常所需的转换和生成结构(例如目录)的能力。

文档样式语义和规范语言(The Document Style Semantics and Specification Language),

为 SGML 设计的 ISO 样式表语言,具有高级发布项目所需的功能。但是 DSSSL(与"whistle"同韵)有一个基于模式(Scheme)程序设计语言的句法,许多人会发现很难学习。它也缺乏一个丰富的声明层,这使得几乎不可能保证独立开发的样式表编辑器能互操作。

这正是扩展样式语言(XSL)的切入点。作为一开始的 XML 大计划的一部分,XSL 是一种新的语言,将结合 DSSSL 的功能和 XML 的简单性以及层叠样式表已建立的"样式属性"术语。1998 年 1 月建立的一个 W3C XSL 工作组正忙于定义这一使基于 XML 的 Web 发布成为可能的语言。

虽然一个最终的 XSL 规范还需要几乎一年时间,第一个 XSL 工作草案现在已经在 W3C Web 网站上发布了,网址是 <http://www.w3.org/TR/WD-xsl>。随着它的进入新世纪,这一初期规范值得任何想进行电子发布的人审慎关注。

12.1.3 XML 与数据

由于还没有一个足够强大的样式表语言以使 XML 说明它作为一种发布方法的优越性,第一批 XML 应用是基于它自己能做的事情的:传输结构化数据。可以说 XML 只是一个单一的、可被人阅读的句法。

通过串行排列任何种类的结构化数据,包括相关数据,以一种使其能用简单、随处可见的标化工具处理和显示的方式,XML 给我们提供了一个单一的、人可阅读的句法。一个标准的、易于处理的串行数据格式蕴涵的更大意义是难以想象的,但是它们显然将对电子商务有巨大的影响。另外,看来很清楚,电子商务将最终在一般意义上成为商务的同义词。

XML 之于数据正如 Java 之于程序,将使数据与平台和厂商无关。这一能力正在推动一波 XML 中间件应用,1999 年初将开始随处可见。然而,XML 支持数据和元数据交换的能力不应使我们的注意力从 XML 最初的设计目的上转移。XML 的设计者考虑的不仅仅是一个数据的传输层,而且是一个通用的媒体无关的发布格式,这将支持使用每一种语言的所有技术水平的用户。

媒体无关的发布实际上是一个比数据交换要难的多的问题。事实上,可以说一般意义上的发布的需求是数据交换需求的超集。XSL 的到来将使通用的发布解决方案成为可能,其后果还很少有人认识到。

理解 XML 革命性潜力的关键在于它只是一个更宏伟图景的一部分。XML 本身可以提供标准化的数据库和电子表格的交换格式。这很好。但是 XML 和 XSL 在一起也可以取代现有的字处理和桌面出版格式。它实际上可以给我们一个单一的、完全国际化的格式,具有几乎无限的打印和在线发布的能力,在所有产品和所有平台上都完全能互操作。这一点所意味的远远超越了数据交换和 Web。

XML 和 XSL 的组合可能比今天的 HTML 要复杂和难以处理的多,所以它最初将由一些手工处理大而专业化的出版应用的专家使用。这些应用将要求最高程度的自动化和媒体无关性,即报纸、工商行名录、百科全书、产品目录、电视节目表等等。

只有当普通字处理和桌面出版程序开始用 XML 和 XSL 的组合而不是专有格式来存储文件时,标准化的处理才会开始从这一专家的专业化群体向外扩展。这不是一个技术问题,而是一个经济问题,因为出版工具的大厂商在历史上依赖于专有格式以限制他们的用户。只

有当普通用户开始意识到标准化、开放的格式的益处，并且开始要求对其的支持时，厂商才会向其转变。

一个标准化的数据和表达格式的好处是无法抵御的。它们包括：

- ☒ 应用和平台间内容和样式的完全互操作性；
- ☒ 内容创建者脱离厂商对生产工具的控制；
- ☒ 用户选择他们自己对内容查看方式的自由。
- ☒ 容易创建强有力的处理大规模内容的工具；
- ☒ 独立软件开发商的一个公平竞争的场地；
- ☒ 所有媒体上的真正国际化发布。

