

SAMS

北京科海培训中心

XML 揭秘

—入门·应用·精通

[美] Michael Morrison, et al. 著
陆新年 陆新宇 等译



清华大学出版社

清华大学出版社最新计算机图书推荐

计算机技术培训与辅导丛书:

| | |
|-----------------------------|---------|
| 计算机维护技术 (第二版) | 29.00 元 |
| PC 技术内幕 (第 8 版) | 56.00 元 |
| 电脑工程师维修教程 | 62.00 元 |
| 中文 Word 2000 培训教程 | 28.00 元 |
| 中文 Excel 2000 培训教程 | 26.00 元 |
| 中文 Access 97 培训教程 | 24.00 元 |
| Windows 98 学习捷径 (中文版) | 38.00 元 |
| Windows me 培训教程 | 28.00 元 |
| 中文 Windows 98 培训教程 | 28.00 元 |
| 中文 Office 97 培训教程 | 48.00 元 |
| 新编微机培训教程 (2000 版) | 24.00 元 |
| 新编微机培训教程 | 24.00 元 |
| 最新微机操作技术 (修订版) | 29.00 元 |
| 最新微机操作技术 | 28.00 元 |
| 图文双解 Office 2000 实用大全 (中文版) | 62.00 元 |

网页设计、网站建设培训系列教材:

| | |
|----------------------------------|---------|
| Dreamweaver UltraDev 4 培训教程 | 29.00 元 |
| ISDN 联网与应用指南 | 29.00 元 |
| Flash 5 网页设计自学通 | 20.00 元 |
| JSP 编程入门与实例 | 24.00 元 |
| 网页标题制作技巧与实例 | 22.00 元 |
| CSS & DHTML 网页制作特效与实例 | 32.00 元 |
| FrontPage 2000 & ASP 网页设计技巧与网页维护 | 56.00 元 |
| 动态网页设计 Flash 4 攻略手册 | 22.00 元 |
| 梦幻网页设计——Dreamweaver 3.0 培训教程 | 28.00 元 |
| 网页设计创意与精品赏析 (含光盘) | 50.00 元 |
| 用 Linux Free BSD 轻松构建网络世界 (含光盘) | 49.00 元 |
| 上网实用培训教程 | 24.00 元 |
| 学电脑不求人 | 18.00 元 |

ISBN 7-900635-28-9



9 787900 635280 >

读者联系电话: (010) 62562449 62589259
网址: www.khp.com.cn

定价: 69.50 元 (光盘)

封面设计: 付剑飞

SAMS

北京科海培训中心

XML 揭秘

——入门·应用·精通

[美] Michael Morrison, et al. 著

陆新年 陆新宇 等译

清华大学出版社

(京)新登字 158 号

著作权合同登记号:01-2000-3937

内 容 提 要

本书由在 Web 技术领域中名声显赫的 Michael Morrison 编写。该书全面介绍了 XML 的相关技术,并对 XML 深层次的技术给出宝贵而实用的建议,远见卓识地描述了 XML 用于 Web 信息结构化的光明前景,使你可以从中获取丰富的技巧和经验,理解和拓宽知识的深度,掌握 XML 的全部潜能。作者在本书的后半部分详细地讲述了 XML 在多个领域中的应用。为了帮助读者更好地理解书中的内容,作者提供了大量简明实用且又具代表性的示例。

本书内容丰富、由浅入深,既适合初学者学习掌握 XML 语言,也适合有一定基础的读者精通 XML。

XML Unleashed

Copyright ©2000 by Sams Publishing

本书中文简体字版由美国 SAMS 公司授权清华大学出版社和北京科海培训中心出版。未经出版者书面允许不得以任何方式复制或抄袭本书内容。

版权所有,盗版必究。

本书封面贴有清华大学出版社激光防伪标签,无标签者不得进入各书店。

出版者:清华大学出版社(北京清华大学校内,邮编 100084)

印刷者:北京门头沟胶印厂

发行者:新华书店总店北京科技发行所

开 本:787×1092 1/16 印张:42.375 字数:1030 千字

版 次:2001 年 6 月第 1 版 2001 年 6 月第 1 次印刷

印 数:0001~5000

盘 号:ISBN 7-900635-28-9

定 价:69.50 元(光盘)

本书的主要作者

Michael Morrison 不但是一位作家、程序员、玩具发明家,而且还写了很多书。这些书包括 Complete Idiot's Guide to Java 2 (Que Publishing, 1999), Java 1.1 Unleashed (Sams. net Publishing, 1997), How to Program JavaBeans (Ziff-Davis Press, 1997) 以及 Teach Yourself Internet Game Programming with Java in 21 Days (Sams. net Publishing, 1996)。Michael 是 Gas Hound Games 的富有创造力的领导者, Gas Hound Games 的网址是 <http://www.gashound.com>。Michael 还是基于 Web 的几门课程的教师,其中包括 DigitalThink 的“Introduction to Java 2”以及“JavaBeans for Programmers”等等。平时,Michael 会给好朋友 Sarah 和 Travis 上网球课,或是和朋友 Keith 和 Josh 玩曲棍球,或是和他的妻子 Mahsheed 看电影,其他的时间里,他会坐在池塘边幻想。你可以访问 Michael 的站点 <http://www.thetribe.com>。

本书的其他作者

Frank Boumphrey 是一位医药文件顾问,他预见到了用 XML 和微软的技术来编写 Internet 和 Intranet 应用的前景。他广泛地编写和推广 XML, ASP 和 VB, 同时,他还为 HTML 程序员协会 (HTML Writers Guild, HWG) 讲授 XML 和 ASP 的在线课程。他是 HWG Special Events 的副主席,同时他还是 XHTML 规范、扩展表单规范和 XHTML 模块化规范的编辑,这些规范都属于万维网联盟 (W3C)。

他有一个美满的婚姻,五个孩子和几只小猫。

David Brownell 最近在 JavaSoft 工作。在那里他提供了 Sun 的 XML 和 DOM 软件的原始版本, SSL 和公钥结构以及 JavaServer 分页技术。他还致力于 Web 服务器和 Java Web 服务器的 Java Servlet API 的开发。

在此之前,他已经和 Internet 技术打了十几年的交道。最初是在波士顿地区,后来他搬到了旧金山地区从事 CORBA 的开发。他是 CORBA 互操作性和 IIOP 规范的主要制订者之一。

目前他是一位独立的 Web 和 Internet 顾问。

简介

在 1996 年 2 月的一次采访中, Bill Gates 说“HTML 是我们的数据类型。”尽管这句话是在不朽的 Web 时代中产生的, 但是它的确说明了 XML(可扩展标记语言)与 HTML(超文本标记语言)的相关性。XML 这项技术为我们共同创建的, 但目前依然十分混乱的 HTML 世界提供了框架结构。我们知道, 计算机是用来存储和处理数据的, 而数据最终是通过 Web 来共享的。然而, 在我们为能够如此快速和高效地共享信息而兴奋不已的时候, 我们忽略了周密思考的必要性。让我们回到正题, Bill Gates 是错误的: HTML 实际上不是一种好的数据类型。幸运的是, 一些人在几年以前已认识到这个问题, 并致力于开发解决方案。如果现在让 Bill Gates 说那句话, 他会说“XML 是我们的数据类型。”

XML 被设计为一把“铲子”, 它可以把我们在这几年的 HTML 标记以及 Web 页面中的描述标记所造成的“洞”中挖出来。XML 为正文和软件开发提供了框架结构, 而这在过去一直是几乎没有的。为了实现这一目的, XML 使软件的能力有了突飞猛进的提高。XML 显著地改变着我们所熟知的 Web(如果你认为这些都仅仅是一些广告, 请参看本书所述的 XML 功能。本书的最后一部分, 说明了现实世界中都有哪些公司和个人正在使用 XML)。

你可能和大多数人一样, 并不十分了解 XML。一个准确的定义是: “XML 是设计标记语言的规范”。换句话说, XML 是用来描述 HTML 之类的标记语言的元语言。当前形式的 HTML 不是一种 XML 标记语言, 不过已经有将 HTML 进行略微修改以符合 XML 标准的计划被启动。

XML 的伟大之处并不在于你能够通过它来创建标记语言, 而在于你能够通过它来创建高度结构化的标记语言。XML 实际上仅仅是一种标准化了的可以在 Web 上表示结构化信息的文本格式, 事实上就是这么简单。然而, 当你为像 Web 这样的巨大的信息仓库提供结构的时候, 什么事情都可能发生。新的自动化可以通过将结构化的数据在应用程序之间通过 Web 来回传递来实现。搜索引擎突然变得聪明起来而且可以进行基于上下文而不是正文的搜索。至少 Web 上的数据变得可以自我描述, 这就使应用程序可以更充分地利用它们。

说实话, XML 对 Web 影响的程度尚未知晓, 但是如果你对使用一种全新的数据驱动的 Web 应用感兴趣, 你一定会十分激动。这本书将会包括你所需要了解的 XML 并在广泛的应用模式中使用它所需要的全部内容。下面是本书包括的主要内容:

- XML 现状
- 用 DTD 来模式化数据
- 用 XML 模式来模式化数据
- 通过 XML 模式来创建文件
- 将 DTD 转化为 XML 模式
- 定义名字空间
- 样式表基础
- XSL 和 CSS 的比较

- XML 到 HTML 的桥梁
- XML 解析器揭密
- 用 DOM 和 SAX API 解析 XML
- 浏览 XML
- XHTML:XML 和 HTML 结合的产物
- 把 HTML 文件转换为 XHTML
- XLink 揭密
- XPointer 揭密
- 编写 XML 脚本
- 使用 DSO
- 理解 XSL 模式
- 用 XML 实现 EDI
- XML 编写和正文管理工具
- 生成 XML 文件
- 用 XMLNews 创建新闻故事
- 用 SMIL 集成多媒体对象
- 用 CDF 创建通道
- 用 VML 和 SVG 描述向量图形
- 用 3DML 创建 3D 世界
- 用 MathML 创建数学公式
- 用 P3P 创建私有策略
- 用 RELML 列出财产
- 用 HRMML 管理人力资源
- 用 VoxML 创建语音应答应用

本书是如何组织的

本书被设计为随时可用的最全面的 XML 开发手册。我们尽量将 XML 非常广泛的内容组织为手册的形式。这本书被分为以下几部分：

- 第 1 部分：定义 XML 文件。介绍了 XML 数据的模式化以及它和 XML 文件的关系。我们详细探讨了 XML 数据的模式化，包括在创建有效的 XML 文件时数据模式化的重要意义。对 XML 数据模式化的两个主要实现方法：DTD 和 XML 模式也进行了详细的讨论。这一部分的最后讨论了名字空间，名字空间提供了在 XML 词汇中元素和属性名的唯一性。
- 第 2 部分：格式化 XML 文件。这一部分描述了样式表以及两种主要的用来格式化 XML 文件的样式表技术：Cascading Style Sheets (CSS, 层叠样式表) 和 eXtensible Style Language (XSL, 可扩展样式语言)。有趣的话题是关于哪种技术更好的大争论。这一部分还提供了用来说明各种样式表技术细节的实际的例子。
- 第 3 部分：XML 文件的处理。这一部分讨论的是大多数 XML 应用的关键功能。在

说明了 XML 处理的基础之后,这一部分介绍了用 Java 及 C++ 来解析 XML 的方法。同时还提供了对两种主流的 XML 处理 API——文件对象模型(DOM)和 XML 的简单 API(SAX)的详细讨论。

- 第 4 部分:XML 与 Web。这一部分主要讲述了 XML 是为 Web 而设计的这一事实。Web 技术主要决定于支持它的浏览器,所以这一部分主要讨论浏览器对 XML 的支持。你将学到许多 XHTML 的许多知识,XHTML 是 HTML 和 XML 的结合,它指明了 Web 的未来。对 XML 链接技术:XLink 和 XPointer 也进行了详细的讨论,同时讲述了如何用脚本语言来处理 XML。
- 第 5 部分:使用 XML 操作数据。这一部分讨论了 XML 数据访问。XML 的关键应用之一就是在数据库之间即时地交换数据。这一部分介绍了辅助 XML 这一功能的技术,诸如 XML 数据源对象(DSO)和 XML 查询语言(XQL)。你将会看到如何使用 XML 在 Active Server Pages 中访问数据库。最后讨论了 XML 怎样适应于 WIDL(Web 接口定义语言)和 EDI(电子数据交换)的商业数据交换。
- 第 6 部分:XML 工具。这一部分介绍了一些简化 XML 文件和应用创建过程的工具。你将会学习使用 XML 编写和正文管理工具,而不是那些用来实现非常特殊的 XML 任务的特殊工具。
- 第 7 部分:XML 词表探讨。这一部分实际上是本书最有趣的部分。因为这一部分探讨了在现实世界中有哪些公司和个人使用了 XML。这一部分非常实用(你将会看到几个现成的例子)而且涉及了令人难以置信的广泛课题——执行了诸如标出新闻故事、表示多媒体数据,构建 3D 世界、通过 Web 管理隐私、生成简历和求职广告,描述语音响应的应用等等功能的 XML 词汇。

谁应该读这本书

尽管这本书的读者可以是任何对学习 XML 感兴趣的人,但是它更适合于那些想要随时获取 XML 细节的 Web 开发者。如果读者曾对 XML 有所了解或者曾经创建过 XML 文件,那么这对于阅读本书是很有帮助的。千万要记住:XML 和 HTML 密切相关,所以如果你有 Web 页面编写经验,那么你能够很快学会 XML。问题是这本书涉及到了 XML 的使用。因而你可以立即着手用 XML 做一些有趣的事情。本书还涉及了一些非常实用的工业标准 XML 应用,它们可以被用来完成从构建语音响应应用到创建 3D 世界的任何工作。

如果你非常想钻研 XML 技术并了解它能够做什么,那么这本书正适合你。你最好是顺序地一直读下去,不过你也可以直接跳到你特别感兴趣的章节。最重要的是,我鼓励你保持良好的心情并享受发现 XML 技术力量的乐趣。

本书中的约定

本书遵守下列约定:占位符可以被替换为实际的文件名、参数,或者其他任何它所表示的元素。作为揭秘系列图书,本书还包含一些注释、提示以及警告,来帮助读者更快地发现重点和有用的信息。本书中所引用的所有代码都可以在随书的 CD-ROM 中找到。

告诉我们您的想法

作为本书的读者,您是本书最重要的批评者和评论者。我们非常重视您的观点并且希望了解我们哪些做得不错,哪些还能做得更好,您希望我们出版哪些内容,以及您所愿意给予我们的任何建议。

作为 Sams 的一位出版者,我非常欢迎您的建议。您可以发传真、电子邮件或者直接给我写信,让我知道这本书中您所喜欢的和不喜欢的地方,以及我们怎样才能把书写得更好。

请注意,我无法回答您对本书中的技术问题,这是由于我已经收到了很多信件,因而无法一一回答。

在您的来信中,请务必包括本书的标题和作者,以及您的名字和电话或传真。我会认真考虑您的建议并和本书的所有作者和编辑共享。

Fax: 317-581-4770
Email: office_sams@mcp.com
Mail: Publisher
Michael Stephens
Sams
201 West 103rd Street
Indianapolis, IN 46290 USA

目 录

第 1 部分 定义 XML 文件

| | |
|--|-------------|
| 第 1 章 XML 新手 | (1) |
| 1.1 XML 历史简介 | (1) |
| 1.2 XML 是怎样的 | (2) |
| 1.3 XML 语法基础 | (4) |
| 1.3.1 标记 | (6) |
| 1.3.2 实体引用 | (6) |
| 1.3.3 注释 | (7) |
| 1.3.4 处理指令 | (7) |
| 1.3.5 文件类型声明 | (8) |
| 1.3.6 CDATA 片段 | (8) |
| 1.4 XML 的现状 | (9) |
| 1.4.1 XML 的显示样式 | (9) |
| 1.4.2 浏览 XML | (9) |
| 1.4.3 解析 XML | (10) |
| 1.5 总结 | (11) |
| 第 2 章 数据模型:文件类型定义和 Schema | (12) |
| 2.1 XML 数据模型基础 | (12) |
| 2.2 用 DTD 来建立数据模型 | (15) |
| 2.3 用 XML Schema 建立数据模型 | (17) |
| 2.4 比较两种建立数据模型的方法 | (18) |
| 2.5 总结 | (20) |
| 第 3 章 DTD 基础 | (21) |
| 3.1 DTD 和文件结构 | (21) |
| 3.1.1 内部 DTD 和外部 DTD | (22) |
| 3.1.2 有效的和结构良好的文件 | (24) |
| 3.2 理解元素和属性 | (25) |
| 3.3 使用元素 | (27) |
| 3.3.1 空元素 | (27) |
| 3.3.2 只含子元素的元素 | (28) |
| 3.3.3 混合元素 | (29) |
| 3.3.4 ANY 元素 | (31) |
| 3.4 使用属性 | (31) |
| 3.4.1 字符串属性 | (32) |

| | |
|-------------------------------------|-------------|
| 3.4.2 枚举属性 | (33) |
| 3.4.3 标记属性 | (33) |
| 3.4.4 声明多重属性 | (34) |
| 3.5 用 DTD 创建有效的文件 | (34) |
| 3.6 总结 | (36) |
| 第 4 章 进一步深入 DTD | (37) |
| 4.1 实体和文件结构 | (37) |
| 4.2 字符和实体 | (38) |
| 4.3 使用实体 | (40) |
| 4.3.1 一般实体 | (41) |
| 4.3.2 参数实体 | (42) |
| 4.4 外部实体 | (43) |
| 4.4.1 引用非解析实体 | (43) |
| 4.4.2 使用外部参数实体 | (44) |
| 4.5 声明表示法 | (45) |
| 4.6 在文件中引用实体 | (46) |
| 4.7 使用条件段 | (48) |
| 4.8 总结 | (49) |
| 第 5 章 XML Schema 基础 | (50) |
| 5.1 了解 XML Schema | (50) |
| 5.1.1 XML Schema 和 W3C | (50) |
| 5.1.2 XML Schema 的好处 | (51) |
| 5.2 深入 XML Schema 词表 | (52) |
| 5.2.1 schema 元素 | (54) |
| 5.2.2 datatype 元素 | (55) |
| 5.2.3 ElementType 元素 | (55) |
| 5.2.4 element 元素 | (57) |
| 5.2.5 group 元素 | (58) |
| 5.2.6 AttributeType 元素 | (59) |
| 5.2.7 attribute 元素 | (60) |
| 5.2.8 description 元素 | (60) |
| 5.3 XML Schema 数据类型 | (61) |
| 5.4 从 XML Schema 创建文件 | (62) |
| 5.5 总结 | (64) |
| 第 6 章 XML Schema 构造技术 | (65) |
| 6.1 将 DTD 转换为 Schema | (65) |
| 6.2 根据 Schema 验证文件 | (69) |
| 6.3 用 XML Authority 生成 Schema | (70) |
| 6.4 理解内容模型 | (75) |
| 6.5 XML Schema 的未来 | (76) |
| 6.5.1 元素继承 | (76) |
| 6.5.2 受约束的数据类型 | (76) |

| | |
|---|--------------|
| 6.6 总结 | (77) |
| 第 7 章 使用 XML 名字空间 | (78) |
| 7.1 名字空间基础 | (78) |
| 7.2 声明名字空间 | (79) |
| 7.2.1 缺省声明 | (80) |
| 7.2.2 显式声明 | (80) |
| 7.3 将 Schema 作为名字空间引用 | (81) |
| 7.4 在 Schema 中使用名字空间 | (87) |
| 7.5 名字空间和文件对象模型 | (89) |
| 7.6 总结 | (90) |
| 第 2 部分 格式化 XML 文件 | |
| 第 8 章 对比两种样式规范:可扩展样式表语言和层叠样式表..... | (91) |
| 8.1 样式表基础 | (91) |
| 8.1.1 HTML 是一种非常糟糕的描述语言 | (91) |
| 8.1.2 将描述踢出 HTML | (92) |
| 8.1.3 通过样式表显示 XML | (92) |
| 8.2 CSS 和 XSL 发展史..... | (93) |
| 8.2.1 理解 CSS | (94) |
| 8.2.2 理解 XSL | (95) |
| 8.3 XSL 与 CSS 的比较..... | (95) |
| 8.4 XSL 和 CSS 联合使用..... | (96) |
| 8.5 总结 | (97) |
| 第 9 章 使用层叠样式表格式化 XML | (98) |
| 9.1 CSS 初步 | (98) |
| 9.2 深入 CSS 样式表 | (99) |
| 9.3 CSS 样式属性 | (101) |
| 9.3.1 display 属性 | (101) |
| 9.3.2 width 和 height 属性 | (102) |
| 9.3.3 border 属性 | (102) |
| 9.3.4 margin 属性 | (102) |
| 9.3.5 color 属性 | (103) |
| 9.3.6 text 属性 | (103) |
| 9.3.7 font 属性 | (103) |
| 9.4 创建 CSS 样式表 | (103) |
| 9.5 总结 | (111) |
| 第 10 章 理解 XSL | (112) |
| 10.1 处理 XSL 样式表 | (112) |
| 10.2 XSL 体系结构 | (113) |
| 10.2.1 XSL 转换 | (114) |

| | |
|--|--------------|
| 10.2.2 XPath | (114) |
| 10.2.3 XSL 格式化对象 | (114) |
| 10.3 总结 | (115) |
| 第 11 章 创建 XSL 样式表 | (116) |
| 11.1 XSL 与 Internet Explorer 5.0 | (116) |
| 11.2 深入 XSL 样式表 | (116) |
| 11.2.1 模板 | (117) |
| 11.2.2 模式 | (118) |
| 11.3 XSLT 模板结构 | (118) |
| 11.3.1 <code>xsl:value-of</code> 元素 | (119) |
| 11.3.2 <code>xsl:if</code> 元素 | (119) |
| 11.3.3 <code>xsl:for-each</code> 元素 | (119) |
| 11.3.4 <code>xsl:apply-templates</code> 元素 | (120) |
| 11.4 开发 XSL 样式表 | (120) |
| 11.5 总结 | (132) |
| 第 3 部分 XML 文件的处理 | |
| 第 12 章 XML 处理基础 | (133) |
| 12.1 XML 文件的处理 | (133) |
| 12.2 为什么要解析 XML | (134) |
| 12.3 为什么要验证 XML 文件 | (135) |
| 12.3.1 文件类 | (135) |
| 12.3.2 用 DTD 来验证内容 | (136) |
| 12.3.3 在 XML EDI 中使用 DTD | (137) |
| 12.3.4 用 DOM 来使用 DTD | (138) |
| 12.4 深入 XML 解析器 | (138) |
| 12.4.1 解析树 | (138) |
| 12.4.2 结构良好的解析器 | (140) |
| 12.4.3 验证解析器 | (142) |
| 12.5 解析 XML 的两种方案的比较 | (144) |
| 12.5.1 将文件作为树来解析 | (144) |
| 12.5.2 将文件作为平面数据结构来解析 | (147) |
| 12.6 作为对象的解析器 | (149) |
| 12.7 总结 | (149) |
| 第 13 章 用 Java 解析 XML | (151) |
| 13.1 Java 的 XML 解析器库剖析 | (151) |
| 13.1.1 核心功能 | (152) |
| 13.1.2 核心 API | (152) |
| 13.1.3 这个核心没有揭示 DTD | (152) |
| 13.2 市面上的工具 | (153) |
| 13.2.1 Sun 的 Java Project X package | (154) |

| | | |
|---------------|--------------------------------|--------------|
| 13.2.2 | IBM 的 XML4J v2 包 | (154) |
| 13.2.3 | Oracle 的 v2 XML 包 | (155) |
| 13.2.4 | DataChannel 的 XJParser 包 | (155) |
| 13.2.5 | Microstar 的 Aelfred 解析器 | (156) |
| 13.2.6 | James Clark 的 XP 解析器 | (156) |
| 13.2.7 | 其他 XML 解析器 | (156) |
| 13.3 | 用 Java 解析 XML 文件 | (158) |
| 13.3.1 | 使用 SAX 解析器 | (159) |
| 13.3.2 | 取得一个 DOM 文件 | (160) |
| 13.3.3 | 在解析之后:DOM 扩展 | (167) |
| 13.4 | 总结 | (172) |
| 第 14 章 | 用 C++ 解析 XML | (173) |
| 14.1 | 为什么用 C 编写解析器 | (173) |
| 14.2 | Expat 解析器 | (173) |
| 14.2.1 | 取得和使用 Expat | (174) |
| 14.2.2 | 用 Expat 解析 | (174) |
| 14.3 | IBM 的 C++ XML 解析器 | (175) |
| 14.3.1 | 取得和使用 IBM 解析器 | (175) |
| 14.3.2 | 用 IBM 解析器验证文件的有效性 | (176) |
| 14.4 | Microsoft 的 C 解析器 | (178) |
| 14.4.1 | Xmlint.exe 解析器 | (178) |
| 14.5 | MSXML 解析器 | (178) |
| 14.5.1 | 获得并使用解析器 | (180) |
| 14.5.2 | 实现解析器 | (180) |
| 14.5.3 | MS 解析器错误处理 | (181) |
| 14.5.4 | MS 解析器的选项和模式 | (182) |
| 14.5.5 | 用 IE5 打开 XML 文件 | (183) |
| 14.6 | 用 MSXML 构建解析器接口 | (184) |
| 14.6.1 | 在解析器中详述脚本 | (188) |
| 14.6.2 | 使用 MSXML 解析器解析和验证 XML 文件 | (189) |
| 14.7 | MSXML 解析器的错误 | (190) |
| 14.7.1 | 巨大 DTD 的不正确操作 | (190) |
| 14.7.2 | DOM 中的不正确语法 | (190) |
| 14.8 | 在文件和 DTD 设计中使用 C 解析器 | (190) |
| 14.8.1 | 设计文件类型 | (190) |
| 14.8.2 | 调试模板文件 | (191) |
| 14.8.3 | 检查 DTD 语法 | (191) |
| 14.8.4 | 使用 C++ 解析器调试 DTD | (191) |
| 14.8.5 | 解释错误信息 | (192) |
| 14.9 | 总结 | (192) |
| 第 15 章 | 使用文件对象模型 | (193) |
| 15.1 | W3C 和 XML DOM | (193) |

| | |
|--|-------|
| 15.1.1 DOM 的 API 接口 | (193) |
| 15.2 文件对象模型 | (194) |
| 15.3 文件树和解析树 | (195) |
| 15.4 W3C DOM 基础 | (196) |
| 15.4.1 DOM 对象 | (196) |
| 15.5 DOM 接口 | (196) |
| 15.6 节点和对象 | (197) |
| 15.7 访问 DOM 中的节点 | (198) |
| 15.7.1 遍历树 | (198) |
| 15.7.2 用名字来访问节点 | (198) |
| 15.8 DOM 方法返回的数据类型 | (198) |
| 15.9 Document 接口 | (200) |
| 15.9.1 Document 接口的只读属性 | (200) |
| 15.9.2 Implementation 属性 | (201) |
| 15.9.3 Document 接口的方法 | (201) |
| 15.9.4 使用 Node 和 nodeList 接口 | (202) |
| 15.9.5 使用 getElementsByTagName()方法 | (204) |
| 15.10 DocumentFragment 接口 | (204) |
| 15.11 node 接口 | (205) |
| 15.11.1 node 接口的属性 | (205) |
| 15.11.2 节点属性特征的调查 | (207) |
| 15.11.3 node 接口的方法 | (209) |
| 15.11.4 使用节点方法:示例 A | (211) |
| 15.11.5 使用节点方法:示例 B | (212) |
| 15.12 NodeList 接口 | (213) |
| 15.12.1 item()方法 | (213) |
| 15.13 NamedNodeMap 接口 | (213) |
| 15.13.1 getNamedItem(),setNamedItem()和 removeNamedItem()方法 | (213) |
| 15.13.2 length 属性 | (213) |
| 15.14 CharacterData 接口 | (215) |
| 15.14.1 data 属性 | (215) |
| 15.14.2 length 属性 | (215) |
| 15.14.3 subStringData 方法 | (215) |
| 15.14.4 appendData()方法 | (215) |
| 15.14.5 insertData()方法 | (215) |
| 15.14.6 deleteData()方法 | (216) |
| 15.14.7 replaceData()方法 | (216) |
| 15.15 Attr 接口 | (218) |
| 15.15.1 name 属性 | (218) |
| 15.15.2 value 属性 | (218) |
| 15.15.3 specified 属性 | (218) |
| 15.16 Element 接口 | (218) |
| 15.16.1 处理属性的方法 | (219) |

| | |
|---|--------------|
| 15.16.2 其他 element 接口的属性和方法 | (220) |
| 15.17 不常用的 DOM 接口 | (220) |
| 15.17.1 text 接口 | (220) |
| 15.17.2 CDATASection 接口 | (220) |
| 15.17.3 DocumentType 接口 | (220) |
| 15.17.4 Notation 接口 | (221) |
| 15.17.5 Entity 和 EntityReference 接口 | (222) |
| 15.17.6 ProcessingInstruction 接口 | (224) |
| 15.18 实现 DOM | (224) |
| 15.18.1 Microsoft 的 DOM 引擎 | (224) |
| 15.18.2 IBM 的 DOM 引擎 | (224) |
| 15.19 为应用程序加入 DOM 支持 | (225) |
| 15.19.1 Java | (225) |
| 15.19.2 C++ | (225) |
| 15.19.3 Visual Basic | (225) |
| 15.19.4 ASP | (225) |
| 15.20 脚本和 DOM | (225) |
| 15.20.1 使用 C++ DSO | (225) |
| 15.20.2 使用 Microsoft XMLDOM ActiveX 数据对象(ADO) | (226) |
| 15.21 Gecko 中的 DOM 支持 | (227) |
| 15.22 永久保存文件 | (227) |
| 15.23 总结 | (228) |
| 第 16 章 在 Java 中使用 SAX API | (229) |
| 16.1 SAX 1.0 的结构 | (229) |
| 16.2 SAX 1.0 包基础 | (230) |
| 16.2.1 解析 XML 文本 | (230) |
| 16.2.2 HandlerBase:统一的 SAX 处理函数 | (231) |
| 16.2.3 SAX DocumentHandler 接口 | (231) |
| 16.3 SAX 应用程序的例子 | (234) |
| 16.3.1 简单的模板处理 | (234) |
| 16.3.2 构建 DOM 树 | (236) |
| 16.4 SAX 的其他核心功能 | (238) |
| 16.4.1 使用 Parser.parse()的其他方法 | (238) |
| 16.4.2 ErrorHandler 对象 | (240) |
| 16.4.3 InputSource 对象 | (241) |
| 16.4.4 其他 DocumentHandler 回调 | (243) |
| 16.4.5 不是所有的解析器都一样 | (244) |
| 16.5 SAX 的高级特征 | (245) |
| 16.5.1 对外部解析实体使用 EntityResolver | (245) |
| 16.5.2 诊断 Locale | (247) |
| 16.5.3 对未解析实体和注释使用 DTD Handler | (248) |
| 16.6 SAX 2.0 | (249) |

| | |
|----------------------|-------|
| 16.6.1 解析器特征标志 | (250) |
| 16.6.2 解析器属性值 | (250) |
| 16.7 总结 | (251) |

第 4 部分 XML 与 Web

第 17 章 浏览 XML

(252)

| | |
|--|-------|
| 17.1 XML 和 Web 浏览器现状 | (252) |
| 17.2 Microsoft Internet Explorer | (253) |
| 17.2.1 XML 原始代码 | (253) |
| 17.2.2 XML 错误处理 | (253) |
| 17.2.3 CSS 和 XSL | (255) |
| 17.2.4 XML 名字空间 | (256) |
| 17.2.5 XML Schema | (256) |
| 17.2.6 CDF 和 VML | (257) |
| 17.3 Netscape Navigator(Mozilla) | (257) |
| 17.3.1 CSS 和 XSL | (258) |
| 17.3.2 Expat 解析器 | (258) |
| 17.3.3 XUL 和 RDF | (258) |
| 17.4 其他浏览器 | (259) |
| 17.4.1 CITEC DocZilla | (259) |
| 17.4.2 W3C Amaya | (259) |
| 17.4.3 Opera | (260) |
| 17.5 总结 | (260) |

第 18 章 XHTML:XML 与 HTML 结合的产物

(261)

| | |
|---------------------------------|-------|
| 18.1 为什么要用 XHTML | (261) |
| 18.2 XHTML 和 HTML 4.0 的差别 | (262) |
| 18.3 XHTML 和文件的有效性 | (263) |
| 18.3.1 声明 XHTML DTD | (264) |
| 18.3.2 声明 XHTML 名字空间 | (264) |
| 18.3.3 验证 XHTML 文件 | (264) |
| 18.4 创建 XHTML 文件 | (265) |
| 18.5 将 HTML 文件转化为 XHTML | (267) |
| 18.5.1 手工转换文件 | (268) |
| 18.5.2 使用 HTML 整理工具 | (271) |
| 18.6 总结 | (273) |

第 19 章 使用 XLink 和 XPointer 链接文件

(274)

| | |
|-------------------------|-------|
| 19.1 链接 HTML 的方法 | (274) |
| 19.2 超越传统 HTML 链接 | (275) |
| 19.3 W3C 链接规范 | (276) |
| 19.3.1 XPath | (277) |
| 19.3.2 XPointer | (278) |

| | | |
|---------------|-----------------------------|--------------|
| 19.3.3 | XLink | (278) |
| 19.4 | 深入 XLink | (279) |
| 19.4.1 | 链接的类别 | (280) |
| 19.4.2 | XLink 属性 | (280) |
| 19.4.3 | 创建 XLink | (281) |
| 19.5 | 深入 XPointer | (283) |
| 19.5.1 | 位置路径 | (283) |
| 19.5.2 | 创建 XPointer | (284) |
| 19.6 | 总结 | (285) |
| 第 20 章 | 编写 XML 脚本 | (286) |
| 20.1 | 为什么编写 XML 脚本 | (286) |
| 20.2 | XML 脚本选择 | (288) |
| 20.2.1 | JavaScript | (288) |
| 20.2.2 | VBScript | (289) |
| 20.2.3 | Perl | (289) |
| 20.2.4 | Python | (289) |
| 20.2.5 | AppleScript | (290) |
| 20.2.6 | Tcl | (290) |
| 20.3 | JavaScript 入门 | (290) |
| 20.3.1 | 语句 | (290) |
| 20.3.2 | 变量 | (291) |
| 20.3.3 | 表达式 | (291) |
| 20.3.4 | 注释 | (291) |
| 20.3.5 | 函数 | (291) |
| 20.3.6 | 对象 | (292) |
| 20.4 | 再谈 XML DOM | (292) |
| 20.4.1 | XMLDOMDocument 对象 | (293) |
| 20.4.2 | XMLDOMNode 对象 | (294) |
| 20.4.3 | XMLDOMNodeList 对象 | (295) |
| 20.4.4 | XMLDOMNamedNodeMap 对象 | (296) |
| 20.4.5 | XMLDOMElement 对象 | (296) |
| 20.4.6 | XMLDOMAttribute 对象 | (297) |
| 20.4.7 | XMLDOMText 对象 | (297) |
| 20.4.8 | XMLDOMParseError 对象 | (297) |
| 20.5 | 开发 XML 脚本 | (298) |
| 20.6 | 总结 | (307) |

第 5 部分 使用 XML 操作数据

| | | |
|---------------|------------------------|--------------|
| 第 21 章 | XML 数据源对象 | (309) |
| 21.1 | 数据库和远程数据访问 | (309) |
| 21.1.1 | 数据是如何组织的 | (309) |

| | | |
|---------------|------------------------------|--------------|
| 21.1.2 | 什么是记录集 | (310) |
| 21.1.3 | 浏览记录集 | (310) |
| 21.1.4 | 什么是 DSO | (311) |
| 21.1.5 | DSO 和 XML | (311) |
| 21.2 | 远程数据服务和 DSO | (312) |
| 21.2.1 | 表格式数据控制(TDC) | (312) |
| 21.2.2 | 远程数据服务(以前的 ADC) | (312) |
| 21.2.3 | JDBC DataSource Applet | (313) |
| 21.2.4 | XML 数据源 | (313) |
| 21.3 | 理解表格式数据控制(TDC) | (313) |
| 21.3.1 | 创建 TDC DSO | (313) |
| 21.3.2 | 导入文本文件 | (314) |
| 21.3.3 | TDC DSO 的数据绑定 | (314) |
| 21.3.4 | 使用脚本和 SQL 语法 | (317) |
| 21.3.5 | DSO 事件 | (317) |
| 21.3.6 | 数据与 HTML 表格(table)绑定 | (321) |
| 21.4 | XML DSO | (323) |
| 21.5 | XML Java DSO | (323) |
| 21.6 | XML C++ DSO | (323) |
| 21.6.1 | 实例化 C++ XML DSO | (323) |
| 21.6.2 | 加载扩展文件 | (324) |
| 21.6.3 | 验证 XML | (324) |
| 21.6.4 | 加载一个内嵌文件 | (325) |
| 21.7 | 使用 XML 岛 | (326) |
| 21.7.1 | 在 XML 岛中验证 XML | (326) |
| 21.7.2 | 加载嵌入 XML | (326) |
| 21.7.3 | 加载外部文件 | (327) |
| 21.8 | 编写 XML DSO | (327) |
| 21.8.1 | 使用 SQL | (327) |
| 21.8.2 | 使用 DOM | (328) |
| 21.9 | XML 数据绑定 | (328) |
| 21.10 | 复杂 XML 的数据绑定 | (328) |
| 21.10.1 | 绑定属性 | (329) |
| 21.10.2 | 绑定嵌套的元素 | (330) |
| 21.11 | 总结 | (332) |
| 第 22 章 | 使用 XSL 模式和 XQL | (333) |
| 22.1 | 查询语言的需求 | (333) |
| 22.1.1 | 为什么要有 XML 查询语言 | (333) |
| 22.1.2 | 作为数据存储的 XML | (333) |
| 22.2 | XML 查询语言(XQL) | (334) |
| 22.2.1 | 什么是 XQL 查询 | (335) |
| 22.2.2 | XQL 查询样例 | (335) |

| | | |
|---------------|----------------------------|--------------|
| 22.3 | XQL 语法 | (336) |
| 22.3.1 | 层叠查询字符串 | (337) |
| 22.3.2 | 连续关系查询 | (338) |
| 22.3.3 | 查询以找到属性 | (338) |
| 22.3.4 | 处理空白空间 | (339) |
| 22.3.5 | 使用过滤器 | (339) |
| 22.3.6 | 布尔语句 | (340) |
| 22.3.7 | 等式 | (340) |
| 22.3.8 | 比较 | (340) |
| 22.3.9 | XQL 方法 | (340) |
| 22.3.10 | 在查询中使用圆括号 | (342) |
| 22.3.11 | 名字空间查询方法 | (343) |
| 22.3.12 | 聚合方法 | (343) |
| 22.4 | XQL 执行工具和其他资源 | (344) |
| 22.5 | XSLT 模式 | (344) |
| 22.5.1 | XSLT 查询的例子 | (344) |
| 22.6 | 使用 DOM | (346) |
| 22.6.1 | 何时使用 DOM 查询 | (348) |
| 22.7 | 自编函数 | (348) |
| 22.7.1 | 什么时候使用自编函数查询 | (350) |
| 22.8 | 总结 | (350) |
| 第 23 章 | 使用 XML 和 ASP 访问数据库 | (351) |
| 23.1 | 在数据库管理系统中使用 XML | (351) |
| 23.1.1 | XML 用于归档 | (351) |
| 23.1.2 | XML 作为包 | (351) |
| 23.1.3 | XML 用于显示 | (352) |
| 23.1.4 | XML 作为数据存储 | (352) |
| 23.2 | 个人万维网服务器(PWS)和动态服务器网页(ASP) | (352) |
| 23.2.1 | 使用 PWS | (353) |
| 23.2.2 | 理解 ASP | (353) |
| 23.3 | 本机的 ASP 对象 | (355) |
| 23.3.1 | server 对象 | (356) |
| 23.3.2 | request 对象 | (356) |
| 23.3.3 | response 对象 | (357) |
| 23.4 | 脚本对象 | (357) |
| 23.4.1 | FileSystemObject 对象 | (357) |
| 23.4.2 | TextStream 对象 | (358) |
| 23.5 | ActiveX 和 DOM 对象 | (359) |
| 23.5.1 | ADODB 对象 | (360) |
| 23.5.2 | XML DOM 对象 | (360) |
| 23.6 | 理解数据库 | (360) |
| 23.7 | 使用 ASP 连接数据库 | (361) |

| | |
|------------------------------------|--------------|
| 23.7.1 ODBC 和 OLE-DB | (361) |
| 23.7.2 访问数据库 | (361) |
| 23.8 将数据读入 XML 流 | (365) |
| 23.8.1 扩展 connect2.asp | (368) |
| 23.8.2 使用 XML 备份记录 | (368) |
| 23.8.3 使用 XML 进行信息封装 | (368) |
| 23.9 使用 XML 作为数据存储 | (369) |
| 23.9.1 使用 ASP 访问 XML 文件 | (369) |
| 23.9.2 以简单结构文本文件获得 XML 文件 | (369) |
| 23.9.3 以 DOM 对象方式获取 XML | (371) |
| 23.10 从 XML 数据存储中访问数据 | (372) |
| 23.11 XML 与 RDBMS 对比 | (373) |
| 23.11.1 什么时候使用 XML 作为数据库的附属 | (374) |
| 23.11.2 什么时候将 XML 作为数据存储 | (374) |
| 23.11.3 什么时候不使用 XML 作为数据存储 | (374) |
| 23.12 总结 | (374) |
| 第 24 章 使用 WIDL 链接商用数据 | (376) |
| 24.1 商业上使用 WIDL | (376) |
| 24.1.1 自动数据收集 | (378) |
| 24.1.2 使用 WIDL 作为解决方案 | (380) |
| 24.2 WIDL 2.0 简介 | (383) |
| 24.2.1 对象模型 | (384) |
| 24.2.2 WIDL 元素 | (384) |
| 24.2.3 SERVICE 元素 | (385) |
| 24.2.4 SERVICE 链 | (385) |
| 24.2.5 BINDING 元素 | (386) |
| 24.2.6 VARIABLE 元素 | (386) |
| 24.2.7 CONDITION 元素 | (387) |
| 24.2.8 REGION 元素 | (387) |
| 24.3 WIDL 3.0 | (388) |
| 24.4 总结 | (388) |
| 第 25 章 EDI 和 XML | (389) |
| 25.1 EDI 初步 | (389) |
| 25.1.1 历史与现状 | (389) |
| 25.1.2 EDI 的商业特征 | (390) |
| 25.1.3 技术描述 | (390) |
| 25.2 EDI 有什么问题 | (391) |
| 25.2.1 商业问题 | (391) |
| 25.2.2 技术问题 | (392) |
| 25.3 与 EDI 接近的 XML | (393) |
| 25.3.1 基础技术 | (393) |
| 25.3.2 网上电子商务 | (396) |

| | |
|--------------------------------------|-------|
| 25.3.3 XML 中的 EDI 消息——XML/EDIT | (397) |
| 25.4 总结 | (398) |

第 6 部分 XML 工具

| | |
|-------------------------------------|--------------|
| 第 26 章 XML 制作和内容管理工具 | (399) |
| 26.1 XML 工具类型 | (399) |
| 26.2 XML 制作工具 | (400) |
| 26.2.1 SoftQuad XMetaL | (400) |
| 26.2.2 Adobe FrameMaker+SGML | (401) |
| 26.2.3 Arbortext ADEPT-Editor | (401) |
| 26.2.4 Stilo WebWriter | (402) |
| 26.2.5 IBM Xena | (402) |
| 26.2.6 Vervet Logic XML Pro | (402) |
| 26.2.7 Bluestone Visual-XML | (403) |
| 26.2.8 Microsoft XML Notepad | (403) |
| 26.2.9 Emile | (403) |
| 26.3 XML 内容管理工具 | (404) |
| 26.3.1 Poet 内容管理套件 | (404) |
| 26.3.2 Arbortext Epic | (404) |
| 26.3.3 Chrystal Astoria | (405) |
| 26.3.4 Oracle 8i | (405) |
| 26.4 其他 XML 工具 | (405) |
| 26.4.1 XML Authority | (406) |
| 26.4.2 Near and Far Designer | (406) |
| 26.4.3 Stylus | (406) |
| 26.5 总结 | (407) |
| 第 27 章 使用 XML 生成器移植数据 | (408) |
| 27.1 XML 生成器基础 | (408) |
| 27.2 使用 XML 生成器应用程序 | (409) |
| 27.2.1 分隔的文本数据文件 | (409) |
| 27.2.2 XML 模板 | (410) |
| 27.3 生成 XML 文件 | (411) |
| 27.4 总结 | (419) |
| 第 28 章 XFA 脚本系统 | (420) |
| 28.1 XFA 脚本基础 | (420) |
| 28.1.1 解释 XFA 脚本 | (420) |
| 28.2 XFA 和 XML 的关系 | (421) |
| 28.3 XFA 数据类型 | (422) |
| 28.4 XFA 词表 | (423) |
| 28.4.1 xfa:val 元素 | (424) |
| 28.4.2 xfa:block 元素 | (424) |

| | | |
|------------------------------------|-------------------------------------|--------------|
| 28.4.3 | xfa:function 元素 | (425) |
| 28.4.4 | xfa:use 元素 | (425) |
| 28.4.5 | xfa;if、xfa:orif 和 xfa:else 元素 | (425) |
| 28.4.6 | xfa:for 元素 | (426) |
| 28.4.7 | xfa:while 元素 | (426) |
| 28.4.8 | xfa:string 元素 | (426) |
| 28.4.9 | xfa:break 元素 | (427) |
| 28.4.10 | xfa:let 元素 | (427) |
| 28.4.11 | xfa:data 元素 | (428) |
| 28.4.12 | xfa:tag 元素 | (428) |
| 28.4.13 | xfa:ref 元素 | (428) |
| 28.4.14 | xfa:form 元素 | (428) |
| 28.4.15 | xfa:note 元素 | (430) |
| 28.5 | 编写 XFA 脚本 | (431) |
| 28.5.1 | 从文本文件中读取 | (431) |
| 28.5.2 | 处理 XML 代码 | (431) |
| 28.6 | XFA 和 DTD | (435) |
| 28.7 | 总结 | (440) |
| 第 29 章 | 使用 DDbE 生成 DTD | (441) |
| 29.1 | 理解 DDbE | (441) |
| 29.2 | 深入 DDbE 命令行工具 | (442) |
| 29.3 | 生成 DTD | (443) |
| 29.4 | 使用 DDbE 库 | (448) |
| 29.5 | 总结 | (449) |
| 第 30 章 | 了解 IBM 的 XML 工具集 | (450) |
| 30.1 | 工具箱一瞥 | (450) |
| 30.2 | 使用 Xplorer 和 XML 浏览器 | (451) |
| 30.3 | 使用 XML 转换生成器 | (456) |
| 30.3.1 | 转换 XML 文件 | (456) |
| 30.3.2 | 从网页中提取数据 | (459) |
| 30.4 | 使用 XML 区分和归并工具 | (463) |
| 30.5 | 总结 | (466) |
| 第 7 部分 XML 词表探讨 | | |
| 第 31 章 | 用 XMLNEWS 制作新闻 | (467) |
| 31.1 | 理解 XMLNews | (467) |
| 31.2 | 深入 XMLNews-Story 词表 | (468) |
| 31.2.1 | ntif 元素 | (473) |
| 31.2.2 | body.head 元素 | (474) |
| 31.2.3 | body.content 元素 | (475) |
| 31.3 | 使用 XMLNews 创建新闻故事 | (477) |

| | |
|---|--------------|
| 31.4 总结 | (481) |
| 第 32 章 使用 SMIL 集成多媒体对象 | (482) |
| 32.1 SMIL 基础 | (482) |
| 32.2 深入 SMIL 词表 | (483) |
| 32.2.1 smil 元素 | (487) |
| 32.2.2 head 元素 | (487) |
| 32.2.3 body 元素 | (488) |
| 32.3 创建 SMIL 内容 | (491) |
| 32.4 SMIL 播放器和制作工具 | (494) |
| 32.4.1 SMIL 播放器 | (494) |
| 32.4.2 SMIL 制作工具 | (498) |
| 32.5 SMIL 和 HTML+TIME | (501) |
| 32.6 总结 | (502) |
| 第 33 章 使用 CDF 推出 Web 内容 | (503) |
| 33.1 Active 频道基础 | (503) |
| 33.2 使用 Active 频道 | (504) |
| 33.3 深入 CDF 词表 | (509) |
| 33.3.1 Channel 元素 | (511) |
| 33.3.2 Title 元素 | (512) |
| 33.3.3 Abstract 元素 | (512) |
| 33.3.4 Author 元素 | (512) |
| 33.3.5 Publisher 元素 | (512) |
| 33.3.6 Copyright 元素 | (512) |
| 33.3.7 PublicationDate 元素 | (513) |
| 33.3.8 LastMod 元素 | (513) |
| 33.3.9 Schedule 元素 | (513) |
| 33.3.10 Logo 元素 | (514) |
| 33.4 使用 CDF 创建频道 | (514) |
| 33.5 CDF 生成器工具 | (515) |
| 33.6 总结 | (519) |
| 第 34 章 使用 VML 和 SVG 描述矢量图形 | (520) |
| 34.1 结构化矢量图形的重要性 | (520) |
| 34.2 XML 矢量图形支持 | (522) |
| 34.2.1 矢量标记语言(VML) | (522) |
| 34.2.2 精确矢量图形(SVG) | (523) |
| 34.3 深入 VML 词表 | (524) |
| 34.3.1 shape 元素 | (525) |
| 34.3.2 path 元素 | (526) |
| 34.3.3 line 元素 | (527) |
| 34.3.4 polyline 元素 | (529) |
| 34.3.5 curve 元素 | (529) |
| 34.3.6 rect 元素 | (530) |

| | | |
|---------------|---|--------------|
| 34.3.7 | roundrect 元素 | (531) |
| 34.3.8 | oval 元素 | (531) |
| 34.3.9 | arc 元素 | (532) |
| 34.3.10 | group 元素 | (533) |
| 34.4 | 深入 SVG 词表 | (535) |
| 34.5 | 总结 | (541) |
| 第 35 章 | 虚拟现实和 3DML | (543) |
| 35.1 | DML 基础 | (543) |
| 35.2 | 深入 3DML 词表 | (544) |
| 35.2.1 | SPOT 元素 | (545) |
| 35.2.2 | HEAD 元素 | (545) |
| 35.2.3 | BODY 元素 | (549) |
| 35.3 | 创建 3DML 世界 | (554) |
| 35.4 | 在网页中嵌入 3DML 世界 | (556) |
| 35.5 | Spotnik 3DML 场景制作工具 | (558) |
| 35.6 | 总结 | (560) |
| 第 36 章 | 用 MathML 表达数学 | (561) |
| 36.1 | MathML 基础 | (561) |
| 36.2 | 深入 MathML 词表 | (562) |
| 36.2.1 | 表述标记 | (564) |
| 36.2.2 | 内容标记 | (565) |
| 36.3 | 创建 MathML 内容 | (568) |
| 36.4 | MathML 工具 | (571) |
| 36.4.1 | Amaya 编辑器和浏览器 | (571) |
| 36.4.2 | WebEQ 数学浏览器 Java Applet | (571) |
| 36.4.3 | IBM 的 techexplorer Hypermedia 浏览器 | (574) |
| 36.4.4 | MathType 等式编辑器 | (574) |
| 36.4.5 | EzMath 编辑器和插件 | (575) |
| 36.4.6 | Mathematica 科学计算工具 | (576) |
| 36.5 | 总结 | (576) |
| 第 37 章 | 使用 P3P 管理个人隐私信息 | (577) |
| 37.1 | P3P 基础 | (577) |
| 37.2 | RDF 速成 | (578) |
| 37.3 | 深入 P3P 协调词表 | (579) |
| 37.3.1 | PROP 元素 | (579) |
| 37.3.2 | ASSURANCE 元素 | (579) |
| 37.3.3 | REALM 元素 | (580) |
| 37.3.4 | VOC;DISCLOSURE 元素 | (580) |
| 37.3.5 | STATEMENT 元素 | (580) |
| 37.3.6 | DATA;REF 元素 | (581) |
| 37.4 | 使用 P3P 创建个人策略 | (583) |
| 37.5 | Privacy Wizard | (584) |

| | |
|--------------------------------------|--------------|
| 37.6 总结 | (589) |
| 第 38 章 使用 RELML 列出实际不动产 | (591) |
| 38.1 RELML 基础 | (591) |
| 38.2 深入 RELML 词表 | (592) |
| 38.3 创建 RELML 内容 | (596) |
| 38.4 RELML 和 OpenMLS | (599) |
| 38.5 总结 | (603) |
| 第 39 章 使用 HRMML 管理人才资源 | (604) |
| 39.1 HRMML 基础 | (604) |
| 39.2 深入 HRMML 词表 | (605) |
| 39.2.1 hr-comm.mod 模块 | (607) |
| 39.2.2 hr-org.mod 模块 | (609) |
| 39.2.3 hr-pstd.mod 模块 | (610) |
| 39.2.4 hr-jobd.mod 模块 | (611) |
| 39.2.5 hr-phon.mod 模块 | (613) |
| 39.2.6 hr-addr.mod 模块 | (614) |
| 39.2.7 hr-resm.mod 模块 | (614) |
| 39.3 评价 HRMML 履历结构 | (616) |
| 39.3.1 resume prolog 元素 | (617) |
| 39.3.2 resume body 元素 | (618) |
| 39.4 创建 HRMML 履历 | (620) |
| 39.5 浏览 HRMML 履历 | (623) |
| 39.6 总结 | (628) |
| 第 40 章 交互谈话和 VoxML | (629) |
| 40.1 VoxML 基础 | (629) |
| 40.2 深入 VoxML 词表 | (630) |
| 40.2.1 DIALOG 元素 | (632) |
| 40.2.2 STEP 元素 | (633) |
| 40.3 VoxML 工具 | (636) |
| 40.3.1 VoxML 模拟器 | (636) |
| 40.3.2 VoxML 浏览器 | (636) |
| 40.3.3 VoxML 开发节点 | (637) |
| 40.4 使用 VoxML 创建语音应用 | (637) |
| 40.5 总结 | (642) |

第 8 部分 附 录

| | |
|---------------------------|--------------|
| 附录 A XML 缩略词 | (644) |
| 附录 B XML 资源 | (647) |
| B.1 一般 XML 资源 | (647) |
| B.2 XML 规范 | (647) |

| | |
|--------------------|-------|
| B.3 XML 浏览器 | (648) |
| B.4 XML 工具 | (648) |
| B.4.1 验证服务 | (649) |
| B.4.2 解析器 | (649) |
| B.4.3 制作工具 | (649) |
| B.4.4 内容管理工具 | (650) |
| B.4.5 其他工具 | (650) |
| B.5 XML 词表 | (650) |
| B.6 其他资源 | (651) |

第1部分 定义XML文件

第1章 XML 新手

本章是为帮助那些 XML 新手学习本书的其他部分而写的。如果你已经是一个经验丰富的 XML 高手,你可以轻松地略过这一章。然而值得注意的是,本章讲述了一些不错的 XML 语法,很值得你再看一下以便加深理解。如果你是一个 XML 新手,那么你一定要认真阅读本章,因为本章列出了 XML 语法的基本规则,而这正是整个 XML 技术的基础。

1.1 XML 历史简介

你可能会对 XML 的历史有点了解,对 XML 和早期的信息技术比如 SGML 等之间的联系也有所了解。然而,我认为快速浏览一下 XML 的发展历程对于我们正确理解这一技术是很有帮助的。同时,这还会帮助你理解那些创建了 XML 的人是怎样看待 Web 的发展现状的,他们的观点可能和广大 Web 用户在很多方面都有所不同。XML 的创建者们把 Web 看作是需要整理的无结构的数据仓库,这种观点使他们考虑如何为 Web 上的信息添加上下文并使这些信息更结构化。

早在 20 世纪 60 年代,人们就想要将文件结构化成标准的格式,以促进数据的交换和操作。IBM 创建了 GML (Generalized Markup Language, 通用标记语言) 以在其出版系统内部实现这一需求。IBM 使用 GML 处理来自单一类型源文件的书籍、报表以及其他文件。信息结构化的其他解决方案也由某些组织和公司提出,但是整体上解决这个问题的工作却毫无进展。

SGML (Standard Generalized Markup Language, 标准通用标记语言) 是第一个标准化的信息结构化技术,它也是从 IBM 的 GML 演化而来的。SGML 被用来提供一种在 IBM 内部格式化和维护合法文件的手段。SGML 后来被扩展和修改,作为一种全面的信息标准以适应工业范围的广泛应用。但直到 1986 年 SGML 才成为 ISO 标准。尽管 SGML 的功能极其强大,然而它也非常复杂而且需要一大堆昂贵的软件来配合运行。正是由于 SGML 太复杂和太昂贵了,在 Internet 的初期,它显然不是表示超文本的最佳选择。

在 1989 年,Particle Physics 欧洲实验室 (CERN) 的研究员 Tim Berners-Lee 和 Anders Berglund 两人创建了一种基于标记的语言来给在 Internet 上共享的技术文件做标记。这种语言最终发展成为一种简化了的 SGML 应用,这就是 HTML。现在 HTML 已经成为 Web 上信息的标准格式。相信你已经了解 HTML 接下来的故事了,这里我就不赘述了。只要看看许多人挣到了那么多钱,计算机界引起了所有人的注意,而现在我们所有的宠物照片都可以很容易地与他人共享就足够了。对了,还有以前那些对计算机一无所知的人都很快地学会

了如何将字母 w 重复三次,即 WWW。

事实证明 HTML 获得了巨大成功。而且每一个人都急于上网冲浪。随着新奇感的退去,厂家和个人都开始不满足于静态的图片和文本。接下来就是导致用来为 Web 添加交互性的技术令人眩目的成功。在这些技术之中,有一些技术,比如 Java,现在仍然被广泛使用,而更多的其他技术已经渐渐消失了。除了 Java 之类的交互技术之外,HTML 本身很快也得到了系列改进。我称之为“改进”,是因为尽管 HTML 加入了许多新的很好的特征,但是 HTML 也开始变得混乱和庞大。甚至,HTML 已经被延伸和扭曲,以便实现一些本来根本不应由 HTML 完成的功能。

请记住,HTML 最初仅仅是被当作一种表示静态信息以便显示的方法来设计的。为 HTML 添加交互能力,希望它可以表示动态信息,使 HTML 陷于改变它自身特性的境地。比将 HTML 从静态改为动态更糟糕的是,人们希望 HTML 能存储金融交易、产品目录、简历之类的特定类型数据。Web 开发者很快就认识到将 Web 页面和后台数据库相连的好处,这时,HTML 又成为了一种数据库接口。HTML 也因此从一种基于表示的信息标记语言发展成为一种全能的软件技术。完全由 HTML 实现的一些应用程序也已出现,当然还要加上各种标记语言、Java applet、plug-in(插件)以及其他相关技术。

千万不要误解我的意思——尽管有着先天的限制,HTML 仍然令人惊讶的满足了 Web 开发者的需求。但是,W3C(World Wide Web Consortium,万维网联盟)的一些成员认识到制订一种比 HTML 更具备可扩展性的语言来适应未来 Web 发展需求的必要性。HTML 最明显的一个限制可能就是它固定的标记集合。Web 开发者为 HTML 添加新标记,这是非常重要的,这对于处理属于特定应用或工业的数据非常有用。比如,一个书商可能更多的使用<title>、<author>和<isbn>这样的标记而不是通用的标记<p>。

当然,SGML 完全支持自定义标记功能。不过……

到 1996 年,W3C 开始寻找一种在 Web 中使用 SGML 的灵活性和强大功能的方法。正如我们看到的,SGML 具备 HTML 所没有的三种优势:可扩展性、结构化和有效性。技术上,HTML 也有结构,但是大多数浏览器不可能对你是否遵守这种结构考虑太多。SGML 比 HTML 更结构化,而 SGML 的软件对结构的要求也明显比 Web 浏览器严格得多。因此,从实践上讲,HTML 是松散结构化的语言。

W3C 组织了一个 SGML 专家组,他们的主要目标就是创建一种新的标记技术,这种技术既要具备 SGML 的核心特征,还要具备 HTML 的简单性。“为 Web 设计的 SGML”专家组努力而有些平静地工作以创建他们的标记技术,直到 1998 年 2 月,W3C 发布了 XML 1.0 规范。尽管还提出了一些别的名字,但最终这种语言被命名为 XML,即可扩展的标记语言(eXtensible Markup Language)。那么,XML 究竟是怎样的呢?

1.2 XML 是怎样的

XML 是 SGML 简化了的子集。它继承了 SGML 的许多特征,包括其中重要的三个:可扩展性、结构化和有效性。某种程度上,XML 开创了 Web 的新纪元,它建立了一种传输结构化数据的方法。XML 实际上不过是 Web 上表示结构化信息的一种标准文本格式。我希望能把 XML 描述得尽量简单,因为一些像文本格式这样的看上去简单的事可能在软件界产生

巨大的影响。

注意:XML 规范不到 SGML 规范的十分之一,这也说明了对 SGML 精减的程度。

XML 不仅仅是 HTML 的扩展版本,事实上,XML 根本不是 HTML 的版本。HTML 是一种用来对用 Web 浏览器表示的信息进行编码的特定标记语言,而 XML 是一个设计标记语言的 Schema。换句话说,XML 是一种元语言。这意味着作为 SGML 特殊应用的 HTML 可以被重新设计成 XML 的应用。事实上,类似的 XML 应用正在设计之中,这就是 XHTML,我们将要在第 18 章“XHTML:XML 与 HTML 结合的产物”中介绍。

对于那些只把 Web 当作图形设计的展示橱窗的 Web 开发者来说,XML 初看起来可能有点多余。尽管对于任何 Web 站点来说描述都起着重要的作用,现代的 Web 应用是由某种特定类型的数据驱动的。正如你所知道的那样,HTML 表示这种数据的能力很弱。而 XML 则具有客户定制标记词表,因此 XML 使得描述数据以及数据片之间的关系成为可能。事实上,XML 的主要目标之一就是 will 将 Web 文件的内容(数据)和描述(数据的表示形式)分割开来。

XML 关注于内容,因此它提供了一种在 Web 文件中包含元数据(metadata)的方法。元数据是关于信息的信息,这听起来的确有点奇怪。事实上,这是一个很容易理解的概念,只要我们想一想实际的术语就可以理解了。

最近我想了解曲棍球设备市场,所以我在网上使用“Warp”来查找一种设备的生产线。而使用“Warp”进行搜索所得到的结果中有一大堆是关于 OS/2 Warp 操作系统的。于是我用“Warp”和“hockey”两个词进行搜索,我得到了一个包含 arcade 游戏清单的页面,包括一个叫做“Warp”的游戏和另外一个叫做“Wayne Gretzky's 3D Hockey”游戏的人。很显然,我很快就无所适从了。

问题在于当需要 HTML 对 Web 页面上的信息进行分类搜索的时候,对这种搜索 HTML 是非常弱的。而元数据正是这个问题的一个解决方案。

理想情况下,每一个人都在他的 XML Web 文件中使用元数据,我可能就会通过声明“Warp”是曲棍球设备生产线或者是体育设备生产线来对我的搜索加以限制。含有这种设备生产线信息的曲棍球站点会用特定的标记来指出“Warp”实际上指的是一种曲棍球设备或运动健身设备。我的搜索也因此拥有了上下文,从而使返回的结果不但含有“Warp”这个词而且还要属于曲棍球设备这一类。很明显,XML 使搜索技术的前途一片光明。

你可能会奇怪为什么引用 XML 内容时,我使用“文件”这个词而不是“页面”。XML 要求你用一种与构成 Web 完全不同的观点来考虑问题。你不再将 Web 看作是一大堆相互链接的页面,相反,你应该把它看成一系列相互链接的文件。我想这之间的区别的确有点微妙,但是在很大程度上这揭示了你对 Web 内容的理解。文件可以是简历到股市报价到数据库记录之类的各种东西,而页面本来是可见的。用文件的观点来看 Web 的内容的确是向理解整个 XML 迈出的重要的一步。

说到整个 XML,我最初学习 XML 的时候所遇到的一个问题就是不明白要如何利用它。XML 被描述成一种如此通用的技术,以至于好像你可以用它来做任何事情。与 HTML 不同,XML 本身并不是一个解决方案。XML 定义了一个框架结构,你可以用它来创建解决方案,但是单独的 XML 本身并做不了什么。由于 XML 的承诺是创建定制的标记集合以对

特定类型的信息进行编码,因此也就没有一种通用的XML浏览器,而Web浏览器就是一种HTML浏览器。是的,实际上也有通用的XML浏览器,但是它们仅仅允许你浏览XML标记的结构,而不是XML内容实际意义的显示。

为了用一种有意义方式浏览XML数据,你必须描述如何表示信息。请记住,XML关心的是结构化内容而不是显示。XML文件的显示通常要通过样式表来实现,可以使用XSL(eXtensible Stylesheet Language,可扩展的样式表语言)或者CSS(Cascading Style Sheets,层叠式样式表)。目前,Internet Explorer 5允许你使用XSL或者CSS来为XML提供显示格式。需要指出的是Microsoft对XSL和CSS的实现在很多方面都是有限的,它并没有严格遵守W3C两种技术的标准。

1.3 XML语法基础

XML实际上就是一种规范,所以正确掌握XML语法是非常重要的。如果你熟悉SGML,你会看到在XML语法和SGML语法之间存在很多相同点。如果你不熟悉也没关系,你可以将XML语法和HTML语法联系起来。在XML世界里,HTML只能说是一个特定的XML词表,所以XML语法肯定能够完全描述HTML。

XML文件的基本构成单元是实体(entity),它包含解析的或未解析的数据。解析的数据由字符构成,它可以被看作字符数据或者标记,并由XML处理函数处理。我们很快就会学习它。未解析的字符数据被当作原始文本,而不被处理。下面的例子就是一个解析数据,<name>和</name>是标记,而Maximillian是字符数据。

```
<name>Maximillian</name>
```

标记被用来描述文件的存储结构(实体)和逻辑结构(元素)。XML允许你通过指定标记组件之间的关系来对文件布局和结构进行约束。XML语法实质上描述了用来定义XML文件的结构和布局的构造,当然也就包含了约束关系。请记住XML文件需要XML处理函数来处理,因此XML文件必须遵守严格的语法。

XML处理函数是一种读入XML文件并提供访问其内容和结构的软件模块。XML处理函数通常根据应用的需要来处理XML文件。换句话说,XML应用通过XML处理函数来获得对XML内容和结构的访问。对于用户来说,XML应用和XML处理函数实际上可能并没有区别。但是对于XML应用开发者来说,两者的不同是非常重要的,因为XML处理函数是一种现成的软件组件,你可以将它加入到应用中以处理XML文件。当你可以使用现存的XML处理函数并关注于特定应用的编码时,为什么还要进行重复的设计呢?Internet Explorer 5.0就是一个XML应用的好例子。在Internet Explorer 5.0中,XML处理函数根据浏览器的行为来进行XML文件的处理。

注意:XML处理函数和XML解析器常常被混淆。XML解析器是XML处理函数的一个组成部分,它分析XML标记并确定文件数据的结构。有时你会看到人们交替使用这两个术语,这种情况下,他们很可能指的是XML解析器。

在你刚才看到的例子中,包括了name元素,这也揭示了XML语法的某些机制,但是显然还有更多。你看到在<name>和</name>标记字段之间的字符数据是字符串

Maximillian。字符数据表示了 XML 文件实际的信息内容,而剩下的标记说明了文件的结构。

在 XML 的物理结构和逻辑结构之间有一个有趣的区别。文件的物理结构是由文件中的实体决定的。通常实体相当于文件,不过并不是必须如此。XML 不鼓励你将 XML 文件看成是文件或者是文件的集合。相反,你应当将 XML 文件看成是实体的集合。另外,总是存在一个文件实体作为文件的主实体。

如果实体描述了 XML 文件的物理结构,那么是什么描述了逻辑结构呢?当然是元素。文件的逻辑结构是由文件中的元素以及这些元素之间的关系决定的。XML 文件的物理/逻辑结构的一个类比是关系数据库。关系数据库用表和字段来描述逻辑结构,而通常使用二进制文件来描述物理结构。

注意:元素有起始标记(
)和结束标记(</br>),或者是空标记(
),而且可能包含字符数据、子元素,或者二者兼有。

前面已经提到,XML 文件由字符数据和标记组成部分。下面就是 XML 1.0 所支持的几种不同的 XML 标记组成部分。

- 元素标记
- 处理指令
- 文件类型声明
- 实体引用
- 注释
- 标记片段

下面介绍这些 XML 语法组成部分。你将在接下来的章节中看到 XML 语法的细节,但是对于新手来说,在这里介绍一下要点还是很有帮助的。

为了更好地理解 XML 语法的主要组成部分,有必要使用一个 XML 文件的例子。清单 1.1 包含一个用来保存联系信息的 XML 地址簿。下面的几节中将要使用这个例子来解释不同 XML 语法组成部分的作用。

清单 1.1 XML 地址簿

```
<? xml version="1.0"? >
<! DOCTYPE addressbook SYSTEM "AddressBook.dtd" [
  <! ENTITY amp "&#38;#38;">
  <! ENTITY apos "&#39;">
]>
<addressbook>
  <!-- This is my good friend Frank. -->
  <contact>
    <name>Frank Rizzo</name>
    <address>12 12 W 304th Street</address>
    <city>New York</city>
    <state>New York</state>
    <zip>10011</zip>
    <phone>
```

```
<voice>212-555-1212</voice>
<fax>212-555-1213</fax>
</phone>
<email>frizzo@fruity.com</email>
<web>http://www.fruity.com/rizzo</web>
<company>Frank's Ratchet Service</company>
</contact>
<!-- This is my old college roommate Sol. -->
<contact>
  <name>Sol Rosenberg</name>
  <address>1162 E 412th Street </address>
  <city>New York</city>
  <state>New York</state>
  <zip>10011</zip>
  <phone>
    <voice>212-555-1818</voice>
    <fax>212-555-1819</fax>
  </phone>
  <email>srosenberg@fruity.com</email>
  <web>http://www.fruity.com/rosenberg</web>
  <company>Rosenberg's Shoes & Glasses</company>
</contact>
</addressbook>
```

1.3.1 标记

标记(tag)是 XML 语法中最显而易见的组成部分,它被用来描述元素。比如,在地址簿例子中,元素 city 是由标记<city>和</city>构成的。为了便于理解,可以将“元素”这个术语理解为逻辑上的标记片段,而“标记”是指用来在 XML 文件中表示元素的文本字符串。

与大多数人对 HTML 的理解相反,XML 元素可以是空的,这意味着元素可以不含有任何解析的字符数据。比如,HTML 中的元素 br 就是一个空元素,因为它不含有任何字符数据。然而,XML 语法不允许你在传统的 HTML 风格下用
标记来使用元素 br。XML 中空元素的标记必须在元素名之后用一个斜杠(/)来指出它是空的。因此,
标记在 XML 中的用法就变成了
。

如果你不看连接,HTML 中空元素的语法来自结束标记的格式。比如,HTML 中元素 head 被编为一对标记——<head>和</head>。第二个标记中的斜杠指出它是这对标记中的结束标记。空元素使用相同的方法来说明没有相应的结束标记。然而,有必要指出你仍然可以以“起始标记/结束标记”的形式来表示空元素。比如,你可以用
</br>来表示
标记,在 XML 语法中,这也是合法的。

1.3.2 实体引用

正如你所了解的那样,实体是构成 XML 文件的基本单位,而这就是实体本身以及通常通过实体引用构成的其他实体。实体引用在 XML 中被用来为数据片赋予别名。实质上,实体引用还是 XML 数据片的惟一名称。比如,地址簿文件通过实体引用来使用单引号(')与和号(&)作为解析字符数据:

```
<company>Frank&apos;s Ratchet Service</company>  
<company>Rosenberg&apos;s Shoes & Glasses</company>
```

实体引用在和号和分号之间。在这个例子中,'和&是作为单引号与和号的别名的实体引用。通常,XML 解析器可能会根据这些字符在XML中不同的结构作用来解析这些字符。然而,通过实体引用,你可以使用它们而不必考虑XML解析器的存在及其带来的问题。

注意:在XML中,大多数实体引用在使用之前都必须显式地声明。然而,下列实体引用是自动声明的,因此可以在XML中随意使用:&、'、"、<和>。这些实体引用对应于下列字符:和号(&)、单引号(')、引号(")、小于号(<)和大于号(>)。奇怪的是,你仍然必须声明这些实体以便使XML文件有效。这些实体的预定义主要应用于那些具有良好格式的文件,而不是有效的文件。

1.3.3 注释

在XML文件中,注释用来表示那些技术上不是文件内容的信息。与编程语言中的注释一样,XML注释被用来提供文件数据的说明,这完全是为了用户了解程序。换句话说,XML解析器和应用程序通常会忽略注释。技术上,解析器将注释返回给应用程序是有可能的,这可能对开发支持注释的XML应用很有好处。

注释可以被用在XML文件有解析字符数据出现的任何地方。未解析字符数据(CDATA)片段是这一规则的惟一例外;很快我还会在接下来的章节中回顾这一点。注释可以出现在CDATA片段,但是他们并不作为注释来处理。

注释以<! 一开始并以->结束。对注释的惟一限制就是在注释中你不能包含两个连续的连接符。这是因为它们会和XML的注释语法发生冲突。在地址簿的例子中,下面的注释被用来描述地址簿中特定的联系人。

```
<! --This is my good friend Frank. -->  
<! --This is my old college roommate Sol. -->
```

在这些注释中包含的信息不被当作是XML文件数据的一部分。

1.3.4 处理指令

XML语法不仅仅涉及了字符数据和标记,其中还包含了处理指令。处理指令是一些用来由那些处理XML文件的应用程序使用的特殊指令。XML解析器并不处理处理指令,相反,它将处理指令返回给应用程序。处理指令通常以小于号和问号(<?)开始而以问号和大于号(>?)结束。最明显的一个处理指令的例子就是众所周知的xml处理指令,在地址簿的例子中是:

```
<? xml version="1.0"? >
```

这个处理指令指出了这个文件是基于XML版本1.0的。在本书编写期间,XML的最新版本就是1.0。xml处理指令中的version属性使得XML可以持续发展而不会由于存在不同的版本而产生问题。这还使应用程序在试图处理一个不支持版本的文件时,可以提示给用户。

1.3.5 文件类型声明

文件类型声明在 XML 中用来在文件中详细地说明文件信息,其中包括文件根元素和文件类型定义(Document Type Definition, DTD)。

文件的文件类型声明对于确定一个文件是否有效或是否仅仅是结构良好的来说是非常重要的。文件类型声明有如下三个主要作用:

- 指定文件的根元素。
- 为文件定义元素、属性和实体的细节(内部 DTD)。
- 指出文件的外部 DTD。

注意:结构良好的文件是一个符合 XML 语法规则的文件,而有效的文件不仅需要是结构良好的文件,还必须符合 DTD。

地址簿文件的例子说明了文件类型声明的这些功能是怎样实现的。在地址簿文件中使用了下面的文件类型声明:

```
<!DOCTYPE addressbook SYSTEM "AddressBook.dtd">
```

在文件类型声明中明确地指定了文件的根元素是 addressbook 元素,文件的外部 DTD——AddressBook.dtd 在文件的类型声明中也被明确地引用。请记住在应用程序需要的时候,XML 解析器使用这些 DTD 来验证文件是否有效。

文件类型声明的第二个用途在地址簿文件中没有出现,因为这个文件不需要特定文件的元素、属性或实体。不过,复杂些的文件肯定可以从这种特定文件的元素、属性和实体中受益。另外,如果你不打算重复使用外部 DTD,那么也可以将外部 DTD 的内容直接插入到文件类型声明中。在这种情况下,文件有效性的验证将依赖于内部 DTD。

1.3.6 CDATA 片段

未解析字符数据片段,也就是 CDATA 片段,被用来在 XML 文件中将那些不需要 XML 解析器处理的内容分开。具体地说,XML 文件的 CDATA 片段包含的是那些你不打算作为 XML 字符数据解析的内容。你可以通过将 CDATA 片段放在<![CDATA[和]]>之间来定义它。下面就是一个 CDATA 片段的例子:

```
<![CDATA[
  <name>Jack Tors</name>
  <address>1816 N 708th Street</address>
]]>
  <city>New York</city>
  <state> New York</state>
```

在这个例子中,元素 name 和元素 address 不被当作 XML 标记,而其中的数据也不被当作解析的字符数据,因为这些标记被放在 CDATA 片段中。为了使文件中的内容被当作 XML 标记和解析的字符数据来处理,XML 解析器必须能够解析它。由于 name 标记和 address 标记被放在了未被解析的 CDATA 片段中,所以它们不会被解析。尽管这个例子说明了如何将一般的 XML 元素放在 CDATA 片段中,通常使用 CDATA 片段来将一段 XML

代码引用出来,尤其是在 XML 帮助以及其他 XML 文件中。

注意:通常你不应当将注释放在文件的 CDATA 片段中,因为它们不会被当作注释来处理。

1.4 XML 的现状

我们已经对 XML 有了大致的了解,而且你也看到了 XML 的基本语法,接下来我希望你能够跟上当前的 XML 发展速度。很明显,如果没有可利用的软件来组织构建 XML,XML 再有多少好处也是白费。因此,尽管本书的其他部分主要介绍 XML 的特征和应用,现在我打算先让你看一看它可以做些什么。本章的剩下部分将着重介绍当今 XML 的应用。

已经出现了许多与 XML 相关的不同技术。除了我们马上就要看到的样式表技术之外,还有很多其他有趣的技术。XML Namespaces 就是其中之一,它提供了一种为 XML 元素分配惟一名字的方法。XML 文件对象模型(DOM)也是一项重要的技术,它使程序可以访问 XML 文件的内部结构。Microsoft 的 XML-Data Schema 技术提供了另外一种用传统的 DTD 设计 XML 文件类的方法,而且在很多方面这种方法更为有效。XLink 和 XPointer 技术提供了一种在 XML 资源之间建立高级链接的方法。

注意:所有上述的 XML 技术在后面的章节中都将一一讨论。这里我希望你能够了解 XML 技术的广泛性。考虑到 XML 新近才被发布,出现这么多相关的 XML 技术真是令人惊讶。这充分说明了 XML 发展的是多么快。随着越来越多的 Web 开发者开始认识到 XML 的巨大潜力,相信 XML 会继续快速的发展。

1.4.1 XML 的显示样式

因为 XML 是一种基于内容的元语言,所以它并不关心 XML 文件如何被显示。我们怎样才能看到这种根本没有任何关于显示的信息的文件呢?答案是,为了浏览 XML 文件,我们必须提供如何来显示它的相应信息。这可以通过使用 CSS 或者 XSL 定义样式表来实现。

CSS 是一种为 HTML 而创建的样式表技术,不过对于 XML 它也很有效。XSL 是一种更先进的专门为 XML 使用的样式表技术。XSL 比 CSS 更复杂,功能也更为强大。目前,关于这两种样式表技术哪个好以及哪一个更适于 XML 使用的讨论正在进行着。将在第 8 章“对比两种样式规范:可扩展样式表语言和层叠式样式表”中详细地讨论。

尽管样式表是一种不错的显示 XML 文件的手段,你也可以编写直接处理 XML 文件的代码来修饰显示效果。高级的 XML 应用使用这种方法来获得在显示 XML 文件内容时更为精确的控制。这样的应用程序可能会使用 Java servlet 利用现有的 XML 解析器来处理 XML 文件,然后将结果用 HTML 方式进行格式化。

1.4.2 浏览 XML

不久之前,浏览 XML 文件还仅仅是一个概念。现在,Web 已经成为支持 XML 文件的网络。Microsoft 的 Internet Explorer 5.0 是第一个完全支持 XML 1.0 的商业 Web 浏览器。Internet Explorer 还支持一些其他与 XML 密切相关的技术,比如 XML DOM(XSL 的子

集),XML Namespaces 1.0 建议以及一种作为 XML Schema 出现的技术。另外,Internet Explorer 5.0 还支持两种 XML 词表:频道定义格式(Channel Definition Format,CDF)和向量标记语言(Vector Markup Language,VML)。Microsoft 显然非常重视 XML 的未来,尽管它没有完全遵守 W3C 制订的 XML 标准。关于 Internet Explorer 5.0 的更多信息,请参阅 Internet Explorer 的站点 <http://www.microsoft.com/ie>。

Netscape 也宣布了在 Navigator 的下一个版本中完全支持 XML 1.0 的计划,尽管它已经公开了 Mozilla 源代码。Navigator 的下一个版本可能也将包括对 XUL 的支持。XUL 读做“zool”,是一个用来描述跨平台的图形用户界面的 XML 词表。更多关于 Navigator 下一个版本的信息,请参阅 Mozilla 的 Web 站点:<http://www.mozilla.org>。在这个版本发布之后,我们就能够从 Netscape 的 Web 站点 <http://www.netscape.com> 上看到它了。

W3C 也有一种称为 Amaya 的浏览器可以浏览 XML 文件。它更多的被当作编辑工具来使用,因为它还支持 Web 文件的编辑。同时,它还可以作为一个像样的浏览器测试平台。比如,Amaya 支持 MathML 的一个子集,MathML 是一个用来描述数学公式的词表。你将在第 36 章“用 MathML 来谈数学”中学习到 MathML,另外,在第 17 章“浏览 XML”中将对当前浏览器对 XML 的支持会有更深刻的讲解。

1.4.3 解析 XML

XML 解析器实际上是现在最成熟的 XML 软件。已经存在许多 XML 解析器,可以用来解析 XML 代码,使你可以访问 XML 文件中的基本内容。大多数 XML 解析器既可以验证 Schema 操作也可以不验证 Schema 操作。另外,大多数可用的 XML 解析器都被设计成软件组件,这样,它就可以很容易的被加入到应用程序之中。换句话说,你只要简单地在应用程序中插入 XML 解析器就可以使之支持 XML 了。

下面是目前可用的一些比较流行的 XML 解析器的清单:

- Lark and Larval XML Parsers for Java
- Sun's Project X Parser for Java
- Datachannel's XML Parser for Java
- IBM's XML Parser for Java
- The Oracle XML Parser for Java
- Microstar's Aelfred XML Parser for Java
- IBM's XML Parser for C++
- The XML Parser for C++ Builder

正如你所看到的,大多数可用的 XML 浏览器是用 Java 设计的,这没有什么可奇怪的,因为 Java 是 Web 的首选编程语言。然而,也存在一些 C 和 C++ 的 XML 解析器,其他语言的解析器也有一些。在第 12 章“XML 处理基础”、第 13 章“用 Java 解析 XML”、第 14 章“用 C++ 解析 XML”中,你将会学习到如何利用这些解析器来解析 XML 文件。

1.5 总 结

本章以一个不错的技术论述作为本书的开始。通过本章讲述能够使你对 XML 的语法和 XML 文件的整体结构有一个大致的认识。另外,我还希望你能够对 XML 的起源、它们之间的关系和相关的技术,比如 HTML 和 SGML 有所了解。最后,我希望你对 XML 的现状有所了解——它是如何通过样式表来显示的,哪些浏览器支持它以及它是如何被解析的。

在本书的其他章节中将会详细讨论本章中的所有问题。不过,在学习某一特定内容之前,建立你对 XML 的基本概念和大致印象是非常重要的准备。

第2章 数据模型:文件类型定义和 Schema

本章介绍了使用文件类型定义(DTD)和XML Schema的XML数据模型。一般地讲,Schema描述了XML文件的数据模型。有两种基本的方法来定义XML:DTD和XML Schema。在本章中“Schema”一词一般用来描述XML数据模型,而XML Schema则是取代DTD的一种特殊的Schema技术。

在本章中,我们首先学习了XML数据模型的基础,包括为了验证而描述XML文件的结构;然后比较了DTD和XML Schema并且提供了一些信息,以便帮助你决定你真正需要的是哪种方法;最后,你将了解到这两种方法对于大多数文件来说都足够了,而各自又有各自的好处。DTD已经被广泛使用并能共享扩展工具的支持,而XML Schema则功能更为强大并且提供了一些高级特征,诸如开放内容模型,名字空间集成以及复杂数据类型。

2.1 XML数据模型基础

XML文件本质上是保存信息的结构化载体。为了得到XML文件的有效性,你必须明确确定文件中的信息必须遵守哪些结构。这是通过Schema来实现的,Schema是一种描述XML文件中信息结构的模型。在XML中,Schema被用来建立某类数据的模型。一旦数据模型适合特定的数据类,我们就可以创建遵守该模型的结构化XML文件。

注意:如果你已经对数据库有所了解,那么你可能已经听说过Schema这个词。数据库Schema被用来描述关系数据库表中的数据的结构。

Schema描述了在有效的XML文件中的标记和字符数据的排列。在这种情况下,术语“有效的”非常重要,因为创建没有Schema的XML文件是完全合法的,在这种情况下,文件是结构良好的但不是有效的。换句话说,XML文件必须符合Schema才能成为有效的。你可以认为Schema是在XML应用和应用所用的XML文件之间的一个协议。

比如,金融方面的应用可能希望XML文件中金融交易的代码可以遵守特定的XML词表,比如OFX(Open Financial eXchange,开放的金融交换)。如果文件没有遵守这个词表,那么应用程序就无法从中读取数据。但是应用程序是如何知道文件是否含有有效数据的呢?答案是应用程序应当使用OFX Schema来在它试图从文件中取得数据之前确认所有文件都是有效的。很明显,在这个例子中,Schema在使应用程序更健壮方面起到了关键的作用。

如果Schema的概念仍然有点抽象,可以考虑一下现实世界的类比。我们假设有个朋友向你推荐了一个Web站点。他可能告诉你这个网站的地址是http://www.cnn.com。你知道域名是不会用逗号做分隔符的,于是你就有了一个很好的线索,可以知道你的朋友犯了错误。你用一个非常简单的Schema来得到这个结论——你知道域名肯定是用句点做分隔符而不是逗号。

这样我们就可以知道Schema为什么在XML中如此重要了:目的是为了使机器可以验

证文件的结构。在这个无效 URL 的例子中,你可很容易地发现问题,因为你知道域名中不能含有逗号。但是 Web 浏览器是如何作出这种判断的呢?浏览器的开发者可以写一段特定的代码来确保 URL 的结构符合给定的语法,比如域名部分用句点做分隔符。

既然浏览器开发者可以编写代码来检查 URL 的有效性,XML 文件的作者也可以使用 Schema。之后,这个 Schema 可以被应用程序使用,用来确保文件的有效性。Schema 使软件获得 XML 文件的有效性成为可能。

注意:XML 文件即使是有效的也不意味着它是无错的。有效的 XML 文件仍然可能有语义上的错误——人为的错误,而这些错误没有破坏 XML 语法或 Schema 规则。最好的消息是语义错不会影响 XML 文件的处理。因此,尽管应用程序可能由于语义错而没有返回预期的结果,但是它仍然可以继续处理 XML 文件而不会失败。

要理解 Schema 在 XML 文件验证中的作用,最好的办法就是考虑一下没有 Schema 的文件。尽管没有 Schema,一个结构良好的 XML 文件仍然必须遵守 XML 语法。根据 XML 1.0 建议,文件必须遵守 XML 语法才能被认为是 XML 文件。然而,创建那些数据的结构和组织没有限制的 XML 文件仍然是可能的。Schema 就是为了解决这个问题。Schema 建立了文件中标记和字符数据的约束,这样就决定了 XML 文件的结构。

在前面,我们已经了解到 Schema 被用来为数据类建立模型,它也可以被称为 XML 词表。Schema 明确定义了词表中元素之间的关系,并描述了一个特定的 XML 词表。没有 Schema,XML 就没有什么用处,因为它仅仅是一种元语言,而这在很多方面都太泛化了。Schema 使得创建文件必须遵守才能使之有效的特定 XML 词表成为可能。根据特定词表创建的 XML 文件要通过将它和该词表的 Schema 进行比较来检查有效性。

刚才我提到过,Schema 建立了文件内容结构的约束。Schema 是用下列方法来实现的:

1. Schema 为文件建立内容模型,这个模型定义了元素的顺序以及元素的嵌套。
2. Schema 建立了文件数据的数据类型。

文件内容模型是极其重要的,因为它决定了元素的顺序和嵌套。这是 DTD 控制的首要约束,而这是源于 SGML 的传统解决方案规划,DTD 不具有约束数据类型功能,高级的 XML 数据 Schema 支持数据类型约束,而这在提供数据更高层次的合法性方面比 DTD 方法具有更大的优势。

为了理解为什么 Schema 约束如此重要,可以考虑上一章中的地址簿示例文件,详见清单 2.1。这个文件遵守地址簿 DTD(AddressBook.dtd),我们将要在下一节“用 XML Schema 来建立数据模型”中介绍这个 DTD。

清单 2.1 地址簿 XML 文件

```
<? xml version="1.0"? >
<! DOCTYPE addressbook SYSTEM "AddressBook.dtd" [
<! ENTITY amp "&#38;#38;">
<! ENTITY apos "&#39;">
]>
```

```

<addressbook>
  <! -- This is my good friend Frank. -->
  <contact>
    <name>Frank Rizzo</name>
    <address>1212 W 304th Street</address>
    <city>New York</city>
    <state>New York</state>
    <zip>10011</zip>
    <phone>
      <voice>212-555-1212</voice>
      <fax>212-555-1213</fax>
    </phone>
    <email>frizzo@fruity.com</email>
    <web>http://www.fruity.com/rizzo</web>
    <company>Frank's Ratchet Service</company>
  </contact>
  <! -- This is my old college roommate Sol. -->
  <contact>
    <name>Sol Rosenberg</name>
    <address>1162 E 412th Street </address>
    <city>New York</city>
    <state>New York</state>
    <zip>10011</zip>
    <phone>
      <voice>212-555-1818</voice>
      <fax>212-555-1819</fax>
    </phone>
    <email>srosenberg@fruity.com</email>
    <web>http://www.fruity.com/rosenberg</web>
    <company>Rosenberg's Shoes & Glasses</company>
  </contact>
</addressbook>

```

在这个示例文件中,含有具体联系信息的元素都嵌套在 `contact` 元素中,这一点是非常重要的。另外,voice 元素和 fax 元素在 phone 元素的上下文之外是没有意义的。这些关系要反映在文件的内容模型之中,这个模型是由 Schema 定义的。XML 处理函数使用 Schema 来确定因为不正确的元素嵌套,下面的代码是错误的:

```

<name>Sol Rosenberg</name>
<contact>
  <address>1162 E412th Street</address>
  <city>New York</city>
  <state>New York</state>
  <zip>10011</zip>
  <phone>
    <voice>212-555-1818</voice>
  </phone>
  <fax>212-555-1819</fax>
  <email>srosenberg@fruity.com</email>
  <web>http://www.fruity.com/rosenberg</web>
  <company>Rosenberg's Shoes & Glasses</company>

```

</contact>

元素 name 出现在 contact 元素之外,这和内容模型是冲突的。同样, fax 元素也出现在 phone 元素之外,这也违反了文件的内容模型。

除了指定元素的嵌套关系之外,内容模型还可以指定元素的顺序。比如,你可能想要让 name, address, city, state 和 zip 这几个元素以严格的顺序出现。下面的例子违反了该内容模型,因此它不是有效的文件:

```
<contact>
  <name>Sol Rosenberg</name>
  <state>New York</state>
  <address>1162 E 412th Street</address>
  <zip>10011</zip>
  <city>New York</city>
  <phone>
    <voice>212-555-1818</voice>
    <fax>212-555-1819</fax>
  </phone>
  <email>srosenberg@fruity.com</email>
  <web>http://www.fruity.com/rosenberg</web>
  <company>Rosenberg's Shoes & Glasses</company>
</contact>
```

Schema 产生的另外一种约束是数据类型约束,它会影响数据的实际类型和标记。比如,你可以指定某个元素包含 0 到 100 的整数,其他任何数据放在该元素中都会违反该 Schema 并影响文件的有效性。

注意到 DTD 不支持数据类型约束是十分重要的。为了使用数据类型约束,我们必须创建遵守 XML-Data 规范或者遵守该规范的一个子集的规范,这个规范支持数据类型约束。Microsoft 的 Internet Explorer 5.0 目前支持一个称为 XML Schema 的 XML-Data 的子集,它支持数据类型约束。

2.2 用 DTD 来建立数据模型

DTD 是建立 XML 文件的 Schema 的一种方法。DTD 最初出现在 SGML 中,并且用作验证 SGML 文件有效性的一种标准的 Schema 机制。由于 XML 是 SGML 的一个子集,它们使用了同样的 Schema 方法就可以理解了。使用 DTD 作为 Schema 的一个主要好处是现有的 SGML 工具可以很容易地被修改为支持 XML,因为它们已经在 SGML 中支持了 DTD。然而,DTD 不是完美的。事实上,它非常严重的局限性已经足以让人们考虑使用其他方法来为 XML 数据建立模型了。

DTD 依靠特定的语法来描述 XML 词表的结构。这是 XML 团体对 DTD 最不满的地方之一。为什么还要学习一种特殊的语法呢?XML 本身已经提供了描述任何类型(也包括文件结构)的数据的理想语法。在下一节“用 XML Schema 建立数据模型”中,我们将介绍另外一种建立数据模型的方法。

尽管 DTD 语法相当紧凑,它描述文件结构的方式仍相当含糊而且不直接。比如,它用

单独的字符如问号、星号以及加号来描述文件的内容模型。幸运的是,学习 DTD 的语法并且创建 DTD 并不复杂。清单 2.2 包含了描述我们前面看到过的地址簿例子的数据模型的 DTD 的代码。

清单 2.2 地址簿 XML 文件的 DTD

```
<! ELEMENT addressbook (contact)+>
<! ELEMENT contact (name, address+, city, state, zip, phone, email, web, company)>
<! ELEMENT name (#PCDATA)>
<! ELEMENT address (#PCDATA)>
<! ELEMENT city (#PCDATA)>
<! ELEMENT state (#PCDATA)>
<! ELEMENT zip (#PCDATA)>
<! ELEMENT phone (voice, fax?)>
<! ELEMENT voice (#PCDATA)>
<! ELEMENT fax (#PCDATA)>
<! ELEMENT email (#PCDATA)>
<! ELEMENT web (#PCDATA)>
<! ELEMENT company (#PCDATA)>
```

尽管这个 DTD 没有说明 DTD 语法的所有特征,比如如何定义属性,但是它仍然揭示了用 DTD 语法来定义文件模型的基本机制。请注意 contact 元素的内容模型是由括号中的一系列元素来提供的。在地址元素后面的加号(+)说明它会一个接一个地出现。Phone 元素也包含一个列出了 phone 子元素的内容模型。在这个例子中,问号说明 fax 元素是可选的,但是如果出现,它也只能出现一次。

DTD 中列出的所有其他元素都含有 #PCDATA。PCDATA 代表了已解析的字符数据。元素也可以含有未解析的字符数据(CDATA)或者子元素,就像例子中的 contact 元素和 phone 元素一样。

为了更简洁起见,你还可以将地址簿 DTD 中的元素组织成下面的样子:

```
<! ELEMENT addressbook (contact)+>
<! ELEMENT contact (name,address+,city,state,zip,phone,email,
web,company)>
<! ELEMENT phone (voice, fax?)>
<! ELEMENT (name, address, city, state, zip, voice, fax, email,
web, company)(#PCDATA)>
```

这段代码将所有的 #PCDATA 元素都集合在一起,使它们的声明更为简洁。这段代码的功能和清单 2.2 中的代码是一样的。像这样将元素集合在一起,或是把它们展开仅仅是个人编程风格的问题,因为这根本不会影响 DTD 的结构。

在后两章中我们会更详细地讨论 DTD,因此这里我就不多说了。此外,我还想向你介绍建立数据模型的另外一种方法,这种方法使用了你非常熟悉的语法。

2.3 用 XML Schema 建立数据模型

XML Schema 是一种定义 XML 文件的 Schema 的新方法,它使用了称为 XML-Data 的 XML 词表。在写这本书的时候,XML-Data 仍在 W3C 的草案修订阶段,这意味着正式建议很快就会出台。Microsoft 迫不及待地提供了 DTD 的替代品,并决定在 Internet Explorer 5.0 中支持 XML-Data 的一个子集。这个 XML-Data 的子集被 Microsoft 称为 XML Schema。

XML Schema 和 DTD 的目的非常类似,都是为了建立某一类 XML 文件的 Schema。和 DTD 一样,XML Schema 描述了元素和元素的内容模型,这样文件就可以被验证了。然而,XML Schema 比 DTD 更进了一步,它还将元素和数据类型联系起来。这就使 XML 处理函数可以对数据内容进行验证,在使用 XML Schema 而不是 DTD 的时候,这是一个非常显著的优越之处。对于 DTD,元素的内容仅仅限于字符串类型,而 XML Schema 可以将元素的数据设置成非常特别的类型,比如整数型和日期型。XML Schema 还支持一些其他特征比如开放内容模型和名字空间集成,我们很快就会学习它们。

XML Schema 最有意思的地方也许就是它使用了 XML 语法。这意味着你可以像创建 XML 文件那样创建 XML Schema。因此,为了编写 XML Schema,你只要掌握编写 XML 文件的那些标记方法就可以了。当然,你还要学习 XML Schema 词表(XML-Data 的一个子集),但是这非常容易。清单 2.3 包含了相当于实现地址簿示例文件的 DTD 的 XML Schema。

清单 2.3 地址簿 XML 文件的 XML Schema

```
<? xml version = "1.0" ? >
<Schema name="AddressBookSchema"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name = "name" content="textOnly"/>
  <ElementType name = "address" content = "textOnly"/>
  <ElementType name = "city" content = "textOnly"/>
  <ElementType name = "state" content = "textOnly"/>
  <ElementType name = "zip" content = "textOnly"/>
  <ElementType name = "voice" content = "textOnly"/>
  <ElementType name = "fax" content = "textOnly"/>
  <ElementType name = "phone" content = "elOnly"/>
    <element type = "voice" minOccurs="1" maxOccurs="1"/>
    <element type = "fax" minOccurs="0" maxOccurs="1"/>
  </ElementType>
</ElementType name = "email" content = "textOnly"/>
</ElementType name = "web" content = "textOnly"/>
</ElementType name = "company" content = "textOnly"/>
</ElementType name = "contact" content = "eltOnly">
```

```
<element type="name" minOccurs="1" maxOccurs="1" />
<element type="address" minOccurs="1" maxOccurs="2" />
<element type="city" minOccurs="1" maxOccurs="1" />
<element type="state" minOccurs="1" maxOccurs="1" />
<element type="zip" minOccurs="1" maxOccurs="1" />
<element type="phone" minOccurs="1" maxOccurs="1" />
<element type="emial" minOccurs="0" maxOccurs="1" />
<element type="web" minOccurs="0" maxOccurs="1" />
<element type="company" minOccurs="0" maxOccurs="1" />
</ElementType>
</ElementType name="addressbook" content="eltOnly">
  <element type="contact" minOccurs="1" />
</ElementType>
</Schema>
```

正如你所看到的,XML Schema 没有 DTD 那么简洁。然而,有时 XML Schema 更容易理解,因为它明确地讲述了数据模型的细节。举例说来,每一个元素的 content 都明确地在 content 属性中指出为 eltOnly 或者 textOnly,这说明元素可以只含有元素或者只含有文本。XML Schema 还支持 mixed 和 empty 作为 content 属性可接受的值。

XML Schema 还为元素建立数据类型,这明显地出现在 zip 元素、voice 元素和 fax 元素中。这些元素都被设置为 int 数据类型,这意味着它们含有整数值。当然,这要求地址簿文件指定这些元素为数字,而没有和电话号码以及扩展邮编相联系的分隔符。

XML Schema 一个非常强大的功能是它为元素建立内容模型的方式。更确切地说,在嵌套时可以用 minOccurs 和 maxOccurs 属性非常明确地指出元素出现的次数。比如,在 contact 元素中嵌套的 address 元素必须至少出现一次,而且至多两次。这样就提供了至多两个地址行,这是标准的指定邮件地址的方法。请记住 DTD 并没有提供类似的方法,使其能很好地控制元素在另外一个元素中出现的次数。

这个例子实际上仅仅肤浅地涉及了 XML Schema 能力。在第 5 章“XML Schema 基础”和第 6 章“XML Schema 构造技术”中,我们将学习更多 XML Schema 的知识。现在,我只想介绍一些基础知识,以便你能够对比 XML Schema 和 DTD。

2.4 比较两种建立数据模型的方法

你已经对于 Schema 和这两种用来构造 XML 文件数据的基本方法有了一些了解了,现在是评估每种方法的相关性的时候了。你可能非常想知道你应该使用哪个方法,以及相应的理由。或者是否会有不同的情况来决定使用哪一种方法?

评估任一 Schema 方法的最佳开始点就是清楚的概述出它们之间的区别。迄今,我已经在本章中提到了一些区别,但我希望确认你已经牢牢掌握了 DTD 和 XML Schema 之间是如何区别的。

下面是 DTD 和 XML Schema 之间主要区别的一个列表:

- DTD 基于专门的语法,而 XML Schema 基于 XML。
- XML Schema 基于 XML,因此它们可以像其他 XML 文件一样被解析和管理。

- DTD 很简洁但 Schema 相反。
- 有很多工具可用来处理有 DTD 的 XML 文件,而存在很少工具可用于 XML Schema。
- XML Schema 支持大量的数据类型(整数、浮点数、布尔数、日期等等),而 DTD 将所有数据看作字符串或枚举字符串。
- XML Schema 表示一种无限的数据模型,它允许你扩展变量并在没有验证文件的情况下设置元素之间的继承关系。DTD 使用有限数据模型,它不允许任何形式的扩展。
- XML Schema 支持名字空间集成,它允许你将文件的独立节点与 Schema 中的类型声明结合。DTD 值只允许文件和它的 DTD 之间通过文件类型声明来进行一种结合。
- XML Schema 支持属性组,它允许你逻辑的组合属性。DTD 支持通过参数实体进行组合的基本形式,这是有局限的,因为 XML 处理函数对于组合一无所知。

你也许会认为,我给出 DTD 和 XML Schema 间的区别列表是在给 XML Schema 做推销。实际上,这两种 Schema 方法之间的大多数可知的区别都是在 XML Schema 的改善中涉及到的。这并不是巧合。记住 DTD 从 SGML 之初就出现了。XML Schema 是新的品种,它被设计以有益于 XML 的需求。

你也许会想,那么就不用再考虑了,直接选择 XML Schema 而不是 DTD 来描述所有 XML 文件的数据模型。虽然 XML Schema 非常可能代表着 XML 数据建模的未来,但 DTD 并没有被废弃。在所有文件上使用 DTD 的一个重要理由就是 DTD 已经存在了很长的时间并获得了广泛的接受。如果你考虑到初等的技术没有能够在它的使用期中独立的生存并且还没有得到广泛的接受,你就要再好好考虑一下。MS-DOS 的存在已经超过它的实际期望寿命很长的时间,因为有太多的为它编写的软件以及太多的依赖它的系统。我并不准备在类似的事情上抨击 Microsoft,但我希望你要明白 DTD 还不能仅仅因为 XML Schema 代表的是更强大的数据构造的方法就会被取消。

事实是市面上有很多的 SGML 工具已经过改进以使用 XML 工作,而它们依赖 DTD 进行文档数据构造。因此如果你要在不久的将来寻找一个 XML 工具来使用的话,你几乎必须使用 DTD 来工作。加之,XML 词表的绝大部分当前都是用 DTD 表示的。因此,如果你计划使用任何存在的 XML 词表,你需要通过它们的 DTD 来学习它们是如何构造的。

要记住的另一点就是 DTD 实际上在描述 XML 文件的结构方面非常优秀。XML Schema 做的更好的事实并不足以完全抹杀 DTD。我的建议是你在 DTD 和 XML Schema 之间作决定之前要评估你的需求以及你计划使用的工具。如果你急于很快就推出解决方案,那么你也许更偏重于 DTD。如果你有时间实际工作,并且你喜欢使用一下 XML Schema 的某些高级特性,那就按各种可能的方法使用它吧。幸运的是,任何一种方法都足够构造绝大多数的 XML 文件。

2.5 总 结

本章探讨了 XML 数据构造的基础以及用来描述 XML 文件结构的两个方法:DTD 和 XML Schema。这两种数据构造方法主要涉及 Schema 是从数据库领域借用的。Schema 描述了在一个有效 XML 文件中的标记和字符数据的安排。Schema 不需要创建结构良好的 XML 文件,但它们本质上完全是为了创建有效的文件。DTD 起源于 SGML 并且作为一种验证 SGML 文件的标准 Schema 机制。XMLSchema 是由 Microsoft 创建的技术,它是 W3C 的 XML-Data 工作草案的部分实现。虽然 XML Schema 功能更强且更灵活,但 DTD 已经存在很长时间了,并从广泛的应用中获益更多。

第3章 DTD 基础

文件类型定义(DTD)形成了有效文件的基础,因为它建立了 XML 词表的语法,而它将决定 XML 文件的结构。DTD 对于执行文件验证而言是必要的,而验证是 XML 内容开发和调配的重要组成部分。没有对文件中声明的元素及其相互关系的理解,评价文件的有效性是不可能的。因此,使用某种 DTD 来创建有效文件是必须的。

本章介绍了 DTD 构成的基础并讨论了一些关于在 DTD 中声明元素和属性的有趣细节。

3.1 DTD 和文件结构

在我们的介绍中,我使用术语“某种 DTD”,这是因为,在 XML 中技术上没有惟一的 DTD。尽管许多 XML 开发者根据单一的外部 DTD 来思考,然而外部 DTD 并不是 XML 文件必需的选项。为了彻底理解什么是 DTD 以及 DTD 的结构,你必须从学习文件类型声明的作用开始。

XML 文件的结构是由文件类型声明决定的,这个声明出现在 XML 文件的开头。在文件类型声明中提供了标记声明来为文件或者文件类建立语法。DTD 是位于文件类型声明或者通过文件类型声明引用的实际语法。我知道这好像是文件类型声明和 DTD 之间的细微区别,但是理解这种区别是非常重要的。换句话说,文件类型声明提供了描述文件结构的引用位置,而 DTD 是实际的描述。

注意:XML 处理函数使用文件的 DTD 来决定文件的 Schema,而这个 Schema 是文件结构的定义。

文件类型声明可以完成下面的功能:

- 在文件的开头直接包含标记声明,这些声明被命名为内部 DTD 子集,或者本地 DTD。
- 应用外部标记声明,这些声明被称为外部 DTD 子集,或者外部 DTD。

因此,DTD 被分为两部分(子集):内部子集和外部子集。当提到文件的 DTD 时,你实际上说的是内部和外部子集的和。将 DTD 分为两部分的原因是为了提供灵活性。外部 DTD 子集一般描述了一类文件的文件结构,而内部 DTD 主要描述一个给定文件的结构 Schema。XML 处理函数优先使用内部 DTD 子集,这意味着你可以通过在内部 DTD 子集中重新声明外部 DTD 子集中的定义并覆盖它们。这为 DTD 提供了一种基本的继承方式,尽管它只限于一层。

DTD 描述了关于使用标记声明的文件结构关键信息。更确切的说,DTD 中可能包括下列标记声明:

- 文件允许的元素类型,以及它们的内容模型
- 每个元素所能具有的属性
- 文件允许的实体
- 对外部实体允许使用的注释

综上所述,这些标记声明定义了一类文件(或者叫做文件类型)的逻辑和物理结构。在DTD中的标记声明描述了文件类型的许多方面,这就使文件可以根据文件类型被验证。尽管有点神秘,DTD中的标记声明还成为内容开发者的重要参考。如果你不知道如何使用某种元素或者对属性可以接受的值十分好奇,那么 DTD 是取得这些答案的最基本资源。

3.1.1 内部 DTD 和外部 DTD

使用内部 DTD 和外部 DTD 中的一种或者两种的结合来描述文件的 DTD 是完全可能的。如果你使用两者之一,你必须在文件类型声明中声明它或者引用它。更确切的说,内部 DTD 必须在文件类型声明中声明,而外部 DTD 必须在那里被引用。下面是文件类型声明的语法:

```
<! DOCTYPE RootElem SYSTEM ExternalDTDRef [InternalDTDDecl]>
```

正如你所看到的,ExternalDTDRef 引用了外部 DTD,这是一个含有外部 DTD 的文件的 URI(统一资源标识符)。内部 DTD 在方括号([])之间声明,也就是文件类型声明语法的 InternalDTDDecl 的位置。除了内部和外部 DTD,还有一个非常重要的信息在文件类型声明中被提到:根元素。在文件类型声明的语法中,RootElem 指出了文件类的根元素。

警告:文件类型声明必须出现在 XML 声明之后,文件第一个元素(标记)之前。

下面的例子有助于区分文件类型声明中两种 DTD 的不同作用,并且还指出了根文件元素:

```
<! DOCTYPE movies SYSTEM "Movies.dtd" [  
  <! Element actor (#PCDATA)> ]>
```

这个例子可能被用于含有电影集的文件。该文件类的根元素是 movies,这意味着所有这种类型的文档都必须包含在 movies 元素中。文件类型声明引用了一个保存在文件 Movies.dtd 中的外部 DTD。另外,叫做 actor 的元素作为内部 DTD 的一部分被声明。请记住内部标记声明一般会覆盖同名的外部声明,如果这样的声明存在的话。

DTD 和文件的关系受另外一个消息的影响:XML 声明。如果你还记得,XML 声明是一个处理指令,它一般指出文件的 XML 版本,比如下面的例子:

```
<? xml version="1.0"? >
```

除了 XML 版本,XML 声明还可以描述文件的独立状态,以及字符编码。你将在下一章中了解到文件的字符编码。文件的独立状态决定了文件是否依赖于外部信息源,比如外部 DTD。使用独立文件声明,你可以显式地设置文件的独立状态,看上去就像是 XML 声明的一个属性:

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

standalone 的两个可能的值是 yes 和 no。Yes 指出文件是独立的,因此它不依赖于外部信息源,包括外部标记声明。No 指出文件不是独立的,因此可能依赖于外部信息源。很明显,依赖于外部 DTD 进行验证的文件不是独立的,而且你必须将其 standalone 值设置为 no。因此,no 是 standalone 的缺省值,这意味着如果你打算使用外部 DTD,你可以放弃独立文件声明。

如果你正在创建一个不依赖于外部标记声明的文件,你是否考虑过独立文件声明的影响?事实上,如果你没有使用任何外部标记声明,那么独立文件声明没有任何意义。换句话说,独立文件声明的惟一的实际意义在于在内部 DTD 中忽略外部 DTD 或者任何外部标记声明。当然,忽略任何外部标记声明而不将它们迁移到内部 DTD 中会使文件成为非法,但是文件依然是结构良好的,在很多情况下,这就足够了。

另外,参数实体是只用在 DTD 中的一般实体,通过保存通常使用的声明组件,它可以帮助我们模块化 DTD 的结构。

现在你知道了内部 DTD 和外部 DTD 的区别,你可能会想知道如何决定使用哪一个。请记住同时使用这两个 DTD 子集是可能的,而且有时非常有用。外部 DTD 是用于 XML 文件验证的最常见的 DTD,因此值得先讨论它。下面是你可能会考虑将文件标记声明放在外部 DTD 中的原因:

- 你正在创建的文件必须是有效的。
- 你正在创建同一类的许多文件。
- 你希望使用现有的 DTD。
- 你希望使你的文件尽可能简洁。

第一个原因相当显而易见,因为文件要有效就需要 DTD。然而,即使标记声明被放在文档的内部 DTD 子集中,文件仍然可以是完全有效的,尽管如此,由于组织的需要,将标记声明放在外部 DTD 中一般仍是一个好主意。理想的情况下,你需要在文件的标记声明和文件的实际内容之间有一个逻辑上的分隔。将标记声明放在外部 DTD 中可以帮助你实现这一点。

显而易见,如果你考虑创建许多同类的文件,外部 DTD 更为可用。很明显,内部文件在这种情况下是缺乏效率的,因为你还必须维护每一个文件中的标记声明的拷贝。然而,将文件相关的标记声明,比如本地实体或者注释,放在每一个文件的内部 DTD 中也是可以的。

如果你正好发现一个现存的 DTD 符合你的需求,你完全可以使用它而不必试图从头做起。在这种情况下,你可以将这个现有的 DTD 用作外部 DTD 并且在内部 DTD 中进行必要的修改。如果你因为必须在所有使用这个 DTD 的文件中完全复制它们而不覆盖任何外部标记声明,则那是最好的了。

最后一个使用外部 DTD 的原因是为了帮助我们使文件尽可能的简洁。请记住内部 DTD 出现在文件的开始,在第一个标记元素之前。标记了内部 DTD 的标记声明会使文件变得混乱。因此,如果你想要让你的文件简洁而易懂,那么将这些标记声明放在外部 DTD 中会很有帮助。

尽管使用外部 DTD 有许多优点,但是也有一些情况你可能会想要利用内部 DTD 所具

有的优势。下面是一些你需要将文件标记声明放在内部 DTD 中的一些原因：

- 你的文件是否有效并不重要。
- 你正在创建的是一个单一文件。
- 你想要使文件的开销尽可能的小。

如果你的文件是否有效并不重要,那么显然不必使用外部 DTD。技术上本不必使用内部 DTD,但是你可能会需要本地声明一些实体和注释。你的文件仍然必须是结构良好的,但是不必是有效的,使你可以避开整个 DTD 问题。

如果你正在创建一个单一文件,而不是一类多个文件,那么使用外部 DTD 是不必要的。文件类的整个概念就是基于一个单一的模板(外部 DTD)来创建多个文件的。如果你要创建的仅仅是一个单一文件,那么没有道理去建立文件类。在这种情况下,一个含有内部标记声明和文件内容的单一的、自包含的文件实际上是更好的选择。当然,你可能仍然选择将标记声明放在外部 DTD 中来组织,但是这并不是必须的。

使用内部 DTD 而不是外部 DTD 的最后一个原因就是效率。内部 DTD 比外部 DTD 的效率更高一些,因为内部 DTD 是物理文件文件的一部分。XML 处理函数必须在所有标记声明都包含在内部 DTD 的时候才能处理单一文件。假设 XML 处理函数读入其他文件(外部 DTD)不是一个大问题,但是在 Web 环境中,它会需要额外的 HTTP 链接。

3.1.2 有效的和结构良好的文件

如你所知,DTD 对于文件的验证绝对是必需的。一般而言,一个正确的文件必须声明文件类型并遵守由这个文件类型所规定的逻辑和物理结构。下面是确认文件有效性的四个主要的必要条件:

- 文件必须是结构良好的。
- 文件的根元素的名字必须与文件类型中的名字匹配。
- 文件必须有一个声明所有用于文件中的元素、属性和实体的 DTD。它可以是内部 DTD 子集,扩展 DTD 子集或者是两者的结合。
- 文件必须遵守由 DTD 确定的文法。

第一个必要条件非常的显而易见,所有的有效文件都一定是结构良好的,这意味着它们遵循着常规 XML 语法。下面是确认文件是否结构良好的几个主要的必要条件:

- 必须有一个包含所有的其他元素的单一文件元素。
- 每一个元素必须有一个开始标记和对应的结束标记,或者有一个单一空标记。
- 所有的元素都是完全可嵌套的。
- 每个属性值必须放在引用记号中("或')。
- 使用的所有实体必须在 DTD 中声明(内部或外部)。

注意:所有的有效 DTD 文件都是结构良好的,但并不是所有的结构良好的文件都是有效文件。

所有的这些需求对你来说应该是很熟悉的了。然而,重申这些决不会有任何的损害,因为它们对于所有的 XML 文件都非常的重要。完整的规范位于 <http://www.w3.org/TR/>

1998/REC-xml-19980210。这个 XML 规范解释了“结构良好的绑定”，它清楚的指出了结构良好的文件的不同需求。

让我们回到有效文件上面。除了结构良好以外，文件的根元素必须与文件类型声明中提供的名称匹配。下面是阐明这一点的例子：

```
<! DOCTYPE animals SYSTEM "Animals.dtd" >
<animals>
  <!--Animal content goes here! -->
</animals>
```

在这个例子中，animals 根元素与在文件类型声明中提供的名称相匹配，这确保了文件有效性的一个规则。另一个文件有效性的重要规则要求有效的文件有一个声明用于文件中的元素、属性和实体的 DTD。换句话说，如果没有通过 DTD 的说明，文件中将什么都没有。整个 DTD 可以放置在文件的内部 DTD 子集中、扩展 DTD 中或是这两者的结合。

当然，只是拥有 DTD 并不足以保证文件有效——文件必须遵守 DTD。遵守 DTD 意味着支持在 DTD 中声明的元素的内容模型，以及使用适当的属性类型。

请理解上述的有效性需求是文件有效性的主要需求，它们在详述组成有效文件的细节方面并不是详尽的。正像 XML 规范包含清楚的指出结构良好文件的需求的结构良好绑定一样，它也包含有效性绑定。

注意：虽然创建用于某一类文件的 DTD 的要点是支持文件有效性，而这一点也很重要，就是要注意不有效的文件并不是毫无价值的。只要文件是结构良好的，它依然可以进行某种程度的处理和管理，这对于某些应用已经足够了。

3.2 理解元素和属性

在 DTD 中描述的基本部件是元素和属性，它们负责确定 XML 文件的逻辑结构。你可以把元素看作是信息的一个逻辑单位，而属性是这个信息的特征。换句话说，元素表示一个信息对象，而属性表示这个对象的性质。这是一个很重要的区别，因为在使用元素与使用属性构造信息时，这通常会造成混淆。

当你决定是使用元素还是使用属性构造信息时，一个好的处理方法是确认信息在整个文件的内容中的角色。通过将文件看作是一组信息对象，我们就可能将每个对象与元素相结合。任何剩余的信息就应该使用属性来构造。当你使用这种对象方法来构造文件数据时，这有助于将属性考虑为构造对象内部状态的对象特征。换句话说，属性不用来表示可以作为独立对象的信息。确定对象独立性的一个好方法是确认任何其他的对象是否与对象中的信息有直接的关系。如果有的话，你可能会希望将信息封装在对象的子元素中。如果没有的话，很有可能这个信息更适合作为对象属性。

当你评估构造信息的最好方法时，另一个有用的方法是确认信息的类型以及它将如何使用。这很重要，因为属性最终提供对信息的严格绑定，这一点非常的有用。属性可以绑定在一个预定义的可能值列表中，也可以有默认的值。元素内容没有绑定，非常的适合于长字符串的文本。让我们考虑一下属性超过元素的优势：

- 属性可以绑定在一个枚举值的预定义列表中。
- 属性可以有默认值。
- 属性有数据类型,虽然有些局限性。
- 属性非常的简练。
- 属性比元素更容易解析。

属性并不是完美的,下面是一些属性的缺点:

- 属性对于长文本字符串处理很不方便。
- 属性不能包含嵌套信息。
- 属性值中空白区域不能忽略。

这个有关属性缺陷的列表中没有什么让人吃惊的部分。当你考虑到了属性的优缺点后,很明显,你应该尽可能的尝试使用属性来构造信息。什么时候使用元素可以从属性缺点列表中明显的看出:如果一条信息是长文本字符串、需要在其中嵌套信息或是需要忽略空白区域,你就要将它放在元素中。除此之外,属性可以说是你的最佳选择。

关于属性一个重要的考虑就是混合内容,它是由解析字符数据和子元素共同组成的元素内容。我们最好尽可能的避免使用混合内容,因为它复杂化了文件的结构。避免使用混合内容,属性可以变得非常的便于使用。你可以将解析字符数据从混合元素提出,放入到一个属性中或是将混合子元素转化为属性。有混合内容的元素的一个例子就是 HTML 中的段落元素(p),它包含文本内容同时使用诸如 center 这样的属性。在下一节,你将会学到更多有关混合内容的知识。

元素和属性在 DTD 中使用元素声明和属性列表声明描述,这些将在本章的下面几节中进行介绍。由于每个 XML 文件都需要有一个根元素,从这个根元素开始构造 DTD 通常是很有益的。根元素然后分解为几个子元素,这些子元素再继续分解。最后,你就会到达不能再分的文本内容。这种自顶向下的 DTD 设计方法不仅从设计的角度上看是有益的,如果你打算将它共享的话,它还有助于其他人理解你的 DTD。

例如考虑一个包含电影收藏的文件。文件中的每个电影实体包含在 movie 元素中。它们有着多种不同的信息,这些信息需要通过元素或是属性来表示。

下面是一些你会将之与电影实体结合的信息列表:

- Title——电影的题目
- Type——电影的类型(浪漫、戏剧、科幻等等)
- Rating——电影的级别(G、PG、R 等等)
- Review——电影的数字级别(例如,1 到 5 之间的数)
- Year——电影最初发布的年份
- Writer——电影的作者
- Producer——电影的制片人
- Director——电影的导演
- Actor——电影中出现的演员
- Comments——有关电影的主要评论

认识到所有的这些信息都必须要在 movie 元素中说明,你能决定哪些信息更适合用元素表示而哪些信息更适合用属性表示吗?这可不是一个很容易的要求,而且这方面实际上没有绝对正确的答案。然而,我将尽力尝试。下面说明了我是如何组织这些信息的:

- 属性——Type、rating、review、year
- 子元素——Title、writer、producer、director、actor、comments

电影的类型,级别和评价非常适合于使用属性,因为它们都可以绑定在枚举列表中的可能值范围内。XML 设计的黄金法则是你在文件上能使用的绑定越多,它的内容就越结构化——而结构可是一个好东西。例如,你可以在 DTD 中指定电影级别的允许值是 G、PG、PG-13 和 R(是否将 X 和 XXX 作为你个人电影收藏的可接受的级别属性值,这一点我留给你自己决定)。

电影的年份和题目实际上既可以使用属性也可以使用元素来构造。我决定将年份作为属性的理由是它是非常短的信息。电影题目可以非常的长(Things to Do in Denver When You're Dead 就是一个好例子),因此我决定最好将题目构造为元素内容。

电影信息的其余部分(作者、制片人、导演、演员和评论)作为元素构造,因为它们可以包含长文本字符串。实际上,我将作者、制片人、导演、演员作为元素还有一个更重要的理由就是,它们可以在一个电影中出现多次。让我们考虑一个我个人非常喜欢的电影,古怪喜剧 Raising Arizona。这部电影由 Ethan Coen 和 Joel Coen 著作,这意味着 XML 文件中这部电影的实体需要 writer 元素出现两次。XML 严格禁止在一个元素中同名属性出现两次。因此,我们需要将这些信息以元素方式构造。

3.3 使用元素

元素在 DTD 中使用元素声明进行声明,元素声明有如下的格式:

`<! ELEMENT ElementName Type>`

元素的名称决定在 XML 文件中标记元素所使用的标记,它就是元素声明中相应的 ElementName。这个名称在单个 DTD 内容中必须是惟一的。元素的类型在 Type 中指定,这也可以被认为是元素的内容规范。XML 支持四种元素类型:

- Empty——元素不包含内容,但它可以包含属性。
- Element Only——元素只包含子元素。
- Mixed——元素包含子元素和字符数据的组合。
- Any——元素包含任何 DTD 允许的内容。

警告:元素的名称决不能包含和号(&),在任何情况下都不能以字符串 XML 的任何大小写形式(XML、xml、XmL 等等)开头。

下面的几节将更详细的探讨这些元素类型。

3.3.1 空元素

如它的名字所意味的,空元素不包含任何内容。如果一个空元素要包含信息,它要通过

属性来实现。实际上,空元素主要是一种通过属性集而不是文本内容表达 XML 内容的一种方法。空元素使用如下形式声明:

```
<! ELEMENT ElementName EMPTY>
```

下面是使用这个方法声明一个空元素的例子:

```
<! ELEMENT img EMPTY>
```

空元素在文件中以下面两种途径之一来定义:

- 使用开始标记/结束标记组合。
- 使用空标记。

在你有 SGML 知识背景的情况下,第一种方法就是你通常所见的元素形式,它用于较少扩展的 HTML。下面是一个使用开始标记/结束标记组合定义空元素的例子:

```
</img>
```

注意在标记间没有内容出现。如果任何内容出现,甚至只是一个空格,文件都将是非法的。定义空元素的一个更简明的方法是使用空标记。下面是一个使用空标记定义空元素的例子:

```

```

如你所见,空标记是开始标记和结束标记的一种合并形式。除了更简明之外,文件中空标记有助于使下面这一点更为清楚:元素是空的,因此不能包含内容。

3.3.2 只含子元素的元素

只含子元素的元素除了子元素外什么也不包含。你只需要指定只由子元素组成元素的内容模型就可以将元素声明为只含子元素。这通过在元素声明中列出在元素中出现的子元素来实现。下面是声明元素的内容模型需要的形式:

```
<! ELEMENT ElementName ContentModel>
```

内容模型本身使用专门元素声明标记和子元素名的组合指定。标记描述子元素和容器元素的关系。在内容模型中,子元素组织成序列或使用圆括号的选择组。序列中的子元素通过逗号分开,而选择组中的子元素通过管道符“|”分开。下面是用来确定元素的内容模型的不同元素声明标记:

- 圆括号——封装一个子元素的序列或是选择组。
- 逗号——分开序列中的各项,确定它们必须出现的次序。
- 管道符——分开选择组中的各个选项。
- 无符号——确定子元素必须出现且只出现一次。
- 问号——确定子元素或出现一次,或不出现。
- 星号——确定子元素可以出现任意次。
- 加号——确定子元素必须起码出现一次。

理解这些标记如何使用的最佳途径就是看一些例子。下面是一个你声明 resume 元素的

内容模型的例子：

```
<! ELEMENT resume (intro, (education|experience)+, hobbies?, references * )>
```

这个例子实际上使用每一种元素声明标记,考虑到它实际上并不是一个复杂的元素,这确实让人吃惊。简要说, resume 元素是只含元素类型的,这是显然的,因为它的内容模型只包含子元素。它技术上可以在内容模式中将子元素和字符数据混合,这一点将在本章的后面进行讨论。

Intro 元素在 resume 元素中必须且只能出现一次。Intro 元素后, education 和 experience 元素必须出现至少一次,这由在圆括号后面的加号决定。在圆括号中, education 元素必须出现一次,而 experience 元素必须出现至少一次并且可以出现多次。这里的意思是你可以列出你的教育程度——例如文理学士——后面跟着任意数量的相关工作经验。

在列出了你的教育程度和经验后,你可以使用 hobby 元素提到任何的业余爱好,这可以出现一次或不出现。这里的意思是你可以单个 hobby 元素中列出所有的你的业余爱好。最后的 references 元素可以出现任意次。

3.3.3 混合元素

混合元素包含字符数据和子元素。最简单的混合元素是声明只包括字符数据的元素。这种类型的元素有时指的是只有文本的元素。只有文本的元素使用如下形式声明：

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

你可能会想这个只有文本的元素的声明的最后部分看起来有点像内容模型。这是因为它本来就是。只有文本的元素的内容模型只由包含在圆括号中的 #PCDATA 组成,它指出元素包含解析字符数据。下面是一个简单的只有文本的元素声明的例子：

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

这个元素可以用来标记 XML 文件中的一个包含 CD 收藏的区域。下面是一个你如何在文件中定义 band 元素的例子：

```
<band>The Trashcan Sinatras</band>
```

注意：元素也可以包含非解析字符数据(CDATA),但你必须在解析字符数据中使用 CDATA 段。换句话说,元素的内容模型中不直接支持 #CDATA 类型。

在这里,你已经有了所有必要的元素声明技巧来创建可用 DTD 的框架。这里本来还要提到另一个元素类型,但现在也许是看一个制作中的 DTD 的好时候。清单 3.1 包含了电影收藏 DTD 的元素声明代码。

这个 DTD 看起来让人熟悉,虽然你在此前从没有看到过它。如果你回想一下,本章的前面我讨论了如何通过用元素和属性构造信息评定电影收藏 DTD 的结构。整个 DTD 是认识电影收藏数据模型的第一步。这个 DTD 的根元素是 movies,它作为一个容器,在其中你可以放入独立的电影列表。每个电影列表放置在 movie 元素中,它本身包含着一系列的子元素。

清单 3.1 电影收藏 DTD 的元素声明

```
<! ELEMENT movies (movie)+>
<! ELEMENT movie (title, writer+, producer+, director+, actor *, comments?)>
<! ELEMENT title (#PCDATA)>
<! ELEMENT writer (#PCDATA)>
<! ELEMENT producer (#PCDATA)>
<! ELEMENT director (#PCDATA)>
<! ELEMENT actor (#PCDATA)>
<! ELEMENT comments (#PCDATA)>
```

在 movie 元素中, title 元素必须且只出现一次。writer、producer 和 director 元素必须出现至少一次,但它们可以出现多次。这反映了电影的实际特性就是通常有起码一个作者、制片人和导演,有时还要多于一个。actor 元素可以出现任意多次,它反映了电影中有多个演员;记录片的特性就是它可以不包含任何的演员,而一些电影包含上百个演员。当然,你可能只对电影中列出的主要演员感兴趣。Movie 内容模型中的最后的元素是 comments 元素,它允许你列出有关这部电影的任何评论。

注意:在本章稍后,你将回到这个 DTD,通过加入属性支持来进一步描述 movies。

要谨记只含文本的元素实际上是不包含任何子元素的混合元素。

我希望将注意力转移到真正混合的混合元素上。除了一些必要的微妙差别,混合元素的声明与只含子元素的元素非常的类似。最为特别的是,混合元素的内容模型必须包含一个使用如下语法的重复选择列表:

```
<! ELEMENT Elementname (#PCDATA|Elementlist)* >
```

如你所见,选择列表开始处的 #PCDATA 指出混合元素可以包含字符数据。选择列表的剩余部分包含子元素和类似于只含子元素类型的内容模型。附加的 #PCDATA 标记可以散布在内容模型中,以指出字符数据的出现可以遍及子元素。还有一点很重要就是要注意星号(*)必须出现在内容模型的选择列表的后面,以指出整个选择组是可选的。下面是一个在选择列表中使用 #PCDATA 的例子:

```
<! ELEMENT memo (#PCDATA |title|subject)* >
```

警告:在混合元素的内容模型中,字符数据(#PCDATA)必须在选择组中首先指定,选择组本身必须后跟星号以声明它是可重复的。

要谨记文件中的混合元素可以看起来与空元素或是只含子元素的元素没有任何的不同,因为混合元素中的内容是完全可选的。例如,混合元素可以除了子元素之外什么也不包含,甚至它也可以声明具有包含字符数据的能力。纯粹从标记的角度来看,如果不在 DTD 中查询,你无法知道这样的元素是不是混合元素。类似的,混合元素可以根本不包含任何内容,这时它在文件中看起来就像是一个空元素。惟一的办法是你只能在 DTD 查找获得

元素最相近文字。

注意:虽然混合元素声明包含在混合元素中允许的子元素类型,但并不包含元素的次序或是它们出现的次数。

虽然混合元素提供了大量的灵活性,但它们缺乏只含子元素的元素的结构。因此,除了只含文本的元素以外,尽可能的避免使用混合元素是一个好的想法。你总是可以通过在只含文本的元素上声明属性获得混合元素的等价物。

3.3.4 ANY 元素

在 XML 中元素支持的最后一个类型被称作 ANY 元素,因为它使用标记 ANY 声明。ANY 元素实质上没有结构,在它之中可以包含字符数据、任何声明的元素类型或两者的混合。你可以将 ANY 元素考虑为有着广泛开放内容模型的混合元素。你可以使用如下形式声明 ANY 元素:

```
<! ELEMENT ElementName ANY>
```

由于它在结构上的欠缺,你应该不惜任何代价的避免使用 ANY 元素。典型情况下,ANY 元素只在开发 DTD 期间为了快速聚集元素以进行测试时才使用。实际上,ANY 元素在 DTD 产品中应该永远不出现。

3.4 使用属性

就像你在本章前面所学到的,属性用来指定元素的附加信息。在元素中,属性用来形成名称/值对,它从某方面描述了元素的特殊性质。属性使用属性列表声明在 DTD 中进行声明,它采用如下的形式:

```
<! ATTLIST ElementName AttrName AttrType Default>
```

如你所见,属性有名称(AttrName)和类型(AttrType),以及默认值(Default)。属性的默认值指的是一个值或一个指出属性用法的标记。下面有四种不同类型默认值,你可以在属性的 Default 中指定:

- #REQUIRED——属性是必需的。
- #IMPLIED——属性是可选的。
- #FIXED VALUE——属性有一个确定值。
- default——属性的默认值。

#REQUIRED 值指出属性是必需的,这意味着如果你看到了这个元素,你就必须定义这个属性。#IMPLIED 值指出属性是可选的,意味着使用这个元素时并不一定要定义这个属性。#FIXED 属性用来给属性设置一个固定值,它有效的制作一个包含常量信息的属性。你必须提供在声明属性时在 #FIXED 标记后提供固定属性值。当你声明一个默认的属性值时,属性将在没有在元素中明确设置的情况下采用这个值。这有利于确定属性值的一般情况。

注意:虽然属性列表并不需要在 DTD 中的特定位置上说明,一般惯例是将它们立即放置在他们所属的元素的声明的下面。这就使得理解 DTD 的结构以及元素和它们的属性之间的关系更加的容易。

返回到属性列表声明,除了属性默认值外,你还必须指定属性的类型。XML 支持十种不同的属性类型,它们的概述如下:

- CDATA——非解析字符数据。
- Enumerated——一系列字符串值。
- NOTATION——位于 DTD 中其他位置的符号声明。
- ENTITY——外部二进制实体。
- ENTITIES——通过空白空间分开的外部二进制实体。
- ID——惟一标识。
- IDREF——对位于 DTD 中其他位置的外部二进制实体的引用。
- NMTOKEN——由 XML 标记字符(字母、数字、句点、破折号、冒号和下划线)组成的名称。
- NOTOKENS——多个由 XML 标记字符组成的名称。

这些字符类型可以分为三个组:字符串,枚举和标记。下面的几节将详细的讨论这些不同分类的详细内容,以及如何声明不同类型的属性。

3.4.1 字符串属性

到目前为止,字符串属性是最为普遍使用的属性,它通过 CDATA 类型表示。CDATA 类型指出属性包含简单的文本字符串。下面是声明简单的 CDATA 属性的例子,它必须在 player 元素中定义:

```
<! ATTLIST player team CDATA #REQUIRED>
```

在这个例子中,比赛者所在的队是 player 元素所必需的字符数据属性。如果你希望使 team 属性可选,你可以使用 #IMPLIED 标记,就像这样:

```
<! ATTLIST player team CDATA #IMPLIED>
```

注意:有关 CDATA 属性类型一个潜在的容易搞混的问题是标记 CDATA 的使用。CDATA 属性与用来指出开始字符数据段的 CDATA 标记非常的不同。如果你回想一下,CDATA 片段用来定义不能被 XML 处理函数解析的字符数据,它允许你包含不能被解析的标记代码。CDATA 属性类型只是指出属性包含字符数据。与在 CDATA 段中的字符数据不同的是,CDATA 属性数据可被 XML 处理函数处理,这意味着你可以把它看作是解析字符数据。

有一点很重要就是要理解 XML 解析器如何处理字符串属性类型,因为这与如何处理其他的类型有所不同。XML 解析器在处理 CDATA 属性值时保存大部分的空白区域。不保存的空白区域包括行结束符,它由空格替换。字符属性中将行结束符转化为空格的过程称作属性值标准化。其他类的数据也要标准化,但使用不同的方法。例如包含字符引用和解析内

部实体应用的属性通过标准化过程扩展。

3.4.2 枚举属性

枚举属性由那些值包含在预定义文本字符串列表的属性组成。除了可接受的属性值包含在由属性列表声明中提供的列表以外,枚举类型类似于 CDATA 类型。使用前面的包括 player 元素的例子,你可以提供枚举属性,用来指定曲棍球运动员的位置:

```
<! LTTLIST player position (center | forward | defenseman | goalie) "center">
```

这个例子阐明了枚举属性如何被用来提供可选的常数列表。你在使用 POSITION 属性时,你需要从这个列表中选择一个值。如果你选择不定义这个属性,它将采用默认值 center。

枚举属性的特殊类型是符号属性,它指定在 DTD 中的其他位置声明的符号列表。通过使用关键字 NOTATION 预先声明的属性列表声明一个符号属性,就像这样:

```
<! ATTLIST image format NOTATION (gif|jpeg) #REQUIRED>
```

注意 #REQUIRED 标记被指定以指出属性是必需的。枚举属性也是可选的,这种情况下,你指定 #IMPLIED 标记作为默认值。

3.4.3 标记属性

在 XML 中属性支持的最后一个是标记属性,它被 XML 解析器作为标记处理的属性。当属性作为标记处理的时候,解析器将把所有的邻近的空白区域转化为一个空格符并消除所有的前导和后续空白区域。这个过程称为属性值标准化。除了消除标记属性值中的大多数空白区域外,XML 解析器还根据声明的属性类型验证值。下面是标记属性类型: ENTITY、ENTITIES、ID、IDREF、IDREFS、NMTOKEN 和 NMTOKENS。

ENTITY 和 ENTITIES 类型用来应用实体。例如,图片是二进制实体的经典引用,这种情况下,你可以使用 ENTITY 属性值将图片与元素类型关联。ENTITIES 类型类似于 ENTITY,但它允许你指定通过空格分开的实体列表。下面是一个使用 ENTITY 属性类型设置实体应用的例子:

```
<!ATTLIST logo image ENTITY #IMPLIED>
```

ID、IDREF 和 IDREFS 属性类型都与惟一标识有关。ID 类型已可以用来指定文件中的一个元素的惟一标识。下面是使用 ID 属性类型设置 part 类型元素的惟一标识:

```
<! ATTLIST part id ID #REQUIRED>
```

警告:给定的元素类型只能设置一个类型为 ID 的属性。同样,ID 属性必须使用默认 #IMPLIED 或 #REQUIRED 声明。

IDREF 和 IDREFS 类型允许你指定对 ID 的引用。这些类型的使用类似于 ID 类型,但要谨记它们自身不是惟一的。它们只是引用惟一标识。

NMTOKEN 和 NMTOKENS 属性类型用来指定属性包含的名称标记值。名称标记值就是有着某种附加绑定的文本字符串。一般而言,名称标记由一个名称组成,这意味着它不能包含空白区域。更特别的是,名字标记可以由数字文字字符加上如下字符组成: .、-、_、

和：。

3.4.4 声明多重属性

迄今为止,我只向你展示了声明独立属性的例子。很明显,元素可以包含多个属性。虽然你可以提供包含独立属性声明的多个属性列表声明,而在一个属性列表中列出所有给定元素的属性会更加方便。要做到这一点,只要在属性列表声明中依次列出属性就可以了。下面是一个在一个属性列表中声明多个属性的例子:

```
<! ELEMENT movie (title,writer+, producer+, director+, actor * ,comments?)>
<! ATTLIST movie
type (drama | comedy |adventure |sci-fi |mystery | horror | romance |
documentary) "drama"
rating (G | PG | PG-13 | R | X) "PG"
review(1 | 2 | 3 | 4 | 5) "3"
year CDATA #IMPLIED>
```

这个例子使你回忆起了什么? 我已经在本章的前面你所学的电影收藏例子中提供的属性。在这里,type、rating、review 和 year 属性被声明以进一步描述电影的的性质。注意如何使用枚举类型提供预定义值列表,它有助于改善元素的结构。

3.5 用 DTD 创建有效的文件

在下一章讲解一些附加的 DTD 结构,比如实体和标记。现在是看如何创建一个基于 DTD 的文件的好时候了。清单 3.2 包含电影收藏 DTD 的完整代码,这你已经在本章的各部分到了接触中,它存储在文件 Movies.dtd 中。

清单 3.2 电影收藏 DTD,使用元素和属性实现

```
<! ELEMENT movies (movie)+>
<! ELEMENT movie (title, writer+, producer+, director+, actor * , comments?)>
<! ATTLIST movie
type (drama | comedy |adventure |sci-fi |mystery | horror | romance |
documentary) "drama"
rating (G | PG | PG-13 | R | X) "PG"
review(1 | 2 | 3 | 4 | 5) "3"
year CDATA #IMPLIED>
<! ELEMENT title (#PCDATA)>
<! ELEMENT writer (#PCDATA)>
<! ELEMENT producer (#PCDATA)>
<! ELEMENT director (#PCDATA)>
<! ELEMENT actor (#PCDATA)>
<! ELEMENT comments (#PCDATA)>
```

这个 DTD 的所有代码在本章的各部分都已经讲解到了。虽然如此,这个清单还是有用

的,因为它展示给你元素和属性是如何共同形成一个完整的 DTD 的。

更重要的一点是基于这个 DTD 的文件的创建。由于大多数的 DTD 使用自顶向下的方法设计,都是从根元素开始的。以同样的方式编写文件是有益的。要谨记,在文件类型声明之后,你将希望在所有 XML 文件的开始处加入一个 XML 声明。对于电影收藏文件来说,它需要引用包含在 Movies.dtd 文件中的扩展 DTD 子集。清单 3.3 包含基于清单 3.2 中的电影收藏 DTD 的电影收藏文件的代码。

清单 3.3 基于电影收藏 DTD 的电影收藏 XML 文件

```
<? xml version = "1.0" standalone = "no"? >
<! DOCTYPE movies SYSTEM "Movies.dtd" >

<movies>
  <movie type = "comedy" rating = "PG-13" review = "5" year = "1987">
    <title>Raising Arizona</title>
    <writer>Ethan Coen</writer>
    <writer>Joel Coen</writer>
    <producer>Ethan Coen</producer>
    <director>Joel Coen</director>
    <actor>Nicolas Cage</actor>
    <actor>Holly Hunter</actor>
    <actor>John Goodman</actor>
    <comments>A classic one-of-a-kind screwball love story. </comments>
  </movie>

  <movie type="comedy" rating="R" review="5" year="1988">
    <title>Midnight Run</title>
    <writer>George Gallo</writer>
    <producer>Martin Brest</producer>
    <director>Martin Brest</director>
    <actor>Robert De Niro</actor>
    <actor>Charles Gridin</actor>
    <comments>The quintessential road comedy. </comments>
  </movie>

  <movie type="mystery" rating="R" review="5" year="1995">
    <title>The Usual Suspects</title>
    <writer>Christopher McQuarrie</writer>
    <producer>Bryan Singer</producer>
    <producer>Michael McDonnell</producer>
    <director>Bryan Singer</director>
    <actor>Stephen Baldwin</actor>
    <actor>Gabriel Byrne</actor>
    <actor>Benicio Del Toro</actor>
    <actor>Chazz Palminteri</actor>
    <actor>keVin Pollak</actor>
    <actor>KeVin Spacey</actor>
    <comments>A crime mystery with incredibly intricate plot twists. </comments>
  </movie>

  <movie type = "sci-fi" rating="PG-13" review="4" year="1989">
    <title>The Abyss</title>
    <writer>James Cameron</writer>
    <producer>Gale Anne Hurd</producer>
```

```
<director>James Cameron</director>
<actor>Ed Harris</actor>
<actor>Mary Elizabeth Mastrantonio</actor>
<comments>A very engaging underwater odyssey.</comments>
</movie>
</movies>
```

如你所见,在它定义的元素和元素的引用方面,这个文件都严格遵守 DTD。此外, movie 元素的 type、rating 和 review 属性遵守在 DTD 中的枚举属性声明中声明的可用属性。在这个文件中列出了四个电影,DTD 允许你在 movies 元素中加入你希望数目的 movie 元素。你可以在一个 XML 文件中创建一个全面的电影收藏数据库,这是非常可行的。

3.6 总 结

XML 文件的结构通过文件类型声明确定,它包含在文件类型定义或是 DTD 中。DTD 可以作为一类文件的模板,它由设置这类文件的语法的标记声明组成。DTD 对于 XML 文件的设计和开发十分的重要。因为它们提供了验证文件的方法。

第4章 进一步深入 DTD

虽然文件的逻辑结构非常的重要,但实际上是物理结构负责文件内容的存储。本章通过研究 DTD 在设置文件物理结构的过程中所扮演的角色进一步深入学习 DTD。更重要的是,你将学习到实体是如何用来作为文件数据的存储容器的。我们还将涉及到字符和实体的关系。

一个与实体关系最近的 DTD 就是表示法,它用来设置实体引用所引用的数据对象的类型。表示法非常的重要,因为它们帮助 XML 处理函数分辨数据类型,这样数据就可以被智能的处理了。在本章中,你将学习到你知道的有关表示法的所有知识,以及一些你在描述文件物理结构时顺便涉及的其他东西。

4.1 实体和文件结构

XML 文件被分成名为实体的多个存储单位,它们通常使用惟一的名称来确认。一个不用名称确认的特殊实体是文件实体,它是文件的顶级实体。文件实体通常包含整个文件。这可能有点奇怪,但当你考虑到实体被分为子实体,而子实体通常还要继续分解时就不会有这样的感觉了。实体的分解一直进行,直到你达到除了内容没有其他东西的时候就结束了。

实体定义文件的物理结构,因此它们经常与文件相关。实际上,许多实体与位于文件系统某处(在本地或网络上)的物理文件相关联。然而,实体并不是必须在任何方式下都要与文件相关联的。实际上,实体可以与一个数据库记录甚至一块内存区域相关联。有关实体,需要记住的最主要的方面就是它们声明了一个存储单位。存储的媒介并不非常的重要。

另一个不命名的实体就是文件的外部 DTD 子集,如果它们存在的话。例如,你在第3章“DTD 基础”中看到的 Movies.dtd 外部 DTD。就是一个实体它与在文件中定义的根文件元素 movies 有着相同的作用。下面是一个电影集文件的节选,这部分例证了外部 DTD 和根文件元素之间的关系:

```
<? xml version="1.0" standalone="no"? >
<! DCTYPE movies SYSTEM "Movies.dtd">
<movies>
  <!--Document markup -->
</movies>
```

如你所见,文件类型声明涉及到了两个我们提到的实体:movies 根元素和 Movies.dtd 外部 DTD 子集。我们也可以把 Movies.dtd 外部 DTD 子集移动,直接放入到文件类型声明中,这种情况下,它就不再被认为是一个实体。这里的关键之处就是外部 DTD 子集的分离存储使得它成为一个实体。

当我学习 XML 时,我为了文件、元素和 DTD 子集都可以被认为是实体这一事实而努力奋斗。对我来说,几乎 XML 文件中的任何东西都可以被认为是一个实体,这就使得明确

这个术语的定义变得很困难。理解实体的最好方式就是带着问题来评估代码或数据的物理存储。如果 XML 代码和数据物理上被放置在分离的位置,它就是一个实体。在前面电影集 DTD 的例子中,文件的标记声明都包含在一个外部 DTD 子集中,而它被存储在物理分离的位置。因此,显然外部 DTD 子集是一个实体。

4.2 字符和实体

在进入实体如何声明和引用的细节之前,有一点很重要,就是理解字符是如何用在实体中的。更重要的是,你需要牢固掌握字符编码,这是一种通过位 Schema 方式表示字符的方式。这很重要,因为 XML 有足够的灵活性来支持多种不同的字符编码。XML 字符编码的基础是 ISO/IEC 10646 统一字符编码标准,它在使用多语言字符上提供了很强的灵活性。

除了统一字符编码 ISO/IEC 100646 标准以外,你也可以使用 ISO 8859 标准和 JIS X-0208-1997 标准。如果这些字符编码标准对你并没有更多的意义,你可能希望依靠一种统一字符编码。你甚至几乎不用考虑你的 XML 文件所使用的字符编码,但如果你需要处理非西文语言的话,这一点就是非常重要的问题。

你使用字符编码声明来声明文件的字符编码,这是 XML 声明中出现在文件开始处的部分。实际上字符编码声明看起来像是一个 XML 声明的属性,如下面的代码所示:

```
<? xml version="1.0" encoding="UTF-8"? >
```

值 UTF-8 赋给了字符代码声明,它指定文件使用 UTF-8 字符编码,这是统一字符编码 ISO/IEC 100646 标准的子集。UTF-8 和 UTF-16 是最常用的字符编码,而实际上只有它们两个是 XML 处理函数保证支持的。XML 处理函数当然还支持其他的字符编码,但你只能指望它们支持 UTF-8 和 UTF-16。要获得有关统一字符编码标准的更多信息,请访问统一字符编码的 Web 站点 <http://www.unicode.org>。

注意:在你没有使用字符编码声明进行明确声明的情况下,XML 默认使用 UTF-8 字符编码。

下面是 XML 实体可用的字符编码清单:

- UTF-8(统一字符编码)
- UTF-16(统一字符编码)
- ISO-10646-UCS-2(统一字符编码)
- ISO-10646-UCS-4(统一字符编码)
- ISO-8859-1 到 ISO-8859-9
- ISO-2022-JP(JIS X-0208-1997)
- EUC_JP(JIS X-0208-1997)
- Shift_JIS(JIS X-0208-1997)

前四种是统一字符编码 ISO/IEC 10646 标准的变体。其中,UTF-8 和 UTF-16 是 XML 实体最常用的编码。虽然这两种编码可以支持统一字符编码的全部字符,UTF-8 只能一次对应 255 个字符。大多数情况下,你可以使用 UTF-8 字符编码,与标准 ASCII 字符相映射。

然而,如果你需要使用所有的统一字符编码字符集,最好是使用 UTF-16 编码。下面是一个如何使用字符编码声明指定 UTF-16 作为文件的字符编码的例子:

```
<? xml version="1.0" encoding="UTF-16"? >
```

不论你在 XML 文件中使用什么编码来为字符编码,你需要有一种方法来通过它们的位所表现的数值指定字符。每个所给编码 Schema 中的字符都有这样的用作字符引用的数字值。字符引用在你尝试输入一个使用键盘或从其他的输入设备上不能输入的字符时是非常方便的。例如,版权符 © 就是只能使用字符引用指定字符的例子。

XML 支持两种指定字符引用的技术:

- 十进制引用(基数为 10)
- 十六进制引用(基数为 16)

十进制字符引用依靠一个十进制数作为字符数字引用的基础。十进制引用使用和号和英镑号(&#)指定,它们后面跟着数字,并以分号结束。因此一个完整的十进制字符引用形式如下所示:

&#Num;

使用这个形式,十进制数字使用 Num 指定。下面是一个十进制字符引用的例子:

©

这个字符引用确定了与十进制数 169 相关的字符,这正好就是版权符。下面是版权符在其他字符数据的内容中使用的字符引用:

©1999 Michael Morrison

指定字符引用的另一种方法就是使用十六进制数据(基数为 16)进行引用。编程人员习惯于使用十六进制数,因为它们计算机使用的低级二进制(基数为 2)数字系统的排列。除了数字前要加 x 之外,十六进制引用的指定类似于十进制引用。下面是用来指定十六进制字符引用的形式:

&#xNum;

使用这种形式,版权字符通过十进制值 169 调用,如下所示:

©

如果你并不熟悉使用十六进制数字,它们在起初看起来有些奇怪,因为它们使用十进制数码 0—9,并使用字母 A-F 表示数字 10—15。例如,十进制数 42 在十六进制中表明为 2A;十六进制的 A 表示十进制的 10,而十六进制的 2 表示十进制数 32(2×16)。大多数的编程语言通过加上前缀 x 表示十六进制数,因而在 XML 表示十六进制字符引用时这无疑不用改变。

使用十进制与十六进制字符引用的对比结果最终是一个个人喜好问题。XML 完全支持这两种方法,因此从字符编码方面看这一点确实无关紧要。

4.3 使用实体

就像你所知道的一样,实体是用来保存数据的存储容器,它们是构造 XML 文件的物理构造块。每个 XML 文件有起码一个实体作为整个文件的基本实体。毫无疑问,这个实体就是文件实体。当处理 XML 文件的时候,XML 处理函数从文件实体开始,因为它构成了文件层次结构的根。

XML 在很广泛的范围下使用术语“实体”,因此实体是一个信息物理块的想法很容易被误解。虽然将实体指作一个存储的物理单元是正确的,这个存储本身可以通过一些看起来很怪异的方法实现。例如,对 XML 内容来说,非常可能通过应用方便的将其生成并通过网络发送给另一个应用。在这种情况下,XML 内容本身被认为是一个实体,即使网上的物理存储中介是数据包。

对于大多数 XML 文件的成分来说,实体必须在使用前被声明。一个实体声明有一个惟一实体名和与这个名称相关联的数据块组成。下面是一些你可以与实体名相关联的典型类型。

- 文本字符串
- DTD 的节
- 包含 XML 文本的文件的外部调用
- 包含二进制数据的文件的外部调用

注意:在实体声明中需要惟一名称的例外情况是文件实体和外部 DTD 子集,它们是未命名实体。

如你所见,实体在继承可以通过它进行存储的数据类型方面非常的灵活。虽然实体的特定内容变化范围非常的广,在 XML 文件中要使用两种基本的实体类型:解析和非解析。解析实体存储被 XML 解析器解析的数据,这意味着它们只包含文本。另一方面,非解析实体存储没有解析的数据,它们可以是文本或二进制数据。

由于解析实体使用 XML 解析器解析,它们在与它们所应用的文件内容合并时结束。换句话说,解析实体被直接插入到文件中,就好像它们从开始就是文件的一部分。解析实体的内容有时指的是实体的替换文本。非解析实体在这种情况下不能操作,因为 XML 解析器不能解析它们。因此,非解析实体从不直接有文件结合。

你也许会想 XML 解析器可以在非解析实体不是二进制数据的情况下解析它们。然而,你不能假定一个非解析的文本实体包含 XML 代码,这是 XML 解析器可以解析它的惟一途径。换句话说,或许一个非解析文本实体除了包含 XML 代码的文本外,还可能包含诸如 HTML 或 RTF 代码的文本。

如果 XML 解析器不能解析一个非解析的实体并把它融合到文件中,它是如何使用这种实体作为文件数据的呢?这个问题的解答在于表示法,这是一个用来向 XML 处理机确认实体类型的 XML 结构。一个非解析实体的辅助应用的优秀例子是图片浏览器,这主要是与 GIF 或 JPEG 图像实体相关。

虽然在本章后面的题为“在文件中引用实体”的小节中你将学到更多有关实体引用的知识。这里还是很值得简略的提一下解析和非解析实体间的一个重要区别。解析实体使用实体引用在文件中进行调用,而非解析实体通过作为元素的 ENTITY 或 ENTITIES 属性的值进行调用。这种调用属性的区别只有在解析属性用它们的替换文本在文件中进行替换时才会出现。另一方面,非解析实体不能混入到文件当中,这就是为什么它们要作为专门属性进行调用的原因。

你将在本章的后面学到更多有关实体声明和调用的知识。现在,我希望集中考虑一下两种解析实体:

- 一般实体——用在文件内容中。
- 参数实体——只用在 DTD 中。

4.3.1 一般实体

一般实体是设计在文件内容中使用的解析实体。如果你有一个你希望作为一个复用文件数据来分离的文本字符串,一般实体正是你所需要的。在你可以在文件中引用实体前,你必须使用实体声明来声明它。一般实体的声明采用如下的格式:

```
<! ENTITY EntityName EntityDef >
```

如你所见,声明一般实体非常的直接。实体的唯一名称在 EntityName 中指定,而替换文本在 EntityDef 中指定。除非你声明过,否则你不能引用这个实体,而实体的声明必须放置在 DTD 中。下面是一个一般实体声明的例子:

```
<! ENTITY nickname "T-Bone" >
```

一旦实体被声明,你就可以在文件内容的任何地方自由的使用它了。一般实体通过使用夹在和号(&)和分号(;)之间的实体名称进行引用,如下所示:

```
&EntityName;
```

下面是一个引用 nickname 实体的例子:

```
My name is George but my friends call me &nickname;
```

在这个例子中,nickname 实体的内容在引用发生的情况下被替换进去。

使用一般实体存储在文件中重复使用的文本是非常方便的,这些文本还可以改变。在文本修改的情况下,我们可以只对单个位置进行修改,而不用遍历内容文件进行多处修改。下面是一个实体的例子,这个实体会在文件中被重复的使用:

```
<! ENTITY notice "&#169;1999 Michael Morrison " >
```

这个实体包含版本信息,它用在文件中作为版本归属的法律信息。在一个文件中多处输入这样的信息真是没有意义,因此一个一般实体的使用就变得非常有意义。

虽然你要在使用之前明确的声明一般实体,但是还有一些预定义的实体可以在所有的 XML 文件中使用。更重要的是,下面的实体是自动声明的,因此在每个 XML 文件中均可用:

- &——和号(&)
- '——撇号(')
- "——引号(")
- <——小于号(<)
- >——大于号(>)

这些实体都是预定义的,它们是使用前必须声明所有实体这一规则的例外情况。然而,除了这五个实体,所有的实体在文件中使用前都必须声明。

警告:虽然这五个一般实体在 XML 中是预定义的,它们并没有在最初的 SGML 规范中被预定义。名为 WebSGML 的修订的 SGML 规范包含了这些预定义的实体。然而,如果你计划使用 SGML 软件(不能使用新的 WebSGML 规范)的话,你应该明确的声明这些实体。

4.3.2 参数实体

XML 支持的另一种解析实体就是参数实体,这是只能在 DTD 中使用的一般实体。参数实体通过存储一般使用声明的组件来帮助模块化 DTD 结构。例如,你也许使用参数实体存储在多个元素间共享的一般属性的清单。与一般实体类似,在 DTD 中使用之前,你必须声明参数实体。参数实体声明有如下的格式:

```
<! ENTITY %EntityName EntityDef>
```

参数实体声明与一般实体声明非常的类似,惟一的不同就是存在百分号(%)。参数实体的惟一名称在 EntityName 中指定,而实体内容在 EntityDef 中指定。要谨记参数实体只可以在 DTD 中使用。下面是一个参数实体声明的例子:

```
<! ENTITY % dimension "length, width, height?">
```

这个参数实体描述了可以在 DTD 的元素中引用的内容模型的一部分。参数实体通过使用夹在和号(&)和分号(;)之间的实体名称进行引用,就像下面所显示的格式:

```
%EntityName;
```

下面是一个在假设结构的 DTD 中引用 dimensions 参数实体的例子:

```
<! ELEMENT floor (%dimensions;)>  
<! ELEMENT wall (%dimensions;)>  
<! ELEMENT ceiling (%dimensions;)>
```

虽然参数实体在给 DTD 的结构提供模块化方面非常的有用,你仍应该小心的使用它们,因为它们还会带来不必要的复杂性。在你创建可互相引用的参数实体的层时这一般就会成为问题,它会使得 XML 处理函数难以断定实体如何解析。

人力资源管理标记语言(HRMML, Human Resource Management Markup Language)是非常有效的使用参数实体来模块化其 DTD 结构的 XML 应用的优秀范例。你将在本章后面的主题“使用外部参数实体”以及第 39 章“使用 HRMML 管理人才资源”中学习 HRMML 和它的模块化 DTD 的知识。

4.4 外部实体

实体的物理位置对于确定它是如何在 XML 文件中被引用方面非常的重要。迄今为止,你已经学习了一般实体和参数实体,这两个都被认为是内部实体,因为它们的内容直接在 XML 文件中声明。内部实体的值直接的包含在实体声明中。实体内容也可以放置在文件的外部,这种情况下的实体称为外部实体。外部实体依旧在文件中声明,但它们引用其他文件作为它们的实际内容。

注意:从定义上来看,任何非内部的实体都必然是外部的。这意味着外部实体存储在实体被声明的文件之外。

与内部实体不同,内部实体都是解析实体,外部实体可以是解析或非解析的。解析外部实体必须包含 XML 内容,而非解析外部实体可以包含文本或二进制数据。非解析外部实体通常是诸如图片这样的显然不能被 XML 处理机解析的二进制文件。非解析外部实体在它们的实体声明中使用关键字 NDATA 确定。NDATA 简单地指出实体内容不是 XML 数据。

外部实体声明与内部实体声明有很大区别,因为它必须引用外部文件。与外部实体相关的文件可以用一或两种途径指定,这取决于它们是放在本地文件系统中还是放在公共可用网络上。下面的关键字用来辨别这两种指定外部实体文件的途径:

- SYSTEM——文件放在本地文件系统中或是在网络上。
- PUBLIC——文件是放置在公共可访问位置的公共域文件。

警告:当你指定外部实体的位置的时候,你必须使用 SYSTEM 关键字确定在本地系统和网络上的文件。PUBLIC 关键字可选但必须在不使用 SYSTEM 关键字的情况下使用。

外部实体的文件通过统一资源标识符 (URI, Uniform Resource Identifier) 指定,它与我们非常熟悉的 URL 非常的类似。URI 是 URL 的更一般形式,它可以用来确定文件,也可以用来确定其他资源。你可以使用相关的 URI 指定文件,这比列出文件的全路径稍微简单一些。XML 需要相关 URI 在实体声明中输入相关的资源文件。下面是一个使用相关 URI 声明 JPEG 图片实体的例子:

```
<! ENTITY jphoto SYSTEM "photo.jpg" NDATA JPEG>
```

在这个例子中,photo.jpg 文件需要放置在本地文件系统中与包含实体声明的文件所在目录相同的位置。注意 NDATA 关键字用来指出实体不包含 XML 数据;在这种情况下,实体是二进制实体。而外部实体的类型被指定为 JPEG。XML 不支持任何诸如 JPEG 的内置二进制实体类型,即使它们是知名的图片格式也是如此。你必须使用表示法来确定非解析实体的实体类型。在本章稍后你将学到有关表示法的知识。

4.4.1 引用非解析实体

由于不可能将二进制实体的内容以文本方式直接嵌入到文件中,所以二进制实体通常是外部存储的。二进制(非解析)实体的引用也不同于解析实体。更重要的是,你必须使用类

型为 ENTITY 或 ENTITIES 的属性引用非解析实体。下面是一个这样属性声明的例子：

```
<! ELEMENT person EMPTY>
<! ATTLIST person
  name CDATA #REQUIRED
  photo ENTITY #IMPLIED>
```

在这段代码中, photo 属性被指定为 ENTITY 类型, 这意味着你可以给属性用非解析实体赋值。下面是一个如何在文件内容中实现的例子：

```
<person name="Jack Tors" photo="jtphoto">
```

在这行代码中, 二进制实体 jtphoto 赋给了 photo 属性。即使二进制实体已经被适当的声明并赋值了, XML 应用在没有表示法声明的情况下还是不知道如何处理它, 这个问题你马上就会解决的。

4.4.2 使用外部参数实体

在本章的前面, 我提到过参数实体可以被用来创建模块化的 DTD。虽然参数实体在作为内部实体声明时非常又有, 但它作为外部实体的功能同样非常的强大。换句话说, 你可以声明 DTD 的整个部分为一个外部参数实体, 并把它包含到需要它的主 DTD 中。HRMML 应用使用这种方法创建两个共享公用外部参数实体的模块化词表(Resume 和 Job Posting)。清单 4.1 包含了 HRMML Resume 的 DTD, 它非常的依赖于外部参数实体。

清单 4.1 HRMML Resume DTD

```
<! ENTITY % SpecialMod "jobtitle | joblocation | address | voice | fax |
  pager | email | employername | skillsqualif | experiencequalif |
  softwarequalif | progmlangqualif | educationqualif | licensequalif |
  certificationqualif | equipmntqualif | otherqualif | hardwarequalif ">
<! ENTITY % commonmod SYSTEM "hr-comm. mod">
%commonmod;
<! ENTITY % hiringorgmod SYSTEM "hr-org. mod">
%hiringorgmod;
<! ENTITY % postdetailmod SYSTEM "hr-pstd. mod">
%postdetailmod;
<! ENTITY % jobdescmod SYSTEM "hr-jobd. mod">
%jobdescmod;
<! ENTITY % phonemod SYSTEM "hr-phon. mod">
%phonemod;
<! ENTITY % addressmod SYSTEM "hr-addr. mod">
%addressmod;
<! ENTITY % resumemod SYSTEM "hr-resm. mod">
%resumemod;
<! ELEMENT resumedatabase (resume+)>
<! ELEMENT resume (resumeprolog?, resumebody)>
<! ATTLIST resume %attrglobal;>
```


你将在第 39 章中学到有关这个 DTD 结构和相关外部参数实体的更多知识。但还是值得看一下这些外部参数实体以了解一下 DTD 是如何被划分的。清单 4.2 包含了 addressmod 参数实体的代码,它存储在文件 hr-addr.mod 中。

清单 4.2 HRMML 模块 hr-addr.mod

```

<! ELEMENT joblocation (#PCDATA | comment) * >
<! ATTLIST joblocation %attrglobal; >

<! ELEMENT address (addressline+, city, (state | province), postalcode,
country?) >
<! ATTLIST address %attrglobal; >

<! ELEMENT addressline (#PCDATA) >
<! ATTLIST addressline %attrglobal; >

<! ELEMENT city (#PCDATA) >
<! ATTLIST city %attrglobal; >

<! ELEMENT state (#PCDATA) >
<! ATTLIST state %attrglobal; >

<! ELEMENT province (#PCDATA) >
<! ATTLIST province %attrglobal; >

<! ELEMENT postalcode (#PCDATA) >
<! ATTLIST postalcode %attrglobal; >

<! ELEMENT country (#PCDATA) >
<! ATTLIST country %attrglobal; >

```

注意这段代码主要由直接在 DTD 中出现的元素声明组成。通过将这些声明移入到外部文件中,HRMML 创造者就使得在两个词表间共享声明成为了可能。这个模块化方法不止对于一般组织有好处,它也使得对这两个 DTD 的管理更为的高效。当你开发 DTD 时,你可以自由的将所有的元素、属性和实体声明加入到单个 DTD 中,或者你可以使用参数实体将它们分成模块。这些方法的最终结果都是相同的,因为不论它们在哪里被声明,参数实体都是要插入到 DTD 中的。

4.5 声明表示法

从定义上讲,非解析实体不能通过 XML 解析器解析,因此需要提供有关非解析实体的附加信息。这个信息包括实体的类型,以及用来处理实体内容的辅助信息。表示法用来指定非解析实体的这些信息,它是所有非解析实体所必需的。下面是一个描述 GIF 图片格式的表示法的例子:

```
<! NOTATION GIF SYSTEM "GIF">
```

在这个例子中,表示法的名称是 GIF,辅助信息也是 GIF。它正是所期望的,XML 应用可以以某种方式使用辅助信息询问系统 GIF 类型,以解决如何浏览 GIF 图片的问题。另一方面,你可以从表示法中获得更详细的信息并指示一个应用程序,就像这样:

```
<! NOTATION GIF SYSTEM "Iexplore.exe">
```

这行代码将 GIF 图片与 Internet Explorer 网络浏览器应用程序(IExplore.exe)相联系, 这样 XML 应用可以使用 Internet Explorer 来浏览 GIF 图片。

4.6 在文件中引用实体

现在,你应该对有关实体如何声明和应用有了非常扎实的理解。然而为了有助于在内容中使用实体,因此我希望完成一个有关如何使用二进制实体的示例。我将使用的例子就是你已经在本章和前一章中多次见到的电影集文件。改进这个文件的一种方式就是提供一个用来指定在收藏中的每个电影的电影海报的方法。这可以通过使用二进制实体方便的完成。

在电影集文件中支持二进制实体的第一步是声明一个具有 ENTITY 类型属性的 poster 元素。下面是这个元素和属性的声明,它将放置在电影集 DTD 中:

```
<! ELEMENT poster EMPTY>
<! ATTLIST poster
  image ENTITY #REQUIRED>
```

记住 ENTITY 和 ENTITIES 属性是在 XML 文件中引用二进制(非解析)实体的唯一手段。这段代码声明了一个名为 poster 的实体,它具有一个准备好接收二进制实体的 image 属性。清单 4.3 包含了加入 poster 元素的新的电影集 DTD。

清单 4.3 支持实体的新的电影集 DTD

```
<! ELEMENT movies (move)+>
<! ELEMENT movie (title, writer+, producer+, director+, actor+, poster?,
  comments?)>
<! ATTLIST movie
  type (drama | comedy | adventure | sci-fi | mystery | horror | romance |
    documentary) "drama"
  rating (G | PG | PG-13 | R | X) "PG"
  review (1 | 2 | 3 | 4 | 5) "3"
  year CDATA #IMPLIED>
<! ELEMENT title (#PCDATA)>
<! ELEMENT writer (#PCDATA)>
<! ELEMENT producer (PCDATA)>
<! ELEMENT director (#PCDATA)>
<! ELEMENT actor (#PCDATA)>
<! ELEMENT poster EMPTY>
<! ATTLIST poster
  image ENTITY #REQUIRED>
<! ELEMENT comments (#PCDATA)>
```

在给文件内容中 poster 元素的 image 属性赋予二进制实体之前,你必须声明这个实体

类型的表示法,以及 image 实体本身。下面是这些声明:

```
<! NOTATION JPEG SYSTEM "IExplore.exe" >
<! ENTITY raposter SYSTEM "RAPoster.jpg" NDATA JPEG>
<! ENTITY mrposter SYSTEM "MRPoster.jpg" NDATA JPEG>
<! ENTITY tusposter SYSTEM "TUPoster.jpg" NDATA JPEG>
<! ENTITY taposter SYSTEM "TAPoster.jpg" NDATA JPEG>
```

注意 JPEG 实体类型指派给了 IExplore.exe 辅助应用程序。同样,每个图片实体在它声明的最后应用 JPEG 表示法。由于这些声明被指定给一个已有文件,它们应该放置在内部 DTD 子集中,而不是放在外部 DTD 子集中。

声明了表示法和实体后,你可以通过 poster 元素的属性引用实体了。下面是如何完成这项工作的例子:

```
<poster image="raposter"/>
```

到这里,实体都被声明了,表示法也已在适当的位置上供 XML 处理机使用了,因此任务就非常的简单。清单 4.4 包含了电影集文件的新的代码,它加入了 poster 元素和图片实体引用。

清单 4.4 完成的使用实体和表示法的已完成电影集 XML 文件

```
<? xml version = "1.0" standalone = "no"? >
<! DOCTYPE movies SYSTEM "Movies.dtd" [
<! NOTATION JPEG SYSTEM "IExplore.exe" >
<! ENTITY raposter SYSTEM "RAPoster.jpg" NDATA JPEG>
<! ENTITY mrposter SYSTEM "MRPoster.jpg" NDATA JPEG>
<! ENTITY tusposter SYSTEM "TUPoster.jpg" NDATA JPEG>
<! ENTITY taposter SYSTEM "TAPoster.jpg" NDATA JPEG>
<movies>
  <movie type="comedy" rating="PG-13" review="5" year="1987">
    <title>Raising Arizona</title>
    <writer>Ethan Coen</writer>
    <writer>Joel Coen</writer>
    <producer>Ethan Coen</producer>
    <director>Joel Coen</director>
    <actor>Nicolas Cage</actor>
    <actor>Holly Hunter</actor>
    <actor>John Goodman</actor>
    <poster image="raposter"/>
    <comments>A classic one-of-a-kind screwball love story.</comments>
  </movie>
  <movie type="comedy" rating="R" review="5" year="1988">
    <title>Midnight Run</title>
    <writer>George Gallo</writer>
    <producer>Martin Brest</producer>
    <director>Martin Brest</director>
    <actor>Robert De Niro</actor>
    <actor>Charles Grodin</actor>
    <poster image="mrposter"/>
```

```

    <comments>The quintessential road comedy. </comments>
  </movie>

  <movie type="mystery" rating="R" review="5" year="1995">
    <title>The Usual Suspects</title>
    <writer>Christopher McQuarrie</writer>
    <producer>Bryan Singer</producer>
    <producer>Michael McDonnell</producer>
    <director>Bryan Singer</director>
    <actor>Stephen Baldwin</actor>
    <actor>Gabriel Byrne</actor>
    <actor>Benicio Del Toro</actor>
    <actor>Chazz Palminteri</actor>
    <actor>Kevin Pollak</actor>
    <actor>Kevin Spacey</actor>
    <poster image="tusposter" />
    <comments>A crime mystery with incredibly intricate plot twists. </comments>
  </movie>

  <movie type="sci-fi" rating="PG-13" review="4" year="1989">
    <title>The Abyss</writer>
    <writer>James Cameron</writer>
    <producer>Gale Anne Hurd</producer>
    <director>James Cameron</director>
    <actor>Ed Harris</actor>
    <actor>Mary Elizabeth Mastrantonio</actor>
    <poster image="taposter" />
    <comments>A very engaging underwater odyssey. </comments>
  </movie>
</movies>

```

4.7 使用条件段

一个你有时会觉得非常有用的 DTD 特性就是条件段,它允许你在 DTD 中有条件的加入或排除代码。在 DTD 开发阶段这非常的有用。条件段使用 IGNORE 和 INCLUDE 关键字指定(导致段中的代码忽略或加入)。仅仅是通过 IGNORE 和 INCLUDE 关键字修改包含代码的段的指示,你就可以修改 DTD 代码的内含。下面是一个指示被 XML 处理函数忽略的代码段:

```

<! [IGNORE[
  <! ELEMENT title (#PCDATA)>
  <! ELEMENT track (#PCDATA)>
]]>

```

在这个例子中,title 和 track 元素没有声明,因为它们的声明包含在 IGNORE 段中。你可以通过将这个段改成 INCLUDE 段,在 DTD 中加入这些元素声明:

```

<! [INCLUDE[
  <! ELEMENT title (#PCDATA)>
  <! ELEMENT track (#PCDATA)>
]]>

```


这里有一些关于条件段使用的限制：

- 条件段只能用在文件的外部 DTD 子集中；它们不允许在内部 DTD 子集中出现。
- 条件段必须由完整的标记声明或声明组成。
- 任何关键字(IGNORE 或 INCLUDE)都不能出现在标记声明的内部。

条件段本身功能就足够的强大,但当它们与参数实体配合使用时才会变得更加有用。例如你可以使用参数实体作为条件段的基础,为你提供文件的调试标记。

检验一下下面的例子：

```
<! ENTITY % DEBUG "INCLUDE">
```

如果你使用 DEBUG 实体封装文件的调试部分,那么通过修改参数实体为下面的形式就很容易将这部分去除：

```
<! ENTITY % DEBUG "IGNORE">
```

使用这个例子中的 DEBUG 设置,任何包含在 DEBUG 段中的代码都会被 XML 处理机忽略。

注意:如果你熟悉 C 或 C++,你就会发现这个条件段和参数实体的使用类似于预处理宏 #ifdef。

4.8 总 结

虽然 DTD 的逻辑结构对于确定信息片之间的关系非常的重要,但物理结构确定了 XML 文件数据是如何存储的。一个 XML 文件可以分成名为实体的存储单元,这些实体是构成文件的物理构成块。与其他的文件结构部件类似,实体在 DTD 中声明。XML 使用两种基本类型的实体:解析和非解析。非解析实体不能被 XML 解析器解析,因此它们依赖于表示法进行处理。

第5章 XML Schema 基础

传统上,确认XML文件需要使用文件类型定义(DTD,Document Type Definition),这是继承自SGML的。虽然DTD在以确认为目的设置文件格式方面非常的有用,它们也有相当多的缺点。XML Schema是由Microsoft创建的技术,它实现了W3C关于改善DTD的摘录的一个子集。这个W3C摘录实际上描述了一个DTD的改进方法,它比DTD功能要强大的多。XML Schema本身以XML词表方式实现,这就使它比DTD更容易学习和使用。XML Schema的真正意义是它修改了大量的DTD的缺点,比如含糊的语法、匮乏的数据类型、有限的内容模型和对名字空间的不支持。

本章探讨了XML Schema技术并向你展示了如何使用它描述XML文件的结构。在进一步深入之前,有一点对你非常重要,就是要理解XML仍然是一项新的技术,它实际上只是完全的W3C摘录中有关文件Schema如何工作部分的实现。同样,由于XML Schema是Microsoft技术,本章以Internet Explorer网络浏览器为主。这是实际的需要,因为在本书编写期间,Internet Explorer 5.0是惟一支持各种格式的XML Schema的网络浏览器。

5.1 了解XML Schema

第2章“数据模型:文件类型定义(DTD)和Schema”讲解了XML Schema技术的基础和它与DTD的关系,因此我不希望花费太多的时间用在XML Schema的基本概念上。然而,还是很值得以W3C的观点了解XML Schema,因为现在有大量正在进行的工作已提供XML的先进Schema技术。我还希望捎带讲一下,Schema为什么会对XML的未来非常重要。

5.1.1 XML Schema 和 W3C

按照W3C的定义,Schema是“一个XML文件的结构绑定和信息集管理的规则的集合”。从这个描述中可以明显看出DTD符合XML Schema定义。然而,从中也可以显然的了解到,DTD有一些需要在XML获得广泛普及之前进行改善的明显的缺陷。这项工作起源于1998年1月的XML-Data摘录。它建议使用XML词表来描述文件结构。

有一点很重要就是,要理解W3C的规范可以达到的不同等级。最高等级是推荐标准状态。工作草案在它描述工作的过程中是与推荐标准有所不同的,虽然W3C基本将工作草案转为推荐标准。记录只是一个想法、评论或文件的日志性公共记录。它是W3C标准的最低等级,因此它并没有承诺在W3C的部分中进一步解释它的内容。然而,这并不是说记录不能最终变为草案或推荐标准。

在1998年7月,另一个记录提交给了W3C,它略述了一个名为文件内容描述(DCD,Document Content Description)的程序,它基于资源描述框架(RDF,Resource Description

Framework)词表。RDF 是设计用来提供处理元数据的基础的 XML 词表,它允许 Web 应用以机器可读格式共享和表述信息。DCD 本身是一个设计用来描述文件结构的 RDF 词表,它更像在 XML-Data 记录中介绍的词表。

注意:要获得有关 XML-Data 和 DCD 的更多信息,请访问以下网址:<http://www.w3.org/TR/1998/NOTE-XML-data-0105> 和 <http://www.w3.org/TR/NOTE-dcd>。

XML Schema 是一个由 Microsoft 创建的技术,它基于 XML-Data 和 DCD。Microsoft 研究了这两个 W3C 的草案并根据自己的实现进行了选择,它通过一个更类似于 XML-Data 的语法结合了两个 W3C 提交的特性。Microsoft 自己定义 XML Schema 为“一个符合 DCD 早期特征集的 XML-Data 提交的子集”。Microsoft 并没有严格的遵守任何一个提交,可能的理由是 XML-Data 和 DCD 都依旧处于开发的早期阶段。严格遵守一个不完整的标准并不必要,因此 Microsoft 决定开发自己的 Schema,以吸取 W3C 记录的精华。这就是可以在 Internet Explorer 5.0 中找到的 XML Schema。

如果你了解 Schema 与 W3C 相关的消息,你也许知道在 1999 年 5 月,W3C 也提出了已成为 XML Schema 的另一个 Schema 标准。这个最近的 Schema 标准是所有在它之前的 Schema 提议(XML-Data、DCD 等等)的思想的综合。在本书编写期间,W3C XML Schema 标准已经处于工作草案阶段,这意味着它仍然要经历重大的修改。然而,工作草案看起来非常的有前途,无疑将会给未来的 XML 文件结构带来意义深远的冲击,并得到确认。

在 W3C XML Schema 工作草案变成了成熟的推荐标准之前,你只能使用在 Internet Explorer 5.0 中的 Microsoft 的 XML Schema 实现。同时,希望 Microsoft 跟上 W3C 的动作,升级在 Internet Explorer 5.0 中支持的 XML Schema。本章将集中讲解 Internet Explorer 5.0 中的 XML Schema 实现。

注意:XML Schema 工作草案可在 <http://www.w3.org/1999/05/06-xml-schema-1> 中获得。

5.1.2 XML Schema 的好处

在进入 XML Schema 词表的细节之前,我希望给你提醒一下 XML Schema 在描述 XML 文件的标准 DTD 方法方面的主要好处。使用 XML Schema 描述的 Schema 在用来确定一类 XML 文件的 Schema 方面与 DTD 非常类似。如同 DTD 一样,XML Schema 的作用是描述元素和它的内容模型以让文件可以被验证。然而,XML Schema 比 DTD 走得更深入,它允许你给元素结合丰富的数据类型,最终也是作为属性体现。

在 DTD 中,元素内容局限在字符串和基本的数据类型上。XML Schema 支持大量丰富的数据类型,比如整数、浮点数、日期和时间。XML Schema 还包含对其他特性的支持,比如开放内容模型和综合名字空间。除了提供这些特性外,XML Schema 明显地与 DTD 有区别,因为它是依赖 XML 来标记 Schema 文件的。换句话说,XML 语法作为创建 XML Schema 文件的基础。这是对 DTD 的重大的改善,因为它并不需要你学习含糊的语言。作为替代,你可以学习如何使用 XML Schema 词表。

下面是 XML Schema 提供的主要优势的清单,以及与 DTD 的对比:

- XML Schema 基于 XML, 没有专门的语法。
- XML 可以像其他 XML 文件一样解析和处理。
- XML Schema 支持一系列的数据类型(int、float、Boolean、date 等等)。
- XML Schema 提供可扩充的数据模型, 它允许你扩充词表并在没有无效文件的情况下确定元素之间的关系。
- XML Schema 支持综合名字空间, 它允许你在 Schema 中将类型声明与文件个体节点想关联。
- XML Schema 支持属性组, 它允许你逻辑的联合属性。

显而易见, XML Schema 提供了很多的功能。本章剩下的部分将探讨 XML Schema 词表并向你展示如何从这些优势中获得利益。

5.2 深入 XML Schema 词表

如你所知, XML Schema 本身实际上是一个 XML 应用, 这意味着它是以 XML 词表实现的。使用 XML Schema 创建 Schema 与使用专门词表创建其他 XML 文件非常的类似。然而, 在这里你要在 XML Schema 文件中描述一个新的词表。XML Schema 基于 XML-Data 记录以及 DCD 提出的同类的特征集。同样的, 这一点上还确实没有用于 XML Schema 的标准 DTD。

由于 Microsoft 是当前惟一的直接支持 XML Schema 的公司, 它提供了一个可以用来验证 Schema 文件的 XML SchemaDTD。然而这个 DTD 有一些需要处理的局限性。DTD 只是不够强大和富于表现性来模型化 XML Schema 的全特征集。下面是 XML SchemaDTD 造成的主要局限:

- DTD 内容模型是有限的, 因此不能反映 XML Schema 的开放性, 它提供了一个可扩充内容模型, 允许你指定用在描述 Schema 中的新的元素和属性。
- DTD 不支持本章前面提到的十种基本类型以外的数据类型。因此, 它不能表达在 XML Schema 中允许的丰富数据类型。
- DTD 不能反映 XML Schema 中的名字空间的灵活性, 它允许你在 Schema 中的任何位置放置名字空间声明。
- DTD 不能使用在 maxOccurs 属性的枚举值(1, *), 因为 * 不是一个有效的 DTD 名字记号。

注意: 这似乎有一点奇怪, 就是 XML Schema 本身需要 DTD。实际上, XML Schema 对 DTD 并不是严格需要的, 除非你希望验证你的 Schema 文件。DTD 插入描述的惟一理由就是 XML Schema 是一个 XML 词表。然而, XML SchemaDTD 只用于 XML Schema 文件, 没有文件的结构是使用 XML Schema 描述的。因此, 如果你使用 XML Schema 创建了一个 Schema 文件, XML SchemaDTD 将只用来验证这个 Schema 文件, 没有文件使用这个 Schema 创建。

即使有这些局限性, 我们仍然值得学习一下 XML SchemaDTD 来获得对 XML Schema 词表是如何构造的这一问题的更好的理解。清单 5.1 包含了 XML SchemaDTD。

清单 5.1 XML Schema 文件类型定义 (DTD)

```

<! ENTITY % datatypes "(entity | entities | enumeration | id | idref | idrefs |
nmtoken | nmtokens | notation | string | bin.base64 | bin.hex | boolean |
char | date | dateTime | dateTime.tz | fixed.14.4 | float | int | number |
time | time.tz | i1 | i2 | i4 | r4 | r8 | ui1 | ui2 | ui4 | uri | uuid)">

<! ELEMENT datatype (description) * >
<! ATTLIST datatype
  dt:type %datatypes; #IMPLIED
  xmlns:dt CDATA #FIXED "urn:schemas-microsoft-com:datatypes">

<! ELEMENT description (#PCDATA)>

<! ELEMENT element (description) * >
<! ATTLIST element
  type IDREF #REQUIRED
  minOccurs CDATA #IMPLIED
  maxOccurs CDATA #IMPLIED>

<! ELEMENT attribute (description) * >
<! ATTLIST attribute
  type IDREF #REQUIRED
  default CDATA #IMPLIED
  required (yes | no) "no">

<! ELEMENT AttributeType (datatype | description) * >
<! ATTLIST AttributeType
  name ID #REQUIRED
  default CDATA #IMPLIED
  dt:type %datatypes; #IMPLIED
  dt:values CDATA #IMPLIED
  required (yes | no) #IMPLIED
  xmlns:dt CDATA #FIXED "urn:schemas-microsoft-com:datatypes">

<! ELEMENT ElementType (datatype | description | AttributeType | attribute |
element | group) * >
<! ATTLIST ElementType
  name ID #REQUIRED
  model (open | closed) #IMPLIED
  content (empty | textOnly | eltOnly | mixed) #IMPLIED
  order (one | seq | many) #IMPLIED
  dt:type %datatypes; #IMPLIED
  dt:values CDATA #IMPLIED
  required (yes | no) #IMPLIED
  xmlns:dt CDATA #FIXED "urn:schemas-microsoft-com:datatypes">

<! ELEMENT group (group | element | description) * >
<! ATTLIST group
  minOccurs CDATA #IMPLIED
  maxOccurs CDATA #IMPLIED
  order (one | seq | many) #IMPLIED>

<! ELEMENT Schema (AttributeType | ElementType | description) * >
<! ATTLIST Schema
  name CDATA #IMPLIED
  xmlns:dt CDATA #FIXED "urn:schemas-microsoft-com:datatypes">

```

这个 DTD 非常的简单易懂, 主要是因为 XML Schema 是一个大多情况下非常简单的

词表。然而,它仍然要提供多个描述XML文件细节的结构。所有XML Schema文件的根元素是Schema,它在DTD中被声明并潜在的包含三个子元素:AttributeType、ElementType和description。除了这些元素外,XML Schema词表还声明了一些用来描述文件Schema的其他元素。下面是标记XML Schema词表的元素:

- Schema——作为XML Schema文件的根元素。
- datatype——描述元素和属性的数据类型。
- ElementType——描述元素的类型。
- element —— 确定一个可以在其他元素类型中存在的元素。
- group —— 为了次序目的将元素组织进组中。
- Attribute Type——属性类型的描述。
- attribute——确定可以在元素类型中存在的属性。
- description —— 提供元素或属性的文件。

下面的几节将更详细的探讨这些元素并提供如何使用它们描述文件Schema的主要思想。

5.2.1 schema 元素

schema元素是XML Schema文件的根(文件)元素,它们是所有其他Schema内容的容器。schema元素加入了用来设置Schema文件的姓名和名字空间的两个属性:

- name —— Schema 的名字
- xmlns——Schema 的名字空间

name属性确定了Schema的名称。xmlns属性在确定Schema的名字空间方面非常的重要。这个属性必须设置为urn:schema s-microsoft-com:xml-data 以使用Microsoft的XML Schema实现。下面是XML Schema的框架结构,它适当的设置了name和xmlns属性:

```
<schema name="Myschema" xmlns="urn: schemas-microsoft-com:xml-data">
  <!--schema content goes here -->
</schema>
```

注意:名字空间在XML文件中用来保证与所给XML词表相关的元素和属性名称具有惟一性。名字空间具有URI的格式,URI通常是用来指定网络资源的常见URL。名字空间将在第7章“使用XML名字空间”介绍。

使用xmlns属性确定Schema的名字空间的技术似乎有一点奇怪。然而,它符合名字空间用在XML Schema文件中的方式。XML Schema文件本身在XML内容文件中使用名字空间进行引用;你将在本章中“从XML Schema创建文件”一节中学到更多的知识。

注意:xmlns属性实际被认为是一个处理过程,因为它确定了文件或元素的名字空间。设置这个属性的理由是使用名字空间时隐藏那些要在所有的XML Schema声明上追加的前缀。这也是为什么你将看到所有的XML Schema元素和属性输入就像是它们与名字空间没有关系。你设置xmlns属性,这意味着元素和属性是默认名字空间的一部分。

除了指定 Schema 的名字空间外,通常还需要指定 XML Schema 数据类型的名字空间。数据类型名字空间主要赋给 `xmlns:dt` 属性,并且设为 `urn:schemas-microsoft-com:datatypes`。你必须设置这个名字空间以使用任意的 XML Schema 数据类型,诸如 `date`、`time`、`int` 和 `float`。本章稍后部分,你将找到所有可用的不同的数据类型。下面是包含 XML Schema 数据类型名字空间的修改的 XML Schema 框架结构文件:

```
<schema name="Myschema" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <!--schema content goes here -->
</schema>
```

`schema` 元素可以包含子元素 `AttributeType`、`ElementType` 和 `description`。`AttributeType` 和 `ElementType` 元素定义属性类型和元素类型,它用来与其他 XML Schema 元素结合来描述 Schema 中元素和属性的结构。`description` 元素提供了输入 Schema 的文件的一种方法。

5.2.2 datatype 元素

`datatype` 元素用来指定元素或属性类型的数据类型。在 XML Schema 中,元素和属性通过使用 `ElementType` 和 `AttributeType` 元素的元素和属性描述。在 `ElementType` 和 `AttributeType` 元素中,`datatype` 元素包括一个属性 `dt:type`,它用来指定元素或属性的类型。下面的值允许作为 `dt:type` 属性的数据类型: `entity`、`entities`、`enumeration`、`id`、`idrefs`、`nmtoken`、`nmtokens`、`notation`、`string`、`bin.base64`、`bin.hex`、`boolean`、`char`、`date`、`dateTime`、`dateTime.tz`、`fixed.14.4`、`float`、`int`、`number`、`time`、`time.tz`、`i1`、`i2`、`i4`、`r4`、`r8`、`ui1`、`ui2`、`ui4` 和 `uuid`。你将在本章的稍后了解到使用这些类型表述的数据。

下面是一个使用 `datatype` 元素和 `dt:type` 属性定义数据类型的例子:

```
<datatype dt:type="float"/>
```

要谨记这个元素存在于 `ElementType` 或 `AttributeType` 元素中,因为 `datatype` 元素在元素和属性的内容之外没有意义。

注意: XML Schema 支持一个可选的方法,在不使用 `datatype` 元素的情况下指定元素和属性数据类型——使用 `ElementType` 和 `AttributeType` 元素的 `dt:type` 属性。

5.2.3 ElementType 元素

`ElementType` 元素用来定义确定文件的 Schema 的元素类型。与 `AttributeType` 元素配合,`ElementType` 元素构成了所有 XML Schema 文件的结构基础。`ElementType` 元素可以包含 `datatype`、`element`、`group`、`AttributeType`、`attribute` 和 `description` 子元素。`datatype` 子元素确定元素类型的数据类型。`element` 属性确定一个元素中的子元素的实例。你使用 `element` 属性来确定元素类型的内容模型。`group` 子元素用来将元素组织起来。

元素类型的属性使用 `AttributeType` 和 `attribute` 元素确定。`AttributeType` 元素定义了属性的类型,而 `attribute` 元素确定元素类型的实际属性。任何在 `ElementType` 元素中定义的属性类型都被认为是该元素的局部。这意味着在 Schema 文件的其他作用域内不会有具有相同名称的其他属性类型;局部作用域总是优先的。

注意:任何你希望在整个 Schema 文件中使用的属性类型必须放置在 Schema 的顶层,在 Schema 元素下。这与定义在局部作用域中的属性类型相反。同样的观念也用在元素中,它可以定义为全局化或局部化。

ElementType 元素包含一些属性定义元素类型的具体参数:

- name——元素的名称
- model——内容模型是开放型还是封闭型
- content——包含在元素中的内容类型
- order——包含在元素中的子元素和组的次序
- dt:type——元素的类型

下面是一些使用 ElementType 元素定义元素类型的例子:

```
<ElementType name="duration" content="textOnly" dt:type="time" />
<ElementType name="distance" content="textOnly" dt:type="float" />
<ElementType name="session" model="closed" content="eltOnly" order="seq" />
  <element type="duration" />
  <element type="distance" />
</ElementType>
```

要注意到 duration 和 distance 元素首先使用 ElementType 元素声明,之后使用 element 元素将它们确定在 session 元素的内容模型中。这些例子元素来自一个运动训练 Schema,你将在本章的后面构造这个 Schema。

name 和 model 属性

name 属性用来确定 ElementType 的名称,它是必要的属性。这个值在它被定义的作用域内必须是惟一的。Model 属性指定 Schema 文件遵守开放还是封闭内容模型。开放模型允许在元素类型中定义没有在 Schema 中声明的附件元素,它们确定了一个可扩展的 Schema。元素类型在你没有设置 model 属性的情况下采用开放模型,这也就是说 model 属性主要用来设置封闭内容模型。

注意:Element 元素的 model 属性用来指定 Schema 是开放的还是封闭的内容模型。开放内容模型是使用 XML Schema 而不是 DTD 的好处之一。DTD 只支持封闭内容模型。

content 属性

ElementType 元素的 content 属性用来确定包含在元素类型中的内容的类型。下面是这个属性可取的值:

- empty——元素类型不包含任何内容。
- textOnly——元素类型只包含文本(如果内容模型是开放的,元素类型也可以包含其他非特定的元素)。
- eltOnly——元素类型只包含指定的子元素。
- mixed——元素类型可以包含文本和指定子元素的混合体(如果内容模型是开放的,

元素类型也可以包含其他非特定的元素)。

这些内容类型对你来说应该非常的熟悉,因为它们实质上与 DTD 内容模型相对应。惟一的不同就是在内容模型是开放的时候使用非特定元素的可能性。第 6 章“XML Schema 构造技术”更为详细的解释了开放和封闭内容模型。

order 属性

order 属性用来确定包含在元素类型中的子元素组的次序和次数。下面是这个属性可取的值:

- one——只允许有一个元素集合
- seq——元素必须以指定次序出现
- many——元素可以以任何次序出现多次

order 属性非常的重要,因为它确定子元素的次序(或者没有次序),以及子元素的组可以出现多少次。这一点很重要,就是要理解只有结合 group 元素使用,你才会真正意识到 order 属性的能力。使用 group 元素,你可以组织元素并确定复杂的次序 Schema。过一会儿,你将详细的学习组织元素的知识。

dt:type 属性

dt:type 属性用来确定包含在元素类型中的内容的类型。在 dt:type 属性中允许的类型与那些在 datatype 元素中允许的类型相匹配。XML Schema 数据类型将在本章稍后部分详细的进行讲解。

5.2.4 element 元素

element 元素用来声明一个组和元素类型的元素的实例。element 元素就是用来表示内容模型是如何在 XML Schema 文件中声明的。element 元素包含三个属性,用来描述元素实例的附加信息:

- type——元素的类型
- minOccurs——元素必须出现的最小次数
- maxOccurs——元素必须出现的最大次数

type 属性用来确定元素的类型。赋给 type 属性的值必须是已经在 Schema 中声明的元素类型的名称。type 属性就是用来将元素实例与它们相应的元素类型相结合的。

minOccurs 和 maxOccurs 属性用来确定一个组或元素类型中元素出现的次数。这两个属性在 XML-Data 摘录中的默认值是 1,这意味着元素默认必须且只能出现一次。表 5.1 阐明了 minOccurs 和 maxOccurs 属性是如何影响元素出现的次数的。

警告:虽然 XML-Data 摘录允许这个属性具有大于 0 的任何值,但在 XML Schema 中,maxOccurs 只支持值 1 或 *。

表 5.1 minOccurs 和 maxOccurs 属性与元素和组可以出现的次数之间的关系

| minOccurs | maxOccurs | 元素/组可以出现的次数 |
|------------|------------|----------------|
| 0 | 1 | 0 或 1 |
| 1 | 1 | 1 |
| 0 | * | 任何次数 |
| 1 | * | 起码一次 |
| >0 | * | 起码 minOccurs 次 |
| >maxOccurs | >0 | 0 |
| 任何值 | <minOccurs | 0 |

注意:表 5.1 也适用于 group 元素,因为组也有起同样作用的 minOccurs 和 maxOccurs 属性。

下面是一个用来在元素类型中声明元素实例的 element 元素的例子:

```
<ElementType name="location" content="textOnly" />
<ElementType name="comments" content="textOnly" />
<ElementType name="session" model="closed" content="eltOnly" order="seq">
  <element type="location" minOccurs="1" maxOccurs="1">
  <element type="comments" minOccurs="0" maxOccurs="1">
</ElementType>
```

这段代码有些类似于你在本章前面看到的例子。这很有趣,因为它揭示了元素类型和元素实例之间的关系。Location 和 comments 元素类型首先使用 ElementType 元素被声明,之后它们作为另一个元素类型(session)的元素的实例被定义。Location 元素的 minOccurs 和 maxOccurs 属性都被设置为 1,这意味着它必须且只在 session 元素类型中出现一次。Comment 元素将 minOccurs 设置为 0,这意味着它可以出现一次或不出现。

5.2.5 group 元素

group 元素用来组织元素并确定复杂内容模型。复杂内容模型由多于一组的元素组成。group 元素包含三个属性来调整组:

- order——包含在组中的子元素的次序
- minOccurs——组必须出现的最少次数
- maxOccurs——组必须出现的最大次数

order 属性的工作与 ElementType 元素中的相应属性非常的类似。下面是这个属性可取的值。

- one——组中只允许有一个元素集合
- seq——元素在组中必须以指定次序出现
- many——元素可以在组中以任意次序出现多次

minOccurs 和 maxOccurs 属性在 group 元素中所扮演的角色与在 element 元素中的角色完全相同,它们用来绑定组可以出现的次数。我们可以重温一下表 5.1,以了解有关这些属

性是如何影响组的出现的。

5.2.6 AttributeType 元素

AttributeType 元素用来定义使用在元素中的属性类型。类似于 ElementType 元素, AttributeType 元素简单的定义了属性类型。为了实际的将属性声明为元素的一部分,你必须使用 attribute 元素,它用来引用 AttributeType 元素。属性类型可以定义在 Schema 文件的顶层或是在个别的元素类型中。它允许你创建全局属性或是给定作用域内的局部属性。全局属性是便利的,因为它们可以用在多个元素中。另一方面,局部属性可以用在所给作用域中以取代相同名称的其他属性。

AttributeType 元素包括如下的属性,以允许你完整的描述属性类型:

- name——属性类型的名称
- dt:type——属性类型的数据类型
- dt:values——枚举属性的可能值清单;只有在 dt:type 设置为 enumeration 时有效
- default——属性的默认值
- required——标志这个属性是否必须在元素中提供

name 属性指定属性类型的名称,它是必要属性。这个名称在给定作用域中的属性里必须是惟一的。dt:type 属性指定属性数据类型。虽然 XML-Data 摘录允许你使用大范围的数据类型,但 XML Schema 的 Internet Explorer 5.0 限制你只能使用标准 XML 属性类型,这些类型也就是基本数据类型。你将在本章中“XML Schema 数据类型”一节学到更多 XML Schema 支持的数据类型。

警告:虽然 Internet Explorer 5.0 支持 XML Schema 属性,它并不允许你使用所有范围的数据类型。更重要的是,Internet Explorer 5.0 只允许你使用标准 XML 属性类型,这些也就是在 XML Schema 中的基本类型。

dt:values 属性用来指定枚举类型的可能值清单。这个属性只有在 dt:type 设置为 enumeration 时才是可用的。枚举属性值的清单通过一个字符串指定,每个可能值之间用空格分开。下面是一个枚举属性定义的例子:

```
<AttributeType name="type" dt:type="enumeration"
dt:value="running cycling swimming" />
```

在这个例子中,可以赋给 type 属性的可能值是 running、cycling 和 swimming。任何不属于这三个值的值都被认为是错误的。

AttributeType 元素的 default 属性用来确定属性类型的默认值。当所给类型属性没有在元素中设置,属性将假定其为在 AttributeType 的 default 属性中定义的默认值。下面是一个确定属性默认值的例子:

```
<AttributeType name="type" dt:type="enumeration"
dt:values="running cycling swimming" default="running" />
```

这段例子代码确定枚举属性 type 的默认值为 running。

required 属性主要是用来指定属性类型在它被定义的元素中是否是必需的。required 属

性的可取值是 yes 和 no,它指出了对属性类型的需求。

5.2.7 attribute 元素

attribute 元素用来为元素类型声明属性的实例。attribute 元素包括三个属性来描述有关属性实例的附加信息:

- type——属性的类型
- default——属性的默认值
- required——表示属性是否必须在元素中提供

type 属性用来指定属性的类型。赋给 type 属性的值必须是在 Schema 中声明的属性类型的名称。type 属性用来结合属性实例和它们相应的属性类型。default 属性和 required 属性与在 AttributeType 元素中的等价物的作用相同,它们将取代那些可能在属性类型中设置的等价属性。

下面是一个用来在元素类型中声明属性实例的 attribute 元素的实例:

```
<AttributeType name="type" dt:type="enumeration"
dt:type="running cycling swimming" />
<AttributeType name="date" dt:type="date" />
<ElementType name="session" content="eltOnly" order="seq">
  <element type="duration" minOccurs="1" maxOccurs="1" />
  <element type="distance" minOccurs="1" maxOccurs="1" />
  <element type="location" minOccurs="1" maxOccurs="1" />
  <element type="comments" minOccurs="0" maxOccurs="1" />
  <attribute type="type" default="running" />
  <attribute type="date" />
</ElementType>
```

在这个例子中,type 和 date 属性首先使用 AttributeType 元素声明,然后使用 attribute 元素与元素类型相关。注意 type 属性的默认值在 attribute 元素中而不是在 AttributeType 元素中设定。这充分说明了 XMSchema 在声明和使用属性时给予你的灵活性。

注意:在元素中没有属性次序的绑定,但每个元素中给定名称的属性最多出现一次。

5.2.8 description 元素

最后要介绍的在 XML Schema 中使用的元素是 description 元素,它只是提供一种在 Schema 中放置文本描述的方法。description 元素是一个为了资料目的而设计的纯文本的元素。你可以用任何你所选的方法使用 description 元素以提供有关 XML Schema 机构的资料。下面是一个给元素类型添加资料的实例:

```
<ElementType name="trainlog" content="eltOnly">
  <description>
    This element type represents training log consisting of one or more
    Training sessions.
  </description>
<element type="session" minOccurs="1" maxOccurs="*" />
```


</ElementType>

5.3 XML Schema 数据类型

如你所知,XML DTD 提供有限数量的数据类型,它们都是非常基本的。为了实用的目的,XML 实际只支持字符串数据类型,这在你要创建结构文件 Schema 时限制很大。XML-Data 摘录定义了一定数量的可以用来指定常见数据类型的丰富数据类型,比如整数、浮点数、日期和时间等等。对于 Internet Explorer 5.0,XML Schema 在元素中支持所有这些数据类型,我们期望在将来它能够在属性中支持它们。

XML Schema 数据类型通过数据类型名字空间 `urn:schema-microsoft-com:datatypes` 应用。为了更简单的引用数据类型,你必须在你的 Schema 文件的文件级声明这个名字空间。数据类型名字空间主要赋给属性 `xmlns:dt`,这意味着你只要加上前缀就可以应用 XML Schema 数据类型。在本章前面你已经看到了如何声明这个名字空间,但是我要再展示给你以进行解释:

```
<schema name="Myschema" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
<!--schema content goes here -->
</schema>
```

声明 XML Schema 数据类型名字空间的惟一目的就是,这样你可以使用它支持的数据类型。下面是这些数据类型清单,它远远多于 XML1.0 中定义的有限数据类型:

- char——字符(单字符字符串)
- boolean——布尔量(0 或 1)
- int——整数(整数)
- float——实(浮点)数,有小数部分和可选的指数部分
- number——具有 14 位整数和 4 位小数的实数
- i1——单字节整数
- i2——双字节整数
- i4——四字节整数
- r4——四字节实数
- r8——八字节实数(与 float 相同)
- ui1——单字节无符号整数
- ui2——双字节无符号整数
- ui4——四字节无符号整数
- bin, hex——十六进制数(基数 16)
- bin, base64——六十四进制数
- date——日期(没有时间和时区)
- dateTime——有可选时间的日期(没有时区)
- dateTime, tz——有可选时间和时区的日期
- time——时间(没有日期和时区)

- time.tz——有时区的时间(没有日期)
- uri——统一资源标识符(URI)

你已经看到了使用 XML Schema 数据类型声明属性和元素的一些例子,因此我将不再细述。此外,我希望阐明规范 XML1.0 数据类型,这样你就可以理解所有 XML Schema 文件的选择。标注 XML 数据类型有时指的是基本类型,这有助于将它们与 XML Schema 类型区分。下面是可用在 XML Schema 中的基本数据类型。

- string——字符串类型
- enumeration——枚举类型(属性专有类型)
- notation——NOTATION 类型
- entity——ENTITY 类型
- entities——ENTITIES 类型
- id——ID 类型
- idref——IDREF 类型
- idrefs——IDREFS 类型
- nmtoken——NMTOKEN 类型
- nmtokens——NMTOKENS 类型

5.4 从 XML Schema 创建文件

现在你已经完全理解了 XML Schema 词表,已经准备好看一看文件是如何从 XML Schema 中创建的。然而,在看一个详细的文件之前,你需要一个可以完全工作的 Schema 文件。清单 5.2 包含了完整的运动训练 Schema,其中的部分片段你已经在本章中看到了。

清单 5.2 Train 运动训练文件 Schema

```
<? xml version="1.0"? >
<Schema name="TrainSchema"
  xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:
  datatypes">
  <ElementType name="duration" content="textOnly" dt:type="time"/>
  <ElementType name="distance" content="textOnly" dt:type="float"/>
  <ElementType name="location" content="textOnly"/>
  <ElementType name="comments" content="textOnly"/>
  <AttributeType name="type" dt:type="enumeration"
    dt:values="running cycling swimming"/>
  <AttributeType name="date" dt:type="date"/>
  <ElementType name="session" content="eltOnly" order="seq">
    <description>
      This element type represents a single training session.
    </description>
    <element type="duration" minOccurs="1" maxOccurs="1"/>
```

```

    <element type="distance" minOccurs="1" maxOccurs="1"/>
    <element type="location" minOccurs="1" maxOccurs="1"/>
    <element type="comments" minOccurs="0" maxOccurs="1"/>
    <attribute type="date"/>
  </ElementType>

  <ElementType name="trainlog" content="eltOnly">
    <description>
      This element type represents training log consisting of one or more
      training sessions.
    </description>
    <element type="session" minOccurs="1" maxOccurs="*" />
  </ElementType>
</Schema>

```

这个 Schema 文件描述了一个名为 TrainSchema 的 XML 词表,它被设计用来输入运动训练信息。例如,如果你正在为夏威夷铁人三项运动锻炼,你可以使用这个词表创建一个训练记录。实际上,词表中声明的根元素名为 trainlog,它由多个 session 元素组成,它们用来表现各自的训练内容。Schema 其余的部分非常简单易懂,包括日期、训练类型、持续时间、距离、位置和与训练相关的意见这样的信息。

假设你要增加一些严格的运动训练,你将需要一种将 Train SchemaSchema 与实际 XML 文件相关联的途径。在 Internet Explorer 5.0 中,你可以从名字空间声明中直接引用 Schema。在你使用这种方法来将文件与 Schema 联系的时候,Internet Explorer 5.0 中的 XML 处理函数会自动用 Schema 验证文件。

要使用名字空间声明来声明 Schema,你必须设置名字空间为包含 Schema 文件 URI 的字符串,该 URI 的前缀为字符串 x-schema。下面是一个如何在运动训练中实现的例子(假设文件与 Schema 文件放置在同一个目录下):

```

<trainlog xmlns="x-schema:Trainschema.xml">
</trainlog>

```

当 Internet Explorer 5.0 中的 XML 处理函数看到了这个名字空间声明,他知道调用 TrainSchema.xml 作为 Schema 并使用它验证运动训练文件。清单 5.3 包含一个有着多个训练项目的运动训练文件。

清单 5.3 基于 Trainschema 运动训练 Schema 的运动训练记录

```

<? xml version="1.0" standalone="no"? >
<trainlog xmlns="x-schema:TrainSchema.xml">
  <session type="running" date="12/15/99">
    <duration>01:00:00</duration>
    <distance>5.5</distance>
    <location>Warner Park</location>
    <comments>Late afternonn run, felt strong.</comments>
  </session>

  <session type="cycling" date="12/17/99">
    <duration>02:10:0</duration>

```

```
<distance>37.0</distance>
<location>Natchez Trace Parkway</location>
<comments>Hilly ride, felt weak toward the end.</comments>
</session>

<session type="running" date="12/18/99">
  <duration>01:20:00</duration>
  <distance>8.5</distance>
  <location>Warner Park</location>
  <comments>Mid-morning run, tough pace.</comments>
</session>
</trainlog>
```

如你所见,这个文件严格的遵守 TrainschemaSchema。trainlog 和 session 元素中的元素遵循 Schema 列出的内容模型,所有的元素和属性都是正确的数据类型。从设计良好的 Schema 中创建内容通常是非常简单的,这就是在开始创建文件之前花大量时间来考虑 Schema 设计的主要动机。

5.5 总 结

XML Schema 是实现 W3C 的 XML-Data 摘要子集的一项 Microsoft 技术,它与文件内容描述(DCD)W3C 摘录的功能非常的类似。XML Schema 在描述 XML 文件结构方面比主要继承自 SGML 的传统的 DTD 方法提供了更大的益处。XML Schema 最有趣的地方是:它本身就是作为 XML 词表实现的。

第 6 章 XML Schema 构造技术

XML Schema 相对 DTD 而言,提供了一种建立 XML 文件结构模型的高级方法。然而 DTD 已经出现很久了,并且仍然作为一种得到接受了描述文件结构而验证的方法来使用。随着 XML Schema 逐渐成为标准,XML 开发者们会愿意将他们的 DTD 转换为 XML Schema 文件以利用它的许多优点。本章将讨论与将 DTD 转换为 Schema 相关的问题,包括如何手工将 DTD 转换为 Schema。你还会看到一个自动化这一过程的工具。

本章还涉及了其他一些上一章没有涉及到的问题。比如,Microsoft 有一个叫做 XML Validator 的工具可以用来根据 Schema 验证文件。内容模型在本章中也有所涉及,并且特别强调了它们是如何应用于 XML Schema。本章还介绍了一些在 XML Schema 工作草案中描述的特征,Microsoft 的 XML Schema 版本还不支持这些特征。

6.1 将 DTD 转换为 Schema

因为 XML Schema 提供了未来在 XML 文件中描述和验证的方法,因此你可能会考虑在某些情况下将现有的 DTD 转化为 Schema。幸运的是,XML Schema 提供了 DTD 功能的一个超集,因此当你从 DTD 向 Schema 迁移的时候不会损失任何东西。事实上,通过支持 XML Schema 的复合数据类型等特征的支持,从 DTD 到 XML Schema 的移植为你提供了一个提高文件描述能力的机会。

注意: 尽管技术上 DTD 和 XML Schema 都可以被认为是文件 Schema,但这里以及本章的其他部分中将使用术语 Schema 来引用 XML Schema 文件。

为了成功地从 DTD 中创建 Schema,你必须访问 DTD 的结构,这包括为构成 DTD 的不同元素和属性的值,你将不得不在 Schema 中创建元素和属性类型,这意味着当你结构化 Schema 文件的时候你将不得不做更多的规划。这些额外的规划从长远上讲是值得的,因为 XML Schema 文件通常以一种比 DTD 更结构化的方式来描述文件类。

当你第一次从 DTD 中构建 Schema 的时候,你应当从 DTD 中的第一个文件元素开始并且从此向下展开。然而,Schema 要求元素和属性类型在使用之前定义,因此必须从后向前构建 Schema。比如,如果 DTD 中的第一个元素是 contacts,而且它被声明为含有 person 类型的子元素,你需要在 Schema 的最后定义 contacts 元素类型,而 person 必须在其之前定义。person 元素类型中所引用的任何实体或属性类型都必须在此之前定义。这种从下向上的构建 Schema 的方法可以帮助我们满足 XML Schema 对于元素和属性类型必须在引用之前定义的要求。如果你考虑到从下向上地阅读 Schema,那么这种方法就更可以理解了。

注意在 DTD 中声明的内容模型是非常重要的,因为它们表示了 DTD 和 Schema 的最显著的不同。如果回忆一下,在 Schema 中一个元素的内容模型是通过引用其他元素类型的内容中的元素来构成的。内容模型中元素的顺序和出现是由 ElementType 和 element 元素

的属性决定的。比如,为了指定一个元素必须至少出现一次,你可以设置 minOccurs 属性为 1 而设置 maxOccurs 属性为 *。DTD 中相应的定义是在内容模型中的元素名称后面加上一个加号(+)。表 6.1 列出了主要的 DTD 内容模型标记以及如何将它们转换为 XML Schema 结构。

表 6.1 DTD 内容模型标记和 XML Schema 结构之间的关系

| DTD 记号 | 等价 XML Schema 结构 |
|--------|--|
| 括号 | 列出在 ElementType 或 group 元素中的元素 |
| 逗号 | 设置在 ElementType 或 group 元素中属性的顺序为 seq |
| 管道符() | 设置在 ElementType 或 group 元素中属性的顺序为 many |
| 无符号 | MinOccurs="1", maxOccurs="1" |
| 问号 | MinOccurs="0", maxOccurs="1" |
| 星号 | MinOccurs="0", maxOccurs="*" |
| 加号 | MinOccurs="1", maxOccurs="*" |

正如你所了解的,属性的处理在 DTD 中和在 XML Schema 中也有所不同。最显著的不同在于枚举的属性,在 Schema 中这种属性的值是通过 dt:value 属性来指定的。在 DTD 中,枚举属性的值是通过括号中用管道分割开来的一系列值来指定的。另外,缺省的属性值在 Schema 中用 default 属性来指定,而不是像在 DTD 中那样在属性声明的后面列出缺省值。

尽管 DTD 和 Schema 之间还有许多不同,我已经列出的已经足够让我们开始一个转换并且做一些实际的例子。此外,阅读将 DTD 转化为 Schema 的实例能比列出需要改变哪些地方更好地说明转化过程。清单 6.1 包含了第 3 章“DTD 基础”中的电影集 DTD。如果你略过了第 3 章,我现在说明一下这个 DTD 被设计来创建和验证含有电影集的 XML 文件。

清单 6.1 具有完整的元素和属性的电影集 DTD

```

<! ELEMENT movies (movie)+>
<! ELEMENT movie (title, writer+, producer+, director+, actor *, comments?)+>
<! ATTLIST movie
type (drama | comedy | adventure | sci-fi | mystery | horror | romance |
documentary) "drama"
rating (G | PG | PG-13 | R | X) "PG"
review(1 | 2 | 3 | 4 | 5) "3"
year CDATA #IMPLIED>
<! ELEMENT title (#PCDATA)>
<! ELEMENT writer (#PCDATA)>
<! ELEMENT producer (#PCDATA)>
<! ELEMENT director (#PCDATA)>
<! ELEMENT actor (#PCDATA)>
<! ELEMENT comments (#PCDATA)>

```

这个 DTD 不太复杂,这使得将它转化为 XML Schema 文件相对比较简明。movie 元素实际上是基本内容模型中惟一的元素,因此大多数的 Schema 工作都围绕这个元素展开。然而,movies 元素是这个 DTD 中最高层的元素,因此它将作为转换这个 DTD 的起始点。请记住我们从最高层的元素开始但是我们将它放在 Schema 的最后。

在 Schema 中定义 movies 元素之前,首先命名这个 Schema 是非常重要的。在这种情况下 movieSchema 足以描述这个 Schema。下面是电影集 Schema 的框架:

```
<? xml version="1.0"? >
<Schema name="MovieSchema"
xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
<!--the rest of the schema goes here -->
</Schema>
```

这个框架 Schema 主要设置了 Schema 的名字并且定义了适当的名字空间。这个 Schema 现在就可以加入元素了,就从 movies 元素开始:

```
<ElementType name="movies" content="eltOnly">
  <element type="movie" minOccurs="1" maxOccurs="*" />
</ElementType>
```

这段代码定义了 movies 元素类型,它仅仅是为了含有子元素。在 movies 元素中,movie 类型的元素可以出现一次或若干次。请注意这个内容模型强制使用 minOccurs 和 maxOccurs 属性。

因为 movie 元素类型必须在 movies 元素类型引用它之前定义,所以我们必须先定义 movie 元素类型。于是,在前面的 movies 元素类型之前,我们定义了下面的 movie 元素类型:

```
<ElementType name="movie" content="eltOnly" order="seq">
  <element type="title" minOccurs="1" maxOccurs="1" />
  <element type="writer" minOccurs="1" maxOccurs="*" />
  <element type="producer" minOccurs="1" maxOccurs="*" />
  <element type="director" minOccurs="1" maxOccurs="*" />
  <element type="actor" minOccurs="0" maxOccurs="*" />
  <element type="comments" minOccurs="0" maxOccurs="1" />
  <attribute type="type" />
  <attribute type="rating" />
  <attribute type="review" />
  <attribute type="year" />
</ElementType>
```

正如你所见到的,movie 元素类型表示了电影集 Schema 中的一大堆结构。然而,在 movie 元素中定义的所有元素和属性都必须被定义为元素和属性类型。下面是 rating 属性的定义,它包括一个枚举值清单,以及一个缺省值:

```
<AttributeType name="rating" dt:type="enumeration"
dt:value="G PG PG-13 R X" default="PG" />
```

现在,我们就可以完成电影集 Schema 从 DTD 到 Schema 的转换了。如清单 6.2 所示。

清单 6.2 电影集 Schema

```

<? xml version = "1.0"? >
<Schema name="MovieSchema"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name = "title" content="textOnly"/>
  <ElementType name = "writer" content = "textOnly"/>
  <ElementType name = "producer" content = "textOnly"/>
  <ElementType name = "director" content = "textOnly"/>
  <ElementType name = "actor" content = "textOnly"/>
  <ElementType name = "comments" content = "textOnly"/>
  <ElementType name = "type" dt:type = "enumeration"
    dt:value =
      "drama comedy adventure sci-fi mystery horror romance documentary"
    default="drama"/>
  <AttributeType name="rating" dt:type="enumeration"
    dt:values="G PG PG-13 R X" default="PG"/>
  <AttributeType name="review" dt:type="enumeration"
    dt:values="1 2 3 4 5" default="3"/>
  <AttributeType name="year" dt:type="int"/>
</ElementType name="movie" content="eltOnly" order="seq">
  <element type="title" minOccurs="1" maxOccurs="1"/>
  <element type="writer" minOccurs="1" maxOccurs="*" />
  <element type="producer" minOccurs="1" maxOccurs="*" />
  <element type="director" minOccurs="1" maxOccurs="*" />
  <element type="actor" minOccurs="0" maxOccurs="*" />
  <element type="comments" minOccurs="0" maxOccurs="1"/>
  <attribute type="type"/>
  <attribute type="rating"/>
  <attribute type="review"/>
  <attribute type="year"/>
</ElementType>
  </ElementType name="movies" content="eltOnly">
    <element type="movie" minOccurs="1" maxOccurs="*" />
  </ElementType>
</Schema>

```

尽管电影集 Schema 和相应的 DTD 不完全精确,但是它更为强大。这是因为 Schema 中的元素和属性是用 XML Schema 数据类型定义的,这就为文件结构提供了更为准确的描述。附加的 XML Schema 特征还可以通过使用名字空间将 Schema 扩展为开放内容模型。

有了合适的 Schema 来表示电影集,我们就该考虑如何用它来验证电影集了。下一节中我们将看到如何使用 Internet Explorer 5.0 中的 XML 处理函数来根据 Schema 验证文件。

6.2 根据 Schema 验证文件

尽管 XML Schema 提供了描述 XML 文件类更好的方法,但是如果没有支持用 Schema 来验证 XML 文件的 XML 处理函数,它也没什么作用。Internet Explorer 5.0 中的 XML 处理函数完全支持 Microsoft 的 XML Schema 版本,而且可以被用来根据 Schema 验证文件。然而,你必须首先在名字空间声明中指定 Schema 来将 Schema 和文件联系起来。

回忆一下上一章的内容,Internet Explorer 5.0 允许你在名字空间声明中直接引用一个 Schema。当 Internet Explorer 5.0 的 XML 处理函数遇到这个声明的时候,它会自动根据 Schema 验证文件。为了用名字空间声明来声明 Schema,我们必须将名字空间设置为一个含有 Schema 文件 URI 并且以 x-schema 开头的字符串。清单 6.3 包含了一个电影集 XML 文件,它指定使用电影集 Schema 来验证它自己。

清单 6.3 用 Schema 来验证的电影集文件

```
<? xml version="1.0" standalone="no"? >
<movies xmlns="x-schema:MovieSchema.xml">
  <movie type="comedy" rating="PG-13" review="5" year="1987">      <title>
Raising Arizona</title>
    <writer>Ethan Coen</writer>
    <writer>Joel Coen</writer>
    <producer>Ethan Coen</producer>
    <director>Joel Coen</director>
    <actor>Nicolas Cage</actor>
    <actor>Holly Hunter</actor>
    <actor>John Goodman</actor>
    <comments>A classic one-of-a-kind screwball love story. </comments>
  </movie>
  <movie type="comedy" rating="R" review="5" year="1988">
    <title>Midnight Run</title>
    <writer>George Gallo</writer>
    <producer>Martin Brest</producer>
    <director>Martin Brest</director>
    <actor>Robert De Niro</actor>
    <actor>Charles Gridin</actor>
    <comments>The quintessential road comedy. </comments>
  </movie>
  <movie type="mystery" rating="R" review="5" year="1995">
    <title>The Usual Suspects</title>
    <writer>Christopher McQuarrie</writer>
    <producer>Bryan Singer</producer>
    <producer>Michael McDonnell</producer>
    <director>Bryan Singer</director>
    <actor>Stephen Baldwin</actor>
    <actor>Gabriel Byrne</actor>
    <actor>Benicio Del Toro</actor>
    <actor>Chazz Palminteri</actor>
    <actor>keVin Pollak</actor>
```

```
<actor>KeVin Spacey</actor>
<comments>A crime mystery with incredibly intricate plot twists.</comments>
</movie>

<movie type="sci-fi" rating="PG-13" review="4" year="1989">
  <title>The Abyss</title>
  <writer>James Cameron</writer>
  <producer>Gale Anne Hurd</producer>
  <director>James Cameron</director>
  <actor>Ed Harris</actor>
  <actor>Mary Elizabeth Mastrantonio</actor>
  <comments>A very engaging underwater odyssey.</comments>
</movie>
</movies>
```

通过在XML文件中正确地指定Schema,你可以用Internet Explorer的XML处理函数来验证这个文件。你可能会认为简单地在Internet Explorer中打开一个XML文件会导致处理函数的加载并且验证该文件。然而,当你打开XML文件的时候,这个处理函数仅仅检查确认它是否是结构良好的,而不进行验证。在Internet Explorer 5.0中要根据Schema来验证XML文件,我们必须用脚本语言来和XML文件交互并调用XML处理函数。

与其开发脚本代码来执行这样一个公共的工作,为什么不使用Microsoft提供的验证文件的工具呢?XML Validator是一个通过含有调用XML处理函数并根据Schema验证XML文件的HTML文件实现的一个工具。XML Validator可以从下面的网址使用或下载:

<http://msdn.microsoft.com/downloads/samples/internet/xml/xml-validator>。

要使用XML Validator,只要简单地在文本框中输入要验证的XML文件的路径名然后点击VALIDATE按钮就可以了。图6.1显示了在成功地根据电影集Schema验证了电影集XML文件后的XML Validator。

注意:除了用脚本来验证文件之外,Microsoft还有一个叫做XML Validation Tool的命令行验证工具。这个工具在命令行中运行,它根据DTD或者Schema检查并验证文件。这个XML Validation Tool可以从<http://msdn.microsoft.com/downloads/tools/xmlint/xmlint.asp>下载。

6.3 用XML Authority生成Schema

在本章的前面,你可能就已经考虑到DTD应当可以自动转化为XML Schema文件。我们要感谢一个叫做XML Authority的Schema开发工具,它可以自动完成DTD到XML Schema文件的转化。XML Authority是一个Extensibility的Schema开发工具,它提供了一般Schema开发方法,特别是在XML Schema的开发方面。

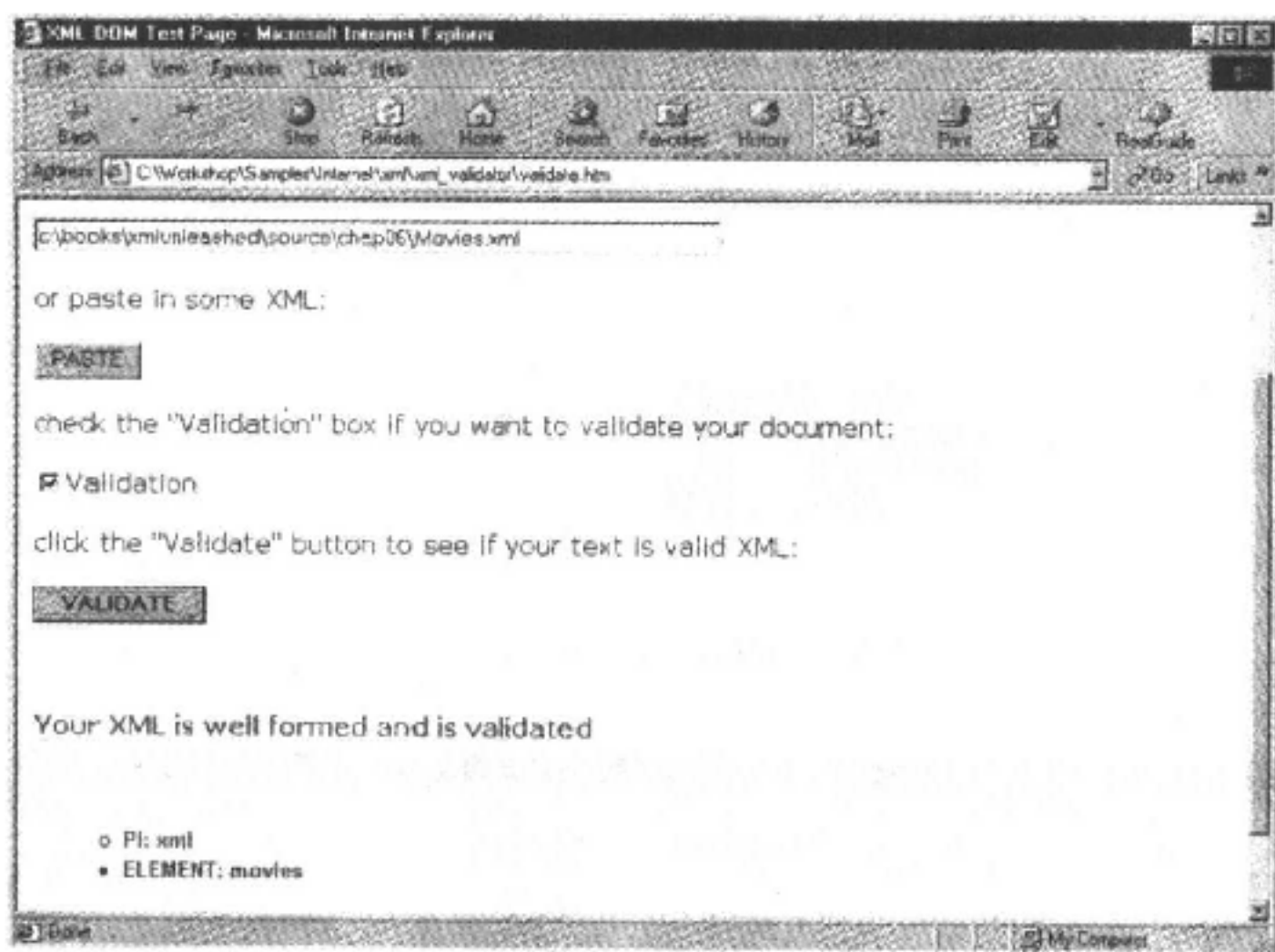


图 6.1 在成功地根据电影集 Schema 验证了电影集 XML 文件后的 XML Validator 工具

通过建立一个用户接口来以一种非常通用的方式创建 Schema, XML Authority 简化了 Schema 的概念。XML Authority 内部使用的是 DTD 格式,但是它具有将 Schema 输出为 XML Schema 文件的功能。在不同的 Schema 格式之间无缝地迁移这一功能非常强大而且在 Schema 标准正在不断变化的时候特别有用。因为它支持多种不同的 XML Schema 标准, XML Authority 已经被当作一种标准件,这是一种看待它的非常有趣的方法。当你在 XML Authority 中创建 Schema 的时候,你拥有使用传统的 DTD 或者以包括 XML Schema 在内的多种格式输出 Schema 的灵活性。

XML Authority 提供了一个图形用户界面来可视地构建 Schema。虽然我宁愿手工编写 Schema 代码,但是我仍然发现 XML Authority 在浏览以及检查 Schema 错误的时候非常有用。而且再次重申,将 Schema 从一种格式转化为另外一种格式的功能是其非常强大的特征。XML Authority 1.0 有 Windows 98/NT 和 UNIX 的版本,对于单用户的许可是 99.95 美元。请访问 Extensibility 的网站 <http://www.extensibility.com> 来购买 XML Authority 或下载免费的试用版。

本节的其他部分将简要介绍 XML Authority 并演示如何用它来处理 DTD 以及将 DTD 转化为 XML Schema 文件。图 6.2 显示了 XML Authority 的开始画面。

开始使用 XML Authority 的最好的方法也许就是打开一个你已经很熟悉的 DTD。这样,你可以看到不同的 DTD 结构是如何在 XML Authority 图形用户界面显示出来的。图 6.3 显示了在 XML Authority 中打开的电影集 DTD。

这个图显示了在电影集 DTD 中声明的元素类型。为了查看和访问这些属性类型,你必须点击工具栏上的 Attribute Types 按钮,这将会打开一个独立的窗口并列出这些属性类型(如图 6.4)。

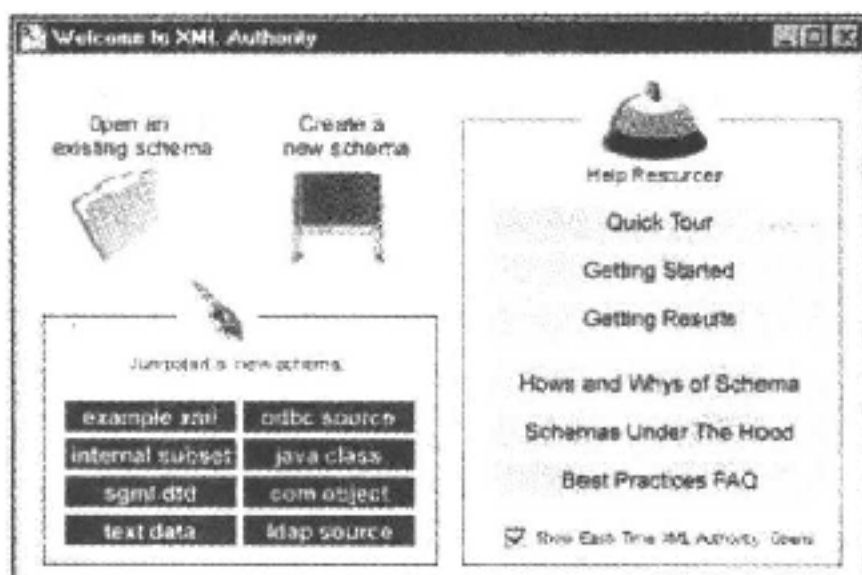


图 6.2 XML Authority 的开始画面

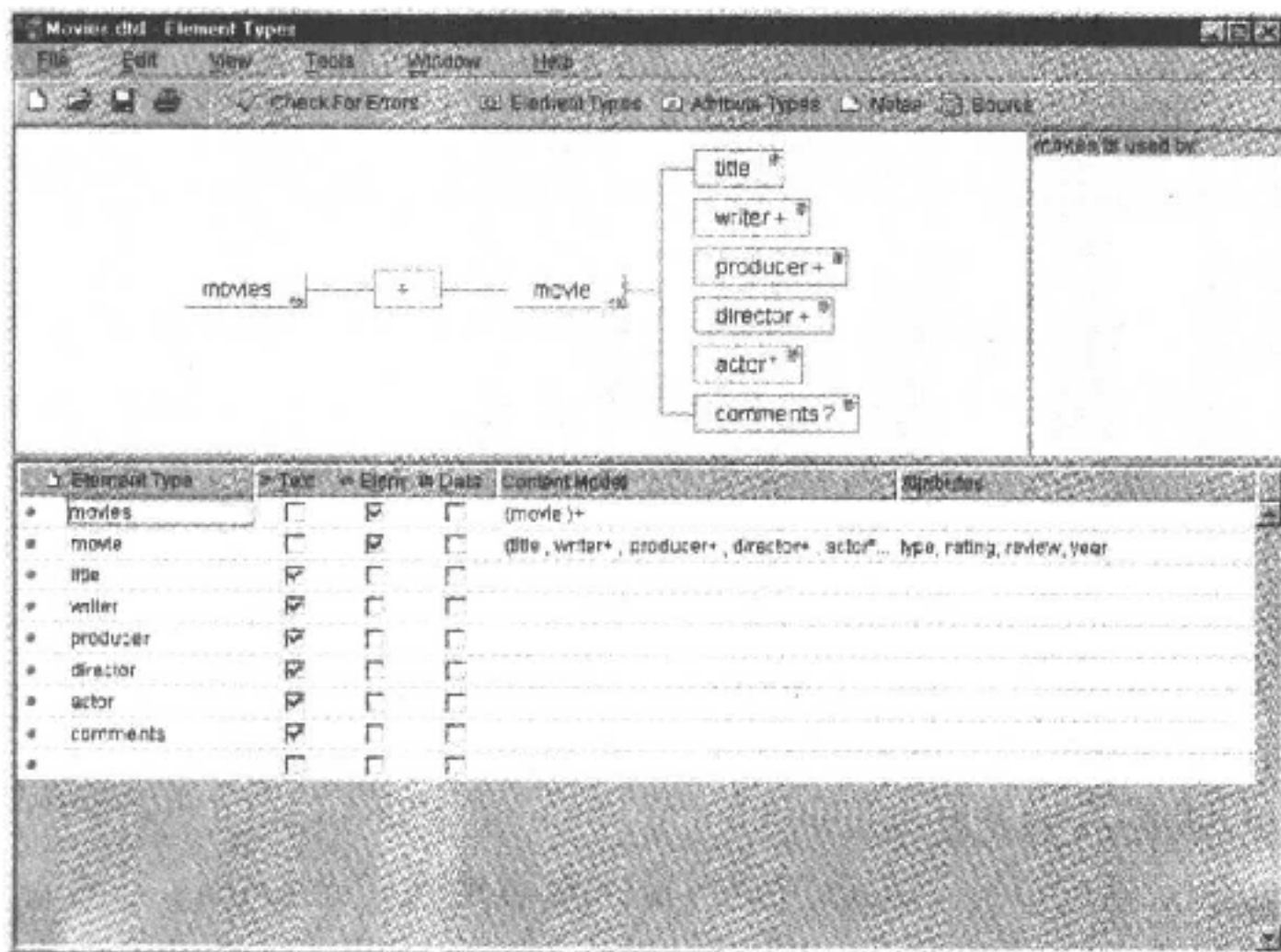


图 6.3 在 XML Authority 中打开的电影集 DTD

XML Authority 包括一个检查 DTD 错误的功能。只要点击工具条上的 Check For Error 按钮就可以了。图 6.5 显示了在按下这个按钮之后，显示出检查电影集 DTD 的错误检查的结构对话框。

正如你所看到的，电影集 DTD 的检查结果是好的，因此就可以将它转化为 XML Schema 文件。XML Authority 中所有的 Schema 转换都是通过 File 菜单中的 Export 开始的。XML Authority 会生成一个和 DTD 同名的文件，但是其扩展名将是 xdr。清单 6.4 显示了 XML Authority 从 DTD 生成的电影集 XML Schema。

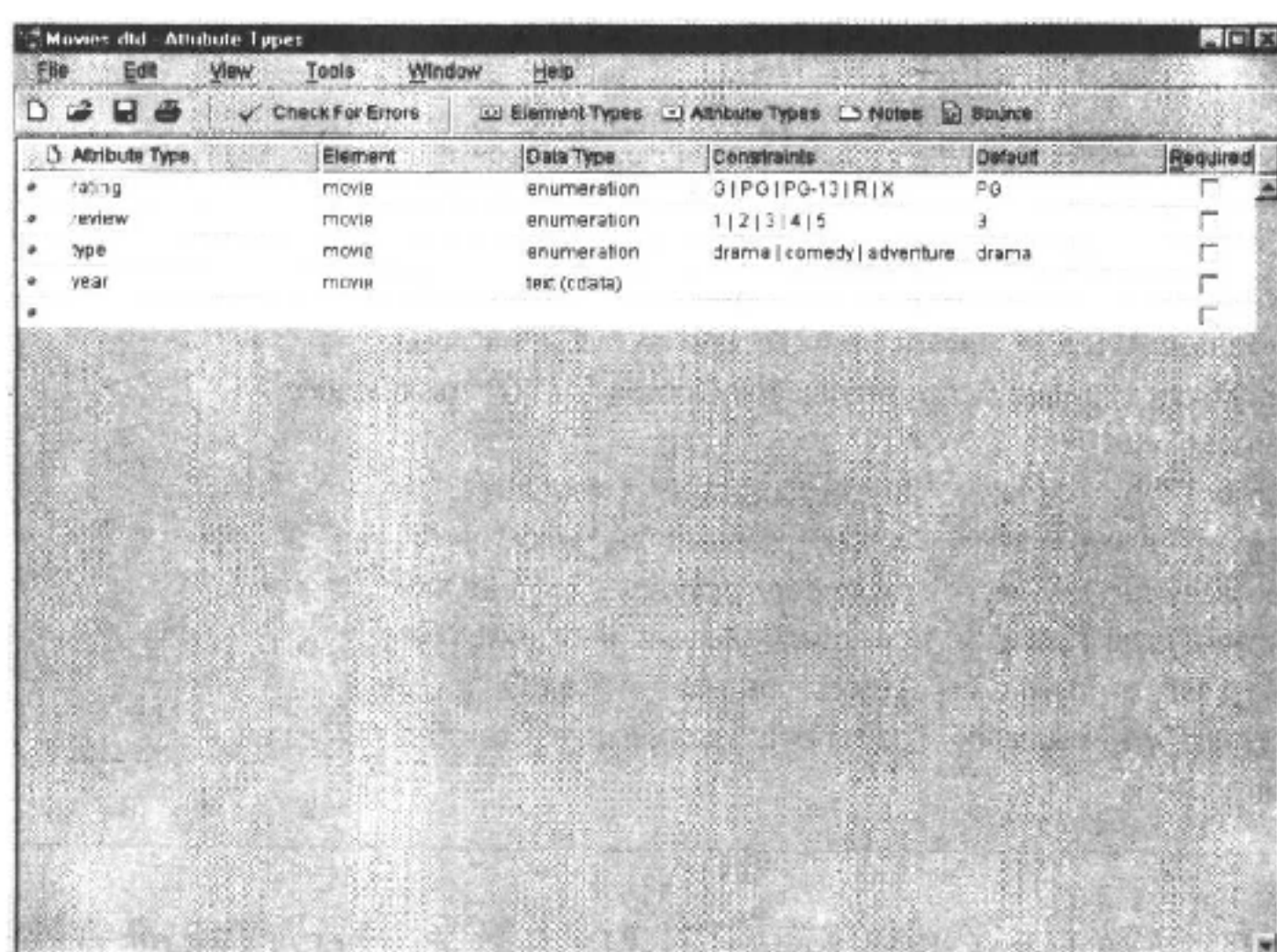


图 6.4 在 XML Authority 显示的电影集 DTD 的属性类型

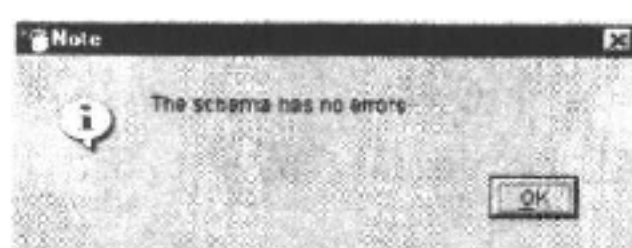


图 6.5 在 XML Authority 检查了电影集 DTD 的错误之后显示的对话框

清单 6.4 XML Authority 生成的电影集 Schema

```

<? xml version = "1.0" ? >
<! -- Generated by XML Authority. Conforms to XML Data subset for IE 5 -->
<Schema name = "Movies.dtd"
  xmlns = "urn:schemas-microsoft-com:xml-data"
  xmlns:dt = "urn:schemas-microsoft-com:datatypes">
  <ElementType name = "movies" content = "eltOnly">
    <group order = "seq" minOccurs = "1" maxOccurs = " * ">
      <element type = "movie" />
    </group>
  </ElementType>
  <ElementType name = "movie" content = "eltOnly" order = "seq">
    <AttributeType name = "type" dt:type = "enumeration" dt:values =
      "drama comedy adventure sci-fi mystery horror romance documentary"
      default = "drama" />
    <AttributeType name = "review" dt:type = "enumeration"
      dt:values = "1 2 3 4 5" default = "3" />
    <AttributeType name = "year" />
    <AttributeType name = "rating" dt:type = "enumeration"
      dt:values = "G PG PG-13 R X" default = "PG" />
    <attribute type = "type" />
  </ElementType>

```

```

    <attribute type = "review" />
    <attribute type = "year" />
    <attribute type = "rating" />
    <element type = "title" />
    <element type = "writer" minOccurs = "1" maxOccurs = "*" />
    <element type = "producer" minOccurs = "1" maxOccurs = "*" />
    <element type = "director" minOccurs = "1" maxOccurs = "*" />
    <element type = "actor" minOccurs = "0" maxOccurs = "*" />
    <element type = "comments" minOccurs = "0" maxOccurs = "1" />
  <ElementType>
    <ElementType name = "title" content = "textOnly" />
    <ElementType name = "writer" content = "textOnly" />
    <ElementType name = "producer" content = "textOnly" />
    <ElementType name = "director" content = "textOnly" />
    <ElementType name = "actor" content = "textOnly" />
    <ElementType name = "comments" content = "textOnly" />
  </Schema>

```

这些代码和前面清单 6.2 中我们手工创建的电影集 Schema 代码非常接近。这个测试说明 XML Authority 可以智能地将 DTD 转化为一般的 Schema。

XML Authority 的另外一个有趣的输出特征是它支持输出基于 DTD 的 XML 文件示例。基本上讲,XML Authority 会创建一个遵守 DTD 中定义的文件结构的框架 XML 文件。这为我们提供了一个创建有效文件的捷径。为了创建基于 DTD 的 XML 文件示例,在 Export 菜单中选择 Example XML Document。清单 6.5 包含了 XML Authority 生成的电影集文件的示例。

清单 6.5 XML Authority 生成的电影集文件的示例

```

<? xml version = "1.0"? >
<! DOCTYPE movies SYSTEM "C:\Books\XMLUnleashed\Source\Chap06\Movies.dtd" >
<!--Generated by XML Authority. -->
<movies>
  <!-- (movie )+-->
  <movie type = "enumeration" rating = "enumeration" review = "enumeration"
year = "text">
    <!--(title, writer+, producer+, director+, actor*, comments?)-->
    <title>only text</title>
    <writer>only text</writer>
    <producer>only text</producer>
    <director>only text</director>
    <actor>only text</actor>
    <comments>only text</comments>
  </movie>
</movie>

```

这段代码相当直接的指出了如何使用每一个元素和属性。请注意被列为 enumeration 或 text 的属性值说明你可以从枚举值中选择或是输入任何你需要的值。另外,movies 和 movie 元素的文件模型被列为注释,因为没有办法在元素定义中直接和内容模型通信。

6.4 理解内容模型

说到内容模型,上一章已经涉及了 XML Schema 是如何支持两种不同的内容模型类型:开放的和封闭的。开放的内容模型允许 Schema 开发者在未在文件 Schema 中声明的元素中使用子元素和属性。DTD 只支持封闭的内容模型,它要求你声明所有的元素和属性以便在文件中使用它们。毫无疑问,因为 XML Schema 才成为可能的开放的内容模型比 DTD 的实现更具有扩展性。

注意:在 XML Schema 中,内容模型缺省就是开放的。

开放的内容模型提供了许多灵活性,它允许你扩展 Schema 的词表而且同时可以创建有效的文件。比如你正在开发一个应用程序,它依赖于存储在 XML 文件中的数据。你发现一个现有的 Schema 几乎可以适合你的要求,但是它需要一些额外的元素和属性才能建立你的数据的模型。你可以用这些额外的元素和属性构成所有的文件,尽管这个 Schema 不支持它们。你的应用程序可以解析并识别这些额外的元素和属性。

在 XML Schema 文件中,你可以通过名字空间声明引用外部 Schema 来扩展一个开放的内容模型。比如,你可能想要在你的电影集中支持另外一种等级体系。假设电影批评家 Roger Ebert 的 thumbs-up/thumbs-down 的选择被模型化在文件 Ebert.xml 中,你可以像下面这样使用他的这种评价:

```
<movies xmlns="x-schema:MovieSchema.xml" xmlns:ebert="urn:ebert.xml">
  <movie type="comedy" rating="PG-13" review="5" year="1987">
    <title>Raising Arizona</title>
    <ebert:rating>thumbs-up</ebert:rating>
    <writer>Ethan Coen</writer>
    <writer>Joel Coen</writer>
    <producer>Ethan Coen</producer>
    <director>Joel Coen</director>
    <actor>Nicolas Cage</actor>
    <actor>Holly Hunter</actor>
    <actor>John Goodman</actor>
    <comments>A classic one-of-a-kind screwball love story.</comments>
  </movie>
</movies>
```

请注意 EbertSchema 和名字空间 ebert 相关,它允许你使用定义在这个 Schema 中的任何元素,只要在前面加上 ebert。在这个例子中,ebert:rating 被用在这个文件中来支持 thumbs-up/thumbs-down 电影等级体系。

当然,有些情况下你不想为其他开发者提供通过你的 Schema 来使用外部元素和属性的自由。在这种情况下,你可能想要设置 Schema 的内容模型,使之成为封闭的。因为 XML Schema 内容模型缺省是开放的,因此你必须使用 ElementType 元素的 model 属性来改变元素类型的内容模型。下面是一个设置 Schema 的根文件元素为封闭的内容模型的例子,它还使得所有子元素都使用这一模型:

```
<Schema name="MovieSchema"
```



```
xmins="urn:schemas-microsoft-com:xml-data"
xmins:dt="urn:schemas-microsoft-com:datatypes">
<ElementType name="movies" content="eltOnly" model="closed">
  <element type="movie" minOccurs="1" maxOccurs="*" />
</ElementType>
</Schema>
```

在这个例子中,model 属性被设置为 closed 以指定 movies 元素类型(以及所有的子元素)都遵守封闭的内容模型。你还可以设置 model 属性为 open,这样对未知的元素类型就使用开放的内容模型。

6.5 XML Schema 的未来

尽管 Internet Explorer 5.0 的 XML Schema 实现提供了一种使用文件 Schema 的实际方法,但是它仍然没有达到 W3C 的 XML Schema 工作草案的目标。W3C 所提出的 XML Schema 的全部功能,将极大地改变我们思考 XML 文件结构的方式。XML Schema 工作草案中描述了 Internet Explorer 5.0 尚不支持的两个主要功能是元素继承和受绑定的数据类型。

6.5.1 元素继承

元素继承涉及 DTD 中参数实体的限制。尽管参数实体在 DTD 中非常有用,但是它们不足以在共享同样属性的元素之间建立关系。即使两个元素通过参数实体共享相同的属性集,在这两个元素之间仍然没有任何联系,因为参数实体实质上是一种文本替换机制。换句话说,使用参数实体的元素不能包含任何类型的信息。

XML Schema 工作草案所定义的元素继承,使用面向对象的方法来支持元素之间的关系。在 XML Schema 中,你可以创建 archetype,它是元素的模板,你可以通过它来创建其他元素。在描述联系人清单的 Schema 中,你可能会创建一个 person archetype 来描述人的一般情况。之后,你可能会从这个 archetype 中取得更特别的元素,比如 friend、salescontact、supplier 等等。Friend 元素可能会加入一些个人信息,比如生日,而 salescontact 元素则可能会加入产品类型。

元素继承的作用是 XML 处理函数可以将取得的元素和其他元素建立关系并保持它们的关系。它使得比参数实体更进一步的处理成为可能,参数实体的方法不能保持元素之间的关系。

6.5.2 受约束的数据类型

XML Schema 工作草案中描述的另外一个强大的功能是受绑定的数据类型。Microsoft 当前的 XML Schema 实现支持扩展的 XML Schema 数据类型,但是它不支持对这些类型的绑定。对数据类型的绑定允许你紧紧地控制保存在指定元素或属性中的数据的范围和格式。比如,假设你创建了一个称为 bid 的元素来保存网上拍卖的竞价。如果允许的最低竞价是 50.00 美元,你会想要绑定 bid 元素使它不能低于 50.00 美元。

控制数据类型的范围非常有用,受绑定的数据类型在建立允许的数据格式上甚至更为强大。考虑电话号码的例子,通常都出现为 ###-#### 或 ###-###-#### 的

格式。在 XML Schema 中你可以指定电话号码的语法,这样电话号码的内容就必须符合这些格式之一。下面是一个的例子:

```
<datatype name="phone">
  <basetype name="string"/>
  <lexicalRepresentation>
    <lexical>999-9999</lexical>
    <lexical>999-999-9999</lexical>
  </lexicalRepresentation>
</datatype>
```

在这个例子中,lexicalRepresentation 和 lexical 元素被用来定义 phone 元素的语法表述。通过适当的语法表述,XML 处理函数可以验证文件中的 phone 元素是否以正确的格式输入。

6.6 总 结

尽管现在从 DTD 上跳开并全面使用 Schema 还有些早,但是 XML 共同体显然准备迁移到 Schema。Microsoft 的部分 XML Schema 实现成为 XML 开发者处理 XML Schema 的一个很好的开始。这也是本章讨论将 DTD 转化为 XML Schema 的长短,包括手工方法和自动方法的原因。这些方法同样重要,因为许多 DTD 转化要同时使用它们;首先使用工具来自动转化 DTD,然后手工调整转化的结果。

除了将 DTD 转化为 Schema 之外,本章还探讨了 Microsoft 的 XML Validator 工具,它可以用来根据 Schema 验证文件。我们还讨论了 XML Schema 对内容模型的支持,以及如何通过名字空间使用外部 Schema。本章以未来将会实现的 XML Schema 建设中的一些特征作为结束。

第 7 章 使用 XML 名字空间

XML 的一个主要目标就是创建包含自定义元素(标记)的标记词表。当你考虑在一个封闭的环境中创建自定义元素的时候,对元素的命名并没有什么严格的要求,因为你可以创建惟一的元素名。然而,当考虑更广泛一些的情况,比如在 Web 上使用许多自定义的 XML 词表时,惟一命名就成为一个即使不是不可能也是非常困难的一件事,除非使用某种附加的命名规则。幸运的是,的确有这么一种命名规则,而且它还包括了名字空间。

本章讨论了 XML 的名字空间以及如何用名字空间来保证 XML 词表的惟一性。你已经在前两章中大致了解了名字空间。但是,这些介绍实际上仅仅局限于名字空间的表面。本章将深入探讨名字空间,并且提供了说明名字空间在 XML 中重要性的完整例子。

7.1 名字空间基础

如果你已经用 C++ 或 Java 之类的语言编过程序,那么你或许对名字空间已经有所了解了。名字空间在编程语言中的作用是为定义在其他对象中的结构封装在指定对象中定义的程序结构的名称。Java 使用了名字空间的扩展来解决名字冲突的问题,它使得集成多个源代码而不必关心命名问题成为可能。

尽管 XML 不是一种编程语言,但是当处理元素名字的时候它面对了类似的问题。考虑前两章中的电影集 Schema。这个 Schema 定义了一个叫做 movie 的元素,它含有一个叫做 title 的子元素。但是如果你要扩展你的电影集,使它还包括音乐又会怎样呢?你可能会找到一种合适的 Schema 来表示音乐的信息,并且也使用名叫 title 的元素来保存音乐 CD 的名字。因为 XML Schema 支持使用多重 Schema,你可以同时使用这两种 Schema,但是 XML 处理函数是如何知道该怎样区别这两种不同的 title 元素呢?

