

目 录

前言.....	(1)
第一章 XML 的兴起	(3)
“所见即所得”带来的麻烦.....	(3)
HTML 的快速成长	(5)
追溯起源:结构和 SGML	(7)
HTML 的根源	(8)
XML 的出现	(9)
第二章 内容与形式分离:标记和样式.....	(11)
HTML 的根:原来的规范	(12)
结构化格式:级联样式表.....	(15)
XSL	(32)
树状结构(转换)	(33)
格式化	(36)
XSL 的未来	(37)
第三章 简单的 XML:建造结构	(39)
浏览器和分析程序	(39)
建造块	(41)
元素和标记	(41)
元素和属性	(44)
XML 和 HTML	(46)
创建自己的标记:具有良好构造的文档.....	(48)
用分析程序和浏览器测试文档	(51)
第四章 项目规划	(55)
XML 用户.....	(56)
结构	(57)

文档结构	(57)
数据结构	(60)
元素和属性	(65)
XML 和 SGML	(66)
即将到来的“模式”	(67)
为处理做规划	(68)
第五章 文档类型定义	(69)
分析简介	(69)
从最简单的开始	(71)
DTD 序言部分	(80)
<? xml? >	(80)
文档类型声明	(83)
批注	(85)
处理指令	(85)
逻辑结构	(86)
元素	(86)
属性	(92)
名称空间	(98)
数据结构和类型	(101)
实体	(102)
通用实体	(102)
参数实体	(105)
注解声明	(108)
DTD 中的标记区域: IGNORE 和 INCLUDE	(109)
综述	(110)
第六章 利用 XML 重新建立 Web 和网页文档	(113)
从 HTML 到 XML	(113)
关于本书	(124)
第一关: 看起来象旧样式表的 DTD	(126)
第二关: 使 DTD 更加清晰	(140)
建立包装文档	(146)
第七章 XML 用于商务	(149)

谁将看我的 XML	(149)
一种更好的电子目录.....	(150)
把脚本加到 HTML 中	(156)
商务交易.....	(158)
信息交换.....	(164)
ECIX: SGML 的例子	(164)
开放贸易协议(OPT)	(165)
银行 Internet 支付系统(BIPS).....	(165)
XML/EDI	(166)
信息和内容交换(ICE)	(166)
美国报纸协会标准分类广告数据	(167)
 第八章 XML 用于文档管理	(169)
XML 的继承性:SGML 和文档管理.....	(169)
XML 文档管理的未来	(170)
向着无纸化办公迈出的一小步.....	(171)
创建历史:DTD 用于公用内存	(182)
 第九章 XML 用于数据驱动应用程序	(189)
用于互相交换的数据.....	(189)
用于控制的数据.....	(190)
房屋控制	(190)
用工具标记语言控制工具	(197)
对象文档.....	(199)
Bean 标记语言(BeanML)	(199)
MDSAX 和 Coins	(200)
XML-RPC	(201)
使用 XML 元数据描述资源	(203)
通道定义格式	(203)
XML 软件自动更新(XSA)	(205)
资源描述框架(RDF)	(206)
Dublin Core	(206)
未来.....	(208)
 第十章 Xpointer 规范说明	(209)

通过查询指向	(209)
XPointer 介绍	(210)
绝对定位项	(210)
相对定位项	(211)
属性定位项	(212)
范围定位项	(213)
字符串定位项	(213)
合并定位项	(214)
第十一章 Xlink 规范说明	(217)
简单的链接	(217)
HTML 中的链接	(218)
Xlink 中的简单链接	(220)
用 XML 重构 HTML	(223)
定位器和分段	(225)
更复杂的链接	(226)
链接的奥秘	(231)
第十二章 处理 XML: 仓库、处理器和网关	(233)
用 XML 创建分布的体系结构	(233)
仓库(服务器)	(233)
处理器(CGI、Middleware 和 Beyond)	(235)
网关(浏览器、编辑器和其它应用程序)	(236)
创建用户的 XML 应用程序	(246)
编写 XML 程序的工具	(247)
第十三章 XML 及其未来	(249)
当前 Web 站点的体系结构	(249)
变迁的体系结构	(250)
从 HTML 语法到 XML 语法	(252)
在 Web 上 XML 的含义	(253)
作为应用程序体系结构的 Web 浏览器	(256)
XML 和 Web 的未来	(257)
名词解释	(259)

前言

可扩展标记语言(XML)早已被新闻媒体大肆炒作。本书《XML 基础教程》(第二版)将展示 XML 的真实面目,论述其目前现状及将来的几个发展方向。

本书面向的读者

本书适合于开始学习 XML 的任何一位读者。虽然它侧重于信息处理的文档方面,但是那些对如何使用 XML 进行原始数据交换感兴趣的读者同样会在本书中找到有用的信息。HTML 开发人员已经具备一定的基础,他们在接触 XML 时已对基本的标记有一定的理解,但我希望任何一位具有一定 Web 基础知识的读者都能够理解本书。本书是一本入门读物,书中不会对一些不常见的 XML 代码和面向程序员的代码示例作过多的解释,但这并不妨碍从中获得 XML 的基础知识及对 XML 标记的通盘了解。

本书的组织结构

本书的第一版在组织编排上力求向具有一定 HTML 开发经验的人员提供一条学习 XML 的尽可能快的途径。这种倾向仍然在一定程度上存在于本版中,具有其它背景的开发人员可以根据自己的实际情况跳过其中的一些章节。

本书前三章介绍 XML 的基本情况。第一章着重介绍 HTML,目前 HTML 仍旧是在国际互联网传输信息的主导方式。第二章论述显示信息的工具,以及实现标记以使标记可以集中面向内容的样式表。第三章讨论基本的 XML 语法并介绍分析程序;分析程序是 XML 的一个关键的基础工具。

接下来的六章讨论文档类型定义(DTD),文档类型定义是 XML 中一个更加复杂同时又更加强大的部分。第四章描述数据建模中涉及到的一些任务,从而为后面的工作做好准备。第五章向你提供建造 XML 文档类型定义所必需的工具。第六、

七、八、九章展示如何一步一步地建造 XML 文档类型定义。第九章可能会令程序员更感兴趣。

再接下来的两章讨论 XLink 和 XPointer,二者虽然放在最后讨论,但对 XML 来说是两个很有发展前途的补充工具。这两个标准将有助于建造下一代 Web 导航,其涵义将远远地超过目前的狭窄范畴。

第十二和十三章是全书的总结,将论述 XML 给几个技术领域带来的影响。这些领域包括开发 Web 浏览器和客户-服务器体系结构。

本书使用的图标

本书大部分是文本和图片的一般性混和编排,但经常有一些重要的信息需要单独列出,这些信息全部放在提示、参照或警告中。



“注意”提供有关正在讨论的话题的额外细节。当然,这些细节并非人人都感兴趣。“注意”中提供的信息是有用的,但不是关键性的。



“提示”中提供的信息通常不是传统文档中出现的,而是通过经验得来的。任何情形下都可以使用“提示”,但当“提示”出现时,它一定能够为你节省时间。



“当心”图标很重要。你可以认为不存在问题,但看上去很正常的行为完全可以带来可怕的结果。为避免潜在的灾难发生,请务必阅读这些警告。



本书中会经常提到某些问题在其它地方讨论会更好。按照“交叉参考”所指向的地方,你会找到有关某话题的更进一步的信息,这些信息通常是非常有用的。

第一章

XML 的兴起

XML 的目标就是要改变 Web 的基本结构,超越 HTML 并代之以更强大、更具有可扩展性的体系结构。XML 旨在使 Web 返回到基于内容的结构,而不再是开发人员强加给它的基于格式的结构,因为开发人员已被不成熟的 Web 设计工具弄得灰心丧气。XML 还可以通过结束浏览器开发巨头在 Web 元素开发和实现上的垄断地位而将 Web 从他们的手中解放出来。同时,XML 还向应用开发人员承诺,不论他们是否在 Web 上工作,XML 都会向他们提供储存不同种类信息的非常方便的格式。

国际互联网联盟(简称 W3C,见站点 <http://www.w3.org>)以一种很有前途的新的标记方法而远远地走在商业浏览器开发商的前面。XML(可扩展标记语言)使开发人员创建自己可交互操作式标记语言成为可能,它包括 HTML 但又不局限于 HTML。由于新增加的特征转移到组件模型而不是单个程序,XML 的使用可能会导致 Netscape 和 Microsoft 之间浏览器大战的停火,甚至可能会促使新的浏览技术的出现。XML 带来的更直接的影响是,它使得开发人员能够创建以逻辑内容而不是以格式为基础的标记结构。这将使人和计算机能够更容易地在文档内搜索具体的基于内容的信息,而不是仅仅搜索一个页面上的文本。与样式技术相一致的 XML 将使作者能够创建出很容易管理的漂亮网页,使开发人员能够更好地控制信息,并使这种控制具有极大的灵活性。

“所见即所得”带来的麻烦

我曾经使用的第一个文字处理程序是一个非常简单的文本编辑器。当时看到通过在屏幕上四处移动光标使只能显示 40 列字符的屏幕可以将具有 80 列字符的页面显示出来。但它仅仅有助于做家庭作业,或编写其它类似的需要在点阵打印机上输出的枯燥文档。在使用计算机(一边使用计算机编程一边咒骂计算机)许多年后,我抛弃计算机而购买了一台电动打字机。它让我能够做一些有趣的事情,如不需要键入奇异的换码字符就可以给文本加上下划线。尽管没有敲打黑体文本的好方法,但

我不用再担心因为编排换码字符出错而浪费大量的纸张。打字机以传统的纸张和墨水的方式给了我“所见即所得”的感性认识。

我坚持使用这台打字机好几年,直到发现 Macintosh 计算机。当 Mac 计算机第一次出现时,我对它并没有什么好感,因为当时发行的许多杂志都以我不曾拥有的这种昂贵计算机做封面。而它甚至连一个象样的编程软件包都没有。但四年后当再一次遇到 Mac 计算机时,我被惊呆了。用它书写论文实在太棒了!因为我可以锁定使用所有样式信息,可以将页面分成多栏,甚至可以偶尔使用 72 点字型。尽管在我的 ImageWriter 打印机上输出的页面看上去不是很好,但比原来的点阵计算机文本强多了。在呈交的论文上,我使用了斜体字的标题和参考书目,将页面分成几栏,甚至还使用了一两张图片。书写再也不是简单的组织语句,而完全可以创建标题、子标题、表格和脚注,还可以使用其它各种格式,使得即使一篇很短的论文也可以包含多种结构,从而看上去很漂亮。通过使用样式,可以一次应用一组格式化工具,给这组格式化工具命名,以后可以对它进行调用。这看上去简直就跟魔术一样。

十年后我仍旧使用标题和子标题为我的文档安排格式,我并不关心脚注。但这时出现了一个新问题:很难重新使用老文档。当我在撰写某学年的论文时这无关紧要——写完交上去之后就再也不用想它了。但我现在整天跟若干年前由距离几千里外的人所写的大量信息打交道时,我很少需要关心如何将这文件转化成相同的文字处理格式。我发现自己经常花费数小时时间所做的并不是编辑这些材料,而是重新格式化它们,这当然不是因为我喜欢这样做。滥用制表符和空格符(使用打字机养成的习惯)成长起来的一代人创建的文档无法剪切和粘贴到其它文档,因为所有东西都断开了。行断点彻底错误,文本全部堆积到左边或右边,表格已不成形,甚至象行间隔等简单的东西也出现问题。给创建具有漂亮外观的文档带来极大方便的神奇格式化手段现在却带来了这么多严重的问题。

另外,还有一些其它细微的问题。那些年我认为自己是在创建标题和子标题,而实际上却不然,我实际上创建的是格式上象标题的文本。我们可以将这些样式叫做“标题”,但对计算机来说它们只不过是另外一组没有内在意义的字符。“所见即所得”也改变了人们。那些可能连五年级美术班都没有毕业的人也可以在页面上使用多达三十种字体。当这些字体渐渐失去新意后,他们中的许多人开始采用较为保守的方法来排布格式,其意图仍旧是使他们的文档看上去符合他们所想要的样子。设计人员已经习惯于指定位置时精确到千分之一英寸——仿佛人人可以分辨出这么小的尺寸。

在“所见即所得”之前,文档无疑是不美观的,但它们也有一些其它优点,尽管这些优点往往不被注意。有一些创建文档管理和文档标记系统以使计算机能够有效地管理大规模文档库的行动曾被酝酿过。简单的纯文本虽然单调,但与常见的文字处理程序或桌面出版程序的输出相比,纯文本更容易管理。这些文档管理工具在“所见

即所得”的早期还是很新的,直到后来用户逐渐习惯于使用基于纸张的媒体的系统之后,这些工具才慢慢地变得不算昂贵,也慢慢地多起来。目前市场上大多数程序的最终目标仍旧是文档的打印输出结果。

HTML 的快速成长

当国际互联网第一次在 1994 年受到广泛的关注时,一小伙业余爱好者和专业设计人员开始着手创建他们能够创建的最激动人心的网页。但许多人很快就打了退堂鼓,因为习惯于完全“所见即所得”环境的他们实在因 HTML 格式化工具的缺乏而失望。各浏览器之间莫名其妙的差异使得很难预测一个网页看上去会使什么样子,公司用户需要象控制纸张页面文档那样控制他们的电子文档。在一定时期内这些抱怨和需求刺激了 HTML 的发展。象之前许多 Internet 技术一样,HTML 是因为它本身有趣而被出于兴趣的爱好者们传播开来的。HTML 很简单,只需一两天时间就能够学会,它向人们提供了一种全新的阅读和写作体验。这些早期的爱好者们所产生的冲力以及新闻媒体的相关报导,为 HTML 发展成后来的引人注目的事物提供了巨大的推动力。

虽然 Web 的发展已使其在经济上成为一块很有生命力的市场,但 HTML 必须改变自身以满足用户的需求。设计人员和他们的雇主希望能够创建出看上去完全符合他们需要的网页,而且希望能够达到象一般桌面出版系统所提供的那样的控制水平。经常爆发的浏览器大战(这时受伤的通常是网页开发人员)使 HTML 变得更加强大(尽管不兼容),国际互联网联盟(W3C)已或多或少地兼顾各方,出台了 HTML 4.0。尽管准确地讲,HTML 对普通用户来说仍旧不是一个简单的页面排版系统,但它提供的工具已方便多了。Web 设计已经成为遍布全球的设计人员和通信专家的一项专门性工作,使公司和个人创建高级(尽管并非总是在视觉上令人愉悦)站点成为可能。

表格是 HTML 设计的一大进步,尽管它们的广泛使用推翻了关于应用表格的许多观点。设计人员使用表格创建与许多打印设计中使用的传统单元格相似的文档,使用这些表格的情形远远地超过了一般的成行成列的表格状信息显示。可点图技术的不断进步,使设计人员在 HTML 不能产生出他们所需要的界面时,可以使用可点图创建出自己的指向-点击界面。框架和弹出式窗口使开发人员可以将精力集中于页面元素,而不需要每次想改动某个小地方时都得重建整个屏幕的信息。〈FONT〉标记使得能够比基于结构的格式化更加精确地指定文本显示。随着 Microsoft 和 Netscape 争夺市场份额的竞争不断升级,两个公司都向调色板添加了各种工具。Netscape 创建了〈BLINK〉,Microsoft 则创建了与之相对应的〈MARQUEE〉。两个公司都创建了 HTML 元素及其属性的扩展,这使开发人员大感困惑,需要花费许多时间和设备在多个浏览器中检查站点。更糟糕的是,两个公司都不以完全相同的方式

实现标记。间隔可能会改变,颜色也可能改变,就是精心对齐的元素也可能会散乱在页面上。

注意

- 如果有关标准的争论或者销售商明显的无意遵循标准令你烦恼,可以查看互联网标准工程站点(<http://www.webstandards.org>)。HTML 的半标准地位销售商对其它标准(包括 CSS 和 XML 本身)的部分实现已经驱使许多 Web 开发人员以实际行动来捍卫标准。

同时,大型 Web 站点上的网页数目在不断膨长,站点经常膨长到包括 10,000 个以上网页。这些网页经常由于开发人员对超文本知之甚少,对组织管理知道得更少,而在层次模式方面组织得很松散。许多站点是根据混乱的目录结构进行组织的,建造这些目录结构的开发人员往往习惯于原来的 FTP 档案库和 Gopher 站点的结构,而 FTP 和 Gopher 都是 Web 的先驱。大型站点将接下来的困难交给必须维护它们的管理人员和试图阅读它们的用户。导航用的超文本本身是一种奇怪的艺术形式,它是组织技能、记忆、好的设计和运气的结合。设计导航超文本尤为困难。搜索引擎可以帮助用户寻找其路径,但很快就会发现即使拥有强大计算能力的管理人员也很难跟得上这种新媒体(站点)的爆炸式膨长。

诸如 Crawler 和 Robot 之类的自动化工具已开始搜索大量的 Web 文档。有一些只是索引标题,更高级的搜索工具则开始索引页面的整个内容。AltaVista(<http://www.altavista.com>)最初是由 Digital 公司创建来演示和促销其 Alpha 处理器一个搜索引擎,它应用共享千兆内存和巨大带宽的多个处理器,以“暴力”方法索引 Web。尽管 AltaVista 和其它许多搜索引擎能够提供服务,但它们的工作标准最宽松:一个文档的整个内容。我们设法给搜索引擎赋予一定的智能,甚至让它们区分语言和处理文字的形狀,但要让它们在我们不用指出哪一部分是哪一部分的情况下读取、分类和组织文档,还要走很长的路。大多数情况下,人们用于识别文档之关键部分的格式化信息通常是被搜索引擎抛弃的。

这种容量和日益复杂的格式化的结合使开发人员想知道是否有更好的标记方法。HTML 已经走了很长的路,它走得很快,但其本身作为一门被设计为用于格式化的标记语言的局限性已日益明显。随着浏览器大战进入一个新的阶段,开发人员开始需要一种替代者,以使浏览器能够决定如何显示某个标记。每次 Web 站点在规模上翻一番时,搜索引擎的局限性都显得越来越明显。最后,随着 Web 的逐渐普及,HTML 在显示不容易符合标准文本和图形模型的信息时,表现出来的局限性越来越突出。开发人员需要能够创建他们自己的标记集,而且需要在创建自己的标记集时,所采用的方式能够与其客户使用的浏览器相一致。

追溯起源:结构和 SGML

当吉姆·伯纳斯·李于 1991 年创建 HTML 时,他将 HTML 基于一门更加强大但却更加复杂的标记语言,它就是标准通用标记语言,简称 SGML。是时 SGML 已经以各种形式存在了 20 年,但由于其复杂性,除出版、政府和大规模信息处理等部门之外,很少有其它组织采用它。因此,SGML 标记、管理和处理是专门的技术,只有一少部分政府、公司和学术界的用户能够掌握。

那些包括觉得 HTML 标准发展太慢的开发人员应该回过头来看看 SGML 发展的艰难步伐。最早形成于 19 世纪 60 年代的通用标记语言(GML),于 1969 年在 IBM 公司由研究人员 Goldfarb、Mosher 和 Lorris 同时创建。后来 Charles Goldfarb 于 1978 年出任美国国家标准协会(ANSI)的文本处理计算机语言委员会的主持人,此时 GML 已经成为出版行业中的一个重要标准。该委员会在 1980 年公布了它的第一个工作草案,截止 1983 年,它的第六个工作草案已被国内税务局和国防部等用户采购,它们又命令其合同商也使用 SGML。在 1984 年,该委员会发展成一组协作共事的子委员会,它们为国际标准组织(ISO)和美国国家标准协会(ANSI)开发标准。在 1986 年,也就是 SGML 的标准化进程进行了八年之后,SGML 成为国际标准组织的 ISO 8879:1986 号标准。当然,有关 SGML 的工作仍在继续。有一组委员会定期评估 SGML 的发展变化,包括脚本样式表、多媒体、链接扩展和各种文档的管理问题。

注意

要想了解 SGML 的所有规范(差不多过几年就有一些改动),请查阅可以称得上是 SGML 宝典的下面这本书:《SGML 手册》,作者是 Charles F. Goldfarb,牛津大学出版社 1992 年出版。

与 HTML 不同,SGML 并不指定文本如何显示。SGML 不是一门格式化语言,甚至不是一门特定的标记语言。SGML 是允许人们创建他们自己的标记语言的一套规范。它规定的内容识别符使文本的一致性格式化非常容易,从而使文档管理系统能够快速地为信息定位。SGML 非常适合于涉及大量结构相似的数据工程,如目录、手册、清单、转录和统计摘要等。SGML 最受联邦政府、IBM 和其它大公司的喜欢。它使开发人员能够集合在一起方便地开发数据结构规范,创建文档类型定义(DTD),然后将它运用到整个组织内的文档。

更重要的是,在许多情况下,用 SGML 创建的文档能够容易地移植到不同的格式。因为 SGML 使用基于内容的标记,而不是基于格式的标记,所以改变格式化规

则很容易,仅取决于文档输出到点阵式直打打印机、激光打印机、四色印刷机、CD-ROM、Web 站点或者甚至是音频喇叭。设计小组决定使用的格式与开始时 DTD 开发人员所做的工作相匹配,从而以适合具体输出媒体的样式显示信息。计算机化的存储系统也可以把文档当作小的数据库来对待,使用基于内容标记和索引信息的搜索对它们进行查询。重复使用相同信息的公司也因为事先制作好了适用于新文档的文本而受益匪浅。SGML 不会使所编写的东西更加漂亮,但可以使它们更容易管理。

HTML 的根源

SGML 对 HTML 的贡献大部分是语法,即使用“<标记 属性=值>内容</标记>”这一形式的标记语言。SGML 将内容与格式分离的意图有一些也保留在 HTML 中,从不同的浏览器对同一标记的完全不同的解释可以看出这一点。用表示强调,用<ADDRESS>表示地址信息,通过这种描述元素的方式,伯纳斯·李创建了一门足以灵活应付许多种不同信息的简单的格式化语言。从<H1>到<H6>这六个标记描述不同层次的标题,向文档提供一种比较自然的结构。<HEAD>和<BODY>标记将元信息(首先是<TITLE>与文档中的可见文本分离开来。最重要的是,标记为超文本链接提供了一个简单而强大的结构。

HTML 为简化 SGML 做了很好的工作,使业余爱好者也能够使用标记,这是扩大标记的影响力所必需做的。极具讽刺意味的是,伯纳斯·李当初从没有打算让用户必须手工键入代码。最初在欧洲核研究委员会所做的试验,使用的是一个不可视地管理代码的简单的标记处理器。虽然 HTML 只是一小部分具有少量属性的标记集合,但 Web 却是用来交换网页和共享信息的。

但是,正如我们已经看到的,面对设计人员借助“所见即所得”工具而毁坏的这个世界,HTML 显得很不适合。尽管超链接和很容易使用的 Web 都是好东西,但从一开始,就有异议认为这些空谈到底能为无用的格式化语言带来什么起色。在浏览器大战刚开始时,想要在 Web 上创建出与在纸上同样细致(不考虑屏幕分辨率的限制)的文档的设计人员和开发人员,从长远看肯定不会接受 HTML 最初极其简单的格式化工具。简单标准固有的灵活性缺乏号召力。唯一职责是格式化的标记充斥着 HTML 领域,从和<I>直到都远远超过和<ADDRESS>的使用频率。设计人员动手制作 HTML,将各种标记混合搭配在一起创建出他们想要的外观,而不考虑文档结构。因为 HTML 标记只是用来指定格式而没有可替代的格式化结构,所以它注定是一门格式化语言,而不是用于文档的结构化框架。早期所有的问题都慢慢地暴露出来,而这些问题在一开始时,只是光彩夺目的 HTML 中一些基本的瑕疵。

XML 的出现

当初 Web 标准的创建者和他们的接班组织 W3C(国际互联网联盟)一时蜂拥而上,试图赶上商用浏览器开发商的步伐。“Netscape 扩展”向设计人员提供了 HTML 早期版本所缺乏的控制,促进了 Netscape 的壮大。直到最近 W3C 才追赶上浏览器开发商并以以下这套强大的标准拦住了他们的去路。级联样式表(CSS)是 W3C 发出的第一击。CSS 使开发人员能够为文档声明他们自己想要的格式,而无须将一堆标记和图形混合到一起。CSS 使标记从携带格式信息的重担下解放出来,允许它们再次携带内容信息。通过提供一套完整的描述格式的术语(比 HTML4.0 全面),CSS 提供了一种能够与 HTML 和其它任何标记语言相兼容的样式工具。

一旦有哪种格式化标准与标记不匹配,W3C 完全可以创建一种利用 SGML 灵活性的标记标准。XML 强调提内容信息的重要性,它使设计人员能够创建和管理他们自己的元素集。设计人员可以使用这些与 CSS 相一致的元素集来创建产生格式的标签(如果愿意的话,他们当然可以这么做),但主要的重点在于管理内容,包括超文本链接,使它们本身得到极大增强。XML 的崛起来自于 W3C 的 SGML 社评委员会的关注,因为他们感觉到 HTML 在朝着错误的方向发展。为了改善现状,他们提出了一种与现有 Web 技术相一致的标记语言,使用的一些工具仍旧是为使用 HTML 开发的,但以其更加容易管理的技术而向前迈进了一大步。XML 提供的是 SGML 功能的一个子集合,而不仅仅是使用 SGML 语法的一套标记。它虽然是 SGML 的简化(有一些 SGML 用户认为简化的程度不够高),但发誓要恢复 Web 最初的承诺,将混乱复杂的网页创作现状简单化,若想经获得了多大的成功,只需要看看 W3C 为下一代 HTML 的产生所订的计划(见 <http://www.w3.org/MarUp/Activity.html>),这一计划将使当前巨大的 HTML 工程分解为许多用 XML 定义的子模块。

XML 由从 SGML 社评委员会演变而来的 W3C 的 XML 工作组控制。XML 工作组(见 <http://www.w3.org/XML/>)控制与 XML 直接相关的大部分规范,包括 XML 语法、XML 扩展链接和 XML 分段使用。同时,W3C 的其它工作组控制其它支持性标准,如文档对象模型(DOM)、级联样式表(CSS)和可扩展样式语言(XSL),以及补充或应用 XML 的标准,如数学标记语言(MathML)、多媒体同步一体化语言(SMIL)以及资源描述框架(ROF)等。虽然以上这些由 W3C 控制,但 XML 还有许多其它应用(它们当然也有自己的名称缩写)由其它组织、公司和个人所控制。尽管很有希望在将来由开发人员坚持 XML 标准,但 XML 现在是由销售商实现的。

XML 的实现已花了一段时间。当 W3C 于 1998 年 2 月发布 XML1.0 语法的推荐标准(官方标准)时,对 XML 的支持大部分停留在程序员能够使用的水平,但一般的 HTML 开发人员还不能够使用。这种状况现在正在慢慢改变,特别是 Microsoft 公司在 Internet Explorer5 中为 XML 提供了较高级的支持,以及 Netscape 的 Mozilla

工程最近也提供了对 XML 的巨大支持。但是,由于当时安装时没有此设置(人们并不经常升级他们的浏览器),还要等一段时间才能人人可以得到启用 XML 的阅读器、XML 编辑器及其它 XML 应用程序。

在本书中,我们要详细讨论 XML 提供的工具,以及开发人员怎样将它们用于常见的任务。HTML 开发人员将会发现许多信息是面熟的,但是也有一些内容(如创建 DTD)比较陌生。这本书不是面向 SGML 开发人员的,但熟悉 SGML 的读者会看到,许多熟识的概念与来自更广阔的 Web 世界的概念结合在一起。尽管本书着重讲述使用 XML 创建文档,我们也会论及管理 XML 以及将 XML 与其它 Web 技术相结合的一些技术。软件开发成为 XML 的重要市场,所以本书讨论的一些工程也会对软件开发人员有益。XML 可能乍看上去有些抽象,但随着你进一步阅读本书和体会书中的示例,其含义会越来越清晰。我们将看到,尽管 XML 文档与严格定义的层次结构或表式结构非常不相像,但 XML 文档集合仍旧可以当作数据库来对待;我们将探讨 XML 使之成为可能的新的文档结构和数据。

第 二 章

内容与形式分离:标记和样式

在讲述 XML 如何为内容建模之前,先得简单介绍一下 XML 是如何把我们从 HTML 的展示形式中解脱出来的。尽管我们已经在前一章指出了 HTML 的一些不完善的地方,但是从总体上来说 HTML 使标记得以普及,只是它的现有结构值得改进。我们将看一看 HTML 的一些不完善的地方,但是从总体上来说 HTML 使标记得以普及,只是它的现有结构值得改进。我们将看一看 HTML 在样式表之前(浏览器 3.0 及以前的版本)和之后(4.0 以后的版本)是如何被使用的。从 XML 的角度看 HTML,你会理解 XML 即将带来的新东西。样式表对 HTML 和 XML 都很重要,它将标记从束缚 HTML 很久的格式化结构中解放出来,使得标记能够专注于内容和重复使用,而不用将注意力集中于如何展示。

注意

如果你不关心展示的形式,或者不是具有开发人员的背景,那么可以跳过这一章直接进入使用 XML 为内容建模。本章最适合于将 XML 用于 Web 网页或打印文档等供人阅读的媒体的那些用户,而对于使用 XML 连接计算机和数据库则不太适合。本章有关 XSL 转换的讨论可能会对后者有用(尽管 XSL 转换现在还非常不稳定),但建议这些读者还是先跳过本章直接阅读后面的内容,等到发现需要这方面的信息时再返回来查阅。

HTML 是 SGML 的许多应用中的一个,国际互联网联盟(W3C)使用 SGML 文档类型定义为创建 HTML 的规则提供正式的定义。尽管有些 HTML 元素(例如指示行断开的 BR 元素)很难在结构更紧凑的 SGML 世界中有意义地表达出来,在至今出现的所有 HTML 版本中都可以使用文档类型定义。HTML 的语法一直比较宽松,倾向于 SGML 的多种可能性。例如,闭端标记传统上总是可用可不用的。直近随着 HTML 创建工具的发展,关闭每个标记才变成很寻常的事,而且仍有许多 HTML 创建工具依然不关闭标记。

浏览器长期以来一直忍受着语法使用中的多样性,尽管它们经常会因具体使用的语法而在显示时略有不同,通常它们会计算出一个元素在哪里结束以及另一个元素从哪里开始。这种语法的多样性产生了许多后果——一种浏览器显示文档的方式往往与同样的文档在其它浏览器中显示的样子完全不同。再加上浏览器如何格式化特定标记的定义很松散,这种语法上的灵活性,害得设计人员经常彻夜不眠地在多个浏览器中重复创建同样的格式,而从没找到给他们带来如此痛苦的那些标记。

HTML 的根:原来的规范

在表格、框架、字体标记和客户端可点图,以及 HTML 现在所拥有的所有神奇的工具出现之前,已经有一少部分标记提供常见的学术论文结构所具有的格式(毕竟,当初欧洲核研究委员会的研究人员创建 Web 时就是用它作为物理学家们交流学术的场所)。跟 SGML 不同,HTML 为其标记定义了格式用途,但是它们不象一般的“所见即所得”文字处理程序或桌面出版软件包中的提供的格式那样具体。

嵌入在开端 HTML 标记中的是 HTML 文档的两个重要的部分:HEAD 元素和 BODY 元素,二者分别携带不同的信息。HEAD 元素包含有关文档的数据,如:TITLE 元素,为文档中的所有超级链接设置基址 URL 的 BASE 元素,以及 META 元素。META 元素可以包含信息(由于它是有关数据的数据,因此也叫做元数据),这些信息涉及文档的作者、创建它的组织、供搜索引擎查找的关键字,以及页面创建和文档管理软件在跟踪该网页在更大的站点组织中的位置时使用的信息。同样可以出现在 HEAD 元素中的 LINK 标记将文档与样式表等外部资源连接在一起。

注意

在本章(乃至全书中)我所使用的 HTML 元素名称是常见的大写形式,如:使用 BODY 而不是 body,HTML 而不是 html。W3C 好象要在 HTML 的下一个版本中使用小写,而且随着 XML 的影响不断扩大,大小写的区分将对 HTML 的使用更加重要。

BODY 元素几乎是所有内容出现的地方。除了位于浏览器窗口顶部的标题之外,HEAD 的信息通常对用户来说是不可见的。BODY 部分中的信息产生 Web 页面的真正外观并吸引用户的大部分注意力。例如,大多数 Microsoft Word 用户让文件的属性框(其作用类似于 HEAD 元素)处于关闭状态,因为太没有必要为每个文件都键入搜索关键字。一般用户关心的是文档中的文本内容,再就是其格式。HTML 的 BODY 元素与文字处理文档中的正文相似。在 BODY 元素中,所有文本的标记是按顺序排列的,其顺序是常见的从左到右,从上到下(但是,如果你使用提非欧洲字符内

码,情形可能不同)。BODY 内部的大多数元素定义格式或创建图像、Java 应用程序、表单域、按钮和检查框等。HTML 元素为页面上的文本和其它对象的外观及摆放提供标记。

最初的 HTML 标记以总体方式定义文档结构,格式与这些结构大致相关。结构是逻辑结构,而不是以外观为基础的。H1 指示的是顶层标题,而不是 24 点加下划线的 Helvetica 粗体字。EM 的意思是强调,不是加粗、倾斜或加下划线。因为 Web 原来是设计成在多种设备上运行的,这些设备包括 NeXT 显像管、VT-100 终端、PC 机和 Mac 计算机,所以其创始者避开这些提供具体展示形式的标记,留给浏览器来实现每个标记的具体显示。一些早期的浏览器甚至把允许用户指定标记的样式作为它们的一个选项。

```
<HTML>
<HEAD> <TITLE> Simple Document, early HTML </TITLE> </HEAD>
<BODY>
<H1> Introduction to HTML </H1>
<P> This page has been created purely with logical tags. No additional formatting has been specified by the designers. </P>
<P> While it might be nice to specify text like we could in Quark XPress, we'll settle for applying
<EM> emphasis </EM> where appropriate, <CITE> citations </CITE> when necessary, and
maybe highlight a <VAR> variable </VAR> along the way. We can also indicate code listings: </P>
<
<CODE>
10 PRINT "HELLO WORLD" <BR>
20 END <BR>
</CODE>
<P> Bulleted lists are easy too: </P>
<UL>
<LI> HTML Structures </LI>
<LI> CSS Structures </LI>
<LI> XML Structures </LI>
</UL>
<P> Numbered and lettered lists are also fun: </P>
<OL>
<LI> Item #1 </LI>
<LI> Item #2 </LI>
</OL>
</BODY> </HTML>
```

即使在最新的浏览器中,这个简单的例子也会因浏览器的不同而产生不同的结果。在图 2-1 和图 2-2 中可以看到, Netscape Navigator 3.0 将 EM、CITE 和 VAR 显示为斜体,而 Internet Explorer 3.0 则将 VAR 显示为等宽字体,而且使用的背景颜色

也不同。

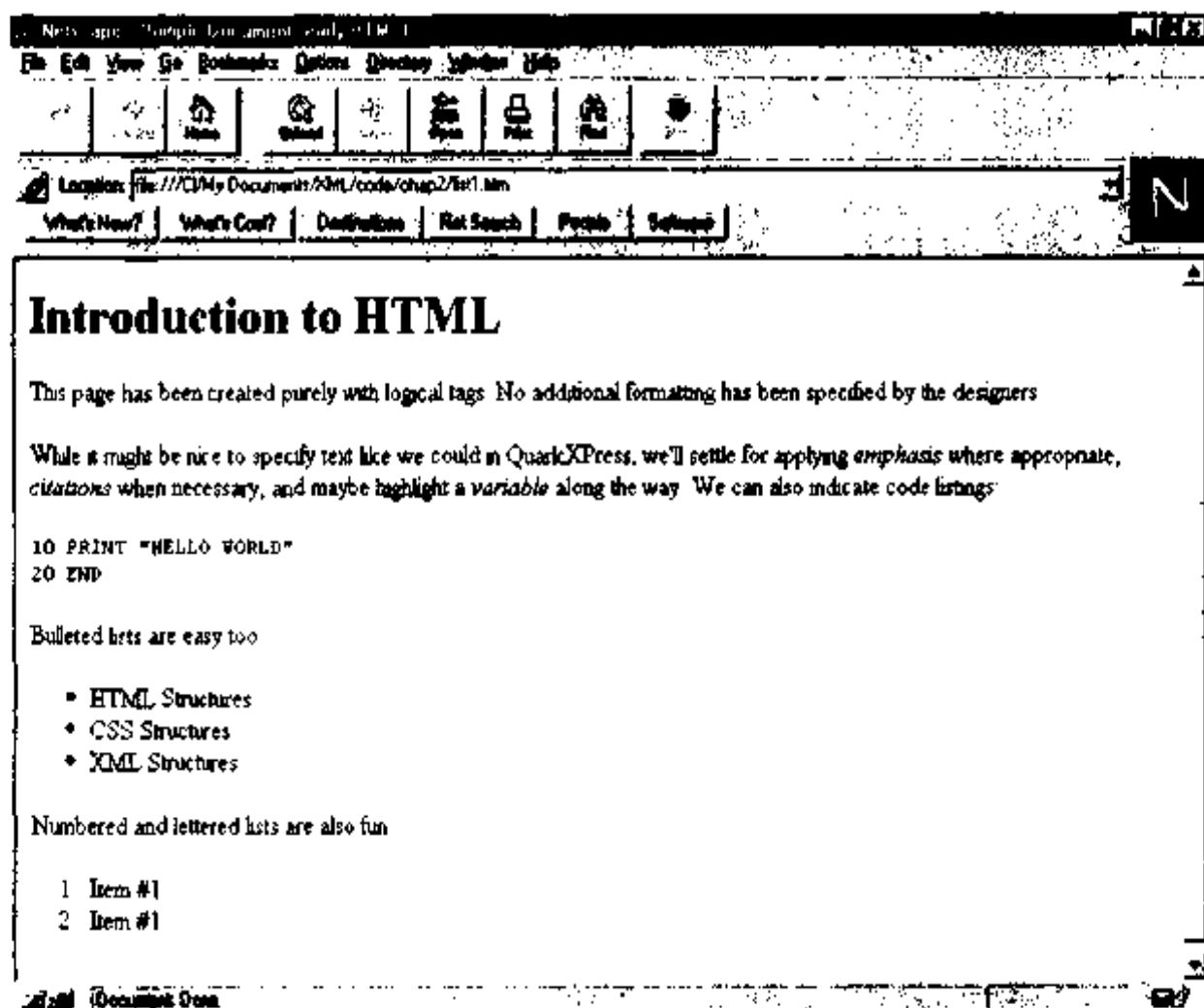


图 2-1 Netscape Navigator 3.0 中面向结构的标记

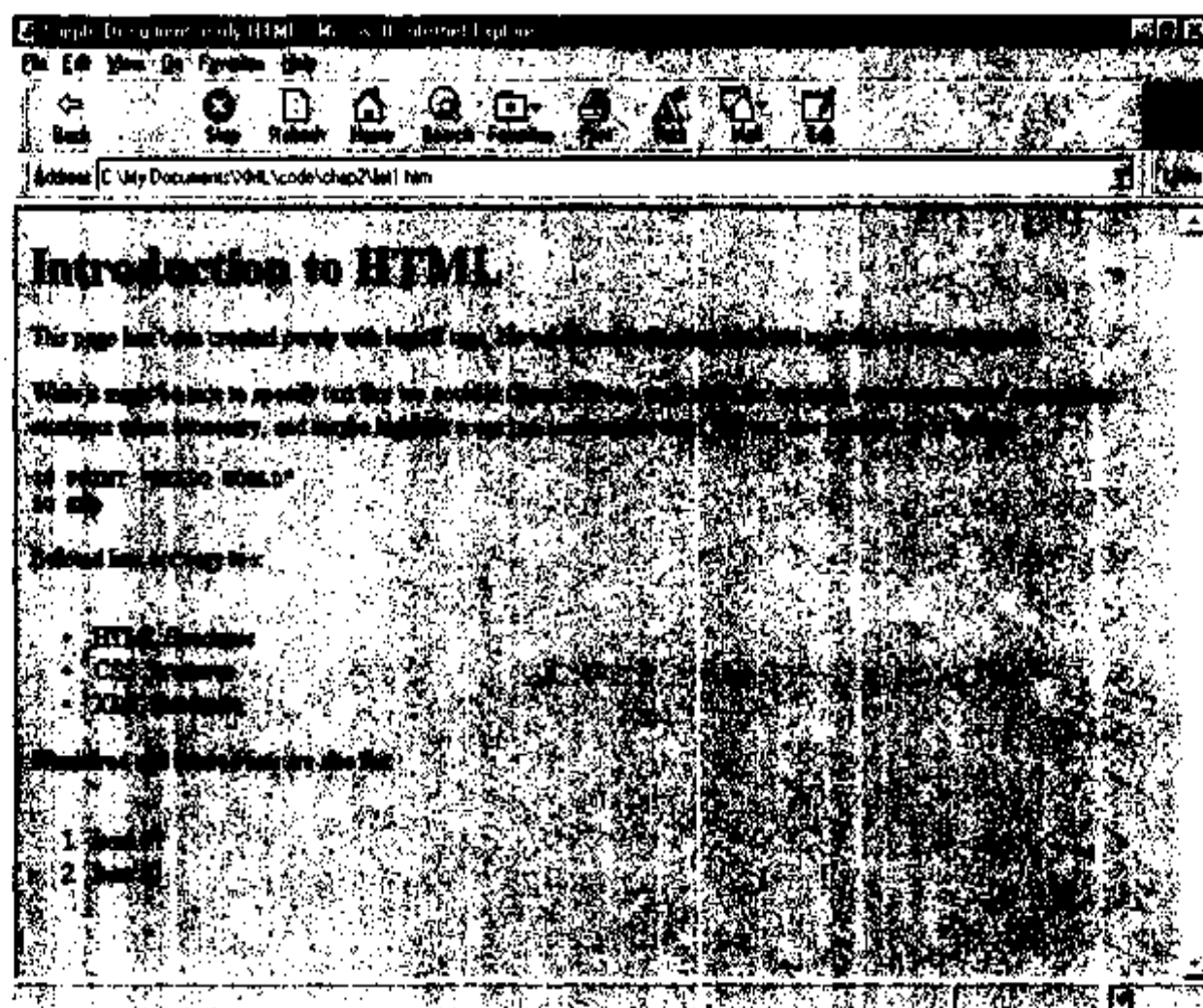


图 2-2 Microsoft Internet Explorer 3.0 中面向结构的标记

这些显示中的差别使设计人员非常不愉快。习惯了“所见即所得”工具的她

快要求更强大、更具有一致性的格式化工具的出现。FONT 元素提供了许多他们想要的工具,而表格、框架和围绕图形的文本的新属性使得可以更容易地把格式化后的文本精确地放到设计人员想要的位置。更重要的是,除了一些跨平台的字体问题,各浏览器所显示的网页也差不多相同。以下代码使用格式化代码(而不是逻辑代码)产生出与上一个示例中差不多相似的文本。

```
<HTML>
<HEAD><TITLE>Formatted Document, later HTML</TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF">
<FONT FACE="Arial, Helvetica" SIZE=7><B>Introduction to HTML</B></FONT>
<FONT FACE="Arial, Helvetica" SIZE=3>
<P>This page has been created with specific formatting tags instead of logical tags.</P>
<P>Since we'd like to specify text like we could in QuarkXPress, we'll use bold for <B>em-
phasis</B> where appropriate, italic for <I>citations</I> when necessary, and maybe
highlight a <FONT FACE="Courier"><B>variable</B></FONT> with bold Courier. We
can also indicate code listings with Courier;</P>
</FONT><FONT FACE="Courier" SIZE=2>
10 PRINT "HELLO WORLD"<BR>
20 END<BR>
</FONT>
</BODY></HTML>
```

这次, Netscape Navigator 中显示的文档(见图 2-3)与 Microsoft 的 Internet Explorer 中显示的(见图 2-4)几乎一模一样。所有的字体都具有相同的尺寸,所有格式也是相同的,因此我们没有必要担心各浏览器之间的太多差异。

尽管这给我们带来了一定的方便,但也有一些不幸的副作用,而且没有周到地考虑网页设计的各个方面。以上简单的示例并没有滥用 HTML,但使用自传统的“所见即所得”文档创建程序(如 Microsoft Word)加载来的表格和文档,会以许多个 FONT 元素结尾,这些 FONT 元素有时占用的空间比实际内容还多。此外,网页布局的方方面面也没有得到足够的考虑,这又给许多布局设计人员摆出了一个大问题。Netscape 和 Microsoft 两大公司正继续致力于一些布局问题的解决,特别是空白问题。基于标记的格式化实在做得太不够了!

结构化格式:级联样式表

国际互联网联盟(W3C)为了将 HTML 返回到更加结构化的过去,同时又满足 Web 设计人员为了有效使用 Web 而要求的克服总体外观的控制这一最大的拦路石, W3C 于 1996 年底公布了级联样式表一级规范。Microsoft 公司立即对此做出反响,

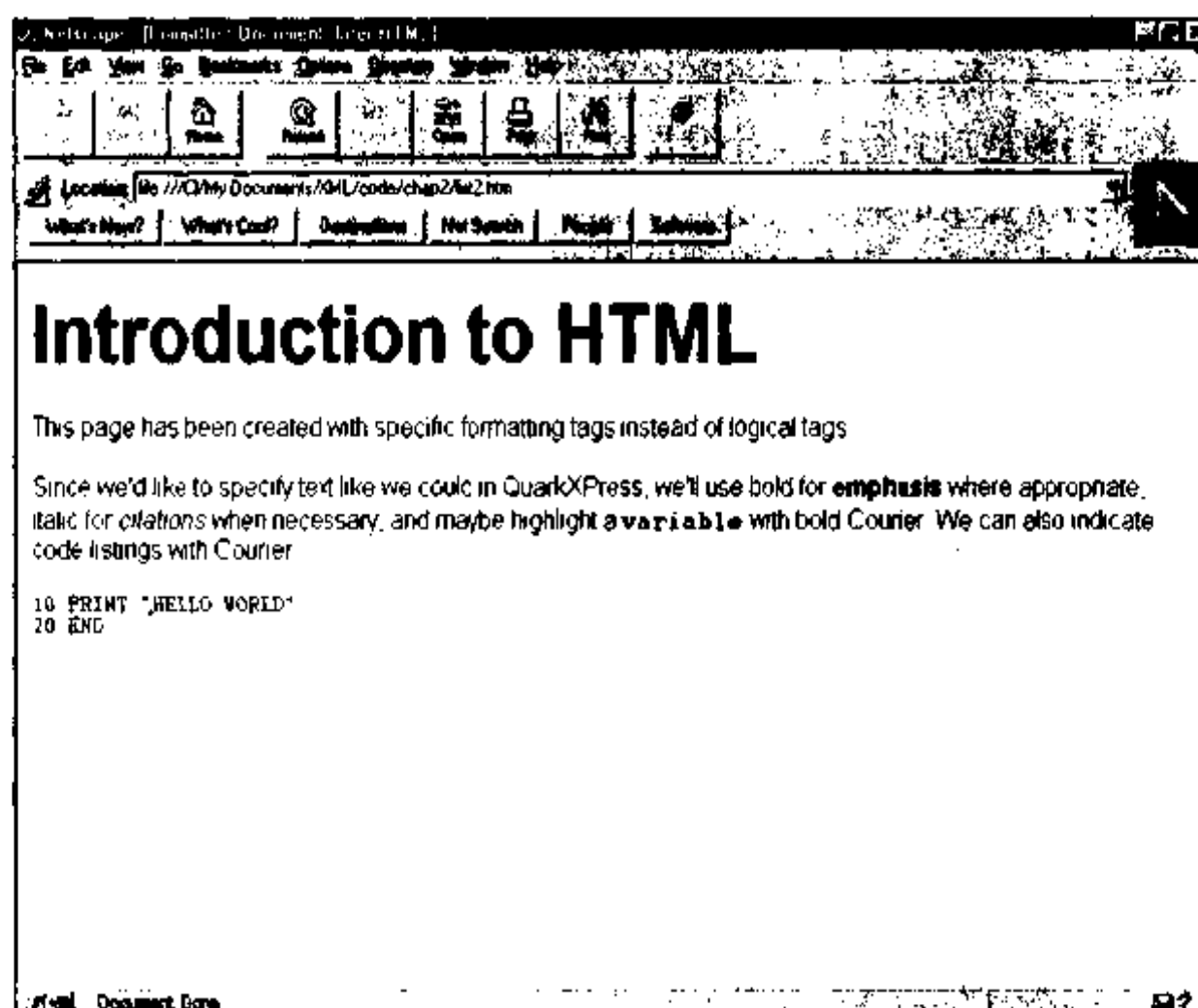


图 2-3 Netscape Navigator 3.0 中的明确的格式

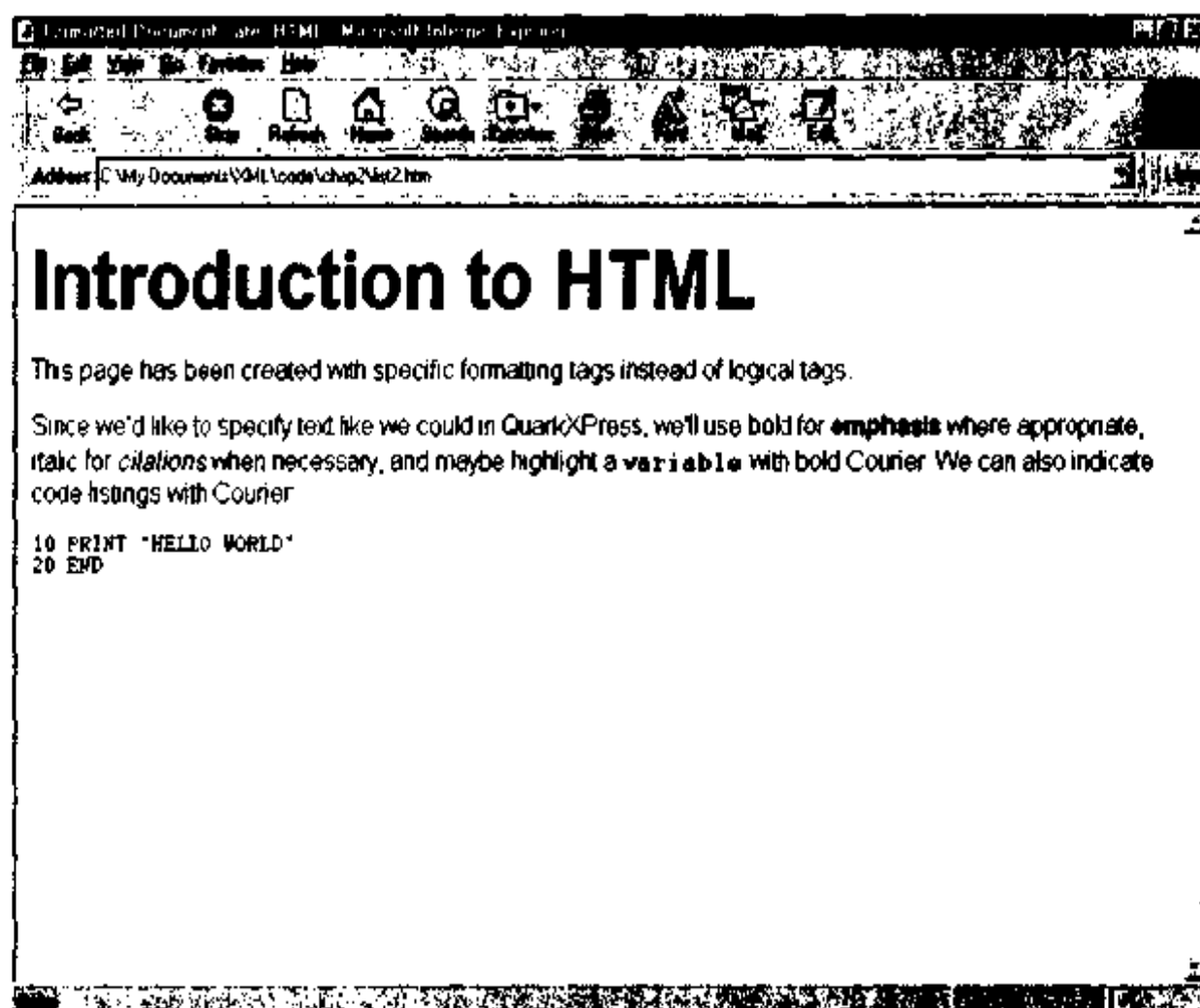


图 2-4 Internet Explorer 3.0 中的明确的格式

在 Internet Explorer 3.0 中实现了一些级联样式表的特征,并在 4.0 版本中增加

了更加强大的功能。当初提出它自己的 JavaScript 样式表标准的 Netscape 公司也追随级联样式表(CSS),在 Netscape Communicator 4.0 中实现了 CSS 的相当大一部分。于 1998 年 5 月推荐的二级级联样式表继续发展了该项技术,尽管二级 CSS 在目前的浏览器中还没有得以实现。

级联样式表把格式化信息与文档的正文分离开来,单独把格式化信息存放在 STYLE 元素或另外一个文档中。通过使用 STYLE 属性来指示特定元素的格式,也可以得到内联样式的功能。“级联”是指能够将多个样式和内联样式结合到一起,简化创建主模板的任务然后在需要的时候对主模板进行改动。级联样式表使用文档结构作为框架,然后被注释为格式化信息并被应用程序(现在一般是 Web 浏览器)显示(或打印、读取等其它展示形式)出来。

作为第一个关于样式表的示例,我们使用前面列举的演示逻辑标记的例子,为其中的所有标记提供具体的格式化定义。除了增加了一个简单的 STYLE 元素之外,它与我们以前使用的 HTML 完全相同:

```
<HTML>
<HEAD><TITLE>Formatting with CSS, modifying standard HTML</TITLE>
<STYLE TYPE="text/css"><!--
H1 {font-family: Arial, Helvetica; font-weight: bold; font-size: 24pt}
EM {font-weight: bold; font-style: normal}
CITE {font-style: italic}
VAR {font-family: Courier; font-weight: bold}
CODE {font-family: Courier}
LI {font-family: Arial, Helvetica}
--></STYLE>
</HEAD>
<BODY>
<H1>Introduction to HTML</H1>
<P>This page has been created purely with logical tags. No additional formatting has been specified by the designers. </P>
<P>While it might be nice to specify text like we could in QuarkXPress, we'll settle for applying
<EM>emphasis</EM> where appropriate, <CITE>citations</CITE> when necessary,
and maybe highlight a <VAR>variable</VAR> along the way. We can also indicate code listings:</P>
<CODE>
10 PRINT "HELLO WORLD"<BR>
20 END<BR>
</CODE>
<P>Bulleted lists are easy too:</P>
<UL>
<LI>HTML Structures</LI>
```

```

<LI>CSS Structures< /LI>
<LI>XML Structures< /LI>
< /UL>
<P>Numbered and lettered lists are also fun;< /P>
<OL>
<LI>Item #1< /LI>
<LI>Item #2< /LI>
< /OL>
< /BODY>< /HTML>

```

注意

STYLE 元素内部的限定不会影响对样式表的处理,但它会使不理解样式表的老式浏览器不把样式表的内容作为 Web 页面的一部分显示出来。

正如你在图 2-5 和图 2-6 中所看到的,我们能够更好地控制 Netscape Communicator 4.0 或 Internet Explorer 4.0 中这个文档的外观。(除 Netscape Navigator 3.0 之外,早期的浏览器会忽略该 STYLE 元素,因为我们在它的内容周围放置了注释符号。Internet Explorer 3.0 会解释该标记,但不会让它们重载它原来的 HTML 计划的格式,如在这里它将 EM 标记显示为加粗的斜体)。

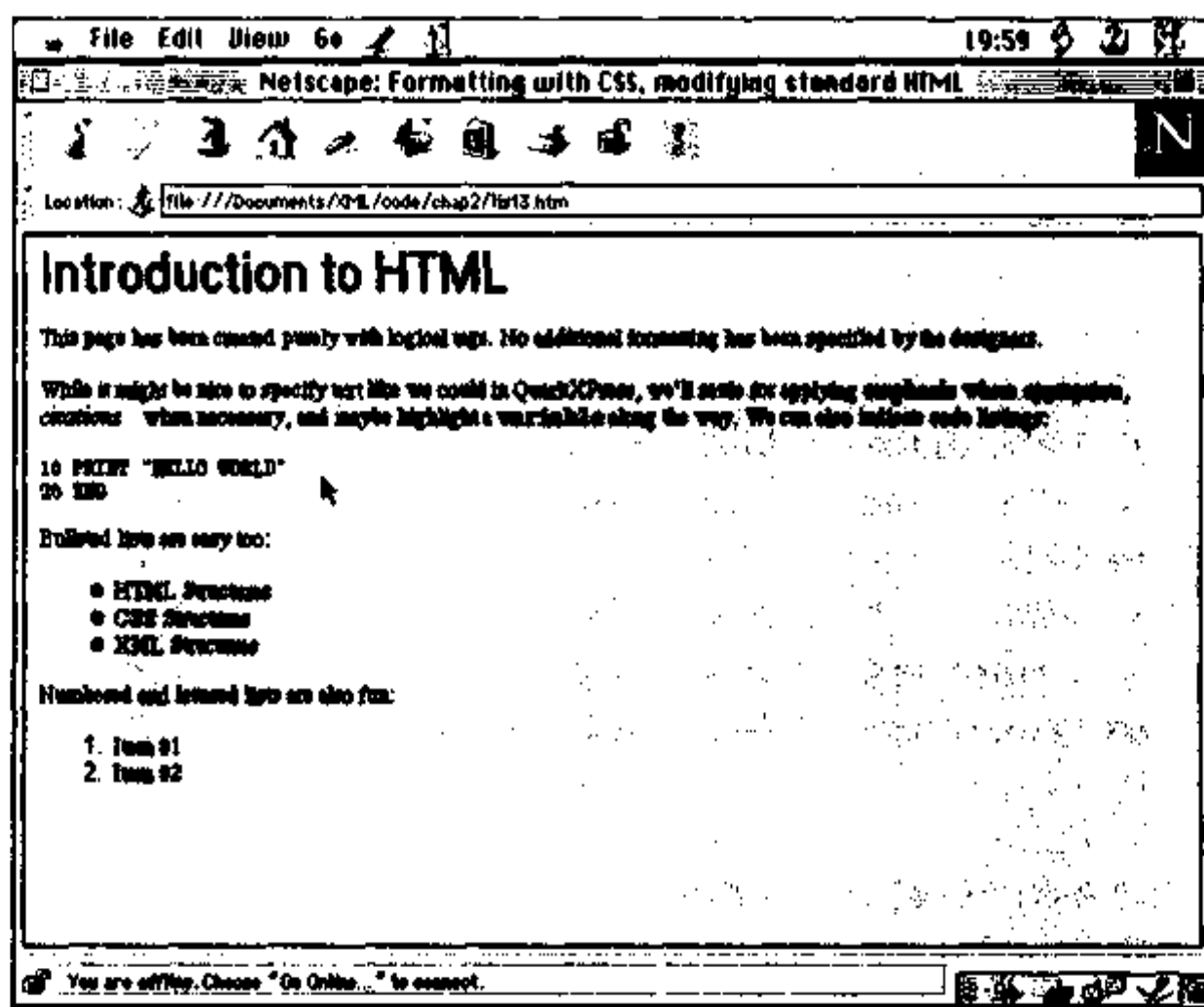


图 2-5 Netscape Communicator 4.0 中简单的样式表

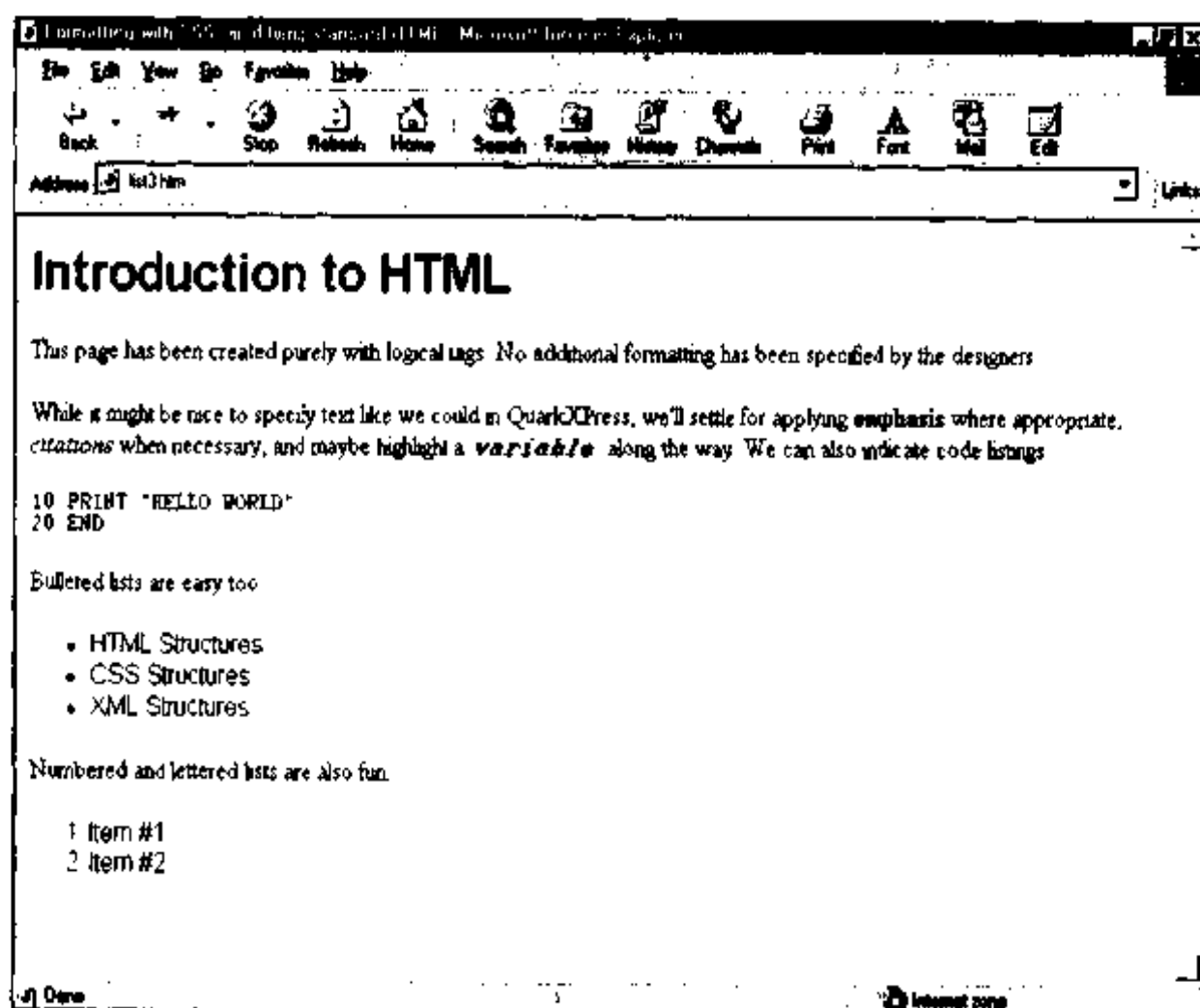


图 2-6 Internet Explorer 4.0 中简单的样式表

这是怎样工作的呢? CSS 使用选择符(它们以其最简单的形式对元素名称做出响应)为格式识别目标。紧跟选择符之后的花括弧中的信息是一组属性及其取值,它们将被应用于符合该选择符所建立标准的所有元素。一般的语法规则如下:

选择符{属性名称:值;属性名称:值}

有无空白并不重要,但使样式表具有一定的格式可以便于读者阅读。许多设计人员让每个属性值占用一行,以将信息排成清单的形式便于挑选其中某个属性。大多数情况下,阅读样式表是一件很容易的事情。在上面使用的样式表中,包括下面这一行:

```
H1 {font-family: Arial, Helvetica; font-weight: bold; font-size: 24pt}
```

在这里,“选择符”是指所有的 H1 元素。如果有 Arial 字体的话, H1 元素将被显示为 Arial 字体;否则,它被显示为 Helvetica 字体(如果二者都没有,浏览器将转向其默认字体。使用“sans-serif”作为值可以简化这一过程)。H1 元素将被加粗并以 24 点的尺寸显示。

样式表使开发人员能够精确地指定格式,使他们可以准确地控制字体、颜色、位置和空白等问题。表 2-1 列举了一小部分 CSS 属性,它们将对 XML 特别有用(也就是说,当销售商为 XML 实现 CSS 时这些属性就可以突出显示出其有用性来了)。

表 2-1 一些有用的 CSS 属性

属性	解释	可接受值	级别
background	在一个值中指定背景的所有可能	其它背景属性的一组取值	1,2
background-attachment	指定背景是随着内容滚动还是保持在一个地方不动	Scroll, fixed 或 inherit	1,2
background-color	元素的背景颜色	颜色名称(如“white”)或十六进制表示(如白色用#FFFF)	1,2
background-image	元素的背景图像	指明图像所在位置的 URL	1,2
background-repeat	指定背景是否重复	repeat, repeat-x, repeat-y, no-repeat 或 inherit	1, 2
border	在一个值中指定对元素边框所有可能的描述	其它边框属性的一组取值	1,2
border-bottom-color	设置底部边框的颜色值	颜色名称或其十六进制表示	1,2
border-bottom-style	设置底部边框的样式	none(默认值), dotted, dashed, solid, double, groove, ridge, inset 或 outset	1, 2
border-bottom-width	设置底部边框的宽度	thin, medium, thick 或具体的宽度数值	1,2
border-color	设置整个边框的颜色	颜色名称或其十六进制表示	1,2
border-left-color	设置左端边框的颜色	颜色名称或其十六进制表示	2
border-left-style	设置左端边框的样式	none(默认值), dotted, dashed, solid, double, groove, ridge, inset 或 outset	2
border-left-width	设置左端边框的宽度	thin, medium, thick 或具体的宽度数值	1,2
border-right-color	设置右端边框的颜色	颜色名称或其十六进制表示	2
border-right-style	设置右端边框的样式	none(默认值), dotted, dashed, solid, double, groove, ridge, inset 或 outset	2
border-right-width	设置右端边框的宽度	thin, medium, thick 或具体的宽度数值	1,2

属性	解释	可接受值	级别
border-style	设置整个边框的样式	none (默认值), dotted, dashed, solid, double, groove, ridge, inset 或 outset	1,2
border-top-color	设置顶部边框的颜色	颜色名称或其十六进制表示	2
border-top-style	设置顶部边框的样式	none (默认值), dotted, dashed, solid, double, groove, ridge, inset 或 outset	2
border-top-width	设置顶部边框的宽度	thin, medium, thick 或具体的宽度数值	1,2
clear	指定浮动块是否可以沿着对象的边浮动	none, left, right, both 或 inherit	1,2
color	设置元素的前景颜色	颜色名称或其十六进制表示	1,2
direction	指定文本流的方向,这对于网页的国际化很重要	ltr(从左向右), rtl(从右向左) 或 inherit	2
display	为如何格式化某个元素提供基本的描述	block, inline, list-item, none, run-in, compact, marker, inherit, table, inline-table, table-row-group, table-header-group, table-row, table-column-group, table-column, table-cell, table-caption	1,但在2级标准中得到了更大的加强
float	让某个块浮动	none, left, right 或 inherit	1,2
font-family	指定某个元素的字体或字体族	常见的有 serif, sans-serif 和 monospace;也可以是某个具体的字体族名称,如“Arial”,“Times”等,但并非在任一台计算机上都能使用所有的字体	1,2
font-size	设置元素的字体大小	可以使用 small, medium 或 large,也可以使用具体的点数	1,2
font-style	设置字体的样式,这里的样式是指水平和倾斜的类型	normal, italic 或 oblique	1,2
font-variant	用来创建小型大写字	normal 或 small-caps	1,2
font-weight	指定字体的粗细轻重程度	从 100 到 900 之间的整数值(如果支持的话),或者使用 normal, bold, bolder 或 lighter	1,2

属性	解释	可接受值	级别
height	指定元素的高度;用来在一个空间(一般是浏览器窗口)中定位元素	具体的尺寸	1,2
left	指定元素从窗口左边缘算起的左间距;用来在一个空间(一般是浏览器窗口)中定位元素	具体的尺寸	2
letter-spacing	提供字母及文本之间额外的间隔	具体的尺寸	1,2
line-height	指定文本的基线之间的距离,使得块内部存在行间距	可以是 normal,与点数尺寸相乘的数字,绝对尺寸或百分比	1,2
list-style-image	使图像(不是标准的项目符号)显示为项目符号	图像所在的 URL, none 或 inherit	1,2
list-style-position	指定如何将项目符号放入一个项目列表项中	inside, outside 或 inherit	1,2
list-style-type	设置默认的项目符号的外观	disk, circle, square, decimal 或 inherit	1,2
margin	在一个属性值中指定所有的间距属性	其它间距属性的一组取值	1,2
margin-bottom	指定元素底部保留多少空白间隔	具体的尺寸	1,2
margin-left	指定元素左端保留多少空白间隔	具体的尺寸	1,2
margin-right	指定元素右端保留多少空白间隔	具体的尺寸	1,2
margin-top	指定元素顶部保留多少空白间隔	具体的尺寸	1,2
overflow	告诉显示应用程序在元素内容超出指定的高度和宽度时应该怎么处理	visible(可见)或 scroll(滚动)	2
page	识别打印的页面	识别符或自动(默认时是自动)	2

属性	解释	可接受值	级别
page-break-after	指定页面(在打印输出时)如何在元素的后面断开	auto, always, avoid, left, right 或 inherit	2
page-break-before	指定页面(在打印输出时)如何在元素的前面断开	auto, always, avoid, left, right 或 inherit	2
page-break-inside	指定页面(在打印输出时)如何在元素的内部断开	auto, always, avoid, left, right 或 inherit	2
text-align	指定文本的对齐方式	left, right, center 或 justify	1,2
text-decoration	指定文本的其它标记形式(一般使用直线)	none, underline, overline, line-through 或 blink	1,2
text-indent	元素的第一行文本如何缩进	具体的尺寸或百分比	1,2
text-transform	允许文本的转换	none(默认时), capitalize, uppercase, lowercase 或 inherit	1,2
top	指定顶边相对于窗口最顶部的位置,用于在空间(一般是浏览器窗口中)定位元素的位置	通常是一个具体的尺寸,也可以是 auto 或 inherit	2
vertical-align	指定如何相对于基线对齐元素的内容,用于创建下标和上标	baseline(默认时), sub, super, top, text-top, middle, bottom, text-bottom 或位于基线之上的具体尺寸或百分比	1,2
visibility	让对象是否透明;与 display:none 不同,它并不妨碍子元素的出现	inherit, collapse, visible 或 hidden	2
whitespace	预先告诉浏览器显示元素内容中的空白,而不用使用 标记。nowrap 告诉浏览器除非遇到 , <P>或其它强制断行元素,否则不断行	normal, inherit, pre 或 nowrap	1,2
width	指定图像的宽度	具体的尺寸	1,2
word-spacing	指定文字之间的额外间距	具体的尺寸, normal 或 inherit	1,2
z-index	指定某个块的 z 层次,取值越高层次越靠前	一个整数值	2

注意

这个表并不完整。要想查看属性及其取值的完整列表,以及更详尽的解释。请访问 CSS 一级规范站点 <http://www.w3.org/TR/REC-CSS1> 和 CSS 二级规范站点 <http://www.w3.org/TR/REC-CSS2>。这些表格中的属性提供了一套使 XML 网页更加吸引人的基本工具。

样式表的应用规则向开发人员提供了巨大的灵活性,并在一定程度上向他们提供了创建自己的格式化标记的能力。样式表之所以称为“级联”,是因为你可以将多个样式表应用到同一份文档中。样式表使用倒序优先级,也就是说,最新的定义优于最先的定义。样式(及其样式表)可以几种方式使用。我们早已看到过包含在 HEAD 元素中的 STYLE 元素,如果你要创建影响整个文档(但只能是一个文档)的样式,会发现这一技术十分有用。如果你要将样式应用于多个文档,那么应该创建一个样式表文档,也就是一个包含能够通过 LINK 元素连接到 HTML 网页的样式信息的单独的文件。你还可以将样式应用于单个元素,从而能够使用样式象老式 FONT 标记提供的方式那样格式化文档。通过 STYLE 属性可以指定“内联”样式,为创建元素的标记内部的 CSS 属性设置值。

首先,让我们分别使用一些不同的技术将样式应用于单个元素。以这种方式格式化文本的最简单的途径,是创建一个包含所有样式信息的 SPAN 或 DIV (HTML 4.0 新增加的)元素。DIV 使用其后的空格创建类似于段落的结构,而 SPAN 格式化文本时不用创建行断点。使用 SPAN 和 DIV 可以避免其它 HTML 元素携带格式化信息的包袱。二者都可以独立地应用许多格式,使你能够轻易地创建事先可预知的结果。

注意

老式浏览器 (Microsoft Internet Explorer 或 Netscape 4.0 之前的版本) 不支持 SPAN 和 DIV,这使得很难将二者用于 CSS 的格式化。即使 4.0 版本的各浏览器在实现 DIV 元素时采取的方式也不尽相同--Microsoft 在 DIV 元素之后放置空格而 Netscape 则不然。如果你将 SPAN 和 DIV 元素之后放置空格而 Netscape 则不然。如果你将 SPAN 和 DIV 元素用作从 HTML 到 XML 的过渡性元素(如本书第 6 章中那样),那么请注意 SPAN 和 DIV 本身是比较新的。

```
<HTML>
<HEAD><TITLE>Formatting with CSS, atomized DIVs and SPANs</TITLE></HEAD>
<BODY BGCOLOR = #FFFFFF>
<DIV STYLE = "font-size:24pt; font-weight:bold">This is a big DIV, </DIV>
```

```
<P>This is a normal paragraph with an odd <SPAN STYLE="font-size: 14; font-weight: bold; font-style: oblique; color: red">SPAN< /SPAN> stuck in the middle of it. < /P>  
< /BODY>< /HTML>
```

它将产生出图 2-7 中所示的结果。

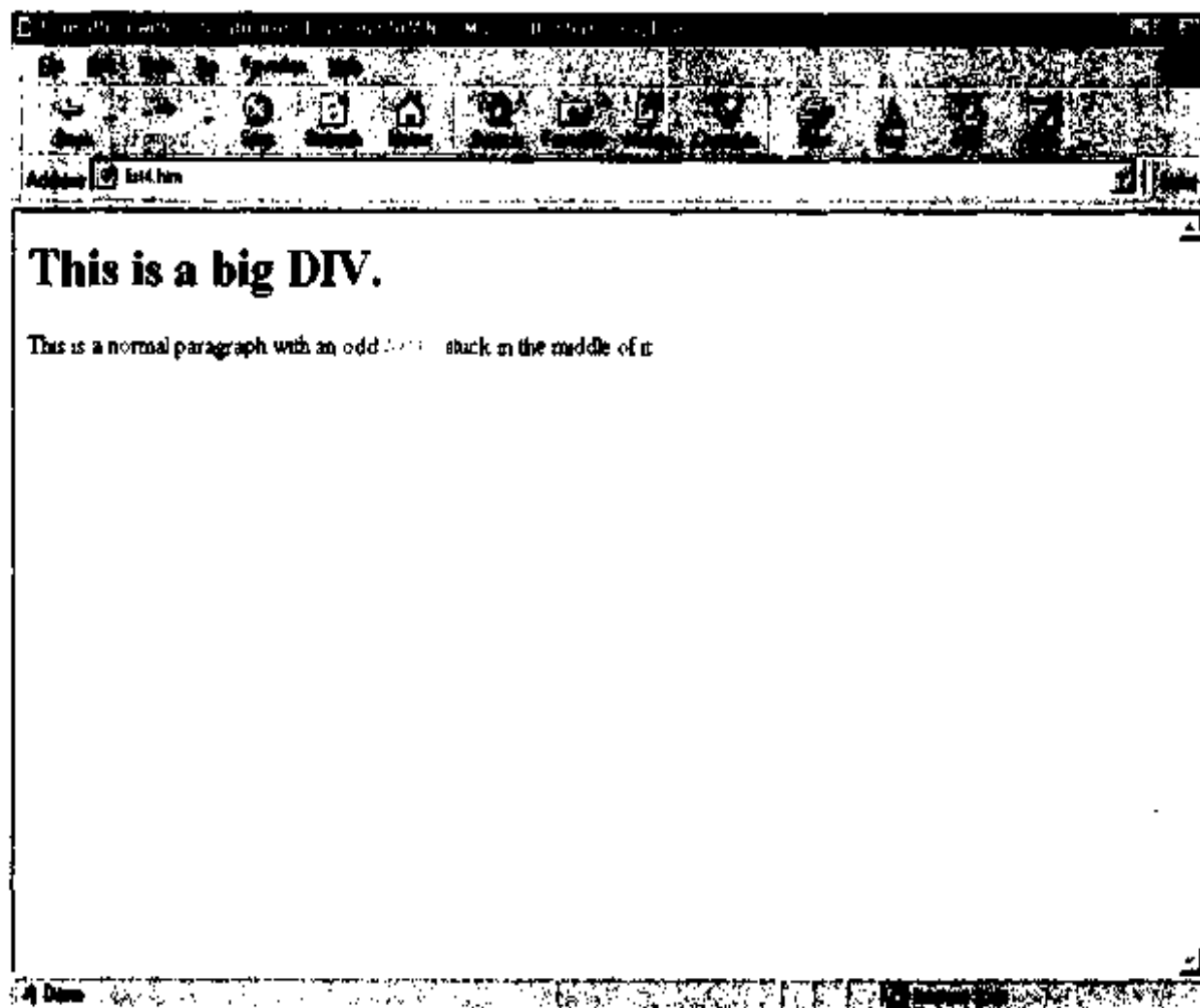


图 2-7 Internet Explorer 4.0 中的样式属性

虽然你可以用这种方式使用样式来达到格式化的目标,但你很快就会发现,当需要重复使用同一样式时这种方法就不灵了。使用单个样式多次产生同一结果,这很快也很方便,而且不需要破坏现有的标记。SPAN 和 DIV 标记有两个额外的属性 (ID 和 CLASS) 供你为同一格式的多个实体创建样式使用。ID 属性用来识别单个元素,因此所有的 ID 值都应该是唯一的(我们在后面将会看到,XML 中提供相应的实施办法)。CLASS 属性用于识别多个元素组成的元素组。五个不同的 DIV 可以使用相同的 CLASS 属性值,从而以同样的方式格式化。标记可以同时拥有 CLASS 值和 ID 值,ID 特有的任一格式都将优先于 CLASS 的格式。

注意



所有的 HTML 元素都可以拥有 ID 和 CLASS 属性,通常情况下 Internet Explorer 4.0 和 Netscape Communicator 4.0 都可以支持它们。作为示例,我们只使用 SPAN 和 DIV。XML 元素可以使用 ID 属性,但要使用 CLASS 属性可能得等到新的 CSS 规范的出现,新版本的 CSS 规范中必须利用名称空格(详见第五章)来创建所有文档中都可以识别的 CLASS 属性。

我们的下一个示例将使用 STYLE 元素创建一个文档,STYLE 元素控制该文档的 BODY 中的几个 SPAN 和 DIV 元素。请注意 CSS 使用 CLASS 和 ID 的方式不同。虽然 CSS 允许你这样做,但你最好不要在文档中使用相同的 CLASS 和 ID 名称。

```
<HTML>
<HEAD><TITLE>Formatting with CSS, using CLASS and ID</TITLE>
<STYLE>
DIV.bold {font-size:24pt; font-weight:bold }
DIV.italic {font-size:24pt; font-style:italic }
SPAN.test {font-weight:bold}
SPAN#freaky {color: green; font-size:90pt; font-style:italic}
</STYLE>
</HEAD>
<BODY BGCOLOR = #FFFFFF>
<DIV CLASS="bold">This is a big bold DIV. </DIV>
<DIV CLASS="italic">This is a big italic DIV. </DIV>
<P>This is a normal paragraph with an odd <SPAN ID="freaky">SPAN</SPAN> stuck in the
middle of it, as well as a <SPAN CLASS="test">bold</SPAN> bump and another <SPAN
CLASS="test">bold</SPAN> bump in the middle of it. </P>
<DIV CLASS="style1">This is another big bold DIV. </DIV>
</BODY></HTML>
```

这个例子比较容易编写,它产生的结果如图 2-8 所示。

虽然 STYLE 元素可以帮助你同一文档中重复使用同样的格式,但用这种方法管理大量文档的格式仍旧十分困难。根据非常具体的规则建造的 Web 站点在每次设计人员想要改变字型或颜色时都需要大量的微调工作。要解决这一站点级问题,应当借助于样式表。样式表使你能够大规模管理样式。样式表不是直接把样式信息放在文档中,而是使用选择符和伪元素来识别位于文档树中具体位置上的特定元素的样式。下面的表 2-2 完整地列出一 CSS 选择符和伪类别,它们执行的功能(不论在 HTML 还是 XML 中)与选择符类似。

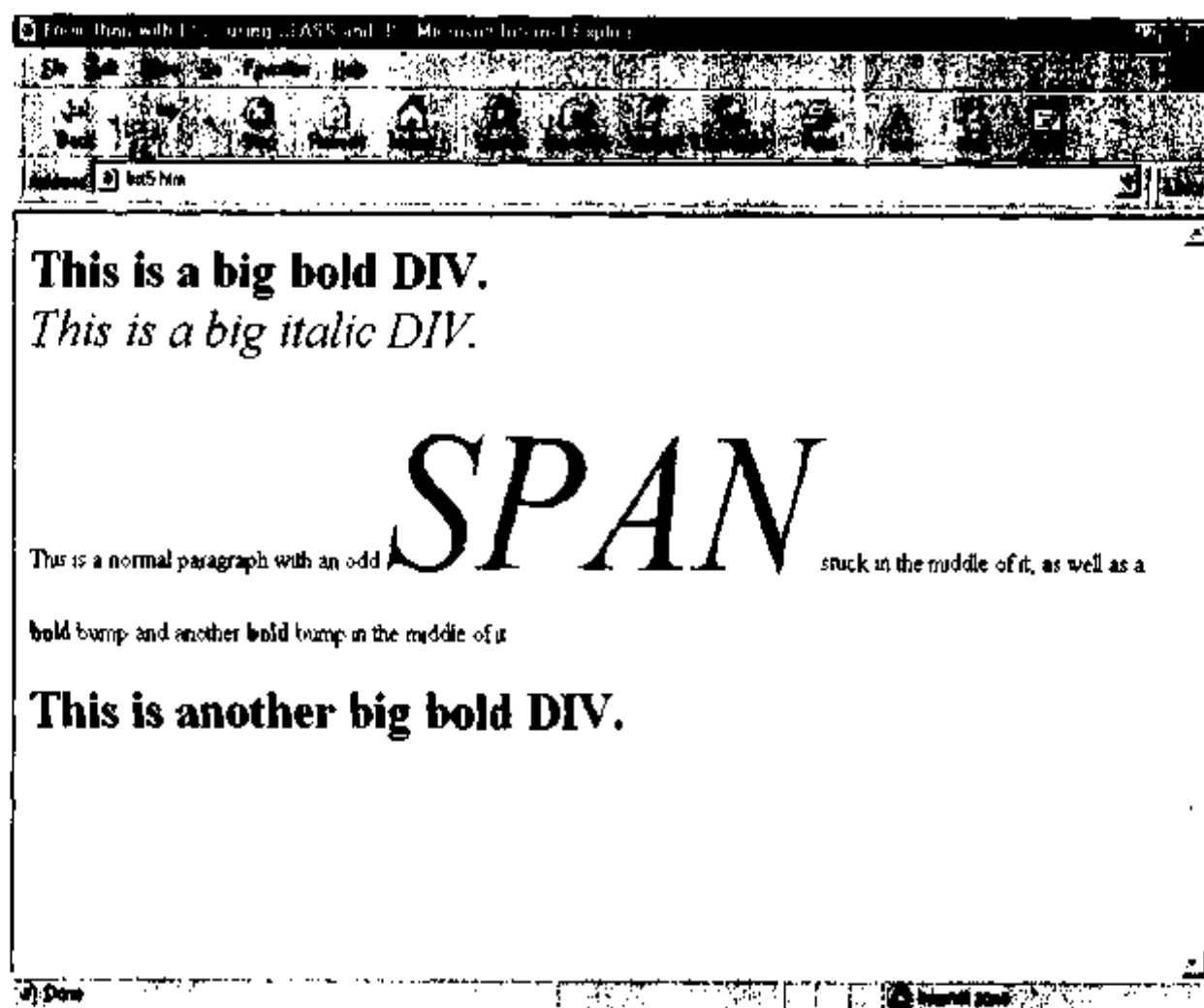


图 2-8 Internet Explorer 4.0 中的类别(class)和 ID 样式

表 2-2 级联样式表选择符

选择符	意义	级别
元素名称	选择具有该元素名称的每个元素	1,2
元素名称 1 元素名称 2	选择具有元素名称 2 的从元素名称 1 继承下来(未必是元素 1 的子元素)所有元素	1,2
元素名称 1,元素名称 2	选择具有列表中所列选择符名称的所有元素[,元素名称 3...]	1,2
元素名称 1>元素名称 2	选择自具有元素名称 1 的元素直接继承来的(不只是它的后裔)具有元素名称 2 的子元素	2
元素名称 1+元素名称 2	选择在文档树中与元素名称 2 系同属兄弟的具有元素名称 1 的元素	2

