

# 《XML 实用大全》

第一部分 XML 简介.....	9
第 1 章 XML 概览.....	9
1.1 什么是 XML.....	10
1.1.1 XML 是元标记语言.....	10
1.1.2 XML 描述的是结构和语义，而不是格式化.....	10
1.2 为什么开发人员对 XML 感到激动.....	12
1.2.1 设计与特定领域有关的标记语言.....	12
1.2.2 自描述数据.....	12
1.2.3 应用间交换数据.....	13
1.2.4 结构化和集成的数据.....	13
1.3 XML 文档的“生命”.....	14
1.3.1 编辑器.....	14
1.3.2 语法分析程序和处理程序.....	14
1.3.3 浏览器和其他工具.....	14
1.3.4 处理过程总结.....	14
.4 相关技术.....	16
1.4.1 超文本标记语言（Hypertext Markup Language）.....	16
1.4.2 级联样式单（Cascading Style Sheets）.....	16
1.4.3 可扩展的样式语言（Extensible Style Language）.....	16
1.4.4 URL 和 URI.....	17
1.4.5 XLink 和 XPointer.....	17
1.4.6 Unicode 字符集.....	18
1.4.7 如何将这些技术融合在一起.....	18
1.5 本章小结.....	19
第 2 章 XML 应用简介.....	20
2.1 什么是 XML 应用程序.....	20
2.1.1 化学标记语言（Chemical Markup Language）.....	20
2.1.2 数学标记语言（Mathematical Markup Language）.....	21
2.1.3 频道定义格式.....	24
2.1.4 经典文学.....	25
2.2 用于 XML 的 XML.....	27
2.2.1 XSL.....	27
2.2.2 XLL.....	27
2.2.3 DCD.....	28
2.3 XML 的后台应用.....	29
2.4 本章小结.....	32
第 3 章 第一个 XML 文档.....	33
3.1 Hello XML.....	33
3.1.1 创建一个简单的 XML 文档.....	33
3.1.2 保存 XML 文件.....	34
3.1.3 将 XML 文件装入 Web 浏览器.....	34
.2 考察简单的 XML 文档.....	36
3.3 赋予 XML 标记以意义.....	38
.4 为 XML 文档编写样式单.....	39
.5 将样式单附加到 XML 文档上.....	40

3.6 本章小结.....	42
第 4 章 数据的结构化.....	43
4.1 检查数据.....	43
4.1.1 击球手.....	43
4.1.2 投球手.....	45
4.1.3 XML 数据的组织.....	45
4.2 数据的 XML 化.....	47
4.2.1 开始编写文档：XML 声明和根元素.....	47
4.2.2 联赛（League）、（分部）Division 和（球队）Team 数据的 XML 化.....	48
4.2.3 球员数据的 XML 化.....	51
4.2.4 球员统计数据的 XML 化.....	52
4.2.5 将 XML 组装在一起.....	55
4.3 XML 格式的优点.....	70
4.4 编制样式单以便显示文档.....	71
4.4.1 与样式单连接.....	72
4.4.2 为根元素指定样式规则.....	73
4.4.3 为标题指定样式规则.....	74
4.4.4 为球员和统计元素指定样式规则.....	78
4.4.5 本节小结.....	79
4.5 本章小结.....	84
第 5 章 属性、空标记和 XSL.....	85
5.1 属性.....	85
5.2 属性与元素的对比.....	94
5.2.1 结构化的元数据.....	94
5.2.2 元元数据.....	98
5.2.3 有关元数据的说明.....	99
5.2.4 元素更具扩展性.....	99
5.2.5 使用属性的最佳时机.....	99
5.3 空标记.....	101
5.4 XSL.....	102
5.4.1 XSL 样式单模板.....	102
5.4.2 文档的主体.....	103
5.4.3 标题.....	105
5.4.4 联赛、分部和球队.....	107
5.4.5 球员.....	113
5.4.6 区分投手与击球手.....	116
5.4.7 元素内容与 select 属性.....	124
5.4.8 CSS 还是 XSL.....	128
5.5 本章小结.....	129
第 6 章 结构完整的 XML 文档.....	130
6.1 XML 文档的组成.....	130
6.2 置标和字符数据.....	131
6.2.1 注释.....	131
6.2.2 实体引用.....	133
6.2.3 CDATA.....	134
6.2.4 标记.....	135
6.2.5 属性.....	137
6.3 独立文档中结构完整的 XML.....	139
6.4 结构完整的 HTML.....	144
6.4.1 现实的 Web 页面存在的问题.....	144

6.4.2 HTML 整理工具 .....	152
6.5 本章小结.....	155
第 7 章 外文和非罗马文本.....	156
7.1 Web 上的非罗马文字.....	156
7.2 文字、字符集、字体和字形.....	160
7.2.1 文字的字符集.....	160
7.2.2 字符集的字体.....	160
7.2.3 字符集的输入法.....	160
7.2.4 操作系统和应用软件.....	161
7.3 传统字符集.....	162
7.3.1 ASCII 字符集 .....	162
7.3.2 ISO 字符集 .....	163
7.3.3 MacRoman 字符集.....	165
7.3.4 Windows ANSI 字符集.....	166
7.4 Unicode 字符集.....	168
7.4.1 UTF-8.....	170
7.4.2 通用字符系统.....	170
7.5 如何使用 Unicode 编写 XML .....	172
7.5.1 利用字符引用在 XML 文件中插入字符 .....	172
7.5.2 其他字符集与 Unicode 字符集之间的转换 .....	172
7.5.3 如何使用其他字符集编写 XML.....	173
7.6 本章小结.....	175
第二部分 文档类型定义.....	176
第 8 章 文档类型定义和合法性.....	176
8.1 文档类型定义.....	176
8.2 文档类型声明.....	178
8.3 根据 DTD 的合法性检验 .....	181
8.4 列出元素.....	186
8.5 元素声明.....	194
8.5.1 ANY .....	194
8.5.2 #PCDATA .....	194
8.5.3 子元素列表.....	197
8.5.4 序列.....	199
8.5.5 一个或多个子元素.....	199
8.5.6 零或多个子元素.....	200
8.5.7 零或一个子元素.....	200
8.5.8 完整的文档和 DTD .....	202
8.5.9 选择.....	213
8.5.10 带括号的子元素.....	214
8.5.11 混合内容.....	216
8.5.12 空元素.....	217
8.6 DTD 中的注释 .....	219
8.7 在文档间共享通用的 DTD .....	227
8.7.1 远程 URL 上的 DTD.....	234
8.7.2 公共的 DTD .....	235
8.7.3 内部和外部 DTD 子集.....	236
8.8 本章小结.....	240
第 9 章 实体和外部 DTD 子集 .....	241
9.1 什么是实体? .....	241
9.2 内部通用实体.....	243

9.2.1 定义内部通用实体引用.....	243
9.2.2 在 DTD 中使用通用实体引用.....	245
9.2.3 预定义通用实体引用.....	246
9.3 外部通用实体.....	247
9.4 内部参数实体.....	250
9.5 外部参数实体.....	252
9.6 根据片段创建文档.....	259
9.7 结构完整的文档中的实体和 DTD.....	273
9.7.1 内部实体.....	273
9.7.2 外部实体.....	275
9.8 本章小结.....	282
第 10 章 DTDs 中的属性声明.....	283
10.1 什么是属性? .....	283
10.2 在 DTD 中声明属性 .....	284
10.3 声明多个属性.....	286
10.4 指定属性的缺省值.....	287
10.4.1 #REQUIRED .....	287
10.4.2 #IMPLIED .....	287
10.4.3 #FIXED.....	288
10.5 属性类型.....	289
10.5.1 CDATA 属性类型.....	289
10.5.2 Enumerated 属性类型 .....	289
10.5.3 NMTOKEN 属性类型.....	290
10.5.4 NMTOKENS 属性类型.....	290
10.5.5 ID 属性类型 .....	291
10.5.6 IDREF 属性类型 .....	291
10.5.7 ENTITY 属性类型 .....	292
10.5.8 ENTITIES 属性类型 .....	293
10.5.9 NOTATION 属性类型.....	293
10.6 预定义属性.....	295
10.6.1 xml: space.....	295
10.6.2 xml: lang.....	296
10.7 基于属性的棒球统计数据的 DTD.....	299
10.7.1 在 DTD 中声明 SEASON 的属性.....	301
10.7.2 在 DTD 中声明 DIVISION 和 LEAGUE 属性.....	301
10.7.3 在 DTD 中声明 TEAM 属性 .....	301
10.7.4 在 DTD 中声明 PLAYER 的属性 .....	302
10.7.5 棒球比赛统计数据示例的完整 DTD.....	305
10.8 本章小结.....	308
第 11 章 嵌入非 XML 数据.....	309
11.1 记号.....	309
11.2 不可析外部实体.....	313
11.2.1 声明不可析实体.....	313
11.2.2 嵌入不可析实体.....	313
11.2.3 嵌入多个不可析实体.....	316
11.3 处理指令.....	318
11.4 DTD 的条件部分.....	321
11.5 本章小结.....	323
第三部分 样式语言.....	324
第 12 章 级联样式单级别 1.....	324

12.1 什么是 CSS?	324
12.2 样式单与文档的链接	326
12.3 选择元素	330
12.3.1 成组选择符	330
12.3.2 伪元素	330
12.3.3 伪类(pseudo-classe)	331
12.3.4 由 ID 来选择	334
12.3.5 上下文的选择符	334
12.3.6 STYLE 特性	335
12.4 继承性	336
12.5 级联过程	338
12.5.1 @import 指令	338
12.5.2 !important 声明	338
12.5.3 级联顺序	338
12.6 在 CSS 样式单中添加注释	340
12.7 CSS 中的单位	341
12.7.1 长度值	341
12.7.2 URL 值	343
12.7.3 颜色值	344
12.7.4 关键字值	345
12.8 块、内联或列表项元素	346
12.8.1 列表项	351
12.8.2 whitespace 属性	353
12.9 字体属性	356
12.9.1 font-family 属性	356
12.9.2 font-style 属性	357
12.9.3 font-variant 属性	358
12.9.4 font-weight 属性	358
12.9.5 font-size 属性	359
12.9.6 font 简略属性	361
12.10 颜色属性	363
12.11 背景属性	364
12.11.1 background-color 属性	364
12.11.2 background-image 属性	364
12.11.3 background-repeat 属性	366
12.11.4 background-attachment 属性	367
12.11.5 background-position 属性	368
12.12 文本属性	372
12.12.1 word-spacing 属性	372
12.12.2 letter-spacing 属性	373
12.12.3 text-decoration 属性	373
12.12.4 vertical-align 属性	374
12.12.5 text-transform 属性	375
12.12.6 text-align 属性	376
12.12.7 text-indent 属性	377
12.12.8 line-height 属性	377
12.13 框属性	379
12.13.1 页边距属性	379
12.13.2 边框线属性	380
12.13.3 贴边属性	383

12.13.4 大小属性.....	384
12.13.5 定位属性.....	385
12.13.6 float 属性 .....	385
12.13.7 clear 属性.....	386
12.14 本章小结.....	388
第 13 章 级联样式单级别 2.....	388
13.1 CSS2 中有哪些新特点? .....	388
13.1.1 新的伪类.....	389
13.1.2 新的伪元素.....	389
13.1.3 媒体类型.....	389
13.1.4 分页媒体.....	389
13.1.5 国际化.....	389
13.1.6 可视格式化控制.....	390
13.1.7 表格.....	390
13.1.8 生成的内容.....	390
13.1.9 有声样式单.....	390
13.1.10 新工具.....	390
13.2 选择元素.....	392
13.2.1 式样匹配.....	392
13.2.2 通配符.....	393
13.2.3 后代和子代选择符.....	393
13.2.4 直系同属选择符.....	395
13.2.5 特性选择符.....	395
13.2.6 @规则.....	395
13.2.7 伪元素.....	399
13.2.8 伪类.....	400
13.3 格式化页面.....	402
13.3.1 大小属性.....	402
13.3.2 页边距属性.....	402
13.3.3 标记属性.....	402
13.3.4 页面属性.....	402
13.3.5 分页符属性.....	403
13.4 可视格式化.....	404
13.4.1 显示属性.....	404
13.4.2 宽度和高度属性.....	406
13.4.3 overflow 属性 .....	406
13.4.4 clip 属性.....	407
13.4.5 visibility 属性 .....	407
13.4.6 cursor 属性.....	408
13.4.7 相关的颜色属性.....	409
13.5 框.....	411
13.5.1 轮廓属性.....	411
13.5.2 定位属性.....	412
13.6 计数器和自动编号.....	416
13.7 有声样式单.....	418
13.7.1 说话属性.....	419
13.7.2 音量属性.....	419
13.7.3 暂停属性.....	419
13.7.4 提示属性.....	420
13.7.5 同期播放属性.....	420

13.7.6 空间属性.....	420
13.7.7 音质属性.....	421
13.7.8 话音属性.....	423
13.8 本章小结.....	424
第 14 章 XSL 变换.....	425
14.1 何为 XSL? .....	425
14.2 XSL 变换概述.....	427
14.2.1 树形结构.....	427
14.2.2 XSL 样式单文档.....	429
14.2.3 在何处进行 XML 变换.....	430
14.2.4 如何使用 XT .....	431
14.2.5 直接显示带有 XSL 样式单的 XML 文件.....	433
14.3 XSL 模板.....	435
14.3.1 xsl:apply-templates 元素.....	435
14.3.2 select 特性.....	437
14.4 使用 xsl:value-of 来计算节点值.....	439
14.5 使用 xsl:for-each 处理多个元素.....	441
14.6 匹配节点的模式.....	443
14.6.1 匹配根节点.....	443
14.6.2 匹配元素名.....	444
14.6.3 使用/字符匹配子节点.....	446
14.6.4 使用//符号匹配子代.....	447
14.6.5 通过 ID 匹配.....	448
14.6.6 使用@来匹配特性.....	448
14.6.7 使用 comments()来匹配注释.....	450
14.6.8 使用 pi()来匹配处理指令.....	451
14.6.9 用 text()来匹配文本节点.....	451
14.6.10 使用“或”操作符 .....	452
14.7 选择节点的表达式.....	453
14.7.1 节点轴.....	453
14.7.2 表达式类型.....	461
14.8 缺省的模板规则.....	470
14.8.1 元素的缺省规则.....	470
14.8.2 文本节点的缺省规则.....	470
14.8.3 两个缺省规则的含义.....	470
14.9 决定输出要包含的内容.....	472
14.9.1 使用特性值模板.....	472
14.9.2 使用 xsl:element 将元素插入到输出文档中.....	474
14.9.3 使用 xsl:attribute 将特性插入到输出文档中.....	475
14.9.4 定义特性集合.....	476
14.9.5 使用 xsl:pi 生成处理指令.....	477
14.9.6 使用 xsl:comment 生成注释.....	477
14.9.7 使用 xsl:text 生成文本.....	478
14.10 使用 xsl:copy 复制当前节点.....	479
14.11 使用 xsl:number 为节点计数.....	482
14.11.1 缺省数值.....	483
14.11.2 数字到字符串的变换.....	485
14.12 对输出元素排序.....	487
14.13 CDATA 和<符.....	490
14.14 方式.....	492

14.15 使用 xsl:variable 定义常数.....	495
14.16 命名模板.....	495
14.16.1 参数.....	497
14.17 删除和保留空白.....	500
14.18 选择.....	502
14.18.1 xsl:if .....	502
14.18.2 xsl:choose.....	502
14.19 合并多个样式单.....	504
14.19.1 使用 xsl:import 进行录入 .....	504
14.19.2 使用 xsl:include 进行包括 .....	504
14.19.3 使用 xsl:stylesheet 在文档中嵌入样式单 .....	504
14.20 本章小结.....	507
第 15 章 XSL 格式化对象 .....	507
15.1 XSL 格式化语言概述 .....	508
15.2 格式对象及其属性.....	509
15.2.1 fo 命名域 .....	510
15.2.2 格式化属性.....	512
15.2.3 转换成格式化对象.....	517
15.2.4 使用 FOP .....	519
15.3 页面布局.....	521
15.3.1 主控页面.....	521
15.3.2 页序列.....	524
15.4 内容.....	529
15.4.1 块级格式化对象.....	529
15.4.2 内联格式化对象.....	530
15.4.3 表格格式化对象.....	531
15.4.4 外联格式化对象.....	531
15.5 水平线.....	532
15.6 图形.....	533
15.7 链接.....	534
15.8 列表.....	535
15.9 表格.....	537
15.10 字符.....	541
15.11 序列.....	542
15.12 脚注.....	543
15.13 浮动.....	543
15.14 XSL 格式化属性 .....	545
15.14.1 单位和数据类型.....	545
15.14.2 消息属性.....	547
15.14.3 段落属性.....	547
15.14.4 字符属性.....	550
15.14.5 句子属性.....	551
15.14.6 区域属性.....	554
15.14.7 听觉属性.....	559
15.15 本章小结.....	561
第四部分 补充技术.....	562
第 16 章 XLink .....	562
16.1 XLink 与 HTML 链接的对比.....	562
16.2 简单链接.....	564
16.2.1 本地资源的描述.....	565



16.2.2 远程资源的描述.....	566
16.2.3 链接行为.....	567
16.3 扩展链接.....	574
16.4 外联链接.....	578
16.5 扩展链接组.....	580
16.5.1 一个实例.....	580
16.5.2 steps 特性.....	582
16.6 重命名 XLink 特性 .....	584
16.7 本章小结.....	585
第 18 章 命名域.....	587
18.1 何为命名域.....	587
18.2 命名域句法.....	590
18.2.1 命名域的定义.....	590
18.2.2 多个命名域.....	591
18.2.3 特性.....	594
18.2.4 缺省的命名域.....	595
18.3 DTD 中的命名域 .....	599
18.4 本章小结.....	600

---

## 第一部分 XML 简介

本部分包括以下各章：

第 1 章——XML 概览

第 2 章——XML 应用简介

第 3 章——第一个 XML 文档

第 4 章——数据的结构化

第 5 章——特性、空标记和 XSL

第 6 章——结构完整的 XML 文档

第 7 章——外国语言与非罗马文字

### 第 1 章 XML 概览

本章将向读者介绍 XML 的基本知识以及概略地解释什么是 XML 以及如何使用 XML。还要向读者说明如何将各种不同的 XML 表达式组合在一起，XML 文档是如何创建的并如何向人们发送这种文档。

本章的主要内容包括：

- 什么是 XML
- 为什么开发人员对 XML 感到激动
- XML 文档的“生命”
- 相关的技术

## 1.1 什么是 XML

XML 代表 Extensible Markup Language (eXtensible Markup Language 的缩写, 意为可扩展的标记语言)。XML 是一套定义语义标记的规则, 这些标记将文档分成许多部件并对这些部件加以标识。它也是元标记语言, 即定义了用于定义其他与特定领域有关的、语义的、结构化的标记语言的句法语言。

### 1.1.1 XML 是元标记语言

关于 XML 要理解的第一件事是, 它不只是像超文本标记语言 (Hypertext Markup Language, HTML) 或是格式化的程序。这些语言定义了一套固定的标记, 用来描述一定数目的元素。如果标记语言中没有所需的标记, 用户也就没有办法了。这时只好等待标记语言的下一个版本, 希望在新版本中能够包括所需的标记, 但是这样一来就得依赖于软件开发商的选择了。

但是 XML 是一种元标记语言。用户可以定义自己需要的标记。这些标记必须根据某些通用的原理来创建, 但是在标记的意义上, 也具有相当的灵活性。例如, 假如用户正在处理与家谱有关的事情, 需要描述人的出生、死亡、埋葬地、家庭、结婚、离婚等, 这就必须创建用于每项的标记。新创建的标记可在文档类型定义 (Document Type Definition, 在以后的篇幅中常简称为 DTD) 中加以描述。在本书的第二部分中将会学到有关 DTD 的更多的知识。现在, 只需把 DTD 看作是一本词汇表和某类文档的句法。例如, 在 Peter Murray-Rust 的 Chemical Markup Language (化学标记语言, 简称为 CML) 中的 MOL.DTD 文件中描述了词汇表和分子科学的句法: 其中包括 chemistry (化学)、crystallography (结晶学)、solid state physics (固体物理) 等词汇。它包括用于 atoms (原子)、molecules (分子)、bonds (化学键)、spectra (光谱) 等的标记。这个 DTD 可与分子科学领域中的许多不同的人共享。对于其他领域也有其他的 DTD, 用户还可以创建自己的 DTD。

XML 定义了一套元句法, 与特定领域有关的标记语言 (如 MusicML、MathML 和 CML) 都必须遵守。如果一个应用程序可以理解这一元句法, 那么它也就自动地能够理解所有的由此元语言建立起来的语言。浏览器不必事先了解多种不同的标记语言使用的每个标记。事实是, 浏览器在读入文档或是它的 DTD 时才了解了给定文档使用的标记。关于如何显示这些标记的内容的详细指令是附加在文档上的另外的样式单提供的。例如, 考虑薛定格 (Schrodinger) 方程:

$$i\hbar \frac{\partial \psi(\vec{r}, t)}{\partial t} = -\frac{\hbar^2}{2m} \frac{\partial^2 \psi(\vec{r}, t)}{\partial x^2} + V(r) \psi(\vec{r}, t)$$

科学论文中充满了这一类方程, 但是科学家还必须等待多年, 才能让浏览器的开发商支持书写最基本的数学公式所需的标记。音乐家也有同样的局限性, 因为 Netscape Navigator 和 Internet Explorer 还都不支持乐谱。

有了 XML 就意味着不必等待浏览器的开发商来满足用户的需要了。用户可以创建自己需要的标记, 当需要时, 告诉浏览器如何显示这些标记就可以了。

### 1.1.2 XML 描述的是结构和语义, 而不是格式化

关于 XML 要了解的第二件事是, XML 标记描述的是文档的结构和意义。它不描述页面元素的格式化。可用样式单为文档增加格式化信息。文档本身只说明文档包括什么标记, 而不是说明文档看起来是什么样的。

作为对照, HTML 文档包括了格式化、结构和语义的标记。<B>就是一种格式化标记, 它使其中的内容变为粗体。<STRONG>是一种语义标记, 意味着其中的内容特别重要。<TD>是结构标记, 指明内容是表中的一个单元。事实上, 某些标记可能具有所有这三种意义。<H1>标记可同时表示 20 磅的 Helvetica 字体的粗体、第一级标题和页面标题。

例如, 在 HTML 中, 一首歌可能是用定义标题、定义数据、无序的列表和列表项来描述的。但是事实上这些项目没有一件是与音乐有关的。用 HTML 定义的歌曲可能如下:

<dt>Hot Cop

```
<dd> by Jacques Morali Henri Belolo and Victor Willis
```

```
<ul>
```

```
<li>Producer: Jacques Morali
```

```
<li>Publisher: PolyGram Records
```

```
<li>Length: 6:20
```

```
<li>Written: 978
```

```
<li>Artist: Village People
```

```
</ul>
```

而在 XML 中，同样的数据可能标记为：

```
<SONG>
```

```
<TITLE>Hot Cop</TITLE>
```

```
<COMPOSER>Jacques Morali</COMPOSER>
```

```
<COMPOSER>Henri Belolo</COMPOSER>
```

```
<COMPOSER>Victor Willis</COMPOSER>
```

```
<PRODUCER>Jacques Morali</PRODUCER>
```

```
<PUBLISHER>PolyGram Records</PUBLISHER>
```

```
<LENGTH>6:20</LENGTH>
```

```
<YEAR> 978</YEAR>
```

```
<ARTIST>Village People</ARTIST>
```

```
</SONG>
```

在这个清单中没有使用通用的标记如<dt>和<li>，而是使用了具有意义的标记，如<SONG>、<TITLE>、<COMPOSER>和<YEAR>等。这种用法具有许多优点，包括源码易于被人阅读，使人能够看出作者的含义。

XML 标记还使非人类的自动机器人易于找出文档中的所有歌曲。在 HTML 中，机器人只能告诉我们这个元素是 dt。机器人不能决定 dt 到底代表一首歌的题目还是定义，抑或只是一些设计者喜爱的缩进文本格式。事实上，单一文档中可以很好地包括带有三种意义的各种 dt 元素。

可以选择 XML 的元素名称，以便使其在附加的上下文中具有额外的意义。例如，元素名称可以是数据库的域名。XML 比 HTML 更为灵活而且适用于各种应用，因为有限数目的标记不必用于许多不同的目的。

## 1.2 为什么开发人员对 XML 感到激动

XML 使许多只利用 HTML 难以解决的任务变得简单，使只利用 HTML 不可能完成的任务得以完成。因为 XML 是可扩展的，开发人员喜爱 XML 有许多原因。到底是哪个更令人感兴趣，取决于每个人的需要。但有一点是肯定的，一旦用上 XML，就可发现，它正是解决许多令人感到棘手的问题的有力工具。本节研究一些令开发人员激动的一般应用。在第 2 章中，还会看到已经用 XML 开发出来的一些特殊应用。

### 1.2.1 设计与特定领域有关的标记语言

XML 允许各种不同的专业（如音乐、化学、数学等）开发与自己的特定领域有关的标记语言。这就使得该领域中的人们可以交换笔记、数据和信息，而不用担心接收端的人是否有特定的软件来创建数据。特定领域的开发人员甚至可以向本领域外的人发送文档，有相当的理由可以认为，至少接受文档的人能够查看文档的内容。

更进一步说，为特别的领域创建标记语言不会产生“病件”（bloatware）或是对于本专业外的人来说产生不必要的复杂性。一般人也许不会对电力工程图感兴趣，但是电力工程师却对此感兴趣。一般人也许不需要在他的 Web 页面中包括乐谱，但是作曲家却要这样做。XML 让电力工程师描述他们的电路图，让作曲家写乐谱，而不会互相干扰。对于浏览器开发商来说，都不需要对特定的领域提供特殊的支持，也不需要提供复杂的插件。这一点现在已经实现了。

### 1.2.2 自描述数据

过去 40 年来的大多数计算机数据都丢失了，不是因为自然损害或是备份介质的磨损（虽然这也是一个问题，这个问题在 XML 中也没有解决），而只是因为没有人来写出如何读取这些数据介质和格式的文档。在十年前的 5.25 英寸的软盘上的 Lotus 1-2-3 文档在今天的大多数公司内都已经读不出来了。以不常用的格式保存的二进制数据，如 Lotus Jazz 也许会永远地消失了。XML 在基本水平上使用的是非常简单数据格式。可以用 100% 的纯 ASCII 文本来书写，也可以用几种其他定义好的格式来书写。ASCII 文本是几乎不会“磨损”的。丢失一些字节甚至是相当多的字节，剩下的数据还是可以读取的。这就与许多格式形成了鲜明的对比，如压缩数据或是串行的 Java 对象，这些数据即使丢失一个字节，剩余的数据也变得不可读取了。

从高水平上来说，XML 是自描述的。假设在 23 世纪有一个信息考古学者，他在软盘上发现了如下一大段经过时间的“冲刷”而保存下来的 XML 代码：

```
<PERSON ID="p1100" SEX="M">

<NAME>

<GIVEN>Judson</GIVEN>

<SURNAME> McDaniel</SURNAME>

</NAME>

<BIRTH>

<DATE>2 Feb 1834</DATE> </BIRTH>

<DEATH>

<DATE>9 Dec 1905</DATE> </DEATH>

</PERSON>
```

即使这个考古学家不熟悉 XML，但假设他可以讲 20 世纪时的英语，那么就可以很好地了解名为 Judson McDaniel 的人，此人出生在 1834 年 2 月 21 日，而死于 1905 年 12 月 9 日。事实上，数据中有一些空白或是损坏，还是可以得到这些信息。但对于专有格式的电子表格或是字处理程序的格式，就不是这么回事了。

更进一步说，XML 有很好的规格文档。W3C 的 XML 1.0 规范和大量的论文书籍，如本书，都向人们准确地说明如何来阅读 XML 数据。没有什么秘密使得人们发生失误。

### 1.2.3 应用间交换数据

由于 XML 是非专有的并易于阅读和编写，就使得它成为在不同的应用间交换数据的理想格式。当前正在开发的一种这样的格式是 Open Financial Exchange（开放财务交换，简称为 OFX）格式。OFX 是为个人财务程序，如 Microsoft Money 和 Quicken 交换数据而设计的。数据可以在程序间来回交换，还可以与银行、经纪事务所和其他机构交换数据。



有关 OFX 的内容将在第 2 章加以讨论。

正如上面所讨论的一样，XML 使用的是非专有的格式，不受版权、专利、商业秘密或是其他种类的知识产权的限制。XML 的功能是非常强大的，同时对于人类或是计算机程序来说，都容易阅读和编写。因而成为交换语言的首选。

使用 XML 而不是专有格式，人们就可以利用任何理解 XML 的工具来处理数据。还可以为不同的目的使用不同的工具。一个程序用来查看而另一程序用来编辑。XML 使用户不必因为数据已经用专有格式编写好了或是接受数据的人只接受专有格式而限制在一个特定的程序上。

例如，许多出版商需要用 Microsoft Word 发稿。这就意味着大多数作者必须使用 Word，即使他们更愿意使用 WordPerfect 或是 Nisus Writer。因而这就使得其他出版字处理软件的公司陷入困境，除非他们的软件能够读写 Word 文件。由于要想达到这个目的，就得让开发人员反向了解未载入文档的 Word 文件格式，这使得在时间和资源上的投资大增。大多数其他字处理软件具有有限的读写 Word 文件的能力，但是通常都会丢失图形、宏、样式、修订标记和其他重要的特性。问题就在于 Word 文档的格式是不公开的专有格式，而且还在不断地变化。这样 Word 就成为最后的胜利者，即使作者更喜爱其他的更简单的程序。如果在 XML 中开发了一种通用的字处理格式，作者们就会使这个程序成为他们的首选程序。

### 1.2.4 结构化和集成的数据

XML 对于大型和复杂的文档是理想的，因为数据是结构化的。这不仅使用户可以指定一个定义了文档中的元素的词汇表，而且还可以指定元素之间的关系。例如，如果要销售客户的地址一起放在 Web 页面上，这就需要有每个客户的电话号码和电子邮件地址。如果向数据库中输入数据，可确保没有漏下的字段。还需要每部书都有一个作者。当没有数据输入时还可提供一个缺省值。XML 也提供客户端的包括机制，可以根据多种来源集成数据并将其作为一个文档来显示。数据还可以马上进行重新排列。数据的各个部分可以根据用户的操作显示或隐藏。当处理大型的信息仓库，比如关系型数据库时是极为有用的。

## 1.3 XML 文档的“生命”

从基本上来说，XML 是一种文档格式。它是一系列的关于 XML 文档看起来是什么样子的规则。与 XML 标准的符合程度有两种级别。第一级是结构完整性，第二级是正确性。本书的第一部分向读者介绍如何编写结构完整的文档。而第二部分向读者介绍如何编写具有正确性的文档。

HTML 是设计用于 Internet 上和 Web 页面内部的文档格式。正如本书所叙述的，XML 当然也可以用在这些方面。但是 XML 具有更为广泛的适用性。正如前面所讨论的，可用于字处理器的保存文件的格式，可用于不同程序间的数据交换格式，可用作与 Intranet 模板一致化的工具，还可用作以人类可读的形式保存数据的手段。

虽然如此，如所有的数据格式一样，XML 在有用之前也需要程序和内容。因而对于数据看起来应该是什么样子的，光了解 XML 本身还是不够的，这不光是一个规范所能解决的问题。用户还需要了解 XML 文档是如何编辑的，处理程序是如何读取 XML 文档并将其读取的信息传送给应用程序的，以及这些应用程序是如何处理数据的。

### 1.3.1 编辑器

XML 文档大多数情况下都是用编辑器创建的。编辑器可以是基本的文本编辑器如 Notepad（记事本）或是 vi，这些编辑器并不真正理解 XML。另一方面，也可以用所见即所得的编辑器，如 Adobe FrameMaker，这种编辑器可将用户完全隔离于 XML 底层格式之外。另外也可以是一个结构化的编辑器，如 JUMBO，它可将 XML 文档显示为树状结构。对于最重要的部分，有趣的编辑器并不是太有用，因而本书将注意力集中于用普通的文本编辑器来编写 XML 文档。

其他程序也可以创建 XML 文档。例如，本书在讲述设计新的 DTD 的稍后章节中将可看到某些 XML 数据可直接从 FileMaker 的数据库中得出。在这种情况下，数据是先输入到 FileMaker 数据库中的，然后 FileMaker 的计算字段将数据转换为 XML。一般来说，XML 与数据库可协同工作得很好。



准确地说，我们可在第 23 章“设计新的 XML 应用”中看到这种情况。

无论在何种情况下，都是编辑器或其他程序创建了 XML 文档。通常，这一文档是某种计算机硬盘上的实际文件。但也不是必须如此。例如，文档可能是数据库中的记录或是字段，或者可能是从网络上接收来的字节流。

### 1.3.2 语法分析程序和处理程序

XML 的语法分析程序（即所谓的 XML 处理程序）读取文档并检查其中包括的 XML 是否是结构完整的。它还要确定文档是否合法，虽然这种测试不是必需的。这种测试的详细情况将在本书的第二部分中讲述。如果文档通过了测试，则处理程序就将文档转换为元素的树状结构。

### 1.3.3 浏览器和其他工具

最后语法分析程序将树状结构或是树的节点传送给用户端应用程序。这个应用程序可能是浏览器，如 Mozilla，或是其他能够理解如何处理数据的程序。如果这个应用程序是浏览器的话，数据就显示给用户。但是其他程序也可以接受数据。例如，可将数据翻译成数据库的输入、一系列要演奏的乐谱或是要运行的 Java 程序。XML 是非常灵活的，可以用于许多不同的目的。

### 1.3.4 处理过程总结

总结一下，首先由一个编辑器创建了 XML 文档。语法分析程序将树状结构传送给浏览器，由浏览器显示出来。图 1-1 显示了这个处理过程。



图 1-1 XML 文档的处理流程

请注意，所有这些部分都是独立的，互相分离的。将这些部分联系在一起的是 XML 文档。改变编辑程序与终端应用程序无关。事实上，很可能在编写文档时就根本不知道最终的应用程序是什么。可能是最终用户来阅读文档，也可能是数据库从中提取数据，甚至还可能是未发明出来的程序，也可能是所有这些情况。文档与读取它的程序是无关的。



HTML 也在某种程度上与读写它的程序无关，但是它只适用于浏览器。其他应用，如数据库输入已经不在它的有效范围之内了。例如，HTML 没有提供某种方法来包括所需的内容，如每本书都必须有 ISBN 号码一样。在 XML 中可以包括这个。甚至可以强制安排元素出现的顺序（如第二级标题必须出现在第一级之后）。

## . 4 相关技术

XML 并不是在真空中操作的。如果将 XML 用于不只是一种数据格式的话，就需要与多种相关的技术相互作用。这些技术包括为了向后兼容老式的浏览器的 HTML、CSS (Cascading Style Sheet, 级联样式单) 和 XSL (eXtensible Style Languages, 可扩展的样式语言)、URL 和 URI、XLL (eXtensible Linking Language, 可扩展的链接语言) 和 Unicode 字符集。

### 1.4.1 超文本标记语言 (Hypertext Markup Language)

Mozilla 5.0 和 Internet Explorer 5.0 是首先对 XML 提供支持 (虽然并不完全) 的浏览器。但是, 要使大多数用户升级到这两种浏览器的新版本上来, 可能还要花两年的时间。(我的妻子 Beth 在 1999 年还在使用 Netscape 1.1。)因而在今后一段时间内, 还需要将 XML 内容转化为经典的 HTML。

因而, 在转向 XML 之前, 对使用 HTML 还不应感到别扭。用户不必完全成为一个时髦的图形设计者, 但是应该了解如何将一个页面与另一个页面链接起来, 了解如何在文档中包括图像, 如何使文本变成粗体等等。由于 HTML 是 XML 的最普通的输出格式, 所以对 HTML 了解得越多, 也就越容易了解如何创建所需的效果。

另一方面, 如果已经熟悉了利用表格或是单像素的 GIF 来安排页面上的对象, 或是如果开始借助于画出草图而不是借助于内容来创建 Web 站点的话, 那么也就必须要忘记某些坏的习惯。正如前面所讨论的一样, XML 将文档的内容与文档的外观相分离。首先开发内容, 然后再用样式单将格式附加其上。将内容与样式分开是非常有效的技术, 这既改善了文档内容也改善了文档外观。除此之外, 还允许作者和设计者更加互相独立地工作。但是, 对于设计 Web 站点来说, 确实需要有不同的思路, 如果涉及多人的话, 或许要利用不同的项目管理技术。

### 1.4.2 级联样式单 (Cascading Style Sheets)

由于 XML 允许在文档中包括任意的标记, 所以对于浏览器来说, 没有办法事先知道如何显示每个元素。当将文档送给用户时, 还要向用户发送样式单, 通过样式单告诉浏览器如何格式化每个元素。可以使用的一种样式单是级联样式单 (Cascading Style Sheet, 简称为 CSS)。

CSS 开始是为 HTML 设计的, 它定义字号、字族、字重、段落缩进、段落对齐和其他样式等格式化属性, 这些属性都可以施加到个别的元素上。例如, CSS 允许 HTML 文档来指定所有的 H1 元素应该被格式化为 32 磅、中间对齐的 Helvetica 字体的粗体。单独的样式可以施加到大多数 HTML 标记上, 它能够覆盖浏览器的缺省设置。多个样式单可施加到一个文档上, 而多个样式也可用于单个元素上。样式根据特定的一套规则级联起来。



CSS 规则和属性将在第 12 章“级联样式单, 第一级”和第 13 章“级联样式单, 第二级”中详细介绍。

向 XML 施加 CSS 规则是很容易的。只要改变施加规则于其上的标记名称即可。Mozilla 5.0 直接支持 CSS 样式单与 XML 的结合, 虽然到目前为止, 此浏览器时常发生崩溃。

### 1.4.3 可扩展的样式语言 (Extensible Style Language)

可扩展的样式语言 (Extensible Style Language, 简称为 XSL) 是更为先进的专门用于 XML 文档的样式单语言。XSL 文档本身就是结构完整的 XML 文档。

XSL 文档包括一系列的适用于特定的 XML 元素样式的规则。XSL 处理程序读取 XML 文档并将其读入的内容与样式单中的模式相比较。当在 XML 文档中识别出 XSL 样式单中的模式时, 对应的规则输出某些文本的组合。与级联样式单不同, 输出的文本比较任意, 也不局限于输入文本加上格式化信息。



CSS 只能改变特定元素的格式，也只能以元素为基础。但 XSL 样式单可以重新排列元素并对元素进行重排序。这种样式单可以隐藏一些元素而显示另外一些元素。更进一步说，还可以选择应用样式的标记，而不仅是基于标记的，而且还基于标记的内容和特性，还基于标记在文档中相对于其他元素的位置，以及基于各种其他的准则。



CSS 的优越性在于具有广泛的浏览器支持。但是 XSL 更为灵活和强大，可更好地适用于 XML 文档。而且带 XSL 样式单的 XML 文档可以很容易地转换为带 CSS 样式单的 HTML 文档。

XSL 样式单将第 14 章“XSL 变换”和第 15 章“XSL 格式化对象”中更为详细地论述。

#### 1.4.4 URL 和 URI

XML 文档可用于 Web，正如 HTML 和其他文档一样。使用时，也如 HTML 文档一样，被统一资源定位符（Uniform Resource Locator，简称为 URL）所引用。例如，在 URL <http://www.hypermedic.com/style/xml/tempest.xml> 处，可以找到以 XML 标记的莎士比亚的歌剧 *tempest* 的全文。虽然 URL 已被人们广泛理解并被广泛支持，但 XML 规范使用的是更为通用的统一资源标识符（Uniform Resource Identifier，简称为 URI）。URI 对于定位 Internet 上的资源是更为通用的架构，更为注重资源而不太注重位置。理论上说，URI 可找出镜像文档的最为近似的副本或是找出已经从一个站点移动到另一站点的文档。实际上，URI 仍然处于进一步的研究之中，被当前的软件所唯一支持的一种 URI 正是 URL。

#### 1.4.5 XLink 和 XPointer

只要将 XML 张贴到 Internet 上，用户当然希望能够对此文档寻址并且可以将这些文档链接起来。标准的 HTML 链接标记可用在 XML 文档中，而且 HTML 文档也可与 XML 文档加以链接。例如，下面的 HTML 代码将链接指向了前文提到的以 XML 形式出现的 *Tempest* 的副本：

```
<a href="http://www.hypermedic.com/style/xml/tempest.xml">
```

The *Tempest* by Shakespeare

```
</a>
```



如果用户跟随着链接，浏览器能否显示这个文档，依赖于该浏览器处理 XML 文件的能力。目前大多数浏览器还不能很好地处理 XML 文档。

然而，XML 利用 XLink 来与文档链接，用 XPointer 来确定文档个别部分的位置，就可以有更多的功能。.



XLink 使任意元素成为链接，而不只是 A 元素。进一步说，链接可以是双向的、多向的或是指向多个镜像的站点，并选择这些站点中最近的一个。XLink 利用普通的 URL 来标识它链接的站点。.

XLink 将在第 16 章中加以讨论。

XPointer 能使链接不仅指向特定位置处的特定文档，而且还可指向特定文档的特定部分。XPointer 可以引用文档中的特定的元素，如第一个、第二个或是第十七个特定的元素。XPointer 提供了文档间连接的非常强大的功能，而这些文档不必有包括附加标记的目的文档，正因为如此，其中的个别部分才可以被链接。



进一步说，与 HTML 的锚（anchor）不同，XPointer 不只是引用文档中的一点。XPointer 可以指向一个范围或是一个区域。因而 XPointer 可以用来选择文档的特定部分，或许这样一来，就可以将这部分复制或或是将其装入其他程序。

XPointer 将在第 17 章中加以讨论。

#### 1.4.6 Unicode 字符集

Web 是国际性的，到目前为止其上主要文本部分仍为英文。XML 是改变这种状况的开始。XML 对双字节的 Unicode 字符集及其紧凑的表示提供了完全的支持。这一字符集几乎可以支持地球上的每一种常用的字符。遗憾的是，光有 XML 还是不够的。为了阅读一种文字，需要三个条件：

1. 该种文字的字符集
2. 该字符集的字体
3. 操作系统和应用软件能够理解这种字符集



如果想要以这种文字写作，并阅读这种文字，还需要该种文字的输入法。当然，XML 定义了字符引用，可使用户使用纯 ASCII 字符将未列在本地字符集中的字符加以编码。这对于偶尔引用一下希腊或是中文字符也足够了，当然不能指望用这种办法以其他语言来写一部小说。

在第 7 章“外国语言和非罗马文本”中，读者将会看到国际文本在计算机中是如何来代表的，XML 如何来理解文本，以及如何来利用不得不以非英语来读写的软件。

#### 1.4.7 如何将这些技术融合在一起

XML 定义了一些标记的语法规则，可用来标记文档。XML 文档是用 XML 标记来标记的。XML 文档的缺省编码方法是 Unicode。

XML 文档的许多好处之一是，可以包括与其他文档和资源的超链接。这些链接是根据 XLink 规范创建的。XLink 用 URI（理论上）或是用 URL（实际上）标识出链接的文档。一个 XLink 可进一步指定它所链接文档的个别部分。这些个别部分是通过 XPointer 来寻址的。如果打算由人来阅读 XML 文档，那么样式单就提供个别元素格式化的指令（并不是所有的 XML 文档都如此）。样式单可用几种样式语言中的任一种来编写。CSS 和 XSL 是两种最常用的样式语言，虽然也存在其他基于 XSL 的样式语言，如 DSSSL（Document Style Semantics and Specification Language，文档样式语义和规格语言）。



我已经在本章中概述了许多令人激动的技术。但是，良知让我告诉读者，我还没有全讨论到。事实上，我所叙述的大部分是 XML 的前景而不是当前的现实。XML 让软件产业中的许多人激动不已，许多程序员正在奋发工作，以便将梦想变为现实。层出不穷的新软件正将我们带入 XML 的“天堂”，但是由于这一领域非常新，许多新软件还没有经过充分地考验。在本书的其余部分，我将小心地不仅要指出什么将可能出现，而且也指出什么实际已经上出现了。令人沮丧的是，这两件事常常不是一回事。不管怎么说，当前还是可以小心地用 XML 来做一些实际工作的。

## 1.5 本章小结

在本章中，读者了解了某些 XML 可以为我们做的事情。更明确地说，了解了以下几个方面：

- 一种能够为特定文档和领域创建标记语言的元语言。
- XML 标记描述了文档内容的结构和语义，而不是内容的格式。格式是在另外的样式单中描述的。
- XML 的起因是，用户受到 SGML 复杂性的挫伤和 HTML 的不充分。
- XML 是用编辑器创建的，由语法分析程序来读取，而由浏览器来显示的。
- 在 Web 上的 XML 是建立在由 HTML、级联样式单和 URL 提供的基础之上的。
- 许多支持技术处于 XML 之上，包括 XSL 样式单、XLink 和 XPointer。这些技术使用户可以比只使用 CSS 和 URL 完成更多的任务。
- 一定要小心。XML 并未彻底完成。它随时会发生变化或是扩展，而在当前的 XML 软件中可能会遇到这样或那样的错误。

在以下几章中，读者可以看到几个 XML 应用，学到某些将 XML 用到现实中的方式。例子包括音乐乐谱、数学、化学、人力资源、Web 广播以及其他一些应用。

## 第 2 章 XML 应用简介

在本章中，我们将要查看 XML 的几个应用实例、用来进一步改进 XML 的标记语言和在后台使用的 XML。看一看 XML 的某些应用，即使只是发展的初级阶段，也是令人鼓舞的。本章将向读者讲述 XML 的广泛应用性的某些看法。在我写作本书时，更多的 XML 应用正在创建并与其他格式的应用接轨。

第五部分更为详细地讲述了本章中讨论过的一些 XML 应用程序。

本章的主要内容包括：

- 什么是 XML 应用程序
- 用于 XML 的 XML
- XML 的后台应用

### 2.1 什么是 XML 应用程序

XML 是一种元标记语言，可用来设计与特定专业领域有关的标记语言。每种基于 XML 的标记语言都叫做 XML 应用程序。这种应用不是像 Mozilla Web 浏览器、Gnumeric 电子表格或 XML Pro 那样的编辑器一样地使用 XML，而是在特定的领域中应用 XML，如化学上用的化学标记语言（Chemical Markup Language，简称为 CML）或是家谱上用的 GedML。每种 XML 应用程序有它自己的句法和词汇表。这种句法和词汇表遵守 XML 的基本规则。

这有点像人类语言，每种语言都有它们自己的词汇表和语法，但同时遵循人体解剖学和大脑结构所要求的基本规则。

XML 是以文本数据为基础的非常灵活的格式。在本章中讨论的广泛的应用都选择了 XML 作为基础的原因是（排除大肆宣传的因素），XML 提供了切合实际的并清楚地描述了易于读写的格式。应用程序将这种格式用于它的数据，就能够将大量的处理细节让几个标准工具和库函数去解决。更进一步说，对于这样的程序也容易将附加的句法和语义加到 XML 提供的基本结构之上。

#### 2.1.1 化学标记语言（Chemical Markup Language）

Peter Murray-Rust 的化学标记语言（Chemical Markup Language，简称为 CML）可能是第一个 XML 应用。CML 原来是要发展成 SGML 应用的，但随着 XML 标准的发展，逐步演化成了 XML。在 CML 的最简单的形式下，CML 是“HTML 加分子”，但是它的用处却超出了 Web 的范围。

分子文档常常包括成千上万个不同的详细的对象。例如，单个中等大小的有机分子可能含有几百个原子，每个原子有几个化学键。CML 寻求以一种直接方式组织这种复杂的化学对象，以便能够让计算机理解，并显示和能够加以检索。CML 可以用于分子结构和序列、光谱分析、结晶学、出版、化学数据库和其他方面。它的词汇表包括分子、原子、化学键、晶体、分子式、序列、对称、反应和其他化学术语。例如，清单 2-1 是描述水（H<sub>2</sub>O）的基本 CML 文档：

清单 2-1：水分子 H<sub>2</sub>O

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

<CML>

<MOL TITLE="Water">

<ATOMS>

<ARRAY BUILTIN="ELSYM">H O H</ARRAY>
```

</ATOMS>

<BONDS>

<ARRAY BUILTIN="ATID1">1 2</ARRAY>

<ARRAY BUILTIN="ATID2">2 3</ARRAY>

<ARRAY BUILTIN="O DE " >1 1</ARRAY>

</BONDS>

</MOL>

</CML>

CML 提供的对传统的管理化学数据的方法的最大改善在于数据的检索。CML 还使得复杂的分子数据可在 Web 上发送。由于 XML 的底层是与平台无关的，所以可以避免由于使用不同的平台而引起的二进制格式不兼容的问题，这种问题在使用传统的化学软件和文档（如 Protein Data Bank (PDB) 格式或者 MDL Molfiles）时常常可以遇到。

Murray-Rust 还创建了第一个通用目的的 XML 浏览器 JUMBO。图 2-1 是 JUMBO 正在显示的一个 CML 文件。Jumbo 将每个 XML 元素赋给能够显示这些元素的 Java 类。为了使 Jumbo 支持新的元素，只要编写用于该元素的 Java 类即可。Jumbo 是与显示基本的一套 CML 元素（其中包括分子、原子和化学键）的类一起发布的。Jumbo 可从 <http://www.xml-cml.org/> 站点处得到。

### 2.1.2 数学标记语言 (Mathematical Markup Language)

传说 CERN 的 Tim Berners-Lee 发明了 World Wide Web 和 HTML，这样一来，高能物理学家们就可以交换论文和印前出版物了。从我个人角度来说，我从不相信这个传说。我是学物理学的，而且我曾在物理、应用数学、天文学和计算机科学等几个学科之间徜徉多年。这几个学科的论文有一点是共同的，就是论文中充满了大量的方程。直到目前为止，Web 已经出现了有九年时间了，还没有找到一种在 Web 页面上包括方程的好办法。

现在有几种办法如 Java 小程序，可以分析自定义的句法，还有一种转换程序，可将用 LaTeX 软件编辑的方程转化为 GIF 图像，另一种是自定义的浏览器，可以读取 TeX 文件，但所有这些办法都不能产生高质量的结果，而且这些都不能满足 Web 作者（即使是科学领域的作者）的需求。最终，只有 XML 才能开始改变这种状况。



图 2-1 显示 CML 文件的 JUMBO 浏览器

数学标记语言（Mathematical Markup Language, MathML）是一种用于数学方程的 XML 应用。MathML 具有足够的能力来处理大多数形式的数学问题从初中的算术到微积分和微分方程。它也可以处理许多更为高级的课题，但还存在一些空白，如在某些数学的分支中使用的更为高级也更为晦涩的记号。虽然对于 MathML 来说，在纯数学和理论物理的高端还有局限性，但是却足以处理几乎所有的教育、科学、工程、商业、经济和统计学上的要求。而且将来 MathML 必然要加以扩展，因而可以认为，即使是最纯粹的数学和纯理论的理论物理都能够在 Web 上出版和研究工作。MathML 完成了 Web 向着科学研究和通信方面的有用工具方向的发展（尽管说它也适用于作为新媒体来制作广告小册子有点离题太远）。

Netscape Navigator 和 Internet Explorer 还不支持 MathML。但是许多数学家都抱着热烈的希望，希望这些浏览器在不久的将来能够对此加以支持。W3C 已经将某些对 MathML 的支持集成到他们的浏览器测试平台 Amaya 中了。图 2-2 是 Amaya 显示的用 MathML 编写的 Maxwell 方程的协变形式。



Amaya 软件可以在本书所附 CD-ROM 的 browsers/amaya 目录中找到。

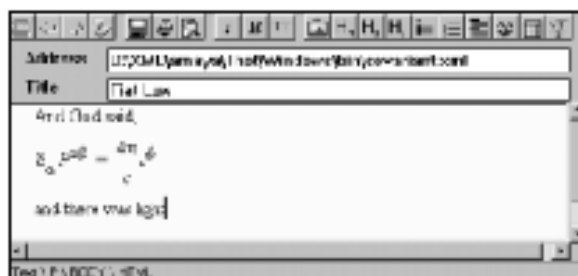


图 2-2 Amaya 浏览器显示的用 MathML 编写的协变形式的 Maxwell 方程

清单 2-2 列出了 Amaya 浏览器正在显示的 XML 文件：

清单 2-2：MathML 中的麦克斯韦（Maxwell）方程

```

<?xml version="1.0"?>

<html xmlns="http://www.w3.org/TR/REC-html40"

xmlns:m="http://www.w3.org/T / EC-MathML/"

>

<head>

<title>Fiat Lux</title>

<meta name="GENERATOR" content="amaya V1.3b" />

</head>

<body>

<P>

And God said,

</P>

<math>

<m:mrow>

<m:msub>

<m:mi>&delta;</m:mi>

<m:mi>&alpha;</m:mi>

</m:msub>

<m:msup>

<m:mi>F</m:mi>

<m:mi>&alpha;&beta;</m:mi>

</m:msup>

<m:mi></m:mi>

<m:mo>=</m:mo>

<m:mi></m:mi>

<m:frac>

```

```

<m:mrow>

<m:m >4</m:m >

<m:mi>&pi;</m:mi>

</m:mrow>

<m:mi>c</m:mi>

</m:mfrac>

<m:mi></m:mi>

<m:msup>

<m:mi>J</m:mi>

<m:mrow>

<m:mi>&beta;</m:mi>

<m:mo></m:mo>

</m:mrow>

</m:msup>

</m:mrow>

</math>

<P>

and there was light

</P>

</body>

</html>

```

清单 2-2 是混合使用 HTML/XML 的页面的例子。其中文本（“Fiat Lux”、“Maxwell’ s Equations”、“And God said”、“and there was light”）的标题和段落是用经典的 HTML 编写的。实际的方程是用 MathML 编写的，这是一个 XML 应用。

一般来说，这种混合页面需要浏览器的特殊支持，这里也正是这种情况，否则就得有插件、ActiveX 控件或是 JavaScript 程序来分析和显示内嵌的 XML 数据。当然最终用户需要像 Mozilla 5.0 或是 Internet Explorer 5.0 这样的浏览器，这两种浏览器可以分析和显示纯 XML 文件，而不需要 HTML 作为中介。

### 2.1.3 频道定义格式



Microsoft 的频道定义格式（Channel Definition Format，简称为 CDF）是用于定义频道的 XML 应用。Web 站点使用频道向预订站点的用户传送信息，一改过去那种坐等用户前来浏览并获取信息的状况。这也叫做 Web 广播或是“推”。CDF 首先是在 Internet Explorer 4.0 中引入的。

CDF 文档是一个 XML 文件，与被推的站点的 HTML 文件分别存放，但是却链接到此 HTML 文件上。CDF 文档中的频道定义决定了要发送哪个页面。页面可以通过发送通知向预订者加以推送，但也可以发送整个站点，或是由阅读者在方便的时候自己来“拉”信息。

用户可向自己的站点添加 CDF，而不用改变现存的所有内容。只要在页面上添加与 CDF 文件的一个不可见的链接即可。当浏览者访问这个页面时，浏览器显示一个对话框，询问浏览者是否要预订频道。如果浏览者选择了预订，则浏览器就下载描述频道的 CDF 文档。然后浏览器将 CDF 文档用指定的参数与用户自己的选项结合起来，以便决定什么时候检查服务器上的新内容。这实际上不是真正的“推”，因为客户必须初始化连接，但是这确实是在没有浏览请求的情况下发生的。图 2-3 是 IDG 的 Active Channel（活动频道）显示在 Internet Explorer 4.0 中的情况。

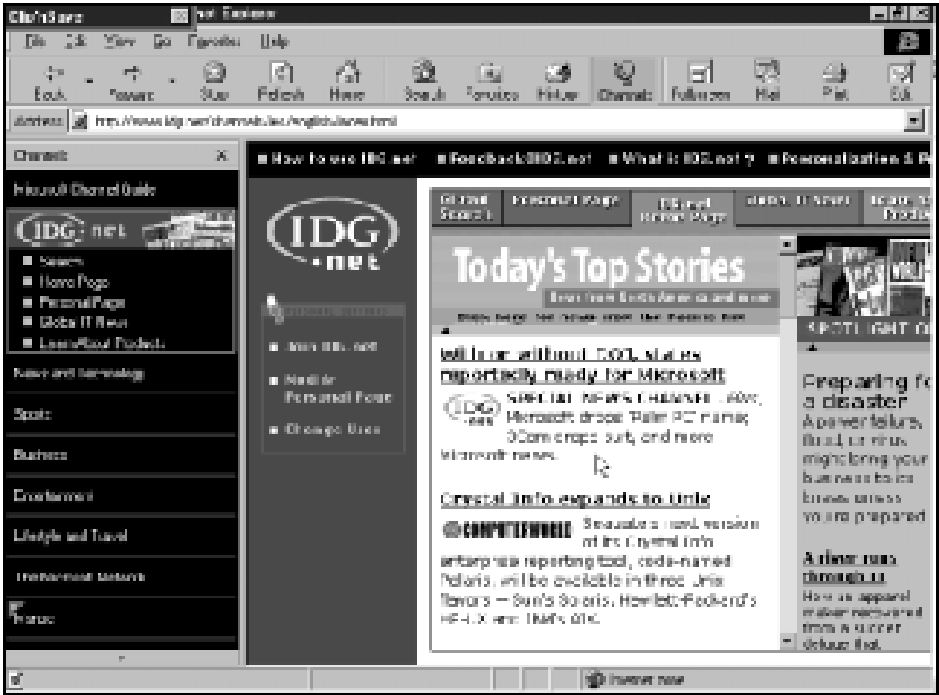


图 2-3 在 Internet Explorer 4.0 中显示的 IDG 的 Active Channel（活动频道）



在第 21 章“用 CDF 推送 Web 站点”中将详细地讨论 CDF。



Internet Explorer 4.0 可在本书所附 CD-ROM 上的 browsers/ie4 目录中找到。

### 2.1.4 经典文学

Jon Bosak 曾经将 Shakespeare（莎士比亚）的全部话剧翻译成了 XML。这些剧本的全文都包括其中了，用 XML 标记来区分剧名、每幕标题、舞台指导、对白、台词、旁白等。



莎士比亚的全套话剧可以本书所附 CD-ROM 上的 examples/shakespeare 目录中找到。

读者可能要问，对于一本书或是一个普通的文本文件来说，这样做有什么好处呢？对于人类读者来说，这没有什么不同，但对分析文字的计算机来说，这样做就使得容易区分组成话剧的不同元素。例如，要让计算机在全文中找出 Romeo（罗密欧）的台词就变得简单了。

进一步说，借助于改变格式化文档的样式单，某个演员就很容易地打印出该剧的一个副本，其中他（她）的所有台词都格式化为粗体，而他（她）前面和后面的台词都用斜体来表示。另外还可以想像出来的事是，将剧本分成不同人的道白时，利用 XML 格式化的版本也比原来的文本要容易得多。

Bosak 曾经将新旧约全书、古兰经和摩门教教义的英文译本用 XML 加以标记。这些书中的标记有些不同。例如，它并不对讲话人加以区分。因而（比如说）也就不能利用这种特殊的 XML 文档来创建带红色字母的圣经，虽然使用不同的一套标记可以达到这一目的。（带红色字母的圣经将耶稣说的话用红色印刷。）而且由于这些文件是用英语写成的，而不是原来的语言，这对于学术上的文本分析来说，就不是那么有用了。如果时间和资源允许的话，只要愿意，用 XML 来书写原文也是可以办得到的。这时只要设计一套与 Bosak 使用的不同，但却是描述同样的数据的词汇表和句法即可。



经 XML 标记了的圣经、古兰经和摩门教教义都可在本书所附的 CD-ROM 上的 `examples/religion` 目录中找到。

## 2.2 用于 XML 的 XML

XML 对于文本数据来说是最通用的格式。它所用于的某些事物还进一步地完善了 XML 本身。这包括 XSL 样式单语言、XLL 链接语言和用于 XML 的文档内容描述 (Document Content Description, 简称为 DCD)。

### 2.2.1 XSL

XSL (Extensible Style Language, 可扩展的样式语言) 本身就是 XML 应用。XSL 有两个主要部分。第一部分定义了将 XML 文档加以转换的词汇表。这一部分的 XSL 包括用于树的 XML 标记、节点、式样、模板和其他用于将 XML 文档从一种标记词汇转换成另一种 (或是同一种却以不同的顺序) 所需要的元素。

XSL 的第二部分定义了用于格式化转换后的 XML 文档 (由第一部分产生的) 的词汇表。这包括用于格式化对象 (如分页、块、字符、列表、图形、方框、字体和其他) 的 XML 标记。清单 2-12 中列出了一个典型的 XSL 样式单:

清单 2-12: 一个 XSL 样式单

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

<xsl:stylesheet

xmlns:xsl="http://www.w3.org/T /WD-xsl"

xmlns:fo="http://www.w3.org/T /WD-xsl/FO"

result-ns="fo">

<xsl:template match="/">

<fo:basic-page-sequence >

<xsl:apply-templates/>

</fo:basic-page-sequence>

</xsl:template>

<xsl:template match="ATOM">

<fo:block font-size="10pt" font-family="serif" space-before="12pt">

<xsl:value-of select="NAME"/>

</fo:block>

</xsl:template>

</xsl:stylesheet>
```



我们将在第 14 章和 15 章中详细讨论 XSL。

### 2.2.2 XLL

可扩展的链接语言（Extensible Linking Language，简称为 XLL）定义了新的名为 XLink 的更一般种类的链接。XLinks 可完成用 HTML 中的以 URL 为基础的超链接所能完成的所有任务。例如，脚注元素可像下例一样直接链接注解的文本：

```
<footnote xlink:form="simple" href="footnote7.xml">7</footnote>
```

进一步说，XLink 可以做 HTML 链接不能做的事。XLink 可以是双向的，因而读者可以返回原来所在的页面（跳转前所在页面）。XLink 可以链接到文档中的任意位置。XLink 可将文本或是图形数据嵌入文档内部，而不需要用户去激活链接（更像 HTML 中的<IMG>标记，但更灵活）。简短说，XLink 使超链接的功能更为强大。



在第 16 章“XLink”中将要更加详细地讨论 XLink 方面的内容。

### 2.2.3 DCD

XML 的用于声明 XML 元素内容应该如何格式化的工具对于不存在的内容显得功能不足。例如，假设作为数据的一部分，像下面一样建立了 MONTH 元素：

```
<MONTH>9</MONTH>
```

我们能看到 MONTH 元素的内容应该是字符数据。我们不能说必须给这个元素以从 1 到 12 的整数。

已经提出了几种 XML 本身的方案，以便更严格地限制什么可以出现在任意给定的内容中。有一种方案就是文档内容描述（Document Content Description，简称为 DCD）例如，这里有一个 DCD，声明了 MONTH 元素只能含有 1 到 12 的整数：

```
<DCD>
```

```
<ElementDef Type="MONTH" Model="Data" Datatype="i1"
```

```
Min="1" Max="12" />
```

```
</DCD>
```

我还可以向读者展示好多的用于 XML 的例子，但是上例已经表明了基本的观点：XML 强大得足以来描述和扩展本身。此外，这还意味着，XML 规范可以保持短小和简单。完全可以没有 XML 2.0，因为任何主要的所需的附加内容都可以根据原来的 XML 加以建立，而不必成为 XML 的新功能。需要加强功能的人们和程序员们可以使用这些新功能，而不需要的人可以将其忽略。用户不必了解什么是不使用的。XML 提供了“砖和泥”，利用这些“砖和泥”既可以建起“小屋”也可以建起高耸的“城堡”。

## 2.3 XML 的后台应用

并不是所有的 XML 应用都是公开的、开放的标准。有许多软件开发商正在将其自身的数据转向 XML，只是因为 XML 是被公众很好理解的、通用目的格式，可以用容易获得的、便宜或免费的工具加以处理。

Microsoft Office 2000 已将 HTML 变为与它的内建二进制格式同等的格式。不过，HTML 4.0 还不能提供对 Office 所需的所有功能的全面支持，如修订跟踪、脚注、批注、索引和术语表项等等。不能用 HTML 表达的附加数据嵌入到 XML 的小型代码块中。Word 的矢量图形保存在 VML 中。在这种情况下，嵌入的 XML 在标准的浏览器中的不可见性是个关键因素。

Federal Express 公司将详细跟踪的信息用作为与其他送货公司（如 UPS（美国快寄服务公司和 Post Office（邮局））相比更有竞争力的优点。首先这种信息来源于顾客软件，然后通过 Web。最近，FedEx 公司开始对其 API（应用程序接口）和库函数（第三方和内部开发者可使用这些 API 将他们的软件和系统与 FedEx 的加以集成）的测试。这种服务的数据格式就是 XML。

Netscape Navigator 5.0 支持 XML 在 Web 浏览器上的直接显示，但是，Netscape 实际在内部早在 4.5 版时就已经开始使用 XML 了。当用户请求 Netscape 显示与当前站点相联系的站点的列表时，浏览器就连接到运行在 Netscape 服务器上的一个 CGI 程序上。服务器送回来的数据就是 XML。清单 2-13 就是与站点 <http://metalab.unc.edu/> 相联系的站点的 XML 数据：

清单 2-13：与 <http://metalab.unc.edu/> 相联系的站点的 XML 数据

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

<RDF:RDF>

<RelatedLinks>

<aboutPage

href="http://info.netscape.com/fwd/rl/http://metalab.unc.edu:80/*">

</aboutPage>

<child instanceOf="Separator1"></child>

<child

href="http://info.netscape.com/fwd/rl/http://www.sun.com/"

name="Sun Microsystems">

</child>

<child

href="http://info.netscape.com/fwd/rl/http://www.unc.edu/"

name="Unc">

</child>

<child
```

`href="http://info.netscape.com/fwd/rl/http://sunsite.sut.ac.jp/"`

`name="SunSITE Japan">`

`</child>`

`<child`

`href="http://info.netscape.com/fwd/rl/http://sunsite.nus.sg/"`

`name="SunSITE Singapore">`

`</child>`

`<child`

`href="http://info.netscape.com/fwd/rl/http://sunsite.berkeley.edu/"`

`name="Berkeley Digital Library SunSITE">`

`</child>`

`<child`

`href="http://info.netscape.com/fwd/rl/http://www.sun.com/sunsite"`

`name="SunSITE on the net">`

`</child>`

`<child`

`href="http://info.netscape.com/fwd/rl/http://www.sunsite.auc.dk/"`

`name="SunSITE Denmark">`

`</child>`

`<child`

`href="http://info.netscape.com/fwd/rl/http://sunsite.edu.cn/"`

`name="SunSITE China">`

`</child>`

`<child`

`href="http://info.netscape.com/fwd/rl/http://sunsite.stanford.org/"`

`name="Stanford University SunSITE">`

```

</child>

<child

href="http://info.netscape.com/fwd/rl/http://www.cdromshop.com/

cdshop/desc/p.061590000085.html" name="SunSITE Archive">

</child>

<child instanceOf="Separator1"></child>

<child instanceOf="Separator1"></child>

<child href="http://home.netscape.com/escapes/smart_browsing"

name="Learn About Smart Browsing...">

</child>

</RelatedLinks>

</RDF:RDF>

```

这一切都完全发生在幕后。用户决不会知道那些数据正在用 XML 加以传送。实际上显示的是 Netscape Navigator 中的菜单，而不是 XML 或 HTML 页面。

这些实际上还只是将 XML 用于内部数据的不成熟的表面现象。许多其他使用 XML 的项目还刚刚起步，还有一些项目将在明年起步。大多数这样的项目不会受公开注意，也不会在商业出版物上受到吹捧，但是不管怎样，在其项目存活期内它们都具有潜力可为公司节约成千上万美元的开发费用。XML 的自说明性对于公司内部的数据也是很有用的。例如，许多公司现在正在匆忙地设法找出 20 年前退休的程序员是否用了两位数字的日期。如果你正在干这样的事情，你是愿意将数据写成下面的样子呢：

```
3c 79 65 61 72 3e 39 39 3c 2f 79 65 61 72 3e
```

还是下面的样子：

```
<YEAR>99</YEAR>
```

不幸的是，许多程序员现在还坚持将数据写成第一种格式。XML 还可使错误容易发现和修改。

## 2.4 本章小结

本章只是刚刚接触到已经和将要使用 XML 的应用。一些应用，如 CML、MathML 和 MusicML 很明显是用于 Web 浏览器的 HTML 扩展。但是许多别的应用，如 OFX、XFDL 和 HRML 完全走的是另一条路。所有这些应用都有建立在 XML 之上的自己的语义和句法。在某些情况下，XML 的“根”是很明显的，但在另外一些情况下，即使在其上工作达一月之久，也不一定会发现它与 XML 有什么关系。在本章中，我们讨论了下面的可使用 XML 的应用：

- 使用 CML 的分子科学
- 使用 MathML 的科学数学
- 使用 CDF 的 Web 广播
- 古典文学
- 使用 SMIL 和 HTML+TIME 的多媒体应用
- 通过 OSD 的软件更新
- 使用 PGML 和 VML 的矢量图形
- 用 MusicML 表示的音乐记号
- 使用 VoxML 的自动语音响应
- 使用 OFX 的财务数据
- 与 XFDL 合法捆绑的表单
- 使用 HRML 的人力资源工作信息
- 通过 RDF 表示的元数据 (Meta-data)
- XML 本身包括 XSL、XLL 和 DCD 使 XML 更加完善
- 许多公司在因特网上应用 XML，这些公司包括 Microsoft、Federal Express 和 Netscape

在下一章中，读者将要学习编写自己的 XML 文档并在 Web 浏览器上加以显示。



## 第 3 章 第一个 XML 文档

本章教读者用自己定义的可为文档所理解的标记来创建简单的 XML 文档。读者将学到如何编写样式单，以便用于在文档中描述标记内容如何显示。最后，还要学到如何将文档装到 Web 浏览器中以便查看。

由于本章利用示例来加以讲解，而不是从原理出发，因而不会涉及许多细节。有经验的读者将会注意到几处例外和特殊情况没有在本章加以讨论。对此不必担心。在下几章中将会讨论到。对于大部分内容，不必太关心技术内容。正如 HTML 一样，也可通过复制其他人创建的简单的示例并按自己的需要加以修改来学习。

为了达到上述目的，我鼓励大家按我在本章中给出的示例键入程序逐步进行，并将这些代码装入讨论过的不同的程序中。这将使读者对 XML 产生基本感受，这将使在未来几章中提到的技术细节在特定示例的环境中容易掌握。

本章的主要内容包括：

- 创建简单的 XML 文档
- 仔细研究这个简单的 XML 文档
- 赋给 XML 标记以具体意义
- 在 XML 文档上附加样式单

### 3.1 Hello XML

本节遵照老程序员介绍新语言的传统，先用一个能够在屏幕上打印出“Hello World”的程序加以介绍。XML 是标记语言，而不是编程语言，但是基本原理还是适用的。最简单的方法是以一个完全的可运行的有扩展能力的示例开始，而不要尝试以更基本的无任何功能的程序开始。如果用户在使用基本的工具时确实遇到了问题，在简短的文档环境中也比在复杂的文档环境下更容易调试和改正。

在本节中，读者将学到如何创建一个简单的 XML 文档并将其保存在文件中。然后我们对其中的代码及其意义再加以仔细考察。

#### 3.1.1 创建一个简单的 XML 文档

在本节中，读者将学到如何键入一个实际的 XML 文档。我们从能够想像得到的最简单的 XML 文档开始。这个文档列在清单 3-1 中：

清单 3-1: Hello XML

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

<FOO>

Hello XML!

</FOO>
```

这虽然不太复杂，但却是一个“好”的 XML 的文档。更准确地说，这是一个结构完整的 XML 文档（XML 中有一些用于文档的专门术语，依照到底满足了哪条规则而被认为是“好”的。其中“结构完整的”就是一条这样的术语，在本书的后面要对此加以讨论。）可在任何使用方便的文本编辑器，如 Notepad、BBEdit 或是 emacs 中键入这个文档。



结构完整性将在第 6 章“结构完整的 XML 文档”中加以讨论。

### 3.1.2 保存 XML 文件

当键入了上面的代码之后，请将该文档保存在名为 hello.xml 的文件中。也可以使用诸如 HelloWorld.xml、MyFirstDocument.xml 或是其他文件名，但三个字母的扩展名.xml 是标准的，一般不要更改。而且还要确保以普通的文本格式加以保存，而不要用某些字处理程序，如 WordPerfect 或 Microsoft Word 的内建格式。

如果使用的是 Windows 95/98 上的 Notepad 来编辑文件，当保存文档时，一定要将文件名用双引号括起来，即“Hello.xml”，而不要只是 Hello.xml，正如图 3-1 所示的一样。如果没有引号，Notepad 会在文件名后再加上.txt 扩展名，也就是文件名变成了 Hello.xml.txt，这完全不是我们所希望出现的。

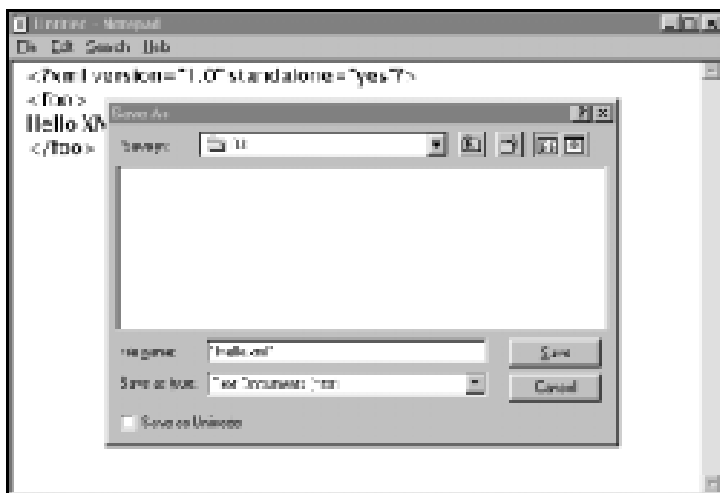


图 3-1 在 Notepad 中用带引号的文件名来保存 XML 文档

Windows NT 版本的 Notepad 还会给出将文件保存为 Unicode 格式的选项。令人惊奇的是，这样保存也可以，不过我们还是坚持使用基本的 ASCII 文本格式比较好。XML 文件既可以是 Unicode 格式也可以是 Unicode 的名为 UTF-8 的压缩版本，这是严格的 ASCII 的超集，因而纯 ASCII 文件也是合法的 XML 文件。



UTF-8 和 ASCII 将在第 7 章“外国语言和非罗马文本”中加以更为详细的讨论。

### 3.1.3 将 XML 文件装入 Web 浏览器

既然已经创建了第一个 XML 文档，当然想看一看了。这个文件可以在支持 XML 的浏览器，如 Internet Explorer 5.0 中直接打开。图 3-2 显示的就是结果。

我们看到的结果将依不同的浏览器而有所不同。在本例情况下，文件是格式化得很好的，以不同的颜色来表示不同的句法。不过所看到的并没有吸引人的地方。问题在于浏览器并不了解如何处理 FOO 元素。我们必须指示浏览器如何处理每个元素，这就要用到样式单了。我们将要简单地介绍一下，但首先还是仔细地考察一下这个文档。



## .2 考察简单的 XML 文档

让我们检查一下列在清单 3-1 中的这个简单的 XML 文档，以便更好地理解每行代码的意义。第一行是 XML 声明：

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

这是 XML 处理指令的例子。处理指令以<?开始，而以?>结束。在<?后的第一个单词是处理指令名，在本例中是 xml。

XML 声明有 version 和 standalone 两个特性。特性是由等号分开的名称-数值对。位于等号左边的是特性名，而其值位于等号的右边，并用双引号括起来。

每一个 XML 文档都以一个 XML 声明开始，用以指明所用的 XML 的版本。在上例中，version 特性表明这个文档符合 XML 1.0 规范。XML 声明还可以有 standalone 特性，这告诉我们文档是否在这一个文件里还是需要从外部导入文件。在本例中，以及在以后的几章中，所有的文档都在一个文件里完成，因而 standalone 特性的值要设置为 yes。

现在让我们看一下清单 3-1 中的下面的三行：

```
<FOO>
```

```
Hello XML!
```

```
</FOO>
```

总体上说，这三行组成了 F00 元素。分开说，<F00>是开始标记，而</F00>是结束标记，Hello XML!是 F00 元素的内容。

读者可能要问，<F00>标记的意义是什么？回答是“你要让它是什么就是什么”。除了几百个预定义的标记之外，XML 还允许用户创建所需的标记。因而<F00>标记可以具有用户赋予的任何意义。同一个 XML 文档可以用不同的标记名编写，正如清单 3-2、3-3 和 3-4 所表明的：

清单 3-2: greeting.xml

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

```
<GREETING>
```

```
Hello XML!
```

```
</GREETING>
```

清单 3-3: paragraph.xml

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

```
<P>
```

```
Hello XML!
```

```
</P>
```

清单 3-4: document.xml

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

```
<DOCUMENT>
```

```
Hello XML!
```

```
</DOCUMENT>
```

清单 3-1 到 3-4 这四个文档用的标记名各不相同，但都是等价的，因为具有相同的结构和内容。

### 3.3 赋予 XML 标记以意义

标记可有三类意义：结构、语义和样式。结构将文档分成元素树。语义将单个的元素与外部的实际事物联系起来。而样式指定如何显示元素。

结构只是表达文档的形式，而不管单个标记和元素间的差别。例如，上面清单 3-1 到 3-4 中的四个 XML 文档结构是相同的。它们都指定文档具有一个非空的基本元素。标记的不同名称没有结构上的意义。

语义的意义存在于文档之外，在作者的心中或是读者或是某些生成或读取这些文件的计算机程序中。例如，理解 HTML 但不理解 XML 的 Web 浏览器，可能会将段落的意义赋给<P>和</P>标记，但不会赋给标记<GREETING>和</GREETING>、<FOO>和</FOO>或是<DOCUMENT>和</DOCUMENT>。讲英语的人可能会比<FOO>和</FOO>或<P>或</P>更容易理解<GREETING>和</GREETING>或是<DOCUMENT>和</DOCUMENT>的意义。正如“美丽”的意义存在于观察者心中。

计算机作为一个哑机器，不能说是真正地理解任何事物的意义。计算机只是根据预先确定的公式来处理位和字节而已（虽然非常快）。对于一台计算机而言，用<FOO>或是<P>与使用<GREETING>或<DOCUMENT>标记没有什么差别。即使对于 Web 浏览器来说，也不能说它理解什么是段落。所有的浏览器了解的是，当遇到一个段落时，在下一个元素前面要放置一个空行。

自然地，使标记的名称能够尽可能反映其包含的意义更好一些。许多学科，如数学和化学正在创建该学科的工业标准和标记集。如果合适的话，应该使用这些标准和标记集。但是大多数情况下，还是需要什么标记就创建什么标记。

以下是一些其他可能的标记：

<MOLECULE> <INTEGRAL>

<PERSON> <SALARY>

<author> <email>

<planet> <sign>

<Bill> <plus/>

<Hillary> <plus/>

<Gennifer> <plus/>

<Paula> <plus/>

<Monica> <equals/>

<divorce>

可以与标记相联系的第三类意义是样式意义。样式意义指定标记的内容如何在计算机屏幕上或是其他输出设备上展示。样式意义说明特定的元素是否是用粗体、斜体、绿色的 24 磅的字体还是其他字体加以表示。计算机在理解样式时比理解语义意义要好一些。在 XML 中，样式意义是通过样式单来施加的。

## . 4 为 XML 文档编写样式单

XML 允许用户来创建任何所需要的标记。当然，由于用户在创建标记上有完全的自由，因而通用的浏览器无法预期用户的标记的意义，也无法为显示这些标记而提供规则。因而，用户必须为文档编写样式单，告诉浏览器如何显示特定的标记。与标记集类似，用户创建的样式单可由不同的文档不同的人所共享，还可将自己创建的样式单与其他人编写的样式单集成在一起。

正如在第 1 章中所讨论的，现在有不只一种样式单语言可以使用。这里所用的是级联样式单（Cascading Style Sheets，简称为 CSS）。CSS 的优势在于它是 W3C 制定的标准，为编写 HTML 的许多人所熟悉，且被前卫的具有 XML 能力的浏览器所支持。



正如在第 1 章所注意到的，另一种可能的选择是可扩展的样式语言（Extensible Style Language）。XSL 是当前最强大和灵活的样式语言，是特别为应用 XML 而设计的。但是，XSL 比 CSS 更为复杂，而且未被很好地支持，同时还没有完成。



XSL 将在第 5、14 和 15 章中加以讨论。

清单 3-2 中的 `greeting.xml` 示例只包括一个标记 `<GREETING>`，因而所需做的一切是为 `GREETING` 元素定义样式。清单 3-5 是一个很简单的样式单，指定 `GREETING` 元素的内容应该以 24 磅的粗体显示为块级的元素。

清单 3-5: `greeting.xml`

```
GREETING{display: block; font-size: 24pt; font-weight: bold;}
```

清单 3-5 应该在文本编辑器中键入，保存为名为 `greeting.css` 的新文件，放在与清单 3-2 中的文件所在的同一目录中。扩展名 `.css` 代表级联样式单（Cascading Style Sheet）。同样 `.css` 扩展名是重要的，而文件名却不怎么重要。如果打算将这一样式单只用在 XML 文档上的话，那么与 XML 具有同样的文件名（扩展名为 `.css` 而不是 `.xml`）常常更为方便。

## 5 将样式单附加到 XML 文档上

在编写好 XML 文档和用于该文档的 CSS 样式单之后，还需要告诉浏览器将样式单作用到该文档上。长时期以来，可能有许多不同的方法可达到这一目的，包括浏览器-服务器通过 HTTP 文件头协商、命名约定和浏览器一侧的缺省方法。但是目前，唯一的有效方法是在 XML 文档中包括另一个处理指令，以便指定所要使用的样式单。

处理指令是<?xml-stylesheet?>和它的两个特性，type 和 href。type 特性指定所用的样式语言，而 href 特性指定一个可以找到样式单的 URL（可能是相对的）。在清单 3-6 中，xml-stylesheet 处理指令指明施加于文档的样式单文件名为 greeting.css，是用 CSS 样式单语言编写的。

清单 3-6：带有 xml 样式单处理指令的 greeting.xml

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

<?xml-stylesheet type="text/css2" href="greeting.css"?>

<GREETING>

Hello XML!