你可能会说可以通过查看它所出现的地方来解析 title 元素。尽管这可能是不错,但是你将要求 XML 处理函数要具有更多的责任。而且在某些情况下这种方法还是不可能的。比如,如果发生冲突的元素名是文件的第一个元素,那么会怎样呢?因为它影响这个文件,所以要决定它的 Schema 是不可能的。底线是 XML 需要有一种方法来避免名字冲突。其解决方案就是 XML 名字空间。

XML 名字空间是一个 W3C 的提议,但是没有成为原始的 XML 1.0 规范的一部分。在那个时候,W3C 还没有确定名字空间的细节以及如何在 XML 中结构化并使用它。XML 名字空间在 1999 年 1 月成为了 W3C 的建议的一部分,因此它还是一个比较新的技术。然而,它已经迅速被采纳,这主要是因为有这样的需求。许多开发者都希望在 XML 文件中避免名字冲突的发生。

注意:关于 XML 名字空间的 W3C 规范的进一步信息,请参阅下面的网址:<http://www.w3.org/TR/REC-xml-names>。

XML 名字空间是如何确保 XML 元素和属性的名称的惟一性的呢? 这个用来保证惟一性的技术相当聪明而且还使用了 URI(统一资源标识符)。因为 URI 通常引用 Internet 上的物理资源,所以它们是惟一的。比如,我的 Web 站点是 `http://www.thetribes.com`。为了保证在我创建的所有 Schema 中确保名字的惟一性,在创建基于某种 Schema 的文件时,我可以将我的名字空间和 Schema 联系起来。这样的名字空间声明可能如下:

```
<movies xmlns:movie="http://www.thetribes.com/movies">
```

你可能还使用 Web 资源的 URN(统一资源命名)来保证惟一性,而不是 URL。URN 和 URL 有一点点不同,它为映射到一个或多个 URL 上的资源定义了一个惟一的,与位置无关的名字。下面是使用 URN 来指定我的 Web 站点的名字空间的例子:

```
<movies xmlns:movie="urn:thetribes.com:movies">
```

XML 开发者经常会出现 URL、URN 和 URI 的混淆。也许最主要的区别就是 URI 包含了 URL 和 URN。换句话说,URI 是包含了 URL 和 URN 的一个超集。URN 区别于 URL 的地方是 URL 描述了特定资源的物理位置,而 URN 为映射到一个或多个 URL 上的资源定义了惟一的、与位置无关的名字。一个区别 URL 和 URN 的简单办法就是检查它们的名字:URL 都以 Internet 服务前缀如 `ftp:`、`http:` 等等开头,而 URN 一般都以 `urn:` 开头。更令人混淆的是,在 URN 和 URL 之间有一些重叠,但是这就在我们的讨论之外了。

7.2 声明名字空间

XML 文件中使用名字空间声明来声明名字空间,其具有如下格式:

```
xmlns:Prefix="Namespace"
```

Xmlns 关键字告诉 XML 处理函数正在声明名字空间。名字空间声明的 Prefix 部分允许你设置前缀,来简化声明名字空间的元素中对名字空间的引用。名字空间声明的 Prefix 部分是可选的,而且最终决定你要在文件中使用验证的还是非验证的元素和属性名。验证的名字包括名字空间声明的 Prefix 部分而且包含两部分:前缀和名字的本地部分。验证名字的例子有 `movie:title`、`movie:director` 以及 `movie:rating`。要使用验证名字,你必须在名字空间声明中提供 Prefix。

非验证名字不包括前缀而且和缺省的名字空间相关,或者根本和名字空间无关。在声明缺省名字空间时,不必指定名字空间声明的 Prefix。非验证的名字的例子有 `title`、`director` 以及 `rating`。

回到名字空间声明,Namespace 部分是名字空间标识自己的地方。声明的这部分标识了一个用来保证在名字空间声明的范围内元素和属性名字的惟一性 URI。

文件中使用验证还是非验证名字决定了你以哪种方式来声明名字空间。有两种不同的方法来声明名字空间:

- 缺省声明——名字空间的声明没有前缀,在它范围内所有的元素和属性名字都作为非验证名字引用并且假设在这个名字空间之中。

- 显式声明——名字空间的声明带有前缀,在名字空间中的所有元素和属性名字都必须使用前缀作为他们验证名字的一部分。

下面的几节将进一步探讨这些名字空间声明。

注意:理解名字空间是用在XML文件中而不是用在DTD中的这一点是非常重要的。尽管名字空间被用来惟一标识DTD中的元素和属性,但是名字空间本身一般用在XML文件之中。

7.2.1 缺省声明

缺省声明是声明名字空间的两种方法中比较简单的一种。如果你想要使名字空间应用于这个文件或者文件的某一部分,那么它是非常有用的。在声明缺省名字空间时,你不必在xmlns关键字中使用前缀。因此,名字空间应用于名字空间声明范围内的所有非验证元素。缺省声明的一个很好的例子就是用来创建XML Schema文件的XML-Data名字空间。下面的名字空间声明已声明了这个名字空间,它必须出现在所有XML Schema文件中:

```
<Schema name="MovieSchema">
  xmlns="urn:schemas-microsoft-com:xml-data">
  <!-- Schema content goes here -->
</Schema>
```

在这个例子中,XML-Data名字空间(urn:schemas-microsoft-com:xml-data)被定义为文件的缺省名字空间。这意味着声明范围内的Schema元素和所有非验证元素都被认为是XML-Data名字空间的一部分。注意,XML-Data使用URN作为名字空间的名字。

注意:缺省名字空间可以被设置为空字符串(xmlns=""),这意味着在声明的范围内没有缺省的名字空间。

7.2.2 显式声明

当你要创建依赖于多个名字空间的文件时,显示名字空间声明非常有用。你可以声明一个缺省的名字空间,但是你必须使用显式名字空间声明来表示附加的名字空间。显式声明必须有一个前缀来决定元素和属性到底属于那个名字空间。显式声明的一个很好的例子是XML Schema Data Type名字空间,如果你使用任何XML Schema的数据类型,如int、float以及time,那么你必须使用它。

在显式声明中的前缀被用来在名字空间声明的范围内简化名字空间的标识。更确切地说,这个前缀和本地元素或属性的名字一起构成Prefix:Local这样的验证名字。下面是声明XML Schema Data Type名字空间的一个例子:

```
<Schema name="MovieSchema"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatype">
  <ElementType name="title" content="textOnly"/>
</Schema>
```

正如你所见到的,XML-Data名字空间仍然被声明为缺省名字空间,但是XML Schema Data Type名字空间也被显式地声明以提供对XML Schema的数据类型的访问。在这种情

况下,前缀 dt 被用来标识属于 XML Schema Data Type 名字空间的元素和属性。下面的代码显示了如何在一个 Schema 文件中使用这个前缀:

```
<Schema name="MovieSchema"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatype">
  <AttributeType name="type" dt:type="enumeration"
    dt:values="drama comedy adventure sci-fi mystery horror romance documentary"
    default="drama" />
</Schema>
```

这段代码仅仅是电影集 Schema 的一部分,但是它说明了如何使用显式声明的名字空间以及验证的名字。所有使用非验证名字的元素和属性都属于 XML-Data 名字空间,而 dt: 和 dt:values 元素属于 XML Schema Data Type 名字空间。

7.3 将 Schema 作为名字空间引用

关于这个问题,我曾经对 Schema 究竟是怎样和名字空间联系起来的这个问题非常不解。XML 处理函数是如何知道怎样通过名字空间声明来访问 Schema 并用它来验证文件的呢?实际上这和处理文件的应用程序有关,目前没有多少应用程序支持名字空间。但有一个应用程序支持它,那就是 Internet Explorer 5.0。

在 Internet Explorer 5.0 中,Schema 和名字空间通过许多不同的方式联系起来。第一种方法是通过 URN,这也是 XML-Data 和 XML Schema Data Type 名字空间与它们对应的 Schema 联系起来的方法(正如你在上一节中看到的)。通过在 Schema 声明的 URI 中指定 Schema 文件的名称你还可以引用你自己的 Schema。在这种方法中,你必须在 SchemaURI 前面加上字符串 x-schema:,它告诉 Internet Explorer 这是一个外部 Schema。下面的示例说明在一个介质集文件中这一切是如何实现的,它假设电影和音乐 Schema 文件都和文件在相同的目录之中:

```
<mediacollection xmlns:movie="x-schema:MovieSchema.xml"
  xmlns:music="x-schema:MusicSchema.xml">
  <!-- movie and music content goes here -->
</mediacollection>
```

当 Internet Explorer 5.0 的 XML 处理函数看到名字空间声明的时候,它就知道要加载 MovieSchema.xml 和 MusicSchema.xml 文件作为 Schema 并且用它们来验证介质集文件。更确切地说,前缀 movie 和名字空间 x-schema:MovieSchema.xml 相绑定,这使得电影 Schema 被加载并被用来验证文件。音乐 Schema 的引用也是一样。

上一个例子说明了通过在名字空间声明中声明,如何在一个文件中同时使用两种 Schema。清单 7.1 包含了修改后的电影集 Schema 的代码,它被一般化以便可以和介质集文件一起工作。

清单 7.1 一般化了的电影集 Schema

```

<? xml version="1.0"? >
<Schema name="MovieSchema"
  xmlns="urn:schemas-microsoft-com:datatypes">
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <ElementType name="title" content="textOnly"/>
    <ElementType name="writer" content="textOnly"/>
    <ElementType name="producer" content="textOnly"/>
    <ElementType name="director" content="textOnly"/>
    <ElementType name="actor" content="textOnly"/>
    <ElementType name="comments" content="textOnly"/>
    <AttributeType name="type" dt:type="enumeration"
      dt:values="
        drama comedy adventure sci-fi mystery horror romance documentary"
      default="drama"/>
    <AttributeType name="rating" dt:type="enumeration"
      dt:values="G PG PG-13 R X" default="PG"/>
    <AttributeType name="review" dt:type="enumeration"
      dt:values="1 2 3 4 5" default="3"/>
    <AttributeType name="year" dt:type="int"/>
    <ElementType name="movie" content="eltOnly" order="seq">
      <element type="title" minOccurs="1" maxOccurs="1"/>
      <element type="writer" minOccurs="1" maxOccurs="*/>
      <element type="producer" minOccurs="1" maxOccurs="*/>
      <element type="director" minOccurs="1" maxOccurs="*/>
      <element type="actor" minOccurs="0" maxOccurs="*/>
      <element type="comments" minOccurs="0" maxOccurs="1"/>
      <attribute type="type"/>
      <attribute type="rating"/>
      <attribute type="review"/>
      <attribute type="year"/>
    </ElementType>
  </Schema>

```

和你在前面几章中看到的那个 Schema 的主要不同在于这个 Schema 没有为 movie 元素定义容器元素。相反,它允许介质集文件在 movie 元素出现的时候提供容器元素。类似的修改也发生在音乐集 Schema 中,如清单 7.2。

清单 7.2 一般化了的音乐集 Schema

```

<? xml version="1.0"? >
<Schema name="MusicSchema"
  xmlns="urn:schemas-microsoft-com:xml-data">
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <ElementType name="title" content="textOnly"/>

```

```

<ElementType name="artist" content="textOnly"/>
<ElementType name="label" content="textOnly"/>
<ElementType name="comments" content="textOnly"/>
<AttributeType name="length" dt:type="time"
<AttributeType name="medianum" dt:type="int"
<AttributeType name="tracknum" dt:type="int"
<ElementType name="track" content="textOnly">
  <attribute type="length"/>
  <attribute type="medianum"/>
  <attribute type="tracknum"/>
<ElementType>
<ElementType name="tracks" content="eltOnly">
  <element type="track" minOccurs="1" maxOccurs="*" />
<ElementType>
<AttributeType name="type" dt:type="enumeration"
  dt:values="jazz blues rock country hip-hop pop indy other" default="jazz"/>
<AttributeType name="review" dt:type="enumeration"
  dt:values="1 2 3 4 5" default="3"/>
<AttributeType name="year" dt:type="int"/>
<ElementType name="music" content="eltOnly" order="seq">
  <element type="title" minOccurs="1" maxOccurs="1"/>
  <element type="artist" minOccurs="1" maxOccurs="*" />
  <element type="label" minOccurs="1" maxOccurs="*" />
  <element type="comments" minOccurs="0" maxOccurs="1"/>
  <element type="tracks" minOccurs="1" maxOccurs="1"/>
  <attribute type="type"/>
  <attribute type="review"/>
  <attribute type="year"/>
</ElementType>
</Schema>

```

这个 Schema 和电影集 Schema 非常相似,只不过它建立了音乐相关的模型(CD、磁带等等)而不是电影。有些元素和属性与电影 Schema 依然十分类似,这也正是当你在介质集文件中使用两种 Schema 时名字空间如此重要的原因。对于介质集文件,清单 7.3 是介质集文件的一个例子,它说明了如何用两个 Schema 中的元素和属性来合成数据。

清单 7.3 使用了电影集和音乐集 Schema 的介质集 XML 文件

```

<? xml version="1.0" standalone="no"? >
<mediacollection xmlns:movie="x-schema:MovieSchema.xml"
  xmlns:music="x-schema:MusicSchema.xml">
  <movie:movie type="comedy" rating="PG-13" review="5" year="1987">
    <movie:title>Raising Arizona</movie:title>
    <movie:writer>Ethan Coen</movie:writer>
    <movie:writer>Joel Coen</movie:writer>
    <movie:producer>Ethan Coen</movie:producer>
  
```

```

    <movie:director>Joel Coen</movie:director>
    <movie:actor>Nicolas Cage</movie:actor>
    <movie:actor>Holly Hunter</movie:actor>
    <movie:actor>John Goodman</movie:actor>
    <movie:comments>A classic one — of — a — kind screwball love story. </movie:
comments>
  </movie:movie>

  <movie:movie type="comedy" rating="R" review="5" year="1988">
    <movie:title>Midnight Run</movie:title>
    <movie:writer>George Gallo</movie:writer>
    <movie:producer>Martin Brest</movie:producer>
    <movie:director>Martin Brest</movie:director>
    <movie:actor>Robert De Niro</movie:actor>
    <movie:actor>Charles Grodin</movie:actor>
    <movie:comments>The quintessential road comedy. </movie:comments>
  </movie:movie>

  <movie:movie type="mystery" rating="R" review="5" year="1995">
    <movie:title>The Usual Suspects</movie:title>
    <movie:writer>Christopher McQuarrie</movie:writer>
    <movie:producer>Bryan Singer</movie:producer>
    <movie:producer>Michael McDonnell</movie:producer>
    <movie:director>Bryan Singer</movie:director>
    <movie:actor>Stephen Baldwin</movie:actor>
    <movie:actor>Gabriel Byrne</movie:actor>
    <movie:actor>Benicio Del Toro</movie:actor>
    <movie:actor>Chazz Palminteri</movie:actor>
    <movie:actor>Kevin Pollak</movie:actor>
    <movie:actor>Kevin Spacey</movie:actor>
    <movie:comments>A crime mystery with incredibly intricate plot twists. </movie:
comments>
  </movie:movie>

  <movie:movie type="sci-fi" rating="PG-13" review="4" year="1972">
    <movie:title>The Abyss</movie:title>
    <movie:writer>James Cameron</movie:writer>
    <movie:producer>Gale Anne Hurd</movie:producer>
    <movie:director>James Cameron</movie:director>
    <movie:actor>Ed Harris</movie:actor>
    <movie:actor>Mary Elizabeth Mastrantonio</movie:actor>
    <movie:comments>A very engaging underwater odyssey. </movie:comments>
  </movie:movie>

  <music:music type="indy" review="5" year="1990">
    <music:title>Cake</music:title>
    <music:artist>The Trash Can Sinatras</music:artist>
    <music:label>Polygram Records</music:label>
    <music:comments>Excellent acoustical instruments and extremely witty
    lyrics. </music:comments>
    <music:tracks>
      <music:track length="04:13" medianum="1" tracknum="1">
        Obscurity Knocks
      </music:track>
      <music:track length="03:40" medianum="1" tracknum="2">
        Maybe I Should Drive
      </music:track>
    </music:tracks>
  </music:music>

```



```

<music:track length="05:15" medianum="1" tracknum="3">
  Thrupenny Tears
</music:track>
<music:track length="03:25" medianum="1" tracknum="4">
  Even the Odd
</music:track>
<music:track length="03:43" medianum="1" tracknum="5">
  The Best Man's Fall
</music:track>
<music:track length="02:40" medianum="1" tracknum="6">
  Circling the Circumference
</music:track>
<music:track length="04:16" medianum="1" tracknum="7">
  Funny
</music:track>
<music:track length="03:45" medianum="1" tracknum="8">
  Only Tongue Can Tell
</music:track>
<music:track length="04:10" medianum="1" tracknum="9">
  You Made Me Feel
</music:track>
<music:track length="04:47" medianum="1" tracknum="10">
  January's Little Joke
</music:track>
</music:tracks>
</music:music>

<music:music type="rock" review="5" year="1985">
  <music:title>Love</music:title>
  <music:artist>The Cult</music:artist>
  <music:label>Sire Records</music:label>
  <music:comments>The dark moody beginnings of The Cult.</music:comments>
  <music:tracks>
    <music:track length="05:24" medianum="1" tracknum="1">
      Nirvana
    </music:track>
    <music:track length="04:56" medianum="1" tracknum="2">
      Big Neon Glitter
    </music:track>
    <music:track length="05:31" medianum="1" tracknum="3">
      Love
    </music:track>
    <music:track length="06:47" medianum="1" tracknum="4">
      Brother Wolf, Sister Moon
    </music:track>
    <music:track length="03:55" medianum="1" tracknum="5">
      Rain
    </music:track>
    <music:track length="05:04" medianum="1" tracknum="6">
      The Phoenix
    </music:track>
    <music:track length="04:45" medianum="1" tracknum="7">
      Hollow Man
    </music:track>
    <music:track length="05:25" medianum="1" tracknum="8">

```

```

    Revolution
  </music:track>
  <music:track length="04:21" medianum="1" tracknum="9">
    She Sells Sanctuary
  </music:track>
  <music:track length="05:25" medianum="1" tracknum="10">
    Black Angel
  </music:track>
</music:tracks>
</music:music>

<music:music type="rock" review="5" year="1991">
  <music:title>Travelers and Thieves</music:title>
  <music:artist>Blues Traveler</music:artist>
  <music:label>A&M Records</music:label>
  <music:comments>The best Blues Traveler recording, period.</music:comments>
  <music:tracks>
    <music:track length="01:30" medianum="1" tracknum="1">
      The Tiding
    </music:track>
    <music:track length="06:08" medianum="1" tracknum="2">
      Onslaught
    </music:track>
    <music:track length="05:15" medianum="1" tracknum="3">
      Ivory Tusk
    </music:track>
    <music:track length="03:45" medianum="1" tracknum="4">
      What's for Breakfast
    </music:track>
    <music:track length="04:12" medianum="1" tracknum="5">
      I Have My Moments
    </music:track>
    <music:track length="03:28" medianum="1" tracknum="6">
      Optimistic Thought
    </music:track>
    <music:track length="04:49" medianum="1" tracknum="7">
      The Best Part
    </music:track>
    <music:track length="07:41" medianum="1" tracknum="8">
      Sweet Pain
    </music:track>
    <music:track length="04:15" medianum="1" tracknum="9">
      All In the Groove
    </music:track>
    <music:track length="06:54" medianum="1" tracknum="10">
      Support Your Local Emperor
    </music:track>
    <music:track length="04:21" medianum="1" tracknum="11">
      Bagheera
    </music:track>
    <music:track length="09:07" medianum="1" tracknum="12">
      Mountain Cry
    </music:track>
  </music:tracks>
</music:music>

```

```

<music:music type="jazz" review="5" year="1974">
  <music:title>Live Evil</music:title>
  <music:artist>Miles Davis</music:artist>
  <music:label>Sony Records</music:label>
  <music:comments>Vintage Miles early in his fusion era. </music:comments>
  <music:tracks>
    <music:track length="15:19" medianum="1" tracknum="1">
      Sivad
    </music:track>
    <music:track length="03:18" medianum="1" tracknum="2">
      LittleChurch
    </music:track>
    <music:track length="05:57" medianum="1" tracknum="3">
      Medley — Gemini — Double Image
    </music:track>
    <music:track length="21:13" medianum="1" tracknum="4">
      What I Say
    </music:track>
    <music:track length="04:06" medianum="1" tracknum="5">
      Nem Um Talvez
    </music:track>
    <music:track length="02:17" medianum="2" tracknum="1">
      Selim
    </music:track>
    <music:track length="23:31" medianum="2" tracknum="2">
      Funky Tonk
    </music:track>
    <music:track length="26:32" medianum="2" tracknum="3">
      Inamorata and Narration
    </music:track>
  </music:tracks>
</music:music>
</mediacollection>

```

对于这个清单的长度我深表歉意,但是我想要说明在一个文件中使用多个 Schema 所带来的灵活性。请注意在这个文件中大量使用前缀来创建验证名字以引用显式名字空间声明。

7.4 在 Schema 中使用名字空间

因为 XML Schema 本身是 XML 文件,所以你可以在其中使用名字空间来引用 Schema。正是这种能力使得 XML Schema 成为一种具有扩展性的技术——它可以通过其他 Schema 中的元素和属性来扩展 Schema。XML Schema 文件必须支持开放内容模型以便利用这种能力。幸运的是,所有的 XML Schema 文件都缺省地具有开放内容模型。

因为 Internet Explorer 5.0 中微软实现的 XML Schema 不支持绑定的数据类型,因此 Schema 中的名字空间是使用这种功能的绝好例子。更确切的说,对于许多你想要将其限制在一个可接受的取值范围内的数值属性来说,具有 min 和 max 属性是非常有用的。一个 XML 应用程序可以处理包含这些受绑定的属性文件并且确认它们在取值区间之内。清单

7.4 包含了一个受绑定的数据 Schema,它定义了 min 和 max 属性。

清单 7.4 受绑定的数据 Schema

```
<? xml version="1.0" ? >
<Schema name="TrainSchema"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes"
  <ElementType name="attribute" content="empty">
    <AttributeType name="min" dt:type="int" />
    <attribute type="min" />
    <AttributeType name="max" dt:type="int" />
    <attribute type="max" />
  </ElementType>
</Schema>
```

请注意 min 和 max 属性被定义为 attribute 元素类型。这意味着 min 和 max 可以被用来绑定定义在 Schema 中的属性。更重要的是,这意味着你正在扩展 XML Schema 词表,它具有不可估量的能力。清单 7.5 显示了第 5 章中运动训练 Schema 的一个新版本,它使用了 min 和 max 属性。

清单 7.5 新的具有受绑定的心率属性的运动训练 Schema

```
<? xml version="1.0" ? >
<Schema name="TrainSchema"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes"
  xmlns:cd="x-schema:ConstrainedDataSchema.xml" >
  <ElementType name="duration" content="textOnly" dt:type="time" />
  <ElementType name="distance" content="textOnly" dt:type="float" />
  <ElementType name="location" content="textOnly" />
  <ElementType name="comments" content="textOnly" />
  <AttributeType name="type" dt:type="enumeration"
    dt:values="running cycling swimming" />
  <AttributeType name="date" dt:type="date" />
  <AttributeType name="heartrate" dt:type="int" />
  <ElementType name="session" content="eltOnly" order="seq" >
    <description>
      This element type represents a single training session.
    </description>
    <element type="duration" minOccurs="1" maxOccurs="1" />
    <element type="distance" minOccurs="1" maxOccurs="1" />
    <element type="location" minOccurs="1" maxOccurs="1" />
    <element type="comments" minOccurs="0" maxOccurs="1" />
    <attribute type="type" default="running" />
    <attribute type="date" />
    <attribute type="heartrate" cd:min="60" cd:max="220" />
  </ElementType>
```

```

<ElementType name="trainlog" content="eltOnly">
  <description>
    This element type represents training log consisting of one or more
    training sessions.
  </description>
  <element type="session" minOccurs="1" maxOccurs="*" />
</ElementType>
</Schema>

```

这段代码显示了名字空间是如何使用 min 和 max 属性的。这些属性由 cd 前缀引用,它建立了一个输入绑定数据 Schema 的名字空间的声明。受 min 和 max 属性绑定的属性是 heartrate,它指定了在某个特定的训练阶段中所测量的平均心率,注意 XML-Data 规范被声明为缺省 Schema,而 XML Schema Data TypeSchema 被声明为在 dt 前缀后。这意味着在运动训练 Schema 中有三个不同的 Schema 一起工作。

清单 7.6 包含了使用新的具有受绑定的 heartrate 属性的运动训练 Schema 运动训练 XML 文件的例子。

清单 7.6 使用新的具有受绑定的 heartrate 属性的运动训练 Schema 的运动训练 XML 文件

```

<? xml version="1.0" standalone="no"? >
<trainlog xmlns="x-schema:TrainSchema.xml">
  <session type="running" date="12/15/99" heartrate="155">
    <duration>01:00:00</duration>
    <distance>5.5</distance>
    <location>Warner Park</location>
    <comments>Late afternoon run, felt strong. </comments>
  </session>
  <session type="cycling" date="12/17/99" heartrate="153">
    <duration>02:10:00</duration>
    <distance>37.0</distance>
    <location>Natchez Trace Parkway</location>
    <comments>Hilly ride, felt weak toward the end. </comments>
  </session>
  <session type="running" date="12/18/99" heartrate="160">
    <duration>01:20:00</duration>
    <distance>8.5</distance>
    <location>Warner Park</location>
    <comments>Mid-morning run, tough pace. </comments>
  </session>
</trainlog>

```

min 和 max 属性没有出现在这个文件中,因为它们在 Schema 中已经出现。然而,处理这个文件的应用程序仍然会将 min 和 max 作为绑定 heartrate 属性的方法来保证它在可接受的取值区间之中。

7.5 名字空间和文件对象模型

尽管直到第 3 部分“XML 文件的处理”时你才学习到文件对象模型(DOM)的细节,但

是在这里看一下在 DOM 中如何访问名字空间还是值得的。如果你不太熟悉 DOM, 你可以将 DOM 看作是用来访问和操纵 XML 文件的程序接口。DOM 接口是平台无关并且是语言无关的, 这意味着它可以被用来在任何平台上开发程序和脚本, 并动态地访问和修改 XML 文件的内容和结构。

DOM 是由层次性对象接口构成的, 这些接口表示了 XML 文件的结构。每一个 DOM 对象通常都描述了文件特定的节点(元素或属性)并且包含关于这个对象的信息的属性, 比如它的名字。下面是关于名字空间的 DOM 对象属性, 以及它们包含的信息:

- NamespaceURI——节点的名字空间
- NodeName——节点的验证名字
- Prefix——节点的名字空间前缀
- BaseName——节点的本地名字

这些属性可以被程序或脚本访问以取得关于元素或属性的名字空间的信息。

7.6 总 结

名字空间解决了 XML 固有的一个重要问题, 这就是在自定义词表中的名字冲突。没有名字空间, 在无数开发出来以及 Web 上使用的 XML 词表间是不可能保证惟一性的。我可能会创建一个词表, 它具有名叫 productid 的元素以在一个基于 Web 的仓储应用中标识我的一个待销产品。如果我决定要在这个应用程序中集成其他的 XML 词表, 而这个词表也使用了名叫 productid 的元素, 那么应用程序将不可能区别这两种元素。通过提供了一种在 XML 词表中的元素和属性间建立惟一性的机制, 名字空间解决了这个问题。

第2部分 格式化XML文件

第8章 对比两种样式规范:可扩展 样式表语言和层叠样式表

XML 被用来标记文件内容。理想情况下,它使得我们可以不再考虑在网络浏览器和其他应用程序上是如何处理文件内容显示的。为了显示 XML 文件,你必须特别定义与多个不同的元素和属性相关的字体尺寸和颜色值这类的显示信息。对 XML 文件进行样式格式化的一般应用结构被称作样式表。我们在这里说“一般”主要是因为,现在有两种不同的途径使用样式表来样式化 XML 文件样式:可扩展样式表语言(CSS,Cascading Style Sheets)和层叠样式表(XSL,eXtensible Style Language)。

本章将要探讨 XSL 与 CSS 之间的关系,并着重讲解每一种途径的重要性。在 XML 组织中,人们对于哪一种样式表技术更适合于将来的 XML 技术而进行过数次激烈的争论。我将尝试突破这些争论来说明这两项技术不仅可以和平的共存还可以互相完善互为补充。在本部分“格式化 XML 文件”的后面章节中将着重讲解使用 CSS 和 XSL 创建样式表的各自特点。让我们就把这一章作为对这两项技术的简要介绍,更重要的是要通过这一章了解围绕这两项技术的争论。

8.1 样式表基础

要理解 XSL 和 CSS,你首先要理解样式表后面的基本前提。显然,Web 在样式表引入图片以前可以设计的很好,那么什么是重要事物呢。

这个重要事物主要指的就是 Web 团体,特别是 XML 团体认识到将内容标记与描述布局样式混合的方法在维护、灵活性和复杂性方面是昂贵的。HTML 就是明显的例子,当你随机地将内容和描述放入到单个标记语言中时上述情况就会发生。

8.1.1 HTML 是一种非常糟糕的描述语言

实际上,HTML 从没有被设计成一种描述式的语言。实际上,标记语言普遍被固定化地设计成一种使用元数据来描述文件结构的语言,它对于文件是如何显示的毫不知情。解决的办法是使用标记来描述文件的内容,为了满足显示效果,你还要将布局样式应用于内容并加以修饰。SGML 提供了必要的语法来创建可以用于这种方式的标记语言,同时还有很多语言可应用这种方式。但 HTML 并不是其中的一种。

HTML 是 SGML 的一种应用。通过使用独创的表单,HTML 的原始形式只是一种单纯

的基于内容的标记语言(它的设计目的是允许物理学家们共享技术资料)。较早的浏览器当然可以支持显示 HTML 文件,但是,是这种浏览器而不是 HTML 标记来决定文件的布局。例如,在某一个浏览器中,所有标记了<P>标记的段落将会按照 12 点大小的 Times New Roman 字体显示,而另一种浏览器可能会用 14 点的 Courier 字体。这是因为浏览器已经设置了布局效果,而不是通过文件自身来设置的。而布局效果设置本应是由标记语言来承担的。

随着网络作为一种电子传输媒介以一种令人不可思议的速度迅速发展,HTML 被用了一些原设计本没有考虑到的领域。更重要的是,Marc Andreessen(Netscape 创始人)在 HTML 中加入了标记,事情就已经不像原来那么简单了。Netscape 之后又加入了<blink>标记。这种改动就更前进了一大步,它标志着结束网络上仅限于内容标记的时代开始了,至少对于 HTML 来说是这样。一旦描述的闸门开启,HTML 迅速转变为一个拥有内容和描述标记的杂乱的混合体。要记住标记从没有打算加入描述信息。然而,HTML 是网络文件的基础,因此它只能尽量扩展自身来创建好看的网络页面。当然,随之而来的,浏览器提供商们也开始在加入最好的新颖的特有描述特征上开始了一场小小的竞争。

结果就是由于各种不同的理由,每个人潜心于发展 HTML 而导致网络失去原来简单的方式。没人会想到在一些年后对 HTML 添加描述标记后还要添加什么标记。现在网络开发组织已经推翻以往的做法并退回到 SGML 和一般标记的基础前提,这里将描述和内容做了区分。样式表使得翻新改进 HTML 并理清内容和描述的混乱这件工作成为了可能。

8.1.2 将描述踢出 HTML

一个样式表定义了布局原则,它告诉一个浏览器如何显示一个 HTML 文件的各个部分。更重要的是,样式表的规则将一个文件的逻辑结构转换为一个适合于描述的表单。样式表对于 HTML 非常的重要,这是因为它们提供了一种使用 HTML 标记纯粹的描述网页内容的方法。这也是样式表的最初目的。

很抱歉我把这章变成了对 HTML 的讨论,但是样式表用它们对 HTML 施加影响的同一种方式影响着 XML。然而,与 HTML 不同,XML 并不提供欺骗性和在一个文件中加入描述信息的方式。XML 是一种纯粹基于内容的解析式语言,因此它完全地依赖于样式表来满足显示的要求。所以,尽管样式表改善了对 HTML 文件的结构和组织,实质上,它对于显示 XML 文件来说是完全必要的。

8.1.3 通过样式表显示 XML

在某些情况下,一个 XML 文件通过样式表的显示方式仍然要依赖于 XML 应用程序(浏览器)。例如,在样式表中被样式化成重点的文本在一个传统的浏览器中可能会按照斜体字显示,但是为了加强视觉效果,实际上它在浏览器中应加粗显示。这种区别对于创建样式表并没有大的影响,但在之后创建新的网络允许的设置时要好好注意。这些新设置中的一部分可能会按照与我们通常习惯的截然不同的方式显示文件。

不同设置通过不同方式显示 XML 文件的方法称为交错媒体描述(cross-medium rendering),因为各设备一般代表着不同的媒体。历史上,HTML 在交错媒体描述上争论不休,这也导致了不同的浏览器支持不同的描述标记。即使样式表减轻了交错媒体浏览器的问

题,它们还是不能完全处理好交错媒体问题。

为了理解这一点,考虑一下 CSS 样式表,它提供了一种应用布局规则的方式来显示 XML。CSS 在样式化 XML 上的过分简单化的方法在处理交错媒体问题上的功能并不足够强大,这主要是由于它不能将 XML 文件转换为其他的格式。而这种转换是成功的显示含有不同媒体的文件所必需的。而实际上这并没有打击 CSS,因为它本来就不是设计用来转换文件的。

可以转换文件的样式表技术是 XSLT(XSL Transformation,可扩展类型语言转换),它是 XSL 技术的子集。XSLT 完全支持 XML 文件从一种格式到另一种格式的转换。因此,你可以获取一个 XML 文件并将它转换为 HTML 3.2 或 HTML 4.0 格式的文件或其他自定义的 XML 文件格式。此外,由于 XSLT 是一种转换技术,你可以使用 CSS 来辅助样式化最终的输出显示。这里可能会比较混乱,因为 XSLT 和 CSS 都是样式表技术,但实际上,它们强调处理不同的问题。

注意:XSL 技术支持两种组件:XSLT 和 XSL 格式化对象。写这本书的时候,XSLT 基本上已是 W3C 推荐的组件,而 XSL 格式化对象则还没有出头之日。

XSLT 强调根据一套高级结构化的模式来转换 XML 文件的需求。为了实现显示,你可以使用 XSLT 将一个 XML 文件转换为一个 HTML 文件。这也是 XML 开发人员使用 XSL 的最主要目的,因为它并不比 XSLT 对处理函数的支持要求需要更多的浏览器的支持要求。他们并不需要能够在一个直接来自于 XML 的文件上处理显示。CSS 不包含任何的转换功能,它只提供一个描述文件的不同部分如何显示的方法。

一些人认为 XSL 和 CSS 是对立的技术,而实际上并非如此。要承认即将来临的 XSL 格式化对象将可以做 CSS 能够做的任何事情,并且还可以做得更多。然而,它被设计成一个 CSS 的功能扩展集,也就是说它将有效的取代 CSS 而不是与它竞争。你将在第 10 章“理解 XSL”中学到有关 XSL 格式化对象的更多知识。

8.2 CSS 和 XSL 发展史

虽然样式表只能在未来的一年到两年中在 HTML 世界中流行,但它们已经存在了一段时间。实际上,几年以前,早期的 SGML 系统就使用样式表将内容从描述中分离了出来。只是我们中大多数人都在忙于制造 HTML 表单以至于根本不知道。甚至早期的 HTML 浏览器也使用内部样式表以解决如何显示 HTML 文件数据的问题。有一段时间,标记从来不为人所知,浏览器决定一个文件如何显示。

直到 1994 年,CSS 第一次被开始讨论,可能是又过了几年,它被正式的提交给 W3C。在 1996 年 12 月它被 W3C 正式推荐。这里推荐的 CSS 被称为 CSS1,它基于 CSS 第一级。非常令人惊奇,Microsoft 在 1996 年 8 月发布的 Internet Explorer 3.0 中支持 CSS1 的早期版本。在通过 W3C 推荐后,有近一年的时间 Internet Explorer 4.0 依旧只提供对 CSS1 的部分支持。甚至在本书编写时,Internet Explorer 5.0 依然没有依照 W3C 的推荐完全的支持 CSS1。这件事在网络发展组织中引起了不小的轰动。

注意:在 1999 年 1 月,美国专利局批准了 Microsoft 的大量专利,包括“在电子出版

系统中使用样式表”。这个专利将引起网络发展组织的重大分歧,因为它描述了包括CSS和XSL样式表的多种重要思想。本质上,Microsoft正在声称它领导了一个将样式表应用于文件的新的机制——一个la CSS或XSL。这是一个可疑的专利,因为它给予了Microsoft决定样式表命运的法律上的优势。Microsoft声称它只是为了好的目的,声称它申请专利是为了网络标准。如果Microsoft诚实的话,它就应该将专利转还给W3C,并且让一个标准组织而不是一个执著于开发既定的私有技术的私人公司来控制标准。

不要认为Microsoft是惟一的慢慢提供对CSS支持的浏览器供应商,Netscape于1997年6月发布的Navigator 4.0已经部分的支持了CSS1。在本书编写时,Navigator 5.0还在开发当中,但它已经体现出了对CSS1的非常强大的支持。然而,CSS1现在已经是老新闻了。在1998年5月,CSS2(CSS第二级)已经在W3C达到了推荐的地位。在CSS语言的适应性上,CSS2取得了重大的飞跃。然而由于主流浏览器的提供商还正在为完全支持CSS1而奋斗,因此CSS2在这段时间上并不是一个好的方案。

注意:如果你确实希望了解主流浏览器在CSS支持方面的进展情况,请查询在<http://webreview.com/pub/guides/style/style.html>上的WebReview的浏览器兼容性列表。在本书编写时,到目前为止与CSS1最为一致的浏览器是Opera浏览器,一个相对于Microsoft和Netscape所把持的浏览器市场来说,它是非常小的角色。

XSL是一种比CSS更新的技术,现在还没有浏览器兼容性问题。当然,这是因为现在只有一种浏览器支持XSL:Internet Explorer 5.0。XSL来源于1997年8月Microsoft和ArbotText给W3C提交的一封信。ArbotText是一个网络工具开发商。W3C在1998年7月完成了第一份XSL工作草案。XSL经历了多个工作草案,而在本书编写时,它在W3C中依旧处于工作草案阶段。不久的将来将要成为推荐技术。

注意:技术上,Internet Explorer 5.0只支持XSL中的XSLT组件。

XSL的最初构想是XML所需的样式设计的一个解决方案。一般而言,你可以认为XSL和XML的关系与CSS和HTML的关系是一致的。这种比较并不完全正确,因为XSL有效的定义了一个当前CSS可用的样式函数集的扩展集,而XSL的文件转换能力又普遍的适用于XML。XSL基于DSSSL(文件样式语义和规范语言 Document Style Semantics and Specification Language)和CSS,因此它声称是对最大范围的样式表技术的融合。

注意:DSSSL是一种样式表规范,它被设计用来允许SGML文件格式化。虽然XSL很大程度上基于DSSSL,但它声称在很多方面对DSSSL进行了简化。XSL和DSSSL的关系与XML和SGML的关系是基本一致的。

8.2.1 理解CSS

CSS是一种样式表语言。它被设计为提供一种样式化HTML文件的手段,从而允许网络开发人员将内容从描述中分离出来。在CSS之前,样式化HTML文件的惟一办法是脚本语言和如动态HTML(DHTML)这样的混合型解决方案。CSS在学习和使用方面与这些应

用相比更加简单,这就使得它成为了样式化 HTML 文件的理想技术。虽然 CSS 是为 HTML 的使用而设计的,但它也可以用于样式化 XML 文件。

当一个 CSS 样式表被一个 XML 文件使用时,它将 XML 树结构作为应用设计规范的基础。虽然对于那些只需要样式化文件的现有结构的 XML 应用程序来说,它非常有用。但许多应用程序在 XML 文件被样式化和显示之前,需要 XML 内容以进行转换。也就是说,一些应用程序将需要比较、分类甚至重新排列文件数据,这对于 CSS 来说是不可能的。结果就是,CSS 更加适合于基于现有文件结构的 XML 文件的简单样式化。当然,你也可以使用 XSLT 来转换文件并用 CSS 来对它进行样式化。

注意:顽固的 CSS 鼓吹者指出你可以在使用 CSS 样式表之前使用脚本语言来转换文件和 DOM(Document Object Model, 文件对象模型)。DOM 在转换文件上可以基本上与使用 XSL 达到同样的效果。虽然 DOM 确实提供了转换 XML 文件的一种选择,但我们宁可使用结构化的转换语言也不依赖于定制脚本。

8.2.2 理解 XSL

就如你在本章前面所学到的,XSL 是一种样式表语言,它可以处理 XML 文件显示的两方面问题。

- 将文件从一种类型转换为另一种类型(XSLT)。
- 根据格式化规则样式化文件(XSL 格式化对象)。

XSLT 和 XSL 格式化对象是一种 XML 词表的实现。这些词表由相应的元素和属性组成,它们携带了文件每一部分的显示实现方法。使用这些词表使网络开发人员可以控制对文件内容的转换和随后的对转换后内容的显示。

由于所有的 XSL 元素以 XML 词表的方式实现,根据它们建立的样式表就是 XML 文件。这就允许你使用常见的 XML 语法来创建 XSL 样式表,而不必要求你使用 XML 开发工具。你可能看到了在 XSL 和另一种相当的 XML 新技术:XML 规范(XML Schema)的亲近关系。如果你回忆一下,XML 规范以 XML 词表的方式实现,它取代了早先 XML 的应用(DTD)来描述 XML 文件的结构。XSL 也是类似的,它大量的使用 XML 词表以最终取代早先的 XML 方法(CSS)来样式化 XML 文件。

当一个 XML 文件使用 XSL 进行转化时,转换使用了 XML 文件的逻辑树结构。这就意味着一个 XSL 转换的结果是一个 XML 数据树,而不是一个 XML 文件。我认为这是一个比较好的特性,这一点非常重要,因为 XSL 根据包含元素的树来进行工作,而不是根据如标记这样的 XML 语法构造。

8.3 XSL 与 CSS 的比较

虽然我已经描述了 XSL 和 CSS 作为样式表技术在一些方面上的不同,将它们进行一次彻底的比较还是十分值得的。更重要的是,我希望指出一些你为什么选择这种技术而不是另一种的理由。这两项技术非常的相近,很难决定什么时候选择这种或是那种。幸运的是,当前形势下浏览器对 CSS 和 XSL 的支持使得这个问题有了一个较为直接的答案。

在 XSL 和 CSS 之间有两点不同,这两点在前面的章节中我已经隐含的提到了:

- CSS 可被用于样式化 HTML 文件,而 XSL 则不能。
- XSL 可用来转换 XML 文件,而 CSS 则不能。

当然,本书主要是有关 XML 的,而不是 HTML 的。因此,第一点不同并没有太大的关系。然而,当你考虑到大多数的 XML 应用程序普遍的在某种程度上包含了 HTML 文件后,在你确定为给定的对象使用适当的样式表时,这一点就是一个问题。

第二个不同点非常紧要,因为 CSS 并没有提供用于转换 XML 文件的直接的方法(你可以使用带 DOM 的脚本语言来转换 XML 文件,但它自身就会带来其他问题)。然而,即使 CSS 不能被用来转换 XML 文件,它仍然可以用来样式化它们。

要记住 CSS 明确是为了非程序员使用而设计的,这也就解释了为什么它在学习和使用上是如此的容易。CSS 的明显缺点就是它实际上并不比一个格式化语言做的更多。它把样式属性放入到一个 XML/HTML 文件的元素中。同样的,CSS 有一些重要的限制:

- 它不能获得一块文件数据并在其他地方重新利用。
- 它没有节点间的同属关系的概念。
- 它不支持结果结构(有条件的)。
- 它不能计算数量或存储变量值。

如上面所指出的,XSL 显而易见是一个比 CSS 更为强大的技术,但是所附加的能力是以随之而来的复杂性为代价的。如果你并不介意费更多事,XSL 用于查询和重排列文件内容的功能当然也就没有意义了。在 XSL 中的特别的样式化工具是与 CSS 中的相关工具非常接近的,因此 XSL 的转换部分在它变得复杂时成为主要的优势。

此刻,在 XSL 和 CSS 之间的主要区别就是浏览器的支持。Internet Explorer 5.0 部分的支持 CSS1 并支持一个 XSLT 的私有版本。Netscape Navigator 5.0 在本书编写过程中还处于开发阶段,它将支持绝大部分的 CSS1 而几乎不支持 XSL。因此,为保证两个浏览器的使用者都可以浏览你的 XML 文件,看来 CSS 是解决的方式。虽然这两个浏览器对 CSS 的支持仍然存在着问题,但它的发展符合 W3C 的标准。一个 XSL 的试验性的(私有的)版本可以被 Internet Explorer 5.0 支持,但在现实世界的应用程序的基础上,它几乎什么都做不到。

注意:在一个网络服务器上使用 XSLT 将 XML 文件转换成 HTML 文件是可能的,它也减轻了浏览器对 XSL 的支持要求。你可以使用 CSS 来样式化作为结果的 HTML 文件。

8.4 XSL 和 CSS 联合使用

我已经提到了 XSL 和 CSS 并不是相互排斥的竞争对手,它们完全可以联合工作。实际上,当前情况下的浏览器要支持 XSL,几乎都需要一个包含两种技术的混合应用。下面有三种方式可以使 XSL 和 CSS 联合工作以提供一个混合的 XML 样式表解决方案:

- 在服务器端使用 XSL 将 XML 文件转换为由 CSS 样式表样式化的 HTML 文件。
- 在服务器端使用 XSL 将 XML 文件转换为由 CSS 样式表样式化的 XML 文件。

- 在客户端使用 XSL 将 XML 文件转换为由 CSS 样式表样式化的 HTML 文件。

可能你感觉到,这些列出来的使用 CSS 和 XSL 的应用是按照实现起来的困难程度排列的。这样看来,第一种方式可以提供最佳的 XSL 和 CSS 的结合,因为它可以不需要考虑在网络浏览器部分的任何特殊性,所有浏览器所看到的是由服务器产生的正规 HTML 文件。

第二种方式有利之处就在于它在浏览器中直接包含样式化 XML 的代码。这就意味着并不需要将 XML 文件转换为 HTML 文件。惟一的转换就是将一种 XML 文件转换为另一种格式的更适用于描述的 XML 文件。这种方法需要浏览器支持使用 CSS 直接浏览 XML 文件。虽然这种支持在 Internet Explorer 5.0 中已经实现,但它并不完善。

第三种方式有趣之处在于 XML 文件是在浏览器中转换为 HTML 文件并直接样式化的。然而,浏览器对这种功能的支持还没有实现。Internet Explorer 5.0 试验性的支持 XSLT 是一个很好的尝试,但目前为止,它还不足以用在基于现实世界的 XML 应用程序开发上。

8.5 总 结

由于 HTML 如此迅速的被采用为网络的出版格式,它正在被用在本身设计并没有考虑到的很多方式下。不用多久 HTML 就变成了一个内容标记和描述标记混在一起的糟糕的混合体,它变成了一个非常糟糕的语言。值得承认的是,有一段时间,人们努力通过多种可能的途径来扩展 HTML 以建立更好看的网络页面。但最终的结果并不美好。样式表提供了翻新 HTML 并解决内容/描述混乱的机制。

在这一章中介绍了样式表的基础并引导你学习了适用于 XML 的两种样式表技术: XSL 和 CSS。虽然 XSL 和 CSS 解决的是同样的问题,但它们是不同的技术。在 XML 组织中对于使用哪个样式表更好有着大量的争论。我的意见是这并没有什么关系。两项技术各自适用于适当的条件。本人认为,不久的将来一个合适的特定样式化 XML 文件的技术将是 XSL 和 CSS 的混合体,这也就打消了对于从这两个方法中选择其一的争论。

第9章 使用层叠样式表格式化 XML

CSS(Cascading Style Sheets,层叠样式表)是一种样式表语言。它支持对 HTML/XML 文件为在网上显示而进行的格式化。CSS 最初是为了将 HTML 文件中的内容与描述相区分而创建的。因为 XML 是一种纯粹的基于内容的解析型语言,CSS 在格式化 XML 文件方面已经被证明确实非常的有效。CSS 非常容易学习与使用,它还提供了一种快速、直接的方法来描述 XML 内容以进行浏览。CSS 的相对简化也正是它的缺点所在,这是因为你控制 XML 文件格式化的能力有限。

本章将向你介绍 CSS 及其涉及 XML 的部分。CSS 是一种全面的网络技术,这种技术的很多部分将超出本书的涉及范围。然而,我将为你尽量讲到 CSS 各部分,以便即使没有任何 CSS 经验的情况下,你也可以理解它是如何协同 XML 工作的。如果你已经有了 CSS 的相关经验,你就会发现在 XML 上使用 CSS 与在 HTML 上使用 CSS 非常的相近。

9.1 CSS 初步

虽然 CSS 样式表机制最初是为 HTML 而设计的。它在为实现描述而进行的格式化 XML 文件这方面的功能也同样做得很出色。作为一种描述性样式语言,CSS 定义了如字体、颜色和位置等样式结构。这些样式结构可以被用来说明基于内容的数据的描述部分,例如 XML 文件中的数据。CSS 中的一个重要的思想就是你可以为给出的一类文件只创建一个样式表,然后你就可以使用这个样式表显示任何属于这类的文件。这种方法允许你将一组文件的描述特征分离出来并放在一个单独的位置。这就使得维护文件的工作变得更加容易(起码从描述的观点来说)。

一个 CSS 样式表包括一组规则,这些规则应用到给定类型的元素上。样式表规则可以直接放置在一个 HTML 或是 XML 文件中,它们也可以被放置在扩展名为.css 的外部样式表文件中。我更喜欢第二种方法,因为它对文件的内容和描述部分做了清楚的区分。当然,你必须在 HTML 或者 XML 文件中声明一个外部样式表文件,这样才可以使用它作为格式化文件内容的基础。

刚刚接触 CSS 的网络开发人员经常对名称的“cascading”感到迷惑不解。Cascading 指的是样式表规则互相比权重并被元素应用的方法。这些权重是非常必要的,因为样式可以被多种不同的方法指定,这样就使得一个单一元素类型拥有多种样式规则成为了可能。如何确定哪个规则优先并被元素应用是 CSS 的责任。这就组成了元素应用规则的层次,产生层叠的效果。

在某些方面,CSS 的层叠机制与面向对象编程语言中的继承类似——基本的样式规则被样式表继承,但样式表可以被更多的特殊样式规则重写。

9.2 深入 CSS 样式表

如果你对 CSS 没有任何经验,你将会高兴的了解它在结构上是如此的简单。一个 CSS 样式表由定义如何在文件中应用样式的规则组成。样式规则的运用被定义为选择器,它是一个识别 HTML 或 XML 文件部分的 CSS 结构。选择器设置 HTML 或 XML 文件与一个或一组样式之间的链接。在 CSS 中有三种选择器可供使用:

- 元素类型
- 属性类
- 属性 ID

元素类型选择器通过所给类型选择一个元素,并为它应用一个或一组样式。这是使用 CSS 最简单的方法,因为这只是在元素类型和样式集间进行一一映射。例如,你可以使用元素类型选择器定义一组样式,这些样式为一个 HTML 文件中的 p 元素指定边界和字体效果。所有使用这个元素标记的段落将要遵样式规则中 p 元素类型选择器定义的样式进行显示。下面是一个这样的样式规则的例子:

```
p{
    display: block;
    margin-bottom: 10px;
    font-family: Times, serif;
    font-size: 12pt;
}
```

这个例子使用元素类型选择器选择 p 元素并给它应用一系列的样式。最终结果就是文件中所有的类型 p 的段落元素都将有 10 像素的边距并将显示为 12 磅的 Times 字体。下面是在一个 HTML 文件中如何标记这样的段落代码的例子:

```
<p>
this is a paragraph of text.
</p>
```

属性类选择器使用起来更加灵活,它允许你依据一个专门的属性给元素应用属性。另外, CSS 使用属性选择器来寻找包含有某一个值的特定属性的特定元素。这种属性被称为类(class),实质上它可以有你所期望的任何值。你使用类属性来定义一个给定元素类型的样式类。下面是一个说明如何定义一个重点格式化段落的特殊类的例子:

```
p. excite{
    display: block;
    margin-bottom: 10px;
    color: green;
    font-family: Times, serif;
    font-size: 12pt;
    font-style: italic;
}
```

这个例子中的类名是 excite,它通过句点(.)与元素类型区分开。下面是一个如何标记一

个使用该样式规则的相应段落的例子：

```
<p class="excite">
this is a paragraph of text.
</p>
```

如你所见,excite 类名被作为<p>标记的类属性的值。这是在 HTML 文件中使用 CSS 属性类选择器的标准方法。

为了在最大程度上实现样式灵活性,你可以使用属性 ID 选择器。这个选择器设置一个可以适用于任何元素的样式规则,而不用管它的类型。与属性类选择器类似,属性 ID 选择器依赖特定的属性。虽然如此,但属性是被 ID 命名并且不与任何常见的元素类型关联。下面是创建一个使用属性 ID 选择器的样式类型的例子：

```
#feelingblue {
    color: blue;
    font-family: Arial;
}
```

这个例子用一个名为 feelingblue 的属性 ID 选择器创建了一个样式规则。井号(#)用来指出这是一个属性 ID 选择器。为了使用这个样式,你只要在任何元素类型中将它引用为一个 id 属性的值就可以了。下面是一个告诉你如何在一个 HTML 头元素中使用这个样式的例子：

```
<h1 id="feelingblue"> My Blues Collection</h1>
```

虽然选择器在如何将样式应用于文件方面扮演着重要的角色,它们也只是样式表语法中的一部分。另一个重要部分就是样式声明。它用来指定一个样式属性及其相应的值。CSS 中的样式声明与 HTML 和 XML 中的属性类似。它们都是给属性赋值。CSS 支持大量的样式属性,它们可以用来定制一个给定样式规则的样式。

与选择器结合使用,样式声明组成了 CSS 样式规则的语法,结构如下：

```
Selector {
    Property1: value1;
    Property2: value2;
}
```

下面这个样式规则的示例清楚的说明了 CSS 样式规则语法的使用方法：

```
name {
    display: block;
    font-family: Times, serif;
    font-size: 16pt;
    font-weight: bold;
}
```

这个样式规则的例子用于类型为 name 的 XML 元素,这个类型用在 XML 文件的地址簿中显示一个人的姓名。这里共定义了四个样式属性:display、font-family、font-size 和 font-weight。每个属性都有一个用于设置 name 元素样式的相关的值。同样,注意每一个样式属性/值对要通过分号(;)隔开。

如果你有了一组公用的样式声明要用于多个元素,只要在一个样式规则中简单的用逗号(,)将元素类型区分开就可以了。就像这样:

```
phone,email,web,company{  
    display:none;  
}
```

在这个例子中,值 none 被赋给 phone、email、web 和 company 这些元素的样式特征 display。当包含这些元素的 XML 文件显示的时候,这些元素都不会显示出来。

9.3 CSS 样式属性

迄今为止,你已经看到了一些 CSS 样式表的例子。它们使用了多种样式特征来设置 HTML/XML 内容特有的格式化方式。这些特征中的一部分相当简单明了,我基本上可以不用解释它们的使用方法。回忆一下以前的章节可知,现在有两种不同的 CSS 规范:CSS1 和 CSS2。虽然 CSS2 与 CSS1 相比加入了重要的功能,但 CSS1 支持大多数的用于格式化 HTML 和 XML 的样式特征。由于在本书编写时,CSS1 是浏览器支持的惟一 CSS 规范。我希望能够尽量涉及 CSS1 中定义到的样式特征。

CSS1 定义了大量的样式特征,用来控制字体、颜色、对齐和边距等等。下面是一些 CSS1 中支持的最常见的样式特征:

- | | |
|-----------------|--------------------|
| • display | • margin-left |
| • width | • margin-right |
| • height | • margin-top |
| • border | • margin-bottom |
| • border-width | • background-color |
| • border-color | • color |
| • border-style | • text-align |
| • border-left | • text-indent |
| • border-right | • font-family |
| • border-top | • font-size |
| • border-bottom | • font-style |
| • margin | • font-weight |

在下面的几小节将更加详细的讲解这些样式特征。要谨记这些只是 CSS1 支持的样式特征的一小部分。要了解有关 CSS1 特征的更多信息,请浏览位于 <http://www.w3.org/TR/REC-CSS1> 中的 CSS1 W3C 介绍。

9.3.1 display 属性

display 特征描述了如何显示一个元素,以及该元素是否显示。display 特征有 4 个可取值:

- block——元素在一个区域中显示。

- list-item——元素在一个区域中显示并在其后添加一个列表元素标记(着重号)。
- inline——元素在一个与相邻内容在同一行的内嵌区域中显示。
- none——元素不显示。

display 特征说明元素是在矩形区域中显示的。这些区域起初的目的是用于定位,不必可见。display 特征是你需要在每个样式规则中定义的少数几个特征之一。下面是一个如何设置 display 特征的例子:

```
display: block;
```

9.3.2 width 和 height 属性

width 和 height 特征定义了元素显示所在区域的宽度和高度,或者是定义一幅图片的具体尺寸。和许多与尺寸相关的 CSS 特征类似,width 和 height 特征值可以按照不同的尺寸单位进行设置。下面是 CSS1 中支持的尺寸单位:

- | | |
|-----------------------|--------------------|
| • Inches(英寸)——in | • Points(磅)——pt |
| • Centimeters(厘米)——cm | • Picas(12 点制)——pi |
| • Millimeters(毫米)——mm | • Ems(英里)——em |
| • Pixels(像素)——px | • Ens——en |

虽然你可以混合和匹配你选择的尺寸单位,在类似的样式特征中尽量保持一致的度量单位通常是一个很好的想法。例如,你可能希望在字体特征中使用磅而在区域定制上使用像素。下面是一个使用像素单位设置一个区域宽度的例子:

```
width: 350px;
```

9.3.3 border 属性

border 特征提供了一种描述元素边界的方法。如果四个边界的设置相同,你可以使用 border 特征将它们一次全部描述。另外,你也可以利用 border-left、border-right、border-top 和 border-bottom 对它们进行分别的描述。下面是一个使用 border 特征设置边界的示例。这个边界由两条宽度为 4 个像素的黑线组成:

```
border: 4px double black;
```

这个例子说明了如何使用单一的特征来描述边界。你也可以使用单一特征 border-width、border-color 和 border-style 来分别描述边界样式。

9.3.4 margin 属性

margin 特征用来描述一个元素的边距。如果四面的边距是同样的,你可以使用 margin 特征对它们一次完成描述。另外,你可以使用 margin-left、margin-right、margin-top 和 margin-bottom 对各个边距分别进行描述。下面是使用 margin-bottom 特征定制一个 10 个像素宽度的边距的例子:

```
Margin-bottom: 10px;
```

9.3.5 color 属性

background-color 和 color 特征分别用来设置元素的背景颜色和元素内容的字体颜色。下面是一个使用预定义颜色设置这两个特征的例子：

```
background-color:silver;  
color:blue;
```

你也可以给这些特征通过指定十六进制颜色值或 RGB 颜色值来进行赋值。下面是这种赋值的一个例子：

```
background-color:#999999;  
color:rgb(0,0,255);
```

9.3.6 text 属性

text-align 和 text-indent 特征分别用来设置一个元素的对齐和缩进。下面是设置这些特征的一个实例：

```
text-align:center;  
text-indent:12px;
```

9.3.7 font 属性

font 特征用来设置与字体相关的多个参数。font-family 特征指定一个 font-family 的优先顺序列表。除非只有一个值，一个优先顺序列表可以在所给系统没有相应字体的情况下提供其他的选择。font-size 特征使用前面所提到的某一种长度单位（通常是磅）来指定字体尺寸。font-style 特征设置字体样式，它可以是下面的值：normal、italic 或是 oblique。font-weight 特征设定字体粗细，它可以是下面这些值之一：normal、bold、bolder 或 lighter。下面是一个设置 font 特征的例子：

```
font-famil:Helvetica, sams-serif;  
font-size:24pt;  
font-style: italic;  
font-weight: bold;
```

9.4 创建 CSS 样式表

在你用样式规则组成一个 CSS 样式表之前，你一定要确定你希望如何将样式表与 XML 文件关联。下面有两种方法将 CSS 样式表与文件相关联：

- 外部样式表
- 文件级样式表

虽然 XML 支持这两种引用样式表的方式，我仍建议使用第一种方法，因为它提供了对文件内容和描述的最清晰的区分。它也有助于将一个样式表用于多个文件当中，这也是使用样式表的首要好处之一。文件级样式表也不坏——它们可以在你开发一个单个的 XML 文

件方面节省你的时间。然而,从最合适的角度来说,我推荐你创建外部样式表。

一个外部样式表是一个扩展名为.css 的无格式文本文件。它存储着一组基本的 CSS 样式规则。这个文件将会被要使用样式表进行显示的 XML 文件引用。这种引用是通过特定的 XML 过程指令 xml-stylesheet 实现的。它本身定义了样式表文件和样式表类型。下面是一个使用 xml-stylesheet 过程命令将一个 CSS 样式表与 XML 文件相关联的示例:

```
<? xml-stylesheet type="text/css" href="AddressBook.css"? >
```

在这个例子中,type 属性指定样式表的类型是 text/css,或者说是一个 CSS 样式表。样式表文件在 href 属性中被引用,在这里我们引用了 AddressBook.css 文件。这个文件听起来很熟悉。实际上,这个地址簿 XML 文件在第 1 章“XML 新手”和第 2 章“数据模型:文件类型定义(DTD)和 Schema”中已经使用过了。清单 9.1 列出了 AddressBook.xml 文件的代码。这是创建一个简单的 CSS 样式表的基础。

清单 9.1 地址簿 XML 文件(AddressBook.xml)

```
<? xml version="1.0"? >
<? xml-stylesheet type="text/css" href="AddressBook.css"? >
<! DOCTYPE addressbook SYSTEM "AddressBook.dtd" [
<! ENTITY amp "&#38;#38;" >
<! ENTITY apos "&#39;" >
]>
<addressbook>
  <!-- This is my good friend Frank. -->
  <contact>
    <name>Frank Rizzo</name>
    <address>1212 W 304th Street</address>
    <city>New York</city>
    <state>New York</state>
    <zip>10011</zip>
    <phone>
      <voice>212-555-1212</voice>
      <fax>212-555-1213</fax>
    </phone>
    <email>frizzo@fruity.com</email>
    <web>http://www.fruity.com/rizzo</web>
    <company>Frank&apos;s Ratchet Service</company>
  </contact>
  <!-- This is my old college roommate Sol. -->
  <contact>
    <name>Sol Rosenberg</name>
    <address>1162 E 412th Street</address>
    <city>New York</city>
    <state>New York</state>
    <zip>10011</zip>
    <phone>
      <voice>212-555-1818</voice>
      <fax>212-555-1819</fax>
    </phone>
```



```
<email>srosenberg@fruity.com</email>
<web>http://www.fruity.com/rosenberg</web>
<company>Rosenberg's Shoes & Glasses</company>
</contact>
</addressbook>
```

在这个文件中,要知道的最重要的部分就是 xml-styleSheet 过程命令,它引用了 AddressBook.css 样式表。我们在清单 9.1 中列出这个样式表。

清单 9.2 地址簿 CSS 样式表(AddressBook.css)

```
contact {
    display: block;
    width: 350px;
    padding: 10px;
    margin-bottom: 10px;
    border: 4px double black;
    background-color: silver;
    color: blue;
    text-align: center;
}

name {
    display: block;
    font-family: Times, serif;
    font-size: 16pt;
    font-weight: bold;
}

address {
    display: block;
    font-family: Times, serif;
    font-size: 14pt;
}

city, state, zip {
    display: inline;
    font-family: Times, serif;
    font-size: 14pt;
}

phone, email, web, company {
    display: none;
}
```

这个样式表说明了为一个现有的 XML 词表建立样式规则是多么的简单。要承认,地址簿的词表非常的简单,而事实上也是如此。那就是创建一个样式表并不比为词表中的每一个元素创建样式规则更困难。在这个样式表中最复杂的样式规则就是 contact。它描述了 contact 元素的样式。在这个 XML 文件中,contact 元素实际上并不包含任何的文字内容。也就是说在 contact 样式规则中的样式特征是用于 contact 的子元素的。当然,很多的子元素重写了这些特征,这也就说明了 CSS 的层叠特性。

大多数的子元素在使用样式表特征方面简单而又直接。然而,值得指出的是 phone、email、web 和 company 元素,它们将 display 特征设置为 none。因此,这些元素在 XML 文件使用这个样式表显示的时候是隐藏的。图 9.1 显示了文件在 Internet Explorer 5.0 中的显示情况。

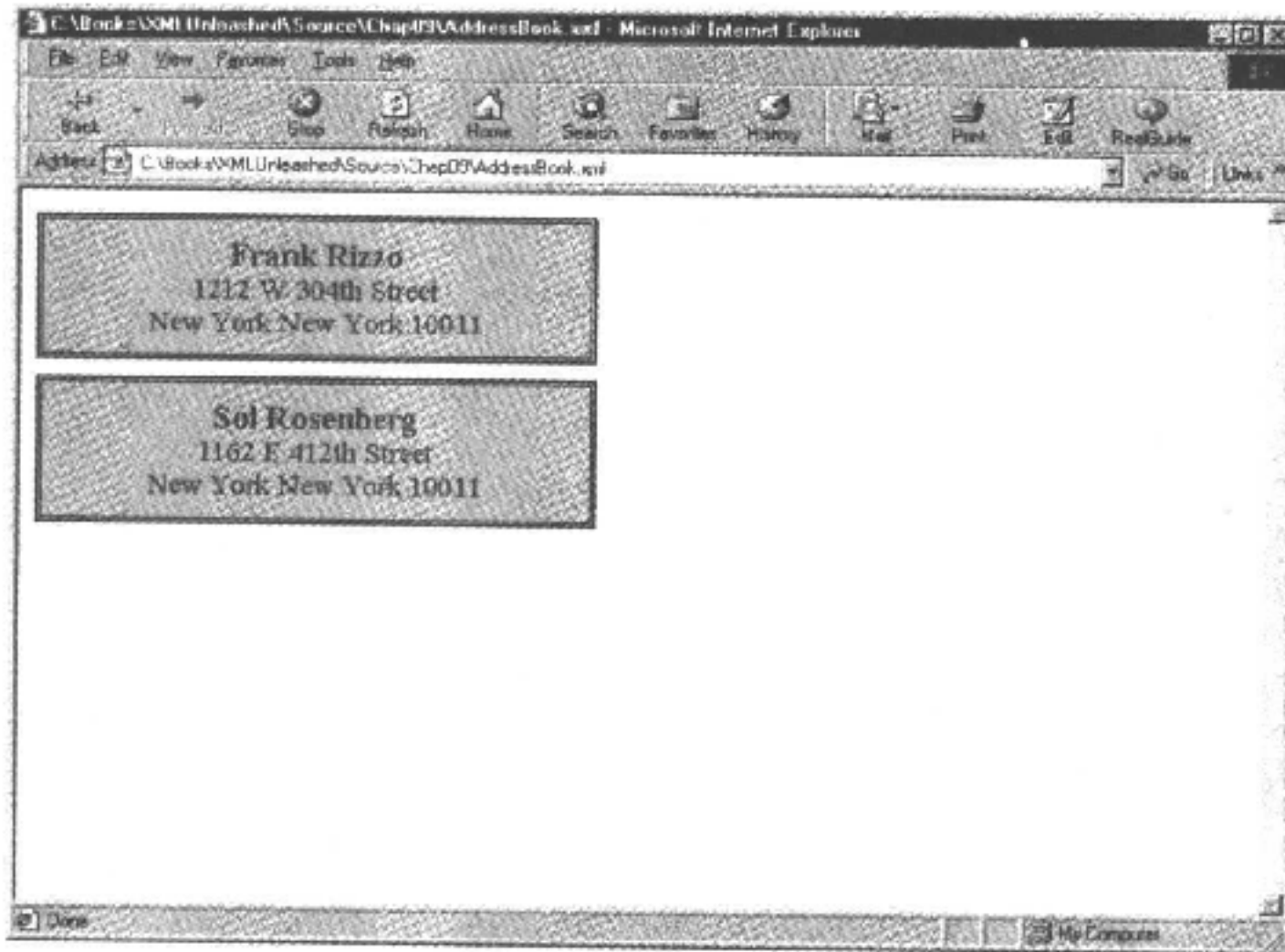


图 9.1 使用 CSS 样式表的地址簿 XML 文件在 Internet Explorer 5.0 中的显示情况

CSS 最大的局限性就是除非使用如 XSL 或一个脚本语言之类的其他技术,否则在显示样式化后的 XML 内容中不能加入任何附加信息。例如,提供一个文本标头来描述通过样式表转换显示的 XML 内容将是非常有用的。在有这种文本标头的场所,你可以使用样式表中的颜色来协助与数据类型相区分。清单 9.3 列出了 Movies.xml 文件(已在第 3 章“DTD 基础”和第 4 章“进一步深入 DTD”中进行讲解),这是一个使用颜色样式化文件的好例子。

清单 9.3 电影 XML 文件(Movies.xml)

```
<? xml version="1.0" standalone="no" ? >
<? xml-stylesheet type="text/css" href="Movies.css" ? >
<! DOCTYPE movies SYSTEM "Movies.dtd" >
<movies>
  <movie type="comedy" rating="PG-13" review="5" year="1987">
    <title>Raising Arizona</title>
    <writer>Ethan Coen</writer>
    <writer>Joel Coen</writer>
    <producer>Ethan Coen</producer>
    <director>Joel Coen</director>
    <actor>Nicolas Cage</actor>
    <actor>Holly Hunter</actor>
    <actor>John Goodman</actor>
    <comments>A classic one-of-a-kind screwball love story.</comments>
```

```

</movie>
<movie type="comedy" rating="R" review="5" year="1988">
  <title>Midnight Run</title>
  <writer>George Gallo</writer>
  <producer>Martin Brest</producer>
  <director>Martin Brest</director>
  <actor>Robert De Niro</actor>
  <actor>Charles Grodin</actor>
  <comments>The quintessential road comedy. </comments>
</movie>
<movie type="mystery" rating="R" review="5" year="1995">
  <title>The Usual Suspects</title>
  <writer>Christopher McQuarrie</writer>
  <producer>Bryan Singer</producer>
  <producer>Michael McDonnell</producer>
  <director>Bryan Singer</director>
  <actor>Stephen Baldwin</actor>
  <actor>Gabriel Byrne</actor>
  <actor>Benicio Del Toro</actor>
  <actor>Chazz Palminteri</actor>
  <actor>Kevin Pollak</actor>
  <actor>Kevin Spacey</actor>
  <comments>A crime mystery with incredibly intricate plot twists. </comments>
</movie>
<movie type="sci-fi" rating="PG-13" review="4" year="1989">
  <title>The Abyss</title>
  <writer>James Cameron</writer>
  <producer>Gale Anne Hurd</producer>
  <director>James Cameron</director>
  <actor>Ed Harris</actor>
  <actor>Mary Elizabeth Mastrantonio</actor>
  <comments>A very engaging underwater odyssey. </comments>
</movie>
</movies>

```

这个文件的问题是它不能显示每个人的头衔——作者、导演、演员等等。另一种方法就是用不同的颜色和字体显示每一个名字。要承认,这种方法并不完美,但它给你一个机会来试验 CSS 中的颜色和字体。清单 9.4 列出了 Movies.css 样式表清单。这个样式表使用颜色和字体来区分电影中的不同成员。

清单 9.4 电影 CSS 样式表 (Movies.css)

```

movie {
  display: block;
  margin-bottom: 10px;
  background-color: white;
  text-align: left;
}

title {

```

```
display: block;
color: black;
font-family: Helvetica, sans-serif;
font-size: 24pt;
font-weight: bold;
}

writer {
display: block;
color: blue;
font-family: Courier, monospace;
font-size: 14pt;
text-indent: 12px;
}

producer {
display: block;
color: red;
font-family: Western, fantasy;
font-size: 12pt;
text-indent: 12px;
}

director {
display: block;
color: green;
font-family: Times, serif;
font-size: 14pt;
text-indent: 12px;
}

actor {
display: block;
color: fuchsia;
font-family: Zapf-Chancery, cursive;
font-size: 20pt;
text-indent: 12px;
}

comments {
display: none;
}
```

如你所见,这是一个比较简单的样式表。其中大多数的样式规则非常简单,只有颜色和字体除外。图 9.2 给出了在显示 Movies.xml 文件时这个样式表的效果(可惜的是,你只能想像一下这幅图中的颜色)。

前述的两个 CSS 的例子都比较简单,它使用有较少文字的 XML 文件。当你使用它来样式化更多的文字的时候,CSS 的真正价值就开始显示出来。在第 31 章“用 XMLNEWS 制作新闻”中,你将学习到一个称为 XMLNews 的 XML 词表,它是用来标记新闻故事的。清单 9.5 显示了一个 XMLNews 文件的示例,这个示例将在第 31 章中详细阐述。至于放在这一章的目的,我是希望示范一下如何使用一个 CSS 样式表显示这个文件。

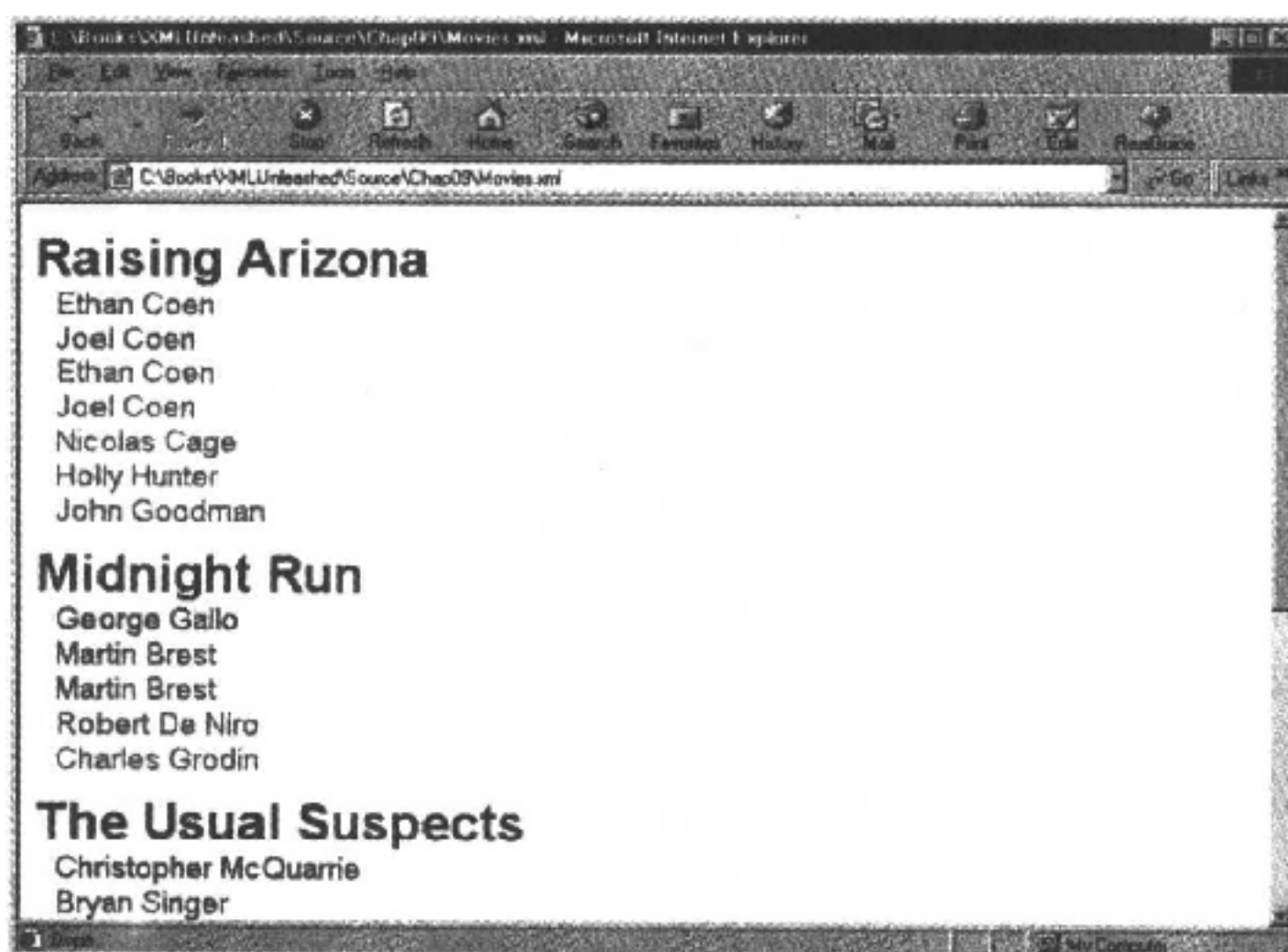


图 9.2 使用 CSS 样式表的电影 XML 文件在 Internet Explorer 5.0 中的显示情况

清单 9.5 新闻 XML 文件 (News.xml)

```

<? xml version="1.0" ? >
<? xml-stylesheet type="text/css" href="News.css" ? >
<nitf>
  <head>
    <title>Nerd Author Strikes XML Gold</title>
  </head>
  <body>
    <body.head>
      <headline>
        <h1>Nerd Author Strikes XML Gold</h1>
      </headline>
      <byline>
        <bytag>By Josh Timm</bytag>
      </byline>
      <dateline>
        <location>Nashville, Tennessee</location>
        <story.date>Monday March 20 2000 9:43 ET</story.date>
      </body.head>
    <body.content>
      <p>Acclaimed nerd author Michael Morrison strikes gold with his immensely popular book XML Unleashed. At approximately 8:55 ET last night, the one millionth copy of XML Unleashed was sold at a Barnes and Noble bookstore in suburban Nashville, Tennessee, which makes the book the best selling computer book of all time. The book explores the ins and outs of the XML information technology that has quickly thaken the Web by storm. It's not really apparent
    </p>
  </body.content>
</nitf>

```

```

whether the book's success can be attributed to Morrison's genius
and command over the XML technology, or just dumb luck. </p>
<p>When asked about the sudden success of his book, Morrison
replied <q>I'm just a regular guy trying to state the facts. </q>
Regular or not, Morrison will now be spending most of his time
at the bank depositing royalty checks. When asked what he plans
to do with his newfound riches, Morrison stated I don't plan on
changing my lifestyle much, except for buying a new house, a half
dozen cars, a fat diamond for my wife, and constructing my own
extreme sports park on the 1000 acre farm that I'm currently in
negotiations to buy. </q></p>
</body.content>
</body>
</niff>

```

虽然你可能对 XMLNews 这个词表一无所知,但理解这个文件的基本结构并不困难。这个故事的标题使用 headline 元素保存,作者信息在 byline 元素中保存,而日期和地点信息在 dateline 元素中保存。此外,p 元素用来保存新闻故事的实际文本。这些正是创建这个文件的样式表所需要的所有信息。News.css 样式表如清单 9.6 所示。

清单 9.6 新闻 CSS 样式表(News.css)

```

headline {
  display: block;
  width: 400px;
  border-bottom: 5px double black;
  text-align: right;
  color: olive;
  font-family: Times, serif;
  font-size: 36pt;
}

byline {
  display: inline;
  width: 200px;
  text-align: left;
  color: black;
  font-family: Times, serif;
  font-size: 14pt;
}

dateline {
  display: inline;
  width: 200px;
  text-align: right;
  color: black;
  font-family: Times, serif;
  font-size: 11pt;
  font-style: italic;
}

p {

```

```
display: block;  
width: 400px;  
color: black;  
font-family: Times, serif;  
font-size: 12pt;  
}  
  
title {  
display: none;  
}
```

考虑到复杂的 XML 新闻故事文件,这个样式表可能比你想像的要简单。这也正说明了 CSS 的结构是如此的简单,它可以不用费很大的功夫就创建一个用于复杂词表的样式表。News.css 样式表努力将新闻故事文件以与报纸上显示的同样样式显示。图 9.3 说明了使用样式表的新闻故事文件在 Internet Explorer 5.0 中的显示情况。

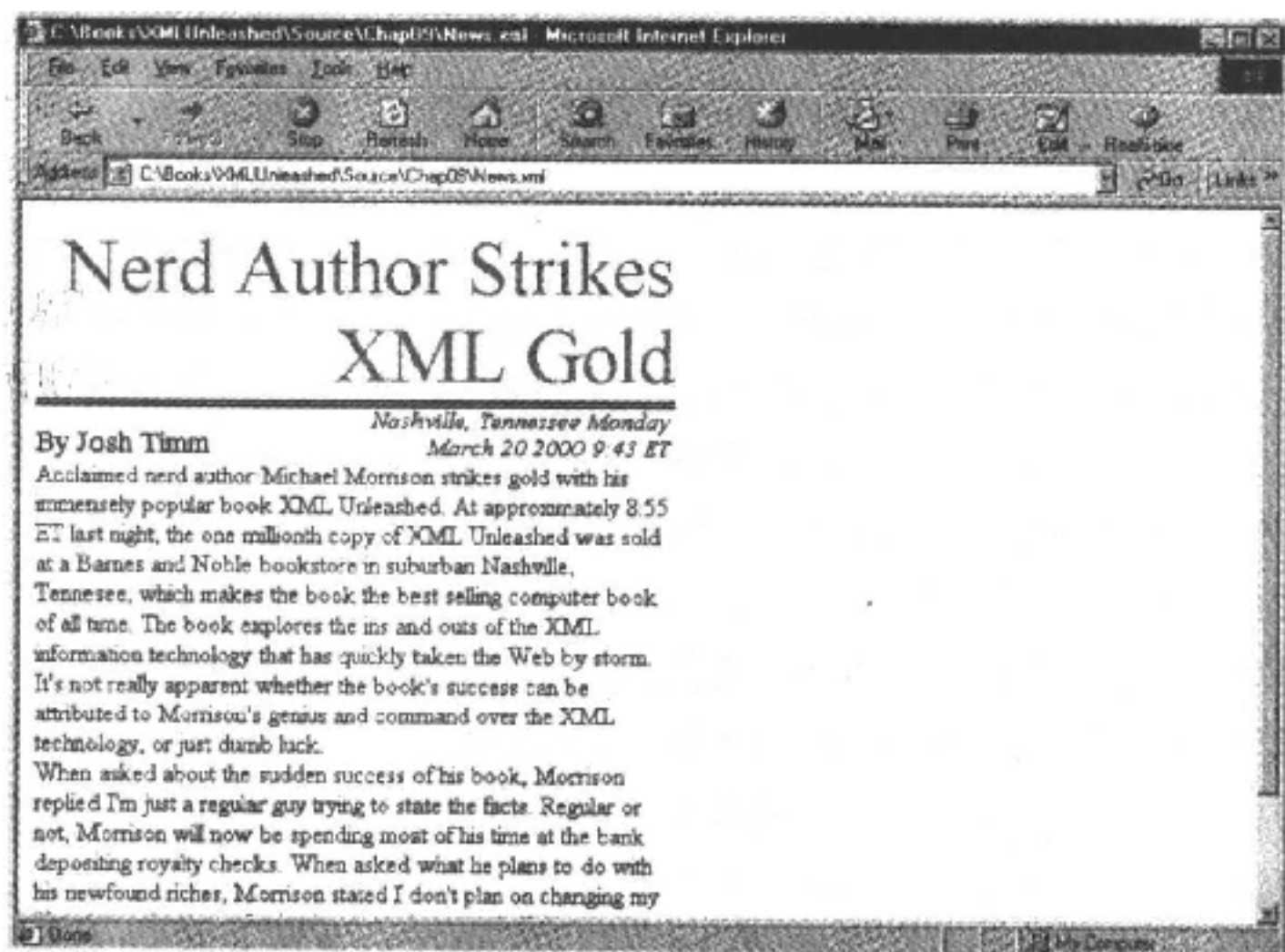


图 9.3 使用 CSS 样式表的新闻 XML 文件在 Internet Explorer 5.0 中的显示情况

9.5 总 结

CSS 是一个为用于 HTML 而设计的样式表技术,但它也恰好非常适用于 XML。CSS 样式表定义样式规则,这些规则被 XML 文件中的元素用来确定这些元素如何在网络浏览器上显示。如果你所需要做的只是使用描述样式来格式化 XML 文件,CSS 是完美的解决方案。然而,许多的 XML 应用程序需要在它们使用样式之前用过滤、顺序化、排列或其他操作来处理 XML 内容。既然如此,CSS 就需要其他技术的帮助。比如 XSL 或是一个可以使用 DOM 的脚本语言。

第10章 理解XSL

XSL是一个特别为XML而设计的样式表技术。在许多方面,XSL实现了CSS所能做到的同样的事情。而XSL在处理XML文件的逻辑结构方面比CSS做的更多。这个附加的能力是有代价的,XSL包含了一些高性能的编程结构,这同时就带来了复杂性。除了它比CSS更难掌握以外,XSL通常因为过于新而遭到非议。

本章将从概念上探讨XSL,并尽量勾画出XSL完善XML的执行流程。一些主要的XSL概念将在第8章“对比两种样式规范:可扩展样式表语言和层叠样式表”中进行讲解,而本章则更为深入。如果你对于理解XSL的执行流程并不感兴趣,那就跳过去直接学习下一章。深刻的理解XSL的任务是值得的,因为在将来它会继续发展并成为W3C推荐技术。

10.1 处理XSL样式表

为了全面的理解XSL,你一定要确实的掌握一个XSL处理机是怎样处理XML文件的。在XSL处理机处理文件之前需要两个东西:文件的XML树式描述和XSL样式表。文件的XML树式描述通过解析文件得到,也就是说XSL处理机要能够执行就必须要与XSL解析器结合使用。这同样也就是说,分析XML文件并不是XML处理机的工作。如果一个文件成功的分析为一个树形结构,XSL处理机就可以开始工作了。

XSL处理机从树的根节点开始,用它在样式表中进行模式匹配。一个XSL样式表由模板组成,这些模板使用模式确定对XML文件的哪一部分进行格式化。XSL处理机分析这些模板和与它们相关的模式来处理树的不同部分。当出现了匹配,树匹配的部分传递给样式表中的模板进行处理。XSL处理机按照模板规则来产生一棵结果树这一点非常重要,就是它把一棵树作为输入并同时产生另一棵树作为输出。

由XSL处理机产生的结果树将包含XSL格式化对象,这些对象用来格式化结果树以进行显示。虽然格式化对象声称它们是用来格式化XML数据以进行显示的XSL方式,但并不要求你使用它们。实际上,由于XSL格式化对象工作草案到现在还没有被推荐,不要求你使用它们是一件好事。选择使用格式化对象来样式化结果树的一个有用而独特的情况就是使用HTML代码。实际上,你不能使用传统的HTML代码,因为能够通过样式表产生的结果树必须是XML代码。然而,更高层结构的HTML(XHTML)代码可以使用XSL样式表产生用于格式化对象的结果树。

注意:你不能使用传统的HTML代码作为一个XSL结果树的生成基础,因为传统的HTML不是XML的词表。然而,一个称为XHTML的新版本的HTML被开发了出来,它在HTML中加入了更多的结构使它成为了XML词表。

XSL处理机继续处理XML树中的每一个节点,在处理过程中使用在样式表中定义的所有模板。当所有的节点都经过了处理,所有的样式表模板都被使用后,XSL处理机返回一

个完成的结果树。它通常是一个适用于显示的格式(我说“通常”是因为使用 XSL 将一个文件在一种 XML 词表和另一种 XML 词表间进行转换也是可能的。这样也就并不需要文件必须适用于显示)。

10.2 XSL 体系结构

W3C 在设计 XSL 体系结构方面已经做了大量的工作,因此它尽可能的具有灵活性和可扩张性。也许 XSL 可能会发展成一项独立的技术,它包含了多种技术,而其中的一部分在 XSL 以外也是十分有用的。不止是 XSL,还包括整个的 XML,这些技术是它们的重要组成部分。为了理解这些技术的相关性,我们有必要再一次体验 XSL 处理机的作用。

XSL 处理机扮演着两个角色:

- 从资源树构造一棵结果树。
- 为格式化而解释结果树。

XSL 处理机的第一个角色是将一棵资源树转换成一棵结果树,这被称为树转换。基本上,这是将 XML 内容从一种词表转换成另一种词表的过程。XSL 处理机的第二个角色包括检查结果树的格式化信息并依次格式化每一个节点的内容。

虽然将 XSL 处理过程分成两个任务似乎比较方便,但这样做还有一个比仅仅是方便更重要的原因。考虑一下 CSS,它只支持格式化 XML 内容。当你发现资源文件并不能用任何方式修改时,CSS 的限制就变得非常明显。另一方面,使用 XSL,在转换文件过程期间,你可以完全自由的任意揣摩资源文件。这种两步方案——根据格式化需求转换——在显示 XML 文件上提供了令人难以置信的灵活度。

XML 处理机所具有的两个基本任务直接导致了两项 XSL 技术:

- XSL 转换(XSLT)
- XSL 格式化对象

这两项技术都是以 XML 词表的方式实现的,这就使它们的语法非常相近。这也就是说,它们所创建的样式表就是 XML 文件。XSL 的这两个组件的有趣之处就是它们可以分别使用。你可以使用 XSLT 来转换文件而不必考虑文件是怎么格式化的。同样,你可以使用 XSL 格式化对象来格式化 XML 文件而不必要对它们进行任何转换。

注意:在本书编写期间,XSLT 已经基本成为 W3C 的推荐技术,而 XSL 格式化对象尚处于开发的初期阶段。

要谨记的另一个重要的方面就是 XSL 实际上是两个语言。XSLT 是 XSL 转换语言,它用于将 XML 文件从一种词表转换为另一种。XSL 格式化对象是 XSL 格式化语言,它用来以显示为目的将格式化样式应用于 XML 文件。迄今为止,大多数开发人员主要关注 XSLT,而我非常希望 XSL 格式化对象最终能作为 XML 文件的一种可选择的格式化语言。我说“最终”是因为到现在为止,我们可能只执著于使用 XSLT 和 CSS 从 XML 文件中产生 HTML 文件并传给网络浏览器。

10.2.1 XSL 转换

XSL 转换(XSLT)是 XSL 样式表技术中的转换部分。XSLT 由一个 XML 词表组成,它用来创建转换用样式表。它以将 XML 解析成一个树结构为运行前提,并输出一个由转换数据组成的结果树。XSLT 使用强有力的模式匹配机制来选取一个 XML 文件的部分进行转换。当一个模式与一棵树的一个部分相匹配时,一个模板就会用来确定这个树的这个部分如何进行转换。在下一章中,你将会了解到更多有关模板和模式的细节。

XSLT 使用另一项 XML 技术来提供它的文件转换特征。XPath 技术在 XSLT 中被用来选取用于执行的元素并产生文本。下一小节将更详细的讨论 XPath。

10.2.2 XPath

XPath 是一种表达式语言(非 XML),用来记录一个 XML 文件的各部分,但它并不用 XML 词表实现。这是因为,XPath 表达式是在 XML 标记不能真正适用的场合下使用的,比如属性值。XPath 提供了记录 XML 文件各部分的抽象方法,它组成了 XSLT 中文件记录的基础。XPath 使用的语法是为了在 URI 和 XML 属性值中使用而设计的,这就需要它非常的简明。XPath 的名字来源于使用路径符号记录 XML 文件的思想。

如同 XSLT,XPath 需要在假定文件已经分析成一棵树的情况下执行。它定义了节点的不同类型用来描述节点在树中的出现位置。通常总是有一个单独的根节点作为 XPath 树的根节点,它通常是树的第一个节点。文件中的每一个元素在根节点以下的树上有一个相对应的元素节点。一个元素节点内是其他类型节点,它与元素的内容一致。元素节点有一个独特的标记与它们关联,这些标记用于在 XPath 中引用节点。

XSLT 使用 XPath 作为遍历 XML 文件得到所给节点的基础。这种遍历通过使用表达式实现。XPath 规范描述了如何使用 XPath 语法建立表达式。这个规范内容位于 <http://www.w3.org/TR/xpath>。当 XPath 表达式被运算时,它们的结果属于指定类型的数据对象,如布尔型或是数型值。XPath 表达式运算的方式完全依赖于内容的表达式,这并不由 XPath 确定。也就是说,这是 XPath 的抽象部分,它可以允许作为 XSLT 的辅助技术来使用,以用来记录文件各部分。一个 XPath 表达式的内容由 XSLT 来确定,它还依次定义表达式如何被运算。

注意: XPath 也与 XPointer 和 XLink 技术联合使用以提供 XML 的高级链接特性。

第 19 章“使用 XPointer 和 XLink 链接文件”中讲述了 XPath 在 XPointer 和 XLink 中的作用。

10.2.3 XSL 格式化对象

XSL 格式化对象是 XSL 技术中的格式化部分。它正在被设计成一种 CSS 的功能扩展集,也就是说它将实现 CSS 的所有功能并有更多功能。XSL 格式化对象的语法不必与 CSS 相匹配,但要具有同样的功能。与 XSLT 类似,XSL 格式化对象是一个 XML 词表,它既适用于最小化初级程序员的学习负担,又为使用已知 XML 工具开发样式表打开了方便之门。

XSL 格式化对象在一个 XML 数据树中执行,这既可以是直接从文件分析出来也可以是使用 XSLT 从文件中转换而来。为了完成格式化,XSL 格式化对象将树的每个节点作为

一个格式化对象来处理,每个节点都支持大量的描述样式。你可以通过设置树上的给定元素(节点)的属性来应用样式。

有一些格式化对象符合文件格式化的不同方面。例如分页、布局和内容样式。每一个格式化对象都有用于描述某种对象的特征。一些特征直接指定了一个格式化结果,如颜色或是字体。而其他的特征只是对可能的格式化结果进行限制。

XSL 格式化对象使用的模型按照矩形区域和空间进行描述。当你认识到这种方法被大部分的平台应用程序使用时,你就不会对此感到过于惊讶。在 XSL 格式化对象中的矩形区域本身并不是对象,直到 XSL 格式化对象设置矩形区域和它们之间的关系时为止。这方面有点类似于 CSS 中的区域,在那里你通过确定一个段落或图片的长度和宽度来设置一个区域的尺寸,而不是实际使用区域元素。

当 XSL 处理机处理一个格式化对象时,它在显示方面被映射到一个矩形域中。对象的特征确定了它如何被格式化,以及它所被映射的区域的参数。

我认为在这次讨论中,我们使用了非常普通的术语来描述 XSL 格式化对象背后的机制。但在本书编写期间,这项技术还处于开发阶段。一旦浏览器通过补丁实现了对它的支持,XSL 格式化对象将变成一项更加有趣的技术。Microsoft 在开始阶段处于领导的地位,它支持 XSL 格式化对象的一个早期版本,就像它对 XSLT 支持一样。

10.3 总 结

XSL 是一种特别为 XML 使用而设计的强大的样式表技术。与 CSS 不同,XSL 允许你将 XML 内容从一种词表转换到另一种。而 CSS 只支持格式化 XML 内容。它包含的编程结构允许你对 XML 数据执行功能更强大的操作,诸如过滤、分类和搜索。即使 XSL 还处于开发阶段,它仍然具有优秀技术的素质,并必将保证 XML 拥有一个光明的未来。

第11章 创建XSL样式表

XSL 样式表用来转换 XML 文件内容并将它格式化以在网络浏览器中显示。虽然 XSL 的完整规范在本书编写时还处于制作当中,但 XSL 转换(XSLT)部分已经基本要作为 W3C 的推荐技术。Microsoft 冒失的在 Internet Explorer 5.0 中支持 XSLT 的一个早期版本,这就给网络开发人员一个绝好的机会来修补 XSLT 并开始创建 XSL 样式表。

本章将解决创建 XSL 样式表的基础知识。由于 XSL 的格式化部分还不能被 Internet Explorer 5.0 支持,这就需要使用 XSLT 将 XML 文件转换为可以用于显示的 HTML 文件。本章将为你讲解使用 XSL 样式表将 XML 文件转换为 HTML 文件的基本知识。

11.1 XSL 与 Internet Explorer 5.0

在你能够创建 XSL 样式表之前,有一点是非常重要的,那就是确切的理解什么浏览器支持 XSL 并支持到什么程度。在本书编写时,Internet Explorer 5.0 是惟一支持 XSL 的商业浏览器。这种支持由 XSL 技术中的 XSLT 部分的部分实现组成,它可以将文件从一种 XML 词表转换为另一种。这样看来,XSL 的 XSL 格式化对象部分要成为浏览器中的可运行版本还要过一段日子。

有一点很值得提及,那就是 Microsoft 并没有严格的遵循 W3C 提出的 XSLT 最新的工作草案。实际上,对 XSL 的一些私人扩展很明显将只能被 Internet Explorer 5.0 支持。这并不是一个太大的问题,因为 XSLT 还处在草案阶段。但如果你尝试在建立用于 Internet Explorer 5.0 的样式表中遵循 XSLT 工作草案操作,那么这就会出现问題。这方面你最好的解决方法就是按照 Microsoft 对 XSL 的要求来工作,同时谨记当对 XSLT 的推荐完成时,W3C 会使这一切变得有所不同。我们只能希望 Microsoft 在并成为 W3C 推荐技术并要求严格遵守时可以对 Internet Explorer 进行必要的修改。

由于 Microsoft 的 XSL 版本与 W3C 制订的 XSL 工作草案并不一致,一些 XSL 开发人员选择使用扩展名为 .msxsl 的用于 Internet Explorer 5.0 的 XSL 样式表文件来取代扩展名为 .xsl 的文件。我们就要注意这个事实,那就是这个样式表是基于 Microsoft 对 XSLT 的支持而编写的,这个支持的版本与 W3C 的工作草案不同。在本章中,我将用 .xsl 文件作为例子,这是因为现在 Microsoft 对 XSL 的浏览支持的版本只对它自身有效。然而,如果 XSL 被其他的浏览器支持但又与 Microsoft 的 XSL 版本不同,你可能更希望了解这种命名惯例。

11.2 深入 XSL 样式表

虽然 XSL 样式表功能强大,在样式化复杂的 XML 文件的时候仍可能会有点复杂。但它的基本结构是非常简单易懂的。一个 XSL 样式表由一个或多个用来描述模式的模板组成,这些模式用来为样式化与 XML 内容进行匹配。在 XSL 样式表中有两个基本构件,它们

是：

- 模板
- 模式

注意：要谨记，这次对 XSL 样式表的讨论主要是集中在 XSLT 方面。因此很自然的重点将放在使用样式表转换 XML 文件的问题上，而不是使用 XSL 格式化对象进行格式化。虽然，XSL 格式化对象并不属于本讨论的范畴，但 XML 文件还是使用 XSL 来格式化的。然而，这种转换是通过将 XML 文件转换为 HTML 文件来实现的。在某种意义上，这一章将 HTML 作为一种描述性标记语言来使用。这也是在 XML 样式化技术被广泛支持前，XSL 的普遍使用方式。

下面两节将对模板和模式进行更多的探讨。然而在进一步深入探讨之前，你需要学习一下 stylesheet 元素以及在 XSL 样式表中 XSL 名字域是如何使用的。

stylesheet 元素是一种 XSL 样式表的文件元素。这个元素是 XSL 名字域的一部分，它与其他元素和属性一起标记 XSL。你需要声明 XSL 名字域来使用 XSL 元素和属性。下面是一个在 stylesheet 元素中声明 XSL 名字域的例子：

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

在这个例子中，XSL 名字域的前缀规定为 xsl，这是使用 XSL 样式表的公用方法。将这个前缀放在所有的 XSL 元素和属性之前。

11.2.1 模板

模板是一个 XSL 构件，它描述基于某种模式匹配标准的所生成的输出。它的主要思想是定义一个应用于 XML 文件的某部分的转换结构。我们可以创建一个只包括单个模板的样式表，这样模板操作检索 XML 树的细节来格式化指定的内容。多个模板也可同时被使用，这种情况下每个模板与 XML 树中的某个部分相对应。多个模板实质上提供了一种转换 XML 文件的程序化途径。

要理解使用单独模板或多个模板的决定并不只是一个个人的编程爱好，这一点非常重要。这个决定极大的影响着样式表的功能，因此当你创建样式表的时候，明智的选择使用模板是非常重要的。幸运的是，当一种方法比另一种方法更有意义时，这种决定变得非常容易。

模板在 XSL 样式表中通过 xsl:template 元素定义，这个元素本质上是一个模式和转换数据的容器元素。xsl:template 元素使用名为 match 的可选属性在 XML 文件中进行模式匹配。match 属性指定一个文件的 XML 树的一部分。对文件最大可能的匹配就是将 match 属性设置为“/”，它象征着树的根被匹配。结果就是这个树都被模板选中：

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

模板中的 match 属性有点像数据库语言中的查询——与 match 属性相匹配的 XML 文件中的数据传给模板以进行处理。例如，如下所示的程序只匹配名为 state 的元素：

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

```
<xsl:value-of/>  
</xsl:template>
```

在某些方面,XSL 的匹配机制与 CSS 的选择器相同,而 XSL 模式匹配功能更强。事实上,它实际是一个 XML 的全特征查询语言。

在前面的例子中,你可能注意到了 `xsl:value-of` 元素。这个元素用来插入一个元素或属性的内容以进行输出。例如, `state` 元素的内容作为文件转换的一部分被输出。当你在本章后面部分看到了 `xsl:value-of` 元素被用在一个复杂样式表的内容中时,你将对这个元素获得更多的感性认识。

11.2.2 模式

模式用在 `xsl` 模板中以执行匹配,基本上,它们可以可靠的确定 XML 文件的哪一部分被传送给指定的模板以进行转换。一个模式描述了 XML 树的一个分支,它依次由一组分层结构节点组成。XSL 模式使用的语法与在磁盘驱动器中使用指定路径有点类似。例如,模式 `addressbook/contact/phone` 选择在 `contact` 元素下出现的 `phone` 元素,而 `contact` 本身又出现在 `addressbook` 元素之下。

要选择这个文件树,你需要使用根模式。它由一个单独的正斜杠(/)组成。根模式也可以被设定在你认定是根的另一个模式上。例如,`addressbook/contact/phone` 被设定为文件根的开始,这就意味着 `addressbook` 作为根来引用。

模式用来遍历 `xsl` 来描述文件树的各部分以对之应用模板。除了我在这一节所展示的模式的简单应用外,XSL 模式还可以执行更多的与复杂模式匹配的任务。实际上,模式构成了一个小型查询语言,它可以用来提供对 XML 文件被选择用模板转换的部分的严格控制。

在第 22 章“使用 XSL 模式和 XQL”中,你将学习更多创建 XSL 模式的输入和输出知识。通过检验 Microsoft 的 XSL Visual Pattern Builder,我们可以体会一下用可视化方法创建模式。这个工具允许你在图形环境下创建模式,并通过将它们应用到 XML 文件上以进行检验。XSL Visual Pattern Builder 是免费软件,它可以从 <http://msdn.microsoft.com/downloads/samples/internet/xml/xsl-pattern-builder> 上下载。图 11.1 显示了 XSL Visual Pattern Builder 用来在地址簿 XML 文件中匹配 `name` 元素的情况。

11.3 XSLT 模板结构

XSLT 词表定义了一些控制 XSL 样式表中的模板应用的模板构件。这些模板构件就是在 XSL 名字域中定义的元素。下面所列的是一些最常见的 XSLT 元素:

- `xsl:value-of`
- `xsl:if`
- `xsl:for-each`
- `xsl:apply-templates`

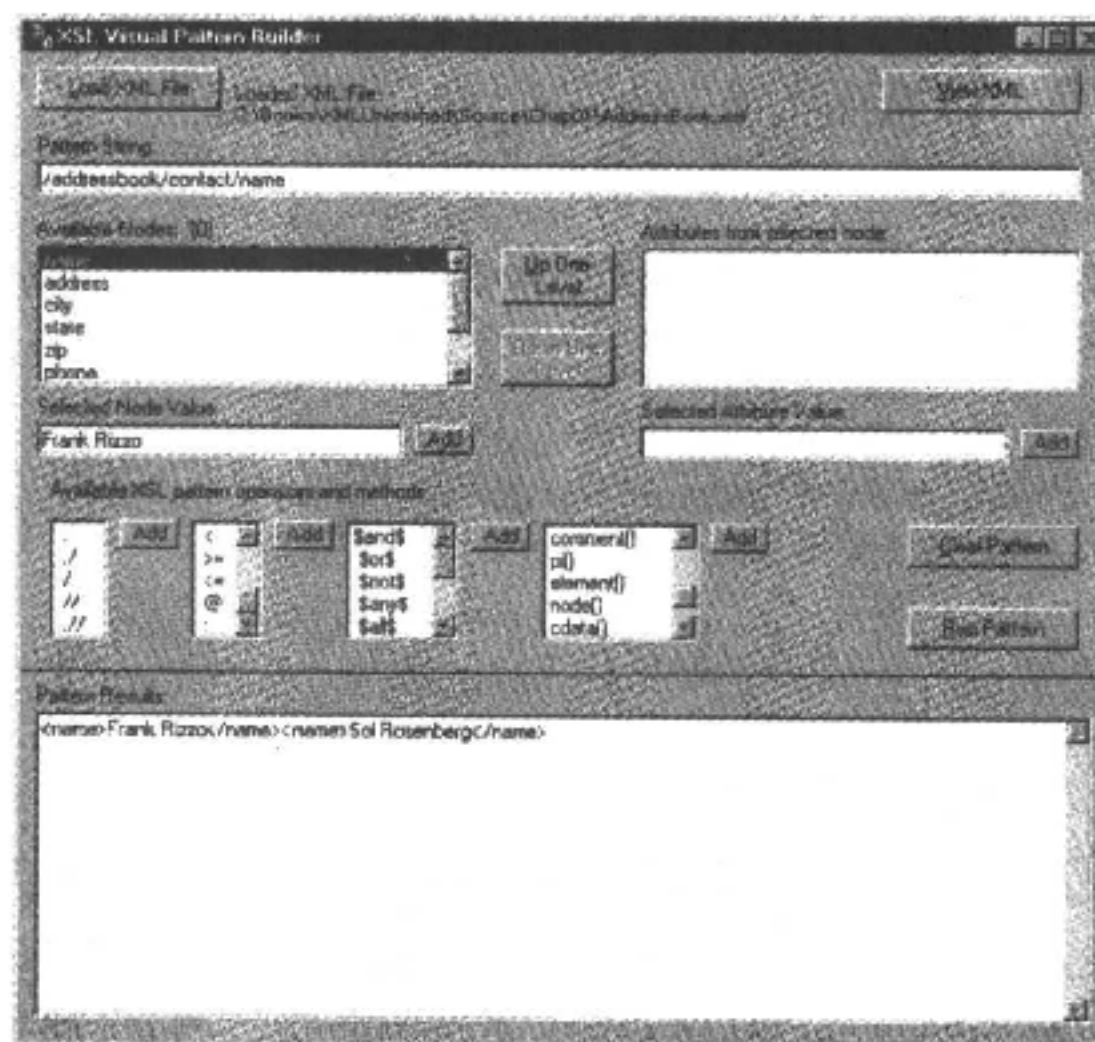


图 11.1 运行中的 XSL Visual Pattern Builder

下面的几小节将更详细的考察这些 XSLT 元素。你也可以在本章稍后的“开发 XSL 样式表”小节中学到如何在实际的 XSL 样式表的内容中使用它们。

11.3.1 xsl:value-of 元素

xsl:value-of 元素用来给样式表的结果输出中的元素和属性插入值。它提供了 XML 文件转换机制,这样它可以允许你以事实上的所有内容形式输出 XML 数据,比如使用 HTML 标记。下面是一个使用单独模板的例子,它使用 xsl:value-of 元素将名为 name 的元素的值按照 HTML 的 h2 标题样式输出:

```
<xsl:template match="name">
  <h2><xsl:value-of/></h2>
</xsl:template>
```

11.3.2 xsl:if 元素

xsl:if 元素用来执行在模板中的有条件匹配。这个元素使用与 xsl:template 元素相同的 match 属性来设置模板中的条件分支。下面是一个使用 xsl:if 元素的例子,它测试电影的 rating 属性值是否等于 5:

```
<xsl:if match="@rating=5">
  <xsl:apply-templates select="movies"/>
</xsl:if>
```

11.3.3 xsl:for-each 元素

xsl:for-each 元素用来设置一个对于文件中元素的反复处理循环。select 属性确定是哪个元素被选进循环重复部分。另一个 xsl:for-each 提供的重要属性就是 order-by,它指定被

xsl:for-each 处理的数据的区分标准。下面是一个使用 xsl:for-each 的例子。它重复处理一个汽车分类表。

```
<xsl:for-each order-by=" + price" select="vehicles/vehicle">
  <tr>
    <td><xsl:value-of select="@year"/></td>
    <td><xsl:value-of select="@make"/></td>
    <td><xsl:value-of select="@model"/></td>
    <td><xsl:value-of select="@mileage"/></td>
    <td><xsl:value-of select="@color"/></td>
    <td><xsl:value-of select="@price"/></td>
  </tr>
</xsl:for-each>
```

11.3.4 xsl:apply-templates 元素

xsl:apply-templates 元素用来使用在样式表中定义的模板。它支持 select 和 order-by 属性,它们扮演着与在 xsl:for-each 的相关属性中相同的角色。当 XSL 处理机在样式表中遭遇到一个 xsl:apply-templates 元素时,与在 select 属性中的模式相对应的模板就会被启用。也就是说,相应的文件数据就会填入模板并被转换。下面是一个使用 xsl:apply-templates 元素应用模板的例子:

```
<xsl:apply-templates select="state"/>
```

11.4 开发 XSL 样式表

理解 XSL 样式表的关键就是看它们的运行。我可以对 XSL 模式和模板结构的复杂细节写整整一本书,但你实际需要的是看一些如何在内容中体现 XSL 样式表的功能和灵活性的例子。我将重新回到前几章令人熟悉的地址簿 XML 文件来示范如何创建一个简单的 XSL 样式表。清单 11.1 列出了 AddressBook.xml 文件的代码。

清单 11.1 地址簿 XML 文件(AddressBook.xml)

```
<? xml version="1.0"? >
<? xml-stylesheet href="AddressBook.xsl" type="text/xsl"? >
<! DOCTYPE addressbook SYSTEM "AddressBook.dtd" [
<! ENTITY amp "&#38;#38;" >
<! ENTITY apos "&#39;" >
]>
<addressbook>
  <!-- This is my good friend Frank. -->
  <contact>
    <name>Frank Rizzo</name>
    <address>1212 W 304th Street</address>
    <city>New York</city>
    <state>New York</state>
    <zip>10011</zip>
    <phone>
```



```

    <voice>212-555-1212</voice>
    <fax>212-555-1213</fax>
  </phone>
  <email>frizzo@fruity.com</email>
  <web>http://www.fruity.com/rizzo</web>
  <company>Frank's Ratchet Service</company>
</contact>

<!-- This is my old college roommate Sol. -->
<contact>
  <name>Sol Rosenberg</name>
  <address>1162 E 412th Street</address>
  <city>New York</city>
  <state>New York</state>
  <zip>10011</zip>
  <phone>
    <voice>212-555-1818</voice>
    <fax>212-555-1819</fax>
  </phone>
  <email>srosenberg@fruity.com</email>
  <web>http://www.fruity.com/rosenberg</web>
  <company>Rosenberg's Shoes & Glasses</company>
</contact>
</addressbook>

```

用于这个文件的 XSL 样式表需要重复处理 contact 元素, 并使用 HTML 代码格式化 contact 信息。xsl:for-each 元素用来做处理 contact 元素的循环十分合适。样式表如清单 11.2 所示。

清单 11.2 地址簿 XML 样式表(AddressBook.xsl)

```

<? xml version="1.0" ? >
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <html><head><title>Address Book XML Example</title></head>
    <body bgcolor="FFFFFF">
      <xsl:for-each select="addressbook/contact">
        <xsl:apply-templates select="name"/>
        <xsl:apply-templates select="address"/>
        <xsl:apply-templates select="city"/>
        <xsl:apply-templates select="state"/>
        <xsl:apply-templates select="zip"/>
      </xsl:for-each>
    </body>
  </html>
</xsl:template>

  <xsl:template match="name">
    <h2><xsl:value-of/></h2>
  </xsl:template>

  <xsl:template match="address">
    <xsl:value-of/><br/>
  </xsl:template>

```

```
</xsl:template>
<xsl:template match="city">
  <xsl:value-of/> ,
</xsl:template>
<xsl:template match="state">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="zip">
  <xsl:value-of/><br/>
</xsl:template>
</xsl:stylesheet>
```

这个样式表说明了如何使用 `xsl:for-each` 元素来建立循环,重复处理 `contact` 元素。要注意的是用于 `xsl:for-each` 的 `select` 属性被设置为“`addressbook/contact`”,这样循环只选择文件中的 `contact` 元素进行处理。`xsl:value-of` 元素用来将地址簿元素写入到用于样式表的结果树。要注意 XSL 名字域要在 `xsl:stylesheet` 元素中声明。

还要注意样式表中混合着 HTML 标记,这一点非常重要。这说明了一个事实,就是样式表的任务是将一个 XML 文件转换为一个严格的 HTML 文件。这就是使用 XSL 将 XML 文件转换为 HTML 的全过程。图 11.2 显示了地址簿 XML 文件在 Internet Explorer 5.0 中显示时样式表的处理结果。

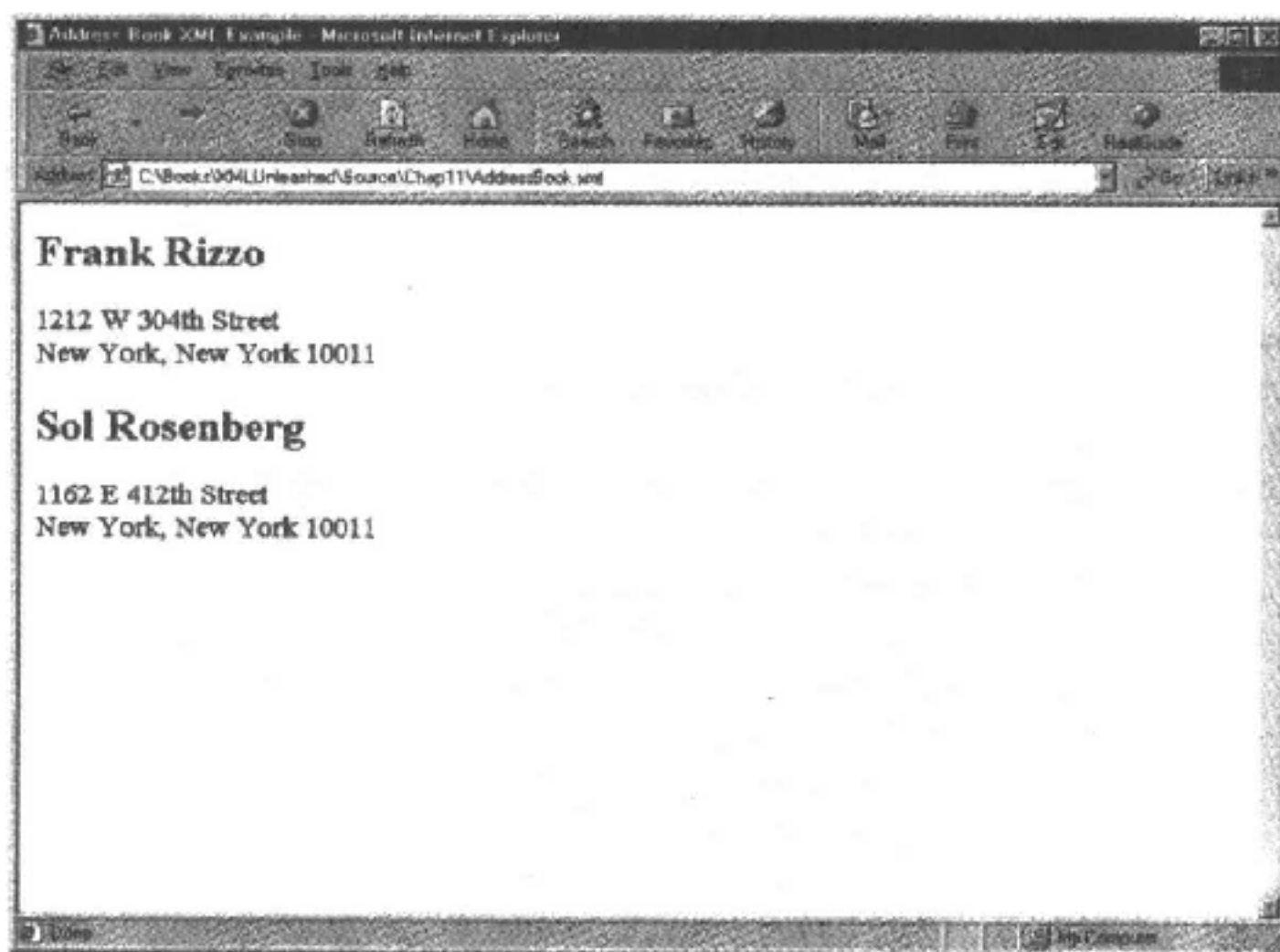


图 11.2 使用 XSL 样式表处理地址簿 XML 文件在 Internet Explorer 5.0 中的显示情况

地址簿样式表确实非常的简单,但它几乎没有使你对 XSL 的先进功能留下印象。另一个有趣的例子中包含了一个用某种方法处理的 XML 内容显示的例子。下一个例子展示了如何使用样式表对 XML 数据进行分类并用 HTML 表的形式显示出来。

在第 27 章“使用 XML 生成器移植数据”中,我们将创建一个名为 Vehicles.xml 的 XML 文件,它将包含一个二手车销售列表。这个文件是一个极好的例子,因为它包含了可用于分类的 XML 数据。清单 11.3 列出了 Vehicles.xml 文件的代码。

清单 11.3 车辆 XML 文件(Vehicles.xml)

```
<? xml version="1.0" ? >
<? xml-stylesheet href="Vehicles.xsl" type="text/xsl" ? >
<vehicles>
  <vehicle year="1996" make="Land Rover" model="Discovery">
    <mileage>36500</mileage>
    <color>black</color>
    <price>$ 22100</price>
  </vehicle>
  <vehicle year="1998" make="Land Rover" model="Discovery">
    <mileage>15900</mileage>
    <color>teal</color>
    <price>$ 32000</price>
  </vehicle>
  <vehicle year="1997" make="Land Rover" model="Discovery">
    <mileage>46000</mileage>
    <color>silver</color>
    <price>$ 27900</price>
  </vehicle>
  <vehicle year="1997" make="Land Rover" model="Defender 90">
    <mileage>21050</mileage>
    <color>white</color>
    <price>$ 41900</price>
  </vehicle>
  <vehicle year="1994" make="Land Rover" model="Defender 90">
    <mileage>42450</mileage>
    <color>green</color>
    <price>$ 31250</price>
  </vehicle>
  <vehicle year="1996" make="Toyota" model="Land Cruiser">
    <mileage>34800</mileage>
    <color>black</color>
    <price>$ 35995</price>
  </vehicle>
  <vehicle year="1997" make="Toyota" model="Land Cruiser">
    <mileage>47150</mileage>
    <color>green</color>
    <price>$ 37000</price>
  </vehicle>
  <vehicle year="1999" make="Chevrolet" model="Suburban 2500">
    <mileage>3550</mileage>
    <color>pewter</color>
    <price>$ 31995</price>
  </vehicle>
```

```

<vehicle year="1996" make="Chevrolet" model="Suburban 2500">
  <mileage>49300</mileage>
  <color>green</color>
  <price>$ 25995</price>
</vehicle>
</vehicles>

```

就像你看到的,这个文件包含着二手车的信息,它们通过简单的 XML 词表标记。使用一个样式表将这个 Vehicles.xml 文件转换为一个按价格排列的汽车列表将是非常方便的。清单 11.4 列出了执行这个特殊任务的样式表。

清单 11.4 车辆 XSL 样式表(Vehicles.xsl)

```

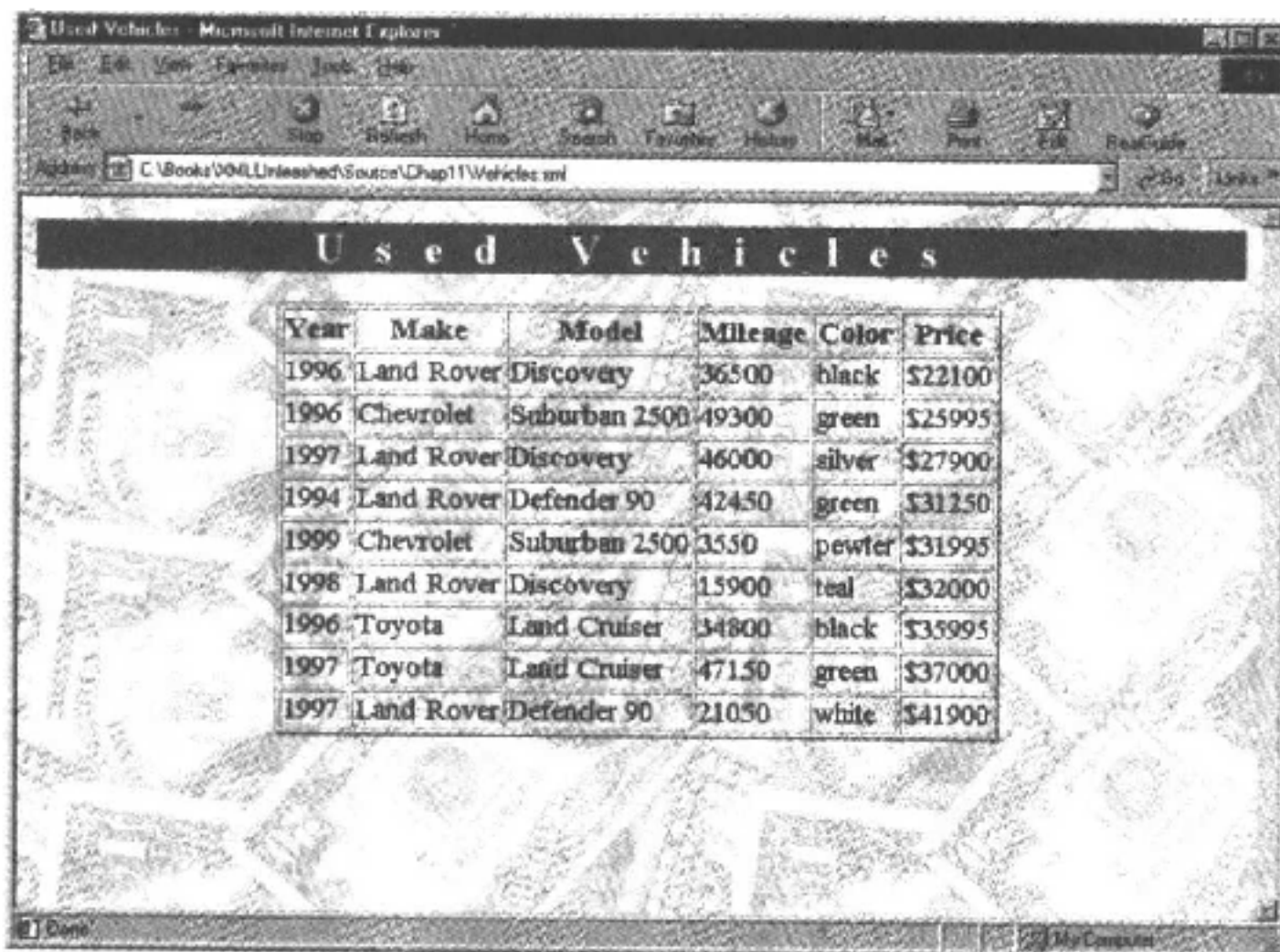
<? xml version="1.0" ? >
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD=xsl">
  <xsl:template match="/">
    <html>
      <head>
        <title>Used Vehicles</title>
      </head>
      <body background="Money.jpg" >
        <h1 style="background-color: #446600;
        color: #FFFFFF;font-size: 20pt;text-align: center;
        letter-spacing: 1.0em">Used Vehicles</h1>
        <table align="center" border="2">
          <tr>
            <th>Year</th>
            <th>Make</th>
            <th>Model</th>
            <th>Mileage</th>
            <th>Color</th>
            <th>Price</th>
          </tr>
          <xsl:for-each order-by=" + price" select="vehicles/vehicle">
            <tr>
              <td><xsl:value-of select="@year"/></td>
              <td><xsl:value-of select="@make"/></td>
              <td><xsl:value-of select="@model"/></td>
              <td><xsl:value-of select="mileage"/></td>
              <td><xsl:value-of select="color"/></td>
              <td><xsl:value-of select="price"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>

```

样式表使用一个模板排列并显示车辆信息。xsl:for-each 元素用来重复处理 vehicle 元

素。每一个车辆的使用期、制造、车型、行驶英里数、颜色和价格作为 HTML 表单数据输出。要注意使用期、制造和车型的前面都被特别加了符号@,这说明它们是属性。而其余的信息是元素,它们不需要加前导记号。

这个样式表的真正关键之处是通过价格对车辆进行排列。这是通过 `xsl:for-each` 元素中的 `order-by` 属性来实现的。通过设置“+ price”来说明车辆元素是根据 price 元素的值升序排列的。图 11.3 显示了 Vehicles.xml 文件使用这个样式表在 Internet Explorer 5.0 的显示效果。



| Year | Make | Model | Mileage | Color | Price |
|------|------------|---------------|---------|--------|---------|
| 1996 | Land Rover | Discovery | 36500 | black | \$22100 |
| 1996 | Chevrolet | Suburban 2500 | 49300 | green | \$25995 |
| 1997 | Land Rover | Discovery | 46000 | silver | \$27900 |
| 1994 | Land Rover | Defender 90 | 42450 | green | \$31250 |
| 1999 | Chevrolet | Suburban 2500 | 3550 | pewter | \$31995 |
| 1998 | Land Rover | Discovery | 15900 | teal | \$32000 |
| 1996 | Toyota | Land Cruiser | 34800 | black | \$35995 |
| 1997 | Toyota | Land Cruiser | 47150 | green | \$37000 |
| 1997 | Land Rover | Defender 90 | 21050 | white | \$41900 |

图 11.3 使用 XSL 样式表处理的车辆 XML 文件在 Internet Explorer 5.0 中的显示情况

你已经看到这两个简单的样式表与依照相关的简单 XML 词表建立的 XML 文件联合使用的情况。让我们看另一个更现实的基于较复杂 XML 词表的例子。为了看到 XSL 样式表的真正强大之处,这样做是非常值得的。

第 39 章“使用 HRMML 管理人事资源”中将讨论如何使用人事管理标记语言(Human Resource Management Markup Language,HRMML)来标记履历,它会创建一个虚拟的名为 Resume.xml 的履历。HRMML 是一个全面的 XML 词表,用来描述履历和工作方向的细节。Resume.xml 是一个相当复杂的 XML 文件。清单 11.5 列出了 Resume.xml 文件的代码清单。这里是 Peter Parker(蜘蛛人)的一个假想履历。

清单 11.5 履历 XML 文件(Resume.xml)

```
<? xml version="1.0" ? >
<? xmlstylesheet href="Resume.xsl" type="text/xsl" ? >
<resumefatabase>
  <resume>
    <resumeprilog>
      <revisiondate>
```

```

        <date><month>November</month><day>30</day><year>1999</
year>
        </date>
    </revisiondate>
    <availabilitydate>Two weeks notice</availabilitydate>
    <compensationdetail>
        <salary>
            <required>$ 50,000</required>
            <current>$ 47,500</current>
        </salary>
        <benefits>
            <current>Health, Life, 401(k)</current>
            <required>Would consider contract positions without
                benefits</required>
        </benefits>
    </compensationdetail>
    <postdetail>
        <postwhere>
            <website>www.monster.com</website>
            <postwhen>
                <poststart>
                    <date><month>December</month><day>2</day><year>
1999</year>
                    </date>
                </poststart>
                <postexpire>
                    <date><month>June</month><day>2</day><year>2000
</year>
                    </date>
                </postexpire>
            </postwhen>
        </postwhere>
        <postwhere>
            <website>www.hotjobs.com</website>
            <postwhen>
                <poststart>
                    <date><month>December</month><day>2</day><year>
1999</year>
                    </date>
                </poststart>
                <postexpire>
                    <date><month>June</month><day>2</day><year>2000
</year>
                    </date>
                </postexpire>
            </postwhen>
        </postwhere>
    </postdetail>
</resumebody>
</resumebody>
    <personaldata>
        <name>Peter Parker</name>

```

```

<address>
  <addressline>1234 Walnut Lane</addressline>
  <city>Spiderville</city>
  <state>NY</state>
  <postalcode>55512</postalcode>
</address>
<email>spidey@aol.com</email>
<voice>
  <areacode>212</areacode>
  <telnumber>555-1212</telnumber>
</voice>
</personaldata>
<resumesection sectype="qualifsummary">
  <sectiontitle>Summary of Qualifications</sectiontitle>
  <secbody>
    <p>I have three years of experience as a research scientist
    specializing in the field of molecular biology. More
    importantly, I have six years of experience as a superhero,
    operating under the pseudonym "Spiderman". My superpowers give
    me the capability of scaling walls and lifting objects several
    times my own weight. I am a vigilant fighter of crime, capable
    of using my superpowers to match wits and defeat criminal
    masterminds. </p>
  </secbody>
</resumesection>

  <resumesection sectype="experience">
    <sectiontitle>Experience</sectiontitle>
    <secbody>
      <resumesection sectype="experience">
        <sectiontitle>
          <jobtitle>Research Scientist</jobtitle> for
          <employername>Spiderville High School</employername> from
          <startdate><date><month>February</month>
          <year>1993</year></date></startdate> to
          <enddate><date><month>May</month>
          <year>1996</year></date></enddate>.
        </sectiontitle>
        <secbody>
          <p>Researched the applications of low power nuclear energy
          in controlled environments. </p>
          <ul>
            <li>Formed theories, wrote papers, gathered data, and
            performed experiments. </li>
            <li>Assessed the potential of experimental findings. </li>
            <li>Ensured that safety standards were met. </li>
            <li>Left the position after an accident involving a
            leakage of radiation. </li>
          </ul>
        </secbody>
      </resumesection>
      <resumesection sectype="experience">
        <sectiontitle>
          <jobtitle>Superhero</jobtitle> for the town of Spiderville

```


清单 11.6 履历 XSL 样式表 (Resume.xsl)

```

<? xml version="1.0"? >
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template>
    <xsl:value-of/>
  </xsl:template>
  xsl:template match="/">
    <html><head><title>HR XML Example</title></head>
    <body bgcolor="#FFFFFF">
      <table>
        <tr>
          <td width="40">
          </td>
          <td width="580">
            <h1 style="padding-left: 15px; background-color: #000000;
            color: #ffffff; font-family: Verdana,Arial;
            font-size: 18pt; text-align: center; font-style: italic;
            letter-spacing: 0.5em">Resume</h1>
            <xsl:for-each select="resumedatabase/resume/resumebody">
              <xsl:apply-templates select="personaldata"/>
              <xsl:apply-templates select="resumesection"/>
            </xsl:for-each>
          </td>
        </tr>
      </table>
    </body>
  </html>
</xsl:template>
<xsl:template match="personaldata">
  <xsl:apply-templates select="name"/>
  <xsl:apply-templates select="address"/>
  <xsl:apply-templates select="voice"/>
</xsl:template>
<xsl:template match="resumesection">
  <xsl:apply-templates select="."/sectiontitle"/>
  <xsl:apply-templates select="secbody"/>
</xsl:template>
<xsl:template match="sectiontitle">
  <h4><font face="arial, geneva, lucida sans unicode, helvetica">
    <xsl:value-of/></font></h4>
  </xsl:template>
  <xsl:template match="secbody">
    <xsl:apply-templates select="."/secbody/resumesection/sectiontitle"/>
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="."/secbody/resumesection/sectiontitle">
    <font face="arial, geneva, lucida sans unicode, helvetica" size="-1">
      <b><xsl:apply-templates/></b></font>
    </xsl:template>
  <xsl:template match="p">
    <p><font face="arial, geneva, lucida sans unicode, helvetica"
    size="-1"><xsl:apply-templates/></font></p>

```

```

/xsl:template>
<xsl:template match="ul">?
  <ul><xsl:apply-templates select="li"/></ul>
</xsl:template>
<xsl:template match="li">
  <li><font face="arial, geneva, lucida sans unicode, helvetica"
    size="-1"><xsl:value-of/></font></li>
</xsl:template>
<xsl:template match="name">
  <h2><font face="arial, geneva, lucida sans unicode, helvetica">
    <xsl:apply-templates/></font></h2>
</xsl:template>
<xsl:template match="address">
  <font face="arial, geneva, lucida sans unicode, helvetica" size="-1">
    <b><i><xsl:apply-templates select="addressline"/>
      <xsl:apply-templates select="city"/>
      <xsl:apply-templates select="state"/>
      <xsl:apply-templates select="postalcode"/></i></b></font>
</xsl:template>
<xsl:template match="voice">
  <font face="arial, geneva, lucida sans unicode, helvetica" size="-1">
    <br/><b><i>(<xsl:apply-templates select="areacode"/>)
      <xsl:apply-templates select="telnumber"/></i></b></font>
</xsl:template>
<xsl:template match="addressline">
  <xsl:value-of/>,
</xsl:template>
<xsl:template match="state">
  <xsl:value-of/>,
</xsl:template>
<xsl:template match="city">
  <xsl:value-of/>,
</xsl:template>
<xsl:template match="postalcode">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="jobtitle">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="employername">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="month">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="year">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="day">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="startdate">
  <xsl:value-of/>
</xsl:template>

```

```
<xsl:template match="enddate">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="skillsqualif">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="experiencequalif">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="educationqualif">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="softwarequalif">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="prgmlangqualif">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="educationqualif">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="licensequalif">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="certificationqualif">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="otherqualif">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="hardwarequalif">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="employername">
  <xsl:value-of/>
</xsl:template>
</xsl:stylesheet>
```

虽然这个 XSL 样式表与你所见到的其他样式表相比相当的长。但如果你一步一步的学习就会发现它并不非常复杂。样式表从 `xsl:for-each` 元素开始,用来反复对 `resumebody` 元素进行处理。这就允许你显示保存在单独文件中的多个履历。`personaldata` 和 `resumesection` 模板也被应用,同时也导致对其他附加模板的调用。这个样式表提供了一个说明如何使用层次方法应用模板的很好的实例。样式表的其余部分主要是使用 `html` 标记对指定的履历信息进行格式化。

图 11.4 显示了履历 XML 文件使用 XSL 样式表处理后在 Internet Explorer 5.0 中的显示情况。

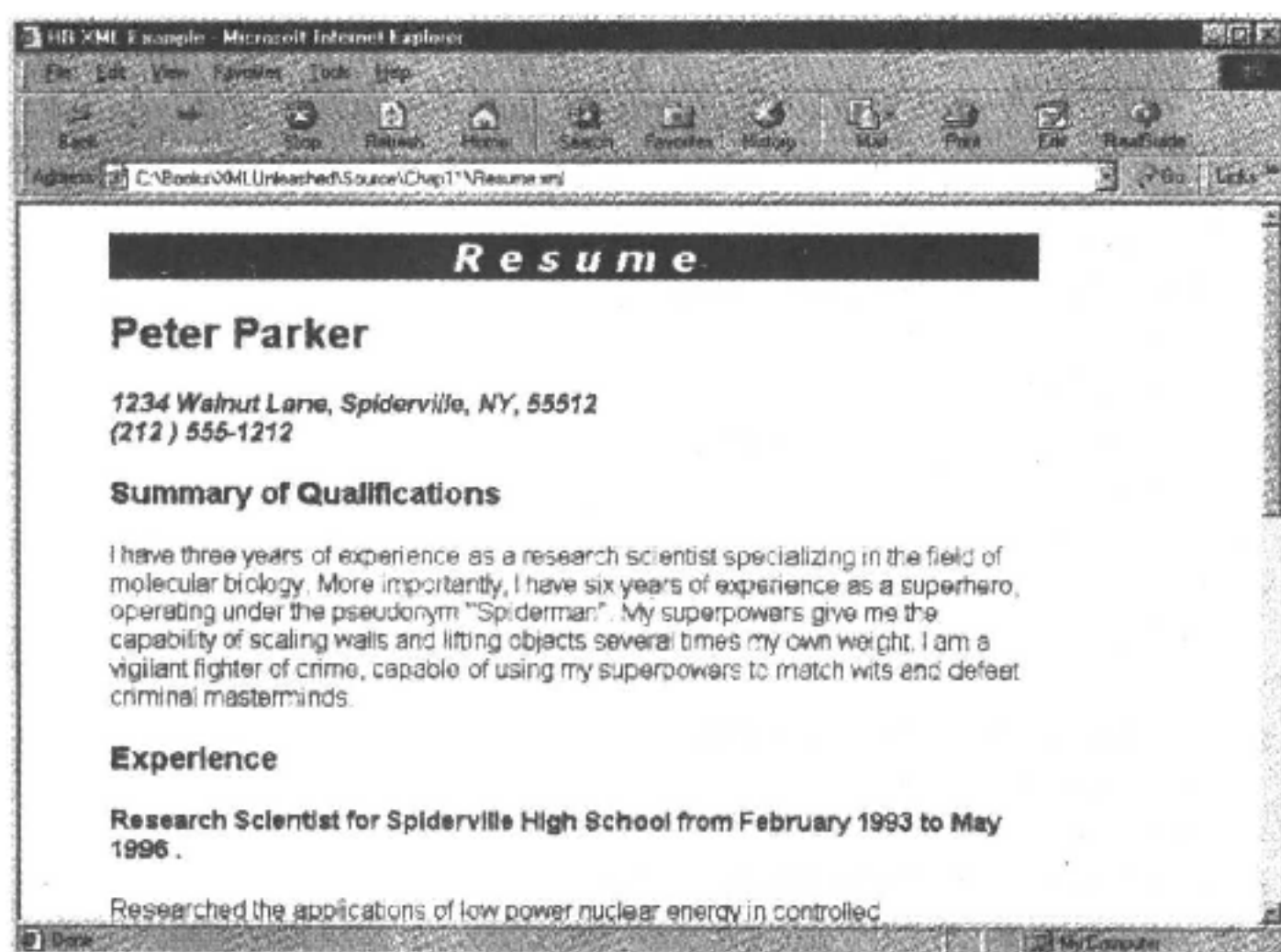


图 11.4 使用 XSL 样式表处理的履历 XML 文件在 Internet Explorer 5.0 中的显示情况

11.5 总 结

在将来,xsl 会被用来转换和格式化 XML 文件以进行显示,而现在使用 XSL 可以做一些有趣的事情。特别是,XSLT 可以被用来将 XML 文件转换为更高级结构的 HTML 文件,它可以在当今的浏览器中很容易的进行显示。在本书编写期间,Internet Explorer 5.0 是惟一支持 XSLT 的商用网络浏览器,但其他的浏览器也很有希望跟着做。当然,我们要密切注意什么时候 W3C 可以最终推荐 XSLT 技术,这样就可以要求浏览器供应商提供对 XSLT 的标准实现。

第3部分 XML 文件的处理

第12章 XML 处理基础

只有遵守 XML 规范的文件才能称为 XML 文件。在这一点上,XML 规范的制定者们达成了高度的一致,因为他们不希望看到 XML 又遇到现在困扰着 HTML 的那些问题。

HTML 的一个问题就是有许多根本不是 HTML 格式的文件也被当作是 HTML 文件。真正的 HTML 文件必须遵守某一个 HTML DTD 的规则。然而,作为事实上的工业标准的 HTML 文件概念被大大扩展了,任何可以在主流 HTML 浏览器上显示的文件都被称为 HTML 文件。浏览器厂商在他们的浏览器中加入了一个又一个复杂的引擎以便处理任何可能与 HTML 有关的事情。这导致了一个恶性循环——作者和用户代理们竞争着编写或者说是忍受着 HTML。

XML 的创建者们决心将这些不严格的地方去掉。他们制定的最基本的规则之一就是,如果一个文件不是结构良好的,那么用户代理就不能——事实上是必须不——按照通常的方法解析这个文件。从长远看来,这将使每一个人都受益。作者必须遵守 XML 标准而软件开发者也必须严格地实现这些标准。这样,大家就都知道自己应该作些什么以及可以从别人那里得到什么。

12.1 XML 文件的处理

解析 XML 文件是处理 XML 文件的第一步。下面就是用户代理通常处理 XML 文件的步骤。

解析器取得 XML 文件并检查该文件是结构良好的或是有效的。在大多数情况下,会生成一个解析树,该树含有 XML 文件中的所有对象。然后解析器将生成的解析树传给显示代理,并由显示代理来以某种方式显示该树。如果有样式表或者脚本与该 XML 文件相关联,那么解析器就生成该文件的样式以及一系列流式对象以显示它。图 12.1 说明了这个过程。

XML 解析器分为两种:

- 标准解析器,检查文件是否为结构良好的。
- 验证解析器,检查文件是否符合它的 DTD。

所有的 XML 解析器都必须检查文件是否为结构良好的。如果文件不是结构良好的,那么解析器就一定不能以正常方式来处理该文件了。如果有 DTD 而解析器是一个验证解析器,那么解析器就会根据 DTD 来检查文件的内容。

为什么让一个用户代理来解析 XML 文件呢?为什么不像 HTML 浏览器那样,直接尽可能地显示 XML 文件呢?为什么 XML 不可能像 SGML 那样,无需总是提供结束标记呢?

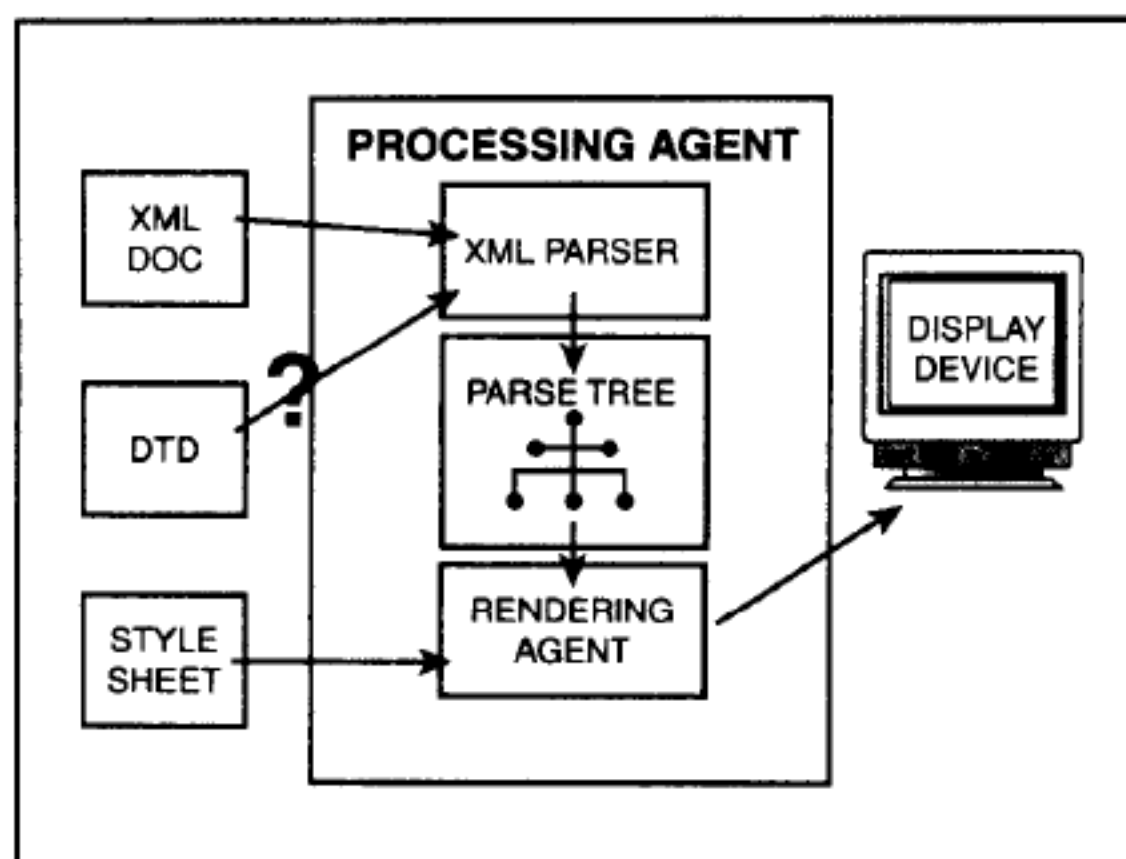


图 12.1 XML 文件的处理过程

12.2 为什么要解析 XML

正如本书前面提到的那样,XML 不是 HTML。在 HTML 中,处理代理(也就是 HTML 浏览器),具有内建的理解 HTML 的功能,而且它也很可能具有内建的“混杂识别”引擎并通过该引擎来理解最糟糕的 HTML 语法。

XML 也不是 SGML。在 SGML 中总有一个 DTD,所以 SGML 处理代理可以参照这个 DTD 并且决定如何显示这个 SGML 文件。

由于 XML 文件被设计为独立的文件,所以它必须包含所有 XML 处理代理需要的资源才能被正确地解析。这些资源就是使 XML 文件结构良好的元素。比如,如果没有结束标记,解析器就不可能辨别哪里是元素的结束。实际上,能使 XML 处理器知道什么时候才处理完一个元素的惟一办法就是使它遇到一个结束标记。和 SGML 中的不同,XML 处理器不能依赖 DTD 来确定某个结束标记是否可以没有,或者该元素是一个空元素。

XML 处理器辨别是否遇到了一个空元素的惟一办法就是让空元素具有特别的形式。在 SGML 中,处理代理可以从 DTD 中得到这样的信息。

只有所有元素都嵌套,XML 处理代理才能建立解析树。这是因为 XML 处理器并没有给定文件中元素的信息,所以它也不能识别出它们。

正因为这样,处理器所做的第一件事就是解析文件以确认它是结构良好的。如果文件不是结构良好的,那么其他的处理就毫无意义了。

制作一个可以显示不是结构良好的 XML 文件的软件也不是不可能,但是,所有编写过应用程序的人都知道,编写这样的程序是非常困难的。这让人想起了 XML 规范的目标之一:

“处理 XML 文件的程序必须容易编写。”

因此,大多数 XML 解析器在结构良好限制上是很严格的。一旦遇到了这样的错误,

XML 解析器会立即结束并显示错误代码。

图 12.2 显示了 IE5 对于不是结构良好的文件的处理结果。



图 12.2 IE5 显示了结构不是良好的文件

```
<? xml version="1.0"? >
<xdoc>
  <greeting>Hello XML! </Greeting>
</xdoc>
```

如果 IE5 发现了一个格式错误,它既不会去显示也不会试图去显示 XML 文件。

12.3 为什么要验证 XML 文件

验证解析器检查 XML 文件是否符合 DTD。如果符合,那么解析器就根据 DTD 来检查该文件的结构和内容。

为什么处理器不显示那些不是结构良好的文件比较容易理解,但是为什么还要根据 DTD 来验证文件的有效性呢? 如果 XML 文件被设计为在 Internet 上使用,为什么还要一起发送那些附带的 DTD 呢? 为什么不显示文件呢?

这些都是很好的问题。如果 XML 的目标仅仅是为了文件的显示,那么就完全不需要 DTD 了。然而,XML 文件不仅仅是为显示设计的。XML 文件是用于存储和结构化内容。以一种一致的方式存储内容的文件可以被分类,因而完成各种与之相关的功能就成为可能。

12.3.1 文件类

如果文件符合 DTD,那么这个文件就属于某一文件类。使用 DTD 和验证解析器来验证给定的 XML 文件实际上是否属于它所声明的那个文件类。一旦文件被确认属于某一文件类,就可以认为该文件具有某种特性,而且可以对它进行一些有趣的操作。比如,DTD 可以用来提供某种基本的内容检验,文件可以作为电子交易中的内容包来使用,可以用 DOM 来对文件进行排序、校勘或者其他操作。

举一个例子更能说明问题。清单 12.1 显示了一个可以用来记录一次交易的简单文件。

清单 12.1 记录了一次交易的 XML 文件

```

<? xml version="1.0"? >
<! DOCTYPE salesdoc SYSTEM "sale1.dtd" >
<salesdoc id="oh123">
  <item type="s234">Garden Hoe</item>
  <quantity >144</quantity>
  <price price="4.99">4.99</price>
  <purchaser history="new" name="acme">Acme hardware</purchaser>
  <deliver_date date="19990817">1999-08-17</deliver_date>
  <conditions payment="COD" />
</salesdoc>

```

12.3.2 用 DTD 来验证内容

请看清单 12.1, 元素的内容主要是元素的属性。通过 XML 处理器, 元素的文本内容可以产生人可读的显示。属性只能由机器理解, 文本内容也是如此。更重要的是, 如果信息被放在属性中, 就可以用 DTD 来检验是否填写了必要的文件字段。在 XML 格式成为公共的而专门通过它来验证文件有效性的软件被开发出来以后, 你还可以验证属性值是否是正确的数据类型。而使用传统的 DTD, 你所能够做的仅仅是验证是否设置了该值。事实上, 在应用程序中使用 XML 的时候, 通常我们用一个简单的程序或者是脚本来验证属性是日期还是数字。

下面是清单 12.1 的 DTD:

```

<!--This is the DTD for sale class of document -->
<! ELEMENT salesdoc (item,quantity,price,purchaser,deliver_date,conditions) >
<! ATTLIST salesdoc
  id ID #REQUIRED
>
<! ELEMENT item (#PCDATA) >
<! ATTLIST item
  type CDATA #REQUIRED
>
<! ELEMENT quantity (#PCDATA) >
<! ATTLIST quantity
  number CDATA #REQUIRED
>
<! ELEMENT price (#PCDATA) >
<! ATTLIST price
  price CDATA #REQUIRED
>
<! ELEMENT purchaser (#PCDATA) >
<! ATTLIST purchaser
  name CDATA #REQUIRED
  history CDATA #IMPLIED
>
<! ELEMENT deliver_date (#PCDATA) >

```



```
<! ATTLIST deliver_date
    date CDATA #REQUIRED
>
<! ELEMENT conditions EMPTY>
<! ATTLIST conditions
    payment CDATA #REQUIRED
>
```

如果你用一个解析器来验证这个文件,你会得到一个错误提示!图 12.3 显示了基于 Microsoft 的验证解析器的这个错误提示。

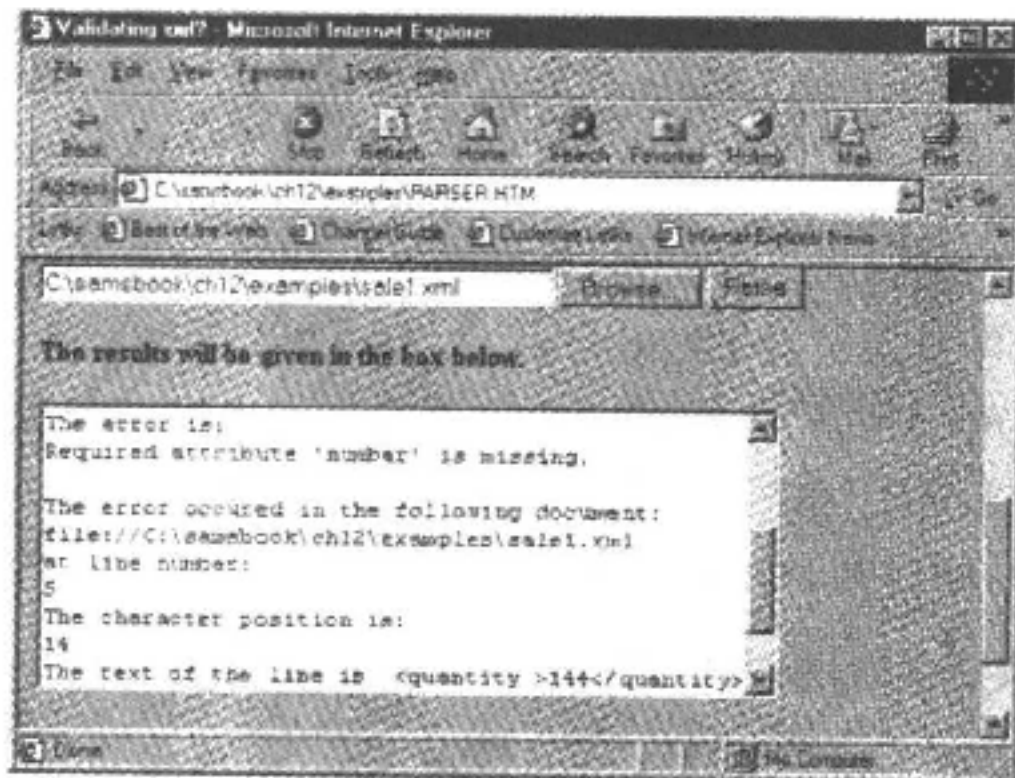


图 12.3 解析器显示的验证错误

解析器显示了 quantity 元素的属性 number 没有被设置。
在文件中加入下面的文本就可以解决这个问题了:

```
<quantity number="144">144</quantity>
```

注意:在第 14 章中我们已经看到了如何用 microsoft.XMLDOM ActiveX 对象来建立一个验证解析器。这个对象是和 IE5 浏览器捆绑在一起的,不过 IE5 浏览器仅仅检查文件是否是结构良好的,而不会进行有效性检查。要验证一个文件的有效性,你必须在某种应用程序中加入 MSXML.dll。

这是一个用 DTD 来进行简单的内容检查的例子。当然,如果作者在属性值中填写了一些垃圾,或者根本没有填写,文件仍然是有效的。但是,这种彻底的愚蠢是任何机器都解决不了的。

12.3.3 在 XML EDI 中使用 DTD

XML 在电子文件交换(EDI)中扮演着重要角色。当两台主机交换文件的时候,它们必须保证它们是在用苹果和苹果相比,用橘子和橘子相比。验证解析器将文件和 DTD 进行匹配使得这种比较成为可能。

在第 25 章“EDI 和 XML”中,我们将详细讨论 EDI。你将会看到,在应用程序之间会发生复杂的消息交换,在这些消息之中有一条是“你是否接受这样或者这种类型的文件?”。

XML 的 DTD 定义了文件类型,这样,交易的双方就可以知道它们是否在处理相同类型的文件了。

12.3.4 用 DOM 来使用 DTD

为了提供某种视图,许多有趣的 XML 实现需要对 XML 文件进行检索和排序。这是通过用脚本或者程序语言来遍历文件树来实现的。如果用户代理可以确定文件的结构,它的工作就简单多了。比如,如果它知道元素 first-name 总是元素 name 的第一个孩子,它就会在它需要得到元素 first-name 的内容的时候,去取元素 name 的 firstChild 的内容。在第 15 章“利用文件对象模型(DOM)”中,我们将详细讨论这个问题。

12.4 深入 XML 解析器

如前所述,XML 解析器可以分为两种:标准的解析器仅仅检查文件是否是结构良好的,而验证的解析器检查文件是否符合 DTD。

解析器也可以分成两种类型:生成解析树的和仅仅将文件作为平面结构进行解析的。第一种类型的解析器最为普通。尽管第一种解析器没有第二种解析器快,不过它的功能更为丰富。更重要的是,第一种解析器是可扩展的。

历史简介:下面是关于可扩展性的重要性认识的简要历史

Netscape 的 HTML 浏览器是基于第二种解析器的,它把文件当作平面结构进行解析。在 HTML 的初期,这样是完全没有问题的。然而,当出现样式表和高级脚本语言的时候,问题出现了。Netscape 浏览器在实现样式表和文件对象模型以及回送内容的时候遇到了巨大的困难。事实上,在 Netscape Navigator 的后续版本中平面被打破了。Netscape 不得不从头开始重新设计整个浏览器以便建立一个解析树,而且还不得不在竞争越来越激烈,市场份额不断减少的情况下完成这一切。

结果就是 Gecko 系列浏览器,截止到发稿时,它还尚未发布。这种浏览器将更为强大,不过你也知道故事的结局了。Netscape 是否一定要花费大量的资源来重新开始是另人怀疑的,但它仍将是重要的浏览器厂商。教训就是如果你打算将浏览器嵌入你的软件,那么一定要确认该浏览器建立了解析树。

12.4.1 解析树

在讨论浏览器是如何工作的之前,你必须理解解析树。在面向对象编程中,所有 XML 文件都被当作一系列具有某种属性的对象。文件是一个对象,所有独立的元素都是对象,注释是对象,所有的文本串也都是对象。解析树是一个结构,它不但作为这些文件对象的框架而且还要显示它们之间的关系。

关系语法

类似的语法被用来定义对象之间的关系。每一个对象(除了文件对象)都有一个父亲,而许多对象都有后代或者后代对象。具有相同父亲的对象被称做兄弟。

注意:在 XML 文件中有一些其他类型的对象,这些对象将在第 15 章中介绍。为了

简化问题,这里仅介绍最基本的对象:文件对象本身、文本对象、元素对象和注释对象。

节点语法

鉴于类似的语法描述了解析树的关系,我们借用生物学的语言来描述这棵树。我们将派生了所有其他元素的元素成为根(root)元素。而将树中的每一个独立对象称为节点,节点相当于这棵树的枝杈。我们将树中没有子节点的对象称为叶子。

现在我们就可以将两种语法合并到一起,讨论兄弟节点、子节点、父节点了。

清单 12.2 是一个说明这些关系的简单的 XML 文件。

清单 12.2 简单的 XML 文件

```
<? xml version="1.0"? >
<! -demonstration of a parse tree-->
<xdoc>
  <greeting>Hello <emph>parsed</emph>XML!! </greeting>
  <applause type="sustained"/>
</xdoc>
```

图 12.4 以人们可以理解的方式显示了这个文件的解析树。当然,解析器将这个解析树保存在一系列数组中。每一个信息点都是一个节点。

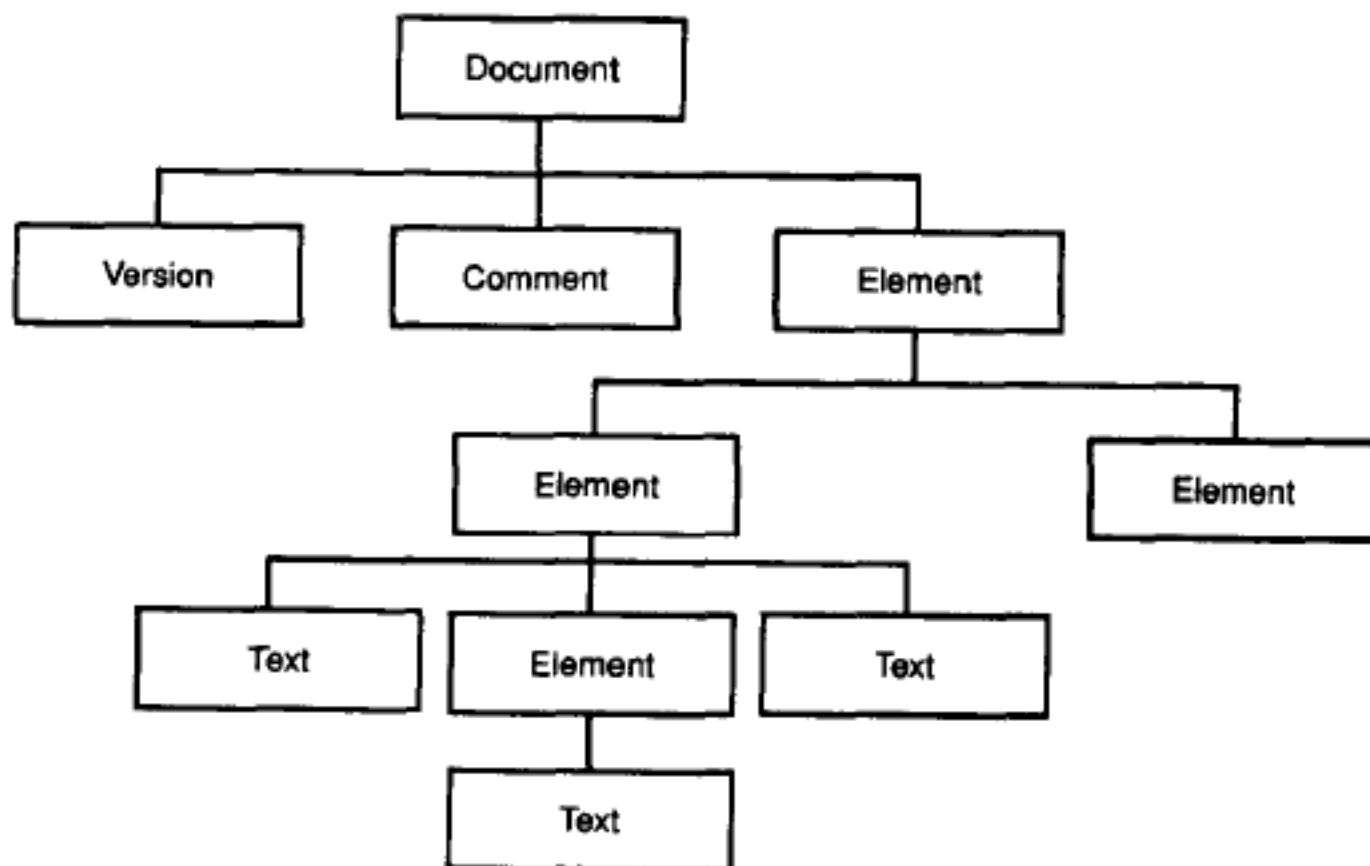


图 12.4 简单的 XML 文件树

总的来说,这个 XML 文件中有 10 个节点或者说对象,每一个对象都作为解析树的节点保存在内存中的解析树节点数组里。表 12.1 列出了这些对象可能有的属性。

表 12.1 节点和属性

| 树节点 | 属性 |
|--------|--------------------|
| 节点类型 | 注释、元素版本声明等等 |
| 节点名 | 对于元素来说,它是元素名称 |
| 节点值 | 对于元素来说,它是元素的内容 |
| 父节点 | 是父节点的节点 |
| 是否有子节点 | 确定节点是否有孩子的布尔属性 |
| 子节点表 | 元素孩子的详细列表,可能是另一个数组 |

注意:这些属性都是假想的,不过它们和大多数解析器所使用的属性类似。我们特别将这些名字都取得和 W3C DOM 所用的十分相似。我们将要在第 15 章中详细讨论 W3C DOM。

这些属性使得解析器可以重建整个文件。形成这样的列表的前提就是该文件必须是结构良好的。下面我们就一步一步地看一看解析器是如何解决创建解析树和检查文件是否是结构良好的这两个问题的。

12.4.2 结构良好的解析器

这一部分将使用伪码和描述来看一看清单 12.3 中列出的文件的处理过程。

解析器的伪码

下面是解析器操作的一系列伪码:

```
Make an object array for the children of the document
Read the properties of each node into this array
When you come across the root element make an array for the root element
Do this by calling a function "make_element_array[element name]"
Comment: "make_element_array[]" reads the properties of the children
of the element into an array.
If you across another non-empty element call yourself again.
(Comment: This is a process of recursion)
Carry on reading the properties until you come to the
closing tag for the element.
When you come to the closing tag pass the array you have built to
the "child list" property of the element that called you.
When you come to the closing tag for the root element you should be finished.
However check to make sure there are no more elements.
If there are more elements then call an error. If not stop.
The document is well-formed.
```

中文意思如下(译者补注):

建立用来保存文件的子节点的数组。

将每一个节点的属性读入并保存在这个数组中。

当遇到根元素的时候,通过调用“make_element_array[element name]”来为根元素建立一个数组。

当遇到其他非空元素的时候,再次调用自身(注:这是一个递归过程)。

继续读取属性直到遇到元素的结束标记。

当遇到结束标记的时候将已经建立的数组加入到调用本过程的元素的“子节点列表”属性中。

当遇到根元素的结束标记的时候,过程结束。不过还要检查文件中没有其他元素了。

如果文件中还有其他元素,那么就调用错误处理。否则结束。

这个文件是结构良好的。

描述

下面叙述了典型的解析器构建解析树的过程:

解析器开始从头读取文件。通过调用 `mainarray()`,它为文件的第一个节点建立一个数组(主数组没有出现在代码中,不过它很可能是解析器设计者起的一个用来保存主解析树的名字)。数组中的每一项都是一个对象,这些对象的属性是从前一节中枚举得来的。

当解析器遇到非空的元素的时候,它开始解析这个元素并建立一个数组来保存该元素的内容。解析器正在解析的对象暂时挂起,直到它解析完这个新的元素及其内容。所有的程序员都会看出,这是一个递归的过程。

解析器遇到的第一个节点(在读取根文件之后)就是版本声明。

解析器首先检查该声明的语法是否正确。如果正确,解析器就将版本声明读取到主数组中;如果解析器发现该声明的语法有错误,它就停止解析并且产生一个错误。解析器仅仅是报告这个简单的错误并且停止文件的处理,因为它不必试图去修订这个文件或是理解文件的一部分。只有这样,才能保证解析器的精巧(lean and mean)。

接下来的节点是注释。同样,解析器检查合法性并将注释读入主数组。

第三个节点是一个非空节点。解析器是通过它的开始标记了解到这一点的。由于解析器知道非空元素可能有自己的子孙,所以它开始一个递归的过程。只要解析器遇到一个元素的开始标记,它就建立一个新的数组。这个数组就成了该元素的属性。为了便于讨论,我们称之为 `xdocarray()`。

在 `xdocarray()` 中,解析器遇到的第一个节点是另一个元素,于是建立了又一个数组: `greetingarray()`。

在 `greetingarray()` 中,解析器遇到的第一个节点是文本节点。这个文本节点的属性被读入数组。

解析器遇到的第二个节点是另一个非空元素——`emph`,为了处理它,又建立一个新数组。

`emph` 是一个单值的文本节点。在解析器将这个文本节点属性读入数组之后,它继续执行。接下来解析器遇到的是 `emph` 的结束标记(这是它遇到的第一个结束标记),于是解析器知道元素 `emph` 已经处理完了,它就返回去解析元素节点 `greeting`。

节点 `greeting` 有另外一个文本节点被读入 `greetingarray()`。当解析器遇到 `greeting` 的结束标记的时候,它就返回去解析节点 `xdoc`。

元素 `xdoc` 的下一行仍然是一个元素——`applause`。这个元素是一个空元素,因此不需要处理它。它的属性也被读入 `xdoc` 数组。

现在解析器遇到了元素 `xdoc` 的结束标记,于是它知道元素 `xdoc` 已经处理完了。如果解析器在此之后又发现了任何其他元素,它就会产生一个错误。

如果由于某种原因导致解析器遇到一个不能和开始标记对应的结束标记的话,解析器也会产生错误。

图 12.5 显示了节点解析的顺序。

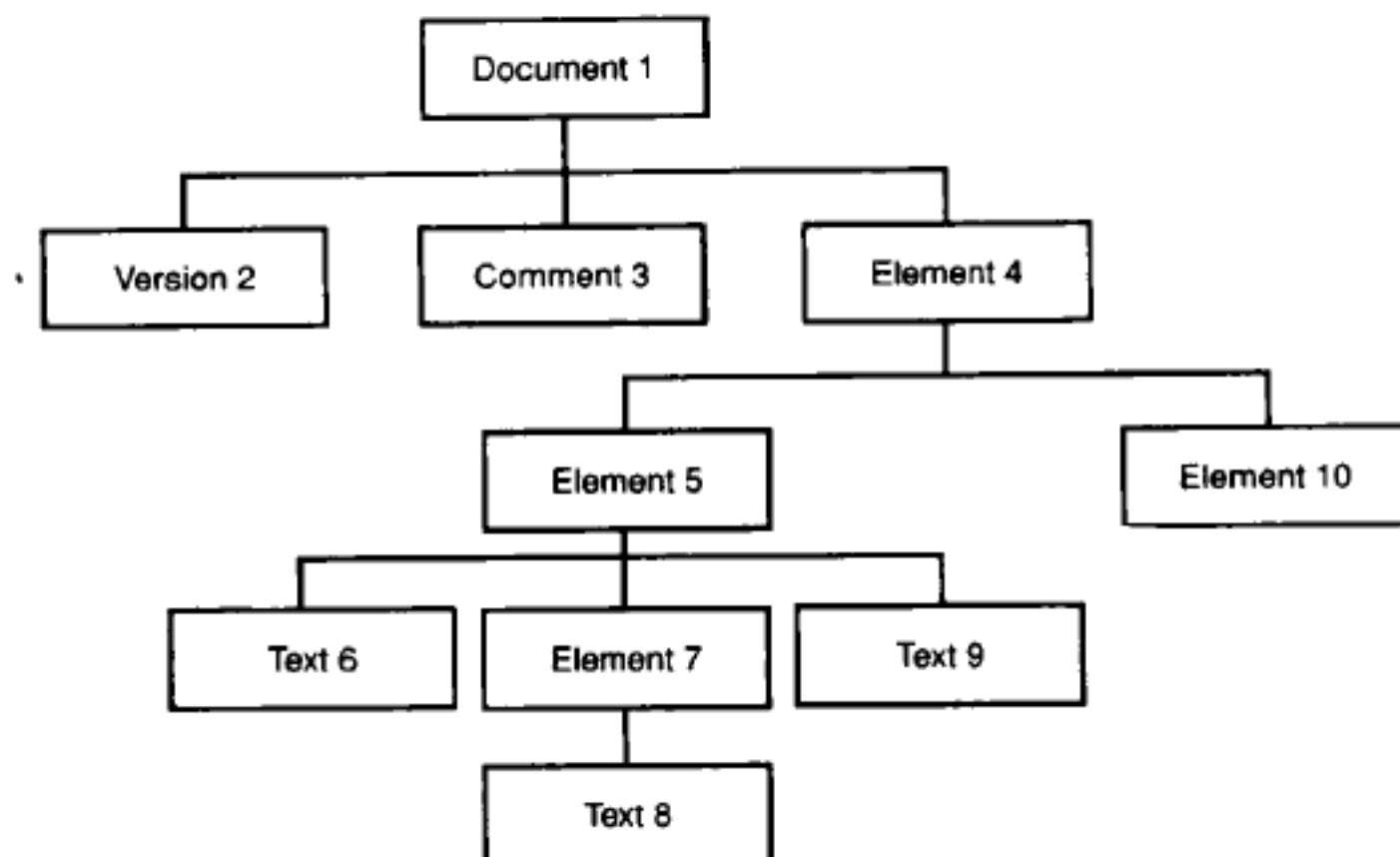


图 12.5 递归处理的顺序

尽管这个过程看起来十分复杂,不过程序实现却十分简单。如前所述,整个过程中只要出现错误处理就立即终止而无须恢复。

在这个过程结束的时候,解析器就有了整个这一系列对象数组。这些对象数组可以交给一个处理器来按照它的需要设置样式或者显示。

图 12.6 显示了 IE5 是如何显示解析树的。



图 12.6 在 IE5 中显示的解析树

在很多情况下,仅仅是结构良好的文件就足够了。然而,如果解析器要验证一个文件的有效性,那么它还要做很多工作。

12.4.3 验证解析器

为 XML 文件生成解析树比验证 XML 文件的有效性所做的工作要少得多。第一个

Microsoft 的 C 解析器仅仅检查文件是否为结构良好的并且构建了解析树,它的大小只有大约 45K。相比之下,IE5 内建的验证解析器(MSXML.dll)几乎有 400K。几乎是前者的十倍!

验证文件有效性的第一步就是建立一个解析树并且检查它是否是结构良好的。接下来解析树中的每一个元素都必须和 DTD 进行比较。清单 12.3 是前一个例子的 DTD。

清单 12.3 解析树的例子

```
<! -DTD for parsetree3.xml-->
<! ELEMENT xdoc (greeting|applause)+ >
<! ELEMENT greeting (#PCDATA|emph)* >
<! ELEMENT emph (#PCDATA)>
<! ELEMENT applause EMPTY >
<! ATTLIST applause
    type CDATA #IMPLIED
>
```

在根据 DTD 解析文件的时候,解析器可以用下面的两个策略之一。或者是从 DTD 建立一个逻辑结构然后用这种逻辑结构和文件进行匹配,或者是重新解析 XML 文件然后将每一个独立的元素和 DTD 进行匹配。大多数解析器采用第一种策略。

用逻辑结构解析

下面是上述 DTD 的逻辑结构的伪码:

| | |
|------------|--|
| 'xdoc' | must be the root element |
| 'xdoc' | must contain at least one greeting or one applause |
| 'element' | can contain nothing, PCDATA or 'emph' elements |
| 'applause' | no content |
| 'applause' | can contain 'type' attribute |
| 'applause' | type attribute can be any CDATA value |

中文意思如下(译者补注):

- 'xdoc' 一定是根元素
- 'xdoc' 必须包含至少一个 greeting 或者一个 applause
- 'element' 可以什么也不包含,也可以包含 PCDATA 或者'emph' 元素
- 'applause' 没有内容
- 'applause' 可以包含'type' 属性
- 'applause' type 属性可以是任何 CDATA 值

典型的,使用这种方法的解析器建立一系列布尔条件语句。比如,当它遇到元素 xdoc 的时候,解析器可能会有下面的语句。找到 XML 文件中的每一个 xdoc 元素:

```
For each xdoc element:
    Make child list for xdoc
    If xdoc child list contains greeting OR applause then there is NO ERROR
    If xdoc child list contains PCDATA then throw an ERROR
    If xdoc child list contains element other than greeting or applause
```

then throw an ERROR

中文意思如下(译者补注):

对于每一个 xdoc 元素:

为 xdoc 建立子节点列表

如果 xdoc 的子节点列表中含有 greeting 或者 applause, 那么就没有错误

如果 xdoc 的子节点列表中含有 PCDATA, 那么就产生一个错误

如果 xdoc 的子节点列表中含有不是 greeting 或者 applause 的元素, 那么就产生一个错误

当然实际的代码比这复杂得多。

实际上, 在解析器开始解析 XML 文件之前, 它必须首先检查 DTD 以确认 DTD 是否符合语法并且内部一致。换句话说, 如果 DTD 要求 xdoc 含有一个 greeting 元素, 那么元素 greeting 就必须被声明。

在你编写 DTD 的时候, 你需要理解这些内容。通常, 你要编写 DTD 和一个用来验证这个 DTD 的文件, 然后你在一个验证解析器中运行它。解析器每次只能显示一个错误。在你处理下一个错误之前, 你必须解决这个问题。

12.5 解析 XML 的两种方案的比较

我所描述的解析 XML 的方案都包括建立解析树, 这是几乎所有验证解析器使用的方法。正如你所见到的, 这种方法将文件当作一系列具有文本内容的相互关联的对象来处理。在第二种方案中, 解析器将文件看作是一个平面的结构或者是文本文件。

我们这就来用这两种方案来解析同一文件, 然后比较它们的异同。

12.5.1 将文件作为树来解析

本节介绍了创建文件树的解析器可能的处理过程。我们还是使用伪码(当然, 实际处理比这复杂一些, 但是这也可以说明主要的一些原则)。

下面是这个文件:

```
<!--a simple example -->
<xdoc>
  <greeting>Hello World! </greeting>
</xdoc>
```

接下来生成一个具有下列属性的 nodeobject 数据类型:

```
nodeobject  [name]
            [type]
            [value]
            [parentname]
            [childlist() array]
```

文件具有下列基本功能, 伪码如下:

```
function getchildren(nodeobject, docfragstr)
```



```

//comment: docfragstr is the fragment of the document
// that is passed to the function.
//注: docfragstr 是文件的框架,作为参数传给这个函数(译者补注)
do until there are no more tags in docfragstr
    find a "<"
        if there is any text before the "<" this is a text node. Add it to the child list. Shorten string
        if this belongs to a comment make a node object and add to the child list. Shorten string
        if this belongs to a closed element make a node object and add to the child list. Shorten
        string
        if this belongs to a closing tag which matches me stop
        if this belongs to a closing tag that doesn't match me THROW AN ERROR
        if this belongs to an opening tag make a node object, add to child list and call this function.
        Shorten string
loop till done

```

我将整个文件都提交给上述过程。这是循环递归调用这个函数的开始。

在文件中的第一个“<”属于一个注释,所以解析器建立一个注释节点对象并把它加入到文件节点对象的子节点列表数组中。

这时文件节点对象如下:

```

nodeobject    [# document]
    [4]
    [null]
    [null]
    [childlist(0)]
        [# comment]
        [3]
        [' A simple example' ]
        [# document]
        [false]      //no parent

```

文件字符串被适当地简化:

```

<xdoc>
    <greeting>Hello World! </greeting>
</xdoc>

```

下一个“<”属于一个元素。一个元素对象被生成并加入到文件对象中。

这时文件节点对象如下:

```

nodeobject    [# document]
    [4]
    [null]
    [null]
    [childlist(0)]
        [# comment]
        [3]
        [' A simple example' ]
        [# document]
        [null]
    [childlist(1)]
        [xdoc]
        [1]

```

```
[null]
[# document]
[true]
```

这时 xdoc 数组被生成并且再次调用这个函数。这是它的第二次调用。第一次调用被挂起。

下面的字符串被提交给这个函数：

```
<greeting>Hello World! </greeting>
</xdoc>
```

第一个“<”属于一个 greeting 元素。

这时 xdoc 节点元素如下：

这时 xdoc 节点对象如下：

```
nodeobject  [xdoc]
  [1]
  [null]
  [# document]
  [childlist(0)]
    [greeting]
    [1]
    [null]
    [xdoc]
    [true]
```

生成一个 greeting 数组，第三次调用这个函数，这一次递交给它的字符串是：

```
hello Woeld! </greeting>
</xdoc>
```

这里有一段文字在“<”之前，所以生成一个文本节点对象。

这时 greeting 节点元素如下：

```
nodeobject  [greeting]
  [1]
  [null]
  [xdoc]
  [childlist(0)]
    [# text]
    [2]
    [Hello World!]
    [greeting]
    [false]
```

接下来的“<”属于一个结束元素，这是 greeting 的结束元素，于是函数无错返回（或者说第三次调用返回）。

这时解析器返回第二次调用，而剩下的字符串如下：

```
</xdoc>
```

函数的第二次调用看到这这也是一个开始标记对应的结束标记，于是第二次调用也返回

了。

这使解析器回到了第一次调用,这是最初的文件调用。

至此已经没有任何元素了,而且也没有发现任何错误,因此这个文件是结构良好的,另外还产生了一个系列有用的数组可以传递给 XML 处理器。使用下面的方法之一,这些数组可以被提交给应用程序:

- 在 API 中作为对象属性。这个方法通常用在分布式的应用中,比如 Internet 应用。
- 作为应用程序可以引用的类。在解析器作为对象或类嵌入应用程序之中的情况下使用这种方法。
- 应用程序可以直接访问这些数组。尽管这种方法提高了应用程序的速度,但是它不符合现代面向对象编程的概念,因此建议不要使用这种方法。

无论这个解析树是如何递交给用户代理的,这些数组都可以被用来创建流式对象,都可以被设置不同的样式,而且可以通过文件对象模型来操纵。

注意:实质上,流式的树就是那些为表示而格式化了的节点。

同样还要注意如果解析器是验证解析器,那么有了这些数组,根据 DTD 来检查文件的有效性就简单得多了。这是因为与平面的文件相比,数组的访问和操纵都更为容易。

12.5.2 将文件作为平面数据结构来解析

如果仅仅是检查文件是否是结构良好的,那么将文件作为平面结构来解析就相对容易一些。下面我们就来看一看这种机制。

注意:在 XML 最早出现的时候,还没有任何可用的解析器。我不得不自己来编写它。我大概只用了一天就编写了一个平面解析器,它也能很好地工作。但是当我试图将它变成验证解析器的时候,我马上遇到了很多问题。作为练习,我编写了一个建立树结构的标准解析器。我大概全力编了两个月。幸运的是,现在已经有了一些非常出色的(而且是免费的)解析器,我也就不必自己编写验证解析器了。

下面还是那个例子:

```
<!--a simple example-->
<xdoc>
  <greeting>Hello World! </greeting>
</xdoc>
```

下面是一个简单的解析器是如何以一种平面数据结构来解析文件的。

首先,解析器去掉所有的注释和 CDATA 段(实际上在这个例子中并没有 CDATA 段),因为它们不会影响文件的有效性或者结构良好与否。下面就是剩下的文件:

```
<xdoc>
  <greeting>Hello World! </greeting>
</xdoc>
```

接下来解析器检查开始标记和结束标记是否匹配:

First tag is <xdoc>

Is last tag </xdoc>?

中文意思为(译者补注):

第一个标记是<xdoc>

最后一个标记是不是</xdoc>?

如果答案是肯定的,就去掉这些标记,否则产生一个错误。

注意:这并不意味着下面的情况是合法的:

```
<xdoc></xdoc>
```

```
<xdoc></xdoc>
```

在解析器去掉标记之后,会剩下:

```
</xdoc><xdoc>
```

这显然是不合法的。

这时解析器得到下面的字符串:

```
<greeting>Hello World! </greeting>
```

解析器注意到下一个标记是 greeting 标记,于是它就找下一个具有相同名字的结束标记。换句话说,就是查找</greeting>。接下来解析器要回答下面的问题:

is there a <greeting> tag in between?

中文意思(译者补注):

在此期间是否又出现了<greeting>标记?

如果答案是没有,它就将这两个 greeting 标记去掉:

if no then strip greeting tags

中文意思(译者补注):

如果没有则去掉 greeting 标记

答案是没有,于是去掉这两个 greeting 标记而解析器就得到了下面的字符串:

```
Hello World!
```

由于没有其他标记了,所以解析的结果是:这个文件是结构良好的。

想一想解析器是如何处理下面的字符串(其中含有嵌套的 greeting 标记):

```
<greeting>Hello World! <greeting>Hello XML! </greeting></greeting>
```

next tag is greeting

look for next </greeting> tag

is there a <greeting> tag in between?

中文意思是(译者补注):

下一个标记是 greeting

查找下一个</greeting>标记

在此之间有没有<greeting>标记?

这一次答案是有,于是解析器就对第二个 greeting 标记重复上述问题。在它和这个 greeting 结束标记之间还有没有其他 greeting 标记?答案是没有,于是将第二个标记和这个结束标记去掉。换句话说,当具有相同名字的标记发生嵌套的时候,解析器去掉嵌套在最里边的一对标记。

这时解析器遇到的字符串变成了:

```
<greeting>Hello World! Hello XML! </greeting>
```

重复相同的处理,解析器就得到了一个没有标记的字符串:

```
Hello World! Hello XML!
```

因此这个文件是结构良好的。

还有其他一些小问题,但是它们不必在这里讨论。最基本的就是如果没有错误产生,那么文件就是结构良好的。除此之外,没有建立任何数组也没有建立解析树。这个文件被从头到尾地读了一遍,没有发现任何错误。

如果解析器还要验证文件的有效性,它就必须对它遇到的每一个标记重复更为复杂的处理过程。解析器将不得不根据 DTD 检查每一个标记以及所有内容。它也可以这么做,但是这样做的主要问题是你将无法对文件使用 DOM 或是样式表。解析器将不是重复地使用数组,而将要一而再再而三地重复整个过程。

总之,如果你仅仅要检查文件的结构良好性,平面的文件解析器也还不错,不过要将它用在全功能的 XML 处理器中是极其困难的。如果你想要将解析器用在处理器中,你最好还是使用一个解析树型的解析器。

12.6 作为对象的解析器

本质上,在应用程序中使用解析器有两种方式:

- 检查 XML 文件是不是结构良好的。
- 作为处理器的一部分。

在第二种情况下(这也是最常见的情况),解析器可以被当作对象处理。对于基于 Java 的解析器来说,这意味着解析器将作为一个类出现在 Java 应用程序之中。对于基于 C 的解析器来说,既可以使用源代码直接访问数组和其他的一些变量,也可以将解析器作为具有自己的方法和属性的类来处理。

对于 Windows 应用程序来说还有另外一种方法,就是将解析器编译成 DLL,然后在 COM 中使用它。实际上第 14 章“用 C++ 解析 XML”中的所有 C 解析器都有相应的 DLL 支持(比如 msxml.dll)。

12.7 总 结

本章讨论了 XML 解析和 XML 解析器的实质。本章还讨论了一些解析器所要作出的选

择,是作为验证解析器还是作为非验证解析器。

XML 对解析器的要求中最本质的一点就是在遇到不是结构良好的文件时,解析器必须以一种严格的方式工作,而且必须停止以正常的方式解析该文件。

下面是两个基本的结果:

1. 这迫使 XML 作者必须编写正确的文件。
2. 这使得解析器的工作简单化,因为它不必理解那些不好的文件。

你还看到,有两种基本的解析器类型,建立文件树的和将文件当作平面文件解析的。我们还讨论了在什么情况下选择哪种解析器及其原因。在接下来的两章里,我们将要介绍特定的几个解析器,那里,将会介绍 Java 和 C 解析器的使用。

第 13 章 用 Java 解析 XML

从 XML 诞生起,Java 程序员就一直是 XML 开发集体中的先锋。许多开发者认为 Java 是 XML 的理想开发工具。这两种技术相互补充:XML 具有数据的可移植性而 Java 具有软件的可移植性。另外,还有很多 Java 特别适合同 XML 一起工作的特征:

- 使用 Java 则程序员产量很高。通常,你用 Java 来做一个新技术的健壮的产品,可以比用 C 快两倍。
- Java 适用于 Web 编程。无论是在客户端还是在服务器端都是如此,而 XML 的主要目标就是下一代的 Web 技术。
- Java 对 Unicode 强有力的支持,使它非常适合 XML 对字符集的要求。所以在不讲英语的地方,不会发生任何不便。

通过使用众多出色的为 Java 程序员提供的 XML 包,我们可以体会到这些好处。Java 的免费 XML 工具比其他语言的都多,而且通常质量都很好。如果你正在寻找一个基本的解析级的工具,这些 Java 工具可以为你提供最多的选择。基于同样的原因,Java 的高级工具也很丰富。

本章覆盖了很多领域。首先我们将讨论解析器的基本结构,然后大致介绍一下最通用的几个包,在此之后,还要对每一个包进行仔细的分析。尽管所有 SAX 解析器看上去都差不多,但是 DOM 解析器则完全不同。你可能会在掌握了第 15 章“使用文件对象模型(DOM)”和第 16 章“在 Java 中使用 SAX API”的内容后,又重温本章的后一部分。

13.1 Java 的 XML 解析器库剖析

提到 XML 解析器,人们把它当作是一个东西,但是实际上并非如此。实际上,解析器有很多 API,大多数 API 不仅仅是解析,但是都被认为是解析器 API。有些 API 是应用程序的组件提供的。简而言之,XML 解析器库的含义很大程度上决定于谁在讨论它。

这种库的核心是被 XML 规范称为 XML 处理器的东西,而它的上面是其他 API 层。从 XML 标准的角度讲,这些层都是应用程序的一部分,尽管这些层大多都是库代码实现的,但是这些层可能支持一些特征,诸如 XML 模式验证、XSLT 转换引擎、查询和连接引擎、servlet 或 applet 集成以及本书讨论的其他特征,还有应用相关的代码模块。应用程序可以结合使用这些工具,如 DTD 或模式编译器。

本章仅讨论那些除了用在嵌入式系统之外,所有 XML 开发者用 Java 时都可用的一个很小的 API 子集。如果你没有看到你想要的特征,那么这个特征很可能属于其他的层。目前已经开发出许多这样的层。

这个核心 API 的集合主要针对那些 Sun Microsystem 公司声明要在它的 XML Standard Extension for Java 中实现的功能。这样的标准扩展最终很可能成为 Java 平台

的核心运行系统的一部分。目前,这些 API 中的一部分仍在定义中。

13.1.1 核心功能

理解 XML 核心 API 的最重要一点就是知道我们能够用这些 API 做些什么。我们需要能够编写全过程的处理组件,这些组件能够解析 XML 文本、处理内存中的数据并且生成 XML 文件。任何一个应用程序组件本身都仅仅是处理组件流水线上上的一个部件。如果这些组件的任何一部分不能正常工作,应用程序就不得不编写自己的代码来解决问题。所有这些功能都是必不可少的。

13.1.2 核心 API

多种 XML 软件包都支持下面的这些功能:

- 标准的 Simple API to XML(SAX),这是解析 XML 内容最底层的方法。它是一个事件驱动的 API,而且是在大多数厂商还对 XML 的 Java API 不太感兴趣的时候,以一种统一的开放的方式开发的。
- 标准的文件对象模型(DOM)API。这组 API 是在内存中表示 XML 内容的一个可选的方式。它是一个数据结构,这个结构可以表示 XML 内容的逻辑树结构。这组 API 是 W3C 开发的。
- 一些 API 用来访问那些 W3C 没有指定为 DOM 的一部分的基本区域。包括实际取得 DOM Document 以表示 XML 内容的方法,以及处理 XML 名字空间信息的方法。
- 打印 API。通常它们是基于 DOM 的。

后面的几章将重点介绍标准的可移植 API:SAX 和 DOM,这些 API 都是 Java 标准 XML 扩展的一部分。

本章主要使你可以编写使用这些专用 API 的一个最小的应用程序。本章将会用这些专用 API 取得 DOM 然后打印它,因为你需要用它来编写程序。请注意,如果定义了标准 API,你应当用标准 API 来代替这些专用 API。

最新的 Java 语言关于标准 XML API 的信息,请参阅 <http://java.sun.com/products/xml>。

13.1.3 这个核心没有揭示 DTD

经验告诉我们,几乎所有的应用程序都仅仅处理了一小部分信息——这些信息基本上都可以从 SAX 取得。在某些情况下,生成 HTML 的应用也需要访问注释,并且用注释来隐藏诸如 CSS 样式表以及 JavaScript 代码之类的内嵌内容。DOM 也提供了这样的信息(新版的 SAX 也可以)。上述 API 已经得到广泛的使用。

然而,有一类特别的应用需要比这些 API 所能提供的信息更多的信息。比如,一个只能创建有效的 XML 文件的 XML 编辑器,这样的应用需要访问 DTD 中元素和属性的声明,以及解析实体的声明。SAX 和 DOM 都不能取得这些信息,不过在新改版的 SAX 中应该可以取得它们。

本章中的大部分解析器都可以取得 DTD 信息,通常是在 SAX 层下,我们就不更多介绍

了。这是因为大多数工业档案中的计划是改造 XML Schema 系统,使之可以访问有效文件的结构,以及修正验证部分,使之可以对数据进行更多的绑定。这样的系统不需要 DTD 语法。大多数厂商在基于模式的解决方案上投资,而不是进一步开发 DTD 信息,这将在本书所讨论的 API 的上面实现的,它将在 DTD 信息领域中留下一个功能的鸿沟。

13.2 市面上的工具

尽管有很多 XML 解析包可以选择,但是恐怕没有一个单独的 XML 解析包可以完全的适合你的需求。你可能不得不在众多品牌中权衡作出选择。为了帮助你理解如何进行权衡,本节将要介绍一些 XML 解析器,它们特别值得研究。

所有这些解析器都支持 SAX API,而且它们中的大多数都支持 DOM 和其他特征。

你可以发现研究多样的 XML 资源以取得最新的信息是有用的,因为可能会有新的包可以使用。下面是一些有用的站点:

- <http://www.oasis.open.org/cover/>——这个页面有关于 SGML 和 XML 各个方面的丰富资料,而且经常更新。它覆盖了开发工具,但是对于 Java 工具,你还得好好找一找。
- <http://www.xml.com/>——这个站点覆盖了多样的 XML 工具,而且包括解析器的性能和一致性的评论。
- <http://wdvl.com/>——Web 开发者的虚拟图书馆,有关于 Web 客户端应用的丰富资源。

所有这些站点都主要介绍全功能的包,除了用于特殊用途的少数 SAX 解析器以外,它们既支持 SAX 又支持 DOM。

在本章中除非特别说明,目前我们使用的包都是商业软件提供的。另外,除非特别说明,软件的改版要按照开放源代码许可策略来发布。这种策略可以保证不断发展的软件可用性和你修改基本构件错误的能力。当然,和你要求的软件相关的实际许可最终将决定这些问题,这里就不赘述了。

性能将不在本章中讨论。实现发展得很快,解析速度的差别可能达到 20%(SAX 是这样,DOM 还要多),如果你换到其他的 Java 虚拟机或者 CPU 上,这种区别很容易消失。另外,所有这些解析器现在都“足够快”。在大多数系统中,一般解析器的速度不成问题。

在本章中有时会提到的问题是对国际化的字符编码的支持。Java 使这个问题变得简单,因为它在 SDK 中提供了这样的支持。除非解析器没有使用 JDK 的这种功能或是出现了其他问题,否则我们就不讨论编码的问题了。假设只要 JDK 支持这种编码,每一个 XML 解析器都能处理你给它的任何编码。对于日语的 Shift_JIS 和 EUC-JP,中文的 Big5 和 GB-2312,EBDIC 以及其他超过 100 种的编码来说,JDK 1.1 开始都支持,当然这些包也需要至少是 JDK 1.1 的支持。

验证还是非验证?

验证解析器一度被认为肯定比非验证解析器慢很多。早期的实现的确如此,这是因为最初的几个用 Java 编写的验证 XML 处理器都比当时的非验证解析器慢 20 到

30 倍。

幸运的是,这仅仅是因为这些实现没有经过优化。在 Sun 发布了它比当时所有的其他处理器(包括非验证的处理器)都快的验证 XML 处理器以后,那些很慢的实现就渐渐地被取代了。

现在验证的代价大概是 10% 到 15%,无论是大小还是空间。(这个数值还可能更小。)选择是否使用验证解析器主要决定于系统体系结构。

13.2.1 Sun 的 Java Project X package

Sun 的包提供了核心 XML 工具:DOM、SAX 以及 XML 名字空间 API 的高质量的实现。它不但同时提供了验证解析器和非验证解析器,而且它还是最遵守协议的一个,这可以通过对 SAX、DOM 和 XML 规范的单独测试证实。解析器诊断也比较有用,而且你可以将诊断信息翻译成任何非英语的语言,只要为这些消息加上适当的新资源就可以了。

这个 API 的扩展主要支持前面描述的核心 XML 功能。它支持 XML 名字空间而且和现行的 DOM Level 2 建议紧密一致,还可以打印 XML 内容。它还有 SAX 扩展支持编辑和 DTD 回调,它们遵守早期的 SAX 2.0 API 建议。

Sun 致力于将这个实现变成未来用于 Java 的 XML 标准扩展的参考实现。这将包括 API 的修改,这个版本是实验性的,这些 API Sun 也不支持。当它成为参考实现的时候,会用一个公共源码许可来发布它,这样就可以允许代码修改后进行发布了(这个许可证不是开放源代码的,但是它和先行的开放源代码许可证非常接近)。

目前,当前版本是 1999 年 5 月 21 日发布的 Technology Release 2(TR2)。当前的许可证还不是开放源代码的,因为它不允许修改源代码以后再发布。

- <http://java.sun.com/xml/>——检索 Java Project X 包,包括源代码、例子和文件。

13.2.2 IBM 的 XML4J v2 包

IBM 在 XML 和 Java 上做了很多工作。在它的 alphaWorks 站点上有很多面向 XML 的软件包可以使用。这个站点上有一大堆 alpha 版的软件(有时也有一些 better 版的),它们主要是为了取得尽早的反馈。举例说来,有一些包将 IBM 对 XML 的支持集成到 IBM 的 VisualAge Java 图形开发环境中(在第 30 章“了解 IBM 的 XML 工具包”中,你可以找到关于 alphaWorks 工具更多的信息)。

XML4J 包是通过 alphaWorks 提供的 XML 包之一。它提供了验证解析器和非验证解析器,而且还具有对 DOM 的支持以及丰富的专用 API 来访问 DTD 并实现其他功能(许多这些 API 都是内部 API,它们既没有出现在文件中,也不打算被应用程序使用)。你可以将诊断信息翻译成任何非英语的语言,只要为这些消息加上适当的新资源就可以了。

这个包只提供了对 XML 名字空间最小限度的支持。它可以在解析过程中检查名字空间声明,但是对 DOM 来说就仅此而已了,它不能直接得到元素和属性的名字空间。它也没有内建的打印 XML 内容的功能。

目前,这些解析器提供的 XML 和 SAX 一致性水平低得惊人,测试上仅能够达到中下级水平。DOM 支持更一致一些,但是它也在为一些地方和标准不一致而造成应用程序移植方面的问题而苦恼。

尽管这个解析器可以接受JDK支持的所有编码(大约150种),但是它的缺省配置只允许使用这些编码中的有限几种。

目前最新的版本是1999年8月30日发布的XML4J 2.0.15。和所有alphaWorks软件一样,注册后可以用于商业用途,但是不能得到正式支持。许可也不是开放源代码的,因为还不允许修改源代码后再发布。

- <http://www.alphaworks.ibm.com/>——检索XML4J包,包括源代码和相当丰富的文件。

13.2.3 Oracle的v2 XML包

Oracle已经修订了它原来的XML解析器包。它有验证解析器和非验证解析器的高质量的实现,在这种解析器中它位居前茅。除了对DOM的支持,它还提供了一个XSLT模块。这个实现完成后,它许诺会作得更好。

就其他前面提到的核心功能来说,这个包提供了名字空间的支持和XML打印。如果没有这些功能也没有什么,主要是它对XSLT的支持适应了最近的工作草案。

目前的版本是1999年9月9日发布的2.0.2版。这是一个试验性的版本,尚不允许商业使用。它不是源代码开放的,没有提供源代码。

- <http://www.oracle.com/>——检索Oracle XML解析器。它没有提供源代码但是提供了很多例子以及文件。

13.2.4 DataChannel的XJParser包

DataChannel和Microsoft合作提供了这个包,它致力于提供Internet Explorer 5.0 Web浏览器中的XML包的功能。它包括对验证解析器和对非验证解析器的支持,以及DOM和其他核心级的API。

注意:你可能还找到了一些关于Microsoft称做MSXML的早期XML Java包的资料。这个软件已经被废弃了,你不应当使用它,尽管最近它变得可用;Microsoft现在将MSXML和新版的Java虚拟机(版本5.00.3186)捆绑在一起,但是没有修改错误。名称“MSXML”现在更经常地在IE5 COM解析器中使用。目前,XJParser包是Microsoft进入Java/XML领域的敲门砖。应当使用它而不是较早的MSXML包。

在核心功能之外,这个包还包括早期版本的XQL和XSLT功能以及Microsoft最新版本的模式功能和XML Data Reduced。这些都没有提供标准的W3C功能。这个包主要是为了在Java平台上支持Microsoft的功能。如果你正是需要这些,那么这个包可能就是你需要的。目前,这个包是否会升级到和W3C标准完全一致,尚不明朗。

如果你考虑把这个包用做SAX解析器,你应当考虑是否还有更合适的版本。目前的版本在一致性测试中排在后面,有很多和其他包大量不匹配的明显问题。目前,DataChannel还没有对这个问题(以及类似问题)作出答复。

这个包的当前版本还没有版本号,但是它是1999年4月15日发布的。如果你进行了增值,就可以商业发布,但是它不是开放源代码的(它不允许修改源代码以后再发布)。

- <http://www.datachannel.com/>——下载 DCXJP 包,包括源代码和文件。

13.2.5 Microstar 的 Aelfred 解析器

Aelfred 解析器,是用一位倡导文学的英国国王的名字命名的。特别值得一提的是它非常精巧。不到 30K(包括 SAX 和 SAX 驱动器——很少没有 SAX),它是最小的 XML 非验证解析器之一。

Aelfred 的设计要求包括有限地牺牲正确性和诊断的质量来取得精巧和速度。在这样的前提下,Aelfred 取得了惊人的一致性。在测试中它的一致性超过了 IBM 的解析器,而它主要的错误都是由于比较宽松地处理了文本和名字中字符的 XML 规则。这使它几乎可以位居 XML 解析器的前列。它主要和 applet 一起使用,但是显然在很多要求解析器精巧的环境中它也非常有用。

Aelfred 只包括一个非验证解析器而且不包括 DOM。它是和 SAX 非常接近的专用 API,但是它可以访问一些更低级的信息,比如解析器在解析过程中保存的 DTD 信息。它只直接支持有限几种字符编码,而且没有通过在 XML 数据开始处的 `<? xml version='1.0' encoding='...' ?>` 声明来使用 JDK 的编码支持。

目前,最新的版本是 1998 年 7 月 2 日发布的 1.2a 版(还有一个升级版,但不是 Microstar 的)。

- <http://www.microstar.com/>——搜索 Aelfred 解析器,包括源代码和少量文件。

13.2.6 James Clark 的 XP 解析器

XP 解析器是最快的且一致性最好的非验证解析器之一。在一致性上,它和 Sun 的包一起名列前矛。它是 James Clark 编写的,他是 XML 规范的技术编辑。这个解析器是在应用程序中使用(不过它的诊断信息可能对许多用户来说都是没有什么用处的)。James Clark 编写的其他 XML 应用程序中还包括一个 XSLT 引擎。

XP 解析器只是一个非验证解析器而且不支持 DOM。除了支持 SAX 之外,它有一套专用的 API 来在它的回调中传递事件对象以及输出 DTD 声明之类的低级信息。它只直接支持有限几种字符编码,而且没有通过在 XML 数据开始的 `<? xml version='1.0' encoding='...' ?>` 声明来使用 JDK 的编码支持。

这个解析器在这里列出的非验证的 SAX 解析器中是独一无二的,因为它不报告可忽略的空格。尽管 XML 规范完全允许这样(只有验证解析器才必须报告这样的信息),但是当应用程序需要很快地在数据结构中删除这些无关数据的时候,就会遇到困难。

目前,最新的版本是 1999 年 1 月 2 日发布的 XP 0.5 beta 版。尽管它是一个高质量的软件,但是它不适于那些需要所有控制和支持的开发者。

- <http://www.jclark.com/xml/>——下载 XP 解析器,包括源代码,但是没有文件和例子。

13.2.7 其他 XML 解析器

尽管上面的这些 XML 解析器是许多开发者现在的选择,但是领先的 XML 解析器难以

尽述。你可以在进行 Java 编程的时候考虑一些其他的 XML 解析器。至少有一打 SAX 解析器。

Netscape 的 Mozilla Web 浏览器

即将诞生的 Netscape 浏览器的 Java DOM 支持将提供对 W3C 标准接口的支持。这个浏览器,目前的代码名称为 Mozilla(细节请参阅 <http://www.mozilla.org>),在本章中就不仔细研究了。目前,Mozilla 的 DOM 支持正在变得可以为 Java 所用,而且它的安装 API 还可以改变(据称,这个浏览器的大部分代码将会公开,尽管 Java 支持是单独的一部分而且现在没有公开源代码)。

Microsoft 的 Internet Explorer 5.0 Web 浏览器

Microsoft 的浏览器为 Java 程序员提供了 XML 支持,但是必须通过专用的 COM API。你不能用 DOM 接口来编写可移植的代码,因为带有 DOM 的 `org.w3c.dom` 包不支持。如果你需要在 Internet Explorer 5.0 中运行标准的 Java/XML 代码,你需要使用其他的 XML 包,比如本章前面所介绍的那些包(这个浏览器不是源代码开放的;你不能修改其中的错误)。

OpenXML

这个包是一个正确的开放源代码 XML 包,它可以被集成到其他开放源代码的 XML 项目中,比如 Java 应用服务器。在这样的服务器上,XML 工具通常被用来在将信息翻译到 HTML(或 XHTML)以便输出到 Web 浏览器之前,操纵信息结构的基本构架。

这个包是惟一不提供 SAX 支持的一个,尽管它实现了 DOM 支持,包括 DOM 的 HTML 部分。SAX 支持被计划在以后的版本中实现。

这个包的当前版本是 1999 年 4 月 11 日发布的 1.0.6 版(beta 版)。1.0.8 版致力于在 XML 和 HTML 解析器中增加 SAX 支持。

- <http://www.openxml.org/>——除了发布的包以外,还有一般信息以及对源代码的访问。

Lark/Larval XML 解析器

Tim Bray,XML 规范的制定者之一,在制定 XML 规范的同时,编写了最初的 Lark 解析器。事实上,Lark 被用来从符合 XML 规范的 XML 文本中生成 HTML 文本。Lark 是一个非验证解析器。Larval 是 Lark 的扩展版本,它是一个验证解析器。关于 Lark/Larval 的一篇文章发表在 1997 年冬季的 Web Developer's Journal 上,所以你可能已经对它有所了解。

Lark 的 API 是事件驱动的解析器 API。创建 Lark 的原因之一就是测试 XML 处理器 API。在 SAX 的主页上有针对 Lark 的 SAX 驱动程序。

Lark 从没有脱离 beta 阶段,它最近一次更新是在 1998 年 1 月 5 日,和 XML 规范第一次出版的时间很接近。现在它已经不被维护了,发布的源代码不包含生成和维护它许多表的工具。允许商业使用。

- <http://www.textuality.com/Lark>——Lark 软件。
- <http://www.megginson.com/SAX>——Lark 的 SAX 驱动程序(以及一般的 SAX 信息)。

Silfide 的 SXP

French Silfide 组使用 Java 和 XML 作为基于 Web 的客户机/服务器应用环境的基础。它包括一个用 Java 写的 Web 服务器。

这个包包括一个非验证解析器,它支持 SAX API。这个解析器要比你前面看到的解析器使用更多的内存,而且在使用 SAX 访问的时候有一些一致性问题。它的本地代码就好多了。

这个包的最新版本是 1999 年 7 月 22 日的 0.88 版。

- <http://www.loria.fr/projets/XSilfide/EN/sxp/>——关于整个项目的信息,这个 XML 包是项目的一部分。

MSXML

正如在介绍 DataChannel XJParser 包时所介绍的,Microsoft 的 MSXML 包已经被废弃而且不应当再使用了。尽管它最近的版本被捆绑在 Microsoft 的 Java 虚拟机中,这仍然是不争的事实。如果你使用它,你可能会发现你正在创建的文件不符合 XML 规范,其他的 XML 包会拒绝它们。

最新的版本是 1998 年 5 月 29 日发布的 1.9 版。在 1999 年 8 月它被捆绑在 Microsoft 的 Java 虚拟机中面世。

13.3 用 Java 解析 XML 文件

如前所述,XML 解析器有两个基本接口:SAX,支持低代价、事件驱动的处理;DOM,用树形数据结构提供解析的结果以便 XML(或 HTML)使用。

我们来看一看如何用这些接口来解析 XML。在后面的章节中我们将仔细讨论如何处理解析的结果:第 15 章讲述 DOM,第 16 章讲述 SAX。本章仅仅总结最重要的非标准的特征并告诉你如何开始。你需要了解这些非标准的特征以使用 DOM,你还需要了解 DOM 或 SAX 以编写有用的程序。

为了简化问题,你将了解到如何用每一个解析器(SAX 和 DOM)完成特定的任务——在不处理任何验证错误的情况下,通过一个在名为 document URI 的变量中保存的 URI 解析一个特定的 XML 文件。你不必了解如何从输入流中解析 XML(比如你从网络消息中取得的 XML),但是所有这些包都支持这种用法。

什么是 URI?