相信用户对这些好处的认识，将最终迫使厂商支持标准化方法，正如用户对访问因特网的需求迫使厂商支持 Web 一样。

作为其后果，许多种类的桌面软件应用生产者和消费者之间的关系将重新建立，这将被证明为对我们大家都有巨大的好处。这将意味着少数大公司对市场的控制的终结，另外，可能更重要的是，少数大国对市场的控制的终结。结果将是更好的产品和人类之间更好的通讯。

12.2 XML 扩展

XML 是新出现的事物，作为一种数据的载体，它已经被广泛的应用于各个行业，比较成熟的技术当属微软 IE 的 CDF（通道定义格式）和 ForML（公式标记语言）。当然 XML 还有其它很多的应用，其中有些已经在应用了，有些明显是可以预见的。下面就介绍 XML 的应用。

12.2.1 通道定义格式（CDF）

通道定义格式 CDF（Channel Definition Format）是微软在 IE4.0 上使用的 XML 数据格式，用于描述活动通道的内容和桌面部件，并指明通道的信息及其更新情况。用户通过 CDF 可以在最短时间获取网页的更新内容或更新通知。Web 站点使用频道向预订站点的用户传送信息，改变了过去那种坐等用户前来浏览并获取信息的状况。这也叫做 Web 广播或是“推”。

CDF 文档是一个 XML 文件，与被“推”的站点的 HTML 文件分别存放，但是却链接到此 HTML 文件上。CDF 文档中的频道定义决定了要发送哪个页面。页面可以通过发送通知向预订者加以推送，但也可以发送给整个站点，或是在访问者方便的时候自己来“拉”信息。

用户可向自己的站点添加 CDF，而不用改变现存的所有内容。只要在页面上添加与 CDF 文件的一个不可见的链接即可。当浏览者访问该页面时，浏览器显示一个对话框，询问浏览者是否要预订频道。如果浏览者选择了预订，则浏览器就下载描述频道的 CDF 文档。然后浏览器将 CDF 文档用指定的参数与用户自己的选项结合起来，以便决定什么时候检查服务器上的新内容。这实际上不是真正的“推”，因为客户必须初始化连接，但是这确实又是在没有浏览请求的情况下发生的。

12.2.2 数学标记语言 (MathML)

MathML 可以说是最“古老的”XML 语言之一，它使得数学成为得到 XML 恩泽的第一个领域。在 MathML 出现之前，由于 HTML 缺乏描述数学表达式的标记，这些表达式不得不利用其它公式编辑工具先存为图像格式再插到网页中去，从而大大降低了传输速度。而且制作一个包含了众多数学公式的论文页面也相当繁琐，但浏览者却不得不花许多时间来等待众多公式图像的下载，这甚至已经成为对科学交流的阻碍。MathML 正是在这种情况下诞生的，W3C 于 1998 年 4 月 7 日发布了 MathML 的 1.0 版本，随后又于 1999 年 7 月 7 日发布了升级版 1.01。现在 MathML2.0 草案也在讨论之中。

MathML 专门用于描述数学符号并且捕获其结构与内容，它的目的是使数学公式及科学内容可以在 Web 上使用和重用，并且可以在其他应用系统上，如计算代数系统、排版打印系统及语音合成系统等。MathML 从表现形式和语意两个不同的角度定义了两大类标记，其中 28 个 MathML 标记描述了抽象符号结构，而另外 75 个标记提供了一种明确指定表达式意义的方法。这样一来，MathML 不但能够用于高质量显示系统编码数学符号，也可以用来为科学软件或声音合成软件等基于语义的应用软件编码其数学内容。