</GREETING>
```

既然我们已经创建好了第一个 XML 文档和样式单，那么当然想看一看结果了。我们所要做的就是将清单 3-6 装入 Mozilla 或是 Internet Explorer 5.0。图 3-3 是显示在 Internet Explorer 5.0 中的具有样式的欢迎画面。图 3-4 是显示在早期开发版本的 Mozilla 中的具有样式的欢迎画面。



图 3-3 在 Internet Explorer 5.0 中显示的 styledgreeting.xml 文件



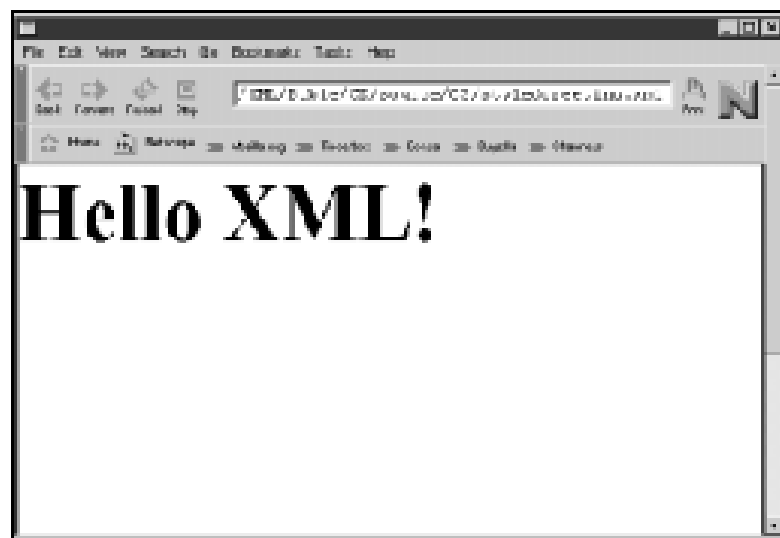


图 3-4 在早期的开发者版本的 Mozilla 中显示的 styledgreeting.xml 文件

## 3.6 本章小结

在本章中，读者学到了如何创建一个简单的 XML 文档。总的来说，包括以下内容：

- 如何编写和保存简单的 XML 文档
- 如何把三种类型的意义（结构、语义和样式）赋给 XML 标记
- 如何为 XML 文档编写 CSS 样式单，从而告诉浏览器如何显示特定的标记
- 如何将带有 `xml-stylesheet` 处理指令的 CSS 样式单附加到 XML 文档上
- 如何将 XML 装入浏览器中

在下一章中，我们将要研究 XML 文档的更为大型的例子，用来演示在选择 XML 标记时的更多的实际考虑。

## 第 4 章 数据的结构化

在本章中，我们将要研究一个较长的示例，用来说明一个较长的有关棒球统计和其他类似数据的列表是如何以 XML 格式保存的。像这样的文档有好多潜在的应用。最明显的，它可以显示在 Web 页面上。还可以用作其他分析数据或是整理数据程序的输入。通过这个示例，读者将学到如何用 XML 来标记数据、为什么要选用 XML 标记、如何为文档编制 CSS 样式单等内容。

本章的主要内容包括：

- 检查数据
- 数据的 XML 化
- XML 格式的优越性
- 为文档的显示编制样式单

### 4.1 检查数据

当我写作这本书时（1998 年 10 月），纽约的 Yankees 队在四场比赛中击败圣·迭格的 Padres 队，取得了 24 届世界系列赛的冠军。Yankees 队在 American League 的普通赛季结束时，取得了 114 场胜利。总体来说，1998 是一个令人赞叹的赛季。圣·路易斯 Cardinals 队的 Mark McGwire 和芝加哥 Cubs 队的 Sammy Sosa 为了创造新的单一赛季的本垒打纪录在整个 9 月份展开了争夺，原来的纪录是由 Roger Maris 保持的。

是什么使 1998 赛季这样激动人心呢？玩世不恭的人会告诉你，1998 是一扩展年，有三个新队加盟，因而总体上来说投手能力减弱了。这就使得著名的击球手如 Sosa 和 McGwire 以及著名的球队，如 Yankees 得到了出风头的机会，因为，虽然他们仍然像他们在 1997 年一样实力强大，但面对的对手的平均能力弱了许多。当然真正的棒球爱好者了解真正的原因，这是由于统计上的原因造成的。

这实在有点滑稽。在大多数体育项目中，我们都说过心脏、勇气、能力、技巧、决心和其他名词。但是，只有棒球爱好者需要面对这么多原始数字，如平均击球率、平均得分、平均跑垒数、平均进垒数、对左手投手的平均击球率、对右手投手的平均击球率等。

棒球爱好者都被这些数字所迷住了，数字越多越好。在每个赛季中，因特网成了成千上万的棒球爱好者的大本营，狂热的网民们在其中“管理”球队并交换球员，计算他们喜爱的球队在现实中表现的各种数字。STATS, Inc. 公司跟踪了每个球员在主要联赛的赛事上的表现，因而可以计算出一个击球手是否表现得比他的平均成绩要好。在以下两节中，为了照顾对棒球不太感兴趣的读者，我们检查一下描述单个球员的击球和投球率的常用统计数字。现场统计数字也可以找到，但是我将把这些数字略去，以便将示例局限于好管理的大小。我使用的这个特殊的例子是纽约的 Yankees 队，对于任何队的击球手，同样的统计数字也可以得到。

#### 4.1.1 击球手


几年前，Bruce Bukiet、Jose Palacios 和我写过一篇名为 A Markov Chain Approach to Baseball（用于棒球的马尔可夫链式方法）的文章（刊登在 Operations Research（运筹学研究杂志），45 卷第 1 期，1997 年 1-2 月号，pp. 14-23，还可在以下网址上看到这篇文章 <http://www.math.njit.edu/~bukiet/Papers/ball.pdf>）。在这篇文章中，我们分析了 1989 年全国棒球联赛中的所有球队的所有可能的比赛顺序。那篇文章的结果还是比较有意思的。球队中的最坏的击球手（通常是投球手）应该是第 8 位出场击球的人，而不应该是第 9 位，至少在全国棒球联赛上是如此。但是这里我所关心的是产生那篇文章的工作。作为一个低年级的研究生，用手工算出每个球员在全国棒球联赛上的全部击球历史记录正是我的工作。如果我能够使那些数据变得像 XML 一样的方便，那个夏季我会过得更愉快一些的。现在，让我们将精力集中于每个球员的数据上。典型的，这种数据是以一行行的数字表示的，如表 4-1 所示的是 1998 年 Yankees 队的进攻队员的数据。在美国棒球联赛的比赛上，由于投球手很少击球，只有实际上击球的队员才列在表中。

每一列有效地定义了一个元素。因而就需要为球员、位置、进行的比赛、击球、跑垒 、击球数、两垒、三垒、本垒打、跑入和步行等建立元素。单垒通常都不单独报告。这个数据是从总击打数中减去双垒、三垒和本垒打的总和后得到的。

表 4-1 The 1998 年 Yankees 队的进攻队员数据

Name	Postion	Game Played	At Bats	Runs	Hits	Doubles	Triples	Home Runs	Runs Batted In	Strike Walks	Outs	Hit by Pitch
Scott Brosius	Third Base	152	530	86	159	34	0	19	98	52	97	10
Homer Bush	Second BBase	45	71	17	27	3	0	1	5	5	19	0
Chad Curtis	Outfield	151	456	79	111	21	1	10	56	75	80	7
Chili Davis	Designated Hitter	35	103	11	30	7	0	3	9	14	18	0
Mike Figga	catcher	1	4	1	1	0	0	0	0	0	1	0
Joe Girardi	catcher	78	254	31	70	11	4	3	31	14	38	2
Derek Jeter	Shortsho	149	626	127	203	25	8	19	84	57	119	5
Chuck Knoblauch	Second Base	150	603	117	160	25	4	17	64	76	70	
Ricky Ledee	Outfield	42	79	13	19	5	2	1	12	7	29	0
Mike Lowell	Third Base	8	15	1	4	0	0	0	0	0	1	0
Tino Martinez	First Base	142	531	92	149	33	1	28	123	61	83	6
Paul O’ Neill	Outfield	152	602	95	191	40	2	24	116	57	103	2
Jorge Posada	catcher	111	358	56	96	23	0	17	63	47	92	0
Tim Raines	Outfield	109	321	53	93	13	1	5	47	55	49	3
Luis Sojo	Shortshop	54	147	16	34	3	1	0	14	4	15	0
Shane Spencer	Outfield	27	67	18	25	6	0	10	27	5	12	0
Darryl Strawberry	Designated Hitter	101	295	44	73	11	2	24	57	46	90	3
Dale Sveum	First Base	30	58	6	9	0	0	0	3	4	16	0
Bernie Williams	Outfield	128	499	101	169	30	5	26	97	74	81	1

译者注：棒球数据不过是一种演示。在棒球统计数据的 XML 文档中，由于使用的是英文专用名词，故这里未翻译成中文。如果翻译过来反而无法相互对照。表 4-2 也同样处理。

前面表中的数据和下一节中的投球手数据都是加以限制后的列表，只是用来表明在一个典型的棒球赛中收集的数据。除了列出的以外，还有许多其他数据没有在这里列出。我打算使用这些基本信息，以便使示例容易管理。

### 4.1.2 投球手

人们并不指望投球手成为全垒跑的击球手或是偷袭能手。确实偶尔到达第一垒的投球手是对一个队的意外奖励。对投球手的评价要根据表 4-2 中列出的全场的不同种类的数字。这个表的每列也定义了一个元素。这些元素中的一部分，如姓名和位置对于投球手和击球手都是有的。其他元素如解救（saves）和成功防守（shutouts）只适用于投球手。还有几个，如得分（runs）和全垒跑（home runs）与击球手统计中的名称相同，但是具有不同意义。例如，击球手的得分是击球手获得的分数。而对于投球手来说，是指对方在这个投球手下得到的分数。

### 4.1.3 XML 数据的组织

XML 是建立在容器模型的基础之上的。每个 XML 元素可以包含文本或是称为子元素的其他 XML 元素。有几个 XML 元素既可以包含文本也可以包含子元素。虽然通常来说，这并不是一种好形式，是应该尽量避免的。

不过，常常有不止一种组织数据的方法，这要取决于需要。XML 的一个好处是，它使得编写程序来以不同形式组织数据变得相当直接。在第 14 章我们讨论 XSL 变换时还要讨论这一问题。

作为开始，必须注意的第一个问题是什么包含什么？例如，相当明显的是，联赛包含分部，分部包含球队，球队又包含球员，而球员又可在指定的时间进行交易，每个球员必定属于一个球队，每个球队又必定属于一个分部。类似的，一个赛季包含许多场比赛，每场比赛又包含几局，而局又包含击球阶段，击球阶段又包含投球阶段。

但是，赛季包括联赛吗或是联赛包括赛季吗？这个问题就不是很明显。确实对这样的问题没有唯一的答案。将赛季元素定义为联赛元素的子元素还是将联赛元素变为赛季元素的子元素有更多的意义，这要依赖于数据要用来干什么。用户甚至可以创建新的既包含赛季也包含联赛的根元素，哪个元素也不是另外元素的子元素（虽然要有效地这样做，还需要某些先进的技术，在以下几章还讨论不到这些技术）。用户可按用户的意愿来组织数据。

表 4-2 1998 年 Yankees 队的投球手

Name	P	W	L	S	G	GS	CG	SHO	ERA	IP	H	HR	R	ER	HB	WP	BK	WB	SO
Joe Borowski	Relief Pitcher	1	0	0	8	0	0	0	6.52	9.2	11	0	7	7	0	0	0	4	7
Ryan Bradley	Relief Pitcher	2	1	0	5	1	0	0	5.68	12.2	12	2	9	8	1	0	0	9	13
Jim Bruske	Relief Pitcher	1	0	0	3	1	0	0	3	9	9	2	3	3	0	0	0	1	3
Mike Buddie	Relief Pitcher	4	1	0	24	2	0	0	5.62	41.2	46	5	29	26	3	2	1	13	20
David Cone	Starting Pitcher	20	7	0	31	31	3	0	3.55	207.2	186	20	89	82	15	6	0	59	209
Todd Erdos	Relief Pitcher	0	0	0	2	0	0	0	9	2	5	0	2	2	0	0	0	1	0
Orlando Hernandez	Starting Pitcher	12	4	0	21	21	3	1	3.13	141	113	11	53	49	6	5	2	52	131
Darren Holmes	Relief Pitcher	0	3	2	34	0	0	0	3.33	51.1	53	4	19	19	2	1	0	14	31
Hideki Irabu	Starting Pitcher	13	9	0	29	28	2	1	4.06	173	148	27	79	78	9	6	1	76	126
Mike Jerzembeck	Starting Pitcher	0	1	0	3	2	0	0	12.79	6.1	9	2	9	9	0	1	1	4	1
Graeme Lloyd	Relief Pitcher	3	0	0	50	0	0	0	1.67	37.2	26	3	10	7	2	2	0	6	20
Ramiro Mendoza	Relief Pitcher	10	2	1	41	14	1	1	3.25	130.1	131	9	50	47	9	3	0	30	56
Jeff Nelson	Relief Pitcher	5	3	3	45	0	0	0	3.79	40.1	44	1	18	17	8	2	0	22	35
Andy Pettitte	Starting Pitcher	16	11	0	33	32	5	0	4.24	216.1	226	20	10	2	6	5	0	87	146
Mariano Rivera	Relief Pitcher	3	0	36	54	0	0	0	1.91	61.1	48	3	13	13	1	0	0	17	36
Mike Stanton	Relief Pitcher	4	1	6	67	0	0	0	5.47	79	71	13	51	48	4	0	0	26	69
Jay Tessmer	Relief Pitcher	1	0	0	7	0	0	0	3.12	8.2	4	1	3	3	0	1	0	4	6
David Wells	Starting Pitcher	18	4	0	30	30	8	5	3.49	214.1	195	29	86	83	1	2	0	29	163



熟悉数据库理论的读者可能会将 XML 模型看作为分支型的数据库,因而也就认为与分支数据库具有同样的缺点(和少数优点)。许多时候以表为基础的关系型方法更有实际意义。在本例中,也属于有实际意义的情况。但是,XML 并不遵循关系模型。

## 4.2 数据的 XML 化

让我们用 XML 处理 1998 年的 Major League 赛季数据的标记开始。请记住，在 XML 内，允许我们创建标记。我们已经决定，文档的根元素是赛季（season）。赛季包括联赛（leagues），而联赛包括分部（divisions），分部又包括球队（teams），球队包括队员（players）。队员的统计数字包括参加的场数（games played）、击球次数（at bats）、得分数（runs）、击中数（hits）、双垒（doubles）、三垒（triples）、全垒得分（home runs）、击球得分（runs batted in）、走步数（walks）和被投手击中数（hits by pitch）。

### 4.2.1 开始编写文档：XML 声明和根元素

XML 文档可由 XML 声明加以识别。这是放在所有 XML 文档的开头的一条处理指令，标识正在使用的 XML 版本。当前可理解的唯一版本号是 1.0。

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

每个合格的 XML 文档（所谓合格有特定的意义，这将在下一章中加以讨论）必须有一个根元素。这是一个完全包括文档中其他所有元素的元素。根元素的起始标记要放在所有其他元素的起始标记之前，而根元素的结束标记要放在所有其他元素的结束标记之后。对于我们的根元素 SEASON，其起始标记是<SEASON>，而结束标记是</SEASON>。文档现在看起来像下面的样子：

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

```
<SEASON>
```

```
</SEASON>
```

XML 声明既不是元素也不是标记。它是处理指令。因而不需要将声明放在根元素 SEASON 之内。但是，我们在文档中放入的每个元素都得放在起始标记<SEASON>和结束标记</SEASON>之间。

根元素的这种选择方法说明我们已经不能在一个文件中保存多个赛季的数据了。如果想要保存多个赛季的数据的话，可以定义一个新的包括赛季（seasons）的根元素，例如，

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

```
<DOCUMENT>
```

```
<SEASON>
```

```
</SEASON>
```

```
<SEASON>
```

```
</SEASON>
```

```
</DOCUMENT>
```

### 命名约定

在开始之前，我还要说几句关于命名约定的话。正如我们在下一章中所见到的，XML 的元素名是比较灵活的，可以包括任意数目的字母和数字，既可是大写的也可是小写的。可以将 XML 标记写成下面的任何样子：

<SEASON>

<Season>

<season>

<season1998>

<Season98>

<season\_98>

这就会有成千上万种可能的变化。全使用大写、全使用小写或是混合大小写都是可以的。但是，我推荐使用一种约定，并坚持下去。

当然，我们对所谈到的赛季加以标识。为达此目的，可为 SEASON 元素定义一个名为 YEAR 的子元素。例如：

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

```
<SEASON>
```

```
<YEAR>
```

```
1998
```

```
</YEAR>
```

```
</SEASON>
```

我在此处以及其他例子中使用了缩进，以便指明元素 YEAR 是元素 SEASON 的子元素，而文本 1998 是元素 YEAR 的内容。这是一种很好的编程习惯，但这不是必须的。XML 中的空白没有特殊的意义。同样的例子也可写成下面的样子：

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

```
<SEASON>
```

```
<YEAR>1998</YEAR>
```

```
</SEASON>
```

确实，我经常将元素压缩到一行上（当一行上可以放得下，而空间又比较紧张时）。还可以将文档再加以压缩，即使压缩成一行也可以，但这要失去可读性。例如：

```
<?xml version="1.0"?><SEASON><YEAR>1998</YEAR></SEASON>
```

当然这样的文档是比较难以阅读和理解的，这也就是为什么我没有这样书写的原因。XML 1.0 规范中的第十条目的中写道：“Terseness in XML markup is of minimal importance.” 翻译成中文是，“XML 标记中的简捷性是不太重要的。” 棒球示例完全反映出了这个目的。

#### 4.2.2 联赛 (League)、(分部) Division 和 (球队) Team 数据的 XML 化



主要棒球联赛分成两个联赛：American League 和 National League。每个联赛都有名称。两个名称可如下编码：

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

<SEASON>

<YEAR>1998</YEAR>

<LEAGUE>

<LEAGUE_NAME>National League</LEAGUE_NAME>

</LEAGUE>

<LEAGUE>

<LEAGUE_NAME>American League</LEAGUE_NAME>

</LEAGUE>

</SEASON>
```

我在这里将联赛的名称定义为元素 LEAGUE\_NAME，而不是简单的 NAME 元素。因为 NAME 太普遍了，而且还打算将其用在其他场合。例如，分部、球队和球员都有名称。



带有相同的名称的不同领域的元素可以利用命名域（namespaces）结合在一起。命名域的问题将在第 18 章中加以讨论。但是，即使使用命名域，也不要将同一领域（如本例中的 TEAM 和 LEAGUE）的多个术语给予同样的名称。

每个联赛可分为东部（east）、西部（west）和中部（central）分部，可编码如下：

```
<LEAGUE>

<LEAGUE_NAME>National League</LEAGUE_NAME>

<DIVISION>

<DIVISION_NAME>East</DIVISION_NAME>

</DIVISION>

<DIVISION>

<DIVISION_NAME>Central</DIVISION_NAME>

</DIVISION>

<DIVISION>

<DIVISION_NAME>West</DIVISION_NAME>

</DIVISION>
```

```
</LEAGUE>

<LEAGUE>

<LEAGUE_NAME>American League</LEAGUE_NAME>

<DIVISION>

<DIVISION_NAME>East</DIVISION_NAME>

</DIVISION>

<DIVISION>

<DIVISION_NAME>Central</DIVISION_NAME>

</DIVISION>

<DIVISION>

<DIVISION_NAME>West</DIVISION_NAME>

</DIVISION>

</LEAGUE>
```

元素的实际值依赖于包括该元素的父元素。American League 和 National League 都有 East 分部，但是这不是一回事。

每个分部又分为多个球队。每个球队都有一个队名和城市名。例如，与 American League 联赛 East 分部有关的名称可编码如下：

```
<DIVISION>

<DIVISION_NAME>East</DIVISION_NAME>

<TEAM>

<TEAM_CITY>Baltimore</TEAM_CITY>

<TEAM_NAME>Orioles</TEAM_NAME>

</TEAM>

<TEAM>

<TEAM_CITY>Boston</TEAM_CITY>

<TEAM_NAME>Red Sox</TEAM_NAME>

</TEAM>

<TEAM>
```

```

<TEAM_CITY>New York</TEAM_CITY>

<TEAM_NAME>Yankees</TEAM_NAME>

</TEAM>

<TEAM>

<TEAM_CITY>Tampa Bay</TEAM_CITY>

<TEAM_NAME>Devil Rays</TEAM_NAME>

</TEAM>

<TEAM>

<TEAM_CITY>Toronto</TEAM_CITY>

<TEAM_NAME>Blue Jays</TEAM_NAME>

</TEAM>

</DIVISION>

```

#### 4.2.3 球员数据的 XML 化

每个球队是由球员组成的。每个球员都有姓和名。将姓和名分开是重要的，这样一来既可以根据名来分类也可以根据姓来分类。1998 年 Yankees 阵容第一个出场的投球手的数据可编码如下：

```

<TEAM>

<TEAM_CITY>New York</TEAM_CITY>

<TEAM_NAME>Yankees</TEAM_NAME>

<PLAYER>

<GIVEN_NAME>Orlando</GIVEN_NAME>

<SURNAME>Hernandez</SURNAME>

</PLAYER>

<PLAYER>

<GIVEN_NAME>David</GIVEN_NAME>

<SURNAME>Cone</SURNAME>

</PLAYER>

<PLAYER>

```

```
<GIVEN_NAME>David</GIVEN_NAME>
```

```
<SURNAME>Wells</SURNAME>
```

```
</PLAYER>
```

```
<PLAYER>
```

```
<GIVEN_NAME>Andy</GIVEN_NAME>
```

```
<SURNAME>Pettitte</SURNAME>
```

```
</PLAYER>
```

```
<PLAYER>
```

```
<GIVEN_NAME>Hideki</GIVEN_NAME>
```

```
<SURNAME>Irabu</SURNAME>
```

```
</PLAYER>
```

```
</TEAM>
```



为了更明显起见，使用标记<GIVEN\_NAME>和<SURNAME>比使用<FIRST\_NAME> 和<LAST\_NAME>或者<FIRST\_NAME>和<FAMILY\_NAME>更好一些。由于不同国家的文化背景不同，可能名（given name）在先也可能姓（family name）在先。同时所有的文化背景下，别号（surnames）不一定是姓（family names）。

#### 4.2.4 球员统计数据的 XML 化

以下几个步骤提供了每个球员的统计数据。统计数据看起来对于投球手和击球手并没有一点不同，特别是对于 American League 联赛，这里没有几个投球员击过球。下面是 Joe Girardi 在 1998 年的统计数据。他是一个接球手，因而我们使用击球的统计数据：

```
<PLAYER>
```

```
<GIVEN_NAME>Joe </GIVEN_NAME>
```

```
<SURNAME>Girard </SURNAME>
```

```
<POSITION>Catcher</POSITION>
```

```
<GAMES>78</GAMES>
```

```
<GAMES_STARTED>76</GAMES_STARTED>
```

```
<AT_BATS>254</AT_BATS>
```

```
<RUNS>31</RUNS>
```

```
<HITS>70</HITS>
```

```
<DOUBLES>11</DOUBLES>

<TRIPLES>4</TRIPLES>

<HOME_RUNS>3</HOME_RUNS>

<RBI>31</RBI>

<STEALS>2</STEALS>

<CAUGHT_STEALING>4</CAUGHT_STEALING>

<SACRIFICE_HITS>8</SACRIFICE_HITS>

<SACRIFICE_FLIES>1</SACRIFICE_FLIES>

<ERRORS>3</ERRORS>

<WALKS>14</WALKS>

<STRUCK_OUT>38</STRUCK_OUT>

<HIT_BY_PITCH>2</HIT_BY_PITCH>

</PLAYER>
```

现在让我们看一下一个投球手的统计数据。虽然投球手在 American League 中很少击球，但在 National League 中却常常击球，到目前为止，投球手击球的次数还是比其他球员少。根据投球手的投球表现，雇用或解雇、表扬或批评。如果投球手偶尔击中一球，则会得到额外的奖励。投球的统计包括比赛场数 (games played)、得胜场数 (wins)、失败场数 (losses)、投球局数 (innings pitched)、得分 (earned runs)、成功防守次数 (shutouts)、击中数 (hits against)、走步放弃 (walks given up) 和其他数据。下面是 Hideki Irabu 1998 年统计数据的 XML 编码：

```
<PLAYER>

<GIVEN_NAME>Hideki</GIVEN_NAME>

<SURNAME>Irabu</SURNAME>

<POSITION>Start ing P tcher</POSITION>

<WINS>13</WINS>

<LOSSES>9</LOSSES>

<SAVES>0</SAVES>

<GAMES>29</GAMES>

<GAMES_STARTED>28</GAMES_STARTED>

<COMPLETE_GAMES>2</COMPLETE_GAMES>
```

```

<SHUT_OUTS> </SHUT_OUTS>

<ERA>4.06</ERA>

<INNINGS> 73</INNINGS>

<HOME_RUNS>148</HOME_RUNS>

<RUNS>27</RUNS>

<EARNED_RUNS>79</EARNED_RUNS>

<HIT_BATTER>78</HIT_BATTER>

<WILD_PITCHES>9</WILD_PITCHES>

<BALK>6</BALK>

<WALKED_BATTER>1</WALKED_BATTER>

<STRUCK_OUT_BATTER>76</STRUCK_OUT_BATTER>

</PLAYER>

```

### XML 标记的简洁性不是太重要

从整个示例来看，我已经遵循了 XML 的明显的原则：“Terseness in XML markup is of minimal importance.”（XML 标记的简洁性不是太重要的。）这当然对非棒球文化下的读者很有帮助。这些读者对于棒球术语及其简写不是很熟悉。比如无法了解为什么 walk 的简写是 BB(base on balls)而不是人们以为的 W。如果文档的大小是个问题的话，将文档用 Zip 一类的工具进行压缩还是很容易的。

但是，这并不意味着 XML 是相当长的，也不意味着手工键入是非常枯燥无味的。我承认，本例强烈地吸引我使用简略语来书写，这样一来，清晰性就丧失殆尽。如果我使用简写，那么典型的 PLAYER 元素可能如下：

```

<PLAYER>

<GIVEN_NAME>Joe</GIVEN_NAME>

<SURNAME>Girard </SURNAME>

<P>C</P>

<G>78</G>

<AB>254</AB>

<R>31</R>

<H>70</H>

<DO>11</DO>

```

<TR>4</TR>

<HR>3</HR>

<RBI>3 </RBI>

<BB>14</BB>

<SO>38</SO>

<SB>2</SB>

<CS>4</CS>

<HBP>2</HBP>

</PLAYER>

#### 4.2.5 将 XML 组装在一起

到目前为止，我向读者展示的只是一段一段（每段一个元素）的 XML 文档。但现在是该将各段组装在一起，看一看包括 1998 年 Major League 赛季的统计数据的全部文档的时候了。清单 4-1 列出了完成了的 XML 文档，其中包括两个联赛、六个分部、三十个队和九个球员。

##### 清单 4-1：一份完整的 XML 文档

<?xml version="1.0"?>

<SEASON>

<YEAR>1998</YEAR>

<LEAGUE>

<LEAGUE\_NAME>National League</LEAGUE\_NAME>

<DIVISION>

<DIVISION\_NAME>East</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Atlanta</TEAM\_CITY>

<TEAM\_NAME>Braves</TEAM\_NAME>

<PLAYER>

<SURNAME>Malloy</SURNAME>

<GIVEN\_NAME>Marty</GIVEN\_NAME>

<POSITION>Second Base</POSITION>

<GAMES>11</GAMES>

<GAMES\_STARTED>8</GAMES\_STARTED>

<AT\_BATS>28</AT\_BATS>

<RUNS>3</RUNS>

<HITS>5</HITS>

<DOUBLES>1</DOUBLES>

<TRIPLES>0</TRIPLES>

<HOME\_RUNS>1</HOME\_RUNS>

<RBI>1</RBI>

<STEALS>0</STEALS>

<CAUGHT\_STEALING>0</CAUGHT\_STEALING>

<SACRIFICE\_HITS>0</SACRIFICE\_HITS>

<SACRIFICE\_FLIES>0</SACRIFICE\_FLIES>

<ERRORS>0</ERRORS>

<WALKS>2</WALKS>

<STRUCK\_OUT>2</STRUCK\_OUT>

<HIT\_BY\_PITCH>0</HIT\_BY\_PITCH>

</PLAYER>

<PLAYER>

<SURNAME>Guillen</SURNAME>

<GIVEN\_NAME>Ozzie </GIVEN\_NAME>

<POSITION>Shortstop</POSITION>

<GAMES>83</GAMES>

<GAMES\_STARTED>59</GAMES\_STARTED>

<AT\_BATS>264</AT\_BATS>



<RUNS>35</RUNS>

<HITS>73</HITS>

<DOUBLES>15</DOUBLES>

<TRIPLES>1</TRIPLES>

<HOME\_RUNS>1</HOME\_RUNS>

<RBI>22</RBI>

<STEALS>1</STEALS>

<CAUGHT\_STEALING>4</CAUGHT\_STEALING>

<SACRIFICE\_HITS>4</SACRIFICE\_HITS>

<SACRIFICE\_FLIES>2</SACRIFICE\_FLIES>

<ERRORS>6</ERRORS>

<WALKS>24</WALKS>

<STRUCK\_OUT>25</STRUCK\_OUT>

<HIT\_BY\_PITCH>1</HIT\_BY\_PITCH>

</PLAYER>

<PLAYER>

<SURNAME>Bautista</SURNAME>

<GIVEN\_NAME>Danny</GIVEN\_NAME>

<POSITION>Outfield</POSITION>

<GAMES>82</GAMES>

<GAMES\_STARTED>27</GAMES\_STARTED>

<AT\_BATS>144</AT\_BATS>

<RUNS>17</RUNS>

<HITS>36</HITS>

<DOUBLES>1</DOUBLES>

<TRIPLES>0</TRIPLES>

<HOME\_RUNS>3</HOME\_RUNS>

<RBI>17</RBI>

<STEALS>1</STEALS>

<CAUGHT\_STEALING>0</CAUGHT\_STEALING>

<SACRIFICE\_HITS>3</SACRIFICE\_HITS>

<SACRIFICE\_FLIES>2</SACRIFICE\_FLIES>

<ERRORS>2</ERRORS>

<WALKS>7</WALKS>

<STRUCK\_OUT>21</STRUCK\_OUT>

<HIT\_BY\_PITCH>0</HIT\_BY\_PITCH>

</PLAYER>

<PLAYER>

<SURNAME>Williams</SURNAME>

<GIVEN\_NAME>Gerald</GIVEN\_NAME>

<POSITION>Outfield</POSITION>

<GAMES>129</GAMES>

<GAMES\_STARTED>51</GAMES\_STARTED>

<AT\_BATS>266</AT\_BATS>

<RUNS>46</RUNS>

<HITS>81</HITS>

<DOUBLES>18</DOUBLES>

<TRIPLES>3</TRIPLES>

<HOME\_RUNS>10</HOME\_RUNS>

<RBI>44</RBI>

<STEALS>1</STEALS>

<CAUGHT\_STEALING>5</CAUGHT\_STEALING>

<SACRIFICE\_HITS>2</SACRIFICE\_HITS>

<SACRIFICE\_FLIES>1</SACRIFICE\_FLIES>

<ERRORS>5</ERRORS>

<WALKS>17</WALKS>

<STRUCK\_OUT>48</STRUCK\_OUT>

<HIT\_BY\_PITCH>3</HIT\_BY\_PITCH>

</PLAYER>

<PLAYER>

<SURNAME>Glavine</SURNAME>

<GIVEN\_NAME>Tom</GIVEN\_NAME>

<POSITION>Starting Pitcher</POSITION>

<WINS>20</WINS>

<LOSSES>6</LOSSES>

<SAVES>0</SAVES>

<GAMES>33</GAMES>

<GAMES\_STARTED>33</GAMES\_STARTED>

<COMPLETE\_GAMES>4</COMPLETE\_GAMES>

<SHUT\_OUTS>3</SHUT\_OUTS>

<ERA>2.47</ERA>

<INNINGS>229.1</INNINGS>

<HOME\_RUNS>202</HOME\_RUNS>

<RUNS>13</RUNS>

<EARNED\_RUNS>67</EARNED\_RUNS>

<HIT\_BATTER>63</HIT\_BATTER>

<WILD\_PITCHES>2</WILD\_PITCHES>

<BALK>3</BALK>

<WALKED\_BATTER>0</WALKED\_BATTER>

<STRUCK\_OUT\_BATTER>74</STRUCK\_OUT\_BATTER>

</PLAYER>

<PLAYER>

<SURNAME>Lopez</SURNAME>

<GIVEN\_NAME>Javier</GIVEN\_NAME>

<POSITION>Catcher</POSITION>

<GAMES>133</GAMES>

<GAMES\_STARTED>124</GAMES\_STARTED>

<AT\_BATS>489</AT\_BATS>

<RUNS>73</RUNS>

<HITS>139</HITS>

<DOUBLES>21</DOUBLES>

<TRIPLES>1</TRIPLES>

<HOME\_RUNS>34</HOME\_RUNS>

<RBI>106</RBI>

<STEALS>5</STEALS>

<CAUGHT\_STEALING>3</CAUGHT\_STEALING>

<SACRIFICE\_HITS>1</SACRIFICE\_HITS>

<SACRIFICE\_FLIES>8</SACRIFICE\_FLIES>

<ERRORS>5</ERRORS>

<WALKS>30</WALKS>

<STRUCK\_OUT>85</STRUCK\_OUT>

<HIT\_BY\_PITCH>6</HIT\_BY\_PITCH>

</PLAYER>

<PLAYER>

<SURNAME>Klesko</SURNAME>

<GIVEN\_NAME>Ryan</GIVEN\_NAME>

<POSITION>Outfield</POSITION>

<GAMES>129</GAMES>

<GAMES\_STARTED>124</GAMES\_STARTED>

<AT\_BATS>427</AT\_BATS>

<RUNS>69</RUNS>

<HITS>17</HITS>

<DOUBLES>29</DOUBLES>

<TRIPLES>1</TRIPLES>

<HOME\_RUNS>18</HOME\_RUNS>

<RBI>70</RBI>

<STEALS>5</STEALS>

<CAUGHT\_STEALING>3</CAUGHT\_STEALING>

<SACRIFICE\_HITS>0</SACRIFICE\_HITS>

<SACRIFICE\_FLIES>4</SACRIFICE\_FLIES>

<ERRORS>2</ERRORS>

<WALKS>56</WALKS>

<STRUCK\_OUT>66</STRUCK\_OUT>

<HIT\_BY\_PITCH>3</HIT\_BY\_PITCH>

</PLAYER>

<PLAYER>

<SURNAME>Galarrraga</SURNAME>

<GIVEN\_NAME>Andres</GIVEN\_NAME>

<POSITION>First Base</POSITION>

<GAMES>153</GAMES>

<GAMES\_STARTED>151</GAMES\_STARTED>

<AT\_BATS>555</AT\_BATS>

<RUNS>103</RUNS>

<HITS>169</HITS>

<DOUBLES>27</DOUBLES>

<TRIPLES>1</TRIPLES>

<HOME\_RUNS>44</HOME\_RUNS>

<RBI>121</RBI>

<STEALS>7</STEALS>

<CAUGHT\_STEALING>6</CAUGHT\_STEALING>

<SACRIFICE\_HITS>0</SACRIFICE\_HITS>

<SACRIFICE\_FLIES>5</SACRIFICE\_FLIES>

<ERRORS>1</ERRORS>

<WALKS>63</WALKS>

<STRUCK\_OUT>146</STRUCK\_OUT>

<HIT\_BY\_PITCH>25</HIT\_BY\_PITCH>

</PLAYER>

<PLAYER>

<SURNAME>Helms</SURNAME>

<GIVEN\_NAME>Wes</GIVEN\_NAME>

<POSITION>Third Base</POSITION>

<GAMES>7</GAMES>

<GAMES\_STARTED>2</GAMES\_STARTED>

<AT\_BATS>13</AT\_BATS>

<RUNS>2</RUNS>

<HITS>4</HITS>

<DOUBLES>1</DOUBLES>

<TRIPLES>0</TRIPLES>

<HOME\_RUNS>1</HOME\_RUNS>

<RBI>2</RBI>

<STEALS>0</STEALS>

<CAUGHT\_STEALING>0</CAUGHT\_STEALING>

<SACRIFICE\_HITS>0</SACRIFICE\_HITS>

<SACRIFICE\_FLIES>0</SACRIFICE\_FLIES>

<ERRORS>1</ERRORS>

<WALKS>0</WALKS>

<STRUCK\_OUT>4</STRUCK\_OUT>

<HIT\_BY\_PITCH>0</HIT\_BY\_PITCH></PLAYER>

</TEAM>

<TEAM>

<TEAM\_CITY>Florida</TEAM\_CITY>

<TEAM\_NAME>Marlins</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Montreal</TEAM\_CITY>

<TEAM\_NAME>Expos</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>New York</TEAM\_CITY>

<TEAM\_NAME>Mets</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Philadelphia</TEAM\_CITY>

<TEAM\_NAME>Phillies</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>Central</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Chicago</TEAM\_CITY>

<TEAM\_NAME>Cubs</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Cincinnati</TEAM\_CITY>

<TEAM\_NAME>Reds</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Houston</TEAM\_CITY>

<TEAM\_NAME>Astros</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Milwaukee</TEAM\_CITY>

<TEAM\_NAME>Brewers</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Pittsburgh</TEAM\_CITY>

<TEAM\_NAME>Pirates</TEAM\_NAME>

</TEAM>



<TEAM>

<TEAM\_CITY>St. Louis</TEAM\_CITY>

<TEAM\_NAME>Cardinals</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>West</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Arizona</TEAM\_CITY>

<TEAM\_NAME>Diamondbacks</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Colorado</TEAM\_CITY>

<TEAM\_NAME>Rockies</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Los Angeles</TEAM\_CITY>

<TEAM\_NAME>Dodgers</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>San Diego</TEAM\_CITY>

<TEAM\_NAME>Padres</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>San Francisco</TEAM\_CITY>

<TEAM\_NAME>Giants</TEAM\_NAME>

</TEAM>

</DIVISION>

</LEAGUE>

<LEAGUE>

<LEAGUE\_NAME>American League</LEAGUE\_NAME>

<DIVISION>

<DIVISION\_NAME>East</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Baltimore</TEAM\_CITY>

<TEAM\_NAME>Orioles</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Boston</TEAM\_CITY>

<TEAM\_NAME>Red Sox</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>New York</TEAM\_CITY>

<TEAM\_NAME>Yankees</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Tampa Bay</TEAM\_CITY>

<TEAM\_NAME>Devil Rays</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Toronto</TEAM\_CITY>

<TEAM\_NAME>Blue Jays</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>Central</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Chicago</TEAM\_CITY>

<TEAM\_NAME>White Sox</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Kansas City</TEAM\_CITY>

<TEAM\_NAME>Royals</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Detroit</TEAM\_CITY>

<TEAM\_NAME>Tigers</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Cleveland</TEAM\_CITY>

<TEAM\_NAME>Indians</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Minnesota</TEAM\_CITY>

<TEAM\_NAME>Twins</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>West</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Anaheim</TEAM\_CITY>

<TEAM\_NAME>Angels</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Oakland</TEAM\_CITY>

<TEAM\_NAME>Athletics</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Seattle</TEAM\_CITY>

<TEAM\_NAME>Mariners</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Texas</TEAM\_CITY>

<TEAM\_NAME>Rangers</TEAM\_NAME>

</TEAM>

</DIVISION>

</LEAGUE>

</SEASON>

图 4-1 显示的是将本文档装入 Internet Explorer 5.0 的情况。



图 4-1 在 Internet Explorer 5.0 中显示的 1998 年主要联赛的统计数据

即使现在这个文档也是不完全的。此文档只包括一个队的球员（Atlanta Braves 队）而且只有该球队的九个球员。如果将全部都写出来的话，则示例就会变得太长，以至于本书无法将其包括。



在名为 1998statistics.xml 的更为完整的 XML 文档中，包括了 1998 年度两大联赛的所有球员的统计数据，这个文档附在本书光盘中，目录为 examples/base-ball。同时，我故意将所包括的数据加以限制，以便符合本书的篇幅。实际上，可以包括更为详细的数据。我已经间接提到可按比赛场次、投球次数等来安排数据的可能性。即使没有那样做，还是有许多细节可以添加到每个元素中。球队还有教练、经理、老板（说到 Yankees 队怎能不提到 George Steinbrenner 呢？）、室内运动场和其他项目。

我还故意忽略了可以从这里给出的其他项目中计算出来的数字，如平均击球数等。不管如何，球员还有许多其他数据，如臂长、身高、出生日期等。当然球员远不止这里列出的几个。所有这一切都是很容易加进 XML 文档的。但是我们的 XML 化就到此为止了，这样我们才能往下进行，首先要简短地讨论一下为什么这一数据格式是有用的，然后再讨论在 Web 浏览器上实际显示该文档所用的技术。

## 4.3 XML 格式的优点

表 4-1 对于显示一个球队的击球数据是简捷且易于理解的。我们从改写成的简单 4-1 中的长得多的形式中会得到什么好处呢？好处有如下几种：

- 数据是自说明的
- 数据可用标准工具加以处理
- 数据可用标准工具查看
- 用样式单可容易地生成同样数据的不同视图



XML 格式的第一条主要好处是数据是自描述的。每个数字的意义是清楚的，且不会错误地与数字本身相联系。当读取文档时，用户了解<HITS> 2 </HITS>中的 2 指的是击中数而不是得分或是防守。如果键入文档的打字员漏掉了一个数字，不会造成其后的数字都错了位。HITS 就是 HITS，即使它前面的 RUNS 元素丢失也没关系。

在本书第二部分中，读者会看到，XML 还可以使用 DTD 来加强限制，使得某些元素，如 HITS 或 RUNS 必须存在。

第二条好处是 XML 提供的数据可用广泛的具有 XML 处理能力的工具加以处理，从相当贵的软件，如 Adobe FrameMaker 到免费软件，如 Python 和 Perl。数据量可以很大，但是数据额外的冗余就允许使用更多的工具来处理它。

当查看数据时，也同样有这样的問題。XML 文档可装入 Internet Explorer 5.0、Mozilla、FrameMaker 5.5.6 和许多其他工具，所有这些工具都提供唯一的、有用的一种数据的视图。数据还可以装入简单的文本编辑器中，如 vi、BBEdit 和 TextPad。这就使得数据或多或少的可在多种平台上查看。

使用新软件也不是获得数据的不同视图的唯一方法。在下一节中，我们将为棒球统计数据创建一个样式单，来提供一种与图 4-1 完全不同的查看数据的方法。每当对同一文档施加不同的样式单，都可以看到不同图景。

最后，要向自己发问，文件大小真是很成问题吗？当前硬盘容量已经相当大了，可以存入大量数据，即使存储得不太节省也没有太大的关系。同时，XML 文件的压缩率很大。全部的两大棒球联赛 1998 年统计数据的文档是 653K。如果用 gzip 压缩一下的话，只有 66K，几乎压缩了 90%。先进的 HTTP 服务器，如 Jigsaw 可以发送压缩文件，而不必解压缩，因而文档所用的网络带宽与其实际信息内容已相当接近。最后，我们不能认为二进制文件格式（特别通用的格式）必定是高效的。包含 1998statistics.xml 文件同样数据的 Microsoft Excel 文件的大小达到了 2.37MB，比 XML 格式大了三倍多。虽然我们能够创建更为有效的文件格式和编码方法，但实际上简单并不是必须的。

## 4.4 编制样式单以便显示文档

图 4-1 中的 XML 文档的原始视图对于某些应用来说也是不错的。例如，此视图允许折叠和展开单个的元素，因而可以只看文档中要看的部分。但大多数时候，人们总希望看到更好的形式，特别是，想要在 Web 上显示数据时。为了提供更好的外观，必须为文档编写样式单。

在本章中，我们使用的是 CSS 样式单。CSS 样式单将特定的格式化信息与文档中的每个元素联系起来。我们的 XML 文档中使用的元素的完全列表如下：

*SEASON*

*YEAR*

*LEAGUE*

*LEAGUE\_NAME*

*DIVISION*

*DIVISION\_NAME*

*TEAM*

*TEAM\_CITY*

*TEAM\_NAME*

*PLAYER*

*SURNAME*

*GIVEN\_NAME*

*POSITION*

*GAMES*

*GAMES\_STARTED*

*AT\_BATS*

*RUNS*

*HITS*

*DOUBLES*

*TRIPLES*

*HOME\_RUNS*

*RBI**STEALS**CAUGHT\_STEALING**SACRIFICE\_HITS**SACRIFICE\_FLIES**ERRORS**WALKS**STRUCK\_OUT**HIT\_BY\_PITCH*

一般来说，我们要用重复的过程来为每个元素增加样式规则，一次一个元素地进行，然后检查是否达到了要求，再处理下一个元素。在本例中，这种办法对于不熟悉样式单属性的人来说也有好处。

#### 4.4.1 与样式单连接

样式单的名称可随便取。如果只是为一个文档编制样式单，那么习惯上样式单的文件与文档的文件名一样，但是三字母的扩展名是.css而不是.xml。例如，对于XML文档1998shortstats.xml来说，样式单文件可以叫做1998shortstats.css。另一方面，如果同样的样式单还要用于许多文档，那么，可能需要更为普通的文件名，如baseballstats.css。

由于CSS样式单是级联的，同一文档可不止一个样式单。因而baseballstats.css可向文档施加某些一般的样式规则，而1998shortstats.css可覆盖其中的几条规则，以便在同一文档（1998shortstats.xml）中处理特定的细节。我们将第12章“级联样式单（级别1）”中讨论这一问题。

为了将样式单与文档联系起来，只要像下面所示简单地在XML声明和根元素间增加一个<?xml-stylesheet?>处理指令就可以了：

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

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

<SEASON>

...
```

这条指令告诉浏览器读取文档并施加保存在文件baseballstats.css中的样式单。这个文件是假设放在与XML文件同一服务器上的同一目录中的。换句话说，baseballstats.css是个相对的URL。完全的URL也是可以使用的。例如：

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

<?xml-stylesheet type="text/css"

href="http://metalab.unc.edu/xml/examples/baseballstats.css"?>
```



<SEASON>

...

开始时，用户可以简单地将一个名为 baseballstats.css 的空文件放在与 XML 文档相同的目录中。然后向 1998shortstats.xml（清单 4-1）中增加适当的指令，该文档现在在浏览器中的外观如图 4-2 所示。只显示了元素内容。可折叠的大纲视图（图 4-1）不见了。元素内容的格式使用的是浏览器的缺省格式，在本例中是黑色 12 磅的 Times Roman 字体放在白色背景上。

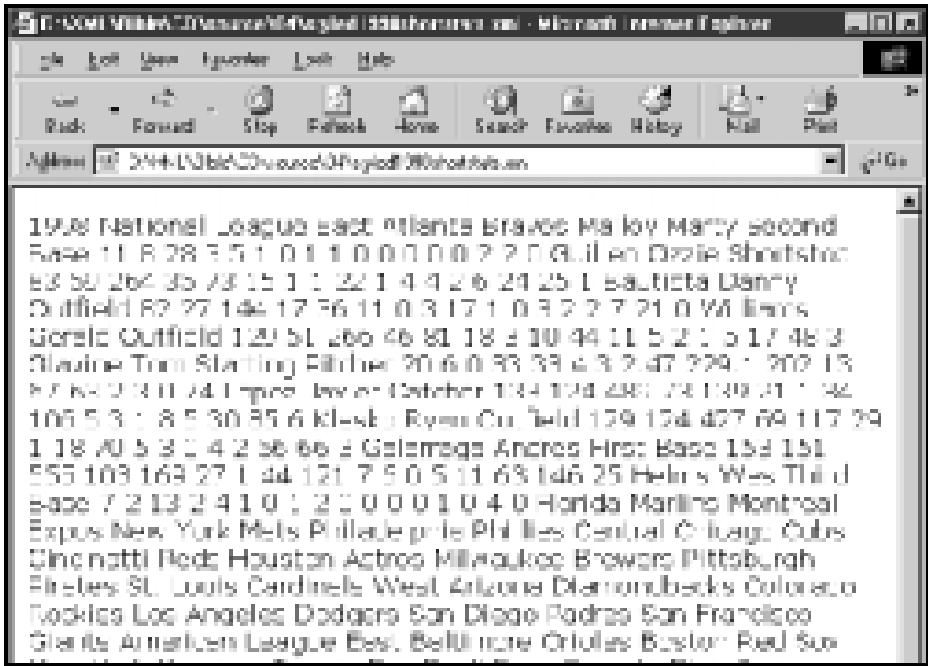


图 4-2 使用了空白样式之后的 1998 年两大棒球联赛的统计数字显示



如果在指定位置找不到样式单处理指令（xml-stylesheet）中指定的样式单文件名，也可看到一个很像图 4-2 的视图。

4.4.2 为根元素指定样式规则

用户不必为每个元素指定样式规则。许多元素允许将其父元素的样式串接下来。因而最重要的样式是根元素的样式，在本例中就是 SEASON 元素。这个样式定义了页面上所有其他元素的缺省样式。大致为 72 dpi 的分辨率的计算机显示器不如纸上 300dpi 或更大的分辨率那样高。所以，Web 页面通常应该使用较大磅数的字号。首先将缺省样式定义为白色背景上的 14 磅黑色字，定义如下：

```
SEASON {font-size: 14pt; background-color: white;
color: black; display: block}
```

将这条语句放在一个文本文件中，将其以文件名 baseballstats.css 与清单 4-1 中的文件（1998shortstats.xml）保存在同一目录中。在浏览器中打开 1998shortstats.xml。我们就会看到如图 4-3 所示的情况。

在图 4-2 和图 4-3 之间字号发生了变化，但文本颜色和背景颜色没有变化。其实这没有必要加以设置，因为黑色文本和白色背景是缺省的。但明确地加以设置也没有损失什么。

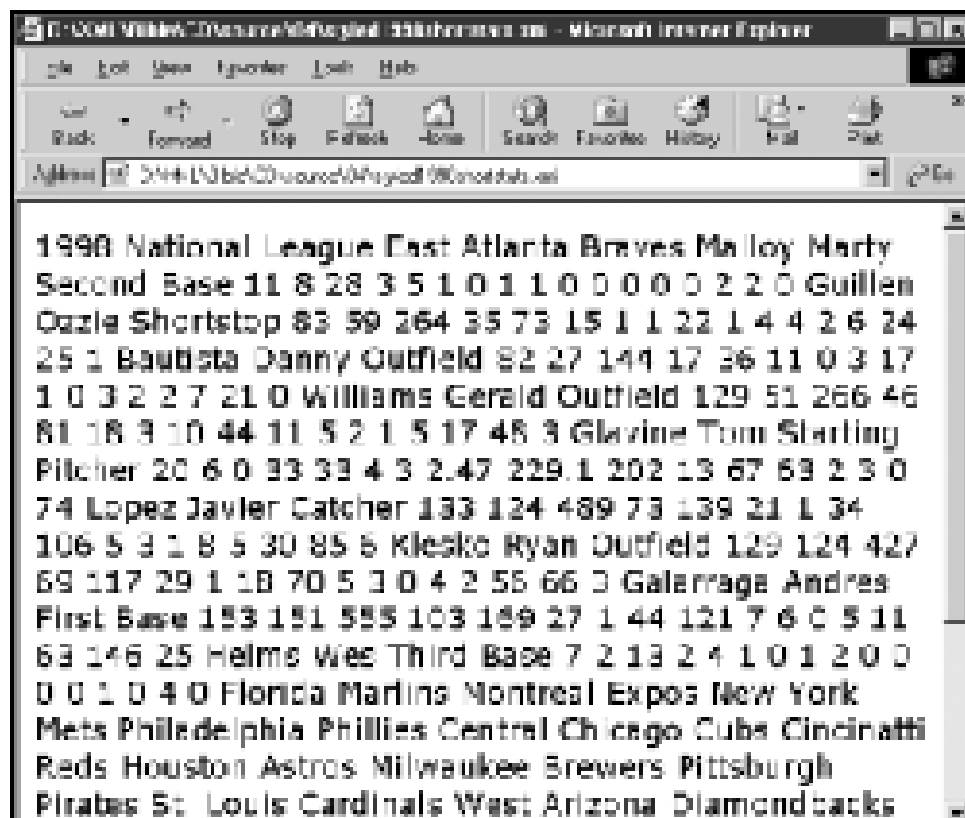


图 4-3 以 14 磅白地黑字显示的棒球统计数据

#### 4.4.3 为标题指定样式规则

元素 YEAR 或多或少可算是文档的标题。因而使其显示得大一些，用 32 磅的字号也就足够大了。同时，它还应该从文档的其余部分突出出来，而不是简单地与其他内容混在一起。利用下面的样式规则可以达到这些目的：

```
YEAR {display: block; font-size: 32pt; font-weight: bold;
text-align: center}
```

图 4-4 显示的是将此规则增加到样式单中之后的文档。请特别注意，在“1998”后面的换行。有这个换行是由于 YEAR 是块级元素。而在文档中的其他元素都是内联元素。我们只能使块级元素居中（或左对齐、右对齐或两端对齐）。



图 4-4 将 YEAR 元素格式化为标题

在使用了这种样式单的文档中，YEAR 元素与 HTML 中的 H1 标题元素的功能重复了。由于这个文档是非常整齐地分支结构，几个其他元素的功能与 HTML 中的 H2、H3 等相似。这些元素都可以用相似的规则加以格式化，只是将字号略微减小一些罢了。

例如，SEASON 由两个 LEAGUE 元素组成。每个 LEAGUE 的名称，即 LEAGUE\_NAME 元素，起了 HTML 中的 H2 元素一样的作用。每个 LEAGUE 元素又由三个 DIVISION 元素所组成。每个 DIVISION 的名称，也就是 DIVISION\_NAME 元素，具有 HTML 中的 H3 元素的作用。这两条规则分别将这两种元素加以格式化：

```
LEAGUE_NAME {display: block; text-align: center; font-size:
28pt; font-weight: bold}
```

```
DIVISION_NAME {display: block; text-align: center; font-size:
24pt; font-weight: bold}
```

图 4-5 显示的是最后的文档。

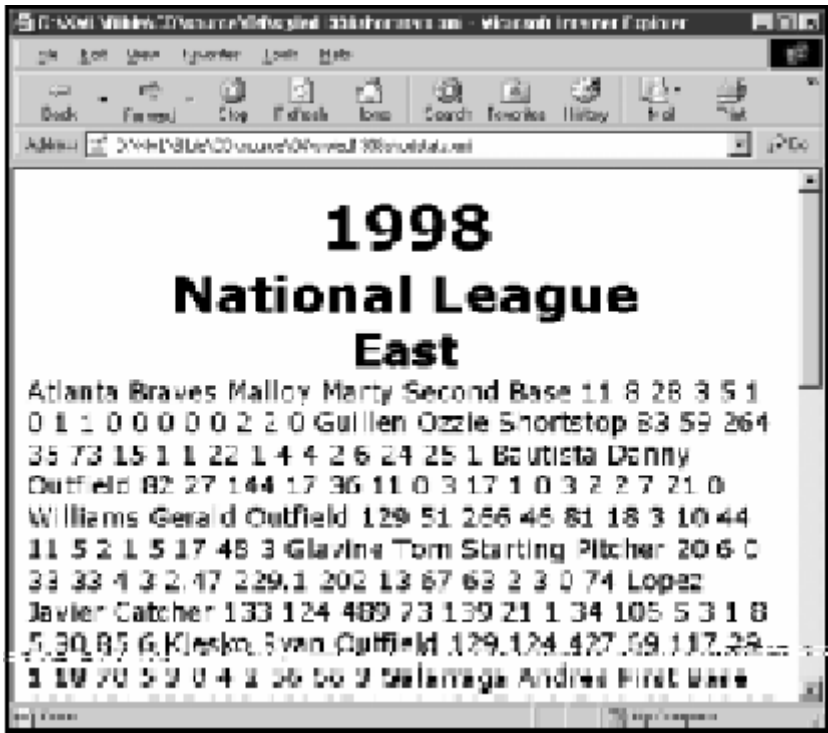


图 4-5 将 LEAGUE\_NAME 和 DIVISION\_NAME 元素格式化为下级标题



HTML 和 XML 的一个重要区别是，在 HTML 中通常不会出现在一个元素中既包括节标题（H2、H3、H4 等），又包括该节的完整内容的情况。节的内容必须包括在一级标题的结束和下一个同级标题的开始之间。这对于必须分析 HTML 文档的语法的软件来说是非常重要的，例如，要自动生成目录时。

Divisions 又分成为 TEAM 元素。要将此格式化需要一些技巧，因为球队的标题并不就是 TEAM\_NAME 元素，而是 TEAM\_CITY 元素与 TEAM\_NAME 拼接在一起的。所以这需要的是内联元素而不是单独的块级元素。然而，它们仍然是标题，因而我们将其设置为粗斜体的 20 磅字体。图 4-6 显示的是将这两条规则加到样式单中的结果。

```
TEAM_CITY {font-size: 20pt; font-weight: bold;
font-style: italic}

TEAM_NAME {font-size: 20pt; font-weight: bold;
font-style: italic}
```

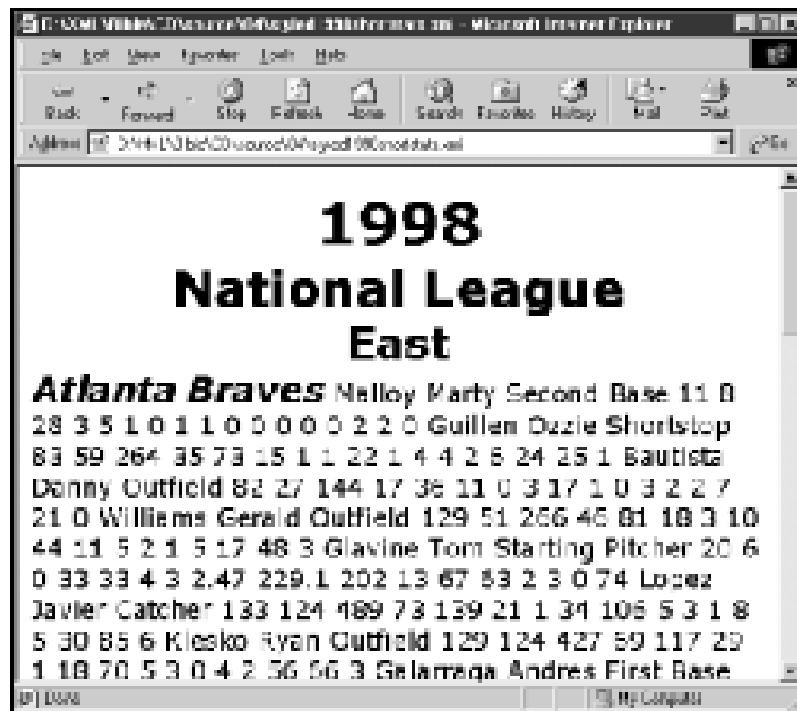


图 4-6 为队名设置样式

到此为止，将队名与城市名作为结合起来的块级元素来排列结果可能会是不错的。有几种办法可达到这个目的。例如，可以向 XML 文档中增加一个附加的 TEAM\_TITLE 元素，其目的只是为了包括 TEAM\_NAME 和 TEAM\_CITY。例如：

```
<TEAM>

<TEAM_TITLE>

<TEAM_CITY>Colorado</TEAM_CITY>

<TEAM_NAME>Rockies</TEAM_NAME>

</TEAM_TITLE>

</TEAM>
```

接着，可以增加一条向 TEAM\_TITLE 施加块级格式化的样式规则：

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

但是，绝不应该为了使样式单简单一些而重新排列 XML 文档。毕竟，样式单的总的目的是将格式化信息保存于文档之外。不过，用户可以通过别的办法达到同样的效果。其办法是，使紧挨着的上一个和下一个元素变成块级元素，也就是说，将 TEAM 和 PLAYER 变成块级元素。这就将 TEAM\_NAME 和 TEAM\_CITY 放在了由它们本身组成的隐式块级元素之中了。图 4-7 显示了其结果。

```
TEAM {display: block}

PLAYER {display: block}
```

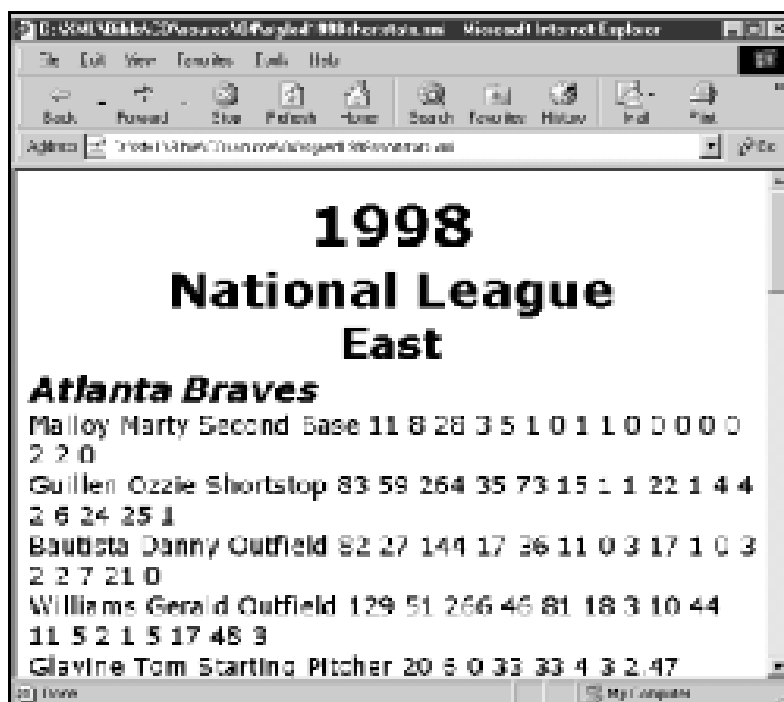


图 4-7 作为段标题而格式化的队名和城市名

#### 4.4.4 为球员和统计元素指定样式规则

本文档需要的最具技巧的格式化是对每个球员及其统计数据的格式化。每个队有几十个球员。每个球员都有统计数据。应该将 TEAM 元素看作是由 PLAYER 元素组成的，且将每个球员放在他自己的块级节中，正如前一个元素所做的那样。不过，排列这些数据的更为吸引人且更为有效的方法是使用表格。达到这一目的的样式规则如下所示：

```
TEAM {display: table}
```

```
TEAM_CITY {display: table-caption}
```

```
TEAM_NAME {display: table-caption}
```

```
PLAYER {display: table-row}
```

```
SURNAME {display: table-cell}
```

```
GIVEN_NAME {display: table-cell}
```

```
POSITION {display: table-cell}
```

```
GAMES {display: table-cell}
```

```
GAMES_STARTED {display: table-cell}
```

```
AT_BATS {display: table-cell}
```

```
RUNS {display: table-cell}
```

```
HITS {display: table-cell}
```

```

DOUBLES {display: table-cell}

TRIPLES {display: table-cell}

HOME_RUNS {display: table-cell}

RBI {display: table-cell}

STEALS {display: table-cell}

CAUGHT_STEALING {display: table-cell}

SACRIFICE_HITS {display: table-cell}

SACRIFICE_FLIES {display: table-cell}

ERRORS {display: table-cell}

WALKS {display: table-cell}

STRUCK_OUT {display: table-cell}

HIT_BY_PITCH {display: table-cell}

```

遗憾的是，只有 CSS2 才支持表格属性，而 Internet Explorer 5.0 和其他写作本书时已存在的浏览器还不支持 CSS2。由于还不能使用表格的格式化方法，我们只好使 TEAM 和 PLAYER 成为块级元素，而让其他数据保持缺省格式。

#### 4.4.5 本节小结

清单 4-2 列出了完成后的样式单。CSS 样式单除了一条一条的规则之外，这种样式单没有什么结构。实际上，样式单只是我在上面分别介绍过的所有规则的列表。列表中的顺序不是很重要，只要每条规则都包含进去也就可以了。

##### 清单 4-2: baseballstats.css

```

SEASON {font-size: 4pt; background-color: white;

color: black; display: block}

YEAR {display: block; font-size: 32pt; font-weight: bold;

text-align: center}

LEAGUE_NAME {display: block; text-align: center;

font-size: 28pt; font-weight: bold}

DIVISION_NAME {display: block; text-align: center;

font-size: 24pt; font-weight: bold}

TEAM_CITY {font-size: 20pt; font-weight: bold;

```

```
font-style: italic}
```

```
TEAM_NAME {font-size: 20pt; font-weight: bold;
```

```
font-style: italic}
```

```
TEAM {display: block}
```

```
PLAYER {display: block}
```

到此就完成了棒球统计数据的基本格式化的任务。不过很清楚，还有许多工作要做。支持真正表格格式化的浏览器将会大有帮助。然而还有其他工作。下面指出这些工作，其顺序没有什么关系：

- 只是列出了原始的数字，而没有说明数字代表了什么。每个数字应该有一个为其命名的标题，如“RBI”或是“At Bats”。
- 令人感兴趣的数据，如平均击球数由于是可能从这里列出的数据中计算出来的，这样的数据就没有包括进来。
- 某些标题有点太短。如果文档的标题是“1998 Major League Baseball”而不是简单的“1998”可能会更好。
- 如果 Major League 中的所有球员都包括进来，这一文档就会如此之长，以至于难以阅读。在这种情况下，与 Internet Explorer 中的可折叠的大纲视图类似的东西可能会更有用。
- 由于投手统计数据与击球手的数据是如此不同，在花名册中分别排序可能会更好。

像这样一类的许多看法都应该向文档中增加更多的内容加以体现。例如，为了将标题从“1998”改为“1998 Major League Baseball”，所要做的工作只是将 YEAR 元素改写如下：

```
1998 Major League Baseball
```

在每个花名册的顶部，用一个假想的球员名，为球员的统计数据加进小标题，如下所示：

```
<PLAYER>
```

```
<SURNAME>Surname</SURNAME>
```

```
<GIVEN_NAME>Given name</GIVEN_NAME>
```

```
<POSITION>Position</POSITION>
```

```
<GAMES>Games</GAMES>
```

```
<GAMES_STARTED>Games Started</GAMES_STARTED>
```

```
<AT_BATS>At Bats</AT_BATS>
```

```
<RUNS>Runs</RUNS>
```

```
<HITS>Hits</HITS>
```

```
<DOUBLES>Doubles</DOUBLES>
```

```
<TRIPLES>Triples</TRIPLES>
```

```
<HOME_RUNS>Home Runs</HOME_RUNS>
```



<RBI>Runs Batted In</RBI>

<STEALS>Steals</STEALS>

<CAUGHT\_STEALING>Caught Stealing</CAUGHT\_STEALING>

<SACRIFICE\_HITS>Sacrifice Hits</SACRIFICE\_HITS>

<SACRIFICE\_FLIES>Sacrifice Flies</SACRIFICE\_FLIES>

<ERRORS>Errors</ERRORS>

<WALKS>Walks</WALKS>

<STRUCK\_OUT>Struck Out</STRUCK\_OUT>

<HIT\_BY\_PITCH>Hit By Pitch</HIT\_BY\_PITCH>

</PLAYER>

关于这种方法还有一些基本问题需要解决。年份是 1998 年，而不是 1998 Major League Baseball 。小标题 “At Bats” 与击球数不是一回事。（这正是事物的名称与事物本身之间的差别。）这时可增加一些标记如下（加以解决）：

<TABLE\_HEAD>

<COLUMN\_LABEL>Surname</COLUMN\_LABEL>

<COLUMN\_LABEL>Given name</COLUMN\_LABEL>

<COLUMN\_LABEL>Position</COLUMN\_LABEL>

<COLUMN\_LABEL>Games</COLUMN\_LABEL>

<COLUMN\_LABEL>Games Started</COLUMN\_LABEL>

<COLUMN\_LABEL>At Bats</COLUMN\_LABEL>

<COLUMN\_LABEL>Runs</COLUMN\_LABEL>

<COLUMN\_LABEL>Hits</COLUMN\_LABEL>

<COLUMN\_LABEL>Doubles</COLUMN\_LABEL>

<COLUMN\_LABEL>Triples</COLUMN\_LABEL>

<COLUMN\_LABEL>Home Runs</COLUMN\_LABEL>

<COLUMN\_LABEL>Runs Batted In</COLUMN\_LABEL>

<COLUMN\_LABEL>Steals</COLUMN\_LABEL>

<COLUMN\_LABEL>Caught Stealing</COLUMN\_LABEL>

```

<COLUMN_LABEL>Sacrifice Hits</COLUMN_LABEL>

<COLUMN_LABEL>Sacrifice Flies</COLUMN_LABEL>

<COLUMN_LABEL>Errors</COLUMN_LABEL>

<COLUMN_LABEL>Walks</COLUMN_LABEL>

<COLUMN_LABEL>Struck Out</COLUMN_LABEL>

<COLUMN_LABEL>Hit By Pitch</COLUMN_LABEL>

</TABLE_HEAD>

```

不过这样一来，基本上是重新“发明”了HTML，而且使我们又回到了使用标记来格式化而不是用于意义了。同时，我们还重复了已经包括在元素名称中的信息。整个文档还相当大，我们还是希望文档不要太大为好。

增加击球和其他的平均数并不复杂。只要将数据作为附加的元素包括进来就可以了。例如，下面是一个带有该种数据的球员：

```

<PLAYER>

<SURNAME>Malloy</SURNAME>

<GIVEN_NAME>Marty</GIVEN_NAME>

<POSITION>Second Base</POSITION>

<GAMES>1</GAMES>

<GAMES_STARTED>8</GAMES_STARTED>

<ON_BASE_AVERAGE>.233</ON_BASE_AVERAGE>

<SLUGGING_AVERAGE>.321</SLUGGING_AVERAGE>

<BATTING_AVERAGE>.179</BATTING_AVERAGE>

<AT_BATS>28</AT_BATS>

<RUNS>3</RUNS>

<HITS>5</HITS>

<DOUBLES>1</DOUBLES>

<TRIPLES>0</TRIPLES>

<HOME_RUNS>1</HOME_RUNS>

<RBI>1</RBI>

```

```
<STEALS>0</STEALS>
```

```
<CAUGHT_STEALING>0</CAUGHT_STEALING>
```

```
<SACRIFICE_HITS>0</SACRIFICE_HITS>
```

```
<SACRIFICE_FLIES>0</SACRIFICE_FLIES>
```

```
<ERRORS>0</ERRORS>
```

```
<WALKS>2</WALKS>
```

```
<STRUCK_OUT>2</STRUCK_OUT>
```

```
<HIT_BY_PITCH>0</HIT_BY_PITCH>
```

```
</PLAYER>
```

但是，这种信息是多余的，因为这些数据可从已经包括进来的数据中计算出来。例如，平均击球数是击中的垒数被击球数除的结果，也就是 HITS/AT\_BAT。多余数据使得维护和更新数据变得非常困难。对一个元素的简单的改变或是增加都会引起多个位置的改变和重新计算。

真正所需要的是一种不同的样式单语言，能使我们向元素中增加样板内容并根据现存的元素内容执行转换。这样的语言是存在的，这就是可扩展的样式语言（Extensible Style Language，简称为 XSL）。



可扩展的样式语言（Extensible Style Language，XSL）将在第 14、15 章中加以讨论。

CSS 比 XSL 简单，对于基本的 Web 页面来说，也更适合一些，而且也是更为直接的文档。XSL 变得相当复杂，但功能也更为强大。XSL 是建立在我们已经在上面学到的简单的 CSS 格式化的基础之上的，但是也提供了将源文档转换为读者可以查看的不同形式的方法。在调试 XML 时，首先使用 CSS 寻找问题，然后再转到 XSL，以便获得更大的灵活性，这通常是不错的主意。

## 4.5 本章小结

在本章中，读者看到了几个展示如何从头创建 XML 文档的示例。我们特别学到了如下内容：

- 如何检查包括在 XML 文件中的数据，以便标识元素。
- 如何用自己定义的 XML 标记来标记数据。
- XML 格式所提供的比传统格式的优越性。
- 如何编写样式单，使文档格式化并显示出来。

本章中充满了枯燥的代码。文档是在没有太多的细节的情况下编写出来的。在下一章中，我们将要探讨在 XML 文档中嵌入信息的附加意义，包括特性、注释和处理指令，并看一看在 XML 中用另一种对棒球统计数据编码的方法。

## 第 5 章 属性、空标记和 XSL

使用 XML 对一组给定的数据进行编码,有很多种方法。但是没有哪一种方法是唯一正确的,只是一些方法相较而言更可取,在特定的应用中更合适。本章采用前面章节中所用的棒球示例,仔细探讨使用 XML 创建棒球统计的不同方法。文中会特别强调使用属性存储信息和使用空标记定义元素位置。另外,鉴于 CSS(级联样式单)对缺乏内容的 XML 元素执行起来并不顺利,我们将检验另一种功能更强大的样式单语言——XSL。

本章内容包括:

- 属性
- 属性与元素的对比
- 空标记
- XSL

### 5.1 属性

在上一章中,所有的数据可分为标记名或者元素的内容两类。这种方法直接易懂,但不是唯一的。XML 元素与 HTML 中的元素一样,有自己的属性。元素的每个属性是一个名称-数值对,名称和数值分别为一个字符串,一个元素不能有两个同名的属性。

大家都熟悉 HTML 的属性句法,请看下面的<IMG>标记实例:

```
<IMG SRC=cup.gif WIDTH=89 HEIGHT=67 ALT="Cup of coffee">
```

该标记有 4 个属性, SRC 属性的值是 cup.gif, WIDTH 属性的值是 89, HEIGHT 属性的值是 67, ALT 属性的值是 Cup of coffee。然而,与 HTML 不同, XML 中属性的值必须加引号,并且必须有与起始标记匹配的终止标记。上述标记实例用 XML 表示为:

```
<IMG SRC="IMAGE\cup.gif" WIDTH="89" HEIGHT="67" ALT="Cup of coffee">
```

```
</IMG>
```



HTML 与 XML 的另一个不同点是: XML 没有赋予 **IMG** 标记及其属性任何特殊意义。特别是不能保证 XML 浏览器会把该标记翻译成装载并显示 cup.gif 文件中的图像的指令。

可以很容易将属性句法应用到棒球示例中,这样会使标记显得简洁明了。例如,我们可以用 SEASON 元素中的一个 YEAR 属性代替一个 YEAR 子元素:

```
<SEASON YEAR="1998">
```

```
</SEASON>
```

另一方面, LEAGUE 应当是 SEASON 的一个子元素而不是一个属性。因为在一个赛季中可能有两个联赛,而且子元素在任何时候都有可能指代不同的事物。但是,一个元素的属性名是不能重复的。因此,不能像下面的示例那样编写 SEASON 元素。

```
<SEASON YEAR="1998" LEAGUE="National" League="American">
```

```
</SEASON>
```

LEAGUE 确实是一个子元素而不是一个属性的另一个原因是，它含有子结构，可进一步分成多个 DIVISION 元素，其属性值是无格式文本。XML 元素可对结构方便地加以编码，而属性值却不能。

联赛名称是无结构的普通文本，每一个联赛只有一个名称，因此，LEAGUE 元素含有一个 NAME 属性，而不是一个 LEAGUE\_NAME 子元素：

```
<LEAGUE NAME="National League">
```

```
</LEAGUE>
```

由于属性与元素的联系比子元素更加紧密，上述的属性名应使用 NAME，而不是 LEAGUE\_NAME，不会出错。各分部和球队这些子元素同样有 NAME 属性，不必担心与联赛名混淆。一个标记可以有多个属性，只要这些属性不同名即可。我们可以将各队所在的城市看作一个属性，如下所示：

```
<LEAGUE NAME="American League">
```

```
    <DIVISION NAME="East">
```

```
        <TEAM NAME="Orioles" CITY="Baltimore"></TEAM>
```

```
        <TEAM NAME="Red Sox" CITY="Boston"></TEAM>
```

```
        <TEAM NAME="Yankees" CITY="New York"></TEAM>
```

```
        <TEAM NAME="Devil Rays" CITY="Tampa Bay"></TEAM>
```

```
        <TEAM NAME="Blue Jays" CITY="Toronto"></TEAM>
```

```
    </DIVISION>
```

```
</LEAGUE>
```

如果把每一项统计选作一个属性，一个队员将包括许多属性。下面的示例是用属性表示的 Joe Girardi 在 1998 年的统计数据。

```
<PLAYER GIVEN_NAME="Joe" SURNAME="Girardi"
```

```
    GAMES="78" AT_BATS="254" RUNS="31" HITS="70"
```

```
    DOUBLES="11" TRIPLES="4" HOME_RUNS="3"
```

```
    RUNS_BATTED_IN="31" WALKS="14" STRUCK_OUT="38"
```

```
    STOLEN_BASES="2" CAUGHT_STEALING="4"
```

```
    SACRIFICE_FLY="1" SACRIFICE_HIT="8"
```

```
    HIT_BY_PITCH="2">
```

```
</PLAYER>
```

清单 5-1 应用这种新的属性样式展示了一个完整的 XML 文档，文档是 1998 年重要棒球联赛的统计。展示的信息与上一章清单 4-1 中的一样（包括 2 个联赛、6 个分部、30 个球队和 9 名运动员），只是标记的方式不同。图 5-1 显示了装入到 Internet Explorer 5.0 中的没有任何样式表的文档。

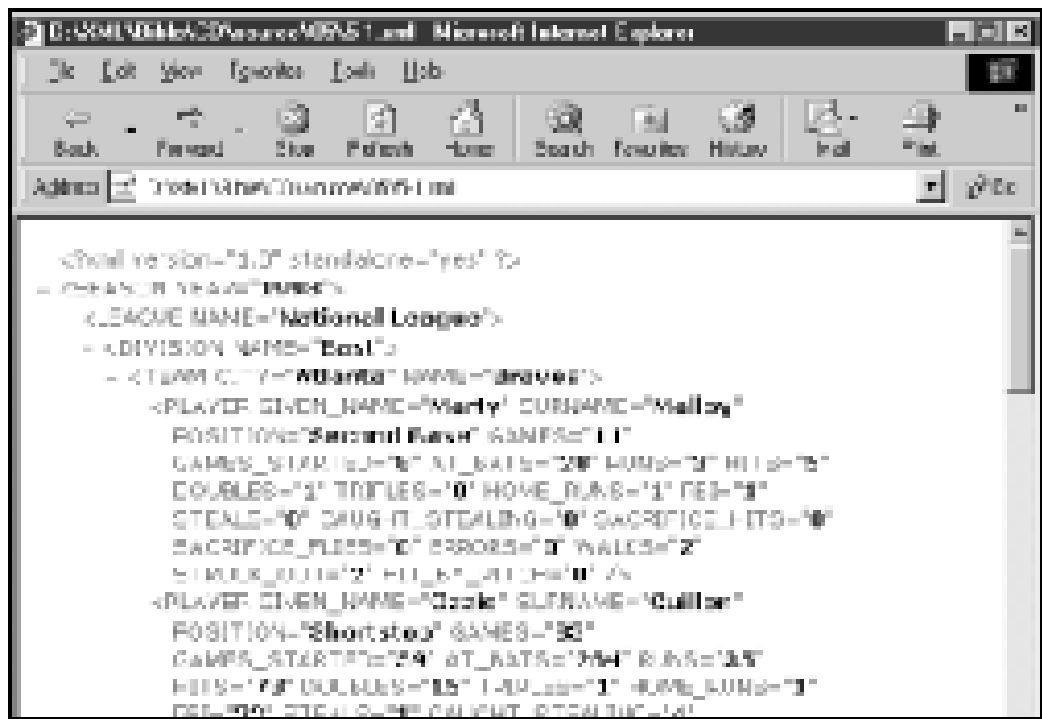


图 5-1 1998 年主要棒球联赛统计，使用属性表示信息

清单 5-1: 使用属性存储棒球统计的完整的 XML 文档

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

<SEASON YEAR="1998">

  <LEAGUE NAME="National League">

    <DIVISION NAME="East">

      <TEAM CITY="Atlanta" NAME="Braves">

        <PLAYER GIVEN_NAME="Marty" SURNAME="Malloy"

          POSITION="Second Base" GAMES="11" GAMES_STARTED="8"

          AT_BATS="28" RUNS="3" HITS="5" DOUBLES="1"

          TRIPLES="0" HOME_RUNS="1" RBI="1" STEALS="0"

          CAUGHT_STEALING="0" SACRIFICE_HITS="0"

          SACRIFICE_FLIES="0" ERRORS="0" WALKS="2"

          STRUCK_OUT="2" HIT_BY_PITCH="0">
```

</PLAYER>

<PLAYER GIVEN\_NAME="Ozzie" SURNAME="Guillen"

POSITION="Shortstop" GAMES="83" GAMES\_STARTED="59"

AT\_BATS="264" RUNS="35" HITS="73" DOUBLES="15"

TRIPLES="1" HOME\_RUNS="1" RBI="22" STEALS="1"

CAUGHT\_STEALING="4" SACRIFICE\_HITS="4"

SACRIFICE\_FLIES="2" ERRORS="6" WALKS="24"

STRUCK\_OUT="25" HIT\_BY\_PITCH="1">

</PLAYER>

<PLAYER GIVEN\_NAME="Danny" SURNAME="Bautista"

POSITION="Outfield" GAMES="82" GAMES\_STARTED="27"

AT\_BATS="144" RUNS="17" HITS="36" DOUBLES="11"

TRIPLES="0" HOME\_RUNS="3" RBI="17" STEALS="1"

CAUGHT\_STEALING="0" SACRIFICE\_HITS="3"

SACRIFICE\_FLIES="2" ERRORS="2" WALKS="7"

STRUCK\_OUT="21" HIT\_BY\_PITCH="0">

</PLAYER>

<PLAYER GIVEN\_NAME="Gerald" SURNAME="Williams"

POSITION="Outfield" GAMES="129" GAMES\_STARTED="51"

AT\_BATS="266" RUNS="46" HITS="81" DOUBLES="18"

TRIPLES="3" HOME\_RUNS="10" RBI="44" STEALS="11"

CAUGHT\_STEALING="5" SACRIFICE\_HITS="2"

SACRIFICE\_FLIES="1" ERRORS="5" WALKS="17"

STRUCK\_OUT="48" HIT\_BY\_PITCH="3">

</PLAYER>

<PLAYER GIVEN\_NAME="Tom" SURNAME="Glavine"



POSITION="Starting Pitcher" GAMES="33"  
GAMES\_STARTED="33" WINS="20" LOSSES="6" SAVES="0"  
COMPLETE\_GAMES="4" SHUT\_OUTS="3" ERA="2.47"  
INNINGS="229.1" HOME\_RUNS\_AGAINST="13"  
RUNS\_AGAINST="67" EARNED\_RUNS="63" HIT\_BATTER="2"  
WILD\_PITCHES="3" BALK="0" WALKED\_BATTER="74"  
STRUCK\_OUT\_BATTER="157">

</PLAYER>

<PLAYER GIVEN\_NAME="Javier" SURNAME="Lopez"

POSITION="Catcher" GAMES="133" GAMES\_STARTED="124"  
AT\_BATS="489" RUNS="73" HITS="139" DOUBLES="21"  
TRIPLES="1" HOME\_RUNS="34" RBI="106" STEALS="5"  
CAUGHT\_STEALING="3" SACRIFICE\_HITS="1"  
SACRIFICE\_FLIES="8" ERRORS="5" WALKS="30"  
STRUCK\_OUT="85" HIT\_BY\_PITCH="6">

</PLAYER>

<PLAYER GIVEN\_NAME="Ryan" SURNAME="Klesko"

POSITION="Outfield" GAMES="129" GAMES\_STARTED="124"  
AT\_BATS="427" RUNS="69" HITS="117" DOUBLES="29"  
TRIPLES="1" HOME\_RUNS="18" RBI="70" STEALS="5"  
CAUGHT\_STEALING="3" SACRIFICE\_HITS="0"  
SACRIFICE\_FLIES="4" ERRORS="2" WALKS="56"  
STRUCK\_OUT="66" HIT\_BY\_PITCH="3">

</PLAYER>

<PLAYER GIVEN\_NAME="Andres" SURNAME="Garraga"

POSITION="First Base" GAMES="153" GAMES\_STARTED="151"

AT\_BATS="555" RUNS="103" HITS="169" DOUBLES="27"

TRIPLES="1" HOME\_RUNS="44" RBI="121" STEALS="7"

CAUGHT\_STEALING="6" SACRIFICE\_HITS="0"

SACRIFICE\_FLIES="5" ERRORS="11" WALKS="63"

STRUCK\_OUT="146" HIT\_BY\_PITCH="25">

</PLAYER>

<PLAYER GIVEN\_NAME="Wes" SURNAME="Helms"

POSITION="Third Base" GAMES="7" GAMES\_STARTED="2"

AT\_BATS="13" RUNS="2" HITS="4" DOUBLES="1"

TRIPLES="0" HOME\_RUNS="1" RBI="2" STEALS="0"

CAUGHT\_STEALING="0" SACRIFICE\_HITS="0"

SACRIFICE\_FLIES="0" ERRORS="1" WALKS="0"

STRUCK\_OUT="4" HIT\_BY\_PITCH="0">

</PLAYER>

</TEAM>

<TEAM CITY="Florida" NAME="Marlins">

</TEAM>

<TEAM CITY="Montreal" NAME="Expos">

</TEAM>

<TEAM CITY="New York" NAME="Mets">

</TEAM>

<TEAM CITY="Philadelphia" NAME="Phillies">

</TEAM>

</DIVISION>

<DIVISION NAME="Central">

<TEAM CITY="Chicago" NAME="Cubs">

</TEAM>

<TEAM CITY="Cincinnati" NAME="Reds">

</TEAM>

<TEAM CITY="Houston" NAME="Astros">

</TEAM>

<TEAM CITY="Milwaukee" NAME="Brewers">

</TEAM>

<TEAM CITY="Pittsburgh" NAME="Pirates">

</TEAM>

<TEAM CITY="St. Louis" NAME="Cardinals">

</TEAM>

</DIVISION>

<DIVISION NAME="West">

<TEAM CITY="Arizona" NAME="Diamondbacks">

</TEAM>

<TEAM CITY="Colorado" NAME="Rockies">

</TEAM>

<TEAM CITY="Los Angeles" NAME="Dodgers">

</TEAM>

<TEAM CITY="San Diego" NAME="Padres">

</TEAM>

<TEAM CITY="San Francisco" NAME="Giants">

</TEAM>

</DIVISION>

</LEAGUE>

<LEAGUE NAME="American League">

<DIVISION NAME="East">

<TEAM CITY="Baltimore" NAME="Orioles">

</TEAM>

<TEAM CITY="Boston" NAME="Red Sox">

</TEAM>

<TEAM CITY="New York" NAME="Yankees">

</TEAM>

<TEAM CITY="Tampa Bay" NAME="Devil Rays">

</TEAM>

<TEAM CITY="Toronto" NAME="Blue Jays">

</TEAM>

</DIVISION>

<DIVISION NAME="Central">

<TEAM CITY="Chicago" NAME="White Sox">

</TEAM>

<TEAM CITY="Kansas City" NAME="Royals">

</TEAM>

<TEAM CITY="Detroit" NAME="Tigers">

</TEAM>

<TEAM CITY="Cleveland" NAME="Indians">

</TEAM>

<TEAM CITY="Minnesota" NAME="Twins">

</TEAM>

</DIVISION>

<DIVISION NAME="West">

<TEAM CITY="Anaheim" NAME="Angels">

</TEAM>

<TEAM CITY="Oakland" NAME="Athletics">

</TEAM>

<TEAM CITY="Seattle" NAME="Mariners">

</TEAM>

<TEAM CITY="Texas" NAME="Rangers">

</TEAM>

</DIVISION>

</LEAGUE>

</SEASON>

在清单 5-1 中，队员的信息是用属性表示的，清单 4-1 是用元素内容表示的。当然，也有合二为一的表示方法。例如，队员的名字作为元素内容，而其他部分作为属性，如下所示：

<P>

On Tuesday<PLAYER GAMES="78" AT\_BATS="254" RUNS="31"

HITS="70" DOUBLES="11" TRIPLES="4" HOME\_RUNS="3"

RUNS\_BATTED\_IN="31" WALKS="14" STRIKE\_OUTS="38"

STOLEN\_BASES="2" CAUGHT\_STEALING="4"

SACRIFICE\_FLY="1" SACRIFICE\_HIT="8"

HIT\_BY\_PITCH="2">Joe Girardi</PLAYER>struck out twice

and...

</P>

这样处理后，Joe Girardi 的名字会以一个超链接文本的脚注或者工具提示包含在页面文本中，同时能保证希望深究它的读者得到该统计。一组相同的数据可以用多种方法进行编码，具体选择哪一种编码方法取决于用户特定应用的需要。

## 5.2 属性与元素的对比

何时使用子元素或属性没有严格的规则可循，通常要看哪一种更适合自己的应用的需要。随着经验的增长就会有一种感觉，知道在何时使用属性比子元素更简单，反之亦然。一个好的经验规则是数据本身应当存储在元素中，而有关数据的信息（元数据）应当存储在属性中。不知道怎么做时，就把信息放在元素中。

为区分数据与元数据，首先要问自己是否会有些读者希望看到一条特别的信息。如果答案是肯定的，该信息应当包含在一个子元素中。相反，则应包含在一个属性中。如果从该文档中删除所有标记与属性，文档的基本信息应当还存在。属性是放置 ID 号、URL、参考资料及其他与读者不直接相关的信息的好地方。但是，把元数据作为属性存储的基本规则还有许多例外。这些例外包括：

- 属性不能很好地保持原文的结构。
- 元素允许包括元数据（有关信息的更深层次的信息）。
- 每个人对元数据和非元数据的理解是不一样的。
- 面对以后的变化，元素更具扩展性。

### 5.2.1 结构化的元数据

需要特别记住的是元素可以有子结构而属性没有。这使元素更加灵活，更方便我们将元数据编译成子元素。例如，设想我们在写一篇论文，而且希望其中包含某件事情的出处，结果可能是这样：

```
<FACT SOURCE="The Biographical History of Baseball,
```

```
Donald Dewey and Nicholas Acocella (New York:Carroll &
```

```
Graf Publishers, Inc. 1995)p. 169">
```

```
    Josh Gibson is the only person in the history of baseball to
```

```
    hit a pitch out of Yankee Stadium.
```

```
</FACT>
```

很明显，信息 “The Biographical History of Baseball, Donald Dewey and Nicholas Acocella(New York:Carroll &Graf Publishers, Inc. 1995)p. 169” 是元数据。它不是事情本身而更像事情的有关信息。SOURCE 属性暗含了许多子结构。按照下文的方法组织上面的信息可能更有效：

```
<SOURCE>
```

```
<AUTHOR>Donald Dewey</AUTHOR>
```

```
<AUTHOR>Nicholas Acocella</AUTHOR>
```

```
<BOOK>
```

```
<TITLE>The Biographical History of Baseball</TITLE>
```

```
<PAGES>169</PAGES>
```

```
<YEAR>1995</YEAR>
```

</BOOK>

</SOURCE>

此外，使用元素代替属性包含附加的信息更容易、直接，例如作者的 e-mail 地址，可找到文档的电子副本的 URL，日报特刊的标题或主题以及其他看似重要的信息等。

日期是另外一个常见的例子。与学术论文有关的一个常用的元数据是第一次收到论文日期，它对建立发明创造的优先权很重要。在 ARTICLE 标记中很容易包含一个 DATE 属性，如下所示：

<ARTICLE DATE="06/28/1969">

Polymerase Reactions in Organic Compounds

</ARTICLE>

DATE 属性中含有用/表示的子结构，如果要从属性值中获得该结构要比读取 DATE 元素的子元素困难得多，如下所示：

<DATE>

<YEAR>1969</YEAR>

<MONTH>06</MONTH>

<DAY>28</DAY>

</DATE>

例如，使用 CSS 或 XSL 很容易将日期或月份格式化为看不见形式，因此只会出现年份。请看下面使用 CSS 的例子：

YEAR {display:inline}

MONTH {display:none}

DAY {display:none}

如果 DATE 是作为属性存储的，几乎没有简单的办法可以访问其中任何一部分。我们只有用一种类似 ECMAScript 或 Java 的编程语言写一个单独的程序，才能分析其日期格式。使用标准的 XML 工具和子元素做起来就比较容易。

另外，属性句法显得模糊不清，“10/11/1999”究竟表示 10 月 11 日还是 11 月 10 日？不同国家的读者对它的理解是不同的。即使语法分析程序能够识别某种格式，但不能保证其他人能够正确输入日期。作此对照用 XML 表示就不会摸棱两可。

最后，使用 DATE 子元素允许一个元素有多个日期。例如，学术论文通常要交还作者修改。在此情况下，记录再次收到修改过的论文的日期也很重要。例如：

<ARTICLE>

<TITLE>

Maximum Projectile Velocity in an Augmented Railgun

```
</TITLE>

<AUTHOR>Elliott Harold</AUTHOR>

<AUTHOR>Bruce Bukiet</AUTHOR>

<AUTHOR>William Peter</AUTHOR>

<DATE>

<YEAR>1992</YEAR>

<MONTH>10</MONTH>

<DAY>29</DAY>

</DATE>

<DATE>

<YEAR>1993</YEAR>

<MONTH>10</MONTH>

<DAY>26</DAY>

</DATE>

</ARTICLE>
```

再比如，在 HTML 中，IMG 标记的 ALT 属性被限定为一个单独的文本字符串。虽然一幅图片比成千的单词更能说明问题，但还是应该用已标记的文本来代替一个 IMG 标记。例如，考虑图 5-2 中的饼形图。

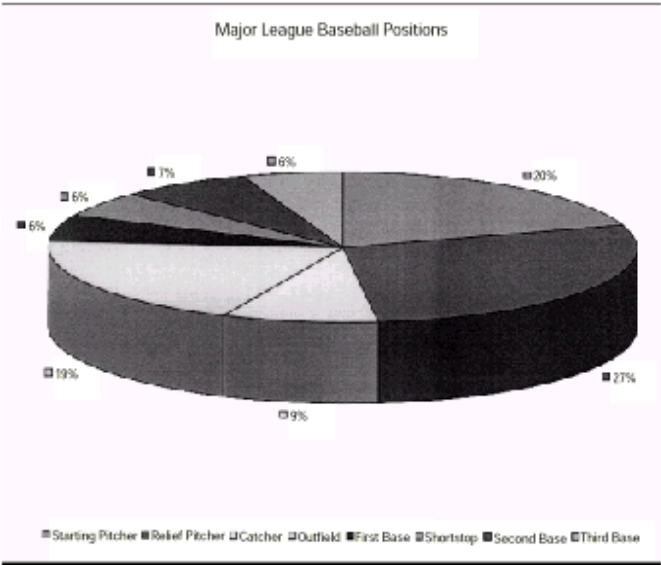


图 5-2 主要棒球联赛中各位置球员的分布情况

使用 ALT 属性对该图的最好描述如下：



<IMG SRC="IMAGE\05021.gif"

ALT="Pie Chart of Positions in Major League Baseball"

WIDTH="819" HEIGHT="623">

</IMG>

如果对上图使用一个 ALT 子元素描述，会更具灵活性，因为我们可以其中嵌入标记。例如，使用一个写有相关数字的一览表去替代饼形图：

<IMG SRC="IMAGE\05021.gif" WIDTH="819" HEIGHT="623">

<ALT>

<TABLE>

<TR>

<TD>Starting Pitcher</TD><TD>242</TD><TD>20%</TD>

</TR>

<TR>

<TD>Relief Pitcher</TD><TD>336</TD><TD>27%</TD>

</TR>

<TR>

<TD>Catcher</TD><TD>104</TD><TD>9%</TD>

</TR>

<TR>

<TD>Outfield</TD><TD>235</TD><TD>19%</TD>

</TR>

<TR>

<TD>First Base</TD><TD>67</TD><TD>6%</TD>

</TR>

<TR>

<TD>Shortstop</TD><TD>67</TD><TD>6%</TD>

</TR>

```

<TR>

<TD>Second Base</TD><TD>88</TD><TD>7%</TD>

</TR>

<TR>

<TD>Third Base</TD><TD>67</TD><TD>6%</TD>

</TR>

</TABLE>

</ALT>

</IMG>

```

在得不到位图图片的情况下，甚至可以使用实际的 Postscript、SVG 或 VML 代码来形成该图片。

## 5.2.2 元元数据

元素可用于元数据，同样也可用于元元数据，或者信息的深层相关信息。例如，一首诗的作者是一首诗的元数据，书写作者姓名所用的语言就是这首诗的元元数据。特别是对于明显的非罗马语言，这并非是无要紧要的。例如，Odyssey 的作者是 Homer 还是 Ω μ η ο δ ? 如果使用元素就可以很容易写出：

```

<POET LANGUAGE="English">Homer</POET>

<POET LANGUAGE="Greek">Ω μ η ο δ </POET>

```

但是，如果 POET 是一个属性而不是一个元素，如下所示的这种不易操作的结构会让人感到纠缠不清：

```

<POEM POET="Homer" POET_LANGUAGE="English"

POEM_LANGUAGE="English">

Tell me, O Muse, of the cunning man...