当你处理 XML 的时候,你经常会遇到 URI 这个缩写。你可能还记得,URI 代表了统一资源标识符(Universal Resource Identifier)。这不用多解释,但是一个简单的等式可能会有帮助:URI=URL+URN。URI 是所有的 URL 加上所有的 URN。URL 代表了统一资源定位符(Universal Resource Locator)——是 <http://www> 和 <ftp://> 这样的传给 Web 浏览器的字符串。URL 被用来在 Web 上定位对象。URN 是统一资源命名(Universal Resource Name)。和 URL 一样,它标识了 Web 上的对象。但是与 URL 不同的是,它不能准确地告诉你这个对象在什么地方。

URN 看上去有点不同。比如,urn:uuid:221ffe10-ae3c-11d1-b66c-00805f8a2676 是一个基于 UUID 的 URN(可以将它看作是一组随机数,人们保证在世界范围不会重用它)。如果你使用了 URN,那么 Internet 可以很容易地识别出错误而且知道该如何解决它,但是还不能非常细致。

XML 指定使用 URI 而不是 URL,这样当使用 URN 的细节产生时,基于 XML 的系统就不必进行大的修改。在某些情况下,比如用 SAX,你可以提供给自己这些细节。

首先我们看一看 SAX。所有的 SAX 解析器都以相同的方式工作,所以它比较容易理解。然后我们介绍 DOM,这其中每个解析器都有不同的 API 来解析文件。

在我们介绍了如何解析文件之后,我们还将了解 DOM 实现提供的一些其他的重要功能,这些功能还没有被标准化,但是对于编写 XML 应用程序来所是非常重要的。

13.3.1 使用 SAX 解析器

SAX 定义了一个接口,org.xml.sax.Parser,所有 SAX 解析器都必须遵守它。它还定义了一个含有静态方法的类 org.xml.sax.helpers.ParserFactory,它使你的应用程序很容易就成为解析器。

通常有一个缺省解析器,由系统属性(org.xml.sax.parser)来控制,这个解析器可以用在你调用解析方法而又不给出参数时。许多生产环境都已经有了标准的解析器。这组 API 使用这些解析器变得更容易,而无论这些解析器是什么样的。根据 SAX 模型,解析器应当象网络驱动程序一样容易替换。你可以根据它提供的 API 以外的理由来选择解析器,比如因为它比较方便,特别正确,有很好的诊断信息,或者特别快。

如果你需要某一个解析器,只要直接将它的类名传给 makeParser 方法就可以了。这个字符串可以作为应用程序的配置信息的一部分来控制。通常你要想控制你所使用的解析器,至少要知道这个解析器是否是验证解析器。根据解析器是否验证输入数据,你的应用程序可能有很不相同的行为。比如,它可能拒绝那些在另外一种情况下它接受的文件。一些应用程序在不同的时候分别使用验证解析器和非验证解析器。

表 13.1 前面讨论的 SAX 解析器的类名

| 解析器包 | SAX 解析器类名 |
|----------------|--|
| Java Project X | com.sun.xml.parser.Parser com.sun.xml.parser.ValidatingParser |
| IBM XML4J | com.ibm.xml.parsers.SAXParser com.ibm.xml.parsers.ValidatingSAXParser |
| Oracle | oracle.xml.parser.v2.SAXParser |
| DataChannel | com.datachannel.xml.sax.SAXDriver |
| Elfred | com.microstar.xml.SAXDriver |
| XP | com.jclark.xml.sax.Driver |

注意:Oracle 和 DataChannel 两者的 SAX 解析器有两种不同的模式标志可以用来

控制给定的解析器实例是否验证它的输入(SAX2 为此标准化了一个特征标志)。缺省地,这些解析器不进行验证。

你可能希望创建和使用这两个解析器的子类,而且它还可以自动设置这个标志。这样你可以用和传给 SAX ParserFactory API 不同的类名来取得验证解析器或非验证解析器,而不必在应用程序代码的主体中使用非标准的 API 了。这样你在必要的时候在两种解析器间进行切换也比较容易。

你会在第 16 章中对 SAX 了解更多,但是现在重要的是这些小代码段须可以在任何 SAX 解析器上正常工作。应用程序必须提供 DocumentHandler,它会用解析事件来调用。这个处理函数可以构建使用这些事件的数据结构(可能是 DOM 树),而且它经常会处理一些任意数据。你提供 URI(通常是一个 URL,比如 `http://www.example.com/stocks/portfolio`,尽管它也可能是一个文件的 URL),解析器来完成工作。

```
import org.xml.sax.*;
import org.xml.sax.helpers.ParserFactory;
...
try{
    Parser p;
    //just get the parser, configure it, and parse!
    p = ParserFactory.makeParser(parserName);
    p.setDocumentHandler(handler);
    p.parse(new InputSource(documentURI));
}catch (Exception e){
    e.printStackTrace();
}
```

在你学习那些在 SAX 之外的多样的专用 API 的时候请记住这段代码。既是为了可选事件驱动的 API 也是为了所有基于 DOM 的 API。

你可能会注意到前面这段代码没有考虑验证。这是因为无论解析器是否报告验证错误,它的行为都一样。你可以控制它的行为;参照第 16 章以取得关于错误处理和验证的更多信息。

尽管 SAX 解析器经常加入扩展,通常作为解析器对象 JavaBeans 属性,你可能注意到你不会比 DOM 扩展更需要它们。参阅第 16 章的 SAX 2.0 API 的讨论以了解定义和访问这些扩展的标准方法。

13.3.2 取得一个 DOM 文件

DOM Level 1 规范没有定义可移植的 API 以取得 DOM Document 对象。这意味着不先用专用 API 就使用 DOM API 来取得 DOM 文件是不可能的,比如这里介绍的。每一个 DOM 实现者都不得不提供这样的专用 API,因为在这一领域还没有标准。

实际上应用程序有两种方法来取得 DOM document 对象:

- 当处理现有文件的时候,应用程序需要取得 document 实例,这个实例会影响 XML 解析器的输出:元素节点、内容以及其他在 XML 文件中的数据。

每一个实现者都不得不发明一个这个问题的解决方案,但是他们中的很多人都采取

了简单的配置并运行的模式。首先你用构造选项配置一个或多个对象,通常使用 JavaBeans 属性。然后你就可以运行了:解析数据并收集结果。

- 在创建新文件的时候,应用程序还需要取得一个空文件,这个文件可以产生自身。

Java 是这类问题的标准解决方案——为实现类使用缺省的构造函数(DOM Level 2 可能提供了这个问题的部分解决方案)。

关于如何使用 DOM 的更详细的信息请阅读第 15 章。在本节中,你需要了解 DOM 是一个到树形数据结构的 API,这个树是由许多 Node 的子类构成的。特别值得注意的子类包括 Document,它是树根,还有 Element,它对应于 XML 元素并且被用在树中几乎所有非叶子节点上。另外 Text 节点构成了大部分叶子。

你可能想要定义一个“glue”层,使你的应用程序和所有不同的 DOM Document 实现无关。这样的层要使用本节中的信息以访问每一个包。

请记住许多解析器在与 DOM 结合上都有对空格的特殊处理。这是指,它们删除了某些空格类来简化 DOM 数据结构的使用。

1. Sun 的 Java Project X 包

com.sun.xml.tree.XmlDocument 类实现了 DOM 的 Document 接口,而且你可以直接调用它的构造函数。

解析过程

有几种方法可以让这个包为你创建 DOM Document 树。这是惟一提供“单步”选项的包,你可以调用几个静态方法之一,就像更灵活的配置运行选项一样。

单步 API 致力于使用的简单化,考虑到大多数开发者不需要或者不希望了解太复杂的 API。一个基本的非验证解析器看上去如下:

```
import org.w3.dom.*;
import com.sun.xml.tree.XmlDocument;
...
try{
    Document doc;

    //single step:get the document!
    Doc = XmlDocument.createXmlDocument ( documentURI, false);

    //operate on the DOM data structure here
}catch (Exception e){
    e.printStackTrace ();
}
```

最后的 false 意味着“不要验证”。如果你希望验证这个文件,那么你应该返回 true。除了 URI,你还可以提供一个 SAX InputSource。

使用 Sun 的包,解析器会报告在解析过程中产生的大多数错误,包括 I/O 异常和 SAXParseException。参阅第 16 章以了解如何在解析异常时访问位置信息以便诊断和解决

问题。

更灵活配置运行的取得 DOM 文件方法包括使用 SAX 解析器(它不一定是 Sun 的),并用一个 XmlDocumentBuilder 对象作为你的 SAX 文件处理函数。它们可以分别配置,这样就提供了比其他包更多的控制。

构造器的 JavaBeans 属性是:

- `disableNamespaces(boolean)`(缺省为 `true`)——它控制 XML 名字空间的一致性检查。通常不进行这个检查,因为名字空间不是 XML 规范的一部分,而且这样的检查会降低解析的速度。如果打开了 Sun 解析器的检查功能,那么 SAX 错误处理函数就会被用来报告名字空间错误,就像报告验证错误那样。如果对其他解析器打开了这个功能,这些错误将作为致命错误被报告。
- `elementFactory(com. sun. xml. tree. ElementFactory)`(缺省为 `null`)——它是一个可以创建客户的 `ElementNode` 子类来填充 DOM 树方法。这个方法利用了 XML 名字空间的信息。这个 `ElementNode` 子类可以影响 DOM 树的构造函数并增加类型相关的方法(请阅读后面关于客户化 DOM 的讨论)。
- `ignoringLexicalInfo(boolean)`(缺省为 `true`)——它控制生成的 DOM 树是否保存一些有用的编辑信息,这些信息通常会被忽略。可忽略的空格被保存起来(关于可忽略的空格的更多信息请参阅第 16 章)。隐藏在 CDATA 段中的内容在 DOM 树中作为 `CDATASection` 文本节点而不是一般的 `Text` 节点来显示。你还会看到 `Comment` 节点和(只读的) `EntityReference` 节点。
- `locale(java. util. Locale)`(缺省是系统位置)——它提供了错误诊断中使用的位置信息。网络服务器有时需要能够提供客户可以理解的语言的诊断信息而不是服务器管理员所习惯的那种语言(如果你的消息有数字代码,这可能是由于系统位置不支持诊断消息。试一试将位置设为 `English`)。
- `parser(org. xml. sax. Parser)`(只写)——如果你使用 Sun 解析器并设置了这个属性,许多其他特征会工作得更好,比如在树创建时从 DOM 元素中处理位置信息以及在 Sun 解析器中访问扩展的 SAX 特征。

如果你使用 Sun 的非验证 SAX 解析器,你可能想要设置它的 `fastStandalone` 这个取布尔值的 JavaBeans 属性,这个值缺省是 `false`。如果被设成 `true`,这个属性使解析某些文件的过程更快,这是因为,由于文件设成了独立的,所以解析器略过了读取文件的外部子集。如果文件是有效的,这个设置不会影响解析的结果。

Sun 的配置运行版本的 API 看上去如下(如果你没有解析器或者构造器属性可以配置,那么一般你不使用这种 API):

```
import org.w3.dom.*;
import org.xml.sax.*;
import com.sun.xml.tree.*;
...
try{
    Document          doc;
    XmlDocumentBuilder builder;
```

```
Parser                p;  
  
//configure . . . first a parser, then the builder  
//normally JavaBeans properties would be set on both  
p = new com.sun.xml.parser.Parser();  
builder = new XmlDocumentBuilder();  
builder.setParser(p);  
  
//go!  
p.parse(new InputSource(documentURI));  
doc = builder.getDocument();  
  
//operate on the DOM data structure here  
} catch (Exception e) {  
    e.printStackTrace();  
}
```

如果加上这些选项,你就可以调整 SAX 解析器和文件构造器了,如果你使用配置运行模式而不是单步 API 的话,你会注意到可以使用许多配置选项。Sun 提供了这些选项的缺省值以减少生成的 DOM 树中的节点数。

空格

如果你使用的是静态方法,或者构造器的 `ignoringLexicalData` 属性没有被修改,所有 SAX 解析器报告为可忽略的空格都将被忽略。然后,作为 DOM 正规化的一步,连续的空格符将被变成一个空格,除非内容用“`sml:space="preserve"`”属性保护起来(目前不能阻止空格正规化过程)。

如何对待空格?

当你处理 XML 的时候,空格是非常讨厌的,因为它有三种类型。不但难以辨别某些空格可能属于哪种类型,而且也没有处理这三种空格的标准方法。

第一种空格完全是为了创建 XML 文本的人的方便。它使文件的源文件更易于理解。这样的例子包括表示嵌套的元素、段与段之间的空行等等。一个 XML 解析器可以通过依靠可用的内容模型声明来判断一些这样的可忽略的空格。许多 DOM 实现都删除这样的空格。

第二种空格实际上有一些含义。诗经常是由短句构成的,有些编程语言的语法依赖回行,诸如此类。XML 应用程序可以辨别这种空格,如果它遇到了具有 `preserve` 值(而不是缺省值)的 `xml:space` 属性。这个属性值是从祖先继承来的。这样的空格不会被删除。

第三种空格是容易混淆的一种类型——这种空格不能被忽略,而 `xml:space` 处理却是缺省的。XML 规范没有说这种情况下该怎么办。但是还必须做些什么因为大多数空格一般都是可忽略的(诸如缩进)。

一般地,应用程序假设有空格是很重要的,因为文本需要分割,但是大量的空格是不重要的。比如,将缩进和回行用一个空格来代替是没有问题的。但是还有很多不同的解决方案,依赖于特定解决方案的应用程序可能会遇到麻烦。

这个故事是说,文件的创建者应当将他要保留的空格标记出来,而且应当能够将所有其他空格删除。应用程序需要观察它们的 DOM 对空格的处理并且需要做一些

工作以便更正确的处理它们。

2. IBM 的 XML4J v2 包

Com.ibm.xml.dom.DocumentImpl 类实现了 DOM Document 接口,你可以直接调用它的构造函数。

请注意这个 DOM 是可序列化的。序列化的输出比同样的 XML 文本大很多,因为它包括了用来在内存中表示整个 DOM 文件的二进制数据结构。也就是说,它在 XML 上下文中使用了 Java 的序列化模型,而不是将 XML 作为一种更好的序列化语法(因为 XML 可以基于文本而且容易修改和纠错)。

解析过程

IBM 的包有两个 DOM 解析器,其中一个验证解析器(com.ibm.xml.parsers.DOMParser),另外一个非验证解析器(com.ibm.xml.parsers.NonValidatingDOMParser)。它们总是使用配置运行模型。

这些解析器的可设置的 JavaBeans 包括许多属性,在这个包中的解析器共享它们:

- allowJavaEncodingName(boolean)(缺省值是 false)——如果是 true,那么数以百计的编码名称可以通过 encoding="..." 声明接受,这个声明应被放在 XML 文件的开头。否则,只有相对少量的一些名字可以被接受(这些名字只包括那些公共编码的一个很小的集合)。
- checkNamespace(boolean)(缺省值是 false)——控制验证是否检查某些(不是所有)基本名字空间约束是否满足。
- continueAfterFatalError(boolean)(缺省值是 false)——控制解析器是否和 XML 规范一致。XML 处理器在遇到致命错误的时候必须停止向应用程序输出数据。如果它被设为 true,那么这个解析器将破坏 XML 规范,它继续返回数据,就像它在处理结构良好的文件一样。
- documentClass(String)(只写)——使不同的基础文件类可以在解析过程中用做文件(和节点方法)。
- expandEntityReferences(boolean)(缺省值是 false)——控制解析器是否创建 EntityReference 节点(用只读节点)或者用它的子节点来替换这些引用。这两种行为都是合法的,但是如果 DOM 树知道没有 entityReference 节点,通常处理起来会容易一些。
- nodeExpansion(int)(缺省值是 1)——控制某些 DOM 节点的创建是推迟(缺省)还是立即发生(0)。一些开发者已经报告了使用缺省值时遇到了问题。
- warningOnDuplicateAttDef(boolean)(缺省值是 false)——控制解析器在同一元素类型的同一属性被定义多次的时候,是否产生警告。这样的重复定义是合法的而且是有效的,尽管它可能说明 DTD 没有做它打算做的事情。

提示:如果你第一次在文件解码上出现什么问题,可以设置 allowJavaEncodingName 属性为 true,然后看看怎么样。JDK 支持大约 150 种编码,每一种编码都有许多名字。如果你设置 allowJavaEncodingName 为 true,你就可以接受任何这些编码的文件,甚至在它们使用不太常用的名字的时候。如果你没有这

样做,那么你的应用程序可能会不必要地拒绝文件。

“宽进严出”是造成 Internet 全球奇迹的策略。这个策略在这里也适用!

下面是如何取得文件的 DOM 树而不验证它。请注意它采用了配置运行模式:

```
import com.ibm.xml.parsers.NonValidatingDOMParser;  
import org.w3.dom.Document;  
...  
try{  
    NonValidatingDOMParser    parser;  
    Document                    doc;  
  
    //configure . . . in this case, no JavaBeans properties are set  
    parser = new NonValidatingDOMParser ();  
  
    //go! . . . parse and collect the resulting DOM document  
    parser.parse(documentURI);  
    doc = parser.getDocument ();  
  
    //operate on the DOM data structure here  
}catch (Exception e){  
    e.printStackTrace();  
}
```

同样,你可以设置大约半打的 JavaBeans 属性,来控制如何产生这些 DOM 树。IBM 解析器所提供的这些选项和其他大多数解析器不太一样。特别地,其他解析器提供了对于是否包括可忽略空格的某种直接控制。

空格

根据在 DTD 中发现的信息,解析器报告为可忽略空格的文本被当作 DOM 扩展标记。否则,对空格什么都不做。特别地,根本不去尝试删除多余的空格或者使用 `xml:space` 属性来控制任何行为。

3. Oracle 的 XML 包

`oracle.xml.parser.v2.XMLDocument` 类实现了 DOM Document 接口,你可以直接调用它的构造函数。

请注意这个 DOM 是可序列化的。序列化的输出比同样的 XML 文本大很多,因为它包括了用来在内存中表示整个 DOM 文件的二进制数据结构。和 IBM 的 DOM 一样,它在 XML 上下文中使用原始的序列化模型,而不是将 XML 用做更好的序列化语法。

解析过程

Oracle 有一个配置运行模型,你可以配置 `oracle.xml.parser.v2.DOMParser` 对象,然后解析你的输入数据。解析器所用的选项是 SAX 方法的一个超集,一些高级选项使你可以在文件尚未准备好的情况下预先装入 DTD(尽管这是一个有趣的模型,但是它不太符合 DTD 的精神。没有几个解析器支持这个模型)。

`DOMParser` 对象的 JavaBeans 属性是:

- `nodeFactory (oracle.xml.parser.v2.NodeFactory)` (只写)——它使你可以定制所有用来构建 DOM 文件的节点。

- `preserveWhitespace(boolean)` (只写)——如果设为 `true`, 那么将导致在 DOM 文件创建的时候保留可忽略空格。这样的空格一般被丢弃。
- `validationMode(boolean)` (缺省值是 `false`)——它使你控制是否进行验证。

为了用保存了所有节点的 DOM 输出进行标准的非验证解析, 使用下面的代码:

```
import org.w3c.dom.Document;
import oracle.xml.parser.v2.DOMParser;
...
try{
    Document      doc;
    DOMParser     parser;

    //configure . . . could set JavaBeans properties here too
    parser = new DOMParser ();

    //go! . . . just parse, and collect the result
    parser.parse(documentURI);
    doc = parser.getDocument ();

    //operate on the DOM data structure here
} catch (Exception e){
    e.printStackTrace ();
}
```

尽管 Oracle 的解析器提供了较少的可配置属性, 但是在这个包中你还有一些构造选项。

空格

Oracle 的解析器有一个 `preserveWhitespace` JavaBeans 属性, 这个属性可以被用来停止正常的对可忽略空格的丢弃。`xml:space` 属性被忽略。

4. DataChannel 的 XJParser 包

`com.datachannel.xml.om.Document` 类实现了 DOM Document 接口, 你可以直接调用它的构造函数。

请注意这个类的名字和标准的 DOM 接口中的一个接口重名。这意味着你不能在同一代码中同时输入这两个名字。它们中至少要有一个必须通过全类名来访问。在本节中, 你对 DataChannel 解析器使用全类名, 这样 DOM 接口的名字就是指这个接口了。

解析过程

这个包的配置运行模型独一无二, 因为解析器和生成的文件是同一对象。其他的包都使它们成为分离的对象。

可以在 Document 上设置以控制解析器如何解析的 JavaBeans 属性不是标准 DOM 属性。它们中一部分的缺省值和 XML 1.0 不完全兼容。如果你不重新设置它们, 有些 XML 文件可能会被不恰当地报告为有错。影响解析的属性有:

- `encodingInfo(com.datachannel.xml.om.IXMLEncodingInfo)`——封装关于文件所用的字符编码的信息。你可能需要使用它来告诉解析器那些不能被自动检测出来的编码。

- `parserClassName(String)`——使你可以控制所用的解析器。缺省的解析器不是严格的 XML 1.0 解析器,尽管它是验证解析器。它加入了遵守 XML 名字空间以及 Microsoft 定义的数据类型和模式框架的要求。如果需要标准的 XML 支持的话,其他两个解析器也被支持。这些解析器之一不读取所有的外部实体,因此它不能用于验证。另外一个读取所有这样的实体并且进行验证。
- `preserveWhiteSpace(boolean)`(缺省为 `false`)——如果是 `false`,那么在产生的 DOM 中任何没有将 `xml:space` 属性设置为 `preserve` 的元素中的那些空格将被删除。
- `validateOnParse(boolean)`——如果为 `true`(缺省值),而且这个解析器被用来支持它,那么文件将被验证(它影响 `parserClassName` 属性)。

为了用这个 DOM 实现来解析文件,用文件类的一个实例来调用 `load()` 方法之一。缺省地,这将加强许多被解析文本的绑定,这些绑定可能会阻止合法的 XML 被接受。它还会验证它的输入。为了取得和其他例子同样的结果,你必须修改一些缺省值:

```
import org.w3c.dom.Document;

try{
    Document doc;
    Com.datachannel.xml.om.Document loader;

    //configure . . . we had to configure some options to get
    //straight XML 1.0 parsing and to ignore validity errors
    loader = new com.datachannel.xml.om.Document ();
    loader.setParserClassName(
        "com.datachannel.xml.tokenizer.DTDValidatingParser");
    loader.setValidateOnParse(false);

    //go!
    loader.load(documentURI);
    doc = loader;

    //now operate on the DOM document
} catch (Exception e){
    e.printStackTrace ();
}
```

`load()` 方法还有另外两个版本,它们支持直接从文件、输入流、字符串等等中加载。

这些方法报告的异常都是 `RuntimeException` 的子类,而文件不区别那些应当被期望并被处理的特别子类。

空格

这个解析器会删除某些类型的空格。Datachannel 没有整理规范的算法。看上去它只应用于 `xml:space` 属性没有 `preserve` 值的地方,它还包括删除有用信息前后的空格。所有通过 DTD 定义为可忽略的空格都没有被忽略。

13.3.3 在解析之后:DOM 扩展

在将 XML 文本变成 DOM 树之后,通常你的应用程序会以某种方式来操纵这个数据结构。第 15 章将会介绍你处理 DOM 的所有方法。

成功取得表示你 XML 内容的 DOM 树仅仅是一个开始。还有许多应用程序在处理

DOM 时一般需要执行的关键任务,而且这些任务还不能可移植地完成。

本节将大致介绍这些任务中最重要的几个,然后演示上述 DOM 实现如何帮助实现这些任务。

1. 名字空间相关的处理

你所看到的大多数 DOM 实现(Sun 的,Oracle 的以及 DataChannel 的)都试图通过提供 DOM 的 Element 节点对名字空间 URI、名字空间前缀以及名字空间本地部分全名的访问方法来使处理名字空间更为简单。一些实现(特别是 Sun 的)还提供了其他一些功能来允许在任何 DOM Level 1 接受元素名或属性名的地方使用名字空间信息。

这意味着现在你可以编写名字空间相关的应用程序。因为名字空间是 XML 如何在从 XSLT 到模式的应用程序中使用的关键问题和事实上 W3C 最新的基于 XML 的工作,你需要依赖于这些 API。

对名字空间支持的强烈关注已经在 DOMLevel 2 的工作草案有所体现,它看上去很像 Sun 的 API,但是也听取了其他地方的意见。希望大多数这些实现都遵守 Level 2 API,删除可能对应用程序可移植性造成问题的那些 API。然而,这些新的 API 还要一段时间才能面世。

2. 打印 DOM 树

有一个任务几乎所有人都需要它,但是 DOM 却不支持,这就是将数据结构像 XML 文本一样打印出来。这可能很容易实现,因为 Web 浏览器继承了 DOM,这里希望可以将所有数据显示在位图上而不是交给数据处理过程的下一阶段。

本章提到的大多数 DOM 实现都可以很容易地打印 DOM 数据结构,无论是整个文件还是单独节点。它们是非标准的 API,这也是为什么在这里和其他 DOM 的关键扩展一起介绍它们。第 16 章讨论了打印 DOM 树的其他方法,那是通过 SAX API。

如果你仔细想想,正确地打印 XML 就不是那么简单了。有许多特殊的情况,以至于在库代码中隐藏一部分以避免不可避免的误差是值得的。这些情况包括需要略过半打 XML 分隔符,正确表示实体引用(接收者会看到实体声明吗?),通过用适当的数字引用替代来处理字符编码限制,以及甚至仅仅是正确地表示回行。

字符编码问题

XML 支持的字符比许多开发者习惯使用的多很多。实质上,它是在 Unicode 基础上建立的,Unicode 有数十万个字符而且可能最终有上百万字符(有时间的话,用一个小时来浏览一下 Unicode 规范,主要体会一下 XML 可以处理的多种脚本,并且理解如果英语不是第一语言的话,生活是多么复杂!)

这意味着你不必再依赖于你习惯使用的那些编码。ASCII 仅仅支持 128 个字符,它甚至不够处理所有英文单词。广泛使用的 ISO Latin/1 字符集(ISO8859-1)好一点,因为它支持超过 200 个字符,但是它不能处理大多数欧洲语言,就更不用说亚洲语言了。

输出时这特别重要,这里你要选择使用哪种字符集。你可能不能容易地说出你的数据是否含有不适合这些常用编码的字符。

因为 UTF-8 和 UTF-16 编码可以处理所有的 XML 字符而不丢失数据,而且因为所有的 XML 处理器都必须接受这些编码,所以我们通常使用这两种编码之一。Internet

engineering Task Force(IETF)在协议中选择了 UTF-8,因此你可能想要设置它。然而 UTF-8 对于许多亚洲语言的编码有一个缺点——它比 UTF-16 或者专门的编码(比如 EUC-JP, Shift_JIS, Big5, 和 GB-2312 这些广泛使用的日语和中文编码)要占用更多空间。

提示:Java 支持这些 UTF 编码,但是有些 Java 实现不认识官方的 UTF-8 和 UTF-16 名字。如果你不能使用这些名字,试一试非标准的名字 UTF8 和 Unicode。

在你采用一种用不是 UTF-8 或 UTF-16 的编码策略来编写你的 XML 输出的时候,一定要考虑周到。如果你不使用 UTF-8 或 UTF-16,你可能会陷入由缺省的 Java 字符编码类的行为造成的非常神秘的问题。这个问题是一个没有被报告的错误。当这些编码类需要对一个不能为所用编码表示的字符进行编码时,竟然没有产生 java.io.CharConversionException 异常!相反,它被替代为一个问号,静静地破坏你的数据。选择很简单——使用 UTF-8 或者 UTF-16。下一版本的 JDK 应当允许令应用程序来产生这些编码器报告的错误而不是破坏你的数据,但是你采取的恢复过程可能会使用一种不同的编码……,可能是 UTF-8 或者 UTF-16!

用 Sun 的 DOM 打印

XmlDocument 类有一个方便的 write()方法可以使打印整个文件变得非常简单。如果使用 InputStream,那么将会使用 UTF-8,但是如果希望使用其他编码,那么应当使用 Writer:

```
import com.sun.xml.tree.XmlDocument;  
...  
((XmlDocument)doc).write(System.out);
```

为什么漂亮的打印很重要?

和我一起工作的一位开发者正在将那些在 Web 服务器之间交换 XML 消息的 Web 商业应用放在一起。他发现他根本无法很容易地调试它。为什么呢?他简单的打印算法在一行中产生了所有输出,而这一行的长度超过 100K。试图发现其中的错误几乎是没有任何希望的。甚至在即使加入一些行结束符时,也不能发现数据逻辑结构上的问题。

我建议他使用漂亮的打印。它不但将他的输出分成多行,而且还用缩行来表示数据的逻辑结构。这使他几乎立刻就发现了他的错误(一个嵌套错误)。聪明的打印被用来帮助开发者直接理解他的协议消息。这就是简单的调试协议消息的能力,它使 Internet 上基于文本的协议比基于二进制的协议更多。漂亮的打印是这种应用的重要目标。它还在出版应用中非常有用。

Sun 的 DOM 的每一个节点都实现了用一个 XmlWritable 接口来打印。如果你使用这个接口,你可以关闭漂亮的打印并且控制如何打印实体引用(请注意 Sun 的 DOM,通常你不能看到实体引用的节点。你不得不告诉构造器在构建 DOM 树或者自己加入这样的节点时,不要忽略这些编辑信息)。

为了编写不使用漂亮打印的文件,需要如下工作:

```
import com.sun.xml.tree.XmlDocument;  
import com.sun.xml.tree.XmlWriteContext;
```

```

...
XmlDocument          doc;
XmlWriteContext      context;

doc = (XmlDocument) doc;
//creates a write context that doesn't pretty print
context = doc.createWriteContext(
    new OutputStreamWriter (System.out,"UTF8"));
doc.writeXml (context);

```

如果打开了漂亮打印,那么它将造成结构化的缩进,你可以通过解析输出来使之恢复。恢复的能力是通过前面提到的对空格文本的正规化操作来实现的——如果 `xml:space="preserve"` 应用于这个元素,那么也不会加入空格。

文件类型声明被特殊处理。这个解析器中的某些 SAX 扩展被用来报告足够的信息以逐字地重建输入文件中的 `<!DOCTYPE ...>` 声明,而这些信息被用在输出上。

为了方便起见,所有这些 DOM 节点的 `toString()` 方法都提供了一个字符串,它是和相应节点内容一致的结构良好的 XML。

用 IBM 的 DOM 打印

这个版本的 IBM 包没有包括对打印 DOM 的特别支持。

用 Oracle 的 DOM 打印

在 Document 接口的实现类中提供了基本的打印方法。非漂亮打印可以执行。

```

import oracle.xml.v2.XMLDocument;
...
((XMLDocument)doc).print (System.out, "UTF-8");

```

如果你已经有了一个所需编码的 writer,你可以将它隐藏在 PrintWriter 中,但是你不能安排将编码声明放入到输出中。

用 DataChannel 的 DOM 打印

这个 DOM 的每一个节点都支持将该节点作为结构良好的 XML 写出。一定程度的漂亮打印总可以完成。打印的方法要求构造两个辅助数据结构以封装输出流和所用的字符编码。ToString() 方法为任何 DOM 节点提供了结构良好的 XML 文本。

```

import com.datachannel.xml.util.XMLEncodingInfo;
import com.datachannel.xml.util.XMLOutputStream;
...
XMLEncodingInfo      info;
XMLOutputStream      out;

info = new XMLEncodingInfo ("UTF-8",false,false);
out = new XMLOutputStream (System.out, info);

((com.datachannel.xml.om.Document)doc).print (out);

```

3. 定制 DOM

DOM 用一个核心树形数据结构(通常等价于 XML 支持,尽管 DOM 核心没有 DTD 支

持)和许多可选模块构成。Level 1 中,只定义了一个可选模块——为了 HTML 支持。DOM Level 2 在这个模块之外又加入了附加模块。

为了不每次增加新特征时都要实现新 DOM,将 DOM 实现看作是一个和 DOM 密切相关的而又紧密合作的家族的开始是很自然的。XML DOM 仅仅是骨骼。HTML DOM 使用特定的元素和文件类,为它扩展了一些特征。SMIL DOM 也是这样,它用和多媒体相关的特殊方法来使用元素。带有 Traversal API 支持的 DOM 被放在报告 MutationEvents 的最上面。一般地,任何 XML 词表都可以从 DOM 中得到好处,节点增加了读该词表的特殊操作的支持。正如 DOM for HTML 是操纵数据显示变得容易一样,交易词表的 DOM 可能使构建 Web 商务系统更容易。

这里列出的大多数 DOM 实现已经使扩展元素的基本框架比较容易,而且在一些情况下对于其他类型的节点也是如此。也许你想要让文本节点和显示引擎非常高效地集成。如果是这样,你会想要修改 DOM 处理文本节点的方式。下面是一个简短的总结:

- Sun 的 DOM 支持 ElementFactory API。它可以附加在一个独立的文件或者构造器(构造器会将它附加在所有构造器构建的文件中)中。这是惟一可以在决定构建哪一个元素类的时候利用 XML 名字空间的 API。构造器还可以被设置为使用客户文件对象。只有元素可以被客户化。
- IBM 的 DOMParser 可以被设置为使用客户文件对象。因为这个文件是它所有节点(包括元素)的装配方法,它相当普遍。然而,因为 DOM Level 1 在它的装配方法中没有提供名字空间的支持,因此这个 DOM 不能在装配中利用这些名字空间的信息。
- Oracle 的 DOMParser 支持“节点装配”API,它的特征大致上和 IBM 的实现相似。一个不同点就是文件不能创建为使用这样一个装配方法。这个机制只应用于解析器创建的文件,而不适用于独立创建的文件。

用这些特征已经编写了许多应用程序,它们已经被证明是模块化代码非常有用的技术。大多数需要在给定的元素(或其他节点)类型上工作的代码将出现在它特别的子类的方法中。

甚至已经有了多种 DTD 编译风格的应用程序,它们使用一个外部子集文件,并用它来生成理解验证规则的客户元素子类(哪些子节点以哪种顺序排列以及具有哪些属性是合法的,等等)。一旦存在这样的子类,定制的方法就可以被很方便的加入进去。这就使得编写用来加强某个特定 DTD 的规则的系统变得非常的容易。

对于应用程序的作者来说,这些特别的实现技术的一个缺点是它包括了创建某些特定 DOM 实现的 Element 类的子类(DOM 规定了许多行为,而这些行为不在 DOM API 所能实现的范围之内。这些行为必须用实现相关的 API 来实现)。相应的,这些代码不能在 DOM 实现之间移植,可能不会有这类功能的以厂商为中心的 API。

4. 其他通用 DOM 扩展

大多数这些 DOM 实现除了一般的扩展如名字空间、打印和客户类之外,还有许多其他扩展。

当你处理多个文件的时候,如果能够在文件之间移动子树将是很方便的。比如可以从一个 XSLT 样式表中取得一个模板然后放到输出文件中。Sun 的 DOM 支持一个

XmlDocument.changeNodeOwner 操作来高效地在树之间移动节点。如果被改变的节点是现有节点的克隆,那么这个功能类似于 IBM 的 DOM 的 DocumentImpl.importNode 功能,尽管对客户类的处理不同。

Sun 的 DOM 和 IBM 的 DOM 都允许节点被标为只读。这还是递归的。IBM 的 DOM 允许标记被改变。

作为在某些应用中非常有用的元素子类化的替代方案,Sun 的 DOM 和 IBM 的 DOM 都允许元素有一个 userData 字段。在 IBM 的 DOM 中,这个功能不仅限于元素,所有节点都可以有用户数据。

Sun 的 DOM 和 DataChannel 的 DOM 都支持 ID 功能,这是实现 XSLT 或者 Xpath 功能的关键(XML ID 值惟一标识了元素,IDREF 功能是一种内建的链接机制,它可以在除了基本的 XML 1.0 特征以外,没有 Xpath 或者任何其他支持的情况下工作)。

13.4 总 结

本章介绍了几个使用 Java 编程语言的 XML 开发者可以使用的非常出色的 XML 解析器包。你学习了这些包的基本结构,并且看到了每一个包和其他包的比较。然后你学习了使用这些包的第一步——用这些包提供的 SAX API 解析文件。对于许多包来说,你已经学到了许多关于如何将 XML 文本解析为 XML 树的知识,在每一个 DOM 实现中它都有所不同。

有了这个背景,现在你就可以在更仔细地学习后面章节中的 SAX 和 DOM 时选择一个 XML 包来使用。这时你就会看到如何处理 XML 数据,以便开始编写用 XML 来解决问题的 Java 应用程序。

第 14 章 用 C++ 解析 XML

解析器被用在两种环境中,作为独立的解析器或者作为其他应用程序的嵌入设备。正如你在第 12 章“XML 处理基础”中所学习到的,解析器可分为两种风格:

- 检查文件是否是结构良好的。
- 根据 DTD 验证文件的有效性。

在本章中,我们将看到一些可以用 C 编写的解析器。你将知道 James Clark 的 C 解析器,它叫做 Expat,是 Gecko 和 Mozilla 系列浏览器的基础。另外,我们将学习 IBM 和 Microsoft 的解析器。Expat 是一个确认文件是结构良好的解析器;IBM 和 Microsoft 的解析器是验证解析器。所有的解析器都构建成可以为其他应用程序使用的树,而且 IBM 和 Microsoft 的解析器还完全支持 DOM。IBM 解析器目前处于 alpha 测试阶段,但是 Microsoft 的解析器和 Expat 都没有完全开发。Expat 和 IBM 解析器作为独立的解析器来使用(从命令行运行),而 Microsoft 的解析器可以很容易地建在独立的解析器中。我们将在本章中做这个练习。这三个解析器都是 DLL,Microsoft 的解析器还可以作为 COM 对象使用。对 Expat 和 IBM 解析器来说,源代码也可以使用。

本章将着重介绍使用 Microsoft 解析器的实现,因为它是目前发布的惟一可用的 C 解析器。

14.1 为什么用 C 编写解析器

正如你在上一章中所学习到的,大多数 XML 解析器是用 Java 编写的。在开发团体中,Java 是和 XML 紧密相关的语言,可能因为它们都许诺提供完全的跨平台的兼容性。

然而大多数工业强度的应用程序仍然用 C 编写,而 Java 赶不上 C 的执行速度。大多数 Windows 程序是用 C 或者 Visual Basic 之类的快速开发工具编写的。实际上,VB 依然是快速开发 Windows 应用程序的最流行的环境,而且它还是最流行的编写专用数据库接口的语言。它还是 Microsoft Office 套件的编程语法。

因为 IBM 解析器和 Expat 的源代码是公开的,它们可以直接加入到 C++ 应用中,所有三种解析器在 Windows 程序中都作为 DLL 来调用。Microsoft 的解析器还可以作为 ActiveX 对象使用,使得它在应用程序中的使用更方便了(我们将会演示它)。

注意:在本章中交替使用了 C 和 C++。尽管一些解析器比如 IBM 和 Microsoft 的解析器都命名为“用 C 解析”,但它们是用 C++ 编写的。

14.2 Expat 解析器

Expat 非常快,而且它的目标是和 XML 规范 100%的一致。它只检查文件是否是结构

良好的,它构成了 Mozilla 和 Gecko 系列浏览器的 XML 解析器的基础,这些解析器是 Netscape 浏览器的新产品。

14.2.1 取得和使用 Expat

Expat 可使用源代码或二进制版本,可从下列网址获得:

<http://www.jclark.com/xml/expat.html>

Expat 在一个 ZIP 压缩的文件中包含了源代码;它也可以用做 DLL 或者执行程序。后两个可以在 bin 目录中找到,分别是 `xmlparse.dll` 和 `xmlwf.exe`。

`Xmlwf.exe` 是一个可以在命令行执行的独立解析器。

Expat 属于 Mozilla Public License Version 1.1。另外,Expat 可以在 GNU General Public License 下使用。这实质上意味着如果你需要修改它而又没有版权,你可以修改代码。James Clark 欢迎协商任何其他许可证。

如果存在 Expat 的批评,那么文件不太可以参考它。

Expat 的作者,James Clark 假设你对命令行中运行程序的细小差别都非常熟悉,而且你还是一个熟练的 C 程序员。如果你符合上述条件,那么使用 Expat 就没有问题了。然而,如果你是一个初学者,那么准备研读文件吧。

14.2.2 用 Expat 解析

为了在 Windows 中使用这个解析器以检查文件是否是结构良好的,应当打开一个 DOS 窗口然后进入包含 `xmlwf.exe` 的 bin 目录。然后键入 `xmlwf`,一个空格,以及需要解析的文件的全路径名。如果文件解析成功,那么不返回任何错误。如果解析中出现错误,那么会返回一个错误。要解析 `movies.xml` 可以键入:

```
xmlwf filepath.movies.xml
```

图 14.1 显示了当对 `movies.xml` 进行下列修改后返回的错误(修改后文件就不是结构良好的了)。

```
<movies>
  <movie type="comedy" rating="PG-13" review="5" year="1987">
    <Title>Raising Arizona</title>
```

`<title>` 标记被改为 `<Title>`。

这个错误消息给出了错误的行号和字符数。

看到这个快速的例子之后,Expat 是又小又快又免费而且 100%一致。如果你仅仅需要检查文件的结构是否完整,那么它绝对是首选的解析器。Expat 会返回一个解析树给应用程序,但是目前它还不支持 DOM API(尽管 James Clark 正在做这件事)。

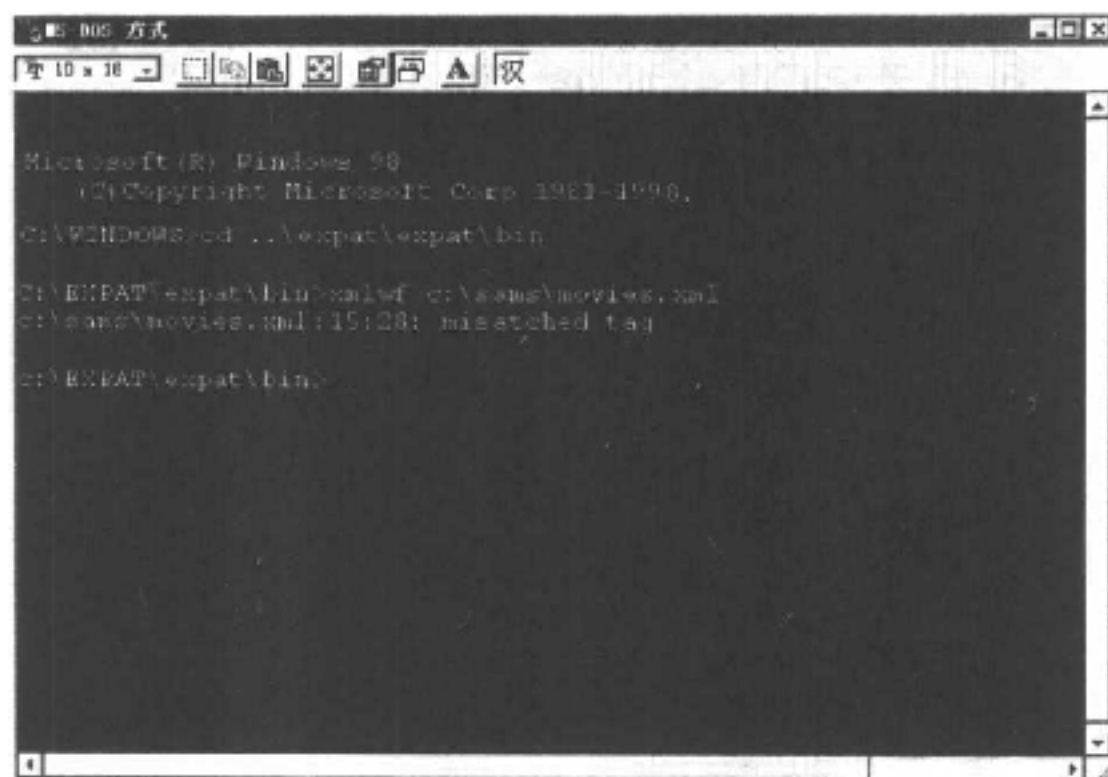


图 14.1 Expat 的 xmlwf.exe 返回的错误

14.3 IBM 的 C++ XML 解析器

IBM 的 C++ 解析器具有一个优秀解析器的所有特征。它快速而且许诺在它的最终版本中将会和 W3C 标准完全一致。它还可以用做 DOM 引擎,提供对 DOM API 的完全支持,它还可以通过 SAX API 来使用。这种解析器有 Windows 版和 UNIX 版,Windows 版是由几个编译好了的执行程序构成的。

14.3.1 取得和使用 IBM 解析器

IBM 解析器可以在 <http://www.alphaWorks.ibm.com/tech/xml4c> 上取得。这个解析器的 ZIP 文件是 xml4c2_2_0.zip。在解压之后,你可以在 bin 目录中看到这个解析器的所有源代码和一系列可执行文件和动态链接库。它提供了丰富的文件。解压之后,它的目录如图 14.2。

DOMCount.exe 都被用来验证独立的 XML 文件。

XML4C 的 alpha 版的许可证极其宽松。IBM 允许你下载并使用你喜欢的源代码(包括修改和发布),为你在程序中对它进行增值提供了支持。其他细节可以在这个你下载源代码的网站上取得。

在 IBM 的软件版权中,IBM 向你授权全世界的、非独占的、不可转变的、免费使用的许可:1)下载,重新制作并准备从软件的目标代码和源代码派生的工作;2)发布这个软件及其派生的工作(软件修改)。

这个许可证其他细节可以从 IBM alphaworks 主页 <http://www.alphaworks.ibm.com/tech/xmltoolkit> 上取得。

不像 Expat,这个 IBM 解析器在 ZIP 文件中的 doc 目录里提供了很好的文件。

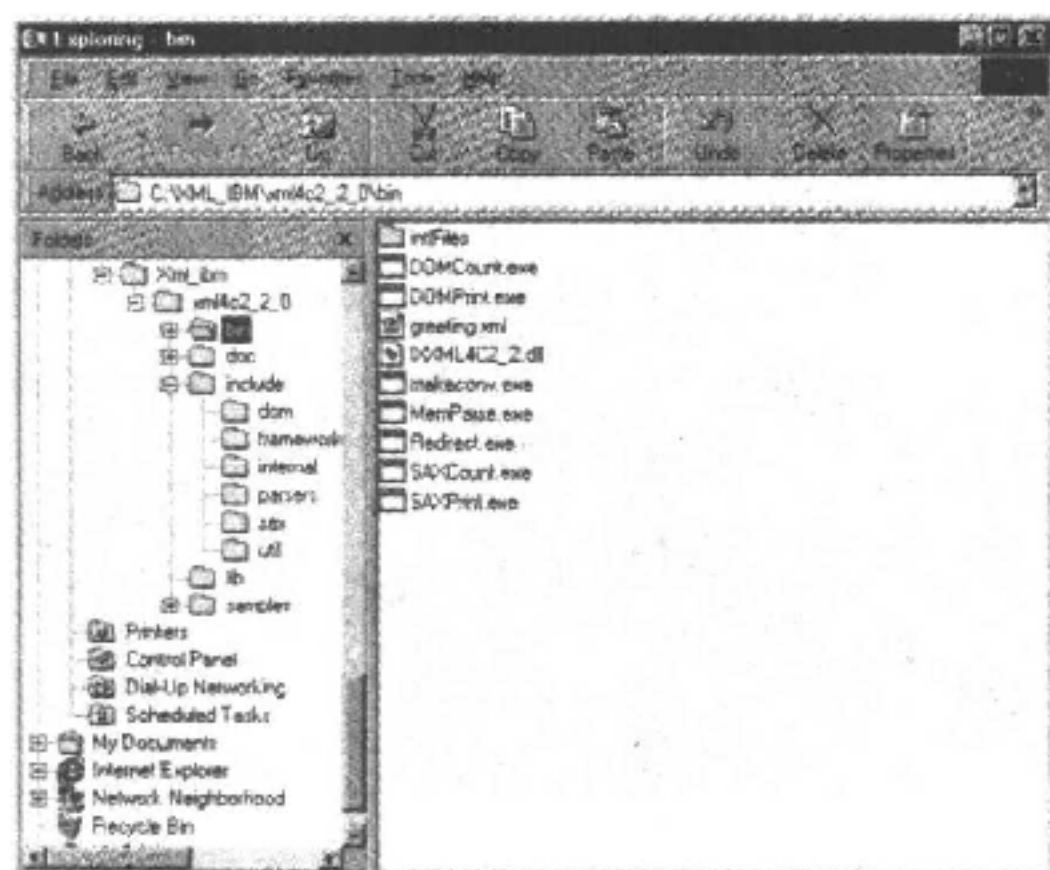


图 14.2 C++解压后的 IBM 解析器

14.3.2 用 IBM 解析器验证文件的有效性

本节简要地介绍了如何用独立解析器来检查文件结构是否完整以及如何用它来验证 XML 文件的有效性。DOMCount.exe 是完成它最好的方式；你可以在 bin 文件夹中看到它。

如果你使用的是 Windows，打开一个 DOS 窗口并且进入 bin 文件夹。在命令行中键入下面的命令：

```
Domcount [filepath]
```

如果你想要得到验证错误的细节，键入：

```
Domcount -v [filepath]
```

为了验证 movies.xml，我键入了下面的命令：

```
domcount c:\sams\movies.xml
```

现在修改 XML 文件为不是结构良好的；在第一个标题中加上一个 T，如下：

```
<movies>
  <movie type="comedy" rating="PG-13" review="5" year="1987">
    <Title>Raising Arizona</title>
```

再次运行这个解析器。图 14.3 显示了修改前后的验证结果。

如果解析器以全面验证的模式运行(如下)：

```
domcount -v c:\sams\movies.xml
```

然后你可以取得许多信息，可以告诉你对于 movie 元素来说 Title 是无效的内容模型(如图 14.4)。


```

Microsoft (R) Windows 95
(C) Copyright Microsoft Corp. 1991-1998.

C:\WINDOWS>cd ..\xml_14\xml4c2_2_0\bin

C:\XML_IBM\xml4c2_2_0\bin>domcount c:\sams\movies.xml
C:\sams\movies.xml: 340 ns (44 elems).

C:\XML_IBM\xml4c2_2_0\bin>domcount c:\sams\movies.xml
Fatal Error at (file c:\sams\movies.xml, line 15, char
340): Exp
'title'
C:\sams\movies.xml: 32 ns (44 elems).

C:\XML_IBM\xml4c2_2_0\bin>

```

图 14.3 用 IBM 解析器验证 movies.xml

```

Microsoft (R) Windows 95
(C) Copyright Microsoft Corp. 1991-1998.

C:\WINDOWS>cd ..\xml_14\xml4c2_2_0\bin

C:\XML_IBM\xml4c2_2_0\bin>domcount -v c:\sams\movies.xml
C:\sams\movies.xml: 37 ns (44 elems).

C:\XML_IBM\xml4c2_2_0\bin>domcount -v c:\sams\movies.xml
Error at (file c:\sams\movies.xml, line 15, char 11): unknown
element 'title'
Fatal Error at (file c:\sams\movies.xml, line 15, char
14): Expected end of tag
'title'
Error at (file c:\sams\movies.xml, line 25, char 11): Element
'title' is not va
id for content model: '(title, writer+,
producer?, director?, actor*, poster?, comment
?)'
C:\sams\movies.xml: 35 ns (44 elems).

C:\XML_IBM\xml4c2_2_0\bin>

```

图 14.4 带-v模式的 IBM 解析器验证 movies.xml

如果你正在用一个成熟的 DTD 来验证 XML 文件,那么这可能对你没有什么用处,但是如果你要调试一个 DTD,那么它就非常有用。图 14.5 显示了解析器在对 movies.dtd 的修改前后运行的结果。

```
<! ELEMENT movie
```

```
(title, writer+, producer+, director+, actor *, poster?, comments?)
```

请注意结束的尖括号被删除了。

```

Microsoft(R) Windows 98
(C) Copyright Microsoft Corp. 1981-1998.

C:\WINDOWS>cd ..\xml_ibm\xml4c2_2_0\bin
C:\XML_IBM\xml4c2_2_0\bin>xmllint c:\sams\movies.xml
Fatal Error at (file c:\sams\movies.dtd, line 5, char 1):
Unterminated element declaration
c:\sams\movies.xml: 26 ms (44 elems).
C:\XML_IBM\xml4c2_2_0\bin>xmllint -v c:\sams\movies.xml
Fatal Error at (file c:\sams\movies.dtd, line 5, char 1):
Unterminated element declaration
Error at (file c:\sams\movies.xml, line 14, char 15): Attribute
'type' is not declared for element 'movie'
Error at (file c:\sams\movies.xml, line 14, char 31): Attribute
'rating' is not declared for element 'movie'
Error at (file c:\sams\movies.xml, line 14, char 45): Attribute
'review' is not declared for element 'movie'
Error at (file c:\sams\movies.xml, line 18, char 55): Attribute
'year' is not declared for element 'movie'
c:\sams\movies.xml: 45 ms (44 elems).
C:\XML_IBM\xml4c2_2_0\bin>

```

图 14.5 带-v 选项的 IBM 解析器验证 movies xml(含一个错误)

14.4 Microsoft 的 C 解析器

Microsoft 有两个用 C 编写的解析器。第一个是一个简单的连接检查器,它检查文件是否是结构良好的并且是有效的。第二个是一个成熟的解析器和对 DOM 引擎声称 100%一致的——实际上,它也就是 95%一致——并提供对 DOM 的完全支持和有限 XSL 支持的 DOM 引擎。第二个解析器将在本章 14.5“MSXML 解析器”一节中进行讨论。

14.4.1 Xmlint.exe 解析器

连接检查器可在 <http://msdn.microsoft.com/xml/default.asp> 中获得。你只要搜索 xmlint.exe,它非常的快。同样,与另外两个解析器类似,xmlint.exe 以命令行方式运行。错误信息几乎是所有命令行解析器的最大优点,因此它在调试 XML 文件或 DTD 时非常的有用。

要使用 xmlint,只要解开文件,打开一个 DOS 窗口,定位包含 xmlint.exe 的目录并输入你希望验证的文件。例如:

```
xmlint filepath. filename
```

图 14.6 和 14.7 显示了与在 Expat 和 IBM C++ 实现的解析器上显示的相同的错误。

14.5 MSXML 解析器

Microsoft 用 C 实现的解析器并不提供源代码,但它提供 DLL 和 ActiveX 对象。DLL 是 MSXML.dll,而 ActiveX 对象是 Microsoft.XMLDOM。Microsoft 对它的解析器进行了如下要求:

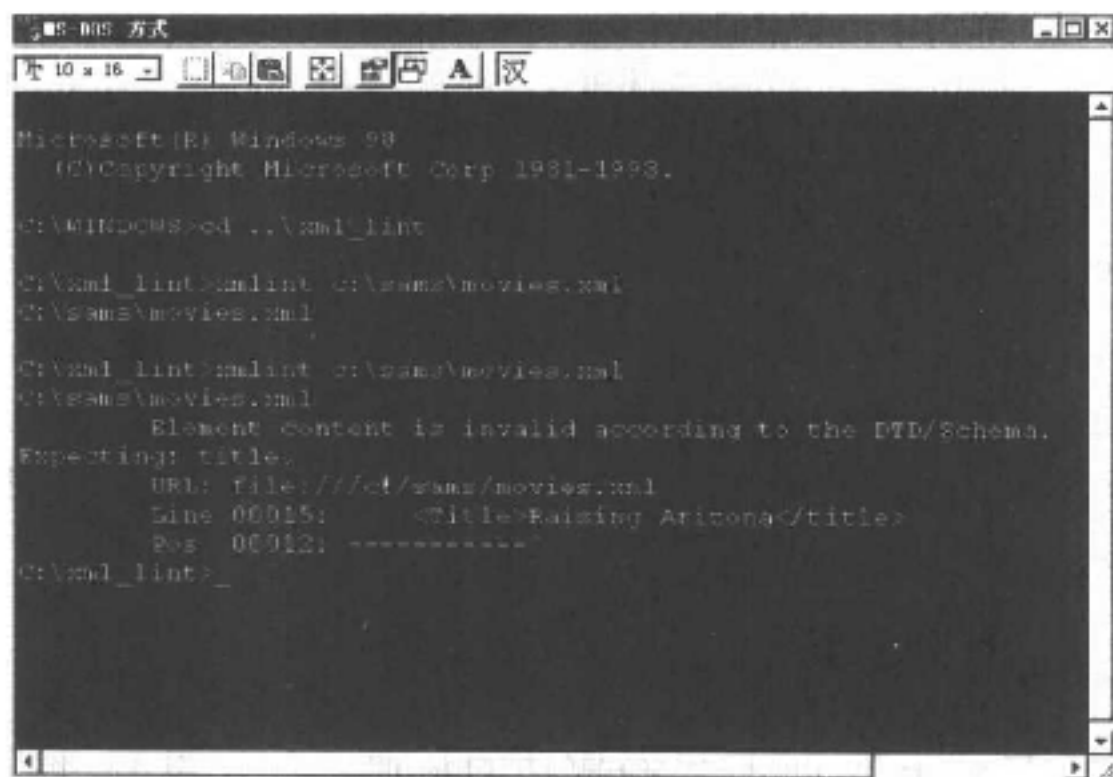


图 14.6 xmllint 验证 movies.xml 中的<title>标记不正确

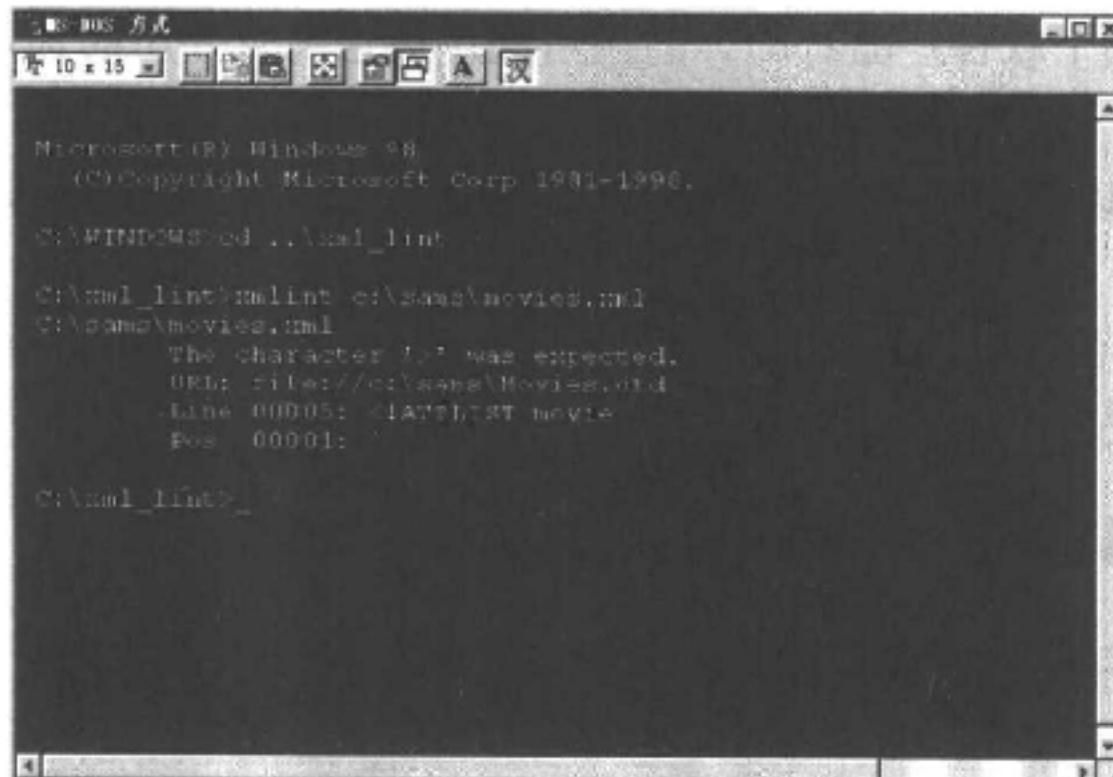


图 14.7 xmllint 验证 movies.xml, 它缺少结束符>

- 100%符合 W3C XML 1.0
- 100%符合 XML DOM
- 支持 ECMAScript、Java、Perl、Python、SQL、Visual Basic 开发系统、Visual C++ 开发系统和 Visual Basic Scripting(VBScript)
- 完全支持 DTD 和有效性
- 支持 XSL
- 支持查询
- 支持名字空间
- Schema 技术预览

14.5.1 获得并使用解析器

解析器可以从 <http://msdn.microsoft.com/downloads/tools/xmlparser/xmlparser.asp> 中获得,它是一个自动装载可执行文件,它将自动的在你的系统上注册 DLL。实际上,如果你有 Internet Explorer 5(IE5),你已经自动的拥有的这个解析器。这个解析器是内建的,作为 COM 对象实现的。

如果你通过运行来自 Microsoft 的可执行程序或是作为 IE5 包的一部分获得了这个 DLL,这个 DLL 就会被注册。如果没有,你必须注册它。如果你使用的是 Window 95 或 98,你可以选择“开始”、“运行”并键入:

```
REDSVR32 MSXML.DLL
```

你应该得到一个信息告诉你 DLL 已经成功的注册。如果没有,你就会得到一段错误代码告诉你出现的问题。

解析器服从标准的 Microsoft 中断用户许可协议。这个协议的相关部分如下:

你只可以通过对象代码形式复制和分发这个软件产品,规定:(a)你将它作为你的应用程序的一部分分发这个软件产品;(b)你的应用程序给这个软件产品加入了重要的和主要的功能。

换句话说,你可以免费使用它来构建你的应用程序。但不要尝试拆除它而使用它们的类!

请在 Microsoft 的 Web 站点上看一下完整的协议。这个解析器的完整文件可以从 <http://msdn.microsoft.com/xml> 中获得。

14.5.2 实现解析器

解析器作为对象实现。实际上,它已经完全的实现并可以在 IE5 中使用了!这里有两种调用它的情况:

- 当 XML 文件导入 IE5 时。
- 当使用 IE5 浏览器打开一个 XML 数据源对象时(要了解有关 DSO 的更多信息,请参见第 2 章)。

解析器还可以包含在网页中和诸如 COM 对象这样的应用程序中。下面将提供如何实现的例子。

用 JavaScript 实现解析器

为了实例化 JavaScript 的解析器对象,使用如下语法:

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

用 VBScript 实现解析器

为了实例化 VBScript 的解析器对象,使用如下语法:

```
Dim Objectname  
Set Objectname=createObject("Microsoft.XMLDOM")
```


用 ASP 实现解析器

为了实例化 ASP 的解析器对象,使用如下语法:

```
Dim Objectname  
Set Objectname=Server.createObject("Microsoft.XMLDOM")
```

用 Visual Basic 应用程序实现解析器

这里的语法与 VBScript 的完全相同:

```
Dim Objectname  
Set Objectname=createObject("Microsoft.XMLDOM")
```

将 XML 调入到解析器对象中

一旦解析器对象被创建,他就需要被 XML 文件调用。记住,每个 XML 文件都必须有一个解析器对象的分离的实例。XML 是如何调用的依赖于它是字符串或字符串变量形式还是外部文件。要调用字符串,使用解析器对象的 loadXML()方法。调用文件,使用解析器对象的 load()方法。

这是调用字符串或是字符串变量的语法:

```
Objectname.loadXML(string or string variable)
```

要调用 XML 文件,使用如下语法:

```
Objectname.load(file path)
```

解析器将在调用时解析和验证文件。它也将分解所有的实体和分解外部文件。如果在这个过程中出现了任何的问题,它将迅速的停止解析文件并返回一个错误。这个错误可以使用代码获取。

14.5.3 MS 解析器错误处理

在装载 XML 文件期间,解析器将使用解析器的 parseError 属性将错误以 parseError 对象形式返回。这个对象提供有关位置和错误类型的可用信息。ParseError 对象有以下只读属性,如表 14.1 所示。

表 14.1 parseError 对象属性

| 属性 | 定义 |
|----------------------|--------------------------------|
| ParseError.errorCode | 这是错误代码的代号。大多数情况下用来在条件状态下进行错误检查 |
| ParseError.line | 给出错误出现的行号 |
| ParseError.reason | 给出出现错误的理由 |
| ParseError.url | 给出包含错误的文件地址 |
| ParseError.linepos | 给出损坏状态的字符号 |
| ParseError.srcText | 给出包含错误的整行的文本 |
| ParseError.filepos | 用长整数形式给出错误的绝对文件位置 |

parseError 属性的详细细节可以在如下地址找到:

<http://www.msdn.microsoft.com/xml/c-frame.htm#/xml/default.asp>。

14.5.4 MS 解析器的选项和模式

与 MS 解析器相关的有众多属性、方法和事件。它们的大部分实际上都是 W3C DOM 属性和方法,这些将在下一章进行讨论。然而,它们中的部分是专门用于解析器执行的。你已经在这一章学习到了 `load()` 和 `loadXML()` 方法。在这一节,你将会看到更多。

解析器的默认行为是分解所有的外部文件和实体,并在加载时验证 XML 文件。然而,你可以使用 `validateOnParse` 和 `resolveExternals` 启用或关闭任一个或者两个行为。然而,首先让我们看一下 `readyState` 属性,它提供有关文件是否被完全加载的信息。

readyState 属性

`readyState` 属性是一个长整数,它提供有关文件加载到解析器对象过程的信息。这里有 4 个确认状态:

- `LOADING(1)`——加载在进行中,但解析器还没有开始解析数据。
- `LOADING(2)`——文件被加载并被解析,但对象模型还不可用。
- `INTERACTIVE(3)`——对象模型可用于只读操作。
- `COMPLETED(4)`——文件被完全的加载,操作成功或不成功。如果不成功,将会产生错误代码。

validateOnParse 属性

`validateOnParse` 属性是一个可读/写属性,它返回布尔值。如果设置为真(true),文件将被 DTD 检验(如果可用的话)。

resolveExternals 属性

`resolveExternals` 属性是另一个可读/写布尔属性。如果设置为真,所有的实体和 SYSTEM 扩展将在文件验证前被分解。如果没有找到文件就会返回错误(Error)。

save() 方法

在使用文件对象模型构建文件时你会遇到一个问题,即如何不断的保存文件。它通常使用手工回卷函数完成。这个函数的一个例子将在下一章介绍。MSXML 解析器使用它自己的 `save()` 方法,它将使用 DOM 构造的文件保存为文件或支持持续性的另一个对象(例如另一个 XMLDOM 文件)。`save()` 方法的语法如下:

```
xdoc.save( file path and name)
```

注意:文件位置必须是一个文件路径——不是 URL。

这个方法将不在浏览器上工作(由于安全的原因),但它在构建应用程序方面非常的有用。清单 14.1 来自一个 VB 应用程序,它创建 XML 文件,使用 DOM 加入元素并保存结果。

清单 14.1 简单 VB 子程序

```
Private Sub SaveXML()  
    Dim xdoc As Object  
    Set xdoc = CreateObject("Microsoft.XMLDOM")
```

```

Xdoc.loadXML("<greeting> Hello Save property! </greeting>")
    Set dummy = xdoc.createElement("farewell")
    Set dummy1 = xdoc.documentElement.appendChild(dummy)
    Xdoc.save("c:\savit.xml")
End Sub
c:\savit.xml will contain the following:
<greeting>
    Hello Save property!
    <farewell/>
</greeting>

```

有关方法、属性和事件的完整列表(将超过 100 项)可以通过在 <http://msdn.microsoft.com/workshop/> 上搜索找到。我们在本章和下一章已经接触到了它们中的最重要的部分。

14.5.5 用 IE5 打开 XML 文件

在学习如何构建 MSXML 为工作解析器之前,让我们看一下我们使用 IE5 浏览器可以做什么。IE5 浏览器确实可以作为一个解析器使用! IE5 混合了 MSXML 浏览器,其中 `resolveExternals` 属性设置为真, `validateOnParse` 属性设置为假。这些在 IE5 浏览器中是用硬件设置的,不能被修改。如果用你在以前用过的同样错误来重写 `movies.xml`(如下所示):

```

<movies>
    <movie type="comedy" rating="PG-13" review="5" year="1987">
        <Title>Raising Arizona</title>

```

然后将文件加载到 IE5 中,你所看到的就如图 14.8 所示。



图 14.8 IE5 验证 `movies.xml`

在 IE5 中展开实体

IE5 浏览器将尝试展开所有的实体并分解所有的外部文件。如果文件中的实体有什么

问题的话,它就会返回一个错误。

例如:

```
<greeting>Hello XML Entities! &.ent;</greeting>
```

将在 IE5 中返回一个错误。下面的文件将返回一个错误,要求 greetingblorp.txt 存在!

```
<? xml version = "1.0"? >
<! DOCTYPE greeting[
<! ELEMENT farewell (#PCDATA)>
<! ENTITY ent SYSTEM "greetingblorp.txt">
]>
<greeting> Hello XML Entities! &.ent;</greeting>
```

在 IE5 中验证

IE5 不按照 DTD 验证文件。下面是一个完全不合法的文件:

```
<? xml version = "1.0"? >
<! DOCTYPE junk[
<! ELEMENT farewell (#PCDATA)>
<! ENTITY ent SYSTEM "greetingblorp.txt">
]>
<greeting> Hello XML Entities! &.ent;</greeting>
```

IE5 将非常高兴的显示它!

然而,IE5 有点荒谬的检查大部分 DTD 的语法! 如果我们像下面这样移去这个低劣的 DTD 的尖括号:

```
<! ELEMENT farewell (#PCDATA)
```

它将发出强烈的抱怨(如图 14.9 所示)。对错误的语法将产生错误,但不尝试验证文件内容。

由于 IE5 将在显示文件之前自动分解任何外部文件。即便是它并不准备检查文件的有效性,它还将分解外部的 DTD。如果 DTD 不可用,或者如果它包含错误的语法,你将得到一个错误消息。你应当注意这个事实,因为它将导致一些多余的错误信息,这些在你尝试加载一个有外部 DTD 的 XML 文件到 IE5 的时候就可能见到了。

14.6 用 MSXML 构建解析器接口

本节将展示如何使用 MSXML 和 IE5 浏览器构建一个有效的解析器。解析器实际上是一个结合 ActiveX 对象“Microsoft.XMLDOM”作为它的引擎的 HTML 页面,它使用 IE5+ 浏览器打开。这个浏览器可以用来解析本地文件和 Internet 上的文件,它既可以作为非验证浏览器也可以作为验证浏览器解析。这是在下一节中将要用到的浏览器,在那里我们将学习验证文件和调试 DTD。

在这里介绍它主要是因为它阐明了大部分的将 MSXML 解析器引入应用程序的技术。


```

</form>

<script>
//Declare script wide variables.
var oXDOM,errstr,mydoc

Function parsexml(xpath)
{
//This function creates an instance of the XML dom from the file
//and attempts to load it with the path passed to it
errst = ""
errstr="parsing:\n" + xpath + "\n"

// create an instance of the XMLDOM parser object.
OXDOM=new ActiveXObject("Microsoft.XMLDOM")
//check to see the type of parser required
if(parseselect.parsechoice[0].checked==true)
{
errstr=errstr + "Parsing in the validating mode.\n\n"
oXDOM.validateOnParse = true
oXDOM.resolveExternals=true
}
else if (parseselect.parsechoice[1].checked==true)
{
errstr=errstr + "Parsing in the well-formed mode only.\n\n"
oXDOM.validateOnParse = false
oXDOM.resolveExternals=false
}
else if (parseselect.parsechoice[2].checked==true)
{
errstr=errstr + "Parsing in the well-formed mode only
and expanding entity references.\n\n"
oXDOM.validateOnParse = false
oXDOM.resolveExternals=true
}
// load the document
oXDOM.load(xpath)

// callisloaded function
var dummy=isloaded()
}

function isloaded()
{
//This function checks to see whether the document has been loaded
//and is ready for parsing.
//If not it calls itself again till the document is
//loaded or an error is returned.
//Is the document loaded and ready for parsing?
If (oXDOM.readyState==4)
{
//if it is were there errors in the document
if (oXDOM.parseError==0)
{
errstr=errstr + "the above document parses\n\n"

```

```

        errstr=errstr + "the root node of this document is: \n"
        + oXDOM.documentElement.nodeName
    }
    //If there were errors build an error string
    else
    {
        errstr=errstr + "The above document does not parse.
        The following information is available about
        this error. Fix the error and try again. \n\n"
        errstr=errstr + "The error is: \n"
        errstr=errstr + oXDOM.parseError.reason + "\n"
        errstr=errstr + "The error occurred in the following document: \n"
        errstr=errstr + oXDOM.parseError.url + "\n"
        errstr=errstr + "at line number: \n"
        errstr=errstr + oXDOM.parseError.line + "\n"
        errstr=errstr + "The character position is: \n"
        errstr=errstr + oXDOM.parseError.linepos + "\n"
        errstr=errstr + "The text of the line is"
        errstr=errstr + oXDOM.parseError.srcText + "\n"
        errstr=errstr + "the absolute file position of the error is: \n"
        errstr=errstr + oXDOM.parseError.filepos + "\n"
    }

    //display the error string in the text area
    document.parser.ta.value=errstr
}

//the document is not loaded yet.
//Alert the user and call isloaded() again
else
{
    alert("document '" + parser.xmlpath.value + "' is loading")
    var dummy=window.setTimeout("dummy=isloaded()",100)
}
}

//this function displays the file
function getxmlfile()
{
    // alert (parser.xmlpath.value)
    window.location.href=parser.xmlpath.value
}

</script>
<form>
<h4>Click this button to view the XML file</h4>
<input type="button" value="View XML File" onclick="getxmlfile()">
</form>
</html>

```

这个解析器代码可分为五个部分：

- 构建接口的 HTML 代码。
- 允许用户输入的表单元素。

- 显示解析器结果的表单元素。
- 创建 MSXML 解析器的实例,检验用户的选择并尝试加载文件的脚本函数。
- 检验文件加载的过程以及确认在加载过程中是否会产生错误的脚本函数。

你应该理解前三个部分。而后两个部分需要进行一些解释。

14.6.1 在解析器中详述脚本

变量需要是整个脚本范围的,因为必须在两个函数中被操作(如下所示):

```
//Declare script wide variables
var oXDOM, errstr, mydoc
```

下面,你需要检查用户选择。我们使用简单的条件式来检验用户的解析选择,如下所示:

```
if(parseselect.parsechoice[0].checked == true)
{
    errstr=errstr + "Parsing in the validating mode. \n\n"
    oXDOM.validateOnParse = true
    oXDOM.resolveExternals = true
}
else if (parseselect.parsechoice[1].checked == true)
{
    etc.....
}
```

每个语句将设置 validateOnParse 和 resolveExternals 属性为必要的模式。

一旦实现了,就要尝试加载文件,如下:

```
//load the document
oXDOM.load(xpath)
```

isloaded()函数被调用,首先看一下调用过程是否完成,然后看在加载过程中是否出现了什么错误。

```
//callisloaded function
var dummy=isloaded()
```

isloaded 函数检查是否文件已经被加载,然后检查在解析过程中是否有错误出现。如果有,它就构建一个描述错误的错误字符串。

首先你必须确定文件被完全的加载:

```
//Is the document loaded and ready for parsing?
if(oXDOM.readyState == 4)
{
```

如果文件从本地资源加载的话,这通常不是问题。除非文件很大或是 DTD 很复杂,但从 Internet 上加载是一个缓慢的过程。如果文件没有加载,警告窗口将通知用户并且 isloaded()函数会在短时间的间隔后重新调用。例如:


```

else
{
    alert("document ' " + parser.xmlpath.value + "is loading")
    var dummy=window.setTimeout("dummy=isloaded()",100)
}

```

实际上,警告是给文件加载时间的一个托词! 如果文件被加载,parseError 属性被检验以确定在调用过程中是否有什么错误发生:

```

if(oXDOM.parseError==0)
{
    errstr=errstr + "The above document parses\n\n"
    errstr=errstr + "The root node of this document is:\n"
    + oXDOM.documentElement.nodeName
}

```

如果没有错误,用户就会被告知文件被解析。如果有错误,错误字符串就会通过 parseError 对象的 reason、line、linepos、srcText 和 filepos 属性进行书写:

```

else
{
    errstr=errstr + "The above document does not parse.
    The following information is available about
    this error. Fix the error and try again.\n\n"
    errstr=errstr + "The error is: \n"
    errstr=errstr + oXDOM.parseError.reason + "\n"
    errstr=errstr + "The error occurred in the following document:\n"
    errstr=errstr + oXDOM.parseError.url + "\n"
    errstr=errstr + "at line number:\n"
    errstr=errstr + oXDOM.parseError.line + "\n"
    errstr=errstr + "The character position is:\n"
    errstr=errstr + oXDOM.parseError.linepos + "\n"
    errstr=errstr + "The text of the line is"
    errstr=errstr + oXDOM.parseError.srcText + "\n"
    errstr=errstr + "The absolute file position of the error is:\n"
    errstr=errstr + oXDOM.parseError.filepos + "\n"
}

```

错误字符串会在 text 域中显示。

```

//display the error string in the text area
document.parser.ta.value=errstr

```

14.6.2 使用 MSXML 解析器解析和验证 XML 文件

要验证任何文件,只要定位需要解析的文件或输入它的 URL 地址,选择解析模式并点击解析按钮。

任何的解析错误都会在本框中出现。

14.7 MSXML 解析器的错误

与任何类型的软件类似,MSXML 解析器也有一些小错误!这里有两点需要引起你的注意:

- 巨大 DTD 的不正确操作。
- DOM 中的不正确语法。

提示:Microsoft 产品使用的基本解析引擎包含在 MSXML.dll 中。它同时包括一个验证解析器和一个 DOM 管理引擎。若引用是用于 Microsoft 的 C 解析器、MSXML 解析器、MSXML.dll 或 ActiveX 对象 Microsoft.XMLDOM,它们所指的都是同样的东西——基本 Microsoft XML 引擎。

14.7.1 巨大 DTD 的不正确操作

这个解析器对于有着多个参数实体的巨大 DTD 来说是一个粗劣的操作者。我可以告诉你,问题出在解析器一块块的读入文件。如果块中包含需要扩展的参数实体,或还包含一个参数实体(例如,嵌套的参数实体)的参数实体,解析器将不能继续并读取文件的下一块。这将在本书的稍后有关 XHTML DTD 的部分进行分析。

14.7.2 DOM 中的不正确语法

DOM 包含一个名为 doctype(都是小写)的属性。MSXML 解析器使用“驼峰”符号 docType。这将在讨论 DOM 的 Document 接口时涉及到。

14.8 在文件和 DTD 设计中使用 C 解析器

到目前为止,我们已经讨论了将 C 解析器放入到我们的应用程序中的方法。这里假定有一个文件类型可从我们提供的确定的 DTD 中获得。实际上,很多开发人员放弃设计符合他们自己的 DTD 的专有文件类型这项任务。甚至对于非常简单的文件,这也是一个费劲的差事。本节将粗略地讲解一些有关使用 C++ 解析器设计文件和 DTD 的一些问题,以及不同解析器的强壮性及其缺陷;本节将只讨论基本原理。

14.8.1 设计文件类型

设计和构建文件类型并不是简单的编写代码。或许最为重要的步骤是与那些使用包含在文件中的信息的人交谈以及与那些使用文件的人进行交谈。

例如,对于医学类文件,交谈将不能只包括记录部门的职员。你应该还要会见医生、护士和任何将不仅要完成文件还需要从文件中获取额外信息的人。

一旦文件的内容被决定,文件类型的设计通常要经过如下几个阶段:

1. 构造元素列表。
2. 确定元素顺序、嵌套和它们应该包含的属性。

3. 构造 XML 模板文件。
4. 构造 DTD。

解析器通常用在过程的每一个阶段,以确保遵守 XML 的规则。这里也许还有一些天才,他们可以只用眼睛就精确理解 XML 文件,而对解析器的需求却很小。

14.8.2 调试模板文件

模板文件的惟一需求是它必须是结构良好的!任何解析器都可以轻易的实现这一点。然而,这里还要强调的就是在设计阶段,文件应当在你验证它之前经常的被解析以确定结构良好性。让一个错误混进文件实在是太容易了。通过这个简单的规则,我们可以节省很多小时的工作。

14.8.3 检查 DTD 语法

下一件要做的事就是检查 DTD 语法。做这件事的方法是创建一个最小限度的 XML 文件,依靠 DTD 运行这个文件。任何的语法错误都会被给出。你也可以通过运行无验证但有扩展实体模式的 MSXML 来检查语法。或者你可以用无验证模式运行 IBM 解析器。

在以上两种情况下,DTD 将导入到文件中,语法错误将会被列出。然而,文件将不能检查内容准确性。

14.8.4 使用 C++ 解析器调试 DTD

当解析和调试一个长而复杂的文件的时候,通常最好要使用两个和三个解析器。我推荐 MS 和 IBM 解析器,并在出现问题时使用基于 Java 的解析器。

MSXML 解析器在解析和调试巨大 DTD 方面有一些缺点。

首先,它对等的解析每一个错误,而错误已经在解析器处理前出现了。其次,它不能很好的处理巨大的 DTD,特别是在如果它们有枚举参数实体的时候。MSXML 使用简单的 DTD 可以在解析巨大文件方面做的很好,但它对于更大的 DTD 的工作是粗劣的。

一个可以从公共场合获得的巨大 DTD 的例子就是 XHTML DTD。XHTML DTD 可从 <http://www.w3.org/TR/xhtml1/DTD/strict.dtd> 上获得,并且任何人都可以免费使用它。让我们尝试在 MSXML 基本解析器中解析如下 XHTML 文件:

```
<? xml version="1.0">
<! --
<! DOCTYPE html SYSTEM "http://www.w3.org/TR/xhtml1/DTD/strict.dtd">
-->
<! DOCTYPE html SYSTEM "strict.dtd">

<html>
<head>
  <title>A simple XHTML document</title>
</head>
<body>
  <p>a paragraph</p>
</body>
</html>
```

你将得到如下的错误消息：

```
The error is:  
The element 'html' is used but not declared in the DTD/Schema.  
The error occurred in the following document:  
file:///C:/SAMS/CH14/EXAMPLES/xhtmll.xml  
at line number:  
7  
The character position is:  
7  
The text of the line is<html>  
The absolute file position of the error is:  
149
```

可悲的事实就是 MSXML 解析器包含错误,这使得它不能很好的处理参数实体。它似乎逃避了两层实体扩展!

这个文件将在 IBM C++ 解析器上很好的解析。

14.8.5 解释错误消息

由解析器提供的错误消息是专有消息。换句话说,用来描述错误的语言对于每个解析器来说是不同的。MSXML 解析器可以提供的错误的完整列表可以在下面两个 URL 中找到:

<http://msdn.microsoft.com/xml/reference/xml/dom/err-messages.asp>

<http://msdn.microsoft.com/xml/reference/xml/dom/DOM-error-message.asp>

当解释错误时,要记住错误在错误消息给出之前已经发生了。

如果所给的错误位置是第一行第一个字符,则通常在 DTD 中存在语法错误。

14.9 总 结

本章讲解了一些使用 C++ 的解析器,有 IBM、MS 解析器和 James Clark 的 Expat 解析器。

解析器有两个用途:

1. 它们可以用来解析纯粹的简单的 XML 文件。
2. 它们可以结合在 XML 文件上执行有用工作的应用程序中。

在 XML 逐渐成为主流的时候,我们可以找到越来越多的应用程序使用 C++ 解析器,从而获得 C++ 语言在功能和速度上的优势。

第 15 章 使用文件对象模型

本章比较长,不过它介绍的是 XML 最重要的一个方面:文件对象模型(DOM)。DOM 为你提供了一种访问和操纵 XML 文件的方法。没有 DOM,XML 就仅仅是一种可以通过多种方法访问的存储系统。通过 DOM,XML 渐渐成为一种真正统一的、跨平台的、应用无关的编程语言。毫不夸张地说,DOM 是 XML 系列建议(recommendation)中最有用的一个方面,这个建议可以使我们立即着手开始作有用的工作而不是在将来的什么时候才作。

和 XML 一样,所有标准的主要问题是那些实现标准的应用程序会产生它们自己的方法来操纵这些标准。就像 HTML 那样,Netscape 的浏览器和 Microsoft 的浏览器使用各自不同的语法和方法来动态地操纵 HTML 文件。结果是,那些希望使用 DOM 的作者们必须留下来编写单独的脚本和页面以适应不同的浏览器或者是使用最基本的公共标准集。

HTML 中 DOM 的事实上的工业标准是 JavaScript 支持的那个模型。尽管这个模型很有用,然而它是不完整的,也不允许访问整个文件的各个部分。

好的 DOM 必须是完整的。完整的 DOM 应该可以从模型自身就重建整个文件,而且应当允许对文件任何一个部分进行访问。另外,DOM 还应当具有一系列方法来对原始文件进行操纵、增加和删除等操作。幸运的是,为 XML,W3C 已经制定了一个建议,它满足了大部分上述要求。

15.1 W3C 和 XML DOM

W3C 有一个工作组专门为 XML 和 HTML 构建 DOM。尽管他们目前发布的版本在某些方面还不完善,但他们的 DOM 已经允许访问 XML 文件的所有部分。

这个工作组定义了不同级别的支持:

- level 0 重申了 JavaScript 的语法,它可以用来操纵 HTML 文件。我们将要讨论这一级别。
- Level 1 允许访问 XML 文件体的所有部分。它不允许访问 DTD 或与文件相关联的样式表。这是当前的建议,我们将要在本章的后面学习它。
- Level 2 是现在正在研究的模型。当它被完成之后,它可能会在现有模型的基础上允许对 DTD、样式表以及名字空间进行访问。

工作组实际上正在制作一套 API(应用编程接口),以便被不同的应用所共享。

15.1.1 DOM 的 API 接口

在面向对象程序设计中,公用的 DOM API 使程序员可以编写一系列代码来解释运行在任何应用程序或是平台之中的文件。只要应用程序支持同样的 DOM,它是用 VB,C 还是 Java 编写的就无所谓了。它们将会为程序员提供相同的一组属性和方法。

这种情况就像是证券经纪人和客户之间的关系,同样的语言可以用在现实社会中也可以用在应用程序的接口上。

如果我想要买进一些股票,我可以自己来完成,在相应的证券交易所买一个座位,并且了解所有有关客户以及交易方法的信息(另外,如果这家证券交易所是外国的交易所,我还不得不同时学习它的语言)。或者,我可以打电话给我的经纪人,并且要求他为我购买这些股票。我已经和经纪人达成协议,我同意使用某些方法并且使用共同的语言(买进、卖出、短期、保留等等)。

如果我有一个应用程序想要和另外一个应用程序通信,也会产生类似的过程。在协议达成之后,这个应用程序为我的应用程序提供一组属性方法。这两个应用程序通信所使用的语言就是 API。在上一章中,我们学习了分别在 VB 应用程序和 HTML 文件中创建一个 C++ 解析器实例时,这些 API 是如何工作的。接下来你将一个 XML 文件装入解析器并验证它的有效性。在本章中,你将会学习同样的解析器是怎样为你的应用程序提供作为 DOM API 的属性和方法并且允许你操纵文件的。

在学习本章的内容之前,我们先来专门学习一下 DOM 和 W3C API。

15.2 文件对象模型

人们很容易主观地认为自己所用的 DOM 就是惟一的 DOM。实际上 DOM 有很多种。毕竟,模型仅仅是提供文件模型的一些计划。一个 DOM 应当是完整的,对文件最小的细节它都应当允许进行模型化而且可以访问。否则它就是不完整的,只允许对文件进行部分的访问。W3C DOM 是前者的一个例子,而 HTML 的 JavaScript DOM 则是后者的一个例子。

模型可能使用下面三个共同的样式:

- 线性模型。这是最简单的模型,它以一种线性的方式描述文件。就拿这本书来说,如果我说“翻到第 42 页,第 14 行,读第 3 个词”,你会觉得有点跟不上我的指令。我用的就是线性模型。然而,它有一个重要的缺陷,那就是对文件前一部分的修改会使对后面文件的所有引用都变得无效。
- 树形模型。这种模型用树的术语来描述文件。还是使用本书做比方,本段是本书第 15 章第 2 节的第 2 项。树形模型借用生物学和纹章学的语言来描述内容。树形模型讲述文件的根。树上每一项都被认为是节点,每一个终端节点被称做叶子。树中的节点和其他节点之间的关系用纹章学的语言来表述。父节点是相邻的前驱节点。前辈节点是指任何前驱节点。子节点是共享相同父节点的那些节点。一个关于树的流行概念是:文件的任何一部分都可以从文件的其他任何部分通过“遍历树”到达。树形模型与线性模型相比对修改就不那么敏感了,但是,它还是有些敏感的,因为如果整个节点被删除,那么与之相关的很多联系就都被改变了,而整个节点列表都必须重建。
- 对象模型。这是对修改最不敏感的一种模型。在对象模型中,文件的每一节都被称为属性。比如,你正在阅读的就是“XML 揭密”一书中“使用文件对象模型”一章“文件对象模型”一节的“对象模型”。无论文件如何改变,这个对象的描述总是有效。

树形和对象模型都是非常有用的模型, W3C DOM 用两者的一部分产生了它自己的混合的语法。本章的其他部分将集中介绍 W3C DOM Level 1 API, 不过未来的 API 扩展可能还会加入这些一般模型的其他部分。

注意: 扩展之一可能就是加入迭代程序, 它将允许应用程序遍历文件树。迭代程序和光标类似, 可以用来遍历树。迭代程序可以用 go to first sibling, go to parent, go to first child 之类的命令来定位。要取得内容可以对迭代程序进行定位, 然后读取紧跟在迭代程序后面的节点内容。迭代程序被用在处理数据库和访问记录集的应用之中。在第 21 章“XML 数据源对象(DSO)”和第 23 章“使用 XML 和 ASP 访问数据库”中还要重温这些内容, 那里我们将要处理数据源对象(DSO)和记录集。

在本章的其他部分里, 如果没有特别声明, 术语 DOM 特指 W3C DOM。

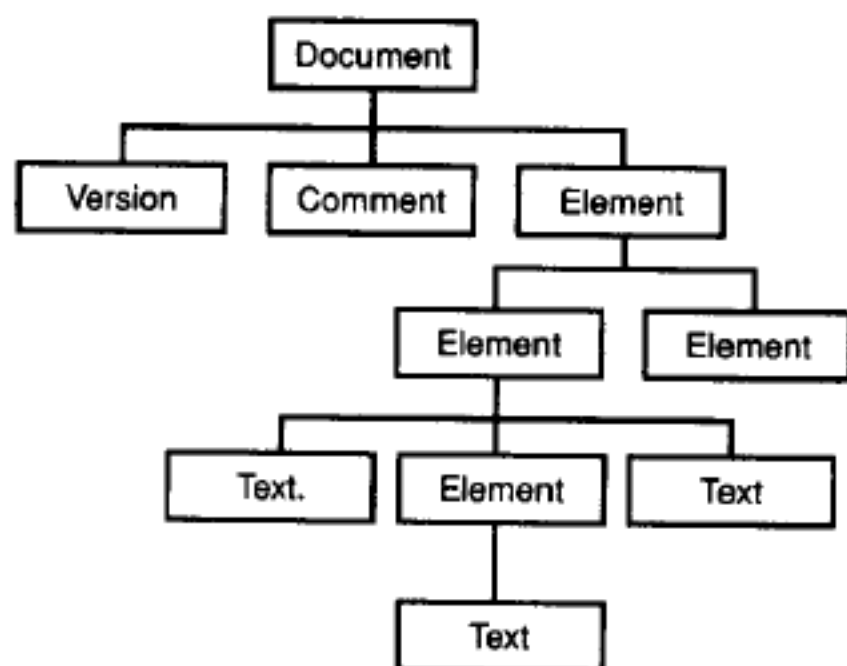
15.3 文件树和解析树

清单 15.1 和我们在第 12 章“XML 处理基础”中看到的 XML 文件一样。

清单 15.1 一个简单的 XML 文件

```
<? xml version = "1.0"? >
<! -demonstration of a parse tree-->
<xdoc>
  <greeting>Hello <emph>parsed</emph>XML!! </greeting>
  <applause type="sustained"/>
</xdoc>
```

这个文件的文件树显示在图 15.1 中。



A simple XML document tree

图 15.1 清单 15.1 中的文件生成的文件树

再次重申, 树根就是文件自身。这个根有三个子节点:

- 一个版本声明节点,或者在 DOM 用法中是处理指令节点。
- 一个注释节点。
- 一个元素节点。这个元素节点被称做文件的根节点。事实上,正如你刚才学到的,DOM 为你提供了一种专门的方法来访问这个节点。

15.4 W3C DOM 基础

W3C DOM 规范可以在 <http://www.w3.org/TR/REC-DOM-Level-1> 找到。W3C DOM 使用 OMG IDL(对象管理协会的接口定义语言)来描述它的工作。简单的说来,OMG IDL 是一种使用不同语言编写的应用程序可以用来相互通信的一种语言。在使用 IDL 的用法里,对象对外界提供多种接口。每一个接口都有一系列属性来描述在接口后面的对象的属性。对象可以通过一系列方法来操作,这些方法返回调用接口后面的功能,这些方法返还会向调用它的应用程序返回某种类型的结果。所有这些属性和方法一起构成了这个对象的 API。

在本节中,使用对象的应用程序将使用一般的语法。

访问和取得对象的属性,并将它读入到应用程序的 myVar 变量中的语法如下:

```
myVar = object_name.interface_name.interface_attribute
```

执行对象的一个方法或函数并将返回值保存在变量 dummy 中的语法如下:

```
Dummy = object_name.interface_name.interface_method(parameters)
```

如果没有结果返回,那么 dummy 就取得空值。

不幸的是,XML 属性名和 OMG IDL 的属性名之间存在着冲突,因为根据上下文,属性可能有着不同的含义。在 XML 中,它当然指的是元素的属性。在 OMG IDL 中,它指的是接口的特征。然而,如果你还记得 OMG IDL 的属性和对象的属性相类似,那么就没有问题了。通常情况下,属性这个词的含义是相当明确的。如果在本章中它不太明确,那它可能是指接口或 IDL 的属性,或者是元素或 XML 的属性。

15.4.1 DOM 对象

任何支持 DOM 的对象都必须能够装入 XML 文件。另外,它还必须能够用适当的 DOM 规范规定的属性和方法来提供所有的接口。和它接口的应用程序必须能够通过指定的语法来访问整个 XML 文件。

15.5 DOM 接口

DOM 定义了一些接口,其中的一部分比其他的更有用。每一个接口的方法和属性都将在本章中被仔细研究。

- DOMImplementation。这是对对象自身的一个查询,而且和文件是否装入无关。它将返回关于该对象所支持的 DOM 级别的信息。

- DocumentFragment。这是轻量级的一部分文件。
- Document。这个接口提供了关于被装入到对象中的主文件本身的信息。它提供了一些方法来创建节点和属性。
- Node。在文件中的所有内容都可以被看作是节点。元素是节点、注释是节点,诸如此类。这个接口包含了几个一般的方法和属性来操作任何类型的节点。没有什么可大惊小怪的,这些方法和属性可以被其他接口覆盖。比如,在处理元素的时候, nodeName 节点属性的返回值和 tagName 元素属性是一致的。
- NodeList。这是一个节点的有序集合,它可以通过一个索引来访问。
- NamedNodeMap。这是一个节点的集合,它可以通过名字来访问。元素的属性被作为 NamedNodeMap 来保存。
- CharacterData。这个接口处理文件的字符数据和文本。
- Attr。这个接口处理节点的 XML 属性。简化成 Attr 是为了避免和以前的版本发生混淆。
- Element。XML 中的大多数节点都是元素。这个接口具有处理元素及其 XML 属性的方法和特征。
- Text。这个接口处理元素的文本内容。它主要的功能要由 CharacterData 接口来完成。

下面的接口又自成一体,它们处理名字相关的节点。

- CDATASection
- Notation
- Entity
- EntityReference
- ProcessingInstruction

15.6 节点和对象

W3C DOM 使用双重引用来描述文件。对象模型中的所有东西都是:

- 一个节点
- 一个命名了的对象类型

如前所述,这导致了语法上的交叉。然而,这不一定是件坏事。通常,节点和对象都被表示在文件树中。然而,与某种节点相连的 Attr 节点未被表示在对象树中,而节点树则包含这种节点。

Attr 节点表示的是元素节点的 XML 属性。称之为 Attr 是为了和 IDL 属性相区别。Attr 对象不在树结构中。换句话说,它被当作是任何节点的子节点。也可以认为它是 Element 节点的一个属性。Attr 节点的内容可以通过 Element 元素的属性和方法来访问。

警告: DOM 工作组的 DOM 建议使用了 OMG IDL 的语法。这是一种允许应用程序及其对象相互通信的一种语言。换句话说,OMG IDL 是一种定义文件接口的语

言。

任何对象都具有一系列接口。这些接口为其他应用程序提供属性并且实现方法。这些属性实际上和对象的特征非常类似。事实上,属性和特征根本就是一个东西。你可能会被 W3C DOM 的上下文搞得晕头转向,因为元素的特征要通过属性及其取值描述,而属性又被用来描述 DOM 对象提供给其他应用程序的接口中的特征。当你阅读 W3C DOM 规范,看到属性这个词的时候,它又指的是节点的特征。无论何时看到 Attr,它都是指代表元素属性的一个节点。为了避免混淆,在属性一词的含义有可能发生混淆的时候,你可以既使用接口属性或(IDL)属性,也可以使用元素属性或(XML)属性这些术语。

15.7 访问 DOM 中的节点

有两种方法可以从 DOM 中取得节点。你可以遍历树或者用名字来直接访问。实际上,我们结合使用这两种方法来访问文件。

15.7.1 遍历树

为了遍历 DOM 树,要从树中的任何地方开始并且使用下面这些 DOM 节点接口的方法:

- parentNode()。这个方法可以访问父节点。
- firstChild()。这个方法可以访问该节点的第一个子节点。如果没有子节点就返回空。
- nextSibling()。这个方法访问下一个兄弟节点。如果这样的节点不存在就返回空。
- previousSibling()。这个方法访问上一个兄弟节点。如果这样的节点不存在就返回空。

很容易看到你是怎样使用这些方法来在整个文件树中移动的。讨论 Node 接口的时候,你将进一步学习如何使用这些方法。

15.7.2 用名字来访问节点

使用 getElementByTagName(elementname)来取得一个在文件或者是某一部分文件中具有这个名字的所有元素的列表。一旦创建了这样的 NodeList,就可以通过索引来访问这些命名了的节点了。实际上,这种方法使得用脚本或者程序在许多命名元素间实现循环成为可能。

15.8 DOM 方法返回的数据类型

无论何时使用 DOM 方法,都会返回某种类型的数据或对象。表 15.1 列出了 DOM 中所使用的数据和对象的类型。

表 15.1 DOM 中使用的数据和对象类型

| 数据类型 | 使用这种数据类型的节点描述 |
|----------------|---|
| unsigned short | 所有的节点类型是无符号短整数 |
| unsigned long | nodeList 或 NameNodeMap(例如它所包含项目的数目)的长度是无符号长整数 |
| Node | 这是包含所有节点细节的对象,包含有它的类型、名称、值、父节点、子节点等等 |
| NodeList | 节点对象索引列表的对象 |
| NamedNodeMap | 节点对象非索引列表的对象(实用中,它使用 Attr、实体或符号节点) |
| DOMString | 由统一字符编码标准字符组成的字符串 |
| Boolean | 真或假 |
| Void | 一些不返回值的方法 |

现在我们就来逐个仔细的讨论每一个接口,看一看它们特别的属性和方法。对每一个接口我们都使用一些简单的例子,通过运行这些例子,我们也可以看一看这些方法。

下面的几节里将会出现很多简单的例子。如果你想要运行这些例子,你需要一个实现了 DOM 的应用程序。Internet Explorer 5.0(IE5)浏览器是最常见的应用程序——实际上,它是目前惟一可用的应用程序。在 IE5 中运行这些例子最简单的办法是创建如清单 15.2 所示的 HTML 文件(它利用了 C++ XML DSO)。更详细的信息,请参阅第 21 章。

通常,你可以为这些例子使用 movies.xml。

清单 15.2 使用 DSO 的 HTML 文件

```

<html>
<title></title>
<xml id="isle">
<!--[paste XML file here]-->
<greeting type="official" manner="cordial">Hello DOM</greeting>
</xml>
<script>
myDoc=isle
//alert(myDoc.parseError.reason)
//[paste or type example code here]
var rootEl=myDoc.documentElement
var attlist=rootEl.attributes
document.write(attlist.item(0).nodeName)
</script>
</html>

```

下面的代码可以作为可选的一部分加入到上述代码中,它可以用 xml 元素的 src 属性来引用:

```
<xml id="isle" src="[uri of xml document]">
```

注意:这将创建一个 XML 数据源对象。除了允许数据绑定到元素上之外,XML DSO 还支持 DOM 脚本。你可以在第 21 章中学习 DSO。

请注意 xml 元素是 IE5 的一个特征元素。它不能在其他浏览器上工作。本节不是详尽无遗的,但是也涉及了我们日常可能使用的所有方法。更详细的信息请参阅规范本身。比如,我们没有讨论建议的错误代码。

15.9 Document 接口

Document 接口返回文件自身的信息。其中最常用的是 documentElement 属性,这个属性创建文件根元素的节点。如果你要遍历文件树来找到一个节点,这是一个不错的开始。document 接口具有下列只读属性:

doctype
implementation
documentElement

下面列出的 Document 接口方法中的前八个方法主要和不同类型的节点的创建有关。(你将会学到,第 9 个方法 getElementsByTagName() 也特别重要。)

这些方法是:

- createElement()
- createDocumentFragment()
- createTextNode()
- createComment()
- createCDATASection()
- createProcessingInstruction()
- createAttribute()
- createEntityReference()
- getElementsByTagName()

上面列出的方法中的最后一个属性对于通过名字访问元素来说非常重要。getElementsByTagName() 方法会建立一个包含了所有具有作为参数传给它的那个名字的文件元素的 NodeList。如果你想要用名字来访问元素,你可以使用这个方法。

15.9.1 Document 接口的只读属性

下面几节将讨论 Document 接口的只读属性。

doctype 属性

doctype 属性返回文件首部的“<! DOCTYPE”信息。在 DOM level 1 中,它不会返回 DTD 的信息。它返回实体和注释声明的信息,而且将它们作为子节点。要了解这是如何实现的,请参阅实体和注释接口一节。

警告: W3C DOM 规范将 doctype 属性小写。而在规范的其他部分,使用了一般的 camelback 标记(比如 nodeName)。至少有一个 DOM 的实现(Microsoft 的 DOM 引擎)会因此而产生错误!它要求使用 camelback 标记。希望这个 bug 会在软件的未来版本中得到纠正。

15.9.2 Implementation 属性

这个属性返回 implementation 节点。在 level 1 中,它有一个称做 hasFeatures() 的布尔方法。HasFeatures() 有两个参数:文件类型(合法值是 HTML 或 XML)和版本号。对于 movies.xml,下面的代码返回 false:

```
Dummy = myDoc.implementation.hasFeature("HTML", "4.0")
```

而下面的代码返回 true:

```
Dummy = myDoc.implementation.hasFeature("XML", "1.0")
```

documentElement 属性

如前所述,documentElement 属性是一个重要的属性,因为它返回文件的根元素对象。这是你为取得一个节点而遍历树的一个理想的起始点。

提示: 当你处理具有 DTD 要求相当严格的结构(如 movies.xml)的文件时,你可以确信下面的语句会返回你所期望的值(在这种情况下是第一个作者的名字):

```
myDoc.documentElement.firstChild.nextSibling.firstChild.nodeValue
```

如果你不能确认文件的结构,你最好还是用不同元素的名字特性来遍历文件,然后用 getElementsByTagName() 来访问元素节点本身。

15.9.3 Document 接口的方法

Create * () 方法创建如它们名字所示的节点。在创建的时候,这些节点都是孤儿。换句话说,它们还没有被放进文件树。只有在被一个 node 接口方法(比如 insertBefore() 或者 appendChild()) 收养之后,它才成为文件树的一部分。所有这些方法都返回如它们名字所示的节点。如果在创建过程中遇到了非法字符,createElement() 和 createAttribute() 方法返回错误。

createElement 方法

这个方法将新元素的标记名称作为参数。请注意被创建的元素对象可以接受属性及其取值。如果该名字不是合法的 XML 名字,DOM 实现应当返回错误。

createDocumentFragment() 方法

这个方法创建一个 documentFragment 节点。

createTextNode()、createComment() 和 createCDATASection() 方法

这三个方法创建如它们名字所示的节点。他们的参数都是将要成为节点内容的字符串。

createProcessingInstruction() 方法

这个方法有两个字符串参数:

- PI 对象作为第一个字符串。
- PI 数据作为第二个节点。

如果字符串中还有非法字符,该方法产生一个错误。

createAttribute()方法

这个方法创建具有给定名字的 Attr 节点。如果使用了非法字符就会产生一个错误。属性一旦被创建,就必须用 element 接口的 setAttribute()方法赋给一个元素。因为 setAttribute()方法允许你直接创建 Attr 节点,createAttribute()方法实际上很少使用。

createEntityReference()方法

这个方法的参数是实体引用的名字。如果名字中有非法字符,那么就产生一个错误。

getElementsByTagName()方法

这个方法取得所有具有所传参数指定的名字的后续元素的列表。它返回一个 NodeList 对象。这个方法是 Document 接口的方法中最有用的一个,它被广泛地使用在 XML 文件的检索过程中。

15.9.4 使用 Node 和 nodeList 接口

下面的例子稍稍前进了一步,它使用了 Node 接口和 nodeList 接口的一些方法。不过,它说明了如何创建一个元素并且将它插入到文件之中。清单 15.3 显示了我们使用的 XML 文件。

清单 15.3 一个简单的 XML 文件:greeting2.xml

```
<xdoc>
  <greeting>Hello Dom</greeting>
</xdoc>
```

清单 15.4 列出了我们使用的 HTML 文件。

清单 15.4 docinterface2.htm

```
<html>
<Head>
<title>Creating an element</title>
</head>
<body bgcolor=white">
<xml id="isle" src="../../greeting2.xml">
</xml>
<script>
  myDoc=isle
  var dummy
  if(myDoc.parseError! =0)
  {
    alert(myDoc.parseError.reason)
  }
```

```

else
{
    var dummy
    var rootEl=myDoc.documentElement
    var childList=rootEl.childNodes
    document.write(childList.length + "<br \>")
    document.write(rootEl.lastChild.nodeName + "<br \>")
    var newEl=myDoc.createElement("farewell")
    dummy=rootEl.appendChild(newEl)
    document.write(rootEl.lastChild.nodeName + "<br \>")
    document.write(childList.length + "<br \>")
}
</script>
</body>
</html>

```

注意：每一个列表都包含路径作为 XML 节点的 src 属性，该路径是与我的平台的配置相适应的。如果你要运行这些例子，你必须提供自己的路径。

我们来看看清单 15.4 中的代码。首先，创建了一个根元素的节点对象以及一个 nodeList 作为它的子节点。

```

Var rootEl = myDoc.documentElement
Var childList = rootEl.childNodes

```

childNodes 是 node 接口的一个属性，它返回一个 nodeList 对象。这个列表的长度以及列表中最后一个元素的名字被打印出来。长度是 1 而元素名是 greeting：

```

document.write(childList.length + "<br \>")
document.write(rootEl.lastChild.nodeName + "<br \>")

```

这时用 node 接口的一个方法来创建一个元素：

```

var newEl = myDoc.createElement("farewell")

```

我们刚刚看到的 createElement 方法生成的元素被加入到根元素的孩子中：

```

dummy = rootEl.appendChild(newEl)

```

这时再打印 NodeList 的长度以及最后一个孩子的名字以证明已经创建了这个节点：

```

document.write(childList.length + "<br \>")
document.write(rootEl.lastChild.nodeName + "<br \>")

```

请注意孩子列表的长度被自动更新了。

注意：根据计算资源的情况，更新 NodeLists 可能是一个代价高昂的操作。在具有若干很长 NodeList 的大型应用程序中，这会显著减慢执行的速度。在设计应用程序的时候，你一定要牢记这一点。同时，请参阅下面的 DocumentFragment 接口一节。实际上，DocumentFragment 接口提供了一种更新文件的一小部分而无需不断地更新这个文件的 NodeList 和 NameNodeMaps 的方法。

15.9.5 使用 getElementsByTagName() 方法

清单 15.5 显示了如何使用 getElementsByTagName() 并且打印一个 movies.xml 中所有电影标题的列表。

清单 15.5 创建一个电影标题的列表: movies.xml

```
<xml id="isle" src="../../movies.xml">
</xml>
<script>
myDoc=isle
var dummy
if (myDoc.parseError! =0)
{
    alert(myDoc.parseError.reason)
}
else
{
    var movieList=myDoc.getElementsByTagName("movie")
    for (var i=0;i < movieList.length;i++)
    {
        document.write(movieList.item(i).firstChild.firstChild.nodeValue)
        document.write("<br \>")
    }
}
```

在这个例子中,我再次使用了一些尚未涉及的 document 接口的方法。首先,建立一个所有 movie 元素的节点列表:

```
var movieList = myDoc.getElementsByTagName("movie")
```

然后用一个简单的循环来打印它们。你通过 nodeList 接口的 item() 方法来访问单独的节点。

```
movieList.item(i)
```

接下来你用 node 接口的属性来遍历树,直到你得到含有你所需要的信息的文本节点为止。

```
firstChild.firstChild.nodeValue
```

15.10 DocumentFragment 接口

DocumentFragment 接口是一个有用的接口,它使你可以在文件的一小部分上工作。如果你创建了许多新元素,这特别的有用。如果你在主文件中创建这些新元素,则每次创建一个新节点,整个 NodeList 就需更新一次。使用 documentFragment 接口,你可以创建文件的一个完整的小部分,而只在你将文件片段插入到文件中去的时候,NodeList 才更新一次。

DocumentFragment 接口使我们可以在文件的一个独立的部分上工作,这的确带来了不少便利。

15.11 node 接口

node 接口是 DOM 中的关键接口。文件中的所有内容都可以被认为是节点,包括文件本身。正是由于它的重要性,本节将要更仔细地讨论这个接口。

下面是 node 接口的只读属性:

- nodeName
- nodeValue
- nodeType
- parentNode
- childNodes
- firstChild
- lastChild
- previousSibling
- nextSibling
- attributes
- ownerDocument

下面是 node 接口的方法:

- insertBefore()
- replaceChild()
- removeChild()
- appendChild()
- hasChildNodes()
- cloneNode()

请注意所有这些方法和属性都是区分大小写的。

15.11.1 node 接口的属性

node 接口包含了所有接口的大部分属性和方法。

nodeType 属性

W3C DOM 中描述了 12 种节点类型。每一种节点类型都用一个无符号的短整数来表示。对于每一个节点也定义了相应的常量。下面是所有这些不同的节点类型:

- Element。这是在文件树中可能遇到的最常用的节点类型。它也是惟一一种有孩子的元素类型。所有的其他节点都是树中的叶子。元素还具有属性。这些属性被当作节点,不过 W3C DOM API 中没有把它们当作树的一部分,而是把它们当作了元素的一部分。换句话说,他们是元素的特征。

- Attr。一个 Attr 节点表示了元素的一个属性。它没有被当作是元素的孩子。相反,它被当作是元素的特征。选择术语 Attr 是为了避免和接口的属性发生混淆。下面的语法为所有的元素属性建立了一个 NamedNodeMap。

[NamedNodeMap object] = [element node object]. attributes

- Text。在 DOM 中,文本节点通常是叶子。通常它还是元素的孩子。
- CDATA 段。在 DOM 中,这种节点通常是叶子对象。

下面的节点将在讨论相应的接口时介绍:

- Entity reference
- Entity NODE
- Processing instruction
- Comment
- Document
- Document type
- Document fragment
- Notation

表 15.2 节点类型以及定义的常量

| 节点类型 | 整数 | 定义的常量值 |
|------------------------|----|-----------------------------|
| Element | 1 | ELEMENT_NODE |
| Attr | 2 | ATTRIBUTE_NODE |
| Text | 3 | TEXT_NODE |
| CDATA section | 4 | CDATA_SECTION_NODE |
| Entity reference | 5 | ENTITY_REFERENCE_NODE |
| Entity node | 6 | ENTITY_NODE |
| Processing INSTRUCTION | 7 | PROCESSING_INSTRUCTION_NODE |
| Comment | 8 | COMMENT_NODE |
| Document | 9 | DOCUMENT_NODE |
| Document type | 10 | DOCUMENT_TYPE_NODE |
| Document fragment | 11 | DOCUMENT_FRAGMENT_NODE |
| Notation | 12 | NOTATION_NODE |

nodeName 和 nodeValue 属性

表 15.3 列出了各种不同节点类型的 nodeName 和 nodeValue。

表 15.3 不同节点类型的名字和值

| 节点类型 | 节点名 | 节点值 |
|------------------------|--------------------|-------------------------|
| Element | 标记名 | 空 |
| Attr | 属性名 | 属性的值 |
| Text | #text | Text 节点内容 |
| CDATASection | #cdata-section | CDATA 段内容 |
| Entity Reference | 实体名引用 | 空。实体引用的替换值作为这个节点的第一个子节点 |
| Entity | 实体名 | 实体的替换值作为这个节点的第一个子节点 |
| Processing instruction | PI 的目标 | 除目标之外的整个内容 |
| Comment | #comment | 注释的内容 |
| Document | #document | 符号的替换值作为这个节点的第一个子节点 |
| Document Type | 文件类型名 | 空 |
| Document Fragment | #document Fragment | 空 |
| Notation | 符号名 | 空 |

parentNode 属性

除了 Document, documentFragment, entity, notation 和 Attr 之外的所有节点都有父节点。

childNodes 属性

这个属性返回这个节点的所有子节点的 NodeList。为了防止你的程序或者脚本产生错误,最好在条件语句中使用 hasChildNodes() 方法来检查子节点是否真的存在。

firstChild, lastChild, previousSibling 和 nextSibling 属性

这些属性的名字就说明了它们的返回值——具有相应位置关系的节点对象。如果这种关系不存在,那么就返回空。

attributes 属性

这个(IDL)属性只对元素类型的节点有效。所有其他的节点都会返回空值。这个属性返回包含所有属性的一个 NamedNodeMap。

ownerDocument 属性

这个属性将文件对象作为一个节点返回。

15.11.2 节点属性特征的调查

清单 15.6 使用了一个简单的递归函数打印了 movies.xml 文件树中的所有节点。每一个节点打印了节点类型、节点名称和节点值。

清单 15.6 递归调用节点:nodeattributes1.htm

```

<html>
<head>
<title>Calling Nodes Recursively</title>
</head>
<body bgcolor="white">
<xml id="isle" src="../../movises.xml">
</xml>
<script>
  myDoc=isle
  var dummy
  if(myDoc.parseError! =0)
  {
    alert(myDoc.parseError.reason)
  }
  else
  {
    dummy=getChildNodes(myDoc)
  }

  function getChildNodes(x)
  //this is a recursive function that reads all the child nodes of a node
  //and calls it self if the node can have children
  {
    var chlist
    chlist=x.childNodes
    for (var l=0;l<chlist.length;l++)
    {
      //alert(chlist.item(i).nodeName)
      document.write("Node Type:-"
        + chlist.item(i).nodeType
        + " Node Name:(b)" + chlist.item(i).nodeName
        + "</b> Node Value: - <b>" + chlist.item(i).nodeValue + "</b><br \
>")

      if (chlist.item(i).nodeType==1 && chlist.item(i).hasChildNodes()==true)
      {
        document.write("<br \>CHILDREN OF <b>"
          + chlist.item(i).nodeName
          + "</b> Element Node <br =\>")
        getChildNodes(chlist.item(i))
      }
      if(chlist.item(i).nodeType==10 && chlist.item(i).hasChildNodes()==true)
      {
        document.write("<br \>CHILDREN OF <b>"
          + chlist.item(i).nodeName
          + "</b> Document Type Node <br\><br \>")
        getChildNodes(chlist.item(i))
      }
    }
  }
</script>

```



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

这个 JavaScript 非常直接,但是要注意以下几点:

- `getChildNodes(x)` 是一个递归函数。它遍历节点的每一个子节点,打印每一个节点的详细内容,并且看看这个节点自己是否还有子节点。如果有,那么就将这个节点作为参数调用这个函数自身。
- `getChildNodes(x)` 函数产生了一个所有它经过的节点的子节点的列表,然后,遍历它们,打印它们的详细信息:

```
var chlist
```

```
chlist = x.childNodes
```

- 如果节点是一个元素节点或者文件类型的节点(那些有孩子的节点类型),它首先测试该节点是否真的有子节点。如果有,就把这个节点返回给函数。比如:

```
if(chlist.item(i).nodeType == 1 && chlist.item(i).hasChildNodes() == true)
{
    document.write("<br \>CHILDREN OF <b>"
    + chlist.item(i).nodeName
    + "</b>Element Node <br \><br \>")
    getChildNodes(chlist.item(i))
}
```

这是 JavaScript 很好的一点。请注意你必须回避 XHTML 版本中行分隔符 `<br \><br \>` 中的斜杠。这段代码的输出显示在图 15.2 中。

15.11.3 node 接口的方法

node 接口的方法——除了 `hasChildNodes()` 和 `cloneNode()` 方法之外——都使用一个或者若干节点对象作为参数。通常,当你使用 `insertBefore()`, `replaceChild()` 或者 `appendChild()` 方法的时候,某一个文件方法或者是 `cloneNode()` 方法会创建一个新节点。这个节点用 `new_node` 来引用。

insertBefore() 方法

这个方法将新的子节点插入到引用子节点之前并返回 `new_node`:

```
dummy = node-object.insertBefore(new_node,reference_node)
```

这时 `dummy` 包含被插入的节点对象的一个副本。

replaceChild() 方法

这个方法替换引用的子节点并返回被替换的节点:

```
dummy = node-object.replaceChild(new_node,reference_node)
```

这时 `dummy` 包含被替换的那个原来的节点的一个副本。

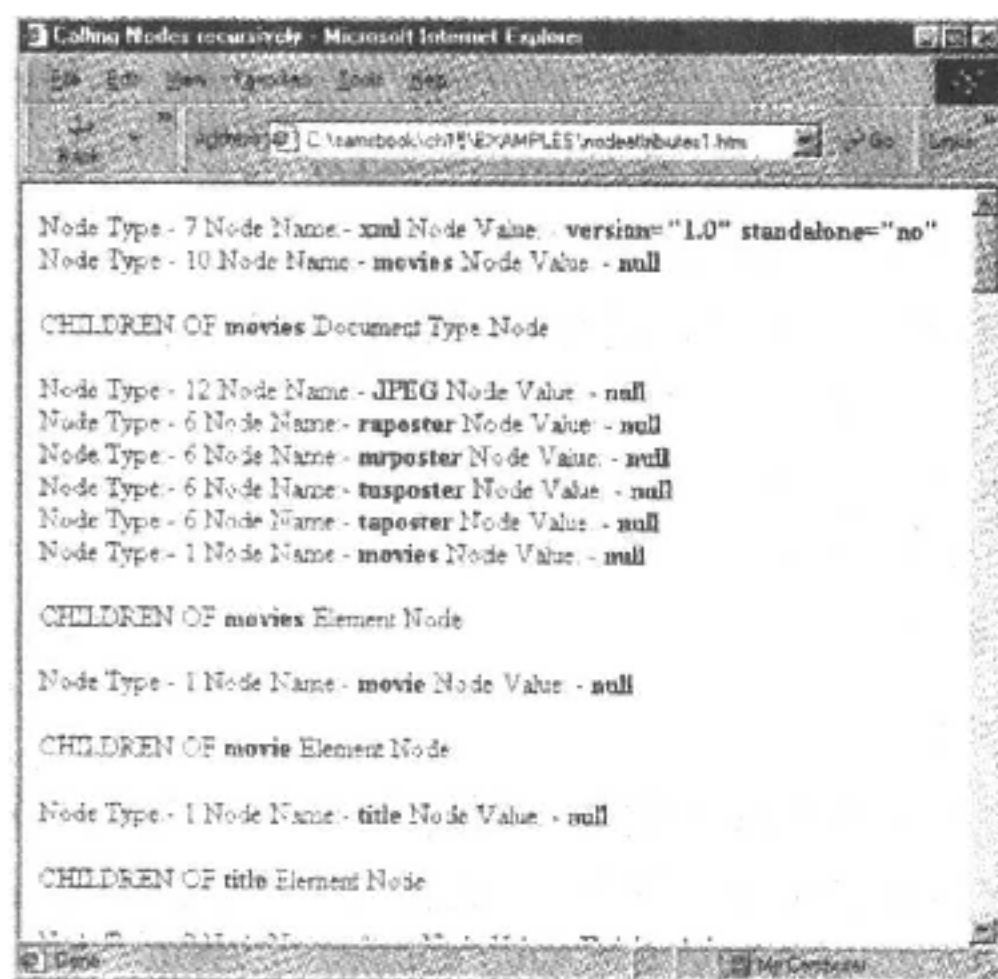


图 15.2 清单 15.6 的屏幕显示

removeChild()方法

这个方法删除引用的子节点并返回被删除的节点：

```
dummy = node-object.removeChild(reference-node)
```

这时 dummy 包含被删除的节点的部分。

appendChild()方法

这个方法将新节点加入到其他子节点的后面并且返回新节点：

```
dummy = node-object.appendChild(new-node)
```

这时 dummy 包含新节点的一个副本。

hasChildNodes()方法

这个方法返回一个布尔值，它是给定节点是否有子节点的测试结果。

cloneNode()方法

cloneNode()方法建立被克隆的节点的一个副本。一个元素被克隆的时候，它使用 true 或 false 做参数，这个参数指出了克隆的深度。在 movie.xml 的例子中：

```
Var dummy = myDoc.documentElement.firstChild.cloneNode(true)
```

除了克隆 movie 元素本身之外，还会克隆它所有的内容，而下面的代码仅仅克隆元素本身：

```
var dummy = myDoc.documentElement.firstChild.cloneNode(false)
```

请注意克隆了的节点是一个孤儿。换句话说，如果在上一个例子中使用 dummy.getParent，那么将会返回空值。

15.11.4 使用节点方法:示例 A

在这个例子中,我们创建一个注释并把它当作子节点加入到 movies.xml 的根元素中。如果你想要运行这个例子,请用清单 15.7 中的代码修改清单 15.6。

清单 15.7 用 DOM 建立注释:nodeattribute2.htm

```
if (myDoc.parseError != 0)
{
    alert(myDoc.parseError.reason)
}
else
{
    //make a new node
    var new_comment=myDoc.createComment(" A brand new comment!!")
    //append this comment to the end of movies children
    var rootEl=myDoc.documentElement

    dummy=rootEl.appendChild(new_comment)
    //dummy now contains
    alert(dummy.nodeValue)
    dummy=getChildNodes(myDoc)
}
```

这个警告会告诉我们 dummy 是否真的含有了新注释的值。图 15.3 显示了注释是如何被加入到文件的最后的。

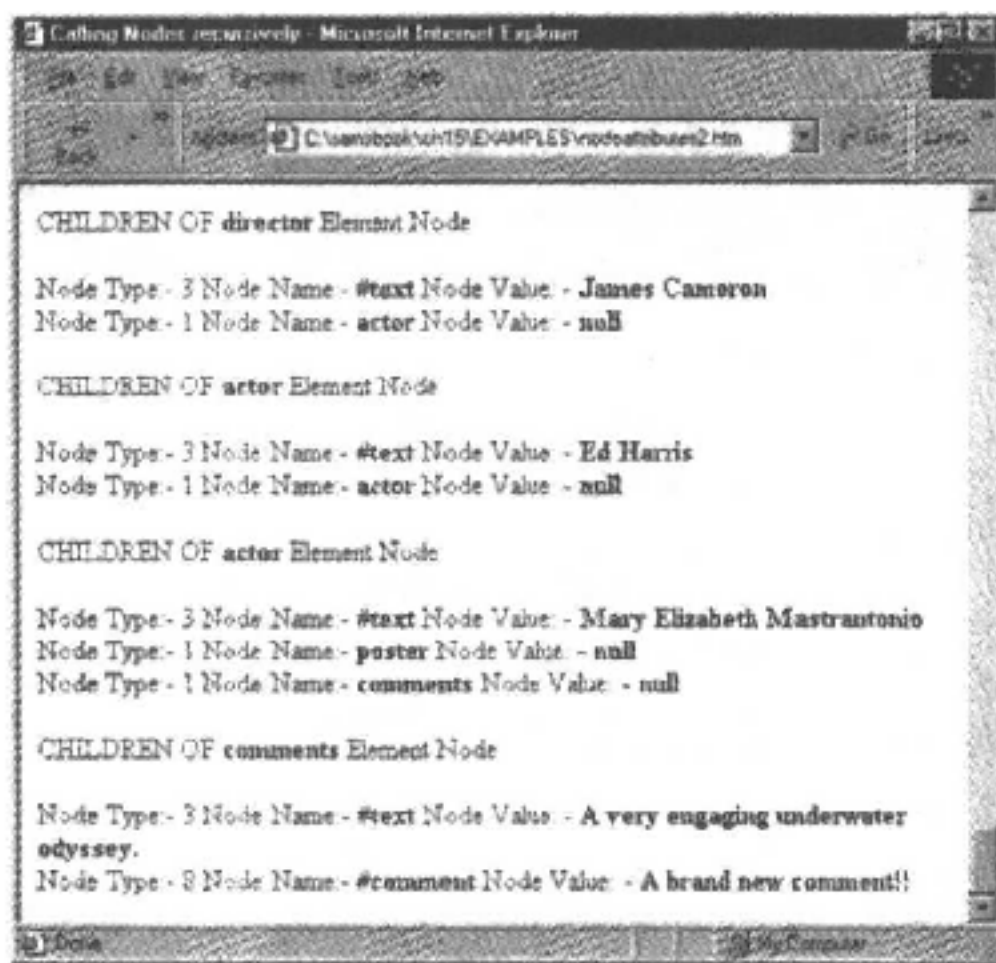


图 15.3 清单 15.7 的结果,显示了加入注释的文件树

15.11.5 使用节点方法: 示例 B

在这个例子中, 我们创建 movies.xml 中第一个 movie 元素的一个深克隆, 并且将它作为 movies.xml 根元素的子节点插入到文件中。如果你想要运行这个例子, 对清单 15.6 的 script 元素进行如下修改。

清单 15.8 创建一个深克隆

```
if (myDoc.parseError != 0)
{
    alert(myDoc.parseError.reason)
}
else
{
    //make a new node
    var new_node = myDoc.documentElement.firstChild.cloneNode(true)
    //append this comment to the end of movies children
    var rootEl = myDoc.documentElement

    dummy = rootEl.appendChild(new_node)
    alert(dummy.nodeName)
    dummy = getChildNodes(myDoc)
}
```

这里我们建立了 movies.xml 中第一个 movie 元素的一个深克隆。图 15.4 不但显示了这个 movie 元素如何被加入, 还显示了它所有的内容是如何加入的。作为练习, 你可能想要将前一个例子中的参数修改为 false 并再次运行它。这一次的结果显示在图 15.4 中。

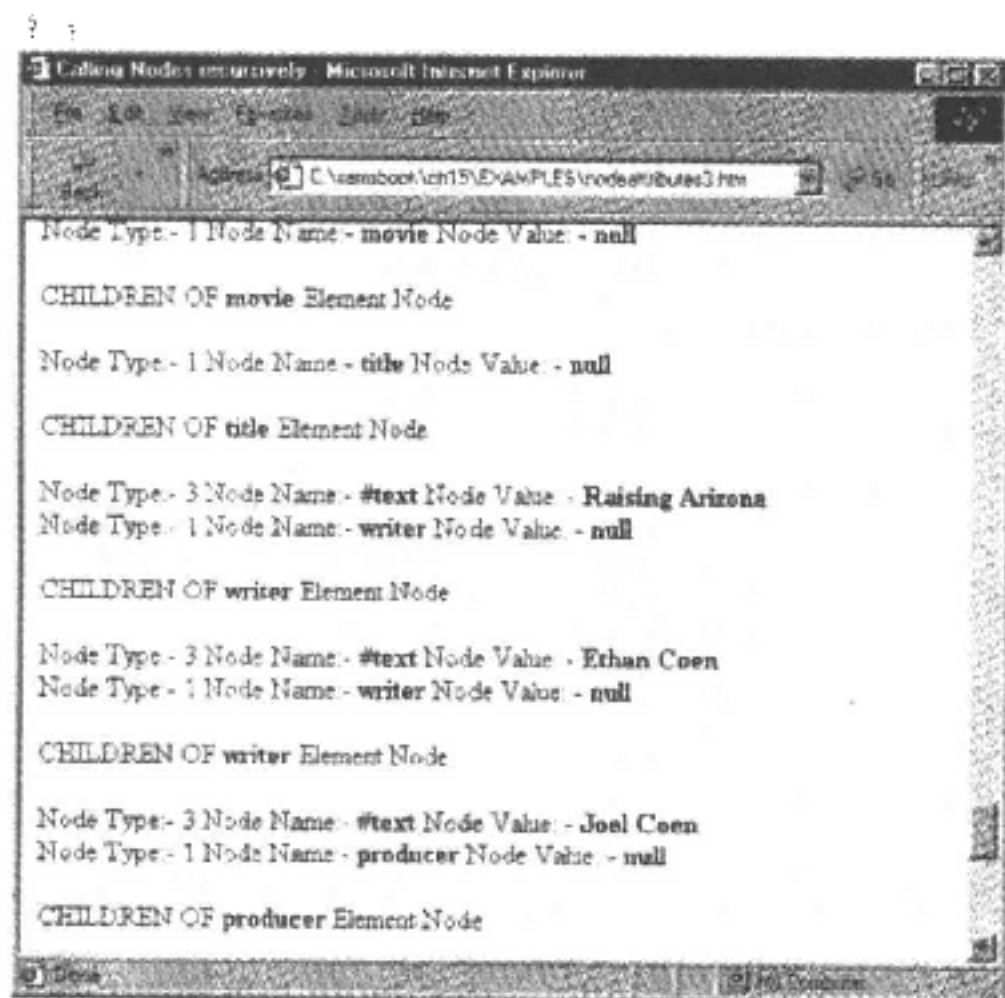


图 15.4 清单 15.8 的结果, 显示了被加入到文件树中的克隆后的 movie 元素

15.12 NodeList 接口

你已经在前面的例子中使用了 `nodeList` 接口的属性和方法。`getElementsByTagName()` 方法和 `childNodes` 属性都会创建一个 `NodeList`。实际上,这是一个可以通过 `NodeList` 接口的 `item()` 方法来访问的列表。

15.12.1 `item()` 方法

这个方法将一个索引作为参数并且返回这个索引位置上的节点。如果这个索引超出了范围,那么就返回空值。请记住,计数器总是从零开始。

length 属性

这个属性返回一个无符号的长整数,并指出了 `NodeList` 中节点的个数。

15.13 NamedNodeMap 接口

当使用 `node` 接口的 `attributes` 属性时,会返回 `NamedNodeMap`。尽管为了方便, `NamedNodeMap` 中的项是有索引的,每一项的位置却不像 `NodeList` 那么重要。在命名的节点映射中的项通常用名字来访问。

15.13.1 `getNamedItem()`, `setNamedItem()` 和 `removeNamedItem()` 方法

顾名思义,这些方法获取、增加和删除给定名字的节点。`setNamedItem()` 要求将节点对象当作参数。其他两个方法的参数是字符串。

item() 方法(已经弃用)

这个方法返回节点对象的索引。

15.13.2 **length** 属性

返回 `NamedNodeMap` 的长度。清单 15.9 是使用 `NamedNodeMap` 接口的一个例子。

清单 15.9 使用 `NamedNodeMap` 接口: `namednodemap1.htm`

```
<html>
<head>
<title>NamedNodeMaps</title>
<style>
body{font-size:16pt;}
</style>
</head>
<body bgcolor="white">
<xml id="isle" src="../../movies.xml">
</xml>
<script>
myDoc=isle
```

```

var dummy
if (myDoc.parseError! = 0)
{
    alert(myDoc.parseError.reason)
}
else
{
    var movieList=myDoc.getElementsByTagName("movie")
    //make a NamedNodeMap of the first movie tag
    var NNMap=movieList(0).attributes
    for (var i=0;i<NNMap.length;i++)
    {
        document.write(NNMap.item(i).name + "=" + NNMap.item(i).value + "<br \
>")
    }
    document.write("<br \>The year attribute will now be removed<br \><br \>")
    var dummy=NNMap.removeNamedItem("year")
    for (var i=0;i<NNMap.length;i++)
    {
        document.write(NNMap.item(i).name + "=" + NNMap.item(i).value + "<br \
>")
    }
}
</script>
</body>
</html>

```

清单 15.9 中的代码实际上非常简单。它为第一个 movie 元素的所有属性建立了一个 NamedNodeMap:

```
var NNMap = movieList(0).attributes
```

然后循环打印每一个属性的名字和值:

```

for (var i=0;i<NNMap.length;i++)
{
    document.write(NNMap.item(i).name + "=" +
NNMap.item(i).value + "<br \>")
}

```

删除 year 属性:

```
var dummy = NNMap.removeNamedItem("year")
```

再次遍历这个列表,看一看你的修改是否成功。
输出显示在图 15.5 中。

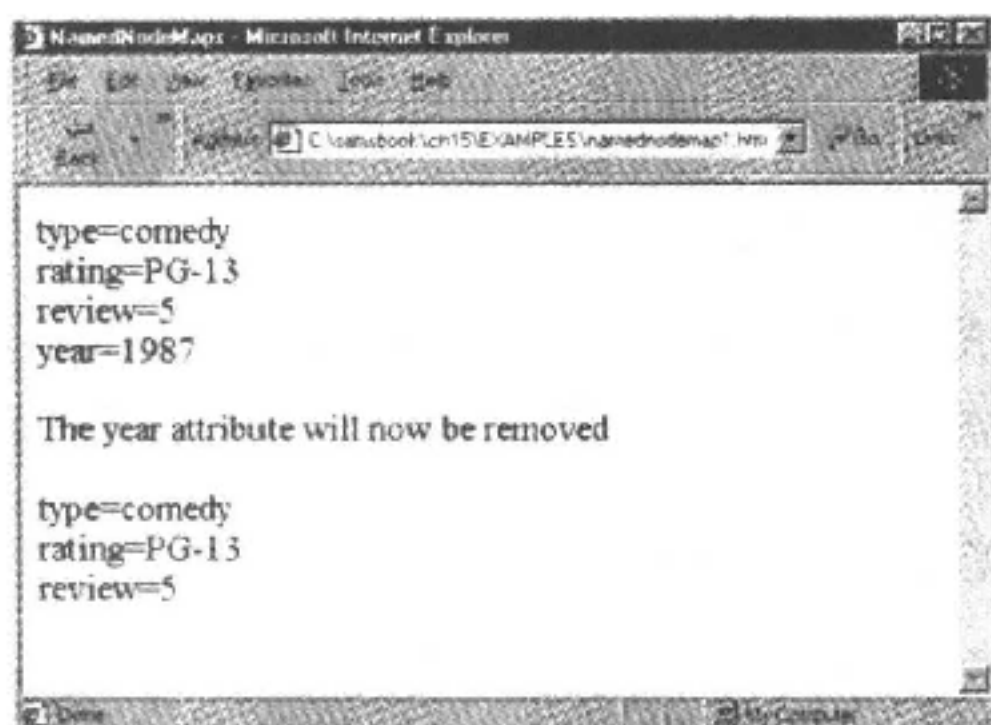


图 15.5 清单 15.9 的屏幕输出

15.14 CharacterData 接口

CharacterData 接口包含访问和编辑接口中字符串部分的属性和方法,无论这些字符串是文本节点,注释节点还是其他带有字符串值的节点都是如此。

15.14.1 data 属性

data 属性以 Unicode 串返回这个节点中的所有内容。

15.14.2 length 属性

length 属性返回这个字符串中字符的个数。

15.14.3 substringData 方法

这个方法根据要求返回字符串的一个子串。它有两个参数,都是无符号长整数。第一个参数是偏移量参数,它指出子串从什么地方开始。第二个参数是计数参数,它指出返回起始位置后多少个字符。

15.14.4 appendData()方法

这个方法的参数是一个字符串,它将被追加到这个节点的后面。

15.14.5 insertData()方法

这个方法有两个参数:偏移量参数指出了插入数据的位置,而字符串参数是被插入的字符串。

15.14.6 deleteData()方法

这个方法用两个整数作为参数：偏移量参数和计数参数。

15.14.7 replaceData()方法

这个方法有三个参数：偏移量、指出替换多少字符的计数器以及替换字符串。

下面的例子使用两个方法打印了 dreamexerpt.xml：传统的 JavaScript 函数和使用 DOMCharacterData 接口提供的方法的递归函数。

清单 15.10 是《仲夏夜之梦》的摘录。

清单 15.10 《仲夏夜之梦》的摘录：dreamexerpt.xml

```
<DREAM>
<SPEECH speaker="TITANIA">
These are the forgeries of jealousy:
And never,since the middle summer's spring,
Met we on hill,in dale,forest or mead,
By paved fountain or by rushy brook,
Or in the beached margent of the sea,
To dance our ringlets to the whistling wind,
But with thy brawls thou hast disturb'd our sport. </SPEECH>
</DREAM>
```

清单 15.11 是说明 CharacterData 方法的 HTML 文件。

清单 15.11 演示 CharacterData 接口的属性和方法：Character1.htm

```
<html>
<Head>
<title>CharacterData Interface</title>
<style>
body{font-size:16pt;}
</style>
</head>
<body bgcolor="white">
<xml id="isle" src="../../dreamexerpt1.xml">
</xml>
<script>
myDoc=isle
var dummy
if (myDoc.parseError! =0)
{
    alert(myDoc.parseError.reason)
}
else
{
    var speech=myDoc.documentElement.firstChild.firstChild.data
    var dummy=printdata(speech)
```

```

var dummy2=printdata2(myDoc.documentElement.firstChild.firstChild)

function printdata(x)
{
    while (x.indexOf("\n",0) != -1)
    {
        var nl=x.indexOf("\n",0)
        document.write(x.substring(0,nl) + "<br \>")
        x=x.substring(nl+1)
    }
}

function printdata2(x)
{
    var str=x.data
    var slen=str.indexOf("\n",0)
    if(slen != -1)
    {
        document.write(x.substringData(0,slen) + "<br \>")
        x.deleteData (0,slen+1)
        dummy=printdata2(x)
    }
}
</script>
</body>
<html>

```

这段代码的主体被包含在一个文本节点中,这个节点可以以下面的方法访问:

```
myDoc.documentElement.firstChild.firstChild
```

你可以用两种方法将它打印出来。第一种办法是在 printdata() 中嵌入传统的 JavaScript 循环。

为了使用这个函数,要用 CharacterData 接口的 data 属性来从节点中提取字符串内容(尽管你也可以使用 node 接口的 nodeValue 属性)。

```
var = speech = myDoc.documentElement.firstChild.firstChild.data
```

它先被传给第一个函数,在这里用传统的 JavaScript 循环来找到每一个回车并且打印 Titania 的讲话。

第二种方法是用使用 DOM 方法的递归函数。

首先,将文本节点传给函数。这个函数找到文本节点的第一行:

```

function printdata2(x)
{
    var str = x.data
    var slen = str.indexOf("\n",0)

```

然后用 substringData() 方法将第一行打印出来:

```
document.write(x.substringData(0,slen) + "<br \>")
```

接下来这个函数将这一行文本以及回车从节点中删除：

```
x.deleteData(0,slen+1)
```

最后，用修改后的节点作为参数调用自身。

```
dummy=printdata 2(x)
```

实际上，JavaScript 函数的效率比第二个函数更高——考虑到计算机资源的利用。然而，这里的目的仅仅是为了说明如何使用 CharacterData 接口。

15.15 Attr 接口

这个接口表示了元素对象的属性。Attr 对象不被当作文件树的一部分，相反，它们被当作是文件树中元素的特征。

15.15.1 name 属性

name 属性返回属性的名称。

15.15.2 value 属性

value 属性返回属性值。

15.15.3 specified 属性

这是一个布尔属性。无论是在原始文件还是代码，如果这个属性被赋了值，specified 属性就返回 true。

如果这个属性在 DTD 中有默认值，或者这个值是 #FIXED 或由一个枚举属性提供的默认值，那么 specified 的值为 false。

15.16 Element 接口

文件中的大多数节点都属于 element 或是 text 接口。然而，这两种接口都不是特别巨大。element 节点上的大部分操作——比如发现关系和编辑 element 节点——都是由 node 接口完成的。文本节点上的大多数操作则是由 CharacterData 接口实现的。

除了很少的几个属性和方法之外，大多数 element 接口的方法都和管理元素的 XML 属性特征有关。

在 XML DOM 中，XML 属性被看作是单个元素的特征而不被包含在文件树结构模型之中。GetAttribute() 方法是一个接受属性名字符串并以 DOMString 返回属性值的简单方法。在很多情况下，只需要这些即可。然而，XML 属性可以含有实体而且可以具有它们需要的复杂结构。当你处理这一类属性的时候，最好是用 Attr 节点处理这些属性而不是把它们当作简单的字符串。Node 接口的 Attributes IDL 属性会返回含有该元素所有 Attr 节点的 NamedNodeMap。element 接口的 getAttributeNode()，setAttributeNode() 和 removeAttributeNode() 方法是对它的补充。

下面是一些和处理元素属性相关的方法：

- `getAttribute()`
- `setAttribute()`
- `removeAttribute()`
- `getAttributeNode()`
- `setAttributeNode()`
- `removeAttributeNode()`

`element` 接口还复制了重要的 `getElementsByTagName()` 方法。同样,它还有自己的 `tagname` 属性,这个属性是 `node` 接口的 `nodename` 属性的副本。它还有个十分奇怪的 `normalize()` 方法。下面就是 `element` 接口其他的一些属性和方法。

- `Tagname` 属性
- `GetElementsByTagName()` 方法
- `Normailze()` 方法

15.16.1 处理属性的方法

下面介绍用来操纵 XML 属性的方法。

getAttribute()方法

这个方法通过名字来取得属性的值。它的参数是一个简单的 `DOMString`,它的返回值也是一个简单的 `DOMString`。

setAttribute()方法

这个方法实际上被用来创建属性并设置属性值。两个参数都是 `DOMString`。

注意: `Document` 接口的 `createAttribute()` 方法可以被用来创建属性,但是用 `setAttribute()` 来在创建的同时设置属性值则更为简单。

removeAttribute()方法

这个方法将具有给定名字的属性删除。请注意它和 `NamedNodeMap` 接口的 `removeNamedItem()` 方法是一样的。

getAttributeNode()方法

如果传给它一个名字,它就返回一个 `Attr` 节点。

setAttributeNode()方法

如果传给它一个 `Attr` 节点,它就设置一个属性。这个 `Attr` 节点可能已经用 `Node` 接口的 `cloneNode()` 方法或者 `Document` 接口的 `createAttribute()` 方法创建。这个方法没有返回值。

removeAttributeNode()方法

如果传给它节点的名字,实际的 `Attr` 节点对象就被删除。

15.16.2 其他 element 接口的属性和方法

这里是 element 接口的其他属性和方法,其中的一些你已经见过了。

getElementsByTagName()方法

这个方法用在 document 接口和用在 DocumentFragment 接口上具有相同的语法。在 element 接口中,它仅仅返回一个元素节点中具有这个名字的后续节点的 nodeList,而不是整个文件。

tagname 属性

这个属性复制了 node 接口中 nodeName 属性的功能。请注意出于某种原因对这个属性 W3C 工作组没有用 camelback 风格。不幸的是,Microsoft 的解析器用了 camelback 风格。所以如果你在 Microsoft 的解析器中使用了正确的标记,你可能得到一个错误。

normalize()方法

如果 factory 方法创建了一个或多个相邻的文本子节点而不影响元素的建立,那么尽管它们是彼此相邻的,但是仍然作为独立的节点存在。normalize()方法可以将它们结合成为一个节点。

15.17 不常用的 DOM 接口

下面的接口很少使用。如果想要更多的信息请参阅 W3C 建议。

15.17.1 text 接口

尽管文本节点会在特定的文件中构成大量的节点,但是大部分文本的操作都是由 CharacterData 接口来完成的。实际上,text 接口只有一个方法,这就是 splitText()方法。这个方法接受一个偏移量——一个无符号长整数——作为参数,它将一个单独的文本节点分成两个文本节点。

15.17.2 CDATASection 接口

CDATASection 接口没有它自己的属性和方法,它继承了 CharacterData 接口的属性和方法。

15.17.3 DocumentType 接口

每一个文件都有一个 doctype 属性。如果前言中声明了一个<!DOCTYPE,那么这个接口的属性会返回这个值。如果没有<!DOCTYPE 声明,那么这个接口返回的值就是空。DocumentType 接口有下列属性。

- name
- entities
- notations

name 属性

这个属性返回一个指明 DTD 名字的 DOMString,这也是根元素名字。

entities 属性

这个属性返回 DTD 中声明的所有实体(无论是内部的还是外部的)的一个 NamedNodeMap。

notations 属性

这个属性返回 DTD 中声明的所有标志的一个 NamedNodeMap。

15.17.4 Notation 接口

用 DOM 可以访问注释。注释声明可以被当作 documentType 的特征。一个注释声明的 NamedNodeMap 可以通过下面的语法得到:

```
[NamedNodeMap object] = [documentType object].notations
```

注释的个数可以用 length 属性取得。

```
[Unsigned Long integer] = [NamedNodeMap object].length
```

这样我们就可以用 DOM 的 item()方法来以通常的方式访问注释了。语法如下:

```
[Notation object] = [NamedNodeMap object].item(0)
```

这个语句取得第一个注释对象。

Notation 属性除具有通常节点的属性外还有两个特殊属性:

- PublicId。它标识了任何 PUBLIC 标记。
- SystemId。它表示了任何 SYSTEM 标记。

为了返回 NOTATION 的名字,可以使用 nodeName 属性。清单 15.12 明确了这些概念。

清单 15.12 使用注释,ex-notations1.htm 中的 XML 文件

```
<! DOCTYPE greeting [  
  <! ELEMENT greeting (#PCDATA)>  
  <! NOTATION gif PUBLIC "someidentifier">  
]>  
<greeting>Hello DOM &ent;</greeting>
```

清单 15.13 打印了节点名 gif 和公共标识符 someidentifier。

清单 15.13 HTML 文件中说明 Notation 接口的部分:ex-notations1.htm

```
//myDoc is an object variable with the  
//above xml document assigned to it  
var rootEl  
var docType
```

```

var notList
//make an object of the documents doctype section.
DocType=MyDoc.doctype
//make a nodelist of the notation declarations
notList=docType.notations
//print out the node name and the public id
document.write(notList.item(0).nodeName)
document.write(notList.item(0).publicId)

```

警告: docType 属性没有使用 camelback 标识。你使用 camelback 是为了避免 MS DOM 引擎产生错误。

15.17.5 Entity 和 EntityReference 接口

内部实体声明和实体引用可以由 level 1 的 DOM 访问。外部实体声明不能直接由目前的 DOM 访问。有些解析器直接扩展了实体引用,但是不让它们进入解析树。在这种情况下,引用显然不能被 DOM 访问。

使用 DocumentType 接口的 entities 属性可以得到含有所有实体的一个 NamedNodeMap。如下:

```
NamedNodeMap = [document object].doctype.entities
```

使用 nodeName 属性可以获得实体或实体引用的名字:

```
[Entity or EntityReference name] = [Entity or EntityReference object].nodeName
```

然而这两个属性的替换值被认为是实体的子节点。因此,为了取得替换值要使用下面的语法:

```
[Substitution string] = [Entity or EntityReference object].firstChild.nodeValue
```

下面的 CharacterData 接口的 data 属性也可以使用:

```
[Substitution string] = [Entity or EntityReference object].firstChild.data
```

下面的例子可以说明这些语法。在这个例子中,有三个实体和两个实体引用。其中一个实体是一个外部文件。这个例子建立一个实体的 NamedNodeMap 并用一个循环来显示实体的名字和替代值。然后它遍历 XML 文件树并显示 XML 文件本身的实体引用。

清单 15.14 实体和实体声明

```

<html>
<head>
<title>Entities and entity declarations</title>
<style>body {font-size: 12pt;}</style>
</head>
<body bgcolor="white">
<xml id="isle">
<! DOCTYPE greeting {
<! ELEMENT greeting (#PCDATA)>

```

```

<! ATTLIST greeting
  type CATA #IMPLIED
  manner CDATA #IMPLIED
>
<! ENTITY dummy "not used">
<! ENTITY ent "Hello Entity Declaration">
<! ENTITY doc SYSTEM "someurl.xml">
]>
<greeting type="official" manner="cordial">Hello DOM &ent;&doc;</greeting>
</xml>
<script>
  myDoc=isle
  if(myDoc.parseError != 0)
  {
    alert(myDoc.parseError.reason)
  }
  else
  {
    var rootEl=myDoc.documentElement
    var docType=myDoc.firstChild
    //note camelback notation required
    //for doctype because of MS bug
    var entList=docType.entities
    document.write("<pre>")
    for (var i=0;i<entList.length;i++)
    {
      document.write("the value of the entity named <b>"
+entList.item(i).nodeName + "</b>is:<br \>")
      document.write("&quot;" +entList.item(i).firstChild.nodeValue
+ "&quot;<br \>")
    }

    document.write(myDoc.documentElement.firstChild.nodeValue)
    document.write(myDoc.documentElement.firstChild.nextSibling.
                    firstChild.nodeValue)
    document.write(myDoc.documentElement.lastChild.firstChild.nodeValue)
    document.write("</pre>")
  }
</script>
</body>
</html>

```

在上一个例子中,最后引用了 someurl.xml。

下面是 someurl.xml 的内容:

```

Long live the King!
Let it be known to all those present that from
henceforth entities referred to in a SYSTEM identifier
must be physically present or an error will be thrown.
Long Live Elvis!

```

如果愿意,你可以将一些 XML 标记放在这个文件中并看一看会发生什么。如果它是结构良好的,那么它就会被显示出来。

在清单 15.14 中,建立了一个实体的 NamedNodeMap:

```
var entList = docType.entities
```

然后,用一个循环来显示实体的名字:

```
entList.item(i).nodeName
```

以及它们的替代值:

```
entList.item(i).firstChild.nodeValue
```

在从根元素开始的遍历树的过程中,遇到了第一个实体引用(&ent):

```
MyDoc.documentElement.firstChild.nextSibling
```

替代值被包含在这个实体引用的子节点中,可以用如下语句取得:

```
MyDoc.documentElement.firstChild.nextSibling.firstChild.nodeValue
```

15.17.6 ProcessingInstruction 接口

ProcessingInstruction 接口含有两个属性:target 属性和 data 属性。它们都返回相应的 DomStrings。

15.18 实现 DOM

我们已经学习了 DOM 的接口、属性和方法。现在我们就来学习怎么让它们工作。为了实现它们,你显然需要一个支持 DOM 的应用程序。本书前一节的例子已经使用了一个实现:IE5 DSO,它依赖于 Microsoft 的 DOM 引擎。

15.18.1 Microsoft 的 DOM 引擎

Microsoft 的实现是相当准确的一个。它只有一个明显的错误。对 DocumentType 接口的 doctype 属性它使用 camelback 标识——而不是小写。DocumentType 接口还意味着没有子节点。然而 Microsoft 的引擎允许你使用 childNodes()方法来在 DocumentType 中建立实体和注释的 nodeList。实际上,这复制了由 documenttype 接口的合法属性所提供的 NamedNodeMap。

Microsoft 的 DOM 引擎还出现在最新的 MSXML.dll 中,而且成为了 ActiveX 对象。它们都包含 MS 验证解析器。想了解如何获得它们以及它们使用的条件和费用,请参阅第 14 章或浏览 Microsoft 主页。浏览 <http://www.msdn.microsoft.com/default.asp> 以取得当前的位置。

15.18.2 IBM 的 DOM 引擎

IBM 的 Java 和 C 解析器都完全支持 DOM。它们完全遵守 level 1 DOM。

其他 DOM 实现,无论是 Java 的还是 C 的,每天都在不断地发布。找到适合你需求的实现的最好办法就是在 Robin Cover's XML 页面中冲浪,地址是 <http://www.oasis.open>。

org/cover/, 或者从 W3C DOM 的主页开始。

15.19 为应用程序加入 DOM 支持

我希望你会对在你的应用程序中使用 DOM 感到兴奋。然而,这个问题的焦点是你如何在你的应用程序中与 DOM 引擎协作。

15.19.1 Java

Java 程序员最早开始这方面的尝试。只要将 Java DOM 引擎注册为一个类然后就可以使用它所有的特性和方法了。

15.19.2 C++

IBM 解析器的 C 源代码可以使用,该引擎的类可以直接调用。IBM 引擎也可以提供 DLL 接口。Microsoft 的 DOM 引擎只能通过 DLL 或者是 COM 对象来使用。和 C++ 代码协作最简单的办法可能就是通过 COM 使用 Microsoft 的 ADO。

15.19.3 Visual Basic

Microsoft 的 DOM 引擎可以在 VB 应用程序中作为对象被实例化。可以使用下面的代码:

```
Set xmlobject = CreateObject("Microsoft.XMLDOM")
```

一旦创建了这个对象,就可以以一般的方式来使用它:

```
xmlobject.load(file path)
```

而且可以通过所有 DOM 接口来访问文件。

15.19.4 ASP

在 ASP 页面中创建 Microsoft 的 ADP 实例的最典型语法如下:

```
Dim xmlobject
```

```
Set xmlobject=server.createObject("MICROSOFT.XMLDOM")
```

这使你可以使用强大的 DOM 来在服务器上操纵你的 XML 文件。

15.20 脚本和 DOM

一旦注册了 MSXML ActiveX 对象,任何脚本语言都可以访问这些接口的属性和方法。

15.20.1 使用 C++ DSO

在接口这一节的所有例子中都使用了 C++ DSO。关于使用 DSO 的更详细信息请参阅第 21 章。

15.20.2 使用 Microsoft XMLDOM ActiveX 数据对象(ADO)

Microsoft XMLDOM ActiveX 数据对象可以加入到 JavaScript 页面和 VBScript 页面中。在客户端,只有 IE5 才支持它。在服务器端,在任何支持 Active Server Pages 2.0 以上的主机上都可以使用它。

JavaScript 语法

使用下面的语法来创建一个解析器对象和 DOM 引擎的实例:

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

用下面的语法来加载 XML 文件:

```
oXDOM.load(XML 文件的路径)
```

用下面的语法来加载 XML 字符串:

```
oXDOM.load(XML 文件的字符串变量)
```

请记住,XML 文件会在加载时被解析。如果遇到了任何错误,加载就会被终止。缺省的,在 XML 文件加载的时候会验证并解析所有外部实体。如果出于某种原因,你不希望这种功能可用,你可以用下面的语法来关闭它:

```
oXDOM.validateOnParse = false
oXDOM.resolveExternals = false
```

VBScript 语法

VBScript 的语法和 JavaScript 的略有不同。在 VBScript 中,使用下面的语法来创建一个 ADO 的实例:

```
Dim oXDOM
oXDOM = new CreateObject("Microsoft.XMLDOM")
```

下面是如何在服务器端的 ASP 页面中创建 ADO 的实例:

```
Dim oXDOM
oXDOM = new server.CreateObject("Microsoft.XMLDOM")
```

在 VBScript 中加载文件和在 JavaScript 中加载一样。用下面的语法来加载 XML 文件:

```
oXDOM.load(XML 文件的路径)
```

用下面的语法来加载 XML 字符串:

```
oXDOM.load(XML 文件的字符串变量)
```

同样的代码也可以用在需要关闭验证和外部文件解析的情况下:

```
oXDOM.validateOnParse = false
oXDOM.resolveExternals = false
```

15.21 Gecko 中的 DOM 支持

目前,在很多开发者看来 Gecko 还处于开发阶段。然而,在我看来,它具有非常不错的 DOM 本地支持。虽然如此,我还没有彻底地测试每一个接口、属性以及方法的功能。

15.22 永久保存文件

编辑 XML 文件的一种方法是将 XML 文件作为平面文件。如果应用程序仅仅是对特定元素类型做很少的修改,那这可能是最好的一种方法。然而,如果你要编辑不同元素类型而且还要对文本和文件树进行大量插入操作——而且你还要保证有效性——绝对应当使用 DOM 和编辑 DOM 树。

如果你在使用 DOM 来编辑 XML 文件,你需要一种方法来在编辑结束的时候将这个新树写回到平面 XML 文件中。下面的列表是为你提供了方便。它是从一个 Visual Basic 6 应用程序中摘出来的,不过这段代码也可以转换成 Java 或者是 C 代码。

警告: Visual Basic 编译器改变了许多属性的大小写,对于 VB 来说这无关紧要,但是如果你将这段代码转换成 Java 或者 JavaScript,那么就会产生错误。

清单 15.15 递归地将 DOM 对象编译成平面的 XML 文件的一个函数

```
Function compile_element(element As Object)
' compiles all DOM objects into a flat XML file
' this is a recursive function
' xmlstr is a global variable which needs to be set
' to empty before the first call of this function

' variables with local scope
Dim i As Integer
Dim j As Integer
Dim tempList As Object
If element.hasChildNodes = True Then
    Set tempList = element.childNodes
    For i = 0 To tempList.length - 1

        If tempList.Item(i).nodeType = 1 Then
            ' open the tag
            xmlstr = xmlstr & "<" & tempList.Item(i).nodeName
            ' write out the attributes for each element
            If tempList.Item(i).Attributes.length > 0 Then
                For j = 0 To tempList.Item(i).Attributes.length - 1
                    ' MsgBox tempList.Item(i).Attributes.Item(j).nodeName
                    xmlstr = xmlstr & sp & tempList.Item(i).Attributes.Item(j).nodeName
                    & "=" & chr(34)
                    & tempList.Item(i).Attributes.Item(j).nodeValue & Chr(34) & sp
                Next
            End If
            ' close the tag
            xmlstr = xmlstr & ">" & nl
        End If
    Next
End If
```

```

' call our self if an element
    dummy = compile_element(tempList.Item(i))
' closing tag after all the recursive elements have been called.
    Xmlstr = xmlstr & "</" & tempList.Item(i).nodeName & ">" & nl
' text nodes
    Elself tempList.Item(i).nodeType = 3 Then
        Xmlstr = xmlstr & tempList.Item(i).nodeValue & nl
' processing instructions
    Elself tempList.Item(i).nodeType = 7 Then
        Xmlstr = xmlstr & "<?" & tempList.Item(i).nodeName
        & sp & tempList.Item(i).nodeValue & "? >" & nl
' comments
    Elself tempList.Item(i).nodetype = 8 Then
        Xmlstr = xmlstr & "<!--" & tempList.Item(i).nodeValue & "-->" & nl
' document types
    Elself tempList.Item(i).nodeType = 10 Then
        Xmlstr = xmlstr & "<! DOCTYPE" & sp & tempList.Item(i).nodeName
        & " SYSTEM 'patient-record1.dtd' >" & nl
    'Else; MsgBox tempList.Item(i).nodeType
    'xmlstr = xmlstr & "KKKKK"
End If
Next
End If
' xmlstr will contain the complete flat XML file after allt he calls.
' compile_element = xmlstr
End Function

```

这个函数没有处理实体、注释以及CDATA段。然而，如果愿意的话，你可以很容易地加入它们。

15.23 总 结

本章仔细讨论了 W3C XML DOM。它讨论了一个好的统一的 DOM 的必要性。在讨论了不同的 DOM 模型类型之后，我们学习了所有的 W3C DOM 接口。最后，本章讨论了一些 DOM 的实现。我们还学习了在应用程序中它们是如何协作的。

DOM Level 1 是允许开发者实际操纵 XML 文件的一个伟大开始。尽管提出这个建议才不过一年，很多主要的开发商都已经发布了软件产品。

目前，Level 2 建议就要公开了。它将允许对全部 DTD、样式表以及名字空间进行访问。这的确有点像蛋糕上的糖霜，但是在出现支持它的工具的时候，它还会允许更好地操纵 XML 文件。

然而，即使没有实现 Level 2，而使用 Level 1，开发者也可以使用 XML 作为数据存储方法的所有功能。

第 16 章 在 Java 中使用 SAX API

我们已经仔细讨论了如何使用 DOM。作为一种数据结构 API,DOM 对于大多数人来说都是容易掌握的。然而,DOM 数据结构可能不是最适合你的应用程序。而这正是 SAX 的长处——你可以选择那种最适合你的应用程序的数据结构,或者什么结构都不用,直接使用流来处理。另外,你可以访问许多 DOM 隐藏了的 XML 处理。

SAX(Simple API for XML)是一个 XML 解析器的接口,而不是某一种可以用解析器来生成的数据结构的 API。它工作在 DOM 的下面,这也是它可以更灵活、更高效的原因。另外,它还容易掌握。你可以用很少的几个调用就完成大部分工作。当你需要灵活性并且强调性能的时候,你应当认真考虑使用 SAX。

本章,分几个阶段介绍了 SAX。首先是一些背景知识。然后是最主要的调用,用它们就足可以写出有趣的例子了。接下来介绍了其他一些广为使用的调用以及一些高级调用。最后是对未来的展望。

你可以假设本章中所有的代码段都加载了所有的 SAX 核心接口:

```
import org.xml.sax.*;
```

请注意,尽管在包括 C 和 Python 在内的很多语言中都有 SAX 版本存在,我们却没有在本章中涉及它们,因为绝大多数的 SAX 编程都使用的是 Java。本章还给出了将 Java 代码转化为其他语言的高级语言模块,不过并没有给出实际代码。

16.1 SAX 1.0 的结构

在第 13 章“用 Java 解析 XML”中,我们看到了如何用标准的 ParserFactory 方法来取得 SAX 解析器对象。这些解析器对象为应用程序中的对象提供了一系列方法调用。你的处理器必须支持回调(或回调事件)。在本章中,我们将学习如何配置和使用这些对象。简要说来,我们在单个线程中按照如下顺序使用它:

1. 设置事件处理器。
2. 命令它解析你的 XML 内容。
3. 响应解析过程中的所有回调。
4. 重复 1-3 步直到结束。

本章中我们将讨论这些处理器。图 16.1 说明了解析器以及事件处理器的数据流程。

所有的处理器都是缺省的,因此你仅仅为你的应用程序需要处理的那些事件提供事件处理器就可以了。

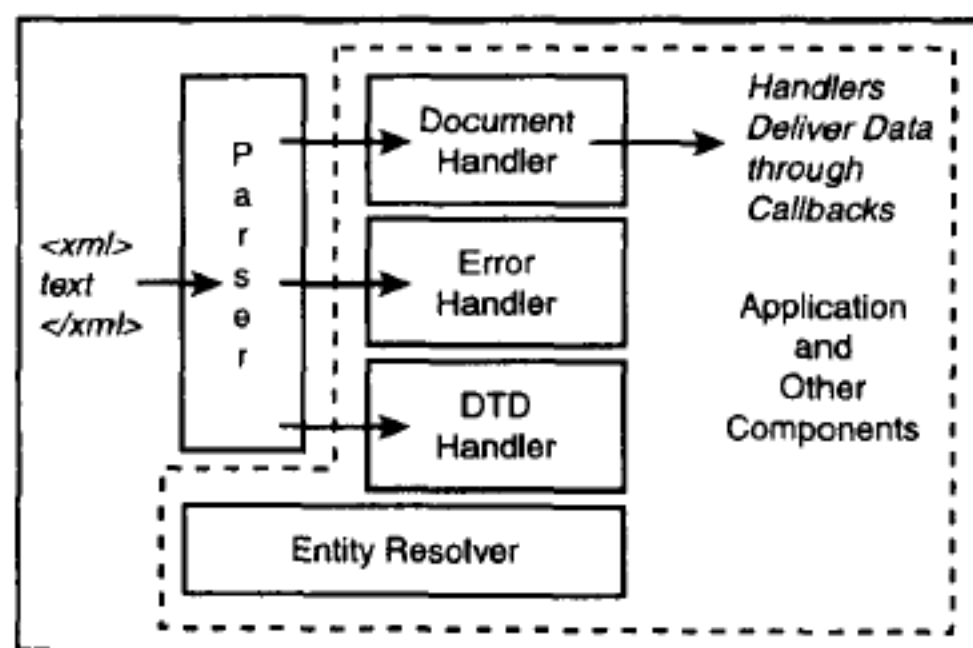


图 16.1 解析器的 XML 数据流程,处理封装在事件中,应用程序可以修改这些事件处理器

16.2 SAX 1.0 包基础

SAX 1.0 包只有不到 12 个接口和类。它们都不难掌握。除了有几个高级的 API 不是经常使用,大多数接口都经常会被用到。本节主要介绍一个基本的 API 子集,使用它们就可以完成 SAX 的大部分工作了。这个子集只有六个调用,但足以让你编写有用的 SAX 应用程序。你可以使用 `parser` 而且可以在你的 `HandlerBase` 的子类中添加一些回调处理器。

16.2.1 解析 XML 文本

有两种方法可以用来让解析器去解析文件。首先我们看一看第一种办法,也是使用最广泛的一种,也就是将 URI 传给解析器,我们已经在第 13 章中简要地介绍过。然后我们会在“从 `InputSource` 中解析”看到另一种办法(对于解析已经保存在数据缓冲区或流中的数据特别有用),以及错误处理的信息。

从 URI 中解析

解析 XML 文件的最简单办法就是直接用它的 URI(一般就是 URL)调用 `Parser.parse()` 方法。解析器打开这个 URI,取得它的内容并且调用相应的事件处理器。下面是你最可能使用的一些调用:

```

String      documentURI = ...;
Parser      parser = ...;
try{
    // First assign one or more handlers, though!
    parser.parse(documentURI);
} catch (Exception e) {
    e.printStackTrace();
}
  
```

将文件名变成 URI

不幸的是, SAX API 不直接涉及一个常见的问题, 这就是可能需要解析一个本地文件系统中的 XML 文件。解决这个问题最好的办法就是将文件名转化成 URL 并用生成的 URI 来调用。这可以确保相应的 URI 可以被解释, 诊断行数是正确的, 等等。

如何将一个文件变成 URL 呢? 最简单的办法就是使用 `java.io.File` 类中的 JDK 1.2 方法:

```
String documentURI = new File(pathname).toURL().toString();
```

这个方法在平台相关的情况下仍能正确工作。比如, 许多人记不住 `file:/d:/xml/xhtml.dtd` 是 Microsoft 操作系统中正确的语法。如果你使用了相对于 Java 虚拟机的当前目录的路径名, 这个方法还负责取得文件的全路径名。

如果你使用的是 JDK 1.1, 你不得不在你的代码中做下面的一些事情:

```
File    file = ...;
String  temp;
String  documentURI;

temp = file.getAbsolutePath();
if (File.separatorChar != '/')
    temp = temp.replace(File.separatorChar, '/');
if (!temp.startsWith("/"))
    temp = "/" + temp;
if (!temp.endsWith("/") && file.isDirectory())
    temp = temp + "/";
documentURI = "file:" + temp;
```

16.2.2 HandlerBase: 统一的 SAX 处理函数

在我们讨论处理函数和它们的方法之前, 我们需要了解类 `HandlerBase`, 这样可以节约我们的时间。这个类提供了所有 SAX 解析器所要提供的默认处理, 也就实现了每一个回调的处理方法。

这为什么是个捷径呢? 因为你可以对它派生它的子类, 增加你关心的那些方法, 并将你的子类作为事件处理函数——而无论这个处理函数是什么! 你无须关心存根方法的提供, 也无须指出哪些接口定义了哪些回调。仅仅是派生一个子类, 定义一些方法, 就行了。

16.2.3 SAX DocumentHandler 接口

对于大多数 SAX 应用程序来说, `DocumentHandler` 接口是最重要的接口。在这个接口中定义了那些向你的应用程序输出 XML 文本和标记的回调。

在大多数应用程序中, 你需要注册你的应用程序的文件处理函数。你可以在任何时候, 甚至是在解析的过程中, 注册它。注册会立即生效:

```
parser.setDocumentHandler(docHandler);
```

编写一个 SAX 应用程序来操纵 XML 数据而不使用定义在这个类中的回调函数, 是非常诱人的工作。你将会学习到关于客户的 `DocumentHandler` 的大多数解析器, 仅仅是确认解

析器提供了你需要看到的回调。

Element 回调

在 SAX 中两个 element 回调是基本的回调。它们告诉应用程序核心标记已经应用于文件。

第一个回调告诉你解析器已经解析了一个元素的开始标记。你可以得到元素的名字及其所有属性(有很多 XML 处理都会接近这些属性,包括值的缺省和标准化)。然后,在任何文本或者子元素等其他回调之后,会得到第二个回调,它告诉你解析器处理了这个元素的结束标记。

大多数应用程序以一种相同的方式使用这些信息。你的应用程序可以将那些会对后续数据的处理产生影响的元素结构的信息保存在栈里。应用程序可以用这两个回调来控制如何建立一个数据结构或者如何像流一样处理数据。

元素的 `AttributeList` 是 SAX 提供的最复杂的数据结构,尽管它也非常简单。清单 16.1 显示了如何在 `startElement` 回调函数中访问它,并且打印出与这两个回调提供的数据相对应的 XML 开始标记和结束标记。

清单 16.1 打印出它们的标记的元素回调

```
public void startElement (String name,AttributeList attrs)
throws SAXException
{
    int    attLen = 0;
    System.out.print("<" + name);
    if (attrs != null && ((attLen = attrs.getLength()) > 0)) {
        for (int i = 0; i < attLen; i++) {
            // NOTE: the value could have single quotes; it could
            // need to be escaped using "&apos;"; (NOT SHOWN)
            System.out.print(" " + attrs.getName(i) + "=");
            System.out.print(attrs.getValue(i) + "'");
        }
    }
    System.out.print(">");
}

public void endElement (String name)
throws SAXException
{
    System.out.print("</" + name + ">");
}
```

Text 回调

在 XML 中,所有没有标记的都是内容(除非它在文件根元素的外面)。你的应用程序可以访问所有这些内容。和元素开始标记和结束标记一样, `text` 回调可能是需要了解的最重要的回调。

有两种类型的内容:可以忽略的空格,它是在那些仅仅满足模型的元素中的空格;以及其他的所有东西。SAX 有两个 `text` 回调,分别对应于这两种内容。尽管只有验证解析器才需

要提示出现了可忽略的空格,但是大多数解析器都这样做了,因为这对于立即忽略这样的数据非常有用。

这两个回调都提供了字符缓冲区,如果你的应用程序需要,可以通过它来拷贝那些必要的的数据(你不应该对缓冲区进行任何修改,如果你修改了,那么你可能会终止解析!)

通常,你会编写与清单 16.2 类似的代码并忽略可忽略的空格而缓冲其他字符。在很多情况下,你可能不会立即处理其他字符,因为不能保证处理函数不会在多个块中用到同一块文本。实际上,大多数 SAX 处理函数都是以效率最高的方式来报告这样的字符,并且假设应用程序可以缓冲它们,直到遇到某个元素的边界为止。

清单 16.2 将内容用一个 StringBuffer 缓冲的 Text 回调函数

```
private Boolean          saveIgnorable = false;
private StringBuffer     charBuf = new StringBuffer ();

public void ignorableWhitespace (char buf [],int offset,int length)
throws SAXException
{
    if (saveIgnorable)
        characters (buf,offset,length);
}

public void characters (char buf [],int offset,int length)
throws SAXException
{
    charBuf.append (buf,offset,length);
}

// this might be called by startElement(),endElement(),
// and other callbacks. It is NOT a SAX callback.
private void doneText ()
{
    // what does YOUR application want to do with its text?
    Text Available (charBuf.toString ());
    Char Buf = new StringBuffer();
}
```

将内容转化成 String 对象并不是 Java 处理这类数据最有效的方法,不过这样非常方便。使用字符数组更为高效,而且对于操纵许多内容的应用程序来说有明显的差别。

处理指令

在 XML 中,处理指令(PI)被用来为应用程序提供那些不属于正常的文件结构的指令。有些人甚至认为 PI 是 XML 工作组所宣布的不必要的垃圾,但也有人非常喜欢它。

一个有趣的意见是 PI 可以出现在一个 XML 文件的任何地方而不必与 DTD 内容模型一致。这使 PI 可以用来表达一些不能用元素结构来表达的信息。

一个简单的回调会报告 PI,它报告了指令对象的名字以及这个指令的数据。XML 处理函数不解释这些数据。有些应用程序采用一种使 PI 看上去就像元素属性的策略。也有人嵌入一些程序语言或脚本语言代码。

在清单 16.3 中,你会看到你可以使用任何你喜欢的处理指令名,并用它来影响你自己

的状态或者调用某个子系统来进行那种极端复杂的处理。请注意如果遵守 XML 名字空间建议,你应当避免处理那些含有冒号的处理指令名。这个规范不允许像 prefix:local 这样的名字(尽管 XML1.0 允许)。

清单 16.3 processingInstruction 回调

```
public void processingInstruction (String target,String data)
throws SAXException
{
    if ("halt-and-catch-fire".equals (target)) {
        System.err.println( "HALT - engine room is on fire!");
        System.err.println ("more data: " + data);
        System.exit(1);
    } else if ("save-ignorable".equals (target)) {
        saveIgnorable = "true".equalsIgnoreCase(data);
    } else if ("tcl".equals (target)) {
        // deliver to the TCL interpreter; may change GUI
        tcl.eval (data);
    } else if ("mode".equals (target)) {
        if (data == null || "".equals (data))
            toolMode = null;
        else
            toolMode = data.toLowerCase (). Trim ();
    }
}
```

XML 规范有一个注释会影响可能用于 PI 对象“格式声明”的注释声明。其目的尚不明朗。尽管使用注释的 SYSTEM 标记作为要执行的命令的名字是极端危险的,有些系统仍然这样做了(而且忘记了相应的安全问题)。在本章后面的“对不解析实体和注释使用 DTD 处理函数”一节中,将讨论一种更安全的建议。

16.3 SAX 应用程序的例子

这可能难以置信,但是你刚才学习的 SAX 知识已经足以创建和理解简单的 SAX 应用程序。

16.3.1 简单的模板处理

许多 XML 应用程序都要生成 HTML(或者 XHTML)内容。有很多方法都可以生成这样的输出,而限于用它的样式表来使用强大的 XSL 转换功能,在第 10 章“理解扩展样式语言”和第 11 章“创建 XSL 样式表”中已介绍过它。这个例子是一个非常简单的工具,模板仅仅执行一些文本替换。

清单 16.4 中显示的文件处理函数可以被传给一个 SAX 解析器。当用这个解析器来处理 XML 文件的时候,文件中的所有元素和内容都用 HTML 语法复制到输出。像<?substitutue NAME?>这样的 PI 被输入字典中与键 NAME 相关的内容替代,这个输入字典作

为一个 XML 内部实体应当是结构良好的。

清单 16.4 SAX DocumentHandler 复制输入,进行一些替代和 HTML 化的修改

```
public class TemplateHandler extends HandlerBase
{
    private Dictionary    dict;
    private Writer        out;

    // constructor ... echo relevant input to 'w', substituting per 'd'
    public TemplateHandler (Dictionary d, Writer w)
    { dict = d; out = w; }

    // pass text through unaltered (except drop any ignorable whitespace)
    public void characters (char buf [], int offset, int length)
    throws SAXException
    {
        try { out.write (buf, offset, length); }
        catch (IOException e) { throw new SAXException ("write text", e); }
    }

    // optionally substitute fields
    public void processingInstruction (String target, String data)
    throws SAXException
    {
        if (!"substitute".equals (target))
            return;
        try {
            String value = (String) dict.get (target);
            if (value == null)
                value = "<b>UNKNOWN SUBSTITUTION VALUE</b>";
            out.write (value);
        } catch (IOException e) {
            throw new SAXException ("substitution", e);
        }
    }

    // write HTML end tag if needed
    public void endElement (String name) throws IOException
    {
        // Many HTML parsers get unhappy if they
        // see end tags for these elements
        if ("img".equals (name)
            || "hr".equals (name) || "br".equals (name)
            || "meta".equals (name) || "link".equals (name))
            return;
        try { out.write ("</" + name + ">"); }
        catch (IOException e) {
            throw new SAXException ("write end tag", e);
        }
    }

    // write start tag and attributes
    public void startElement (String name, AttributeList attrs)
    throws IOException
    {
```

```

        try {
            int attLen;
            out.write("<" + name);
            if (attrs != null && ((attLen = attrs.GetLength()) > 0)) {
                for (int i = 0; i < attLen; i++) {
                    // NOTE: value should be escaped using "&apos;"
                    out.write(" " + attrs.getName(i) + "=" + attrs.getValue(i));
                }
            }
            out.write(">");
        } catch (IOException e) {
            throw new SAXException("write start tag", e);
        }
    }

    // runs template processor on argv [0]
    // output is HTML-ish, and in ISO-8859-1 encoding
    // current directory must have a "subst.properties"
    // file with substitutions
    public static void main (String argv [])
    {
        try {
            Parser      p = ParserFactory.makeParser ();
            Properties  props = new Properties ();
            Writer      out = new OutputStreamWriter (out, "8859-1");

            Props.load (new FileInputStream ("subst.properties"));
            p.setDocumentHandler (new TemplateHandler (Props, out));
            p.parse (argv[0]);
            out.close();

        } catch (Exception e) {
            e.printStackTrace (System.err);
        }
    }
}

```

请注意这段代码流化了数据,因此,你不必在内存中建立一个备份。在内存紧张的时候,这样做效率更高。

另外,因为它的输入是结构良好的 XML,它的输出也可以在几乎所有的 HTML 引擎上显示(请注意处理像
这样的空标记的那行代码。这是使 XML 要像 HTML 那样工作所要作出的主要让步。XHTML 采用了一种稍有不同的解决方法,它在一般的 XML 结束标记中加入一个额外的空格。
在大多数早期的 Web 浏览器上都能工作而且完全符合 XML 语法)。这是在类似应用中使用 XML 得到的一个好处。许多其他的工具会生成格式不良的 HTML,这些 HTML 经常被 Web 浏览器拒绝显示,而且对于不同的浏览器可能产生不同的效果。

16.3.2 构建 DOM 树

理解 SAX 和 DOM 之间的关系的一个办法就是看一看 SAX 的 DocumentHandler 回调

是如何被用来产生空文件的。这并不能说明所有那些组成 DOM 树的包是如何工作的,因为 DOM 没有定义一种可移植的方法来实现一些 DOM 规范所描述的行为。(比如,你不能组装一个 DOM DocumentType,也不能将节点置为只读)。

在阅读清单 16.5 时,请思考你应该如何使用 SAX 回调来组装你现有代码的数据结构,而不是先构建 DOM 树,翻译,然后删除 DOM 树。

清单 16.5 只使用基本 SAX 回调组装 DOM 文件的文件处理函数

```
import java.util. Stack;
import org. w3. dom. * ;

public class DomDocHandler extends HandlerBase
{
    private      Document doc;
    private      Stack stack = new Stack ();
    // this handler populates the single Document it's passed;
    // it can't be used twice!
    Public DomDocHandler (Document d)
    {
        doc = d;
        stack. push (doc);
    }
    public void processingInstruaction (String target,String data)
    throws SAXException
    {
        try {
            ProcessingInstruction    pi;
            Node                      top = (Node)stack. peek ();
            pi = doc. createProcessingInstruction (target,data);
            top. appendChild (pi);
        } catch (Exception e) { throw new SAXException (e); }
    }
    public void characters (char buf [],int offset,int length)
    throws SAXException
    {
        try {
            Text                      text;
            Node                      top = (Node)stack. peek ();
            Text = doc. createTextNode (new String (buf,offset,length);
            Top. appendChild (text);
        } catch (Exception e) { throw new SAXException (e); }
    }
    public void startElement (String name,AttributeList attrs)
    throws SAXException
    {
        try {
            Elemenet                  element;
            Node                      top = (Node)stack. peek ();
            element = doc. createElement (Name);
            top. appendChild (element);
            stack. push (element);
            //NOT SHOWN: loop calling element. setAttribute()
```

```
        // for each attribute provided by SAX
    } catch (Exception e) { throw new SAXException (e); }
}

public void endElement (String name)
{
    stack.pop();
}
}
```

在你掌握了 SAX 和 DOM 的其他部分之后,请回头看一看这段代码。SAX 提供了哪些被 DOM 所隐藏了的信息呢?有哪些类型的信息 DOM 数据结构允许你保存,而 SAX 却没有提供呢?

SAX 和 XML 流水线处理

连接 DOM 和 SAX 的另外一种办法是建立一个 DOM 树然后遍历它...,产生 SAX 事件,就像 XML 文件被解析一样。事实上,有几个 SAX 解析器就是这样做的。(在 Internet 上找一找它们!)

通过将使用 SAX 的组件和那些使用 DOM 的组件或者其他表示形式建立多种多样的连接,这两种技术允许你组成处理流水线。

一些使用 SAX 解析器的组件将它们的输出打包,使输出看上去好像是 SAX 解析器对象产生的,这样,这种流水线处理就更容易了。

16.4 SAX 的其他核心功能

尽管我们已经看到的 SAX 已经足以胜任很多工作,然而应用程序还需要更多的一些特征。

16.4.1 使用 Parser.parse()的其他方法

健壮的应用程序需要用比我们已经在前面的基本程序接口所展示的更灵活的方法来向解析器提供数据,以及进行某些特殊的错误处理。SAX 提供了这种灵活性。

从 InputSource 解析

我们已经看到了,第一种,也是最常用的调用解析器的方法——将 URI 传给它。告诉 SAX 解析器解析文件的第二种方法是用一个 InputSource 对象来描述这个文件 (InputSource 对象会在本章的后面仔细讨论)。我们创建并设置 InputSource 对象,然后用这个对象来调用解析器。

```
InputSource source = new InputSource(byteStream);
parser.parse(source);
```

使用 InputSource 对象作为解析器输入的主要原因是有时我们需要解析没有全局访问 URI 的对象。这样的情况包括:

- 你得到的可能是 applet 或者其他客户端组件响应 POST 而输出的 XML 文本。

- 你得到的可能是 Web 应用服务器内部的 servlet 的 POST 输入的 XML 文本。
- 你得到的可能是邮件中 MIME 附件的 XML 文件。
- 你得到的可能是从字符串或者消息缓冲区中的 XML 文本,并且要解析它。

如果你的确有文件的 URI,应当避免用 `InputSource` 对象向解析器描述 `InputSource` 对象。传递 URI 是一种确认你没有忽略某些细节的简单办法,而且解析器也经常需要 URI 来处理它。

带异常处理的解析

如果你加入使用 SAX 所需要的异常处理,它会看上去像下面的代码:

```
String      documentURI = ...;
Parser      parser = ...;          //Assign handlers!

try{
    parser.parse(documentURI);
    //similarly, parsing with an input source
} catch(SAXParseException e){
    System.err.println(" * * Parse Error: " + e.getMessage());
    System.err.println("   Line: " + e.getLineNumber());
    System.err.println("   URI: " + e.getSystemId());
    ((e.getException() == null)
     ? e
     : e.getException()).printStackTrace();
} catch(SAXException e){
    System.err.println(" * * SAX Error: " + e.getMessage());
    ((e.getException() == null)
     ? e
     : e.getException()).printStackTrace();
} catch(IOException e){
    System.err.println(" * * I/O Error: " + e.getMessage());
    System.err.println("   Class: " + e.getClass().getName());
} catch(RuntimeException e){
    System.err.println(" * * Runtime Error: " + e.getMessage());
    System.err.println("   Class: " + e.getClass().getName());
}
```

这些 catch 子句显示了你可能遇到的主要错误报告,包括解析文件的那些方法和前面提到的 `InputSource` 方法:

- `SAXParseException` 对象可能包括位置信息。通常这包括发现错误的 URI (SYSTEM 标记,有时是文件)和行号。它可能还包括列号和一个 PUBLIC 标记。这些异常通常是解析器产生的,但是一些应用程序也会产生它们以保证更好地诊断应用程序级的 XML 数据绑定错误(关于解析器产生的异常的更多信息,请参阅本章后面的 `ErrorHandler` 的讨论)。
- 通常,其他类型的 `SAXException` 对象可以在应用程序的任何事件回调中产生。但是,它们也可以由解析器自己产生。
- 如果需要接收一个异常信号,任何 `SAXException` 都可能会隐藏另一个异常。这些被

隐藏的异常可能描述了产生这个 SAXException 的问题。

- 作为一个规则,IOException 对象是在 SAX 解析器读入 XML 数据时产生的。比如,文件没有找到,字符解码错误,网络连接中断。
- RuntimeException 对象实际上应当是不可见的,但是有的时候可以看到它们。一般的,它们指出了应用程序或者解析器的内部错误。然而,如果你的应用程序向解析器提供了错误的数据,那么结果有时就会是一个 NullPointerException。

你应当小心谨慎地为你的用户(或者是管理员)对你的应用程序提供有效的诊断。这些诊断不但要描述什么出了错,而且要根据应用程序的行为提出解决问题的建议。

线程化和 SAX 解析器

同一时刻只能有一个线程使用 SAX 解析器。如果你并发地使用多个线程来解析文件,那么每一个线程都必须取得并使用它自己的解析器对象。

另外,给定的解析器在完成第一个文件的解析之前,不能开始对第二个文件的解析。如果你想要让一个线程并发地解析两个文件,则必须取得并使用两个不同的 SAX 解析器。

16.4.2 ErrorHandler 对象

当 SAX 解析器需要向调用者报告一个解析错误的时候,它会调用它的 ErrorHandler。这使应用程序可以控制如何报告错误(或警告),以及是否报告它们。应用程序可以用图形用户界面和打印两种方式来报告,或者完全忽略这些错误。可能产生一个异常并停止解析,或者继续处理 XML 数据。应用程序可能需要用户干预。

通过 ErrorHandler 回调,解析器可以报告三种不同类型的错误:

- fatalError()回调指出有些东西不能恢复,比如发现文件不是结构良好的。如果应用程序试图继续,则会报告更多的错误,但是不会再报告更多数据(通过 DocumentHandler,DTD Handler,以及类似的处理函数)。只有 ErrorHandler 可以在致命错误之后被调用。
- Error()回调指出可以恢复的错误。XML 规范要求所有有效的错误都是可恢复的,因此,这些错误通常用这个调用来报告。如果你不产生给出的异常,解析器会继续处理应用程序的数据。
- Warning()回调指出可能用来进行错误处理的消息,但是这些消息不是 XML 规范定义的错误。举例说来,你可能有对特定实体或属性的多个定义。这不是一个 XML 错误,但是解析器值得产生这个警告,以防实际运行与预想不符。

解析器为每一个这样的方法提供了一个 SAXParseException 对象。这样的异常可以用来描述解析器消息(实体没有声明)以及由 Locator 对象提供的同样信息,本章的后面还要讨论。

如果你希望在发现错误的时候停止解析,那么就产生一个传给处理方法的异常对象(实际上你可以产生任何 SAXException,但是通常没有理由创建不同的对象)。

缺省的致命错处理函数会和异常一起产生一些参数,这样,所有的 SAX 解析器就可以在遇到第一个结构良好的错误之后停止解析。其他错误回调的处理函数仍然继续处理。

提示:许多程序员首先使用验证 SAX 解析器,他们希望验证错误可以终止解析,而且通常还需要一个诊断提示。毕竟,如果不需要拒绝不合法的文件,为什么还要使用验证解析器呢?

然而,XML 规范要求报告所有的验证错误(在“用户选项”),而 SAX 规范要求用户来控制 SAX 解析器在遇到验证错之类的非致命错误后是否继续解析。通常,这意味着验证错误不会使 SAX 解析器停下来。初看起来这的确引起很多混乱。

这也就是说,除了停止解析和打印消息之外,如果你想要对验证错误进行其他操作,你的程序必须设置 SAX 解析器的 ErrorHandler。

如果你希望 ErrorHandler 打印每一个解析器可能报告的诊断消息,而又想要在致命错之后收集额外的诊断信息,那么可能需要如清单 16.6 所示的代码。它打印除了退栈之外的 SAXParseException 报告的所有信息。

清单 16.6 一个 ErrorHandler,它可以打印到解析结束为止 SAX 解析器产生的所有消息

```
public class MyErrorHandler implements ErrorHandler
{
    private void print (String label, SAXParseException e )
    {
        System.err.println(" * * Parser " + label + " : "
            + e.getMessage ());
        System.err.println (" Line: " + e.getLineNumber ());
        System.err.println (" Column: " + e.getColumnNumber ());
        System.err.println (" URI: " + e.getSystemId ());
        System.err.println (" PUBLIC id: " + e.getPublicId ());
    }

    public void warning (SAXParseException e )
    { print ("Warning", e); }
    public void error (SAXParseException e)
    { print ("Error", e); }
    public void fatalError (SAXParseException e)
    { print ("FATAL ERROR", e); }
}
```

当然,你必须在解析器中注册 ErrorHandler 以便调用它。这可以在任何时候完成。它将立即生效,甚至在解析的过程之中:

```
parser.setErrorHandler(errHandler);
```

目前,大部分 SAX 解析器在遇到致命错的时候立即终止,尽管只要它们所做的仅仅是报告它们遇到的其他错误,XML 规范允许它们继续处理。即使应用程序提供的 ErrorHandler 试图继续,解析器也会停止(正常情况下产生它们提供给 fatalError 回调的异常)。

16.4.3 InputSource 对象

InputSource 对象表示了 SAX 解析器试图读取的数据,在 SAX API 中有两处用到了

它。最常见的用法是作为 `Parser.parse()` 方法的输入。另外一种用法是作为 `EntityResolver.resolveEntity()` 方法的输出,我们将要在“对外部解析实体使用 `EntityResolver`”一节的 SAX 高级特征中对此进行详细介绍。

`InputSource` 对象是由四个部分结合起来的。所有这些部分在不同情况下都是可选的。如果你创建了一个 `InputStream`,可以试着使用这里列出的一种配置,建议使用第一种配置:

- `SYSTEM` 标记以及可选的 `PUBLIC` 标记。`SYSTEM` 标记是完全解析了的 URI,解析器可以直接从中读取数据。XML 没有标准化 `PUBLIC` 标记,除非你用 `EntityResolver` 试图解释它,否则 `PUBLIC` 标记在解析过程中被忽略:

```
InputSource    source;
String         documentURI = . . . ;

source = new InputSource (documentURI);
source.setPublicId (publicID);           //OPTIONAL
```

- `Java.io.InputStream` 以及可选的编码名称。如果你提供了一个输入流,那么只有在你绝对确认它是正确的的情况下,你才要提供编码名称。XML 解析器可以检测到 XML 编码声明,因此,你不必试图猜测编码方式。比如,如果你从 MIME 内容类型的 `charset="..."` 属性中知道了字符编码,你就知道它是正确的(MIME 内容类型总是在任何 MIME 体之前定义。特别的,假设使用了这种编码而不是使用在 XML 文件的 `encoding="..."` 声明中所指出的编码)。另外,你还可以记录你用来存储数据的编码方式。

你使用 `InputStream` 的时候,如果存在 `SYSTEM` 标记,那么就应试着将它作为数据来提供。这允许在解析器或者应用程序在数据中遇到 URI 的时候,可以正确地解释它们。

```
InputSource    source;
InputStream    inputStream = . . . ;
String         documentURI = . . . ;

source = new InputSource (inputStream);
source.setSystemId (documentURI);           //WHEN POSSIBLE
source.setEncoding (encodingName);         //OPTIONAL
```

- `Java.io.Reader`。如果 SAX 解析器不必考虑其内容的编码方式,因为某些其他的系统组件可能已经处理了这个问题,那么可以直接传递 `Reader`。

再说一次,如果有 `SYSTEM` 标记,一般要尽量使用它,这样相对的 URI 才能被翻译:

```
InputSource    source;
Reader         reader = . . . ;
String         documentURI = . . . ;

source = new InputSource (reader);
source.setSystemId (documentURI);           //WHEN POSSIBLE
```

通常,最好应当避免用 `InputStream` 或者 `Reader` 来创建输入流。如果你一定要这样做,那么只要可以提供 `SYSTEM` 标记(URI/URL),就应当提供它。如果你不用 `SYSTEM` 标

记,那么不但相对的 URI 不可能被翻译(这会给你的应用程序带来一些问题),而且你从解析器得到的诊断也不那么有用了。

为什么从它的 I/O 机制中创建 `InputSource` 不好呢?一个原因是这很容易造成字符解码的错误,而正确解码正是 XML 解析器的工作(有空看一看 XML 解析器的字符集自动检测的代码)。另外一个原因是 SAX 规范没有指定谁来关闭数据流。这意味着要自己来关闭数据流(你可以确认已经读到了 EOF),并且如果解析器已经关闭了数据流,还要复制这个数据流(一般要尽快关闭 I/O 资源。I/O 资源会浪费你有限的 JVM 和操作系统资源,而且如果你等到对象成为垃圾时才回收和消灭它们,你很快就会耗尽所有的资源)。

16.4.4 其他 `DocumentHandler` 回调

`DocumentHandler` 对象不仅仅被用来报告文件中发现的元素和内容内容。它还提供两大类信息。

文件回调

SAX 提供了两个回调来记录你的处理的开始和结束。在开始解析文件的时候,调用 `startDocument` 回调,而在解析完成之后,从 `Parser.parse()` 调用返回之前,调用 `endDocument` 回调。

`startDocument` 回调是分配和初始化资源的良好时机。而 `endDocument` 回调则是释放资源的时机。举例说来,维护复杂状态的文件处理函数可以在 `startDocument` 回调中分配状态对象,而在 `endDocument` 中将这些变量置为空值以便回收。有些应用程序在这些回调中不做很多工作,在清单 16.7 中,在这些回调中打印了一些调试信息,就是这样的一个例子。

清单 16.7 打印诊断信息的文件回调

```
public void startDocument ()
throws SAXException
{
    if (debug)
        System.out.println("Starting to Parse ...");
}

public void endDocument()
throws SAXException
{
    if (debug)
        System.out.println("Done parsing!");
}
```

Locator 对象

文件处理函数还有另外一个可选的回调。SAX 解析器不一定要使用它。如果解析器使用了这个回调,那么它总是在 `startDocument` 之前被调用,这样你可以很早就知道你所用的解析器能否提供这样的信息。

这个回调提供了一个 `Locator` 对象,你所做的仅仅是保存它:

```

class MyDocHandler ... {
    private Locator    locator;

    public void setDocumentLocator(Locator l)
        {locator = l;}

    //the rest of this class is not shown
    ...
}

```

Locator 对象仅仅告诉其他回调触发回调的文件的位置。这些信息包括输入的 SYSTEM 标记、行号、列号以及公共的 ID。这些信息仅仅在给定应用程序回调中才是有效的 (SAXParseException 对象也同样提供这些信息, 刚才我们已经在介绍解析的异常处理以及 ErrorHandler 对象时介绍了它)。不可用的信息被报告为空字符串或者负数。

这些信息可以用于实现多种目的。一个目的是使其他回调在它们创建 SAXParseException 对象来向应用程序报告错误的时候使用这些信息 (SAXParseException 构造函数可以将 Locator 作为参数, 使得实现这个目的更容易了)。举例说来, 一个处理日销售记录的处理函数可能需要在交易被重复或者在某个客户当发生某种行为的时候报告一个错误。

这些信息不太明显的一个用处是解析在应用程序数据中发现的相对 URI, 如清单 16.8 所示。作为规律, 这样的 URI 应当根据发现它们的实体来解析。它们可能在基本文件中, 或者在某个外部实体中。通过使用 Locator 对象, 回调可以确保正确地解析这样的相对的 URI。(这是一类不能通过 DOM 访问的信息, 因此如果你需要这样的处理, SAX 是你惟一的选择。)

清单 16.8 startElement 回调片段——使用 Locator 来解析相对的 XHTML URI

```

public void startElement (String name, AttributeList attrs)
    throws SAXException
{
    if ("a".equals (name)) {
        String uri = attrs.getValue ("href");
        if (uri != null) {
            URL    url = new URL (locator.getSystemId ());
            url = new URL (url, uri);
            uri = url.toString();
            // now the URI is fully resolved relative to the
            // correct base URL
        }
        ...
    }
    ...
}

```

16.4.5 不是所有的解析器都一样

在第 13 章中, 我们看到了一个支持 SAX API 的 XML 解析器的列表, 而且学习了它们

之间基本的不同。有些是非常一致的——也有一些有严重的错误。有些比较小,而有些比较快。有了SAX,你可以选择解析器而不必考虑如何用API来访问它……,而且你还可以日后改变你的选择。

API允许有一些不同而且它们可能会影响你的应用程序。对于SAX 2.0 API来说,你可以通过询问解析器回答下面一些问题,但是对于SAX 1.0来说,你必须在解析器的文件中找到它们:

- 解析器是否报告可忽略的空格? 只有验证解析器才必须报告它,但是如果它尽早地丢弃这些空格,你的应用程序写起来可能更容易一些。目前,只有XP解析器不报告可忽略的空格,所有其他的解析器(甚至一些非验证解析器)都报告可忽略的空格。
- 解析器是否读入全部外部实体? 验证解析器一般要读入它们。非验证解析器有另外的选择。它们可以忽略所有的外部实体,或者是选择性地,很可能是只在独立文件的情况下才略过外部参数实体。一般的,如果忽略外部实体,解析器的输出是不同的,因此大多数解析器都读入它们。这样可以确保正确地展开一般的实体,一致地规范属性,以及正确的报告属性值的类型。目前,极少数Java解析器略过外部实体。如果外部实体被略过,那么解析器只能用于声明为独立文件的文件。
- 解析器是否处理你所面对的所有字符编码? 处理函数最起码要能处理UTF-8和UTF-16。许多解析器支持Unicode的严格子集,比如US-ASCII(一种7位子集)和ISO-8859-1(一种8位子集)。在你的应用程序中处理这些问题非常麻烦。可以找一个能够利用JDK对150种编码(以及数百种它们的名字)支持的解析器。
- 解析器是否进行验证? 你可以用验证解析器做任何事,但是速度会慢一些。一定要注意你是如何配置ErrorHandler的,还要确认在ErrorHandler中这个解析器一定要遵守SAX API。
- 解析器是否遵守SAX API? 不幸的是,有些解析器在实现这个API的时候实际上做得不够好。它们没有很好处理的一个方面就是错误处理。

还有很多实际问题你需要考虑,比如许可和发布。总的来说,很多解析器都很好地遵守了SAX API。这使得我们这些软件开发者的工作更为简单,因为我们有很多出色的选择。

16.5 SAX的高级特征

SAX API还有一些特征,可能不像前面的那些特征那样,被你的应用程序经常用到。然而,这些特征也是值得在这里探讨的。

16.5.1 对外部解析实体使用EntityResolver

EntityResolver可能是SAX API最重要的高级功能。XML文件可能由多个实体构成。在这种情况下,除了主文件之外,文件还要和其他组件协作。这通常是通过由DTD引用外部声明来完成的,这样文件可以将它的内容分散在若干个文件之中。就像是用很多章来构成一本书一样。每一章都可以自成一个文件,然后作为外部实体来访问。不同的作者和编辑可以同时每一章上工作。这也是大规模文件的典型结构。

这个结构更突出了访问这些实体的问题。如果保存 DTD 的服务器关机了会发生什么？复制了的 Web 服务器以及使用缓冲的 Web 代理服务器会涉及一些和世界范围访问这些实体的扩展性以及冗余之类的问题。但是它们不能涉及 XML 应用程序面对的其他问题。特别是，它们不支持不太被了解的 URI 类型，PUBLIC 标记的解释以及对断连操作的支持。

当你使用不太通用的 URI(比如 URN)或者 XML 数据，像用 SGML 的 Formal Public Identifiers(FPI)使用了 XML PUBLIC 标记的时候，EntityResolver 实际上是惟一的解决方案。同样，如果你从数据库中生成了数据，EntityResolver 可能被用来将数据库和解析器连接起来(如果数据库不支持的话，甚至将数据翻译为 XML)。

EntityResolver 非常简单。无论何时，只要 SAX 解析器需要解析外部实体，它就会要求解析器来传递数据。解析器取得传给它的数据然后返回 InputSource 或者返回空值。空值表明解析器应当使用它自己的智能，通常这意味着要将 SYSTEM 标记作为 URL 来处理，并要读取内容。

用 EntityResolver 解决的最常见问题可能就是使用本地的 DTD 副本。举例说来，有些应用程序使用“金”副本，不但要确认它们可用而且要加速它们的处理。这些 DTD 用 PUBLIC 标记给出，这样文件才能使用它。Resolver 可能如清单 16.9 所示。

清单 16.9 使用 PUBLIC 标记来触发对本地缓冲数据的访问的 EntityResolver

```
import java.io. * ;
import java.util. Dictionary ;
import org.xml.sax. * ;

public class MyResolver implements EntityResolver
{
    private Dictionary    publidMapping;

    //dictionary maps PUBLIC identifiers to local filenames
    public MyResolver (Dictionary dict)
    { publidMapping = dict; }

    public InputSource resolveEntity (String publid,String sysid)
    throws IOException,SAXException
    {
        String    filename;
        If (publid != null
            && ((filename = (String)publidMapping.get (publid)) != null)) {
            InputSource    retval;
            File            f = new File (filename);

            retval = new InputSource (f.toURL ().toString ());
            retval.setPublicId (publid);
            return retval;
        }
        return null;
    }
}
```

为了注册这样的一个 resolver:

```
parser.setEntityResolver(resolver);
```

请注意这个例子遵守了前面提到的构造 `InputSource` 对象的指导。特别是,它确认解析器所用的系统标记正是实际使用的,而不是在映射之前的那一个。这就确保了任何报告出来的错误都反映了真正的错误源。

提示:一定要真正理解如何在实体声明中用 `EntityResolvers` 和 `PUBLIC` 标记来取得文件的本地副本。XML 文件的作者不能确定它们的外部实体所造成的许多错误(典型的是 DTD 片段),这些文件的接受者都可以使用。`EntityResolvers` 经常解决这类问题的关键。

Catalog

你可能听说过一种叫做 `catalog` 的东西,特别是 `SGML/Open Catalog(SOCAT)`。这是一种提供了从 `SGML FPI` 到文件名映射的语法。它比前一个例子中的查表能力强一些。它支持在网络文件系统支持的 `catalog` 之间进行同样的转发。如果你的项目和 `SGML` 联系密切,那么你取得或者构建一个 `SAX EntityResolver` 的实现来在 XML 系统中实现 `SOCAT` 的那部分功能是值得的。

同样的解析器可以用在类似的 `SYSTEM` 标记的映射上——但是仅仅限于表示 `URN` 的那些(所有的 `URN` 都具有 `urn:` 模式前缀)。不要对 `URL` 做这样的映射(`URL` 具有 `http://`, `ftp://`, `file:`, 或者类似的模式前缀)。相反,可以问一问文件的提供者他们的 `SYSTEM` 标记是否正确。

请注意 `URN` 是处理那些 `SGML` 用 `FPI` 来解决的问题的基于 `Web` 的解决方案。它们也有类似的实际问题——为专横的客户端软件来注册标记是非常困难的。`SGML` 团体用 `catalog` 文件解决了这个问题,但是 `Internet` 团体计划用分布的名字服务来解决这个问题。

16.5.2 诊断 Locale

许多应用程序非常简单,因为使用它们的人都讲相同的语言。如果你的应用程序就是这样,那么大部分 API 都可以做得很好,因为他们可以使用全局的 `System.getDefaultLocale` 方法来确定生成消息的时候应当使用哪种语言。然后应用程序应当使用 `java.util.ResourceBundle` 对象来组织这些消息,并用 `java.text.MessageFormat` 对象来格式化这些消息(然而不是所有的应用程序都这么做!)

`Web` 不是这么简单。服务器机械地接收全世界的请求,而很多用户可能更愿意使用他们的母语,尽管他们的母语可能不是服务器管理员的语言。不幸的是,大多数 API 在这个问题上都犯了错误,因为它们假设只使用一种语言。

`SAX` 使应用程序可以指定解析器在它的诊断中所使用的语言。比如,如果一个 `HTTP/1.1` `Web` 服务器要求诊断使用英文、德文和日文,`HTTP` 语言协商功能可以选出每一个客户最合适的诊断语言。

作为配置 `SAX` 解析器的一部分,使用 `Parser.setLocale` 方法(在解析文件之前的任何时候——它不像处理函数)来指定选用的诊断 `locale`:

```
parser.setLocale(locale);
```

这个方法会在所选的 locale 不被支持的时候产生一个异常,这样就允许调用者尝试使用其他的 locale。

16.5.3 对未解析实体和注释使用 DTD Handler

除非你面对的是具有 SGML 背景的系统,否则你将不会使用 XML 的未解析实体和注释。如果你使用了这些特征,那么你会很高兴看到 DTD Handler 也是 SAX 的一部分。

这个接口有两个回调。你可以认为它们是在解析器在 DTD 中遇到未解析实体或注释的声明时被调用。但是,解析器实际上可以推迟对它们的调用并且可以用任何顺序来调用它们。你已经知道这些回调都是在 DocumentHandler 遇到一个 startDocument 回调之后并且是在它遇到 startElement 回调之前被调用的。

你现在应当熟悉设计的模式了——实现一个处理函数,然后将它告诉解析器。下面是如何将这个处理函数告诉解析器:

```
parser.setDTDHandler(handler);
```

清单 16.10 中的处理函数仅仅打印与回调相关的那些声明。

清单 16.10 打印 DTD 声明的一个 DTDHandler 的实现

```
public class EchoDTDHandler implements DTDHandler
{
    public void unparsedEntityDecl (
        String name,
        String pubid,
        String sysid,
        String notation
    ) throws SAXException
    {
        System.out.print("<! ENTITY " + name);
        if (pubid != NULL)
            System.out.print("PUBLIC ' " + pubid + " ' ' ")
        else
            System.out.print("SYSTEM ' ");
        System.out.println(sysid + "' NDATA " + notation + ">");
    }

    public void notationDecl (String name, String pubid, String sysid)
    throws SAXException
    {
        System.out.print("<! NOTATION " + name + " _");
        if (pubid != NULL) {
            System.out.print("PUBLIC ' " + pubid);
            if (sysid != null)
                System.out.print("' ' " + sysid);
        } else
            System.out.print("SYSTEM ' " + sysid);
        System.out.println(">");
    }
}
```

你还记得 NOTATION 声明在 XML 语法中是一个可以具有和 SYSTEM 标记不匹配的 PUBLIC 标记的地方吗?

通常的应用程序都保存这些声明。它可以建立两个集合(或字典),用每个注释或实体的名字作为入口,并用它们来保存其他回调参数的值。

你还需要了解什么时候用它来保存数据。我们使用 SAX 实在是一件好事,因为 DOM 将一些事情隐藏了起来!一般你只要看一看传给文件处理函数的 startElement 回调的 AttributeList 就可以了(除非你的解析器不读外部参数实体,而且在那里定义了这些属性)。对每一个属性使用 AttributeList.getType。如果属性的类型是:

- ENTITY——这个属性值可以是一个和这个元素相关的单一未解析实体的名字。
- ENTITIES——这个属性值是和这个元素相关的一系列用空格隔开的未解析实体的名字。
- NOTATION——这个属性值可以是一个单一注释的名字,这个注释用来解释该属性所属的元素。

说明系统为什么要使用这些特征的一个经典例子就是可以用来描述图形格式(如 JPEG 或 PNG)的注释,以及使用这样格式的图形的未解析实体。应用程序需要处理任何与元素相关的未解析实体——可能是要处理的图形(这些实体是未解析的,因为 XML 解析器显然不能解析或处理二进制的图形文件!这些都要由应用程序来处理)。

和面向 SGML 的解决方案不同,XML 面对的是 Web。这意味着新的 XML 系统都设计为使用 Web 相关的技术,而不是建筑在其他原有机制之上。注释解决了 MIME 类型机制所涉及的那些问题,但是大多数 Web 软件都更能理解 MIME 类型 image/gif,而不是它相应的公开 ID——+//ISBN 0-7923-9432-1:: Graphic Notation//NOTATION CompuServe Graphic Interchange Format//EN。使用 MIME 类型也比较好记而且 MIME 类型的名称比较短。另外,还有很多链接到其他类型 Web 资源上的链接都需要分别声明实体,这些实体看上去不必要那么复杂。想像 HTML 页面中的每一个图形和链接都必须在独立使用之前被定义,即可明白。

XML 规范还注意到 NOTATION 声明可能被用来正式地声明 PI 目标名。解释这个建议的一个比较安全的方法是 PI 目标名应当被用在检查注释的 processingInstruction 回调中,而且在文件使用 PI 时所做的工作应当决定于和注释相关的数据。比如,PI 目标 abc 可能是指具有某个特定 PUBLIC 标记的 NOTATION。这个 PUBLIC 标记(或者对于 URN 来说是 SYSTEM 标记)应当和值得相信的代码相关,这个代码可以用传给 PID 的数据来调用。应用程序可以控制这种联系,而不是由文件来控制。

16.6 SAX 2.0

SAX 1.0 在 1998 年初就比较成熟了。一些应用程序上传它所提供的特征。举例说来,DOM Level 1 的 XML 功能需要访问某些 XML 编辑器应用程序可能使用的一些信息,比如注释和解析实体的信息。一些应用程序需要了解一些其他信息,比如文件类型、元素、属性声明,这样它们就可以验证结构是否是构建 XML 内容的一部分。

相应地,一些 SAX 的扩展也被定义了,称为 SAX 2.0。目前,它们还处在草案阶段。功能上可能没有太大改变,但是 API 可能会发生一些变化。这些 API 我们就不赘述了。

解析器会支持一个新的 Configurable 接口来输出这些功能。它被分为两组:布尔特征标志和可以取任何值的属性。它们被称为 URI,这样这套功能就可以是无限制的。在 SAX 2.0 中定义了这样的 URI,但是所有这些特征都不是必须的——解析器可以不必认识它们。

16.6.1 解析器特征标志

如前所述,不是所有的解析器都是一样的。许多实质上的差别可以用布尔特征标志来描述。在很多情况下,解析器可以打开或者关闭这些特征。SAX 2.0 包括了取得和设置这些标志的 API,而且还定义了这些标准的标志。这些标志的名字以 `http://xml.org/sax/features/` 开头并且用特定特征的名字结尾。

- validation——解析器是否验证输入?
- external-general-entities——解析器是否读取外部一般实体?
- external-parameter-entities——解析器是否读取外部参数实体?
- use-locator——解析器是否提供 locator 对象?
- normalize-text——解析器是否内部缓冲内容回调,这样应用程序就不必再作了?
- namespaces——解析器是否修改元素名或属性名以包括它们名字空间中的任何 URI (请注意这样的修改在 XML 1.0 中是非法的)?