下面通过一个例子来说明它们的表示方法：

$$c = \sqrt{a^2 + b^2} / 2$$

上面的公式表示的是 a 的平方加上 b 的平方开跟号后在除以 2 的结果等于 c。先用表形的标记上式，则为：

程序清单 12.1:

```
<mrow>
<mi>c</mi>
<mo>=</mo>
<mfrac>
  <mrow>
    <msqrt>
      <msup>
        <mi>a</mi>
        <mn>2</mn>
      </msup>
      <mo>+</mo>
      <msup>
        <mi>b</mi>
        <mn>2</mn>
      </msup>
    </msqrt>
  </mrow>
  <mrow>
    <mn>2</mn>
  </mrow>
</mfrac>
```

</mrow>

上例中<mrow>表示水平的部件，起到封装的作用，其子元素排列在一行；<mfrac>表示分式，它的第一个子元素位于分号之上，第二个子元素位于分号之下；<msup>表示乘方，它的第二个子元素位于第一个的右肩；而<msqrt>是根号，其子元素放在根号下。另外，在 MathML 中不能直接引用数字、变量和运算符，分别需要用标记对<mn>，<mi>，<mo>括起来。

下面给出它的表义公式：

程序清单 12.2：

```
<reln>
<eq/>
<ci>c</ci>
<apply>
<divide/>
  <apply>
    <root/>
  <apply>
    <plus/>
  <apply>
    <power/>
    <ci>a</ci>
    <cn>2</cn>
  </apply>
  <apply>
    <power/>
    <ci>b</ci>
    <cn>2</cn>
  </apply>
</apply>
</reln>
```

在表义的标记中使用<apply>来对数据进行封装，数字和变量同样不能直接出现，分别用<cn>和<ci>来表示；在表义的标记中各种各样的运算符号都用专门的标记来表示，例如<reln>，<root>，<times>，<divide>等等。

在实用中，表形表达式和表义表达式有时候可以结合起来使用，使用表形表达式作表义表达式的注释，而用表义表达式作表形表达式的注释。使用时需要用到 semantics 元素和 annotation-xml 元素，semantics 元素把内容和注释全部包含起来，annotation-xml 元素用于指明注释的类型。比如：

程序清单 12.3:

```
<semantics>
  <mrow>
    ...
  </mrow>
  <annotation-xml encoding="MathML-Content">
    <apply>
      ...
    </apply>
  </annotation-xml>
</semantics>
```

或者写成下面的形式:

程序清单 12.4:

```
<semantics>
  <apply>
    ...
  </apply>
  <annotation-xml encoding="MathML-Presentation">
    <mrow>
      ...
    </mrow>
  </annotation-xml>
</semantics>
```

最后要指出的是假如要把 MathML 插入到 HTML 等网页中,需要把这些内容都用 `<math>` 元素包含起来。

12.2.3 可扩展矢量图形规范 (SVG)

可扩展矢量图形规范 SVG (Scalable Vector Graphics) 是一种基于 XML 的用来描述二维矢量图形和矢量/点阵混合图形的置标语言, SVG 规范定义了 SVG 的特征、语法和显示效果,包括模块化的 XML 命名空间 (namespace) 和 SVG 文档对象模型 (DOM)。在新近出台的 SVG 的第八个草案中,为 SVG 提供了两种不同的形式,即样式化 SVG 和交换型 SVG,它们各自有不同的 DTD 及 MIME 类型。样式化 SVG 允许对图形对象进行样式添加,它可以通过引用外部样式文件、在文件头中预先进行样式声明和通过属性为元素定义样式三种方式使用样式单,是 SVG 用于网络环境的推荐存储格式。而交换型 SVG 取消了对样式单的支持,完全使用元素属性描述各个图形对象的显示效果。在未来的网络传输中,交换型 SVG 将会充当现在印刷业广泛使用的图形格式—EPS 格式的网络版。并且,由于所有显示信息都封装到 XML 的属性中,交换型 SVG 还可以作为 XSLT 转换后所得到的结果文件格式,广泛应用于 XML 文档显示效果的描述中。