</POEM>

```

而且如果想要同时提供诗人的英文名与希腊名的时候，这种表示方法会更显得重要：

```

<POEM POET_NAME_1="Homer" POET_LANGUAGE_1="English"

POET_NAME_2=" Ω μ η ο δ " POET_LANGUAGE_2="Greek"

POEM_LANGUAGE="English">

Tell me, O Muse, of the cunning man...

</POEM>

```

### 5.2.3 有关元数据的说明

判断元数据的决定权掌握在读者手中，不同的读者和他们的阅读目的决定哪些是元数据，哪些是数据。例如，阅读一份学报上的文章，作者的名字与文章的内容相比就显得无足轻重。但是，如果作为晋升委员会的委员浏览学报来确定发表与未发表文章的人员，作者的名字与所发表文章的数量比其内容更重要。

事实上，人们也许会改变对数据和元数据的看法。今天看似无关紧要的东西，下周可能会变得很有用。你可以使用样式单隐藏今天看似不重要的元素，在以后可改变样式单将其显示出来。但是，显示一个原先存储在属性中的信息很困难。通常在此情况下需要重写整个文档，而不是简单地修改样式单。

### 5.2.4 元素更具扩展性

在只需要传达一两个字的非结构性信息时，使用属性是很方便的。在此情况下，显然不需要一个子元素。但是这并不排除日后需要它。

例如，目前可能只需要存储一篇文章的作者名而不必区分名和姓。但将来可能会需要存储姓名、e-mail 地址、机构、邮政通信处、URL 以及更多的东西。如果把文章的作者保存为一个元素，在其中添加子元素包含这些附加的信息会很容易。

尽管上述任何改动都需要重新修改文档、样式单和相关的程序，但是把一个简单的元素修改为元素树比把一个属性修改为元素树简单得多。而且使用了属性就只好继续使用下去。扩展属性句法使之超越最初的设计范围也很困难。

### 5.2.5 使用属性的最佳时机

在前面已经详尽阐述了应当使用子元素代替属性的原因，然而，必须指出的是，有时候使用属性是有意义的。首先，同前面提到的一样，属性非常适用于那些读者未必想看见的没有子结构的简单数据。例如，IMG 中的 HEIGHT 和 WIDTH 属性，尽管这些属性值随图片的改变而改变，但是无法想象属性中的数据除了一个很短的字符串外还能是什么。HEIGHT 和 WIDTH 分别是一维的数，因此作为属性执行起来很顺利。

此外，属性也适用于与文档有关而与文档内容无关的简单信息。例如，给每一个元素指定一个 ID 属性常常是有用的，这是文档中仅隶属于元素的唯一字符串。该字符串可用于不同的目的，包括链接到文档中的特殊元素。甚至在文档发生改变时，这些元素会随之移动。例如：

```
<SOURCE ID="S1">
```

```
<AUTHOR ID="A1">Donald Dewey</AUTHOR>
```

```
<AUTHOR ID="A2">Nicholas Acocella</AUTHOR>
```

```
<BOOK ID="B1">
```

```
<TITLE ID="B2">
```

```
The Biographical History of Baseball
```

```
</TITLE>
```

```
<PAGES ID="B3">169</PAGES>
```

```
<YEAR ID="B4">1995</YEAR>
```

```
</BOOK>
```

</SOURCE>

利用 ID 属性使链接文档中的特定元素成为可能。这样它们就有与 HTML 中 A 元素的 NAME 属性一样的功能。其他与链接有关的数据——HREF 属性指明的链接目标，SRC 属性指定的图像和二进制数据等等——作为属性都很合适。



在第 16 章“Xlink”和第 17 章“XPointer”中讨论 XLL——可扩展链接语言时，会看到更多的这种例子。

属性也常用于存储文档的特定样式信息。例如，TITLE 元素一般是以粗体出现，但是如果使一个 TITLE 元素有粗体和斜体两种字体，可以这样描述：

```
<TITLE style="font-style:italic">Significant Others</TITLE>
```

这样做可以在不改变文档树状结构的情况下嵌入样式信息。虽然最理想的方法是使用一个单独的元素，但当不能在处理的标记集里添加元素时，这个方案会给文档作者更多的控制权。例如，一个站点的管理员需要使用某一特定的 DTD，而且不希望任何人修改该 DTD。除此之外，还要允许他人对个别的页面做微小的校正。使用这种方案时要有所节制，否则很快会发现自己又陷入了 HTML 的“地狱”中，使用 XML 的本意是要避免这一“地狱”的。

使用属性的最后一个原因是为了保持与 HTML 的兼容性。甚至扩展到使用的标记，对于诸如<IMG>、<P>和<TD>看起来与 HTML 相似的标记还是使用标准的 HTML 属性为好。这样做有双重好处，至少使传统的浏览器能够部分地分析和显示你的文档，而且对于文档的作者来说更熟悉这种方式。

## 5.3 空标记

上一章中没有属性的方式是一种极端的情况，由此可能会想到另一个极端——将所有信息全部存储在属性中，而不是存储在内容中。通常不推荐使用这种方式。把信息全部存储在元素内容中同样也是极端的，只是实际处理起来更容易。这一节考虑仅使用属性来说明的可能性。

只要元素中没有内容，就可以使用空标记来简化。可以只包含一个空标记而不是一个起始标记和一个终止标记。空标记与起始标记的区别在于结束标记使用“/>”而不是简单的“>”。例如，不是<PLAYER></PLAYER>而是<PLAYER/>。

空标记可以包含属性。例如，下面是关于 Joe Girardi 的一个空标记，含有 7 个属性：

```
<PLAYER GIVEN_NAME="Joe" SURNAME="Girardi"
  GAMES="78" AT_BATS="254" RUNS="31" HITS="70"
  DOUBLES="11" TRIPLES="4" HOME_RUNS="3"
  RUNS_BATTED_IN="31" WALKS="14" STRUCK_OUT="38"
  STOLEN_BASES="2" CAUGHT_STEALING="4"
  SACRIFICE_FLY="1" SACRIFICE_HIT="8"
  HIT_BY_PITCH="2"/>
```

XML 句法分析器对空标记的处理与非空标记是一样的。下面的 PLAYER 元素与前面的空标记元素 PLAYER 精确地说是等价的（尽管不是完全一致）：

```
<PLAYER GIVEN_NAME="Joe" SURNAME="Girardi"
  GAMES="78" AT_BATS="254" RUNS="31" HITS="70"
  DOUBLES="11" TRIPLES="4" HOME_RUNS="3"
  RUNS_BATTED_IN="31" WALKS="14" STRUCK_OUT="38"
  STOLEN_BASES="2" CAUGHT_STEALING="4"
  SACRIFICE_FLY="1" SACRIFICE_HIT="8"
  HIT_BY_PITCH="2"></PLAYER>
```

<PLAYER/>与<PLAYER></PLAYER>之间的不同只是句法表面的不同，而没有别的不同。如果不喜欢空标记句法或者阅读起来感到困难，就不要使用它。

## 5.4 XSL

如图 5-1 所示，属性在文档的 XML 源视图中是可见的。但是一旦把 CSS 样式单施加其上，属性就会消失。图 5-3 显示了清单 5-1 使用前面章节中棒球统计样式单后的样子。它看起来是一个空白文档，因为 CSS 样式单仅适用于元素内容，而不适用于属性。在使用 CSS 时，希望显示给读者的任何数据应当是元素内容的一部分，而不是它的属性。

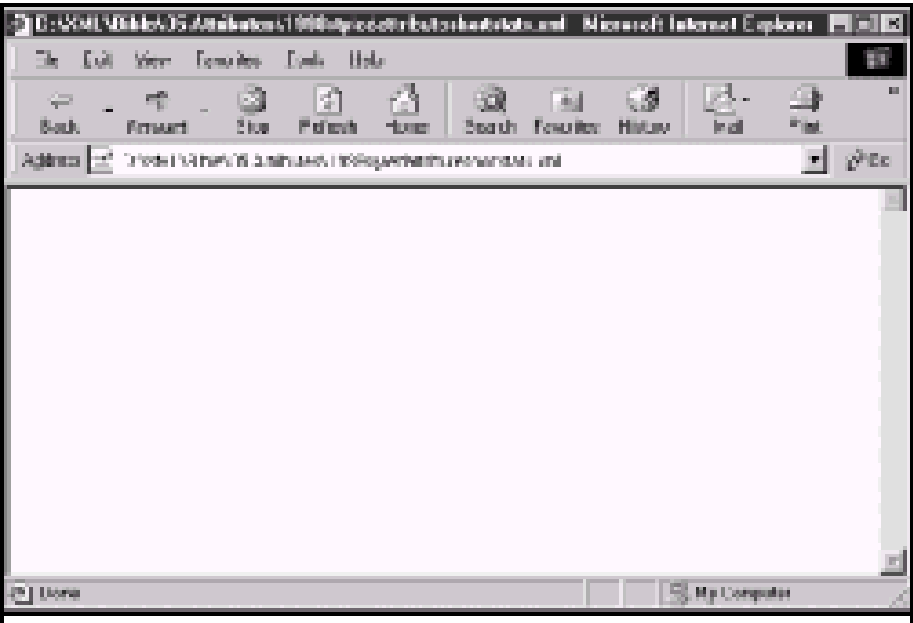


图 5-3 当 CSS 施加于一个元素中不含任何字符数据的 XML 文档时显示的空白文档

但是，仍然有一种可选择样式单语言能够访问并显示属性数据。这就是 Extensible Style Language (XSL)；Internet Explorer 5.0 至少部分支持它。XSL 分为两部分：变换部分和格式化部分。

XSL 替换部分能够将一个标记替换为另一个标记。通过定义替换规则，使用标准的 HTML 标记代替 XML 标记或者使用 HTML 标记与 CSS 属性来替换 XML 标记。同时还可以在文档中重新安排元素和在 XML 文档中添加没有出现过的附加内容。

XSL 格式化部分把功能强大的文档视图定义为页面。XSL 格式化功能能够指定页面的外观和编排，包括多个专栏、围绕主题的字数、行间距、相配的字属性等等。它的功能非常强大，足可以为网络和打印自动处理来自于相同源文档的编排任务。例如，XSL 格式化允许包含有 show times（在线播放）和广告的 XML 文档生成本地报纸上电视节目单的打印及在线版本。但是 IE 5.0 和大多数其他工具还不支持 XSL 格式化。因此，本节重点介绍 XSL 变换。



XSL 格式化将在第 15 章 XSL 格式化对象中讨论。

### 5.4.1 XSL 样式单模板

每个 XSL 样式单包括一些模板，XML 文档中的数据会注入其中。例如，某一模板如下所示：

<HTML>

<HEAD>

<TITLE>

XSL Instructions to get the title

</TITLE>

</HEAD>

<H1>XSL Instructions to get the title </H1>

<BODY>

XSL Instructions to get the statistics

</BODY>

</HTML>

斜体部分将由特定的 XSL 元素取代，这些元素把基本的 XML 文档中的数据复制到该模板中。该模板可用于许多不同的数据集。例如，模板设计用于处理棒球示例，那么相同的样式单能够显示不同赛季的统计。

这令人想起了用于 HTML 的某种服务器端嵌入方案。事实上，这与服务器端嵌入方案极其类似。但是，XML 源文档与 XSL 样式单的实际变换发生在客户端，而不是服务器端。而且输出的文档可以是任何一种结构完整的 XML 文档，不必是 HTML 文档。

XSL 指令能够提取存储于 XML 文档中的任何数据。包括元素内容、元素名称和对我们的示例很重要的元素属性。特定的元素由一种模式选定，该模式会考虑元素的名称和值、元素的属性名和值以及在 XML 文档树状结构中的绝对和相对位置等等。数据一经从一个元素中取出，就可以移动、复制和经过其他多种处理。在这个简要的介绍中描述了使用 XML 变换部分所能做的事情。读者将学到使用 XSL 编写一些能够立即在网上看到的令人吃惊的文档。



在第 14 章的“XSL 变换”中对 XSL 的变换作了彻底的阐述。

#### 5.4.2 文档的主体

请看下面的简单例子，并把它应用于清单 5-1 所示的棒球统计的 XML 文档中，清单 5-2 是一个 XSL 样式单。它提供 XML 数据将要注入的 HTML “模子”。

清单 5-2：一个 XSL 样式单

<?xml version="1.0"?>

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

<xsl:template match="/">

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

<HEAD>

<TITLE>

Major League Baseball Statistics

```

</TITLE>

</HEAD>

<BODY>

<H1>Major League Baseball Statistics</H1>

<HR></HR>

Copyright 1999

<A HREF="http://www.macfaq.com/personal.html">

Elliott Rusty Harold

</A>

<BR />

<A HREF="mailto:elharo@metalab.unc.edu">

elharo@metalab.unc.edu

</A>

```

```

</BODY>

</HTML>

</xsl:template>

</xsl:stylesheet>

```

该清单像一个包含在 XSL:template 元素中的 HTML 文件，也就是说它的结构更像是这样：

```

<?xml version="1.0"?>

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

<xsl:template match="/">

HTML file goes here

</xsl:template>

</xsl:stylesheet>

```

清单 5-2 不仅是一个 XSL 样式单，同样是一个结构完整的 HTML 文档。它以一个 XML 名称开始，文档的根元素是 xsl:stylesheet。该样式单包含唯一的模板，把 XML 数据编码为一个 xsl:template 元素。xsl:template 元素有一个 match 属性，其值为 /，内容是一个结构完整的 HTML 文档。输出的 HTML 结构完整不是一种巧合。因为 HTML 首先必须是一个 XSL 样式单的一部分，并且 XSL 样式单是结构完整的 XML 文档，因此在一个 XSL 样式单中的所有 HTML 一定结构完整。



Web 浏览器尽量使 XML 文档各部分与每个 xsl: template 元素相匹配。/模板与文档的根即整个文档本身相匹配。浏览器读取模板并将来自 XML 中的数据插入 XSL 指令指明的位置。但是该特定模板不包含 XSL 指令。因此它的内容只是被逐字逐句地复制到 Web 浏览器中，产生如图 5-4 所示的输出结果。请注意该图不显示 XML 文档的任何数据，只显示 XSL 模板中的数据。把清单 5-2 中的 XSL 样式单与清单 5-1 中的 XML 文档连接起来很方便，只需增加一个<?XML-stYLESHEET?>处理指令，该指令位于 XML 声明和根元素之间，含有一个值为 text/xsl 的 type 属性和一个指向样式单的 href 属性。例如：

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

<?xml-stYLESHEET type="text/xsl" href="5-2.xsl"?>

<SEASON YEAR="1998">

...

```

这与在文档上连接 CSS 样式单的方法一样，唯一不同的是 type 属性的值为 text/xsl 而不是 text/css。

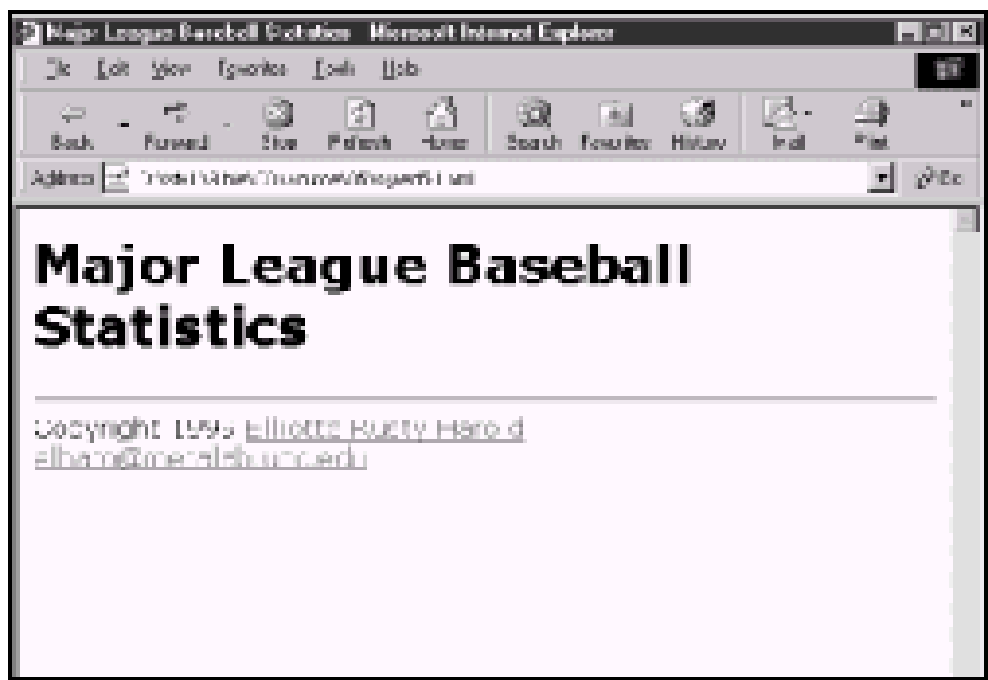


图 5-4 采用清单 5-2 中 XSL 样式单后，XML 文档中的数据而不是 XSL 模板中的数据消失了

5.4.3 标题

图 5-4 很明显丢失了数据。尽管清单 5-2 中的样式单显示了一些内容（与图 5-3 所示的 CSS 样式单不同），但是它没有显示 XML 文档中的任何数据。要添加这些数据需要使用 XSL 指令元素把 XML 源文档中的数据复制到 XSL 模板中。清单 5-3 增加了必要的 XSL 指令，从 SEASON 元素中抽取 YEAR 属性并把它插入到结果文档的 TITLE 和 H1 标头之间。图 5-5 显示了处理后的文档。

清单 5-3：一个含有抽取 SEASON 元素和 YEAR 属性指令的 XSL 样式单

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

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

<xsl:template match="/">

```

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

<HEAD>

<TITLE>

<xsl:for-each select="SEASON">

<xsl:value-of select="@YEAR"/>

</xsl:for-each>

Major League Baseball Statistics

</TITLE>

</HEAD>

<BODY>

<xsl:for-each select="SEASON">

<H1>

<xsl:value-of select="@YEAR"/>

Major League Baseball Statistics

</H1>

</xsl:for-each>

<HR></HR>

Copyright 1999

<A HREF="http://www.macfaq.com/personal.html">

Elliotte Rusty Harold

</A>

<BR />

<A HREF="mailto:elharo@metalab.unc.edu">

elharo@metalab.unc.edu

</A>

</BODY>

</HTML>

</xsl:template>

</xsl:stylesheet>

下面的新 XSL 指令能够从 SEASON 元素中抽取 YEAR 属性。

<xsl:for-each select="SEASON">

<xsl:value-of select="@YEAR"/>

</xsl:for-each>



图 5-5 清单 5-1 采用清单 5-3 所示的 XSL 样式单后的显示结果

这些指令出现两次是因为我们希望年份在输出结果中出现两次，一次在 H1 主题中，一次在 TITLE 中。这些指令每次出现都执行同样的功能。<xsl:for-each select="SEASON">寻出全部 SEASON 元素。<xsl:value-of select="@YEAR"/>插入 SEASON 元素中的 YEAR 属性值——这就是由<xsl:for-each select="SEASON">找到的字符串“1998”。

这非常重要，重述如下：xsl:for-each 选出源文档（例如清单 5-1）中的某一特定的 XML 元素，数据就从此元素中读取。Xsl:value-of 把所选取元素的某一特定部分复制到输出文档中。因此，必须使用两个 XSL 指令。使用任何一个都是无效的。

XSL 指令不同于输出的 HTML 和 H1 元素是因为这些指令都处于 XSL 的命名域内。也就是说所有的 XSL 元素名称都以 xsl: 开头。命名域由样式单根元素中的 xmlns:xsl 属性辨别。本书中的清单 5-2，5-3 和所有其他示例中 xmlns:xsl 属性的值都是 http://www.w3.org/tr/wd-xsl。



命名域将在第 18 章中详细阐述。

#### 5.4.4 联赛、分部和球队

下面通过添加一些 XSL 指令取出前面出现过的两个 LEAGUE 元素，并把这两个元素映射到 H2 标题中，如清单 5-4 所示。图 5-6 显示了使用该样式单后的文档。

清单 5-4：一个带有提取 LEAGUE 元素指令的 XSL 样式单

<?xml version="1.0"?>

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

<xsl:template match="/">

<HTML>

<HEAD>

<TITLE>

<xsl:for-each select="SEASON">

<xsl:value-of select="@YEAR"/>

</xsl:for-each>

Major League Baseball Statistics

</TITLE>

</HEAD>

<BODY>

<xsl:for-each select="SEASON">

<H1>

<xsl:value-of select="@YEAR"/>

Major League Baseball Statistics

</H1>

<xsl:for-each select="LEAGUE">

<H2 ALIGN="CENTER">

<xsl:value-of select="@NAME"/>

</H2>

</xsl:for-each>

</xsl:for-each>

<HR></HR>

Copyright 1999



```
<xsl:for-each select="LEAGUE">
```

```
<H2 ALIGN="CENTER">
```

```
<xsl:value-of select="@NAME"/>
```

```
</H2>
```

```
</xsl:for-each>
```

```
</xsl:for-each>
```

最外层的指令用于选取 SEASON 元素，只有找到该元素才能找到它的 YEAR 属性，并把它与另外的文本 Major League Baseball Statistics 一起放到<H1>与</H1>之间。下一步浏览器会循环选取 SEASON 元素的每一个 LEAGUE 子元素，并把它 NAME 属性值放到<H2 ALIGN="CENTER"> 与</H2>之间。尽管只有一个 xsl:for-each 与 LEAGUE 元素相配，但是它对 SEASON 元素内所有直接的 LEAGUE 子元素进行循环。因此，该模板在没有联赛和联赛数目不定的情况下都能工作。

同样的技巧可以用于设计表示小组的 H3 标题和表示各球队的 H4 标题。清单 5-5 演示了该程序，图 5-7 显示了使用这一样式单后的文档。各小组名称和各球队名称是从 XML 数据中读取的。

清单 5-5：一个带有提取 DIVISION 和 TEAM 元素指令的 XSL 样式单

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

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

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

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

```
<HEAD>
```

```
<TITLE>
```

```
<xsl:for-each select="SEASON">
```

```
<xsl:value-of select="@YEAR"/>
```

```
</xsl:for-each>
```

```
Major League Baseball Statistics
```

```
</TITLE>
```

```
</HEAD>
```

```
<BODY>
```

```
<xsl:for-each select="SEASON">
```

```
<H1>
```

<xsl:value-of select="@YEAR"/>

Major League Baseball Statistics

</H1>

<xsl:for-each select="LEAGUE">

<H2 ALIGN="CENTER">

<xsl:value-of select="@NAME"/>

</H2>

<xsl:for-each select="DIVISION">

<H3 ALIGN="CENTER">

<xsl:value-of select="@NAME"/>

</H3>

<xsl:for-each select="TEAM">

<H4 ALIGN="CENTER">

<xsl:value-of select="@CITY"/>

<xsl:value-of select="@NAME"/>

</H4>

</xsl:for-each>

</xsl:for-each>

</xsl:for-each>

</xsl:for-each>

<HR></HR>

Copyright 1999

<A HREF="http://www.macfaq.com/personal.html">

Elliotte Rusty Harold

</A>

<BR />

```
<A HREF="mailto:elharo@metalab.unc.edu">
```

```
elharo@metalab.unc.edu
```

```
</A>
```

```
</BODY>
```

```
</HTML>
```

```
</xsl:template>
```

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

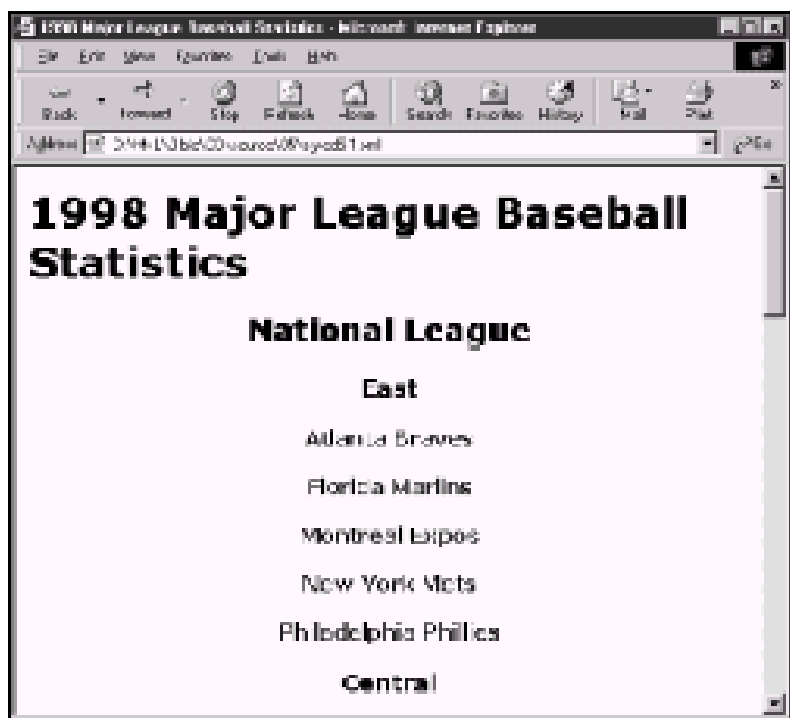


图 5-7 采用清单 5-5 所示的样式单后显示的小组名和队名

对于 TEAM 元素，它的 CITY 和 NAME 属性值是 H4 标题的内容。同时，请注意嵌套的选取赛季、小组和各队的 xsl:for-each 元素，它反映了文档自身的分支结构。这并不是一种巧合。其他方案可能不要求与文档的体系相匹配，但这一方案是最简单的，尤其适用于像清单 5-1 所示的棒球统计这样的高度结构化的数据。



## 5.4.5 球员

下一个步骤是为各队的每一个队员添加统计数字，最基本的方法是用一个表格表示。清单 5-6 展示的 XSL 样式单把队员以及他们的统计数据安排在一个表格中。其中没有引入新的 XSL 元素。相同的 `xsl:for-each` 和 `xsl:value-of` 元素被用于 `PLAYER` 元素和它的属性中。输出的是标准的 HTML 表格标记。图 5-8 显示了结果。

清单 5-6: 一个将球员及其统计数据放入表格的 XSL 样式单

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

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

<xsl:template match="/">

<HTML>

<HEAD>

<TITLE>

<xsl:for-each select="SEASON">

<xsl:value-of select="@YEAR"/>

</xsl:for-each>

Major League Baseball Statistics

</TITLE>

</HEAD>

<BODY>

<xsl:for-each select="SEASON">

<H1>

<xsl:value-of select="@YEAR"/>

Major League Baseball Statistics

</H1>

<xsl:for-each select="LEAGUE">

<H2 ALIGN="CENTER">

<xsl:value-of select="@NAME"/>

</H2>
```

```
<xsl:for-each select="DIVISION">

<H3 ALIGN="CENTER">

<xsl:value-of select="@NAME"/>

</H3>

<xsl:for-each select="TEAM">

<H4 ALIGN="CENTER">

<xsl:value-of select="@CITY"/>

<xsl:value-of select="@NAME"/>

</H4>

<TABLE>

<THEAD>

<TR>

<TH>Player</TH><TH>P</TH><TH>G</TH>

<TH>GS</TH><TH>AB</TH><TH>R</TH><TH>H</TH>

<TH>D</TH><TH>T</TH><TH>HR</TH><TH>RBI</TH>

<TH>S</TH><TH>CS</TH><TH>SH</TH><TH>SF</TH>

<TH>E</TH><TH>BB</TH><TH>SO</TH><TH>HBP</TH>

</TR>

</THEAD>

<TBODY>

<xsl:for-each select="PLAYER">

<TR>

<TD>

<xsl:value-of select="@GIVEN_NAME"/>

<xsl:value-of select="@SURNAME"/>

</TD>
```

<TD><xsl:value-of select="@POSITION"/></TD>

<TD><xsl:value-of select="@GAMES"/></TD>

<TD>

<xsl:value-of select="@GAMES\_STARTED"/>

</TD>

<TD><xsl:value-of select="@AT\_BATS"/></TD>

<TD><xsl:value-of select="@RUNS"/></TD>

<TD><xsl:value-of select="@HITS"/></TD>

<TD><xsl:value-of select="@DOUBLES"/></TD>

<TD><xsl:value-of select="@TRIPLES"/></TD>

<TD><xsl:value-of select="@HOME\_RUNS"/></TD>

<TD><xsl:value-of select="@RBI"/></TD>

<TD><xsl:value-of select="@STEALS"/></TD>

<TD>

<xsl:value-of select="@CAUGHT\_STEALING"/>

</TD>

<TD>

<xsl:value-of select="@SACRIFICE\_HITS"/>

</TD>

<TD>

<xsl:value-of select="@SACRIFICE\_FLIES"/>

</TD>

<TD><xsl:value-of select="@ERRORS"/></TD>

<TD><xsl:value-of select="@WALKS"/></TD>

<TD>

<xsl:value-of select="@STRUCK\_OUT"/>

</TD>

<TD>

<xsl:value-of select="@HIT\_BY\_PITCH"/>

</TD>

</TR>

</xsl:for-each>

</TBODY>

</TABLE>

</xsl:for-each>

</xsl:for-each>

</xsl:for-each>

</xsl:for-each>

<HR></HR>

Copyright 1999

<A HREF="http://www.macfaq.com/personal.html">

Elliotte Rusty Harold

</A>

<BR />

<A HREF="mailto:elharo@metalab.unc.edu">

elharo@metalab.unc.edu

</A>

</BODY>

</HTML>

</xsl:template>

</xsl:stylesheet>

#### 5.4.6 区分投手与击球手

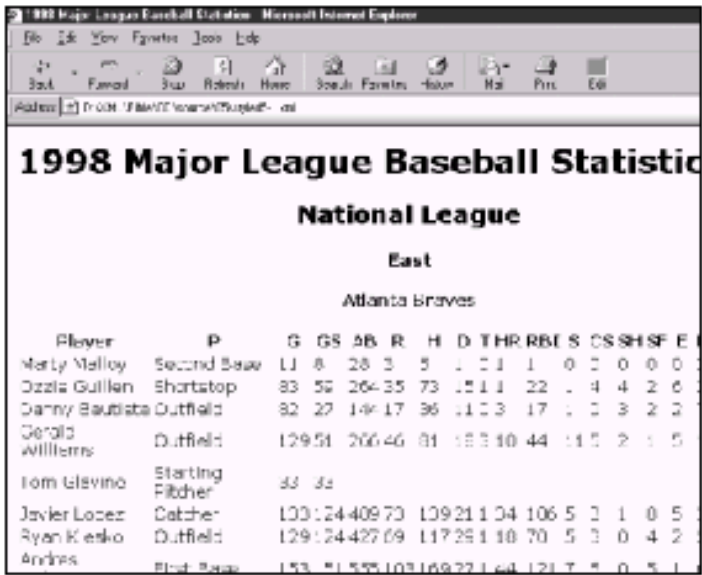
可以注意到图 5-8 中的一个缺点是没有正确处理投手。贯穿本章和第 4 章，投手是用完全不同的统计数字集表示的，无论他们的统计数据是存储在元素内容中还是属性中。因此，投手确实需要一个表格以区别于其他队员。在把队员放入表格之前必须查看他是不是投手。如果队员的 POSITION 属性包含“pitcher”就忽略他。然后在只包含投手队员元素的第二个表格中反转上面的过程，投手的 PLAYER 元素中 POSITION 属性值是字符串“pitcher”。

要完成这些还必须给 xsl:for-each 增加另外的选择队员的代码。我们不需要选择所有队员，相反只需要选择那些 POSITION 属性不是投手的队员，句法如下：

```
<xsl:for-each select="PLAYER [(@POSITION != Pitcher )]">
```

因为 XML 文档对首发投手和替补投手做了区分，正确的答案应该检查这两种情况：


```
<xsl:for-each select="PLAYER [(@POSITION != StartingPitcher )  
$and$(@POSITION != Relief Pitcher )]">
```



1998 Major League Baseball Statistics																	
National League																	
East																	
Atlanta Braves																	
Player	P	G	GS	AB	R	H	D	THR	RB	CS	SH	SF	E				
Marty Malloy	Second Base	11	8	28	3	5	1	1	1	0	0	0	0				
Ozzie Guillen	Shortstop	83	59	264	35	73	15	1	22	1	4	4	2	6			
Danny Bautista	Outfield	82	27	144	17	36	11	3	17	1	3	2	2				
Gerald Williams	Outfield	129	51	266	46	81	18	3	44	1	5	2	1	5			
Tom Glavino	Starting Pitcher	33	33														
Javier Lopez	Catcher	103	124	409	73	109	21	1	54	106	5	3	1	0	5		
Ryan Klesko	Outfield	129	124	427	69	117	25	1	10	70	5	3	0	4	2		
Andres	First Base	153	81	555	103	169	22	1	64	121	7	8	0	5	1		

图 5-8 采用清单 5-6 的 XSL 样式单后队员统计的显示情况

关于投手的清单，只需要把 Staring Pitcher 或者 Relief Pitcher 前的不等号变为等号。（仅仅把不等号改为等号是不能满足的，同时必须把 and 改为 or。）句法如下：



```
<xsl:for-each select="PLAYER[(@POSITION= Starting Pitcher )  
$or$(@POSITION= Relief Pitcher )]">
```

与 C 或 JAVA 语言不同，这里比较相等只用一个等号而不是双等号，因为 XSL 中没有赋值操作。

清单 5-7 显示的 XSL 样式单把投手和击球手区分在两个不同的表格中，投手表格为所有投手添加了常规统计项目。清单 5-1 将这些项目编码在以下属性中：wins（投中）、losses（失球）、saves（救球）、shutouts（被封杀）等等。相应省略了一些列标签以保持表格预定的宽度。图 5-9 显示了最后的结果。

Figure 5-9: A screenshot of a web browser displaying Major League Baseball statistics. The page is titled 'Major League Baseball Statistics' and shows a table of statistics for the Atlanta Braves and Florida Marlins. The table is styled with a light blue background and uses XSL stylesheets to distinguish between pitchers and batters. The table is organized into two main sections: 'Pitchers' and 'Batters'. The 'Pitchers' section lists players like Martin Muller, Ozzie Guillen, and Denny Bautista. The 'Batters' section lists players like Gerald Williams, Javier Lopez, Ryan Klesko, Andres Galarraga, and Wes Helms. The table includes various statistics such as games played, at-bats, runs, hits, doubles, triples, home runs, and batting average.

图 5-9 采用清单 5-7 的 XSL 样式单能够区分投手和击球手

清单 5-7: 区分投手和击球手的样式单

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

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

<xsl:template match="/">

<HTML>

<HEAD>

<TITLE>

<xsl:for-each select="SEASON">

<xsl:value-of select="@YEAR"/>

</xsl:for-each>

Major League Baseball Statistics

</TITLE>

</HEAD>

<BODY>

<xsl:for-each select="SEASON">

<H1>

<xsl:value-of select="@YEAR"/>
```

# Major League Baseball Statistics

</H1>

<xsl:for-each select="LEAGUE">

<H2 ALIGN="CENTER">

<xsl:value-of select="@NAME"/>

</H2>

<xsl:for-each select="DIVISION">

<H3 ALIGN="CENTER">

<xsl:value-of select="@NAME"/>

</H3>

<xsl:for-each select="TEAM">

<H4 ALIGN="CENTER">

<xsl:value-of select="@CITY"/>

<xsl:value-of select="@NAME"/>

</H4>

<TABLE>

<CAPTION><B>Batters</B></CAPTION>

<THEAD>

<TR>

<TH>Player</TH><TH>P</TH><TH>G</TH>

<TH>GS</TH><TH>AB</TH><TH>R</TH><TH>H</TH>

<TH>D</TH><TH>T</TH><TH>HR</TH><TH>RBI</TH>

<TH>S</TH><TH>CS</TH><TH>SH</TH><TH>SF</TH>

<TH>E</TH><TH>BB</TH><TH>SO</TH>

<TH>HBP</TH>

</TR>

```

</THEAD>

<TBODY>

<xsl:for-each select="PLAYER [ (@POSITION
!= Starting Pitcher )

                $and$ (@POSITION != Relief Pitcher ) ] ">

<TR>

<TD>

<xsl:value-of select="@GIVEN_NAME"/>

<xsl:value-of select="@SURNAME"/>

</TD>

<TD><xsl:value-of select="@POSITION"/></TD>

<TD><xsl:value-of select="@GAMES"/></TD>

<TD>

<xsl:value-of select="@GAMES_STARTED"/>

</TD>

<TD><xsl:value-of select="@AT_BATS"/></TD>

<TD><xsl:value-of select="@RUNS"/></TD>

<TD><xsl:value-of select="@HITS"/></TD>

<TD><xsl:value-of select="@DOUBLES"/></TD>

<TD><xsl:value-of select="@TRIPLES"/></TD>

<TD>

<xsl:value-of select="@HOME_RUNS"/>

</TD>

<TD><xsl:value-of select="@RBI"/></TD>

<TD><xsl:value-of select="@STEALS"/></TD>

<TD>

<xsl:value-of select="@CAUGHT_STEALING"/>

```



```
</TD>

<TD>

<xsl:value-of select="@SACRIFICE_HITS"/>

</TD>

<TD>

<xsl:value-of select="@SACRIFICE_FLIES"/>

</TD>

<TD><xsl:value-of select="@ERRORS"/></TD>

<TD><xsl:value-of select="@WALKS"/></TD>

<TD>

<xsl:value-of select="@STRUCK_OUT"/>

</TD>

<TD>

<xsl:value-of select="@HIT_BY_PITCH"/>

</TD>

</TR>

</xsl:for-each><!-- PLAYER -->

</TBODY>

</TABLE>

<TABLE>

<CAPTION><B>Pitchers</B></CAPTION>

<THEAD>

<TR>

<TH>Player</TH><TH>P</TH><TH>G</TH>

<TH>GS</TH><TH>W</TH><TH>L</TH><TH>S</TH>

<TH>CG</TH><TH>SO</TH><TH>ERA</TH>
```

<TH>IP</TH><TH>HR</TH><TH>R</TH><TH>ER</TH>

<TH>HB</TH><TH>WP</TH><TH>B</TH><TH>BB</TH>

<TH>K</TH>

</TR>

</THEAD>

<TBODY>

<xsl:for-each select="PLAYER[ (@POSITION= Starting Pitcher )

\$or\$( @POSITION= Relief Pitcher )]">

<TR>

<TD>

<xsl:value-of select="@GIVEN\_NAME"/>

<xsl:value-of select="@SURNAME"/>

</TD>

<TD><xsl:value-of select="@POSITION"/></TD>

<TD><xsl:value-of select="@GAMES"/></TD>

<TD>

<xsl:value-of select="@GAMES\_STARTED"/>

</TD>

<TD><xsl:value-of select="@WINS"/></TD>

<TD><xsl:value-of select="@LOSSES"/></TD>

<TD><xsl:value-of select="@SAVES"/></TD>

<TD>

<xsl:value-of select="@COMPLETE\_GAMES"/>

</TD>

<TD>

<xsl:value-of select="@SHUT\_OUTS"/>

</TD>

<TD><xsl:value-of select="@ERA"/></TD>

<TD><xsl:value-of select="@INNINGS"/></TD>

<TD>

<xsl:value-of select="@HOME\_RUNS\_AGAINST"/>

</TD>

<TD>

<xsl:value-of select="@RUNS\_AGAINST"/>

</TD>

<TD>

<xsl:value-of select="@EARNED\_RUNS"/>

</TD>

<TD>

<xsl:value-of select="@HIT\_BATTER"/>

</TD>

<TD>

<xsl:value-of select="@WILD\_PITCH"/>

</TD>

<TD><xsl:value-of select="@BALK"/></TD>

<TD>

<xsl:value-of select="@WALKED\_BATTER"/>

</TD>

<TD>

<xsl:value-of select="@STRUCK\_OUT\_BATTER"/>

</TD>

</TR>

```

</xsl:for-each><!-- PLAYER -->

</TBODY>

</TABLE>

</xsl:for-each><!-- TEAM -->

</xsl:for-each><!-- DIVISION -->

</xsl:for-each><!-- LEAGUE -->

</xsl:for-each><!-- SEASON -->

<HR></HR>

Copyright 1999

<A HREF="http://www.macfaq.com/personal.html">

Elliott Rusty Harold

</A>

<BR />

<A HREF="mailto:elharo@metalab.unc.edu">

elharo@metalab.unc.edu

</A>

</BODY>

</HTML>

</xsl:template>

</xsl:stylesheet>

```

#### 5.4.7 元素内容与 select 属性

本章集中讨论了使用 XSL 样式单格式化存储在一个元素属性中的数据，因为使用 CSS 无法访问属性。如果想要包含一个元素的字符数据而不是属性，XSL 同样做得很好。只要简单地把元素名称当作 xsl:value-of 元素的 select 属性值就能表明一个元素的文本将被复制到输出文档中。请看清单 5-8：

清单 5-8: greeting.xml

```

<?xml version="1.0" standalone="yes"?>

<?xml-stylesheet type="text/xsl" href="greeting.xsl"?>

```

```
<GREETING>
```

```
Hello XML!
```

```
</GREETING>
```

假如要向标题 H1 中复制致词 “Hello XML!” 首先，使用 `xsl:for-each` 选择 GREETING 元素：

```
<xsl:for-each select="GREETING">
```

```
<H1>
```

```
</H1>
```

```
</xsl:for-each>
```

只用这一段语句足以把两个 H1 标记复制到输出中。使用没有 `select` 属性的 `xsl:value-of` 在两个 H1 标记之间放置 GREETING 元素的文本，当前元素（GREETING）的内容就会被默认选中。清单 5-9 显示了完整的样式单。

清单 5-9: greeting.xsl

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

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

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

```
<HTML>
```

```
<BODY>
```

```
<xsl:for-each select="GREETING">
```

```
<H1>
```

```
<xsl:value-of/>
```

```
</H1>
```

```
</xsl:for-each>
```

```
</BODY>
```

```
</HTML>
```

```
</xsl:template>
```

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

使用 `select` 同样可以选择一个子元素中的内容，只需把该子元素的名称当作 `xsl:value-of` 的 `select` 属性值。例如，在上一章的棒球示例中，队员统计被存储在子元素而不是属性中。假定文档的结构是这样（事实上这种结构比本章中的基于属性的结构更常见），表示击球员表格的 XSL 如下所示：

```

<TABLE>

<CAPTION><B>Batters</B></CAPTION>

<THEAD>

<TR>

<TH>Player</TH><TH>P</TH><TH>G</TH>

<TH>GS</TH><TH>AB</TH><TH>R</TH><TH>H</TH>

<TH>D</TH><TH>T</TH><TH>HR</TH><TH>RBI</TH>

<TH>S</TH><TH>CS</TH><TH>SH</TH><TH>SF</TH>

<TH>E</TH><TH>BB</TH><TH>SO</TH><TH>HBP</TH>

</TR>

</THEAD>

<TBODY>

<xsl:for-each select="PLAYER [(POSITION
!= Starting Pitcher ) $and$(POSITION != Relief Pitcher )]">

<TR>

<TD>

<xsl:value-of select="GIVEN_NAME"/>

<xsl:value-of select="SURNAME"/>

</TD>

<TD><xsl:value-of select="POSITION"/></TD>

<TD><xsl:value-of select="GAMES"/></TD>

<TD>

<xsl:value-of select="GAMES_STARTED"/>

</TD>

<TD><xsl:value-of select="AT_BATS"/></TD>

<TD><xsl:value-of select="RUNS"/></TD>

```

```
<TD><xsl:value-of select="HITS"/></TD>

<TD><xsl:value-of select="DOUBLES"/></TD>

<TD><xsl:value-of select="TRIPLES"/></TD>

<TD><xsl:value-of select="HOME_RUNS"/></TD>

<TD><xsl:value-of select="RBI"/></TD>

<TD><xsl:value-of select="STEALS"/></TD>

<TD>

<xsl:value-of select="CAUGHT_STEALING"/>

</TD>

<TD>

<xsl:value-of select="SACRIFICE_HITS"/>

</TD>

<TD>

<xsl:value-of select="SACRIFICE_FLIES"/>

</TD>

<TD><xsl:value-of select="ERRORS"/></TD>

<TD><xsl:value-of select="WALKS"/></TD>

<TD>

<xsl:value-of select="STRUCK_OUT"/>

</TD>

<TD>

<xsl:value-of select="HIT_BY_PITCH"/>

</TD>

</TR>

</xsl:for-each><!-- PLAYER -->

</TBODY>
```

</TABLE>

在这种情况下，在每个 PLAYER 元素的子元素中，该元素的 GIVEN\_NAME、SURNAME、POSITION、GAMES、GAMES\_STARTED、AT\_BATS、RUNS、HITS、DOUBLES、TRIPLES、HOME\_RUNS、RBI、STEALS、CAUGHT\_STEALING、SACRIFICE\_HITS、SACRIFICE\_FLIES、ERRORS、WALKS、STRUCK\_OUT 和 HIT\_BY\_PITCH 子元素的内容被抽取出来并被复制到输出文档中。因为本章使用了与上一章 PLAYER 子元素名称相同的属性名，该示例与清单 5-7 几乎是一致的。主要差别是@符号没有了。它表明这是一个属性而不是一个元素。

select 属性的功能很多。可选择元素：按元素位置（例如第一、第二、最后、第十七个元素等等）；按特定的内容；按特殊的属性值；或者按照元素的父或子元素含有一定的内容或属性值进行选择。甚至可以使用全部布尔逻辑运算符来组合各种不同的选择条件。在 14 章的 XSL 中将要探讨使用 select 属性的更多可能。

#### 5.4.8 CSS 还是 XSL

CSS 与 XSL 在某种程度上是重复的。XSL 的功能确实比 CSS 更强大，但是 XSL 的功能与其复杂性是分不开的。这一章仅仅涉及了 XSL 最基本的用途。实际上 XSL 更复杂，而且比 CSS 更难学习和使用，同时也带来了一个问题：“什么时候应该使用 CSS，什么时候应该使用 XSL？”

CSS 比 XSL 得到更广泛的支持。部分 CSS Level 1 被 Netscape 4 和 Internet Explorer 4 支持作为 HTML 元素（尽管存在一些令人头疼的区别）；此外，Internet Explorer 5.0 和 Mozilla 5.0 能很好支持可以同时用于 XML 和 HTML 的大部分 CSS Level 1 的内容和一些 CSS Level 2 的内容。因此，选择 CSS 会与更广泛的浏览器相互兼容。

另外，CSS 更成熟一些，CSS Level 1（包含目前为止我们已经看到的大部分 CSS 内容）和 CSS Level 2 是 W3C 的推荐规范。XSL 仍然是一个早期的工作草案，而且直到本书出版后也不会最终定型。早期的 XSL 采纳者曾经接受过考验，而且将在形式统一的标准之前接受再一次的考验。选择 CSS 意味着无须为了追随软件 and 标准的发展不停地重写自己的样式单。但是，XSL 将最终形成一个可用的标准。

因为 XSL 是一种新事物，不同的软件实现方式不同，实现的是草案标准的不同的子集。在写作本书的 1999 年春天至少有三种主要不同形式的 XSL 在广泛应用，到本书出版前将会有更多。如果当前浏览器中不完善的 CSS 操作已经让人头疼的话，那么众多的 XSL 变种就会使人发疯。

但是，XSL 的功能很明显比 CSS 强大。CSS 仅允许格式化元素内容，不允许改变或重新安排这些内容，必须根据元素的内容或属性为元素选择不同的格式化方式或者增添诸如署名之类简单、额外的文本。XSL 非常适用于 XML 文档仅包含最少的数据，并且数据周围没有 HTML 装饰的情况。

使用 XSL 能够从页面上分离出关键数据，如刊头、向导栏和署名等。使用 CSS 不得不在数据文档中包含全部这些项目。XML+XSL 允许数据文档与 Web 页面文档分离单独存在，从而使得 XML+XSL 文档更容易维护和处理。

XSL 终将成为现实世界和大量数据应用的最佳选择，CSS 更适合于简单的页面，如祖母用于向她们孙子寄送图片的页面。但对于这些用途，HTML 已经足够。如果使用 HTML 行不通，XML+CSS 不会有多大的帮助。相较而言，XML+XSL 能够解决更多 HTML 不能解决的困难。对于传统的浏览器来说，仍然需要 CSS，但长远看来使用 XSL 才是发展方向。



## 5.5 本章小结

在本章中，读者看到了从头创建的 XML 文档的示例。特别是学到如下内容：

- 信息可以保存在元素的属性中。
- 属性是包含在元素起始标记中的一个名字-数值对。
- 属性主要用来保存关于元素的元信息，而不是元素的数据。
- 属性比元素内容更不便处理。
- 对于非常简单并且不随文档改变其形式的信息，使用属性较好。特别是样式信息和链接信息，作为属性执行起来很顺利。
- 空标记给没有内容的元素提供了句法修饰。
- XSL 是一种功能强大的样式单语言，使我们能够访问和显示属性数据和转换文档。

下一章将详细介绍结构完整的 XML 文档必须严格遵循的规则。我们还将研究另外一些在 XML 文档中嵌入信息如注释和处理命令的方法。

## 第 6 章 结构完整的 XML 文档

HTML 4.0 有大约 100 个不同的标记，大部分标记都有多个可能的属性用于几百种不同的变化。因为 XML 的功能比 HTML 强大，你也许认为需要懂得更多标记，但不是这样。XML 凭借其简洁性和可扩展性具有强大的功能，并不是大量的标记。

事实上，XML 几乎没有预先定义任何标记，相反允许用户需要时定义自己的标记。但是由自定义标记建立的这些标记和文档并不是随意的，必须遵循一组特定的规则，本章将详细阐述这些规则。遵守这些规则的文档被认为是结构完整的。结构完整是 XML 处理器和浏览器阅读文件必要的最起码的标准。本章将阐述用于结构完整的 XML 和 HTML 文档的规则。请特别注意 XML 与 HTML 的区别。

本章的主要内容包括：

- XML 文档的组成
- 置标和字符数据
- 独立文档中的结构完整的 XML
- 结构完整的 HTML

### 6.1 XML 文档的组成

XML 文档包含由 XML 标记和字符数据组成的文本。它是一个有固定长度的有序字节的集合，并遵守特定的约束。它可能是或者不是一个文件。例如，XML 文档可能：

- 存储在数据库中
- 由 CGI 程序在内存中瞬间创建的
- 由几个相互嵌套的不同文件组合而成
- 不存在于自身的文件中

但是如果把一个 XML 文档看作一个文件也是可以的，只要记住它可能并不是存在于硬盘上的真实文件。

XML 由称为“实体”的存储单元组成，每个实体包含文本或者二进制数据，但不能同时存在。文本数据由字符组成，二进制数据用于图片和小程序等类内容。用一个具体的示例说明就是，一个含有<IMG>标记的原始 HTML 文件是一个实体而不是文档。一个 HTML 文件加上所有使用<IMG>标记嵌入的图片就组成一个文档。

在本章和后续几章中我们只针对由一个实体构成的简单的 XML 文档，即文档本身。而且这些文档只包含文本数据，不包含诸如图片小程序一类的二进制数据。这些文档能够完全独立被理解而无需读取其他文件。换句话说，它们是独立存在的。这种文档通常在它的 XML 标头中含有一个值为 yes 的 standalone 属性，如下所示：

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

外部实体和实体引用用于组合多个文件和其他数据源以创建一个独立的 XML 文档。这样的文档如果不引用其他文件就不能进行句法分析。这些文档通常在 XML 声明中含有一个属性值为 no 的 standalone 属性：

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



外部实体及实体引用将在第 9 章“实体与外部 DTD 子集”中讨论。

## 6.2 置标和字符数据

XML 文档是文本。文本由字符组成。字符是字母、数字、标点符号、空格、制表符号或类似的东西。XML 使用 Unicode 字符集（统一的字符编码标准集），它不仅包含来自英语和其他西欧字母表中的常见字母和符号，也包含来自古斯拉夫语、希腊语、希伯来语、阿拉伯语和梵语的字母表。另外还包含汉语和日语的象形汉字和韩国的 Hangul 音节表。在本章中只使用英语文本。



国际化字符集将在第 7 章“外语和非罗马文本”中讨论。

一个 XML 文档的文本可有两种用途，字符数据和置标。字符数据是文档的基本信息。另一方面，置标主要描述一个文档的逻辑结构。例如，回想一下第三章清单 3-2 中的 `greeting.xml`：

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

```
<GREETING>
```

```
    Hello XML!
```

```
</GREETING>
```

其中 `<?xml version="1.0" standalone="yes"?>`、`<GREETING>` 和 `</GREETING>` 是置标。`Hello XML!` 是字符数据。XML 比其他格式优越的一点是它把实际数据与置标明显地分隔开。

更确切地说，置标包括所有的注释、字符引用、实体引用、CDATA 段定界符、标记、处理指令和 DTD。其他的就是字符数据。但是文档被处理后，一些置标会变成字符数据。例如，置标 `&gt;` 变成了大于号 (`>`)。文档经处理后留下的字符数据和所有的代表特定字符的数据称为可分析的字符数据。

### 6.2.1 注释

XML 的注释与 HTML 的注释很相似，它们以 `<!--` 开始，以 `-->` 结束。介于 `<!--` 和 `-->` 之间的全部数据均被 XML 处理器忽略，就像它们根本不存在一样。注释用于提醒自己或临时标注出文档中不完善的部分。例如：

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

```
<!--This is Listing 3-2 from The XML Bible-->
```

```
<GREETING>
```

```
    Hello XML!
```

```
<!--Goodbye XML-->
```

```
</GREETING>
```

在使用注释时必须遵循以下几条规则，大致如下：

1. 注释不能出现在 XML 声明之前，XML 声明必须是文档最前面的部分。例如，下面这种情况是不允许的：

```
<!--This is Listing 3-2 from The XML Bible-->
```

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

```
<GREETING>
```

```
Hello XML!
```

```
<!--Goodbye XML-->
```

```
</GREETING>
```

2. 注释不能放在标记中，例如：下面这种情况是非法的：

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

```
<GREETING>
```

```
Hello XML!
```

```
</GREETING <!--Goodbye--> >
```

3. 注释可以包围和隐藏标记。在下面例子中，<antigreeting>标记及其内容被当作注释；而且文档在浏览器中显示时不会出现，好像不存在一样：

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

```
<DOCUMENT>
```

```
<GREETING>
```

```
Hello XML!
```

```
</GREETING>
```

```
<!--
```

```
<ANTIGREETING>
```

```
Goodbye XML!
```

```
</ANTIGREETING>
```

```
-->
```

```
</DOCUMENT>
```

由于注释有效地删除了文本的一些部分，必须保证剩余的文本仍然是一个结构完整的 XML 文档。例如，在没有注释掉相应的结束标记前千万不要注释掉起始标记。例如，下面的语句是非法的：

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

```
<GREETING>
```

Hello XML!

<!--

</GREETING>

-->

一旦删除注释文本，剩余的是：

<?xml version="1.0" standalone="yes"?>

<GREETING>

Hello XML!

因为<GREETING>标记没有与之匹配的结束标记</GREETING>，这已经不再是一个结构完整的 XML 文档。

4. 两个连字符号（--）除了作为注释起始和结束标记的一部分外，不能出现在该注释中。例如，下面的是非法注释：

<!--The red door--that is, the second one--was left open-->

这意味着不能像下面的语句这样嵌套注释：

<?xml version="1.0" standalone="yes"?>

<DOCUMENT>

<GREETING>

Hello XML!

</GREETING>

<!--

<ANTIGREETING>

<!--Goodbye XML!-->

</ANTIGREETING>

-->

</DOCUMENT>

这也意味着如果注释掉带有表达式如 `i--` 或 `numberLeft--` 的 C、Java 或者 JavaScript 源代码时就会出现这个问题。通常只要意识到这个问题就不难解决。

### 6.2.2 实体引用

实体引用是指分析文档时会被字符数据取代的置标。XML 预先定义了 5 个实体引用，列在表 6-1 中。实体引用用于 XML 文档中的特殊字符，否则这些字符将被解释为置标的组成部分。例如，实体引用&lt; 代表小于号（<），否则会被解释为一个标记的起始部分。

表 6-1 XML 预定义的实体引用

实体引用	字 符
&amp;	&
&lt;	<
&gt;	>
&quot;	"
&apos;	'



XML 中的实体引用与 HTML 中不同，必须以一个分号结束。因此&gt;是正确的实体引用写法，&gt 是不正确的。

未经处理的小于号（<）同表示“和”的符号（&）在一般的 XML 文本中往往被分别解释为起始标记和实体引用（特殊文本是指 CDATA 段，将在后面讨论）。因此，小于号同“和”号必须分别编码为&lt;和&amp;。例如，短语“Ben & Jerry s New York Super Fudge Chunk Ice Cream”应当写成 Ben &amp;Jerry s New York Super Fudge Chunk Ice Cream 。

大于号、双引号和撇号在它们可能会被解释成为置标的一部分时也必须编码。但是，养成全部编码的习惯要比努力推测一个特定的应用是否会被解释为置标容易得多。

实体引用也能用于属性值中。例如：

```
<PARAM NAME="joke" VALUE="The diner said,  
"Waiter,There&apos;s a fly in my soup!">  
  
</PARAM>
```

6.2.3 CDATA

大多数情况下，出现在一对尖括号（<>）中的是置标，不在尖括号中的是字符数据。但是有一种情况例外，在 CDATA 段中所有文本都是纯字符数据。看起来与标记或者实体相似的仅仅是它们各自相应的文本。XML 处理器无论如何是不会解释它们的。

CDATA 段用于需要把整个文本解释为纯字符数据而并不是置标的情况。当有一个包含许多<、>、&或"字符而非置标的庞大文本时，这是非常有用的。对于大部分 C 和 Java 源代码，正是这种情况。

如果想使用 XML 写有关 XML 的简介，CDATA 段同样非常有效。例如，在本书中包含许多小的 XML 代码块，而我正在使用的字处理器又不能顾及这些情况。但是如果把本书转换为 XML，我将不得不很辛苦地用&lt;代替全部小于号，&amp;代替所有“和”字符。如下所示：

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

```
<GREETING>
```

```
Hello XML!
```

```
</GREETING>
```

为了避免这种麻烦，可以使用一个 CDATA 段表示一个不需翻译的文本块。CDATA 段以<![CDATA[ 开始并以 ]]>结束，例如：

```
<![CDATA[
```

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

```
<GREETING>
```

```
Hello XML!
```

```
</GREETING>
```

```
]]>
```

唯一不许出现在 CDATA 段中的文本是 CDATA 的结束界定符]]>。注释可能会出现在 CDATA 段中，但不再扮演注释的角色。也就是说两个注释标记和包含在它们之间的全部文本都将显示出来。



因为]]>不能出现在 CDATA 段中，所以 CDATA 段不能嵌套。这使得使用 XML 写有关的 CDATA 段相对困难些。如果需要的话，必须去掉项目符号，并使用<lt;、&amp;和实体引用。

CDATA 段不常需要，一旦需要时，它是非常有用的。

## 6.2.4 标记

置标能够区分 XML 文件与无格式文本文件。置标的最大部分是标记。前一章讲隼吮县塹氛褂梅椒ǎ 窘谡 丁灞县遣(8)崩 褂梅椒ā?/p>

简而言之，标记在 XML 文档中以<开始，以>结束，而且不包含在注释或者 CDATA 段中。因此，XML 标记有与 HTML 标记相同的形式。开始或打开标记以<开始，后面跟有标记名。终止或结束标记以</开始，后面也跟标记名。遇到的第一个>该标记结束。

### 6.2.4.1 标记名

每个标记都有一个名称。标记名必须以字母或下划线（\_）开始，名称中后续字符可以包含字母、数字、下划线、连字符和句号。其中不能含有空格（经常用下划线替代空格）。下面是一些合法的 XML 标记：

```
<HELP>
```

```
<Book>
```

```
<volume>
```

<heading1>

<section.paragraph>

<Mary\_Smith>

<\_8ball>



冒号出现在标记名中从语法上讲是合法的，但是它们被保留用于命名域。命名域可以混合和匹配可能使用同名标记的标记集合。命名域将在第 18 章讨论。

以下是句法不正确的 XML 标记：

<Book%7>

<volume control>

<lheading>

<Mary Smith>

<.employee.salary>



事实上标记名的规则也适用于其他许多名称，如属性名、ID 属性值、实体名和其他一些将在后面几章遇到的结构。

结束标记与起始标记同名，只是在起始尖括号后加了一个/。例如，如果起始标记是<F00>，那么结束标记是</F00>。下面是前面所提到的合法起始标记所对应的结束标记：

</HELP>

</Book>

</volume>

</heading1>

</section.paragraph>

</Mary\_Smith>

</\_8ball>

XML 名称是大小写敏感的。在 HTML 中的<P>和<p>是同一个标记，</p>可以结束一个<P>标记，但在 XML 中却不行。下面所示的并不是我们讨论过的合法起始标记所对应的结束标记：

</help>



</book>

</Volume>

</HEADING1>

</Section.Paragraph>

</MARY\_SMITH>

</\_8BALL>

尽管大小写字母均可以用在 XML 的标记中，从此观点出发，我会尽可能遵循使用大写的约定。这主要是因为大写在本书中可以更突出，但是有时使用的标记集是别人建立的，那么采用别人的习惯约定是必要的。

#### 6.2.4.2 空标记

许多不含数据的 HTML 标记没有结束标记。例如，在 HTML 中没有</LI>、</IMG>、</HR>或</BR>标记。一些页面作者在所列的项目后面确实会包含</LI>标记，一些 HTML 工具也使用</LI>标记。但是 HTML 4.0 标准特别否认了这一点的必要性。同 HTML 中所有没有被公认的标记一样，一个不必要的</LI>的出现对交付的输出没有任何影响。

这在 XML 中不是问题。XML 的总体观点就是在分析文档时允许发现新的标记。因此没有识别的标记就不会被简单地忽略。而且 XML 处理器一定能够判明以前从没出现过的一个标记有没有结束标记。

XML 区分带有结束标记的标记，而不带结束标记的标记称为空标记。空标记以斜杠和一个结束尖括号（/>）结束。例如，<BR/>或<HR/>。

目前的 Web 浏览器处理这种标记的方法不一致，如果希望保持向后的兼容性，可以用结束标记来代替，只要在两个标记之间不包含任何文本。例如：

<BR></BR>

<HR></HR>

<IMG></IMG>

在学了后续几章中的 DTD 和样式单后，将会看到在必须由传统浏览器分析的文档中使用 HTML 可以有多种方法保持向前和向后的兼容性。

#### 6.2.5 属性

在前面的章节中讨论过，起始标记和空标记可以随意地包含属性。属性是用等号（=）分隔开的名称-数值对。例如：

<GREETING LANGUAGE="English">

Hello XML!

<MOVIE SRC="WavingHand.mov"/>

</GREETING>

在此<GREETING>标记有一个 LANGUAGE 属性，其属性值是 English。<MOVIE>标记有一个 SRC 属性，其属性值为 WavingHand. mov。

#### 6.2.5.1 属性名

属性名是字符串，遵循与标记名同样的规则。这就是，属性名必须以字母或下划线（\_）开始，名称中后续字符可以包含字母、数字、下划线、连字符和句号。其中不能含有空格（经常用下划线替代空格）。

同一个标记不能有两个同名的属性。例如，下面的例子是不合法的：

```
<RECTANGLE SIDE="8cm" SIDE="10cm"/>
```

属性名是区分大小写的。SIDE 属性与 side 或者 Side 属性不是同一个属性，因此以下例子是合法的：

```
<BOX SIDE="8cm" side="10cm" Side="31cm"/>
```

但是上面的这种写法很迷惑人，最好不要这样书写。

#### 6.2.5.2 属性值

属性值也是字符串。如下面所示的 LENGTH 属性，即使字符串表示的是一个数，但还是两个字符 7 和 2，不是十进制数的 72。

```
<RULE LENGTH="72"/>
```

如果编写处理 XML 的代码，在对字符串执行算术运算之前必须把它们转换为一个数。

与属性名不同，对属性值包含的内容没有任何限制。属性值可以包含空格，可以以一个数字或任何标点符号（有时单括号和双括号除外）开头。

XML 属性值由引号界定。与 HTML 属性不同，XML 属性值必须加引号。大多数情况下是使用双引号，但是如果属性值本身含有一个引号，就需要使用单引号。例如：

```
<RECTANGLE LENGTH= 7" WIDTH= 8.5" />
```

如果属性值中含有两种引号，那么其中不用于界定字符串的一个必须用合适的实体引用代替。我通常替换两个，这种方法很管用。例如：

```
<RECTANGLE LENGTH= 8&apos;7&quot; WIDTH="10&apos;6&quot;"/>
```

## 6.3 独立文档中结构完整的 XML

尽管可以根据需要编写标记，XML 文档为了保持结构完整必须遵循一定的规则。如果一个文档不是结构完整的，大部分读取和显示操作都会失败。

事实上，XML 规范严格禁止 XML 句法分析器分析和解释结构欠妥的文档。正在执行操作的分析器唯一能做的是报告出错。它不会修改错误，不会作最大的努力显示作者想要的东西，也不会忽略不当的结构欠妥的标记。它所能做的是报告错误和退出。

这样做的目的是为了对错误的兼容性的竞争。这种竞争已使得编写 HTML 语法分析程序和显示程序变得非常困难。因为 Web 浏览器承认畸形的 HTML，而 Web 页面设计者不会特别尽力确保他们的 HTML 正确无误。事实上，他们甚至利用个别浏览器中的错误达到特殊的效果。为了正确显示被大量安装的 HTML 页面，每个新的 Web 浏览器必须支持已有的 Web 浏览器的每一个细微差别和各自的属性。用户将放弃任何一种严格执行 HTML 标准的浏览器。正是为了避免这种遗憾，XML 处理器才只接受结构完整的 XML。

为了使一个文档结构完整，XML 文档中的所有置标和字符数据必须遵守前几节中给出的规则。而且有几条关于如何把置标和字符数据相互联系起来的规则。这些规则总结如下：

1. 文档的开始必须是 XML 声明。
2. 含有数据的元素必须有起始标记和结束标记。
3. 不含数据并且仅使用一个标记的元素必须以 /> 结束。
4. 文档只能包含一个能够包含全部其他元素的元素。
5. 元素只能嵌套不能重叠。
6. 属性值必须加引号。
7. 字符 < 和 & 只能用于起始标记和实体引用。
8. 出现的实体引用只有 &amp;、&lt;、&gt;、&apos; 和 &quot;。

这八条规则稍加调整就能适用于含有一个 DTD 的文档，而且对于定义文档与它的 DTD 之间关系的完整性有另外的规则。我们将在后面几章中介绍。现在请仔细看这些用于没有 DTD 文档的规则。



DTD 将在本书第二部分中讨论。

### #1: 文档必须以 XML 声明开始

下面是 XML 1.0 中独立文档的 XML 声明：

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

如果声明出现，它绝对是该文件最开头部分，因为 XML 处理器会读取文件最先的几个字节并将它与字符串 <?XML 的不同编码作比较来确定正在使用的字符串集（UTF-8、大头（高字节先传格式）或者小头（低字节先传格式））。除去看不见的字节顺序记号，在它之前不能有任何东西，包括空格。例如，下面一行用于 XML 的开始是不能接受的，因为在该行的前面有多余的空白。

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

UTF-8 和 Unicode 的变种在第 7 章“外语和非罗马文本”中讨论。

XML 确实允许完全省略 XML 声明。通常不推荐这样做，但这样做有特殊的用途。例如，省略 XML 声明，通过连接其他结构完整的 XML 文档有助于重新建立一个结构完整的 XML 文档。这种方法将在第 9 章讨论。而且，本章后面将要讲述的一种样式能够编写结构完整的 HTML 文档。

## #2: 在非空标记中使用起始和结束标记

如果忘了结束 HTML 的标记，Web 浏览器并不严格追究。例如，如果文档包含一个<B>标记却没有相应的</B>标记，在<B>标记之后的全部文档将变为粗体。但文档仍然能显示。

XML 不会如此宽松，每个起始标记必须以相应的结束标记结束。如果一个文档未能结束一个标记，浏览器或移交器将报告一个错误信息，并且不会以任何形式显示任何文档的内容。

## #3: 用“/”结束空标记

不包含数据的标记，例如 HTML 的<BR>、<HR>和<IMG>，不需要结束标记。但是 XML 空标记必须由/>结束，而不是>。例如<BR>、<HR>和<IMG>的 XML 等价物是<BR/>、<HR/>和<IMG/>。

当前的 Web 浏览器处理这种标记的方法不一致。但是如果保持向后的兼容性，可以使用结束标记来代替，而且不能在其间包含任何文本。例如：

```
<BR></BR>
```

```
<HR></HR>
```

```
<IMG></IMG>
```

即使这样，Netscape 处理<BR></BR>也有困难（它把这两个标记解释为行间距，而不是前面所讲的）。因此，在 HTML 中包含结构完整的空标记也并非总是可行的。

## #4: 让一个元素完全包含其他所有元素

一个 XML 文档包含一个根元素，它完全包含了文档中其他所有元素。有时候这种元素被称作文档元素。假设根元素是非空的（通常都是如此），它肯定有起始标记和结束标记。这些标记可能使用但不是必须使用 root 或 DOCUMENT 命名。例如，在下面的文档中根元素是 GREETING：

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

```
<GREETING>
```

```
Hello XML!
```

```
</GREETING>
```

XML 声明不是一个元素，它更像是一个处理指令，因此不必包含在根元素中。类似地，在一个 XML 文档中的其他非元素数据，诸如其他处理指令、DTD 和注释也不必包含在根元素中。但是所有实际的元素（除根元素本身）必须包含在根元素中。

## #5: 不能重叠元素

元素可以包含别的元素（大多数情况下），但是元素不能重叠。事实上是指，如果一个元素含有一个起始标记，则必须同时含有相应的结束标记。同样，一个元素不能只含有一个结束标记而没有相应的起始标记。例如，下面的 XML 是允许的：

```
<PRE><CODE>n =n +1;</CODE></PRE>
```

下面所示的 XML 是非法的，因为结束标记</PRE>放在了结束标记</CODE>之前：

```
<PRE><CODE>n =n +1;</PRE></CODE>
```

大部分 HTML 浏览器容易处理这种情况，但是 XML 浏览器会因为这种结构而报告出错。

空标记可随处出现。例如：

```
<PLAYWRIGHTS>Oscar Wilde<HR/>Joe Orton</PLAYWRIGHTS>
```

本规则与规则 4 联系在一起有如下含义：对于所有非根元素，只能有一个元素包含某一非根元素，但是元素不能包含其中含有非根元素的元素。这个直接含有者称为非根元素的父元素，非根元素被认为是父元素的子元素。因此，每个非根元素只有一个父元素。但是一个单独的元素可以有任意数目的子元素或不含有子元素。

请分析如下所示的清单 6-1。根元素是 DOCUMENT 元素，它含有两个元素。第一个 STATE 元素含有 4 个子元素：NAME、TREE、FLOWER 和 CAPITOL。第二个 STATE 元素含有 3 个子元素：NAME、TREE 和 CAPITOL。这些里层的子元素只包含字符数据，没有子元素。

清单 6-1：父元素和子元素

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

```
<DOCUMENT>
```

```
<STATE>
```

```
<NAME>Louisiana</NAME>
```

```
<TREE>Bald Cypress</TREE>
```

```
<FLOWER>Magnolia</FLOWER>
```

```
<CAPITOL>Baton Rouge</CAPITOL>
```

```
</STATE>
```

```
<STATE>
```

```
<NAME>Mississippi</NAME>
```

```
<TREE>Magnolia</TREE>
```

```
<CAPITOL>Jackson</CAPITOL>
```

```
</STATE>
```

</DOCUMENT>

在编程人员的术语中，这意味着 XML 文档形成了一个树。图 6-1 显示了清单 6-1 表示的树形结构以及将该结构称为树的原因。图 6-1 从根开始，逐级地分支延伸到树末端的叶。

树有一些好的特性使计算机程序易于读取，尽管对于文档的作者而言是无关紧要的。

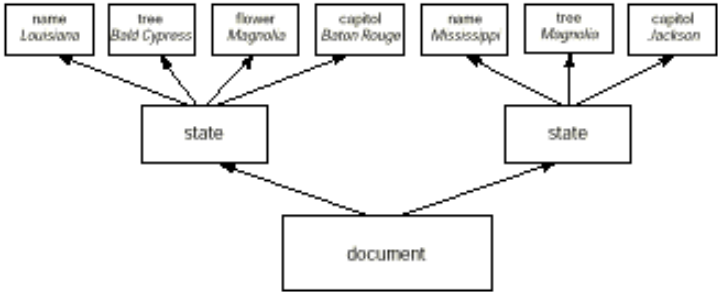


图 6-1 清单 6-1 表示的树形结构



树通常由上向下画，这就是说树的根应该在图片的顶部而不是底部。但这样看起来不像真正的树，不过并不影响数据结构的拓扑形式。

### #6: 属性值必须加引号

XML 要求所有的属性值必须加引号，不管属性值是否包括空白。例如：

```
<A HREF="http://metalab.unc.edu/xml/">
```



HTML 的要求则不是这样。比如，HTML 允许标记含有不带引号的属性。例如，下面是一个合法的 HTML<A>标记：

```
<A HREF=http://metalab.unc.edu/xml/>
```

唯一的要求是属性值不能嵌有空格。

如果一个属性值本身含有双引号，可以使用属性值加单引号来代替。例如：

```
<IMG SRC="sistinechapel.jpg"ALT= And God said,"Let there be light," and there was light />
```

如果一个属性值包含有单引号和双引号，可以使用实体引用&apos;代替单引号，&quot;代替双引号。例如：

```
<PARAM name="joke" value="The diner said,  
&quot;Waiter,There&apos;s a fly in my soup!&quot;">
```

### #7: 只在开始标记和实体引用中使用<和&

XML 假定最先的<是一个标记的开始，&是一个实体引用的开始（HTML 也是如此，如果省略它们，大部分浏览器会假定有一个分号）。例如：

<H1>A Homage to Ben &Jerry s

New York Super Fudge Chunk Ice Cream</H1>

Web 浏览器会正确地显示该标记，但是为了最大限度的安全，应当避免使用&，用&amp;来代替，像下面这样：

<H1>A Homage to Ben &amp;Jerry s New York Super Fudge Chunk

Ice Cream</H1>

开尖括号（<）的情况也类似。请看下面很普通的一行 Java 代码：

```
<CODE> for (int i =0;i <=args.length;i++) {</CODE>
```

XML 与 HTML 都会把<=中的小于号当作一个标记的开始。该标记会延续到下一个>。因此该行会现示成：

```
for (int i =0;i
```

而不是：

```
for (int i =0;i <=args.length;i++) {
```

“=args.length;i++) {” 被解释成一个不能识别的标记的一部分。

把小于号写成&lt;可以出现在 XML 和 HTML 文本中。例如：

```
<CODE> for (int i =0;i &lt;=args.length;i++) {</CODE>
```

结构完整的 XML 要求把&写成&amp;，把<写成&lt;，只要不是作为标记或者实体的一部分时都应如此。

## #8：只能使用现有的 5 个实体引用

读者可能已经熟悉了几个 HTML 中的实体引用，例如&copy;为插入版权号，&reg 为插入注册商标号。但是除了已经讨论过的五个实体引用，XML 只能使用预先在 DTD 中定义过的实体引用。

但是现在读者可能还不了解 DTD，如果与字符&出现在文档中的任何地方，其后必须紧跟 amp;、lt;、gt;、apos;或者 quot;。所有其他的用法都会破坏结构完整性。



在第 9 章“实体和外部 DTD 子集”中将会学习如何使用 DTD 定义插入特殊符号和大块样板文本的新实体引用。

## 6.4 结构完整的 HTML

即使在大部分 Web 浏览器还不能直接支持 XML 的情况下，也可以通过编写结构完整的 HTML 来练习 XML 技巧。这就是遵守 XML 的结构完整性约束，但只是使用标准的 HTML 标记的 HTML。结构完整的 HTML 比多数人和 FrontPage 等工具编写的非标准的 HTML 容易读取，同时容易被 Web 机器人中用动搜索引擎理解。它更为强健，对它作一些改动不会破坏它，在转移的过程中不会因为变换浏览器和操作平台对它产生影响。而且可以使用 XML 工具服务于 HTML 文档，这对于那些服务器不支持 XML 的读者来说仍然保持了向后的兼容性。

### 6.4.1 现实的 Web 页面存在的问题

真正的 Web 页面非常不标准。没有结束标记，元素重叠，在页面中包含未经处理的小于号，忽略实体引用后面的分号。存在这些问题的 Web 页面严格来讲是无效的，但部分浏览器能接受它们。不过如果校正了这些问题，页面将会更整洁，显示更快，更容易维护。

Web 页面包含的一些常见的问题：

1. 起始标记没有对应的结束标记（没有结束元素）
2. 结束标记没有相应的起始标记
3. 元素重叠
4. 属性值未加引号
5. 没有避免使用 <、>、& 和 " 符号
6. 没有根元素
7. 结束标记与起始标记不匹配

清单大致按照其重要性排列，但确切的细节因标记不同而变化。例如，没有被结束的 <STRONG> 标记会把跟随其后的所有元素变为粗体。但是没有被结束的 <LI> 或者 <P> 标记不会引发任何问题。

有几条规则仅适用于 XML 文档，如果试图把它们汇集在已存在的 HTML 页面中，确实会带来问题。这些规则有：

1. 以一个 XML 声明开始
2. 空标记必须以 /> 结束
3. 使用的实体引用只有 &amp;、&lt;、&gt;、&apos; 和 &quot;。

校正这些问题并不难，只是有几个稍不注意就会出现。下面让我们仔细加以研究。

#### 6.4.1.1 结束所有的起始标记

任何含有内容的元素，无论是文本还是别的子元素，应该有一个起始标记和结束标记。HTML 不绝对要求这样做。例如 <P>、<DT>、<DD> 和 <LI> 经常被单独使用。但是，这样做主要依靠 Web 浏览器能够很好地判断一个元素的结束位置，浏览器并不总能确切地按照作者的意愿去做。因此最好是明确地结束所有起始标记。

对于编写 HTML 的方法，这里要求对其所作的最大改变是把 <P> 看作一个容器，而不是一个简单的段落分界符。例如，以前格式化 Federalist Papers 的开始部分，如下所示：

To the People of the State of New York:

<P>

AFTER an unequivocal experience of the inefficiency of the



subsisting federal government, you are called upon to  
deliberate on a new Constitution for the United States of  
America. The subject speaks its own importance; comprehending  
in its consequences nothing less than the existence of the  
UNION, the safety and welfare of the parts of which it is  
composed, the fate of an empire in many respects the most  
interesting in the world. It has been frequently remarked that  
it seems to have been reserved to the people of this country,  
by their conduct and example, to decide the important question,  
whether societies of men are really capable or not of  
establishing good government from reflection and choice, or  
whether they are forever destined to depend for their political  
constitutions on accident and force. If there be any truth in  
the remark, the crisis at which we are arrived may with  
propriety be regarded as the era in which that decision is to  
be made; and a wrong election of the part we shall act may, in  
this view, deserve to be considered as the general misfortune  
of mankind.

<P>

结构完整性要求将上面语句格式化为：

<P>

To the People of the State of New York:

</P>

<P>

AFTER an unequivocal experience of the inefficiency of the  
subsisting federal government, you are called upon to

deliberate on a new Constitution for the United States of America. The subject speaks its own importance; comprehending in its consequences nothing less than the existence of the UNION, the safety and welfare of the parts of which it is composed, the fate of an empire in many respects the most interesting in the world. It has been frequently remarked that it seems to have been reserved to the people of this country, by their conduct and example, to decide the important question, whether societies of men are really capable or not of establishing good government from reflection and choice, or whether they are forever destined to depend for their political constitutions on accident and force. If there be any truth in the remark, the crisis at which we are arrived may with propriety be regarded as the era in which that decision is to be made; and a wrong election of the part we shall act may, in this view, deserve to be considered as the general misfortune of mankind.

</P>

你以前学过的可能是把<P>看作一个段落的结束，现在应当把它看作一个开始。这会带来一些好处，例如可以方便地为一段落指定多种格式化属性。例如，下面是可在 <http://thomas.loc.gov/home/hres581.html> 上看到的 House Resolution 581 的原始 HTML 标题：

<center>

<p><h2>House Calendar No. 272</h2>

<p><h1>105TH CONGRESS 2D SESSION H. RES. 581</h1>

<p>[Report No. 105-795 ]

<p><b>Authorizing and directing the Committee on the Judiciary to investigate whether sufficient grounds

exist for the impeachment of William Jefferson Clinton,

President of the United States.</b>

</center>

下面是同样的文本，但使用的是结构完整的 HTML。Align 属性代替了相应的 center 元素，并且使用 CSS 样式属性代替了 <b> 标记。

<h2 align="center">House Calendar No.272</h2>

<h1 align="center">105TH CONGRESS 2D SESSION H.RES.581</h1>

<p align="center">[Report No.105-795 ]</p>

<p align="center" style="font-weight:bold">

Authorizing and directing the Committee on the Judiciary to

investigate whether sufficient grounds exist for the

impeachment of William Jefferson Clinton,

President of the United States.

</p>

#### 6.4.1.2 删除孤立的结束标记并且不要使元素重叠

在编辑页面时，删除一个起始标记而忘了删除相应的结束标记，这种情况很常见。在 HTML 中，一个孤立的结束标记如 </STRONG>或者</TD>没有任何相匹配的起始标记不会引发问题。但是这样会使文件比需要的更长，下载速度变慢，而且潜在地使人或工具理解和编辑 HTML 源文件发生混淆。因此应当确保每个结束标记都有正确的起始标记。

但是结束标记没有任何起始标记往往意味着那些元素错误地重叠了。在 Web 页面上的大部分重叠元素很容易修改。例如下面这种常见的错误：

<B><I>This text is bold and italic</B></I>

I 元素在 B 元素中开始，也必须在 B 元素中结束。需要做的只是交换两个结束标记的位置：

<B><I>This text is bold and italic</I></B>

同样可以交换两个起始标记：

<I><B>This text is bold and italic</B></I>

偶尔会遇到一个棘手的问题。例如，下面是来自白宫主页的一个片段（<http://www.whitehouse.gov/>，1998 年 11 月 4 日）。其中已醒目地标出了有问题的标记，很容易看出错误所在：

<TD valigr=TOP width=85>

```

<FONT size=+1>

<A HREF="/WH/New"></A><br></TD>

<TD valign=TOP width=225>

<A HREF="/WH/New"><B>What ' s New:</B></A><br>

</FONT>

What' s happening at the White <nobr>House -</nobr><br>

<font size=2><b>

<!-- New Begin -->

<a href="/WH/New/html/19981104-12244.html">Remarks Of The

President Regarding Social Security</a>

<BR>

<!-- New End -->

</font>

</b>

</TD>

```

其中，<FONT size=+1>元素在第一个<TD valign=TOP width=85>元素中开始，但是它的后续部分越过该元素结束于另一个<TD valign=TOP width=225>元素中。在此情况下，正确的处理方法是在第一个</TD>结束标记之前立即结束<FONT size=+1>起始标记，然后在第二个 TD 元素开始之后立即添加一个新的<FONT size=+1>起始标记，如下所示：

```

<TD valign=TOP width=85>

<FONT size=+1>

<A HREF="/WH/New"></A><br>

</FONT></TD>

<TD valign=TOP width=225>

```

```

<FONT size=+1>

<A HREF="/WH/New"><B>What ' s New:</B></A><br>

</FONT>

What ' s happening at the White <nobr>House -</nobr><br>

<font size=2><b>

<!-- New Begin -->

<a href="/WH/New/html/19981104-12244.html">Remarks Of The

President Regarding Social Security</a>

<BR>

<!-- New End -->

</font>

</b>

</TD>

```

#### 6.4.1.3 给所有属性加引号

HTML 属性只有在含有空格时才需要加引号，即使含有引号对它也并无妨碍。而且使用引号有助于以后将属性值修改为含有空格的属性值。很容易忘记加引号，尤其对于<IMG>中 ALT 这样的属性，在使用 Web 浏览器查看文档时它们的错误不很明显。

例如下面的<IMG>标记：

```
<IMG SRC=cup.gif WIDTH=89 HEIGHT=67 ALT=Cup>
```

应将其改写为：

```
<IMG SRC="cup.gif" WIDTH="89" HEIGHT="67" ALT="Cup">
```

#### 6.4.1.4 <、>和&必须转义

HTML 对小于号和与号的要求比 XML 宽松得多。即使这样，在纯 HTML 文本中它们确实也会引起麻烦，特别是在它们直接跟有其他字符时。例如，考虑下面来自 Eudora 软件中的 From：标题中的 email 地址在经过复制和粘贴后显示的样子：

```
Elliotte Rusty Harold <elharo@metalab.unc.edu>
```

如果用 HTML 来显示的话，看到的可能是：

```
Elliotte Rusty Harold
```

elharo@metalab.unc.edu 无意间被尖括号隐藏了。如果想在 HTML 中包含原始的小于号和与号, 应当使用<lt;和&lt;代替。其正确的 HTML 形式是:

```
Elliotte Rusty Harold <elharo@metalab.unc.edu>;
```

没有转义的大于号带来的问题不易察觉, 如果在它之前有一个未结束的标记, 它会被解释为一个置标。文档中会出现这种没有完成的标记, 而且附近的大于号会掩盖它们的存在。例如下面的一段 Java 代码:

```
for (int i=0;i<10;i++){  
  
    for (int j=20;j>10;j--){
```

这很可能显示为: ``

```
for (int i=0;i10;j--){
```

如果这只是一个 100 行程序中的两行, 在随便校正时极有可能错过这种疏忽。另一方面, 如果转义了大于号, 而未转义小于号将会隐藏程序的其余部分, 而且这种问题容易被发现。

#### 6.4.1.5 使用一个根元素

用于 HTML 文件的根元素被假定为 html。大部分浏览器允许不包含它的这种疏忽。尽管如此, 最好把<html>作为文档的第一个标记,</html>作为文档的最后一个标记。如果其他文本或置标出现在<html>之前或</html>之后, 应把它们移到<html>和</html>之间。

这个问题常见的形式是忘记在文档的结尾包括</html>。我通常先键入<html>和</html>, 然后在它们之间键入其他内容, 而不是在编写完整个文档再加</html>标记, 指望几天后还会记得应该加上</html>标记。

#### 6.4.1.6 所有标记使用相同的大小写形式

HTML 对大小写不敏感, XML 则不然。应推荐给标记挑选一个唯一的大小写形式, 要么都大写, 要么都小写, 并且贯穿全文。这样做比记住每一个标记的细节要简单。我通常选小写, 因为它比较容易输入。而且 W3C 将 HTML 再现为 XML 应用程序的结果也使用这个格式。



在第 20 章中读取文档类型定义一节将详细描述 HTML 再现为 XML。但是必须停止更深的探讨, 因为这项工作使用在后面几章中学不到的技巧。

#### 6.4.1.7 用"/>~"结束空标记

把 HTML 转换成结构完整的 XML, 其中最令人讨厌的就是空标记。HTML 在形式上不能识别 XML 的<elementname/>空标记句法。虽然很容易将<br>转换为<br/>, <hr>转换为<hr/>, <img>转换为<img/>, 但是给定的浏览器是否会正确显示变换后的标记是一个未知数。



不要把<br>, <hr>, <img>这样真正的空元素与标准的 HTML 中只有一个起始标记但能够带有内容的标记混淆, 如<p>, <li>, <dt>, 和<dd>。

一个被 XML 规范认可的最简单的解决办法是用不含有内容的起始和结束标记对替换空标记。浏览器将忽略该不能识别的结束标记, 请看下面的实例:

<br></br>

<hr></hr>

<IMG SRC="cup.gif" WIDTH="89" HEIGHT="67" ALT="Cup"></IMG>

在实践中这样做确实没有什么问题，但有一个明显的例外。Netscape 4.5 以及更早的版本把</br>和<br>看成是一样的，当作一个换行符号。因此<br>是单个换行符号，<br></br>则是一对换行符号，实际上更像一个段落标记。而且，Netscape 完全忽略<br/>。必须支持传统浏览器的 Web 站点（几乎是所有的 Web 站点）不能使用<br></br>或者<br/>。在实践中对于 XML 和传统浏览器都适用的解决办法如下：

<br />

请注意<br 和/>之间的空格，确实解释不了这样为什么管用，而其他更多的变化却

不行。如果你确实想使用结构完整的 HTML，我所能做的只是提供可能奏效的解决办法。

#### 6.4.1.8 只使用&amp;、&lt;、&gt;、&apos;和&quot;实体引用

许多 Web 页面除了&amp;、&lt;、&gt;、&apos;、和&quot;之外确实不需要更多的实体引用。但是 HTML 4.0 中规定了许多：

- &trade;为商标号 (™)
- &copy;为版权号 (©)
- &infin;为无穷大号∞
- &pi;为小写的希腊字母 pi, π

还有几百个别的实体引用，但是使用任何一个将破坏文档的结构完整性。解决这个问题的方法是使用一个 DTD。我们将在第 9 章中讨论 DTD 对实体引用的影响。同时下面有几个暂时的解决办法。

最简单的办法是以某一字符集编码一个包含全部所需符号的文档，然后使用一个<META>指令指定正在使用的字符集。例如，指定文档使用 UTF-8 编码（一个字符集，包含了几乎全部可能用到的字符，将在第 7 章讨论），而且应当把它放到文档的开头。

<META http-equiv="Content-Type"

content="text/html;charset=UTF-8">

或者可以简单地告诉 Web 服务器，让它提供必要的内容类型标题。通常使用<META>标记要简单一些：

Content-Type:text/html;charset=UTF-8

采用这种方法的问题是许多浏览器不一定能显示 UTF-8 字符集。对于其他提供所用的特殊字符的字符集也一样。

HTML 4.0 支持 XML 中的字符实体引用。这就是说可以使用&#后跟 Unicode 中字符的十进制或者十六进制代码来代替一个字符。例如：

- &#8482; 为商标号 (™)
- &#169; 为版权号 (©)
- &#8734; 为无穷大号∞
- &#960; 为小写的希腊字母 pi, π

HTML 3.2 只正式支持介于 0 和 255 (ISO Latin-1) 之间的数字字符引用, 而 Navigator 4.0 和以后的版本以及 Internet Explorer 能识别更多的 Unicode 字符。

如果确实需要一个结构完整的向后与 HTML 兼容的 XML, 可以把这些字符作为内联图片。例如:

- `</img>`, `</img>` 商标号 (tm);
- ` a`, 为版权号 (c);
- `</img>`, 无穷大号  $\infty$ ;
- `</img>`, 小写的希腊字母 pi,  $\pi$ 。

事实上, 我不赞成使用这种方法。结构完整性在 HTML 中并不太重要, 它只是强制读者增加了下载和显示出来的时间。

#### 6.4.1.9 XML 声明

HTML 文档不需要 XML 声明, 但有也无妨。Web 浏览器只忽略它们不承认的标记。从这一点看, 下面这一行就是另外一个标记:

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

因为不懂 XML 的浏览器解释不了 `<?xml?>` 标记, 它们会简单地忽略它。懂得 XML 的浏览器会把它当作一个提示, 表明该文档是结构完整的 XML 文档, 并按此处理它。

遗憾的是, 不完全懂得 XML 的浏览器分析这些句法非常困难。特别是 Internet Explorer 4.0 for the Mac (不是指 Netscape Navigator 或者其他版本的 IE) 把它当作下载一个文件的信号, 而不作显示。因此, 不得不从 Web 页面中将 XML 声名删除。

#### 6.4.1.10 遵循规则

按照本章描述的规则编写结构完整的 XML 文档不是特别困难, 但是 XML 浏览器对于不标准的句法不像 HTML 浏览器那样宽容, 因此要细心编写。

如果违反了任何结构完整性约束, XML 分析器和浏览器将报告一个句法错误。因此编写 XML 的过程与用某种编程语言编写代码的过程有些相似, 首先编写, 然后编译, 如果编译失败再根据报告的错误修改。

通常在能够看到完成的文档之前要经过几次从编辑到编译的过程。而且编写 XML 文档比编写 C 和 Java 源代码要容易得多, 只要很少的练习就会达到只出现相当少的错误, 编写 XML 的速度几乎与你输入的速度一样快。

### 6.4.2 HTML 整理工具

有几种工具能够帮助我们整理页面。最引人注目的是来自 XML.COM 的 RUWF (Are You Well Formed?) 和由 W3C 的 Dave Raggett 编写的 HTML Tidy。

#### 6.4.2.1 RUWF

任何能够检验 XML 文档结构完整性的工具同样能够检验结构完整的 HTML 文档。其中最容易使用的工具是来自 XML.COM 的 RUWF 结构完整性检验程序。图 6-2 显示了该检验程序。只要键入想检验的页面的 URL, RUWF 就会返回在页面上发现的几十个错误。



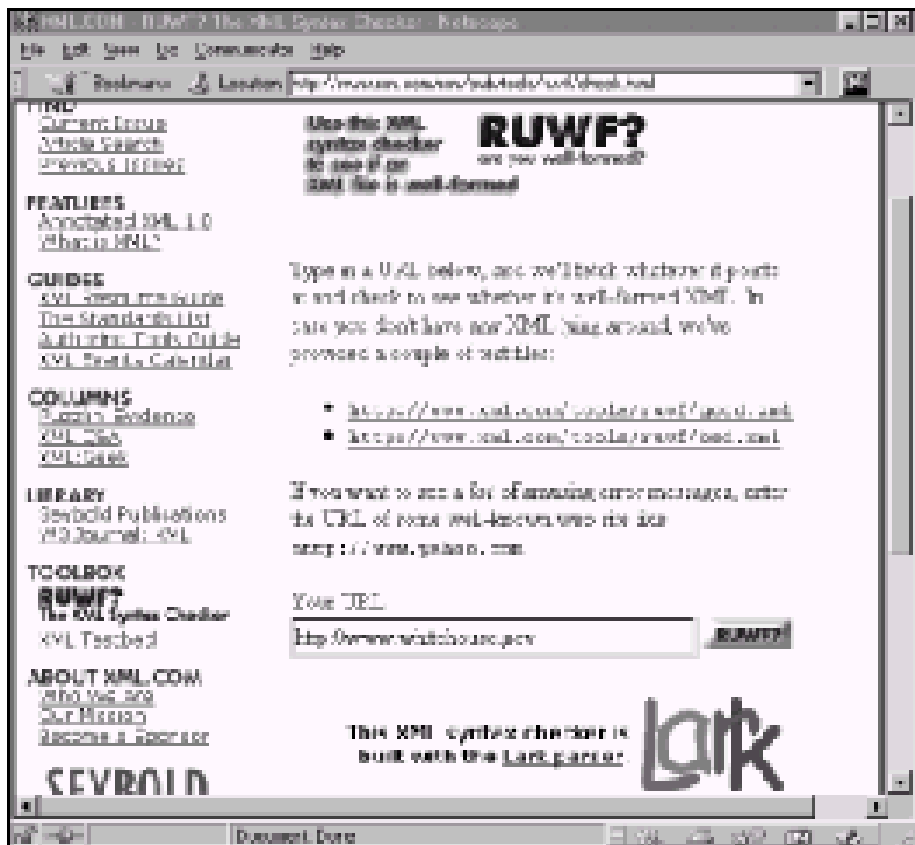


图 6-2 RUWF 结构完整性检验器

下面是 RUWF 在白宫主页上找到的第一批错误。这些错误大部分是不标准的 XML，但是它们是合法的 HTML。但至少有一处错误（“第 55 行，30 列：</FONT>周围没有相应的起始标记”）对 HTML 和 XML 都是一个错误。

Line 28, column 7:Encountered </HEAD>expected </META>

...assumed </META>...assumed </META>...assumed </META>

...assumed </META>

Line 36, column 12, character '0' :after AttrName=in start-tag

Line 37, column 12, character '0' :after AttrName=in start-tag

Line 38, column 12, character '0' :after AttrName=in start-tag

Line 40, column 12, character '0' :after AttrName=in start-tag

Line 41, column 10, character 'A' :after AttrName=in start-tag

Line 42, column 12, character '0' :after AttrName=in start-tag

Line 43, column 14:Encountered </CENTER>expected </br>

...assumed </br>...assumed </br>

Line 51, column 11, character '+' :after AttrName=in start-tag

Line 52, column 51, character '0' :after AttrName=in start-tag

Line 54, column 57:after &

Line 55, column 30:Encountered </FONT>with no start-tag.

Line 57, column 10, character 'A' :after AttrName=in start-tag

Line 59, column 15, character '+' :after AttrName=in start-tag

#### 6.4.2.2 HTML Tidy

一旦确定了问题就会想到要修改它们，许多常见的问题&#0;&#0;例如，给属性值加引号&#0;&#0;是能够自动被修改的。做这种修改最便利的工具是 Dave Raggett 的命令行程序 HTML Tidy。Tidy 是用 ANSI C 写成的一个字符-模式程序，能够在许多操作平台如 Windows、Unix、BeOS 和 Mac 系统上执行。



Tidy 在本书所附的 CD-ROM 的 utilities/tidy 目录中，包含用于 Windows NT 和 BeOS 的二进制代码和用于所有操作平台的可移植代码。可以从站点 <http://www.w3.org/People/Raggett/tidy/> 中下载最新版本的 Tidy。

Tidy 通过几种不同的方式整理 HTML 文件，它们并非都与 XML 结构完整性有关。事实上在默认模式下，Tidy 倾向于删除不必要的结束标记（对 HTML 而言，不是对 XML），像</LI>。并且对破坏结构完整性的地方作一些修改。但是可以使用-asxml 开关指定需要结构完整的 XML 输出。例如，把 index.html 文件转换为结构完整的 XML，需要从 DOS 窗口或者外壳提示符下输入：

```
C:\>tidy -m -asxml index.html
```

-m 标志告诉 Tidy 就地转换文件。-asxml 标志告诉 Tidy 把输出的文档格式转化为 XML 文档。

## 6.5 本章小结

在本章学习了如何编写结构完整的 XML。主要包括以下内容：

- XML 文档是满足一定结构完整性标准的一连串字符
- XML 文档的文本分为字符数据和置标
- 注释可为代码加上说明文字，可能是为了自己看的注释，也可能是通过注释将还没有写好的部分注释掉
- 使用实体引用可以在文档中包含<、>、&、"和
- CDATA 段对于嵌有很多<、>和&字符的文档是很有用的
- 在 XML 文档中的标记以<开始，并以>结束，而且不能出现在注释或者 CDATA 段中
- 起始标记和空标记可以包含描述元素的属性
- HTML 文档稍加处理会变得结构完整

在下一章将要讨论如何使用非英语语言编写 XML，尤其是用与英语差别很大的语言。如阿拉伯语、汉语和希腊语。

## 第 7 章 外文和非罗马文本

Web 是国际性的，然而在其中使用的大多数是英文，XML 正在开始改变这种状况。XML 全面支持双字节 Unicode 字符集及其更简洁的描述形式。这对 Web 作者来说是个好消息，因为 Unicode 支持世界上每种现代文字通常使用的几乎所有的字符。

本章将学习在计算机应用程序中如何描述国际性文本，XML 如何理解文本以及如何利用非英文软件。

本章的主要内容包括：

- 了解非罗马文字在网页上的效果
- 使用文字、字符集、字体和字形
- 传统的字符集
- 使用 Unicode 字符集
- 使用 Unicode 编写 XML 文件

### 7.1 Web 上的非罗马文字

虽然 Web 是国际化的，但它的大部分文本是英文。由于网络的不断扩展，还能领略到法语、西班牙语、汉语、阿拉伯语、希伯来语、俄语、北印度语和其他语言的网页。很多时候这些网页没有理想的那么多。图 7-1 是 1998 年 10 月一份美国信息部宣传杂志的封面页面：*Issues in Democracy* (<http://www.usia.gov/journals/itdhr/1098/ijdr/ijdr1098.htm>)，是用英文编码显示的俄文译本。左上方红色的古斯拉夫文本是一张位图图片文件，因此很清晰（如果懂俄语的话），还有几个清晰的英文单词，如“Adobe Acrobat”。其余的大部分是加重音的罗马元音，不是想象的古斯拉夫字母。

当使用复杂的非西方文字时，如中国或日本文字，网页的质量会更差。图 7-2 是使用英文浏览器显示 *JavaBeans* (IDG Books, 1997, <http://www.ohmsha.co.jp/data/books/contents/4-274-06271-6.htm>) 的日文版主页。同样的结果，位图图片显示了正确的日文（还有英文）文本，页面上其余的文本除了几个可辨认的英文单词像 JavaBeans 之外，就像是一个随机的字符组合。而希望看到的日文字符完全看不到。

如果使用正确的编码和应用软件，并安装正确的字体，这些页面就可以正确显示。图 7-3 是使用古斯拉夫的 Windows 1251 编码显示的 *Issues in Democracy*。可以看到图片下面的文本是可读的（如果懂俄语的话）。

可以从 Netscape Navigator 或 Internet Explorer 的 View/Encoding（视图/编码）菜单中为网页选取编码方式。在理想情况下，网络服务器会告诉网络浏览器使用何种编码，同时 Web 浏览器会接受。如果网络服务器能向网络浏览器传送显示页面的字体就更好。事实上，经常需要人工选择编码方式。当原稿有几种编码时，不得不尝试多个编码直至找到特别合适的一个。例如，一张古斯拉夫页面能用 Windows 1251、ISO 8859-5 或者 KOI6-R 编码。选择错误的编码可能会显示古斯拉夫字母，但单词将是不知所云、毫无意义的。



图 7-1 用一种罗马文字观看的 1998 年 10 月版关于探讨民主政治的俄文译本

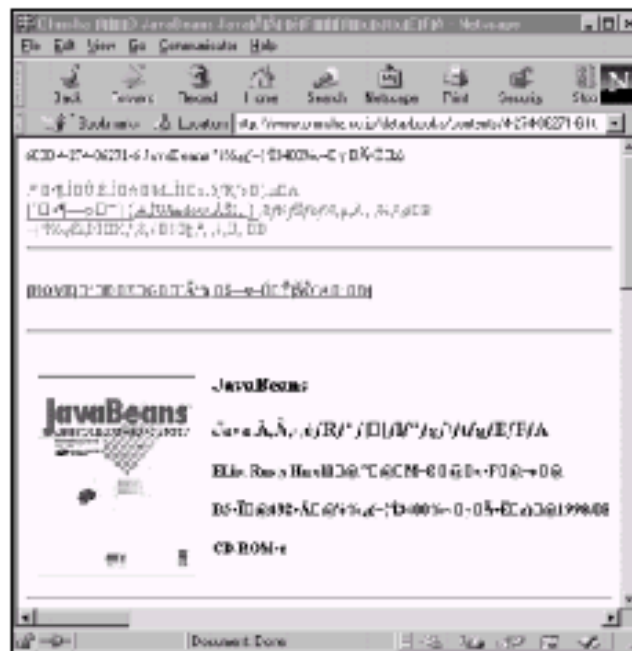


图 7-2 用英文浏览器看到的 JavaBeans 的日文翻译页面



图 7-3 使用古斯拉夫文字看到的 Issues of Democracy

即使能够指定编码,也不能确保有显示它的字体。图 7-4 是使用日文编码的 JavaBeans 日文主页,但是在计算机中却没有任何一种日文字体。文本中的多数字符显示成方框,表明这是一个得不到的字符轮廓。幸运的是, Netscape Navigator 能够辨认出页面上的双字节日文字符和两个单字节的西文字符。

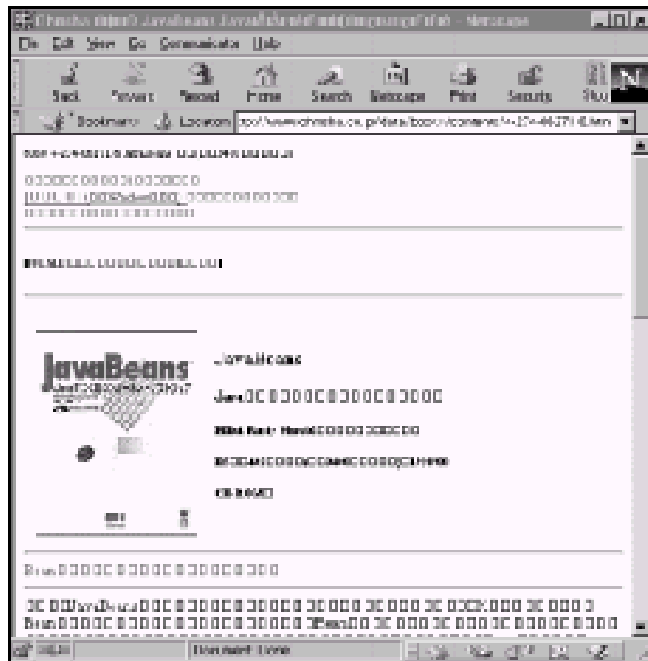


图 7-4 在没有必需的日文字体的情况下所显示的 *JavaBeans* 日文译本

如果有一种日本地方语言操作系统版本,它包含必要的字体或者别的软件,如 Apple 的 Japanese Language Kit 或南极星的 NJWin (<http://www.njstar.com/>), 这样就可以看到文本,大致如图 7-5 所示。

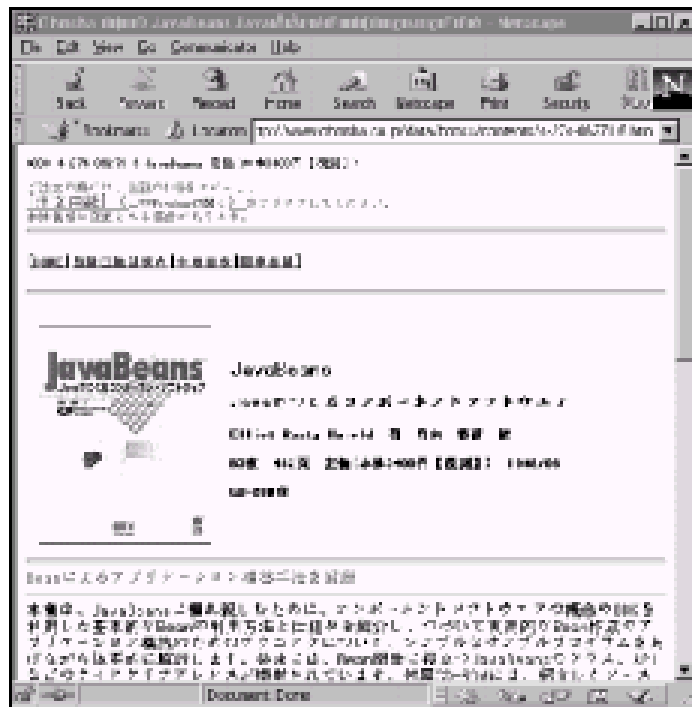


图 7-5 在安装有所要的日文字体的浏览器上显示的 *JavaBeans* 译文

当然，所使用的字体质量越高，文本的效果看起来就越好。中文和日文的字体非常庞大（中文有大约 80,000 多个汉字），而且单个文字间的差别很小。日文出版商比西方出版商对纸张和打印技术的要求更高，以保持必要的细节打印日文字符。遗憾的是一个 72-dpi 的计算机显示器不能很好地显示中文和日文字符，除非使用很大的字体。

由于每个页面只能有一种编码，因而要编写集成了多种文字的网页，如对中文的法文注释，是非常困难的。由于这一原因，网络界需要一种单一的、通用的字符集，使所有计算机和网络浏览器能显示网页中的所有字符。目前仍然没有这样的字符集，XML 和 Unicode 是最好的。

XML 文件是用 Unicode 编写的，这种双字节字符能表示世界各国语言中大部分的字符。如果网页是用 Unicode 编写的 XML 网页，而且所用的浏览器懂得 Unicode，如 XML 浏览器，那么就可以在同一页面中包含不同语种的字符。

浏览器不需要区分不同的编码，如 Windows 1251、ISO 8859-5 或者 KOI8-R。浏览器假定网页都是用 Unicode 编写的。只要双字节字符集有容纳不同字符的余地，就不需要使用多种字符集。因此，浏览器也不必检测使用的是哪一种字符集。

## .2 文字、字符集、字体和字形

大部分现代人类语言都有各自的书写形式。用于书写一种语言的字符集称为一种文字。文字可以是语音字母表，也可以不是。例如，汉语、日语和韩语由能够表示整个词汇的表意文字字符组成。不同语言经常共用一些文字，或者有一些细小的改动。例如，汉语、日语和韩语实质上共用相同的 80,000 多个汉字，尽管大多数字符在不同的语言中表示的意义不同。



单词 Script 也经常用来指用非类型化和非解释语言写的程序，如 JavaScript、Perl 和 TCL。本章中的 Script 指书写一种语言使用的字符，不是指任何一种程序。

一些语言能用不同的文字表达。塞尔维亚语和克罗地亚语实际是相同的，通常被称作 Serbo-Croatian。但是，塞尔维亚语使用经过修改的古斯拉夫文字，克罗地亚语则使用经过修改的罗马文字。只要计算机不想得到所处理的文字的意义，处理一种文字和处理用这种文字所编写的任何一种语言都是相同的。

遗憾的是，单独的 XML 无法读取一种文字，计算机要处理一种文字需要四个要素：

1. 与文字对应的一种字符集
2. 用于该字符集的一种字体
3. 该字符集的一种输入方法
4. 理解该字符集的一个操作系统或应用程序

这四个要素只要缺少其中之一，就不能在这种文字环境下工作，尽管 XML 能够提供一个足可以应急的工作环境。如果在应用过程中只丢失了输入法，还能够读取用该文字写的文本，只是不能用这种文字书写文本。

### 7.2.1 文字的字符集

计算机只懂得数字。在它处理文本之前，必须用一种特定的字符集将文本编码成数字。例如，在大家熟知的 ASCII 字符集中，‘A’的编码是 65，‘B’的编码是 66，‘C’的编码是 67，以此类推。

这些是语意学编码，不提供样式或者字体信息。**C**、*C*或C的编码都是 67。有关如何画出字符的信息存储在别处。

### 7.2.2 字符集的字体

字符集所采用的各种字形的总和形成一种字体，通常包括一定的尺寸、外观和风格。例如 **C**、*C*或C是同一字符，只是书写的形状不一样，但其意义是相同的。

不同的系统存储字形的方式不一样。它们可能是位图或矢量图，甚至是印刷厂中的铅字。它们采用的形式与我们无关，关键是字体可以告诉计算机如何从字符集中调出每一个字符。

### 7.2.3 字符集的输入法

输入文本需要一种输入法，讲英语的人不需要考虑它，只要敲击键盘就可以输入。在大部分欧洲国家也一样，只需要在键盘上附加几个元音变音、变音符号。

基本上，古斯拉夫语、希伯来语、阿拉伯语和希腊语比较难输入。键盘上的按键数目有限，一般不够阿拉伯和罗马字符，或者是罗马和希腊字符使用。假定需要两种字符，键盘上有一个希腊字符锁定键能使键盘在罗马字符和希腊字符之间来回切换，那么希腊字符和罗马字符就能以不同的颜色印在键盘上。这个方案同样适用于希伯来语、阿拉伯语、古斯拉夫语和其他非罗马字符集。



当碰到表意文字如汉语和日语时，上述方法确实不管用。日语的键盘可容纳大约 5000 个不同的键，但还不到日语的 10%！音节、语音和部首表示法能够减少按键的数目，但是键盘是否适合输入这些语种的文本呢？同西方相比，正确的语音和手写体识别在亚洲有更大的潜力。

语音和手写体识别还没有达到足可以让人信赖的程度，目前输入单个字符的方法大部分是使用键盘上的多个键序列。例如，输入汉语的“羊”字，必须按下 ALT 键并按带有（~）的键，然后输入 yang，单击回车键。这种输入方法会显示出一列发音与 yang 差不多的汉字。例如：

佯楊易暘楊洋瘍羊詳錫陽

接下来就可以选择需要的那个字符“羊”。对于不同的程序、不同的操作系统和不同的语言如何把键入的键值转换成文字字符，如“羊”所使用的 GUI（图形用户界面）和翻译系统的细节是不同的。

## 7.2.4 操作系统和应用软件

主要的 Web 浏览器（Netscape Navigator 和 Internet Explorer）能很好地显示非罗马文字。如果潜在的操作系统支持给定的一种文字并存储有相应的字体，Web 浏览器就能够显示这种文字。

MacOS 7.1 及其新版本能够处理当今世界上常见的多数文字。但是基本操作系统仅支持西方欧洲语言。汉语、日语、韩语、阿拉伯语、希伯来语和古斯拉夫语只能从语言工具中获得，每一种 100 美元。同时提供相应语言的字体和输入法。也有印度语工具包，用来处理印度次大陆上常见的梵文、吉吉拉特语和 Gurmukhi 文字。MacOS 8.5 增加了对 Unicode 可选而有限的支持（多数应用软件都不支持 Unicode）。

Windows NT 4.0 把 Unicode 当作本身的字符集使用。NT 4.0 能够很好地处理罗马语、古斯拉夫语、希腊语、希伯来语和其他几种语言。Lucida Sans Unicode 字体覆盖了最常用的 1300 种 Unicode 中的大约 40,000 多个字符。Microsoft Office 97 包括汉语、日语和韩语字体，可以安装它来读取这些语言的文本（在你的 Office CD-ROM 上查询 Valupack 文件夹中的 Fareast 文件夹）。

微软宣称 Windows 2000（以前称为 NT 5.0）将包含能覆盖大部分中-日-韩文字的字体和相应的输入法。但是他们同样许诺过 Windows 95 包含 Unicode 支持软件，尽管失败了。因此不必焦虑等待。当然，如果所有的 NT 版本能够提供世界性的支持软件是非常好的，就不必再依赖于本地化了。

微软的消费类操作系统，如 Windows 3.1、95 和 98 不完全支持 Unicode。相反它们需要依靠能处理基本英文字符和本地化文字的本地化系统。

主要的 Unix 变体包含不同等级的 Unicode 支持软件。Solaris 2.6 支持欧洲语言、希腊语和古斯拉夫语。汉语、日语和韩语由本地化版本支持，它们使用不同于 Unicode 的编码。Linux 对 Unicode 的支持尚在开始阶段，这在不久的将来会很有用。

### 7.3 传统字符集

不同地区的不同计算机使用的默认字符集各不相同，大多数现代计算机使用 ASCII 码扩展字符集。ASCII 码含有英语字母表和大部分常见的标点符号以及空格符的编码。

在美国,Mac 计算机使用 MacRoman 字符集,Windows PC 机使用 Windows ANSI 字符集,大部分 Unix 工作站使用 ISO Latin-1。这些都是扩展的 ASCII 码，支持西方欧洲语言，如法语和西班牙语中的多出来的字符，如 ç 和 ?。在其他地区，如日本、希腊和以色列，计算机仍然使用令人困惑的混合字符集，这些字符集几乎都支持 ASCII 码加本地语言。

上述方法在 Internet 上无效。当你正在互联网上阅读 *San Jose Mercury News*，翻页时不会遇到几个用德语或汉语写的栏目。但是在 Web 页面上，这完全可能。用户将跟随一个链接并停止在一个日文界面的开始。即使网上冲浪者不懂日语，他们如果能看到一个好的日本版面也是不错的。如图 7-5 所示，而不是图 7-2 显示的那种随意的字符组合。

XML 处理这个问题是通过把小的、局部的字符集以外的字符集合并到一个大的字符集中，并假定它包含了地球上现存语言（和某些已消失的语言）使用的文字。这种字符集称为 Unicode。同前面提到的一样，Unicode 是一个双字节字符集，它能表示多种文字和几百种语言中的 40,000 多个不同字符。即使不能全部显示 Unicode，所有的 XML 处理器必须识别 Unicode。

在第 6 章中学过，一个 XML 文档分成文本和二进制实体两部分，每个文本实体有一种编码方法。如果编码在实体定义中没有明确指定，就会默认为 UTF-8 一种 Unicode 的压缩形式，将保持纯 ASCII 文本不变。因此，只包含普通 ASCII 字符的 XML 文件，不会用处理 Unicode 这种多字节字符集的复杂工具对它进行编辑。

#### 7.3.1 ASCII 字符集

ASCII，即 American Standard Code for Information Interchange（美国标准信息交换码），是一个原始的字符集，而且是到目前为止最通用的。它形成了所有字符集必须支持的最主要部分。它基本上只定义了书写英语需要的全部字符，这些字符的编码是 0~127。表 7-1 显示了 ASCII 字符集。

表 7-1 ASCII 字符集

编码	字 符	编码	字符	编码	字符	编码	字符
0	空字符(Control-@)	32	Space	64	@	96	`
1	标题开始字符(Control-A)	33	!	65	A	97	A
2	正文开始字符(Control-B)	34	“	66	B	98	B
3	正文结束字符(Control-C)	35	#	67	C	99	C
4	传输结束字符(Control-D)	36	\$	68	D	100	d
5	询问字符(Control-E)	37	%	69	E	101	e
6	应答字符(Control-F)	38	&	70	F	102	f
7	响铃字符(Control-G)	39	‘	71	G	103	g
8	退回字符(Control-H)	40	(	72	H	104	h
9	制表符(Control-I)	41	)	73	I	105	i
10	回行字符(Control-J)	42	*	74	J	106	j
11	垂直制表符(Control-K)	43	+	75	K	107	k
12	进纸字符(Control-L)	44	,	76	L	108	l
13	回车字符(Control-M)	45	-	77	M	109	m
14	移出字符(Control-N)	46	.	78	N	110	n

15	移入字符(Control-0)	47	/	79	O	111	o
16	数据连接转义符(Control-P)	48	0	80	P	112	p
17	设备控制 1 (Control-Q)	49	1	81	Q	113	q
18	设备控制 2 (Control-R)	50	2	82	R	114	r
19	设备控制 3 (Control-S)	51	3	83	S	115	s
20	设备控制 4 (Control-T)	52	4	84	T	116	t
21	拒绝应答字符 (Control-U)	53	5	85	U	117	u
22	同步等待字符 (Control-V)	54	6	86	V	118	v
23	传输块结束符 (Control-W)	55	7	87	W	119	w
24	删除字符 (Control-X)	56	8	88	X	120	x
25	媒体结束符 (Control-Y)	57	9	89	Y	121	y
26	替换字符 (Control-Z)	58	:	90	Z	122	z
27	转义字符 (Control-[)	59	;	91	[	123	{
28	文件分隔符 (Control-\)	60	<	92	\	124	
29	组群分隔符 (Control-])	61	=	93	]	125	}
30	记录分隔符 (Control-^)	62	>	94	^	126	~
31	单元分隔符 (Control-_)	63	?	95	_	127	delete

在 0~31 之间的字符是非打印控制字符，包括回车、送纸、制表、响铃和其他类似的字符。其中有许多字符是以纸为基础的电传打印机时代遗留下来的。例如，回车在字面上表示把支架移回到左边空白处，就像在打字机上做一样。送纸使打印机滚筒向上移动一行。除了提及的几个字符外，其他的这些字符使用率不高。

人们所碰到的大多数字符集可能是 ASCII 的扩展字符集。换句话说，它们定义在 0 到 127 之间的字符同 ASCII 一样，只是增加了 127 以后的字符。

### 7.3.2 ISO 字符集

ASCII 中的“A”代表美国，因此 ASCII 码专门用于书写英语，严格来说是美式英语也就不足为奇了。ASCII 码中缺少 &、u、?和许多书写其他语言和地区所需的字符。

可通过指定 128 以后的更多字符扩展 ASCII 码。国际标准组织 (ISO) 定义了几个不同的字符集，它们是在 ASCII 码基础上增加了其他语言和地区需要的字符。其中最突出的是 ISO8859-1，通常叫做 Latin-1。Latin-1 包括了书写所有西方欧洲语言不可缺少的附加字符，其中 0~127 的字符与 ASCII 码相同。表 7-2 给出了 128~255 之间的字符，同样前 32 个字符是极少使用的非打印控制字符。

表 7-2 ISO 8859-1 Latin-1 字符集

编码	字符	编码	字符	编码	字符	编码	字符
128	未定义	160	不可分空格	192	À	224	À
129	未定义	161	?	193	Á	225	Á
130	Bph	162	¢	194	Â	226	Â
131	Nbh	163	£	195	Ã	227	Ã
132	未定义	164	¤	196	Ä	228	Ä
133	Nel	165	¥	197	Å	229	Å
134	Ssa	166	ß	198	Æ	230	Æ
135	Esa	167	§	199	Ç	231	Ç
136	Hts	168	¨	200	È	232	È
137	Htj	169	©	201	É	233	É
138	Vts	170	ª	202	Ê	234	Ê
139	Pld	171	¿	203	Ë	235	Ë
140	Plu	172	?	204	Ì	236	Ì
141	Ri	173	任意字符	205	Í	237	Í
142	ss2	174	®	206	Î	238	Î
143	ss3	175	¯	207	Ï	239	Ï
144	Dcs	176	°	208	Ð	240	é
145	pu1	177	±	209	Ñ	241	Ñ
146	pu2	178	²	210	Ò	242	Ò
147	Sts	179	³	211	Ó	243	Ó
148	Cch	180	´	212	Ô	244	Ô
149	Mw	181	µ	213	Õ	245	Õ
150	Spa	182	¶	214	Ö	246	Ö
151	Epa	183	·	215	×	247	÷
152	Sos	184	¸	216	Ø	248	Ø
153	未定义	185	¹	217	Ù	249	Ù
154	Sci	186	º	218	Ú	250	Ú
155	Csi	187	»	219	Û	251	Û
156	St	188	¼	220	Ü	252	Ü
157	Osc	189	½	221	Ý	253	Ý
编码	字符	编码	字符	编码	字符	编码	字符
158	Pm	190	¾	222		254	
159	Apc	191	?	223	ß	255	?

Latin-1 仍然缺少许多有用的字符，如希腊语、古斯拉夫语、汉语和其他文字及语言需要的字符。你也许会想到从 256 开始定义这些字符，这样做存在一个问题，单个字节只能表示 0~255 的数值，如果超出这个范围，需要使用多字节字符集。出于历史的原因，许多程序都是在一个字节表示一个字符的假定下编写的，这些程序在遇到多字节字符集时就会出错。因此，目前大多数操作系统（Windows NT 例外）使用不同的单字节字符集而不是一个庞大的多字节字符集。Latin-1 是最常见的这种字符集，其他字符集用于处理别的语言。

ISO 8859 另外定义了 10 个适用于不同文字的字符集(8859-2 到 8859-10 和 8859-15),还有 4 个字符集(8859-11 到 8859-14)正在开发。表 7-3 列出了 ISO 字符集以及使用它的语言和文字。这些字符集共享 0~127 的 ASCII 码,只是每个字符集都包含了 128~255 的其他字符。

表 7-3 ISO 字符集

字符集	又 名	语 言
ISO 8859-1	Latin-1	ASCII 码加大部分西欧语言要求的字符,包括阿尔巴尼亚语、南非荷兰语、巴斯克语、加泰罗尼亚语、丹麦语、荷兰语、英语、法罗群岛语、芬兰语、佛兰德语、加利尼西亚语、德语、冰岛语、爱尔兰语、意大利语、挪威语、葡萄牙语、苏格兰语、西班牙语、瑞典语。但是其中忽略了 ij(荷兰语)、? (法语)和德语的引号
ISO 8859-2	Latin-2	ASCII 码加中欧语言要求的字符,包括捷克语、英语、德语、匈牙利语、波兰语、罗马尼亚语、克罗地亚语、斯洛伐克语、斯洛文尼亚语和 Sorbian
ISO 8859-3	Latin-3	ASCII 码、英语、世界语、德语、马耳他语和加利尼西亚语要求的字符
ISO 8859-4	Latin-4	ASCII 码加波罗地海语、拉托维亚语、立陶宛语、德语、格陵兰岛语和拉普兰语中要求的并且被 ISO 8859-10、Latin-6 取代的字符 Latvian, Lithuania
ISO 8859-5		ASCII 码加古斯拉夫字符,用于 Byelorussian、保加利亚语、马其顿语、俄语、塞尔维亚语和乌克兰语
ISO 8859-6		ASCII 码加阿拉伯语
ISO 8859-7		ASCII 码加希腊语
ISO 8859-8		ASCII 码加希伯来语
ISO 8859-9	Latin-5	就是 Latin-1,但用土耳其字母代替了不常用的冰岛语字母
ISO 8859-10	Latin-6	ASCII 码和日耳曼语、立陶宛语、爱斯基摩语、拉普兰语和冰岛语中的字符
ISO 8859-11		ASCII 码加泰国语
ISO 8859-12		适用于 ASCII 码和梵文
ISO 8859-13	Latin-7	ASCII 码加波罗地海周边的语言,特别是拉托维亚语
ISO 8859-14	Latin-8	ASCII 码加盖尔语和威尔士语
ISO 8859-15	Latin-9 Latin-0	本质上与 Latin-1 相同,但是带有欧元符号,而不用国际货币符。而且用芬兰字符代替了一些不常用的字符。用法语字母 CE、ce 代替了 1/4、1/2

这些字符集常有重叠。有几种语言,特别是英语和德语可以使用多种字符集书写。在一定程度上不同的字符集允许结合不同的语言。例如, Latin-1 结合了大部分欧洲语言和冰岛语言,而 Latin-5 结合了大部分西方语言和土耳其语而不是冰岛语。因此如果需要一个包括英语、法语和冰岛语的文档,应当使用 Latin-1。相反,一个文档含有英语、法语和土耳其语,则需要 Latin-5。但是,对于一个要求英语、希伯来语和土耳其语的文档,必须使用 Unicode 来书写,因为没有单字节字符集能够完全处理这三种语言和文字。

单字节字符集不能满足汉语、日语和韩语的要求。这些语言含有的字符多于 256 个,因此必须使用多字节字符集。

7.3.3 MacRoman 字符集

Macos 比 Latin-1 早几年出现,ISO 8859-1 标准在 1987 年第一次被采用(第一个 Mac 计算机是在 1984 年出现)。这意味着苹果公司不得不定义自己的扩展字符集 MacRoman。其中大部分扩展符同 Latin-1 一样(除冰岛语中的”),只

是字符对应的编码不同。MacRoman 中前 127 个字符与 ASCII 码和 Latin-1 中的一样。因此使用扩展字符的文本文件从 PC 机移到 Mac 时会显示混乱，反之亦然。表 7-4 列出了 MacRoman 字符集的后半部分。

表 7-4 MacRoman 字符集

编码	字符	编码	字符	编码	字符	编码	字符
128	Â	160	?	192	?	224	?
129	Ã	161	°	193	?	225	•
130	Ç	162	℥	194	?	226	?
131	É	163	£	195	√	227	?
132	Ñ	164	§	196	?	228	%
133	Ö	165	•	197	?	229	À
134	Û	166	?	198	?	230	Ê
135	Á	167	ß	199	?	231	Á
136	À	168	®	200	?	232	
137	Â	169	©	201	...	233	È
138	Ä	170	™	202	非换行空格	234	í
139	Å	171	'	203	À	235	î
140	Ä	172	¨	204	Ä	236	ï
141	Ç	173	≠	205	Ö	237	ì
142	É	174	Æ	206	?	238	í
143	È	175	Ø	207	?	239	ó
144	Ê	176	∞	208	-	240	ô
145	Ë	177	±	209	_	241	Apple
146	í	178	≤	210	”	242	ò
147	ì	179	≥	211	”	243	ú
148	ï	180	¥	212	‘	244	û
149	ï	181	μ	213	‘	245	ı
续表							
编码	字符	编码	字符	编码	字符	编码	字符
150	ñ	182	¶	214	÷	246	?
151	ó	183	Σ	215	à	247	~
152	ò	184	Π	216	?	248	-
153	ô	185	Π	217	?	249	
154	ö	186	ƒ	218	/	250	.
155	õ	187	a	219	☒	251	°
156	ú	188	°	220	?	252	?
157	Û	189	Ω	221	?	253	”.
158	Û	190	Æ	222	fi	254	.
159	Ü	191	Ø	223	fl	255	?

7.3.4 Windows ANSI 字符集

第一个被广泛使用的 Windows 版本比 Mac 晚几年出现，因此它能够采用 Latin-1 字符集。它使用更多的可打印字符代替介于 130 和 159 之间的非打印控制字符，从而进一步扩展了使用范围。这个经过修改的 Latin-1 版本通常被称作 Windows ANSI。表 7-5 列出了 Windows ANSI 字符集。

表 7-5 Windows ANSI 字符集

编码	字符	编码	字符	编码	字符	编码	字符
128	未定义	136	?	144	未定义	152	~
129	未定义	137	%	145	‘	153	™
130	,	138	146	‘	154	154	§
131	□	139	?	147	”	155	?
132	”	140	?	148	”	156	?
133	...	141	未定义	149	&#0;	157	未定义
134	?	142	未定义	150	-	158	未定义
135	?	143	未定义	151	—	159	?

## 7.4 Unicode 字符集

为了使不同的字符集能够处理好不同的文字和语言，必须满足：

1. 不同时引用多种文字。
2. 不与使用不同字符集的人交换文件。

由于 Mac 和 PC 机都使用不同的字符集，越来越多的人无法遵循以上原则。很明显的是需要一种得到大家的认可并且编码了全世界各种文字的字符集。建立这样的字符集很难，需要对成百上千种语言和文字有细致的了解。要使软件开发商们同意使用这种字符集就更难了。不过这方面的努力一直在进行，终于创建了一个符合要求的字符集——Unicode。而且主要卖方（微软、苹果、IBM、Sun、Be 等）正逐步趋向于使用它。XML 把 Unicode 当作自己的默认字符集。

Unicode 使用 0~65,535 的双字节无符号数对每一个字符进行编码。目前已经定义了 40,000 多个不同的 Unicode 字符，剩余 25,000 个空缺留给将来扩展之用。其中大约 20,000 个字符用于汉字，另外 11,000 左右的字符用于韩语音节。Unicode 中 0~255 的字符与 Latin-1 中的一致。

如果在本书中显示所有的 Unicode 字符，那么除了这些字符表格外，书中将容纳不下别的任何东西。如果需要知道 Unicode 中不同字符的确定编码，买一册 Unicode 标准（第二版，ISBN 0-201-48346-9，Addison-Wesley 出版）。该书共 950 页，包括对 Unicode 2.0 的全部详细说明，还包括 Unicode 2.0 中定义的所有字符集的图表。还可以在 Unicode 协会的网址：<http://www.unicode.org/>和 <http://charts.unicode.org/>中发现在线信息。表 7-6 列出了由 Unicode 编码的文字，由此可知 Unicode 的广泛性。每一种文字的字符通常编码在 65,536 个号码中的一个连续区域内。许多语言都能使用其中某一区域的字符书写（例如，使用古斯拉夫语书写俄语），尽管有一些语言，如克罗地亚语或土耳其语需要混合匹配前 4 个拉丁文区域中的字符。

表 7-6 Unicode 文字块

文 字	范 围	目 的
Basic Latin 基本拉丁语	0-127	ASCII 码，美式英语
Latin-1 Supplement 拉丁语补充-1	126-255	ISO Latin-1 前半部分结合 Basic Latin 能处理丹麦语、荷兰语、英语、法罗群岛语、佛兰德语、德语、夏威夷语、冰岛语、印度尼西亚语、爱尔兰语、挪威语、葡萄牙语、西班牙语、斯瓦西里语和瑞典语
Latin Extended-A 拉丁文扩展集-A	256-383	该字符块增添了 ISO 8859 字符集 Latin-2、Latin-3、Latin-4 中的字符，而且是 Basic Latin 和 Latin-1 没有的字符。同它们结合能够编码南非荷兰语、法国布里多尼语、巴斯克语、加泰罗尼亚语、捷克语、世界语、爱沙尼亚语、法语、Friesland 语、格陵兰岛语、匈牙利语、拉脱维亚语、立陶宛语、马耳它语、波兰语、普罗旺斯语、罗马尼亚语、吉普塞语、斯洛伐克语、斯洛文尼亚语、土耳其语和威尔士语
Latin Extended-B 拉丁文扩展集-B	383-591	大部分字符用于扩展 Latin 文字以处理使用非传统文字写的语言，包括许多非洲语言、克罗地亚连字符，与塞尔维亚古斯拉夫字母、中国的拼音和 Latin-10 中的 Sami characters 相匹配
IPA 扩展字符集	592-687	国际音标字母
间距调节字符	686-767	通常能够改变前面字母发音的小符号
可识别的连接字符	766-879	不独立存在，一般与前面的字母连用(放置在上边)的可识别的记号，如：~、‘and ??
希腊	880-1023	基于 ISO 8859-7 的现代希腊语，同时提供古埃及语字符
续表		



文 字	范 围	目 的
古斯拉夫	1024-1279	基于 ISO 8859-5 上的语言, 俄语和多数斯拉夫语(乌克兰语、Byelorussian 等), 前苏联的许多非斯拉夫语言(Azerbaijani, Ossetian, 卡巴尔德语, Chechen, Tajik 等). 几种语言(库尔德语, 阿布哈西亚语)需要 Latin 和古斯拉夫字母
美国	1326-1423	美语
希伯来	1424-1535	希伯来语(古典和现代)、依地语、Judezmo、早期美语。
阿拉伯	1536-1791	阿拉伯语, 波斯语、Pashto、Sindhi、库尔德语和早期土耳其语
梵文字母	2304-2431	梵语, 北印度语, 尼泊尔语和印度次大陆语言, 包括: Awadhi, Bagheli, Bhatneri, Bhili, Bihari, BrajBhasha, Chhattisgarhi, Garhwali, Gondi, Harauti, Ho, Jaipuri, KachchhiKanauji, Konkani, Kului, Kumaoni, Kurku, Kurukh, Marwari, Mundari, Newari, Palpa, and Santali
孟加拉语	2432-2559	一种北印度文字, 使用于印度的西孟加拉州和孟加拉国的孟加拉语、阿萨姆语、Daphla、Garo、Hallam、Khasi、Manipuri、Mizo、Naga、Munda、Rian、Santali
Gurmukhi	2560-2687	Punjabi
Gujarati	2686-2815	Gujarati
Oriya	2816-2943	Oriya、Khondi、Santali
泰米尔语	2944-3071	泰米尔语和 Badaga、使用于南印度、斯里兰卡、新加坡和马来西亚部分地区
Telugu	3072-3199	Telugu、Gondi、Lambadi
埃纳德语	3200-3327	埃纳德语、Tulu
Malalayam	3326-3455	Malalayam
泰国语	3584-3711	泰国语、Kuy、Lavna、巴利语
老挝语	3712-3839	老挝语
藏语	3840-4031	喜马拉雅语包括藏语、Ladakhi 和 Lahuli
乔治亚语	4256-4351	乔治亚语, 黑海边乔治亚前苏维埃共和国语
Hangul Jamo	4352-4607	朝鲜、韩国音节的字母组成部分
Latin 的附加扩展集	7680-7935	标准的 Latin 字母如 E 和 Y 与可识别的记号组合在一起, 除了用于越南语元音中, 很少使用
希腊语扩展集	7936-8191	希腊字母与可识别记号的组合, 用于正统的希腊语中
通用的标点符号	8192-8303	各种标点符号
上标和下标	8304-8351	普通的上标和下标
货币符号	8352-8399	货币符号, 一般在别的地方找不到
用于符号的组合记号	8400-8447	给多个字符做记号
像字母的符号	8446-8527	像字母的符号, 如™
数表	8526-8591	分数和罗马数字
箭头符号	8592-8703	箭头符号
数学符号	8704-8959	不常出现的数学运算符
技术杂项	8960-9039	APL 编程语言需要的符号和其他各种技术符号
控制图形	9216-9279	ASCII 控制字符图形, 常用于调试
光学字符识别	9280-9311	在打印支票上的 OCR-A(光学字符识别)和 MICR(磁性墨水字符识别)符号
续表		
文 字	范 围	目 的
附加字符	9312-9471	放在圆和括号中的字母和数字

画方框字符	9472-9599	用于在等间距终端上画方框的字符
块元素	9600-9631	用于 DOS 和其他用途的等间距终端图形
几何形状	9632-9727	正方形、菱形、三角形等
杂项符号	9726-9983	纸牌、象棋、占卜等
Dingbats	9984-10175	Zapf Dingbat 字符
CJK 符号和标点	12286-12351	用于中国\日本和韩国的标点符号
平假名	12352-12447	日文字母的草体.
片假名	12446-12543	非草体的日文字母, 通常用于西方的外来词汇, 像“keyboard”
汉语拼音字母	12544-12591	中国的发音字母表
Hangul Compatibility Jamo	12592-12687	与 KSC 5601 代码兼容的韩国字符
Kanbun	12686-12703	在日文中用于指示古典中文的阅读顺序的记号
括起来的 CJK 字母和月份	12800-13055	用圆和括号括起来的 Hangul 和片假名字符
CJK Compatibility	13056-13311	只用于编码 KSC 5601 和 CNS 11643 的字符
统一的 CJK 象形文字	19966-40959	用于中文、日文和韩文的 Han 象形文字
Hangul 音节	44032-55203	一种韩国音节
Surrogates	55296-57343	目前还不能使用, 将来可用于扩展 Unicode, 使它包括超过百万的字符
个人使用	57344-63743	软件开发者可以在此包含自己的术语, 与正在执行的字符不同
CJK 兼容性象形文字	63744-64255	为了保持与现有的标准的一致性如 KSC 5601, 而使用的一些汉字象形文字
字母的表现方式	64256-64335	使用于 Latin、美语和希伯来语中的连字和变种
阿拉伯表象形式	64336-65023	各种阿拉伯字符的变种
组合半记号	65056-65071	把跨越多个字符的多个可识别记号连成一个可识别的记号
CJK 兼容性形式	65072-65103	用于台湾汉字象形文字
小型变种	65104-65135	用于台湾的 ASCII 标点符号的小的版本
附加的阿拉伯表象形式	65136-65279	各种阿拉伯字符变种
半宽和全宽形式	65280-65519	能够在中文和日文的不同代码间转换的字符
特殊字符	65520-65535	字节顺序记号和零宽度的非中断性空格, 常用于 Unicode 文件的开始

#### 7.4.1 UTF-8

Unicode 使用双字节表示一个字符, 因此使用 Unicode 的英文文本文件大小是使用 ASCII 码或 Latin-1 文件的两倍。UTF-8 是一个压缩的 Unicode 版本, 使用单个字节表示最常用的字符, 即 0 到 127 的 ASCII 字符, 较少见的字符使用三个字节表示, 特制是韩国音节和汉字。如果主要使用英文, UTF-8 能够将文件压缩为原来的一半。如果主要使用汉语、朝语或者日语, UTF-8 会使文件的尺寸增加 50%; 因此应当谨慎使用 UTF-8。UTF-8 几乎不能处理非罗马文字和非 CJK 文字, 如希腊语、阿拉伯语、古斯拉夫语和希伯来语。

XML 处理器在没有被预先通知的情况下假定文本数据是 UTF-8 格式。这意味着 XML 处理器能够阅读 ASCII 码文件, 但是使用它处理其他格式的文件像 MacRoman 或者 Latin-1 会有困难。我们很快就能学会如何在短时间内解决这个问题。

#### 7.4.2 通用字符系统

Unicode 因为没有包含足够多的语言和文字而受到批评, 特别是亚洲东部的语言。它只定义了中国、日本、朝鲜和古越南使用的 80,000 象形文字中的 20,000 个左右。(现代越南语使用一种罗马字母。)

UCS (Universal Character System) 通用字符系统, 也称作 ISO 10646, 使用四个字节 (确切地说是 31 位) 表示一个字符, 以给 20 多亿不同的字符提供足够的空间。这样能容易地覆盖地球上任何一种文字和语言使用的每个字符。而且还可以给每一种语言指定一个完整的字符集, 使法语中的“e”不同于英语和德语中的“e”等等。

与 Unicode 一样，UCS 定义了许多不同的变种和压缩形式。纯粹的 Unicode 有时指 USC-2，是双字节的 UCS。UTF-16 是一种特别的编码，它把一些 UCS 字符安排在长度变化的字符串中，在这种方式下 Unicode（UCS-2）数据不会改变。

UCS 超越 Unicode 的优点主要是理论方面的。在 UCS 中实际定义过的字符就是 Unicode 中已有的字符。但是 UCS 为以后的字符扩充提供了更多的空间。

## 7.5 如何使用 Unicode 编写 XML

Unicode 是 XML 自己的字符集，至少在能得到的字体范围内，XML 浏览器会很好的显示它。但是支持全部 Unicode 的文本编辑程序不是很多。因此，不得不使用下面两种方法之一解决这个问题：

1. 使用本地字符集如 Latin-3 编写，然后把文件转换成 Unicode 文件。
2. 在文本中包含 Unicode 字符引用，它们在数值上等同于特定的字符。

在主要使用一种文字或一种文字附加 ASCII 码输入大量文本的情况下，第一种方法更可取。文档需要掺少量的多种文字时，可使用第二种方法。

### 7.5.1 利用字符引用在 XML 文件中插入字符

一个 Unicode 字符是介于 0 和 65,535 之间的一个数。如果没有使用 Unicode 书写的文本编辑程序，通常可以使用字符引用在 XML 文件中插入字符。

Unicode 字符引用由两个字符 `&#` 组成，后面跟有要插入字符的编码和分号。例如，希腊字母  $\pi$  的 Unicode 字符值是 960，因此需要在 XML 文件中插入 `&#960;`。古斯拉夫字母  $\text{ч}$  的 Unicode 值是 1206，需要在 XML 文件中插入 `&#1206;`。

Unicode 字符引用也可以用十六进制数指定，尽管多数人习惯使用十进制数，Unicode 规范中给出的字符值是双字节十六进制数。直接使用十六进制数更简单一些，不必把它们转换成十进制数。

使用十六进制数需要在 `&#` 之后添加一个 `x` 来指明。例如， $\pi$  的十六进制数是 3C0，因此插入 XML 文件中的是 `&#x03C0;`；古斯拉夫语字母  $\text{ч}$  的十六进制数是 4B6，因此在 XML 文件中的应当是 `&#x04B6;`。两个字节表示 4 个十六进制位，通常在十六进制字符引用中包含一个起始的 0 来构成 4 位十六进制数。

十六进制和十进制 Unicode 字符引用可用来嵌入那些会被解释为置标的字符。例如，与字符 (`&`) 的编码是 `&#38;` 或 `&#x0026;`，小于号 (`<`) 的编码是 `&#60;` 或 `&#x003C;`。

### 7.5.2 其他字符集与 Unicode 字符集之间的转换

输出 XML 文件的应用软件如 Adobe Framemaker，能够自动转换为 Unicode 或 UTF-8 文件。否则必须使用一种转换工具。Sun 的免费工具包 Java Development Kit (JDK) 包含一个名为 `native2ascii` 的简单命令行实用工具，能够完成多种常见和不常见的本地字符集与 Unicode 之间的转换。

例如，下面的命令把文件名是 `myfile.txt` 文本文件从操作平台默认的编码转换为 Unicode。

```
C:\>native2ascii myfile.txt myfile.uni
```

可使用 `-encoding` 选项指定其他编码。

```
C:>native2ascii -encoding Big5 chinese.txt chinese.uni
```

还可使用 `-reverse` 选项，把 Unicode 转换为本地编码。

```
C:>native2ascii -encoding Big5 -reverse chinese.uni chinese.txt
```

如果没有输出文件名，转换后的文件将打印输出。

native2ascii 程序同样能处理 java 类型的 Unicode 转义符，它们是以 \u09E3 的格式嵌入的。这与 XML 中的数值字符引用不同，尽管比较相似。使用 native2ascii 把文件转化为 Unicode，仍然可以使用 XML 字符引用 &#0;&#0; 查看程序能够识别它们。

### 7.5.3 如何使用其他字符集编写 XML

在没有被预先告知的情况下，XML 处理器默认文本实体字符使用 UTF-8 编码，因为 ASCII 码是包含在 UTF-8 中的一个子集，所以 XML 处理器同样可以分析 ASCII 码文本。

除了 UTF-8，XML 处理器必须能读懂的唯一字符集是原始 Unicode。当不能把文本转换成 UTF-8 或原始 Unicode 时，可以使文本保持原样并告诉 XML 处理器文本所使用的字符集。这是最后一种手段，因为这样做并不能保证一个尚未成熟的 XML 处理器能够处理其他编码。除此之外，Netscape Navigator 和 Internet Explorer 都能很好地解释常见的字符集。

在文件开始的 XML 声明中包含一个 encoding 属性，告诉 XML 处理器正在使用的是非 Unicode 编码。例如，说明整个文档使用默认的 Latin-1（除非在嵌套的实体中有别的处理指令），可使用下面的 XML 声明：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

也可以在 XML 声明之后包含一个编码声明作为一个单独的处理指令，但是一定要在所有字符数据之前：

```
<?xml encoding="ISO-8859-1"?>
```

表 7-7 列出了目前大部分常用的字符集的正式名称，即出现在 XML 编码属性中的名称。清单中没有的编码请参考由 Internet Assigned Numbers Authority (IANA) 提供的正式清单，网址是：  
<http://www.isi.edu/in-notes/iana/assignments/character-sets>。

表 7-7 常用字符集名称

字符集名称	语言/国家
US-ASCII	英语
UTF-8	压缩 Unicode
UTF-16	压缩 UCS
ISO-10646-UCS-2	原始 Unicode
ISO-10646-UCS-4	原始 UCS
ISO-8859-1	Latin-1, 西欧
ISO-8859-2	Latin-2, 东欧
ISO-8859-3	Latin-3, 南欧
ISO-8859-4	Latin-4, 北欧
ISO-8859-5	ASCII 码加古斯拉夫语
ISO-8859-6	ASCII 码加阿拉伯语
ISO-8859-7	ASCII 码加希腊语
ISO-8859-8	ASCII 码加希伯来语
ISO-8859-9	Latin-5, 土耳其语
ISO-8859-10	Latin-6, ASCII 码加北欧语
ISO-8859-11	ASCII 码加泰国语
ISO-8859-13	Latin-7, ASCII 码加波罗地海周边语言和独特的拉托维亚语
ISO-8859-14	Latin-8, ASCII 码加盖尔语和威尔士语
ISO-8859-15	Latin-9, Latin-0, 西欧
ISO-2022-JP	日语
Shift_JIS	日文版 Windows
EUC-JP	日文版 Unix
Big5	中国台湾地区, 汉语
GB2312	中国大陆, 汉语
KOI6-R	俄罗斯
ISO-2022-KR	韩语
EUC-KR	韩语版 Unix
ISO-2022-CN	汉语

## 7.6 本章小结

在本章中你会了解到以下内容：

- Web 页面应当指明使用的编码。
- 什么是文字，文字与语言有何关系，以及与文字有关的四个因素。
- 在计算机中如何使用文字、字符集、字体、字形和输入法。
- 什么样的字符集通常使用在以 ASCII 码为基础的不同操作平台上。
- 在没有 Unicode 编辑程序的情况下，如何使用 Unicode 编写 XML 文件（使用 ASCII 码和 Unicode 字符引用编写文档）。
- 使用别的编码编写 XML 文件时，应当在 XML 声明中包含一个 `encoding` 属性。

在下一章将研究 DTD 和如何使用 DTD 给文档定义及执行词汇表、句法和语法并强制执行。

## 第二部分 文档类型定义

本部分包括以下各章：

第 8 章——文档类型定义和合法性

第 9 章——实体和外部 DTD 子集

第 10 章——DTD 中的属性声明

第 11 章——嵌入非 XML 数据

### 第 8 章 文档类型定义和合法性

XML 被作为一种元标记语言，是一种描述标记语言的语言。在本章中您将学到如何说明和描述所创建的新标记语言。这些新的标记语言（也叫标记集）要通过文档类型定义（DTD）来定义，这正是本章要讲述的内容。各个文档要与 DTD 相比较，这一过程称为合法性检验。如果文档符合 DTD 中的约束，这个文档就被认为是合法的，否则就是不合法的。

- 本章的主要内容包括：
- 文档类型定义（DTD）
- 文档类型声明
- DTD 的合法性
- 元素清单
- 元素声明
- DTD 中的说明
- 可在文档间共享的通用 DTD

#### 8.1 文档类型定义

缩略词 DTD 代表文档类型定义。一项文档类型定义应规定元素清单、属性、标记、文档中的实体及其相互关系。DTD 为文档结构制定了一套规则。例如，一项 DTD 指定一个 BOOK 元素有一个 ISBN 子元素、一个 TITLE 子元素、一个或多个 AUTHOR 子元素，有或没有 SUBTITLE。DTD 以元素、实体、属性和记号的标记声明来做到这一点。



本章重点是元素声明。第 9、10、11 章分别介绍实体、属性和标记。

DTD 可以包括在包含它描述的文档的文件中，或者与外部的 URL 相链接。这些外部 DTD 可以被不同文档和网站所共享。DTD 为应用程序、组织和兴趣组提供了共同遵循的方法，同时也以文档形式阐述了标记标准并强制遵守此标准。

例如，为了使一部书易于排版，出版商会要求作者遵循一定的格式。作者可能不管是否与本章前面的小标题列出的关键点相符合，而只管成行地写下去。如果作者用 XML 写作，那么出版商就能很容易地检查出作者是否遵守了 DTD 作出的预定格式，甚至找出作者在那里以及怎样偏离了格式。这比指望编辑们单纯地从形式上通读文档而找出所有偏离格式的地方要容易得多。

DTD 有助于不同的人的程序互相阅读文件。例如，如果化学家们通过专业机构（如美国化学协会）为中介同意将单一的 DTD 用于基本的化学记号，那么他们就能够阅读和理解他们当中任何人的文章。DTD 精确地定义了什么允许或不允许在文档中出现。DTD 还为查看和编辑软件必须支持的元素建立了标准。更重要的是，它建立了超出 DTD 声明的非法范围。这就使它有助于防止软件商乘机利用和扩展开放协议以便将用户锁定在他们的专利软件上。



而且，DTD 可以在没有实际数据的情况下展现出页面上的不同元素是如何安排的。DTD 使人们能脱离实际数据看到文档结构。这意味着可以将许多有趣的样式和格式加在基本结构上，而对基本结构毫无损害。这正如涂饰房子而不必改变基本的建筑计划。页面的读者可能看不见甚至不了解基础结构，但是只要有了 DTD，任何人类作者和 JavaScript 程序、DTD 程序、小服务程序、数据库和其他程序就可以使用它。

用 DTD 还可以做更多的事。可以使用它们来定义词汇实体以插入署名块或地址一类的模板文本。您可以确定输入数据的人们是否遵循了您的格式。您可以从关系数据库或对象数据库中移出数据或把数据送往目标数据库。甚至可以用适当的 DTD 利用 XML 作为中间格式来转换不同的格式。所以让我们开始看一看 DTD 到底是什么样的。

## 8.2 文档类型声明

文档类型声明指定了文档使用的 DTD。文档类型声明出现在文档的序言部分，处在 XML 声明之后和基本元素之前。它可能包括文档类型定义或是标识文档类型定义所在文档的 URL。有些情况下文档类型定义有内外两个子集，则文档类型声明可能同时包括以上两种情况。



文档类型声明同文档类型定义不是一回事。只有文档类型定义缩写为 DTD。文档类型声明必须包含或者引用文档类型定义，但文档类型定义从不包括文档类型声明。我同意这造成了不必要的混乱。遗憾的是 XML 似乎与这术语密不可分，幸运的是多数情况下二者的区别并不重要。

请回顾一下第 3 章清单 3-2 (greeting.xml)，如下所示：

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

```
<GREETING>
```

```
Hello XML!
```

```
</GREETING>
```

这个文档包含单一元素 GREETING。（请记住，`<? xml version="1.0" standalone="yes"? >` 是一条处理指令，不是元素。）清单 8-1 显示了这一文档，但这次带有文档类型声明。文档类型声明声明了基本元素是 GREETING。文档类型声明也包含文档类型定义，它声明了 GREETING 元素包含可析的字符数据。

清单 8-1：带有 DTD 的 Hello XML

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

```
<!DOCTYPE GREETING [
```

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

```
]>
```

```
<GREETING>
```

```
Hello XML!
```

```
</GREETING>
```

清单 3-2 与清单 8-1 的唯一区别在于清单 8-1 增加了 3 行：

```
<!DOCTYPE GREETING [
```

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

```
]>
```

这几行是清单 8-1 的文档类型声明。文档类型声明在 XML 声明与文档本身之间。XML 声明与文档类型声明统称为文档序言 (Prolog)。在本例中, <?xml version="1.0" standalone="yes"?> 是 XML 声明; <!DOCTYPE GREETING [ <!ELEMENT GREETING (#PCDATA)> ]> 是文档类型声明; <!ELEMENT GREETING (#PCDATA)> 是文档类型定义; <GREETING> Hello XML! </GREETING> 是文档或基本元素。

文档类型声明以 <!DOCTYPE 为开始, 以 ]> 结束。通常将开始和结束放在不同的行上, 但断行和多余的空格并不重要。同一文档类型声明也可以写成一行:

```
<!DOCTYPE GREETING [ <!ELEMENT GREETING (#PCDATA)> ]>
```

本例中基本元素名称——GREETING 跟在 <!DOCTYPE 之后。这不仅是一个名称, 也是一项要求。任何带有这种文档类型声明的合法文档必须有基本元素。在 [和] 之间的内容是文档类型定义。

DTD 由一系列声明了特写的元素、实体和属性的标记声明所组成。其中的一项声明基本元素。清单 8-1 中整个 DTD 只是如下简单的一行:

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

通常情况下 DTD 当然会更长更复杂。

单个行 <!ELEMENT GREETING (#PCDATA)> (正如 XML 中的大多数对象一样是区分大小写的) 是一项元素类型声明。在本例中, 声明的元素名称是 GREETING。它是唯一的元素。这一元素可以包含可析的字符数据 (或 #PCDATA)。可析的字符实质上是除标记文本外的任何文本。这也包括实体引用如 &amp; ;, 在对文档进行语法分析时, 实体引用就被文本所取代。

可以把这一文档像通常一样装入一种 XML 浏览器中。图 8-1 显示了清单 8-1 在 Internet Explorer 5.0 中的情况。结果可能正如人们所料, 文档源以可折叠的大纲视图出现。Internet Explorer 使 <!DOCTYPE GREETING (View Source for full doctype...)> 一行变蓝指明有文档类型声明。

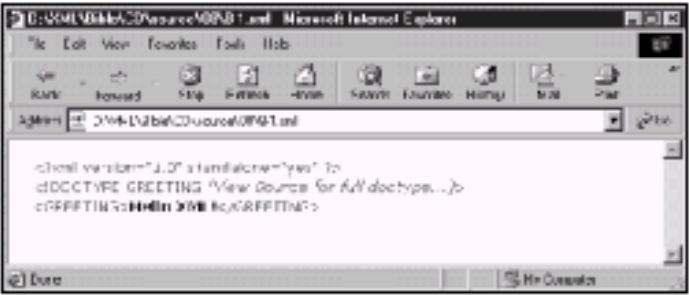


图 8-1 Internet Explorer 5.0 中显示的带有 DTD 的 Hello XML

当然, 文档可以与样式单结合起来, 就像第 3 章的清单 3-6 中一样。实际上可以用同一个样式单。如清单 8-2 所示, 只要在序言中增加通常的 <?xml-stylesheet?> 处理指令。

清单 8-2: 带有 DTD 和样式单的 Hello XML

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

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

<!DOCTYPE GREETING [

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

]>

<GREETING>

Hello XML!

</GREETING>

图 8-2 显示的是结果网页。这同第 3 章中没有 DTD 的图 3-3 相同。格式化时通常不考虑 DTD。



图 8-2 Internet Explorer 5.0 所示的带 DTD 和样式单的 Hello XML

## 8.3 根据 DTD 的合法性检验

一个合法的文档必须符合 DTD 指定的约束条件。而且，它的基本元素必须是在文档类型声明中指明的。清单 8-1 中的文档类型声明和 DTD 说明一个合法的文档必须是这样的：

```
<GREETING>
```

```
various random text but no markup
```

```
</GREETING>
```

一个合法的文档不能是这样的：

```
<GREETING>
```

```
<someTag>various random text</someTag>
```

```
<someEmptyTag/>
```

```
</GREETING>
```

也不能是这样的：

```
<GREETING>
```

```
<GREETING>various random text</GREETING>
```

```
</GREETING>
```

这个文档必须由放在<GREETING>开始标记和</GREETING>结束标记之间的可析的字符所组成。与只是结构完整的文档不同，合法文档不允许使用任意的标记。使用的任何标记都要在 DTD 内声明。而且，必须以 DTD 允许的方式使用。在清单 8-1 中，<GREETING>标记只能用作基本元素的开始，且不能嵌套使用。

假设我们对清单 8-2 做一点变动，以<foo>和</foo>替换<GREETING>和</GREETING>标记，如清单 8-3 所示。清单 8-3 是合法的。它是一个结构完整的 XML 文档，但它不符合文档类型声明和 DTD 中的约束条件。

清单 8-3：不符合 DTD 规则的不合法的 Hello XML

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

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

```
<!DOCTYPE GREETING [
```

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

```
]>
```

```
<foo>
```

```
Hello XML!
```

</foo>



不是所有的文档都必须合法，也不是所有的语法分析程序都检查文档的合法性。事实上，多数 Web 浏览器包括 IE 5 和 Mozilla 都不检查文档的合法性。

进行合法性检查的语法分析程序读取 DTD 并检查文档是否合乎 DTD 指定的规则。如果是，则分析程序将数据传送到 XML 应用程序（如 Web 浏览器和数据库）。如果分析程序发现错误，它将报告出错。如果手工编写 XML，应在张贴前检查文档的合法性以确保读者不会遇到错误。

在 Web 上可找到几十种不同的进行合法性检查的语法分析程序。其中多数是免费的。大多数是以库文件的形式存在的接近完成的产品，以便程序员可将其结合到自己的程序中。这些产品用户界面（如果有的话）较差。这类分析程序包括 IBM 的 alphaWorks' XML for Java、Microsoft 和 DataChannel 的 XJParser 和 Silfide 的 SXP。

XML for Java: <http://www.alphaworks.ibm.com/tech/xml>

XJParser: [http://www.datachannel.com/xml\\_resources/](http://www.datachannel.com/xml_resources/)

SXP: <http://www.loria.fr/projets/XSilfide/EN/sxp/>

一些库文件也包括在命令行上运行的独立的分析程序。这些程序读取 XML 文件并报告发现的错误，但不加以显示。例如，XJParse 是一个 Java 程序，包括在 IBM 的 Samples. XJParse 软件包中的 XML for Java 1.1.16 类库中。要运行这一程序，必须首先将 XML for Java 的 jar 文件添加到 Java 类库的路径上。然后就可以打开 DOS 窗口或外壳程序提示符，向 XJParse 程序传送要检查合法性的文档的本地文件名或远程 URL，以便对文档进行检查，如下所示：

```
C:\xml4j>java samples.XJParse.XJParse -d D:\XML\08\invalid.xml
```



本书写作时，IBM 的 alphaWorks 推出了 XML for Java 的 2.0.6 版本。在这一版本下，启动的只是 XJParse 而非 Samples. XJParse。但是，1.1.16 版本提供了更多的用于独立检查的功能。

您可以使用 URL 代替文件名，如下所示：

```
C:\xml4j>java samples.XJParse.XJParse -d
```

```
http://metalab.unc.edu/books/bible/examples/08/invalid.xml
```

在任一情况下，XJParse 将列出发现的错误后跟树状结构的文档作为反应。例如：

```
D:\XML\07\invalid.xml: 6? 4: Document root element, "foo", must
```

```
match DOCTYPE root , "GREETING".
```

```
D:\XML\07\invalid.xml: 8, 6: Element "<foo>" is not valid in
```

```
this context.
```

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

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

```
<!DOCTYPE GREETING [
```

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

```
]>
```

```
<foo>
```

```
Hello XML!
```

```
</foo>
```

这个输出不是特别吸引人。但是，像 XJParse 这样的合法性检查程序的目的是显示 XML 文件。相反，分析程序的任务是把文档分成为树状结构并把树的结点传送给显示数据的程序。这个程序可能是 Netscape Navigator 或 Internet Explorer 等 Web 浏览器。也可能是一个数据库。甚至可能是自己写成的定制程序。使用 XJParse 或其他命令行合法性分析程序来验证是否编写了其他程序可以处理的良好 XML。实质上这是一种校对或质量保证阶段而不是最后的输出。

因为 XML for Java 和多数合法性分析程序是用 Java 写成的，它们也就具有跨平台的 Java 程序的所有缺点。首先，在能够运行分析程序之前必须安装 Java 开发工具（JDK）或 Java 运行环境。其次，需要将 XML for Java 的 jar 文件添加到类路径上。这两项工作都不是太简单。它们都不是为非程序员的最终用户设计的。这些工具有点设计欠佳，使用不便。

如果正在为浏览器编写文档，验证文档的最简易方法是把文档装入浏览器看一看报告出什么错误。但是并不是所有的浏览器都对文档进行合法性检查，某些浏览器仅接受结构完整的文档，而不管其合法性如何。Internet Explorer 5.0 β 2 版对文档进行合法性检查，但正式发行版都不进行了。

如果将文档装入 Web 服务器且无需特别保密，基于 Web 的合法性检查程序是一种替代方法。这些分析程序只需要以简单的形式输入文档的 URL。它们明显的优点是不需要面对 Java 运行软件、类路径和环境变量等麻烦。

图 8-3 显示的是 Richard Tobin 的基于 RXP 的以 Web 为宿主的 XML 结构完整性和合法性检查程序。可以在 <http://www.cogsci.ed.ac.uk/%7ERichard/xml-check.html> 处找到此程序。图 8-4 显示的是使用这一程序检查清单 8-3 显示出的错误结果。

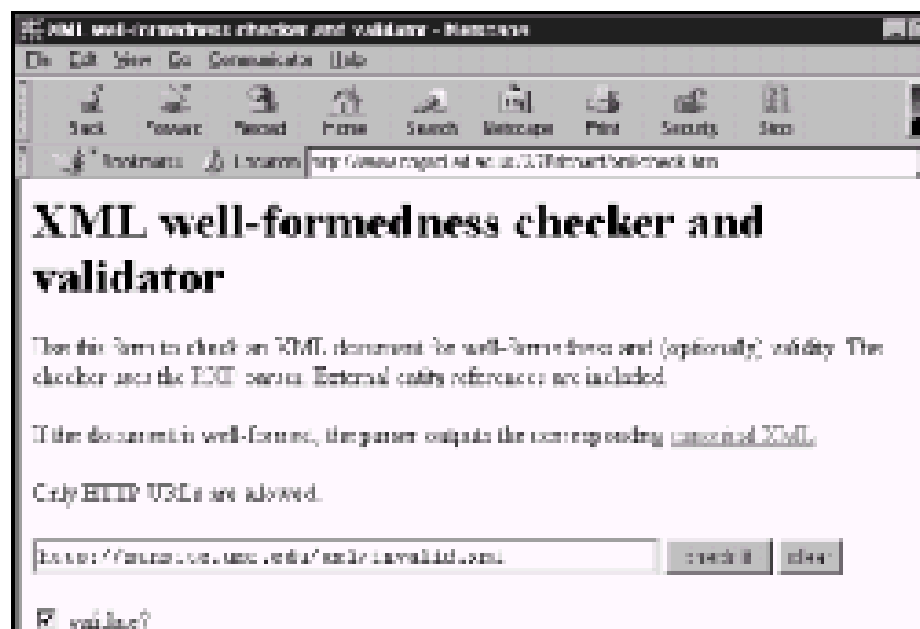


图 8-3 Richard Tobin 的基于 RXP 的以 Web 为宿主的 XML 结构完整性和合法性检查程序

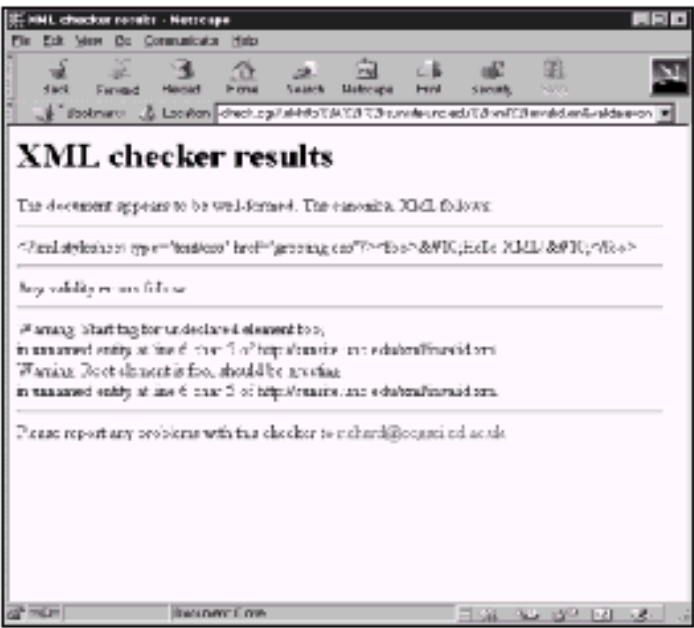


图 8-4 Richard Tobin 的 XML 合法性检查程序报告的清单 8-3 中的错误

布朗大学的 Scholarly Technology Group 在 <http://www.stg.brown.edu/>

service/xmlvalid/处提供了一种检查程序。这一程序以允许从本地计算机上载文件而不必把文件装入公共服务器而著称。如图 8-5 所示，图 8-6 显示了用这一程序检查清单 8-3 的结果。

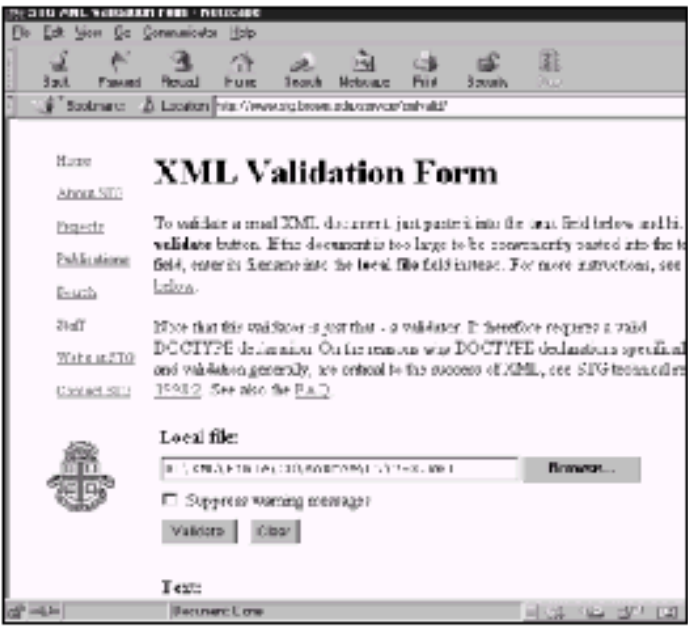


图 8-5 布朗大学的 Scholarly Technology Group 的以 Web 为宿主的 XML 合法性检查程序



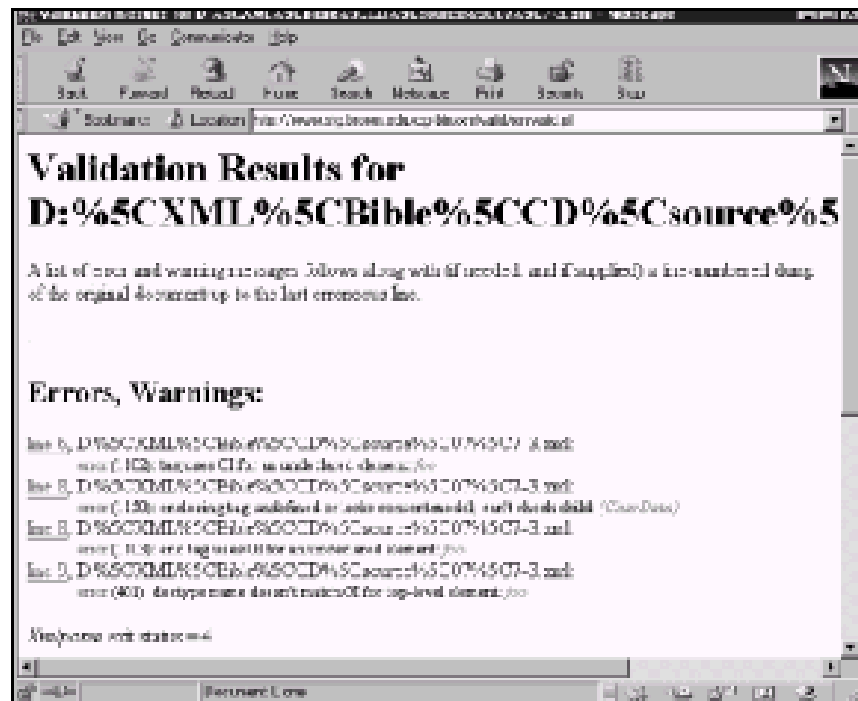


图 8-6 布朗大学的 Scholarly Technology Group 的合法性检查程序报告的清单 8-3 中的错误

## 8.4 列出元素

要为一个文档创建适当的 DTD 的第一步是了解用 DTD 中定义的元素编码的信息结构。有时候信息就像通讯地址列表一样。有时则具有相对自由的形式，如说明短文或杂志文章。

让我们以已经相对结构化的文档为例，回到第 4 章所示的棒球统计示例中。在那份文档上加一个 DTD，就使我们能把以前只有通过约定才能遵守的约束条件付诸实施。例如，我们可以要求 SEASON 元素包含正好两个 LEAGUE 子元素，每个 TEAM 有 TEAM\_CITY 和 TEAM\_NAME 子元素，并且 TEAM\_CITY 总在 TEAM\_NAME 之前。

回想起来，完整的棒球统计文档包含下面一些元素：

SEASON RBI

YEAR STEALS

LEAGUE CAUGHT—STEALING

LEAGUE—NAME SACRIFICE\_ HITS

DIVISION SACRIFICE\_FLIES

DIVISION\_NAME ERRORS

TEAM WALKS

TEAM\_CITY STRUCK\_OUT

TEAM\_NAME HIT\_BY\_PITCH

PLAYER COMPLETE\_GAMES

SURNAME SHUT\_OUTS

GIVEN\_NAME ERA

POSITION INNINGS

GAMES HOME\_RUNS

GAMES\_STARTED RUNS

AT\_BATS EARNED\_RUNS

RUNS HIT\_BATTER

HITS WILD\_PITCHES

DOUBLES BALK

TRIPLES WALKED\_BATTER

HOME\_RUNS STRUCK\_OUT\_BATTER

WINS COMPLETE\_GAMES

LOSSES SHUT\_OUTS

SAVES

所编写的 DTD 要为每个元素作元素声明。每一元素声明列出元素名和它的子元素。例如，DTD 规定一个 LEAGUE 元素有三个 DIVISION 子元素。还规定 SURNAME 要放在 PLAYER 之内而不能在它外面。还规定每个 DIVISION 有不确定的 TEAM 元素数目但绝不能少于一个。

DTD 可要求 PLAYER 只有一个 GIVEN\_NAME、SURNAME、POSITION 和 GAMES 元素，但是否有 RBI 和 ERA 元素则任意。而且要求 GIVEN\_NAME、SURNAME、POSITION 和 GAMES 元素以特定的顺序出现。例如，GIVEN\_NAME、SURNAME、POSITION 和 GAMES 只能在 PLAYER 元素内使用。

如果头脑中有一份具体的结构完整的示例文档，该文档使用了想要出现在 DTD 中所有的元素，那么开始就很容易了。第 4 章中的例子在这里就可起这一作用。清单 8-4 是经过整理的第 4 章清单 4-1 的简化版。尽管它只有两名球员，却可以说明所有基本的元素。

清单 8-4：需要编写 DTD 结构完整的 XML 文档

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

```
<SEASON>
```

```
<YEAR>1998</YEAR>
```

```
<LEAGUE>
```

```
    <LEAGUE_NAME>National</LEAGUE_NAME>
```

```
    <DIVISION>
```

```
        <DIVISION_NAME>East </DIVISION_NAME>
```

```
            <TEAM>
```

```
                <TEAM_CITY>Florida</TEAM_CITY>
```

```
                <TEAM_NAME>Marlins</TEAM_NAME>
```

```
            <PLAYER>
```

```
                <SURNAME>Ludwick</SURNAME>
```

```
                <GIVEN_NAME>Eric</GIVEN_NAME>
```

```
                <POSITION>Starting Pitcher</POSITION>
```

```
            <WINS>1</WINS>
```

```
            <LOSSES>4</LOSSES>
```

<SAVES>0</SAVES>

<GAMES>13</GAMES>

<GAMES\_STARTED>6</GAMES\_STARTED>

<COMPLETE\_GAMES>0</COMPLETE\_GAMES>

<SHUT\_OUTS>0</SHUT\_OUTS>

<ERA>7.44</ERA>

<INNINGS>32.2</INNINGS>

<HOME\_RUNS>46</HOME\_RUNS>

<RUNS>7</RUNS>

<EARNED\_RUNS>31</EARNED\_RUNS>

<HIT\_BATTER>27</HIT\_BATTER>

<WILD\_PITCHES>0</WILD\_PITCHES>

<WALKED\_BATTER>0</WALKED\_BATTER>

<STRUCK\_OUT\_BATTER>17</STRUCK\_OUT\_BATTER>

</PLAYER>

<PLAYER>

<SURNAME>Daubach</SURNAME>

<GIVEN\_NAME>Brian</GIVEN\_NAME>

<POSITION>First Base</POSITION>

<GAMES>10</GAMES>

<GAMES\_STARTED>3</GAMES\_STARTED>

<AT\_BATS>15</AT\_BATS>

<RUNS>0</RUNS>

<HITS>3</HITS>

<DOUBLES>1</DOUBLES>

<TRIPLES>0</TRIPLES>

<HOME\_RUNS>0</HOME\_RUNS>

<RBI>3</RBI>

<STEALS>0</STEALS>

<CAUGHT\_STEALING>0</CAUGHT\_STEALING>

<SACRIFICE\_HITS>0</SACRIFICE\_HITS>

<SACRIFICE\_FLIES>0</SACRIFICE\_FLIES>

<ERRORS>0</ERRORS>

<WALKS>1</WALKS>

<STRUCK\_OUT>5</STRUCK\_OUT>

<HIT\_BY\_PITCH>1</HIT\_BY\_PITCH >

</PLAYER>

</TEAM>

<TEAM>

<TEAM\_CITY>Montreal</TEAM\_CITY>

<TEAM\_NAME>Expos</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>New York</TEAM\_CITY>

<TEAM\_NAME>Mets</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Philadelphia</TEAM\_CITY>

<TEAM\_NAME>Phillies</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>Central</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Chicago</TEAM\_CITY>

<TEAM\_NAME>Cubs</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>West</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Arizona</TEAM\_CITY>

<TEAM\_NAME>Diamondbacks</TEAM\_NAME>

</TEAM>

</DIVISION>

</LEAGUE>

<LEAGUE>

<LEAGUE\_NAME>American</LEAGUE\_NAME>

<DIVISION>

<DIVISION\_NAME>East </DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Baltimore</TEAM\_CITY>

<TEAM\_NAME>Orioles</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>Central</DIVISION\_NAME>

<TEAM>