在取得这些标志值的时候,解析器可能会报告不认识这个特征。当设置这些标志的时候,解析器可能报告不支持这样的设置。这样就允许解析器报告修改了的属性的值(“是的,我提供一个定位标志”)而不必要求这些值都可以设为非缺省值。

16.6.2 解析器属性值

解析器的属性被称做对象。目前有两种属性——普通属性和处理一些新型事件回调的函数。

预定义的属性名是以 `http://xml.org/sax/properties/` 开头的字符串,而且要以下面这些特定的名字结尾:

- namespace-sep——如果支持名字空间特征,这个字符串被用来修改元素名或属性名。
- dom-node——SAX 解析器可能被用来遍历 DOM 树,就像在前面 DOM-to-SAX 的练习中做的那样。这个属性为解析器提供对当前 DOM 节点的访问,这样就可以将 DOM 树作为输入而不是 XML 内容了。
- xml-string——一些解析器可以提供对未解析字符的访问,并产生一个指定的 SAX 事件,比如完整的开始标记产生了 `startElement` 事件。它提供了对这样的字符串的访问。

URI 处理函数以 `http://xml.org/sax/handlers/` 开头,后跟一个名字:

- Declhandler——如果解析器想要输出元素、属性和解析实体的 DTD 声明,那么它可

能会支持这个新的处理函数。

- `Lexicalhandler`——如果解析器想要看到编辑事件并报告一般不作为语法特征来处理的信息(除了那些 XML 编辑应用),那么它可能会支持这个新的处理函数。这些事件包括作者注释、CDATA 内容分隔符、解析实体引用的开始和结束以及 DTD 边界。
- `Namespacehandler`——如果解析器集成了 XML 名字空间处理,那么它可能会支持这个新的处理函数。这个接口在名字空间前缀声明(`xmlns * 属性`)的开始和结束时报告事件。

和 SAX 2.0 标志一样,取得属性值(或处理函数)可能会报告不认识这个特征,而设置其值的时候可能会报告所设定的值不支持。

16.7 总 结

SAX 1.0 得到了 XML 解析器的广泛支持。它是一个简单的基于事件的 API,你可以在不太了解系统的情况下使用它。尽管核心 API 有近 12 个类,但是你可以只用大概 6 个基本的调用来设置和调用解析器,处理解析器的元素和内容回调,就可以编写很好的应用程序了。

本章介绍了 SAX 解析器对象模型,你可以用 4 种不同类型的可插拔处理组件来配置它。你还学到了这些组件中最基本的是 `DocumentHandler`,而许多应用程序会配置 `ErrorHandler` 来确保它们可以有效地报告和处理错误。你看到了如何配置解析器,以便让它从你提供的源程序中访问实体内容,你还看到了如何使用 SGML 社团共享的这些 XML 模型。最后你还看到了即将出台的改版的与 SAX API 相关的一些问题。

第4部分 XML与Web

第17章 浏览XML

如你所知,Web 是只有通过浏览器才能看到的世界。虽然 XML 解决了许多 HTML 的问题,但是 XML 文件与能够看到这些文件的浏览器同样重要。这就要求浏览器支持 XML 及其许多密切相关的技术,诸如 XML 名字空间,CSS,XSL,DOM 等。没有可靠的浏览器的支持,XML 的命运也会像其他没有受到欢迎的 Web 技术一样转瞬即逝。Web 技术的存亡取决于能否得到大型浏览器厂商的支持。

不幸的是,在我编写本章的时候,浏览器对 XML 的支持还问题重重。在各大浏览器中,Microsoft 的 Internet Explorer 是惟一支持 XML 的商业 Web 浏览器,而它的支持还远非完美。Netscape 作出许多承诺,它的即将问世的浏览器将配有 Gecko 自动布局机。从这一点来看,它将给予 XML 相当可靠的支持。然而,使用者现在还无法使用即将问世的软件。我们只能着眼于现在,XML 和浏览器的问题就在于此。本章将探讨主要的 Web 浏览器对 XML 支持的现状,并对未来的发展情况进行展望。

17.1 XML 和 Web 浏览器现状

XML 是当前最激动人心、功能最强的 Web 技术。然而,Web 浏览器对它的支持却是惊人的可怜。只要想一下 XML 问世已经有多久(1998 年 2 月问世),你就会吃惊于至今 Web 浏览器对它的支持仍显不足。诚然,Microsoft 还是在 Internet Explorer 5.0 中迅速的对 XML 提供了支持。但是,在本章中你会看到,Microsoft 所达到的 XML 标准还是十分勉强的。在本书编写的时候,Netscape 准备推出 Navigator 5.0。Navigator 5.0 可能更接近 XML 标准,结果如何,我们将拭目以待。

我不想在此絮叨不已,但标准使得计算机有可能彼此可靠的相互协作。W3C 的宗旨就是提供描述网络技术标准及其具体实现的稳定基础。作为 Web 开发人员和使用者,我们曾经历过在没有任何标准的情况下的网络运行,其结果是提供商专有 HTML 标记以及其他相关技术的一团混乱。这就造成了使用者实际上不可能在不同的平台和浏览器上使用同一 Web。

要使 XML 成功,Web 浏览器厂商必须严格的遵守 W3C 的 XML 标准,否则就会使 XML 失去意义并使 Web 上的大量信息格式无法统一。XML 的处境十分有趣,它即将成为一种无所不在的 Web 技术,而这又仰仗于 Web 浏览器厂商的支持。

注意:要了解如何有助于浏览器厂商支持包括 XML 在内的 Web 标准的相关信息,请查看网址 <http://www.webstandards.org>。

下面是当前支持 XML 的 Web 浏览器以及将在未来的版本中提供对 XML 支持的浏览器:

- Microsoft Internet Explorer
- Netscape Navigator(Mozilla)
- Citec 的 DocZilla
- W3C 的 Amaya

本章的其余部分将讨论这些浏览器及它们对 XML 的支持。显然,我将用更多的篇幅关注两大浏览器,Internet Explorer 和 Navigator,因为它们是目前广泛使用的浏览器。

17.2 Microsoft Internet Explorer

尽管 Microsoft 热衷于在 Microsoft Internet Explorer 5.0 中加入对 XML 不一致的支持,你还是要承认这是第一家由厂商推出的具有 XML 支持的商业浏览器。Internet Explorer 对 XML 的支持存在的主要问题是它必须通过 Microsoft 专有的方式处理 XML 问题。这是因为,Internet Explorer 发布时,XSL 还处于 W3C 草案阶段,而 Microsoft 走在了前头,它具有对 XSL 工作草案稍加改进的版本的的支持。但它没有考虑到这对于开发者依据即将成为官方标准的技术进行代码编写方面所产生的负面影响。

此外,Microsoft 是 XSL 的有力支持者。这可能是它在 Internet Explorer 中迅速推行 XSL 的部分原因。Internet Explorer 对 CSS 的支持多少有点不尽如人意可能也是原因之一。在 XML 组织中就 CSS 和样式化 XML 代码的 XSL 的使用问题上有过重大的争论。显然,使 Web 浏览器同时支持这两种方式符合开发人员和使用者双方的利益。

以下几部分将探讨对 Internet Explorer 5.0 中与 XML 相关的特定部分。

17.2.1 XML 原始代码

Internet Explorer 5.0 最具特色之处就是你能在浏览器上直接看到 XML 原始代码。当你打开 XML 文件且其并没有与样式表结合时,Internet Explorer 就会以树型结构显示文件。这个树,在每一分枝旁有一个加/减符号:这是你可以用于展开或隐含树的相应部分。同时,还以不同的颜色显示出树的内容,以区别 XML 标记、数据及处理机指令。图 17.1 显示了在 Internet Explorer 5.0 中看到的 XML 原始代码的情况。

17.2.2 XML 错误处理

Internet Explorer 5.0 看来十分擅长发现 XML 文本中的错误。尽管一些 Web 开发人员认为挑剔的浏览器真是令人讨厌,但浏览器越挑剔越好。若你拼写错了标记或创建了不适当的标记,你还是希望浏览器能够立刻反映出来。人们认为在显示 XML 文件时,如果 Web 浏览器没有显示错误,那么文件则被认为是格式正确的。图 17.2 显示了在你指定 DTD 而忘记包括 DTD 文件的时候所发生的情况。

如果你违反了 XML1.0 规范中的语法规则,Internet Explorer 就会显示出来。这是一件好事。图 17.3 显示了在 XML 文件中,你拼错了 start 标记所发生的情况。



图 17.1 XML 文件以原始代码形式在 Internet Explorer 5.0 中的显示情况



图 17.2 Internet Explorer 5.0 中缺少 DTD 时产生的错误

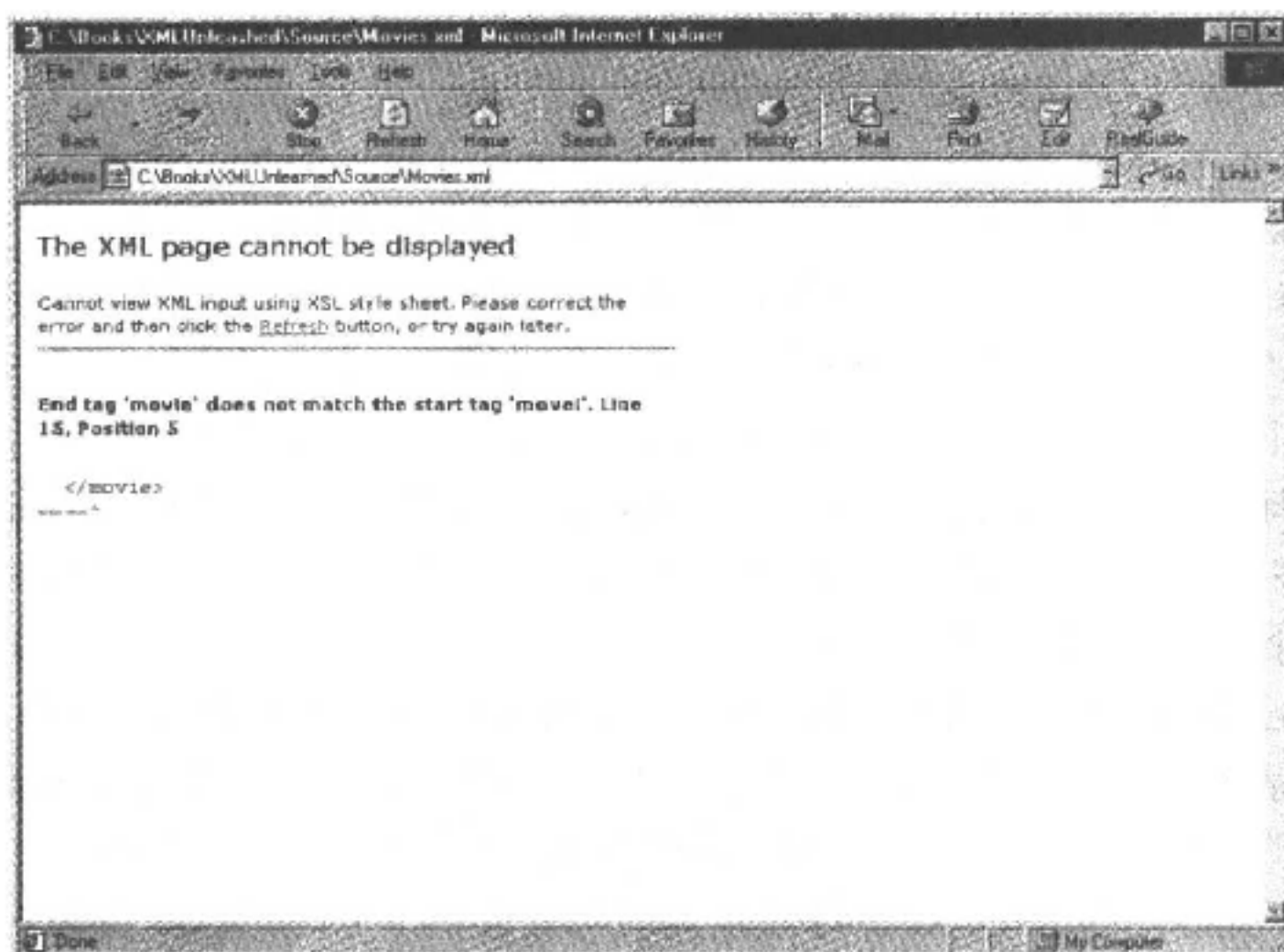


图 17.3 Internet Explorer 5.0 中拼错 start 标记时产生的错误

17.2.3 CSS 和 XSL

至此,我已讲到 Internet Explorer 5.0 的一些优点及其对 XML 的支持情况。现在该是关注 Internet Explorer 5.0 的不足之处的时候了。尽管层叠样式表(CSS)并不是专门用于 XML 技术的,但却被用作样式化 XML 文件的一种手段。在第 9 章“使用层叠样式表格式化 XML”中,你已经知道了如何做到这一点。

CSS 的目的是提供简单的样式表机制,以允许 Web 开发人员在 HTML 文件中加入样式,包括控制诸如字体、色彩和间距等的样式特征。在 1996 年 12 月,CSS 已被作为 Web 标准,但从当前浏览器的支持情况来看,你也许会认为它是几周前才被标准化了的。Internet Explorer 4.0 并没有完全遵循 CSS 标准,而且从表面上看,就 Internet Explorer 5.0 来说,Microsoft 也没有把遵循 CSS 标准作为优先考虑的问题。W3C 已经使 Internet Explorer 5.0 经历了一系列的 CSS 测试,大部分以失败告终。这也就是说,CSS 是不能在 Internet Explorer 5.0 运行的,因为它没有遵循 W3C 标准。

Microsoft 通过提供自身对 CSS 的扩展,忽略了 CSS 的某些功能。这也就造成了与其他浏览器的不兼容。这种扩展虽然可能对 Internet Explorer 的使用者有利,但却造成了 Web 的不完备,使 Web 的发展更加困难。Microsoft 的态度是你应该使用 XSL 来样式化 XML 文件,而不是使用 CSS。然而,XML 开发人员并不想被束缚在像 XSL 这样指定的解决方案上。而且 Microsoft 的 XSL 实现并不是基于一个现存的标准。这意味着在将来的版本中,可能会有所改变。

可扩展样式语言(XSL)为执行 XML 样式表提供了更有力的方法。这种样式表用来将 XML 文件转换为可以显示在 Web 浏览器上的一种格式。XSL 语言比 CSS 功能更强,但也

更为复杂。这种复杂性招致了 CSS 忠实拥护者的批评。XSL 和 CSS 将在第 8 章“对比两种样式规范:可扩展样式表语言和层叠样式表”中进行比较与对照。XSL 将在第 10 章“理解 XSL”和第 11 章“创建 XSL 样式表”中进行详细论述。

与 CSS 不同,XSL 依然处于 W3C 工作草案阶段。这说明 XSL 依然处于 Web 标准的界定时期。XSL 的最新工作草案在 1999 年 4 月推出。XSL 依然处于开发中这一事实并没有使 Microsoft 停止在 Internet Explorer 5.0 中对 XSL 进行支持。尽管为此 Microsoft 受到批评,但它仍然以实现对 XML 的支持作为工作目标。多数 XML 开发人员认为 Microsoft 对 XSL 的支持的实际问题就是 Internet Explorer 5.0 偏重于 XSL。由于 CSS 是多年来已被证明有效的标准,而 XSL 还只是工作草案。从浏览器的支持角度上看,选择 XSL 而非 CSS 似乎没有什么道理。创建可同时使用 CSS 和 XSL 样式表的 XML 文件本身就有问题。Internet Explorer 将使用 XSL 样式表。

Internet Explorer 5.0 支持 XSL 的另一个问题就是 Microsoft 对只在 Internet Explorer 5.0 下工作的 XSL(第 2 章已论及)进行了专门的扩展(这种扩展不可能被收入最终的 W3C 的 XSL 标准中)。这意味着使用这种扩展的样式表只能在 Internet Explorer 5.0 上工作。甚至于符合 W3C 工作草案的 XSL 特性也可能与最终的 XSL 不兼容,因为 XSL 必然要有变化。

为维护 Microsoft,我只好说应该称赞它跨入领先地位并提供了要修补的 XSL 实现。如果你认为 Internet Explorer 对 XSL 的支持不过是一种修修补补的技术,那就太对了。然而,如果你要认真的开发样式表,你可能要考虑使用 CSS,直到有最终的 XSL 规范出现为止。另一个选择是使用 Internet Explorer 5.0 支持的 XSL,只是在最终的 XSL 规范实施时,做好准备修改你的代码。

17.2.4 XML 名字空间

XML 名字空间最早出现在 W3C 的议案中,并在 1999 年 1 月得到了推荐。尽管“XML 中的名字空间”建议相当新,但是由于需要,它已经迅速被采用。XML 名字空间提供了在 XML 文件中避免名字冲突的一种机制。Internet Explorer 5.0 是第一种支持 XML 名字空间的商用 Web 浏览器。然而,Microsoft 的实现有些已经超出了 W3C 的“XML 的名字空间”建议。

Internet Explorer 5.0 在名字空间上的主要问题看上去并没有什么,但是无论如何,它是和 W3C 建议相抵触的。我是指 Internet Explorer 5.0 使用硬编码的 html 前缀来标识 HTML 名字空间。根据 W3C 的建议,开发者应当能够使用任何前缀。硬编码的 html 前缀与这一规则相抵触。我再次声明,这不是一个大问题,但是却值得指出。

17.2.5 XML Schema

正如我们在第 5 章“XML Schema 基础”和第 6 章“XML Schema 构造技术”中学习的,XML Schema 是 Microsoft 的一项技术,它实现了 W3C 替代 DTD 的一种供选择的建议的子集。W3C 的这个建议旨在改善用 DTD 来描述 XML 文件结构的方法。XML Schema 本身可以作为 XML 词表来使用,这样,它比 DTD 更容易学习和使用。XML Schema 修改了 DTD 的很多缺点,随着 XML 的不断发展,它可能会取代 DTD。在设计 Internet Explorer 5.0 的时

候, W3C 的 XML Schema 最终版本还没有出台, 这大概就是 Microsoft 决定使用它自己版本的 XML Schema 的原因。幸运的是, Microsoft 是根据 W3C 建议的精神来使用 XML Schema 的。

尽管 Internet Explorer 5.0 支持的是正在修改中的 XML Schema, 但是它还是很成功的。Microsoft 的 XML Schema 版本在某些方面有些不足, 但是它为规范(而不是 DTD)描述 XML 文件的结构打下了很好的基础。我们应当像看待 Microsoft 的 Internet Explorer 5.0 对 XSL 的支持一样来看待它对于 XML Schema 的支持。Microsoft 的 XML Schema 是一项很有趣的技术, 也是可应用的实现, 但是如果你真的要进行开发的话, 你一定要十分小心, 因为这一技术尚不成熟。希望 W3C 能尽快开发出 XML Schema, 也希望 Microsoft 的 Internet Explorer 能够尽快地完全遵守它。

17.2.6 CDF 和 VML

除了你到目前所了解的 XML 核心技术之外, Internet Explorer 5.0 还支持两个特别的 XML 词表, 用来表示某些特定的信息类型。这两个词表就是频道定义格式(CDF, Channel Definition Format)和矢量标记语言(VML, Vector Markup Language)。

CDF 是第一种商用的 XML 词表, 它是 Microsoft 的 Active 频道(Active Channel)技术的基础。Active 频道描述了可以自动更新的 Web 页面的层次。使用 CDF 使用者可以预订频道, 一旦频道中的内容有所改变, 就可以得到通知或者自动地进行更新。我们将在第 33 章“使用 CDF 推出 Web 内容”中详细介绍 CDF。

VML 是一种首先由 Microsoft 开发的词表, 它支持多种图形特征矢量, 这些矢量的基础是描述相连线段和曲线的轨迹。1998 年 5 月, VML 作为一个规范被提交给 W3C。尽管这个规范已经实行了一段时间, VML 仍然不得不努力实现 Web 上的广泛使用。事实上, Internet Explorer 5.0 是第一个支持 VML 的 Web 浏览器。这对于 XML 来说是一件好事, 因为就浏览器来讲, 还没有其他的矢量图形词表得到直接的支持。我们将在第 34 章“使用 VML 和 SVG 描述矢量图形”中详细介绍 VML。

17.3 Netscape Navigator(Mozilla)

目前, Netscape Navigator 的最新版本是 4.0 版, 它是一个独立的浏览器, 也是 Netscape Communicator 4.61 软件包的一部分。这个 Netscape Navigator 版本不支持 XML。因此, 关于 XML 和 Netscape 的话题只好着眼于未来。Netscape Navigator 的未来将决定于 Mozilla 源代码开放行动。

注意: 源代码开放行动是由分布于世界各地的一些软件开发者发起的, 他们认为应当公开源代码而且认为其他人也应当这样做。Netscape 公开 Navigator 的源代码也并非纯粹出于利他主义, 而是它认识到了保持并有望扩展 Navigator 的用户基础的商业价值, 这样, 他们可以继续开发那些收费产品。

1998 年 1 月, Netscape 曾透露, 不仅 Netscape Navigator 的未来版本将是免费的, 而且源代码也可以无偿地使用。到 1998 年 3 月底, 它就发布了 Navigator 的一个专业版, 并且包

含了完整的源代码。这次源代码开放行动使得 Mozilla 的开发达到顶峰,这样可以利用 Navigator 集体对源代码进行开放的开发,这正是 Netscape 计划的一部分。尽管 Mozilla 是 Netscape 的一部分,但在某种程度上却可以看作是一个独立的实体。Mozilla 的网址是 <http://www.mozilla.org>。

注意: Mozilla 是 Netscape 监控 Netscape Navigator (Communicator) 的未来版本以及 Mozilla 本身的未来版本的那个部门的名字。你也许听说过,Navigator 的下一个版本叫做“Gecko”,它是 Mozilla/Navigator 浏览器内部使用的显示引擎的代码的名字。

因为目前 Navigator 的下一个主版本仍然处在开发之中,所以很难做到对 XML 进行非常精确的支持,然而,浏览器的预发布版表明 Mozilla 将相当认真的支持 XML 标准。

17.3.1 CSS 和 XSL

在 Internet Explorer 5.0 对 XML 的支持中,受到批评最多的地方可能就是对 XML 文件的样式化了,因为这既可以用 CSS 又可以用 XSL 来实现。Microsoft 受到批评是因为它在不完全地支持已经确定的 CSS 技术的同时,又仓促行动,支持还处于开发之中的 XSL 技术。Microsoft 并不十分重视对 XSL 的支持,因为它认为 XSL 只是“参考实现 (reference implementation)”。这种对 CSS 的不完全支持,造成了 Web 开发界的混乱。

显然,Netscape 已经从 Microsoft 的错误中吸取了教训,在 Mozilla 中看来并没有对 XSL 的支持。相反,它坚持使用经过证明的 CSS 标准,而这一标准也完全可以满足为了显示而对 XML 文件进行的样式化。千万不要误解我的意思——XSL 是一种具有难以置信的功能的技术。我本人喜欢它,因为它代表了未来。CSS 是现行的标准,这仅仅说明现在使用 CSS 具有其合理性。这并不是说,人们应当在 XML 标准化以后再去关注它。这仅仅说明无论是现在还是不远的未来,CSS 对于提供实际的 XML 解决方案来说都是更为实用的。浏览器的预发布版表明了 Mozilla 对 CSS 支持的承诺,如果它通过了 W3C 的布局测试,那么 Mozilla 的境况要比 Internet Explorer 好得多。

17.3.2 Expat 解析器

Mozilla 没有使用它自己的 XML 解析器,而是获得了非常流行的由 James Clark 编写的 Expat XML 解析器的许可。Expat 是用 C 开发的,完全遵守非验证的 XML 1.0 解析器。一般认为,它是最可靠的 XML 解析器。Mozilla 利用这样稳定的 XML 软件组件是很明智的。关于 Expat 解析器的更多信息,可以在 James Clark 的 Expat 主页中找到,网址是 <http://www.jclark.com/xml/expat.html>。

17.3.3 XUL 和 RDF

除了在文件级对 XML 支持之外,Mozilla 在浏览器本身的开发中也使用了 XML。它开发了一个叫做 XUL (可扩展的用户界面接口) 的 XML 词表,用来描述窗口和对话框中各构件的布局。XUL 构成了 Mozilla 浏览器设计的主要部分,最终它将负责 Mozilla 的高级图形用户界面。

XUL 是基于 RDF 的。RDF 是一个 XML 词表,它可以用来为 Web 资源增加语义。大多数网络资源只含有内容,而没有对于内容性质的信息。RDF 提供了一种用一个 XML 词表来描述 Web 资源的方法。

搜索引擎大概是自动处理 Web 资源的最好的例子。大多数搜索引擎不停地查看 Web 资源,试图确定它所包含的数据的性质。每一个搜索引擎都用某种方法对 Web 资源的内容进行分析,并且用它自己的办法来提高搜索的准确度。通过允许资源的作者描述 Web 内容的确切性质,RDF 避免了大部分这种猜测。搜索引擎可以将 RDF 信息作为搜索的基础,因而不必对内容再进行猜测了。

17.4 其他浏览器

尽管在 Web 浏览器行业中,Microsoft 的 Internet Explorer 和 Netscape 的 Navigator 是最主要的产品,但是还有其他一些浏览器也值得一提。下面就是几种浏览器,有的已经支持 XML,也有的可能在不久引入对 XML 的支持。

- CITEC DocZilla
- W3C Amaya
- Opera

下面几节将介绍这些浏览器以及 XML 在它们中的作用。

17.4.1 CITEC DocZilla

CITEC DocZilla 是 Mozilla 源代码开放行动中形成的第一个非 Netscape 商业产品。CITEC 在技术文件浏览工具的开发方面有丰富的经验。它的 Multidoc Pro SGML 浏览器被认为是最快的 SGML 浏览器之一。DocZilla 浏览器是 Multidoc Pro 的替代产品,它基于 Netscape 的 Mozilla 公开源代码。目前,DocZilla 已经有了一个在 alpha 上的预发布版。

DocZilla 是一种支持包括 SGML 在内的多种文件类型的浏览器。这使它在浏览器市场中取得了独一无二的地位,因为任何的面向 Web 的浏览器去费力实现或者设计对 SGML 的支持都是不可取的。尽管 DocZilla 可能不会与其他的面向 Web 的对手直接竞争,但是在技术文件市场,它可能是相当受欢迎的。

对于 XML 的支持,DocZilla 提供了一系列特征,包括 SGML、XML、CSS、DOM 以及对 RDF 的支持。关于 DocZilla 的进一步信息,请参阅 DocZilla 主页 <http://www.doczilla.com>。

17.4.2 W3C Amaya

Amaya 是一个浏览和编辑工具。它是 W3C 作为新的 Web 协议和数据格式的展示工具和测试环境来创建的。基本思路是,W3C 将新的 Web 技术加入到 Amaya 中进行演示和测试,这样比 Web 浏览器厂商提供支持来说快一些。因此,技术上讲,Amaya 是一种主要用于测试的浏览器。

Amaya 包括对使用 CSS 来对 XML 样式化的有限支持。更为有趣的可能是 Amaya 支持 MathML 的表示标记部分,MathML 是一个设计用来对数学内容进行建模的 XML 词表。

Amaya 还以一种试验性的 XML 格式支持二维矢量图的创建和显示。在文件中,这些图形可以和 HTML 以及 MathML 混合在一起,并在 Amaya 中显示。关于 Amaya 的进一步信息,请参阅 Amaya 主页 <http://www.w3.org/Amaya>。

17.4.3 Opera

Opera 对效率的承诺得到了大量的关注。与两大主流浏览器不断的膨胀相反,与它的功能相比,Opera 的轻巧令人难以置信。不幸的是,目前 Opera 尚没有支持 XML 的计划。然而,Opera 非常稳定地实现了 CSS,这使它成为未来加入 XML 支持的理想后选者。让我们期望 Opera Software 在它未来的版本中支持 XML。关于 Opera 的进一步信息,请参阅 Opera 主页 <http://www.opera.com>。

17.5 总 结

本章试图准确地描绘出 Web 浏览器中 XML 的状态——主要是事实,一部分是观念。老实讲,评价当前的几种 Web 浏览器对 XML 的实际支持的时候,会发现 Web 浏览器厂商的大力宣传和它们实际的开发进展相差甚远。然而,我认为我已使你对使用目前的以及那些还未出现的浏览器有所了解。

公平地讲,浏览器的支持可能仍然是 XML 最不稳定的领域。这实在是非常遗憾,因为这个也是接受和使用 XML 最严格的领域。尽管目前对主要的浏览器厂商有众多的批评,但我仍然相信它们会作出正确的选择并且接受 W3C 推出的 XML 标准。时间会告诉我们是盲目的轻信还是英明的远见。

第 18 章 XHTML:XML 与 HTML 结合的产物

对刚接触 XML 的人来说,一个主要的困惑就是 XML 和 HTML 两者之间的关系。诚然,读者已经接触过了 XML,知道 XML 是一种元语言,而 HTML 是 SGML 的具体应用。正如你已经知道的,HTML 实际上是 SGML 的应用而不是 XML 的应用,这有很大差别。XML 描述了一种更为简洁的元语言,这使它比 SGML 更适用于基于 Web 的应用。这样,就可以理解为什么应当让 HTML 基于 XML 而不是 SGML 了。

尽管没有人能够挥动魔杖,将 Web 一下子转换成为基于 XML 的 HTML 版本,但是我们可以一步一步地实现 HTML 结构更完备的未来。W3C 引入了一种称为 XHTML 的方法,这是经过 XML 改造之后的一个 HTML 版本。在很多方面 XHTML 都和 HTML 4.0 相似,而且它无疑代表了 HTML 的未来。本章将介绍 XHTML 并讨论将网页从 HTML 移植到 XHTML 过程中可能出现的问题。

18.1 为什么要用 XHTML

在讨论 XHTML 是如何实现的并将它和 HTML 进行比较之前,有必要问这样一个简单的问题:为什么要使用 XHTML? 答案很简单,而且和最初创建 XML 的原因一致。

目前,Web 被搞得一团糟,因为陈旧的 HTML 代码结构很差。低劣的编码加上宽容的浏览器以及浏览器各不相同的扩展使 Web 结构非常糟糕,这的确不是什么好事。这里无须论述为什么这不是一件好事——本书的每一章讲的都是结构良好的文件的优点。

对 HTML 这个问题的一个合乎逻辑的答案也许就是将一切都转化为 XML 文件并用样式表来显示它们。要是地球上到处是和平,到处是美味的无脂食品,到处都缴很少的税,那将是多么的美好!但是生活却不是这样。我想要说的是,HTML 在 Web 上总还有一席之地,因为它根深蒂固,无可替代。此外,尽管与 CSS/XML 同时出现的 XML 比纯描述性的 HTML 网页更具有结构的优势,但是它的确要做很多工作。当然就出现了这样的情况:内容本身及显示内容的方式是否分离并不重要,这时 HTML 无疑是更为有效的解决方案。

最根本的一点是 HTML 依旧是存在的,不管是以何种形式。因此为什么不设法用 XML 来改进它? 将 HTML 表示为 XML 的应用(XHTML),你会轻而易举地享受到 XML 的许多益处。最基本的好处显然是结构上的。这最终将迫使浏览器厂商和 Web 开发人员都按照规则行事。浏览器要严格执行 HTML Schema 以确保文件是结构良好的并且有效。在这一点上,要求 HTML 文件必须是结构良好的就已向正确方向迈出了重要一步,而检查文件的有效性就更是锦上添花了。

XML 被看作是对 HTML 的有益改进的另外一个主要原因是,正在为 Internet 设备开发的新型浏览器就不必非要处理那些结构很差的 HTML 文件了。例如掌上机,个人数字设备,先进的手机就是使用这种浏览器的设备。这些无须办公桌的 Internet 设备靠高度结构化的 HTML 文件来使它们的处理代价达到最小。计划在 2002 年,Internet 访问量的 75% 将由

这样的设备产生。这更引起了对 HTML 文件规范结构化的热烈讨论。XML 是实现这种结构的一种完善的技术。

在可预见的未来,浏览器仍然要支持旧的、结构不完备的 HTML,然而最终这些过时的文件将被有效的、结构良好的文件取代,即使对于那些最顽固反对 XML 的人也将使用这此具有大量结构的文件。值得一提的另外一点是大部分的 Web 页面开发工作是通过可视化的编辑工具进行的,这些工具可以为开发人员自动生成 HTML 代码。只要改变这些工具,使之支持 XHTML 就将有助于迈向 HTML 的光明未来。

为了总结为什么用一个 XML 词表来编写 HTML 文件是 Web 发展中重要的一步,请考虑以下 XHTML 带来的好处,在 W3C 的 XHTML 工作草案中提到了它们:

- XHTML 文件基于 XML 语法,这意味着可以用标准的 XML 工具来显示、编辑和验证它们。
- XHTML 文件可以作为 text/html 介质类型。
- XHTML 文件还要能够作为 text/xml 介质类型,并支持适当的样式表。
- XHTML 文件要能够利用 HTML 文件对象模型或 XML 文件对象模型这样的编程技术。
- 符合 XHTML 的文件要尽可能地在不同的 XHTML 环境之中以及之间具有互操作性。

注意:XHTML1.0 是正式的 W3C 建议。这意味着它是需要浏览器厂商实现的稳定的规范。请访问 W3C 的 XHTML 网址 <http://www.w3c/TR/xhtml1>,以获得 XHTML 的最新信息。

18.2 XHTML 和 HTML 4.0 的差别

HTML 4.0 是 SGML 的一个作为标准 Web 出版语言的应用。正如你所知道的,XML 被描述为具有 SGML 的功能和灵活性而又没有不必要的复杂性的一种方法。XHTML 1.0 是 HTML 4.0 的一个改进版,它用 XML 的更严格的规则来结构化 HTML 4.0。幸运的是,XHTML 1.0 和 HTML 4.0 的主要区别是在语法上的,也就是说这些区别不会影响 HTML 文件的整个结构。把 HTML 4.0 文件移植为 XHTML 1.0 文件仅仅是整理代码的问题而不是要将它用新语言重写。

尽管在本章后面的“将 HTML 文件转化为 XHTML 文件”一节中,将详细介绍如何将 HTML 文件转化为 XHTML 文件,但是在这里介绍一下 XHTML 1.0 和 HTML 4.0 之间的一些基本区别仍然是值得的。下面是对 XHTML 文件的一些要求,它们揭示了 XHTML 与 HTML 的区别:

- 文件必须是结构良好的。
- 元素名和属性名必须小写。
- 非空元素必须使用结束标记。
- 空元素必须由一个开始标记/结束标记对或者一个空元素构成。
- 属性值必须加引号。

- 属性名没有赋值就不能使用。
- XHTML 名字空间必须在 html 元素中声明。
- head 和 body 元素不能省略。
- title 元素必须是 head 元素的第一个元素。
- 所有的 script 和 style 元素都必须在 CDATA 节中。
- 文件必须使用 id 属性来定义片段标记;为了定义文件片段,在 XHTML 中 name 属性已经被弃用。

根据你对 XML 的了解,这些区别不应当有什么问题。幸运的是,在 HTML 文件中找到和修改这些不同是非常容易的,这使得从 HTML 到 XHTML 的移植相当直截了当。

18.3 XHTML 和文件的有效性

为了验证 XHTML 的有效性,必须要有详细描述了 XHTML 结构的 DTD。W3C 没有开发验证 XHTML 文件的惟一的 DTD,相反,它更进一步,开发了三个 DTD。它们都包含在 XHTML 规范中。这些 DTD 和相应的 SGML DTD 类似,而且提供了 XHTML 不同级别的细节。这就产生了三个不同的 XHTML 文件类。也就是说,如果你不使用某些 XHTML 语言特征,就可以使用更低级的 XHTML DTD,相反,你也可以使用更完备的 DTD。这三种 XHTML DTD 如下,功能递增:

- 严格的 (Strict)——没有 HTML 表示元素 (font, table 等等) 可用,必须用 CSS/XSL 来显示文件。
- 过渡的 (transitional)——有可用的 HTML 表示元素来格式化文件。
- FrameSet——除 HTML 表示元素之外还定义了 Frame。

严格的 DTD 是最低级的 DTD,它被用来创建没有任何表示标记的 XHTML 文件。用这种 DTD 创建的文件必须用样式表来格式化显示,这是因为文件中不含任何表示标记。传统的 DTD 建立在严格的 DTD 之上,它加入了对表示标记的支持。如果你不想花时间开发样式表时,这样的 DTD 对快速转化 HTML 很有用。Frameset DTD 是三种 DTD 中应用范围最广的一种,它还支持创建带有框架的文件。

这些 DTD 形成了验证 XHTML 文件的基础。XHTML 文件的验证是任何 XML 词表中重要的一部分。除了符合上述三种 XHTML DTD 之外,有效的 XHTML 文件还必须声明与所用 DTD 相应的 XHTML 名字空间。下面是适用于所有 XHTML 文件的 XHTML 验证要求:

- 在文件的根元素之前必须有文件类型声明 (DOCTYPE)。
- 根据文件类型声明中定义的 DTD,文件必须有效;这个 DTD 必须是 XHTML 规范所提供的三种 DTD 之一。
- 文件的根元素必须是 html。
- 文件的根元素必须用 xmlns 属性在 XHTML 名字空间中指定。

18.3.1 声明 XHTML DTD

文件开头的文件类型声明中,必须声明 XHTML 文件的 DTD。在文件类型声明中用一个 Formal Public Identifier(FPI)来引用任一标准 XHTML DTD。下面是如何在文件类型声明中声明严格 DTD 的一个例子:

```
<! DOCTYPE html PUBLIC "-//W3c//DTD XHTML 1.0 Strict//EN"  
http://www.w3c.org/TR/xhtml1/DTD/strict.dtd>
```

理解这段代码中 FPI 的细节并不特别重要。主要是这个标记指出了 XHTML 的严格 DTD。用类似的代码也可以指定过渡 DTD,例子如下:

```
<! DOCTYPE html PUBLIC "-//W3c//DTD XHTML 1.0 Transitional//EN"  
http://www.w3c.org/TR/xhtml1/DTD/transitional.dtd>
```

最后,用下面的代码来指定 frameset DTD,看了前面两个文件类型声明,这就没有什么了:

```
<! DOCTYPE html PUBLIC "-//W3c//DTD XHTML 1.0 FrameSet//EN"  
http://www.w3c.org/TR/xhtml1/DTD/frameset.dtd>
```

18.3.2 声明 XHTML 名字空间

在文件类型声明中除了要声明正确的 DTD 之外,有效的 XHTML 文件还必须在 html 元素中声明 XHTML 名字空间。下面是与前面提到的三种 DTD 相关的三种 XHTML 名字空间:

- 严格的——<http://www.w3.org/TR/xhtml1/strict>。
- 过渡的——<http://www.w3.org/TR/xhtml1/transitional>。
- frameset——<http://www.w3.org/TR/xhtml1/frameset>。

在 html 元素中用 xmlns 名字空间属性来声明 XHTML 名字空间。这个名字空间必须和在文件类型声明中指定的 DTD 对应。下面是如何指定严格的名字空间的一个例子:

```
<html xmlns= http://www.w3.org/TR/xhtml1/strict>  
</html>
```

18.3.3 验证 XHTML 文件

什么时候进行 XHTML 文件的验证还没有确定。开发者在发布他们的 XHTML 文件之前验证它们的有效性,这的确比较快,可以减少浏览器进行验证的需求。另一方面,有很多 HTML 代码是脚本语言和其他交互工具随意生成的,因此有时让浏览器来验证 XHTML 文件的有效性可能非常必要。目前,还没有直接支持 XHTML 的浏览器,更不用说 XHTML 的验证了。

然而,不是说文件都不能进行 XHTML 验证了。W3C 提供了一个可以验证 XHTML 文件有效性的验证服务。这个服务是 <http://validator.w3c.org>。图 18.1 显示了 W3C 的验证服务的界面。

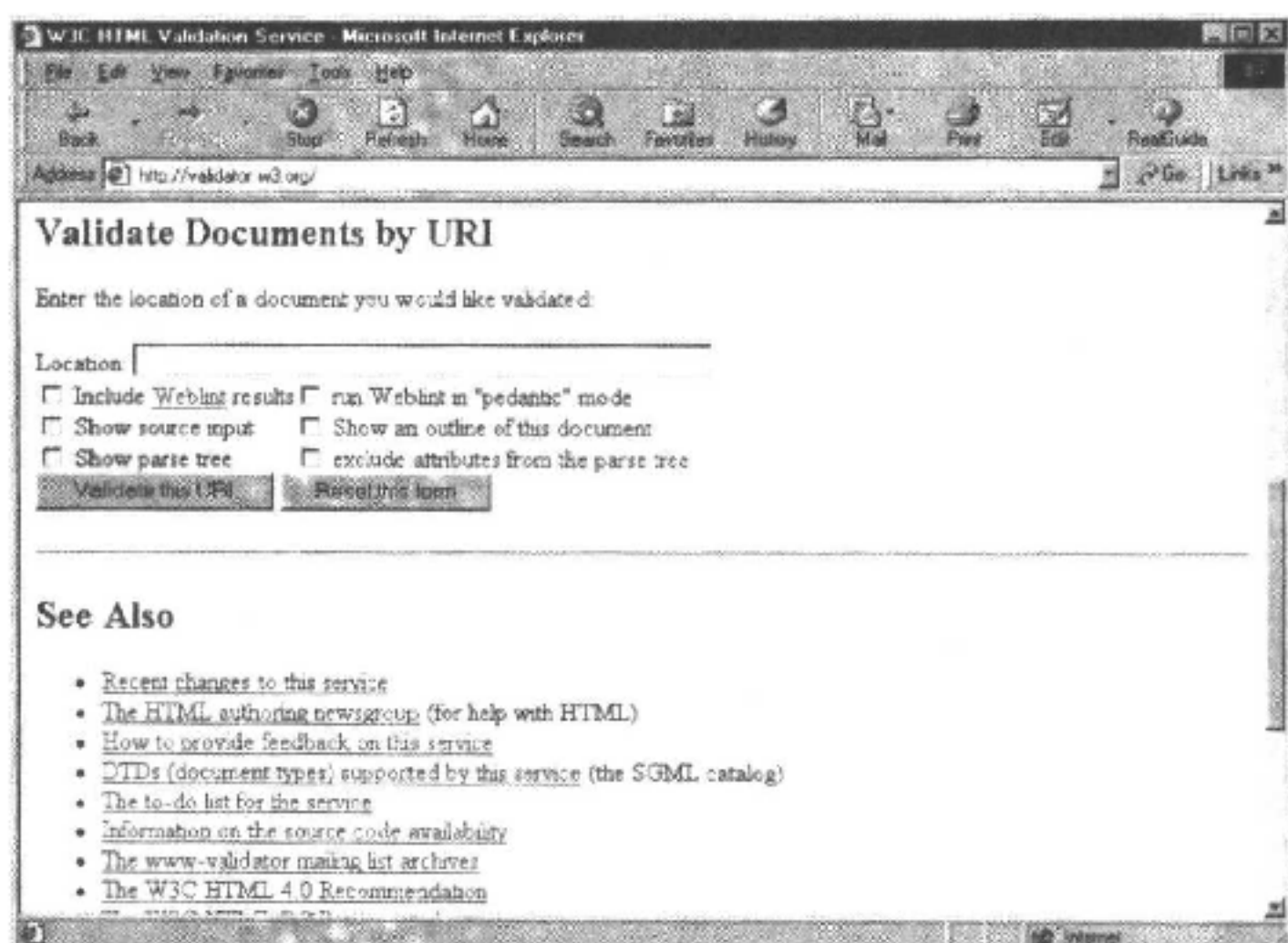


图 18.1 W3C 的验证服务

如图所示,我们可以输入 XHTML 文件的 URL,然后验证它。还有一种简化为验证服务指定文件的办法:简单地在文件中包括到页面 <http://validator.w3.org/check/referer> 的链接。当点击这个链接的时候,验证服务就会验证你的文件。

18.4 创建 XHTML 文件

实际上,创建 XHTML 文件和创建 HTML 文件没有什么不同。你只要记住 XHTML 和 HTML 的区别就可以了,这些区别在前面已经介绍过。清单 18.1 包含了一个 XHTML 框架文件的代码。

清单 18.1 一个 XHTML 框架文件(Skeleton.xhtml)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  http://www.w3.org/TR/xhtml1/DTD/strict.dtd>
<html xmlns="http://www.w3.org/TR/xhtml1/strict">
  <head>
    <title>Skeletal XHTML Document</title>
  </head>
  <body>
    <p>
      This is a skeletal XHTML document.
    </p>
  </body>
</html>
```

尽管这个 Web 框架文件做不了什么,但是它遵守了所有 XHTML 文件有效的规则,因此它是一个不错的 XHTML 模板。这个框架文件可以利用的主要构件是文件类型声明和名字空间的声明,它们都指定了 XHTML 严格 DTD。图 18.2 显示了 Internet Explorer 5.0 中显示的这个文件。

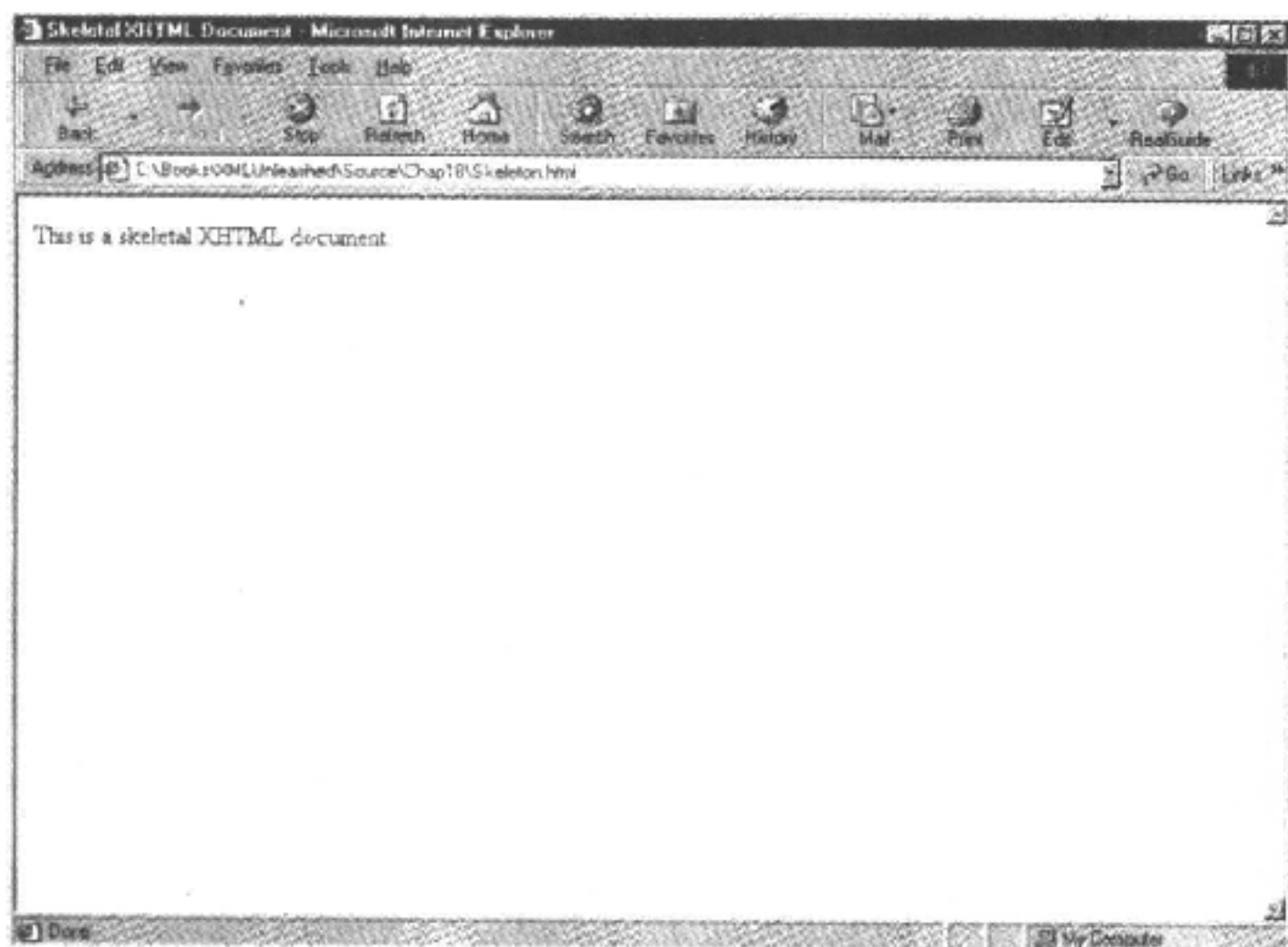


图 18.2 在 Internet Explorer 5.0 中显示的 XHTML 框架文件

前一节已经提到,W3C 提供了一个可以用来验证 XHTML 文件有效性的验证服务。我还说到可以在文件中加入一个链接,以便当点击这个链接的时候用这个验证服务来验证文件的有效性。清单 18.2 包含了修改后的 XHTML 框架文件,它可以用 W3C 验证服务来验证。

清单 18.2 包含了 W3C 验证服务的链接的 XHTML 框架文件(SkeletonVal.xhtml)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
http://www.w3.org/TR/xhtml1/DTD/strict.dtd>

<html xmlns="http://www.w3.org/TR/xhtml1/strict">
  <head>
    <title>Skeletal XHTML Document</title>
  </head>
  <body>
    <p>
      Click <a href="http://validator.w3.org/check/referer">here</a> to
      Validate this XHTML document.
    </p>
  </body>
</html>
```

当你点击“here”这个词的时候,文件会链接到 W3C 的验证服务并被验证。图 18.3 显示了验证的结果。

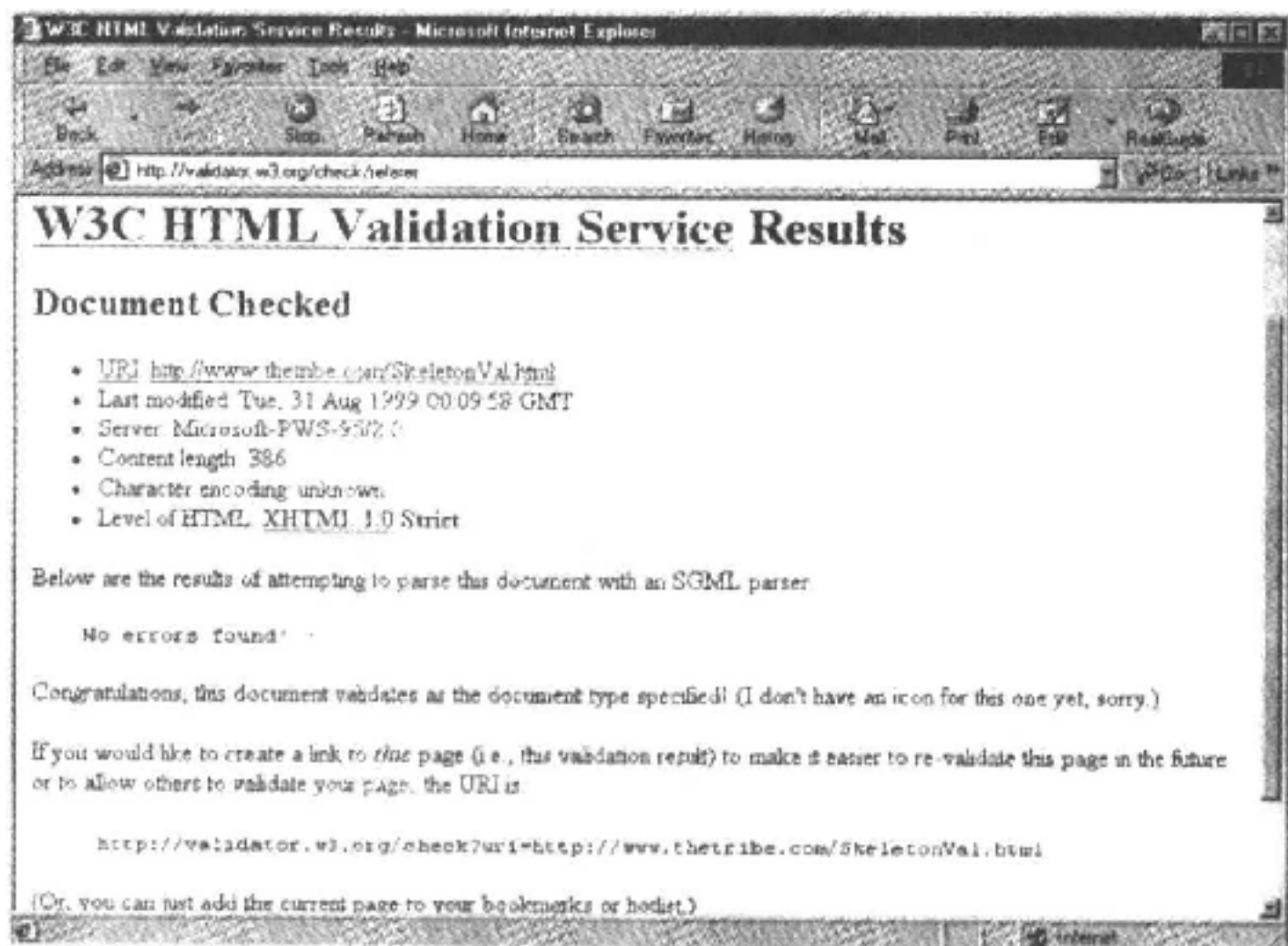


图 18.3 XHTML 框架文件经过 W3C 验证服务验证的结果

毫不奇怪,这个框架文件成功地通过了 W3C 验证服务的验证。当我们创建 XHTML 文件的时候,这是一个我们手头的工具,至少在主流 Web 开发工具支持 XHTML 之前是这样。

18.5 将 HTML 文件转化为 XHTML

尽管用 XHTML 重新创建 Web 页面是一个不错的想法,但是现实是你可能已经有了 HTML 文件,而且为了将来的应用,你需要把它们转化成 XHTML。幸运的是,将 HTML 4.0 文件修改为符合 XHTML 规范并不困难。你已经学习了 XHTML 文件和 HTML 4.0 文件不同的地方。这些不同将指导你将 HTML 转化为 XHTML。

有两种方法可以将 HTML 转化为 XHTML 文件:

- 手工转换。
- 用自动转换工具来转换。

后一种方法显然可以自动化转换过程并且省去许多枯燥的工作。然而,和任何自动处理一样,从 HTML 到 XHTML 的转换并不总是十分顺利。且这也是第一种方法比第二种更准确的原因,尽管它需要做大量的工作。一个折中的办法就是先用转换工具然后进行手工调整。

18.5.1 手工转换文件

尽管你已经学习了 HTML 和 XHTML 的区别,且这是在它们之间进行文件转换的根本基础,但是还是值得看一看这个检查清单,你可以用它来指导手工转换工作。下面就是在将 HTML 代码转化为 XHTML 代码的过程中,你所要完成的主要步骤:

1. 加入文件类型声明,以声明正确的 XHTML DTD(通常是传统 DTD)。
2. 在 html 元素中声明 DTD 相应的 XHTML 名字空间。
3. 将所有元素名和属性名转化为小写。
4. 将所有开始标记和结束标记匹配。
5. 将所有空元素的 > 替换为 />。
6. 将所有属性值用引号括起来。
7. 确认所有元素和属性都在文件类型声明中声明的 XHTML DTD 中定义。

如果你完成了上述步骤,你就得到了一个有效的 XHTML 文件。

一个简单的例子可以帮助我们解释这些步骤之间的关系。清单 18.3 包含了描述在线 Fruit Stand 的页面的 HTML 文件的代码。该页面如图 18.4 所示。

清单 18.3 Vernie 的 Fruit Stand Web 页面的原始 HTML(Vernie.html)

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
  <HEAD>
    <TITLE>Vernie's Fruit Stand</TITLE>
  </HEAD>
  <BODY BACKGROUND=Ivy.gif BGCOLOR=#FFFFFF>
    <P ALIGN=CENTER>
      <IMG SRC=Vernie.gif ALT="Vernie's Fruit Stand">
      <P ALIGN=CENTER>
        <FONT SIZE=4>Welcome to Vernie's Fruit Stand,an online marketplace
        dedicated to bringing you the freshest in fruit at rock bootm
        prices. </FONT>
      <P ALIGN=CENTER>
        <IMG SRC=Strawberry.gif HSPACE=15>
        <FONT SIZE=6>Today's Specials</FONT>
        <IMG SRC=Strawberry.gif HSPACE=15>
        <DIV ALIGN=CENTER>
          <TABLE BORDER=1 CELLPADDING=8 CELLSPACING=1>
            <TR>
              <TD><FONT SIZE=4>Fruit</FONT></TD>
              <TD><FONT SIZE=4>Price per unit</FONT></TD>
              <TD><FONT SIZE=4>Unit</FONT></TD>
            </TR>
            <TR>
              <TD>Navel Oranges</TD>
              <TD>$ 2. 49</TD>
              <TD>pound</TD>
            </TR>
```



```
<TR>
  <TD>Strawberries</TD>
  <TD>$ 2. 99</TD>
  <TD>quart</TD>
</TR>
<TR>
  <TD>Limes</TD>
  <TD>$ 1. 49</TD>
  <TD>pound</TD>
</TR>
<TR>
  <TD>Granny Smith Apples</TD>
  <TD>$ 1. 99</TD>
  <TD>Pound</TD>
</TR>
</TABLE>
</DIV>
</BODY>
</HTML>
```

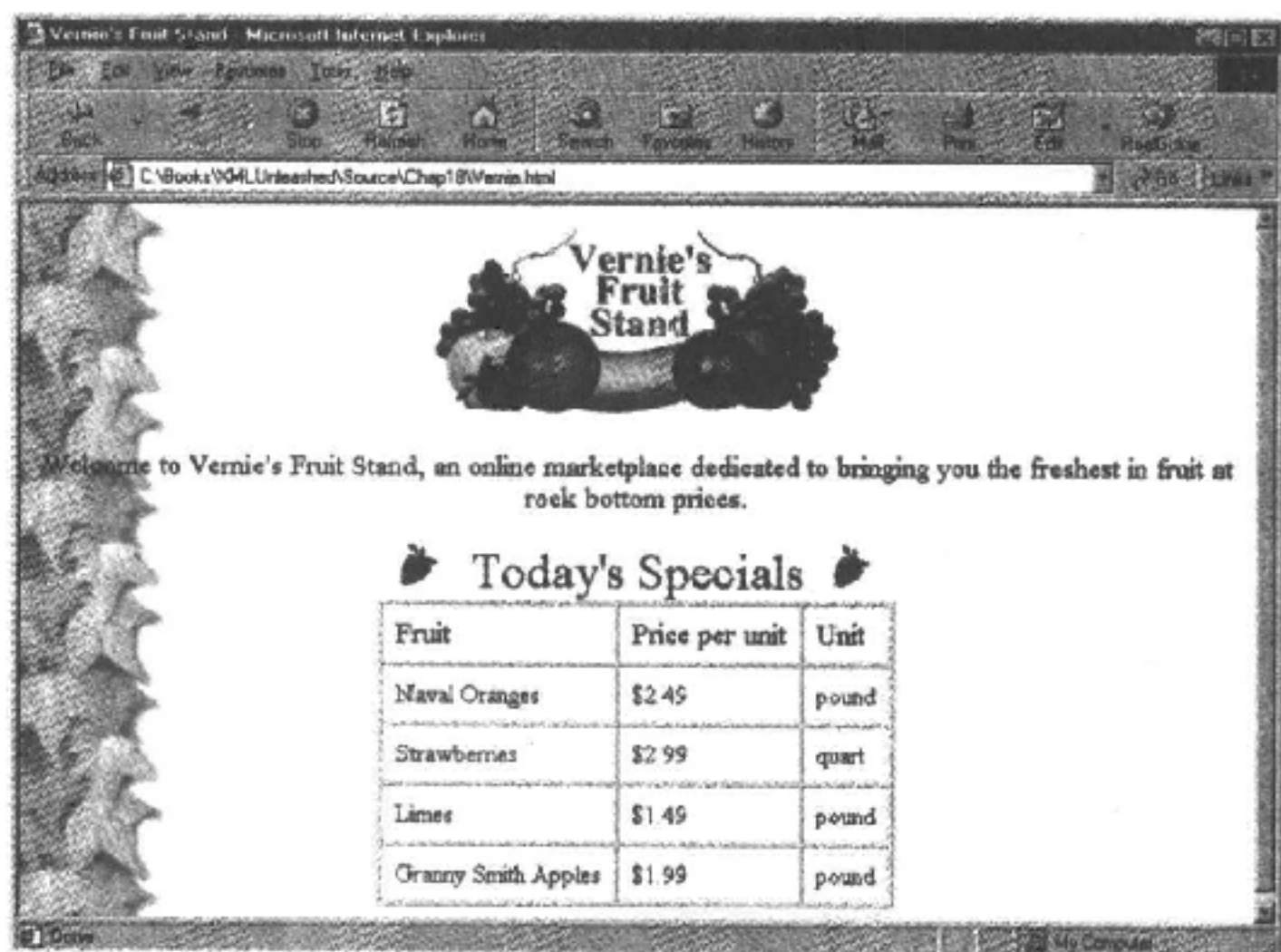


图 18.4 Vernie 的 Fruit Stand Web 页面在 Internet Explorer 5.0 中的显示

这个 Web 页面的代码相对规整,但绝对不是一个有效的文件,根据 XHTML 的规则它甚至不是结构良好的。下面是为了符合 XHTML 规则,这段代码需要解决的主要问题:

- 文件类型声明没有引用 XHTML DTD 之一。
- 没有声明 XHTML 名字空间。
- 元素名和属性名全都大写。
- 不是所有的开始标记都有相应的结束标记。

- 空元素(IMG)没有以/>结束。
- 不是所有属性值都用括号括起来。
- 有些元素(IMG)缺少必要的属性(ALT)。

HTML 文件的这些问题恰恰和前面提到的转换过程一致,这绝对不仅仅是一个巧合。我是通过对 HTML 文件执行转化才得到这个清单的。解决了所有这些问题的 XHTML 文件如清单 18.4 所示。

清单 18.4 Vernie 的 Fruit Stand Web 页面手工编码的 XHTML(Xvernie.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
http://www.w3.org/TR/xhtml1/DTD/transitional.dtd>

<html xmlns="http://www.w3.org/TR/xhtml1/DTD/transitional">
  <head>
    <title>Vernie's Fruit Stand</title>
  </head>
  <body background="Ivy.gif" bgcolor="#FFFFFF">
    <p align="Center">
      
    </p>
    <p align="center">=
    <font size="4">Welcome to Vernie's Fruit Stand,an online marketplace
      dedicated to bringing you the freshest in fruit at rock bottom
      prices. </font>
    </p>
    <p align="center">
      
      <font size="6">Today's Specials</font>
      
    </p>
    <div align="center">
      <table border="1" cellpadding="8" cellspacing="1">
        <tr>
          <td><font size="4">Fruit</font></td>
          <td><font size="4">Price per unit</font></td>
          <td><font size="4">Unit</font></td>
        </tr>
        <tr>
          <td>Navel Oranges</td>
          <td>$ 2.49</td>
          <td>pound</td>
        </tr>
        <tr>
          <td>Strawberries</td>
          <td>$ 2.99</td>
          <td>quart</td>
        </tr>
        <tr>
          <td>Limes</td>
          <td>$ 1.49</td>
```

```

        <td>pound</td>
      </tr>
      <tr>
        <td>Granny Smith Apples</td>
        <td>$ 1.99</td>
        <td>pound</td>
      </tr>
    </table>
  </div>
</body>
</html>

```

如果你仔细研究这个文件,你会发现它符合了所有有效的 XHTML 文件的要求。为了确认没有遗漏,我用 W3C 的验证服务来验证了这个文件。它成功地进行了验证,这说明这个文件的确是一个 XHTML 文件。

18.5.2 使用 HTML 整理工具

如果你不认为手工将 HTML 转化为 XHTML 是一件令人愉快的工作,那么你可以考虑使用转换工具。目前已经出现了一些这样的工具,它们相当成功地完成 HTML 到 XHTML 的转换。这个工具被称为 HTML Tidy,它是由一位惠普 UK 实验室的工程师 Dave Raggett 开发的。HTML Tidy 是一个命令行工具,本来它是为了整理不规整的 HTML 代码编写的,不过它也可以用来将 HTML 转化为 XHTML。仔细想一想,其实 HTML 到 XHTML 的转化只不过是整理了一下不规整的代码而已,这正是 HTML Tidy 工作十分出色的原因。

注意:你可以通过访问 HTML Tidy 的网站 <http://www.w3.org/People/Raggett/tidy> 来免费地下载 HTML Tidy 这个工具。另外还有一个叫做 HTML-Kit 的图形 HTML 开发工具和 HTML Tidy 一起工作。你可以通过访问 HTML-Kit 的网站 <http://www.chami.com/html-kit> 来免费地下载这个工具。

缺省地,HTML Tidy 读入 HTML 文件然后在标准输出中打印整理后的文件。你必须使用一个工具选项来指定你想要输出的 XHTML 格式。-asxml 选项指出 HTML Tidy 要把一个 HTML 文件整理为 XHTML。下面是在 MS-DOS 中用 HTML Tidy 将 HTML 文件 Vernie.html 转换为 XHTML 的命令:

```
tidy -asxml Vernie.html > Tvernie.html
```

请注意 tidy 应用程序的输出被重定向到文件 Tvernie.html。这就是将取得的 XHTML 文件保存在文件中的办法。清单 18.5 包含了 HTML Tidy 生成的 XHTML 文件的代码。

清单 18.5 用 HTML Tidy 生成的 Fruit Stand 主页的 XHTML 版(Tvernie.html)

```

<? xml version="1.0"? >
<! COTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  http://www.w3.org/TR/xhtml1/DTD/transitional.dtd>
<html xmlns=http://www.w3.org/TR/xhtml1>

```

```

<head>
  <title>ernie's Fruit Stand</title>
</head>
<body background="ivy.gif" bgcolor="#FFFFFF">
  <p align="CENTER">
    
  </p>
  <p align="CENTER">
    <font size="4">Welcome to Vernie's Fruit Stand,an online marketplace
      dedicated to bringing you the freshest in fruit at rock bottom
      prices.</font>
  </p>
  <p align="CENTER">
    
    <font size="6">Today's Specials</font>
    
  </p>
  <div align="CENTER">
    <table border="1" cellpadding="8" cellspacing="1">
      <tr>
        <td><font size="4">Fruit</font></td>
        <td><font size="4">Price per unit</font></td>
        <td><font size="4">Unit</font></td>
      </tr>
      <tr>
        <td>Navel Oranges</td>
        <td>$ 2. 49</td>
        <td>pound</td>
      </tr>
      <tr>
        <td>Strawberries</td>
        <td>$ 2. 99</td>
        <td>quart</td>
      </tr>
      <tr>
        <td>Limes</td>
        <td>$ 1. 49</td>
        <td>pound</td>
      </tr>
      <tr>
        <td>Granny Smith Apples</td>
        <td>$ 1. 99</td>
        <td>pound</td>
      </tr>
    </table>
  </div>
</body>
</html>

```

这段代码和手工转换 HTML 得到的代码惊人的一致。说良心话,我手工把生成的文件

修改为缩行的形式以便大家阅读。然而除了缩行之外,所有这些代码都是 HTML Tidy 生成的。请注意,文件类型声明和名字空间的声明都被给出,更不要说所有的那些元素名和属性名了,它们都被转换为小写。空元素 `img` 被修改为以 `/>` 结尾。另外所有的属性值都被用括号括了起来。

生成的 XHTML 文件还有两个地方需要人工修改。其一是 `align` 属性的 `CENTER` 值仍然要用大写。XHTML DTD 将这个值定义为小写。文件的另外一个问题是最后两个 `img` 元素没有提供 `alt` 属性,而这是 XHTML DTD 所要求的。HTML Tidy 还输出了这个问题的警告,但是要求 HTML Tidy 自己来产生 `alt` 属性值实在有点勉为其难。

在手工修改之后,生成的 XHTML 文件就可以使用。正如你所看到的,HTML Tidy 明显地改进了 HTML 到 XHTML 的转换工作,尽管你还是不得不手工进行一些修改。

18.6 总 结

在 Web 开发人员中有一种共识,尽管 Web 发展到今天,HTML 起了关键性的作用,但是一定要为未来考虑考虑。特别的,需要某种长期的解决方案来解决 HTML 文件缺乏结构的问题。它造成了低劣的代码,浏览器的宽容以及浏览器各自独特的扩展。HTML 发展的如此迅速,以至于人们没有时间考虑 HTML 是否适用于它正在解决的问题。即使现在,依然如此。W3C 一直在认真地考虑这个问题,并且提出了一种解决方案,它将极大地改变 HTML 的未来。无疑,XML 在 W3C 对 HTML 的解决方案中举足轻重。

XHTML 是 W3C 对 Web 共同体的长期需要的一个回答。在 1999 年 8 月,W3C 发布了 XHTML 建议的最终版本,本质上它将允许浏览器厂商更进一步,在主流浏览器上支持 XHTML。时间会告诉我们,XHTML 是否最终取代 HTML,但是对于我们这些计划在未来使 Web 更加结构化的人来说,它的确作出了很多承诺。

第19章 使用 XLink 和 XPointer 链接文件

你可能知道,链接是 Web 最具意义的特征,它使 Web 得以实现。没有把网页链接在一起的能力,那么 Web 就与存在多年的许多独立的公告牌系统没有什么区别。

尽管链接对于 Web 的发展和使用已经起到了如此重要的作用,但它目前还依然处于十分初级的阶段。XML 引导除了具有更为强大功能的链接机制,它在如何在 Web 上连接信息方面也有着戏剧性的影响。

本章将讨论 XLink 和 XPointer 这两个密切相关的 W3C 规范,它描述了用不同的方法链接 XML 内容的功能惊人的机制。XLink 和 XPointer 使得有可能用 HTML 建立令 Web 开发人员难以想像的链接。这听起来好像言过其实,其实不然。XLink 和 XPointer 是非常重要的 XML 技术,在不远的将来他们将对网页的结构和连结产生巨大的影响。本章将讲述他们为什么是如此重要的技术,并举例说明他们的用法。目前,XLink、XPointer 以及相关的 XPath 规范都处于 W3C 工作草案阶段。

19.1 链接 HTML 的方法

为了解 XLink 和 XPointer 的重要性,了解 HTML 所支持的链接的局限性是很重要的。HTML 链接,又称为超级链接,总是基于两种共享资源:源和目标。HTML 链接的源表现为指出它与另一资源链接。文本链接用下划线显示,当用户在已链接的资源(文本或图像)上活动鼠标时,鼠标会改变。遍历 HTML 的一个链接通常是点击源资源,这导致 Web 浏览器定位到目标资源。这一定位可能出现在同一浏览器窗口中——在这种情况下,目标资源取代当前页——或者可能出现在一个新的浏览器窗口中。

要理解的关键是 HTML 链接包括链接在单一方向的两个共享资源。换句话说,链接的一边总是源,而另一边总是目标。你也许认为 Web 浏览器的 Back 按钮将 HTML 链接限定为双向的,但是那只是浏览器的特征,它保存了网页目录以便用户可以返回。HTML 链接内不具备支持从链接对象返回到源的功能。

说到浏览器的特征,值得注意的是就链接方面我们期待的许多常规与 HTML 并没有直接关系。例如,HTML 本身对如何向用户显示没做任何规定(色彩、下划线等等)。这要由专用浏览器以及用户的选择来决定如何显示链接。虽然现在这看来还不是很重要,但是当浏览器开始支持 XLink 时,浏览器显示链接的作用就会变得重要起来。要知道,XLink 支持多资源与多方向间的链接,这实际上不可能用简单的下划线或鼠标来显示。了解浏览器厂商如何支持 XLink 上一些有趣的链接,将是很有意思的,不过,现在做这方面的讨论还有一点早。

回到 HTML 链接的讨论上来,你大概知道 HTML 中用于建立链接的元素是 a 元素,也称为锚元素。锚元素运用 href 属性标识 HTML 链接的目标资源,href 属性可引用一个完整的 URL 或一个相对的 URL。在后一种情况下,资源被假定定位在当前 URL 路径下,或者在当前文件内的文件段落下。HTML 链接可以链接到整个文件也可以链接到文件段落。当你

在链接到文件段落时,href 属性使用符号(#)接在 URL 后,然后是段落标记。

段落标记用来识别文件的一部分。用锚元素和 name 属性把段落标记和文件的一部分联系起来。这一属性值是链接源的锚元素中符号#右边的名称。下面是 HTML 链接的一个例子,它引用了一个被命名为 footnote 的文件段落:

```
<p>Click <a href="Chapter03.html#footnote">here</a>to view the  
footnotes for this chapter.</p>
```

在这个例子中,footnote 文件段落从单词“here”被链接。实际上 footnote 段落在文件中用锚元素和 name 属性来标识。

```
<a name="footnote">This is a footnote for Chapter 3.</a>
```

我想你对 HTML 锚元素也许十分熟悉了,但是这样简洁的回顾会有助于你理解 XML 链接方法的基本理论,这远远超出锚元素的局限。

19.2 超越传统 HTML 链接

仅仅基于 Web 的功能和用途,很难对 HTML 链接文件的简化了的方法提出异议。然而有许多方面可以改进,有一些你可能从未考虑过。

其一,如果链接是双向的,那么要回到前一资源就不必依赖浏览器的 Back 按钮,那有多好。尽管这看来很平常,但却是极其有用的,双向遍历链接,从而无需固定源资源和目标资源。双向链接把两个资源都看作是源和目标。

除了双向链接,引用多个目标资源的链接也是十分方便的。这使得 Web 开发人员不必为提供链接源这一单一目的而复制内容。具有多个目标的链接可以提供目标选择的弹出式菜单,供用户选择。这种链接的例子可以是关于 Amazon 公司的书籍清单。一本书的封面图像的多对象链接可以呈现为包含诸如概要、评论以及章节样本这样的文件的链接的弹出式菜单。通过减少为定位目的而使用的内容,压缩了网址用户接口。

对 HTML 链接另一个有趣的扩充是遍历链接的方法。除了 HTML 传统的选择,或用对象资源取代当前文件或打开拥有此资源的浏览器窗口,你还可以在当前文件内嵌入此资源。换句话说,对象资源的内容会取代源文件内的链接。用这种方法遍历链接会导致复合文件的汇编,这对网页来说是很有意思的功能。

HTML 链接是有局限性的,用户必须触发他们的遍历。例如调用网页上的链接,惟一的办法是点击已链接的文本或图像。你可能感到奇怪为什么还要希望使用其他的办法。那末,好吧,考虑一下这样一种情况,为了形成一个复合文件,已链接资源将被直接嵌入到文件中。当这个文件打开时,你也许想要这一嵌入立即发生,在这种情况下,用户就与正在被调用的链接没有任何关系。在这种意义上,链接就作为复合 Web 文件的各组件的一种连接组织,这远远超出 HTML 中链接的功用。

这种复合 Web 文件的一个例子是由几个其他的文件组成的网页。在 Microsoft FrontPage 中,你可以使用专用的 WebBot 对象把一页作为另一页的一部分。运用这种专用的部件,你可以创建由其他网页组成的复合网页。这对于在某一网址内所有含有共同的页脚的网页特别有用。当然这种功能可以在 FrontPage 中获得,因为 Microsoft 提供了专用的对

象做这一切工作。传统的 HTML 链接不提供由其他文件建立文件的通路链接。

我一直在描述 XLink,即 XML 文件链接机制的特征。我希望你开始意识到 XML 链接比 HTML 链接要抽象的多,事实上,它不只是为用户提供从一页移动到下一页的方法。XLink 完全是为了改变 Web 上资源间的基本连接。显然,为了充分理解 XML 的全部,你有必要回过头从更抽象的意义上考虑一下链接。好消息是一旦了解了它的重要性,你就会对 Web 有完全不同的看法。

注意:链接在 XML 1.0 规范中并没有提到。事实上完全没有提到。尽管链接总是被看作是 XML 方案中重要的部件,但它直到 XML 1.0 初步规范出台后才被正式定义。目前的 XML 链接规范,即 XLink,还依然出于 W3C 工作草案阶段。

为使你了解 XLink 技术的前瞻性,可以想想 XLink 支持创建它们所链接文件外面的链接。换句话说,在一个文件中你可以创建一个与其他文件中的两个资源相连接的链接。当你不能编辑其他文件时,这一点特别有用。创建这种非线性链接的功能可能可以创建一个全新的 Web 商业应用:链接库。这是一个链接数据库,它描述了 Web 上资源间有用的连接。这种数据库的例子可能是一个错综复杂的相互参照的法律数据库;法庭审理的案件可以用这样的方式进行链接,这样法律工作者就能够迅速地找到并核实已审判的案例,并处理类似的案子。

外部链接的另一个巨大的益处是与所链接的文件分离时链接依然保持。把链接与链接资源分开可以极大地减少链接中断,这可能是复杂的大型网址上所要解决的难题。事实上,追踪中断的链接一直是 Web 上的一个主要问题,这可能通过外部链接的方法得以改善。

要执行更为有趣的功能,HTML 链接能够得以改善的最后一个方法是使用可扩展链接来创建网环。网环是一系列按题目分类的相关网址,在其间你可以往返移动。通过锚元素来支持下一个/前一个链接机制,这主要是个人网址所做的事。外部链接会允许你建立独立于网址的网环,因为链接常驻在各自的文件。

19.3 W3C 链接规范

既然我已经讲述了 XLink 先进的链接特征所赋予的一些优点,现在该详细地看一下 XLink 的结构。与许多其他的 XML 技术一样,XLink 是 W3C 监测下的正式规范。目前,XLink 依然处于工作草案发展阶段,这就是说在 W3C 推荐标准出台前,它可能有改动。XLink 实际上包含三项技术,每项技术符合不同的规范:

- XPath——用于处理 XML 文件的语言。
- XPointer——用于处理 XML 文件内部结构的语言(依据 XPath)。
- XLink——一种 XML 语言,它指定 XML 文件中先进的链接结构(依赖于 XPointer 作为文件的链接部分)。

这其中可能有些混乱,因为 XLink 不仅是用于这项技术的链接语言的名称,而且是全部 XML 链接技术的名称。幸运的是,已知讨论的内容通常会使你明了“XLink”是用来作为整体上指 XLink 技术还是指 XLink 语言。

注意: XLink 原来是指 XLL(XML 链接语言),它是 W3C 设想的 XML 技术家族中的第二个部件。第一个部件是 XML 语言本身,而第三个部件是 XSL 样式表语言。

XLink 技术原来只包含两个部件,XPointer 和 XLink。然而,W3C 意识到 XPointer 不仅仅是需要处理文件方法的 XML 技术。XSL 变换技术也需要处理文件的方法,因此决定将文件寻址纳入 XPath 规范。依据 XPath 创建的 XPointer 为寻址进入文件内部结构提供了支持。XLink 运用 XPointer 来描述在 XML 文件内与专用结构的链接。下面几部分将更为详尽地探讨 W3C 有关这些技术的规范。要记住这些规范在成为推荐标准之前可能有某种程度的改变。

19.3.1 XPath

XPath 是用于处理 XML 文件的非 XML 语言。它是非 XML 语言,因为虽然它用来处理 XML 文件,但是它不用 XML 结构来描述,如元素和属性。XPath 提供了处理 XML 文件的抽象方法,构成了 XPointer 和 XSLT 上文件寻址的基础。XPath 使用的语法是为在 URI 上的使用和 XML 属性值而设计的,这意味着它必须十分简洁。这就是 XPath 不作为 XML 词表来执行的原因。名称 XPath 主要是根据使用路径表示法来处理文件这一概念。

为提供处理 XML 文件的方法,XPath 把文件看作是一棵节点树,这是从逻辑上认识 XML 文件的十分常见的方法。一个 XML 文件内的节点可分为元素节点,属性节点和文本节点。一些节点有名称,它可以含有一个可选名字空间 URI 和一个本地名:包括一个名字空间,其名称是可扩展名称。下面是可能出现在 XPath 上的不同类型的节点:

- 根节点
- 元素节点
- 文本节点
- 属性节点
- 名字空间节点
- 结构处理节点
- 注释节点

你应该十分熟悉这些节点了,因为它们代表用于 XML 汇编文件的主要结构。总有一个单一的根节点作为 XPath 树的根,以它的第一节点出现。文件中每一个元素都有一个相应的元素节点出现在树的根节点下。在元素节点内是所有的其他类型的节点,对应于元素的内容。元素节点可能有与之相联系的惟一标记,这在用 XPath 引用节点时很有用。

XPath 背后的大前提是遍历 XML 文件,以到达已知节点。运用表达式完成这一遍历。XPath 规范描述了如何运用 XPath 语法来建立表达式。当对表达式求值时,结果是下面几种类型的数据对象之一:

- 节点集合——类节点
- 布尔值——真/伪值
- 数——浮点数
- 字符串——文本串

XPath 表达式的求值方法完全取决于表达式的内容,这并不由 XPath 来确定。换句话说,这是 XPath 的抽象部分,允许与 XPointer 和 XSLT 共同处理文件。XPath 表达式的内容由 XPointer 或 XSLT 来确定,这反过来又决定了表达式如何求值。

关于 XPath 规范还有许多内容,虽然对于了解如何使用 XPointer 和 XLink,并不是所有这些内容都是重要的。然而,如果你渴望更多的了解 XPath,请到 <http://www.w3.org/TR/xpath> 查阅其规范。

19.3.2 XPointer

XPointer 是非 XML 语言,它基于 XPath,用来处理 XML 文件中的内部结构。它支持简洁的语法,使你可以横穿 XML 文件的逻辑树结构到参考元素、属性和文本串。XPointer 是 XLink 重要的一部分,因为它确定用于创建片段标记的语法,它代表文件内部结构的地址。对于访问 XML 全部文件的链接,与内部文件部分不同是,XPointer 是不必要的。

注意: W3C XPointer 工作草案的网址是 <http://www.w3.org/TR/WD-xptr>。

查看一下 W3C 初始的设计对象有助于了解 XPointer 在 XLink 中的作用。下列是摘自 XPointer 规范的设计参数,它清楚地揭示了 W3C 在创建 XPointer 时使用的准则:

1. XPointer 应当在 XML 文件中处理。
2. Xpoointer 应直接用在 Internet 上。
3. XPointer 可以直接用在 URL 上。
4. XPointer 设计应迅速准备好。
5. XPointer 设计应是正式的,简明的。
6. XPointer 语法应是合理的简洁,具有人可读性。
7. XPointer 应尽可能完善其使用。
8. XPointer 必须适宜于执行。

这些设计参数都是直截了当的,合乎逻辑的,因为 XPointer 是全部 XML 技术的一部分,它本身遵循了一套相似的设计参数。注意“正式的”、“简明的”、“简洁的”这些字眼都是用来描述 XPointer 的。这一点很重要,因为 XPointer 被设计用在相对紧凑的地方,如属性值。在本章后面,当你学习如何运用 XPointer 处理文件时,这些 XPointer 设计参数的基本原理就会更清楚了。

19.3.3 XLink

XLink 是 XML 词表,它使 XML 元语言具有高级链接功能。XLink 被设计用来不仅支持那些提供有趣的新方法链接文件的各种不同的可扩展链接,而且支持简单的单向 HTML 式链接。使用 XLink,你可以由第三个文件链接两个文件,实质上这使得链接本身成为有活力的资源。你还可以指定遍历链接的方式。例如,你可以创建一个链接,文件一经载入,就可以自动遍历此文件。这也许有些奇怪,但你想一想已链接文件可以在当前文件的内容内被打开,从而形成一个复合文件的情况,你就会明白了。这些例子只是 XLink 扩展网上链接的概念和功能的几个方面。XLink 是作为 XML 词表来执行的,这就是说它可以很容易的并入 XML 应用。XML 开发人员应该把 XLink 看作是 XML 技术的另一个方面,而不是某些专用

的东西。

注意: W3C XLink 工作草案的网址是 <http://www.w3.org/TR/xlink>。

与创建 XPointer 的开发进程相似, W3C 概括的论述了一系列用作创建 XLink 基础的设计参数。下列是摘自 XLink 规范的设计参数, 它揭示了 W3C 在创建 XLink 时所遵循的准则:

1. XLink 必须能直接用在 Internet 上。
2. XLink 必须适用于广泛的各种各样的链接使用域和各种类别的链接应用软件。
3. XLink 必须支持 HTML4.0 链接结构。
4. XLink 表达式语言必须是 XML。
5. XLink 设计必须是正式的, 简洁的并能说明性质的。
6. XLink 必须是人类可读的, 人类可写的。
7. XLink 可以驻留在共享资源驻留的文件的内部或外部。
8. XLink 必须表现链接的抽象结构和重要性。
9. 不论用来表示链接的介质是什么, XLink 必须适于执行。
10. 已建立的超介质系统和标准的知识必须使 XLink 充满活力。

这些设计参数使我们对 XLink 略见一斑。首先, 显而易见的是, XLink 将与 HTML4.0 中的链接结构相容。这一点对于 W3C 很重要, 因为用户不可能接受一个不向后兼容 HTML 而且不基于 XML 的 XML 链接机制。请记住 XML 共同体已经在向基于 XML 的模式语言发展 (XML 模式), 这与 DTD 使用的非 XML 语言不同。如果 XML 文件中的链接机制不是基于 XML, 那么, 就可能出现类似的问题。

设计参数的另一个重要方面是链接可以驻留在共享资源驻留的文件的内或外。这与 HTML 链接完全不同, 在 HTML 链接中, 源链接资源总是与链接在同一个文件。同时, 注意不论用来表示链接的介质是什么, 设计参数要求 XLink 适于执行。这种执行的可行性是 XLink 的一个重要部分, 因为一个复杂的, 难于执行的技术需要相当时间才能得到浏览器和相关厂商的支持。一项功能强大的技术可能仅仅因为难以执行而被弃置不用。这是 SGML 首先被简化为 XML 的原因之一; SGML 要求掌握 Web 技术, 简直太难以执行了。

19.4 深入 XLink

在 XLink 中用于确定链接的基本结构是链接元素, 它声明链接的存在并描述其特征。HTML 中的锚元素就是链接元素的一个例子。虽然链接元素构成了 XLink 的基础, 但是在 XLink 语言中并没有预定义的链接元素。换句话说, 在 XLink 语言中 你不会找到一个锚元素, 或者类似的东西。这是因为 XML 的整个前提是创建专用标记集, 这避免了使用固定的链接元素。换句话说, 你可能想要针对已知的 XML 词表定义你自己的链接元素, 而不是被锁定进一个固定的元素, 例如 HTML 的锚元素。

为了避免出现使用 HTML 路径进入固定的链接元素这一问题, XLink 定义了可以链接任何元素的标准链接属性。一个链接元素使用一个叫做定位器的结构来链接一个链接中的资源。在 HTML 和 XML 中, href 属性作为链接的定位器。虽然 HTML 和 XML 共有这一

属性,但是 XML 中的链接比 HTML 相应部分描述的要详尽得多。也许最重要的不同在于 XML 链接描述了其中的全部资源,即使对象资源是文件片段。在 HTML 中,必须把锚元素置于对象片段资源并用 name 属性来识别。在 XML 中不是这种情况,因为 XLink 提供了必要的充分描述链接内在资源的内容。

19.4.1 链接的类别

有两种 XLink 支持的链接元素:

- 在线链接
- 外部链接

在线链接的内容作为链接共享资源之一。典型的在线链接有一个链接元素,含有作为链接源的内容。HTML 锚链接是在线链接的例子,因为它直接包含作为链接源的文本。

外部链接的内容不作为链接共享资源之一。这就是说外部链接是独立于共享资源之外的,因此其目的与在线链接的目的完全不同。外部链接可用于链接只读文件中的信息,并将为网上使用链接开创新的有趣的机会。更为奇特的是,有可能开创链接数据库,来描述网上传播信息之间的关系。

搜索引擎很可能发展成为外部链接数据库。一个搜索引擎已经是一个链接数据库,但是它只用来为在线的单向链接增加内容。换句话说,一个搜索引擎使链接返回到个人页面,在此你可以遍历找到你在寻找的信息。使用 XLink,搜索引擎能够储存那些描述页面间复杂关系的链接。链接本身并不是资源的一部分,因此,可以独立的被处理和更新。

外部链接是 XML 可扩展链接的主要元素。可扩展链接是任何一种扩展 HTML 链接功能的链接,显然外部链接被认为是可扩展链接,因为 HTML 不支持任何一种类似的链接机制。可扩展链接还支持把多个对象资源与已知链接相连接的想法。你可以为一个网址建立一个目录,在这个网址中只有指向网址中各个页面的可扩展链接。如果这些链接收集在一个与目录页分离的文件内,那么这些链接也被认为是外部链接。

19.4.2 XLink 属性

回到前面讨论的有关 XLink 是如何被执行的,我曾提到 XLink 定义了用来描述链接的标准属性。下面是 XLink 定义的属性,用来创建已链接元素:

- type——一个决定链接类型的预定义字符串。
- href ——使用 URI 处理对象资源的定位器。
- from——在描述 arc 时确定链接来源的资源的字符串。
- to——在描述 arc 时确定链接去向的资源的字符串。
- show——确定对象资源如何展示给用户的预定义字符串。
- actuate——确定链接如何启动的预定义字符串。
- role——用于描述链接内容的功能的指定应用字符串。
- title——用作链接名称的字符串。

type 属性决定链接的类型,可以具有下列任一个值:simple、extended、locator、arc 和 group。type 属性的 Group 值用来建立可扩展链接组,这是组合成为一个链接结构的可扩展

链接组合。关于 href 属性,通过在 HTML 上的使用,你肯定已经熟悉它了。from 和 to 属性用来描述 arc,它是定义链接遍历行为的逻辑结构。arc 不仅定义链接去向何方,而且定义它来自何方。arc 可以用来建立网环,其中用 from 和 to 属性链接一系列网页。

show 属性决定链接的对象资源如何展示给用户。对于 show 属性有三个可能值:

- replace——对象资源取代当前文件。
- new——对象资源显示在一个新的窗口中。
- parsed——对象资源取代链接插入当前文件。

actuate 属性决定链接如何启动并具有下列其中一值:

- user——链接必须由用户手动遍历。
- auto——源文件载入时,自动遍历链接。

role 和 title 属性主要用于描述目的。role 属性描述链接中内容的作用,而 title 则提供可以在浏览器上展示的人类可读的链接标题。

19.4.3 创建 XLink

因为 XLink 并不为建立链接定义任何固定元素,所以要创建资源间的链接,你必须把 XLink 属性与你自己的元素联系起来。这种把链接合成为用户的 XML 词表的方法令人难以置信的无止境,这正符合 XML 总的前提。下面是用来把一个文本与一个包含某个运动队队员名单的 XML 文件链接起来的简单 XLink 代码:

```
<! ELEMENT players ANY>
<! ATTLIST players
  xmlns:xlink CDATA #FIXED "http://www.w3.org/TR/xlink"
  xlink:type(simple|extended|locator|arc) #FIXED "simple"
  xlink:href CDATA #REQUIRED
  xlink:role CDATA #IMPLIED
  xlink:title CDATA #IMPLIED
  xlink:show(new|parsed|replace) "replace"
  xlink:actuate(user|auto) "user">
```

这个链接使用了前面你所了解的大部分 XLink 标准属性。它也简单涉及了我将要谈到的问题:XLink 名字空间。在 XLink 名字空间定义了标准的 XLink 属性,这就是说要使属性易于参考,你必须有名字空间说明。

除了 xlink 名字空间前缀,还有几个属性有必要更为详细的关注。type 属性有固定值“simple”,它表明这是一个简单链接。show 属性有缺省值“replace”,它表明对象资源是在遍历此链接时取代当前文件。最后,actuate 属性有缺省值“user”,它表明链接必须由用户激活,以使遍历发生。

下面是这个链接如何出现在 XML 文件中的例子:

```
<players xmlns:xlink=http://www.w3.org/TR/xlink
  xlink:href="Players.xml" xlink:role="player roster"
  xlink:title="Player Roster" xlink:show="new" xlink:actuate="user">
```

```
Roster of Current Players
</player>
```

这里相当直接的使用 players 链接元素。然而,它还可以简化如下:

```
<players xmlns:xlink=http://www.w3.org/TR/xlink
  xlink:href="Players.xml" >
  Roster of Current Players
</player>
```

尽管简单链接十分重要,但是在 XML 中并没有什么新的东西。可扩展链接实际上代表了由 XLink 提供的附加链接功能。下面是链接元素 players 如何作为可扩展链接来执行的例子:

```
<! ELEMENT players ((xlink:arc|xlink:locator)*) >
<! ATTLIST players
  xmlns:xlink CDATA #FIXED "http://www.w3.org/TR/xlink"
  xlink:type(simple|extended|locator|arc) #FIXED "extended"
  xlink:role CDATA #IMPLIED
  xlink:title CDATA #IMPLIED
  xlink:showdefault(new|parsed|replace) #IMPLIED
  xlink:actuatedefault (user|auto) #IMPLIED>
```

这一链接与前一个的主要区别在于 type 属性被置于“extended”以表明这是可扩展链接。可扩展链接也不同于简单链接,它们需要定位元素来代表实际的对象资源链接。下面是与 players 可扩展链接一起使用的定位器的例子:

```
<! ELEMENT players ANY>
<! ATTLIST player
  xmlns:xlink CDATA #FIXED "http://www.w3.org/TR/xlink"
  xlink:type(locator) #FIXED "locator"
  id ID #REQUIRED
  xlink:href CDATA #REQUIRED
  xlink:role CDATA #IMPLIED
  xlink:title CDATA #IMPLIED>
```

注意 player 定位器包含 href 属性,它表明此元素实际链接到对象资源。下面是一个在 XML 文件中如何同时使用 players 可扩展链接和 player 定位器的例子:

```
<players xmlns:xlink="http://www.w3.org/TR/xlink"
  xlink:type="extended" xlink:role="player roster"
  xlink:title="Player Roster" xlink:show default="replace" xlink:actuate default="user" >
  <player xlink:href="Player1.xml">Keith Nash</player>
  <player xlink:href="Player2.xml">Josh Timm</player>
  <player xlink:href="Player3.xml">Travis Foster</player>
  <player xlink:href="Player4.xml">Randy Hood</player>
  <player xlink:href="Player5.xml">Kevin Burns</player>
```

</players>

正如你所看到的,可扩展链接 players 作为 player 定位器的一个组。记住这只是结构可扩展链接的一种方式。由于 XLink 开始得到浏览器和 XML 开发工具的支持,你可能会看到可扩展链接以一些有趣的方式来连接 XML 资源。

19.5 深入 XPointer

XPointer 是 XLink 技术的组成部分,用来访问 XML 文件。XPointer 允许你通过构成 XML 文件的树的节点来处理特定的节点。XPointer 定义了用于描述段落标记的语法,反过来又用来指定文件。XPointer 对可寻址的 XML 文件提供了更细微的控制,这与 HTML 提供的通用方法大不相同。

在 HTML 中,你可以使用锚标记和 name 元素来访问一个文件内的通用点。XPointer 的寻址更为具体,允许你在已知类型的元素清单中定位具有已知值的元素。XPointer 的文件寻址比 HTML 提供了更多的控制,XML 文件严密的结构使其成为可能。

XPointer 是用于建立表达式的简洁语言。XPointer 表达式附加在 URI 的末尾,中间用符号(#)隔开。一个 XPointer 表达式由含有语法的位置路径组成,类似传统编程语言中的函数调用。

19.5.1 位置路径

位置路径是用来描述在 XML 树内节点路径的结构,它由一个节点或一组节点组成。位置路径串接在一起形成一个完整的 XPointer 表达式,从左向右求值,到达树内的绝对位置。XPointer 不仅提供了控制属性的专用位置路径,而且还提供了位置路径,可以遍历兄弟节点,祖先节点,子节点和子孙节点。位置路径分为两个基本类型:绝对路径和通用路径。

绝对位置路径指向 XML 树内的绝对位置。下面是 XPointer 定义的绝对位置路径:

- 根(/)——定位整个资源,它是 XML 树的父节点。
- id(Name)——定义具有 id 类型属性和一个与 Name 相配的值元素。
- here()——定位含有 XPointer 表达式的节点。
- origin()——定位子资源,过去用户由此开始遍历(与外部链接一起使用)。

最重要的绝对位置路径是根路径和 id() 路径。根路径表示为斜线(/),经常用在 XPointer 表达式的开始作为相对位置路径的基础。id() 位置路径用于定位一个具有特定属性值的元素。

注意:如果在 XPointer 表达式的开始没有指定绝对位置路径,那么根路径就被假定为组成后继相关位置路径的基础。

除了绝对位置路径,XPointer 还定义了一套相对位置路径。相对位置路径在已知的内容节点上进行操作,这一节点被假定为已经使用绝对位置路径建立,例如根路径。下面是在 XPointer 表达式中使用的相对位置路径。

- child——定位内容节点的子节点。

- descendant——定位内容节点内的内容节点。
- descendant-or-self——除了包括内容节点外,与 descendant 相同。
- parent——定位那些直接含有内容节点的元素节点。
- ancestor——定位那些含有内容节点的元素节点。
- ancestor-or-self——除了包括内容节点外,与 ancestor 相同。
- preceding-sibling——定位那些先于内容节点的兄弟节点。
- following-sibling——定位那些跟随内容节点的兄弟节点。
- preceding——定位那些先于整体内容节点的节点。
- following——定位那些跟随整体内容节点的节点。
- self——定位内容节点表内的专用内容节点。
- attribute——定位内容节点的属性。

如果感到这些位置路径令人困惑,不要担心。下一部分将从基础讲述如何运用位置路径建立 XPointer 表达式。

19.5.2 创建 XPointer

XPointer 是那些听起来十分复杂的技术之一,但是实际上它并不难使用。当然,XPointer 具有一定的灵活性,要花时间才能适应,但是自如地运用你自己的 XPointer 表达式并非难得高不可攀。只要看几个 XPointer 表达式的例子,就可以在了解 XPointer 是如何用来定义文件片段标记的情况下,自如的修改运用。下面是一个简单的 XPointer 表达式的例子:

```
Child: :factoid
```

这个例子使用 child 相对位置路径来定位内容节点的所有属于 factoid 类型的子节点。要记住这个 XPointer 表达式只描述了一个资源的片段标记。当用在内容时,可将这个片段标记与属于 href 属性的 URI 相配,如:

```
href=http://www.thetribune.com/factoids.xml#child: :factoid
```

在这个例子中,规定 URI 引用被命名为 factoids.xml 的 XML 文件。然后这个 XPointer 表达式被用做片段标记,与 URI 用符号(#)分开。这是 XPointer 使用的典型方法。

前面提到的各种位置路径可以用来创建更有趣的 XPointer 表达式。

```
Child: :factoid/following-sibling: :legend[position()=2]
```

这个例子首先定位了内容节点的那些属于 factoid 类型的所有子元素,然后,找到那些属于 legend 类型的元素节点下面的每一个兄弟节点。当然,这样使用 XPointer 看来有些奇特,不过要记住它是为 XML 文件的内部结构编址而设计的通用语言。我们几乎不能区分应用程序处理文件部分有怎样的不同,这也就是为什么 XPoint 为何如此的灵活。

你可能已经注意到前面例子中的 position() 函数。XPointer 继承了 XPath 的一些功能,运用 XPointer 表达式执行不同的任务。其中之一是节点测试,用来确定节点的类型。当然,你可以使用元素名称来核查节点是否属于某一元素类型,但是节点测试功能允许你核查节点是注释、文本,还是处理机指令。表 19.1 包含 XPath 上定义的节点测试功能,以及专为

XPointer 定义的几项功能。

表 19.1 在 XPath 和 XPointer 中定义的节点测试函数

| 函数 | 用法 |
|--------------------------|--------------------------|
| count() | 获得节点集中的节点数目 |
| position() | 获得节点清单中的当前节点的指针 |
| last() | 获得节点清单中前一个节点的指针 |
| id() | 获得符合所给标识的节点 |
| name() | 获得所给节点集合的第一个节点的展开名称 |
| local-name() | 获得所给节点集合的第一个节点的展开名称的本地部分 |
| namespace-uri() | 获得所给节点集合的第一个节点的名字空间 URI |
| comment() | 检查当前节点是不是注释 |
| text() | 检查当前节点是不是文本节点 |
| processing-instruction() | 检查当前节点是不是一条处理命令 |

下面是使用节点测试功能的一个例子。

```
/child::processing-instruction()
```

这一表达式为文件元素的兄弟节点处理指令定位。表达式能找到文件元素的兄弟节点的原因是根元素(/)是文件元素的父母,这就是说根的子节点是文件元素的兄弟节点。

注意: XPointer 还继承了 XPath 的串,布尔以及数码功能,用来控制它们各自的数据类型。

记住 XPointer 是一项新兴的 XML 技术,这就是说,在最终的 XPointer 问世之前,相关信息有可能发生变化。我鼓励你自己去开发 XPointer 并试验创建 XPointer 表达式。

19.6 总 结

正如你所知道的,Web 是大量的可链接文件的组合。没有链接,网上的文件就会是不相关联的信息的孤岛。显然,链接对于 Web 的成功起着重要的作用。令人惊讶的是,尽管 Web 的其他任何方面实际上都在经历着发展变化,但是却没有多少人对 Web 的链接能力表示置疑。XML 为我们提供了重新考虑 Web 链接的好机会,使其有可能发展,使文件连通性达到一个全新的水平。

第 20 章 编写 XML 脚本

正如你所了解的,XML 是一种标记含有结构数据文件的元语言。作为数据表示语言,XML 本身并没有多大作用。为了用 XML 做些有趣的事情,你必须在可操作技术的上下文中使用 XML,例如脚本语言。脚本语言为开发由 XML 数据驱动的基于 Web 的交互应用提供了便捷的方法。主要要求是脚本语言必须能够存取 XML 文件的逻辑结构,实质上就是 XML 树。

本章将向你介绍 XML 脚本,这是使用脚本语言存取和处理 XML 内容的过程。你将不仅了解如何使用脚本语言用 XML 数据做有趣的事情,而且会了解对脚本 XML 的主要选择。

20.1 为什么编写 XML 脚本

你已经知道一项启动技术对于 XML 在 Web 上的应用而言是非常必要的。这种启动可以以多种不同的方式出现:程序设计语言、脚本语言、样式表语言等等。这些 XML 启动方法都各有利弊。我想着重讲一下把脚本语言作为 XML 应用的启动技术,因为它或许是在网页上下文中存取 XML 数据的最简单方法。

成熟的程序设计语言,例如 Java,正在广泛的用于处理 XML 数据,如果你既会编程又能轻松的综合应用这些技术,那么效果是很不错的。特别是 Java,它是处理 XML 一种理想的程序设计语言,因为它相对简单,开销要求较小,而且具有平台独立性。另一方面,尽管 Java 比 C/C++ 简单,但是它毕竟是程序设计语言,因此,最适合那些具有程序设计知识的人。

在网页中使用 Java 的另一个问题是实际上它并没有直接对一个页面的文件对象模型(DOM)进行存取,这就是说,你不能使用 Java 来修改网页的结构。一般来说,Java applet 是独立的存在于网页中的一部分。因此,如果你计划把 XML 数据结合到网页的整个 Schema 中,那么你用 Java 就很难做到。要明白我决无诋毁 Java 之意;只是要说明它原本就不是为直接处理网页而设计的程序设计语言。然而,脚本语言可以做到这一点。

脚本语言与网页的结构紧密相连,因此可以直接存取网页的任何部分,将其作为可编程管理的对象。例如,你可以使用脚本语言来动态的修改一幅图像的 URL,这将导致标记的实际 src 属性发生改变。清单 20.1 展示了使用 JavaScript 编写的脚本代码,它自动的修改标记以创建一个滚动广告条网页。

清单 20.1 使用 JavaScript 实现的滚动广告条网页(AdBanner.html)

```
<! DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN? >
<html>
  <head>
```

```

<title>Ad Banner Example HTML Document</title>

<script language="JavaScript">
    var bannerNum = 1;

    function linkBanner() {
        window.alert("You just clicked Ad Banner " + bannerNum + ".");
    }

    function rotateBanner() {
        if (++bannerNum > 3)
            bannerNum = 1;
        document.images["banner"].src = "Banner" +
            bannerNum + ".gif";
        window.setTimeout('rotateBanner();', 2000);
    }
</script>
</head>

<body onload="window.setTimeout('rotateBanner();', 2000);">
    The banner advertisement image below is set to rotate every 3 seconds.
    You can click the banner to find out more about the advertiser. In a real
    usage scenario, clicking the banner would link to a Web page. You would
    also probably lengthen the rotation time since 3 seconds is pretty quick.
    <br>
    <p align="center">
        <a href="javascript:linkBanner();">
            
        </a>
    </p>
</body>
</html>

```

注意脚本是如何动态地修改 标记的 src 属性的。这个图像通过为它指定惟一的名称“ banner”来识别。然后,这个名称在 JavaScript 代码中被用作找到这个图像对象并修改其 src 属性的基础。这种修改事件每 3 秒钟发生一次,就产生了旋转的广告旗帜的效果。图 20.1 表示一个广告条图像在滚动广告条网页上的显示情况。

这个滚动广告条的例子说明 Web 脚本语言是与 HTML 密切相关的。脚本语言被设计用来处理简单文本,因为它是 HTML 代码的基础。简单文本也正好是 XML 的基础,这就是为什么脚本语言适合用来进行基于 Web 的 XML 文件处理。

除了与 Web 网页结构密切相关,脚本语言还有另一个有益之处。传统的程序设计语言,例如 Java 和 C/C++,要求你使用专用的编译器来编译程序。这对开发过程来说增加了步骤。大多数脚本语言并不涉及编译问题,这就是说,你可以在不附带任何步骤的情况下,编辑和检测这些语言。这使得开发过程对于那些没有太多时间或者并不是经验丰富的程序员的 Web 开发人员来说变得容易了许多。

与 XML 脚本语言有关的最后一个问题就要谈到 XSL。与 HTML 相应的技术 CSS 相比较,XSL 包含一些程序设计结构,使你可以用许多不同的方法来处理 XML 数据。XSL 这一标准的 XML 样式语言,结合了一些程序设计结构,这一事实表明需要程序设计处理。而脚本正是对 Web 网页增加程序设计控制的最直接、最简单的办法。

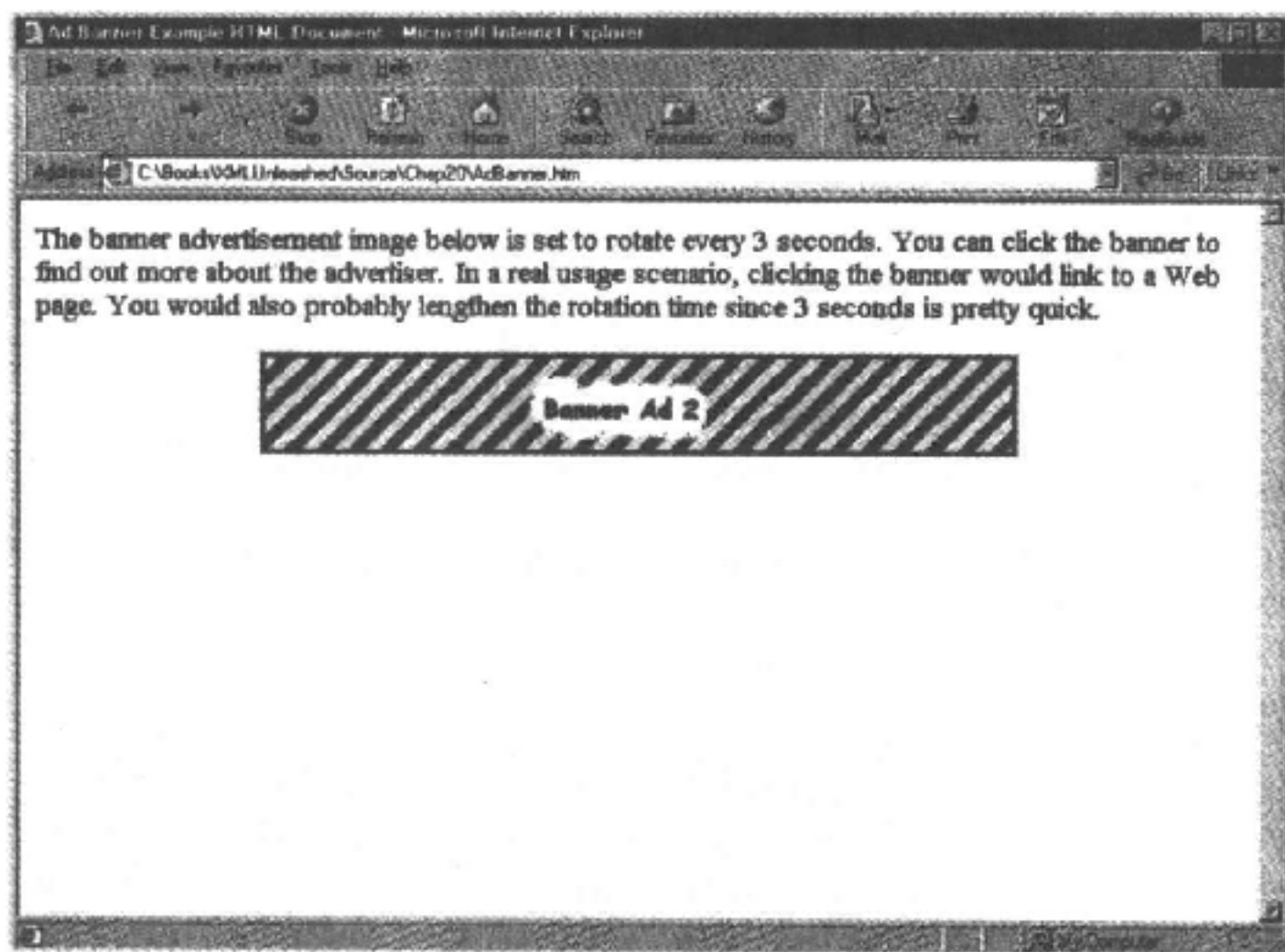


图 20.1 在 Internet Explorer 5.0 中显示滚动广告条网页

20.2 XML 脚本选择

作为一个追求 XML 脚本开发人员,你有多种脚本语言可供挑选。现在使用的大多数受欢迎的脚本语言都可用来处理 XML 数据。如果你是编写 Web 脚本的新手,了解脚本开发人员可能会钟情于哪一种脚本语言是很重要的。因此,我鼓励你试用几种脚本语言,然后决定哪一种语言最适合你的需要。公平地讲,各 Web 脚本语言完成了许多同样的事情,因此每两种语言间在功能上并没有太大的区别。

下面是当前几种支持 XML 的主要 Web 脚本语言:

- JavaScript
- VBScript
- Perl
- Python
- AppleScript
- Tcl

下面将详细论述这些脚本选项。

20.2.1 JavaScript

Netscape 创建 JavaScript 作为 Java 的脚本选择。JavaScript 的主要目的是为 Web 开发人员提供一种方法,使他们可以在 Web 网页上增加交互活动而不必建立大型的 Java applets。因此,JavaScript 编入了与 Java 相似的语法,但是它有比 Java 更为简单的结构,也更易于使

用。虽然 JavaScript 的语法与 Java 类似,但是两种语言实际上并没有直接的关系。Netscape 开发 JavaScript 作为脚本语言原本是希望 Java 的成功能给它带来好运。除了有相似的语法,Java 和 JavaScript 是完全不同的技术。

JavaScript 完全支持 Internet Explorer 5.0 下的 XML DOM 处理 XML 文件。要更多的了解 JavaScript 和 XML,你可以访问 Microsoft 的 XML 网址:<http://msdn.microsoft.com/xml>。你也可以继续阅读本章,因为在这一章中我将把 JavaScript 作为 XML 脚本举例的基础。我还将为你提供 JavaScript 快捷入门。

你也许听说过 JScript,它是 JavaScript 的 Microsoft 执行版本。尽管 JScript 与 JavaScript 在许多方面相似,但是它还包含了 Microsoft 的扩充及更大范围的对象模式,并综合了 Microsoft 的技术。有时你会看到 JScript 在执行 ECMA-262 语言规范。为了有助于使 JavaScript 和 JScript 标准化,European Computer Manufacturers Association (ECMA 欧洲计算机制造协会)根据 Microsoft 和 Netscape 的联合建议创建了脚本语言规范。ECMA 是这一规范的名称,如今是 JScript 和 JavaScript 的基础。

20.2.2 VBScript

VBScript 基于 Visual Basic 程序设计语言,它是由 Microsoft 创建的一种脚本语言。它本来是打算作为 Microsoft 对 JavaScript 的回应,因为 Microsoft 一直是 BASIC 程序设计语言的支持者。Microsoft 终于认识到并不是每个人都想要参加它的 BASIC 圣战,于是创建了 JScript,这是它自己的 JavaScript 执行版本。尽管 VBScript 和 JavaScript 明显地使用不同的语法和代码结构,但它们的功能相同。VBScript 完全支持 Internet Explorer 5.0 下的 XML DOM 处理 XML 文件。要更多地了解 VBScript(和 JScript),你可以访问 Microsoft 的脚本网址:<http://microsoft.com/scripting>。有关用 XML 使用 VBScript 的资料,可访问 Microsoft 的 XML 网址:<http://msdn.microsoft.com/xml>。

20.2.3 Perl

Perl 代表使用实用析取和报告语言 (Practical Extraction and Report Language),它是网上极受欢迎的一种脚本语言。Perl 语言的语法在许多方面与 C 程序设计语言相似。然而,Perl 是一种脚本语言,因此不涉及编译。在网上 Perl 的主要用途是创建 CGI (公共网关接口: Common Gateway Interface) 脚本,它是在网络服务器上运行的,并在回应浏览器要求时处理 HTML 代码的专用脚本。

XML 以 Perl XML 解析器的形式来支持 Perl,其中的几种现在正在使用。或许最受欢迎的是 James Clark 大众化的 Expat XML 解析器的 Perl 版本。这种 XML 解析器可在如下网址获得:<http://wwwx.netheaven.com/-coopercc/xmlparser/intro.html>。

20.2.4 Python

Python 是依据受欢迎的 Monty Python 的 Flying Circus BBC 戏剧系列而命名的一种脚本语言。然而,它是使网络开发人员享受到极大成功的一种严格的脚本语言。与 Perl 一样,Python 已被大量的应用作为开发 CGI 脚本的手段。因为 Python 擅长处理文本 (HTML 文

件),因此对于XML来说是合适的脚本语言。

有几种XML解析器在使用Python脚本处理XML数据。想要更多地了解Python和XML,请访问Python/XML网址:<http://www.python.org/topics/xml>。

20.2.5 AppleScript

AppleScript作为Apple Macintosh计算机标准的脚本语言,看来不可能被选择作为XML脚本。然而,AppleScript是一种功能很强的脚本语言,它具有文本处理能力,当然能够处理XML数据。况且,AppleScript被装入Macintosh操作系统,这使得它像脚本语言一样具有可访问性。

AppleScript以软件包的形式支持XML,Late Night Software把这个软件包叫做XML Tools AppleScript Scripting Addition。这个成套工具箱靠James Clark大众化的Expat XML解析器来分析XML文件。要想更多的了解XML Tools AppleScript Scripting Addition包,请访问XML Tools网址:<http://www.latenightsw.com/freeware/XMLTools>。

20.2.6 Tcl

Tcl(发音为tickle)代表工具命令语言(Tool Command Language),它是一种功能强大的脚本语言,适合于组织综合任务。它已成功地作为基于命令的脚本语言来控制交互应用。例如,你可以使用Tcl脚本,使向应用发出命令这一过程自动进行,如C/C++调试程序。Tcl也可以用来直接向操作系统shell发出命令。

注意:一般提到Tcl,与Tk的发音相同,Tk是图形用户接口(GUI)工具包,用来创建功能强大的GUI。Tk是Tcl的标准组件,这也是Tcl经常被看作是Tcl/Tk的原因。

现在,至少有一种支持XML的Tcl软件包在使用。这种开放源码软件包叫做TclXML,使用Clark的Expat XML解析器来分析XML文件。要了解TclXML,可访问TclXML网址:<http://www.zveno.com.zm.cgi/in-tclxml>。

20.3 JavaScript入门

因为本章基本上以JavaScript为例讲解脚本语言,所以我认为快速复习JavaScript语言的基础语法对我们的工作是有帮助的。如果你已经是JavaScript的行家里手,你可以略过这一部分。如果不是,那么紧张起来尽快入门吧。

JavaScript脚本作为简单的文本代码而创建,包括结构、语句和语法,与Java程序设计语言有些类似。JavaScript语言语法可分为以下几个主要部分。

20.3.1 语句

JavaScript语句以逐行翻译为基础。一个典型的语句包括JavaScript关键字、直接数据、变量和运算符。尽管另起一行就可以表明是一个新的语句,但是明确的用分号(;)来终止语句仍被认为是良好的JavaScript程序设计风格。下面是一个例子:

```
var myName = "Michael";
```

20.3.2 变量

在 JavaScript 中,变量被用来储存数据。在代码行的前面,关键字 `var` 用来声明一个名称为 `myName`(包含我的名字)的变量。JavaScript 支持多种不同的数据类型,例如文本串、数字、布尔真/伪值。这种语言对分类要求不严格,这就是说,在使用变量之前你不必明确声明此变量的数据类型。JavaScript 允许你使用不同类型的变量,如果必要的话,它会自动进行不同类型间的转换。

20.3.3 表达式

表达式有些像编程等式,它使你可以进行数学运算。然而,表达式并不仅限于数学运算;譬如,你也可以把字符串加在一起。下面是 JavaScript 表达式的例子:

```
var area = length * width;
```

这个表达式用变量 `length` 乘以变量 `width` 计算出面积。结果储存在变量 `area` 中。乘法运算符(`*`)被用来使数字相乘,而赋值运算符(`=`)表明结果将储存在变量 `area` 中。下面举例说明把字符串相加组成信息的表达式:

```
var msg = "hello, " + myName + "!"
```

这个例子中使用前面的变量 `myName`,变量 `msg` 结果值是“Hello,Michael!”。注意加法运算符(`+`)被用来把字符串加在一起。

20.3.4 注释

你可以使用注释使 JavaScript 代码中含有注解和描述,这有助于解释一节代码的作用。JavaScript 支持两种注释:单行注释和多行注释。单行注释以两个前斜线开始(`//`),表明本行后面部分为注释。下面是单行注释的例子:

```
var area = length * width ; //calculate the area
```

你也可以另起一行写单行注释,如:

```
//Calculate the area  
var area = length * width;
```

与单行注释不同,多行注释可以书写多行。它以一个前斜线和一个星号开始(`/*`)并以一个星号和一个前斜线结束(`*/`)。多行注释中,`/*` 和 `*/`之间的内容都被认为是注释部分。JavaScript 程序运行时,所有注释都被忽略。下面是多行注释的例子:

```
/* Calculate the area using  
   the length and the width  
   variables, and store the  
   result in the area variable. */  
var area = length * width;
```

20.3.5 函数

函数是一组 JavaScript 语句,它与 JavaScript 程序的其他部分分离。函数执行某一动作,

常常又返回结果。JavaScript 程序调用函数来委派工作,这样产生有益的影响是得到更好的脚本结构。JavaScript 提供了一些标准的函数供你调用。你也可以创建自己的函数执行常见的任务。例如,JavaScript 标准函数是 `eval`,它用来求一个串表达式的值。你把表达式置于一对括号()中间,就是把这个串表达式传送给了函数 `eval()`。下面是如何使用函数 `eval()` 的例子:

```
var expression = "length * width";  
var area = eval(expression);
```

在这个例子中,储存在变量 `expression` 中的串表达式通过将其传递给函数 `eval()` 来计算。表达式的结果储存在变量 `area` 中。

20.3.6 对象

JavaScript 是一种面向对象的脚本语言,这就是说,它支持对象,这是一种由特征和方法组成的专用数据结构。特征构成了对象的数据部分,而方法则更像函数。有关 JavaScript 对象的例子很多,如内部的 `Date` 对象,它代表日期和时间。通过调用这个方法,就可以得到 `Date` 对象的具体信息。下面的例子得到了与含有当前日期和时间的 `Date` 对象有关的年代:

```
var date = new Date();  
var year = date.getFullYear();
```

注意,`Date` 对象必须先创建,你才可以调用其中的 `getFullYear()` 方法。这要由 `new` 运算符来完成,`new` 运算符用来创建对象。在这种情况下,新的 `Date` 对象代表当前日期和时间。这样 `getFullYear()` 方法就得到了年代,这是一个四位数字,例如 1999 或 2001。

20.4 再谈 XML DOM

尽管在第 15 章“利用文件对象模式”中已经详细讲述了 XML DOM,但是我还要再谈一下,因为它对于 XML 脚本来说十分重要。JavaScript 完全依赖 XML DOM 来存取 XML 文件的逻辑结构,这是基本的文件处理。XML DOM 提供了一组对象和接口,不仅用来抽取信息、修改信息,处理 XML 内容,而且用来装入和解析 XML 文件。

XML DOM 是 Internet Explorer 5.0 支持的第一层 W3C DOM。DOM 把 XML (和 HTML) 文件揭示为由节点组成的分层树形结构。你可以使用 DOM 对象遍历这一树形结构并处理文件数据。树上的每个节点都有一个类型,它对应于出现在文件中的数据类型。例如,大多数节点是属于某种类型的,如元素、属性或文本。

DOM 实际上是 W3C 推荐标准,因此其中所定义的对象和接口都是标准化的。这是非常有益的,因为这使脚本可以在任何支持标准 DOM 的浏览器上运行。Internet Explorer 和 Netscape Navigator 都支持 W3C 的第一层 DOM 推荐标准,这在以下网址可获得:<http://www.w3.org/TR/REC-DOM-Level-1>。还有更宽泛的 DOM 规范,称作第二层 DOM,目前还处于工作草案阶段。

Internet Explorer 5.0 提供了一系列对象,可用来通过第一层 DOM 存储 XML 文件。下

面是一些 XML DOM 揭示的最有用的对象：

- XMLDOMDocument
- XMLDOMNode
- XMLDOMNodeList
- XMLDOMNamedNodeMap
- XMLDOMElement
- XMLDOMAttribute
- XMLDOMText
- XMLDOMParseError

注意：XML DOM 提供了对象和接口来访问 DOM 的功能。接口实际上是 COM (Component Object Model, 组件对象模型) 接口, 可使用 C/C++ 和 Visual Basic。对象实际上是 ActiveX 对象, 可使用脚本语言。关键是 XML DOM 中的对象和接口提供了同一功能, 不过, 两者揭示的方式却不同。

对于脚本, 这些 XML DOM 对象中最重要的是 XMLDOMDocument 对象, 它揭示了一个 XML 文件的全部结构并提供了处理这个文件的特征和方法。与 XMLDOMDocument 对象相配合, 其他对象通过脚本代码用来处理 XML 文件。下面几部分将向你详细介绍这些对象。

20.4.1 XMLDOMDocument 对象

XMLDOMDocument 对象代表一个 XML 文件并提供对代表这个文件的逻辑结构的树形结构的访问。这一对象包含用来遍历文件树并存取和处理文件数据的特征和方法。尽管在 Microsoft 的 XML 网址 (<http://msdn.microsoft.com/xml>) 中你可以看到 XMLDOMDocument 对象的完整文件, 但是表 20.1 还是列出了 XMLDOMDocument。

表 20.1 XMLDOMDocument 对象中所包含的重要特征

| 属性 | 用法 |
|--------------------|---|
| doctype | 含有文件类型节点, 该节点指定了文件的 DTD |
| documentElement | 含有文件的根元素 |
| childNodes | 含有出现在紧跟在文件元素下面的元素节点的清单 |
| firstChild | 含有文件元素的第一个子文件元素 |
| lastChild | 含有文件元素的最后一个子文件元素 |
| text | 含有节点的文字内容和它的子树 |
| async | 指出文件下载的时候是否异步 |
| validateOnParse | 指出文件是否需要被验证 |
| readyState | 指出文件的当前状态 |
| onreadystatechange | 指出在 readyState 属性改变时要调用的时间处理函数 |
| parseError | 含有 XMLDOMParseError 对象, 这个对象含有最后一个解析错误的信息 |

这些特征中有一些在本章后面当你通过一些脚本样例工作时会用到。你也将使用 XMLDOMDocument 对象中所包含的一些方法。表 20.2 列出了 XMLDOMDocument 对象中的方法,这对建立 XML 脚本是十分有用的。

表 20.2 XMLDOMDocument 对象中所包含的重要方法

| 方法 | 用法 |
|------------------------|-------------------------|
| load() | 从文件(URL)中加载 XML 文件 |
| loadXML() | 直接从文本的 XML 代码中加载 XML 文件 |
| getElementsByTagName() | 取得给定元素类型(标记名称)的元素的集合 |

注意:重要的是要明白 XMLDOMDocument 对象包含比上面列举的要多很多的特征和方法。许多这些特征和方法使用节点,这一点只要留意所有的 XML 文件都是以一个根节点开始的就不感到奇怪了。要想更多地了解如何运用 DOM 特征和方法,请查阅第 15 章“使用文件对象模型(DOM)”。你也可以访问 Microsoft 的 XML 网址: <http://microsoft.com/xml> 全面查阅 XML DOM。

20.4.2 XMLDOMNode 对象

XMLDOMNode 对象在 XML 逻辑数据树内代表一个个人节点。因为 XML 文件中每一个主要结构都是以树节点为模式的。所以 XMLDOMNode 对象实际上是 XML DOM 中最重要的对象。事实上 XMLDOMDocument 对象本身就是一个 XMLDOMNode 对象,这说明许多包含其中的特征和属性都使用节点。XMLDOMNode 对象也是其他几个重要 XML DOM 对象的基础对象,如:XMLDOMElement,XMLDOMAttribute 和 XMLDOMText。表 20.3 列出了 XMLDOMNode 对象的特征,这对建立 XML 脚本是很有用的。

表 20.3 XMLDOMNode 对象所包含的重要特征

| 属性 | 用法 |
|--------------|-------------------|
| dataType | 含有节点的数据类型 |
| baseName | 含有节点的基础名称(包括名字空间) |
| prefix | 含有名字空间前缀 |
| namespaceURI | 含有名字空间 URI |
| nodeName | 含有节点的名字 |
| nodeType | 含有 XML DOM 节点类型 |
| nodeValue | 含有节点的文字内容 |
| attributes | 含有节点的属性清单 |
| parentNode | 含有节点的父节点(如果存在) |
| childNodes | 含有节点的子节点清单(如果存在) |
| firstChild | 含有节点的第一个子节点 |

续表

| 属性 | 用法 |
|-----------------|-------------------|
| lastChild | 含有节点的最后一个子节点 |
| previousSibling | 含有节点的上一个兄弟 |
| nextSibling | 含有节点的下一个兄弟 |
| parsed | 指出节点及其全部后继是否已经被解析 |
| text | 含有节点及其所有后继的文字内容 |

XMLDOMNode 对象还提供了在你开发 XML 脚本时可能会用到的几种方法。表 20.4 列出了这些方法。

表 20.4 XMLDOMNode 对象所包含的重要方法

| 方法 | 用法 |
|--------------------------|---------------------------------------|
| hasChildNodes() | 指出节点是否有子节点 |
| appendChild() | 在节点的子节点清单最后加入一个新的子节点 |
| insertBefore() | 在节点的子节点清单中的指定节点之前插入一个新的子节点 |
| replaceChild() | 用一个新的子节点替换节点子节点清单中的特定子节点 |
| removeChild() | 在节点的子节点清单中删除特定子节点 |
| selectNodes() | 根据给定的模式匹配操作选出节点中一组节点 |
| selectSingleNode() | 根据给定的模式匹配操作选出节点中一组节点中的第一个 |
| transformNode() | 用指定的 XSL 样式表来转化节点及其所有后继节点 |
| transformNodesToObject() | 用指定的 XSL 样式表来转化节点及其所有后继节点并且将结果放在指定对象中 |

20.4.3 XMLDOMNodeList 对象

XMLDOMNodeList 对象代表一系列节点,一般来说是 XML 文件主树的子树。这个对象基本上是一组节点,你可以通过使用特征和方法穿行于其间。一个 XMLDOMNodeList 类型的对象可以通过 childNodes 属性和 XMLDOMNode 对象的 selectNodes() 方法,或者用 XMLDOMDocument 对象的 getElementsByTagName() 方法返回。

XMLDOMNode 对象只包含一个特征 length,它包含所列的节点数目。这一特征对于建立重复通过所列节点的循环是有用处的。表 20.5 列出了 XMLDOMNodeList 对象所支持的常见方法。

表 20.5 XMLDOMNodeList 对象所包含的重要方法

| 方法 | 用法 |
|------------|---------------|
| item() | 从节点清单中取得独立的节点 |
| nextNode() | 从节点清单中取得下一个节点 |
| reset() | 重置节点清单,从头开始遍历 |

20.4.4 XMLDOMNamedNodeMap 对象

XMLDOMNamedNodeMap 对象代表一系列与已知元素有关的属性,尽管从技术上讲属性并不认为是 XML DOM 中的节点。这一对象在某些方面与 XMLDOMNodeList 相似,它也是一个集合分类,你可以通过使用方法重复穿行其间。然而,XMLDOMNamedNodeMap 对象专门设计用来保持一系列属性。XMLDOMNamedNodeMap 对象的意义就在于其所列的节点可以通过名字来存取,这对于你在处理属性时是特别重要的。XMLDOMNamedNodeMap 类型的对象可以通过 XMLDOMNode 对象的 attributes 特征以及由此派生的其他对象返回。

XMLDOMNamedNodeMap 对象包含一个特征 length,它包含节点 map 集合中的属性数目。这一特征常被用做重复通过节点 map 中属性的基础。表 20.6 列出了 XMLDOMNamedNodeMap 对象中的方法,这对开发 XML 脚本是有用的。

表 20.6 XMLDOMNamedNodeMap 对象所包含的重要方法

| 方法 | 用法 |
|-------------------|----------------------------|
| getNamedItem() | 取得 map 中符合特定名字的属性 |
| setNamedItem() | 通过在 map 中加入或替换属性值来设置指定的属性值 |
| removeNamedItem() | 从 map 中删除指定属性 |
| nextNode() | 从 map 中取得下一个属性 |
| reset() | 重新设置 map,从头开始遍历 map |

20.4.5 XMLDOMElement 对象

XMLDOMElement 对象代表 XML 文件中的一个元素,是从 XMLDOMNode 对象派生出来的。因为 XMLDOMNode 对象是 XMLDOMElement 的基础对象,所以 XMLDOMElement 对象提供了与 XMLDOMNode 相同的所有方法。它还提供了一些它自己的方法,列在表 20.7 中。这些方法主要与属性有关,而属性通常和元素相关联。

表 20.7 包含在 XMLDOMElement 对象中的几个重要的方法

| 方法 | 用法 |
|------------------------|--------------------------------------|
| getElementsByTagName() | 取得给定元素类型(标记名)的元素的集合 |
| getAttribute() | 取得特定属性的值 |
| getAttributeNode() | 取得特定的属性节点 |
| setAttribute() | 通过在和该元素相关的节点 map 中加入或替换元素的值来设置特定元素的值 |
| setAttributeNode() | 通过在和该元素相关的节点 map 中加入或替换节点来设置特定元素的节点 |
| removeAttribute() | 将特定的属性从与元素相关的节点 map 中删除 |
| removeAttributeNode() | 将特定的属性节点从与元素相关的节点 map 中删除 |

20.4.6 XMLDOMAttribute 对象

XMLDOMAttribute 对象表示了一个与元素相关联的属性,这个元素本身是用 XMLDOMElement 对象来表示的。属性被保存在一个节点 map 中,这个节点 map 可以用 XMLDOMNamedNodeMap 对象来表示。为了编程的需要,XML DOM 将属性当作节点来处理,尽管在文件的 XML 树中属性并不被当作节点。因为属性实际上不是 XML 树的一部分,所以那些处理基于树形结构的层次关系的那些 XMLDOMNode 属性和方法(比如 previousSibling 和 nextSibling)不能应用于属性。

XMLDOMAttribute 对象提供的方法都没有超出 XMLDOMNode 对象所能够提供的。不过,它加入了一些用来访问属性的名字和值的属性:name 属性和 value 属性。

20.4.7 XMLDOMText 对象

XMLDOMText 对象表示了元素或属性的文字内容,它是从 XMLDOMNode 对象中导出的。因为 XMLDOMText 对象是基于 XMLDOMNode 对象的,所以 XMLDOMText 对象提供了和 XMLDOMNode 相同的方法。然而,XMLDOMText 对象还提供了一些它自己的方法,如表 20.8 所示。这些方法和元素的文字内容有关。

表 20.8 XMLDOMText 对象中的几个重要的方法

| 方法 | 用法 |
|-----------------|------------------------|
| appendData() | 将特定的字符串追加到节点的文本内容之后 |
| deleteData() | 从节点的文本内容之中删除特定的子串 |
| insertData() | 在节点的文本内容的指定位置插入特定字符串 |
| replaceData() | 用指定字符串替换节点文本内容中指定数量的字符 |
| splitText() | 将节点文本内容在指定位置处分成两个字符串 |
| substringData() | 取得节点的文本内容的一个子串 |

20.4.8 XMLDOMParseError 对象

XMLDOMParseError 对象表示了在解析 XML 文件过程中发生的错误。这个对象对于确定错误为什么发生非常有用,因为它描述了这个错误。表 20.9 列出了 XMLDOMParseError 对象的属性,它们提供了与对象相关联的解析错误的信息,包括错误代码、行号、错误描述以及其他一些信息。

表 20.9 XMLDOMParseError 对象中的重要属性

| 属性 | 用法 |
|-----------|-----------------------|
| errorCode | 包含最后一个解析错误的错误代码 |
| filepos | 包含 XML 文件中发生错误的绝对字符位置 |
| line | 包含 XML 文件中发生错误的行号 |

续表

| 属性 | 用法 |
|---------|-------------------------|
| linepos | 包含在发生错误的行中错误的相对字符位置 |
| reason | 包含错误的原因 |
| srcText | 包含 XML 文件中发生错误的那一行文本 |
| url | 包含出现最后一个错误的 XML 文件的 URL |

20.5 开发 XML 脚本

尽管理解脚本语言和 XML DOM 都是非常重要的,然而真正理解如何开发 XML 脚本的最好办法是学习一些实际的例子。本章的其余部分将着重介绍几个例子,从而说明如何在 Internet Explorer 5.0 中使用 JavaScript 和 XML DOM 来访问和操纵 XML 文件数据。我们在本书的前面所看到的 MoviesXML 文件将被用作这些脚本实例的基础。这个文件显示在清单 20.2 中,因为你可能已经忘记它的结构是什么样子的了。

清单 20.2 XML 文件 Movies.xml

```
<? xml version="1.0" standalone="no" ? >
<! DOCTYPE movies SYSTEM "Movies.dtd" >
<movies>
  <movie type="comedy" rating="PG-13" review="5" year="1987">
    <title>Raising Arizona</title>
    <writer>Ethan Coen</writer>
    <writer>Joel Coen</writer>
    <producer>Ethan Coen</producer>
    <director>Joel Coen</director>
    <actor>Nicolas Cage</actor>
    <actor>Holly Hunter</actor>
    <actor>John Goodman</actor>
    <comments>A classic one-of-a-kind screwball love story. </comments>
  </movie>
  <movie type="comedy" rating="R" review="5" year="1988">
    <title>Midnight Run</title>
    <writer>George Gallo</writer>
    <producer>Martin Brest</producer>
    <director>Martin Brest</director>
    <actor>Robert De Niro</actor>
    <actor>Charles Grodin</actor>
    <comments>The quintessential road comedy. </comments>
  </movie>
  <movie type="mystery" rating="R" review="5" year="1995">
    <title>The Usual Suspects</title>
    <writer>Christopher McQuarrie</writer>
    <producer>Bryan Singer</producer>
    <producer>Michael McDonnell</producer>
    <director>Bryan Singer</director>
```

```

    <actor>Stephen Baldwin</actor>
    <actor>Gabriel Byrne</actor>
    <actor>Benicio Del Toro</actor>
    <actor>Chazz Palminteri</actor>
    <actor>Kevin Pollak</actor>
    <actor>Kevin Spacey</actor>
    <comments>A crime mystery with incredibly intricate plot twists. </comments>
</movie>

<movie type="sci-fi" rating="PG-13" review="4" year="1989">
    <title>The Abyss</title>
    <writer>James Cameron</writer>
    <producer>Gale Anne Hurd</producer>
    <director>James Cameron</director>
    <actor>Ed Harris</actor>
    <actor>Mary Elizabeth Mastrantonio</actor>
    <comments>A very engaging underwater odyssey. </comments>
</movie>
</movies>

```

理解 XML 脚本首先要做的事情之一就是,XML 处理函数将 XML 文件解析为 XML 树时会花费有限的一些时间。此外,XML 处理函数可能会在解析文件的过程中遇到一些错误,在这种情况下你可能想要知道是什么出错了。清单 20.3 包含了一个叫做 ReadyState 的 Web 页面,它用 JavaScript 来跟踪 XMLDOMDocument 对象的 readyState 属性。这个页面中的脚本说明了如何跟踪文件状态,以及如何用 XMLDOMParseError 对象来处理解析错误。

清单 20.3 ReadyState Web 页面(ReadyState.html)

```

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>Ready State Example HTML Document</title>
    <script language="JavaScript">
      var xmldoc;

      function LoadDoc() {
        xmldoc = new ActiveXObject("Microsoft.xmlDOM");
        READYINFO.innerHTML += "<br>" + "Tracking ready states:" + "<br>";

        Xmldoc.onreadystatechange = TrackReadyState;
        Xmldoc.load(URL.value);
        DocInfo();
      }

      function TrackReadyState() {
        var state = xmldoc.readyState;
        READYINFO.innerHTML += "Ready state = " + state + "<br>";
        if (state == 4) {
          var err = xmldoc.parseError;
          if (err.errorCode != 0)
            READYINFO.innerHTML += err.reason + "<br>";
          Else
            READYINFO.innerHTML += "Document loaded successfully." + "<br>";
        }
      }
    </script>
  </head>
</html>

```

```

    ><br>"
  }
  }
  function DocInfo() {
    DOCINFO.innerHTML += "Document URL: " + xmldoc.url + "<br>";
    DOCINFO.innerHTML += "Document type: " + xmldoc.doctype.name + "<
br>";
  }

</script>
</head>
<body>
  Please enter a URL for an XML document:
  <br>
  <input type="text" size="50" id = "URL">
  <input type="button" value="Load XML Document" onclick="LoadDoc()">
  <div id="READYINFO" style="color:blue;"></div>
  <div id="DOCINFO" style="color:green;"></div>
</body>
</html>

```

这个 Web 页面用 JavaScript 来加载和处理通过一个文本输入框指定的 XML 文件。首先用 JavaScript 的 new 操作符来把这个文件当作一个 ActiveX 对象创建,并将“microsoft.xmlDOM”作为参数传递给 ActiveObject() 创建函数(构造函数)。这个函数返回一个 XMLDOMDocument 对象,这个对象被保存在 xmldoc 变量中。接下来 TrackReadyState() 函数被设置为响应文件的 onreadystatechange 属性的事件处理方法。这将导致只要文件的状态发生了变化 TrackReadyState() 函数就会被调用。然后用文件的名字调用 load 方法来加载 XML 文件。最后,调用 DocInfo() 方法来显示新加载的文件的信息。

TrackReadyState() 函数在 readyState 属性改变的时候将被调用,在文件被加载和解析的过程中这种改变会发生若干次。下面是一些可能的不同 readyState 属性值,以及它们的意义。

- 1——文件开始被加载但是还没有开始解析数据。
- 2——文件加载结束但是仍然在解析过程中,树结构尚不可用。
- 3——文件已经被加载并且被部分解析,而且部分只读的树结构已经可用了。
- 4——文件完全被加载并且被解析,或者成功或者失败。

文件加载和解析的处理过程将经历以上几个步骤并且最后 readyState 的取值总是 4,无论文件的加载和解析是否成功。通过检查 parseError 对象的 errorCode 属性值可以判断是否成功,parseError 本身是 XMLDOMDocument 对象的一个成员。TrackReadyState() 函数显示 readyState 属性的当前值,并且在文件加载结束之后检查错误。

DocInfo() 函数在文件被加载之后被调用,它显示了文件的 URL 和文件类型。图 20.2 显示了在 Internet Explorer 5.0 中打开的成功加载了 Movies.xml 文件的 ReadyState Web 页面。

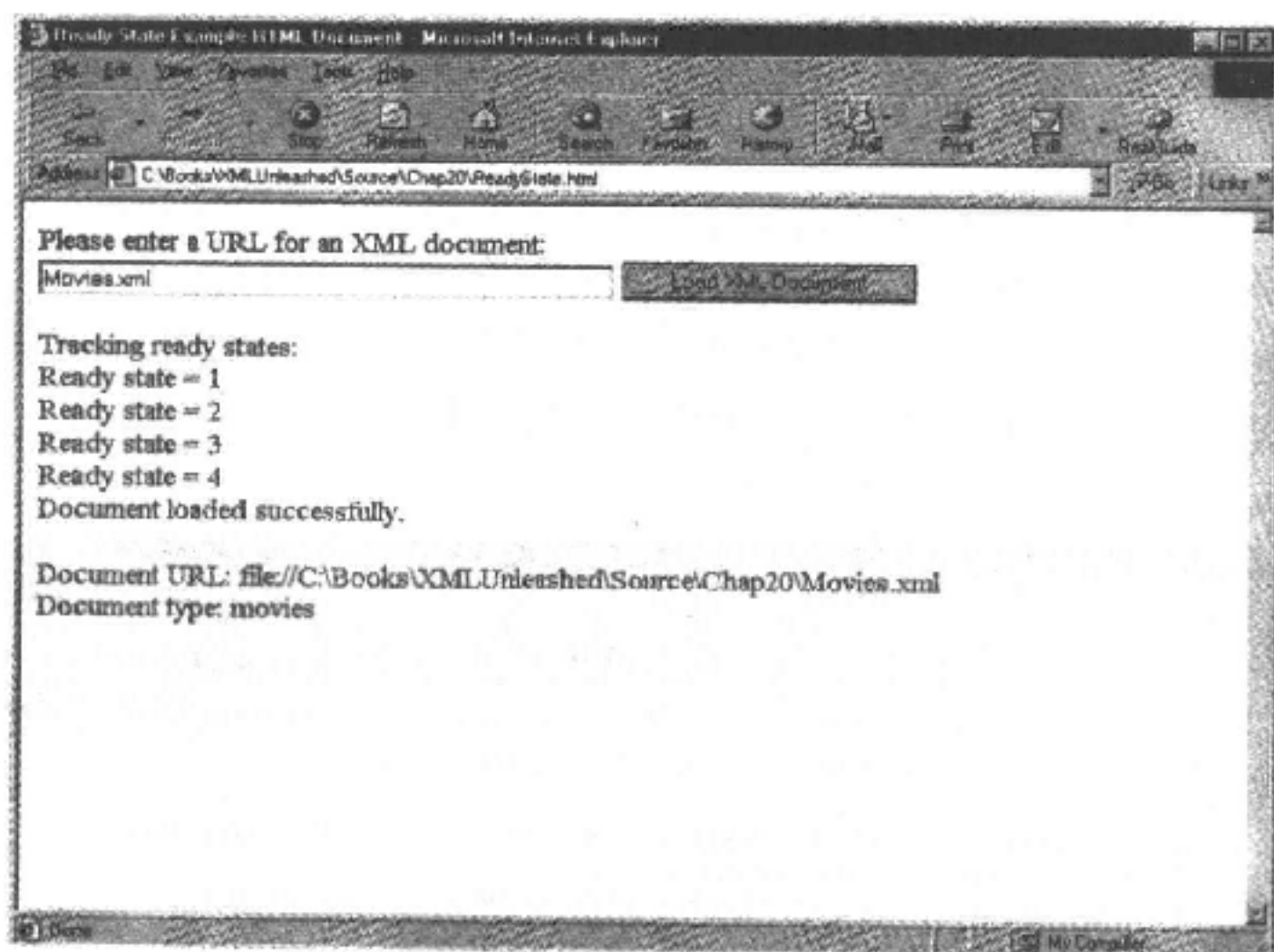


图 20.2 在 Internet Explorer 5.0 中显示成功加载了 Movies.xml 文件的 ReadyState Web 页面

尽管跟踪加载 XML 文件过程的状态提供了了解 XML DOM 如何工作的一个有趣的例子,但是它并不是用脚本处理实际问题的全部。XML 脚本的实际能力在于挖掘 XML 数据并提取信息。清单 20.4 包含了被称为 ListMovies1 的 Web 页面的代码,这个页面用 JavaScript 来列出在 Movies.xml 文件中标记出的所有电影。

清单 20.4 Web 页面 ListMovies1 (ListMovies1.html)

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>List Movies 1 Example HTML Document</title>
    <script language="JavaScript">
      var xmldoc;

      function LoadDoc() {
        xmldoc = new ActiveXObject("Microsoft.xmlDOM");
        xmldoc.load("Movies.xml");
      }
    </script>
  </head>
  <body>
    Click the button to list the movies in the Movies XML document.
    <br>
    <input type="button" value="List Movies" onclick="ListMovies()">
    <div id="DOCCONTENT"></div>
  </body>
</html>
```



```

function ListMovies() {
    LoadDoc();
    root = xmldoc.documentElement;
    for(i = 0; i < root.childNodes.length; i++) {
        DOCCONTENT.innerHTML += "<b>" +
            root.childNodes.item(i).firstChild.text + "</b><br>";
        DOCCONTENT.innerHTML += root.childNodes.item(i).lastChild.text + "<br
><br>";
    }
}
</script>
</head>
<body>
    Click the button to list the movies in the Movies XML document.
    <br>
    <input type="button" value="List Movies" onclick="ListMovies()">
    <div id="DOCCONTENT"></div>
</body>
</html>

```

例子 ListMovies2 和例子 ListMovies1 的基本结构是一样的,它还显示了 Movies.xml 文件中每一个电影的 title 和 comment。firstChild 和 lastChild 属性被用来访问 title 和 comment 子元素。一定要注意到每一个电影的标题被加粗显示,这说明了如何编写脚本代码来在将它转化为 HTML 标记的时候装饰 XML 数据。图 20.4 显示了 Internet Explorer 5.0 中打开的 Web 页面 ListMovies2。

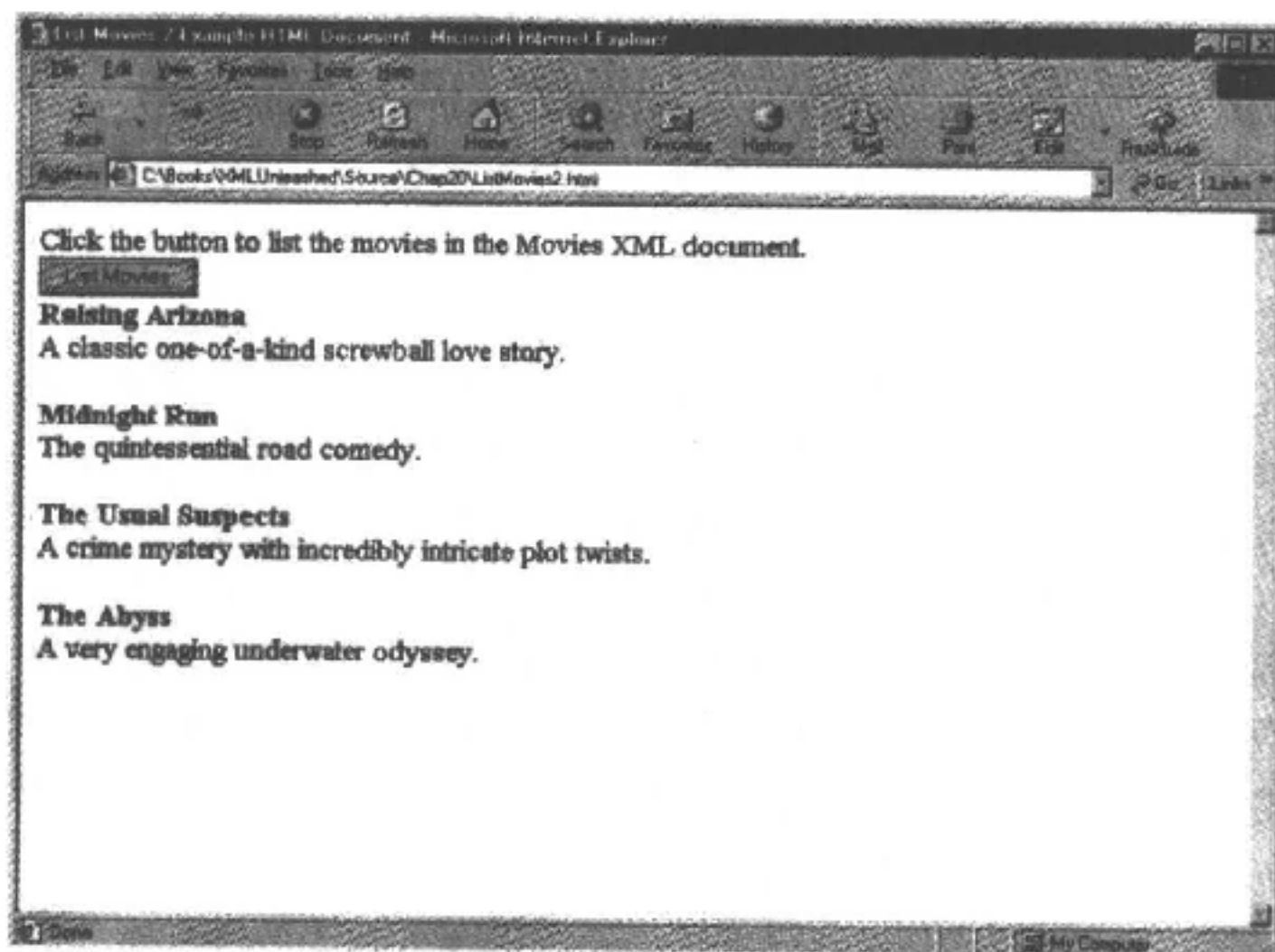


图 20.4 Internet Explorer 5.0 中显示 Web 页面 ListMovies2

这两个 ListMovie 的例子都使用了相关的方法来访问元素,因为 title 和 comment 元素

是通过 firstChild 和 lastChild 属性来访问的。清单 20.6 包含了形成 Web 页面 ListActors 的代码,这个页面的代码使用绝对的方法来访问特定类型的元素。

清单 20.6 Web 页面 ListActors(ListActors.html)

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>List Actors Example HTML Document</title>
    <script language="JavaScript">
      var xmldoc;

      function LoadDoc() {
        xmldoc = new ActiveXObject("Microsoft.xmlDOM");
        xmldoc.load("Movies.xml");
      }

      function ListActors() {
        LoadDoc();
        actors = xmldoc.getElementsByTagName("actor");
        for (i = 0; i < actors.length; i++) {
          DOCCONTENT.innerHTML += actors(i).text + "<br>";
        }
      }
    </script>
  </head>
  <body>
    Click the button to list the actors in the Movies XML document.
    <br>
    <input type="button" value="List Actors" onclick="ListActors()">
    <div id="DOCCONTENT"></div>
  </body>
</html>
```

例子 ListActors 说明了如何使用 getElementsByTagName() 函数来取得符合给定类型的元素的清单。在这个例子中,只取得 actor 类型的元素。请理解所有 actor 元素都被取得,而无论它们在 XML 树中的位置。通过遍历 Actor 元素的清单可以显示 Movies.xml 文件中的所有演员。图 20.5 显示了在 Internet Explorer 5.0 中打开的 Web 页面 ListActors。

上面我们说明的一点是如何从 XML 文件中取得数据而并未了解这些数据的特定内容。然而,实际上选出那些符合某种模式的或者具有固定值的数据通常是必要的。比如,你可能想要只显示那些喜剧电影,或者你可能想要排除那些恐怖片。清单 20.7 含有形成 Web 页面 FilterMovies 的代码,它只显示那些喜剧电影。

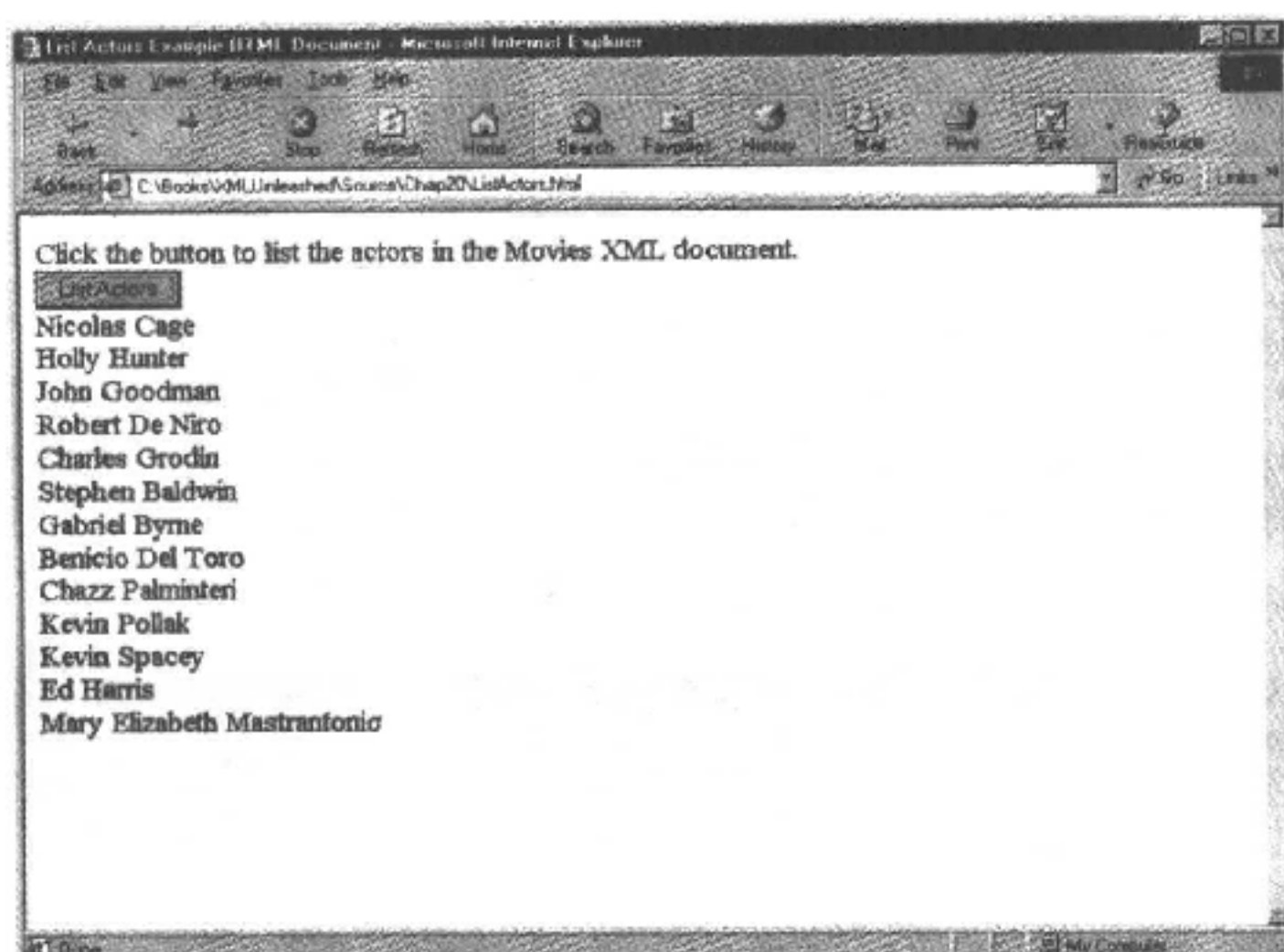


图 20.5 在 Internet Explorer 5.0 中显示 Web 页面 ListActors

清单 20.7 Web 页面 FilterMovies(filterMovies.html)

```

<! DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>Filter Movies Example HTML Document</title>
    <script language="JavaScript">
      var xmldoc;

      function LoadDoc() {
        xmldoc = new ActiveXObject("Microsoft.xmlDOM");
        xmldoc.load("Movies.xml");
      }

      function FilterMovies() {
        LoadDoc();
        root = xmldoc.documentElement;
        for (i = 0; i < root.childNodes.length; i++)
          for (j = 0; j < root.childNodes.item(i).attributes.length; j++)
            if ((root.childNodes.item(i).attributes(j).name == "type") &&
                (root.childNodes.item(i).attributes(j).value == "comedy")) {
              DOCCONTENT.innerHTML += "<b>" +
                root.childNodes.item(i).firstChild.text + "</b><br>";
              DOCCONTENT.innerHTML += root.childNodes.item(i).lastChild.text
+ "<br><br>";
            }
      }
    </script>
  </head>

```

```

<body>
  Click the button to filter comedies in the Movies XML document.
  <br>
  <input type="button" value="Filter Movies" onclick="FilterMovies()">
  <div id="DOCCONTENT"></div>
</body>
</html>

```

例子 FilterMovies 使用一对 for 循环和 attributes 属性来先检查“type”类型的属性然后检查值“comedy”。含有符合上述条件的属性的电影将被显示。图 20.6 显示了在 Internet Explorer 5.0 中打开的 Web 页面 FilterMovies。

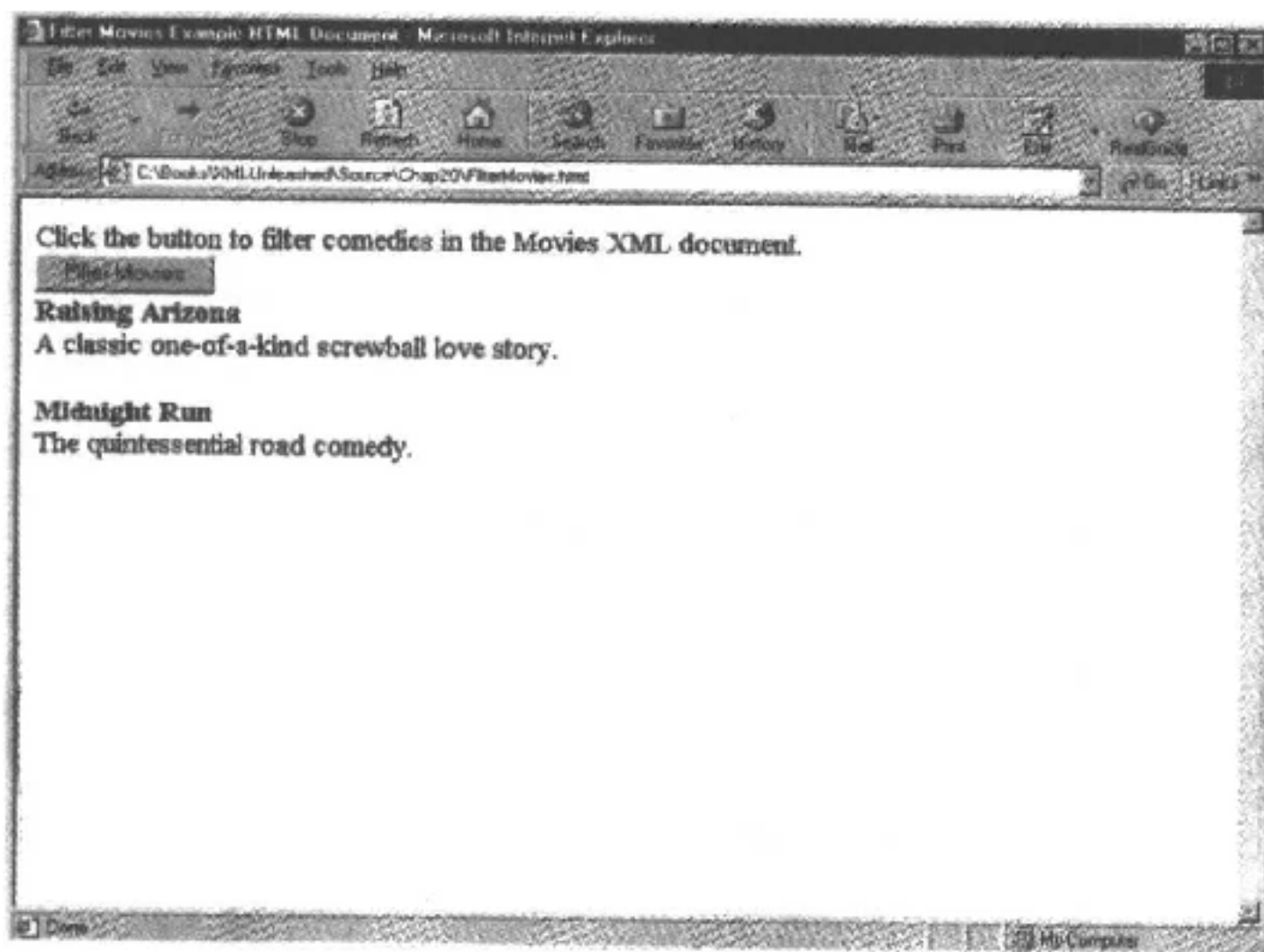


图 20.6 在 Internet Explorer 5.0 中显示 Web 页面 FilterMovies

已经有许多关于 XML 如何改变 Web 页面的检索方式的讨论。XML 是为 Web 页面中的数据添加上下文的完美的技术,因而它为搜索引擎提供了检索数据的更准确的方法。清单 20.8 含有形成 Web 页面 SearchMovies 的代码,它使用简化了的方法来根据 movie 属性检索电影。

清单 20.8 Web 页面 SearchMovies(SearchMovies.html)

```

<! DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>Search Movies Example HTML Document</title>
    <script language="JavaScript">
      var xmldoc;

```

```

function LoadDoc() {
    xmldoc = new ActiveXObject("Microsoft.xml.dom");
    xmldoc.load("Movies.xml");
}

function SearchMovies() {
    LoadDoc();
    DOCCONTENT.innerHTML = " ";
    root = xmldoc.documentElement;
    for (i = 0; i < root.childNodes.length; i++)
        if ((root.childNodes.item(i).attributes(j).name == NAME.value) &&
            (root.childNodes.item(i).attributes(j).value == VALUE.value)) {
    DOCCONTENT.innerHTML += "<b>" +
        root.childNodes.item(i).firstChild.text + "</b><br>" ;
        DOCCONTENT.innerHTML += root.childNodes.item(i).lastChild.text + "
<br><br>" ;
        }
    }
</script>
</head>
<body>
    Please enter the movies search attribute: <input type = "text" size = "15"
        id = "NAME">
        <br>
    Please enter the movie search value: <input type = "text" size = "20"
        id = "VALUE">
        <br>
    <input type="button" value="Search Movies" onclick="SearchMovies()">
    <div id="DOCCONTENT"></div>
</body>
</html>

```

Web 页面 SearchMovies 使用两个文本输入框来让用户输入一个属性和一个值来作为检索的基础。movies.xml 文件中的每一个电影都被分析,以了解它是否符合检索条件。如果符合,那么就显示这个电影。图 20.7 显示了在 Internet Explorer 中打开的 Web 页面 SearchMovies。

这个图显示了检索 Pg-13 类型的电影的结果。你可以使用 SearchMovies Web 页面来根据 movie 元素的任何属性进行检索。图 20.8 显示了检索重看率为 5 的那些电影的结果,这是电影文件中可用的最高比率。

20.6 总 结

作为描述结构化信息的元语言,XML 需要一种技术来构建交互的、数据驱动的应用程序。脚本语言可以作为 XML 的这种技术,特别是在 Web 页面的上下文中。在 Web 页面中的 XML 脚本语言的功能来自于它们对 HTML 和 XML 文件的逻辑结构的直接访问。脚本语言对于基于 Web 的 XML 开发是非常重要的,因为学习和使用它们相对比较容易,而且脚本语言一般是跨平台的。

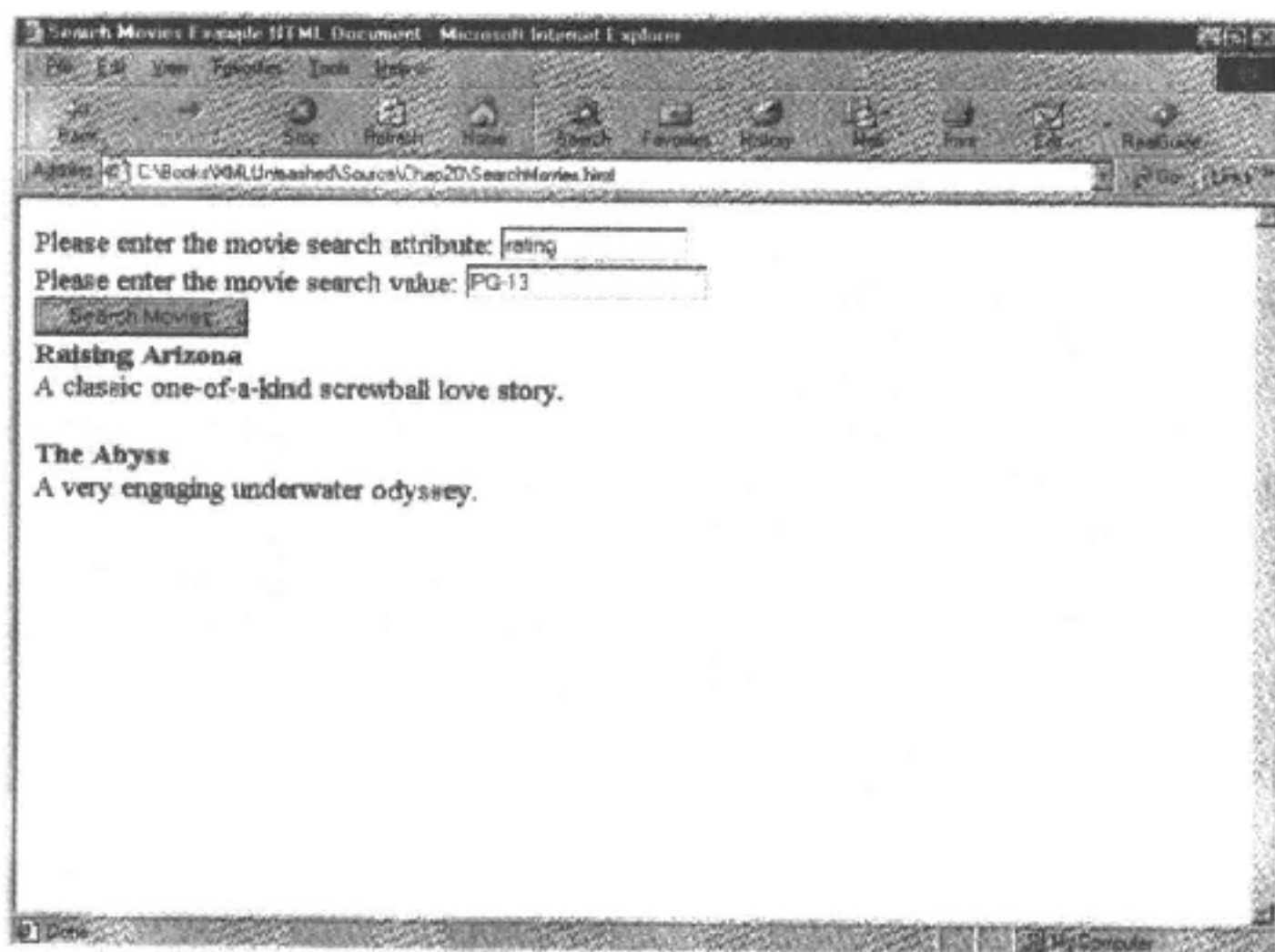


图 20.7 在 Internet Explorer 5.0 中显示 Web 页面 SearchMovies

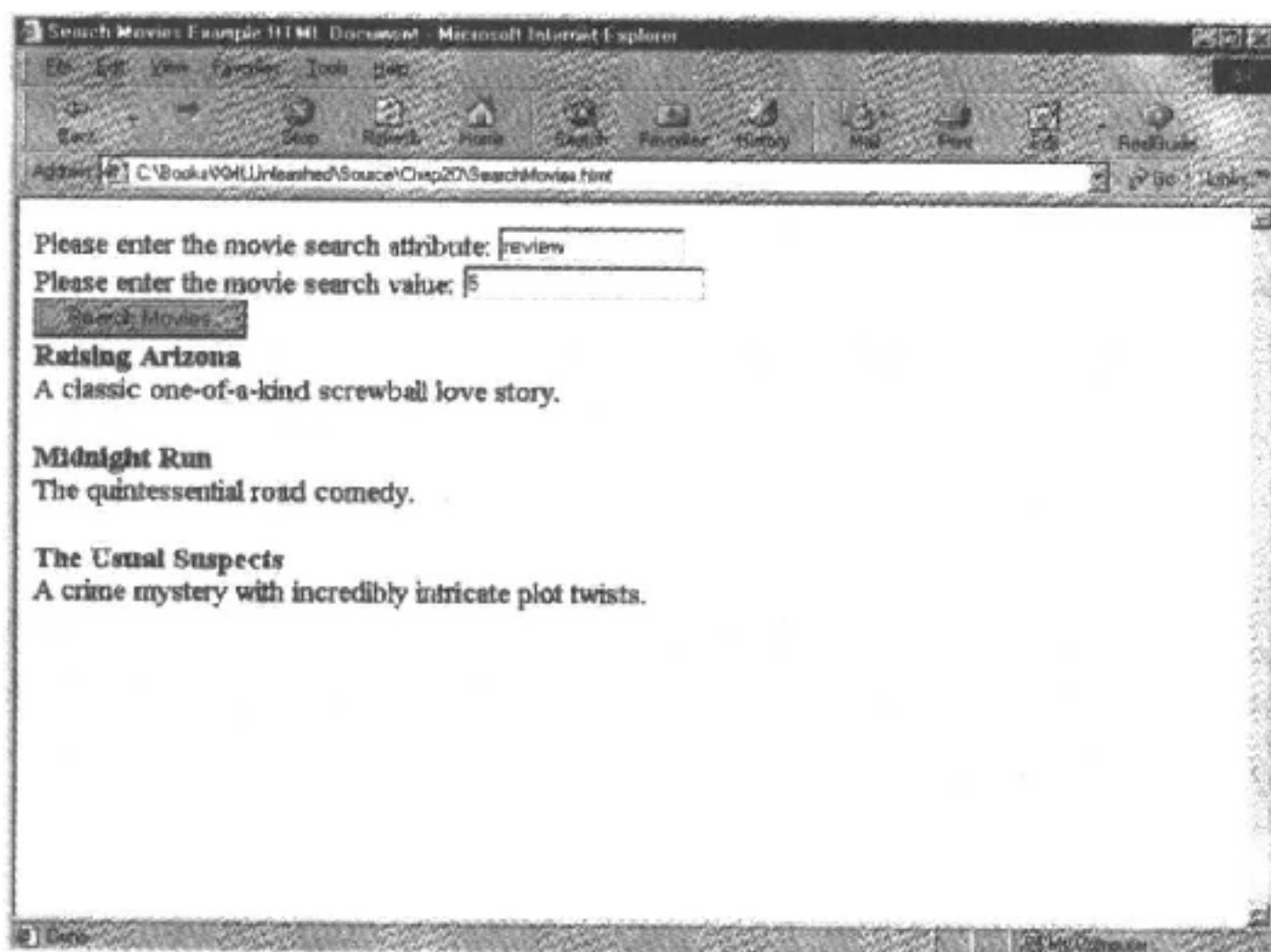


图 20.8 在 Internet Explorer 5.0 中显示 SearchMovies Web 页面

第 5 部分 使用 XML 操作数据

第 21 章 XML 数据源对象

数据源对象(Data Source Object,DSO)是在 HTML 主页中可以加入结构化数据(包括 XML)的一类对象。

几乎从网络诞生那时起,访问数据库并将结构化的数据下载到网络客户端就是一个必要的事情。通常这要经过下面的几个步骤:

1. 客户端请求来自服务器上的数据。
2. 服务器访问数据存储器并找到数据。
3. 数据被封装在一个 HTML 流中并发送给客户端。

这个过程要为每一个客户端请求的新信息项目而重复的执行。这就导致了对网络的数目众多的遍历。

在很多情况下,从数据存储器中返回一整块数据并把它发送给客户端进行处理是十分有利的。这一点显而易见。实现它的方法有很多种。一种方法是将所有的数据合并成一个长字符串,把这个字符串发送给客户端并依赖于客户端的代码对字符串操作和分类。另一种可选的方法是提供一个实际上可以说是小型数据库的客户端对象,用一个来自服务器的记录集来启动这个数据库。这种方法在 Internet Explorer 4(IE4)中被微软公司引入,并被正式命名为远程数据服务(Remote Data Services)。一整系列的 ActiveX 对象被开发出来用于支持整个庞大的远程数据访问技术。这些小型数据库对象的总称就是数据源对象(DSO, Data Source Objects),不同的 DSO 被开发出来以处理不同类型的数据。

这些 DSO 已经被验证非常的有用。然而,现在为止,它们只能用于 IE4 和 IE5 浏览器上。从头对这项 Microsoft 技术进行讲解会使得我们对它的认识更清晰。因此,不想赶时髦的人可以跳过这一章。

21.1 数据库和远程数据访问

在开始研究 DSO 的复杂技术之前,我们将对数据库进行简单的复习。对于可随意访问数据库的资深读者可以跳过这一节。

21.1.1 数据是如何组织的

通常情况下,数据库是所有信息的集合。然而,数据库在计算机上的通常应用定义是按一种定义方式存储的相关信息的集合。数据库中有一些专用语:信息被收集在一个表(table)中,表中的信息被分为记录(record)和字段(field)。这些记录和字段与行和列相对应,有时这

些术语是可互换的。数据库中的数据通过给数据库发出一个查询(query)指令来访问。数据库有专门的程序来处理所有的查询和操作。这些被称为数据库引擎(database engines)。

有特色的是,一个数据存储由大量的行组成,而每一行都有数目绝对相同的字段。

这里是一个简单的数据库,它存储了有关于我的猫的一些详情:

| Name | Age | Sex | Color |
|----------|-----|-----|----------------|
| Pierre | 9 | M | Gray and white |
| Hergie | 16 | F | Black |
| Calloway | 2 | M | Gray |

在这个表中,列相当于字段而行相当于记录。

21.1.2 什么是记录集

记录集(recordset)是以前述格式组成的记录的集合。从一个大数据存储中提取出一个记录的集合并将它们作为一个记录集,这是很标准的操作。例如,在 cats 数据库中,可能感觉猫的年龄和颜色是对当前目的没有实质意义的,并且只有公猫的信息是所需要的。这样,就可能从前面的数据存储中创建一个限定记录集,它只包含如下信息。

| Name | Sex |
|----------|-----|
| Pierre | M |
| Calloway | M |

这是一个由两个记录组成的记录集。

甚至数据库中一个单独的记录也可以是一个记录集:

| Name | Age | Sex | Color |
|----------|-----|-----|-------|
| Calloway | 2 | M | Gray |

要注意的一点是这些仍然只是结构化的信息。第一个例子是一个有两个记录(record)的记录集,每个记录包含两个字段。第二个例子是一个有四个字段的单一记录。

21.1.3 浏览记录集

SQL (Structured Query Language, 结构化查询语言) 被开发以用来对记录集进行筛选。幸运的是,只要知道 SQL 的最一般的知识就可以做很有用的工作了。要掌握的关键要素就是迭代器 iterator, 它是一个在记录集记录中上下移动的指针。迭代器使用简单的命令移动。moveFirst 将把迭代器移动到第一个记录上, moveLast 将把迭代器移动到最后一个记录上, 而 move(indexnumber) 将会把迭代器移动到指定标号的记录上。当迭代器在某一个记录上时, 就可以运行 SQL 方法从多个字段中提取数据。通过在记录集中移动迭代器, 数据库引擎就可能实现分类、过滤功能和比较记录以构造出对数据不同视角的观察或构造不同的记录集。

经过了这些对数据库编程的简单介绍, 让我们看一下什么是真正的 DSO。

21.1.4 什么是 DSO

实质上,DSO 是一个在客户端的像数据库一样运行的对象。它可以接收按某种方式组织的信息,并使用自身的数据库引擎对它进行操作。与其他对象一样,它有一个有着自己特征和方法的接口。你可以使用这些方法在对象中对数据存储进行操作。

与其他对象一样,除非创建了一个实例,否则它是不能被使用的。由于我们正在讨论网页的技术,这项工作就通过使用 HTML object 元素来完成了。一旦一个 DSO 的实例被创建,在 HTML 页面中包含的小型数据引擎就启动了。对于任何的数据库来说,除非是被数据启动,它本身没有任何用处,每一个 DSO 在处理数据方面都有一些不同。要注意,对于不同类型的数据格式,有着不同类型的 DSO:

- 表格式数据控制(Tabular Data Control,TDC)用于结构化文本文件。
- 远程数据服务(原来的 ADC)用于 SQL 数据库中的记录集。
- XML 数据源用于 XML 文件格式的数据。

这些 DSO 中每一种都需要被它自身格式的数据启动。当我们在了解单独类型的 DSO 时我们将看看它们是怎么实现的。

一旦这些对象被数据启动,你就可以使用 SQL 语法的子集在多个记录中迭代,并在它们上执行任务。

21.1.5 DSO 和 XML

最初,这些对象只能应用于数据库和结构化文本文件。然而,随着 XML 的出现,用于 XML 数据存储的 DSO 被开发出来。这就使得给 XML 文件一个与记录集相类似的结构成为了可能。这一点在清单 21.1 中得到了验证。在这里 cats 数据库通过一个与简单记录集相类似的结构组织到 XML 文件中。

清单 21.1 简单的 Cats.xml 文件

```
<? Xml version="1.0"? >
<!--this is cats.xml-->
<cats>
  <cat>
    <name>Pierre</name>
    <age>9</age>
    <sex>m</sex>
    <color>Gray and white</color>
  </cat>
  <cat>
    <name>Hergie</name>
    <age>16</age>
    <sex>f</sex>
    <color>Black</color>
  </cat>
  <cat>
    <name>Calloway</name>
```

```
<age>2</age>
<sex>m</sex>
<color>Gray</color>
</cat>
</cats>
```

然而,与记录集中的简单的记录/字段结构相比,XML 文件可以包含更丰富更复杂的结构。一些 XML 文件有数层嵌套,这也就带来了问题。那就是你需要将他们按记录集对待。这些问题和他们的解决办法将在我们研究 XML DSO 时更详细的进行讲解。

21.2 远程数据服务和 DSO

对于记录来说,多种可利用的 DSO 类型有一个详细的分类。最简单的 DSO 是 TDC DSO,我将在稍后的“理解表格式数据控制”中复习如何对它进行使用。理解了 TDC DSO 之后,你就可以学习如何使用 XML DSO。

21.2.1 表格式数据控制(TDC)

TDC 是一种简单的 DSO,它提供对有界定文本文件的访问功能。这是 DSO 的最基本功能。下面是一些你可能需要使用 TDS DSO 的理由:

- 你有一个简单的数据集要显示,而它并不能保证符合 XML 的数据限制。
- 你需要离线浏览你的数据,或者你在客户端有一个脚本用来对数据进行不同的显示。通过 TDC 发送过来的文件可以缓存在客户端并可以在离线状态下读取。
- 你不希望客户端直接的访问你的数据库管理系统(DBMS)。你可以写一个简单的服务脚本,用来从你的数据库中提取数据并放入到一个有界定文本文件中。
- 你的数据存储方式可能是简单的文本文件。然而,在这种情况下,将你的数据按照 XML 格式存储当然要更好。

21.2.2 远程数据服务(以前的 ADC)

远程数据服务是适用于 IE4 的一个成熟的 DSO。RDS 使用 OLE-DB 或 ODBC 从数据库中获取数据。需要指出的是,使用 RDS 要包括如下条件:

- 你有数据保存在诸如 SQL Server、Microsoft Access 或 Oracle 这样的 OLE-DB 或与 ODBC 相适应的数据库管理系统(DBMS)中。
- 你希望使用 SQL 声明和语法来访问数据。
- 你希望你的客户端提供更新、插入、删除的功能。
- -你希望直接的、实时的访问数据。
- 你并不担心客户端会威胁你的数据存储的安全。

远程数据服务(RSO)是一个强大的成熟的对象。在这里我们将不进行更详细的介绍。要了解更多信息可以在 <http://msdn.microsoft.com/workshop/author/databind/datasource.asp> 中找到。

21.2.3 JDBC DataSource Applet

JDBC DataSource Applet 也是一个实用性的 DSO。它可以从位于 <http://msdn.microsoft.com/downloads/default.asp> 的 Microsoft 数据源对象库上下载到。在如下情况下,你可以考虑使用这个 applet:

- 你有数据保存在诸如 SQL Server、Microsoft Access 或 Oracle 这样的与 ODBC 相适应的数据库管理系统(DBMS)中。
- 你希望使用 SQL 声明指定数据。
- 你需要提供更新、插入、删除功能。
- 你更喜欢使用 JDBC 进行数据传输。

21.2.4 XML 数据源

XML 数据源 DSO 有两种风格:Java DSO,它可以作为一个 applet 使用并可以在 IE4 中运行;另一种是 C++ DSO,它被嵌入到 IE5 中。在本章,我们将详细的了解它们。

更多的信息可以在 <http://msdn.microsoft.com/workshop/author/databind/datasources.asp> 中查到,它还包含到其他页面的链接。

经过对 TDC DSO 的简要介绍后,在本章的其余部分中,我们将集中讲解 XML DSO——特别是结合进 IE5 的 C++ XML DSO。

21.3 理解表格式数据控制(TDC)

表格式数据控制最为容易理解。它访问简单的文本文件并循环遍历文件中的每一行。清单 21.2 是一个简单的文本数据存储,存储了有关于三只猫的详细资料。它以 cats.txt 为名字存储。要注意文本文件是如何做到包含与 cats.xml 和记录集存储的相同的信息的。

清单 21.2 cats.txt 数据文件

```
Name, Age, Sex, Color
Pierre, 9, m, Gray and white
Hergie, 16, f, black
Calloway, 2, m, gray
```

这里描述一个记录集,只用了四行文本。第一行(或者说记录)描述了每个字段的头。其余行表述了每一个字段的值。表格式数据控制 DSO 将把这个文本文件导入进 HTML 页面。由于它是一个 ActiveX 对象,object 元素被用来在 HTML 页面中调用 TDS DSO。

21.3.1 创建 TDC DSO

清单 21.3 在 HTML 客户端(IE4 或 IE5)上创建了 TDC DSO,并导入了 cats.txt 文件。

清单 21.3 object 标记以及用来创建 TDC DSO 元素的示例

```
<object id="cats" width="100" height="100"
  classid="clsid:333C7BC4-460F-11D0-0080C7055A83">
  <param name="FieldDelim" value=",">
  <param name="DataURL" value="cats.txt">
  <param name="UseHeader" value="true">
</object>
```

object 开始标记包含了所有创建 TDC DSO 所必需的信息。

```
<object id="cats" width="100" height="100"
  classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
```

在这里给出了一个 ID, 这样不论是当文本文件中的值被限制在 HTML 元素中还是当使用代码对文件进行访问时, DSO 都可以在稍后被引用。

得到 clsid 要满足的非常重要的一点是: 数字绝对正确。要注意那些数字是零而不是字符 O。这是一个十六进制数。高和宽是需要设置的, 它们的数字可以任意设置。

21.3.2 导入文本文件

param 元素用来设置 TDC DSO 的值和属性。FieldDelim、DataURL 和 UseHeader 是 TDC DSO 的全部属性。

FieldDelim 属性设置在文本文件中, 用于区分不同字段的字符。

```
<param name="FieldDelim" value=",">
```

在这里, 逗号界定字符串。

DataURL 属性是将被导入到 TDC DSO 中的引用文件的 URL 地址。

```
<param name="DataURL" value="cats.txt">
```

UseHeader 属性告诉 DSO 文件的第一行用来作为所有数据绑定的标题。

```
<param name="UseHeader" value="true">
```

这些元素值可以通过代码设定, 我们将在稍后探讨 XML DSO 时, 再涉及这部分知识。

注意: 注意一个对象的实例是如何被 object 元素创建的, 以及这些对象如何导入文本文件中的数据。如果要将另一个文本文件下载到客户端, 就要创建一个 fresh 对象。我们不能凭空作这些事情。事实上, 需要在一个新的页面上创建一个新的对象。显而易见, 这就意味着创建数据对象的代码必须由作者来包括在初始的 HTML 页面上。

21.3.3 TDC DSO 的数据绑定

一旦 DSO 被创建, 字段中的不同的值就要被绑定在不同的 HTML 元素中——从而允许他们被显示。

数据绑定的概念对 DSO 来说是全新的。它的思想就是记录的字段值可以与 HTML 元

素相关,它可以显示当前记录(由当前迭代器位置决定)的字段值。如果记录索引发生变更,HTML 元素的内容或值就会随之改变。这个功能是通过使用元素自身的 `datasrc` 和 `datafld` 属性实现的。`datasrc` 属性详述了元素要被绑定的 DSO,而 `datafld` 属性选取字段。它们的语法如下:

```
<element name datasrc="# id of DSO" datafld="field name from file">
```

要注意,DSO 的 ID 一定要通过散列符号预定义。`datafld` 属性把它的值作为它想要绑定的字段的标题。

并不是所有的 HTML 元素都可以作为附着 `datafld` 属性的“吊钩”。例如,TD 元素就不能接受这个属性。如果你想要使用这些值填充表格(如清单 21.4 显示),这个属性就必须附着在其他的元素上。在这里,我们是用 `input` 元素。

清单 21.4 显示了一个 TDC DSO 是如何被用来提供一系列数据的。每一个记录的字段都被绑定在一个文本域中,一些简单的脚本被用来实现在记录集中的上下移动。

清单 21.4 使用 TDC DSO 提供记录

```
<form>
<table border="1">
<caption>Cats</caption>
<tr>
<th>Name</th>
<td><input type="text" datasrc="# cats" datafld="name" /></td>
</tr>
<tr>
<th>Age</th>
<td><input type="text" datasrc="# cats" datafld="age" /></td>
</tr>
<tr>
<th>Sex</th>
<td><input type="text" datasrc="# cats" datafld="sex" /></td>
</tr>
<tr>
<th>Color</th>
<td><input type="text" datasrc="# cats" datafld="color" /></td>
</tr>
</table>
<table border="0">
<tr>
<td align="center">
<input type="button" value="first" name="buttonFirst" />
<input type="button" value="<" name="buttonPrevious" />
<input type="button" value=">" name="buttonNext" />
<input type="button" value="Last" name="buttonLast" />
</td>
</tr>
</table>
```

图 21.1 显示了上述代码产生的结果。要注意,在使用 `input` 元素的情况下,`datafld` 的值

被赋成了元素的值。

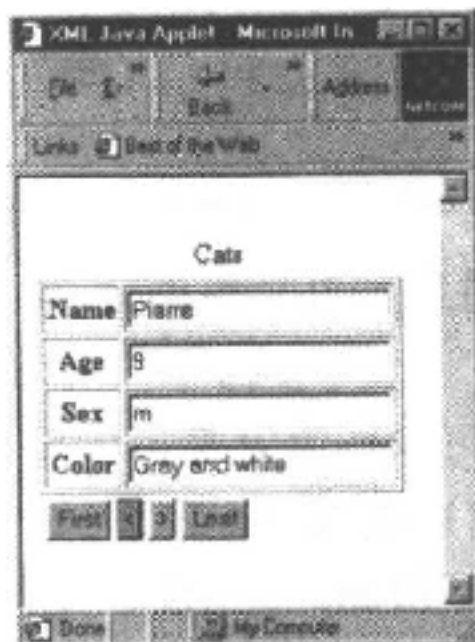


图 21.1 对清单 21.4 第一次调用的情况

一旦文本文件被 HTML 元素绑定,我们可以使用标准的 SQL 调用来遍历文件。

这里使用的语言是 VBScript,同时也使用了 JavaScript。现在 VBScript 是专门在 Microsoft 浏览器中使用的,因此使用更通用的 JavaScript 的价值就降低了。实际上,使用 VBScript 主要是因为可以使用[Buttonname]_onclick()语法。

清单 21.5 显示了使用 SQL 语言遍历被 TDC DSO 绑定的记录集脚本。

清单 21.5 遍历记录集

```
<script language="VBScript">
  sub buttonFirst_onclick()
    cats.recordset.moveFirst
  end sub

  sub buttonPrevious_onclick()
    if not cats.recordset.bof then cats.recordset.movePrevious
  end sub

  sub buttonNext_onclick()
    if not cats.recordset.eof then cats.recordset.moveNext
  end sub

  sub buttonLast_onclick()
    cats.recordset.moveLast
  end sub
</script>
```

代码中没有什么太困难的地方,但下面的几点一定要注意:

- cats 指定 TDC DSO 的 ID。
- TDC DSO 的 recordset 属性是一个有自主权的对象,有着自己的属性和方法。这些都通过使用 cats.recordset 访问。
- SQL 语法用来在多个记录间进行遍历。

- Recordset 对象的方法,绝大部分是 SQL 语法。
- 记录集对象的文件结束(eof)和文件起始(bof)属性在记录集中前后移动之前会进行检查。如果检查没有作,当移动超出记录集结尾的时候就会返回错误。

21.3.4 使用脚本和 SQL 语法

有关于 DSO 最大的好处就是它不仅可以在客户端遍历记录集,它还可以在客户端修改、添加、删除记录。修改后的记录可以迅速的传回到服务器上。当然,这还要在服务器端编写代码以将记录并入到数据源中。

就如上面例子所看到的,在 recordset 对象中起用了多个 SQL 方法。实际上,所有的 DSO 都使用 SQL 方法作为他们的 recordset 对象特征的一部分。下面是一些你用得到的 recordset 属性和方法(如表 21.1 所示)。

表 21.1 recordset 对象的属性和方法

| 属性 | 行为 |
|---------------|---|
| MoveFirst | 将迭代器移动到第一个记录 |
| MoveLast | 将迭代器移动到最后一个记录 |
| MoveNext | 将迭代器移动到下一个记录 |
| MovePrevious | 将迭代器移动到前一个记录 |
| Eof | recordset 属性,如果迭代器到达文件结束处,其值为真 |
| Bof | recordset 属性,如果迭代器到达文件开始处,其值为真 |
| move(integer) | 将迭代器移动到作为参数传入的记录代号指代的记录上。如果数值大于记录的数目,将会抛出一个错误 |
| Delete | 删除当前记录 |
| AddNew | 在文件尾加入新的记录。在加入一个新的记录之前,迭代器通常会移动到文件的尾部 |
| CancelUpdate | 从它调用开始禁止所有的对记录集的更新。这个属性的实现在 DSO 中是十分诡异的。不要使用它 |
| Fields() | 当传入了字段的名字或是索引号时,就返回字段的值。如果索引号超出了记录的结束号(或者没有相应的字段名),就会返回一个错误 |

21.3.5 DSO 事件

DSO 同样有一整套的事件。有些事件通过浏览器驱动,而有一些是由 DSO 对象本身驱动的。典型的浏览器事件在卸载页面和数据经过 DSO 时发生。如果一个记录元发生了改变,或者有信息被加入或是删除,或者是在记录中加入了一行,DSO 初始化事件就会启动。在 Microsoft 网站上对这些事件及其使用的语法有详细的文件可以查阅。

在讨论稍为复杂一些的 XML DSO 之前,让我们看一个实例。它显示了如何在典型位置启用脚本。清单 21.6 将 cats.txt 加载到 DSO 中,并允许你遍历记录,修改单独的元和添加删除记录。

清单 21.6 简单的 TDC DSO

```

<html>
<head>
<title>XML Java Applet</title>
</head>
<body bgcolor="white">
<object id="cats" width="100" height="100"
  classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
  <param name="FieldDelim" value=",">
  <param name="DataURL" value="cats.txt">
  <param name="UseHeader" value="true">
</object>
<form>
<table border="0">
<tr>
  <td align="center" <input type="text" value="" name="Name" datafld="name"
datasrc="# cats" />
  </td>
</tr>
<tr>
  <td align="center">
    <input type="text" value="" name="Age" datafld="age" datasrc="# cats" />
  </td>
</tr>
<tr>
  <td align="center">
    <input type="text" value="" name="Sex" datafld="sex" datasrc="# cats" />
  </td>
</tr>
<tr>
  <td align="center">
    <input type="button" value="First" name="buttonFirst" />
    <input type="button" value="<" name="buttonprevious" />
    <input type="button" value=">" name="buttonNext" />
    <input type="button" value="Last" name="buttonLast" />
  </td>
</tr>
<tr>
  <td align="center">
    <input type="button" value="Add record" name="buttonAdd" />
    <input type="button" value="Delete" name="buttonDelete" />
    <input type="button" value="Change" name="buttonChange" />
    <input type="button" value="Show file" name="buttonShow" />
  </td>
</tr>
</table>
</form>
<script language="VBScript">
sub buttonDelete_onclick()

```

```

cats.recordset.delete
end sub

sub buttonChange_onclick()
    cats.recordset.fields(0)=document.forms(0).Name.Value
    cats.recordset.fields(1)=document.forms(0).Age.Value
    cats.recordset.fields(2)=document.forms(0).Sex.Value
    cats.recordset.fields(3)=document.forms(0).Color.Value
end sub

sub buttonAdd_onclick()
    cats.recordset.addNew
    cats.recordset.moveLast
    ' cats.recordset.fields(0)=document.forms(0).Name.Value
    ' cats.recordset.fields(1)=document.forms(0).Age.Value
    ' cats.recordset.fields(2)=document.forms(0).Sex.Value
    ' cats.recordset.fields(3)=document.forms(0).Color.Value
    alter("Fill in the new values, and click the change button") end sub

    sub buttonShow_onclick()
dim recordStr
        recordStr=""
        cats.recordset.moveFirst
        do until cats.recordset.eof=true
            recordStr=recordStr & cats.recordset("name") & ", "
            recordStr=recordStr & cats.recordset("age") & ", "
            recordStr=recordStr & cats.recordset("sex") & ", "
            recordStr=recordStr & cats.recordset("sex") & ", "
            recordStr=recordStr & cats.recordset("color") & chr(13) & chr(10)
            cats.recordset.moveNext
        loop
        alert(recordStr)
    end sub

sub buttonFirst_onclick()
    cats.recordset.moveFirst
end sub

sub buttonPrevious_onclick()
    if not cats.recordset.bof then cats.recordset.movePrevious
end sub

sub buttonNext_onclick()
    if not cats.recordset.eof then cats.recordset.moveNext
end sub

sub buttonLast_onclick()
    cats.recordset.moveLast
end sub
</script>
</body>
</html>

```

图 21.2 显示了在 IE5 中加载这个文件的结果。

加载 TDC DSO 对象并给它加载一个文件的代码我们已经解释过了。上下遍历记录集的代码,我们也已经讲解过了。这里新的内容就是删除、添加和修改记录的代码。

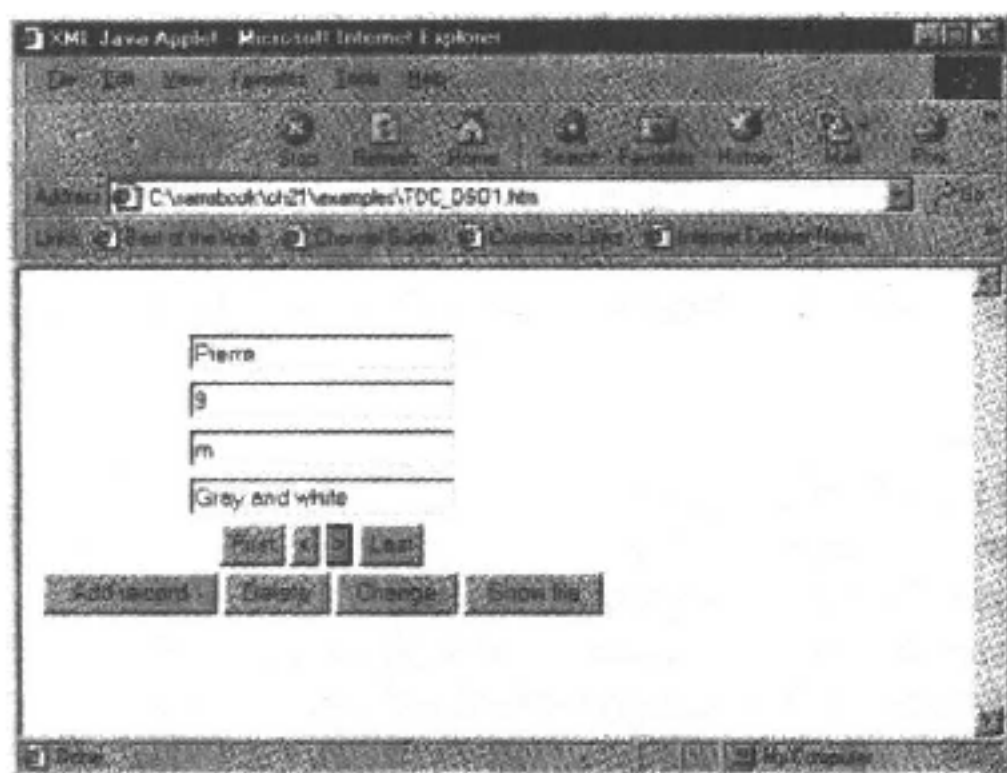


图 21.2 对清单 21.6 第一次调用的情况

删除记录

删除记录非常的简单。只需简单的调用 recordset 对象的 delete 方法,就像这样:

```
cats.recordset.delete
```

我们还可以通过 move()方法设置迭代器选取记录,并将所选记录删除。例如:

```
cats.recordset.move(2)
cats.recordset.delete
```

这段代码删除第三个记录(记录索引从 0 开始)。

修改记录

修改记录非常容易。只是给 recordset 对象的 fields()属性赋予新的值就可以了。前面的例子中,相应的文本域的值就是被简单的赋予一个值。

提示:在 DSO 中,我们处理过许多值。value 指的是文本域的内容,它也与 input 元素中的名为 value 的属性的值完全相同。当 input 元素被 DSO 绑定时,这个值也与 DSO 中的值完全相同。无论哪一个值发生了变化,变化将在所有的三个地方得到反应。

```
cats.recordset.fields(0)=document.forms(0).Name.Value
cats.recordset.fields(1)=document.forms(0).Age.Value
cats.recordset.fields(2)=document.forms(0).Sex.Value
cats.recordset.fields(3)=document.forms(0).Color.Value
```

在这里,我们使用 field 属性的索引,而使用字段名也同样非常的简单。例如:

```
cats.recordset.field("name")="Charlie"
```

它将会把值 Charlie 赋给 name 字段。

在 IE5 中,当一个字段被绑定成一个文本域时,改变文本域中的值就会自动的修改

TDC DSO 中的值。如果文本域没有与 DSO 发生绑定,我们就要使用子程序来实现这个功能。

提示:如果数据与 DSO 方面的绑定被使用,只要改变绑定文本域中的文本就将会自动的改变 DSO 中的数据。然而,有时这会导致无法预知的结果。相比依赖于数据绑定,我更喜欢编写代码在文本域中显示相应的值:

```
document.forms([integer]).[input tb name].value =  
    [DSO id].recordset.fields([integer])
```

添加记录

addNew 方法将自动的在记录集末尾追加一个新的记录:

```
cats.recordset.addNew  
cats.recordset.moveLast
```

迭代器将移动到新的记录上,那里将是一个空域。在前面的例子中,一个提示被用来要求读者填写这些值。

构造文件

所有的对记录集的变更完成了以后,记录集就会以一个 HTTP 数据流的形式传送给服务器。当然,一定要在服务器上编写脚本,用来切分字符串并将信息写入数据库。下面的代码显示了在客户端组合字符串的任务通常是怎样完成的。

```
dim recordStr  
recordStr=""  
cats.recordset.moveFirst  
do until cats.recordset.eof=true  
    recordStr=recordStr & cats.recordset("name") & ", "  
    recordStr=recordStr & cats.recordset("age") & ", "  
    recordStr=recordStr & cats.recordset("sex") & ", "  
    recordStr=recordStr & cats.recordset("color") & chr(13)&chr(10)  
    cats.recordset.moveNext  
loop
```

注意:chr(13)&chr(10)是在 VB 方法中用来描述代码另起一行的。chr()函数以一个长整数为参数,这个长整数是统一字符编码(Unicode)标准码。这个函数返回适当的字符。13 是新的—行的(Unicode)标准码号,而 10 是回车符的 Unicode 标准码号。

简单的循环将所有的记录读入到一个字符串中。在实际应用中,我们通常把这个值赋给一个隐藏的 input 元素中,它将会被发还给服务器。

21.3.6 数据与 HTML 表格(table)绑定

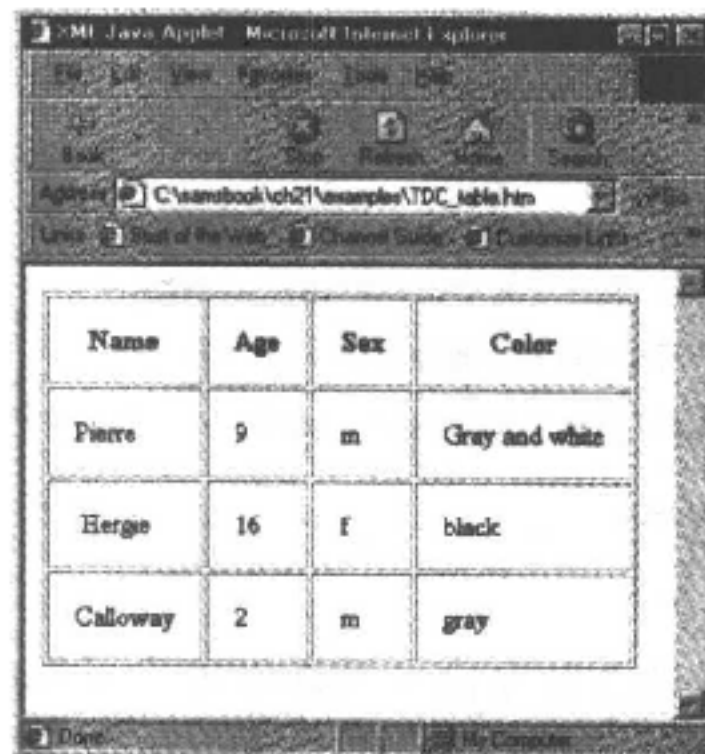
DSO 一个非常酷的特色就是他们允许数据与 HTML table 元素发生绑定的方法。当一个 table 元素被绑定时,我们就不需要为记录集中的每一个记录的单独行的处理编写代码

了。DSO 将自动的对每一条记录进行处理。例如,将清单 21.7 的代码加入到 HTML 文件中。这个 HTML 文件已经创建了一个 TDC 并在其中加载了 cats.txt。

清单 21.7 数据与 HTML 元素绑定

```
<table datasrc="#cats" border="1" cellpadding="15">
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Sex</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td><xspan datafid="name"></span? </td>
<td><span datafid="age"></span></td>
<td><span datafid="sex"></span></td>
<td><span datafid="color"></span></td>
</tr>
</tbody>
</table>
```

图 21.3 显示了这段代码的效果。



The screenshot shows a web browser window with a table containing cat data. The table has four columns: Name, Age, Sex, and Color. The data rows are: Pierre (Age 9, Sex m, Color Gray and white), Herge (Age 16, Sex f, Color black), and Calloway (Age 2, Sex m, Color gray).

| Name | Age | Sex | Color |
|----------|-----|-----|----------------|
| Pierre | 9 | m | Gray and white |
| Herge | 16 | f | black |
| Calloway | 2 | m | gray |

图 21.3 清单 21.7 的实现效果

DSO 引擎将自动的遍历记录,并为记录集中的每一个记录在 HTML 表的每一行上填入适当的值。

TDC 是最简单的 DSO。所有 DSO 的基本概念都通过使用上述的 TDC DSO 进行了示范。如果你理解了这些基本的概念,你就可以理解 XML DSO 控制的实现理论。XML 控制不仅像其他的控制一样做着同样的工作,他们之间也同样非常的通用,因而他们可以使用 XML 提供的丰富的结构。

21.4 XML DSO

这里有两种实用的 XML DSO。一种是 Java applet,它可以在 IE4 和 IE5 上工作,通过使用 applet 元素进行加载。第二种 XML DSO 只适用于 IE5 浏览器。它是使用 C++ 编写的 ActiveX 对象。Java applet 和 ActiveX 控制都可以将 XML 数据与 HTML 元素进行绑定。C++ DSO 内嵌有对文件对象模型(Document Object Model,DOM)的支持。实际上,它支持 W3C DOM 的所有属性和方法。

C++ DSO 也可以在内部 DTD 中确认 XML 文件。

21.5 XML Java DSO

Java DSO 是一个 Java applet。同样的,它也必须使用 applet 元素嵌入到 HTML 页面中:

```
<applet
  code="com.ms.xml.dso.XMLDSO.class"
  id="xmldso"
  width="0"
  height="0"
  mayscript="true">
  <param name="URL" value="cats.xml">
</applet>
```

code 属性指定代码实现所在的包。mayscript="true" 属性允许在客户端运行脚本。Param 元素指定了数据的位置。XML DSO 从这个位置接收 XML,解析它并为页面上的绑定元素提供数据。

21.6 XML C++ DSO

IE5 有一种由 C++ 编写的 DSO。它不仅支持数据与 HTML 元素的绑定,同时它还创建了一个加载 XML 文件的 DOM。XML 文件就可以使用脚本进行访问。C++ XML DSO 可以使用 object 元素直接实例化,或者它也可以经由 XML 岛(稍后将详细讲述 XML 岛)来调用。首先我们看一下第一项选择。

21.6.1 实例化 C++ XML DSO

一个 XML DSO 实例通过使用 OBJECT 元素被创建。下面是实例化对象的语法:

```
<OBJECT width=0 height=0
  classid="clsid:550dda30-0541-11d2-9ca9-0060b0ec3d39"
  id="xmldso">
</OBJECT>
```

要把 XML 文件加载到这个对象中,你必须使用脚本,这胜于设置元素参数的值。一个

XML 文件可以通过调用扩展文件或是作为内嵌 XML 文件被加载到对象中。这两种方案需要不同的脚本技术。

21.6.2 加载扩展文件

这里是一个 JavaScript 的例子。它将加载一个名为 cats.xml 的扩展文件：

```
<SCRIPT for=window event=onload>

    var doc = xmldso.XMLDocument;
    //doc.validateOnParse = false
    //doc.resolveExternals = false

    doc.load("cats.xml");
    if (doc.parseError != 0)
    {
        HandleError(doc);
    }
</SCRIPT>
```

这个过程有两个部分。第一部分包括创建一个 DOM 对象。这是通过使用 DSO 的 XMLDocument 属性实现的：

```
var doc = xmldso.XMLDocument;
```

一旦 DOM 对象被创建,就可以使用 DOM 对象的 load 方法加载一个文件：

```
doc.load("cats.xml");
```

21.6.3 验证 XML

XML DSO 有一个内置的验证解析器(C++中的 Microsoft 解析器)。DSO 的默认行为是确认所有的文件都已经被加载了,并解析所有的外部实体。如果由于某种原因,你不希望验证文件是否已被加载,或者你不希望解析所有的外部实体,你可以关掉这些特性(例如,通过 Internet 传送 DTD 和实体是不值得的甚至是不可能的)。

关闭验证模式的语法如下：

```
[DOM object name].validateOnParse = false
```

避免解析外部实体的语法如下：

```
[DOM object name].resolveExternals = false
```

解析器仍然会验证文件是结构完好的。如果不这样的话,就会返回一个错误。这些特征必须在文件加载之前,在 DOM 对象的实例中被修改!在前面的例子中,我们提供了做这件事情的代码,但在这里被注释掉了。

如果由于某些原因,文件加载失败,它将返回与解析器同样的错误代码。这些错误代码(在第 14 章中详述)被 parser 对象的 parseError 特征检测。如果 parseError 返回了一个错误,它将会下传一个错误处理函数。

21.6.4 加载一个内嵌文件

要加载一个内嵌文件,你需要能够通过代码引用它。做这件事情的一个简单的方法就是在 object 元素中放置文件本身:

```
<OBJECT width=0 height=0
  classid="clsid:550dda30-0541-11d2-9ca9-0060b0ec3d39"
  id="xmldso">
  <favorites>
    <favorites>
      <name>HTML Writers Guild</name>
      <url>http://www.hwg.org</url>
    </favorites>
  </favorites>
</OBJECT>
```

现在,文件将以 XML DSO 对象的 altHtml 属性出现。

这项技术在实际应用中非常的普遍。这样的 XML 文件将通过服务器从数据存储中直接被创建。然后,它就会作为 HTML 数据流的一部分传递给客户端。

为了把这个文件加载到我们的 DSO 中,我们要使用 DSO 的 XMLDocument 属性创建一个 DOM 对象:

```
<SCRIPT for=window event=onload>
  var doc = xmldso.XMLDocument;
```

现在,DOM 对象的 loadXML 方法可以用来将任何 XML 字符串加载到 DOM 对象中:

```
doc.loadXML(xmldso.altHtml);
```

altHtml 属性包含放置在 object 元素中的字符串(注意,xmldso 是这个元素的识别 ID)。

此外,如果这有一个错误,这个错误就会传入到一个错误处理函数。在这里,一个简单的警告将通告这个错误类型:

```
if (doc.parseError != 0)
{
  HandleError(doc);
}
alert(doc.documentElement.nodeName)
function HandleError(x)
{
  alert(x.parseError.reason)
}
</SCRIPT>
```

由 Microsoft C++ 中的解析器产生的错误代码已在第 14 章“用 C++ 解析 XML”中进行详细的讲解。

在IE5中还有一种更好的方法加载XML——使用XML岛。这是一个实例化XML DSO的速成方法。引擎与刚才描述的完全相同。

21.7 使用XML岛

XML岛提供了一种实例化XML DSO、创建DOM对象并加载XML文件的简单方法。一个XML岛通过使用xml元素创建。这个元素是Microsoft专有的HTML元素。

警告:注意这是一个HTML元素,而不是XML元素。实际上,这个元素在XML中是违法的,因为它以被禁止的字符串xml开头。

警告:XHTML是作为XML应用的一个HTML的改进版。因此,xml元素在XHTML中也同样是被禁用的!然而,让Microsoft操心去修改它是非常令人怀疑的事情。如果你使用XHTML,在XML解析器中就会抛出一个错误。

这个元素的重要属性是id和src。前者识别XML DSO,DSO是在浏览器看到该元素时由浏览器自动生成的,后者引用一个外部XML文件。让我们来看一看,如何使用xml元素加载一个外部XML文件和内嵌的XML字符串。然而,在这之前,我们需要知道,我们使用XML岛时,文件已经被自动确认,外部实体也将被完成解析。

21.7.1 在XML岛中验证XML

要注意的是,XML DSO默认的行为是在XML文件被加载时,验证文件和解析外部实体。

关掉这些属性的惟一方法就是通过脚本实现。当然,一定要在XML DOM对象被实例化后,开始加载文件前,将这些属性关掉。由于XML岛方法实现了这两个功能,因此在使用岛方法实例化XML DSO时没有可能将这些属性关掉。如果你希望关掉这些属性,你需要利用更多的使用object元素预定义的详细的方法。

21.7.2 加载嵌入XML

使用XML岛方法加载嵌入XML非常的简单。下面的HTML代码实现了这个功能:

```
<xml id="greetingisle">
  <greeting>hello XML Islands</greeting>
</xml>
```

这几行简单代码不仅创建了一个XML DSO,它们还将封装在元素标记中的XML字符串加载进去。要注意,只有当字符串是结构良好并且是有效的XML时,它才能够被加载。如果不是,字符串将被丢弃。你需要使用脚本来确定它是否被丢弃。

XML岛的错误检查

为了要检查错误,我们必须使用脚本来创建一个DOM对象并检查错误。这里说明了在JavaScript中这是如何被实现的:

```
var oXMLdoc
```

```
oXMLdoc=greetingisle
if (oXMLdoc.parseError! =0)
{
    alert(oXMLdoc.parseError.reason)
}
```

你必须首先创建一个脚本对象。将它加载到 DOM 对象中。这个 DOM 对象是 DSO 通过引用 xml 元素的 ID 创建出来的。

下一步,看一下 DOM 对象是否返回了一个错误。如果没有,XML 文件就被顺利的加载了。

21.7.3 加载外部文件

加载一个外部文件甚至比加载一个内嵌文件更简单:

```
<xml id="cats" src="cats.xml"></xml>
```

这个简单的 HTML 代码创建了一个有 DOM 对象的 XML DSO,并加载了文件 cats.xml。在这个文件被加载的同时,cats.xml 被确认并且所有的外部实体被扩充。

注意:由于这是一个 HTML 标记,下面的语法是不可接受的:

```
<xml id="cats" src="cats.xml"/>
```

提示:使用 XML 岛也可能创建一个简陋的验证解析器。只需加载文件并跳过所有的错误。

21.8 编写 XML DSO

本质上,XML DSO 使用了两套脚本接口:支持 SQL 子集的 SQL 接口和支持所有的 Level1 DOM(请参阅第 15 章“使用文件对象模型(DOM)”)的 W3C XML DOM 接口。

21.8.1 使用 SQL

moveNext、movePrevious、moveLast 和 moveFirst 方法都可以支持,但语法与 TDC DSO 中的有少许的不同。在 TDC DSO 中,没有参数的方法后跟的插入语是随意的。也就是说,下面是可以被接受的,并且它们将把迭代器移动到下一个记录集:

```
cats.recordset.moveNext
```

在 XML DSO 中,方法后必须插入包含空的圆括号。例如:

```
cats.recordset.moveNext()
```

下面的 VBScript 将要遍历所有的记录并显示 cats.xml 中字段的值:

```
do until cats.recordset.eof=true
    for i=0 to 2
        document.write(cats.recordset.fields(i) & ",")
    next
```

```
document.write("<br />")  
cats.recordset.movenext()  
loop
```

使用 SQL 语法和脚本就提供了一种遍历简单结构 XML 文件的简便方法。

注意：一个简单结构 XML 文件类似于我们的 cats.xml 文件。所有的根元素的子树都有这同样的名字，在其中也只有一层嵌套。XML DSO 将丢根元素并将文件的其余部分作为记录集对待。根元素的子树与单独记录相应，嵌套元素的标记名称作为记录集的字段。

21.8.2 使用 DOM

由于 XML DSO 创建了一个 DOM 对象，这就使得使用所有的 DOM 接口属性和方法访问 XML 文件成为了可能。实际上，使用 DOM 你可以做使用 SQL 语法所能做的所有事情。下面的 VBScript 将打印出 cats.xml 中的猫的名字：

```
dim catslist  
set catslist =cats. documentElement. ChildNodes  
for i=0 to catslist. length-1  
document.write catslist. item(i). firstChild. nodeValue+"<br />"  
next
```

DOM 在第 15 章中已经进行了详细的阐述，因此在这里不在对它进行讨论。

21.9 XML 数据绑定

在简单结构文件中，数据绑定的工作与它对任何一个数据集所做的完全相同。下面的代码将要产生一个与图 21.3 所示基本相同的结果（只不过省去了标题）。

```
<xml id="cats" src="cats.cml"></xml>  
<table datasrc="#cats" border="1">  
<tr>  
  <td><span datafld="name"></span></td>  
  <td><span datafld="age"></span></td>  
  <td><span datafld="color"></span></td>  
</tr>  
</table>
```

要注意元素名是如何被用于数据与 HTML 元素绑定的，以及 xml 元素的内容是如何作为字段的值对待的。

我们可以将输入文本域与所有的其他元素用同样的方式进行绑定。然而，当 XML 文件变得不再简单时，事情就变得更加复杂了。当有多于一级的嵌套时将会发生什么事情？

21.10 复杂 XML 的数据绑定

清单 21.8 是一个修改过的 cats.xml。这里有两方面的改变：每个 cat 元素都被给出了一

个 names 属性;name 元素被再分成两个嵌套的元素。这个文件不再是一个可以与记录集进行简单映射的简单结构 XML 文件。

清单 21.8 深嵌套 XML 文件:Cats2.xml

```
<? xml version="1.0"?>
<!--this is cats2.xml -->
<cats>
  <cat names="Pierre">
    <name>
      <givenname>Pierre</givenname>
      <nickname>PiPi</nickname>
    </name>
    <age>9</age>
    <sex>m</sex>
    <color>Gray and white</color>
  </cat>
  <cat name="Hergie">
    <name>
      <givenname>Hergie</givenname>
      <nickname>Beautiful</nickname>
    </name>
    <age>16</age>
    <sex>f</sex>
    <color>Black</color>
  </cat>
  <cat names="Calloway">
    <name>
      <givenname>Calloway</givenname>
      <nickname>Baby</nickname>
    </name>
    <age>2</age>
    <sex>m</sex>
    <color>Gray</color>
  </cat>
</cats>
```

这个文件如何与 HTML 元素绑定呢?我将在下一章进行讲解。

21.10.1 绑定属性

如果 name 元素现在还是与一个 HTML 元素绑定,什么都不会显示出来。甚至如果 name 包含 PCDATA 这样的元素,这仍然是个问题。这个的原因主要是 XML DSO 只是为那些包含 PCDATA 的元素的绑定而编制的。

然而,一个 names 属性被加入到了第一元素级别——cat 元素,且 XML DSO 将会将它解释为记录中的一个字段。让我们看一下下面的代码:

```
<xml id="cats" src="cats2.xml"></xml>
<table datasrc="#cats" border="1">
```

```

<tr>
  <td><span datafld="names"></span></td>
  <td><span datafld="age"></span></td>
  <td><span datafld="color"></span></td>
</tr>
</table>

```

四个字段中的三个与元素进行绑定,包括被属性描述的 names 字段。这个代码的结果如图 21.4 所示。



The screenshot shows a web browser window titled 'XML Islands - Microsoft In...'. The browser's address bar shows 'http://www.xml-islands.com/'. The main content area displays a table with three rows of cat data. The table has three columns: Name, Age, and Color. The data is as follows:

| | | |
|----------|----|----------------|
| Pierre | 9 | Gray and white |
| Herge | 16 | Black |
| Calloway | 2 | Gray |

图 21.4 使用 XML DSO 存储猫的细节的代码如清单 21.8 所示,本图是该 XML 文件代码的输出情况

21.10.2 绑定嵌套的元素

我们如何绑定 givenname 和 nickname 元素的值? 这个更加困难。实际上,我们只能对第一个 cat 元素进行绑定。

看一下下面的代码:

```

<table datasrc="#cats" datafld="name" border="1">
<tr>
  <td><span datafld="givenname"></span></td>
  <td><span datafld="nickname"></span></td>
</tr>
</table>

```

HTML table 元素与 name 元素进行绑定。如果能够显示所有的 givenname 和 nickname 元素就好了,但这不大可能。图 21.5 显示了代码所产生的结果。

只有第一个 cat 元素中的元素被显示出来。然而,这并不意味着显示所有的元素是不可能。这只能意味着,它们一次只能显示一个。

如果一个记录控制被加入在上述的代码中,就有可能遍历数值并独立的显示它们。按钮栏的代码与使用 TDC DSO 来做的没什么不同。



图 21.5 使用 XML DSO, 显示清单 21.8 所示的 XML 文件的相应部分

```

<form>
<table>
  <td align="center">
    <input type="text" value="" name="tb1" /><br />
    <input type="button" value="First" name="buttonFirst" />
    <input type="button" value="<" name="buttonPrevious" />
    <input type="button" value=">" name="buttonNext" />
    <input type="button" value="Last" name="buttonLast" />
  </td>
</table>
</form>

```

用户的接口如图 21.6 所示。



图 21.6 浏览记录

子程序之间只有一些区别, 因为代码被加入的目的是用来使用 names 属性的值填充文本域的值。这是 buttonFirst 子程序的代码:

```

sub buttonFirst_onclick()
  cats.recordset.moveFirst
  document.forms(0).tb1.value=cats.recordset("names")
end sub

```

这里显示的技术允许 XML 数据与一个 HTML 元素绑定, 不论它的嵌套有多深。

21.11 总 结

虽然 DSO 只能在 IE5 中完全工作(在 IE4 中只支持更小的范围),但它们仍然非常的有用——特别是在一个内部网情况下。在这种情况下,程序员可以控制客户端使用者的浏览器类型。

总而言之,DSO(独特的 XML DSO 同时支持 SQL 语法和 DOM)提供了一个非常强大的工具,用来对客户端的数据进行操作和分类。

第 22 章 使用 XSL 模式和 XQL

从最简单的层次来说,XML 查询可以被认为是对文件内容的查询。更深一步来讲,它可以被认为是一种返回文件信息的方法。显而易见,它与 SQL 类似,SQL 语言用于查询随机访问数据库管理系统(Random Access DataBase Management Systems,RADBMS)。一个 SQL 查询返回一个新的记录集,该记录集从数据库中创建并包含具有所需要信息的记录。一个 XQL 查询将会返回一个新的,从旧文件中构造出来的结构良好的 XML 文件。本章将讲解 XML 是如何与传统意义上的数据库的数据集相区别的,也就是 XQL 为什么与 SQL 不同。我们还将接触可利用于 XML 的初始查询语言。这些包含内容如下:

- XQL。一种由当前领导软件企业的代表设计的新的语法——实际上,它是 XSL 工作的继续。
- XSLT 模式。XSL 使用 XQL 的子集选取要转换的节点。
- Xpointer。Xpointer 语法可以把特殊的节点从文件中隔离出来。它遍历文件并将文件的一处进行隔离,而它实际上的设计并不返回任何值。在本章,我们将不讨论 Xpointer(请参看第 19 章“使用 XLink 和 XPointer 链接文件”)。
- W3C DOM。使用 DOM 和脚本,我们就可能返回包含任意描述的任意节点的集合。
- 自主(hand-rolled——自己动手)函数。XML 的结构类型使它可以很容易的使用程序或脚本语言来编写简单的函数完成搜索。这些函数可以使用所讨论的语言本身的字符串处理方法。

本章将集中讲解 XQL 的语法,同时我们还要大概的学习一下如何使用 DOM 和自编函数。

22.1 查询语言的需求

22.1.1 为什么要有 XML 查询语言

XML 文件是可被用于数据存储或作为一般文件的结构化文件。有一些文件高度的结构化,有着几层嵌套和严格的层次。而另外一些只包含着一些初始的标题。无论如何,当一个 XML 文件作为数据被运用时,你就需要访问这些数据。为了在不同的操作平台上完成这项任务,同时还要维护 XML 作为真正互操作性语言的承诺,你需要一种一致的语法在各种类型的应用程序中对数据进行查询。

22.1.2 作为数据存储的 XML

显然,SQL 与 XQL 最为类似。为了理解为什么 XQL 要与 SQL 不同。我们需要简单的了解一下数据库与 XML 作为数据存储的不同。

数据库通过简单的、定义的方式存储数据。每一组相关的资料被存入记录,每一个记录有着固定数目的字段,每一个字段包含着记录的一部分。

XML文件也可以用上述方式构造,它还可以通过元素的嵌套包含更深层次的内涵。与数据库中固定数量的字段相对,XML文件通常有不定数量的子元素。

在记录集中,记录的次序是没有意义的。当然,次序没有意义是RADBMS的理论所需要的。记录可以被记录的更容易搜索——使用名为索引的过程——但是它们所产生出来的次序并没有传达语义上的意义。这对XML就没有问题。例如,医药记录中operation元素的次序通常是传达着一定意义的。它通常是操作执行的次序。在记录集中,这种信息只有通过查找date字段才能够得到。

当这些因素都要考虑到时,很明显XQL就必须拥有SQL所不能提供的下列功能:

- 它必须能够索引同属元素的次序。
- 它必须能够操作绑定数目的字段。
- 它必须能够描述父/子、祖先/后代这样的关系。

因此,你应该知道了,什么时候你应该用XML?而什么时候你应该用数据库存储数据?

考虑一下这个简单的XML文件,它包含了一个家庭的信息。这个信息很容易用XML文件存储,而使用数据库存储就非常的困难。

```
<family>
  <name rel = "father">
    <first></first>
    <last></last>
  </name>
  <name rel = "mother">
    <first></first>
    <last></last>
  </name>
  <name rel = "child" order = "1">
    <first></first>
    <last></last>
  </name>
</family>
```

XML非常适合用于结构化但结构中包含无规则信息的存储。在数据库中,字段的数目必须是恒定的,因此你就不能为家庭中的每个孩子建立一个单独的字段。此外,每一个字段通常有一个定长空间用来对数据进行存储。这在你存储叙述性文件的时候,会浪费空间。

22.2 XML 查询语言(XQL)

查询XML文件的一致性语法大纲在1998年9月完成,当时以Microsoft、Texcel和webMethods为代表,将这个语法提交给了XML工作组。从此,虽然不是一个官方标准,这个语法占领了查询语法方面的大量阵地,基于这个语法开发的应用程序不断增多。这个语法

的最初的提案可以在 <http://www.w3.org/Style/XSL/Group/1998/09/XQL-proposal.html> 中找到。设计的理论基础可以在位于 <http://www.texcel.no/whitepapers/xql-design.html> 的 Jonathan Robie 的白页中找到。

22.2.1 什么是 XQL 查询

XQL 查询是一种对一个或多个 XML 文件进行的查询。查询从进行搜索的这些文件中获得一个或多个节点。它返回一个封装在根元素(xql:result)的节点集。它们创建了一个结构良好的文件。

提示:为了要区别是通过 XQL 返回的查询还是有专门方法返回的查询,本章将使用约定。将 XQL 结果封装在<xql:result></xql:result>标记中,而将其他结果封装在<xql:return></xql:return>标记中。这样做还可以达到另一个目的:结构良好的文件。

查询搜索的节点集称为文件的搜索内容(search context)。查询(query)是形成这组节点的查询。结果集(result set)是返回的节点集。

搜索内容

搜索内容可能是多个 XML 文件、单个 XML 文件或是文件的一部分。例如,在 movies.xml 中,搜索内容可以是整个文件或者是文件中的一部电影。如果对标题的搜索用于整个文件,就会返回四个节点。如果只用于一部电影,就只返回一个节点。根节点必须包含被搜索的所有节点。

查询

查询是形成文件的查询。它是通过查询语言的语法表达的。它可以是单独节点类型的请求,或者它也可以通过限定词和过滤器进行限制。用于这方面的特定语法将在稍后进行讨论。

结果集

结果集是搜索返回的一个节点集。虽然 XQL 对这些节点返回的方法只字不提,但是大量的实现把它们作为结构良好的 XML 文件进行返回。没有什么可以阻止应用程序把它们作为 DOM nodeList、数组或其他合适的类型进行返回。在本章所使用的例子中,它们将在大部分地方作为 XML 文件出现。结果集被封装文件封装以确保文件是结构良好的。

22.2.2 XQL 查询样例

在下面的例子中。我们使用 title 查询 movies.xml(我们还没有讲解语法,因此在这里我们使用伪代码)。如下所示,我们收到了一个结构良好的文件:

```
Search context
  Movies.xml
    Query
      ./title
    Result Set
```

```

<xql:result>
  <title>Raising Arizoma</title>
  <title>Midnight Run</title>
  <title>The Usual Suspects</title>
  <title>The Abyss</title>
</xql:result>

```

22.3 XQL 语法

一个虚拟的 XQL 引擎要得到两部分信息：搜索内容和查询。

搜索内容可能是一个简单结构的文件或是一个文件的引用。使用这种形式相当于单独的查询。除非特别的声明，我们将在我们所有的例子中使用 movies.xml (如清单 22.1 所示)。

清单 22.1 Movies.xml 清单

```

<? xml version="1.0" standalone="no"? >
<! DOCTYPE movies SYSTEM "Movies.dtd" [
  <! NOTATION JPEG SYSTEM "c:\progra~1\intern~1\explore.exe">
  <! -
  <! NOTATION JPEG SYSTEM "Iexplore.exe">
  -- >
  <! ENTITY raposter SYSTEM "RAPoster.jpg" NDATA JPEG>
  <! ENTITY mrposter SYSTEM "MRPoster.jpg" NDATA JPEG>
  <! ENTITY tusposter SYSTEM "TUSPoster.jpg" NDATA JPEG>
  <! ENTITY taposter SYSTEM "TAPoster.jpg" NDATA JPEG>
]>
<movies>
  <movie type="comedy" rating="PG-13" review="5" year="1987">
    <title>Raising Arizona</title>
    <writer>Ethan Coen</writer>
    <writer>Joel Coen</writer>
    <producer>Ethan Coen</producer>
    <director>Joel Coen</director>
    <actor>Nicolas Cage</actor>
    <actor>Holly Hunter</actor>
    <actor>John Goodman</actor>
    <poster image="raposter"/>
    <comments>A classic one-of-a-kind screwball love story.</comments>
  </movie>
  <movie type="comedy" rating="R" review="5" year="1988">
    <title>Midnight Run</title>
    <writer>George Gallo</writer>
    <producer>Martin Brest</producer>
    <director>Martin Brest</director>
    <actor>Robert De Niro</actor>
    <actor>Charles Grodin</actor>
    <poster image="mrposter"/>
  </movie>
</movies>

```



```

    <comments>The quintessential road comeday. </comments>
  </movie>

  <movie type="mystery" rating="R" review="5" year="1995">
    <title>The Usual Suspects</title>
    <writer>Christopher McQuarrie</writer>
    <producer>Bryan Singer</producer>
    <producer>Michael McDonnell</producer>
    <director>Bryan Singer</director>
    <actor>Stephen Baldwin</actor>
    <actor>Gabriel Byrne</actor>
    <actor>Benicio Del Toro</actor>
    <actor>Chazz palinteri</actor>
    <actor>Kevin Pollak</actor>
    <poster image="gusposter" />
    <comments>A crime mystery with incredibly intricate plot twists. </comments>
  </movie>

  <movie type="sci-fi" rating="PG-13" review="4" year="1989">
    <title>The Abyss</title>
    <writer>James Cameron</writer>
    <producer>Gale Anne Hurd</producer>
    <director>James Cameron</director>
    <actor>Ed Harris</actor>
    <actor>Mary Elizabeth Mastrantonio</actor>
    <poster image="taposter" />
    <comments>A very engaging underwater odyssey. </comments>
  </movie>

```

查询是一个根据 XQL 语言的语法构造的字符串。一个查询字符串由用于搜索的节点和一系列的限定词和过滤器组成。搜索节点通常放在字符串的右端。限定词预定节点,而过滤器放置在节点后跟的方括号中。正规的语法是:

Qualifiers/searchnode[filters]

多看一些例子就会对它了解的更清楚。现在我们将探讨多种限定词。

22.3.1 层叠查询字符串

最简单的查询字符串就是层叠式查询字符串。层间关联可以用单斜杠表示成一种“父/子”式关联,或者用双斜杠表示成“祖先/后代”式关联。

“父/子”式查询

一个单独的正斜杠(/)象征着“父/子”式关联。因而,下面的命令是用来寻找以 movie 为父节点的 title 节点。

' movie/title'

它将返回我们在前面的例子中同样的结果集。另一方面,下面的代码将返回一个空的结果集,这是因为这里没有以 movies 为父节点的 title 节点:

' movies/title'

“祖先/后代”式查询

两个正斜杠(//)表明了一种“祖先/后代”式的关联。因而,下面的代码将返回一个与 movie/title 相同的结果集,因为 title 确实需要将 movies 作为祖先。

```
'movies//title'
```

通配符

星号(*)表示一个通配符。例如,下面的代码将寻找以 movies 为祖先的 title 节点。

```
'movies/* /title'
```

要注意,这与 movies//title 并不相同。因为在这种情况下(movie/* /title)下,我们不是寻找一个祖先节点,而是要寻找所有的祖先。它将返回一个包含所有 title 节点的结果集。

22.3.2 连续关系查询

连续关系通过分号指明。这种连续关系并不能被所有的执行工具支持。

直接领先

一个分号表明了一种直接领先的同属关系。例如,下面的代码将解析那些被另一个 producer 节点领先的节点:

```
'producer; producer'
```

在 movies.xml 中,它的返回如下所示:

```
<xql:result>
  <producer>Michael McDonnell</producer>
</xql:result>
```

领先

连续的分号表明一个通过限定性节点排序的同属在任何位置都被领先的节点。实际上,在 movies.xml 中,下面的代码将产生与前面的例子完全相同的结果。

```
'producer;; producer'

<xql:result>
  <producer>Michael McDonnell</producer>
</xql:result>
```

这种限定词在大多数的 XQL 执行工具中都不能使用。

22.3.3 查询以找到属性

要找到属性,就要使用符号@与属性名联合形成一个过滤器。做这件事情的正规语法如下:

```
'nodename[@attributename]'
```

因而,在 movies.xml 中,下面的查询将返回所有拥有 type 属性的 movie 节点:

```
'movie[@type]'
```

换句话说,它将返回所有的 movie 节点,因为他们都有 type 属性!然而,我们也可以进一步用属性的值来进行查询,这样,下面的代码将返回第一个和第二个 movie 节点:

```
'movie[@type="comedy"]'
```

而下面的代码将返回另外两个节点:

```
'movie[@type!="comedy"]'
```

要注意“!=”,在这里它是不等号。

注意:在稍后我们就会看到,'movie[@type="comedy"]'是'movie[type!nodeValue="comedy"]'的缩写形式。

返回属性

下面的语法将返回一个 movie 节点中的所有 type 属性:

```
'movie/@type'
```

在一个执行工具返回的 XML 文件中,结果集将会是一个包含指定属性和相应值的元素节点的集合。我们并不清楚一个 XQL 执行中返回的 DOM 结果是如何被返回的。可假定,当这个语法在返回 DOM 集的执行工具中运行的时候,结果应该在元素节点的 NameNodeMap 或 nodeList 的结构中。

22.3.4 处理空白空间

在本章的所有例子中,都加入了空白空间以便于清楚显示。然而,XQL 查询字符串是被特意设计成在没有空白空间情况下可用的。这就使它们能够在使用空白空间是违法的地方运行(比如一个 URL)。

22.3.5 使用过滤器

当我们在上一节中讨论属性选择时,我们超前了一些。我们实际上是使用了一个过滤器。过滤器子句可以通过使用方括号([])加入到节点中。当我们给节点加入了一个过滤器时,我们就加入了一个布尔子句的注解。如果方括号中的语句为真,节点就被返回。过滤器的正规语法如下所示:

```
Nodename[filter syntax]
```

过滤器可以在层的任意位置上使用。请看下面的代码:

```
'movie[@type='SciFi']/title'
```

它将返回下面的结构良好的 XML 文件:

```
<xql:result>
  <title>The Abyss</title>
</xql:result>
```

在这里,过滤器被应用在搜索节点的父节点上。

22.3.6 布尔语句

布尔语句使用 AND、OR 和 NOT 运算符。在很多的编程语言中,这些被分别表示成 &&、|| 和 !。XQL 使用下面的语法作为它的操作符(注意这些都是区分大小写的)。

- \$ and \$ —— 它与 AND 或 && 等价。
- \$ or \$ —— 它与 OR 或 || 等价。
- \$ not \$ —— 它与 NOT 或 ! 等价。

22.3.7 等式

等号(=)在等式中使用。而 \$ eq \$ 也可以使用。对于不等式,可以使用通常的 != 或是 \$ ne \$。我们已经在前面例子的属性中使用了它们。下面的两行是相同的:

```
'movie[@type != "comedy"]'
'movie[@type $ ne $ "comedy"]'
```

下面的这两行也是相同的:

```
'movie[@type = "comedy"]'
'movie[@type $ eq $ "comedy"]'
```

22.3.8 比较

下面是比较操作符的清单。表中每一行的符号是等价的。这些是可以自我解释的。

| 符号 | 等作字符串 |
|----|----------|
| = | \$ eq \$ |
| != | \$ ne \$ |
| < | \$ lt \$ |
| <= | \$ le \$ |
| > | \$ gt \$ |
| >= | \$ ge \$ |

大小写非常的重要,这是因为字符串要被返回。也就是说,M 的值要小于 m。为了避免这个问题,XQL 设计了一系列的操作符来忽略字符串的大小写。这些操作符在所有前述的标记前加前缀 i(例如, \$ eq \$、\$ ine \$、\$ ilt \$ 等等)。因此,下面构造的值为真:

```
'M' $ ieq $ 'm'
```

22.3.9 XQL 方法

XQL 也有一些方法用于对集合进行高级操作。这些方法在语法的过滤器部分被使用。下面是一些最常用方法的清单:

```
text()
value()
```



```
nodeType()
```

```
nodeName()
```

```
index()
```

```
end()
```

text()方法

text()方法将在元素中搜索所包含的文本。它将关联所有元素后代的文本。下面的例子将找出所有节点文本是"James Cameron"的 director 节点:

```
'movie/director[text() = "James Cameron"]'
```

它将返回 movies.xml 中相应的 title 节点。要注意,这里有一个更简单的方法运行这个方法,并且它将产生同样的结果:

```
'movie/director[ "James Cameron"]'
```

如果我们相比返回 title 节点,更希望返回 movie 节点(可以假定这将更为有用),我们将使用下面的语法:

```
'movie[director ! text() = "James Cameron"]'
```

或者,我们可以使用更简短的版本:

```
'movie[director = "James Cameron"]'
```

这里,我们使用 director 节点作为 movie 节点的过滤器,而 movie 节点就是搜索节点(或者说是被返回的节点)。

一定要注意惊叹号,它在节点名放置在过滤器中时,用来将节点名与方法区分开。

提示:使用层叠往往比使用过滤器更为有效,这是因为大多数的执行工具将建立一个查询节点,层叠式的将它们分类,然后对它们进行过滤。然而,一些应用程序正好相反,也就是说,它们使用过滤器将更为有效。

value()方法

如果支持数据类型,它将会返回节点的类型矢量。否则,它就和 text()方法有着完全相同的结果。实际上,如果没有设计方法,在下面的例子中:

```
'movie[director = "James Cameron"]'
```

节点的值被返回,这是因为值是默认的。因此,我们在 text()方法一节中看到的缩写形式返回的值恰好就是文本值。

nodeType()方法

nodeType()方法返回一个相当于字符串的整数值。这些整数与 W3C DOM 中规定返回的整数常量相同,即:

- element 1
- attribute 2
- text 3

- PI 7
- Comment 8
- Document 9

在 movies.xml 中,下面的查询将返回 movies.xml 中所有将注释作为孩子包含的 movie 节点(对于记录,这就没有)。

```
'movie[*! nodeType=8]'
```

注意通配符的使用是要选择 movie 元素的所有子节点,并使用叹号! 将子节点与我们作为过滤器使用的方法区分开来。

nodeName()方法

nodeName 方法将返回元素的 tagName。因而,下面的查询将返回所有包含 title 的 movie 节点:

```
'movie[*! nodeName()="title"]'
```

这将是所有的 movie 节点。同样,注意下面的语法将获得同样的节点集:

```
'movie[title]'
```

index()方法

index()方法返回节点在父节点中的索引号。因而,在 movie.xml 中,下面的查询将返回前三个 movie 节点:

```
'movie[index()<3]'
```

当然索引是从零开始的。

end()方法

end()方法将得到同级节点集中的最后一个节点。下面的查询将返回最后一个 movie 节点:

```
'movie[end()]'
```

而这个查询将返回每个电影中最后一个被列出的演员:

```
movie/actor[end()]
```

换句话说,它将返回如下结构良好的 XML 文件:

```
<xql:result>
  <actor>John Goodman</actor>
  <actor>Charles Grodin</actor>
  <actor>Kevin Spacey </actor>
  <actor>Mary Elizabeth Mastrantonio</actor>
</xql:result>
```

22.3.10 在查询中使用圆括号

注意这个圆括号可以用来聚合节点集。在前面的例子中,'movie/actor[end()]' 返回每

部电影最后一个被列出的演员,下面的查询将返回所有 movie 元素的最后一个演员:

```
'(movie/actor)[end()]'
```

也就是:

```
<xql:result>
  <actor>Mary Elizabeth Mastrantonio</actor>
</xql:result>
```

22.3.11 名字空间查询方法

下面的方法可以用来返回节点的名字空间信息:

- `baseName()`——返回节点的名字部分。
- `namespace()`——返回作为节点名字空间的 URI。
- `prefix()`——返回节点的称谓部分,例如允许你搜索所有的 `xhtml:element`。

22.3.12 聚合方法

聚合方法可以将所有相同类型的节点聚成一个集合。下面是一些公认的聚合:

- `textNode()`——返回在搜索内容中的所有文本节点。
- `comment()`——返回在搜索内容中的所有注释节点。
- `pi()`——返回在搜索内容中的所有 PI 节点。
- `element('name')`——如果没有提供可选的 `name` 参数,方法将会返回所有的节点。否则,只有与 `name` 匹配的元素节点被返回。
- `attribute('name')`——我们不清楚在查询不返回 DOM 对象的执行操作中,这是如何执行的。在返回 DOM 对象的应用程序中,将会返回一个 `NameNodeMap`。
- `node()`——返回一个除所有属性节点外的所有节点的集合。

每个集合都要被包括进去,同时每个集合都要被索引访问。例如,下面的代码返回所有 `title` 元素的第一个文本节点:

```
'title/textNode()[0]'
```

注意下面的代码返回一个空集,这是因为 `title` 元素中没有第二个文本节点:

```
'title/textNode()[1]'
```

下面的代码显示 `movies.xml` 的第三个标题:

```
'movies/element('title')[2]'
```

这个例子将返回如下结构良好的 XML 文件:

```
<xql:result>
  <title>The Usual suspects</title>
</xql:result>
```

`count()`方法

`count()`方法用来与集合关联以得出集合中项目的具体数目。

22.4 XQL: 执行工具和其他资源

每天都有越来越多的执行工具可以达到投入市场的水平,它们其中的大部分都是免费的。下面是在本书编写时,一些有用资源的清单:

- <http://metalab.unc.edu/xql/>——这是初始计划书的作者之一(Jonathan Robie)撰写的常见问题解答(FAQ)。
- <http://www.openxql.org/>——这是一个改良版XQL的免费资源组织。
- <http://xml.darmstadt.gmd.de/xql/>——GMD是一个商业产品,但它有一个可用版本正在评价过程中。它是一个基于Java的产品。
- <http://www.globit.com/infonyte.htm>——这也是一个基于Java的商业产品。

在你读本书的时候,应该已经有一些其他的可用执行程序了。

22.5 XSLT 模式

XSLT 提供了一个有用的XQL语法子集用来模式化。实际上,XQL本来是作为XSL模式语法的扩展集描述的。在IE5中可以对xsl提供有限的支持,在这一节中将提供一些在IE5中使用XSL模式的例子。

你应该还对第10章和第11章有印象,XSL中的模式是选取用于转换的节点语法。这里,我们就这个语法给出一些例子。

22.5.1 XSLT 查询的例子

为了浏览这些例子,你一定要使用IE5,它提供对XSL说明的一些支持。我们只要简单的在movies.xml的开始部位加入一个执行命令,如下所示:

```
<? xml version="1.0" standalone="no"? >
<? xml-stylesheet type="text/xsl" href="movies4.xsl"? >
```

当然,执行指令中的href必须与运行选择的XSL样式表相匹配。

选择title元素的例子

这个例子简单的选取所有title元素进行显示,如清单22.2所示。效果如图22.1所示。

清单 22.2 movies1.xsl

```
<xsl:stylesheet xmlns:xsl=http://www.w3.org/TR/WD-xsl>
  <xsl:template match="/">
    <xsl:apply-templates select="//title" />
  </xsl:template>
  <xsl:template match="title">
    <h3><xsl:value-of /></h3>
  </xsl:template>
</xsl:stylesheet>
```

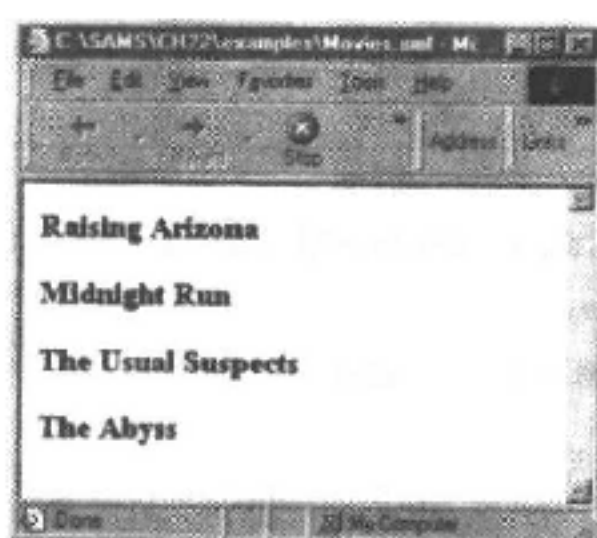


图 22.1 title 元素清单

匹配所用属性值的例子

这个例子选取所有的 movie 元素,在它们之中匹配出 type 属性值为 comedy 的元素,以把它们进行显示,如清单 22.3 所示(效果如图 22.2 所示)。

清单 22.3 movies2.xsl

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <xsl:apply-templates select="//movie" />
  </xsl:template>
  <xsl:template match="movie[@type='comedy']">
    <p><xsl:value-of /></p>
  </xsl:template>
</xsl:stylesheet>
```

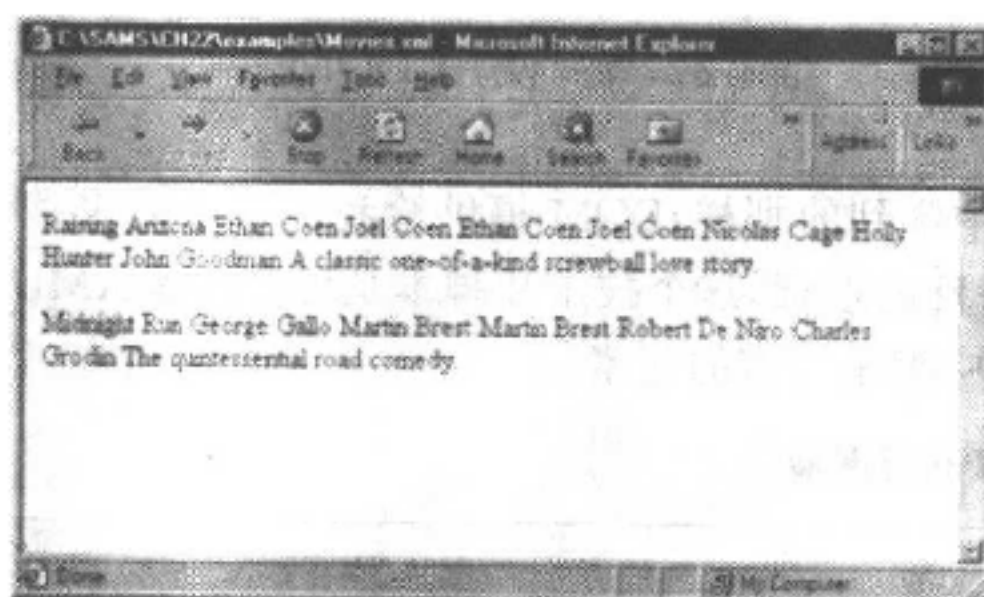


图 22.2 当清单 22.2 中的代码在 IE5 中显示时,产生的警告信息

匹配最后一个元素的例子

这例子选取所有的 title 元素,并匹配出那些是 movie 元素的最后一个 title 后代的 title

元素,如清单 22.4 所示(效果如图 22.3 所示)。

清单 22.4 movies3.xsl

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <xsl:apply-templates select="//title" />
  </xsl:template>
  <xsl:template match="movie[end()]/title">
    <h3><xsl:value-of /></h3>
  </xsl:template>
</xsl:stylesheet>
```

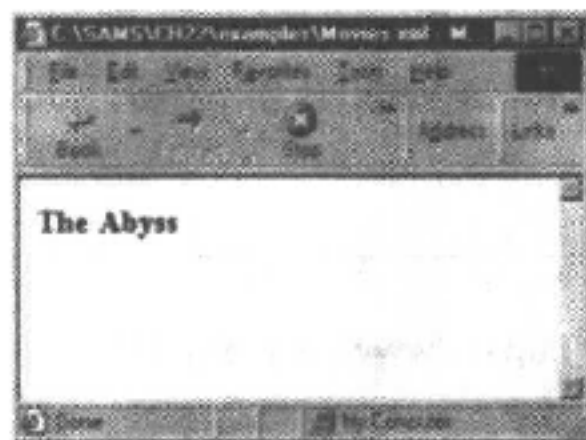


图 22.3 清单 22.4 代码在 IE5 中运行的情况

使用 XSL 查询的明显场合就是在查询结果要显示的时候!然而,大多数 XSLT 的执行程序可以产生一个可被另一个应用程序处理的新的 XML 文件。例如,MSCLSCOM 可以用来从使用 XSL 查询的结果中产生一个 XML 文件树,这个树可以按照你希望的任何方式使用。

22.6 使用 DOM

就像你在第 15 章中学到的那样,DOM 提供给我们一个用来搜索 XML 文件的非常有效的方法。同样,对于特殊的搜索,这个技术和脚本是程序搜索 XML 文件所需要的全部。清单 22.5 在 movies.xml 中搜索所有的元素。

清单 22.5 使用 DOM 执行搜索

```
<html>
<title>DOM XQL functions</title>
<!--use your own source path!!-->
<xml id="isle" src="movies.xml"></xml>
<script>
xDoc=isle
if (xDoc.parseError != 0)
  {alert(xDoc.parseError.reason)}
```

```

else
{
    var dummy=getQuery("title")
    alert(dummy)
}

function getQuery(x)
{
    var returnString= "<xxql:return> \n"
    var tagList=xDoc.getElementsByTagName(x)
    for (var i=0;i<tagList.length;i++)
    {
        returnString=returnString + "<" + x + ">"
        returnString=returnString + tagList.item(i).firstChild.data
        returnString=returnString + "</" + x + ">\n"
    }

    returnString= returnString + "</xxql:return>"
    return returnString
}
</script>
</html>

```

当这个例子运行的时候,将会出现一个警告窗口。这个警告窗口包含一个 xql:return 文件(如图 22.4 所示)。



图 22.4 清单 22.5 所产生的警告

为了方便起见,这个例子使用 XML DSO 调用 XML 文件并创建一个 DOM 对象。当然,你应该使用你自己的资源路径:

<xml id="isle" src=" movies. xml"></xml>

下面的函数用来将 XML 文件的元素名传入来进行搜索：

```
function getQuery(x)
{
```

对于查询字符串,创建了一个封装开始语句:

```
var returnString= "<xql:return> \n"
```

然后一个 nodeList 对象就被建立了,它包含文件中所有节点的名字:

```
var tagList=xDoc.getElementsByTagName(x)
```

这个列表的长度可以得到。然后，一个 for 循环返回所要原件的所有文本数据：

```

for(var i=0;i<tagList.length;i++)
{
    returnString=returnString + "<" + x + ">"
    returnString=returnString + tagList.item(i).firstChild.data
    returnString=returnString + "</" + x + ">\n"
}

```

要注意 title 元素的内容是怎么成为 firstChild 节点的 data 特征的：

```
returnString=returnString + tagList.item(i).firstChild.data
```

再创建查询文件的封装关闭语句：

```
returnString= returnString + "</xql:return>"
```

然后将上述字符串返回：

```

return returnString
}

```

22.6.1 何时使用 DOM 查询

DOM 可在多个应用程序中被执行。就如你在第 15 章中所学的，使用 MS 或 IBM 解析器，DOM 可以很容易的结合在你的 C++ 程序中。Java 解析器同样支持为 Java 程序员提供 DOM 支持。虽然 DOM 查询与使用 XQL 引擎相比并不方便（这是由于这个引擎并没有开发完成），在不久的将来，DOM 查询将是搜索 XML 文件的一种可选方法。

22.7 自编函数

的确，我们不必在自编函数上花费过多的篇幅——我相信在这方面你已经比我超前一些了。我们给出一些小例子就可以了。

清单 22.6 包含一个用 VB 编写的函数，它将 XML 文件和一个元素名作为参数，并以 XQL 文件形式返回所有那个名字元素的实例。

清单 22.6 用于搜索的 VB 程序

```

Function getTagContent(doc As String,tag As String) As String
' this function returns anXQL query as an xml string containing
' the elements asked for of the name tag in the xml document doc
' declare variables
Dim xdoc $ ,otagstr $ ,ctagstr $ ,lstr $ ,returnstr $
Dim otagpos! ,ctagpos!
' initialize variables
xdoc = doc
' construct tags
otagstr = "<" & tag
ctagstr = "</" & tag
' Chr(13) & Chr(10) is a new line

```



```
returnstr = "<xql:return>" & Chr(13) & Chr(10)

' return all tag element as a xml strings
Do While InStr(xdoc,otagstr) <> 0
    otagpos = InStr(xdoc,otagstr)
    ctagpos = InStr(xdoc,ctagstr)
    lstr = Mid(xdoc,otagpos,ctagpos - otagpos)
    returnstr = returnstr & lstr & ctagstr & ">" & Chr(13) & Chr(10)
    xdoc = Mid(xdoc,(ctagpos + 1))
Loop

returnstr = returnstr & "</xql:return>" & Chr(13) & Chr(10)
getTagContent = returnstr
End Function

Private Sub Command1_Click()
' opens movies.xml and passes its content to the function
Dim xdoc $
' provide your own path!!
Open "c:\sams\movies.xml" For Input As #1
xdoc = Input(LOF(1),1)
Close #1
dummy = getTagContent(xdoc,"title")
MsgBox dummy
End Sub
```

当上述查询执行的时候,你会看到一个包含 `xql:return` 文件的消息框(如图 22.5)。



图 22.5 清单 22.6 运行时所产生的消息框

main 函数在传入用于搜索的 XML 文件和要搜索的元素时,只返回一个 xql:return 文件。

封装提供给这些返回的元素：

```
returnstr = "<xql:return>" & Chr(13) & Chr(10)
... content here ...
returnstr = returnstr & "</xql:return>" & Chr(13) & Chr(10)
```

提示:对于 VB 语法不熟悉的读者,我们要知道 Chr()是一个根据给出的 ASCII 码返回字符的方法。在 Windows 编程中,新起的一行要包括换行(ASCII 13)和回车(ASCII 10)。

元素的内容通过一个简单的循环从 XML 文件中得到：

```
Do While InStr(xdoc, otagstr) <> 0
```

```
Otagpos = InStr(xdoc, otagstr)
ctagpos = InStr(xdoc, ctagstr)
lstr = Mid(xdoc, otagpos, ctagpos - otagpos)
returnstr = returnstr & lstr & ctagstr & ">" & Chr(13) & Chr(10)
xdoc = Mid(xdoc, (ctagpos + 1))
Loop
```

提示:同样,对于不了解VB语法的读者,InStr(string 1, string 2)是返回 string 2 在 string 1 中开始位置的方法。Mid(string, startpos, num characters)方法返回了一个按照参数得到的传入字符串的子串。两个附加参数是子串的起始位置和返回的字符数。如果第三个参数被忽略,就像循环中使用 mid() 的第二个地方用的一样,到字符串末尾的所有字符都会被返回。

22.7.1 什么时候使用自编函数查询

如果你要处理大量相同类型的记录——这种情况经常出现——使用自己编写的函数进行搜索将比使用他人的程序和 DOM 更为有效,速度更快。自编函数可以在搜索药物记录的整个过程中使用。

22.8 总 结

本章探讨了用于 XML 文件的查询语言的需求,同时也研究了这种语言的主要要求和它们与数据库查询语言的需求是如何不同的。特别是,XML 查询语言可以做下面的工作:

- 描述层间关系。
- 描述层内关系。
- 返回一个结构良好的 XML 文件。

本章还讲了一下 XQL,它是符合这些需求的语言。之后,你学习了这种语言语法上的要点部分。

本章在对 XSL 中 XQL 执行情况的清醒认识中结束。同样,你也学习了使用 W3C DOM 和自编程序在 XML 文件上执行搜索。

第 23 章 使用 XML 和 ASP 访问数据库

数据库和 XML 有很多相似之处。它们都是结构化并存储信息的途径。数据库是由记录和字段组成的表格存储信息,这样就允许方便的访问和搜索。XML 文件也可以按照类似的方式结构化来存储信息。

本章讨论动态服务页面(Active Server Pages, ASP)是如何被用来开发 XML 数据存储的,以及它是如何将标准数据库转化为 XML 或是 XHTML 文件的。

ASP 是使用脚本语言和 COM 对象创建交互 Web 页面的 Microsoft 基础技术,它可以方便的访问数据库和 XML 文件。

23.1 在数据库管理系统中使用 XML

数据库有可以允许在它们的内存中进行搜索、过滤和对照记录的引擎。在一个简单的搜索中,数据库引擎通常从第一个记录开始,依次遍历每一个记录,将每个记录与在搜索要求中的准则进行比较。随着 DOM 的出现,在 XML 文件上执行简单的搜索操作也变得同样的简单。

只要创建一个活动的数据库描述,数据库就可以自动更新。同样,通过使用 ASP 或类似的引擎,XML 也可以享有同样的特性。

有着这么多的相似点,因此 XML 在一开始被用作数据存储也就并不奇怪了。当前,XML 在数据库管理方面有四个主要用途。

- 备份和归档数据存储。
- 作为不同数据库和数据存储之间的发送数据的包。
- 作为显示数据的包。
- 本身作为数据存储。

23.1.1 XML 用于归档

就如你将要看到的,将数据库读入 XML 文件是非常简单的,用途也很广泛。一旦进入了 XML 文件的表中,数据不会受到二进制废弃(如何方便的在你的老系统上访问数据呢?)。数据的所有者也就不必在数据管理中只使用单独的专门表格了。令人惊讶的是,按照 XML 存储数据还可以节约存储空间。

23.1.2 XML 作为包

如何将 Oracle 数据库的信息发送给 SQL 服务器?除非使用 XML,否则这件事将困难重重。使用 XML,来自一个专有系统的数据表可以封装在一个结构表格中并被另一个专有系统读入。

23.1.3 XML 用于显示

在现代浏览器中,显示 XML 是非常简单的。只要给已经按照 XML 包装的记录集加上一个样式表,你就可以进行实时的显示。对于老一些的浏览器,只要使用 DOM 或 XSLT 做一个到 HTML 的转换就可以了。通过使用 ASP 它可以完全实现。

23.1.4 XML 作为数据存储

XML 越来越多的按照它自己的方式作为数据存储使用。当遇到下列情况的时候可以考虑使用 XML 作为信息的基本存储。

- 信息是一个复杂的表格。一些记录的表格并不能很好的适用于数据库。一个医生病人记录,可能同时会有几个工作访问或者是一百个甚至更多。一个数据库要怎么构造才能应付这种巨大的变化呢? XML 可以方便的处理这种情况。
- 单独的字段复杂而又庞大。同样,诸如药品记录这样的记录,其中的官方鉴定可以是几个字或是数页。数据库同样不能很好的对它进行存储。数据库中的每个字段必须是等长的,这样就会浪费大量的存储空间。
- 搜索的速度并不是问题。数据库引擎在速度上经过了优化。虽然 DOM 可以让你在 XML 文件上执行任何的搜索操作,与数据库搜索相比仍然是慢速的处理。例如,据说 ESPN 服务器峰值时期每秒可以执行 1200 次搜索。现在,让 XML 搜索应付这种数量的情况是不可能的。
- 数据类型并不重要。XML 中的所有数据都是以字符串存储的。虽然 Schema 允许数据描述,但信息仍然是按照字符串的方式存储的。另一方面,数据库可以在它们的本身格式中存储数据类型,这样就很容易将数据传送给处理这些数据的程序(比如有关星辰位置的天文程序)。可使用传统的数据库来存储这种类型的信息。
- 数据库很小,但要有可伸缩性。XML 是建立诸如刚开业公司所需要的那种小型数据存储的理想选择。因为,如果需要,它将非常方便的将信息传给 DBMS。XML 数据存储具有非常好的可伸缩性。

本章将对所有的这些问题进行探讨,并且我们还将向你展示如何使用 ASP 访问和处理数据库和 XML 文件以履行数据管理所必需的功能。这里条件非常简单,但到本章的最后,你将会有一个扎实的基础来使用 XML 文件开始创建你自己的 ASP 页面。你可以理解这种出自 Microsoft 的强大的服务器端技术是如何使得在 XML 文件和数据库文件间交互信息方面的工作更为简单。

23.2 个人万维网服务器(PWS)和动态服务器网页(ASP)

在你开始使用本章所提供的例子之前,你需要有一个可解读 ASP 的浏览器。如果你正在使用 Windows,那么这就意味着要在你的机器上加载个人万维网服务器(Personal Web Server,PWS)。这是 Microsoft 的免费和可用的程序,它允许你在你自己的桌面上开发和测试 Web 页面。

如果你正在使用其他的操作系统,比如 UNIX 或基于 Linux 的系统,或者你在操作一个非窗口化的 NT 服务器,这里也有一些实用技术允许你在这些系统中启用 ASP。Chilisoft (<http://www.chilisoft.com>)和 Halcyon Software (<http://www.halcyonsoft.com>)都有可以在不同操作系统上工作的 ASP 版本。Halcyon Software 还有一个可以免费下载的开发实用版本。

23.2.1 使用 PWS

从 Web 开发人员的角度来看,ASP 的最大优势之一就是所有的 ASP 页面在送到主服务器之前可以通过桌面机进行浏览。这可以通过在你自己的桌面机上运行 PWS 或简单的应用程序来实现。因此,我强烈推荐安装 PWS 或简单的应用程序,比如 Halcyonsoft 的 ASP 引擎。

PWS 是 NT 选项包的一部分。它可以从位于 <http://www.microsoft.com/windows/ie/pws/default.htm?/windows/ie/pws/main.htm> 的 Microsoft 网站上下载。

它可以在 Windows 95、98 或 NT 上运行。

NT 选项包在 Window NT 和 Window 98 上被预加载,而 PWS 必须在运行选项包的情况下才能被激活。在 Windows 98 上,它可以在 Internet Explorer 目录下找到。PWS 同时还在 Visual Studio 6 以及许多的 Microsoft 网络应用程序中被加载,因此如果你不想麻烦的去下载它,你可以在这些地方寻找。不幸的是,Microsoft 在 Mac 中不再支持 PWS,但是如果仔细寻找,你可以找到一个旧的版本。

当 PWS 被安装,它通常放置在一个名为 inetpub 的文件夹中。ASP 文件将被存入 wwwroot 文件夹下。ASP 将分配一个本地内部网用户名,任何在 wwwroot 文件夹下的文件都可以通过如下的输入进行访问:

`http://[local user name]/[path under wwwroot]/[file name].asp`

提示:你可以在工作栏中通过双击 PWS 图标找到 PWS 分配的用户名,同时还启动了个人网站管理器。当你点击“开始”按钮,你的本地用户名就会被显示。

如果输入的本地地址并不是内部网地址,文件将不会传给 ASP 引擎并且也不会解释任何的代码。

23.2.2 理解 ASP

ASP 是一项 Microsoft 技术,它被设计用来创建交互式、动态的 Web 页面。与其他的技术协作,它允许客户端和服务端之间的持续连接,发展客户端/服务器端会话,对数据库的简单访问,对文件系统的简单访问。ASP 也使用公共对象模型(Common Object Model, COM)透明的添加诸如 XML 解析器和处理函数这样的对象。ASP 使用 VBScript 作为本机语言。但它也可以使用 JavaScript 或 Perl。

基本上,在服务器上的 ASP 页面接收来自客户端的请求并处理这个请求,可能还要访问数据存储器,然后返回一个 HTML(或 XML)流给服务器。它通过使用在 ASP 引擎中创建的一系列对象来实现。

ASP 文件可以通过使用 suffix.asp 与 HTML 文件区别。当可识别 ASP 浏览器得到后

缀为 .htm 或 .html 的页面的请求,它将运行如下几个步骤:

- 寻找所给地址的页面。
- 如果找到了页面,就将它以适当的 http 标头送给客户端。
- 如果出现了任何问题,它就会产生一个错误。

这个步骤与不是可识别 ASP 的浏览器的正规响应是没有区别的。然而,当一个可识别 ASP 服务器接收到一个后缀为 .asp 的页面时,它的执行步骤如下:

- 寻找所给地址的页面。
- 如果找到了页面,将它发送给 ASP 引擎进行进一步处理。
- ASP 引擎处理页面并创建一个 HTML 流。
- HTML 流被发送给客户端。
- 如果出现了任何问题,它就会产生一个错误。

处理 ASP 页面

ASP 页面是 HTML 标记和可被 ASP 引擎处理的脚本代码结合的产物。ASP 的本身代码是 VBScript,尽管 JavaScript 和 Perl 也可以被使用。本章的例子将使用 VBScript。ASP 引擎能够识别文件中期望处理的部分,因为这些部分被放置在如下的分隔符中:

```
<% '[code to be processed goes here]%'>
```

这里的代码可以横跨几行:

```
<%  
'[several lines  
'of code to be  
'processed  
'go here]  
%>
```

清单 23.1 就是一个简单的 ASP 页面。

清单 23.1 简单的 ASP 页面

```
<html>  
<title>Simple ASP page</title>  
<h1>Server Time</h1>  
<p>The time at this server is  
<%  
response.write now  
%>  
Eastern Standard Time  
</p>  
</html>
```

下一节中,我们将讨论 ASP 引擎是如何处理这个简单页面的。

创建 HTML 和 XML 流

ASP 引擎所要做的就是从 ASP 页面创建一个 HTML 流。它将所有事物在 `<%... %>` 分隔符以外的 HTML 标记毫不改变的传送给客户端。任何在分隔符内的事物将被解释并转换成 CDATA 流。

在清单 23.1 中,整个 HTML 流由三个部分组成。第一部分是开始 HTML 标记:

```
<html>
<title>Simple ASP page</title>
<h1> Server Time</h1>
<p>The Time at this server is
```

第二部分需要进行解释:

```
<%
response.write now
%>
```

`response.write` 是 `response` 对象的一个方法(请参阅“Response 对象”一节)。它与位于客户端的 `document.write` 相同,同时发送一个 CDATA 流给客户端。

提示:发送给客户端的文本流可以包含 HTML 和 XML 标记。客户端将对它们进行适当的解释。因而,如果发送的文本流是:

```
This is <i> italic text</i>
```

客户端将解释发送过来的标记,因此字符串“italic text”将以斜体字方式显示。

`now` 是一个 Visual Basic 方法,它返回当前的日期和时间。因而,日期和时间也就会作为文本流被发送给客户端。

第三部分也是 HTML 标记:

```
Eastern Standard Time
</p>
</html>
```

图 23.1 是清单 23.1 在 IE5 中的显示情况。

为了完成这些目的,ASP 要使用一整系列的对象。这些对象一部分是 ASP 内部构建的,而其他的对象要由程序员创建。下一节,我们将看一些更为重要的对象,并将提供足够的信息以便于理解本章中的例子。再一次强调,这只是让你起步的一个摘要。

23.3 本机的 ASP 对象

ASP 包含一定数目的本机的对象,这些对象有着它们自己的属性和方法。在本章将要看到的对象是 `server` 对象、`request` 对象和 `response` 对象。

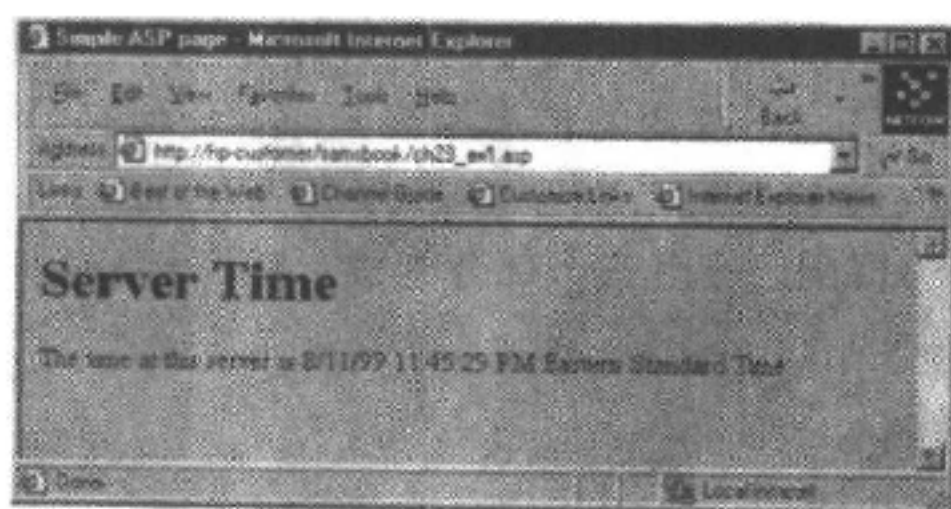


图 23.1 清单 23.1 在 IE5 中的显示情况

23.3.1 server 对象

server 对象是一个低级别对象,它提供一些真正底层的方法。在本章中你将要使用的方法是用于创建 server 组件或 ActiveX 对象的实例的方法。这就是 createObject 方法。Visual Basic Script 中使用它的语法是:

```
set [objectname] = server.createObject("string describing object")
```

在本章中,你将会多次使用这一行。基本上,你使用这一行代码所要做的就是创建一个所要求对象的实例。对于使用面向对象的编程语言,你需要实例化(instantiate)这个对象。然后,你要使用数据填充这个对象。

23.3.2 request 对象

request 对象是 ASP 的基本对象。任何时候你运行 ASP 页面,它都会出现,因此它不必按照前面的语法创建。换句话说,任何一个 ASP 引擎的内建对象都不需要实例化。

request 对象处理从客户端发送来的请求,它由一定数量的集合组成。例如,考虑一下下面的来自客户端的代码:

```
<form action="myasp.asp" method="post">  
<input type="text" value="[enter first name]" name="firstname" />  
<input type="text" value="[enter last name]" name="lastname" />  
<input type="submit">  
</form>
```

如果我在文本域中填入了我的姓名并点击“提交(submit)”按钮,下面的字符串将通过 http 标头送给 myasp.asp:

```
"firstname=Frank&lastname=Boumpfrey"
```

我可以使用在 myasp.asp 中的下列代码访问这些值:

```
dim a,b  
a=request.Form("firstname")  
b=request.Form("lastname")
```


变量 a 现在包含字符串 "Fank", 而变量 b 包含字符串 "Boumphrey".