选择符	意义	级别
[属性名称]	选择为属性名称指定取值(不管是什么值)的那些元素	2
[属性名称="属性值"]	选择为属性名称指定取值为“属性值”的那些元素	2
[属性名称="属性值"]	选择“属性值”包含在“属性名称”的记号列表中(以空格分隔开)的那些元素	2
[属性名称 ="属性值"]	选择属性为“属性名称”,其值以“属性值”开头后跟以连字号的元素。(用于选择特定的语言类型。)	2
.类别名称	选择包含名称为 class 或 CLASS 的属性且取值为“属性值”的元素	1,2(仅限 HTML)
# ID 值	选择包含类型为 ID(在 HTML 中,名称为 ID)的属性且其值为“属性值”的元素	1,2
*(星号)	选择所有的元素,不管它们的名称或内容为何。星号可以用来代替选择符内任意位置上的元素名称	2
:first-child(第一个子元素)	选择在父元素内部第一个出现的子元素	2
:link(链接)	选择所代表的超文本链接还未被用户访问过的那些元素	1,2
:visited(以被访问过的)	选择所代表的超文本链接已经被用户访问过的那些元素	1,2
:active(激活的)	选择用户正在激活的超文本链接	1,2
:hover(悬停)		
:focus(焦点)	选择在特定情形下用户正在与之交互(通常在浏览器)的元素	2

选择符	意义	级别
:lang(语言)	选择指定语言的元素内容。CSS2 规范建议将它与 xml:lang 属性结合使用以识别元素语言内容	2

交叉参考



这些选择符的用法将在第六章中进行更细致的讨论。CSS 不难走近,但要完完全全地论述其所有的可能性则需要一本专门的书籍。这儿推荐一本这样的书:《Designing for the Web》,作者是 Hakon Lie 和 Bert Bos, Addison-Wesley 出版社 1997 年版。本书著者是级联样式表规范的编辑人员。

建造样式表与建造 STYLE 元素非常相似,只是所有的样式信息都放入一个独立的文件(该文件连接到所有需要它的文档)中。CSS 文件看上去非常象我们以前所做过的:

```
DIV.style1 {font-size:24pt; font-weight:bold;}
DIV.style2 {font-size:24pt; font-style:italic;}
SPAN.test {font-weight:bold;}
SPAN#freaky {color:green; font-size:90pt; font-style:italic;}
```

在保存好这个样式表之后(不妨将它保存为 demo.css),可以将它应用到你的网页。不是使用 STYLE 元素,而要使用 LINK 元素。LINK 元素早在 HTML 2.0 中就已经出现了,但直近才得以广泛应用。我们将使用它把样式表连接到前面的 HTML 代码:

```
<HTML>
<HEAD><TITLE>Formatting with CSS, using DIVs, SPANs, and LINKs</TITLE>
<LINK REL=stylesheet HREF="demo.css" TYPE="text/css">
</HEAD>
<BODY BGCOLOR=#FFFFFF>
<DIV CLASS="style1">This is a big bold DIV.</DIV>
<DIV CLASS="style2">This is a big italic DIV.</DIV>
<P>This is a normal paragraph with an odd <SPAN ID="freaky">SPAN</SPAN> stuck in the
middle of it, as well as a <SPAN CLASS="test">bold</SPAN> bump and another <SPAN
CLASS="test">bold</SPAN> bump in the middle of it.</P>
<DIV CLASS="style1">This is another big bold DIV.</DIV>
</BODY></HTML>
```

必须使用 LINK 元素的 REL 性来指定这是一个样式表。HREF 属性仅仅为浏

览器提供样式表文件所在的位置。TYPE 指定文档的 MIME 类型,使用 CSS 的样式表是“text /css”类型。如图 2-9 中所显示的,该样式表的结果看上去跟在 HEAD 元素中包含进样式信息的结果相同。

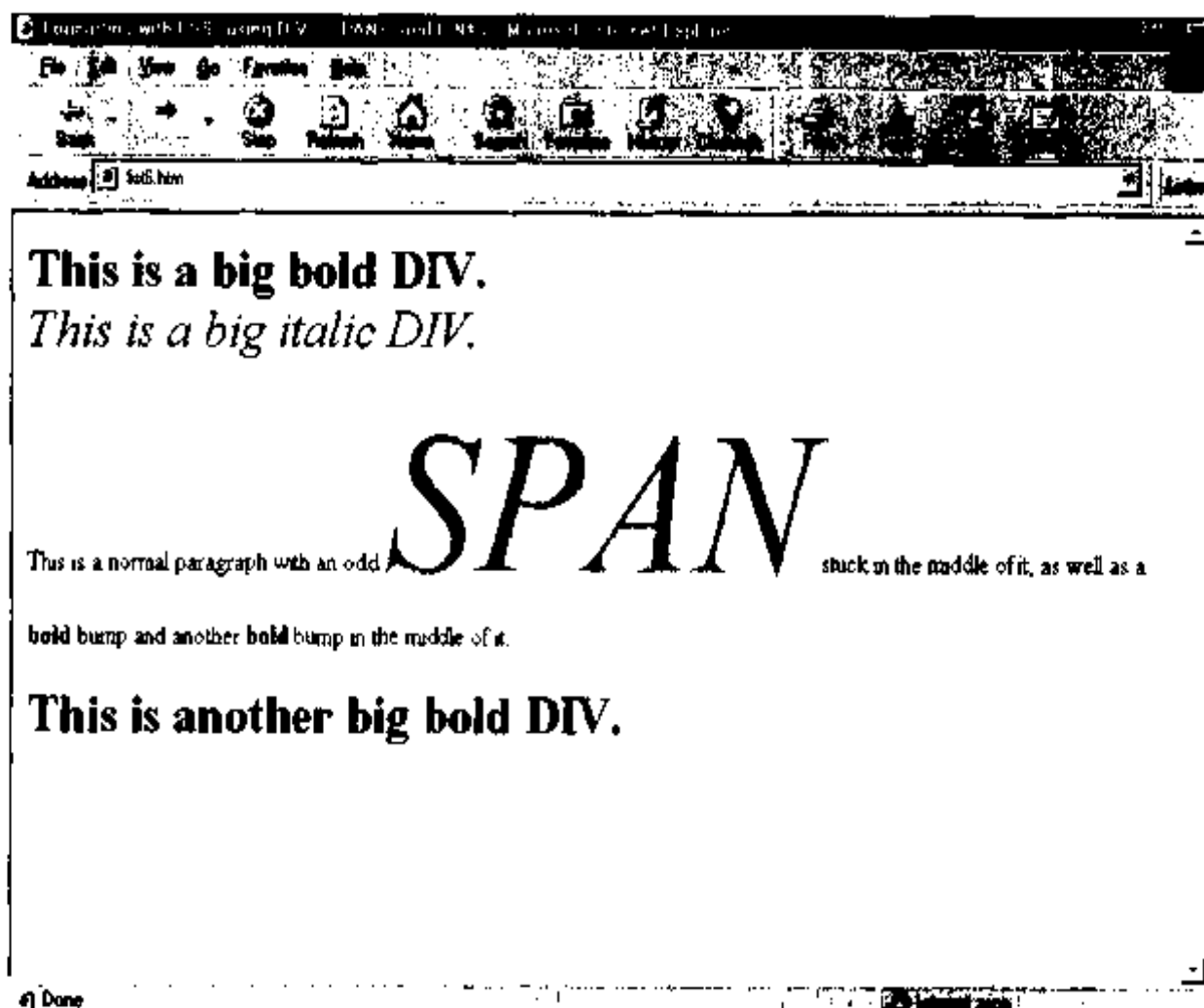


图 2-9 使用单独的样式表的 CSS

你还可以提供另外两个属性。一个是 MEDIA 属性,它帮助浏览器决定为哪个输出应用程序使用什么样式表。一般说来,在向文档应用多个样式表时,要提供这一信息。可接受的取值有 Screen(屏幕)、Print(打印)、Projection(投影)、Braille(盲文)、aural(音响)和 all(全部)。MEDIA 属性对于创建向文档中包含进或从中移走图例及其它非文本性材料的样式表非常有用。另一个属性是 TITLE,它使你给样式表赋以与用户友好的名称,以便于用户从多个样式表中进行选择。例如,你可以给样式表冠以“Large-print”、“normal”和“microtype”等名称。

关于样式表需要记住的另一件重要的事情是它们是级联的。尽管“级联”一词可能会让人形象地想象出瀑布和湍流的样子,但它实际上只是意味着一个 HTML 元素可以接受来自与之最接近的样式的格式。最接近的样式直接嵌入在元素内部,作为一个 STYLE 属性,它重载其它所有的属性。按优先级排列的下一个样式是文档的 HEAD 元素中的 STYLE 元素。任何通过 LINK 元素的加载的样式都处于最低的优先级,但浏览器可能会允许用户从连接到文档的多个样式中进行选择。这样,用户可以根据自己的需要选择大型打印页,或者选择隐藏文档中所有图像的样式仅显示未加装饰好文本。

注意

你也可以使用@IMPORT URL(stylesheetURL)语法从(STYLE)标记内部连接样式表。但应该优先使用 LINK,因为它的灵活性更大。

实际上,公司或组织可以创建供整个组使用的样式表,但是单个网页能够重载整个站点的外观。例如,一个大公司可以为其所有文档创建一个标准的样式表,但又允许各部门创建能够重载公司标准的自己的样式表。然后单个网页的设计人员就可以(当然不要冒犯公司)将公司和部门的样式表应用到他们的文档中,并使用 STYLE 元素和属性进行改动。

尽管你可以使用样式表随意地格式化你的文档,但我们不提倡这么做。CSS 除取代 FONT 元素之外还能做其它许多事情,但它要求合作和协同。成功地使用 CLASS 属性意味着创建出供多个网页使用的标准,使设计人员(或者一组设计人员)能够轻松地创建规范,这些规范不需要每次有人需要一个新的页面时,都得从头重新建造。ID 标记能够从类似的协作中受益,帮助设计人员实现一页一个 ID 的模式,从而突出最近的新内容或创建一种新的设计。尽管 CSS 是完全关于格式的,但它也强调结构,以便把格式放入可重复的被控制环境中,这一结构可能与小型站点无关紧要,但可以使大型站点的管理方便许多。

至此,你可能开始奇怪为什么还需要 CSS 之外的东西来创建自己的标记集和超越 CSS。即使还没有见识 CSS 能做的所有事情,但请相信它的确能使你创建出所需要的几乎任何一种格式。你可以创建字符和段落级的格式,控制空白空间,以像素为单位指定元素的 x 和 y 位置,将文档转换为想要的颜色,还可以为段落的第一个字母或第一行创建特殊的效果。就重建“所见即所得”布局的精彩世界而言,CSS 足以满足需要。但不幸的是,通过 CLASS 和 ID 创建自己的标记实际上仍旧是关于格式化的,而不是面向数据,这正是必须使用 XML 的原因。因为 XML 能够让我们将 CSS 的功能与管理数据的强大结构顺利地结合起来,帮助我们创建出既好看又好用的文档。

注意

级联样式表仍在不断发展壮大,并在最初的 HTML 之外的 W3C 的其它项目中得到了应用。这些项目包括同步化多媒体一体化语言(SMIL)和结构化向量图形(SVG)等。欲了解最新的信息,可以访问站点 <http://www.w3.org/Style/CSS>。W3C 的邮件列表 www-style@w3.org(可以给 www-style-request@w3.org 发信征订)和新闻组 comp.infosystems.www.authoring.stylesheets 是讨论 CSS 问题及其实现的论坛。

XSL

级联样式表是通过给现有的文档结构注释以显示信息而工作的。可扩展样式语言(XSL)采取的方法更加引人注目。它将文档转换到一个新的文档树中,该文档树由格式对象组成,然后应用程序就可以将文档显示出来。XSL 对其样式表使用 XML 语法,而且以 XML 为中心。XSL 可以用来产生 HTML(如果你需要向浏览器中不存在 XML 支持的用户提交 XML 信息,会发现这一点很方便),但是 XSL 工作组对 HTML 方式考虑的很少。

当心

这部分有关 XSL 的所有信息都有可能变化,而且还可能发生很大的变化。你在这里读到的信息是从 1998 年 12 月 16 日的工作草案为基础的,并不是 W3C 的最终推荐标准。要想了解关于 XSL 的最新消息,请访问站点 <http://www.w3.org/Style/XSL/>。也可以使用讨论 XSL 的邮件列表 <http://www.mulberrytech.com/xsl/xsl-list> 获取这方面的信息。

XSL 大部分是从文档样式语义和规范语言(DSSSL)演变而来的,后者经常用来为 SGML 文档提供格式化信息。DSSSL 是一个巨大的标准,能够为所有类型的文档提供精确的格式。但学习和实现 DSSSL 却是一项庞大的任务。正如 XML 是 Web 的 SGML 一样,XSL 可以称得上是 Web 的 DSSSL(还有一个叫做 DSSSL-O,它是 DSSSL 的一级在线简本,可以与 XSL 抗争)。DSSSL 及由它演变来的其它语言使用 SGML 而不是 XML,它们的语法也与 XML 不相兼容。XSL 开发人员试图建造一种象 DSSSL 一样强大(或与之相近)的样式语言,而且这种样式语言使用的是 XML 语法,并且很简单,以使其在 Web 上得以广泛应用。至于这两种标准能够达到怎样的兼容程度还要等到最终推荐标准发布时才能定论。与 XML 不同,XML 明确地把与 SGML 兼容作为其目标,而 XSL 则不受向后兼容的限制。

交叉参考

要想获得对 DSSSL 的全面了解,请参考 Paul Prescod 的“DSSSL 介绍”,其所在站点是 <http://cito.uwaterloo.ca:80/papresco/dsssl/tutorial.html>。要想了解 DSSSL-O 方面的信息,可以访问站点 <http://sunsite.unc.edu/pub/sun-info/standards/dsssl/dsssl-o/dsssl-o.htm>。

XSL 有两个关键工具组成。一个是转换引擎(“树状结构”),它将原始文档树转

换成能够显示的文档树;另一个是格式化符号集,它描述布局、排字等信息的显示。到现在为止,大多数实现(包括 Microsoft 的 Internet Explorer5 Beta 版)都集中于 XSL 的转换引擎,而不是它的格式化对象。因此,当前的 XSL 使用者大部分是从一个 XML 符号集转换到另一个 XML 符号集的开发人员,而不是排布文档模板的设计人员。将 XSL 分成两个独立标准(一个用于转换,另一个用于格式化)的提议已经在 XSL 邮件列表中出现。但工作组对这些提议的反应尚未出现——在编写本书时,XSL 仍旧是由两个不同的部分组成的。下文将分别讨论这两个组成部分。

注意

如果你现在就想知道 XSL 的细节,请继续阅读。如果你想先了解 XML 的更多信息,可以直接跳到下一章,在做好足够的准备后再返回来阅读 XSL。

交叉参考



要想了解用于处理 XSL 的工具列表,请访问以下站点:<http://www.xmlsoftware.com/>。

树状结构(转换)

当 XSL 处理器被应用程序唤起时,它加载一个样式表,该样式表包含将原始文档结构转换成新的文档结构的规则。这些规则表示为一组模板,这组模板应用于原始文档中的各种元素并在原始文档的基础上产生一个新的文档。这种方法使得能够非常容易地做一些事情,如重新定做文档、隐藏和显示有关或无关的信息、将表格转换为图形(使用向量图形标记语言,如 W3C 的结构化向量图形,即 SVG)或将 XML 文档转换成 HTML 以在较旧版本的浏览器中显示(我们将在示例中做这件事情)。

XSL 使用“方案”来识别需要处理的文档树的各个部分,使用模板规则来描述应该进行的处理。在某些方面,XSL 象是一门编辑语言,例如它具有测试相等与否的结构,并能根据处理的结果执行其它处理。但在总的方法,XML 与编程语言完全不同(而且,最起码到现在为止,对于 Microsoft、Inso 和 ArborText 公司在其原始草案中提议的 ECMAScript,还没有出现任何支持)。方案有些象 CSS 选择符,它识别将应用某个具体模板的元素,但它使用的语法跟 CSS 选择符完全不一样。表 2-3 包含了 XSL 方案的语法例子,这些例子是从 1998 年 12 月 16 日的工作草案中摘录来的。

表 2-3 XSL 方案的语法例子

方案	意义
elementName	匹配名称为 elementName 的所有元素。
elementName1 elementName2	匹配名称为 elementName1 或 elementName2 的所有元素。
elementName1 /elementName2	匹配名称为 elementName2 并且是 elementName1 的子元素(直接包含在 elementName1 的内部)的所有元素。
elementName1 //elementName2	匹配名称为 elementName2 并且是 elementName1 的后代(包含在 elementName1 内部的任何地方)的所有元素。
@attributeName	匹配为属性 attributeName 赋值的所有元素。
*	匹配所有的元素。
@ *	匹配所有的属性。
elementName1[elementName2]	匹配名称为 elementName1 并且包含子元素 elementName2 的所有元素。
elementName[@attributeName]	匹配名称为 elementName 并且具有属性 attributeName 的所有元素。
elementName[@attributeName="value"]	匹配名称为 elementName 并为属性 attributeName 赋以值 value 的所有元素。
,	匹配当前正在处理的节点(通常是一个元素)。
"	匹配当前正接受处理的节点的父亲。
comment()	匹配当前节点中的所有批注。

然后这些方案与文档进行比较,并根据方案的匹配和模板规则的指定进行转换。模板规则的解释最好使用实例,下面我们就从一个非常简单的原始文档开始。

```
<? xml version="1.0"? >
<? xml-stylesheet href="test.xsl" type="text/xsl"? >
<test>this is a test</test>
```

交叉参考



看上去很奇怪的结构<? xml-stylesheet href="test.xsl" type="text/xsl"? >是一条 XML 处理指令,第五章中将对此加以论述。

为将此文档转化成一个可视的文档,我们将应用下面的 XSL 样式表。

```
<? xml version="1.0"? >
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template><xsl:apply-templates /></xsl:template>
<xsl:template match="textnode()"><xsl:value-of /></xsl:template>

<xsl:template match="*" />

<html>
<body>
<xsl:for-each match="test">
  <p><b>
    <xsl:apply-templates />
  </b></p>
</xsl:for-each>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

注意

你也会觉得 XML 太古怪。请不要因为看不懂这里使用的 XML 而着急,我们将在第五章讨论名称(作用域)空间,它将对元素名称之前的 xsl:和 xmlns 属性作出解释。

它然后产生结果:

```
<html>
<body>
  <p><b>
this is a test
  </b></p>
</body>
</html>
```

Internet Explorer 5.0 显示这些结果,如图 2-10 所示。

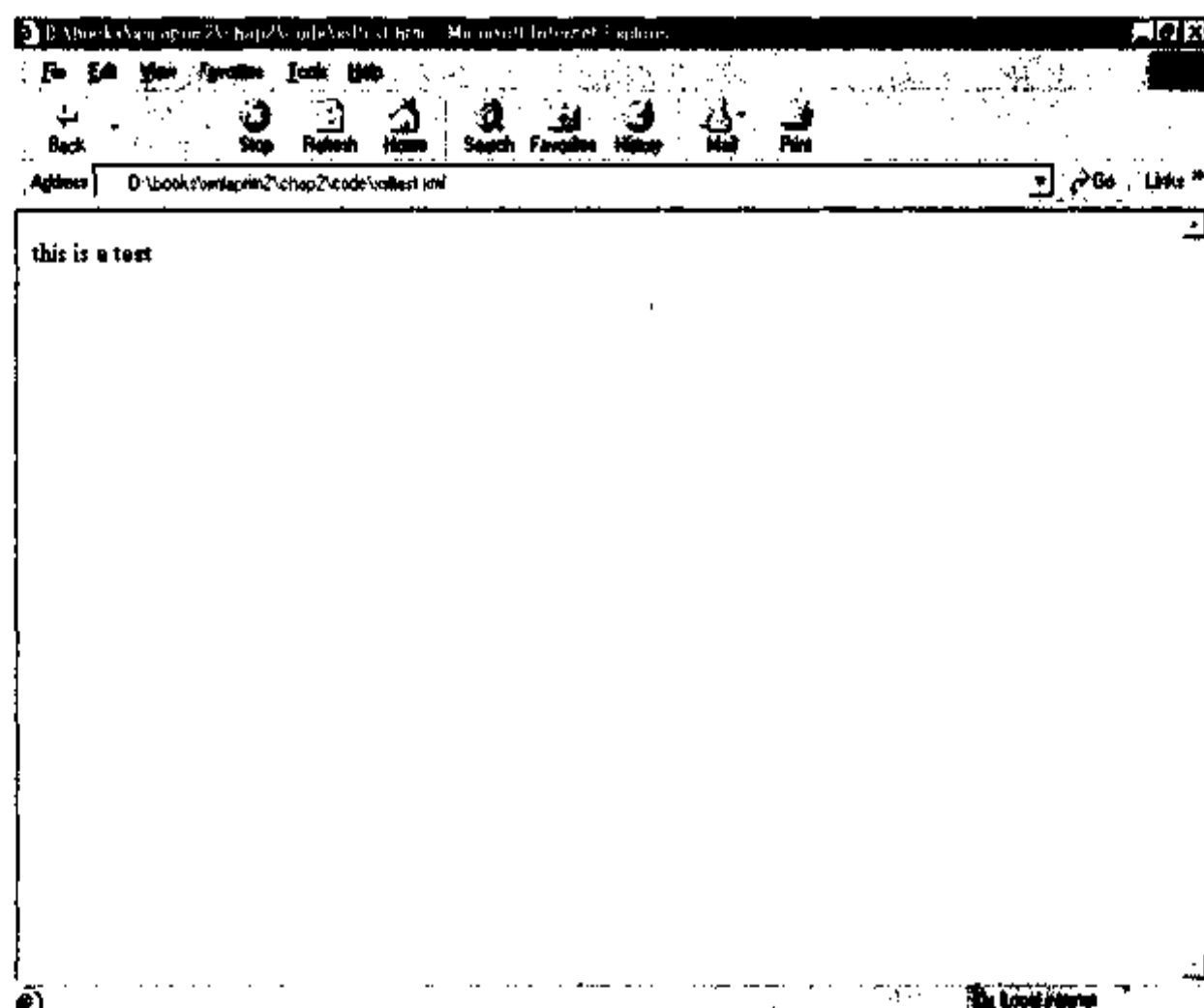


图 2-10 在 Internet Explorer 5.0 中使用 XML 和 XSL 产生简单的 HTML

XSL 包含一大套规则,把这些规则结合起来使用可以把属性值转化成元素文本、以不同的顺序重组内容(包括排序)、添加编号及其它以文档为中心的信息、为以后重新使用建造宏、对不同的条件做出反应,可能最后还能使用前面提到的 ECMAScript 对 XSL 本身进行扩展。

格式化

XSL 不对文档树作注释,而是由 XSL 的树状结构建造一个新的格式对象文档,这些格式化对象具有的属性进一步定义它们如何显示。XSL 的格式化对象是以显示为中心的 XML 符号集,虽然在搜索能力和重新使用方面不比 HTML 强,但它们是纯粹作为显示前台而设计出来的。下面给出 1998 年 12 月 16 日工作草案中的格式化对象清单:

注意

格式化对象很容易改变,之所以在这里将它们列出来是为了让你了解 XSL 具有的能力,而其最终版本很可能与此有很大的不同。

fo: basic-page-se- quence (基本页 面序列)	fo: block(块)	fo: character (字 符)	fo: display- graphic (显示图 形)
fo: display-link (显示链接)	fo: display-rule (显示规则)	fo: display -sequence (显示 序列)	fo: inline-graphic (内联图形)
fo: inline-link(内 联链接)	fo: inline-rule (内联规则)	fo: inline-se- quence (内联序 列)	fo: link-end-loca- tor(链接结束定 位器)
fo: list-block (列 表块)	fo: list-item(列 表项)	fo: list-item-body (列表项主体)	fo: list-item-label (列表项标签)
fo: page-number (页编号)	fo: queue (队列)	fo: simple-page- master (简单页 面管理员)	

以上每个格式化对象都可以使用格式化属性(与 CSS 的属性非常相似)作进一步的定义。格式化属性包含从简单的颜色、字体、字体尺寸和线高,直到较高级的行距优先级、换码空格结尾、连字符号和内容旋转。

XSL 的未来

作为转换工具和强大的格式化引擎,XSL 具有光明的前景。它在 Web 上价值很大程度上取决于浏览器销售商为其提供多大的支持。虽然 Microsoft 公司在其 Internet Explorer 5 上积极地倡导 XSL 转换引擎(它也可以通过 Active Server Page 用于服务器一方),但 Netscape 对此来表现出很大兴趣,而且也没有出现其它竞争商。在这样的环境下 XSL 能否实现或取代 CSS 还不好回答。但在服务器端,各种各样的 XSL 转换工具可能会大大有利于从 HTML 到 XSL 的过渡,以及为不理解 XM 斩浏览器将“陌生”的 XML 转化成“友好”的 HTML(或 HTML+CSS)。XSL 还可能在许多服务器端处理工具中扮演重要的角色(从 Active Server Page 到 Java 小服务及 CGI 编程),为程序员创建模板和修改不需要具体编码的文档结构提供一般性工具。

交叉参考



要想了解与 XSL 方法形成鲜明对比的有趣的观点,请见 Hakon Wium Lie 的“格式化对象的坏处”,其所在的站点是:<http://www.operasoftware.com/people/how-come/1999/loch.html>。

第三章

简单的 XML:建造结构

现在我们知道有一套完整的格式化工具可供我们创建出具有理想外观的文档,因此这里先暂时不用考虑这些文档是如何显示的。而使用 XML 则不同,它要求设计人员检查他们的文档是如何建造的,而不是它们是如何被格式化的。

如果以前曾经在英语课堂上画过英语句子的图解,那么你已经受过这方面的训练,尽管这种训练的水平与现在对设计人员的要求不同。不是在句子一级检查其结构,我们将在文档和数据层次检查其结构,识别出标题、节、分节、段落、列表、图形、项目编号和项目描述,而在句子图解中我们要识别的是名词、动词和介词。这两件事情的确非常相象。XML 向开发人员提供使用固有框架创建文档的机会,框架可使重复创建同样的结果变得非常容易,因为它能够携带有用的和可重复使用的数据。

浏览器和分析程序

Web 之所以能够飞速成长壮大,有几个因素是缺一不可的。一是 HTML 的简单性,二是 HTTP 服务器(即 Web 服务器)设置的相对容易性,三是浏览器的迅速普及,而其中 Netscape 和 Microsoft 的 Internet Explorer 是最为杰出的。即使最早期的浏览器(1991 年在欧洲核研究委员会创建的)其宗旨也是使用户能够快速地访问文档,让他们能够不费周折地轻易地从一个文档转移到另一个文档。这是 HTML 对 SGML 的最大突破之一,它象将标记用于格式化和结构一样具有重要意义。HTML 浏览器不用担心检查文档语法,而是对文档进行分析(这等于是用计算机来阅读文档)然后显示结果。但这些结果过去并不总是很完美,现在也依然这样。在一个格式化的大型文档中查找一个漏掉的结束标记是很困难的,设计人员必须将自己的代码与某个特定的浏览产生的结果逐一对照。浏览器总是尽量接受尽可能多的代码,但提交的结果往往因此而丑陋。这种丑陋性不一定显示在屏幕上,而有时存在于创建结果的代码中,这些信息很难再次使用。

注意

在浏览器发展的主流之外也有一些浏览器在一定程度上检验 HTML。例如, W3C 试验的一种浏览器, 名字叫 Arena, 它就有一个选项可以指出不完整的标记。但是市场引导的方向显然是将所有东西都显示在屏幕上, 不管显示的结果有多么怪异, 都不向访问者提供出错信息。

SGML 总是把分析文档看得比提交(或显示)文档重要。分析程序接收一个大的文件(通常是文本文件), 然后将它分解成各个组成部分。在 SGML 中这意味着分析程序检查一个文件, 将它与 DTD 进行对照, 把文档分解为各个组成部分, 并按照定义检验该文档。尽管不好确定 SGML 为什么会不完整, 但“破碎”的 SGML 总是很容易被查出来。因为 SGML 分析程序通常检验文件, 并由于 SGML 不直接关心格式, 从而管理 SGML 的程序以确保 SGML 文档的代码正确, 至于其显示方面则没有花费那么多心思。所以标记错误可能会导致整个文档都不被文档管理系统的高级工具接受。

浏览器将分析引擎和显示引擎结合在一起, 尽管它们通常为分析工作采用另外一种不同的模型。HTML 浏览器不检验 HTML 文件, 甚至连拿它们的内部标准对 HTML 文件进行检验这一步工作都不做。相反地, 它们总是分析该文件并根据自己的理解尽可能地对标记进行显示。如果不理解某个标记, 它们会将该标记忽略掉。如果有某个结束标记漏掉了, 它们会尽其所能地推测这个标记最有可能出现的位置。属性可能起作用, 也可能不起作用; 这取决于浏览器是否理解该属性。这种不加控制的模型使业余爱好者能够很容易地发布 Web 网页, 从而极大地增加了网页创作者的数目。其缺点是导致出现了一大批编写得很差的 HTML 网页的产生。这一状况使得浏览器开发商能够趁机给 HTML 语言添加他们自己的扩展, 因为一个公司的私自扩展反正不会给另一个公司的浏览器带来什么麻烦。但是因为有一些标记被忽略掉, 而使得网页看上去不是太好。尽管网页可能会在一个浏览器中看上去很棒, 但当用户在另一个“错误”的浏览器打开它们时有一些信息可能会丢失或显示得很怪异。想要使所有东西在任何浏览器中都很完美的设计人员肯定会失望, 但至少有一些共同的特征可以使用。

XML 要求遵循标准, 这一点可能使 HTML 开发人员感到震惊。XML 分析程序(它位于显示已分析信息的浏览器之后)远比 HTML 浏览器挑剔语法和结构。通过要求网页创建正确使用语法和结构, 而不是强迫浏览器推测文档中应该是什么东西, XML 使得分析程序不论在性能还是稳定性方面都更容易实现。XML 文档每次的分析结果都是一致的(正如我们将要看到的, XML 文档因分析程序类型的不同其分析结构会有一些变化, 但这些变化是很有限的)。

因为分析程序不需要花时间重建不完整的文档,所以它们能比同类 HTML 更有效地执行其任务。它们能全力以赴地根据已经包含在文档中的那个树结构建造出树结构来,而不用在信息流中的混合结构的基础上进行显示。XML 标准是对处理器(分析程序)而言的,而不是针对浏览器。因为 XML 的发展方向将致力于机器可读的数据应用,而不是具有丰富图形的 Web 网页。任何类型的应用都可以在分析程序的上面建造,浏览器只是 XML 的一个小的(尽管很重要)组成部分。浏览仍旧极其重要,它为 XML 工作人员提供用于阅读信息的友好工具,但对更大的项目来说它不过是一个窗口。

Netscape 和 Microsoft 两大公司正在将 XML 集成到它们的浏览器中,但浏览器将只是 XML 众多工具中的一个小的组成部分。XML 还允许大量 HTML 样式的形式自由的开发,但是它对规则的要求更加严格,这一点我们将会在后面看到。从长远看来,如果使用最强大的工具建造结构,开发人员肯定会收到最满意的结果。但并非所有的应用都需要使用这么强大的工具,完全有可能以检验分析程序和浏览器都能理解的格式轻易地创建出文档。

建造块

XML 使用的结构与 HTML 非常相似,这一点并不奇怪,因为它们共同的根源都是 SGML。在这些语法结构之下是定义文档结构的语言,它使 XML 具备 HTML 所没有的功能。我们将首先检查 HTML 表面上的结构,然后再由此深化,直到能够开始建造一些简单的 XML 文档。

注意

- 文章创建的文档在不进行检验的分析程序中能够很好地工作,这种分析程序不检查文档的总体结构。第五章将论述在检验分析程序环境中工作时需要的额外工具。较小的应用和试验性开发更有可能使用非检验分析程序,而大型的文档管理及其它标准的开发活动则需要检验。

元素和标记

虽然 HTML 设计人员经常交换使用“元素”和“标记”这两个术语,但二者之间的差异是很大的。标记是用于标示目的的标签,如<P>、、等。元素则是这些标记的应用形式。例如,一个段落元素可能是下面的样子:

```
<P> This is a <EM>sample< /EM> paragraph element. It includes several other elements, including an emphasis (EM) element that includes the word 'sample' and a <B>bold< /B> ele-
```

ment that includes the word bold. < /P>

这段文本包含六个标记(三个开端标记,三个产端标记),但只有三个元素——段落元素及其包含的另外两个元素。自 HTML 第一次出现时就使设计人员头疼的嵌套规则就是依赖于标记和元素之间的这种差异的。以下代码尽管在有些浏览器中能得到正确的处理,但它是非法的:

```
<B>This is bold. <I>This is bold italic. < /B>This is italic. < /I>
```

在我的文字处理程序中可以侥幸地成功使用这段代码。我不需要将文本转换成正常状态就可以添加其它格式,格式被理解为附加物,能够彼此互相叠加。如果<I>和标记只是用作格式,这段代码会以相同的方式工作。但它在 HTML(或 XML、SGML)中却不是这样。这样的代码试图创建的元素的始端标记和末端标记相互交叠,如图 3-1 所示。

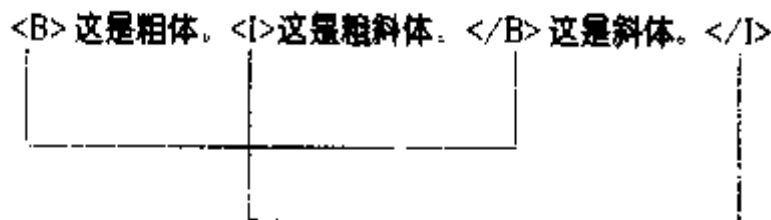


图 3-1 交叠元素是被禁止使用的

交叠元素会产生各种模棱两可的结果,特别是当内容(和格式)更加复杂时,在 HTML 中产生如图 3-2 中所示结果的正确方法是使用以下代码:

```
<B>This is bold. < /B> <I><B>This is bold italic. < /B>< /I> <I>This is italic. < /I>
```

或

```
<B>This is bold. < /B> <I><B>This is bold italic. < /B> This is italic. < /I>
```

或

```
<B>This is bold. <I>This is bold italic. < /I>< /B> <I>This is italic. < /I>
```

这三种方法使用的标记较多,但始端标记和末端的标记不会交叠。第一种方法使用的标记最多,但它是创建此类文本的最安全的方法,特别是当你打算剪切、粘贴或将它放到别的地方使用时。另外两种方法比第一种方法使用的标记少,它们利用了 HTML 能够嵌套标记并允许元素接受包围它们的元素的格式这一功能。二者唯一的区别在于一个是将倾斜元素嵌套于其前面的加粗元素中产生加粗倾斜元素,另一个是将加粗元素嵌套于紧跟其后的倾斜元素中产生加粗倾斜元素。图 3-3 显示了这三种方法在嵌套方式上的变化。

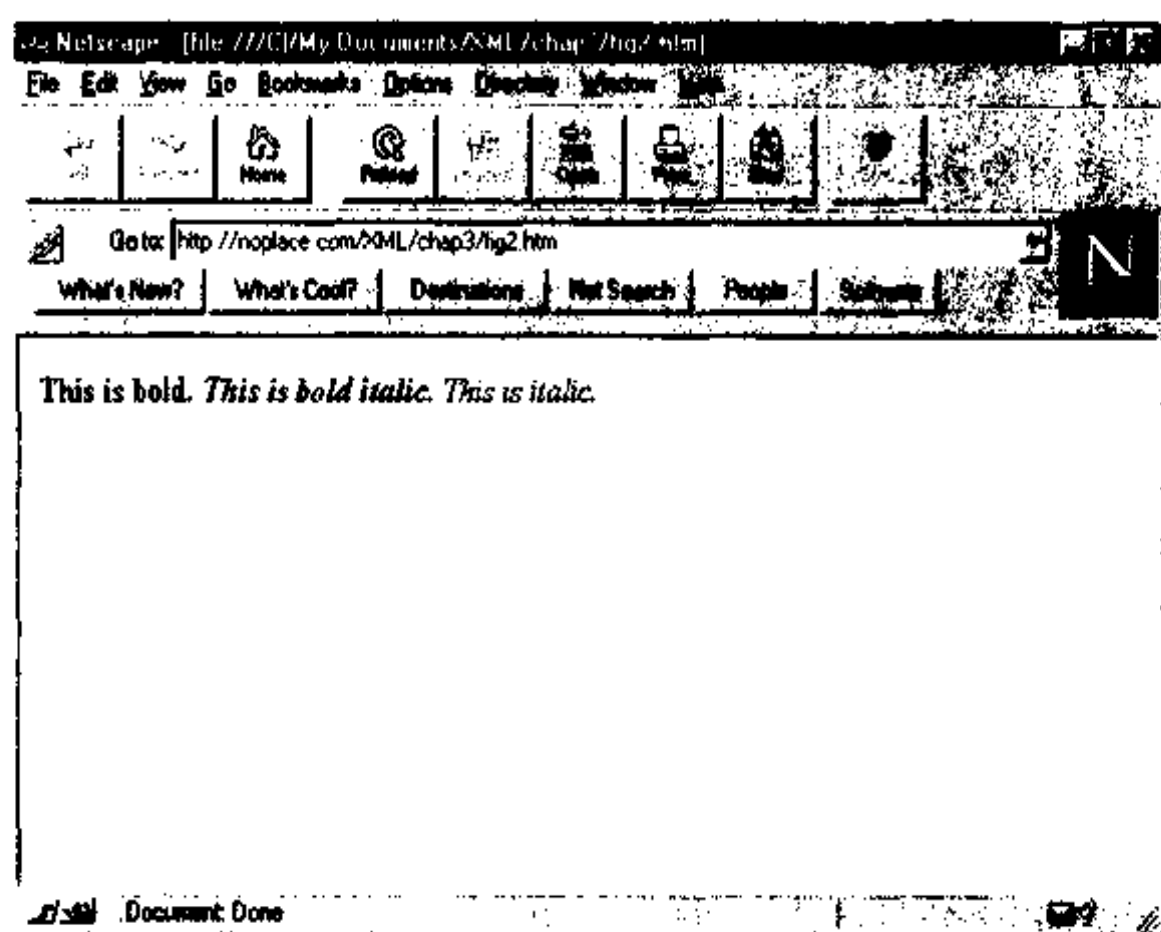


图 3-2 加粗、加粗倾斜和倾斜字体

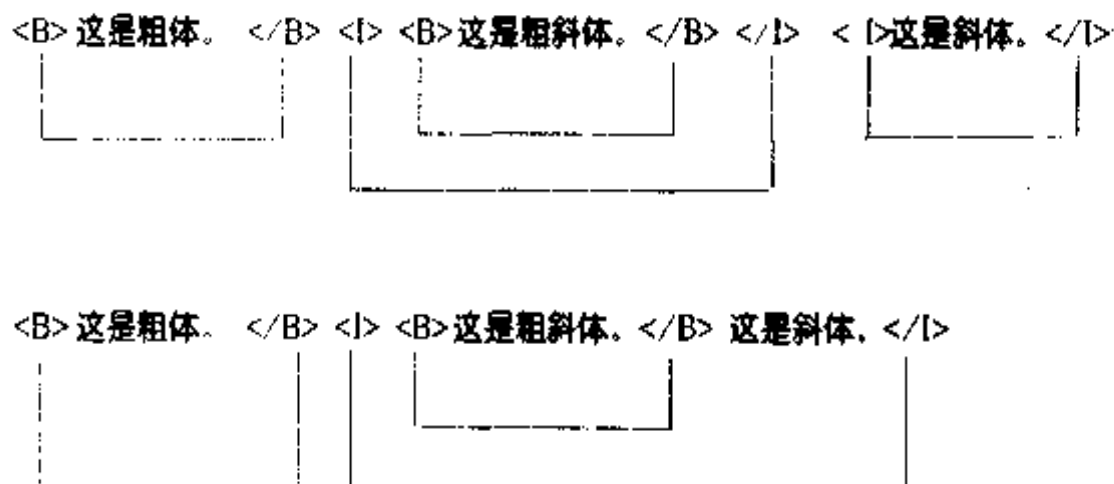


图 3-3 可接受的元素创建方式

我一贯推荐使用“包裹”得最好的方案(如这里的第一种方法),这样你可以挑选其中的元素将它们放到别的地方而不用担心丢失另一半结构。

注意

一组不完整的 HTML 标记可能会被解释为三种不同的意义这一事实说明了形式不好的语法可能会给需要在其脚本和样式中引用文档结构的开发人员带来麻烦和问题。那些应用样式表或试图通过文档对象模型引用元素的开发人员将会在与“破碎”的代码打交道时得到混乱的、不可预知的结果。XML 通过禁止使用不完全的代码使这一问题得以解决。

XML 在其它一些重要方面比 HTML 更加具体细致。HTML 对于在属性周围漏掉闭端标记和引号表现得非常宽容,它甚至可以容忍在文本中漏掉<、>和 & 符号。浏览器总是尽可能地分析 HTML,不过于计较元素应该在什么位置结束。许多 HTML 元素(包括经常使用的 IMG、HR 和 BR 元素)一般都没有闭端标记。XML 可没有这么宽容。对初学者来说,首先应该知道 XML 是区分大小写的,这一点与 HTML(甚至 SGML)不同。标记必须既得在大写形式上匹配,又得在意义上匹配。为严格形式,XML 最起码要求文档中所有的开端标记必须有某个闭端标记与之搭配,以及所有的属性都必须加以引号。漏掉这些标记或引号会在加载页面时产生错误,不包围任何文本的空元素必须被给予闭端标记(如
</BR>)或在它们的标记末尾加上斜线(如
或<BT />),两种方式是等价的。有些 HTML 浏览器将无用的末端标记当作另一个标记看待(如添加一个额外的断点),而不喜欢位于一般标记末尾的斜线号。但大多数浏览器都接受在元素名称和斜线号之间放置一个空格的方式(如
),这些浏览器包括最常见的 Netscape Navigator、Internet Explorer 和 Lynx,因此在 XML 浏览器成为标准之前这种方法可能是最佳方案。文本中的标记符号可能不产生错误,但它们将来一定会导致问题并使文档不能在技术上具备好的形式。XML 中<、>和 & 的内码与在 HTML 中相同,分别是 < > 和 &。

当心

XML 提供的内置条目(内码)不如 HTML 全面,本书第五章将讨论弥补这种方法。现在尚不清楚浏览器和分析程序在执行这些规则时的严格程度如何,但鉴于 HTML 浏览器造就的传统,估计将来的浏览器可能仍旧比较宽容,起码比标准所要求的宽容。尽管如此,我不是强烈建议遵循 XML 的所有规则。因为这样你可以拥有结构良好的文档,它们在调测和操作起来也更加容易。

元素和属性

虽然有些元素非常的简单(如 HTML 中的和
),但也有许多元素需要较多的信息。在 HTML 文档中写入不产生任何结果,这是 IMG 标记需要 SRC 属性来告知显示什么图形。在最近的 HTML4.0 标准中,大多数元素(即使是最简单的元素)都拥有属性。属性使设计人员能够更好地控制元素,使他们能够为脚本指定格式和标签元素、响应用户的动作以及定义默认的行为。

XML 对于在代码中使用引号比 HTML 严格许多,因为它总是用引号包围属性以避免出现令人烦恼的莫名其妙的错误。不论单引号还是双引号都是被接受的。双引号更加常见,但当属性必须包含一个双引号时使用单引号会更加方便。

有时很难确定信息是属于一个属性还是本身就是一个元素,尤其当它出现在格式化标记中时更不好确定。尽管 W3C 倾向于支持把格式信息放入元素中,而不是创建新的元素,但实际上往往没有这么明确。作为例子,试比较下面两段 HTML 代码:

```
<P><FONT FACE="Arial">This paragraph is in Arial, except for <B><FONT FACE="Times  
New Roman">this piece,< /FONT>< /B> which is in Times New Roman bold.< /FONT><  
/P>
```

和

```
<P STYLE="font-face:Arial">This paragraph is in Arial, except for <SPAN STYLE="font-face:  
Times New Roman; font-weight: bold">this piece,< /SPAN> which is in Times New Roman  
bold.< /P>
```

上面的第二个样式是较新的标准,使用的是样式信息而不是字体标记。它要求创建一些较新的元素,我们在后面将会看到,这在当你通过编写脚本动态地使用样式表和对元素进行动态操作时会提供很大的帮助。FONT 元素已在 HTML4.0 中受到“贬低”,它不再是被推荐的元素。尽管它现在仍是 HTML 标准的一部分并有可能仍旧被将来的浏览器所支持,但在 HTML 标准的下一个版本中可能不会再提到它。

注意

如果你从事文档对象模型的话,元素和属性的区别将会更加重要。文档对象模型的本意是把复杂的动态 HTML 和 XML 带给浏览器;并且,如果销售商忠实地实现了 W3C 规范,那么 HTML 与 XML 在各浏览器及其它应用程序上都是一致的。你在文档中创建的元素可以作为对象来编写脚本。这些对象的属性就是元素的属性,这使得即便在页面被加载后也能够很容易地交互式改变外观、位置甚至内容。

属性和元素的定义背后隐藏的逻辑有时很难说清楚。列表的定义方式可能是把列表项作为主列表元素的属性,但这样创建的开端标记应该很大,而且会包含多个重复的部分散布在页面上。因为用这种方式建造页面显然讲不通,所以 HTML 的创建者定义特定类型的元素、列表,它们包含子元素和子列表。另一方面,INPUT 元素使用同一元素名称和不同组属性分别创建几种不同的界面构件——域、检查框和无线按钮等。定义 INPUT 元素是很复杂的,解释创建不同类型输入所需要相关属性的可能组合方式得花费几个页面的篇幅。INPUT 元素这一例子说明了过分扩展一个元素的属性来完成的任务可能用几个独立的元素来管理会更加容易。

在简单的 XML 中给文档添加过多的属性是很困难的,因为标准的缺乏使得很难开发属性并使它们具有意义。随着 XML 应用的日益成熟,属性将变得更加重要(尤其是在用 XML 创建的标准中)。属性使 XML 标记能够向专门的程序传递更多的信息,这些信息包括程序的信息如何处理、如何显示以及是否显示。

对于刚开始从事文档设计的人来说,学习和掌握属性及元素可不是一件容易事儿。HTML 并不总是提供最好的例子,而且其它得以广泛应用的格式化语言也不多。因此,开发人员应该创建一些实例文档然后试着给它们加标记。当觉得自己已经达到一定的水平之后,把你的文档拿给别人看,看他们能否理解你所加的标记并说出文档的组成。如果他们对文档的理解与你的本意相似。那说明你已经做得不错了。

提示

▲ 要想真正精益求精,可以让另一个人使用你所提供的同样的规则标记一个新的文档。虽然你觉得需要构造良好文档的灵活性,但这样会阻止你使用 XML 的许多高级潜能。用这种方式从事具有良好结构的 XML 一段时间之后,把你的临时标准转换成正式文档类型定义(DTD)。它将使你的文档更加容易管理。

专业设计人员可以定期地与其他同事进行这种类型的相互切磋,但创建自己文档结构的单个设计人员却没有这样的便利条件。使用 HTML 的设计人员关心的是当文档出现在读者的屏幕上时是否显示正确,而使用 XML 的设计人员更加关心的是该文档在创作者的屏幕上好不好创建。元素和属性使用不当会使看上去很合理的文档结构表示很快变成一团糟。

注意

▲ 要想全面了解有关文档结构设计问题的论述,请参考 Rich Jelliffe 编写的《XML 和 SGML 详细说明》(Prentice-Hall 出版社 1998 年版)和 David Megginson 编写的《结构化 XML 文档》(Prentice-Hall 出版社 1998 年版)。有关何时使用元素何时使用属性的论战已持续了好几年,但这两门书提供了许多中立的示例和理论。

XML 和 HTML

W3C 将 XML 看作是其结构体系的一部分,而把 HTML、样式表和文档对象模型描述成其用户界面的一部分。但现实要比这复杂:XML 发誓给用户界面和 HTML 本身带来重大影响;XML 本身不是 HTML 正式替代品,但 W3C 计划在较近

的将来把 HTML 重新设计为一组 XML 模块。这种结构体系在一定程序上是合适的,因为 XML 更是创建结构的工具,而不单单是将这些结构应用于特定的界面。但是,正如我们将在整本书中所看到的,这些角色往往很模糊,很难说出 XML 的哪些部分是结构体系,哪些部分与界面直接相关。

XML 比 HTML 庞大。HTML 是 SGML 的一个应用,是由为 SGML 分析程序编写的 DTD 所定义的一组特定的标记(当初,HTML 只是看上去象 SGML 标记的一组标记,但后来逐渐变为 SGML 的一个应用)。XML 是一个子集,从技术上讲,它是 SGML 的一个“应用侧影或有限形式”,包含用于定义实例的 SGML 工具子集。大多数 HTML(也有一些例外不容忽视,如 SCRIPT 元素)可以用 XML 描述。因此,将 XML 与现有的 HTML 站点结合在一起完全是可行的,而且比较容易。XML 允许你扩展 HTML,在大多数情况下保持兼容(记住上文有关闭端标记的要点),并允许你远远超越老式标记语言中现有的标记集。

注意

将 XML 和 HTML 混合仍有许多争议,W3C 还需要宣布如何实现这一项目。HTML 会议笔记中的 XML(参见站点 <http://www.w3.org/TR/NOTE-xml>)会提供一些指导原则,尽管它总是将旧的 HTML 称为“沉积下来的渣滓”。

大多数 HTML 文档可以非常轻易地转换成构造良好的 XML 语法。接下来将对我们的第一个 XML 文档做这件事情。该文档的前面有一个处理指令声明这份文档是 XML 文档,如下:

```
<? xml version="1.0" ? >
<HTML><HEAD><TITLE>Our first XML Document< /TITLE>< /HEAD>
<BODY BGCOLOR="#FFFFFF">
<H1>Welcome to XML< /H1>
<P>Welcome to your first well-formed XML document. There isn't too much exciting going on
here, but there will be soon.< /P>
< /BODY>< /HTML>
```

上面代码中的第一行是 XML 声明,这在普通的 HTML 中是没有的。在这里,该声明仅仅宣布这是一个 XML 1.0 文档。尽管具有良好构造的文档应该拥有一个 XML 声明以宣布它们实际上是 XML,但这种声明并非总是必需的。这里的声明将文档标明使用 XML 1.0 版本,以便于浏览器和解释程序对其进行识别。如果不指定版本号,则将其默认为 1.0 版本。指定版本看上去可能不重要,但会使文档在 XML 规则发生变化时仍旧能够工作。

交叉参考

本书第五章将对 XML 声明作大量的介绍。

在上面代码中我们对 HTML 所作另外一个改动是保证所有标记的均匀搭配(所有开始标记都有对应的结束标记)。一般说来,空元素(如
)可以通过在元素名称和 />之间放入空格(如
)而转换为 XML 分析程序和老式浏览器都能够处理的形式。对于新文档,这些任务都不是太困难,但旧的 HTML 会产生许多问题,特别是手工编写代码的 HTML(大多数“所见即所得”工具能够默认地使用闭端标记)。老的 HTML 可能仍旧能够在浏览器中工作,但它们不被认作是 XML。如果你的 HTML 文档有问题的话,要么对其进行修复,要么不把它标示为 XML。

注意

文档的转换不是最麻烦的,因为你可以借助于一些工具,如 Dave Raggett 的 TIDY(参见站点 <http://www.w3.org/People/Ragget/TIDY>)。真正麻烦的应当是输出 HTML 的应用程序,它需要大量的修改、调试和检测,特别是当其编码逻辑产生“破碎”的 HTML 时。

创建自己的标记:具有良好构造的文档

你可以在你的 XML 文档开始处使用上文讨论的 XML 声明。尽管这不会向你提供安全保障(因为你没有定义任何结构防止自己出错),但它向你提供了开始设计自己的元素的合理位置。有一些设计是通过检查问题的顶层并对其进行细心分析来实现最佳构造的,而另一些设计则是通过从底向上来创建的。在试验过程中选择样本文档然后给它们加以标记会有助于更细心地选择自己的元素及属性。

这一部分中的文档具有良好的构造,它们使用语法正确的标记产生计算机能够解释的 XML,但不包含指定所有标记之需求并检验文档的 DTD(文档类型定义)。形式良好的文档更加方便并更容易保持与大多数 HTML 文档的向后兼容性。虽然创建自己的标记具有彻底的解放意义,但这不过是 XML 要完成的一部分。DTD 强加的规定虽然令人不愉快,但它的确使解释文档和重复使用文档容易了许多(应用建造人员特别欢迎经过检验后提供的有保证的结构)。形式良好的 XML 文档比 HTML 的组织结构性更强,但只有当你利用其更加强大的工具创建用于一组文档(而不是单个文档)的结构时,才能认识到 XML 的所有潜能。

作为例子,我们将在这一部分使用两个简单的文档:一个菜谱页面和一个目录页面。菜谱相当简单,包括标题、配料和指示。而且,这个文档要工作的话还必须组织

几层结构。配料和指示都是列表,当然,如果认为方便的话也可以将指示表示为一个段落。配料经常有替代者,甚至指示也不相同,大部分指示是关于准备工作的,但也有一些指示是提供建议。为这个菜谱编写标记比面向浏览器对其进行格式化稍微有些复杂。

```
<? xml version="1.0"? >
<RECIPE AUTHOR="Simon St. Laurent">
<RECIPENAME>Super-Duper Grilled Cheese< /RECIPENAME>
<DESCRIPTION>Succulent grilled cheese sandwiches that go beautifully with soup but are still de-
lightful on their own, < /DESCRIPTION>
<INGREDIENTLIST>
<TITLE>Ingredients: < /TITLE>
<INGREDIENT><REALINGREDIENT>2 Tablespoons butter< /REALINGREDIENT> (or <AL-
TERNATEINGREDIENT>non-stick spray< /ALTERNATEINGREDIENT> if preferred) < /INGREDI-
ENT>
<INGREDIENT><REALINGREDIENT>8 slices wheat bread< /REALINGREDIENT> (or <AL-
TERNATEINGREDIENT>other bread < /ALTERNATEINGREDIENT>)< /INGREDIENT>
<INGREDIENT><REALINGREDIENT>1 /4 pound jalape? o Monterey Jack< /REALINGREDIENT
> (or <ALTERNATEINGREDIENT>other cheese< /ALTERNATEINGREDIENT>)< /INGREDI-
ENT>
<INGREDIENT REQUIRED="no"><REALINGREDIENT>2 bottles beer< /REALINGREDIENT>
< /INGREDIENT>
< /INGREDIENTLIST>
<INSTRUCTIONS>
<STEP>Melt butter in frying pan over low to medium heat. < /STEP>
<STEP>Slice cheese into thin slices. < /STEP>
<STEP>Place cheese slices evenly on 4 slices bread; cover with other slices. < /STEP>
<STEP>Fry sandwich carefully in butter, flipping repeatedly to avoid burning. < /STEP>
<STEP CLASS="Serving">Cut sandwiches into quarters diagonally. < /STEP>
<STEP CLASS="Serving">Serve hot with beer. < /STEP>
< /INSTRUCTIONS>< /RECIPE>
```

这份标记文档并非一定要跟上面写得完全一模一样,你可以使用你喜欢的标记名称也可以使用现实生活中的菜谱标准。关键要注意的是,该标记文档中的所有元素都有开端标记和闭端标记,其中有些元素具有一些属性标示它们是可选择步骤或充当建议。当然,实际生活中的菜谱肯定比这详细许多,也更加复杂。

这份文档没有任何格式,这在大多数 Web 浏览器中是个问题。如果它是最终应用的话你必须创建提供显示信息的样式表以使其在浏览器中被正确显示。在 XML 中,分析程序应该能够接受行断点(即便没有
标记)、空格及其它空白。现在,还没有哪个 Web 浏览器能够这样工作。最起码在短时间内你还必须使用 HTML 格式元素或 CSS 空白及显示样式来产生想让别人阅读的文档。XML 还有自己的机制

声明是否处理空白,这在第五章中将进行讨论。

我们的下一个例子是一个简单的目录。常见的目录是价格列表,当然在这种简单的文档中价格和商品部件列表是最重要的组成部分。目录中的一个条款包含多个项目(如产品和辅助部件)是不常见的。这个目录例子可以用于包含标准商品格式、标准报价信息及运货信息的目录。你也可以以不同的方式制作标记文档,如果有现成的工业标准存在,最好使用或扩展该标准,这样的创建自己的标准省事且有效。

```
<? xml version="1.0"? >
<CATITEM CATEGORY="Clock">
  <ITEMNAME>Jimbo's Super Clock< /ITEMNAME>
  <DESCRIPTION><STORY>Ever wake up in the morning to discover that your alarm clock didn't
  go off because the power failed? Or that your roof leaked, it rained, and the stupid thing just plain
  shorted out when it got wet? Now you don't have to worry about waking up two hours after you
  were supposed to be at work. < /STORY>
  <FEATURES>Our latest, greatest Super Clock is a dream come true. It plugs into the wall but
  has its own set of batteries and protection from short circuits. The batteries even warn you when
  they're starting to fade - and they come with a twenty-five year guarantee! This clock is completely
  watertight, a sealed sphere of time in a stainless steel case. The clock face is large enough to
  read from a distance, and lights up with a touch for those nights when you're stumbling in the dark.
  The alarm starts off quiet, but gets louder and louder when you don't turn it off-guaranteed to wake
  even the soundest sleepers. A snooze feature let you sleep just a little bit more, but it won't let
  you sleep in for more than an hour past the alarm. This clock is ready to adorn your bedroom, and
  even includes connections for lamp controls to brighten your morning, and electroshock clips for
  those who can't wake up any other way. < /FEATURES>< /DESCRIPTION>
  <PICTURE SRC="supclock.gif" />
  <ITEM><PRODNAME>Jimbo's Super Clock< /PRODNAME>; <PART>SC45-A< /PART> <
  PRICE> $ 199.95< /PRICE> (<AIRF> $ 19.95< /AIRF> freight /air, <GROUNDF> $ 7.
  95< /GROUNDF> ground) <WARRANTY>Twenty-five year< /WARRANTY> Warranty. Made
  in <ORIGIN>Canada< /ORIGIN>< /ITEM>
  <ITEM><PRODNAME>Lamp Controller< /PRODNAME>; <PART>LC45-X< /PART> <
  PRICE> $ 25.95< /PRICE> (<AIRF> $ 9.95< /AIRF> freight /air, <GROUNDF> $ 4.95<
  /GROUNDF> ground) <WARRANTY>Ten year< /WARRANTY> Warranty. Made in <ORIGIN
  >Canada< /ORIGIN>< /ITEM>
  <ITEM><PRODNAME>Electroshock Clips< /PRODNAME>; <PART>ES45-L< /PART> <
  PRICE> $ 59.95< /PRICE> (<AIRF> $ 9.95< /AIRF> freight /air, <GROUNDF> $ 4.95<
  /GROUNDF> ground) <WARRANTY>One-year< /WARRANTY> warranty. Made in <ORIGIN
  >USA< /ORIGIN>< /ITEM>
< /CATITEM>
```

正如你所看到的,这个例子开始变得比较复杂。使用合适的样式表和其它格式化工具,你可以在打印纸上、Web 网页上或屏幕上创建出吸引人的布局,也可以使用

脚本在订购信息部分提供商品的价格及样品图片。尽管 XML 看上去有些挑剔,要求你关闭所有标记并小心语法,但你会很快发现通过坚持这些约束所得到的文档管理中的灵活性远比失去的文档创建中的灵活性重要。

用分析程序和浏览器测试文档

尽管在开发自己的标记时手工编写文档代码非常有意思,但大多数 XML 使用与开发人员目前用于编写 HTML 相类似的程序来编写(许多 XML 将直接从数据库产生出来,这使得编写代码的个人与 XML 的产生离得更加遥远)。XML 要求满足更严格的标准,对于遗漏的标记也远远没有 HTML 那么宽容。找到在哪个地方漏掉的标记往往不是一件容易事儿,而且这方面的工具目前还很低级。

用于检查你的工作的一个比较好的工具是 Lark,它是用于 XML 文档的一个非检验分析程序。检验分析程序对照描述文档结构的文档类型定义对 XML 标记进行检查,而非检验分析程序只检查确保文档具有良好的构造。Lark 不是很完美,但它能够很好地显示文档的结构并把各部分以浏览器或分析程序对其进行解释时所用的方式揭示出来(如果你想让自己大吃一惊,将它用于 HTML 文档看它能发现多少错误)。Lark 是一个由 Tim Bray(XML 规范的编辑人员之一)开发的 Java 应用程序,它的许多源代码(不是全部)可在以下站点上得到:<http://www.textuality.com/~lark/>。

提示

▲ 一个基于 Web 的 Lark 叫做 RUWF(系“Are you well-formed”的谐音),可在站点 <http://www.xml.com> 处得到。它接收 URL,返回一个该文档构造良好的祝贺消息或一个错误消息列表。构造太差的文档会返回一个巨大的错误列表。

Lark 目前是命令行工具(其版本号是 1.0 最终 Beta 版),没有图形用户界面。要运行 Lark,需要具备能够在浏览器之外运行的 Java 虚拟机。在这个例子中,我使用 Microsoft Visual J++ for Windows 95 所带的 jview 工具,但在 Sun 公司的 Java 开发工具(JDK)、Macintosh 公司的 Runtime for Java SDK 和 OS/2 Warp 4.0 版中也有类似的工具。大多数人仅仅通过浏览器中的小应用程序窗口看到过 Java,但 Java 也能够仅在仅有文本的命令行环境中很好地工作。jview 应用程序以 Java 应用程序的名称(它与类文件的名称相同,不带 .class 扩展名)及该程序所需的参数作为其参数。运行一个没有任何参数的 Java 应用程序通常会产生一条消息说明所需要的参数。Sun 公司 JDK 的 Java 应用程序运行方式与此相似,在我的测试中产生同样的结果。本书将不时使用这两个 Java 应用程序。

提示

如果你没有 Visual J++，那么不要去下载 Sun JDK 的 8.75 兆字节，也不需要开发 Java 应用程序，因为 Sun 公司提供了一个类似的 Java 运行时环境(JRE)，请访问站点 <http://java.sun.com/>。它也有 2.2 兆字节的大小，但这已足以使你轻松对其进行管理。

Lark 只对阅读文档结构有用。虽然它能够很好地匹配标记和建造文档树，但它不把文档中的结构与 DTD 中的结构进行比较。我们将在后面的章节讨论做这种工作的工具。但 Lark 是一个值得收藏的工具，特别是当你计划创建自己的文档树或必须把旧的 HTML 转换成 XML 时。

注意

对于某些参数和 XML，Lark 还存在着一定的问题。详细情况请查看 Lark 文档。

我们的 Lark 示例以一个简单的 XML 文档开始，这个文档是一份天气报告，只包括一些基本的信息：

```
<? xml version="1.0"? >
<WEATHERREPORT>
<DATE>7 /14 /97< /DATE>
<CITY>NORTH PLACE< /CITY>, <STATE>NX< /STATE> <COUNTRY>USA< /COUN-
TRY>
High Temp:< HIGH SCALE="F">103< /HIGH>
Low Temp:< LOW SCALE="F">70< /LOW>
Morning:< MORNING>Partly Cloudy, Hazy< /MORNING>
Afternoon:< AFTERNOON>Sunny and Hot< /AFTERNOON>
Evening:< EVENING>Clear and Cooler< /EVENING>
< /WEATHERREPORT>
```

为使 Lark 对其进行分析，需要告诉 Driver 类分析我们的 XML 文件。在 Windows 95 中(如果你已安装了 Sun JDK，并且 Lark 的文件位于 C:\lark)，命令及其结果如以下屏幕输出所示。使用 Microsoft 的 Visual J++ 环境的用户应该使用命令 jview 代替 java。

```
C:\My Documents\xmlaprim2\lark>java Driver weather.xml
Hello Tim
```



```
Lark V1.0 final beta Copyright (c) 1997-98 Tim Bray.  
All rights reserved; the right to use these class files for any purpose  
is hereby granted to everyone.  
Parsing...  
Done.
```

如果一切进行顺利并且 Lark 没有报错,那行你的代码是构造良好的。幸运的是,Lark 提供的可理解的错误信息能够帮助你查找侵入代码的任何错误。例如,如果遗漏了最后的 `< /WEATHERREPORT >`,会产生以下信息:

```
C:\My Documents\xmlaprim2\lark>java Driver weather.xml  
Hello Tim  
Lark V1.0 final beta Copyright (c) 1997-98 Tim Bray.  
All rights reserved; the right to use these class files for any purpose is hereby granted to  
everyone.  
Parsing...  
Lark:weather.xml:9:43:E:Fatal: End of document entity before end of root element.  
Done.
```

如上面所示,Lark 提供有意义的消息(指出在根元素关闭之前文档就结束了)以及错误出现的行位置(9)和列位置(43)。

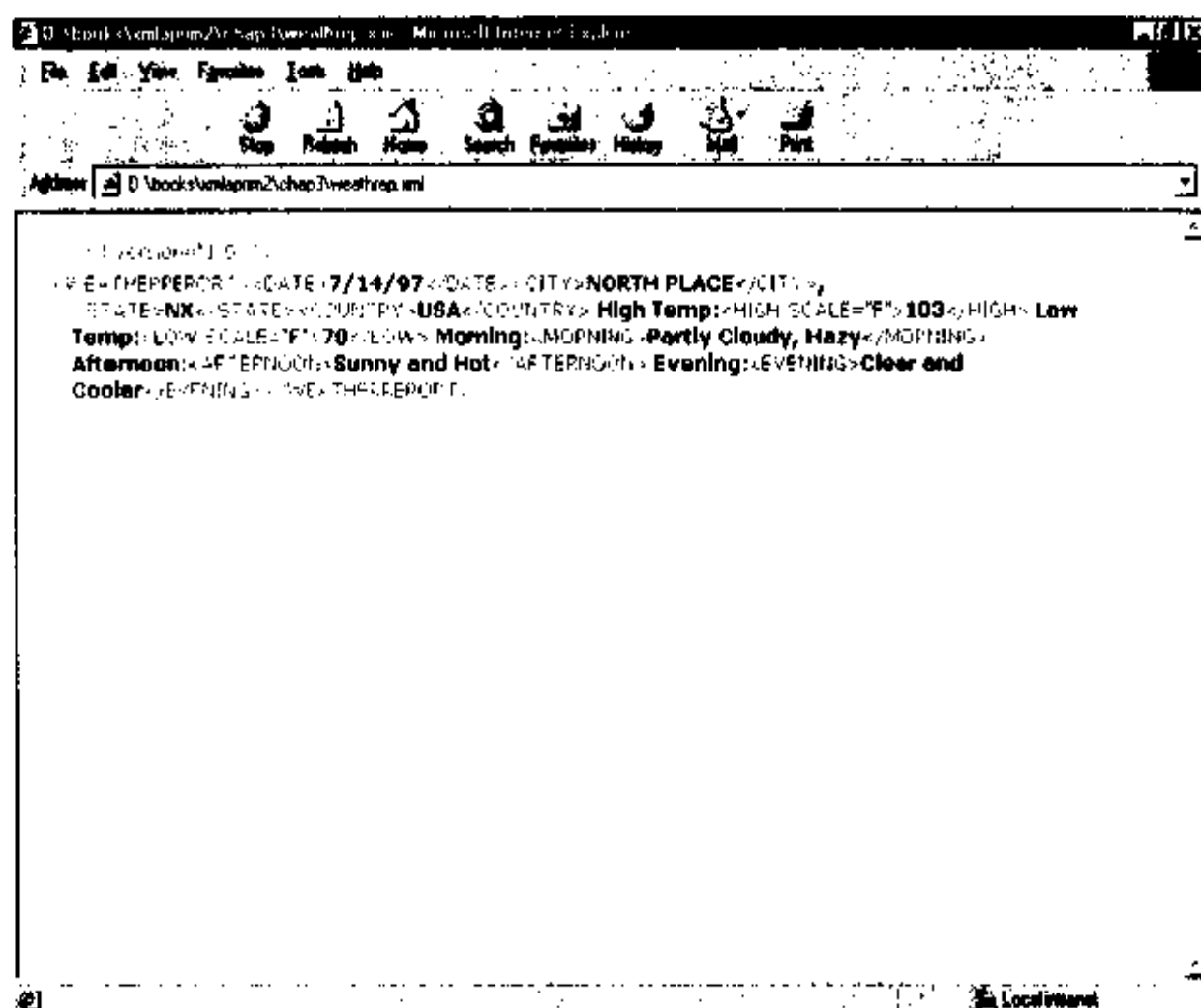


图 3-4 当给予没有样式的 XML 文档时,Internet Explorer 5 将分析该文档并把它连标记一块儿显示出来

对于有些用户来说,一种更容易使用的工具是 Internet Explorer 5,它也检查文档是否构造良好。载入上面的天气报告文档(没有任何样式信息)会产生图 3-4 所示的屏幕显示。

它虽然看上去不好,但不失为检查代码一种好方法。如果你出错了,Internet Explorer 5 同样会告诉你,如图 3-5 所示。

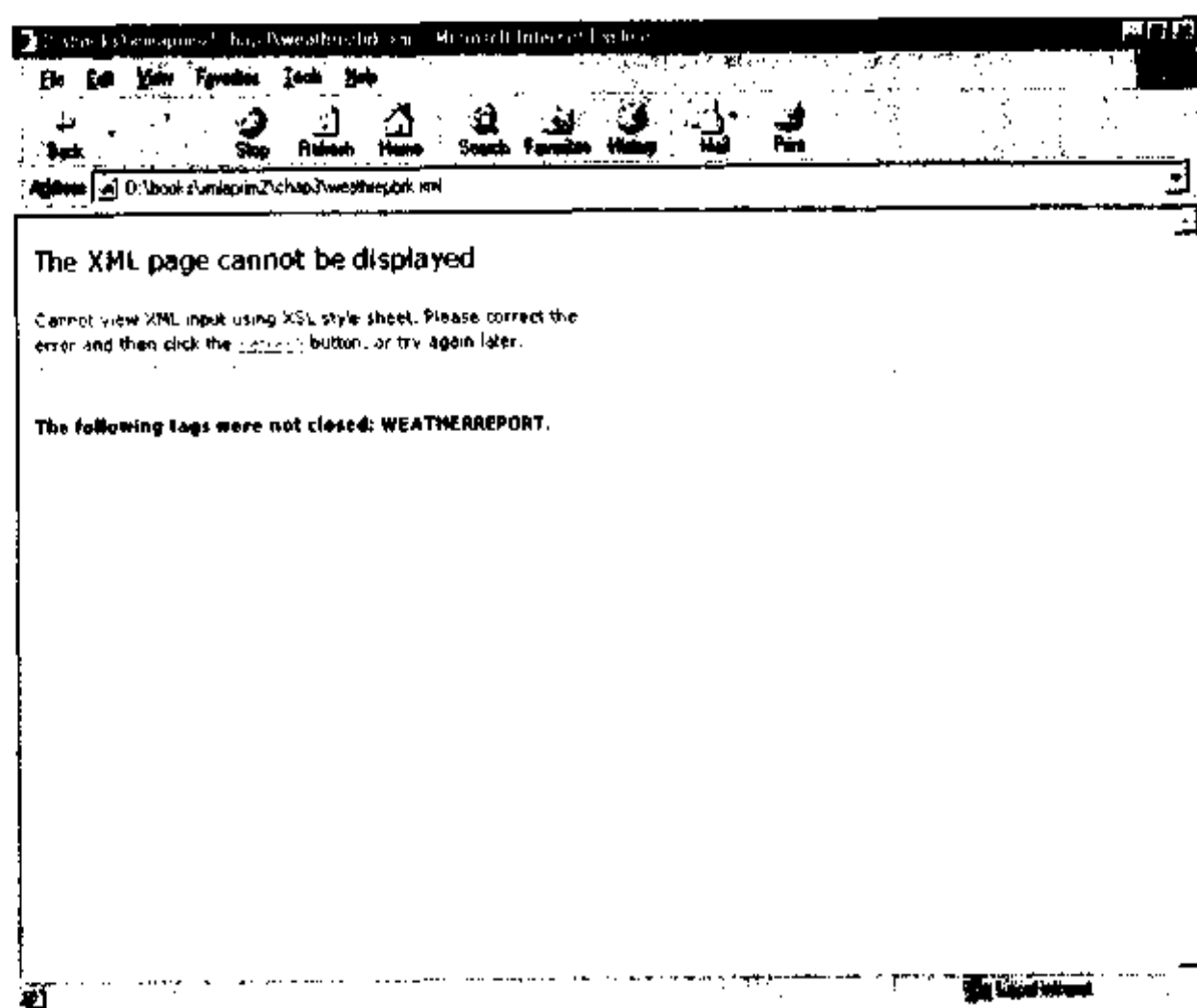


图 3-5 Internet Explorer 5 同样显示构造不好的文档所产生的错误消息

Internet Explorer 5 的错误消息不如 Lark 产生的那样全面,但二者可以满足不同的需要。Lark 实际上是开发人员的分析程序,旨在用于测试和处理 XML 文档。作为一个命令行工具,它只是报告发现的问题(前面提到的 RUWF 工具是在 Lark 之上建造的,其行为与 Lark 类似)。Internet Explorer 5 中内置的分析程序是一个批量生产的分析程序,它被优化为具有更快的速度并放入一个友好的多功能界面中。这两个不同的模型已被证明适用于不同的项目。

第 四 章

项目规划

现在你已经看到过 XML 的样子,并见识过它向设计人员提供的巨大功能,接下来先让我们对此作一简单综述。XML 允许设计人员创建自己的标记。级联样式表(CSS)和可扩展样式语言(XSL)能够与 XML 非常好地结合,以让你定义自己的格式化语言。你几乎可以做任何想做的事情,而不用关心某人几年前在一次委员会上(离你很远)就你的特定学科领域所说过的话。XML 的语法是固定的,但它的符号集是开放的。XML 是一个非常强大的工具,能够做令人惊奇的事情。不幸的是,XML 有点儿象是链条钢锯——它无比强大,无须费多大努力就能很好地完成工作,但同时又很危险。链条钢锯需要定期维护,它能够杀害用户或导致巨大的危险。XML 可能不会对你的身体造成伤害,但它会因为项目规划得不好而带来极大的危险性。如果用户认为他们不用怎么了解这一工具就能开始用它工作的话,那么他们很快就会受到教训。

XML 有可能成为有史以来在 Web 上发生的最大灾难。由写得很糟糕的 XML 组成的站点表面上看来可能不错,但它们很快就会给维护工作带来恶梦,要求一组开发人员必须挑选出同一站点上的网页之间的不兼容性。大多数人可能选择重新创建,将本来可以高效率编写代码的时间浪费在对站点的重复工作上。公司可能会继续浪费几千美元的资金把信息从一个规划很差的系统转换到另一个系统。HTML 并非总是“美丽的”的,但编写得不好的 XML 简直会更加“丑陋”。

XML 并非仅仅用来创建在某人的屏幕上看起来很好的文档。XML 可以用于任何类型的数据传送,包括嵌入在过去一直被当作“普通”文档中的数据传送。不幸的是,这意味着该过程必须卷入更多的人力。当 Web 第一次涉足商业时,它通常首先发起于任何组织或个人操作的电脑和网络,然后慢慢转移到市场和设计。尽管一个公司的不同部门都向同一 Web 站点提供信息,但不同部门的所属领域不需要全部相同,而且也可以以不同的格式到达。Web 应用至今主要是被关系数据库驱动的,二者都有自己的格式和结构。XML 发誓要统一这些不同的部分,使部门之间的数据传送更加通畅,通过向外面 Web 的传送更加容易。但随着这些传送的增长,将需要更

多的协调。

当心

如果你只想使用 XML 扩展你现有的 Web 开发工具,对于协调的文档管理不感兴趣,那么你在阅读本书其余部分时将看到应该怎么做。但要记住,这样你会丢失 XML 的许多优势,你将仍旧花费许多时间去建造转换程序和给网页手工编写代码,这与使用 HTML 没有多大不同。能够创建自己的标记是一件十分激动人心的事情,但这仅仅是开始。另外,如果你已作为文档编写并使用了 DTD,或者公司开发了一个新的不兼容的 DTD 要求你来使用,那么从长远处讲你会使自己面临更大的问题。

使这一过程工作需要一个组织内部的各方甚至整个产业界的合作。SGML 从最开始就在委员会中陷入了困境,XML 可能会激发起关于它本身的许多次会议。这种组织性的困境并非一定意味着无休止的会议和连续的重新组织或公司新 Web 站点发布日期的无限期拖延。XML 能够大大地推进协作,但在开始时也需要一定的协作以使工作启动起来。开发 XML DTD 未必是整个公司的事情,但使 Web 开发人员之外的更多人参与进来是极具重要性的第一步。

XML 用户

因为 XML 有潜力发展成为一个组织当中所有文档的标准格式,所以许多以前从来不考虑数据格式的人(可能现在他们也不想考虑)可能要受到 XML 发展的影响。在商业文档管理(大多数是文字处理)工具上实现标准化的公司有机会转向更加灵活并且实际上是自己定制的解决方案。对这种巨大改变的规划可能要花费几年的考虑时间,然后慢慢地从老的方式过渡到新的方式。

在老式方法中,Web 开发小组是一群“奇怪的”技术和设计人员,他们呆在一间以前用作仓库的房子里从不与公司的其他人打交道。这种“美好”的昔日时光正在走向结束。当 Web 移向公司的内部,而不仅仅向外部发送数据的时候,开发小组再也不是生活在公司的“边缘”。Intranet 早已开始了这一过程。即使没有计算机背景的公司雇员已经成为基层网络管理员,他们使用个人 Web 服务器及其它小型工具来分发数据。有些公司已经为各部门建立起中心 Web 服务器,让它们承担自己内容的组织责任。

XML 有能力比最近一波 Internet 应用还要更加深入地渗透到企业内部,成为几乎所有桌面上的标准格式。XML 远远不止是 Web 和打印文档的格式,它能够提数据库格式、控制指示容器、通用交换格式以及必将出现的许多其它应用。XML 的

极大灵活性使其具有走进几乎所有数据处理程序的巨大能力,而不仅仅是用于文字处理和 Web 开发。即使被创建来进行有限的文字处理和 Web 开发的 XML 文档也能够进一步挖掘出潜能(如果文档是使用优秀的 XML 创建的,那么要完成这一任务非常容易)。

XML 的这种扩展性能意味着开发人员必须与公司内部更多的人进行对话才能开发出满足组织需要的标准。Web 的开发工作在过去不过是设计接口和使用内容自动化——从各种来源获取内容,将它们转变成 HTML,然后以吸引人的形式把它们展示出来。而 XML 的开发则要求仔细检查内容设计和工作的自动化。内容展示也很重要,但 XML 允许开发人员让文档使用 XML 和标准的 DTD 作为它们的原始格式,这样就可以省去转换工作,并有可能开辟强大的新的文档管理途径。这显然不会立即实现,该过程可能需要几年时间;但如果开发人员不肯放弃导致很多问题的旧的文档结构的话,它永远都不会实现。

要避免导致恶梦性的问题必须对用户进行大量的咨询。即使你的 Web 开发工作已从计算部门转移到市场部门,也应该回到计算部门与它们磋商还可以做些什么工作以保证在长远的将来具有兼容性。在以往工作生涯中强迫 HTML 以他们想要的方式提交页面的设计人员必须与用大量的表来组织数据的数据库管理人员进行对话,并在一些共同的解决方案上达成一致。有特定一组桌面应用程序上实现标准化以避免经常发生文件冲突的那些公司可能会发现他们的大量资金只是投向了一个暂时的解决方案,而真正的文档交换工具才刚刚开始出现。即便软件销售商扩展对新的 XML 工具的支持(有几家销售商已承诺这么做),要说服用户进行改变也是很困难的。

结构

XML 令开发人员最感到困难的要求是他们必须将其文档结构标准化。这并非意味着每个文档都必须相同,而是要求开发人员必须检查进入其页面的组件并创建。HTML 提供了一些用于创建结构的工具,这些结构包括图形、列表和标题等,但除格式化之外,从不要求开发人员应用结构。HTML 标记有些时候必须工作一致,但标题标记从来没有要求它后面必须跟一个段落。XML 具有这些要求,并且通过分析程序的检验强制它们的执行。要完全利用好 XML 要求开发人员仔细检查他们的文档结构和数据结构并更加清楚地重新表述它们。

文档结构

本章就具有一定的结构。它以一个题目开头,后跟一些段落。接下来是一个标题,再有一些段落。然后又出现一个标题,其后紧跟一个介绍性段落和一个子标题。现在你正在阅读的文本是一个位于子标题之下的段落。因此,我们在本章的文档结

构中有几个层,如图 4-1 所示。

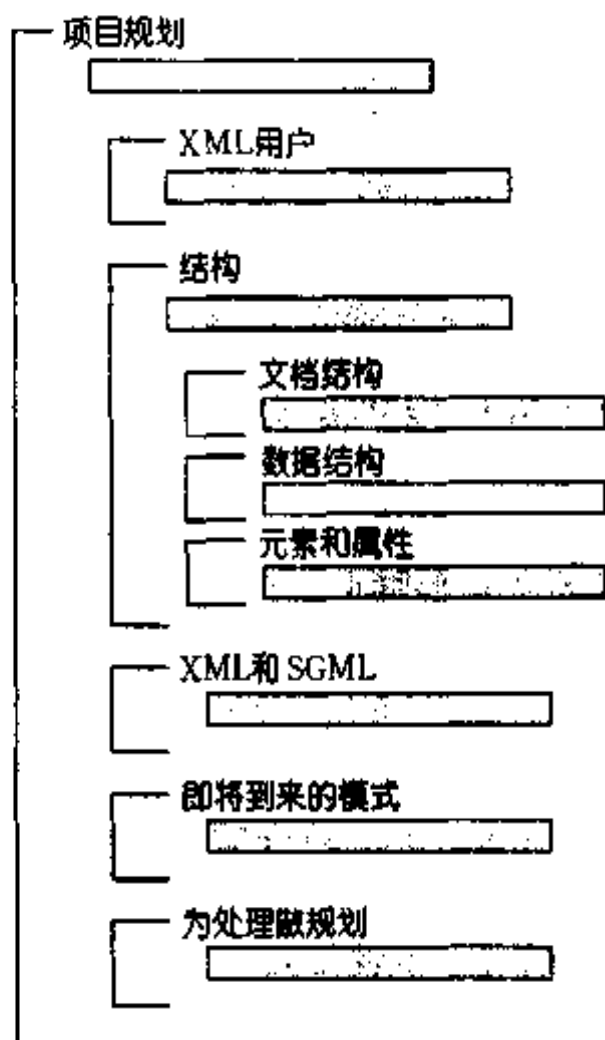


图 4-1 本章结构图

并非每个文档的结构都象本章这样复杂,但大多数文档都有某种形式的结构。以通知书为例,它以收件人和发件人开头,并在头部包含日期和其它有关公司的信息,接下来的文本就基本上没什么结构。信件通常提供更多的信息(特别是在商业信函中),经常包含发信人和收信人的地址,以及称呼(如 Dear John)、结束语(如 Sincerely)、签名,并再复制一份发信人的姓名。还有可能包含一份附件,以及参与文档准备工作的打字员和其他人。文档可能比这些基本的例子复杂许多,但也有可能比它们简单(尽管这种可能性很小)。

以上讨论并不意味着 Web 开发小组应该给公司的信函和其它文档强加上一些标准,当然这最终可能会成为合理的目标(关于这件事我们将在第八章中处理)。许多公司试图对经常使用的文档的格式实施标准化,但是遇到了阻力。出于各种原因,人们不喜欢以与别人相同的方式格式化自己的信函或便笺。当为公司文档开发标准时,尽量保留一些灵活的余地——最起码在字体和大小等风格上不要强求一律。太强硬地推行标准化会使标准更加不能付诸应用,特别是在现阶段用于使用标准的友好工具尚未出现之时。

HTML 采用一种相对简单的方法建造文档——识别不同的组件并创建出用于重复产生这些组件的工具。但是它不以任何具体的方式将这些零部件连接起来(那

当然,列表、表单和表格是例外)。HTML 文档的 BODY 部分中的大多数元素能够以任意顺序出现在任何地方。没有什么规则要求 H2 元素必须出现在 H1 元素之后, H2 元素能够出现在文档中的任何地方,有没有其它标题都没有关系。HTML 中的唯一限制在于创建块元素时。例如, H2 元素在 H1 元素内部不能工作:

```
<H1>This is the top <H2>This is the middle </H2>This is the end </H1>
```

这一行代码不会在单个行中产生两种尺寸的标题,其结果如图 4-2 所示。

除此之外,HTML 很少对其文档组件的使用强加其它限制。列表元素应该出现在列表中,但如果不是这样的话浏览器也会,表单元素也与此类似。表格元素(行和列)如果出现在表格之外的话不具有任何意义,浏览器将把它们忽略掉。HTML 在结构上的限制很少,使得初学者能够很容易地创出自己的网页。即使 HTML 有这种结构限制,它们也不会强行实施,因为 HTML 不要求作文档检验。

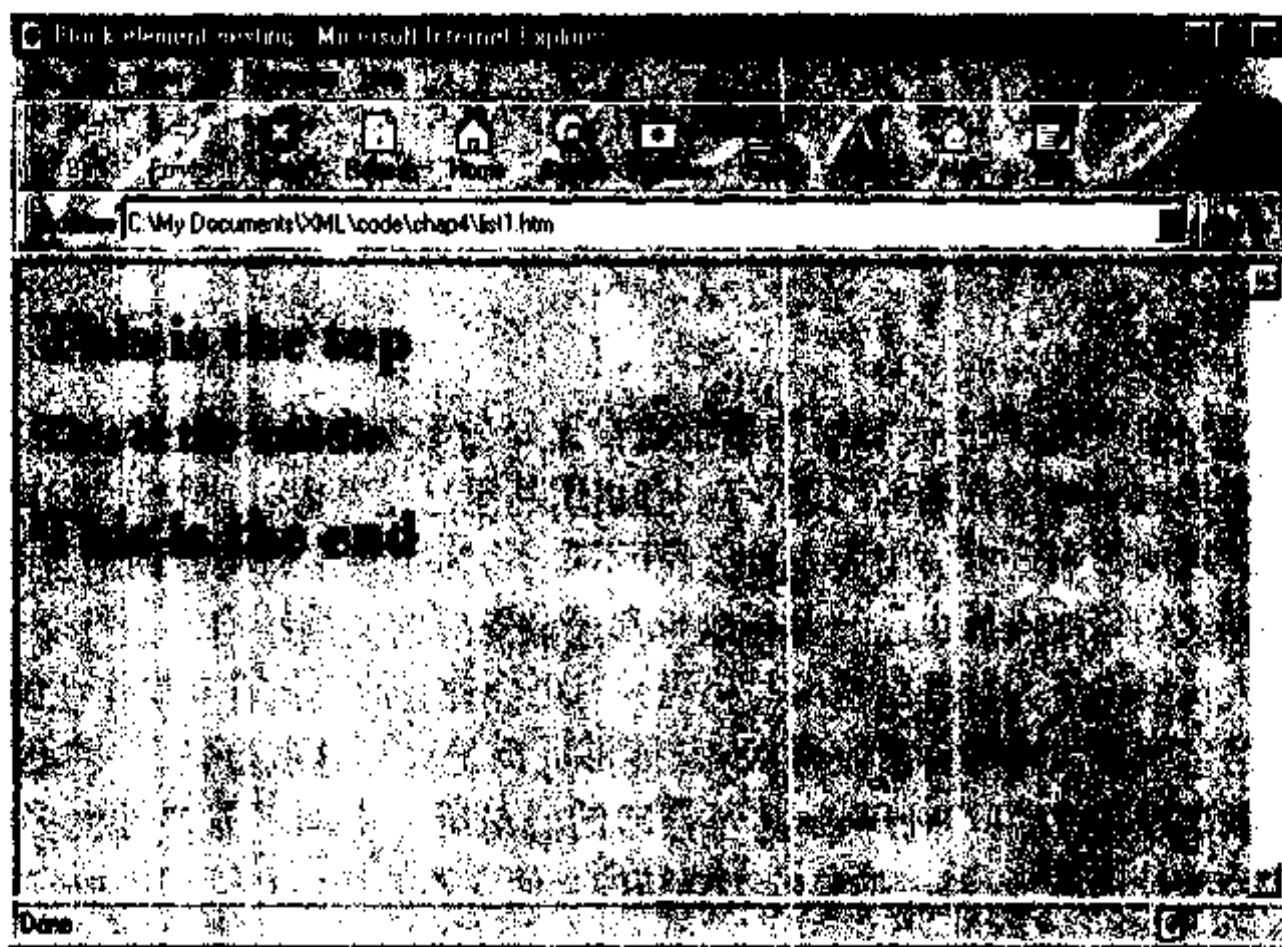


图 4-2 块元素中嵌入的不当行为

XML 能够使开发人员创建出比使用 HTML 可以得到的复杂许多倍的文档。文档结构非常有用,因为它们能够确保文档拥有所需要的元素,防止文档作者胡作非为地创建出乱七八糟的格式或把信息放入错误的位置。这些 XML 结构使文档管理系统能够就完整性进行检查,并帮助格式化引擎产生引人注目的 XML 文档显示形式。与样式表结合后,文档结构使得能够使用 XML 创建可读的、高度格式化的 Web 页面,使其具有标题、子标题、段落、引文、嵌入块及以前在 HTML 中可以得到的所有结构。

查看文档很容易就可以识别出文档结构。文档结构的主要任务是提供信息的路线图,使读者一眼就能找到他们需要的信息。不用惊奇,大多数 SGML 实现都是以文档工程为目标的,这些工程产生的大量信息需要通过结构帮助读者在其中穿行。文档通常遵循严格的惯例,为便于参考引用经常借助于段落编号(例如,“见段落 1、3、2 有关该装置的详细信息”)。已经具有强壮结构的工程对标记语言来说是很容易的目标,真正对其具有挑战性的是包含多种类型信息的结构化不强的那些文档。

当在文档结构的基础上开发 DTD 时,开发人员应该检查哪些事情已经做过了。大型公司或其它组织可能已经拥有他们自己的 SGML 标准。不论多大或多小的组织都应该坚持标准,以使信息很容易地流通起来。与数据结构相比,文档结构更不可能需要它们自己唯一的 DTD(不论什么组织或个人创建的商业信函都大致相仿)。

需要一些灵感的开发人员可以查看 TEI(站点是 <http://www.uic.edu/orgs/tei>),这个学术组织已经为学术文档的编码产生了许多成立的标准。为了能够向从印刷书转换成学术人员可以更乐意使用的电子格式提供标准,TEI DTD 为各种各样的材料(从散文到诗歌,从副本到评论)提供扩展的框架。TEI 标准经常跨越文档结构编码和数据编码之间的界线,实际上,它们演示了二者的区别是相当模糊的。尽管如此,他们在将标记结构改编为各种文档类型时一般都是经过深思熟虑的,都有过事先的规划。

一般说来,好的文档结构系统通常比好的数据结构系统明显。大多数组织都已经考虑过一些这方面的问题,有一些组织甚至已经考虑过这些问题对文档显示、存储和管理的影响。桌面出版人员已经(不得不)为一些不同类型的信息建造了主页,技术和文档写作人员也已经普遍地与预先设置好的文档结构开始打交道。XML 向开发人员提供了为这些结构编写代码并有可能使它们更具有互操作性的机会。从格式化信息解放出来的自由使得 XML 能够灵活地处理这些事情,并使新闻信息有可能重新出现在 CD-ROM、Web 甚至打印的公司历史中而不需要做太多的重新建造工作。从长远看来,建造抽象的文档结构会使文档的产生和管理更加容易。

数据结构

如果说文档结构为文档提供了一个内容表,那么数据结构提供的是索引。文档结构组织你的文档,以帮助读者理解文章的结构或使他们跟得上展开的讨论而不会迷失。数据结构直接反映内容,很少关心它们在整个文档结构中出现的位置。尽管 DTD 可能要求它们出现在某个特定的文档结构中,但它们经常可以自由地在文档内部“漂浮”。

基于数据内容的创建文档的能力,使 XML 具有了优于 HTML 的大部分优点。

虽然文档结构非常有用,但它们对于计算机重复使用文档中信息的方式很少有直接的影响。它们可以使管理系统更精确地识别信息所在的位置,但对于实际检查信息帮助甚微。文档结构帮助人们阅读文档,但它们很少能帮助计算机在没有人辅助的情况下得到所需要的关键性数据。文档内部的表格要以被轻易地识别出来,但如果没有其它信息的话,计算机根本无法从该表格中抽出数据供分析使用。表头可能会在某种程度上有用,但一旦文档中出现具有相似表头的多个表格(这种情况是很常见的),计算机就会在决定哪些信息相关时受阻。

XML 发誓要帮助计算机和人(二者兼顾)检查文档中的数据并从中抽取数据供重新使用或修改。XML 能使用户将分散在文档中的数据和细心组织在表格中的数据以及存在于其它格式结构中的数据收集起来。当在一个合适标示的文档上工作时,分析程度能够以各种格式返回信息,将文本数据转换成更加结构化的数据库格式或电子表格格式。虽然 XML 缺乏操作文档内部数据(如对域进行计算)的工具,但它提供了围绕独立的数据格式建造应用所必需的原始材料。

基于内容的元素也提供了文档的结构化能力。除了能够(与样式表配合)提供基于内容的格式以外,这些元素可以与 DTD 连接在一起提供数据集(而不仅仅是单个数据)。例如,在本书第三章的目录例子中,最后几行实际上是定货和运输信息:

```
<ITEM><PRODNAME>Jimbo's Super Clock</PRODNAME>;  
<PART>SC45-A</PART> <PRICE>$ 199.95</PRICE>  
(<AIRF>$ 19.95</AIRF> freight /air,  
<GROUNDF>$ 7.95</GROUNDF> ground) <WARRANTY>Twenty-five  
year</WARRANTY> Warranty. Made in  
<ORIGIN>Canada</ORIGIN></ITEM>  
<ITEM><PRODNAME>Lamp Controller</PRODNAME>; <PART>LC45-  
X</PART> <PRICE>$ 25.95</PRICE> (<AIRF>$ 9.95</AIRF>  
freight /air, <GROUNDF>$ 4.95</GROUNDF> ground)  
<WARRANTY>Ten-year</WARRANTY> Warranty. Made in  
<ORIGIN>Canada</ORIGIN></ITEM>  
<ITEM><PRODNAME>Electroshock Clips</PRODNAME>;  
<PART>ES45-L</PART> <PRICE>$ 59.95</PRICE>  
(<AIRF>$ 9.95</AIRF> freight /air, <GROUNDF>$ 4.95</GROUNDF>  
ground) <WARRANTY>One-year</WARRANTY> Warranty. Made in  
<ORIGIN>USA</ORIGIN></ITEM>
```

每个 ITEM 元素包含一组其它元素,所有这些元素都可以使用 DTD 来要求它们。这些元素组在数据库中被当作行来对待,如表 4-1 所示。

表 4-1 元素组在数据库中被看作行

Prodname	Part	Price	AirF	GroundF	Warranty	Origin
Jimbo's Supper Clock	SC45 - A	\$ 199.95	\$ 19.95	\$ 7.95	25 years	Canada
Lamp	LC45 - X	\$ 25.95	\$ 9.95	\$ 4.95	10 years	Canada
Electroshock Clips	ES45 - L	\$ 59.95	\$ 9.95	\$ 4.95	1 years	USA

这些数据集可以以任何顺序保存在数据库中。信息在 XML 中的显示序列不受数据库中序列的限制,数据库不受 XML 中序列的限制。使用的序列应该看上去对每种媒体都是最合适的。使这些数据集有效工作需要比我们原来使用的构造良好的元素编写更多的代码,但这能够比使用 HTML 提供更强壮的起点。数据表格的关系模型并非总是适合 XML,因为 XML 具有更大的灵活性。但使这两种信息模型之间轻松传送信息的基础非常容易建造。

以前,使用基本文本(通常是 ASCII 码)格式的数据交换工具依靠的是固定长度域或分界符。固定长度域是针对较老的技术而设计的,例如将其所有信息存放在紧凑的表格中的主框架、如果信息适合该容器并料定它永远不会超出预设的最大长度,对于这类情形,使用固定长度域无疑很方便。接收的机器需要知道各个域之间的边界位置,然后将接收到的字符串截取为适合为该域的长度。但是,往往一个字符的遗漏就会导致整个数据作废。分界符使用另外一种不同的技术指定边界,经在各个域之间插入一个事先约定的字符(如逗号),又在各记录之间插入另一个字符(通常是回车字符)。包含数据的文件的第一行通常是域名列表,它们本身是被分界过的。分界符现在仍旧很流行,甚至象 Internet Explorer 4.0 的表格数据控制(TDC)这样的用于数据显示的最新工具都依赖于分界的文本数据。

但是,对于结构化不强的信息或不是严格的文本或数字的数据,这两种方法都不十分有用(如果二进制文件很大的话,在其中有些地方就有可能插入分界符,这样就会给试图读取该文件的程序带来问题。这种事情也会发生在 XML 身上,为此 XML 还专门提供了相关机制以使其不会有合法的或构造良好的 XML 上发生)。这两种方法都可以充当管理信息传送的临时性技术,但 XML 发誓提供更加灵活并真正更加可靠的机制以用于信息传送。

考虑以下数据表格:

FirstName	LastName	ClassName
John	Nickelson	Introductory French
John	Nickelson	Introductory Geometry
Sarah	Angleton	Advanced Calculus
Carrie	Milton	Introductory French

Carrie	Milton	Advanced Calculus
Timothy	Shore	Introductory Geometry

如果将其输送到一个文本文件,用逗号给其分界,该文本文件的信息就会成为:

```

FirstName, LastName, ClassName
John, Nickelson, Introductory French
John, Nickelson, Introductory Geometry
Sarah, Angleton, Advanced Calculus
Carrie, Milton, Introductory French
Carrie, Milton, Advanced Calculus
Timothy, Shore, Introductory Geometry

```

在 XML 文件中,这些信息就会与对人和计算机有意义的其它信息混合在一起,如下所示:

```

<? xml version="1.0" standalone="yes"? >
<HEADER><HEADLINE>Alert! </HEADLINE>
To:<TO>All teachers</TO>
From:<FROM>Registrar's Office</FROM>
Re:<SUBJECT>Students left off rosters</SUBJECT>
Date:<DATE>9 /13 /1997</DATE></HEADER>
<MESSAGE>The following students were inadvertently left off the course lists previously distributed. Please add them to your lists. If you have any questions, please contact the Registrar's Office.</MESSAGE>
<COURSELIST>
<COURSEITEM>
<STUDENT IDNUM="A0653B"><FIRSTNAME>John</FIRSTNAME> <LASTNAME>Nickelson</LASTNAME></STUDENT> <CLASSNAME>Introductory French</CLASSNAME>
</COURSEITEM>
<COURSEITEM>
<STUDENT IDNUM="A0653B"><FIRSTNAME>John</FIRSTNAME> <LASTNAME>Nickelson</LASTNAME></STUDENT> <CLASSNAME>Introductory Geometry</CLASSNAME>
</COURSEITEM>
<COURSEITEM>
<STUDENT IDNUM="A0653C"><FIRSTNAME>Sarah</FIRSTNAME> <LASTNAME>Angleton</LASTNAME></STUDENT> <CLASSNAME>Advanced Calculus</CLASSNAME>
</COURSEITEM>
<COURSEITEM>
<STUDENT IDNUM="A0653D"><FIRSTNAME>Carrie</FIRSTNAME> <LASTNAME>Milton</LASTNAME></STUDENT> <CLASSNAME>Introductory French</CLASSNAME>
</COURSEITEM>

```

```
<COURSEITEM>
<STUDENT IDNUM = "A0653D"><FIRSTNAME>Carrie</FIRSTNAME> <LASTNAME>Mil-
ton</LASTNAME></STUDENT> <CLASSNAME>Advanced Calculus</CLASSNAME>
</COURSEITEM>
<COURSEITEM>
<STUDENT IDNUM = "A0653E"><FIRSTNAME>Timothy</FIRSTNAME> <LASTNAME>
Shore</LASTNAME></STUDENT> <CLASSNAME>Introductory Geometry</CLASSNAME>
</COURSEITEM>
</COURSELIST>
```

以上 XML 文件包含的信息,即便是其中携带与前面的分界文本文件信息相同的部分,也看上去非常冗长。但它能够满足多种需要。在样式表的帮助下,该文档可被张贴到 Web 站点,被打印出来,或者被以电子邮件的形式发送给启用 MIME 的邮件阅读程序。虽然直接发送给人非常重要,该文档的自动潜能比这还要巨大。它既提供文档结构又提供数据结构,二者互相叠加,互相加强。商业信笺的传统样式仍旧有所保留,HEADER 元素包含的信息就是通常出现在商业信笺的头部的。但是,该信息被加以标记以指示较少量的数据,使文档管理系统能够保存该信笺并提供多条途径对其进行访问,读者可以通过按收件人、发件人、日期、主题甚至标题排序在列表中找到它。对其进行检索可找到 FirstName 为“Carrie”或选修“Introductory Geometry”(几何引论)课程的学生们的有关信息。

维护学生信息数据库的老师,可以连接 COURSELIST 元素中提供的 XML 信息,并直接载入列表,而不需要重复键入学生的名字。一个好的分析程序甚至能从 STUDENT 元素的 IDNUM 属性中,抓取学生的 ID 号,并将它与姓名和班级号保存在一起,使维护数据库的老师能够连接到中心数据库中的其它信息。最激动人心的可能是该数据库能够自动地产生这些文件,从而使管理员无须保持永久连接或将每个人都接到庞大的中心系统,就能将信息推送到老师自己的数据库中。如果你能联想到在完全不同的系统和应用程序之间可靠地传送信息的剪切—粘贴机制,就会看到 XML 在这方面的光明前途。

然而这些系统现在还不存在,即使在广泛使用 SGML 的少数几个地方,也还没有通过商业信笺将中心文档中的数据连接到分散的数据库。这一水平的办公自动化将建造许多新的数据驱动工作流应用,并最终减少重复性数据条目的数量(即使在今天普遍存在的计算环境中这也是一项经常性的任务)。根据 Byte 杂志 1997 年 9 月公布的结果,目前有 90% 的商业数据以信笺、电子表格、信函、提议、档案和其它信息形式存在于数据库之外。将这些文档与文档管理系统(与数据库相对)连接起来是 XML 的承诺。使文档管理系统具有一定意义将需要建造基础设施的大量工作,而其中最重要的部分是创建 DTD,由它提供文档中所包含内容的有关信息。

开发反映数据结构的 DTD 经常比开发文档结构难。象关系数据库一样,数据结构在高度结构化的环境中非常清楚,但在一般文档中却是非常模糊的。决定哪些算

作数据和寻找将其有意义地標示出来的方法是两件困难的任务。一个组织内的不同部门可能以非常不同的方式应用数据。例如,对于运输部门来说,以一定顺序列表的单个部件(产品)可能是关键的信息。但公司的管理部门却只对其总数感兴趣。优先考虑的因素不同导致对数据结构和数据管理的提议也不同,这与在信息技术的其它应用领域类似。

在 XML 中漏掉的一个关键性元素是数据类型。许多开发人员,特别是数据库开发人员和编程人员,已经发现这一“遗漏”的特征是一大缺陷。这一问题经常出现在 XML 开发邮件列表中,这可能是 XML 由于起源于文档处理(不是数据处理)而背负的最重的负担。因为 SGML 世界全力以赴地面向文档、文档管理和使文档展示更加容易,所以数据类型从来没有成为它所关心的问题。XML 1.0 作为 SGML 的一个子集很难添加这类特征,也不过在最近才提出了“模式”这一提议(本章最后将对此进行讨论)。编程人员(而不是 Web 开发人员和其他文档创建人员)对 XML 的早期攻击是认为 XML“装备”不伍,不好使用,现在又不断出现新的攻击措词。

注意

XML 的确通过注解的方法为描述数据类型提供了一些支持,这在下一节中将进行讨论。至于“注解”,有许多关于它的争议。普遍认为它是不够规范的,适合于描述大宗信息(如文件格式)的类型,而对于小块儿信息(如整数)则做得不够好。在“模式”真正到来之前“注解”可能是有用的权宜之计。

尽管有些混乱,但在决定哪些数据片断应该使用自己的元素和它们应该怎样分解开来时,基本的数据设计规则仍旧是有用的。数据始终应被分解成所需要的最小的部分,就跟在创建正规的关系数据库时那样。前一个例子可以把名和姓作为一个元素(NAME),而不是两个元素(FIRSTNAME 和 LASTNAME)。这对于文档创建人员来说肯定会比较容易,因为他们必须单独地为每一个元素作标记。但这会给需要通过“姓”来给班级列表排序的其他人带来麻烦。通过把这些较小的片断嵌入容器元素(在这个例子中是 STUDENT 元素)中可以建造出更复杂的结构。如果信息来自数据库,那么这一结构很容易实现自动化(不论是将数据输出到 XML 还是从 XML 加载进来)。被迫应付复杂的嵌套标记的手工编码人员和文档创作人员可能会不同意,他们要求各种情形的折中。为满足不同的用户,对在较大的容器元素中嵌入子元素进行折中可能是必需的。

使这些数据结构工作要求不光仅仅创建 DTD,还要求开发人员不断与他们的用户群体进行协商。很幸运能够为自己建造标准的开发人员毕竟在 XML 群体中只占一小部分。使 XML 基于内容文档的承诺成为现实要求很高的“政治”技巧和技术技能,并要求一支能同时处理好两方(开发人员和用户)关系的队伍。

元素和属性

在开发 DTD 中经常遇到有关元素和属性的问题。HTML 为使其格式准确而要

求许多属性,HTML 开发人员总是习惯于操作属性。尽管如此,在 XML 中限制使用属性(除非它们对达到某个特定的目标有用)可能会更好。创建具有良好构造的 XML 的开发人员不可使用许多属性(HTML 的 STYLE 属性除外),但创建 DTD 的开发人员可能需要经常处理这一问题。

属性是一个优秀的工具,用来将有关你的元素的准确信息传送给自动处理器——分析程序、浏览器或转换工具。但它们不是存放数据的好地方。在需要存放数据时使用的属性(例如, <STUDENT FIRSTNAME="John" LASTNAME="Nickel-son"> </STUDENT>)仅当 XML 在两台计算机之间传送数据时工作,如果用户打算在浏览器将它打开,只会在屏幕上产生空格。另外,你会失去在该属性内部嵌入更多信息的机会。元素可以包含其它元素,但属性只能包含五个值。上面使用的 DINUM 属性是对属性的正确运用。IDNUM 是只与某个地方的中心数据库有关联的学生的十六进制识别符。它不应该是便笺的可视内容的组成部分,但对于数据库来说是有用的,因为它使数据库能够连接到起始数据源并收集更多的信息。一般说来,你应该使用属性来存放对人没有直接用处但可以帮助计算机正确处理该元素的那些信息。否则,你会给将来的维护工作带来麻烦。

提示



记住,你不需要使用属性向用户隐藏信息。将 CSS 的显示属性设置为“none”就可以不让元素出现的浏览器窗口中。XSL 也有类似的机制。

XML 和 SGML

关于通用文件格式的承诺已不是什么新鲜事儿。SGML 已经在过去的 15 年中发誓要有类似的突破,但除了在一些大型组织(如 ISR、国防部、IBM)中得到重要应用外并没有什么明显的迹象。SGML 能够做 XML 所能做的任何事情,而且远远不止这些。尽管它非常强大,但只有一种 SGML(即 HTML)引起了公众的广泛关注,而 HTML 只包含 SGML 的一点点能力。那么 XML 有什么不同呢?

XML 有几个特点优于 SGML。它最大的优点是 HTML 已经为其铺平了道路。象嵌入元素和属性这一类语法现在已为许多人所熟知,而他们在五年前还没有听说过这一切。这些都是 HTML 的功劳。但 HTML 开发人员经常因为他们不得不使用的工具对其所加的限制而苦恼,希望他们的工具能有重大改善。通过让 Web 开发人员能够创建满足自己的规范(而不是浏览器开发商的规范)的文档,XML 和 GSS 提供了在 HTML 之上的改进。尽管 XML 更是一种用于 Web 开发的工具,但开发人员以前使用 Web 的经验为 XML 进入许多组织打开了门户。

XML 的第二个优势是它比 SGML 简单许多。尽管它可能会不断扩展并最终变得象其父辈(SGML)那样庞大,但 XML 要求开发人员一开始需要学习的东西比 SGML 大大地减少,也只要开发人员掌握不多的细节。虽然开发 XML 的 W3C 工作组是从 SGML 工作组分化出来的,但 XML 开发人员看上去坚决不会象 SGML 那么

复杂。XML 发誓成为普通用户能够理解的标记语言标准。

XML 具有的最后一个重大优势是支持 SGML 的用户群体,他们看上去很有兴趣推广 SGML 的这个新的后代。目前 SGML 群体正在调整一些 SGML 标准以使 XML 能够更容易遵循。虽然许多 SGML 仍旧是高层次的教科书,书价超过 50 美元,但 SGML 的知识已经渗透到更多的普及型书籍中,其中包括一些商用计算机书籍。了解该产品(SGML)的文档管理用途的顾问人员能够在需要的时候向公司提供服务。当然,具有重大区别的 SGML 和 HTML 业界能否在二者的交叉领域携手合作,现在还不明朗。

交叉参照



如果你想学习以前的 SGML 实现,请参考 Chet Ensign 编写的《SGML: The Billion Dollar Secret》(SGML: 十亿美元的秘密)一书,该书是 Prentice - Hall 出版社于 1997 年出版的。

即将到来的“模式”

DTD 在描述文档结构方面做得很好,但由于众多原因它们已受到非议并可能最终成为陈旧的技术。DTD 被其批评者认为做得太多或太少,其语法为 SGML 用户所熟识但对其他人来说却是怪异的。说 DTD 做得太多是因为它们为提供文档内容和描述文档结构提供了工具。说 DTD 做得太少是因为它们几乎没有为描述文档内容(如前面提到的数据类型)提供工具。XML 1.0 中的 DTD 语法显然受到 XML 与 SGML 兼容的限制,要求使用 SGML 的 DTD 语法的子集。

已经出现许多要求取代或补充 DTD 的建议,W3C 的 XML 工作组已经筹建了一个模式工作组。一个由 Microsoft、ArborText 和 DataChannel 三大公司共同提议的模式 XML-Data(见站点 <http://www.w3.org/TR/1998/NOTE-XML-data/>)早在 XML 1.0 完成之前就出现了。文档内容描述(DCD,见站点 <http://www.w3.org/TR/NOTE-dcd>)是一个较近期的后继者,是由 Microsoft 和 IBM 公司提出的。XML-Data 和 DCD 都提供取代 DTD 的一种“完善”的解决方法,包括支持实体声明和数据类型。在 1998 年夏天,XML-Dev 邮件列表的成员就创建了他们自己的模式语言,然后将其叫做 XSchema,但在提交给 W3C 时叫做文档描述标记语言(DOML,见 <http://purl.oclc.org/net/doml>)。与 XML-Data 和 DCD 不同,DDML 期望实体声明会有自己的标准,并计划采用更加模块化的方法在其它数据类型标准之一进行建造。另一个提议是面向对象的 XML 模式(SOX,见 <http://www.w3.org/TR/NOTE-SOX>),它采用面向对象的编程方法进行模式的开发。这些提议都没有得到普遍使用或完全的实现,但它们都提供了使用其它 XML 文档来描述 XML 文档的工具。在 W3C 中正处于发展阶段的另一个工具是资源描述框架(RDF),它可以用于描述文档结构,但它的目标是针对元数据的。

交叉参考



要想了解 XML-Dev 邮件列表的有关信息,请见位于站点 <http://www.lists.ic.ac.uk/hypemail/xml-dev/> 处的档案。

目前很值得学习 DTD, 尽管它使用了许多奇怪的特征和不常见的语法。你将能够更好地理解出现的模式提议, 而且用来将 DTD 转换成模式的工具也会在模式的推荐提议出现之后紧跟着到来。不管你喜不喜欢 DTD, 它们仍旧是 XML 的基础, 并无疑是用于描述其后继者或补充性技术的符号集的重要组成部分。

为处理做规划

XML 在许多方面比 HTML 和 SGML“大”, 它能够进入更多的计算领域, 包括 Web 浏览器、文字处理程序、资源文件、数据库和控制系统等领域。因此, 开发人员需要更加小心——为一种目的设计的结构最后被处理时有可能完全用于另一种目的。XML 不再将数据永远粘附在特定的应用程序上。其简单明确的文档结构使应用程序能够走进任何 XML 文档并对它进行处理。XML 创建的强大通用框架可以存放类型完全不同的信息。虽然这不一定是每个创建 XML 文档结构的人的目标(毕竟近期的目标仍旧很重要), 但在开发 XML 文档结构和应用程序的过程中应该始终记住这一点。

Web 开发人员、文档创作人员和管理人员以及应用编程人员现在已经因为缺乏工具而受阻。XML 创建仍旧是开创性的工作, 开发人员可以借助的东西还很少, 原因是支持 XML 的程序才刚刚开始出现。尽管如此, XML 的最具魅力的特征之一是今天的文档可以跟明天的应用程序一块儿工作。基本的元素和属性结构是固定的, 但它们非常灵活, 足以满足各种各样的需要。XML 开发人员可以从 SGML 世界获得丰富的示例, 并从主要的销售商那里得到日益增多的 XML 支持。

交叉参考



需要详细了解使用 XML 建造大型文档标准的开发人员可以参考《Developing SGML DTDs: From Text to Model to Markup》一书(建造 SGML 文档类型定义: 从文本到模型再到标记, 作者是 Eve Maler 和 Jeanne El Andaloussi, Prentice-Hall 出版社 1996 年版)。尽管该书是关于 SGML 的(它问世的时候还没有 XML), 但它提供了在使用 SGML 建造标准和应用时有关信息建模及其过程的许多有用信息。

第 五 章

文档类型定义

到现在为止我们已经学习了 XML 文档创建的许多理论方面的知识,现在是打开这个工具箱的时候了。尽管创建被检验 XML 的工具看上去有一点儿怪异,但它们背后的逻辑实际上并没有那么复杂。创建文档类型定义不是一个容易的过程,但一个写得很好的 DTD(文档类型定义)会让你觉得对所做的这些工作是值得的。DTD 提供的一套有用的特征可以减小 XML 文档的大小。即使你不计划编写自己的 DTD,了解如何阅读 DTD 对于你在深夜从事用写的不好的 DTD 创建的文档时,也是可以派上用场的。

当心

这里描述的许多特征是在几个不同的地方工作的。有一些只在外部的 DTD 中工作,有一些只在 DTD(内部的或外部的)中工作,还有一些只在文档中工作。SGML 开发人员更需要细心,因为在 SGML 中各种地方都工作的一些工具在 XML 中被限制为仅仅提供原来功能的一个子集。

分析简介

分析基本上是对文本的解释。计算机实际上不能阅读,但它们能够解释文本文件。标记语言只不过是辅助这种解释,向计算机(偶尔也向人)清楚地指出文本块的本质。在 HTML 中这一点更加明显:在起始标记和结束标记之间放入文本意味着该文本将被以特定的方式格式化。<I>This is italic</I> 产生的结果是倾斜的 *This is italic*。HTML 浏览器理解位于<I>和</I>之间的所有字符应该显示为斜体。

SGML 和 XML 采取更高级的方法来解释文本。HTML 浏览器根据硬性规定的一套规则来解释文本,这些规则是浏览器开发商根据他们对 HTML 的解释及与此有关的各种标准来创建的。HTML 浏览器尽其最大能力对文本进行解释,这在以前已

经讲过。而 XML 和 SGML 分析程序则要检查文档的标记以确保它们符合一套规则。XML 分析程序最起码要检查文档的良好构造性,这是最少的一套规则。SGML 和 XML 都可以要求文档遵循在文档类型定义中描述的一组复杂的规范。符合 DTD 的文档被说成是有效的;能够解释 DTD 并根据它们的规定检查文档结构的分析程序叫做检验分析程序。

XML1.0 推荐版本提到了阅读和解释 XML 的一个更大系统的两个组件。第一个组件是 XML 处理器,它(在 XML1.0 规范本身之外)一般叫作分析程序。XML 处理器的工作是加载 XML 文件和它们需要的任何支持文件,检查它们是否遵循必要的规则,并建造能够传递给应用程序的文档树结构。XML 处理器的规则因处理器是否检验而略有不同。基本上,检验分析程序既检查语法又检查文档结构。应用程序是处理该树结构的系统的一部分,它处理树结构包含的数据。应用程序可以是在屏幕上显示树结构中的信息浏览器,或者是使用样式表产生更漂亮视图的浏览器,也可以是为打印机格式化信息的打印应用程序。它还可以是为盲人用户将计算机化的文本转换成声音的读者应用程序。该应用程序不一定非得产生人类能够阅读的输出信息,而是将 XML 当作控制信息来对待,用于控制机器工具或一组急需运输的定单。XML 应用程序能够实现依赖数据的任何过程。

这种标记语法(由分析程序处理)和格式(由应用程序处理)的分离可以使解释 XML 比解释 HTML 更加复杂。<I>标记在任何 HTML 浏览器上都具有相同的意义——开始使用斜体,但在 XML 中就没有这么简单。<I>可以表示开始使用斜体,也可以表示一种冰淇淋味道,或者表示对 IBM 公司的评价。实际上,它可以表示任何东西,这完全取决于应用程序如何对其解释。在 HTML 中,浏览器结合了分析程序和应用程序并遵循一级有关如何解释标记的比较严格的规则。尽管 XML 对标记本身的要求比 HTML 严格许多,但它对于标记数据的最终解释却相当灵活。

创建 DTD 是建造健壮的应用所必需的步骤。DTD 提供了关键的信息以使 XML 处理器能够分析代码并确定它包含了应用程序需要的所有信息,它所采取的形式也是应用程序所能接受的。DTD 在给 XML 处理的数据文件和自 XML 处理器发送到应用程序的数据之间提供了关键性的连接。DTD 帮助计算机理解对人类来说是非常清楚的结构。在本章和全书中,讨论的焦点是关于创建 DTD 和有效的文档。因为 XML 还是崭新的事物,所以有关它的应用不是很多。尽管本书将提供几个路线图,但开发使用 XML 的完善的应用程序,不属于本书作为入门读物的范畴。在本章中我们将使用 Microsoft 的 Internet Explorer 5.0(它包含有分析程序)来产生 XML 代码。

提示

可以使用许多种分析程序对 XML 进行处理。在第三章我们已经看到过 Lark

分析程序,它是一个非检验分析程序,能够检查文档的良好构造性但不能对照 DTD 对其进行检验。一个向 Lark 添加检验功能的附加模块——Larval,现在已在 Lark 的发布中出现。此外,还有其它一些分析程序。SP(见站点 <http://www.jclark.com>)是一个检验 SGML 的分析程序,它也能够分析 XML 文档。SP 的作者 James Clark 还创建了作为 Perl 的 XML 支持和 Netscape 的 Mozilla 浏览器项目核心的 Expat,这是一个用 C 开发的分析程序。Aelfred 是来自 Microstar 公司的一个小巧的非检验分析程序(该公司的站点是 <http://www.microstar.com>),是为用于小应用程序中而设计的。IBM 公司(<http://www.ibm.com/XML>)开发的 XML4J 是一个检验分析程序,它能够跟 IBM 公司提供的许多支持工具一起使用。要了解更详细列表信息,请访问 <http://www.xmlsoftware.com/parsers>。如果你不是创作人员,而只是一名读者,那么你将使用更加友好的浏览器或其它应用程序;但对于编程人员来说,这些免费的分析程序是很有价值的工具。

从最简单的开始

建造 DTD 的细节即使对有经验的 HTML 编码人员、SQL 开发人员和 C++ 及 Java 程序员来讲都是很令人胆怯的。XML 已经降低了 SGML 的复杂度,但它仍然有一些奇怪的地方。XML 标准的各个部分总是互相引用,要求在阅读它们时经常来回翻页。为避免突然接触复杂的事物不好理解掌握,我们将从简单的文档开始,以此演示 XML 的一些工具。这一部分的示例中使用的许多 XML 在此不作深入的解释。详细的解释将在下一部分内容中出现。不幸的是,除非你亲眼看到一些实际运用中的 XML,否则这些解释不会产生多大的效果。因此,这一部分仅让你对 XML 文档的外貌有一个总体的了解,而不解释其细节。在列举下面的例子时进度会比较快,因此请你记住这只是一次快速的“游览”,目的是让你看看合法的 XML 文档是什么样子。

注意

尽管你可以操作这些例子并检验它们,但这一部分内容主要是以粗糙的形式向你展示 XML 文档和 DTD 的样子。本章所有的例子都可以从 <http://www.simon-stl.com/xmlprim2/> 处下载。

一开始,我们的例子使用一个内部 DTD。DTD 可以出现在它们描述的文档中或单独的文件中。多数大规模工程使用存放在集中式文件结构中的外部 DTD,但我们这个简单的文档不可能用大规模的系统来管理。该文档以 XML 声明作为开始,然后是包含一些元素、属性和实体的文档类型声明。如下所示:

```
<? xml version = "1.0" encoding = "UTF-8"? >
<! DOCTYPE DOCUMENT [
```

```
<! ELEMENT DOCUMENT ( #PCDATA) >
<! ENTITY Description "This is entity content." >
]>
<DOCUMENT>This is element text and an entity follows: &Description; < /DOCUMENT>
```

Internet Explorer 5.0 将让我们检验这一文档,当然这还需要做些工作。默认时,Internet Explorer 5.0 仅检查构造良好性,还必须将 JavaScript 开关设置成打开检验。因此,如果想对照 DTD 检验文档,必须遵循下面几个步骤(总的来讲,在 IE 5.0 中检验比在命令行上使用原始的分析程序检验要容易许多)。

首先,必须获得执行检验的代码。如果你的 XML 页面是在 HTTP 服务器上,那么可以直接从 Microsoft 的 Web 网页进行检验。否则,必须将页面的一份拷贝保存到你的系统,在 Internet Explorer 5.0 中将其作为一个文件打开,然后对它们执行检验。“检验器”可以从 Microsoft 公司等到,站点是:<http://msdn.microsoft.com/downloads/samples/internet/xml/xml-validator/default.asp>。

提示

▲ 将检验页面的一份拷贝保存到你的 XML 文件所在的目录下。这样,你只需要在这里打开该检验页面然后使用文件的相对路径(即文件的名称)。

该检验器(如图 5-1 所示)让你键入一个 URL 或粘贴一些用于检验的 XML 代码。结构将出现在页面的底部。

将前面给出的示例代码载入该检验器中会产生在图 5-2 中 Web 网页的底部显示的结果。

Internet Explorer(从技术上讲,应该是 Internet Explorer 内部的分析程序)解释声明,允许创建 DOCUMENT 元素和扩充 &Description;这一实体。这段代码没有做多少事情,但它提供了一个在此基础上可以成长出结构的基本框架。文档第一行上的 XML 声明,告诉分析程序该 XML 使用的版本号,并说明该文档的编码方式与 UTF-8 标准兼容,该标准为大多数欧洲语言包含标准的 Latin-1 字符集。位于下一行的文档类型声明(<! DOCTYPE 名称[...]>)创建一个 DOCUMENT 定义,该定义包括两个关键部分:一个元素和一个实体(要看到完整的 DTD,需要从 View 菜单中选择 Source。浏览器将它隐藏起来以显示更多的内容)。第三行上的元素声明(<! ELEMENT 名称数据>)宣布了一个能够包含被分析字符数据(#PCDATA)的 DOCUMENT 元素。第四行上的实体声明(<! ENTITY 名称实体定义>)提供了一个具体的值与名称“Description”搭配。真正的标记只包含一个 DOCUMENT 元素。这个元素包含一些文本和一个实体引用(即在上边声明的实体名称的前面加一

个“&”号,后面加一个分号)。在 IE 5 显示文档时它已经对该实体进行了扩充。在这个简单文档的基础上,我们可以创建更加复杂的类型来开始定义一个样本文档。

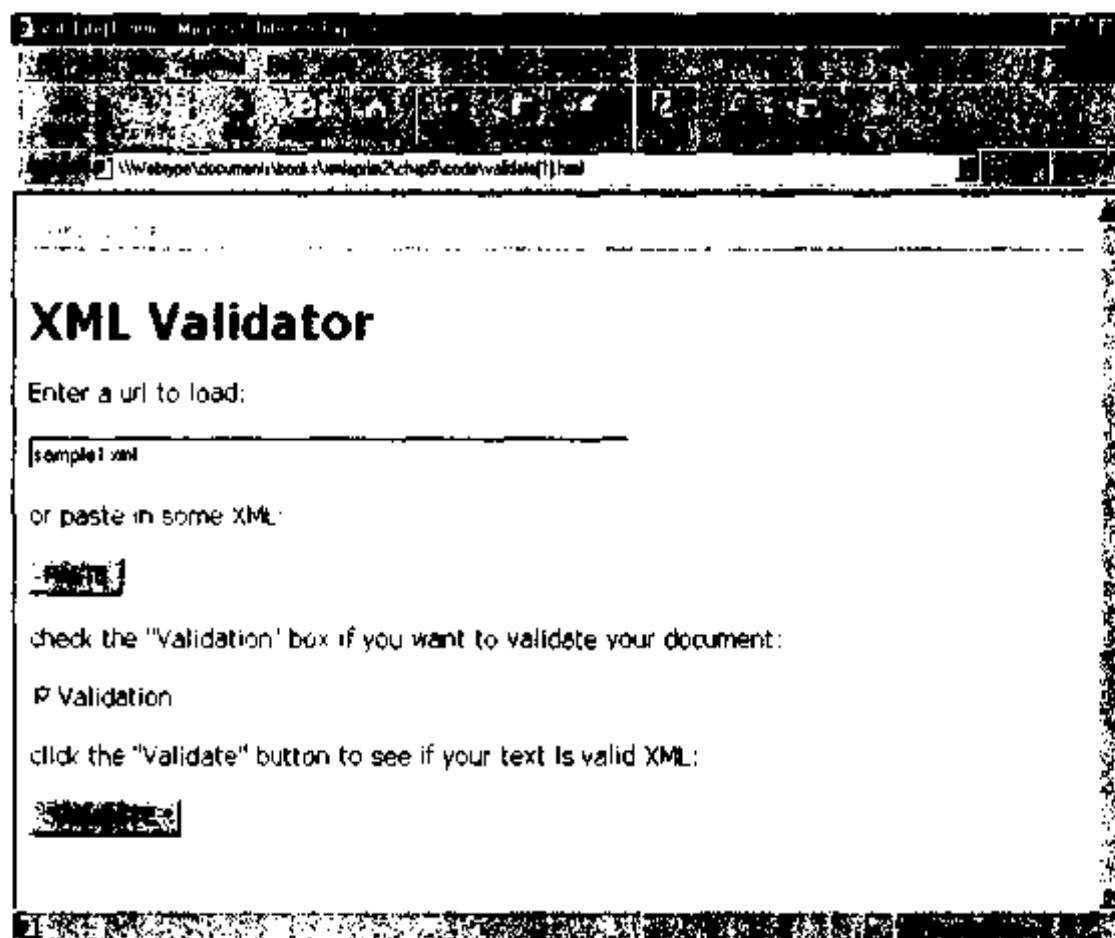


图 5-1 Internet Explorer 的检验工具必须通过一个 Web 网页来访问

下一个例子将向 DOCUMENT 元素添加几个属性,提供一些文档管理系统可以用来进行跟踪的信息。<! ATTLIST 名称数据...>声明将使 DOCUMENT 元素能够携带一个跟踪号码和一个安全级别。如下所示:

```
<? xml version="1.0" encoding="UTF-8"? >
<! DOCTYPE DOCUMENT [
<! ELEMENT DOCUMENT (#PCDATA)>
<! ATTLIST DOCUMENT
    trackNum CDATA #REQUIRED
    secLevel (unclassified|classified) "unclassified" >
<! ENTITY Description "This is a very simple sample document." >
]>
<DOCUMENT trackNum="1234"> This is element text and an entity follows: &Description; < /
DOCUMENT >
```

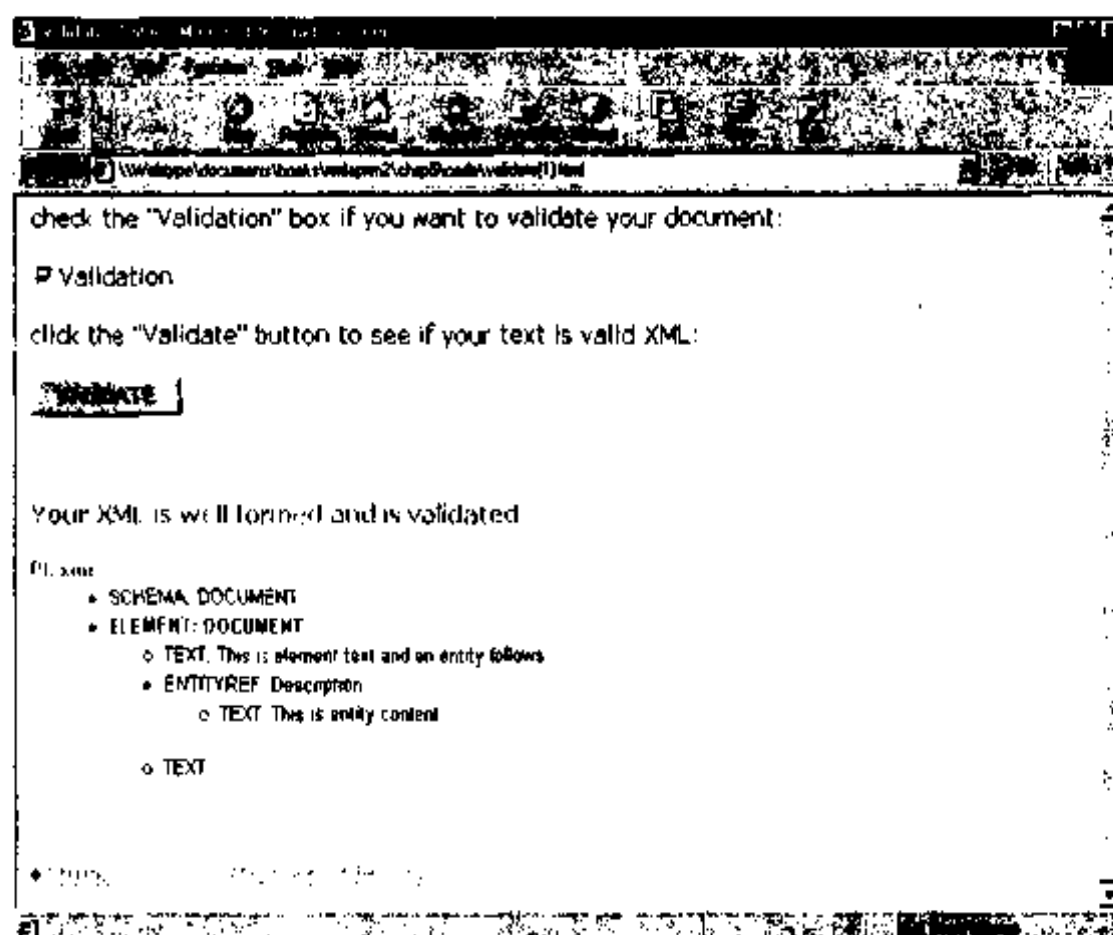


图 5-2 Internet Explorer 在检验文档时将扩充实体

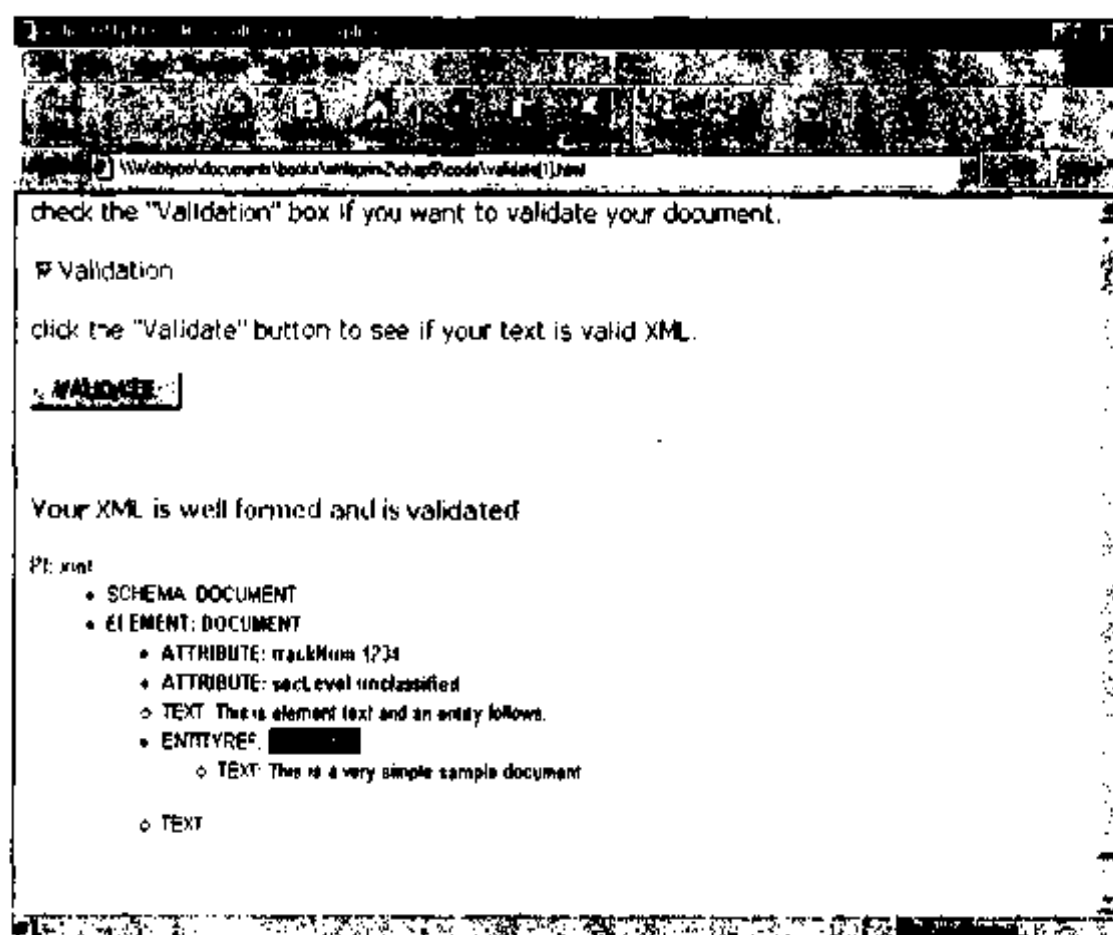


图 5-3 Internet Explorer 将为文档使用缺省的属性值

属性列表声明(<! ATTLIST 名称值>)应该位于它所引用的元素的下方,尽管从技术上讲不必这样做(因为它指出了元素的名称)。这两个属性是不同的类型。trackName 属性可以拥有类型为 CDATA 的任何值。CDATA(我们将在后面详细讨论它)是一个字符数据。大多数属性也可以是可选的(用 #IMPLIED 指示,或使用默

认值)或使用缺省时赋予它们的固定值(# FIXED 后跟缺省值)。第二个属性 secLevel 只能从两个值(unclassified 和 classified)中取一个(在 XML 标记中,符号 1 表示“或”的意思)。列表后面指定其缺省值是 unclassified。真正的文档元素包含 trackNum 但没有 secLevel 属性,因此缺省值 unclassified 将得以使用。

Internet Explorer 5.0 隐藏了 DTD,但它的作用是明显的——属性 secLevel 被以值“unclassified”显示出来,而这正是在 DTD 中声明的缺省值(即使在 DTD 中没有声明该缺省值的话它也会显示出来)。如果你向 DOCUMENT 元素添加一个值为“NONE”的 secLevel 属性,Internet Explorer 就会如图 5-4 所示报告错误。

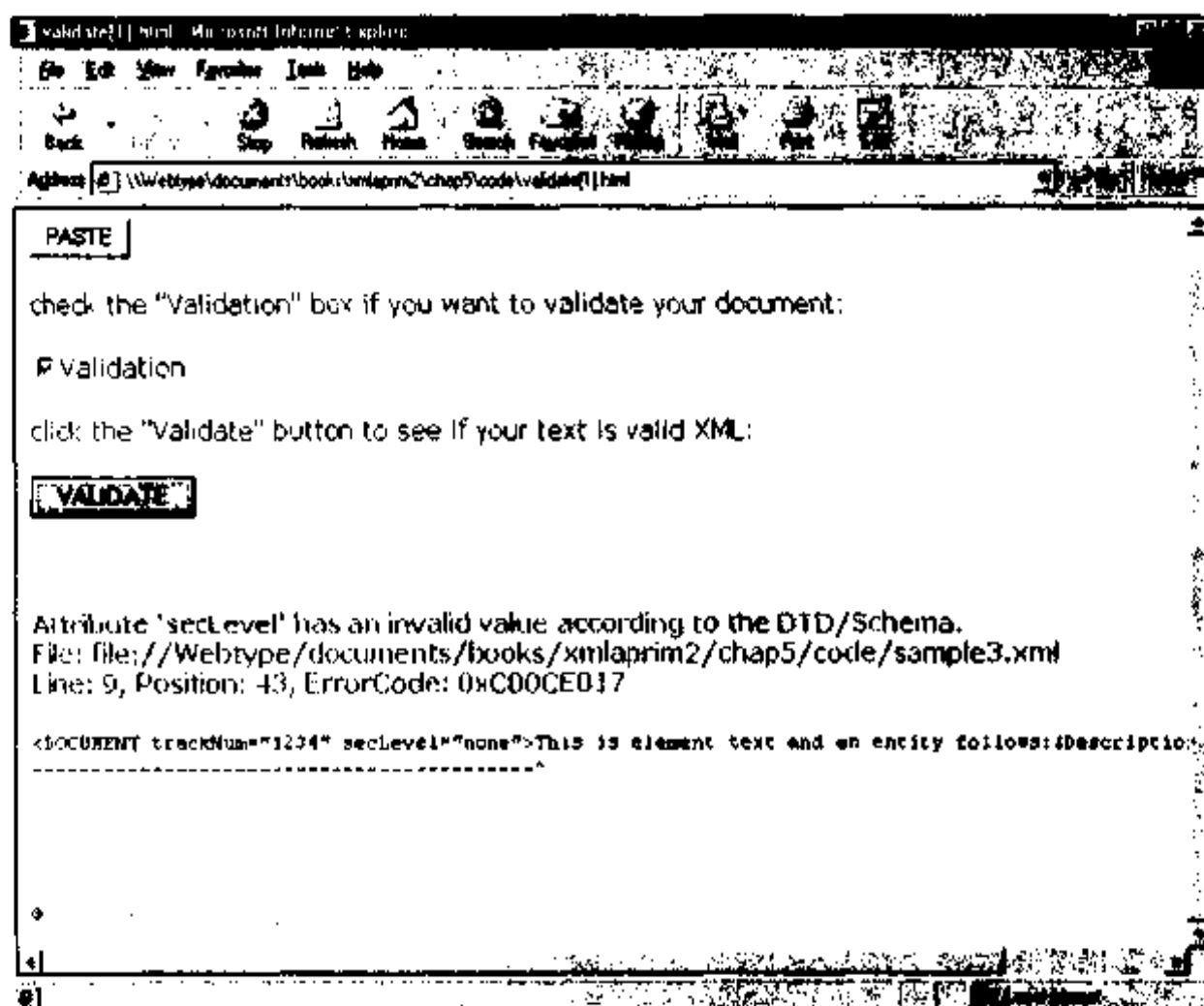


图 5-4 Internet Explorer 报告文件中的第一个检验错误

现在该示例元素已经拥有属性,让我们再给它增加几个元素。XML 能够让开发人员识别出可能包含哪些内容元素,这些元素的类型、顺序和频率。我们的例子将获得一个题目、作者和对该文档的描述。题目只能出现一次,作者域必须最少出现一次,总结元素是可选的,具体的注释元素可以出现一次或一次也不出现。所有这些元素必须按它们被列出的顺序出现。

```
<? xml version="1.0" encoding="UTF-8"? >
<! DOCTYPE DOCUMENT [
<! ELEMENT DOCUMENT (TITLE, AUTHOR+, SUMMARY*, NOTE?)>
<! ATTLIST DOCUMENT
  trackNum CDATA #REQUIRED
  secLevel (unclassified|classified) "unclassified">
```



```
<! ELEMENT TITLE (#PCDATA)>
<! ELEMENT AUTHOR (#PCDATA)>
<! ELEMENT SUMMARY (#PCDATA)>
<! ENTITY Description "This is a very simple sample document.">
]>
<DOCUMENT trackNum="1234">
<TITLE>Sample Document</TITLE>
<AUTHOR>Simon St. Laurent</AUTHOR>
<SUMMARY>This is element text and an entity follows: &Description; </SUMMARY></
DOCUMENT>
```

这段代码产生出一个比较严格的文档结构,该结构是由 DOCUMENT 元素声明的(TITLE,AUTHOR+,SUMMARY*,NOTE?)部分建造的。因为所有实体都是用逗号分隔开的,所以它们必须按所列顺序出现。由于没有后缀,TITLE 必须出现一次且只能出现一次。AUTHOR 跟的加号要求 AUTHOR 最少出现一次,可以是多次。SUMMARY 后面的星号使任意多个(包含 0 个)SUMMARY 元素都可以在这个位置出现。NOTE 后面的部号表示它是可选元素,但它只能出现一次。Internet Explorer 5.0 返回的文档结构如图 5-5 所示。

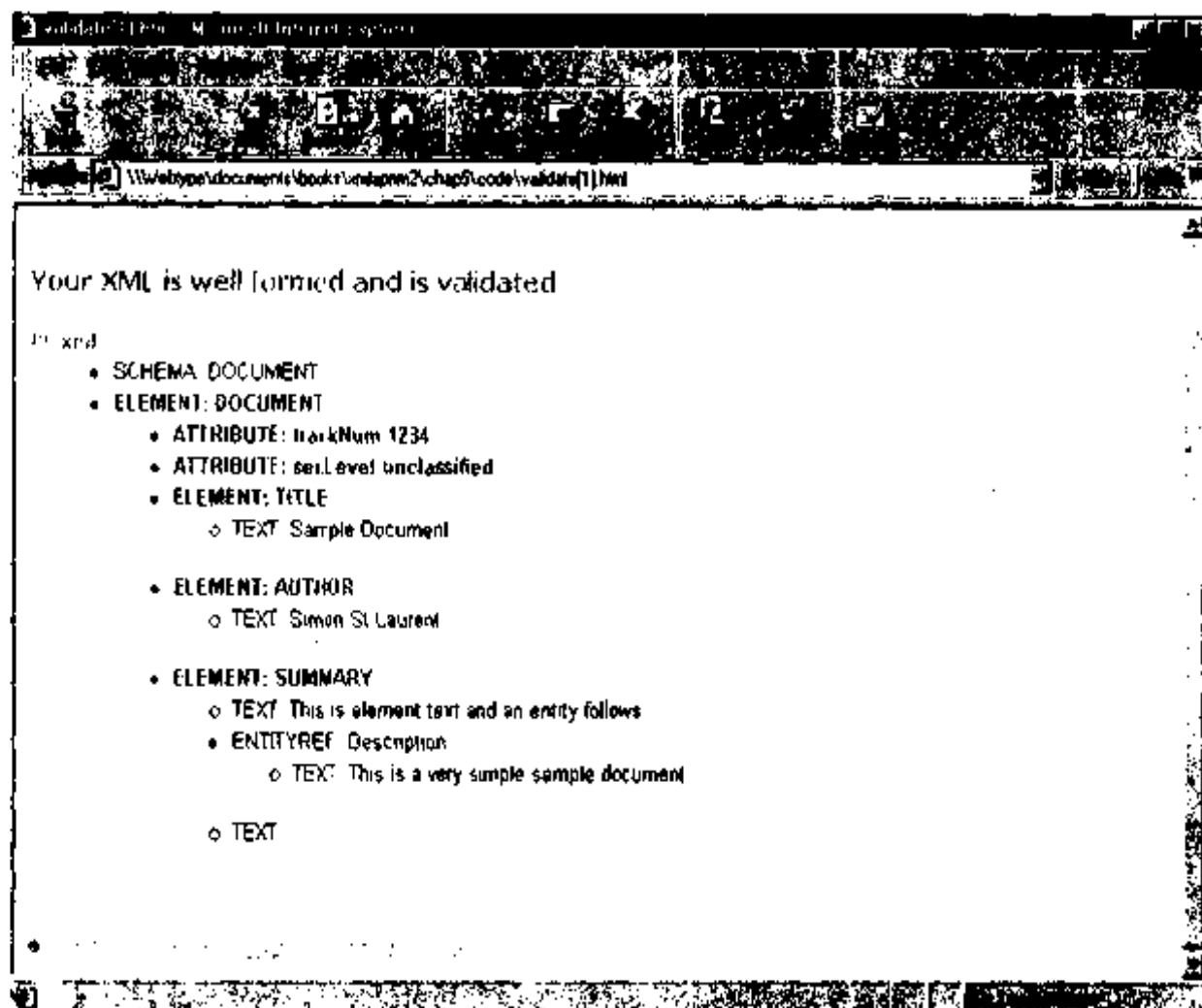


图 5-5 Internet Explorer 将检验嵌套的结构

在这个文档中改变元素的顺序将使分析程序出错,因为这样以来文档就不再遵循 DTD 中指定的结构,从而不再有效。例如,把 SUMMARY 元素移到 TITLE 元素的上边会产生图 5-6 中所示的错误。

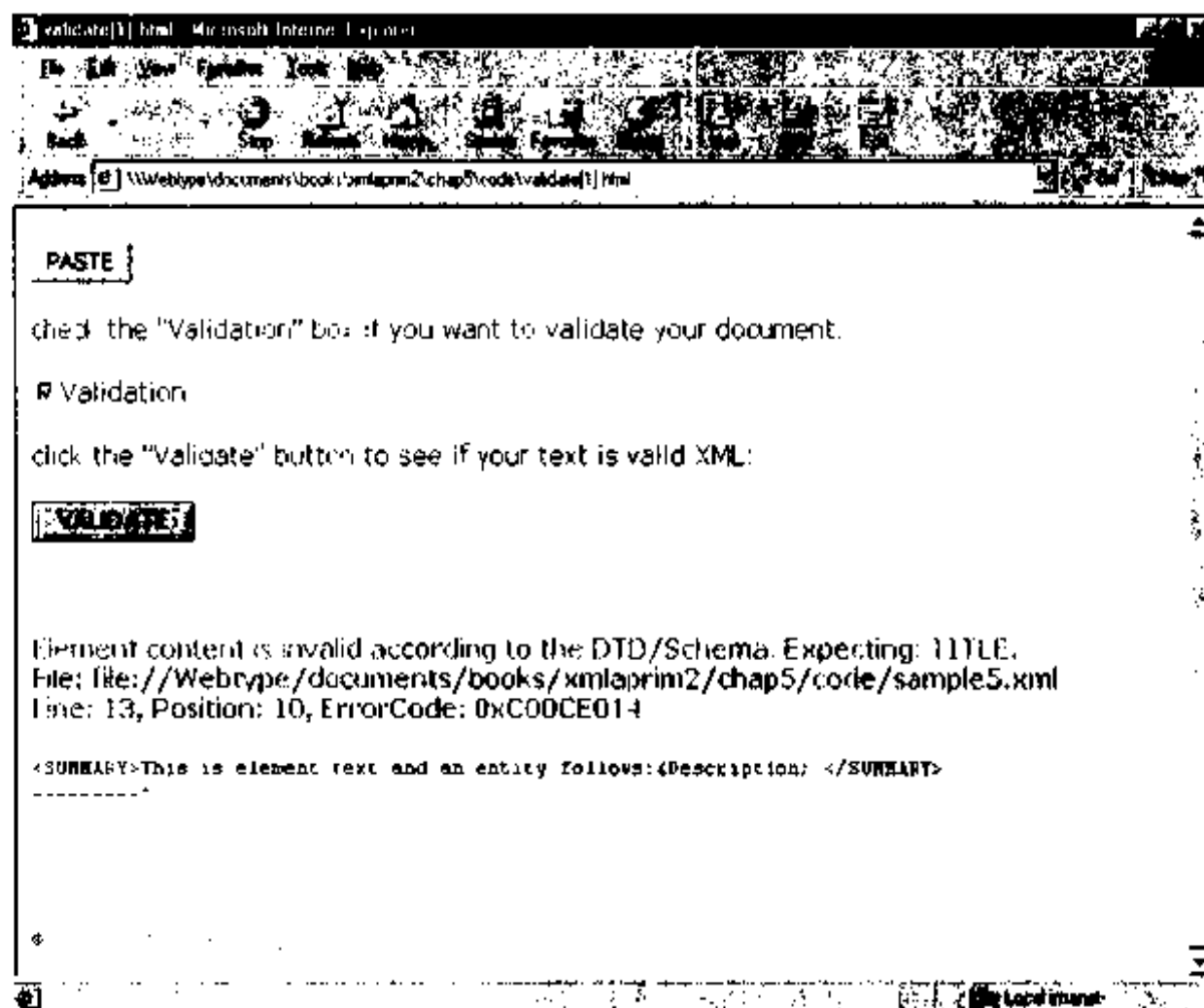


图 5-6 Internet Explorer 5.0 将报告它们检验的文档中出现文档结构错误

我们的下一个例子将把上面的 DTD 往深处扩展一层,在 AUTHOR 元素下提供更多的信息。在这里,作者信息将包含组织(公司或大学)和人名。

```
<? xml version="1.0" encoding="UTF-8"? >
<! DOCTYPE DOCUMENT [
  <! ELEMENT DOCUMENT (TITLE,AUTHOR+ ,SUMMARY* ,NOTE?)>
  <! ATTLIST DOCUMENT
    trackNum CDATA #REQUIRED
    secLevel (unclassified|classified) "unclassified">
  <! ELEMENT TITLE ( #PCDATA)>

  <! ELEMENT AUTHOR (FIRSTNAME, LASTNAME, (UNIVERSITY | COMPANY)?)>
  <! ELEMENT FIRSTNAME ( #PCDATA)>
  <! ELEMENT LASTNAME ( #PCDATA)>
  <! ELEMENT UNIVERSITY ( #PCDATA)>
  <! ELEMENT COMPANY ( #PCDATA)>
  <! ELEMENT SUMMARY ( #PCDATA)>
  <! ENTITY Description "This is a very simple sample document.">
]>
<DOCUMENT trackNum="1234">
<TITLE>Sample Document</TITLE>
<AUTHOR><FIRSTNAME>Simon</FIRSTNAME>
<LASTNAME>St. Laurent</LASTNAME>
```

```
<COMPANY>XML Mania< /COMPANY>< /AUTHOR>
<SUMMARY>This is element text and an entity follows:
&Description; < /SUMMARY>< /DOCUMENT>
```

除增加了一些元素之外,最大的变动是在 AUTHOR 元素中的声明:<! ELEMENT AUTHOR (FIRSTNAME, LASTNAME, (UNIVERSITY /COMPANY)?) >。这一声明使开发人员能够创建更加灵活的结构。在这里,AUTHOR 元素必须包含(按所列顺序)一个 FIRSTNAME 元素、一个 LASTNAME 元素、一个 UNIVERSITY 元素或一个 COMPANY 元素(两个元素同时都使用会产生分析错误)。Internet Explorer 5.0 看上去对这种安排很满意,如图 5-7 所示。

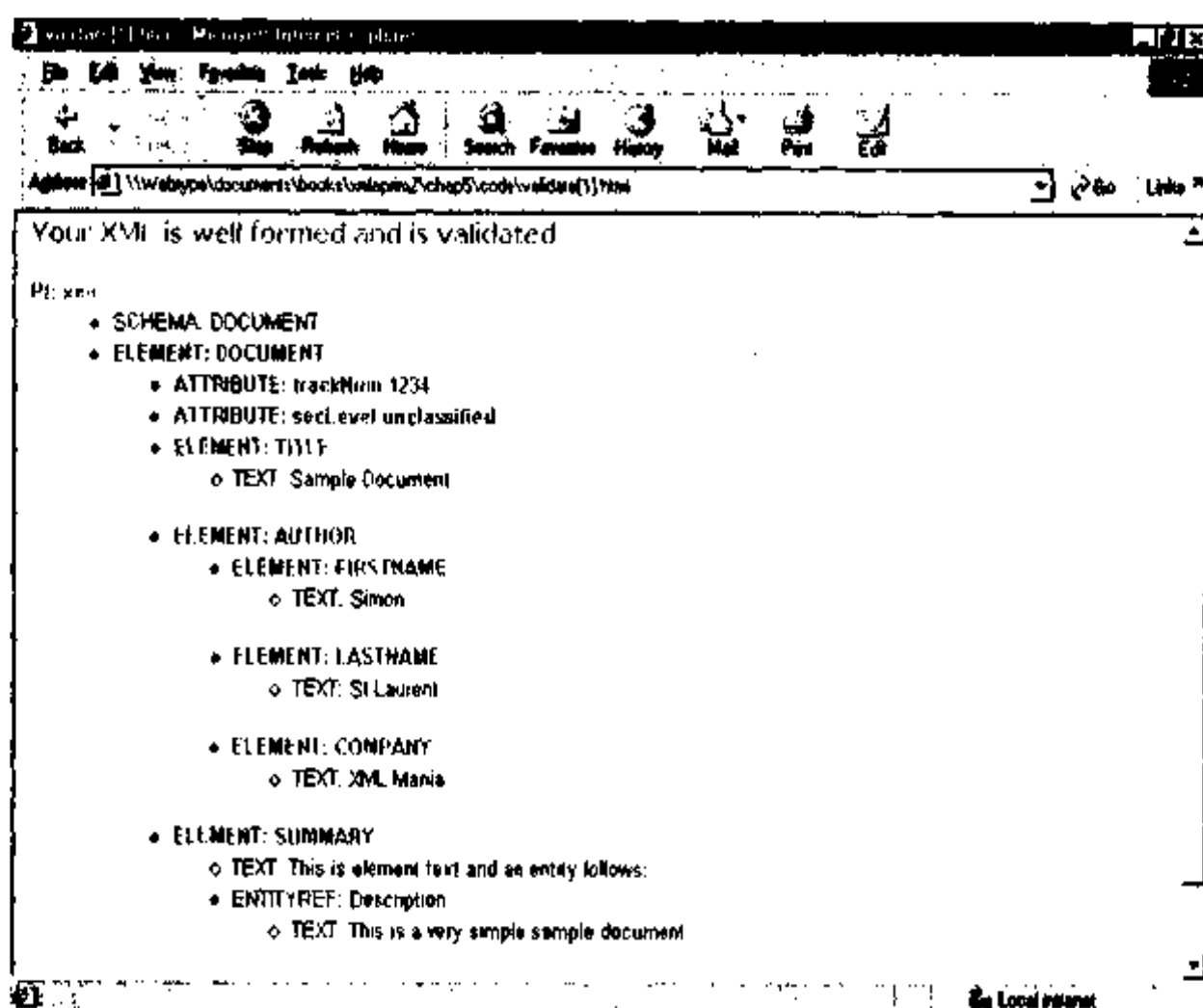


图 5-7 Internet Explorer 能够分析具有多层深度的结构

到此为止,这个文档已经有点儿长度了,大部分篇幅只是在定义文档的类型。作为最后一个例子,我们将把该 DTD 文件与实际的文档分开。这样,文档变得短了许多:

```
<? xml version="1.0" encoding="UTF-8"? >
<! DOCTYPE DOCUMENT SYSTEM "simple.dtd">
<DOCUMENT trackNum="1234">
<TITLE>Sample Document< /TITLE>
<AUTHOR><FIRSTNAME>Simon< /FIRSTNAME>
<LASTNAME>St. Laurent< /LASTNAME>
<COMPANY>XML Mania< /COMPANY>< /AUTHOR>
```


以下内容将讨论把 XML 文档连接到 DTD 所需要的技术,然后较深入地探讨 XML 数据和文档结构。

DTD 序言部分

尽管从技术角度讲一开始的序言不是 DTD 的一部分,但它包含的 `<? xml? >` 声明和后面的文档类型声明是把 DTD 与应用它们的代码联系在一起的“胶水”。这些看上去怪异的新的声明执行 HTML 中的 HEAD 元素的某些功能,但它们回答的问题不同。它们只包含一些信息片段,所有这些片段都对于告诉浏览器如何解释后面的代码非常关键。尽管 HEAD 元素可以包含一些有趣的信息,这些信息只能影响显示中的一些特定部分,如题目和一些脚本信息。指出文档中使用了哪个版本的 HTML 对于设计人员或自动化 HTML 编辑器都十分有用,但浏览器实际不会关心它的存在——因为浏览器只是按照自己的规范解释代码,而不是按照远方某个委员会制定的规范。在 XML 中,序言以比较具体的术语告诉分析程序(不是应用程序)怎样解释文档。

`<? xml? >`

有效的 XML 文档总是以 XML 声明开头的。XML 声明包含版本信息、内码信息以及文档将使用(如果使用的話)的 DTD 的有关信息。虽然 XML 声明的内容仅给出了后面的 XML 文档的大致情况,但它们向解释该文档的分析程序提供了具有关键作用的基本信息。XML 声明看上去象是处理指令,但它不是。XML 声明包括几个部分:开始端 `<? XML`、版本信息、独立性声明、内码声明以及结束端 `? >`。所有这些都不是技术上所要求的。XML 声明中可以放弃一些部分,没有 XML 声明的文档也可以是构造良好的文档。版本信息和其它声明都有分析程序可以使用的缺省值。

当心

与 HTML 元素不同,XML 声明没有关闭标记。在文档中绝对不应该出现 `< /? xml>`。XML 声明是一个开头语句,除此之外没有任何其它东西。

这第一个版本的 XML 的版本信息十分简单,就是 `version="1.0"`,`version 1.0` 是默认的版本信息,它是未来 XML 的所有实现的基础。不管将来该标准有何变化,让版本信息缺省或指定版本号 1.0,都意味着为版本 1.0 编写的文档和文档类型定义被按照最初编写它们时的意图进行解释,不论 XML 已达到 7.3 版本甚至 20.0 版本

都会这样。

注意

跟 XML 不一样,HTML 在为公众采用时其版本是 0.9,而 XML 的工作草案首次亮相时已经是 1.0 版本。

独立性声明宣布文档是否包含一个引用外部文档类型的声明。它的取值是“yes”或“no”。如果没有出现独立性声明,缺省值为“no”。有效的文档必须在这一声明中提供真实的回答。文档可以引用外部实体,仍旧声称“yes”,但不可以引用外部 DTD。W3C 推荐标准中,建议任何文档都可以在需要时转换成独立的文档,有些简单的应用程序可经选择拒绝所有的非独立文档。但一般说来,建造成套有效文档的开发人员最可能回答的是“no”,或者干脆略去这一声明。

当心

对大多数人来说,独立性声明更可能引起混乱而不是解决问题。你应该在完全确定文档没有依赖于任何外部 DTD 资源或实体信息时使用独立性声明,但在这些情况下实际上不用作此声明。W3C 正在考虑规范 XML 文档,要求它们必须是独立的,这样的话将使这一特征具有更好的用途。在此之前最好不要使用独立性声明。要了解更多的信息请见本章末尾对分析程序行为的讨论。

内码声明解决了与国际化有关的复杂问题。XML 使开发人员能够指定多种不同字符内码模式中的哪一种应该用于某个文档。缺省时的默认内码模式是 UTF-8,它对于 ASCII 字符集使用从 0 到 127 之间的值,从而包含英语中使用的大多数字符的直接表示。它还可为具有更高值的 Unicode 字符提供多个字节的内码。UCS-2 是 XML 分析程序被要求对其提供支持的,它使用 Unicode /ISO /IEC 10646 标准,该标准将字符空间扩展到 16 位,使其能够表示从 0 到 65,535 之间的任何值,从而能够包含大多数现代语言(在表示中文字符和其它几种字符及雕刻符号时仍旧存在重大问题,目前这些问题正处于解决之中)。XML 分析程序还能够(尽管并没有对它们有此要求)支持其它几种内码,包括从 ISO8859-1 到 ISO8859-9,它们代表了大多数欧洲语言;还包括 EUC-JP、Shift-JIS 和 ISO-2022-JP,它们代表日本语言。内码模式的名称必须包含在单引号或双引号中,并使用拉丁字符集对其描述。例如,要声明某个文档使用 XML 1.0 版本和 UTF-8 内码,以下 XML 声明是正确的:

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

Unicode 和其它内码

虽然大多数讲英语的开发人员已经习惯使用该语言的标准字符集,但其它地方的开发人员却习惯于在计算机中(尽管计算机实际上不是面向他们设计的)使用他们自己的语言。经过几年的发展之后,Unicode 标准终于获得了一些应用,打破了自 7 位 ASCII 字符集出现以来 8 位字符集统治计算机领域的局面。Unicode 使用 16 位内码,从而能够表示多达 65,536 个字符。

尽管 ASCII 足以应付大多数英语文档,但随着计算机逐步扩展到使用其它语言的领域,ASCII 只有 128 个字符成员的局限性已越来越明显(即使扩充到 256 个成员字符也无济于事)。甚至以拉丁语为基础的欧洲字母也有足够的重音字符和其它特殊字符很快将 256 个成员位置填满。再加上希腊字符、土耳其字符、希伯来字符、阿拉伯字符和希里耳字符,这些已远远超过可用的位置。亚洲语言的字符和雕刻文字具有成千上万个表意符号。创建能够用于所有这些语言的标准是一项复杂的任务,涉及许多政治的和技术的因素。

XML 要求分析程序支持 UTF-8、UCS-2 和 UTF-16。这些在 XML 标准中被要求支持或推荐支持的内码模式如下所示:

表 5-1 常用的 XML 内码模式

内码模式	内码位数	注释
UCS-2	16	规范的 Unicode 字符集
UCS-4	32	规范的 Unicode 字符集,使用 32 位内码
UTF-8	8	Unicode 转化而来,8 位内码
UTF-7	7	Unicode 转化而来,7 位内码(用于邮件和新闻)
UTF-16	16,32	避免 32 位字符的 Unicode 格式
ISO-8859-1	8	1 号拉丁字母表(西欧、拉丁美洲)
ISO-8859-2	8	2 号拉丁字母表(中欧 / 东欧)
ISO-8859-3	8	3 号拉丁字母表(东南欧 / 混杂)
ISO-8859-4	8	4 号拉丁字母表(斯堪底纳维亚 / 波罗的海)
ISO-8859-5	8	拉丁语 / 西里耳语
ISO-8859-6	8	拉丁语 / 阿拉伯语
ISO-8859-7	8	拉丁语 / 希腊语

ISO-8859-8	8	拉丁语 / 希伯来语
ISO-8859-9	8	拉丁语 / 土耳其语
ISO-8859-10	8	拉丁语 / 拉普语 / 日耳曼语 / 爱斯基摩语
ISO 10646	32	32 位扩展集, 将 Unicode 作为它的一个字集
EUC JP	8	日语(使用多字节内码)
Shift-JIS	8	日语(使用多字节内码)
ISO-2022-JP	7	日语(使用多字节内码, 用于邮件和新闻)

习惯于使用 ASCII 字符集的开发人员不会有太多的困难, 因为以上所列标准中的大多数都是以 ASCII 字符集作为其基础的。由于 XML 的默认字符集是 UTF-8, 所以开发人员可以毫无困难地显示大多数英文网页。UCS-2 标准是以一段能够识别它的代码序列开头的, 分析程序应该能够自动检测它。编写得好的分析程序能够在一定程度上避免用户在访问用默认内码之外的其它码编写的网页时屏幕上显示乱七八糟的随机字符。

不幸的是, 显示所有这些字符集仍需要操作系统的支持。Windows NT 和 Solaris 2.6 及 7.0 都提供 Unicode 支持, 但其它平台上的开发人员还需要另外的语言工具完成格式间的转化。收集一套表示多种语言的字体仍旧是一个问题, 但这方面的支持正在迅速增多。编程语言也存在类似的问题, 大多数用 8 位来表示字符数据。Java 已经使用 Unicode 作为字符信息的默认格式, 向开发人员提供了处理 Unicode 文的现成语言。对 Unicode 的支持正在增多, XML 将其用作标准必然会拓宽它的使用范围。

交叉参考



要想了解使 XML 在多语言环境中更加有用的另一个工具, 请参考下文“属性”一节中的 `xml:lang`。

文档类型声明

在序言已经宣布这是一个 XML 文档之后, 文档类型声明宣布它将是一个什么样的 XML 文档。文档类型声明将 DTD“粘附”到实际的文档或者提供有关文档结构的声明。尽管内部 DTD(在它们用于的文档中被声明)可以用于并适合某些特定的情形, 但外部 DTD 经常得到优先使用。使 DTD 与文档分离可以让它们更加具有在多个文档上的重复使用性, 向管理人员保证开发人员和创作人员不会肆意使用 DTD

迎合他们自己的需要而建出不兼容的文档类型。虽然对外部 DTD 的使用有所减少(这一点将在本章后面进行讨论),但大多数大型文档系统(甚至 Web 站点)将引用一个用于该系统中多个文档的 DTD 库。

注意

本书和其它 XML 文档中出现的“DTD”都代表“文档类型定义”不是“文档类型声明”。

文档类型声明总是以<! DOCTYPE 开头,后接 DTD 的名称,再往后是该 DTD 的声明或者是一个指向该 DTD 的链接,最后以>结束这个声明。DTD 的名称不需要与指定的 DTD 文件的文件名一致,但它应该传递某种对该 DTD 用于什么的易于理解的描述,最重要的是,它应该与该文档使用的根元素名称相匹配。然后,文档类型声明可以在该声明内部提供一个 DTD,将它放于方括号[]中(我们将在下面的几个例子中演示),也可以提供一个链接指向一个包含 DTD 的文件。对该 DTD 文件的引用是一个外部实体,关于外部实体将在后面的实体部分中作进一步的讨论。现在,我们只是讨论如何在当前情形中应用它们。

有些 DTD 是公共标准,以标准化格式供许多用户使用。其它 DTD 则是在当地开发的,仅对某个 Web 站点、企业或一个小的行业有用。对于前者,关键字 PUBLIC 更加合适;而对于后者,只能使用 SYSTEM 关键字。PUBLIC 关键字首先提供一个公共识别符(放在引号中)供分析程序用来定位该标准(如果它连接到一个标准库中)。然后提供一个 URL(也放在引号中),该 URL 将分析程序指领到该 DTD 的一份拷贝(除非知道你的用户将通过一个能够理解公共识别符并根据此识别符定位文档的应用程序来访问这一文档,否则一律提供 URL)。SYSTEM 关键字之后只跟有一个 URL。大型文档管理系统很可能有供分析程序使用的 DTD 库,但其它项目的开发人员一般没有这种资源。以下两个文档类型声明将文档连接到同一个 DTD,但第一个声明还提供了该 DTD 的一个公共识别符。