```
<TEAM_CITY>Chicago</TEAM_CITY>

<TEAM_NAME>White Sox</TEAM_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION_NAME>West</DIVISION_NAME>

<TEAM>

<TEAM_CITY>Anaheim</TEAM_CITY>

<TEAM_NAME>Angels</TEAM_NAME>

</TEAM>


</DIVISION>

</LEAGUE>

</SEASON>
```

表 8-1 列出了本例中的元素及它们必须遵守的条件。每一元素都有它必须包含的元素、它可能包含的元素以及必须包含它的元素。有些情况下，一个元素可能包含不止一个同一类型的子元素。SEASON 元素包含一个 YEAR 和两个 LEAG UE 元素。一个 DIVISION 通常包含不止一个 TEAM 元素。较不明显的是，一些击球手在各场比赛中在指定的投球手和外场之间交替出现。这样，一个 PLAYER 元素就可能有不止一个 POSITION。在该表格中，要求的子元素的数目是通过在元素前加数字来指明的（如 2 LEAGUE），而多子元素的可能性是通过在元素名尾加（s）指明的，如 PLAYER（s）。

清单 8-4 遵守了这些条件。如果把两个 PLAYER 元素和一些 TEAM 元素省略，文档可以短些。如果包括进其他一些 PLAYER 元素，文档就会长些。但是其他元素的位置都不能变动。



XML 元素有两种基本类型。简单元素包含文本，也就是所谓的可析字符数据，即上下文中的#PCDATA 或 PCDATA。复合元素包含其他元素，有的还包含文本和其他元素。标准 XML 没有整数、浮点、日期或其他数据类型。因而不能使用 DTD 说明走步数一定是一个非负的整数，或 ERA 一定是 0.0 和 1.0 之间的一个浮点数，尽管在如本例一样的例子中这样做是有用的。有人做过努力来定义一种方案，以便使用 XML 句法描述传统上 DTD 中编码的信息以及数据类型信息。直到 1999 年中期，这些努力仍主要是理论上的，很少有实际的实现方式。

表格 8-1 棒球统计中的元素

元素	必须包含的元素	可能包含的元素	必须包含它的元素
SEASON	YEAR	2 LEAGUE	
YEAR	文本		SEASON
LEAGUE	LEAGUE_NAME,  3 DIVISION		SEASON

LEAGUE_NAME	文本		LEAGUE
DIVISION	DIVISION_NAME, TEAM	TEAM(s)	LEAGUE
DIVISION_NAME	文本		DIVISION
TEAM	TEAM_CITY, TEAM_NAME	PLAYER(s)	DIVISION
TEAM_CITY	文本		TEAM
TEAM_NAME	文本		TEAM
PLAYER	SURNAME, GIVEN_NAME, POSITION, GAMES	GAMES_STARTED, AT_BATS, RUNS, HITS, DOUBLES, TRIPLES, HOME_RUNS, RBI, STEALS, CAUGHT_STEALING, SACRIFICE_HITS, SACRIFICE_FLIES, ERRORS, WALKS, STRUCK_OUT, HIT_BY_PITCH, COMPLETE_GAMES, SHUT_OUTS, ERA, INNINGS, HIT_BATTER, WILD_PITCHES, BALK, WALKED_BATTER, STRUCK_OUT_BATTER	TEAM
SURNAME	文本		PLAYER
GIVEN_NAME	文本		PLAYER
POSITION	文本		PLAYER
GAMES	文本		PLAYER
GAMES_STARTED	文本		PLAYER
AT_BATS	文本		PLAYER
RUNS	文本		PLAYER
HITS	文本		PLAYER
DOUBLES	文本		PLAYER
TRIPLES	文本		PLAYER
HOME_RUNS	文本		PLAYER
RBI	文本		PLAYER
STEALS	文本		PLAYER
CAUGHT_STEALING	文本		PLAYER
SACRIFICE_HITS	文本		PLAYER
SACRIFICE_FLIES	文本		PLAYER
ERRORS	文本		PLAYER
WALKS	文本		PLAYER
STRUCK_OUT	文本		PLAYER



HIT_BY_PITCH	文本		PLAYER
COMPLETE_GAMES	文本		PLAYER
SHUT_OUTS	文本		PLAYER
ERA	文本		PLAYER
INNINGS	文本		PLAYER
HOME_RUNS_AGAINST	文本		PLAYER
RUNS_AGAINST	文本		PLAYER
HIT_BATTER	文本		PLAYER
WILD_PITCHES	文本		PLAYER
BATTER	文本		PLAYER
STRUCK_OUT_BATTER	文本		PLAYER

既然已经标识了要存储的数据，以及这些元素间可选的和必然的关系，就可以为简明概括那些联系的文档建立 DTD 了。

从一个 DTD 剪切和粘贴到另一个往往是很可行和方便的。许多元素可以在其他上下文中再使用。例如，对 TEAM 的描写同样可应用于足球、曲棍球和很多其他在队间进行的运动。

可以把一个 DTD 包括在另一个之内，这样文档就可以从两个 DTD 中得到标记。例如，可以使用一份详细地描写单个队员的统计数据的 DTD 然后把该 DTD 嵌套在更广泛的球队运动的 DTD 内。如想从棒球转换到足球，只要简单地把棒球球员 DTD 换为足球球员 DTD 就可以了。



为达到此目的，包含 DTD 的文档就被定义为外部实体。外部参数实体引用将在第 9 章“实体”中讨论。

## 8.5 元素声明

在合法的 XML 文档中使用的每项标记都要在 DTD 中的元素声明中加以声明。一项元素声明指明了元素名称和元素可能的内容。内容清单有时称为内容规格。内容规格使用一种简单的语法精确地指明文档中允许什么和不允许什么。这听起来复杂，却只需在元素名称上加上如\*、? 或+的标点以便指明它可能出现不止一次，可能出现或可能不出现，或必须出现至少一次。

DTD 很保守，没有明确允许的就是禁止的。然而，DTD 句法使您能够严格地区分那些用语句很难说清的关系。例如，DTD 很容易地说明 GIVEN\_NAME 要在 SURNAME 前，而 SURNAME 必须放在 POSITION 前，POSITION 要放在 GAME 前，GAME 要放在 GAMES\_STARTED 前，GAMES\_STARTED 要放在 AT\_BATS 前，AT\_BATS 要放在 RUNS 前，RUNS 要在 HITS 前，所有这些只能出现在一个 PLAYER 元素内。

从外到内，逐级建立 DTD 是最容易的。这使您能在建立 DTD 的同时建立一份样本文档来验证 DTD 本身是合法的和真正地描述您想要的格式。

### 8.5.1 ANY

要做的第一件事是标识基本元素。在棒球的例子中，SEASON 是基本元素。!DOCTYPE 声明指明了这一点：

```
<!DOCTYPE SEASON [  
  
]>
```

但是，这仅仅是说基本标记是 SEASON，而没有提到元素能或不能包含的内容，这就是为什么接下来要在元素声明中声明 SEASON 元素。这可通过下列一行代码来实现：

```
<!ELEMENT SEASON ANY>
```

所有的元素类型声明都以<!ELEMENT（区分大小写）开头而以>结束。其中包括声明的元素名称（本例中为 SEASON）后接内容规格。关键词 ANY（也要区分大小写）表明所有可能的元素以及可析的字符数据都可以是 SEASON 元素的子元素。

基本元素使用 ANY 是通常的作法——尤其是对未结构化的文档——但对多数其他元素则应避免使用 ANY。通常每项标记的内容应尽可能准确。DTD 经常是在整个开发过程中逐步完善的，随着反映应用情况和首制作中未预料的情况，严格性将减少。所以，最好是开始时严格，以后再放松些。

### 8.5.2 #PCDATA

尽管文档中可以出现任何元素，但出现的元素必须声明。第一个需要声明的元素是 YEAR，下面是 YEAR 元素的元素声明：

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

该声明说明 YEAR 只能包含可析的字符数据，即非标记文本，但它不能包含自己的子元素。所以，下面这个 YEAR 元素是合法的：

```
<YEAR>1998</YEAR>
```

以下这些 YEAR 元素都是合法的：

```
<YEAR>98</YEAR>
```

```
<YEAR>1998 C. E. </YEAR>
```

<YEAR>

The year of our lord one thousand,

nine hundred, & ninety-eight

</YEAR>

甚至下面这个 YEAR 元素也是合法的，因为 XML 不会去检查 PCDATA 的内容，只要是不包括标记的文本就可以。

<YEAR>Delicious, delicious, oh how boring</YEAR>

但是，下面的 YEAR 元素是非法的，因为它包含了子元素：

<YEAR>

<MONTH>January</MONTH>

<MONTH>February</MONTH>

<MONTH>March</MONTH>

<MONTH>April</MONTH>

<MONTH>May</MONTH>

<MONTH>June</MONTH>

<MONTH>July</MONTH>

<MONTH>August</MONTH>

<MONTH>September</MONTH>

<MONTH>October</MONTH>

<MONTH>November</MONTH>

<MONTH>December</MONTH>

</YEAR>

SEASON 和 YEAR 元素声明应包括在文档类型声明中，如下所示：

<!DOCTYPE SEASON [

<!ELEMENT SEASON ANY>

<!ELEMENT YEAR (#PCDATA)>

]>

通常，空格和缩进无关紧要。元素声明的顺序也不重要。下面这一文档类型声明的作用与上面的声明相同：

```
<!DOCTYPE SEASON [  
  
  <!ELEMENT YEAR (#PCDATA)>  
  
  <!ELEMENT SEASON ANY>  
  

```

上面两个文档声明都是说一个 SEASON 元素可以包含可析的字符数据和以任意顺序声明的任意数量的其他元素。本例中如此声明的元素只有 YEAR，它只能包含可析的字符数据。例如考虑清单 8-5 中的文档。

清单 8-5：一个合法的文档

```
<?xml version="1.0" standalone="yes"?>  
  
<!DOCTYPE SEASON [  
  
  <!ELEMENT YEAR (#PCDATA)>  
  
  <!ELEMENT SEASON ANY>  
  
  
<SEASON>  
  
  <YEAR>1998</YEAR>  
  
</SEASON>
```

因为 SEASON 元素也可以包含可析的字符数据，所以可以在 YEAR 元素之外附加文本，如清单 8-6 所示。

清单 8-6：包含 YEAR 元素和正常文本的合法的文档

```
<?xml version="1.0" standalone="yes"?>  
  
<!DOCTYPE SEASON [  
  
  <!ELEMENT YEAR (#PCDATA)>  
  
  <!ELEMENT SEASON ANY>  
  
  
<SEASON>  
  
  <YEAR>1998</YEAR>  
  
  Major League Baseball  
  
</SEASON>
```

我们最后是不接受这样的文档的。但是此时它是合法的，因为 SEASON 被声明为可以接受 ANY 内容。大多数时候，在定义一个元素的所有子元素之前以 ANY 代替一个元素，就比较容易起步。然后再用实际的子元素来替换 ANY。

您可以向清单 8-6 上附加一份简单的样式单，如第 4 章中创建的 baseballstats.css，如清单 8-7 所示。然后将其装入 Web 浏览器，结果显示在图 8-7 中。baseballstats.css 样式单包含一些没有出现在 DTD 或是清单 8-7 列出的文档部分中的元素的样式规则，但这没有问题。Web 浏览器会忽略任何文档中没有的元素的样式规则。

清单 8-7：包含样式单、一个 YEAR 元素和正常文本的合法文档

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

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

<!DOCTYPE SEASON [

<!ELEMENT YEAR (#PCDATA)>

<!ELEMENT SEASON ANY>

]>

<SEASON>

<YEAR>1998</YEAR>

Major League Baseball

</SEASON>
```



图 8-7 在 Internet Explorer 5.0 中显示的包含样式单、YEAR 元素和正常文本的文档

8.5.3 子元素列表

由于 SEASON 元素被声明为可以接受任何元素作为子元素，因而可以接受各种各样的元素。当遇到那些多多少少有些非结构化的文本，如杂志文章时，这种情况就很有用。这时段落、副栏、项目列表、序号列表、图形、照片以及子标题可出现在文档的任意位置。然而，有时可能想对数据的安排上多实行些规则和控制。例如，可能会要求每一个 LEAGUE 元素有一个 LEAGUE\_NAME 子元素，而每个 PLAYER 元素要有一个 GIVEN\_NAME 和 SURNAME 子元素，并且 GIVEN\_NAME 要放在 SURNAME 之前。

为了声明 LEAGUE 元素必须有一个名称，只要声明 LEAGUE\_NAME 元素，然后在 LEAGUE 声明后的括号内加入 LEAGUE\_NAME，如下面这样：

```
<!ELEMENT LEAGUE (LEAGUE_NAME)>
```

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

每个元素只能在其<!ELEMENT>内声明一次，即使它以其他<!ELEMENT>声明的子元素出现也一样。这里，我把 LEAGUE\_NAME 声明放在引用它的 LEAGUE 声明之后，这没有关系。XML 允许这一类提前引用。只要声明全部包含在 DTD 中，元素标记出现的顺序无关紧要。

可以向文档中添加这两项声明，然后在 SEASON 元素中包括 LEAGUE 和 LEAGUE\_NAME 元素。如清单 8-8 所示。图 8-8 是显示出来的文档。

清单 8-8：有两个 LEAGUE 子元素的 SEASON 元素

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

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

<!DOCTYPE SEASON [

<!ELEMENT YEAR (#PCDATA)>

<!ELEMENT LEAGUE (LEAGUE_NAME)>

<!ELEMENT LEAGUE_NAME (#PCDATA)>

<!ELEMENT SEASON ANY>

]>

<SEASON>

<YEAR>1998</YEAR>

<LEAGUE>

<LEAGUE_NAME>American League</LEAGUE_NAME>

</LEAGUE>

<LEAGUE>

<LEAGUE_NAME>National League</LEAGUE_NAME>

</LEAGUE>

</SEASON>
```



图 8-8 包含样式单、YEAR 元素和两个 LEAGUE 子元素的合法的文档

#### 8.5.4 序列

让我们限制一下 SEASON 元素。一个 SEASON 元素包含正好一个 YEAR 元素和其后的两个 LEAGUE 子元素。不把 SEASON 元素声明为可以包含 ANY 元素，我们在 SEASON 元素声明中包括这三个子元素，用括号括起来并用逗号分隔开，如下所示：

```
<!ELEMENT SEASON (YEAR, LEAGUE, LEAGUE)>
```

用逗号隔开的一系列子元素称为一个序列。利用这一声明，每个合法的 SEASON 元素必须包含正好一个 YEAR 元素，后面正好是两个 LEAGUE 元素，没有别的。整个文档类型定义现在看上去是下面的样子：

```
<!DOCTYPE SEASON [
```

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

```
<!ELEMENT LEAGUE (LEAGUE_NAME)>
```

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

```
<!ELEMENT SEASON (YEAR, LEAGUE, LEAGUE)>
```

```
]>
```

清单 8-8 所列的文档部分确实符合这项 DTD 的规定，因为它的 SEASON 元素包含一个 YEAR 子元素，后接两个 LEAGUE 子元素，再没有别的。但是，如果文档只包括一个 SEASON 元素，那么这个文档尽管结构完整，也将是非法的。同样，如果 LEAGUE 在 YEAR 之前而不是在其后，或者如果 LEAGUE 有 YEAR 子元素，或者文档在其他任何方面不符合 DTD，那么文档就是不合法的，合法性检查程序将拒绝这样的文档。

可直接将此种技术推广到 DIVISION 元素。每个 LEAGUE 有一个 LEAGUE\_NAME 和三个 DIVISION 子元素。例如：

```
<!ELEMENT LEAGUE (LEAGUE_NAME, DIVISION, DIVISION, DIVISION)>
```

#### 8.5.5 一个或多个子元素

每个 DIVISION 有一个 DIVISION\_NAME 和四到六个 TEAM 子元素。指定 DIVISION\_NAME 很容易，方法如下：

```
<!ELEMENT DIVISION (DIVISION_NAME)>
```

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

但是，TEAM 子元素就很棘手。指明 DIVISION 元素有四个 TEAM 子元素很容易，如下所示：

```
<!ELEMENT DIVISION (DIVISION_NAME, TEAM, TEAM, TEAM, TEAM)>
```

五个和六个也不难。但是您怎样说明有四到六个 TEAM 子元素呢？实际上，XML 没有提供实现的简单方法。但是可以在子元素清单的元素名后放一个加号（+）来说明有一个或多个子元素，例如：

```
<!ELEMENT DIVISION (DIVISION_NAME, TEAM+)>
```

这就是说一个 DIVISION 元素必须包含一个 DIVISION\_NAME 子元素，后接一个或多个 TEAM 子元素。



说明 DIVISION 元素有四到六个 TEAM 元素，而不是三到七个，这就难了。由于非常复杂，实际上很少有人使用。当读完本章时，看一看您是否已经想出怎样做了。

### 8.5.6 零或多个子元素

每个 TEAM 要包含一个 TEAM\_CITY，一个 TEAM\_NAME 和不确定数目的 PLAYER 元素。实际上，棒球队至少要九名球员。但是，本书的很多例子中由于篇幅的原因而没有列出球员。因而，我们要指明一个 TEAM 元素可包含零或多个 PLAYER 子元素。在子元素清单中在元素名上附加一个星号（\*）来实现这一目的。例如：

```
<!ELEMENT TEAM (TEAM_CITY, TEAM_NAME, PLAYER*)>
```

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

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

### 8.5.7 零或一个子元素

文档中出现的最后的元素是 PLAYER 子元素。它们全部是只包含文本的简单元素。下面是它们的声明：

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

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

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

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

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

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



<!ELEMENT RUNS (#PCDATA)>

<!ELEMENT HITS (#PCDATA)>

<!ELEMENT DOUBLES (#PCDATA)>

<!ELEMENT TRIPLES (#PCDATA)>

<!ELEMENT HOME\_RUNS (#PCDATA)>

<!ELEMENT RBI (#PCDATA)>

<!ELEMENT STEALS (#PCDATA)>

<!ELEMENT CAUGHT\_STEALING (#PCDATA)>

<!ELEMENT SACRIFICE\_HITS (#PCDATA)>

<!ELEMENT SACRIFICE\_FLIES (#PCDATA)>

<!ELEMENT ERRORS (#PCDATA)>

<!ELEMENT WALKS (#PCDATA)>

<!ELEMENT STRUCK\_OUT (#PCDATA)>

<!ELEMENT HIT\_BY\_PITCH (#PCDATA)>

<!ELEMENT COMPLETE\_GAMES (#PCDATA)>

<!ELEMENT SHUT\_OUTS (#PCDATA)>

<!ELEMENT ERA (#PCDATA)>

<!ELEMENT INNINGS (#PCDATA)>

<!ELEMENT EARNED\_RUNS (#PCDATA)>

<!ELEMENT HIT\_BATTER (#PCDATA)>

<!ELEMENT WILD\_PITCHES (#PCDATA)>

<!ELEMENT BALK (#PCDATA)>

<!ELEMENT WALKED\_BATTER (#PCDATA)>

<!ELEMENT WINS (#PCDATA)>

<!ELEMENT LOSSES (#PCDATA)>

<!ELEMENT SAVES (#PCDATA)>

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

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

现在我们可以编写 PLAYER 的元素声明了。所有球员有一个 GIVEN\_NAME、一个 SURNAME、一个 POSITION、一个 GAMES。我们可声明每个 PLAYER 元素有一个 AT\_BATS、RUNS、HITS 等等。但是，对于没有击球的投手列出零得分是否准确还不敢确定。因为这可能出现这样一种情况，就是在开始计算平均击球数等问题时会导致被零除的错误。如果某一特定的元素不适合于给定的球员，或没有这一元素，那么就应该从该球员信息中忽略这一元素的统计。对于给定的球员我们不允许多于一个这样的元素。因而，我们就需要给定类型的零个或一个元素。在子元素列表后面附加一个问号（?）可表明这一点，如下所示：

```
<!ELEMENT PLAYER (GIVEN_NAME, SURNAME, POSITION,
GAMES, GAMES_STARTED, AT_BATS?, RUNS?, HITS?, DOUBLES?,
TRIPLES?, HOME_RUNS?, RBI?, STEALS?, CAUGHT_STEALING?,
SACRIFICE_HITS?, SACRIFICE_FLIES?, ERRORS?, WALKS?,
STRUCK_OUT?, HIT_BY_PITCH ?, WINS?, LOSSES?, SAVES?,
COMPLETE_GAMES?, SHUT_OUTS?, ERA?, INNINGS_EARNED_RUNS?, HIT_BATTER?, WILD_PITCHES?,
BALK?, WALKED_BATTER?, STRUCK_OUT_BATTER?)
>
```

这就是说每个 PLAYER 元素有一个 GIVEN\_NAME、SURNAME、POSITION、GAMES 和 GAMES\_STARTED 子元素。而且，每名球员可能有或可能没有 AT\_BATS、RUNS、HITS、DOUBLES、TRIPLES、HOME\_RUNS、RBI、STEALS、CAUGHT\_STEALING、SACRIFICE\_HITS、SACRIFICE\_FLIES、ERRORS、WALKS、STRUCK\_OUT 和 HIT\_BY\_PITCH。

### 8.5.8 完整的文档和 DTD

我们现在有了棒球统计的完整的 DTD。这一 DTD 连同清单 8-4 中的文档部分一起，列在清单 8-9 中。

清单 8-9 只包括一个队和九名球员。在本书后附 CD-ROM 上的 examples/baseball/1998validstats.xml 目录下可



找到 1998 年主要联赛球队和队员的统计文档。

清单 8-9：一份合法的棒球统计文档和 DTD

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE SEASON [
<!ELEMENT YEAR (#PCDATA)>
<!ELEMENT LEAGUE (LEAGUE_NAME, DIVISION, DIVISION, DIVISION)>
<!ELEMENT LEAGUE_NAME (#PCDATA)>
```

<!ELEMENT DIVISION\_NAME (#PCDATA)>

<!ELEMENT DIVISION (DIVISION\_NAME, TEAM+)>

<!ELEMENT SEASON (YEAR, LEAGUE, LEAGUE)>

<!ELEMENT TEAM (TEAM\_CITY, TEAM\_NAME, PLAYER\*)>

<!ELEMENT TEAM\_CITY (#PCDATA)>

<!ELEMENT TEAM\_NAME (#PCDATA)>

<!ELEMENT PLAYER (GIVEN\_NAME, SURNAME, POSITION, GAMES, GAMES\_STARTED, WINS?, LOSSES?, SAVES?,

AT\_BATS?, RUNS?, HITS?, DOUBLES?, TRIPLES?, HOME\_RUNS?,

RBI?, STEALS?, CAUGHT\_STEALING?, SACRIFICE\_HITS?,

SACRIFICE\_FLIES?, ERRORS?, WALKS?, STRUCK\_OUT?,

HIT\_BY\_PITCH?, COMPLETE\_GAMES?, SHUT\_OUTS?, ERA?,

INNINGS?, EARNED\_RUNS?, HIT\_BATTER?, WILD\_PITCHES?,

BALK?, WALKED\_BATTER?, STRUCK\_OUT\_BATTER?)

>

<!ELEMENT SURNAME (#PCDATA)>

<!ELEMENT GIVEN\_NAME (#PCDATA)>

<!ELEMENT POSITION (#PCDATA)>

<!ELEMENT GAMES (#PCDATA)>

<!ELEMENT GAMES\_STARTED (#PCDATA)>

<!ELEMENT COMPLETE\_GAMES (#PCDATA)>

<!ELEMENT WINS (#PCDATA)>

<!ELEMENT LOSSES (#PCDATA)>

<!ELEMENT SAVES (#PCDATA)>

<!ELEMENT AT\_BATS (#PCDATA)>

<!ELEMENT RUNS (#PCDATA)>

<!ELEMENT HITS (#PCDATA)>

<!ELEMENT DOUBLES (#PCDATA)>

<!ELEMENT TRIPLES (#PCDATA)>

<!ELEMENT HOME\_RUNS (#PCDATA)>

<!ELEMENT RBI (#PCDATA)>

<!ELEMENT STEALS (#PCDATA)>

<!ELEMENT CAUGHT\_STEALING (#PCDATA)>

<!ELEMENT SACRIFICE\_HITS (#PCDATA)>

<!ELEMENT SACRIFICE\_FLIES (#PCDATA)>

<!ELEMENT ERRORS (#PCDATA)>

<!ELEMENT WALKS (#PCDATA)>

<!ELEMENT STRUCK\_OUT (#PCDATA)>

<!ELEMENT HIT\_BY\_PITCH (#PCDATA)>

<!ELEMENT SHUT\_OUTS (#PCDATA)>

<!ELEMENT ERA (#PCDATA)>

<!ELEMENT INNINGS (#PCDATA)>

<!ELEMENT HOME\_RUNS\_AGAINST (#PCDATA)>

<!ELEMENT RUNS\_AGAINST (#PCDATA)>

<!ELEMENT EARNED\_RUNS (#PCDATA)>

<!ELEMENT HIT\_BATTER (#PCDATA)>

<!ELEMENT WILD\_PITCHES (#PCDATA)>

<!ELEMENT BALK (#PCDATA)>

<!ELEMENT WALKED\_BATTER (#PCDATA)>

<!ELEMENT STRUCK\_OUT\_BATTER (#PCDATA)>

]>

<SEASON>

<YEAR>1998</YEAR>

<LEAGUE>

<LEAGUE\_NAME>National</LEAGUE\_NAME>

<DIVISION>

<DIVISION\_NAME>East</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Florida</TEAM\_CITY>

<TEAM\_NAME>Marlins</TEAM\_NAME>

<PLAYER>

<GIVEN\_NAME>Eric</GIVEN\_NAME>

<SURNAME>Ludwick</SURNAME>

<POSITION>Starting Pitcher</POSITION>

<GAMES>13</GAMES>

<GAMES\_STARTED>6</GAMES\_STARTED>

<WINS>1</WINS>

<LOSSES>4</LOSSES>

<SAVES>0</SAVES>

<COMPLETE\_GAMES>0</COMPLETE\_GAMES>

<SHUT\_OUTS>0</SHUT\_OUTS>

<ERA>7.44</ERA>

<INNINGS>32.2</INNINGS>

<EARNED\_RUNS>31</EARNED\_RUNS>

<HIT\_BATTER>27</HIT\_BATTER>

<WILD\_PITCHES>0</WILD\_PITCHES>

<BALK>2</BALK>

<WALKED\_BATTER>0</WALKED\_BATTER>

<STRUCK\_OUT\_BATTER>17</STRUCK\_OUT\_BATTER>

</PLAYER>

<PLAYER>

<GIVEN\_NAME>Brian</GIVEN\_NAME>

<SURNAME>Daubach</SURNAME>

<POSITION>First Base</POSITION>

<GAMES>10</GAMES>

<GAMES\_STARTED>3</GAMES\_STARTED>

<AT\_BATS>15</AT\_BATS>

<RUNS>0</RUNS>

<HITS>3</HITS>

<DOUBLES>1</DOUBLES>

<TRIPLES>0</TRIPLES>

<HOME\_RUNS>0</HOME\_RUNS>

<RBI>3</RBI>

<STEALS>0</STEALS>

<CAUGH T\_STEALING>0</CAUGHT\_STEALING>

<SACRIFICE\_HITS>0</SACRIFICE\_HITS>

<SACRIFICE\_FLIES>0</SACRIFICE\_FLIES>

<ERRORS>0</ERRORS>

<WALKS>1</WALKS>

<STRUCK\_OUT>5</STRUCK\_OUT>

<HIT\_BY\_PITCH>1</HIT\_BY\_PITCH>

</PLAYER>

</TEAM>

<TEAM>

<TEAM\_CITY>Montreal</TEAM\_CITY>

<TEAM\_NAME>Expos</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>New York</TEAM\_CITY>

<TEAM\_NAME>Mets</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Philadelphia</TEAM\_CITY>

<TEAM\_NAME>Phillies</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>Central</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Chicago</TEAM\_CITY>

<TEAM\_NAME>Cubs</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>West</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Arizona</TEAM\_CITY>

<TEAM\_NAME>Diamondbacks</TEAM\_NAME>

</TEAM>

</DIVISION>

</LEAGUE>

<LEAGUE>

<LEAGUE\_NAME>American</LEAGUE\_NAME>

<DIVISION>

<DIVISION\_NAME>East</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Baltimore</TEAM\_CITY>

<TEAM\_NAME>Orioles</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>Central</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Chicago</TEAM\_CITY>

<TEAM\_NAME>White Sox</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>West</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Anaheim</TEAM\_CITY>

<TEAM\_NAME>Angels</TEAM\_NAME>

</TEAM>

</DIVISION>

</LEAGUE>

</SEASON>



清单 8-9 不是符合这项 DTD 的唯一可能的文档，清单 8-10 也是一份合法的文档，因为它按规定的顺序包含了需要的所有元素，并且不包含未经声明的任何元素。这也许是您根据 DTD 创建的最短的合法文档。限定因素是这样的要求，每个 SEASON 包含两个 LEAGUE 子元素，每个 LEAGUE 子元素包含三个 DIVISION 子元素，每个 DIVISION 包含至少一个 TEAM 子元素。

清单 8-10: 另外一份符合棒球 DTD 的合法的 XML 文档

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

<!DOCTYPE SEASON [

<!ELEMENT YEAR (#PCDATA)>

<!ELEMENT LEAGUE (LEAGUE_NAME, DIVISION, DIVISION, DIVISION)>

<!ELEMENT LEAGUE_NAME (#PCDATA)>

<!ELEMENT DIVISION_NAME (#PCDATA)>

<!ELEMENT DIVISION (DIVISION_NAME, TEAM+)>

<!ELEMENT SEASON (YEAR, LEAGUE, LEAGUE)>

<!ELEMENT TEAM (TEAM_CITY, TEAM_NAME, PLAYER*)>

<!ELEMENT TEAM_CITY (#PCDATA)>

<!ELEMENT TEAM_NAME (#PCDATA)>

<!ELEMENT PLAYER (GIVEN_NAME, SURNAME, POSITION, GAMES,
GAMES_STARTED, COMPLETE_GAMES?, WINS?, LOSSES?, SAVES?,
AT_BATS?, RUNS?, HITS?, DOUBLES?, TRIPLES?, HOME_RUNS?,
RBI?, STEALS?, CAUGHT_STEALING?, SACRIFICE_HITS?,
SACRIFICE_FLIES?, ERRORS?, WALKS?, STRUCK_OUT?,
HIT_BY_PITCH?, COMPLETE_GAMES?, SHUT_OUTS?, ERA?, INNINGS?,
EARNED_RUNS?, HIT_BATTER?, WILD_PITCHES?, BALK?,
WALKED_BATTER?, STRUCK_OUT_BATTER?)
>

<!ELEMENT SURNAME (#PCDATA)>

<!ELEMENT GIVEN_NAME (#PCDATA)>

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

<!ELEMENT GAMES (#PCDATA)>

<!ELEMENT GAMES\_STARTED (#PCDATA)>

<!ELEMENT COMPLETE\_GAMES (#PCDATA)>

<!ELEMENT WINS (#PCDATA)>

<!ELEMENT LOSSES (#PCDATA)>

<!ELEMENT SAVES (#PCDATA)>

<!ELEMENT AT\_BATS (#PCDATA)>

<!ELEMENT RUNS (#PCDATA)>

<!ELEMENT HITS (#PCDATA)>

<!ELEMENT DOUBLES (#PCDATA)>

<!ELEMENT TRIPLES (#PCDATA)>

<!ELEMENT HOME\_RUNS (#PCDATA)>

<!ELEMENT RBI (#PCDATA)>

<!ELEMENT STEALS (#PCDATA)>

<!ELEMENT CAUGHT\_STEALING (#PCDATA)>

<!ELEMENT SACRIFICE\_HITS (#PCDATA)>

<!ELEMENT SACRIFICE\_FLIES (#PCDATA)>

<!ELEMENT ERRORS (#PCDATA)>

<!ELEMENT WALKS (#PCDATA)>

<!ELEMENT STRUCK\_OUT (#PCDATA)>

<!ELEMENT HIT\_BY\_PITCH (#PCDATA)>

<!ELEMENT SHUT\_OUTS (#PCDATA)>

<!ELEMENT ERA (#PCDATA)>

<!ELEMENT INNINGS (#PCDATA)>

<!ELEMENT HOME\_RUNS\_AGAINST (#PCDATA)>

<!ELEMENT RUNS\_AGAINST (#PCDATA)>

<!ELEMENT EARNED\_RUNS (#PCDATA)>

<!ELEMENT HIT\_BATTER (#PCDATA)>

<!ELEMENT WILD\_PITCHES (#PCDATA)>

<!ELEMENT BALK (#PCDATA)>

<!ELEMENT WALKED\_BATTER (#PCDATA)>

<!ELEMENT STRUCK\_OUT\_BATTER (#PCDATA)>

]>

<SEASON>

<YEAR>1998</YEAR>

<LEAGUE>

<LEAGUE\_NAME>National</LEAGUE\_NAME>

<DIVISION>

<DIVISION\_NAME>East</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Atlanta</TEAM\_CITY>

<TEAM\_NAME>Braves</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Florida</TEAM\_CITY>

<TEAM\_NAME>Marlins</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Montreal</TEAM\_CITY>

<TEAM\_NAME>Expos</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>New York</TEAM\_CITY>

<TEAM\_NAME>Mets</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Philadelphia</TEAM\_CITY>

<TEAM\_NAME>Phillies</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>Central</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Chicago</TEAM\_CITY>

<TEAM\_NAME>Cubs</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>West</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Arizona</TEAM\_CITY>

<TEAM\_NAME>Diamondbacks</TEAM\_NAME>

</TEAM>

</DIVISION>

</LEAGUE>

<LEAGUE>

<LEAGUE\_NAME>American</LEAGUE\_NAME>

<DIVISION>

```

<DIVISION_NAME>East </DIVISION_NAME>

<TEAM>

<TEAM_CITY>Baltimore</TEAM_CITY>

<TEAM_NAME>Orioles</TEAM_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION_NAME>Central</DIVISION_NAME>

<TEAM>

<TEAM_CITY>Chicago</TEAM_CITY>

<TEAM_NAME>White Sox</TEAM_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION_NAME>West</DIVISION_NAME>

<TEAM>

<TEAM_CITY>Anaheim</TEAM_CITY>

<TEAM_NAME>Angels</TEAM_NAME>

</TEAM>

</DIVISION>

</LEAGUE>

</SEASON>

```

### 8.5.9 选择

通常，一个父元素会有许多子元素。为了指明各子元素必须按顺序出现，可将这些子元素用逗号隔开。每个这样的子元素还可以以问号、加号或星号作后缀，以便调节它在那一位置按顺序出现的次数。

到目前为止，已经假定子元素以一定的顺序出现或不出现。还可以使 DTD 更加灵活，如允许文档作者在给定的地方选择不同的元素。例如，在一项描述顾客购物的 DTD 中，结帐方式信息中的每项 PAYMENT 元素都有 CREDIT\_CARD 子元素或 CASH 子元素以便提供付款方式的信息。然而，单独的 PAYMENT 元素不能同时使用两者。

在父元素声明中，可以使用竖线（|）而不是逗号来分开子元素，以便指明文档作者需要输入一个或另一个子元素。例如，下面的语句就说明 PAYMENT 元素必须有 CASH 或 CREDIT\_CARD 中的一个子元素。

```
<!ELEMENT PAYMENT (CASH | CREDIT_CARD)>
```

这种内容规格称为选择。当只使用它们当中的一个时就可使用竖线分开任意数目的子元素。例如，下面语句说明 PAYMENT 元素必须有 CASH、CREDIT\_CARD 或 CHECK 中的一个子元素。

```
<!ELEMENT PAYMENT (CASH | CREDIT_CARD | CHECK)>
```

当用括号对元素分组时竖线还会更有用。可以把元素组合在括号内分组，然后在括号后加星号、问号和加号后缀来指明一定的元素组合会出现零次或多次、零次或一次或者一次或多次。

#### 8.5.10 带括号的子元素

在父元素声明中，必须了解有关子元素安排的最后一件事是如何用括号分组元素。每一对括号把数个元素合为一个独立元素。括号内的元素可以作为独立元素嵌套在其他括号内。而且，还可以加上加号、逗号或问号后缀。您还可以将这些括号组合成更大的括号组合来构成复杂的结构。这是一项功能强大的技术。

例如，考虑一份由两个互相可交换的元素组成的清单。这基本上是 HTML 中定义清单的方法。每项<dt>标记要与一项<dd>标记相匹配。如果用 XML 来复制这一结构，dl 元素的声明看起来是这样的：

```
<!ELEMENT dl (dt , dd)*>
```

括号表明要重复的是相互匹配的<dt><dd>元素对。

元素经常以或多或少的随机顺序出现。新闻杂志文章通常有一个标题，绝大多数后接文章段落，带有图形、照片、副栏、副标题、通篇夹杂的引文，也许在末尾还有作者行。可以在父元素声明中在括号内用竖线分组列出所有子元素来指明这些安排。然后您在括号外加星号来指明允许括号内元素出现零或多次。例如：

```
<!ELEMENT ARTICLE (TITLE, (P | PHOTO | GRAPH | SIDEBAR  
| PULLQUOTE | SUBHEAD)*, BYLINE?)>
```

再举一例，假设要说明一个 DOCUMENT 元素，它没有很多子元素，但必须有一个 TITLE 后接任意数量的混合文本段落和图像，以及一个任选的 SIGNATURE 块。该元素声明书写如下：

```
<!ELEMENT DOCUMENT (TITLE, (PARAGRAPH | IMAGE)*, SIGNATURE?)>
```

这不是描述这一结构的唯一方法。实际上这甚至不是最好的方法。另一种方法是声明一个包含 PARAGRAPH 和 IMAGE 元素的 BODY 元素并把它夹在 TITLE 和 SIGNATURE 元素之间，例如：

```
<!ELEMENT DOCUMENT (TITLE, BODY, SIGNATURE?)>
```

```
<!ELEMENT BODY ((PARAGRAPH | IMAGE)*)>
```

这两种途径的区别在于第二种途径在文档中使用了 BODY 元素。这一元素对读取文档的应用程序提供了有用的组织层次。问题是文档的读者（可能是另一种计算机程序）是否要把 BODY 作为单一的项目，并同 TITLE 和 SIGNATURE 分开，并可从元素总和区别出来。

再举一个国际地址的例子。美国以外国家的地址并不遵循美国的约定。尤其是邮政编码有时在国家名之前，有时则在其后，如下两例：

Doberman-YPPAN

Box 2021

St. Nicholas QUEBEC

CAN GOS-3L0

或者

Editions Sybex

10/12 Villa Coeur-de-Vey

75685 Paris Cedex 14

France

虽然地址项不是按照顺序，邮件也可能邮到目的地，但最好还是让地址更加方便灵活些。允许灵活性的地址元素声明可以是这样：

```
<!ELEMENT ADDRESS (STREET+, (CITY | STATE | POSTAL_CODE  
| COUNTRY)*)>
```

这表明 ADDRESS 元素必须有一个或多个 STREET 子元素后接任意数目的 CITY、STATE、POSTAL\_CODE 或 COUNTRY 元素。如果要每个元素不多于一个，那这就不够理想了。遗憾的是，这超出了 DTD 的能力。您要使元素的顺序更加灵活方便，就要放弃一些控制每一元素最大数的能力。

另一方面，可能有一份由任意顺序排列的不同元素组成的清单，如一份录音清单就可能包含 CD，唱片集和音带。区别各类不同元素的元素声明可能如下：

```
<!ELEMENT MUSIC_LIST (CD | ALBUM | TAPE)*>
```

在棒球 DTD 中，可以使用括号来为投手和击球手做不同的统计数据集。每名队员能用一套或另一套数据，但不能用两者。元素声明如下：

```
<!ELEMENT PLAYER (GIVEN_NAME, SURNAME, POSITION, GAMES,  
GAMES_STARTED, ((COMPLETE_GAMES?, WINS?, LOSSES?, SAVES?,  
SHUT_OUTS?, ERA?, INNINGS?, EARNED_RUNS?, HIT_BATTER?,  
WILD_PITCHES?, BALK?, WALKED_BATTER?, STRUCK_OUT_BATTER? )  
| (AT_BATS?, RUNS?, HITS?, DOUBLES?, TRIPLES?, HOME_RUNS?,  
RBI?, STEALS?, CAUGHT_STEALING?, SACRIFICE_HITS?,
```

SACRIFICE\_FLIES?, ERRORS?, WALKS?, STRUCK\_OUT?,

HIT\_BY\_PITCH ? )))>

在元素声明中还有一些不好处理的事情。例如，没有好的方法来说明一份文档要以 TITLE 元素开始而以 SIGNATURE 元素结束，两者之间可包含其他元素。这是因为 ANY 不能与其他子元素合用。

还有，通常对元素出现的位置掌握得越不准确，就越不能控制它们的数目。例如，不能说文档应该有一个可能出现在文档任何地方的 TITLE 元素。

但是，用括号来建立元素块，按顺序的元素用逗号分隔，平行出现的用竖线分隔，能让我们建立带有详细的元素出现的位置规则的复杂结构。但是不要无止境地这样做。简单的解决方法会更好。DTD 越复杂，就越难编写出满足要求的合法的文档，更不要说维护 DTD 自身的复杂性了。

### 8.5.11 混合内容

读者可能已经注意到了，在以前的多数例子中，元素或者包含子元素，或者包含可析的字符数据，但不能同时包含两者。唯一的例外是以前例子中的一些基本元素。在这些例子中，全部标记的列表还没有完成。由于基本元素可以包含 ANY 数据，因而就既可以包含子元素又可以包含原始文本。

可以声明同时包含子元素和可析字符数据的标记。这就叫做混合内容。这样就可以给每个 TEAM 后面加上任意的文本块。例如：

```
<!ELEMENT TEAM (#PCDATA | TEAM_CITY | TEAM_NAME | PLAYER)*>
```

带有可析的字符数据的混合子元素会严重地限制文档的结构。特别是，只能指定可出现的子元素的名称。不能限定它们出现的顺序，每个元素的出现次数，或者它们是否出现。借助于 DTD，利用下面的 DTD 中的一部分可实现这一要求：

```
<!ELEMENT PARENT (#PCDATA | CHILD1 | CHILD2 | CHILD3 )* >
```

除了改变子元素数目以外的其他事情都是不合法的。不能在包括#PCDATA 的元素声明中使用逗号、问号或加号。用竖线分隔的元素和#PCDATA 的列表是合法的。其他用法是不合法的。例如，下面的例子就不合法：

```
<!ELEMENT TEAM (TEAM_CITY, TEAM_NAME, PLAYER*, #PCDATA)>
```

使用混合内容的最基本的理由是，当将老式的文本数据转换成 XML 的过程中，随着新标记的增加逐步测试 DTD 的合法性，而不要在完成全部转换后再试图去发现错误。这是一个很好的技巧，我建议大家这样做，毕竟从刚完成的代码中立即找出错误比几小时后要容易一些。但是，这仅仅是开发时的一个技巧。最终的用户是不应该看到这些的。当 DTD 完成后不能把子元素同可析的字符数据混合起来。一般总可以建立一个包括可析的字符数据的新标记。

例如，可以声明只包含#PCDATA 数据的 BLURB 元素并把它增加为 TEAM 的最后一个子元素，这样就在每个 TEAM 元素的末尾包括一个文本块。下面是该声明：

```
<!ELEMENT TEAM (TEAM_CITY, TEAM_NAME, PLAYER*, BLURB)>
```

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

这对文档的文本改变不大。所有的变化只是向每个 TEAM 元素增加了一个或多个带有开始标记和结束标记的可选元素。但是这就使文档更加健全。而且，从 XML 程序接收到文档树的 XML 应用程序就能在更短的时间内处理数据，因为文档具有非混合内容所允许的更为结构化的格式。



### 8.5.12 空元素

前面讨论过，定义一个没有内容的元素有时是有用的。HTML 中的例子包括图像、水平线和中断<IMG>、<R>和<BR>。在 XML 中，这类空元素是通过以/>结束的空标记来标识的，如<IMG/>、<HR/>和<BR/>。

合法的文档必须声明使用的空元素和非空元素。因为按定义，空元素没有子元素，声明很容易。可像通常一样使用包含空元素名的<!ELEMENT>来声明，但用关键词 EMPTY （像所有 XML 标记一样区分大小写）代替了子元素的列表。例如：

```
<!ELEMENT BR EMPTY>
```

```
<!ELEMENT IMG EMPTY>
```

```
<!ELEMENT HR EMPTY>
```

清单 8-11 是同时使用了空元素和非空元素的合法文档。

清单 8-11：使用了空标记的合法文档

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

```
<!DOCTYPE DOCUMENT [
```

```
<!ELEMENT DOCUMENT (TITLE, SIGNATURE)>
```

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

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

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

```
<!ELEMENT BR EMPTY>
```

```
<!ELEMENT HR EMPTY>
```

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

```
<!ELEMENT SIGNATURE (HR, COPYRIGHT, BR, EMAIL,
```

```
BR, LAST_MODIFIED)>
```

```
]>
```

```
<DOCUMENT>
```

```
<TITLE>Empty Tags</TITLE>
```

```
<SIGNATURE>
```

```
<HR/>
```

```
<COPYRIGHT>1998 Elliotte Rusty Harold</COPYRIGHT><BR/>
```

<EMAIL>elharo@metalab.unc.edu</EMAIL><BR/>

<LAST\_MODIFIED>Thursday, April 22, 1999</LAST\_MODIFIED>

</SIGNATURE>

</DOCUMENT>

## 8.6 DTD 中的注释

像一份 XML 文档的其他部分一样，DTD 也可以包含注释。这些注释不能在声明中出现，但可以在声明外出现。注释通常用来组织不同部分的 DTD，为一些元素的许可内容提供说明，并对元素作进一步的解释。例如，YEAR 元素的声明可以有这样的注释：

```
<!--A four digit year like 1998, 1999, or 2000 ?-->
```

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

像所有注释一样，这只是为了便于人们阅读源代码，XML 处理程序会忽略注释部分。

注释的一个可能用法是定义标记中用到的缩略语。例如，在本章及前些章中，我极力避免使用棒球术语的缩略语，因为对一些人来说难以弄清楚。一种可能的途径是使用缩略语但在 DTD 中用注释加以定义。清单 8-12 同以前的棒球例子相似，但使用了 DTD 注释和缩略标记。

清单 8-12：使用缩略标记和 DTD 注释的合法 XML 文档

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

```
<!DOCTYPE SEASON [
```

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

```
<!ELEMENT LEAGUE (LEAGUE_NAME, DIVISION, DIVISION, DIVISION)>
```

```
<!--American or National ?
```

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

```
<!--East , West , or Central ?
```

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

```
<!ELEMENT DIVISION (DIVISION_NAME, TEAM+)>
```

```
<!ELEMENT SEASON (YEAR, LEAGUE, LEAGUE)>
```

```
<!ELEMENT TEAM (TEAM_CITY, TEAM_NAME, PLAYER*)>
```

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

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

```
<!ELEMENT PLAYER (GIVEN_NAME, SURNAME, P, G,
```

```
GS, AB?, R?, H?, D?, T?, HR?, RBI?, SB?, CS?,
```

```
SH?, SF?, E?, BB?, S?, HBP?, CG?, SO?, ERA?, IP?,
```

```
HRA?, RA?, ER?, HB?, WP?, B?, WB?, K?)
```

>

<!--=====-->

<!--Player Info-->

<!--Player' s last name-->

<!ELEMENT SURNAME (#PCDATA)>

<!--Player' s first name-->

<!ELEMENT GIVEN\_NAME (#PCDATA)>

<!--Position-->

<!ELEMENT P (#PCDATA)>

<!--Games Played-->

<!ELEMENT G (#PCDATA)>

<!--Games Started-->

<!ELEMENT GS (#PCDATA)>

<!--=====-->

<!--Batting Statistics-->

<!--At Bats-->

<!ELEMENT AB (#PCDATA)>

<!--Runs-->

<!ELEMENT R (#PCDATA)>

<!--Hits-->

<!ELEMENT H (#PCDATA)>

<!--Doubles-->

<!ELEMENT D (#PCDATA)>

<!--Triples-->

<!ELEMENT T (#PCDATA)>

<!--Home Runs-->

<!ELEMENT HR (#PCDATA)>

<!--Runs Batted In-->

<!ELEMENT RBI (#PCDATA)>

<!--Stolen Bases-->

<!ELEMENT SB (#PCDATA)>

<!--Caught Stealing-->

<!ELEMENT CS (#PCDATA)>

<!--Sacrifice Hits-->

<!ELEMENT SH (#PCDATA)>

<!--Sacrifice Flies-->

<!ELEMENT SF (#PCDATA)>

<!--Errors-->

<!ELEMENT E (#PCDATA)>

<!--Walks (Base on Balls)-->

<!ELEMENT BB (#PCDATA)>

<!--Struck Out-->

<!ELEMENT S (#PCDATA)>

<!--Hit By Pitch-->

<!ELEMENT HBP (#PCDATA)>

<!--=====-->

<!--Pitching Staistics-->

<!--Complete Games-->

<!ELEMENT CG (#PCDATA)>

<!--Shut Outs-->

<!ELEMENT SO (#PCDATA)>

<!--ERA-->

<!ELEMENT ERA (#PCDATA)>

<!--Innings Pitched-->

<!ELEMENT IP (#PCDATA)>

<!--Home Runs hit Against-->

<!ELEMENT HRA (#PCDATA)>

<!--Runs hit Against-->

<!ELEMENT RA (#PCDATA)>

<!--Earned Runs-->

<!ELEMENT ER (#PCDATA)>

<!--Hit Batter-->

<!ELEMENT HB (#PCDATA)>

<!--Wild Pitches-->

<!ELEMENT WP (#PCDATA)>

<!--Balk-->

<!ELEMENT B (#PCDATA)>

<!--Walked Batter-->

<!ELEMENT WB (#PCDATA)>

<!--Struck Out Batter-->

<!ELEMENT K (#PCDATA)>

<!--=====-->

<!--Fielding Statistics-->

<!--Not yet supported-->

]>

<SEASON>

<YEAR>1998</YEAR>

<LEAGUE>

<LEAGUE\_NAME>National</LEAGUE\_NAME>

<DIVISION>

<DIVISION\_NAME>East </DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Atlanta</TEAM\_CITY>

<TEAM\_NAME>Braves</TEAM\_NAME>

<PLAYER>

<GIVEN\_NAME>Ozzie</GIVEN\_NAME>

<SURNAME>Guillen</SURNAME>

<P>Short stop</P>

<G>83</G>

<GS>59</GS>

<AB>264</AB>

<R>35</R>

<H>73</H >

<D>15</D>

<T>1</T>

<HR>1</HR>

<RBI>22</RBI>

<SB>1</SB>

<CS>4</CS>

<SH>4</SH>

<SF>2</SF>

<E>6</E>

<BB>24</BB>

<S>25</S>

<HBP>1</HBP>

</PLAYER>

</TEAM>

<TEAM>

<TEAM\_CITY>Florida</TEAM\_CITY>

<TEAM\_NAME>Marlins</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Montreal</TEAM\_CITY>

<TEAM\_NAME>Expos</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>New York</TEAM\_CITY>

<TEAM\_NAME>Mets</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Philadelphia</TEAM\_CITY>

<TEAM\_NAME>Phillies</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>Central</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Chicago</TEAM\_CITY>

<TEAM\_NAME>Cubs</TEAM\_NAME>

</TEAM>



</DIVISION>

<DIVISION>

<DIVISION\_NAME>West</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Arizona</TEAM\_CITY>

<TEAM\_NAME>Diamondbacks</TEAM\_NAME>

</TEAM>

</DIVISION>

</LEAGUE>

<LEAGUE>

<LEAGUE\_NAME>American</LEAGUE\_NAME>

<DIVISION>

<DIVISION\_NAME>East</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Baltimore</TEAM\_CITY>

<TEAM\_NAME>Orioles</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>Central</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Chicago</TEAM\_CITY>

<TEAM\_NAME>White Sox</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>West</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Anaheim</TEAM\_CITY>

<TEAM\_NAME>Angels</TEAM\_NAME>

</TEAM>

</DIVISION>

</LEAGUE>

</SEASON>

当整个联赛全包含在内时，产生的文档从带长标记的 699K 缩短到带短标记的 391K，节约了 44%。但是信息的内容是相同的。两个文档压缩后很接近，短标记文档 58K，长标记文档 66K。

在注释内可以包括和应该包括的信息量没有限制。包括得多使 DTD 更长（这样就使检测更难，下载更慢）。然而，在下面的几章中，您将学会如何在多个 XML 文档间共享同一 DTD 以及将 DTD 拆成更好管理的多个部分。这样，使用注释的缺点就是暂时的了。我建议 DTD 中自由地使用注释，尤其是对于打算公用的 DTD。

## 8.7 在文档间共享通用的 DTD

前面的合法的例子都在文档的序言部分包含了 DTD。但是 XML 真正的功能来自于不同的人们编写的可为许多文档共享通用的 DTD。如果 DTD 不是直接包含在文档内，而是从外部联结而来，则 DTD 的改变会自动传播给使用它的所有文档。另一方面，当 DTD 改变时并不能确保其向后兼容性。不兼容的改变会破坏文档。

当使用外部 DTD 时，文档类型声明要加以改变。DTD 不再是包括在方括号中，而是在 SYSTEM 关键词后接一个能找到 DTD 的绝对或相对的 URL。例如：

```
<!DOCTYPE root_element_name SYSTEM "DTD_URL">
```

这里 root\_element\_name 像以前一样是基本元素的名称，SYSTEM 是一个 XML 关键词，DTD\_URL 是能找到 DTD 的绝对或相对的 URL。例如：

```
<!DOCTYPE SEASON SYSTEM "baseball.dtd">
```

为说明这一过程让我们来转换一个熟悉的例子。清单 8-12 包括了棒球统计的内部 DTD。我们要把这份清单转换为外部 DTD。首先，去掉 DTD 并把它放入自己的文档。DTD 是起始于<!DOCTYPE SEASON [终止于]>之间的所有内容。但不包括<!DOCTYPE SEASON [和]>。可以将其保存在名为 baseball.dtd 的文档内，如清单 8-13 所示。文档名并不重要，通常用的扩展名为 .dtd。

清单 8-13：棒球的 DTD 文档

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

```
<!ELEMENT LEAGUE (LEAGUE_NAME, DIVISION, DIVISION, DIVISION)>
```

```
<!--American or National-->
```

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

```
<!--East, West, or Central-->
```

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

```
<!ELEMENT DIVISION (DIVISION_NAME, TEAM+)>
```

```
<!ELEMENT SEASON (YEAR, LEAGUE, LEAGUE)>
```

```
<!ELEMENT TEAM (TEAM_CITY, TEAM_NAME, PLAYER*)>
```

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

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

```
<!ELEMENT PLAYER (GIVEN_NAME, SURNAME, P, G,
```

```
GS, AB?, R?, H?, D?, T?, HR?, RBI?, SB?, CS?,
```

```
SH?, SF?, E?, BB?, S?, HBP?, CG?, SO?, ERA?, IP?,
```

```
HRA?, RA?, ER?, HB?, WP?, B?, WB?, K?)
```

>

<!--=====-->

<!--Player Info-->

<!--Player' s last name-->

<!ELEMENT SURNAME (#PCDATA)>

<!--Player' s first name-->

<!ELEMENT GIVEN\_NAME (#PCDATA)>

<!--Position-->

<!ELEMENT P (#PCDATA)>

<!--Games Played-->

<!ELEMENT G (#PCDATA)>

<!--Games Started-->

<!ELEMENT GS (#PCDATA)>

<!--=====-->

<!--Batting Statistics-->

<!--At Bats-->

<!ELEMENT AB (#PCDATA)>

<!--uns-->

<!ELEMENT R (#PCDATA)>

<!--Hits--> ?

<!ELEMENT H (#PCDATA)>

<!--Doubles-->

<!ELEMENT D (#PCDATA)>

<!--Triples-->

<!ELEMENT T (#PCDATA)>

<!--Home Runs-->

<!ELEMENT HR (#PCDATA)>

<!--Runs Batted In-->

<!ELEMENT RBI (#PCDATA)>

<!--Stolen Bases-->

<!ELEMENT SB (#PCDATA)>

<!--Caught Stealing-->

<!ELEMENT CS (#PCDATA)>

<!--Sacrifice Hits-->

<!ELEMENT SH (#PCDATA)>

<!--Sacrifice Flies-->

<!ELEMENT SF (#PCDATA)>

<!--Errors-->

<!ELEMENT E (#PCDATA)>

<!--Walks (Base on Balls)-->

<!ELEMENT BB (#PCDATA)>

<!--Struck Out-->

<!ELEMENT S (#PCDATA)>

<!--Hit By Pitch-->

<!ELEMENT HBP (#PCDATA)>

<!--=====-->

<!--Pitching Staistics-->

<!--Complete Games-->

<!ELEMENT CG (#PCDATA)>

<!--Shut Outs-->

<!ELEMENT SO (#PCDATA)>

<!--ERA-->

<!ELEMENT ERA (#PCDATA)>

<!--Innings Pitched-->

<!ELEMENT IP (#PCDATA)>

<!--Home Runs hit Against-->

<!ELEMENT HRA (#PCDATA)>

<!--Runs hit Against-->

<!ELEMENT RA (#PCDATA)>

<!--Earned Runs-->

<!ELEMENT ER (#PCDATA)>

<!--Hit Batter-->

<!ELEMENT HB (#PCDATA)>

<!--Wild Pitches-->

<!ELEMENT WP (#PCDATA)>

<!--Balk-->

<!ELEMENT B (#PCDATA)>

<!--Walked Batter-->

<!ELEMENT WB (#PCDATA)>

<!--Struck Out Batter-->

<!ELEMENT K (#PCDATA)>

<!--=====-->

<!--Fielding Statistics-->

<!--Not yet supported-->

接下来，需要改动文档本身。因为要依赖于另一文档中的 DTD，XML 声明不再是独立的文档。所以 standalone 属性要改为 no，如下所示：

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

然后还要改变<!DOCTYPE>标记，借助于包括 SYSTEM 关键字和 URL（通常是相对的）使它指向 DTD。

<!DOCTYPE SEASON SYSTEM "baseball.dtd" >

文档的其余部分与以前相同。但是，现在序言部分只包含 XML 声明和文档类型声明而不包括 DTD。清单 8-14 显示了这些代码。

清单 8-14：带有外部 DTD 的棒球统计

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

<!DOCTYPE SEASON SYSTEM "baseball.dtd" >

<SEASON>

<YEAR>1998</YEAR>

<LEAGUE>

<LEAGUE_NAME>National</LEAGUE_NAME>

<DIVISION>

<DIVISION_NAME>East</DIVISION_NAME>

<TEAM>

<TEAM_CITY>Atlanta</TEAM_CITY>

<TEAM_NAME>Braves</TEAM_NAME>

<PLAYER>

<GIVEN_NAME>Ozzie</GIVEN_NAME>

<SURNAME>Guillen</SURNAME>

<P>Shortstop</P>

<G>83</G>

<GS>59</GS>

<AB>264</AB>

<R>35</R>

<H>73</H>

<D>15</D>

<T>1</T>

<HR>1</HR>

<RBI>22</RBI>
```

<SB>1</SB>

<CS>4</CS>

<S >4</S >

<SF>2</SF>

<E>6</E>

<BB>24</BB>

<S>25</S>

<HBP>1</HBP>

</PLAYER>

</TEAM>

<TEAM>

<TEAM\_CITY>Florida</TEAM\_CITY>

<TEAM\_NAME>Marlins</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Montreal</TEAM\_CITY>

<TEAM\_NAME>Expos</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>New York</TEAM\_CITY>

<TEAM\_NAME>Mets</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Philadelphia</TEAM\_CITY>

<TEAM\_NAME>Phillies</TEAM\_NAME>

</TEAM>



</DIVISION>

<DIVISION>

<DIVISION\_NAME>Central</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Chicago</TEAM\_CITY>

<TEAM\_NAME>Cubs</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>West</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Arizona</TEAM\_CITY>

<TEAM\_NAME>Diamondbacks</TEAM\_NAME>

</TEAM>

</DIVISION>

</LEAGUE>

<LEAGUE>

<LEAGUE\_NAME>American</LEAGUE\_NAME>

<DIVISION>

<DIVISION\_NAME>East</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Baltimore</TEAM\_CITY>

<TEAM\_NAME>Orioles</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

```
<DIVISION_NAME>Central</DIVISION_NAME>
```

```
<TEAM>
```

```
<TEAM_CITY>Chicago</TEAM_CITY>
```

```
<TEAM_NAME>White Sox</TEAM_NAME>
```

```
</TEAM>
```

```
</DIVISION>
```

```
<DIVISION>
```

```
<DIVISION_NAME>West</DIVISION_NAME>
```

```
<TEAM>
```

```
<TEAM_CITY>Anaheim</TEAM_CITY>
```

```
<TEAM_NAME>Angels</TEAM_NAME>
```

```
</TEAM>
```

```
</DIVISION>
```

```
</LEAGUE>
```

```
</SEASON>
```

一定要确保清单 8-14 和 baseball.dtd 在同一目录下，然后像通常一样把清单 8-14 装入 Web 浏览器。如果一切正常，就会看到同装入清单 8-12 一样的输出。现在可以使用这个 DTD 来编写其他文档，如其他年度的统计数据。

如果添加了样式单，那么就在三个不同的文档中保存了文档的三个重要部分。数据在文档文件中，数据应用的结构和语义在 DTD 文件中，而格式在样式单中。这种结构使我们能相对独立地检查和改变其中任一部分或全部。

DTD 与文档之间比文档与样式单之间联系更紧密。改变 DTD 一般要重新检查文档的合法性，并需要编辑文档使它与 DTD 相符。这样的顺序必要性取决于编辑方法；增加元素没什么问题，但移走元素就可能有问题。

### 8.7.1 远程 URL 上的 DTD

如果一个 DTD 适用于多份文档，就不能总把它放在应用它的每份文档的同一目录下。可以使用 URL 来准确指明 DTD 的地址。例如，让我们假设棒球 DTD 在 <http://metalab.unc.edu/xml/dtds/baseball.dtd>，可在序言中使用下面的<!DOCTYPE> 标记将其链接到文档上：

```
<!DOCTYPE SEASON SYSTEM
```

```
"http://metalab.unc.edu/xml/dtds/baseball.dtd">
```

本例中使用了完整的 URL，从任何地方都是合法的。有时也希望从相对于 Web 服务器文档根目录或当前目录找出 DTD 来。通常，任何相对于文档位置所形成合法的 URL 的引用都可以接受。例如，下面这些都是合法的文档类型声明：

```
<!DOCTYPE SEASON SYSTEM"/xml/dtds/baseball.dtd">
```

```
<!DOCTYPE SEASON SYSTEM"/dtds/baseball.dtd">
```

```
<!DOCTYPE SEASON SYSTEM "../baseball.dtd">
```



一个文档不能有多于一个的文档类型声明，即不能有多于一个的<!DOCTYPE >标记。要使用不止在一个 DTD 中声明的元素，就需要使用外部参数实体引用。这些内容将在下一章中讨论。

## 8.7.2 公共的 DTD

关键词 SYSTEM 是为单个作者或小组所用的私有的 DTD 使用的。但作为 XML 承诺的一部分，可令覆盖整个产业的广泛组织（如 ISO 或 IEEE）能够将公共 DTD 加以标准化，以便用于各自的专门领域。这样的标准化可以让人们不用为同一项目重复作标记，并且使用户共享公用文档更容易。

为创建组织之外的编写者设计的 DTD 使用 PUBLIC 关键词而不使用 SYSTEM 关键词。并且 DTD 有一个文件名。句法如下：

```
<!DOCTYPE root_element_name PUBLIC "DTD_name" "DTD_URL">
```

root\_element\_name 仍然是基本元素名称。PUBLIC 是 XML 关键词，说明这一 DTD 是公共使用并具有名称。DTD\_name 是与此 DTD 联系名称。有些 XML 处理程序会使用名称从中心库中检索 DTD。最后，如果 DTD 不能根据名称从熟知的库中检索到，则 DTD\_URL 是一个能找到该 DTD 的相对或绝对 URL。

DTD 名称与 XML 名称略有不同。它们只能包含 ASCII 字母字符、空格、软回车符、换行符和下面的标点符号：- ' ( ) + , / : = ? ; ! \* # @ \$ % \_ 。而且，公共 DTD 要遵守一些约定。

如果一项 DTD 是 ISO 标准，它的名称要以字符串“ISO”开始。如果是非 ISO 标准组织批准的 DTD，它的名称以加号 (+) 开始。如果不是标准组织批准的 DTD，它的名称以连字符 (-) 开始。这些开始字符串后接双斜线 (//) 和 DTD 所有者的名字，其后接另一个双斜线和 DTD 描述的文档类型，然后又是一个双斜线后接 ISO639 语言标识符，如 EN 表示英语。在 <http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt> 处列有完整的 ISO639 标识符。例如，棒球 DTD 可以如下命名：

```
-//Elliott Rusty Harold//DTD baseball statistics//EN
```

本例说明这个 DTD 不是由任何标准组织批准的 (-)，为 Elliott Rusty Harold 所有，描述棒球统计，用英语编写。通过 DTD 名称指向这一 DTD 的完整的文档类型声明如下：

```
<!DOCTYPE SEASON PUBLIC
```

```
"//Elliott Rusty Harold//DTD baseball statistics//EN"
```

```
"http://metalab.unc.edu/xml/dtds/baseball.dtd">
```

读者也许注意到了许多 HTML 编辑器如 BBEdit 会在其创建的每个 HTML 文档开端放入下列字符串：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML//EN">
```

现在可能 了解这些字符串是什么意思了！它表明文档符合一项非标准 (-) 的 HTML 的 DTD，由 W3C 用英语制作。



从技术上说, W3C 不是一个标准组织, 因为它的成员限于交纳会费的公司而不是官方批准的实体。它只出版建议而不是标准。实际上这种区别没有关系。

### 8.7.3 内部和外部 DTD 子集

尽管大多数文档由易于定义的部分组成, 但不是所有的文档都使用共同的模板。许多文档为自己使用而增加特定元素时, 可能需要像 HTML 4.0 DTD 这样的标准 DTD。其他文档可能只使用标准元素, 但需要对它们重新排序。例如, 一个 HTML 主页可能有一个 BODY 元素, 它必须包含一个 H1 标题标记后接一份 DL 定义列表, 而另一个 HTML 主页可能有一个 BODY 元素, 它包含许多不同的顺序不定的标题标记、段落和图像。如果特定的文档与同一站点上其他页面具有不同的结构, 在文档本身内定义结构比在单独的 DTD 中定义更有用。这种方法也使文档更易于编辑。

为达此目的, 文档可使用内部和外部 DTD。内部声明放在<!DOCTYPE>标记尾部的方括号中。例如, 假如需要一个包括棒球统计并有页眉和页脚的主页。这样的文档可如清单 8-15 所示。棒球信息从文档 baseball.dtd 中得到, 构成外部 DTD 子集。基本元素 DOCUMENT 以及元素 TITLE 和 SIGNATURE 的定义来自包含于文档中的内部 DTD 子集。这有点不寻常。一般的, 更为通用的部分可能应该是外部 DTD 的一部分, 而内部内容则更与特定专题有关。

清单 8-15: DTD 具有内部和外部 DTD 子集的棒球文档

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

<!DOCTYPE SEASON SYSTEM "baseball.dtd"> [

<!ELEMENT DOCUMENT (TITLE, SEASON, SIGNATURE)>

<!ELEMENT TITLE (#PCDATA)>

<!ELEMENT COPYRIGHT (#PCDATA)>

<!ELEMENT EMAIL (#PCDATA)>

<!ELEMENT LAST_MODIFIED (#PCDATA)>

<!ELEMENT SIGNATURE (COPYRIGHT, EMAIL, LAST_MODIFIED)>

]>

<DOCUMENT>

<TITLE>1998 Major League Baseball Statistics</TITLE>

<SEASON>

<YEAR>1998</YEAR>

<LEAGUE>

<LEAGUE_NAME>National</LEAGUE_NAME>

<DIVISION>
```

<DIVISION\_NAME>East</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Atlanta</TEAM\_CITY>

<TEAM\_NAME>Braves</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Florida</TEAM\_CITY>

<TEAM\_NAME>Marlins</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Montreal</TEAM\_CITY>

<TEAM\_NAME>Expos</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>New York</TEAM\_CITY>

<TEAM\_NAME>Mets</TEAM\_NAME>

</TEAM>

<TEAM>

<TEAM\_CITY>Philadelphia</TEAM\_CITY>

<TEAM\_NAME>Phillies</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>Central</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Chicago</TEAM\_CITY>

<TEAM\_NAME>Cubs</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>West</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Arizona</TEAM\_CITY>

<TEAM\_NAME>Diamondbacks</TEAM\_NAME>

</TEAM>

</DIVISION>

</LEAGUE>

<LEAGUE>

<LEAGUE\_NAME>American</LEAGUE\_NAME>

<DIVISION>

<DIVISION\_NAME>East </DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Baltimore</TEAM\_CITY>

<TEAM\_NAME>Orioles</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>Central</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Chicago</TEAM\_CITY>

<TEAM\_NAME>White Sox</TEAM\_NAME>

</TEAM>

</DIVISION>

<DIVISION>

<DIVISION\_NAME>West</DIVISION\_NAME>

<TEAM>

<TEAM\_CITY>Anaheim</TEAM\_CITY>

<TEAM\_NAME>Angels</TEAM\_NAME>

</TEAM>

</DIVISION>

</LEAGUE>

</SEASON>

<SIGNATURE>

<COPYRIGHT>Copyright 1999 Elliotte Rusty Harold</COPYRIGHT>

<EMAIL>elharo@metalab.unc.edu</EMAIL>

<LAST\_MODIFIED>March 10, 1999</LAST\_MODIFIED>

</SIGNATURE>

</DOCUMENT>

在内部和外部 DTD 子集中的同名元素之间发生冲突的情况下, 内部声明的元素具有优先权。这种优先权提供了不完善部分的继承机制。例如, 如要推翻 PLAYER 元素的定义, 以便只包含击球统计数据, 而不要投球统计数据。这时可使用大多数的棒球 DTD 的相同声明, 但却要将 PLAYER 元素作如下改变:

<!DOCTYPE SEASON SYSTEM "baseball.dtd" [

<!ELEMENT PLAYER (GIVEN\_NAME, SURNAME, P, G,

GS, AB?, R?, H?, D?, T?, HR?, RBI?, SB?, CS?,

SH ?, SF?, E?, BB?, S?, HBP?)

>

]>

## 8.8 本章小结

在本章中，学习了如何使用 DTD 来描述文档结构，包括文档包含的必需元素和任选元素，以及这些元素间的相关关系。特别是学习了以下内容：

- 文档类型定义（DTD），它提供了文档包含的元素、标记、属性和实体及相互关系的清单。
- 文档序言包含文档类型声明，文档类型声明指明基本元素并包含 DTD。DTD 处在 XML 声明与实际文档开始之间。由<!DOCTYPE *ROOT* [和]>加以界定，*ROOT* 是基本元素名称。
- DTD 列出了文档的可允许的标记和结构。遵守 DTD 规则的文档才是合法的。
- 元素类型声明声明元素名称和子元素。
- 元素类型声明中用逗号分隔的子元素在文档中出现的顺序必须与声明中的顺序相同。
- 加号表示元素可以出现一次或多次。
- 星号表示元素可以出现零次或多次。
- 问号表示元素可以出现零次或一次。
- 竖线表示可以使用这一个也可以使用另一个元素。
- 括号可以组合子元素，以便使元素声明更详尽。
- 混合内容包含元素和可析的字符数据，但会限制父元素可实现的结构。
- 空元素用 EMPTY 关键词声明。
- 注释使 DTD 更具可读性。
- 在文档类型声明中利用 SYSTEM 关键词和一个 URL 可以定位外部 DTD。
- 在文档类型声明中用 PUBLIC 关键词可以定位标准 DTD。
- 内部 DTD 子集中的声明可推翻外部 DTD 子集中的声明。

在下一章中，读者可学到有关 DTD 的更多知识，包括实体引用如何提供文本替换，如何将 DTD 与它所描述的文档分开，以便易于在文档间共享。还会学到如何用多份 DTD 描述单个文档。



## 第 9 章 实体和外部 DTD 子集

一个简单的 XML 文档从许多不同的资源和文件中取得数据和声明。实际上，有些数据直接来自数据库、CGI 脚本或其他非文件格式资源。无论采取何种形式，保存 XML 文档片段的项目称为实体。实体引用把实体载入到 XML 主文档中。通用实体引用载入数据到 XML 文档的基本元素中，而参数实体引用载入数据到文档的 DTD 中。

本章的主要内容如下：

- 什么是实体？
- 内部通用实体
- 外部通用实体
- 内部参数实体
- 外部参数实体
- 怎样从局部开始创建文档
- 结构完整文档中的实体和 DTD

### 9.1 什么是实体？

从逻辑上说，一个 XML 文档由一个序进程构成，序进程后有一严密地包含了所有其他元素的基本元素。但 XML 文档的实际数据可以扩展分布在若干文档中。例如，即使一个棒球联盟中包含了大约 900 个的所有球员，每个 PLAYER 元素也可以以独立的文件形式存在。包含 XML 文档细节内容的存储单元称为实体（entities），实体可能是由一个文件、一个数据库记录或其他包含数据的项目组成。例如，本书中所有完整的 XML 文件都是实体。

包含 XML 声明或文档类型声明的存储单元和基本元素称为文档实体（document entity）。不过基本元素和它的派生元素也可包含指向即将插入文档的附加数据的实体引用。进行正确性检查的 XML 处理器在提交文档给最终应用程序以前或显示文件以前，将把所有不同的实体引用结合为单一逻辑结构的文档。



不进行正确性检查的处理器可以但不一定插入外部对象；他们必须插入内部对象。

实体的主要目的在于保存如下内容：结构完整的 XML，其他形式的文本或二进制数据。序进程和文档类型声明是它们所属文档的基本元素的一部分。仅当 XSL 样式单本身就是一个结构完整的 XML 文档时，才能作为一个实体。组成 XSL 样式单的实体并不是应用该样式单的 XML 文档的组成实体之一。CSS 样式单根本就不是一个实体。

大多数实体具有一个可以引用的名。唯一的例外是包含 XML 文档的主文件与文档实体（与数据库记录、CGI 程序的输出结果或其他数据相对比，文档实体也不一定是文件）。文档实体无论采取何种结构，都是一种存储单元，用于储存 XML 声明、文档类型声明（如果有）和基本元素。因此每个 XML 文档至少拥有一个实体。

有两种类型的实体：内部实体和外部实体。完全在文档实体内部定义的实体称为内部实体。文档本身就是这样的实体，所以所有的 XML 文档至少有一个内部实体。

相反，经由 URL 定位的资源中获取的数据称为外部实体。主文档仅包含一个实际引用数据位置的 URL。在 HTML 中，包含于<HTML>和</HTML>标记之间的文档本身是内部实体时，而 IMG 元素（实际的图像数据）代表外部实体。

实体分为两类：可析和不可析实体。可析实体包含结构完整的 XML 文本。不可析实体包含二进制数据或非 XML 文本（类似电子邮件信息）。如果从本质上说，当前大多数 XML 处理器不能很好地支持（如果不是完全支持的话）不可析实体，本章所关注的是可析实体。



第 11 章，非 XML 数据和不可析对象的嵌套。

通过实体引用，可把来源于多个实体的数据合并在一起构成一个单一的文档。通用实体引用把数据并入到文档内容中。参数实体引用把声明并入到文档的 DTD 中。实中<lt; 、<gt; 、<apos; 、<quote; 、<amp; 是预定义的体引用，分别指的是文本实体<、>、’、”、&符号。然而也可在文档 DTD 中定义新的实体。

.2 内部通用实体

内部通用实体引用可看作经常使用的文本或强制格式文本的缩写。DTD 中的<!ENTITY>标记定义缩写，并且该缩写就代替了文本。例如，可在 DTD 中简单地把页脚定义为实体 footer，然后每页只需键入&footer;，而无需在每页底部键入相同的页脚。此外，若决定更改页脚块（也许是因为你的电子邮件地址更改了），就仅需在 DTD 中作一次更改即可，无需对共享同一页脚的页面逐个进行更改。

通用实体引用以“&”符号开始，以“;”结尾，两个符号之间为实体名。例如，“&lt;”就是小于符号(<)的通用实体引用，实体名为 lt，该实体的替换文本就是一个字符“<”。实体名由字母和数字的混合排列以及下划线构成，禁止使用空格和其他标点符号字符。类似 XML 中的其他内容，实体引用是区分大小写的。



尽管从技术上说，允许在对象名中使用冒号“:”，但正如第 18 章中所提及，此符号要保留用于命名域(namespace)。

9.2.1 定义内部通用实体引用

在 DTD 中使用标记<!ENTITY>标记定义内部通用实体引用，具有如下格式：

```
<!ENTITY name "replacement text">
```

name 是 replacement text 的缩写。替换文本需放置于双引号中，因为其中可能包含空格和 XML 标记。可在文档中键入实体名，而读者所见为替换文本。

例如，我的名字为“Elliott Rusty Harold”（这得怪我父母取了一个长名）。即使经过多年习惯，我依然常常打错。我可以为我的名字定义通用实体引用，这样每次当我键入&ERH;时，读者将会看见“Elliott Rusty Harold”，这个定义如下：

```
<!ENTITY ERH " Elliott Rusty Harold">
```

清单 9-1 示例说明了&ERH; 通用实体引用，图 9-1 中显示了载入到 Internet Explorer 的文档。可看出，源代码中的&ERH; 实体引用输出时被替换为“Elliott Rusty Harold”。

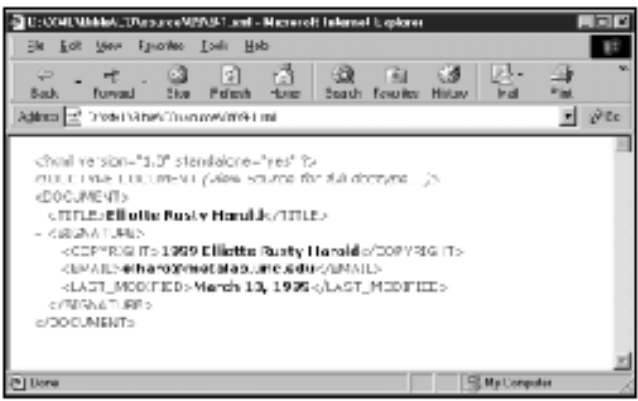


图 9-1 清单 9-1 在内部通用实体引用被实际实体替换后的情形

清单 9-1：ERH 内部通用实体引用

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

```
<!DOCTYPE DOCUME [
```

```

<!ENTITY ERH "Elliott Rusty Harold">

<!ELEMENT DOCUME (TITLE, SIGNATURE)>

<!ELEMENT TITLE (#PCDATA A)>

<!ELEMENT COPYRIGHT (#PCDATA)>

<!ELEMENT EMAIL (#PCDATA A)>

<!ELEMENT LAST_MODIFIED (#PCDATA)>

<!ELEMENT SIGNATURE (COPYRIGHT, EMAIL, LAST_MODIFIED)>

]>

<DOCUMENT>

<TITLE>&ERH;</TITLE>

<SIGNATURE>

<COPYRIGHT >1999 &ERH;</COPYRIGHT>

<EMAIL>elharo@metalab.unc.edu</EMAIL>

<LAS _MODIFIED>March 10, 1999</LAS _MODIFIED>

</SIGNATURE>

</DOCUMENT>

```

注意其中的通用实体引用，即使 COPYRIGHT 和 TITLE 声明为仅接受#PCDATA 的子类，&ERH; 依然出现在它们之中。因&ERH; 实体引用的替换文本是可析的字符数据，所以这种排列是合法的。所有的实体引用被实体值替换后，才对文档进行正确性检查。

在使用样式单时，也会发生相同的情形。当存在样式单引用，在实体引用替换为实体值后，才把样式应用于替换后实际存在的元素树状结构中。

可按下列方式把版权、电子邮件或最后的修改日期声明为通用实体引用：

```

<!ENTITY COPY99 "Copyright 1999">

<!ENTITY EMAIL "elharo@metalab.unc.edu">

<!ENTITY LM "Last modified: ">

```

因日期对不同的文档可能会发生改变，故而忽略了&LM; 中的日期。若把日期作为一个实体引用，不会带来任何好处。

现在，就可把清单 9-1 内容重写成更加简洁的形式：

```

<DOCUMENT>

```

<TITLE>&ERH;</TITLE>

<SIGNATURE>

<COPYRIGHT>&COPY99; &ERH;</COPYRIGHT>

<EMAIL>&EMAIL;</EMAIL>

<LAS \_MODIFIED>&LM; March 10, 1999</LAST\_MODIFIED>

</SIGNATURE>

</DOCUMENT>

应用实体引用代替书写文本全文的一个好处是使得更改文本更加简便,在简单的 DTD 被若干文档共享的情况下,特别有用。例如,假设把电子邮件地址从 elharo@metalab.unc.edu 更改为 eharold@solar.stanford.edu, 仅需按如下内容更改 DTD 中的一行内容,而无需在多个文档中寻找和替换邮件地址:

<!ENTITY EMAIL "eharold@solar.stanford.edu">

### 9.2.2 在 DTD 中使用通用实体引用

读者或许对是否能像下面的代码一样在一个通用实体引用中包含另一个通用实体引用表示怀疑,如下所示:

<!ENTITY COPY99 "copyright 1999 &ERH;">

实际上该例是合法的,因为 ERH 实体作为 COPY99 实体的一部分存在,而 COPY99 实体本身最终又成为文档内容的一部分。尽管存在某些限制,对于 DTD 中的其他地方,若最终能转换成文档内容的一部分(例如作为缺省属性值),则也可在此处使用通用实体引用。第一条限制:语句中不能使用如下的循环引用:

<!ENTITY ERH "&COPY99 Elliotte Rusty Harold">?

<!ENTITY COPY99 "copyright 1999 &ERH;">?

第二条限制:通用实体引用不能插入仅为 DTD 的一部分且不能作为文档内容的文本。例如,下述简略用法的企图无法实现:

<!ENTITY PCD "(#PCDATA)">

<!ELEMENT ANTIMAL &PCD;>

<!ELEMENT FOOD &PCD;>

然而,利用实体引用把文本合并到文档的 DTD 中的方法常常是有用的。为此目的,XML 使用将在下章中讲述的参数实体引用来实现这一目的。

对通用实体值的限制仅在于不能直接包含三种字符: % 、&、” ,可是能经过使用字符引用包含这三种字符。若&和%仅作为实体引用的开头,而不代表自身含义,则可包含其中。限制很少意味着实体可包含标记和分割为多行。例如下面的 SIGNATURE 实体是有效的:

"<SIGNATURE>

<COPYRIGHT>1999 Elliotte Rusty Harold</COPYRIGH >

<EMAIL>elharo@metalab.unc.edu</EMAIL>

<LAST\_MODIFIED>March 10, 1999</LAST\_MODIFIED>

</SIGNATURE>”

>

下一个关心的问题是实体是否可以拥有参数。能否使用上面的 SIGNATURE 实体,但却改变每页中每一独立的 LAST\_MODIFIED 元素的数据? 答案是否定的; 实体仅为静态的替换文本。若需要给实体传送数据, 应改为使用标记符, 并在样式单中随同提供适当的实现指令。

9.2.3 预定义通用实体引用

XML 预定义五个通用实体引用, 如表 9-1 所示。五个实体引用出现在 XML 文档中用来代替一些特殊的字符, 这些字符如果不用引用方式就会被解释为标记。例如实体引用&lt; 代表小于号<, 小于符号<可解释为标记的开头。

考虑到最大限度的兼容, 若计划使用预定义实体引用, 就该在 DTD 中声明这些引用。因为需要避免 DTD 中字符的递归引用, 所以声明时必须相当小心。为方便引用的声明, 字符引用使用字符的十六进制 ASCII 值。清单 9-2 显示了所需要的声明。

表 9-1 XML 中的预定义实体引用	
实体引用	字符
&amp;	&
&lt;	<
&gt;	>
&quot;	”
&apos;	

清单 9-2: 预定义通用实体引用声明

<!ENTITY lt ”&#38;#60;”>

<!ENTITY gt ”&#62;”>

<!ENTITY amp ”&#38;#38;”>

<!ENTITY apos ”&#39;”>

<!ENTITY quot ”&#34;”>

## 9.3 外部通用实体

包含基本元素/文档实体的主文件以外的数据称为外部实体。通过外部实体引用可在文档中嵌入外部实体和从若干相互独立的文件中获取数据组建为 XML 文档。

仅使用内部实体的文档非常类似于 HTML 模式。文档的完整文本存放于单一文件中，图像、JAVA 小程序、声音和非 HTML 数据也可链接入文件中，但至少在文件中要有所有的文本。当然，HTML 模式存在一些问题。特别在文档中嵌入动态信息的过程是一件非常困难的事情。可通过使用 CGI、JAVA 小程序所爱好的数据库软件、服务器方面提供的手段和其他各种各样的方法嵌入动态信息，但 HTML 仅提供静态文档支持。从若干文件中获取数据组建文档的行为必须在 HTML 外部进行。HTML 中解决这问题的最简单的方法是使用框架，但这是非常糟糕的用户界面，通常令用户迷惑和讨厌。

部分问题是 HTML 文档不能自然地插入到另一个 HTML 文档中，每个 HTML 文档有且仅有一个 BODY，服务器端嵌入法仅能提供把 HTML 片段嵌入文档的能力，而不是嵌入有效的文档实体，此外服务器端提供的引用需依赖于服务器的存在，而不是真正的 HTML 文档部分。

然而，XML 更加灵活。一个文档的基本元素没有必要与另一文档基本元素相同。即使两个文档共享同一基本元素，DTD 也可声明元素对自身的包含。在适当的时候，XML 标准并不制止结构完整的 XML 文档嵌入另一结构完整的 XML 文档的做法。

但是，XML 走得更远一些，可定义一个机制，利用这机制可在若干本地或远程系统上的、较小的 XML 文档的基础上建立新的 XML 文档。语法分析器的任务就是按固定的序列把所有不同文档组合起来。文档中可包含另一文档，或许这个还包含其他文档。只要没有递归的文档包含行为（处理器可报告的错误），应用程序就仅能看见一个单一、完整的文档。从本质上说，这种机制提供客户端嵌入的功能。

对 XML 而言，可使用外部通用实体引用的方法，在文档中嵌入另一文档。在 DTD 中，可按下述语法结构声明外部引用：

```
<!ENTITY name SYSTEM "URI">
```

URI 就是 Uniform Resource Identifier，与 URL 类似，但允许更加精确的资源链接规范。从理论上说，URI 把资源与其储存位置分开，所以 Web 浏览器可以选择最近的或最空闲的几个镜像几个镜像站点，而无需明确指明该链接。URI 领域的研究非常活跃，争论激烈，因此在实际应用中和在本书中，URI 就是多用途的 URL。

例如，可能想在站点的每个页面中都放置相同的签字块。为明确所见，我们假设签字块为清单 9-3 所示的 XML 代码，而且假定可从 URL <http://metalab.unc.edu/xml/signature.xml> 上获得该代码。

清单 9-3: XML 签字文件

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

<SIGNATURE>

<COPYRIGHT>1999 Elliotte Rusty Harold</COPYRIGHT>

<EMAIL>elharo@metalab.unc.edu</EMAIL>

</SIGNATURE>
```

在 DTD 中添加如下声明，可把实体引用&SIG;与这个文件联系在一起：

```
<!ENTITY SIG SYSTEM "http://metalab.unc.edu/xml/signature.xml">
```

也可使用相关的 URL。例如：

```
<!ENTITY SIG SYSTEM "xml/signature.xml">
```

如果被引用的文件放置于与引用该文件的文件所处的同一目录中，那么仅需使用一文件名进行引用。例如：

```
<!ENTITY SIG SYSTEM "signature.xml">
```

利用上述任一种声明，仅需使用&SIG;，就可在文档的任意位置引用签字文件的内容。如清单 9-4 中的简单的文档，图 9-2 显示的是在 Internet Explorer 5.0 中交付的文档。

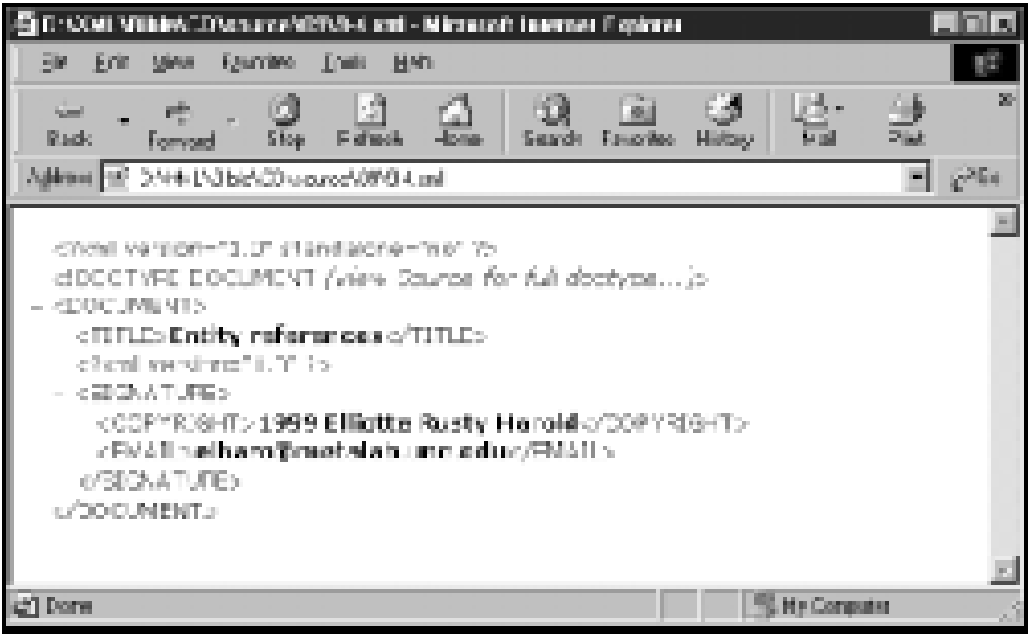


图 9-2 使用外部通用实体引用的文档

清单 9-4: SIG 外部通用实体引用

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

<!DOCTYPE DOCUMENT [

<!ELEMENT DOCUMENT (TITLE, SIGNATURE)>

<!ELEMENT TITLE (#PCDATA)>

<!ELEMENT COPYRIGHT (#PCDATA)>

<!ELEMENT EMAIL (#PCDATA)>

<!ELEMENT SIGNATURE (COPYRIGHT, EMAIL)>

<!ENTITY SIG SYSTEM

"http://metalab.unc.edu/xml/signature.xml"?