SVG 除了单独使用外，还可以在 XML 文件中作为命名空间引入，或者用作 HTML 文件中的特殊对象。同样，SVG 作为一种基于 XML 的语言规范，也具有 XML 的可扩展性，可以在 SVG 文件中引入其它置标语言的命名空间。这些特性使得 SVG 在互联网上将畅通无阻，比起采用二进制文件格式不能与其它网页语言兼容的 Flash 更胜一筹。

SVG 除了支持 HTML 中常用的标记，如文本、图像、链接、交互性、CSS 的使用、脚本（Script）外，还提供了大量针对图形、图像、动画的特定标记。这些包括 SVG 对矢量图形的支持，SVG 对图像过滤的支持和 SVG 对于动画的支持。

下面给出一个实例，具体的信息大家可以到 W3C 的相关网站获取：

程序清单 12.5:

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20000302 Stylable//EN"
"http://www.w3.org/TR/WD-SVG-20000302/DTD/svg-20000302-stylable.dtd">
<svg width="37mm" height="24mm">
<title>Sample SVG graphic</title>
<desc>A single rectangle with a white through yellow to red sunburst fill</desc>
<style type="text/css">
<![CDATA[.foo rect { fill:url(#burst);stroke:#FEFEFE } ]]>
</style>
<defs>
<radialGradient id="burst" cx="50" cy="80" r="90">
<stop offset="0" style="stop-color: #FFC"/>
<stop offset="0.2" style="stop-color: #FF3"/>
<stop offset="1" style="stop-color: #C33"/>
</radialGradient>
</defs>
<g class="foo">
<rect x="2mm" y="2mm" height="20mm" width="33mm"/>
</g>
</svg>
```

12.2.4 其它应用

XML 在其它方面应用包括 HDML, OEB, OSD, EDI 等很多，当然更包括 CXML（电子商务）的应用。

HDML 是一种为小显示屏的手持设备提供定义，类似于超文本内容的简单语言。同时，它也是一种通过 WWW 为移动通讯设备等手持设备提供服务的高效置标语言。HDML 的重点是内容的表现与布局。HDML 提供一种清晰的导航模式，该模式不依赖于 HTML 的可视内容。例如：HDML 提供一种高效的方式。通过这种方式，WWW 基础设施可以为手机、呼机、无线 PDA 等手持设备提供服务。

EDI（Electronic Data Interchange）使用电子技术代替基于纸张的操作手段，用于公司之间的单据交换。XML 的丰富格式语言可以用来描述不同类型的单据，例如信用证、贷款

申请表、保险单、索赔单以及各种发票等。结构化的 XML 送至 Web 的数据可以被加密，并且附上数字签名。XML 的安全性在 EDI 上能充分发挥。而 EDI 与 XML 的结合势必将推动电子商务的发展。

与 EDI 的命运相仿，软件包的上网发行一直处于试行阶段。OSD (Open Software Description) 是 XML 的一组用来描述各种软件产品的标记集，可以详细的说明软件的规格，使用以及运行情况。

XML 的应用不仅如此，XML 以把长远的目标放在工业组织上，一些着重于工业应用的初步尝试已经宣布了。比如 Open Buying On the Internet (OBI, 一个在 Internet 上进行国际性商业间购物的标准)，Open Trading Protocol (OTP, 一个在 Web 上向消费者售物的一致的、可共同操作的环境)，Internet Content and Exchange (ICE, 一个能在站点之间交换在线资产的环境)。

12.3 小 结

本章内容可能比较广泛，但是总体是承接前面几章的，介绍了 XML 在各个方面的应用，这些应用可能没有 WML、电子商务等的应用广泛，但正是细小的应用体现出 XML 受到广泛的重视和使用程度。另外，本章对 XML 的体系结构也作了必要的介绍。