23.3.3 response 对象

response 对象也是一个内建 ASP 对象, 它并不用手工配置。顾名思义, 它是对客户端请求进行响应的。

你将一次次使用的方法就是 Response.write 方法。它类似于浏览器文件对象中的 document.write 方法:

```
Response.write "Here is some CDATA to be sent to the <b>client</b>"
```

它将发送 CDATA 流给客户端, 在那里它将通过浏览器显示。当然, 标记将会被解释, "client" 将会以粗体形式显示。

提示: JavaScript 需要方法后面跟圆括号。在 VBScript 中你也可以这样选择, 但前一种格式更为普遍。

下面的代码在 VBScript 中都是正确的:

```
Response.write "Hello ASP"
```

和

```
Response.write("hello ASP")
```

JavaScript 与 VBScript 的另一点不同就是 VBScript 中的方法不区分大小写。Response.Write、response.write 和 Response.write 将会被同样对待。

通常, 你将在同一个 ASP 文件中处理 JavaScript 和 VBScript。VBScript 将在服务器端运行, 而 JavaScript 将在客户端运行。你可能希望按照我的习惯, 用小写字母书写所有的 VBScript 方法。这也可以使你大概确定一下你是使用 JavaScript 还是 VBScript。

由于 response.write 语句被非常频繁的采用, 这里有一个它的缩写方式:

```
<%="Hello ASP"%>
```

它与下面的代码产生同样的结果:

```
<%response.write "hello ASP"%>
```

23.4 脚本对象

脚本对象包括 Dictionary 对象、FileSystemObject 对象和 TextStream 对象。在本章中, 你将使用 FileSystemObject 对象和 TextStream 对象。这些对象并不属于 ASP 引擎本身, 因此他们需要进行实例化。

Dictionary 对象超出了本书的范围。如果你希望对这个对象有更多的了解, 请参阅《Active Server Pages 2.0 Unleashed》。

23.4.1 FileSystemObject 对象

FileSystemObject 对象允许对服务器的文件系统和任何附属于网络的服务器的文件系

统进行访问。它通过使用 `createObject` 语法创建：

```
Set [variable]=server.createObject("Scripting.FileSystemObject")
```

这个对象有大量的方法，它们基本上可以分成四组。我将主要讨论其中部分更为重要的方法。如果你对这些方法的整体描述有兴趣，请参阅《Active Server Pages 2.0 Unleashed》。

文件操作方法

使用下面的方法，你可以打开、创建、移动、删除和拷贝文件。所有的这些方法需要全路径名和文件名作为参数。

- `CreateTextFile()`——创建参数中路径和文件名称指定的文件。
- `OpenTextFile()`——打开指定的文件。这个方法将在你确实需要打开一个文本文件时进行讨论。
- `DeleteFile()`——删除指定的文件。
- `FileExists()`——返回一个布尔值。
- `CopyFile()`——需要两个参数，初始文件位置和新文件位置。
- `MoveFile()`——也需要两个参数，初始文件位置和新文件位置。

文件夹操作方法

下面是一些文件夹操作的方法。它们需要文件夹路径作为参数。当然 `MoveFolder` 和 `CopyFolder` 需要两个参数：

- `CopyFolder()`
- `CreateFolder()`
- `DeleteFolder()`
- `FolderExists()`[a Boolean]
- `MoveFolder()`

路径操作方法

`BuildPath()`允许你创建一个文件路径。

资料方法

对象中有很多的资料方法。下面列出一些最为有用的：

- `GetAbsolutePathName`——当传入一个相关的路径(文件的URL地址)时，返回完全路径名称。
- `GetParentFolderName`——当传送进文件的名称时，得到父目录的名称。
- `GetDriveName`——当传送进文件的名称时，返回驱动器的名称。

23.4.2 TextStream 对象

`TextStream` 对象是 `FileSystemObject` 对象的对象属性。当使用 `CreateTextFile()` 或 `OpenTextFile()` 方法时，它将被返回。和其他对象一样，必须使用关键字 `set` 把它赋给一个变量。下面的代码显示了如何创建一个 `TextStream` 对象：

```
dim oFSO ' this will be assigned the FileSystemObject object
```

```
dim oTXTstream      ' this will be assigned the TextStream object

' first create a FileSystemObject object

set oFSO = server.CreateObject("Scripting.FileSystemObject")

' now open a file. This method returns a TextstreamObject

set oTXTstream = oFSO.OpenTextFile("c:\junk\junk.txt")
```

一旦 TextStream 对象被创建,它就可以使用它所有的方法和属性。然而,要注意文件是以确定的模式被打开的。默认是以读方式打开文件,就像这里做的一样(因此缺省了第二个参数)。如果以读方式打开,你就不能对它进行写操作,反之亦然!为了打开一个文件以使你可以给它添加数据,你要像下面这样操作:

```
set oTXTstream = Ofso.OpenTextFile("c:\junk\junk.txt", 8)
```

整数 8 告诉 ASP 以追加方式打开文件。

提示:三种模式分别是读(1)、写(2)和追加(8)。读模式只允许读取文件的内容。要注意,如果文件以写模式打开,任何先前的内容都要被抹去!追加模式允许你在文件结束处进行写操作。默认的情况是以读模式打开文件。如果你尝试在一个以读模式打开的文件上进行写操作,程序就会抛出一个错误。当你在一个以追加模式打开的文件上进行读操作时同样的情况也会发生!

这里有一些 TextStream 的方法,它们可以在相应的文件打开模式下被使用:

- ReadAll——它将整个文件内容作为一个字符串读取。在前面的例子中,下面的代码将会把文件 c:\junk\junk.txt 的整个内容读入到变量 strTextstring 中:

```
dim strTextstring

strTextstring = OTXTstream.ReadAll
```

- WriteLine——在文件末尾写入以参数传入的字符串,并加上一个行结束符。
- Write——将以参数传入的字符串写入文件的末尾。
- Close——关闭文件。

23.5 ActiveX 和 DOM 对象

在 ASP 中,如果程序中使用了 ActiveX 对象,这个对象的实例首先要被创建。这是通过首先声明变量,再将对象的实例赋给变量这两个步骤实现的。主要的语法如下:

```
dim [objectName]

set [objectName] = server.createObject("[string representing object template]")
```

set 是用于实例化对象的 Visual Basic 关键字。Server.createObject() 是 server 对象的方法,它可以用来创建一个对象的实例。传给 createObject() 的参数字符串告诉 ASP 创建对象的种类。

ASP 创建对象之后,对象就可以用相应的数据进行填充。这里有很多可以被实例化的

对象。甚至,你也可以在C++或Visual Basic中编写自己的对象,在你的服务器上注册并使用它们!然而,本章仅限于对数据库ActiveX数据对象(ActiveX Data Object for Databases——ADODB)和XML DOM对象进行讲解。

提示:要注意ASP自身创建的对象和ASP使用的对象是有区别的。ASP创建的对象在每次ASP程序运行时自动的被实例化。而ASP使用的对象要通过脚本实例化。

23.5.1 ADODB 对象

本章将使用两个ADODB对象。Connection对象提供一种到数据库的连接模型,Recordset对象提供一种对来自数据库的特定数据集的表述。这里说明了这些对象是如何被实例化的:

```
dim oCom
dim oProp

set cCom=server.createObject("ADODB.Connection")
set cCom=server.createObject("ADODB.Recordset")
```

23.5.2 XML DOM 对象

XML对象Microsoft.XMLDOM将提供一种可载入任何XML文件的DOM。下面说明了这个对象是如何被实例化的:

```
dim oXMLdoc

set oXMLdoc=server.createObject("Microsoft.XMLDOM")
```

给这个对象加载一个XML字符串,可以像下面这样做:

```
oXMLdoc.loadXML([wellformed XML string])
```

要给它加载一个XML文件,可以这样做:

```
oXMLdoc.load([absolute file path])
```

提示:ADODB对象模板来自于ASP引擎。XMLDOM对象是用于IE5的MSXML.dll的实例,你可以在第14章“使用C++解析XML”中看到。这个DLL需要在服务器上注册。如果在服务器上有IE5的拷贝,那么它就已经注册了。

23.6 理解数据库

在继续深入之前,我们要讲一些数据库的知识。如果你对数据库和数据库术语很熟悉,你可以跳过这一节。

简单来讲,数据库是任何信息的集合。然而在普通的计算机应用中,数据库是按照定义方式存储的相关信息的集合。下面是一些数据库的术语:

- 表——出于某些逻辑原因集合在一起的信息的集合。比如一个目录,用户地址和姓

名列表或者甚至是有关宠物的信息。表被分成记录(行)和字段(列)的。

- 记录——表中的一个独立实体。也被称为行(row)。
- 字段——协助建立完整记录的独立元素。也称为列(column)。
- 索引——用来按某些逻辑要求分类表的一个字段或一个字段集合。
- 主键——这是表中可以惟一鉴别记录的字段。它经常是数字。有时,主键和索引是同样的。
- 数据库引擎——归档、组织和获取包含在数据库中的所有信息的程序。
- 查询——从数据库中得到所需信息的操作。
- 记录集——逻辑相近数据构成的不同记录或表的集合。理论上,它用在 ADO 对象中;它是来自表的记录的集合。

23.7 使用 ASP 连接数据库

数据库的完全连接指南超出了本书的范围。本章向你展示的东西,将足够使你熟悉数据库中的术语和 TLAs(three-letter acronyms,三个字的缩写)。你将学到的不仅会使你更为危险,也使你更为重要。你将使用 ASP 连接数据库!

你所使用的例子的数据库提供者是 Access,而在相应的地方还将需要 SQL Server 的语法。所用的数据库是 biblio.mdb,它是一个有关 Microsoft 提供的应用程序种类的 Access 数据库,包含所有的 Visual studio 程序和 Microsoft Office。然而,如果你没有到这个数据库的入口,你完全可以使用任何一个你希望的数据库来运行示例。

本节的策略就是提供一个连接 biblio.mdb 的 ASP 页面的简单例子,然后对它进行分析。

23.7.1 ODBC 和 OLE-DB

要连接数据库,理解这一节的信息实际并不真的必要。然而,对于连接数据库的“螺钉”和“螺母”的基本理解,可以使你更容易理解后面介绍的代码(更不用说,这会使你所听到的术语更加容易理解)。

ODBC 是开放数据库连通性的简称。它也是一项 Microsoft 技术,设计用来创建一个访问数据存储的方法和例程的公共集合。在很大程度上,它正在被 OLE-DB(Object Linking and Embedding Database,对象链接和嵌入式数据库)所取代。OLE-DB 速度更快也更容易使用。ActiveX 数据对象(ActiveX Data Object,ADO)是位于 OLE-DB 上的接口,它提供用于连接和处理数据的属性和方法。

图 23.2 显示了 OLE-DB 技术的体系结构。

23.7.2 访问数据库

在这个例子中,connect1.asp 是一个 ASP 页面。它用于连接 biblio.mdb,并从数据库中读出所有由 Sams 出版的书的标题。

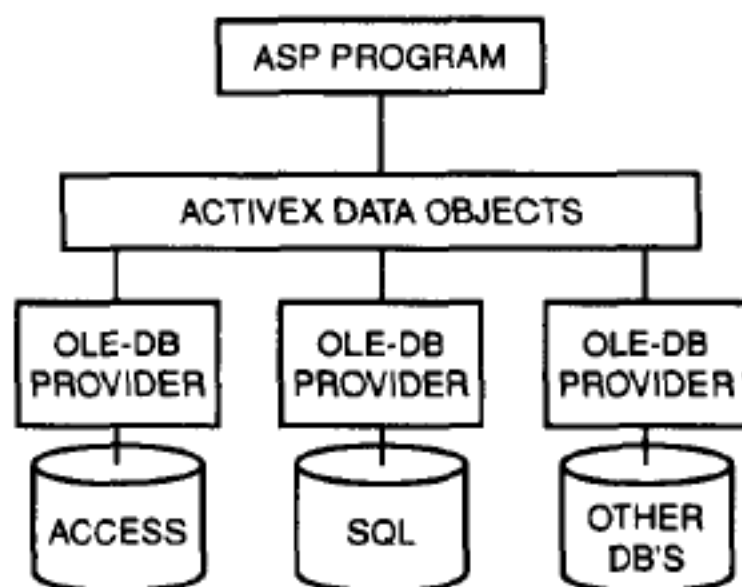


图 23.2 OLE-DB 技术体系结构图

警告：这是一个老数据库，标题已经非常过时了。

警告：Biblio.mdb 是许多 Microsoft 产品的数据库，包括一些版本的 Office 和 Visual Studio。如果你不能访问这个拷贝。你可以将任何你希望的数据库用在这些例子上。

清单 23.2 是 connect1.asp 代码的完整清单。

清单 23.2 Connect1.asp

```

<html>
<head>
<title>Connecting to a database with asp</title>
</head>
<body bgcolor="white">
<h3 align="center">'Sams' titles in biblio.mdb</h3>
<%
dim strConnect
strConnect="Driver={Microsoft Access Driver (*.mdb)};
DBQ=C:\book_web\chap6b\biblio.mdb"

dim oConn
dim oProp

set oConn=server.createObject("ADODB.Connection")
set oProp=server.createObject("ADODB.Recordset")

oConn.Open strConnect
oProp.Open 'Titles',strConnect

while not oProp.eof
  If oProp("PubId")=721 then
    response.write oProp("Title") & "<br />"
  end if
  oProp.MoveNext
wend

oConn.close
set oConn=nothing
oProp.close
  
```

```
set oProp=nothing
%>
</body>
</html>
```

下面对代码如何工作进行解释。

创建 ADO 对象

和其他对象一样,当 ADO 对象在程序中被使用前,首先要创建它的一个实例。这里,我们创建了两个对象——connection 对象和 recordset 对象。首先,要声明包含对象的变量:

```
dim oConn
dim oProp
```

然后对象本身要赋给这些变量:

```
set oConn=server.createObject("ADODB.Connection")
set oConn=server.createObject("ADODB.Recordset")
```

语法的细节我们已经详细讨论过了。

提示:在上面的例子中,你创建了 connection 对象和 recordset 对象。实际上,如果只使用 recordset ADODB,创建 connection 对象并不必要。recordset 对象将提供它到数据库的自己的连接。如果移去对 oConn 的所有的引用并且不再创建 connect 对象,这个页面同样可以很好的工作。connection 对象用来实现使用 SQL 的高级搜索。而实际上,在本章中 connection 对象根本没有使用。

加载 ADO 对象

一旦对象被创建,你就需要加载它:

```
oProp.Open "Titles", strConnect
```

Titles 是你希望在数据库中连接的表。StrConnect 是一个变量,它包含 oProp 打开数据库所需要的所有必要的信息。这一点将在下一节进行详细介绍。

recordset 对象的 Open 方法需要五个参数,参数间用逗号隔开。如果它们没有直接设置,就会使用相应的默认值。下面是 Open 方法的正规语法:

```
[recordset].Open [Source],[connection],[recordset type],[lock type],[options]
```

记录集类型(recordset type)、加锁类型(lock type)和选项(options)的值在这里都被忽略了,以使用它们的默认值。

提示:许多特殊的方法都有一些可选参数。它们可以设置数据库的日常操作,其中大部分并不需要。如果没有给出,ASP 就会使用默认的参数。具体的可参阅 ASP 全文或 Microsoft 站点上有关可选参数的文档。

记录集 ADO 的 Open 方法也可以设置其他参数。记录集类型(也称为指针类型)是一个整数,用来指定如何浏览记录集。加锁类型提供了编辑记录集的不同保护程度。选项参数最终确定源代码特征如何被解释。更深的细节超出了本章的范围,你可以参考一本 ASP 的书

或是从 Microsoft 网站中得到更多的细节。请浏览 <http://msdn.microsoft.com> 并搜索 ASP。

提示:ASP 的文件可以在 Microsoft 站点的所有地方都可找到。通常最好的文件在论文中。最好的方法是到:

<http://msdn.microsoft.com>

或

<http://msdn.microsoft.com/library>

中查找。

并使用关键字 ASP 进行搜索。

连接字符串

要注意, strConnect 变量包含着所有你的 ADO 到数据库的连接所需要的信息。一个 Access 连接字符串的正规语法是:

```
Driver={Microsoft Access Driver (*.mdb)};DBQ=[you data base]
```

下面是用来链接 biblio.mdb 数据库的字符串:

```
strConnect="Driver={Microsoft Access Driver (*.mdb)};  
DBQ=c:\book-web\chap6b\iblio.mdb"
```

字符串的第一部分告诉 ADO 使用什么样的驱动器:

```
Driver={Microsoft Access Driver (*.mdb)};
```

第二部分告诉 ADO 数据库的物理位置(这是我在写这一章时将它存储在我的系统中的位置。当然,你要在这里提供你自己的地址):

```
DBQ=c:\book-web\chap6b\iblio.mdb
```

SQL Server 连接字符串

如果你使用 SQL Server, 连接字符串的语法有少许不同:

```
[connection string]="Driver={SQL Server};  
server={name of you server};  
Database=[The name of you SQL server Database];  
UID=[your SQL user id];PWD=[Your SQL server Pass word]"
```

当然,它们要写在一行上。

在 ASP 中使用 SQL

我们可以就 SQL 写整整一本书。幸运的是,你只是要将数据库记录读入 XML 文件,因此你只要知道最少量的 SQL。下面的代码可以工作在任何数据库上。要记住的地方就是当你打开数据库时,你应该使用数据库相应的语法。

下面的代码查找 Titles 表中的 PubId 字段。如果它是整数 721(这恰好是 Sams 出版的书的号码),它将打印书的标题。


```

While not oProp.eof
    If oProp("PubId")=721 then
        Response.write oProp("Titles")&"<br />"
    end if
    oProp.MoveNext
wend

```

图 23.2 显示了你将要看到的情况。

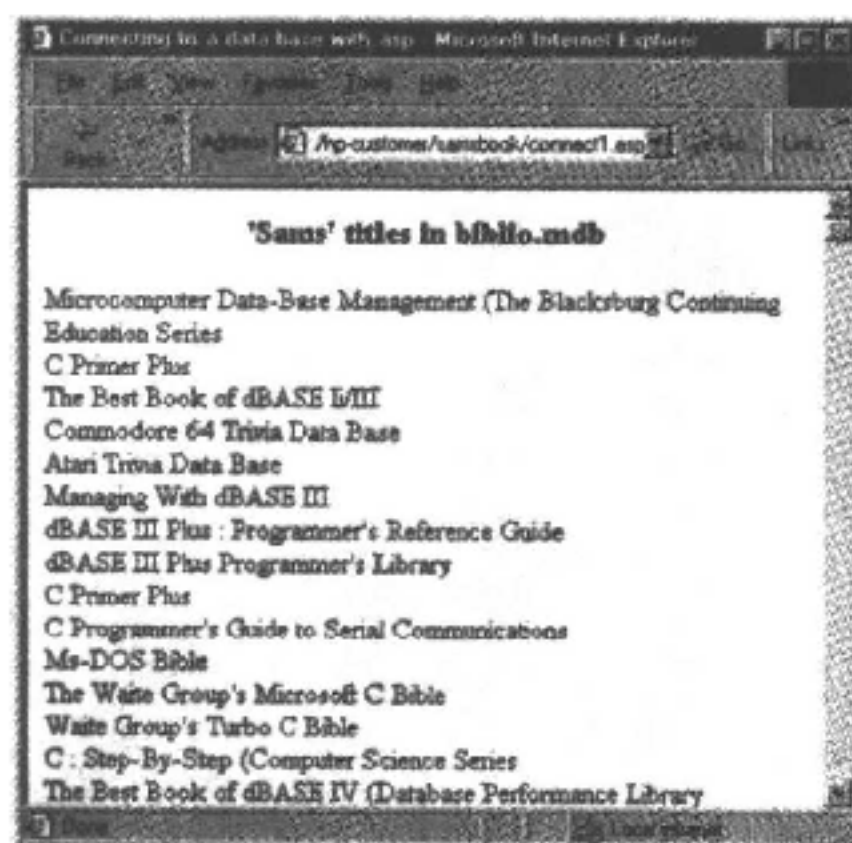


图 23.3 清单 21.2 运行的标题显示

清理

在你完成了工作后,清理是非常重要的!下面的几行代码关闭你所创建的对象并把它们的值设为 nothing。这样就防止了内存流失并释放了在服务器上的空间。

```

oConn.close
set oConn=nothing
oProp.close
set oProp=nothing

```

现在,你已经看到连接到数据库并访问它的记录是多么的简单,让我们研究一下将记录读入 XML 流的工作吧。

23.8 将数据读入 XML 流

如果你知道表名和字段名,从记录集中构造一个 XML 文件就是一个非常简单的事情。

同样,这一节的策略是提供一个连接 biblio.mdb 的 ASP 页面、并将其中的一个表读入 XML 文件的简单的例子,然后对代码进行分析。

下面列出的 connect2.asp 与 biblio.mdb 数据库连接并打开 publishers 表。然后它将创建一个名为 bib_pub.xml 的 XML 文件并且将每个记录中的 PubId、电话和姓名字段读入

XML 文件。它是通过遍历记录并将每一个记录写在 XML 文件的如下格式的行中来实现这一点的。

```
<publisher> <Publd> [value] </Publd><name>[value] </name>
<telephone>[value]</telephone></publisher>
```

清单 23.3 是它的完整代码。

清单 23.3 connect2.asp——创建 bib-pub.xml 文件

```
<html>
<!--this is connect2.asp-->
<head>
<title>Connecting to a db and writing a table to XML</title>
</head>
<body bgcolor="white">
<h3 align="Center"> Biblio.mdb: Writing 'publishing' table to XML</h3>
<%
dim strConnect
strConnect="Driver={Microsoft Access Driver (*.mdb)};
DBQ=C:\book-web\chap6b\biblio.mdb"
(good fb)

dim oProp
dim oFSO
dim oXMLfile

' create ADODB object and open "Publishers" table from Biblio.mdb
set oProp=server.createObject("ADODB.Recordset")
oProp.Open "Publishers",strConnect

' create the scripting and textstream objects
set oFSO=server.CreateObject("Scripting.FileSystemObject")
set oXMLfile=oFSO.CreateTextFile("c:\junk\bib-pub.xml")

' write your XML file.
oXMLfile.WriteLine "<table-publishers>"

' loop through the recordset
while not oProp.EOF
    if Instr(oProp("name"),"&")=0 then
        oXMLfile.WriteLine "<publisher><publd>" & oProp("Publd")
        & "</publd>" & "<name>" & oProp("name") &
        "</name>" & "<telephone>" & oProp("telephone") &
        "</telephone>" & "</publisher>"
    end if
    oProp.MoveNext
wend

oXMLfile.WriteLine "</table-publishers>"

' housekeeping
oProp.close
set oProp=nothing

oXMLfile.close
set oXMLfile=nothing
%>
```

```

<h3 align="center"> Biblio.mdb: Publishing table written
- to c:\junk\bib-pub.xml</h3>
</body>
</html>

```

这段 ASP 代码所做的第一件事就是创建一个 recordset 对象。然后它与 biblio.mdb 连接并加载 Publishers 表。这与清单 23.2 中所做的完全相同。

现在代码需要做的就是创建一个文本文件以让 XML 字符串写入。你将在下一节中做这项工作。

创建 FileSystemObject

FileSystemObject 是脚本对象的子对象。它以通常的方式创建：

```
set oFSO = server.CreateObject("Scripting.FileSystemObject")
```

FileSystemObject 可以使用所有的方法和属性对文件系统进行处理。这些方法之一就是 CreateTextFile() 方法，这也是你用来创建 XML 文件的方法。

创建一个新的文件和一个 TextStream 对象

CreateTextFile() 方法创建一个可以用文本方式打开的文件，而不是二进制文件。XML 属于如下范畴：

```
set oXMLfile = oFSO.CreateTextFile("c:\junk\bib-pub.xml")
```

由于一个对象要被创建，我们必须使用关键字 set。创建的对象是 TextStream 对象。TextStream 对象有一些属性和方法，其中包括 writeline 方法：

```
oXMLfile.writeline "[some text string]"
```

这个函数将一行文本和一个行结束符写入文本文件。

要注意，如果文件名已经存在，原来的文件就会被覆盖。

写入新文件

首先根元素要被写入：

```
oXMLfile.writeline "<table-publishers>"
```

然后使用一个简单的循环遍历记录集并按照 XML 格式写入文件：

```

' loop through the recordset
while not oProp.eof
  if instr (oProp("name"), "&") = 0 then
    oXMLfile.writeline "<publisher><publd>" & oProp("Publd") & "</publd>" &
      "<name>: " & oProp("name") & "</name>" & "<telephone>" &
      oProp("telephone") & "</telephone>"
  end if
  oProp.MoveNext
wend

```

最后，写入根元素的结束标记：

```
oXMLfile.writeline "</table-publishers>"
```

要注意,如果姓名字段的值包含“与”号(&),它将不会被写入 XML 文件中。这是好的结构应该避免的错误:

```
if instr(oProp("name"),"&")=0 then
```

在实际情况下,这个值应该传入一个函数,将“与”号转换为实体 & 同样也要对 < 字符执行同样的方法。

23.8.1 扩展 connect2.asp

connect2.asp 例子被故意设计得十分简单以突出重点。然而,使用 ASP 就可能将这个清单扩展为一个接口。使用这个接口,使用者可以做如下事情:

1. 选择数据库。
2. 列出数据库中的表,选择它所需要的进行备份。
3. 列出表中它希望加入到 XML 代码中的字段。

在本章中没有使用足够的篇幅介绍如何做这些事情,但它却是可以实现的,并可以作为备份系统应用于任何数据库。

23.8.2 使用 XML 备份记录

你所创建的这两个小应用程序——connect1.asp 和 connect2.asp。它们看起来非常的简单但不要低估它们的能力。它们是使用 XML 进行数据备份的关键。你为什么要使用 XML 进行备份?最主要的原因就是,这样你就有了一个应用独立的存储。XML 可以从一种数据库中下载数据。并将数据上传到完全不同的数据库中。另一点比较重要的是,XML 已经存在了很长时间。ASP 到现在出现还不到 10 年,但有一点可以保证——新的系统 SSO (Super Server Object, 超级服务对象)将能够接受 XML,并把它下载到二进制存储中。

还有一个理由是 XML 的压缩格式非常的紧凑。这可能会让你大吃一惊。毕竟,所有的标记不是肯定要占据大量的存储空间吗?这在 XML 的非压缩格式中的确如此,但根据运行法则并且由于标记被不断的重复,压缩后的 XML 只比压缩后的文本占据稍多的空间,而比二进制数据存储占据的空间要少的多。

Biblio.mdb 在我的机器上占用 3600KB 的空间。当它被转换为 XML 并使用如 WinZip 这样的简单应用软件压缩后,它只有 22KB。压缩率为 0.6%。

23.8.3 使用 XML 进行信息封装

XML 在传送来自 Internet 或内部网中的数据库的信息方面非常的有用。小数量的记录集转换成 XML 并被传给服务器端的 DSO 进行处理。更多的信息还可从服务器 A 上的 Oracle 数据库传送给服务器 B 的 SQL 数据库。XML 甚至可以用来将信息从老式大型机传送给 Access 数据库。所有的这些都因为 XML 的众多功能而变得非常容易。

如果 XML 这么好,为什么不将 XML 作为存储数据的首选呢?下一节我们将讨论这个特别的问题。

23.9 使用 XML 作为数据存储

下面有很多的议论,认为使用 XML 作为存储数据的首选手段非常有意义,但有些人觉得这没有意义。在探讨这一点之前,让我们看一下如何获得 XML 存储的数据。

23.9.1 使用 ASP 访问 XML 文件

这里有两种方式获得 XML 文件:

- 作为文本流的简单结构文件。
- 作为 XML DOM 对象。

当执行第一种方法,所使用语言(VBScript)中的所有字符串处理程序,包括 TextStream 方法,都会被用来处理字符串。

23.9.2 以简单结构文本文件获得 XML 文件

将 XML 文件以文本文件获得并对它进行改造,我们将用一个简单的例子进行示范。在这个例子中,movies.xml 即将被打开并修改。一个注释将被加在文件的末尾,另一个注释将被加在版本声明后的适当位置。代码中的过程示范可以被认为是对 XML 文件的复杂编辑。

为了编辑 movies.xml,ASP 代码需要执行四个步骤:

1. 文件首先以追加方式打开,将注释放在文件末尾。
2. 文件以只读方式再一次打开,内容被读入一个字符串变量中。
3. 使用简单的 VBScript 方法将字符串分为两部分:版本声明,以及它之后的所有字符。将注释插入到版本声明之后。
4. 现在,文件再一次以写模式打开。它自动清空了整个文件。不要着急!你现在可以使用 TextStream 对象的 write 方法将你修改后的字符串读入文件。

清单 23.4 就是 connect5.asp 的代码。

清单 23.4 访问 XML 文件

```
<html>
<head>
<title>Accessing XML files</title>
</head>
<body bgcolor="white">
<h3 align="center"> Opening movies.xml and adding a
comment to the beginning of the file</h3>
<%

dim xmlString
dim oFSO
dim oXMLfile

' OpenTextFile parameters
' for reading (default)    1
```

```

' for writing                2
' for appending             8

set oFSO=server.CreateObject("Scripting.FileSystemObject")
' open the text file for appending
set oXMLfile=oFSO.OpenTextFile("c:\samsbook\movies.xml",8)
oXMLfile.write(" <! - A comment at the end of the file -- >")
oXMLfile.close
set oXMLfile=nothing

' open the text file for reading
set oXMLfile=oFSO.OpenTextFile("c:\samsbook\movies.xml",1)

xmlString= oXMLfile.ReadAll

' we want to insert a comment right after the version declaration
dim endVer,strLtxt,strRtxt
endVer=instr(xmlString,"? >")+2
strLtxt=left(xmlString,endVer)
strRtxt=mid(xmlString,endver)

xmlString=strLtxt & " <! -a comment right after
the version declaration-- >" & strRtxt
' close the file
oXMLfile.close
set oXMLfile=nothing

' open the xml file for writing
set oXMLfile=oFSO.OpenTextFile("c:\samsbook\movies.xml",2)
oXMLfile.write(xmlString)
' response.write xmlString

oXMLfile.close
set oXMLfile=nothing
%>

</body>
</html>

```

警告: 要注意当 movies.xml 加载进 DOM 对象时,它将会被确认和解析。这就意味着,不仅是 movies.xml 一定要在你的电脑的正确位置上,movies.dtd 和符号的引用也要同样在正确位置。解决的最简单方法就是注释掉 prolog。另一种选择就是,我们可以将 validateOnParse() 和 expandEntities() 属性设置为假(false)(请看第15章“使用文件对象模型”)。

在不同方式下打开和关闭 TextStream 对象

注意你已经创建了用于你准备操作文件的每种类型操作的新 TextStream 对象。OpenTextFile() 方法可以带四个参数,具体如下:

```
Object variable=OpenTextFile(filename,mode,create,format)
```

其中 filename(文件名)参数是必须提供的。

Mode 参数是如下的整数:

1 以只读方式打开。

2 以写方式打开。这个模式自动覆盖文件中以前的数据。

8 以追加方式打开。

如果文件以某种方式打开却尝试在另一种方式下对它进行操作(比如对一个以只读方式打开的文件进行写操作),就会产生一个错误。

creat 参数是一个布尔型参数。如果设为真,在文件不存在的情况下,就会创建一个新的文件。

format 参数是有符号整数,用来指定使用的格式化类型:

0 以 ASCII 格式打开文件。

-1 以统一字符编码标准格式打开文件。

-2 以默认格式打开文件。

在清单 23.4 中,只有一个脚本对象需要被创建:

```
set oFSO=server.CreateObject("Scripting.FileSystemObject")
```

然而,在每次使用不同的写入方式时,都要创建一个新的 TextStream 对象:

```
' open the text file for appending  
set oXMLfile=oFSO.OpenTextFile("c:\samsbook\movies.xml",8)  
oXMLfile.write("<!--a comment at the end of the file -->")
```

当然,每次对象都要被关闭并从内存中移出。

```
oXMLfile.close  
Set oXMLfile=nothing
```

使用脚本以字符串方式处理文件

VBScript 和 JavaScript 有复杂的字符串处理的方法和函数。在下面的小节中,XML 被读入到一个字符串变量中。它被分为两个部分,并将一个注释插入到两部分之间。所有的三个字符串之后被重新组合写入一个新的 XML 文件:

```
xmlString= oXMLfile.readAll  
  
' we want to insert a comment right after the version declaration  
dim endVer, strLtxt, strRtxt  
endVer=instr(xmlString,"?>")+2  
strLtxt=left(xmlString,endVer)  
strRtxt=mid(xmlString,endVer)  
  
xmlString=strLtxt&"<!--a comment right after the version declaration-->"& strRtxt
```

在实际情况下,要执行的主要就是一整系列类似的操作。通常的操作是遍历文件寻找指定记录,取出记录,编辑它,然后再将文件的三个部分重新结合起来。

这些方法对于简单操作非常的有用,但对很复杂的操作,最好是将 XML 文件作为 DOM 进行搜索。

23.9.3 以 DOM 对象方式获取 XML

在大多数情况下,你会希望以文件对象模型(DOM)来处理文件。虽然这与把文件作为

纯文本文件处理相比,速度有点慢,但它可以节省大量的代码。

为了对给出的文件创建 DOM,你首先要使用 createObject()方法创建 DOM 的实例。这里是如何使用 ASP 在服务器上创建 DOM 实例的例子:

```
set [object]=server.createObject("Microsoft.XMLDOM")
```

一旦 XMLDOM 被创建,它必须带上一个 XML 文件:

```
[xmldom object].load("filepath")
```

一旦你创建了 DOM 对象并载入了 XML 文件,你就可以使用 DOM 的方法和属性来访问文件。

23.10 从 XML 数据存储中访问数据

下面的例子访问 movies.xml 并寻找类型为 comedy 的所有电影的标题(如清单 23.5 所示)。注意 DTD 最终确定 title 必须是 movie 的第一个子项:

```
<! ELEMENT movie(title,writer+,producer+,
director+,actor*,poster?,comments?)>
```

这样,你的代码非常的简短。

清单 23.5 列出了 movie.xml 中的所有 comedy 电影的标题。

清单 23.5 提取 comedy 类型的电影标题

```
<html>
<head>
<title>Accessing XML files</title>
</head>
<body bgcolor="white">
<h3 align="center"> Opening movies.xml as a dom object</h3>
<%

dim xmlString
'line10
dim oXMLDOM
dim oXMLfile

set oXMLDOM=server.CreateObject("Microsoft.XMLDOM")
oXMLDOM.load("c:\samsbook\movies.xml")

dim recordlist
set recordlist=oXMLDOM.getElementsByTagName("movie")
for i=0 to recordlist.length-1
    if recordlist.item(i).getAttribute("type")="comedy" then
        response.write
            recordlist.item(i).firstChild.firstChild.nodeValue & "<br />"
        end if
    next
%>
```



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

在做其他事情之前,我们需要在服务器上创建一个 XMLDOM 的实例并给它加载 XML 文件:

```
set oXMLDOM=server.CreateObject("Microsoft.XMLDOM")  
oXMLDOM.load("c:\samsbook\movies.xml")
```

接下来,使用 DOM 的 `getElementsByTagName()` 方法创建所有 movie 元素的节点清单:

```
dim recordlist  
  
set recordlist=oXMLDOM.getElementsByTagName("movie")
```

由于这是一个对象,因此一定要使用 `set` 关键字。通过建立一个 movie 元素的清单,我们就可以使用一个简单的循环遍历它们。

```
for i=0 to recordlist.length-1  
    if recordlist.item(i).getAttribute("type")="comedy" then  
        response.write recordlist.item(i).firstChild.firstChild.nodeValue  
        & "<br />"  
    end if  
next
```

要注意循环次数是节点清单长度减 1。这是因为清单的索引是从 0 开始,而不是从 1 开始:

```
for i=0 to recordlist.length-1
```

在每一次循环中,都是通过使用 DOM 方法测试 type 属性的值:

```
if recordlist.item(i).getAttribute("type")="comedy" then
```

如果值为 comedy,标题就被输出:

```
response.write recordlist.item(i).firstChild.firstChild.nodeValue
```

DTD 指定 title 必须是第一个元素,这就使得访问 title 元素的内容变得特别的简单。

警告: 如果使用 JavaScript 代替 VBScript,DOM 方法的大小写就非常的重要。`getAttribute()` 可以在 Visual Basic 下很好的运行,但在 JavaScript 中它只能写作 `getAttribute()`。

在大部分时候,这确实就是那么的简单。如果你要对你的 XML 存储进行添加或删除操作,你可以使用 DOM 或简单结构文件。如果你用 DOM 来做,你就要使用递归的函数持续的保存字符串(第 15 章中已对它进行了讲解)。

23.11 XML 与 RDBMS 对比

在本章开始的时候,当你希望使用 XML 作为存储时,你已看到一些争论,而在其他地

方就没看到争论。这不是要在使用 XML 或是 RDBMS 之间作出选择。XML 是管理数据的有用工具,一个综合的数据管理系统几乎都是同时使用 XML 和数据库。

23.11.1 什么时候使用 XML 作为数据库的附属

如果你的系统有多个服务器和数据存储,就要使用 XML。XML 是将数据从 Linux 服务器发送到 NT 服务器上的理想方式,反之亦然。就如你所看到的,XML 也是将数据从一个数据库传到另一个数据库的理想方法。

也可以考虑使用 XML 作为数据的长期存储。即使是对于非常大的数据库,如果信息被分存在更小的文件中(比如通过月份),搜索可以变得非常快。例如,在药物记录系统中,XML 就很适合作为存储方式。大量的数据需要长时间的保存,但对数据的访问却很少。这种月份报告可以用 RDBMS 方式存储并以 XML 作为长期的存储。

也可以把 XML 作为所有数据库的备份。

23.11.2 什么时候将 XML 作为数据存储

使用 XML 作为所有以下复杂文件的数据存储,这些文件的结构并不能很好的应用于记录和字段。Movies.xml 就是这类文件的一个典型例子。它允许数目灵活的作者、制造商、导演和演员的字段。

仅在只有很少数目的数据要存储的情况下,你应该使用 XML。由于 XML 以字符串作为搜索,搜索过程远远慢于 RDBMS 的搜索。RDBMS 的搜索通常是经过优化的。然而,如果你有不到 10000 条记录并且你访问的频率不超过每秒 10 次,XML 会是一个很好的选择。你几乎感觉不到它们之间在速度上的差别。

23.11.3 什么时候不使用 XML 作为数据存储

在存储元素的变量数据类型有重要意义的情况下,不要使用 XML 作为数据存储。例如,如果一个天文应用程序搜索天体目录并对找到的编号进行操作,这时将字符串转换成数字就有很大的影响。在这里,应用程序最好是从同一数据类型数据库中获得数据。

如果对数据库有非常高的峰值需求,XML 也不能作为解决的办法。而解决这种情况的惟一途径就是,建立一个典型应用程序,并对它进行强度测试!

23.12 总 结

在本章中,你看到了 XML 和数据库具有很强的结合性。就像熏肉和鸡蛋、船舵和桅杆、伏尔加酒和鱼子酱,它们好像就是为在一起而设计的。你看到了 ASP 在连接 XML 和数据存储之间是如何起到杠杆作用的,如何使用 XML 作为不同数据存储之间的中间人,以及如何使用 XML 本身作为一种数据存储。

ASP 在服务器端使用 ActiveX 对象访问数据库和 XML 文件。数据库通过使用 ADO DB 对象访问,而 XML 文件可以使用采用 Microsoft XML DOM 对象的 DOM 树或使用脚本对象的简单结构文件。

由于空间方面的原因,ASP 占用的空间已经变得如此精简。如果你想获得更多的信息,

可以看一下下面的资源：

位于 <http://www.asptoday.com/default.asp> 的高级 ASP 论坛上的资源。

在 ASP 方面我推荐以下书籍：

《Active Server Pages 2.0 Unleashed》

《Sams Teach Yourself Active Server Pages in 24 Hours》

《Beginning Active Server Pages 2.0》

第24章 使用WIDL链接商用数据

WIDL(Web Interface Definition Language)就是Web接口定义语言。它由一个名为webMethods的公司定义,作为实现Web自动控制的基础XML应用。Web是信息时代的恩惠,它使得日益增长的信息可以在世界范围内使用。基于Web的信息重点集中在Web浏览器上。通过浏览器获取信息包含人的交互操作,将信息从Web传送到其他的应用程序和数据库也同样包含人的交互操作。这样作是低效率的,同时会加大发生错误的可能性。

WIDL被设计成一种接口语言,它可以被用来作为自动执行将信息从Web传送到本地应用程序的操作的中间件。接口定义语言(Interface Definition Language,IDL)定义和管理以不同语言编写的应用程序间的相互操作的语言。每个应用程序都将提供一个外部接口,这个接口描述了程序的属性和它可以执行的方法。IDL是一个通用的词表,所有使用IDL的程序都可以理解这些词表。因而,程序A可以要求程序B为它执行任务。

WIDL尝试为基于Web的应用程序做上述同样的事情。它使用的语言是XML。当使用适当的中间件时,它允许Web资源与以不同的编程语言编写的应用程序之间出现映射关系。这些编程语言可以是C、Java、COBOL、VB等等。Web资源不仅包括HTML和XML页面,但它还包括诸如CGI、ASP等等这样的服务器端Web程序。

虽然WIDL是作为webMethods软件的私有语言编写的,它也开发了符合广域网组织主要精髓的标记语言。具体语法的描述可以在下面的地址中找到:

<http://www.w3.org/TR/NOTE-widl>

这里有WIDL的三个版本。版本1用来开发HTML语法。我们将要在本章中讨论的版本2也可以在上述的URL地址中找到。版本3从来没有在公共场合中被公布。在本书编写时,它是webMethods软件使用的版本。我不讨论这个方法,因为不管是DTD还是它的语法都还没有被公布以全面使用。

本章将探讨WIDL在典型商业网站上的需求,同时我们还要学习版本2中的一些语法的细节。

24.1 商业上使用WIDL

让我们看一个典型的WIDL辅助的站点。

Smith医生和Brown医生运行了一个有效的实例。每一周他们对病人进行多项检查,并将检查交给三个实验室中的一个进行:ACME实验室、Hippocratic实验室或Mercy医院实验室。这些实验室效率非常的高并且拥有自动的服务器。在那里,所有检查的结果被送入到数据库中。每个实验室有一个数据库。从那里,结果可以通过有密码保护的网站进行访问。

在许多方面,这可以给医生的任务带来更大的便利。他们再也不会遇到这样的问题,病人返回就诊却没有带它们的检查结果。现在,他们只需这样的护士,他在工作大部分时间中

访问网站,下载结果并将它们输入到医生自己的数据库中。这种情况的总体过程如图 24.1 所示。

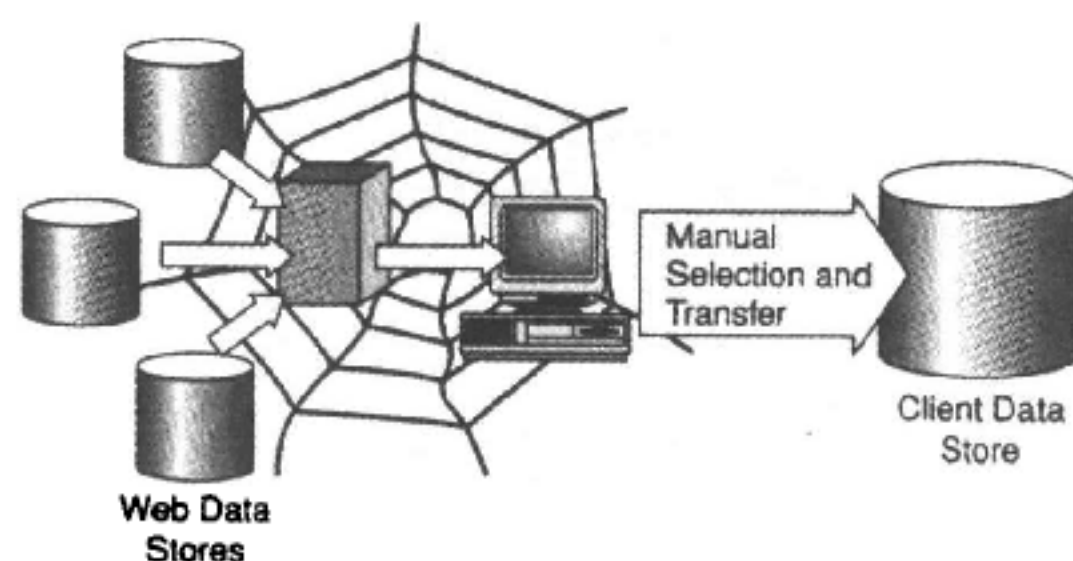


图 24.1 从 Web 手动的获取数据

三个实验室使用了不同的方式向 Web 显示他们的资料。

Acme 实验室使用一个简单的表单,在其中可以输入病人的姓名(如图 24.2 所示)。

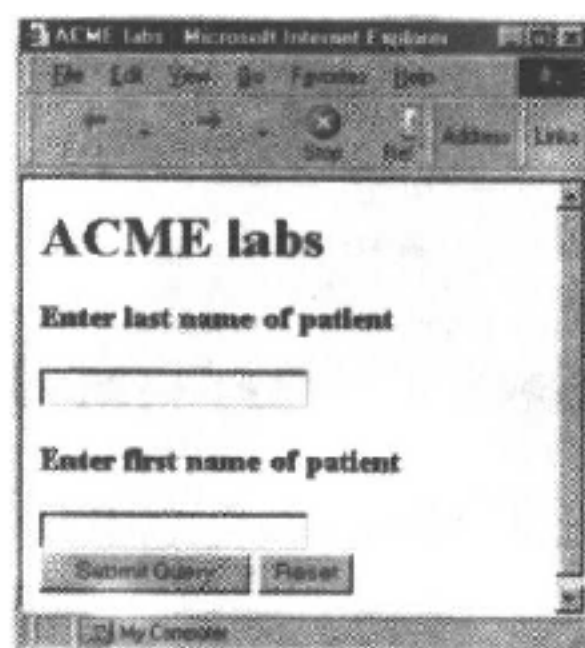


图 24.2 getlabs 接口

Hippocratic 实验室以超级链接的方式列出病人的姓名(如图 24.3)。

Mercy 医院实验室,为了让医生的助理更容易工作,根据需要检查结果的医生的要求列出结果。

这些接口使得护士们可以方便的查寻结果。他有一个计算机生成的清单,里面列出了上一周所做的检查。这个清单包括如下信息:检查类型、病人姓名、检查日期和使用的实验室。他访问相应的实验室网站并查寻结果。如果检查反映出任何事关重大的异常,他就立刻通知医生。如果确有异常,但是无关大局,他就产生一个备忘录提交给医生。如果检查是正常的,他就直接通过浏览器将检查结果粘贴到助理的 Access 数据库中。虽然这项工作非常的简单,但还是需要有一定医学知识并要集中精力。由于 Stephen 要做大量的对数据的剪切和粘贴工作,有时这就有可能出现将错误的检查粘贴进错误的字段之中的情况。

Smith 和 Brown 医生在收到错误的报告时一定很不愉快。好的护士是很难找到的。而不幸的是这项任务相当复杂并且需要足够的医学知识,因此它不能委托给缺乏技术的员工。

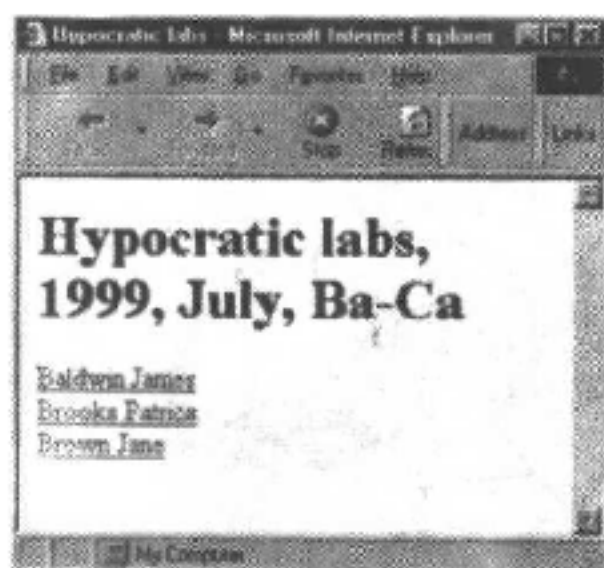


图 24.3 Hypocratic 实验室接口的样例

| Patient Name | Date of Test | Type of Test | Result |
|--------------|--------------|--------------|---------|
| John Doe | July 4 1999 | Hemoglobin | 14.2mg% |
| Jane Doe | sept2 1999 | Sed Rate | 38 |
| John Smith | Oct4 1999 | Blood Sugar | 123 |

图 24.4 Mercy 医院实验室接口示例

24.1.1 自动数据收集

Smith 和 Brown 医生和他们的经理讨论了这个问题,并得出了结论。他们要使这个过程自动化!他们将雇用程序员为他们编写一个程序。由这个程序访问每个实验室的网站,读取数据并将它们记录。

这是该程序必须做到的事情:

- 生成一个约定检查的清单(这已经做了)。
- 浏览这个清单并连通正确的实验室。
- 对每个实验室调用不同的函数,这些函数将病人的姓名、约定的检查和约定的时间作为参数传入并返回检查结果。
- 分析结果,如果它超出了确定的范围,就以适当的紧急程度通知医生。

这里困难的部分是用于连接实验室并读取结果的操作。这个称为 getlabs 的程序要访问正确的页面,在页面上定位病人的姓名并读取结果。

这里是一个程序从 Mercy 医院实验室得到的结果页面。

```
<html>
```

```

<head>
<title> Mercy Hospital Labs</title>
</head>
<body>
<h1>Mercy Hospital Labs Dr. Smith</h1>
<table border="1" cellpadding="10">
<tr>
<th>Patient Name</th><th>Date of Test</th><th>Type of Test</th><th> Result</
th>
</tr>
<tr>
<td>John Doe</td><td> July 4 1999</td><td>Haemoglobin</td><td>14.2mg%</
td>
</tr>
<tr><td>Jane Roe</td><td>setp2 1999</td><td>Sed Rate</td><td>38</td>
</tr>
<tr><td>John Smith </td><td>Oct4 1999</td><td>Blood Sugar</td><td>123</td>
</tr>
</table>
</body>
</html>

```

幸运的是, Mercy 医院有先见之明, 它使用 XHTML 生成所有的页面, 因此程序 getlabs 可以使用 XML 分析工具。

下面是一个 VB 函数的简单版本(相关的检查类型和日期都被忽略了)。一旦定位了正确的页面, 函数就返回检查的结果。

```

Function getTest(xpage As String, pname As String,
    testtype As String, testdate as date) As String
Dim xhpage
Dim xhdoc As Object
xhpage = xpage
Set xhdoc = CreateObject("Microsoft.XMLDOM")
xhdoc.Load(xhpage)
If xhdoc.parseError = 0 Then
    Set rows = xhdoc.getElementsByTagName("tr")
    For i=0 To rows.length
        If rows.Item(i).firstChild.firstChild.nodeValue = pname Then
            getTest = rows.Item(i).lastChild.firstChild.nodeValue
            Exit For
        End If
    Next
Else
    MsgBox xhdoc.parseError.reason
End If
End Function

```

这里有一些艰涩的代码并且需要调试以使它平稳的工作(这个反映在程序的成本中), 但所有人都高兴。Stephen 又可以回去做护士, Smith 和 Brown 医生可以比以前更快的得到

异常结果的通知。这里没有抄写错误只是有一笔较大的用于程序的初始花销,经理希望它可以在六个月内付清!

现在存在的情况如图 24.5 所示。

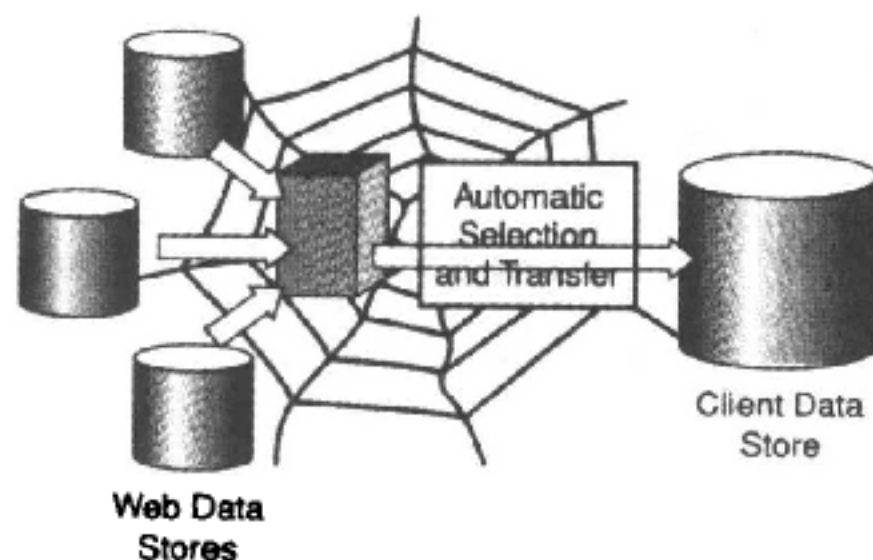


图 24.5 从 Web 上自动选择数据

然而,当实验室改变了回报结果的函数将会发生什么事情那?

24.1.2 使用 WIDL 作为解决方案

Mercy 医院实验室,为了要提高对客户的服务,将加入新的一列以存储对检查结果的解释。图 24.6 显示了他们新的网页的显示情况。

| Patient Name | Date of Test | Type of Test | Result | Interpretation |
|--------------|--------------|--------------|-----------|--|
| John Doe | July 4 1999 | Haemoglobin | 14.2mg/dl | Normal |
| Jane Roe | Sept 2 1999 | Sed Rate | 38 | High, compatible with numerous causes including infection. |
| John Smith | Oct 4 1999 | Blood Sugar | 123 | High if fasting. Suggest GTT. |

图 24.6 新的 Mercy 医院接口

现在,医生的程序不用返回检查结果——而要返回对结果的解释!

Smith 和 Brown 医生是缺乏通知和计划的牺牲者。他们所创建的是一个典型的两层应用程序,而他们不能对其中的任何一层进行控制。每当这些实验室中的一个实验室的 Web 页面发生改变,它们的程序必须重新编写并且要重新进行调试。

他们所需要的是一个包含下列几部分的多层应用程序:

- 一个程序与他们的数据库进行交互,并写入检查结果。

- 一个程序用来生成一个约定检查的请求清单。
- 中间件将作为这两个程序和网络之间的接口。

这样,如果网页发生了任何变化,只有中间件需要重新编写。当然,重编中间件也是一个很大的工程,这里还有更好的解决办法。中间件的指令可以被包含在一个 WIDL 文件中,这个文件告知中间件如何运行以及请求什么(如图 24.7)。如果网页被修改,所需要的就只是一个相关的 WIDL 文件的修改版本。

还要注意,计划中还要加入另一个解释性程序(用 VB 程序描述,如图 24.7 所示)。在本章中,我们将不涉及这个程序的细节。

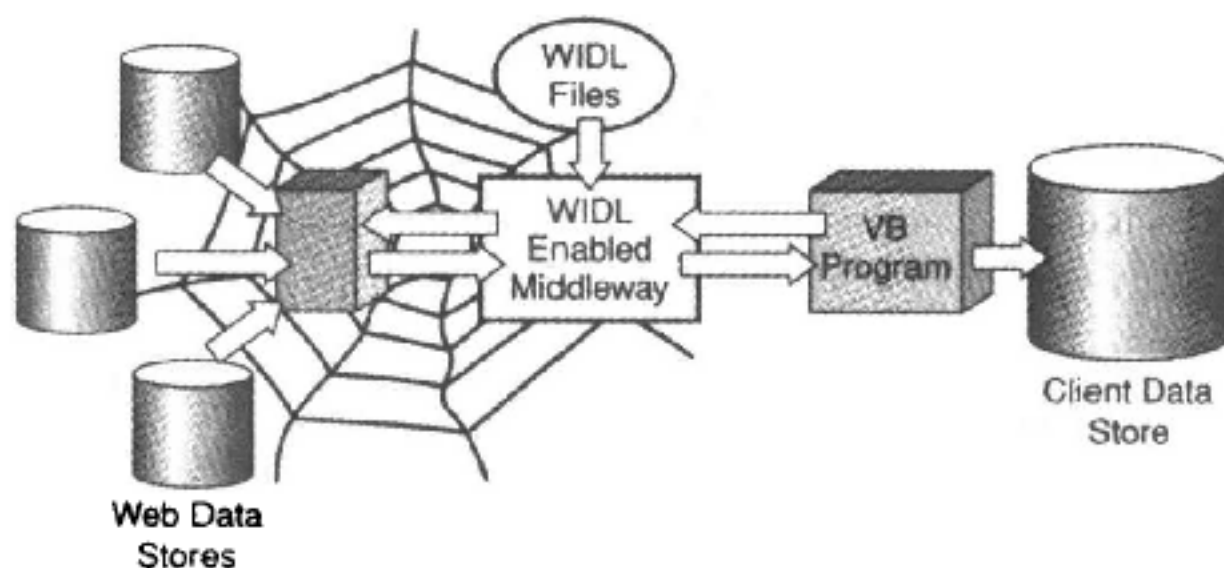


图 24.7 使用 WIDL 软件的计划

下面是用 WIDL 版本 2 编写 WIDL 文件的情况。这个文件被设计用来访问 Mercy 医院的数据库:

```
<WIDL
  NAME="mercyhosp"
  BASEURL="http://www.mercyhospital.org"
  VERSION="2.0"
>

<SERVICE
  NAME="getlab"
  METHOD="get"
  URL="labs/smith"
  INPUT="testinfo"
  OUTPUT="testresult"
/>

<BINDING NAME="testinfo" TYPE="INPUT">
  <VARIABLE NAME="patientname" TYPE="String"/>
  <VARIABLE NAME="testtype" TYPE="String"/>
  <VARIABLE NAME="testdate" TYPE="String"/>
</BINDING>

<BINDING NAME="testresult" TYPE="OUTPUT">
  <VARIABLE NAME="testresult" TYPE="String"
  REFERENCE="doc.tr[* patientname * testtype * testdate].td[4].text"
/>
```

```
</BINDING>
</WIDL>
```

WIDL 有一个必需的 NAME 属性：

```
<WIDL
  NAME="mercyhosp"
  BASEURL="http://www.mercyhospital.org"
  VERSION="2.0"
>
```

BASEURL 给出了 Mercy 医院实验室的基本 URL 地址。SERVICE 元素告知中间件需要的服务。在这个例子中使用的所有 SERVICE 元素的属性都是必需的：

```
<SERVICE
  NAME="getlab"
  METHOD="get"
  URL="labs/smith"
  INPUT="testinfo"
  OUTPUT="testresult"
/>
```

URL 中的地址与基本的 URL 地址相结合以启动中间件获得相应的文件。INPUT 和 OUTPUT 属性引用了被绑定的将要使用的中间件。

一个输入绑定提供了用户填写的表格中相同种类的信息。既然这样,这种信息就是病人的姓名、约定检查的种类和约定的检查时间：

```
<BINDING NAME="testinfo" TYPE="INPUT">
  <VARIABLE NAME="patientname" TYPE="String" />
  <VARIABLE NAME="testtype" TYPE="String" />
  <VARIABLE NAME="testdate" TYPE="String" />
</BINDING>
```

要注意这一点,就是 Mercy 医院实验室为每个医生提供了一个单独的文件。这个信息对于在文件内查找方面并不需要。然而,它要在搜索文件时使用。

OUTPUT 绑定是中间件需要从文件挑选出的信息：

```
<BINDING NAME="testresult" TYPE="OUTPUT">
  <VARIABLE NAME="testresult" TYPE="String"
  REFERENCE="doc.tr[* patientname * testtpe * testdate].td[4].text"
/>
```

在这个例子中,一个私有的文件对象模型被使用,并作为通配符与一个正则表达式进行比较。REFERENCE 告诉中间件在它的字符串内容中定位 tr 元素,这个元素包含着病人的姓名、约定检查的种类和约定的检查时间：

```
doc.tr[* patientname * testtpe * testdate].td[4].text
```

为了要返回 tr 元素的第四个 td 元素的文本内容：

```
doc.tr[* patientname * testtpe * testdate].td[4].text
```

中间件必须理解 WIDL 语法并且必须有文件对象模型(DOM)的知识,DOM 被用来从 Mercy 医院实验室服务器中获得信息。由于本章的编写目的,你不必过多的关注所使用的文件对象模型的准确类型。

现在,中间件要传送四项信息给你的数据库的 VB 前端程序。所有的这四项信息实际上都是字符串。

这一切的好处就是同一个中间件可以被用来访问任何的网页和任何的 Web 服务器。而所需要提供给中间件的全部东西就是一个正确的 WIDL 文件。

当然,这个例子非常的简单,而使用 WIDL2.0 可以执行更为复杂的任务。

24.2 WIDL 2.0 简介

WIDL 规范由 webMethod 编写,并描述了开发 B2B 中间件服务的文法。前面使用的 DOM 的语法是由他们的中间件执行的语法。

当然,没有什么可以阻止程序员编写自己的中间件。同时由于 WIDL 语法已经被公布,没有什么可以阻止程序员使用 WIDL 文件来驱动中间件。如果必要的话,程序员可以开发自己的语法来驱动中间件。重要的是这种处理内部使用的原理。也就是,在 Web 和你的 VB 应用程序之间的一个接口定义语言(如图 24.8)。

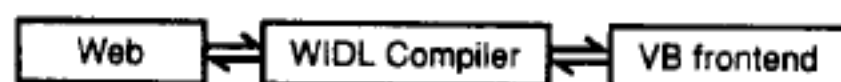


图 24.8 在 Web 数据、基于 WIDL 中间件和客户端前端之间的交互作用规划

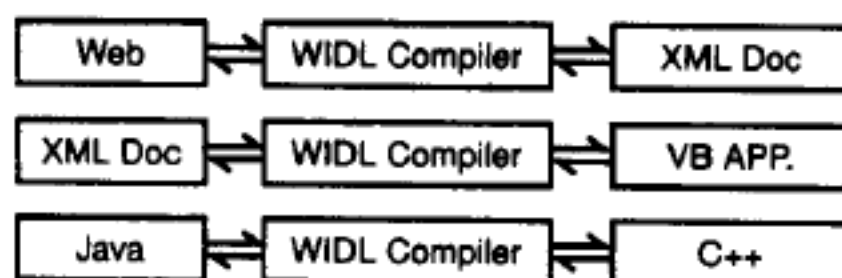


图 24.9 WIDL 编译器作为不同应用程序之间的接口

没有什么可以阻止你使用 WIDL 编译器作为文件和应用程序之间的接口。每个应用程序的存根以 XML 文件格式书写。重要的是,WIDL 编译器是用于多个目标的一般软件。

注意:使用 IDL 的应用程序将提供一个外部接口,这个接口描述它使用的方法以及它的属性(特征)。这个概念在第 15 章“使用文件对象模型(DOM)”中已被广泛的应用。总体上,这些方法和属性被称为存根。

虽然没有什么可以阻止你编写自己的 IDL,但使用一个经过广泛验证的语言也许更为明智。

WIDL 的公开版本可以在 <http://www.w3.org/TR/NOTE-widl> 中找到。

你已经看到了 WIDL 文件的简短例子,它使用了 Smith 和 Brown 医生的例子。下面是更为正式的语言描述。

WIDL 描述和自动化了应用程序以及链接在国际互联网、内部网或外部网的网络资源。惟一的要求就是应用程序是网络可用的。这就允许了使用不同方法的 business-to-business (商家到商家)系统的集成可用。

24.2.1 对象模型

为了使用 WIDL 在网上成功的交互文件,中间件应该有多种模型性能。下面是所有需要的:

- HTML 解析
- XML 解析
- 文本模式匹配

W3C DOM 提供了解析 HTML 和 XML 文件的模式,如果你要处理编写良好的 HTML 文件或是 XML 文件(XML 文件必须编写良好!),这非常简单。然而,如果你处理书写欠缺的 HTML 文件(多数都是这样的文件)和文本文件,中间件必须使模式匹配。

24.2.2 WIDL 元素

WIDL 是 WIDL 语言的根元素。每个 WIDL 文件都定义一个有着多种服务和绑定的接口。

WIDL 元素可以将 SERVICE 和 BINDING 元素作为子元素。它有如下的属性:

NAME

这是一个必需的属性,用来设置接口的名称。

VERSION

这是一个可选属性。在本章中,你所研究的 WIDL 的版本号为 2.0。WebMethods 当前在它的 business-to-business 系统上使用的是 3.0 版,但这个版本还没有公布。

TEMPLATE

许多接口可以被不同的服务器和网站使用。例如,你的约定检查 WIDL 文件可以扩展用于多个网站上的页面。这一点可以通过使用 TEMPLATE 属性实现。

BASEURL

也是一个可选属性。如果需要,完整的 URL 可以放入到 SERVICE 元素中。而 BASEURL 属性可以用来描述基本地址,这样在服务中只使用相对路径名就可以了。

OBJMODEL

有大量的对象模型可以被使用,中间件要理解多个模型。这个属性的值描述了被这个 WIDL 接口使用的 DOM。

值可能包括 wmdom (Web 对象 DOM)、w3domxml (XML 一层的 W3C DOM)、w3cdom2xml、w3cdom1html、perlre (Perl 正则表达式)、mydom 等等。重要的是中间件要理解使用的 DOM。

24.2.3 SERVICE 元素

SERVICE 元素是 WIDL 元素的子元素,它描述了一个 Web 服务,获得一组输入参数并返回动态生成的 XML 和文本文件。它们本身就可以传给其他的应用程序。

NAME

这是必要属性,用于命名所提供的服务。

URL

这也是必要属性。用于命名用来产生服务的网页的 URL 地址。如前所述,它可以和 BASEURL 联合使用。

METHOD

这个必要属性的可能的值就是 get 和 post。描述了将信息发送给 Web 服务器所用的协议。

INPUT

BINDING 元素的 NAME 属性的值可以用来定义调用 WIDL 接口的这个服务程序的 input 参数。

BINDING 元素必须在同一个 WIDL 文件中。

OUTPUT

也是一个必要属性。定义了调用服务的程序的输出参数。

AUTHUSER 和 AUTHPASS

这两个参数是可选的。用于设置 HTTP 验证的用户 ID 和密码。

TIMEOUT 和 RETRIES

这些参数也是可选的。用于定义服务终止之前的时间和重试服务的次数。

24.2.4 SERVICE 链

SERVICE 元素可以链在一起。每个服务访问一个单独的 URL 并从那里获取信息。如果信息实际上是在另一个 URL 上,就像在搜索程序中经常遇到的一样,这个 URL 可以传到同一个 WIDL 文件的另一个服务中。换句话说,一个服务可以用来为另一个服务获得信息。在前面的例子中,Hippocratic 实验室为每个病人的结果提供了一个单独的 URL。

第一个 SERVICE 将获得病人文件的 URL 地址:

```
<SERVICE
  NAME="patientsearch"
  METHOD="get"
  URL="hippocraticlabs.com/pname/index.htm? firstletter=b"
  INPUT="patientname"
  OUTPUT="pnamurl"
/>
```

这个输出 BINDING 将在下面的 XHTML 的行中寻找 URL:

```
<a href="july99.htm? name=brown-j">Brown Jane</a><br />
```

下面是一个 BINDING 的例子：

```
<BINDING NAME="pnameurl" TYPE="OUTPUT">
  <VARIABLE NAME="pturl" TYPE="string"
    REFERENCE = "doc. a[ * patientname * ]. href. value"
  />
</BINDING>
```

它将返回值 july99.htm? name=brown-j。

另一个 SERVICE 可以被用来访问这个 URL：

```
<SERVICE
  NAME="getresult"
  METHOD="get"
  URL="hippocraticlabs.com/%pturl%"
  INPUT="patientname"
  OUTPUT="pnamurl"
/>
```

这样，需要的信息就可从这个 URL 中提取出来。

24.2.5 BINDING 元素

BINDING 元素是 WIDL 元素的子元素。它定义了服务的输入和输出变量。它有两个必要的属性，NAME 和 TYPE。

NAME

NAME 属性简单的提供了绑定的名称。与其他的 NAME 属性一样，它的值在 WIDL 文件中必须是惟一的。

TYPE

TYPE 属性可以有两个取值，INPUT 或 OUTPUT。

BINDING 的 INPUT 类型表明数据由网络资源提供，它就是那种类似于用户填写的表单一类的资料。

BINDING 的 OUTPUT 类型表明元素或文本是从返回自网络的文件中提取出来的。这个文件作为提供给 INPUT 参数的结果从网络返回。虽然如你所见，在大多数情况下这将会是一个完整文件，但 DOM 表达式仍被用来返回文件的一个子集。

24.2.6 VARIABLE 元素

VARIABLE 元素用来描述输入和输出 BINDING 元素的参数。将被使用的属性的类型取决于要被描述的参数的类型。

VARIABLE 元素是 BINDING 元素的子元素，BINDING 元素的变量由它来描述。

NAME

这是一个必要的属性。它用来识别元素。同样，它的值在 WIDL 文件中必须是惟一的。

TYPE

它指定数据的类型。它既可以是一个字符串变量,也可以是一个一维或二维数组 (String[],String[][])。

REFERENCE

它与 BINDING TYPE='OUTPUT' 元素共同使用。它识别使用 WIDL 文件的 DOM 的文件的一部分。通常以字符串形式返回的是节点的值。

MASK

它被用来与模式匹配结合,剥离所有环绕在数据元素周围的任何无用字符串。

NULLOK

通常,如果什么都没有返回,中间件就会产生一个错误。如果 NULLOK 被置为真 (true),错误就会忽略。

24.2.7 CONDITION 元素

CONDITION 元素作为 BINDING TYPE='OUTPUT' 元素的子元素使用。它用来确认绑定是否成功。同样的,它允许在 WIDL 服务中使用分支逻辑。如果第一个绑定失败了,它可以提供一个可选绑定。它也可以用来创建一个不同服务的链。你在“SERVICE 链”一节已经学习了这方面的知识。

TYPE

这是必要属性,它可以取值为 success 和 failure。在这个例子中,你已经设置了一个操作来确定病人的名字是否被找到。

```
<BINDING NAME="pnameurl" TYPE="OUTPUT">
  <CONDITION TYPE="FAILURE"
    RESONTEXT="Patients name not found"/>
  <CONDITION TYPE="SUCCESS" SERVICE="getresult"/>
  <VARIABLE NAME="pturl" TYPE="String"
    REFERENCE="doc. a[ * patientname * ].href. value"
  />
</BINDING>
```

RESONTEXT

这个文本在绑定失败和其他不成功的情况下传给服务。

SERVICE

这是服务的名称,如果绑定成功,服务就可以被调用。在前面的例子中,getresult 服务在找到病人姓名后被调用。

24.2.8 REGION 元素

REGION 元素与 CONDITION 元素类似,它是 BINDING TYPE='OUTPUT' 元素的子元素。它用来指定在一个大型文件中的搜索区域。例如,它可以用来搜索在 1995 年的检

查结果。其有下面的属性：

NAME

与所有的 WIDL 元素中的 NAME 一样，它必须给定一个惟一名称。

START 和 END

这些属性是必要的。它们获取 DOM 引用的值，并指定搜索从何处开始，从何处结束。

24.3 WIDL 3.0

WIDL 2.0 定义了一个指定接口和接口到网站的规范映射的独立语言。WIDL 3.0 将接口和文件映射接口的规范放置在指定的 XML 文件中。这是 webMethods 当前用于 business-to-business 中间件的接口。不幸的是，现在还没有公开 WIDL 3.0 版本，因此它还纯粹是一个私有规范。WIDL 3.0 的说明已经由 Joe Lapp 编写，它可以在 Goldfarb 和 Prescod 编著的第一版的 XML 手册中找到。实际上，webMethods 正在从这个规范中发展出中间件图形化基本接口结构。由于它是私有规范，本章也就无法讲的更多了。

24.4 总 结

本章讲解了在尝试自动接受 Web 上的信息时，你可能会遇到的一些问题。同时，对这个问题给出了解决方案。

WIDL 是 webMethods 开发的私有 XML 应用程序。WebMethods 将它应用在它的中间件产品中。在全面发布的 WIDL 2.0 中使用了 DTD。

程序员可能希望使用这个应用程序作为创建他们自己的中间件的基础，或者他们会希望探究由 webMethods 提供的已实现版本的使用方法。

第 25 章 EDI 和 XML

与网络的公布应用程序共同使用,一些 XML 技术的“冲浪”机会已经在覆盖 Internet 的电子商务中出现。这就是说,网络电子系统是 XML 非常适用的领域之一,这有很多意义重大的成功事例可以作为证明。在 Web 商务中,内部使用 XML 设计和实现数据管理可能是首先实现的。实际上,现在有很多开发商将这样的系统用于他们各自的应用领域中。又有很多种途径可以使用 XML 创建 Web 商务系统,并且可以肯定将有不只一种的方法取得显著的成功。

在讨论这样的系统的时候,通常都会讨论 EDI 的论题。EDI 自 1970 后就出现了,它用来在 Internet 前身的通讯互联网上交换电子商务信息。在 EDI 和 XML 之间存在什么样的协作? EDI 是一种 Web 商务的最好模型吗?

25.1 EDI 初步

EDI 准确来讲到底是什么? 作为“Electronic Data Interchange”(电子数据交换)的缩写,它容易让人糊涂。这个名字的范围太大了,而 EDI 的实际意义要比这个范围小的多。EDI 主要被大的机构使用来管理他们的供应关系,而它只是被设计用来记录那些非常特殊的公司到公司(business-to-business)关系。它远远达不到对所有种类关系的支持,这样将使 EDI 结构极大的复杂化。由于潜在的混淆,大多数人在谈到包含在电子通讯网络上交换数据的其他应用程序时,都使用不同的术语(例如非常广泛使用的“电子商务”)。

这是有关 EDI 和 XML 的第一节课:XML 的潜力要比 EDI 的潜力广阔的多。将 XML 作为一种主要工具,而把 EDI 作为特殊应用程序是很有益的。此外,EDI 使用这个工具的不同版本。

25.1.1 历史与现状

EDI 在 70 年代被开发,它在某些行业中还被广泛的使用。它被设计成与现在被大多数 Web 应用开发者所使用的不同的商业和技术假设的集合。我们将分别探讨一下这些假设:他们深深影响着 EDI 技术和相关应用的发展方向。

EDI 的一个重要方面就是,实际上它有两个相关的标准。一个是 X.12 系列标准(来自 ANSI——美国国家标准化组织),它在美国国内大量的使用。另一种标准是 EDIFACT (Electronic Data Interchange for Administration, Commerce, and Transport, 管理、商务和运输的电子数据交换)系列标准(来自联合国),它在世界的其他地区使用。虽然它们使用了一些共同的基础技术,但 X.12 和 EDIFACT 消息通常不能混合。

虽然还有一些可以处理 EDI 所处理的一些问题的其他技术,但它们不能与 X.12 或 EDIFACT 系统通过应用层网关技术通话。例如,转化软件有时要用来将 EDI 消息转化成文本文件,这样就可以将消息传真给无法交互的 EDI 应用程序使用了(消除基于纸张的处理

过程是大多数 EDI 系统的目标,在 EDI 网络中,并不是每个机构都可以达到这个目标)。

多个标准的存在通常被认为是一个缺点,对于那些希望在他们的 EDI 操作上使用单一标准的组织来说,更是如此。然而,对另外的一些组织来说,则会有不同的权衡。

25.1.2 EDI 的商业特征

理解 EDI 的最好方法之一就是根据它所支持的商务关系的类型来理解。这些关系推动着 EDI 组织的发展,它们的真实价值使得如此多的人对它发生兴趣。除了那些有关实际消费投资的技术,实际的技术对这些商务在某种程度上来说是次要的,它很难被取代。除非与其他多种技术和一个服务基础相结合,XML 本身是不能取代 EDI 的。

这些商务关系在 EDI 中作为消息的模式显示。消息通过预先设定的关系在公司间交换,典型的关系就是买方和卖方。下面很重要:模型是不公开的,相互作用并不向所有的来者开放。消息用来启动那些关系的某些方面,比如提供产品详细目录和管理运输后勤。

一个 EDI 消息交换过程中有三个基本的工具:

- 消息发送器。例如,一个供应者可能发送一个预定出货通知,说某一个指定的出货将在下午到达指定的仓库。
- EDI 增值网(Value Added Network——VAN)。它负责递送消息(通常这立刻就可完成)。
- 消息接收器。例如,消费者需要知道出货里有什么,这样仓库职员可以计划在卸货时将出货放在那里。

EDI VAN 的角色是重要的。这个角色并不是 Internet 的,因为在那里网络只是接收包(这样的包交换技术直到 EDI 被开发出来后才得以被广泛使用)。作为代替,EDI VAN 是一个可以接收和排序消息以延期发送的“存储与发送”网络。

VAN 的“增值”来源于它将消息在甚至容器还不能立即可用的时候放入容器的途径。在某些方面,VAN 是服务提供的一部分,这个服务包括通向无法交互商务消息系统的 EDI 通路。VAN 在发送方和接收方解决争端方面也能够起作用。如果它说没有接收到订单,这就可以巩固供应者的主张。另一方面,如果它说接收到了来自供应者的对这个定单的确认,这就可以巩固接收者的主张。由于 EDI 网络常常是被用来消除传统的纸张记录的,这个可信任第三方的角色可以在许多商务关系中被验证。所有这些 VAN 角色的代价是昂贵的。EDI VAN 的运作花费非常大。

这里我们不能探讨所有的成千上百的 EDI 消息类型,因为我们的兴趣并不在这里。理解下面的一点非常重要。就是这些消息是为了某一类的商务关系而设计的,比如汽车制造商或国立折扣店这样的需要管理成百上千甚至成千上万的关系,或类似的,一个政府购买机构。这样的商务结构是 EDI 支持的核心:有着多个供应商的大量消费者结构。

另一个启动这个关系的方面就是启动的任务已经被设计成完全自动化的过程。人类的干涉只在处理一些错误的时候需要。如果启动的任务不适合指定商务的需求,就必须协商出新的消息。这可能不是一个快速过程。

25.1.3 技术描述

当 EDI 被开发时,VANs 使用了具有划时代意义的技术:使用定制协议的拨号调制解

调制器或某些情况下的专线。虽然已经有了更高数据传输率的线路,这样的技术仍然在使用。当 EDI 最初被开发出来时,许多调制解调器支持约 1200 波特的传输速率;而现在的调制解调器的速度成量级的提升。

在过去的几年中,有一些关于“Internet EDI”的运动,在之中使用 TCP/IP 协议作为 EDI 参与者和 VAN 联合的纽带。这就出现了 TCP/IP 中的卖方杠杆存在的投资并提供了在消息传输速率上的实质性提升的潜力。

EDI 技术的最根本的方面之一就是使用的消息编码。当你正在使用很难达到 1200 波特的调制解调器工作的时候,在一条消息中多加的十个字节意味着增加了一秒甚至更长的传输时间。因此,编码使用了多项技术以达到一定水平的节省。消息的格式是二进制的,因此比特率是一项设计要求。

有了现在的网络技术,这些编码的复杂性被广泛的认识到是一个问题。这样编码的动机已经过时了。然而,如果不破坏现有的 EDI 网络,这样的编码还不能轻易的改变。

25.2 EDI 有什么问题

在前面的讨论中,按照现在被广泛支持的不同的展望,我们已经谈到了 EDI 的一些基本问题。但我们还要懂得在 EDI 中也有着很多正确的东西。它是一个被广泛开发的技术,可工作并节省资金。那些问“有什么问题”的人也正在从更加现代的角度做着这件工作,这样公司就可以采用高质量的安全的 Internet 连接。今天,多种多样的商务关系转移到了网上,Web 应用程序被充分的开发。

25.2.1 商业问题

EDI 关系趋向于依赖一方的优势来使事务工作并指定交互中的商业和技术标准。首先这是使得它工作的部分原因,但是这样的“庞大系统”的实现途径和偏斜政策并不总是当今所希望的。

特别是,由于网络技术对商业模型支持甚少,这就带来了网络组织的蓬勃成长。以往的需求所关注的通信和控制问题,通过使用更为有组织的商业结构已经可以解决了。这个改变是网络革命的本质体现,它与“优势方”模型的对抗带动了 EDI 的成长。

事务处理模型

使用 EDI 第一个要考虑的商务问题就是它的主要听众是由大的企业集团组成的。经典的 EDI 应用程序是一个 hub-and-spoke(中心与对话)系统,大的公司在中心,许多小公司与这个中心对话。通常,这些小公司加入 EDI 网络的惟一动机就是它是要与某个大集团进行商务活动的需求方。这些大的集团获得了大部分的直接利益。

相反的,EDI 并不真正处理其他类型商务的需求。它并没有被设计用来为各类的有着许多新的(并且通常是短期的)商务关系和多种付款方式的在线组织工作,而这是 Internet 的基本要求。EDI 并不是一个前端技术。它是后端技术,它需要可分离的前端以支持其他的商务类型。

许多人惊讶于 EDI 并没有打算处理公司间的其他商务关系,这可以带来极大的自动化。但是并不是所有的 business-to-business(企业对企业)关系包含一个大公司和一个标准的

事务处理,许多关系包含着部分的或是完全的用户化情况。并不是所有的公司希望自动化他们所有的交互关系。许多公司更喜欢只是部分的自动化,并在某些或是全部的事务处理中包含人的工作。甚至对于那些希望将他们的事务处理全部自动化的公司,EDI也许并不是最佳解决方案,因为如果不开发新的消息,它就不能充分灵活的处理公司的特殊需求。EDI毕竟并没有为小公司进行优化。

花费

投入EDI系统的整体花销是实质问题。部分来说,这可归于如下EDI系统的目标:协助管理大的获利业务,这在定义上需要巨大的花销。EDI网络的软件、硬件和网络连通性是专业性强并且昂贵的。

EDI系统在运行和维修上也是耗费巨大的。先前涉及的主动Internet EDI是通过在Internet基础构造中的杠杆存在型投资减少消费的部分结果。最初的EDI VAN基础构造并不是多用途的,它相对于Internet技术的进步是滞后的。现在,公司只通过展开Internet链接就可以获得更大的利益和更好的成绩。

多标准的出现也是一个花销的问题。这关系到X.12/EDIFACT的分歧。每个行业都有着它自己的不同的EDI消息,因此技能和工具并不是总可以转换的。这就意味着在与一个新的卖主联系的时候,如果这个卖主提供另一种行业服务,这就会带来令人吃惊的花费。

25.2.2 技术问题

EDI技术的年龄也是一个不可避免的问题。这是一项陈旧的技术,许多新的参与到广阔的电子商务领域的新成员都不希望使用它工作。这个问题是商务和技术两方面的反映。

什么使得EDI成为了一项“陈旧”的技术?一个事实是EDI VANs并没有采用Internet样式化技术。他们在更普遍的Internet基本服务方面,不能提供引人注目的优势。由于其他到这样的VANs的访问途径对现有的用户基础来说还是必需的,因而Internet EDI可以部分的解决这样的问题。它是一项边缘的改进,而不是技术上的根本改变。另外,它不处理长期以来使用的低速网络的结果。

使EDI觉得“陈旧”的最重要的原因之一就是它的协议设计用来区分比特率优先的途径。在20/20标尺上,这样的方法被称为“看似有点聪明,实则非常愚蠢”,这是因为与这样的比特填充相关的花费已经完全超出了它能够带来的利益。这些花费的出现是因为解释那些消息的软件非常复杂,因此它是易错的并且昂贵的。为了避免这种问题,只有有限的卖方满意提供用于所给EDI应用的软件。这方面并没有很多的价格竞争。

大多数近十年设计的应用级协议已经作为开发人员优先使用的协议。在Internet上最广泛使用的协议(用于email、Web、NetNews等等)都是基于文本的。文本协议比二进制协议更容易调试,这是因为开发人员可以直接的读取消息。极少有开发人员愿意解读十六进制消息,或是能够不出错的做到这一点。任何编程环境都可以支持基于文本的消息。当你评估系统花费时,很明显二进制协议对应用来说更为低效,即使它们有时可以节省空间也是如此。二进制协议极度的缺乏灵活性,而在其他领域又没有显著的提高。

简而言之,低级EDI技术在一个变化的商务环境中没有充分的灵活性可使其进行迅速的改变。这种灵活性是当今技术的重要商务需求。

25.3 与 EDI 接近的 XML

如果 XML 假设被用来协助电子商务,在 EDI 的情况下,它要如何工作呢。这是一个令人头疼的问题,因为这个大多数人要处理的问题并不属于 EDI 组织有兴趣处理的问题。

这一节首先讨论一下包含 XML 的网络技术如何处理与 EDI 基础结构相同的多个问题。接下来,我们将讨论一下 Internet 组织的一些更广泛的问题,包括简要讨论一些主动 XML 激活的问题。最后,我们将返回来讨论 EDI 的特殊问题。

就像你所读到的,要谨记广义的“电子数据交换”问题确实关系到如何使用 Web 技术构造新的商务组织。根本的问题是与商务过程有关的,而不是与技术相关的。EDI 组织已经创建完成了,它处理一组有限的问题集合。因此,现在它将面对另一个不同的问题,它如何选择:怎样和是否要彻底改造自身。

25.3.1 基础技术

当今 Web 的一大特点就是开发工具被广泛的使用。当然,以客户为中心的工具已经大量用来编辑 Web 页面,它们有的使用可视化客户端编码构架(使用如 Java 或 Javascript 这样的语言)。更重要的是,客户端和服务端支持通过 HTTP 交换数据的代码库——也就是读取和书写 XML 文本——本质上这可以被所有的操作平台使用。

这种广泛的应用是网络化应用环境的独特一点。主流的公司“在 RPC(Remote Procedure Call——远程过程调用)战争”中进行了十年甚至更久的战斗,尝试使诸如 ONC、DCE、COM 和 CORBA 协议这样的二进制 RPC 协议达到应用水平。今天,这些协议没有一个作为简单 HTTP 协议使用。EDI 组织也萎缩了。网络协议普遍的存在,新的类型的应用程序相继出现,这已经创建了有大量投资经济上的“网络现象”。XML 是这种现象的受益人,因此网络组织协议要使用这项技术。

Email 存储与转发消息

在 EDI 中,VAN 作为存储转发(store-and-forward)网络使用。这与 email 网络是同样的角色。SMTP 服务器用来转发消息,在传输中进行暂时保存,使用诸如 IMAP 和 POP 这样的协议来访问存储的消息。

应用程序可以使用 email 通过 XML 内容发送消息。通过与其他应用层协议和安全机制的结合,递送确认可以被用来在必要的时候建立对消息的确认。

基于 Web 的请求/响应消息

Web 服务器上使用 HTTP 是常见的。它有一个客户端——Web 浏览器——建立对服务器的请求,它可以直接递交请求数据(HTML、XML、图片等等),调用插件(用于 PDF 文件或多媒体表述)或是将它存入磁盘。

这并不是使用 HTTP 的惟一方法。作为核心,HTTP 是一个简单的基于文本的请求/响应协议,它规定了大多数的数据消息体。当你使用 HTTP 中的 POST 命令时,请求和响应都有这样的一个主体。

这些消息可以是任何东西:特别是,通过采用协定,客户端和服务端可以交换高级请

求和响应。这个协定可以是“这里是我们用于交换的XML文件类型”，或者它可以使用某种非XML数据格式。你可以在类似于远程过程调用的多个协定中选择一个。更典型的是，你可以选择在交换 workflow 文件方面有很多功能的协定。我们将简要的讨论这个问题；要谨记，即使在使用XML时，这样的协定也有很多。

在最流行的编程语言中，软件已经可以很容易的让你制作一个HTTP POST请求了（作为一个客户端）。作为它们支持动态创建内容的一部分，大多数的Web服务器将让你在你所喜欢的编程语言中运用这样的POST请求。Java、Perl、C/C++、TCL和其他大量的语言普遍支持这个功能。如果你还有其他的软件来运用XML，这两个的性能——HTTP RPC的客户端与服务器端——是你创建请求/响应消息的真正需要。现在甚至有的XML服务器产品被设计作为中间件平台，以XML格式公布数据并从多个资源中结合这样的数据。

你可能听到了一些对HTTP作为RPC协议的批评，这是因为HTTP使用的是可变长度文本而不是固定长度二进制编码。虽然文本轻微的提高进程的花销这一点是肯定的，但我们要结合系统其他部分的花销来进行衡量。通常这种编码的花销可以通过应用程序的“实时工作”削减。这种应用程序最起码也需要巨大的额外花销。这就意味着目前文本协议是低比特率的，但这无关紧要。

消息保密和安全

当两个公司在Web上进行商务上的操作时，确保这些消息交换的合理保密程度是首先要关注的问题。这种保密操作在请求/响应消息和存储/转发消息时都需要使用到。在包含防火网的网络设置中提供这样的保密操作也是同样重要的。当今的机构普遍使用防火墙来保护他们的内部网免遭攻击。

消息完整性是另一个消息安全的讨论点。一个消息的发送器和接收器必须有足够强的担保，确保消息没有被篡改过。这是因为，在很多情况下它们没有硬件备份记录来指出是否有疑点存在。与完整性相关的就是真实性，它要理性的安全的确认是谁发送的初始消息。

Web的非常有价值的特性之一就是与其他网络构架相比，它提供了更大范围的安全特征：

- HTTP的HTTPS(HTTP Plus Secure Sockets Layer or SSL)变量提供了对交互网络消息的保密功能。
- 在保留秘密、保护共有网不受开放、免于Internet上多种的威胁的情况下，HTTPS可以按“隧道式”穿过网络防火墙。
- 数位标志的消息可以使用PGP或S/MIME发送。它们提供了一种方法，为RPC样式和存储转发消息安全的识别它们的发送器并确认消息的完整性。通过运用可选择的编码技术，使用HTTP或SSL还可以提供保密性，并可在传输层处理保密性问题。

为什么与Web和电子邮件协议的外部网络安全相关的特性这么难以达到？

简短的回答就是美国政府在这方面有政策，很大程度的阻碍了密码系统实现保密。这些政策使得在美国和全球出售使用任何形式密码技术的软件变得昂贵而又困难。在美国国内发行强大的密码系统所需要的成本是花费巨大的。取得出口许可的脆弱版本的许可甚至更为昂贵（那些脆弱版本实际上已经被破解了，只提供无效

的对有实际价值秘密的保护)。

网络的影响已经抑制了密码技术在软件系统上的应用和实现保密。代价是包括政府和私人,公司和个人在内的各方对于间谍组织都是完全公开的。通常,公司在他们可以选择的情况下,不会选择这样的系统。

在 Web 上,HTTP 和 SSL 的结合——称为 HTTPS——受益于脆弱系统和强壮系统的普遍使用。主要的浏览器提供商在取得美国出口控制代理上进行了花费昂贵的工作,而这些浏览器产品的巨大成功令人吃惊的获得了大多数的客户。国际公司提供了他们自己的强大版本,而其余的也就成为了历史;现在我们就可能在世界的任何地方访问这样的“强大的密码机”。如此多的人开始谈论 HTTPS 也就说明了这样做是符合大众需求的。没有任何的其他 RPC 协议达到了发行安全版本的目标。使用 HTTP 更经济一些。

现在有合法的努力正在试图推翻这些政府的政策。我们可以浏览 <http://www.eff.org> 和 <http://www.epic.org> 来获得更多的信息。

基于 XML 的消息格式

很明显在 Internet(和 Web)技术的菜单中有足够数量的技术来构造完整的系统用于交换 XML 编码的数据。当需要或者要基于资源的实用性进行排队时,这些工作可以实时的安全的实现。下一个出现的问题就是数据如何进行编码。与它相结合的问题就是第一个问题应该如何解答。

考虑一下各大机构已经使用多少种方法来交换数据。这些方法中的一部分通过某些类别的工业协议被高度的形式化。另一部分方法来自于少数组织间的非正式的特别协议。一些消息可以用于多种形式,而另一些则只适用于一种。虽然在某些情况下这么多种类的格式是十分麻烦的,但这样通常是值得的甚至是基本需求。不同类的交互需要不同的格式,对于不同组织间的同一类交互,这也是需要的。

也就是说,有很多的机构正在定义 XML 协定来编码商务数据。没有任何一种协定可以达到世界统一,它们中的一些已经被广泛的使用了。实际上,由于现在有太多这样的机构,这里我们只能提及最知名的一些:

- CommerceNet(<http://www.commerce.net/>)长期参与制定用于 Internet 上的电子商务解决方案。
- Rosettanet(<http://www.RosettaNet.org/>)是一个新的相关组织,建立在处理连接完整性问题的基础上。
- BizTalk(<http://www.biztalk.org>)是 Microsoft 发起的,与 Microsoft 的 BizTalk XML 服务器有关的组织。它使用现在正在开发的一些 Schema 系统中的一个。
- 其他制造商的有关 XML business-to-business 提供的链接工作流的解决方案也正在开发中,比如来自 SAP 的方案(<http://www.sap.com>)。

更显著的成就之一就是通用商务函数库(Common Business Library,CBL)。这是一个包含可重复利用元素定义和可以用多种方式重新组织的文件构架库。EDI 在 CBL 开发时获得了很多的关注,CBL 有更广的适用性:它可以被小的机构使用,通过在它们 business-to-business 关系中使用 XML,将业务从简单的 Web 应用移植到更广的网络。CBL 可以被上面

所列出的所有面向商务的组织支持(当前的版本是 2.0)。它可以通过主要的 XML 库来使用,这些库包括 <http://www.xml.org> 和 <http://www.commerce.nt/>。

Schema 和 XSL 的角色

XML 文件类型声明(DTDs)是定义 XML 消息语义的脆弱工具,这是因为它们不提供数据类型,面向对象类型继承和其他被广泛使用的抽象化设计。它们也不能处理文件设计问题,这些是维护和发展大系统的基础。为了处理这些局限性,我们开发了 Schema 系统(具体介绍请看第 5 章“XML Schema 基础”和第 6 章“XML Schema 构造技术”)。

目前,多种 Schema 系统使用的情况已经存在了很长的时间。其中一部分高度面向对象(例如 SOX),一些更加面向相关(比如一些初期的 Xschema 提议),而一些系统更加面向文件。在本书编写期间,W3C 有起码 4 种 Schema 建议(一些是记录,而一些是工作草案)。一些人相信没有单一的 Schema 系统可以占支配地位,理由就是在计算机技术的其他领域中的没有单一的设计方法可以占支配地位。

使用 XML 交换商务消息的结果就是开发工具和模型还不能轻易的实现可转换。简而言之,标准依然在变更中,并且即使是在作为协议和标准开始在某些领域中实行的情况下,这个状态可能在以后的几年中还会继续。DTDs 被广泛的支持并可被用来与相同类型的商议协定相结合,作为满足任何需要的 Schema 解决方案。然而,许多的机构设想绕过 DTDs 而直接考虑 Schema 解决方案的一些类型。

由于多个词表和不同 Schema 系统的存在,我们需要有可以在它们之间进行转换的工具。这将是 XML 转换的应用之一:转换文件(或 Schema)从一种构架转到另一种构架。设计机构接受了好的建议,它们计划将它们的 Web 商务基础中的 XSL/T 作为数据处理的工具,以及用 XHTML 公布数据的工具。

25.3.2 网上电子商务

作为一种人与人交流的更好的途径,万维网变得越来越普及。通过它,人们可以通过电子邮件和专业性信息的公布栏进行交流。对于一些种类的产品、特别是计算机软件和数据来说,Web 迅速的成为一个新的销售渠道。对于其他的销售方式,诸如通过邮件方式销售 CD-ROM 或在混凝土的商店中购买来说,这种方式有着独特优势。

商业中对这方面的反映并不慢。它们迅速将 Web 作为一种新的销售、通讯和发布的渠道。一种新的商务可以在 Internet 前台建立并迅速获得非网上业务所需要的部分投资。对于老式的商务来说,重大的问题之一就是如何在这样的新世界中做生意,在这个世界里公司可以更直接的与用户接触。这是消费者所重视的,因为这样可以削减大量的花费。然而,这种花费是从发行人的口袋中削减的。而这些人长期是那些已知商务的重要合伙人。

因此,Web 成为了革新的商务模型生长的温床。发行人只能在消费者监督并满意的情况下提高价格,否则它们就不能在商务中立足。公司可以建立一个前台来对任何他们所选择的商务模型提供支持。

这并不让人吃惊,EDI 假定的模型并不是那些针对消费者或其他新的 Web 商务这样的基于 Web 商务的最普遍模型。EDI 并不直接的处理消费者,它也不处理作为 Web 发展如此迅猛的主要支柱的特殊商务的需求。虽然它可以处理网上超市供应商方面的某些类型的需要,这个模型将隐藏在后台之中。

网上商务系统使用这类新模型的一个例子就是 ICE 协议(<http://www.w3.org/TR/NOTE-ice>)。这个协议用于连接诸如新闻故事和软件的目录的发布。这是一个在 XML 基础上的基于 DTD 的协议,位于 HTTP 头部用来与工业特殊词表联合构造商务 Web。用于交换新闻故事的 XMLNews 规范(<http://www.xmlnews.org/>)是另一个例子,这是一个文本发布的更为专业化的规范(请看第 31 章“用 XMLNEWS 制作新闻”以获取更多信息)。

提供新的 Internet 上的基于 XML 的商务模型的其他成就中还包括开放式贸易协议(Open Trading Protocol——OTP)。这个协议现在由 Internet 工程任务组(Internet Engineering Task Force——IETF)的 OTP 工作组(<http://www.ietf.org/html.charters/trade-charter.html>)处理。这个协议处理自动化的付款协议协商。通过与其他商务协议联合使用,它可以提供基于 Web 的商务交互新方式。

这些成果只是使用 XML 建立网上商务的成就的一部分。EDI 模型是多种商务模型中惟一在网上被广泛开发的模型。

25.3.3 XML 中的 EDI 消息——XML/EDI

当你在考虑如何使用替换的 Web 技术解决 EDI 要解决的问题时,一个解决方案已经迅速形成了。它将保留现有的 EDI 消息集,将它们翻译成 XML,然后使用一个 VAN 的 Internet 变量来交换它们。消息结构将是同样的但是会使用 XML 编码,因此你可以轻松的将这项技术移植到一个 EDI 网络上。实际上对于这类系统还有很多工作要做,包括使用这种 XML 消息的系统的 EDI 网关。

这个方法有着很多的问题。它在哪里真正的增值?这种转换的代价是很高的,它的好处是什么?它能给那些在过去的十几年里使用其他非常不同的商务模式的公司带来 Web 上的利益吗?它可以在有很多 EDI 供应者存在的市场上开拓出足够的市场份额,或者将 EDIWeb 有效的加入到 Web 商务系统中吗?它足够保持相同的商务和规范模型,并可以转换到某些新的底层技术中吗?

这些问题在这里并不能解决,但现在你应该理解一些折衷和选择。EDI 处理的提供综合链接这类的问题正在通过其他的途径解决。我们并不清楚 EDI 组织除了大量的经验和一个庞大的确定应用的 Web 外,还能贡献多少东西。但我们可以肯定,EDI 组织将继续存在一定时间。

将 EDI 商务模型直接转换到 Web 中将不鼓励动态的开发一直推动 Web 迅猛发展的商务过程。Web 的优势一直以来就是灵活性和反应速度。将这种优势引入到 EDI 世界是一个许多人避之唯恐不及的挑战。

Web 和 EDI 组织之间的联合将 XML 作为转换的代理包括进去。由于基于 XML 的消息可以迅速的开发、配置和发展。它们正在促进在线商务的灵活性方法的使用。其中,EDI 消息到 XML 的直接转换,对于许多在“.com”公司还是一个令人好奇的技术之前就从事在线商务的公司来说是一道非常有用的“主菜”,这些公司有需求和财力来将它们的已有系统与 WEB 相连。

25.4 总 结

网上电子商务是 XML 被认为最有价值的领域之一。早先由 EDI 处理的提供综合链接这样的问题现在从众多参与者的角度来看和大量市场的目录中来看有了更新的面貌。

EDI 组织有很多知识可以教授给 XML 组织,但我们不清楚是否那是这些问题基于 Internet 的解决方案的来源。许多处理电子商务的 Web 开发人员将它作为 EDI 所实现的一方面来学习。然而,他们也吸取当今技术的优点,创建比 EDI 更为丰富和灵活的底层,并有着更广的应用和更低的花费。这个结构通常连接 EDI 网络,创建在商务模型类型和技术间达到最佳平衡的混合系统。

第 6 部分 XML 工具

第 26 章 XML 制作和内容管理工具

XML 工具很大程度上使得 XML 如此快的变为每个 Web 开发人员的开发手段的一部分。虽然我们中的一些人还满足于在一个文本编辑器中编写 XML 代码,但更多的 XML 开发人员都期待着出现一种完美的可视化开发工具,使得组织和管理 XML 文件变得更加直观。幸运的是,XML 是 SGML 的产物,并部分的继承了 HTML。这两种技术都存在着丰富的工具。因此,我们有相当数量的专业 XML 工具可以利用。

这一章将对写作本书时一些主要的 XML 工具进行综述。你将学习到用于 XML 的不同种类的工具,这其中的一些非常普及的工具已经用于 XML 的开发之中。虽然本章涵盖了很大范围的工具,但这并不意味着我们已经对每一个所知的 XML 工具进行了彻底的综述。对在仅仅一章中提到的工具来说,没有提及的工具实在是太多了。因此,你可以把这一章作为对当前普及的 XML 工具的指南。

26.1 XML 工具类型

当谈到 XML 工具时,我所说的是这样一些软件,它们简化了创建或处理 XML 代码的任务。无可否认,这是一个非常含糊的描述,但当你认识到支持 XML 的工具有多大的范围时,这一点就是非常必要的。与 HTML 不同,一个 XML 工具的定义非常的广泛,而 HTML 的工具只是与 HTML 编辑器等同的开发工具。这是因为 XML 是一种比 HTML 更为复杂的技术。XML 文件可以结合 DTDs 和 Schema,更不用说还有样式表和复杂的脚本。应该承认,HTML 也可以使用样式表和脚本,但它们并不是 HTML 所必需的。XML 需要这些部分以使 XML 文件可以被显示。

现在有两种主要的 XML 工具:

- 制作工具
- 内容管理工具

XML 制作工具理论上与 HTML 编辑器相当,它允许你在其中创建和编辑 XML 文件。一些制作工具还支持创建 DTDs、Schema 和样式表。XML 内容管理工具用来共享、管理和存储 XML 数据。这些工具主要包括某种类型的数据库管理系统和用于保持文件修改记录的版本管理系统。本章的其余部分将探讨这两种主要类型的 XML 工具,同时还有一些其他你可能会发现非常有用的有趣工具。

注意:你也许注意到我并没有把 XML 浏览器作为 XML 工具的一种类型。从技术

上讲,浏览器是 XML 工具,因为它使你可以浏览 XML 文件。然而,本章是从开发的角度来讲述 XML 工具的,这就意味着工具要能对 XML 文件进行创建和编辑。

26.2 XML 制作工具

XML 制作工具用来创建和编辑 XML 文件。所有这些制作工具中最简单的一种就是文本编辑器。它并不是 XML 专用的但是它还是可以提供一种创建和编辑 XML 文件的方法。例如,我轮流的使用 Microsoft Visual Studio 编辑器和 Windows 的记事本来创作本书中的例子。不管你是否将使用更为完美的 XML 制作工具,你都可以依照你的喜好作任何一件事。

在本书编写时,已经有大量的可使用的 XML 制作工具可用,基本上可满足任何一种需求和预算。下面是我所遇到的一些很普及的 XML 制作工具的列表:

- SoftQuad XMetaL
- Adobe FrameMaker+SGML
- Arbortext ADEPT-Editor
- Stilo WebWriter
- IBM Xena
- Vervet Logic XML Pro
- Bluestone Visual-XML
- Microsoft XML Notepad
- Emile

下面将带你一个个了解上面的这些制作工具。这将帮助你确定哪个工具更适合于编写你的 XML。

26.2.1 SoftQuad XMetaL

XMetaL 是一个由 SoftQuad 开发的 XML 制作工具,它适合于确认用于在 Web 上显示的 XML 文件的开发。它支持使用层叠式样式表(CSS)来提供对 XML 文件的显示支持。XMetaL 还支持 Window 脚本主机(Windows Scripting Host),这是一种 Windows 中的脚本引擎支持诸如 JavaScript 和 VBScript 这样的脚本语言。XMetaL 工具本身提供了一个图形化用户接口(graphical user interface,GUI)来创建和编辑 XML 文件。这个用户接口提供了对 XML 文件的三种不同的显示方式:

- 文本显示——以原始 XML 代码方式显示文件。
- 有标签显示——以 XML 树方式显示文件。
- 普通显示——以 CSS 设计的方式显示文件。

注意:与一些制作工具不同,XMetaL 需要所有的 XML 文件都是有效的,这就意味着它们必须基于 DTD。

SoftQuad 也是 HoTMetaL 的制造者。这个工具是一种用于创作 Web 页面的基于 HTML 的制作工具。它在使 Web 页面方便制作方面的经验影响着 XMetaL 的可用性和设

计。它的用户接口非常直观并在很多方面模仿了文字编辑器的接口。XMetaL 还支持 SGML。这对于存在 SGML 到 XML 转换或在这两者间的内容共享的对象非常的有用。为了保持 XML 的精髓,XMetaL 在数据内容和描述间提供了清晰的区分。

XMetaL 比较严格的地方就是它需要所有的 XML 文件都是有效的。如果你希望使用一个更为灵活的制作工具来创建结构良好的文件而不用于 DTD 结合,那么你就应该尽量避免使用 XMetaL。然而,如果结构是你所希望的,XmetaL 就是一种可靠的 XML 制作工具。要注意 XMetaL 是专门作为一个 XML 制作工具而设计的。这意味着要与一个内容管理系统相结合来提供完整的 XML 解决方案。

XMetaL 当前可在 Windows 98/NT/2K 中使用。要了解更多信息,请查看 XMetaL Web 站点 <http://www.softquad.com/products/xmetal>。

26.2.2 Adobe FrameMaker+SGML

Adobe FrameMaker 是一个流行工具。它用来以电子和打印的方式集合信息。Adobe FrameMaker+SGML 是 FrameMaker 的扩展,它支持创建 SGML 和 XML 格式的结构化文件。实际上,Adobe FrameMaker+SGML 支持结构化和非结构化文件创建。但无疑你希望走使用 XML 的结构化文件的道路。

Adobe FrameMaker+SGML 提供了完美的 GUI 来操作 SGML/XML 文件。一个独特的优秀特性就是你可以打开非法的文件,Adobe FrameMaker+SGML 将指导你修正合法的问题。同时,Adobe FrameMaker+SGML 被设计为适合于整体的 XML 解决方案的 XML 内容管理系统的综合。

Adobe FrameMaker+SGML 适用于 Windows 98/NT/2K、Macintosh 和 UNIX (Solaris, HT-UX, IBM AIX 和 SGI IRIX)。要获得有关 Adobe FrameMaker+SGML 的更多信息,请访问 Adobe FrameMaker+SGML 的 Web 站点 <http://www.adobe.com/prodindex/framemaker/prodinfosgml.html>。

26.2.3 Arbortext ADEPT-Editor

另一个从 SGML 中继承的支持 XML 的制作工具就是由 Arbortext 公司开发的 ADEPT-Editor。ADEPT-Editor 是一个有效的 SGML/XML 编辑器,有着非常用户化的 GUI。通过 ADEPT-Editor 用户接口,你可以以原始 XML 代码的形式、XML 解析树或经修饰的 WYSIWYG 型预览文件这几种方式来浏览 XML 文件。作为最初是以 SGML 工具开发的工具,ADEPT-Editor 完全的支持 SGML 和 XML 之间的内容转换。

在 XML 出现之前,ADEPT-Editor 作为一种 SGML 制作工具被广泛的使用。因此,ADEPT-Editor 是一个强大的工具,专为有经验的内容开发人员设计。作为 ADEPT-Editor 的补充,Arbortext 使用一个 XSL 样式表编辑器来创建样式表以显示 XML 内容。

ADEPT-Editor 当前可用于 Window 98/NT/2K 和 UNIX (Solaris, HP-UX 和 IBM AIX)。要获取有关 ADEPT-Editor 的更多信息,请访问 ADEPT-Editor 的 Web 站点 http://www.arbortext.com/products/ADEPT_Series/adept-series.html。

26.2.4 Stilo WebWriter

WebWriter 是一个 Stilo 出品的基于 Windows 的 XML 制作工具。它设计的目的是能够方便的用来创建 XML 文件。与其他大多数的 XML 制作工具类似,WebWriter 使用图形化的方法来构造 XML 文件。它支持创建有效 XML 文件和结构良好的 XML 文件,这只取决于你是否提供 DTD。在没有提供 DTD 的情况下,WebWriter 可以基于机构良好文件的结构自动产生一个 DTD。这是一个有趣的特性,刚刚入门的 XML 开发人员会发现这个功能很有用,它在学习如何创建 DTDs 方面给予了开发人员一定的帮助。

WebWriter 的 GUI 是内容敏感的,也就是说菜单和对话框只显示对当前内容可用的命令和选项。这里的内容可以是 XML 文件中一个被选上的元素或是属性。WebWriter 的 GUI 通过利用附随于 DTD 的元素和属性,强制开发有效的 XML 文件。这些元素的内容模型同样被支持,这是维持文件有效性的一个重要的部分。当创建一个基于 DTD 的新文件时,WebWriter 将自动插入文件所需要的最小元素集合以与 DTD 结合。WebWriter 当前可用于 Windows 98/NT/2K。要获得有关 WebWriter 的更多信息,请访问 WebWriter 的 Web 站点 <http://www.stilo.com/webwriter.htm>。

26.2.5 IBM Xena

IBM 很早就重视 XML 技术,这导致大批的 XML 工具和应用的产生。第 30 章“了解 IBM 的 XML 工具包”中讲述了一些 IBMalphaWorks XML 开发小组开发的非常有趣的工具。在制作工具方面,IBM 有一个称为 Xena 的工具,它按照一个有效的 XML 文件编辑器设计。Xena 完全是用 Java 开发的,也就是说它需要运行一个 Java 解释程序。这也同样意味着它被任何支持 Java 的平台所支持。

Xena 提供了一个可视化编辑有效 XML 文件的 GUI。它读取一个 DTD 并创建一个元素选项板用来创建文件。Xena 中的初始编辑方法是一个显示着元素间的层次关系的 XML 树。当你在这个树上添加元素的时候,只有那些对于在树上的被选节点有效的元素会在选项板上被高亮显示。它支持元素的内容模型,这在 DTD 中被设置。Xena 还包含一个 XML 资源浏览器以测试原始 XML 代码。

Xena 当前可用于 Windows 98/NT/2K、Macintosh 和 UNIX。要获得有关 Xena 的更多信息或要下载它,请访问 Xena 的 Web 站点 <http://www.alphaworks.ibm.com/tech/xena>。

26.2.6 Vervet Logic XML Pro

XML Pro 是一个 Vervet Logic 开发的 XML 制作工具。它也提供层次化 XML 树格式的 XML 内容。然而,XML Pro 也可以显示与树上所选节点相联系的数据,诸如属性及它们各自的值。XML Pro 的 GUI 直截了当、易于学习并包含着诸如拖曳和元素及属性向导这样的有趣特性。你还可以在 XML 文件中寻找指定数据,以及通过确定 DTD 创建有效 XML 文件。

XML Pro 的设计使得它在数据实体方面的创建比创建大尺寸文件要更好一些。或许你会发现 XML Pro 在开发基于事务的 XML 文件方面非常的有用,而对于诸如技术规范和文

件这样的大尺寸 XML 文件则正好相反。

XML Pro 当前可用于 Windows 98/NT/2K、Linux 和 Solaris。要找到有关 XML Pro 的更多信息,请访问 Vervet Logic 的 Web 站点 <http://www.vervet.com/product-index.html>。

26.2.7 Bluestone Visual-XML

Visual-XML 是 Bluestone 开发的 XML 制作工具,它适用于包含 XML 的企业级数据库解决方案的创建。它是 Bluestone 的 XML Suite 的一部分,这个套件支持 Visual-XML 和 XML 服务器。XML 服务器通过与后台的数据库接触产生 XML 代码,而 Visual-XML 提供一个适用于整个企业级应用的创建 XML 文件的方法。

这并不令人惊讶,Visual-XML 是一个可视化 XML 工具,它提供一个用于编辑 XML 内容的 GUI。它允许你访问和显示数据库和 XML 文件,它还可以执行 SQL 命令和已保存程序。Visual-XML 不只是一个 XML 编辑器,实际上它还是一个企业级的开发工具,而它对 XML 编辑的支持更为重要。一个独特的有趣特性就是它的数据发布向导(Data Publish Wizard),它直接从数据库的表中生成 DTDs 和 XML 文件。在 Visual-XML 编辑器中,你还可以使用拖拽功能将 XML 元素和数据库对象进行绑定。

Visual-XML 完全使用 Java 编写,因此它可以在任何完全支持 Java 的操作平台上运行。要获取更多的有关 Visual-XML 的信息并免费下载,请访问 Visual-XML 的 Web 站点 <http://www.bluestone.com/xml/visual-XML>。

26.2.8 Microsoft XML Notepad

Microsoft XML Notepad 被公认是一个低级的 XML 制作工具,但它对于简单的制作任务还是非常有用的。作为 Windows 附带的简单 Notepad 文本编辑器的 XML 等价物,XML Notepad 依赖层次树提供对 XML 文件的显示。如果你安装了 Internet Explorer 5.0,它可以验证文件。XML Notepad 的 GUI 非常的简单,工具也非常容易学习。这就使它在创建和编辑小的 XML 数据集方面非常有用。

XML Notepad 可用于 Windows 98/NT/2K 并且需要安装 Internet Explorer 4.01 或更高版本。要找到更多有关 XML Notepad 的信息或下载它的免费版本,请访问 XML Notepad 的 Web 站点 <http://msdn.microsoft.com/xml/notepad>。

26.2.9 Emile

由 Media Design in-Progress 开发的 Emile 是第一个专为 Macintosh 平台设计的 XML 制作工具。它使用 GUI 创建和编辑 XML 文件。Emile 重视在 DTD 中定义的 XML 词表并依次改造用户接口,这样用户可以停留在 DTD 的结构化范围内。名为 Emile Lite 的 Emile 的特殊版本支持对可扩展 HTML(XHTML—Extensible HTML)的创建。XHTML 在第 18 章“XHTML:XML 与 HTML 结合的产物”中进行了介绍。

Emile 当前可用于 Macintosh 平台。要获得有关 Emile 的更多信息和下载这个软件,请访问 Emile 的 Web 站点 <http://www.in-progress.com/emile>。

26.3 XML 内容管理工具

与 XML 制作工具不同,内容管理工具提供了管理 XML 内容的方法。而制作工具只提供编辑 XML 文件的方法。当谈到管理的时候,我实际指的是保管 XML 内容。举例而言,这相当于一个用来存储 XML 文件的数据库或文件系统。不管你使用哪种 XML 制作工具,如果你在开发一个大的 XML 信息库,你都希望使用某种类型的内容管理工具来进行浏览。另一方面,许多只具有小数量 XML 数据的 Web 站点和应用程序也就能够脱离在 Web 服务器上简单存储 XML 文件的无聊工作。

下面是本书编写期间一些最常用的 XML 内容管理工具:

- Poet Content Management Suite
- Arbortext Epic
- Chrystal Astoria
- Oracle 8i

下面的几节将带你了解上面的这些内容管理工具并在 XML 内容管理工具的使用方面给你一些建议。

26.3.1 Poet 内容管理套件

Poet 内容管理套件是一个使用面向对象数据库来存储 XML 内容的内容管理工具。套件的各个部分提供了对管理、复用和在协作环境下发布 XML 内容的支持。除了支持 XML 文件的存储,Poet 内容管理套件中的面向对象数据库可以存储任何类型的继承数据文件,包括多媒体内容。

Poet 支持 XML 和 SGML,通过其强大的部件将协作发布的过程流水线化。对于基于 Web 的 XML 解决方案,Poet 包含一个 Web Factory(Web 工厂)部件来自动的从 XML 文件中产生可浏览的 HTML。通过对复用和版本管理的支持,Poet 提供了一个完整的 XML 发布系统。它支持内容的复用以及内容开发人员间的有效协作。

注意: Poet 起先是用用于医药组织的 XML 内容管理工具,那时它用来作为医药发布的内容管理解决方案。

Poet 内容管理套件当前可用于 Window 98/NT/2K、Solaris(只用于服务器)和 HP-UX(只用于服务器)。有关 Poet Content Management Suite 的更多信息请访问 Poet 内容管理套件的 Web 站点 <http://www.poet.com/products/cms/cms.html>。

26.3.2 Arbortext Epic

由 Arbortext 开发的 Epic 可用作 Web、CD-ROM 或出版方面发布 XML 文件的完全的 XML 内容管理系统。Epic 通过版本管理系统支持内容开发人员间的协作,这个版本管理系统要求开发人员检查进出的文件。它还包括一个基于 Web 的反馈工具。通过它,用户可以将意见通过一个用于识别的自增号码自动的被记录。

与其他的内容管理系统不同,Epic 包含一个模仿传统文字处理软件的 XML 编辑器。这

就使得 Epic 在创建和配置 XML 内容工作可以贯穿始终。另一方面,这对于那些需要专门的内容管理工具和专门的制作工具的公司来说,就没有多大的吸引力了。

要获得有关 Epic 的更多信息,请访问 Epic 的 Web 站点。<http://www.arbortext.com/Products/Epic/epic.html>。

26.3.3 Chrystal Astoria

Chrystal Software 出品的 Astoria 是一个在支持 SGML/XML 的领域上开发的 XML 内容管理工具。它主要是针对那些必须创建和管理含有结构化文件的 XML 信息库——诸如技术和参考文件——的机构而设计的。Astoria 与 XML 制作工具相结合,并允许 XML 内容开发人员协作和复用内容。为了便于这样的协作,Astoria 在内容版本控制方面提供了广泛的支持。

Astoria 所使用的数据库是 Object Design 出品的 ObjectStore。这是一个面向对象的数据库,它支持包括多媒体内容在内的大量数据类型。ObjectStore 使得 Astoria 对从结构化的 XML 内容到物理数据库结构的映射的顺利控制成为了可能。

Astoria 当前可用于 Windows98/NT/2K 和 Solaris(只用于服务器)。要获得有关 Astoria 的更多信息,请访问 Chrystal Software 的 Web 站点 <http://www.chrystal.com>。

26.3.4 Oracle 8i

Oracle 数年来一直是 DBMS 的业界领头羊,它的最新版本 Oracle 8i 也包含了 XML。Oracle 8i 对 XML 的支持始于数据库文件系统。在那里建立了一个在 XML 文件和物理数据库间移动数据的机制。这需要一个 XML 解析器,当然完全是用 Java 开发的。Oracle 8i 还包含一个功能强大的 XML 搜索工具以执行 XML 文本搜索。

很值得指出的是 Oracle 8i 根本上就是一个 DBMS,而不是一个完全的 XML 内容解决方案。换句话说,一个完全的 XML 内容管理系统可以在 Oracle 8i 基础上实现,但它仍然需要现实的实现。我提及它需要另外的完全的内容管理工具是因为 Oracle 在数据库市场上是一个如此重要的角色,以至于它对 XML 的支持非常的重要。Oracle 8i 主要满足那些希望设计和开发自己的 XML 内容管理系统的公司的需求。

Oracle 8i 当前可用于 Windows NT、OS/2 和大量的 UNIX 平台。要学习有关 Oracle 8i 的更多知识,请访问 Oracle 8i 的 Web 站点 <http://www.oracle.com/database/oracle8i>。

26.4 其他 XML 工具

虽然制作工具和内容管理工具构成了 XML 工具的两大主要类型,但还有大量的并不容易以此区分的工具。你将在第 6 部分的其余部分学到一些这类工具。而在本章中介绍一些工具还是值得的。下面是三个 XML 工具,它们在开发 Schema、模型化 DTDs 和创建样式表方面提供了有趣的功能:

- XML Authority
- Near and Far Designer
- XML Stylus

下面的几节将探讨这些工具以及它们是如何满足 XML 开发人员的热望的。

26.4.1 XML Authority

Extensibility 出品的 XML Authority 是一个 Schema 开发工具,它在常规的 Schema 开发和特殊 XMLSchema 开发方面提供了一个有趣的方法。通过建立一个在非常普通意义上创建 Schema 的用户接口,它抽象出 Schema 的概念。与其他工具相比,XML Authority 使用的自身 Schema 格式是 DTD,而它还有用于以 XML Schema 文件方式输出 Schema 的输出工具。它在不同的 Schema 格式间无缝传送的能力非常强大,这在 Schema 标准仍处于变化阶段的情况下非常的有用。

注意:Schema 主要在第 2 章“数据模型:文件类型定义(DTD)和 Schema”中讲解,并在第 5 章“XMLSchema 基础”和第 6 章“XMLSchema 构造技术”中进行更深入阐述。第 6 章实际的指导你如何使用 XML Authority 创建 XMLSchema。

XML Authority 提供了一个可视化创建 Schema 的 GUI。一旦 Schema 被构造,你就决定了你希望存储的格式。例如,你可以创建一个独立的 Schema 并把它存成一个 DTD 或是 XMLSchema 文件。在 XML 工具中,将 Schema 从一种格式转换到另一种格式的功能是一种非常强大并且十分独特的特性。

XML Authority 当前可用于 Windows98/NT/2K 和 UNIX。要获得有关 XML Authority 的更多知识,或要下载它的免费版本,请访问 Extensibility 的 Web 站点 <http://www.extensibility.com>。

26.4.2 Near and Far Designer

另一个有趣的 Schema 工具就是 Microstar 出品的 Near and Far Designer。它专为构造 XML DTD 而设计。Near and Far Designer 包含一个用于构造 DTD 细节的 GUI。对于那些还不熟悉 DTD 神秘语法的初级 XML 开发人员来说,这无疑是十分有利的。甚至有经验的开发人员也可以从 DTD 可视化创建方法中获益,因为它支持诸如拖拽和剪贴这样的时髦 GUI 便捷工具。Near and Far Designer 依赖于 DTD 结构的层次树显示,这也是 XML 文件在可视化工具中的主要显示模式。

Near and Far Designer 具有分析和验证 DTD 错误的能力,它可以实现 XML 和 SGML 之间的转换。它还有一个根据 DTD 的结构组件产生详细报告的报告工具。

Near and Far Designer 当前可用于 Windows 98/NT/2K。要获得有关 Near and Far Designer 的更多信息并下载它的演示版,请访问 Near and Far Designer 的 Web 站点 <http://www.microstar.com/near.html>。

26.4.3 Stylus

与 XML Authority 和 Near and Far Designer 不同,Transformis 出品的 Stylus 是一个为创建 XSL 样式表而设计的 XML 工具。实际上,Stylus 是第一个开发 XSL 样式表的组合开发工具。XML 开发人员对 XSL 样式表的最大的抱怨之一就是它太复杂了。Stylus 提供了一个用于开发 XSL 样式表的简单易学的 GUI。Stylus 还包含一个 XML 解析器和一个 XSL 处

理函数,这两个都被结合在 Stylus 环境中,用来提供解析 XML 代码和 XSL 样式表并报告错误的方法。

Stylus 的一个有趣的特性就是样式表后台映射(backmapping),它在样式表和它们的输出之间建立了一个可视化的连接。这就使所给 XSL 代码的片断如何影响 XML 文件的翻译和并发显示的问题变得非常明显。样式表后台映射的工作方式是,通过高亮显示相关的 XSL 代码以响应所选的输出文件部分。另一个便捷的特性就是 Stylus 的 Sense:X 技术,它可以智能的对曾经使用的元素和属性提供建议。

注意:在本书编写期间,Microsoft Internet Explorer 是惟一支持 XSL 的 Web 浏览器。然而,Microsoft 的 XSL 版本与 W3C XSL 转换(XSLT)工作草案有很多的差异。Stylus 支持 XSLT 工作草案以及 Microsoft 的私有 XSL 版本。

Stylus 当前可用于 Windows 98/NT/2K,并需要安装 Internet Explorer 4.0 或以上的版本。要获得有关 Stylus 的更多信息,请访问 Transformis 的 Web 站点 <http://www.transformis.com/products.shtml>。

26.5 总 结

XML 工具是一种在多方面简化创建和操控 XML 代码任务的软件。虽然从技术上讲,一个文本编辑器可以满足你在创建 XML 文件、DTD 和样式表方面的需求,但更为先进的工具可以使事情变得更简单。因此,XML 工具将影响着 XML 被接受并在以后几年中被使用的速度。想一想 XML 是如此的年轻,但它却有着如此多数量的 XML 开发工具。有如此多的 XML 工具的原因之一就是 XML 与 SGML 的关系,而 SGML 是一个存在了很长时间的技術。

本章向你介绍了一些当前最为常用的 XML 工具。这并不意味着我们要对每种可以想到的 XML 工具进行毫无遗漏的描述。我只是希望在如何选择制作和管理 XML 内容的工具方面提供一些建议,这是更为重要的。要访问本章所提及的工具发行商的 Web 站点,请查阅附录 B“XML 资源”以获得更多的寻找 XML 工具的指导。现在有很多 XML 工具,而更多的还处于开发之中,因此要花时间寻找一些适合你个人需求的工具。

第 27 章 使用 XML 生成器移植数据

虽然,XML 作为传输和在大多数情况下存储数据的格式方面非常的超前,但你在利用 XML 的各种功能之前必须将数据转换为 XML 格式。它的主要理由是,将数据转换为适当的 XML 格式是将一个数据驱动应用移植到 XML 上的重要步骤。如你所知,数据可以通过多种途径储存,包括文本文件、数据库和 Web 页面。同时,许多的应用依赖于海量的数据,而这些数据很难手工的转换为 XML。这一点就产生了对自动化移植数据到 XML 的方法的巨大需求。

本章将向你介绍一个名为 XML 生成器的工具,它是一个自动化移植数据到 XML 的方法。XML 生成器可以将一个有分隔数据字段的文本文件转换为一个高度结构化的 XML 文件。由于从数据库或其他数据源中生成文本文件相对比较简单,因此这个方法可以用来将各种类型的数据转换为 XML 格式。

27.1 XML 生成器基础

由于 XML 不断把自己确立为共享和存储结构化信息的强大方法,越来越多的开发人员被要求更改他们的基于 Web 的应用程序以支持 XML。修改的部分将包括将大量的数据以结构化 XML 文件存储。当你考虑到一些数据库的大小,你就会了解手工转换数据不是可选的方案。因此,就需要一个自动完成 XML 数据转换的方法。

有分隔符文本文件是数据表述的一种常用格式,并被用来在不同的数据库间移动信息。基本上,它由文本行组成,这些行与数据库中的记录相对应。在每一行文本中,记录的每一个字段通过一个特殊字符——如逗号(,)或制表符(tab)——来区分。大多数的数据库应用程序支持有分隔符文本文件的导入和导出。这就使这种文本易于获得。下面是一个有分隔符文本文件的简单例子:

```
Name,Address,City,State,ZIP,Phone
```

```
Frank Rizzo,1212 W 304th Street,New York, New York,10011.212-555-1212
```

```
Sol Rosenberg,1162 E 412th Street,New York,New York,10011.212-555-1818
```

这个有分隔符的文本文件通过一个联系管理软件导出,它由两个记录组成。第一行是标题,定义了每一个记录的字段,注意,标题本身不是记录。第二三行是从联系数据库中导出的两个人的记录。由逗号区分字段,既然这样,这个文件就是一个逗号分隔的文本文件。要记住,每个记录的字段与标题行中的相应字段名对应。这个字段的对应非常的重要,因为它确定文件中数据的结构。

XML 生成器是一个由 DataChannel 出品的 Java 应用程序,这个公司同时也是 XML 解析器 XJParser 的制作者。该生成器允许你将有分隔符的文本文件转换为一个结构化 XML 文件。XML 文件的结构由你创建的模板决定,这个模板描述了文本文件中的字段与 XML

元素及属性的影射关系。你可以自由的实质描述任何你所希望的 XML 文件结构,条件是:只需文本文件的字段可以清楚的映射成元素和属性。

当你运行 XML 生成器的时候,你必须指定源有标记文本文件,以及生成 XML 文件的 XML 模板。还有就是你必须指定在文本文件中用来分隔字段的分隔符。你提供给 XML 生成器的 XML 模板必须本身是一个结构良好的 XML 文件。你还要使用模板中特殊的处理命令告知 XML 生成器文本字段与 XML 结构之间的映射。

27.2 使用 XML 生成器应用程序

由于 XML 生成器是作为 Java 应用程序运行的,所以你必须使用 Java 解释器来运行它,比如一个包含 Java 开发工具包(Java Development Kit-JDK)的系统。JDK 可以从 JavaSoft 的 Web 站点 <http://java.sun.com> 上免费下载。下面是 XML 生成器应用程序的语法:

```
java XMLGenerator XMLTemplate TextFile Delimiter XMLFile
```

注意:XML 生成器应用程序包含在 DataChannel 的 XML 解析器 XJParser 中,这个软件可以从 DataChannel 的 Web 站点 <http://www.datachannel.com> 中免费下载。

如你所见,XML 生成器应用程序称为 XMLGenerator,它接受下面四个命令行参数:

- XMLTemplate——描述生成的 XML 文件结构的 XML 模板文件。
- TextFile ——分隔的文本文件。
- Delimiter ——在文本文件中的每一行用来分隔字段的分隔用字符。
- XMLFile——生成的 XML 文件的输出文件名。

下面是一个调用 XML 生成器的例子,这里文本文件名为 Contacts.txt 而 XML 模板名为 ContactTemplate.xml:

```
java XMLGenerator ContactTemplate.xml Contacts.txt , Contacts.xml
```

在这个例子中,分隔符是逗号(,),而输出的 XML 文件名为 Contacts.xml。

重温一下 XML 生成器应用程序的参数,两个最重要的参数是 XMLTemplate 和 TextFile,因为它们基本上确定了要转换的数据以及数据转换结果的格式。因此,我们很值得进一步看一下这些文件。

27.2.1 分隔的文本数据文件

分隔的文本数据文件传给 XML 生成器,它包含着希望你希望转换到 XML 格式的数据。数据由含有分隔的字段的行组成。文件的每一排被认为是一行,这个行中每一个分隔的字符串被认为是一个字段。当你使用 XML 生成器转换文件时,你要指定一个分隔符。逗号(,)是最为常用的分隔符,而分号(;)不允许作为分隔符。除此之外,你可以使用任何一个不在文本数据内容中出现的单一字符作为分隔符。

警告:分隔用的字符不能出现在文本数据的内容中,这一点非常的重要。如果出现这种情况,XML 生成器将不能把它们区分出来,并将假定这个字符是分隔符,造成

数据错误。

分隔的文本数据文件的第一行是一个特殊行,它指定每一个记录(行)中的字段名称。这些字段名称就是你在 XML 模板中所用到的用来实现文本数据和 XML 元素及属性间的映射的工具。

清单 27.1 包含了一个分隔的文本文件(Vehicles.txt)的例子,这个文件包含了可以用来更新汽车记录的数据。

清单 27.1 分隔的文本文件 Vehicles.txt

```
year,make,model,mileage,color,price
1996,Land Rover,Discovery,36500,black,$ 22100
1998,Land Rover,Discovery,15900,teal,$ 32000
1997,Land Rover,Discovery,46000,silver,$ 27900
1997,Land Rover,Defender 90,21050,white,$ 41900
1994,Land Rover,Defender 90,42450,green,$ 31250
1996,Toyota,Land Cruiser,34800,black,$ 35995
1997,Toyota,Land Cruiser,47150,green,$ 37000
1999,Chevrolet,Suburban 2500,3550,pewter,$ 31995
1996,Chevrolet,Suburban 2500,49300,green,$ 25995
```

这个分隔的文本文件包含着旧车的数据,它可以作为电子清单的一部分发给旧车销售商。这是一个很好的例子,这些数据以一种不好管理但能够从 XML 结构中受益的格式存储。此外,汽车销售商可能希望使用这个数据作为列出最近清单的网页的基础。将数据转换为 XML 同时满足这两项需要。然而,不首先创建一个 XML 模板,你就无法使用 XML 生成器将数据转换为 XML。

27.2.2 XML 模板

XML 模板是一个结构良好的 XML 文件,它描述从分隔的文本文件中生成的 XML 文件的结构。由于分隔的文本文件由包含分离字段的数据记录组成,XML 模板主要描述文本数据字段和 XML 元素及属性之间的映射。XML 生成器反复在每个文本文件中的记录上运行,将记录的数据转换为 XML 结构。

注意:任何在 XML 模板中出现的注释都不会写入到 XML 输出文件中。

文本数据到 XML 结构的映射通过在 XML 模板中使用专门处理命令描述。下面是在 XML 模板文件中使用的处理命令:

- XG-Iteration-XG "Start"——数据重复循环的开始。
- XG-Iteration-XG "End"——数据重复循环的结束。
- XG-InsertField-XG "FieldName"——将数据字段作为元素内容插入。
- XG-InsertField-XG FieldName——将数据字段作为属性值插入。

当 XML 生成器将分隔的文本文件转换为 XML 文件时,它在数据的每一行重复运行,并将适当的数据字段插入到模板中描述的 XML 文件的结构中。XG-Iteration-XG "Start"和

XG-Iteration-XG "End" 处理命令用来确定 XML 模板中的重复循环。任何在这些处理命令外出现的 XML 代码只在结果文件中出现一次。任何在这些处理命令那里的代码将应用到数据的每一行中。

警告: 在一个 XML 模板中只能有一个重复循环,这也就意味着你不能嵌套重复循环。要注意一个重复循环由一对封装其他 XML 模板内容的 `<? XG-Iteration-XG "Start"? >` 和 `<? XG-Iteration-XG "End"? >` 处理命令组成。

文本数据字段通过使用 XG-InsertField-XG "FieldName" 处理命令映射为 XML 元素内容。这样 FieldName 指出要插入到结果 XML 文件中的字段的名称。下面是一个这个处理命令如何使用的例子:

```
<book><? XG-InsertField-XG "Book"? ></book>
```

在这个例子中,分隔的文本文件中的 Book 字段被插入到 XML 标记 `<book>` 和 `</book>` 中,构成了一个完整的 book 元素。

要将文本字段映射为属性,你要使用 XG-InsertField-XG FieldName 处理命令。同样,FieldName 确定作为属性值插入的字段的名称。下面是一个如何使用 XG-InsertField-XG FieldName 处理命令的例子:

```
<book isbn=" XG-InsertField-XG ISBN"><? XG-InsertField-XG "Book"? ></book>
```

在这个例子中,ISBN 文本字段作为 book 元素中 isbn 属性的值被插入。下面是一个该 XML 模板如何将文本数据转换为 XML 文件代码的例子:

```
<book isbn="0-7897-2131-7">The Complete Idiot's Guide to Java</book>
```

现在你已经看到了一些示范 XML 模板处理指令的代码片断,现在让我们看一个实际的例子。清单 27.2 包含了一个 XML 模板 (VehicleTemplate.xml) 的代码,它被设计用来将你早先看到的有关旧车的分隔的文本文件转换为 XML。

清单 27.2 XML 模板 VehicleTemplate.xml

```
<vehicles>
  <? XG-Iteration-XG "Start"? >
    <vehicle year="XG-InsertField-XG year" make="XG-InsertField-XG make" model="
XG-InsertField-XG model">
      <mileage><? XG-InsertField-XG "mileage"? ></mileage>
      <color><? XG-InsertField-XG "color"? ></color>
      <price><? XG-InsertField-XG "price"? ></price>
    </vehicle>
  <? XG-Iteration-XG "End"? >
</vehicles>
```

27.3 生成 XML 文件

到这里,你已经看到了 XML 生成器的两个主要困惑:分隔的文本文件和 XML 模板。使

用这两个文件生成 XML 文件同运行 XML 生成器一样简单。下面是使用有关旧车的例子文件运行 XML 生成器的命令。

```
Java XMLGenerator vehicleTemplate.xml Vehicles.txt , Vehicles.xml
```

运行这个命令就创建了 XML 文件 Vehicles.xml,它以结构化的 XML 格式包含了旧车的文本数据。清单 27.3 显示了 Vehicles.xml 文件的代码。

注意:使用 Java 解释程序运行 XML 转换生成器应用程序的例子以附带在 Java 开发工具包(JDK)中的标准 Java 解释器为基准。

清单 27.3 由 XML 生成器生成的 XML 文件 Vehicles.xml

```
<vehicles>
  <vehicle year="1996" make="Land Rover" model="Discovery">
    <mileage>36500</mileage>
    <color>black</color>
    <price>$ 22100</price>
  </vehicle>
  <vehicle year="1998" make="Land Rover" model="Discovery">
    <mileage>15900</mileage>
    <color>teal</color>
    <price>$ 32000</price>
  </vehicle>
  <vehicle year="1997" make="Land Rover" model="Discovery">
    <mileage>46000</mileage>
    <color>silver</color>
    <price>$ 27900</price>
  </vehicle>
  <vehicle year="1997" make="Land Rover" model="Defender 90">
    <mileage>21050</mileage>
    <color>white</color>
    <price>$ 41900</price>
  </vehicle>
  <vehicle year="1994" make="Land Rover" model="Defender 90">
    <mileage>42450</mileage>
    <color>green</color>
    <price>$ 31250</price>
  </vehicle>
  <vehicle year="1996" make="Toyota" model="Land Cruiser">
    <mileage>34800</mileage>
    <color>black</color>
    <price>$ 35995</price>
  </vehicle>
  <vehicle year="1997" make="Toyota" model="Land Cruiser">
    <mileage>47150</mileage>
    <color>green</color>
    <price>$ 37000</price>
  </vehicle>
  <vehicle year="1999" make="Chevrolet" model="Suburban 2500">
```



```

    <mileage>3550</mileage>
    <color>pewter</color>
    <price>$ 31995</price>
  </vehicle>
  <vehicle year="1996" make="Chevrolet" model="Suburban 2500">
    <mileage>49300</mileage>
    <color>green</color>
    <price>$ 25995</price>
  </vehicle>
</vehicles>

```

这个文件明显比初始的分隔文本文件包含更多的结构和内容。更重要的是,它现在可以被任何 XML 工具处理以进行显示或管理。

我在本章中曾提到,一个汽车销售商可能希望在网页上显示旧车数据来提醒客户新的清单的到来。由于这个数据是作为 XML 文件存储的,这就使使用 XSL(扩展样式语言)显示它成为了可能。清单 27.4 包含了一个 XSL 样式表(Vehicles.xsl)的代码,它用来将旧车数据以表格形式显示在网页上。

清单 27.4 用来显示旧车 XML 文件的 XSL 样式表 Vehicles.xsl

```

<? xml version='1.0'? >
<xsl:stylesheet xmlns:xsl=http://www.w3.org/TR/WD-xsl>
  <xsl:template match="/" >
    <html>
      <head>
        <title>Used Vehicles</title>
      </head>
      <body background="Money.jpg">
        <h1 style="background-color: #446600;
          color: #FFFFFF; font-size: 20pt; text-align: center;
          letter-spacing: 1.0em">Used Vehicles</h1>
        <table align="center" border="2">
          <tr>
            <th>Year</th>
            <th>Make</th>
            <th>Model</th>
            <th>Mileage</th>
            <th>Color</th>
            <th>Price</th>
          </tr>
          <xsl:for-each select="vehicles/vehicle">
            <tr>
              <td><xsl:value-of select="@year"/></td>
              <td><xsl:value-of select="@make"/></td>
              <td><xsl:value-of select="@model"/></td>
              <td><xsl:value-of select="mileage"/></td>
              <td><xsl:value-of select="color"/></td>
              <td><xsl:value-of select="price"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </template>

```

```

</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

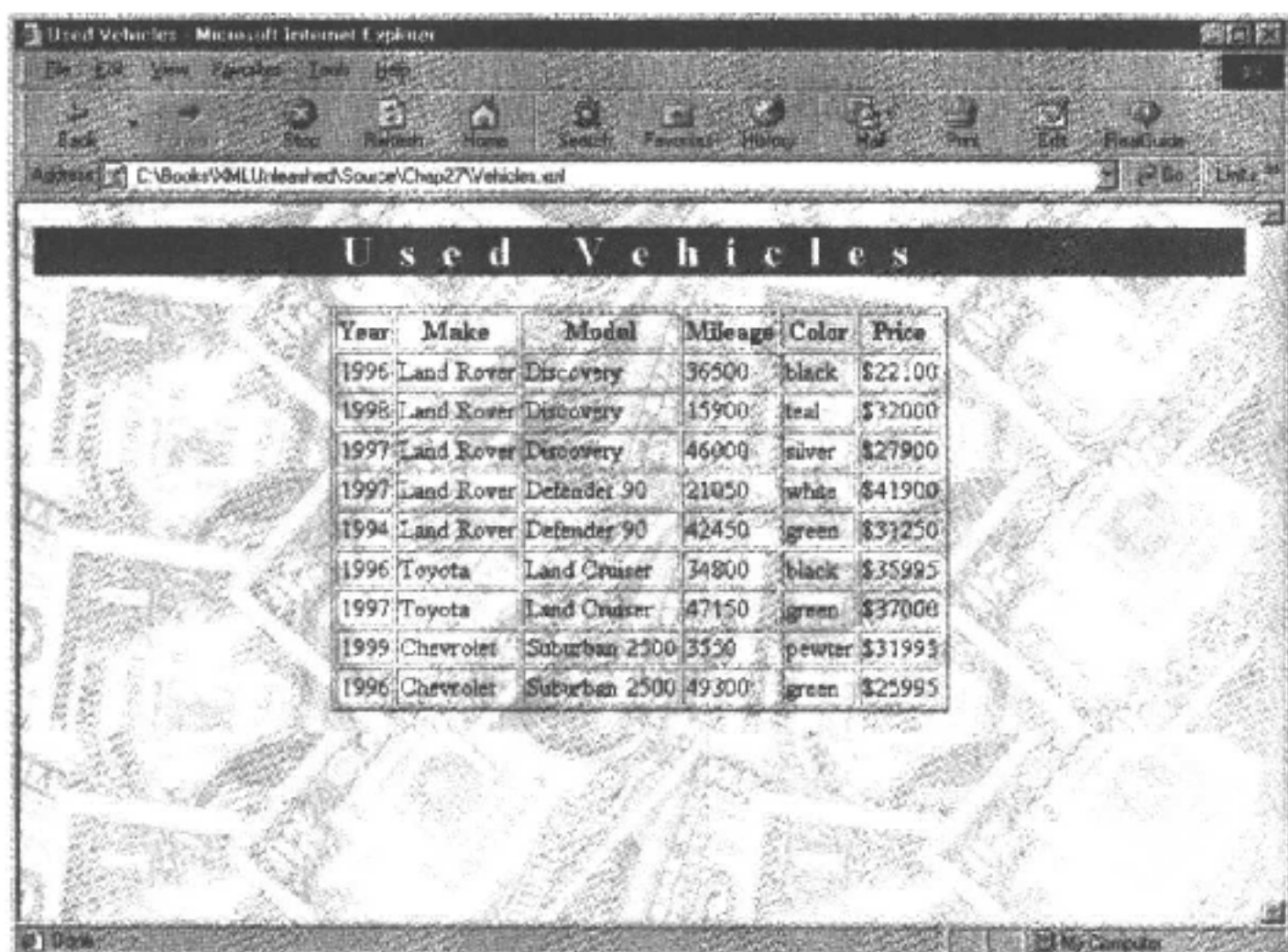
这里不对这段 XSL 代码的细节进行讲解,这是因为 XSL 将在第 10 章“理解扩展样式语言(XSL)”和第 11 章“创建 XSL 样式表”中进行详细讲解。然而,我还是要指出,为了使用样式表,我们有必要在 Vehicles.xml 文件中作一些细微的增补。下面的两行代码必须放置在文件的开始部分以将样式表与文件结合:

```

<? xml version="1.0" ? >
<? xml-stylesheet href="Vehicles.xsl" type="text/xsl" ? >

```

包含着旧车数据的网页(如图 27.1 所示)是使用机动车 XML 文件和 XSL 样式表的结果。



| Year | Make | Model | Mileage | Color | Price |
|------|------------|---------------|---------|--------|---------|
| 1996 | Land Rover | Discovery | 36500 | black | \$22100 |
| 1998 | Land Rover | Discovery | 15900 | teal | \$32000 |
| 1997 | Land Rover | Discovery | 46000 | silver | \$27900 |
| 1997 | Land Rover | Defender 90 | 21050 | white | \$41900 |
| 1994 | Land Rover | Defender 90 | 42450 | green | \$31250 |
| 1996 | Toyota | Land Cruiser | 34800 | black | \$35995 |
| 1997 | Toyota | Land Cruiser | 47150 | green | \$37000 |
| 1999 | Chevrolet | Suburban 2500 | 3550 | pewter | \$31999 |
| 1996 | Chevrolet | Suburban 2500 | 49300 | green | \$25995 |

图 27.1 在网页中显示的旧车数据

虽然这是文本数据如何被处理成 XML 并在网页上显示的好例子,但这并不是必要的后台共享信息反馈的典型情况,在这种情况下,XML 普遍的被结合使用。因此,我希望讲解另一个例子来示范 XML 生成器可达到的自动化水平。看一看 XML 生成器是如何被用来获取直接从数据库中析取的数据并将它转换为结构化 XML 的。我想这一定是一件非常有趣的事情。

与大多数的数据库应用程序类似,Microsoft 的 Access 允许你将数据库的表以分隔的文本文件方式导出。Access 实际上有一个非常聪明的导出向导(Export Wizard),你可以用它来指定分隔符和其他的导出选择。清单 27.5 给出了一个直接从一个 Microsoft Access 数据

库中导出的产品表的分隔的文本文件。

清单 27.5 直接从一个 Microsoft Access 数据库中导出的分隔的文本文件 Products.txt

```
ProductID,ProductName,SupplierID,CategoryID,QuantityPerUnit,UnitPrice,
③UnitsInStock,UnitsOnOrder,ReorderLevel,Discontinued
1,Chai,1,1,10 boxex x 20 bags, $ 18. 00,39,0,10,0
2,Chang,1,1,24 - 12 oz bottles, $ 19. 00,17. 40,25,0
3,Aniseed Syrup,1,2,12 -550 ml bottles, $ 10. 00,13.70,25,0
4,Chef Anotn' s Cajun Seasoning,2,2,28 - 6 oz jars, $ 22. 00,53,0,0,0
5,Chef Anton' s Gumbo Mix,2,2,36 boxex, $ 21. 35,0,0,0,1
6,Grandma' s Boysendberry Spread,3,2,12 - 8 oz jars, $ 25. 00,120,0,25,0
7,Uncle Bob' s Organic Dried Pears,3,7,12 -1 lb pkgs. , $ 30. 00,15,0,10,0
8,Northwoods Cranberry Sauce,3,2,12 - 12 oz jars, $ 40. 00,6,0,0,0
9,Mishi Kobe Niku,4,6,18 - 500 g pkgs. , $ 97. 00,29,0,0,1
10,Ikura,4,8,12 - 200 ml jars, $ 31. 00,31,0,0,0
11,Queso Cabrales,5,4,1 kg pkg. , $ 21. 00,22,30,30,0
12,Queso Manchego La pastora,5,4,10 - 500 g pkgs. , $ 38. 00,86,0,0,0
13,Konbu,6,8,2 kg box, $ 6. 00,24,0,5,0
14,Tofu,6,7,40 - 100 g pkgs. , $ 23. 25,35,0,0,0
15,Genen Shouyu,6,2,24 - 250 ml bottles, $ 15. 50,39,0,5,0
```

与本章前面的旧车例子不同(那个例子是我手工编辑的),Microsoft Access 从一个数据库表中生成这个分隔的文本文件。如你所见,文本文件中的数据由一个产品清单组成,可以用来从在线食品商店中购买产品。这个例子中的惟一手工编制的代码是 XML 模板的创建,它只需要创建一次。清单 27.6 给出了产品 XML 模板的代码。

清单 27.6 XML 模板 ProductTemplate.xml

```
<products>
  <? XG-Iteration-XG "Start" ? >
  <product id="XG-InsertField-XG ProductID"
    name="XG-InsertField-XG ProductName"
    discontinued="XG-InsertField-XG Discontinued">
    <supplier id="XG-InsertField-XG SupplierID"/>
    <category id="XG-InsertField-XG CategoryID"/>
    <unit price="XG-InsertField-XG UnitPrice">
      <quantity><? XG-InsertField-XG "QuantityPerUnit"? ></quantity>
      <instock><? XG-InsertField-XG "UnitsInStock"? ></instock>
      <onorder><? XG-InsertField-XG "UnitsOnOrder"? ></onorder>
      <reorder><? XG-InsertField-XG "ReorderLevel"? ></reorder>
    </unit>
  </product>
  <? XG-Iteration-XG "End" ? >
</products>
```

这个 XML 模板的完整检查显示出所有分隔的文本文件都在重复循环中被说明进 XML 元素和属性当中。不止如此,通过在 unit 元素中嵌套 unit 信息,我们在模板中附加了额外的结构。要使用这个模板生成一个 XML 文件,所需要的就是下面的命令:

```
java XMLGenerator ProductTemplate.xml Products.txt , Products.xml
```

这个指令生成了一个名为 Products.xml 的 XML 文件,它以新的 XML 格式包含文本数据。清单 27.7 显示了由 XML 生成器生成的 XML 文件 Products.xml 的代码。

清单 27.7 由 XML 生成器生成的 XML 文件 Products.xml

```
<products>
  <product id="1" name="Chai" discontinued="0">
    <supplier id="1"></supplier>
    <category id="1"></category>
    <unit price=" $ 18.00">
      <quantity>10 boxes x 20 bags</quantity>
      <instock>39</instock>
      <onorder>0</onorder>
      <reorder>10</reorder>
    </unit>
  </product>
  <product id="2" name="Chang" discontinued="0">
    <supplier id="1"></supplier>
    <category id="1"></category>
    <unit price=" $ 19.00">
      <quantity>24 - 12 oz bottles</quantity>
      <instock>17</instock>
      <onorder>40</onorder>
      <reorder>25</reorder>
    </unit>
  </product>
  <product id="3" name="Aniseed Syrup" discontinued="0">
    <supplier id="1"></supplier>
    <category id="2"></category>
    <unit price=" $ 10.00">
      <quantity>12 - 550 ml bottles</quantity>
      <instock>13</instock>
      <onorder>70</onorder>
      <reorder>25</reorder>
    </unit>
  </product>
  <product id="4" name="Chef Anton's Cajun Seasoning" discontinued="0">
    <supplier id="2"></supplier>
    <category id="2"></category>
    <unit price=" $ 22.00">
      <quantity>48 - 6 oz jars</quantity>
      <instock>53</instock>
      <onorder>0</onorder>
      <reorder>0</reorder>
    </unit>
  </product>
  <product id="5" name="Chef Anton's Gumbo Mix" discontinued="1">
    <supplier id="2"></supplier>
    <category id="2"></category>
    <unit price=" $ 21.35">
```



```

    <quantity>36 boxes</quantity>
    <instock>0</instock>
    <onorder>0</onorder>
    <reorder>0</reorder>
  </unit>
</product>

<product id="6" name="Grandma's Boysenberry Spread" discontinued="0">
  <supplier id="3"></supplier>
  <category id="2"></category>
  <unit price="$ 25.00">
    <quantity>12 - 8 oz jars</quantity>
    <instock>120</instock>
    <onorder>0</onorder>
    <reorder>25</reorder>
  </unit>
</product>

<product id="7" name="Uncle Bob's Organic Dried Pears" discontinued="0">
  <supplier id="3"></supplier>
  <category id="7"></category>
  <unit price="$ 30.00">
    <quantity>12 - 1 lb pkgs. </quantity>
    <instock>15</instock>
    <onorder>0</onorder>
    <reorder>10</reorder>
  </unit>
</product>

<product id="8" name="Northwoods Cranberry Sauce" discontinued="0">
  <supplier id="3"></supplier>
  <category id="2"></category>
  <unit price="$ 40.00">
    <quantity>12 - 12 oz jars</quantity>
    <instock>6</instock>
    <onorder>0</onorder>
    <reorder>0</reorder>
  </unit>
</product>

<product id="9" name="Mishi Kobe Niku" discontinued="1">
  <supplier id="4"></supplier>
  <category id="6"></category>
  <unit price="$ 97.00">
    <quantity>18 - 500 g pkgs. </quantity>
    <instock>29</instock>
    <onorder>0</onorder>
    <reorder>0</reorder>
  </unit>
</product>

<product id="10" name="Ikura" discontinued="0">
  <supplier id="4"></supplier>
  <category id="8"></category>
  <unit price="$ 31.00">
    <quantity>12 - 200 ml jars</quantity>
    <instock>31</instock>

```

```
<onorder>0</onorder>
<reorder>0</reorder>
</unit>
</product>
<product id="11" name="Queso Cabrales" discontinued="0">
  <supplier id="5"></supplier>
  <category id="4"></category>
  <unit price="$ 21.00">
    <quantity>1 kg pkg.</quantity>
    <instock>22</instock>
    <onorder>30</onorder>
    <reorder>30</reorder>
  </unit>
</product>
<product id="12" name="Queso Manchego La Pastora" discontinued="0">
  <supplier id="5"></supplier>
  <category id="4"></category>
  <unit price="$ 38.00">
    <quantity>10 - 500 g pkgs.</quantity>
    <instock>86</instock>
    <onorder>0</onorder>
    <reorder>0</reorder>
  </unit>
</product>
<product id="13" name="Konbu" discontinued="0">
  <supplier id="6"></supplier>
  <category id="8"></category>
  <unit price="$ 6.00">
    <quantity>2 kg box</quantity>
    <instock>24</instock>
    <onorder>0</onorder>
    <reorder>5</reorder>
  </unit>
</product>
<product id="14" name="Tofu" discontinued="0">
  <supplier id="6"></supplier>
  <category id="7"></category>
  <unit price="$ 23.25">
    <quantity>40 - 100 g pkgs.</quantity>
    <instock>35</instock>
    <onorder>0</onorder>
    <reorder>0</reorder>
  </unit>
</product>
<product id="15" name="Genen Shouyu" discontinued="0">
  <supplier id="6"></supplier>
  <category id="2"></category>
  <unit price="$ 15.50">
    <quantity>24 - 250 ml bottles</quantity>
    <instock>39</instock>
    <onorder>0</onorder>
    <reorder>5</reorder>
```

```
</unit>  
</product>  
</products>
```

这段代码显示了 XML 代码和分隔的文本在结构方面的戏剧性区别,虽然 XML 代码相当的长,但它结构良好并更容易理解。同时,XML 代码的真实值在这样的一种格式下,这种格式可以使用大量的 XML 工具中的任何一个显示或操作。对于这个特定的例子,这意味着可以通过诸如命令执行系统这样的后端系统传送 XML 产品信息以进行处理。

27.4 总 结

术语 legacy(遗产)用于软件世界,用来指代那些不再时髦的技术。更准确的说,它指代不能跟上当前计算机技术的软件或数据。由于计算机工业每 18 个月更新一次,一项技术从前端技术到称为遗产用不了多长时间。

legacy 数据指代以老格式存储的数据,这种格式的数据不易被现代的软件访问。虽然有桥和软件层可以允许现代软件访问 legacy 数据,但大多数的开发人员还是认为将 legacy 数据移植成一种现代信息格式才是一个好方法。随着 XML 变得更为普及,许多的开发人员将认识到将数据从 legacy 格式转换为 XML 的益处。

本章讲解了一个名为 XML 生成器的工具,它自动的将 legacy 数据转换为 XML。XML 生成器以分隔的文本文件作为基础,将 legacy 数据转换为 XML 文件。贯穿全章,你学习到如何使用 XML 生成器从分隔的文本文件中生成结构良好的 XML 文件,并使用 XSL 以网页方式浏览 XML 文件。你还学到了如何从数据库表中提取数据并使用 XML 生成器将它转换为 XML。

第 28 章 XFA 脚本系统

当你考虑脚本和 XML 的时候,你一般都会想到诸如 JavaScript 或 Vbscript 这样的网络脚本语言。它们用来浏览使用文件对象模型的 XML 文件树。实际上,这就是在第 20 章“脚本化 XML”中提到的 XML 脚本的方法。然而,这不是惟一的在 XML 中加入脚本的惟一方法。实际上,我们可以使用一个本身基于 XML 的脚本语言。

XFA——也就是 XML For All,是一种根据 XML 词表事项的脚本语言。当你认识到 XML 是为描述数据设计的,而不是为编写代码,你就会觉得这很奇怪。然而,创建 XFA 的人们提出了一种创新的使用 XML 语法实现脚本语言的途径。本章将讨论 XFA 技术并向你展示如何使用 XFA 编写大量不同的 XML 脚本。

28.1 XFA 脚本基础

网上的脚本通常是包括两种或多种技术的结合。例如,要制作一个网页,你必须使用 HTML 代码编写页面并使用 JavaScript 和 VBScript 代码来注入脚本功能。在用来描述文件内容的标记语言和用来处理文件的脚本语言间有着明显的区分。XFA 在使用 XML 作为脚本语言基础编写范例上提供了一个有趣的手法。

XFA 脚本语言通过一个由在常见编程结构(如条件句、循环和变量)中出现的元素和属性组成的 XML 词表实现。XFA 描述了一个来自 XML 起源的动人的出发点,它包含静态数据结构。虽然 XML 作为描述信息的高度结构化的方法功能非常强大,但如果没有诸如脚本语言或可执行应用程序这部分软件的支持,它将一事无成。XFA 需要你从一个全新的视点来审视 XML,这是因为它本身允许 XML 作为一项可交互的技术。

XFA 实际上包含对所有与现代基于 Web 的脚本语言相结合的结构的支持。下面是一些它的主要特性:

- 表达式
- 条件句
- 循环
- 数据类型
- 数据类型间的转换
- 有作用域的结构化块
- 函数
- 基于堆栈的递归运行

本章的其余部分将深入的探讨 XFA 脚本并通过例子讲解这些特性。

28.1.1 解释 XFA 脚本

虽然 XFA 模糊了文件数据和脚本代码的区别,它仍然需要一个离散的脚本引擎来处

理和解释 XFA 脚本代码。这个脚本引擎由 XFA 附带的名为 xfa.exe 的可执行应用程序组成。XFA 脚本引擎也指的是 XFA 解释器。你可以通过两种途径使用 XFA 解释器。

- 作为一个 Web 服务器的脚本引擎。
- 本身作为一个独立的应用程序。

第一种途径是使用 XFA 解释器的最常用方法,因为它允许你将 XFA 代码嵌入到网页之中。在这种情况下,XFA 解释器在 Web 服务器下作为 CGI 应用程序运行。不论何时遇到一个 XFA 脚本,XFA 解释器就被用来运行这个脚本,并以 HTML 代码的形式将结果返回给 Web 服务器。当在 Web 服务器下安装后,XFA 理论上与其他的脚本引擎扮演着同样的角色。

第二种使用 XFA 解释器的途径包括以命令行方式将它作为独立的应用程序运行。当作为独立应用程序运行时,XFA 解释器接收一个 XGA 脚本并输出一个文本结果。与 Web 服务器的情形不同,这个文本结构可以是 HTML 代码,也可以不是。无可否认,这是 XFA 的一种非常专业化的使用方式用的广泛,远远比不上它作为 Web 服务器下的脚本引擎的使用方式。

不论你如何使用 XFA 解释器,它都需要运行解释脚本并生成结果的基本函数。在解释脚本时,XFA 解释器要执行下面三个基本步骤:

- 解析脚本以创建一个 XML 树。
- 通过检查 XML 树的正确性分析脚本。
- 通过运行树上的每一个节点运行脚本。

解释 XFA 脚本的第一步就是解析脚本,它由 XFA 解释器实现,这一步与对任何 XML 文件的解析非常的相似。脚本解析的结果就是一个由与元素和属性相对应的节点组成的树。接下来,XFA 解释器根据 XFA 语法检查树中的每个节点的语法正确性。一旦脚本确定是语法正确的,XFA 就通过处理树中的每一个节点执行这个脚本。

注意: XFA 解释器包含在 XFA 软件包中,它可以从 XFA 的 Web 站点 <http://www.xmlforall.com> 上下载。

28.2 XFA 和 XML 的关系

虽然 XFA 脚本可以作为结构良好的 XML 来进行编写,但 XFA 也可以描述一个扩展 XML 的语法。扩展 XML 的原因就是要使得描述表达式更为的简便,而使用传统的 XML 标记来完成这件任务本身是非常困难的。XFA 的扩展版本可以看作是 XML 的扩展集。虽然大多数的 XML 支持者基本不赞成扩展 XML 的建议,但 XFA 已经提供了对将扩展的 XFA 脚本转换为结构良好的 XML 文件的支持。

XFA 扩展 XML 的惟一理由就是使它在标记脚本代码的编写上更加便利。下面是 XFA 扩展的 XML 语法部分:

- XFA 不需要一个单独的根元素。
- XFA 支持扩充的属性语法。

- XFA 字符串可以包含小于号(<)。
- XFA 名可以以句点(.)和冒号(:)起头。

注意:在本书编写期间,XFA 并不支持与 XML 1.0 具有相同的字符范围。非常特殊,XFA 只支持 8 位统一字符编码字符,而不是完全的 16 位统一字符编码集。总之,由于 XML 默认是 8 位字符,所以除非你已经使用了 16 位字符,否则这并不会有什么问题。

与 XML 不同,XFA 并不需要包含一个单独的包含所有其他元素的根元素。换句话说,XFA 允许有多个顶层元素。XFA 还扩展了属性的 XML 语法使得标记表达式更为方便。例如,下面的代码显示了一个使用 XFA 扩展属性语法的 XFA 表达式代码:

```
<xfa:val salutation("Eddie")/>
```

这行代码显示了如何使用 XFA 的扩展属性语法调用函数。你可以争辩说这行代码根本没有包含属性,但实际上函数调用确实是 xfa:val 元素的一个属性。在这个例子中,salutation()函数被调用并且字符串"Eddie"作为惟一的参数被传入。函数本身要这样编写:

```
<xfa:function salutation(name)>
  <p>Hello <i><xfa:val name/></i>,how are you</p>
</xfa:function>
```

这段函数代码揭示了 XFA 脚本是如何包含 XFA 代码和 HTML 代码的混合的。XML 的名字域功能就是允许这种词表的混合。在 salutation()函数代码中,xfa:val 元素用来引用传入到函数中的 name 变量。效果就是 name 参数的内容被插入到了句子之中。

我早先提到 XFA 脚本可以编写来支持 XML 1.0 语法。实际上,XFA 附带的一个工具可以将扩展 XFA 脚本转换为结构良好的 XML 1.0 文件。你将在本章的稍后部分学习到这个软件。给你一个有关扩展的 XFA 脚本是如何类似于结构良好的 XML 文件的认识,下面是 salutation()函数作为结构良好 XML 代码实现的情况:

```
<xfa:function name="salutation(name)">
  <p>Hello <i><xfa:val name/></i>, how are you? </p>
</xfa:function>
```

代码实际改变的地方就是函数名指定的方式。在这个例子中,函数名使用 name 属性表达,函数名和参数表作为属性值被提供。函数的调用在严格的 XML 1.0 语法中也稍有不同:

```
<xfa:val exp='salutation("Eddie")' />
```

在这种情况下,函数调用以属性值方式提供。exp 属性包含函数调用,单引号(')被使用以避免与用在函数参数中的双引号(")相冲突。

28.3 XFA 数据类型

与大多数的脚本语言相同,XFA 具有大量的用于构造不同数据的不同的数据类型。XFA 数据分成三个主要的类别:

- 字符串
- 树
- 对象

字符串描述大多数在 XML 中使用的数据类型,它完全由字符组成。树描述那些被解析成层次式树结构的 XML 代码。XFA 允许你在以字符串方式存储的 XML 源代码和以树存储的解析 XML 代码之间进行方便的转换。对象描述如数据库结构这样的复杂数据类型。在本书编写期间,XFA 的当前版本并不支持对象,但对象支持已经在未来版本的开发任务中。

这些数据类型中,主要的数据类型是你用来存储数据的特殊数据类型。下面是 XFA 当前版本中支持的数据类型:

- string——一系列的字符。
- boolean——值为"true"或"false"的特殊字符串。
- integer——由数字组成的特殊字符串,字符串前可加负号(-)。
- currency——由美元符(\$)及后跟的数字组成的特殊字符串。
- date——由日期和当前时间组成的特殊字符串。
- function——函数的引用。
- tree——包含链接的 XML 节点的层叠式数据结构。
- tree sequence——从树中选择的一系列的子树。
- void——一个空数据(通常作为评价函数的返回值)。

这些数据类型的大多数实际上可被认为是字符串数据类型,因为它们在 XFA 代码中以文本方式表述。更为明确的是,string、Boolean、integer、currency、date 和 function 类型都是字符串。tree 和 tree sequence 类型是 tree 类型,它们的独特之处在于它们描述了解析 XML 代码的树。剩下的 void 类型是用来指示不含数据的特殊类型。表达式有时返回一个 void 类型以指出没有与表达式结构相关的数据。

对表达式来说,它们构造了为什么数据类型在 XFA 中如此重要的基础。当表达式运行时,返回的结构通常是一个特定的数据类型。例如,下面是你在本章前面看到的表达式,它包含一个函数调用:

```
<xfa, val exp='salutation("Eddie")' />
```

当 XFA 解释器计算运行这个表达式时,返回的字符串结果如下:

```
<p>Hello <i>Eddie</i>, how are you? </p>
```

所有的表达式在结果被运行出来之后结束,但并不是所有的表达式都返回数据。如果一个表达式的运行并没有导致有数据返回,那么返回的值就是类型 void 的。

28.4 XFA 词表

XFA 以 XML 词表实现,这就意味着创建一个 XFA 脚本非常类似于创建 XML 文件。第一步就是要学习词表的元素集(标记集),然后你要学习如何使用这些元素创建有意义的脚本。所有的 XFA 元素都包含在 XFA 名字域中,这个域通过前缀 xfa 定义。这意味着所有

的 XFA 标记都以前缀 xfa: 开头。下面是组成 XFA 词表的元素：

- xfa:val
- xfa:block
- xfa:function
- xfa:use
- xfa:if
- xfa:orif
- xfa:else
- xfa:for
- xfa:while
- xfa:string
- xfa:break
- xfa:let
- xfa:data
- xfa:tag
- xfa:ref
- xfa:form
- xfa:note

下面的几节将逐一讲解如何在 XFA 脚本中使用这些元素。

除了构造 XFA 词表的元素以外，预定义函数弥补了 XFA 脚本语言的一个重要的方面。XFA 提供预定义函数来运行各种各样的任务，如数学运算和文件输入输出。你还可以使用 XFA 中的库函数，它们是在特定库中定义的预定义函数，它们必须在使用前在脚本中声明引用。如果没有预定义函数和库函数可以运行所给任务，你还可以创建你自己的函数。预定义函数和库函数将在本章的剩余部分中使用。你也可以在本章的“编写 XFA 脚本”一节中学习如何创建你自己的函数。

28.4.1 xfa:val 元素

xfa:val 元素用来运行一个表达式。要使用 xfa:val 元素，你要以属性方式提供一个表达式并用 XFA 解释器运行它。运算后的表达式的结构放入到 xfa:val 元素中。下面是一个使用应用扩展 XFA 语法的 xfa:val 元素的例子。

```
<xfa:val Add(x,y)/>
```

在这个例子中，预定义函数 add() 用作为 xfa:val 元素的属性，这个函数用来将两个变量相加。

28.4.2 xfa:block 元素

xfa:block 元素用来将代码组织进名为块的逻辑区域内，块非常的重要，因为它们确定了变量的作用域，并控制名字的可见性。作用域确定了变量的生存期，这就意味着给定作用域的变量只有代码在指定作用域中运行时才有效。下面是一个使用 xfa:block 元素的例子。


```
<xfa:block>
  <xfa:val Add(x,y)/>
</xfa:block>
```

28.4.3 xfa:function 元素

xfa:function 元素用来声明函数。函数是脚本中可以执行特定功能的过程,比如数值相加或字符串排序。

函数可以接收参数,这些参数是一些可以传入到函数中并在某种程度上影响函数输出的值。例如,x 和 y 是你在前面两小节看到的 Add()函数的参数。

Add()函数是 XFA 内建的预定义函数。也许你需要更专门的数学计算函数,比如计算圆面积的函数。下面是一个使用 xfa:function 元素创建你自己的 CircleArea()函数的例子。

```
<xfa:function CircleArea(radius)>
  <xfa:val Mult(3.14,Mult(radius,radius))>
</xfa:function>
```

在这个例子中,预定义的 Mult()函数用来计算圆半径的平方并将它与圆周率(3.14)相乘。最后乘积的结果就是 Mult()函数的返回值。

28.4.4 xfa:use 元素

xfa:use 元素用来引用另一个 XFA 脚本。在 xfa:use 元素中指定的名字是一个脚本的路径,这个路径是脚本在 XFA 目录结构中的路径。在脚本中通过 xfa:use 引用来定义的顶层名称被导入到当前脚本中 xfa:use 元素出现的位置。

28.4.5 xfa:if、xfa:orif 和 xfa:else 元素

xfa:if、xfa:orif 和 xfa:else 元素用来设置脚本的条件分支。更重要的是,这些元素允许脚本分析信息并根据结果运行不同的脚本代码。当联合使用时,这些元素与其他脚本语言和编程语言中的条件分支结构非常类似。xfa:orif 元素的作用相当于正规编程语言(如 C/C++ 或 Java)中的 else-if 语句。在给定的内容中只允许出现一个 xfa:if 元素,而在 xfa:if 元素后可以出现多个 xfa:orif 元素。

xfa:if 和 xfa:orif 的条件属性必须得出一个 Boolean 值,这也是确定元素的条件分支的因素。如果条件属性是真,元素的内容就被运行。Xfa:else 元素允许你提供一个预备分支,当 xfa:if 或 xfa:orif 的条件运算为假时运行这个分支。在一个 xfa:if 元素和一组 xfa:orif 元素后只能出现一个 xfa:else 元素。下面是一个使用这些元素设置条件分支的例子:

```
<xfa:if IntLs(i,0)>
  <xfa:val "The number is less than zero."/>
</xfa:if>
<xfa:orif IntEq(i,0)>
  <xfa:val "the number is equal to zero."/>
</xfa:orif>
<xfa:else>
```

```
<xfa:val "The number is greater than zero." />
</xfa:else>
```

在这个例子中,数字被检查,看它是否小于 0、等于 0 或大于 0。xfa:if、xfa:orif 和 xfa:else 元素的联合使用实现了这个条件分支。

28.4.6 xfa:for 元素

xfa:for 元素用来重复运行脚本中的代码。它允许你仔细的构造循环执行代码预定的次数。xfa:for 元素运行的代码包含在元素的内容中。代码运行的次数由元素的属性提供。如果属性值是一个整数,循环将运行整数指定的次数。xfa:for 元素的属性值也可以是树或树序列类型,这种情况下,代码将对树中的每一个节点都运行一次。

注意: xfa:for 元素确定的循环可以被 xfa:break 元素终止。

下面是一个使用 xfa:for 元素创建循环的例子:

```
<p> Count:
<xfa:for i=20>
  <xfa:val i />
</xfa:for>
</p>
```

这个例子显示了如何用 xfa:for 元素计数至 20,并显示结果。要记住循环反复的次数是由名为 i 的变量指定的。这个变量计数循环中成功循环的值。

警告: 给 xfa:for 循环的循环索引数赋值的企图将导致错误。

28.4.7 xfa:while 元素

xfa:while 元素与 xfa:for 循环功能相同,只是在循环反复次数的控制上有所不同。除了运行固定次数的循环代码外,xfa:while 元素在元素的属性为真的时候执行循环代码。一旦属性表达式计算结果为假,循环就被终止。你也可以使用 xfa:break 元素来终止循环。

下面是使用 xfa:while 元素实现的与上面同样计数的例子。

```
<p> Count:
<xfa:let i=0 />
<xfa:while IntLs(i,20)>
  <xfa:val Assign(i,Add(i,1)) />
  <xfa:val i />,
</xfa:while>
</p>
```

在这个例子中,我们有必要使用一个变量来记录循环次数,因为 xfa:while 元素并不是为指定次数循环而设计的。IntLs() 预定义函数作为循环的条件部分,确定何时跳出循环。同样,Assign() 预定义函数用来递增 i 变量。

28.4.8 xfa:string 元素

xfa:string 元素是另一个循环元素,它允许你循环处理字符串中的数值。xfa:string 元素

的属性包括一个整数表达式,用来确定重复循环的次数。`xfa:string` 元素的内容必须是整数值,它代表的是从字符串中选取的字符。下面的例子显示了如何使用 `xfa:string` 元素从一个著名画家的名字中提取字符。

```
<xfa:let s="Frank Lloyd Wright"/>
<xfa:string i=5>
  <xfa:val s(Add(i,6))/>
</xfa:string>
```

警告:给 `xfa:` 循环的循环索引赋值的企图将导致错误。

在这个代码中,`xfa:string` 元素用来在字符串变量 `s` 中的五个字符中重复执行。这些字符通过使用圆括号和循环变量 `i` 被字符串变量 `s` 引用。

注意:与 `xfa:for` 和 `xfa:while` 元素相同,`xfa:string` 元素确定的循环可以被 `xfa:break` 元素终止。

28.4.9 xfa:break 元素

`xfa:break` 元素用来终止 `xfa:for`、`xfa:while` 和 `xfa:string` 的循环。当循环遇到了 `xfa:break` 元素时,当前的循环重复就被终止并跳出循环,不论循环条件或循环重复变量的状态如何。下面是一个使用 `xfa:break` 元素跳出 `xfa:for` 循环的例子:

```
<p>Count:
<xfa:for i=20>
  <xfa:val i/>
  <xfa:if IntEq(i,11)>
    <xfa:break/>
  </xfa:if>
</xfa:for>
</p>
```

在这个例子中,`xfa:for` 循环在计数到 11 时,由于遇到了 `xfa:break` 元素而退出循环。

28.4.10 xfa:let 元素

`xfa:let` 元素用来声明一个脚本变量并给它赋初始值。变量和它的初始值使用 `xfa:let` 元素的属性指定。下面是使用 `xfa:let` 元素声明变量的例子:

```
<xfa:let i=0/>
<xfa:let s="Frank Lloyd Wright"/>
```

这两个例子示范了如何使用 `xfa:let` 元素声明整数和字符串变量。要在声明后修改这些变量,你必须使用预定义函数 `Assign()`。下面是一个使用 `Assign()` 函数修改变量 `i` 的值的例子。

```
<xfa:val Assign(i,65)>
```

28.4.11 xfa:data 元素

xfa:data 元素用来将 XML 数据结构插入到脚本中。评估放置在 xfa:data 元素内容中的 XML 数据的结果是一个由 xfa:block 元素组成的解析树。

28.4.12 xfa:tag 元素

xfa:tag 元素用来将一个标识标记指定为字符串,通常是在 HTML 中。xfa:tag 元素运行的结果是 xfa:tag 元素的属性中一个包含标识标记的字符串。

28.4.13 xfa:ref 元素

xfa:ref 元素用来确立两个网页间的链接。这是 XFA 中一个非常强大的元素,因为它允许你在页面间调用函数和传递参数。xfa:ref 元素中定义的实际链接是一个定义在另一个 XFA 脚本上的顶层 XFA 函数的调用。在被调用的函数中传送链接结果,这就生成了一个新的网页。下面是一个使用 xfa:ref 元素链接网页的简单的例子:

```
<xfa:ref CountFor()>Click here to go count! </xfa:ref>
```

这个元素的内容作为网页的超级链接出现。这个代码连接了一个函数调用给超级链接,这意味着点击超级链接文本将调用这个函数。点击在 XFA 解释器中加载的名为 CountFor.xfa 的文本并运行名为 CountFor()的顶层函数。这里有一些选择属性可以让你指定如何显示新的页面。这并不在本次讨论范围之内,请查阅 XFA 文件以获得有关这些属性的更多信息。

28.4.14 xfa:form 元素

xfa:form 元素用来处理 HTML 表单输入的信息。HTML 表单控制被放置在 xfa:form 元素中,一个 XFA 函数可以被调用处理表单体中的结果。输入表单中的输入信息被所调用的函数处理,这就允许你编写专门的脚本来处理输入信息。脚本函数必须是一个 XFA 脚本文件中的顶层函数。清单 28.1 包含了名为 GetAge.xfa 的 XFA 脚本,它示范了如何使用 xfa:form 元素来处理 HTML 表单。

清单 28.1 处理 HTML 表单的 XFA 脚本 GetAge.xfa

```
<xfa:function GetAge>
  <head>
    <title>
      <GetAge XFA Example
    </title>
  </head>
  <body bgcolor="white">
    <h2> GetAge XFA Example</h2>
    <xfa:form AssessAge>
      Please enter your age: <input type="text" name="Age">
      <input type="submit" value="AssessAge">
    </xfa:form>
```



```
</body>
</xfa:function>
```

这段代码是你所看到的第一个完整的 XFA 脚本例子。脚本使用 `xfa:form` 元素调用 `AssessAge()` 函数响应用户在 HTML 表单中输入的他的年龄。`AssessAge()` 函数在脚本文件 `AssessAge.xfa` 中作为顶层函数出现。这个文件如清单 28.2 所示。

清单 28.2 处理 HTML 表单中用户输入的 XFA 脚本 `AssessAge.xfa`

```
<xfa:function AssessAge>
  <head>
    <title>
      AssessAge XFA Example
    </title>
  </head>
  <boy bgcolor="white">
    <h2>AssessAge XFA Example</h2>
    <xfa;if IntLs(passed ^ Age,20)>
      <p>I sense that you're a bright-eyed youngster. </p>
    </xfa;if>
    <xfa;orif IntLs(passed ^ Age,30)>
      <p>Clearly, you're one of those pesky Generation X'ers. </p>
    </xfa;orif>
    <xfa;orif IntLs(passed ^ Age,40)>
      <p>I foresee the forties in your future. </p>
    </xfa;orif>
    <xfa;orif IntLs(passed ^ Age,50)>
      <p>Just think, you're only about 20 years away from retirement! </p>
    </xfa;orif>
    <xfa;orif IntLs(passed ^ Age,60)>
      <p>Don't worry, it's all downhill from here. </p>
    </xfa;orif>
    <xfa;else IntLs(passed ^ Age,70)>
      <p>Why bother with these newfangled technologies? </p>
    </xfa;else>
  </body>
</xfa:function>
```

这段代码中,最需要注意的一点就是对变量 `Age` 的引用。这个变量实际以 `Passed ^ Age` 方式引用,它包含了在前面的网页上的 HTML 表单中用户输入的年龄。脱字符号(^)用来访问结构中的数据成员,这在 XFA 代码中十分常用。这样,传来的结构中就包括了从 HTML 表单中传入到 `AssessAge()` 函数的 `Age` 数据成员。图 28.1 和 28.2 显示了 `age` 脚本在 Internet Explorer 5.0 中的运行情况。

你可以使用 `xfa:form` 元素的更多属性以更精确的处理 HTML 表单数据。由于这并不在本次讨论范围之内,请查阅 XFA 文件以获得有关这些属性的更多信息。

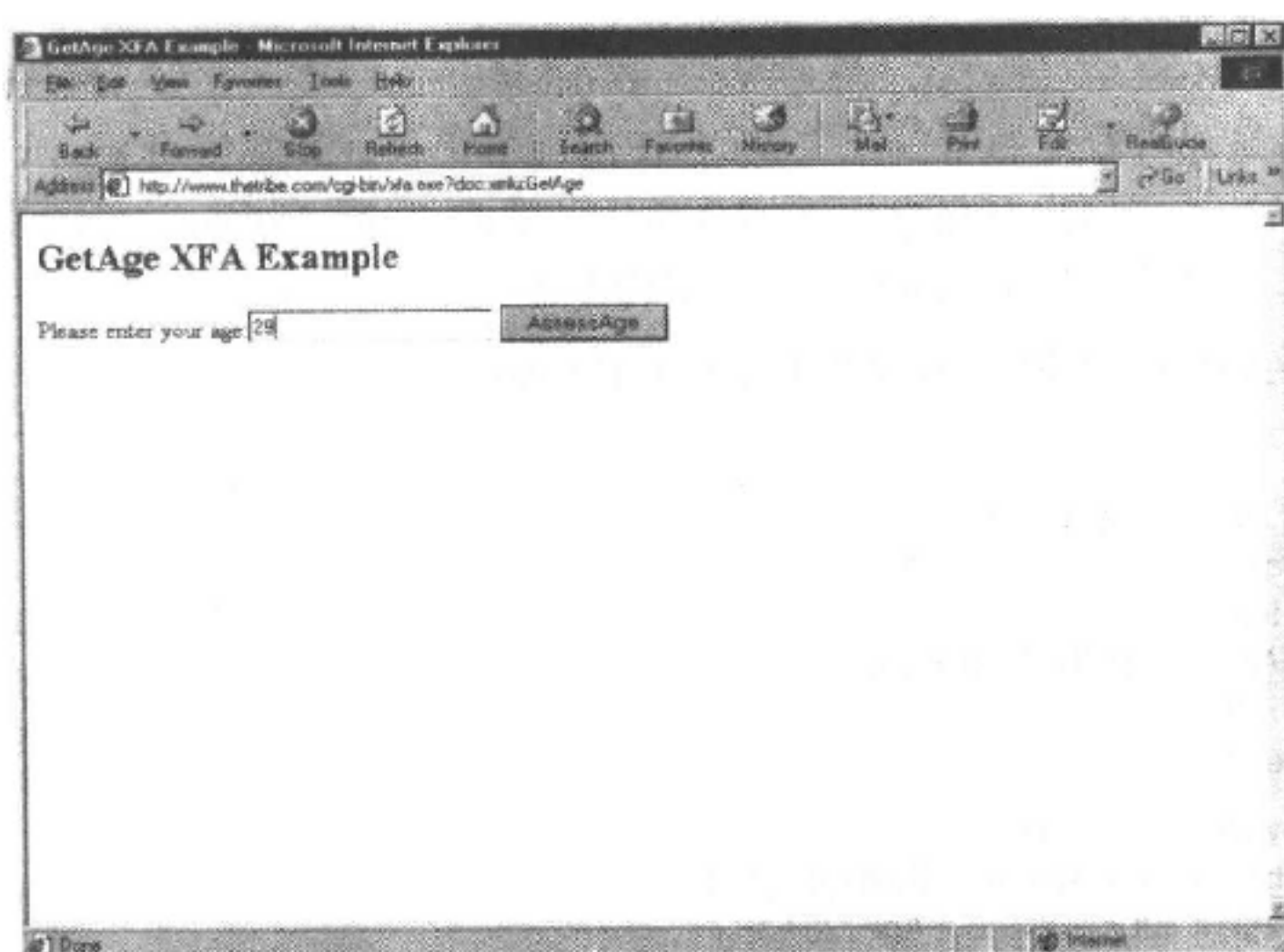


图 28.1 XFA 脚本 GetAge 在 Internet Explorer 5.0 中的运行情况

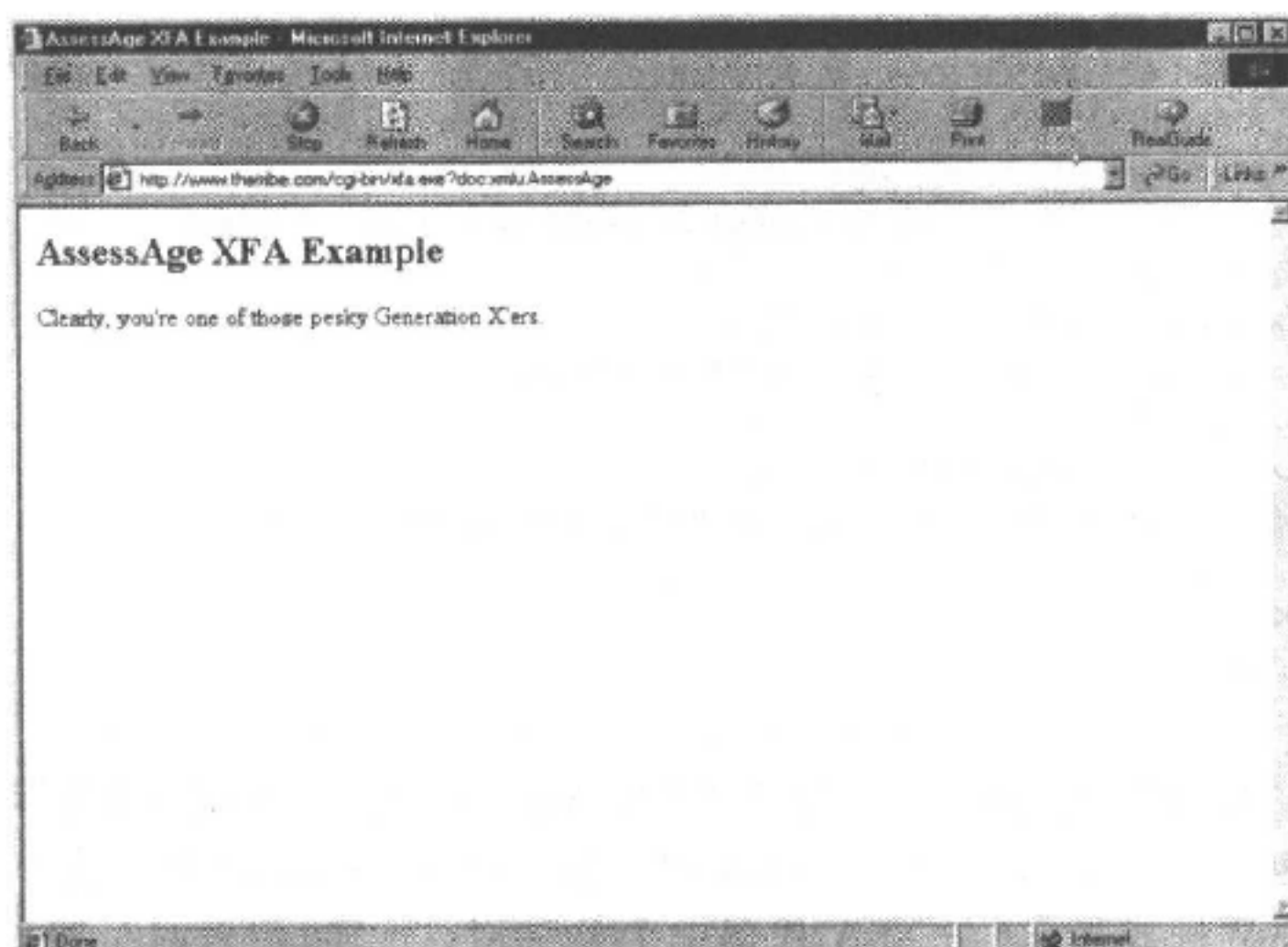


图 28.2 XFA 脚本 AssessAge 在 Internet Explorer 5.0 中的运行情况

28.4.15 xfa:note 元素

xfa:note 元素用来提供 XFA 脚本中的注释。你可以使用 xfa:note 元素指定注释为属性

或是元素内容。下面是有关如何使用 xfa:note 元素指定注释的例子：

```
<xfa:note "This is just a comment." />
<xfa:note> This is another comment. </xfa:note>
```

28.5 编写 XFA 脚本

只是学习构成 XFA 词表的所有元素,对于学习创建有用的 XFA 脚本还并不足够。然而,这样做也确实可以提供一个理解 XFA 脚本语言的基础的能力。这一节将向你介绍一些通过 XFA 创建的实用脚本实例,它们可以执行一些有趣的任务。

28.5.1 从文本文件中读取

XFA 的一个有趣的应用就是从文本文件中读取数据并将它合并到网页之中。清单 28.3 包含了脚本 Fortune.xfa 的代码,它从一个文本文件中读取报价清单,并把它们在网页上显示出来。

清单 28.3 从文本文件中读取并显示报价的 XFA 脚本 Fortune.xfa

```
<xfa:function Fortune>
  <xfa:use library:path/>
  <head>
    <title>
      Fortune XFA Example
    </title>
  </head>
  <body bgcolor="white">
    <h2>Fortune XFA Example</h2>
    <xfa:val Hencode(Read(FilePath("fortunes.txt")))/>
  </body>
</xfa:function>
```

也许这个脚本中最为有趣的部分就是元素 xfa:use,它指出脚本需要在 library:path 中定义的代码的支持。这个库中指定的代码是函数 FilePath()。它实现一个对 XFA 所使用的当先目录下的相关的文件引用。预定义函数 Read()读取一个文本文件并以字符串的形式返回结果。这个字符串通过使用预定义函数 HEncode()被转换成 HTML 以进行显示。这个转换需要将特殊的实体(比如 <)转换为与之等同的文字(这里是<)。

图 28.3 显示了 Fortune.xfa 脚本在 Internet Explorer 5.0 中的运行情况。

28.5.2 处理 XML 代码

XFA 在处理 XML 代码方面非常的老到。你可以使用预定义的 XFA 函数将 XML 代码解析成树,也可以将一个树反解析为 XML 代码。清单 28.4 包含了一个名为 ParseXML.xfa 的代码,它以文本的方式读取一个 XML 文件,将文件解析成一个树,并在之后反解析这棵树并显示 XML 代码。

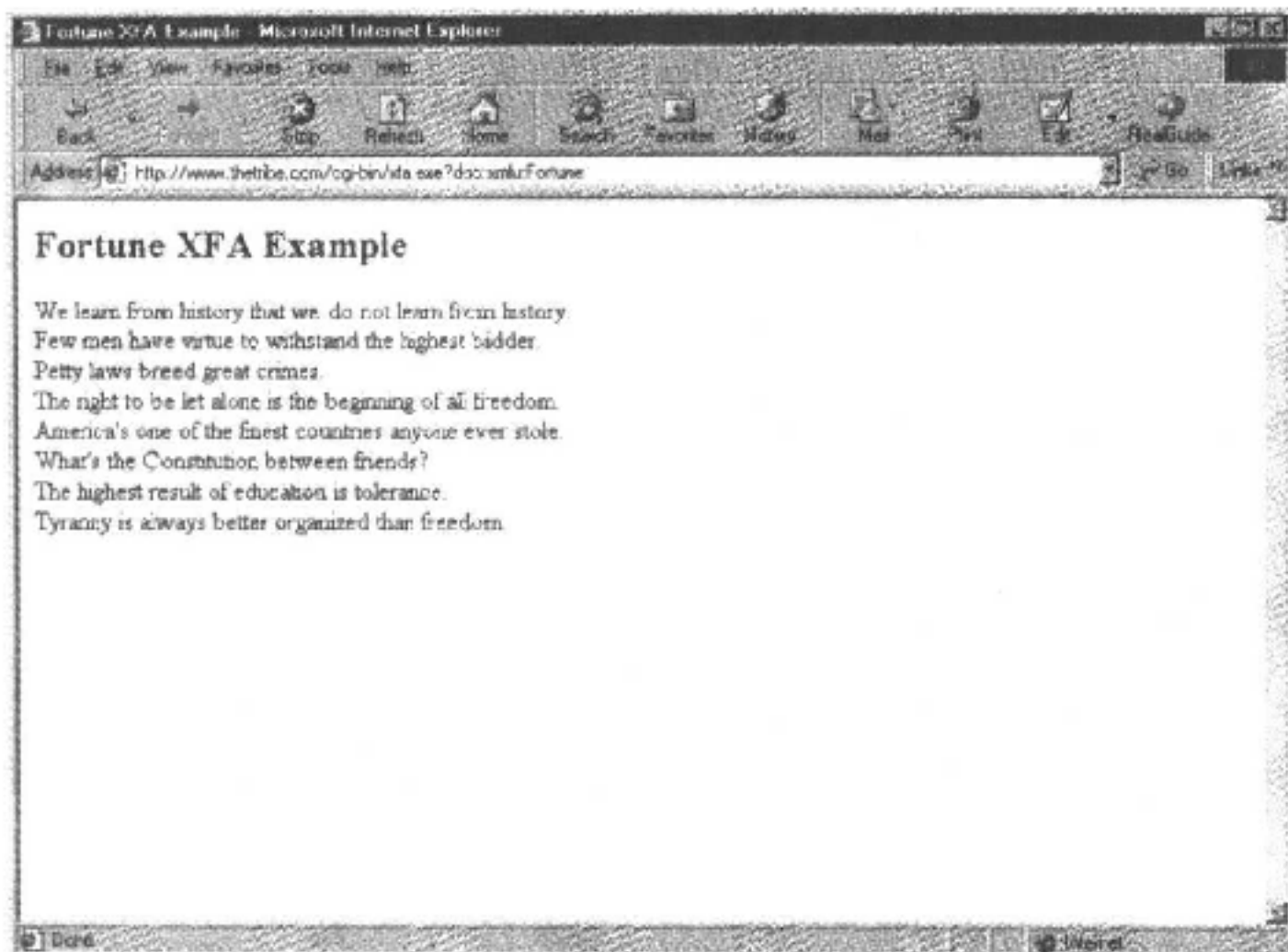


图 28.3 XFA 脚本 Fortune 在 Internet Explorer 5.0 中的运行情况

清单 28.4 解析和反解析 XML 文件的 XFA 脚本 ParseXML.xfa

```
<xfa:function ParseXML>
  <xfa:use library:path/>
  <head>
    <title>
      Parse XFA Example
    </title>
  </head>
  <body bgcolor="white">
    <h2>parse XFA Example</h2>
    <xfa:let tree=Parse(Read(Filepath("Vehicles.xml")))/>
    <xfa:val Hencode(Unparse(tree))/>
  </body>
</xfa:function>
```

这个脚本如此简单,很难想到它完成了所有要做的工作。XML 文件 Vehicles.xml 首先被读入,之后使用预定义函数 Parse() 解析成 tree 变量。tree 变量包含一个表述 XML 文件的树,它包含能够用来访问每个元素和属性的节点。之后 tree 变量使用预定义函数 Unparse() 被反解析回 XML 代码。我们需要将反解析后的 XML 代码传入到预定义函数 Hencode() 中以确保所有的特征都确实的转换了。

图 28.4 显示了执行 ParseXML.xfa 脚本的结果。

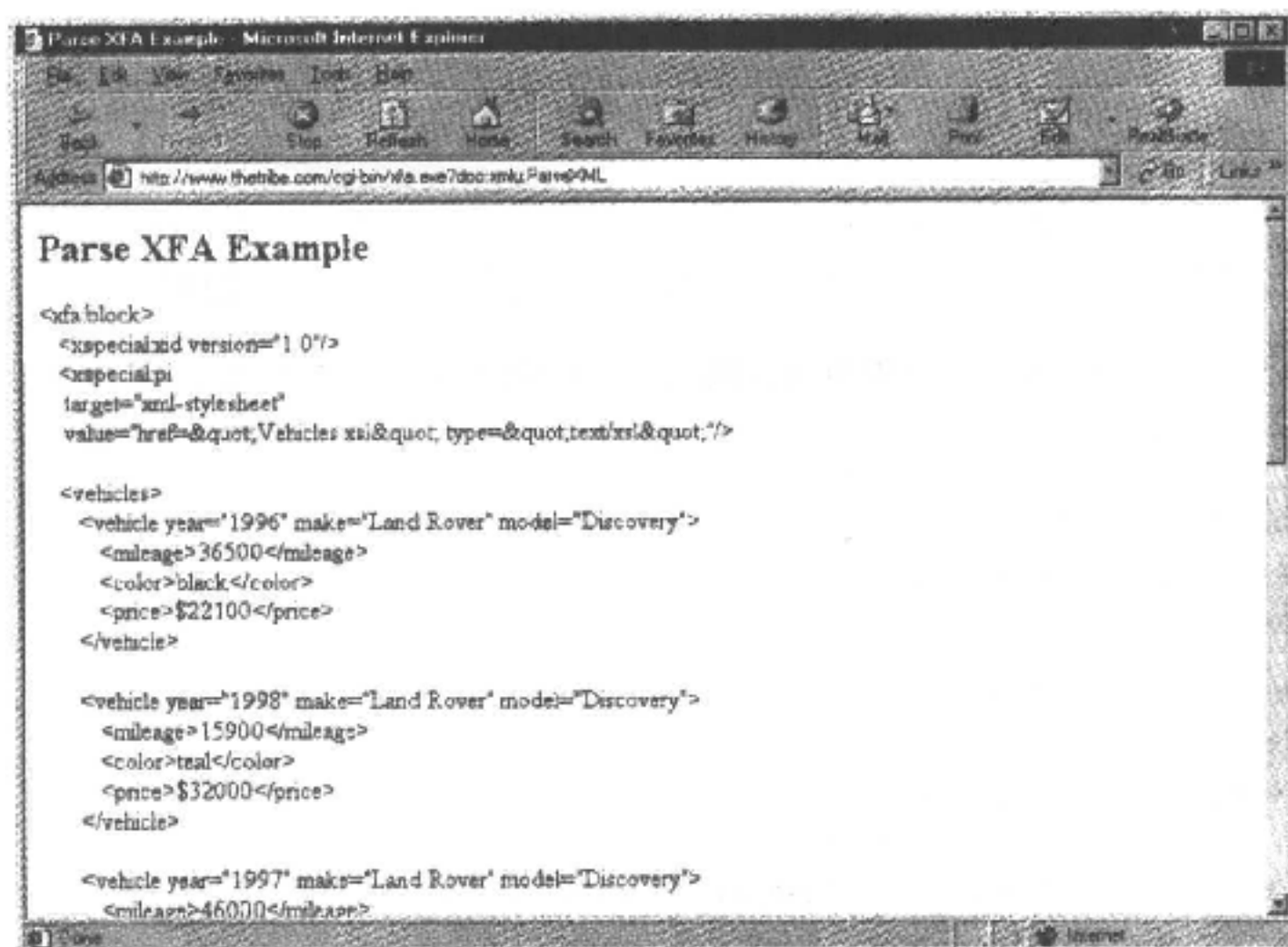


图 28.4 XFA 脚本 ParseXML 在 Internet Explorer 5.0 中的运行情况

如果你回忆一下,你就会想起 Vehicles.xml 文件是在以前的章节中产生的。然而,你可能会注意到,在清单中显示的 XML 代码与原始的 XML 文件有一些不同。这是因为 Unparse() 函数在反解析时独立承担着格式化 XML 代码的工作,这就导致了文件之间的一些差异。虽然如此,反解析后的文件与原始的文件在功能上是等价的。Unparse() 函数也将 XML 代码放入到 xfa:block 元素中,这在 XML 文件包含多个顶级元素的时候有助于排除问题。

与我们知道了 XFA 可以解析和反解析 XML 代码同样有趣的是,实际上 XFA 的有用之处是遍历一个 XML 数据的树。例如,你可能希望遍历 Vehicles.xml 的数据树并确定在文件中给出了多少种汽车规格。清单 28.5 包含一个脚本的代码,它提示使用者在 HTML 表单中输入汽车的规格。

清单 28.5 通过使用 HTML 表单来输入汽车规格的 XFA 代码 GetMake.xfa

```
<xfa: function GetMake>
  <head>
    <title>
      GetMake XFA Example
    </title>
  </head>
  <body bgcolor="white">
    <h2>GetMake XFA Example</h2>
    <xfa: form CountMake>
      Please enter the make of car: <input type="text" name="Make">
      <input type="submit" value="CountMake">
    </xfa: form>
```

```

</body>
</xfa:function>

```

这个例子与你在本章前面所看到的脚本 `GetAge` 非常相似。然而,在这里,用户的输入将用来从一个 XML 文件中提取信息。清单 28.6 包含一个脚本的代码,它遍历 `Vehicles.xml` 文件并计算与用户在 `GetMake` 脚本中输入的规格相匹配的汽车的数目。

清单 28.6 计算与在 `GetMake` 脚本中输入的规格相匹配的汽车的数目的 XFA 脚本 `CountMake.xfa`

```

<xfa:function CountMake>
  <xfa:use library:path/>
  <xfa:let makeCount=0/>

  <xfa:function GetMakeCount(tree)>
    <xfa:if E1(tree ^ TreeKind,"tag")>
      <xfa:for t1=tree ^ TreeAttrs>
        <xfa:if Eq(t1 ^ TreeKind,"equal")>
          <xfa:if And(E1(t1 ^ TreeLeft ^ TreeKind,"name"),
            Eq(t1 ^ TreeRight ^ TreeKind,"string"))>
            <xfa:if And(E1(t1 ^ TreeLeft ^ TreeValue,"Make"),
              Eq(t1 ^ TreeRight ^ TreeValue,passed ^ Make))>
              <xfa:val Assign(makeCount,Add(makeCount,1))/>
            </xfa:if>
          </xfa:if>
        </xfa:if>
      </xfa:for>
      <xfa:for t2=tree ^ TreeBody>
        <xfa:val GetMakeCount(t2)/>
      </xfa:for>
    </xfa:if>
  </xfa:function>

  <head>
    <title>
      CountMake XFA Example
    </title>
  </head>

  <body bgcolor="white">
    <h2>CountMake XFA Example</h2>
    <xfa:let tree=Parse(Read(filepath("Vehicles.xml")))/>
    <xfa:val GetMakeCount(tree ^ TreeBody(1))/>
    <p>There are <xfa:val makeCount/> vehicles made by <xfa:val
      passed ^ Make/>. </p>
  </body>
</xfa:function>

```

这段代码示范了 XFA 的一个非常有趣的特性。它显示了如何遍历一个 XML 数据树并寻找指定的信息。脚本 `CountMake` 非常依赖于 XFA 的数据类型 `tree`,而这个类型确实很复杂。我不准备在数据类型 `tree` 方面再浪费更多的口舌,但我希望解释 `CountMake` 脚本的基础。在代码的底部是这样的代码,它读取 `Vehicles.xml` 文件并将它解析为一个树。函数

GetMakeCount()被调用以确定与使用者的规格相匹配的汽车的数目。

GetMakeCount()函数首先确认树中有类型标记,这就意味着它由标记元素组成。XFA 支持不同的树的类型,因此一定要确保你知道你正在做什么。下一步就是使用 xfa:for 循环在树中反复运行,一次一个元素。实际上,树遍历寻找 equal 类型的子树,这种子树是一个名称/值对。当这样的子树被找到,它的名称和值就被检验是否匹配用户选择的规格。如果符合,变量 makeCount 就加一。

由于第一个 xfa:for 循环只有沿一个树分支执行的能力,因而你需要使用一个附加的 xfa:for 循环来继续遍历整个树。GetMakeCount()函数进行自身递归的调用以实现这个功能。要了解有关数据结构 tree 的更多信息以及它的不同的成员表现,请查阅 XFA 文件。

图 28.5 和图 28.6 显示了汽车规格脚本在 Internet Explorer 5.0 中的运行情况。

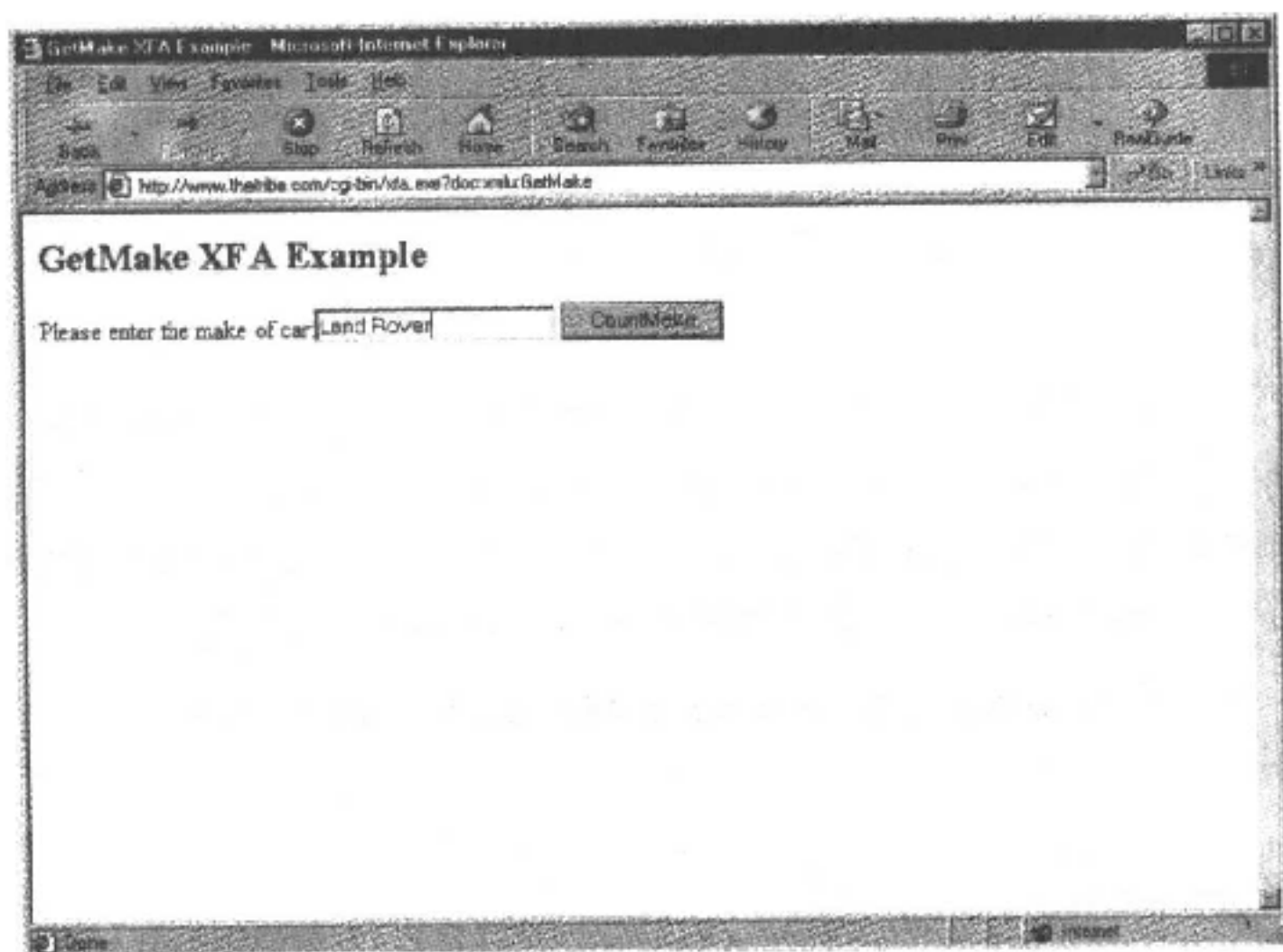


图 28.5 XFA 脚本 GetMake 在 Internet Explorer 5.0 中的运行情况

28.6 XFA 和 DTD

考虑到 XFA 脚本使用扩展的 XML 语法,你可能想要知道 DTD 如何适应 XFA。除了来自于它自身所扩展的语法外,XFA 也有着它自己的 DTD 语法。术语“DTD 语法”有些容易令人误解,因为对传统 XML DTD 来说,它与 XMLSchema 更为类似。与 XMLSchema 类似,XFA 的 DTA 语法依赖常见的 XML 语法来描述文件结构。换句话说,为了代替使用在传统 DTD 中的神秘代码,你要使用元素和属性来表述一个文件的结构。使用 XFA 的 DTD 语法创建的文件也就是 XFASchema。

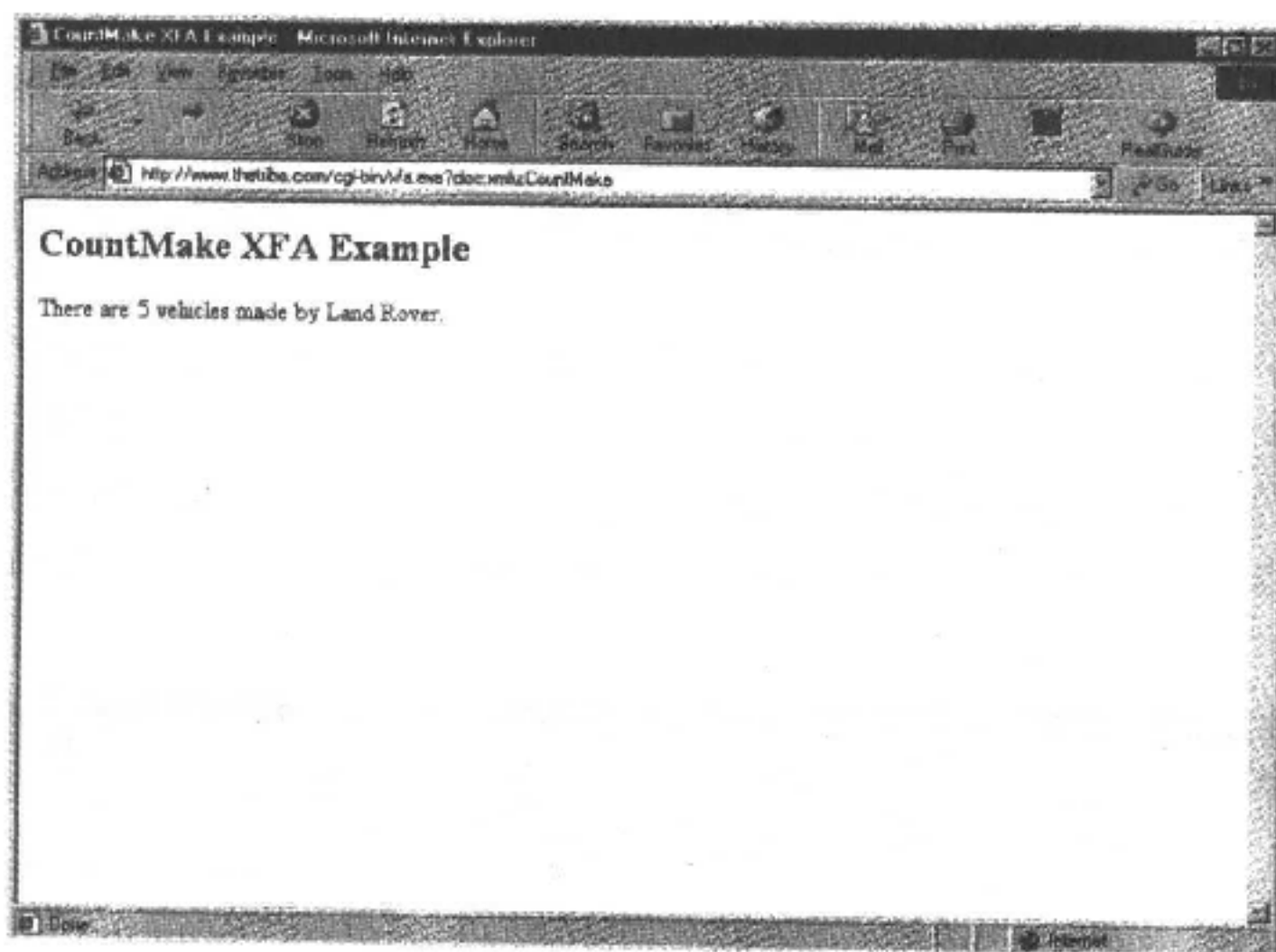


图 28.6 XFA 脚本 CountMake 在 Internet Explorer 5.0 中的运行情况

为了帮助你实现从传统 DTD 到 XFASchema 的转换, XFA 提供了一个特殊的函数, 它允许你自动的将 DTD 转换为 XFASchema。清单 28.7 包含了一个脚本的代码, 它使用库函数 DTDNormalize() 将 XML DTD 数据树转换为 XFASchema 数据树。

清单 28.7 将 XML DTD 转换为 XFASchema 语法的 XFA 脚本 DTDtoXFA