]>

</DOCUMENT>
```



<TITLE>Entity references</TITLE>

&SIG;

</DOCUMENT>

注意外部实体引用的附加作用，因为文件不再完整，所以 XML 声明中的 standalone 属性值为 no。解析文件表明该文件需要外部文件 signature.xml 中的数据。

## . 4 内部参数实体

通用实体成为文档的一部分，而非 DTD 的组成成分。通用实体引用仅可用于 DTD 中能成为文档组成部分的位置上；通用实体引用不能插入那些仅为 DTD 而非文档内容的文本。然而在 DTD 中的实体引用通常是有用的，因此 XML 提供了参数实体引用的手段。

除了如下两个关键处不同，参数实体引用与通用实体引用非常相似：

1. 参数实体引用以百分号%开始，而非&符号。
2. 参数实体引用仅可在 DTD 中出现，而不能位于文档内容中。

参数实体引用在 DTD 中声明的方法与通用实体类似，只是在实体名前加一百分号。句法结构如下：

```
<!ENTITY % name "replacement text">
```

实体名为实体内容的缩写。读者所见为双引号间的替换文本。例如：

```
<!ENTITY % ERH "Elliott Rusty Harold">
```

```
<!ENTITY COPY99 "Copyright 1999 %ERH;">
```

当使用参数实体引用替换通用实体引用后，前文中无法实现的缩写（#PCDATA）变为有效：

```
<!ENTITY % PCD "(#PCDATA)">
```

```
<!ELEMENT ANIMAL %PCD;>
```

```
<!ELEMENT FOOD %PCD;>
```

在元素间的子元素和属性共享通用列表中呈现参数实体引用的真实值。若替换的文本块较大且使用次数较多，则参数实体引用用处更大。例如，假设在 DTD 声明中有许多层次结构元素，如 PARAGRAPH、CELL 和 HEADING。每个该类元素都有确定的内联元素数目，类似 PERSON、DEGREE、MODEL、PRODUCT、ANIMAL、INGREDIENT 等等的内部元素。这些元素的声明可能表现为下述方式：

```
<!ELEMENT PARAGRAPH
```

```
(PERSON | DEGREE | MODEL | PRODUCT | ANIMAL | INGREDIENT)*>
```

```
<!ELEMENT CELL
```

```
(PERSON | DEGREE | MODEL | PRODUCT | ANIMAL | INGREDIENT)*>
```

```
<!ELEMENT HEADING
```

```
(PERSON | DEGREE | MODEL | PRODUCT | ANIMAL | INGREDIENT)*>
```

容器元素都有相同的内容。假如创建了一个新元素如 EQUATION、CD 或 ACCOUNT，则该元素必须声明为所有三个元素的子元素。若在前两个元素中添加新元素，却忘了在第三个元素中添加，就会引发问题。若元素的数目为 30 或者 300 个，而非 3 个，则问题将成倍增加。

若对每个容器元素不是给出单独的子元素列表，则 DTD 的维护将较为简便。替代的方法是子元素列表变为参数实体引用，然后在每个声明元素声明中应用参数实体引用。例如：

```
<!ENTITY % inlines
```

```
"(PERSON | DEGREE | MODEL | PRODUCT | ANTIMAL | INGREDIENT)*">?
```

```
<!ELEMENT PARAGRAPH %inlines;>
```

```
<!ELEMENT CELL %inlines;>
```

```
<!ELEMENT HEADING %inlines;>
```

当添加新元素时，仅需简单地改变一个参数实体声明，而无需改变 3 个、30 或 300 个元素声明。

参数实体引用必须在使用前声明。下例是非法的，因为%PCD；引用在使用两次后才加以声明：

```
<!ELEMENT FOOD %PCD;>
```

```
<!ELEMENT ANTIMAL %PCD;>
```

```
<!ENTITY % PCD "(#PCDATA)">
```

参数实体引用仅能用于提供外部 DTD 子集中的声明。也就是说，参数实体引用仅能出现在外部 DTD 子集的声明中，上述例子若用于内部 DTD 子集，则所有引用无效。

在内部 DTD 子集中，参数实体引用仅能用于声明之外。例如下述语句在内部和外部 DTD 子集中均有效。

```
<!ENTITY % hr "<!ELEMENT HR EMPTY">">
```

```
%hr;
```

当然，这与将 HR 元素声明为不带参数实体引用相比没有带来使用上的便利：

```
<!ELEMENT HR EMPTY>
```

参数实体引用主要用于内部 DTD 子集引用外部参数实体的情况；也就是引入不同文件中的声明或部分声明。下一节将讲述这部分内容。

## 9.5 外部参数实体

前述例子中使用单一的 DTD，用于定义文档中所有的元素。然而文档越长，这种技术应用越少。此外通常希望将 DTD 中的部分内容用于许多不同的地方。

例如，对描述很少发生变化的邮件地址 DTD 来说，地址定义非常普遍，且可很方便地应用在不同的上下文中。类似地，清单 9-2 中列出的预定义实体引用可用于大部分 XML 文档中，但并不想总是对此清单进行拷贝和复制的操作。

可用外部参数实体把较小的 DTD 组成大型的 DTD。也就是说，一个外部 DTD 可以链接到另一外部 DTD，第二个 DTD 引入第一个 DTD 中声明的元素和实体。尽管严禁使用循环——若 DTD2 引用 DTD1，则 DTD1 不能引用 DTD2；但嵌套的 DTD 也会大型化和复杂化。

同时，将 DTD 分解为小的、更便于管理的组块，使得对 DTD 的分析处理更加简便。由于一个实体文档和完整的 DTD 存储在单一的文件中，在前几章中的许多例子都过于庞大。若文档和文档的 DTD 分割为几个独立的文件，就变得更加易于理解。

此外，描述一组元素的 DTD 中采用较小的、模块化的结构，使得不同的人或组织创建的 DTD 之间的组合和匹配更加简便。例如，在写一篇关于高温超导的文章，可能会用到描述其中分子的分子科学 DTD、记录公式的数学 DTD、描述图形的向量 DTD 和处理解释性文本的 HTML DTD。



特殊情况下，可使用 Peter Murray-Rust 的 Chemical Markup Language 中的 mol.dtd DTD、W3C 的 Mathematical Markup Language 中的 MathML DTD、W3C 的 Scalable Vector Graphics 中的 SVG DTD 和 W3C 的 XHTML DTD。

我们还可以想出许多混合或者匹配来自不同领域的概念（也就是标记）的例子。人类的想法不会局限在狭窄的定义范围内，总是试图遍及所有领域。所编写的文档就反映了这种思想。

让我们研究如何把棒球比赛统计表组织为几个不同的 DTD 的联合体。本例的层次非常分明。一个可能的分割方法是为 PLAYER、TEAM 和 SEASON 分别编写一个 DTD。分割 DTD 为更便于管理的方法远不止一种，但这也不失为一个很好的例子。清单 9-5 显示的是只为 PALYER 建立的单独的 DTD，保存在 player.dtd 文件中。

清单 9-5: PLAYER 元素和它的子元素的 DTD(player.dtd)

```
<!--Player Info -->

<!ELEMENT PLAYER (GIVEN _N AME, SURNAME, P, G,
GS, AB?, R?, H?, D?, ?, HR?, RBI?, SB?, CS?,
SH?, SF?, E?, BB?, S?, HBP?, W?, L?, SV?, CG?, SO?, ERA?,
IP?, HRA?, RA?, ER?, HB?, WP?, B?, WB?, K?)
>

<!--Player s last name -->

<!ELEMENT SURNAME (#PCDATA)>

<!--Player s first name -->
```

<!ELEMENT GIVE \_ NAME (#PCDATA)>

<!--Position -->

<!ELEMENT P (#PCDATA)>

<!--Games Played -->

<!ELEMENT G (#PCDATA)> <!--Games Started -->

<!ELEMENT GS (#PCDATA)>

<!--===== -->

<!--Batting Statistics -->

<!--At Bats -->

<!ELEMENT AB (#PCDATA)>

<!--Runs -->

<!ELEMENT R (#PCDATA)>

<!--Hits -->

<!ELEMENT H (#PCDATA)>

<!--Doubles -->

<!ELEMENT D (#PCDATA)>

<!--Triples -->

<!ELEMENT T (#PCDATA)>

<!--Home Runs -->

<!ELEMENT HR (#PCDATA)>

<!--Runs Batted In -->

<!ELEMENT RBI (#PCDATA)>

<!--Stolen Bases -->

<!ELEMENT SB (#PCDATA)>

<!--Caught Stealing -->

<!ELEMENT CS (#PCDATA)>

```
<!--Sacrifice Hits -->

<!--ELEMENT SH (#PCDATA)>

<!--Sacrifice Flies -->

<!--ELEMENT SF (#PCDATA)>

<!--Errors -->

<!--ELEMENT E (#PCDATA)>

<!--Walks (Base on Balls) -->

<!--ELEMENT BB (#PCDATA)>

<!--Struck Out -->

<!--ELEMENT S (#PCDATA)>

<!--Hit By Pitch -->

<!--ELEMENT HBP (#PCDATA)>

<!--===== -->

<!--Pitching Statistics -->

<!--Complete Games -->

<!--ELEMENT CG (#PCDATA)>

<!--Wins -->

<!--ELEMENT W (#PCDATA)>

<!--Losses -->

<!--ELEMENT L (#PCDATA)>

<!--Saves -->

<!--ELEMENT SV (#PCDATA)>

<!--Shutouts -->

<!--ELEMENT SO (#PCDATA)>

<!--ERA -->

<!--ELEMENT ERA (#PCDATA)>
```

```

<!--Innings Pitched -->

<!ELEMENT IP (#PCDATA)>

<!--Home Runs hit Against -->

<!ELEMENT HRA (#PCDATA)>

<!--Runs hit Against -->

<!ELEMENT RA (#PCDATA)>

<!--Earned Runs -->

<!ELEMENT ER (#PCDATA)>

<!--Hit Batter -->

<!ELEMENT HB (#PCDATA)>

<!--Wild Pitches -->

<!ELEMENT WP (#PCDATA)>

<!--Balk -->

<!ELEMENT B (#PCDATA)>

<!--Walked Batter -->

<!ELEMENT WB (#PCDATA)>

<!--Struck Out Batter -->

<!ELEMENT K (#PCDATA)>

<!--===== -->

<!--Fielding Statistics -->

<!--Not yet supported -->

```

当时用这个文件，这个 DTD 还无法让你创建非常有趣的文档，清单 9-6 显示的是仅使用清单 9-5 中 PLAYER DTD 的简洁有效的文件。从这来说，这简单的文件并不重要；然而，可在这些较小的部分外创建更加复杂的文件。

清单 9-6：使用 PLAYER DTD 的有效文档

```

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

<!DOCTYPE PLAYER SYSTEM "Player.dtd">

<PLAYER>

```

<GIVEN\_NAME>Chris</GIVEN\_NAME>

<SURNAME>Hoiles</SURNAME>

<P>Catcher</P>

<G>97</G>

<GS>81</GS>

<AB>267</AB>

<R>36</R>

<H>70</H>

<D>12</D>

<T>0</T>

<HR>15</HR>

<RBI>56</RBI>

<SB>0</SB>

<CS>1</CS>

<SH>5</SH>

<SF>4</SF>

<E>3</E>

<BB>38</BB>

<S>50</S>

<HBP>4</HBP>

</PLAYER>

文档的哪部分可拥有自己的 DTD？这是显而易见的，TEAM 就是其中的主要部分，可按如下方式书写它的 DTD：

<!ELEMENT TEAM ( EAM\_CITY, EAM\_NAME, PLAYER\*)>

<!ELEMENT TEAM\_CITY (#PCDATA)>

<!ELEMENT TEAM\_NAME (#PCDATA)>

然而作仔细的检查之后，就会注意到遗漏了某些东西：PLAYER 元素的定义。该定义位于 player.dtd 独立文件中，需要连接到这个 DTD 中。



可通过外部参数实体引用连接 DTD。对私有的 DTD，可按下列格式进行连接：

```
<!ENTITY % name SYSTEM "URI">
```

```
%name;
```

例如：

```
<! ENTITY % player SYSTEM "Player.dtd">
```

```
%player;
```

本例中使用了相对的 URL (player.dtd)，且假定 player.dtd 文件所在位置与进行链接的 DTD 的位置相同。若非这种情况，可使用完整的 URL 如下：

```
<! ENTITY % player SYSTEM
```

```
"http://metalab.unc.edu/xml/dtds/player.dtd">
```

```
%player;
```

清单 9-7 显示的是包含了对 PLAYER DTD 引用的完整 TEAM DTD：

清单 9-7：TEAM DTD (team.dtd)

```
<!ELEMENT EAM ( EAM_CITY, EAM_ NAME, PLAYER*)>
```

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

```
<!ELEMENT EAM_ NAME (#PCDATA)>
```

```
<!ENTITY % player SYSTEM "Player.dtd">
```

```
%player;
```

SEASON 包含 LEAGUE、DIVISION 和 TEAM 元素。尽管 LEAGUE 和 DIVISION 元素可拥有自己的 DTD，也没有必要过分追求使用各自独立的 DTD。除非希望拥有包含 LEAGUE 或 DIVISION 元素的文档，该文档不是 SEASON 的一部分，在这种情况下，才可在同一 DTD 中引用所有三个 DTD。如清单 9-8 中说明了这种情况。

清单 9-8：SEASON DTD (seasom.dtd)

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

```
<!ELEMENT LEAGUE (LEAGUE_NAME, DIVISION, DIVISION, DIVISION )>
```

```
<!--NAMErican or National -->
```

```
<!ELEMENT LEAGUE_ NAME (#PCDATA)>
```

```
<!--East, West, or Central -->
```

```
<!ELEMENT DIVISION_ NAME (#PCDATA)>
```

<!ELEMENT DIVISION (DIVISIO \_ NAME, EAM+)>

<!ELEMENT SEASON (YEAR, LEAGUE, LEAGUE)>

<!ENTITY % team SYSTEM "team.dtd">

%team;

## .6 根据片段创建文档

棒球的例子已相当庞大，尽管本书中的例子仅为缩减的版本，其中球员数目受到限制，但全文已超过 0.5MB，内容过于庞大，不便于装载和查询；特别是在读者仅对其中某一队员、球队或分部感兴趣时，尤其如此。本章中上一节讲述的技术可允许把这个文档分割为许多不同的、较小的、便于管理的文档，每个球队、队员、分部和联盟各自对应一个文档。通过外部实体引用，队员组成球队，球队组成分部，分部构成联盟，联盟构成赛季。

遗憾的是，无法按外部可析实体的样式嵌入 XML 文档。考虑一下，例如清单 9-9 ChrisHoiles.xml，这是清单 9-6 的修订版本。然而，若仔细检查两个清单，将发现它们的序进程是不同的。9-6 清单的序进程为：

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

```
<!DOCTYPE PLAYER SYSTEM "Player.dtd">
```

清单 9-9 的序进程是简单的 XML 声明，没有 standalone 属性，但却有 encoding 属性；而且忽略了文档类型声明。像清单 9-9 这样的文件表明将被嵌入另一文件中，其中的 XML 声明称为文本声明，虽然正如我们所看到的，它实际上正是一个合法的 XML 声明。

清单 9-9: ChrisHoiles.xml

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

```
<PLAYER>
```

```
<GIVEN_ NAME>Chris</GIVEN_NAME>
```

```
<SURNAME>Hoiles</SURNAME>
```

```
<P>Catcher</P>
```

```
<G>97</G>
```

```
<GS>81</GS>
```

```
<AB>267</AB>
```

```
<R>36</R>
```

```
<H>70</H>
```

```
<D>12</D>
```

```
<T>0</T>
```

```
<HR>15</HR>
```

```
<RBI>56</RBI>
```

```
<SB>0</SB>
```

```
<CS>1</CS>
```

<SH>5</SH>

<SF>4</SF>

<E>3</E>

<BB>38</BB>

<S>50</S>

<HBP>4</HBP>

</PLAYER>



虽然可在本书附带的 CD-ROM 上的 example\baseball\player 目录中找到所有队员名单，但这里省略了大约 1200 名的队员名单。

文档声明必须具有 encoding 属性（与 XML 声明不同，XML 声明可以拥有 encoding 属性，但不是必要的），encoding 属性规定实体使用的字符集。允许使用不同字符组写出的复合文档。例如，Latin-5 字符组写出的文档可与 UTF-8 字符集写出的文档结合为一体。处理器或浏览器依然必须理解不同实体使用的编码。

本章中的所有例子以 ASCII 编码形式给出。因 ASCII 编码是 ISO Latin-1、UTF-8 的严格子集，所以可以使用如下的任一文本声明：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

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

清单 9-10 mets.dtd 和清单 9-11 mets.xml 显示如何利用外部可析实体组建完整的球队文档。在 DTD 中为球队中的每个队员定义外部实体引用。利用文档内部 DTD 子集中的外部参数实体引用，XML 文档载入该 DTD；然后，该文档包括许多外部通用实体引用来载入分立的队员数据。

清单 9-10：具有 player 实体引用的 New York Mets DTD (mets.dtd)

```
<!ENTITY AllLeiter SYSTEM "mets/AllLeiter.xml">
```

```
<!ENTITY ArmandoReynoso SYSTEM "mets/ArmandoReynoso.xml">
```

```
<!ENTITY BobbyJones SYSTEM "mets/BobbyJones.xml">
```

```
<!ENTITY BradClontz SYSTEM "mets/BradClontz.xml">
```

```
<!ENTITY DennisCook SYSTEM "mets/DennisCook.xml">
```

```
<!ENTITY GregMcMichael SYSTEM "mets/GregMcMichael.xml">
```

```
<!ENTITY HideoNomo SYSTEM "mets/HideoNomo.xml">
```

```
<!ENTITY JohnFranco SYSTEM "mets/JohnFranco.xml">
```

```
<!ENTITY JosiasManzanillo SYSTEM "mets/JosiasManzanillo.xml">

<!ENTITY OctavioDotel SYSTEM "mets/OctavioDotel.xml">

<!ENTITY RickReed SYSTEM "mets/RickReed.xml">

<!ENTITY RigoBeltran SYSTEM "mets/RigoBeltran.xml">

<!ENTITY WillieBlair SYSTEM "mets/WillieBlair.xml">
```

图 9-3 显示了载入到 Internet Explorer 中的 XML 文档。请注意即使主文档仅包含存储队员数据的实体引用，所有队员数据也能被显示出来。Internet Explorer 解决了所有外部引用，这可不是所有的 XML 语法分析程序或者浏览器都能做到的。

在 CD-ROM 上的 example\baseball 目录中可找到其余球队。请特别需要注意，简洁的外部实体引用是如何嵌入多个队员数据的。

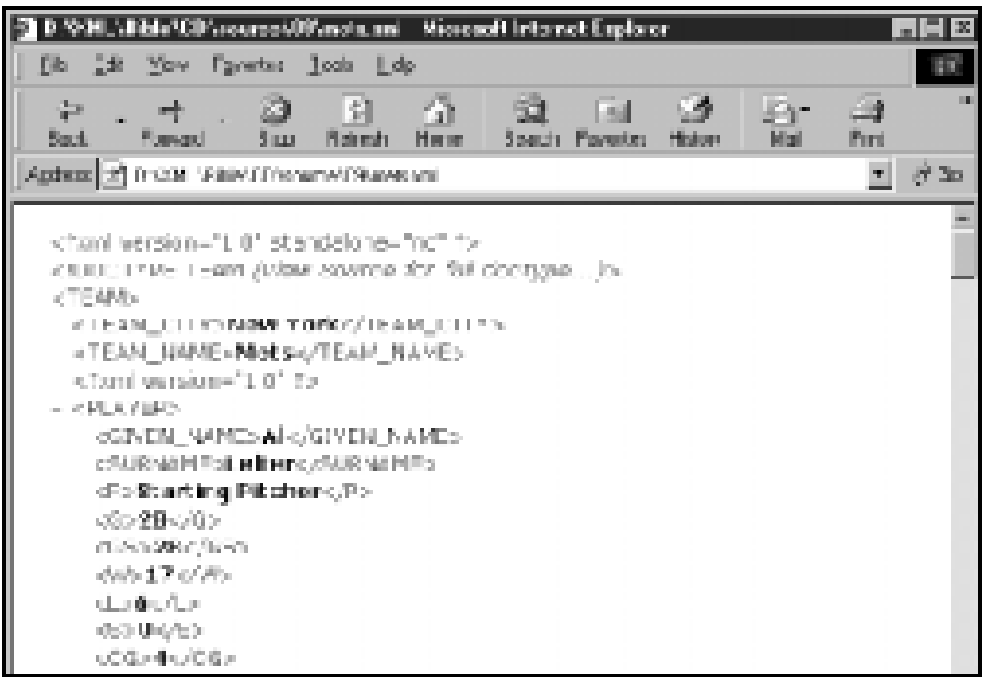


图 9-3 XML 文档显示 1998 年 New York Mets 队中的所有球员。

清单 9-11: 具有从外部实体中载入的队员数据的 New York Mets (mets.xml)

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

<!DOCTYPE TEAM SYSTEM "team.dtd"[

<!ENTITY % players SYSTEM "mets.dtd">

%players;

]>

<TEAM>

< TEAM_CITY>New York</ TEAM_CITY>
```

```
< TEAM_ NAME>Mets</ TEAM_ NAME>
```

```
&AlLeiter;
```

```
&ArmandoReynoso;
```

```
&BobbyJones;
```

```
&BradClontz;
```

```
&DennisCook;
```

```
&GregMcMichael;
```

```
&HideoNomo;
```

```
&JohnFranco;
```

```
&JosiasManzanillo;
```

```
&OctavioDotel;
```

```
&RickReed;
```

```
&RigoBeltran;
```

```
&WillieBlair;
```

```
</TEAM>
```

通过组合球队文件创建分部、通过组合分部文件创建联盟、通过组合联盟文件创建赛季的过程的延续，具有一定的好处。但遗憾的是，所有努力只会带来灾难性的后果。通过外部实体的方法嵌套的文档不能拥有自身的 DTD。最多只能是序进程包含文本声明。这就是说，仅能拥有单一层次的文本嵌入。与此不同的是，DTD 嵌入可进行任意层次的嵌套。

因此唯一可用的方法就是，在引用了许多不同球员文档的单一文档中包括所有球队、分部、联盟和赛季。需要 1200 多个实体声明（每个队员对应一个声明）。因为 DTD 可以深层嵌套，就引入如清单 9-10 所示包含所有球队定义的 DTD 开始。如清单 9-12 所示。

清单 9-12: 球员的 DTD (players.dtd)

```
<!ENTITY % angels SYSTEM "angels.dtd">
```

```
%angels;
```

```
<!ENTITY % astros SYSTEM "astros.dtd">
```

```
%astros;
```

```
<!ENTITY % athletics SYSTEM "athletics.dtd">
```

```
%athletics;
```

<!ENTITY % bluejays SYSTEM "bluejays.dtd">

%bluejays;

<!ENTITY % braves SYSTEM "braves.dtd">

%braves;

<!ENTITY % brewers SYSTEM "brewers.dtd">

%brewers;

<!ENTITY % cubs SYSTEM "cubs.dtd">

%cubs;

<!ENTITY % devilrays SYSTEM "devilrays.dtd">

%devilrays;

<!ENTITY % diamondbacks SYSTEM "diamondbacks.dtd">

%diamondbacks;

<!ENTITY % dodgers SYSTEM "dodgers.dtd">

%dodgers;

<!ENTITY % expos SYSTEM "expos.dtd">

%expos;

<!ENTITY % giants SYSTEM "giants.dtd">

%giants;

<!ENTITY % indians SYSTEM "indians.dtd">

%indians;

<!ENTITY % mariners SYSTEM "mariners.dtd">

%mariners;

<!ENTITY % marlins SYSTEM "marlins.dtd">

%marlins;

<!ENTITY % mets SYSTEM "mets.dtd">

%mets;

<!ENTITY % orioles SYSTEM "orioles.dtd">

%orioles;

<!ENTITY % padres SYSTEM "padres.dtd">

%padres;

<!ENTITY % phillies SYSTEM "phillies.dtd">

%phillies;

<!ENTITY % pirates SYSTEM "pirates.dtd">

%pirates;

<!ENTITY % rangers SYSTEM "rangers.dtd">

%rangers;

<!ENTITY % redsox SYSTEM "redsox.dtd">

%redsox;

<!ENTITY % reds SYSTEM "reds.dtd">

%reds;

<!ENTITY % rockies SYSTEM "rockies.dtd">

%rockies;

<!ENTITY % royals SYSTEM "royals.dtd">

%royals;

<!ENTITY % tigers SYSTEM "tigers.dtd">

%tigers;

<!ENTITY % twins SYSTEM "twins.dtd">

%twins;

<!ENTITY % whitesox SYSTEM "whitesox.dtd">

%whitesox;

<!ENTITY % yankees SYSTEM "yankees.dtd">

%yankees;



清单 9-13 为主控文档,把所有队员的子文档和定义每个队员的 DTD 组合为一体。尽管该文档比以前产生的单一文档小(32k 与 628k 之比),但仍然太长,所以无法在此引入所有队员。清单 9-13 的完整版本要依靠 33 个 DTD 和 1000 多个 XML 文件来生成最终文档。这种方法最大的问题在于,在显示文档之前,需要 1000 多个与 Web 服务器的链接。



完整的例子位于光盘上的 example/baseball/players/index.xml 文件中。

清单 9-13: 利用球员的外部实体引用的 1998 年赛季的主控文档

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

<!DOCTYPE SEASO SYSTEM "baseball.dtd"[

<!ENTITY % players SYSTEM "players.dtd">

%players;

]>

<SEASO >

<YEAR>1998</YEAR>

<LEAGUE>

<LEAGUE_NAME>ational</LEAGUE_NAME>

<DIVISION >

<DIVISION _NAME>East</DIVISION _NAME>

<TEAM>

<TEAM_CITY>Florida</TEAM_CITY>

<TEAM_NAME>Marlins</TEAM_NAME>

</TEAM>

<TEAM>

<TEAM_CITY>Montreal</TEAM_CITY>

<TEAM_NAME>Expos</TEAM_NAME>

</TEAM>

<TEAM>

<TEAM_CITY> New York</TEAM_CITY>
```

<TEAM\_NAME>Mets</TEAM\_NAME>

&RigoBeltran;

&DennisCook;

&SteveDecker;

&JohnFranco;

&MattFranco;

&ButchHuskey;

&BobbyJones;

&MikeKinkade;

&HideoNomo;

&VanceWilson;

</TEAM>

<TEAM>

<TEAM\_CITY>Philadelphia</TEAM\_CITY>

<TEAM\_NAME>Phillies</TEAM\_NAME>

</TEAM>

</DIVISION >

<DIVISION >

<DIVISION \_NAME>Central</DIVISION \_NAME>

<TEAM>

<TEAM\_CITY>Chicago</TEAM\_CITY>

<TEAM\_NAME>Cubs</TEAM\_NAME>

</TEAM>

</DIVISION >

<DIVISION >

<DIVISION \_NAME>West</DIVISION \_NAME>

<TEAM>

<TEAM\_CITY>Arizona</TEAM\_CITY>

<TEAM\_NAME>Diamondbacks</TEAM\_NAME>

</TEAM>

</DIVISION >

</LEAGUE>

<LEAGUE>

<LEAGUE\_NAME>American</LEAGUE\_NAME>

<DIVISION >

<DIVISION \_NAME>East</DIVISION \_NAME>

<TEAM>

<TEAM\_CITY>Baltimore</TEAM\_CITY>

<TEAM\_NAME>Orioles</TEAM\_NAME>

</TEAM>

</DIVISION >

<DIVISION >

<DIVISION \_NAME>Central</DIVISION \_NAME>

<TEAM>

<TEAM\_CITY>Chicago</TEAM\_CITY>

<TEAM\_NAME>White Sox</TEAM\_NAME>

&JeffAbbott;

&MikeCameron;

&MikeCaruso;

&LarryCasian;

&TomFordham;

&MarkJohnson;

```

    &RobertMachado;

    &JimParque;

    &ToddRizzo;

</TEAM>

</DIVISION >

<DIVISION >

<DIVISION _NAME>West</DIVISION _NAME>

<TEAM>

<TEAM_CITY>Anaheim</TEAM_CITY>

<TEAM_NAME>Angels</TEAM_NAME>

</TEAM>

</DIVISION >

</LEAGUE>

</SEASON>

```

在选择主文档和嵌套数据的层次结构上具有一定的灵活性。例如，一种可选择的结构就是在清单 9-12 中使用的，把球队和所有队员的数据放在不同的文件中；然后把球队数据组合为带外部实体的赛季文件，如清单 9-14 所示。使用尺寸更小、数目更少的 XML 文件的好处在于 Web 服务器中所占的空间更小以及下传和显示更加快捷。可是老实地说，一种方法或其他方法所带来的内在的益处很小。请放心大胆使用任意更严密地与数据组织相匹配，或者任一感觉使用方便的简洁方式。

清单 9-14：利用对球员的外部实体引用球队的 1998 年赛季的主控文档

```

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

<!DOCTYPE SEASO SYSTEM "baseball.dtd"[

<!ENTITY angels SYSTEM "angels.xml">

<!ENTITY astros SYSTEM "astros.xml">

<!ENTITY athletics SYSTEM "athletics.xml">

<!ENTITY bluejays SYSTEM "bluejays.xml">

<!ENTITY braves SYSTEM "braves.xml">

<!ENTITY brewers SYSTEM "brewers.xml">

<!ENTITY cubs SYSTEM "cubs.xml">

```

<!ENTITY devilrays SYSTEM "devilrays.xml">

<!ENTITY diamondbacks SYSTEM "diamondbacks.xml">

<!ENTITY dodgers SYSTEM "dodgers.xml">

<!ENTITY expos SYSTEM "expos.xml">

<!ENTITY giants SYSTEM "giants.xml">

<!ENTITY indians SYSTEM "indians.xml">

<!ENTITY mariners SYSTEM "mariners.xml">

<!ENTITY marlins SYSTEM "marlins.xml">

<!ENTITY mets SYSTEM "mets.xml">

<!ENTITY orioles SYSTEM "orioles.xml">

<!ENTITY padres SYSTEM "padres.xml">

<!ENTITY phillies SYSTEM "phillies.xml">

<!ENTITY pirates SYSTEM "pirates.xml">

<!ENTITY rangers SYSTEM "rangers.xml">

<!ENTITY redsox SYSTEM "red sox.xml">

<!ENTITY reds SYSTEM "reds.xml">

<!ENTITY rockies SYSTEM "rockies.xml">

<!ENTITY royals SYSTEM "royals.xml">

<!ENTITY tigers SYSTEM "tigers.xml">

<!ENTITY twins SYSTEM "twins.xml">

<!ENTITY whitesox SYSTEM "whitesox.xml">

<!ENTITY yankees SYSTEM "yankees.xml">

]>

<SEASON >

<YEAR>1998</YEAR>

<LEAGUE>

<LEAGUE\_NAME> ational</LEAGUE\_NAME>

<DIVISION >

<DIVISION \_NAME>East</DIVISION \_NAME>

&marlins;

&braves;

&expos;

&mets;

&phillies;

</DIVISION >

<DIVISION >

<DIVISION \_NAME>Central</DIVISION \_NAME>

&cubs;

&reds;

&astros;

&brewers;

&pirates;

</DIVISION >

<DIVISION >

<DIVISION \_NAME>West</DIVISION \_NAME>

&diamondbacks;

&rockies;

&dodgers;

&padres;

&giants;

</DIVISION >

</LEAGUE>

<LEAGUE>

<LEAGUE\_NAME>American</LEAGUE\_NAME>

<DIVISION >

<DIVISION \_NAME>East</DIVISION \_NAME>

&orioles;

&redsox;

&yankees;

&devilrays;

&bluejays

</DIVISION >

<DIVISION >

<DIVISION \_NAME>Central</DIVISION \_NAME>

&whitesox;

&indians;

&tigers;

&royals;

&twins;

</DIVISION >

<DIVISION >

<DIVISION \_NAME>West</DIVISION \_NAME>

&angels;

&athletics;

&mariners;

&rangers;

</DIVISION >

</LEAGUE>

</SEASON >

最后，较少使用的方法是，从外部球员实体的基础上创建各分立的球队文件，然后组合所有球队文件为分部、联盟和赛季。主控文档中可定义用于子球队文档中的实体引用。可是在这种情况下，因为实体引用集合在主控文档以前未被定义，所以球队文档不可用于自身。

真正的缺点是仅有顶层文档可附加于 DTD 之上。这是对外部可析实体用途的一种限制。无论如何，当学习了 XLinks 和 XPointers 后，可以明白创建大型、复杂文档的其他方法。然而，那些技术不是 XML 标准的核心部分内容，进行正确性检查的 XML 处理器和 Web 浏览器并无必要像支持本章讲述的技术一样去支持这些技术。



Xlinks 将在第 16 章讲述，XPointers 将在第 17 章讲述。



## 9.7 结构完整的文档中的实体和 DTD

本书第一部分研究了无 DTD 的结构完整的 XML 文档，第二部分研究包含 DTD 和包含 DTD 中的约束条件的文档，也就是正确的文档。但是还有与 XML 标准相符合的第三个层次：由于 DTD 不完整或文档不符合 DTD 中的约束条件，所以该包含 DTD 的文档结构完整但不合法；这是三种类型中最不普遍的情况。

可是，没有必要要求所有的文档都是合法的。有时 XML 文档仅需结构完整就足够了。DTD 在结构完整文档中也占有一席之地（虽然不是必需的，但是对合法的文档来说确实是必需的），并且不进行合法性检查的 XML 处理器可以在 DTD 中获取有用的信息，而不必完全符合 DTD 的要求。在本节中将研究该项内容。

若结构完整但无效的 XML 文档中包含 DTD，则该 DTD 需具有上一章所研究的相同的通用形式。那就是说，开头为文档类型声明，且可包含 ELEMENT、ATTLIST 和 ENTITY 声明。与有效文档的区别在于处理器仅处理其 ENTITY 声明。

### 9.7.1 内部实体

在结构完整的无效文档中使用 DTD 的主要益处在于还可以使用内部通用实体引用，除了五个预定义引用 >、<、&quot;、&apos; 和 &amp; 之外。可按通常的方法简单地声明所需的实体，然后在文档中使用它们。

例如，回顾前面的例子，假如需要实体引用 &ERH; 用于替换字符串 “Elliott Rusty Harlod”（好吧，那就假设我需要实体引用 &ERH; 用于替换字符串 “Elliott Rusty Harlod”），但不想为文档编写一个完整的 DTD。可按清单 9-15 所示，在 DTD 中简单地声明 ERH 实体引用。该文档仅仅是结构完整，但却是不合法的文档；若不追求合法性，该文档完全可以接受。

清单 9-15：DTD 中的 ERH 实体引用产生了结构完整但不合法的文档

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

<!DOCTYPE DOCUMENT [

<!ENTITY ERH "Elliott Rusty Harold">

]>

<DOCUMENT >

<TITLE>&ERH;</TITLE>

<SIGNATURE>

<COPYRIGHT >1999 &ERH;</COPYRIGHT>

<EMAIL>elharo@metalab.unc.edu</EMAIL>

<LAST_MODIFIED>March 10, 1999</LAST_MODIFIED>

</SIGNATURE>

</DOCUMENT>
```

清单 9-15 中的文档类型声明是少见的。除了在定义 ERH 实体引用的之外，只是简单地说明了基本元素为 DOCUMENT。可是文档的结构完整性并不要求文档满足这一小小的约束。例如清单 9-16，显示的是另一个使用了 PAGE 基本元素的文档，但文档类型声明中却说明该基本元素应该是 DOCUMENT。该文档结构依然完整，但是与清单 9-15 的文档一样都是不合法的。

清单 9-16：结构完整，但不合法的文档

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

<!DOCTYPE DOCUMENT [

<!ENTITY ERH "Elliotte Rusty Harold?">

]>

<PAGE>

<TITLE>&ERH;</TITLE>

<SIGNATURE>

<COPYRIGHT >1999 &ERH;</COPYRIGHT >

<EMAIL>elharo@metalab.unc.edu</EMAIL>

<LAST_MODIFIED>March 10, 1999</LAST_MODIFIED>

</SIGNATURE>

</PAGE>
```

这个 DTD 同样也可包含其他的<!ELEMENT>、<!ATTLIST>和<!NOTATION>声明，所有这些声明均被不进行合法性检查的处理器忽略，仅处理<!ENTITY>声明。清单 9-17 中的 DTD 与其本身的内容相矛盾。例如，根据 DTD 定义，ADDRESS 元素本应为空，但实际上该元素包含几个未声明的子元素。另外，要求每个 ADDRESS 元素都具有 OCCUPANT、STREET、CITY 和 ZIP 属性值，但是却无处可寻。基本元素本应为 DOCUMENT，而不是 ADDRESS。DOCUMENT 元素本应包含的 TITLE 和 SIGNATURE 均未在 DTD 中进行声明。本文档结构依然完整，却无半点合法性。

清单 9-17：结构完整却无效的文档

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

<!DOCTYPE DOCUMENT [

<!ENTITY ERH "Elliotte Rusty Harold?">

<!ELEMENT ADDRESS EMPTY>

<!ELEMENT DOCUMENT ( TITLE, ADDRESS+, SIGNATURE)>

<!ATTLIST ADDRESS OCCUPANT CDATA #REQUIRED>

<!ATTLIST ADDRESS DEPARTMENT CDATA #IMPLIED>
```

<!ATLIST ADDRESS COMPANY CDATA #IMPLIED>

<!ATLIST ADDRESS STREET CDATA #REQUIRED>

<!ATLIST ADDRESS CITY CDATA #REQUIRED>

<!ATLIST ADDRESS ZIP CDATA #REQUIRED>

]>

<ADDRESS>

<OCCUPANT>Elliott Rusty Harold</OCCUPANT >

<DEPARTMENT >Computer Science</DEPARTMENT >

<COMPANY>Polytechnic University</COMPANY>

<STREET >5 Metrotech Center</STREET >

<CITY>Brooklyn</CITY>

<STATE>NY</STATE>

<ZIP>11201</ZIP>

</ADDRESS>

### 9.7.2 外部实体

不进行合法性检查的处理器可处理外部实体引用，但这不是必须的。详细的说，例如 Mozilla 使用的开放资源 XML 语法分析器并不处理外部实体引用。包含 IE 5.0 在内的其余大部分处理器却要处理外部实体引用。可是不进行合法性检查的处理器可能仅处理可析实体，不处理包含非 XML 数据（像图像或声音）的外部实体引用。

外部实体对存储样式文本特别有用。例如，HTML 预定义非 ASCII ISO Latin-1 字母的实体引用，这些引用比数字化字符实体引用稍便于记忆。例如，å 预定义为&ring;, þ 预定义为&thorn;, ý 预定义为&yacute;等等。清单 9-18 为定义这些引用的正式 ISO DTD（对注释进行一些轻微的修改，文中巧妙地应用空格，使得文档看起来形式优美整洁）。

清单 9-18: 非 ASCII ISO Latin-1 字符的 DTD

<!--(C) International Organization for Standardization 1986

Permission to copy in any form is granted for use with

conforming SGML systems and applications as defined in

ISO 8879, provided this notice is included in all copies.

-->

<!--Character entity set. Typical invocation:

<!ENTITY % ISOlat1 PUBLIC

"ISO 8879-1986//E I IES Added Latin 1//E //XML->

%ISOlat1;

->

<!-- his version of the entity set can be used with any SGML

document which uses ISO 8859-1 or ISO 10646 as its

document character set. This includes XML documents and

ISO HTML documents.

Version: 1998-10-01

->

<!ENTITY Agrave "&#192; "><!--capital A, grave accent ->

<!ENTITY Aacute "&#193; "><!--capital A, acute accent ->

<!ENTITY Acirc "&#194; "><!--capital A, circumflex accent ->

<!ENTITY Atilde "&#195; "><!--capital A, tilde ->

<!ENTITY Auml "&#196; "><!--capital A, dieresis umlaut ->

<!ENTITY Aring "&#197; "><!--capital A, ring ->

<!ENTITY AElig "&#198; "><!--capital AE diphthong ligature->

<!ENTITY Ccedil "&#199; "><!--capital C, cedilla ->

<!ENTITY Egrave "&#200; "><!--capital E, grave accent ->

<!ENTITY Eacute "&#201; "><!--capital E, acute accent ->

<!ENTITY Ecirc "&#202; "><!--capital E, circumflex accent ->

<!ENTITY Euml "&#203; "><!--capital E, dieresis umlaut ->

<!ENTITY Igrave "&#204; "><!--capital I, grave accent ->

<!ENTITY Iacute "&#205; "><!--capital I, acute accent ->

<!ENTITY Icirc "&#206; "><!--capital I, circumflex accent ->

<!ENTITY Iuml "&#207; "><!--capital I, dieresis umlaut ->

<!ENTITY ETH "&#208;"><!--capital Eth, Icelandic -->

<!ENTITY Ntilde "&#209;"><!--capital N, tilde -->

<!ENTITY Ograve "&#210;"><!--capital O, grave accent -->

<!ENTITY Oacute "&#211;"><!--capital O, acute accent -->

<!ENTITY Ocirc "&#212;"><!--capital O, circumflex accent -->

<!ENTITY Otilde "&#213;"><!--capital O, tilde -->

<!ENTITY Ouml "&#214;"><!--capital O dieresis/umlaut mark-->

<!ENTITY Oslash "&#216;"><!--capital O, slash -->

<!ENTITY Ugrave "&#217;"><!--capital U, grave accent -->

<!ENTITY Uacute "&#218;"><!--capital U, acute accent -->

<!ENTITY Ucirc "&#219;"><!--capital U circumflex accent -->

<!ENTITY Uuml "&#220;"><!--capital U dieresis umlaut -->

<!ENTITY Yacute "&#221;"><!--capital Y, acute accent -->

<!ENTITY THORN "&#222;"><!--capital THORN, Icelandic -->

<!ENTITY szlig "&#223;"><!--small sharp s, (sz ligature) -->

<!ENTITY agrave "&#224;"><!--small a, grave accent -->

<!ENTITY aacute "&#225;"><!--small a, acute accent -->

<!ENTITY acirc "&#226;"><!--small a, circumflex accent -->

<!ENTITY atilde "&#227;"><!--small a, tilde -->

<!ENTITY auml "&#228;"><!--small a dieresis/umlaut mark-->

<!ENTITY aring "&#229;"><!--small a, ring -->

<!ENTITY aelig "&#230;"><!--small ae, diphthong ligature -->

<!ENTITY ccedil "&#231;"><!--small c, cedilla -->

<!ENTITY egrave "&#232;"><!--small e, grave accent -->

<!ENTITY eacute "&#233;"><!--small e, acute accent -->

<!ENTITY ecirc "&#234;"><!--small e, circumflex accent -->

<!ENTITY euml "&#235;"><!--small e, dieresis or umlaut ->

<!ENTITY igrave "&#236;"><!--small i, grave accent ->

<!ENTITY iacute "&#237;"><!--small i, acute accent ->

<!ENTITY icirc "&#238;"><!--small i, circumflex accent ->

<!ENTITY iuml "&#239;"><!--small i, dieresis or umlaut ->

<!ENTITY eth "&#240;"><!--small eth, Icelandic ->

<!ENTITY ntilde "&#241;"><!--small n, tilde ->

<!ENTITY ograve "&#242;"><!--small o, grave accent ->

<!ENTITY oacute "&#243;"><!--small o, acute accent ->

<!ENTITY ocirc "&#244;"><!--small o, circumflex accent ->

<!ENTITY otilde "&#245;"><!--small o, tilde ->

<!ENTITY ouml "&#246;"><!--small o, dieresis or umlaut->

<!ENTITY oslash "&#248;"><!--small o, slash ->

<!ENTITY ugrave "&#249;"><!--small u, grave accent ->

<!ENTITY uacute "&#250;"><!--small u, acute accent ->

<!ENTITY ucirc "&#251;"><!--small u, circumflex accent ->

<!ENTITY uuml "&#252;"><!--small u, dieresis or umlaut ->

<!ENTITY yacute "&#253;"><!--small y, acute accent ->

<!ENTITY thorn "&#254;"><!--small thorn, Icelandic ->

<!ENTITY yuml "&#255;"><!--small y, dieresis or umlaut ->

可简单地应用参数实体引用链接到清单 9-18 所示的实体引用，然后在文档中使用通用实体引用，而不需要把清单 9-18 包含在文档 DTD 的内部子集中。

例如，假设需要以结构完整的 XML 文档将中世纪的 Hlidgebrandslied 文档放入 Web 上，可是原稿为德语书写的，文中使用了非 ASCII 字符 æ、ê、î、ô 和 û。

为使文档具有最大的可移植性，可按 ASCII 字符键入诗文，而把这些字母分别编码为&ecirc;、&icirc;、&ocirc;、&ucirc;和&aelig;的实体引用。即使不需要有效完整的文档，也依然需要一个 DTD，该 DTD 声明使用的各种实体引用。获取所需扩展字符的最简单方法就是简单地引用清单 9-18 中的外部 DTD。清单 9-19 说明了这种情况。

清单 9-19：为使用 ASCII ISO Latin-1 字母而使用实体引用的无效完整文档

```

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

<!DOCTYPE DOCUMENT [

<!ENTITY % ISOlat1

PUBLIC "ISO 8879-1986//E I IES Added Latin 1//E //XML"

"http://www.schema.net/public-text/ISOlat1.pen">

%ISOlat1;

]>

<DOCUMENT>

<TITLE>Das Hildebrandslied, circa 775 C.E. </TITLE>

<LINE>Ik gih&ocirc;rta dhat seggen,</LINE>

<LINE>dhat sih urh&ecirc;ttun &aelig;non muot&icirc;n,</LINE>

<LINE>Hiltibrant enti Hadhubrant untar heriun tu&ecirc;m.

</LINE>

<LINE>sunufatarungo: iro saro rihtun,</LINE>

<COMMENT>I ll spare you the next 61 lines</COMMENT>

</DOCUMENT>

```

文档部分是由使用现场编写的标记的结构完整的 XML 所组成。这些标记未在 DTD 中声明过，也没有必要去维护文档的结构完整性。可是实体引用需要在内部或外部子集的 DTD 中声明。在清单 9-19 中，通过外部参数实体引用%ISOlat1 载入清单 9-18 中声明的实体，也就在外部子集中声明了实体引用。

DTD 也可用于储存通用样式文本，该文本用在整个 Web 站点上的结构完整的 XML 文档，有利于维护 XML 文档的合法性。当仅仅处理结构完整的 XML 文档时，可体现出一定的简便性，这是因为插入到文档中的样式文本与父文档 DTD 的约束条件上不会出现任何匹配问题。

首先，把不带 DTD 的样式插入到一个文件中，如清单 9-20 所示。

清单 9-20: 不带 DTD 的 Signature 样板

```

<?xml version="1.0"?>

<SIGNATURE>

<COPYRIGHT>1999 Elliotte Rusty Harold</COPYRIGH >

<EMAIL>elharo@metalab.unc.edu</EMAIL>

```

</SIGNATURE>

接下来，按清单 9-21 所示编写一个小型的 DTD，该 DTD 为清单 9-20 中的文件定义一实体引用。在这里，假设可在文件 signature.xml 中找到清单 9-20 所示内容，该文件位于 Web 服务器根目录上的 boilerplate 目录中；也假定可在文件 singature.dtd 中找到清单 9-21 所示内容，该文件位于 Web 服务器根目录上的 dtds 目录中。

清单 9-21：定义实体引用的 Signature DTD

```
<!ENTITY SIGNATURE SYSTEM "/boilerplate/signature.xml">
```

现在，就可在文档中引入 signature.dtd，然后使用通用实体引用&SIGNATURE;，就可在文件中嵌入 signature.xml 的内容。清单 9-22 说明了这种用法：

清单 9-22：使用&SIGNATURE;的文件

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

```
<!DOCTYPE DOCUMENT [
```

```
<!ENTITY % SIG SYSTEM "/dtds/signature.dtd">
```

```
%SIG;
```

```
]>
```

```
<DOCUMENT>
```

```
<TITLE>A Very Boring Document</TITLE>
```

```
&SIGNATURE;
```

```
</DOCUMENT>
```

似乎这种间接的做法与真正所需的相比较，多了一个层次。例如清单 9-23 直接在其内部 DTD 子集中定义了&SIGNATURE;，实体引用，且确实有效。但是这种间接做法所增加的层次可保护 Web 站点免于被更改，这是因为无法通过编辑一个文件的方式仅更改所有页面使用的 signature 内容。也可通过编辑一个文件的方式更改所有 Web 页面使用的 signature 的位置。另一方面，清单 9-22 中使用的方法越直接，就越便于在不同的页面上使用不同的 signature。

清单 9-23：使用&SIGNATURE;减少了一层非直接引用的文件

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

```
<!DOCTYPE DOCUMENT [
```

```
<!ENTITY % SIGNATURE SYSTEM "/dtds/signature.dtd">
```

```
]>
```

```
<DOCUMENT>
```

```
<TITLE>A Very Boring Document</TITLE>
```



&SIGNATURE;

</DOCUMENT>

## 9.8 本章小结

从本章中，可了解如何从内部和外部实体开始创建 XML 文档。详细地说，学习了以下内容：

- 实体就是组成文档的物理存储单元。
- 实体内容为：结构完整的 XML 文档、其他形式的文本和二进制数据。
- 内部实体完全在文档内部定义，外部实体可引入通过 URL 定位的不同资源的内容。
- 通用实体引用具有 “&name;” 的形式，通常用于文档的内容中。
- 内部通用实体引用由实体声明中给定的实体值所替换。
- 外部通用实体引用由 URL 定位的数据所替换，该 URL 为实体声明中 SYSTEM 关键词后的内容规定。
- 内部参数实体引用具有 “%name;” 的格式，只在 DTD 中使用。
- 可用外部参数实体引用和不同的 DTD。
- 外部实体引用提供创建大型复杂文档的能力。
- XML 标准一致性的第三层含义：结构完整，但不合法。不合法的原因在于 DTD 不完整或文档不满足 DTD 的约束条件。

当文档使用了属性时，必须在 DTD 中对属性加以声明。下一章讲述如何在 DTD 中声明属性，以及如何将约束条件附加于属性值进行限制。

## 第 10 章 DTDs 中的属性声明

一些 XML 元素具有属性。属性包含应用程序使用的信息。属性仅在程序对元素进行读、写操作时，提供元素的额外信息（如 ID 号等），对于人类读、写元素来说是毫无意义的。在本章中学习各种属性类型和如何在 DTD 中声明属性。

本章内容如下：

- 什么是属性？
- 如何在 DTD 中声明属性
- 如何声明多个属性
- 如何指定属性的缺省值
- 属性类型
- 预定义属性
- 基于属性的棒球比赛统计数据的 DTD

### 10.1 什么是属性？

在第 3 章曾经讨论过开始标记和空标记可包含由等号 “=” 分割开的成对的属性名和属性值。例如：

```
<GREETING LANGUAGE= "English">
```

```
Hello XML!
```

```
<MOVIE SOURCE= "WavingHand.mov" />
```

```
</GREETING>
```

上述例子中，GREETING 元素具有 LANGUAGE 属性，其属性值为 ENGLISH。MOVIE 元素具有 SOURCE 属性，其属性值为 WavingHand.mov。GREETING 元素内容为 Hello XML!。书写内容的语言对内容本身来说是一个有用的信息，可是语言不是内容的一部分。

与此相似，MOVIE 元素内容为保存在 WavingHand.mov 文件中的二进制数据。尽管文件名告诉我们到何处可找到元素内容，但它本身不是元素内容。再次强调，属性包含有关元素内容信息，而不是元素内容本身。

元素可具有多个属性，例如：

```
<RECTANGLE WIDTH= "30" HEIGHT= "45" />
```

```
<SCRIPT LANGUAGE= "javascript" ENCODING= "8859_1" >...</SCRIPT>
```

上例中，SCRIPT 元素属性 LANGUAGE 的值为 javascript，SCRIPT 元素属性 ENCODING 的值为 8859\_1；RECTANGLE 元素属性 WIDTH 的值为 30；RECT 元素属性 HEIGHT 的值为 45。这些属性值均为字符串数据，不是数字型数据。

结束标记不能带属性，下例视为非法：

```
<SCRIPT>...</SCRIPT LANGUAGE= "javascript" ENCODING= "8859_1" >
```

## 10.2 在 DTD 中声明属性

与元素和实体相似，为保持文档的合法性，需要在文档的 DTD 中声明属性。<!ATTLIST>标记用于声明属性，其形式如下：

<!ATTLIST Element\_name Attribute\_name Type Default\_value>

Element\_name 为拥有该属性的元素名。Attribute\_name 为属性名，Type 为表 10-1 列出的 10 种有效属性类型的一种。最常用的属性类型为 CDATA。最后，若未规定属性值，则属性值为 Default\_value。

例如，研究下列元素：

<GREETING LANGUAGE= "Spanish">

Hola!

</GREETING>

在 DTD 中，可按如下格式声明该元素：

<!ELEMENT GREETING (#PCDATA)>

<!ATTLIST GREETING LANGUAGE CDATA "English">

<!ELEMENT>标记简单地说明 greeting 元素包含可析字符数据，这里没什么新内容。<!ATTLIST>标记表明 GREETING 元素拥有 LANGUAGE 属性，其值为 CDATA 类型，本质上与元素内容的#PCDATA 相同。若所看见的 GREETING 标记中没有 LANGUAGE 属性，则 LANGUAGE 属性值为缺省指定的 English。

表 10-1 属性类型

类 型	含 义
CDATA	字符数据不是标记的文本
Enumerated	可能取值的列表，可从中选出正确的值
ID	不能被文档中其他任何 ID 类型属性共享的数字，具有唯一性
IDREF	文档中元素的 ID 类型属性的值
IDREFS	由空格分开的若干个 ID
ENTITY	在 DTD 中声明的实体名
ENTITIES	在 DTD 中声明的若干个实体的名字，彼此间由空格分开
NMTOKEN	XML 名称
NOTATION	在 DTD 中声明的注释名
NMTOKENS	由空格分开的多个 XML 名称

在各自的标记中分别声明各自的属性列表。属性所属元素的名字包含在<!ATTLIST>标记中，如上例中的属性声明仅用于 GREETING 元素。如果其余元素也具有 LANGUAGE 属性，就需要各自独立的<!ATTLIST> 声明。

对大部分声明而言，属性声明在文档中出现的顺序并无严格要求，可位于与其相连的元素声明之前或之后。实际上，甚至可以对同一属性进行多次声明。这时，第一个声明首先执行。

尽管非同寻常，甚至可以为并不存在的标记声明属性。在最初编辑 DTD 时，可以声明一些并不存在的属性，计划在以后再返回这里继续这些工作，为这些属性声明元素。

## 10.3 声明多个属性

元素通常具有多个属性。HTML 的 IMG 元素可有 HEIGHT、WIDTH、ALT、BORDER、ALIGN 和其他几个属性。实际上，大部分 HTML 标记都具有多个属性，XML 标记也是如此。例如，很自然的 RECTANGLE 元素需要 LENGTH 和 WIDTH 属性：

```
<RECTANGLE LENGTH= "70px" WIDTH="85px"/>
```

也可用几个属性声明来声明这些属性，一个属性声明对应一个属性。例如：

```
<!ELEMENT RECTANGLE EMPTY>
```

```
<!ATTLIST RECTANGLE LENGTH CDATA "0px">
```

```
<!ATTLIST RECTANGLE WIDTH CDATA "0px">
```

上例说明，RECTANGLE 元素具有 LENGTH 和 WIDTH 属性，它们的缺省值均为 0px。

可按如下方式，组合两个<!ATTLIST>标记为一个单一声明：

```
<!ATTLIST RECTANGLE LENGTH CDATA "0px"
```

```
WIDTH CDATA "0px">
```

该声明声明了 LENGTH 和 WIDTH 属性，两个属性类型均为 CDATA，缺省值为 0px。若各属性的类型或缺省值不同，也可用这种语法结构进行声明。如下所示：

```
<!ATTLIST RECTANGLE LENGTH CDATA "15px"
```

```
WIDTH CDATA "34pt">
```

从个人角度来说，我不喜欢这种风格。看起来很混乱；且为易于辨认，过于依赖于额外的空格在其中的正确放置（尽管这些空格对标签的实际意义而言并不重要）。可是你一定会遇到其他人书写的、这种风格的 DTD，所以必须掌握这种书写方法。

## 10.4 指定属性的缺省值

若不采用明确指定一个缺省属性值（如 0px）的方式，属性声明可以要求作者提供属性值，或者完全忽略该属性值，甚至总是使用缺省值。这三种类型分别由三个关键词#REQUIRED、#IMPLIED、#FIXED 加以指定。

### 10.4.1 #REQUIRED

有时要选一个恰当的缺省属性值并不容易。例如，在为创建一个用于内部网的 DTD 时，可能要求所有的文档都至少有一个空的<AUTHOR>标记；这些标记通常情况下并不显示，但可用来识别创建文档的作者。标记中拥有 NAME、EMAIL 和 EXTENSION 属性，以便与作者联系。例如：

```
<AUTHOR NAME="Elliotte Rusty Harold"
      EMAIL= elharo@metalab.unc.edu EXTENSION= "3459"/>
```

假设要强制要求在内部网上张贴文档人的表明身份，就不采取为这些属性提供缺省值的方法。然而 XML 无法阻止任何人把作者身份定为“Luke Skywalker”（洛克天行者），但至少可通过使用#REQUIRED 的缺省值方式，要求指定作者身份为某个人。例如：

```
<!ELEMENT AUTHOR EMPTY>

<!ATTLIST AUTHOR NAME CDATA #REQUIRED>

<!ATTLIST AUTHOR EMAIL CDATA #REQUIRED>

<!ATTLIST AUTHOR EXTENSION CDATA #REQUIRED>
```

如果语法分析器遇到一个<AUTHOR/>标记，该标记没有包含这些属性中的一个或几个时，将返回一个错误。

也可使用#REQUIRED 强迫作者提交 IMG 元素的 WIDTH、HEIGHT 和 ALT 属性。例如：

```
<!ELEMENT IMG EMPTY>

<!ATTLIST IMG ALT CDATA #REQUIRED>

<!ATTLIST IMG WIDTH CDATA #REQUIRED>

<!ATTLIST IMG HEIGHT CDATA #REQUIRED>
```

任何试图忽略这些属性的行为（这样的 Web 页面太多了）都将产生一个不合法文档。XML 处理器将注意到这种错误，并且将通知缺少这些属性的作者。

### 10.4.2 #IMPLIED

有时可能找到一个好的属性缺省值，但也不想要求文档作者包含这属性值。例如，假设在内部网上张贴文档的一些人拥有电子邮件地址，但它们没有电话分机号；为此，不想要求它们在<AUTHOR/>标记中包含 EXTENSION（分机号）属性部分。例如：

```
<AUTHOR NAME="Elliotte Rusty Harold"
      EMAIL="elharo@metalab.unc.edu "/>
```

如果依然不想为 EXTENSION（分机号）提供缺省属性值，但是想提供作者引入这种类似属性的能力。在这种情况下，就可使用#IMPLIED 的缺省值。如下所示：

```
<!ELEMENT AUTHOR EMPTY>
```

```
<!ATTLIST AUTHOR NAME CDATA #REQUIRED>
```

```
<!ATTLIST AUTHOR EMAIL CDATA #REQUIRED>
```

```
<!ATTLIST AUTHOR EXTENSION CDATA #IMPLIED>
```

如果 XML 处理器遇到没有 EXTENSION 属性的<AUTHOR/>标记，就不向 XML 应用程序提供有用的属性值。应用程序按收到的通知进行相应的选择。例如，应用程序把元素送入 SQL 数据库中，属性映射为字段，应用程序或许在数据库相应字段中插入空的数据。

### 10.4.3 #FIXED

最后，可能想提供一个不允许作者更改的属性缺省值。例如，希望为在内部网上张贴文档的人员的 AUTHOR 元素指定一个同等的 COMPANY 标识属性。方法如下：

```
<AUTHOR NAME= "Elliott Rusty Harold" COMPANY="TIC"
```

```
EMAIL= "elharo@metalab.unc.edu" EXTENSION="3459"/>
```

可通过指定缺省值为#FIXED，其后跟随实际的缺省值，来要求所有的人员对 COMPANY 属性使用该缺省值。例如：

```
<!ELEMENT AUTHOR EMPTY>
```

```
<!ATTLIST AUTHOR NAME CDATA #REQUIRED>
```

```
<!ATTLIST AUTHOR EMAIL CDATA #REQUIRED>
```

```
<!ATTLIST AUTHOR EXTENSION CDATA #IMPLIED>
```

```
<!ATTLIST AUTHOR COMPANY CDATA #FIXED "TIC">
```

文档作者不需要在它们各自的标记中真正地引用固定的属性。如果它们没有包括固定属性，则使用缺省值；如果包括了固定的属性，无论如何它们使用的属性值必须一致，否则语法分析器将返回一个错误信号。



## 10.5 属性类型

前面的所有例子都具有 CDATA 类型的属性。CDATA 是最通用的类型，但此外还允许使用其他九种属性类型。所有十种类型如下：

- CDATA
- Enumerated(枚举)
- NMTOKEN
- NMTOKENS
- ID
- IDREF
- IDREFS
- ENTITY
- ENTITIES
- NOTATION

上述属性类型中的九种在类型字段中的值为常数，而 Enumerated 是一种特殊的类型，表示属性值必须为一可能取值列表中的一个。下面分别对各类型进行深入的研究。

### 10.5.1 CDATA 属性类型

最通用的属性类型 CDATA，表明属性值为不包括小于号(<)和引号(“) 的任意文本字符串。可通过普通的实体引用(&lt;和&quot;) 的方式或由字符引用 Unicode 值的方式插入小于号和引号字符。原始的和号(&) 不是字符或实际引用开始的和号，必须使用换码符&amp;。

实际上，即使属性值中不可避免的要包含双引号(“) 的情况下，也不可直接使用；替代的方法是用单引号把双引号括起来。如下例所示：

```
<RECTANGLE LENGTH= 7" WIDTH= 8.5" >
```

如果属性值中包含单引号和双引号，而且并不用作界定属性值的用途；它们必须替换为实体引用&apos; (单引号)和&quot; (双引号)。例如：

```
<RECTANGLE LENGTH= 8& apos;7" WIDTH="10 6&quot;;"/>
```

### 10.5.2 Enumerated 属性类型

Enumerated 类型不是 XML 的关键词，而是由竖线分隔的可能的属性值列表。任一值均需为有效的 XML 名称。文档作者可选取列表中的一个成员为属性的值，缺省值必须为列表中的一个值。

例如，假设希望某个元素具有可见和不可见属性。希望该元素具有一个 VISIBLE 属性，其属性值为 TRUE 或 FALSE。如果该元素为简单的 P 元素，那么<!attlist>声明可如下所示：

```
<!ATTLIST P VISIBLE (TRUE | FALSE) "TRUE">
```

上述声明表示 P 元素的 VISIBLE 属性可有可无，若拥有 VISIBLE 属性，则属性值必须为 TRUE 或 FALSE；如果没有 VISIBLE 属性，则假定该值为 TRUE。例如：

```
<P VISIBLE= "FALSE">You can t see me! Nyah! Nyah!</P>
```

```
<P VISIBLE= "TRUE">You can see me.</P>
```

```
<P>You can see me too.</P>
```

就其自身而言，这声明并不是一个提供隐藏文本能力的魔术般的咒语。这种能力依然依靠应用程序去理解不应该显示不可见元素。为决定元素的显示或隐藏，可以通过对元素应用 VISIBLE 属性的样式单规则来进行设置。例如：

```
<xsl:template match= "P[@VISIBLE= FALSE ]" >
```

```
</xsl:template>
```

```
<xsl:template match= "P[@VISIBLE= TRUE ]" >
```

```
<xsl: apply-templates/>
```

```
</xsl:template>
```

### 10.5.3 NMTOKEN 属性类型

NMTOKEN 属性类型限定属性值为有效的 XML 名称。如第 6 章所述，XML 名称必须以字母或下划线开头。名字中后面的字符可以为字母、数字、下划线、连字符和句号。但不可包括空格（下划线通常作为空格的替代品）。从技术上说，名字中可包含冒号（:）但不应该使用冒号，因为冒号被保留为与命名域（namespace）一起使用。

当使用编程语言处理 XML 数据时，证明了 NMTOKEN 的价值。这并不是一种偶然，除了允许使用冒号以外，上述规则与 JAVA，JavaScript 和其他程序语言标识符规则一致。例如，可在元素中使用 NMTOKEN 属性访问特别的 JAVA 类。那么就应用 JAVA 的 API 映射把数据传送到专有类的特有方法中。

当需要从大量名字中选取不是 XML 的规定部分但与 XML 命名要求相符的名字时，就能体现 NMTOKEN 的用途。这些要求的最重要部分就是对空格的限制。例如，NMTOKEN 可以用于下述属性，其值必须映射为 8.3 的 DOS 文件名，另一方面该属性也能用于 UNIX、Macintosh 或 Windows NT 文件名，而这些文件名中通常包含空格。

例如，假如要求<ADDRESS/>标记中的州（state）属性为两个字母缩写；不能用 DTD 强制这些特性的执行，但可应用如下<!ATTLIST>声明防止人们输入类似“New York”或“Puerto Rico”的值：

```
<!ATTLIST ADDRESS STATE NMTOKEN #REQUIRED>
```

无论何种情况，像“California”、“Nevada”和其他一个单词的州名依然为合法值。当然，可以利用具有几十个两个字母的代码的枚举列表的简单方法；但是这种方法将导致巨大的工作量，比人们想象的大得多。举个例子，想一想，如果用两个字母代码代表美国 50 个州、所有的领土和属地、所有的国外的军事基地和加拿大所有的省份会是一个什么样的情况？另一方面，如果曾经在 DTD 文件中的参数实体引用定义了这样的列表，就可重复多次使用这个文件。

### 10.5.4 NMTOKENS 属性类型

NMTOKENS 属性类型几乎就是 NMTOKEN 的复数形式。这种类型的属性可以使如下情况合法——属性由若干 XML 名称字组成，彼此间由空格分隔。通常可为使用 NMTOKEN 属性相同的理由而使用 NMTOKENS 属性，但仅仅在需要多个名字的时候。

例如，如果 state 元素的属性值需要多个两个字母州代码时，就可应用下例所示的方法：

```
<!ATTLIST ADDRESS STATES NMTOKENS #REQUIRED>
```

然后，就可编写如下所示的标记：

```
<ADDRESS STATES="MI NY LA CA">
```

不幸的是，如果应用这种技术，就不能再排除类似“New York”这样的州名，因为州名中每一独立的部分都为一个合格的 NMTOKEN。如下所示：

```
<ADDRESS STATES="MI New York LA CA">
```

### 10.5.5 ID 属性类型

一个 ID 类型的属性标识文档中唯一的元素，编辑工具和其余应用程序通常使用 ID 列举文档中的元素，并不关心元素的实际意义和各元素彼此之间的关系。

一个 ID 类型属性值必须为有效的 XML 名称，该名称以字母开头，由字母数字混排的字符和下划线组成，并且其中不带空格。一个特定的名字不能用作多个标记的 ID 属性。若在一个文档中两次使用同一 ID 将导致语法分析器返回一个错误信息；另外，一个元素不能具有超过一个的 ID 类型的属性。

一般来说，ID 属性的存在只是为了处理数据的程序方便。在许多情况下，除了 ID 属性值外，多个元素可能会是一样的，如果以可以预见的方式来选取 ID 的话，程序就可以列举出文档中所有不同类型的元素或同一类型的不同元素。

ID 类型属性与 #FIXED 类型的属性不兼容。ID 类型属性不能同时具有 #FIXED 类型的属性，因为 #FIXED 类型的属性仅能拥有一个单一的值，而每个 ID 类型属性都具有不同的值。大部分 ID 属性使用 #REQUIRED 值。如清单 10-1 例所示：

清单 10-1: required ID 属性类型

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

<!DOCTYPE DOCUMENT [

<!ELEMENT DOCUMENT (P*)>

<!ELEMENT P (#PCDATA)>

<!ATTLIST P PNUMBER ID #REQUIRED>

]>

<DOCUMENT>

<P PNUMBER="p1">The quick brown fox</P>

<P PNUMBER="p2">The quick brown fox</P>

</DOCUMENT>
```

### 10.5.6 IDREF 属性类型

IDREF 类型的属性值为文档中另一个元素的 ID。例如，清单 10-2 表明 IDREF 和 ID 属性用于子元素和父元素之间的连结。

清单 10-2: family.xml

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

<!DOCTYPE DOCUMENT [
```

```

<!ELEMENT DOCUMENT (PERSON*)>

<!ELEMENT PERSON (#PCDATA)>

<!ATTLIST PERSON PNUMBER ID #REQUIRED>

<!ATTLIST PERSON FATHER IDREF #IMPLIED>

<!ATTLIST PERSON MOTHER IDREF #IMPLIED>

]>

<DOCUMENT>

<PERSON PNUMBER= "a1" >Susan</PERSON>

<PERSON PNUMBER= "a2" >Jack</PERSON>

<PERSON PNUMBER= "a3" MOTHER= "a1" FATHER= "a2" >Chelsea</PERSON>

<PERSON PNUMBER= "a4" MOTHER= "a1" FATHER= "a2" >David</PERSON>

</DOCUMENT>

```

当在文档结构树上并不冲突的两个元素之间需要建立连结时，通常使用这种并不普遍但很重要的类型。在清单 10-2 中，每个子元素都有 FATHER 和 MOTHER 属性给出，这两个属性包含的是对应的 ID 属性。

在清单 10-2 中，无法简洁地使用 IDREF 建立父元素到子元素之间的链接，这是因为每个父元素都具有不确定的子元素数目。解决的方法就是，可以把所有同一父元素的子元素组成 FAMILY 元素，然后链接到 FAMILY 上。即使使用这种方法，当面对半同属（共享唯一一个父元素）的元素时，也不好。简而言之，IDREF 适用于多对一的关系，不适合一对多的关系。

### 10.5.7 ENTITY 属性类型

ENTITY 类型的属性提供把外部二进制数据和外部不可析实体链接到文档中的能力。ENTITY 属性值为 DTD 中声明的不可析通用实体名，该实体名链接到外部实际数据。

经典的 ENTITY 属性的例子就是图像。图像由另一 URL 处可用的二进制数据组成。假如 XML 浏览器支持 ENTITY 类型属性，在 DTD 中按如下方式声明，就可在 XML 文档中包括一幅枷癍?/p>

```

<!ELEMENT IMAGE EMPTY>

<!ATTLIST IMAGE SOURCE ENTITY #REQUIRED>

<!ENTITY LOGO SYSTEM "logo.gif">

```

然后在期望图像出现在文档中的位置处，就可插入如下的 IMAG 标记：

```
<IMAGE SOURCE="LOGO"/>
```

所有 XML 浏览器自动识别的过程并不是在变魔术，这仅仅是一种简单的技术，浏览器和其余应用程序可能采用也可能不采用这种技术在文档中嵌入非 XML 数据。



这种技术在第 11 章 “嵌入非 XML 数据” 中有更深入的探讨。

### 10.5.8 ENTITIES 属性类型

ENTITIES 属性类型几乎就是 ENTITY 的复数形式。若干由空格分隔的不可析实体名组成 ENTITIES 类型属性的值。每一实体名指向一个外部非 XML 数据资源。这种类型属性的用途之一为：使不同图片之间的切换变得光滑平顺，如下例所示：

```
<!ELEMENT SLIDESHOW EMPTY>
```

```
<!ATTLIST SLIDESHOW SOURCES ENTITIES #REQUIRED>
```

```
<!ENTITY PIC1 SYSTEM "cat.gif">
```

```
<!ENTITY PIC2 SYSTEM "dog.gif">
```

```
<!ENTITY PIC3 SYSTEM "cow.gif">
```

然后在文档中希望显示图片的位置上插入如下标记：

```
<SLIDESHOW SOURCES="PIC1 PIC2 PIC3">
```

再一次声明，这不是所有（或任意）XML 浏览器可以自动识别的通用格式；仅仅是某些浏览器和其余的应用程序可能采用也可能不采用的在文档中嵌入非 XML 数据的方法而已。

### 10.5.9 NOTATION 属性类型

NOTATION 属性类型指定属性值为 DTD 中声明的记号名。这一属性的缺省值也必须为 DTD 中声明的记号名。在下一章中介绍记号的详细内容。简单地说，记号可标识非 XML 数据的格式；例如为不可析实体指定一帮助程序。



第 11 章 “嵌入非 XML 格式数据” 讲述了这方面的内容。

例如，SOUND 元素的 PLAYER 属性具有 NOTATION 类型和缺省值 MP，从而标识非 XML 数据的格式，记号 MP 表示一个特殊类型的声音文件：

```
<!ATTLIST SOUND PLAYER NOTATION (MP) #REQUIRED>
```

```
<!NOTATION MP SYSTEM "mplay32.exe">
```

也可提供不同记号的选择。这样做的用法之一是为不同的平台指定不同的帮助应用程序。浏览器可从中选取一可用的值。这种情况下，NOTATION 关键词后紧跟一对圆括号，括号内包含由竖直线分隔的、许可的记号名列表。例如：

```
<!NOTATION MP SYSTEM "mplay32.exe">
```

```
<!NOTATION ST SYSTEM "soundtool">
```

```
<!NOTATION SM SYSTEM "Sound Machine">
```

```
<!ATTLIST SOUND PLAYER NOTATION (MP | SM | ST) #REQUIRED>
```

这表明 SOUND 元素的 PLAYER 属性值可设置为 MP、ST 或 SM。下一章对此再作进一步的研究。



乍看上去，这种处理方法与其余列表属性（如 ENTITIES 和 NMTOKENS）处理方法好像不一致；但其实这两种方法截然不同。ENTITIES 和 NMTOKENS 在文档的实际元素中具有一个属性列表，但在 DTD 的属性声明中仅有一个值。可是文档中实际元素的 NOTATION 属性值仅有一个。可取值的列表位于 DTD

的属性声明中。

## 10.6 预定义属性

在某种程度上说，可以在 XML 中预定义两个属性。必须在 DTD 中为将应用的每一元素声明这两个属性，但是仅仅可以为原定目标而使用这些声明的属性。通过在属性名前加 `xml:` 来标识这类属性。

这两类属性分别为 `xml: space` 和 `xml: lang`。`xml: space` 属性描述如何对待元素中的空格；`xml: lang` 属性描述书写元素的语言（以及可选的方言和国别）。

### 10.6.1 `xml: space`

在 HTML 中，空格并不重要。尽管一个空格和没有空格之间的差别是非常重要的，但是一个空格和两个空格、一个空格和一个回车符、一个空格三个回车符和 12 个制表符之间的差别并不重要。对某些文本来说，空格非常重要，如计算机源代码、某些大型机数据库报告或者 e. e. cumming 的诗文，可使用 PRE 元素指定等宽的字体和保留空格。

不过，XML 的缺省方式为保留空格。XML 处理器毫不改变地传送全部空格字符给应用程序。应用程序通常忽略额外的空格。可是 XML 处理器可通知应用程序某些特定的元素包含需保留的、意义重大的空格。作者可使用 `xml: space` 属性，为应用程序说明这些元素。

如果元素包含重要的空格，DTD 将为 `xml: space` 属性提供一个 `<!ATTLIST>` 标记。这个属性具有枚举类型，其值为 `default` 和 `preserve`。如清单 10-3 所示。

清单 10-3：用 XML 编码的具有重要空格的 Java 源代码

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

<!DOCTYPE PROGRAM [

<!ELEMENT PROGRAM (#PCDATA)>

<!ATTLIST PROGRAM xml:space (default|preserve) preserve >

]>

<PROGRAM xml:space="preserve">public class AsciiTable {

public static void main (String[] args) {

for (int i = 0; i < 128; i++) {

System.out.println(i + " " + (char) i);

}

}

}

}

</PROGRAM>
```

不管 `xml: space` 值为 `default` 或 `preserve`，全部空格都传送给应用程序。可是，若值为 `default`，应用程序按正常方式处理额外的空格；若值为 `preserve`，则按有特殊意义的态度对待这些额外的空格。



空格的特殊意义在某种角度来说依赖于数据的最终目标。例如，Java 源代码中的额外空格仅与源代码的编辑器有关，但与编译器无关。

已定义 `xml:space` 属性元素的子元素，除非子元素定义了与原值相矛盾的 `xml:space` 属性，否则将表现出与其父元素相似的行为（保留或不保留空格）。

### 10.6.2 `xml:lang`

`xml:lang` 属性标识书写元素内容的语言。属性值可以为 CDATA、NMTOKEN 或者枚举列表类型。理想的情况下，每一个属性值均为原始 ISO-639 标准定义的两个字母语言代码。代码列表可在下述 Web 地址处找到：

<http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt>。

例如，研究下面两个例子，其中的句子取自 Petronius 的 *Satiricon*，分别用拉丁文（Latin）和英文（English）书写。一个 SENTENCE 标记包装了这两个句子，但是第一个句子标记使用的 `xml:lang` 属性值为“Latin”，而第二个句子中其值为“English”。

拉丁文（Latin）：

```
<SENTENCE xml:lang="la">
```

```
Veniebamus in forum deficiente now die, in quo notavimus frequentiam rerum venalium, non quidem pretiosarum  
sed tamen quarum fidem male mbulantem obscuritas temporis facillime tegeret.
```

```
</SENTENCE>
```

英文（English）：

```
<SENTENCE xml:lang="en">
```

```
We have come to the marketplace now when the day is failing, where we have seen many things for s le, not for  
the valuable goods but rather that the darkness of the time may most easily conceal their shoddiness.
```

```
</SENTENCE>
```

说英语的读者很容易区分哪个为原稿，哪个为译本；计算机可根据 `xml:lang` 属性提供的线索进行区分。这个差别可确定拼写检查器决定如何检查特定的元素和决定使用哪个词典。搜索引擎可以检查这些语言属性，然后确定如何检索页面内容和返回基于用户优先选择的匹配结果。

#### 语言种类过多，而代码不足

XML 在使用语言代码问题上依然有点落后。原始的 ISO-639 标准语言代码由两个不分大小写的 ASCII 字母字符构成；这标准允许的代码数目不能超过 676 种不同的代码（26\*26），可是现在地球上使用的语言远远不止 676 种（即使不包括类似 Etruscan 这种已经消亡的语言）。实际上，合理的代码还少于 676 种，因为语言的缩写必须与语言的名字具有一定的关系。

ISO-639 标准的第二部分使用三个字母的代码，可处理地球上使用的所有语言。可是 XML 标准规范要求使用两个字母代码。



语言（language）属性应用在元素及其所有的子元素上，除非其中某个子元素声明了不同的语言。前面的 SENTENCE 元素可按如下方式进行声明：

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

<!ATTLIST SENTENCE xml:lang NMTOKEN "en">
```

假如没有适当的 ISO 代码可利用，也可利用在 IANA 注册的代码之一，虽然现在 IANA 仅增加了四种附加代码（如表 10-2 所示）。最新的列表可在下面的地址处找到：

<http://www.isi.edu/iana/assignments/language/tags>。

表 10-2 IANA 语言代码	
代 码	语 言
no-bok	Norwegian "Book language"（挪威的书面语言）
no-nyn	Norwegian "New Norwegian"（新挪威语言）
i-navajo	Navajo（印第安语）
i-mingo	Mingo

例如：

```
<P xml:lang="no-nyn">
```

如果需要使用的语言代码（或许是 Klingon）既不包含在 ISO 代码中也不包含在 IANA 代码中，就可定义新的语言代码。这些“x-codes”必须以字符串 x-或者 X-开始，标识为用户自定义、私人使用的代码。例如：

```
<P xml:lang="x-klingon">
```

xml: lang 属性值可包含附加的子代码部分，用连字符“-”把子代码与主要的语言代码区分开。最常见的情况是第一个子代码为 ISO-3166 规定的两个字母的国家代码。最新的国家代码列表可在下面的地址中找到：

<http://www.isi.edu/in-notes/iana/assignment/country-codes>。

例如：

```
<P xml:lang="en-US">Put the body in the trunk of the car.</P>

<P xml:lang="en-GB">Put the body in the boot of the car.</P>
```

如果第一个子代码不是 ISO 规定的两个字母国家代码，就必须是设置 IANA 注册的语言的字符集子代码，如 csDEC MCS、roman8、mac、cp037 或 ebcdic-cp-ca。当前使用的代码列表可以在下述地址中找到：

<ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets>。

示例如下：

```
<P xml:lang= "en-mac">
```

最终的结果可能是第一个子代码，另一个以 x-或 X-开头的 x-code。例如：

```
<P xml:lang= "en-x-tic">
```

根据惯例，语言代码写为小写格式，国家代码为大写格式。可是这仅仅是一个惯例。这是 XML 少数对大小写敏感部分中的一个，因为它继承了 ISO 对大小写不敏感的特性。

与 DTD 中使用的其他所有属性相同，为保持文档的合法性。必须明确地声明 xml:lang 属性，必需直接用于它所施加的元素（对于指定 xml:lang 属性的元素的子元素是间接施用）。

或许不希望 xml:lang 取任意值。其允许值也应为合法的 XML 名称字，所以通常赋予属性 NMTOKEN 类型。这种类型可限制属性值为合法的 XML 名称字。例如：

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

```
<!ATTLIST P xml:lang NMTOKEN #IMPLIED "en">
```

另外，如果仅允许很少的几个语言或方言，就可以应用枚举类型。例如，下述 DTD 说明 P 元素可以为 English 或 Latin。

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

```
<!ATTLIST P xml:lang (en | la) "en">
```

也可以使用 CDATA 类型，但是没有什么理由要这样做。使用 NMTOKEN 或者枚举类型有助于发现某些潜在的错误。

## 10.7 基于属性的棒球统计数据的 DTD

在第 5 章中，为 1998 Major League Season 创建了一个结构完整的 XML 文档，文档中应用属性来保存赛季（SEASON）的年份（YEAR）、联盟名（NAME）、分部名、球队名、球队比赛的城市名（CITY）和每个球员的详细统计资料。下面的清单 10-4 是清单 5-1 的一个缩略版本，完整的 XML 文档中包括两个联盟、六个分部、六个球队和两个球员的数据，以便于去理解元素的位置所在和元素具有什么属性。

清单 10-4：完整的 XML 文档

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

<SEASON YEAR= "1998" >

  <LEAGUE NAME= "Nation l League" >

    <DIVISION NAME="East">

      <TEAM CITY=" Atlant NAME=" Braves">

        <PLAYER GIVEN_NAME=" Marty SURNAME=" Malloy"

          POSITION=" Second Base" GAMES=" 11" GAMES_STARTED=" 8"

          AT_BATS=" 28" RUNS=" 3" HITS=" 5" DOUBLES=" 1"

          TRIPLES=" 0" HOME_RUNS=" 1" RBI=" 1" STEALS=" 0"

          CAUGHT_STEALING=" 0" SACRIFICE_HITS=" 0 "

          SACRIFICE_FLIES=" 0" ERRORS=" 0" WALKS=" 2"

          STRUCK_OUT=" 2" HIT_BY_PITCH=" 0" />

        <PLAYER GIVEN_NAME=" Tom" SURNAME=" Glavine"

          POSITION=" Starting Pitcher" GAMES=" 33"

          GAMES_STARTED=" 33" WINS=" 20" LOSSES=" 6" SAVES=" 0"

          COMPLETE_GAMES=" 4 SHUTOUTS=" 3 ERA=" 2.47"

          INNINGS=" 229.1" HOME_RUNS_AGAINST=" 13"

          RUNS_AGAINST=" 67" EARNED_RUNS=" 63" HIT_BATTER=" 2"

          WILD_PITCHES=" 3" BALK=" 0" WALKED_BATTER=" 74"

          STRUCK_OUT_BATTER=" 157" />

      </TEAM>

    </DIVISION>
```

```

<DIVISION NAME=" Central" >

<TEAM CITY=" Chicago NAME=" Cubs" >

</TEAM>

</DIVISION>

<DIVISION NAME=" West >

<TEAM CITY=" San Francisco" NAME=" Giants" >

</TEAM>

</DIVISION>

</LEAGUE>

<LEAGUE NAME=" American League" >

<DIVISION NAME=" East" >

<TEAM CITY=" New York NAME=" Yankees" >

</TEAM>

</DIVISION>

<DIVISION NAME=" Central" >

<TEAM CITY=" Minnesota" NAME=" Twins" >

</TEAM>

</DIVISION>

<DIVISION NAME=" West" >

<TEAM CITY=" Oakland" NAME=" Athletics" >

</TEAM>

</DIVISION>

</LEAGUE>

</SEASON>

```

为了此文档的合法性和结构的完整性，就需要提供 DTD。该 DTD 中必须声明清单 10-4 中使用的所有元素和属性。元素的声明与旧版类似，只是由于大部分信息被转移到属性中的缘故，显得更为简短：