```
<! DOCTYPE manual PUBLIC "-//loopbackinc //DTD manual //EN" "http://www.simonstl.com/dtds/manual.dtd">  
<! DOCTYPE manual SYSTEM "http://www.simonstl.com/dtds/manual.dtd">
```

DTD 也可以使用嵌套——一个 DTD 文件可以通过参数实体调用另一个 DTD 文件。DTD 是叠加的,但内部 DTD 总是比外部 DTD 的优先级更高。

公共识别符结构使用的格式与 SGML 公共识别符相同。如果该实体或描述的 DTD 是 ISO 标准,识别符以“ISO”开头。否则,如果该标准是被某个标准机构正式批准的,识别符的第一个字符是加号“+”;如果没有得到正式认可,识别符的第一个字

符是减号“-”。然后是两条前向下划线“//”，再后面是该 DTD 的持有者。在紧接着两条前向下划线之后出现文档的类型(如 DTD 或 TEXT)，其后跟有空格和该文档的名称。然后又是两条前向下划线，随后是语言识别符(它使用 ISO639 中指定的代码，如英语用“EN”表示)。

批注

批注是 XML 的另一个关键部分。它们出现在文档中，也出现在 DTD 中。XML 中的批注与在 HTML 中相似，以 `<!--` 开始，以 `-->` 结束，处理器忽略它们的内容，人们可以在里边放入想放的任何东西。XML 批注不能在内容中包含两个连续的破折线(——)，因为这样会使分析程序误认为它是 SGML 批注的结束。XML 批注可以出现在文档中和 DTD 中，不能出现在标记或声明的内部，也不能在 CDATA 中工作(在 CDATA 中，批注符号被当作普通的字符来对待，将作为文档的一部分被显示出来)。分析程序总是忽略批注的内容，但它会将批注传递给应用程序(如浏览器)。以下是一个批注示例：

```
<!-- This is a comment. Please ignore me if you are parsing. -->
```

XML 中的批注可能用于 DTD 中，也可能用于文档中，但它们在 DTD 中至关重要。将来的编辑人员需要借助于 XML 批注作为理解你所创建的结构的路标。批注能够解释看上去很神秘的实体引用，对于标示声明也很有用，特别是当元素名称为缩写形式时。对于记忆力很强的开发人员来说批注可能是浪费空间，但没有批注的 DTD 对于下一位从事它的开发人员来说可能更是浪费空间。

处理指令

从技术上讲，以 `<?` 开始和以 `>` 结束的处理指令并不直接影响文档结构，只是应用程序注意到它们的存在(XML 处理指令必须以 `>` 结束，而不是 SGML 标准中的 `>`)。处理指令中的第一个(即“目标”)必须由字母、数字、句点、破折线、下划线或冒号组成，并由字母或下划线开头(由于名称空间的缘故，最好不用冒号——见下文中的描述)。处理指令中的其余部分可以由任意字符组成，包括等于号和引号。处理指令外部程序(如格式化程序)提供信息，这些外部程序往往更加关心如何使文档看上去符合设计人员的意愿，而不太关心文档的语法结构。大多数面向外部格式化程序的处理指令有它们自己的语法，它们不必遵循一般的 SGML 语法。尽管不太常见，但如果处理程序是一个叫做 Jimmy 的小孩子的话，以下代码：

```
<? Jimmy - use the burnt umber crayon crayon for this? >
```

可以是被接受的处理指令。更典型的处理指令是如下形式：

```
<? FormatWhiz azure-embossed-type? >
```

FormatWhiz 可能是(未必非得是)处理程序的名称,而该指令的其余部分则指定不常见的格式,如商业明信片或婚礼邀请贴。处理指令在将样式连接到文档时扮演重要的角色。尽管它们不能通过使用 DTD 得以标准化,但处理指令(通常记作 PI)的标准对于创建在多个文档类型上工作的功能十分关键。

尽管有批评者认为,处理指令创建的不必要那么复杂的 SGML 不能很好地在不同的分析程序之间转移,但 XML 工作小组看起来已认定它是告诉分析程序如何处理文档的最合适的语法。

当心

一定不要使用以<? XML、<? xml 或将大小写字母混合在一起的形式开头的处理指令,因为这些形式已被 XML 规范保留下来以备将来使用。

逻辑结构

元素和属性声明是 XML 的核心。构造良好的文档在特定情形中是有用的,但它们使用的元素结构仅存在于该文档及其设计人员的头脑之中。DTD 使开发人员能够将他们的思想具体化,为成组的文档(而不仅仅是单个文档)创建规范。一套编写得不错的元素和属性能使程序易于提取出有用的信息,并将它们漂亮地展现出来。虽然 XML1.0 推荐标准中的其它部分可以有助于这一任务,但 XML 的主要工作仍旧是使用元素和属性创建文档结构。

元素

在进一步讨论元素之前,需要来看看两个相关的概念:父元素和子元素。HTML 开发人员习惯于在使用元素时不过多地关心上下文,当然列表元素和表格元素例外。理解上下文是建造有效工作的 DTD 的先决条件。在 XML 中,上下文提供者是父元素,子元素可以向嵌入在它内部的元素提供上下文。例如,在下列结构中:

```
<SECTION>
  <PARAGRAPH>
    <SENTENCE>
      </SENTENCE>
    </PARAGRAPH>
  </SECTION>
```

SECTIONF 元素是 PARAGRAPH 元素的父元素, PARAGRAPH 又是 SENTENCE 元素的父元素。同样地, SENTENCE 元素是 PARAGRAPH 元素的子元素, 而 PARAGRAPH 本身又是 SECTION 元素的子元素。XML 还包含了一个文档实体, 它提供一个根, 从该根上可以生长出标记树结构。如果 SECTION 是文档如的第一个元素, 那么它就是文档实体的子元素(没有其它特别方式可以定义一个文档实体, 文档实体只是向分析程序提供一个开始的地方)。

正如我们在本章前面所看到的, 元素的创建非常简单。元素声明的语法如下:

```
<! ELEMENT 名称 内容>
```

元素的名称必须遵循与实体名称相同的规则: 必须只能由字母、数字、句点、破折线、下划线或冒号组成。冒号被保留用于名称空间(名称空间是在 XML 1.0 推荐标准之后出现的, 是一个与 XML 有关的新事物。下文将对它加以描述)。元素名称可以使用参数实体来定义, 内容也一样(记住, 如果该声明是在内部 DTD 子集之中, 参数实体必须包含整个声明, 而不仅仅是其组成部分)。元素的内容有四种类型, 分别是混合内容声明、元素列表、关键字 EMPTY 和关键字 ANY。其中 ANY 是最简单的声明, 它宣布该元素可以包含所有类型的数据和标记。其形式如下:

```
<! ELEMENT BOXOSTUFF ANY>
```

使用这一声明, 所有的 BOXOSTUFF 元素都允许将任何类型的数据或元素包含在它们的内容之中。使用上面声明的 BOXOSTUFF 元素的文档可以是下面的样子:

```
<BOXOSTUFF><DARKSPACE>emptiness</DARKSPACE>more junk</BOXOSTUFF>
```

其中的 DARKSPACE 元素需要在其它地方声明, 否则 BOXOSTUFF 将对其内容不施加任何规则。

尽管使用 ANY 是可以接受的 XML, 但我还是强烈建议开发人员对文档结构规定地更具体一些。因为 HTML 和一些其它形式的标记主要用于格式化, 所以有效使用 ANY 的标记是必要的, 禁止在某个段落中使用粗体文本无疑会导致混乱。XML 改变了所有这一切。通过向开发人员提供创建文档结构的机会, XML 承诺要帮助创建更加智能的文档。这种智能的一大部分是当给定一个开发完全的 DTD 后, XML 能够提供某种程度的查错, 判断哪些元素能够放在什么地方。尽量将 ANY 的使用限制在 DTD 开发的早期阶段, 并尽早地将其替换为更加完美的内容规范。

EMPTY 是与 ANY 相对的一个关键字, 它不是允许任何内容, 而是什么内容也不允许。用 EMPTY 内容定义的元素可以具有属性, 但不允许在开始标记和结束标记之间存放信息。空元素的元素声明很简洁:

```
<! ELEMENT EMPTYSPACE EMPTY>
```

在文档中使用这一空标记占用的空间更少, 如下:

<EMPTYSPACE />

大多数时候 EMPTY 元素只被写作以 /> 结束的空元素标记(如 <BT />)。当然也允许使用一般的开始标记和结束标记,但在这些标记之间不能出现元素或数据。

大多数元素声明都包含元素列表,设置所要求的元素、这些元素出现的顺序以及它们可以出现的次数(记住,列表中的所有元素必须在它们自己的声明中单独定义)。参数实体也可以出现在列表中,使开发人员创建多个相似文档时更加容易。表 5-2 列出了一些使用元素和属性(至于属性,将在后面讨论)的规则符号。

表 5-2 用于指定元素结构的一些符号

符号	符号类型	描述	示例	示例注解
	垂直条	被指名的任何元素都可以出现	thisone thatone	thisone 或者 thatone 必须出现
,	逗号	要求按指定的顺序出现	thisone, thatone	thisone 必须出现,其后跟以 thatone
?	问号	可以选择,但是只能出现一个	thisone?	thisone 可以出现
	无符号	必须出现一个,且只能出现一个	thisone	thisone 必须出现
*	星花号	允许按顺序出现任意多个,包括零个	thisone *	thisone 可以出现;多次出现(包括零次) thisone 是可接受的
+	加号	要求最少出现一个;可以按顺序出现多个	thisone +	thisone 必须出现;可以出现多个 thisone 元素
()	括号	给元素分组	(thisone thatone), whichone	thisone 或者 thatone 可以出现,其后跟以 whichone

注意

SGML 开发人员可能想知道为什么上面表格中没有“与”符号(&)。其实,XML(是起码在 1.0 版本中)不支持这种内容模型组合。在 XML 中可以通过混合声明和创造性地使用“或”语句来提供类似的功能。

这些选择项可以创建出巨大数目的不同的可能性,吸引人们以各种可能的方式

使用子元素来创建复杂的结构。下面的例子提供了一些使用可以接受的声明的简单示例及其解释。

最简单的元素内容是一个元素只包含另外一个元素的情形。开发人员可能需要用这种方式为位于外部 DTD 中无法被修改的元素创建包围元素。WRAPPER(包围元素)的声明如下:

```
<! ELEMENT WRAPPER(UNTOUCHABLE)>
```

在这个例子中,对 WRAPPER 元素唯一可接受的使用方式如下:

```
<WRAPPER><UNTOUCHABLE>untouchable content< /UNTOUCHABLE>< /WRAPPER>
```

更可能的声明依次包含两个元素。例如,一个简报可能包含一个题目和一些内容。其声明为:

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

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

```
<! ELEMENT BRIEFING (TITLE, CONTENT)>
```

这可以带来较复杂的标记,如下:

```
<BRIEFING>
```

```
<TITLE>Another dull briefing< /TITLE>
```

```
<CONTENT>Today, too much happened for me to adequately discuss it. < /CONTENT>
```

```
< /BRIEFING>
```

如果你的上级看腻了这种标题,可以修改 DTD 使 TITLE 可选:

```
<! ELEMENT BRIEFING(TITLE?,CONTENT)>
```

在个别情形中,也可以给 CONTENT 后加一个问号使它变为可选。如果简报变长了,开发人员可以通过添加一个加号使网页作者能够在一个简报元素下面包含多个 CONTENT 元素。如下所示:

```
<! ELEMENT BRIEFING(TITLE?,CONTENT+)>
```

将加号替换为星号会使简报更短,也可以使它更长,从而使网页创作者能够创建包含从零个 CONTENT 元素到几千个 CONTENT 元素的简报。

虽然多数简单文档和一些较大的文档以合理的顺序包含单一元素,但更加复杂的文档需要更多的组合来产生更大的灵活性。段落和列表经常能够互相替换。例如,菜谱可以包含一个配料列表或一个去往杂货店的途中故事。以下声明使这两种方法都有效:


```
<! ELEMENT STORY ( #PCDATA ) >
<! ELEMENT ITEM ( #PCDATA ) >
<! ELEMENT INGREDIENTLIST ( ITEM+ ) >
<! ELEMENT INGREDIENTS ( STORY | INGREDIENTLIST ) >
```

然后一个传统菜谱的配料部分的 XML 代码就可能如下所示:

```
<INGREDIENTS><INGREDIENTLIST>
<ITEM>Butter, enough to coat frying pan</ITEM>
<ITEM>Hot dogs, as many as needed</ITEM>
</INGREDIENTLIST></INGREDIENTS>
```

想要讲述他们的配料故事的作者则可以使用下面的格式,而 XML 分析程序也会接受它:

```
<INGREDIENTS><STORY>
No one wants to talk about where hot dogs come from, but I know their origin. They come from
trucks that unload them regularly into the backs of grocery stores. Clerks open the boxes and put
the hot dogs on the shelf. It's really not that complicated, and at least it's plastic wrapped. Butter
works the same way, but it comes in boxes and wax paper.
</STORY></INGREDIENTS>
```

使用“或”结构还可以创建出供有限数目的元素在其中进行自由组合的空间。例如,一个关于诗歌的讨论可能由评论穿插其间的引文组成。该序列无法事先预料,但它应该符合章节结构。以下声明创建了一个 CHAPTER 元素,它有一个标题,并允许出现多个引文和评论。声明如下所示:

```
<! ELEMENT TITLE ( #PCDATA ) >
<! ELEMENT QUOTE ( #PCDATA ) >
<! ELEMENT COMMENT ( #PCDATA ) >
<! ELEMENT CHAPTER ( TITLE, ( QUOTE | COMMENT ) * ) >
```

以下 XML 文档将得到正确的分析,(QUOTE|COMMENT)* 表示以任何顺序使用多个 QUOTE 元素和 COMMENT 元素

< /CHAPTER>

QUOTE 元素和 COMMENT 元素将以任意顺序出现。两个引文后跟一个评论是可以接受的,十个评论没有一个引文也是可以接受的。

XML 使开发人员可以通过括号、|、* 和 + 等运算符创建出精心策划的一套规则。可以定义整个文档以使所有子元素出现在根父元素中,这只需采取一种非常极端的声明就能做到,但由此产生的标记通常是不太实用的。如果标记声明变得过于复杂,通常需要将它分解开来创建一些子元素。例如:

```
<! ELEMENT CRAZY((TITLE | ART) + , (HEADLINE | PARAGRAPH | SUBHEAD | PICTURE |  
TABLE | POP-UP) * , CONCLUSION)>
```

这个 CRAZY 元素是完全合法的。它最少以一个 TITLE 元素或 ART 元素作为开头,下一部分(可能是文章的正文)可以以任意顺序和任意序列包含 HEADLINE、PARAGRAPH、SUBHEAD、PICTURE、TABLE 和 POP-UP 元素。在这一大堆乱七八糟的东西之后是一个必须出现的 CONCLUSION 元素作为文章的结尾。这一声明可以更好地分解为具有较多结构的多个元素:

```
<! ELEMENT CRAZY (HEADER, ARTICLE+ ,CONCLUSION)>  
<! ELEMENT HEADER (TITLE?, ART?)>  
<! ELEMENT ARTICLE (HEADLINE, CONTENT)>  
<! ELEMENT CONTENT (PARAGRAPH | SUBHEAD | PICTURE | TABLE | POP-UP) * >  
<! ELEMENT CONCLUSION ( #PCDATA)>
```

其中的 CONTENT 元素可以保留混合的形式,因为每篇文章都是不同的。但文档的其它部分获得了更多的结构。

最后一个选项是混合内容声明。从技术角度讲,本章大多数示例中使用的(#PCDATA)内容本身就是混合内容声明。混合内容声明能够使多个元素作为子元素出现,而不必要求它们出现或对它们出现的序列作任何具体的要求。最简单的混合内容声明是经常使用的:声明内容为 PCDATA 以使文本和实体(但没有其它的任何元素)出现。如下所示:

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

现在 ORDINARY 元素能够以任意组合形式包含文本或实体标记。XML 为需要包含文本的元素仅仅提供了 PCDATA。因此,这一基本的声明几乎可以用于任何“叶子”元素(叶子元素有父元素但没有任何子元素;形象地说,它们是树的最远端分枝,是动作实际发生的地方)。但在有些情形中,开发人员可能想允许其它元素出现在叶子元素中。并非所有叶子数据都对每个元素适合。例如配料不应该包含一个内

容表格,但它可以包含一条注释。创建包含一个配料描述和 /或一条注释的元素,其声明如下:

```
<! ELEMENT INGREDIENT( #PCDATA /NOTE) * >
```

这一声明将允许 INGREDIENT 元素包含文本信息供它们用来展示配料,并包含一个 NOTE 元素来解释奇特的或不容易找到的配料。

如果 DTD 包含一大组能够在多个父元素中使用的子元素,那么该 DTD 可以通过使用列出这些元素的参数实体而得以简化。分析程序将分析该参数实体并把其标记添加到元素内容声明中。

```
<! ENTITY % parts "prologue | detail | moral | punchline | joke">
<! ELEMENT STORY ( #PCDATA | %parts; ) * >
<! ELEMENT TALE ( #PCDATA | %parts; ) * >
<! ELEMENT FABLE ( #PCDATA | %parts; ) * >
```

在这个例子中,STORY、TALE 和 FABLE 元素能够包含文本和 PROLOGUE、DETAIL、MORAL、PUNCHLINE 或 JOKE 元素中的任何一个。这些子元素可以按任何顺序任意多次出现。所有其它的元素都禁止出现在 STORY、TALE 或 FABLE 元素中。

参数实体也可以包含括号和 #PCDATA 声明。每种方法都在不同的情形下具有其自己的优势。

属性

属性提供了 HTML 的许多能力,但它们在 XML 中将用得稍微少一些。属性用得更多的是存放有关计算机的信息,而不是人类的信息。即便在 HTML 中,属性也是用来为浏览器包含关键的格式化信息,而不是存放有关元素内容的信息。但属性仍旧是 XML 的重要组成部分,它提供了超出元素本身所能提供的灵活性,并巩固了作为基础的结构。属性使用的一些语法与元素声明相似,但倾向于向它们允许的内容提供更精确的定义。

属性是使用以下语法来定义的:

```
<! ATTLIST ElementName
    AttributeName Type Default
    (AttributeName Type Default...)>
```

属性声明中的第一个值是该属性将应用于的元素名称。虽然紧跟元素声明之后包含属性声明会使 DTD 更具有可读性,但并不一定非得这样做不可。实际上,同一元素可多个属性声明,所有声明将被组合成一个大的集合。如果同一属性在这个集合中被声明了多次,那么只有第一次声明被采用。这使得无须对现有的 DTD 作大幅改动就能扩展它们。

在指明元素名称之后,接下来是属性定义或一个属性定义列表。属性定义由属性名称、类型及其默认值(或具体的指定值)组成。属性名称必须遵循与实体和元素名称相同的命名规则:只能包含字母、数字、句点、破折线、下划线和冒号。再次强调,冒号被保留用于名称空间,这将在后面的小节中详述。属性类型与以前讨论过的结构完全不一样,它们定义当在元素实例中使用属性时允许的数据种类(元素实例是在文档中对该元素的一次使用)。表 5-3 列出了属性类型所有可能的取值。

表 5-3 属性类型

类型	释义
CDATA	该属性可以仅仅包含字符数据。
ID	该属性的取值必须是唯一的,它用来识别元素。如果在一个文档内部的两个 ID 属性具有相同的取值,分析程序会返回错误。(注意,类型为 ID 的属性不可以使用默认值或固定值。)
IDREF	该属性的值必须引用在文档中其它地方声明的 ID 值。如果该属性的取值不与文档内部的某个 ID 值匹配,分析程序就会返回错误。
ENTITY, ENTITIES	ENTITY 属性的值必须对应一个在文档内声明的未分析过的外部实体。ENTITIES 属性也与此相似,但允许使用以空白分隔的多个实体名称。
NMTOKEN, NMTOKENS	NMTOKEN 属性必须是一个与 CDATA 非常相似的名称记号,但该值中使用的字符必须是字母、数字、句点、破折线、下划线或冒号。NMTOKENS 也与此类似,但允许使用以空白分开的多个值。
NOTATION	该属性的值必须引用在文档中其它地方声明的某个注解的名称。
Enumerated, (thisone thatone)	该属性的值必须匹配所列出的某一个值。所列值必须出现在括号中并以“或”符号()隔开。
NOTATION (enumerated)	该属性的值必须匹配 NOTATION 名称列表中的某个名称。例如,NOTATION(picture /slide)需要拥有“picture”或“slide”的值,而且必须存在 picture 和 slide 的 NOTATION 声明。

大多数时候开发人员需要使用 CDATA、ID 和枚举(enumerated)类型,当然使用其它类型的可能性也不是没有的。属性声明中最后一个必需部分是默认。默认可以

采用表 5-4 中所列出的四个值之一。

表 5-4 属性默认

值	释义
#REQUIRED	告诉分析程序在某个元素的所有实例中都必须有该属性的值。若没有包含该属性将导致分析错误。
#IMPLIED	如果没有指定任何值的话允许分析程序忽略该属性。XML 工作草案规定:XML 处理器必须通知应用程序没有指定任何值;对该应用程序的行为不加任何限制。
#FIXED value	宣布为该属性指定某个值的元素实例必须指定所列出的值。如果某个元素实例不包含这一属性,其值将被认为是所指定的值。
defaultvalue	为属性提供一个默认值。如果该属性未在某个元素实例中明确声明,它将被默认为使用值 defaultvalue。

到此为止我们已经解释了属性声明的各个部分,接下来看看它们都做些什么:随着脚本工具和其它处理程序逐渐广泛用于处理文档中的单个元素,ID 类型的属性正变得越来越常见。但要使这些系统有效工作,通常要求所有元素(至少将被操作的那些元素都有一个 ID 值,该值通常在 ID 属性中列出。为某个元素声明一个必需的 ID 值并不费劲,如下所示:

```
<! ELEMENT DATABRICK( #PCDATA)>
<! ATTLIST DATABRICK
    id ID #REQUIRED>
```

来自该 DTD 的文档中创建的所有 DATABRICK 元素,都必须为它们的 ID 属性提供唯一的值。其它属性也可以根据它们的类型让处理程序以不同的其它方式对待 DATABRICK 元素。定义一个类型列表会使处理程序能够更顺利地运行。

```
<! ELEMENT DATABRICK ( #PCDATA)>
<! ATTLIST DATABRICK
    id ID #REQUIRED
    status (proceeding | accepted | rejected | deferred) "proceeding">
```

DATABRICK 元素现在有一个状态属性来通知应用程序它们的状态。如果创建的某个元素没有为该属性指定取值,那么该 DATABRICK 元素就会被认为沿着一条最终结果为“被接受”、“拒绝”或“延期”的“proceeding”路线前进。文档管理系统或数据库可以使用这一属性限制它们对 DATABRICK 元素的行为。

当属性的有无对正确处理元素无关紧要时,可以使用隐含的默认值。将文档从一个 DTD 转换到另一个 DTD 的应用程序可以使用某种形式的批注来记录它们对该文档实施的行为。只有经过这一过程的文档才需要这种批注;直接在目标 DTD 中创建的文档则没有这一批注。要创建这种批注属性,可以使用以下声明:

```
<! ELEMENT MASTERPIECE ( #PCDATA )>
<! ATTLIST MASTERPIECE
    TranslationNote    CDATA    #IMPLIED>
```

这已足以满足多种情形。但最好再提供一个默认值,而不是让分析程序报告没有任何值。如下:

```
<! ELEMENT MASTERPIECE ( #PCDATA )>
<! ATTLIST MASTERPIECE
    TranslationNote    CDATA    "None">
```

“None”的语气比说明该属性没有包含任何值更加肯定一些,更能醒目地证实没做任何转换。这样可以使程序员很容易弄清楚到底是分析程序有问题还是实际上没有任何值。

固定属性类型比较不常见。当把文档输入到的处理程序需要使用格式化属性以确保用该 DTD 创建的所有文档的处理结果看上去相似时,固定属性类型可能会派得上用场。通过指定一个固定的属性值,DTD 可以确保在一组使用相似 DTD 的文档中保留它的特征。

```
<! ELEMENT ARTICLE ( #PCDATA )>
<! ATTLIST ARTICLE
    FormatModel    CDATA    #FIXED "Contemporary">
```

另一个 DTD 文件使用的声明与此类似,但使用的固定值不同:

```
<! ELEMENT ARTICLE ( #PCDATA )>
<! ATTLIST ARTICLE
    FormatModel    CDATA    #FIXED "Country">
```

处理程序可能注意到了上面的 FormatModel 属性并选择一套适合该模型的样式。也可以使用该信息给库中的文档分类,以便于浏览器从中选择适合它们的当前设计模式的那些文档。

引用外部数据源的属性值也可以用于处理。虽然分析程序本身不会对这些信息做任何事情(除了将它传递下去),但处理程序能够将这些信息与标记材料结合在一起创建出(比如说,)带有图片的文档:

```
<! NOTATION ourFormat1 SYSTEM "http://www.simonstl.com/ourViewer.exe">
<! NOTATION ourFormat2 SYSTEM "http://www.simonstl.com/pictures/ourPlayer.exe">
<! ELEMENT DOCUMENT (#PCDATA | PICTURE) * >
<! ELEMENT PICTURE empty>
<! ATTLIST PICTURE
    TYPE      NOTATION (ourFormat1 | ourFormat2) "ourFormat1"
    IMAGE CDATA #IMPLIED>
```

在这个例子中,文档的编辑人员已经宣布所有的图片都必须使用两种格式中的一种。使用这个 DTD 的 XML 文档可能是下面的样子:

```
<DOCUMENT>I hate my boss. He makes me use this picture all the time:
<PICTURE TYPE="ourFormat2" IMAGE="FROGS.fm2" />
Sometimes he lets me use this image:
<PICTURE TYPE="ourFormat1" IMAGE="BIRDS.fm1" />
But I hate it more, so I usually stick with the frogs.
</DOCUMENT>
```

XML 推荐标准中还提供了两个预定义的属性来辅助 XML 的处理,并帮助解决 XML 处理中遗留的一些模棱两可的问题。第一个是 `xml:space`,它帮助应用程序决定是否注意空白。第二个是 `xml:lang`,它使文档能够更容易地展示使用多种语言的内容,补充了 Unicode 显示使用不同字符集的文档的能力。它将为文字的表达(有些语言为同一字符使用不同的表示符号)带来巨大帮助,并有可能用于翻译。

xml:space

XML 推荐标准的第 2.10 节定义了一种新的属性允许元素向应用程序表明它们的空白是否有意义。它将广泛用于与 XSL 或 CSS 空白属性结合以正确显示文档,还可能对于处理未必用作显示的文档产生一定的影响。检验处理程序(XML 处理器)早就必须把所有的非标记元素传递给应用程序,并通知应用程序它们出现在哪个元素中。这一属性可以充当标志,告诉应用程序是否应该注意空白字符。

注意

- 是否给予空白字符任何处理是由应用程序来决定的。虽然浏览器和其它一些 XML 显示程序将很好地利用 `xml:space`,但肯定还有许多应用程序会发现这一属性与它们没有关系。

xml:space 属性的声明方式如下(注意,如果用于将被检验的文档中,该属性仍然需要声明):

```
<! ATTLIST element xml:space(default /preserve) 'default' >
```

xml:space 的形为是从父元素继承来的;如果包含一个 xml:space 值的元素还包含其它元素,那么被包含的这些元素也会按父元素指定的方式处理空白。它可以被子元素中新的 xml:space 属性重载。

因为在 DTD 中可以设置默认,所以创建默认时不忽略空白的文档非常容易,只要把根元素的 xml:space 的默认值设置为“preserve”即可。要想规定一律不忽略空白(记住,你的 XML 文档的各个部分可以通过 XML 链接返回),可把“preserve”作为默认值赋给所有的元素类型。

xml:lang

xml:lang 这一属性向 XML 创作者提供了一条标识特定元素中所包含的语言的途径。结合 XML 对 Unicode 提供的支持,它使得提供国际化的信息变得更加容易。开发人员可以创建具有内置翻译功能的文档,或使应用程序能够更容易地知道何时提供翻译。例如,假设某人试图展示使用拉丁语言的引文,并带有使用英文的描述,如下:

```
<SECTION>
<DESCRIPTION xml:lang="en">
Caesar begins by describing the geography of Gaul.
</DESCRIPTION>
<QUOTE xml:lang="la">
Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur.
</QUOTE>
<EXPLANATION xml:lang="en">
It isn't the most thrilling opening to a great work on war, but it does explain some key issues to Romans who probably have never been anywhere near Rome.
</EXPLANATION></SECTION>
```

在这个例子中,英语用“en”表示,拉丁语用“la”表示。备有拉丁语翻译器的应用程序可以在收到提示(也可以没有提示)时将上面的拉丁语部分转换成如下的形式(英语):

All of Gaul is divided into three parts, one of which is inhabited by the Belgae, another by the Aquitani, and the third by those who are called Celts in their own language, and Gauls in ours.

翻译程序有可能将原文翻译得比这还准确,但在这里你已经能看懂的它的大意了。`xml:lang` 属性的行为与 `xml:space` 非常相似,它用于子元素的内容,也可以用于实际使用它的元素中。如果 `xml:lang` 用于有效的文档中,它必须被声明(这种声明对于声明默认语言也很有用)。以下声明语法将创建一个没有默值的属性:

```
<! ATTLIST element xml:lang NMTOKEN #IMPLIED>
```

要创建一个使用英语作为默认语言的 `QUOTE` 元素,可以使用以下声明:

```
<! ATTLIST QUOTE xml:lang NMTOKEN 'en'>
```

这可以被重载为使用拉丁语、法语、德语和具有 ISO 639 中定义的代码或被 Internet 授权机构 IANA 注册过的其它任何语言。ISO 639 代码可以直接拿来使用,也可以后跟一个国家代码以更加精确地定义该语言。如“en-GB”(英国英语),与它相对的是“en-US”(美国英语)。IANA 代码必须加以前缀 i-或 I-;其它代码也可以使用,但必须给它们加上前缀 x-或 X-。与其它大多数 XML 不同,这些代码不区分大小写字母(IETF RFC 1766 中对此有更加详细的解释)。

`xml:lang` 属性提供比 Unicode 数据多的信息,并为应用程序节省了许多本来用判断为特定元素采用何种语言所花费的时间。但它并不是万能的,要有效使用这一元素需要一致的应用程序支持,还可能需要一些相当复杂的样式用法。浏览器开发商有可能抓住这一机会整理当前 Web 上的语言混乱现象。与 Unicode 结合使用,这一属性将使文档的国际化更加容易。

名称空间

名称空间是在 XML 1.0 推荐标准完成后出现的,已在各种论坛中引起了不少的轰动。名称空间的本意是避免不同组织中使用的相同的标记名称发生冲突,它已经创建了具有潜在复杂性的新层次。在实际操作中看上去非常简单的名称空间已在分析程序和应用程序的职责、DTD 的缺点、XML 与 SGML 兼容的重要性、标记和语义的区别等重大问题上产生了许多争议。在经过一年的 W3C 内外的争论之后,已经出台了一个推荐标准(见 <http://www.w3.org/TR/REC-xml-names>)。尽管名称空间才刚刚崭露头角(本书其余部分的示例中不会用到它),但着手建造(或偶尔阅读)XML DTD 的开发人员还是应该知道它们是干什么的。

名称空间通过使用前缀(前缀与元素名称之间被冒号隔开)保留了名称的唯一性(最简单的或默认的名称空间没有前缀,也不使用冒号)。这些前缀然后可以被解析的统一资源识别符(URI),以利用 Web 上早已存在的命名体系(特别是域名)。这种解析不用为 XML 名称空间前缀建立新的注册表,还可以使默认的名称空称解析为有意义的值,避免了向使用单个名称空间(或是单个名称空间的作用域)的文档中键

人多个前缀的必要性。在讨论如何声明名称空间之前,先检查一下使用名称空间的文档中的一些元素。ssl 名称空间的 URI 是 `http://www.simonstl.com/default`, 默认名称空间的 URI 是 `http://www.w3.org/TR/REC-html40` (实际上在 URI 位置上不需要做任何事情;真正有关的是该 URI 唯一地识别这个名称空间)。

```
<HTML>
<BODY>
<P>This is a paragraph in a document that uses the HTML namespace as a default, but which contains
<ssl:hedge>some</ssl:hedge> elements from <ssl:adjective>another</ssl:adjective> <ssl:noun>namespace</ssl:noun> as well. </P>
</BODY>
</HTML>
```

应该告诉应用程序(或让它能够想到——因为它现在还不知道)HTML、BODY 和 P 元素属于以 `http://www.w3.org/TR/REC-html40` 标识的名称空间,而 ssl:hedge、ssl:adjective 和 ssl:noun 元素则属于标识为 `http://www.w3.simonstl.com/default` 的名称空间。应用程序将从处理器那里接受到合格的名称;在处理器中前缀被删掉,元素(或属性)的名称被成对地(名称空间识别符和元素名称)交付给应用程序。如果碰到两个组织使用同一个前缀的情形,该前缀将被改变——只有前缀背后的 URI 与应用程序有关系。这时只有该名称空间的声明需要改动,仅此即可。

名称空间是使用以 xmlns 作为开头的属性来声明的(记住,最初不是 W3C 创建的属性和元素不能以 xml 开头,因此使用 xmlns 安全一些)。这些属性可以在 DTD 中声明,并可以使用默认值;也可以在文档中声明(请见这一节的最后一个段落中关于名称空间和检验的警告)。这些声明的行为有点儿怪,有点儿象 xml:lang 和 xml:space 表现出来的继承效应;但因为声明的名称可以改变,从而有点儿复杂。其机理相当简单:如果一个元素包含一个 xmlns 开头的属性,那么该属性为这个元素本身和任何子元素(除非在子元素中出现新的定义将它重载)定义了一个名称空间。在这种情形中,新的定义对该元素及其所有子元素都适用。如果属性名称仅仅是 xmlns,那么它定义的是默认空间,适用于所有名称中没有冒号的元素。如果属性的名称是 xmlns:prefix,那么该属性为所有以 prefix:开头的元素定义了一个名称空间。名称空间 URI 在这两种情形中都被定义为属性的值。例如:

```
<DOCUMENT xmlns="http://www.simonstl.com/" xmlns:HTML="http://www.w3.org/TR/REC-html40">
<HEADER>This is <HTML:I>my</HTML:I> header. </HEADER>
<CONTENT>
<P xmlns="http://www.w3.org/TR/REC-html40">This is a paragraph filled with <B>HTML
</B> elements, so I figured it was <I>better</I> to set the default namespace to <EM>
HTML </EM></P>
<PAR>This is a paragraph in the http://www.simonstl.com namespace, again. </PAR>
```

< /CONTENT>< /DOCUMENT>

这个文档中的元素位于表 5-5 所列的名称空间之中。

表 5-5 按顺序列出的名称空间结果

前缀	元素名称	元素位于名称空间	设置名称空间
Default	DOCUMENT	http: //www. simonstl. com HTML: http: //www. w3. org / TR /REC-html40	default: http: // www. simonstl. com
Default	HEADER	http: //www. simonstl. com	
HTML	I	http: //www. w3. org /TR / REC-html40	
Default	CONTENT	http: //www. simonstl. com	
Default	P	http: //www. w3. org /TR / REC-html40	default: http: //www. w3. org /TR /REC-html40
default	B	http: //www. w3. org /TR / REC-html40	
default	I	http: //www. w3. org /TR / REC-html40	
default	EM	http: //www. w3. org /TR / REC-html40	
default	PAR	http: //www. simonstl. com	没有设置,但归还到 父元素(CONTENT) 的默认名称空间

这种机制提供了许多灵活性,并能产生许多不同的名称空间。在构造良好的文档中添加一个名称空间象添加属性一样容易;在有效的文档中,需要创建一个属性,属性的名称应该与使用的前缀相适合。这种机制使得可以在文档内部很容易地混合和搭配名称空间,并能支持无限多个(直到你的分析程序或应用程序崩溃为限)名称空间而不发生冲突。

有关名称空间的问题主要来源于早已为 XML 1.0(和 SGML)开发好的分析软件不支持改变前缀或报告名称空间。在启用名称空间的世界里,前缀只是名称空间

的缩写,是可以改变的暂时的东西。在当前检验 DTD 的过程中,改变前缀意味着改变元素名称,从而意味着该文档无效。名称空间给检验带来的影响现在还不能尽述,但它们的使用,意味着 SGML 软件的坚实基础和为 XML 1.0 建造的软件将发生重大的改变。有些分析程序已经开始更新以应付这种形势,但要直接解决这些问题可能得对 XML 推荐标准本身作大的修改,甚至导致下一个版本的出现。但在目前来说,名称空间还很有前途。

数据结构和类型

既然我们已经有了文档结构,现在必须看一看这些结构中包含的数据。正如 HTML 开发人员在试图将老的文件转换成 HTML 时所发现的,用作标记的<、> 和 & 等字符能够产生重大问题。例如,我曾经从 Macintosh HyperCard 堆栈转换过来一个站点,这个站点工作得不错,但在 AT&T 机器上网页总是时常有一些遗失的字符。XML 应用了一些 SGML 方法来解决 HTML 的这一问题的,虽然增加了一些复杂性但可以将这一问题解决得相当彻底。

虽然 XML 已经大大地简化了 SGML 的数据类型,但我们仍旧需要考虑许多在 HTML 中从没有出现过的问题。XML 可以在文档中使用两种类型的数据。一种是 #PCDATA, 或叫被分析的字符数据,它是常见的标记字符数据。另一种是 CDATA,它是没有任何标记的字符数据。当文档不包含任何标记而只有许多<、> 和 & 符号时,可以使用 CDATA。默认时,XML 认为所有信息都是 PCDATA。CDATA 也可以出现在文档中,位于被标记为纯字符数据的部分。要将某一小节声明为 CDATA,将该节的开始端用<![CDATA[标记,结束端用]]标记(如果数据包含任何]]>序列,标记就会失败;但这种可能性很小)。例如:

```
<? xml version="1.0" encoding="UTF-8"? >
<DOCUMENT><![CDATA[@#X! <<<< >> & <<<<< >>>]]></DOC-
UMENT>
```

其中的 CDATA 部分将被解释为字符“@#X! <<<<>>&<<<<<>>>”而不会产生分析错误。当在这个文档中查询文本时,分析程序会返回包含在 CDATA 部分中的信息:

```
@#X! <<<< >> & <<<<< >>>
```

如果这段文本没有使用 CDATA 声明进行“换码”,那么分析程序就会在第一个<处停止,因为它看上去好象打开一个没有正确关闭的标记。虽然使用 CDATA 时禁止开发人员在文本部分中使用标记,但如果该部分基本上不需要标记的话,这种让步是值得作出的。如果它需要标记,可以将不合适的字符替换成它们的实体等价符

号,这将在本章后面介绍。

注意

使用 CDATA 来为文本换码实际上是一种更通用的 SGML 技术——标记区域的一个具体的例子。标记区域遵循 `<![keyword[data...]]>` 语法。虽然开发人员可以对 CDADA 使用这一语法,但 SGML 的 RCDATA、TEMP、IGNORE 和 INCLUDE 关键字在 XML 文档中都不能使用。但 IGNORE 和 INCLUDE 标记区域却可以在 DTD 内部使用,本章后面将作介绍。

许多开发人员已经抱怨的一个遗漏的特征是,XML 没有提供任何途径用来要求元素包含更具体类型的数据,如文本、数字或货币等。几种有关这一“更强类型”的提议正处于发展阶段。W3C 当前正在“模式”中检查这一问题(详细内容见第 13 章),可能使其提供远比 DTD 还多的功能。同时,数据类型的任何增强都将是处理它的应用程序的责任,而不是 XML 分析程序的责任。

实体

XML 提供两种实体——通用实体和参数实体。HTML 开发人员很熟悉使用预定义的通用实体为不常用的字符和用于标记的字符(如 `<`、`>` 和 `&` 等)编码。虽然是 DTD 中定义的,但通用实体主要用来向文档添加信息,用它们的值替换实体引用。其形式为 `&name`。参数实体是在外部 DTD 中定义的,只能用于 DTD。它们能够象通用实体那样节省开发人员的工作,还能向开发人员提供将其它 DTD 和其它信息包含在自己的 DTD 之中的巨大能力。参数实体能使开发人员重新使用较老的 DTD 和为其建立子集,避免重复性工作并使现有的 DTD 更容易扩展。

通用实体

HTML 中到处都用到通用实体来表示位于基本 ASCII 字符集之外或与标记相冲突的字符。XML 在这方面问题很少,主要原因有下面几个。首先,之前已经描述过的字符编码使开发人员能够更直接地包含其它语言和其它系统中的字符,从而缓解了对那些字符表示实体的需求。其次,用 CDATA 标记区域为字符“换码”这一选择使得能够更容易地包含与标记相冲突的字符(特别是大的文本块)。当然,CDATA 有些“笨重”,不便于经常使用。XML 也有一些内置的实体(虽然没有 HTML 那么多),表 5-6 列出了五个内置的实体:

表 5-6 XML 标准中内置的实体

实体	表示的字符
&	“与”号(&)
<	小于号(<)
>	大于号(>)
'	撇号('),表省略
"e;	引号(")

虽然这些实体确实很有用,能帮助开发人员使他们的内容与标记分开,但它们只提供了 XML 通过通用实体所能做的最小的事情。开发人员可以象定义元素那样来定义实体。通用实体非常简单,而且能使许多复杂烦扰的任务简单起来,特别是当它用于填充新闻稿样式的文本时。定义通用实体的语法也比较简单:

```
<! ENTITY Name EntityDefinition>
```

实体的名称必须由字母、数字、句点、破折线、下划线或冒号组成,而且第一个字符必须是字母或下划线(由于名称空间的原因,不鼓励使用冒号,如前面所述)。实体定义可以包含任何合法标记,而且必须放在引号中。在标记中使用实体也很简单,语法如下:

```
&Name;
```

一开始必须是“与”号(&),最后必须带分号。在实体内部或周围不允许出现空白。

对于在文档的生命周期中经常改变的重复性信息,用这种方法创建实体非常有用。例如,在为产品的第一个版本开发操作手册时,开发人员可能连该产品的名称还不知道。当产品最终出品时不要使用查找一替换的方法,因为这样有可能引入错误,而应该使用一个实体引用来保证正确指向该产品。例如,某个项目的代码名称可能是“Crystal”,在该文档的预发布版本中,开发人员可以创建如下实体引用:

```
<! ENTITY ProdName "Crystal">
```

以后不论该产品什么时候被提到,它们都会使用实体引用而不是该产品的实际名称。以下代码:

```
&ProdName; is a remarkable advance, guaranteeing users happier days.
```

将被解释为:

Crystal is a remarkable advance, guaranteeing users happier days.

当产品的名称最终确定下来后,将 Crystal 转换为 RF-2000-QJ-46,开发人员只需修改该引用实体:

```
<! ENTITY ProdName = "RF-2000-QJ-46">
```

以上文本将被解释为:

RF-2000-QJ-46 is a remarkable advance, guaranteeing users happier days.

这看上去有一点儿夸张做作,但有决心的开发人员如果需要的话,他们完全可以创建出寻找不同语法结构(如所有格)的实体。实体对于法律合同和其它这类主要包括新闻稿式文本的文档极其有用。在简单的合同中唯一变化的部分是合同的双方及涉及到的钱数,该合同不妨写为:

```
< CONTRACT > &boilerplate1; < PAYMENT > $ 100, 000, 00 ( US) < /PAYMENT >
&boilerplate2; < RECIPIENT > Lucky Author < /RECIPIENT > < /CONTRACT >
```

虽然多数合同都要求多于这种格式所允许的改动,但许多合同可以分解为标准的句子,从而能够有规律地使用实体。

尽管如此,通用实体的主要用途仍旧是表示常见字符集(一般是 ASCII)之外的字符。字符引用能使开发人员包含那些不容易输入或有歧义的字符,如:

```
<! ENTITY THORN "§ # 222;">
```

当它用于使用 Latin-1 或 Unicode 内码的系统时,将产生一个北日耳曼语的 Thorn,这个字符在美式键盘上一般没有。内置的“与”实体被声明为:

```
<! ENTITY amp "& # 38">
```

虽然开发人员可以在 XML 中直接使用这些代码而不用自己费力创建实体声明,但给这些字符指定名称会使产生的文档更加干净利落。ISO 8879(SGML)包含一套完整的在 SGML 中使用的实体标准,但多数早期的 XML 分析程序可能不对 SGML 标记使用的代码给予处理。一大套为 XML 编写的实体 DTD 可在站点 <http://www.schema.net/entities/> 处得到,它也包含了多种不同的字符和记号。你可以检查这些代码并将所需要的部分添加到自己的 DTD,或者使用参数实体(本章后面将描述)将整个实体 DTD 包含到自己的 DTD 中。

注意

实体也有自己的“小脾气”。在实体内部使用标记是允许的,但在实体内部使用实体就得做些额外工作了。分析程序将在标记被添加到文档文本时检查它的实

体,实体代码中的错误会产生奇怪的分析错误。一定要在使用实体之前先检查它们,并确保实体的内容必须适合它在文档中的目的地。

有一点必须专门提一下。XML 分析程序根据一套严格的规则来解释实体,这些规则往往导致对实体的分析不止一次。这使得在一些情形中很难在实体中包含实体。这种情形的发生是因为当计算机分析文档时有些实体(所有的参数实体和字符引用)被分析的缘故。当该实体放入文档中时,所有实体都要被分析。虽然这类情形非常少见(大多是当开发人员使用“与”号和小于号的等价字符引用时发生),但当实体莫名其妙地给文档带来混乱时,开发人员必须分析他们的实体并检查其结果。

并非所有的实体都引用 XML 数据。“未分析”的实体必须引用存放在 XML 文档本身之外的文件中的信息。在 SGML 世界中,未分析实体被用来以一种比 HTML 的更正式的方式,将图像和其它文件连接到 SGML 文档。引用二进制(非文本)数据的实体必须是外部的(即,它们必须使用 SYSTEM 或 PUBLIC 识别符),必须在实体定义之后,使用关键字 NDATA 以注解识别符指定该实体的数据类型。例如,一个引用 GIF 文件的外部二进制实体应该以 NDATA gif 作为结束。外部未分析实体不能使用 &name; 语法直接出现在 XML 文档中。而是该实体的名称应该出现在已被声明为 ENTITY 或 ENTITIES 类型的属性中。然后应用程序就可以判断怎样处理该外部未分析实体中的信息。

注意

外部未分析实体位于提供传统 SGML 文档的规范之中。包含二进制数据文件的“SGML 方式”要求的步骤比“HTML 方式”还要多,它依靠服务器(和文件扩展名)来识别内容的类型。许多 XML 可能永远不需要声明这些实体或支持这种处理,这取决于他们使用的应用程序所要求的样式。

参数实体

作为另一种实体类型的参数实体只用于 DTD 内部。参数实体携带用于标记声明中的信息,通常是被几个元素共享的一组共同属性或指向外部 DTD 的一个链接。参数实体的引用纯粹位于 DTD 内部,因此又叫做内部实体;而从外部文件获取信息的引用叫做外部实体。虽然参数实体极大地简化了 DTD 的创建,但应该小心使用它们。实体的本性要求查看它们以确定它们的内容,这对于计算机来说不是太难,但对于必须分类归并纷繁的 XML 的个人来说就变得相当复杂了。

参数实体在其引用和声明中都使用百分号(%),这一百分号将通用实体与参数实体区别开来。参数实体的声明语法如下:

```
<! ENTITY % Name EntityDefinition>
```

百分号和实体名称之间的空格是必需的。象通用实体一样,参数实体的名称也必须由字符、数字、句点、破折线、下划线或冒号组成,并且第一个字符必须是字母或下划线。实体定义可以包含任何有效的标记,并且必须位于引号中。实体定义的值必须在使用实体的上下文中有意义,否则分析程序将无法理解该 DTD 并返回错误。

内部实体的行为与通用实体非常相似,但它在 DTD 中操作,而不是在文档内容中操作。例如,包含多种被引用材料的文档的 DTD 可以为这些不同种类的材料(信、日记、小说、诗和引文)使用一组共同的属性。要使文档更容易搜索,所有这些元素都必须具有相应的属性来识别它们使用的语言,以及判断它们是否超出了版权有效日期。提供有这些属性的实体声明可以是下面的样子:

```
<! ENTITY % sourceinfo  
"LANGUAGE CDATA #REQUIRED  
COPYRIGHTDATE CDATA #REQUIRED">
```

在声明中使用该实体只需要用 %name; 语法调用它即可:

```
<! ELEMENT LETTER ( #PCDATA )>  
<! ATTLIST LETTER % sourceinfo ;>  
<! ELEMENT QUOTATION ( #PCDATA )>  
<! ATTLIST QUOTATION % sourceinfo ;>
```

(当使用实体时在百分号和实体名称之间没有空格,空格仅在声明实体名称时使用)。上面的语言和版权日期信息将“流入”QUOTATION 和 LETTER 的属性声明。当分析程序碰到 %sourceinfo; 时,它会扩充该实体使其包含在该 DTD 中其它地方宣布的全称值。QUOTATION、LETTER 以及使用该源信息实体内容的其它任何元素都将要求指示语言和版权日期的属性。这将随着重复性元素的增多和属性列表的变长而更加有用。

当心

XML 1.0 推荐版本对在内部子集中使用参数实体(即直接在文档中声明的 DTD 信息)加了一个稍微复杂的限制。在内部子集中使用的参数实体只可以包含完整的声明,而不是上面使用的部分声明。这极大地限制了内部子集中参数实体的灵活性,也正是因为这个原因(当然还有其它原因),最好将你的 DTD 放于外部文件中。

外部实体让开发人员可以连接完全位于他们的文档或 DTD 之外的材料,使用的语法与前面用于连接外部文件的文档类型声明相同。实体值必须使用 SYSTEM 识别符,也可以在合适的时候使用 PUBLIC 识别符。多数开发人员可能会发现他们通常在使用 SYSTEM 识别符连接外部文档,除非文档管理系统已经变为供大多数文档用户访问使用。

通过外部实体连接的文件内容只需要在使用它们的上下文中有意义。一般说来,它们由声明或声明的各个部分组成,但在某些特定的情况下也可以包含二进制数据(如 GIF 文件)。参数实体常常用于结合几个 DTD 子集或包含较大的通用实体列表。外部实体引用的文件还可以包含其它外部实体。如果这些应用是循环的话分析程序将返回错误。例如,如果文档 A 引用文档 B,文档 B 指向文档 C,而文档 C 又返回来引用文档 A,那么分析程序就会停止。外部实体可以是带有许多分枝的树状结构,但这些分枝不允许“生长”回到树干里。当然,如果分枝太多的话,就会产生无法控制的 DTD,这种 DTD 几乎无法让人理解。

我们的外部实体示例将简单地结合两个通用实体,在单个 DTD 中使用它们(本章后面将介绍使用参数实体来嵌入更复杂的 DTD)。使用外部实体声明来包含批注总是好主意——因为 SYSTEM 识别符中的 URL 和 PUBLIC 识别符中更完整的信息经常是隐藏的。我们的第一个实体文件(companies.dtd)包含:

```
<! ENTITY GLW "Corning Incorporated" >
<! ENTITY IBM "International Business Machines" >
<! ENTITY T "American Telephone and Telegraph" >
```

我们的第二个文件(states.dtd)包含:

```
<! ENTITY NC "North Carolina" >
<! ENTITY ND "North Dakota" >
<! ENTITY NJ "New Jersey" >
<! ENTITY NM "New Mexico" >
<! ENTITY NY "New York" >
```

我们的 DTD(尽管很简单)也存放在一个外部文件中:

```
<! -The following entity connects to a list of companies using stock ticker symbols as entity references. ->
<! ENTITY % companies SYSTEM "companies.dtd" >
<! -The following entity connects to a list of states using postal abbreviations as entity references. ->
<! ENTITY % states SYSTEM "states.dtd" >
```

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

```
%companies;
```

```
%states;
```

使用这些引用的样本 XML 文件如下所示：

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

```
<! DOCTYPE DOCUMENT SYSTEM "penex.dtd" >
```

```
<DOCUMENT>The company &GLW; is headquartered in &NY;,. as is &IBM;,. &T; is head-  
quartered in &NJ;.< /DOCUMENT>
```

分析程序对该 XML 文件进行分析后将产生图 5-9 中所示的结果：

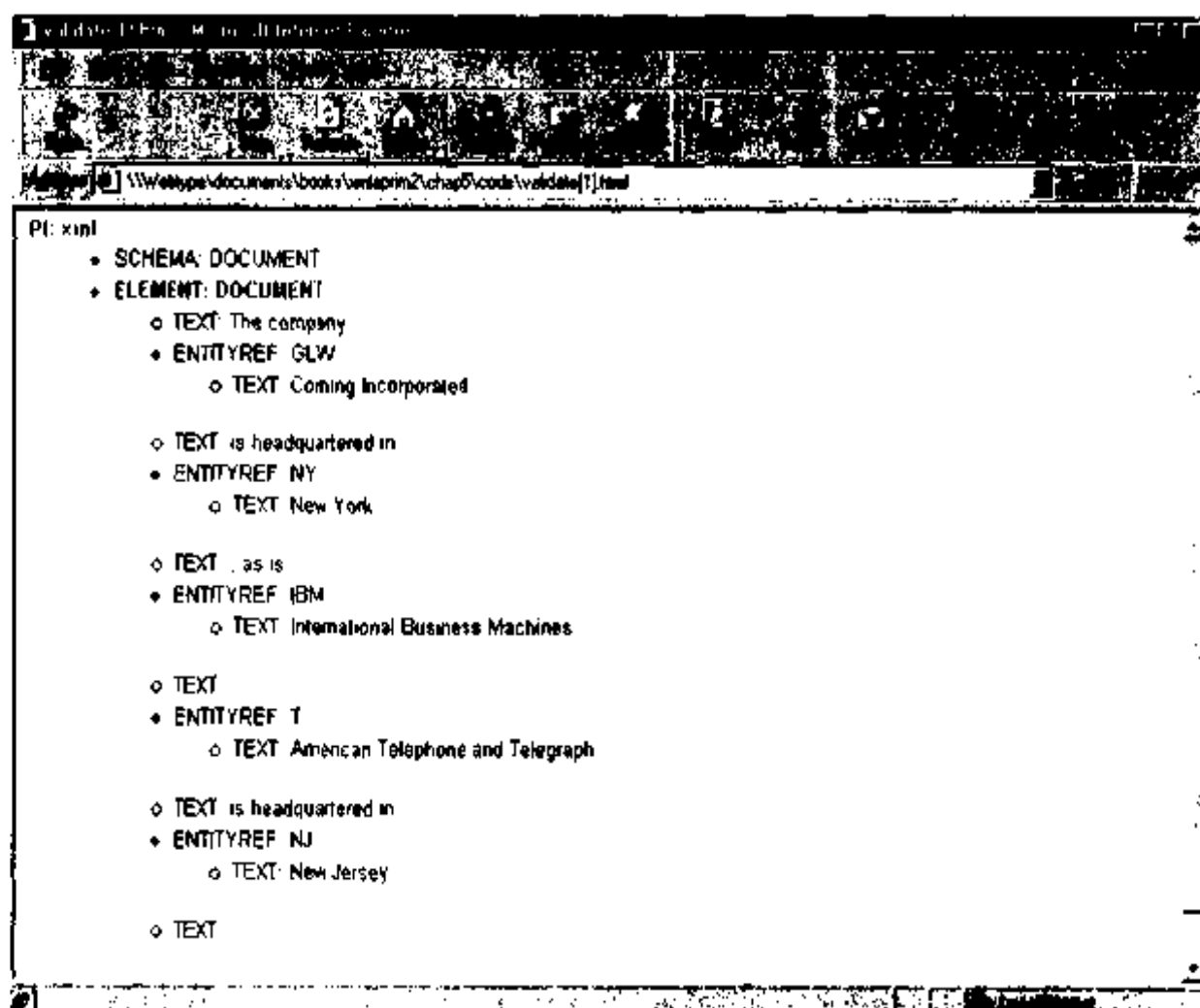


图 5-9 Internet Explorer 5.0 将解析参数实体引用的通用实体

我们在以后的章节中将会看到,参数实体将是简化复杂标记和管理多个 DTD 的非常有用的工具。

注解声明

注解声明就是宣布文档中需要来自外部(非 XML)源的数据并通知应用程序这些非 XML 数据需要处理。注解声明必须与外部未解析实体一块儿使用以识别该实

体代表的信息类型,有时也与处理指令结合使用以提供处理文档中非文本数据的方式。注解声明告诉处理器有什么类型的信息,处理指令宣布应该对它怎么处理。注解声明也可以用作属性值。

注解声明的语法与文档类型声明相似,如下:

```
<! NOTATION Name ExternalID>
```

注解的名称必须由字母、数字、句点、破折线、下划线或冒号组成,并且第一个字符只能是字母或下划线(如前面所描述的,由于名称空间的原因,不鼓励使用冒号)。典型的注解声明可能是下面的样子:

```
<! NOTATION gif SYSTEM "image/gif">
```

分析程序不对 ExternalID(外部 ID)处的信息作任何检查,只是把它传递给处理应用程序。如果处理应用程序能够处理该信息,那最好不过了;如果不能,那也与分析程序无关。关键字 SYSTEM 后面可以引用一个能够显示该数据的应用,但并不要求处理程序使用这一应用(如果某个 Macintosh 或 Unix 用户在阅读这个文件,那么 Windows 的可执行文件无法为其提供帮助)。更典型的是在 Web 上,注解声明将指定 MIME 内容类型。标识符 PUBLIC 也可以用来识别注解,这在 SGML 中很常见。处理应用程序无法理解的注解可能是错误。但不是 XML 错误,分析程序将继续工作而不报错。当然,该应用程序定会宣布它自己的错误。

DTD 中的标记区域:IGNORE 和 INCLUDE

需要测试不同结构且同时跟踪替代结构的开发人员可以使用 DTD 中的 IGNORE(忽略)和 INCLUDE(包含)标记区域。IGNORE 和 INCLUDE 让开发人员打开或关闭 DTD 中的某些部分,它们对于结合几个 DTD 需要限制多个文件冲突的开发人员或者是需要创建具有可选子集的单个核心 DTD 的开发人员尤其有用。IGNORE 和 INCLUDE 区域可以嵌入到其它 IGNORE 区域之中,但跟元素一样,它们的始端和尾端不能交叠。

IGNORE 和 INCLUDE 的语法与 CDATA 相似,如下:

```
<![IGNORE[ declarations ]]>
```

```
<![INCLUDE[ declarations ]]>
```

IGNORE 和 INCLUDE 都不可以出现在声明中间;二者必须指定单个声明或一组声明。例如:

```
<![IGNORE[<![ELEMENT YUCK (#PCDATA)]>]]>
```

```
<![INCLUDE[<![ELEMENT HOORAY (#PCDATA)>]]]>
```

以上例子将使 YUCK 元素不被分析,并允许 HOORAY 接受正常的分析。通过这种方式使用的 IGNORE 看上去是将无用部分编辑出 DTD 的方便途径,而 INCLUDE 在这里看上去纯属多余。参数实体使 INCLUDE 和 IGNORE 能够成为有意义的 XML 符号集添加物。开发人员不是直接使用 INCLUDE 和 IGNORE 在通篇 DTD 中改变代码,而是使用参数实体在一个地方作所有的改动。这使得 INCLUDE 和 IGNORE 相当方便,在有些时候甚至是必不可少的。以下例子简单地演示了这一点:

```
<![ ENTITY % invoice "IGNORE" >
<![ ENTITY % receipt "INCLUDE" >
<![ [%invoice; [
<![ ENTITY notice "Please remit the following payment within thirty days.">]]>
<![ [%receipt; [
<![ ENTITY notice "Thank you for your prompt payment. The sums below have been collected and
recorded.">]]>
<![ ENTITY address "555 Twelvetwelve Lane">
```

依照赋给 invoice 和 receipt 的值,该通用实体消息将提供收帐员的话音或售货机的感谢话语。要改变其输出,只需要切换两个实体的值即可。而 address 实体的值(可能是公司的地址)在两种情形中是一样的。类似的标记在通篇 DTD 中都会出现,不适合 receipt 的部分都被删去。要将 DTD 在 receipt 和 invoice 之间切换,只需要编辑该文件中的两行代码,而不用查找和替换整个文档。在下一章我们将进一步探讨这一强大工具的更多用途。

注意

在 SGML 中,IGNORE 和 INCLUDE 也可以在文档中工作。但 XML 却限制它们只能在 DTD 中工作。

综述

虽然 XML 文档类型定义的语法看上去有点儿怪,并且还遗漏了一些东西(如数据类型),但它实际上并不难理解。不幸的是,在 XML 的表面之下潜伏着一些复杂的问题,它们有可能带来严重的麻烦。良好构造性和有效性之间的区别是最危险的区域之一,本章提供的有关非检验分析器的示例使这种区别虽然按照第 3 章的描述,简单的构造良好的文档在多数情况下都能工作,但许多良好的文档也使用本章描述的一些工具。并非所有这些工具都必须在只检查构造良好性的分析程序中具备,这

使得解释 XML 文档的文式可能有所变化。

基本问题来自文档和解释它们的分析程序(在 XML 规范中叫做处理器)之间的分离。分析程序可能是检验的,也可能是非检验的。检验分析程序检查文档以确保它是有效的,而非检验分析程序只检查它们的构造良好性。但非检验分析程序的确在读取 DTD 信息,为属性赋予默认值并根据 DTD 中的信息扩充实体。仅凭这一点就可以认为即使为不打算对其进行检验的文档创建 DTD 也经常是很有用的。此外,最好确保你为文档使用的任何 DTD 都真正提供检验所需的完整结构,即便你只是为属性获取默认值。

文档可能遵守使它们有效和构造良好的成套规则,也可能不遵守。如果检验是一个简单的构造良好性超集就好了,因为这样可以使所有有效文档都是构造良好的。但实际上它并不完全是这样。所有有效文档在技术上都是构造良好的,但非检验分析程序(即只检查构造良好性的分析程序)可以不把外部的东西(一般是 DTD 或实体)载入该文档。因此,在检验分析程序下能够工作的文档可能在不载入文档所引用信息的非检验分析程序中不能工作。它可以接受分析,但可能出现分析程序不报告的丢失信息和错误。

这使得有可能潜伏着一些灾难性的情况,非检验分析程序的行为各不相同,Microsoft 的 Aelfred 读取外部 DTD 并使用默认值,而 James Clark 的 Expat 就不这样(它可以选择性地读取外部的通用实体)。Netscape 的 Mozilla 源代码开放浏览器项目和 Perl 的 XML::Parser 都使用 Expat。因此,依靠属性默认值的应用程序可能会发现当用非检验分析程序取代检验分析程序时这些值丢失了。发生这种现象的原因有如下几个:

- 用户将浏览器的设置改成了快速载入模式
- 开发人员已决定改变应用程序中使用的分析程序(由于 SAX 和 DOM 标准,这一点不难做到)
- 为在某个应用程序中使用而设计的文档现在需要在另一个不同的应用程序中重新使用

如果打开你的 XML 文档发现有未被扩充的实体(如:“段落中间的这个 &sunset 是什么?”)或者有些功能不见了(可能是因为丢失了属性的默认值),那么你遇见的就是这个问题。避免发生这种问题的方法看上去很合理,但不一定总是奏效。一种方法是在内部子集中包含所有的 DTD 信息。将所有声明都放入每个文件中,会确保即使分析程序不获取外部资源也有一套完整的声明供它操作使用。这要求集中收集所有声明并把它们堆放到位于使用外部资源和不使用外部资源的应用程序之间关键接

口处的主要文件中(目前这方面还没有有效的开发工具)。W3C 的 XML 文档语法工作组正在考虑一种规范的 XML 格式,在这种格式中所有声明都是已被扩充的,希望这种格式能解决此问题。另一种避免该方法只有那些完全控制环境的人(一般是不在使用浏览器作为界面的开发人员)才能使用,这种方法使用外部资源并直接认为其它工具不能使用这些文件。

交叉参考



要完全地了解非检验分析程序必须做的事情,请参考 John Cowan 的《Non-Validating XML Parsers: Requirements》(非检验 XML 分析程序之要求)一书,所在的站点是 <http://www.lists.ic.ac.uk/hypermail/xml-dev/9808/0019.html>。要更多地了解 Expat 的行为,请见站点 <http://www.jclark.com/xml/expat.html>。

最后一个关于 XML 分析程序要注意的是:有些分析程序比其它分析程序更加遵循 W3C 的推荐标准。有些分析程序(如 Aelfred)已被建造得具有最高的效率和最小的文件尺寸。因此,它故意放弃了一些比较模糊的构造良好性检查(这是经常变动的,详细细节请见站点 <http://www.microstar.com/aelfred> 上最新的 readme 文件)。所以 Aelfred 可以接受一些比它严格的分析的分析程序所忽略的文档。至于有些分析程序宣称它们百分之百地遵循 W3C 规范,由于现在还没有完善的官方测试工具,所以还不能证实。OASIS(<http://www.oasis-open.org>, 是一个 SGML 联盟组织)正在从事测试工具的工作,但还没有发布任何测试工具。到现在为止。混合分析程序看上去在大多数情形中都还可以。W3C 的规范虽然不好读懂,却相当精确。但如果你发现有一两个小故障发生时,一定要检查你的工作是否使用了多个分析程序(或使用不同分析程序的多个应用程序)并通知分析程序或应用程序开发人员。

既然我们已经拥有了这些工具,接下来应该开始让它们工作。下一章将把这些工具用于真正的文档和实际工作中去。

第 六 章

利用 XML 重新建立 Web 和网页文档

现在我们已经学习了各种基础知识,该是用较复杂的数据类型定义 DTD 来创建一些有用的 XML 文档的时候了。不管这将包含多少部分的内容,创建 DTD 并不使你感到痛苦。开发者在把 HTML 或字处理类型文档转换成 XML 的过程中,发现它们的文档结构的确是非常简单的。本章将检查并实现两个例子 DTD 文档的生成。其中包括 DTD 的开发、文档编码、为浏览文档而进行的样式表创建以及包装 DTD 以便能处理传统的文本内容。

注意



本章的所有示例均不包括对其它文档的链接。在 XML 中的链接要比在 HTML 中复杂得多,在第 10、11 章将会有专门的介绍。现在,使自己尽快熟悉 XML DTD 的语法并且学会一个文档而不是整个站点的结构。

从 HTML 到 XML

许多机构已经用 HTML 存储了大量的信息,在劳神费时地从其它文档格式转换成 HTML 后,维护这些信息的人极不愿意听到要从 HTML 转向新型文档格式的消息。但是,并不是所有已创建的 HTML 文档都将转换成 XML,即使要转换,也会是自动进行的,不会再是痛苦的转换过程。因为 HTML 在开发时都遵循以特定的方式显示出来,所以完成的 HTML 文档在屏幕上的可视性比代码的结构性有更高的优先级。开发者在最后期限到来之时必须给客户看完成的文档,如果打乱的代码并不影响浏览器的工作,同样也不会影响客户审查文档。这样做的前提是 HTML 文档是用手工形成的(或者所用的工具用附加的标记弄乱了页面),不固定的 HTML 用良好形式的 XML 来表达,降低了其随意度和急躁性。

HTML 提供了一套结构来代表文档类型,并加上一套工具用来格式化这套结

构。HTML 并没有为特体的文档或数据特征提供符号集,这一段时间以来,HTML 似乎失去了其描述结构的符号集的地位,而变成了描绘格式的符号集。随着样式表地位的提升 W3C 希望提升其作为 HTML 规范所有者的地位,HTML 逐渐重新回到了其描述结构的本位,把格式化的功能扔给了样式表。最近的 HTML 标准已经把 SGML 用作其正式的符号集,在 SGML 中创建了对“合法”HTML 的描绘。下一版本的 HTML,也就是称为 XHTML,将在后面的第十三章进行深入讨论,它把 HTML 包含在了一系列 XML 模块中。如果所需要的仅仅是技术上合法的 XML 文档,那么就可以从目前的各种格式转换而来。但需要牢记的是合法的 XML 并不需要进行容易的管理。目标符号集反映的仅仅是特定的内容,而不是普通结构(或者更差,表达方式)。这就是把 XML 当成一种文档格式的优势所在,并且是用 HTML DTD 并不能提供的。

注意

如果你在文档中使用了一些 HTML 元素,John Cowan 的 Itsy-Bitsy Teeny-Weeny Simple Hypertext (IBTWSH) DTD 也许会有一些作用。在下列网址中可以找到它:<http://www.ccil.org/~cowan/XML/ibtwsh.dtd>,描述了 HTML 的一个子集并把所有元素的名字都以大写字母表示。

对于许多 Web 文档,转换过程会手工执行,因为它可能是书面文档。工具可以从一系列页面中学会格式并且产生所需的内容,但是不由机器生成的 HTML 系列,页面很少遵循持续的格式。在一页上的文本也许包含了五个段落,但另外一页也许根本就没有文本。例如:

```
<HTML>
<HEAD> <TITLE> Joe's Catalog - Money Counters< /TITLE>< /HEAD>
<BODY BGCOLOR="#FFFFFF">
<H1>Money Counting Equipment< /H1>
<H2>Basic Money Counter< /H2>
<H4>Count your cash without spending all of it! < /H4>
Joe's is pleased to announce this NEW addition to our line. People with piles of change can sort
their money easily, and wrap it for the bank. Makes the change box a lot more useful! <BR>
Price:<B> $ 14.95< /B>, <FONT SIZE=1>plus $ 4.95 shipping and handling. < /FONT>
<P>
Also available: Paper Coin Wrappers, bag of 100: <B> $ 2.95< /B><P>
<H2>Standard Money Counter< /H2>
<H3>Count and collect your cash automatically! < /H3>
This money counter wraps your change automatically-just feed it the plastic change rolls. You'll feel
just like the bank when you read its LED display announcing how much change you've gathered. <P>
>
```

```

Special guarantee: 100% accuracy on wrapping or your money back! <BR>
Price:<B> $ 64.95< /B> , <FONT SIZE=1>plus $ 7.95 shipping and handling. < /FONT>
<P>
Also available: Plastic Coin Wrappers, box of 400: <B> $ 10.95< /B><P>
<H2>Super-Duper Money Counter< /H2>
Tired of change? This machine counts bills as well. Feed it the take from a cash register and watch
it count away. Spits out old bills in a separate tray for easy counting. Saves hours of effort spent
counting pennies and twenties! <BR>
Price:<B> $ 649.95< /B> , <FONT SIZE=1>plus $ 29.95 shipping and handling. < /FONT>
><P>
Uses plastic coin wrappers above, and Paper Bill Wrappers, box of 1000: <B> $ 10.95< /B>
<P>
< /BODY>< /HTML>

```

这个文档在浏览器中展现了一个简单的点钞机的目录,如图 6-1 所示。

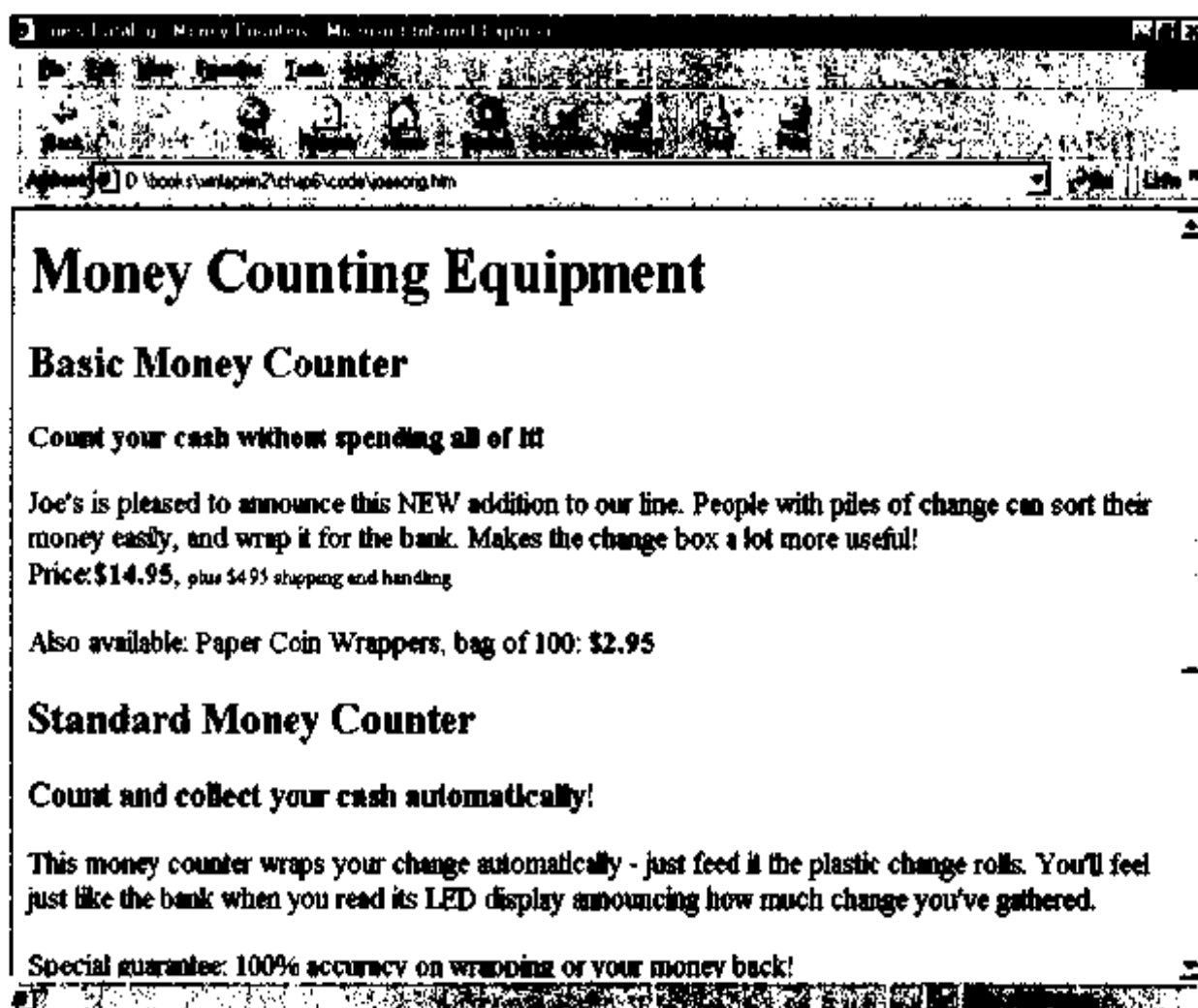


图 6-1 Joe's Catalog 的 HTML 版本

虽然这个文档有一些明确的部分能非常容易地转换成 DTD,但编码的样式并不能准确地引导进行面向 XML 的转换。文本被“extra guarantees”和“also availables”弄乱了,而且表现得既不连续而且不能使用相同的格式。这看起来象老式的手工编码的 HTML 文档,它在段落结尾使用<P>标记而不是用<P>...< /P>。虽然可以写一个程序来进行转换,joe 公司既可以通过发展 DTD 来手工转换,也可以通过增加结束标记来形成的格式。

一个趋向于内容而不是表现方式的 DTD 能适用于 Joe's Catalog。如下图所示：

```
<! ELEMENT CATALOGPAGE (HEADER,CATALOGITEM+ )>
<! ELEMENT HEADER ( #PCDATA)>
<! ELEMENT CATALOGITEM ( ITEMNAME, ITEMSUBHEAD?, ITEMDESCRIPTION *,
ITEMPRICING,SUBITEM* )>
<! ELEMENT ITEMNAME ( #PCDATA)>
<! ELEMENT ITEMSUBHEAD ( #PCDATA)>
<! ELEMENT ITEMDESCRIPTION ( #PCDATA)>
<! ELEMENT ITEMPRICING ( #PCDATA|ITEMNOTE|PRICE|SHIPPING) * >
<! ELEMENT PRICE ( #PCDATA)>
<! ELEMENT SHIPPING ( #PCDATA)>
<! ELEMENT ITEMNOTE ( #PCDATA)>
<! ELEMENT SUBNAME ( #PCDATA)>
<! ELEMENT SUBITEM ( #PCDATA|SUBNAME|PRICE|SHIPPING) * >
```

Joe's Catalog 页面还需要进行细致的修改。举个例子来说,在 DTD 中增添 SHIPPING 元素,在以前并没有正式提出过。另外必须添加的元素层在以前也没有标示出来。基于这些原因,Joe's Catalog 也许需要使用人工而不是机器进行重建。新的页面看起来如下所示：

```
<? xml version="1.0" encoding="UTF-8"? >
<! DOCTYPE CATALOGPAGE SYSTEM "joes.dtd">
<CATALOGPAGE>
<HEADER>Money Counting Equipment< /HEADER>
<CATALOGITEM>
<ITEMNAME>Basic Money Counter< /ITEMNAME>
<ITEMSUBHEAD>Count your cash without spending all of it! < /ITEMSUBHEAD>
<ITEMDESCRIPTION>Joe's is pleased to announce this NEW addition to our line. People with
piles of change can sort their money easily, and wrap it for the bank. Makes the change box a lot
more useful! < /ITEMDESCRIPTION>
<ITEMPRICING>Price: < PRICE> $ 14. 95< /PRICE>, plus < SHIPPING> $ 4. 95< /SHIP-
PING> shipping and handling. < /ITEMPRICING>
<SUBITEM>Also available: < SUBNAME>Paper Coin Wrappers, bag of 100< /SUBNAME>:
< PRICE> $ 2. 95< /PRICE>< /SUBITEM>
< /CATALOGITEM>
<CATALOGITEM>
<ITEMNAME>Standard Money Counter< /ITEMNAME>
<ITEMSUBHEAD>Count and collect your cash automatically! < /ITEMSUBHEAD>
<ITEMDESCRIPTION>This money counter wraps your change automatically-just feed it the plastic
change rolls. You'll feel just like the bank when you read its LED display announcing how much
change you've gathered. < /ITEMDESCRIPTION>
<ITEMPRICING>< ITEMNOTE>Special guarantee: 100% accuracy on wrapping or your money
```

```

back! < /ITEMNOTE>
Price: < PRICE> $ 64. 95 < /PRICE> , plus < SHIPPING> $ 7. 95 < /SHIPPING> shipping and
handling. < /ITEMPRICING>
< SUBITEM> Also available: < SUBNAME> Plastic Coin Wrappers, box of 400 < /SUBNAME> :
< PRICE> $ 10. 95 < /PRICE> < /SUBITEM>
< /CATALOGITEM>
< CATALOGITEM>
< ITEMNAME> Super-Duper Money Counter < /ITEMNAME>
< ITEMDESCRIPTION> Tired of change? This machine counts bills as well. Feed it the take from a
cash register and watch it count away. Spits out old bills in a separate tray for easy counting.
Saves hours of effort spent counting pennies and twenties! < /ITEMDESCRIPTION>
< ITEMPRICING> Price: < PRICE> $ 649. 95 < /PRICE> , plus < SHIPPING> $ 29. 95 < /SHIP-
PING> shipping and handling. < /ITEMPRICING>
< SUBITEM> Uses plastic coin wrappers above, and < SUBNAME> Paper Bill Wrappers, box of
1000 < /SUBNAME> : < PRICE> $ 10. 95 < /PRICE> < /SUBITEM>
< /CATALOGITEM>
< /CATALOGPAGE>

```

这个文档的结构使得 Joe's Catalog 页面看起来更象能进行机器可读的, 正如图 6-2 所示。许多不同的部件被逻辑分离开, 使得应用程序能轻易地存取和处理信息。

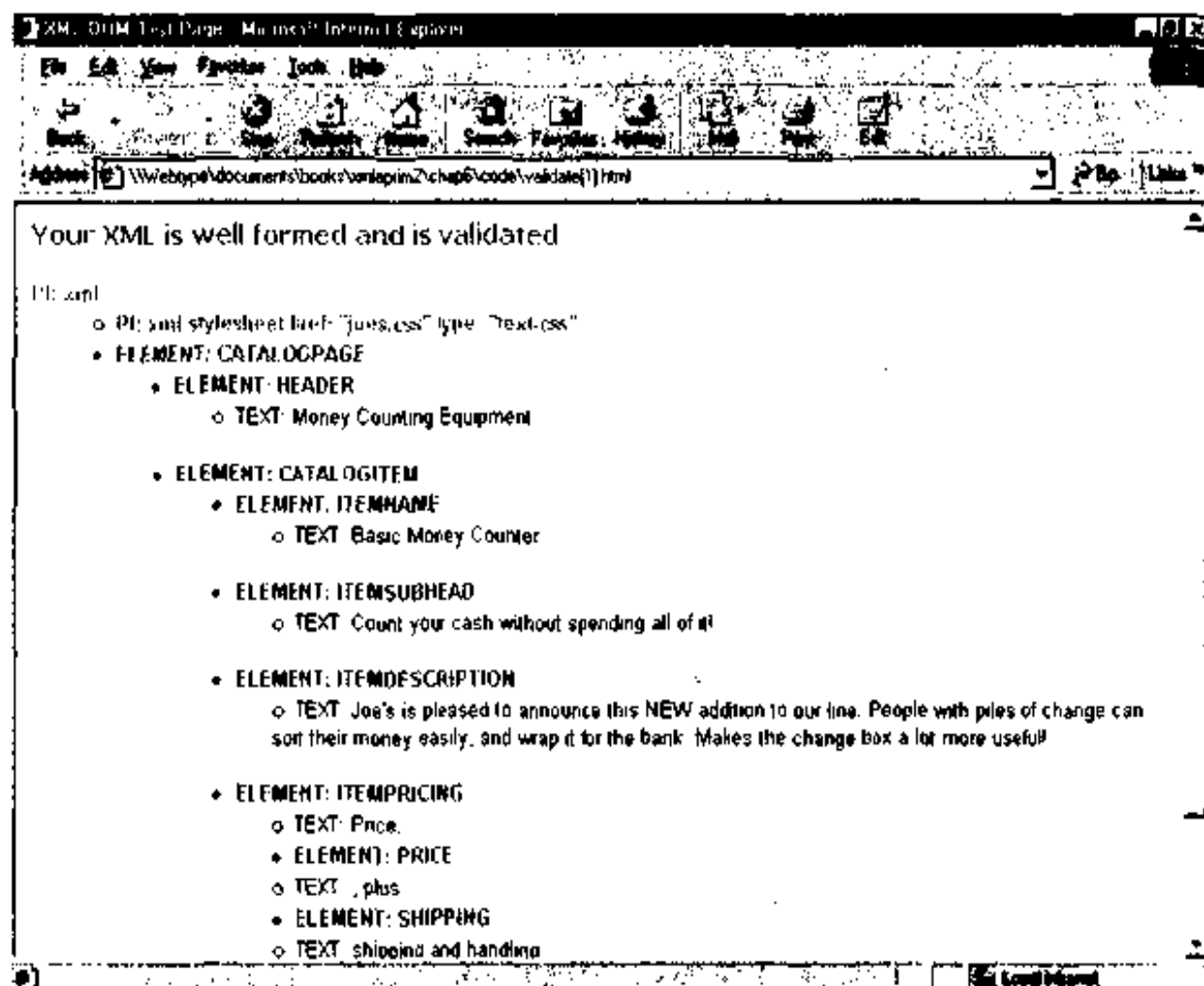


图 6-2 用针对内容的符号表示的 Joe's Catalog

这仍然是一个未完成的解决方案, 虽然也许它已经满足了 Joe's Catalog 的所有要求。一个更为深入的 DTD 设计也许能改变表现信息的种类和顺序, 使得它和第 3

章中提到的 Jimbo's Super Clock Catalog 例子中的部分顺序和信息非常类似。目录现在看起来更便于人们阅读,但是对于库存系统和目录管理工具来说并不是很有效。

提示

由计算机产生的页面将会很容易地进行转换,至少代码或者标记写得很智能。寻找替换机制在所有的材质都使用相同的标记时能转换静态的机器产生的页面。更好的是,如果数据在原来的结构下仍然有效(数据库表格)并能很快地创建页面时,创建 HTML 的脚本可以很容易地进行修改来生成 XML。脚本可以通过采用<PRICE>元素来取代围绕 price 的各种格式标记。

有些 HTML 文档现在已经为 XML 作好了准备,HTML 开发者能在设计页面时提前准备,使得转换更加容易。在第二章中,我们讨论了级联样式表(CSS)如何提供 CLASS 属性,来使 HTML 开发者方便有效地创建他们自己的标记。CSS 已经使用了近一年的时间,许多站点已经利用了它容量上的优势来创建网站。开发者使用 CSS 类别来反映内容的种类,并不仅仅用来进行格式化。使用 CSS CLASS 属性创建的结构和 XML 创建结构的主要区别是:CSS 并不需要类似特定顺序进行嵌套,而且因为 CSS 提供了在元素之间进行继承性质格式化的功能(子元素将被默认接收到其父辈的样式属性),许多利用 CSS 创建的页面反映出 XML 的良好结构并能很好地进行转换。它们能在新版本的 Netscape 和 Microsoft 浏览器中工作得很好,但在老版本中将会出现问题。如下的例子使用了一个小的分类元素来显示 CSS CLASS 属性如何工作在 Joe's Catalog 页面上:

```
<HTML>
<HEAD><TITLE>Joe's Catalog - Money Counters</TITLE>
<STYLE TYPE="text/css">
DIV. HEADER {font-size:24pt;font-weight:bold}
DIV. ITEMNAME {font-size:18pt;font-weight:bold}
DIV. ITEMSUBHEAD {font-size:14pt;font-weight:bold}
DIV. ITEMDESCRIPTION {font-size:12pt}
DIV. ITEMPRICING {font-size:10pt}
SPAN. PRICE {font-weight:bold}
SPAN. SHIPPING {font-weight:normal}
DIV. SUBITEM {font-size:11pt}
SPAN. SUBNAME {font-weight:bold}
</STYLE></HEAD>
<BODY BGCOLOR="#FFFFFF">
<DIV CLASS="HEADER">Money Counting Equipment</DIV>
<DIV CLASS="ITEMNAME">Basic Money Counter</DIV>
<DIV CLASS="SUBHEAD">Count your cash without spending all of it! </DIV>
```

```
<DIV CLASS="ITEMDESCRIPTION">Joe's is pleased to announce this NEW addition to our line.  
People with piles of change can sort their money easily, and wrap it for the bank. Makes the  
change box a lot more useful! </DIV>  
<DIV CLASS="ITEMPRICING">Price: <SPAN CLASS="PRICE"> $ 14.95</SPAN>, plus <  
SPAN CLASS="SHIPPING"> $ 4.95</SPAN> shipping and handling. </DIV>  
<DIV CLASS="SUBITEM">Also available: <SPAN CLASS="SUBNAME"> Paper Coin  
Wrappers, bag of 100</SPAN>; <SPAN CLASS="PRICE"> $ 2.95</SPAN></DIV>  
</BODY></HTML>
```

如果目录被组织成这样,向 XML 的转换也许需要顾客代码来替代 DIV 和 SPAN 元素,而相应地使用适当的元素(在这种情况下,用它们的 CLASS 属性)并用相应的父元素来对它们进行分组。大多数开发者并没有如此幸运,除非他们刚刚开始建立自己的 HTML 文档。

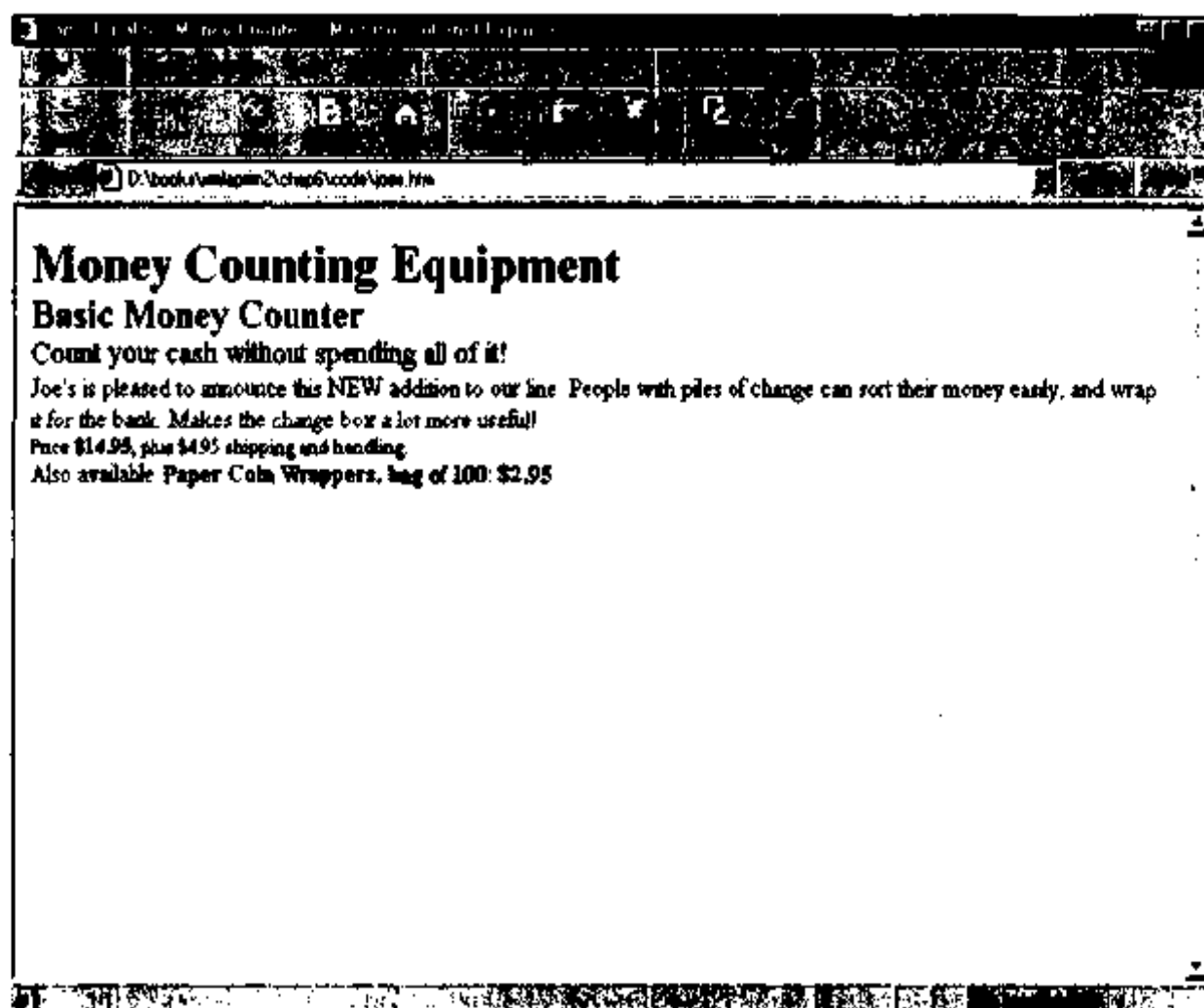


图 6-3 使用 DIV 和 SPAN 元素的 CSS 允许开发者区分 HTML 内部的格式和内容

交叉参考



并于目录文档需求的更加复杂的分析将在下一章里实现,Joe's Catalog 已经超过了 HTML 的范畴,但它将仍然面对更多有趣的重建过程。

为 Joe's Catalog 使用 CSS 建立一个样式表并不是很困难,各个部分很容易地进行区分,并且在 HTML 版本中格式化过程已经存在了。把一个样式表链接到 XML

文档中需要一些额外的标记。每个使用样式表的文档都需要在定义根元素外观前使用下列的一行代码：

```
<? xml-stylesheet href="joes.css" type="text/css"? >
```

现在最大的问题是 Joe 公司是否想改变其产品目录的外观。如果不想,那么就能非常容易地建立相应的样式。CSS 显示如表 6-1 所示的属性,使浏览器能计算出在这个文档中的 XML 元素进行渲染时所需要的基本结构。

表 6-1 显示建立文档展示结构的属性值

显示值	意义	备注
Block	将元素标示为一个独立的块,其文本从一个新行开始,在该元素末尾的断点处结束。通常使用块来对根元素进行格式化,就象许多类似段落的子元素一样。	出现在一级标准中,但最近才得到支持。其行为与 HTML 的 DIV 元素相似。
Inline	将元素格式化为当前块的一部分,前后都没有断点。一般用于格式化部分句子或段落的元素。	出现在一级标准中,但最近才得到支持。其行为与 HTML 的 SPAN 元素相似。
ListItem	使元素的行为与 HTML 的列表项元素(LI)类似,将其以嵌入或项目列表形式出现取决于其它 CSS 属性。	出现在一级标准中,但最近才得到支持。其行为与 HTML 的 LI 元素相似。
None	不显示元素,也不显示子元素。(要隐藏元素但允许它们的子元素出现,应使用 visible 属性)。	出现在一级标准中,常用于动态 HTML 中。
run-in	如果元素跟有一个块元素,该元素将加入块元素中,仿佛它是内联的;否则,该元素将被格式化为一个块。	二级标准中的新成员。
compact	将列表容纳在某个空间中。	二级标准中的新成员。
marker	用 CSS 内容标示某元素。	二级标准中的新成员。
inherit	让元素使用其父元素的显示属性。	出现在一级标准中。
table, inline-table, table-row-group, table-header-group, table-footer-group, table-row, table-column-group, table-column, table-cell, table-caption	用于使元素进入表格的组件,详见后面的章节示例。	二级标准中的新成员。

通过用其它格式属性来组合这些显示属性,我们可以为 Joe's Catalog 建立一个简单的样式表。

```
HEADER {
    display: block;
    font-size: 24;
    font-family: serif;
    font-weight: bold;
}
CATALOGITEM {
    display: block;
    font-family: serif;
}
ITEMNAME {
    display: block;
    font-size: 18;
    font-weight: bold;
}
ITEMSUB-HEAD {
    display: block;
    font-size: 14;
}
ITEMDESCRIPTION {
    display: block;
    font-size: 12;
}
ITEMPRICING {
    display: block;
}
PRICE {
    display: inline;
    font-family: bold;
}
SHIPPING {
    display: inline;
}
SUBITEM
    display: block;
```

这里运用到的主要工具是样式显示属性,它使得我们可以声明变量是否在它本身的段落中或是紧跟在文本中。如果元素能排列自身段落或行的等级,那么它就属于一个块类型。如果不能,它就是一个内联类型。还有一个选项就是独立行,它比块少些空白。它不使用任何的选项来显示其内容,另一些选项(在第二章中进行了详尽的描述)提供了表结构和其它特征。虽然它可能费点神,但上面代码显示的样式产生了一个合理的目录草稿,正如图 6-4 和 6-5 所示。

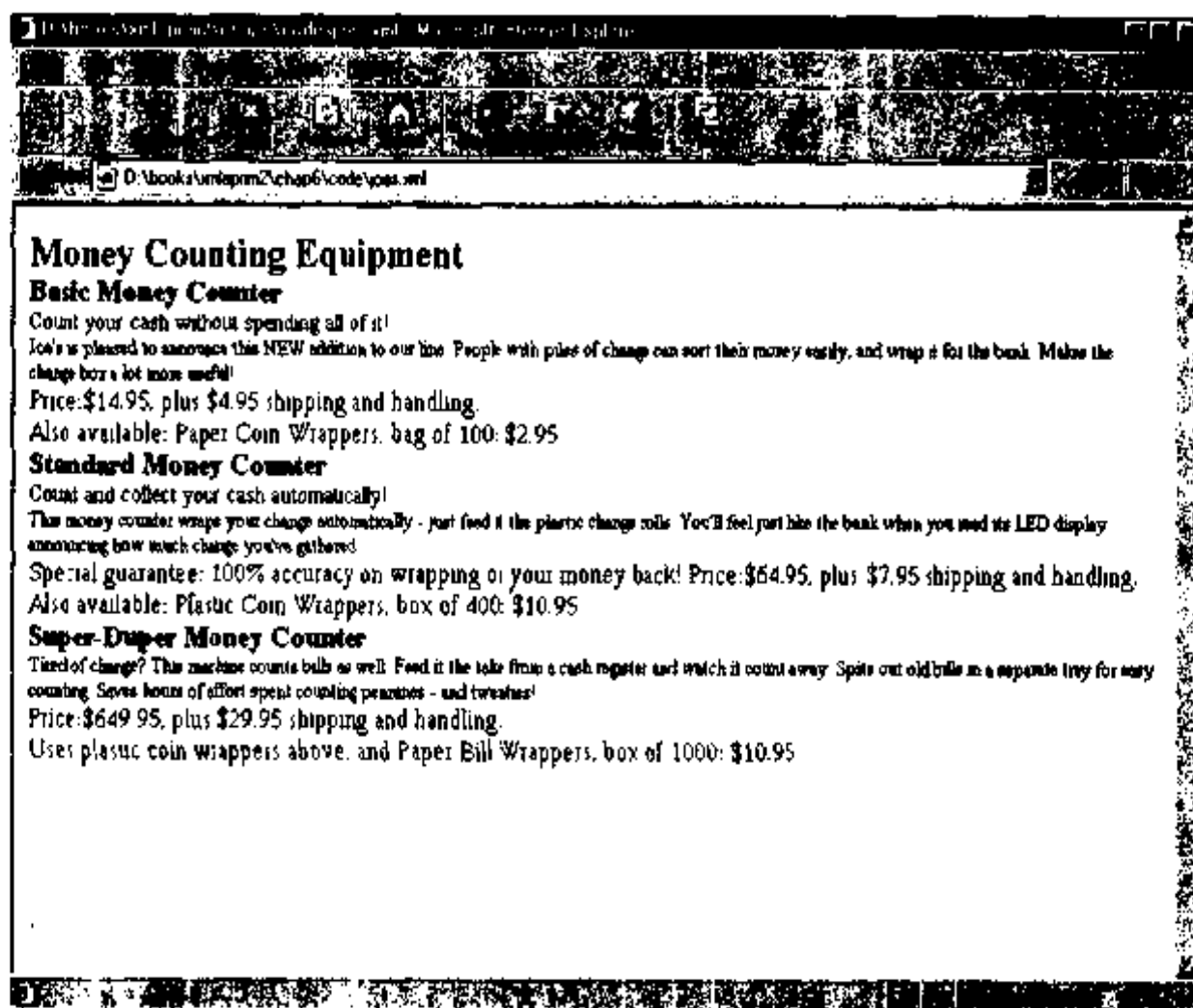


图 6-4 XML 的样式化版本(在 Microsoft 的 Internet Explorer5 中)展示了同一信息的不同版本,而且没有丢失其基本结构

当心

虽然 CSS 和 XML 看起来似乎是天生配对的,但 CSS 一级标准仅仅在 Netscape 和 Microsoft 4.0 版本浏览器中得到了部分的执行(而且当时还有些矛盾)。在这两种浏览器的下一版本中将对 CSS 二级标准给予多大的支持,现在还没有定论。你可以通过浏览器厂商的网站(<http://www.webreview.com> 或 <http://www.webstandards.org>)得到关于指定浏览器支持 CSS 的更详细消息。

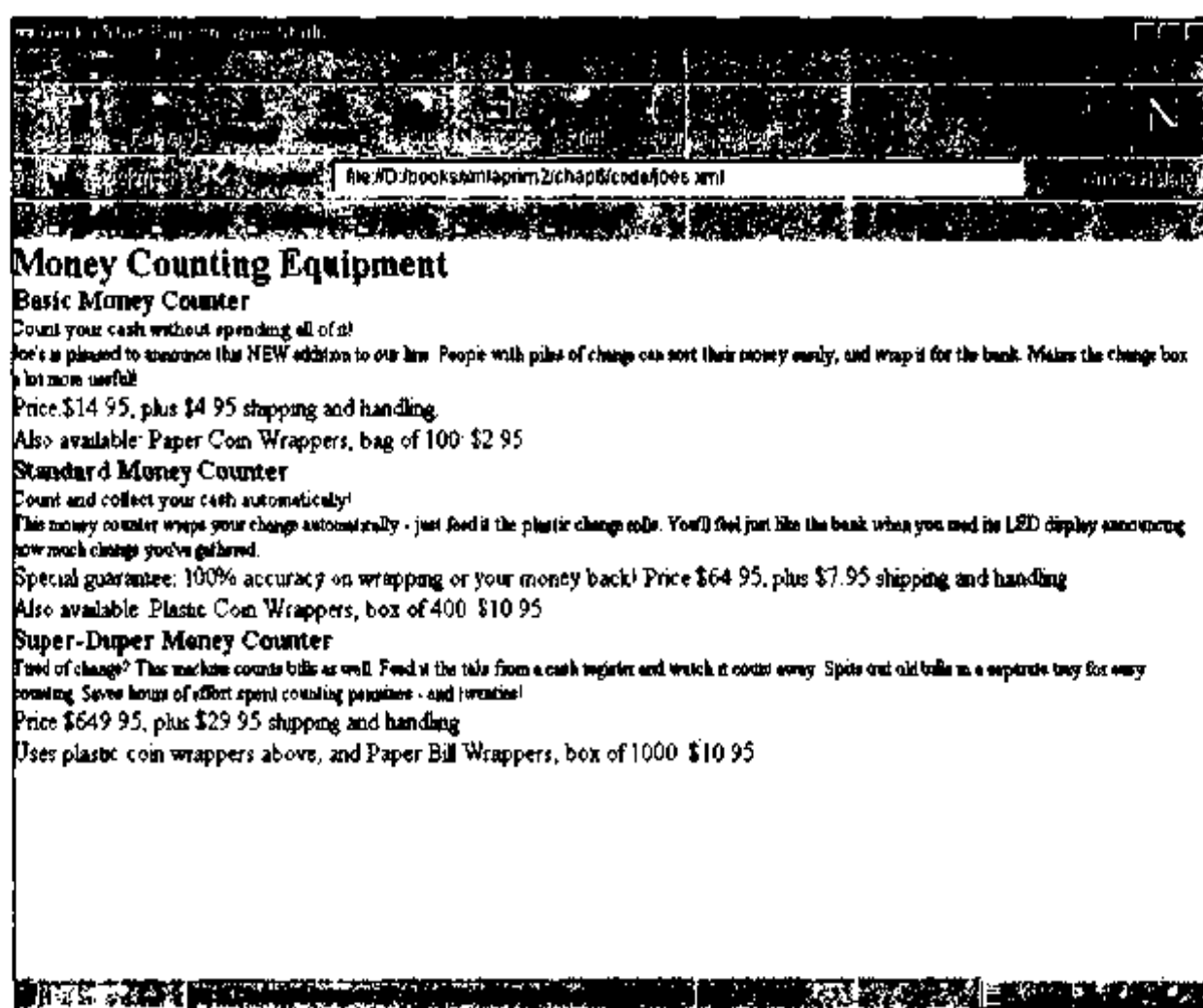


图 6-5 Netscape 的 6ecko 浏览器预览显示了文档的类似视图

当你建立了一个样式表后,对其进行改动很容易——只需在样式表上的作改动,所有的改动就会全部流入该调用样式表的文档。扩展样式表就没有这么容易。假设 PRICE 元素必须以红色显示出来,但仅仅对于 markdown 使用,CSS 提供了选择器来响应属性值(用 XML 则 CLASS 属性并没有任何的特权,这一点和 HTML 不一样)。

```
PRICE [ status = "markdown" ] {
    display: inline;
    font-weight: bold;
    color: red;
}
```

记住在使用属性时,当用到合法值时你必须将它在 DTD 中声明。在所声明的样式中,浏览器或其它格式的应用程式将返回如下的价格为红色的页面,提示价格非常便宜。

```
<price STATUS="MARKDOWN"> $ 1.25< /PRICE>
```

这将持续有效,直到有人决定价格颜色应该在折扣很多而且合格时改变为止。价格需要固定或以计数方式变动。即使下面的样式能在一段时间内工作,它也不能称为一流的代码:

```
PRICE.  
    display:inline;  
    font-weight:bold;  
}  
PRICE.[status="set"]  
    color:blue;  
}  
PRICE.[status="byquantity"]  
    color:green;  
}  
PRICE.[status="markdownset"]  
    color:red;  
}  
PRICE.[status="markdownbyquantity"]  
    color:brown;  
}
```

关于本书

在写本书的第一版时,我使用了两种样式符号。大多数样式都由 Microsoft Word 的内建样式工具和样式表通过 MIS:Press 提供。一些信息——如提示、输出、警告以及边栏——使用的标记符号与 XML 非常类似。举个例子来说,对于 Notes 来说,使用<NOTE>开始,用< /NOTE>结尾。虽然这些解决工具工作得很好,还是存在一些方法进行改进。XML 就定位在许多这样的需要上,能简单化样式表的结构并且在同时能打开关于这个文档的新媒介。

在 Word 中定义的样式结合了许多结构信息和格式化信息,就和表 6-2 所示的一样。

表 6-2 一个书本章节的示例样式表

样式	用法
TOCO CT	章编号和题目。该文本用于内容表格中。
TOC1 A	标题 A,一章中的最顶层标题。该文本用于内容表格中。
TOC2 B	标题 B,标题 A 的下一级标题。该文本也出现在内容表格中。

C	标题 C,标题 B 的下一级标题。不在内容表格中。
D	标题 D,标题 C 的下一级标题。不在内容表格中。
E	标题 E,标题 D 的下一级标题。不在内容表格中。
BT	正文文本,在标题、图标、列表、图形字面或其它独立图形之后的第一个段落或文本。不嵌入。
BT INDENT	BT 元素后面的正文文本。不嵌入。
BL /NL1 TOP	项目列表或编号列表的顶行(由上面的额外空格对其格式化)。
BL /NL2 MID	项目列表或编号列表的中间行(没有上面或下面的额外空格对其进行格式化)。
BL /NL3 BOT	项目列表或编号列表的底行(由下面的额外空格对其格式化)。
GL	词汇表文本,被定义的文字应该是 粗体 ,定义用普通文本。
CC1 TOP	多行代码清单的第一行。由上面的额外空格对其格式化。
CC2 MID	多行代码清单的中间行。上下没有额外空格。
CC3 BOT	多行代码清单的最后一行。由下面的额外空格对其格式化。
CC4 SINGLE	多行代码清单中的单个行。由上面和下面的额外空格对其格式化。
ICON	图标文本,与标记(<NOTE>、<WARNING>等)结合以标示需要从正文文本 调出使用的文本。
FG	图形字面,即图形 chapterNum。chapterNum 应该是 粗体 ,其它用纯文本。
LC	清单字面,位于代码清单的上边。
TBC	表格字面,位于表格的上边。
TBH	表头,在所有列的上边。
TB	表格主体。
UL	未编号的列表。
SN	源注解,即脚注或表格底部的注解。

用标记定义的结构大多数是图标和项目。项目在 PC 中以的形式进入,避免了操作系统间的冲突。Note 的图标通过<NOTE>进行提示。一些结构用简单的格式化进行提示。菜单和关键词用**粗体**表示。标题用*斜体*表现。即使当前模式正在很好地为 MIS:Press 工作,我们将可以通过 XML 来使文档结构定义更简单明了,除非 MIS:Press 开始使用基于 XML 的出版工具,这个努力并不能为它们带来很多的

利益,但它表明了如何把一种格式样式转换成结构样式。这种样式可以从一些冗余中得到解脱,而冗余恰好能形成格式化样式并能把转换文档类型的工作变得简单。

第一关:看起来象旧样式表的 DTD

我们将通过设置章节结构来开始。章节将在最后变成一本原书的一部分。章节包含一个标题,一个介绍性的简介和一系列用 A 头标志的节。为了确保进行平滑的转变,原有的样式名字尽量使用,但空格必须删去:

```
<! ELEMENT CHAPTER (-HEAD?, TOC0CT, INTRODUCTORY, ALEAF*) >
```

HEAD 元素使用 DTD 来描述进行样式表链接。TOC0CT 元素与 TOC0 CT 有同样的职责:识别章节的标题。它仅包含有文本:

```
<! ELEMENT TOC0CT (#PCDATA) >
```

INTRODUCTORY 是一个引入章节中一些老版本中所没有的特性的新元素。所有的章节都在章节标题线后以文本开始。INTRODUCTORY 元素要求必须由至少一个主文本段落组成(目前在介绍中不需要出现任何代码或子元素)。

```
<! ELEMENT INTRODUCTORY (BT+) >
```

BT 元素有一点点复杂,因为其中的文本能包含斜体和粗体。为了满足这些需求,我们将创建一些元素和参数来包含它们,使我们能很容易地把它们包含在其它的元素中。

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

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

```
<! ENTITY % textual-elements "(#PCDATA|CITATION|USERACTION)*" >
```

```
<! ELEMENT BT % textual-elements; >
```

注意

因为文本元素实体描述了混合内容模型,它不能和其它内容模型组合。

下一个目标是 ALEAF 元素。为了避免和 HTML 的 A 标记和 B 标记混淆,我在所有的部分都在字母后面增添了 LEAF。因为 Leaves 可能包含许多内容的种类,所以在该提前计划并创建一个参数实体。内容元素实体将包含一外罗列着所有可用内容类型的列表,并把它给 LEAF 元素,目前其仅仅包含 PARAGRAPH 元素。

```
<! ENTITY % content-elements "BT" >
```

所有 LEAF 元素将包含一个头和段落或子级别。

```

<! ELEMENT ALEAF (TOC1A,BT,(%content-elements;|BLEAF)* )>
<! ELEMENT BLEAF (TOC2B,BT,(%content-elements;|CLEAF)* )>
<! ELEMENT CLEAF (C,BT,(%content-elements;|DLEAF)* )>
<! ELEMENT DLEAF (D,BT,(%content-elements;|ELEAF)* )>
<! ELEMENT ELEAF (E,BT,(%content-elements;)* )>
<! ELEMENT TOC1A (#PCDATA)>
<! ELEMENT TOC2B (#PCDATA)>
<! ELEMENT C (#PCDATA)>
<! ELEMENT D (#PCDATA)>
<! ELEMENT E (#PCDATA)>

```

请注意我们并没有把 LEAF 元素作为内容元素实体的一部分。这就使得作者可以把 ELEAF 元素直接放在 ALEAF 元素的下面,违反了需要的层次性。相反,它们继续分离开,使设计者能看到结构。使用(% Content-element|XLEAF)结构允许 0 或多层内容:ALEAF 并不需要有 BLEAF 元素,但除非它被包含在 BLEAF 元素中,否则不能具有 CLEAF 元素(在本章的后面,我们将改变这种提法,使结构相对简单些)。

我们已经为章节提供了基本的结构,建立了基本的表单轮廓并能深入到 5 个级别(如果一个章节超越了 5 个级别,那要么认为它是章节的组织或给 DTD 增添了新的级别)。

现在该是回来给基本的文本段落增添内容的时候了。最容易管理的上下文信息种类是代码。目前的样式表使用了 4 种不同的代码样式,取决于代码出现的位置。在 XML 中,最麻烦的不是代码的位置而是代码的信息。样式表清除了表现方式并申请了额外的空间来插入到代码元素的前后。我们的代码元素,为了匹配样式珍,将命名为 CC。默认的 CC 元素 TYPE 属性将会是“CODE”,因为很少有书还有原文的输出,对于这些书来说,我们在 TYPE 中提供了一个“OUTPUT”选项,使设计者能格式化这两个类似的种类的过程有轻微的不同。一个 CC 元素也许能包含一外列表标题(LC)并且必须包括至少一个 CLINE(代码行)元素,使用多行就可以将必须的代码进行存储。

```

<! ELEMENT CC (LC?,CLINE+ )>
<! ATTLIST CC TYPE (CODE|OUTPUT) "CODE">
<! ELEMENT LC (#PCDATA)>
<! ELEMENT CLINE (#PCDATA)>

```

另外一组我们需要解释的样式是列表样式。当前,这种样式表包括标上号码的列表和独立样式的未标号码列表。这就呈现出许多选择。我们现在可以维护存在的样式表,建立一个元素来处理标号的和未标号的列表,也可以分解这些列表,使所有

的列表样式相对独立。最后,我们还可以把所有这些元素在一把“大伞”下组合起来。

虽然使用一把“大伞”的主意能产生清晰的代码(并且我们将在下一步中使用它),但对于现在来说,兼容性至关重要。对于样式表(特别是 XSL)的最新进展来说,似乎我们需要为独立的样式处理分隔开标号列表。这就是样式机制使用标号的正确途径,使得作者能不用重新标号就能移动单元。未标号列表仍然属于它自己的目录。

```
<! ELEMENT BL (BLINE+)>
<! ELEMENT NL (NLINE+)>
<! ELEMENT UL (ULINE+)>
<! ELEMENT BLINE (#PCDATA)>
<! ELEMENT NLINE (#PCDATA)>
<! ELEMENT ULINE (#PCDATA)>
```

下一种使用的样式是 GL,它是为词汇表文本准备的。GL 自身并不需要更多的处理,作者必须设置词汇表为粗体,因为词汇表是处理文档时最好处理的种类之一,而且在这个方面应该加强。新的 GL 元素将包含 GLITEM 和 GLDEFINITION,它们用来解释包含在 GLITEM 中的单词。

```
<! ELEMENT GL (GLITEM, GLDEFINITION)>
<! ELEMENT GLITEM (#PCDATA)>
<! ELEMENT GLDEFINITION %textual-elements;>
```

ICON 文本显得比较难于处理,ICON 文本定义图象在其左边浮动,但图象在样式表中并没有指定。虽然可以为每种文本建立一个独立的 NOTE,SHORTCUT, WARNING 和 TIP 元素,从而取代 ICON 元素,但这种方法在当前的练习中并不很协调。现在的任务是,我们将要创建 ICON 元素并包括这种类型的属性特性,使应用程序能在以后添加图象,继而区分它,并为其内容使用 BT 信息。

```
<! ELEMENT ICON (BT+)>
<! ATTLIST ICON
    TYPE (NOTE|SHORTCUT|WARNING|TIP) "NOTE">
```

表格是下一个挑战,一个有竞争力的 XML 并不是要即将面对的。表格经常和内容一样涉及到格式化的问题,并且它们在描述各种类型的内容时显得非常必要。样式提供了 TBC 来处理表格标题(应该说它在表格之上),TH 和 TBH 来处理栏目的标题头,TB 处理表格主体文本,以及用 SN 来处理底端的源注释。理想的作者应该能为每一个表格创建它们自己的微型 DTD,区分单元内容和元素。但不幸的是出版用的样式表并不提供使用这种扩展所需要的更多空间。现在,我们临时准备使用 HTML 样式的 TR 和样式表的 TB(处理起来非常类似 HTML 的 TD),和 HTML 不

一样,我们时刻需要 TB 元素适合其在 TR 元素中的位置。

```
<! ELEMENT TABLE (TBC* ,TH?,TR+ ,SN* )>
<! ELEMENT TBC (#PCDATA)>
<! ELEMENT TH (TBH+)>
<! ELEMENT TBH (#PCDATA)>
<! ELEMENT TR (TB+)>
<! ELEMENT TB %textual-elements;>
<! ELEMENT SN %textual-elements;>
```

注意

在 XML 中创建表格元素是一个有技巧的工作,仅声明这些元素并不意味着这个过程的结果,要使表格工作起来,需要把这些元素和样式连接起来,反映出它们的天然属性。

最后一个需要处理的元素是轮廓,轮廓图像并不是直接嵌在文件中的,只有有限数量的格式才是可以接受的轮廓。包含三部分:图像、轮廓编号、描述轮廓的标题。图像在解释该如何去得到轮廓图的属性控制下,有可能是空元素。这种文件定义使用和以前用 NOTATION 声明的属性相匹配的属性。轮廓编号将是标题的一部分。

```
<! NOTATION tiff SYSTEM "image /tiff">
<! NOTATION bmp SYSTEM "image /bmp">
<! NOTATION eps SYSTEM "image /eps">
<! ELEMENT FIGURE (FIGREF, FG)>
<! ELEMENT FIGREF EMPTY>
<! ATTLIST FIGREF
    SPC CDATA #REQUIRED
    TYPE NOTATION (tiff | bmp | eps) "tiff">
<! ELEMENT FG (FIGNUM, FDESC)>
<! ELEMENT FDESC %textual-elements;>
<! ELEMENT FIGNUM (#PCDATA)>
```

现在我们已经创建了所有这些必需的内容元素,该是升级内容元素参数实体的时候了。

```
<! ENTITY % content-elements "(BT | CC | BL | NL | UL | GL | ICON | TABLE | FIGURE) * ">
```

Combining all this into one gigantic DTD file gives us something we can work with.

把所有这些组合成一个巨大的 DTD 文件,看看能给我们实现些什么功能:

```

<! -Chapter DTD, version 1 ->
<! -Mandatory starting elements ->
<! ELEMENT CHAPTER (HEAD?,TOC0CT, INTRODUCTORY, ALEAF*)>
<! - Chapter Title ->
<! ELEMENT TOC0CT (#PCDATA)>
<! -Chapter Introduction ->
<! ELEMENT INTRODUCTORY (BT+)>
<! -Common text elements ->
<! ELEMENT CITATION (#PCDATA)>
<! ELEMENT USERACTION (#PCDATA)>
<! -Textual elements allows mixing w / regular data ->
<! ENTITY % textual-elements
"( #PCDATA| CITATION| USERACTION ) * ">
<! -BT - Body text. Used for each paragraph ->
<! ELEMENT BT %textual-elements;>
<! -Content elements allows multiple content types in
section contents->
<! ENTITY % content-elements "(BT | OC | BL | NL | UL | GL
| ICON | TABLE | FIGURE) * ">
<! -XLEAF elements behave like X-heads in previous->
<! -Note the structure for XLEAF elements - BLEAF
elements can only appear in ALEAF elements, CLEAF
elements can only appear in BLEAF elements, etc. ->
<! ELEMENT ALEAF (TOC1A,BT,(%content-elements;|BLEAF)*)>
<! ELEMENT BLEAF (TOC2B,BT,(%content-elements;|CLEAF)*)>
<! ELEMENT CLEAF (C,BT,(%content-elements;|DLEAF)*)>
<! ELEMENT DLEAF (D,BT,(%content-elements;|ELEAF)*)>
<! ELEMENT ELEAF (E,BT,(%content-elements;)*)>
<! -Headers for leaf sections ->
<! -A Head - appears in Table of Contents ->
<! ELEMENT TOC1A (#PCDATA)>
<! -B Head - appears in Table of Contents ->
<! ELEMENT TOC2B (#PCDATA)>
<! -C, D, E heads do not appear in Table of Contents->
<! ELEMENT C (#PCDATA)>
<! ELEMENT D (#PCDATA)>
<! ELEMENT E (#PCDATA)>
<! -Declarations for marking code listings ->
<! ELEMENT OC (LC?,CLINE+)>
<! -TYPE attribute differentiates code listings from text
output->
<! ATTLIST OC TYPE (CODE|OUTPUT) "CODE">
<! -LC is list caption ->

```

```

<! ELEMENT LC ( #PCDATA ) >
<! -CLINE represents a single line of code->
<! ELEMENT CLINE ( #PCDATA ) >
<! -Declarations for lists (bulleted, numbered, unordered
lists) ->
<! ELEMENT BL ( BLINE+ ) >
<! ELEMENT NL ( NLINE+ ) >
<! ELEMENT UL ( ULINE+ ) >
<! -List contents (bulleted, numbered, unordered lists)->
<! ELEMENT BLINE %textual-elements; >
<! ELEMENT NLINE %textual-elements; >
<! ELEMENT ULINE %textual-elements; >
<! -Glossary Declarations ->
<! ELEMENT GL ( GLITEM, GLDEFINITION ) >
<! -Word being defined->
<! ELEMENT GLITEM ( #PCDATA ) >
<! -Definition->
<! ELEMENT GLDEFINITION %textual-elements; >
<! -Icon /Note Declarations ->
<! ELEMENT ICON ( BT+ ) >
<! ATTLIST ICON
    TYPE ( NOTE|SHORTCUT|WARNING|TIP) "NOTE" >
<! -Table Declarations ->
<! ELEMENT TABLE ( TBC* , TH?, TR+ , SN* ) >
<! -Table Caption ->
<! ELEMENT TBC ( #PCDATA ) >
<! -Table Headers ->
<! ELEMENT TH ( TBH+ ) >
<! ELEMENT T3H ( #PCDATA ) >
<! -Table Rows ->
<! ELEMENT TR ( TB+ ) >
<! -Table Body (sim to HTML TD) ->
<! ELEMENT TB %textual-elements; >
<! -Table Source Note ->
<! ELEMENT SN %textual-elements; >
<! -Graphic Type Declarations ->
<! NOTATION tiff SYSTEM "viewer.exe" >
<! NOTATION bmp SYSTEM "viewer.exe" >
<! NOTATION eps SYSTEM "viewer.exe" >
<! -Figure Declarations ->
<! ELEMENT FIGURE ( FIGREF, FG ) >
<! ELEMENT FIGREF EMPTY >
<! ATTLIST FIGREF
    SRC CDATA      #REQUIRED

```



```

        TYPE NOTATION (tiff | bmp | eps) "tiff" >
<! -FG is figure description ->
<! ELEMENT FG (FIGNUM, FDESC) >
<! ELEMENT FDESC %textual-elements; >
<! ELEMENT FIGNUM ( #PCDATA) >

```

不幸的是,一个巨大的 DTD 需要一个巨大的文档来测试。如下的代码仅仅是测试代码,并不是一个真正的章节(尽管我可以通过加入这个代码来戏剧性地把这本书的厚度再增加一些),然而,这个代码应该能测试出大多数元素的自然属性。

```

<? xml version="1.0" encoding="UTF-8"? >
<! DOCTYPE CHAPTER SYSTEM "chapter.dtd" >
<CHAPTER>
<TOC0CT>How to code XML< /TOC0CT>
<INTRODUCTORY>
<BT>This is a chapter on how to code XML.< /BT>
<BT>This is another bit of intro info.< /BT>
< /INTRODUCTORY>
<ALEAF>
<TOC1A>Subhead A1< /TOC1A>
<BT>This is A1, from <CITATION>my
document< /CITATION>.< /BT>
<CC><LC>This is a listing< /LC>
<CLINE>I don't know what to do in this program! < /CLINE>
<CLINE>I don't know what to do in this program! < /CLINE>
< /CC>
<BT>You could have typed <USERACTION>exit< /USERACTION>
instead of running the parser.< /BT>
< /ALEAF>
<ALEAF>
<TOC1A>Subhead A2< /TOC1A>
<BT>This is A2. We'll start with a table, and move on to
a B leaf.< /BT>
<TABLE><TBC>This is a sample table< /TBC>
<TH><TBH>Column 1: Time< /TBH><TBH>Column 2:
Money< /TBH>< /TH>
<TR><TB>Never enough< /TB><TB>Can always use
more< /TB>< /TR>
<TR><TB>More than enough< /TB><TB>Had enough to begin
with< /TB>< /TR>
<SN>From <CITATION>The book of nonsense< /CITATION>< /SN>
< /TABLE>
<BLEAF>

```

```

<TOC2B>Subhead B1< /TOC2B>
<BT>A C leaf will follow, after the bulleted list.< /BT>
<BL>
<BLINE>This is the first item of a bulleted list< /BLINE>
<BLINE>This is the second item of a bulleted list< /BLINE>
< /BL>
<CLEAF>
<C>Subhead C1< /C>
<BT>This is a C leaf which contains a numbered list and a
figure.< /BT>
<NL>
<NLINEL>1. This is the first item.< /NLINEL>
<NLINEL>2. This is the second item.< /NLINEL>
< /NL>
<FIGURE><FIGREF SRC="image.tif" TYPE="tiff" />
<FG><FIGNUM>101, 12< /FIGNUM><FDESC> - The wild boar at
rest.< /FDESC>< /FG>< /FIGURE>
< /CLEAF>
<CLEAF>
<C>Subhead C2< /C>
<BT>This C leaf contains an unnumbered list and a
glossary item.< /BT>
<UL>
<ULINEL>This isn't in any order.< /ULINEL>
<ULINEL>Who needs order?< /ULINEL>
<ULINEL>Order just gets in the way.< /ULINEL>
< /UL>
<GL><GLITEM>Order< /GLITEM><GLDEFINITION> Something that
gets in the way frequently.< /GLDEFINITION>< /GL>
< /CLEAF>
< /BLEAF>
<ICON TYPE="NOTE"><BT>Hope you enjoyed the
chapter!< /BT>< /ICON>
< /ALEAF>
< /CHAPTER>

```

虽然你能仅仅看到顶部的一小段显示,但图 6-6 还是在 Internet Explorer 5.0 浏览器中正确地显示了 XML。

章节 DTD 的一个样式表

使用 XML 来开发一个章节 DTD 的样式表并不太难,即使出版者也许会抱怨说丢失了以前他们的桌面发布工具的良好控制,如 QuarkXPress。尽管 XML 支持的样

式技术能进行复杂的格式化,它仍然不能做打印设计者所做的工作。这种特殊样式表的格式对最后进行打印时所呈现出的方式没有什么帮助。它仅仅是一个巨大的表现方式并能帮助作者和编辑来看这本书。当完成编辑工作后,生产部门的职员从理论上是用和这个样式表类似的,并且看起来更象书本的样式表来替换,这样使得相同的元素放置在合适的位置上进行打印输出。

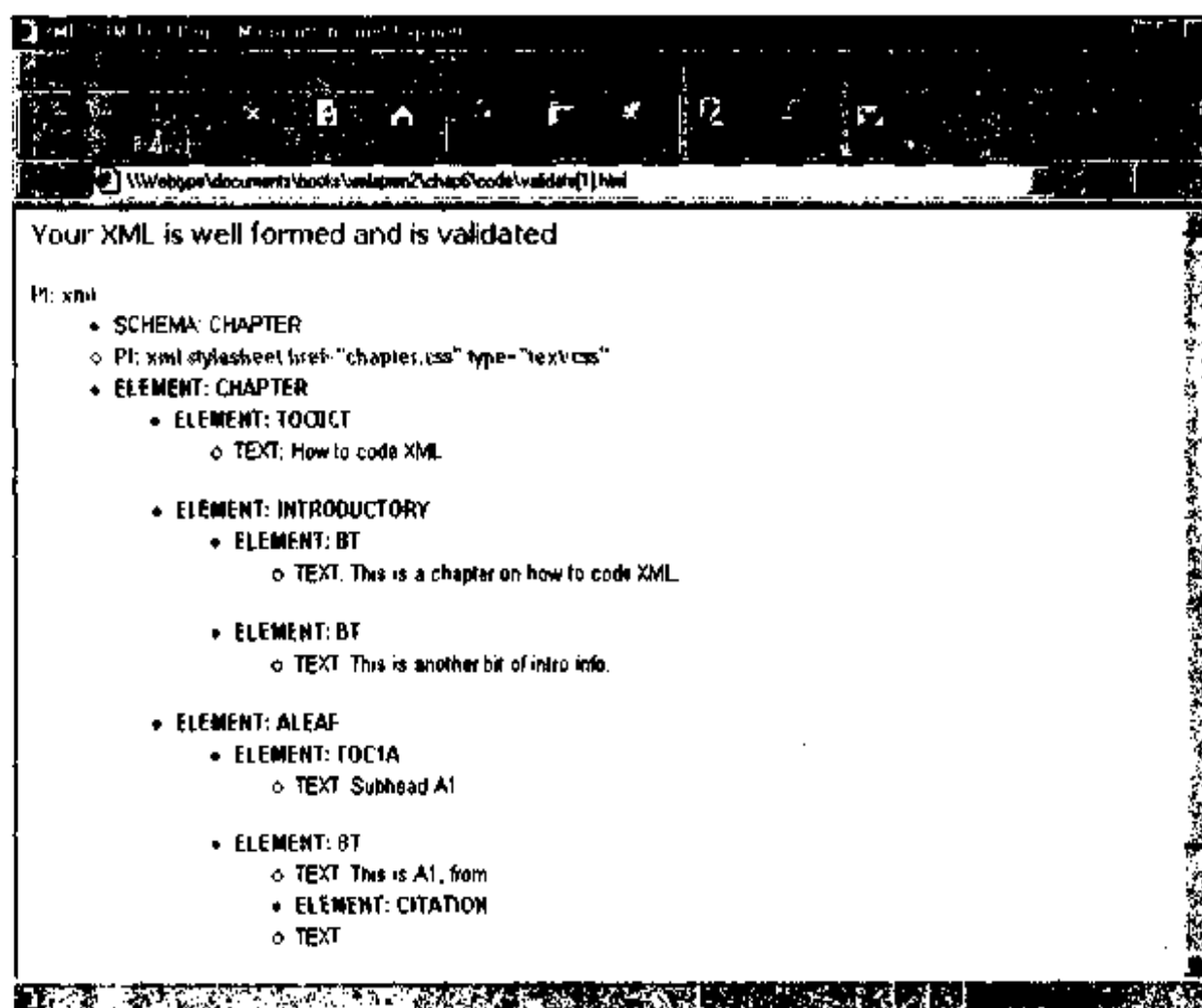


图 6-6 一个被检验过的(尽管未必有意义)章节文档

注意

用章节 DTD 创建的文档将会和其它的章节组合在一起,形成一个大结构的一部分,它将具有自己的格式规则。为了这个样式表,我们假定章节在它自己的页面上开始并终止,而且大文档的结构和它的任何样式都不会互相妨碍。

CHAPTER 元素的样式把自己标记为块元素,使自己的文本和其它环绕的材料相分开。它同时还设置了全文档的默认字体信息。

```
CHAPTER {
    display: block;
    font-family: serif;
    font-size: 12pt;
    font-weight: normal;
```

```
font-style:normal;  
}
```

我已经避免在样式表中使用真实的字体名,而使用普通的字体名如 serif 等。你能凭自己的爱好来替换成真实的字体名,但时刻需要记住的是,并不是所有的计算机都具备你所选择的字体。

TOCOCT 在打开标题时以 24 磅粗体的类型显示。注意底部留空的属性特征,它指定了标题下应该留出多少空间。

```
TOCOCT {  
display:block;  
font-size:24pt;  
font-weight:bold;  
padding-bottom:9pt;  
}
```

CITATION 和 USERACTION 元素将在文本中以斜体字显示单词,但它们并不建立独立的块。这些元素通过内处理(默认设置)来使段落不致于打碎而显得零乱。

```
CITATION {  
display:inline;  
font-style:italic;  
}  
USERACTION {  
display:inline;  
font-weight:bold;  
}
```

BT 是显示文本的基本元素,每个 BT 单元都可以粗略地认为等同于一个段落,所以 BT 元素是一外清晰的块。

```
BT {  
display:block;  
line-height:24pt;  
}
```

XLEAF 元素共享所有类似的格式,它们每个都以一个块元素开始,用预置的空间来与以前的元素相分隔,这一步可以在 CSS 的顶端预留设置中设定。

```
XLEAF {  
display:block;
```

```

|
D {
    display: block;
    font-size: 12pt;
    font-weight: bold;
    font-style: italic;
}

E {
    display: block;
    font-size: 14pt;
    font-style: italic;
}
```

代码元素和以前的章节样式表相比显得非常简单,这是由于其充分利用了 CSS 和 XML 的优点。Word 在处理 4 种样式时需要的额外的空间,可以通过对 CC 元素的定义中得到。因为 CC 元素把标题和代码完全封装起来了,它能提供必要的顶部和底端预留空间,同时不需要作者在真实代码中申请额外的样式。

```

CC {
    display: block;
    padding-top: 9pt;
    padding-bottom: 9pt;
    text-indent: .5in;
    font-family: monospace;
    line-height: 18pt;
}

LC {
    display: block;
    padding-bottom: 6pt;
    font-weight: bold;
}

CLINE {display: block;}
```

列表元素可以使用级联样式表的比较全面的一套列表样式来为列表项设置必需的样式。

```

BL {
    display: block;
    text-indent: 1in;
    line-height: 18pt;
```

```

        list-style-type: disc;
        list-style-position: outside;
    }

    NL {
        display: block;
        text-indent: 1in;
        line-height: 18pt;
        list-style-type: decimal;
        list-style-position: outside;
    }

    UL {
        display: block;
        line-height: 18pt;
        list-style-type: square;
        list-style-position: outside;
    }

```

那些列表的内容只需要将自身标示为一个列表项:

```

BLINE {display: list-item}
NLINE {display: list-item}
ULINE {display: list-item}

```

GL 元素有点复杂, 因为词汇表中用一行来描述后面的多行定义, GL 元素设定其文本方向为负方向 1 英寸, 从而产生了需要的悬挂方向。

```

GL {
    display: block;
    text-indent: -1in;
    padding-bottom: 9pt;
    line-height: 24pt;
}

GLITEM {
    display: inline;
    font-weight: bold;
}

GLDEFINITION {display: inline}
]

```

ICON 元素简单地把它文本以英寸为单位进行选位,从而找到合适的位置进行填空和间隔。NOTE、SHORTCUT、WARNING 以及 TIP 元素并不需要在这个意义上的样式,因为它们并没有任何内容。在最后的产物中,它们已经被替换成了图象。

```
ICON {  
    display: block;  
    text-indent: 1in;  
    padding-top: 9pt;  
    padding-bottom: 9pt;  
    line-height: 18pt;  
}
```

级联样式表一级标准提供了文档主流中文本对象的格式修饰,它依赖 HTML 的 TABLE 元素和它的子元素来建立表格。CSS 二级标准提供了全套显示属性来从任意元素中组装表格。表、内联表、行组表、头组表、脚注表、行表、列组表、列表、单元格表、标题表等显示属性值提供了比 HTML 模式更为灵活的构建工具。

```
TABLE {display: table;  
    padding-top: 9pt;  
    padding-bottom: 6pt;  
}  
  
TBC {  
    display: caption;  
    font-weight: bold;  
    font-style: italic;  
    padding-bottom: 9pt;  
}  
  
TH {  
    display: table-header-group;  
    padding-bottom: 9pt;  
}  
  
TBH {  
    display: table-cell;  
    font-weight: bold;  
}  
  
TR {  
    display: table-row;
```



```
        line-height: 18pt;
    }

    TB { display: table-cell;

    SN { display: table-footer-group;
        font-size: 10pt;
        padding-top: 6pt;
        padding-bottom: 6pt;
    }
```

FIGURE 元素总是在页面的正中位置。一般来说,FIGREF 元素在放置轮廓图像时并不需要样式,除非需要额外的留空。FIGREF 将从 FIGURE 元素处继承下来居中的特性,轮廓标题(FG 元素)也将被继承下来,而且 FIGNUM 将以粗体显示。

```
FIGURE {
    display: block;
    text-align: center;
}

FG {
    display: block;
    font-style: italic;
}

FIGNUM {
    display: inline;
    font-weight: bold;
}

FDESC {
    display: inline;
}
```

最后,我们将能够毫不费劲地使用这些样式来在浏览器中展现复杂的 XML 文档。改变级联样式表的样式将非常容易,使得在不同媒介上以不同格式展现文档也变得容易。虽然 CSS 并不提供所有的途径,但它已经提供了足够多的结构来建立一个良好形式的 XML 文档。更加复杂的文档将需要 XSL 的更加强大的工具,但在目前级别的格式下,CSS 已经足够了。

第二关:使 DTD 更加清晰

对先前的 DTD 的真正试验是作者和生产部门拿来使用它,随意拉伸其性质并找

出其缺点。在这个过程中找到许多明显和显著的缺点与薄弱之处,但这些方面为了和以前的基于 Word 的版本相兼容,也就放过去了。第二轮,为了测试所需要的政策性的兼容性,需要增强 DTD 使其更易扩充。定义了三个区域,首先是叶子结构 (ALEAF 和 BLEAF 等)、列表、图标许多元素都可以利用更加友好的名字。

注意

这些增强大多是技术方面的增强,并演示出 XML 的更加有趣的特征,但并没有用户生产量提高的担保。一些用户也许能找到更老的结构进行理解。DTD 开发者将会频繁地需要在多个一流的技术方案中或用户友好介面中进行折衷。能输入信息的自动工具将会被用来构建跨越沟痕的桥梁,但它们的可接受性还有待考验。

ALEAF /BLEAF /CLEAF 结构对于读者来说需要比普通叶子版本相对更多的元素,一个更普通的结构将使用不被标记为 A、B、C 但功能类似的元素。建立一个普通的叶子结构(包括许多递归——允许一个元素在它自身中包含另一个相同类型的元素)。递归在不同的环境中有许多不同的含义,许多将导致无休止的循环。XML 允许元素包含它们自己的子元素,但不允许实体指向文件,而那个文件又指向原文件类型的递归。对于大多数用途来说,下列的递归是 XML 中受到限制的实用型。

在下面的 DTD 中,一个 LEAF 元素必须包括一个 TITLE 元素,并且包含一个另外的 LEAF 元素和 PCDATA。

```
<! ELEMENT TEST (LEAF+)>
<! ELEMENT LEAF (HEADER,(LEAF|CONTENT)*)>
<! ELEMENT CONTENT (#PCDATA)>
<! ELEMENT HEADER (#PCDATA)>
```

上下查看这个文档可看出其结构上的层次。

```
<? xml version="1.0" encoding="UTF-8"? >
<! DOCTYPE TEST SYSTEM "recurs.dtd" >
<TEST>
<LEAF>
<HEADER>Top-layer Leaf 1</HEADER>
<CONTENT>This is a leaf. </CONTENT>
<LEAF>
<HEADER>Second-layer Leaf 1</HEADER>
<CONTENT>This is a leaf. </CONTENT>
</LEAF>
```

```

<LEAF>
<HEADER>Second-layer Leaf 2</HEADER>
<CONTENT>This is a leaf.</CONTENT>
<LEAF>
<HEADER>Third-layer Leaf 1</HEADER>
<CONTENT>This is a leaf.</CONTENT>
</LEAF>
</LEAF>
</LEAF>
<LEAF>
<HEADER>Top-layer Leaf 2</HEADER>
<CONTENT>This is a leaf.</CONTENT>
</LEAF>
</TEST>

```

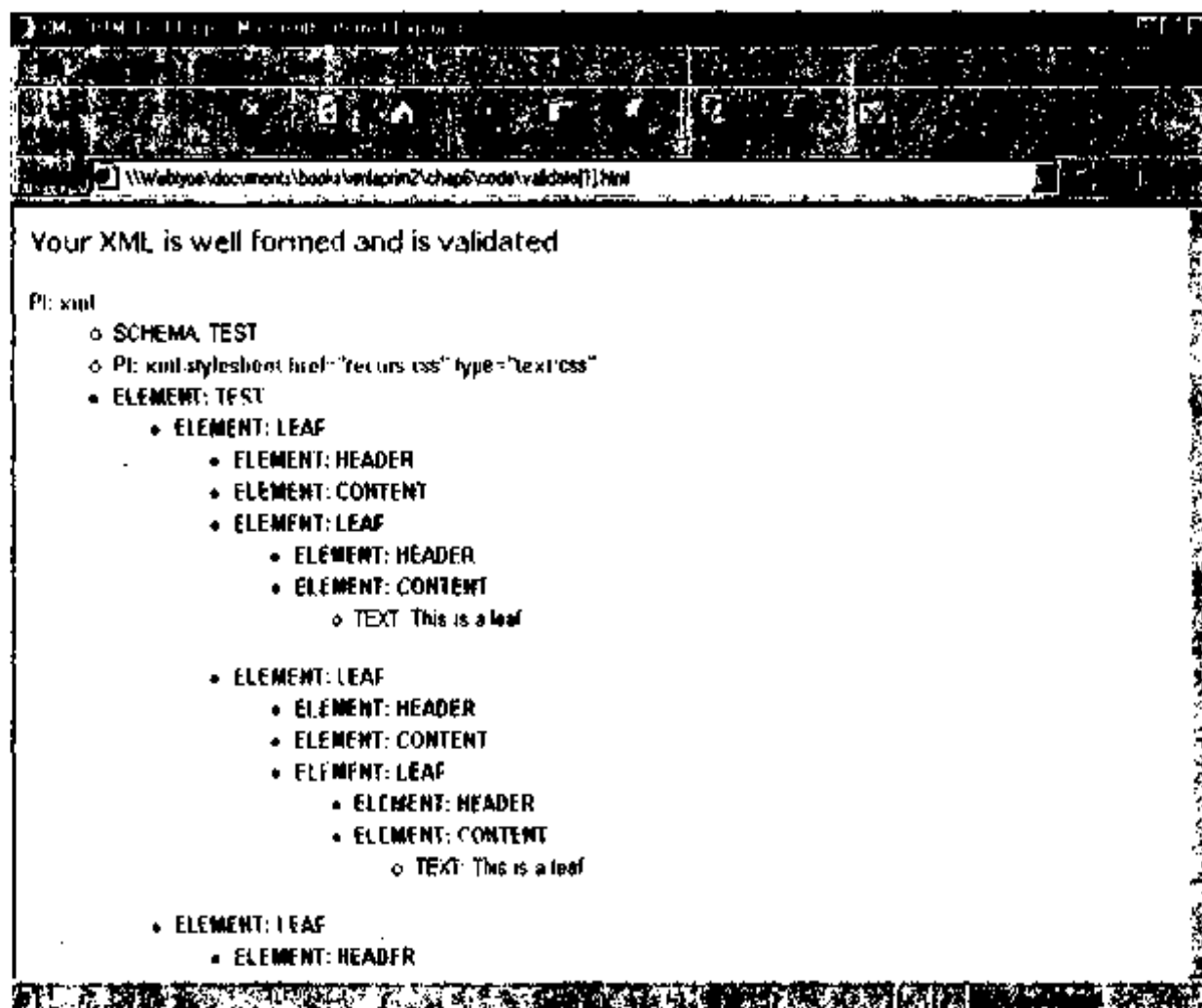


图 6-7 使用叶子结构提供一个更清晰和更抽象的嵌套内容

使用这种结构使我们能简化下列代码：

```

<! ELEMENT ALEAF (TOC1A,BT,(%content-elements;|BLEAF)*)>
<! ELEMENT BLEAF (TOC2B,BT,(%content-elements;|CLEAF)*)>
<! ELEMENT CLEAF (C,BT,(%content-elements;|DLEAF)*)>
<! ELEMENT DLEAF (D,BT,(%content-elements;|ELEAF)*)>
<! ELEMENT ELEAF (E,BT,(%content-elements;)*)>
<! ELEMENT TOC1A (#PCDATA)>

```

```

<! ELEMENT TOC2B ( #PCDATA ) >
<! ELEMENT C ( #PCDATA ) >
<! ELEMENT D ( #PCDATA ) >
<! ELEMENT E ( #PCDATA ) >
to this:
<! ELEMENT LEAF ( HEADER, BT, ( %content-elements; | LEAF ) * ) >
<! ELEMENT HEADER ( #PCDATA ) >

```

一个 LEAF 元素代替了所有的叶子结构,而且 HEADER 元素代替了 TOC1A, TOC2B,以及 C、D、E。一旦作者建立了新的叶子,它将被认为是文档新的子部分。它所显示出来的级别(如 A、B、C、D 或 E)完全由上下文来决定。

这个过程也带来了一些关于样式表的困难,需要在以前的样式表示中作一些小的改动。CSS 规范允许上下文选择器基于元素类型和所嵌套的样式来进行样式定义。在我们的示例中,因为所有的子叶子(在 AHEAD 之下)使用相同的留空值,新的 LEAF 元素的样式则显得非常简单。

```

LEAF {
    display: block;
    padding-top: 12pt;
}

LEAF LEAF {
    display: block;
    padding-top: 9pt;
}

```

这个样式定义表明 LEAF 元素将使用默认的 12 个点顶端留空,而且,所有的以其它 LEAF 元素封装的 LEAF 元素将有 9 个点来留顶端空间。标题更困难些,但遵循相同的规则。

```

LEAF HEADER {
    display: block;
    padding-bottom: 6pt;
    font-size: 18pt;
    font-weight: bold;
}

LEAF LEAF HEADER {

```

```

        display: block;
        padding-bottom: 6pt;
        font-size: 14pt;
        font-weight: bold;
    }

    LEAF LEAF LEAF HEADER {
        display: block;
        font-size: 14pt;
        font-weight: bold;
        font-style: italic;
    }

    LEAF LEAF LEAF LEAF HEADER {
        display: block;
        font-size: 12pt;
        font-weight: bold;
        font-style: italic;
    }

    LEAF LEAF LEAF LEAF LEAF HEADER {
        display: block;
        font-size: 14pt;
        font-style: italic;
    }

```

这些样式设定中的顶行是老的 TOC1A 标题——一种内嵌在 LEAF 元素层中的标题。第二行是 TOC2B, 以下类推。为了替换这种麻烦的命名方式, 标题样式采用了依据其在 LEAF 元素层次中的位置的方法来定义, 如图 6-8 所示。

列表也能得到显著的增强, 对于兼容性来说, 我们建立了三种元素: BL, NL 和 UL, 这些能用属性区分列表合并成一个简单的 LIST 元素。

```

<! ELEMENT LIST (LINE+)>
<! ATTLIST LINE
    TYPE CDATA (BULLETED|NUMBERED|UNORDERED)>

```

即使这是一个良好的愿望, 现在也必须依靠处理程序的能力来跟上其属性的发展。

级联样式表能对元素的上下文进行响应,但仅以回应 CLASS 属性(对于 TYPE 和 CLASS 的重命名将解决这个问题,但 CLASS 属性需要反应比列表类型更多的东西)。当 XSL 和其它一些上下文敏感的样式系统进入更加广泛的使用后,它将是否进行这个改变取决于使用工具的参照扩展能力。

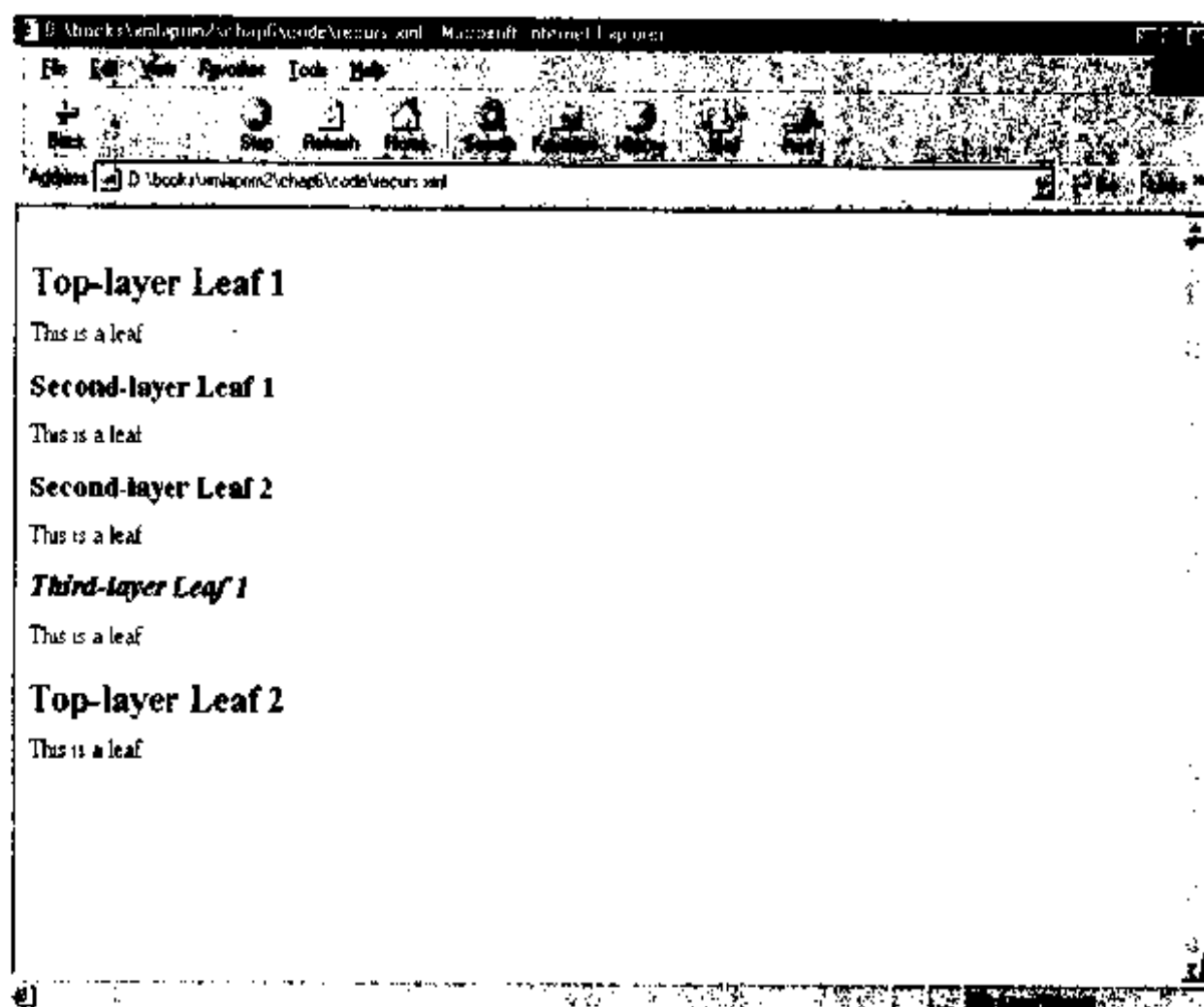


图 6-8 CSS 样式可以使用上下文来决定格式

目前,列表项拥有其所有的元素——NLINE、BLINE 和 ULINE。这些能被组合成 LINE 元素,使得处理软件能及时跟上这个进度。这可以通过并不困难的改变声明列表属性的类型来实现。声明是:

```
<! ELEMENT BL (LINE+) >
<! ELEMENT NL (LINE+) >
<! ELEMENT UL (LINE+) >
<! ELEMENT LINE %textual-elements; >
```

这将从一种列表类型转换成另一种类型。BL, NL, UL 元素保持它们以前的样式定义,而且行元素使用的样式声明与它的前辈非常类似。

```
LINE {display:list-item}
```

最后一个增强是用长的名字来命名样式。XML 与 SGML 不同,它并不用 SHORTREF 声明来提供全名的缩写版本,结果就是产生了更多的标记和更长的文件。而且,如果有人必须要阅读真实的文件,他们会更容易找到位置的所在。

建立包装文档

当为信息建立新的 XML 格式时,对于许多情况下的正确解决方案是从传统的格式向 XML 转化,不是所有的工作都会有一个良好的回报(我本人非常支持 XML,并坚定地相信把大文档分成多个小文档部分是一个很好的主意,但即使如此,并不是所有的文档都是那么有价值,而且并不是所有的项目都值得扔掉传统的遗留信息)。在许多情况下,使用 XML 包装遗留的信息是一个良好的开始,它并不对所有的信息都管用,有些信息并不容易以一系列字母和数字的形式来表现,而更适合单独放置。那样就留下了大量的信息用非 XML 的文本格式来进行处理。

最简单的包装信息就是普通文本文档。当所见即所得的编辑模式风行起来后,许多文档(包括 Internet 工程任务组的 RFC 文档)还继续使用纯普通文本进行散发,仅用空格的空白——制表符、隔线和平常的空间来进行格式化。老式打字机模式在 Web 上是最简单的格式。从这些文档向 XML 的转换非常困难,但实现第一级别的转换相对来说还可以实现:

```
<? xml version = "1.0" encoding = "UTF-8"? >
<? xml-stylesheet href = "wrap.css" type = "text /css"? >
<wrapper xml:space = "preserve">
```

```
Text with line breaks goes here.
```

```
< /wrapper>
```

这个样式表显得同样的简单:

```
wrapper {
    display: block;
    white-space: pre;
}
```


若文档中的空白不明显,样式表处理和 `xml:space` 属性的作用将会下降,形成一个极为简单的包装(因为空白不允许自动行间断,如同 HTML 的 PRE 元素,显示文档不包括中断行将出现问题)。

表 6-3 显示了空白在 CSS 中的所有可能取值。没有包装值是可用的——所有的情况下空白都被认为是明显有用的,断行必须明确地指示出来。

表 6-3 CSS2 中空白属性的取值

空白值	意义
normal	认为空白字符有意义,多个连续的空白字符将被压缩当成一个空格看待(不论 <code>xml:space</code> 属性是否被设置为“preserve”,它都是默认值)。这一行为与一般 HTML 浏览器的行为相同。
pre	所有空白字符都被看作有意义;文本行仅在新行字符(或产生的新行字符字符,如 HTML 中的 BR)处断开。不对多个空白字符进行压缩。
nowrap	空白有意义,但忽略新行字符。文本行仅在产生的新行字符(如 HTML 中的 BR)处断开。
inherit	元素将使用其父元素空白字符的行为。

包装整个文档对于文档管理来说并不是良好的长久之计,它丢失了许多 XML 的优势。但是它对于未来的标记来说能提供一个好的开始位置。允许处理不同级别期望的策略,即从完全没有到大量内容被加上标记,使得人们和组织在使用标准时有着不同的级别(它也会使人们很难超越最低级的共同水准,因此要小心)。

把遗留下来的数据使用完整无缺的 XML 标记文档记录下来仍然会是一个好主意。举例来说,天气观察标记格式(OMF,见站点 <http://zowie.metnet.navy.mil/~spawar/JMV-TNG/XML/OMF.html>)仍然使用纯文本来保留其遗留下来的报告数据使得新系统和旧系统之间能很好地进行通讯。在许多情况下,XML 可以表现一个文档的许多附加信息,比如作者、创建日期、版权等等,但是对于有些文档则显得不是很重要。在进行一个简单标准的迁移过程中,这种混合格式的方案显得非常有远见而且明智。

在下一章,我们将要从简单的基于打印和 Web 模式的文档转向更为复杂的文档,目的是研究那些商业交换用途的文档性质。我们将使用许多和 XML 相同的工具,但扩充了其能力,使之能创建机器可读的文档。

第 七 章

XML 用于商务

XML 最重要的优点就是它使人们和公司能够比用以前的格式更清晰、更完全地交换信息。这可以提高主页的平均水平和页面文档的产生效率,在商业领域,XML 可以给投资带来最大的回报。XML 允许在具有两大趋势的电子世界进行资本运做,这两大趋势是:日益增加的 Web 站点信息发布;日益增加的电子购物与结算。XML 通过使 Web 站点更容易搜索,而使它更加有用,同时通过提供数据交换的智能标准,减小了商务间通信的困难。

注意

在本章中最初的两个例子,就是探究用 XML 创建以商务为基础的 DTD 的过程。当你对创建自己的 DTD 前景感到令人激动时,你应该明白工业标准 DTD 是不存在的。创建你自己的 DTD 可以满足你的系统的特殊要求,但是它也会使你隔绝于其它工业。当相同的词表(特别是以 DTD 形式表达的)被一些人和工具使用时,XML 是最有用的。在标准化方面,商业应用大多数还处在危机中,因为搜索引擎和其它的应用可以依靠有相同意义的元素,而不管它的来源如何。在包含多个组织的案例中,兼容性是最重要的。在本章的后面,将讨论工业在这方面的一些进展情况,你对此将有一个大概了解。

谁将看我的 XML

以前章节的文档最终是供人阅读的。XML 使文档更易于机器控制,并提供了应用格式和存储数据的好方法,但是从长远来看,所有的文档都是由人来阅读。本章的应用利用了 XML 另外的优点。第一,XML 使开发者创建的文档可供人和机器阅读。对开发者来说,标记看起来像英语(或其它语言)中的标注,但是对计算机来说,它们只是简单的标签,指向计算机要处理的数据,这些数据整齐地存在嵌套结构中。第二,XML 为系统间的数据交换提供了更大的灵活性。XML 可以用来显示关系数

数据库的内容,也可以显示老的层次目录结构数据库的内容,甚至显示最新的面向对象数据库的内容。同时 XML 可以方便地显示文档信息、分组信息和简单的列表。最后,XML 为程序员提供结构,它使用递归结构以便于操作。这种结构在大多数编程工具中都会找到。编写 XML 解析器是不难的,特别是如果解析器的有效性和验证性能提前构成的话,编写起来就更简单了。

这些特点使 XML 比其它的交换格式具有更强大的功能,和更广泛的适应性。虽然现在有数千系统用于计算机之间的商业数据交换,但没有一个可以提供易于在结构上编程的灵活性。XML 显然也不能解决数据交换的所有问题,因为不是每一个程序都可以处理 XML 显示的数据结构,但是它仍向前迈了一大步。XML 毫无疑问是通用的工具,拿出足够的金钱和时间,将有更有效更好的方法来解决这些问题。

XML 把灵活性和结构融合在一起,适合用于搜索信息的应用程序组。当搜索引擎和代理服务器对信息进行分类和检索后,它们能在 DTD 和这些文档的标记中找到这些信息,充分利用这些功能需要标准化的标记。像<TODAYSSPECIALPRICE>和<SALETODAYONLY>,程序很难对这样的元素解释成某种意义(<PRICE>也可能夸大它的意义)。

搜索引擎在收集和分类无序状态的信息上特别困难。加上有意义的标准化的标识,可以使搜索引擎更好地找到相关信息。代理服务器通常在小规模上为特定用户搜寻有选择的信息。但是他们仍然坚持用同样的方法,就能确保达到计算机科学家很久以来一直追求的目标。自动搜索工具带来了额外的问题,让程序在 Web 上冲浪。如果你的站点从数据库上产生 XML 文档,那么通过自动工具装载,就会使你的系统过载甚至崩溃。因此,你应该为你的站点产生 robots.txt 文件,当代理服务器或搜索引擎访问一个站点时,它将检查 robots.txt 文件确定站点的那里可以进入,那里禁止进入。详细情况可访问站点:

<http://info.webcrawler.com/mak/projects/robots/norobots.html> 虽然 robots.txt 不是正式的标准,但是它已被搜索引擎的开发者广泛接受,并帮助阻止程序在你的站点上失去控制地运行,降低站点的效率,甚至可能使站点崩溃。

开发供计算机阅读的文档并不比开发供人阅读的文档困难。计算机有很强的可预测性,而且 XML 结构使它很容易产生可供多个应用程序处理的信息,甚至包括不同的应用程序所使用的相同信息。

一种更好的电子目录

虽然在前几章中的 HTML 目录可以用来在页面上显示信息,但是把它转换成

XML 后,可以增强它的搜索能力。Joe 目录的主要竞争对手 Jane 目录正在进行这一艰难的过程。创建一个全新的目录格式,需要 Jane 花费许多精力,但是这将产生一个更自动化的、更便于管理的目录。

只要定单通过电话下达,Jane 就可以轻松地用 Web 目录重新创建页面目录。因为 800 线的费用在增加,Jane 开始考虑是否改变她的系统,允许 Web 用户通过较便宜的 Internet 连接下达定单。同时,Jane 把下一级组的桌上型电脑的发布文件目录,移到新的数据库系统,这一系统可以保留她的项目和所有相关数据——甚至图片。由于 Jane 没有做大量的广告,Jane 喜欢用能使 Jane 的目录容易理解的自动工具,而使用户容易地找到他想要的项目。XML 听起来好象要给 Jane 一个推力,并且它也很适合 Jane 所安装的数据库系统。Jane 所面临的挑战是,创建可以显示其数据库目录应用程序,这对客户是有吸引力的,与此同时,减少了对其定单入口处的压力。

我们不是从 Web 上正在显示的目录开始工作,而是通过检查 Jane 的方法,从组装目录和处理定单开始工作的。当 Jane 的计算机接收一个定单时(现假设通过电话),它就会用信用卡来处理付费,并打印出商场的货物清单。商场的工人找到这一清单后,就会马上送货。定单入口处要查看 Web 应用程序流入到他们计算机的一系列简单信息:付费信息、送货信息、订购的产品清单和数量。用现有的 Web 工具,这是一组十分容易发送的数据。最后在客户这边,Jane 需要应用程序给客户显示完整的发票及送货费用。这样做的部分动机是让用户的 Web 浏览器尽量做更多的工作,节省了服务器处理这些信息的开销,而客户端的计算机也能很容易地处理这些信息,这样节省了 Jane 的昂贵的带宽和处理信息的机器开销。即使这样,今天的 XML 也没有达到这一目标,几种其它的标准正在开发中,不久的将来会使它变得更加容易,我们将在本章的后面看到。

在这一点上,XML 对 Web 应用程序的开发存在一些缺点。为 JavaScript 代码读取 XML 信息要加上必要的链接,会使 June 产生的简捷结构变得复杂起来。DTD 必须对在某一级尚未变得复杂的脚本提供支持,且不使它的结构变的复杂。幸运地是,June 已经计划让其应用程序采用这种框架结构,这将使其能把另外一个框架结构中的代码安全地放到 XML 文档外。所有的 XML 文档要做的是,要把有关用户想要的项目信息,送到另一个框架的脚本中,它将把信息存在定单栏中,并且处理 Jane 需要的所有销售的下部结构。事实上,Jane 的计算机将从数据库中产生静态 XML,通过允许一些重复信息在 XML 标记中出现,减小了困难。Jane 目录是由相关产品松散地组织到一起,而形成页面。组是由创建页面的队决定的。电子目录应该保留这些分组,方便客户在两个版本间前后翻页查讯页面。Jane 需要一套能美观地显示目录数据的元素,并能适应脚本和文档对象模型的改善,同时仍能满足当前的需要。

DTD 的顶级是组,一个组的所有成员都在同一个文档中。

```
<! ELEMENT GROUP (GROUPNAME, ITEM+, LEGALNOTICE) >
<! ATTLIST GROUP
    GROUPLINK CDATA #IMPLIED>
```

GROUPLINK 的属性包含一个关键值,它能使解析器方便地链接到原始数据库记录,以便故障排除或收集其它信息。GROUP 元素用 GROUPNAME 元素作为开头,它是页面的有效标题,并以 LEGALNOTICE 元素结尾,它包含有关排版印刷和价格的常用通告。搜索引擎和代理服务器不会把 LEGALNOTICE 当作元素寻找,但是它使格式化打印更加容易。

```
<! ELEMENT GROUPNAME (#PCDATA) >
<! ELEMENT LEGALNOTICE (#PCDATA) >
```

在标题下面是目录 ITEM 元素。ITEM 元素包含各种其它的元素,如 ITEM 子元素,它代表了附属于父 ITEM 的产品。像 GROUP 一样,ITEM 元素也有一个连接到它的数据库入口的属性。

```
<! ELEMENT ITEM (PRODUCTNAME, DESCRIPTION?, PRICING, ITEM*) >
<! ATTLIST ITEM
    ITEMLINK CDATA #IMPLIED>
```

PRODUCTNAME 元素像 GROUPNAME 元素一样,只是一个标题。由于元素受到不同的格式化,因此要通过脚本赋予不同的地址,两个元素要分开创建。

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

DESCRIPTION 元素必须能够处理各种内容(记住,DESCRIPTION 元素是可选的,它不适用于子项目)。目录入口一般不包括斜体和粗体文本,但是有时也可以包括,且大多数用来做书的标题。大多数产品的描述只有一段,但是偶尔,一个入口可以有几个段。许多段甚至还包含图片。作为结果,DESCRIPTION 元素必须包含 PARAGRAAPH 元素,PARAGRAAPH 元素包含各种不同的其它元素分类。IMF 元素是从 HTML 版本中模仿的。

```
<! ELEMENT DESCRIPTION (PARAGRAPH | IMG)* >
<! ELEMENT IMG EMPTY >
<! ATTLIST IMG
    SRC CDATA #REQUIRED
    HEIGHT CDATA #REQUIRED
    WIDTH CDATA #REQUIRED>
<! ELEMENT PARAGRAPH (#PCDATA | CITATION | EMPHASIS | HIGHLIGHT)* >
<! ELEMENT CITATION (#PCDATA) >
<! ELEMENT EMPHASIS (#PCDATA) >
```

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

这一套简单的标记能够使 Jane 和它的职员为目录中的所有项目创建描述。虽然对设计人员来说,这很令人激动,但它对以 PRICING 元素为内容的定单处理没有任何影响。PRICING 元素包含所有的信息,如价格、船舶吨数、交货、有效性和保证书信息,以及把货物放到小车上的执行按钮。

```
<! ELEMENT PRICING (PRODNUM, MARKER?, PRICE, MARKER?, SHIPPING, MARKER?, DELIV-  
ERY, MARKER?, AVAIL, MARKER?, WARRANTY, PURCHASE) >
```

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

```
<! ATTLIST PRODNUM
```

```
    ID    id    #REQUIRED >
```

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

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

```
<! ATTLIST PRICE
```

```
    ID    id    #REQUIRED >
```

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

```
<! ATTLIST SHIPPING
```

```
    ID    id    #REQUIRED >
```

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

```
<! ATTLIST DELIVERY
```

```
    ID    id    #REQUIRED >
```

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

```
<! ATTLIST AVAIL
```

```
    ID    id    #REQUIRED >
```

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

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

```
<! ATTLIST PURCHASE
```

```
    onclick    CDATA    #REQUIRED >
```

Jane 有理由相信,它能把商场的现货项目列表,并用 AVAIL 元素表示即将到来的货物,进行预先订购。公司的货物经常变化,需要创建动态页面,用更准确的标记反映它当前的货物情况。如 ONHAND 表示现有货物的编号,ONORDER 表示货物什么时候到达。然而要记住,你不能把你的所有货物清单显示给你的竞争对手。

PRICING 元素是定单所需处理部分的清单,它包括所有需要创建的非客户的包装清单信息。可选择的 MARKER 元素可以分散描述诸如价格之类的东西,避免 PRICE 元素出现混乱(当 XSL 有效地出现后,就不再需要它了)。MARKER 信息只是格式化,且必须和其它元素的内容分开。

现在,需要为把信息送到商店小车的所有元素分配 ID 值。在这个例子中, ID 值将用包括产品编号的命名协议来创建(为了避免在同一页面上复制 ID 值)。这个 ID

值可以使脚本找到元素,并提取处理内容。PURCHASE 元素看起来像以下形式:

```
<PURCHASE onclick = " javascript:parent.cart.add('12323')">
Add to Cart< /PURCHASE>
```

使用 HTML4.0 处理事件的属性,该脚本将对单击做出响应,用 NAME 属性“cart”调用 FRAME 的添加方法。

注意

EVENT 模型由 Internet Explorer 4.0 和 5.0 支持,并且将来的 Netscape 浏览器也支持它。

综合所有的信息产生了 DTD,而 DTD 封装了 Jane 目录所需要显示给客户的信息,并把这些数据返回到定单处。