```
<xfa:function DTDtoXFA>
  <xfa:use library:xml>
  <xfa:use library:dtd/>
  <xfa:use library:path/>

  <head>
    <title>
      DTDtoXFA XFA Example
    </title>
  </head>

  <body bgcolor="white">
    <h2>DTDtoXFA XFA Example</h2>
    <xfa:let text=Read(Filepath("Movies. dtd"))/>
    <xfa:let tree=Parse(text) ^ TreeBody(1)/>
    <xfa:val DTDNormalize(tree)/>
    <xfa:val Hencode(Unparse(tree))/>
  </body>
</xfa:function>
```

这个脚本令人难以置信的简单, 特别是当你考虑到它要将 XML DTD 自动转换为 XFASchema 时更是如此。DTD 文件 movies 首先被读入并解析为一个 XML 树。转换通过函

数 DTDNormalize() 实现, 这是 XFA 库 library:dtd 的一部分。转换后的树被反解析, 编码并显示。图 28.7 显示了运行 DTDtoXFA 脚本的结果。

movieDTD 的 XFASchema 表示对运行任何对 XFA 的 movies 有效的文件来说, 都是非常必要的。清单 28.8 包含了由 DTDtoXFA 脚本转换的 movieDTD 的 XFASchema 版本。图 28.8 包含了通过 DTDtoXFA 脚本转换的 movie DTD 文件的 XFASchema 版本。

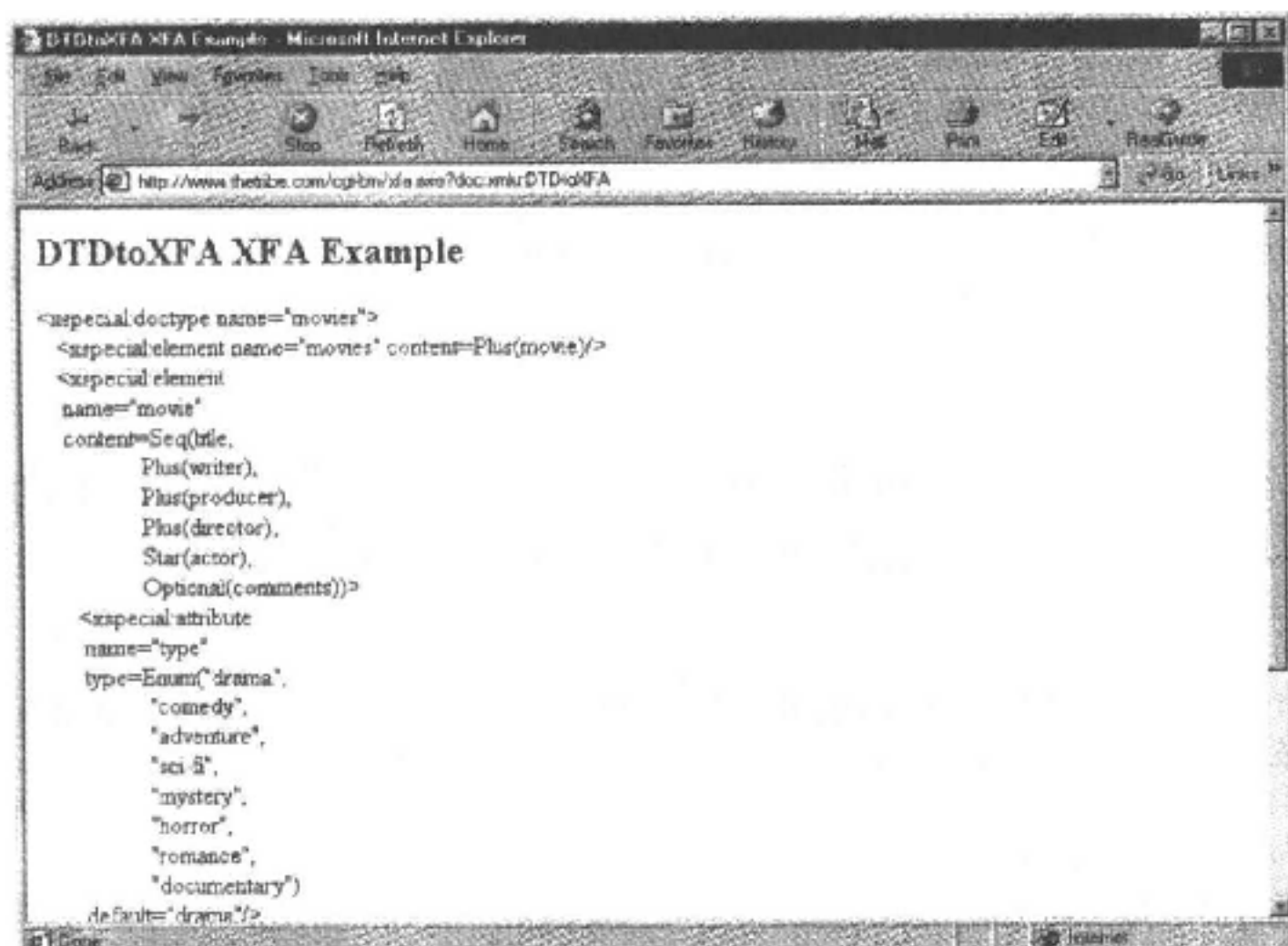


图 28.7 XFA 脚本 DTDtoXFA 在 Internet Explorer 5.0 中的运行情况, 它显示了被转换为 XFASchema 语法的 DTD 文件 movies

清单 28.8 包含转换 DTD 文件 movies 功能的 XFA 脚本 MoviesDTD.xfa

```
<xspecial:doctype name="movies">
  <xspecial:element name="movies" content=Plus(movies)/>
  xspecial:element
    name="movie"
    content=Seq(title,
      plus(writer),
      Plus(producer),
      Plus(director),
      Star(actor),
      Optional(comments))>
  <xspecial:attribute
    name="type"
    type=Enum("drama",
      "comedy",
      "adventure",
      "sci-fi",
      "mystery",
      "horror",
      "romance",
      "documentary")
    default="drama"/>
```

```

    "romance",
    "documentary" )
    default="drama"/>
    <xspecial:attribute name="rating" type=Enum("G","PG","PG-13","R","X")
    default="PG"/>
    <xspecial:attribute name="review" type=Enum("1","2","3","4","5")
    default="3"/>
    <xspecial:attribute name="year" type=text optional="true"/>
    <xspecial:element>
    <xspecial:element name="title" content=Text()/>
    <xspecial:element name="writer" content=Text()/>
    <xspecial:element name="producer" content=Text()/>
    <xspecial:element name="director" content=Text()/>
    <xspecial:element name="actor" content=Text()/>
    <xspecial:element name="comments" content=Text()/>
    </xspecial:doctype>

```

现在你就有了 XFASchema 格式的 DTD 文件 movies, 你可以使用它来验证 XML 文件 movie。清单 28.9 包含了一个脚本的代码, 它用 XFASchemamovies 验证(检查)一个 XML 文件。

清单 28.9 依据相应的 XFASchema 检查一个 XML 文件的 XFA 脚本 CheckXFA.xfa

```

<xfa:function CheckXFA>
  <xfa:use library:dtd/>
  <xfa:use library:error/>
  <xfa:use library:path/>

  <head>
    <title>
      CheckXFA XFA Example
    </title>
  </head>

  <body bgcolor="white">
    <h2>CheckXFA XFA Example</h2>
    <xfa:let dtdText=Read(FilePath("MoviesDTD.xfa"))/>
    <xfa:let dtdTree=Parse(dtdText)^TreeBody(1)/>
    <xfa:let errorTree=TreeTag("errors")/>
    <xfa:let xmlText=Read(FilePath("Movies.xml"))/>
    <xfa:let xmlTree=Parse(xmlText,errorTree)^TreeBody(1)/>
    <p>Checking Movies.xml...</p>
    <xfa:val DTDCheckXFA(xmlTree,dtdTree)/>
    <xfa:if IntEq(Size(errorTree^TreeBody),0)>
      <p>No errors detected.</p>
    </xfa:if>
    <xfa:else>
      <p><xfa:val Size(errorTree^TreeBody)/>errors were found:</p>
      <xfa:val HorrorMerge(xmlText,errorTree)/>
    </xfa:else>
  </body>
</xfa:function>

```

这个脚本在耗费它绝大部分精力的读取和解析两个文件(XML 文件和 XFASchema)方面非常的直截了当。一旦文件被成功的读入并被解析成 XML 树,库函数 DTDCheckXFA()就负担起实际使用 DTD 检查文件的杂务。错误的树被检验以看是否能找出什么错误。图 28.8显示了在 Internet Explorer 5.0 中运行 CheckXFA 脚本的结果。

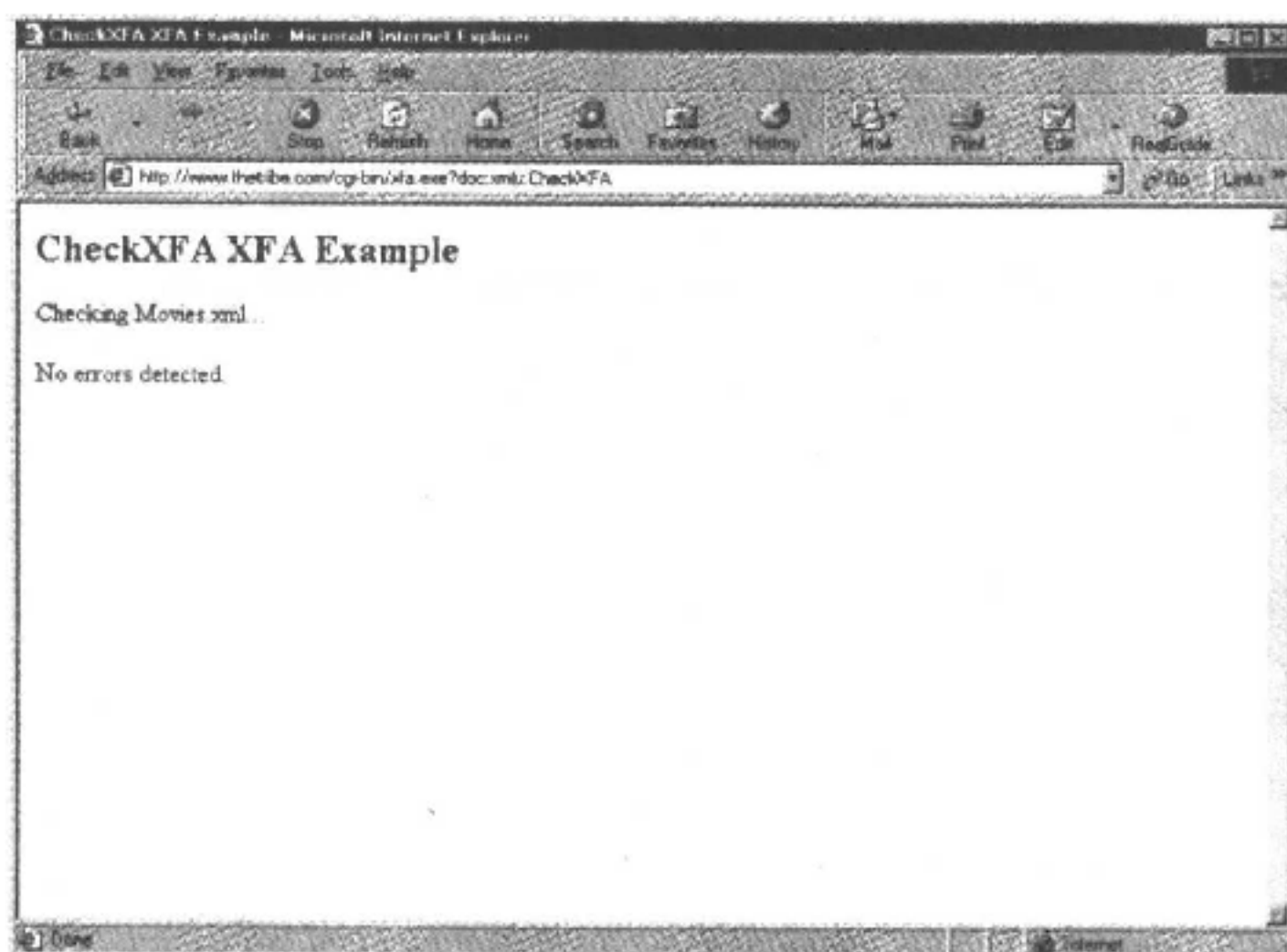


图 28.8 XFA 脚本 CheckXFA 在 Internet Explorer 5.0 中的运行情况,它显示了使用 XFASchema 检查的 XML 文件 movies

我要讲的最后一个话题是:XFA 和 DTD 就是用来将 XFA 脚本转换为严格的 XML 代码的。虽然在 XFA 系统中使用 XFA 的扩展语法非常的好,但你会发现你要使用的 XML 工具和实用程序都需要完全兼容 XML 1.0 格式的 XML 文件。因此,能够将 XFA 文件转换为结构良好的 XML 文件就变得非常的重要。幸运的是,XFA 包含一个库函数,可以毫不费神的完成这项任务。清单 28.10 包含一个脚本,它将 XFA 文件转换为结构良好的 XML 文件。

清单 28.10 将 XFA 代码转换为严格 XML 代码的 XFA 脚本 XFAtXML.xfa

```
<xfa:function XFAtXML>
  <xfa:use library:xml/>
  <xfa:use library:path/>
  <xfa:let errorTree=TreeTag("errors")/>
  <xfa:let text=Read(FilePath("CountMake.xfa"))/>
  <xfa:let tree=Parse(text,errorTree,HTML=true)^TreeBody/>

  <head>
    <title>
      XFAtXML XFA Example
    </title>
  </head>

  <body bgcolor="White">
```

```

<h2>XFAtXML XFA Example</h2>
<xfa:val XFAtXML(tree)/>
<xfa:val Hencode(Unparse(tree,HTML=true))/>
</body>
</xfa:function>

```

XFAtXML 脚本使用脚本代码 CountMake.xfa 作为可以转换为严格 XML 的 XFA 文件的例子。在读取文件并将它解析为一个 XML 树后,脚本调用 XFAtXML() 库函数执行转换。结果树被反解析,编码并显示。这个脚本的结果见于图 28.9。

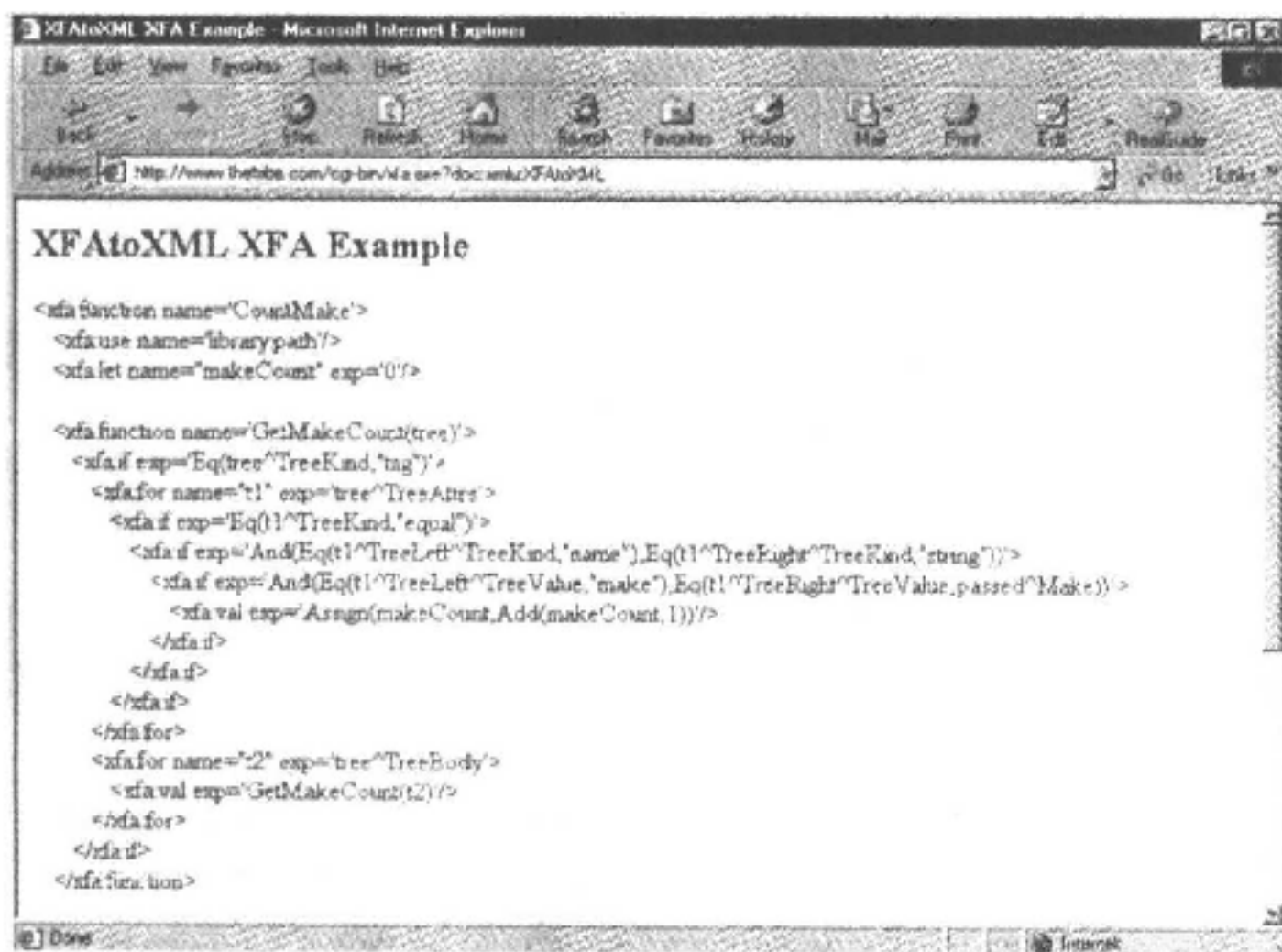


图 28.9 XFA 脚本 XFAtXML 在 Internet Explorer 5.0 中的运行情况,它显示了被转换为严格 XML 代码的脚本 CountMake

28.7 总 结

Web 上的脚本主要使用传统的编程语言,实际上主要的两种 Web 脚本语言显然就是 JavaScript 和 VBScript。由于这些脚本语言来自于传统的编程语言,它们的语法与相应的语言非常相仿。然而,脚本语言的语法与它们所在的网页的语法就非常的不同。依赖于与 Web 内容相同的语法的脚本语言被设计进行使用,这一定非常的有趣,而 XFA 就是这样的一种脚本语言。

XFA 是一种独特的脚本语言,因为它使用 XML 作为它的语法基础。更重要的是,XFA 脚本语言通过 XML 词表实现,而这个词表由描述常用编程结构的元素和属性组成,比如条件句、循环和变量。诚然,本章只是浅显的讲解了 XFA 能作什么。我鼓励你自己作有关 XFA 的试验,看看你能设计出多少种脚本。

第 29 章 使用 DDbE 生成 DTD

要创建正确的 XML 文件,你必须从开发 DTD 或 XMLSchema 开始,以确定 XML 词表的结构。这个 DTD 或 XMLSchema 就是为创建 XML 文件和验证它们的基础。从 DTD 或 XMLSchema 开始,你就给你自己指定了创建正确 XML 的基本原则。然而,并非一定要遵守这个过程。

IBM 的 alpha 工作组开发了一个名为 DDbE(Data Descriptors by Example,数据示例描述)的技术,它允许你从结构良好的 XML 文件或文件集中推断出 DTD。使用 DDbE,你不需要为了验证文件创建 DTD;你可以创建一个文件并用 DDbE 来产生一个 DTD。DDbE 适合于那些还没有深入到 DTD 中的初级 XML 开发人员,XML 应用程序可以从为当前验证目的产生一个 DTD 的功能中获得益处。DDbE 的未来版本可望也支持 XMLSchema。这一章将探讨 DDbE 技术以及它是如何由 XML 文件生成 DTD 的。

29.1 理解 DDbE

DDbE 是一个 Java 库,它被设计用来从 XML 文件和文件集中生成 DTD。DDbE 背后的思想就是,对于初级 XML 开发人员来说,在开始就要掌握 DTD 和 XMLSchema 开发技术是十分困难的,虽然创建一个结构良好的 XML 文件并不很困难。通过允许开发人员能够从 XML 文件中自动的生成 DTD,DDbE 可以使初级 XML 开发人员越过第一步。

DDbE 并不只是为那些没有掌握如何创建 DTD 的初级 XML 开发人员设计的。它还可以被 XML 应用程序使用,这些应用程序需要从文件中生成 DTDs 以确保正在生成文件的正确性。这样的应用程序可以处理一个成熟的 XML 文件并使用 DDbE 从文件中确定 DTD。后来的文件就可以在没有一般的 DTD 开发的情况下自动的用 DTD 进行验证。

注意:就如你可能会怀疑的,DDbE 是一个非常新的技术。在本书编写期间,当前的版本只支持 DTDs 的创建,并不支持 XMLSchema。即使这样,DDbE 作为自动生成 DTD 的基础使用起来也足够的稳定。要了解更多和下载 DDbE,请访问 alpha 工作组的 Web 站点 <http://www.alphaworks.ibm.com>。

由于 DDbE 作为 Java 库实现,它本身并不作为可运行工具发布。它被描述为一个名为 DDbE.jar 的包含一组 Java 类的 JAR 文件。要使用 DDbE 来生成 DTDs,你或者要创建使用 DDbE 库的你自己的 Java 应用程序,或者使用 DDbE 命令行工具来加载 DDbE。在下一节,你将学习到如何使用这个工具。这个工具的源代码将在本章稍后的题为“使用 DDbE 库”中讨论。

警告:DDbE 需要安装 alphaWorks XML4J Java 解析器以正常工作。这个解析器可以从 alpha 工作组的 Web 站点 <http://www.alphaworks.ibm.com> 上免费下载。

29.2 深入 DDbE 命令行工具

DDbE 命令行工具是一个附带在 DDbE 安装集中的简单的 Java 应用程序。这个工具附带源代码,它也令人吃惊的简单。现在,你一定要理解这个工具是被用来从 XML 文件中生成 DTD 的。这一点非常的重要。由于 DDbE 是作为 Java 应用程序实现的,你一定要在一个 Java 实时运行系统中运行它。这样的系统的一个例子就是包含在 Java 开发工具包(JDK)中的系统。DDbE 命令行应用程序名为 DdbECommandLine,它存储在文件 DdbECommandLine.class 中(如果你不是很熟悉 Java,你只要知道可运行的 Java 代码存储在扩展名为.class 的文件中就行了)。

DDbE 命令行工具支持一些命令行选项,它们用来确定 DTD 从文件中生成的方式:

- w——输出 DTD 文件名。
- n——DTD 根元素的名字(如果没有指定,默认就是 DTD 文件名)。
- D——所有文件名前的路径。
- s——包含 XML 源文件列表的文件的名字(用于批处理)。

所有的这些命令行选项使用连字号(-)后面跟选项相关信息进行制定。例如,w 选项需要你提供用来存储 DTD 结果的输出文件的文件名。下面是一个如何在 DDbE 命令行工具中使用这个选项的示例:

```
java DDbECommandLine Books.xml -W Out
```

要记住,java 解释器被用来运行 DDbE 命令行工具。这是一个包含在 JDK 中的标准的解释器。它用来将生成 DTD 的 XML 文件以 Books.xml 存储。输出文件被指定为 Out 并被自动的加上.dtd 的后缀。也就是说,生成的 DTD 将写入到文件 Out.dtd 中。这个例子描述了 DDbE 命令行工具的最简单应用。

注意:Java 开发工具包(JDK)可以从 JavaSoft 的 Web 站点, <http://java.sun.com> 中免费获得。

默认情况下,生成的 DTD 根元素的名字就是 DTD 文件的名字,没有扩展名.dtd。因此,在前面的例子中,DTD 的根元素就名为 Out。显然,这并不总是一个合人心意的结构,这也就是使用 n 命令行选项的原因所在。这个选项允许你指定 DTD 的根(文件)元素的名称。下面是一个使用 n 选项指定生成的 DTD 的根元素的例子:

```
java DDbECommandLine Books.xml -w Out -n booklist
```

DDbE 命令行工具的 D 选项用来提供与 DTD 生成相关的所有文件的路径前缀。这个路径前缀放在输入和输出的文件之前。与通过 DDbE 应用程序自身生成相比,当你想要从一个在不同目录中的文件集合中生成 DTD 时这一点将变得更为有用。下面是一个使用这个选项的例子:

```
java DDbECommandLine Books.xml -w Out -n booklist -D \docs\books
```

s 选项是另一个与处理多个文件生成 DTD 工作相关的选项。这个选项允许你指定一个

包含为生成 DTD 所需要处理的 XML 文件的专门的清单文件。下面是一个例子,相关要处理的文件包含如下:

```
Book1.xml
Book2.xml
Book3.xml
```

假设上面的文件列表被存储在一个名为 Source.txt 的文件中,下面的命令将产生一个基于在 Source.txt 文件中列出的 XML 文件内容的 DTD:

```
java DDbECommandLine Books.xml -w Out -n booklist -s Source.txt
```

以上说明了使用 DDbE 命令行应用程序的基础。还有一些更为高级的选项,你可以通过 DDbE 应用程序来调整实际的 DTD 代码,但在本书编写期间,它们还没有得到很好的证明。

29.3 生成 DTD

在前面一节,我讲解了 DDbE 命令行应用程序的使用方法,但我并没有涉及如何从实际的 XML 文件中生成 DTDs。我们有必要用一些文件来考验一下 DDbE,并分析结果来看看它在从 XML 文件推断 DTD 方面做的有多好。我要处理的第一个文件是第 1 章和第 2 章中的地址簿文件。请回忆一下,这个文件用来存储包含联系信息的地址簿。

下面是使用 DDbE 命令行工具从地址簿文件中生成 DTD 所需的命令:

```
java DDbECommandLine AddressBook.xml -w AddressBook -n addressbook
```

如你所见,地址簿文件存储在名为 AddressBook.xml 的文件中。而 DTD 输出文件在命令中指定为 AddressBook,这使得 DTD 将被存储在文件 AddressBook.dtd 中。生成的地址簿 DTD 如清单 29.1 所示。

清单 29.1 由 DDbE 生成的地址簿 DTD

```
<!-- DTD generated by DDbE. Contact L. Berman, namreb@watson.ibm.com
Addressbook.xml -w AddressBook -n addressbook
-->
<! DOCTYPE addressbook [
<! ELEMENT address (#PCDATA)>
<! ELEMENT addressbook (contact+)>
<! ELEMENT city (#PCDATA)>
<! ELEMENT company (#PCDATA)>
<! ELEMENT contact (name,address+,city,state,zip,phone,email,web,company)>
<! ELEMENT email (#PCDATA)>
<! ELEMENT fax (#PCDATA)>
<! ELEMENT name (#PCDATA)>
<! ELEMENT phone (voice,fax?)>
<! ELEMENT state (#PCDATA)>
<! ELEMENT voice (#PCDATA)>
<! ELEMENT web (#PCDATA)>
<! ELEMENT zip (#PCDATA)>
]>
```

你可能还没有很详细的回忆起原来的地址簿 DTD,但可以说清单 29.1 中所示的这个与原来的 DTD 惊人的相似。当然,地址簿文件结构非常的简单,因此 DDbE 可以非常得体的为它工作。即便如此,DDbE 如何能够探测到 contact 元素的多个出现位置,并在 DTD 中使用加号(+)来说明它可以出现多次(至少一次),这本身就是非常有趣的问题。DDbE 还能够探测 phone 元素的结构并将它分到子元素 voice 和 fax 中。此外,这是一个简单且最为规范的 XML 文件结构,且在自动推断一个结构的能力方面非常的强大。

清单 29.2 包含原来手写的地址簿 DTD,它有助于你理解 DDbE 的正确性。

清单 29.2 原来的手写地址簿 DTD

```
<! ELEMENT addressbook(contact)+>
<! ELEMENT contact (name,address+,city,state,zip,phone,email,web,company)>
<! ELEMENT name (#PCDATA)>
<! ELEMENT address (#PCDATA)>
<! ELEMENT city (#PCDATA)>
<! ELEMENT state (#PCDATA)>
<! ELEMENT zip (#PCDATA)>
<! ELEMENT phone (voice,fax?)>
<! ELEMENT voice (#PCDATA)>
<! ELEMENT fax (#PCDATA)>
<! ELEMENT email (#PCDATA)>
<! ELEMENT web (#PCDATA)>
<! ELEMENT company (#PCDATA)>
```

这个 DTD 暴露了 DDbE 的一些弱点,也就是内容模型。DDbE 不能推断出 address 元素中的加号(+)或 fax 元素中的问号(?)。然而,地址簿文件本身就对此负有责任。这个文件如清单 29.3 所示。

清单 29.3 用来通过 DDbE 生成 DTD 的 XML 地址簿文件

```
<? xml version="1.0"? >
<addressbook>
  <!-- This is my good friend Frank. -->
  <contact>
    <name>Frank Rizzo</name>
    <address>1212 W 304th Street</address>
    <city>New York</city>
    <state>New York</state>
    <zip>10011</zip>
    <phone>
      <voice>212-555-1212</voice>
      <fax>212-555-1213</fax>
```



```

    </phone>
    <email>frizzo@fruity.com</email>
    <web>http://www.fruity.com/rizzo</web>
    <company>Frank&apos;s Ratchet Service</company>
  </contact>

  <! -- This is my old college roommate Sol. -->
  <contact>
    <name>Sol Rosenberg</name>
    <address>1162 E 412th Street</address>
    <city>New York</city>
    <state>New York</state>
    <zip>10011</zip>
    <phone>
      <voice>212-555-1818</voice>
      <fax>212-555-1819</fax>
    </phone>
    <email>srosenberg@fruity.com</email>
    <web>http://www.fruity.com/rosenberg</web>
    <company>Rosenberg&apos;s Shoes &amp; Glasses</company>
  </contact>
</addressbook>

```

这个地址簿文件中的两个 `contact` 元素包含一个独立的 `address` 子元素,这就使得 DDbE 不可能在用于 `contact` 的内容模型中确定是否允许有多于一个的 `address` 元素存在。同样, `fax` 子元素在两个 `phone` 元素中都有出现,这就使得 DDbE 无法知道 `fax` 元素是可选的。这个例子阐明了给 DDbE 提供已填充 XML 文件的丰富集合的重要性。大集合的数据更可能揭示出指定文件类的内容模型的限制。

清单 29.4 显示了修正的 XML 地址簿文件,它给 DDbE 提供了更多的信息。

清单 29.4 用来通过 DDbE 生成 DTD 的更为准确的 XML 地址簿文件

```

<? xml version="1.0"? >
<addressbook>
  <! -- This is my good friend Frank. -->
  <contact>
    <name>Frank Rizzo</name>
    <address>1212 W 304th Street</address>
    <city>New York</city>
    <state>New York</state>
    <zip>10011</zip>
    <phone>
      <voice>212-555-1212</voice>
      <fax>212-555-1213</fax>
    </phone>
    <email>frizzo@fruity.com</email>
    <web>http://www.fruity.com/rizzo</web>
    <company>Frank&apos;s Ratchet Service</company>
  </contact>

  <! -- This is my old college roommate Sol. -->

```

```

<contact>
  <name>Sol Rosenberg</name>
  <address>1162 E 412th Street</address>
  <city>New York</city>
  <state>New York</state>
  <zip>10011</zip>
  <phone>
    <voice>212-555-1818</voice>
    <fax>212-555-1819</fax>
  </phone>
  <email>srosenberg@fruity.com</email>
  <web>http://www.fruity.com/rosenberg</web>
  <company>Rosenberg&apos;s Shoes &amp; Glasses</company>
</contact>

<! -- This is my landlord. -->
<contact>
  <name>Brett Weir</name>
  <address>9214 W 442nd Street</address>
  <address>Apt. 7A3</address>
  <city>New York</city>
  <state>New York</state>
  <zip>10012</zip>
  <phone>
    <voice>212-555-1819</voice>
  </phone>
  <email>bweir@fruity.com</email>
  <web>http://www.fruity.com/weir</web>
  <company>Brett's Rentals</company>
</contact>
</addressbook>

```

这个文件包含一个附加的 `contact` 元素，它声明了内容模型在 `contact` 和 `phone` 元素中的绑定。更重要的是 `contact` 的 `address` 子元素有重复，而 `phone` 中去掉了 `fax` 子元素。用这个文件再次运行 DDbE 得到的 DTD 如清单 29.5 所示。

清单 29.5 DDbE 生成的更为正确的地址簿 DTD

```

<! -- DTD generated by DDbE. Contact L. Berman, namreb@watson.ibm.com
Addressbook.xml -w AddressBook -n addressbook
-->
<! DOCTYPE addressbook [
  <! ELEMENT address (#PCDATA)>
  <! ELEMENT addressbook (contact+)>
  <! ELEMENT city (#PCDATA)>
  <! ELEMENT company (#PCDATA)>
  <! ELEMENT contact (name,address+,city,state,zip,phone,email,web,company)>
  <! ELEMENT email (#PCDATA)>
  <! ELEMENT fax (#PCDATA)>
  <! ELEMENT name (#PCDATA)>
  <! ELEMENT phone (voice,fax?)>
  <! ELEMENT state (#PCDATA)>

```

```
<! ELEMENT voice (#PCDATA)>
<! ELEMENT web (#PCDATA)>
<! ELEMENT zip (#PCDATA)>
]>
```

这个 DTD 比前面的一个更为正确。实际上,这个 DTD 正好匹配原来我手工编制的 DTD 的结构。无疑,传入到 DDbE 中的 XML 文件和文件集的数据的变化越多,工具在生成 DTD 时就会做的越好。

虽然 DDbE 结束了有关地址簿 DTD 的繁重工作,但地址簿文件是非常简单且最为规范的 XML 文件。第 5 章中的运动训练文件就更为复杂,它确实是一个测试 DDbE 的 DTD 推断能力的更好的测试文件。下面是使用 DDbE 从运动训练文件中生成 DTD 的命令:

```
java DDbECommandLine Train.xml -w Train -n trainlog
```

这个命令使用文件 Train.xml 作为生成 DTD 的基础。根元素被设置为 trainlog,这非常重要,因为如果你不使用 n 选项,DDbE 就将使用 DTD 文件名作为根元素的名称。清单 29.6 包含了通过 DDbE 生成的 DTD 代码。

清单 29.6 通过 DDbE 生成的“运动训练”DTD

```
<!-- DTD generated by DDbE. Contact L. Berman, namreb@watson.ibm.com
When designing data descriptors, it is good practice to constrain attributes
as much as possible. Possible declarations for attributes (all currently
declared as CDATA) are indicated just prior to the attribute declarations.
Key:   E ==>> enumeration consistent,
       I ==>> ID possible,
       R(S) ==>> IDREF(S) possible,
       N(S) ==>> NMTOKEN possible,
       S implies plural declaration is required.

Since an ID may always be an ENUMERATION, and NAMES are NMTOKENS, only the most
constraining possibilities are mentioned. Note: the declarations mentioned
below may not be simultaneously consistent.

Train.xml -w Train -n trainlog
-->
<! DOCTYPE trainlog [
<! ELEMENT comments (#PCDATA)>
<! ELEMENT distance (#PCDATA)>
<! ELEMENT duration (#PCDATA)>
<! ELEMENT location (#PCDATA)>
<! ELEMENT session (duration,distance,location,comments)>
<!--
    session.type: E
    session.heartrate: N
-->
<! ATTLIST session
    type CDATA #IMPLIED
    date CDATA #IMPLIED
    heartrate CDATA #IMPLIED
>
```

```
<! ELEMENT trainlog (session+) >
]>
```

这个 DTD 与地址簿 DTD 看起来有一些不同, 因为 DDbE 在确定一些属性的类型方面遇到了一些麻烦。例如, DDbE 认识到 session 元素的 type 属性可以作为一个枚举类型, 而它并不能推断出可能的枚举值集合。因此, DDbE 不能在 DTD 中设置属性为枚举类型, 而可以通过构造属性为 CDATA 类型来简单的处理这个问题。然而, 注释将被加入, 以告诉你这个属性需要作为枚举类型。这确实是 DDbE 在这些迂回解决办法中所能作的最好的一种了。

DDbE 也承认属性 heartrate 在文件全局只能赋予整型值, 因此它加入了一个推荐注释: 类型被定义为 NMTOKEN。这种信息对于在 DDbE 生成 DTD 后调整 DTD 方面非常的有用。

29.4 使用 DDbE 库

我在本章前面提到了 DDbE 实际是作为 Java 库实现的。DDbE 命令行应用程序就是建于 DDbEJava 库之上的简单应用程序。知道了这一点, 你就能够不太麻烦的将 DDbE 结合进你自己的 Java applet 和应用程序中。清单 29.7 包含 DDbEJava 应用程序的源代码, 它将告诉你使用 DDbE 库进行工作是多么的简单。

清单 29.7 DDbE 命令行应用程序的 Java 源代码

```
import com.ibm.DdbE.Models.DDModel;
import com.ibm.DdbE.InterfacesEvents.*;
import com.ibm.DdbE.Utilities.Parameter;
import com.ibm.DdbE.utilities.FileStringRW;
import com.ibm.DdbE.Utilities.ExampleGenerator;

public class DdbECommandLine {
    public static void main(String args[]) {
        DDModel      dtdEngine = new DDModel(args);
        FileStringRW  xmlSamples = new FileStringRW();
        Parameter     parameters = dtdEngine.getParameter();

        if(parameters.getInputFileSpecified()) {
            xmlSamples.setFile(parameters.getSources(),
                parameters.getInputDirectory());

            dtdEngine.setExampleGenerator(new ExampleGenerator(xmlSamples));
            dtdEngine.getTagArray();
            dtdEngine.saveDTD();
        }
        else {
            System.out.println("DdbECommandLine: parameter error");
        }
    }
}
```

代码的前面几行导入适当的包含应用程序所需要类的 DDbE 包。这些包和类都包含在

DDbE 附带的 JAR 文件 DDbE.jar 中。这个 JAR 文件本身就是 DDbE 库。

DDbE 引擎的基础就是 DDMoel 类,它在 DDbECommandLine 应用程序中首先被创建。这个类接收应用程序的参数并理解在本章前面提到的 DDbE 选项。然后,输入文件就基于 DDbE 的这些参数确定。ExampleGenerator 对象就被创建并使用 setExampleGenerator() 方法传入到 DTD 引擎中。最后 DTD 被生成并被保存。

你可以使用简单的代码在你自己的 Java applet 和应用程序中完成从文件生成 DTD 的功能。不幸的是,在本书编写期间,DDbE 库还基本没有正式文件。希望将来的版本将包含更多的文件并有一些例子,这将可以展示出使用 DDbE 技术的更多方法。

29.5 总 结

DDbE(Data Descriptors by Example,数据示例描述)是 IBM 的 alpha 工作组的技术,它可以用来从 XML 文件和文件集中推断出 DTD 和 XMLSchema。这种从文件中生成 DTD 的方法违反了正规的 XML 开发流程——首先设计一个 DTD,然后创建依附于它的文件。虽然这看起来是非传统的,但 DDbE 确实是非常的功能强大,因为它允许初级 XML 开发人员在没有深厚的 DTD 设计基础的情况下,迅速的创建 DTD。通过使用一些应用程序也将发现 DDbE 非常有用,因为有这样的情况,它从动态生成 DTD 中受益(例如帮助验证那些传送结构信息的 XML 文件)。计划中的 DDbE 的未来版本将支持 XMLSchema 文件。我还希望 DDbE 库的内部工作文件化,这样 Java 开发人员可以从很低的层次使用 DDbE。即使是对于初级版本,DDbE 仍然是一项非常值得一看的有趣技术。

第30章 了解 IBM 的 XML 工具集

前面一章讲解了 DDbE(Data Descriptors by Example,数据示例描述)技术,它由 IBM 的 alpha 工作组创建的。Alpha 工作组的主要工作就是 XML 和 Java 的开发,它实际上有了大量的 XML 工具来执行多种不同的任务。集合起来,这些工具就构成了一个功能强大的工具集,它可以通过令人惊讶的方式来访问和管理 XML 文件。

本章将讲述由 IBM 的 alpha 工作组制作的部分非常有趣的工具。将这些工具加入到你的 XML 工具集中将非常有益,因为它们提供了一些有趣的并且强大的功能。在这里,你可以学习到如何使用 alpha 工作组的工具来搜索、浏览、转换、区分和归并 XML 文件。

30.1 工具箱一瞥

IBM 的 alpha 工作组从开始就致力于 XML 的工作。IBM 对 XML 技术的支持承诺从 alpha 工作组开发的工具和实用程序的数量这方面就已经非常明显了。下面是 IBM 的 alpha 工作组开发的部分非常有趣的工具:

- Xplorer
- XML 浏览器
- XML 转换生成器
- XML 区分和归并工具

所有的这些工具都用 Java 开发,这一点并不奇怪,因为 Java 被普遍认为是 XML 软件开发的首选编程语言。

Xplorer 是一个允许你搜索和验证 XML 文件的应用程序。它绑定在 XML 浏览器中,这个浏览器提供对 XML 文件的可视化树状浏览以及标准的文本浏览。

XML Translator Generator(转换生成器)是一个将 XML 文件从一种词表(标记集)转换为另一种词表的工具。你也能够使用 XML 转换生成器从 HTML 文件中提取数据并放入到 XML 文件中。

XML Diff and Merge(区分和归并工具)用来对 XML 文件进行相互比较以检测任何区别。这个工具对于那些经历了多种方式编辑的文件来说非常有用。它允许你比较所有修改的文件并把它们归并成一个单独的文件。

本章的其余部分将探讨这些工具并向你展示如何使用它们来访问和管理 XML 文件。所有的这些工具都通过 Java 实现并需要 Java 运行解释器,比如附带在 Java 开发工具包(JDK)中的解释器。这些工具可以从 alpha 工作组的 Web 站点 <http://www.alphaworks.ibm.com> 上免费下载。JDK 可以从 JavaSoft 的 Web 站点 <http://java.sun.com> 上免费下载。

30.2 使用 Xplorer 和 XML 浏览器

Xplorer 是一个 Java 应用程序,它提供一个搜索和验证 XML 文件的图形化用户接口。如果你在管理多种 XML 文件,你可以使用 Xplorer 来搜索它们以获得指定的信息,比如元素的内容和属性值。图 30.1 显示了启动后的 Xplorer 应用程序。

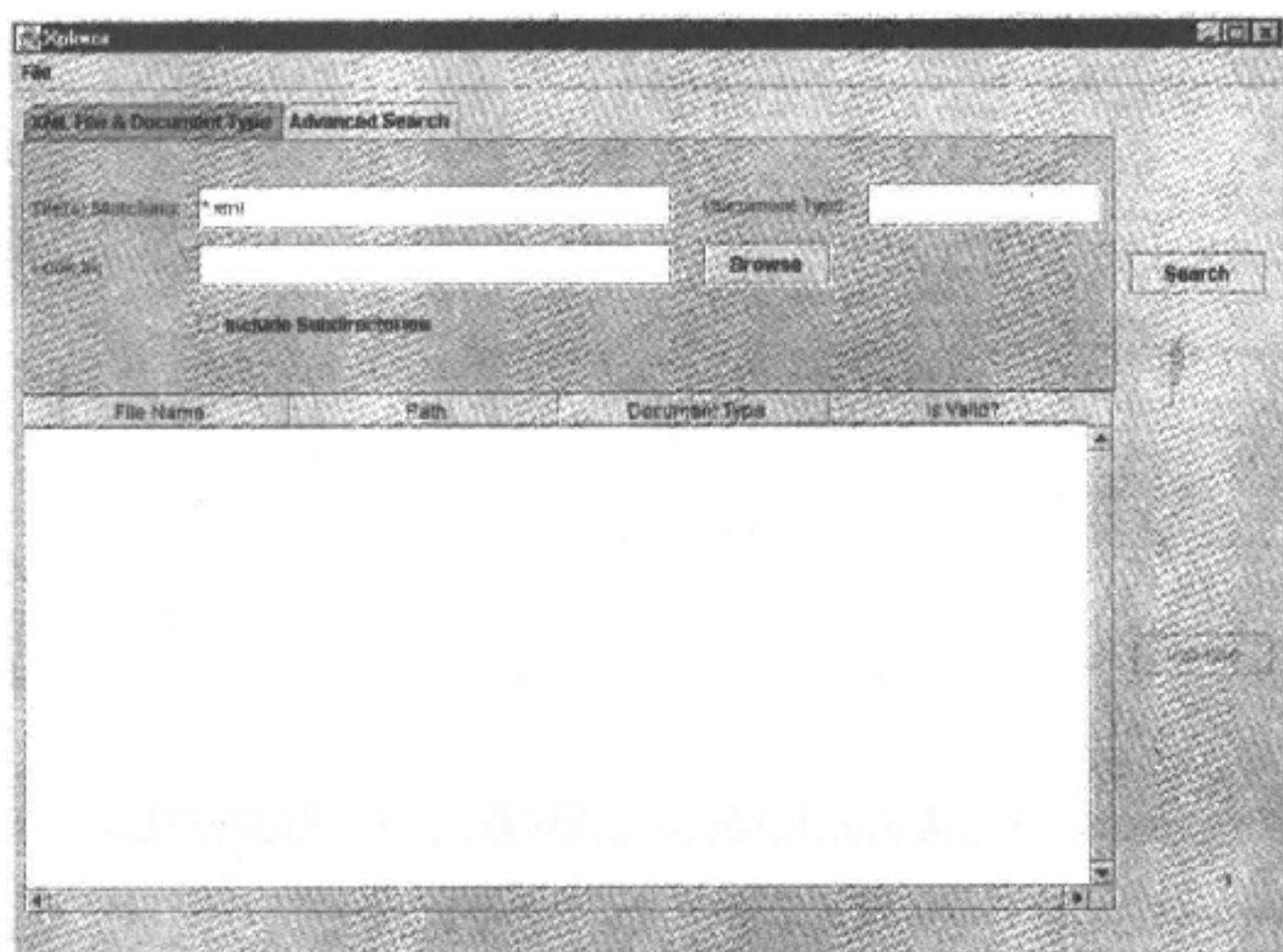


图 30.1 启动后的 Xplorer 应用程序

要搜索 XML 文件,就要提供一个开始搜索的路径并点击 Search(搜索)按钮。图 30.2 显示了在我的硬盘上的一个目录中搜索所有扩展名为.xml 的 XML 文件的结果。

你可能认为不需要特殊的 XML 工具就可以执行这类文件搜索工作。这一点你是对的。当你基于诸如文件类型(根元素)这样的 XML 特殊标准开始搜索时,Xplorer 的强大之处才显示了出来。图 30.3 显示了执行对类型为 movies 的文件的搜索情况。

如你所见,结果文件集在你限制搜索指定文件类型后变得更小。搜索和获取一个文件集后,你可以通过选择它们并点击 Validate(检验)按钮来检验任何一个或全部的文件。图 30.4 显示了检验电影收藏文件的结果。

文件列表中的“Is Valid?”列指出哪些文件有效而哪些无效。只有一个电影收藏文件是有效的。这是因为另外两个文件没有包含文件类型声明,而这是实现文件有效所需要的。

你也许已经注意到了 Xplorer 用户接口分为两个窗格:XML File & Document Type 窗格和 Advanced Search 窗格。Advanced Search 窗格允许你执行基于特定 XML 文件内容——诸如元素和属性值——的更高级的搜索。图 30.5 显示了 Xplorer 的 Advanced Search 窗格。

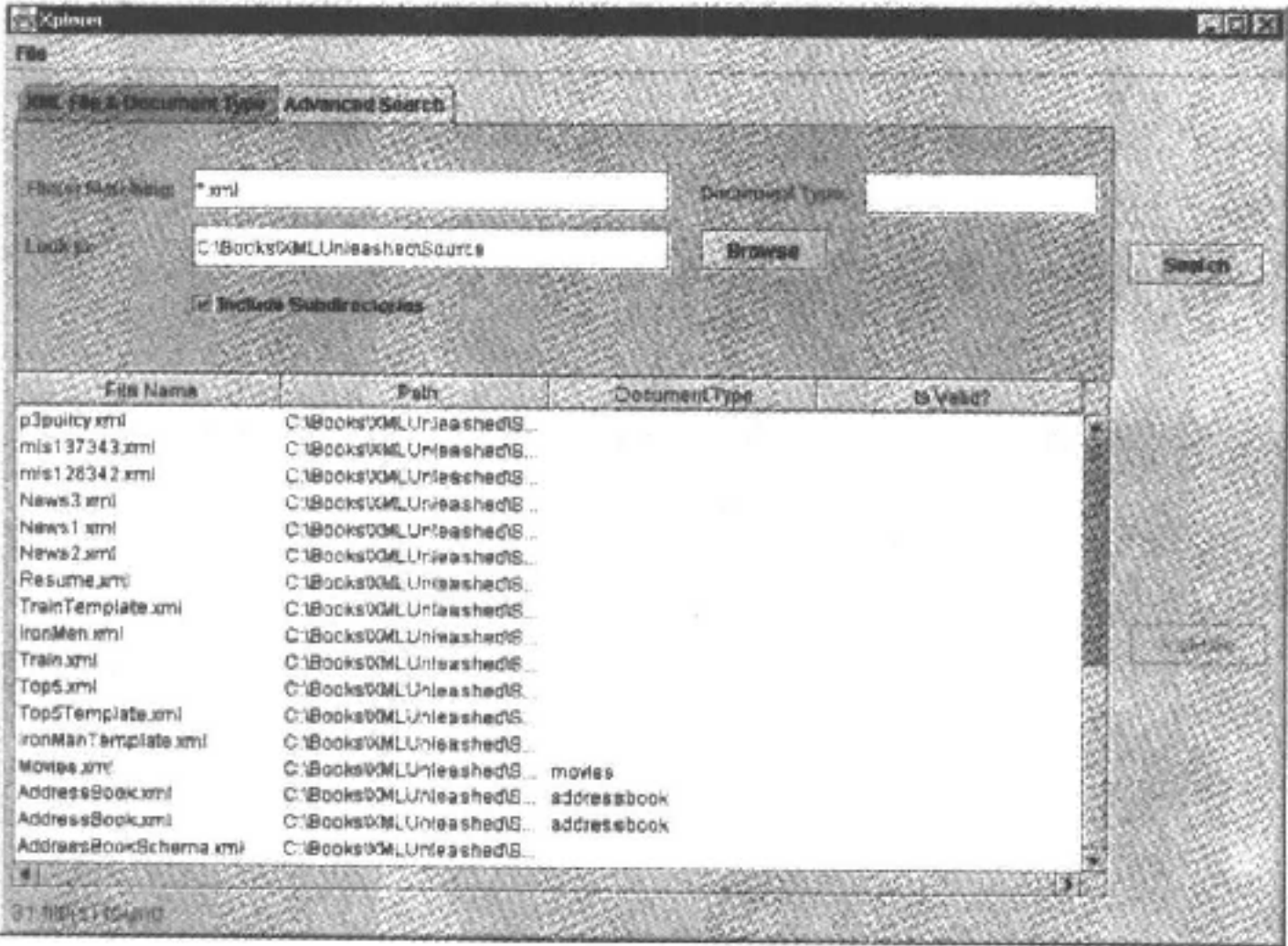


图 30.2 在我的硬盘上的一个目录中搜索所有的 XML 文件 (*.xml)的结果

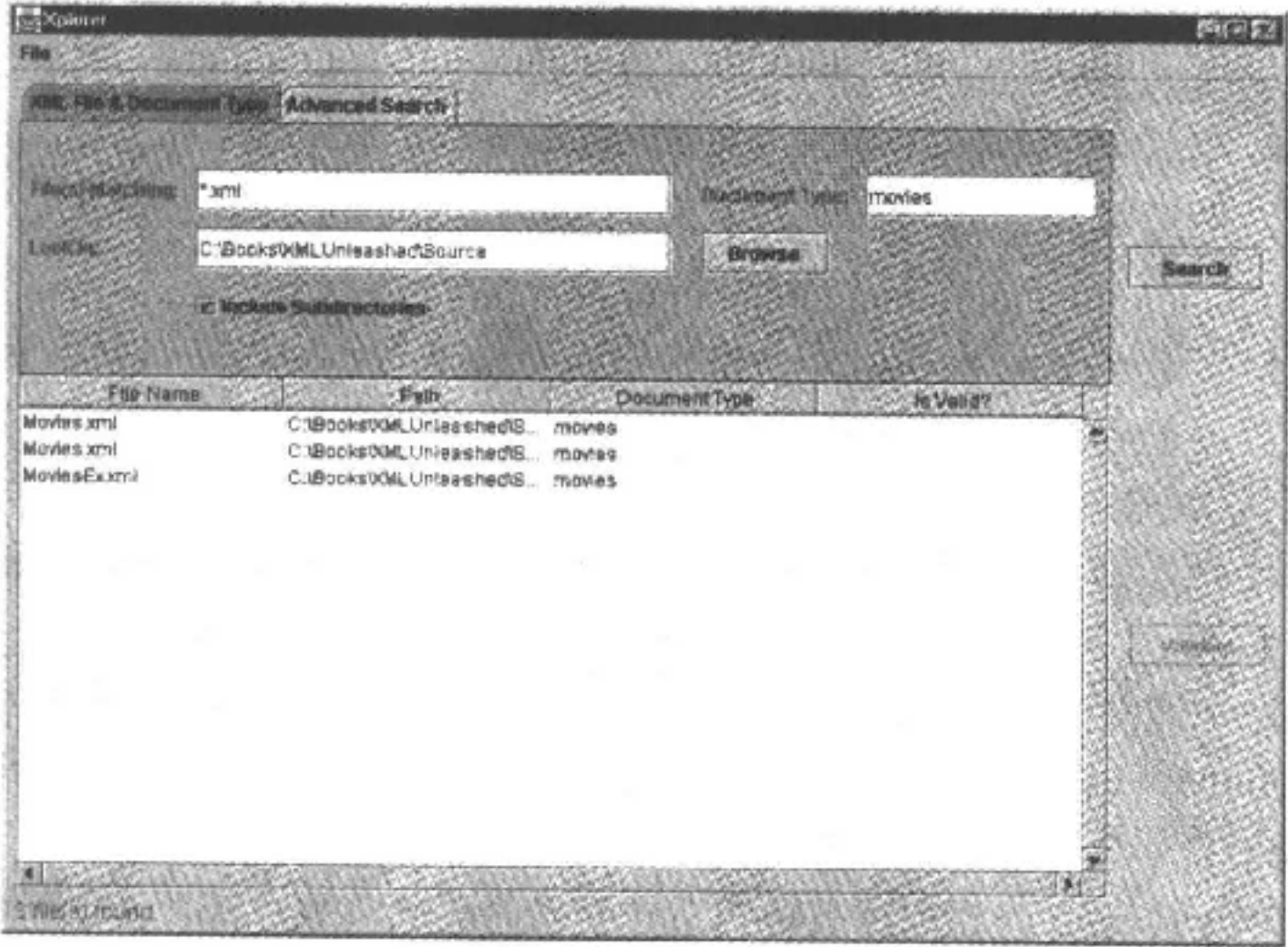


图 30.3 限定为 movies 类型文件的 XML 文件搜索

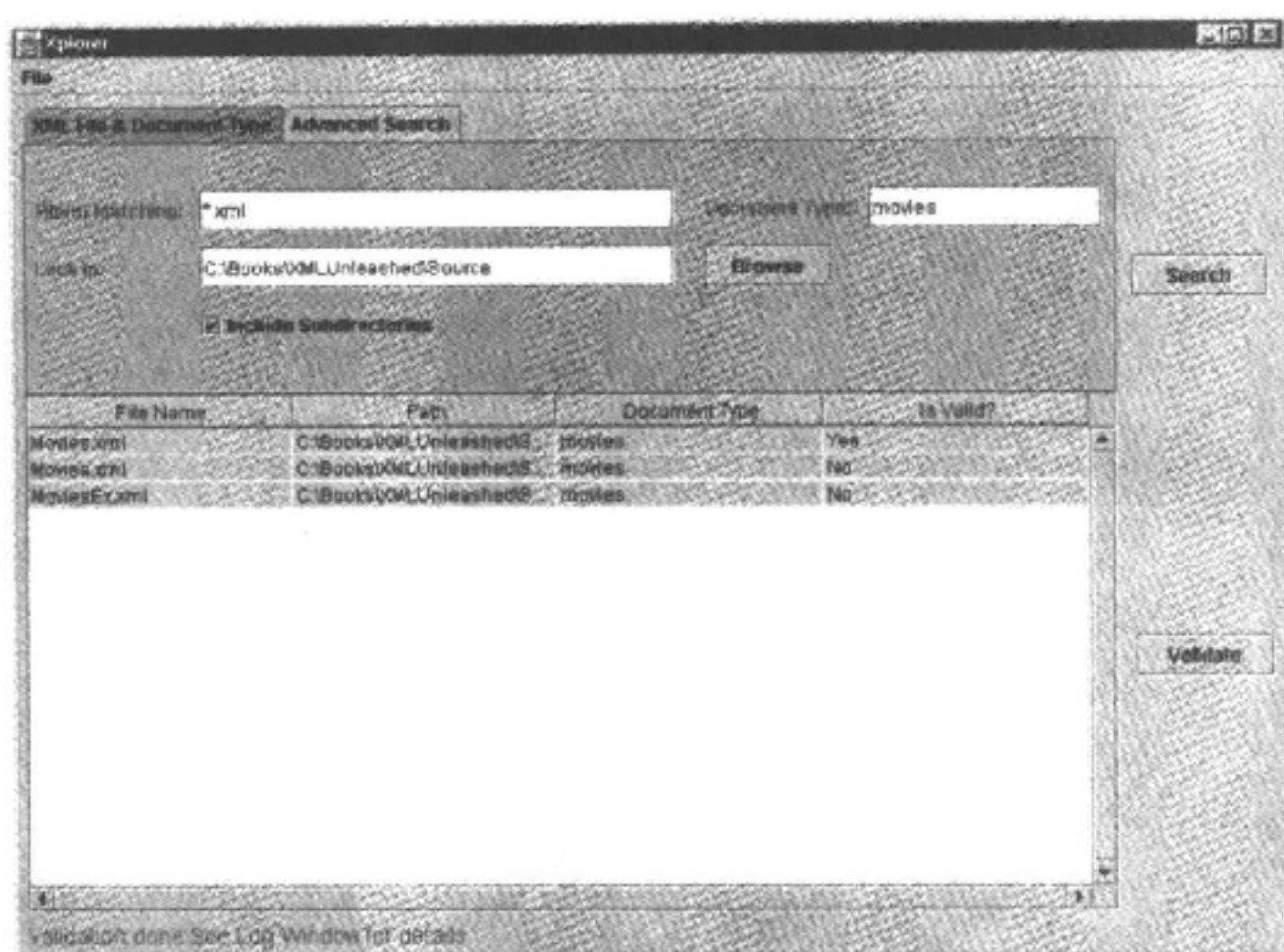


图 30.4 Xplorer 允许你自动化验证 XML 文件的过程

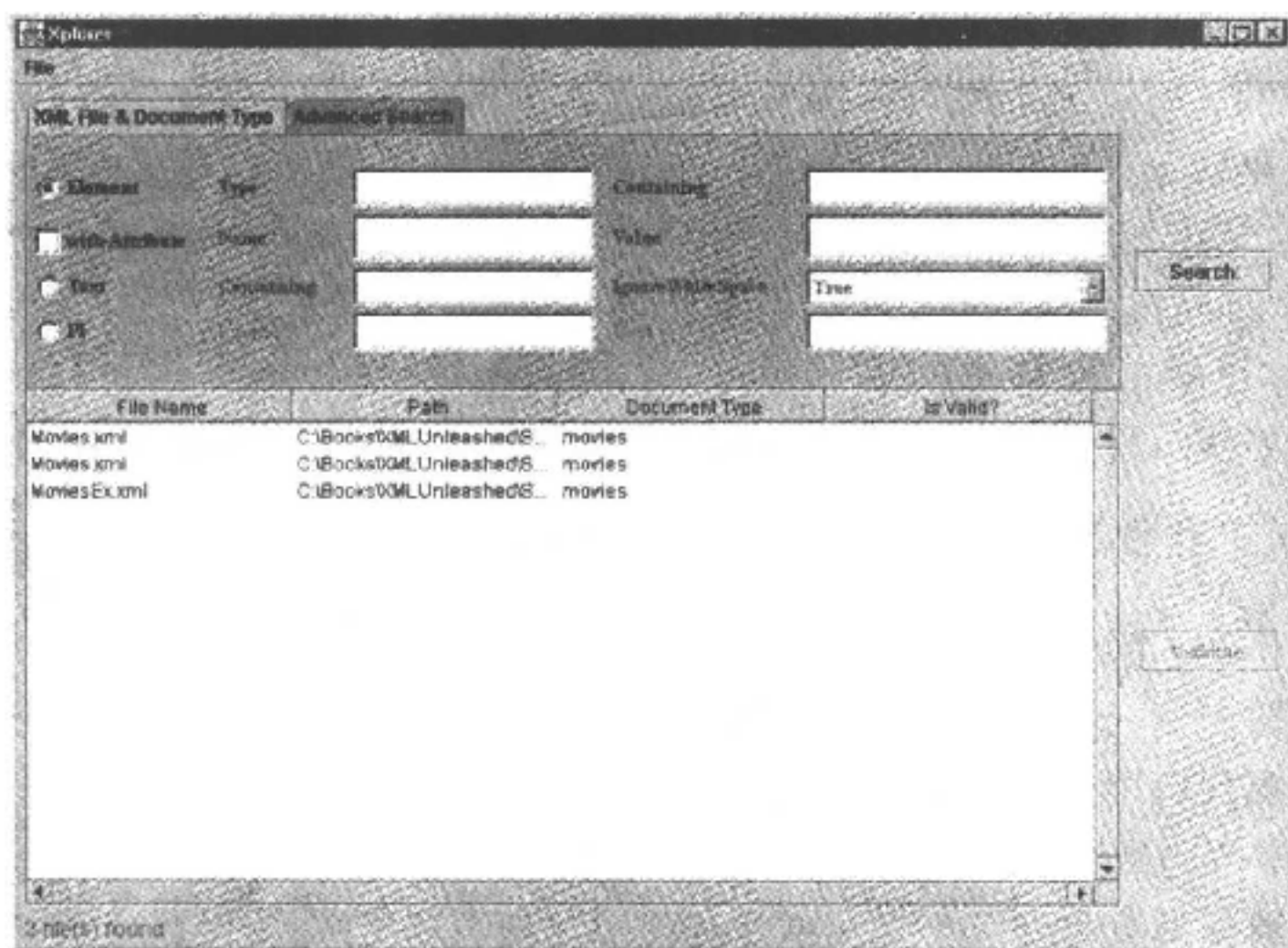


图 30.5 Xplorer 应用程序的 Advanced Search 窗格

如你所见,这个窗格提供了大量的对用来搜索 XML 文件的准则的控制。更重要的是,你可以搜索指定的元素、属性、文本和处理指令。为了展示这种类型搜索的有用之处,图30.6显示了对电影收藏文件的搜索结果——一个 year 属性包含值 1987 的名为 movie 的元素。换

句话说,这个搜索寻找包含 1987 年发行的电影的电影收藏文件。

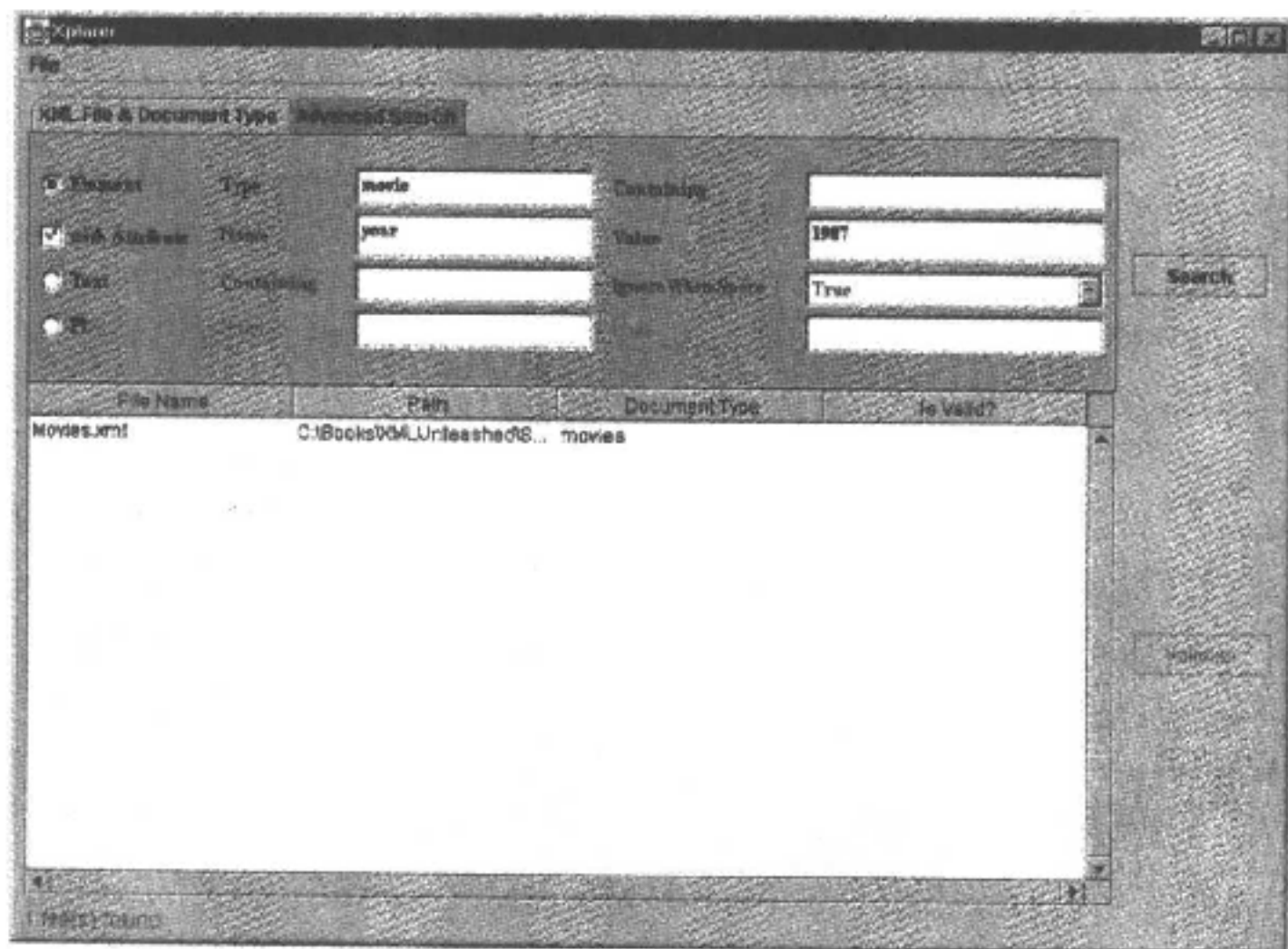


图 30.6 year 属性包含值为 1987 的名为 movie 的元素的搜索结果

虽然这是一个非常简单的例子,但它揭示了基于内容数据的能力,而本质上 XML 文件是由它们组成的。即使 Xplorer 是一个独一无二的应用程序,这种基于文件内容的指定类型的搜索的方法是对未来的 Web 的一种暗示。我非常希望支持与 XML 相同的基于内容搜索方法的 HTML 标准搜索引擎能够在 Web 上变得更加流行。

我在本章前面提到 Xplorer 应用程序绑定在 XML 浏览器中以允许你图形化的浏览 XML 文件。你可以在 XML 浏览器中通过双击 Xplorer 的搜索结果文件清单中的文件来浏览一个 XML 文件。图 30.7 显示了在 XML 浏览器中打开的电影收藏文件 Movies.xml。

注意:只有有效的 XML 文件可以在 XML 浏览器中浏览。

这个图显示出 XML 浏览器由四个窗格组成:

- XML 树浏览——文件的图形树浏览
- XML 源代码浏览——文件的文本浏览
- DTD 源代码浏览——文件的 DTD 的文本浏览
- XML 属性浏览——与所选元素相关的属性列表浏览

综合起来,这些浏览提供了一种访问 XML 文件结构和内容的有趣途径。也许最重要的浏览就是树浏览,因为它揭示了 XML 文件的层次化本质。XML 和 DTD 源代码浏览就是对文件和它自身 DTD 的简单的文本浏览。DTD 源代码浏览的存在应该是 XML 浏览器为什么只能浏览有效 XML 文件的原因之一。XML 属性浏览用来列出当前在树浏览中被选择元素的相关属性。

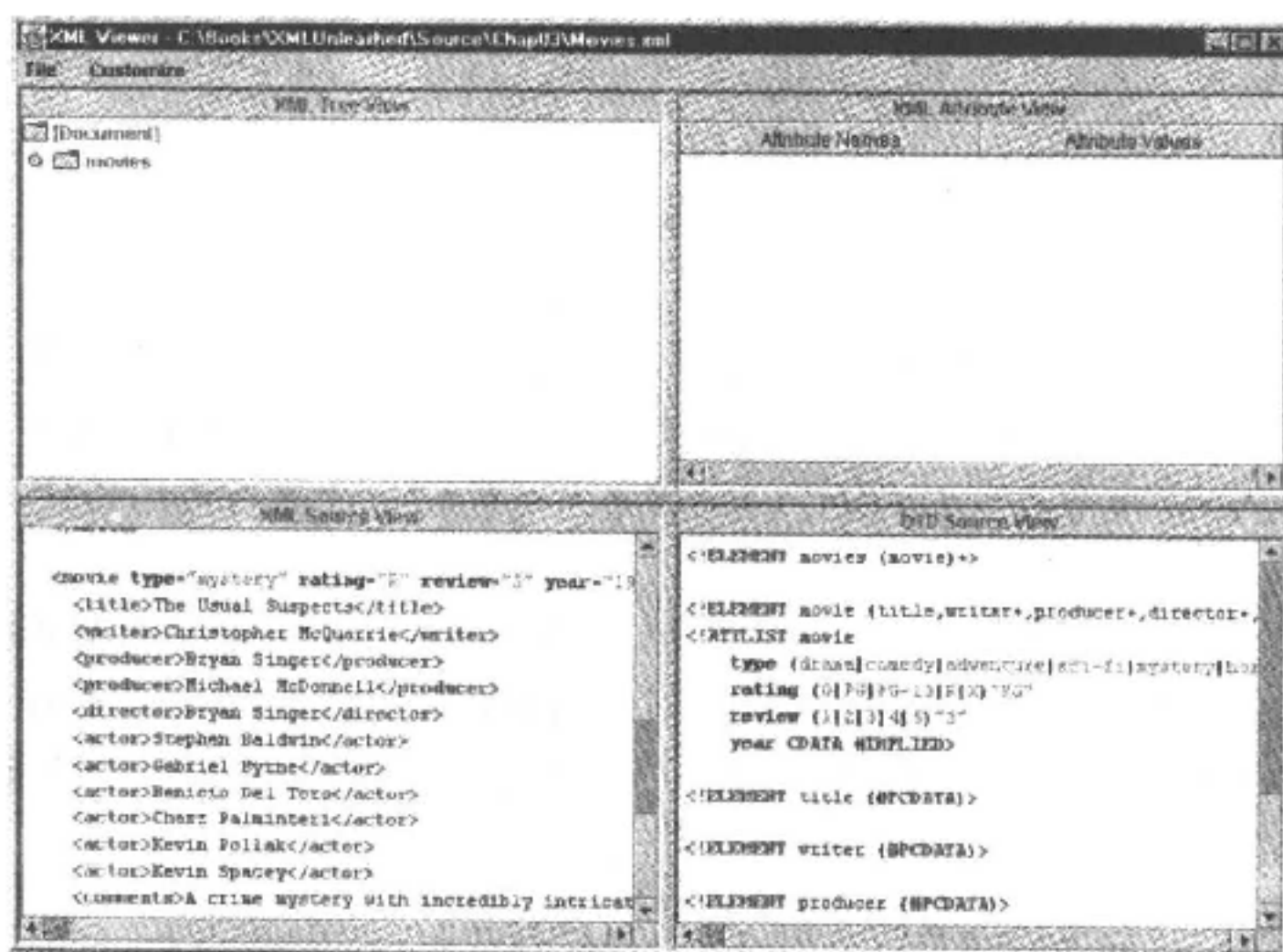


图 30.7 在 XML 浏览器应用程序中浏览 XML 电影收藏文件

在 XML 浏览器中所有这些浏览的真正强大之处在于，它们可以相互同步，也就是说树浏览中选择的元素在源代码中被高亮显示。任何元素的属性在属性浏览中列出。图 30.8 显示了在 XML 浏览器中的电影收藏文件在一个 movie 元素被选择时的显示情况。

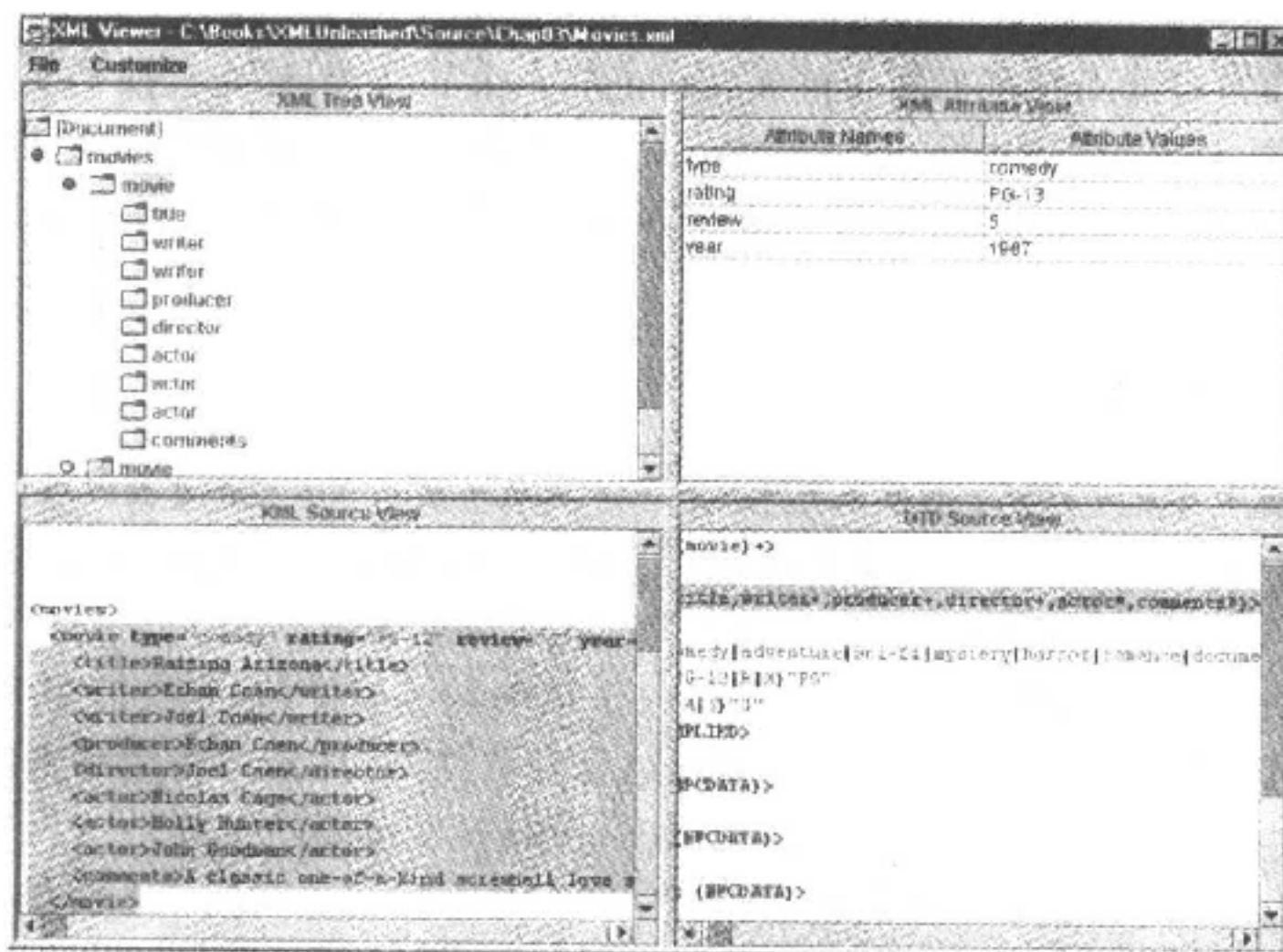


图 30.8 在 XML 浏览器的树浏览中选择一个元素使得在其他窗格中的相关信息被高亮显示

XML 源代码视图高亮显示了元素的 XML 文件数据,同时 DTD 源代码浏览显示了 movie 元素的元素声明。最后,属性浏览列出了与元素相关的属性的列表以及它们相应的值。

30.3 使用 XML 转换生成器

XML 转换生成器是一个将 XML 文件从一个词表转换到其他词表的工具。你可以将整个集合的 XML 文件的数据转换为不同的词表,并且这项工作是完全自动化的。例如你可以使用 XML 转换生成器将以专有 XML 词表存储的文件转换为流行应用程序支持的标准词表。

XML 转换生成器作为过滤器可从 HTML 文件中提取数据。这在你希望从网页中获取数据以进行处理的情况下非常有用。要谨记网页是面向显示的,这就使它们难于处理。从网页中提取数据并将它们以诸如 XML 这样的面向内容的数据格式存储,这在要处理数据的时候非常有价值。

使用 XML 转换生成器转换 XML 文件需要两个基本步骤:

1. 生成一个基于 XML 文件模板的转换器。
2. 使用转换器转换包含实际数据的 XML 文件。

第一步中所指的文件模板也就是适应你转换所涉及的两个词表的例子文件。例如,如果你尝试将数据从文件类型 A 转换为文件类型 B,你手工的创建每种类型的例子程序作为模板。很重要的一点是这些模板文件中的数据要严格的匹配,这样 XML 转换生成器可以适当的确定每种文件类型中元素和属性的相互关系。虽然两个模板文件中的数据必须严格匹配,但文件的结构显然可以不同。

XML 转换生成器工具基于两个模板文件生成一个转换器。这个转换器实际上是一个包含每个文件中的元素和属性的详细映射关系的文本文件。你只需要创建这个转换器文件一次。从此以后,你就可以使用 XML 转换生成器和这个转换器文件将文件从类型 A 转换为类型 B。

30.3.1 转换 XML 文件

XML 转换生成器的最为明显的应用就是将 XML 文件从一种词表(DTD)转换为另一个。例如,考虑一下在本书前面的章节提到的 XML 运动训练例子文件。如果你还记得,运动训练文件包含有关训练活动的信息,比如持续时间和活动举例。

为了说明这个例子,让我们假设你买了一个运动训练软件包,它支持名为 Iron Man 的假想标准 XML 词表。为了将你的训练记录结合到软件中,你希望将运动训练文件转换为 Iron Man 格式。XML 转换生成器正是你完成这种转换所需要的工具。

转换运动训练文件的第一步是创建每种 XML 文件格式的模板。下面是你用来描述训练活动的运动训练文件格式的模板:

```
<? xml version="1.0"? >
```

```
<session type="running" date="0/0/0" heartrate="120">
```



```

<duration>00:00:00</duration>
<distance>0.0</distance>
<location>Somewhere</location>
<comments>I ran. </comments>
</session>

```

这个单独的模板将被用来作为转换所有的运动训练文件的基础,因此它必须是全面的。换句话说,你必须确定包含了所有你希望转换的元素和属性;任何遗漏的东西都不能被转换。源模板完成了,下一步就是创建新的 XML 文件格式的目标模板。下面是 Iron Man 文件模板:

```

<? xml version="1.0"? >
<training date="0/0/0" location="Somewhere">
  <sport>running</sport>
  <duration>00:00:00</duration>
  <distance>0.0</distance>
  <avghr>120</avghr>
  <notes>I ran. </notes>
</training>

```

这个模板基于假想的 Iron Man XML 词表。在现实世界情况下,我们有必要学习目标 DTD 并确信元素和属性结构正确。最为重要的是要知道这个模板是一个数据,它严格匹配源模板中的数据。XML 模板生成器使用这个数据作为匹配元素和属性的方法以创建转换文件。

注意:如果你好奇,我告诉你 avghr 元素存储训练过程中的平均心跳速率,这对任何认真的耐力运动员都是有价值的信息。Avghr 的值标识每分钟的心跳次数,在整个训练过程中取它的平均值。一个人的平均心跳速率为每分钟 60 次左右,而最大心跳速率为每分钟 200 次左右,这取决于你的年龄。你可以通过在一段时间内持续计数心跳次数或是使用心率监听器来测量你的心跳速率。

说到转换文件,它是通过在 Java 解释器中运行 XML 转换生成器的 Map 应用创建的。Map 应用位于 com.ibm.xtrans 包中,因此它必须使用如下的命令行来运行:

```
java com.ibm.xtrans.Map TrainTemplate.xml IronManTemplate.xml Train.cnv
```

注意:使用 Java 解释器运行 XML 转换生成器应用的例子基于附带在 Java 开发工具包(JDK)中的标准 Java 解释器。

Map 应用的三个参数分别是源模板、目标模板和转换器文件。当这个命令运行的时候,Train.cnv 转换器文件被创建。这个转换器文件如清单 30.1 所示。

清单 30.1 由 XML 转换生成器生成的转换文件 Train.cnv

```

[(0, #element, session), (0, #element, duration), (0, #text, #string)]
[(0, 0, #element, training), (1, 0, #element, duration), (0, 0, #text, #string)]

```

```
[ (0, # element, session), (2, # attr, heartrate) ]
[ (0,0, # element, training), (3,0, # element, avghr), (0,0, # text, # string) ]
[ (0, # element, session), (0, # element, comments), (0, # text, # string) ]
[ (0,0, # element, training), (4,0, # element, notes), (0,0, # text, # string) ]
[ (0, # element, session), (0, # element, distance), (0, # text, # string) ]
[ (0,0, # element, training), (0, # element, distance), (0,0, # text, # string) ]
[ (0, # element, session), (0, # element, location), (0, # text, # string) ]
[ (0,0, # element, training), (1,0, # attr, location) ]
[ (0, # element, session), (0, # attr, type) ]
[ (0,0, # element, training), (0,0, # attr, date) ]
```

虽然这个转换器文件无可否认有些神秘,但我们并不难看出它是如何工作的。每两行为一对,映射了源文件中数据和相应的目标文件中的数据。例如,下面两行显示了 session 元素的 type 属性是如何映射为 training 元素的 sport 子元素的:

```
[ (0, # element, session), (0, # attr, type) ]
[ (0,0, # element, training), (0,0, # element, sport), (0,0, # text, # string) ]
```

有了转换器文件,你就可以自由的将运动训练文件转换为 Iron Man 的 XML 文件。下面是原始格式的运动训练文件的例子:

```
<? xml version="1.0" ? >
<session type="cycling" date="12/17/99" heartrate="153">
  <duration>02:10:00</duration>
  <distance>37.0</distance>
  <location>Natchez Trace Parkway</location>
  <comments>Hilly ride, felt weak toward the end</comment>
</session>
```

要将这个代码转换为 Iron Man 格式,你必须使用 XML 转换生成器的 Transform 应用。这个应用也是 com.ibm.xtrans 包的一部分,因此同样在命令行中使用下面的命令执行:

```
java com.ibm.xtrans.Transform Train.cnv Train.xml IronMan.xml
```

Transform 应用的三个参数分别是转换器文件、源文件和目标文件。但这个命令执行的时候, IronMan.xml 文件被创建,它包含了转换为 Iron Man 文件格式的转换数据。下面是这个文件的转换代码:

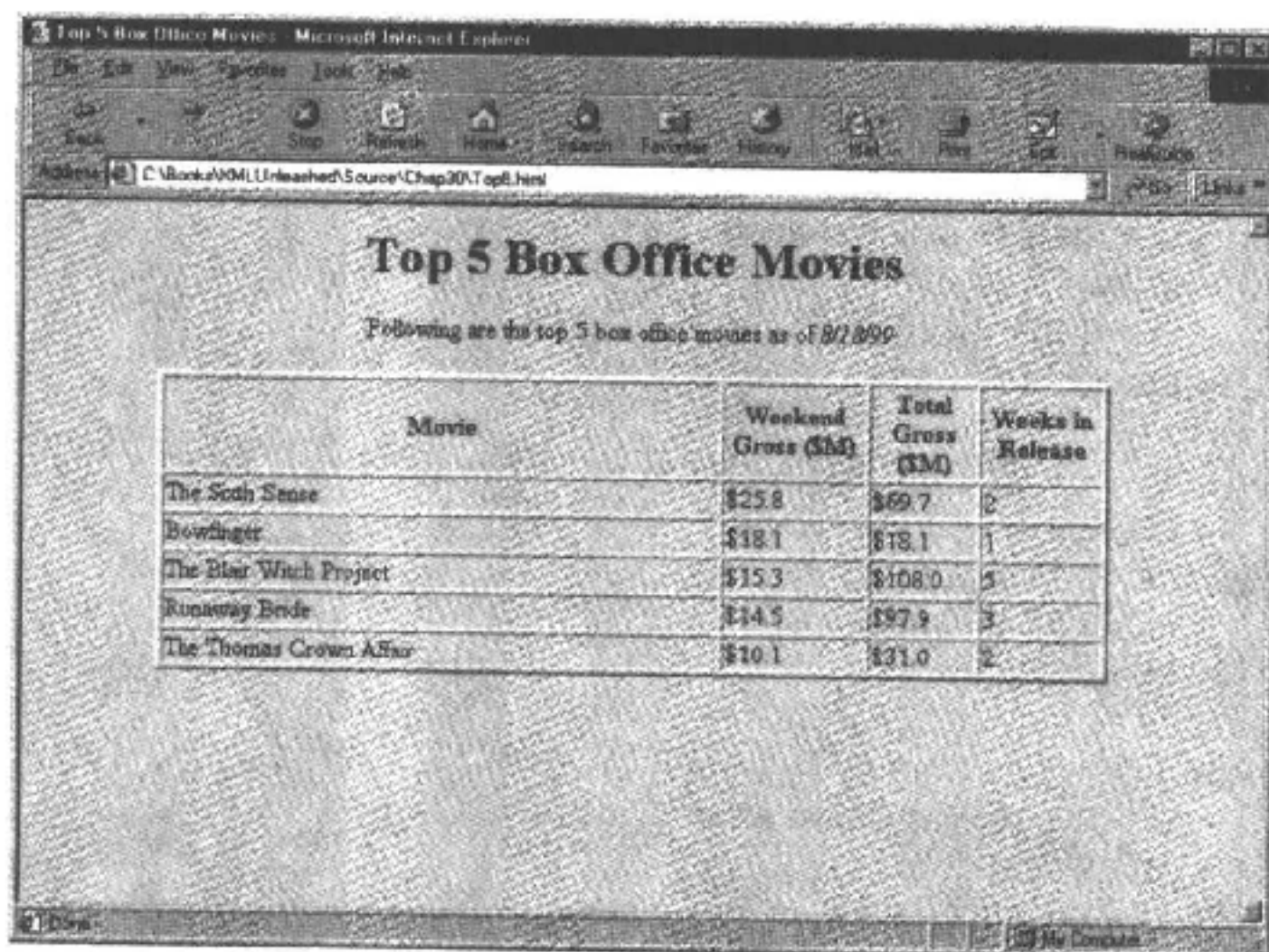
```
<? xml version="1.0" ? >
<training date="12/17/99" location="Natchez Trace Parkway">
  <sport>cycling</sport>
  <duration>02:10:00</duration>
  <avghr>153</avghr>
  <notes>Hilly ride, felt weak toward the end. </notes>
</training>
```

如你所见,来自源文件的数据被转换以适合 Iron Man XML 文件格式的结构。

30.3.2 从网页中提取数据

另一个 XML 转换生成器的有趣应用就是从网页中提取数据。由于 HTML 文件可以(应该)作为良好结构的 XML 文件构造,这就使通过使用 XML 转换生成器将网页转换为你需要的另一种 XML 文件这项工作成为了可能。然而,对于网页来说,这里有很多你不感兴趣的显示信息存在。这就是我为什么说从网页中“提取”数据,而不是简单的转换数据。

从网页中提取数据的主要目的是获得基于内容格式的数据,这样它可以更容易的被处理。事实上,任何包含感兴趣数据的网页都是对数据提取的挑战。一个有趣的例子是包含本周电影票房记录前五名信息的网页。图 30.9 显示了这样的一个网页。



| Movie | Weekend Gross (\$M) | Total Gross (\$M) | Weeks in Release |
|-------------------------|---------------------|-------------------|------------------|
| The Sixth Sense | \$25.8 | \$69.7 | 2 |
| Bowfinger | \$18.1 | \$18.1 | 1 |
| The Blair Witch Project | \$15.3 | \$108.0 | 5 |
| Runaway Bride | \$14.5 | \$97.9 | 3 |
| The Thomas Crown Affair | \$10.1 | \$31.0 | 2 |

图 30.9 在 Internet Explorer 中浏览的电影票房收入前五名的 HTML 文件的例子
这个网页的 HTML 代码如清单 30.2 所示。

清单 30.2 电影票房收入前五名的 HTML 文件 (Top5.html)

```
<? xml version="1.0"? >
<html>
  <head>
    <title>Top 5 Box Office Movies</title>
  </head>
  <body bgcolor="#B1F5B1">
    <h1 align="center">Top 5 Box Office Movies</h1>
    <p align="center">Following are the top 5 box office movies as of
    <i>8/18/99</i>;</p>
    <table align="center" border="2" width="80%">
      <tr>
```

```

        <th width="60%">Movie</th>
        <th Weekend Gross ( $ M)</th>
        <th>Total Gross ( $ M)</th>
        <th>Weeks in Release</th>
    </tr>
    <tr>
        <td>The Sixth Sense</td>
        <td>$ 25. 8</td>
        <td>$ 69. 7</td>
        <td>2</td>
    </tr>
    <tr>
        <td>Bowfinger</td>
        <td>$ 18. 1</td>
        <td>$ 18. 1</td>
        <td>1</td>
    </tr>
    <tr>
        <td>The Blair Witch project</td>
        <td>$ 15. 3</td>
        <td>$ 108. 0</td>
        <td>5</td>
    </tr>
    <tr>
        <td>Runaway Brier</td>
        <td>$ 14. 5</td>
        <td>$ 97. 9</td>
        <td>3</td>
    </tr>
    <tr>
        <td>The Thomas Crown Affair</td>
        <td>$ 10. 1</td>
        <td>$ 31. 0</td>
        <td>2</td>
    </tr>
</table>
</body>
</html>

```

电影前五名网页使用一个表来列出 1999 年 8 月 18 日的当周收入前五名的电影。要将电影数据提取到基于内容的 XML 文件中,你必须从创建 HTML 文件和结果 XML 文件类型的模板开始。清单 30.3 包含了 HTML 文件模板的代码,它与电影前五名文件本身非常类似。

清单 30.3 电影票房收入前五名 HTML 文件模板(TopTemplate.html)

```

<? xml version="1.0"? >
<html>
  <head>
    <title>Top 5 Box Office Movies</title>
  </head>

```



```

<body bgcolor="#B1F5B1">
  <h1 align="center">Top 5 Box Office Movies</h1>
  <p align="center">Following are the top 5 box office movies as of
  <i>8/18/99</i>:</p>
  <table align="center" border="2" width="80%">
    <tr>
      <th width="60%">Movie</th>
      <th>Weekend Gross( $ M)</th>
      <th>Total Gross( $ M)</th>
      <th>Weeks in Release</th>
    </tr>
    <tr>
      <td>Movie 1</td>
      <td>$ 1. 0</td>
      <td>$ 2. 0</td>
      <td>3</td>
    </tr>
    <tr>
      <td>Movie 2</td>
      <td>$ 4. 0</td>
      <td>$ 5. 0</td>
      <td>6</td>
    </tr>
    <tr>
      <td>Movie 3</td>
      <td>$ 7. 0</td>
      <td>$ 8. 0</td>
      <td>9</td>
    </tr>
    <tr>
      <td>Movie 4</td>
      <td>$ 10. 0</td>
      <td>$ 11. 0</td>
      <td>12</td>
    </tr>
    <tr>
      <td>Movie 5</td>
      <td>$ 13. 0</td>
      <td>$ 14. 0</td>
      <td>15</td>
    </tr>
  </table>
</body>
</html>

```

图 30.10 显示了这个模板文件在 Internet Explorer 中的显示情况。

HTML 文件模板与前五名电影文件本身非常类似。实际上,惟一的区别是类属数据如 Movie 1、Movie 2 等等的使用。我还在所有的美元和整型值上使用惟一的数字来帮助确保每个离散的信息都可辨别。

要生成一个转换文件,你还需要一个接收提取的电影信息的 XML 文件的模板。清单 30.4 包含了 XML 电影模板的代码。

| Movie | Weekend Gross (\$M) | Total Gross (\$M) | Weeks in Release |
|---------|---------------------|-------------------|------------------|
| Movie 1 | \$1.0 | \$2.0 | 3 |
| Movie 2 | \$4.0 | \$5.0 | 6 |
| Movie 3 | \$7.0 | \$8.0 | 9 |
| Movie 4 | \$10.0 | \$11.0 | 12 |
| Movie 5 | \$13.0 | \$14.0 | 15 |

图 30.10 电影票房收入前五名模板 HTML 文件在 Internet Explorer 中的显示情况

清单 30.4 电影票房收入前五名 XML 文件模板 (Top5Template.xml)

```
<? xml version="1.0" ? >
<top5>
  <movie weekend=" $ 1.0" total=" $ 2.0" weeks="3">
    Movie 1
  </movie>
  <movie weekend=" $ 4.0" total=" $ 5.0" weeks="6">
    Movie 2
  </movie>
  <movie weekend=" $ 7.0" total=" $ 8.0" weeks="9">
    Movie 3
  </movie>
  <movie weekend=" $ 10.0" total=" $ 11.0" weeks="12">
    Movie 4
  </movie>
  <movie weekend=" $ 13.0" total=" $ 14.0" weeks="15">
    Movie 5
  </movie>
</top5>
```

这个文件的关键是数据严格与 HTML 文件模板的相应数据匹配。此外,要注意准确的数据匹配是所有 XML 转换生成器在信息从一种文件格式转换为另一种文件格式和提取数据时所必须做到的。

为了从文件模板生成转换器文件,你要调用 XML 转换生成器中的 Map 应用,就像这

样:

```
java com.ibm.xtrans.Map Top5Template.htm Top5Template.xml Top5.cnv
```

这样创建了转换器文件 Top5.cnv, 这个文件可以用来从包含电影列表的前五名电影网页中提取数据信息。执行 Top5.html 文件的数据提取工作在前面已经提到了, 你调用 XML 转换生成器 Transform 应用。就像这样:

```
java com.ibm.xtrans.Transform Top5.cnv Top5.html Top5.xml
```

这个命令创建了 Top5.xml 文件, 它包含通过 XML 文件模板所指定的 XML 文件格式的电影数据。清单 30.5 包含了 Top5.xml 文件的代码。

清单 30.5 从 HTML 文件生成的电影票房收入前五名 XML 文件(Top5.xml)

```
<? xml version="1.0"? >
<top5>
  <movie weekend=" $ 25.8" total=" $ 69.7" weeks="2">The Sixth Sense</movie>
  <movie weekend=" $ 18.1" total=" $ 18.1" weeks="1">Bowfinger</movie>
  <movie weekend=" $ 15.3" total=" $ 108.0" weeks="5">The Blair Witch Project</
movie>
  <movie weekend=" $ 14.5" total=" $ 97.9" weeks="3">Runaway Bride</movie>
  <movie weekend=" $ 10.1" total=" $ 31.0" weeks="2">The Thomas Crown Affair</
movie>
</top5>
```

这个文件以一种比编程访问网页更容易的格式被清楚的构造。它可以被 XML 工具方便的处理。

30.4 使用 XML 区分和归并工具

XML 区分和归并工具是一个 Java 应用程序, 它用来比较良好结构的 XML 和 HTML 文件获得内容的区别。这个工具可作为内容管理工具, 探测文件版本间的改变并图形化的高亮显示区别。XML 区分和归并工具提供了使用图形化特征显示两个 XML 和 HTML 文件之间不同点的图形化用户接口。这个工具可以探测简单的内容修改, 比如一个属性值的变化和更重大的修改, 如添加和删除一个元素。

在一个多用户环境中的文件创建器将找到用于归并两个不同的人对同一文件所作的修改的工具。如果两个人修改了同一文件的不同拷贝, 这个工具就要使用。XML 区分和归并工具可以用来查明文件中的修改并成功的将它们归并到一个文件中。

XML 区分和归并工具通常处理这样的文件对: 你选择了一个基本的文件和一个与之相比较的文件。工具生成对文件的树浏览, 并且相应的区别用图形化特征和色彩高亮显示。你可以通过选择树浏览中的一个已修改元素来浏览相关文件的指定代码。

警告: 在本书编写期间, XML 区分和归并工具基于 Java 1.1 并附加 Swing 类库, 这两个软件都可以从 JavaSoft 的 Web 站点 <http://java.sun.com> 中免费获得。我尝试使用更新的 Java 2 JDK 来运行这个工具, 但并没有什么好处。alpha 工作组建议

使用支持 Java 2 版本的 XML 区分和归并工具。

作为使用 XML 区分和归并工具探测文件修改的例子,我创建了 Top5.html 网页的一个版本,名为 Top5New.html。它包含了一个不同的日期。图 30.11 显示了指定这两个文件后的 XML 区分和归并工具。

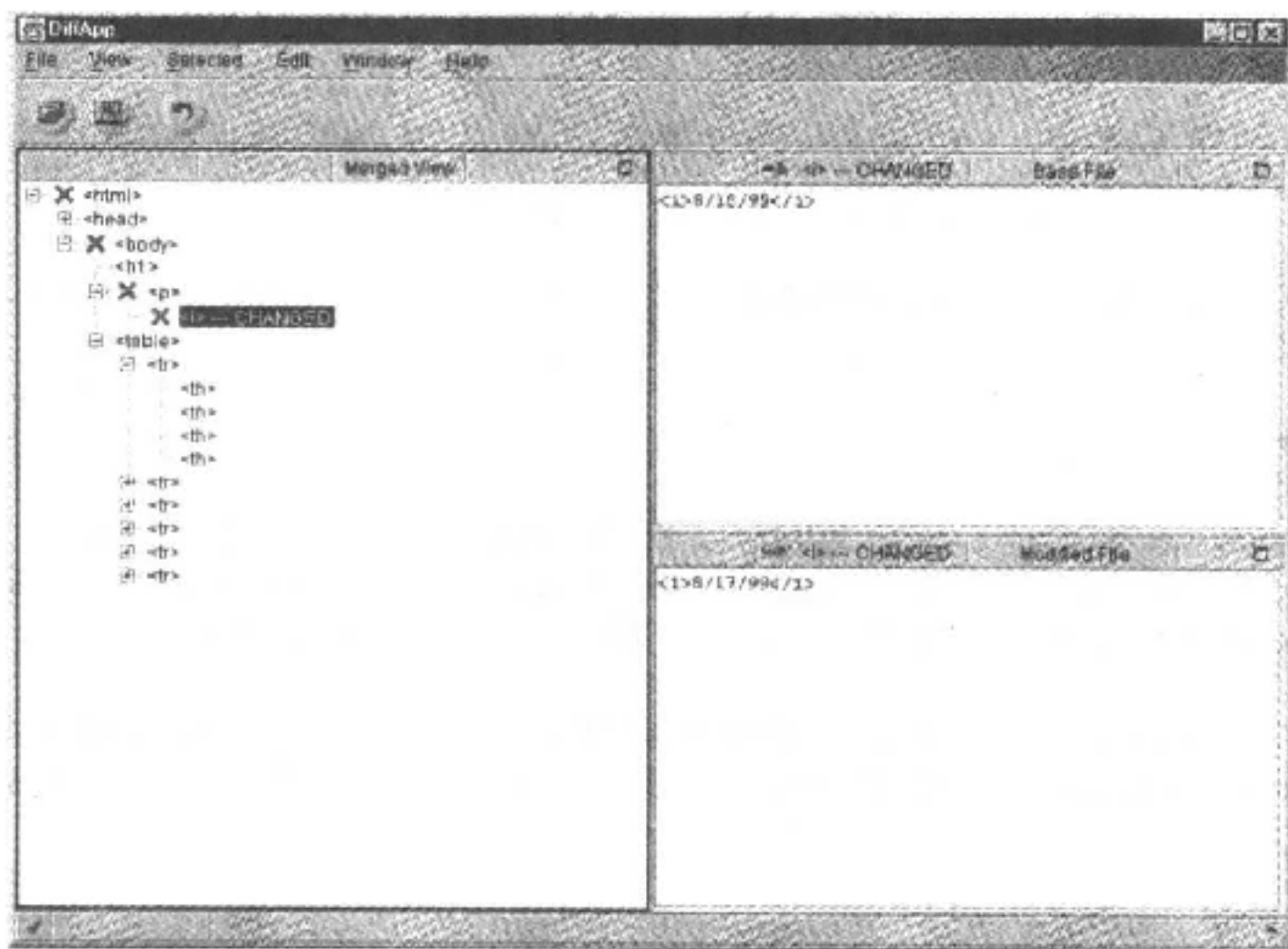


图 30.11 XML 区分和归并工具显示了在新的电影票房收入前五名 HTML 文件中的修改

如你所见,电影票房收入前五名使用 X 来引起我们对两个文件间不同数据的注意。位于屏幕右上方的窗格显示了基本的文件数据,而右下的窗格显示了在比较文件中被修改的数据。为了解决这个不同,你可以右击在树视图中的元素,并选择老(基本)值或是新值(如图 30.12 所示)。

我选择使用日期的新值。图 30.13 显示了树视图中的 X 符号是如何在区别解决后变为对勾符号的。

虽然 XML 区分和归并工具非常便利,但它并不是没有缺陷。特别是,它对于在文件中重复出现的元素的处理方面有困难。这样的元素比如是限定表中的行和列元素。如果重复元素的值被改变,这个工具将在每个文件的元素列表之间失去同步性,并认为所有的元素都改变了。例如,如果你将电影《The Blair Witch Project》的周票价改为 \$17.0。XML 区分和归并工具将错误的报告表中的所有行都改变了。图 30.14 显示了 XML 区分和归并工具做出的不正确文件评估。

很有可能的是,这个工具的未来版本将修改诸如此类的问题。同时,虽然你了解到它的局限性,但它仍然是一个有趣并且功能强大的工具。

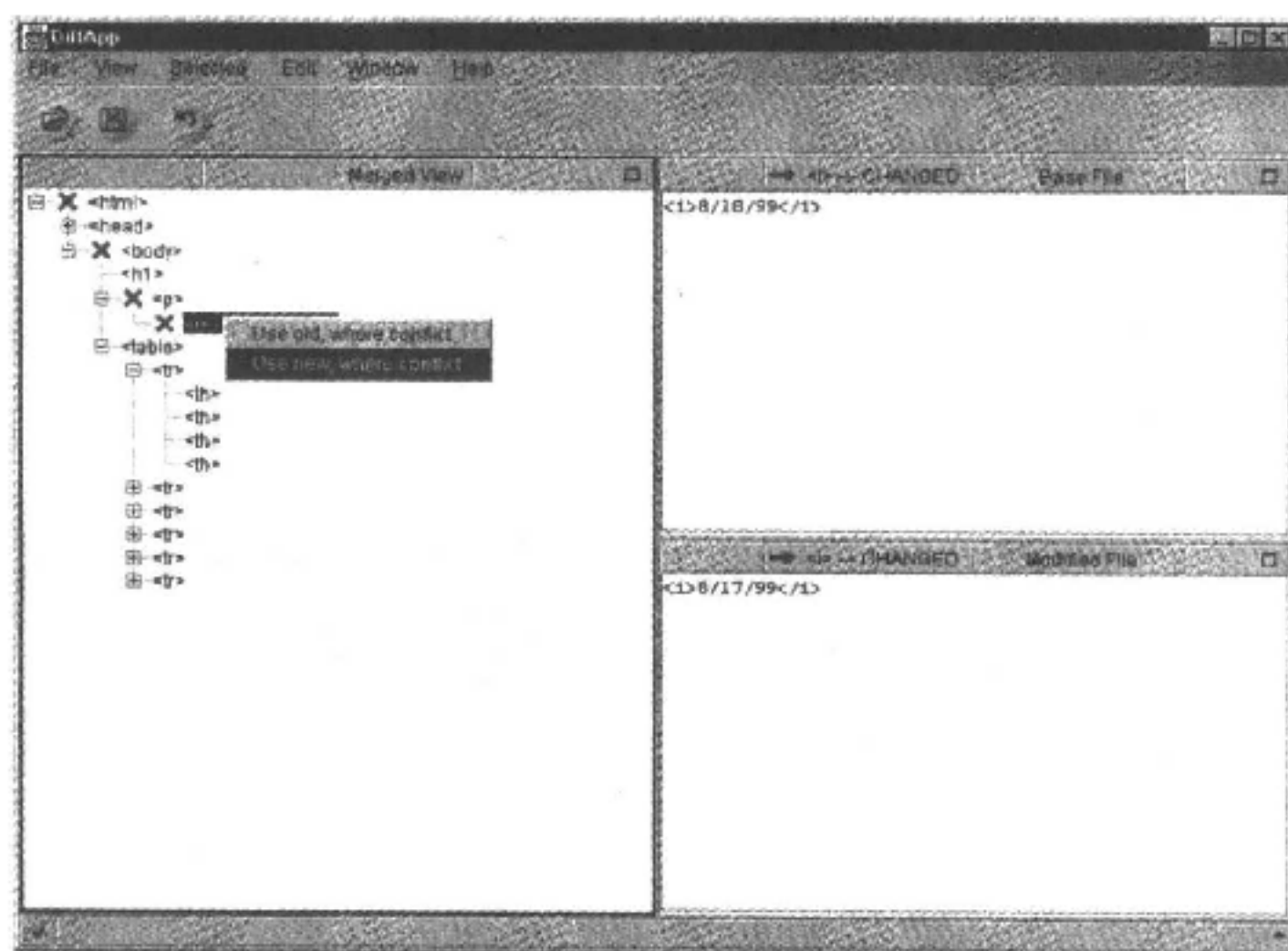


图 30.12 弹出菜单允许你解决两个电影文件间的不同

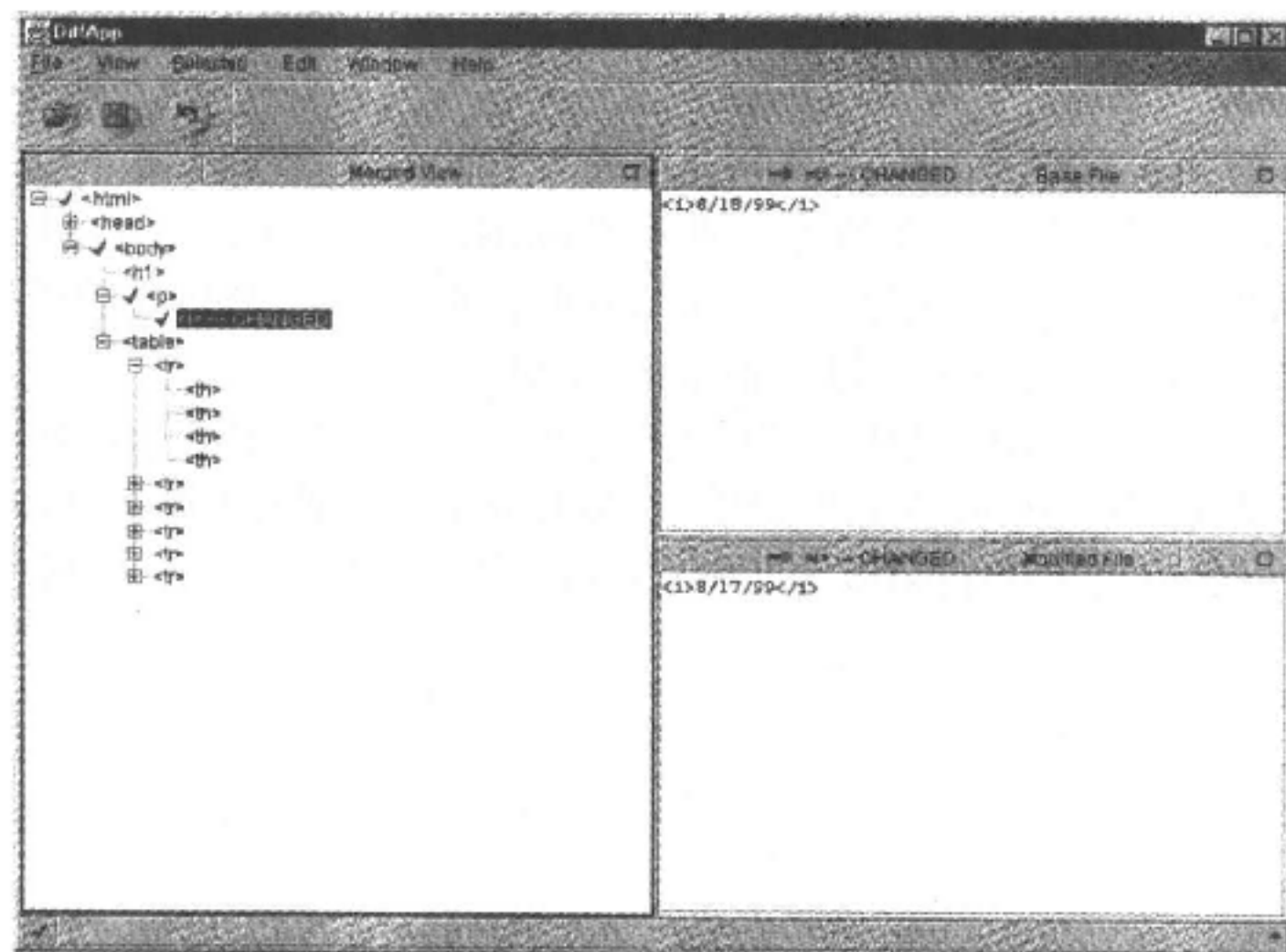


图 30.13 电影文件中的图形标号改变可以反映解决了的节点

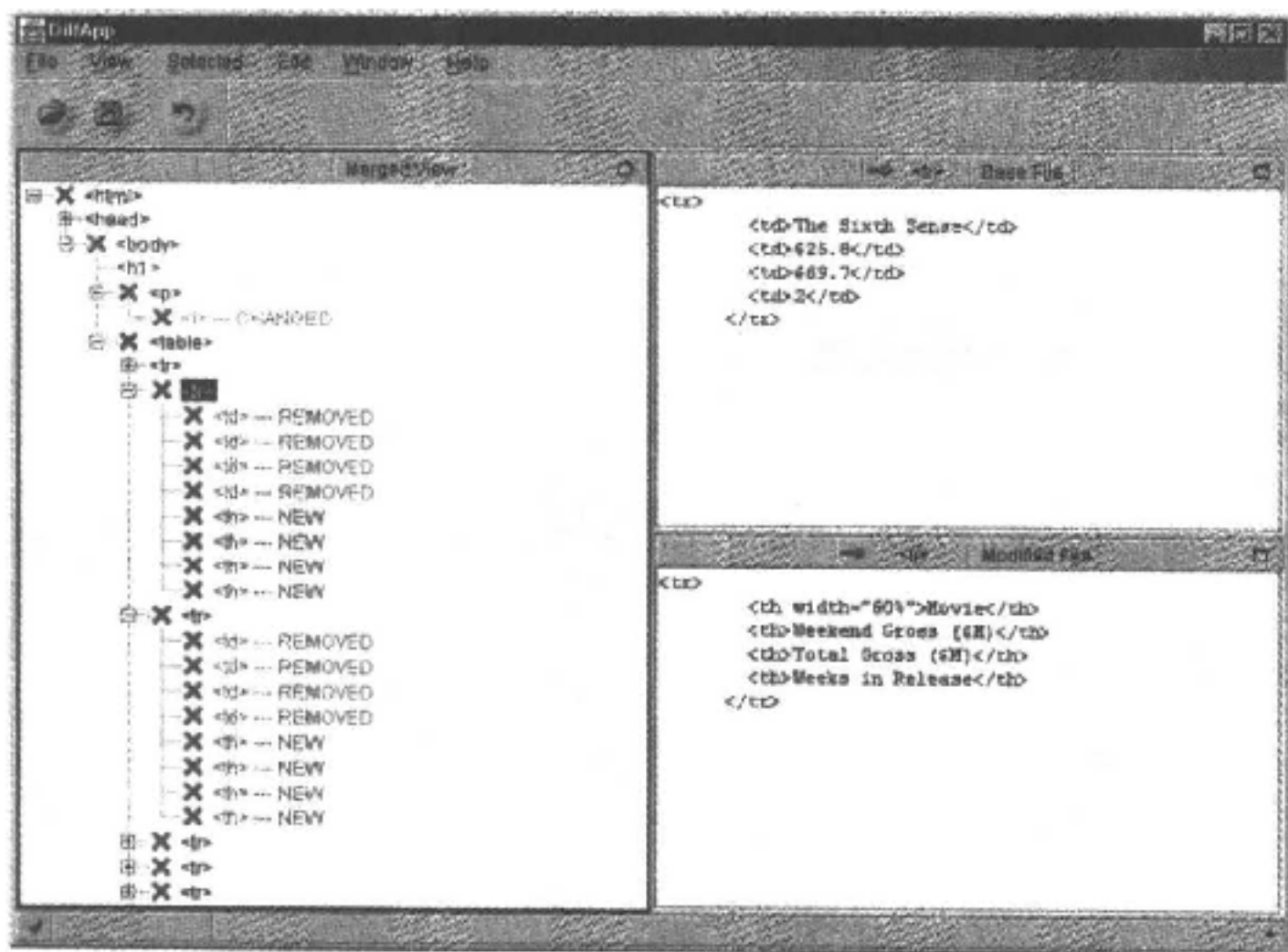


图 30.14 XML 区分和归并工具有时会在评估 XML 文件修改时发生错误

30.5 总 结

作为 Java 的强大支持者,IBM 如此迅速的变为了 XML 的鼓吹者并不令人吃惊。实际上,IBM 的 alpha 工作组从开始就致力于使用 XML 的研究。Alpha 工作组的成就达到了 XML 工具套件的深层,它可以执行一些有关访问和管理 XML 文件的有趣任务。这些工具使用 Java 编写,因此几乎可以在任何计算机平台上使用。

这一章讲解了 alpha 工作组开发的一些有趣的工具。这些工具可以搜索、浏览、验证 XML 文件以及转换、区分和归并文件数据。大多数的工具在不断的取得进步,IBM 也依旧忙于改进它们。然而,它们当前的版本已经足够用来执行重要的 XML 文件管理任务。

第 7 部分 XML 词表探讨

第 31 章 用 XMLNEWS 制作新闻

如你所知,XML 最大的益处就是可以将内容加入到信息中。新闻就是一种特别的信息类型,它可以从内容中获得极大的益处。过去,新闻只是单独的由特别的故事组成并用简单的格式建立的文本。例如,它可能用指定的标号来书写名称和场所,这样 Florence, Alabama (城市名)就可以与 Florence Henderson(演员名)相区分了。

本章向你介绍 XMLNews,这个 XML 规范定义了两组词表以方便新闻故事加入更加丰富的内容。这些 XMLNews 词表定义了书写诸如日期和时间、人物、地点、组织等等的信息的元素。XMLNews 还在加入你要求的大量小内容标识方面给了你充分的自由。对于有严格截止期限的内容开发人员来说,它可以很好的完成工作。他们也许更多的关注于获得一个故事以完成工作,这种情况下他们可能需要回去并加入一些内容标识。

31.1 理解 XMLNews

与任何电子化传输的信息相同,新闻必须以规定的格式存储,这样新闻代理商可以将它与客户共享。你可能已经听说过新闻在“在线”发表。虽然“在线”在这几年中已经有了些改变,实际的新闻格式却并没有变。成熟的新闻标准称为 ANPA 1312,通过提供一种在线服务方式以一致性地为故事提交给报纸,它在新闻发布中扮演着重要的角色。它还声明了用来描述故事显示的排版标识。一般而言,所有的北美在线服务和在线收集软件都依赖 ANPA 1312 作为新闻描述的标准。

在 90 年代初期,很长时间各主流的媒体都在谈论 Y2K(2000 年)问题,它对新闻产业内部产生的影响,不仅在于 ANPA 1312 是一个过时的技术,还因为它在由 Y2K 问题所带来的一系列问题上的缺陷。来自一些新闻组织的代表宣布创建一个新的用于在新闻代理和它们的客户端之间共享文本新闻的标准。这项工作的结果就是新闻工业文本格式(NITF——News Industry Text Format),它是一个由国际新闻通讯委员会(IPTC——International Press Telecommunications Council)维护的基于 XML 的词表。NITF 已经通过了 IPTC 以及美国新闻联合会(NAA——Newspaper Association of America)和广播电视新闻理事联合会的验证。虽然 NITF 极大的迎合了新闻组织的需要,但它比较复杂并且包含了一些并不直接应用于构造在线发布新闻故事的元素。

XMLNews 是由 David Megginson 创建的一个面向 Web 的新闻规范。它定义了两个 XML 词表,用于在标准的格式下共享新闻。XMLNews-Story 是 NITF 的一个简化子集,用来标记新闻故事,而 XMLNews-Meta 是一个基于 RDF 的词表,用来指定存在的新闻对象的

元信息。由于 XMLNews-Story 提供了标记新闻故事的方法,同时还提供了元信息,本章将主要讲述 XMLNews-Story。

注意:David Megginson 是 Megginson 技术的负责人,W3C 的 XML 信息集工作组主席和 SAX(Simple API for XML)的维护人。

你也许想知道为什么在 NITF 已经存在的情况下还需要 XMLNews-Story。XMLNews-Story 的目标是提供一个用于在线加工和描述新闻故事的简单的词表。使用 XMLNews-Story,我们就可能创建有着丰富标识的新闻故事,它将包含故事中各项目的内容信息,比如人物、地点和组织,有时还会是名称。XMLNews-Story 词表直截了当并易于使用。实际上,大多数的内容指定元素是可选的,也就是说内容开发人员可以自由的决定他们希望在故事中加入的元信息的准确数目。XMLNews 的 Web 站点(<http://www.xmlnews.org>)包含着有关 XMLNews 规范的其他信息。

支持 XMLNews 的第一个商业化产品是 NewsPak,这是一个新闻配送软件,包含着大多数主要在线服务的预先许可的安排。NewsPak 由 WavePhore 推出,它也恰好是负责资助 David Megginson 的 XMLNews 规范的公司。下面是用在 NewsPak 中的预先许可的在线服务:国际联合新闻组织(United Press International)、联合新闻在线(Associated Press Online)和体育网(Sports Network)。

本质上,NewsPak 是一个加强的新闻供给器,它减轻了要直接访问多个新闻源的负担。由于 NewsPak 是基于 XML 的,这就可能方便的将来自多个源的新闻信息供给结合为单一的供给。加强的 NewsPak 供给由实时在线新闻、日报内容、期刊和预定报告组成。NewsPak 供给以结构化 XML 数据发布,它可以根据特定 Web 站点的需要被格式化和显示。NewsPak 软件是一个 Windows NT 服务、Perl 脚本和 HTML 网页的结合物。为了使用 NewsPak,你必须要有基于 Windows NT 服务器的 Web 站点,使用 Microsoft Internet Information Server 作为网络服务器软件。要获得有关 NewsPak 的更多信息,请访问 WavePhore 的 NewsPak Web 站点 <http://www.newspak.com>。

31.2 深入 XMLNews-Story 词表

虽然 XMLNews-Story 词表是新闻工业文本格式(NITF)的简化版本,它依然定义了数目可观的用于描述新闻故事的元素。在本章的后面,你将学习到如何使用 XMLNews-Story 来创建新闻故事文件。清单 31.1 包含了完整的 XMLNews-Story 的 DTD。

清单 31.1 XMLNews-Story 的文件类型定义(DTD)

```
<? xml version = "1.0" encoding="UTF-8"? >
<! ELEMENT a (#PCDATA | chron | copyrite | event | function | location |
  money | num | object.title | org | person | virtloc | a + br | em |
  lang | pronounce | 1) * >
<! ATTLIST a
  href CDATA #REQUIRED>
<! ELEMENT audio (audio.cation?,audio.producer?,audio.dta)>
```



```

<! ATTLIST audio
  src CDATA #REQUIRED
  length NMTOKEN # IMPLIED>
<! ELEMENT audio.cation (cation)>
<! ELEMENT audio.data EMPTY>
<! ATTLIST audio.data
  copyright CDATA #IMPLIED>
<! ELEMENT audio.producer (byline)+>
<! ELEMENT base EMPTY>
<! ATTLIST base
  href CDATA #REQUIRED>
<! ELEMENT block (dateline?,copyrite?,(img | audio | video | p | ol |
  ul | dl | table | bq | pre) * ,datasource?)>
<! ELEMENT body (body.head?,body.content?,body.end?)>
<! ELEMENT body.content (p | ol | ul | dl | h1 | h2 | h3 | h4 | block) * >
<! ELEMENT body.end (tagline)>
<! ELEMENT body.head (hedline?,byline * ,distributor?,dateline?,
  series?)>
<! ELEMENT bq ((h1 | h2 | h3 | h4 | block)+,credit?)>
<! ELEMENT br EMPTY>
<! ELEMENT byline (bytag)>
<! ELEMENT bytag (#PCDATA)>
<! ELEMENT caption (#PCDATA | chron | copyrite | event | function |
  location | money | num | object.title | org | person | virtloc | a |
  br | em | lang | pronounce | q) * >
<! ELEMENT chron (#PCDATA)>
<! ATTLIST chron
  norm CDATA # IMPLIED>
<! ELEMENT city (#PCDATA)>
<! ELEMENT copyrite (#PCDATA | copyrite.year | copyrite.holder) * >
<! ELEMENT copyrite.holder(#PCDATA)>
<! ELEMENT country (#PCDATA)>
<! ELEMENT credit (#PCDATA | chron | copyrite | event | function |
  location | money | num | object.title | org | person | virtloc | a |
  br | em | lang | pronounce | q) * >
<! ELEMENT datasource (#PCDATA)>
<! ELEMENT dateline (location,story.date?)>
<! ELEMENT dd (block)>
<! ELEMENT denom (#PCDATA)>
<! ELEMENT distributor (#PCDATA | org) * >
<! ELEMENT dl (lh?,(dt?,dd)+)>
<! ELEMENT dt (#PCDATA | chron | copyrite | event | function |location |
  money | num | object.title | org | person | virtloc | a | br | em |

```

```

    lang | pronounce | q) * >
<! ELEMENT em (#PCDATA)>
<! ELEMENT event (#PCDATA)>
<! ELEMENT frac (numer,denom)>
<! ELEMENT function (#PCDATA)>
<! ELEMENT h1 (#PCDATA | chron | copyrite | event | function |location |
    money | num | object.title | org | person | virtloc | a | br | em |
    lang | pronounce | q) * >
<! ELEMENT h2 (#PCDATA | chron | copyrite | event | function |location |
    money | num | object.title | org | person | virtloc | a | br | em |
    lang | pronounce | q) * >
<! ELEMENT h3 (#PCDATA | chron | copyrite | event | function |location |
    money | num | object.title | org | person | virtloc | a | br | em |
    lang | pronounce | q) * >
<! ELEMENT h4 (#PCDATA | chron | copyrite | event | function |location |
    money | num | object.title | org | person | virtloc | a | br | em |
    lang | pronounce | q) * >
<! ELEMENT head (title,base?)>
<! ELEMENT headline (h1,h2 * )>
<! ELEMENT h1 (#PCDATA | chron | copyrite | event | function |location |
    money | num | object.title | org | person | virtloc | a | br | em |
    lang | pronounce | q) * >
<! ELEMENT h2 (#PCDATA | chron | copyrite | event | function |location |
    money | num | object.title | org | person | virtloc | a | br | em |
    lang | pronounce | q) * >
<! ELEMENT img (img.caption?,img.producer?,img.data)>
<! ATTLIST img
    src CDATA #REQUIRED
    width NMTOKEN #IMPLIED
    height NMTOKEN #IMPLIED>
<! ELEMENT img.cation (cation)>
<! ELEMENT img.data EMPTY>
<! ATTLIST img.data
    copyright CDATA #IMPLIED>
<! ELEMENT img.producer (byline)+>
<! ELEMENT lang (#PCDATA)>
<! ATTLIST lang
    lang NMTOKEN #IMPLIED>
<! ELEMENT lh (#PCDATA | chron | copyrite | event | function |location |
    money | num | object.title | org | person | virtloc | a | br | em |
    lang | pronounce | q) * >
<! ELEMENT li (block)>
<! ELEMENT location (#PCDATA | sublocation | city | state | region |
    country) * >
<! ELEMENT money (#PCDATA)>
<! ATTLIST money
    unit CDATA #IMPLIED>

```

```

<! ELEMENT name.family (#PCDATA)>
<! ELEMENT name.given (#PCDATA)>
<! ELEMENT ntf(head,body)>
<! ATTLIST ntf
  baselang CDATA #IMPLIED
  change.date CDATA #FIXED " $Date:1999/04/05 01:06:04 $"
  change.time CDATA #FIXED "0000"
  unr CDATA #IMPLIED
  version CDATA #FIXED
  "-//XMLNews//DTD XMLNEWS-STORY $Revision: 1.8 $//EN">
<! ELEMENT num (#PCDATA | frac | sub | sup) * >
<! ELEMENT numer (#PCDATA)>
<! ELEMENT object.title (#PCDATA)>
<! ELEMENT ol(lh?,li+)>
<! ELEMENT org (#PCDATA | orgid) * >
<! ELEMENT orgid EMPTY>
<! ATTLIST orgid
  idsrc CDATA #REQUIRED
  value CDATA #REQUIRED>
<! ELEMENT p (#PCDATA | chron | copyrite | event | function | location |
  money | num | object.title | org | person | virtloc | a | br | em |
  lang | pronounce | q) * >
<! ELEMENT person (#PCDATA | name.given | name.family | function) * >
<! ELEMENT pre (#PCDATA)>
<! ELEMENT pronounce EMPTY>
<! ATTLIST pronounce
  guide CDATA #IMPLIED
  phonetic CDATA #IMPLIED>
<! ELEMENT q (#PCDATA | chron | copyrite | event | function | location |
  money | num | object.title | org | person | virtloc | a | br | em |
  lang | pronounce | q) * >
<! ELEMENTN region (#PCDATA)>
<! ELEMENT series EMPTY>
<! ATTLIST series
  series.name CDATA #IMPLIED
  series.part NMTOKEN "0"
  series.totalpart NMTOKEN "0">
<! ELEMENT state (#PCDATA)>
<! ELEMENT story.date (#PCDATA)>
<! ELEMENT sub (#PCDATA)>
<! ELEMENT sublocation (#PCDATA)>
<! ELEMENT sup (#PCDATA)>
<! ELEMENT table (caption?,thead?,tfoot?,tbody+)?
<! ELEMENT tagline (#PCDATA | chron | copyrite | event | function | location |
  money | num | object.title | org | person | virtloc | a | br | em |

```

```

    lang | pronounce | q) * >
<! ELEMENT tbody (tr+)>
<! ATTLIST tbody
    align (left | center | right | justify) #IMPLIED
    valign (top | middle | bottom | baseline) #IMPLIED>
<! ELEMENT td (#PCDATA | chron | copyrite | event | function | location |
    money | num | object.title | org | person | virtloc | a | br | em |
    lang | pronounce | q) * >
<! ATTLIST td
    rowspan NMTOKEN "1"
    colspan NMTOKEN "1"
    align (left | center | right | justify) #IMPLIED
    valign (top | middle | bottom | baseline) #IMPLIED>
<! ELEMENT tfoot (tr+)>
<! ATTLIST tfoot
    align (left | center | right | justify) #IMPLIED
    valign (top | middle | bottom | baseline) #IMPLIED>
<! ELEMENT dt (#PCDATA | chron | copyrite | event | function | location |
    money | num | object.title | org | person | virtloc | a | br | em |
    lang | pronounce | q | p | ol | ul | dl | h1 | h2 | h3 | h4) * >
<! ATTLIST th
    rowspan NMTOKEN "1"
    colspan NMTOKEN "1"
    align (left | center | right | justify) #IMPLIED
    valign (top | middle | bottom | baseline) #IMPLIED>
<! ELEMENT thead (tr+)>
<! ATTLIST thead
    align (left | center | right | justify) #IMPLIED
    valign (top | middle | bottom | baseline) #IMPLIED>
<! ELEMENT title (#PCDATA)>
<! ELEMENT tr (th | td)+>
<! ATTLIST tr
    align (left | center | right | justify) #IMPLIED
    valign (top | middle | bottom | baseline) #IMPLIED>
<! ELEMENT ul (lh?,li+)>
<! ELEMENT video (video.caption?,video.producer?,video.data)>
<! ATTLIST video
    src CDATA #REQUIRED
    length NMTOKEN #IMPLIED>
<! ELEMENT video.caption (caption)>
<! ELEMENT video.data EMPTY>
<! ATTLIST video.data
    copyright CDATA #IMPLIED>
<! ELEMENT video.producer (byline)+>
<! ELEMENTN virtloc (#PCDATA)>

```

如清单 31.1 所示,虽然是 NITF 的子集,XMLNews-Story DTD 仍然非常的复杂。虽然 XMLNews-Story 定义了相当数量的元素,但只有部分元素对于 XMLNews-Story 文件是重

要的。更明确来讲,XMLNews-Story 词表基于三个基本元素:

- ntif
- head
- body

ntif 元素是 NTIF 文件包括 XMLNews-Story 文件的根元素,或者说是文件元素。因此,你总可以将所有的 XMLNews-Story 内容加入到<ntif>和</ntif>标记之间。Head 元素与 HTML 中的 head 元素很相似;它主要作为定义应用于文件整体的头信息的空间。一个 XMLNews-Story 文件的头信息的例子就是文件的标题。Body 元素包含着实际新闻故事的信息,包括诸如标题、标题署名这样的前置信息以及故事的文本和任何附加的内容标识。body 元素更多的以子元素 body.head 和 body.content 方式使用。

让我们详细的考虑一下这些基本的 XMLNews-Story 元素。在本章后面,你将使用这些元素来从草稿开始构造一个 XMLNews-Story 文件。

31.2.1 ntif 元素

ntif 元素是 XMLNews-Story 文件的文件元素。它作为另外两个基本元素(head 和 body)的容器元素。XMLNews-Story 文件的框架结构如下所示:

```
<ntif>
  <head>
  </head>

  <body>
    <body.head>
    </body.head>

    <body.content>
    </body.content>
  </body>
</ntif>
```

如你所见,head 和 body 元素被包含在 ntif 元素中。同样,body 元素被分为两个子元素——body.head 和 body.content,他们描述了新闻故事的不同部分。新闻故事的标题、署名和日期栏必须放置在 body.head 元素中,而实际的故事文本必须放置在 body.content 元素中。虽然 body.head 元素包含故事的标题,但你还必须在 head 元素中使用<title>元素提供故事的题目。下面是使用 title 元素设置文件题目的例子:

```
<head>
  <title>Bigfoot Spotted at Starbucks</title>
</head>
```

一般情况下,可以在任何场合作为故事的简短描述的文件题目是需要的,比如在新闻故事的列表中或在显示故事的窗口的标题栏中。文件题目可以与文件的标题相同,也可以不同。

31.2.2 body.head 元素

body.head 元素用来描述 XMLNews-Story 的题头部分,包括标题,署名和文件的日期栏。下面的子元素用来描述信息的这些部分:

- headline
- byline
- dateline

这些元素都是容器元素,它依赖于子元素来定义每一个指定的题头部分的相关信息。下面几小节将告诉你如何使用这些元素。

headline 元素

headline 元素用来描述新闻故事的标题。元素本身是一个容器元素,在其中,你必须使用 h1 和 h2 子元素来描述标题。使用一个 h1 元素描述主标题,不使用或使用或者多个 h2 元素来描述子标题。下面是一个例子,它告诉我们 h1 子元素是如何用在 headline 元素中描述新闻故事的主标题的:

```
<headline>
  <h1>Bigfoot Spotted at Starbucks</h1>
</headline>
```

这个标题与你在前面见到的文件题目相同,这也是 XMLNews-Story 文件中的通常情况。

byline 元素

byline 元素用来描述新闻故事的署名,确定文章的作者。这个元素实际上依赖 bytag 子元素来确定署名内容,这个内容中作者名要跟在单词“by”之后,并且有可选的新闻组织的从属关系表示内容。下面是一个通过 byline 和 bytag 元素描述的署名的例子:

```
<byline>
  <bytag>By Keith Nash, Sheboygan Sentinel Staff Writer</bytag>
</byline>
```

dateline 元素

dateline 元素用来描述新闻故事的写作地点和日期。Location 和 story.date 子元素用来分别描述日期栏中的各部分。下面是一个讲解用 dateline、location 和 story.date 元素描述日期栏的例子:

```
<dateline>
  <location>Sheboygan, Wisconsin</location>
  <story.date>Wednesday December 8 1999 10:28 ET</story.date>
</dateline>
```

story.date 元素在 XMLNews-Story 的 DTD 中被定义为包含 #PCDATA,也就是说它的格式无关紧要。换句话说,你可以自由的按照你希望的任何方式书写时间。在这个例子中,日期以非常明确的方式组织起来。

31.2.3 body.content 元素

body.content 元素是实际的新闻故事文本放置的位置。在 body.content 元素中有大量的标识元素以实现故事的丰富标识。你可以自由的按照你的期望或多或少的使用这些元素,这取决于你希望在故事中加入多少元信息。在 body.content 元素中只有一个子元素你必须使用——p 元素。这个元素用在 XMLNews-Story 中以包含内容段落,这与它在 HTML 语言中使用的功能相同。

除了 p 元素,下面的这些子元素也可以在 body.content 元素中使用:

- q——直接引用。
- person——人名(真实的或虚构的)。
- function——人的角色和作用。
- object.title——文化作品(书籍、电影、绘画等等)的题目。
- chron——日期和时间。
- location——地理位置。
- org——组织(政府、公司、俱乐部等等)的名称。
- event——事件(历史、体育、灾祸等等)。
- copyrite——版权声明。
- money——货币价值。
- num——数字表达式(包括分数)。
- em——加重表示。
- pronounce——音标。
- br——行结束。
- a——HTML 链接或链接的目标。
- virtloc——URL 或电子邮件地址(有效地址)。
- lang——不同语言中的习语和方言。

虽然所有的这些元素在一定情况下都是很有用的,但我更希望将注意力集中在 body.content 中使用最为广泛的标识元素。下面的几节将更详细的探讨这些元素。

q 元素

q 元素用来标记直接引用。它可以用来代替引用记号。这个元素与引用记号的用法相同,即包围所引用文本。下面是一个使用 q 元素标记直接引用的例子:

```
<p>One bystander was quoted as saying <q>I turned around with my latte and found my self  
staring right at the big hairy beast. </q></p>
```

chron 元素

chron 元素用来指定与文章内容相关的日期和时间。在最简单的格式中,chron 元素可以用来确定一段文本来描述日期/时间信息。chron 的最为完整的应用就是使用 norm 属性指定文章内容的标准日期/时间。norm 属性的格式遵循 ISO 8601 的日期/时间格式,这个格式就像这样:

YYYYMMDD

这个代码向你展示了包含年、月、日在内的日期字段的格式。如果你希望还要包含时间，则代码可以扩展如下：

YYYYMMDDTHHMMSS

在这个格式中，T 用来分隔日期和时间，而时间有时、分和可选的秒组成。为了说明时区，你可以在值的后面加一个 Z 来说明是格林威治时间，或者也可以加上一个 + 号或 - 号和一个 HHMM 偏移量。例如，美国山区时间比格林威治时间晚七个小时，这样你可以在日期/时间值后加 -0700。下面是一个使用 `chron` 元素指定文本确切时间的例子：

```
<p>The elusive creature known to many as Bigfoot was spotted outside of Starbucks in midtown
Sheboygan
<chron norm="19991207T1620-0600">yesterday afternoon</chron>. One bystander was quoted
as saying <q>I turned around with my latte and found myself staring right at the big hairy beast. </q>
</p>
```

在这个例子中，`chron` 元素用来阐明文本“yesterday afternoon”所指代的日期和时间。这种情况下，`norm` 属性被设置为中央标准时间 1999 年 12 月 7 日下午 4:20（美国中央标准时间比格林威治时间晚六个小时）。

location 元素

`location` 元素用来指定与文章内容相关的地点。在最简单的格式下，`location` 元素可以用一段文字来描述地点。然而，使用 `location` 元素中的子元素来定义有关地点的更详细信息这种方式更为有用。下面的子元素可以用在 `location` 元素中提供有关地点的更详细信息：`sublocation`、`city`、`state`、`region`、`country`。这些子元素都是简单的容器元素，包含 #PCDATA 并且没有属性。下面是一个使用 `location` 元素指定地点的例子：

```
<p>The elusive creature known to many as Biffoot was spotted outside of
Starbucks in <location><sublocation>midtown</sublocation>
<city>Sheboygan</city></location>
<chron norm="19991007T1620-0600">yesterday afternoon</chron>. One
bystander was quoted as saying <q>I turned around with my latte and found myself staring right at
the big hairy beast. </q></p>
```

要注意 `sublocation` 元素（midtown）和 `city` 元素（Sheboygan）都是在 `<location>` 标记中确定的。要谨记，在这样的例子中，标识的层次非常的灵活。你可以只使用 `<location>` 标记而不是用 `<sublocation>` 和 `<city>` 标记，或者连 `location` 标识也没有。

object. title 元素

`object. title` 元素用来指定文化作品的题目。与其他的 `body. content` 标识元素类似，`object. title` 元素用来标识文本而不加入任何的属性。下面是使用 `object. title` 指定文学作品的例子：

```
<p>A full account of this Bigfoot sighting will appear in a future
compilation titled <object. title>Coffee with Bigfoot</object. title>. </p>
```


org 元素

org 元素用来指定新闻故事中的组织。这个组织可以是政府、公司、俱乐部和其他任何包含一群人的实体。下面是一个使用 org 元素确定组织的例子：

```
<p>For more information on the <org>Starbucks</org> Bigfoot sighting,  
please attend the next meeting of the <org>Sheboygan Bigfoot watchers  
Club</org>. </P>
```

person 元素

person 元素用来指定新闻中的人物。下面的子元素可以用在 person 元素中以描述一个人物：name.given、name.family 和 function。name.given 和 name.family 元素所对应的就是人的名和姓，而 function 元素用来描述人物的作用。这个作用可以是他在公司中的地位，它在一个事件中的角色或者仅仅是他的职业。下面是一个使用 function 元素指定人物的例子：

```
<p>For more information on the <org>Starbucks</org> Bigfoot sighting,  
please attend the next meeting of the <org>Sheboygan Bigfoot Watchers  
Club</org>. Famed <person><function>Bigfoot tracker</function>  
<name.given>Travis</name.given>  
<name.family>Foster</name.family></person> will be on hand to discuss the  
ramifications of a caffeinated beast roaming the streets of  
Sheboygan. </p>
```

em 元素

em 元素用来着重表示一部分文本。例如，你也许希望将一部分文本着重显示给读者以说明这部分内容非常的重要。用来显示 XMLNews-Story 文件的软件可以通过不同的色彩来显示强调的文本，或者是通过粗体或斜体字方式。下面是一个使用 em 元素着重标识一部分文本的例子：

```
<p>For more information on the <org>Starbucks</org> Bigfoot sighting,  
please attend the next meeting of the <org>Sheboygan Bigfoot Watchers  
Club</org>. Famed <person><function>Bigfoot tracker</function>  
<name.given>Travis</name.given>  
<name.family>Foster</name.family></person> will be on hand to discuss the  
ramifications of a caffeinated beast roaming the streets of  
Sheboygan. <person><name.family>Foster</name.family></person> recommends  
<em>not</em> trying to shake hands with the creature, as it may mistaken  
your outstretched hand as food</p>
```

31.3 使用 XMLNews 创建新闻故事

你已经学习了用在 XMLNews-Story 文件中的主元素。虽然你已经看到了这些元素的一些例子，但现在看一个将它们在单独文件中共同使用的例子仍将更有帮助。你可以使用 XMLNews-Story 文件中的多级别的标识，这只取决于你要提供多少元信息。

在这里,我希望给出一些用不同层标识编写的同一个新闻故事的例子。清单 31.2 包含一个具有最少标识的完整的 XMLNews-Story 文件。

清单 31.2 以最少标识编写的 XMLNews-Story 文件

```
<? xml version="1.0">>
<nitf>
  <head>
    <title>Nerd Author Strikes XML Gold</title>
  </head>
  <body>
    <body.head>
      <headline>
        <h1>Nerd Author Strikes XML Gold</h1>
      </headline>
      <byline>
        <bytag>By Josh Timm</bytag>
      </byline>
      <dateline>
        <location>Nashville, Tennessee</location>
        <story.date>Monday March 20 2000 9:43 ET</story.date>
      </dateline>
    </body.head>
    <body.content>
      <p>Acclaimed nerd author Michael Morrison strikes gold with his immensely popular book XML Unleashed. At approximately 8:55 ET last night, the one millionth copy of XML unleashed was sold at a Barnes and Noble bookstore in suburban Nashville, Tennessee, which makes the book the best-selling computer book of all time. The book explores the ins and outs of the XML information technology that has quickly taken the Web by storm. It's not really apparent whether the book's success can be attributed to Morrison's genius and command over the XML technology, or just dumb luck. </p>
      <p>When asked about the sudden success of his book, Morrison replied <1>I'm just a regular guy trying to state the facts. </q> Regular or not, Morrison will now be spending most of his time at the bank depositing royalty checks. When asked what he plans to do with his newfound riches, Morrison stated <1>I don't plan on changing my lifestyle much, except for buying a new house, a half-dozen cars, and a fat diamond for my wife, and constructing my own extreme sports park on the 1,000 acre farm that I'm currently in negotiations to buy. </q></p>
    </body.content>
  </body>
</nitf>
```

就像你猜到的那样,这并不是一个真实的新闻故事。我只是希望这与事实相差不要太远!总之,这个文件显示了 XMLNews-Story 所有的这些元素是如何协同使用创建一个完整文件的。注意在故事中只使用了很少的标识。实际上,在 body.content 部分只使用了两个标

记,就是段落标记(<p>)和引用标记(<q>)。这个文件示范了在一个 XMLNews-Story 文件中所需要用到的最少标识。

下一级的标识将包含元信息来强调地点、组织、人物和文化作品。清单 31.3 包含了使用附加标识强调这些部分信息的文件代码。

清单 31.3 以中等数量标识编写的 XMLNews-Story 文件

```
<? xml version = "1.0"? >
<nitf>
  <head>
    <title>Nerd Author Strikes XML Gold</title>
  </head>
  <body>
    <body.head>
      <headline>
        <h1>Nerd Author Strikes XML Gold</h1>
      </headline>
      <byline>
        <bytag>By Josh Timm</bytag>
      </byline>
      <dateline>
        <location>Nashville, Tennessee</location>
        <story.date>Monday March 20 2000 9:43 ET</story.date>
      </dateline>
    </body.head>
    <body.content>
      <p>Acclaimed <person><function? nerd author</function> Michael
Morrison</person> strikes gold with his immensely popular book
<object.title>XML Unleashed</object.title>. At approximately
8:55 ET last night, the one millionth copy of
<object.title>XML Unleashed</object.title> was sold at a
<org>Barnes and Noble</org> bookstore in <location>suburban
Nashville, Tennessee</location>, which makes the book the best-
selling computer book of all time. The book explores the ins and
outs of the XML information technology that has quickly taken the
Web by storm. It's not really apparent whether the book's success
can be attributed to <person>Morrison's</person> genius and command
over the XML technology, or just dumb luck. </p>

      <p>When asked about the sudden success of his book,
<person>Morrison</person> replied<1>I'm just a regular guy trying
to state the facts. </q>Regular or not, <person>Morrison</person>
will now be spending most of his time at the bank depositing
royalty checks. When asked what he plans to do with his newfound
riches, <person>Morrison</person> stated <1>I don't plan on
changing my lifestyle much, except for buying a new house, a half-
dozen cars, and a fat diamond for my wife, and constructing my own
extreme sports park on the 1,000 acre farm that I'm currently in
negotiations to buy. </q></p>
    </body.content>
  </body>
```

</nitf>

在这个代码中, location、org、person 和 object.title 元素用来更进一步的标识新闻故事并提供附加的元信息。虽然这些元素给了故事更多的细节, 但从标识的观点来说, 它起码还可以更进一步。如果你回忆一下, 你就可以想起 location 和 person 元素支持那些可以用来提供有关地点和人物的附加信息的子元素。此外, 你还可以使用 chron 元素来提供故事中相应文本的特定日期/时间信息。清单 31.4 包含了新闻故事的一个版本, 它使用了大量标识来描述非常详细的元信息。

清单 31.4 以大量标识编写的 XMLNews-Story 文件

```
<? xml version="1.0"? >
<nify>
  <head>
    <title>Nerd Author Strikes XML Gold</title>
  </head>
  <body>
    <body.head>
      <headline>
        <h1>nerd Author Strikes XML Gold</h1>
      </headline>
      <byline>
        <bytag>By Josh Timm</bytag>
      </byline>
      <dateline>
        <location><city>Nashville</city>,
          <state>Tennessee</state><? location>
        <story.date>Monday March 20 2000 9:43 ET</story.date>
      </dateline>
    </body.head>
    <body.content>
      <p>Acclaimed <person><function>nerd author</function>
        <name.given>Michael</name.given>
        <name.family>Morrison</name.family></person> strikes gold with his
        immensely popular book <object.title>XML Unleashed</object.title>.
        At approximately
        <chron norm="20000320T2055-0600">8:55 ET last night</chron>, the
        one millionth copy of <object.title>XML Unleashed</object.title>
        was sold at a <org>Barnes and Noble</org> bookstore in
        <location><sublocation>suburban</sublocation>
        <city>Nashville</city>, <state>Tennessee</state></location>, which
        makes the book the best-selling computer book of all time. The
        book explores the ins and outs of the XML information technology
        that has quickly taken the Web by storm. It's not really apparent
        whether the book's success can be attributed to
        <person><name.family>Morrison's</name.family></person> genius and
        command over the XML technology, or just dumb luck. </p>
      <p>When asked about the sudden success of his book,
        <person><name.family>Morrison</name.family></person> replied
```



```
<q>I'm just a <em>regular</em>guy trying to state the facts. </q>
Regular or not,
<person><name.family>Morrison</name.family></person> will now be
spending most of his time at the bank depositing royalty checks.
When asked what he plans to do with his newfound riches,
<person><name.family>Morrison</name.family></person> stated
<q>I don't plan on changing my lifestyle much,except for buying a
new house,a half-dozen cars,and a fat diamond for my wife,and
constructing my own extreme sports park on the 1,000 acre farm that
I'm currently in negotiations to buy. </q></p>
</body.content>
</body>
</niff>
```

具有如此大量标识的文件的一个缺陷就是 XMLNews-Story 代码难以阅读。但是,这个代码将永远不会被终端用户读到,因此这只在你手工编写新闻故事的时候才成为问题。实际上,大多数的内容开发人员将使用制作工具来通过其他方式创建 XMLNews-Story 文件。

注意:文字处理函数 Core Wordperfect 9 支持创建 XMLNews-Story 文件。你可以在 Corel 的 Web 站点(<http://www.corel.com>)上找到有关 Wordperfect 9 的更多信息。

31.4 总 结

XMLNews-Story 是一个用来编写在线发布的新闻故事的 XML 词表。XMLNews-Story 是否将作为在线新闻发布的结构化新闻格式标准这一点并不重要也并未明确,但它确实显示了大量承诺。XML 提供了完美的框架结构,因而在电子新闻发布领域对这种结构有着强大的需求。

第 32 章 使用 SMIL 集成多媒体对象

当前在 Web 上支持着大量的多媒体类型,包括图像、声音和视频。某些特定的多媒体实现可以享受本地的浏览器支持,而其他的类型需要插件和辅助应用程序。不论一个给定的媒体对象是如何被支持的,它都独立于其他的媒体对象显示或播放。W3C 考虑创建一个标准来综合媒体对象并提供一种方法来对它们同步化,这将非常有用。在 1998 年的夏季,W3C 发布了一个名为 SMIL(读作“smile”)的 XML 应用程序,它代表同步多媒体综合语言。在本书编写期间,有关 SMIL 的最新 W3C 版本是名为“Boston”的工作草案。它可在 <http://www.w3.org/TR/smil-boston> 上获得。

本章将探讨 SMIL 和它的功能。SMIL 使得使用大量的媒体类型创建丰富的、交互的、同步的多媒体表述成为可能。与它所提供的强大功能相比,SMIL 词表令人惊讶的贫乏。虽然 SMIL 是一个比较新的技术,但现在已经有了一些强力的工具可以用来编写和播放 SMIL 表述。

32.1 SMIL 基础

同步多媒体综合语言(SMIL—Synchronized Multimedia Integration Language)允许 Web 开发人员将多种类型的媒体放在一起并且实现同步。在 SMIL 以前,Web 已经有了大量的可支持的多媒体对象,诸如图片、声音和视频。然而,还没有一个上乘的方法将这些元素结合起来创造出一种丰富的多媒体。你可以使用视频编辑工具将文本和静态图片放入到视频剪辑中,但是这样需要大量的存储空间,这也将导致在 Web 上的巨大带宽需求。更上乘的解决方法就是同步化媒体,使之对带宽带来最小的冲击。

要更好的理解同步化媒体的重要意义,可考虑一下在电视上的篮球赛。这个游戏本身占据了屏幕的大部分可视空间。然而,屏幕的一个小区域——整个底部,是保留用来显示比赛分数的。你可以逻辑上将屏幕分为两个区域:比赛区和分数区。从 SMIL 角度来看,就是一个视频区域和一个文本区域。文本区域与比赛同步,因为分数在每个球得分的时候被刷新。因此这就叫“同步化媒体”!

另一个同步化媒体的优秀例子是在诸如 CNN 和 MSNBC 这样的新闻频道上显示的新闻和股票自动收发信息。同样,在一个区域使用视频,而在另一个区域使用文本,有时甚至在其他区域来做广告。这里屏幕实际可用的部分是很有限的,因此它要最大限度的显示信息。在我们所生活的信息社会中,这只是意味着要尽可能的将每一个波段填满。

你也许想知道为什么我还是用电视类比来解释 SMIL 的功能。事实是 SMIL 可能将电视类型的多媒体引入到 Web 上。不管你认为这件事情是好是坏,你不能怀疑电视的冲击力以及它的普及性并有着大量的观众。或许像 SMIL 这样的同步化媒体语言将完成这种技术融合,将两者融合为基于网络的电视类型播放。

谈到电视播放,几分钟前我将一个电视屏幕的物理部分称为区域。这并不是语误。区域

在 SMIL 中扮演着重要的角色。一个 SMIL 表述可以分为数个矩形区域,这些区域确定不同类型媒体的特定物理空间。有时 SMIL 区域指的是频道(这不要与电视的频道相混淆)。除了在 SMIL 表述中它与物理屏幕空间的一部分相关外,SMIL 频道工作的性质更像一个媒体收集器。SMIL 区域可以播放音频和视频,也可以显示图片和文本。图 32.1 显示了一个 SMIL 表述是如何分成多个区域的,且每个区域包含一种不同类型的媒体。

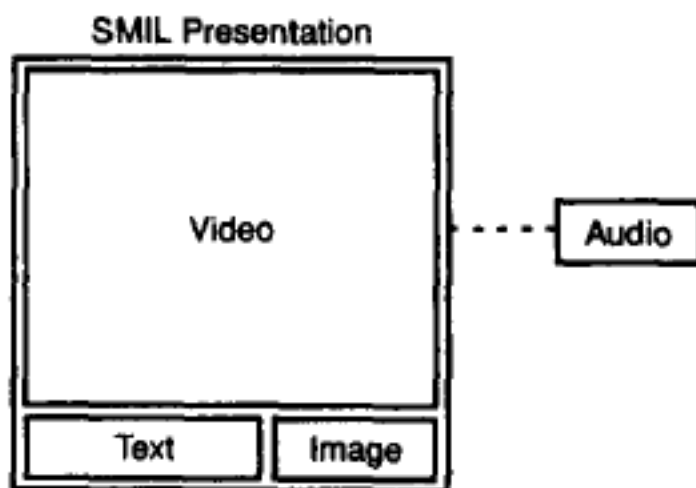


图 32.1 有关 SMIL 表述是如何分为多个区域的例子

注意音频是在一个表述中并不占据物理空间的区域中。你可以把音频区域看作是一个逻辑区域,这样有助于在没有给出任何物理空间的情况下辨别表述中的音频部分。这是一个说明为什么 SMIL 区域有时指的是频道的好例子。这里“频道”的意义与视频和音乐行业中的频道相同。

要谨记将媒体对象简单的分到不同的区域中并不是 SMIL 的主要目的(技术上,你已经可以使用 HTML 结合 Java applet 或插件来实现这一点)。SMIL 不仅规定了不同的媒体对象间的物理关系,它还提供了对它们同步化的方法。一个音频片断可以非常精确的定时,因此它可以在图片的连续显示中的适当位置播放。文本字幕也可以与视频或图片一致性的显示。SMIL 给了你精确控制媒体对象在播放或显示时定时的能力。同时由于 SMIL 是为促进 Web 上的媒体同步化而设计的,它只导致少量的带宽消耗。整个 SMIL 文件才有几十 K 字节。

注意:要谨记 SMIL 表述中的实际媒体对象是与 SMIL 文件分开存储的,因此他们还会导致巨大的带宽问题。你应该尽你所能的选择简洁的媒体格式并使 SMIL 表述中的媒体对象尽可能的“轻”。这样,低速 Internet 连接的用户就不会在等待你的表述装载的时候等的睡着了。

SMIL 的最后特征就是它的基于本地系统参数改变媒体内容传输的功能。例如,SMIL 可以量化的获得用户的链接速度并传输图像或声音片段的更小的版本,它甚至可以在 Internet 连接非常慢的情况下放弃部分媒体对象。这对于基于 Web 的多媒体技术来说是一个强大的并且非常必需的特性。

32.2 深入 SMIL 词表

就如你将在本章后面所学的,SMIL 可以用来创建包含大量同步化媒体类型的非常丰富的多媒体表述。要提供这样的功能和灵活性,你可能会认为 SMIL 词表可能是一个庞大

的、复杂的东西,有很多微小区别。然而,它实际上从结构和使用两方面都非常的简单。清单 32.1 包含了一个初步的 SMIL DTD。

清单 32.1 SMIL 文件类型定义(DTD)

```

<! -Generally usefaul entities -->
<! ENTITY % id-attr "id ID #IMPLIED">
<! ENTITY % title-attr "title CDATA #IMPLIED">
<! ENTITY % title-attr "skip-content (true|false) 'true'">
<! ENTITY % desc-attr "
    %title-attr;
    abstract          CDATA          #IMPLIED
    author            CDATA          #IMPLIED
    copyright         CDATA          #IMPLIED">
<! ELEMENT smil (head?,body?)>
<! ATTLIST smil %id-attr;>
<! ENTITY % layout-section 'ayout|switch">
<! ENTITY % head-element "(meta *,((%layout-section;),meta * ))?">
<! ELEMENT head %head-element;>
<! ATTLIST head %id-attr;>
<! ELEMENT layout ANY>
<! ATTLIST layout %id-attr;>
<! ELEMENT layout ANY>
<! ATTLIST layout %id-attr;
    type      CDATA      "text/smil-basic-layout">
<! ENTITY % viewport-attrs"
    height          CDATA          #IMPLIED
    width           CDATA          #IMPLIED
    background-color CDATA          #IMPLIED">
<! ELEMENT region EMPTY
<! ATTLIST region
    %id-attr;
    %title-attr;
    %viewport-attrs;
    left          CDATA          "0"
    top           CDATA          "0"
    z-index       CDATA          "0"
    fit           (hidden|fill|meet|scroll|slice)      "hidden"
    %skip-attr;>
<! ELEMENT root-layout EMPTY>
<! ATTLIST root-layout
    %id-attr;
    %title-attr;
    %viewport-attrs;
    %skip-attr;>
<! ELEMENT meta EMPTY.
<! ATTLIST meta
    name          NMTOKEN  #REQUIRED
    content       CDATA    #REQUIRED

```



```

%skip-attr; >

<! ENTITY % media-object "audio|video|text|img|animation|textstream|ref" >
<! ENTITY % schedule "par|seq|(%media-object;)" >
<! ENTITY % inline-link "a" >
<! ENTITY % assoc-link "anchor" >
<! ENTITY % link "%inline-link;" >
<! ENTITY % container-content "(%schedule;)|switch|(%link;)" >
<! ENTITY % body-content "(%container-content;)" >
<! ENTITY body (%body-content;)* >
<! ATTLIST body % id-attr; >

<! ENTITY % sync-attributes "
  begin      CDATA      #IMPLIED
  end        CDATA      #IMPLIED" >

<! ENTITY % system-attribute"
  system-bitrate      CDATA      #IMPLIED
  system-language     CDATA      #IMPLIED
  system-required     CDATA      #IMPLIED
  system-screen-size  CDATA      #IMPLIED
  system-screen-depth CDATA      #IMPLIED
  system-captions      (on|off)   #IMPLIED
  system-overdub-or-caption (caption|overdub) #IMPLIED" >

<! ENTITY % fill-attribute "
  fill (remove|freeze) 'remove'" >

<! ENTITY % par-content "%container-content;" >

<! ELEMENT par (%par-content;)* >
<! ATTLIST par
  %id-attr;
  %desc-attr;
  endsync CDATA      "last"
  dur      CDATA      #IMPLIED
  repeat   CDATA      "1"
  region   IDREF       #IMPLIED
  %sync-attributes;
  %system0attribute; ?

<! ENTITY % seq-content "%container-content;" >

<! ELEMENT seq (%seq-content;)* >
<! ATTLIST seq
  %id-attr;
  %desc-attr;
  dur      CDATA      #IMPLIED
  repeat   CDATA      "1"
  region   IDREF       #IMPLIED
  %sync-attributes;
  %system-attribute; >

<! ENTITY % switch-content "layout|(%container-content;)" >

<! ELEMENT switch(%switch-content;)* >
<! ATTLIST switch
  %id-attr;
  %title-attr; >

<! ENTITY % mo-attributes "

```

```

%id-attr;
%desc-attr;
region      IDREF      # @IMPLIED
alt         CDATA      # IMPLIED
longdesc    CDATA      # IMPLIED
src         CDATA      # IMPLIED
type        CDATA      # IMPLIED
dur         CDATA      # IMPLIED
repeat      CDATA      ' 1'
%fill-attribute;
%sync-attributes;
%system-attribute;">

<! ENTITY % mo-content "(% assoc-link;)*">
<! ENTITY % clip-attrs "
clip-begin  CDATA      # IMPLIED
clip-end    CDATA      # IMPLIED">

<! ELEMENT ref      %mo-content;>
<! ATTLIST ref      %mo-attributes;%clip-attrs;>

<! ELEMENT audio     %mo-content;>
<! ATTLIST audio     %mo-attributes;%clip-attrs;>

<! ELEMENT img       %mo-content;>
<! ATTLIST img       %mo-attributes;>

<! ELEMENT video     %mo-content;>
<! ATTLIST video     %mo-attributes;%clip-attrs;>

<! ELEMENT text      %mo-content;>
<! ATTLIST text      %mo-attributes;>

<! ELEMENT textstream %mo-content;>
<! ATTLIST textstream %mo-attributes;%clip-attrs;>

<! ELEMENT animation %mo-content;>
<! ATTLIST animation %mo-attributes;%clip-attrs;>

<! ENTITY % smil-link-attributes "
%id-attr;
%title-attr;
href      CDATA      # REQUIRED
show      (replace|new|pause) 'replace'">

<! ELEMENT a (%schedule;|switch)*>
<! ATTLIST a
%smil-link-attributes;>

<! ELEMENT anchor EMPTY>
<! ATTLIST anchor
%skip-attr;
%smil-link-attributes;
%sync-attributes;
cords      CDATA      # IMPLIED>

```

考虑到 SMIL 可以支持几乎所有当前使用的媒体类型,这并不是一个很庞大的 DTD。SMIL 如此简单的理由之一就是它使用了类似的方法来确定所有媒体对象的时间,而不考虑它们的类型。SMIL 词表基于三个基本元素:

- **smil** ——这是 SMIL 文件的根元素,或文件元素。因此,你总要将所有的 SMIL 内容包括在<smil>和</smil>标记之间。
- **head** ——它与 HTML 中的 head 元素工作原理类似,提供一个方便空间来定义应用与整个文件的头信息。
- **body** ——它包含所有有关媒体表述的专门信息。在 SMIL 文件的主体部分中,你可以列出不同的媒体对象以及它们显示和播放的特定时间。

让我们逐个对这些基本 SMIL 元素进行更详细的讨论。在本章后面,你将使用这些元素从头开始建立一个 SMIL 表述。

32.2.1 smil 元素

smil 元素是 SMIL 文件的文件元素,它作为其他两个基本元素——head 和 body——的容器元素。一个 SMIL 文件的基本框架结构如下:

```
<smil>
  <head>
  </head>

  <body>
  </body>
</smil>
```

head 和 body 元素包含在 smil 元素中。区域和媒体对象在 head 和 body 元素中被定义,这一点你将在下一部分中看到。

32.2.2 head 元素

head 元素提供了描述 SMIL 表述的整体结构的场所。这个元素依赖一个单独的子元素 layout 来完成这个任务。Layout 元素是一个容器元素,在其中你可以放置有关描述的有着多个区域的根布局。

表述的根布局是不考虑表述的物理屏幕真实状态。你使用 root-layout 元素来描述根布局,这是一个空元素,它包含着一些确定根布局性质的属性:width、height 和 background-color。下面是一个设置表述的根布局的宽、高和背景色的例子:

```
<root-layout width="400" height="300" background-color="white"/>
```

除了描述根布局外,layout 元素可以是表述中区域的容器(就像曾提到的那样,区域是表述根布局中的一个矩形区域,它拥有某种类型的媒体对象)。区域主要通过宽、高以及到根布局的左上角的 XY 坐标进行描述。下面是一个使用这些属性创建区域的例子:

```
<region id="slide" title="slide" left="0" top="0" width="400" height="250">
```

id 属性用来当在 SMIL 文件的主体部分指定一个实际的媒体对象时引用这个区域。Title 属性用来给这个区域一个简单易懂的名字,这在你使用可视化 SMIL 制作工具的时候非常的重要。

虽然 layout 元素是被定义用来容纳各种类型内容的,但你仍必须将 root-layout 和

region 元素作为 layout 元素的子元素使用。因此,layout 元素本身必须出现在 head 元素中。下面是一个演示这些元素是如何相互协调共同组成 SMIL 文件头的例子:

```
<head>
  <layout>
    <root-layout width="400" height="300" background-color="white"/>
    <region id="slide" title="slide" left="0" top="0" width="400" height="250"/>
    <region id="caption" title="caption" left="0" top="251" width="400" height="50"/>
    <region id="caption-audio" title="caption-audio"/>
  </layout>
</head>
```

我使用了更多的区域来说明多个区域是如何使用 head、layout 和 region 元素在 SMIL 表述中定义的。要注意最后一个区域 caption-audio,它不包含 left、top、width 或 height 属性。这是因为这是一个只用于播放音频片段的逻辑区域,这将在本章的后面看到。

32.2.3 body 元素

body 元素是用来定义 SMIL 表述的媒体内容的主要元素。SMIL 文件的头部分包含了有关表述的有用结构信息,而主体部分是详细确定媒体时间的地方。Body 元素支持一些不同的子元素,这样你可以使用来安置和同步媒体对象。下面是最常用的子元素:

- text
- img
- audio
- video
- par
- seq

下面的几节将逐个探讨这些元素。

text 元素

text 元素用来显示文本媒体对象,这是一个从文本文件中读取的文本字符串并在指定的时间内在区域中被显示。text 元素最常用的属性是 region、src、type 和 dur。

region 属性使用在文件头部分定义的区域 id,文本将在这个区域中被显示。

src 属性是包含要显示文本的文本文件的 URL。

type 属性是 MIME 类型描述,它确定媒体对象的类型。对于简单文本,MIME 类型是 text/plain。虽然并不总需要明确的声明媒体对象的类型,我想把这样作为一个习惯还是很好的。

注意: MIME 就是多用途网际邮件扩充协议 (Multipurpose Internet Mail Extension),它是一个用来指定如何格式化信息以利用邮件系统交换的 Internet 规范。MIME 在电子邮件领域之外也很有用,它用来描述丰富的媒体对象类型,比如图片、音频和视频。指定媒体格式为相应的 MIME 媒体类型将使它们容易被识别。一个这样的 MIME 类型的例子是 image/gif,它确定 GIF 图片格式。

text 元素 dur 属性用来描述持续时间。这个属性用来指定文本显示的时间。下面是一个如何使用 text 元素的 dur 属性显示 text 媒体对象的例子:

```
<text region="caption" src="Media/Pond01.txt" type="text/plain" dur="12s"/>
```

在这个例子中, text 包含在 Pond01.txt 中, 在表述中 caption 区域里显示 12 秒。让媒体对象显示指定长度的时间并不是一件很困难的事情。

注意: 有时有必要在 SMIL 表述中引用媒体对象, 这是你将希望使用 id 属性来给出媒体对象惟一的标识。

img 元素

img 元素用来显示静态图片。与 text 元素相同, img 元素用 region、src、type 和 dur 属性来确定一个图片显示的效果。在 SMIL 表述中最常用的图片格式是 GIF 和 JPEG, 它们的 MIME 类型分别是 image/gif 和 image/jpeg。下面是使用 img 元素显示 JPEG 图片的例子:

```

```

在这个例子中, Pond01.jpg 图片在表述的 slide 区域中显示 12 秒。

audio 元素

audio 元素用来播放音频片断。与 text 和 img 元素相同, audio 元素依赖 region、src、type 和 dur 属性来确定音频片断播放的参数。与图片不同的是, 图片由 Web 规范(GIF 和 JPEG)来管理使用的类型, 而音频媒体类型变化很大。基于这样的原因, 音频媒体的类型取决于你所使用的特定 SMIL 播放器。一个常用的用于 Java applet 的音频格式是 AU 格式, 这就是在本章后面你创建完整的 SMIL 表述时所用的音频格式。AU 音频格式的 MIME 类型是 audio/basic。下面是一个使用 audio 元素播放 AU 音频片断的例子:

```
<audio region="caption_audio" src="Media/Pond01.au" type="audio/basic" dur="6.20s"/>
```

在这个例子中, Pond01.au 音频片断在 caption_audio 区域中播放 6.20 秒, 这正好是这个片断的准确长度。

video 元素

video 元素用来播放 video 片断。同样, video 元素依赖 region、src、type 和 dur 参数定义视频片断播放的参数。与音频片段的类型相似, 现在有很多的视频媒体类型。因此, 可用的视频媒体类型依赖于你所使用的特定 SMIL 播放器。

video 和 audio 元素描述基于时间的媒体对象, 而 text 和 image 元素并不这样。由于这个原因, video 和 audio 元素支持 begin 和 end 属性, 它们可以作为 dur 属性的一种替代。通过使用 begin 和 end 属性, 除了可以确定持续时间以外, 你还可以更精确的控制视频或音频片断的开始和完成时间。

par 元素

在这里, 你懂得了如何在 SMIL 表述的区域中描述媒体对象以及如何确定它们的持续时间。虽然媒体对象的持续时间确实非常重要, 但它并不真的影响媒体对象的同步性。par 元素就是在 SMIL 中用来协助媒体同步化的两个元素之一。

注意:虽然文本和静态图片是显示而不是播放,但在这一次讨论中,我对所有的媒体对象都说成“播放”以将问题简单化。

par 元素是一个容器元素,在其中你可以放置你希望相互间可以并行播放的媒体对象(换句话说,就是同时播放)。因此,如果你将一个 img 元素和一个 audio 元素放在一个 par 元素中,图片将在音频文件开始播放的时候显示。此外,图片将在音频文件开始播放的同一时刻显现出来。下面是一个使用 par 元素使三个媒体对象同步开始的例子:

```
<par id="slide01">
  
  <text region="caption" src="Media/Pond01.txt" type="text/plain" dur="12s"/>
  <audio region="caption_audio" src="Media/Pond01.au"
    type="audio/basic" dur="6.20s"/>
</par>
```

在这个例子中,par 元素用来包含三个媒体对象:一幅图片、一个文本字符串和一个音频片段。这个 SMIL 表述在显示图片和文本的同时播放音频片段。注意音频片段的持续时间比图片和文本的持续时间短,这意味着音频文件结束播放的时候,图片和文本依旧处于显示状态。你可以指定音频片段一个更长的持续时间,音频片段可以循环播放来填充多余的时间。

seq 元素

与 par 元素相对应的是 seq 元素,它用来连续的播放媒体对象。与 par 可同时播放多个媒体对象相反,seq 元素将它们一个接一个的播放。因此,如果你对 par 例子中的媒体元素使用 seq 元素,则它们将一个接一个的播放,而不是同时播放。

```
<seq>
  
  <text region="caption" src="Media/Pond01.txt" type="text/plain" dur="12s"/>
  <audio region="caption_audio" src="Media/Pond01.au"
    type="audio/basic" dur="6.20s"/>
</seq>
```

在这个例子中,Pond01.jpg 图片显示 12 秒,Pond01.txt 中的文本字符串显示 12 秒,而 Pond01.au 音频片段播放 6.20 秒。很明显,在这个例子中,seq 元素工作的并不很好。一般在 SMIL 表述中都是混合的使用 par 和 seq 元素,以获得所期望的效果。下面是一个使用这些元素显示两个幻灯片段的 SMIL 代码。

```
<seq id="slides">
  <par id="slide01">
    
    <text region="caption" src="Media/Pond01.txt" type="text/plain" dur="12s"/>
    <audio region="caption_audio" src="Media/Pond01.au"
      type="audio/basic" dur="6.20s"/>
  </par>
```

```

<par id="slide02">
  
  <text region="caption" src="Media/Pond02.txt" type="text/plain" dur="12s"/>
  <audio region="caption_audio" src="Media/Pond02.au"
    type="audio/basic" dur="5.42s"/>
</par>
</seq>

```

在这个例子中,代码的两个 par 小节放置在 seq 元素中,这意味着这些 par 小节将分别播放。然而,每个 par 元素中的媒体元素将并行播放。结果就是,第一个图片/文本/音频混合体被播放,然后经过 12 秒,第二个图片/文本/音频混合体被播放。这段代码实际上是从你将在下一节中创建的简单 SMIL 表述中摘录出来的。

32.3 创建 SMIL 内容

有了对 SMIL 词表的初步认识,现在是用它们工作并创建一个完整的 SMIL 表述的时候了。然而,在你要开始式作之前,我希望给你指出创建 SMIL 表述时的三个主要任务:

- 在 SMIL 文件的头部分创建媒体对象所用的区域。
- 在文件的主体部分创建媒体对象。
- 使用<par>和<seq>标记确定播放的次序和媒体对象的定时。

你已经学习了这些步骤中所包括的多种 SMIL 标记(元素)和属性。实际上,你在前面探讨 SMIL 词表时候所看到所有的代码片段都是一个完整 SMIL 文件的一部分。这个文件 pond.smil 是一个记录我的 koi 养鱼池的构造的 SMIL 幻灯片。清单 32.2 包含用于 Pond.smil 的幻灯显示的 SMIL 代码。完整的养鱼池 SMIL 例子可以在本书附带的 CD-ROM 中获得——好好试试吧!

注意: koi 是一种观赏性日本鲤鱼,它有着不寻常的颜色并且可以长到三英尺长。koi 可以生活 60 年,甚至更长,它们非常的昂贵,因为它们在日本具有收藏价值而在美国分布很少。一个 koi 养鱼池是专门作为 koi 的家而设计的,它通常有三英尺深。你可以在我的 Web 站点 <http://www.thetribe.com> 中获得更多有关我的 koi 养鱼池的信息。

清单 32.2 幻灯显示文件 Pond.smil 的 SMIL 代码

```

<smil>
  <head>
    <layout>
      <root-layout width="400" height="300"
        background-color="white"/>
      <region id="slide" title="slide" left="0" top="0"
        width="400" height="250"/>
      <region id="caption" title="caption" left="0" top="251"
        width="400" height="50"/>
    
```

```

    <region id="caption-audio" title="caption-audio"/>
    <region id="background-audio" title="background-audio"/>
  </layout>
</head>
<body>
  <par id="slideshow">
    <audio src="Media/Music.au" region="background-audio"
      type="audio/basic" dur="137.50s"/>
    <seq id="slides">
      <par id="intro">
        
      </par>
      <par id="slide01">
        
        <text region="caption" src="Media/Pond01.txt" type="text/plain"
          dur="12s"/>
        <audio region="caption-audio" src="Media/Pond01.au"
          type="audio/basic" dur="6.20s"/>
      </par>
      <par id="slide02">
        
        <text region="caption" src="Media/Pond02.txt" type="text/plain"
          dur="12s"/>
        <audio region="caption-audio" src="Media/Pond02.au"
          type="audio/basic" dur="5.42s"/>
      </par>
      <par id="slide03">
        
        <audio region="caption" src="Media/Pond03.txt" type="text/plain"
          dur="12s"/>
        <audio region="caption-audio" src="Media/Pond03.au"
          type="audio/basic" dur="4.26s"/>
      </par>
      <par id="slide04">
        
        <text region="caption" src="Media/Pond04.txt" type="text/plain"
          dur="12s"/>
        <audio region="caption-audio" src="Media/Pond04.au"
          type="audio/basic" dur="6.88s"/>
      </par>
      <par id="slide05">
        
        <text region="caption" src="Media/Pond05.txt" type="text/plain"
          dur="12s"/>
        <audio region="caption-audio" src="Media/Pond05.au"
          type="audio/basic" dur="5.77s"/>
      </par>
      <par id="slide06">

```



```


<text region="caption" src="Media/Pond06.txt" type="text/plain"
  dur="12s"/>
<audio region="caption-audio" src="Media/Pond06.au"
  type="audio/basic" dur="5.44s"/>
</par>
<par id="slide07">
  
  <text region="caption" src="Media/Pond07.txt" type="text/plain"
    dur="12s"/>
  <audio region="caption-audio" src="Media/Pond07.au"
    type="audio/basic" dur="4.47s"/>
</par>
<par id="slide08">
  
  <text region="caption" src="Media/Pond08.txt" type="text/plain"
    dur="12s"/>
  <audio region="caption-audio" src="Media/Pond08.au"
    type="audio/basic" dur="4.47s"/>
</par>
<par id="slide09">
  
  <text region="caption" src="Media/Pond09.txt" type="text/plain"
    dur="12s"/>
  <audio region="caption-audio" src="Media/Pond09.au"
    type="audio/basic" dur="4.66s"/>
</par>
<par id="slide10">
  
  <text region="caption" src="Media/Pond10.txt" type="text/plain"
    dur="12s"/>
  <audio region="caption-audio" src="Media/Pond10.au"
    type="audio/basic" dur="2.60s"/>
</par>
<par id="end">
  
</par>
</seq>
</par>
</body>
</smil>

```

这个 SMIL 文件提供了有 10 个幻灯片的幻灯显示,有介绍和总结。它使用图片、文本、音频叙述和背景音乐完成。每个幻灯片包含一幅图片、一段文字和一个描述图片的音频片段,所有的这些都并行的显示。而在幻灯片之间是一个接一个的分别显示。

我希望直接在网页上显示这个池塘幻灯片,因此我使用支持 SMIL 的 SOJA 播放器,这

是一个用 Java applet 实现的 SMIL 播放器。清单 32.3 包含使用 Java applet 程序 SOJA 播放 SMIL 表述 Pond.smil 的 HTML 文件。

清单 32.3 幻灯片播放网页 Pond.html 的 HTML 代码

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
  </head>
  <body>
    <applet code="org.helio.soja.SojaApplet.class" archive="soja.jar"
      width="400" height="300">
      <param name="source" value="Pond.smil">
      <param name="display" value="applet">
      <param name="userpreferences" value="false">
    </applet>
  </body>
</html>
```

SOJA 播放器被描述为一个包含所有的用来工作的相关 Java 类的存档 JAR 文件。实际的 applet 类名为 SojaApplet 并在 <applet> 标记的 code 属性中被引用。Soja.jar 文件在 archive 属性中被引用。最后, SMIL 文件在 <applet> 标记的 source 参数中指定。图 32.2 到 32.4 显示了使用 Pond.html 网页和 SOJA 播放器播放的养鱼池幻灯片(顺便说一下,你将在下一节中学到更多有关 SOJA 播放器的知识)。

32.4 SMIL 播放器和制作工具

与所有的 XML 内容相同,如果没有各种类型的软件来浏览, SMIL 文件什么用处也没有。同时,使用可视化的制作工具非常有助于创建复杂的 SMIL 文件,而不需要手工的编纂它们。下面的几节将探讨一些本书编写期间可用的 SMIL 浏览器和制作工具。

32.4.1 SMIL 播放器

与创建 SMIL 文件同样有趣,直到你在 SMIL 播放器上看到了完成的 SMIL 文件,真正的乐趣才刚刚开始。SMIL 播放器是一种软件,它处理 SMIL 文件并显示最终的多媒体表述。在本书编写期间有三种主要的 SMIL 播放器:

- SOJA
- GRiNS SMIL 播放器
- RealPlayer G2

在下面的几节将对这些 SMIL 播放器逐一进行讲述。

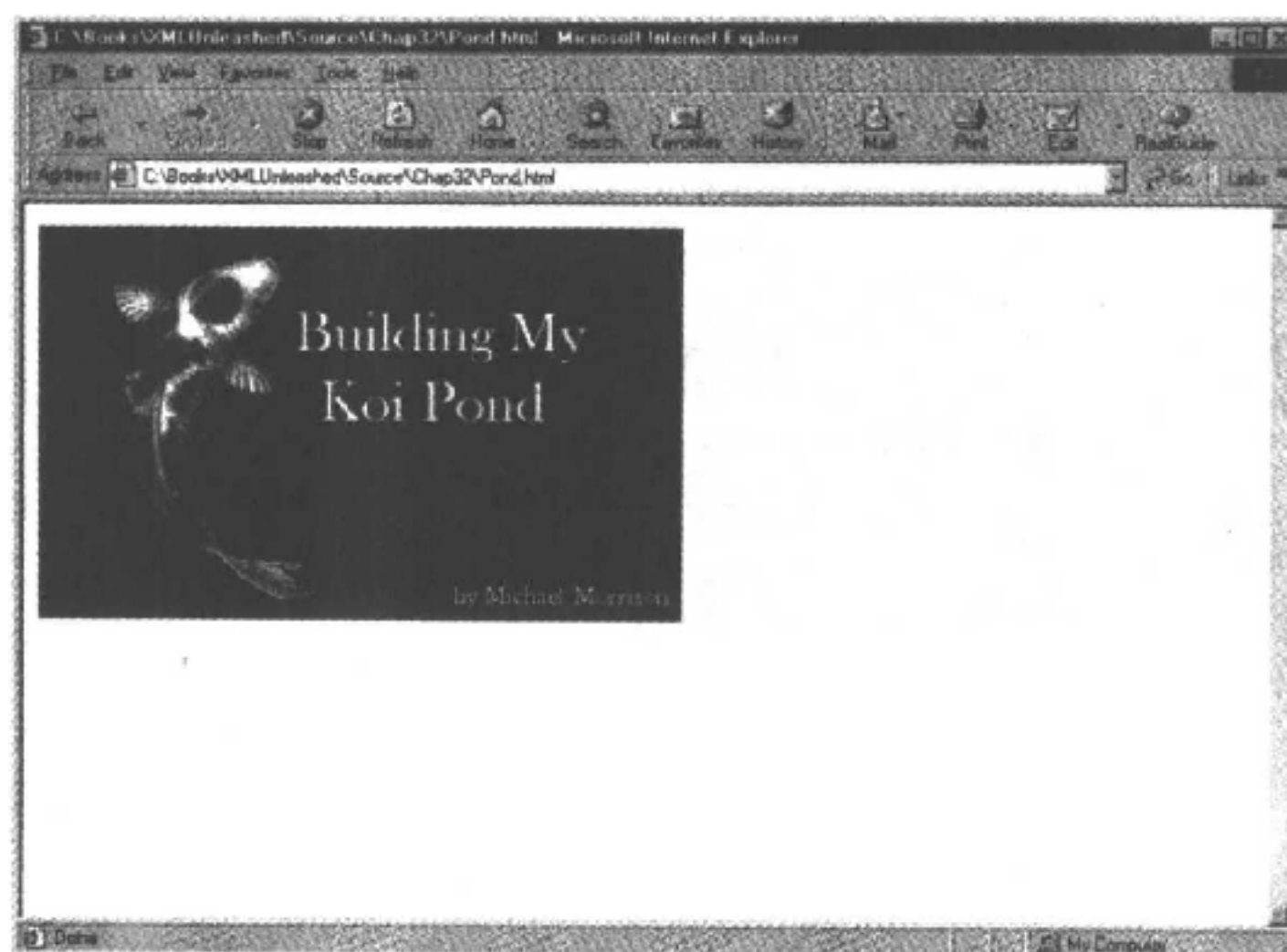


图 32.2 养鱼池幻灯片显示的导言,它通过 SOJA 播放器显示

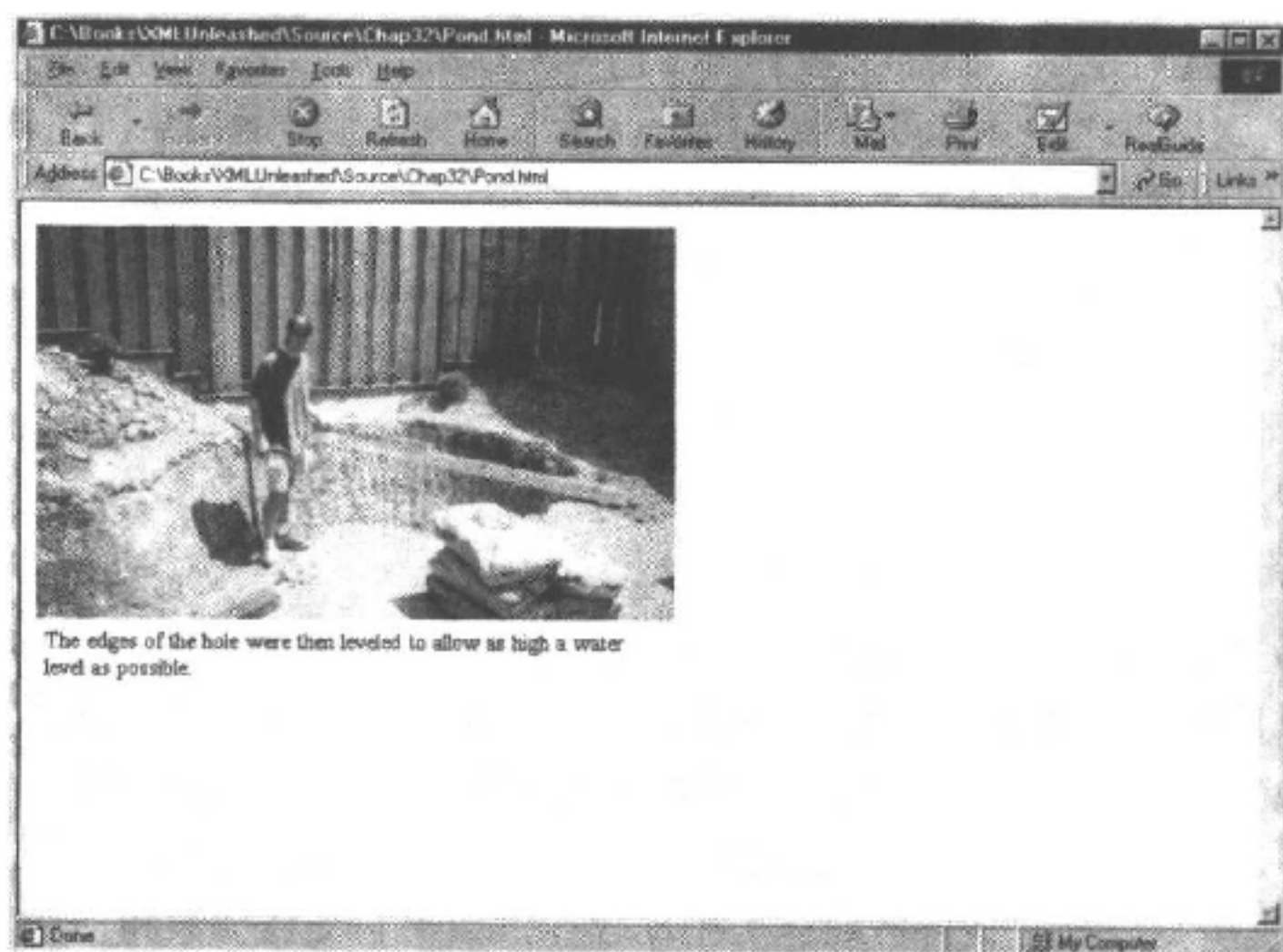


图 32.3 养鱼池幻灯片显示的建筑幻灯片,它通过 SOJA 播放器显示

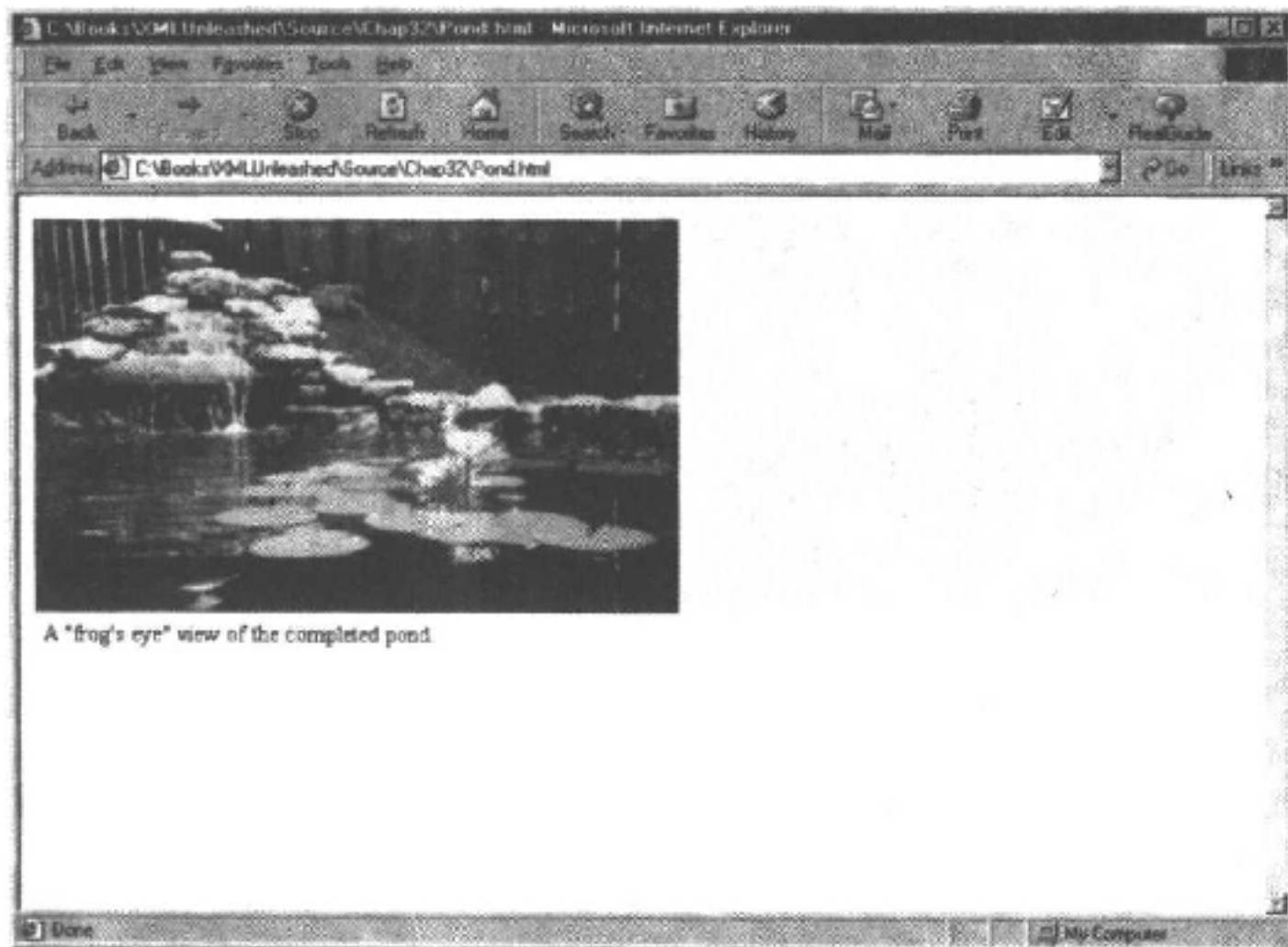


图 32.4 养鱼池幻灯片显示的完成的养鱼池,它通过 SOJA 播放器显示

SOJA

SOJA 是一个由 HELOI 出品的基于 Java 的 SMIL 播放器。它以 Java 1.1 applet 形式实现,它可以嵌入到网页之中并且可以被支持 Java 1.1 或更高版本的网络浏览器使用。SOJA 支持 SMIL 文件中的如下媒体对象:

- 简单文本(text/plain)
- GIF 图片(image/gif)
- JPEG 图片(image/jpeg)
- AU 音频片段(audio/basic)
- 压缩格式 AU 音频片段(audio/x-aux)

我发现 SOJA 提供了一个 SMIL 的稳定并且非常可用的版本,因此我使用它作为开发养鱼池幻灯片播放表述的基本浏览器。SOJA 的一个很大的好处就是它是以 Java applet 方式实现的,这就意味着使用者不需要下载任何其他特殊插件来浏览 SMIL 表述。这也同样意味着 SOJA 是与平台独立的,起码感觉上它可以运行在所有的支持 Java 1.1 的平台上。

SOJA 的惟一局限是它支持的媒体类型比较少。如果它支持视频和一些其他的音频格式,它就更完美了。然而,由于 Java 2 当前提供丰富的媒体支持,可以想像未来的 Java 2 版本 SOJA 将支持更大范围的媒体类型。虽然它对媒体的支持有限,但我还是强烈推荐 SOJA 作为体验 SMIL 的起始点。要获得有关 SOJA 的更多信息,或免费下载这个软件,请访问 SOJA 的 Web 站点 <http://www.helio.org/products/smil>。

GRiNS SMIL 播放器

GRiNS 播放器是 Oratrix Development 开发的 SMIL 播放器。它可用于 Windows、

Macintosh 和 UNIX 平台。与 SOJA 不同,GRiNS 播放器作为独立运行应用程序实现。不幸的是,这意味着 GRiNS 不能作为 SMIL 表述的 Web 配置解决方案的一部分。公正的来说,GRiNS 播放器是庞大的 GRiNS SMIL 编辑器的播放部分,这就使它更像是 SMIL 开发工具,而不是 SMIL 传输工具。

GRiNS 表现优秀的一个地方就是对不同媒体类型的支持。下面的不同类型的媒体对象都可被 GRiNS 播放器支持:

- 简单文本(text/plain)
- HTML 文本(text/html)
- AU 音频片段(audio/basic)
- AIFF 音频片段(audio/x-aiff)
- Wave 音频片段(audio/x-wav)
- JPEG 图片(image/jpeg)
- GIF 图片(image/gif)
- PNG 图片(image/png)
- TIFF 图片(image/tiff)
- RGB 图片(image/x-rgb)
- Windows 点阵图片
- MPEG 视频片段(video/mpeg)

GRiNS 播放器是测试 SMIL 表述的好方法。它独立运行,这在你不希望在网络浏览器中测试表述的情况下可以更好的工作。要获得有关 GRiNS 播放器的更多信息包括免费的下载,请访问 GRiNS 的 Web 站点 <http://www.oratrix.com/GRiNS>。

RealPlayer G2

如果只是将 RealNetworks 出品的 RealPlayer 作为 SMIL 播放器确实是有点不公平,因为它实在是功能非常多。然而对于本次讨论来说,RealPlayer G2 是一个 SMIL 浏览器的充分实现,它支持大范围的媒体类型。这些媒体类型中最有趣的部分就是那些 RealNetworks 自己创建的类型。我所指的就是 RealText、RealPix、RealAudio 和 RealVideo。所有的这些类型取代了那些在 SMIL 表述中使用的基本媒体类型。

RealPlayer G2 以浏览器插件的形式实现,它是为流式媒体设计的。所有的 RealNetworks 扩展的媒体类型在使用 RealPlayer G2 播放的时候都是流式的,这与常见的 GIF 和 JPEG 图片类型相同。RealPlayer G2 可以用来播放包含标准媒体类型的 SMIL 表述,这个播放器的优势是 RealNetworks 媒体类型的强大功能。例如,RealPix 支持所有类型的平滑图像效果,诸如淡化和擦去。这要使用 RealPix 图片来替代静态图片就可以梦幻般的效果。RealText 提供对 3D 文字的支持,以及许多其他的优雅的字体的效果。当然,使用者必须安装 RealPlayer G2 插件以获得所有的这些特性。

即使当你使用 RealNetworks 扩展的媒体类型时,所有的 RealPlayer G2 表述也通过使用 SMIL 集合在一起。实际上,由于扩展的媒体类型已经做了大量的工作,你会发现 RealPlayer G2 的 SMIL 表述在结构上通常要比依赖于标准媒体类型的 SMIL 表述简单。

注意:你也许惊讶的发现 RealNetworks 扩展的媒体类型在使用 XML 上走了与

SMIL 决不相容的途径。例如,RealPix 实际是一个简单的 XML 词表,用来创建图像效果。我将不再继续深入讲解 RealNetworks 的媒体类型,它们本身就可以非常轻易的写出十几章来。

如果你在创建 SMIL 表述上有兴趣或者希望在 Web 站点上加入流式媒体,我强烈推荐你使用 RealPlayer G2 和它的相关技术。顺便请访问一下 <http://www.real.com/g2>。

32.4.2 SMIL 制作工具

应当承认,对这方面我有一点陌生,我喜欢手工编写 SMIL 文件。然而,我也承认手工编写一个复杂的 SMIL 文件将是非常痛苦的。因此,你也许希望考虑使用可视化的 SMIL 制作工具创建 SMIL 表述。下面是本书编写期间一些可用的 SMIL 制作工具:

- GRiNS SMIL 编辑器
- Sausage SMIL 设计器
- RealProducer G2 和 RealSlideshow

下面的几节将描述这些 SMIL 制作工具。除了这些 SMIL 专用制作工具,Allaire 的 HomeSite 4.0 Web 开发工具也支持 SMIL。要获得有关 HomeSite 4.0 的信息,请访问 <http://www.allaire.com/Products/HomeSite>。同时,要谨记你可以使用任何 XML 编辑器编写 SMIL 文件,因为 SMIL 就是 XML 文件。

GRiNS SMIL 编辑器

GRiNS 播放器的制作者 Oratrix 出品了 GRiNSSMIL 编辑器。GRiNS 编辑器是一个可视化 SMIL 文件制作工具,它使用有趣的用户接口表现 SMIL 文件的结构。图 32.5 显示了 Pond.smil 幻灯片播放文件在 GRiNS 编辑器中的状态。

GRiNS 编辑器通过多种显示方式给你对 SMIL 表述的不同视点。在这里,结构化和时序浏览被分别用于养鱼池表述显示中。我更喜欢时序浏览,因为它准确的显示了在一定时刻播放的媒体对象。换句话说,时序浏览提供了对表述当时播放的情况的视角。对于养鱼池幻灯片展示来说,这导致对每个有着相关时间信息的连续画面的显示,这些时间信息就是每个画面的持续时间。GRiNS SMIL 编辑器还结合 GRiNS 播放器作为一个一体化的工具。这样你就可以依照你对他们的处理播放 SMIL 表述。图 32.6 显示了养鱼池表述在 GRiNS 播放器中的播放情况。

Sausage SMIL 设计器

Sausage Software 的 SMIL 设计器是另一个倾向于 RealNetworks 的扩展媒体类型的 SMIL 制作工具。SMIL 设计器依赖于一个有着多个对 SMIL 表述的不同浏览方式的可视化接口。与其他你所知道的编辑工具不同,SMIL 设计器除了时序浏览外还使用一个序列浏览来显示表述中媒体对象的次序。图 32.7 显示了在 SMIL 设计器中的养鱼池 SMIL 文件。

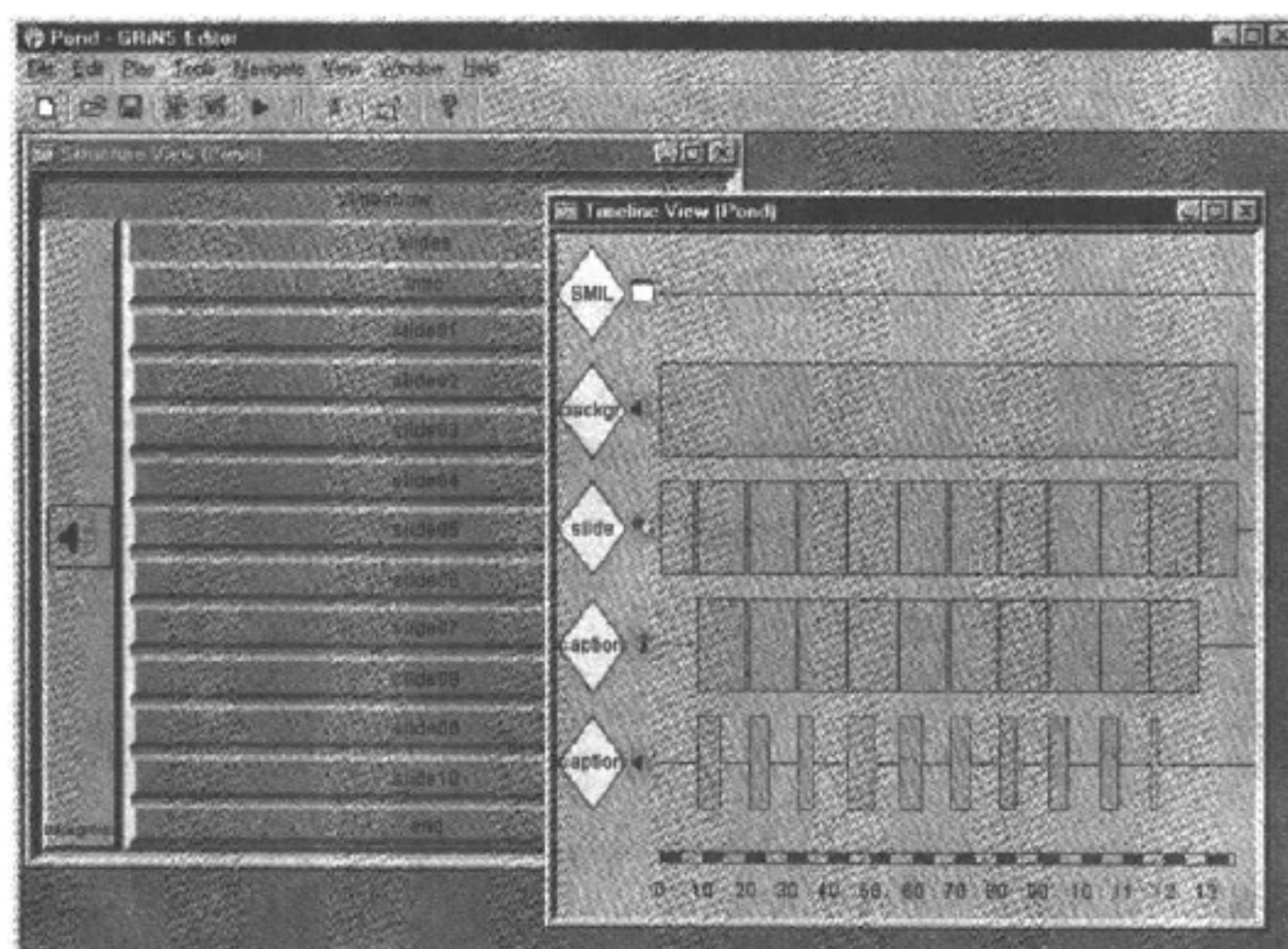


图 32.5 在 GRiNssmil 编辑器中显示的养鱼池幻灯片播放文件

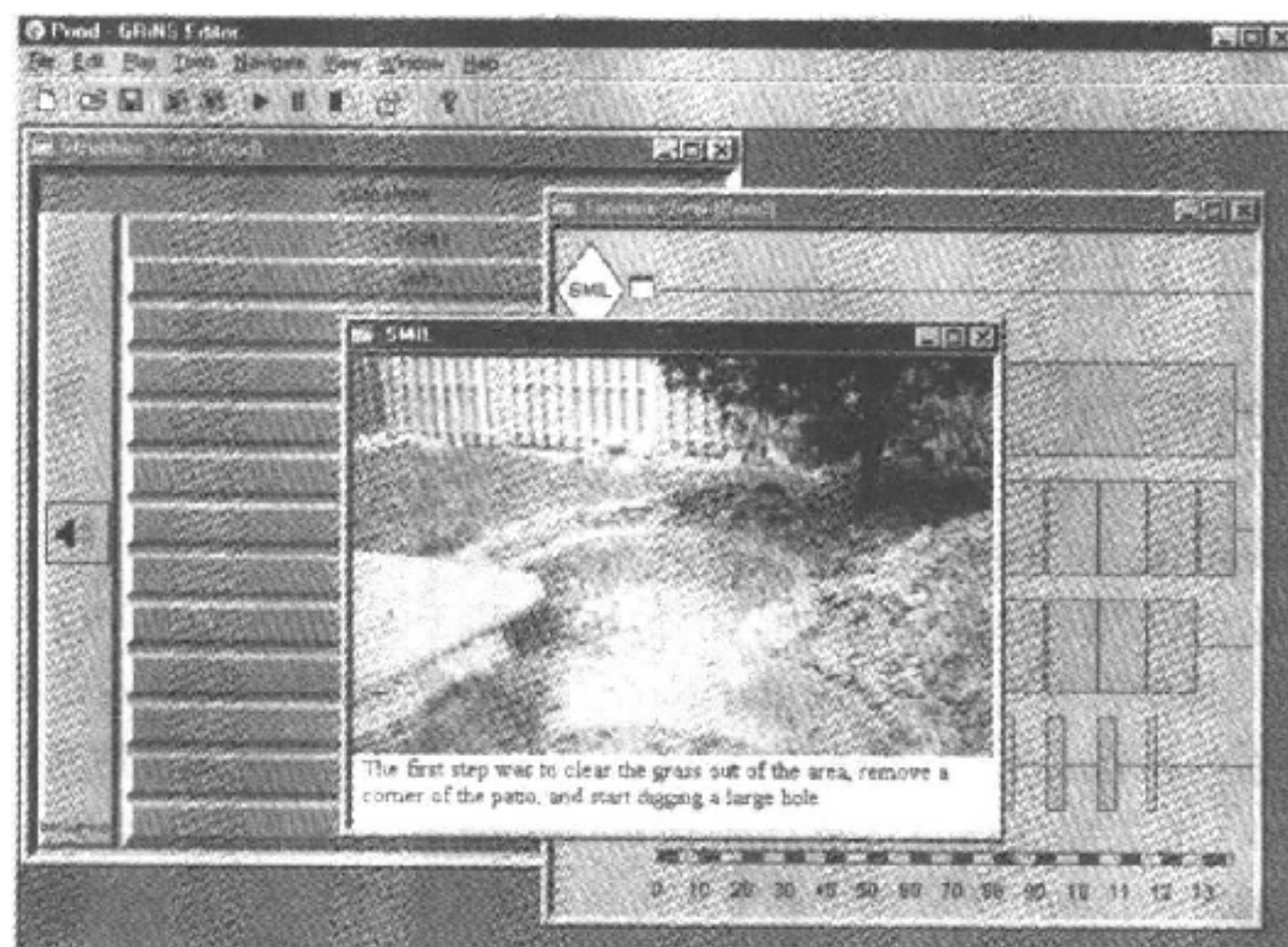


图 32.6 在 GRiNS 播放器中显示的养鱼池幻灯片播放文件

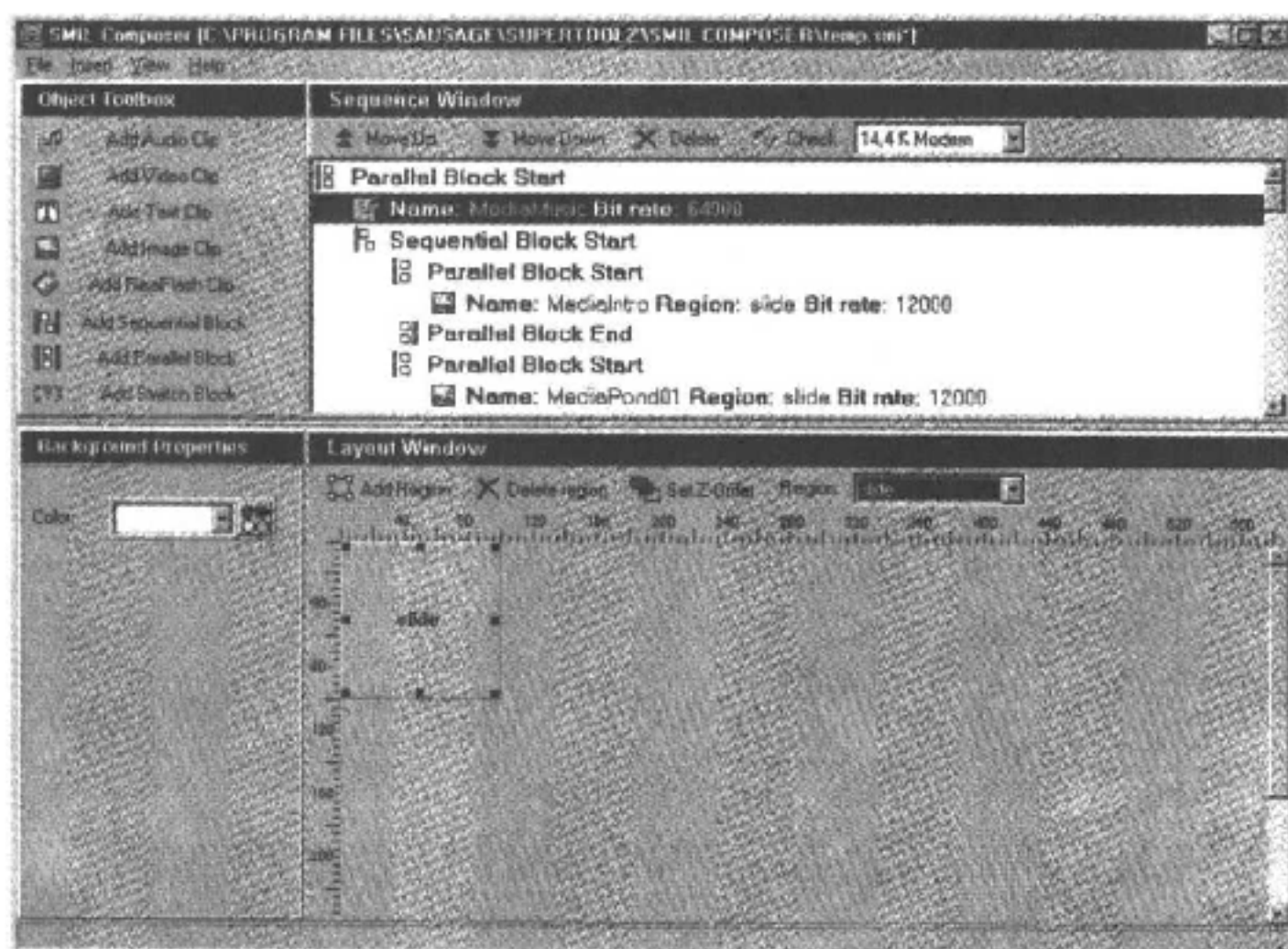


图 32.7 在 Sausage Smil 设计器制作工具中显示的养鱼池幻灯片播放文件

对外形的仔细观察会发现一些有关 SMIL 设计器的有趣事情。首先,它并没有确实打开手工编写的 SMIL 文件,只是内部导入了它们并通过添加修改标记和属性来改变它们。如果你像我一样,希望接触基本的 SMIL 代码。那么你可能并不喜欢 SMIL 设计器的这种外貌。换句话说,如果你希望导入 SMIL 文件,并且从此就使用 SMIL 设计器来编辑它。那么 SMIL 设计器正是适合你的工具。

要获得更多有关 SMIL 设计器制作工具的信息,包括获得免费测试版本,请访问 Sausage Software 的 SMIL 设计器页面 <http://www.sausage.com/supertoolz/toolz.stsmil.html>。

RealProducer G2 和 RealSlideshow

毫无疑问,RealNetworks 有着非常强大的用于 SMIL 的制作工具集。然而,RealNetworks 工具很大程度上依赖于 RealNetworks 的扩展媒体类型(RealText、RealPix 等等),因此如果你正在尝试使用这样的媒体类型,这些工具可能并不是你的最佳选择。换句话说,RealNetworks 的扩展媒体类型要比正规媒体类型更加的灵活并且可以提供的更多。RealNetworks 的 SMIL 制作工具集主要由下面的两个工具组成:

- RealProducer G2。
- RealSlideshow。

RealProducer G2 是一个全用途的 SMIL 制作工具,它可以用来创建使用 RealNetworks 的扩展媒体类型以及大多数标准媒体类型的 SMIL 表述。它具有基于使用者的带宽压缩媒体对象,从而可获得最高效率的强力特性。在本书编写期间有两个版本的 RealProducer G2。RealProducer Plus G2 可用于 Windows、Macintosh 和 Linux 平台。名为 RealProducer Pro G2

的专业版本只用于 Windows。

RealNetworks 的 RealSlideshow 是一个创建本章前面养鱼池例子这样幻灯片表述的 SMIL 制作工具。与其他的 RealNetworks 工具类似,RealSlideshow 通过结合 RealNetworks 的扩展工具类型变得非常的有用。图 32.8 显示了如何使用 RealSlideshow 制作工具创建幻灯片表述。

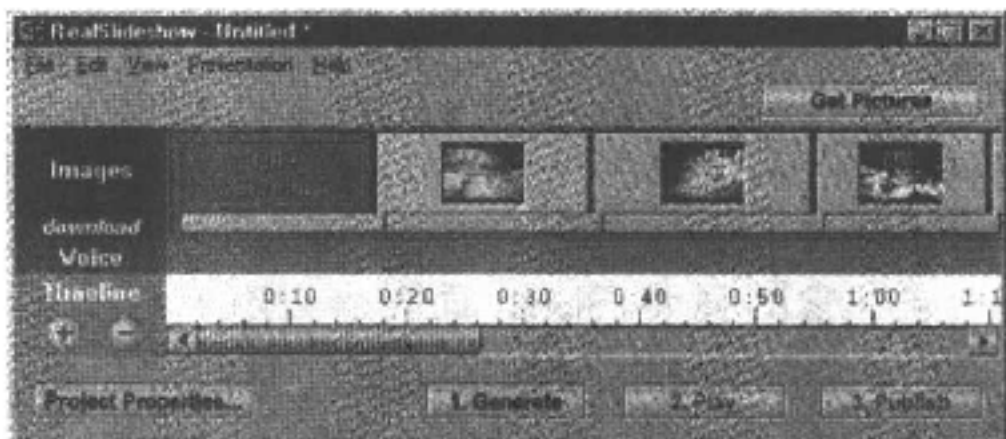


图 32.8 由 RealSlideshow 制作工具创建的 SMIL 幻灯片

RealSlideshow 可以从 RealNetworks 的工具 Web 站点 <http://www.real.com/products/tools> 中免费的获得。这个站点还有对 RealProducer G2 工具的订购信息。

32.5 SMIL 和 HTML+TIME

在你开始考虑 SMIL 在 Web 的同步媒体方面作为绝对的底层之前,请允许我搅点浑水。Microsoft 已经出台了一个名为 HTML+TIME 的同样的技术,它可以提供与 SMIL 相同的功能。HTML+TIME 是 HTML 的扩展,它允许你设定网页中任何元素的定时参数。这与 SMIL 相反,SMIL 只是对于所给的 SMIL 表述中的媒体对象提供特定的定时参数。Microsoft 表达了对 SMIL 这个似乎是的局限性的关注。这也是为什么它要开发 HTML+TIME 的关键所在。

这个争论的另一方面就是 SMIL 表述被设计成在它们自己的物理空间中出现,因为它通常要保持多媒体自包含表述。这方面有些人认为是 SMIL 的局限,而另一些人认为是它设计的明智之处。实际上,大多数 Web 开发人员的需求几乎与任何一种技术都相符。考虑哪一项技术将要胜出还为时过早,或许这两项技术将和平共存。没有理由你不使用 SMIL 进行自包含的多媒体表述而用 HTML+TIME 给网页的元素加入定时特性。

Microsoft 在 HTML+TIME 的潜力上压上了重注。实际上,它还发布了一个名为 Microsoft Vizact 2000 的 HTML+TIME 制作工具。Vizact 与现有的 Office 2000 家族的产品共同工作,并可以通过它们的工具条使用这个工具。真正的事实是 Microsoft 已经在他的庞大的流行 Office 产品中发布了 HTML+TIME 的全功能实现。这足够给 HTML+TIME 成功的机会。

注意:要获得有关 Microsoft Vizact 2000 的更多信息,请访问 Vizact 2000 的 Web 站点 <http://www.microsoft.com/vizact>。

可是这并不确实意味着 SMIL 有了什么麻烦。相反 SMIL 和 HTML+TIME 解决同样

的问题,而它们销售对象并不相同。Microsoft 将 HTML+TIME 作为它的 Office 产品的附件,它还包含文字处理(Word)、电子数据表(excel)和网页制作(FrontPage)以及其他的工具。SMIL 定位于高端的多媒体制作工具,从它在 RealNetworks 工具的存在上这一点是明显的。从我的观点来说,这两项技术在它们所销售的相关领域中都将获得成功。

32.6 总 结

本章探讨了 SMIL(同步多媒体综合元素),这是一个允许你加入媒体对象并同步它们的显示和回放的 XML 应用程序。SMIL 提供了上乘的方法结合媒体元素以创建丰富的基于 Web 的多媒体体验。虽然它非常的强大,SMIL 词表令人吃惊的贫乏。它可以手工创建 SMIL 表述,或者你可以使用与之相关的 SMIL 制作工具自动的创建 SMIL 表述。

本章探讨了 SMIL 的基础以及它作为 XML 应用程序的作用。我们了解了 SMIL 词表中最为常用的元素和属性以及如何使用它们创建包含叙述和背景音乐的多媒体幻灯片表述。HTML+TIME 是一个 Microsoft 提出的技术。本章最后将 SMIL 与这个有着同样功能的技术进行了比较。

第 33 章 使用 CDF 推出 Web 内容

作为 XML 的早期倡导者,Microsoft 是首先创建实际 XML 词表的公司之一,同时它还将 XML 结合进它的 Internet Explorer 网络浏览器中。频道定义格式(CDF——Channel Definition Format)就是一种用来定义频道的 XML 词表。它使网页实现分层,这样就可以实现自动更新。CDF 使得用户预定频道并在频道中的内容改变的时候接收到通知或自动更新这样的功能成为了可能。

本章将向你介绍 CDF 频道以及 CDF 词表。CDF 是一个 XML 应用程序的优秀实例,因为它已经进入应用阶段,而不是像许多的词表那样还依旧处于开发阶段。通过学习 CDF 词表的有关细节,你还将学习到如何使用一个绘画工具创建 CDF 文件。这是一个有关工具是如何使得 Web 开发人员在开发 XML 方面更加方便的优秀例子。

33.1 Active 频道基础

Microsoft 的 Active 频道技术在 Internet Explorer 4.0 中出现,并且依然是 Internet Explorer 5.0 的重要部分。Active 频道的主要假设是:用户从 Web 站点中获取更新信息是很不方便的。大多数的用户有一些他们经常访问的目标站点。如果在这些站点上出现新的内容的时候就能通知用户,这样将带来极大的便利,Active 频道让用户向这些站点预定频道并自动的获取更新信息。

Active 频道还支持离线浏览,也就是它可以将整个站点下载并在离线情况下进行浏览。同样,关键并不在于离线浏览而在于最初的内容获取的自动机制。基本原理是建立更类似于电视或广播内容的 Web 内容的传输,也就是信息移向用户而不是用户获取信息。频道将内容推给用户。

注意:很值得注意在表述频道内容传输的时候,“推”并不完全准确。网络浏览器依然负责从 Web 服务器上获取内容,就如他得到正规的 Web 内容一样。然而,使用频道,用户并不需要访问站点以获得内容。因此,从用户的观点来看,这就好像内容被传输或者推给用户。要谨记并不是所有的用户希望内容推给他们——这依赖于将指定的内容推送,这可能带来重大的带宽花费。

下面是一些使用 Active 频道增强对 Web 内容的浏览和获取的例子:

- 传送分钟更新新闻和信息。
- 告知用户网站内容的改变。
- 智能的将大的 Web 站点改造为更小的子站点的集合。
- 提供对 Web 站点的层次化浏览的支持。

也许 Active 频道最常用的功能是每分钟更新新闻和信息。实际上,新闻和信息并不需要每分钟都更新,但它确实有着一定时间规律的修改。如果它很少修改,那么连使用频道都

不大有必要。通过频道获得新闻和信息的另一个好处就是你可以下载全部的网页并在你空闲的时候离线浏览它们。这对于那些并不经常能够获得 Internet 链接的用户来说是非常有益的。

与频道传输新闻和信息类似的就是告知用户网站内容的改变。例如,你也许有一个有关贵公司的销售力量的内部网联系列表。销售人可以将联系列表预定为频道,然后在联系表添加、删除或修改的时候自动的获得通告。你也可以用这种方式使用频道来更新用户系统上的软件。CDF 实际上包含一个扩展,它允许你定义软件的描述以实现在用户系统上的自动更新和安装。

注意: CDF 扩展(支持软件自动安装)基于开放软件描述(Open Software Description, OSD)。要获得有关 OSD 和软件更新频道的更多信息,请访问如下更新频道 (http://msdn.microsoft.com/workshop/delivery/sdchannel/overview/software_channels.asp) 或 OSD FAQ (<http://msdn.microsoft.com/workshop/management/osd.osdfaq.asp>)。

CDF 是一个层次化语言,这要感谢 XML,它可以用来将 Web 站点分为多层次的子站点。例如,一个在线音乐商店可能希望在某种类型的音乐 CD 发布的时候让用户们知道。也许用户是一个爵士乐迷并希望在任何新的爵士乐专辑发布的时候得到通知。我们可通过创建频道来完成这样的任务。

最后,CDF 的层次化特性还提供了新的用来组织对 Web 站点的导航的方式。你可以有效的创建一个频道站点映射,允许用户访问他们所希望的信息。

33.2 使用 Active 频道

在你学习 CDF 词表的细节之前,让我们简单的看一下 Active 频道实际是如何使用的。从 4.0 版开始,Internet Explorer 网络浏览器支持 Active 频道技术。在那里有一个专门用于 Active 频道的工具条。在 5.0 版本中,Active 频道成为收藏目录中的一个子目录,如图 33.1 所示。

要预定一个频道,你须首先在频道目录中选择频道。图 33.2 显示了在你选择频道目录中的 MSNBC 频道时所显示的页面。

要注意在页面上显示的“Add Active Channel”按钮。这是你要添加新的频道的链接。虽然 Internet Explorer 在收藏目录的频道子目录中附带了一些频道,大多数你要预定的其他频道都可以从他们各自的 Web 站点上获取,这些频道通过“Add Active Channel”标识确认。例如,图 33.3 显示了在 GolfWeb 的 Web 站点(<http://www.golfweb.com>)上的同样按钮。