```

<!ELEMENT SEASON (LEAGUE, LEAGUE)>

```

<!ELEMENT LEAGUE (DIVISION, DIVISION, DIVISION)>

<!ELEMENT DIVISION (TEAM+)>

<!ELEMENT TEAM (PLAYER\*)>

<!ELEMENT PLAYER EMPTY>

### 10.7.1 在 DTD 中声明 SEASON 的属性

SEASON 拥有单一的属性 YEAR。尽管有些语义上的限定，规定什么是，而什么不是一个年份（1998 是年份，March 31 就不是）；DTD 不进行这种限定。因此，最好的方法就是声明 YEAR 属性具有最通用的属性类型 CDATA；另外，希望每个赛季都具有年份值（year），就可使 YEAR 属性为 REQUIRED 类型。

<!ATTLIST SEASON YEAR CDATA #REQUIRED>

尽管确实无法限制作者输入 YEAR 属性文本的格式，但是至少可以提供一个记号，表明可以接受何种格式的文本。例如，规定年份（year）需要四位数的格式就不失为一个好主意。

<!ATTLIST SEASON YEAR CDATA #REQUIRED> <!--e. g. 1998 -->

<!--DO NOT USE TWO DIGIT YEARS like 98, 99, 00!! -->

### 10.7.2 在 DTD 中声明 DIVISION 和 LEAGUE 属性

接下来考虑 DIVISION 和 LEAGUE 元素。它们都具有单一的 NAME 属性。此外自然具有 CDATA 和 REQUIRED 属性类型。因为两个不同的元素具有两个相互独立的 NAME 属性，所以需要两个独立的<!ATTLIST>声明。

<!ATTLIST LEAGUE NAME CDATA #REQUIRED>

<!ATTLIST DIVISION NAME CDATA #REQUIRED>

在这里添加注释可以有助于表明作者期望的格式；例如是否包括单词 *League* 和 *Division* 作为名字的一部分。

<!ATTLIST LEAGUE NAME CDATA #REQUIRED>

<!--e. g. "Nation 1 League" -->

<!ATTLIST DIVISION NAME CDATA #REQUIRED>

<!--e. g. "East" -->

### 10.7.3 在 DTD 中声明 TEAM 属性

TEAM 元素具有 NAME 和 CITY 属性，两个属性均为 CDATA 和 REQUIRED 类型。

<!ATTLIST TEAM NAME CDATA #REQUIRED>

<!ATTLIST TEAM CITY CDATA #REQUIRED>

添加注释有助于建立某些并不明显的东西，例如，在一些情况下，CITY 属性实际上可以是某个州名：

```
<!ATTLIST TEAM CITY CDATA #REQUIRED>
```

```
<!--e.g. "San Diego" as in "San Diego Padres"
```

```
or "Texas" as in "Texas Rangers" -->
```

换一种方式，可以在一个简单的<!ATTLIST>中声明这两个属性。

```
<!ATTLIST TEAM NAME CDATA #REQUIRED
```

```
CITY CDATA #REQUIRED>
```

#### 10.7.4 在 DTD 中声明 PLAYER 的属性

PLAYER 元素可以说是包含了大部分属性类型。首先是 GIVEN\_NAME 和 SURNAME 两个属性，均为简单的 CDATA、REQUIRED 类型。

```
<!ATTLIST PLAYER GIVEN_NAME CDATA #REQUIRED>
```

```
<!ATTLIST PLAYER SURNAME CDATA #REQUIRED>
```

下一个 PLAYER 属性是 POSITION。因为棒球场上球员的位置是一个相当基本的数据，在这里可使用枚举属性类型。可是“First Base”、“Second Base”、“Third Base”、“Starting Pitcher”和“Relief Pitcher”都包含有空格，因此它们都不是有效的 XML 名称字。因此仅能使用 CDATA 属性类型。实际情况没有任何理由为 POSITION 选定缺省值，所以该属性应为 REQUIRED 类型。

```
<!ATTLIST PLAYER POSITION CDATA #REQUIRED>
```

接下来的是各种各样的统计数据：GAMES、GAMES\_STARTED、AT\_BAT、RUNS、HITS、WINS、LOSSES、SAVES、SHUTOUTS 等等。上述每一个属性本应为数字类型，但是因为 XML 不提供数字类型机制，所以把它们简单地声明为 CDATA 类型。因为不是每一个球员都具有如上的每一个统计数据值，所以把上述各属性声明为 IMPLIED 类型，而不是 REQUIRED 类型。

```
<!ATTLIST PLAYER GAMES CDATA #IMPLIED>
```

```
<!ATTLIST PLAYER GAMES_STARTED CDATA #IMPLIED>
```

```
<!--Batting Statistics -->
```

```
<!ATTLIST PLAYER AT_BATS CDATA #IMPLIED>
```

```
<!ATTLIST PLAYER RUNS CDATA #IMPLIED>
```

```
<!ATTLIST PLAYER HITS CDATA #IMPLIED>
```

```
<!ATTLIST PLAYER DOUBLES CDATA #IMPLIED>
```

```
<!ATTLIST PLAYER TRIPLES CDATA #IMPLIED>
```

```
<!ATTLIST PLAYER HOME_RUNS CDATA #IMPLIED>
```

```
<!ATTLIST PLAYER RBI CDATA #IMPLIED>
```

<!ATTLIST PLAYER STEALS CDATA #IMPLIED>

<!ATTLIST PLAYER CAUGHT\_STEALING CDATA #IMPLIED>

<!ATTLIST PLAYER SACRIFICE\_HITS CDATA #IMPLIED>

<!ATTLIST PLAYER SACRIFICE\_FLIES CDATA #IMPLIED>

<!ATTLIST PLAYER ERRORS CDATA #IMPLIED>

<!ATTLIST PLAYER WALKS CDATA #IMPLIED>

<!ATTLIST PLAYER STRUCK\_OUT CDATA #IMPLIED>

<!ATTLIST PLAYER HIT\_BY\_PITCH CDATA #IMPLIED>

<!--Pitching Statistics -->

<!ATTLIST PLAYER WINS CDATA #IMPLIED>

<!ATTLIST PLAYER LOSSES CDATA #IMPLIED>

<!ATTLIST PLAYER SAVES CDATA #IMPLIED>

<!ATTLIST PLAYER COMPLETE\_GAMES CDATA #IMPLIED>

<!ATTLIST PLAYER SHUTOUTS CDATA #IMPLIED>

<!ATTLIST PLAYER ERA CDATA #IMPLIED>

<!ATTLIST PLAYER INNINGS CDATA #IMPLIED>

<!ATTLIST PLAYER HOME\_RUNS\_AGAINST CDATA #IMPLIED>

<!ATTLIST PLAYER RUNS\_AGAINST CDATA #IMPLIED>

<!ATTLIST PLAYER EARNED\_RUNS CDATA #IMPLIED>

<!ATTLIST PLAYER HIT\_BATTER CDATA #IMPLIED>

<!ATTLIST PLAYER WILD\_PITCHES CDATA #IMPLIED>

<!ATTLIST PLAYER BALK CDATA #IMPLIED>

<!ATTLIST PLAYER WALKED\_BATTER CDATA #IMPLIED>

<!ATTLIST PLAYER STRUCK\_OUT\_BATTER CDATA #IMPLIED>

只要你愿意，可以把 PLAYER 可能使用的所有属性组合为一个外形庞大的<!ATTLIST>声明：

<!ATTLIST PLAYER

GIVEN\_NAME CDATA #REQUIRED

SURNAME CDATA #REQUIRED

POSITION CDATA #REQUIRED

GAMES CDATA #IMPLIED

GAMES\_STARTED CDATA #IMPLIED

AT\_BATS CDATA #IMPLIED

RUNS CDATA #IMPLIED

HITS CDATA #IMPLIED

DOUBLES CDATA #IMPLIED

TRIPLES CDATA #IMPLIED

HOME\_RUNS CDATA #IMPLIED

RBI CDATA #IMPLIED

STEALS CDATA #IMPLIED

CAUGHT\_STEALING CDATA #IMPLIED

SACRIFICE\_HITS CDATA #IMPLIED

SACRIFICE\_FLIES CDATA #IMPLIED

ERRORS CDATA #IMPLIED

WALKS CDATA #IMPLIED

STRUCK\_OUT CDATA #IMPLIED

HIT\_BY\_PITCH CDATA #IMPLIED

WINS CDATA #IMPLIED

LOSSES CDATA #IMPLIED

SAVES CDATA #IMPLIED

COMPLETE\_GAMES CDATA #IMPLIED

SHUTOUTS CDATA #IMPLIED

ERA CDATA #IMPLIED



INNINGS CDATA #IMPLIED

HOME\_RUNS\_AGAINST CDATA #IMPLIED

RUNS\_AGAINST CDATA #IMPLIED

EARNED\_RUNS CDATA #IMPLIED

HIT\_BATTER CDATA #IMPLIED

WILD\_PITCHES CDATA #IMPLIED

BALK CDATA #IMPLIED

WALKED\_BATTER CDATA #IMPLIED

STRUCK\_OUT\_BATTER CDATA #IMPLIED>

这种方法不会有什么好处，应用这种方法就无法在个别属性后加上简单的注释。

### 10.7.5 棒球比赛统计数据示例的完整 DTD

清单 10-5 显示的是棒球比赛统计数据基本属性的完整 DTD。

清单 10-5：利用属性包含大部分信息的棒球比赛统计数据的完整 DTD

<!ELEMENT SEASON (LEAGUE, LEAGUE)>

<!ELEMENT LEAGUE (DIVISION, DIVISION, DIVISION)>

<!ELEMENT DIVISION (TEAM+)>

<!ELEMENT TEAM (PLAYER\*)>

<!ELEMENT PLAYER EMPTY>

<!ATTLIST SEASON YEAR CDATA #REQUIRED>

<!ATTLIST LEAGUE NAME CDATA #REQUIRED>

<!ATTLIST DIVISION NAME CDATA #REQUIRED>

<!ATTLIST TEAM NAME CDATA #REQUIRED

CITY CDATA #REQUIRED>

<!ATTLIST PLAYER GIVEN\_NAME CDATA #REQUIRED>

<!ATTLIST PLAYER SURNAME CDATA #REQUIRED>

<!ATTLIST PLAYER POSITION CDATA #REQUIRED>

<!ATTLIST PLAYER GAMES CDATA #REQUIRED>

<!ATTLIST PLAYER GAMES\_STARTED CDATA #REQUIRED>

<!--Batting Statistics -->

<!ATTLIST PLAYER AT\_BATS CDATA #IMPLIED>

<!ATTLIST PLAYER RUNS CDATA #IMPLIED>

<!ATTLIST PLAYER HITS CDATA #IMPLIED>

<!ATTLIST PLAYER DOUBLES CDATA #IMPLIED>

<!ATTLIST PLAYER TRIPLES CDATA #IMPLIED>

<!ATTLIST PLAYER HOME\_RUNS CDATA #IMPLIED>

<!ATTLIST PLAYER RBI CDATA #IMPLIED>

<!ATTLIST PLAYER STEALS CDATA #IMPLIED>

<!ATTLIST PLAYER CAUGHT\_STEALING CDATA #IMPLIED>

<!ATTLIST PLAYER SACRIFICE\_HITS CDATA #IMPLIED>

<!ATTLIST PLAYER SACRIFICE\_FLIES CDATA #IMPLIED>

<!ATTLIST PLAYER ERRORS CDATA #IMPLIED>

<!ATTLIST PLAYER WALKS CDATA #IMPLIED>

<!ATTLIST PLAYER STRUCK\_OUT CDATA #IMPLIED>

<!ATTLIST PLAYER HIT\_BY\_PITCH CDATA #IMPLIED>

<!--Pitching Statistics -->

<!ATTLIST PLAYER WINS CDATA #IMPLIED>

<!ATTLIST PLAYER LOSSES CDATA #IMPLIED>

<!ATTLIST PLAYER SAVES CDATA #IMPLIED>

<!ATTLIST PLAYER COMPLETE\_GAMES CDATA #IMPLIED>

<!ATTLIST PLAYER SHUTOUTS CDATA #IMPLIED>

<!ATTLIST PLAYER ERA CDATA #IMPLIED>

<!ATTLIST PLAYER INNINGS CDATA #IMPLIED>

<!--ATTLIST PLAYER HOME\_RUNS\_AGAINST CDATA #IMPLIED-->

<!--ATTLIST PLAYER RUNS\_AGAINST CDATA #IMPLIED-->

<!--ATTLIST PLAYER EARNED\_RUNS CDATA #IMPLIED-->

<!--ATTLIST PLAYER HIT\_BATTER CDATA #IMPLIED-->

<!--ATTLIST PLAYER WILD\_PITCHES CDATA #IMPLIED-->

<!--ATTLIST PLAYER BALK CDATA #IMPLIED-->

<!--ATTLIST PLAYER WALKED\_BATTER CDATA #IMPLIED-->

<!--ATTLIST PLAYER STRUCK\_OUT\_BATTER CDATA #IMPLIED-->

使用如下的序进程把清单 10-5 的内容引入到清单 10-4 中，当然我们假设清单 10-5 保存为一个文件名为“baseballattribtes.dtd”的文件中。

<?xml version= "1.0" standalone=" yes "?>

<!--DOCTYPE SEASON SYSTEM "baseball attributes.dtd" -->

## 10.8 本章小结

本章中学习了如何在 DTD 中声明元素的属性。具体地说，学习了下述内容：

- 在 DTD 的 `<!ATTLIST>` 标记中声明属性。
- 一个 `<!ATTLIST>` 标记可以声明一个元素任意数目的属性。
- 属性通常具有缺省值，但是可以通过使用关键词 `#REQUIRED`、`#IMPLIED` 或 `#FIXED` 改变这种状态。
- 在 DTD 中，可以声明十种属性类型：`CDATA`、枚举类型、`NMTOKEN`、`NMTOKENS`、`ID`、`IDREF`、`IDREFS`、`ENTITY`、`ENTITIES` 和 `NOTATION`。
- 预定义的 `xml:space` 属性确定元素中的空格是否有意义。
- 预定义的 `xml:lang` 属性规定书写元素内容的语言。

下一章将学习到如何利用记号、处理指令和不可析实体在 XML 文档中嵌入非 XML 数据。

## 第 11 章 嵌入非 XML 数据

不是世界上的所有数据都为 XML 格式。实际上，可以大胆地说世界上积累下来的数据大部分都不是 XML 格式。大量数据按无格式文本、HTML 和微软的 Word 格式保存，这里只举出三种常用的非 XML 格式。在理论上说，如果有兴趣且财力允许的情况下，至少这些数据的大部分可以重写为 XML 格式，但也不是所有的数据都可以。例如把图像编码为 XML 格式就将导致处理效率极端低下。

XML 提供三种结构：记号、不可析外部实体和处理指令，通常用于处理非 XML 格式数据。记号描述非 XML 格式数据；不可析外部实体提供与非 XML 格式数据实际位置的链接；处理指令给出如何观看这些数据的信息。

本章叙述的具体内容尚有许多争议。尽管我所说的每一个事情都符合 XML 1.0 规范，但是不是所有人都同意上述观点。肯定可以写出一个 XML 文档，文档中没有使用注解和外部对象，仅有一些简单的处理说明。可以先跳过本章内容，在后面发现有必要了解这方面内容时，再返回到这一章。

本章的主要内容如下：

- 记号
- 不可析外部实体
- 处理指令
- DTD 中的条件部分

### 11.1 记号

在 XML 文档中使用非 XML 格式数据将会遇到的第一个问题是识别数据格式，并通知 XML 应用程序如何读出和显示这些非 XML 格式数据。例如，企图在屏幕上画出 MP3 声音文件就是不合适的。

在一个有限的范围内只利用一套固定的用于特定种类的外部实体的标记，就可在单一应用程序中解决外部非 XML 数据的读取和显示问题。例如，如果全部图片数据都通过 IMAGE 元素嵌入，全部声音数据通过 AUDIO 元素嵌入；那么开发一个知道如何处理这两个元素的浏览器并不是一件很难的事。实际上这正是 HTML 采用的方法，可是这样的方法不允许文档作者为了能更加清楚地描述它们所需的内容，而创建新的标记；例如 PERSON 元素碰巧就有一个 PHOTO 属性，该属性指向那人的 JPEG 格式图片。

再者，没有一个应用程序可理解所有可能的文件格式。大多数 Web 浏览器可以管理和读出 GIF、JPEG、PNG 图像文件，或许还包括一些其他格式的图像文件；但是它们在 EPS、TIFF、FITS 文件前都束手无策，对于是几百种普遍和特殊的图像格式就更加力不从心了。图 11-1 的对话框或许再熟悉不过了。

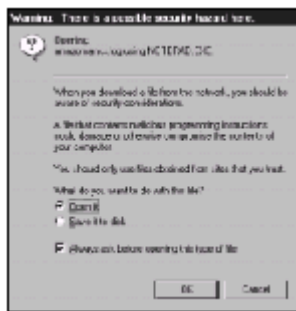


图 11-1 当 Netscape Navigator 无法识别一种文件类型时将发生的事情

理想的情况是希望文档会通知应用程序外部实体的格式，因此不必去依赖应用程序来识别文件类型，或是靠具有魔力的数字或者是并不可靠的文件扩展名。此外，如果应用程序自身无法处理这种格式的图像，也可以为应用程序提供一些关于什么程序可用来显示图像的线索。

记号提供了部分解决这个问题的方法（尽管不能获得很好的支持）。记号描述非 XML 数据的格式。在 DTD 中，NOTATION 声明规定特殊的数据类型。DTD 在与元素、属性和实体同一层次上声明记号。每个记号声明都包含一个名字和一个外部标识符，语法结构如下：

```
<!NOTATION name SYSTEM "externalID">
```

*name* 为文档中使用的特殊格式的标识符；*externalID* 就是用来标识记号的有意义的字符串。例如，实体 GIF 图像的记号可以使用 MIME 类型：

```
<!NOTATION GIF SYSTEM "Image/gif">
```

也可以使用 PUBLIC 代替 SYSTEM 标识符，这样做就必须提供 public ID 和 URL。例如：

```
<!NOTATION GIF PUBLIC
```

```
"-//IETF// NONSGML Media Type image/gif//EN"
```

```
"http://www.isi.edu/in-notes/iana/assignments/media-types/image/gif">
```

对于如何准确地作出外部标识，还存在激烈的争论。像图像/gif、文本/HTML 之类的 MIME 类型是一种可能性；另一个建议是选择 URL，或者其他的标准文档定位方式——像 <http://www.w3.org/TR/REC-html140/>。第三个选择是使用正式的国际标准——如表示日期和时间的 ISO 8601 标准。某些情况下，可能 ISBN 或者国会图书馆为文献文档编目的方法更为适用。此外还有其余许多选择。

选取何种方式，取决于对文档生命期的期望值。例如，如果选择不普遍的格式，就不能依赖每个月都会改变的 URL 方式；如果希望文档在 100 年内都具有活跃的生命力，那么就该考虑使用在 100 年中都具有意义的标识符，而不是使用仅具有 10 年生命力的技术。

也可以使用记号来描述插入到文档中数据。例如，研究下面的 DATA 元素：

```
<DATE>05-07-06</DATE>
```

05-07-06 到底表示哪一天？是公元 1906 年 5 月 7 日还是公元 1906 年 7 月 5 日？答案取决于是按美国格式还是欧洲格式理解这个日期。甚至也可能是 2006 年 5 月 7 日或者 2006 年 7 月 5 日。或者是公元 6 年 5 月 7 日，是西方鼎盛时期的罗马帝国的秋天和中国的汉朝。也有可能这个日期根本不是公元纪年，而是犹太历、穆斯林历法或者中国的农历。没有更多的信息，就无法确定其真正的意义。

为了避免这样混淆不清的情况，ISO 8601 标准为表示日期规定了一个精确的方法。应用这种方法，在 XML 中，公元 2006 年 7 月 5 日写为 20060705，或者是如下格式：

```
<DATE>20060705</DATE>
```

这种格式不是与每个人的想法都相同，对所有人都具有同等程度的迷惑性，不偏向任何一种文化（实际上仍然偏向西方传统日历）。

在 DTD 中声明记号，并且用记号属性描述嵌入 XML 文档中的非 XML 数据的格式。接着再研究日期的例子，清单 11-1 定义两种日期记号：ISO 8601 和美国惯用格式。然后将 NOTATION 类型必需的 FORMAT 属性添加到每一个 DATE 元素中，用来描述特定元素的结构。

清单 11-1: ISO 8601 和美国惯用格式 DATE 元素

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

<!DOCTYPE SCHEDULE [

<!NOTATION ISODATE SYSTEM

"http://www.iso.ch/cate/d15903.html">

<!NOTATION USDATE SYSTEM

"http://es.rice.edu/ES/humsoc/Galileo/Things/gregorian_calendar

.html">

<!ELEMENT SCHEDULE (APPOINTMENT*)>

<!ELEMENT APPOINTMENT (NOTE, DATE, TIME?)>

<!ELEMENT NOTE (#PCDATA)>

<!ELEMENT DATE (#PCDATA)>

<!ELEMENT TIME (#PCDATA)>

<!ATTLIST DATE FORMAT NOTATION (ISODATE | USDATE) #IMPLIED>

]>

<SCHEDULE>

<APPOINTMENT>

<NOTE>Deliver presents</NOTE>

<DATE FORMAT="ISODATE">12-25-1999</DATE>

</APPOINTMENT>

<APPOINTMENT>

<NOTE>Party like it s 1999</NOTE>

<DATE FORMAT="ISODATE">19991231</DATE>

</APPOINTMENT>
```

</SCHEDULE>

记号不能强制作者使用记号描述的格式。因此需要提供除 XML 基本方法以外的几种语言方案——但是在相信作者能正确描述日期的简单应用场合，记号方法是有效的。



## 11.2 不可析外部实体

对所有的数据，特别是非文本数据，XML 格式都不是理想的格式。例如，可以按下面所示的方式，把位图图像的每一个像素存为一个 XML 元素：

```
<PIXEL X="32"Y="28" COLOR="FF5E32"/>
```

可是，这肯定不是一个好主意。任何微小的错误都会导致气球图像文件的比例严重失衡。XML 现在和将来都永远不可能让 XML 文档具有访问数据的能力，因此无法把所有数据按 XML 编码。

一个典型的 Web 页面可以引用 GIF 和 JPEG 图像、JAVA 小程序、ActiveX 控件、各种类型的声音等等。在 XML 中，因为 XML 处理器不会去尝试理解非 XML 格式的数据块，所以把这些数据块称为不可析实体。至多 XML 处理器通知应用程序存在这样的实体，并且为应用程序提供实体名和实体可能包含的内容（可是这并不是必须执行的动作）。

HTML 页面通过各种定制的标记嵌入非 HTML 实体。图片由具有 SRC 属性的<IMG>标记引用，SRC 属性提供图像文件的 URL；JAVA 程序由具有 CLASS 和 CODEBASE 属性的<APPLET>标记包括，CLASS 和 CODEBASE 属性指向 JAVA 程序保存的文件和目录；<OBJECT>标记来嵌入 CODEBASE 属性引用，可从中找到目标数据的 URI。每一种情况下，特定的预定义标记表示一种特定的内容。预定义属性包含其内容的 URL。

XML 应用程序可以但不是必须这样运作，实际上，除了特意为保持与落后的 HTML 之间的兼容性之外，大部分 XML 应用程序都不这样做。相反，XML 应用程序使用不可析外部实体引用这些内容。不可析外部实体提供与非 XML 数据的实际位置的链接。接着文档中特定的元素利用其 ENTITY 属性与实体相连。

### 11.2.1 声明不可析实体

回忆第 9 章的内容，外部实体的声明看起来如下面的形式：

```
<!ENTITY SIG SYSTEM "http://metalab.unc.edu/xml/signature.xml">
```

可是，仅在 URL 指明的外部实体恰好为完整的 XML 文档的时候，才能接受这种格式。如果外部实体不是 XML，则不得不使用 NDATA 关键字指定实体类型。例如，为了用 LOGO 名字连接 GIF 格式文件 logo.gif，就需在 DTD 中放置如下的 ENTITY 声明：

```
<!ENTITY LOGO SYSTEM "logo.gif" NDATA GIF>
```

声明中的最终名字必须是 DTD 中声明的记号名，如本例中的 GIF。记号将 GIF 类的名称与某种类型的外部标识符联系起来，外部标识符标识某种格式。如 MIME 类型、ISO 标准式或者是格式规格的 URL。例如，GIF 的记号类似下面的形式：

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

通常，作为习惯的表示方法，可以使用绝对或相对的 URL 指向外部实体。例如：

```
<!ENTITY LOGO SYSTEM "http://metalab.unc.edu/xml/logo.gif"
```

```
NDATA GIF>
```

```
<!ENTITY LOGO SYSTEM "/xml/logo.gif" NDATA GIF>
```

```
<!ENTITY LOGO SYSTEM "../logo.gif" NDATA GIF>
```

### 11.2.2 嵌入不可析实体

不能与用通用实体引用嵌入可析实体一样，在文档中的任意位置简单地嵌入不可析实体。例如，清单 11-2 就是一个不合法的是 XML 文档，因为 LOGO 是不可析实体。如果这里的 LOGO 是可析实体，本例就为有效的 XML 文档。

清单 11-2：试图用通用实体引用嵌入不可析实体的无效 XML 文档

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

<!DOCTYPE DOCUMENT [

<!ELEMENT DOCUMENT ANY>

<!ENTITY LOGO SYSTEM "http://metalab.unc.edu/xml/logo.gif"

NDATA GIF>

<!NOTATION GIF SYSTEM "image/gif"

]>

<DOCUMENT>

&LOGO;

</DOCUMENT>
```

为了嵌入不可析实体，不采用如&LOGO; 通用实体引用的方法；而是声明一个元素，把该元素作为不可析实体的占位符（例如 IMAGE）。然后声明 IMAGE 元素属性 SOURCE 为 ENTITY 类型，SOURCE 属性仅提供不可析实体名。如清单 11-3 所示。

清单 11-3：正确嵌入不可析实体的合法的 XML 文档

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

<!DOCTYPE DOCUMENT [

<!ELEMENT DOCUMENT ANY>

<!ENTITY LOGO SYSTEM "http://metalab.unc.edu/xml/logo.gif"

NDATA GIF>

<!NOTATION GIF SYSTEM "image/gif"

<!ELEMENT IMAGE EMPTY>

<!ATTLIST IMAGE SOURCE ENTITY #REQUIRED>

]>

<DOCUMENT>

<IMAGE SOURCE="LOGO"/>
```

</DOCUMENT>

等到应用程序读取 XML 文档时，就可认出这个不可析实体且显示出来。应用程序也可能不显示不可析实体（当用户使图像载入失效时，Web 浏览器将选择不显示图像）。

这些例子表明：空元素就像是为不可析实体准备的容器，可是这不是必须采用的方法。例如，假设有一个基于 XML 的公司 ID 系统，就是保安人员使用的查寻进入建筑物的人的系统；PERSON 元素拥有 NAME、PHONE、OFFICE、EMPLOYEE\_ID 子类和 PHOTO ENTITY 属性，如清单 11-4 所示。

清单 11-4：具有 PHOTO ENTITY 属性的非空 PERSON 元素

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

<!DOCTYPE PERSON [

<!ELEMENT PERSON (NAME, EMPLOYEE_ID, PHONE, OFFICE)>

<!ELEMENT NAME (#PCDATA)>

<!ELEMENT EMPLOYEE_ID (#PCDATA)>

<!ELEMENT PHONE (#PCDATA)>

<!ELEMENT OFFICE (#PCDATA)>

<!NOTATION JPEG SYSTEM "image/jpg"

<!ENTITY ROGER SYSTEM "rogers.jpg" NDATA JPEG>

<!ATTLIST PERSON PHOTO ENTITY #REQUIRED>

]>

<PERSON PHOTO="ROGER">

<NAME>Jim Rogers</ NAME>

<EMPLOYEE_ID>4534</EMPLOYEE_ID>

<PHONE>X396</PHONE>

<OFFICE>RH 415A</OFFICE>

</PERSON>
```

这个例子看起来有点做作。实际上，使一个带有 SOURCE 属性的空 PHOTO 元素的成为 PERSON 元素的子元素，而不是 PERSON 元素的属性。再者，或许可以把这个 DTD 分割为内部和外部的子集。如清单 11-5 所示，外部子集声明元素、记号和属性。这些都是可以被不同的文档共享的部分。但是，实体从一个文档到另一个文档会发生变化，因此最好把实体放在如清单 11-6 显示的文档的内部 DTD 子集中。

清单 11-5：外部 DTD 子集 person.dtd

<!ELEMENT PERSON ( NAME, EMPLOYEE\_ID, PHONE, OFFICE, PHOTO)>

<!ELEMENT NAME (#PCDATA)>

<!ELEMENT EMPLOYEE\_ID (#PCDATA)>

<!ELEMENT PHONE (#PCDATA)>

<!ELEMENT OFFICE (#PCDATA)>

<!ELEMENT PHOTO EMPTY>

<!NOTATION JPEG SYSTEM "image/jpeg">

<!ATTLIST PHOTO SOURCE ENTITY #REQUIRED>

清单 11-6: 内含非空 PERSON 元素和一个内部 DTD 子集的文档

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

<!DOCTYPE PERSON [

<!ENTITY % PERSON \_DTD SYSTEM "person.dtd">

%PERSON\_DTD;

<!ENTITY ROGER SYSTEM "rogers.jpg" NDATA JPEG>

]>

<PERSON>

<NAME>Jim Rogers</NAME>

<EMPLOYEE\_ID>4534</EMPLOYEE\_ID>

<PHONE>X396</PHONE>

<OFFICE>RH 415A</OFFICE>

<PHOTO SOURCE="ROGER"/>

</PERSON>

### 11.2.3 嵌入多个不可析实体

在某些特殊场合下, 一个单一的属性甚至一个标识号, 可能需要引用不止一个的不可析实体。就可以声明占位符元素的属性为 ENTITIES 类型。ENTITIES 属性值由空格分隔的多个不可析实体名组成, 每个实体名都指向一个外部非 XML 格式数据资源, 并且必须在 DTD 中声明所有实体。例如, 可以用这种方法编写一个以幻灯放映元素来切换不同的图片, DTD 需要如下形式的声明:

<!ELEMENT SLIDESHOW EMPTY>

<!ATTLIST SLIDESHOW SOURCES ENTITIES #REQUIRED>

<!NOTATION JPEG SYSTEM "image/jpeg"

<!ENTITY HARM SYSTEM "charm.jpg" NDATA JPEG>

<!ENTITY MARJORIE SYSTEM "marjorie.jpg" NDATA JPEG>

<!ENTITY POSSUM SYSTEM "possum.jpg" NDATA JPEG>

<!ENTITY BLUE SYSTEM "blue.jpg" NDATA JPEG>

然后，在文档中需要幻灯放映出现的位置上，就可插入如下标记：

<SLIDESHOW SOURCES="CHARM MARJORIE POSSUM BLUE">

必须再次强调，这不是一个 XML 处理器（甚至任意处理器）可自动理解的具有魔力的方案，这仅仅是一种技巧，在嵌入文档中的非 XML 数据时，浏览器和其余应用程序可能采用也可能不采用的技术。

## 11.3 处理指令

指令经常过多地应用于支持 HTML 的私有范围，如服务端嵌入、浏览器定制脚本语言、数据库模板和其余许多超出 HTML 标准范围的项目。出于这些目的而使用注释的好处是：其余系统可以简单地忽略它们无法理解的外来数据。这种方法的不利之处在于：剥离了注释的文档可能不再是原来的文档了，并且仅仅作为文档的注释会被误解为这些私有范围的输入数据。为了避免滥用注释，XML 提供了处理指令的方法，作为在文件中嵌入信息的明确机制，用于私有的应用程序而不是 XML 语法分析器或浏览器。其余用途包括，处理指令可以提供关于如何查看不可析外部实体的附加信息。

处理指令就是位于<? 和?>标记之间的一行文本。处理指令中的文本只需要如下句法结构，以 XML 名开头，其后紧跟空格，空格后为数据。XML 名可以是应用程序的实际名字（如 latex），或者是在 DTD 中指向应用程序的记号名（例如 LATEX），在 DTD 中的 LATEX 声明具有如下形式：

```
<!NOTATION LATEX SYSTEM "/usr/local/bin/latex">
```

甚至这个名字可以是可被应用程序识别的其他名字。对于使用处理指令的应用程序来说，各个细节部分是非常明确的。确实，大部分依赖处理指令的应用程序在处理指令的内容上利用更多的结构。例如，研究如下在 IBM 的 Bean Markup Language 中使用的处理指令：

```
<?bmlpi register demos.calculator.EventSourceText2Int?>
```

使用处理指令的应用程序名字为 bmlpi。赋予应用程序的数据为字符串 register demos.calculator.EventSourceText2Int，这些数据将包含全部合格的 Java 类名。这就告诉名为 bmlpi 的应用程序使用 Java 类 demos.calculator.EventSourceText2Int，将操作事件转换为整数。如果 bmlpi 在读取文档时遇到这个处理指令，将载入类 demos.calculator.EventSourceText2Int，从此往后利用该类元素将事件转化为整数。

如果这听起来很明确也很详细的话，那是因为它们原来就是如此。处理指令不是文档的通用结构部分，它们为特定的应用程序提供额外的明确的信息，而不是为所有读取该文档的应用程序提供信息。如果其余一些应用程序在读取文档时遇到这个说明，它们将简单地跳过这些说明。

处理指令除了不能位于标记或者 CDATA 字段之内，可以放在 XML 文档中的任意部位。它们可以位于序进程、DTD、元素内容中，甚至可在文档结束标记之后。因为处理指令不是元素，所以不会影响文档的树型结构。没有必要打开或者关闭处理指令，也没有必要考虑它们在其他元素中的嵌套问题。处理指令不是标记，不会对元素进行限定。

到此我们已经很熟悉了一个处理指令的例子：xml-stylesheet 处理指令把样式单与文档相结合：

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

虽然这些例子中的处理指令位于序进程中，但是处理指令可以在文档的任意位置出现。因为处理指令不是元素，所以没有必要声明为包含它们元素的子类元素。

以字符串 xml 开头的处理指令在 XML 规范中留作特殊的用途。此外，在处理指令中，可以自由使用除文档结束标记符(??)外的任意名字和任意文本字符串。例如，下面的例子就是完全有效的处理指令：

```
<?gcc HelloWorld.c ?>
```

```
<?acrobat document="passport.pdf"?>
```

```
<?Dave remember to replace this one?>
```

请记住 XML 处理器不会对处理说明进行任何处理，仅仅是把他们传送给应用程序。应用程序决定如何处理这些说明。大部分应用程序简单地跳过他们无法理解的处理说明。

有些时候了解不可析实体的类型还是不够的。还需要了解应用程序如何运行和查看实体，以及需要提供给应用程序的参数是什么。 这些信息都可以通过处理指令来提供。因为处理指令所包含的数据没有什么限制，所以在制定说明时就相对容易一些，这些说明是决定记号中列出的外部程序将采取什么行为。

这样的处理指令可以是查看数据块的程序名，也可以是几千字节的配置信息。应用程序和文档的作者当然必须采用同样的方法来确定何种不可析外部实体采取何种处理指令。清单 11-7 显示一个方案，该方案使用一个处理指令和 PDF 记号来通知 Acrobat Reader 关于物理纸张的 PDF 格式，以便 Acrobat Reader 显示 PDF 的内容。

清单 11-7：在 XML 中嵌入 PDF 文档

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

<!DOCTYPE PAPER [

<!NOTATION PDF PUBLIC

"-//IETF//NONSXML Media Type application/pdf//EN"

"http://www.isi.edu/in-notes/iana/assignments/media-types/

application/pdf">

<!ELEMENT PAPER (TITLE, AUTHOR+, JOURNAL, DATE_RECEIVED,

VOLUME, ISSUE, PAGES)>

<!ATTLIST PAPER CONTENTS ENTITY #IMPLIED>

<!ENTITY PRLTA0000081000024005270000001 SYSTEM

"http://ojps.aip.org/journal_cgi/getpdf?KEY=PRLTA0&cvips=PR

LTA0000081000024005270000001"

NDATA PDF>

<!ELEMENT AUTHOR (#PNDATA)>

<!ELEMENT JOURNAL (#PNDATA)>

<!ELEMENT YEAR (#PNDATA)>

<!ELEMENT TITLE (#PNDATA)>

<!ELEMENT DATE_RECEIVED (#PNDATA)>

<!ELEMENT VOLUME (#PNDATA)>
```

<!ELEMENT ISSUE (#PNDATA)>

<!ELEMENT PAGES (#PNDATA)>

]>

<?PDF acroread?>

<PAPER CONTENTS="PRLTA0000081000024005270000001">

<TITLE>Do Naked Singularities Generically Occur in  
Generalized Theories of Gravity?</TITLE>

<AUTHOR>Kengo Maeda</AUTHOR>

<AUTHOR>Takashi Torii</AUTHOR>

<AUTHOR>Makoto Narita</AUTHOR>

<JOURNAL>Physical Review Letters</JOURNAL>

<DATE\_RECEIVED>19 August 1998</DATE\_RECEIVED>

<VOLUME>81</VOLUME>

<ISSUE>24</ISSUE>

<PAGES>5270-5273</PAGES>

</PAPER>

任何时候都该记住不是所有的处理器程序都会以你希望的方式去对待这个例子。实际上，大部分处理器都不会。可是，从让一个应用程序支持 PDF 文件和其余非 XML 媒体类型的角度来说，这也是一个值得考虑的方法。



## 11.4 DTD 的条件部分

在创建 DTD 和文档的时候，或许需要文档中没有反映 DTD 的部分作一些注释。除了直接使用注释，也可以把 DTD 中的特定声明组放置在 IGNORE 指令中的方式，从而忽略该声明组。句法结构如下：

```
<![ IGNORE
```

```
declarations that are ignored
```

```
]]>
```

像通常一样，空格不会对句法结构产生实质性的影响，但是必须保证开始符（<![IGNORE）和结束符（]]>）占单独的一行，以便阅读。

可以忽略任意声明或一组声明——元素、实体、属性甚至包括其他的 IGNORE 块，但是必须忽略整个声明。IGNORE 的构造必须完整包含从 DTD 中移走的全部声明。不能仅忽略声明的某个局部（例如在不可析实体声明中的 NDATA GIF）。

也可以指定包括声明的某个特定部分，也就是说不忽略的部分。INCLUDE 指示的句法结构与 IGNORE 相似，但是关键词不同：

```
<![ INCLUDE
```

```
declarations that are included
```

```
]]>
```

当 INCLUDE 位于 IGNORE 之内的时候，INCLUDE 和声明都被忽略。当 IGNORE 位于 INCLUDE 内，位于 IGNORE 之内的声明依然被忽略。换一种说法就是 INCLUDE 不会覆盖 IGNORE。

上述给出的情形中，或许会为 INCLUDE 的存在表示奇怪。简单地移走 INCLUDE 块，仅留下它们的内容，没有任何 DTD 会发生改变。INCLUDE 好像完全是多余的。可是，对于无法单独使用 IGNORE 的参数实体引用的情形中，同时应用 IGNORE 和 INCLUDE 不失为一个灵巧的方法。首先定义一个如下的参数实体引用：

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

将元素包裹在下列结构中，就可将其忽略：

```
<![ %fulldtd;
```

```
declarations
```

```
]]>
```

%fulldtd；参数实体引用求出的值为 IGNORE，因此声明被忽略。现在，假设对一个单词作出修改，把 fulldtd 从 IGNORE 改为 INCLUDE，如下所示：

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

所有的 IGNORE 块立即转换为 INCLUDE 块。实际上，就像是一系列开关，可以打开或者关闭声明块。

在本例中，仅使用了一个开关——fulltdt。可以在 DTD 中的多个 IGNORE/INCLUDE 块中使用这种开关。也可以拥有许多可根据不同条件选择开或关的不同 IGNORE/INCLUDE 块。

当设计其余 DTD 内含的 DTD 时，这种能力特别有用。通过改变参数实体开关值，最后的 DTD 可以改变嵌入的 DTD 行为。

## 11.5 本章小结

在本章中，学习了如何通过记号、不可析外部实体和处理指令的方法，把非 XML 数据与 XML 文档相结合。具体地说，学习了下述概念：

- 记号描述非 XML 数据的格式。
- 不可析外部实体为容纳非 XML 文本和数据的储存单元。
- 使用 ENTITY 和 ENTITIES 属性可在文档中包括不可析外部实体。
- 处理指令包含不作任何改变从处理器传送到最终应用程序的说明。S
- INCLUDE 和 IGNORE 块在文档进行语法分析的时候，分别指定是否处理其中包括的 DTD 中的声明。

在本书的后面几个部分可以看到更多拥有 DTD 的文档的例子。但是关于 DTD 的最基本的句法结构和用法，在本章中已经讨论完毕。在本书的第三部分，我们开始讨论 XML 的样式语言，从下一章的级联样式单（第一级）开始。

## 第三部分 样式语言

本部分包括以下各章：

第 12 章——级联样式单级别 1

第 13 章——级联样式单级别 2

第 14 章——XSL 变换

第 15 章——XSL 格式化对象

### 第 12 章 级联样式单级别 1

对于诸如粗体和 Helvetica 样式应用于特定的 XML 元素来说，CSS 是一种非常简单、易懂的语言。任何常用的字处理软件都具有 CSS 支持的大多数样式。例如，可选择字体、字体的粗细、字号、背景颜色、各种元素的间距、元素周围的边框等等。但是，所有的样式信息并不在文档之内存储，而是放置在一种称之为样式单(style sheet)的独立文档中。仅仅改变样式单就可以以多种不同方式格式化一个 XML 文档。不同的样式单可用于不同的目的——打印、Web、展示和其他用途——所需要的只是适用于指定媒体的样式，并且无需改变文档中的任何内容。

本章的主要内容如下：

- 什么是 CSS?
- 如何将样式单与文档关联
- 怎样选择元素
- 继承父字体的大小
- 级联过程
- 在 CSS 样式单中使用注释
- CSS 单位
- 块、内联和列表项元素
- 字体属性
- 颜色属性
- 背景属性
- 文本属性
- 框属性

#### 12.1 什么是 CSS?

级联样式单(Cascading Style Sheets，以下简称 CSS)是 1996 年作为把有关样式属性信息如字体和边框加到 HTML 文档中的标准方法而提出来的。但是，CSS 与 XML 结合的确比与 HTML 结合得更好，因为 HTML 承担着 CSS 标志和 HTML 标志之间向后兼容的任务。例如，要正确地支持 CSS 的 nowrap 属性就要求废除 HTML 中非标准的但又是经常使用的 NOWRAP 元素。由于 XML 元素没有任何预定义的格式规定，所以不会限制何种 CSS 样式只能用于何种元素。

一个 CSS 样式单就是一组规则(rule)。每个规则给出此规则所适用的元素的名称，以及此规则要应用于那些元素的样式。例如，考察清单 12-1，它是一首诗的 CSS 样式单。此样式单有五个规则。每个规则有一个选择符——规则所应用的元素的名称——和一组适用于此元素实例的属性。第一个规则说明 POEM 元素应以块的形式(Display: block)显示其内容。第二个规则说明 TITLE 元素应以 16 磅(font-size: 16pt)、粗体(font-weight: bold)将其内容显示在块中(Display: block)。第三个规则说明 POET 元素应通过自身显示在块中(Display: block)，并且与紧随其后的下一块相距 10 个像素

(margin-bottom: 10px)。第四个规则与第三个相同，所不同的只是前者应用于 STANZA 元素。最后，第五个规则只简单地说明 VERSE 元素也是显示在自己的块中。

清单 12-1：用于诗作的 CSS 样式单

```
POEM { display: block }
```

```
TITLE { display: block; font-size: 16pt; font-weight: bold }
```

```
POET { display: block; margin-bottom: 10px }
```

```
STANZA { display: block; margin-bottom: 10px }
```

```
VERSE { display: block }
```

在 1998 年，W3C 公布了一个修订的、详述的 CSS 规范，称之为 CSS2(CSS2)。同时，他们又把原来的 CSS 改名为 CSS1(CSS1)。CSS2 几乎是 CSS1 的超集，只有少部分不同，当遇到这些不同处时，我将给出注解。换句话说，CSS2 是在 CSS1 的基础上增添了音频样式单、媒体类型、特性选择符和其他新的功能。因此，本章涉及到的几乎每个例子既适用于 CSS1，也适用于 CSS2。在下一章中，将把 CSS2 看作 CSS1 的扩充来加以介绍。

Netscape Navigator 4.0 和 Internet Explorer 4.0 及 5.0 支持 CSS1 的各部分。遗憾的是，对同一部分往往并非同时支持。Mozilla 5.0 被认为对 CSS1 和大多数 CSS2 提供非完全的支持。Internet Explorer 5.0 比 Internet Explorer 4.0 做得更好，可是它仍失去一些主要部分，特别是有关框模型和伪元素的部分。我将试图指出某种浏览器有特别严重问题的地方。

## 12.2 样式单与文档的链接

要真正地使清单 12-1 中的样式单有意义,就得编写一个使用该样式单的 XML 文档。清单 12-2 是用 XML 标记的来自于 Walt Whitman 的诗集名著 *Leaves of Grass* 中的一首诗。第二行是<?xml-stylesheet?>处理指令,此指令通知 Web 浏览器加载本文档,以便将在文件 poem.css 中找到的样式单应用于本文档。图 12-1 显示了加载到 Mozilla 早期 a 版本中的本文档。

清单 12-2: 用 XML 作出标记的 *Darest Thou Now O Soul*(Walt Whitman 的诗)

```
<?xml version=1.0?>

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

<POEM>

<TITLE>Darest Thou Now O Soul</TITLE>

<POET>Walt Whitman</POET>

<STANZA>

<VERSE>Darest thou now O soul,</VERSE>

<VERSE>Walk out with me toward the unknown region,</VERSE>

<VERSE>Where neither ground is for the feet nor
any path to follow?</VERSE>

</STANZA>

<STANZA>

<VERSE>No map there, nor guide,</VERSE>

<VERSE>Nor voice sounding, nor touch of
human hand,</VERSE>

<VERSE>Nor face with blooming flesh, nor lips,
are in that land.</VERSE>

</STANZA>

<STANZA>

<VERSE>I know it not O soul,</VERSE>

<VERSE>Nor dost thou, all is blank before us,</VERSE>

<VERSE>All waits undream'd of in that region,
```

that inaccessible land.</VERSE>

</STANZA>

<STANZA>

<VERSE>Till when the ties loosen,</VERSE>

<VERSE>All but the ties eternal, Time and Space,</VERSE>

<VERSE>Nor darkness, gravitation, sense,

nor any bounds bounding us.</VERSE>

</STANZA>

<STANZA>

<VERSE>Then we burst forth, we float,</VERSE>

<VERSE>In Time and Space O soul,

prepared for them,</VERSE>

<VERSE>Equal, equipt at last, (O joy! O fruit of all!)

them to fulfil O soul.</VERSE>

</STANZA>

</POEM>

在<?xml-stylesheet?>处理指令中的 type 属性是正在使用的 MIME 类型的样式单。对 CSS 来说, 其值是 text/css, 对 XSL 来说, 则是 text/xsl。

CSS2 将在第 13 章中讨论。而 XSL 将在第 14 和 15 章中讲述。

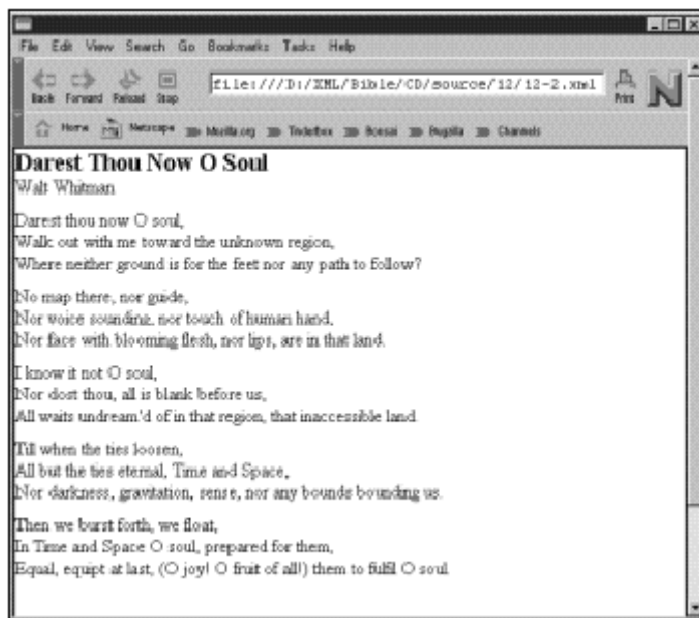


图 12-1 Mozilla 显示的 Darest Thou Now O Soul

`<?xml-stylesheet?>`处理指令中的 href 属性值是一个 URL，通常是相对值，指明在何处可找到样式单。如果样式单没有找到，Web 浏览器很可能使用其缺省的样式单，不过有些浏览器可能会报告一条错误信息。

也可以将同一个样式单用于许多文档，人们通常的确就是这么做的。于是，把样式单放在 Web 服务器上的某个主要位置成了人们的通用方法，所有的文档都会在此位置引用这些样式单；一个便利的位置就是 Web 服务器根目录上的样式目录。

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

甚至还可以使用指向另一个 Web 站点上的样式单的绝对 URL 值，虽然这样做不可避免地会使站点依赖于外部 Web 站点的状态。

```
<?xml-stylesheet type="text/css"
```

```
href="http://metalab.unc.edu/xml/styles/poem.css"?>
```

甚至也可以使用多个`<?xml-stylesheet?>`处理指令，以便应用不同样式单中的规则。例如：

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

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


```
<?xml-stylesheet type="text/css"
```

```
href="http://metalab.unc.edu/xml/styles/poem.css"?>
```

```
<POEM>
```

¼





尽管本书的重点是介绍 XML 的，但 CSS 样式单也与 HTML 文档一起使用。HTML 中的 CSS 与 XML 中的 CSS 之间的主要区别是：

1. 与规则相链接的元素只限于标准的 HTML 元素，如 P、PRE、LI、DIV 和 SPAN。
2. HTML 浏览器不能识别处理指令，所以在 HEAD 元素中，使用 LINK 标志将样式单与 HTML 文档相链接。而且，每个文档样式规则包括在 STYLE 元素中的 HEAD 里面。例如：

```
<LINK REL=STYLESHEET TYPE=^ text/css^ HREF=^ /styles/poem.css^ >
```

```
<STYLE TYPE=^ text/css^
```

```
PRE { color: red }
```

```
</STYLE>
```

3. 由于元素的传统格式化方法，HTML 浏览器不能像 XML 浏览器一样准确无误地体现 CSS 属性，在此方面表格就是众所周知的问题之一。

样式单与 DTD 或多或少地成正交关系。带样式单的文档可能有也可能没有 DTD，而带 DTD 的文档也可能有或没有样式单。但是，DTD 常常充当便于使用的必须为其提供样式规则的元素列表。

在本章以及下面几章中，大多数例子都要使用结构完整、但不合法的文档。不使用 DTD 会使所举的例子更短、相关的部分更加明显。但是实际上，大多数与样式单相链接的文档很可能是合法的带 DTD 的文档。



## 12.3 选择元素

在 CSS 规则中，有个部分是用来指定 CSS 规则适用于哪个元素的，此部分称为选择符(selector)。最普通的选择符就是元素的名称；例如，下面规则中的 TITLE：

```
TITLE { display: block; font-size: 16pt; font-weight: bold }
```

可是，选择符还可指定多个元素、带有特定的 CLASS 或 ID 特性的元素以及与其他元素相关的出现在特定上下文中的元素。

在 CSS1 中，无法做到的一件事就是选择带有特定特性名的元素或除预定义的 CLASS 和 ID 特性之外的值，为此，就得使用 CSS2 或 XSL。

### 12.3.1 成组选择符

如果想把一组属性应用于多个元素，可以用逗号将选择符中的所有元素分开。例如，在清单 12-1 中，POET 和 STANZA 都是被设定为 10 个像素页边距的块显示。于是，可把这两个规则如下列方式组合起来：

```
POET, STANZA { display: block; margin-bottom: 10px }
```

此外，多个规则也可将样式作用于一个特定的元素。所以可以将一些标准的属性组合成带有许多选择符的一个规则中，然后使用更多的指定规则来把定制格式作用于所选元素。例如，在清单 12-1 中，所有的元素都是以块显示的方式列出来的。这样就可以组合成一个规则，而用于 POET、STANZA 和 TITLE 元素的其他格式化放在各自的规则中，于是：

```
POEM, VERSE, TITLE, POET, STANZA { display: block }
```

```
POET, STANZA { margin-bottom: 10px }
```

```
TITLE { font-size: 16pt; font-weight: bold }
```

### 12.3.2 伪元素

CSS1 支持两种伪元素(pseudo-element)，它们指出文档中通常不能作为独立的元素来看待的部分，但常常需要独立样式。通常伪元素是元素的第一行和首字母。

Internet Explorer 5.0 的早期测试版和 Internet Explorer 更早版本都不支持这些伪元素。Mozilla 5.0 早期的测试版的确支持，但仅用于 HTML。

#### 12.3.2.1 强调首字母

要将一个元素的首字母与此元素中其他字母分别格式化的最通用手段就是插入一个下沉的大写字母，如图 12-2 所示。为此，需要编写一条以元素名标识的规则，紧接此元素名后写入:first-letter。例如：

```
CHAPTER:first-letter { font-size: 300%;
```

```
float: left; vertical-align: text-top }
```

正像在图 12-2 中看到的那样，尽管词首的大写字母的大小可以调整，但下沉大写字母(float:left; vertical-align:text-top)的“下沉”部分在 Mozilla 5.0 的早期测试版和 Internet Explorer 5.0 中似乎仍行不通。



图 12-2 在首字母伪元素上的下沉大写字母以及在首行伪元素上使用的小型大写字母

### 12.3.2.2 强调首行

一个元素的第一行也常常被格式化为与此元素文本的其他部分不同。例如，可用小型的大写字母进行打印，而不是以通常的主体文本，如图 12-2 所示。可以将: first-line 选择符加到元素的名称上，以创建只适用于此元素第一行的规则。例如：

```
CHAPTER: first-line { font-variant: small-caps }
```

伪元素到底选择了什么内容依赖于当前窗口的布局。如果窗口较大，因而在第一行中单词也较多，那么，以小型大写字母显示的单词也就越多。如果窗口变小，或字体变大，以致造成文本不同程度的折行，从而使第一行的单词变得较少，那么折行到下一行中的单词就不再以小型大写字母的形式显示了。因此，在文档实际显示出来之后，才能确定 first-line 伪元素包含哪些字母。

### 12.3.3 伪类(pseudo-classe)

有时候，可能想对同一个类型的两个元素设计成不同的样式。例如，有一个段落可能是粗体的，而另一个段落则为正常粗细的字体。要达到此目的，可以把 CLASS 特性加到两个元素之一上，然后为给定的 CLASS 中的元素编写一个规则。

例如，以含有许多 CITATION 元素的一个书目为例。在清单 12-3 中显示了一个示例。现假定要将 Alan Turing 文章中的所有引用着成蓝色，同时又不改变其他的引用。为此，必须将带有指定值(TURING 也行)的 CLASS 属性加到要着色的元素中。

清单 12-3：有三个 CITATION 元素的 XML 书目

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

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

<BIBLIOGRAPHY>

<CITATION CLASS="HOFSTADTER" ID="C1">

<AUTHOR>Hofstadter, Douglas</AUTHOR>
```

"<TITLE>How Might Analogy, the Core of Human Thinking,

Be Understood By Computers?</TITLE>"

<JOURNAL>Scientific American</JOURNAL>,

<MONTH>September</MONTH>

<YEAR>1981</YEAR>

<PAGES>18-30</PAGES>

</CITATION>

<CITATION CLASS="TURING" ID="C2">

<AUTHOR>Turing, Alan M.</AUTHOR>

"<TITLE>On Computable Numbers,

With an Application to the Entscheidungs-problem</TITLE>"

<JOURNAL>

Proceedings of the London Mathematical Society</JOURNAL>,

<SERIES>Series 2</SERIES>,

<VOLUME>42</VOLUME>

(<YEAR>1936</YEAR>) :

<PAGES>230-65</PAGES>.

</CITATION>

<CITATION CLASS="TURING" ID="C3">

<AUTHOR>Turing, Alan M.</AUTHOR>

"<TITLE>Computing Machinery & Intelligence</TITLE>"

<JOURNAL>Mind</JOURNAL>

<VOLUME>59</VOLUME>

(<MONTH>October</MONTH>

<YEAR>1950</YEAR>) :

<PAGES>433-60</PAGES>



</CITATION>

</BIBLIOGRAPHY>

CSS1 中令人讨厌的地方之一是使得混合内容更为必要。清单 12-3 中有很多标点符号，但它们并非真正是内容中的一部分；例如，在 YEAR 元素两边的圆括号，以及 TITLE 元素两边的引号。这些都是表达元素，本应为样式单的一部分。CSS2 允许使用其他文本，如插入到元素之前或之后的标点符号。

清单 12-4 中的样式单使用一个 CLASS 选择符来将 TURING 类中的元素着成蓝色。

IE 5 支持 CLASS 属性，但 Mozilla 的里程碑式的第三版之前的版本不支持此属性。直到 Mozilla 正式发行后，有可能支持 CLASS 属性。

清单 12-4：将 TURING 类中的元素着成蓝色的样式单

```
BIBLIOGRAPHY { display: block }
```

```
CITATION.TURING { color: blue }
```

```
CITATION { display: block }
```

```
JOURNAL { font-style: italic }
```

在一个有效的文档中，CLASS 属性必须声明为已格式化的元素的可能特性。例如，下面是用于清单 12-3 书目的 DTD：

```
<!ELEMENT BIBLIOGRAPHY (CITATION*)>
```

```
<!ATTLIST CITATION CLASS CDATA #IMPLIED>
```

```
<!ATTLIST CITATION ID ID #REQUIRED>
```

```
<!ELEMENT CITATION ANY>
```

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

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

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

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

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

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

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

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



一般地说，我不推荐这种方法。如果可能，你应尝试把附加的元素标记(markup)加到文档中，而不依靠 CLASS 特性。但是，当所选择的信息不太方便地映射到特定的元素中时，CLASS 属性可能是必要的。

#### 12.3.4 由 ID 来选择

有时候，特殊的元素需要特殊的样式。这时，就需要将规则准确地作用于该元素。例如，假定要将一张列表中的一个元素变成粗体，以与此同类进行对照，从而达到强调它的目的。在这种情况下，可编写作用于此元素 ID 特性的规则。以元素名为选择符，紧接在元素名后写上 # 和 ID 特性值。

例如，清单 12-5 是从清单 12-3 的书目中选择了 ID C3 的 CITATION 元素的样式单，此样式单使 ID C3 的元素(并且只有此元素)成为粗体。其他的 CITATION 元素取缺省粗细的字体。所有的 CITATION 元素都是以块的方式显示的，并且所有的 JOURNAL 元素都是斜体的。

清单 12-5: 形成 ID C3 粗体的 CITATION 元素的样式单

```
BIBLIOGRAPHY { display: block }
```

```
CITATION#C3 { font weight: bold }
```

```
CITATION { display: block }
```

```
JOURNAL { font style: italic }
```

IE 5 支持 ID 选择符，Mozilla 则支持用于 HTML 元素的 ID 选择符，但 Mozilla 第三版则不支持 XML 元素的 ID 选择符。等到正式发行时，Mozilla 可能会完全支持 ID 选择符。

#### 12.3.5 上下文的选择符

通常，元素的格式化依赖其父元素。可编写仅作用于在一个命名的父元素内找到的元素的规则。为此，可将父元素名称作为样式化的元素名的前缀。

例如，可以把 PRE 元素内的 CODE 元素显示成 12 磅的 Courier 字体。可是，如果文档的主体文本是以 10 磅 Times 字体写成的，那么与其他主体文本进行内联的 CODE 元素就可能需要以 10 磅 Courier 字体显示。下面的这段规则就可以正确地完成任务：

```
BODY { font family: Times, serif; font-size: 10pt }
```

```
CODE { font-family: Courier, monospaced; font-size: 10pt }
```

```
PRE { font size: 12pt }
```

```
PRE CODE { font-size: 12pt }
```

此段规则说明在 BODY 元素内，字体是 10 磅 Times。但是，在 CODE 元素内，字体变成 Courier，但仍然是 10 磅。而如果 CODE 元素是在 PRE 元素的内部，则字体就变成了 12 磅。

可将此推广到父元素的父元素、父元素的祖元素等等。例如，下列的规则说明在 DATE 元素内的 YEAR 元素里的 NUMBER 元素应该以等宽字体的形式显示。

```
DATE YEAR NUMBER { font-family: Courier, monospaced }
```



实际上，这种特性级是很少需要的。在看来确实需要这样做的情况下，通常可以重写样式单，以便更多地依赖于继承、级联和相对单位，而很少依赖于对格式化的精确规格。

### 12.3.6 STYLE 特性

当手工编创文档时，人们常常想把特定的样式一次性地应用于一个特定的元素而无需为此文档编辑样式单。的确，对无法改变的文档，很可能想忽略某个标准的缺省的样式单。将 STYLE 特性加到元素中就可做到这一点。此特性值是用于此元素的一组以分号隔开的样式属性。例如，下面的 CITATION 使用 STYLE 特性来使自身变为粗体：

```
<CITATION CLASS="TURING" ID="C3" STYLE="font-weight: bold">
```

```
<AUTHOR>Turing, Alan M.</AUTHOR>
```

```
"<TITLE>Computing Machinery & Intelligence</TITLE>"
```

```
<JOURNAL>Mind</JOURNAL>
```

```
<VOLUME>59</VOLUME>
```

```
(<MONTH>October</MONTH>
```

```
<YEAR>1950</YEAR>):
```

```
<PAGES>433-60</PAGES>
```

```
</CITATION>
```

如果在 STYLE 特性中定义的属性与样式单中定义的属性相冲突，那么特性中定义的属性将优先执行。

应尽可能地避免使用 STYLE 特性。如果把样式信息放在独立的样式单中，那么文档将会整洁得多，并且更易于维护。但是，有时 STYLE 特性是如此简单与方便，以致难以将其忽略。

其次，如果在有效的文档中使用这种方法，将需要在 ATTLIST 声明语句中为设计样式的元素声明 STYLE 特性。例如：

```
<!ELEMENT CITATION ANY>
```

```
<!ATTLIST CITATION CLASS CDATA #IMPLIED>
```

```
<!ATTLIST CITATION ID ID #REQUIRED>
```

```
<!ATTLIST CITATION STYLE CDATA #IMPLIED>
```

IE 5 支持 STYLE 特性，Mozilla 支持用于 HTML 元素的 STYLE 特性，但 Mozilla 第三版不支持 XML 元素的 STYLE 特性。等到 Mozilla 正式发行时，它很可能会完全支持 STYLE 特性。

## 12.4 继承性

CSS 不要求为文档中各元素的每个可能的属性明确地定义规则。例如，如果没有指定元素的字号的规则，那么此元素就会继承其父元素的字号。如果没有指定元素颜色的规则，那么此元素也将继承其父元素的颜色。同样对大多数 CSS 属性也是如此。事实上，唯一不被继承的属性就是背景和边框属性。

例如，考虑下列这些规则：

```
p { font-weight: bold;
font-size: 24pt;
font-family: sans-serif }
BOOK { font-style: italic; font-family: serif}
```

现在来考察下面这个 XML 文档片段：

```
<P>
Michael Willhoite's <BOOK>Daddy's Roommate</BOOK> is
the #10 most frequently banned book in the U.S. in the 1990s.
</P>
```


尽管 BOOK 元素没有被明确地赋以 font-weight 或 font-size 属性，但它仍以 24 磅粗体显示，因为它是 P 元素的子元素。它还是斜体的，因为在它自己的规则中被指定过了。BOOK 继承了其父元素 P 的 font-weight 和 font-size。如果此后在文档中 BOOK 元素出现在某个其他元素的上下文中，那么它将继承此元素的 font-weight 和 font-size。

font-family 就有点难以把握，因为 P 和 BOOK 为此属性声明了相互矛盾的值。在 BOOK 元素内部，由 BOOK 声明的 font-family 处于优先位置。在 BOOK 元素外面，使用了 P 的 font-family。所以，Daddy's Roommate 是以 serif(衬线)字体显示的，而 most frequently banned book 则是以 sans serif(非衬线)字体显示的。

通常都想使子元素继承其父元素的格式化。所以，不事先指定任何元素的格式化是很重要的。比如，假设我像下列的那样已声明 BOOK 以 12 磅的字体输出：

```
BOOK { font-style: italic; font-family: serif; font-size: 12pt }
```

那么，此例就会获得如图 12-3 所示的显示结果：



**Michael Willhoite's *Daddy's* Roommate is the  
#10 most frequently banned book in  
the U.S. in the 1990s.**



图 12-3 写成 12 磅字号的 BOOK 元素

可以使用专门的规则改正图中的情况，这个专门规则使用上下文选择符在 P 元素内挑选出 BOOK 元素，但仅仅继承父元素的 font-size 则更容易。

有一种方法可以避免此类问题，而同时又保持对各自元素大小的某种控制，这就是使用相对单位如 em 和 ex，而不使用如磅、十二点活字、英寸、厘米和毫米这样的绝对单位。em 是指在当前字体中字母 *m* 的宽度。ex 是指在当前字体中字母 *x* 的高度。如果字体变大，则以 em 和 ex 测定的所有的尺寸都会随之发生变化。

使用百分比单位对有些属性也会达到相似的结果。例如，下列的规则设置 FOOTNOTE\_NUMBER 元素的字号为其父元素字号的 80%。如果父元素的字体增加或减小，那么 FOOTNOTE\_NUMBER 的字号也会相似地成比例变化。

```
FOOTNOTE_NUMBER { font-size: 80% }
```

将百分比精确地定位于多大，这要根据属性而定。在 vertical-align 属性中，此百分比就是元素本身的行高。但页边距属性中，百分比就是此元素的宽度百分比



## 12.5 级联过程

将一个以上的样式单与一个文档相连接是可能的。例如，一个浏览器可能有一个缺省的样式单，把此缺省的样式单加入到设计者为此页提供的样式单中。在此情况下，有可能有多个规则作用于一个元素，这些规则很可能发生冲突。于是，确定以何种顺序应用这些规则就显得尤其重要。此过程称为级联(cascade)，级联样式单在这个过程中获得自己的名称。

CSS 样式单与 XML 文档相连接有几种方法：

1. 把<?xml-stylesheet?>处理指令包括在 XML 文档中。
2. 使用@import，样式单本身可以导入其他样式单。
3. 用户可以使用浏览器的内部运行机制为文档指定一个样式单。
4. 浏览器为大多数属性提供缺省的样式。

### 12.5.1 @import 指令

样式单中包括@import 指令来加载保存在其他文件中的样式单。可使用绝对或相对的 URL 来标识导入的样式单。例如，

```
@import url(http://www.w3.org/basicstyles.css);
```

```
@import url(/styles/baseball.css);
```

@import 指令必须出现在样式单的开头并在任何规则之前。导入样式单的样式单中的规则总要覆盖被导入的样式单中的规则。被导入的样式单以它们导入的顺序级联排列。禁止循环导入(如，poem.css 导入 stanza.css，而 stanza.css 又要导入 poem.css)。

### 12.5.2 !important 声明

在 CSS1 中，作者规则覆盖读者规则，除非读者规则把!important 声明与属性相连接。例如，下列的规则说明 TITLE 元素应该着成蓝色，即使文档的作者要求是其他颜色。另一方面，只要作者规则不冲突，font-family 应该为 serif。

```
TITLE { color: blue !important font-family: serif }
```

但是，作者规则也可以声明为重要的。在此情况下，作者规则覆盖读者规则。

这是一种很坏的观念。读者总该有权选择自己的浏览方式。不过，要使写出的样式单同时适合于下列几种人，则是不可能做到的：使用彩色显示器和黑白显示器的人们；使用 21 英寸显示器或电视机与使用手持机(PDA)的人们；视觉健康与视觉有缺陷的人们。许多 Web 设计者都指定了太多的样式，只能生成出系统完全无法读取的、与原来不完全一致的网页。幸运的是，CSS2 将这种优先权过程颠倒，以致读者规则决定最终的样式。

### 12.5.3 级联顺序

样式是从适用于一个元素的现有的样式规则中选择出来的。一般地，越专门的规则，优先权越高。例如，考察下面的这个片段：

```
<OEUVRE>
```

```
<PLAY ID=" x02" CLASS=" WILDE" >
```

## The Importance of Being Earnest

</PLAY>

</OUEVRE>

最专门的规则就会优先。于是，通过其 ID 选择 PLAY 元素将优于使用其 CLASS 选择 PLAY 元素。使用其 CLASS 来选择 PLAY 元素的规则将先于包含在 OUEVER 元素中选择 PLAY 元素的规则。当然，如果不使用任何规则，将选择通用的 PLAY 规则。如果选择符不匹配，将使用从父元素那里继承来的值。如果从父元素那里没有继承任何值，就使用缺省值。

如果在给定的特征级中有多个规则，那么级联顺序按下列优先级决定：

1. 作者标记为重要的声明。
2. 读者标记为重要的声明。
3. 作者未标记为重要的声明。
4. 读者未标记为重要的声明。
5. 样式单中的最近的规则。

应尽量避免依赖于级联顺序。指定尽可能少的样式，并且让读者浏览器的优选项处于主控地位，就会很少出现错误。

## 12.6 在 CSS 样式单中添加注释

可以在 CSS 样式单中包含注释。CSS 注解类似于 C 语言的 `/* */` 注释，而不像 XML 和 HTML 的 `<!-->` 注释。清单 12-6 显示了这种注解的使用方法。不过，本样式单不能把样式规则应用于元素。它以英语来描述样式规则可能出现的结果。

清单 12-6：含有注释的诗的样式单

```
/* Work around a Mozilla bug */

POEM { display: block }

/* Make the title look like an H1 header */

TITLE { display: block; font size: 16pt; font-weight: bold }

POET { display: block; margin-bottom: 10 }

/* Put a blank line in-between stanzas,
only a line break between verses */

STANZA { display: block; margin-bottom: 10 }

VERSE { display: block }
```

CSS 比 XML DTD、Java、C 或 Perl 更容易理解，所以不像其他语言那样需要使用注释。可是，包含注释也并非不好。使用注释可有助于那些想要弄明白所写的样式单的意义，而无法直接提出问题的人。

# 12.7 CSS 中的单位

CSS 属性具有名称和值。表 12-1 列举了一些属性名及其值。

CSS 的所有名称都是关键字。但是，值则千变万化。有些值是关键字，如 `display:none` 中的 `none` 或 `border-style:solid` 中的 `solid`；有些值是带有单位的数字，如 `margin-top:0.5in` 中的 `0.5in` 或 `font-size:12pt` 中的 `12pt`；而另外还有一些值则为 URL，如 `background-image: url(http://www.idgbooks.com/images/paper.gif)` 中的 <http://www.idgbooks.com/images/paper.gif>，或者为 RGB 颜色，如 `color:#CC0033` 中的 `#CC0033`。不同的属性可以为不同的值，但是，一个属性只能为下列四种类型的值：

- 1. 长度
- 2. URL
- 3. 颜色
- 4. 关键字

关键字随属性而变，但其他类型的值则与属性无关，保持定值。也就是说，长度值就是一个长度的长短，而不管其是哪个属性的值。如果知道如何指定一个边界的长度，也就知道如何指定页边距 (`margin`)、贴边 (`padding`) 和图像的长度。语法的这种重用性使处理不同的属性变得容易一些。

表 12-1 属性名和属性值的示例

名称	值
<code>display</code>	<code>None</code>
<code>font-style</code>	<code>Italic</code>
<code>margin-top</code>	<code>0.5in</code>
<code>font-size</code>	<code>12pt</code>
<code>border-style</code>	<code>Solid</code>
<code>color</code>	<code>#CC0033</code>
<code>background-color</code>	<code>White</code>
<code>background-image</code>	<code>url(<a href="http://www.idgbooks.com/images/paper.gif">http://www.idgbooks.com/images/paper.gif</a>)</code>
<code>list-style-image</code>	<code>url(/images/redbullet.png)</code>
<code>line-height</code>	<code>120%</code>

## 12.7.1 长度值

在 CSS 中，长度是一种度量尺寸，用于宽度、高度、字号、字和字母间距、文本的缩排、行高、页边距、贴边、边框线宽以及许许多多的其他属性。可以用下列三种方法指定长度：

- 1. 绝对单位
- 2. 相对单位
- 3. 百分比

12.7.1.1 长度的绝对单位

长度的绝对单位，似乎有点用词不当的感觉，因为在计算机屏幕上确实没有绝对的长度单位。将屏幕的分辨率从 640´ 480 改为 1600´ 1200，就会改变计算机中的所有长度，包括以英寸和厘米为单位的长度。但是，CSS 支持五种“绝对”长度单位(至少其字体不会发生变化)。表 12-2 列出了这五种长度的绝对单位。

表 12-2 长度的绝对单位

	英寸(in)	厘米(cm)	毫米(mm)	磅(pt)	十二点(pc)
英寸	1.0	2.54	25.4	72	6
厘米	0.3937	1	10	28.3464	4.7244
毫米	0.03937	0.1	1.0	2.83464	0.47244
磅	0.01389	0.0352806	0.352806	1.0	0.83333
十二点	0.16667	0.4233	4.233	12	1.0

长度是以数值来表示的，其后紧跟着表示长度单位的缩写字母：

英寸	in
厘米	cm
毫米	mm
磅	pt
十二点	pc

长度单位的数值可能有小数点(如，margin-top:0.3in)。有些属性允许为负值，如-0.5in，但并非都如此；甚至有些属性往往限制其负长度值的大小。要使交叉浏览器具有最大的兼容性，最好避免使用负值。

12.7.1.2 长度的相对单位

CSS 还支持下列三种长度的相对单位：

- 1. em：当前字体中字母 *m* 的宽度
- 2. ex：当前字体中字母 *x* 的高度



3. px: 一个像素的大小(假设为方形像素; 如今的普通显示器都使用方形像素, 尽管几乎现在就要被丢进垃圾箱的有些旧 PC 显示器并非如此)

例如, 下列规则设置 PULLQUOTE 元素的左右边为当前字体中字母 *m* 宽度的两倍, 顶和底边为当前字体中字母 *x* 高度的 1.5 倍:

```
PULLQUOTE { border-right-width: 2em; border-left-width: 2em;

border-top-width: 1.5ex; border-bottom-width: 1.5ex }
```

使用 em 和 ex 的目的就是为所给的字体设置合适的宽度, 而没有必要知道字体有多大。例如, 在上面的规则中, 不知道字体大小, 所以边界的准确宽度也不知道。在显示时, 可通过比较当前字体中的 *m* 和 *x* 来确定。字体越大, 所对应的 em 和 ex 也就越大。

以像素为单位的长度是相对于显示器上的像素(或许为方形)的高度和宽度。影像的宽度和高度经常是以像素给出的。

像素测量法通常不是个好方法。首先, 像素的大小依分辨率变化较大。大多数资深用户都会将显示器设置成尽可能高的分辨率, 从而导致像素太小, 而无法阅读。

第二, 在未来的十年中, 200 甚至 300dpi 分辨率的显示器将会普及起来, 最终代替粗糙的每英寸 72 像素(给出或获得 28 像素)的德国事实标准, 这个标准是 1984 年第一台 Macintosh 计算机以来盛行的。以非基于屏幕的单位(如 em、ex、磅、十二点字和英寸)来指定度量的文档, 可以进行转换。但是, 使用像素级规格的文档, 在高分辨率下浏览时, 就会变得太小, 无法阅读。

### 12.7.1.3 长度的百分比单位

最后, 长度也可以用某一百分比来指定。一般地, 它是某一属性当前值的百分比。例如, 如果 STANZA 元素的字号为 12 磅, 并且 STANZA 包括的 VERSE 的字号设置为 150%, 那么 VERSE 的字号将为 18 磅。

### 12.7.2 URL 值

有三个 CSS 属性可以包括 URL 值: background-image、list-style-image 和简略语属性 list-style。此外, 正如已看到的那样, @import 规则也使用 URL 值。按字面理解, URL 放置在 url() 内。允许使用所有的相对和绝对 URL 值。例如:

```
DOC { background-image: url(http://www.mysite.com/bg.gif) }
```

```
LETTER { background-image: url(/images/paper.gif) }
```

```
SOFTWARE { background-image: url(../images/screenshot.gif) }
```

```
GAME { background-image: url(currentposition.gif) }
```

可用单或双引号把 URL 括起来, 当然这样做并非必要。例如:

```
DOC { background-image: url("http://www.mysite.com/bg.gif") }
```

```
LETTER { background-image: url("/images/paper.gif") }
```

```
SOFTWARE { background-image: url( ../images/screenshot.gif ) }
```

```
GAME { background-image: url( currentposition.gif ) }
```



出现在 URL 中的圆括号、逗号、空格字符、单引号 ( ‘ ) 和双引号 ( “ ) 必须用反斜杠加以转义： ‘ \ ( ’ 、 ‘ \ ) ’ 、 ‘ \ , ’ 。在 URL 内出现的任何圆括号、省略语符号、空格或引号 (可能常用于空字符) 应由 URL 标准的 % 转义码来代替。于是：

space %20

, %2C

‘ %27

“ %22

( %2B

) %2D

CSS 为这些字符 ( \ ( 、 \ ) 、 \ , 、 \ ’ 和 \ ” ) 定义自己的反斜杠转义码，但这些只会造成额外的混乱。

### 12.7.3 颜色值

CSS 胜过 HTML 而被广泛采用的原因之一就是它能够将前景色和背景色应用于页面上的任何元素。使用颜色值的属性有 color、background-color 和 border-color。

CSS 提供了四种方式来指定颜色：名称、十六进制分量、十进制分量以及百分比。由名称定义颜色是最简单的。CSS 理解从 Windows VGA 调色板中采用的 16 种颜色的名称：

“ 浅绿 (aqua)	“ 海蓝 (navy)
“ 黑色 (black)	“ 橄榄 (olive)
“ 蓝色 (blue)	“ 紫色 (purple)
“ 紫红 (fuchsia)	“ 红色 (red)
“ 灰色 (gray)	“ 银色 (silver)
“ 绿色 (green)	“ 深青 (teal)
“ 酸橙 (lime)	“ 白色 (white)
“ 栗色 (maroon)	“ 黄色 (yellow)

当然，普通的彩色显示器可以显示几百万种以上的颜色。可以给颜色的红 (red)、绿 (green) 和蓝 (blue) (RGB) 分量提供值，来指定这些颜色。由于 CSS 是假设为 24 位的颜色模式，所以每种黄色都赋为 8 位。一个 8 位无符号整数是 0~255 之间的数值。这个值可以是十进制也可以是十六进制 RGB 值。另外，它也可以用 0% (0) 到 100% (255) 之间的 RGB 百分数来指定。表 12-3 列出了一些可能的颜色以及十进制、十六进制和 RGB 百分数。

表 12-3 CSS 实例的颜色





颜色	十进制 RGB	十六进制 RGB	RGB 百分数
纯红	rgb(255, 0, 0)	#FF0000	rgb(100%, 0%, 0%)
纯蓝	rgb(0, 0, 255)	#0000FF	rgb(0%, 0%, 100%)
纯绿	rgb(0, 255, 0)	#00FF00	rgb(0%, 100%, 0%)
白色	rgb(255, 255, 255)	#FFFFFF	rgb(100%, 100%, 100%)
黑色	rgb(0, 0, 0)	#000000	rgb(0%, 0%, 0%)
浅紫	rgb(255, 204, 255)	#FFCCFF	rgb(100%, 80%, 100%)
浅灰	rgb(153, 153, 153)	#999999	rgb(60%, 60%, 60%)
褐色	rgb(153, 102, 51)	#996633	rgb(60%, 40%, 20%)
粉红	rgb(255, 204, 204)	#FFCCCC	rgb(100%, 80%, 80%)
橙色	rgb(255, 204, 102)	#FFCC66	rgb(100%, 80%, 40%)

许多人仍然使用 256 色的显示器。而且，有些颜色 Mac 和 PC 机上有明显的不同。最可靠的颜色就是 16 个命名的颜色。

次可靠的颜色是只使用十六进制分量 00、33、66、99、CC 和 FF(十进制分别为 0、51、102、153、204 和 255；以百分数单位表示的 0%、20%、40%、60%、80%和 100%)组成的那些颜色。例如，33FFCC 是“浏览器安全”色，因为红色成分是由两个 3、绿色由两个 F 和蓝色由两个 C 组成的。

如果只使用三位数字来指定十六进制 RGB 颜色，那么，CSS 就将数字加以重复；例如，#FC0 实际上就是#FFCC00，#963 实际上是#996633。

12.7.4 关键字值

关键字是 CSS 属性可能具有的四类值中变化最大的。通常，它们随属性而变，但相类似的属性一般支持类似关键字。例如，border-left-style 值可能是 none、dotted、dashed、solid、double、groove、ridge、inset 或 outset 任何一个关键字。border-right-style、border-top-style、border-bottom-style 和 border-style 属性也可认为是下列值之一：none、dotted、dashed、solid、double、groove、ridge、inset 或 outset。各关键字将在有关各自属性的章节中讨论。



## 12.8 块、内联或列表项元素

从 CSS1 来看，所有的元素要么是块级元素、要么是内联元素、列表项元素或不可见。(CSS2 添加了更多的可能性。)某个给定元素的类型是由其 display 属性设置的。此属性根据所给的关键字有四个可能的值：

block

inline

list-item

none

在 CSS1 中，display 属性的缺省值是 block，这意味着此项出现在它自己的方框中，并且以某种方式与其他元素分开。但在 CSS2 中，缺省值已变为 inline，这意味着文本中元素的内容只是依次放在前一元素之后。大多数 Web 浏览器使用 CSS2 缺省值 (inline)，而不是 CSS1 的缺省值 (block)。

在 HTML 中，EM、STRONG、B、I 和 A 都是内联元素。例如，可以把本段中的 EM、STRONG、B、I 和 A 认为是内联码元素。它们没有与其他文本分开。

块级元素通常使用换行与其他块级元素分隔开。在 HTML 中，P、BLOCKQUOTE、H1 直至 H6 和 HR，都是块级元素。在本页中看到的各段都是块级元素。块级元素可以包含内联元素或其他块级元素，但内联元素只能包含其他内联元素，而不能包含块级元素。

列表项元素是在其前有列表项标记的块级元素。在 HTML 中，L1 是列表项元素。列表项元素在将下面章节中进一步讨论。

最后，当把元素的 display 属性设置为 none 时，就会使该元素不可见，即不在屏幕上加以显示。网页上的其他可见元素则不受影响。在 HTML 中，TITLE、META 和 HEAD 可能有 none 值的 display 属性。在 XML 中，display:none 对于元素中的元信息通常是有用的。

考察清单 12-7，它是莎世比亚的《第十二夜》(Twelfth Night) 剧本的大纲。它包括下列元素：

SYNOPSIS ACT\_NUMBER

TITLE SCENE\_NUMBER

ACT LOCATION

SCENE CHARACTER

只使用显示属性就可以使数据格式化。SYNOPSIS、TITLE、ACT 和 SCENE 都是块级元素。ACT\_NUMBER、SCENE\_NUMBER、LOCATION 和 CHARACTER 作为内联元素。清单 12-8 是一个非常简单的样式单，此样式单即可完成这一任务。

清单 12-7：在 XML 中莎世比亚的《第十二夜》的一个大纲

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

<?xml-stylesheet type="text/css" href="12-8.css"?>

<SYNOPSIS>
```

<TITLE>Twelfth Night</TITLE>

<ACT>

<ACT\_NUMBER>Act 1</ACT\_NUMBER>

<SCENE>

<SCENE\_NUMBER>Scene 1</SCENE\_NUMBER>

<LOCATION><CHARACTER>Duke Orsino</CHARACTER>'s palace

</LOCATION>

</SCENE>

<SCENE>

<SCENE\_NUMBER>Scene 2</SCENE\_NUMBER>

<LOCATION>The sea coast</LOCATION>

</SCENE>

<SCENE>

<SCENE\_NUMBER>Scene 3</SCENE\_NUMBER>

<LOCATION><CHARACTER>Olivia</CHARACTER>'s house

</LOCATION>

</SCENE>

<SCENE>

<SCENE\_NUMBER>Scene 4</SCENE\_NUMBER>

<LOCATION><CHARACTER>Duke Orsino</CHARACTER>'s palace.

</LOCATION>

</SCENE>

<SCENE>

<SCENE\_NUMBER>Scene 5</SCENE\_NUMBER>

<LOCATION><CHARACTER>Olivia</CHARACTER>'s house

</LOCATION>

</SCENE>

</ACT>

<ACT>

<ACT\_NUMBER>Act 2</ACT\_NUMBER>

<SCENE>

<SCENE\_NUMBER>Scene 1</SCENE\_NUMBER>

<LOCATION>The sea-coast</LOCATION>

</SCENE>

<SCENE>

<SCENE\_NUMBER>Scene 2</SCENE\_NUMBER>

<LOCATION>A street</LOCATION>

</SCENE>

<SCENE>

<SCENE\_NUMBER>Scene 3</SCENE\_NUMBER>

<LOCATION><CHARACTER>Olivia</CHARACTER>'s house

</LOCATION>

</SCENE>

<SCENE>

<SCENE\_NUMBER>Scene 4</SCENE\_NUMBER>

<LOCATION><CHARACTER>Duke Orsino</CHARACTER>'s palace.