```
<! ELEMENT GROUP (GROUPNAME, ITEM+ ,LEGALNOTICE)>
<! ATTLIST GROUP
    GROUPLINK CDATA #IMPLIED>
<! ELEMENT GROUPNAME (#PCDATA)>
<! ELEMENT LEGALNOTICE (#PCDATA)>
<! ELEMENT ITEM (PRODUCTNAME, DESCRIPTION?,PRICING, ITEM* )>
<! ATTLIST ITEM
    ITEMLINK CDATA #IMPLIED>
<! ELEMENT PRODUCTNAME (#PCDATA)>
<! ELEMENT DESCRIPTION (PARAGRAPH | IMG)*>
<! ELEMENT IMG EMPTY>
<! ATTLIST IMG
    SRC CDATA #REQUIRED
    HEIGHT CDATA #REQUIRED
    WIDTH CDATA #REQUIRED>
<! ELEMENT PARAGRAPH (#PCDATA | CITATION | EMPHASIS | HIGHLIGHT)*>
<! ELEMENT CITATION (#PCDATA)>
<! ELEMENT EMPHASIS (#PCDATA)>
<! ELEMENT HIGHLIGHT (#PCDATA)>
<! ELEMENT PRICING (PRODNUM,MARKER?,PRICE, MARKER?, SHIPPING, MARKER?, DELIV-
ERY, MARKER?, AVAIL, MARKER?, WARRANTY, PURCHASE)>
<! ELEMENT PRODNUM (#PCDATA)>
<! ATTLIST PRODNUM
    ID ID #REQUIRED>
<! ELEMENT MARKER(#PCDATA)>
```



```

<! ELEMENT PRICE( #PCDATA)>
<! ATTLIST PRICE
    ID      ID      #REQUIRED>
<! ELEMENT SHIPPING ( #PCDATA)>
<! ATTLIST SHIPPING
    ID      ID      #REQUIRED>
<! ELEMENT DELIVERY ( #PCDATA)>
<! ATTLIST DELIVERY
    ID      ID      #REQUIRED>
<! ELEMENT AVAIL ( #PCDATA)>
<! ATTLIST AVAIL
    ID      ID      #REQUIRED>
<! ELEMENT WARRANTY ( #PCDATA)>
<! ELEMENT PURCHASE ( #PCDATA)>
<! ATTLIST PURCHASE
    onclick CDATA    #REQUIRED>

```

这个处理的最后结果是 Jane 目录可以使用的结构,它既可以把信息传递给用户,又可以处理这个信息,把定单送回到订购系统(这些定单可以用 XML 格式发送,但是那将是下一个系统的主题。前述的 DTD 只包含客户端处理器需要创建一个完整的包装清单和定购商品的发票信息)。

由 Jane 的数据库产生的目录的一个样本页面,看起来像以下形式:

```

<? xml version="1.0" standalone="no" encoding="UTF-8"? >
<! DOCTYPE GROUP SYSTEM "janes.dtd">
<GROUP GROUPLINK="23AA34FAB1">
<GROUPNAME>Pocket Calculating Devices< /GROUPNAME>
<ITEM>
<PRODUCTNAME>Mortgage Calculator< /PRODUCTNAME>
</DESCRIPTIONS> </PARAGRAPHS> Ever want to know precisely how much of your business is done?

```

```
<AVAIL ID="I1024A">Yes< /AVAIL>
<MARKER>Warranty: < /MARKER>
<WARRANTY>30 years< /WARRANTY>
<PURCHASE onclick="javascript:parent.cart.add('1024')">Add to Cart< /PURCHASE>
< /PRICING>
<ITEM>
<PRODUCTNAME>Carrying Case< /PRODUCTNAME>
<PRICING><PRODNUM ID="I1028">1028< /PRODNUM>
<MARKER>Price: $ < /MARKER>
<PRICE ID="I1028P">2.00< /PRICE>
<MARKER>Shipping: $ < /MARKER>
<SHIPPING ID="I1028S">1.00< /SHIPPING>
<MARKER>Delivery: < /MARKER>
<DELIVERY ID="I1028D">Overnight< /DELIVERY>
<MARKER>In Stock?: < /MARKER>
<AVAIL ID="I1028A">Yes< /AVAIL>
<MARKER>Warranty: < /MARKER>
<WARRANTY>2 years< /WARRANTY>
<PURCHASE onclick="javascript:parent.cart.add('1024')">Add to Cart< /PURCHASE>
< /PRICING>
< /ITEM>
< /ITEM>
<LEGALNOTICE>Jane's Catalog is not responsible for typographical errors. Prices and availability may change at any time. < /LEGALNOTICE>
< /GROUP>
```

正像你看到的那样,嵌入 ITEM 元素很简单,不需要我们在上一章使用过的 SUBITEM 元素,SUBITEM 元素的作用是使结构更加一致。ID 元素可以把有用的信息复制到其它地方(产品编号),但它只是暂时的,下一节我们将做出解释。

把脚本加到 HTML 中

SGML 开发的大部分软件是促使编程逻辑和所用到的数据之间产生分离。SGML 确实在程序可以解释的结构格式中,做出了保存数据的杰出工作,但是它不能为“活动的”内容提供机理去处理,而这类内容正在 Web 上迅速发展。在这一领域 XML 的 SGML 继承性是个问题。即使对普通的 HTML 任务(诸如创建脚本标记,以保留处理文档内容所需的代码),在 XML 中也经常遇到困难。JavaScript(新的标准化 ECMASCRIPT)和 VBSCRIPT 二者都使用符号 &、< 和 > 作为编程目的。对于这两种脚本语言,< 表示小于,而不是开始一个新的标记。关闭 HTML 4.0 DTD 检查,显示 HTML 使用 SGML'S CDATA 类型作为元素内容,它允许这些字符出现:

```

<! ELEMENT SCRIPT - - CDATA - script statements - >
<! ATTLIST SCRIPT
    type      CDATA      # IMPLIED - Internet content
type for script language-
    language CDATA      # IMPLIED - predefined script
language name-
    src       %URL;      # IMPLIED - URL for an external
script-
    >

```

由于 SGML 执行 XML 出现了各种问题,因此 XML 不能为元素提供 CDATA 内容模型,只能提供 #PCDATA。当解析器碰到使用 &、<、或 > 的代码时,它就会崩溃。开发者必须创建一个使用 PCDATA 的 SCRIPT 元素,然后把 PCDATA 段转换成 CDATA 段。方法是用手工对段作标记: <![CDATA [... script...]]>。SCRIPT 元素表示的看起来类似前面的 HTML DTD:

```

<! ELEMENT SCRIPT ( #PCDATA ) * >
<! ATTLIST SCRIPT
    type      CDATA      # IMPLIED
    language  CDATA      # IMPLIED
    src       CDATA      # IMPLIED>

```

然后对使用脚本的所有文档,开发者必须把下列形式:

```

<SCRIPT LANGUAGE="JavaScript">
var x,y; x=1; y=2;
if x<y {alert ("X is less than Y!")}
</SCRIPT>

```

转换成:

```

<SCRIPT LANGUAGE="JavaScript"><![CDATA[
var x,y; x=1; y=2;
if x<y {alert ("X is less than Y!")}
]]></SCRIPT>

```

这就是 HTML Activity's Voyager 为在 XML 中重新创建 HTML 所提出的解决方法。开发者可以求助的另一个解决方法是 SRC 属性,它能够让页面装入一个包含脚本的独立文件中。所有违规的代码可以存在一个独立的文件中,完全避开了 XML 解析器。

Scriptlets 是来自 Microsoft 公司的最新动态 HTML 工具——也可以为脚本提供

隐含的空间。虽然目前,它们只能在 Internet Explorer 4.0 或更高版本上工作。Scriptlets 使用 OBJECT 标记输入另一个可以用来做接口的文件,这对处理商店手推车和现金出纳机所用的电子目录是十分有用的。但是它能否流行,还需等待实践证明。Scriptlet OBJECT 元素看起来像以下形式:

```
<OBJECT width=100 height=300 TYPE="text /x-scriptlet"  
DATA="JSSCRIPTLET.HTM">< /OBJECT>
```

其它用于把脚本链接到元素的工具,使用类似样式表的语法,并且都以 Netscape 和 Microsoft 的建议为基础,这些工具也正在被 W3C 考虑中。W3C 文档对象模型 (DOM) 能够使脚本通过文档结构和 ID 访问元素。遗憾的是,完全可实现的模型仍在开发之中,虽然许多 XML 解析器支持 Core DOM。

商务交易

Internet 上的电子商务才刚刚开始,但是在 Internet 上的零售业就面对安全和消费怀疑论的威胁。许多 Internet 的企业家不是开发大系统来处理信用卡和客户查询,而是转向更安全的商务对商务交易系统。经常涉及交易的团体,已经互相熟悉,甚至互相信任了,避免了和 Internet 商务有联系的匿名问题的出现。帐号系统已经提供了被客户熟悉的信用消费,送货条款也已经建立。正像我们在下一节将看到的那样,并不是所有的交易都是财务上的,许多系统用类似的连结,安全地共享私人的信息。

迄今为止,电子数据交换 (EDI) 系统趋于使用数据库的结构:放在表格中用于处理的固定长度字段或者可变字段。传送这样的信息已经取的很大进展,公司过去用邮寄磁带的方法,定期互相传送数据,但是现在他们经常通过数据网络来交换数据,而信息的形式仍然没有大的改变。XML 提供的商务比当前系统提供的商务拥有更大的灵活性,以及在需要时为附加的数据结构建立简单标准的机会。

在这一节中显示的例子只提供建立一个完善的商务交易系统的基本概要。这里提出的标准是根据标准团体和工业组织的标准而来的。

EDI 用做商业交易,在过去的二十多年里,正在广泛推广,它所创建的结构仍能对我们的 XML 例子提供很多东西。XML 不是电子购物的所有方面的解决方案。商业间可以互相信任到相互送货的程度,但是定单必须放在安全的通道上。这可以简单的就像使用组钥匙加密工具一样,然后把加密的 XML 文档通过电子信箱发送出去;复杂的情况就像通过建立私人网络把它发送出去一样。XML 文档也可以放在磁带上,由私人送快信的人送给那些自己的财务系统与网络相连的公司。

在通道建立后,公司就可以开始考虑他们的订单格式。所有的定单仍需要送货和清单信息,以及一旦计算机出现故障,需要与人联系的信息。数据也是很重要的,给接收者一些有关什么时候订单建立,订单到达的概念。订单的优先级在某些情况下是有用的,虽然对订单的优先级是独立的,它也有整体优先级和部分优先级。订购货物的清单将以货物总数和付费总额结束。结论是很重要的,它用来确定货物在处理过程中有没有丢失。XML 开发者可能检查较老的数据交换形式的上部,以确保他们没有留下老的结构提供的关键部分。

我们最初的 DTD 用于 XML 交易,它为定单提供一个内核,是专用的文摘。在这一级,它不关心订购的货物是什么物品——苹果、拖拉机和手电筒都是公司间传送的物品。第二个 DTD 定义该项目,它强调有关商品的问题。

我们现在开始定义 ORDER 元素,它包括整个文档:

```
<! ELEMENT ORDER (BILLTO, SHIPTO, CONTACT, PRIORITY, ITEM+ , TOTALS)>
```

BILLTO 元素和 SHIPTO 元素有类似的内容:

```
<! ELEMENT BILLTO (REFERENCE | FULLADDRESS)>
<! ELEMENT SHIPTO ((REFERENCE | FULLADDRESS), SHIPVIA)>
<! ELEMENT SHIPVIA (REFERENCE | FULLADDRESS)>
<! ELEMENT REFERENCE (#PCDATA)>
<! ELEMENT FULLADDRESS (COMPANY, ADDRESSLINE + , CITY, STATE, POSTALCODE,
COUNTRY, CONTACT, PHONE, FAX?)>
<! ELEMENT COMPANY (#PCDATA)>
<! ELEMENT ADDRESSLINE (#PCDATA)>
<! ELEMENT CITY (#PCDATA)>
<! ELEMENT STATE (#PCDATA)>
<! ELEMENT POSTALCODE (#PCDATA)>
<! ELEMENT COUNTRY (#PCDATA)>
<! ELEMENT CONTACT (#PCDATA | REFERENCE) * >
<! ELEMENT PHONE (#PCDATA)>
<! ELEMENT FAX (#PCDATA)>
<! ELEMENT PRIORITY (#PCDATA)>
```

大多数信息基本上是文本的,关键的 REFERENCE 元素有可能用做大多数交易,通过使用 REFERENCE,订购公司宣布在接收者系统已存有记录。接受数据的处理应用程序将立刻使用 REFERENCE 给定单系统传送定单——关系已经存在(如果它是一个坏的关系,公司定单将不付费,或者在系统上不存在,定单系统仍然拒绝这个定单)。以全地址方式到达的定单还需要进一步认证,联系将被调用,信用检查也将进行,如果有必要的话,新的购买者将进入定单系统并为将来使用给出一个自己的 REFERENCE 信息。

TOTALS 元素包括用来检查定单的摘要。

```
<! ELEMENT TOTALS (TOTALITEMS, TOTALQUANTITY, TOTALCOST) >
<! -NOTE TOTALS ARE FOR CHECKING DATA ONLY AND DO NOT REFLECT FINAL COSTS OR
QUANTITIES ->
<! ELEMENT TOTALITEMS ( #PCDATA) >
<! ELEMENT TOTALQUANTITY ( #PCDATA) >
<! ELEMENT TOTALCOST ( #PCDATA) >
```

基本的内核(shell)对各类商务是有用的,人们可能想自定义内核,以在某种程度上反映他们的要求。内核最重要的特点是省去一些东西。由于 ITEM 元素从来不定义,DTD 不是完全的,需要一个实际上相匹配的 DTD 来处理定单。

对于这个例子,我们将用于出版业,它是我遇到的使用电子定单以及所有的收益和费用的唯一行业(在无数个订购成千上万种书的定单中,寻找丢失的项目,是我再也不想重复的经历)。对这个相对简单领域的 ITEM 定义,将包括三块:Book 将包含标题信息,QUANTITY 用来说明他们想要多少本书;PRIORITY 使客户能要求更高级别的优先权来处理某些标题;如果其他的信息正确的话,EXTENDED COST 就提供给购买者要付出的总花销。在这个案例中,EXTENDED COST 的作用相当于校验求和,以确保购买者的信息无误(如果他们把价格弄错,他们仍将按正确的价格付费。)

```
<! ELEMENT ITEM (BOOK, QUANTITY, PRIORITY?, DISCOUNT,
EXTENDED COST) >
<! ELEMENT DISCOUNT ( #PCDATA) >
<! ELEMENT EXTENDED COST ( #PCDATA) >
```

早期出版业的标准是对它的产品做一套编号:国际标准出版编号(International Standard Book Number ISBN)。对于本书来说,其 ISBN 是 0-7645-3310-X。其中,第一位数字表示这本书是用英语写的,下面的四位数字表示这本书是由 IDG 出版社发行的,再下面的四位数字代表 IDG 出版社唯一指定的标题,最后一位数字(它可以是 0 到 9 的数字,或者是 X)是校验数。XML 解析器不处理 ISBN,确保校验数正确那是处理应用程序的责任。ISBN 是唯一认定书和与书有关的产品。虽然包装的内容可能包括它们自己的 ISBN 的其它项目,每一个 ISBN 在技术上只允许参考一个项目以作为销售单位(实际上,它是条形码标准,便于存储定单和买书)。至少在理论上,我们的 DTD 甚至不需要标题和价格信息,因为那些信息都要和 ISBN 相连结。在现实中标题和价格经常变化,且客户不会总注意到。加入标题和价格信息也提供了额外的保险,使定单能被正确处理,并对只知道过时信息的客户发出通知,而不会停止处理定单,也不要人工干预。

BOOK 元素包括 ISBN、标题和价格。对于大多数以前的例子,我们都避免用任何和 HTML 有冲突的元素名。虽然在浏览器的任何地方,都不会有这些定单。这个模型最后可扩展到零售,因此重新命名 TITLE 元素可能是最好的。

```
<! ELEMENT BOOK (ISBN, BOOKTITLE, PRICE) >
<! ELEMENT ISBN (#PCDATA) >
<! ELEMENT BOOKTITLE (#PCDATA) >
<! ELEMENT PRICE (#PCDATA) >
```

每行项目的另一个重要部分是数量。发出定单的公司很可能需要的书不是一本。一些批发商想要以整体方式接收图书,不打开装书的箱子。由于书以各种形状和尺寸运到,箱子中所装书的数量,因书的不同而不同。当印刷厂用不同尺寸的纸来装订印刷时,同样标题的书甚至可能用不同数量的箱子运到。在任何情况下,我们必须为客户提供选择,确定他们是否要整箱以及箱子尺寸的大小和如何调整箱子种类的大概说明。大多数小客户不关心整箱,因为他们从商店接收装有各种图书的,重新包装过的箱子。

```
<! ELEMENT QUANTITY (NUMBER, CARTON?) >
<! ATTLIST QUANTITY
    SHIPCQ (NO | ROUNDUP | ROUNDDOWN | ROUNDCLOSEST) "NO" >
<! ELEMENT NUMBER (#PCDATA) >
<! ELEMENT CARTON (#PCDATA) >
```

既然我们有两种完善的 DTDs,那我们就可以开始创建一些定单了。第一种完善的 DTD 提供我们使用的内核环境是:

```
<! ELEMENT ORDER (BILLTO, SHIPTO, CONTACT, PRIORITY, ITEM+, TOTALS) >
<! ELEMENT BILLTO (REFERENCE | FULLADDRESS) >
<! ELEMENT SHIPTO ((REFERENCE | FULLADDRESS), SHIPVIA) >
<! ELEMENT SHIPVIA (REFERENCE | FULLADDRESS) >
<! ELEMENT REFERENCE (#PCDATA) >
<! ELEMENT FULLADDRESS (COMPANY, ADDRESSLINE+, CITY, STATE, POSTALCODE,
    COUNTRY, CONTACT, PHONE, FAX?) >
<! ELEMENT COMPANY (#PCDATA) >
<! ELEMENT ADDRESSLINE (#PCDATA) >
<! ELEMENT CITY (#PCDATA) >
<! ELEMENT STATE (#PCDATA) >
<! ELEMENT POSTALCODE (#PCDATA) >
<! ELEMENT COUNTRY (#PCDATA) >
<! ELEMENT CONTACT (#PCDATA | REFERENCE) >
<! ELEMENT PHONE (#PCDATA) >
<! ELEMENT FAX (#PCDATA) >
```

```

<! ELEMENT PRIORITY (#PCDATA)>
<! ELEMENT TOTALS (TOTALITEMS, TOTALQUANTITY, TOTALCOST)>
<! -NOTE TOTALS ARE FOR CHECKING DATA ONLY AND DO NOT REFLECT FINAL COSTS OR
QUANTITIES ->
<! ELEMENT TOTALITEMS (#PCDATA)>
<! ELEMENT TOTALQUANTITY (#PCDATA)>
<! ELEMENT TOTALCOST (#PCDATA)>

```

第二种 DTD 通过参数实体来包括第一种 DTD, 并且为实际的行项目提供信息。

```

<! ENTITY % ORDER SYSTEM "order.dtd" >
%ORDER;
<! ELEMENT ITEM (BOOK, QUANTITY, PRIORITY?, DISCOUNT, EXTENDED COST)>
<! ELEMENT DISCOUNT (#PCDATA)>
<! ELEMENT EXTENDED COST (#PCDATA)>
<! ELEMENT BOOK (ISBN, BOOKTITLE, PRICE)>
<! ELEMENT ISBN (#PCDATA)>
<! ELEMENT BOOKTITLE (#PCDATA)>
<! ELEMENT PRICE (#PCDATA)>
<! ELEMENT QUANTITY (NUMBER, CARTON?)>
<! ATTLIST QUANTITY
    SHIPCQ (NO | ROUNDUP | ROUNDDOWN | ROUNDCLOSEST) "NO">
<! ELEMENT NUMBER (#PCDATA)>
<! ELEMENT CARTON (#PCDATA)>

```

既然我们有了结构, 现在是我们学习如何使用它的时候了。我们的定单文档只直接调用图书 DTD。定单 DTD 作为图书 DTD 的一部分处理, 并且不需要直接调用。

```

<? xml version="1.0" encoding="UTF-8" ? >
<! DOCTYPE ORDER SYSTEM "book.dtd" >
<ORDER>
<BILLTO>
<REFERENCE>8345A</REFERENCE>
</BILLTO>
<SHIPTO>
<REFERENCE>8345A</REFERENCE>
<SHIPVIA><REFERENCE>2A</REFERENCE></SHIPVIA>
</SHIPTO>
<CONTACT>Burnie Orange</CONTACT>
<PRIORITY>Normal</PRIORITY>
<ITEM>
<BOOK><ISBN>155828592X</ISBN><BOOKTITLE>XML: A Primer</BOOKTITLE><

```



```

PRICE> $ 24.95< /PRICE>< /BOOK>
<QUANTITY SHIPCOQ="ROUNDDOWN"><NUMBER>100< /NUMBER><CARTON>20< /
CARTON>< /QUANTITY>
<DISCOUNT> .42< /DISCOUNT>
<EXTENDED COST> $ 1447.10< /EXTENDED COST>
< /ITEM>
<ITEM>
<BOOK><ISBN>1558285288< /ISBN><BOOKTITLE>MIME, UNENCODE, & amp; ZIP<
/BOOKTITLE><PRICE> $ 24.95< /PRICE>< /BOOK>
<QUANTITY SHIPCOQ="ROUNDDOWN"><NUMBER>100< /NUMBER><CARTON>20< /
CARTON>< /QUANTITY>
<DISCOUNT> .42< /DISCOUNT>
<EXTENDED COST> $ 1447.10< /EXTENDED COST>
< /ITEM>
<ITEM>
<BOOK><ISBN>1558514716< /ISBN><BOOKTITLE>Graphical Applications with Tcl
& amp; Tk< /BOOKTITLE><PRICE> $ 39.95< /PRICE>< /BOOK>
<QUANTITY><NUMBER>16< /NUMBER><CARTON>16< /CARTON>< /QUANTITY>
<DISCOUNT> .42< /DISCOUNT>
<EXTENDED COST> $ 370.74< /EXTENDED COST>
< /ITEM>
<ITEM>
<BOOK><ISBN>155828480X< /ISBN><BOOKTITLE>World Wide Web Bible< /BOOKTI-
TLE><PRICE> $ 29.95< /PRICE>< /BOOK>
<QUANTITY><NUMBER>10< /NUMBER><CARTON>10< /CARTON>< /QUANTITY>
<DISCOUNT> .42< /DISCOUNT>
<EXTENDED COST> 173.71< /EXTENDED COST>
< /ITEM>
<ITEM>
<BOOK><ISBN>1558284783< /ISBN><BOOKTITLE>Introduction to CGI /Perl< /BOOK-
TITLE><PRICE> $ 19.95< /PRICE>< /BOOK>
<QUANTITY SHIPCOQ="ROUNDDOWN"><NUMBER>24< /NUMBER><CARTON>24< /
CARTON>< /QUANTITY>
<DISCOUNT> .42< /DISCOUNT>
<EXTENDED COST> 277.70< /EXTENDED COST>
< /ITEM>
<TOTALS><TOTALITEMS>5< /TOTALITEMS><TOTALQUANTITY>320< /TOTALQUAN-
TITY>
<TOTALCOST> $ 3716.35< /TOTALCOST>
< /TOTALS>
< /ORDER>

```

虽然这种方法,不如以前那有固定长度或灵活不受限制的文件那么紧凑,但它更易于人来阅读。它的灵活性也具有更大的优点,因为它不需要所有的信息一直在线。

网络的发展降低了信息传送的成本,使这种冗长可以接受。在 DTD 的周围,创建一个处理应用程序,并把它和定单系统连接起来,这将付出一些努力。但是这些工作有希望得到回报,使 DTD 增加灵活性,客户可以使用他们选择的任何一种 XML 处理器。

信息交换

虽然定单经常是公司之间信息交换的最重要形式,其它不直接产生收益的信息也需要共享,甚至是在竞争对手之间共享。多个企业必需组织多个部分的信息,为信息交换系统提供丰富的资源。建立这样的交换系统是困难的,因为一些公司认为把他们专有的信息让大家共享的话,他们的公司会损失很大,但是他们通过信息共享得到的东西比失去的东西更多。既然涉及建立这些交换系统的工作类似前面章节描述过的把信息放到文档中,那么多家公司共享文档会产生另外的挑战。本节将不创建 DTD,它可能比已经描述的订单更属于是工业范围的。我们将探索一些已经正在开发中的,有关商务的 XML 标准。

交叉参考

查看更详细的 XML 项目列表,请访问:



<http://www.oasis-open.org/cover/xml.html>。

有关 XML 方案的目录,请访问:

<http://www.schema.net/>

ECIX: SGML 的例子

SGML 是商业应用最广泛引用的成功例子之一,值得 XML 思考的一个例子是 Pinnacles 电子元件信息交换(ECIX)集团。开始时,作为 Hitachi、Intel、松下半导体、飞利浦半导体和 Texas 仪器公司的合作联盟,它现在是 Silicon Integration Initiative (<http://www.si2.org>)公司的一部分。ECIX 开始作为电子图书和文件的标准。当在单晶片集成多个元器件取得进展时,集成电路的平均尺寸也在迅速增大,并且电子元件的文件也在急剧增大。想制造新的集成电路的工程师,必须花费时间查找文件,并在 CAD(计算机辅助设计)软件上重新设计集成电路,而不是在电子芯片上设计。半导体公司联合起来,使制造交换电子信息的集成电路变得容易。他们建立了两个独立的电路芯片标准:Pinnacles Component Information Standard(PCIS Pinnacles 元

件信息标准)和 Component Information Dictionary Stand(CIDS 元件信息词典标准)。PCIS 提供给工程师可读取的、易于输入到 CAD 系统的电子格式的元件信息。CIDS 提供元件的术语和定义词典以便于查找和标准化。

PCIS 原文档提供有关元件包装、功能描述、管脚、焊接安装、指令设置、登记设置和内存映射的详细信息。PCIS 的原文件广泛地参考了 CIDS 词典,使用了标准化定义源文件。作为结果,工程师可以查询 PCIS 文档,而不是没完没了地翻书。所有的定义都是标准化的,大大方便了工程师在设计中使用其它企业生产的集成电路。通过把他们的数据公开,并使用电子搜索和操作,这样的工业环境使得竞争的企业也可以相互买卖他们的产品。设计者可以方便地找到和使用适合他们任务的集成电路,而不是重新设计集成电路。

交叉参考



如需要更多有关 ECIX 的信息,请访问站点:<http://www.si2.org/ecix/>。

开放贸易协议(OPT)

为处理 Internet 上的商务交易,开放贸易协议(OPT)提供了开放的 XML 格式。OPT 正在由 OPT 的合作伙伴开发,这些合作的公司包括系统、软件、网络销售商和财务机构。OPT 希望建立与现存处理购买系统类似的 XML,不管他们是在商家之间,还是在零售客户和商家之间。OPT 支持电子现金机构,鼓励客户和商家之间的交易直接联系(从当前的情况看,许多客户求助于信用卡公司解决纠纷,而不是他们直接处理,这与直接联系处理还有一步之遥)。OPT 的设计用来处理各种现存的机构,从安全电子交易(SET)协议到 HTTP 协议。

交叉参考



如需要更多有关 OPT 的信息,请访问 OPT 联盟站点:<http://www.ofx.net/ofx/ab-main.asp>。

银行 Internet 支付系统(BIPS)

银行 Internet 支付系统(BIPS),财务服务技术联盟(FSTC)项目是以 XML 为基础的系统,其支持诸如有线传输以及自动清算传输之类的银行交易。BIPS 不是一套完整的电子商务系统,它注重电子商务的一个方面即支付系统,把其它的任务交给其它协议(BIPS 甚至不能处理所有的支付问题;FSTC 对检查支付有另一个标准,“E-Check”)。BIPS 是可以组成其它系统的基本单元,如 CommerceNet(另一个工业协

会),而不仅仅是一个单独的应用程序。

交叉参考



如需要更多有关 BIPS 的信息,请访问站点:<http://www.fstc.org/projects/bips/>。

如需要更多有关 CommerceNet 的信息,请访问站点:<http://www.commerce.net/>。

XML /EDI

XML /EDI 是一个工作组织,其目的是建立使用 XML 的、支持在电子数据交换(EDI)网络上运行的交易——主要是商家到商家的交易。XML /EDI 应用 XML 语法,使用更灵活的数据结构和通用工具,作为当前这些交易使用格式(象 X12 和 EDI-FAACT)的替代品。XML 可以使用非上架工具的能力(如 XML 解析器和 Internet 协议)可以大大减少交易成本,同时提供了描述商务交易的更丰富的词表。

交叉参考



如需要更多有关 XML /EDI 的信息,请访问站点:<http://www.xmledi.net/>。如考虑 EEMA 提议的,访问站点:<http://www.edi-tie.nl/edifact/xml-edi.htm>,有关集成 X12 协议和数据交换标准协会(DISA)XML 的信息,请访问站点:<http://www.disa.org/x12xmlfaq.html>。

信息和内容交换(ICE)

信息和内容交换(Information and Content Exchange)是一个很难分类的协议。一方面它管理文档内容,包括 XML,但是最主要的是,它使用 XML 创建一套管理商务的工具,为内容交换建立订户辛迪加模型。它自己不处理安全交易和承担财务责任,而是依靠 HTTP、SSL 和其它支持结构处理这些问题。ICE 专著于管理订户与辛迪加之间的关系,通过明确制定双方认可的稳定规则,简化了让 Web 站点分享信息的任务。如果双方关系出现问题,辛迪加可以在订户上放一个塞子,或者订户可以放弃订阅,而不会产生很大的麻烦。ICE 鼓励信息共享和在 Web 外重复使用,从而降低了发布信息的成本,同时更易于管理和更有潜在市场。

交叉参考



要阅读 ICE 协议,请访问站点:<http://www.w3.org/TR/NOTE-ice>。如需要更多有

关最新的开发信息,可访问站点:<http://www.vignette.com/CDA/Book/0,1038,S1-L1-173-174-156,00.html>。

美国报纸协会标准分类广告数据

分类广告是事务处理的要求,虽然许多比以上定义更全面的分类被设计使用。分类广告有时是商家对消费者的,有时是消费者与消费者之间的,种类繁多,从求助广告到房地产和汽车销售广告,应有尽有。美国报纸协会(NAA)正在着重对雇佣、房地产和车辆(包括自行车)的广告分类制定最初的标准。NAA 分类广告交换标准的制定是为了使报纸共享高度标记化的广告。这已经在一些地区得到批准,高度标记化的广告可以使创建搜索引擎和相关的处理程序成为可能,而不必让读者查看大量的广告才能找到所需广告。与此同时,报纸仍然可以印刷分类广告,这使得报纸可以提供有价值的服务。从长远来看,分类广告也要和交易处理系统集成,形成完善的处理小事务交易的系统。

交叉参考



如需要最新的有关 NAA's 分类广告任务信息,可访问站点:<http://www.naa.org/technology/cisstdtf>。

这些只是公开宣布的使用 XML 的商业应用项目。当更多的工业化的集团和公司开始使用 XML 时,它将变得更普遍,许多规则基础标准也即将萌生。

第 八 章

XML 用于文档管理

XML 允许管理文档的方法产生变革。文档管理系统存储文档、记录文档内容、控制对文档的访问,使用户能够快速找到关键信息。许多当前的文档管理系统只是一个巨大的文件柜,存储的文档只是几个关键词和提供快速搜索的日期。XML 文档携带着创建更稳健文档系统所需的信息和结构,它可以组织收集以前留在文件柜和垃圾箱的信息。传统的文件结构,甚至 Web 只提供最低级别的存储和访问,但是更完善的系统正在开始成为办公室的标准设备。通过给文档管理系统一个清晰的文档内容图,标记语言可以更有效地控制更大的文档。搜索可以限定到单个元素,减少了处理所需访问文档的数量和文档不匹配的数量。如果 XML 可以连接到用 SGML 创建的广泛文档系统,XML 文档管理系统就最终可以代替现存的文件系统。

要取得这样的成果,需要对组织查看文档的方法作出重大调整,并开发新的管理信息工具。无纸化办公难以实现的主要原因是需要处理以前留下的有纸文档。处理文件作为文档管理基本单位,很像在有纸文件系统中处理单个文件作为基本单位。文档整齐地装订(或不装订),存放在相关文档组的文件夹中。有时同一个文件需要存在多个地方;有时整个文件夹都必须搜索来寻找一个文档。把处理文档当作信息集装箱,而不是一个基本单位,是告别过去陈旧工具的第一大步,大多数人还在使用这些陈旧的工具,管理他们的文档,甚至计算机化的文档。通过把重点从文档转移到文档中的信息,管理工具最终可以提供交叉参考的搜索工具,使计算机化的信息系统真正有用。

XML 的继承性:SGML 和文档管理

XML 继承了大量的来自 SGML 文档管理开发的软件。SGML 发现它在大规模文档管理系统最有用,比如出版社和政府机构(特别是国防),著名的 IBM 就是最早开发标记的公司。用户层在过去的二十多年中逐渐扩大,这也包括较小组织、学术界、工程和分类项目中的用户。例如 Linux 文档正在变为以 DocBook DTD 为基础的

SGML 格式。

注意



为了全面了解 SGML 文档管理的思想,包括一些案例的研究,请看:Chet Ensign's \$ GML: The Billion Dollar Secret(Prentice-Hall,1997)。

SGML 文档管理系统变得越来越大,经常需要把 SGML 转换成其它的格式来显示,把多家零售商的工具结合起来构成了完整的解决方法,一家公司的 SGML 是不能解决所有问题的。各种各样的开发工具、咨询服务、文档仓库、搜索引擎和定制的应用程序,需要集成到一起,为共享文档系统提供完整的解决方案。因为 SGML 仍然是定制的解决方案的工具,这些解决方案和它们的单元价格仍然很贵,这也是 SGML 被广泛采用的重要障碍。然而,并不是所有的 SGML 的产品都很贵,James Clark's SP 解析器就是免费的,在它的站点可以得到:<http://www.jclark.com/>。另外 Corel's WordPerfect 8 包含 SGML 工具和 Visual DTD Builder。

许多 SGML 商家正在对使用他们的 XML 产品进行重新定位。虽然在技术上,XML 是 SGML 的子集,但一些商家比另外的商家却要轻松一些。当购买这些以 XML 为重点解决的产品时,总是确保输出为 XML 方式,而不需要在 XML 创建过程中去掉 SGML 的任何特性。掌握了文档设计和信息建模中获得的专门技术,就可以直接把 SGML 转换到 XML。在你学会了一些基本的 XML 后,就可以探索 SGML 上的资源和数据模型,通过勤奋工作获得的智慧可以使你避免犯别人已解决了的错误。

提示



拥有各种 SGML 和 XML 资源的奇特地方是 Robin Cover's SGML/XML Web 页面,它的地址是:<http://www.oasis-open.org/covers/sgml-xml.html>。这个站点列出了书、论文、文章、软件和其它可信的详细材料。

XML 文档管理的未来

XML 可以使文档管理系统以一部分的形式存储文档,而不是以一大块不可辨认的信息形式存储。从文档的内核中去掉格式化的信息,使搜索引擎和类似的工具不用考虑格式代码,就更加方便地解析文本。给元素和属性分配在信息交换的上下文中有含义的名字,是这一过程的关键,为得是有效地存储和恢复文档。从低层为 XML 编写的文档管理工具,甚至能以分层组织的数据库中元素的形式存储文档。XML 代替文件系统,可以运行更复杂的可以反映文档结构方式存储的 XML 文档仓库。

XML 刚刚开始作为文档格式。使 XML 文档管理的梦想变成现实的关键部分是用于创建文档的工具。如果 XML 工具像本书中的那样用手工编码那样愚蠢,那么就没有人愿意使用它们。即使 XML 工具可能需要界面作重大的改变,许多 WYSIWYG 工具也已为这种转变做好了准备。Microsoft 已经宣布支持 XML 作为 Microsoft Office 的普通文件格式,虽然它的能力有限。接下来的例子虽然是手工编码,但是大多数使用它的人不用直接键入标记。

创建文档管理应用程序大大超出了本书的范围。本章余下的部分将探索创建满足实际商务需要的 DTD,创建容易搜索、满足公司多方面需要的存储文档。第一个例子是使商业便函标准化,它是商务文档最普通的类型。第二个例子为较大公司的普遍问题提出有针对性的解决办法。两个例子都是一个小单元,有希望成为大的存储信息的连锁系统,便于访问和处理。

向着无纸化办公迈出的一小步

许多公司每周以商业便函的形式撰写时事通讯,虽然许多人对保存和管理这些商业便函的明智提出疑问,但是商业便函和其它小规模时事通讯正在飞速发展。信息自由法案(FOIA),要求联邦政府必须保留活动记录,并把它们对公众开放。目前处理 FOIA 需要花费几个星期或几个月的时间。基于 DTD 以 XML 为基础的系统,就像我们正在使用的商业便函系统,它大大减少了寻找文档的时间。DTD 也很容易被其它的一些任务重复使用(例如,电子邮件是典型的使用类似模型格式化的)。

实际上,没有人愿意使用 XML 中的手工编码商业便函。在商业便函的例子中,用十分简单的结构,程序便能够读取商业便函 DTD,并把它作为一个模板,创建所需信息的形式。XML 可以使用各种类型的信息,不仅仅是文档表达信息。高级的 XML 处理器可以通过访问程序而不是通常的点击文档中的字段来创建商业便函。

创建商业便函的第一步是访问人和收集商业便函,还要检查便函是如何聚集的。大多数公司使用十分标准的格式,在顶部有信头、接着是分类列表、便函来源、简要的标题、然后是内容。在一些案例中,如果打字员不是作者本人,那么在便函的底部做出提示。在我们的例子中,我们选择假象的公司 Jimmy's Delectable Car Parts Design (JDCPD)作为例子介绍。JDCPD 是出售各种汽车和卡车高性能零件的成功企业。典型的商业便函看起来像图 8-1。

一些商业便函更复杂,像图 8-2 所示的那样。JDCPD 有一个公共关系办公室,它也每周出版内部时事通讯。时事通讯有一些员工感兴趣的简短项目,以友好的非正式的式样发表。

公共关系部门喜欢把便函的格式用到其它的表达中,虽然他们还没有制定计划。他们知道将来的时事通讯,特别是即将来临的 Intranet(企业内部互连网)时事通讯的编辑,他们想让它包含图画,并用各种的回型针或其它东西装饰页面。新闻发布虽然不包含在这个项目中,但是可以想象包括进去。

然一些设计者喜欢在他们的便函中加入图画作为参考材料。

微不足道的便函明显地可以处理各种工作任务,即使它不需要携带很多信息。这些工作任务不都是一致的,它们也不可能一步完成。创建可工作的 DTD 需要实践和许多不完全支持电子便函的人认可。

开始定义文档类型最好的地方,通常是已经有大部分结构的文档区。在这个例子中,它是在标题区。标题总是包含分类列表、来源、主题和日期。目前的分类列表可以是邮件所明白的任何东西,但是为所有的便函使用电子邮件的前景将在不久的将来实现。最初我们创建的便函很不详细,它象以下形式:

```
<! ELEMENT MEMO (HEADER, MAIN)>
<! - MEMO DTD Version 0.1 - Experimental Use Only ->
<! ELEMENT HEADER (DISTRIBUTION, SUBJECT, DATE)>
<! ELEMENT DISTRIBUTION (TO+ , CC?, FROM+)>
<! ELEMENT TO ( #PCDATA)>
<! ELEMENT CC ( #PCDATA)>
<! ELEMENT FROM ( #PCDATA)>
<! ELEMENT SUBJECT ( #PCDATA)>
<! ELEMENT DATE ( #PCDATA)>
<! ELEMENT AUTHOR ( #PCDATA)>
<! ELEMENT TYPIST ( #PCDATA)>
<! ELEMENT MAIN ( #PCDATA | AUTHOR | TYPIST) * )>
```

这对于一个简单的文件已经足够的,要看它可以做什么,我们需要创建一个样本文档:

```
<? xml version="1.0" standalone="no" encoding="UTF-8"? >
<! DOCTYPE MEMO SYSTEM "memo.dtd">
<MEMO>
<HEADER>
<DISTRIBUTION>
<TO>To: Jimmy</TO>
<FROM>From: Simon</FROM>
</DISTRIBUTION>
<SUBJECT>Re: Sample Document Created with Memo DTD</SUBJECT>
<DATE>Date: 10 /11 /1999</DATE>
</HEADER>
<MAIN>
```

I just thought you might like to see what a memo in XML looks like. Thanks for the vote of confidence at the last meeting. With any luck, this will make our transition to electronic documents reasonably painless.

```
<AUTHOR>SSL< /AUTHOR>
< /MAIN>
< /MEMO>
```

即使解析的很好,它也有一些问题:

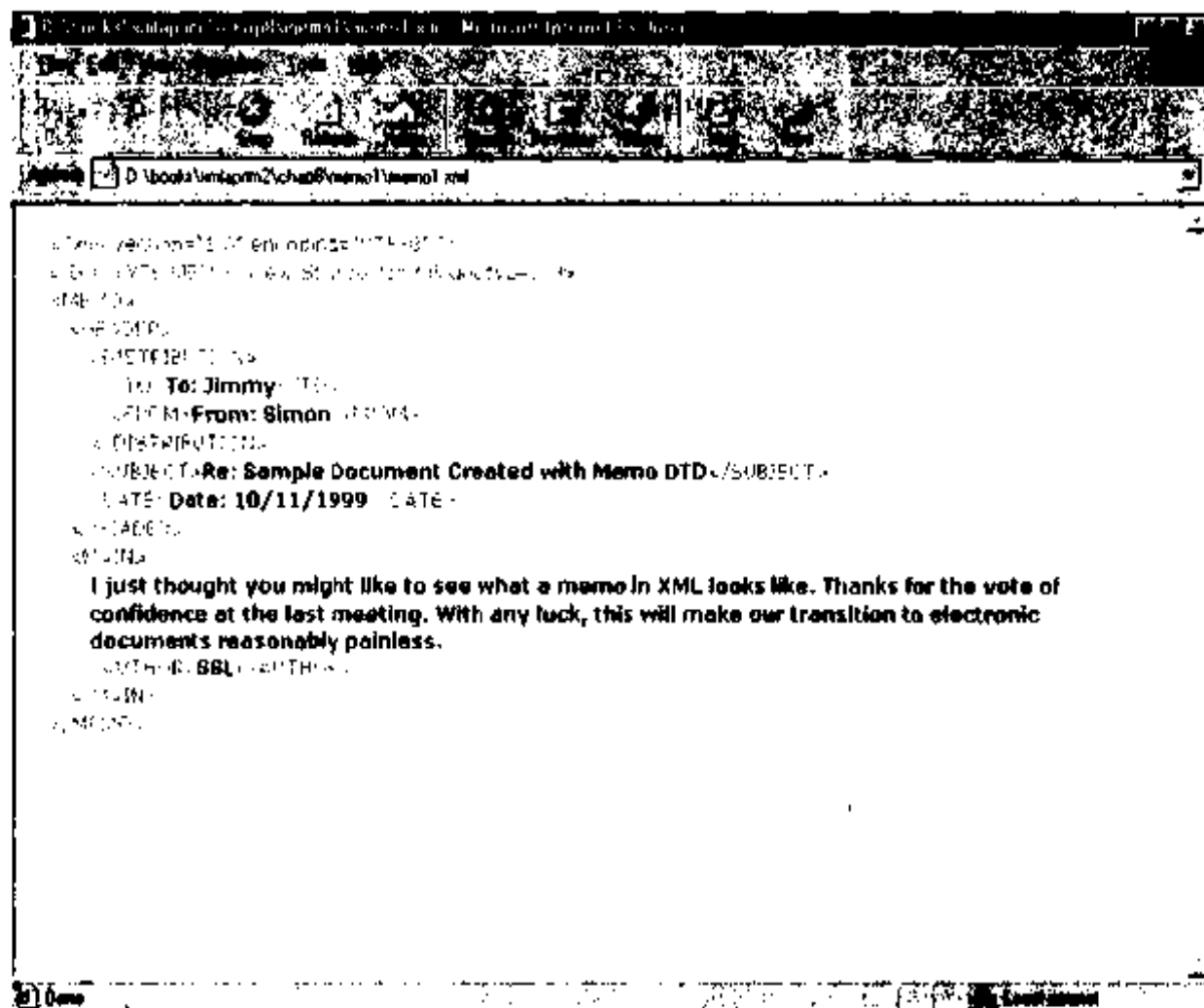


图 8-3 解析的便函

分类字段是主要问题。因为它必须包括“TO:,”“CC:,”和“FORM:”标题,它们没有实际的意义。这对页面文件是可以接受的,人们可以通过读列表使它有意义,但是它将阻止电子邮件正常工作,并使文档管理系统难以记录发送者和接收者。这些字段都需要进一步分解。幸运的是,他们都使用相同种类的在公司内涉及个人或组织的信息。我们的解决办法需要额外元素识别发送者和接收者。由于这两个组是由一套有相同地址的人和组,因此它们可以共享元素:

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

现在,还不能完全确定地址是什么。目前,#PCDATA 类型接受这种不确定性,但是 JDCPD 在制定出目录结构后,有可能移到更加明确的模型上。如果他们只是要把名字和 e-mail 地址结合在一起,他们可以使用下面的代码:

```
<! ELEMENT IDENTITY (NAME, EMAIL?)>
<! NAME ( #PCDATA )>
<! ELEMENT EMAIL ( #PCDATA )>
```

式版本内容的阅读器。重新结合日期单元比解释一个不熟悉的格式的日期容易。给一个足够好的处理应用程序,日期信息也可以以便函文档属性而不是子元素存储。

注意

这种日期解决方法毫无疑问是杂牌的。然而目前它似乎是最安全的方法,所需应用程序的解析很少,及以后容易转换到其它格式。当 XML 能更强地支持数据类型时,更好的办法将会出现。对把属性加到可以保留你喜欢的数据格式的 DATE 元素,提供你坚持的标准格式。

在考虑到所有这些后,我们的 DTD 如下所示:

```
<! ELEMENT MEMO (HEADER, MAIN)>
<! -MEMO DTD Version 0.2 - Experimental Use Only->
<! ELEMENT HEADER (DISTRIBUTION, SUBJECT, DATE)>
<! ELEMENT DISTRIBUTION (TO, CC?, FROM)>
<! ELEMENT TO ( #PCDATA | IDENTITY) * >
<! ELEMENT CC ( #PCDATA | IDENTITY) * >
<! ELEMENT FROM ( #PCDATA | IDENTITY) * >
<! ELEMENT IDENTITY ( #PCDATA)>
<! -May add NAMEID attribute for easier connection to directory structures later ->
<! ELEMENT SUBJECT ( #PCDATA | DESCRIP) * >
<! ELEMENT DESCRIP ( #PCDATA)>
<! ELEMENT DATE (YEAR, MONTH, DAY, (HOUR, MINUTE, SECOND)?)>
<! ELEMENT YEAR ( #PCDATA)>
<! ELEMENT MONTH ( #PCDATA)>
<! ELEMENT DAY ( #PCDATA)>
<! ELEMENT HOUR ( #PCDATA)>
<! ELEMENT MINUTE ( #PCDATA)>
<! ELEMENT SECOND ( #PCDATA)>
<! ELEMENT AUTHOR (IDENTITY+)>
<! ELEMENT TYPIST (IDENTITY+)>
<! ELEMENT MAIN ( #PCDATA | AUTHOR | TYPIST) * >
```

新文档有更多的标记:

```
<? xml version="1.0" standalone="no" encoding="UTF-8"? >
<! DOCTYPE MEMO SYSTEM "memo.dtd">
<MEMO>
<HEADER>
<DISTRIBUTION>
<TO>To: <IDENTITY>Jimmy< /IDENTITY>< /TO>
```

```
<FROM>From: <IDENTITY>Simon</IDENTITY></FROM>
</DISTRIBUTION>
<SUBJECT>Re: <DESCRIP>Sample Document Created with Memo DTD</DESCRIP></SUBJECT>
<DATE><YEAR>1999</YEAR><MONTH>10</MONTH><DAY>11</DAY></DATE>
</HEADER>
<MAIN>
I just thought you might like to see what a memo in XML looks like. Thanks for the vote of confidence at the last meeting. With any luck, this will make our transition to electronic documents reasonably painless.
<AUTHOR><IDENTITY>SSL</IDENTITY></AUTHOR>
</MAIN>
</MEMO>
```

这个解析很好,虽然你可以看到它有好几层:

这个模型适用于大多数简单的文档。标题将使便函正确归档,便于文档管理系统以作者、接收者或标题为基础跟踪文档。主体内容模型仍是空白。现在,在 MAIN 元素中用户可以使用的唯一的内容类型是无格式文本。文档主体需要一些简单的格式化元素来分解文本并为分解便函提供选项。JDCPD 只需要三个选项来产生便函,它们是:PARAGRAPH、HIGHLIGHT(强调重点材料)和 HEADLINE。

```
<! ELEMENT PARAGRAPH ( #PCDATA|HIGHLIGHT|IDENTITY) * >
<! ELEMENT HIGHLIGHT ( #PCDATA) >
<! ELEMENT HEADLINE ( #PCDATA) >
```

注意段元素使编写程序能包括 IDENTITY 信息。从常规的文本中筛选出 IDENTITY 信息,正常情况下是困难的,特别是在非正式的文档中。鼓励规范使用 IDENTITY 信息将使它变得十分简单。所有这些元素都可以容易地把样式连接到用户文档。

整个 DTD 现在如下所示:

```
<! -MEMO DTD Version 0.3 - Experimental Use Only->
<! ELEMENT MEMO (HEAD?,HEADER, MAIN) >
<! -HEADER information for addressing ->
<! ELEMENT HEADER (DISTRIBUTION, SUBJECT, DATE) >
<! ELEMENT DISTRIBUTION (TO, CC?, FROM) >
<! ELEMENT TO ( #PCDATA | IDENTITY) * >
<! ELEMENT CC ( #PCDATA | IDENTITY) * >
<! ELEMENT FROM ( #PCDATA| IDENTITY) * >
<! ELEMENT IDENTITY ( #PCDATA) >
<! -May add NAMEID attribute for easier connection to directory structures later ->
<! ELEMENT SUBJECT ( #PCDATA | DESCRIP) * >
<! ELEMENT DESCRIP ( #PCDATA) >
<! ELEMENT DATE (YEAR, MONTH, DAY, (HOUR,MINUTE,SECOND)?) >
<! ELEMENT YEAR ( #PCDATA) >
<! ELEMENT MONTH ( #PCDATA) >
<! ELEMENT DAY ( #PCDATA) >
<! ELEMENT HOUR ( #PCDATA) >
<! ELEMENT MINUTE ( #PCDATA) >
<! ELEMENT SECOND ( #PCDATA) >
<! ELEMENT AUTHOR (IDENTITY+ ) >
<! ELEMENT TYPIST (IDENTITY+ ) >
<! ELEMENT PARAGRAPH ( #PCDATA|HIGHLIGHT|IDENTITY) * >
<! ELEMENT HIGHLIGHT ( #PCDATA) >
<! ELEMENT HEADLINE ( #PCDATA) >
<! ELEMENT MAIN ((PARAGRAPH | HIGHLIGHT | HEADLINE) * , AUTHOR?, TYPIST?) >
```

对大多数便函来说它都能很好地工作,但是允许文本便函数据以外的内容使用便函 DTD 也可能是有用的。例如,公共关系部门也喜欢使用便函 DTD 做时事通讯。即使他们可以使用 PARAGRAPH、HIGHLIGHT 和 HEADLINE,他们更愿意用对他们的时事通讯更明确的东西,这些将在文档管理系统独立显露。更重要的是,他们想

要以后改变他们的 DTD,而不对整个公司产生反响。解决这些问题的最好方法是分离包含他们元素的 DTD,当需要时,它可以和主要便函 DTD 结合。他们的时事通讯 DTD 看起来如下所示:

```
<! ELEMENT NEWSLETTER (STORY+)>
<! ELEMENT STORY (LEAD, PARAGRAPH*)>
<! ELEMENT LEAD (#PCDATA)>
```

把这个和便函 DTD 结合可能产生一点小问题。最容易的方法是用 NEWSLETTER 元素代替便函中的 MAIN 元素来改变 MEMO 元素:

```
<! ELEMENT MEMO (HEAD?, HEADER, (MAIN | NEWSLETTER))>
```

MAIN 元素可以改变成包含 NEWSLETTER 元素:

```
<! ELEMENT MAIN (((PARAGRAPH | HIGHT HEADLINE)*,
AUTHOR?, TYPIST? | NEWSLETTER))>
```

如果它关掉了,然而公司的其它部门也想重新考虑便函 DTD,这些声明将变的十分愚蠢。如果 XML 允许文档声明元素超过一次(像属性那样),解决方法很简单的:不考虑时事通讯中的 MAIN,在另一个 DTD 中做一个新的 MAIN 声明。这是不可能的,因为在 XML 中超过一次声明元素是错误的,它将使文档无效。最容易的方法是做类似我们第一次的尝试:

```
<! ELEMENT MEMO (HEAD?, HEADER, (MAIN | ALTERNAME))>
```

通过使用 ALTERNATE,我们可以使多用户利用 DTD 和为其它应用程序重新考虑便函成为可能(ALTERNATE 不是关键词——任何元素名,只要在那个 DTD 中的元素没有定义就可以)。公共关系部门可以创建 DTD,它使用和扩展便函 DTD:

```
<! ENTITY % memo SYSTEM "memo.dtd">
%memo;
<! ELEMENT NEWSLETTER (STORY+)>
<! ELEMENT STORY (LEAD, PARAGRAPH*)>
<! ELEMENT LEAD (#PCDATA)>
```

以前显示的时事通讯可以用相当简单的方法转换到 XML:

```
<? xml version="1.0" standalone="no" encoding="UTF-8"? >
<! DOCTYPE MEMO SYSTEM "http://127.0.0.1/newslet2/news.dtd">
<MEMO>
<HEADER>
<DISTRIBUTION>
```



```

<TO>To: <IDENTITY>Jimmy< /IDENTITY>< /TO>
<CC>CC: <IDENTITY>All Employees< /IDENTITY>< /CC>
<FROM>From: <IDENTITY>Lois Turpin, PR Department< /IDENTITY>< /FROM>
< /DISTRIBUTION>
<SUBJECT>Re: <DESCRIP>JDOPD Today - Sample Newsletter Created with Memo DTD< /
DESCRIP>< /SUBJECT>
<DATE>
<YEAR>1999< /YEAR><MONTH>10< /MONTH><DAY>10< /DAY>< /DATE>
< /HEADER>
<ALTERNATE><! -BEGIN ALTERNATE CONTENT TO REPLACE MAIN ->
<NEWSLETTER>
<STORY><LEAD>Design Contest Winner - < /LEAD><PARAGRAPH>Frank Kravitz of the
fuel injection division has won the September award for best car part drawing. Frank's
masterpiece, <HIGHLIGHT>On to the Spark Plug II< /HIGHLIGHT>, will be on display in the
lobby through November 12. Frank also won the award in January and July. < /PARAGRAPH>< /
STORY>
<STORY><LEAD>Donut Reimbursement Ends - < /LEAD><PARAGRAPH>To avoid a
threatened doubling of health insurance premiums, JDOPD has ended its donut reimbursement
program. "We are extremely sad to have to take such unpleasant measures," said Jimmy, "but we
hope our employees will understand the difficult situation we face." < /PARAGRAPH>< /STORY
>
<STORY><LEAD>Rice Cakes Available - < /LEAD><PARAGRAPH>For a limited time, rice
cakes and a variety of herbal teas will be available in Jimmy's office. Employees needing a quick
snack to get them charged up for a hard day of parts designing are welcome to stop by. < /PARA-
GRAPH>< /STORY>
<STORY><LEAD>Remember - < /LEAD><PARAGRAPH>"Parts is parts!" Take pride in
your work. The annual award for best design will be announced in December. < /PARAGRAPH>
< /STORY>
< /NEWSLETTER>< /ALTERNATE>
< /MEMO>

```

这个时事通讯 XML 文件用与便函 DTD 结合的新闻 DTD 能很好地解析。

使用这个模型,其它部分可以创建它们自己的 ALTERNATE 内容模型。例如,设计者可以开发一个包括标注元素的设计摘要,标注元素允许它们包括所有类型的图画和其它额外的信息。公共关系部门可以做同样的包括他们额外的标识语,甚至还可能创建电子版本的公司抬头。虽然便函模型不能处理所有的事情,XML 可以给它提供扩大字段的机会,而不会出现完全过载的问题。

这决不是完成这一任务的唯一方法。除了用 ALTERNATE 元素来扩大字段外,把整个 MAIN 元素定义从主 DTD 移到另外一个文件中。便函的核心作用主要是标题和它的分类结构,因此这将是有意义的。在这个例子中,大约百分之九十的用

DTD 创建的文档是简单的便函,因为时事通讯 DTD 可以重复使用在 MAIN 元素下的 DTD 的部分(例如 PARAGRAPH),保持 DTD 为在便函 DTD 中的简单便函,似乎是最简单的方法。然而其它的选择是把分类信息从便函 DTD 中移出,使它容易被其它 DTD 重新使用。选择选项经常是困难的,要根据你的文档结构和管理系统的特殊需求来决定。



调乏味的。

创建历史:DTD 用于公用内存

大多数公司缺少听取雇员汇报和整理信息的策略,当出现法律纠纷和客户咨询需要重新检查过去的活动时,需要花费大量的时间来确定过去发生了什么。

在这个例子中,我们创建的文档结构是在工程的结尾和当雇员调动工作岗位时,用来保持一些历史的生动性和可使用性。结合文档管理系统,存在结构中的信息,将对过去工程的问题,方便快速全面地做出回答。为大的工程写提议的大企业,经常需要提出工程过去的性能参考。在这个系统里包含的信息将使公司更容易描述他们以前的工作,保留了宝贵的资源,用于开发有远见的提议部分软件。

工程和与它相关的 DTD 可能变的很大,特别是,如果它从工程历史扩大到工程管理。这里的讨论将探索工程管理和开发一些核心文档类型的一些基本需要。

在 DTD 设计开始前,这个工程需要做大量的政治工作。结构化的文档如果不和应用程序兼容,那么它是没用的。即使要特定的元素在文档中出现是容易的,要出门在路上的人填写大量的文字也是困难的。成功地创建这个文档数据库,需要把它加到标准商务处理过程中。这个处理过程需要可能对 DTD 的性质和所需灵活性的级别有直接的影响。如果这个系统用在规范基础上,创建一个客户访问应用程序收集信息还是有价值的。

每个工程可能有一套与之相关的数据,代表从工程整个寿命期间收集到的访问和其它文档。持续数年的工程可以用在单一标题下数量相当的信息结束。即使把所有的信息集合到一个单独的文档中,也可以想象出,大工程很快会为文档管理系统收集十分巨大的信息。虽然单一文档的路径对小工程有很大的意义,如果把信息分解为更容易管理的块,那么一套大的文档将运行的更平稳。即使如此,使用元素可以更容易地找到相关信息,有数千或数十万元素的文档,可能过大而不适合单一文档文件结构。我们的例子将反映通过创建几个文档类型,来满足大的结构的需要,每一个文档类型都可以连接到中心工程档案。

我们将回到复合文档的要求上,对这个特殊的例子,在第十章当我们探究的 XML 链接时使用。现在考虑到作为连接到目录和数据库服务器信息来描述的链接,到这两者的连接都可以用应用程序处理,而不是用 XML 解析器。

这个工程需要一套 DTD,而不仅仅是一个单一的 DTD。因为所有的 DTD 将被合并到一个单一的文档中是可能的,设计者必须注意不要重复使用同一个元素名。

我们将创建一套通用的元素,它们可以在这套所有的文档中使用。IDENTITY 在这样的情况下,是比它在以前的例子中更重要的元素,因为这个工程的参与者和他们的位置需要明确定义。在这个例子中,IDENTITY 将携带把它链接到中心雇员目录,以及为人员名字、当前标题、和位置提供空间的属性值:

```
<! ELEMENT IDENTITY (NAME, TITLE, POSITION?)>
<! ATTLIST IDENTITY
    IDLINK ODATA #REQUIRED>
<! ELEMENT NAME (#PCDATA)
<! ELEMENT TITLE (#PCDATA)>
<! ELEMENT POSITION (#PCDATA)>
```

即使把 NAME 元素分解为姓和名可能是有用的,由于需要 IDENTITY 元素的 IDLINK 属性,它也不是真有必要。IDLINK 将把标识连回到包括一套完整信息的目录。IDENTITY 不对个人限制;它也可以涉及部门甚至公司。对目录的链接将产生一些额外的开销,因为所有通过参考 IDENTITY 的人或组必须进入目录。不是所有的公司在他们的目录中存放客户的信息。

XML 不包括检查 IDLINK 值的方法。它可以在需要时出现,但解析器本身不检查 IDLINK 违反目录。这种逻辑必须放在处理解析器送回信息的应用程序中(应用程序可能是文档管理系统,或其它的准备把信息显示的应用程序)。它可以在用来创建文档的程序中执行。一个简单的查找程序,能够让作者从目录提供的列表中,选择 IDENTITY 值。

LOCATION 元素类似于 IDENTITY 元素——它们涉及个人单位,可以在目录服务器中查找,有可能是存储标识的同一个服务器。然而,由于公司可以简单地使用全世界的定位器,因此对需要所有 LOCATION 元素在目录中有一个链接似乎也不值得。像公司办公室和工作地方的定位器一定要有列表,但是旅馆可能不会需要。

```
<! ELEMENT LOCATION (#PCDATA)>
<! ATTLIST LOCATION
    IDLINK ODATA #IMPLIED>
```

下一个关键元素是日期。文档本身将用它们写的日期标记,使用标准格式的日期使在信息内部搜索其它元素变得更加容易。DATE 元素和以前在便函应用程序中使用的 DATE 元素是相同的,虽然它缺少最初的 #PCDATA 信息:

```
<! ELEMENT DATE (YEAR, MINUTE, SECOND?)>
<! ELEMENT YEAR (#PCDATA)>
<! ELEMENT MONTH (#PCDATA)>
<! ELEMENT DAY (#PCDATA)>
```

```
<! ELEMENT HOUR (#PCDATA)>
<! ELEMENT MINUTE (#PCDATA)>
<! ELEMENT SECOND (#PCDATA)>
```

这个 DTD 毫无疑问地会随着额外元素格式化和标注而急剧膨胀,但是我们可以从简单的开始。我们的内容包括四个元素,它们是:EMPHASIS、RUMOR、QUOTE 和 PARAGRAPH。EMPHASIS 使作者能够找到某一个确定的要点;RUMOR 使作者能够包括不能确定但是有用的内容;QUOTE 使它们包括来自客户和其它的材料。PARAGRAPH 只提供基本的语法结构。这些元素都包含混和的内容,由参数实体 % TEXTELEMENTS; 显示:

```
<! ELEMENT % TEXTELEMENTS "(#PCDATA | EMPHASIS | RUMOR |
QUOTE | IDENTITY | LOCATION | DATE) *">
<! ELEMENT EMPHASIS (%TEXTELEMENT;)>
<! ELEMENT RUMOR (%TEXTELEMENT;)>
<! ELEMENT QUOTE (%TEXTELEMENT;)>
<! ELEMENT PARAGRAPH (%TEXTELEMENT;)>
```

既然我们有一套元素可以在文本内使用,那么让我们定义一些文档级的结构。在这个数据表的信息将存在不同类型的文档中。工程管理者汇报的信息与技术员和会计的不同,这将从不同的文档结构反映出来。对于本例子,我们用工程完成报告,它由工程管理者在工程完后,提供了有关工作完成情况的总体报告。我们的根元素是 FINALREPORT,它包含识别工程、作者、报告日期和文档分类元素。它也给作者提供可以加入对工程总体看法、财务信息、计划信息、有关完成工作的详细信息和来自客户的评价的元素。

```
<! ELEMENT FINALREPORT (PROJECT,IDENTITY,DATE,CLASSIFICATION,
OVERVIEW,FINANCIALS,SCHEDULE,DETAILED,
COMMENDATIONS?)>
```

PROJECT 元素,像 IDENTITY 和 LOCATION 元素一样,链接到其它信息源。这节省了作者重新描述工程、客户、合同类型和其它诸如工程开始日期等稳定信息的工作(开始日期不一定总是稳定的,但是在工程完成时,它就是十分稳定的)。作为结果,PROJECT 元素仍保持简单,把工程名存在 #PCDATA,并通过 PROJLINK 属性链接到更详细的信息:

```
<! ELEMENT PROJECT (#PCDATA)>
<! ATTLIST PROJECT
    PROJLINK CDATA #REQUIRED>
```

接在工程后面的 IDENTITY 和 DATE 元素分别表示这个文档的作者和写作日

期,它们在 PROJECT 元素中的位置是唯一可以识别它们的东西;想要把它做得更清晰的开发者可以为它们创建包装:

```
<! ELEMENT AUTHOR (IDENTITY)>
<! ELEMENT REPORTDATE (DATE)>
```

这需要在 FINALREPORT 元素中有一些小的变化:

```
<! ELEMENT FINALREPORT (PROJECT, AUTHOR, REPORTDATE,
CLASSIFICATION, OVERVIEW, FINANCIALS, SCHEDULE,
COMMENDATIONS?)>
```

CLASSIFICATION 是用来帮助文档管理系统控制对文档的访问。一些文档是 OPEN(开放),对公众不限制,其它可能是 PROPRIETARY(只供公司内部使用),一些可能甚至是 SECRET(在公司内部对特殊的读者使用),其它的也可能是 SPECIAL。SPECIAL 可以对以上的 SECRET 分类,或者它可能是需要对应用程序检查读者标识的总体分类。

不管元素是如何创建的,记住是应用程序而不是解析器必须保证安全。XML 没有内建提供安全的工具,文档管理系统和其它工具必须承担这个责任,在用户的标识和许可生效前,把用户锁在文档之外。

执行这类元素需要做一些选择。执行它的最容易的方法是用 #PCDATA 创建一个 CLASSIFICATION 元素作为它的数据类型。这样,新的类型可以容易加入,但是解析器不检查类型。具有冒险精神的作者可以加入 GOOFY 作为安全分类。

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

执行这个元素的其它方法是用属性把 CLASSIFICATION 做成空元素,表示安全级别:

```
<! ELEMENT CLASSIFICATION EMPTY>
<! ATTLIST CLASSIFICATION
    SECLEVEL (OPEN | PROPRIETARY | SECRET | SPECIAL)
    "SPECIAL"
    SPECIAL CDATA #IMPLIED>
```

SECLEVEL 属性表示安全级别,而 SECLINK 使为 SPECIAL 到安全目录外的链接形式变得容易(根据应用程序的类型,开发者要对不同的值设置缺省,或者简单地到 #REQUIRED)。

我们最后考虑文档的关键部分,在这个工程上,实际完成什么?它的造价是多少?它要花费多长时间。因为这个文档是摘要文档,所有的这些信息可以方便地存贮。有重大责任的帐目和计划信息可以存储在其它的文档(甚至数据库)或通过中心系统链接到工程。用十分类似的方法,PROJECT 元素可以通过 PROJLINK 属性连接。FINALREPORT 文档在这里是为了快速参考,不是每一个买卖的窗口小部件逐行显示。总揽部分是摘要的开始:

```
<! ELEMENT OVERVIEW (PARAGRAPH+) >
```

FINANCIALS 元素不能分解很多,但是仍可以显示工程造价总体说明:

```
<! ELEMENT FINANCIALS (ORIGINALQUOTE, FINALCOST,  
EXPLANATION?) >
```

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

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

```
<! ELEMENT EXPLANATION (PARAGRAPH+) >
```

SCHEDULE 元素提供对工程计划全面的描述,并用 EXPLANATION 元素创建 FINANCIALS:

```
<! ELEMENT SCHEDULE (ORIGINALSCHEDULE, ACTUALSCHEDULE,  
EXPLANATION?) >
```

```
<! ELEMENT ORIGINALSCHEDULE (STATTDATE, ENDDATE?) >
```

```
<! ELEMENT ACTUALSCHEDULE (STATTDATE, ENDDATE?) >
```

```
<! ELEMENT STARTDATE (DATE) >
```

```
<! ELEMENT ENDDATE (DATE) >
```

在许多例子中,DTDS 可以通过在多个文本中使用元素来简化它自己,但是这也会对简单的处理应用程序带来问题。如果某些人需要所有财务交易的说明列表,那么他们需要处理器把在 FINANCIALS 元素中的 EXPLANATION 元素从 SCHEDULE 元素中分离出去。开发者要事先知道处理应用程序的限制。只有这样,创建分离的 FIN-EXPLANATION 和 SCHEDEXPLANATION 元素才会容易。

不管它的名字,DETAIL 元素接收十分简单的 XML 声明。DETAIL 是工程管理者输入详细信息的区,但是从解析器的前景看,所有使用 PARAGRAPH 元素的信息,可以包含在 % TEXTELEMENTS; 参数实体中列出的所有元素。COMMENDATIONS 元素同样是一个文本元素集装箱:

```
<! ELEMENT DETAIL (PARAGRAPH+) >
```

```
<! ELEMENT COMMENDATIONS (PARAGRAPH+) >
```

既然我们已经把所有的部分定义了,那么现在可以把它们合并到 DTD 中。我们

的第一个 DTD 包含这个系统文档所需的全部元素,来提供基本的文本内容类型:

```
<! -TEXT CONTENT ELEMENT INFORMATION ->
<! -IDENTITY INFORMATION ->
<! ELEMENT IDENTITY (NAME, TITLE?, POSITION?)>
<! ATTLIST IDENTITY
    IDLINK CDATA #REQUIRED>
<! ELEMENT NAME (#PCDATA)>
<! ELEMENT TITLE (#PCDATA)>
<! ELEMENT POSITION (#PCDATA)>
<! -LOCATION INFORMATION ->
<! ELEMENT LOCATION (#PCDATA)>
<! ATTLIST LOCATION
    IDLINK CDATA #IMPLIED>
<! -DATE INFORMATION ->
<! ELEMENT DATE (YEAR, MONTH, DAY, (HOUR, MINUTE, SECOND)?)>
<! ELEMENT YEAR (#PCDATA)>
<! ELEMENT MONTH (#PCDATA)>
<! ELEMENT DAY (#PCDATA)>
<! ELEMENT HOUR (#PCDATA)>
<! ELEMENT MINUTE (#PCDATA)>
<! ELEMENT SECOND (#PCDATA)>
<! -OTHER TEXT CONTENT ->
<! ELEMENT EMPHASIS (% TEXTELEMENTS;)>
<! ELEMENT RUMOR (% TEXTELEMENTS;)>
<! ELEMENT QUOTE (% TEXTELEMENTS;)>
<! ELEMENT PARAGRAPH (% TEXTELEMENTS;)>
<! ENTITY % TEXTELEMENTS "(#PCDATA | EMPHASIS | RUMOR | QUOTE | IDENTITY | LO-
CATION | DATE) *">
```

第二块是定义我们的工程报告,它包括使用参数实体的 DTD:

```
<! -DTD for Final Project Reports ->
<! ELEMENT FINALREPORT (PROJECT, AUTHOR, REPORTDATE, CLASSIFICATION,
OVERVIEW, FINANCIALS, SCHEDULE, DETAIL, COMMENDATIONS?)>
<! -Link to text content declarations ->
<! ENTITY % TEXTDECLARATION SYSTEM "textelem.dtd">
%TEXTDECLARATION;
<! -Project Identification ->
<! ELEMENT PROJECT (#PCDATA)>
<! ATTLIST PROJECT
    PROJLINK CDATA #REQUIRED>
<! -AUTHOR AND REPORT DATE WRAPPERS, FOR EASIER SEARCHING ->
```



```
<! ELEMENT AUTHOR (IDENTITY) >
<! ELEMENT REPORTDATE (DATE) >
<! -CLASSIFICATION. ENFORCED BY DOCUMENT MANAGEMENT SYSTEM - >
<! ELEMENT CLASSIFICATION EMPTY >
<! ATTLIST CLASSIFICATION
    SECLEVEL (OPEN | PROPRIETARY | SECRET | SPECIAL) "SPECIAL"
    SECLINK CDATA #IMPLIED >
<! -REPORT ELEMENTS - >
<! ELEMENT OVERVIEW (PARAGRAPH+ ) >
<! ELEMENT FINANCIALS (ORIGINALQUOTE, FINALCOST, EXPLANATION?) >
<! ELEMENT ORIGINALQUOTE ( #PCDATA) >
<! ELEMENT FINALCOST ( #PCDATA) >
<! ELEMENT EXPLANATION (PARAGRAPH+ ) >
<! ELEMENT SCHEDULE (ORIGINALSCHEDULE, ACTUALSCHEDULE, EXPLANATION?) >
<! ELEMENT ORIGINALSCHEDULE (STARTDATE, ENDDATE?) >
<! ELEMENT ACTUALSCHEDULE (STARTDATE, ENDDATE?) >
<! ELEMENT STARTDATE (DATE) >
<! ELEMENT ENDDATE (DATE) >
<! ELEMENT DETAIL (PARAGRAPH+ ) >
<! ELEMENT COMMENDATIONS (PARAGRAPH+ ) >
```

这个 DTD 可能只是一个更大更精致结构的开始。用创建好的文档管理系统,多文档类型可以在 PROJECT 信息数据库周围建立。用于图像、工作执行详细报告、客户定单、工程花费时间、情况报告的文档类型、甚至工程管理信息也可以保留在同一个系统,保证了安全且易于访问和交叉参考的功能。这些系统已经出现,它们中的大多数无疑需要大量的客户,但是它们将是 XML 信息最有效的管理者。幸运的话,文档管理系统将取代今天使用的文件柜和文件系统,记录大量文档,以及它们的单元,并使之容易访问。

第九章

XML 用于数据驱动应用程序

以前章节的文档大多数与页面文档有关,它是一种人们可以筛选(或者在 Web 浏览器上装入)和阅读的文档。XML 不仅只限于这类信息,但它的许多最早的应用程序提供的信息形式不能显现给人们。XML 在这一领域跨越的一大步是在计算机之间和计算机和其它设备之间的通讯。迄今为止,这已经证明是很困难的事情。DTD 和本章的例子提供调用非传统文档的解决方法。数据结构对 XML 是十分适宜的,但是没有供人来正常阅读的文档。除了为人显示信息外,这些文档也为程序显示信息,程序用这些信息决定它的行为,而不是只在屏幕或页面上显示信息。

交叉参考



本章介绍以数据为重点的 XML 文档和它们的使用,包括一些应用程序。但是它不是 XML 处理的通用指南。需要有关处理 XML 模型和工具的更多信息可查看第十二章。

用于互相交换的数据

最简单的(也可能是最复杂的)XML 应用是做不同类型系统间的交换格式传送。虽然它们变化无穷,所涉及的概念却是相当简单的。象在第四章简单显示的那样,XML 可以用做几乎任何以文本为基础的信息的集装箱,简化了从一个应用程序到另一个应用程序间,装运信息的工作,即使应用程序设计者不知到其它的产品,只知道文件格式规则。制成表的数据通常在相关的数据库中传送,可能不适合 XML 的层状结构,XML 可以毫无问题地灵活处理表格,数据库商家(像 Oracle 和 IBM)正在把 XML 解析器和界面加到他们现存的产品中。几乎任何应用程序,不只是数据库,需要把信息转换成 XML 可以使用的,可普遍理解的层次格式。做这样的工作需要把信息从一个应用程序中卸下,并装到另外一个应用程序中。这两个应用程序需要包含能解析 XML 的单元,和到它们内部数据结构的相关文件内容,并用它们内部数据

结构,把它们输出到 XML 文件。图 9-1 演示了这一过程。

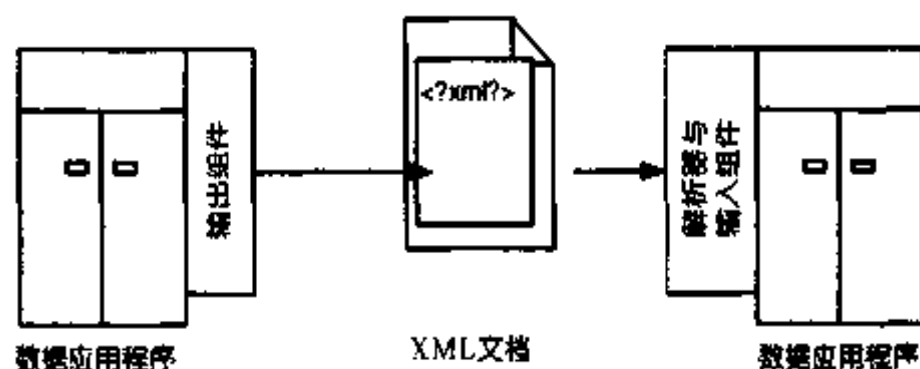


图 9-1 用 XML 文件做输入输出交换

现在,数据库商家宣布能使这一过程简单的工具,但是大多数当前的执行还包括一个 XML 解析器,它使用 SQL(虽然 JDBC 和 ODBC 是典型的)把信息加到数据库或 CGI 样式的程序,然后把数据库信息输出到模板。非数据库的应用程序需要一些专用程序,很像用于输入输出无格式文本和不受限制文本版本的信息。虽然这个技术看起来原始,但 XML 结构能很好地和原始数据交换匹配。它使用双字节字符集作为文本处理的基础,简化了许多涉及使用不同字符集的信息交换的问题。它的结构使它比当前使用的提供基本交换服务不受限制的格式拥有更高的可靠性和可管理性。

少数 XML 中心数据库工具开始出现。对象设计(<http://www.objectdesign.com>)已经创建了 eXcelon,它是提供 XML 服务的首尾相关数据库。Bluestone's XML-Server(<http://www.bluestone.com/xml/XML-Server/>)是一个更加独立的数据库,它可以和客户-服务器体系结构集成。老的层状数据库,在主机计算中仍然重要,可以在 XML 中恢复,正像商家把他们的产品重新改造成以 XML 为重点的工具包一样,如 SoftwareAG's Tamino(<http://www.softwareag.com/tamino>)。

用于控制的数据

许多应用程序执行有限数目的任务,只需要一套数据发送它们。编写一个客户 VAR(录象机)程序,想要给它开始和停止的时间,来记录、频道和磁带速度信息。没有这样的情况(除非一些十分不寻常的 VCR 出现),用户必须写一个代码,告诉 VCR 检查时间并把它和表格里的时间相比,选择频道,移动机械开始记录过程,检查时间,直到停止时间到来。

房屋控制

我们的第一个例子把 XML 应用到与计算机领域十分不同的设备控制。给一个

与一小套输入相对应不需要处理返回值的系统,你可以在纯数据中编写一个程序。开始,我们创建一个能够控制一套电灯开关(或者其它电子设备)的 DTD。灯的开关是最简单的例子,但是许多情况下,需要控制数百或者数千个灯。我们的例子在一个典型的房间以点灯开始,但是它可以扩大到展览灯光甚至舞台灯光。

我的父母过去每几个月常收到来自 DAK(直销电子分类)的目录;它填满了奇怪的立体声设备和小配件。DAK 携带的最怪异的项目是 X-10 系统,它可以使你遥控电子设备。最初是在电源插座和灯或其它电子设备插头之间插入控制单元模块。控制单元通过电线给单个的盒(共有 256 个)发送信号,告诉它们打开、关闭或调暗。最后产生了一系列计算机界面,它可以容易地用程序控制设备在每天的不同时间打开或关闭,这是一个更贵的,但是比刻度盘更精确的设备定时器。现在它变得更加精确(详情请访问站点:<http://www.X10.com>)。遥控可以管理一个房子而不仅是一个电视机,它可以很容易地使你不用起床就能管理房子中的重要部分。

虽然我们开发的 DTD 不是专门为了 X-10 界面,但是你可以容易地把这些文档中的数据转换成控制电子设备的信号。X-10 系统实际上使用一套有限的命令控制它的设备,但是我们的 DTD 不用担心这些命令。它只定义控制系统要达到的特殊状态。处理应用程序将接受由解析器送回的信息,然后决定要达到所要求的状态,应采取的命令。不是给控制器顺序步骤,我们的文档给它一个期望的结果,让控制器自己计算出如何最好地实现它。我们将控制两种不同类型的设备:灯光和器具。灯光可以是打开、关闭和暗淡,而器具只有打开和关闭(如果有人想通过遥控扩展这个控制,不仅只是控制电源,扩展 DTD 也不是困难的)。灯光和器具通过十六进制的地址识别。地址以一个房子的代码 A-P 开始。通常用户在一个字母上建立一个完整的系统,它限定为 16 个设备,避免和其它设备冲突。没有配件意识的邻居可以使用多房子的代码。单位代码识别控制模块。它是从 1 到 16 的编号。设备模块可以设置成相同的地址;所有具有相同地址的设备模块都将同时响应命令。在地址 B10 上的三个灯,当受到发往 B10 的命令时,将同时打开、关闭或暗淡。我们的 DTD 能够使用户创建文档给系统下达命令。计算机将处理所有的这些文档,但是我们将保持它们容易被人阅读和编辑。要实现这一目的,我们的文档必须包括所有在系统中模块的描述,以及所需要的状态和触发它们的条件。

```
<! ELEMENT CONTROLSCHEDULE (MODULE * ,STATE * ,TRIGGER * )>
```

我们的模块需要几个识别器。现在我们仍坚持用 X-10 用来建立地址的系统,并加入两块提供额外信息,处理应用程序和人为编辑器:

```
<! ELEMENT MODULE (ADDRESS, TYPE?, DESCRIPTION?)>
<! ELEMENT ADDRESS (HOUSE, UNIT)>
<! ELEMENT HOUSE (#PCDATA)>
<! ELEMENT UNIT (#PCDATA)>
<! ELEMENT TYPE (#PCDATA)>
```

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

我们可以给 MODULE 元素一点额外的灵活性,方法是让它包含 ADDRESS 和 TYPE 元素。我们可以要求用户或者程序创建这些文档跟踪在某一特殊的地址模块是光模块还是器具模块,并且是完全通过地址识别模块。加入 TYPE 元素增加了灵活性使这些文档更轻便。不关心 TYPE 的处理应用程序可以把它去掉,而第一次建立的新应用程序可以使用 TYPE 元素输入有关控制模块更完全的信息。TYPE 也加入了一些灵活性,以防在新的模块出现后,因为命令会随着模块类型不同,收到它的信号会发生变化。(目前,X-10 系统不能做到这点,但是将来更高级的系统可以做到)。DESCRIPTION 为人们编写这些设备程序提供设备和位置描述。ADDRESS 元素使我们能唯一地识别模块。对 X-10 系统的 A-P、1-16 识别器不耐烦的开发者,可以很容易地把它们转换成在 DTD 中代表它们相等的十六进制数:

```
<! ELEMENT ADDRESS (HEXADDRESS | (HOUSE,UNIT)) >
```

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

现在我们将使用 HOUSE 和 UNIT 识别器。即使 MODULE 元素包含 ADDRESS 元素,它也不能发布命令,因为多模块可以共享单个地址。ADDRESS 是发布命令的关键元素。在文档最初的 MODULE 声明后,一系列 STATE 声明会接着出来。STATE 声明定义最后的位置,而不是到达那里的方法。我们为 STATE 提供名字、描述和单元部分的列表:

```
<! ELEMENT STATE (NAME,DESCRIPTION?,COMPONENT+) >
```

NAME 元素提供我们程序用来发现和执行这个 STATE 的参考。DESCRIPTION 元素,和以前模块使用的一样,为人提供描述。COMPONENT 元素定义在一个单一地址模块上的设备的最终位置:

```
<! ELEMENT COMPONENT (DESCRIPTION?,ADDRESS,POSITION) >
```

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

ADDRESS 元素和以前为识别模块定义的元素相同。POSITION 保留定义位置的数据,以便你设置模块。对器具模块来说,它可以是 ON 或 OFF;对灯模块来说,它可以是 ON、OFF 或者由百分比定义的暗淡位置。如果新模块出现在市场上,POSITION 元素可以随需要保留新的值,因为只有处理应用程序解释这些元素的含义。

告诉系统移到这些状态中的某种状态是比较复杂的,因为用户可能有不同的理由选择某一特殊的状态。定使器控制的许多家庭的自动化,使用这些模块。灯的打开和关闭由每天的时间决定。当主人不在家时,这会使房子看起来好象有人住的样子。它也可以确保灯只在主人下班回家后才打开。用户也需要“翻阅”灯的开关(特别是有线的),选择状态的功能。运动探测器和遥控台也可以选择状态。这需要

把 TRIGGER 元素建立的有点技巧。

```
<! ELEMENT TRIGGER (STATENAME, TIMED *, SWITCH *) >  
<! ELEMENT STATENAME (#PCDATA) >
```

STATENAME 只是以前定义的 STATE 元素的一个参考。TIME 元素定义时间执行所选的 STATE。因为用户可能需要各种定时机构,这里有几种选择,用做每天和每周的事件,以及发生在某一特定日子的事件:

```
<! ELEMENT TIMED ((DAILY | WEEKLY | DATE), TIME) >  
<! ELEMENT DAILY EMPTY >  
<! ELEMENT WEEKLY (WEEKDAY *) >  
<! ELEMENT WEEKDAY (#PCDATA) >  
<! ELEMENT DATE (DAY, MONTH, YEAR) >  
<! ELEMENT DAY (#PCDATA) >  
<! ELEMENT MONTH (#PCDATA) >  
<! ELEMENT YEAR (#PCDATA) >  
<! ELEMENT TIME (HOUR, MINUTE, SECOND?) >  
<! ELEMENT HOUR (#PCDATA) >  
<! ELEMENT MINUTE (#PCDATA) >  
<! ELEMENT SECOND (#PCDATA) >
```

象在以前章节中提到的,许多可得到的日期格式不需要分为天、月和年等等。对于像这样的应用程序,你可以参考那些格式。

SWITCH 元素包含描述——处理应用程序可以理解的名字或地址,而触发状态的开关执行处理应用程序:

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

处理文档的应用程序必需解析文档,并建立内部定时器以及一系列到模块和开关的连接。当任何的触发条件满足时,它将给控制模块发送适当的命令,把灯和其它设备设置到它们合适的位置。

在这些系统中,大多数墙壁和遥控开关是硬件编码的特殊模块。只需要连接一个开关到一个设备的用户,可以直接调用模块,旁路掉处理系统。以后调用模块的系统将不考虑以前发布的命令。当然如果有人把灯关闭,或者没有把插头插上,那么任何事都不会发生。

装配我们的 DTD 建立以下程序:

```
<! ELEMENT CONTROLSCHEDULE (MODULE *, STATE *, TRIGGER *) >
```

```

<! ELEMENT MODULE (ADDRESS, TYPE?, DESCRIPTION?)>
<! ELEMENT ADDRESS (HOUSE, UNIT)>
<! ELEMENT HOUSE (#PCDATA)>
<! ELEMENT UNIT (#PCDATA)>
<! ELEMENT TYPE (#PCDATA)>
<! ELEMENT DESCRIPTION (#PCDATA)>
<! ELEMENT STATE (NAME, DESCRIPTION?, COMPONENT+)>
<! ELEMENT COMPONENT (DESCRIPTION?, ADDRESS, POSITION)>
<! ELEMENT POSITION (#PCDATA)>
<! ELEMENT TRIGGER (STATENAME, TIMED*, SWITCH*)>
<! ELEMENT STATENAME (#PCDATA)>
<! ELEMENT TIMED ((DAILY | WEEKLY | DATE), TIME)>
<! ELEMENT DAILY EMPTY>
<! ELEMENT WEEKLY (WEEKDAY*)>
<! ELEMENT WEEKDAY (#PCDATA)>
<! ELEMENT DATE (DAY, MONTH, YEAR)>
<! ELEMENT DAY (#PCDATA)>
<! ELEMENT MONTH (#PCDATA)>
<! ELEMENT YEAR (#PCDATA)>
<! ELEMENT TIME (HOUR, MINUTE, SECOND?)>
<! ELEMENT HOUR (#PCDATA)>
<! ELEMENT MINUTE (#PCDATA)>
<! ELEMENT SECOND (#PCDATA)>
<! ELEMENT SWITCH (#PCDATA)>

```

我们可以用这个 DTD 创建处理应用程序用来控制一套灯和一台收音机:

```

<? xml version="1.0" encoding="UTF-8"? >
<! DOCTYPE CONTROLSCHEDULE SYSTEM "controller.dtd">
<CONTROLSCHEDULE>
<MODULE>
<ADDRESS><HOUSE>B</HOUSE><UNIT>2</UNIT></ADDRESS>
<TYPE>APPLIANCE</TYPE>
<DESCRIPTION>Radio in livingroom</DESCRIPTION>
</MODULE>
<MODULE>
<ADDRESS><HOUSE>B</HOUSE><UNIT>10</UNIT></ADDRESS>
<TYPE>LAMP</TYPE>
<DESCRIPTION>Lamp in entryway</DESCRIPTION>
</MODULE>
<MODULE>
<ADDRESS><HOUSE>B</HOUSE><UNIT>10</UNIT></ADDRESS>
<TYPE>LAMP</TYPE>

```

```

<DESCRIPTION>Light outside front door< /DESCRIPTION>
< /MODULE>
<MODULE>
<ADDRESS><HOUSE>B< /HOUSE><UNIT>11< /UNIT>< /ADDRESS>
<TYPE>LAMP< /TYPE>
<DESCRIPTION>Lamp in livingroom< /DESCRIPTION>
< /MODULE>
<STATE>
<NAME>AFTWORK< /NAME><DESCRIPTION>Come home to a friendly house.< /DE-
SCRIPTION>
<COMPONENT>
<DESCRIPTION>Turn the radio on< /DESCRIPTION>
<ADDRESS><HOUSE>B< /HOUSE><UNIT>2< /UNIT>< /ADDRESS>
<POSITION>ON< /POSITION>
< /COMPONENT>
<COMPONENT>
<DESCRIPTION>Turn on the living room light< /DESCRIPTION>
<ADDRESS><HOUSE>B< /HOUSE><UNIT>11< /UNIT>< /ADDRESS>
<POSITION>ON< /POSITION>
< /COMPONENT>
< /STATE>
<STATE>
<NAME>AFTDINNER< /NAME><DESCRIPTION>turn on front light, dim lights< /DESCRIP-
TION>
<COMPONENT>
<DESCRIPTION>Turn on porch, front lights< /DESCRIPTION>
<ADDRESS><HOUSE>B< /HOUSE><UNIT>10< /UNIT>< /ADDRESS>
<POSITION>ON< /POSITION>
< /COMPONENT>
<COMPONENT>
<DESCRIPTION>Dim livingroom< /DESCRIPTION>
<ADDRESS><HOUSE>B< /HOUSE><UNIT>10< /UNIT>< /ADDRESS>
<POSITION>80%< /POSITION>
< /COMPONENT>
< /STATE>
<STATE>
<NAME>POWERSAVE< /NAME><DESCRIPTION>Dim front lights< /DESCRIPTION>
<ADDRESS><HOUSE>B< /HOUSE><UNIT>10< /UNIT>< /ADDRESS>
<POSITION>40%< /POSITION>
< /STATE>
<STATE>
<NAME>MORNING< /NAME><DESCRIPTION>Turn off front lights< /DESCRIPTION>
<ADDRESS><HOUSE>B< /HOUSE><UNIT>10< /UNIT>< /ADDRESS>
<POSITION>OFF< /POSITION>

```



```

< /STATE>
< TRIGGER>
< STATENAME>AFTWORK< /STATENAME>
< TIMED>
< WEEKLY> < WEEKDAY> MON< /WEEKDAY> < WEEKDAY> TUES< /WEEKDAY> <
WEEKDAY> WED< /WEEKDAY> < WEEKDAY> THURS< /WEEKDAY> < WEEKDAY> FRI< /
WEEKDAY> < /WEEKLY>
< TIME> < HOUR> 17< /HOUR> < MINUTE> 00< /MINUTE> < /TIME>
< /TIMED>
< /TRIGGER>
< TRIGGER>
< STATENAME>AFTDINNER< /STATENAME>
< TIMED>
< DAILY /> < TIME> < HOUR> 20< /HOUR> < MINUTE> 00< /MINUTE> < /TIME>
< /TIMED>
< /TRIGGER>
< TRIGGER>
< STATENAME>POWERSAVE< /STATENAME>
< TIMED>
< DAILY /> < TIME> < HOUR> 23< /HOUR> < MINUTE> 00< /MINUTE> < /TIME>
< /TIMED>
< /TRIGGER>
< TRIGGER>
< STATENAME>MORNING< /STATENAME>
< TIMED>
< DAILY /> < TIME> < HOUR> 7< /HOUR> < MINUTE> 00< /MINUTE> < /TIME>
< /TIMED>
< /TRIGGER>
< TRIGGER>
< STATENAME>AFTWORK< /STATENAME>
< SWITCH>RMT01 - ON< /SWITCH> < ! -Remote Control in case we get home early ->
< /TRIGGER>
< TRIGGER>
< STATENAME>MORNING< /STATENAME>
< SWITCH>LT01 - OFF< /SWITCH> < ! -If you don't want to leave lights on all night ->
< /TRIGGER>
< /CONTROLSCHEDULE>

```

这个文档告诉控制器有几个模块控制一些灯和一台收音机。在下午 5 点,或者当有人按遥控器按钮时,起居室的灯和收音机打开。在下午 8 点,起居室的灯变暗点(可能是为了更好地看电视)且前面走廊的灯打开,欢迎来访的客人或者吓走小偷。在晚上 11 点,前面的灯变暗,可以节省电费。早晨时(或者当有人关闭开关),走廊灯就会熄灭。

这是为了一个小结果的一个十分精确的练习。这个例子的真正动力是在这种情况下包含的更广的自动控制。虽然象这样直接编写 XML 很乏味且容易出错,但是它可以用来做 GUI 应用程序产生的控制数据的文件格式。象这样的文件格式可以容易地把为特殊控制程序编写的程序,转换到不同的程序,且不丢失所有的逻辑。因为信息是以一系列状态而不是直接的命令显示,它不关心这些状态下面的结构。这些文件与 X-10 系统兼容,使用无线电频率的不同系统,为灯夫打印出说明卡的手工系统,控制数百或数千电灯的大系统。XML 容易解析的结构使它成为各种不同大小的系统之间数据交换的可靠工具。

XML 显然不是编程语言,也不是运送数据的工具。数据流是一条通道的状况最适合 XML 应用程序,但是更复杂的形式产生一些例外甚至错误,需要更加丰富的命令集以更加交互式的方式传送。XML 可以在网络的不同节点上的应用程序之间传输数据,所有的节点可以发送和接收 XML 响应,但是核心逻辑必须保留在处理应用程序中。XML 只提供存储和传输数据的结构方法。

用工具标记语言控制工具

工具标记语言 (IML) 和 IML 的执行,天文学工具标记语言 (AIML),都是由 NASA's Goddard Space Flight Center and Century Computing 和 AppNet 开发的项目。这两个标记语言采用以前描述例子的基本概念,并把它们应用到更困难的(更有用的)状况。天文学家经常处理在遥远的,环境恶劣的位置上的设备,从远离城市光污染的天文望远镜到处在不适合人定居地区的卫星和太空飞船观测站。AIML 和 IML 用 XML 作为描述控制它们的工具和命令的集装箱,支持天文学家和工程师访问遥控任何地方的工具。AIML's 的第一个项目将提供在 SOFIA 上支持工具,它用于波音 747 的改进,携带一个 2.5 米用于观察的反射望远镜。

交叉参考



AIML 和 IML 都仍在开发中,如需要最新有关 AIML 信息,可访问站点:<http://pioneer.gsfc.nasa.gov/public/aiml/>。如需要最新有关 IML 的信息,可访问站点:<http://pioneer.gsfc.nasa.gov/public/iml/>。

IML 是描述和控制工具的十分通用的标记语言,而 AIML 是为天文学的需要而客户化的语言。IML 和 AIML 文档被解析后流到 Java 应用程序,它显示出控制、发送信息和来往于控制的命令界面。硬件设计者可以创建描述它们工具的文档,软件将配置它的显示和控制反映它的描述。这大大地简化了集成硬件和软件的任务。理

解 AIML 词表的通用软件可以显示有关工具的详细信息,不用直接依赖工具特定的代码。下面的分段描述 HAWC 的冷藏单元,在使用 XML 语法的工具上,它提供两个端口(一个用于输出命令,一个用于输入数据)。

```

<Instrument id="ADR"> <! - subsystem ->
  <Port name="ADR" function="command" number="2201" type="ASCII" serverPort="false"
  >

    <Command name="HouseKeeping" >
      <Argument name="tag" type="java.lang.String" required="true" hidden="
      true" />
      <Argument name="Command" type="java.lang.String" required="true" hidden="
      true" />
      <Argument name="RATE" type="java.lang.Integer" required="true" >
        <ValidRange low="0" high="120000" />
      </Argument>
    </Command>

    <!-- Note the special XML decimal-like encoding for NEWLINE terminator -->
    <RecordFormat name="HouseKeeping" size=".1" ordered="true" terminator=" & # 10;" at-
    tributeSeparator=" ">
      <Format name="tag" format="%s" size="16" ordered="true" />
      <Format name="Command" format="HOUSEKEEPING" size=".1" />
      <Format name="RATE" format="%s" ordered="false" header="RATE=" />
    </RecordFormat>
  </Port>
  <Port function="data" name="ADR" number="2200" type="BINARY" serverPort="false"
  >

    <Telemetry name="Status" >
      <Field name="tag" type="java.lang.String" required="true" />
      <Field name="Time" type="java.lang.Integer" required="true" />
      <ArrayField name="Temperatures" required="true" dimensions="10">
        <Field name="dataElement" type="java.lang.Float" required="true" /
        >
      </ArrayField>
      <Field name="Heat Switch" type="java.lang.Integer" required="true" />
    </Telemetry>

    <RecordFormat name="Status" size="64" ordered="true" >
      <Format name="tag" format="%s" size="16" ordered="true" />
      <Format name="Time" format="%d" size="4" ordered="true" />
      <ArrayFormat name="Temperatures" size="40" ordered="true" >

```

```
<Format name="dataElement" format="%f" size="4" ordered="true" />
< /ArrayFormat>
<Format name="Heat Switch" format="%d" size="4" ordered="true" />
< /RecordFormat>
< /Port>
< /Instrument>
```

提示

▲ 注意使用 Java 类型识别从工具发出和返回信息的格式。XML 在得到它自己的类型词表之前,希望看许多其它词表。信息将配置用于控制工具的软件。如果工具得到另一个端口,或者许多命令可用于控制工具,硬件工程师可以简单地通过改变这个文件,来加入一个界面。下一次,更新的文件装入可显示到工具界面的 Java 程序,界面将反映新的信息。做一点快速变化,使得在重复基础上运行项目更加容易。从基础开始,然后一小步一小步地前进,而不是一开始就设计一个全新的版本,作巨大的改变。

对象文档

XML 嵌套的结构与出现在对象定位设计数据结构中的层次数据,具有很强的共同之处。XML 和对象定位设计适合很好,因为两个系统都是在数据表中的数据表中存储数据表。存储包含在对象结构中的信息是困难的,因为大多数用做文档的,线性和制成表的文件类型,不太适合这种层次结构。本节将不产生任何特殊的 DTD,因为从一个程序到另一个程序变化很大。我们将检查少数通用的项目例子,它们可以帮助开发者创建补充的对象和文件结构。

Bean 标记语言 (BeanML)

Bean 标记语言由 IBM's AlphaWorks(<http://www.alphaWorks.ibm.com/formula/bml>) 开发,读起来很象命令语言,使用一套简单的标签和属性例示 Java Beans,并在它们中加入特性和事件处理。BeanML 不真正创建新的对象,它只使你能够描述 Java Beans 的特性,并用简单的 XML 语法连结它们。BeanML 文档使用 Bean 元素作为它们的根元素,然后用子段加入 args、bean 以及 cast 等元素。

一个简单的 BeanML 文档如下所示:

```

<? xml version="1.0" encoding="UTF-8"? >
<bean class="java.awt.Panel" id="mainPanel">
  <add>
    <bean class="java.awt.Label">
      <property name="text" value="This is a label" />
    </bean>
  </add>
</bean>

```

这里用 Java AWT 库创建 Panel,它包含“这是一个标签”的标签。你可以使用任何带有 Java Beans 的 BeanML,Java Beans 为创建复杂的在 Beans 范围外的应用程序提供 XML 词表,而不需要编写扩展和重复的 Java 代码。BeanML 文档甚至可以编译,当程序开始时,消除了由于解析 XML 而引起的延迟。

注意

BeanML 象 alphaWorks 中的大多数东西,仍在进展中。上述的材料有可能变化,并不是准确的。如需要最新的信息可访问站点:<http://www.alphaWorks.ibm.com/>。

MDSAX 和 Coins

MDSAX 和 Coins 是以 Java 为基础的开放资源项目,它可以帮助开发者处理 XML,使 XML 对 Java 开发者作为连续化 Java 对象,并立刻有用。MDSAX 是一个结构,它帮助开发者创建一系列 XML 文档处理器,而 Coins 是一套把存在 XML 文档中的数据流入到 Java 对象并重新收回的工具。二者都使用 XML 文档作为控制 Java 应用程序的主要工具,节省了开发者在编码-编译-调试中的许多重复工作,在许多例子中编码是重复的循环。开发者可以定义 MDSAX 如何使用 ContextML(可参考 Java 种类,它是典型的处理 XML 文档的滤波器)处理 XML 文档。当程序建立 MDSAX 时,它把它传到 ContextML 文档,ContextML 文档告诉它如何建立处理子目录结构。简单的 ContextML 文档如下所示:

```

<context>
  <documentRouter>
    <elementRouter key="context">
      <x key="context"
factory="com.xml.mdsax.MDContextFactoryImpl" />
    <x key="documentRouter"
factory="com.xml.mdsax.MDDocumentRouterFactory" />

```

```
        <x key="stack"
factory="com.xml.mdsax.MDFilterStackFactoryImpl" />
        <x key="elementRouter"
factory="com.xml.mdsax.MDElementRouterFactory" />
        <x key="trace"
factory="com.xml.mdsax.debug.MDTraceFilterFactory" />
        <x key="display"
factory="com.xml.mdsax.debug.MDDisplayFilterFactory" />
        <x key="results"
factory="com.xml.mdsax.debug.examples.counter.ResultsFactory" />
        <x key="element"
factory="com.xml.mdsax.debug.examples.counter.ElementFactory" />
    </elementRouter>
</documentRouter>
</context>
```

关键词通过这个文档连接到 Java 类型。在最初启动过程(也可以定义缺少文档)后,用于描述这些类型是如何安排的 ContextML 文档,可以以很简单的形式出现。见计数器中的 ContextML 文档。

和响应通过 HTTP 协议传输到相应的调用。调用作成 XML 文档,可以使用 POST HTTP 请求来发送。标题告诉接收的 Web 服务器在那里指导 RPC 调用。例如, XML-RPC 请求如下所示:

```
POST /rpchandler HTTP /1.0
User-Agent: MyClient /1.0 (WinNT)
Host: mycomputer.simonsil.com
Content-Type: text /xml
Content-length: 169

<? xml version="1.0" encoding="UTF-8"? >
<methodCall>
  <methodName>test.returnId< /methodName>
  <params>
    <param>
      <value><string>ASCI< /string>< /value>
    < /param>
  < /params>
< /methodCall>
```

接收这个请求的 Web 服务器,把请求传送到 /rpchandler,它可以是一个 servlet、CGI 脚本或者是一个在 URL 中应答的程序。它接收 XML,调用名为带有字符串“UTF-8”的“test.returnID”文件,然后用它自己的 XML-RPC 文档响应,它的形式如下所示:

```
HTTP /1.1 200 OK
Server: JavaWebServer /1.1
Content-Length:
Content-Type: text /xml
Date: Fri, 12 Feb 1999 19:01:32 GMT
<? xml version="1.0? >
<methodResponse>
  <params>
    <param>
      <value><string>Server 12345< /string>< /value>
    < /param>
  < /params>
< /methodResponse>
```

到达服务器识别是一个很长的路程,但是 XML-RPC 可以提供很多东西,特别是在跨平台合作。Unix 服务器、Windows 95、Macintoshes、System /390 主机、其它网络系统以及可以用这些系统通过网络连接起来,共享网络上的透明处理。它不管计算


```
<LOGO HREF="http://www.simonstl.com/craneico.gif" STYLE="ICON" />
<LOGO HREF="http://www.simonstl.com/logo.gif" STYLE="IMAGE" />

<SCHEDULE>
<INTERVALTIME DAY="14">
</SCHEDULE>

<ITEM HREF="http://www.simonstl.com/articles/index.html">
<TITLE>Articles</TITLE>
<ABSTRACT>Articles on XML</ABSTRACT>
</ITEM>

<ITEM HREF="http://www.simonstl.com/projects/index.html">
<TITLE>Projects</TITLE>
<ABSTRACT>Projects, including open-source software development</ABSTRACT>
</ITEM>

<ITEM HREF="http://www.simonstl.com/xmllinks.html">
<TITLE>XML Links</TITLE>
<ABSTRACT>Links to XML Resources, from specifications to news sites to mailing lists. </ABSTRACT>
</ITEM>

</CHANNEL>
```

通道内容仍然在 HTML 中,而不是在 XML 中。XML 只提供使浏览器找到和描述内容的结构。用户可以通过通道栏查看内容,甚至(只要浏览器已经收集)没有连接的时候。这个方案对使用拨号上网的计算机有不同寻常的影响;因为大多数方案下载数据的时间避开了线路繁忙高峰期(下载时间定为深夜到凌晨 4 点),Internet Explorer 4.0 用户可以在深夜醒来欢呼他们的调制解调器连通了 ISP(因特网服务商)。

交叉参考



你可以从几个不同的来源,访问 CDF 的信息。请浏览“The submission to W3C”,它包括 DTD 的完整描述,它的站点为:<http://www.w3.org/TR/NOTECDF-submit.html>。微软的白皮书和其它信息可以在站点 Builder(<http://www.microsoft.com/sitebuilder>)以及 Internet Explorer(<http://www.microsoft.com/ie>)Web 站点得到。

XML 软件自动更新(XSA)

XSA 使得软件设计者能够为他们的软件创建描述文件,这就为站点检查这些文件并记录它们的更新情况成为可能。XSA 是由 Lars Marius Garshol 创建,他是 XML 工具(<http://www.stud.ifi.uio.no/~larsga/linker/XMLtools.html>)列表的管理员,帮助过他的人是 James Tauber(<http://www.xmlsoftware.com> 站点管理员)和 Robin Cover(<http://www.oasis-open.org/~cover/> 站点管理员)。在不同开发阶段免费软件工具的展览会标志着 XML 早期开发时期。开放源程序开发和采取早发布和经常发布的策略,使 XML 站点管理员很难赶上它们的变化,XSA 寻求解决他们的困难,同时帮助开发者宣布新的软件。XSA 文件十分简单。XSA 文档向导仍就在开发中,但是用手工创建 XSA 文件也是相当容易的。XSA 元素包含自动售货机和多种其它产品。自动售货机元素包含名字、电子邮件和 URL 信息,而产品包含 ID、版本信息、上次发布的日期、产品的 URL 信息和与上一个版本不同的变化信息(XSA 不提供描述多版本的产品,虽然它可以存储多样的产品)。例如,XSA 文件描述作者的 XlinkFilter 可能如下所示:

```
<? xml version="1.0" encoding="UTF-8"? >
<xsa>
  <vendor>
    <name>Simon St. Laurent< /name>
    <email>simontl@simonstl.com< /email>
    <url>http://www.simonstl.com< /url>
  < /vendor>
  <product id="xlinkfilter">
    <version>0.20< /version>
    <last-release>19981227< /last-release>
    <info-url>http://purl.oclc.org/NET/xlinkfilter< /info-url>
    <changes>XLinkFilter 0.20 includes an image map demo, more documentation,
and support for generating XPointers through a LocationFilter.< /changes>
  < /product>
< /xsa>
```

XSA 文件必须放置在公共地点(可能的话在商家的 Web 站点上)注册站点,它可以定期检查,看是否有信息变化,并且需要更新。

交叉参考



如需要更多有关 XSA, 包括 DTD、文档和案例的信息,可访问站点:<http://birk105.studby.uio.no/~www-work/xsa/>。

资源描述框架(RDF)

W3C 资源描述框架开始的时间和 XML 大致相同,并且用不同的办法解决了许多相似的问题。RDF 是表现元数据最通用的工具。RDF 提供描述实体和实体之间关系的工具,而 XML 基础来自 SGML 项目文档表示。RDF 使用 XML 作为它的“序列化语法”,但是它不使用 XML DTD。从 RDF 观点看,属性和带有文本内容的子元素是相同的,导致访问信息的方法十分不同。在理论上,你可以使用包含几乎任何种类信息的 RDF,这些信息可以表现为指导图表或一套图表,从有关资源的简单信息,到描述矩阵内容的复杂关系,甚至有可能是文档。然而,XML 是文档语法,而 RDF 用做元数据,为那些文档提供描述。

RDF 在许多 W3C 计划中,作为主要单元执行,包括 Digital Signature Initiative (DSIG)、Platform for Internet Content Selection(PICS)和 Platform for Privacy Preferences(P3P)。RDF 为指定结构和 RDF 材料内容提供它自己的工具。RDF 自己是无用的,象 XML 自己一样。在有人创建词表前,RDF 正式结构看起来比 XML 的结构更复杂,主要是因为 RDF 有更灵活的语法。RDF 的例子将在下面出现。在关于 Dublin Core 这节,使用预先定义好的词表,但是 RDF 的所有含义确实值得有它自己的书,能写出三到四本来。

RDF 处理和 XML 处理之间互相作用,毫无疑问地会增大,但是现在应用程序还没有广泛使用。

交叉参考



有关 RDF 下的图表理论指南,可访问站点:<http://www.utm.edu/departments/math/graph/>。需要有关创建 RDF 文档的优秀(免费)工具,可访问站点:<http://metadata.net/dstc/>。W3C 提供 RDF 有效性服务可在下列站点得到:<http://jigsaw.w3.org:8000/>。

Dublin Core

The Dublin Core Metadata Initiative 在 XML 和 RDF 出现很早以前,便用在 W3C 雷达屏幕上,并提供描述 Web 资源的词表。Dublin Core 词表是以长期存在图书馆卡片上的目录为基础,这些目录帮助读者找到他们所需的信息。Dublin Core Metadata Element Set 最近以 RFC2413 发布,在这个项目中,肯定会出现更具描述力的工具。Dublin Core Metadata Element Set 只是一个词表,不依赖 XML 和 RDF。RFC 不包含 DTD 或 RFC 方案。但是 Dublin Core 信息完美地补充了 RDF,给它一个你可以在各

种 Web 文档上立刻应用的词表。Dublin Core 使用元素 Title、Subject、Description、Type、Source、Relation、Creator、Publisher、Contributor、Right、Date、Format、Identifier 和 Language。并不是每一个文档都需要所有的元素,虽然信息越明确,它就越有用。如果这些标题感觉像图书馆卡片目录卡上的信息,那是因为 Dublin Core 组主要是由图书管理员和信息管理的成功者组成。充分利用这个机会,创建比今天的以严密定义的标准为基础的全文本搜索引擎更可靠的寻找信息的目录。以下是一个样本 Dublin Core 文档,它描述作者的 Web 站点:

```
<? xml version = "1.0"? >
<rdf:RDF xmlns:rdf = "http://www.w3.org/TR/WD-rdf-syntax"
        xmlns:DC = "http://info.internet.isi.edu/in-notes/rfc/files/rfc2413.txt">
<rdf:Description xml:lang="en" about="http://www.simonstl.com">
  <DC:Title>
    Simon St. Laurent's Web Site - Articles, Projects, and Books
  </DC:Title>
  <DC:Creator>
    Simon St. Laurent
  </DC:Creator>
  <DC:Description>
    This Web site contains pointers to XML information, as well as articles by Simon St. Laurent
    and links to books he has written.
  </DC:Description>
  <DC:Subject>
    XML XLink XLinkFilter DDMML XSchema CSS
  </DC:Subject>
  <DC:Publisher>
    simonstl.com
  </DC:Publisher>
  <DC:Identifier DC:Scheme="URI">
    http://www.simonstl.com
  </DC:Identifier>
  <DC:Rights>
    Copyright 1998-9 by Simon St. Laurent
  </DC:Rights>
</rdf:Description>
</rdf:RDF>
```

我希望用于 Subject 区的标准词表将很快到来,幸运的话,Dublin Core 项目和它的词表将使 Web 变成更容易搜索和更有用的地方。

交叉参考



如需要更多有关 Dublin Core Metadata Initiative 的信息,可访问站点:<http://purl.org/dc/>。对于描述 Dublin Core Metadata Element Set 的 RFC2413,可访问站点:<http://info.internet.isi.edu/in-notes/rfc/files/rfc2413.txt>。

未来

XML 在数据应用程序中能否流行,很大程度上依靠开发者发掘如何把结构、灵活性、机器可读性和人可读性结合在一起。不象以前章节中的大多数文档,这些应用程序不是直接为人使用。可读性使得文档容易检查和调试,但是很少有文档能看见样式表或者直接面向人类读者。即使 XML 确实使成本比传统的二进制文件增加,在嵌套结构上的冗长和重点信息就给程序员提供了计算机和应用程序之间通讯的新工具。随着 XML 的扩大传播,我们将看到更多的程序会以共享体系结构为基础,所有的文档都使用一套通用文件格式用于千差万别的项目。

第 十 章

Xpointer 规范说明

XML 整洁定义的结构使它有可能容易地描述文档部分,而不用丢掉带有标识符的文档。利用嵌套结构和元素名字潜在重要意义,XPointer 可以指定你提取文档中一部分内容的路径。当 XPointer 与 Xlink 说明结合在一起时,它就会成为在文档间精确指定链接的功能强大的工具。

注意

本章讨论将以 1998 年 3 月的“XPointer 指定”草稿为基础,你可以在下面站点得到有关信息:<http://www.w3.org/TR/1998/WD-xptr-19980303>。XPointer 和 Xlink 的工作进展很慢,虽然 1999 年有一些好的迹象。工作组将在 1999 年 2 月发布所需文档。如需要最新信息,可查看 W3C XML 站点:<http://www.w3.org/XML/>。XSL 和 XPointer 被认为是把一些项合为一体,因此查看 XSL 站点(<http://www.w3.org/Style/XSL/>)也是有用的。

通过查询指向

XPointer 在搜索文档,并把文档分段的信息子集返回上有强大的功能。然而,XPointer 不是真正的查询语言(它们更像一套公路图的指南),它们有很多相同的功能,即优化文档导航。XPointer 把着眼点放在位置而不是内容上,然而它不是真的指向 XPointer 看起来像的结果。它们典型地以容易处理的树型结构出现(像树的子集,由 W3C 文档对象模型或 DOM 创建),但是在有些情况下,它们可以包含元素部分,甚至属性。

W3C 在 1998 年 12 月拥有了 Query Language Workshop,94 位参加者在那里讨论了查询 Web 的 66 篇不同定位的论文,大多数是关于 XML 的(你可以访问定位论文的列表,<http://www.w3.org/TandS/QL/QL98/pp.html>)。各类开发者和组织正在创建

查询语言,一些是以使用关系数据库的结构化查询语言(SQL)为基础,一些以面向对象数据库的查询工具为基础,还有一些以设置声明为基础,无疑还有许多别的类型。用于查询 XML 的工具有可能在几年后出现重要的题材,但是(除了 XPointer)现在还看不到任何别的迹象。

XPointer 介绍

XPointer(Extended Pointer 的缩写)起源于在第四章描述过的文本编码主动性(TEI)标准。XPointer 用组成定位器的定位项指定资源。XPointer 可以包含单一的定位器(虽然一些定位器可以包含其它的定位器)。我们先建立简单的定位器,然后再把它们联合起来。定位器可以包含绝对、相对、范围和字符串匹配定位项。字符串匹配项有最多的受限制的词表,但是它们提供的精确程度是其它项不能相比的。属性项使得它可以检索属性值以及元素内容。范围项参考了两个定位器之间的信息。相对项使链接能够参考文档的内容,绝对定位项用更传统的 ID 和 NAME 编址方案以及其它基本定位来识别元素。

绝对定位项

绝对定位项提供对少数关键 XML 文档块的访问。`root()`、`origin()`、`id(id)` 和 `html(name)` 关键词,使它容易为一些元素编址,这些元素是:文档的根元素、使用关键词的元素、带有 ID 的元素和标准 HTML A 带有 NAME 属性值的元素。绝对定位项必须出现在定位器的开始部分;如果你不指定绝对定位项,`root()` 将被指定为缺省绝对定位项。

`Root()` 项指定文档的根元素,它是文档树最外面的元素。`Root()` 有效地访问整个文档的内容。`Origin()` 关键词访问链接元素本身,并经常为后来的相对项提供绝对位置。这就使得 XPointer 能够指定内容,例如象“第二段放到链接元素下面”。这样便需要跟在 `root()` 和 `origin()` 关键词后的空插入语。

其它两个绝对关键词访问元素,由作者给它们分配名字,即可用于 HTML,又可用于 XML。`html(name)` 关键词采用匹配 HTML A 元素的 NAME 属性,它提供的服务和在 HTML 中标识符为分段提供的服务完全一样。最主要的是,这将习惯于访问 HTML 文档的 XML 文档,因为 XML 文档使用 ID 属性。`id(name)` 关键词提供类似的且更好的功能。文档中的每一个元素都有 ID 值,这意味着你可以用这种方法快速访问任何数据。缺省情况将在下一章描述,XLink 将处理所有的没有按照这样标记的段标识符。段标识符“#fragmentidentifier”将自动以 `id(fragmentidentifier)` 处理。

相对定位项

相对定位项更灵活,但也更复杂。相对定位项的关键词是导航文档树的工具,它的列表在表格 10-1。

表格 10-1 相对定位项

child	选择定位资源子段元素(必需是直接资源下的嵌套的元素)
descendant	选择在定位资源内容中出现的元素(可以嵌套多个级)
ancestor	选择它的内容在定位资源中被发现的元素(父元素)
preceding	选择在定位资源前出现的元素
psibling	选择在定位资源以前出现的同属元素(同属元素有相同的父元素)
following	选择出现在定位资源后的元素
fsibling	选择在定位资源后出现的同属元素(同属元素有相同的父元素)

所有的相关关键词都使用一套相同的参数,用圆括号封装:

(Instance, NodeType, Attribute, Value)

你必须包括 Instance(它是数值,或“所有”)和 NodeType,Attribute 和 Value 是可选择的,只有当需要用属性值来识别元素时,才使用这两个元素。Instance 使开发者能够在相对定位资源的位置中指定元素。NodeType 定义定位项的候补类型。

Instance 和 NodeType 使开发者可以指定,用相对定位关键词描述的结构中某一节点,第 n 个的外观(版面)。这可以使它容易地用在文档中的位置指定相对位置。Instance 的值即可以是正整数,也可以为负整数(例如,2、-12、+4),或者是关键词“all”。使用 all 做 instance 的值,它将返回到所有满足标准的节点,这个标准是由剩余的定位项指定的。而数字表示沿着文档树移动的方式,这个方式由定位项指定。正整数使计数从符合定位项的第一个结构外观算起,而负整数表示计数从最后一个符合定位项的结构外观算起。例如,CHILD(3,QUOTE)表示的是定位资源元素中的 QUOTE 元素的第三个外观。CHILD(5,EXPLANATION)表示定位资源元素中的 EXPLANATION 元素的第五个外观。使用负数做为 instance 的值,计数的方向相反,例如,CHILD(-1,PRICE)代表在定位资源元素中的最后一个 PRICE 元素,而 CHILD(-3,PRICE)代表在定位资源元素中,从最后一个往前算起的第三个元素。

NodeType 典型地以元素名字出现。然而,开发者可能不会总知道,或者需要知道它们指定目标的名字。为了适应这种情况,XML 可以使 NodeType 采用不是元素

名的三个值。值 #element 使得定位项接受所有的元素作为匹配候选者,CHILD(2, #element)代表定位资源元素中第二个元素。#text 值告诉定位项接受文本节点作为匹配候选者。文本节点是由字符数据构成,它和混合声明中的标记合并。例如在:

```
<CASE>Name:<NAME>Jim</NAME>
Fish bonker:<BONKER>01234</BONKER>
Crime:<CRIME>Fishing without mowing the lawn first</CRIME>.
Status:<STATUS>Dismissed</STATUS></CASE>
```

第一个虚拟-元素(child(1, #text))包含文本“NAME:”,第二个元素(child(2, #text))包含新行字符和“Fish bonket:”,等等。

XPointer 也可以索引注释、处理指令和 CDATA 部分。#command 值索引命令,而 #pi 索引注释, #cdata 索引 CDATA 部分。最后, #all 表示 NadeType 最后一个可以得到的值。这个 NadeType 使元素和虚拟-元素都成为计数的候选者。如果应用到以前的例子,使用 CASE 元素作为定位资源,CHILD(1, #all)将索引“NAME:”,而 CHILD(2, #all)将索引 NAME 元素后面的元素。

在 Attribute 和 Value 中加入参数,执行属性匹配,可以进行另一级别的精确指定定位。Attribute 和 Value 都是可选择的。但是如果 Attribute 参数出现, Value 参数也必须跟随出现。CHILD(1, *, TARGET, ME)索引包含带有 ME 值的属性为 TARGET 的定位资源中的第一个元素。CHILD(2, QUOTE, SPEAKER, ROOSEVELT)索引把 SPEAKER 属性设置为 ROOSEVELT 的定位资源中的第二个 QUOTE 元素。这两个参数都接受 * 值代替指定参数。如果你用 * 代替 Attribute 参数,任何属性的参数值都将和它匹配。如果你用 * 作为 Value 的参数,由属性参数命名的属性可以是任何值。CHILD(1, QUOTE, SPEAKER, *)索引有 SPEAKER 属性声明的第一个 QUOTE 元素,不管它的值是什么。CHILD(3, QUOTE, *, ROOSEVELT)索引,值设置为 ROOSEVELT 的任何属性的第三个 QUOTE 元素。属性可以命名为 PRESIDENT、SPEAKER 或 Q2FD,这没有关系。

Value 参数也可以接受值 #IMOLIED。类似属性列表声明的缺省类型,它是索引已声明的,但没有分配缺省值的属性。在这种情况下,它只索引文档中已声明但还没有指定的属性。如果元素给属性提供了一个值,即使只有一个声明 #IMPLIED,使用 #IMPLIED 作为值的 XPointer 将不再认可它。

属性定位项

属性定位项看起来可能比大多数其它项简单,虽然它使用的(至少在链接)仍旧有某种神秘。用来和识别元素的项结合,它只采用一个属性名作为选择器,并返回属

性值。这在某种意义上是唯一的,因为其它的 XPointer 指定定位而不是对每一个服务进口返回值,但是你很快会认识到这样项的使用。它使用的语法如下:

```
attr(AttributeName)
```

例如,att(id)将把一个来自当前识别的元素的名为 id 的属性值返回(或者根元素,如果在它之前没有其它声明)。

范围定位项

Span() 关键词和 XPointer 合并可以收集从第一个 XPointer 开始到第二个 XPointer 结束的所有信息(还不清楚,如果第一个 XPointer 的索引定位出现在第二个参数之后,将会发生什么事情)。

Spanning 使它可以容易地产生与 XPointer 合并,简化产生用于各种任务的链接。Span() 的语法看起来象以下形式:

```
Span (XPointer1,XPointer2)
```

这两个 XPointer 可以分开单独解释;第一个 XPointer 的选择对第二个 XPointer 的选择没有任何影响。Span() 项然后选择所有的在第一个 XPointer 开始和第二个 XPointer 结束之间的内容。

```
<CASE>Name:<NAME>Jim</NAME>
Fish bonker:<BONKER>01234</BONKER>
Crime:<CRIME>Fishing without mowing the lawn first</CRIME>
Status:<STATUS>Dismissed</STATUS></CASE>
```

索引以上的文档,XPointer(span(child(NAME,1),child(BONKER,1)))识别以下分段:

```
<NAME>Jim</NAME>
Fish bonker:<BONKER>01234</BONKER>
```

类似,XPointer(span(child(NAME,1),child(#text,2)))识别以下分段:

```
<NAME>Jim</NAME>
Fish bonker:
```

字符串定位项

字符串定位项是最后一套定位项。字符串定位关键词使用以下语法:

string (Instance, String, Position, Length)

Instance 参数和它在相对项中工作的方式相同。String 参数只是匹配字符串(在搜索匹配字符串中,处理器将忽略所有的标记字符,以便字符串值可以通过元素边界)。Position 提供从识别定位返回的精确字符位置偏移量,从发现的字符串的第一个字符算起。如果 Position 是零(或出现差错),那么定位将是所发现的字符串的第一个字符;如果它是 2,它是开始后的第三个字符。Length 告诉处理器有多少字符,且只有当你使用位置时才出现(如果 Position 和 Length 不在适当的位置,定位以所发现的字符串为参考)。在代码,

```
<LINE>The worms crawl in and the worms crawl out < /LINE>
<LINE>The ones that crawl in are lean and thin< /LINE>
<LINE>The ones that crawl out are fat and stout< /LINE>
```

项 string (1, worms, 4, 1) 返回定位字母 s,它是第一行第一个单词的外观“worms”的字母 s。string (2, worms, 3, 1) 返回定位字母 m,它是单词第二个外观“worms”中的字母 m。string (1, crawl, 1, 1)、string (2, crawl, 1, 2)、string (3, crawl, 1, 3) 和 string (4, crawl, 1, 1, 4) 返回定位字母 c,它是单词“crawl”第一个外观字母 c,字母“cr”在第二个,“cra”在第三个,“craw”在第四个。

合并定位项

当定位项在合并中使用,定位项便产生很多功能。绝对定位项可以方便地指定新缺省定位器元素,代替缺省 root()。相对定位项可以组合,使你可以在第五个 FRAB 元素(它的 FLIPGELLY 属性设置为“postmaster”)外观上直接指定元素。合并项是简单的,只是把它们按顺序列表。项将从左到右对它们进行估算。例如,给出以下分段:

```
<BOOKSHELF ID="history">
  <BOOK><TITLE>The Shaping of America, Volume 2: Continental America, 1800-1867< /TITLE><AUTHOR>Meinig< /AUTHOR>< /BOOK>
  <BOOK><TITLE>That Noble Dream: The Pursuit of Objectivity in the American Historical Profession< /TITLE><AUTHOR>Novick< /AUTHOR>< /BOOK>
  <BOOK><TITLE>The Origins of the Korean War, Vol. II< /TITLE><AUTHOR>Cumings< /AUTHOR>< /BOOK>
  <BOOK><TITLE>Nature's Metropolis: Chicago and the Great West< /TITLE><AUTHOR>Cronon< /AUTHOR>< /BOOK>
< /BOOKSHELF>
```

定位器 id(history)选择整个 BOOKSHELF 元素。id(history).child(3,BOOK)选

择第三个 BOOK 元素。AUTHOR 元素,在这个例子“Cumings”中,被 `id(history).child(3,BOOK).child(1,AUTHOR)` 选择。`id(history).child(3,BOOK).psibling(1,BOOK)` 选择 BOOK 元素。最后第三和第四个书元素和它们的内容被 `span(id(history).child(3,BOOK), id(history).child(4,BOOK))` 选择。甚至当程序块不能整齐地匹配单个元素时,也能给开发者提供用于产生程序块的所有功能。

XML 使得用于多参数列表的简写符号(对它本身不平常,因为 XML 只允许很少的缩写),`Id(history).child(3,BOOK).(1,AUTHOR)` 等同于 `id(history).child(3,BOOK).child(1,AUTHOR)`;XML 理解的关键词可以在步骤之间重复。

注意



当 XPointer 使识别各种文档分段容易时,而要处理它们时,困难还会出现。一个单独的工作组正忙于这些问题,严密地计算出如何表示不同种类的文档分段。如需要更多的信息,可访问站点:<http://www.w3.org/XML/Activity.html>。

现在我们可以创建一些功能强大的索引,它超过允许开发者通过 HTML A 元素的命名功能。我们的索引可以通过文档结构精确地定位他们的目标。下一步把这些索引用新链接技术合并,进一步扩大工具箱,把链接提高到一个全新的水平。

注意



因为 XPointer 严重的不稳定性,所以没有很多的实现例子。如需要当前的列表,可查阅 Steve DeRose's “XPointer and Xlink Implementation Links”,它的站点是:<http://www.stg.brown.edu/~sjd/XML-Linking/xptr-implementations.html>。

第十一章

Xlink 规范说明

HTML 的爆炸性地增长和它的链接的关系可能比任何其它的单个因素都大。数百个人和团体同时使用超文本系统,一些甚至使用 SGML,但是他们之中没有一个具有 HTML 方便链接系统的简洁性。聚集了数百万的页面,并把它创建成 World Wide Web。当然还有许多可以改进的地方。超文本专家和其他开发者对这些有限的基本链接功能,非常不满。至少一些 HTML 开发者看到了 SGML 的更完善的 HyTime 定义,和它更强的功能,当然没有看中它的复杂性。XML 提供了把工作干的更好的机会,XML 工作组早期专著于链接。XML 在 HTML 上链接的成功,提供了更强的功能,然而也使工具变的更复杂。

注意



我以 1998 年 3 月的“Xlink specification”草稿为基础讨论。它的站点是:<http://www.w3.org/TR/1998/WP-xlink-19980303>。有关 Xlink(和 XPointer)的工作进展很慢,虽然在 1999 年可能会出现希望。工作组将在 1999 年 2 月发布需要的文档,你可访问 W3C XML 站点:<http://www.w3.org/XML/>了解最新信息。

简单的链接

在经过六年的广泛使用之后,HTML 链接系统由于只能提供最简单的链接而受到批评。相反,许多开发者似乎对当前的链接语法非常满意,小规模的开发工具和 Web 映射工具围绕这一主要标准发展起来。HTML 的 HREF 属性做的很好令大多数开发者满意。XML 没有违反以前的标准,它只是加到里面。事实上它加了很多东西,但是基本的 HTML 链接结构仍然保留,HTML 的某些方面引入了 XML 的标准,便于 XML 文档链接到 HTML。我们开始先检查 HTML 文档中可以链接的文档类型,然后看一看我们是如何在 XML 中实现它们。

HTML 中的链接

A 元素几乎是所有 HTML 链接的钥匙,当然 LINK 元素也起有限的作用。图 11-1 显示了一个简单的,用 A 元素创建的 HTML 链接。

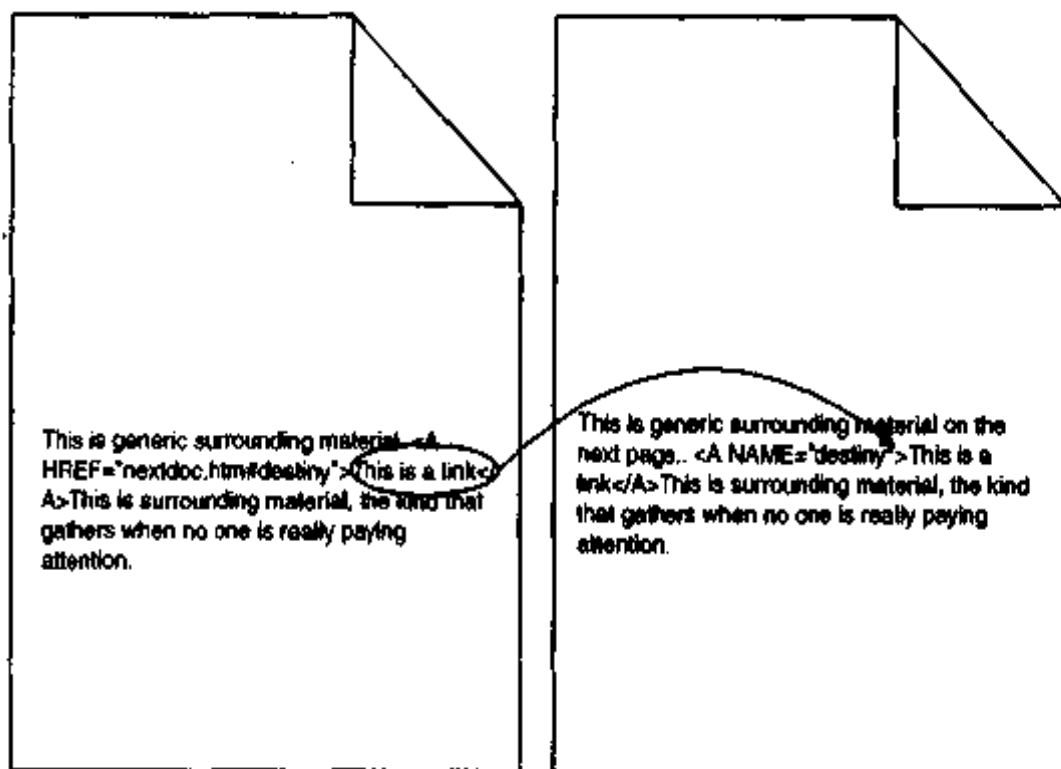


图 11-1 简单的非直接嵌入的链接,HTML

A 元素有几个属性,只有一个属性可以不断从开发者那里得到使用,它就是具有多功能的 HREF。HREF 通常采用 URL(统一资源定位器)作为它的值,它代表了链接目标。URL 可以是绝对地址也可以是相对地址。绝对 URL 以一个模式开始,它用来描述解释 URL 的协议。常用的模式包括 http:、ftp:、gopher:、mailto:、nntp:、file: 和 javascript:。在多数情况下,它将作为到服务器或者服务器上的文件的索引出现,它的前面带有两个斜线。例如一个使用 HTTP 协议的绝对 URL 执行以下语法:

```
http: //hostcomputer:port /path? query
```

端口和查询是可选项。主机必须包含有效 DNS 名字和 IP 地址,端口可选择指定主计算机上的端口号(80 是 http 的缺省值)。路径是指定到主计算机上的特殊文件的路径。只有数字、字母和 \$、-、+、!、*、'、(、) 及句号和逗号才能在路径中出现。其它的字符可以通过 % 标记换码,后面跟着它们的十六进制值。查询选项提供服务器的额外信息,使服务器能以适当的方式响应有关方式和信息。查询的内容限于和路径相同的字符。

注意

对于 javascript: 模式,冒号后面的值可以是任何有效的 javascript 代码。

Javascript:模式即将灭亡,HTML 4.0 提供 onclick 属性,使元素不用把 javascript 放到 HREF 的属性中就能激活脚本。

相对 URL 使用当前页面(如果它存在,URL 由页面的 HEAD 元素中的 BASE 元素设置)的 URL 作为到它们信息的前缀。相对 URL 不包括模式。

交叉参考



如查看在 RFC 1738 和 RFC 1808 中定义 URL 的完整的正式的语法,可访问站点:<http://www.w3.org/Addressing/rfc1738.txt> 和 <http://www.w3.org/Addressing/rfc188.txt>。

绝对 URL 和相对 URL 都可以在查询的末尾包含分段识别器。在 HTML 中的分段识别器包括一个圆括号和一个值(#),值是链接目标文档中 A 元素的 NAME 属性。例如在

```
<A HREF = "#laterlink">Skipping around is fun! </A>
...
...
<A NAME = "laterlink">Aren't you glad you skipped ahead?
```

点击“Skipping around is fun!”将屏幕移动到带有 NAME 属性“laterlink”的 A 元素指定的位置。当然你也可以使用与 URL 联合的分段识别器。

```
<A HREF = "zip.html#nothingness">
```

点击它,将把你带到包含的 zip.html 文件行。

注意



更新的说明,可参考 URI(统一资源识别器)而不是 URL(统一资源定位器)。UTI 为 URL 补充了识别资源的额外工具,象统一资源编号(URN)。然而,按照普通的用法,URI 提供的典型额外特点被忽略了,因为它们需要额外的工具解决他们的内容定位。虽然你也可以使用带有 Xlink 的 URI,本章我们使用更普通的 URL 术语。

A 元素也允许 REV 和 REL 标签用来表示锚点 URL 和目标 URL 之间的关系。REL 表示 URL 到锚点的关系(沿着链接向前移动),而 REV 表示锚点到 URL 的关

系(沿着链接反向移动)。它们都没有在锚点标签中广泛使用。出现在 HTML 文档 HEAD 元素中的 LINK 元素,也支持 HREF、REL 和 REV 属性。在 LINK 例子中(在第二章),REL 习惯用于表示,目标 URL 代表一个样式表。LINK 也提供 TYPE 属性,来表示目标 URL 的 MIME 类型。不象 A 元素,LINK 元素不把用户带到任何地方,它只把外边的资源连接到文档,且用和 IMP、APPLET、SCRIPT 和 OBJECT 元素使用它们的 SRC 属性十分相同的方法。

Xlink 中的简单链接

在 XML 中,我们调用典型的 HTML 链接嵌入式链接。链接连接资源——用链接元素中的定位器可以到达的任何信息,根据 Xlink 工作草图(资源包括文档,但是它也可以包含图形和其它文件)。定位器是 URL,定义的方法和为 HTML 定义 URL 的方法一样,并也可以同样地使用分段识别器。对于嵌入链接,定义链接的元素要作为这些资源之一来计数,并作为其它的目标。标准 HTML href 链接是非直接的链接,因为它只指出一个方向:从提供链接的元素到目标定位。对于嵌入链接,涉及的元素作为一个资源(在这个例子中,Back 按钮不计数,作为提供两个方法链接)。当链接启动,遍历描述采取的动作,即使链接不在任何新地方采取激励者(可以是用户或程序)。

在 XML 中的简单链接携带所有的在链接元素中的链接信息。在对应用程序的简单链接中,你不需要寻找携带有关定位器信息的其它元素,所有的定位器信息放在 href 属性中。创建一个真实的 XML 链接不是简单的;每一个链接的元素需要不只一个属性。使用简单链接元素的声明样本如下:

```
<! ELEMENT simple ANY>
<! ATTLIST simple
    xml:link          CDATA          #FIXED "simple"
    href              CDATA          #REQUIRED
    role              CDATA          #IMPLIED
    title             CDATA          #IMPLIED
    inline            (ture|false)   "ture"
    content-role      CDATA          #IMPLIED
    content-title     CDATA          #IMPLIED
    show             (embed|replace|new) #IMPLIED
    actuate           (auto|user)    #IMPLIED
    behavior          CDATA          #IMPLIED
>
```

你不必简单地调用链接元素——任何元素都是一个链接元素。用在这里和工作草图中的简单元素声明只是为了说明。事实上,任何元素都可以是一个链接——你不需要创建象 HTML A 那样的单独元素,它存在的唯一性是为了执行简单链接。

熟悉的 href 属性,仍然和它在 HTML 中工作的方法一样,虽然在后面我们将会看到它也扩展了,而其余的属性是新的。第一个属性,xml:link,用来对处理应用程序宣布,这个元素表示的是一个简单链接的元素。所有的链接需要 xml:link 属性;在 DTD 中用一个固定的值定义属性,通常可以证明它是一个比在每一个单独的 instance 元素中说明属性的更好方法。避免创建文档和使用没有声明 xml:link 的 href 属性的 DTD 诱惑。即使它们可以在 HTML 浏览器中工作,xml:link 服从处理应用程序,如果失去 xml:link 属性,将忽略 href 属性。Role 属性是选择项(象所有标记 # IMPLIED 的属性,或者在这里提供一个缺省值)。Role 属性是给机器处理器使用的,给人阅读的信息存在 title 属性中。Href 属性给链接提供定位器,URL 已经做过描述。Title 也是选择项,它包括弹出的信息,类似 ALT 标签。

注意

在 XML 中的 title 属性和在 HTML 中的 title 元素没有任何关系;它只是在名字上的不幸重叠。

Inline 属性,当设置为“真”时,声明所有建立在这个元素上的链接以嵌入方式链接。在简单链接中,inline 属性按缺省变为“真”。简单链接的工作草图状态通常是嵌入方式,并且总是单方向的,但是考虑到简单外部链接,它是困难的,而这种外部链接是有用的(一种可能是链接到突出显示信息,而不期望真的浏览)。简单链接向前指向一个单一的定位器,不需要更复杂的用于多方向的工具,或外部链接。即使简单链接可以令人信服地指向其它链接,Xlink 对取得那样的结果,有更好的结构。

在这里表示的下两个属性是 content-role 和 content-title,它们和属性 role 和 title 完成的功能类似;然而它们描述的是定位器指向的内容,而不是链接的内容。Role 和 title 属性把链接元素作为资源描述,而 content-role 和 content-title 描述目标资源。

Show 属性表现了在 HTML 链接上,是 XML 最大的改进之一。Show 属性接受 replace、new 和 embed 作为它的值。替换是 HTML 中的标准操作,链接用目标资源替换处理应用程序中(例如浏览器窗口)当前的资源。New 属性通过 HTML's TARGET 属性提供类似的作用,即打开新的上下文中的目标资源。在浏览器内,新的上下文可能是一个新窗口;在处理应用程序内,新的上下文可以是额外的并行操作的过程。Embed 属性加入一个完整的新维数度。当链接穿过时,由 href 属性指定的资源将嵌入到源资源的体中。换句话说,它将嵌入到作为链接的元素中。Embed 属性使开发者可以创建链接,它所起的更大作用就象 SRC 属性对比 ref 属性所起的作用。实际上,需要对 HTML's SRC 创建对等物的 XML 文档,有可能使用嵌入和由启动属性产生的自动穿过结合。

启动属性接受 auto 和 user 两个值。当链接穿过之前, User 需要外部作用(例如用户点击)。User 也可以是人查看文档,或者另一个处理应用程序在使用它。Auto 需要它一遇到资源,链接就被处理应用程序穿过。当 auto 和 show 属性的 embed 值结合,auto 的作用象客户端,需要处理应用程序寻找资源和链接元素本身。由于这些资源的不同内容,处理应用程序需要多功能来支持它。

最后一个简单链接的属性是 behavior,它提供开发者另一个地方存放指导程序如何穿过这个链接的信息。不象 role 和 content-role 属性,它不是链接到特定一边的连结,并可以把链接描述为单位。Xlink 规范说明不能为 behavior 提供例子的值,解释这个属性完全是文档作者和应用程序开发者的事。

图 11-2 显示由简单元素创建的一种链接。

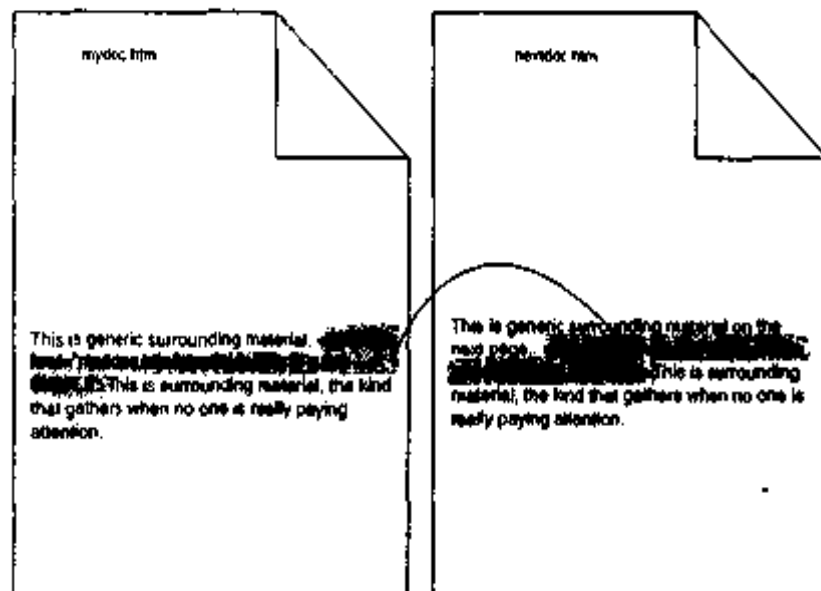


图 11-2 简单嵌入链接,XML

既然我们已经描述了所有这些属性,那么我们将用以前的声明元素来创建一个简单元素,它使用所有的这些属性并描述每一部分做什么。

```
<simple role = "emptiness"
href = http: //www. simonstl. com /zero /zero. html
title = "zero" content-role = "cliff" content-title = "last
stop before nothingness" show = "replace" actuate = "user"
behavior = "gotozero" />
```

href 属性识别目标资源。Role 属性给我们的处理应用程序提供有关在链接中我们目标的 role 属性的信息,在这个例子中是“emptiness”。Content-role 属性给同一个处理应用程序提供有关这个资源的信息,在这个例子中是“cliff”。Behavior 属性把链接描述为 gotozero。Title 属性把目标描述为“zero”,而 content-title 给用户这个资源的描述是“last stop before nothingness”(已到资源的最后)。因为启动属性要设

声明:

```

Xml: attributes    CDATA #FIXED "role REL content-role
REV
REV    CDATA      #IMPLIED
REL    CDATA      #IMPLIED

```

Xml:attributes 声明需要把 REL 属性值当作通常 XML role 属性链接的值来处理,而 content-role 属性的值等于 REV 属性的值。

当心

当重新映射属性时,请不要以 XML-作为属性名的开始。它被保留用于将来 XML 标准的使用。在有关标准的例子中,可使用 xl-作为选择前缀。

A 元素的 HTML 文本包含一个特点,它不能直接把下一页转到 XML 中。TARGET 属性虽然不可使用,但可以通过给 show 一个“new”值产生同样的结果。TARGET 属性与 show 属性结合就可以映射到 behavior 属性,但是成功的策略完全依靠处理应用程序解释 behavior 属性的能力上。Web 浏览器可以处理这个问题,但是 XML 解析器不能。

HTML A 元素在 XML 需要编址的最后一个属性是 NAME 属性,它在 HTML 中用来创建分段识别器。xml:link 采用不同的方法创建分段识别器。现在你可以为你的类型 ID 的 A 元素创建 NAME 属性。

```
NAME    ID    #IMPLIED
```

通常,调用你的 ID-type 要归于 ID。这更便于使用层叠样式表和其它期望发现 ID 属性的结构。从长远看,你需要重新命名 NAME 属性 ID。

Xlink 遇到一个简单的 #fragmentidentifier,它将检查文档中 ID 值列表。ID 值包括所有以类型 ID 定义的属性,不只是名为 ID 的那些(永远要记住声明 ID 属性作为类型 ID;否则,Xlink 将忽略它)。在这个例子中,XML 可以使用 NAME 属性,因为它是类型 ID,作为分段识别器。这就产生了平稳提供 XML 功能的 HTML 语法。我们的(大多数)完整 A 元素现在如下所示:

```

<! ELEMENT A ANY>
<! ATTLIST A
    xml:link                CDATA                #FIXED "simple"
    xml:attributes CDATA    #FIXED

```

"role REL content-role REV"		
href	CDATA	# REQUIRED
title	CDATA	# IMPLIED
inline	(true false)	"true"
show	(embed replace new)	"replace"
actuate	(auto user)	"user"
REV	CDATA	# IMPLIED
REL	CDATA	# IMPLIED
NAME	ID	# IMPLIED
TARGET	CDATA	# IMPLIED>

定位器和分段

即使 Xlink 可以使用简单的 HTML 分段识别器符号,但它使用 XPointer 将更是有意义的东西。许多 XPointer 完成的任务证明是十分有用的,即使在简单的链接中,使作者和开发者能够处理元素,而不是文档,作为涉及链接的主要单位。因为它可以使用 XPointer,Xlink 定位器语法比 HTML 的强大的多,它提供一些可以通过结构,给文档的部分、ID、HTML 锚点甚至文本内容编址的工具。

注意



当 XPointer 使识别所有种类的文档分段变容易时,处理它们就变得更困难了。一个单独的工作组发布了这些问题,并计算出如何表示和处理这些不同类型的文档。需要更多信息,访问站点:<http://www.w3.org/XML/Activity>。

HTML 使分段识别器能够跟踪使用以下 href 属性语法的 URL:

Href = "url # fragmentidentifier"

XML 也有类似的语法功能,但是典型地使用更高级的 XPointer,而不是 HTML 分段识别器:

Href = "url # XPointer"

或

href = "url | XPointer"

或

href = "url? XML-XPTR = XPointer"

在第一个例子中,使用 #,由 URL 参考定位,作为整个文档的定位被处理器取

走(如果 show 属性设置为“replace”,代替当前的文档)。然后被 XPointer 参考的定位由客户定位。在第二个例子中,使用|连结符,根据 Xlink 的标准,把没有意向的信号发到处处理模型,是用于访问指定的资源。使用标准查询语法的最后文本给服务器提供处理 XPointer,减小传输所需带宽,因为服务器可以只返回它所需要的那些文档。

XPointer 给 show 属性大量更强功能的 embed 值,因为它可以使链接访问到文档的部分,用当前文档的上下文显示,而不是完全代替当前文档。ID 值得使用,使作者容易地把文档细分为有很好的结构元素的便于管理的组块。你可以显示一个很长的文件,这个文件包含许多较小的块分解为更小的块,而不是作为一个很大的文件显示它。这就使利用其它系统的已知链接特性摘取其它文档变得容易。

交叉参考



如需要更多有关 Xanadu 的信息,可参阅文章“Embedded Markup Considered Harmful”,它是在 1997 年冬天发表在 World Wide Web Journal 上,可以在 O'Reilly and Associates 得到。要使工作有效,文件结构需要改变,避免处理应用程序装入整个文档,而只是装入所需的块。文件系统本身必须是 XML 处理器(甚至是对象数据库),以元素存储文档而不是以必需要顺序解析的单个文件存储。需要了解更多的信息可参阅文章“Build the File System into the File”,它的站点是:<http://www.simonstl.com/articles/filesys.htm>。

更复杂的链接

扩展链接给链接世界一个全新的几何结构,使建立新的体系结构和界面成为可能。即使 HTML A 元素模型“I am here——click to go there”也能很好地让 Web 开始工作,到时移到“I am one part of a set ——O treat me as such and explore”。扩展链接是多方向的链接,从长远来看,外部链接使开发者可以创建更复杂的结构,更容易对链接进行管理。

扩展链接使开发者可以创建链接组,有效地为用户(或处理应用程序)提供一套不只是单个目标链接的选择。即使处理应用程序处理扩展链接仍是很松散地在工作草图中,你也可以容易地描绘出作为一套选择的扩展链接,它将出现在弹出的菜单(或其它界面)中,让用户选择方向。用在这的标准应用程序是一个辞典。当用户点击单词,一套同义词将出现在弹出的菜单中。用户可以从这些单词中选择,或者在所选择的单词中查看更进一步的信息。

执行这些链接有一点复杂。当我们做简单链接时,我们开始检查一些样本声明,在这个例子中,这个元素能表现扩展外部链接:

```

<! ELEMENT extended ANY>
<! ATTLIST extended
    xml:link      CDATA      #FIXED "extended"
    inline        (ture|flase) "ture"
    content-role  CDATA      #IMPLIED
    content-title CDATA      #IMPLIED
>

```

在以前的简单例子中,执行扩展链接的元素不需要给扩展命名。它们只需要简单地把 `xml:link` 属性设置为“extended”。

我们的扩展元素和简单元素基本类似,有两点变化。第一,`xml:link` 属性值由原来的“simple”改为“extended”。第二,扩展元素没有任何 `href` 属性和任何目标描述。扩展元素必须包括一套包含定位器的子元素。定位器元素如下所示:

```

<! ELEMENT locator ANY>
<! ATTLIST locator
    xml:link      CDATA      #FIXED "locator"
    role          CDATA      #IMPLIED
    href          CDATA      #REQUIRED
    title         CDATA      #IMPLIED
    show          (embed|replace|new) "replace"
    actuate       (auto|user) "user"
    behavior      CDATA      #IMPLIED
>

```

定位器元素携带关键链接信息。`Href` 携带用于启动链接的定位器。`Title` 提供用于给人阅读的信息,而 `role` 携带用于提供给处理应用程序的信息。每一个定位器都有自己的 `show`、`actuate` 和 `behavior` 属性。如果定位器不定义这些属性,它将用在扩展元素中,按缺省指定的属性。这就使创建指向不同位置但共享相同的 `behavior` 属性的链接组变得容易。同时保留了当需要时,单个链接到不同 `behavior` 的权利。

你将会发现扩展链接,不论是嵌入还是外部,在数量大的情况下都有用。嵌入链接可以使用以前介绍的 DTD,如下所示:

```

<extended>History Texts
<locator title="African" href="african.xml" />
<locator title="Asian" href="asian.xml" />
<locator title="European" href="european.xml" />
<locator title="North American" href="namerican.xml" />
<locator title="Pacific" href="pacific.xml" />
<locator title="South American" href="samerican.xml" />

```



```
< /extended>
```

这个链接完成了几件事情。第一,如果用户在文档中遇到它,这个元素表示实时嵌入链接。点击单词“History Texts”将出现一个菜单,它提供在定位器元素中的选择标题的列表。选择这些标题之一,就会把用户带到索引文档。第二,因为这是扩展链接,它建立的链接可以连结到所有这些文档。在定位器中列表的文档,如果它们在某点遇到这个声明,所有相互的链接在文档级。右击它们中任何一个的页边空白,可以调出其它文档的菜单,这个文档点击了链接。当我们到达 `xml:link` 属性文档值,我们还会重复它。

扩展外部链接类似以前的链接,除了扩展链接元素的内容不是它自己链接部分外。点击扩展链接元素的内容,如果它有任何信息出现,就不要点击菜单。扩展外部链接只建立其它元素之间的连结。例如:

```
<extended inline="false">
<locator title="Overview" href="#overview" />
<locator title="Architecture" href="#architecture" />
<locator title="Detailed Design" href="#details" />
<locator title="Parts List" href="#parts" />
< /extended>
<SECTION ID="overview"><title>Overview< /title>
...
< /SECTION>
<SECTION ID="architecture"><title>Architecture< /title>
...
< /SECTION>
<SECTION ID="details"><title>Detailed Design< /title>
...
< /SECTION />
<SECTION ID="parts"><title>Parts List< /title>
...
< /SECTION>
```

在这个例子中的扩展元素除了链接外,没有其它内容。它在跟随的 `SECTION` 元素之间建立链接。不关心 Overview `SECTION` 的读者可以点击它并得到一个更详细描述菜单。类似地,闯进部件列表的人也可以使用同样的菜单返回到 Overview 或 Architecture `SECTION` 元素。这些链接真正具有多方向的功能(如果应用程序支持它),因为用户可以在多个定位之间导航,不用关心向前或向后移动,在这个意义上,正是客户所期望的。这些链接可以向两个方向穿行——点击目标文档(或使用它的链接界面)将出现链接列表,它包括用户要访问的文档。增加的灵活性无疑会搞乱许多在 HTML 链接的 Web 中已经迷惑的用户,但是对有经验的用户是一个功能强

大的导航工具。

为了对比,我在图 11-3 中显示了一个扩展嵌入链接,在图 11-4 中显示了与它对应的扩展外部链接。注意所有的链接都可以在两个方向穿行。

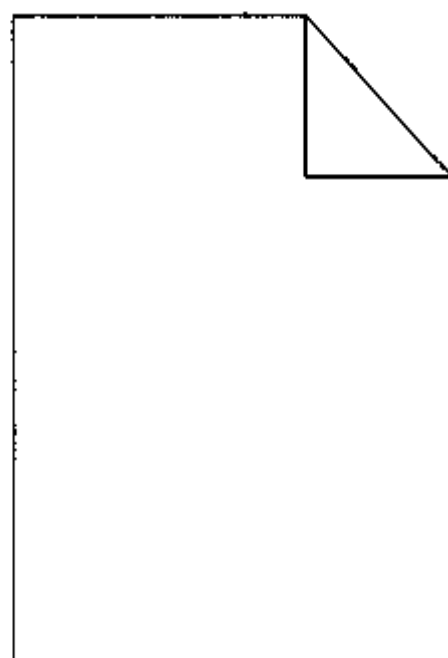


图 11-3 扩展嵌入链接,XML

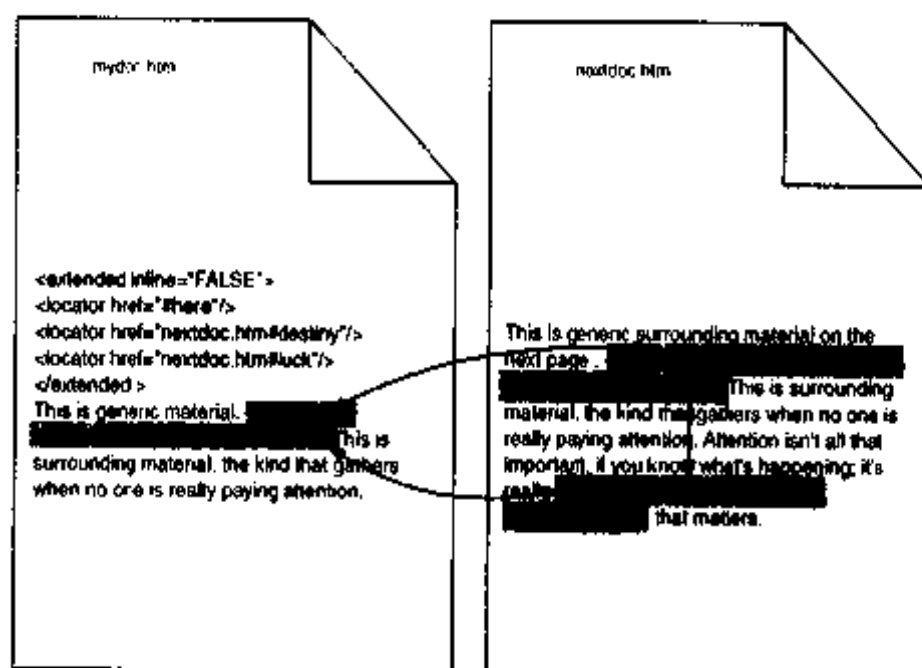


图 11-4 扩展外部链接,XML

这些新工具似乎令人激动,它们使管理链接更加困难。XML 文档不能总是知道和它共享链接的其它文档,特别是如果那些链接存储在属于其他人的内容上。管理链接迷宫是向要求链接信息中心化的挑战。幸运的是,xml:link 提供一些基本工具,专门处理这类问题。用于 xml:link 属性的组和文档值可以创建扩展链接组元素,它帮助管理链接,告诉文档检查彼此相关链接,并为有关数据组建立票据交换所。使用扩展链接组需要创建两个元素——一个用来定义组,另一个识别组中的文档。对这些元素的声明类似以下形式:

```
<! ELEMENT group (document * )>
```

```
<!-- ATTLIST group
      xml:link    CDATA    #FIXED "group"
      step       CDATA    #IMPLIED
-->
<!-- ELEMENT document    EMPTY -->
<!-- ATTLIST document
      xml:link    CDATA    #FIXED "document"
      href        CDATA    #REQUIRED
-->
```

这些声明比跟在它们后面的简单的多。在组元素中,所有在元素实例中需要声明的是 step 属性,它告诉处理应用程序,在停止搜索相关链接之前,有多少链接层。文档元素只包含 href,它把处理应用程序带到你想要搜索链接的文档。当处理应用程序遇到这些元素中的任何一个,它将装入由在组中文档元素 href 属性指定的文档。然后它检查这些文档,并连接到源文档,创建一个链接表格。处理应用程序将装入所有的文档,并处理在它们中的链接信息。如果 step 属性大于 1,处理应用程序把文档装入到源文档被第一个装入圆括号文档链接的地方。

提示



通常,step 的值应该保持较小的数,以阻止文档装入数百个其它文档而挤占带宽。

扩展链接组的优点,是它们可以使处理应用程序容易地获取有关文档的完全信

这个变化的含义是深刻的。扩展链接组使中心化链接信息成为可能,取代了链接迷宫,而使开发者能够检查和管理链接,而不必阅读没完没了的文档。它也减少了带宽成本,使开发者能够要求 XML 文档只下载一个(或几个)额外文档创建完整的链接列表。

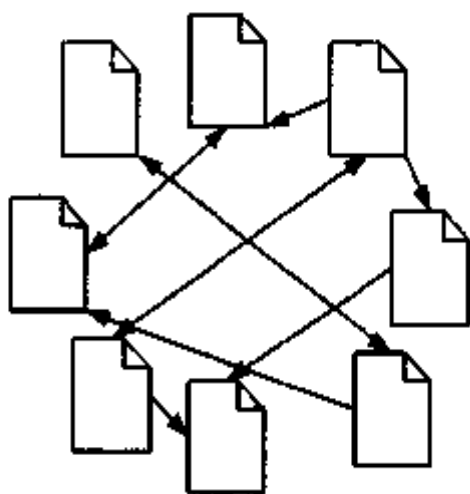


图 11-5 老样式链接,通过非中心化链接,连结一套文档

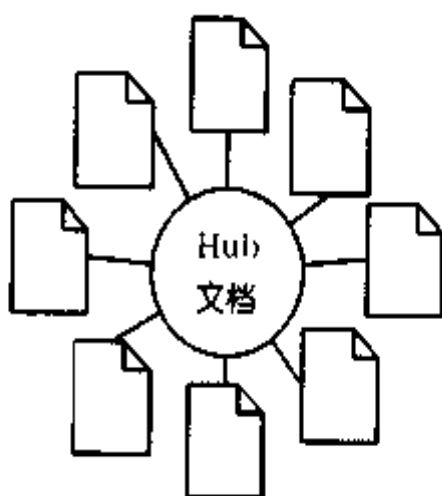


图 11-6 中心化链接,提供网络中心便于链接管理

扩展链接组也使把链接有效地从其它文档加到你自己的文档成为可能。只要其它文档认可你的文档,从你的文档到目标文档的链接将被目标文档认可,并可以作为输出链接出现。即使目标文档不认可你的文档,当浏览文档时,处理应用程序仍能记住来自你的文档的链接,使文档相互之间可以方便地参考。如果处理应用程序记住从文档到文档之间的链接,目标文档可以链接到你的文档,虽然只在处理应用程序的上下文内。这使得注释文档比过去更简单的多。

链接的奥秘

从 Xlink 说明书上看,它能给 Web 带来难以置信的功能,但仍有很长的路要走,问题是难于确定 Web 为设计那些 Xlink 说明所表示的目标应用程序。几个团体已经介入 Xlink 创建中,它们需要所有不同的规则和参考。

一些关键的问题仍没有解决,最基本的是:链接是什么?链接是一套资源或在这些资源中的连接?简单的链接容易决定它的非方向穿行路径,扩展链接可以是多方向的。然而,当前不存在可以识别在资源中的路径的结构是可遍历的。(换句话说,扩展链接象简单链接一样使用同一个词表,但是方式完全不同。)开发者可以用他们喜欢的任何方式执行它。可以给他提示的属性——特别是 `role`、`content-role` 和 `behavior` 属性——没有指定的值。一些开发者建议使用样式表,可能是 XSL 决定链接穿行路径,但是到目前为止,开发还没有在那个方向展开。

并非每一个应用程序都能够使是用遍历。许多开发者把链接看作一套处理它们的不需要路径的无定型装置,因为遍历对他们的应用程序不重要。你可以发现链接装置对许多建模应用程序十分有用,它们和 Web 浏览器没有任何关系,浏览器描述资源和它们在完全不同的上下文中的举动。指定遍历 `behavior` 属——在不存在的字段里穿行。其他开发者坚持弧型首位连结资源(访问站点:<http://www.jfinity.com/xarc/>)。

交叉参考



有关于 Xlink 或 XPointer 说明的问题吗?想做一点贡献吗?你可以访问公共 Xlink/XPointer 邮件列表:<http://collie.fujitsu.com/xlsp-dev/threads.html>。签约参加工作可以到站点:<http://metaleb.unc.edu/xml/maillinglists.html>。

XML 链接似乎很复杂,但是进一步研究和使用将使它们更友好。我们将在下一章看到,这些链接设计的含义可以证明它就象 XML 的含义一样具有生动性。

第十二章

处理 XML:仓库、处理器和网关

虽然本书主要论述如何编写 XML 文档和创建 XML DTD,但是创建文档和 DTD 只是 XML 的一部分。不带应用程序的文档可以对学习语法有用,但是它没有其它吸引力。完全处理解析和处理应用程序需要写另一本书,但是理解这些应用程序是如何工作的基本概念对创建有用的 XML 是很重要的。从事 XML 开发工作的许多队伍,分成单独的组创建文档类型和处理器应用程序,因为这两种工作需要不同的技术。同时这两个组需要共享通用的词表。在本章我们将检查新词表和体系结构,它们是 XML 创建,一些 XML 的含义用于数据处理。

用 XML 创建分布的体系结构

许多开发者不需要接触代码行就能让 XML 为他们工作——压缩包装的服务器和浏览器可以完全胜任存储和显示 XML 信息的工作,它们顺着当前的 Web 服务器和 Web 浏览器的思路,用 XML 创建一个简单的 Web 站点和用 HTML 创建一个简单的 Web 站点大致相同,除了定义你的词表和创建样式表这些额外的工作,来告诉浏览器如何显示 XML 信息。用和 HTML 相同的工具,象 CGI、Active Server Pages、Servlets 和 Cold Fusion 都可以用于 XML。它只是格式的变化,因此更多地关注语法是必要的,但是转换是不困难的。下面几节,我们将探讨现存基本结构及其在以后几年由于 XML 的广泛应用,将面对的变化。

仓库(服务器)

早期的 Web 服务器是非常简单的。它们在现存文件系统和网络结构上建立,并为用户提供访问以十分传统的方法存储的信息。当越来越多的结构化信息,特别是存储在关系数据库中的信息,变成 Web 的一部分,服务器也产生一些表示信息的工具。服务器端的开发使站点管理者可以给用户提供控制对敏感信息的访问,为他们

设置安全提示信息。目前许多服务器端的处理移到其他服务器端的计算机(它然后访问远程数据库或文件服务器)或客户端。因为处理信息可能发生在传统客户和服务端之间的任意一点,本节将集中在服务器仓库部分,即文件系统和数据库。下节将集中处理部分,不管它发生在那里。

在 HTML 世界里,大多数信息以两种结构中的一种结构存储,即数据库和文件系统。典型的关系数据库通过接受用户请求和 Web 相连接,查询服务器,并把结果用 HTML 格式化。文件有容易的路径,Web 服务器把它从客户端收到的 URL 传输到文件路径,并把它在文件中发现的信息发送到客户端。有时数据库潜藏在象文件一样的 URL 后面,有时文件真实地存在数据库中,但是总的来说,两种形式的差别是相当明显的。把文件放到数据库中很少会比把文件放到文件系统更有效,同样把数据库信息放到文件中也是一个失败的建议。

XML 预示在文件请求结构中加入重点,为的是完成它的有效性和链接需要。虽然 DTD 和样式表的额外传输成本不可能达到以文件为基础的 Web 服务器的当前功能的上限,但新的链接特性可以达到。新的嵌入功能将促使开发者创建包含其它文档部分的文档。XPointer 可以方便地描述文档块。如果所有的 XPointer 处理发生在客户端,那么服务器将花费大量精力用于把文件发送到客户端,而在客户端这些文件只是部分被使用。链接到多文档的文档可以大大地增加文件装入,特别是如果处理应用程序贪得无厌地搜寻信息的关键位。而扩展链接组可以通过创建真正的多方向链接,给客户端提供轻松的功能。这些多方向链接允许在服务器上有大量的交通拥挤,特别是如果开发者使他们的链接分散在多个文件,而不是把它们集中在中心化链接的文件中。

XML 允许忽略以前的文件系统和数据库之间的明显差别,因为它的鲜明的层次结构是在数据库内存储的最好选择,同时也是在数据库内存储信息的最佳载体。XML 以系列文件存储,可以很好地表示表格,甚至一套表格,当然处理这样的文件会增加成本。XML 文件十分适合在数据库和数据库信息有关档案存储之间的交换。当包含大量的信息时,使用 XML 文件作为数据库本身可能不是最好的主意。同时,复杂的 XML 文档可以严重地加重关系数据库资源的负担,关系数据库把 XML 层次结构映射到它内部的表格式的结构,使在相关仓库中的 XML 存储为相等的不确定命题,增加了处理成本(许多新关系数据库,对象关系数据库,提供额外的设施帮助解决这些问题)。数据库信息可以平稳地流入 XML 文档,把 XML 文档作为数据库来处理,或者把 XML 信息放到关系数据库中,这不是最好的主意。

还有许多例子,它们有到 XML 文档存储更灵活的方法,这是有很大益处的。在这些例子中,客户端只需要文档分段,或者文档在发送到客户端之前,必须解析或转换,如果 XML 文档以一系列字节按顺序存储。把 XML 文档放到能反映它们的内部层次结构的仓库中,是一个答案。它既满足了有效处理 XML 文档,有满足了对信息

访问的要求。两种数据库,对象存储和层次数据库,都允许流线型处理(它们二者都不象关系数据库在操作表格式的数据那样有效,因此关系数据库肯定会继续占有相应的位置)。

对象存储,可以在 POET(<http://www.poet.com>)软件得到,对象设计公司(<http://www.odi.com>)存储对象(象在 Java 中创建的 C++ 和其它面向对象语言)。它们都是在层次格式中,可以随时返回到创建它们的程序。层次数据库(象 ADABAS, 站点在:<http://www.softwareag.com>)是较旧的。早期关系数据库的竞争者优化存储层次结构化数据,就象 XML 文档一样。而详细情况随着数据库环境的不同而变化。XML 预先解析文档文本,总的效果是可以完成工作的。它减少或消除了串行和解析 XML 的成本。除了装入文档,解析它和摘取有用信息外,Web 服务器可以面向数据库查询,并且把所需的信息返回。把这些信息编织在一起的开销比把它们拆开的小。

从系统中创建高效的仓库需要花费大量工作;理想的界面很象在浏览器上一样,用象结构的文件系统组织文档和查询语言(XPointer 或其它),在不需要整个文档时,重新得到指定信息。在这些系统中加入信息,是我们今天的以文件传输协议(FTP)为基础的文件系统所不能处理的。但是有希望的成功者(象 IETF's Web Distributed Authoring and Versioning, WebDAV)一定会填补这个空白,使这些系统的应用透明有效。

处理器(CGI、Middleware 和 Beyond)

Web 服务器最基本的作用是处理它们服务的文档。一些在 Web 上的处理工作是在服务器上完成的。它可以让用户访问以非 HTML 格式存储信息的仓库。服务器可以管理安全、收集信息和完成其它任务。其它的处理工作是在 Web 浏览器和其它应用程序(如脚本、Java 程序和其它象浏览器的插入程序)上完成的。随着对 XML 支持的软件日益增多,这样的一些处理工作变的更加容易。在任何情况下,它都将变得更加分散,可以从仓库服务器移到中间服务器,然后再到客户端,模糊了客户、服务器、和中间的含义。存储信息的 Web 服务器,把信息送到 Web 客户端(浏览器),把信息显示。不象 HTML 文档,XML 文档容易分解和合并,且转换和处理可以在文档分解时同时进行。在 HTML 世界,用于客户端和服务器的灵活应用程序包括创建 HTML 文档的脚本,它传到浏览器根据客户的需要用不同的脚本激励文档。在 XML 世界,处理模块从相关的数据库收集 XML 文档,然后把它送到另一个模块,作为文档中的表格重新组装。当它到达用户,另一个鼠标点击模块可以把同一个信息传到图表、表单和本地数据库的入口中。

灵活性是 XML 成为交换格式的强大工具的部分原因,但是在我们查看文档的方式上,确实需要对它做一些改变。XML 文档不只是一系列字节(虽然它们一定是

一系列字节),也不是一套表示、修改、控制和摘录的结构。标记样式肯定会对处理有多大的灵活性有影响,但是处理器可以依靠文档的内部的结构,转换这些结构,代表作为新 XML 文档的新结构。如果重复许多次结构,解析和文档重建工作的成本将变得很重要。MDSAX 工具(<http://www.jxml.com/mdsax>)是用来集成多层处理文档,为只解析一次文档或者在最后再建立它(如果不是简单地映射到应用程序数据结构),创建有效的结构。中间工具把普通的处理应用到以结构和内容为基础的特殊文档,并用通用部件创建十分专业化的应用程序。XML 文档的共享结构,使开发者能够创建在词表中通用的工具。

XSL 是通用应用程序的一个很好的例子,它把客户端和服务端两边的处理统一考虑在一起。支持 XSL 的浏览器可以在客户端完成转换和格式化,而服务端完成客户端不能处理的 XSL 本身所需要的转换(典型的是到 HTML)。灵活性为站点设计者提供了使用新浏览器功能的好方法,同时仍然支持不理解 XML 的浏览器和很少量 XSL 的老浏览器。XSL 转换甚至可以用堆栈——XML 转换的结果是可以被另一轮的应用程序处理的 XML。

网关(浏览器、编辑器和其它应用程序)

让信息进出这些仓库和处理器需要另一个类型的应用程序,它和外部世界相互作用。这些通路可以是一个浏览器,把 XML 文档转换成用户屏幕上的一套像素;也可以是一个编辑器,使用户能直接创建新文档,修改旧文档;也可以是通到其它系统的网关,象通过包含在 XML 文档中的命令控制的仪器。输入和输出使存储和处理值得做。象其它领域一样,在这一领域的技术正在开发中,最好的解决方法目前还未出现。

网关以解析器为基础,它把 XML 文档中的信息送到应用程序。应用程序然后把信息显示给非 XML 格式的世界。在一些情况下(如编辑器),信息可以映射回 XML 文档并返回仓库;其它采用同一条路,例如,从网关应用程序到用户屏幕或一些代码控制设备。一些网关只收集不是 XML 格式的信息,并把它转换成 XML 方式,为输入提供一条路径。下面的两节,将详细介绍为人和存储在 XML 中的信息之间打交道的应用程序,即浏览器和编辑器。

浏览器:剖析和未来

在极大程度上,我们通过本书学习使用浏览器,并把 XML 带到生活中去,使它成为看的见摸得着的东西。粗略简单化的浏览器包括四个主要部分:通讯引擎,用 HTTP 和其它网络协议发送请求和接收信息;解析器,用来解释信息;展示引擎,显示由解析器发现的元素;界面,控制用户和所提供的信息之间互相交流。一个简单的浏

览器如何处理 HTML 文档的模型如图 11-2 所示。

通讯引擎从 Web 服务器得到 HTML 文件并把它们送到解析器,解析器把它们分解为离散的元素树。展示引擎检查这些元素的内容,并把它们按屏幕要求的适当方式格式化,下载所需要的其它材料。界面用它的菜单、导航帮助、滚动栏和其它特点,给浏览器提供显示文档的窗口。它处理用户采取的行动,当需要时,打开新的页面,经历通讯→解析→处理这一相同的过程。

今天的浏览器比这个简单的模型要复杂的多。它包括脚本引擎、样式表解释器、Java 虚拟机器、嵌入界面和各种图形引擎,以及数目日益增长的附件,如提供邮件、新闻、组件、HTML 编辑和有其它集成特点的附件。

在第九章描述的通道特性增加了一些额外内务操作。处理所有的这些部件,使浏览器从一个小的简单的程序,增长成为全面的办公应用程序,占用更多的硬件驱动空间,花费更多的时间下载每个新版本软件的发布。

在浏览器商家之间的竞争由于解析器,也改变了规则,表示和界面:文档变成动态的。我们在第七章接触了一些,但是它对浏览器的影响值得更多的关注。文档不再是静态的实体,在它到达浏览器之后,可以变化。脚本可以加入和去掉元素,改变它们的外观,修改它们的内容,让它们在屏幕上来回移动。解析引擎仍然在代码到来时读取它,但是它所产生的结果树,现在对操作和修改开放。脚本可以有效地对文档树进行读写访问,有可能创建一个全新种类的以浏览器为基础的界面。

W3C 在当前的处理标准化中,技术处于领先地位。文档对象模型工作组正在为访问和操作 HTML 和 XML 文档,创建应用程序处理界面(API)。级别 1 提供基础,是 W3C 推荐的,在级别 2 开始工作。图 12-1 摘要表示了文档对象模型对简化模型的影响。

注意

文档对象模型(DOM)级别 1 为软件开发者和 Web 脚本作者,访问和操作解析的 HTML 和 XML 内容,提供结构方式。所有标记以及任何文档类型的声明都是可使用的。DOM 级别 1 有意限制了内容表示的范围和操作范围。交付、有效性和具体化等都推迟到 DOM 较高的级别。

解析器为浏览器创建了最初的状态,在此之后它创建的元素树可以改变、重新组织甚至重新创建。XML 文档甚至它们的文档类型声明,都可以改变形状(这可能出现问题,至少是在较高级别的 DOM 出现明确的有效性之前)。

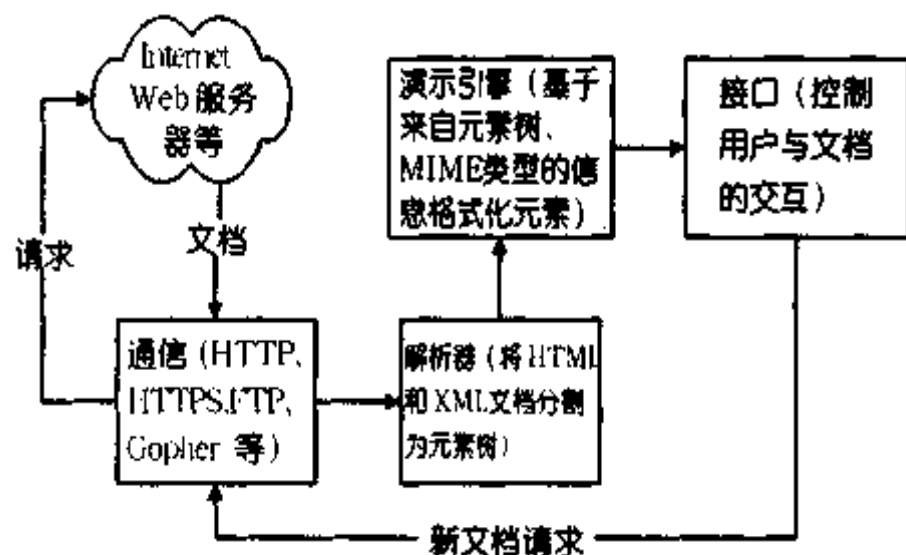


图 12-1 简单的浏览器结构

这些开发是继续扩大浏览器功能的最新刺激因素。Netscape 公司一直梦想创建一个可以提供完全界面的浏览器,这一梦想已接近实现,虽然 Netscape 和 Microsoft 之间的竞争仍在继续。新的极端灵活性的含义是很多的,浏览器环境正在达到足够处理各种数据表示和处理工作的地步,它们中的大多数是用专业客户-服务器工具建立的应用程序。如果 DOM 为开发者继续提供足够的灵活性,在浏览器中编写一个单词处理器,它一定会有足够的灵活性创建比现在的客户界面形式更加强大的客户界面。

除了为主要浏览器商家说明这些可能性外,这些新技术的开发也为更广泛的开发者观众揭开了更大的可能性。当重新创建 Netscape Navigator 或 Microsoft Internet Explorer 的功能时,大大超出了大多数程序员和公司的能力。XML 为定位在小环境的小浏览器打开了新的可能性。第一个真正的 XML 浏览器是 Jumbo(<http://www.xml-cml.org/jumbo/>),它的创建是为了证明化学标识语言的可能性(CML)。Jumbo 是一个 Java 应用程序,它合并了解析器、Java Swing 分类和一些自定义代码,可以一般地显示 XML,也可以以树的方式显示 XML。在右边窗格(如图 12-2 所示),用更进一步的信息作为应用程序对象,做类似在屏面上画分子结构的工作(如图 12-3 所示)。

Jumbo 是一个实验的浏览器,不是商业工具,但是它的创立具有重要意义。Jumbo 的体系结构明显地显示出,XML 可以用当前的体系结构,补充或代替现存的用来显示内容的,嵌入和对象结构。现在所有不包含图形和标记的内容,必需用特殊的标签——EMBED、APPLET 或 OBJECT 声明。把插件程序加到浏览器不是容易的。所有这些暴露给用户的方法,是冒着错误写入和恶意代码的风险。更灵活的体系结构允许用户或文档通过元素和 Java 程序联系,在浏览器窗口或单独弹出的窗口,提供特殊元素处理。用这种方法,只包含文本的元素可以用处理标记的工具显示,而元素需要进一步处理,它可以用它们指定的工具接收它,如图 12-4 所示。

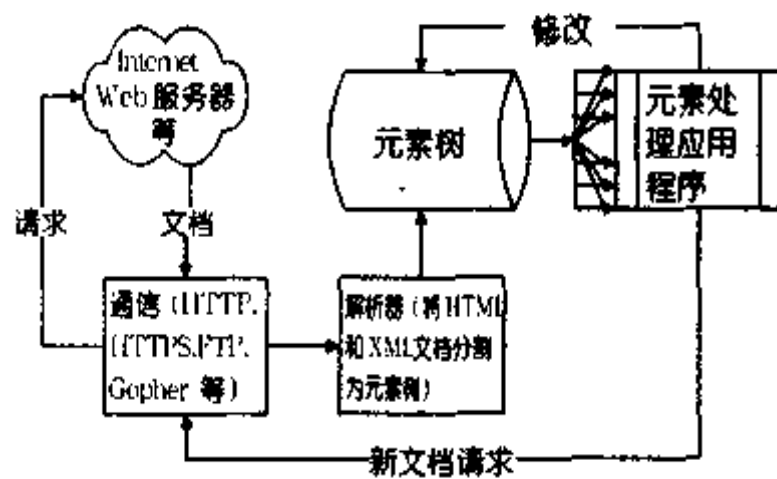


图 12-4 开放的浏览器允许元素被多种工具处理

在当前的浏览器中执行这些体系结构,无疑会产生另一个竞技场,在那里,互相竞争的浏览器开发者将建立不一致的标准,例如 API 拒绝兼容,它们用特殊浏览器的代码工作。XML 本身仍然包括对它们的部分有用的少数特点,象 NOTATION 声明。即使用 NOTATION 指定外部读者,在集成浏览器的时代,它也是陈旧的,作为指定外部处理的手段,对 XML 仍是有用的。链接到 Java 程序的 NOTATION 声明可以用 #FIXED 属性连结到元素,对浏览器宣布这个元素所需要它自己指定的处理工具。与其它属性合并,定义元素是否需要它自己的窗口,或者嵌入显示,这可以很大程度地简化 XML 集成的第一个步骤。硬件部分将开发 API,它允许那些嵌入表示工具越过尺寸和位置,用浏览器刷新屏幕。

注意

Jumbo 走的很远,甚至允许用户选择他们的解析器! 模块性的级别允许很大的灵活性。

为处理器加入这种处理特殊 XML 元素的支持,可能对浏览器产生不可预料的副作用。对外部开放浏览器,象这样的嵌入应用程序比早期加入插入结构或 Java 程序是更戏剧性的变化。浏览器将变得不再是完整的,越来越多的应用程序将插入到同一个浏览器中。在这个方案中,浏览器缩小为通讯引擎、解析器,以及能使不同应用程序通讯和改变元素树的框架。如果需要,浏览器仍然提供界面服务,并是显示文本的一套基本工具,但是即使这些也要从其它的应用程序中获得。把这样的浏览器打上商标出售比出售现在完整的浏览器要困难的多,但是它将是浏览器发展的最终的合理目标。

在浏览器可能的未来,浏览器的表示和界面方面将被处理元素的其它应用程序代替。它们将有它们自己的表示和界面结构,浏览器将继续与这样的应用程序合作。许多浏览器中的功能将分散到处理特殊元素的应用程序中。它们共享一个解析器、

一个通讯引擎,甚至同一个元素树,但是浏览器将不必作为一个单位。解析器、引擎和元素树都是浏览器的服务,而不是一个单位。

在这个级别,深入讨论操作系统和浏览器的集成是可能的,虽然它们都不能集成到一定程度。操作系统仍然需要为运行这些服务提供环境,而浏览器的几个端口变的比它们在过去更重要。元素处理应用程序和保持它们同步的服务为应用程序和 API 的开发提供了新的领域。应用程序可以是 Java 程序、ActiveX 控件甚至 COBOL 程序,这都没有关系,只要它们可以和元素树相互通讯。处理应用程序可以使用 Java's Abstract Windowing Toolkit、Java Foundation Classes、Microsoft's Win32 或者 Mac OS Toolbox。模型也可以在这些系统甚至整个新系统工作,为日益增多的部件提供通讯和协调。

为什么有人想要用 Hydra(一种操作系统)代替浏览器呢?因为它具有一些优点。第一,虽然它由大量不同的部件组成,但是对用户来说,不会看到任何不同。单独的浏览器窗口仍然可以和这些部件协作。第二,它对大量的潜在处理应用程序提供解析服务,而不需要查询每一个包括它自己的解析器的应用程序。它为解析服务提供单一界面,使开发者可以创建浏览器服务 API(应用程序编程接口),而不是选择解析器,认可它,然后编写适合解析器的代码。

目前视窗程序可以使用 Internet Explorer 作为 ActiveX 控件,并且 Internet Explorer 可以给 ActiveX 控件和它们的数据提供主页。如果 Microsoft 进一步开发,那么它有可能发展成以前描述过的共享浏览器服务模型,但是实际上,现在 Internet Explorer 仍然是内部互相通讯的一块程序,但是对外部程序的通讯作为一个单位。Microsoft 一定会打算把更多的浏览器插入到操作系统中,因此这会变化很快。Microsoft 的用户增长的很快,每年都要增加资源。

浏览器服务,象在摘要模型中定义的那样,在我看来它比 Windows 更适合不太大的环境。象 Jon Bosak 在他的白皮书“XML、Java 和 Web 的将来”(http://sunsite.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm)所讲的,“XML 给 Java 一些事情去做”。Java 已经在分类库的标准上建立,完成提供通讯引擎功能的网络界面。JavaBeans 标准和 Java Developer's Kit 1.1 同时出现,它为应用程序和 Java 程序通讯提供结构和大量对象之间的直接通讯通道。

交叉参考



如需要更多有关用 Java 创建 XML 处理应用程序的信息,可参阅“Building XML Applications by Simon St. Laurent and Ethan Cerami (McGraw-Hill, 1999)。

编辑器

创建 XML 编辑器比创建浏览器,在许多方面都面临更大的挑战。创建 XML 编辑器可以完成占主导地位的 WYSIWYG(what-you-see-is-what-you-get)单词处理器,根据提议的条款,需要创建交互式浏览器,为用户提供创建文档结构,而不是仅格式化文本所需要的工具。管理创建的结构,同时用客户习惯的方式表示信息是一个复杂的工作。保持界面是一个简单的工具也是困难的任务。创建一个这样的编辑器作为通用的工具,而不是专注于某一个词表的编辑器,是特别困难的。

创建 XML 文档的方法之一是通过访问模型,虽然它不一定是最好的编辑它们的方法。应用程序可以采用 XML DTD 方案,当它们可以工作时,从根元素开始,把它作为一套可能发生的事表示给用户。DTDs 不能识别那些元素是用来做根元素的,因此应用程序需要一些额外的信息表示给用户。当由编辑器表示时,用户可以根据 DTD 作出选择。编辑器可以实施约束,阻止用户创建非法文档,当表示的元素是可接受时,只允许他们向前移动。许多情况下,象在第八章讨论的工程管理者听取汇报,这个方法是合适的,使作者可以把他所有的信息在大家一就坐就发送出去。更复杂的文档需要一个暂停功能,以便在中间停止。全面的编辑器将远远超过这个范围。

交叉参考



如果你需要把编辑器很快地装配成一个特殊的 XML DTD,请看一看 IBM's alphaWork XML Editormaker(<http://www.alphaWorks.ibm.com/formula/xmleditormaker/>)。它采用 DTD 并创建一个以 DTD 为基础的小编辑器。

第一代通用的 XML 编辑器,著名的 Microsoft's XML Notepad(如图 12-5 所示)和 Vervet Logic's XML Pro,趋于变成树状导航线路,清晰地表示文档结构,使用户适当地插入和删除文档(和元素内容)。对于一些特别简单的文档和高度结构化的文档,这种方法非常有效。但是它对特别大的文档,或者被用户搞乱格式和结构的文档,这些编辑器最不好用。当这些编辑器正在很好地表示 XML 文档的结尾时,许多用户更喜欢他们每天使用的没有层次途径的文档。

交叉参考



XML Pro 可以从 Vervet Logic(<http://www.vervet.com>)得到。Microsoft's XML Notepad 可在站点:<http://www.microsoft.com/xml/notepad/intro.asp> 得到。当使用这些浏览器提供的解析器时,XML Netopad 需要用户安装了 Internet Explorer 4.0 或 5.0 for Windows 的软件。

更多的编辑器即将出现。SoftQuad 流行的 HotMetal HTML 编辑器的生产者,宣布了 Xmetal 计划,它是使用类似界面的 XML 编辑器。包括著名的 Adobe's FrameMaker 和 Corel's WorldPerfect,提出了一些设计方案纲要,包括 XML 的输出 (Microsoft's Office 2000 将在某种程度上使用,但是不是直接作为文档的输出格式。带有嵌入 XML 的 HTML 似乎是 Microsoft's Web 格式的选择)。XML 的出现给开发者提供了新的机会,HTML 提供了产生新的通用工具的方法。

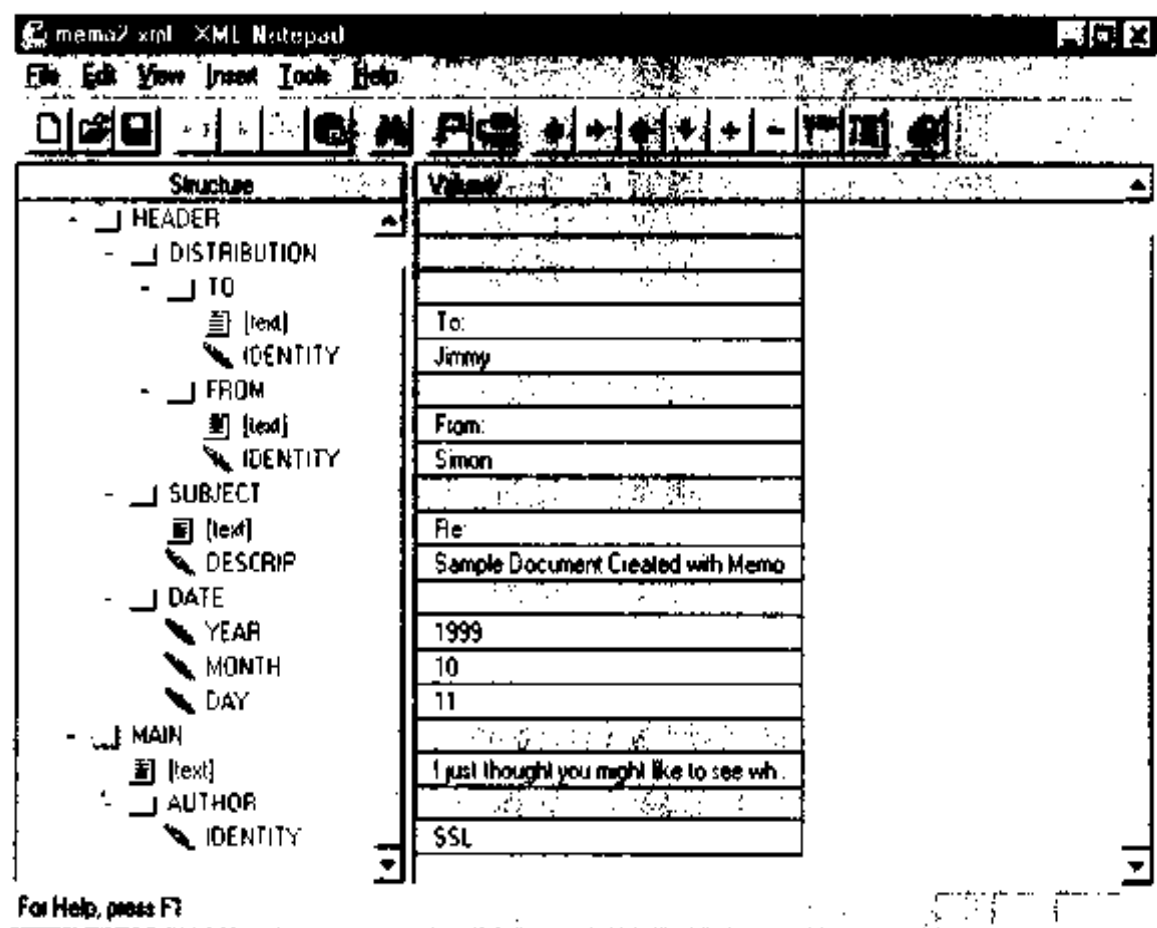


图 12-5 在 Microsoft's XML Notepad 中编辑 XML 文档

集成

有很多部件可以使用 XML,一些正在开发中,另外的一些已经成熟。让这些部件在一起工作,特别是那些最初为 XML 的前期技术 SGML 和 HTML 设计的部件,可能是一个挑战。如果有必要,集成者可以建立他们自己的连接部件的部件,象在下一节描述的。但是现在我们将要探讨使用现有的技术,在最简单的例子中,如图 12-6 所示,可以表示 XML 的浏览器(使用 CSS 或 XSL)连结到 Web 服务器,它重新得到 XML 文件和样式表,然后把它们发送到浏览器。文件用传统的文件服务器和 FTP 结构加到 Web 服务器。

注意

虽然浏览器是这个 XML 转送的客户端,任何网关-接收数据库、一套仪器、一个电子邮件系统或完全是新的东西,都可以在连结的末端。XML 客户端决不仅限于浏览器(浏览器可以提供大家熟悉的结构,简化了讨论)。

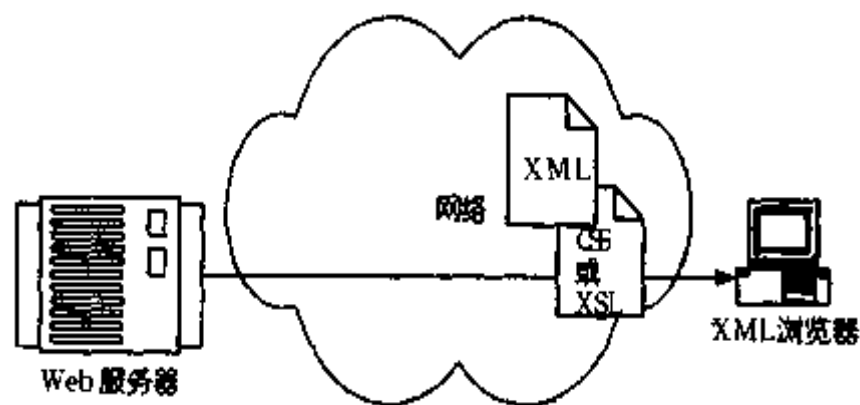


图 12-6 在 Web 上观察 XML 文档

当心

当前没有任何一个浏览器支持 Xlink。(毕竟它只是工作草图)这使得创建包含多 XML 页面的站点没有任何可能,因为链接没有出现。以下的讨论假设 Xlink 支持浏览器。

在另一个例子中,典型的 XML 文档通过 Web 发布,在 Web 服务器上的处理器把 XML 转换成 HTML 用于显示,如图 12-7 所示。

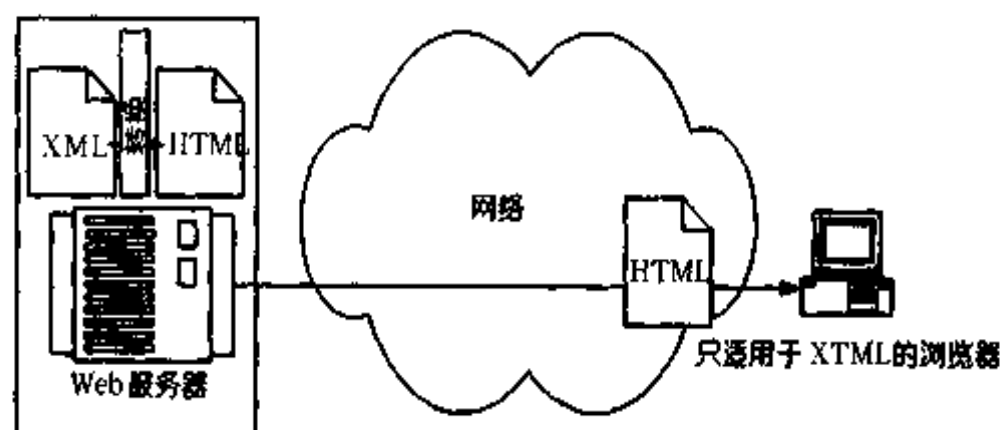


图 12-7 使用服务器端的处理,把 XML 文档转换为 HTML 文档用于 Web 传输

当心

如果你用服务器端的处理(用 XSL 或其它工具)把你的 XML 转换成 HTML,要确保有某种高速缓存结构,避免每次下载文档,都要执行转换。站点的性能可能会受到很大损坏,除非在每一步中你都增大效率。

服务器端的处理可以不管 XML 到其它格式的转换;它可以简单地包括把以非 XML 格式存储的信息转化为 XML 文档。这个任务可以让传统的程序来完成,如

CGI 处理、Active Server Pages 或者只要合适的其它程序,如图-12-8 所示。

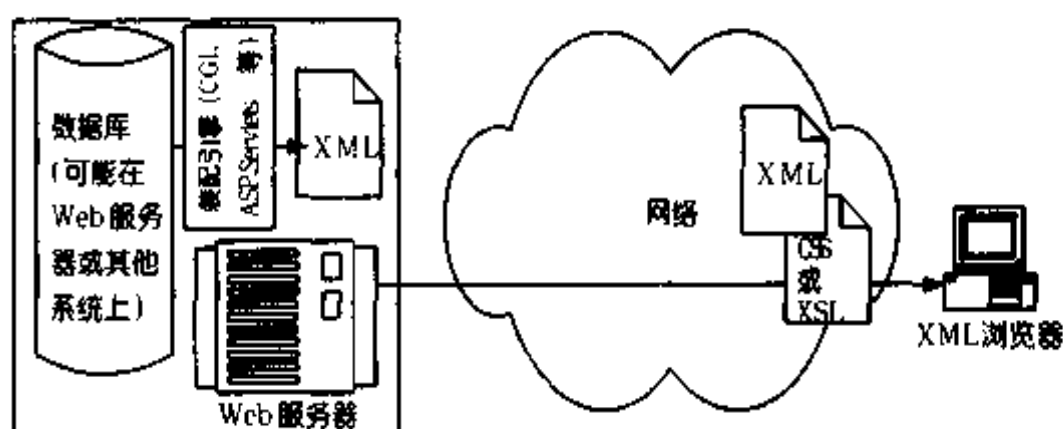


图 12-8 使用服务器端处理装配 XML 文档

额外的处理(安全管理、高速缓存、进一步的转化等等)可以发生在仓库和阅读器/编辑器之间的任何一处。如果它是重复发生的,如图 12-9 所示,我们可以发现传输 XML 比 HTTP 是更有效的途径,至少在 HTTP 把信息传到客户计算机的这点上是如此。

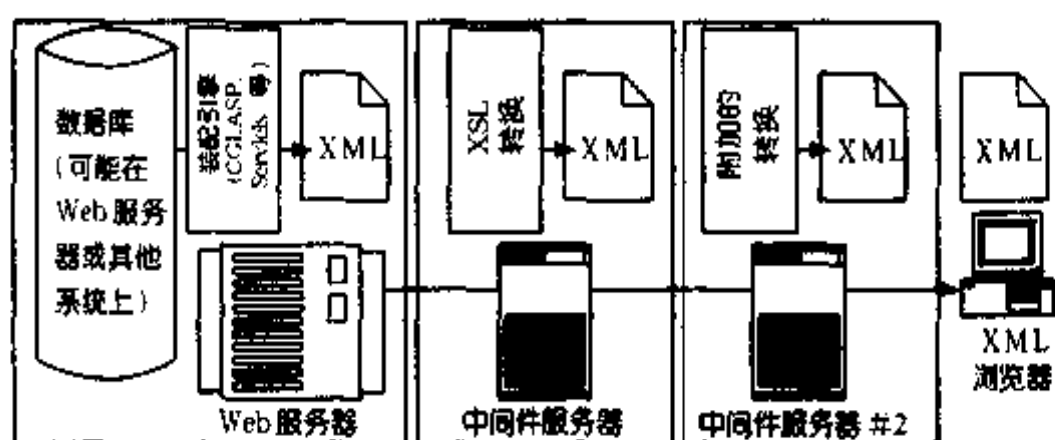


图 12-9 在仓库和阅读器之间处理 XML

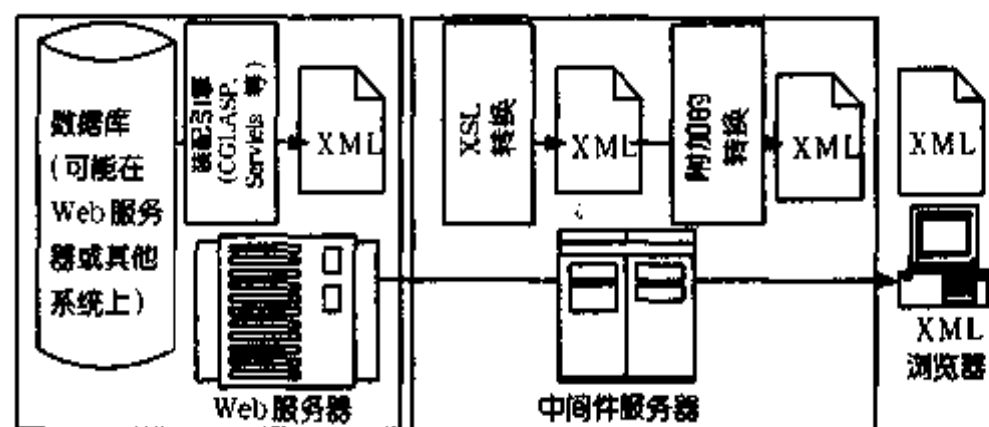


图 12-10 结合 XML 处理减少处理 XML 所需的解析和传输数目

由于 XML 的容易处理的结构,有许多简化这些复杂模型的方法。如果处理在单一的环境中发生,繁重的解析和装配任务经常可以跳过。处理,特别是象 XSL 转换的通用的处理,可以在服务器、客户端或它们之间的任何适当地方完成(当然它也根据支持处理的环境)在图 12-9 所示的处理,也可以用一套较小的系统处理,如图

12-10 所示。

在解决 XML 的灵活性时,产生了许多问题。但是简化 XML 处理使它成为集成的良好工具,无论是处理对你的需要最有效的还是最适当的序列。

创建用户的 XML 应用程序

虽然我们尽量采用 XML 的功能使文档便于人来阅读,真正使 XML 令人激动的原因是 XML 与机器可读性的关系。设计标记是为了便于编程,使用嵌套结构有利于递归作用和面向对象编程。虽然解析有效的 XML 文档不是一个轻松得任务,但解析其它格式的文档更是面对一个巨大的挑战。XML 规范说明的开发者通过严格 XML 文档结构语法的规则,使创建 XML 应用程序变得更容易。同时通过加入合适的构成选项,减小了所需的大量处理。XML 厂商要求所有的元素必须完成 start-和 end-标签,或者通过用关闭标签 `</>` 表示它们是空的,使得编写解析器更容易。SGML 和 HTML 都允许元素跳过结尾标签,它需要有意义的代码决定那里是元素的真正结尾。SGML 也允许元素的缩写名字,但要在解析处理中加入额外的查找级别。XML 的基本结构和文档的表示标准,使解析器可以是一个简单的程序,对文档处理不用增加很多的操作。

注意

虽然 XML 文档使编写解析器是很容易的事,但仍有很多工作要做。大多数应用程序依靠其他开发者创建的解析器模型,而不是他们自己的解析器。

验证 XML 文档仍然是一个挑战,由于参数实体和需要面对 DTD 检查元素结构。DTDs 可以是一个难于解释的,非常复杂的文档,特别是通过几个文件向后扩展,因为多参数实体和 DOCTYPE 声明(没有提到 IGNORE 和 INCLUDE 部分)。把大的 DTDs 应用到小的文件浪费了处理器的循环,同时解析器要解释它从来不使用的额外信息,增加了查找每个元素的操作。验证文档是 XML 处理及简化其它应用程序处理的关键部分。有效性可能发生在文档寿命中的不同点,从最初的创建,到最后的显示。

许多 XML 应用程序可能使用 XML 部分结束。处理 XML 链接的应用程序可能需要做一些验证,除非程序员想要表示,创建每一个单独元素实例链接所需的属性。但是他们确实需要能扩大他们实体参考的解析器。实践与有效性两个标准之间的分歧仍可以看到,但是更多的标准有可能出现。有许多情况不适和完全的 DTD。把文档的极端灵活性和在有效文档中的强大工具结合,名字空间处理的复杂性可能为开

发者引起一些问题。管理系统之间的信息流动,需要最大的有效性和最大的可靠性,这暂时还是一个挑战。

交叉参考



XML-Dev 邮件列表是开发者创建解析器和 XML 应用程序的主要论坛。关于列表和参加论坛信息的档案文件可以在站点:<http://www.lists.ic.ac.uk/hypermail/xml-dev/>得到。当要获取这些列表时,要记住,这些邮件列表是为了高水平的开发提供得,不是 XML 入门指南。开发者要注意的其它三个列表是 xml-app、java-xml-interest 和 Perl-xml。订阅这些邮件列表的表单在 Elliotte Rusty Harold's Café Con Leche 站点,它的地址是:<http://metalab.unc.edu/xml/maillinglists.html>。

编写 XML 程序的工具

XML 不是编程语言;在 XML 中,从开发 XML 中心应用程序到开发简单的内核和语言,都没有任何限制。虽然 W3C 文档对象模型(DOM)指定对 Java 和 JavaScript 捆绑,许多其它的环境,从 C++ 到 Perl、Delphi、Visual Basic 和 Lisp,可能适合你的要求。XML 甚至可以用在完全不同环境中编写的多种程序之间的胶合剂,使它们无须在指定的部件之间创建界面,就可以来回处理数据。只要它们都可以处理 XML,它们就可以使用它作为通用数据格式。

关于这一点,XML 开发的主要竞争者似乎是 Java。从简单的理由来看,Java 比其它语言和开发工具有明显的优势:象 XML 一样,它是从基础开始按统一字符编码标准(Unicode)创建,需要解析器可以处理因使用 C、C++、老版本的 Perl 和其它单字节字符开发工具所开发的应用程序,进行完全两字节的 Unicode 规范编码而引起的问题。作为结果,当前进行的许多 XML 开发工作都是在 Java 中做的。Java 的结构也能和 XML 的结构很好地匹配,使用层次结构容易和 XML 兼容。Java 也提供分类和对象间的简单界面,可以容易地把通用解析器加到数据处理应用程序中。即使 Java 的工具不是最高级的,它们也可以匹配 XML 解析所需的处理级别。Java 程序也适合几个以 Web 为基础的 XML。Java 程序提供显示浏览器中 XML 内容的方便结构,浏览器本身并不理解 XML,Java 填补了这一重要得空白。

Java 还有另外几个优点,包括共同有效的开发框架结构。除了 DOM 和它与 Java 的捆绑,以 Java 为基础的 SAX API(<http://www.megginson.com/SAX>)为以事件流表示 XML 文档,提供许多 XML 工具的标准基础。SAXON(<http://home.iclweb.com/ic12/mhkay/saxon.html>)和 MDSAX(<http://www.jxml.com/mdsax>)提供两个不同的框架结构,开发者可以用来扩展 SAX,为处理和显示 XML 创建 Java 应用程序。IBM's alpha-

Works(<http://www.alphaWork.ibm.com>)不断推出用于解析和处理 XML 的新 Java 工具。C++ 也是 XML 开发的一个可行的环境。象 Java 一样,它的对象结构可以很容易包含嵌套的元素结构。即使在 C++ 项目中加入分类,也只是某些方面比 Java 这样做复杂一些,决不会缺少功能强大的 C++ 工具。C++ 已经广泛应用到各种数据处理项目,包括标记处理。数据库也是可使用的。不幸的是,大多数 C++ 仍然使用单字节字符,使它很难和 Unicode 兼容。用 UTF-8(用它的 ASCII 子集)编码的文档和大多数标准 flavor C++ 兼容,更多的用于 C++ 的 Unicode 开发的工具正在不断出现。

注意

Unicode 关系到任何方面吗? 答案依赖你的需要和你所运行的平台。Unicode 由于有限的应用程序支持,发展很慢。然而 XML 和 Java 在根本上支持处理 Unicode 字符。Unicode 已经成为主流,Microsoft's Windows NT 和 Sun Solaris 2.6 操作系统都支持它。Java 和 XML 又是作为将来文档处理的两个主要部件,因此有希望在以后看到,Unicode 将起到更大的作用。

Perl 是电脑黑客这几年选择的文本,帮助开发者用少数几行代码炸掉了似乎是不可能逾越的障碍。Perl 支持许多丰富的正规表示,帮助无数程序员创建了 CGI 脚本,编写 HTML 和解释被表单送回的数据。同时,Perl 帮助开发者完成改变整个站点,向在一夜之间轻松改变站点上的所有合法布告栏的这一难题进行挑战。Perl 的使用几乎不对 HTML 作限制,SGML 开发者使用 Perl 寻找他们文档中的问题并尽可能地自动修复它们。Perl 依靠外部解析器给它提供 XML 信息。最初的用于 XML 解析的 Perl 模型,XML::Parser,是由 Larry Wall 和 Clark Cooper 编写,使用 James Clark's C-based Expat 解析器,自从 1998 年 10 月便开始使用。除了 XML::Parser,新模型也在不断开发中,把 Perl 连结到 XML 处理的更多方面。需要最新信息,查看 CPAN 档案:<http://www.perl.org>。

Python,在 UNIX 中,是带有根的另一种脚本语言,它已经成为编写 XML 脚本的流行工具。资源包括一个 Python 解析器,一个 SAX 执行工具和各种其它工具,需要详细信息访问站点:<http://www.stud.ifi.uio.no/~larsga/download/python/xml/index.html>。Python 是 XML 众多开发者选择的语言,支持 Python 的工具也会在不远的将来迅猛增长。

交叉参考



如需要发现一些有关 XML 方面的工具,改善开发环境,可访问站点 <http://www.stud.ifi.uio.no/~larsga/linker/XMLtools.html> 寻找免费工具,定期更新包括商业产品的工具列表的站点是:<http://xmlsoftware.com>。

Web 应用程序。Lotus Domino 从 Note 服务器转变为 Web 服务器是值得关注的,它直接把 Note 格式的文档转换成 Web 页面。把 Note 后面的复杂数据结构,应用到公众或私人 Web 上。

然而,在数据库驱动的站点有几个问题,第一,它们需要更强的功能来克服连接到数据库的内务操作;或者数据库服务器需要更强的功能处理对它的日益增长的需求。第二,数据库驱动站点很少是友好的搜索引擎式的;它们中的大多数的前面都有“不要进入”的记号,记号使用的是第七章讨论过的 robots.txt 文件。虽然包含在数据库中的信息有很好的结构和易于搜索,但是对搜索引擎来说,没有容易的方法把它和数据库和收集的结构化数据相联结(它也表明有大量的安全漏洞)。最后,复杂的数据库驱动站点,通常需要相当专业的开发队伍,创建管理数据库的应用程序,除了通常的 HTML 开发者队伍外,还有昂贵的 Web 项目开发队伍。

层叠样式表和动态 HTML 的缓慢发展传播,已经对站点的体系结构产生影响。Web 站点设计的某些方面正在变得更普通。在这个模型中的样式表可以在一个位置上控制,使公司可以提供对他们站点的基本概貌,便于以后修改站点。动态 HTML 界面可以以 JavaScript 代码文件或 Microsoft 新脚本类程序存储,它把脚本和 HTML 结合创建可重复使用的界面控件。今天的 HTML 文档远不只是带有标记的文本。当前可以出现在 Web 页面的项目花名册包括:

- HTML
- Images(GIF、JPEG、PNG、XBM 等)
- Sound (AIEF、AU、WAV 等)
- Video (QuickTime、MPEG、AVI 等)
- 指定插件程序内容(Splash、Shockware、Acrobat 等)
- JavaScript
- VBScript (Internet Explorer)
- Java 小应用程序
- ActiveX 控件 (Internet Explorer)

Web 已经是一个具有良好条件的编程环境,它不断改善编程工具。包括 Web 开发者在内的许多人都认为 Web 已足够复杂了。在这点来看,把 XML 加到 Web 中是不必要的。

变迁的体系结构

把 XML 加到现存的 Web 结构中不是特别困难,就象前几章概述的那样。现存的 Web 服务器和 HTTP 协议可以象它们处理 HTML 那样处理 XML,不用知道它们

传输的文件。许多 HTML 周围的应用程序结构可以重新被 XML 使用,不过需要密切注意文档产生的语法和词表,但是不会有其它太多的变化。虽然这个方案听起来很有希望,但仍有一些主要问题。正式发布的浏览器在开始时表示支持 XML,但在执行层次样式表时有不一致的地方。从 HTML 到 XML 的转换过程中,需要一些处理部件知道 XML——表示信息的浏览器,或发送信息的服务器。

注意

正象在前面章节注解的,缺少 Xlink 是 XML 在 Web 上应用的另一个重要障碍。W3C 正在提出几个建议,它们只用简单的链接结构,暗示前面至少有一条路可以解决这些问题。

Netscape 和 Microsoft 都正在修改他们的浏览器来接受 XML。在我写这本书的时候,两个公司都发布了能够用层次样式表显示 XML 文档的软件。虽然 Netscape 的这个软件当前只是预先展示,还没有准备大规模发行,Netscape's Gecko 专著于用层次样式表的支持,显示 HTML 和 XML 信息。Microsoft's Internet Explorer 5.0 包括早期的 XSL(除层次样式表外)也支持 XML,另外在以 XML 为基础的向量标记语言(VML)中用于向量图形存储的交付引擎也支持 XML。第三个图形浏览器商家 Opera 还没有宣布支持 XML(而以文本为基础的浏览器象 Lynx 是否响应 XML 仍然还是一个问题)。

即使这些浏览器发布了,用户也需要用几年时间更新他们的旧浏览器。在开发者假定有多少用户使用可以处理 XML 格式的信息之前,有可能要花费几年时间。同时,不是只想把显示数据送到 Web 的开发者,因为用户的软件缺乏 XML 支持,所以必须给用户提供显示以 XML 格式存储信息的工具。转换结构可能是图 12.7 所示的之一,在服务器上的处理器检查浏览器用来识别它功能的请求发送的信息,并在需要时把 XML 文档转换成 HTML 文档。除了它将消耗的额外处理周期,它还意味着开发者和设计者需要在 XML 元素和它的 HTML 等价物之间创建映射。这可以

从 HTML 语法到 XML 语法

当开发者面对把 XML 转换成 HTML 的任务时, W3C 正在向其他方向努力, 用 XML 语法重新编写作为一套模型的 HTML。虽然 W3C 向着以 XML 为基础的通用标准发展, 把 HTML 移到 XML 词表涉及的不仅仅是取得语法上的一致, W3C HTML 经过数年已经发展的很大, 从带有少数属性的一套简单标签到能用 SGML DTD 收集大量元素和属性。HTML 很难创建新 HTML 浏览器, 支持需要许多代码的整个标准, 纯粹是个错误, 设计者为少数主导市场的浏览器创建文档, 没有考虑新开发的浏览器, 会遇到很多的麻烦。非 PC 设备, 象蜂窝电话和个人数字助手, 没有显示或执行全版本的 HTML 所需的处理设备, 或者没有成功处理当前 HTML 开发中使用的各种语法的处理设备。忙于这些需求, W3C HTML Activity 正向下一个 HTML 版本, 代码命名“Voyager”努力。象过去的版本一样, 不是加入新特性, 工作组把 HTML 分解成较小的词表(模块), 并把它转变为 XML 词表。以 RDF 为基础的简档文件(profile)系统将帮助设备识别到服务器的能力, 使服务器有可能根据所接设备的能力发送相应内容, 使这个模块更有用。

交叉参考



本节材料必定会随着 Voyager 技术说明而发生重要变化。需要最新有关 W3C 用 HTML 做什么的高层观点, 可访问站点: <http://www.w3.org/MarkUp/Activity.html>。需要最新的可以拼写出新模型和它的文档创建含义的工作草图, 可访问站点: <http://www.w3.org/TR/WR-html-in-xml>。

工作草图描述了 HTML 开发者需要了解的语法变化, 以适应新的以 XML 为基础模型。在很大程度上, 它们中的所有内容都包含在这本书中, 重要的例外是 HTML 元素和属性名, 从现在起它们将用小写字母表示。文档需要很好的结构, 需要脚本和样式元素都采用 CDATA 部分(如果它们包括 <, >, 或 & 字符使用外部脚本), 空元素必需用 /> 语法, 虽然说明书推荐使用在斜线前加空格(象 br />)以便和老的浏览器兼容。一些元素(值得注意的是 title 和 base)需要出现在特殊的序列中和 Voyager DTDs 相符。Voyager 使用 XML 的名字段告诉处理器那个词表在使用, 大多数根据不需要前缀的缺省名字段, 保持了和老浏览器的一致。创建使用 Voyager 的 HTML 无论是对 HTML 浏览器还是对使用样式表作为格式化的通用 XML 处理器都是有用的。当前的工作草图识别 16 个模块, 列表在下面的表 13-1 中。

表 13-1 第 16 Voyager 模块

模块	模块中的元素
Base	html, head, title, base, meta, link, body, h1, h2, h3, h4, h5, h6, p, br, a, bdo, span, div.
Transitional	Basefont, font, center, s, and u. (also contains border, align, and noshade attributes)
Style	style, link (for use with style sheets), and style attribute.
Script	Script, noscript.
Font	tt, b, i, big, small
Phrase	abbr, acronym, address, blockquote, cite, code, dfn, kbd, q, samp, var.
Inflection	em, pre, strong, sub, sup, hr
Editor	del, ins.
List	dd, dt, dl, li, ol, ul.
Forms	Button, fieldset, form, input, isindex, label, legend, optgroup, option, select, textarea, ..
Table	table, caption, col, colgroup, thead, tbody, tfoot, th, tr, td.
Image	img
Image	Map map, area.
Object	Object, param.
Applet	Applet, param.
Frames	Frameset, frame, iframe, noframes.

象 HTML 4.0 一样, Voyager 为了一致定义多个简档文件。Voyager 严格的简档文件匹配 HTML 4.0 严格的简档文件, 而 Voyager 松散是用来把文档从 HTML 4.0 转换, Voyager-frameset 是使用来自 HTML 4.0 frameset 的文档。一旦 HTML 模块化, 其它的 XML 模块象 MathML、MusicML 或者结构化向量图形, 都将很容易适合 HTML 文档。结合设备简档文件(它是在模块化过程中单独的部分, 但是也很重要), HTML 最新的开发适合有许多 XML 信息类型的集装箱。

在 Web 上 XML 的含义

不管 Web 浏览器和 Web 服务器是否进行过我在前面章节提出的转换, 随着

XML 文档在浏览器中显示的出现,有可能改变许多站点下面的体系结构。XML 语法本身和 Xlink 定义将驱动这些在结构和设计上的不同变化。XML 语法允许产生内容、显示和脚本彼此都是分离的 Web,这与在 HTML 中的明显不同,Xlink 通过为文档查找提供丰富的界面和分段模型来区别它自己。

XML 继续趋于把内容与从 HTML 中获取的一些要素分离。层次样式表对许多设计者最有吸引力的方面是它的中心化样式信息,避免了创建和更新 HTML 页面的许多重复性的工作。样式表是图形设计者说明书的自动版本,为了使信息流进入文档,提供顺畅的路径,而不用不断地人工干预。CSS 对应用程序开发也有一些优点,为那些可以应用到元素的提供格式化结构,不用管操作目标的元素类型。

注意



对这个“结构和格式化分离”争论的有趣的对应物是使用 XML 作为标记,象格式化 XSL 的对象那样,纯粹表明格式化。XML 文档可以从结构化文档链接到

元素不友好,对任何包括标记字符在内的代码,需要 CDATA 标记部分,象在第七章讨论的那样。即使开发者可以处理这些要求,XML 也作为潜在通用文件格式,可以进一步促使代码从内容中分离。不是所有的应用程序都使用脚本,一些程序需要从根本上忽略脚本内容。把脚本从内容中分离将会减少在这些应用程序中的数据装入量。虽然 HTML 开发者必需学会记录所有的图象、Java 小应用程序、控件和链接到文档的其它内容材料,但是,从 HTML 集成模型到 XML 模块化结构的改变,开始肯定会引起一些问题。要改变文档,首先需要严格检查文档和他们的结构来决定在多文档中共享什么合适。在理想情况下,一套象在图 13-1 所示那样存储 HTML 文档,可以用图 13-2 中显示的工具重新组织。

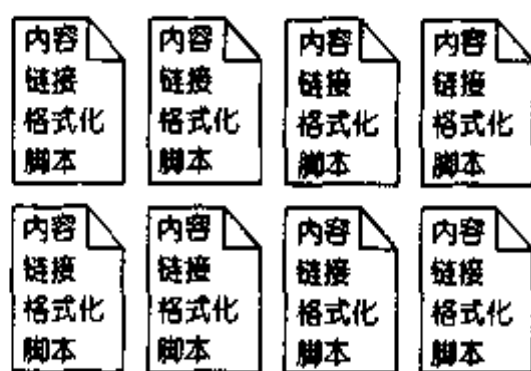


图 13-1 HTML 最初的集成模型趋于在同一个文档中保留内容、链接、格式化和脚本

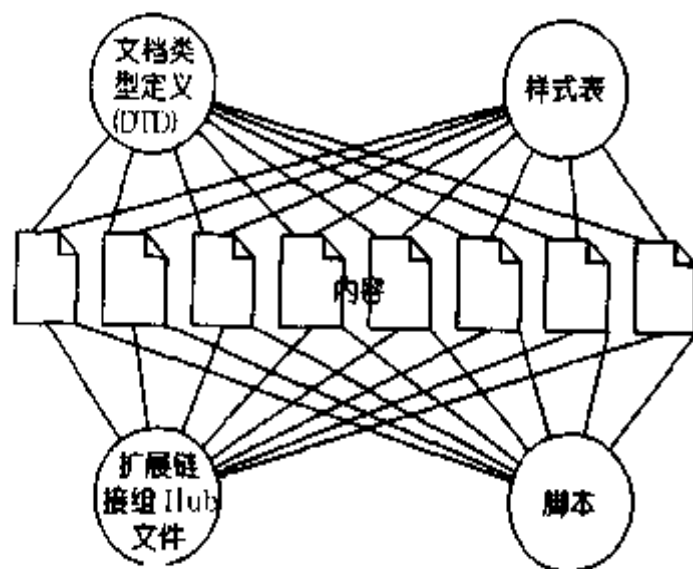


图 13-2 XML 链接结构和模块化促使文档分解为内容和一套共享部件

记录这些关系并使它们有效工作,需要产生另外的工具。目前,Web 创建的工具可以跟上 HTML 文档、图形和少数部件的发展,但是这些工具中很少能以这种方法管理脚本和样式表,且扩展链接组给任务加入了全新的维数。在工具能满足需要前,如果只因为避免要求管理员记录所有部分,那么开发者就可能继续使用混合方法,匹配内容、格式、脚本和链接。Java 小应用程序和 ActiveX 控件仍然对部件起着和现在所起类似的作用,虽然它们被不同的方法启动,正如我们在第十二章所看到的那样。图形、声音和其它数据文件仍是分离的。站点组织者对工具的要求总是在不断提高。

Xlink 对站点结构的长期影响甚至可能更大。Xlink 使开发者可以用过去越过限制传统文件结构的方法,分解和合并文档。虽然充分利用这些功能,需要使用仓库开发(第十二章有介绍),但是分段和链接给开发者提供考虑他们文档的新方法。由于在文件系统中的基础,HTML 文档必须作为一个完整的单位出现,人们可以读文件的一部分,但是没有办法下载文件的一部分。作为结果,大多数开发者就创建了一套较小的文档,它装入速度快,但难于管理。打印分布在一百多个 HTML 页面上的文档的人,直接感受到了这种方法的限制。

分段和链接使它可以参考文档的部分,并把它们嵌入到其它文档创建新文档成为可能。分段和链接有潜在的使当前工作的数据库驱动应用程序失去作用的功能,开发者通过重新使用数据,包括它在多文档的数据,而不用编写专用的应用程序。例如,目录可以在一个大文档中存放它的所有信息,除了记录项目,不需要做任何事情。目录的页面可以参考带有 XPointer 的目录重新找回以属性和内容为基础的信息块。作为结果,模板可以链接到所有待售项,所有的体育用品、客户需要的特殊装饰卡、甚至在目录中所有合理的请求的项目。所有这样的信息可以只用项目所需的部分适当传输。

使分段和链接高效工作需要一整套技术:服务器能有效地处理 XPointer,并适当返回分段,浏览器能够处理 Xlink 指定使链接内容嵌入,编辑工具能帮助站点管理者记录存储在这些新文档结构中的数据。

作为应用程序体系结构的 Web 浏览器

即使 Web 浏览器迅速成为流行界面,它也不能提供给用户所期望的服务。最近一轮的 Web 浏览器和服务器的性能的改善,鼓舞了声称 Web 将成为所有数据类型的通用界面的说法。虽然 Java 在几年前才出现,但是 Java 的发展最后达到了 Java 应用程序可以和成熟操作系统的具体应用程序相竞争的地步。动态 HTML 和文档对象模型(DOM)使在 Web 浏览器内创建精美的界面成为可能,它提供很多功能,不只是表单字段、上下对话框、可点击按钮和图形。脚本语言,特别是 JavaScript,已经成熟,需要面向对象扩展以及从有效性认证到界面管理的一系列改善。在脚本语言成长的过程中,XML 出现了,XML 并把在六年迅速发展中产生的一些混乱清理掉,为所有这些技术提供了坚实的基础。

XML 给 Web 提供真正的潜力,成为信息交换的工具,或成为管理文档的工具——不只是在我们所熟悉的 Web 站点的上下文。但是在用于商务对商务的电子数据交换(EDI)领域,有可能使用多用户共享的新系统来代替熟悉的文件系统,而记录文本,管理他们之间的连结。在这两个例子中,Web 用户和 EDI,以及文档管理系统用户肯定会有一些收获。对于 Web,它将表明连结兆字节信息的混合系统已经成

熟,有一系列功能强大可靠的协议,给有 Web 经历的商家进入额外的市场,提供了新的机会。对 EDI 和文档管理系统,将会发生很大的变化,从专用的系统到部件组的世界,创建系统变成了连接准备好的工具,而不是手工编写复杂的应用程序。在前面章节中所描述的体系结构,仓库、处理器和网关,它们都有很好的可扩展应用程序,超出了传统 Web 分布计算的所有结构空间。这些体系结构是大量不同应用程序的稳定基础。

然而 XML 确实有它的限制。XML 不可能把它的浏览器作成更好的用于创建图形编辑器或视频游戏的界面。XML 不能解决所有的问题。二进制数据的长字符串,象声音、图象和视频都不易使用 XML 格式。当然 XML 对项目处理结构化的信息(从 CAD /CAM 到文档管理)是一个比较好的方法。XML 的结构灵活足够存储层次、表格、电子数据表信息和复杂文档。开发能象我们读 XML 文档那样容易,来编写 XML 文档的界面,需要花费几年的时间以及改善其它关键标准。把 Web 从显示引擎变为主引擎,将需要许多步骤和数年的开发。XML 可能发现“杀手应用程序”,并把它快速移到程序前面,或许它还需要几年的时间渗透。

XML 和 Web 的未来

即使许多 HTML 开发者可能更喜欢 XML 呆在 Web 服务的独立世界中。虽然一些 XML 组看起来对今天在 Web 上使用的工具不友好,但是它不会总是如此。XML 是从 HTML 模型中剧烈变化产生的,但是它仍有一些地方似乎让人感到很熟悉。HTML 开发者花费几年时间尽力解释 Netscape 和 Microsoft 的最新标签,现在他们专注于创建他们自己的标签系统和伴随样式表,不需要严格符合单一的规则。客户-服务器开发者努力和复杂的工具奋斗,但是没能使他们做的 Web 看起来像他们希望的那样,有伴随文档对象模型的 XML 不是他们最好的选择,XML、样式表和 Java 都有它们自己的一套规则,但是它们允许的额外的灵活性,使不同的工作组相信,把灵活性聚在一起形成新的数据交换标准。如果幸运,XML 也促使工具中的读、写、处理和共享数据这些方面发生变化。

尽管它面临的挑战,XML 很可能会成功。SGML 组提供了 应用程序和支持的最初基础,而 Microsoft、Netscape 和 Sun 许诺它有光明的未来。迄今为止,W3C 为 XML 提供了中心,在这里竞争者可以参加讨论,制定统一的标准,并准备执行它。如果 XML 继续象它以前那样发展,它将很快找到合适的体系结构,使 Web 最终能象它许诺的那样,转送方便、友好、便宜和交互式的信息通路。

名词解释

Active Server Pages

微软创建的服务器端工具,它由标记把脚本与对象结合起来,用于创建动态 Web 站点。

Anceter

包含另一个元素的元素,甚至还包含包含元素的元素。例如,根元素是 XML 文档中所有元素的祖先。在<FIRST><SECOND>-<THIRD />< /SECOND>< /FIRST>中,FIRST 元素是 SECOND 和 THIRD 元素的祖先。

Application programming interface(API)

编程人员用于创建内部操作所用软件的说明。

Application

做与 XML 有关的(视图、格式、分类、输入等等)程序或与 XML 创建的标记集有关的程序。例如,HTML 是 SGML 的应用程序,由 SGML DTD(不久会成为 XML 的应用程序)。

ASP

见 Active Server Pages。

attribute declaration

文档类型定义内的声明,用于识别特殊元素所使用的特殊标记,也可以用于识别它的类型,提供默认(固定)值,并限制它所允许的取值。

Attribute

有关元素的更详细的信息源。Attribute 值可以在 DTD 中固定,或者作为元素起始标记中的名称值列出。

C++

通常用于建立高性能软件所使用的面向对象的语言。

Cascading Style Sheets(CSS)

提供给使用包含在<STYLE>标记和 STYLE attribute 中信息的元素的格式控制的标准。它比 XSL 要弱的多,不会有太大的前途。

Case-sensitive

Case-sensitivity(大小写敏感)意味着字符必须完全匹配。非大小写敏感应用程序把小写和大写字符都转换成一种形式,其认为 THIS 和 this 是一样处理的。XML 中的几乎全部内容(除了 xml:lang 属性值)都是大小写敏感的。

CDATA

见 character data。

CDF

见 Channel Definition Format。

CGI

见 Common Gateway Interface。

Channel Definition Format(CDF)

基于 XML 的“推送”标准,描述包含有关自动检索时的描述、图标和信息的文档。

Character data(CDATA)

CDATA 在 XML 中有两种不同的意思。首先,CDATA 用于属性声明中,以表示属性可以字符(非枚举文本)内容,包含能扩展的实体。其次,CDATA 部分使用标记(<! CDATA[and])以表示纯字符数据文档中的文本,没有包含元素和实体。CDATA 部分为其包含需要正常处理的字符(典型的是,<、>和 &)文档的作者提供了一个“逃避”的机理。

Child element

嵌入在另一个元素中的元素。在<First><SECOND />.< /FIRST>中,SECOND 元素是 FIRST 元素的子元素。

Common Gateway Interface(CGI)

Web 服务器常用于把用户请求连接到服务器上正在运行的程序上的接口。

Content

元素或属性内的文本和标记。

content model

描述可以出现在元素内的内容的一组规则,专门用于元素声明。

CSS

见 Cascading Style Sheets。

descendant

由另一个元素所包含的元素,甚至中间还可以包含别的元素。例如,在 XML 文档中所有的元素都是根元素的子孙,在 `<FIRST><SECOND><THIRD /></SECOND></FIRST>` 中,THIRD 元素就是 SECOND 和 FIRST 元素的子孙。

Document object model(DOM)

处理应用程序或脚本文档中的元素和属性的一种手段。W3D Document Object Model Working Group(W3D 文档对象模型工作组)正在为 HTML 和 XML 文档开发一个标准模型。第一级(Level One)是由 W3D 推出的;第二级(Level Two)正在开发中。

Document Style Semantics and Specification Language(DSSSL)

用于 SGML 文档处理和格式化的转换和样式语言。XSL 的处理器。

Document type declaration

在合法的文档中,用于把文档连接到它的文档类型定义中的声明。该声明可以连接到外部文件或包括在它本身中的定义上。

Document type definition(DTD)

用于 SGML 开发核心的文档建设以及所有合法 XML 文档建设的一组规则。处理应用程序和编辑工具都依赖 DTD 去通知特殊文档类型所需要那部分。

Document

“文本对象”。在 HTML 中,文档(或“页”)是包含 HTML 的单个文件。在 XML 中,文档可以包含几个文件(或其他源)内容,并且应该包含使它合法或具有良好结构的标记结构。

DOM

见 Document Object Model。

EBNF

见 Extended Backus-Naur Form。

ECMAScript

JavaScript 语法的标准,由欧洲计算机制造商联合会(European Computer Manufacturers Association)在 ECMA-262 中进行说明。

Element declaration

识别元素名和描述它内容模型的声明(出现在文档类型定义中)。

Element

XML 文档的基本逻辑单元。XML 文档包含几个在根元素后的打开声明,这几个声明可以包含其他元素和内容。

Empty element

没有文本内容的元素。空元素可以由放在紧邻(<EMPTY></EMPTY>)的开始标签和结束标签,以及用 />(<EMPTY />)结束的开始标签表示。空元素仅可以包含属性。

Encoding declaration

在 XML 文档开始的 XML 声明中所做的编码声明,用于指定该文档所使用的映射到字符的数值。

End-tag

关闭元素的标签。结束标签跟在句法 </Name> 的后面,在句法 <Name> 中,Name 匹配声明在开始标签中的元素名。

Entity

对经常作为 abbreviation 或快捷键的其他数据的引用。通过声明实体,开发人员就可以避免在文档或 DTD 中重复输入相同的信息。

Extended Backus-Naur Form(EBNF)

用于 XML1.0 中表达 XML 语法的正式表示法。EBNF 也用于描述不合适的 XML DTD 表示法标准,诸如资源描述构架工作(Resource Description Framework)。

extended link group

为链接到帮助建立两种形式的链接而分析其内容的一组文档,而不需要在每个文档中声明。

Extended link

包含定位元素而不是简单的 HREF 属性的以识别链接目标的链接。

Extensible Markup Language(XML)

由 W3C 创建的标准,它为标记所提供的一组规则比 SGML 更加简单,而比 HTML 更加灵活。

Extensible Protocol

Internet 工程任务组(Internet Engineering Task Force,缩写为 IETF)的开发建议,把 XML 应用到网络通讯的协议。

Extensible Style Language(XSL)

W3C 开发所用的样式表标准。XSL 可使开发人员所指定的格式化要比级联样式表(Cascading Style Sheets)所允许的更加精确。XSL 似乎是保证,但还不是 W3C 的工作草案或建议。

External DTD subset

保存在文档外的文档类型定义部分。外部的 DTD 对于保存由多文档使用的文档类型定义来说非常方便,并使它们共享中央管理的定义。

External entity

其声明包含对外部资源的引用的实体,而不包含实体内容本身。

Fatal error

解析器停止处理文档并向应用程序报告错误(提示:解析器的致命错误并不是应用程序所必需的致命错误)。

filter

在把文档事件传给应用程序之前,用于处理或转换文档事件一小段基于事件的软件。

Formatting object

在 XSL 中,用于描述表达方式的一组工具。源 XML 文档被转换成一组格式化对象,然后被显示或被处理。

Fragment

由 Xpointer 或其他查询机制所识别的文档部分。Fragment 可以是一个元素以及它的全部内容(包括子元素)、一组元素、甚至可以是基于内容的选择。W3C XML Activity 有一个专门处理 fragment hiding 的工作组。

General entity

用于文档内容中的实体。当用在文档中时,常用实体名前面必须放一个 & 号,并在名字后加一个分号“;”。

Generalized Markup Language(GML)

SGML 的前身是由 IBM 在 Charles Goldfarb 领导下于 1969 年开发的,GML 原来为标记使用 <、>、/ 符号,这些符号现在仍然用于文档应用程序中。

Group element

包含一组附加 XML 文件的 Xlink,这些 XML 文件是多方位链接的潜在后选对象。

Hypermedia /Time-based Structuring Language(Hytime)

一组多媒体和 SGML 的链接扩展,正式名称为 ISO /IEC 10744-1992。HyTime 是 XMLLINK 的基础之一。

HyperText Markup Language(HTML)

目前最大众化的标记语言,HTML 是 SGML 的一个应用程序。HTML 是 Web 开发基础之一,通过浏览器应用程序,为文档的表达提供了格式和基本结构。

Hypertext Transfer Protocol(HTTP)

WWW 上控制客户与服务器通讯的协议。HTTP 使客户向服务器发送请求,服务器回答以合适的文档或错误的消息。

In-line link

进行链接声明的元素是链接本身的一部分。

Instance

文档中元素的实际使用或文档类型,与其定义正好相反。实例也可以与整个文档有关,如果文档可以在 DTD 下被 validated 的话,那么它就可以是 DTD 的实例。

Internal DTD subset

出现在文档内的文档类型定义部分。内部 DTD 子集管理起来有难度,但是给开发人员提供了一个测试新特性或开发 DTD 的简单方法,而不需要用到其他文档。

Internal entity

包含声明里面内容的实体,而与外部资源无关。

ISO

国际标准化组织(The International Organization for Standardization),它设置了与字符集测试到运送容器以及 SGML 的质量过程有关的工业标准。

Java

用于 Internet 上的最佳的面向对象的语言。

JavaScript

用于浏览器的脚本语言,它也可以作为激活服务器页面(Active Server Page)或其他开发环境用在服务器上。抛开字面意思,JavaScript 确实不同于 Java,JavaScript 是标准化的 ECMAScript。

Markup declaration

文档类型声明的内容,它可以用于定义元素、属性、实体和符号。它们指定所给出文档中合法的标记的种类。

Markup

保存在相同文件中的作为内容的结构信息。一般情况下,结构信息与内容是分开的,在元素和实体中是独立的。

Mathematical Markup Language(MathML)

由 W3C 为数学结构(特别是方程式)的表达和通讯而开发的 XML 应用程序。

Meta-Content Framework(MCF)

由 Apple 公司开发的,并由 Netscape 为用户导航而表示多维空间的兆字节数据继续开发的标准。MCF 是资源描述格式(RDF)的若干种输入之一。

Mixed content

包括文本的元素中的内容,可能还包括其他子元素。在 XML 中,混合内容模型限制了约束的选项,一旦一个元素包含文本,那么声明在那个模型中的附加的元素,就可以按照作者的意图以任何次序或多或少地出现。

Name characters

字母、数字、连字符、下划线、逗号和句点。在 XML 中,逗号用于名字间的间隔,如果你不支持这种名字间隔,那么就on使用逗号。

Name token

构成名字字符的字符串。

name

名字必须用字母打头,并且只能包含名字字符。

Namespace

命名信息的附加一级使用统一资源识别器为元素提供唯一名字,用于在多 DTD 中定义信息。

Open source

用于由源代码所发布的软件,并具有修改源代码的授权。有各种形式的授权,对修改和重新分类都有不同的限制。

Out-of-line link

离线(Out-of-line)链接能使开发人员独立于文档的内容而声明链接,离线链接可以出现在各个独立的文件中。

Parameter entity

用于表示文档类型定义中信息的实体。参数实体可以用于把附加的 DTD 内容链接到 DTD 上,或者用于经常重复声明的缩写。

Parent element

嵌着另一个元素的元素。在 `<FIRST><SECOND />< /FIRST>` 中, FIRST 是 SECOND 元素的父元素。

Parsed character data(#PCDATA)

分析字符数据是由解析器为实体和标记检测的文本。分析字符数据不应该包含 `&`、`<` 或 `>` 字符,它们需要用 `&`、`<` 和 `>` 表示。

Parser

通过程序把标记序列流(例如 XML 文件)转换成可访问的输出结构的应用程序。解析器可以执行合法性检查或在处理标记时检查标记的组织的是否合理。Perl 为 UNIX 首次开发的脚本语言,它作为一种通用目的的文本处理语言,已经成为了许多 CGI 应用程序的关键部分。

Processing application

产生由解析器(它可以包括解析器或就是解析器自己)生成输出的应用程序,并用它做一些工作,包括表达、计算或其他适合的事情。

Processing instruction

可使 XML 作者直接把指令发送给处理应用程序的指令,该应用程序可以在

XML 容量之外。处理指令通过在打开标记<后和关闭符号>前的问号符号(<? 指令? >),而区别于正常元素标记。XML 声明本身就是处理指令。

Processor

在 XML 1.0 说明中,有关解析器的另一种软件术语。

Prolog

文档的打开部分,包含需要处理文档的 XML 声明、各种文档类型声明或标记声明。

Python

常用于 XML 应用程序的脚本语言。

Recursion

函数可以自己调用自己的技术。递归编程特别适用于嵌入在标记结构中的分析。

Resource Description Framework(RDF)

由 W3C 使用几种 XML 语法开发的用于保存元数据(有关信息的信息)的标准。

Root element

文档中的第一个元素。任何其他元素都不能包含根元素,并且构成了由分析内嵌元素创建的树结构的基础。

SAX

见 Simple API for XML。

servlet

服务器端的 Java 组件,专门用于生成传送到 Web 上的标记。

Sibling

与另一个元素具有相同父元素的元素。在<A><C />< /A>中,B 和 C 元素由于具有共同的父元素,因此它们是兄弟(sibling)。

Simple API for XML

用于把 Java 解析器连接到因程序上的基于事件的 API,也用于创建解析器和应用程序之间的过滤器。大多数 Java 解析器提供 SAX 支持。

Simple link

在 HREF 属性中包括其目标定位器的链接。

SMIL

见 Synchronized Multimedia Integration Language。

standalone

表示文档是否引用外部资源的声明。

Standard Generalized Markup Language(SGML)

HTML 和 XML 的祖先。SGML 为定义文档结构提供了一组复杂的规则。HTML 使用定义在规则下的结构,而 XML 却为定义文档结构提供了一组子规则。SGML 的标准在 ISO /IEC 8879-1986 中解释,但在后来作过一些改进。

Start-tag

开始元素的打开标签。开始标签的一般语法是 `<Name attributes>`, 其中的 Name 是定义元素的名字, attributes 是一组名字值。XML 中所有的开始标签必须具有结束标签或使用空元素语法 `<Name attributes />`。

Structured Vector Graphics

使用 XML 词汇,描述易于规范和处理的矢量图形开发用的 W3C 标准。

Style sheet

关于文档的格式化描述。样式表可以保存在独立于它们描述的文档中各个单独的文件中。

SVG

见 Structured Vector Graphics。

Synchronized Multimedia Integration Language

W3C 提供 XML 词汇的推荐使用方式,其提供有关资源的引用,并把它们组织成基于空间和时间布局二者的表示形式。

Tag

用于描述元素开始和结束的标记部分。在 `<A>< /A>` 中, `<A>` 是元素 A 的开始标签, `< /A>` 是元素 A 的结束标签, `` 是表示元素 B 的空标签。

Tree-based model

处理 XML 文档的模型,在该模型中,构成了表示文档的对象树,然后又进行了处理。一旦树被建立后,那么在任何时候都可以使用整个文档内容了。

UCS-2

国际字符编码标准。unicode 码支持 2 个字节宽的字符,而不是当前大多数系统支持的一个字节的字符,即可包括 65,536 个字符,而不是 1 字节系统可用的 256 个

字符。更详细的信息可访问 <http://www.unicode.org>。

Uniform Resource Locator(URL)

资源识别器,它可以包含统一资源定位器(Uniform Resource Locator)或统一资源编号(Uniform Resource Number)。

Uniform Resource Number

应用程序可以解决引用资源的编号。由于没有分类系统,因此使用 URN 要比使用 URL 少得多。

Unparsed entity

有关外部资源(典型的是二进制文件)的实体。非分析实体被传送给应用程序,然后应用程序就根据情况处理该实体。

URI

见 Uniform Resource Identifier。

URL

见 Uniform Resource Locator。

URN

见 Uniform Resource Number。

UTF-8

提供访问 Unicode 字符集的编码,但它使用了一个在单字节中值小于 128 的字符表示形式的算法(这在英文中很有用),而对值超过 128 的字符则需要二或三个字节。

Valid

如果文档符合所声明它的文档类型定义并且满足条件,那么该文档就是合法的。所有的元素、属性和实体都必须在 DTD 中声明,并且所有的数据类型与它们的定义要求所匹配。

Validity constraint

需要合法的解析器强制执行的规则。非法表示文档与规则组在文档类型定义方面不匹配。

VBScript

由微软基于她的 VB 语言所创建的脚本语言。VBScript 用在 Internet Explorer 浏览器、Active Server Pages 和某些微软应用程序中。

Voyager

W3C HTML 研究机构计划把 HTML 描述成一组 XML 模型。

W3C

World Wide Web Consortium(万维网联合会),负责许多万维网主要标准的标准实体,主要标准包括 HTML、XML、HTTP 和 Cascading Style Sheets(级联样式表)。W3C 站点包括它们的最新标准,以及其他有关 Web 和标准处理的信息。更详细的信息可访问 <http://www.w3.org>。

Well-formed

组织良好的(Well-formed)文档可以也可以没有 DTD。组织良好的文档必须用 XML 声明开始,并且包含正确嵌入和标记的元素。

Well-formedness constraint 所有 XML 解析器都需要强制执行的规则。如果文档与组织良好的约束不匹配,那么解析器就会停止执行,并向应用程序报告发生了一个致命的错误。

Xlink

在 XML 文档中建立链接所用的 W3C 规则集。编写本书时该规则集仍不是正式文稿。XML 声明

XML 最前面的处理指令。它用 `<? xml` 开始,包括版本识别器、所要求的标记声明和编码识别器,用 `? >` 结束(XML 声明是大小写敏感的,虽然标准与 XML 有关,但是 XML 声明用 `<? xml` 开始)。

XML

见 Extensible Markup Language。

XML-RPC

在网络计算机中进行远程过程调用(Remote Procedure Calls,缩写为 RPC)的 XML 标准。

XP

见 Extensible Protocol。

Xpointer

引用文档的一部分。Xpointer 使用从文本编码发起人(Text Encoding Initiative)衍生来的语法,并修改成编码 URL 的 HTTP 协议所需要的帐号。它也可用于描述创建 Xpointer 而用 W3C 规则集。

XSL

见 Extensible Style Language。

W3C

World Wide Web Consortium(万维网联合会),负责许多万维网主要标准的标准实体,主要标准包括 HTML、XML、HTTP 和 Cascading Style Sheets(级联样式表)。W3C 站点包括它们的最新标准,以及其他有关 Web 和标准处理的信息。更详细的信息可访问 <http://www.w3.org>。

Well-formed

组织良好的(Well-formed)文档可以也可以没有 DTD。组织良好的文档必须用 XML 声明开始,并且包含正确嵌入和标记的元素。

Well-formedness constraint 所有 XML 解析器都需要强制执行的规则。如果文档与组织良好的约束不匹配,那么解析器就会停止执行,并向应用程序报告发生了一个致命的错误。

Xlink

在 XML 文档中建立链接所用的 W3C 规则集。编写本书时该规则集仍不是正式文稿。XML 声明

XML 最前面的处理指令。它用 `<? xml` 开始,包括版本识别器、所要求的标记声明和编码识别器,用 `? >` 结束(XML 声明是大小写敏感的,虽然标准与 XML 有关,但是 XML 声明用 `<? xml` 开始)。

XML

见 Extensible Markup Language。

XML-RPC

在网络计算机中进行远程过程调用(Remote Procedure Calls,缩写为 RPC)的 XML 标准。

XP

见 Extensible Protocol。

Xpointer

引用文档的一部分。Xpointer 使用从文本编码发起人(Text Encoding Initiative)衍生来的语法,并修改成编码 URL 的 HTTP 协议所需要的帐号。它也可用于描述创建 Xpointer 而用 W3C 规则集。

XSL

见 Extensible Style Language。

W3C

World Wide Web Consortium(万维网联合会),负责许多万维网主要标准的标准实体,主要标准包括 HTML、XML、HTTP 和 Cascading Style Sheets(级联样式表)。W3C 站点包括它们的最新标准,以及其他有关 Web 和标准处理的信息。更详细的信息可访问 <http://www.w3.org>。

Well-formed

组织良好的(Well-formed)文档可以也可以没有 DTD。组织良好的文档必须用 XML 声明开始,并且包含正确嵌入和标记的元素。

Well-formedness constraint 所有 XML 解析器都需要强制执行的规则。如果文档与组织良好的约束不匹配,那么解析器就会停止执行,并向应用程序报告发生了一个致命的错误。

Xlink

在 XML 文档中建立链接所用的 W3C 规则集。编写本书时该规则集仍不是正式文稿。XML 声明

XML 最前面的处理指令。它用 `<? xml` 开始,包括版本识别器、所要求的标记声明和编码识别器,用 `? >` 结束(XML 声明是大小写敏感的,虽然标准与 XML 有关,但是 XML 声明用 `<? xml` 开始)。

XML

见 Extensible Markup Language。

XML-RPC

在网络计算机中进行远程过程调用(Remote Procedure Calls,缩写为 RPC)的 XML 标准。

XP

见 Extensible Protocol。

Xpointer

引用文档的一部分。Xpointer 使用从文本编码发起人(Text Encoding Initiative)衍生来的语法,并修改成编码 URL 的 HTTP 协议所需要的帐号。它也可用于描述创建 Xpointer 而用 W3C 规则集。

XSL

见 Extensible Style Language。

W3C

World Wide Web Consortium(万维网联合会),负责许多万维网主要标准的标准实体,主要标准包括 HTML、XML、HTTP 和 Cascading Style Sheets(级联样式表)。W3C 站点包括它们的最新标准,以及其他有关 Web 和标准处理的信息。更详细的信息可访问 <http://www.w3.org>。

Well-formed

组织良好的(Well-formed)文档可以也可以没有 DTD。组织良好的文档必须用 XML 声明开始,并且包含正确嵌入和标记的元素。

Well-formedness constraint 所有 XML 解析器都需要强制执行的规则。如果文档与组织良好的约束不匹配,那么解析器就会停止执行,并向应用程序报告发生了一个致命的错误。

Xlink

在 XML 文档中建立链接所用的 W3C 规则集。编写本书时该规则集仍不是正式文稿。XML 声明

XML 最前面的处理指令。它用 `<? xml` 开始,包括版本识别器、所要求的标记声明和编码识别器,用 `? >` 结束(XML 声明是大小写敏感的,虽然标准与 XML 有关,但是 XML 声明用 `<? xml` 开始)。

XML

见 Extensible Markup Language。

XML-RPC

在网络计算机中进行远程过程调用(Remote Procedure Calls,缩写为 RPC)的 XML 标准。

XP

见 Extensible Protocol。

Xpointer

引用文档的一部分。Xpointer 使用从文本编码发起人(Text Encoding Initiative)衍生来的语法,并修改成编码 URL 的 HTTP 协议所需要的帐号。它也可用于描述创建 Xpointer 而用 W3C 规则集。

XSL

见 Extensible Style Language。