当你点击“Add Active Channel”按钮以添加一个新的频道的时候,Add Active Channel 对话框就会显示,如图 33.4 所示。

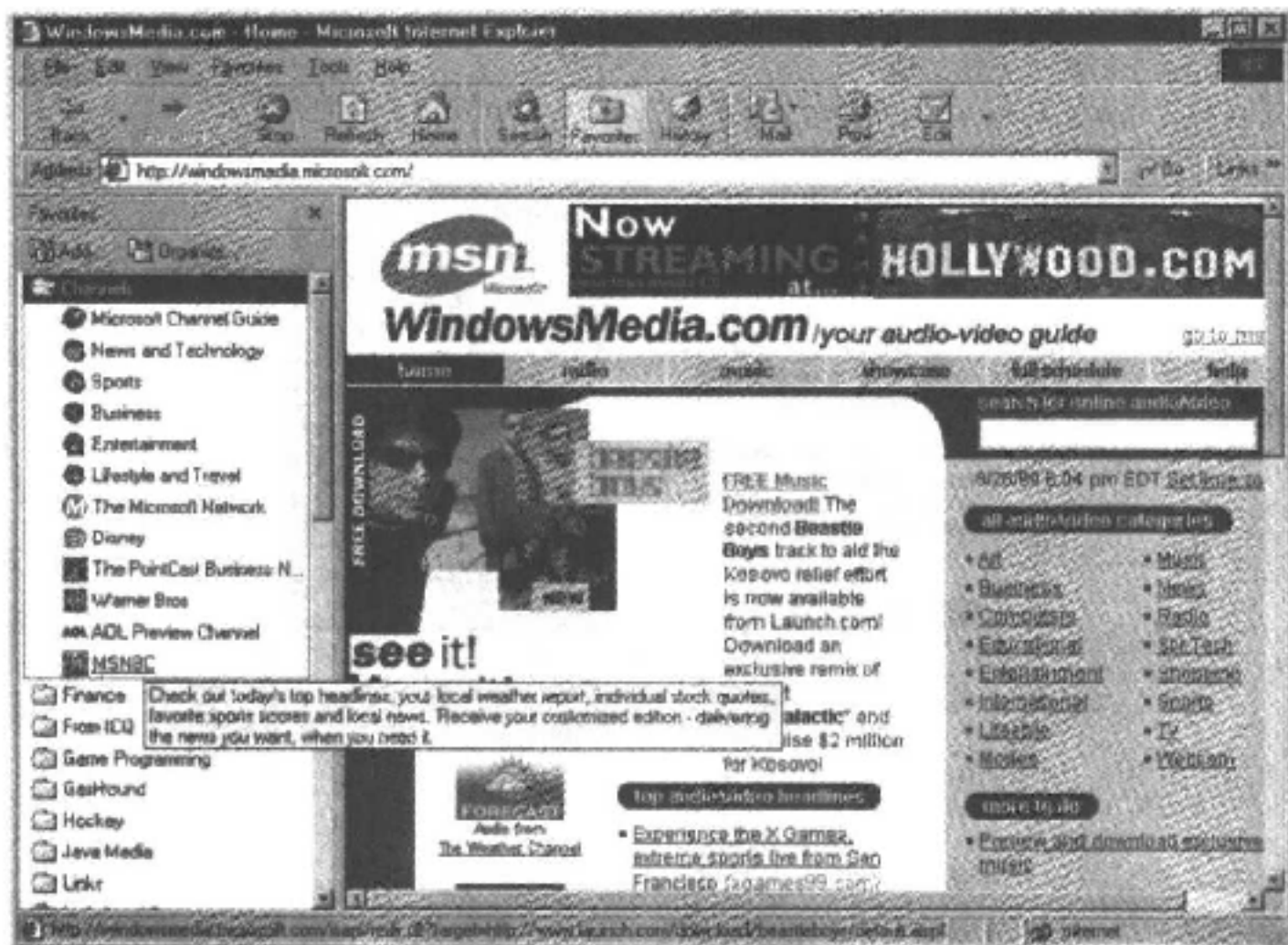


图 33.1 Internet Explorer 5.0 的收藏目录中的 Active 频道子目录

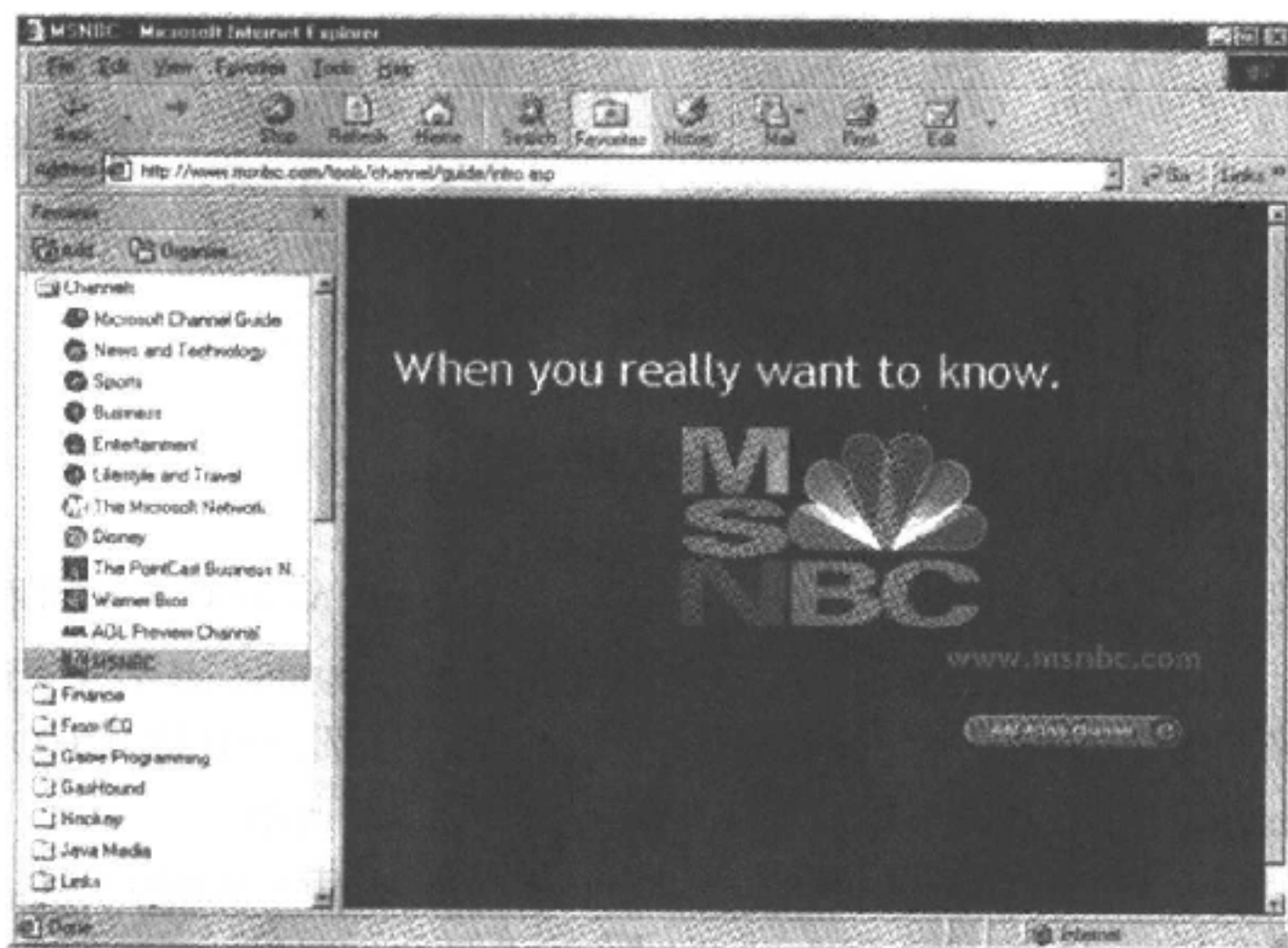


图 33.2 在频道目录中选择 MSNBC 后显示的 MSNBC 的 Active 频道页面

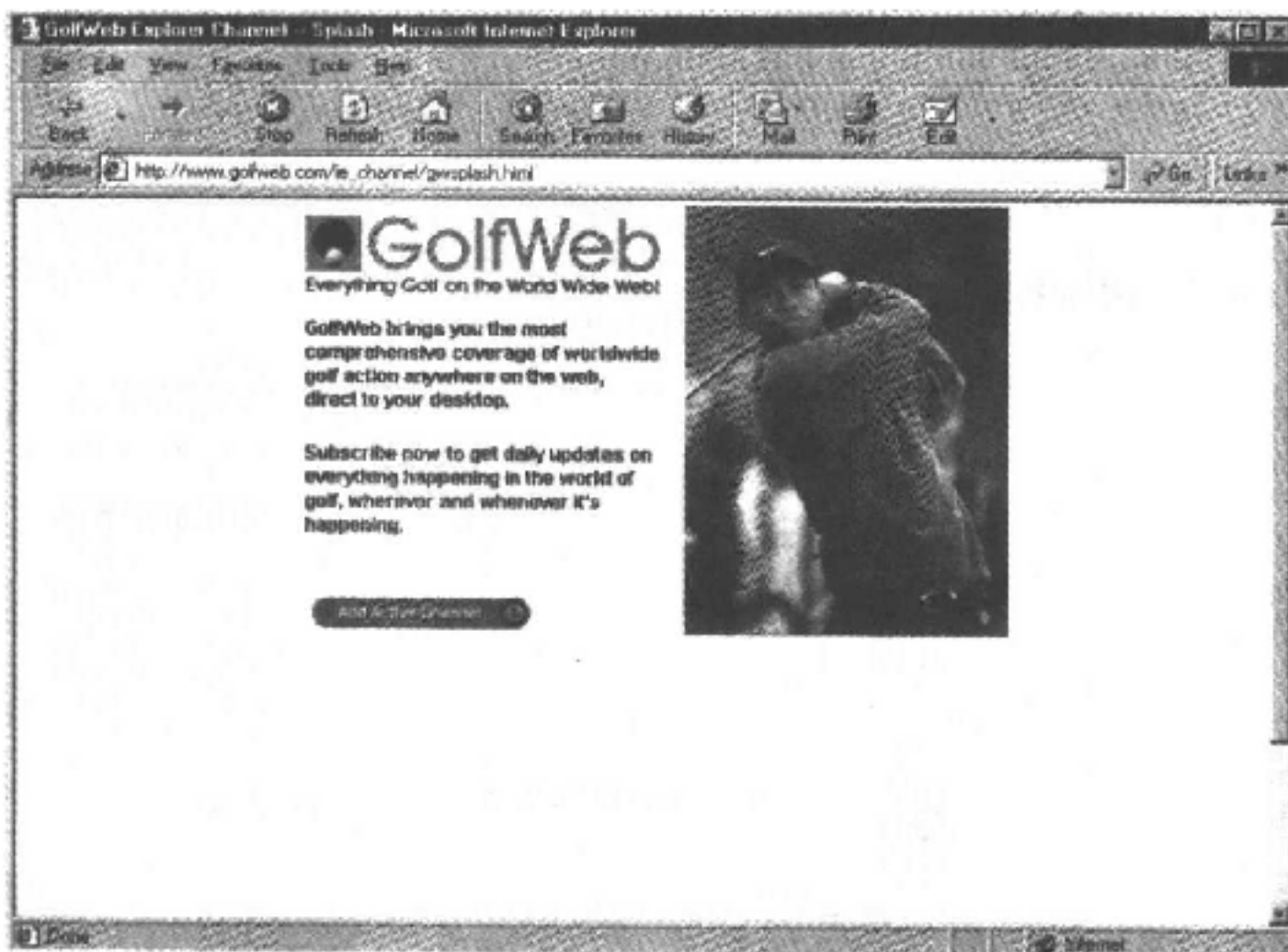


图 33.3 GolfWeb Web 站点上的 Add Active Channel 按钮

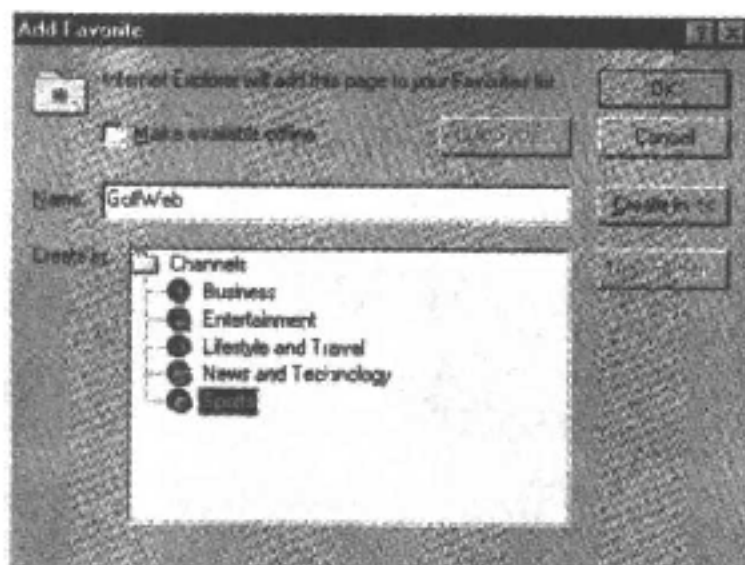


图 33.4 将新的频道添加到 Internet Explorer 5.0 的 Add Active Channel 对话框

在这个对话框中,你可以决定新的频道要放在何处。例如,GolfWeb 频道可能放置到 Sports 目录中。你还可以用这个对话框创建一个新的频道目录。一旦你点击 OK 接受新的频道,Internet Explorer 将在收藏夹上显示这个频道,如图 33.5 所示。

你可以使用收藏夹整理频道并修改频道属性。例如,点击 Organize 按钮来整理你当前预定的所有频道以及收藏的其他 Web 站点。图 33.6 显示了在 Organize Favorites 对话框中选择的 GolfWeb 频道。

GolfWeb 频道非常的简单,并不需要支持对它的属性的改变。另一方面,MSNBS 频道足够的复杂来提供对修改其属性的支持。图 33.7 显示了在 Organize Favorites 对话框中选择的 MSNBC 频道。

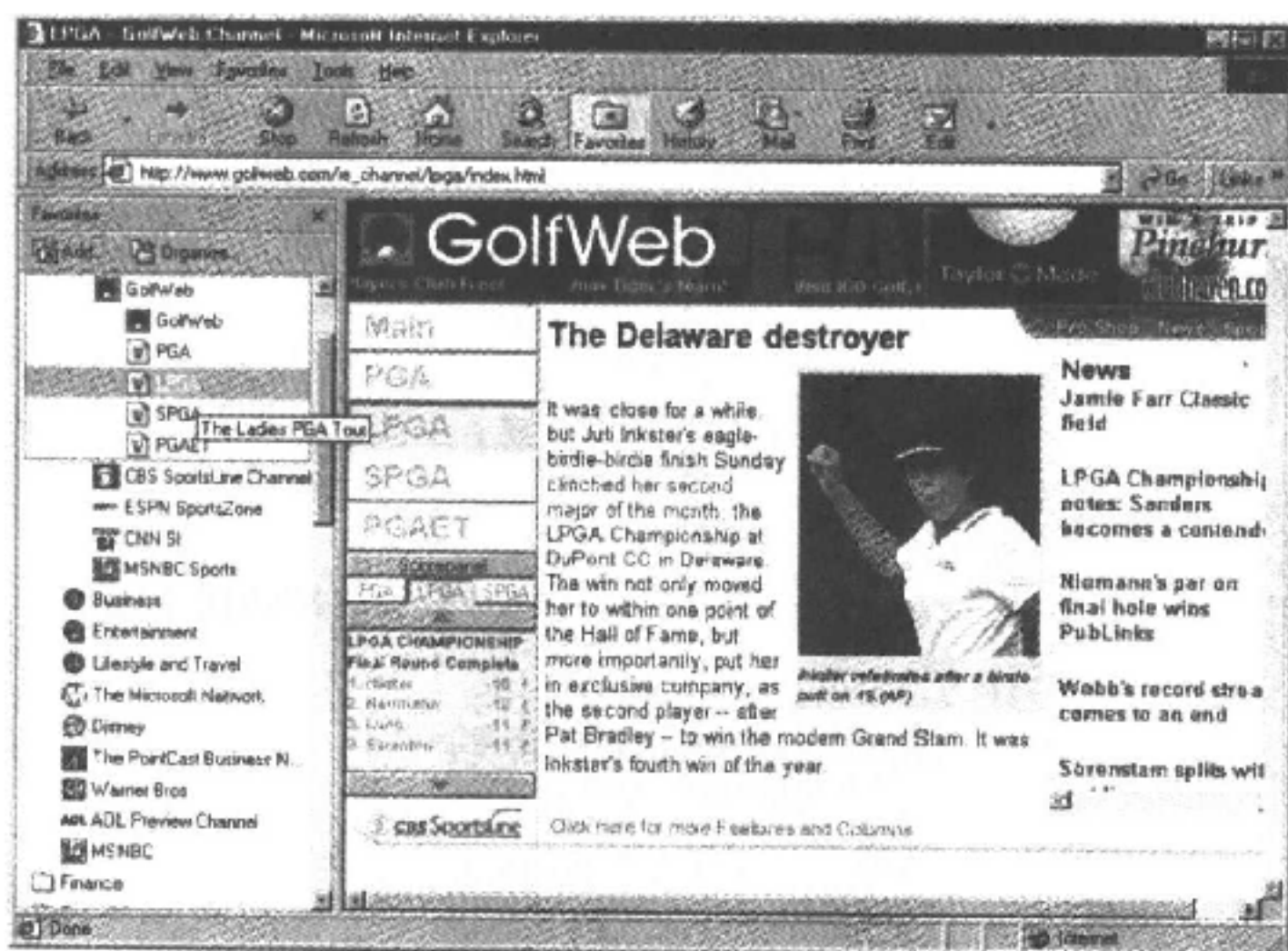


图 33.5 GolfWeb 频道将 GolfWeb 站点分为几个子站点,其中之一是 LPGA 站点

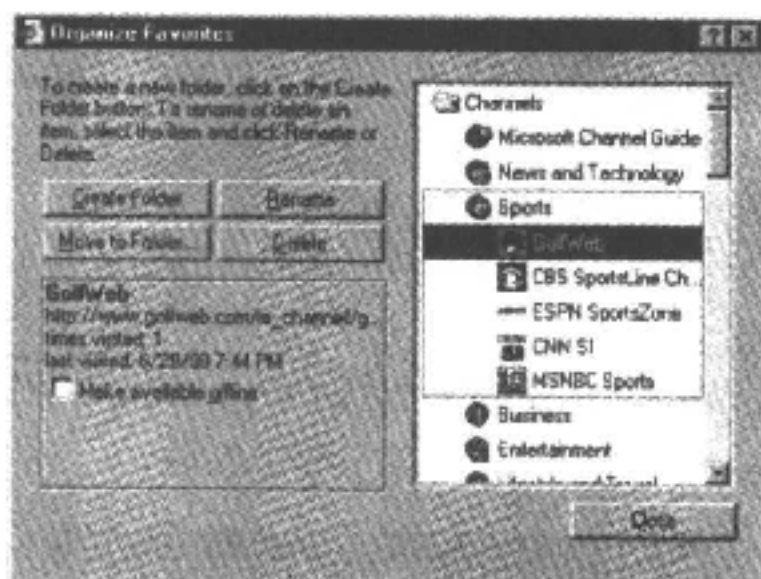


图 33.6 在 Organize Favorites 对话框中选择的 GolfWeb 频道

要注意在对话框的左下角有一个 Properties 按钮。点击这个按钮将允许你调整频道的属性,如图 33.8 所示。

Properties 对话框允许你改变频道的时间表,还可以改变频道内容的下载方式。图 33.9 显示了 Properties 对话框的时间表页面,它允许你改变频道的时间表。

你也可以通过使用 Properties 对话框中的下载页面确定如何将频道的内容下载(如图 33.10)。

这个页面甚至允许你下载站点内容部分中的指定文件,它还可以限制分配给频道的硬盘空间大小。更为重要的是,我希望你已经通过频道定义格式对信息模型的类型有了一定的看法。要谨记所有的定义频道的信息都包装在一个 CDF 文件中。让我们继续,看看 CDF 是如何在 XML 描述中使用的。

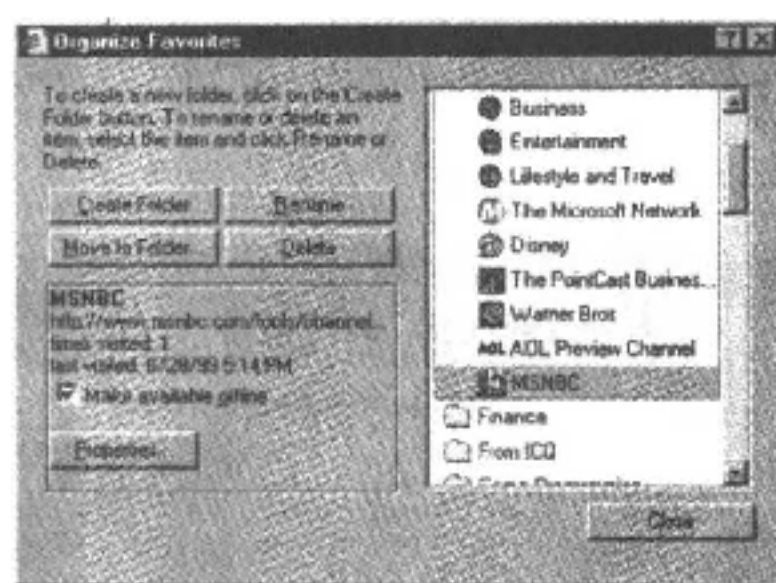


图 33.7 在 Organize Favorites 对话框中选择的 MSNBC 频道

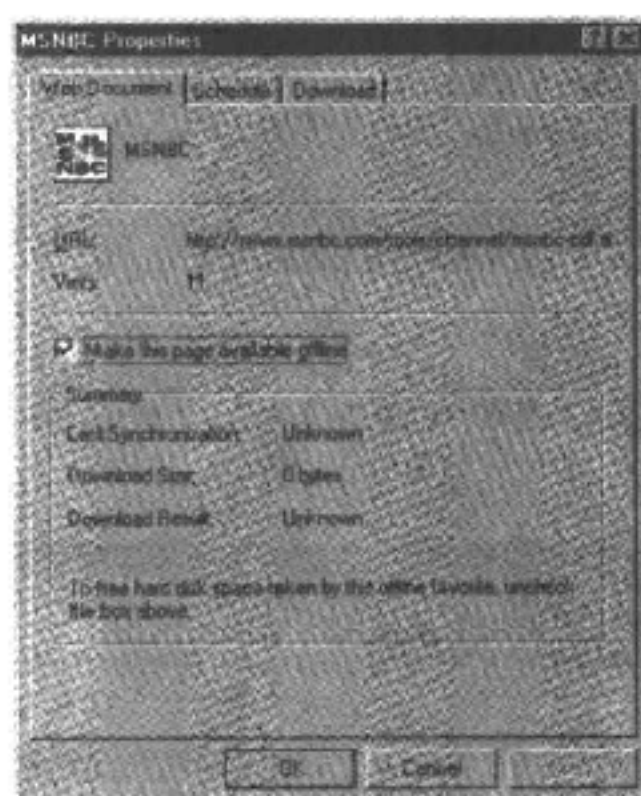


图 33.8 MSNBC 频道的 Properties 对话框

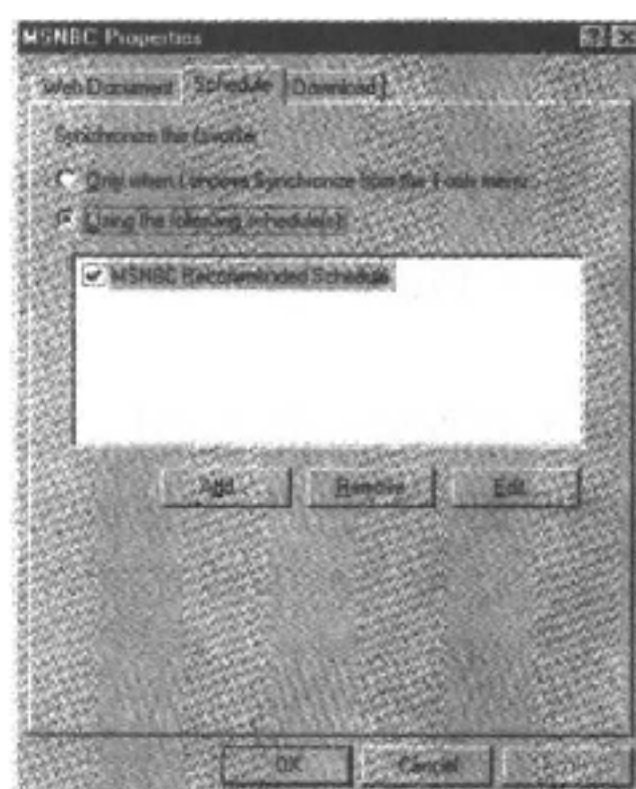


图 33.9 MSNBC 频道的 Properties 对话框中的时间表页面

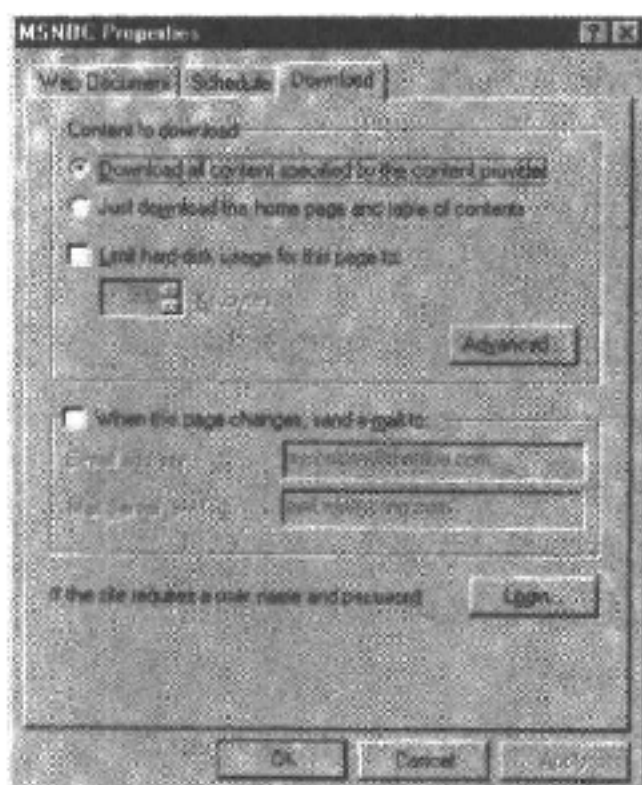


图 33.10 MSNBC 频道的 Properties 对话框中的下载页面

33.3 深入 CDF 词表

现在你已经基本理解了 Active 频道,让我们钻研一下 CDF。如你在本章前面学到的, CDF 就是频道定义格式(Channel Definition Format),也就是说它提供了一个 XML 词表以描述 Web 内容频道。就像所有的 XML 词表一样, CDF 包含一个文件类型定义(DTD),它定义了 CDF 词表的结构和语义。

注意:要获得有关 CDF 词表的更多信息,请访问 Microsoft 的 CDF 发布 Web 站点 <http://msdn.microsoft.com/workshop/delivery/cdf/reference/CDF.asp>。你还可以在 <http://www.w3.org/TR/NOTE-CDFsubmit.html> 中看到最初提交给 W3C 的 CDF 规范。

注意:DTD 是剖析任何 XML 词表结构的基本线索,这也就是为什么它是当你尝试解开一个新的词表的秘密时候的最佳入手点。当然,XML 词表可以是庞大而又笨拙的,因此你在尝试理解 DTD 时通常要依赖于文件。

清单 33.1 包含了 CDF 的 DTD:

清单 33.1 CDF 的 DTD

```
<! ELEMENT Channel ( LastMod | Logo | Title | Abstract | Author |
  Publisher | Copyright | PublicationDate | Keywords | Category |
  Rating | Channel | Item | Schedule | IntroURI | Authorization |
  IsClonable | MinStorage | Tracking ) * >
<! ATTLIST Channel HREF CDATA #IMPLIED>
<! ATTLIST Channel IsClonable (YES | NO) "NO">
<! ELEMENT LastMod EMPTY>
<! ATTLIST LastMod VALUE CDATA #REQUIRED>
```

```

<! ELEMENT Logo EMPTY>
<! ATTLIST Logo HREF CDATA #REQUIRED>
<! ATTLIST Logo TYPE (BIG | WIDE | SMALL | REGULAR) "REGULAR">

<! ELEMENT Title EMPTY>
<! ATTLIST Title VALUE CDATA #REQUIRED>

<! ELEMENT Abstract EMPTY>
<! ATTLIST Abstract VALUE CDATA #REQUIRED>

<! ELEMENT Author EMPTY>
<! ATTLIST Author VALUE CDATA #REQUIRED>

<! ELEMENT Publisher EMPTY>
<! ATTLIST Publisher VALUE CDATA #REQUIRED>

<! ELEMENT Copyright EMPTY>
<! ATTLIST Copyright VALUE CDATA #REQUIRED>

<! ELEMENT PublicationDate EMPTY>
<! ATTLIST PublicationDate VALUE CDATA #REQUIRED>

<! ELEMENT Keywords EMPTY>
<! ATTLIST Keywords VALUE CDATA #REQUIRED>

<! ELEMENT Category EMPTY>
<! ATTLIST Category VALUE CDATA #REQUIRED>

<! ELEMENT Rating EMPTY>
<! ATTLIST Rating PICS-Label CDATA #REQUIRED>

<! ELEMENT Item( Last Mode | Title | Abstract | Author | Publisher |
  Copyright | PublicationDate | Keywords | Category | Rating | Schedule |
  Usage ) * >
<! ATTLIST Item HREF CDATA #REQUIRED>
<! ATTLIST ITEM MIMETYPE CDATA #IMPLIED>
<! ATTLIST Item IsVisible (YES|NO) "YES">
<! ATTLIST Item Priority (HI|NORMAL|LOW) "NORMAL">
<! ATTLIST Item Precache (YES|NO | DEFAULT) "DEFAULT">

<! ELEMENT Usage ANY>
<! ATTLIST Usage VALUE CDATA #REQUIRED>

<! ELEMENT Schedule (StartDate?,EndDate?,IntervalTime?,EarliestTime?,
  LatestTime? ) >

<! ELEMENT StartDate EMPTY>
<! ATTLIST StartDate VALUE CDATA #REQUIRED>

<! ELEMENT EndDate EMPTY>
<! ATTLIST EndDate VALUE CDATA #REQUIRED>

<! ELEMENT IntervalTime EMPTY>
<! ATTLIST IntervalTime DAY CDATA "0">
<! ATTLIST IntervalTime HOUR CDATA "0">
<! ATTLIST IntervalTime MIN CDATA "0">
<! ATTLIST IntervalTime SEC CDATA "0">

<! ELEMENT LatestTime EMPTY>
<! ATTLIST LatestTime DAY CDATA "0">
<! ATTLIST LatestTime HOUR CDATA "0">
<! ATTLIST LatestTime MIN CDATA "0">
<! ATTLIST LatestTime SEC CDATA "0">

```

```

<! ELEMENT IntroUri EMPTY>
<! ATTLIST IntroURI VALUE CDATA # REQUIRED>

<! ELEMENT Authorization EMPTY>
<! ATTLIST Authorization VALUE CDATA # REQUIRED>

<! ELEMENT MinStorage EMPTY>
<! ATTLIST MinStorage VALUE CDATA # REQUIRED>

<! ELEMENT Tracking (PostURL?)>
<! ELEMENT PostURL EMPTY>
<! ATTLIST postURL HREF CDATA #REQUIRED>

<! ELEMENT UserSchedule EMPTY>
<! ATTLIST UserSchedule VALUE (DAILY| WEEKLY| HOURLY) #REQUIRED>

<! ELEMENT Width EMPTY>
<! ATTLISST Width VALUE CDATA # REQUIRED>

<! ELEMENT Height EMPTY>
<! ATTLIST Height VALUE CDATA # REQUIRED>

```

虽然 CDF 的 DTD 定义了一些不同的元素,实际上只有几个元素对理解 CDF 文件是重要的。下面是在 CDF 词表中定义的基本元素。

- Channel
- Title
- Abstract
- Author
- Publisher
- Copyright
- PublicationDate
- LastMod
- Schedule
- Logo

33.3.1 Channel 元素

Channel 元素是 CDF 文件的文件元素,也就是说它作为根元素使用并覆盖所有的其他元素。我们也可以将 Channel 元素嵌套到其他 Channel 元素中,这种情况下,嵌套的 Channel 元素描述了基本频道中的子频道。Channel 元素可以包含如下的子元素:LastMod、Logo、Title、Abstract、Author、Publisher、Copyright、PublicationDate、Keywords、Category、Rating、Channel、Item、Schedule、IntroURI、Authorization、IsClonable、MinStorage 和 Tracking。

Channel 元素接受以下两个属性:Href 和 IsClonable。这个 HREF 属性用来指定频道的主页面。IsClonable 属性指定频道是否可以作为另一个频道的子频道来复制。下面是使用这些属性的例子:

```

<Channel HREF=http://www.thetribe.com/newschannel.html IsClonable="NO">
</Channel>

```

注意:Microsoft 已经向 W3C 提交了一个 CDF 修订版以进行评估。修订版包括对一些 CDF 元素的结构修改,而大体上修改是整体性的。例如,一些元素被转变为属性,如 LastMod。要承认,新的修改似乎将要求对基于原始规定的 CDF 文件的组织重新编写,而内容则基本上是一样的。修订版在本书编写期间还处于等待承认阶段,因此本章将基于 W3C 制作的最初 CDF 提案。

33.3.2 Title 元素

Title 元素描述了频道的标题,它可被用户读取。这是一个空元素,包含一个单独的属性 VALUE。VALUE 属性就是你设置频道标题文本的地方,它将在用户浏览和选择频道时被显示。这个元素描述的标题字符串将在用户平台上的 Windwos Media Showcase 以及 Internet Explaorer 5.0 的收藏夹中指代你的频道。下面是使用 Title 元素设置频道标题的例子:

```
<title VALUE="The Tribe News Channel"/>
```

33.3.3 Abstract 元素

Abstract 元素用来辅助 Title 元素提供对频道的详细描述。这是一个包含单独属性 VALUE 的空元素。VALUE 属性就是你放置频道的摘要文本的地方。下面是使用 Abstract 元素设置频道的摘要文本的例子:

```
<Abstract VALUE="A channel that provides up-to-the-minute news on happenings at The Tribe." />
```

33.3.4 Author 元素

Author 元素用来确定频道和它的内容的作者。它是一个包含一个单独属性 VALUE 的空元素。VALUE 属性就是你放置频道的作者姓名的地方。下面是一个使用 Author 元素设置频道作者的例子:

```
<Author VALUE="Michael Morrison"/>
```

33.3.5 Publisher 元素

Publisher 元素用来确定频道和它的内容的发行人。鉴于 Author 元素决定了频道和它的内容的作者,Publisher 元素确定频道所属的组织。Publisher 元素是包含单个属性 VALUE 的空元素。VALUE 属性就是你放置频道的发行人名称的地方。下面是一个如何使用 Publisher 元素设置频道的发行人的例子:

```
<Publisher VALUE="The Tribe"/>
```

33.3.6 Copyright 元素

Copyright 元素用来提供关系到谁拥有频道和内容的版权的信息。这是一个包含单个属性 VALUE 的空元素。VALUE 就是你放置频道的版权拥有者的名称的地方。下面是一个使用 Copyright 元素确定频道的版权拥有者的例子:


```
<Copyright VALUE="The Tribe"/>
```

33.3.7 PublicationDate 元素

PublicationDate 元素用来设置频道和它的内容的初始公布日期。它是一个有着单个属性 VALUE 的空元素。这个元素 VALUE 属性通过结构为 YYYY.MM.DD 的字符串确定,这个字符串中分别指的是年、月和日。下面是一个设置频道的发布日期的例子。

```
<PublicationDate VALUE="1999.11.21"/>
```

在这个例子中,频道的发布日期定为 1999 年 11 月 21 日。

33.3.8 LastMod 元素

LastMod 元素用来描述频道和它的内容的最近修改日期。它是一个有着单个属性 VALUE 的空元素。这个元素 VALUE 属性通过结构为 YYYY.MM.DD 的字符串确定,这个字符串中分别指的是年、月和日。下面是一个确定频道的最近更改日期的例子:

```
<LastMod VALUE="1999.12.05"/>
```

在这个例子中,频道的最近修改日期是 1999 年 12 月 5 日。

33.3.9 Schedule 元素

Schedule 元素用来描述频道的时间表。这是一个容器元素,它包含如下的元素:StartDate、EndDate、IntervalTime、EarliestTime 和 LatestTime。StartDate 和 EndDate 元素指定频道更新的开始和结束日期。这些元素是有一个单个属性 VALUE 的空元素。这两个元素 VALUE 属性通过结构为 YYYY.MM.DD 的字符串确定,这个字符串中的各部分分别指的是年、月和日。下面是一个确定频道的开始和结束日期的例子:

```
<Schedule>
  <StartDate VALUE="2000.01.01"/>
  <EndDate VALUE="2001.06.30"/>
</Schedule>
```

本例中开始日期为 2000 年 1 月 1 日,结束日期为 2001 年 6 月 30 日。

IntervalTime、EarliestTime 和 LatestTime 元素用来调整频道的更新日期表的时间。这些元素都是空元素,由四个属性确定更新时间:DAY、HOUR、MIN、SEC。IntervalTime 元素确定频道更新的时间间隔长度,附带的属性确定这个长度。同样的方法用来确定 EarliestTime 和 LatestTime 元素的值。这里两个元素定义了频道可以更新的时间窗口。这个窗口根据 IntervalTime 元素计算。下面是一个使用所有的这些元素指定频道的更新时间表的例子:

```
<Schedule>
  <StartDate VALUE="2000.01.01"/>
  <EndDate VALUE="2001.06.30"/>
  <IntervalTime DAY="7"/>
  <EarliestTime HOUR="2"/>
  <LatestTime HOUR="6"/>
</Schedule>
```

</Schedule>

在这个例子中,间隔时间设置为 7 天,这说明频道每星期更新一次。频道最早的更新时间是凌晨 2:00,最晚时间是早上 6:00。这两个时间确定了一个四小时的时间段,频道每七天在这个时间段更新一次。

33.3.10 Logo 元素

Logo 元素用来指定一个指代频道的图形标识。它是一个包含两个属性 HREF 和 TYPE 的空元素。HREF 属性确定频道引用图片的 URL 地址。TYPE 属性指出图片的大小,它的值可以是 REGULAR、WIDE 或 SMALL。REGULAR 是默认的属性值,意思是如果你计划使用常规的图片尺寸(80×32 像素),你可以忽略 TYPE 属性。你可能希望提供多种尺度的标识,这样你的频道就可以按多种不同的途径显示了。不同的标识尺寸如下:

- REGULAR——80×32 像素
- WIDE——194×32 像素
- SMALL——16×16 像素

下面是一个设置频道的多种标识的例子:

```
<Logo HREF="http://www.thetribes.com/images/newschan_r.gif"
  TYPE="REGULAR">
<Logo HREF="http://www.thetribes.com/images/newschan_w.gif"
  TYPE="WIDE">
<Logo HREF="http://www.thetribes.com/images/newschan_s.gif"
  TYPE="SMALL">
```

33.4 使用 CDF 创建频道

你已经看到了组成 CDF 频道所必需的不同元素。让我们将它们组合到一个单独的频道定义文件中。清单 33.2 包含了 Tribe 的新闻频道 CDF 文件的完整代码。

清单 33.2 Tribe 的新闻频道的 CDF 文件

```
<? xml version="1.0" encoding="UTF-8" ? >
<Channel HREF="http://www.thetribes.com/newschannel.html" IsClonable="NO">
  <Title VALUE="The Tribe News Channel"/>
  <Abstract VALUE="A channel that provides up-to-the-minute news on
    happenings at the The Tribe."/>
  <Author VALUE="Michael Morrison"/>
  <Publisher VALUE="The Tribe"/>
  <Copyright VALUE="The Tribe"/>
  <PublicationDate VALUE="1999.11.21"/>
  <LastMod VALUE="1999.12.05"/>
  <Schedule>
    <StartDate VALUE="2000.01.01"/>
    <EndDate VALUE="2000.06.30"/>
```

```

    <IntervalTime DAY="7"/>
    <EarliestTime HOUR="2"/>
    <LatestTime HOUR="6"/>
  </Schedule>
  <Logo HREF="http://www.thetribes.com/images/newschan-r.gif"
    TYPE="REGULAR"/>
  <Logo HREF="http://www.thetribes.com/images/newschan-w.gif"
    TYPE="WIDE"/>
  <Logo HREF="http://www.thetribes.com/images/newschan-s.gif"
    TYPE="SMALL"/>
</Channel>

```

要使用 CDF 扩展名来保存所有的 CDF 文件,这一点很重要。否则,Internet Explorer 将不能把文件识别为 CDF 文件。在成功的创建了 CDF 文件后,你必须创建一个在你的站点上引用这个文件的超级链接。使用 Microsoft 的 Add Active Channel 标识链接这个文件以使用户可以确认你在你的站点上提供了一个 Active 频道。这个标识放在下面的 URL 中:

<http://msdn.microsoft.com/workshop/delivery/channel/cdf1/addChan.gif>

下面是一个创建使用 Add Active Channel 标识并引用名为 news.cdf 的 CDF 文件的超级链接的例子。

```

<A HREF="http://www.thetribes.com/news.cdf">
  <IMG BORDER=0 HEIGHT=136 WIDTH=20
    SRC="http://msdn.microsoft.com/workshop/delivery/channel/cdf1/addChan.gif">
  Click here to subscribe to The Tribe's News Channel. </A>

```

要提到的另一问题是 Active 频道只能够被 Internet Explorer 的 4.0 以上版本支持。因此,你可能希望在启动 Add Active Channel 链接之前检查你的 Internet Explorer 版本。下面是一段 JavaScript 代码,它可以确定你的 Internet Explorer 版本是否是 4.0 及以上版本。

```

<!--
<SCRIPT LANGUAGE="JavaScript">
  function isMsie4orGreater() {
    var ua = window.navigator.userAgent; var msie = ua.indexOf("MSIE");
    if (msie > 0) {
      return (parseInt(ua.substring(msie+5, ua.indexOf(".", msie))) >= 4)
        && (ua.indexOf("MSIE 4.0b") < 0);
    }
    else {return false;}
  }
</SCRIPT>
-->

```

这是一个包含标准代码的脚本,它可以用来检查那些可以包含 Active 频道支持的 Internet Explorer 版本。你会非常希望将这段代码加入到你自己的 Web 开发工具包中,在任何你使用不被 Internet Explorer 4.0 以下版本支持的技术的情形下。

33.5 CDF 生成器工具

现在,你已经学习了如何使用 XML 标记装配一个 CDF 文件。你可能很乐意学习处理

这个过程的自动化的方法。Microsoft 的 CDF 生成器工具为提供创建 CDF 文件可视化接口而设计。CDF 生成器提供了所有的 CDF 标记并且包含一个 XML 解析器,智能的解析 CDF 文件并报告错误。图 33.11 显示了 CDF 生成器工具启动时的状态。

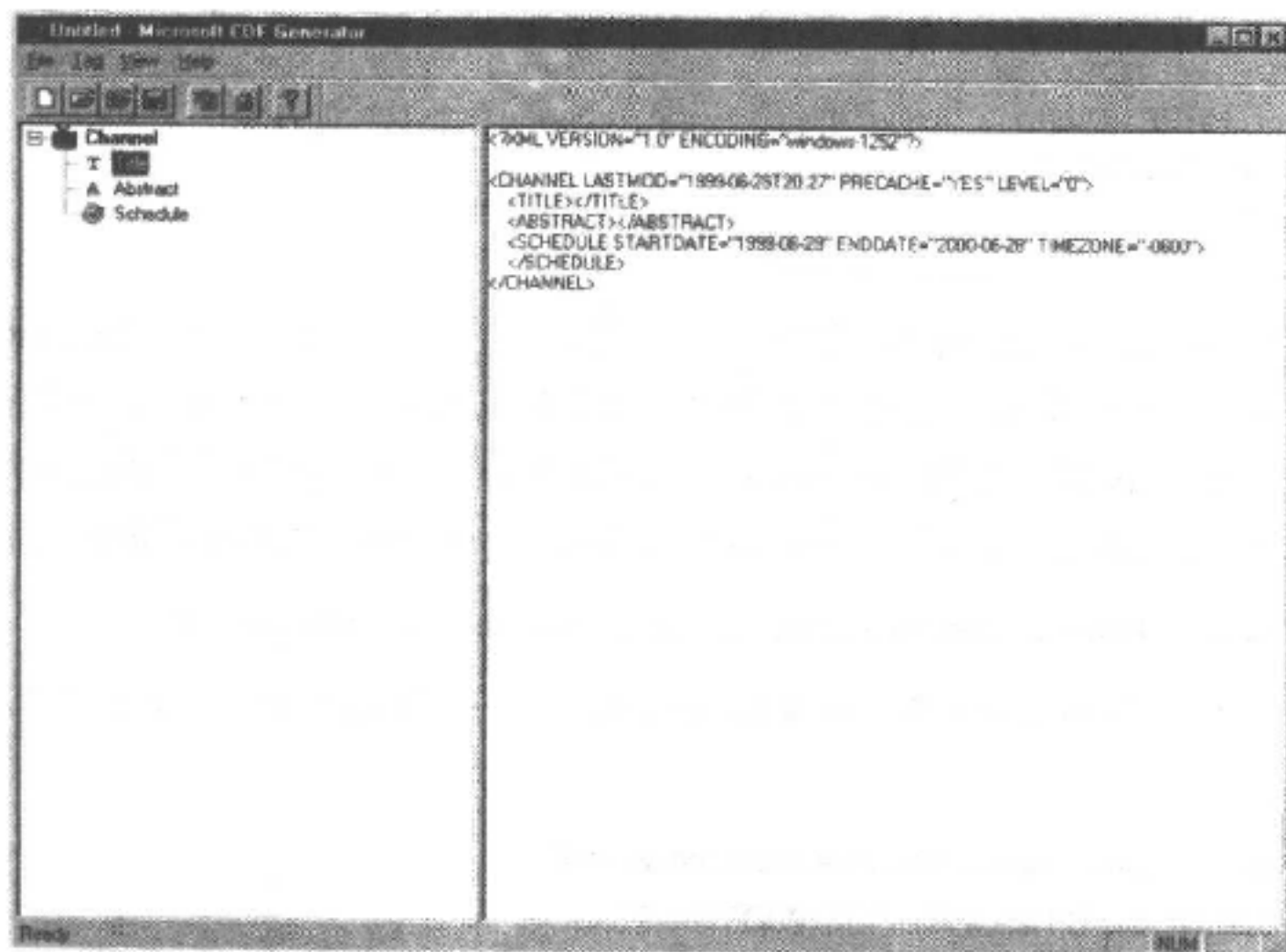


图 33.11 Microsoft 的 CDF 生成器工具

注意:你可以从 Microsoft 的 CDF 生成器 Web 站点 <http://msdn.microsoft.com/workshop/delivery/cdf/cdfgen.asp> 中获得更多的信息并下载 CDF 生成器。

CDF 生成器的用户接口由两个主要的窗口组成:树形窗口(左方)和代码窗口(右方)。这些窗口提供了处理 CDF 文件的优秀接口,而使用 CDF 生成器的真正好处是它的新建频道向导。从 File 菜单中选择 New,你就得到了这个向导的 Channel 页面(如图 33.12)。

注意:CDF 生成器工具的一个问题是根据 Microsoft 的修订 CDF 词表设计的,这个词表还处于 W3C 的承认阶段。在 CDF 生成器中使用修改的词表是可以的,因为 Internet Explorer 支持这个词表。然而,要谨记最终的 CDF 文件将与你在本章中看到的文件在结构上有些许的不同。

Channel 页面有三个可编辑的编辑字段:HREF、BASE 和 SELF。HREF 字段包含频道主页面的 URL,它可以作为属性或是锚(元素)来确定。BASE 字段包含频道发布的站点根页面的 URL,这对于确定相关的路径是必要的。SELF 字段包含频道公布的页面的完全 URL 地址。

图 33.13 显示了 CDF 生成器的新建频道向导的 Common 页面。

Common 页面包含与频道相关的公共属性。这些属性包含频道的题目、频道的摘要文本和记录事件的日志。

图 33.14 显示了 CDF 生成器的新建频道向导的 Logo 页面。

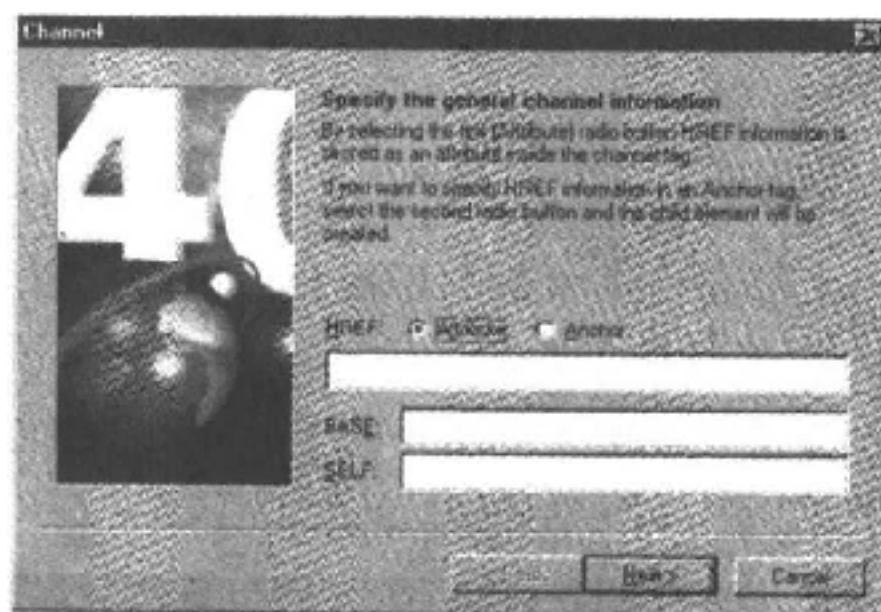


图 33.12 CDF 生成器新建频道向导的 Channel 页面

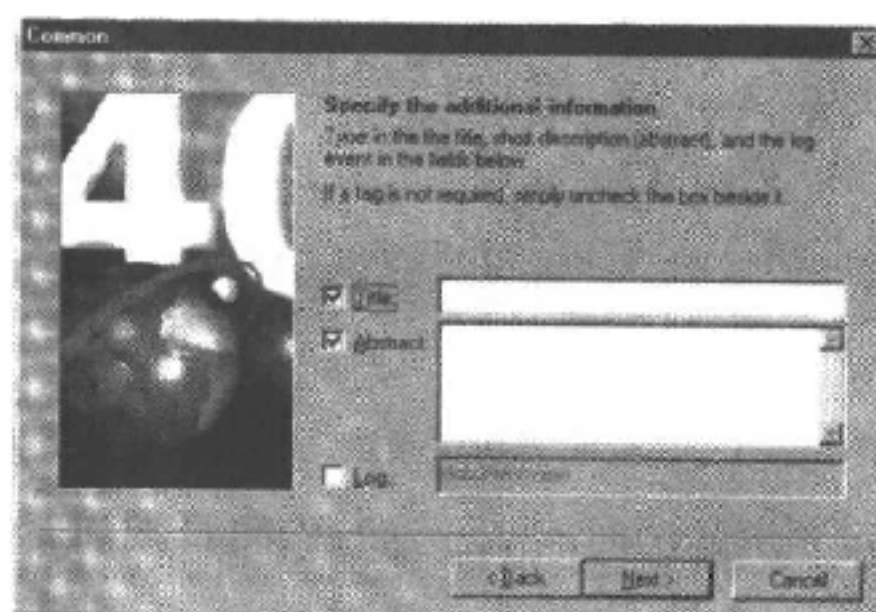


图 33.13 CDF 生成器的新建频道向导的 Common 页面

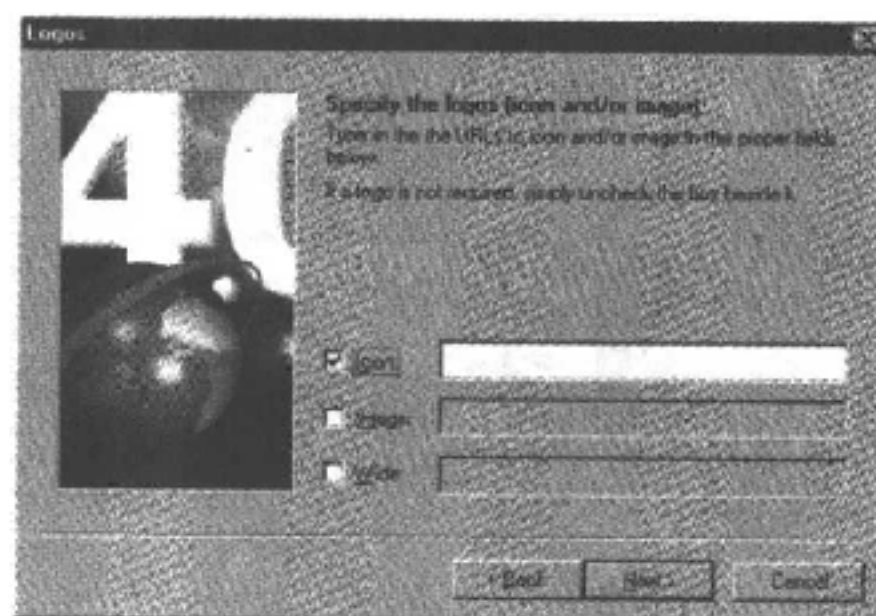


图 33.14 CDF 生成器的新建频道向导的 Logo 页面

Logos 页面允许你指定频道的标识。在这些标识中每一个都包含图片，这确实是一个好主意。注意这些标识与你在本章前面学到的 CDF 词表描述在命名上有些许不同。更为重要的是，下面的列表显示了 Microsoft 的标识名与你在 CDF 词表中所见到的标识名的对应关系。

- Icon——Small
- Image——Regular
- Wide——Wide

修改的标识名表现了 Microsoft 对 CDF 词表的一种修正。这是一个提供各种持续的标识文件的好主意,因为每一个都在提供频道的可视化标识中扮演一个特别的角色。例如 Icon 标识在 Internet Explorer 的频道目录中显示。

图 33.15 显示了 CDF 生成器新建频道向导的 Schedule 页面。

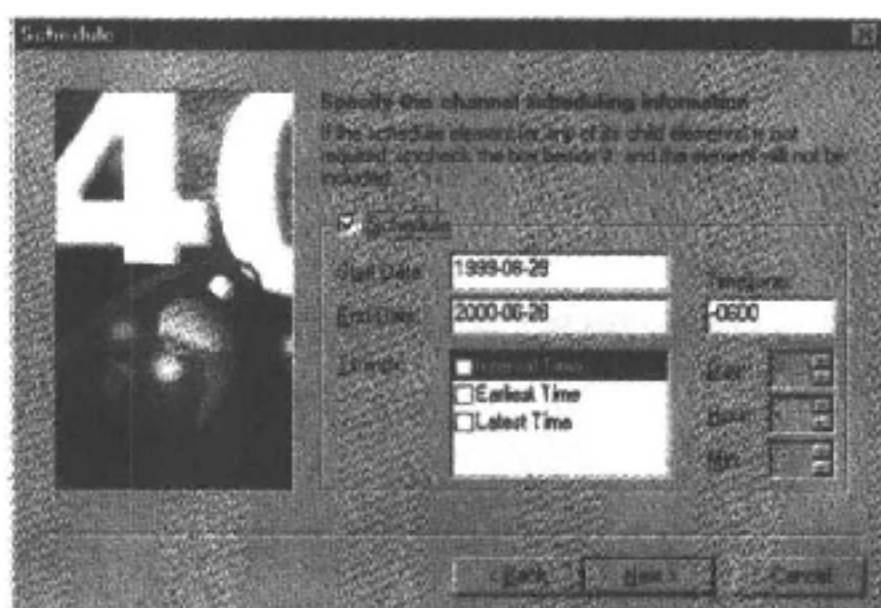


图 33.15 CDF 生成器新建频道向导的 Schedule 页面

Schedule 页面用来确定频道的的时间表。这种编辑日期和时间的可视化接口比手工编写 CDF 文件要直观的多。

图 33.16 显示了 CDF 生成器的新建频道向导的 Log Target 页面。



图 33.16 CDF 生成器的新建频道向导的 Log Target 页面

Log Target 页面允许你指定频道的日志记录目标的 URL、方法和范围。这些属性允许你调整频道日志以跟踪频道活动的情况。

图 33.17 显示了 CDF 生成器的新建频道向导的 Create 页面。

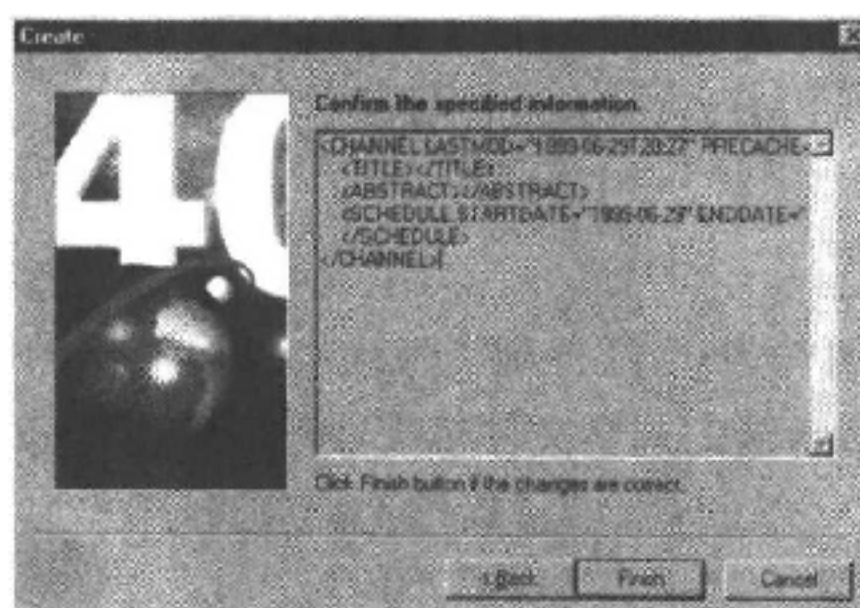


图 33.17 CDF 生成器的新建频道向导的 Create 页面

Create 页面显示了你为最终的 CDF 文件所指定的全部选项。要接受这个文件并将它在 CDF 生成器中自动的打开,只要点击 Finish 按钮。文件就会在 CDF 生成器中打开,在那里你可以使用 CDF 生成器的用户接口进一步对它进行修改。

33.6 总 结

本章向你介绍了最早的一种 XML 词表,频道定义格式(CDF)。CDF 由 Microsoft 创建,用来支持频道,频道是可以自动更新的层次结构网页。有了 CDF,用户可以预定频道并在频道内容改变的时候收到通知或自动的更新。CDF 是 XML 应用的优秀实例,因为它已经被接受并被广泛的使用,这不像其他的许多 XML 词表那样还处于开发阶段。

本章开始部分提供了 Microsoft 的 Active 频道技术的背景信息,它构成了 CDF 文件的基础。之后我们学习了 XML 词表,这部分包括分析 CDF 的 DTD 并探讨组成 CDF 词表的不同元素和属性。接着你学习了如何使用 CDF 词表创建频道。最后,本章主要讲解了 Microsoft 的 CDF 生成器工具,它可以自动的处理创建 CDF 文件的任务。

第34章 使用 VML 和 SVG 描述矢量图形

矢量图是 XML 技术中一个已经得到了极大关注的特殊领域,这一点并不令人惊讶。XML 迅速的证明了自己是未来的共享文件格式,而矢量图当然是比一般的文件格式要更有好处的东西。XML 作为矢量图标记语言提供了如此多的好处,有三个用于处理矢量图的 XML 词表已经提交给 W3C(PGML、VML 和 SVG)。

本章将简单介绍一些矢量图以及一个支持它的 XML 词表的需求。你还可以学习到两个当前正在争夺“头名”的 XML 词表。幸运的是,它们是类似的。因此,只要详细的学习它们其中的一个词表,你就可以理解这两个词表的主要结构。要谨记实际用来描述矢量图的 XML 文件将会通过诸如 Adobe Illustrator 这样的图形工具规范的自动生成。然而,作为 XML 的学习者,在使用之前了解一下 XML 词表的具体细节是很有益处的。

34.1 结构化矢量图形的重要性

当今大多数 Web 上的图形是位图格式的,例如 GIF 和 JPEG。位图格式描述组成图形的独立矩形像素。虽然位图作为诸如照片这样的扫描图片是很优秀的,但它们占据了大量的空间并且在 Internet 上传输很慢。加之,当缩放或变换点阵图片的时候很难获得高质量的效果。这是因为点阵图片除了行像素信息以外,并没有考虑到有关一个图形结构的任何方面的信息。没有方法可以在保留清晰效果的情况下变换位图。图 34.1 示范了对一个图形分别使用点阵和矢量方法放大后的图形的区别。

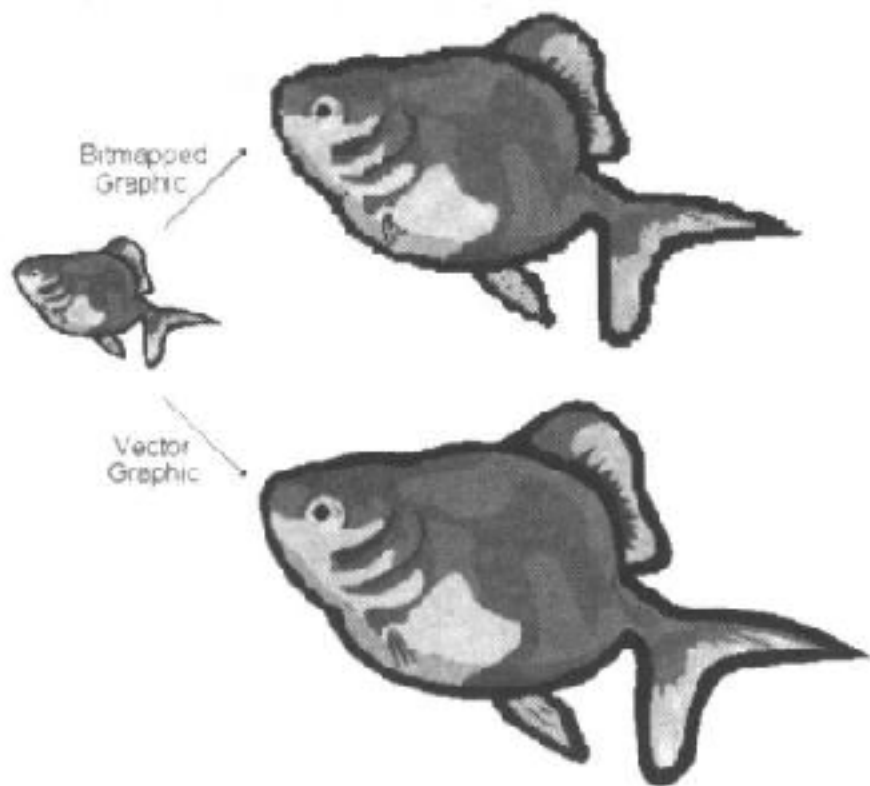


图 34.1 放大图片清楚的揭示出位图和矢量图片之间的区别

在过去的两年中,矢量图片已经开始冲入 Web 图片的世界。像 Macromedia Flash 这样的工具已经使得创建高质量的、互动的、比位图更节省空间的矢量图成为可能,同时它也可

以方便的对矢量图进行变换和操作。如果你对 Web 上的矢量图形并不熟悉,点击这个站点 <http://www.pregnancycalendar.com/first9months/>(如图 34.2)以有所了解。这是我的一个朋友为了纪念他的第一个孩子的出生而创建的。我的这个朋友规范的使用 Flash 创建了动态的、互动的矢量图形。

注意: 矢量图形通过使用几何学元素进行数学的描述,这些元素有线、多边形和椭圆。与基于像素数组的位图不同的是,矢量图形可以方便的进行无损处理。这是因为矢量图形实际是准确的。



图 34.2 First 9 Months 网站是一个具有非常引人瞩目的矢量图形的优秀例子

虽然 Flash 提供了在 Web 上使用矢量图形的一种方法,但它还有自己的二进制格式文件(需要专门的插件)。不仅插件会有一点冲突,二进制文件格式也使得它难以在应用程序间共享文件。可以确认,Macromedia 已经在最近公布了 Flash 文件的格式,这样其他应用程序就可以对它进行支持了。然而,支持可用于多个应用程序的二进制文件比起支持良好结构化的文本格式来,本质上就要困难的多。同样,将二进制数据直接嵌入到网页之中也是不可能的。

注意: 另一个常用的矢量图形格式是 CGM,也就是计算机图形元文件(Computer Graphics Metafile)。作为一个编码矢量图形用于存储和传输的方法,CGM 很早就已经成为了矢量图形的国际标准。虽然 CGM 支持将矢量图形转换为人可读的文本,但它不是一个基于 XML 的标准。

XML 提供了一个方法,它可以定义一种可以在多个应用程序间共享并可直接插入到网页中的矢量图形的 Web 规范。结构化的矢量图形是 XML 的理想应用,在需求和使用方面都可以体现 XML 的实力。我们期望在不久的将来可以看到所有的主要矢量图形工具都可以

支持 XML 矢量图形文件格式。

34.2 XML 矢量图形支持

在本章的导言中,我提到了有三个规范在 XML 矢量图形的解决方案的列表中。下面是这三个 XML 词表以及支持它们的主要公司:

- 精确图形标记语言(Precision Graphics Markup Language, PGML)——Adobe
- 矢量标记语言(Vector Markup Language, VML)——Microsoft
- 可缩放矢量图形(Scalable Vector Graphics, SVG)——Adobe

你也许想知道为什么 Adobe 有两个用于矢量图形的不同 XML 词表。理由是他们开始提供的 PGML 是基于 Adobe 的 PostScript 图形模型。即使 PGML 是一个描述矢量图形的丰富的词表,Adobe 还是觉得需要一个用途更广,功能更强大的词表。它可以结合那些使得 Macromedia 的 Flash 如此成功的特性。因此,他们创建了一个全新的词表 SVG,它是一种对 PGML 和 VML 的综合。

由于 SVG 依据 PGML 和 VML,因而即使不必提及它是 Adobe 支持的,它也将极有可能成为 Web 上的矢量图形标准。然而,SVG 在本书编写时依旧还处于幼年期,因此让我预言这样的结果是不公平的。此外,在研究与之相竞争的词表方面还有许多要学的。本章将花一些时间来研究一些 VML 和 SVG。我们确实不用再提及 PGML 了,因为它现在已经被 SVG 代替了。

注意:要谨记本章的目标不是让你成为 VML 或 SVG 方面的专家。本章没有任何理由要仔细欣赏这些技术中的任何一个。当然,我们的目的是让你了解这两个词表,这样你就可以懂得它们是如何使用 XML 的,同时你也可以在使用 XML 创建矢量图形方面做出明智的决定。

34.2.1 矢量标记语言(VML)

矢量标记语言(Vector Markup Language, VML)是一个最初由 Microsoft 开发的 XML 词表,之后有 AutoDesk、Hewlett-Packard、Macromedia 和 VISIO 的加盟。VML 在 1998 年 5 月作为规范提交给 W3C。虽然这个规范已经存在了一段时间,但 VML 还需要在 Web 上赢得广泛的使用。实际上,Internet Explorer 5.0 是第一个支持 VML 的网络浏览器。其他浏览器都会忽略 VML 代码。然而,这是支持 VML 的最好的方面之一,因为没有任何一种其他矢量图形词表当前可以在浏览器级上被直接的支持。基本原因就是图形词表非常的新,这也就意味着要花些时间让它们被 Web 所容纳。

注意:你可以在 <http://www.w3.org/TR/NOTE-VML> 中看到提交给 W3C 的 VML 标准。

VML 支持广泛的矢量图形特性,它们基于由相连接的直线和曲线描述的路径。在 VML 中使用两个基本的元素:shape 和 group。这两个元素定义了 VML 的全部结构;shape 描述一个矢量图形元素,而 group 用来将这些图形结合起来,这样它们可以作为一个整体进

行处理。大多数图形通过路径描述,路径可以是轮廓图并且可以被填充。在本章稍后的“深入 VML 词表”一节你将学到更多有关 VML 图形结构的知识。

注意: VML 当前版本的一个缺陷是它不能直接支持动画。这并不是说你不能编写脚本来处理 VML 代码并创建动画,只是 VML 自身没有在词表中支持动画特征。

VML 从 Web 角度看还是一项比较新的技术,它已经在 Web 以外广泛使用了。这是因为 Microsoft 将 VML 结合进 Microsoft 的 Office 2000 中,在那里它用来作为在本地 Office 文件和 HTML 文件间图形传输的中间媒介。更重要的是,VML 允许 Office Art 的图形转换为本地 Office 文件格式和 HTML。它允许 Office 用户在这两个文件格式间毫不困难的进行转换。Office 用户获得的另一大好处就是剪切和粘贴的 VML 图形可以在应用程序间方便的共享。对于其他开始支持 VML 的非 Office 类应用程序,这种性能的有效性也将会得到提高。

就像大多数的 XML 词表一样,VML 是非常依赖于标准的,它从现有的 Web 标准中获得利益,这些标准有层叠样式表(CSS)和文件对象模型(DOM)。

34.2.2 精确矢量图形(SVG)

精确矢量图形(Scalable Vector Graphics,SVG)是由 Adobe 创建的 XML 词表,它似乎在更大的程度上震动了 Web。Adobe 通过创建 PGML 开始在 XML 词表上有所作为,这是一个基于 Adobe 的流行的 PostScript 技术的词表。SVG 提供了有关矢量图形的更为现代的方法,它直接处理了 Web 设计者和版面设计者的需求。SVG 使得创建有着成熟排版以及伸缩性、可转换性和高度紧凑的美术品的复杂层成为了可能。

SVG 在 1999 年 4 月作为规范提交给 W3C。虽然在本书编写期间还没有任何一个网络浏览器直接支持 SVG,但现在已经有了一个 Java applet 浏览器,并且 Adobe 承诺将在不久的将来会提供支持的插件(希望就是你在读这本书的时候)。如果一切顺利,最终 W3C 推荐使用 SVG,那么我将期望看到在所有主流浏览器的 6.0 版本中完全的支持对 SVG 的浏览。SVG 的最初实现可能将首先在 Adobe 的专业图形工具系列中看到(Photoshop、Illustrator 等等)。这将允许图形设计者在版面和网络应用程序间使用相同的插图,这是一个很大的好处。

对于 SVG 技术本身来说,其目标是通过支持图形搜索超越传统 Web 图形的多方面的限制,使用样式表,占据更小的存储空间并更易于印刷。SVG 还包括支持面板,变焦和灵活的层技术,SVG 与诸如 CSS2 这样的现有 Web 技术的无缝结合,以及完全的兼容 DOM,这样开发出了大量编写有趣的脚本的机会。

注意: ICC 彩色轮廓允许图形设备在给每个设备保留完整的本地色彩数据同时共享图形内容。ICC 就是国际色彩联盟,这是一个投身于软件色彩管理系统标准化的组织。

SVG 的一个独特的方面就是它支持动态内容,这一点我觉得很有趣。例如,SVG 允许你方便的创建旋转按钮。旋转的按钮在你用鼠标在它上面滑过的时候在外观上会有所改变。传统上,你将会使用脚本语言或 Java applet 来实现旋转按钮。SVG 将使得这项工作更为简单,它使你不用编程就可以创建旋转按钮。

可能你不知道,我对于 SVG 的这个可能的功能非常的兴奋。对我来说,它描绘了一个

“心愿表”技术——Adobe 通过仔细的研究设计者的需求完成它的任务,同时还有着现存矢量图形技术的功能。希望 SVG 在网络浏览器中的实现能够足够的简单并且我们将可以看到所有的图形工具都会迅速的支持它。

34.3 深入 VML 词表

虽然我承认在矢量图形的 XML 解决方案中更偏爱 SVG,但如果不给予 VML 充分介绍将是不公平的。此外,VML 有非常实际的优势,就是它可以被 Internet Explorer 5.0 完全的支持。因此不管你怎么为 SVG 着想,在本书编写期间它还不能被任何一个浏览器直接的支持,而 VML 可被 Internet Explorer 5.0 支持。因此,让我们花些时间了解一下 VML 词表。

Microsoft 并没有提供 VML 规范的完整 DTD,因此我不能给出它的清单。然而,你依然可以从部分的 DTD 中摘要的学习来了解 VML 元素。如果你全心全意要用 DTD 来验证 XML 文件,你可以提取在规范中列出的所有片断并自己创建一个 VML DTD。为防止你忘记,我重复一下,提交给 W3C 的 VML 规范位于 <http://www.w3.org/TR/NOTE-VML>。

VML 规范包括大量的支持多种不同矢量图形特性的元素。为了简化起见,我们将集中于 VML 的预定义图形。下面是在 VML 词表中预定义的图形元素,以及一些用来创建自定义图形的元素:

- shape
- path
- line
- polying
- curve
- rect
- roundrect
- oval
- arc
- group

在详细讨论创建 VML 图形之前,很值得注意 VML 非常依赖于名字空间的使用。用于 VML 的名字空间定义是 `urn:schemas-microsoft-com:vml`。在任何包含 VML 代码的文件中包含这个名字空间是十分重要的,这样网络浏览器就可以正确的处理代码了。下面是一个 html 标记的例子,它设置了 VML 名字空间并赋给前缀 `v`,这是一个推荐的前缀:

```
<html xmlns:v="urn:schemas-microsoft-com:vml">
```

如果你使用这个 html 标记,你就可以使用前缀 `v` 来引用所有的 VML 标记,同时网络浏览器会将它们识别为 VML 元素。

如果一个网络浏览器将元素识别为 VML,除非它访问 VML 处理函数,它将不知道如何处理它们。Internet Explorer 5.0 就包含这样的处理函数,但你仍然要在包含 VML 代码的 HTML 文件中确定它。

下面的代码确定 Internet Explorer 的 VML 处理函数为内嵌对象并使用 style 标记中的

behavior 声明对它进行调用：

```
<head>
  <object id="VMLRender"
    classid="CLSID:10072CEC-8CC1-11D1-986E-00A0C955B42E">
  </object>

  <style>
    v\:* {behavior:url(#VMLRender);}
  </style>
</head>
```

有了这里的代码,你现在已经准备好使用 VML 元素来创建矢量图形了。

34.3.1 shape 元素

shape 元素是 VML 中定义的基本的图形元素。这个元素允许你创建可视化的二维空间图形并仔细的结合它的属性。下面是对 shape 元素的定义：

```
<! ELEMENT shape (%shape.elements)* >
<! ATTLIST shape %coreattrs; %shapeattrs;
  type CDATA #IMPLIED
  adj CDATA #IMPLIED
  path CDATA #IMPLIED >
```

这个元素使用参数实体来包含子元素和属性集。清单 34.1 包含了这些参数实体的定义。

清单 34.1 ML 的参数实体定义

```
<! ENTITY % shape.elements
  (path | formulas | handles | fill | stroke | shadow | textbox |
  textapath | imagedata | %extensions;)>

<! ENTITY %coreattrs
  id ID #IMPLIED
  class CDATA #IMPLIED
  style CDATA #IMPLIED
  title CDATA #IMPLIED
  href CDATA #IMPLIED
  target CDATA #IMPLIED
  alt CDATA #IMPLIED
  coordsize CDATA #IMPLIED
  coordorigin CDATA #IMPLIED
  wrapcoords CDATA #IMPLIED>

<! ENTITY %shapeattrs
  opacity ID #IMPLIED
  chromakey CDATA #IMPLIED
  stroke CDATA #IMPLIED
  strokecolor CDATA #IMPLIED
  strokeweight CDATA #IMPLIED
```

| | | |
|-----------|-------|-----------|
| fill | CDATA | #IMPLIED |
| fillcolor | CDATA | #IMPLIED |
| print | CDATA | #IMPLIED> |

shape_element 参数实体定义 shape 的子元素。这些元素描述了图形的不同方面,比如路径和任何内嵌文字。coreattrs 参数实体定义了用于所有 VML 图形类型的核心属性。Coreattrs 中定义的两个重要的属性是 coordsize 和 coordorigin,它定义了矢量图的坐标系(一会儿你将学到更多有关图形坐标系的知识)。shapeattrs 参数实体定义了用于图形的特定属性。这些属性包括这样的图形属性:不透明性(透明度)、笔色、笔重(浓度)和填充色等等。

注意: VML 还支持 shapetype 元素,它与 shape 非常的类似。shapetype 提供描述图形的方法,这样这个图形就可以在 VML 中重复的使用。

本质上,shape 元素提供了一个装载由图形描述的矢量图形的方框。你使用 shape 元素的 coordsize 和 coordorigin 属性确定这个图形方框的坐标系。之后你使用 CSS2 的位置属性(left、top、width、height 等等)确定相关图形在这个坐标系中的位置。下面是一个在坐标系中创建一个空图形的例子:

```
<v:shape style="width:640px; height: 480px" coordsize="1000,1000"
  coordorigin="-500,-500">
</shape>
```

在这个例子中,图形的物理容器方框尺寸上是 640×480 像素。在这个方框中,坐标系确定有 1000×1000 个单位。这个坐标系 X、Y 轴的数值范围都为 -500 到 +500,因为 coordorigin 属性被设置为 -500,-500。Coordorigin 指定坐标系的左上角。坐标系(方框)的中心定位在 0,0,而 X 轴向右递增,Y 轴向下递增。当你使用 CSS2 定位属性确定图形的位置的时候,任何东西都以这个坐标系为准进行表示。VML 之后就负责将逻辑的坐标值映射为物理像素值。

VML 使用坐标系来辅助以更平稳的方式缩放图形。但图形在逻辑坐标系中被指定,这就有可能在不改变它们的值的情况下对它们进行缩放。换句话说,如果你修改图形的容器方框的尺寸,相关的任何图形数据的改变就是从逻辑坐标到像素坐标的映射的变化。

34.3.2 path 元素

path 元素定义图形的路径,它由一个有着相关路径数据的画笔命令(诸如点)组成。画笔命令确定路径中的点是如何相互连接的。路径可以由直线、弧、椭圆和 Bezier 曲线组成,它们可以左部不相连。下面是 path 元素的定义:

```
<! ELEMENT path (null)>
<! ATTLIST path>
  id ID #IMPLIED
  v CDATA #IMPLIED
  limo CDATA #IMPLIED
  fillok CDATA #IMPLIED
  strokeok CDATA #IMPLIED
```

| | | |
|-----------------|-------|----------|
| shadowok | CDATA | #IMPLIED |
| arrowok | CDATA | #IMPLIED |
| gradientshapeok | CDATA | #IMPLIED |
| textpathok | CDATA | #IMPLIED |
| textboxrect | CDATA | #IMPLIED |

注意: Bezier 曲线是使用特殊表达式数学定义的曲线。它根据法国工程师 Bezier 的名字命名,他将这种曲线用于 Renault 汽车的车体设计上。Bezier 曲线由两个结束点和一个或多个操作点组成,这些操作点位于曲线上并用来协助定义路径。

Path 元素最重要的属性是 *v*,它是一个矢量点与画笔命令相关。下面是一些与 *v* 属性相关的最为常用的画笔命令:

- *m*——移到一个新的点。
- *l*——到新的点画一条直线。
- *c*——到新的点画一条曲线。
- *x*——闭合当前子路径。
- *e*——结束当前子路径集

画笔命令接受与命令使用相关的参数(路径数据)。例如 *m* 和 *l* 命令接受确定新的点的 *x*、*y* 坐标的两个参数。*c* 命令接受描述一个三维 Bezier 曲线的六个参数。*x* 和 *e* 命令不接受任何参数。下面是在 *shape* 元素中使用 *path* 元素画出一个立方体图形的例子:

```
<v:shape style="width:300;height:300" strokecolor="black"
  strokeweight="2.0pt" fillcolor="red" coordsize="200,200"
  coordorigin="0,0">
  <v:path v="m 35,5 l 35,55,85,55,85,5,35,5,5,35,5,85,55,85,55,35,
    5,35 m 55,35 l 85,55 m 55,85 l 85,55 m 5,85 l 35,55 x e"/>
</v:shape>
```

注意 *m* 画笔命令用来将画笔移到坐标 35,5。*l* 命令用来画出链接一系列点的线。最后,*x* 和 *e* 命令用来闭合子路径和各自的闭合集,这个 VML 代码的结果如图 34.3 所示。

34.3.3 line 元素

line 元素简单的在一个点到另一个点之间画一条直线。*line* 元素的两个重要属性 *from* 和 *to* 用来指定标示这个直线的每一个坐标。下面是 *line* 元素的定义:

```
<! ELEMENT line (%shape.elements) * >
<! ATTLIST line %coreattrs; %shapeattrs;
  from      CDATA #IMPLIED
  to        CDATA #IMPLIED >
```

下面是一个使用 *line* 元素画出一条线的例子:

```
<v:line style="width:300;height:300" from="5,5" to="100,50"
  strokecolor="red" strokeweight="3.5pt"/>
```

这段 VML 代码的效果如图 34.4 所示。

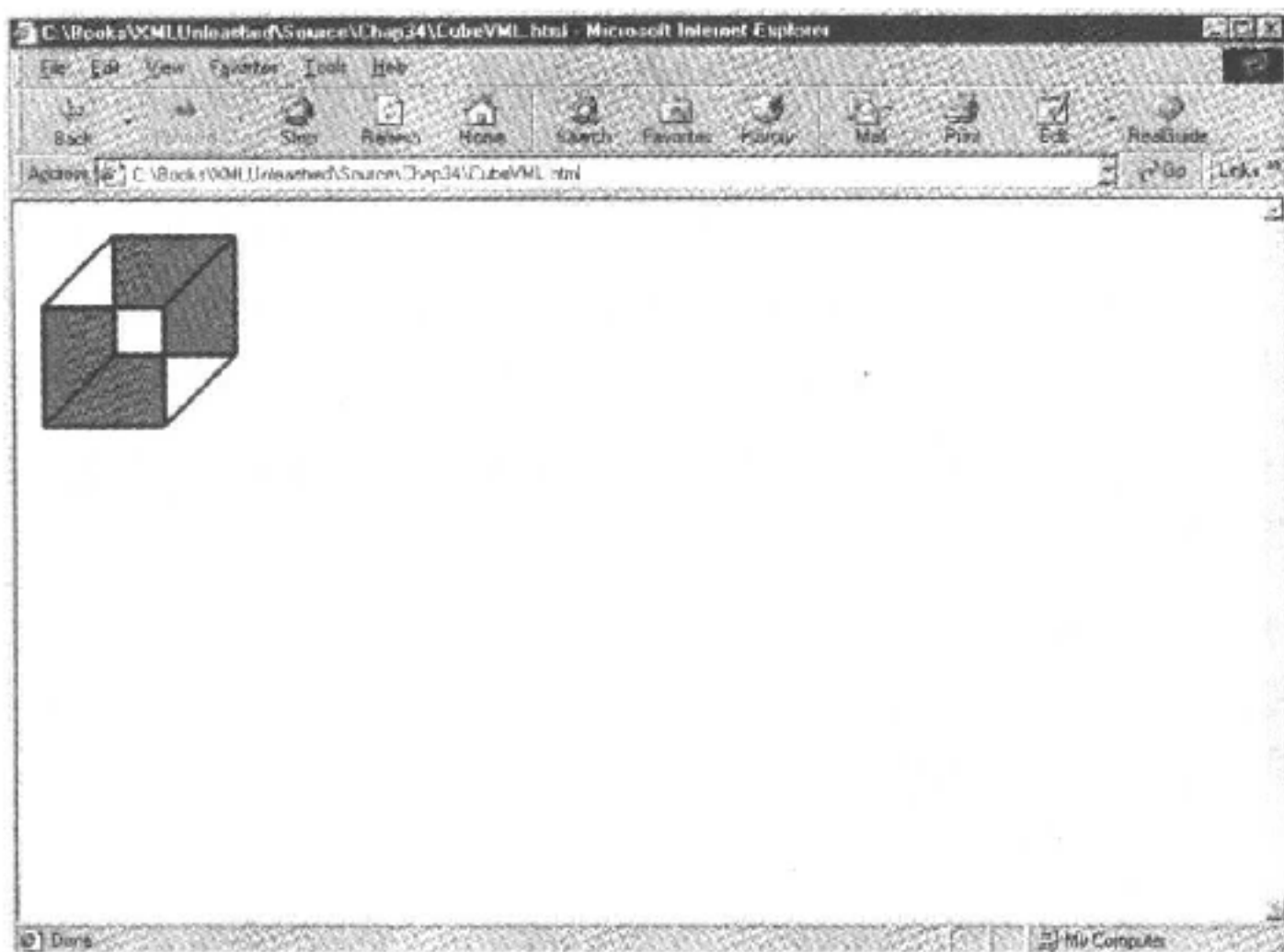


图 34.3 在 shape 元素中使用 path 元素画出的 VML 立方体图形

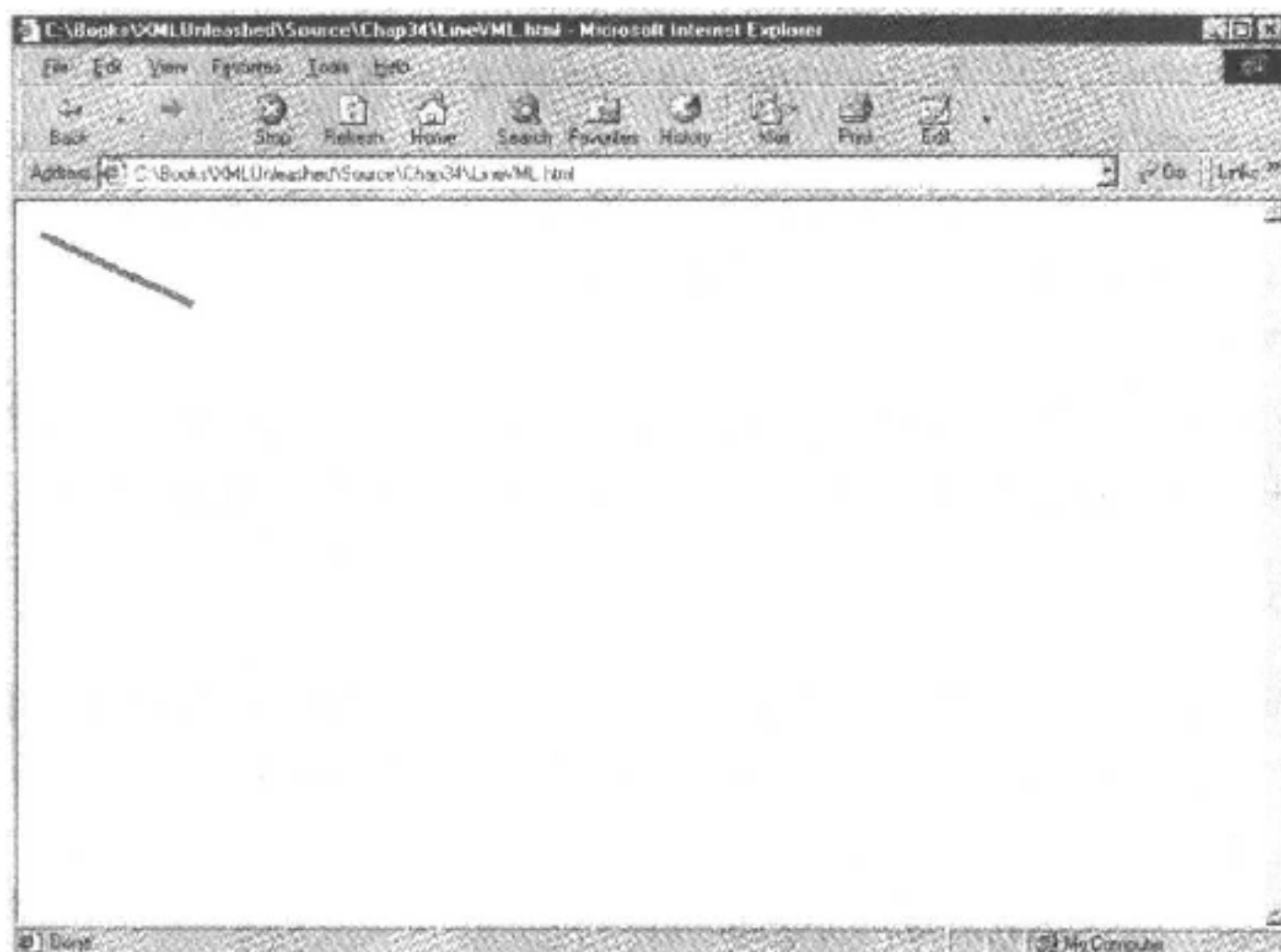


图 34.4 使用 line 元素画出的 VML 线

34.3.4 polyline 元素

polyline 元素用来画出一系列相连的线。points 属性用来指定相互连接的点的列表。下面是 polyline 元素的定义：

```
<! ELEMENT polyline (%shape.elements;) * >
<! ATTLIST polyline %coreattrs; %shapeattrs;
    points CDATA #IMPLIED>
```

下面是一个使用 polyline 元素画出一个折线图形的例子：

```
<v:polyline style="width:300; height:300" points="5.5, 100.50, 25.75, 35.125"
    strokecolor="blue" strokeweight="2.5pt" />
```

这段代码的效果如图 34.5 所示。

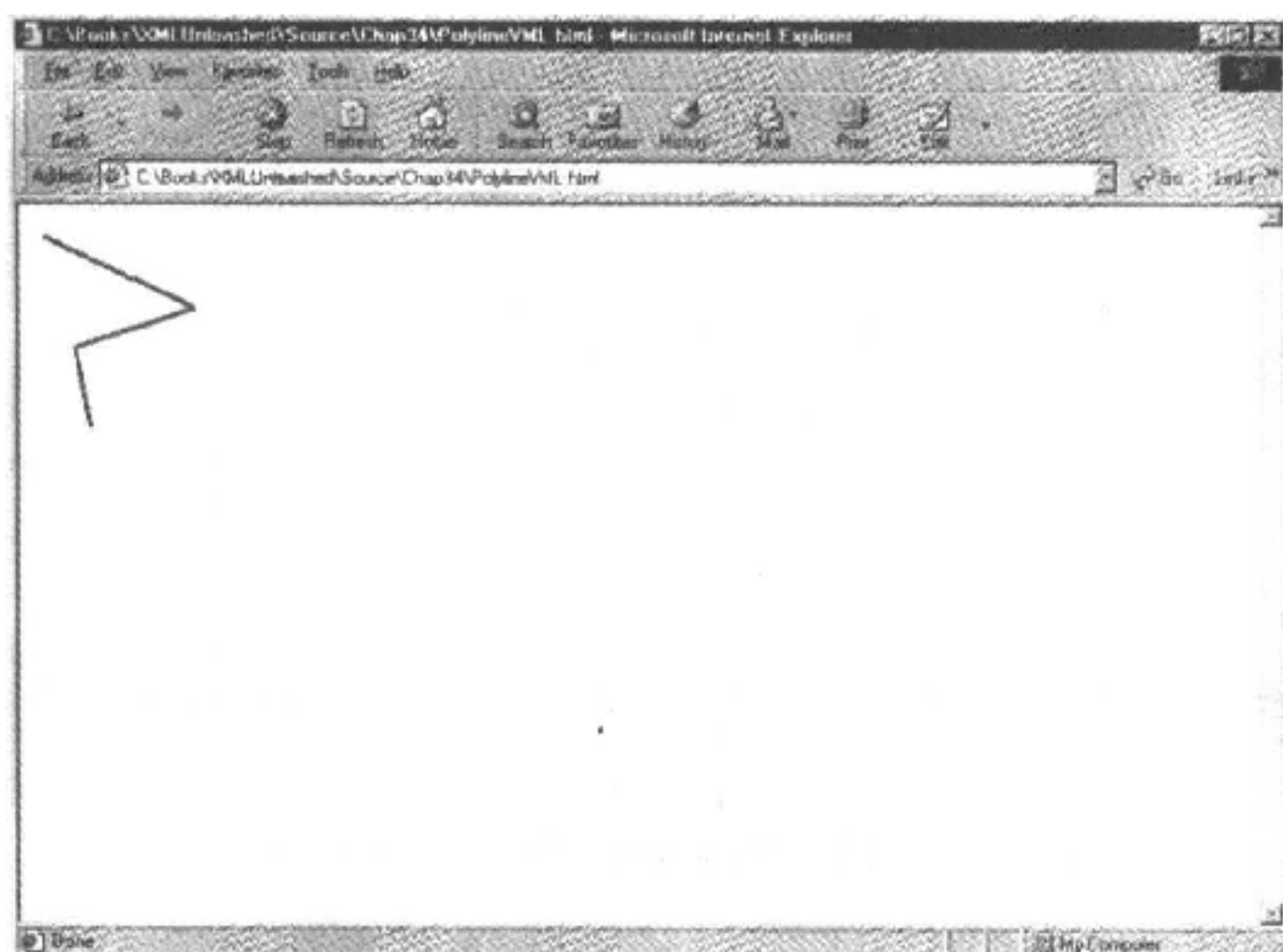


图 34.5 使用 polyline 元素画出的 VML 折线

34.3.5 curve 元素

curve 元素用来画出 Bezier 曲线。它使用两个属性 from 和 to 来描述曲线的起点和终点。curve 元素还包含另外两个属性, control1 和 control2。它们用来确定曲线的控制点。曲线根据这些线确定它从起点到终点的路径。

下面是 curve 元素的定义：

```
<! ELEMENT curve (%shape.elements;) * >
<! ATTLIST curve %coreattrs; %shapeattrs;
    from CDATA #IMPLIED
```

```
control1      CDATA #IMPLIED
control2      CDATA #IMPLIED
to            CDATA #IMPLIED >
```

下面是一个使用 curve 元素画出曲线图形的例子：

```
<v:curve style="width:300;height:300" from="5,5" control1="25,75"
control2="200,50" to="300,300" strokecolor="green"
strokeweight="2.5pt"/>
```

这段 vml 代码的效果如图 34.6 所示。

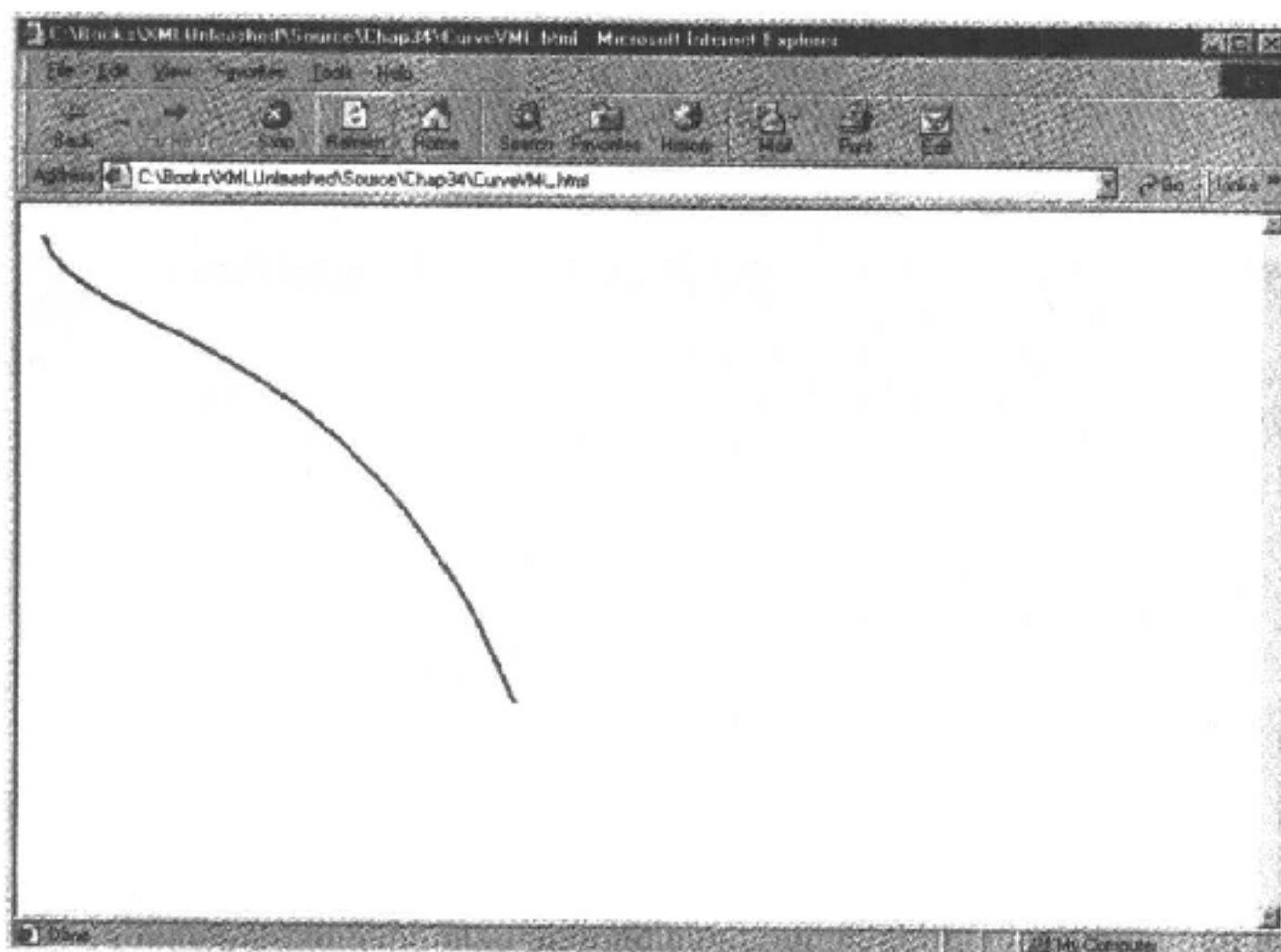


图 34.6 使用 curve 元素画出的 VML 曲线图形

34.3.6 rect 元素

rect 元素用来画矩形，它是最简单的预定义图形元素之一。这个元素没有定义属性，主要的属性从它的参数实体中继承。此外，rect 元素依赖于 CSS2 样式来确定它的宽和高。下面是 rect 元素的定义：

```
<! ELEMENT rect (%shape.elements;) * >
<! ATTLIST rect %coreattrs; %shapeattrs;>
```

下面是一个使用 rect 元素画出一个矩形的例子：

```
<v:rect style="width:150;height:250" fillcolor="blue" strokecolor="green"
strokeweight="3.5pt" />
```

这段 VML 代码的效果如图 34.7 所示。

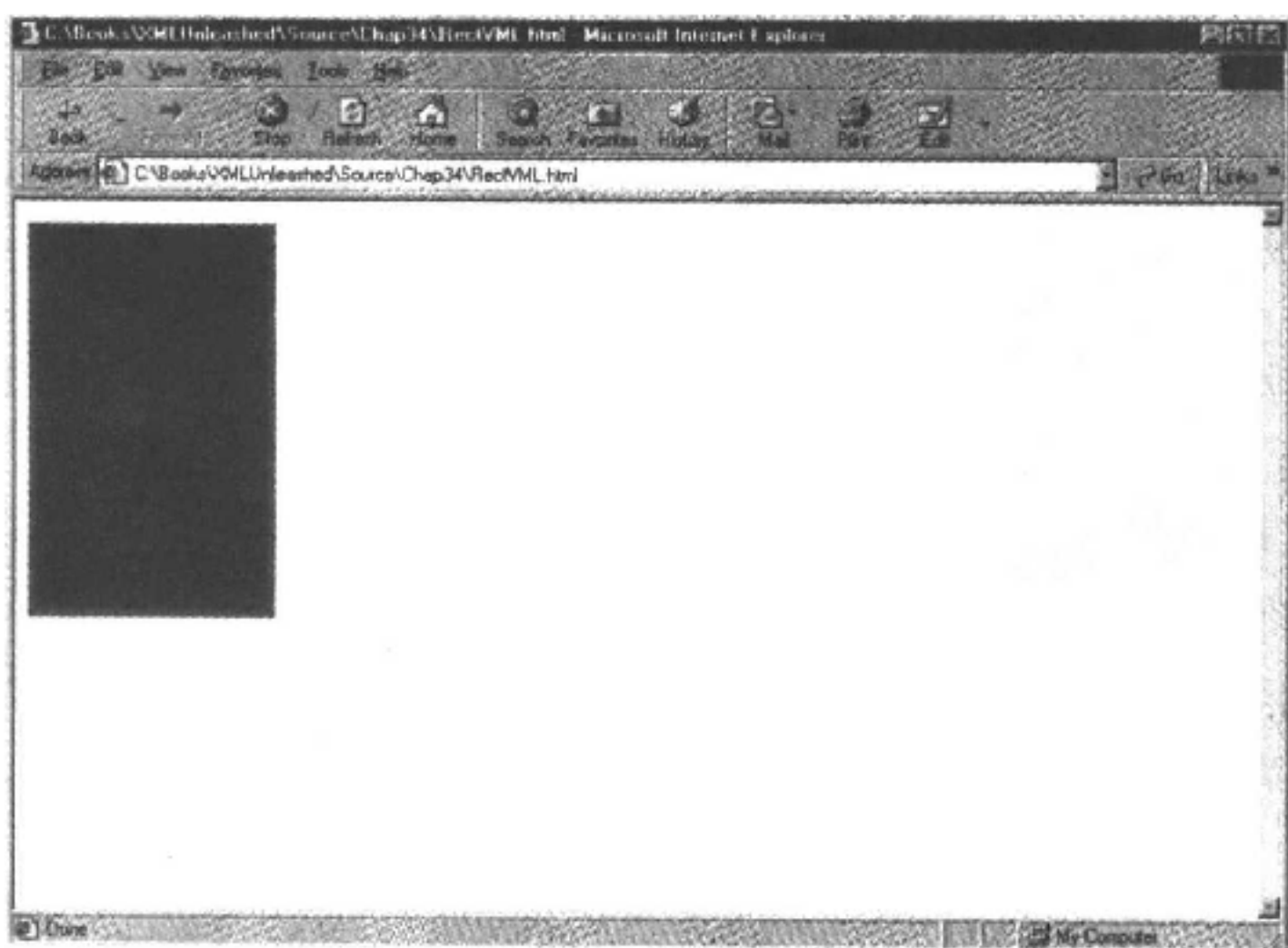


图 34.7 使用 rect 元素画出的 VML 矩形

34.3.7 roundrect 元素

roundrect 元素用来画出一个圆角的矩形。这个元素与 rect 元素非常类似,惟一的不同就是附加了 arcsize 属性,它用来指定圆角的曲线。arcsize 属性以矩形的较小边的百分比来表达曲线尺寸。因此一个值为 0.5 的 arcsize 表示一个直径为较小边长的一半的曲线。同样,值为 1.0 产生一个半径为较小边长度的曲线。最后,值为 0 表示一个直角的矩形。下面是 roundrect 元素的定义:

```
<! ELEMENT roundrect (%shape.elements;) * >
<! ATTLIST roundrect %coreattrs; %shapeattrs;
    arcsize CDATA #IMPLIED>
```

下面是一个使用 roundrect 元素画出一个圆角矩形图形的例子:

```
<v:roundrect stroke="width:150; height:250" arcsize="0.25" fillcolor="blue"
    strokecolor="green" strokeweight="3.5pt" />
```

这段 VML 代码的效果如图 34.8 所示。

34.3.8 oval 元素

oval 元素用来画椭圆和圆。这个元素与 rect 元素非常相似。oval 完全靠参数实体属性和 CSS2 样式。下面是 oval 元素的定义:

```
<! ELEMENT oval (%shape.elements;) * >
<! ATTLIST oval %coreattrs; %shapeattrs;>
```

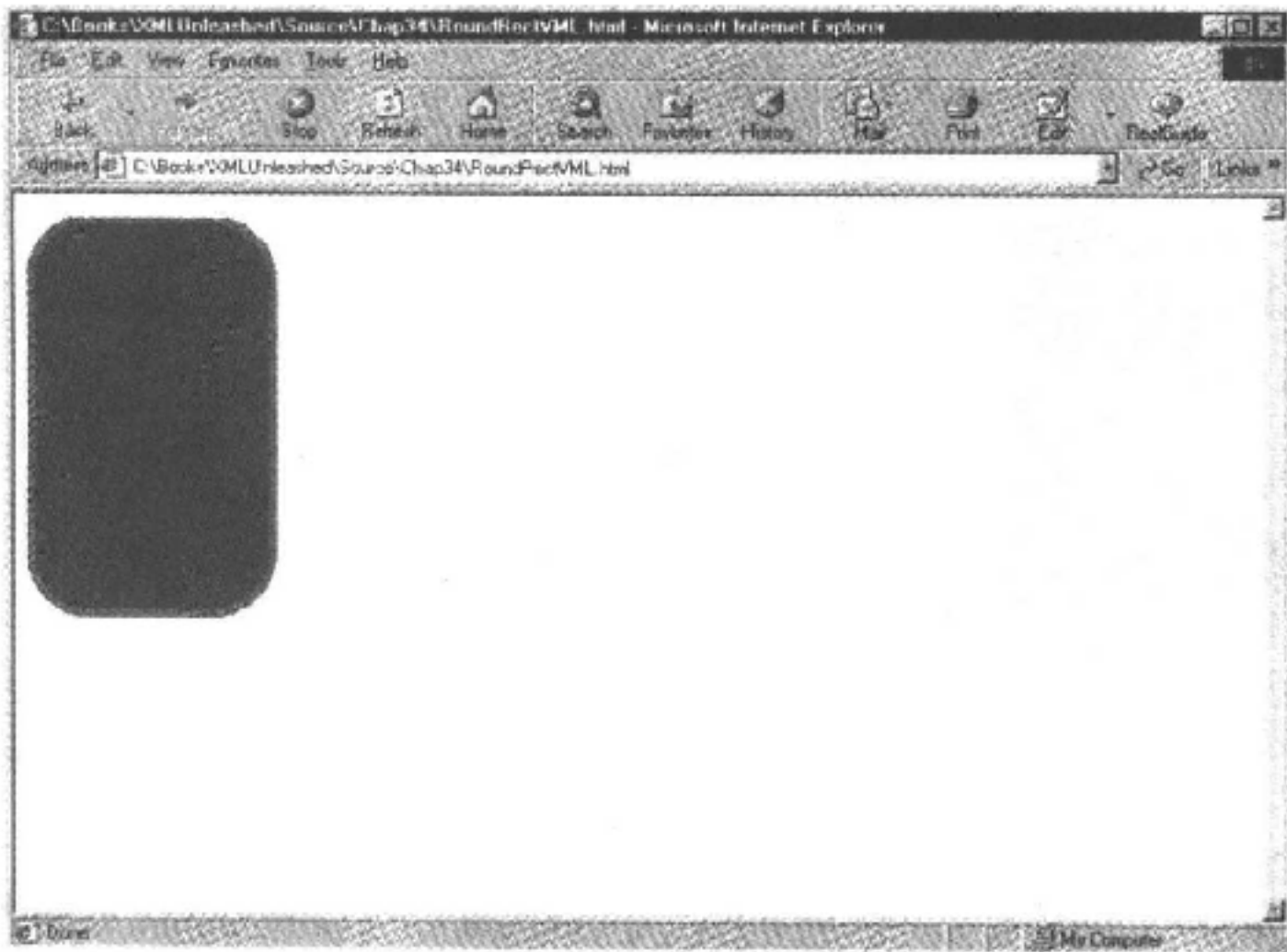


图 34.8 使用 roundrect 元素画出的 VML 圆角矩形

下面是一个使用 oval 元素画出椭圆图形的例子：

```
<v:oval style="width:150;height:250" fillcolor="yellow"
strokecolor="black" strokeweight="7.0pt"/>
```

这段 VML 代码的效果如图 34.9 所示。

34.3.9 arc 元素

arc 元素用来画弧线，实际上就是椭圆的一部分。你使用开始和结束角度指定一个弧线，这也就确定了椭圆中组成弧线的那一部分。这个角度与顺时针方向相一致，0 度指的是北方。下面是 arc 元素的定义：

```
<! ELEMENT arc (%shape.elements;) * >
<! ATTLIST arc %coreattrs; %shapeattrs;
startangle CDATA #IMPLIED
endangle CDATA #IMPLIED>
```

下面是使用 arc 元素画一个弧线图形的例子：

```
<v:arc style="width:150;height:250" startangle="45" endangle="180"
strokecolor="black" strokeweight="4.0pt" />
```

这段 VML 代码的效果如图 34.10 所示。

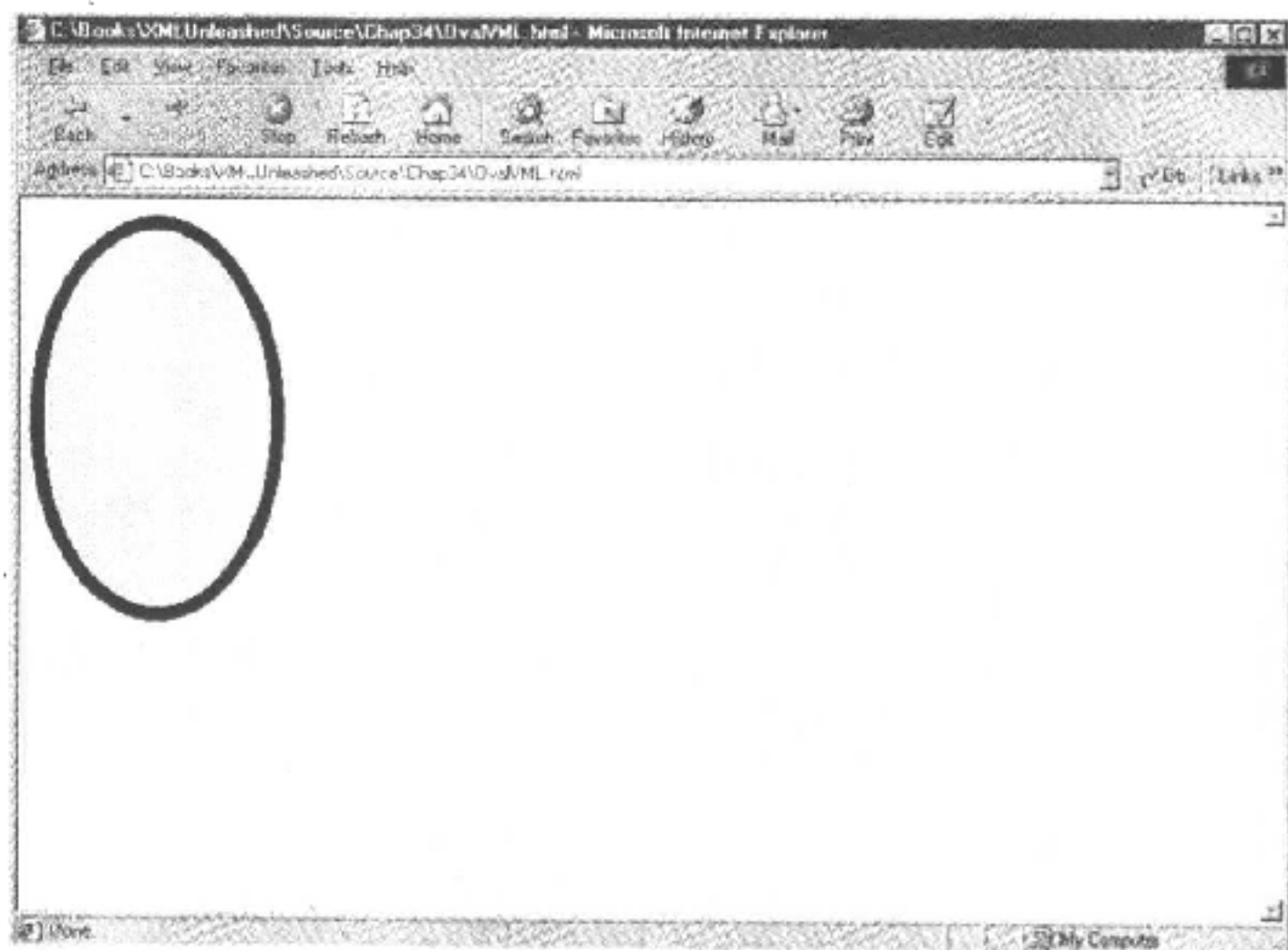


图 34.9 使用 oval 元素画出的 VML 椭圆

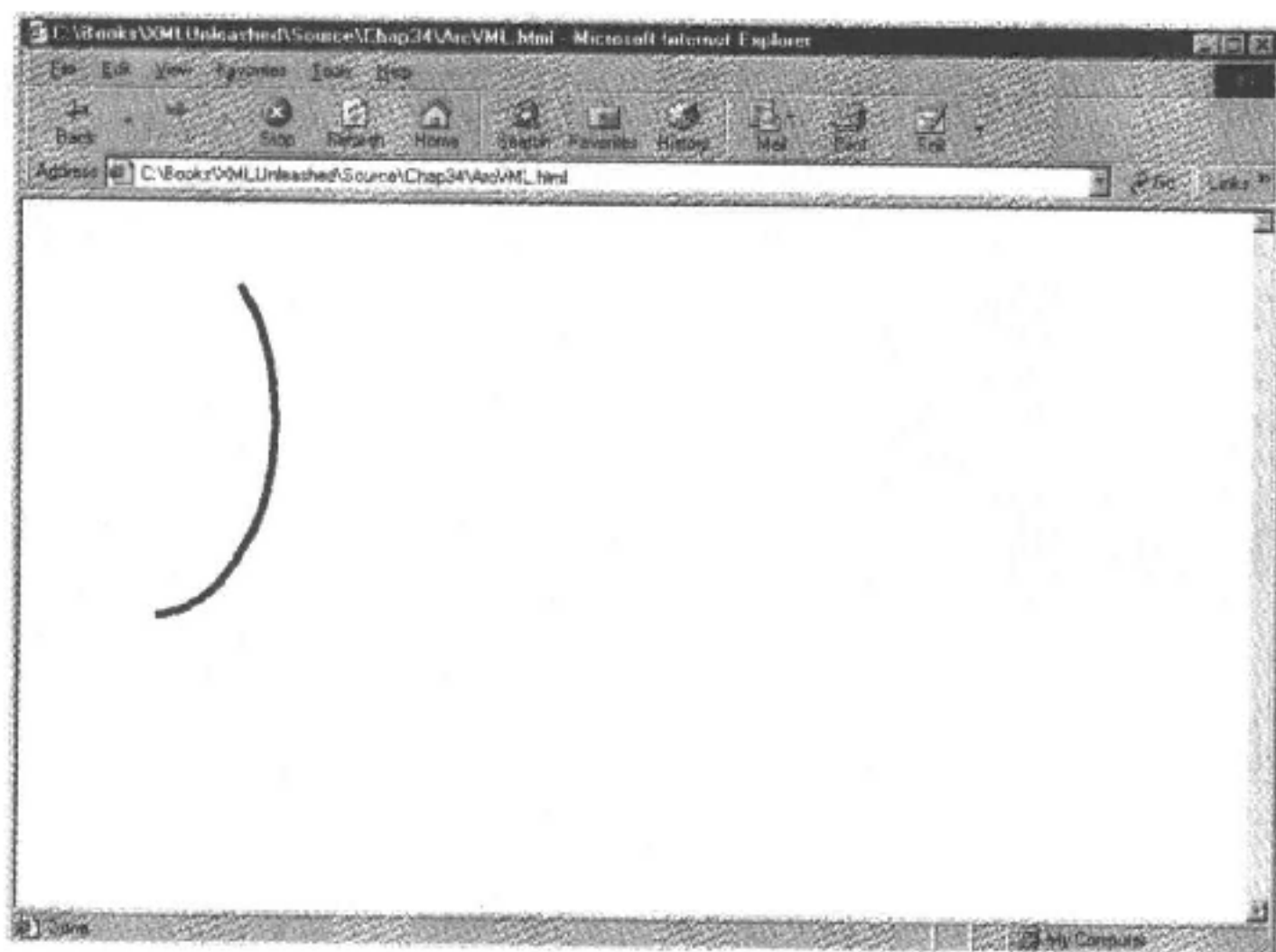


图 34.10 使用 arc 元素画出的 VML 弧线

34.3.10 group 元素

我们对 VML 快速遍历的最后一个元素是 group 元素,它用来结合元素。group 元素非

常便利,因为它允许你将一组元素作为单个实体进行缩放和变换。这个元素是一个容器元素,本身不定义任何的属性。

下面是 group 元素的定义:

```
<! ELEMENT group
  (group | shape | shapetype | line | polyline | curve | rect |
  roundrect | oval | arc | image) * >
```

下面是一个使用 group 元素将一些图形结合在一起的例子:

```
<v:group style="width:1800;height:1800">
  <v:curve style="width:300;height:300" from="5,5" control1="25,75"
    control2="200,50" to="300,300" strokecolor="green"
    strokeweight="2.5pt"/>
  <v:polyline style="width:300;height:300" points="5.5, 100,50,25,75,35,125"
    strokecolor="blue" strokeweight="2.5pt" />
  <v:shape style="width:300;height:300" strokecolor="black"
    strokeweight="2.0pt" fillcolor="red" coordsize="200,200"
    coordorigin="0,0">
    <v:path v="m 35,5 l 35,55,85,55,85,5,35,5,5,35,5,85,55,85,55,35,
      5,35 m 55,35 l 85,55 m 55,85 l 85,55 m 5,85 l 35,55 x e"/>
  </v:shape>
</group>
```

这段 VML 代码的效果如图 34.11 所示。

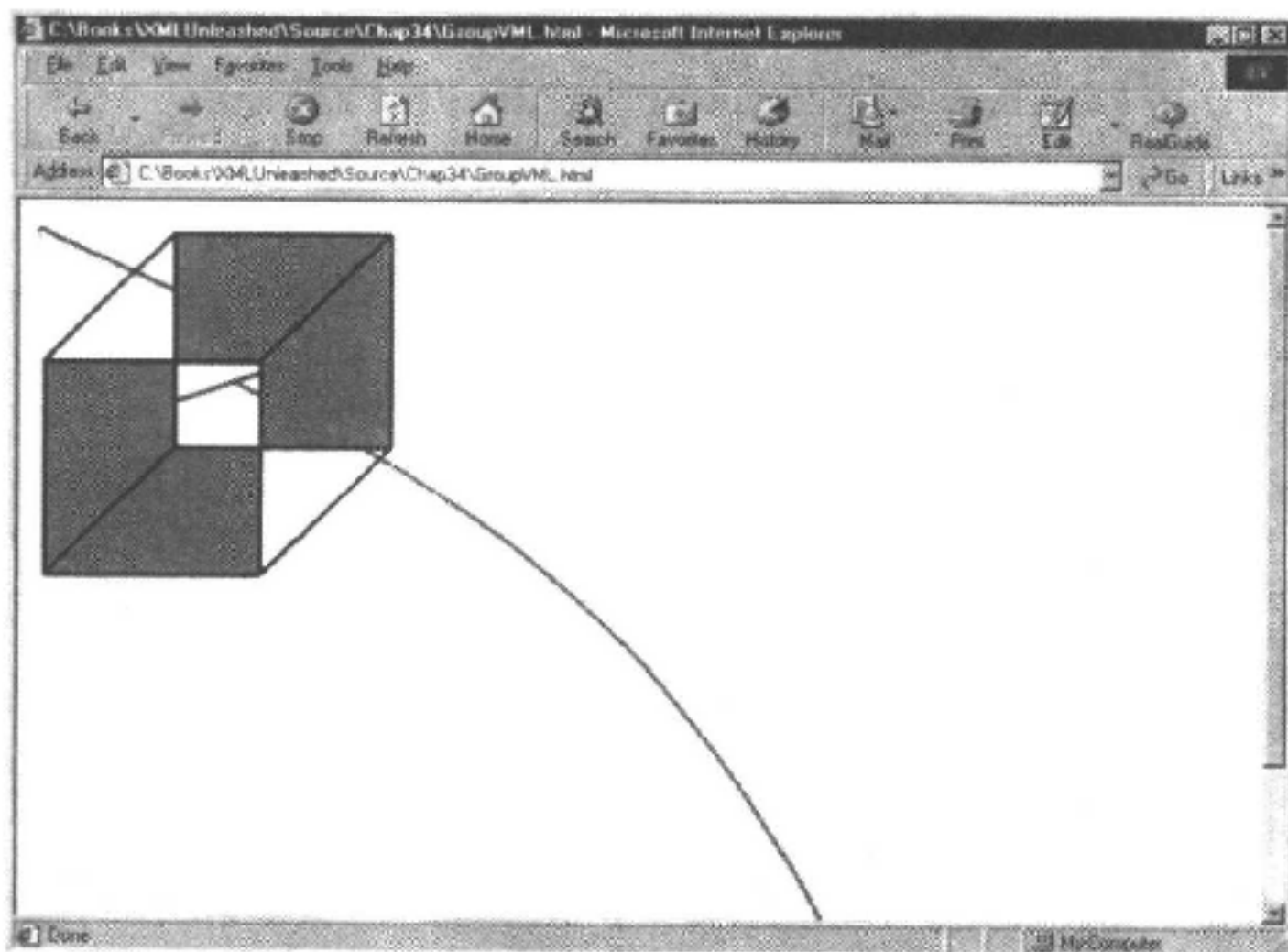


图 34.11 使用 group 元素画出的一组 VML 图形

group 元素的灵活之处在于,你只要简单的改变组的尺寸(宽和高)就可以缩放组中的所有图形。图 34.12 显示了将组尺寸变为 800×1500 后的组。

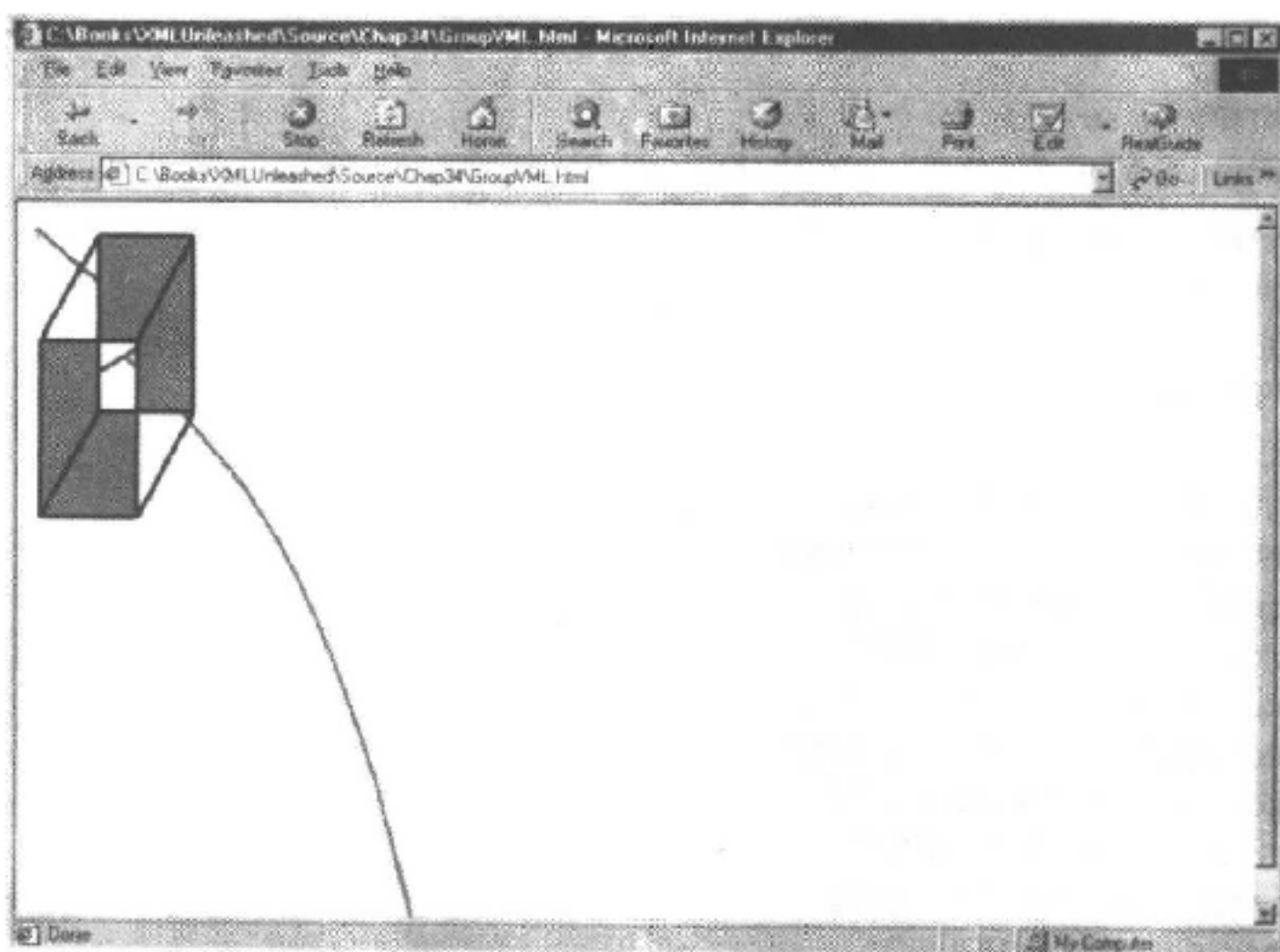


图 34.12 使用 group 元素画出的修改后的 VML 图形

34.4 深入 SVG 词表

SVG 词表在很多方面与 VML 类似。然而,SVG 是更为复杂的技术,它直接支持事件、动画、梯度填充、高级排版等等。大多数 SVG 的附加属性在你看到它的 DTD 时就会一目了然,它包含在清单 34.2 中。

注意:在本书编辑期间,SVG 规范仍未完成。例如,SVG 将如何支持动画的规范还处于概要阶段。因此,虽然清单 34.2 中的 DTD 描述了 SVG 词表的大部分关键特性,但请注意它仍然处于发展阶段。

清单 34.2 SVG 的文件类型定义(DTD)

```
<! ENTITY % namespace
  "xmlns CDATA #FIXED 'http://www.w3.org/Graphics/SVG/SVG-19990325'">

<! ENTITY % class
  "class NMTOKENS #IMPLIED">

<! ENTITY % id
  "id ID #IMPLIED">

<! ENTITY % lang
  "xml:lang NMTOKEN #IMPLIED">

<! ENTITY % style
  "Style CDATA #IMPLIED">

<! ENTITY % global
```

```

    " %class;
    %id;
    %style;">
<! ENTITY % str-text
    "content CDATA # FIXED 'structured text'">
<! ENTITY % global-t
    " %class;
    %id;
    %lang;
    %style;">
<! ENTITY % g-eventhandlers
    "mousedown CDATA # IMPLIED
    mouseup CDATA # IMPLIED
    onclick CDATA # IMPLIED
    mouseover CDATA # IMPLIED
    mousemove CDATA # IMPLIED
    mouseout CDATA # IMPLIED
    keydown CDATA # IMPLIED
    keypress CDATA # IMPLIED
    keyup CDATA # IMPLIED
    onload CDATA # IMPLIED
    onselect CDATA # IMPLIED"
<! ENTITY % r-eventhandlers
    "Onunload CDATA # IMPLIED
    onzoom CDATA # IMPLIED">
<! ENTITY % system-required
    "system-required CDATA # IMPLIED">
<! ENTITY % std-g-attrs
    " %global;
    %system-required;
    %g-eventhandlers;">
<! ENTITY % std-g-attrs-t
    "global-t;
    %system-required;
    %g-eventhandlers;">
<! ENTITY % replaced
    "xml:link CDATA # FIXED 'simple'
    show CDATA # FIXED 'embed'
    actuate CDATA # FIXED 'auto'
    href CDATA # REQUIRED ">
<! ENTITY % hyperlink
    "xml:link CDATA # FIXED 'simple'
    show CDATA # FIXED 'embed'
    actuate CDATA # FIXED 'auto'
    href CDATA # REQUIRED ">
<! ENTITY % hyperlink
    "xml:link CDATA # FIXED 'simple'
    show CDATA # FIXED 'replace'
    autuate CDATA # FIXED 'user'
    href CDATA # REQUIRED ">

```



```

<! ENTITY % symbol_descriptor_attributes
    "min-x CDATA #IMPLIED
    "min-y CDATA #IMPLIED
    "max-x CDATA #IMPLIED
    "max-y CDATA #IMPLIED
    "ref-x CDATA #IMPLIED
    "ref-y CDATA #IMPLIED

<! ENTITY % shapes
    "rect|circle|ellipse|polyline|polygon|line">

<! ENTITY % g_elements
    "(defs?,title?,desc?,(use|image|text|path| %shapes, \g|switch|svg|a) * )">

<! ELEMENT svg %g_elements;?
<! ATTLIST svg
    %namespace;
    %std-g-attrs-t;
    %r-eventhandlers;
    %symbol_descriptor_attributes;
    width CDATA #REQUIRED
    height CDATA #REQUIRED
    allow-zoom-and-pan (true | false) "true">

<! ELEMENT g %g_elements; >
<! ATTLIST g
    %std-g-attrs-t;?

<! ELEMENT switch %g_elements; >
<! ATTLIST switch
    %std-g-attrs-t; >

<! ELEMENT defs (script|style|private|lineargradient|radialgradient|
    symbol|image|text|textflowpath| %shapes; ) * >
<! ATTLIST defs
    %global-t; >

<! ELEMENT private ANY >
<! ATTLIST>

<! ELEMENT title( #PCDATA ) * >
<! ATTLIST title
    %global-t;
    %str-text; >

<! ELEMENT desc( #PCDATA ) * >
<! ATTLIST desc
    %std-g-attrs-t;
    %str-text; >

<! ELEMENT use EMPTY >
<! ATTLIST use
    %std-g-attrs;
    x CDATA #IMPLIED
    y CDATA #IMPLIED
    width CDATA #IMPLIED
    height CDATA #IMPLIED
    %replaced; >

<! ELEMENT marker EMPTY >
<! ATTLIST marker

```

```

    %global;
    %replaced;
    width CDATA #IMPLIED
    height CDATA #IMPLIED>

<! ELEMENT image (desc?,title?) >
<! ATTLIST image
    %std-g-attrs-t;
    x CDATA #IMPLIED
    y CDATA #IMPLIED
    %replaced;>

<! ELEMENT text (#PCDATA|src|tf|textpath) * >
<! ATTLIST text
    %std-g-attrs-t;
    x CDATA "0"
    y CDATA "0">

<! ELEMENT textpath EMPTY >
<! ATTLIST textpath
    start-offset CDATA "0"
    %replaced;>

<! ELEMENT path (data|marker) * >
<! ATTLIST path
    %std-g-atts;
    d CDATA #REQUIRED
    flatness CDATA #IMPLIED
    nominal-length CDATA #IMPLIED>

<! ELEMENT data EMPTY >
<! ATTLIST data
    d CDATA #REQUIRED >

<! ELEMENT rect EMPTY >
<! ATTLIST rect
    %std-g-attrs;
    x CDATA "0"
    y CDATA "0"
    width CDATA #REQUIRED
    height CDATA #REQUIRED>

<! ELEMENT circle EMPTY >
<! ATTLIST circle
    %std-g-attrs;
    cx CDATA "0"
    cy CDATA "0"
    r CDATA #REQUIRED>

<! ELEMENT ellipse EMPTY >
<! ATTLIST ellipse
    %std-g-attrs;
    cx CDATA "0"
    cy CDATA "0"
    major CDATA #REQUIRED
    minor CDATA #REQUIRED
    angle CDATA "0">

<! ELEMENT polyline EMPTY >
<! ATTLIST polyline

```

```

    %std-g-attrs;
    verts CDATA #REQUIRED>
<! ELEMENT polygon EMPTY >
<! ATTLIST polygon
    %std-g-attrs;
    sides CDATA "3"
    angle CDATA "0"
    r CDATA #REQUIRED>
<! ELEMENT line EMPTY >
<! ATTLIST line
    %std-g-attrs;
    x1 CDATA "0"
    x2 CDATA "0"
    y1 CDATA "0"
    y2 CDATA "0">
<! ELEMENT foreignobject (#PCDATA) * >
<! ATTLIST foreignobject
    %std-g-attrs-t;
    width CDATA #REQUIRED
    height CDATA #REQUIRED
    %str-text;>
<! ELEMENT script (#PCDATA) * >
<! ATTLIST script language CDATA "text/ecmascript">
<! ELEMENT style (#PCDATA) * >
<! ATTLIST style type CDATA #FIXED "text/css">
<! ENTITY % gradient. attrs
    "target-type CDATA #IMPLIED
    target-left CDATA #IMPLIED
    target-top CDATA #IMPLIED
    target-right CDATA #IMPLIED
    target-bottom CDATA #IMPLIED " >
<! ELEMENT lineargradient (gradientstop) * >
<! ATTLIST lineargradient
    id ID #IMPLIED
    vector-start-x CDATA #IMPLIED
    vector-start-y CDATA #IMPLIED
    vector-length CDATA #IMPLIED
    vector-angle CDATA #IMPLIED
    spread-method CDATA #IMPLIED
    %gradient.attrs;>
<! ELEMENT radialgradient (gradientstop) * >
<! ATTLIST radialgradient
    id ID #IMPLIED
    innermost-x CDATA #IMPLIED
    innermost-y CDATA #IMPLIED
    outermost-x CDATA #IMPLIED
    outermost-y CDATA #IMPLIED
    outermost-radius CDATA #IMPLIED
    matrix CDATA #IMPLIED
    %gradient.attrs;>
<! ELEMENT gradientstop EMPTY >

```

```

<! ATTLIST gradientstop
  %style;
  id ID #IMPLIED
  offset CDATA #REQUIRED
  color CDATA #REQUIRED>

<! ELEMENT symbol %g-elements;>
<! ATTLIST symbol
  %global-t;
  %symbol-descriptor-attributes;>

<! ELEMENT src EMPTY >
<! ATTLIST src
  %replaced;>

<! ELEMENT textflow (t|text) * >
<! ATTLIST textflow
  %std-g-attrs-t;
  %str-text;>

<! ELEMENT t EMPTY >
<! ATTLIST t
  %replaced;>

<! ELEMENT tf EMPTY >
<! ATTLIST tf
  %replaced;>

<! ELEMENT a ANY >
<! ATTLIST a
  %hyperlink;>

```

与 VML 类似,SVG 规范包含大量的元素。为了使事情更容易理解,我们将主要集中在 SVG 的预定义图形上。下面是在 SVG 词表中预定义的 SVG 词表以及一些用来管理矢量图形的元素:

- defs
- path
- line
- polyline
- polygon
- curve
- rect
- circle
- ellipse
- lineargradient
- radialgradient
- gradientstop
- g

当前还没有可以测试例子的可用实现,因此我们就没必要给你讲解一大堆示例代码了。

此外,我将提供一些例子,让你在 SVG 文件是如何构造的方面能得到一个大概的认识。

注意:技术上,现在有一个可用的 SVG 实现,但在本书编辑时它还不能完全的应用。这个实现就是名为 `svgImage` 的 Java 部件。它可以显示 SVG 文件,主要依赖于 IBM 的 XML 解析器来处理 and 显示它们。你可以从 Black Dirt Software 站点中 (<http://www.blackdirt.com/graphics/svg/>) 下载 `svgImage`。IBM 的 Java XML 解析器可以从 <http://www.alphaworks.ibm.com/formula/xml> 中下载。

就像 VML 一样,SVG 依赖于名字空间的使用。你可以将名字空间设置给一个诸如 `svg` 这样的前缀,或不用前缀声明而默认 SVG 名字空间。用于 XML 的名字空间定义就是 SVG 的 DTD: <http://www.w3.org/Graphics/SVG/svg-19990412.dtd>。下面是一个将 SVG 的 DTD 作为默认的名字空间的 SVG 文件框架结构的例子:

```
<? xml version="1.0"? >
<! DOCTYPE svg PUBLIC ".//W3C//DTD SVG APRIL 1999//EN"
"http://www.w3.org/Graphics/SVG/svg-19990412.dtd">
<svg width="6in" height="4in">
  <!-- Vector graphic elements go here! -->
</svg>
```

注意,SVG 文件的根元素就是 `svg` 元素。这个元素用来做为所有的包含在文件中的 SVG 图形的容器。我们也有可能在父文件中嵌入 SVG 代码,这种情况下 SVG 元素仍然用来承载 SVG 代码。下面是一个 SVG 代码该如何放入到 HTML 文件中的例子:

```
<html xmlns:svg="http://www.w3.org/Graphics/SVG/svg-19990412.dtd">
  <head>
  </head>
  <body>
    <svg width="6in" height="4in">
      <rect x="15" y="20" width="250" height="180"/>
    </svg>
  </body>
</html>
```

这段代码画出了一个矩形。你也许注意到,虽然语法与 VML 有些不同,它仍然易于理解。其他的基本图形也很容易画出。下面是一个用来画填充为绿色的黑圆所需要的代码:

```
<circle style="fill:green;stroke:black"
cx="25" cy="50" r="150"/>
```

我将把所有其他的基本 SVG 图形的细节留给你自己去研究。这是你明确希望通过自己进行试验的事情。

34.5 总 结

如你所知,Web 上的大多数图形都是由很多细小的颜色像素组成的点阵图形(位图)结

构。点阵图形对于近似照片上的大量的色彩当然是非常优秀的,但它们对于所有的图形来说并不是理想的。它们占据了大量的空间,耗费大量的时间进行下载。点阵图形在缩放或转换时还会有损质量。矢量图形提供了解决点阵图形的大部分问题的方案,它们正在逐渐变为 Web 图形的流行格式。XML 是促进矢量图形标准的优秀技术。

本章探讨了两个处于领导地位的基于 XML 的矢量图形词表:VML 和 SVG。到目前为止,你只是匆忙的深入了一下 VML 和 SVG 的词表。我并不企图对词表进行详尽彻底的研究(如果我这样做了你可能就需要一台升降机来搬运书籍了)。此外,我尝试提供一些 XML 是如何用来确定矢量图形标准这个问题的看法。我们并不很清楚哪一个词表会成为支配性的标准。我觉得 SVG 有更多的潜力,且在设计和特征集方面具备完全性,但 VML 也有重要的优势,因为它已经被 Internet Explorer 5.0 所支持。我们将拭目以待!

第 35 章 虚拟现实和 3DML

本章的题目可能会使你很想知道 XML 与虚拟现实有什么可能的关系。说实在的,虚拟现实确实似乎是 XML 的一种奇怪的应用。然而,当你认识到 XML 纯粹就是一个描述信息的方法的时候,你就不会这么认为了。因为虚拟世界实际上就是大量运动的数据点。就如你将要在本章所学到的,XML 实际上提供了模拟 3D 世界的很有趣的方法。

本章将向你介绍 3DML(3D Modeling Language, 3-D 建模语言),这是一个标记语言,它允许你使用标识标记设计名为场景(spots)的虚拟世界。你将发现 3DML 在很多方面非常的简单,而这也恰恰是它的强大之处,因为它非常易于理解和使用。几分钟后,你就可以创建有着图形纹理、光影效果和 3D 声音的虚拟世界。

35.1 DML 基础

3DML 是由 Flatlang Online 创建的标记语言,它允许你使用标记创建 3D 世界。3DML 世界名为场景(spots),它可以由多个层次组成。你也许想知道 3DML 如何与 VRML(Virtual Reality Modeling Language, 虚拟现实建模语言)比较。这是另一个用来描述虚拟世界的语言。让我们花一些时间考虑一下 VRML 以及它与 3DML 的关系。

用负责监视 VRML 的 Web3D 联盟的话说,“VRML 是一个 3D 多媒体的开放标准并在 Internet 上共享虚拟世界”。VRML 已经存在了一些年,它要早于 XML,因此也就不能认为是一种 XML 的应用。这是 VRML 和 3DML 之间的根本区别。另一个巨大的区别就是 VRML 要比 3DML 功能强大的多。当然,这也使得 VRML 要复杂的多并且比 3DML 更难学习。

注意:在 90 年代早期,Web3D 联盟宣布开始将 VRML 的特征集移植到一个名为 X3D 的 XML 应用中。X3D,也就是可扩展 3D(eXtensible 3D),在本书编写期间仍然处于早期规范阶段。请访问 Web3D 联盟的站点 <http://www.vrml.org> 来获取有关 VRML 和 X3D 的更多信息。

3DML 背后的主要思想就是使得创建可在线存在的 3D 场景变得非常的简单。为了实现这个目标,3DML 设计的比较简单,这起码是与其他虚拟现实技术——比如 VRML——相比而言。然而,3DML 的简单性也使它不管是在图形性能方面还是在带宽需求方面,都非常的有效率。换句话说,3DML 在用调制解调器上网的慢速计算机上可以工作的非常好(我在一台奔腾 133 的计算机上测试了一下你将在本章建立的 3DML 世界,它运行得非常流畅)。

3DML 的性能是以削减了一些图形复杂性为代价的。3DML 的最明显的绑定就是场景的基本结构。3DML 中的场景由一组立方体或是块组成,它们每一个的尺寸都是 $256 \times 256 \times 256$ 像素。当你定义了一个 3DML 场景,你要指定在标记场景的每个 3D 坐标轴上有多少

个块。首先你指定 X 轴和 Y 轴上块的数目。然后你指定属于 Z 轴的垂直层的数目。这样一个场景就由在行、列和层上排列的固定数目的块组成了。

你通过指定将哪种类型的块放在每个块空间中,构造一个场景。要理解的关键是 3DML 场景中的任何东西都是由块组成的,这样作非常的灵活。Flatland 甚至提供附加的块集,这样你可以结合不同种类的块来使用。例如,除了基本块集以外,这里还有包含诸如门、窗户和小路这样的自然结构的乡村块集。

你也许渴望了解用户是如何浏览 3DML 场景的。Flatland Online 提供了一个名为 Rover 的 3DML 浏览器,这是一个基于 Windows 的 PC 插件,用于 Netscape Navigator 3.0 及以上版本、Microsoft Internet Explorer 4.0 及以上版本和 American Online 4.0。你可以在 <http://www.flatland.com/download> 上免费下载 Rover。我们并不清楚浏览是否会内建对 3DML 的支持。如果浏览器内建此功能的话,3DML 将变成一个非常流行的技术。同时,Rover 插件非常容易安装并且在浏览 3DML 场景方面工作得非常优秀。

通过学习在 3DML 词表中定义的不同元素,你将可以理解 3DML 场景的结构。让我再提提你的兴趣,图 35.1 显示了由 Flatland Online 创建的 Spot Springs 场景,这是对 3DML 的展示。



图 35.1 Flatland Online 的 Web 站点上的 Spot Springs 3DML 展示场景

35.2 深入 3DML 词表

在我们开始详细讨论 3DML 词表之前,我要向你解释清楚有关 3DML 的技术问题。这并不简单,但是现在说明一下:3DML 实际上并不是一个 XML 的应用。这里,我给出了这个结论,但我希望你不要用这个来批驳我并且认为整部书都是谎言。实际是 3DML 与一个实

实际的 XML 词表令人难以置信的接近,但是它违反了足够多的 XML 规则,因此不能被认为是一个真正的 XML 应用。例如,你可以在 3DML 文件中使用和号(&)。同时,3DML 不支持实体或默认属性值。还有就是 3DML 中的元素和属性并不是事件敏感的,这在 XML 中是一个巨大的问题。

为什么我在这样一本全部是有关 XML 技术的书中要讲解这样的词表呢?好,老实的回答是 3DML 太酷了,不能不被提及。3DML 是在 XML 成为主流前构思的,这也就是为什么它并不完全遵守 XML 的规则。然而,我希望 Flatland Online,3DML 的开发能认识到适应 XML 的巨大好处,并确定一种方法使 3DML 的未来版本能够是一个成熟的 XML 应用程序。同时,你将发现 3DML 的外观和感觉与其他的 XML 词表非常的类似。现在,让我们开始探讨 3DML 词表,看一看场景是如何构造的。3DML 词表主要基于三个基本元素:

- SPOT——3DML 文件的根元素,或称文件元素。因此,你总要将 3DML 内容放在<SPOT>和</SPOT>标记包围的区域内。
- HEAD——它与 HTML 中的 HEAD 元素的作用非常类似,用来提供一个定义用于全文的头信息的方便位置。场景的头信息的例子包括尺寸、环境光和天空纹理等等。
- BODY——它包含所有的有关场景的指定信息。3DML 文件的 BODY 部分包含对场景的映射描述,以及块定制。

为了更好的理解 3DML 场景是如何构造的。更详细的考虑一下这些基本的 3DML 元素将是很有帮助的。在本章后面,你将使用这些元素构造一个 3DML 场景。

35.2.1 SPOT 元素

SPOT 元素是 3DML 文件的文件元素,它作为其他的两个基本元素 HEAD 和 BODY 的容器元素。3DML 文件的基本框架结构如下所示:

```
<SPOT>
  <HEAD>
  </HEAD>
  <BODY>
  </BODY>
</SPOT>
```

如你所见,HEAD 和 BODY 元素包含在 SPOT 元素中。场景的性质和映射信息在 HEAD 和 BODY 中定义,稍后你将会看到。

35.2.2 HEAD 元素

HEAD 元素提供了一个确定场景性质的方便位置,这些性质比如有场景的块尺寸。这个元素支持一些不同的子元素,使用它们你可以填充场景的性质。下面是出现在 3DML 文件首部的最常用子元素:

- DEBUG
- TITLE

- BLOCKSET
- MAP
- SKY
- GROUND
- AMBIENT_LIGHT
- AMBIENT_SOUND

下面几节将对这些元素逐个进行探讨。

DEBUG 元素

DEBUG 元素是一个空元素,它用来告知 3D 浏览器 Rover 提供调试信息。如果你在 3DML 文件中加入了 DEBUG 元素,Rover 将在处理它们的时候,在另一个浏览器窗口中报告错误。然而,在将它们放到 Web 上之前,你会希望将 DEBUG 元素从 3DML 文件中移除。下面是 DEBUG 元素的语法:

```
<DEBUG/>
```

TITLE 元素

TITLE 元素是一个空元素,它用来指定场景的题目。这个元素包含一个单个的属性 NAME,在那里你可以设置场景的名称。下面是 TITLE 元素的语法:

```
<TITLE NAME="spot Title">
```

下面是一个使用 TITLE 元素的例子:

```
<TITLE NAME="Maze"/>
```

BLOCKSET 元素

BLOCKSET 元素是一个空元素,它用来指定场景的块集。这个元素包含一个单个属性 HREF,在那里你可以设置场景块集的 URL 地址。下面是 BLOCKSET 元素的语法:

```
<BLOCKSET HREF="url"/>
```

场景的块集由块和在场景中可用的内建纹理组成。Flatland Online 提供了多个块集,这样你可以使用它们构造场景,开放的块集格式的方法还使你可以创建你自己的块集。下面是 Flatland Online 提供的块集:

- Basic
- Basic2
- Village
- Island

当 Rover 在场景中遇到了一个新的块集的时候,它在下载这个块集前提示用户批准。如果用户同意,块集就被下载并存储在你的硬盘上(缓存)以备将来使用。Rover 在尝试下载之前总是检查已储存的块集,这样在块集被多个场景共享的时候节省了下载时间。贮存的块集以二进制文件格式用扩展名.bset 存储在 Rover 的高速缓存中。要获得有关块集的更多信息,请访问 Flatland Online 的 Block Set 页面 <http://www.flatland.com/build/blockset..>

body.html。

注意:在本书编写期间,Flatland Online 已经发布了“古墓丽影 II”块集,它包含的块具有着流行的“古墓丽影 II”游戏中的感觉和视觉效果的块。要获得有关“古墓丽影 II”块集的更多信息,请访问 Flatland Online 的站点上的 3Directory 页面 <http://www.flatland.com/enter.html>。

这里有一个使用 BLOCKSET 元素的例子,它指定 Basic 块集来构成场景:

```
<BLOCKSET HREF=http://blocksets.flatland.com/flatsets/basic.bset/>
```

MAP 元素

MAP 元素是一个空元素,它指定场景的样式和尺寸。这个元素包含两个属性:STYLE 和 DIMENSIONS,它们用来设定场景的样式和尺寸。下面是 MAP 元素的语法:

```
<MAP STYLE="single|double" DIMENSIONS="(columns,rows,levels)">
```

STYLE 属性可以设置成两个值之一:single 和 double。一个 single 场景指的是地图通过 3DML 文件的 body 部分中的独立字符指定。另一方面,double 场景就是地图通过双字符(字符对)指定。使用独立字符就已经可以满足对大多数场景的指定。有关这个属性,在本章后面的如何创建场景地图的学习中,你还将获得更多的认识。

DIMENSIONS 属性确定场景的行、列和层的数量,单位是块。这个属性非常的重要,因为它确定了场景的整体尺寸。下面是一个使用 MAP 元素创建场景的例子,这个场景有 12 行 12 列 3 层:

```
<MAP STYLE="single" DIMENSIONS="(12,12,3)" />
```

SKY 元素

SKY 元素是一个空元素,它用来指定场景中天空的纹理。这个元素包含三个属性:TEXTURE、COLOR 和 BRIGHTNESS。下面是 SKY 元素的语法:

```
<SKY TEXTURE="image-file-path-or-URL" COLOR="(red,green,blue)"  
BRIGHTNESS="brightness%" />
```

TEXTURE 属性用来设定场景的天空的纹理,这是一个 GIF 图片的 URL 地址,它在任何没有块的位置上显示。你可以使用 COLOR 属性在纹理图片之外指定天空的颜色。颜色通过三个十进制数表示,这三个十进制数各表示一个基本色素(红、绿和蓝)。如果你不指定纹理或是颜色,浏览器就会使用块集中定义的默认纹理。

除了设置天空的纹理和颜色,你还可以设置它的亮度。天空的亮度通过完整的百分数来表示,在你不使用 BRIGHTNESS 属性指定的情况下,默认是 100%。下面是一个使用 SKY 元素设置一个亮度为 80%的默认天空纹理的例子。

```
<SKY BRIGHTNESS="80%">
```

GROUND 元素

GROUND 元素于 SKY 元素类似,只不过它确定地面的外观而不是确定天空的。这个元素包括两个属性 TEXTURE 和 COLOR,它们确定场景的地面层的纹理和颜色。下面是

GROUND 元素的语法:

```
<GROUND TEXTURE="image-file-path-or-URL" COLOR="(red,green,blue)"/>
```

与 SKY 元素中的 TEXTURE 属性类似,这里的 TEXTURE 属性用来设置场景的地面纹理,这是一个 GIF 图片的 URL 地址,它在地面上显示。同时,你也可以使用 COLOR 属性在纹理图片外指定地面的颜色。如果你不指定纹理和颜色,浏览器就使用块集默认的纹理。

如果你在 3DML 文件中使用 GROUND 元素,你并不需要担心在你的场景的第一层包含纯色的地板块。换句话说,如果你希望设计一个没有地面的场景,你可以完全的省去 GROUND 元素。下面是一个使用 GROUND 元素设置默认地面纹理的例子:

```
<GROUND/>
```

AMBIENT_LIGHT 元素

AMBIENT_LIGHT 元素是一个空元素,它用来确定场景的环境光水平。这个元素包含两个属性 BRIGHTNESS 和 COLOR,它们确定了环境光的水平以及它的颜色。下面是 AMBIENT_LIGHT 元素的语法:

```
<AMBIENT_LIGHT BRIGHTNESS="brightness%" COLOR="(red,green,blue)"/>
```

环境光水平本质上是场景可用的光的数量,它通过一个完整的百分数表示。如果你并不准备在场景中使用任何附加的光源,你就可能希望用 100% 来最大化环境光。换句话说,3DML 支持其他类型的光源,比如场景光和点光,它们可以被高等级的环境光淹没。如果你一定要使用附加光,你可能希望降低环境光等级。

注意:除了环境光,3DML 支持 3 种其他类型的光:球形光、点状光和场景光。球形光从天空的一个指定的方向照射,照亮整个虚拟世界。点状光从指定的位置向所有方向照射。场景光从指定的位置向指定的方向照射。为了让你能够获得 3DML 更多的舒适之处,我鼓励你作各种光效果的试验。

COLOR 属性设置场景中环境光的颜色。这里值得指出的是有色光只能被有 3D 加速硬件的系统支持。Rover 在使用有色光之前会检查是否有这样的硬件存在。如果 3D 硬件不可用,场景中的所有光都将忽略 COLOR 属性,成为白光。如果不使用 COLOR 属性,白光是默认的光。

下面是一个使用 AMBIENT_LIGHT 设置场景的环境光水平的例子,这个环境光是 60% 的白光:

```
<AMBIENT_LIGHT BRIGHTNESS="60%" />
```

AMBIENT_SOUND 元素

你要学到的最后一个头元素就是 AMBIENT_SOUND,它是一个空元素,允许你指定在遍历场景的时候播放的声音。3DML 支持定点 3D 声音,它可以在场景的指定位置中播放。环境音是不同的,因为它要在场景中的任何地方听。背景音乐是环境音的一个好例子。

AMBIENT_SOUND 元素包括四个属性: SOUND_FILE、VOLUME、PLAYBACK 和 DELAY。下面是 AMBIENT_SOUND 元素的语法:

```
<AMBIENT_SOUND FILE="wave-file-path-or-URL" VOLUME="volume%"
```



```
PLAYBACK="looped|random" DELAY="minimum..maximum"/>
```

SOUND_FILE 属性允许你指定以 WAV 文件格式存储的声音文件的路径和 URL 地址。声音的音量通过使用 VOLUME 属性以完整的百分数表示。声音可以循环或是随机的播放,这由 PLAYBACK 属性的值确定。循环播放就是重复的播放,在每次重复之间没有停顿。随机播放就是以随机的间隔播放,间隔时间由 DELAY 属性的值确定。这个值确定在下一次声音播放的随机间隔的最大最小范围,而不是确定它的播放完成时间。

注意:WAV 格式是 Window 平台上的流行声音文件格式。附带在 Windows 中的录音机应用程序是一个允许你录制和播放 WAV 格式声音的简单应用程序。同时,大多数的商用声音编辑器允许你将声音转化为 WAV 格式或读取 WAV 格式文件。我希望 3DML 的未来版本将支持 MIDI 声音文件,它提供了一种存储音乐的非常有效的方法。

下面是一个使用 AMBIENT_SOUND 元素的例子,它设置音量为 65% 的循环播放背景音。

```
<AMBIENT_SOUND FILE="Music.wav" VOLUME="65%" PLAYBACK="looped"/>
```

35.2.3 BODY 元素

BODY 元素是用来定义场景结构的主要元素。虽然 3DML 文件的头部分当然包含了有关场景的有用信息,但 body 部分还是包含了场景的内容。BODY 元素支持多个不同的子元素,你可以使用它们确定场景的结构,以及确定使用者如何于场景交互。下面是出现在 3DML 文件中的 body 部分的最常用子元素:

- LEVEL
- ENTRANCE
- EXIT
- CREATE

下面几节将分别探讨这些元素。

LEVEL 元素

LEVEL 元素用来定义场景层的指定的地图结构。场景的每一层包含一组块,它的尺寸通过你在头部分使用 MAP 元素所确定的场景尺寸指定。场景中的每一层有同样数目的行和列,它们有着非常类似的结构。LEVEL 元素包含一个属性 NUMBER,它用来指定层数(层数作为 MAP 元素中场景尺寸的第三维确定)。场景第一层 NUMBER 属性值应该设为 1。

LEVEL 元素的内容就是层地图本身,它通过 ASCII 图表示。不同的 ASCII 字符表示不同的现实对象,例如立方块、天花板、坡道、拐角、空地等等。下面是一些在 BASIC 块集中定义的块对象以及在地图上表达它们的字符:

- # —— 实体块(立方体)
- . —— 空块(空气)

- - ——天花板
- I ——圆柱
- P ——金字塔
- N ——北向斜坡
- S ——南向斜坡
- E ——东向斜坡
- W ——西向斜坡

注意: basic 块集包括的块对象比这里列出的要多很多。要获得有关这些块的更多信息,请访问 Basic Block Set Reference, 位于 <http://www.flatland.com/build/reference/basicset.html>

要使用这些对象来创建一层的地图,你仅仅要将它们放进数组的相应位置中,这个数组被 LEVEL 元素围绕。下面是一个简单的 5×5 层的例子:

```
<LEVEL NUMBER="1">  
  ##.##  
  #---#  
  #---#  
  #---#  
  #####  
</LEVEL>
```

这个层代码描述了一个四面围墙的房间,它在北面墙的中央有一个开口。图 35.2 显示了在用户在房间内并面向北墙时 Rover 所显示的样子。

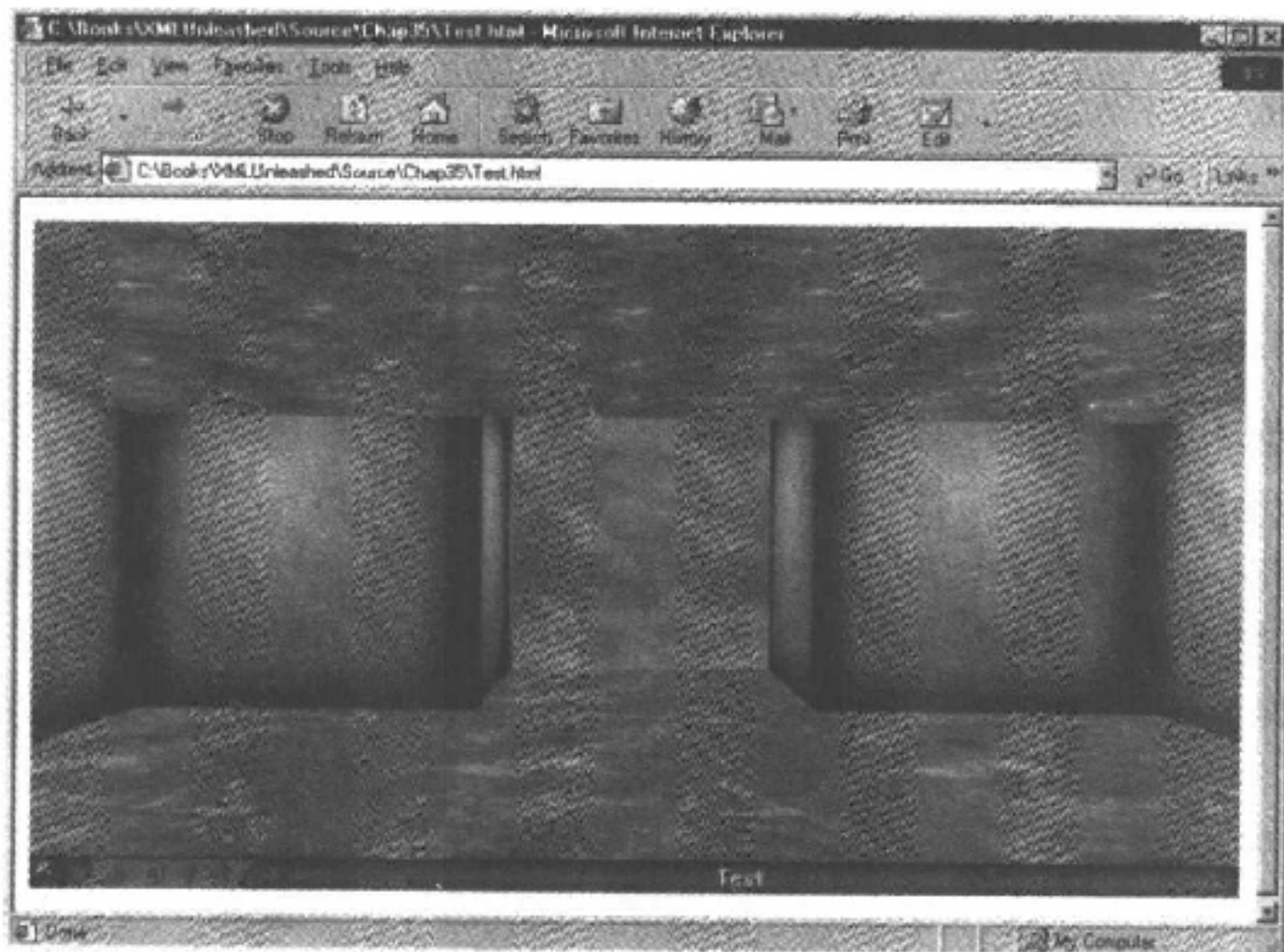


图 35.2 简单的 5×5 层

如你所见, Rover 解释在层地图中描述的对象并产生了一个 3D 视图。你可以自由的在场景中移动, 并与之发生互动。当我说“互动”的时候, 我的意思是墙将限制你的行进, 开口将允许你穿过去, 等等。

你可以通过包含适当编码的多个层元素创建多层场景。例如, 清单 35.1 包含了一个 12×12 的三层场景, 层之间由斜坡相连。

清单 35.1 有连接层的斜坡的三层场景

```
<LEVEL NUMBER="1">
#####
#---^###--.#
#-##-----#S#
#--#-#####
#---#-----#
#-####^####-#
#-----#
#-##-##-##^##
#--^#-#----#
#####-#####-#
#-----#
#####
</LEVEL>

<LEVEL NUMBER="2">
#---#####
#-#----#-#-#
#-#####-#-#-#
#----#---#-#
#-#####-#####
#---^-----#
#-#####-#-#-#
#-#W.#---#-#
#--##-#-#####
#----#-----#
#####
</LEVEL>

<LEVEL NUMBER="3">
#####
#@#.....#..#
#.#.#.#.#.#.#
#.#.#.#.#.#..#
#.....#....#
#.#.#.#.#.#.#.#
#..#.....#.#
#####.#.#.#
#.....#
#.P.P..P.P.#
#.....#
#####
</LEVEL>
```

你可能已经从这些层地图中猜到了,这个场景实际是一个三层的迷宫。请花些时间来学习这些地图并理解场景的整体结构。S 块用来提供一个从第一层到第二层的南向斜坡。类似的,W 块用来提供一个从第二层到第三层的西向斜坡。四个 P 块用来在第三层放置金字塔。

或许你想知道有关 ^ 和 @ 块的知识。^ 块是脉动光,也就是基于强度变化的点状光。^ 块的默认设置产生一个缓慢变暗和变亮的脉动光。@ 块是入口块,它用来作为链接场景的一种方式,这种情况下,入口块用来返回到迷宫的开始处。链接本身使用 EXIT 元素确定,这个你马上就会学习到。

ENTRANCE 元素

ENTRANCE 元素是一个空元素,它指定场景的入口。每个场景起码需要一个入口,用户在第一次访问场景的时候,从这里开始。ENTRANCE 元素包括三个属性:LOCATION, NAME 和 ANGLE。下面是 ENTRANCE 元素的语法:

```
<ENTRANCE LOCATION="(column,row,level)" NAME="name" ANGLE="degrees"/>
```

LOCATION 属性指定入口的位置,它通过块位置的列、行、层表示。场景中的 LOCATION 点的块不会有什么特别,它与其他块看起来没有什么不同。块位置经常考虑指定在场景第一层的左上角。换句话说,场景中第一层的左上角(西北)块的位置为 1×1×1。如果你移动位置,向下一块,向下三块,向上两层,位置就是 2×4×3。

NAME 属性赋给入口一个名称,这很重要,因为稍后你就需要使用这个名称链接回来。每个场景需要有一个名为 default 的入口,这是场景的默认入口。ANGLE 属性确定用户在入口时面对的方向。角度的起点是正北方(0 度),值以顺时针方向增长——90 度是东方,180 度是南方,270 度是西方。ANGLE 属性允许你指定从 0 度到 359 度的任何度数的角度。

下面是一个使用 ENTRANCE 元素指定默认入口的例子,这个入口位于 2×11×1,角度 90 度:

```
<ENTRANCE LOCATION="(2,11,1)" NAME="default" ANGLE="90"/>
```

EXIT 元素

EXIT 元素是一个空元素,它用来确定到另一个 3DML 文件和同一文件的另一部分的链接。这个元素包括五个属性:LOCATION、HREF、TRIGGER、TEXT 和 TARGET。下面是 EXIT 元素的语法:

```
<EXIT LOCATION="(column,row,level)" HREF="file_name#entrance_name"
  TRIGGER="click on, step on" TEXT="text" TARGET="destination name"/>
```

LOCATION 属性确定出口的位置,它通过块位置的列、行、层表示。你可以将场景中的出口放置到任何的一个块上,而通常将出口与入口符号(@)相关联。这告诉用户这个块将会把它们传送到其他的地方。这个地方使用 HREF 属性指定,这与在 HTML 锚标记(<a>)中的 HREF 属性非常类似。在这里,HREF 属性指的是一个 3DML 文件的文件名和文件中的一个入口。你也可以忽略文件名而只是指定同一文件中的入口。

TRIGGER 属性确定出口如何被激活,它有两个值:click on 和 step on。你可以指定任何一个或两个触发值。Click on 值意味着你要在出口处点击鼠标以激活出口。Step on 值意味着你要移动到出口上方以激活出口。

TEXT 属性确定当用户接近出口时在指针后显示的文本消息。TARGET 属性允许你指定出口链接打开的框架和窗口。如果你希望链接在同一浏览器窗口中打开,那么就可以忽略 TARGET 属性。

下面是一个使用 EXIT 元素创建出口的例子,出口位于 $2 \times 2 \times 3$ 并链接到默认入口:

```
<EXIT LOCATION="(2,2,3)" HREF="#default" TRIGGER="click on,step on"
  TEXT="Back to start"/>
```

CREATE 元素

最后一个你要学习的 body 元素就是 CREATE 元素,它用来创建和修改块。CREATE 元素包含两个属性,SYMBOL 和 BLOCK。它们用来指定字符记号和你希望创建或修改的块类型。下面是 CREATE 元素的语法:

```
<CREATE SYMBOL="symbol-or-block-name" >
</CREATE>
```

SYMBOL 属性确定用来表示场景地图中块的字符记号。BLOCK 属性指定块的类型,这可以是一种你使用的块集中定义的标准块类型。下面是定义在 Basic 块集中的部分块类型:

- full——实体块(立方体)
- ceiling——天花板
- column——圆柱
- pyramid——金字塔
- nramp——北向斜坡
- sramp——南向斜坡
- eramp——东向斜坡
- wramp——西向斜坡

CREATE 元素主要用来修改块的方位和纹理。

PARAM 元素用来修改块的方位:

```
<PARAM ORIENT="y,x,z"/>
```

PARAM 的 ORIENT 属性用来指定块的角度方位。ORIENT 属性中指定的每个角度根据所给的轴旋转块。因此如果你只提供了一个 90 度的角,块将围绕 Y 轴沿顺时针方向旋转 90 度。如果你指定两个值,第一个值将围绕 Y 轴旋转块,第二个值将围绕 X 轴旋转块。使用所有的三个角度不是必要的,因为你用两个就可以得到渴望的效果。要注意块的旋转是以 90 度旋转增加的,这很重要。它意味着 ORIENT 中可接受的角度值是 0,90,180 和 270。

要修改块的纹理,你必须使用 CREATE 元素中的 PART 元素。PART 元素包括一些属性以修改块的纹理:NAME、TEXTURE、ANGLE、COLOR、TRANSLUCENCY 和 STYLE。下面是 PART 元素的语法:

```
<PART NAME="name" TEXTURE="image.gif-or-URL" ANGLE="0-359"
  COLOR="(red,green,blue)" TRANSLUCENCY="number%"
  STYLE="tiled|scaled|stretched"/>
```

NAME 属性指定你要设置纹理的块的部分。NAME 属性的值可以有如下值: n、s、e、w、top 和 bottom。你也可以使用 NAME 值 * 指定块的所有面。TEXTURE 属性指定用于块的纹理图片的文件名和 URL 地址。你可以在纹理以外使用 COLOR 属性确定颜色。

ANGLE 属性允许你在应用到块之前旋转纹理。ANGLE 的正确角度值可以是 0 到 359。TRANSLUCENCY 属性允许你修改纹理的透明度。透明度只有在有 3D 加速硬件的系统中才能被支持;其他的系统将忽略透明度的值,将所有块以不透明方式显示。透明度通过完整的百分数表示。

PART 元素的 STYLE 属性用来决定纹理如何映射到块的相应面上。要谨记块的每一面是 256×256 像素。STYLE 属性是很重要的,因为它确定了那些比 256×256 大或小的纹理图片是如何映射到块上的。默认值是 stretched,它意味着图片将四面伸展以准确的与块相符。Scaled 将缩放纹理以适合块。值 tiled 将拼接纹理以适合块。

CREATE 元素将放置在文件的 body 部分中第一个层之前的位置。下面是一个使用 CREATE 元素改变 full 块的东、南、西、北四个面的纹理的例子:

```
<CREATE SYMBOL="#" BLOCK="full">
  <PART NAME="n,s,e,w" TEXTURE="pinkmarble.gif"/>
</CREATE>
```

35.3 创建 3DML 世界

现在,你已经有足够的的能力装配一个完整的 3DML 世界了。要承认,我只是讲述了 3DML 的基础,但你会发现对于开始创建 3DML 场景来说,这些已经绰绰有余了。3DML 包含所多其他有趣的特性,比如 3D 声音,光效,弹出,精灵和动作。我鼓励你自己一旦适应了 3DML 的基础就补充一下这些特性的知识。现在,让我们开始你的第一个 3DML 世界。

可能你没有注意到,通过本章的介绍,你实际上已经学到了基本 3DML 场景的每个方面。清单 35.2 包含了迷宫场景的完整代码,这是一个 12×12 的三层迷宫。

清单 35.2 3DML 场景 Maze.3dml 的 3DML 代码

```
<SPOT>
  <HEAD>
    <DEBUG/>
    <TITLE NAME="Maze"/>
    <BLOCKSET HREF=http://blocksets.flatland.com/flatsets/basic.bset/>
    <MAP STYLE="single" DIMENSIONS="(12,12,3)"/>
    <SKY BRIGHTNESS="80%"/>
    <GROUND/>
    <AMBIENT_LIGHT BRIGHTNESS="60%"/>
    <AMBIENT_SOUND FILE="Music.wav" VOLUME="80%" PLAYBACK="looped"/>
  </HEAD>
  <BODY>
    <CREATE SYMBOL="#" BLOCK="full">
      <PART NAME="n,s,e,w" TEXTURE="pinkmarble.gif"/>
```

```

</CREATE>
<LEVEL NUMBER="1">
  #####
  #----^###, #
  #-##-----#S#
  #--#-#####
  #----#-----#
  #-###^###-#
  #-----#
  #-##-#-#^##
  #--^#-#----#
  #####-#####-#
  #-----#
  #####
</LEVEL>
<LEVEL NUMBER="2">
  #---#####
  #-#----#-#-#
  #-#####-#-#-#
  #----#---#-#
  #-#####-###-#
  #---^-----#
  #-#####-#-#-#
  #-#W.#---#-#
  #--#-#-#-#-#-#
  #----#-----#
  #####
</LEVEL>
<LEVEL NUMBER="3">
  #####
  #@#.....#..#
  #.#.#.#.#.#.#
  #.###.#.#.#.#
  #.....#....#
  #.###.#.###.#
  #..#.....#.#
  #####.#.#.#
  #.....#
  #.P.P..P.P.#
  #.....#
  #####
</LEVEL>
<ENTRANCE LOCATION="(2,11,1)" NAME="default" ANGLE="90"/>
<EXIT LOCATION="(2,2,3)" HREF="#default" TRIGGER="click on,step on"
  TEXT="Back to start!"/>
</BODY>
</SPOT>

```

在本章前面你已经看到了这个代码的各个部分,但清单 35.2 显示了它们如何结合在一起构造一个完整的 3DML 文件。顺便说一下,所有的 3DML 文件都有 .3dml 扩展名。这个文

件名为 Maze. 3dml。

现在你已经有了一个完整的 3DML 文件,但你依然需要在网页中引用它以将它发布。下面一节将告诉你如何将 3DML 文件嵌入到网页之中。

35.4 在网页中嵌入 3DML 世界

要将 3DML 文件嵌入到网页中,就要使用 HTML 标记<embed>。你必须在<embed>中提供 3DML 的 MIME 类型以及 3DML 文件的文件名和 Rover 插件页的位置。下面是一个使用<embed>标记引用名为 MySpot. 3dml 的 3DML 文件的例子:

```
<embed type="model/vnd.flatland.3dml" src="MySpot.3dml"
      pluginspage=http://www.flatland.com/download-direct.html
      width="100%" height="100%">
```

pluginspage 属性在这个例子中很有用,因为它在探测到没有安装插件的情况下自动的将用户带到 Rover 的下载页。

清单 35.3 包含完整的 Maze.html 网页的代码,它使用<embed>标记引用 Maze. 3dml 文件。

清单 35.3 网页 Maze.html 的 HTML 代码

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
  </head>
  <body>
    <embed type="model/vnd.flatland.3dml" src="Maze.3dml"
          pluginspage=http://www.flatland.com/download-direct.html
          width="100%" height="100%">
  </body>
</html>
```

注意<embed>标记用在网页的 body 部分。这个代码显示了将 3DML 嵌入到网页中是多么的容易。图 35.3 显示了迷宫场景的入口。

如果你可以找到到达迷宫顶层的路径,你就会找到在场景地图中使用 P 块字符定义的四个金字塔。我们可以使用 3DML 文件作为找出迷宫的路径的地图。图 35.4 显示了当你进入到迷宫顶层的时候所见到的金字塔。

如果你继续深入迷宫的顶层,最终你将找到将你带回到开始处的入口。图 35.5 显示了迷宫终点的入口。

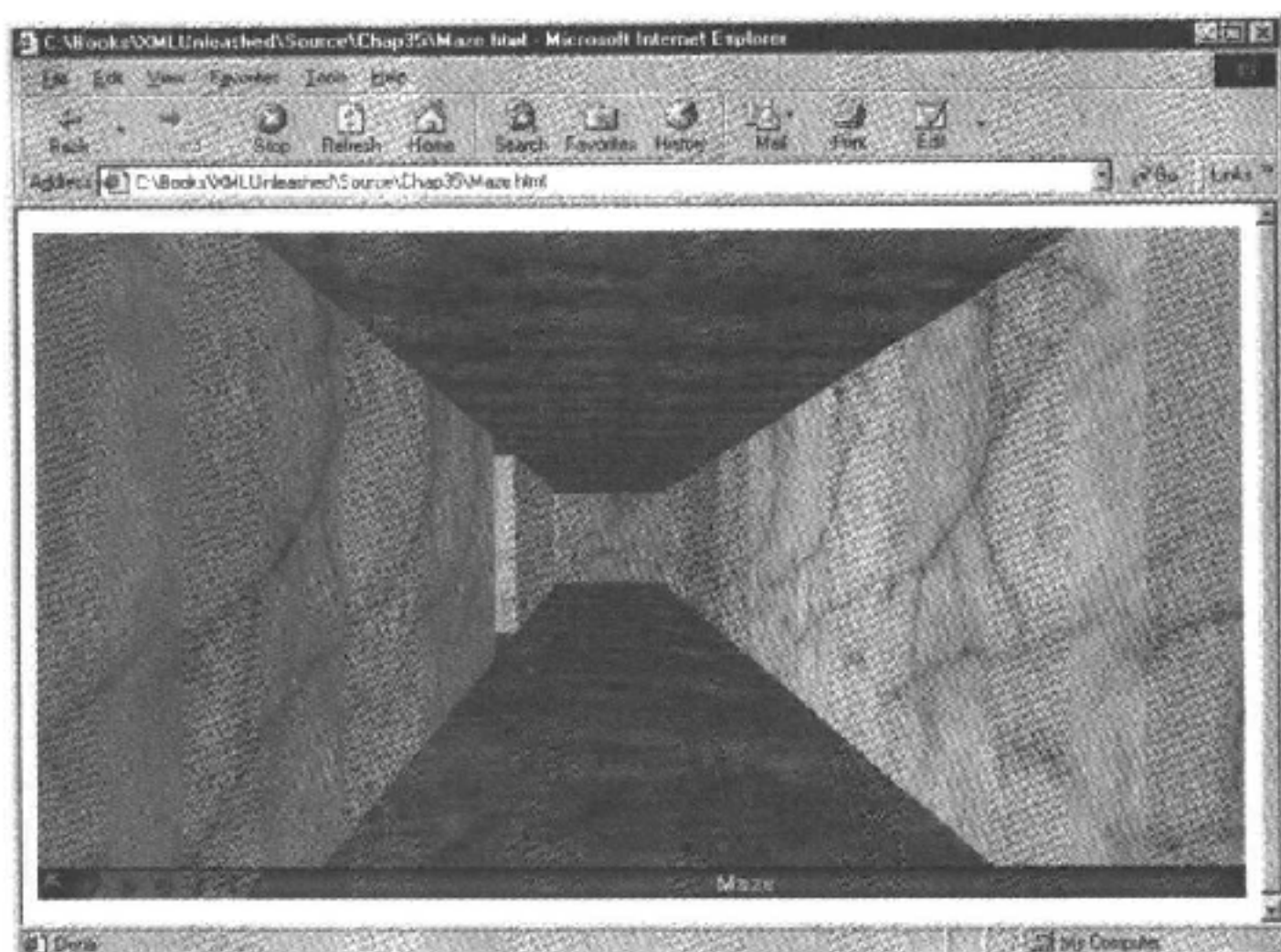


图 35.3 迷宫场景的入口

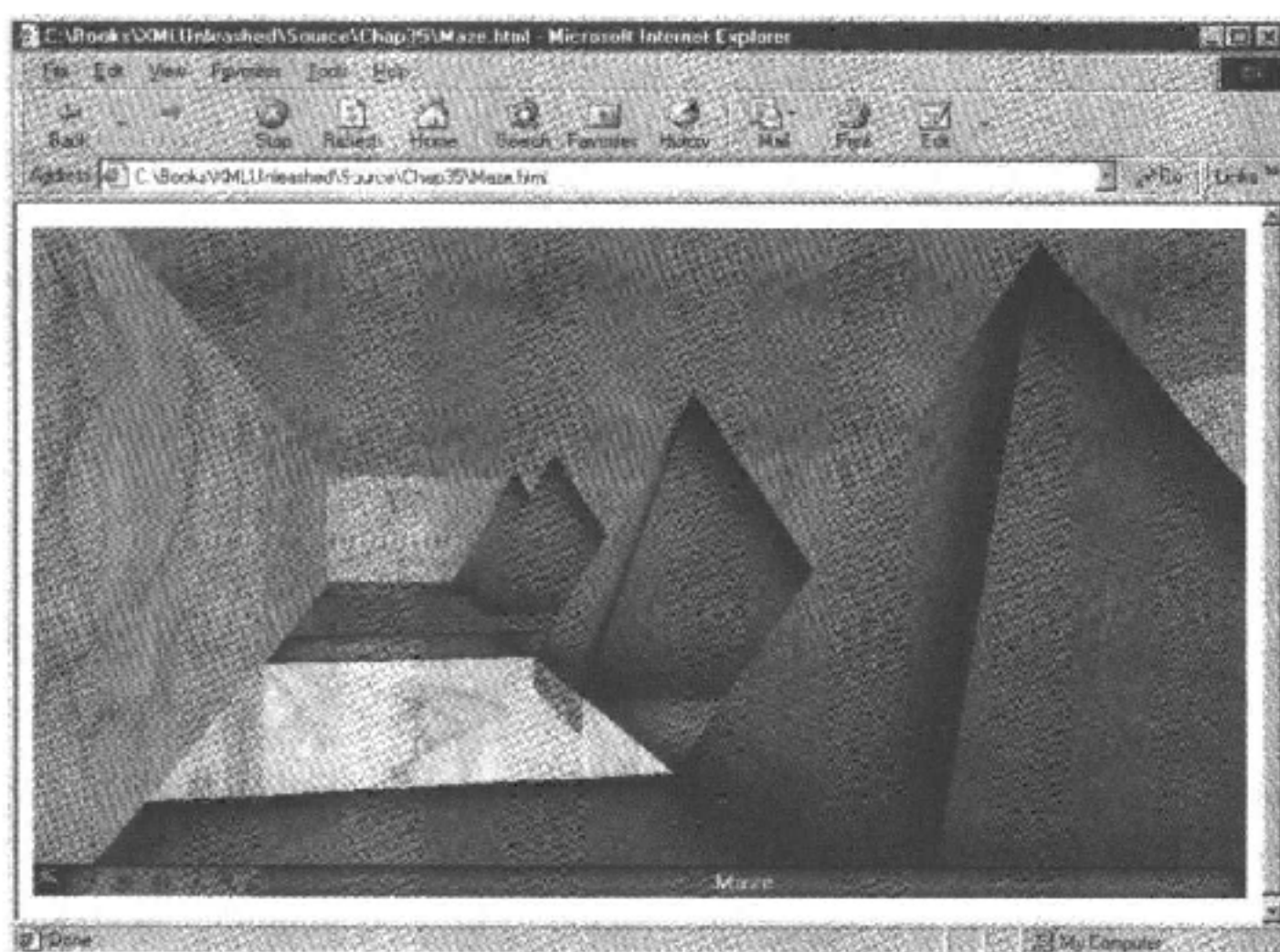


图 35.4 位于迷宫场景顶层的金字塔

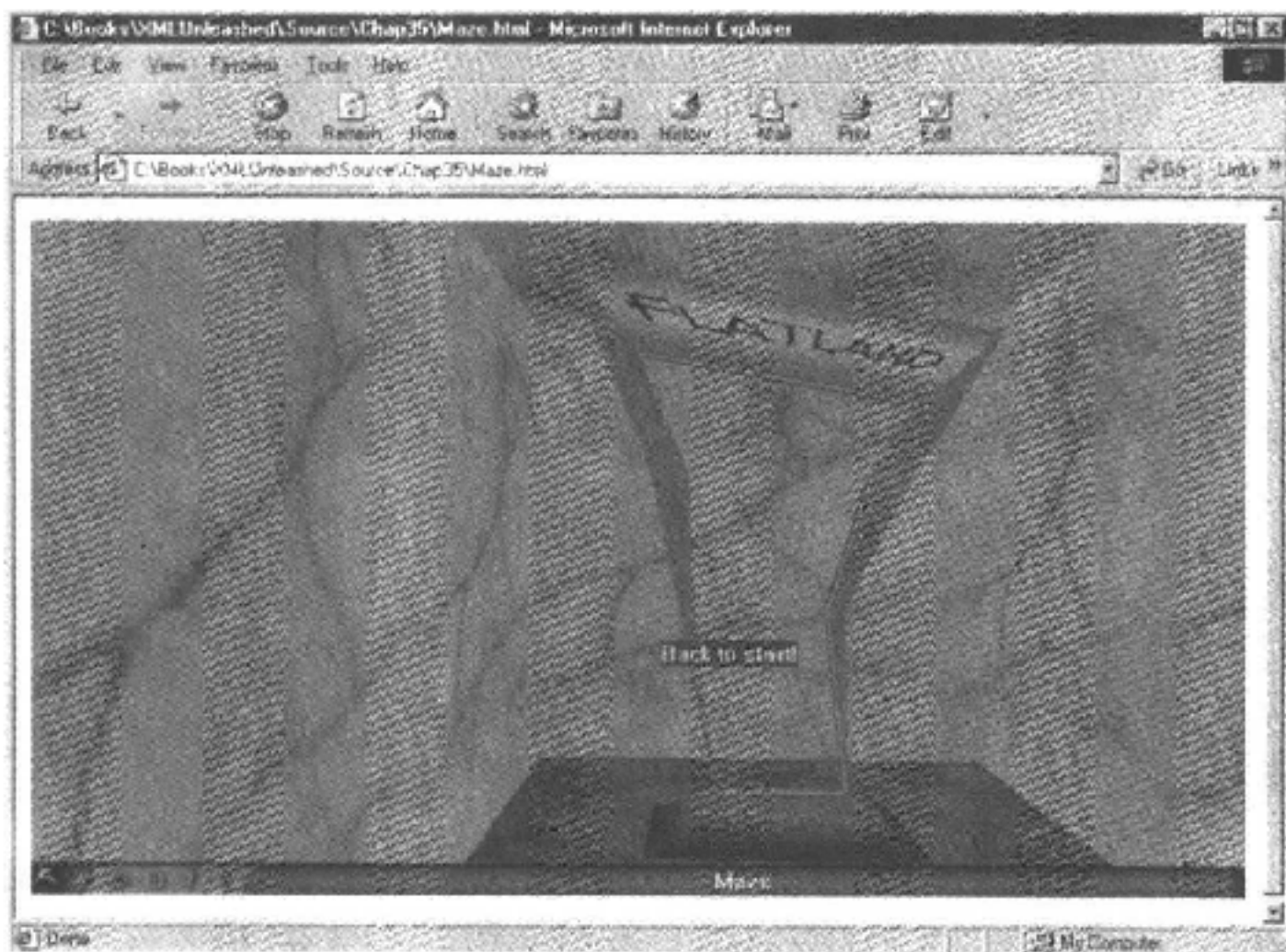


图 35.5 迷宫终点的入口

35.5 Spotnik 3DML 场景制作工具

现在,你已经对 3DML 词表以及它们是如何用来创建场景的有了一定感受。我可以向你提供一个小秘密:Flatland Online 有一个名为 Spotnik 的在线工具,它允许你可视化的创建 3DML 场景。有了 Spotnik,你可以点击鼠标来创建场景,这是使用 3DML 的一种令人难以置信的有趣方式。Spotnik 可以加载已有的 3DML 场景,它还允许你在制作场景的过程中随时浏览 3DML 代码。也许 Spotnik 最有趣的地方是它是完全基于 Web 的,这就是说你要在网络浏览器中使用它。Spotnik 主页位于 <http://spotnik.flatland.com>。

Spotnik 是一个创建场景的优秀工具,而使用它来打开一个已经存在的场景也同样的有趣。图 35.6 显示了在 Spotnik 中显示的迷宫场景;在图中,第一层的大部分都是可见的。

在 Spotnik 工具中显示的第一层地图使用图标来表现各种不同类型的块。图 35.7 显示了第三层的地图,由于缺少天花板块,这个地图更容易看懂。

Spotnik 真正的强大之处在浏览器窗口的左边被揭示出来,这里你将找到一个工具条组。这些工具条有所有不同块类型的按钮,它们可以用来构造场景。要添加或修改块,只需要在工具栏上选择块并在地图上点击块就可以。你甚至可以通过在工具条上点击块并点击位于浏览器窗口左上方的 Customize 按钮,就可以自定义一个块。

在 Spotnik 设计过程中的任何时候,你都可以点击位于浏览器窗口左下方的 Show Source 按钮来浏览场景的 3DML 代码。也许更有用的是 Show 3D Spot 按钮,它打开使用 Rover 浏览的场景的一个新的浏览器窗口。通过这个特性,你可以真实而方便的测试场景。此外,Save 3DML 按钮允许你将你的工作存成本地 3DML 文件。

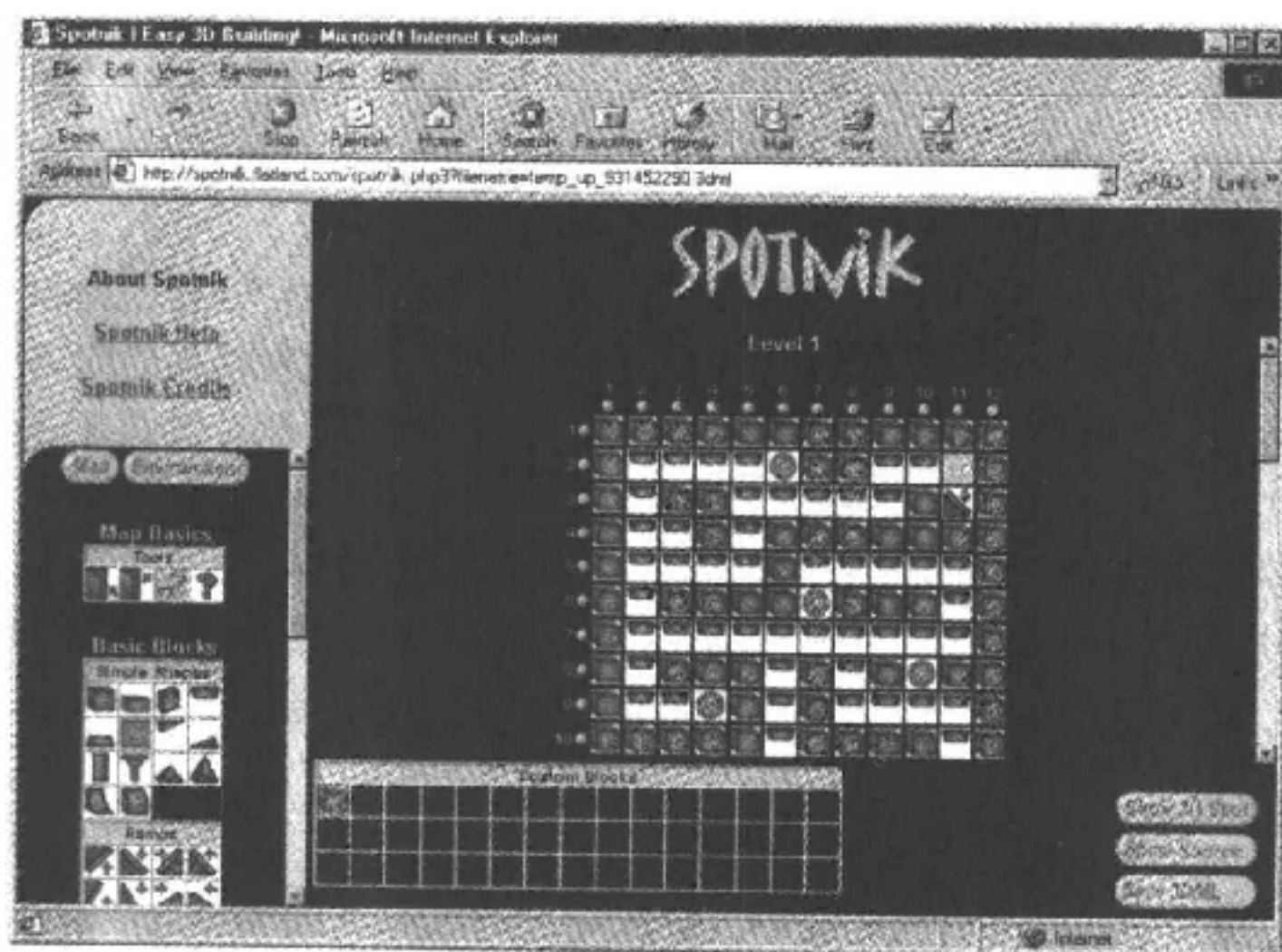


图 35.6 在 Spotnik 工具中看到的迷宫场景的第一层

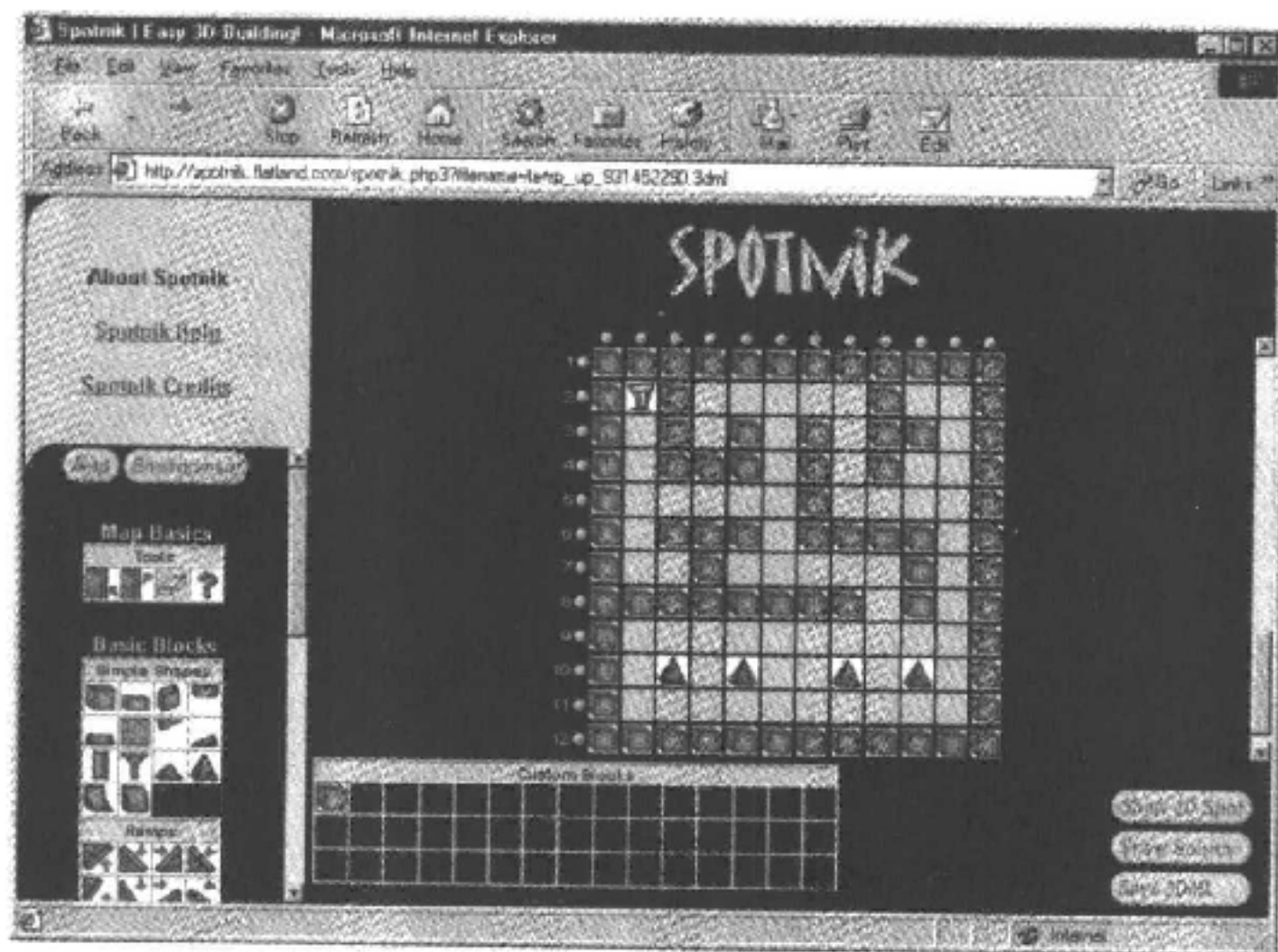


图 35.7 在 Spotnik 工具中浏览的迷宫场景的第三层

Spotnik 是一个如此直观的工具,我将让你自己去体验学习它的优秀之处。我想你也会同意,比起手工编写 3DML 代码来说,Spotnik 在构建场景方面提供了更多的激动人心的方法。

35.6 总 结

本章向你介绍了 3DML,这是一个基于 XML 的有趣且激动人心的技术。诚然,3DML 从技术上讲不是 XML 应用,但它是如此的接近使我不能不给它一些关注。3DML 允许你使用简单的标记创建称为场景的 3D 世界。3DML 比起诸如 VRML 这样的虚拟现实语言更加的简单,同时它也更容易学习和使用。实际上,我认为 3DML 词表结构的简单正是它的强大之处,因为这样它更容易学习。有了使用标记语言的一定经验,你可以在几分钟之内创建一个 3DML 世界。

本章通过介绍最常用的 3DML 元素,引导你创建了一个简单的迷宫场景。在创建了场景的 3DML 代码后,Rover 3DML 插件可以用来在浏览器上测试场景。本章还介绍了 Spotnik 3DML 场景构建工具,它提供了创建 3DML 场景的一个可视化方法。

第 36 章 用 MathML 表达数学

如果你曾经尝试制作一个包含数学表达式的文件,你一定已经对包括图像表达这样的复杂概念内在的难题有所了解。作为一个工科学生,我经常要在文件中空出空白的区域以便在文件打印出来后手工的写入表达式。当然,诸如 Microsoft Word 这样的现代文字处理函数已经包含了改进的表达式编辑器来简化在打印文件中描述数学表达式。然而,现在还没有一个得到广泛认可的用于描述数学信息的真正标准。起码是到目前为止。

本章将介绍 MathML,这是一个为构造数学内容而设计的 XML 词表。MathML 的最初目标是方便数学内容在 Web 上的表达。然而,现在人们对一种可以用来在不同应用程序间共享数学信息以及提供处理数学数据的应用的可存储全方位语言有着更大的需求。MathML 被设计用来方便数学符号的表达以及对数学内容的智能处理。

36.1 MathML 基础

传统上,数学信息在打印中的表达非常的困难,而且在 Web 上的表达更为的困难。大多数的 Web 站点依靠 GIF 图片来显示数学符号,这其中没有包含任何有关数学内容自身的结构和语义方面的信息。加之,使用静态 GIF 图片就无法考虑不同的字体以及其他的主流浏览器可能希望应用在数学内容上的效果。

MathML 是一个在 Web 页上表达数学符号的方法。它在编写数学符号的高质量可见表述以及与实际数学内容相关的结构信息方面具有足够的灵活性。因此,你可以使用 MathML 作为一个显示数学内容的描述语言以及指定数学内容的结构和语义的内容语言。一些应用程序只是简单的浏览 MathML 内容,而其他的则处理 MathML 内容,在这种情况下,MathML 作为一个存储和传输介质使用。

注意:记录表明,现在还有一些其他的用来编写数学内容的解决方案。TeX 就是一个用来编写用于打印的数学表达式的流行排版语言。LaTeX 现在的应用更为广泛,它是一个对最初的 TeX 语言改进的版本。TeX 和 LaTeX 都是有局限的,因为它们只是编写数学表达式的表述而不编写语义和结构信息。MathML 通过支持可包含数学内容中的语义和结构信息,比这些语言要前进了一大步。

要了解 MathML 内容并不是可以用手工编写的,这一点非常的重要。在本章稍后你就会学到,MathML 内容很容易变得很混乱,这就使它非常难于手工编码。此外,估计绝大多数的用户都会依赖专门的工具来编辑、浏览和处理 MathML 内容。由于以上原因,我将不在 MathML 的本质细节上讲的太深。而且还有一个原因是,随便一个对 MathML 词表的深入探讨就可以轻易的写满整整一部书。

本章的目标就是给你一个对 MathML 词表的总体看法,以及说明它是如何构造数学内容的。在本章的最后,我将向你介绍一些处理 MathML 内容的可用 MathML 工具。你将发

现这些工具是不可缺少的。

36.2 深入 MathML 词表

MathML 词表在 1998 年 4 月作为规范提交给 W3C。从此以后,它已经经历了多次修改,但是核心规范依然是原封未动。与大多数的其他 XML 词表不同,这些词表主要提供一个构造信息的单独解决方案,而 MathML 提供了两个构造数学内容的解决方案:表达标记和内容标记。在 MathML 中定义了不同的元素以用于这两种方法中的任一个。

表达标记用来编写用于表达的数学内容,比如用在网页上。表达标记注重于一个数学表达式是如何显示的而不过多的考虑表达式的语义。如果你的目的只是要准确的显示数学内容的话,它可以工作的非常好,例如表达标记可以很好的作为 Microsoft Word 的 Equation Editor 的存储方法。表达标记的主要问题是如果你需要处理和计算表达式的话,它有非常严格的限制。

内容标记用来编写用于处理和计算信息的数学内容。它集中考虑数学表达式的结构化语义,而极少考虑表达式是如何显示给用户的。内容标记对于那些依赖于语音合成的应用程序来说非常的重要,这样的程序有科学工具和辅助学习。一个优秀的例子是 Wolfram Research 的 Mathematica,它是一个用于教学和科研的强大的数字和符号数学工具。Mathematica 是第一个支持 MathML 的商用数学工具。要获得 Mathematica 以及它对 MathML 进行支持的更多信息,请访问 Wolfram Research 的 Web 站点 <http://www.wolfram.com>。

为了明白 MathML 词表是如何便利于内容标记的两种类型的,让我们看一下下面的数学表达式:

$$x^3+6x+2=0$$

并不很复杂,是吧? 好的,让我们看一下编写这个等式的表达标记方法(如清单 36.1 所示)。

清单 36.1 使用 MathML 的表述标记编写的简单数学表达式

```
<math>
  <mrow>
    <mrow>
      <msup>
        <mi>x</mi>
        <mn>3</mn>
      </msup>
      <mo>+</mo>
      <mn>6+x</mn>
      <mo>+</mo>
      <mn>2</mn>
    </mrow>
    <mo>=</mo>
    <mn>0</mn>
  </mrow>
```

</math>

看起来,要表述这样的简单等式需要很多的代码。不幸的是,这是 MathML 的本性,这也就是为什么 MathML 内容不适合于手工编写的原因。就像你能够从代码中所看到的那样,表述标记自身只关注数学表达式的外表。例如,msup 标记用来说明 3 是 x 的上标。同样,mo 标记用来插入操作符,这里就是 + 和 = 操作符。你可以处理这个例子的表述标记并获得有意义的结果,这一点毫无疑问,因为你可以假定上标代表幂的意思。然而,通常,只使用表述标记并不足以处理复杂的等式。

现在让我们看一下内容标记与表述标记有怎样的不同。清单 36.2 包含了使用 MathML 内容标记编写的同一表达式。

清单 36.2 使用 MathML 的内容标记编写的简单数学表达式

```
<apply>
  <plus/>
  <apply>
    <power/>
    <ci>x</ci>
    <cn>3</cn>
  </apply>
  <apply>
    <times/>
    <cn>6</cn>
    <ci>x</ci>
  <apply>
    <cn>2</cn>
  </apply>
```

如你所见,这个代码使用了与表述标记代码完全不同的标记集。这个代码说明表述标记和内容标记实际是存在于同一个 MathML XML 应用中的两个差别很大的词表。与清单 36.1 中的表述标记不同的是,清单 36.2 中的内容标记包含的标记有 plus、power 和 times,这些表述了等式的语义。这些语义标记是计算和处理数学内容的关键。另一个有趣的不同点就是这段代码中使用后缀符号,这包括在操作数前指定数学操作符。

注意:在本书编写期间,已经有了一些用于编辑和浏览 MathML 内容的工具。然而,这些工具的绝大多数都适用于 MathML 的表述标记部分而普遍不支持内容标记。你将在本章后面学到部分这样的工具。

现在你已经对这两种 MathML 提供的标记解决方案之间的区别有了一定的了解。你实际已经深入到了标记 MathML 词表的专门元素。不管 MathML 文件是用表述标记还是内容标记编写的,它必须使用 math 元素。math 元素是 MathML 文件的文件元素,只要编写代码,它就必须使用,作为所有 MathML 内容的容器。你在清单 36.1 和 36.2 中已经看到了 math 元素作为文件元素的使用情况。

注意:要记住 MathML 代码通常嵌入到一个父文件之中,比如网页。这种情况下,

我们仍然需要使用 `math` 元素来承载 MathML 代码,即便父文件是一个 HTML 文件也是如此。你可以把 `math` 元素看作是用来指示 MathML 子文件的。

36.2.1 表述标记

如你在本章前面所学到的,MathML 的表述标记元素主要是为了实现显示的目的来编写数学内容的。由于表述标记本身并不很关注数学内容的结构和内容,它比内容标记需要更少的元素。最为常用的表述标记元素就是记号元素。下面是一些最常用的记号元素:

- `mi`——标识符
- `mn`——数字
- `mo`——操作符、包围符(圆括号)或分离符

记号元素在表述标记中非常的重要,因为它们是惟一一类直接构造文本内容的元素;所有其他的记号元素不是空元素就是容器元素。因此,所有的标识符,操作符和数字使用记号元素编写。

`mi` 元素用来指定表述标记表达式的标识符。标识符通过确定它们是单字符或是多字符标识符区别显示。诸如 `x` 和 `y` 这样的单字符标识符通过斜体显示,而诸如 `sin` 和 `tan` 这样的多字符标识符不用斜体显示。`mn` 元素用来指定数字,它们都不用斜体显示。

`mo` 元素用来指定操作符,这在数学内容中非常的多。具体操作符如何显示主要依赖于操作符自身和用来浏览 MathML 文件的软件。操作符周围的间距是变化的,这主要与操作符的大小设置和在表达式中的位置关系有关。`mo` 元素也可以用来标记其他的数学对象,比如圆括号和重音符,它们是在实际的数学意义中不需考虑的操作符。

虽然记号元素在 MathML 文件中扮演了重要的角色,如果没有把它们安排进表达式中的方法它们也是一无用处的。布局元素用来在 MathML 文件中放置记号。下面是一些表述标记中最常用的布局标记:

- `mrow`——水平地组合任何数量的子表达式
- `mfrac`——构造一个有两个子表达式的分式
- `msqrt`——构造一个根号
- `mroot`——构造一个 n 次根号
- `mfenced`——用一对包围符(圆括号)包围内容

`mrow` 元素用来水平地放置数学内容,它可以包含任何数量的子元素。`mfrac` 元素用来指明一个分式。因此,`mfrac` 元素需要两个子元素:第一个是分子,第二个是分母。

`msqrt` 元素用来指定一个根号,它可以包含任意数量的子元素。所有的包含在 `msqrt` 元素中的元素都在根号的下面显示。`mroot` 元素用来指定一个有开方的表达式。`mroot` 元素与 `msqrt` 元素类似,除了你必须指定开方数以外。因此,`mroot` 需要两个子元素,其中的第二个是开方数。

`mfenced` 元素用来将数学内容组织到括号中。它与 `mrow` 的操作非常的类似,只是在周围附加了括号。

这些就是总体上用来在 MathML 中创建表述标记的基本元素。就像你所猜测的那样,我提到的元素只是表述标记词表中的一小部分。在转移到内容标记之前,让我们花些时间熟

悉一下用在表述标记中的一些其他元素。下面是表述标记的脚标和限制元素：

- `msub`——给基数附加一个下标
- `msup`——给基数附加一个上标
- `msubsup`——给基数附加一个“下标-上标”对
- `munder`——给基数附加一个底标
- `mover`——给基数附加一个顶标
- `munderover`——给基数附加一个“底标-顶标”对
- `mmultiscripts`——给基数附加前置命令符和张量指数

下面是表述标记的表元素：

- `mtable`——一个表或矩阵
- `mtr`——表或矩阵的一行
- `mtd`——表或矩阵中的一个实体
- `maligngroup`——一个队列组标
- `malignmark`——一个队列点标

你也许想知道为什么我没有向你展示如何用表述标记元素来创建 MathML 文件。浏览 MathML 内容需要专门的工具,因此我将在本章的后面介绍一些 MathML 工具的时候列举一些例子。稍后我们再进行举例。

36.2.2 内容标记

内容标记提供了 MathML 等式的另一半。因为内容标记的目的是正确的反映数学内容的语义和结构,MathML 的内容标记部分要比与它相对的表述部分大得多。这是因为数学内容的语义比起表述这个数学内容来说更难构造,最终它们需要更多的元素。因此请不要对 MathML 词表中与内容标记相关的元素的巨大数目感到过于的震惊。

在提供内容标记元素之前,让我重申一下,支持内容标记的浏览器和工具在本书编写期间仍然是滞后的。迄今为止,MathML 软件主要支持表述标记,因为显然它支持起来更简单。在这种情况下,就造成了很难在使用内容标记时能够极度的兴奋,而只是因为没有使用这些结果工作的方法。因此,我决定不在 MathML 的内容标记元素的深入工作中花费过多的时间。

本节的剩余部分将向你介绍内容标记词表的主要领域,以及各自领域中的主要元素。如果你想获得有关 MathML 的内容标记部分的更多信息,请访问 MathML 规范的网址 <http://www.w3.org/TR/REC-MathML>。

下面是内容标记的基本内容元素：

- `apply`——对函数的直接调用
- `reln`——等式或关联
- `fn`——用户定义函数
- `inverse`——逆
- `sep`——数值间的分隔

- declare——声明
- lambda——表达式的函数结构
- compose——两个或多个函数的合成
- ident——恒等函数

下面是内容标记的算术、代数和逻辑元素：

- quotient——对基数做除法所得的商
- exp——求幂
- factorial——阶乘
- divide——除
- max——最大
- min——最小
- minus——减
- plus——加
- power——幂
- rem——除法所得余数
- times——乘
- root——N 次根
- gcd——最大公约数
- and——布尔与
- or——布尔或
- xor——布尔异或
- not——布尔非
- forall——通用量词
- exists——存在量词
- abs——绝对值
- conjugate——复杂共轭

下面是内容标记的关系元素：

- eq——相等
- neq——不等
- gt——大于
- lt——小于
- geq——大于等于
- leq——小于等于

下面是内容标记的微分元素：

- ln——自然对数
- log——给定基数对数
- int——积分

- diff——导数(微分)
- partialdiff——局部导数
- lowlimit——下限
- uplimit——上限
- bvar——绑定变量
- degree——微分次数
- logbase——对数的基数

下面是内容标记的集合论元素：

- set——集合
- list——序列
- union——并集
- intersect——交集
- in——是成员
- notin——不是成员
- subset——是子集
- prsubset——是真子集
- notsubset——不是子集
- notprsubset——不是真子集
- setdiff——集合差别

下面是内容标记的排列和组合元素：

- sum——排列总和
- product——增加排列项
- limit——限制排列值
- tendsto——排列的关系式

下面是内容标记的三角学元素：

- sin——正弦函数
- cos——余弦函数
- tan——正切函数
- sec——正割函数
- csc——余割函数
- cot——余切函数
- sinh——双曲正弦函数
- cosh——双曲余弦函数
- tanh——双曲正切函数
- sech——双曲正割函数
- csch——双曲余割函数
- coth——双曲余切函数

- arcsin——反正弦函数
- arccos——反余弦函数
- arctan——反正切函数

下面是内容标记的统计元素：

- mean——平均数或平分数
- sdev——正则除
- var——方差
- median——中值
- mode——模
- moment——矩

下面是内容标记的线性代数元素：

- vector——矢量
- matrix——矩阵
- matrixrow——矩阵行
- determinant——行列式
- transpose——转置
- select——选择一个矩阵行或实体

36.3 创建 MathML 内容

你已经对组成 MathML 词表中的表述和内容标记部分的元素有了一定的认识。你已经准备好来研究如何用这些元素创建 MathML 内容。实际上,对于内容标记还没有什么软件可以支持,因此我将集中于表述标记。作为开始,让我们看一下如何将 MathML 内容嵌入到 HTML 网页中(如清单 36.3 所示)。

清单 36.3 在 HTML 文件中嵌入 MathML 代码

```
<! DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
  </head>
  <body>
    <math>
      <mrow>
        <mrow>
          <msup>
            <mi>x</mi>
            <mn>3</mn>
          </msup>
          <mo>+</mo>
          <mn>6x</mn>
        </mrow>
      </mrow>
    </math>
  </body>
</html>
```



```

        <mo>+</mo>
        <mn>2</mn>
      </mrow>
      <mo>=</mo>
      <mn>0</mn>
    </mrow>
  </math>
</body>
</html>

```

这个代码显示了 MathML 代码是如何直接嵌入到 HTML 文件的 body 部分的。支持 MathML 的浏览器将识别 math 标记并知道如何显示所包含的 MathML 代码。你也许认出来这个清单中的 MathML 代码表示的就是本章前面的等式。

这个等式非常的简单,且没有包含表达式的分组的方法。下面是一个依靠括号分组的表达式:

$$(5x+3y)^3-3x^2+4y=0$$

清单 36.4 包含了使用 mfenced 标记进行圆括号分组来表述这个等式的 MathML 代码。

清单 36.4 使用 MathML 的表述标记和 mfenced 标记编写的数学等式

```

<math>
  <mrow>
    <mrow>
      <msup>
        <mfenced>
          <mrow>
            <mn>5x</mn>
            <mo>+</mo>
            <mn>3y</mn>
          </mrow>
        </mfenced>
        <mn>3</mn>
      </msup>
      <mo>-</mo>
      <msup>
        <mi>3x</mi>
        <mn>2</mn>
      </msup>
      <mo>+</mo>
      <mn>4y</mn>
    </mrow>
    <mo>=</mo>
    <mn>0</mn>
  </mrow>
</math>

```

注意 mfenced 元素用来在等式的某一位置放置括号。清单 36.5 包含了 MathML 代码的另一个例子,它使用 mroot 和 mfrac 标记来示范如何使用根号和分式。

清单 36.5 使用 MathML 的表述标记和 mroot 和 mfrac 标记编写的数学等式

```

<math>
  <mfrac>
    <mrow>
      <mn>2x</mn>
      <mo>+</mo>
      <mn>5</mn>
    </mrow>
  </mfrac>
  <mn>7</mn>
</mfrac>
</mrow>
<mn>3</mn>
</mroot>
</math>

```

这样你也就会想到 MathML 可以用来将复杂的数学内容放在一起,让我们看一下清单 36.6。

清单 36.6 使用 MathML 的表述标记编写的二次方程式

```

<math>
  <mrow>
    <mi>x</mi>
    <mo>=</mo>
    <mfrac>
      <mrow>
        <mrow>
          <mo>-</mo>
          <mi>b</mi>
        </mrow>
        <mo>&PlusMinus;</mo>
        <msqrt>
          <mrow>
            <msup>
              <mi>b</mi>
              <mn>2</mn>
            </msup>
            <mo>-</mo>
            <mn>4ac</mn>
          </mrow>
        </msqrt>
      </mrow>
      <mn>2a</mn>
    </mfrac>
  </mrow>
</math>

```

如果你的数学有点忘了,或者你并不像我一样经过工科学院的学习,那么我告诉你这个

清单编写了二次方程式。你也许注意到一个实体引用——PlusMinus,用来确定正/负号。另外,二次方程式也可以使用一般的 MathML 表述标记编写。

36.4 MathML 工具

你已经学习了大量有关如何使用 MathML 编写数学内容的知识,起码以表述的目的来说确是如此,但你还要学习如何浏览这个内容。本章的剩余部分将集中讲解使用表述标记创建和浏览 MathML 内容的可用软件工具。下面是本书编写期间一些最流行的 MathML 工具:

- Amaya 编辑器和浏览器
- WebEQ 数学浏览器 Java applet
- IBM 的 techexplorer Hypermedia 浏览器
- MathType 等式编辑器
- EzMath 编辑器和插件
- Mathematica 科学计算工具

下面的几节将探讨这些工具的基础知识以及它们提供的功能。你还将看到那些你在本章已经看过的 MathML 代码是如何在 MathML 工具中显示的。

36.4.1 Amaya 编辑器和浏览器

Amaya 是 W3C 创建的浏览和制作工具。它计划作为新的 Web 协议和数据格式的论证工具和测试平台。主要思想就是 W3C 可以将新的 Web 技术结合进 Amaya 中进行测试和论证。这要比网络浏览器开发商所提供的支持要快的多。在本书编写期间,Amaya 已支持 MathML 的表述标记部分。要获得更多有关 Amaya 的信息,请访问 Amaya 的 Web 站点 <http://www.w3.org/Amaya>。

图 36.1 显示了清单 36.1 中的代码所表述的等式在 Amaya 中的显示效果。

图 36.2 显示了清单 36.4 中的代码所表述的等式在 Amaya 中的显示效果。

图 36.3 显示了清单 36.5 中的代码所表述的等式在 Amaya 中的显示效果。

图 36.4 显示了清单 36.6 中的代码所表述的等式在 Amaya 中的显示效果。

36.4.2 WebEQ 数学浏览器 Java Applet

WebEQ 是来自 Geometry Technologies 的数学浏览器 Java applet,它完全支持 MathML 的表述标记。使用 WebEQ,你可以将 MathML 代码直接嵌入的网页中并在任何支持 Java 的网络浏览器上浏览它们。要获得有关 WebEQ 数学浏览器 Java applet 的更多信息,请访问 WebEQ 的 Web 站点 <http://www.webeq.com>。

图 36.5 显示了 WebEQ 的 MathML 测试页,它显示了使用 MathML 代码的正弦定律。

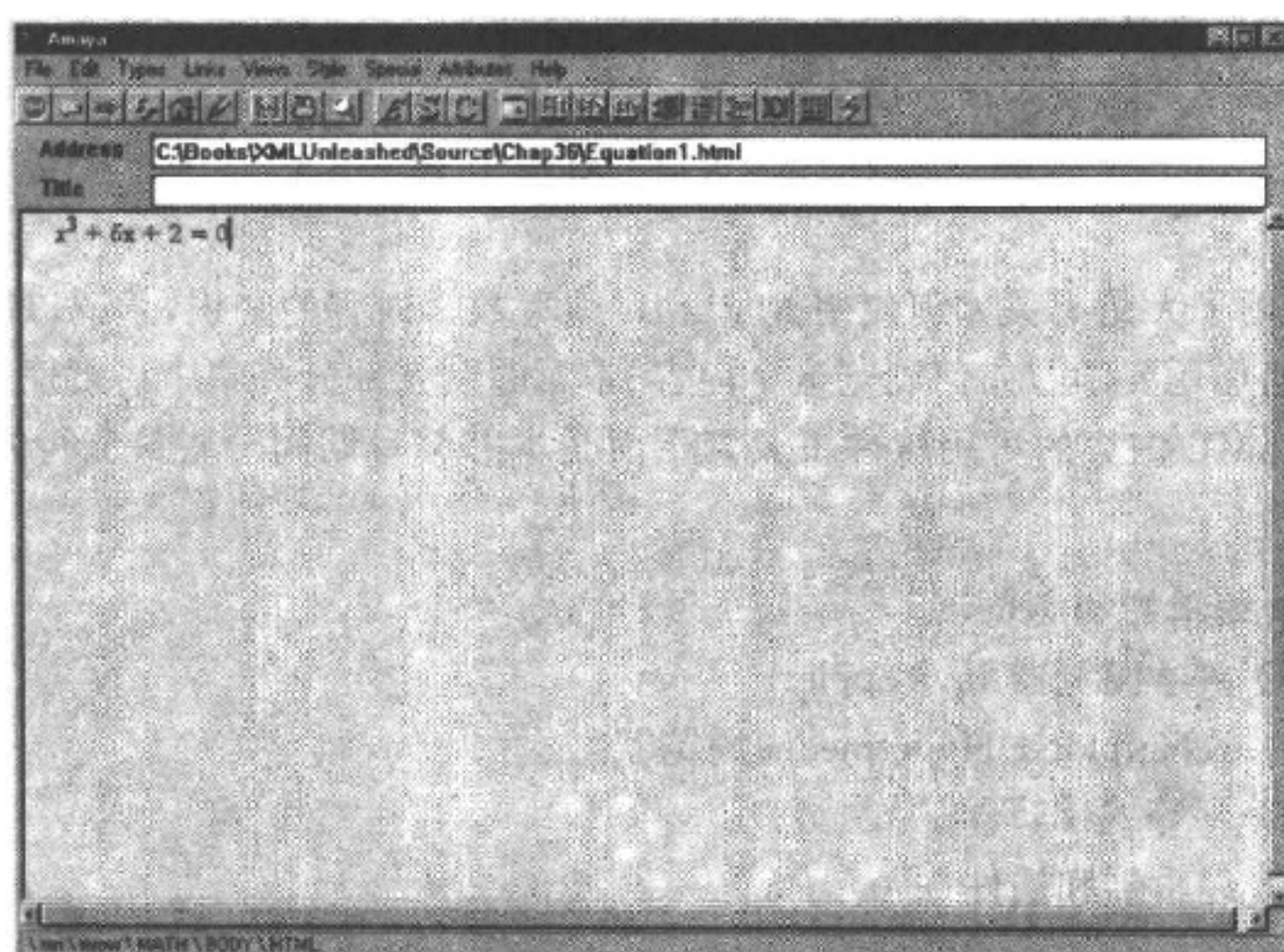


图 36.1 清单 36.1 的等式在 Amaya 中的浏览情况

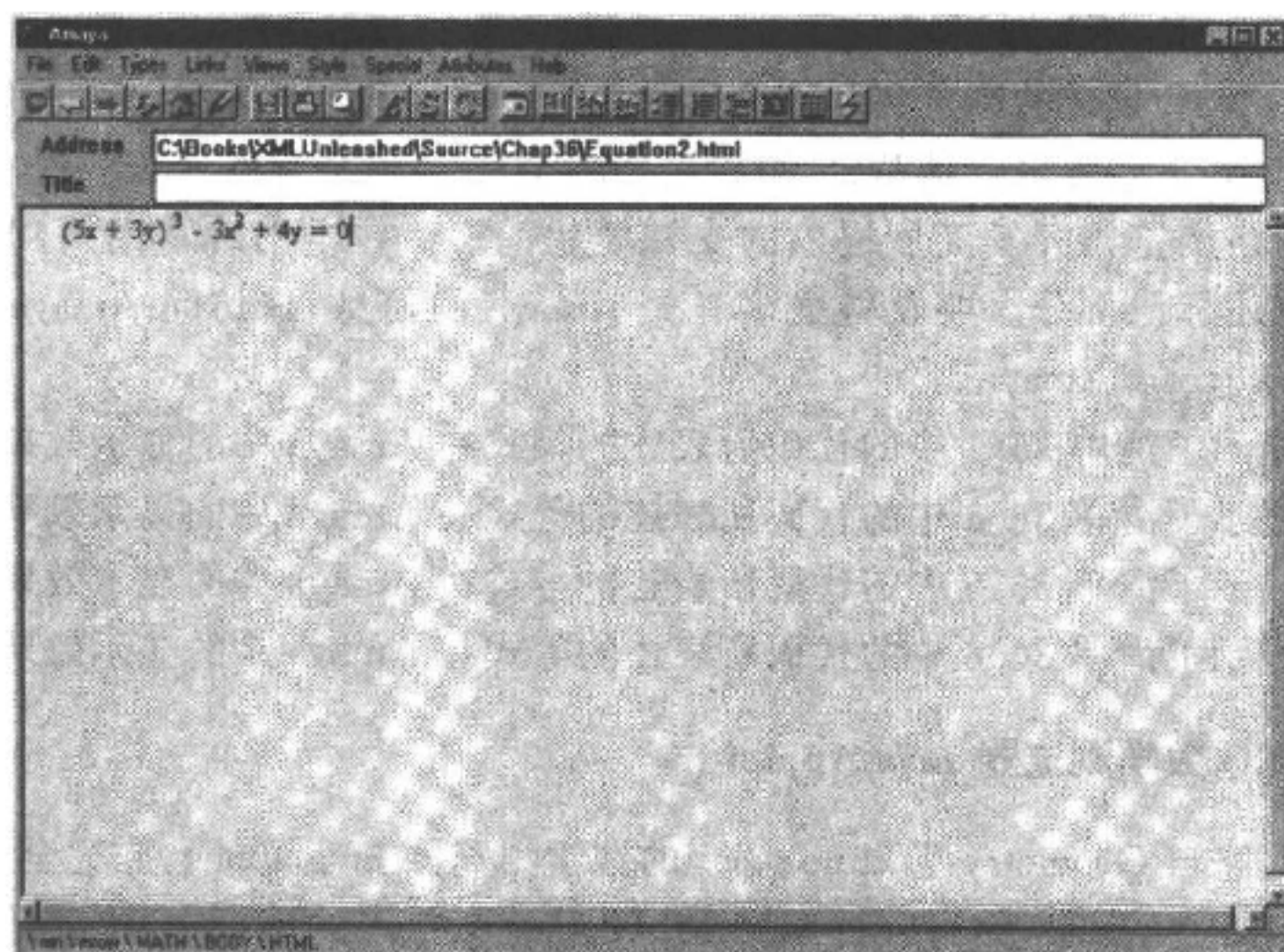


图 36.2 清单 36.4 的等式在 Amaya 中的浏览情况

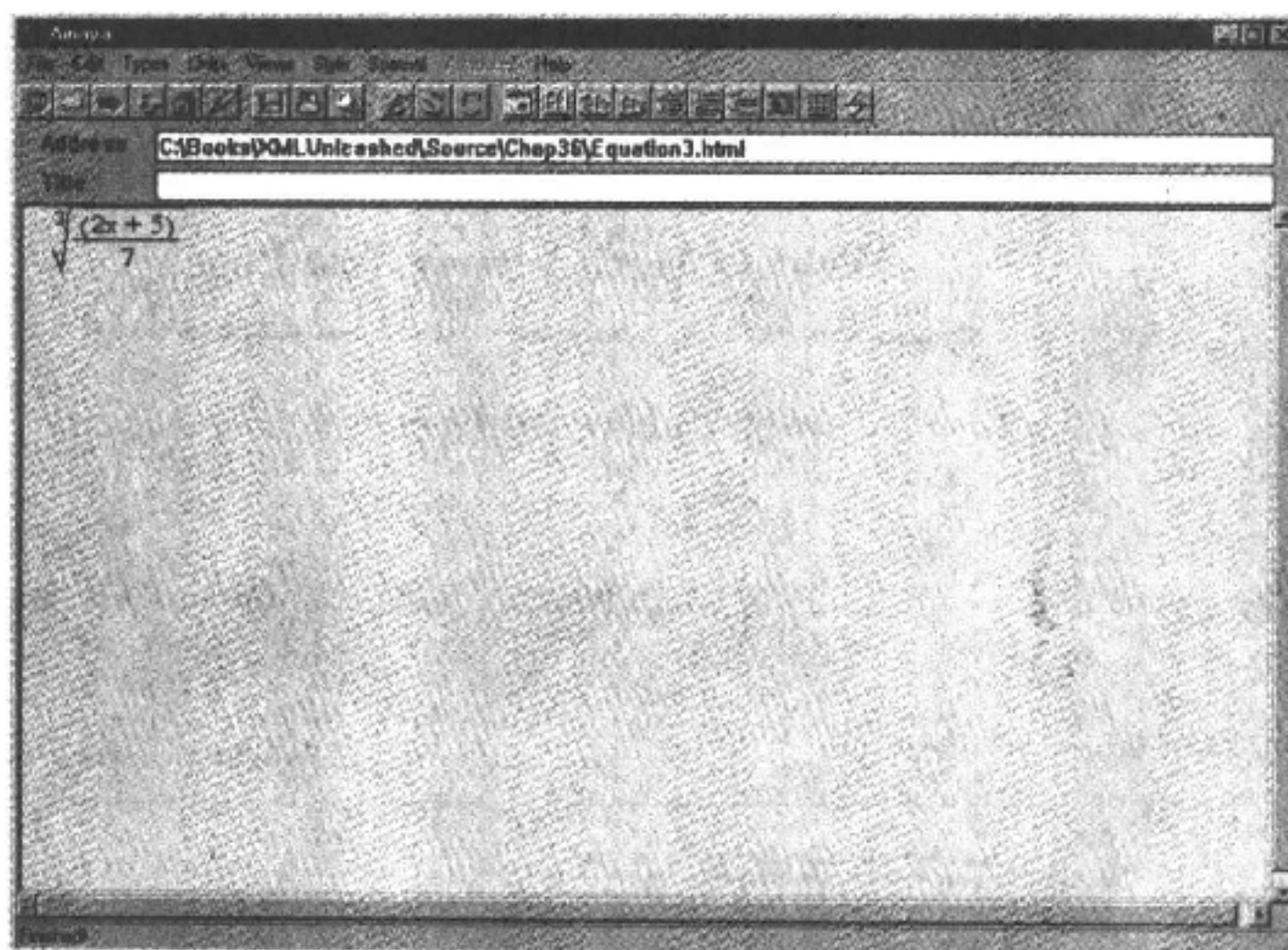


图 36.3 清单 36.5 的等式在 Amaya 中的浏览情况

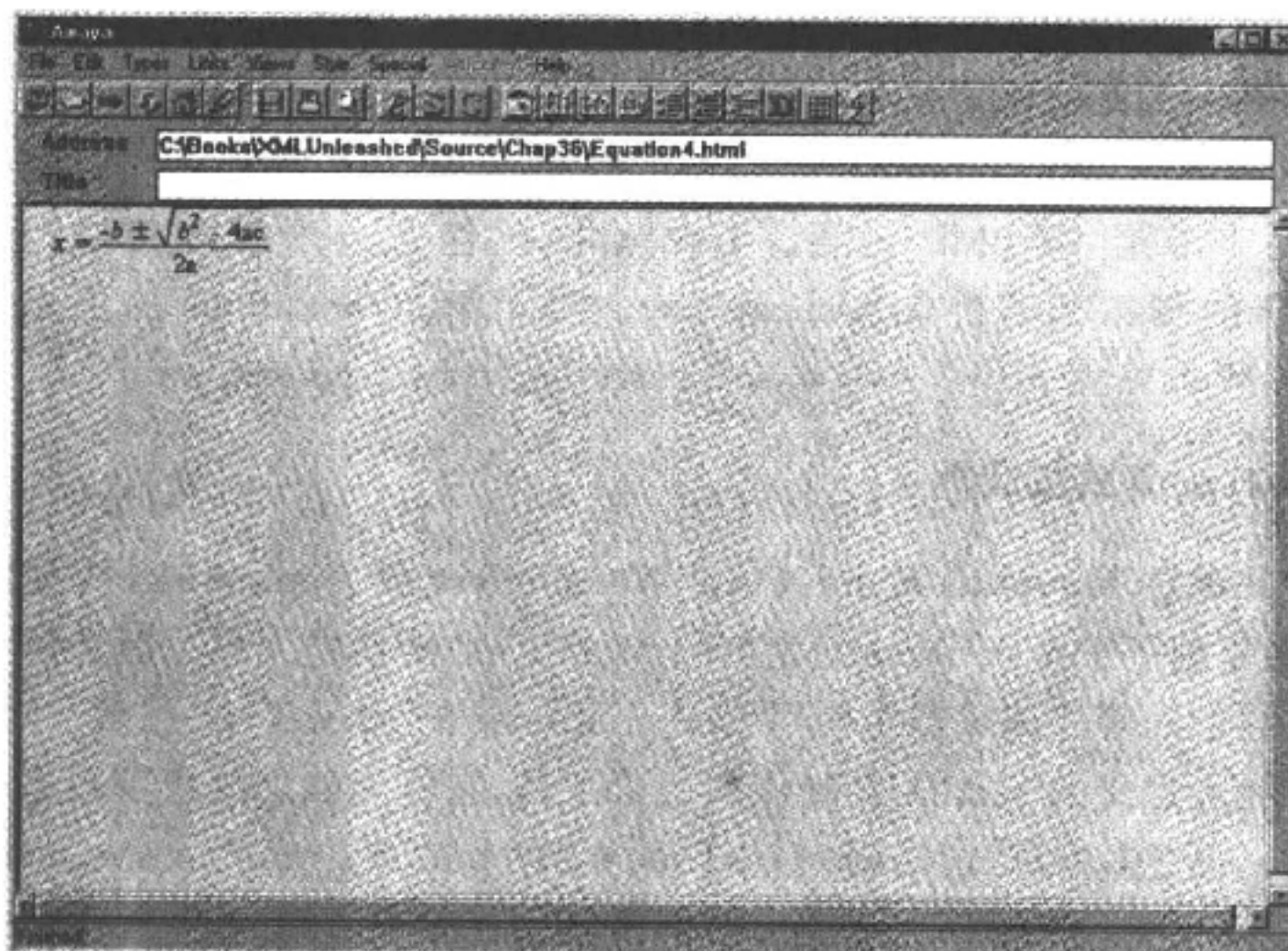


图 36.4 清单 36.6 的等式在 Amaya 中的浏览情况

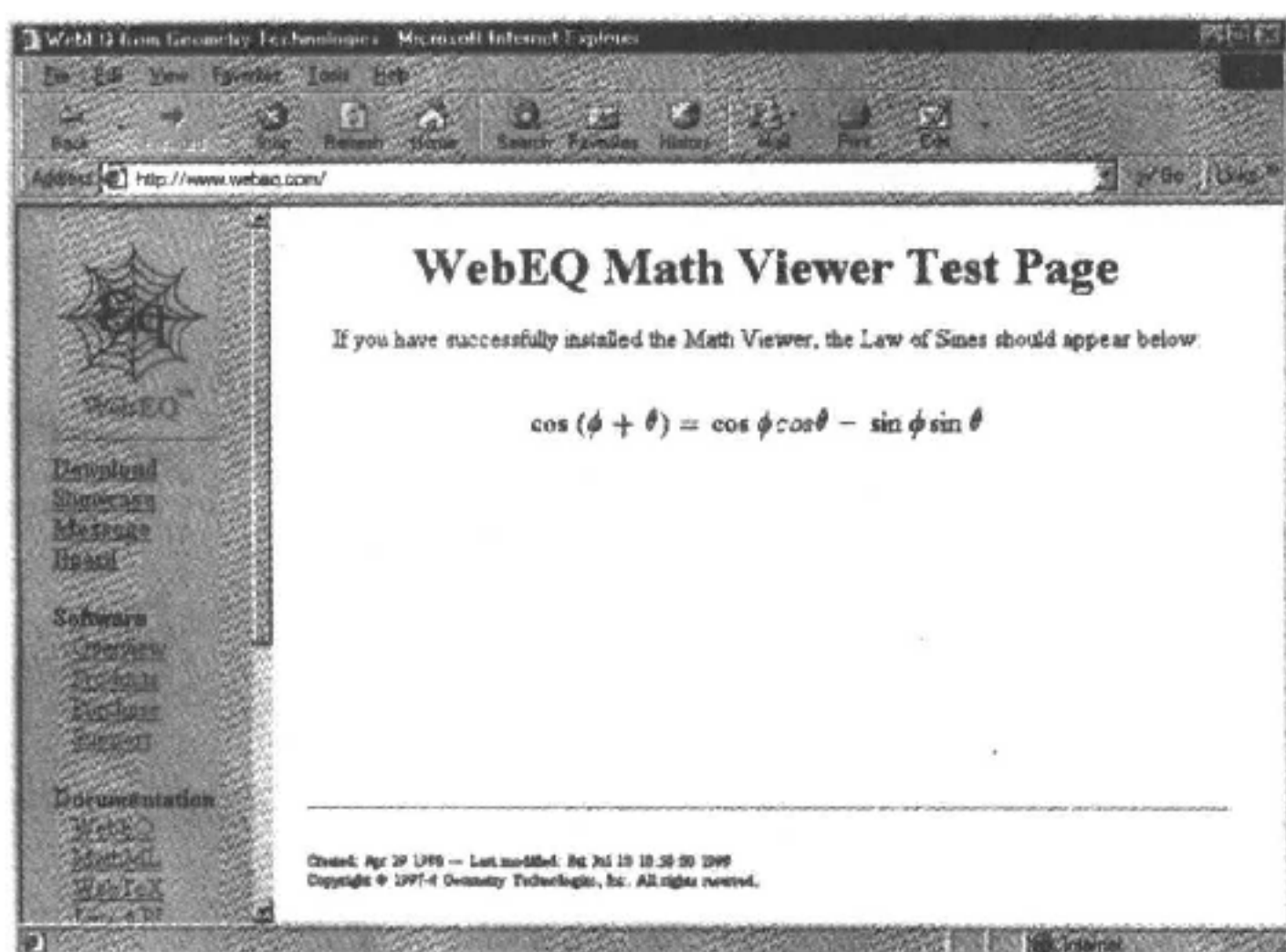


图 36.5 显示正弦定律的 WebEQ 测试页

36.4.3 IBM 的 techexplorer Hypermedia 浏览器

IBM 的 techexplorer Hypermedia 浏览器是为显示 MathML 的表述标记而设计,它也显示 LaTeX 排版代码。由于 techexplorer 是一个插件,它允许使 Internet Explorer 和 Netscape Navigator 的用户浏览 MathML 内容。Techexplorer 包含一个脚本接口,它允许用诸如 JavaScript 这样的脚本语言控制 techexplorer。要了解 techexplorer 的更多信息,请访问 techexplorer 的 Web 站点 <http://www.software.ibm.com/techexplorer>。

36.4.4 MathType 等式编辑器

如果你很熟悉在 Microsoft Word、Corel WordPerfect 或 Appleworks 中使用的等式编辑器,那么你已经了解了 Design Science 生产的 MathType 的功能。这是因为所有的这些文字处理函数都在它们的等式编辑器中使用 Design Science 的 MathType 技术。MathType 等式编辑器是一个完全意义的 MathML 制作和浏览工具,它作为独立的应用程序运行。要获得有关 MathType 的更多信息,请访问 MathType 的 Web 站点 <http://www.mathtype.com>。

图 36.6 显示了一些 MathType 输出的例子,它们在 MathType 的 Web 站点上。这些是复杂 MathML 内容的优秀例子,当然,你不会希望进行手工编写。

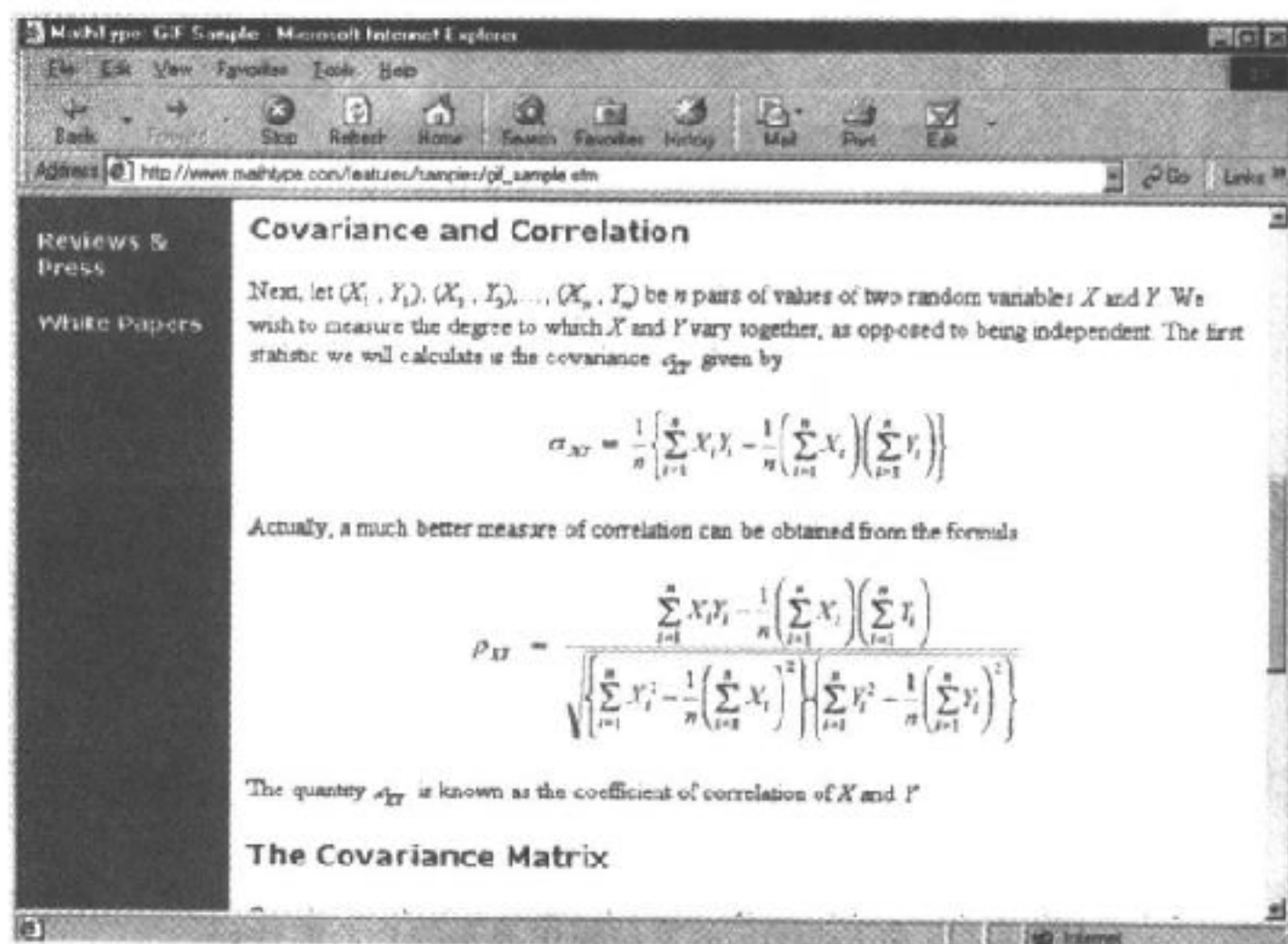


图 36.6 MathType 生成的等式的示例

36.4.5 EzMath 编辑器和插件

EzMath 是一个数学内容编辑和浏览解决方案,它由两部分组成:编辑器和插件。EzMath 编辑器是一个 Window 应用程序,它提供一个编写数学内容并以 MathML 内容标记导出的简单而又直接的接口。实际上 EzMath 中的数学内容并不真实的在可视方式下编辑;此外,你使用 EzMath 自身的数学符号表来创建数学表达式。幸运的是,EzMath 提供了使这个过程更易于理解的大量模板。一旦等式输入进 EzMath,你可以按 MathML 代码方式将它复制到剪贴板上。

注意:在本书编写期间,EzMath 是惟一支持 MathML 的内容标记的工具。

图 36.7 显示了 EzMath 中显示的矩阵等式。

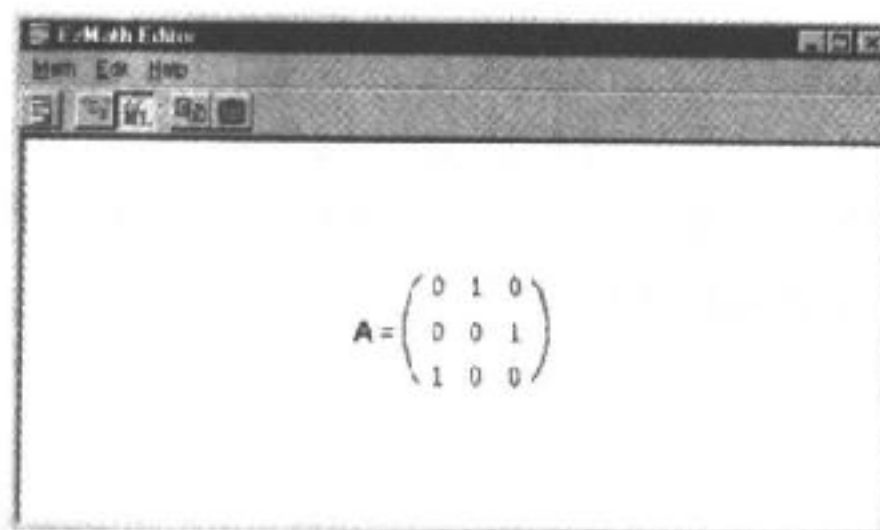


图 36.7 EzMath 中显示的矩阵等式

第 37 章 使用 P3P 管理个人隐私信息

你曾经在要提供给一个 Web 站点个人信息的时候因为不知道网站将如何使用这些信息而犹豫不决吗? 大多数的 Web 用户出于害怕会收到邮件炸弹或是成为一些在线骗局的目標等原因而拒绝提供信息。目前为止, 大多数的 Web 站点基本不存在保护个人隐私策略, 哪怕是最低限度的也没有。就是有一些站点提供了个人隐私机制, 你也没法保证它们可以信守诺言。

W3C 的 P3P (Platform for Privacy Preferences Project, 个人隐私权工程平台) 最初的目标就是标准化在 Web 上的个人隐私策略, 并授权给用户有关它们访问的站点的个人隐私策略的信息。毋庸置疑, P3P 是使用 XML 定义给定站点个人隐私策略的细节的技术。本章将探讨 P3P 以及它作为 XML 应用的相关性能。

37.1 P3P 基础

个人隐私策略是 Web 上急需标准化的一个领域。用户非常需要一些途径来在他们提交个人信息之前确定他们的个人信息将会被如何使用。W3C 认识到了这种需求并创建了 P3P 来处理这个问题。P3P 让用户确定参数来管理信息如何与 Web 站点共享。主要想法是: Web 站点提供个人隐私策略, 用户评定它并决定他以什么等级来在站点上共享信息。起码这个过程的一部分可以自动通过一个基于 XML 的词表来完成, 这个词表详细说明了个人隐私策略。

在 P3P 用语中, Web 站点被称为服务。你可以将 P3P 看成一个仲裁者, 它协助用户和服务在用户信息的维护和使用上达成协议。这个仲裁通过定义和执行个人隐私策略实现。仲裁进程在服务发给用户一个编码的有关负责这个站点的地址组织的个人隐私策略的提议时开始运行。这个提议以 XML P3P 文件方式编写, 这个文件主要是由用户的网络浏览器处理的, 它提供了有关它所收集的個人資料的详细信息, 以及如何处理这些数据的信息。

P3P 使用的 XML 词表就是协调词表, 基于 RDF (Resource Description Framework, 资源描述框架), 这是一个用来给 Web 资源添加语义的标准化词表。马上你就将学到有关 RDF 的更多知识。现在, 让我们继续对 P3P 进行讨论。P3P 协调词表包含描述服务器如何处理信息的元素。

由于个人隐私提议使用 XML 词表编写, 因而它们能够被网络浏览器或其他诸如插件或是代理服务器这样的有趣的应用程序自动地处理就十分正常了。这个处理应用程序被称为用户代理, 因为它为满足用户的利益而行动。用户代理负责将策略与用户的参数进行比较并自动确定什么信息可以提供给服务。主要思想是用户可以确定他们的个人隐私参数, 因此不必担心个人站点的个人隐私策略。如果服务的个人隐私提议与用户的参数冲突, 用户可以拒绝提议并拒绝提供信息。

P3P 被设计用来使组织可以根据他的 Web 站点所提供的不同服务提供多种个人隐私

策略。例如,一个购物网站可以只对那些加入到站点成为注册会员并提供个人信息的用户提供商品。一个专门的个人隐私策略将清楚严密的说明成员信息的组成以及组织如何计划使用这些信息(专门报价等等)。其他的用户仍然可以在普通个人隐私策略下访问站点,但它们不能访问销售页面。

如果迄今为止,P3P 听起来对你非常有用,你或许会自问一个重要的问题:什么使得组织遵守它们的个人隐私策略?回答是现在还没有用来强制遵守个人隐私策略的机制。然而,法律和独立督察组织可以主动的认定该组织是否遵守它们的个人隐私策略。TRUSTe 就是一个通过定期对注册的组织进行检查以协助相关组织遵守个人隐私策略的组织。要获得有关 TRUSTe 的更多信息,请访问它的 Web 站点 <http://www.truste.org>。

37.2 RDF 速成

我在前面提到过,用来编写个人隐私策略的 P3P 协调词表是基于 RDF 词表的。RDF 在第 17 章“浏览 XML”中简要的提到过。让我们花些时间进一步熟悉 RDF 并学习它为什么会构成 P3P 的基础。

RDF 是一个元语言,它用来给 Web 上的资源添加语义。大多数的 Web 资源仅仅包含内容而不包含任何有关内容性质的信息。例如,一个狮子的 GIF 图片并不包含任何机器可读的信息能指出这幅图是一个狮子。如果你希望寻找一个狮子的图片,你将不得不依赖于图片周围的文字,它们或许提供了有关图片内容的信息,或许没有。RDF 使用 XML 词表提供了一种描述 Web 资源的途径。在这个图片例子中,RDF 可以用来提供基于图片内容的强大的搜索,而不是基于周围的文字。

RDF 的本意是 Web 资源的自动处理。搜索引擎也许是这种自动处理的最好的例子。大多数的搜索引擎持续的扫描 Web 资源并尝试确定其中的数据的性质。每个搜索引擎有它们自己的方法改善搜索精度,使用流行的方式分析 Web 资源。RDF 通过提供给资源作者一种描述 Web 资源的确切性质的方法消除了大量的猜测工作。搜索引擎还可以使用 RDF 信息代替对内容本身的猜测信息作为搜索的基础。

RDF 数据模型基于特性和特性值。你可以将 RDF 特性作为 Web 资源的一个属性。RDF 特性还可以用来将资源相互关联。下面是组成基本 RDF 数据模型的三个对象类型:

- 资源(Resource)——这可以是一个完整的 HTML 文件或是 HTML 文件的一部分,比如是一幅图片或照片。它还可以接收更广泛的定义并包含整个 Web 站点。
- 特性(Property)——一个用来以某种方式描述资源的属性。
- 声明(Statement)——它由一个资源,一个特性以及特性值组成,声明的这三个部分分别称为主体(subject)、谓词(predicate)和对象(object)。

虽然这里不再更多的讲解 RDF,但这对于你使用 P3P 来说已经足够了。

注意:要获得有关 RDF 的更多信息,请访问 W3C 的 Web 站点 <http://www.w3.org/RDF>。

37.3 深入 P3P 协调词表

虽然 P3P 规范并不限制个人隐私策略为单一的词表,但 P3P 协调词表仍是制作个人隐私策略所普遍选择的词表。这个协调词表提供了一个基本的元素集合,它可以用来表述大范围的个人隐私策略。下面是在协调词表中定义的核心元素:

- PROP
- ASSURANCE
- REALM
- VOC;DISCLOSURE
- STATEMENT
- DATA;REF

下面将详细讲解这些元素。之后你将看到它们在一个完整的 P3P 个人隐私策略中的表现。

37.3.1 PROP 元素

PROP 元素用来作为保存其他 P3P 元素的容器。从功能上来讲,它有些像 P3P 内容的文件元素。然而,由于 P3P 实际上是基于 RDF 的,所以所有的 P3P 文件需要 RDF 元素作为它们的文件元素。PROP 元素作为 P3P 表述的容器,它使用 STATEMENT 元素来编写。下面是 PROP 定义的最重要的属性:

- entity —— 一个确认对个人隐私策略负责的组织的 URI。
- agrexp —— 个人隐私协议的终止日期(默认是六个月)。

entity 属性是 PROP 元素惟一必要的属性,用来确认对个人隐私策略负责的组织。

37.3.2 ASSURANCE 元素

ASSURANCE 元素用来描述强制执行个人隐私策略的服务。ASSURANCE 元素就是你指定一个诸如 TRUSTe 这样的监督服务的地方。下面是 ASSURANCE 元素中定义的属性:

- service —— 确定担保服务的 URI。
- text —— 对担保服务的简单的文字描述(第三方、合法等等)。
- image —— 担保服务的图标的 URI。
- width —— 图标的宽度,单位是像素。
- height —— 图标的高度,单位是像素。
- alt —— 图标的简单文字叙述。

37.3.3 REALM 元素

REALM 元素用来描述个人隐私策略的相应领域,它由应用策略的资源组成。它定义了单一属性 `uri`,它是一个必要属性,用来确定资源或是资源组。`uri` 属性中指定的 URI 定义了个人隐私策略的应用领域,它由用户代理使用,用于确定用户和服务间是否存在协议。

37.3.4 VOC:DISCLOSURE 元素

VOC:DISCLOSURE 元素用来指定对有关服务的访问能力的披露。下面是 VOC:DISCLOSURE 元素定义的属性:

- `discURI` ——服务的用户可读个人隐私表述的 URI。
- `access` ——指定用户浏览确认信息以及提问和管理服务组织的能力。
- `other` ——指定服务是否更改协议以符合个人隐私表述中已知的个人隐私保持要求。

`DiscURI` 属性是 VOC:DISCLOSURE 元素惟一必要的属性,它包含服务的用户可读个人隐私表述的 URI。`access` 属性指定用户浏览确认信息以及提问和管理服务组织的能力。下面是 `access` 属性允许的值:

- 0——可确定数据没有被使用。
- 1——可确定接触信息。
- 2——其他的可确认信息。
- 3——没有。

`other` 属性用来指定服务是更改协议还是在个人隐私表述中保留已知的个人隐私。下面是 `other` 属性允许的值:

- 0——更改协议。
- 1——保留。

37.3.5 STATEMENT 元素

STATEMENT 元素用来指定 P3P 表述,它定义服务的实际个人隐私策略。下面是 STATEMENT 元素中定义的属性:

- `VOC:purp` ——处理用户数据的目的。
- `VOC:recpnt` ——用户数据发送的范围。
- `VOC:id` ——指定数据是否以用户个人定义的方式使用。
- `action` ——指定数据是只读还是读/写。

`VOC:purp`, `VOC:recpnt` 和 `VOC:id` 属性是 STATEMENT 必要的属性。`VOC:purp` 属性指出处理用户数据的目的。下面是 `VOC:purp` 属性允许的值:

- 0——当前行为的完成和支持。
- 1——Web 站点和系统管理。
- 2——定制个人站点。
- 3——研究和开发。
- 4——将访问者与市场的服务和产品相联系。
- 5——其他用途。

VOC:recpnt 属性指定用户数据传送的范围。下面是 VOC:recpnt 属性允许的值：

- 0——只有我们自己和我们的代理。
- 1——符合我们的标准的组织。
- 2——符合不同的标准的组织。
- 3——不相关的第三方或公共论坛。

VOC:id 属性指定数据是否以用户个人定义的方式使用。下面是 VOC:id 属性允许的值：

- 1——是
- 0——否

action 属性指定数据是只读还是读/写。下面是 action 属性允许的值：

- r——只读(默认)
- rw——读/写

37.3.6 DATA:REF 元素

DATA:REF 元素用来描述要传送的详细用户数据。下面是 DATA:REF 元素定义的属性：

- name —— 识别数据元素或数据集合的名称的字符串。
- value —— 包含在 name 属性中指定的数据元素的值的字符串。
- optional—— 指出数据元素是否必要。
- source—— 指定数据传送的机制。
- VOC:category—— 指明数据元素种类的字符串。

name 和 source 属性是 DATA:REF 元素必要的属性。name 属性指定一个数据元素和数据集合。P3P 的 Base Data Set 规范定义了可用于 P3P 的不同数据元素和集合。下面是一个包含在 User.data 集合中的数据元素和集合的列表，它包含了用户相关的主要信息：

- User.Name. —— 用户的姓名
- User.Bdate. —— 用户的生日

- User.Cert —— 用户的身份证
- User.Gender —— 用户的性别
- User.Employer —— 用户的雇主
- User.Department —— 用户工作的部门和公司
- User.JobTitle —— 用户的职称
- User.Home. —— 用户的家庭联系信息
- User.Business. —— 用户的商务联系信息
- User.BillTo. —— 用户的帐号
- User.ShipTo. —— 用户的执照号

你可以通过名字后的句点来判断这个数据是元素还是集合。句点指出这个数据是一个集合。例如,从前面的列表中,你可以看到 User.JobTitle 是一个数据,而 User.Business. 是数据集合。

注意:完整的 P3P Base Data Set 规范位于 <http://www.w3.org/TR/WD-P3P/basedata.html>。

source 属性是必要属性,它指定数据传送的机制。下面是 source 属性允许的值:

- 0——服务
- 1——代理
- 2——匹配形式
- 3——扩展

optional 属性指出数据元素是否必要。下面是 optional 属性允许的值:

- 0——必要(默认)
- 1——不必要

VOD:category 属性是指明数据元素种类的字符串。下面是 VOD:category 属性允许的值:

- 0——自然联系信息
- 1——在线联系信息
- 2——惟一标识
- 3——金融帐目标识
- 4——计算机信息
- 5——导航和点击流数据
- 6——办公数据
- 7——优先数据
- 8——人口统计和经济信息
- 9——内容

37.4 使用 P3P 创建个人策略

现在你已经基本理解了 P3P 中使用的词表,让我们通过看一个假想的 P3P 文件来对它们进行总体的观察。清单 37.1 包含一个 P3P 文件的代码,它摘录自 W3C P3P 规范。

清单 37.1 一个假想的 P3P 文件

```
<RDF:RDF xmlns:RDF=http://www.w3.org/TR/WD-rdf-syntax>
<PROP xmlns=http://www.w3.org/TR/WD-P3P/syntax
  xmlns:VOC=http://www.w3.org/TR/WD-P3P/vocab
  xmlns:DATA=http://www.w3.org/tr/WD-p3P/basedata
  entity=http://www.CoolCatalog.com>
  <ASSURANCE org=http://www.PrivacySeal.org
    text="third party" image=http://www.PrivacySeal.org/Logo.gif/>
  <REALM uri=http://www.CoolCatalog.com/catalog/>
  <VOC:DISCLOSURE discURI=http://www.CoolCatalog.com/PrivacyPractice.html
    access="3" other="0,1"/>
  <USES>
    <STATEMENT VOC:purp="2,3" VOC:recpnt="0" VOC:id="0"
      consq="a site with clothes you would appreciate">
      <DATA:REF name="Cookies_" source="0"/>
      <DATA:REF name="Form.Data_" VOC:category="8" optional="1" source="0"/>
    >
    <DATA:REF name="User.Gender" source="1"/>
  </STATEMENT>
</USES>
<USES>
  <STATEMENT VOC:purp="1,3" VOC:recpnt="0" VOC:id="0">
    <DATA:REF name="clickStream.Server_" source="0"/>
    <DATA:REF name="http.UserAgent_" source="0"/>
  </STATEMENT>
</USES>
</PROP>
</RDF:RDF>
```

有了在 P3P 词表中新得到的知识作为基础,你应该能够掌握这个文件的全面意义。服务是一个假想的站点,位于 <http://www.CoolCatalog.com>。提供对服务担保的监督机构是另一个假想的机构,位于 <http://www.PrivacySeal.org>。服务的领域是 Web 站点的 catalog 部分,它位于 <http://www.CoolCatalog.com/Catalog/>。服务的人们可读的个人隐私表述位于 <http://www.CoolCatalog.com/Privacypractice.html>。

继续明确个人隐私策略的细节,这个 P3P 文件处理 cookies、表单数据和用户性别的个人隐私问题。为了更明确,文件指定 cookies 使用服务传递。它还指定包含人口统计和经济数据的表单数据使用服务传递。最后,用户的性别被指定通过用户的代理传递。

37.5 Privacy Wizard

如果对于 P3P 词表的细节你已经觉得有点头昏眼花了,那么你一定很高兴得知你并不需要手工编写个人隐私策略。Microsoft 的 LinkExchange 服务作为 Microsoft Network 的一部分,提供了一个名为 Privacy Wizard 的在线工具,它自动完成创建 P3P 个人隐私策略的任务。Privacy Wizard 向你提出一系列有关你的站点在个人隐私方面的问题。在回答了这些问题后,Privacy Wizard 将生成人们可读的个人隐私表述网页以及使用 P3P 协调词表编写的 XML 文件。

在使用 Privacy Wizard 之前,你应该问自己一些有关你的个人隐私解决方案的问题:

- 你要告诉用户你如何使用他们的个人信息吗?
- 你授予用户选择他们的个人信息如何被使用的权利吗?
- 用户可以浏览并更新他们的个人信息吗?
- 用户的信息安全的处理并存储了吗?
- 你需要第三方认证(监督)机构吗?
- 你的站点在信息收集方面已经详细的考虑好了吗?

这些是 Privacy Wizard 在它的个人隐私问卷中的问题。

使用 Privacy Wizard 的第一步就是在 <http://privacy.linkexchange.com/pwSignup.asp> 上签约使用它。图 37.1 显示了 P3P Privacy Wizard 的注册页面。

LinkExchange - Control Center - My Profile - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search Favorites History Mail Print Exit

Address <http://secure.linkexchange.com/lan/ures/profile?returnURL=http://34/privacy.linkexchange.com/pw1at.asp> Go Links

LinkExchange

● Sign Up!

NOTE: If you've already signed up for any other LinkExchange service, you do not need to fill out this form again. Simply [click here](#) to login and continue.

FAQ: "Cookies" must be enabled to join. Need more information?

User Information

Email

City

First Name

State/Province

Last Name

ZIP/Postal code

Title (optional)

Country

Company/Org (optional)

Phone

Address

Fax (optional)

Select Password

Password (6 character minimum)

Enter Password Again

Next Reset

Done Internet

图 37.1 P3P Privacy Wizard 的注册页面

在你选定了用户名和密码后,你就可以使用 Privacy Wizard 了。

在你阅读了说明页面中的用法说明后,点击 Next 按钮进入到站点信息页面。这个页面允许你输入你的 Web 站点的主要信息。必要的信息是站点名称,URL 地址和可用来反馈的电子邮件地址。

Privacy Wizard 的下一步收集有关你的站点如何处理信息的主要信息。这一步允许你指定信息收集方式,比如你的站点是否收集个人用户信息,使用 cookies 和携带其他商业广告。图 37.2 显示了 P3P Privacy Wizard 信息收集步骤。

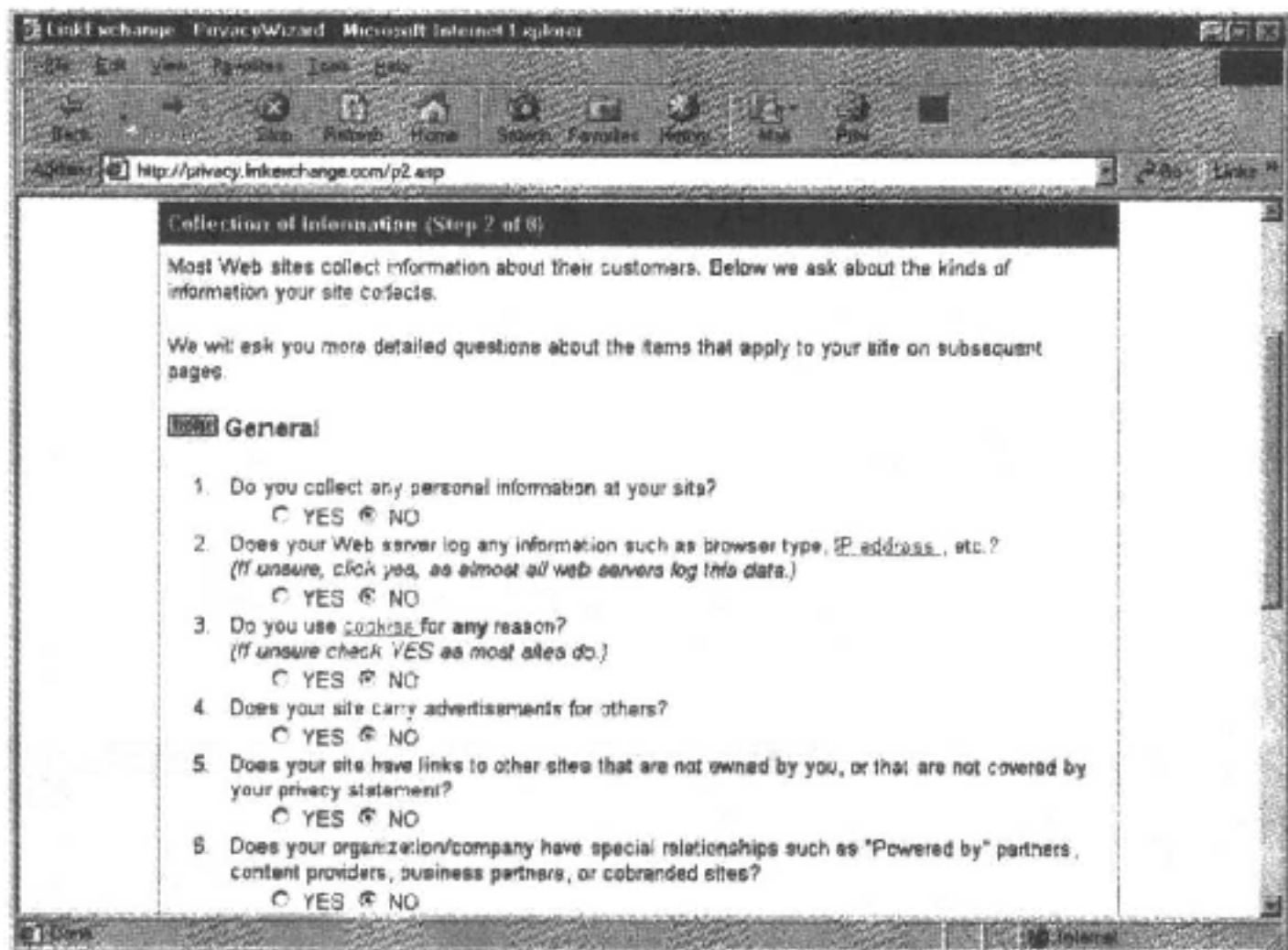


图 37.2 P3P Privacy Wizard 的信息收集步骤

指定了主要的信息后,你就可以继续确定个人用户信息如何被你的站点处理的细节了。这个步骤阐明了你的站点收集的准确的个人信息,比如联系信息和经济信息。图 37.3 显示了 P3P Privacy Wizard 的个人信息收集步骤。

Privacy Wizard 的下一个步骤就是确定你如何使用你的站点收集的信息。例如,大多数的用户关注于他们的联系信息是否被提供给了其他的公司。你可以在这一步中明确的指定这样的行为。图 37.4 显示了 P3P Privacy Wizard 使用收集的信息的步骤。

到这里你已经提供给 Privacy Wizard 足够的信息来产生用于你的站点的个人隐私策略了。然而,在继续之前有必要预览一下你的个人隐私策略。图 37.5 显示了 P3P Privacy Wizard 的个人隐私表述预览步骤。

下一步是接受个人隐私策略并进而生成实际的文件。

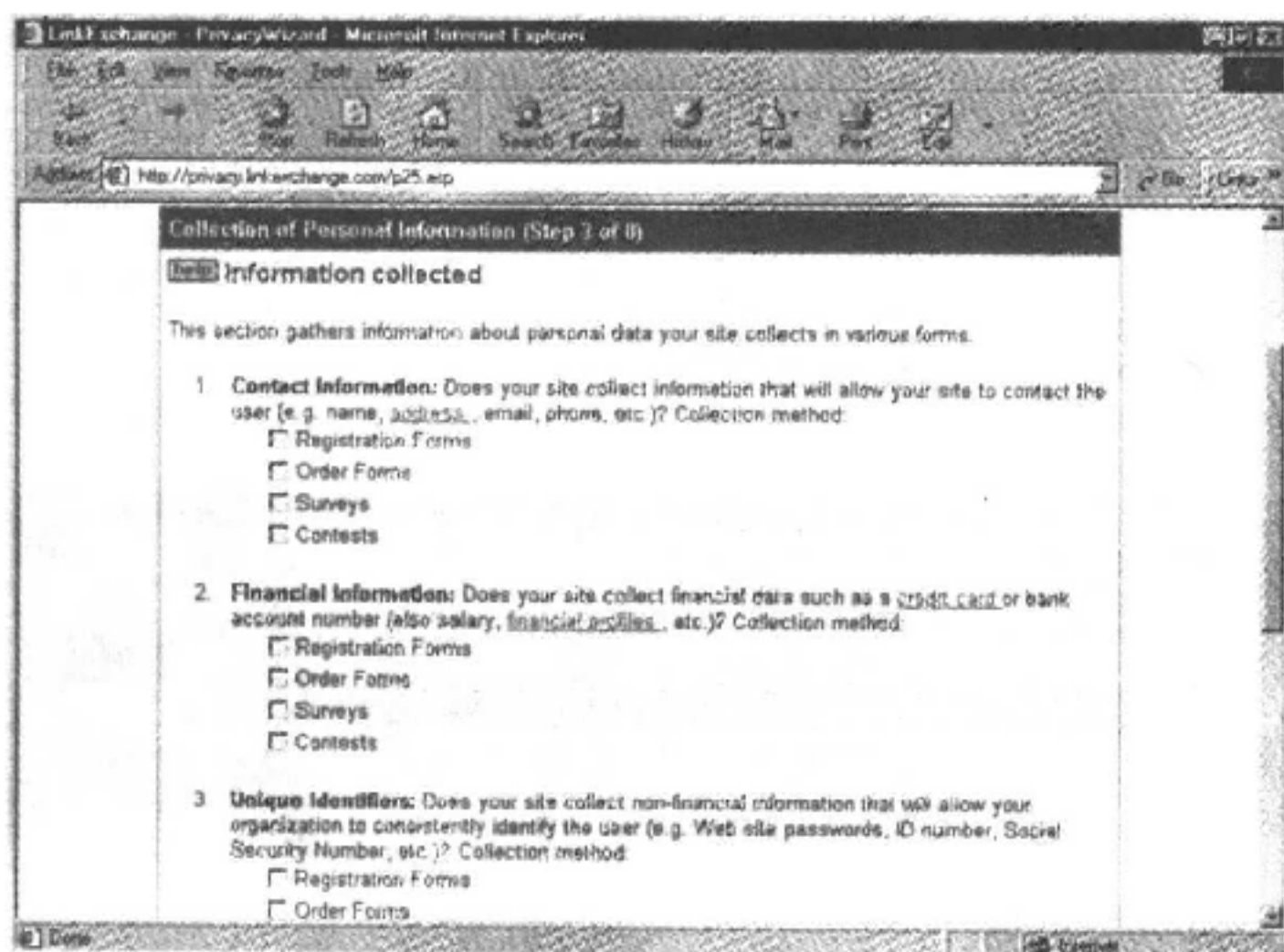


图 37.3 P3P Privacy Wizard 的个人信息收集步骤

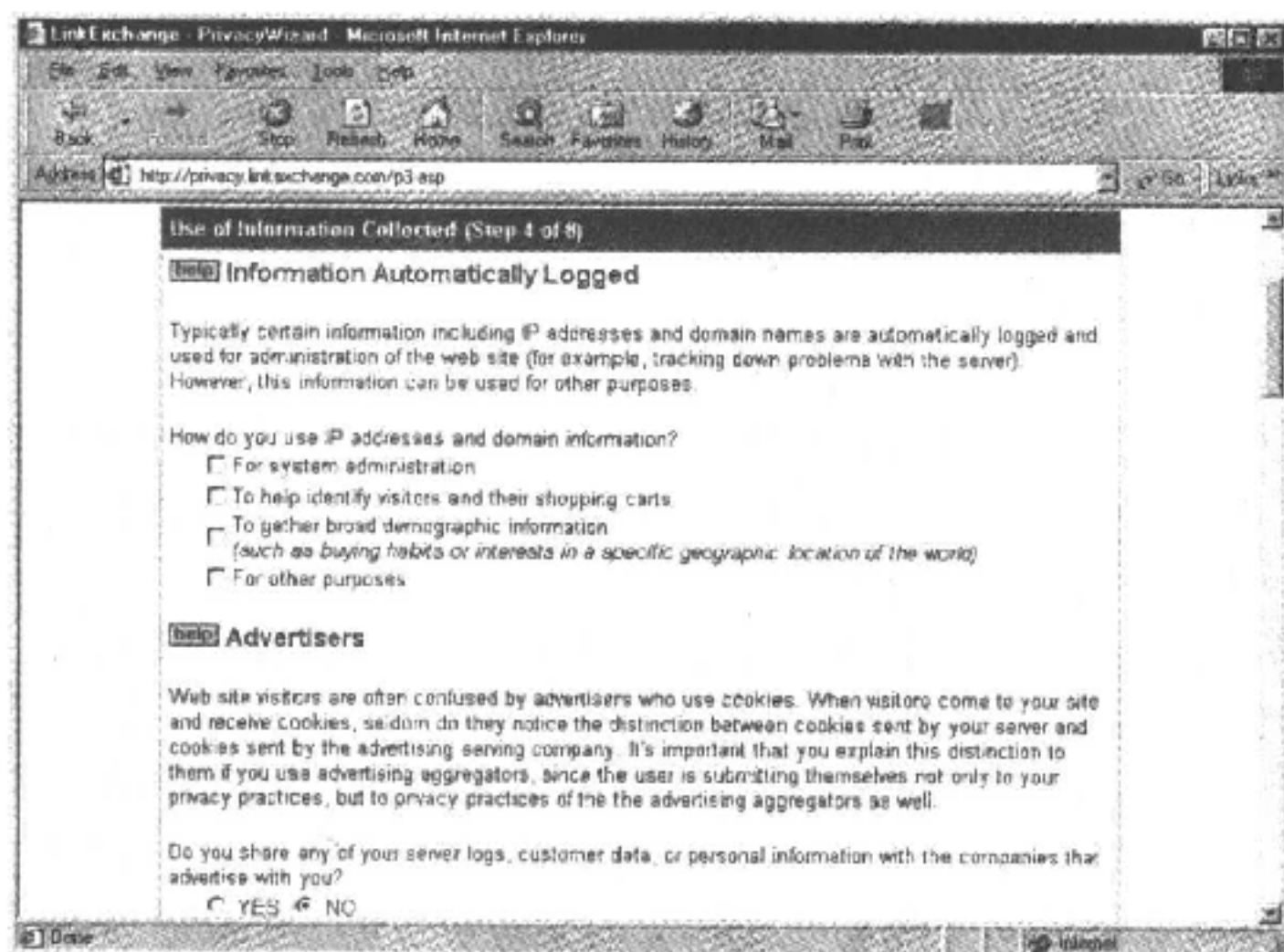


图 37.4 P3P Privacy Wizard 使用收集的信息的步骤

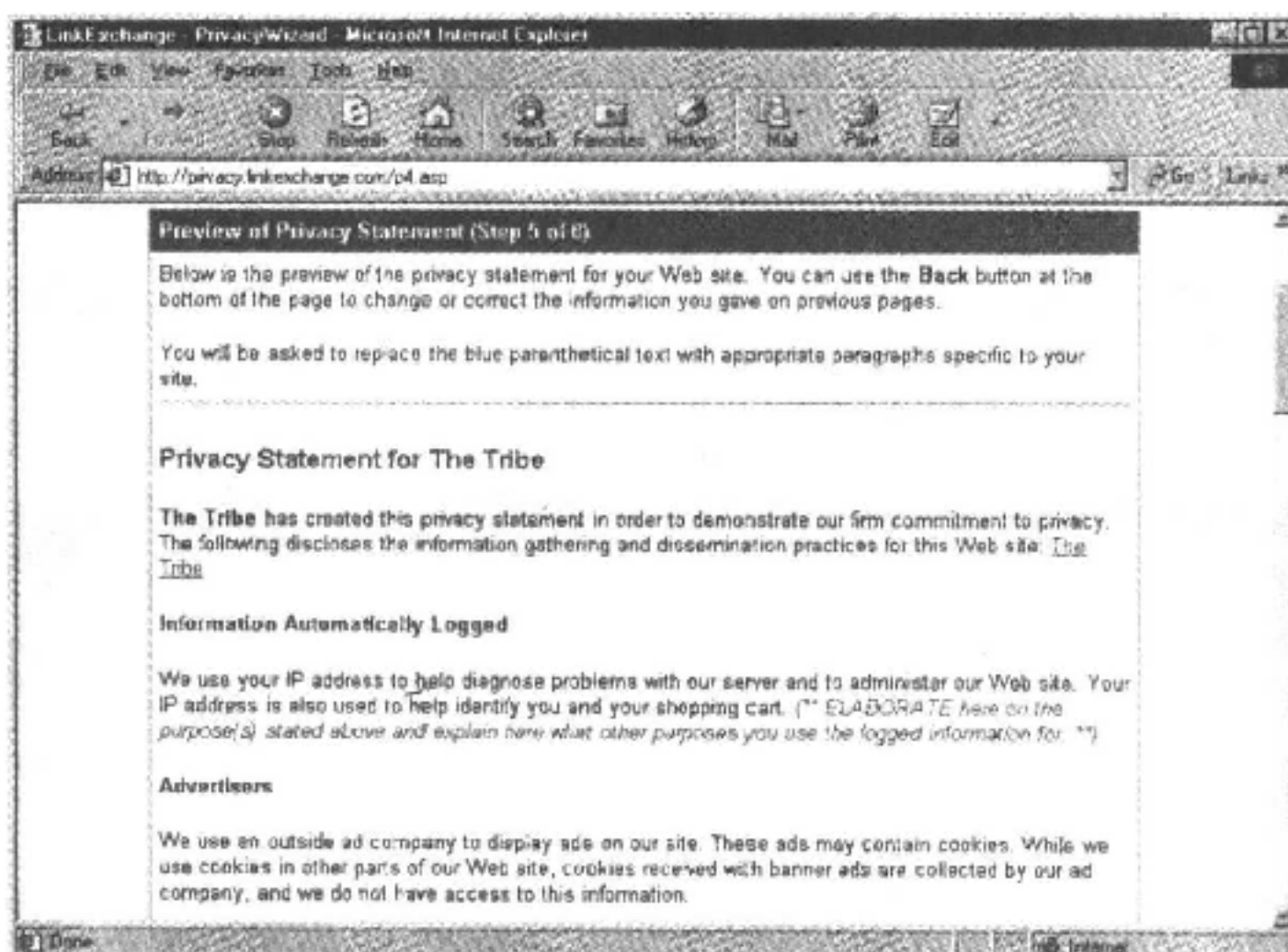


图 37.5 3P Privacy Wizard 的个人隐私表述预览步骤

在确定个人隐私策略之后,你必须提供个人隐私表述文件的位置,这样 LinkExchange 才可以引用它。这个位置非常的重要,因为它允许外部源检查你的个人隐私策略。个人隐私策略必须在完成后在这个位置公布。图 37.6 显示了 P3P Privacy Wizard 的概述。

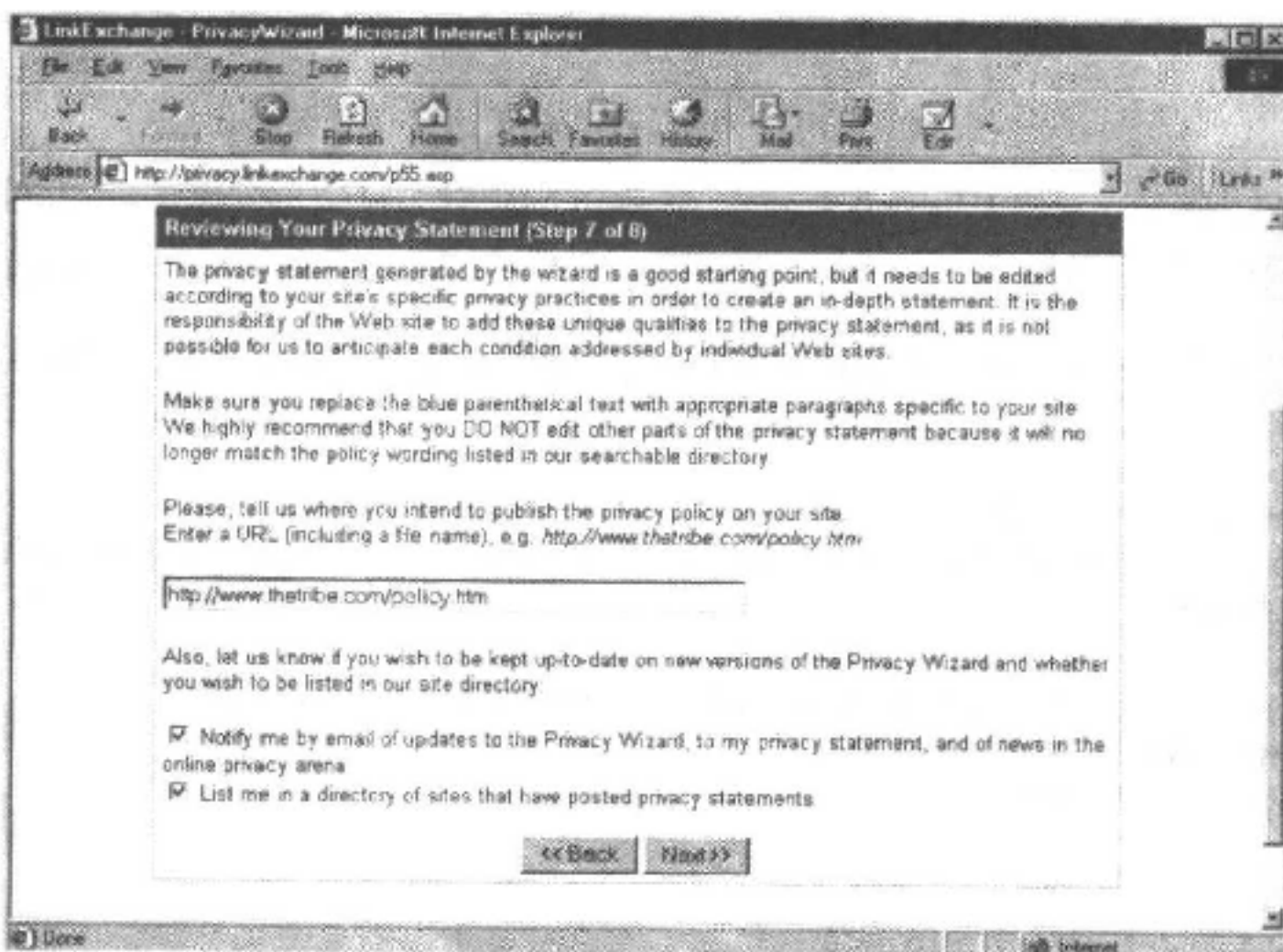


图 37.6 3P Privacy Wizard 的概述

下一步就是下载个人隐私策略的实际 HTML 和 XML 文件。这些文件可以放置在你的站点上以供用户浏览(HTML)和用户代理处理(XML)。下载的文件名为 statement.htm 和 p3policy.xml。Privacy Wizard 自动生成这些文件并指导你下载它们。之后你就要负责将它们在你的站点上公布出来。

在下载了个人隐私策略文件之后,Privacy Wizard 建议你加入 TRUSTe 监督组织。这一步当然是可选的,但它将可以向你的用户提供信心保证,使他们相信你遵守你的个人隐私策略。图 37.7 显示了 P3P Privacy Wizard 的完成步骤。

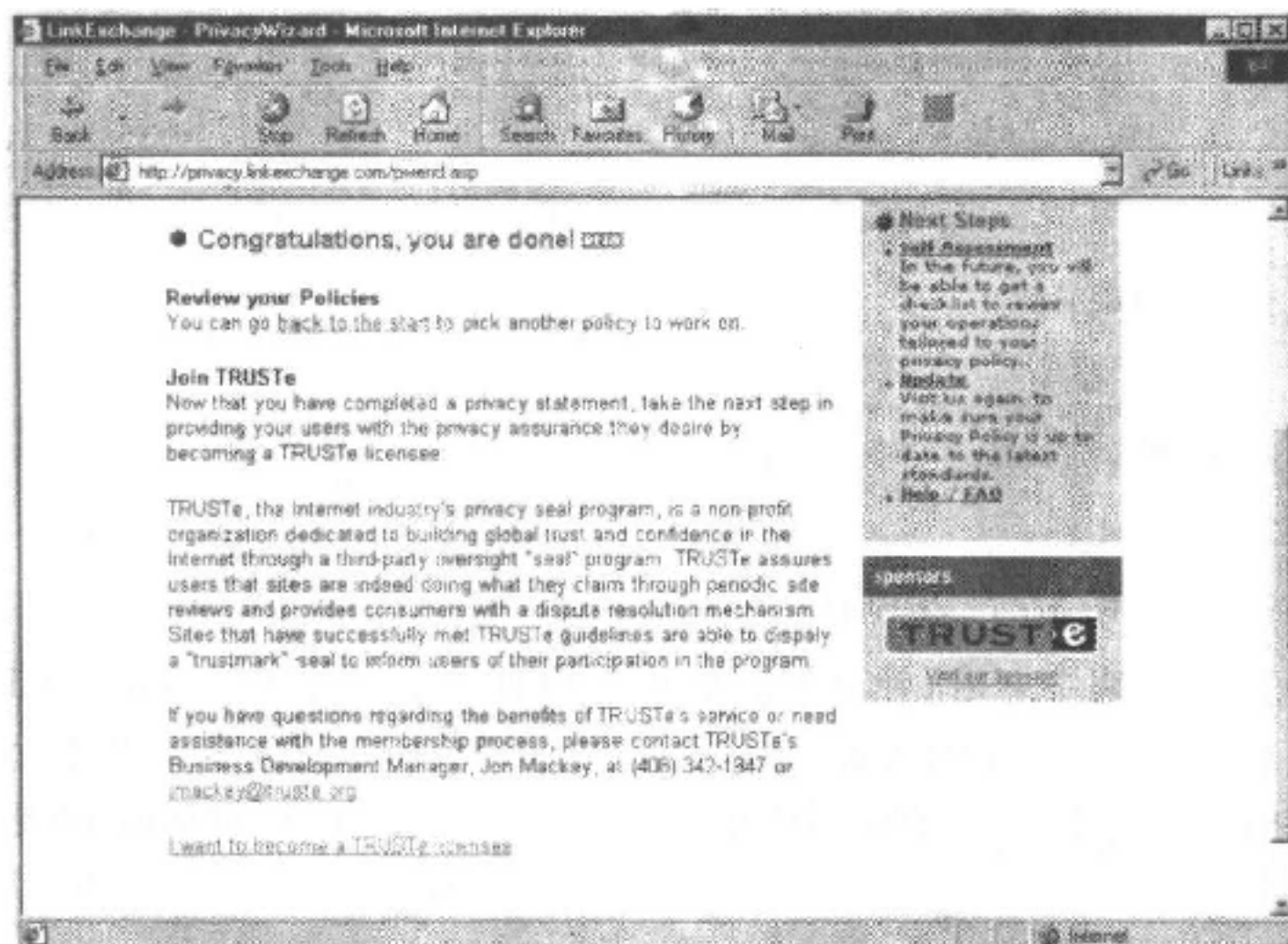


图 37.7 P3P Privacy Wizard 的完成步骤

你可能想知道 Privacy Wizard 生成的最终 XML 代码是什么样子。清单 37.2 包含了 p3policy.xml 文件的完整代码。这是我为我自己的 Web 站点 Tribe 生成的文件。

清单 37.2 使用 Privacy Wizard 生成的用于我自己的 Web 站点 Tribe 的 P3P 文件

```
<? xml version="1.0" ? >
<! -DOCTYPE PROPOSAL SYSTEM http://privacy.linkexchange.com/xml/syntax.dtd-->
<PROPOSAL
  xmlns:VOC=http://privacy.linkexchange.com/xml/vocab.dtd
  xmlns:DATA=http://privacy.linkexchange.com/xml/basedata.dtd
  realm=http://www.thetribe.com
  entity="The Tribe"
>
<USES>
  <STATEMENT VOC:purp="4"
    VOC:recpnt="0"
    VOC:id="1"
  >
    <DATA:REF category="0"/>
```



```

    <DATA:REF category="1"/>
  </STATEMENT>
  <STATEMENT VOC:purp="0 1">
    VOC:recpnt="0 "
    VOC:id="1"
  >
    <DATA:REF category="2"/>
  </STATEMENT>
  <STATEMENT VOD:purp="0"
    VOC:recpnt="0 2"
    VOC:id="1"
  >
    <DATA:REF category="3"/>
    <DATA:REF category="6"/>
  </STATEMENT>
  <STATEMENT VOC:purp="2"
    VOC:recpnt="0"
    VOC:id="0"
  >
    <DATA:REF category="7"/>
  </STATEMENT>
  <STATEMENT VOC:purp="0 1"
    VOC:recpnt="0"
    VOC:id="0"
  >
    <DATA:REF category="4"/>
    <DATA:REF category="5"/>
  </STATEMENT>
  <STATEMENT VOC:purp="0 2"
    VOC:recpnt="0"
    VOC:id="1"
  >
    <DATA:REF category="9"/>
  </STATEMENT>
</USES>
<VOC:DISCLOSURE discURI=http://www.thetribe.com/policy.htm
  access="1" other="0"/>
</PROPOSAL>

```

如你所见,这个代码使用了常见的 P3P 协调词表元素和属性来指定我的 Web 站点的个人隐私策略。图 37.8 显示了 Privacy Wizard 为 Tribe 生成的人们可读的个人隐私表述。

注意:你也许注意到在清单 37.2 中的一些元素和属性与在本章前面所讲的有一些不同。例如,PROP 元素名为 PROPERTY。大概是 Microsoft 大胆的将 P3P 协调词表“改进”成这个样子。

37.6 总 结

随着 Web 的多个领域慢慢的接近规范化,个人隐私策略成为用户普遍感兴趣的部分。没有人喜欢在不确切的知道信息如何被使用的情况下提供自己个人信息。个人隐私策略迫使 Web 站点预先确定它们对个人信息的使用方式,它允许用户对于是否提供个人信息做出明智的决定。基本的前提是 Web 站点提供一个个人隐私策略,用户才会确定将什么信息提

供给站点。XML 支持自动声明个人隐私策略的方法。

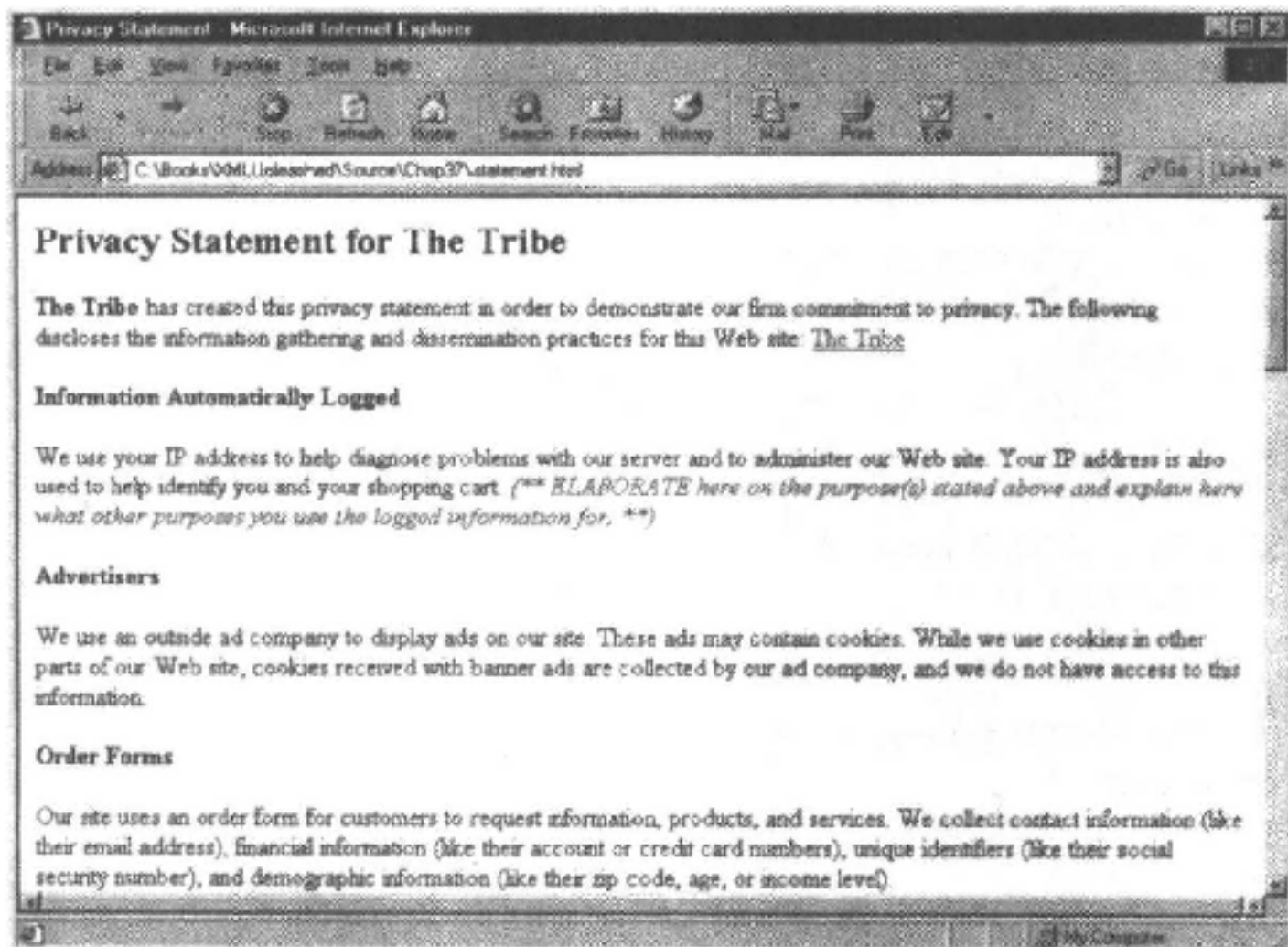


图 37.8 Privacy Wizard 为 Tribe 生成的人们可读的个人隐私表述

本章向你介绍了用来为 Web 站点和资源指定个人隐私策略的 XML 词表。RDF(资源描述框架)是 P3P 词表的基础,这一定并不奇怪,因为 P3P 实际就是元数据。之后我们讲解了 P3P 协调词表,包括如何使用它创建 P3P 文件。最后,我们使用 Privacy Wizard 在线工具生成了一个有着最小限度影响的个人隐私策略。

第 38 章 使用 RELML 列出实际不动产

如果你曾经尝试购买房屋,你一定知道能够估计在实际不动产市场上的什么因素是可用的这一点十分重要。幸运的是,现在已经有了拥有可搜索的实际不动产清单数据库的计算机系统。不幸的是,这些系统的每一个都是只针对一个地区的多重清单服务,极少或是不关注其他地区的服务。这就使得执行对多个地区的大范围搜索事实上成为不可能。

XML 通过构造 RELML(实际不动产清单标记语言—Real Estate Listing Markup Language)提供了对这个问题的解决方案。RELML 是一个丰富的标记语言,它封装了所有在实际不动产清单中需要表述的信息。它实际上是一组适用于诸如住宅、商用、开发地皮等等这样的特殊实际不动产清单类型的 XML 词表。RELML 与住宅有关的词表包含一个 DTD 和一个 XMLData Schema。本章将向你介绍 RELML 以及它所提供的功能。你将明白实际不动产团体的需求以及为什么 RELML 对它的未来如此的重要。之后我们还介绍一些使用 RELML 的软件。

38.1 RELML 基础

你只需访问一下 Realtor.com(<http://www.realtor.com>)就可以理解可搜索的实际不动产清单的重要性。站点的统计显示每个月有六百万用户在 Realtor.com 上搜索房产信息,房产信息每月被浏览 130 遍。这就清楚的说明 Web 是未来实际不动产的搜索方式,起码在购房的初期阶段是如此。令人吃惊的是,实际不动产产业是与实际情况有很大差别的局部化清单服务,它们不能相互联系或共享,对房屋购买者来说这种情况则更为明显。

一些公司已经开始主动为房屋销售清单制定一些标准。HomeSeekers、OpenMLS 和 4th WORLD Telecom 设计发表 RELML 作为实际不动产行业的标记语言选择。RELML 提供了指定所有不动产清单的细节的方法,它包含用于住宅、商用、空地和工地的地产的专门词表。这里还有一个用于住宅地产的 XMLData Schema。

希望 RELML 将允许多个局部 MLS(多重清单服务)供应者用统一的数据格式结合在一起。那么,代理和房屋买主将可以对所有的实际不动产清单进行普遍的访问。特别是考虑到现实的所有 MLS 供应者依赖于不能很好的接入到 Internet 的传统系统,这种情况还只能是一个梦想。然而,HomeSeekers 已经发布了一个名为 MLS2000 的服务,它提供给传统的 MLS 供应者一个转换解决方案。MLS2000 被设计用来通过 RELML 的辅助将现存传统的系统与 Internet 桥接。有多少 MLS 供应者能够从 MLS2000 中获得好处还需要观察,但这确实像是一个将实际不动产信息转换到主流的经济方式。

你也许想知道当它像 Realtor.com 那样拥有对外发布的站点,所有这种公开意味着什么。Realtor.com 已经提供了改进的房产搜索功能。虽然这些站点功能非常的强大,但它们只是房产数据的辽阔海洋中的孤岛。如果没有某种数据的统一格式,对于独立的实际不动产清单站点来说将所有可用的地产清单结合在一起是不可能的。RELML 使得创建在 Web 编目

地产清单上运行的地产搜索引擎成为了可能,这与运行在 Web 编目网页上的 Web 搜索引擎非常的类似。

如果你对于实际不动产清单并不很了解,让我减慢一点速度,阐明一下它们。实际不动产清单由与正在销售的一块地产的所有相关信息组成。下面是一些与实际住宅不动产清单相关的主要信息:

- 清单状态(list status)——有效、未定、卖出等等。
- 清单价格(list price)——价格。
- 住址(address)——街道地址。
- 平米(square footage)——居住面积(平方英尺和平方米)。
- 年龄(age)——房屋的年龄(通常指房屋建造的时间)。
- 占地(lot size)——占地量(亩和公顷)。
- 卧室(bedrooms)——卧室数目。
- 浴室(bathrooms)——浴室数目。
- 汽车间或车库(garage or carport)——房间是否有一个汽车间,一个车库。
- 下水道或化粪池(sewer or septic)——房间是否使用公共下水道或化粪池系统进行废物处理。

所有的这些信息对于正确描述一个待售的住宅来说都是必要的。提供一个有着足够丰富的词表来编写所有的这些信息以及其他我没有提及的信息的词表,是 RELML 的工作。下一部分将让你了解一下所有的可以包含在 RELML 文件中的不同信息。

38.2 深入 RELML 词表

就像你在本章的前面学到的,RELML 进一步分为几个 XML 词表,它们分别针对实际不动产的专门类型。使用不同的词表允许 RELML 不用复杂化并弱化单一的词表就可以处理不同实际不动产类型的细微差别。下面是不同的 RELML 词表:

- Residential
- Commercial
- Vacant land
- Working land

对于本章,我们将集中在 RELML 的住宅(residential)部分。这将是使用 RELML 词表最广泛的领域,仅仅是因为有大批量的住宅待售。清单 38.1 包含了 RELML residential 词表的 DTD。

清单 38.1 RELML residential DTD

```
<! -Parameter Entities -->
<! ENTITY % OTHER "OTHER">
<! ENTITY % NAME "NAME">
<! ENTITY % ACZ "ADDRESS?,CITY?,ZIP? >>
```



```

<! ENTITY % PFW "PHONE?,FAX?,WEB? ">
<! ENTITY % BA "BUILDING-AREA|ADD-INFO">

<! -Notation Declarations -->
<!! NOTATION gif SYSTEM "gview.exe">
<! NOTATION jpeg SYSTEM "jview.exe">
<! NOTATION move SYSTEM "Movie Player">

<! ELEMENT RESIDENTIAL-LISTING (GENERAL, FEATURES, FINANCIAL, REMARKS,
CONTACTS)>
  <! ATTLIST RESIDENTIAL-LISTING VERSION CDATA #FIXED "080698">

  <! ELEMENT GENERAL
    (IMAGE * , APN?, MLS?, TYPE, PRICE, AGE, LOCATION, STRUCTURE, DATES, LAND-
AREA, STATUS,
    (%OTHER;)?, TERMS * )>

  <! ELEMENT IMAGE EMPTY>
  <! ATTLIST IMAGE WIDTH CDATA #REQUIRED
    HEIGHT CDATA #REQUIRED
    SRC CDATA #REQUIRED
    NAME CDATA #IMPLIED
    DESCRIPTION CDATA #IMPLIED>

  <! ELEMENT APN(#PCDATA)>
  <! ATTLIST APN SECURITY (MLS-Only|Restricted|Public) "MLS-Only">

  <! ELEMENT MLS (MLS-CODE,MLS-SOURCE?)>

  <! ELEMENT MLS-CODE(#PCDATA)>
  <! ATTLIST MLS-CODE SECURITY (MLS-Only|Restricted|Public) "MLS-Only">

  <! ELEMENT MLS-SOURCE (%NAME;,%PFW;)>
  <! ATTLIST MLS-SOURCE SECURITY (MLS-Only|Restricted|Public)"MLS-Only">

  <! ELEMENT TYPE(#PCDATA)>

  <! ELEMENT PRICE (#PCDATA)>
  <! ATTLIST PRICE UNITS (PESO|USDOLLAR|CANDOLLAR) "USDOLLAR">

  <! ELEMENT AGE (#PCDATA)>
  <! ATTLIST AGE UNITS (YEARS|MONTHS) "YEARS">

  <! ELEMENT LOCATION (%ACZ; ,ROUGH?)>
  <! ATTLIST LOCATION COUNTRY CDATA #REQUIRED
    STATE CDATA #REQUIRED
    COUNTY CDATA #REQUIRED
    SECURITY (MLS-Only|Restricted|Public) "Public">

  <! ELEMENT ROUGH (#PCDATA)>

  <! ELEMENT STRUCTURE ((NUM-BEDS|NUM-BATHS|SUPER-STRUCTURE| %BA;)* )>

  <! ELEMENT NUM-BEDS (#PCDATA)>
  <! ELEMENT NUM-BATHS(#PCDATA)>
  <! ELEMENT SUPER-STRUCTURE ((NUM-UNITS|NUM-FLOORS| %BA;)* )>
  <! ELEMENT BUILDING-AREA(#PCDATA)>
  <! ATTLIST BUILDING-AREA UNITS (SQ-METRES|SQ-FEET) "SQ-FEET">
  <! ELEMENT NUM-UNITS (#PCDATA)>
  <! ELEMENT NUM-FLOORS (#PCDATA)>

```

```

<! ELEMENT ADD-INFO (#PCDATA)>
<! ELEMENT DATES (LISTING-DATE, LAST-MODIFIED, EXPIRATION-DATE)>
<! ELEMENT LISTING-DATE (#PCDATA)>
<! ELEMENT LAST-MODIFIED (#PCDATA)>
<! ELEMENT EXPIRATION-DATE (#PCDATA)>
<! ELEMENT LAND-AREA (#PCDATA)>
<! ATTLIST LAND-AREA UNITS (HECTARES | ACRES) "ACRES">
<! ELEMENT STATUS (@PCDATA)>
<! ELEMENT TERMS (#PCDATA)>
<! ATTLIST TERMS SECURITY (MLS-Only | Restricted | Public) "MLS-Only">
<! ELEMENT FEATURES (DISCLOSURES, UTILITIES, EXTRAS, CONSTRUCTION, ACCESS,
  (%OTHER; )?)>
<! ELEMENT ASSUMABLE (#PCDATA)>
<! ATTLIST ASSUMABLE SECURITY (MLS-Only | Restricted | Public) "MLS-Only">
<! ELEMENT OWNER-CARRY (#PCDATA)>
<! ATTLIST OWNER-CARRY SECURITY (MLS-Only | Restricted | Public) "MLS-Only">
<! ELEMENT ASSESSMENTS (#PCDATA)>
<! ATTLIST ASSESSMENTS SECURITY (MLS-Only | Restricted | Public) "MLS-Only">
<! ELEMENT DUES (#PCDATA)>
<! ATTLIST DUES SECURITY (MLS-Only | Restricted | Public) "MLS-Only">

<! ELEMENT TAXES (#PCDATA)>
<! ATTLIST TAXES SECURITY (MLS-Only | Restricted | Public) "MLS-Only">

<! ELEMENT LENDER (#PCDATA)>
<! ATTLIST LENDER SECURITY (MLS-Only | Restricted | Public) "MLS-Only">
<! ELEMENT EARNEST (#PCDATA)>
<! ATTLIST EARNEST SECURITY (MLS-Only | Restricted | Public) "MLS-Only">
<! ELEMENT DIRECTIONS (#PCDATA)>
<! ATTLIST DIRECTIONS SECURITY (MLS-Only | Restricted | Public) "MLS-Only">
<! ELEMENT REMARKS (#PCDATA)>
<! ATTLIST REMARKS PROPERTY-REFERENCE CDATA #IMPLIED>
<! ELEMENT CONTACTS (COMPANY?, AGENT+, OWNER?, TENANT?, COMMISSION+)>

<! ELEMENT COMPANY (%NAME;, %ACZ;, %PFW;)>
<! ELEMENT AGENT (%NAME;, %ACZ;, %PFW;)>
<! ELEMENT OWNER (#PCDATA)>
<! ATTLIST OWNER SECURITY (MLS-Only | Restricted | Public) "MLS-Only">
<! ELEMENT TENANT (#PCDATA)>
<! ATTLIST TENANT SECURITY (MLS-Only | Restricted | Public) "MLS-Only">
<! ELEMENT COMMISSION (#PCDATA)>
<! ATTLIST COMMISSION SECURITY (MLS-Only | Restricted | Public) "MLS-Only">
<! ELEMENT RECIPIENT (#PCDATA)>
<! ATTLIST RECIPIENT SECURITY (MLS-Only | Restricted | Public) "MLS-Only">

```

```

<! ELEMENT AMOUNT (#PCDATA)>
<! ATTLIST AMOUNT SECURITY (MLS-Only|Restricted|Public) "MLS-Only">

<! ELEMENT OTHER (#PCDATA)>
<! ATTLIST OTHER SECURITY (MLS-Only|Restricted|Public) "MLS-Only">

<! ELEMENT NAME (#PCDATA)>
<! ELEMENT ADDRESS (#PCDATA)>
<! ATTLIST ADDRESS SECURITY (MLS-Only|Restricted|Public) "Public">

<! ELEMENT CITY (#PCDATA)>
<! ATTLIST CITY SECURITY (MLS-Only|Restricted|Public) "Public">

<! ELEMENT ZIP (#PCDATA)>
<! ATTLIST ZIP SECURITY (MLS-Only|Restricted|Public) "Public">

<! ELEMENT PHONE (#PCDATA)>
<! ELEMENT FAX (#PCDATA)>
<! ELEMENT WEB (E-MAIL,SITE)>
<! ELEMENT E-MAIL (#PCDATA)>
<! ELEMENT SITE (#PCDATA)>

<! ENTITY lt "<">
<! ENTITY gt ">">
<! ENTITY apos "'">
<! ENTITY amp "&">

```

注意:要谨记在 RELML 的 residential 部分还有一个 XML-Data Schema。你可以通过访问位于 4th World Telecom 的 Web 站点 (<http://www.4thworldtele.com/public/design/redesign.html>) 中的网页来浏览这个 Schema。

如你所见, residential 词表非常容易理解。大多数的元素对应我们所熟悉的实际信息。例如 GENERAL 元素包含诸如 TYPE、PRICE、AGE 和 LOCATION 这样的子元素。但虽然 residential DTD 在结构上相当的直接,你仍然不能仅仅通过观察 DTD 就获得 RELML 文件的整体结构。为了获得一个总体的认识,考虑一下下面的这些生成 RELML 文件的主元素:

- RESIDENTIAL_LISTING
- REMARKS
- GENERAL
- CONTACTS
- FEATURES
- OTHER

RESIDENTIAL_LISTING 元素是所有基于 residential 词表的 RELML 文件的文件元素。这意味着一个地产文件中的所有内容必须包含在 RESIDENTIAL_LISTING 元素中。

REMARKS 元素用来加入房地产经纪人的注释,这主要包括住宅格局的简单描述。

GENERAL 元素包含有关地产自身的主要信息。包括 TYPE、PRICE、AGE、

LOCATION 和 STRUCTURE 子元素。STRUCTURE 子元素包含一组描述实际地产细节子元素,包括 NUM-BEDS、NUM-BATHS 和 SUPER-STRUCTURE。

CONTACTS 元素包含所有的联系信息以查明有关地产的更多内容。它包含子元素,有 COMPANY、AGENT、OWNER、TENANT 和 COMMISSION。

FEATURES 元素用来描述有关地产的信息,比如额外部分,设备和地产状况。

最后,OTHER 元素用来提供附加的信息,这通常为特定的 RELML 应用定制。

38.3 创建 RELML 内容

用一个小时好好的学习一下这个 DTD,理解为创建住宅 RELML 内容的最佳途径就是看一个完整的 RELML 文件。清单 38.2 包含了一个基于 residential DTD 的 RELML 文件的 XML 代码。

清单 38.2 住宅实际不动产清单的简单 RELML 文件

```
<? xml version="1.0"? >
<RESIDENTIAL-LISTING VERSION="02241999">
  <REMARKS>Price: 329900 Size (SqFt): 3230 :Everything - This 1-1/2 story
    home features 5 bedrooms, 3 baths, fireplace, over 3,000 sq. ft.
    living area. 2-1/2 acres of landscaping. 157 ft. of sandy West Grand
    Traverse Bay and is only 3 miles from Traverse City. </REMARKS>
  <GENERAL>
    <IMAGE FORMAT="JPEG" WIDTH="150" HEIGHT="150"
      SRC="http://www.nmre.net/search/homes/137343.jpg"/>
  <MLS>
    <MLS-CODE SECURITY="PUBLIC">137343</MLS-CODE>
    <MLS-SOURCE SECURITY="PUBLIC">
      <NAME>REWebcom</NAME>
      <PHONE>616. 933. 3105</PHONE>
      <FAX>616. 486. 1015</FAX>
      <WEB>
        <EMAIL>postmaster@nmre.net</EMAIL>
        <SITE>209. 172. 15. 6</SITE>
      </WEB>
    </MLS-SOURCE>
  </MLS>
  <TYPE>SINGLE-FAMILY</TYPE>
  <PRICE>329900</PRICE>
  <WHEN-BUILT>
    <ORIGINAL-STRUCTURE DATE="1948"/>
    <IMPROVEMENTS DATE="1983"/>
  </WHEN-BUILT>
  <LOCATION COUNTRY="USA" STATE="MI" COUNTY="Township Traverse, Grand
Traverse">
    <ADDRESS>1108 W. Bay Shore Dr. </ADDRESS>
    <CITY>Traverse City</CITY>
    <ZIP>49684</ZIP>
  </LOCATION>
  <STRUCTURE>
```



```

    <NUM-BEDS>Five</NUM-BEDS>
    <NUM-BATHS>Three</NUM-BATHS>
    <BUILDING-AREA UNITS="SQ-FEET">3230</BUILDING-AREA>
  </STRUCTURE>
  <DATES>
    <LISTING-DATE/>
    <EXPIRATION-DATE/>
    <LAST-MODIFIED>7/4/99</LAST-MODIFIED>
  </DATES>
  <LAND-AREA UNITS="ACRES">Parcel</LAND-AREA>
  <STATUS>Active</STATUS>
  <TERMS SECURITY="Public"/>
  <OTHER TITLE="One Up">aXSLTransport.asp? misValue=137343</OTHER>
  <OTHER TITLE="Days On Market"/>
</GENERAL>
<FEATURES/>
<CONTACTS>
  <AGENT>
    <NAME>RON TOUSAIN</NAME>
    <ADDRESS SECURITY="Public"/>
    <CITY SECURITY="Public"/>
    <ZIP SECURITY="Public"/>
    <STATE SECURITY="Public"/>
    <COUNTRY SECURITY="Public"/>
    <PHONE>(616) 946-3263</PHONE>
    <FAX/>
    <WEB>
      <EMAIL>rtousain@rontousain.com</EMAIL>
      <SITE>rontousain.com</SITE>
    </WEB>
    <AGENTIMAGE>http://www.nmre.net/search/agents/rtousain.jpg</AGENTIMAGE>
  </AGENT>
  <COMPANY>
    <NAME>RE/MAX Bayshore Properties, Ltd. </NAME>
    <ADDRESS SECURITY="Public"/>
    <CITY SECURITY="Public"/>
    <ZIP SECURITY="Public"/>
    <STATE SECURITY="Public"/>
    <COUNTRY SECURITY="Public"/>
    <PHONE>(616) 941-4500</PHONE>
    <FAX/>
    <WEB>
      <EMAIL>broker@rewebcom.com</EMAIL>
      <SITE/>
    </WEB>
  </COMPANY>
</CONTACTS>
  <OTHER TITLE="PriceBracket">Price325</OTHER>
  <OTHER TITLE="AreaBracket">Sqft300</OTHER>
  <OTHER TITLE="ListingType">Residential</OTHER>
</RESIDENTIAL-LISTING>

```

这是一个相当大的文件,但如果你一部分一部分的来看,它并不非常的复杂。文件以标记 RESIDENTIAL-LISTING 开始,这是所有的 RELML residential 文件都需要的文件标记。REMARKS 标记被用来提供地产的综述并列出了它最重要的舒适之处。

注意:也许你要知道,我并未手工编写这个 RELML 文件。它实际上是由 OpenMLS 实际不动产清单系统生成,稍后你将会学习这个系统。这并不是说你不能手工创建 RELML 文件。然而,要谨记创建 RELML 文件的典型用户不是 Web 开发人员而是实际不动产代理(或他们的助理)。因此,典型的 RELML 文件创建人将希望使用专门的工具以防止他们陷入编写 XML 的技术细节当中。

GENERAL 标记封装了地产的主要属性,它包括大量的信息。用户愿意看到地产是什么样子。这通过 JPEG 图片和 IMAGE 子元素提供。MLS 子元素定义了有关地产的 MLS 信息,包括地产的惟一 MLS 代号以及管理清单的 MLS 供应者。

本文件包含的其他 GENERAL 标记的子元素还有 TYPE、PRICE、WHEN-BUILT、LOCATION、STRUCTURE、DATES、LAND-AREA 和 STATUS。TYPE 标记指出地产的类型,主要有单元、公寓或多家庭单元(例如双单元)。PRICE 标记指定地产的价格,这里默认的货币单位是美元。WHEN-BUILT 标记是指定最初建造日期和房产改造日期的子元素的容器。LOCATION 标记包含有关地产物理位置的细节(主要是详细的街道地址)。STRUCTURE 标记描述地产的物理结构,包括卧室数、浴室数和总面积。DATES 标记指定清单的日期、清单终止的日期和地产清单的最近改动日期。LAND-AREA 标记指定地产的占地面积,默认的单位是英亩。最后,STATUS 标记指出地产清单的状况,主要的值有有效、未决和售出。

CONTACTS 标记描述了地产的联系信息,它包含 AGENT 和 COMPANY 子元素。AGENT 标记包含有关列出地产的实际不动产代理的信息,包括姓名、地址、电话、传真号码、Web 地址和电子邮件地址。COMPANY 标记包含有关代理的实际不动产公司的信息,包括它的名称、地址、电话等等。

你也许注意到 SECURITY 标记用在这个文件中以表示指定清单信息的安全等级。例如,清单代理的联系信息都通过 SECURITY 标记声明是公用的。你也可以方便的将清单信息的部分声明为 MLS-only,指出只有代理可以访问这些信息。例如有关地产的金融信息将被声明为 MLS-only,这样只有代理能够访问它。代理也能够根据自己的判断将这方面信息提供给买主。

注意:RELML 文件中的安全等级可以更进一步的区分为实际不动产代理和经纪人。如果你不了解实际不动产,我告诉你经纪人是代理的主管。因此,需要提供给经纪人一个单独的安全等级,这样他们可以访问与代理不同的信息。

你所看到的 RELML 文件的例子没有包含 FINANCIAL 标记,它在技术上违反 RELML residential DTD。下面是一个你如何使用 FINANCIAL 标记指定房产清单中有关月租私房房主付款的信息:

```
<FINANCIAL>
```

```
<DUE SECURITY-LEVEL="Public" TITLE="Monthly Dues">100</DUE>
```

</FINANCIAL>

38.4 RELML 和 OpenMLS

OpenMLS 是 OpenMLS Software 出品的基于 Web 的房产管理系统,它是第一个基于 RELML 的商用地产管理系统。这是一个有趣的系统,因为它发布了用于实际不动产清单的普遍存在的数据库,它可以用多种不同的方法访问。OpenMLS 允许地方性的清单服务使用 OpenMLS 数据存储以进行清单管理,它还授予代理从他们的个人 Web 站点引用清单信息的权限。

OpenMLS 作为一个基于 Web 的前端实现,它使用 Microsoft Active Server 页访问一个 SQL Server 后端数据库。当然,RELML 构成地产清单数据的基础。要感谢 RELML,它使得 OpenMLS 使用的特殊数据库并不很重要。只要地产清单信息最终使用的是 RELML 标记,它就可以被买主读取并且由代理访问,并用于实际不动产搜索引擎。

RELML 并不是 OpenMLS 与 XML 的惟一联系。OpenMLS 使用 Active 频道和 CDF(频道定义格式)词表支持频道。频道在 OpenMLS 中扮演重要角色,它提供一种方式,在某种类型的地产开始销售时及时通告买主和代理。例如,你也许对指定地点的一定价格范围内某种规模的房子感兴趣。你可以将这些参数设置为频道的一部分,OpenMLS 将在条件匹配的房产开始销售时自动通知你。它不仅将告知你可能的地产,还在与其他房屋买主的竞争中给你提供帮助。CDF 和 Active 频道在第 33 章“使用 CDF 推出 Web 内容”中进行了详细的介绍。

OpenMLS 的 Web 站点提供了 OpenMLS 应用程序的演示版,以让你感觉一下如何在系统中管理地产清单。在你能够使用 OpenMLS 地产管理系统之前,你必须登陆 OpenMLS 的 Web 站点 <http://www.openmls.com> 并预定 OpenMLS 频道。

在登陆并预定 OpenMLS 频道后,你可能希望进行一个搜索,看看 OpenMLS 如何以 RELML 文件格式返回地产清单数据。图 38.1 显示了 OpenMLS 应用的 Residential Search 页面。

这里有一些与地产清单和 RELML 相关的常用信息。实际上,每一个搜索字段都有一个相应的用来在 RELML 文件中指定相应部分信息的 RELML 元素。了解 XML 解析器是如何用来在 RELML 文件中搜索并找出与搜索标准匹配的项,这件事并不困难。这是使用 RELML 处理实际不动产清单的令人难以置信的强大功能之一。

在图 38.1 中,我输入了一些有关住宅房屋的搜索标准,之后 OpenMLS 显示了一个标准的摘要页面。图 38.2 显示了用于搜索的标准。这个页面主要是用于说明用途的一个信息页面,也就是说它主要是正确向你显示出将被用在搜索中的地产值。

如果你确认了这些搜索标准并发出搜索请求。OpenMLS 就会找出一个匹配地产的清单。图 38.3 列出了匹配搜索标准的地产。

如你所见,OpenMLS 在这次搜索中只返回一个匹配的项。我估计我是一个挑剔的买主!无论如何,OpenMLS 会在表格中提供地产的简明摘要。点击地产图片可以打开有关地产详细信息的页面(如图 38.4)。

Full Property 页面包含有关地产列出部分的详细信息。同时,地产信息使用你所学到的 RELML 元素严格的放置,这是有意义的,因为 OpenMLS 要从 RELML 文件中分离数据。

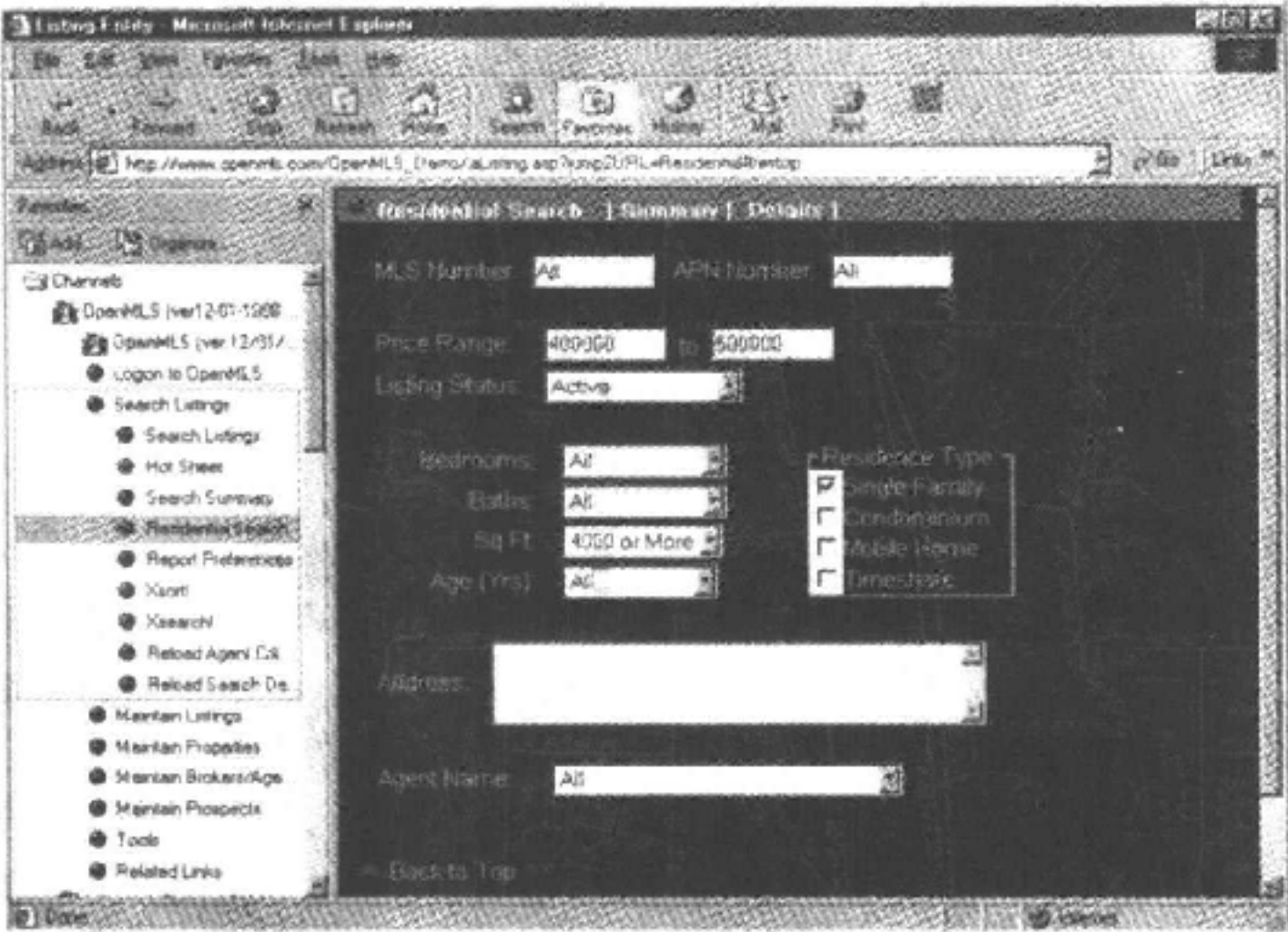


图 38.1 OpenMLS 应用的 Residential Search 页面

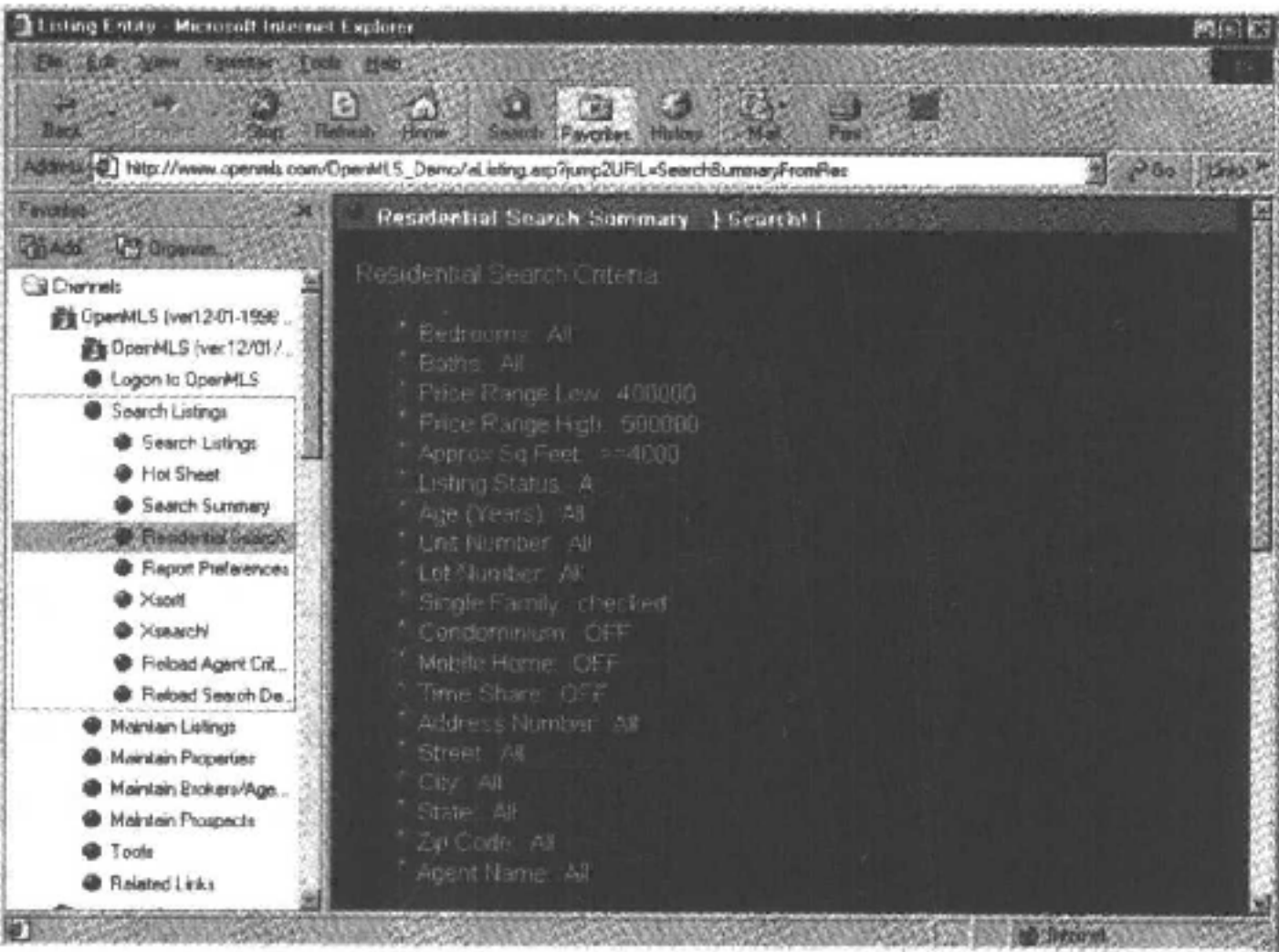


图 38.2 OpenMLS 应用的 Search Criteria 页面

为了满足你的求知欲,清单 38.3 包含了图 38.4 中显示的地产清单的 RELML 代码。

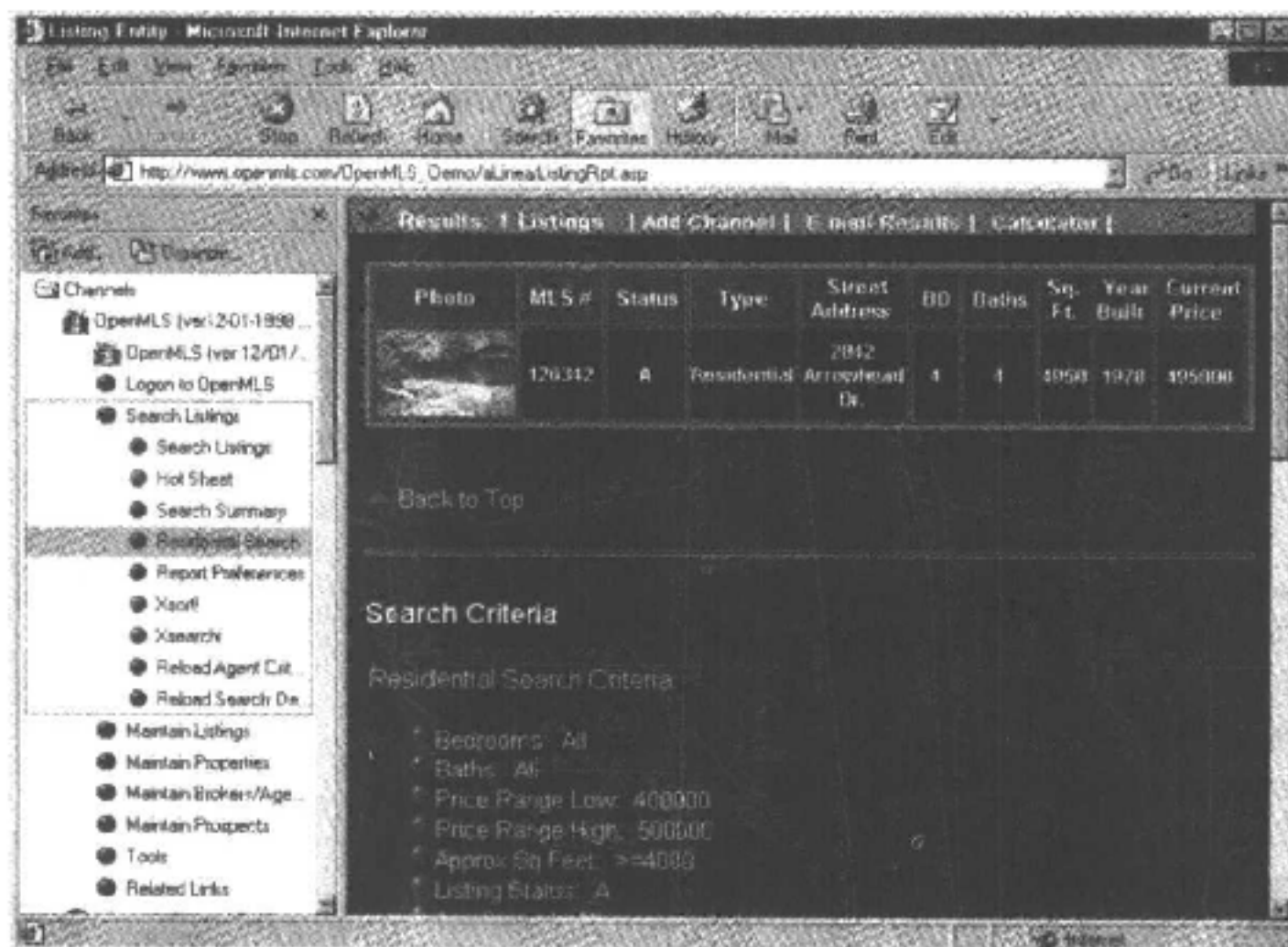


图 38.3 OpenMLS 应用的 Search Results 页面

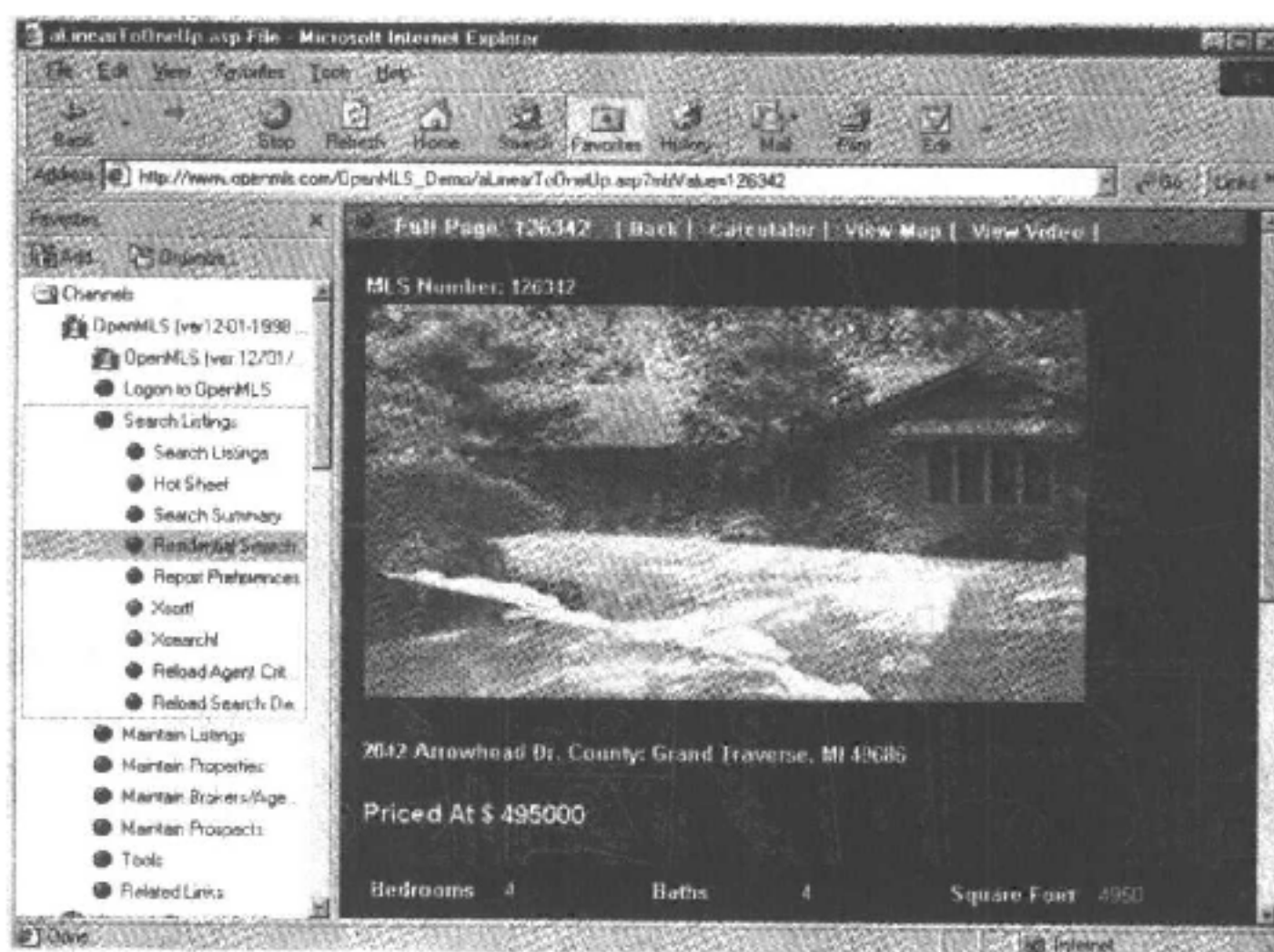


图 38.4 OpenMLS 应用的 Full Property 页面

清单 38.3 图 38.4 显示的住宅实际不动产清单的 RELML 文件

```
<? xml version="1.0" ? >
<RESIDENTIAL-LISTING VERSION="A1">
```

```

<REMARKS>Price:425000 Size (SqFt):4950 :Location, location, beauty, style
and grace--this Huron Hills house, magnificent in redwood, Cherry built ins,
a huge " Cooks Kitchen", vaulted ceilings, huge plush rooms, two decks over
looking East Bay and almost 5000 square feet is, truly one of a kind. A
large master suite with two walk in closets plus a jacuzzi room-on the main
floor. Beyond gorgeous ! </REMARKS>
<GENERAL>
  <IMAGE FORMAT="JPEG" WIDTH="150" HEIGHT="150"
    SRC="http://www.nmre.net/search/homes/126342.jpg"/>
<MLS>
  <MLS-CODE SECURITY="PUBLIC">126342</MLS-CODE>
  <MLS-SOURCE SECURITY="PUBLIC">
    <NAME>REWebcom</NAME>
    <WEB>209.172.15.6</WEB>
    <EMAIL>postmaster@nmre.net</EMAIL>
  </MLS-SOURCE>
</MLS>
<TYPE>SINGLE-FAMILY</TYPE>
<PRICE>425000</PRICE>
<AGE UNITS="YEARS">1978</AGE>
<LOCATION COUNTRY="USA" STATE="MI" COUNTY="Township Garfield, Grand
Traverse">
  <ADDRESS>2042 Arrowhead Dr. </ADDRESS>
  <CITY>Traverse City</CITY>
  <ZIP>49686</ZIP>
</LOCATION>
<STRUCTURE>
  <NUM-BEDS>Four</NUM-BEDS>
  <NUM-BATHS>Four</NUM-BATHS>
  <BUILDING-AREA UNITS="SQ-FEET">4950</BUILDING-AREA>
</STRUCTURE>
<DATES>
  <LISTING-DATE/>
  <EXPIRATION-DATE/>
  <LAST-MODIFIED>7/2/99</LAST-MODIFIED>
</DATES>
<LAND-AREA>Irregular</LAND-AREA>
<STATUS>Inactive</STATUS>
<TERMS SECURITY="Public"/>
<OTHER TITLE="One Up">aXSLTransport.asp? mlsValue=126342</OTHER>
<OTHER TITLE="Days On Market"/>
</GENERAL>
<FEATURES/>
<CONTACTS>
  <AGENT>
    <NAME>JACK LANE</NAME>
    <PHONE>(616) 938-2433</PHONE>
    <WEB>realestateonetc.com/agents/jlane</WEB>
    <EMAIL>jlane@realestateonetc.com</EMAIL>
    <AGENTIMAGE>http://www.nmre.net/search/agents/jlane.jpg</AGENTIMAGE>
  </AGENT>
  <COMPANY>
    <NAME>Real Estate One-TC/Randolph</NAME>
    <PHONE>(231) 946-4040</PHONE>
  </COMPANY>

```

```
<WEB/>
</COMPANY>
</CONTACTS>
<OTHER TITLE="PriceBracket">Price400</OTHER>
<OTHER TITLE="AreaBracket">Sqft400</OTHER>
<OTHER TITLE="ListingType">Residential</OTHER>
</RESIDENTIAL-LISTING>
```

38.5 总 结

RELML(实际不动产清单标记语言)是一个用来描述实际不动产清单的 XML 词表。实际不动产清单非常的局部化并且十分依赖传统系统来管理清单。这就使它难以让代理和卖方从特定场所获得信息。RELML 针对统一实际不动产清单,提供了一个表述清单数据的专一 XML 词表。

本章通过研究 RELML 的 residential 部分的 DTD,很好的展示了 RELML 词表的结构。这个 DTD 之后在一个由 OpenMLS 地产清单系统生成的实际 RELML 文件中使用。最后,你通过学习如何使用 OpenMLS 系统进行基于 RELML 数据的搜索,完成了本章的学习。

第39章 使用 HRMML 管理人才资源

像 Monster(<http://www.monster.com>)和 Hot Jobs(<http://www.hotjobs.com>)这样的 Web 求职站点的流行清楚的说明了 Web 是寻找工作的优秀中介。与其他 Web 上的信息一样,共享工作内容信息的标准格式可以帮助个人发送简历和寻找工作。在另一方面,招聘人员可以使用可以浏览并智能搜索的标准格式列出职位空缺。就如同你所猜想的一样,现在已经有了一个 XML 应用可以应付这样的人才资源信息表述。不幸的是,当前上述的 Web 求职站点还没有一个使用这样的 XML 应用,但如果它们这样做了,将给个人和招聘人员带来同样重要的好处。

HRMML(人才资源管理标记语言)是一个 XML 应用,它包含有对履历和职位的陈述,这就符合了供求双方的需求。本章将向你介绍 HRMML 并让你熟悉 HRMML 词表的履历部分。你将学到如何使用 HRMML 编写履历以及如何使用 XSL 样式表显示履历。

39.1 HRMML 基础

在我们生活的快步前进的社会中,工作是我们生活中最重要的组成部分,不幸的是它还会经常的变更。有些年轻人基本上以两年一次的速度更换工作。因而,职业介绍服务迅速流行起来也就不显得很奇怪了。在这其中,Web 起到了推波助澜的作用。将履历以及空缺职位全面的收集到网上以供所有人浏览的能力已经改变了现有的雇佣途径。然而,每个不同的职业介绍服务有它自己的方法存储履历和工作位置信息。

由 Structured Methods 创建的 HRMML 是一个 XML 应用,它的目标是统一人才资源信息表述和共享的方法。HRMML 使用两个词表来处理人才资源信息的问题,一个用来标记履历,一个用来标记工作岗位。它提供给个人和招聘人员信息标准以完成他们各自的目标。

HRMML 是被设计为通过 XML 对人才资源信息进行表述的非常广泛的规范,要理解这一点非常的重要。它给予用户挑选 HRMML 元素的自由以最好的适应他们的人才资源标记的需求。换句话说,公司可以自由的使用 HRMML 中元素的子集完成所给的任务。只要他们不以任何方式扩展 HRMML,他们的数据就可以与其他的 HRMML 系统相结合。

为了给你一个 HRMML 的应用,让我们看一下位于 <http://www.hotjobs.com> 的 Hot Jobs Web 站点。如果你正在求职的话,你可以使用 Hot Jobs 将你的履历发送给其他人。由于 Hot Jobs 当前并没有使用 HRMML,并且还没有标记履历的其他标准,Hot Jobs 要求你直接通过文本提供你的履历。图 39.1 显示了发送给 Hot Jobs 的一个文本方式的履历(也许你要清楚,这是我为超级英雄蜘蛛人创建的一份假想的履历)。

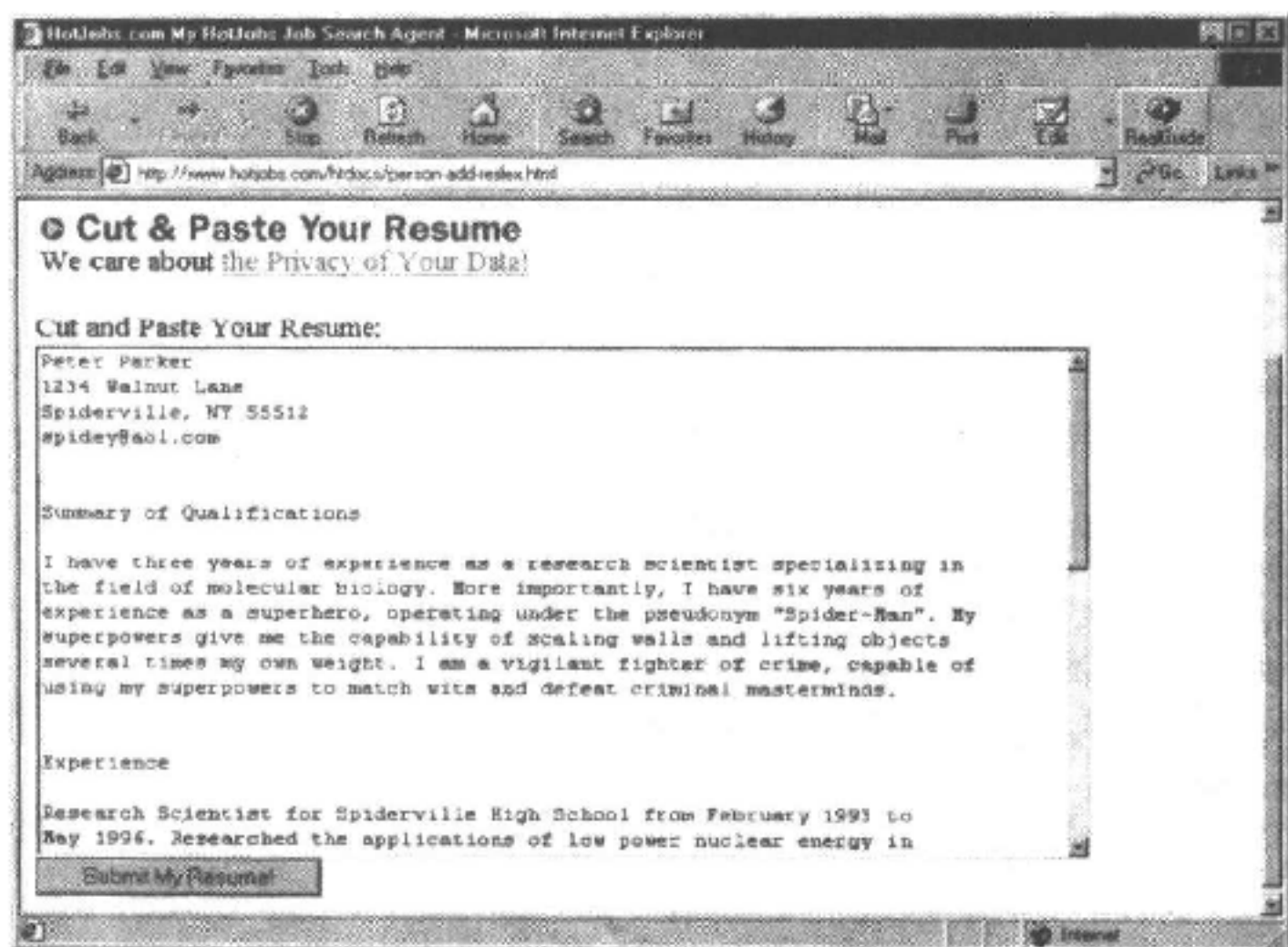


图 39.1 一个使用文本发送到 Hot Jobs Web 站点上的履历

Hot Jobs 有一项专有技术,他可以从履历中提取信息并将它放入到履历数据库记录的相应字段中。这个技术解析履历文本并试图提取普通的履历数据,比如联系信息、工作经验和教育程度。虽然这个履历解析技术有时可以在提取数据上做得很好,但在这里它除了获取 Peter Parker 的联系信息外不能获取其他的更多信息。

这个例子突出说明了对标记履历的标准化方法的需求。如果 Hot Jobs 已经可以处理 HRMML 了,则就不需要履历解析技术,因为所有的履历数据的相关部分已经被明智的标记了。用户——在这里就是 Peter Parker——通过使用 HRMML 制作工具可以更好的得到服务,并且避免了手工编写履历的复杂事务。一旦求职站点开始流行 HRMML,这样的工具就会迅速出现。

到这里,恐怕我已经对 HRMML 作了简单的介绍。不仅求职网站可以重新改版将 HRMML 作为人才资源数据的标准格式,HRMML 对于独立的公司也提供了同样的好处。有人员需求的公司以及专业工作岗位公司都可以从共享人才资源信息的标准化方法中受益匪浅。想像一下,一个公司如果能自动的处理包含元信息的履历,就可以根据需要获得符合要求的申请者,这是一件多么美好的事情。例如,专业教育和技术要求可以通过在一组履历中寻找适当的 HRMML 标记迅速的查找。

39.2 深入 HRMML 词表

前面我提到过 HRMML 词表实际上分为两个词表:履历和工作位置信息。由于这些词表每个都处理许多与人才资源相关的同样的问题,Structured Methods 选择模块化词表,这样它们就可以相互共享公用的元素和实体。更准确的说,Resume 和 Job Posting 的 DTD 非

常的类似,因为它们通过参数实体从其他文件中导入大部分的结构。你将发现 HRMML 词表提供了一个以模块化方法设计 DTD 的有趣的例子。

注意:要获得有关 Structured Methods 和 HRMML 的更多信息,请访问 HRMML 的 Web 站点 <http://www.structuredmethods.com/hrxml>。

清单 39.1 包含了 HRMML Resume DTD 的代码,它存储在文件 `resume.dtd` 中。

清单 39.1 HRMML Resume 文件类型定义(DTD)

```
<! ENTITY % SpecialMod "jobtitle | joblocation | address | voice | fax |
  pager | email | employername | skillsqualif | experiencequalif |
  softwarequalif | prgmlangqualif | educationqualif | licensequalif |
  certificationqualif | equipmntqualif | otherqualif | hardwarequalif ">

<! ENTITY % commonmod SYSTEM "hr-comm.mod">
%commonmod;

<! ENTITY % hiringorgmod SYSTEM "hr-org.mod">
%hiringorgmod;

<! ENTITY % postdetailmod SYSTEM "hr-pstd.mod">
%postdetailmod;

<! ENTITY % jobdescmod SYSTEM "hr-jobd.mod">
%jobdescmod;

<! ENTITY % phonemod SYSTEM "hr-phon.mod">
%phonemod;

<! ENTITY % addressmod SYSTEM "hr-addr.mod">
%addressmod;

<! ENTITY % resumemod SYSTEM "hr-resm.mod">

<! ELEMENT resumedatabase (resume+)>

<! ELEMENT resume (resumprolog?,resumebody)>

<! ATTLIST resume %attrglobal;>
```

如你所见,HRMML Resume DTD 非常的依赖参数实体来导入其结构的大部分。只有两个元素在 DTD 中被直接定义: `resumedatabase` 和 `resume`。这两个元素反映了 Resume DTD 的本质,它用来作为对履历的一种储存库,意味着单一的履历文件可以包含多个履历。比较一下 Resume DTD 和 Job Posting DTD 会十分的有趣,因为它们令人惊讶的一致。

清单 39.2 包含了 HRMML Job Posting DTD 的代码,它存储在文件 `jobpost.dtd` 中。

清单 39.2 HRMML Job Posting 文件类型定义(DTD)

```
<! ENTITY % SpecialMod "jobrefid ">

<! ENTITY % commonmod SYSTEM "hr-comm.mod">
%commonmod;

<! ENTITY % hiringorgmod SYSTEM "hr-org.mod">
%hiringorgmod;

<! ENTITY % postdetailmod SYSTEM "hr-pstd.mod">
```

```

%postdetailmod;
<! ENTITY % jobdescmod SYSTEM "hr-jobd. mod">
%jobdescmod;
<! ENTITY % phonemod SYSTEM "hr-phon. mod">
%phonemod;
The<! ENTITY % addressmod SYSTEM "hr-addr. mod">
%addressmod;
<! ELEMENT jobdatabase (jobposting+)>
<! ELEMENT jobposting (hiringorg,principalemployer?,jobinformation)>
<! ATTLIST jobposting %attrglobal;>

```

也会你会有一点儿惊讶于这个 DTD 与 Resume DTD 是如此的一致。实际上, Job Posting DTD 含有除了 hr-resm. mod 模块以外的所有同样的模块, hr-resm. mod 模块只用在 Resume DTD 之中。与 Resume DTD 类似, Job Posting DTD 作为工作位置的储存库进行构造,这就意味着多个工作位置可以包含在一个单一的工作位置文件中。这一点很明显,因为在 Job Posting DTD 中被直接定义的只有两个元素,就是 jobdatabase 和 jobposting。

你已经了解了 Resume 和 Job Posting DTD 的大体结构,现在让我们看一下实际完成定义 HRMML 词表工作的模块。下面是构成 Resume 和 Job Posting DTD 的模块:

- hr-comm. mod——定义公用的与 HR 信息相关的实体和元素。
- hr-org. mod——定义与职业介绍组织相关的元素。
- hr-pstd. mod——定义投递细节的元素。
- hr-jobd. mod——定义工作描述元素。
- hr-phon. mod——定义联系电话元素。
- hr-addr. mod——定义联系地址元素。
- hr-resm. mod——定义履历元素。

下面的几节将介绍这些 HRMML DTD 模块并解释它们对整个 HRMML 词表的关联性。

39.2.1 hr-comm. mod 模块

清单 39.3 包含了 hr-comm. mod 模块的代码,它定义了公用的与标记人才资源信息相关的实体和元素。

清单 39.3 hr-comm. mod HRMML 模块

```

<! ENTITY % HTMLat1 SYSTEM "HTMLat1x. ent">
%HTMLat1;
<! ENTITY % HTMLsymbol SYSTEM "HTMLsymbolx. ent">
%HTMLspecial;
<! ENTITY % attrface "face (b | i | bi | u | bu | lu) 'b'">
<! ENTITY % attrdist
"distribute (external | internal | restricated) 'external'">

```

```

<! ENTITY % attrglobal "
    name      CDATA      # IMPLIED
    class     CDATA      # IMPLIED
    role      CDATA      # IMPLIED
    style     CDATA      # IMPLIED
    id        ID         # IMPLIED
    %attrdist; ">

<! ENTITY % attrref "refid IDREF # IMPLIED">

<! ENTITY % datemod "startdate | enddate | date">

<! ENTITY % textmod "emph | link | %datemod; | %SpecialMod; | img |
    object">

<! ENTITY % pmod "p | ol | ul | dl | img | object">

<! ELEMENT link (#PCDATA)>
<! ATTLIST link
    mailto CDATA # IMPLIED
    linkend CDATA # IMPLIED
    refid ID # IMPLIED>

<! ELEMENT img EMPTY>
<! ATTLIST img
    %attrglobal;
    src CDATA # REQUIRED
    width CDATA # IMPLIED
    height CDATA # IMPLIED
    alt CDATA # IMPLIED>

<! ELEMENT object EMPTY>
<! ATTLIST object
    %attrglobal;
    src CDATA # REQUIRED
    width CDATA # IMPLIED
    height CDATA # IMPLIED
    alt CDATA # IMPLIED
    mediatype CDATA # IMPLIED
    parm CDATA # IMPLIED>

<! ELEMENT date (month?, day?, year )>
<! ELEMENT mont (#PCDATA )>
<! ELEMENT day (#PCDATA )>
<! ELEMENT year (#PCDATA )>
<! ELEMENT startdate (date )>
<! ELEMENT enddate (date )>

<! ELEMENT name (#PCDATA )>
<! ELEMENT jobtitle (#PCDATA )>

<! ELEMENT emph (#PCDATA)>
<! ATTLIST emph %attrface; %attrglobal;>

<! ELEMENT tbd (comment)?>
<! ATTLIST tbd %attrglobal;>

<! ELEMENT ol (li+)>
<! ATTLIST ol % attrglobal;>

<! ELEMENT ul (li+)>
<! ATTLIST ul %attrglobal;>

```

```

<! ELEMENT li (#PCDATA | %textmod;)* >
<! ATTLIST li %attrglobal;>

<! ELEMENT dl (dt,dd)>
<! ATTLIST dl %attrglobal;>

<! ELEMENT dt (#PCDATA | %textmod;)&>
<! ATTLIST dt %attrglobal;>

<! ELEMENT dd (#PCDATA | %textmod;)* >
<! ATTLIST dd % attrglobal;>

<! ELEMENT p (#PCDATA | %textmod;)* >

```

这个模块中的前三个实体声明用来声明标准 HRML 字符实体。下面是每个实体声明的字符声明的类型：

- HTMLlat1——Latin-1 HTML 字符。
- HTMLsymbol——数学符号、希腊字母和标点符号 HTML 字符。
- HTMLspecial——特殊 HTML 字符。

除了这些实体以外，模块声明了 attrglobal 实体，它包含 HRMML 各词表中许多元素使用的全局属性。你还会注意到这个模块定义了一些元素。下面是这个模块中定义的最重要的元素：

- date
- name
- jobtitle

date 元素用来指定 HRMML 的日期，它由三个子元素组成：month、day 和 year。注意没有特殊的格式和属性用来指定 date 元素和它的子元素。date 部分只是简单的定义为 #PCDATA。name 和 jobtitle 元素用来指定个人的姓名和工作名称。所有的这些元素都被表示为 #PCDATA。

39.2.2 hr-org. mod 模块

hr-org. mod 模块定义了 HRMML 文件中的与职业介绍组织相关的元素。清单 39.4 包含 hr-org. mod 模块的代码。

清单 39.4 hr-org. mod HRMML 模块

```

<! ELEMENT hiringorg (name,website?,bustype?,description?,contact * )>
<! ATTLIST hiringorg
  type (agent | principal | unspecified) 'unspecified'
  %attrdist; %attrglobal;>

<! ELEMENT principalemmployer (name,website?,bustype?,description?,
  contact * )>
<! ATTLIST principalemmployer %attrdist; %attrglobal;>

<! ELEMENT contact (name,jobtitle?,address",phonenumbers,website?,
  email * )>

```

```

<! ATTLIST contact %attrdist,%attrglobal;>
<! ELEMENT sic (#PCDATA) >
<! ATTLIST sic %attrdist,%attrglobal;>
<! ELEMENT bustype (#PCDATA | sic) * >
<! ATTLIST bustype %attrdist,%attrglobal;>
<! ELEMENT description (#PCDATA | %pmod) * >
<! ATTLIST description %attrdist,%attrglobal;>
<! ELEMENT email (#PCDATA) >
<! ATTLIST email %attrdist,%attrglobal;>

```

如清单所示,HRMML 文件中的职业介绍组织使用 hiringorg 元素以及一组子元素指定。虽然大多数的与 hiringorg 联合的子元素通过参数实体定义属性,但典型的来说,它们完全作为内容元素使用。这一点很明显,因为所有的子元素都被定义为包含 #PCDATA。

39.2.3 hr-pstd.mod 模块

hr-pstd.mod 模块定义了 HRMML 文件中的投递细节的元素。清单 39.5 包含 hr-pstd.mod 模块的代码。

清单 39.5 hr-pstd.mod HRMML 模块

```

<! ELEMENT postdetail (postwhere+,postwhen?)>
<! ATTLIST postdetail %attrglobal,%attrdist;>
<! ELEMENT postwhere ((newsgroup | website | mailinglist |
  otherpostvenue) | postwhen) * >
<! ATTLIST postwhere %attrglobal,%attrdist;>
<! ELEMENT newsgroup (#PCDATA | postwhen) * >
<! ATTLIST newsgroup %attrglobal,%attrdist;>
<! ELEMENT mailinglist (#PCDATA | postwhen) * >
<! ATTLIST mailinglist %attrglobal,%attrdist;>
<! ELEMENT website (#PCDATA | postwhen) * >
<! ATTLIST website %attrglobal,%attrdist;>
<! ELEMENT otherpostvenue (#PCDATA | postwhen) * >
<! ATTLIST otherpostvenue %attrglobal,%attrdist;>
<! ELEMENT postwhen (#PCDATA | postwhen) * >
<! ATTLIST postwhen %attrglobal,%attrdist;>
<! ELEMENT poststart (#PCDATA | postwhen) * >
<! ATTLIST poststart %attrglobal,%attrdist;>
<! ELEMENT postexpire (#PCDATA | postwhen) * >
<! ATTLIST postexpire %attrglobal,%attrdist;>

```

这个模块定义了与投递细节相关的元素,它用来描述工作和履历投送的位置。Postdetail 元素是用来描述投递的主要元素。在这个元素中,postwhere 元素用来提供指定履历和工作投递的信息。如果履历和工作描述被投递给多个位置,在 postdetail 元素中就会出现多个 postwhere 元素。

39.2.4 hr-jobd.mod 模块

hr-jobd.mod 模块定义了 HRMML 文件中的工作描述元素。清单 39.6 包含 hr-jobd.mod 模块的代码。

清单 39.6 hr-jobd.mod HRMML 模块

```
<! ELEMENT jobinformation (postdetail?,classifdetail?,jobtitle,
    joblocation?,jobdescription,howtoapply+,eeostatement?)>
<! ATTLIST jobinformation %attrglobal;>

<! ELEMENT jobdescription (comment | jobrefid | jobpurpose |
    essentialfunctions | qualifications | wrkenvironment | preferences |
    compensation | %pmod;)*>
<! ATTLIST jobdescription %attrglobal;>

<! ELEMENT classifdetail (grade | classification | approvedby |
    approveddate)*>
<! ATTLIST classifdetail %attrglobal;%attrdist;>

<! ELEMENT compensation (comment | %pmod;)+>
<! ATTLIST compensation %attrglobal;>

<! ELEMENT howtoapply (comment | %pmod;)+>
<! ATTLIST howtoapply %attrglobal;%attrdist;>

<! ELEMENT grade (comment | %pmod;)+>
<! ATTLIST grade %attrglobal;%attrdist;>

<! ELEMENT eeostatement (comment | %pmod;)+>
<! ATTLIST eeostatement %attrglobal;>

<! ELEMENT jobpurpose (comment | %pmod;)+>
<! ATTLIST jobpurpose %attrglobal;%attrdist;>

<! ELEMENT essentialfunctions (comment | %pmod;)+>
<! ATTLIST essentialfunctions %attrglobal;%attrdist;>

<! ELEMENT wrkenvironment (comment | %pmod;)+>
<! ATTLIST wrkenvironment %attrglobal;%attrdist;>

<! ELEMENT preferences (comment | %pmod;)+>
<! ATTLIST preferences %attrglobal;%attrdist;>

<! ELEMENT approvedby (comment | %pmod;)+>
<! ATTLIST approvedby %attrglobal;%attrdist;>

<! ELEMENT approveddate (comment | %pmod;)+>
<! ATTLIST approveddate %attrglobal;%attrdist;>

<! ELEMENT classification (positiontype?,schedule?,duration?,otstatus?)>
<! ATTLIST classification %attrglobal;%attrdist;>

<! ELEMENT positiontype ((directhire | contract | tdb),comment?)>
<! ATTLIST positiontype %attrglobal;%attrdist;>

<! ELEMENT otstatus ((exempt | nonexempt | tbd),comment?)>
<! ATTLIST otstatus %attrglobal;%attrdist;>

<! ELEMENT schedule ((fulltime | parttime | tbd),comment?)>
<! ATTLIST schedule %attrglobal;%attrdist;>
```

```

<! ELEMENT duration ((temporary | regular | tbd),comment?)>
<! ATTLIST duration %attrglobal;%attrdist;>

<! ELEMENT exempt EMPTY>
<! ATTLIST exempt %attrglobal;%attrdist;>

<! ELEMENT nonexempt EMPTY>
<! ATTLIST nonexempt %attrglobal;%attrdist;>

<! ELEMENT directhire EMPTY>
<! ATTLIST directhire %attrglobal;%attrdist;>

<! ELEMENT contract EMPTY>
<! ATTLIST contract %attrglobal;%attrdist;>

<! ELEMENT fulltime EMPTY>
<! ATTLIST fulltime %attrglobal;%attrdist;>

<! ELEMENT parttime EMPTY>
<! ATTLIST parttime %attrglobal;%attrdist;>

<! ELEMENT hoursperweek (#PCDATA | comment) * >
<! ATTLIST hoursperweek %attrglobal;%attrdist;>

<! ELEMENT temporary (termlength | comment) +>
<! ATTLIST temporary %attrglobal;%attrdist;>

<! ELEMENT termlength (#PCDATA | comment) * >
<! ATTLIST termlength %attrglobal;%attrdist;>

<! ELEMENT regular EMPTY>
<! ATTLIST regular %attrglobal;%attrdist;>

<! ELEMENT qualifications (comment | skillsrequired | experiencerequired |
    softwarerequired | prgmlangrequired | educationrequired |
    licenserequired | certificationrequired | hardwarerequired |
    equipmntrequired | otherrequired) * >
<! ATTLIST qualifications %attrglobal;%attrdist;>

<! ELEMENT hardwarerequired (comment | %pmod;)+>
<! ATTLIST hardwarerequired %attrglobal;%attrdist;>

<! ELEMENT equipmntrequired (comment | %pmod;)+>
<! ATTLIST equipmntrequired %attrglobal;%attrdist;>

<! ELEMENT skillsrequired (comment | skill | %pmod;)+>
<! ATTLIST skillsrequired %attrglobal;%attrdist;>

<! ELEMENT experiencerequired (comment | %pmod;)+>
<! ATTLIST experiencerequired %attrglobal;%attrdist;>

<! ELEMENT softwarerequired (comment | software | %pmod;)+>
<! ATTLIST softwarerequired %attrglobal;%attrdist;>

<! ELEMENT prgmlangrequired (comment | prgmlang | %pmod;)+>
<! ATTLIST prgmlangrequired %attrglobal;%attrdist;>

<! ELEMENT educationrequired (comment | degree | %pmod;)+>
<! ATTLIST educationrequired %attrglobal;%attrdist;>

<! ELEMENT certificationrequired (comment | certification | %pmod;)+>
<! ATTLIST certificationrequired %attrglobal;%attrdist;>

<! ELEMENT licenserequired (comment | licensename | %pmod;)+>
<! ATTLIST licenserequired %attrglobal;%attrdist;>

<! ELEMENT otherrequired (comment | %pmod;)+>

```

```

<! ATTLIST otherrequired %attrglobal;%attrdist;>
<! ELEMENT skill (#PCDATA | yearseqperience | comment) * >
<! ATTLIST skill %attrglobal;%attrdist;>
<! ELEMENT major (#PCDATA | comment) * >
<! ATTLIST major %attrglobal;%attrdist;>
<! ELEMENT software (#PCDATA | yearseqperience | comment) * >
<! ATTLIST software %attrglobal;%attrdist;>
<! ELEMENT prgmlangl (#PCDATA | yearseqperience | comment) * >
<! ATTLIST prgmlang %attrglobal;%attrdist;>
<! ELEMENT degree (#PCDATA | major | comment) * >
<! ATTLIST degree %attrglobal;%attrdist;>
<! ELEMENT certification (#PCDATA | comment) * >
<! ATTLIST certification %attrglobal;%attrdist;>
<! ELEMENT licensename (#PCDATA | comment) * >
<! ATTLIST licensename %attrglobal;%attrdist;>
<! ELEMENT yearsexperience (#PCDATA | comment) * >
<! ATTLIST yearsexperience %attrglobal;%attrdist;>
<! ELEMENT jobrefid (#PCDATA | comment) * >
<! ATTLIST jobrefid %attrglobal;%attrdist;>

```

这个模块是 HRMML 中最大的模块,因为工作描述中有很多的信息。Jobinformation 元素作为提供工作描述的基础。在这个元素中使用 jobtitle、joblocation 和 jobdescription 元素描述一个工作。如果你回想一下,jobtitle 和 joblocation 元素在 hr-comm. mod 模块中被定义。Jobdescription 元素本身包含用来提供有关工作的详细信息的大量子元素。

39.2.5 hr-phon. mod 模块

hr-phon. mod 模块定义了 HRMML 文件中的联系电话元素。清单 39.7 包含了 hr-phon. mod 模块的代码。

清单 39.7 hr-phon. mod HRMML 模块

```

<! ELEMENT phonenumbers ((voice | fax | pager)+)>
<! ATTLIST phonenumbers %attrglobal;>
<! ELEMENT intlcode (#PCDATA)>
<! ATTLIST intlcode %attrglobal;>
<! ELEMENT areacode (#PCDATA)>
<! ATTLIST areacode %attrglobal;>
<! ELEMENT telnumber (#PCDATA)>
<! ATTLIST telnumber %attrglobal;>
<! ELEMENT extension (#PCDATA)>
<! ATTLIST extension %attrglobal;>
<! ENTITY % numbersparts "(intlcode?,areacode?,telnumber,extension?)">
<! ELEMENT voice %numbersparts;>
<! ATTLIST voide %attrglobal;>

```

```

<! ELEMENT fax %numbersparts;>
<! ATTLIST fax % attrglobal;>

<! ELEMENT pager %numbersparts;>
<! ATTLIST pager %attrglobal;>

```

这个 HRMML 模块定义了用来描述履历和工作投递文件中的联系电话信息的元素。phonenumbers 元素作为主要元素,在其中放置了更为详细的电话元素。voice、fax 和 pager 元素是 phonenumbers 元素的子元素,分别描述了电话号码的不同类型。在这些元素的任一个中,attrglobal 实体用来定义电话号码的组成要素。

39.2.6 hr-addr.mod 模块

hr-addr.mod 模块定义了 HRMML 文件中的联系地址元素。清单 39.8 包含了 hr-addr.mod 模块的代码。

清单 39.8 hr-addr.mod HRMML 模块

```

<! ELEMENT joblocation (#PCDATA | comment) * >
<! ATTLIST joblocation % attrglobal;>

<! ELEMENT address (addressline+,city,(state \ province),postalcode,
country?)>
<! ATTLIST address %attrglobal;>

<! ELEMENT addressline (#PCDATA)>
<! ATTLIST addressline % attrglobal;>

<! ELEMENT city (#PCDATA)>
<! ATTLIST city %attrglobal;>

<! ELEMENT state (#PCDATA)>
<! ATTLIST state %attrglobal;>

<! ELEMENT province(#PCDATA)>
<! ATTLIST province %attrglobal;>

<! ELEMENT postalcode (#PCDATA)>
<! ATTLIST postalcode %attrglobal;>

<! ELEMENT country (#PCDATA)>
<! ATTLIST country %attrglobal;>

```

这个模块定义了用来描述履历和工作投递文件中的联系地址信息的元素。address 元素作为基本元素,在其中放置了更为详细的地址元素。addressline、city、state、province、postalcode 和 country 元素是 address 元素的子元素,分别描述地址的不同部分。

39.2.7 hr-resm.mod 模块

hr-resm.mod 模块定义了 HRMML 履历文件中的履历元素。清单 39.9 包含了 hr-resm.mod 模块的代码。

清单 39.9 hr-resm.mod HRMML 模块

```

<! ENTITY % attrResSection " sectype (objective | experience | personal |
  references | education | certifications | licenses | qualifsummary |
  skills | profassociations | unspecified ) 'unspecified' ">

<! ELEMENT skillqualif ( #PCDATA)>
<! ATTLIST skillqualif % attrglobal;>

<! ELEMENT experiencequalif ( #PCDATA)>
<! ATTLIST experiencequalif % attrglobal;>

<! ELEMENT softwarequalif ( #PCDATA)>
<! ATTLIST softwarequalif % attrglobal;>

<! ELEMENT prgmlangqualif ( #PCDATA)>
<! ATTLIST prgmlangqualif % attrglobal;>

<! ELEMENT educationqualif ( #PCDATA)>
<! ATTLIST educationqualif % attrglobal;>

<! ELEMENT licensequalif ( #PCDATA)>
<! ATTLIST licensequalif % attrglobal;>

<! ELEMENT equipmntqualif ( #PCDATA)>
<! ATTLIST equipmntqualif % attrglobal;>

<! ELEMENT certificationqualif ( #PCDATA)>
<! ATTLIST certificationqualif % attrglobal;>

<! ELEMENT otherqualif ( #PCDATA)>
<! ATTLIST otherqualif % attrglobal;>

<! ELEMENT hardwarequalif ( #PCDATA)>
<! ATTLIST hardwarequalif % attrglobal;>

<! ELEMENT employername ( #PCDATA)>
<! ATTLIST employername % attrglobal;>

<! ELEMENT resumeprolog (revisiondate?,availabilitydate?,
  compensationdetail?,postdetail?,distributionrestrications?)?
<! ATTLIST resumeprolog % attrglobal;>

<! ELEMENT revisiondate (date )>
<! ATTLIST revisiondate % attrglobal;>

<! ELEMENT availabilitydate ( #PCDATA | date ) * >
<! ATTLIST availabilitydate % attrglobal;>

<! ENTITY % compMod "salary | rate | benefits">

<! ELEMENT compensationdetail ( %compMod; ) * >
<! ATTLIST compensationdetail % attrgalobal;>

<! ELEMENT salary (current | required) * >
<! ATTLIST salary % attrglobal;>

<! ELEMENT rate (current | required) * >
<! ATTLIST benefits % attrglobal;>

<! ELEMENT current ( #PCDATA)>
<! ATTLIST current % attrglobal;>

<! ELEMENT required ( #PCDATA)>
<! ATTLIST required % attrglobal;>

```

```

<! ELEMENT distributionrestrictions (comment | dontdistributedto) * >
<! ATTLIST distributionrestrictions %attrglobal;>

<! ELEMENT dontdistributedto (#PCDATA | hiringorg) * >
<! ATTLIST dontdistributedto %attrglobal;>

<! ELEMENT resumebody (personaldata,resumesection+)>
<! ATTLIST resumebody %attrglobal;>

<! ELEMENT personaldata (name,(%SpecialMod;)* )>
<! ATTLIST personaldata %attrglobal;>

<! ENTITY % resSectionMod "(sectiontitle?,subtitle?,secbody)">

<! ELEMENT resumesection (%resSectionMod') * >
<! ATTLIST resumesection % attrResSection;%attrglobal;>

<! ELEMENT sectiontitle (#PCDATA | %SpecialMod; | %datemod;)* >
<! ATTLIST sectiontitle % attrglobal;>

<! ELEMENT subtitle (#PCDATA | %SpecialMod;)* >
<! ATTLIST subtitle %attrglobal;>

<! ELEMENT secbody (%pmod; \ resumesection) * >
<! ATTLIST secbody %attrglobal;>

```

这个 HRMML 模块定义了与包含在履历中的详细信息相关的元素。在这个模块中定义的最重要的元素是 resumebody, 它包含 HRMML 履历文件中的大多数内容。如你所见, resumebody 元素包含两个子元素, personaldata 和 resumesection, 这进一步的划分了履历内容的结构。你将在本章稍后的“resumebody 元素”部分学到有关这些元素的更多信息。

39.3 评价 HRMML 履历结构

虽然现在你已经很好的体验了 HRMML Resume DTD, 但研究一个单独基于其 DTD 的 XML 文件的巨大描述还是有一定的困难。因此, 我希望阐明 HRMML 履历文件的主要结构。在本章前面我曾提到过 HRMML 履历文件可以包含多个履历。这解释了为什么 HRMML 履历文件的根文件元素是 resumefirst 元素。

在 resumefirst 元素中, resume 元素用来描述一个个人的履历。Resume 元素是一个容器元素, 它在履历文件中简单的描述个人履历。在 resume 元素中, resumeprolog 和 resumebody 元素将履历分为序言和实体。序言是技术上可选的, 但你一般会希望在大部分的履历中包含它, 因为它包含诸如补偿需求和使用信息这样的有用信息。实体包含履历相关的一般信息, 比如个人信息、工作经验、教育程度和保证人。

为了获取对履历结构的认识, 让我们看一下下面的结构代码, 它在单一的文件中包含了两个履历:

```

<resumefirst>
  <resume>
    <resumeprolog>
    </resumeprolog>
    <resumebody>

```



```

    </resumebody>
  </resume>

  <resume>
    <resumeintrolog>
    </resumeintrolog>

    <resumebody>
    </resumebody>
  </resume>
</resumefield>

```

39.3.1 resumefield 元素

resumefield 用来提供“隐藏”信息，它们主要是在会见过程中从工作候选人中得到的。例如，薪水需求通常在会见过程中说明，而不是显眼的印在履历上。这也就是为什么我把履历序言信息称为“隐藏”——它对于求职很有帮助，但你并不希望在可视的履历中明显的显示出来。下面是用来在履历序言中提供详细信息的子元素：

- revisiondate
- availabilitydate
- compensationdetail
- postdetail

这些元素在下面几节中将详细进行描述。

revisiondate 元素

revisiondate 元素用来指定履历最近修改的日期。revisiondate 元素依靠 date 元素指定修改日期。下面是一个使用 revisiondate 元素指定修改日期的例子：

```

< revisiondate >
  <date><month>November</month><day>30</day><year>1999</year></date>
</ revisiondate >

```

availabilitydate 元素

availabilitydate 元素用来指定工作候选人开始新工作的日期。下面是使用 availabilitydate 元素指定开始日期的例子：

```

< availabilitydate >Two weeks notice</availabilitydate >

```

这个代码揭示了 availabilitydate 和 revisiondate 元素之间的微妙不同。因为开始日期通常用近似值表示（“两周通知”），availabilitydate 元素允许你通过 date 元素或者只作为 #PCDATA 指定日期。在这里，#PCDATA 方法比定死一个指定的日期解决的更好。

compensationdetail 元素

compensationdetail 元素用来指定关系到工作候选人的当前薪水和利益的细节，也就是薪水和利益需求。Compensationdetail 元素依赖 salary 和 benefits 元素描述当前和期望的薪水和利益。在这些元素中，current 和 required 元素用来区别当前和期望的薪水和利益。下面

是一个使用 compensationdetail 元素和相关子元素提供报酬细节的例子：

```
< compensationdetail >
  <salary>
    <required> $ 50,000</required>
    <current> $ 47,500</current>
  </salary>
  <benefits>
    <current>Health, Life, 401(k)</current>
    <required>Would consider contract positions without
      benefits</required>
  </benefits>
</ compensationdetail >
```

postdetail 元素

postdetail 元素用来提供有关履历发送的信息。在 postdetail 元素中, postwhere 元素用来描述指定的发送。如果履历发送到多个位置, 在 postdetail 元素中就会有多个 postwhere 元素。在 postwhere 元素中, website 元素描述发送履历的 Web 站点。postwhen、poststart 和 postexpire 元素也用来提供有关发送定时的细节。为了有助于将这些元素联合使用, 请看一下下面的例子：

```
<postdetail>
  <postwhere>
    <website>www.monster.com</website>
  <postwhen>
    <poststart>
      <data><month>December</month><day>2</day><year>1999</year></
date>
    </poststart>
    <postexpire>
      <data><month>June</month><day>2</day><year>2000</year></date>
    </postexpire>
  </postwhen>
</postwhere>
</postdetail>
```

这段代码描述了一个发向 Monster 工作发送 Web 站点(<http://www.monster.com>)的履历。发送的开始和截止日期通过 poststart 和 postexpire 元素明确的表示出来。

39.3.2 resumebody 元素

就像你在本章前面所学到的, resumebody 元素承载 HRMML 履历的主要内容。一般而言, 你在传统打印格式履历中用到的所有信息都包含在实体中。当然, 这些信息使用 resumebody 的子元素提供。下面是在 resumebody 元素中使用的子元素：

- personaldata
- resumesection

这些元素将在下面几节中进行详细讲解。

personaldata 元素

personaldata 元素用来指定工作候选人的个人信息,包括他或她的名字和联系信息: name、address、email 和多种电话元素(voice、fax 和 pager),用来在 personaldata 元素中指定个人信息。下面是一个使用 personaldata 元素描述个人信息的例子:

```
< personaldata >
  <name>Peter Parker</name>
  <address>
    <addressline>1234 Walnut Lane</addressline>
    <city>Spiderville</city>
    <state>NY</state>
    <postalcode>55512</postalcode>
  </address>
  <email>spidey@aol.com</email>
  <voice>
    <areacode>212</areacode>
    <telnumber>555-1212</telnumber>
  </voice>
</ personaldata >
```

resumesection 元素

与打印履历相似,HRMML 履历分为包含特定类型信息的几个节,诸如工作经验和教育程度。履历实体中的节使用 resumesection 元素表述。履历中的大部分都包含在 resumesection 元素中。resumesection 元素允许你使用 sectype 属性对履历的节分类。下面是 sectype 属性接受的值:

- unspecified(默认)
- qualifsummary
- experience
- education
- reference
- objective
- skills
- certifications
- licenses
- profassociations
- personal

sectype 属性协助将包含在不同节的履历实体进行分类。履历的每一节使用 sectiontitle

和 sectionbody 元素描述。sectiontitle 元素用来描述履历的节,而 sectionbody 元素在属于履历节时,用来提供有关工作候选人的详细信息。要记住 resumesection 元素可以嵌套在 sectionbody 元素中以进一步区分履历的组织结构。下面是一个履历节的例子,它使用 resumesection 元素和相关的子元素描述:

```
< resumesection sectype="qualifsummary">
  <sectiontitle>Summary of Qualifications</sectiontitle>
  <secbody>
    <p>I have three years of experience as a research scientist
    specializing in the field of molecular biology. More
    importantly, I have six year of experience as a superhero,
    operating under the pseudonym "Spider-Man". My superpowers give
    me the capability of scaling walls and lifting objects several
    times my own weight. I an a vigilant fighter of crime, capable
    of using my superpowers to match wits and defeat criminal
    masterminds. </p>
  </secbody>
</ resumesection >
```

如你所见,履历节的实体通过插入标记(<p>)包围下的文本提供。这是描述 HRMML 格式的履历的节部分的标准方法。

39.4 创建 HRMML 履历

现在,你已经看到了有关 HRMML 履历文件的所有部分,但还要把它们装配起来成为完整的履历。大尺寸的例子包含了关于 Peter Parker 的假设履历的内容,Peter Parker 扮演了超级英雄蜘蛛人。清单 39.10 包含了 Peter Parker 的 HRMML 履历的完整文件。

清单 39.10 Peter Parker(蜘蛛人)的 HRMML 履历

```
<? xml version="1.0"? >
<? xml-stylesheet href="Resume.xsl" type="text/xsl"? >
<resumefatabase>
  <resume>
    <resumeprilog>
      <revisiondate>
        <date><month>November</month><day>30</day><year>1999</year>
      </date>
    </revisiondate>
    <availabilitydate>Two weeks notice</availabilitydate>
    <compensationdetail>
      <salary>
        <required>$ 50,000</required>
        <current>$ 47,500</current>
      </salary>
```



```

    <benefits>
      <current>Health, Life, 401(k)</current>
      <required>Would consider contract positions without
        benefits</required>
    </benefits>
  </compensationdetail>
  <postdetail>
    <postwhere>
      <website>www.monster.com</website>
    <postwhen>
      <poststart>
        <date><month>December</month><day>2</day><year>1999
</year>
        </date>
      </poststart>
      <postexpire>
        <date><month>June</month><day>2</day><year>2000</year>
>
        </date>
      </postexpire>
    </postwhen>
  </postwhere>
  <postwhere>
    <website>www.hotjobs.com</website>
    <postwhen>
      <poststart>
        <date><month>December</month><day>2</day><year>1999
</year>
        </date>
      </poststart>
      <postexpire>
        <date><month>June</month><day>2</day><year>2000</year>
>
        </date>
      </postexpire>
    </postwhen>
  </postwhere>
</postdetail>
</resumeprolog>
<resumebody>
  <personaldata>
    <name>Peter Parker</name>
    <address>
      <addressline>1234 Walnut Lane</addressline>
      <city>Spiderville</city>
      <state>NY</state>
      <postalcode>55512</postalcode>
    </address>
    <email>spidey@aol.com</email>
    <voice>
      <areacode>212</areacode>
      <telnumber>555-1212</telnumber>
    </voice>
  </personaldata>
</resumebody>

```

```

</personaldata>
<resumesection sectype="qualifsummary">
  <sectiontitle>Summary of Qualifications</sectiontitle>
  <secbody>
    <p>I have three years of experience as a research scientist
    specializing in the field of molecular biology. More
    importantly, I have six years of experience as a superhero,
    operating under the pseudonym "Spider-Man". My superpowers give
    me the capability of scaling walls and lifting objects several
    times my own weight. I am a vigilant fighter of crime, capable
    of using my superpowers to match wits and defeat criminal
    masterminds. </p>
  </secbody>
</resumesection>
<resumesection sectype="experience">
  <sectiontitle>Experience</sectiontitle>
  <secbody>
    <resumesection sectype="experience">
      <sectiontitle>
        <jobtitle>Research Scientist</jobtitle> for
        <employername>Spiderville High School</employername> from
        <startdate><date><month>February</month>
        <year>1993</year></date></startdate> to
        <enddate><date><month>May</month>
        <year>1996</year></date></enddate>.
      </sectiontitle>
      <secbody>
        <p>Researched the applications of low power nuclear energy
        in controlled environments. </p>
        <ul>
          <li>Formed theories, wrote papers, gathered data, and
          performed experiments. </li>
          <li>Assessed the potential of experimental findings. </li>
          <li>Ensured that safety standards were met. </li>
          <li>Left the position after an accident involving a
          leakage of radiation. </li>
        </ul>
      </secbody>
    </resumesection>
    <resumesection sectype="experience">
      <sectiontitle>
        <jobtitle>Superhero</jobtitle> for the town of Spiderville
        from <startdate><date><month>June</month>
        <year>1996</year></date></startdate> to present.
      </sectiontitle>
      <secbody>
        <p>Use superpowers to fight crime and bring criminal
        masterminds to justice. </p>
        <ul>
          <li>Researched and designed wrist-mounted web spinning
          device that is used to swing across large spaces. </li>
          <li>Use ability to scale walls to perform reconnaissance
          and chase criminal suspects. </li>
        </ul>
      </secbody>
    </resumesection>
  </secbody>
</resumesection>

```

```
<li>Use superhuman strength to move massive objects in  
the pursuit of criminal suspects. </li>  
<li>Cooperate with law enforcement in the pursuit and  
apprehension of criminal suspects. </li>  
</ul>  
</secbody>  
</resumebody>  
</resumebody>  
</resumebody>  
  
<resumebody sectype="education">  
<sectiontitle>Education</sectiontitle>  
<secbody>  
<resumebody sectype="education">  
<sectiontitle>Spiderville High School</sectiontitle>  
<secbody>  
<p><educationqualif>AP Mathematics</educationqualif> ,  
<educationqualif>AP Biology</educationqualif> ,  
<educationqualif>AP English</educationqualif> , graduated  
in 1993</p>  
</secbody>  
</resumebody>  
</secbody>  
</resumebody>  
  
<resumebody sectype="references">  
<sectiontitle>References</sectiontitle>  
<secbody>  
<p>References Available Upon Request</p>  
</secbody>  
</resumebody>  
</resumebody>  
</resumebody>  
</resumebody>
```

你也许有点吃惊于这个文件的长度,但要谨记大的尺寸是由 HRMML 的层次结构造成的。HRMML 比大多数的 XML 词表更多的使用嵌套元素。虽然如此,包含在 HRMML 履历中的信息并不很难解释。你将在本章的前面看到过 Peter Parker 履历的内容,以及一些提供有关他的经验、教育程度和证明人的更详细信息的附加节。

Peter Parker 现在正在使用它的 HRMML 履历来找工作。然而,他没有办法在更常用的条件下浏览履历。似乎 XSL 样式表将可以用来浏览 HRMML 履历。下一节将解决这个非常话题。

39.5 浏览 HRMML 履历

HRMML 的目标是成为标记和表述履历和工作的流行标准。然而在本书编写期间, HRMML 依然是一个非常年轻的技术,还不存在支持它的软件。因此这就带来了一个问题,就是 HRMML 履历很难阅读,并且作为 XML 源代码很难理解。没有任何类型的软件浏览器,这样就没有方法以对大多数人所熟悉的方法来浏览 HRMML 履历。这也就是 XSL 介入

HRMML 描述的插入点。

就像你在第 11 章“创建 XSL 样式表”中学到的,XSL 可以用来将基于内容的 XML 代码转换为基于表述的 HTML 代码。XSL 提供了一个完美的解决方案,用来在没有特殊插件的情况下在网络浏览器上浏览 HRMML 文件。对用户来说,XSL 掩盖了履历是以 HRMML 代码存储的事实。

要谨记 HRMML 包含元素描述履历的那些你并不希望用户可以在简单 HRML 表述中看到的描述部分。因此,XSL 样式表的责任就是表现出 HRMML 履历文件中与在打印式的履历相同的部分。清单 39.11 包含一个完成这个目标的 XSL 样式表。

清单 39.11 将 HRMML 履历以 HTML 显示的 XSL 样式表

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

  <xsl:template match="/">
    <html><head><title>HR XML Example</title></head>
    <body bgcolor="#FFFFFF">
      <table>
        <tr>
          <td width="40">
          </td>
          <td width="580">
            <h style="padding-left: 15px;background-color: #000000;
              color: #ffffff; font-family: Verdana,Arial;
              font-size: 18pt;text-align: center; font-style: italic;
              letter-spacing: 0.5em">Resume</h1>
            <xsl:for-each select="resumetatabase/resume/resumebbody">
              <xsl:apply-templates select="personaldata"/>
              <xsl:apply-templates select="resumesection"/>
            </xsl:for-each>
          </td>
        </tr>
      </table>
    </body>
  </html>
</xsl:template>

  <xsl:template match="personaldata">
    <xsl:apply-templates select="name"/>
    <xsl:apply-templates select="address"/>
    <xsl:apply-templates select="voice"/>
  </xsl:template>

  <xsl:template match="resumesection">
    <xsl:apply-templates select="./sectiontitle"/>
    <xsl:apply-templates select="secbody"/>
  </xsl:template>

  <xsl:template match="sectiontitle">
    <h4><font face="arial, Geneva, lucida sans Unicode, Helvetica">
    <xsl:value-of/></font></h4>
  </xsl:template>
```



```

<xsl:template match="secbody">
  <xsl:apply-templates select=". /secbody/resumesection/sectiontitle" />
  <xsl:apply-templates />
</xsl:template>

<xsl:template match=". /secbody/resumesection/sectiontitle">
  <font face="arial, Geneva, lucida sans Unicode, Helvetica" size="-1">
    <b><xsl:apply-templates /></b></font>
  </xsl:template>

<xsl:template match="p">
  <p><font face="arial, Geneva, lucida sans Unicode, Helvetica"
    size="-1"><xsl:apply-templates /></font></p>
</xsl:template>

<xsl:template match="ul">
  <ul><xsl:apply-templates select="li" /></ul>
</xsl:template>

<xsl:template match="li">
  <li><font face="arial, Geneva, lucida sans Unicode, Helvetica"
    size="-1"><xsl:value-of /></font></li>
</xsl:template>

<xsl:template match="name">
  <h2><font face="arial, Geneva, lucida sans Unicode, Helvetica">
    <xsl:apply-templates /></font></h2>
</xsl:template>

<xsl:template match="address">
  <font face="arial, Geneva, lucida sans Unicode, Helvetica" size="-1">
    <b><i><xsl:apply-templates select="addressline" />
    <xsl:apply-templates select="state" />
    <xsl:apply-templates select="postalcode" /></i></b></font>
  </xsl:template>

  <xsl:template match="voice">
    <font face="arial, Geneva, lucida sans Unicode, Helvetica" size="-1">
      <br /><b><i>(<xsl:apply-templates select="areacode" />)
      <xsl:apply-templates select="telnumber" /></i></b></font>
    </xsl:template>

    <xsl:template match="addressline">
      <xsl:value-of />,
    </xsl:template>

    <xsl:template match="address">
      <font face="arial, Geneva, lucida sans unicode, Helvetica" size="-1">
        <b><i><xsl:apply-templates select="addressline" />
        <xsl:apply-templates select="city" />
        <xsl:apply-templates select="state" />
        <xsl:apply-templates select="postalcode" /></i></b></font>
      </xsl:template>

      <xsl:template match="voice">
        <font face="arial, Geneva, lucida sans Unicode, Helvetica" size="-1">
          <br /><b><i>(<xsl:apply-templates select="areacode" />)
          <xsl:apply-templates select="telnumber" /></i></b></font>
        </xsl:template>

        <xsl:template match="addressline">
          <xsl:value-of />,
        </xsl:template>

```

```
<xsl:template match="state">
  <xsl:value-of/>,
</xsl:template>

<xsl:template match="city">
  <xsl:value-of/>,
</xsl:template>

<xsl:template match="postalcode">
  <xsl:value-of/>
</xsl:template>

<xsl:template match="jobtitle">
  <xsl:value-of/>
</xsl:template>

<xsl:template match="employername">
  <xsl:value-of/>
</xsl:template>

<xsl:template match="month">
  <xsl:value-of/>
</xsl:template>

<xsl:template match="year">
  <xsl:value-of/>
</xsl:template>

<xsl:template match="day">
  <xsl:value-of/>
</xsl:template>

<xsl:template match="startdate">
  <xsl:value-of/>
</xsl:template>

<xsl:template match="enddate">
  <xsl:value-of/>
</xsl:template>

<xsl:template match="skillsqualif">
  <xsl:value-of/>
</xsl:template>

<xsl:template match="experiencequalif">
  <xsl:value-of/>
</xsl:template>

<xsl:template match="educationqualif">
  <xsl:value-of/>
</xsl:template>

<xsl:template match="softwarequalif">
  <xsl:value-of/>
</xsl:template>

<xsl:template match="prgmlangqualif">
  <xsl:value-of/>
</xsl:template>

<xsl:template match="educationqualif">
  <xsl:value-of/>
</xsl:template>

<xsl:template match="licensequalif">
  <xsl:value-of/>
```

```
</xsl:template>
<xsl:template match="certificationqualif">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="otherqualif">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="hardwarequalif">
  <xsl:value-of/>
</xsl:template>
<xsl:template match="employername">
  <xsl:value-of/>
</xsl:template>
```

我将不深入讲解这个样式表的细节,因为你已经在第 10 章和第 11 章中经受过它的折腾了。然而,还是值得指出一些样式表的最终的方面。首先,注意样式表完全的忽略整个包含在 resumeprolog 元素中的履历序言。这是因为履历序言包含有你并不希望在浏览器中可读显示的信息。

在履历实体中, personaldata 和 resumesection 元素被匹配以开始匹配过程。从那里开始,样式表反复匹配元素以从履历实体中得到相关的信息。之后信息根据它在传统打印格式履历中的位置被格式化。图 39.2 显示了 Peter Parker 的履历使用 HRMML XSL 样式表在网络浏览器中的显示情况。

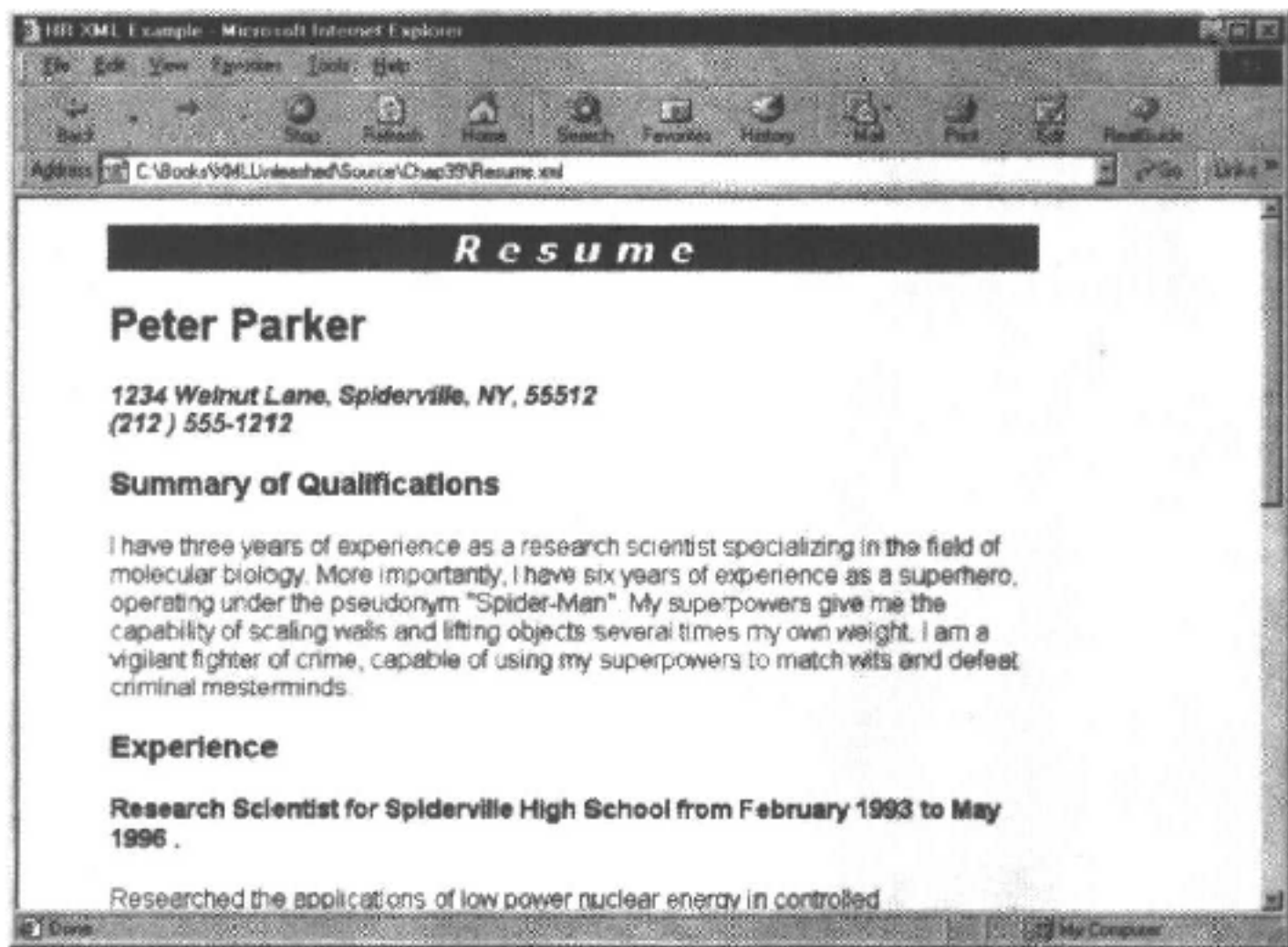


图 39.2 Peter Parker 的履历使用 HRMML XSL 样式表在网络浏览器中的显示情况

39.6 总 结

如果你买这本书学习新的技能以寻找新的工作,你可能会被这一章深深欺骗。HRMML (人力资源管理标记语言)是一个基于 XML 的标准,它表述诸如履历和工作的人才资源信息,它由两个词表组成,这些词表分别针对履历和工作。在本书编写期间,HRMML 依旧是一个相当新的技术,还没有被任何的主流商用 Web 站点采用。然而,使用它能获得很大的好处,因此使用 HRMML 词表编写 XML 的履历这件事可能只是一个时间问题。

第 40 章 交互谈话和 VoxML

如果你曾经打过一个技术支持的电话或尝试追踪你所期望的包裹。你无疑已经涉及声音应答应用,它要求你按 1 键作这个或按 2 键作那个。交互语音应用,或发音应用是声音应答应用的更高级的版本,它解释来自用户的口头请求并从文本生成语音。这两种特性的结合允许你编写代码来完全的驱动一个交互语音应用。

VoxML 是一个基于 XML 的词表,它用来创建交互语音应用。使用 VoxML,你可以定义一系列的用户交互或询问用户信息并提交他的回答的对话。当你考虑一下在从文本生成语音并处理来自用户的说话提交这些过程中实际要包含多少技术时,你就会发现 VoxML 令人惊讶的简单并且功能相当的强大。应该承认,这是处理这些任务的根本技术,而 VoxML 将它们结合在一起。本章将向你介绍 VoxML 并向你展示如何使用 VoxML 创建交互式语音应用。

40.1 VoxML 基础

VoxML 是由 Motorola 创建的 XML 词表,它用来协助交互式语音应用的开发。这是一个有趣的 XML 词表,它用来驱动交互式语音应用,而这些在以前需要包含非常复杂的脚本。有了 VoxML,我们就有可能创建可以询问用户信息并自动提交的强大的语音应用。VoxML 的一个关键的好处是它足够的简单,这就允许了在大多数应用上令人惊讶的短开发周期。由于它是作为 XML 的高度结构化并容易用户化的元语言实现的,有了这个事实,你就有了一个强大的新方法构建语音应用。

在这里,你也许还有一点糊涂就是到底是什么组成了交互式语音应用。交互式语音应用严格意义上就是电话应用,意思是说用户通过使用传统的电话拨入一个系统来运行这些应用。除非你已经做过一些有关语音提交系统的工作,否则你或许并不能立即看出交互语音应用和像 XML 这样的 Web 技术之间的联系。好的,要谨记大多数的交互式语音应用都是通过与存储在计算机上的数据交互的可选途径实现的,通常这是公司的公共 Web 站点或企业内部互联网的一部分。如果被交互语音应用访问的数据存储在 Web 服务器上,使用像 VoxML 这样的 Web 技术来构建应用就显得非常的理想。

经典的 VoxML 语音应用包括一系列的用户交互或步骤。一组步骤结合在一个名为对话的逻辑结构中,它与 VoxML 文件形成一一映射。VoxML 应用可以由多个对话组成,这等同于多个 VoxML 文件。在 VoxML 文件中,你描述用户所说话的文本以及输入选择。用户的口语回答与输入选择相比较,这确定了应用的下一个步骤。VoxML 应用中的例子包括询问用户他的帐号并提供一个选择菜单。步骤可以是任何东西,它组成用户的共享信息和可选回答的处理。

一个考虑 VoxML 的有趣途径就是与 HTML 对照。VoxML 带给交互式语音应用的功能也就是 HTML 带给 Internet 的。这听起来似乎是一个很奇怪的类比,但实际上确实有相

似之处。考虑一下一个包含文本、图片以及用来在页面交互的通常表单的经典网页。VoxML 提供了类似的标记结构来表述包含语音和录音的内容以及通过语音输入实现的交互性。

由于 VoxML 应用是基于电话的,有一点也就不显得奇怪了,这就是它们比一般的基于 Web 的应用需要更多的硬件。附加的硬件主要包括与电话线对接的电话硬件。我们还需要专门的软件来处理电话交互、运行文本到语音转换和识别用户语音输入等等的细节。幸运的是, Motorola 提供了所有的这些软件,其中的一些是由第三方发布的,它们位于 VoxML 的 Web 站点 <http://www.voxml.com>。 Motorola 还提供了名为 VoxML 开发节点的物理测试系统来测试 VoxML 应用,它还提供了两个软件工具来协助创建和测试 VoxML 应用。

40.2 深入 VoxML 词表

你将在本章稍后的“使用 VoxML 创建语音应用”小节中,学习到如何使用 VoxML 不费力气的创建交互语音应用。实际上, VoxML 的真正强大之处在于它的简单性,这可以从 VoxML 词表中明显的看出。清单 40.1 包含了完整的 VoxML DTD。

清单 40.1 VoxML 文件类型定义(DTD)

```
<? xml version="1.0" ? >
<! DOCTYPE dialog [
<! ELEMENT dialog (step|class) * >
<! ATTLIST dialog bargein (Y|N) "Y">
<! ELEMENT step (prompt|input|help|error|cancel|ack) * >
<! ATTLIST step
  name ID #REQUIRED
  parent IDREF #IMPLIED
  cost CDATA #IMPLIED
  bargein (Y|N) "Y">
<! ELEMENT class (prompt|help|error|cancel|ack) * >
<! ATTLIST class
  name ID #REQUIRED
  parent IDREF #IMPLIED
  cost CDATA #IMPLIED
  bargein (Y|N) "Y">
<! ELEMENT prompt (#PCDATA|options|value|emp|break|pros|audio) * >
<! ELEMENT emp (#PCDATA|options|value|emp|break|pros|audio) * >
<! ATTLIST emp level (strong|moderate|none|reduced) "moderate">
<! ELEMENT pros (#PCDATA|options|value|emp|break|pros|audio) * >
<! ATTLIST pros
  rate CDATA #IMPLIED
  vol CDATA #IMPLIED
  pitch CDATA #IMPLIED
  range CDATA #IMPLIED>
<! ELEMENT help (#PCDATA|options|value|emp|break|pros|audio) * >
<! ATTLIST error
  type MNTOKEN "ALL"
```

```

ordinal CDATA #IMPLIED
reprompt (Y|N) "N"
next CDATA #IMPLIED
nextmethod (get|post) "get" >
<! ELEMENT error (#PCDATA|options|value|emp|break|pros|audio) * >
<! ATTLIST error
  type MNTOKEN "ALL"
  ordinal CDATA #IMPLIED
  reprompt (Y|N) "N"
  next CDATA #IMPLIED
  nextmethod (get|post) "get" >
<! ELEMENT cancel (#PCDATA|value|emp|break|pros|audio) * >
<! ATTLIST cancel
  next CDATA #REQUIRED
  nextmethod (get|post) "get" >
<! ELEMENT audio EMPTY>
<! ATTLIST audio src CDATA #REQUIRED>
<! ELEMENT ack (#PCDATA|options|value|emp|break|pros|audio) * >
<! ATTLIST ack confirm NMTOKEN "YORN"
  background (Y|N) "N"
  reprompt (Y|N) "N" >
<! ELEMENT input (option|response|rename|switch|case) * >
<! ATTLIST input
  type (none|optionlist|record|grammar|profile|hidden|yorn|digits|
    number|time|date|money|phone) #REQUIRED
  name NMTOKEN #IMPLIED
  next CDATA #IMPLIED
  nextmethod (get|post) "get"
  timeout CDATA #IMPLIED
  min CDATA #IMPLIED
  max CDATA #IMPLIED
  profname NMTOKEN #IMPLIED
  subtype NMTOKEN #IMPLIED
  src CDATA #IMPLIED
  value CDATA #IMPLIED
  msec CDATA #IMPLIED
  storage (file|request) #IMPLIED
  format CDATA #IMPLIED>
<! ELEMENT switch (case|switch) * >
<! ATTLIST switch field NMTOKEN #REQUIRED>
<! ELEMENT response (switch) * >
<! ATTLIST response
  next CDATA #IMPLIED
  nextmethod (get|post) "get"
  fields CDATA #REQUIRED>
<! ELEMENT rename EMPTY>
<! ATTLIST rename
  varname NMTOKEN #REQUIRED
  recname NMTOKEN #REQUIRED>
<! ELEMENT case EMPTY>
<! ATTLIST case

```

```

    value CDATA #REQUIRED
    next CDATA #REQUIRED
    nextmethod (get|post) "get">
<! ELEMENT value EMPTY>
<! ATTLIST value name NMTOKEN #REQUIRED>
<! ELEMENT break EMPTY>
<! ATTLIST break
    msec CDATA #IMPLIED
    size (none|small|medium|large) "medium">
<! ELEMENT options EMPTY>
<! ELEMENT or EMPTY>
<! ELEMENT option (#PCDATA|value|or)*>
<! ATTLIST option
    value CDATA #IMPLIED
    next CDATA #IMPLIED
    nextmethod (get|post) "get">
]>

```

从其他你在本书中学习到的实际 DTD 的角度来看,这是一个非常简洁的 DTD。这个事实在你考虑到完整功能的 VoxML 文件可以只使用 6 个元素创建时,就会觉得不那么太让人吃惊了。VoxML 词表基于两个基本元素:

- DIALOG
- STEP

每个 VoxML 文件都包含这两个元素。DIALOG 元素在 VoxML 文件中作为承载所有其他元素的容器。STEP 元素用来描述应用中的一个单独的步骤。下面是包含多个步骤的 VoxML 文件的基本框架结构。

```

<DIALOG>
  <STEP>
</STEP>
  <STEP>
</STEP>
  <STEP>
</STEP>
</DIALOG>

```

如你所见,多个步骤在 DIALOG 元素中使用 STEP 元素描述。在下面的几节中,我们将更详细探讨这两个元素以及其他在 VoxML 词表中常用的一些元素。

40.2.1 DIALOG 元素

DIALOG 元素是 VoxML 词表的根文件元素,它作为 VoxML 文件中的所有其他元素的父容器。DIALOG 元素是容器元素,它依赖子元素 STEP 元素来完成描述语音应用中内容的工作。

40.2.2 STEP 元素

STEP 元素用来定义 VoxML 应用中的一个单一状态,它通常由给用户的提示陈述和一些类型的用户定义组成。STEP 元素包含名为 NAME 的属性,这个属性用来分配给这个步骤一个名称。步骤名在所给 VoxML 文件中必须是惟一的。下面是定义一个名为 init 的步骤的例子:

```
<STEP NAME="init">  
</STEP>
```

STEP 元素还支持一些其他的属性,它们并不像 NAME 那么的常用。例如,我们可能使用 COST 属性给步骤赋予成本。成本在 VoxML 应用中出现,以实现根据内容向用户收费。例如,一个娱乐热线也许提供收费的每日股市行情。使用 VoxML 的成本特征有一点超出了本书的范围,但你应该知道 VoxML 支持这样的特征。

每个 VoxML 文件至少需要定义一个名为 init 的 STEP 元素,这个元素用来作为应用中一个对话的开始元素。当应用运行的时候,init 步骤是用户遇到的第一个步骤。这里还有一个名为 end 的步骤用来作为对话的最后步骤。当步骤运行时实际发生的事情是通过 SETP 元素的子元素定义的,最常用的子元素如下:

- PROMPT
- HELP
- INPUT

PROMPT 元素指定当步骤运行时向用户朗读的文本。HELP 元素提供描述步骤的附加帮助文本,它在用户需要帮助时向用户朗读。INPUT 元素用来获得来自用户的输入。在下面的几节中,我们将对这些元素进行更详细的描述。

PROMPT

PROMPT 元素用来定义步骤的实际内容,它由向用户朗读的文本组成,也可以是诸如声音或音乐的录音。从文本到语音软件用来朗读在 PROMPT 元素中定义的文本。VoxML 提供了一些可以在 PROMPT 元素中使用的附加元素来改变文本朗读的方式。例如,你可以使用 BREAK 元素在朗读的文本中插入暂停。下面是一个使用 PROMPT 的例子:

```
<STEP NAME="init">  
  <PROMPT>Welcome to Sparky's Pizza Palace. </PROMPT>  
</STEP>
```

这个例子显示了一个用来欢迎用户的介绍提示。要谨记这个文本将使用从文本到语音软件处理并读给用户。

HELP 元素

HELP 元素用来指定一个提供有关步骤的附加信息的帮助用语。例如,HELP 元素可以向用户阐明所给步骤中可用的选项。HELP 元素中提供的内容在用户需要帮助时提供给用户,这里典型的情况就是用户说出单词“Help”。下面是一个 HELP 元素用来与 PROMPT 元素结合的例子:

```
<STEP name="size">
  <PROMPT>What size pizza would you like? </PROMPT>
  <HELP>Sparky's offers the following pizza sizes: small, medium, and
    large. </HELP>
</STEP>
```

注意 HELP 提供有关步骤的指定选项的信息。如果你不为步骤提供 HELP 元素,默认的用语“No help is available”将读给用户。

INPUT 元素

INPUT 元素用来定义可接受的用户输入响应,以及与它们相关的动作。它支持大量的定义用户响应如何被解释的输入类型。INPUT 元素包括三个主要的属性:TYPE、NAME 和 NEXT。TYPE 属性指定用户所期望的输入类型,这一点稍后我们就会详细学习。NAME 属性指定用来存储用户的输入响应的数据字段的名称。NEXT 属性是可选的,它指定下一个对话以在用户的输入响应被接收后运行。

TYPE 属性允许你从用户处输入不同类型的信息。下面是 TYPE 属性支持的不同类型的输入信息以及它们是如何使用的:

- NONE——不输入任何信息(直接到下一步骤)。
- OPTIONLIST——输入对一个选项列表的选择。
- YORN——输入是/否响应。
- DATE——输入日历日期。
- TIME——输入时间。
- DIGITS——输入一系列的数字。
- NUMBER——输入一个数而不是独立的数字。
- PHONE——输入电话号码。
- MONEY——输入钱数。
- RECORD——输入语音录音。
- GRAMMAR——输入一个语音输入语法。
- HIDDEN——使用户不能交互式的赋值。
- PROFILE——输入用户侧面像(隐藏)。

注意: VoxML 包括一组语音输入的标准,它们定义了诸如数字、日期、时间等等的输入类型。你也可以开发自定义的语音输入语法与标准语法结合使用。

就像你可能从这个输入类型列表中猜测到的一样,VoxML 在得到来自用户的输入信息方面提供了大量的灵活性。INPUT 元素的最简单应用是忽略用户的输入直接转到另一步骤,这是通过 NONE 输入类型实现的。下面是使用 NONE 输入类型的一个例子:

```
<STEP NAME="init">
  <PROMPT>Welcome to Sparky's Pizza Palace</PROMPT>
  <INPUT TYPE="NONE" NEXT="#size"/>
</STEP>
```

在这个例子中,介绍文本被读给用户并且不接受任何输入。此外,INPUT 元素转移到名为 size 的步骤上,这个步骤应该是位于同一个 VoxML 文件中。英镑符号(#)指出 size 确定了同一文件中的一个文件片段。你也完全可以通过在 NEXT 属性中指定文件的 URL 来导向另一个文件。Size 步骤示范了如何使用 OPTIONLIST 输入类型:

```
<TYPE NAME="size">
  <PROMPT>What size pizza would you like? </PROMPT>
  <HELP>Sparky's offers the following pizza sizes:small,medium,and
    large. </HELP>
  <INPUT TYPE="OPTIONLIST" NAME="pizza_size" NEXT="# type">
    <OPTION VALUE="small">small</OPTION>
    <OPTION VALUE="medium">medium</OPTION>
    <OPTION VALUE="large">large</OPTION>
  </INPUT>
</STEP>
```

这段代码显示了如何通过一个可接受响应(选项)列表构造 INPUT 元素。在这个例子中,选项本身并不影响后续的步骤,也就是说下一步将不考虑用户所给的响应。

OPTION 子元素用来定义可以作为用户响应接受的可能选项。OPTION 元素的内容是用来匹配用户响应的文本。VALUE 属性指定在用户提送输入响应后存储到输入数据字段的值。你也可以通过用户的响应改变对话的流程,如同下例所示:

```
<STEP NAME="confirm">
  <PROMPT>Is this order correct? </PROMPT>
  <HELP>Please answer 'yes' or 'no' to confirm or reject your order.
    </HELP>
  <INPUT TYPE="OPTIONLIST" NAME="confirm">
    <OPTION NEXT="# goodbye">yes</OPTION>
    <OPTION NEXT="# size">no</OPTION>
  </INPUT>
</STEP>
```

在这个例子中,OPTION 元素中的 NEXT 属性用来指定根据不同的用户响应导向的不同步骤。注意 NEXT 属性并不在 INPUT 属性中使用,因为导向下一步骤是根据输入选项决定的。

VALUE 元素

你已经看到了 OPTION 元素的 VALUE 属性,它用来指定用户响应选项的值。VoxML 还定义了一个用来引用输入数据字段的值的 VALUE 元素。更重要的是,VALUE 元素用来读取数据字段的值给用户。VALUE 元素主要用在 PROMPT 元素中,如下例所示:

```
<STEP NAME="summary">
  <PROMPT> You have ordered a <VALUE NAME="pizza_size"/>
    <VALUE NAME="pizza_type"/>pizza with a <VALUE NAME="pizza_crust"/>
    crust. </PROMPT>
```

```
<INPUT TYPE="NONE" NEXT="#confirm"/>
</STEP>
```

这个例子使用 VALUE 元素提供了用户所作选择的概要来确认。在这个例子中的 VALUE 元素使用的数据字段名称必须与出现在应用中的这个步骤前面的 INPUT 元素名称相符。否则,字段将不包含有效的信息。

注意:就像你也许已经猜到的那样,在遍历 VoxML 词表时你所见到的例子 VoxML 代码实际上是订购比萨的 VoxML 应用的完整例子的一部分。你将在本章的后面更详细的学到这个例子应用。

40.3 VoxML 工具

Motorola 提供了一些工具来协助 VoxML 应用的创建和测试。这些 VoxML 工具都不是作为制作工具设计的,也就是说你需要使用你所选择的文本或 XML 编辑器来自己编写 VoxML 文件。VoxML 工具主要用来分析 VoxML 文件的结构并进而检测完整的 VoxML 应用。下面是 Motorola 提供的标准 VoxML 工具:

- VoxML 模拟器
- VoxML 浏览器
- VoxML 开发节点

你将在下面的几节中逐个学习到这些工具。

40.3.1 VoxML 模拟器

VoxML 应用需要一个有着大量底层硬件和软件的环境,因此能够在简单的基础下对应用进行测试是非常重要的。VoxML 模拟器提供了这样的应用基础,它可以模拟用户和 VoxML 系统之间的交互。当你使用 VoxML 模拟器时,你通过响应模拟器生成的请求扮演着 VoxML 应用中的用户的角色。

VoxML 模拟器使用 Microsoft 代理技术代表 VoxML 对话的应用方。VoxML 代理使 VoxML 模拟器可以显示一个向用户讲话的动画卡通人物。你可以通过键入或是从下拉菜单中选择以进行响应。VoxML 应用的完整对话在你(用户)和 Microsoft 代理人物之间展开。标准的 VoxML SDK 1.1 附带了人物 Merlin。

注意:要获得有关 Microsoft 代理的更多信息,请访问 Microsoft 代理的 Web 站点 <http://msdn.microsoft.com/workshop.imedia/agent>。

VoxML 模拟器是测试 VoxML 应用的优秀方法,它在指出对话中的问题方面非常的有效。它提供了大量的调试输出选项,可以协助你处理问题。

40.3.2 VoxML 浏览器

VoxML 浏览器用来浏览 VoxML 文件的结构。为了对 VoxML 文件进行图形化的显示,VoxML 浏览器必须处理文件并执行它对结构的确认。换句话说,VoxML 浏览器扮演着

两个角色：

- 它验证 VoxML 文件。
- 它提供对 VoxML 文件的图形化浏览。

虽然对 VoxML 文件进行图形化显示有时是非常有帮助的,但我发现 VoxML 浏览器在验证文件方面更加的有用。

40.3.3 VoxML 开发节点

最后一个 Motorola 提供的用来开发 VoxML 应用的 VoxML 工具就是 VoxML 开发节点,从正规的角度上看它并不是一个实际意义上的软件工具。VoxML 开发节点是一个基于在实际 VoxML 系统基础上应用的类似技术的物理测试环境。下面的部件组成了 VoxML 开发节点,它由 Motorola 提供:

- 语音浏览器
- 语音识别硬件和软件
- 从文本到语音的硬件和软件
- 发音器
- 语音输入语法
- 电话接口(硬件和软件)

应该承认,这些部件有一部分包含在 VoxML SDK 1.1 中,但那些并不足以在实际电话环境下测试 VoxML 应用。VoxML 开发节点提供了一个环境,在其中你可以通过与用户使用相同的电话来全面的测试 VoxML 应用。使用 VoxML 开发节点测试应用是应用开发的最后阶段。你必须使用 VoxML 开发节点将 VoxML 技术支持与测试安排相结合。请访问 VoxML 的 Web 站点 <http://www.voxml.com> 以获取更多的信息。

40.4 使用 VoxML 创建语音应用

现在你已经了解了 VoxML 词表以及用来测试 VoxML 应用的工具,你已经做好准备研究一个完整的 VoxML 应用了。我想一个有趣的 VoxML 应用就是比萨订购系统。大多数的比萨订单是通过电话接到的,因此为什么不使用 VoxML 来自动化这个过程?

在本章前面探讨 VoxML 词表的时候,你已经看到了比萨订购系统的部分代码。清单 40.2 包含了 Sparky 的 Pizza Palace VoxML 应用的完整 VoxML 代码。

清单 40.2 Sparky 的比萨宫交互语音应用的 VoxML 代码

```
<? xml version="1.0"? >
<DIALOG>
  <STEP NAME="init">
    <PROMPT>Welcome to Sparky's Pizza Palace. </PROMPT>
    <INPUT TYPE="NONE" NEXT="#size"/>
  </STEP>
```

```

<STEP NAME="size">
  <PROMPT>What size pizza would you like? </PROMPT>
  <HELP>Sparky's offers the following pizza sizes: small, medium, and
    large. </HELP>
  <INPUT TYPE="OPTIONLIST" NAME="pizza-size" NEXT="# type">
    <OPTION VALUE="small">small</OPTION>
    <OPTION VALUE="medium">medium</OPTION>
    <OPTION VALUE="large">large</OPTION>
  </INPUT>
</STEP>

<STEP NAME="type">
  <PROMPT>What type of pizza would you like? </PROMPT>
  <HELP>Sparky's offers the following types of pizzas: cheese, veggie,
    meaty, and supreme. </HELP>
  <INPUT TYPE="OPTIONLIST" NAME="pizza-type" NEXT="# crust">
    <OPTION VALUE="cheese">cheese</OPTION>
    <OPTION VALUE="veggie">veggie</OPTION>
    <OPTION VALUE="meaty">meaty</OPTION>
    <OPTION VALUE="supreme">supreme</OPTION>
  </INPUT>
</STEP>

<STEP NAME="crust">
  <PROMPT>What type of crust would you like on your pizza? </PROMPT>
  <HELP>Sparky's offers the following pizza crusts: thin and
    pan. </HELP>
  <INPUT TYPE="OPTIONLIST" NAME="pizza-crust" NEXT="# summary">
    <OPTION VALUE="thin">thin</OPTION>
    <OPTION VALUE="pan">pan</OPTION>
  </INPUT>
</STEP>

<STEP NAME="summary">
  <PROMPT>You have ordered a <VALUE NAME="pizza-size" />
    <VALUE NAME="pizza-type" /> pizza with a <VALUE NAME="pizza-crust" />
    crust. </PROMPT>
  <INPUT TYPE="NONE" NEXT="# confirm" />
</STEP>

<STEP NAME="confirm">
  <PROMPT>Is this order correct? </PROMPT>
  <HELP>Please answer 'yes' or 'no' to confirm or reject your order.
    </HELP>
  <INPUT TYPE="OPTIONLIST" NAME="confirm">
    <OPTION NEXT="# goodbye">yes</OPTION>
    <OPTION NEXT="# size">no</OPTION>
  </INPUT>
</STEP>

<STEP NAME="goodbye">
  <PROMPT>Your pizza will be ready in about 30 minutes. Thanks for your
    order. </PROMPT>
  <INPUT TYPE="NONE" NEXT="# end" />
</STEP>
</DIALOG>

```

这个 VoxML 设置了一个对话来询问用户他所希望订购的比萨的相关信息。不可否认,这个例子在某种程度上是简单的,因为它没有询问用户的姓名和电话号码,它也没有提供比萨花费的总计。这些问题当然可以使用 VoxML 代码处理;我只是不希望使例子过分的复杂。此外,像这样的智能系统将可以通过 Caller ID 探测到用户的电话号码并从数据库中获取用户的信息。

回到 VoxML 代码上来,注意这个应用包含一系列的步骤,每个步骤向用户提供信息。一些步骤是纯粹的提供信息,而不考虑用户的响应,而其他的步骤要求用户提供有关他的比萨订单的回答。总体上,这是一个如何装配 VoxML 应用的非常实际的例子。图 40.1 显示了这个文件在 VoxML 浏览器中打开的情况,它提供了对 VoxML 代码的图形化显示。

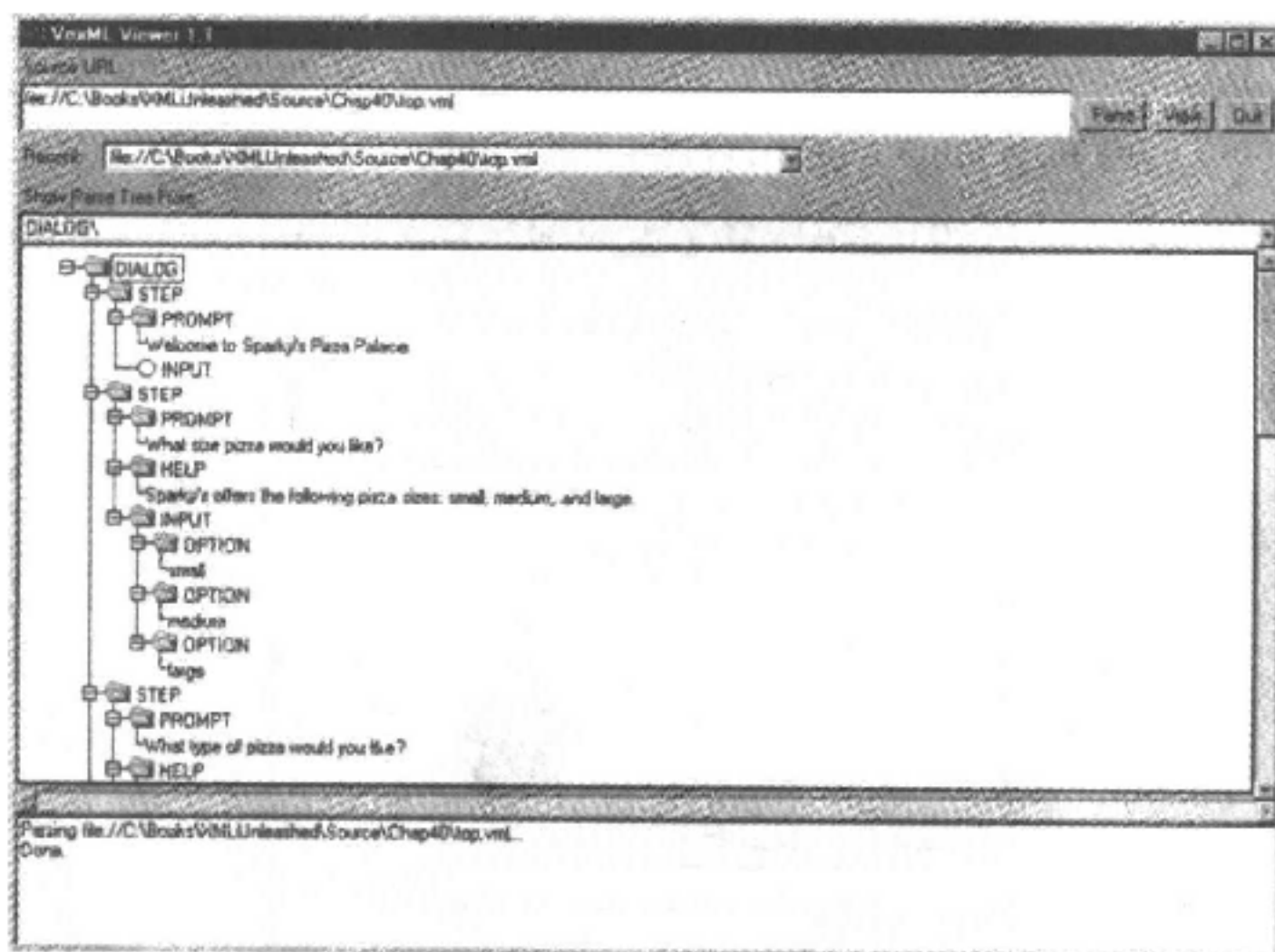


图 40.1 比萨 VoxML 应用在 VoxML 浏览器中的显示情况

为了更好的了解 Sparky 的 Pizza Palace 应用是如何工作的,我们可以使用 VoxML 模拟器来试验一下。图 40.2 显示了这个应用在 VoxML 模拟器中显示时的第一步。本图显示了 Microsoft 代理人物 Merlin 是如何向用户提供对话的应用方的。

图 40.3 显示了应用的第一个用户输入响应,这是比萨的尺寸。

在用户从下拉列表中选择“medium”后,引用向用户询问希望要什么类型的比萨(如图 40.4)。

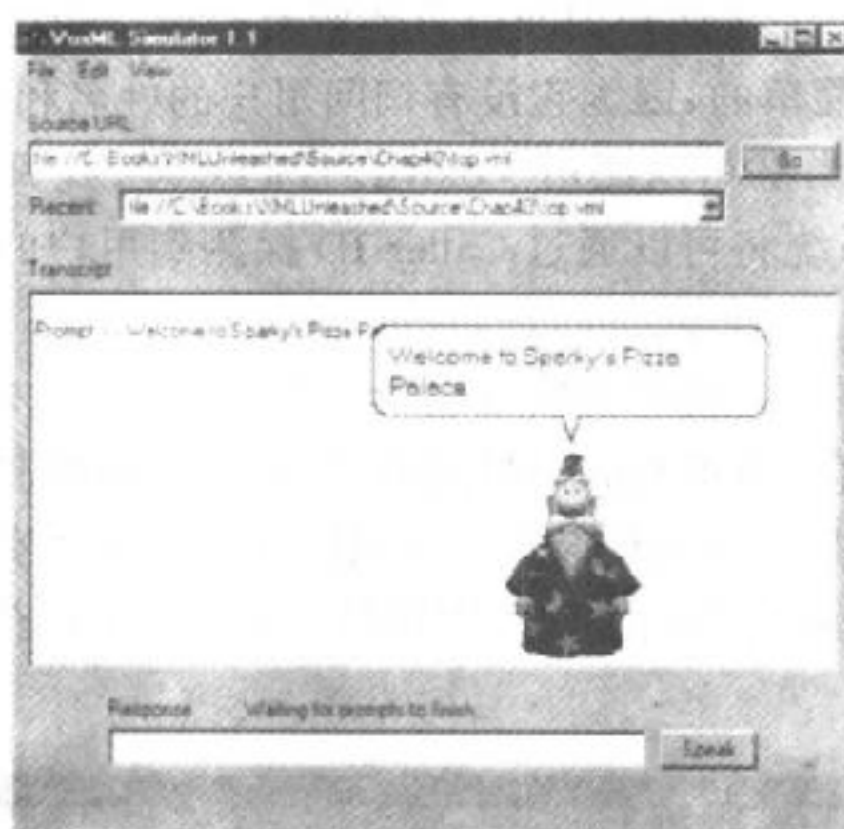


图 40.2 比萨 VoxML 应用在 VoxML 模拟器中的第一步

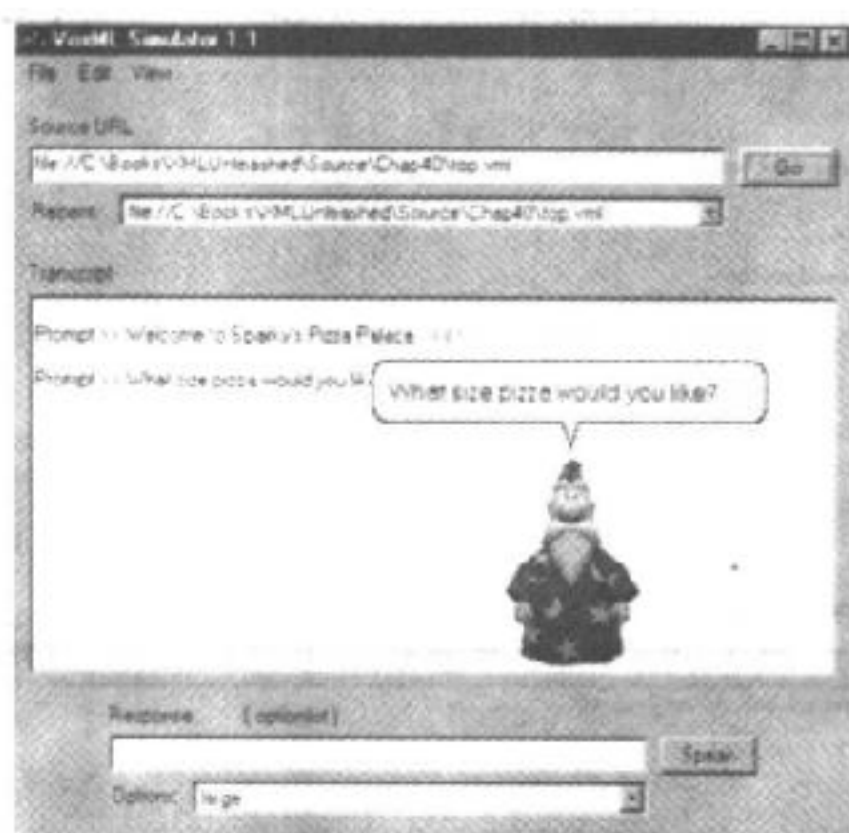


图 40.3 比萨 VoxML 应用要求用户提供比萨的尺寸



图 40.4 比萨 VoxML 应用要求用户提供比萨的类型

如果用户不知道有什么类型的比萨,他可以说单词“Help”来获得比萨类型请求的帮助。图 40.5 显示了应用是如何响应的。

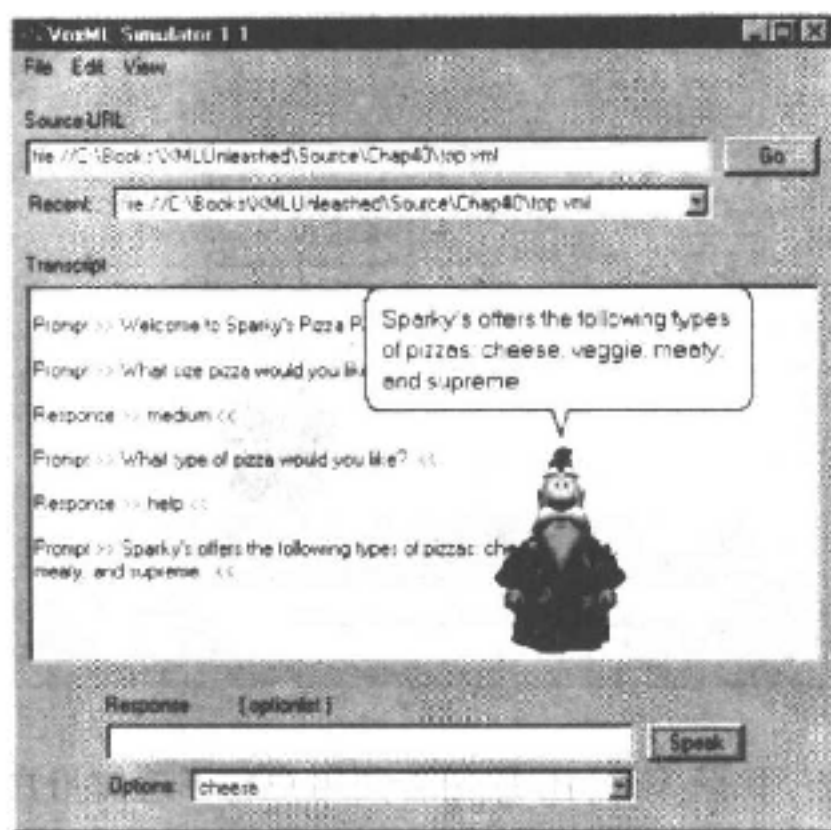


图 40.5 比萨 VoxML 应用提供与比萨类型相关的帮助

在用户从下拉列表中选择“supreme”后,应用询问用户在比萨上喜欢要什么类型的面皮(如图 40.6 所示)。

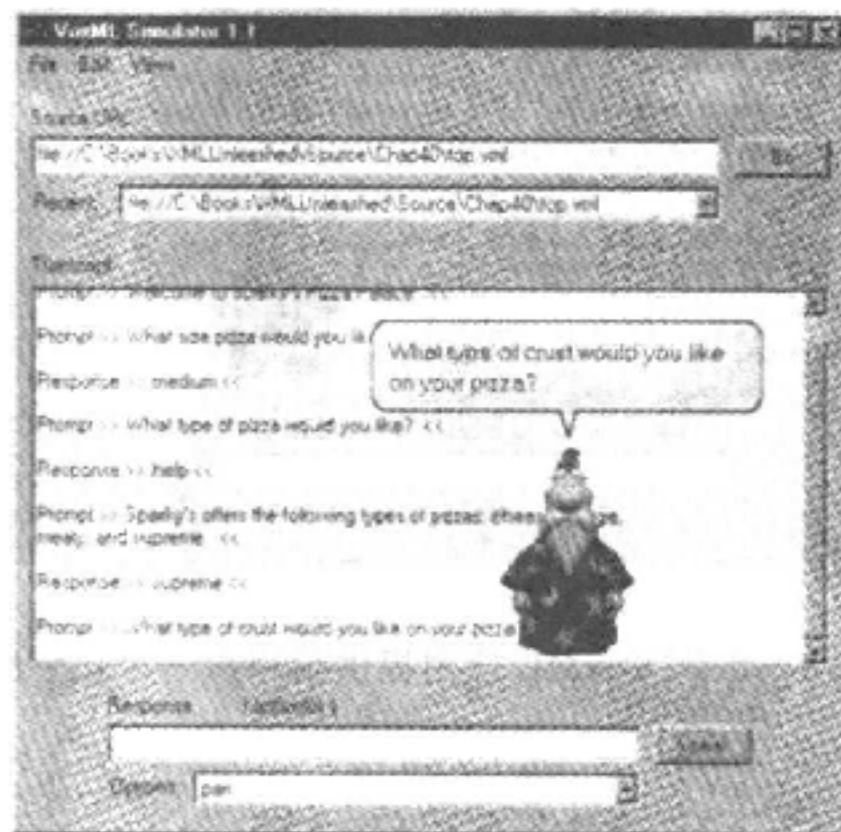


图 40.6 比萨 VoxML 应用要求用户提供比萨面皮的类型

如果用户回答的是无法识别类型的面皮,应用将回答它没有听懂(如图 40.7)。问题接着被重复。

用户从下拉列表中选择“thin”后,应用提供给用户一个比萨的概要(如图 40.8 所示)。

接着一个确认提示就提供给用户,用户要回答“Yes”或者“No”来接受和拒绝这个比萨订单(如图 40.9 所示)。

用户从下拉列表中选择“Yes”后,应用告知用户订单已经被处理并将很快准备好(如图 40.10 所示)。

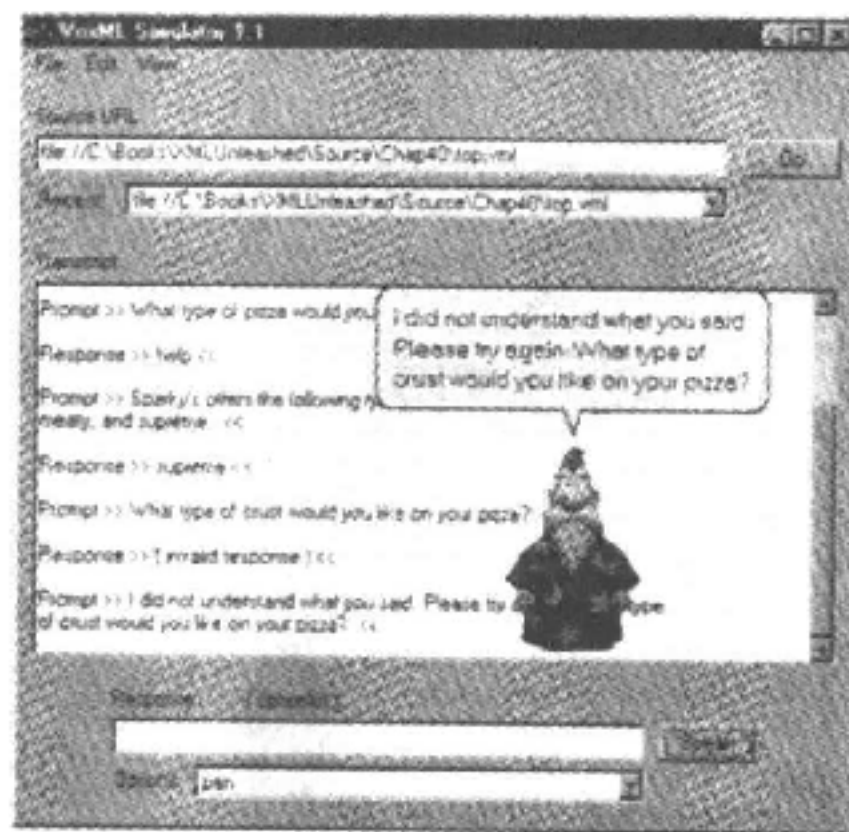


图 40.7 比萨 VoxML 应用告知用户它不理解用户的回答

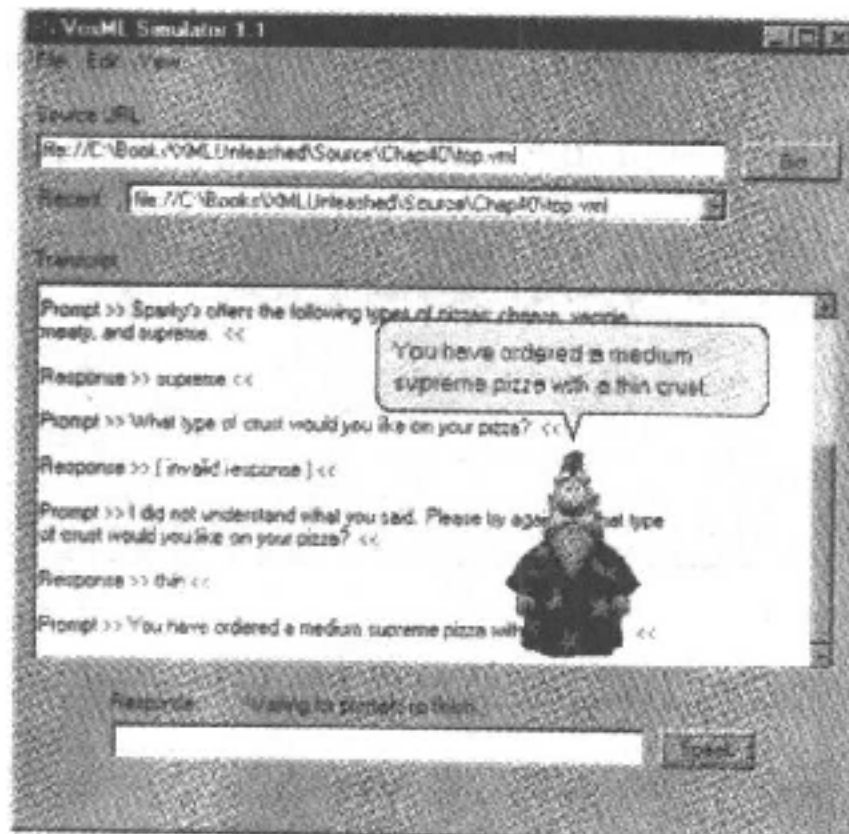


图 40.8 比萨 VoxML 应用提供给用户一个比萨的概要

40.5 总 结

本章向你介绍了名为 VoxML 的 XML 电话应用,它由 Motorola 创建来协助交互式语音应用的创建。VoxML 是当今构建语音响应系统的方式的一个重要的改进。它还是一个有趣的技术的融合,当然要使用 XML 并行语音合成以及语音识别。考虑一下在从文本生成语音和处理来自用户的语音响应方面包含了多少技术,我们就更感到 XML 使 VoxML 非常的简单且非常的强大。VoxML 也许是使我们超越传统的语音响应系统的技术,这些传统系统只会要求你“点击 2 作这个”和“点击 9 作那个”。

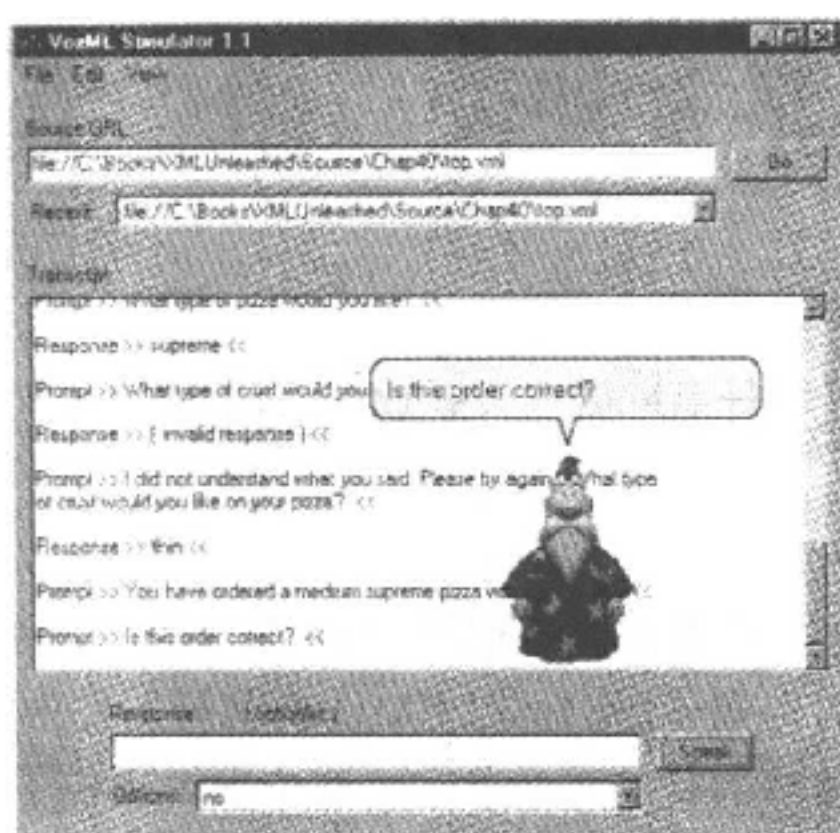


图 40.9 比萨 VoxML 应用要求用户确认比萨订单

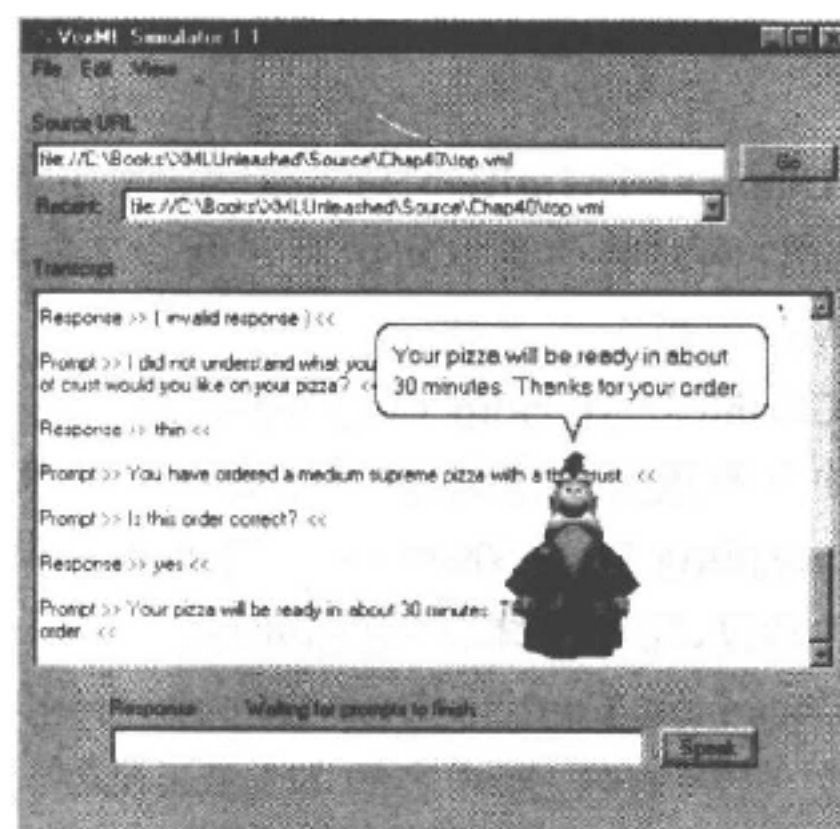


图 40.10 比萨 VoxML 应用告知用户订单已经被处理

第 8 部分 附 录

附录 A XML 缩略词

XML 是一个用来创建标记语言的标记规范,因此 XML 产生了大量名字末尾是“ML”的技术就显得不是那么让人奇怪了。实际上,XML 被有些人称为“缩写生成器”,只是因为它导致了如此多的其名字是缩写的新技术的产生。这个附录提供了与 XML 相关的大量缩写。

注意:我希望将这个附录作为 XML 缩写的 FAQ(常见问题解答),但这样我就不得不将“FAQ”加入到缩写列表中!

ASP(活动服务页:Active Server Pages)——由 Microsoft 创建的用于创建动态网页的服务器方脚本环境。

CDF(频道定义格式:Channel Definition Format)——由 Microsoft 创建的用于设置 Web 频道的 XML 词表,它允许网站将信息发送给客户端。CDF 对于 XML 的历史具有重要的意义,因为它是第一个 XML 应用。

CML(医药标记语言:Chemical Markup Language)——用来描述医药信息的 XML 词表。CML 对于 XML 的历史具有重要的意义,因为它是真正的第一个 XML 应用。

CSS(层叠式样式表:Cascading Style Sheet)——简单的给 Web 文件添加样式的机制。CSS 允许你控制网页的样式参数,比如字体、颜色和间距。

DCD(文件内容描述:Document Content Description)——作为 XML-Data 的子集实现的技术,它支持描述 XML 文件结构和数据内容的 Schema 机制。它与 DTD 不一样。

DDbE(实例型数据描述器 Data Descriptors by Example)——由 IBM 的 alpha 工作小组开发的技术,它由一个 Java 函数库组成,这个函数库设计用来从 XML 文件或文件集合中生成 DTD 或 XML Schema。

EDI(电子文件转换:Electronic Document Interchange)——使两个商业应用之间的结构信息可以进行电子转换的技术。

DHTML(动态 HTML:Dynamic HTML)——由三种技术组成的 Web 表述技术:HTML、JavaScript 和层叠式样式表(CSS)。DHTML 有时指的是“动画 HTML”因为它允许 HTML 页面的一部分动态的改变。

DOM(文件对象模型:Document Object Model)——最初由 Microsoft 创建的与平台和语言无关接口。它提供对 Web 文件的内部结构的动态访问。DOM 对于编写 HTML 和 XML 文件非常的有用。

DSSSL(文件样式语义及规范语言:Document Style Semantics and Specification Language)——最初设计用于 SGML 的一种强大的格式化、转换和搜索语言。DSSSL 的许多特性都结合在扩展样式语言(XSL)之中。

DTD(文件类型定义: Document Type Definition)——继承自 SGML 的标准 XML 文件结构解决方案。DTD 一般用来创建用于 XML 文件的未加工数据模型,它允许这些文件被验证。

HTML(超文本标记语言: Hypertext Markup Language)——当前作为 Web 发布标准的标记语言。在它的当前形式中(版本 4.0),HTML 是 SGML 的应用,而 W3C 的 XHTML 推荐将 HTML 改变为一种 XML 应用。

HRMML(人才资源管理标记语言: Human Resource Management Markup Language)——一个 XML 词表,它是用于表述履历和工作记录的描述结构。

HyTime(超媒体/基于时间的组织语言: Hypermedia/Time-based Structuring Language)——一个 SGML 的应用,它早于 Web。HyTime 的目的是用来作为提供超链接和多媒体特性的 SGML 词表。许多 HyTime 的特性已被结合进 XLink 和 XPointer 中。

MathML(数学标记语言: Mathematical Markup Language)——构造数学公式的 XML 词表,它便于数学记号的表述和数学内容的智能处理。

OFX(开放式金融交易: Open Financial Exchange)——用来便于储蓄、记帐付款、帐单描述和便于信息交换的表单的 SGML 词表。OFX 当前被 Intuit Quicken、Microsoft Money 和 CheckFree 使用。

OSD(开放式软件描述: Open Software Description)——用来便于自动软件安装的 XML 词表。OSD 用来与 CDF 联合提供软件升级频道。

P3P(个人隐私策略平台: Platform for Privacy Preferences)——针对规范化 Web 上的个人隐私策略的基础。P3P 的目标是提供给用户他们所访问站点的个人隐私策略的信息。

PGML(精确图像标记语言: Precision Graphics Markup Language)——由 Adobe 创建的 XML 词表,它基于 Adobe 的 PostScript 映射技术。虽然 PGML 是描述矢量图形的丰富词表,Adobe 仍创建了一个名为 SVG 的并发词表,它依赖于 PGML 并在许多方面超越了它。

PICS(Internet 内容选择平台: Platform for Internet Content Selection)——让 Web 用户控制他们和他们的孩子访问的资源种类的 RDF 词表。PICS 本质上浓缩了一个由隶属于 HTML 内容的标志组成的自主标称系统。

RDF(资源描述框架: Resource Description Framework)——提供给 Web 资源增加语义的方法的 XML 词表。

RELML(实际不动产清单标记语言: real Estate Listing Markup Language)——提供指定有关实际不动产列表相关的详细信息的方法的 XML 词表。

SAX(XML 简单 API: Simple API for XML)——设计用来通过基于事件的结构解析 XML 文件的标准编程接口。

SGML(标准通用标注语言: Standard Generalized Markup Language)——第一个意义重大的标准化结构信息技术,它作为 IBM 工作的一个成果,提供格式化和管管理诸如合法文件这样的结构化文件的方法。XML 是 SGML 的简化子集,它根据 Web 进行了调整。

SMIL(同步多媒体综合语言: Synchronized Multimedia Integration Language)——让 Web 开发人员在一个陈述中结合和同步化多种类型的媒体对象的 XML 词表。

SVG(可缩放矢量图片: Scalable Vector Graphics)——由 Adobe 创建的 XML 词表,它是 PGML 和 VML 思想的合并。SVG 更可能成为 Web 上的矢量图形标准。

VML(矢量标记语言: Vector Markup Language)——主要由 Microsoft 开发的 XML 词表,它支持格式化和样式化矢量图形。

VoxML(声音标记语言: Vox Markup Language)——由 Motorola 创建的 XML 词表,用来便于交互语音应用程序的开发。

WIDL(Web 接口定义语言: Web Interface Definition Language)——由 WebMethods 创建的 XML 词表,它实现了 HTML/XML 文件和表单的自动交互。

XFA(万能 XML: XML For All)——一个基于 Web 的脚本语言,它通过 XML 词表实现。

XHTML(扩展超文本标记语言: eXtensible HyperText Markup Language)——HTML 的一种版本,它是作为一种 XML 词表进行的再阐述。

XLink(XML 链接: XML Link)——完全 XML 链接技术以及使用这个技术的链接语言的名称。XLink 技术提供了将 XML 文件链接起来的先进机制。

XLL(可扩展链接语言: Extensible Link Language)——XLink 技术先前的名称,它提供将 XML 文件链接起来的先进机制。

XML(可扩展标记语言: Extensible Markup Language)——SGML 的简化子集,它结合了许多 SGML 的特性,包括可扩展性、结构化和有效性。XML 象征着 Web 的一个新的纪元,它建立了一种传输结构化数据的方法。

XMLNews(XML 新闻: XML News)——一个专用 XML,它定义了两个 XML 词表以给新闻故事添加丰富的内容。

XML-Data(XML 数据: XML Data)——描述 XML 文件的文件结构(Schema)的 XML 词表。与 DCD 一起,XML-Data 构成了 Microsoft 的 XML Schema 技术的基础。

XPath(XML 路径: XML Path)——一个用来访问 XML 文件各个部分的基于路径的语言。

XPointer(XML 指示器: XML Pointer)——一种基于 XPath 的语言,它用来访问 XML 文件的内部结构。XPath 用来与 XLink 相结合提供 XML 的先进链接机制。

XQL(XML 查询语言: XML Query Language)——以扩展 XSL 为设计目的的通用查询语言。

XSL(可扩展样式语言: eXtensible Style Language)——支持 XML 文件从一种类型到另一种类型转换,以及依照结构化格式规则样式化 XML 文件的样式表技术。XSL 实现 XML 文件显示的这两个方面,主要是通过下面两个技术: XSLT 和 XSL 格式化对象。

XSLFO(XSL 格式化对象: XSL Formatting Objects)——平台无关和目标无关的格式化信息的规范,格式化信息通过 XSL 处理器从 XSL 样式文件和 XML 文件中输出。

XSLT(XSL 转换: XSL Transformation)——用来作为 XSL 技术的一部分的 XML 词表,它支持 XML 文件从一种类型转换到另一种类型。

3DML(3-D 标记语言: 3-D Markup Language)——允许你使用标记设计名为 spots 的虚拟 3D 世界的标记语言。

附录 B XML 资源

与事实上的所有基于 Web 的技术一样,XML 仍然处于变迁的过程中。应该承认,XML 规范已经存在了一段时间,似乎在将来它也不会改变的太多了。然而许多其他的与 XML 非常相关的技术在我写这部书的时候依然处于变化阶段。把握 XML 最新技术的惟一途径就是要在网上冲浪了。Web 充满了 XML 的信息,到目前为止它是学习 XML 世界中新知识的最佳场所。

这个附录将提供大量的 Web 上的 XML 资源,它将协助你能学习到 XML 的最新技术。你可以把这个附录作为你的浏览器上的 XML 书签的起始点。我鼓励你访问这些资源来获得有关 XML 技术的最新信息。

B.1 一般 XML 资源

在研究这本书过程中,我断断续续的插入了大量的网上一般 XML 信息资源,范围从 XML 基础到大型的 XML 应用开发。你还将找到一些有趣的文章,那里记录有关 XML 陈述的注释,以及 XML 技术之间的关系。

- XML 常见问题解答(FAQ)——<http://www.ucc.ie/xml>
- XML.com——<http://www.xml.com>
- XML.org——<http://www.xml.org>
- Robin Cover 的 XML 网页——<http://www.oasis-open.org/cover.xml.html>
- Schema.net——<http://www.schema.net>
- XML 软件——<http://www.xmlsoftware.com>
- XML 区域——<http://www.xml-zone.com>
- XML 文件——<http://www.webdeveloper.com/xml>
- Web Monkey(XML)——<http://www.hotwired.com/webmonkey/xml>
- Web 开发人员的虚拟图书馆(XML)——<http://www.stars.com/Authoring/Languages/XML>
- Microsoft 的 XML 网站——<http://msdn.microsoft.com/xml>
- IBM 的 alpha 工作组网站——<http://alphaworks.ibm.com>
- W3C 的 XML 十要点——<http://www.w3.org/XML/1999/XML-in-10-point>
- Builder.com 的 XML 20 问——<http://builder.cnet.com/Authoring/Xml20/index.html>

B.2 XML 规范

XML 是一项非常基于标准的技术,这意味着事实上它的每一方面都在标准规范中进行

了严格的定义。下面是有关 XML 的主要 W3C 规范：

- XML——<http://www.w3.org/XML>
- XML 名字空间——<http://www.w3.org/TR/REC-xml-names>
- XML-Data——<http://www.w3.org/TR/1998/NOTE-XML-data-0105>
- 文件内容描述(DCD)——<http://www.w3.org/TR/NOTE-dcd>
- XML Schema——<http://www.w3.org/1999/05/06-xmlschema-1>
- XPath——<http://www.w3.org/TR/xpath>
- XPointer——<http://www.w3.org/TR/WD-xptr>
- XLink——<http://www.w3.org/TR/WD-xlink>
- XHTML——<http://www.w3.org/TR/xhtml1>
- 第 1 层 DOM——<http://www.w3.org/TR/DEC-DOM-Level-1>
- 统一字符标准编码——<http://www.unicode.org>

B.3 XML 浏览器

虽然 XML 解决了很多 HTML 的问题,但 XML 实际上只对于那些支持它的浏览器有价值。如果没有可靠浏览器的支持,XML 与其他昙花一现的 Web 技术比起来也好不到哪里去。遗憾的是现在还没有很多浏览器支持 XML。对于局中人,Microsoft Internet Explorer 是唯一支持 XML 的商用 Web 浏览器,但它的支持远未完美。Netscape 最近发布 Navigator 5.0,它对 XML 的支持比较基本并且依然没有精确的遵守规范。但还有一些其他的专用浏览器支持 XML。

由于浏览器支持 XML 的形式依然存在很大的问题,我们就非常的需要不断了解有关浏览器的最新信息。下面是一些资源,它们可以帮助你监视浏览器支持 XML 的最新情况。

- Internet Explorer ——<http://www.microsoft.com/ie>
- Netscape Navigator——<http://www.netscape.com> 和 <http://www.mozilla.org>
- DocZilla——<http://www.doczilla.com>
- Amaya——<http://www.w3.org/Amaya>
- Opera——<http://www.opera.com>
- WebReview 的浏览器兼容性图标——<http://webreview.com/pub/guides/style/style.html>
- Web 标准网站——<http://www.webstandards.org>

B.4 XML 工具

XML 工具在很大程度上使得 XML 如此迅速的变为 Web 上的标准开发技术。有一些 SGML 驱动的工具支持 XML,在我编写本书的时候,许多的 XML 工具正在出现。然而,这里还是有很多的用来创建 XML 内容和构造 XML 应用程序的工具。下面是当前支持 XML 的工具的主要类型:

- 验证服务
- 解析器
- 制作工具
- 内容管理工具

B.4.1 验证服务

验证服务是提供验证 XML 文件服务的网站。要承认许多独立的 XML 工具提供验证特性,但有趣的是这些基于 Web 的工具不需要你下载和安装任何东西。只要点击就可以了!下面是支持 XML 的两个验证工具:

- Microsoft 的 XML 验证器——http://msdn.microsoft.com/downloads/samples/internet/xml/xml_validator
- W3C HTML 验证服务——<http://validator.w3.org>

B.4.2 解析器

虽然验证服务对于快速确定 XML 文件的有效性非常的方便,XML 解析器是深入 XML 文件的真正主力。许多 XML 解析器也可以作为验证器。实际上,XML 解析器根据它们是否在解析 XML 文件期间执行有效性检查而被分为几类。

解析器的主要作用是读取和解释 XML 文件,并把它们存在内存中的一棵逻辑树内。这个树之后就作为访问和管理包含在文件中的信息的基础。所有的 XML 应用依赖于某种类型的 XML 解析器以显示 XML 文件的内容。下面是一些在本书写作期间可以获得的常用 XML 解析器:

- Lark 和 Larval XML Java 解析器——<http://www.textualit.com/Lark>
- Sun 的对象 X Java 解析器——<http://java.sun.com/xml/xml-side1.html>
- Datachannel 的 Java XJPARSER——<http://xdev.datachannel.com/downloads/xjparser>
- IBM 的 XML4J Java 解析器——<http://alphaworks.ibm.com/tech/xml4j>
- Oracle 的 XML Java 解析器——<http://www.oracle.com/xml>
- Microstar 的 Aelfred XML Java 解析器——<http://www.microstar.com/aelfred.html>
- Expat——<http://www.jclark.com/xml/expat.html>
- IBM 的 XML4C C++ 解析器——<http://www.alphaworks.ibm.com/tech/xml4c>

B.4.3 制作工具

XML 制作工具用来创建和编辑 XML 文件。所用制作工具中最简单的就是文本编辑器,它不是 XML 专用的并且不提供任何创建和编辑 XML 文件的方法。下面是在本书编写期间可获得的 XML 制作工具的例子:

- SoftQuad XMetaL——<http://www.softquad.com/products/xmetal>
- Adobe FrameMaker + SGML——<http://www.adobe.com/prodindex/framemaker/>

prodinfosgml.html

- Arbortext ADEPT-Editor——<http://www.arbortext.com/products/ADEPT-Series/adept-series.html>
- IBM Xena——<http://www.alphaworks.ibm.com/tec/xena>
- Vervet Logic Web XML Pro——<http://www.vervet.com/product-index.html>
- Bluestone Visual-XML——<http://www.bluestone.com/xml/Visual-XML>
- Microsoft XML Notepad——<http://msdn.microsoft.com/xml/notepad>
- Emile——<http://www.in-progress.com/emile>
- Emacs PSGML——<http://www.lysator.liu.se/projects/about-psgml.html>

B.4.4 内容管理工具

就像它们的名称一样,内容管理工具提供管理 XML 内容的方法。这些类型的工具主要相当于一个存储 XML 文件的数据库或文件系统。如果你在开发一个 XML 信息的大规模库,内容管理工具是非常有用的。下面是在本书编写期间可获得的 XML 内容管理工具的例子:

- Poet Content Management Suite——<http://www.poet.com/products/cms/cms.html>
- Arbortext Epic——<http://www.arbortext.com/Products/Epic/epic.html>
- Chrystal Astoria——<http://www.chrystal.com>
- Oracle 8i——<http://www.oracle.com/database/oracle8i>

B.4.5 其他工具

这里还有一些难以归类的 XML 工具。大多数这些“其他”工具以某种方式处理 XML 文件,并生成诸如转换文件这样的结果。它们当然不能被认为是确认、制作或是内容管理工具。因此我们将继续称它们为“其他”工具。下面是在本书编写期间可获得的其他工具的列表:

- XML Authority——<http://www.extensibility.com>
- Near and Far Designer——<http://www.microstar.com/near.html>
- Stylus——<http://www.transformis.com/products.shtml>
- HTML Tidy——<http://www.w3.org/People/Raggett/tidy>
- XML For All(XFA)——<http://www.xmlforall.com>

补充一下,XML Authority 是 Schema 开发工具,它允许你不用考虑诸如 DTD 或 XML-Data 这样的专门的 Schema 格式,就可以创建抽象的 Schema。Microstar 出品的 Near and Far Designer 是 DTD 建模工具,它提供构造 DTD 的 GUI 方法。Transformis 出品的 Stylus 是第一个开发 XSL 样式表的完整开发环境。Dave Raggett 出品的 HTML Tidy 是一个 HTML 清除和转化工具,它可以用来将 HTML 代码转化为 XHTML 代码。最后,XML For All 是一个基于 XML 的脚本语言,它依靠 XML 词表在网页上添加脚本功能。

B.5 XML 词表

XML 是元语言,这意味着它用来创建标记语言以表述特殊类型的信息。XML 词表描绘

prodinfosgml.html

- Arbortext ADEPT-Editor——<http://www.arbortext.com/products/ADEPT-Series/adept-series.html>
- IBM Xena——<http://www.alphaworks.ibm.com/tec/xena>
- Vervet Logic Web XML Pro——<http://www.vervet.com/product-index.html>
- Bluestone Visual-XML——<http://www.bluestone.com/xml/Visual-XML>
- Microsoft XML Notepad——<http://msdn.microsoft.com/xml/notepad>
- Emile——<http://www.in-progress.com/emile>
- Emacs PSGML——<http://www.lysator.liu.se/projects/about-psgml.html>

B.4.4 内容管理工具

就像它们的名称一样,内容管理工具提供管理 XML 内容的方法。这些类型的工具主要相当于一个存储 XML 文件的数据库或文件系统。如果你在开发一个 XML 信息的大规模库,内容管理工具是非常有用的。下面是在本书编写期间可获得的 XML 内容管理工具的例子:

- Poet Content Management Suite——<http://www.poet.com/products/cms/cms.html>
- Arbortext Epic——<http://www.arbortext.com/Products/Epic/epic.html>
- Chrystal Astoria——<http://www.chrystal.com>
- Oracle 8i——<http://www.oracle.com/database/oracle8i>

B.4.5 其他工具

这里还有一些难以归类的 XML 工具。大多数这些“其他”工具以某种方式处理 XML 文件,并生成诸如转换文件这样的结果。它们当然不能被认为是确认、制作或是内容管理工具。因此我们将继续称它们为“其他”工具。下面是在本书编写期间可获得的其他工具的列表:

- XML Authority——<http://www.extensibility.com>
- Near and Far Designer——<http://www.microstar.com/near.html>
- Stylus——<http://www.transformis.com/products.shtml>
- HTML Tidy——<http://www.w3.org/People/Raggett/tidy>
- XML For All(XFA)——<http://www.xmlforall.com>

补充一下,XML Authority 是 Schema 开发工具,它允许你不用考虑诸如 DTD 或 XML-Data 这样的专门的 Schema 格式,就可以创建抽象的 Schema。Microstar 出品的 Near and Far Designer 是 DTD 建模工具,它提供构造 DTD 的 GUI 方法。Transformis 出品的 Stylus 是第一个开发 XSL 样式表的完整开发环境。Dave Raggett 出品的 HTML Tidy 是一个 HTML 清除和转化工具,它可以用来将 HTML 代码转化为 XHTML 代码。最后,XML For All 是一个基于 XML 的脚本语言,它依靠 XML 词表在网页上添加脚本功能。

B.5 XML 词表

XML 是元语言,这意味着它用来创建标记语言以表述特殊类型的信息。XML 词表描绘

不同的通过 XML 创建的标记语言,它们是应用 XML 的最终结果,从而来解决现实世界的问题。下面是一些解决这样的问题的主要 XML 词表:

- XML News——<http://www.xmlnews.org>
- 同步多媒体综合语言(SMIL)——<http://www.w3.org/1999/08/WD-smil-boston-1990803>
- 频道定义格式(CDF)——<http://msdn.microsoft.com/workshop/delivery/cdf>
- 矢量标记语言(VML)——<http://www.w3.org/TR/NOTE-VML>
- 可缩放矢量图片(SVG)——<http://www.w3.org/Graphics/SVG>
- 3-D 建模语言(3DML)——<http://www.flatland.com>
- 数学建模语言(MathML)——<http://www.w3.org/TR/REC-MathML>
- 资源描述构架(RDF)——<http://www.w3.org/RDF>
- 个人隐私策略平台(P3P)——<http://www.w3.org/TR/WD-P3P/basedata.html>
- 实际不动产清单标记语言(RELML)——<http://www.4thworldtele.com/rm/rs/reservices.html>
- 人才资源管理标记语言(HRMML)——<http://www.structuredmethods.com/hrxml>
- VoxML——<http://www.voxml.com>

B.6 其他资源

就像有其他 XML 工具一样,我们还有一些其他 XML 资源。这些是一些与 XML 有点关系但并不须被考虑为“XML 技术”的技术资源。下面是一些其他资源,你会在你使用 XML 时发现它们很有用:

- Microsoft 的脚本网站——<http://msdn.microsoft.com/scripting>
- Javasoft 网站——<http://java.sun.com>
- Gamelan——<http://www.gamelan.com>
- Developer.com——<http://www.developer.com>
- 脚本新闻组——<http://www.scripting.com>