</LOCATION>

</SCENE>

<SCENE>

<SCENE\_NUMBER>Scene 5</SCENE\_NUMBER>

<LOCATION><CHARACTER>Olivia</CHARACTER>'s garden

</LOCATION>

</SCENE>

</ACT>

<ACT>

<ACT\_NUMBER>Act 3</ACT\_NUMBER>

<SCENE>

<SCENE\_NUMBER>Scene 1</SCENE\_NUMBER>

<LOCATION><CHARACTER>Olivia</CHARACTER>'s garden

</LOCATION>

</SCENE>

<SCENE>

<SCENE\_NUMBER>Scene 2</SCENE\_NUMBER>

<LOCATION><CHARACTER>Olivia</CHARACTER>'s house

</LOCATION>

</SCENE>

<SCENE>

<SCENE\_NUMBER>Scene 3</SCENE\_NUMBER>

<LOCATION>A street</LOCATION>

</SCENE>

<SCENE>

<SCENE\_NUMBER>Scene 4</SCENE\_NUMBER>

<LOCATION><CHARACTER>Olivia</CHARACTER>'s garden

</LOCATION>

</SCENE>

</ACT>

<ACT>

<ACT\_NUMBER>Act 4</ACT\_NUMBER>

<SCENE>

<SCENE\_NUMBER>Scene 1</SCENE\_NUMBER>

<LOCATION><CHARACTER>Olivia</CHARACTER> s front yard

</LOCATION>

</SCENE>

<SCENE>

<SCENE\_NUMBER>Scene 2</SCENE\_NUMBER>

<LOCATION><CHARACTER>Olivia</CHARACTER> s house

</LOCATION>

</SCENE>

<SCENE>

<SCENE\_NUMBER>Scene 3</SCENE\_NUMBER>

<LOCATION><CHARACTER>Olivia</CHARACTER> s garden

</LOCATION>

</SCENE>

</ACT>

<ACT>

<ACT\_NUMBER>Act 5</ACT\_NUMBER>

<SCENE>

<SCENE\_NUMBER>Scene 1</SCENE\_NUMBER>

<LOCATION><CHARACTER>Olivia</CHARACTER> s front yard

</LOCATION>

</SCENE>

</ACT>

</SYNOPSIS>

清单 12-8：用于剧本大纲的简单的样式单



SYNOPSIS, TITLE, ACT, SCENE { display: block }

图 12-4 显示的是使用清单 12-8 的样式单加载到 Mozilla 中的《第十二夜》大纲。注意，在清单 12-8 中，没有必要明确地指定 ACT\_NUMBER、SCENE\_NUMBER、LOCATION 和 CHARACTER 为内联元素。因为这是缺省值，除非有其他指定。display 属性不能被子属性所继承。因而，仅因为 SCENE 是块级元素并不意味着其子元素 SCENE\_NUMBER 和 LOCATION 也是块级元素。

### 12.8.1 列表项

如果为 display 属性选择了 list-item 值，就有三种其他属性可供设置。这些属性影响列表项如何显示。下面列出了这三个附加属性：

1. list-style-type
2. list-style-image
3. list-style-position

还有一个简略的 list-style 属性，使用它就可以在一个规则中设置所有的三个属性。

Internet Explorer 5.0 和 Mozilla 5.0 第三版仍不支持 display:list-item 属性。Mozilla 把列表项作为简单的块级元素来处理，而 Internet Explorer 则更差，它把列表项作为内联元素来处理。

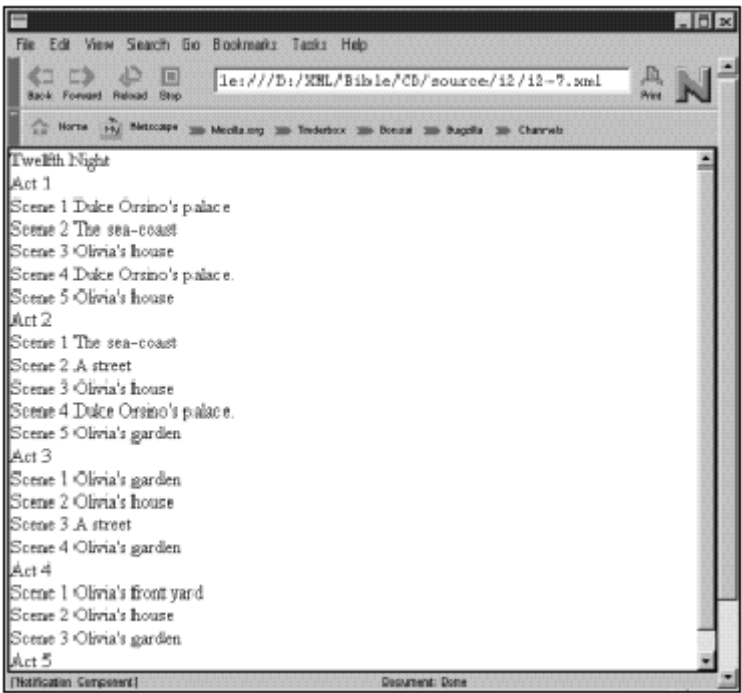


图 12-4 在 Mozilla 5.0 中显示的《第十二夜》大纲

list-style-type 属性确定每个列表项前面的项目符号字符的性质，其可能值为：

- disc
- circle
- square
- decimal

lower-roman

upper-roman

lower-alpha

upper-alpha

none

缺省值为 disc。例如，把清单 12-9 中的样式单应用到清单 12-7 的大纲中，就定义了 ACT 和 SCENE 为列表项。可是，ACT 没有项目符号，而 SCENE 赋以方形的项目符号。

清单 12-9：使用列表项的剧本大纲的样式单

```
SYNOPSIS, TITLE { display: block }
```

```
ACT { display: list-item; list-style-type: none }
```

```
SCENE { display: list-item; list-style-type: square }
```

#### 12.8.1.2 list-style-image 属性

同样，也可使用从文件中加载的点位图影像作为项目符号。为此，应设置 list-style-image 属性为影像的 URL。如果 list-style-image 和 list-style-type 都被设置，那么，列表项前面就既放置影像也放置项目符号。但这情况很少见。清单 12-10 在各个场景前使用保存在 heart.gif 文件中的© 作为项目符号（《第十二夜》毕竟是一部浪漫喜剧）。

清单 12-10：使用 list-style-image 属性的用于剧本大纲的样式单

```
SYNOPSIS, TITLE { display: block }
```

```
ACT { display: list-item; list-style-type: none }
```

```
SCENE { display: list-item;
```

```
list-style-image: url(heart.gif); list-style-type: none }
```

#### 12.8.1.3 list-style-position 属性

list-style-position 属性指定项目符号是在列表项文本之内还是之外绘制。合法的值是 inside 和 outside。缺省值为 outside。当文本折成多行时，才会看出明显的差别。下列是 list-style-position 属性为 inside 时的显示结果：

- If music be the food of love, play on/Give me excess of it, that, surfeiting,/The appetite may sicken, and so die./That strain again! it had a dying fall: <sup>[註]</sup>

下列是 list-style-position 属性为 outside 时的显示结果：

- If music be the food of love, play on/Give me excess of it, that, surfeiting,/The appetite may sicken, and so die./That strain again! it had a dying fall:

#### 12.8.1.4 list-style 简略语属性



最后介绍一下 `list-style` 属性，此属性是一种简写，它允许一次性设置上面描述的所有三个属性。例如下面的规则说明 SCENE 显示在内部，带有红桃影像而没有任何项目符号。

```
SCENE { display: list-item;

list-style: none inside url(heart.gif) }
```

### 12.8.2 `whitespace` 属性

`white-space` 属性确定元素内部空白(空格符、制表符和换行符)的重要性。允许值为：

`normal`

`pre`

`nowrap`

缺省值为 `normal`，仅表示把一连串的空白压缩成一个空格，并且单词折行以与屏幕或页面相适应。这在 HTML 和 XML 中是处理空白的通用办法。

`pre` 值的作用就像 HTML 中的 PRE(预格式化)元素。在输入的文档中，所有的空白都认为是有意义的，并在输出设备上按照原样再现出来，或许有一些稍稍地变化成等宽字体。这对许多计算机源码或一些诗来说，是很有用的。清单 12-11 就是一首诗，名叫 *The Altar*，它是由 George Herbert 创作的，在这首诗中，间距是很重要的。在这首诗中，各行形成了本诗主题的轮廓。

清单 12-11: XML 中的 *The Altar*

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

<?xml-stylesheet type="text/css" href="12-12.css"?>

<POEM>

<TITLE>The Altar</TITLE>

<POET>George Herbert</POET>

<VERSE> A broken ALTAR, Lord, thy servant rears,</VERSE>

<VERSE> Made of a heart, and cemented with tears:</VERSE>

<VERSE> Whose parts are as thy hand did frame;</VERSE>

<VERSE> No workman s tool hath touched the same.</VERSE>

<VERSE> No workman s tool hath touched the same.</VERSE>

<VERSE> A HEART alone</VERSE>

<VERSE> Is such a stone,</VERSE>
```

```

<VERSE> As nothing but</VERSE>

<VERSE> Thy power doth cut.</VERSE>

<VERSE> Wherefore each part</VERSE>

<VERSE> Of my hard heart</VERSE>

<VERSE> Meets in this frame,</VERSE>

<VERSE> To praise thy name:</VERSE>

<VERSE> That if I chance to hold my peace,</VERSE>

<VERSE> These stones to praise thee may not cease.</VERSE>

<VERSE> O let thy blessed SACRIFICE be mine,</VERSE>

<VERSE> And sanctify this ALTAR to be thine.</VERSE>

</POEM>

```

清单 12-12 是一个样式单，它使用 `white-space: pre` 来保留这一形式。图 12-5 为在 Mozilla 中显示的结果。

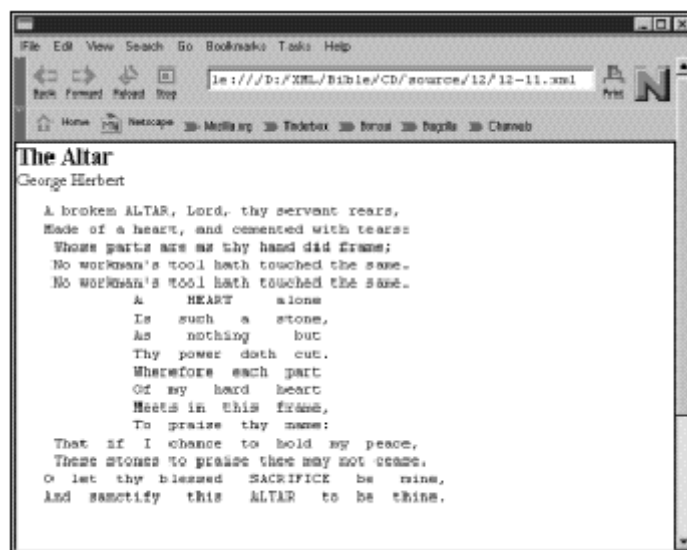


图 12-5 使用 `white-space:pre` 的 George Herbert 诗 *The Altar*



Internet Explorer 5.0 不支持 `white-space:pre`。

清单 12-12: 对空白敏感的诗的样式单

```
POEM { display: block }
```

```
TITLE { display: block; font-size: 16pt; font-weight: bold }
```



```
POET { display: block; margin-bottom: 10px }
```

```
STANZA { display: block; margin-bottom: 10px }
```

```
VERSE { display: block;
```

```
white-space: pre; font-family: monospace }
```

最后介绍一下 `nowrap` 值，它采取一种折衷方案：在源文本中，有明确换行符的位置，准确地折行，但把其他一连串空白压缩成一个单空白。在名著的手稿或其他一些诗中，换行符是非常重要的，而单词之间的间隔则不显得那么重要，此时如实地再现换行符可能是很有用的。

Internet Explorer 5.0 及其早期版本不能正确地支持 `nowrap`。

## 12.9 字体属性

CSS1 支持下列五种基本的字体属性：

- 1. font-family
- 2. font-style
- 3. font-variant
- 4. font-weight
- 5. font-size

此外，还有 font 这个简略属性，它可以一次设置所有的五个属性。

### 12.9.1 font-family 属性

font-family 属性值是一组以逗号分隔开来的字体名，如 Helvetica、Times、Palatino 等。包含空格的字体名(如 “Times New Roman”)应放在双引号内。

这些名称也可能是这五种一般名称之一：serif、sans-serif、cursive、fantasy 和 monospace。浏览器使用本地系统上安装的所需类型的字体来代替这些名称。表 12-4 示例了这些字体。

表 12-4 一般字体

名称	典型的字族	显著特征	示例
衬线体(Serif)	Times, Times New Roman, Palatino	在字母的边缘上形成花体,使以小型的大写形式出现于衬线文本更易于阅读	The quick brown fox jumped over the lazy dog.
非衬线体 (sans-serif)	Geneva, Helvetica, Verdana	印刷体，常用于大字标题	The quick brown fox jumped over the lazy dog.
等宽字体 (Monospace)	Courier, Courier New, Monaco, American, Typewriter	一种打印体,字体中的每个字符的宽度相等,常用于源码和 E-mail	The quick brown fox jumped over the lazy dog.
草书(Cursive)	ZapfChancery	手稿字体，模拟书法	The quick brown fox jumped over the lazy dog.
梦幻体(Fantasy)	Western, Critter	具有特定效果的文本;如热情洋溢的字母、歪斜状特技的字母,以及从动物哪里形成的字母，等等	The quick brown fox jumped over the lazy dog.

由于在特定的客户系统上无法担保所给的任何字体都可以找到或是合适的(10 磅的 Times 在 Macintosh 机上就不如 Palm Pilot, 实际上前者难以辨认), 所以一般提供以一组逗号分开的列表, 以便以优先顺序选择字体。在列表中最后选项总是一个通用名。可是, 即使不指定一个通用名, 以及指定的字体不可用, 那么浏览器也会挑选出某个字体名。这也许不会是人们所希望的那个。

例如, 下面的两条规则的第一条使 TITLE 元素为 Helvetica 字体, 而以任何非衬线体为后备体; 第二条规则使其余元素为 Times 字体, 而用 Times New Roman 和其他衬线体为后备字体。

```
TITLE { font-family: Helvetica, sans serif }

SYNOPSIS { font-family: Times, "Times New Roman", serif }
```

图 12-6 显示的是把这两条规则加到清单 12-8 的样式单之后加载到 Mozilla 5.0 中的剧本大纲。由于在图 12-4 中 Times 通常是缺省字体, 所以没有发生太大的变化。

font-family 属性可由子元素继承。于是, 将 SYNOPSIS 元素的 font-famiey 设置为 Times 字体, 所有的子元素也被设置为 Times, 只是 TITLE 例外, 其自身的 font-family 属性取代了它继承的属性。



图 12-6 标题以 Helvetica 字体显示的《第十二夜》的大纲

12.9.2 font-style 属性

font-style 属性有三个值: *normal*、*italic* 和 *oblique*。读者正在阅读普通文本就是正常体。一般以 HTML 的 EM 元素显示的则为斜体的。倾斜字体(*oblique*)的文本十分类似于斜体文本。但是, 绝大多数情况下, 倾斜字体的文本是由计算机根据简单的算术运算, 使正常的文本倾斜一定的角度实现的。用斜体字印刷的文本, 使用的字体手法, 通常是为了好看。

下面的规则使 SCENE\_NUMBER 元素成斜体:

```
SCENE_NUMBER { font-style: italic }
```

图 12-7 显示的是把上述规则加到大纲样式单之后再加载到 Internet Explorer 5.0 中的剧本大纲。

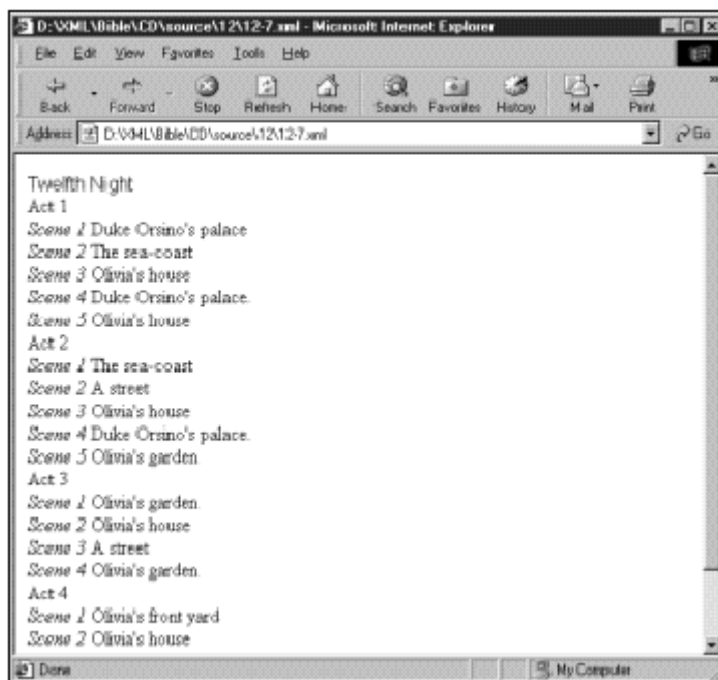


图 12-7 带斜体场景号的《第十二夜》剧本大纲

### 12.9.3 font-variant 属性

在 CSS1 中, font-variant 属性有两个可能值: normal 和 small-caps, 缺省值为 normal。将 font-variant 属性值设置为 small-caps 将会用比主体文本字号小一些的大写字母代替小写字母。

把 font-variant 属性与 first-letter 伪元素结合起来使用, 可获得极好的效果。例如, 定义 ACT\_NUMBER 元素, 使属性为 font-variant:small-caps。接下来, 定义 ACT\_NUMBER 的首字母, 为 font-variant:normal。这样就产生了如下的剧本的幕序号:

ACT 1

下面是其规则:

```
ACT_NUMBER { font-variant: small-caps }
```

```
ACT NUMBER: first-letter { font-variant: normal }
```

第二个规则覆盖第一个, 但仅对幕序号的首字母才如此。

### 12.9.4 font-weight 属性

font-weight 属性决定文本以多黑(粗)或多浅(细)显示。此属性有 13 个值:

normal

bold

bolder

lighter

100

200

300

400

500

600

700

800

900

粗细的范围是从 100(最浅)到 900(最暗)。处于各值之间、非整百的值如 850 是不允许的。正常粗细为 400，粗体为 700。bolder 值会使一个元素变得比其父元素更粗。lighter 值会使一个元素变得比其父元素更细。但是，不能保证某个特定的字体具有多达九个独立的粗体级。

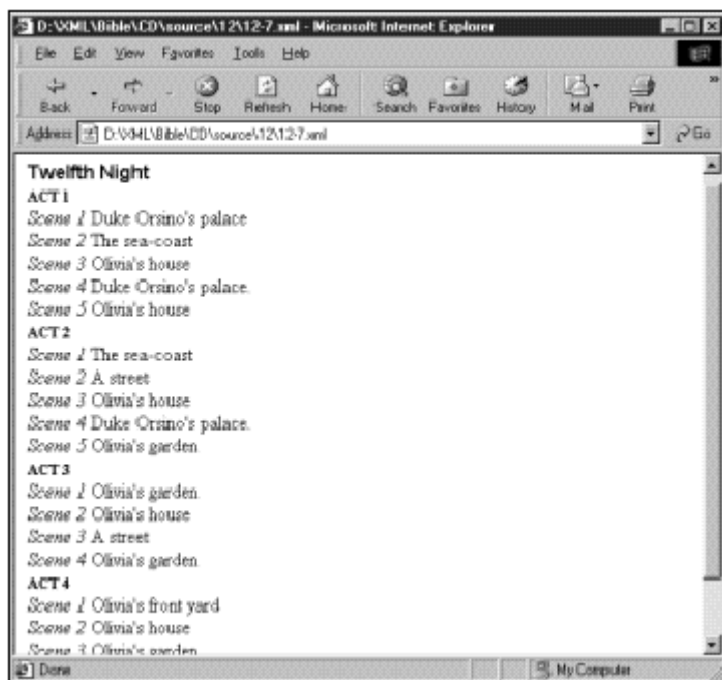
下面举一个简单的规则实例，此规则使 TITLE 和 ACT\_NUMBER 元素成为粗体。

```
TITLE, ACT_NUMBER { font weight: bold }
```

图 12-8 显示的是把上述规则加到清单 12-7 的样式单之后在 Mozilla 浏览器中的结果。

### 12.9.5 font-size 属性

font-size 属性决定字体中典型字符的高和宽度。字号越大，在屏幕中所占的空间也就越大。可以通过这样的几种方式来指定字号：关键字、相对于其父元素的字号、其父元素字号的百分比或绝对数。



### 12.9.5.1 关键字

绝对大小关键字：

xx-small

x-small

small

medium

large

x-large

xx-large

这些关键字是设置字号的首选方式，因为它们仍是相对于页的基本字号的。例如，如果用户由于近视度很高而将缺省字号调整为 20 磅，那么所有其他值也会相应地按比例改变。

在 CSS1 中，每种字号是下一个最小字号的 1.5 倍。缺省值为 medium，因此，如果浏览器的缺省值是 12 磅，那么 large 类型就是 18 磅，x-large 类型就是 27 磅，而 xx-large 类型就是 40.5 磅。同样，small 类型就是 8 磅；x-small 类型就是 5.33 磅；xx-small 类型就是 3.56 磅，此时字迹几乎模糊不清。

下面的简单规则使 TITLE 成为特大字：

```
TITLE { font-size: x-large }
```

### 12.9.5.2 相对于父元素的字号值

也可以将相对于父元素的字号指定为 larger 或 smaller。例如，在下列语句中，SCENE\_NUMBER 将获得比其父元素 SCENE 字体更小的字体。

```
SCENE_NUMBER { font-size: smaller }
```

更小的字体准确地小到多少或更大的字体准确地大到多少，没有任何确切的规则可循。一般地，浏览器会从中等 (medium) 移到小字体 (small)、从小字体移到 x-small，依次类推。对于较大的字体也同样适用 (向另一方向移动)。这样，使一字体变得更大，应将其字号增加到约 50%，而使一字体变得更小，则应将其字号减小约 33%，可是，为了匹配现有的字号，浏览器无法自己产生这些值。

### 12.9.5.3 父元素字号的百分比

如果这些项不够精确，可使用父元素字号的百分比进行更精细的调整。例如，下列规则说明用于 SCENE\_NUMBER 的字体是 SCENE 字号的 50%。

```
SCENE_NUMBER { font-size: 50% }
```

### 12.9.5.4 绝对长度值





最后，可能会把字号以绝对长度给出。尽管可以使用像素、厘米或英寸，但测定字体的最通用的单位是磅。例如，下面规则为 SYNOPSIS 元素设置缺省 font-size 属性，并且将其子元素字体的大小设置为 14 磅。

```
SYNOPSIS { font-size: 14pt }
```

我极力劝说读者不要使用绝对单位来描述字号。要选择一种字号，使它在所有用来浏览网页的不同的平台(从手持机到朝代广场的 Sony 大屏幕)上都可以很清晰、便于阅读是极其困难的(我认为是不可能的)。即使当把它们局限在标准的个人计算机上，大多数设计者选择的字体也会太小。要想让任何文本都能在屏幕上阅读，这些字体应至少为 12 磅，甚至更大。

图 12-9 显示的是将这些规则加到清单 12-7 的样式单中之后，在 Mozilla 中输出的结果。场景的文本并不真正的是粗体，它只是变得稍大。在任何情况下，更容易阅读。

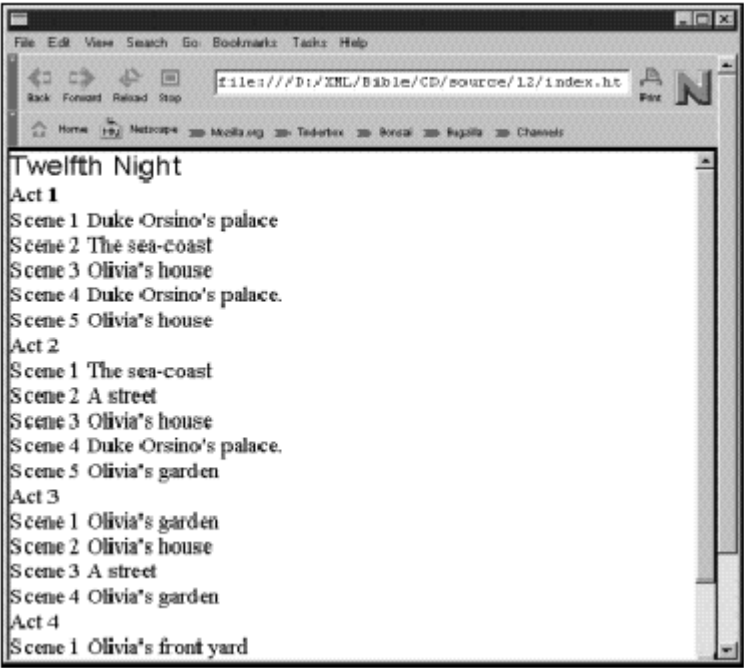


图 12-9 带有各式各样字号的《第十二夜》剧本大纲

12.9.6 font 简略属性

font 属性是一种简略属性，它允许使用一条规则来设置字体样式、变体、粗细、字号或字族。例如，下列用于 TITLE 和 SCENE\_NUMBER 元素的两条规则，就把前面各节中提到的六个不同的规则组合在一起了：

```
TITLE { font: bold x-large Helvetica, sans-serif }  
  
SCENE_NUMBER { font: italic smaller serif }
```

font 属性值必须按下列顺序给出：

- 1. 以任何顺序给出样式、变体和粗细之一，它们中的任何一个都可以忽略
- 2. 字号，不可以忽略
- 3. 可选的斜杠 (/) 和行高
- 4. 字族，不可忽略



如果这听起来很复杂并且难以记住，那是因为本该如此。如不查阅，我肯定记不住这些属性顺序的准确的细节。我宁愿一次只使用一个属性。这样就产生了一个疑问，像这样的简略属性是否真能节省时间。

清单 12-13 使用 font 简略属性，是带有迄今介绍过的所有规则的大纲的样式单。但是，由于 font 属性准确地等同于其代表的各属性之和，所以显示的文档没有任何变化。

清单 12-13：使用 font 简略属性的剧本大纲样式单

```
SYNOPSIS, TITLE, ACT, SCENE { display: block }

SCENENUMBER { font: italic smaller serif }

TITLE { font: bold x-large Helvetica, sans serif }

SYNOPSIS { font: 14pt Times, "Times New Roman", serif }

ACTNUMBER { font-variant: small-caps }

ACTNUMBER:first-letter { font-variant: normal }

ACTNUMBER { font-weight: bold }
```

## 12.10 颜色属性

CSS 允许使用 `color` 属性可将颜色赋给几乎网页上的任何元素。颜色属性的值可以是 16 个命名颜色关键字之一，或以十进制、十六进制或百分数表示的 RGB 三基色。例如，下列规则指定网页上的所有元素都以黑色显示，只是 `SCENE_NUMBER` 除外，它是以蓝色显示的。

```
SYNOPSIS { color: black }
```

```
SCENE_NUMBER { color: blue }
```

`color` 属性可被子属性所继承。所以，除 `SCENE_NUMBER` 之外的大纲中的所有元素都将是黑色的。

下列规则与上面的两个是等价的。在有可能时，我推荐尽可能使用命名颜色，当做不到时，可使用浏览器安全色。

```
SYNOPSIS { color: #000000 }
```

```
SCENE_NUMBER { color: #0000FF }
```

```
SYNOPSIS { color: rgb(0, 0, 0) }
```

```
SCENE_NUMBER { color: rgb(0, 0, 255) }
```

```
SYNOPSIS { color: rgb(0%, 0%, 0%) }
```

```
SCENE_NUMBER { color: rgb(0%, 0%, 100%) }
```



# 12.11 背景属性

元素的背景可设置成一种颜色或一幅影像。如果设置为一幅影像，那么此影像可相对于元素内容加以定位。可使用下列五个基本属性来达此目的：

- 1. background-color
- 2. background-image
- 3. background-repeat
- 4. background-attachment
- 5. background-position

最后，还有一个 background 简略属性，它允许在一条规则中设置某些或所有的五个属性。

如今在 Web 上，人们过度地使用奇特的背景。除了非常明亮的背景以外的其他东西，只会使网页难以阅读，并令用户生厌。出于完整之目的，我在此列出了这些属性，但是，我强烈推荐节制一点使用这些属性(如果不能完全不用的话)。

任何一个背景属性都不会被继承。每个子元素必须指定所需的背景色。但是，由于背景的缺省值为透明的，所以看起来就好像这些背景属性被继承了一样。无论哪个元素的背景绘制于某元素下，它都会透过来。在大多数情况下，这就是父元素的背景。

## 12.11.1 background-color 属性

background-color 属性可设置为与 color 属性一样的值，但是，它并不改变元素内容的颜色，而是改变元素内容绘制其上的背景的颜色。例如，要在蓝色背景上绘制黄色文本的 SIGN 元素，可使用如下的规则：

```
SIGN { color: yellow; background-color: blue }
```

也可以将 background-color 设置为关键字 transparent(缺省值)，这表明背景使用位于其下的元素(通常为父元素)的颜色或影像。

## 12.11.2 background-image 属性

background-image 属性既可以是 none(缺省值)，也可以是指明位图图像文件集团的 URL(通常是相对值)。如果为 URL，则浏览器将加载影像，并把它作为背景使用，这一点极像 HTML 中 BODY 元素的 BACKGROUND 特性。例如，下面显示的是如何将文件 party.gif(如图 12-10 所示)作为 INVITATION 元素的背景。

```
INVITATION { background-image: url(party.gif) }
```



图 12-10 清单 12-14 中舞会请柬背景使用的未平铺且未加裁剪的原始背景影像

由 background-image 属性引用的影像在指定的元素下方绘出,而不是像 HTML 的 BODY 元素的 BACKGROUND 特性那样在浏览器窗格下方绘出。

如果背景影像与基本元素关联,早期的 Mozilla 5.0 测试版把背景影像加到整个文档窗格而不仅仅是元素本身上。但是,对于所有的非基本元素,背景影像只施加于使用它的元素。CSS1 规范并没有明确地表示这一点是否可以接受。

背景影像通常并不是和网页内容的大小精确一致。如果影像比元素框大,影像将会被剪切。如果影像比元素框小,那它就会垂直和水平平铺。图 12-11 显示的背景影像,平铺到足够大以超过下面的内容。注意,这种平铺是在元素窗口中横向穿过,而不是在整个浏览器窗口上进行。

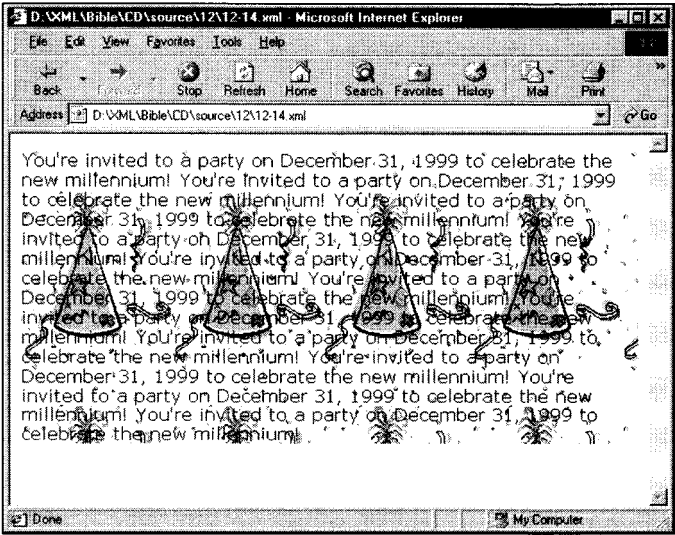


图 12-11 平铺的背景影像

清单 12-14: 以 XML 格式表示的舞会请柬

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

<?xml stylesheet type=" text/css" href=" party.css" ?>

<INVITATION>

You' re invited to a party on December 31, 1999 to celebrate the

new millennium! You' re invited to a party on December 31, 1999

to celebrate the new millennium! You' re invited to a party on

December 31, 1999 to celebrate the new millennium! You' re

invited to a party on December 31, 1999 to celebrate the new

millennium! You' re invited to a party on December 31, 1999 to

celebrate the new millennium! You' re invited to a party on

December 31, 1999 to celebrate the new millennium! You' re
```

invited to a party on December 31, 1999 to celebrate the new  
millennium! You' re invited to a party on December 31, 1999 to  
celebrate the new millennium! You' re invited to a party on  
December 31, 1999 to celebrate the new millennium! You' re  
invited to a party on December 31, 1999 to celebrate the new  
millennium! You' re invited to a party on December 31, 1999 to  
celebrate the new millennium!

</INVITATION>

### 12.11.3 background-repeat 属性

background-repeat 属性调整背景影像如何在屏幕上平铺。可指定背景影像为不平铺或只以水平或垂直方向平铺。此属性可取的值如下：

repeat

repeat-x

repeat-y

no-repeat

例如，要在请柬上仅显示一项舞会帽，可设置 INVITATION 元素的 background-repeat 为 no-repeat。图 12-12 显示的就是这种情景。例如：

```
INVITATION { background-image: url(party.gif);
```

```
background-repeat: no-repeat }
```

要在页面上横向平铺而不是向下平铺，可设置 background-repeat 为 repeat-x，如下所示。图 12-13 显示的背景影像是横向平铺而不是向下平铺。

```
INVITATION { background-image: url(party.gif);
```

```
background-repeat: repeat-x }
```





在 HTML 中，背景影像附加到文档上的。当滚动文档时，背景影像也随之滚动。使用 `background-attachment` 属性，就可以指定背景只附加于窗口或是窗格上。可取的值是 `scroll` 和 `fixed`。缺省值是 `scroll`，这就是说，背景附加于文档而不是窗口之上。

但是，将 `background-attachment` 设置为 `fixed`，文档就会滚动，但背景影像不滚动。当不想使此影像平铺，而对于一般类型的浏览器来说图像已大到充满窗口，但大文档的背景要小时，将此属性设置为 `fixed` 可能是有用的。这时可编码如下：

```
DOCUMENT { background-attachment: fixed;
```

```
background-repeat: no-repeat }
```

无论是 IE 5，还是 Mozilla 都不支持固定的背景影像。在以后的发行版中有可能增加此功能 (CSS1 规范不需要浏览器支持固定背景)。

### 12.11.5 background-position 属性

在缺省条件下，背景影像的左上角与它所连接的元素的左上角对齐 (见图 12-12)。在多数情况下，这样就能满足人们的需要。但是，在少数情况下，可能还需要其他可能，`background-position` 就允许人们相对于元素来移动背景。

可使用父元素的宽和高度的百分数、绝对长度或下列六个关键字中的两个来指定偏移量：

`top`

`center`

`bottom`

`left`

`center`

`right`

#### 12.11.5.1 父元素的宽和高度的百分数

百分数能够把各部分的背景与元素的相应部分紧紧地联系在一起。x 坐标的百分数范围从 0% (左侧) 到 100% (右侧) 变化。y 坐标的百分数变化范围是从 0% (顶端) 到 100% (底部)。例如，下面这条规则把影像的右上角置于 INVITATION 的右上角。图 12-15 为该情况的显示结果。

```
INVITATION { background-image: url(party.gif);
```

```
background-repeat: no-repeat;
```

```
background-position: 100% 0% }
```





top left		top		top right
left top		top center		right top
0% 0%		center top		100% 0%
		50% 0%		
left		center		right
center left		center center		center right
left center		50% 50%		right center
0% 50%				100% 50%
bottom left		bottom		bottom right
left bottom		bottom center		right bottom
0% 100%		center bottom		100% 100%
		50% 100%		

图 12-17 背景影像的相对位置

例如，以请柬为例，最好的效果是把影像牢牢地固定于窗口的中间(如图 12-18 所示)。下列是需要的规则：

```
INVITATION { background-image: url(party.gif);
```

```
background-repeat: no-repeat;
```

```
background-position: center center }
```

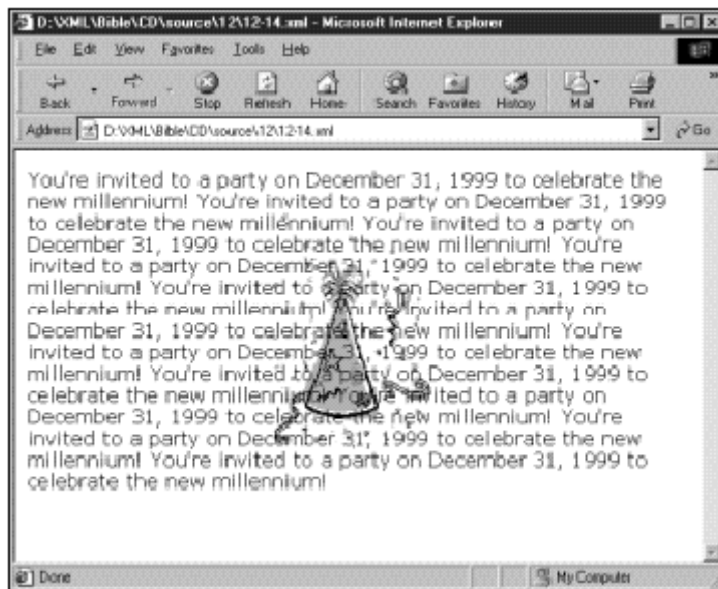


图 12-18 固定于 INVITATION 元素中间的无平铺背景影像

如果 background-attachment 属性值为 fixed，那么影像放置的位置就是窗口边的相对位置，而不是元素的相对位置。

#### 12.11.6 背景简略属性

background 属性是在一条规则中设置 background-color、background-image、background-repeat、background-attachment 和 background-position 属性的简略方法。例如，要在 INVITATION 元素中将 background-color 设置为 white，background-image 设置为 party.gif，background-repeat 设置为 no-repeat 以及 background-attachment 设置为 fixed，可使用下面的规则：

```
INVITATION { background: url(party.gif) white no-repeat fixed }
```

下面的规则与上面的意思是完全一样的，只是句子更长，但更容易理解：

```
INVITATION { background-image: url(party.gif);
```

```
background-color: white;
```

```
background-repeat: no-repeat;
```

```
background-attachment: fixed }
```

当使用 background 简略属性时，对于这五个属性的任何一个或所有的值都可按任何顺序给出。但是，可以省略不只一个值。例如，用于图 12-16 的右上角对齐规则还可以表示如下：

```
INVITATION { background: url(party.gif) no-repeat 100% 0% }
```

## 12.12 文本属性

如不考虑字体，影响文本外观的属性有 8 个：

1. word-spacing
2. letter-spacing
3. text-decoration
4. vertical-align
5. text-transform
6. text-align
7. text-indent
8. line-height

### 12.12.1 word-spacing 属性

word-spacing 属性通过在单词之间附加空格而使文本扩大。负值会删除单词之间的空白。要改变 Web 网页上的词间距，我认为唯一的理由就是，如果你是位学生，工作时受到页面数目的限制，而又想使纸张看起来比实际的大或小一些。



桌面出版商热衷于调节这类属性。问题是他们已了解到的有关如何、什么时候调整间隔的所有准则是基于纸上印刷的，但在转换到以磷光体(一般为 CRT 显示器)表现的电子媒介时就失效。让浏览器来决定字和字母的间距，几乎永远是最好的方法。

另一方面，如果目标媒体是纸上的墨水，那么通过调整这些属性就可以获得更多的空间。主要区别是，在纸上使用墨水就能控制输出媒体。准确地知道字体多大、显示器有多宽和多高、每英寸有多少点可供使用等等。实际是，在 Web 上，没有有关可供输出媒体的足够信息，以便详细地控制所有的情况。

如要改变 word-spacing 的缺省值(normal)，可给属性设置长度。例如：

```
INVITATION { word-spacing: 1em }
```

如浏览器不关注此属性，尤其是如果它与其他属性(如 align: justified)发生冲突时更是如此。Internet Explorer 5.0 不支持 word-spacing，但 Mozilla 则支持(如图 12-19 所示)。

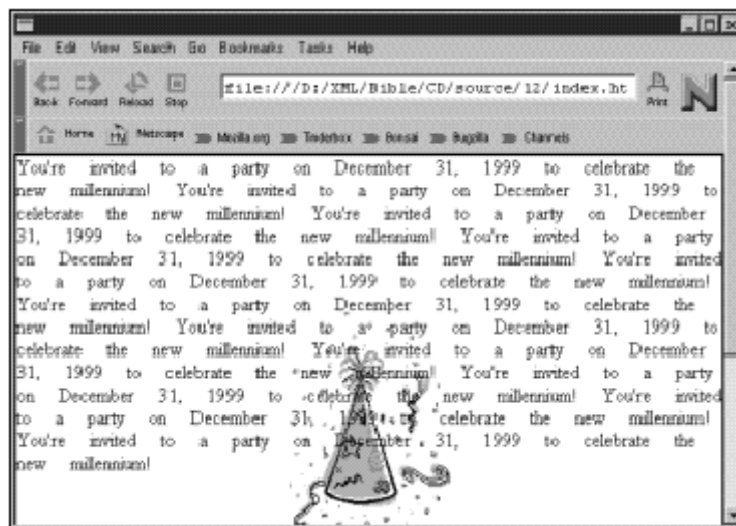


图 12-19 字间距为 1em 的 INVITATION 元素

### 12.12.2 letter-spacing 属性

letter-spacing 属性能在字母之间添加额外的空隙从而扩展文本。设置为负值，就会删除字母之间的空隙。同样，我认为在 Web 网页上这样做的唯一的理由是让纸张看起来比实际的更大或更小，以满足纸张长度要求。

要从 normal 缺省值变为其他值，可为此属性设置一长度。例如：

```
INVITATION { letter-spacing: 0.3em }
```

由于通过调整字母之间空隙大小就可以获得两端对齐，所以手工改变字母间距就可以防止浏览器使文本两端对齐。

浏览器不关注此属性，尤其是如果它与其他属性(如 align: justified)发生冲突时更是如此。但是，Internet Explorer 和 Mozilla 都能体现此属性(如图 12-20 所示)。

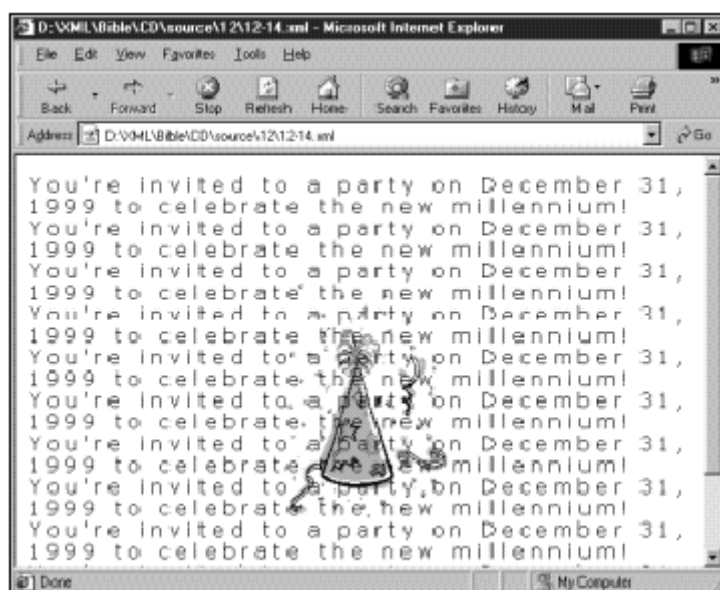


图 12-20 字母间距为 0.3em 的 INVITATION 元素

### 12.12.3 text-decoration 属性

text-decoration 属性可以是下列五个值之一：



none

underline

overline

line-through

blink

除了 none(缺省值)之外，这些值都不互斥。例如，可以为一个段落指定下划线(underline)、上线(overline)、删除线(struck through)和闪烁(blink)。(可我不推荐这样做。)

浏览器不支持闪烁文本。这是一件好事情。

例如，下面的这条规则指定 CHARACTER 元素加上下划线。图 12-21 显示了将此规则应用于清单 12-7 中的《第十二夜》大纲的情景。

```
CHARACTER { text-decoration: underline }
```

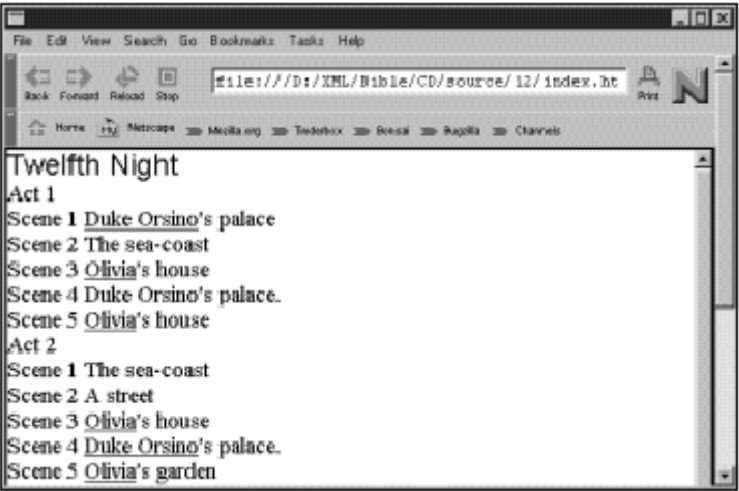


图 12-21 带下划线字符的 *Twelfth Nigh* 大纲

#### 12.12.4 vertical-align 属性

vertical-align 属性指定内联元素如何根据文本的基线定位。有效值是：

baseline

sub

super

top

text-top

middle

bottom

text-bottom

还可以使用元素的行高的百分数来表示。缺省值是 baseline，表示把元素的基线与其父元素的基线对齐。

sub 值使元素成为下标。super 值使元素成为上标。text-top 值使元素的顶端与父元素字体的顶端对齐。middle 值使元素的垂直中心与父元素的基线加上 x-height 一半对齐。text-bottom 值使元素的底部与父元素字体的底部对齐。

top 值使元素的顶端与此行上最高字母或元素的顶部对齐。bottom 值使元素的底部与此行上最低字母或元素的底部对齐。精确的对齐方式随最高或最低字母的高度而变。

例如，用于脚注数字的规则如下面的语句所示，它把数字成为上标，并且大小减小 20%。

```
FOOTNOTE_NUMBER { vertical-align: super; font-size: 80% }
```

### 12.12.5 text - transform 属性

text-transform 属性可用来指定文本应以全部大写字母、全部小写字母或首字母为大写形式显示。例如，这个属性用于标题是很有用的。有效值为：

capitalize

uppercase

lowercase

none

capitalize(以大写字母开头)仅使每个单词的首字母成为大写(如本句：Capitalization Makes Only The First Letter Of Every Word Uppercase)。但如把句子设置成 uppercase(大写体)将使句子中的每个字母成为大写(如 PLACING THE SENTENCE IN UPPERCASE, HOWEVER, MAKES EVERY LETTER IN THE SENTENCE UPPERCASE)。下面的规则将《第十二夜》剧本大纲中的 TITLE 元素转换成大写体。图 12-22 为应用了此规则后所显示的大纲。

```
TITLE { text-transform: uppercase }
```

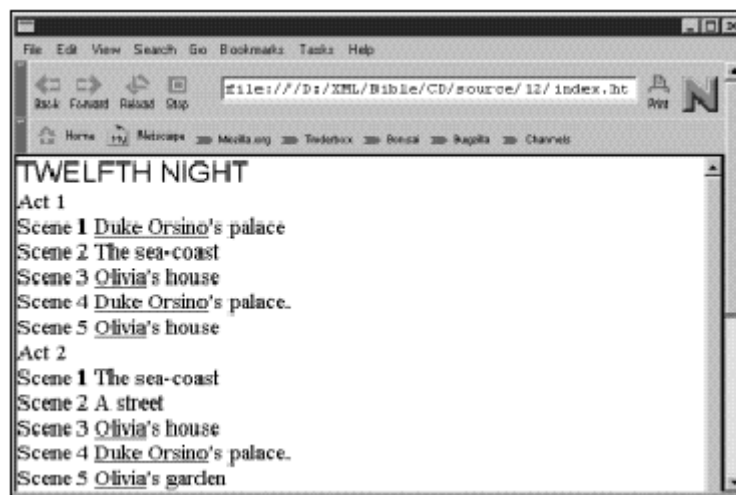


图 12-22 大纲中的 TITLE 现在成为大写

text-transform 属性与语言有关，因为许多语言(如中文)就没有任何大写和小写的概念。

### 12.12.6 text-align 属性

text-align 属性只应用于块级元素。它指定块中的文本是否为左对齐、右对齐、居中或两端对齐。

left

right

center

justify

下面的规则使《第十二夜》剧本大纲中的 TITLE 元素居中，并使其他的两端对齐。图 12-23 显示了应用这些规则之后的大纲。

```
TITLE { text-align: center }
```

```
SYNOPSIS { text-align: justify }
```



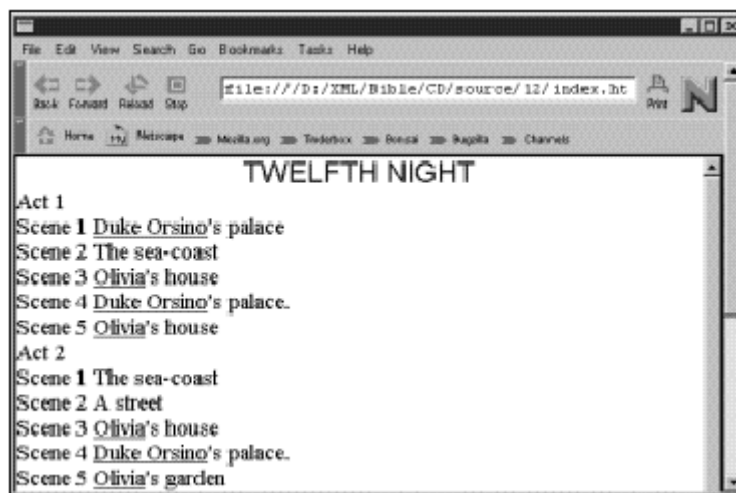


图 12-23 大纲中 TITLE 居中，而文本的其他部分两端对齐

### 12.12.7 text-indent 属性

text-indent 属性只应用于块级元素，它指定块的第一行相对于块其余行的缩排方式，可用绝对长度或父元素宽度的百分数来表示。此值可为负，表示悬挂式缩进。



要使元素的所有行而不仅仅是第一行都缩进，可使用框属性(将在下节讨论)在元素上设置额外的左边距。

例如，下面的规则将大纲中的场景缩进半英寸。图 12-24 显示了在应用了本规则之后的大纲。

```
SCENE { text-indent: 0.5in }
```

### 12.12.8 line-height 属性

line-height 属性指定后续行基线之间的距离，可用绝对数字、绝对长度或字号的百分数来表示。例如，下面的这条规则使 SYNOPSIS 元素的行距加倍。图 12-25 显示在应用此规则之后的《第十二夜》剧本大纲。

```
SYNOPSIS { line-height: 200% }
```



图 12-24 大纲中的 SCENE 及其子元素都缩进半英寸



图 12-25 间隔加倍的大纲

行距加倍不是特别吸引人，所以我要将其删除。在下一节中，在各元素周围添加了一些额外的页边距，以获得良好的效果。清单 12-15 概述了本节中加入到大纲样式单的所有情况（去掉了行距加倍）。

清单 12-15：文本属性的大纲样式单

```
SYNOPSIS, TITLE, ACT, SCENE { display: block }

SCENE_NUMBER { font: italic smaller serif }

TITLE { font: bold x-large Helvetica, sans-serif }

SYNOPSIS { font: 14pt Times, "Times New Roman", serif }

ACT_NUMBER { font variant: small-caps }

ACT_NUMBER:first-letter { font-variant: normal }

ACT_NUMBER { font-weight: bold }

CHARACTER { text-decoration: underline }

TITLE { text-transform: uppercase }

TITLE { text-align: center }

SYNOPSIS { text-align: justify }

SCENE { text-indent: 0.5in }
```

## 12.13 框属性

CSS 描述了两维的绘制输出内容的一块画布。在这块画布上绘制的元素被包围在虚构的矩形中，这些矩形称为框(box)。这些框总是平行于画布的边缘放置。使用框属性使人们能够指定单个框的宽度、高度、页边距、贴边、边、大小和位置。图 12-26 显示这些属性之间的关系。

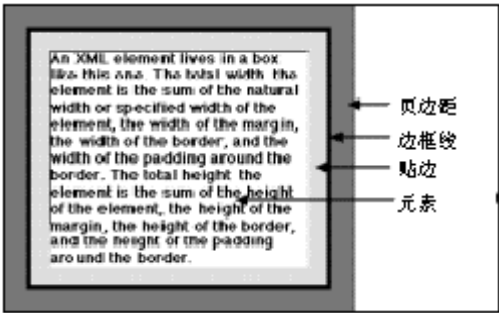


图 12-26 具有页边距、边框线和贴边的 CSS 框

### 12.13.1 页边距属性

页边距属性指定加到框的边框线外面的间距。也可使用 `margin-top`、`margin-bottom`、`margin-right` 和 `margin-left` 分别设置顶端、底、右和左页边距。各页边距可使用绝对长度或父元素宽度大小的百分数来表示。例如，按下面的规则和图 12-27 演示的那样，设置 ACT 的 `margin-top` 属性为 `1ex`，就可以在每个 ACT 元素和其前面的元素之间增加额外的间距。

```
ACT { margin-top: 1ex }
```



图 12-27 ACT 元素的顶端边距变大

也可以使用简略的 `margin` 属性一次设置所有的四个边距。例如，为基本元素(本例为 SYNOPSIS)设置页边距属性，就可以在整个《第十二夜》文档周围增加额外的空间。

```
SYNOPSIS { margin: 1cm 1cm 1cm 1cm }
```

事实上，这与页边距单独使用一个值是一样的，CSS 会将这个值认为可应用于所有四个边上。

```
SYNOPSIS { margin: 1cm }
```

若给出两个 margin 值，则第一个应用于顶端和底部，第二个应用于右和左边。若给出三个 margin 值，则第一个应用于顶端，第二个应用于右和左边，第三个应用于底部。只使用各自的 margin-top、margin-bottom、margin-right 和 margin-left 属性也许更容易。

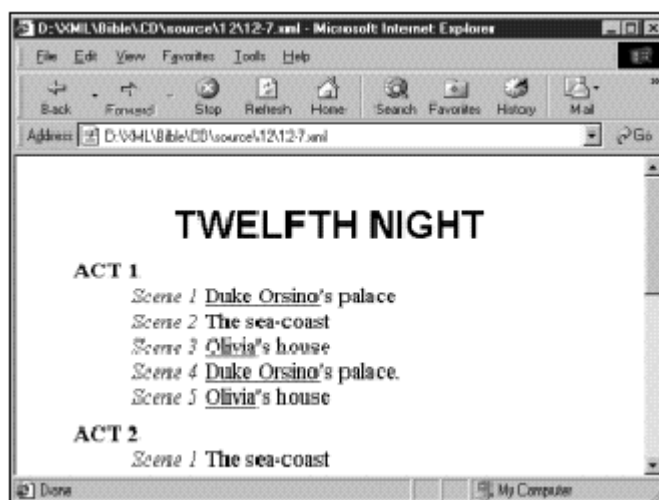


图 12-28 在整个大纲周围增加 1 厘米空间

### 12.13.2 边框线属性

大多数框都有边框。它们是影响基本内容布局的想像中的框，但也许读者无法看见这些框。但是，可以使用边框线属性在其周围绘出线，从而使框可见。边框线属性允许指定边框线的样式、宽度和颜色。

#### 12.13.2.1 边框线样式

缺省条件下，不管边框线的宽度和颜色是否设置，在框的周围不会绘制任何边界。要使某一边界可见，必须把框的 border-style 属性从其缺省值 none 改变为下列值之一：

dotted

dashed

solid

double

groove

ridge

inset

outset

border-style 属性可有一到四个值。正如 margin 属性一样，一个值适用于所有四个边界。使用两个值时，第一个值表示第一样式，用来设置顶和底边界，第二个值表示第二样式，用来设置右和左边界。使用三个值时，则按顺序设置顶、右及左和底边界。当使用四个值时，则表示按从顶、右、底和左的顺序设置每个边界。例如，下列这条规则使用实线边框把整个 SYNOPSIS 围住。图 12-29 是 Internet Explorer 5.0 显示的情景。在这种情况下，边框对使页边距更加明显的效果不太大(记住，页边距在边框之外)。

SYNOPSIS { border-style: solid }



图 12-29 大纲周围的边框



Internet Explorer 5.0 只显示实线边框。其他样式都是作为简单地实线边框绘出。

#### 12.13.2.2 边框线宽度

为了指定沿框的顶端、底部、右和左边缘指定边框线的宽度，共有四个 border-width 属性。它们是：

1. border-top-width
2. border-right-width
3. border-bottom-width
4. border-left-width

每个属性都可以用绝对长度或三个关键字之一：thin、medium 或 thick 来指定。边框线宽度不可为负值，但可以为零。

例如，要使 SYNOPSIS 元素围在 1 个像素宽的实线边框(任何计算机显示器都能显示的最细边框)内，可使用下面的规则来设置这四个属性：

```
SYNOPSIS { border style: solid;
```

```
border-top-width: 1px;
```

```
border-right-width: 1px;
```

```
border-bottom-width: 1px;
```

```
border-left-width: 1px }
```

如果想要把所有的或几个边框设置为相同的宽度，则使用 `border-width` 简略属性更方便。这个属性可有一到四个值。一个值时，设置所有四个边框线宽度。使用两个值时，设置顶和底边为第一个值，右和左边为第二个值。使用三个值时，则表示按顺序设置顶端、右及左和底宽。当使用四个值时，则表示按从顶、右、底和左的顺序设置每个边框。例如，下列的规则与前述的规则是等价的：

```
SYNOPSIS { border-style: solid; border-width: 1px }
```

### 12.13.2.3 边框颜色

`border-color` 属性设置一到四个边框的颜色。只使用一个值，是指设置所有四个边框的颜色。当使用两个值时，表示把顶和底边框设置为第一个颜色，右和左边框设置为第二个颜色。当使用三个值时，表示按顶端、右及左和底边顺序设置。使用四个值时，表示按顶端、右、底和左的顺序设置每个边颜色。有效值是什么公认的颜色名或 RGB 基色值。例如，要把 SYNOPSIS 元素放置在 1 个像素、实线的红边框线内，可使用下面这条规则：

```
SYNOPSIS { border style: solid;
```

```
border-width: 1px;
```

```
border-color: red }
```

由于本书是以黑白形式印刷的，所以我将这张图留给读者自己解决。

### 12.13.2.4 边框线简略属性

有五个简略边框线属性可以使用一条规则同时设置宽度、样式和边框颜色。它们是：

1. `border-top`
2. `border-right`
3. `border-bottom`
4. `border-left`
5. `border`

例如，`border-top` 属性为顶边框提供宽度、样式和颜色。`border-right`、`border-bottom` 和 `border-left` 相类似。省略的属性被设置为父元素的值。例如，图 12-30 显示在每一幕下面有一个 2 像素实线的蓝边框线(如果喜欢可作为水平尺)。要获得这种效果，可使用下面的规则：

```
ACT { border-bottom: 2px solid blue }
```

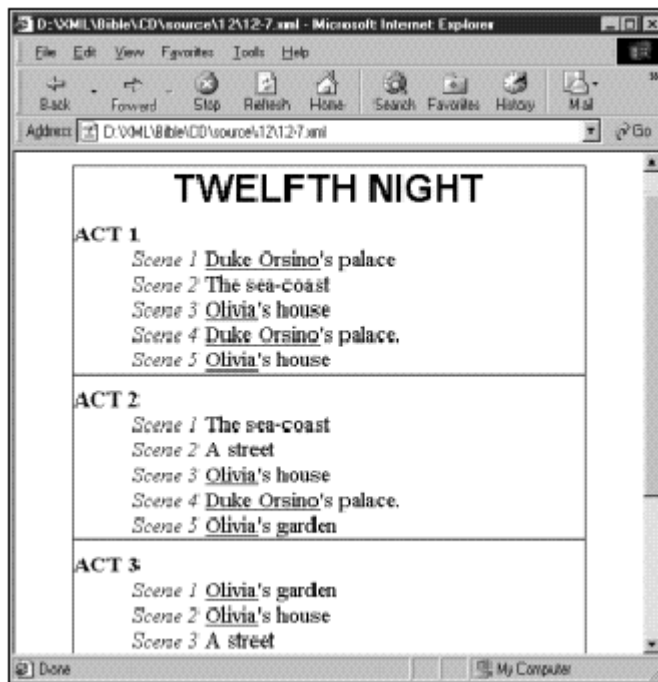


图 12-30 两像素、实线底边框，与 HTML 的 HR 元素相类似

border 属性把所有四个侧边设置成指定的宽度、样式和高度。例如，下列规则在 CHART 元素四周绘出 3 像素宽、实线的红边框。

```
CHART { border: 3pt solid red }
```

### 12.13.3 贴边属性

贴边属性指定框的边界内部间距量。如果显示，框的边框落在页边距和贴边之间。使用 padding-top、padding-bottom、padding-right 和 padding-left 属性，可以对顶、底、右和左贴边分别进行设置。每个贴边都可以用绝对长度或父元素宽度大小的百分数来表示。例如，按下面规则所示来设置其贴边属性，可将 SYNOPSIS 与其边分开。

```
SYNOPSIS { padding-bottom: 1em;
```

```
padding-top: 1em;
```

```
padding-right: 1em;
```

```
padding-left: 1em }
```

还可以使用简略的 padding 属性一次设置所有的四个值。例如，下面的规则与前面的相同：

```
SYNOPSIS { padding: 1em 1em 1em 1em }
```

事实上，这与使用 padding 属性的一个值也是完全一样的，CSS 把此当作适用于所有四个边来看待。

```
SYNOPSIS { padding: 1em }
```

提供两个 padding 值时，第一个应用于顶和底，第二个应用于右和左。提供三个 padding 值时，第一个应用于顶，第二个应用于右和左，第三个应用于底。使用各自的 padding-top、padding-bottom、padding-right 和 padding-left 可能更容易。

在剧本大纲的幕名下面的蓝边似乎靠得太紧，所以我们使用 padding-bottom 属性在剧幕的结尾和边框之间增加 1ex 贴边（如下面的规则所示）。图 12-31 显示的就是这种情景。一般地，在边框周围使用一点贴边是一个很好的主意，这样可以使文本更容易阅读。

```
ACT { padding-bottom: 1ex }
```

#### 12.13.4 大小属性

使用 width 和 height 属性可强制框为给定的大小。框的内容随需要成比例变化，以适应框的要求。尽管可以将此属性应用到文本框上，但更普通、更有用的是将此属性用于像影像和 Java 小程序这样的替代元素。为了指示浏览器应使用真实的大小，其宽度和高度可用绝对长度、父元素高度和宽度的百分数或关键字 auto(缺省值)给出。例如，下面的这条规则把整个 SYNOPSIS 调整为 3 英寸高和 3 英寸宽。

```
SYNOPSIS { padding: 1em; width: 3in; height: 3in }
```



图 12-31 贴边使边框更容易看到

图 12-32 显示的是 Internet Explorer 5.0 中的效果。当面对一个比其框还大的元素时，Internet Explorer 缩小宽度，但扩展其高度。Mozilla 使文本超出框之外，有可能覆盖下面的元素。当内容与有准确大小的框不相适用时，各浏览器的处理结果无法一致、并且也无法预测。



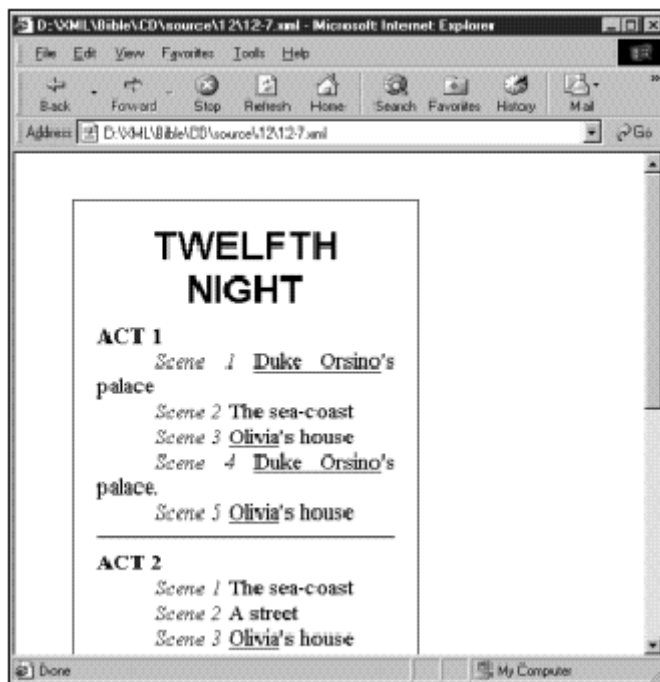


图 12-32 在 Mozilla 中显示的 3 英寸高 3 英寸宽的大纲

如果将 width 设置为绝对或相对单位，将高度设置为 auto，那么高度按照宽度成比例调整。

### 12.13.5 定位属性

在缺省条件下，嵌套在相同父元素内的块级元素在网页上一个挨一个地排列。它们不会并排，也不会折行。可以灵活的使用 float 和 clear 属性改变这种情况。

### 12.13.6 float 属性

float 属性的缺省值为 none，可把它设置为 left 或 right。如果其值为 left，那么元素移到网页的左侧，文本在右边围绕着元素的周围。在 HTML 中，这与使用 ALIGN = "LEFT" 的 IMG 标记的效果一致。如此值为 right，则元素移到网页的右侧，文本在左侧围绕着元素的周围。在 HTML 中，这与使用 ALIGN = "RIGHT" 的 IMG 的标记效果一致。

要在 XML 文件中嵌入影像，没有任何标准方法，所以对于本例，我们用一背景影像，并灵活的使用 CSS 属性来仿造嵌入影像的效果，清单 12-16 是一个稍加修改的舞会请柬，它有一空的 IMAGE 元素。清单 12-17 是一样式单，它将 party.gif 文件设置为 IMAGE 元素的背景。也可以设置 IMAGE 元素的宽度和高度属性。最后，设置 float 为 left。图 12-33 显示运行后的结果。

清单 12-16：有空 IMAGE 元素的舞会请柬

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

```
<?xml stylesheet type="text/css" href="12-17.css"?>
```

```
<INVITATION>
```

```
<IMAGE />
```

```
<TEXT>
```

You re invited to a party on December 31, 1999 to celebrate

the new millennium! You re invited to a party on December 31,  
  
1999 to celebrate the new millennium! You re invited to a  
  
party on December 31, 1999 to celebrate the new millennium!  
  
You re invited to a party on December 31, 1999 to celebrate  
  
the new millennium! You re invited to a party on December 31,  
  
1999 to celebrate the new millennium! You re invited to a  
  
party on December 31, 1999 to celebrate the new millennium!  
  
You re invited to a party on December 31, 1999 to celebrate  
  
the new millennium! You re invited to a party on December 31,  
  
1999 to celebrate the new millennium! You re invited to a  
  
party on December 31, 1999 to celebrate the new millennium!  
  
You re invited to a party on December 31, 1999 to celebrate  
  
the new millennium! You re invited to a party on December 31,  
  
1999 to celebrate the new millennium!

</TEXT>

</INVITATION>

清单 12-17：加载一个 IMAGE 的样式单

```
INVITATION { display:block; }
```

```
IMAGE { background: url(party.gif) no-repeat center center;
```

```
width: 134px;
```

```
height: 196px;
```

```
float: left; }
```

```
TEXT { display: block }
```

### 12.13.7 clear 属性

clear 属性指定元素是否可以在其各边有浮动元素。如果不能，元素将移到任何在此元素之前的浮动元素之下。它与 HTML 中的<BR CLEAR="ALL">元素有关。其可能值为：

none right

left both

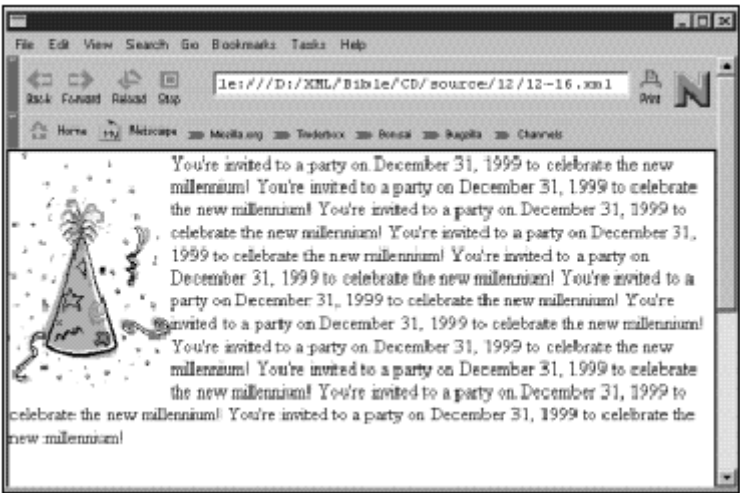


图 12-33 在左边浮动的舞会请柬上的影像

缺省值 `none` 使浮动元素出现在元素的两侧。`left` 值预示浮动元素出现在元素的左侧。`right` 值预示浮动元素出现在元素的右侧。`both` 值预示浮动元素出现在元素的两侧。例如，假设把下列规则加入到清单 12-17 的样式单中：

```
TEXT { clear: left }
```

现在，尽管 `IMAGE` 元素想浮动在 `TEXT` 元素的左边，但 `TEXT` 元素不允许这样做(如图 12-34 所示)。`IMAGE` 元素仍然处于左边，但现在 `TEXT` 被推到影像之下。

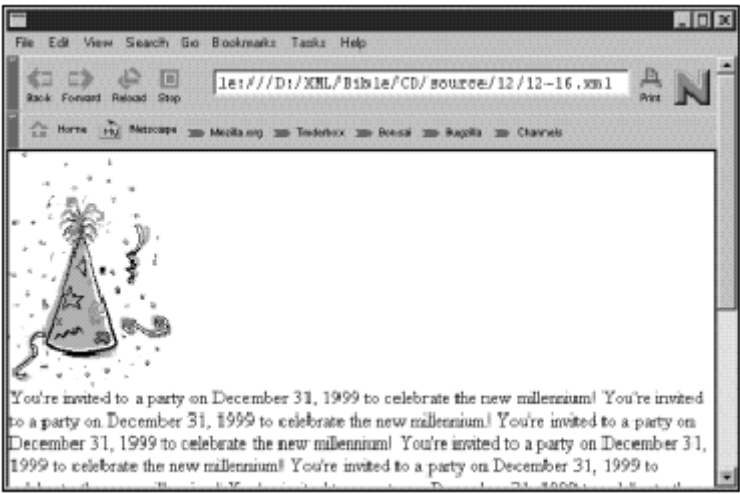


图 12-34 将 `clear` 设置为 `left` 的舞会请柬

## 12.14 本章小结

在本章中，读者已学习了以下内容：

- 对于将样式应用于元素的内容来说，CSS 是一种简单、易懂的语言，它在 HTML 中非常有效，而在 XML 中甚至更加有效。
- 选择符是将规则应用其上的一组以逗号分隔开来的元素。
- CSS 可把多条规则应用于所给类型的元素或带有特定 CLASS 或 ID 特性的元素。
- 许多(尽管不是所有的)CSS 属性都可以被它们所应用的元素的子元素所继承。
- 如果多条规则应用于一个元素，那么格式化属性以预想的方式层叠。
- 在 CSS 样式单中，可使用像 C 语言中的 /\* \*/ 一样的注释。
- 长度可以用相对或绝对单位指定。使用相对单位更好。
- display 属性决定一元素是否是块级、内联或列表项元素。
- 字体属性决定文本字体的外观、样式、大小和粗细。
- 颜色属性可用十进制、十六进制或百分数的 24 位 RGB 空间表示。
- 背景属性包括颜色、影像、影像位置和影像平铺。
- 文本属性可用来调整行距、字间距、字母间距、垂直和水平对齐、修饰和大写。
- 框属性可用来调整网页上元素的相对位置和间距，以及元素周围的边框线。

CSS1 也受到一些限制。首先，CSS1 只能把样式与已经出现在文档中的内容相链接。它无法向文档中添加内容，即使像标点符号这样的内容。此外，它不能把内容以任何方式(如分类或重排)进行变换。这些需要用 XSL(Extensible Style Language，可扩充样式语言)来解决。即使仅仅从格式化内容的角度来考虑，CSS1 所提供的也比人们需要的少。最明显的是，不支持表格。还有其他非明显的缺点。CSS1 不能处理从右到左的文本(如希伯来语、阿拉伯语)或垂直文本(如繁体中文)。在下一章中，我们将学习 CSS2，它可以解决这些问题和 CSS1 的限制。

## 第 13 章 级联样式单级别 2

级联样式单级别 2 (CSS2) 规范由 W3C 在 1998 年出版，它优于 CSS1，对 XML 和 HTML 文档格式化的功能比以前更强大。当然，与 CSS1 一样，CSS2 也对 HTML 向后兼容。但是，使用 XML，CSS2 几乎和桌面出版程序(如 PageMaker 和 Quark XPress)一样，可格式化纸张和 Web 上的内容。

这里讨论的大多数规则，常用的浏览器仍不能执行。Mozilla 可实现一些样式，但完全能够实现所有的样式仍有一段路要走。

本章的主要内容如下：

- CSS2 中有哪些新特点？
- 选择元素
- 格式化网页
- 可视的格式化
- 框
- 计数器和自动计数
- 音频样式单

### 13.1 CSS2 中有哪些新特点？

CSS2 体现了 Web 开发者和设计者对浏览器开发商长期以来一直要求的许多特点。其规范也比 CSS1 多两倍，不仅仅是所有改变的综合和具有一些新特点，而且重写了原来的规范。这使此规范成为所有级联样式单句法、语义和规则的唯一来源。



完整的 CSS2 规范可在 Web 上的 <http://www.w3.org/TR/REC-CSS2> 和本书所附光盘上的 specs/CSS2 文件夹中。这也许是迄今 W3C 创作的最易读的规范文档，并且也值得反复读。

要使用通用的软件完全支持所有的新规范，还需要花费一些时间，CSS2 也不例外。正如读完本章之后所看到的那样，Internet Explorer 5.0 和 Mozilla 也刚刚开始实现这些属性。为了读者的方便，对那些仍没有实现的属性，我将给予指出。

CSS2 的许多新特点能使人们更加准确地选择和格式化文档中的元素。新的伪类和伪元素能够选择元素的第一个子元素，当接收到焦点时调整元素，或者在指定元素选定范围内自动地控制其他元素的位置。媒体类型可把不同的样式应用于在不同媒体（如打印的页面、计算机显示和无线电广播）上出现的文档。对分页媒体（如打印输出和幻灯片显示）的支持也获得了极大地改进，从而可对页面的分页符（page break）实现更加强大的控制。现在不仅可以对块和内联框中的元素格式化，也能对表格中的元素格式化。可自动地对顺序和列表加以编号和缩进排印。对非西方语言（如阿拉伯语和中文）提供了更多的支持。并且，第一次应用听觉样式来指定文档不是以何种方式显示，而是如何让人阅读。此外，CSS2 改变了 CSS1 一些功能的实现方式。

### 13.1.1 新的伪类

伪类选择具有共同部分但不一定是相同类型的元素。例如，`:hover` 伪类指明是光标当前正在其上的那个元素，而不管此元素的类型。CSS2 有七个新的伪类，现简要地叙述如下：

- `:first-child`: `:first-child` 伪类选择元素的第一个子元素。
- `:focus`: `:focus` 伪类选择有焦点的对象；也就是说，如果用户在键盘上击一个键，那么此键就输入到该对象之中。
- `:hover`: `:hover` 伪类选择一个指派的、但非激活的对象。
- `:lang`: `:lang` 伪类选择以特定的语言编写、由 `xml:lang` 特性标识的元素。
- `:first`: `:first` 伪类选择要打印文档的首页。
- `:left`: `:left` 伪类选择文档打印输出的左页（通常是偶数页），就像复印材料装订成书那样。
- `:right`: `:right` 伪类选择文档打印输出的右页（通常是奇数页），就像复印材料装订成书那样。

### 13.1.2 新的伪元素

伪元素通过从 XML 输入中不易获得的其他信息来标识特定的元素。例如，在 CSS1 和 CSS2 中，`:first-line` 和 `:first-letter` 就是伪元素，它们选择元素的第一行和首字母，甚至它们不必用任何元素来表示。

CSS2 添加了两个新的伪元素：`:after` 和 `:before`。`:after` 伪元素可用来在指定的元素后面插入对象。这些对象可以是影像、自动计数器或正文。`:before` 伪元素可用来在指定的元素之前插入对象。这些对象也可以是影像、自动计数器或正文。

### 13.1.3 媒体类型

CSS2 为盲文、计算机显示器、纸张上的墨水和电视所呈现的信息定义了十种媒体类型。CSS2 可以为不同的媒体指定不同的样式。例如，对于低分辨率计算机显示器来说，使用大字体比 1200dpi 打印更重要。

### 13.1.4 分页媒体

CSS2 控制页面的分页符，并提供标识文档中各页面的方法，以便设计者可格式化打印文档，而不影响文档在屏幕上的外观。

### 13.1.5 国际化

由于 Internet 扩展到非英语的国家，所以在支持成千上万的目前正在使用的和历史上的口语和书面语方面，正取得更大的进步。CSS2 增加了对 Unicode 和双向正文的支持，所以为中文和希伯来文施加样式就尤如英语和法语一样容易。

### 13.1.6 可视格式化控制

CSS2 增加了更多的格式化属性，以提供对组成文档的对象更精确的控制。现在可指定元素的绝对位置和尺寸。当创建元素时，还有更多的显示样式可供使用。可以将阴影应用于正文。字体和颜色也可以指定为“同等于”用户界面的元素，就像菜单项或图标标签那样。当光标的指针移动到不同的元素上时，可改变光标的显示形式。

### 13.1.7 表格

display 属性的改进使得把 XML 元素当作类似表格的结构一样处理，并在更好地控制它们的对齐方式方面变得容易。

### 13.1.8 生成的内容

当文档正在显示时，自动生成的计数器、编号系统和列表标记符等使文档的作者强制应用程序能瞬间创建信息。无论在何时文档发生变化时，编号都会瞬间重新计算，而不是艰难地手工插入。

### 13.1.9 有声样式单

为了努力地使信息友好地分发给所有的人，CSS2 中加入了具有语音合成系统功能的特殊属性。这些属性使文档作者能控制文档内每个元素的声音的强度、语调和其他属性。

### 13.1.10 新工具

CSS2 规范也改变了原先包括在 CSS1 中的一些功能。它们包括级联机制、伪类和其他的各种属性。

#### 13.1.10.1 伪类和伪元素

:link、:visited 和:active 伪类不再非得独立地相互指派不可，可一起使用。

#### 13.1.10.2 继承

在 CSS1 中，只有几个属性能够从其父元素那里继承属性值。而在 CSS2 中，将属性值设置为关键字 inherit，此时，所有的属性都能继承其父元素的值。当一属性被继承时，属性就具有与最邻近的父元素一样的值。



由于每个属性都可有 inherit 值，所以在下面对各属性的讨论中，我省略对此值的任何解释。

#### 13.1.10.3 级联机制

在 CSS1 中，!important 指定符可强制作者样式单优先于读者样式单。CSS2 则相反，所以读者选项优先于作者的选项。当作者和读者的样式单一起使用时，缺省结果是，用户样式单覆盖作者样式单。但是，如果作者将一属性声明!important，这就向规范中增加更大的强制力，使它覆盖读者样式单。但是，如果读者也将一条规则声明!important，这也将覆盖作者样式单中的!important 声明。换句话说，读者获得最终的发言权。

#### 13.1.10.4 显示属性

`display` 属性的缺省值现在是 `inline`，而不是 `block`。

#### 13.1.10.5 页边距和贴边

在 CSS1 中，当设置其他属性时，会忽略一些页边距属性，例如，如果同时设置了 `margin-left` 和 `width`，那么 `margin-right` 将被忽略。此结果不依赖于正文的方向和对象的对齐方式。CSS2 根据对象正文的方向来决定如何改变左或右页边距。

## 13.2 选择元素

支持 CSS2 的浏览器（如 Internet Explorer 和 Mozilla）可以更明确地选择将样式规则应用于其上的元素或对象。使用 CSS2，只需通过指定元素名、标识符或将元素和属性设置组合在一起，就可以根据元素在文档结构中创建的式样来选择元素。

### 13.2.1 式样匹配

CSS2 式样匹配可标识文档树状结构中特定的元素。式样匹配选择符的句法可以是从一个简单的元素名到一复杂的上下文式样的系统（如表 13-1 所示）。如果元素满足指定式样的所有要求，则此元素就与该式样匹配。在 XML 中，对大小写是敏感的。

表 13-1 用于式样匹配的 CSS2 选择符句法

句法	意义
*	这是通配符（universal selector），匹配任何元素
X	与名称为 X 的任何元素匹配
XY	匹配带有 Y 名（是带有 X 名的元素派生的）的任何元素。例如，SONNET 元素的所有 VERSE 派生元素
X>Y	匹配任何为 X 元素的子元素的 Y 元素。例如，STANZA 元素的所有 VERSE 子元素
X:first-child	匹配所有的 X 元素（为其父元素的第一个子元素）。例如，在 SONNET 元素中的第一个 STANZA 元素
X:link	匹配链接中其目标未被访问的所有的 X 元素
X:visited	匹配链接中其目标已被访问的所有的 X 元素
X:active	匹配当前选择的所有 X 元素
X:hover	匹配当前有鼠标在其上方移动的所有 X 元素
X:focus	匹配当前通过选择鼠标或准备输入文本数据而获得用户焦点的所有 X 元素
X:lang( <i>i</i> )	匹配应用 xml:lang 特性指定使用人类语言 <i>i</i> 的所有 X 元素
X+Y	匹配其近系同属为 X 元素的所有 Y 元素。例如，REFRAIN 元素就紧接在 STANZA 前面
X[attr]	匹配设置了 attr 特性的所有 X 元素，而不管此特性为何值。例如，带有 NAME 特性的 AUTHOR 元素
X[attr="string"]	匹配其 attr 特性的值为 “string” 的所有 X 元素。例如，值为 19990723 的 DATA 特性的 AUTHOR 元素



X[attr~="string"]	匹配任何 X 元素，但此元素的 attr 特性是以空格分开的、其中之一为“string”的一组单词
X[lang =“langcode”]	匹配其 lang 特性设置为特定的“langcode”的所有 X 元素
X#myname	匹配其 id 特性为“myname”值的任何 X 元素

13.2.2 通配符

\*符号选择文档中的所有元素。用它能为所有的元素设置缺省样式。例如，下面这条规则将缺省字体设置成 New York：


```
*{ font-face: “New York” }
```

可以将\*和特性、伪类和伪元素选择符结合起来，以便把样式应用于带有特定特性、特性值和作用等的元素，从而使这些元素具有指定特性、特性值、角色等。例如：

```
*:before { content: “. “ counter(pgraph) “. “;

counter-increment: pgraph; /*向 pgraph 加 1*/

*[onmouseover] { text-decoration: blink }
```



如果通配符只与一个属性说明一起使用，那么\*就可以忽略。

例如：

```
before { content: “. “ counter(pgraph) “. “;

counter-increment: para }

[onmouseover] { text-decoration: blink }
```

13.2.3 后代和子代选择符

可以使用子代（child）或后代（descendant）选择符来选择指定类型元素的子代（children）或派生代（descendent）的元素。例如，可以选择包含在 SONNET 元素中的任何 VERSE 元素，或成为 STANZA 元素直系子代的 VERSE 元素。参见清单 13-1，显示的是以 XML 格式写成的 Shakespeare（莎士比亚）的第 21 首的十四行诗。

清单 13-1： Shakespeare 的第 21 首的十四行诗

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

<?xml stylesheet type=" text/css" href=" shakespeare.css" ?>

<SONNET>

<AUTHOR>William Shakespeare</AUTHOR>

<TITLE>Sonnet 21</TITLE>
```

<STANZA id=" st1" >

<VERSE>SO is it not with me as with that Muse</VERSE>

<VERSE>Stirr' d by a painted beauty to his verse,</VERSE>

<VERSE>Who heaven itself for ornament doth use</VERSE>

<VERSE>And every fair with his fair doth rehearse:</VERSE>

</STANZA>

<STANZA id=" st2" >

<VERSE>Making a couplement of proud compare</VERSE>

<VERSE>With sun and moon, with earth and sea' s rich  
gems,</VERSE>

<VERSE>With April' s first born flowers, and all things  
rare</VERSE>

<VERSE>That heavens air in this huge rondure hems.</VERSE>

</STANZA>

<STANZA id=" st3" >

<VERSE>O, let me, true in love, but truly write,</VERSE>

<VERSE>And then believe me, my love is as fair</VERSE>

<VERSE>As any mother' s child, though not so bright</VERSE>

<VERSE>As those gold candles fix' d in heaven' s air.</VERSE>

</STANZA>

<REFRAIN>

<VERSE>Let them say more that like of hearsay well,</VERSE>

<VERSE>L will not praise that purpose not to sell.</VERSE>

</REFRAIN>

</SONNET>

所有的 VERSE 都是 SONNET 元素的后代，但不是直系子代（immediate children）。有些 VERSE 元素是 STANZA 的直系子代，有些则是 REFRAIN 元素的直系子代。后代选择符是由一个空格分开的两个或更多个元素标志符组成的。SONNET VERSE 形

式的后代选择符匹配 VERSE 元素，后者是 SONNET 元素的任意后代。为了指定后代的特定层次，需要使用 SONNET\*VERSE 形式，它强制 VERSE 元素至少应为孙代，或为 SONNET 元素更低的后代。

要指定直系子代元素，可使用 STANZA>VERSE 的形式。这种形式把规则只应用于成为 STANZA 元素直系子代的 VERSE 元素，所以不影响 REFRAIN 元素的任何 VERSE 子代。

可以把后代和子代选择符结合起来，以查找特定的元素。例如，下列选择符查找属于 REFRAIN 元素第一个子代的所有 VERSE 元素，而 REFRAIN 元素又是 SONNET 元素的后代。

```
SONNET REFRAIN>VERSE:first { padding: "2cm" }
```

将上述规则应用于清单 13-1，则此规则选择诗句“Let them say more that like of hearsay well,”。

#### 13.2.4 直系同属选择符

直系同属选择符在元素指示符之间使用加号 (+) 来识别与其后的另一元素处于同一层次的元素。例如，下面的代码选择与 STANZA 元素共享一个父元素并紧接在 STANZA 元素之后的所有 REFRAIN 元素。

```
STANZA+REFRAIN { color: red }
```

#### 13.2.5 特性选择符

特性选择符标识特定的元素/特性的组合。把要匹配的特性名用方括号括起来放在元素名之后。例如，下面的规则将带有 NUMBER 特性的所有 STANZA 元素都变成红色：

```
STANZA[NUMBER] { color:red }
```

此规则将具有 NUMBER 特性的所有 STANZA 元素都变成红色，而不管特性值是什么。它包括由 DTD 提供的具有缺省 NUMBER 特性的元素，但不包括没有 NUMBER 特性的 STANZA 元素。

要测试特性值，可使用设置特性值的相同句法；也就是说，等于号放在名称后面，值放在等于号后面，并放在引号中。例如，仅指定其 NUMBER 特性值为 3 的 STANZA 元素变为红色，则可使用下面这条规则：

```
STANZA [NUMBER="3"] { color:red }
```

#### 13.2.6 @规则

@规则用来完成某项任务，而不是选择元素，并把一些样式应用于这一元素。@规则有五种：

1. @page：把样式应用于页面（而不是此页上的元素）
2. @import：在当前样式单中嵌入一外部样式单
3. @media：把只能用于某种媒体的特性组合起来成为样式规则
4. @font-face：描述样式单中用于其他地方的字体
5. @charset：定义样式单使用的字符集

##### 13.2.6.1 @page

@page 规则选择页面框。在其内部，设计者可指定各页面的大小、版面布局、取向和页边距。页面框是矩形区域，大约为打印页的大小，它包含页面区域和页边距。页面区域包括要显示的内容，框边缘作为一种容器，页面版面就处于分页符之间。与其他框不同，页面框没有边界线或贴边，只有页边距。

@page 规则选择文档中的每一页面。可使用下面的页面伪类属性之一: `first`、`:left` 或 `:right` 来为各页面类指定不同属性。

由于@page 规则不知道含有字体的页面内容，所以无法理解以 `em` 和 `ex` 为单位的尺寸。所有的其他度量单位（包括百分数）都是可以接受的。用于设置页边距的百分数也是总页面框的百分数。页边距可为负值，表示把内容放在通常应用程序或打印机可访问的区域之外。在大多数情况下，只保留可见或可打印区域内的信息。

### 13.2.6.2 @import

@import 规则把指定的外部样式单嵌入到现有的样式单中。这样可以根据多个较小、较容易理解的片断生成大样式单。导入的样式单使用 `.css` 扩展名。例如，下面的规则导入 `poetry.css` 文件。

```
@import url(poetry.css);
```

@import 规则可在样式单后面指定媒体类型。如果没有指定媒体类型，@import 规则就没有限制，并且可用于所有媒体类型。例如，下列规则导入 `printmedia.css` 文件。在这个样式单中的声明只适用于印刷媒体(`print media`)。

```
@import url(printmedia.css) print;
```

下面的这条规则导入 `continuous.css` 文件，可用于计算机显示器或电视机显示器。

```
@import url(continuous.css) tv, screen;
```

导入到其他样式单中的样式单，在级联中的等级要比导入它的样式单低。例如，假设 `shakespeare.css` 为 VERSE 指定 New York 字体，而 `shakeprint.css` 为 VERSE 指定 Times 字体。如果将 `Shakespeare.css` 导入到 `shakeprint.css` 中，那么，诗句将以 Times 字体显示。

### 13.2.6.3 @media

可使用多种媒体把信息传递给读者，每种媒体都有其自己的习惯样式和格式。现在还不能使语音合成器较好地以单音调来阅读 Shakespeare 的诗句，如今可以吗？斜体字对于等宽终端也没有什么意义。

CSS2 可为显示在不同媒体中的相同元素指定不同的样式。例如，如果正文使用的是非衬线字体，则在屏幕上就更容易阅读，而如果正文是以衬线字体编写在纸上时，通常最容易阅读。可以将只准备用于一种媒体的多个样式规则放入一条指明媒体名的@media 规则中。在一篇文档中，@media 规则的数量与指定的媒体类型一样多。例如，下面的这些规则将根据是在纸上打印还是在显示器上显示把 SONNET 元素格式化不同的样式。

```
@media print {
```

```
SONNET { font-size: 10pt; font family: Times, serif }
```

```
}
```

```
@media screen {
```

```
SONNET { font-size: 12pt;
```

```
font-family: New York, Times New Roman, serif }
```

```
}
```

```
@media screen, print {
```

```
VERSE { line-height: 1.2 }
```

前两条规则明确定义了为打印机和屏幕媒体类型所使用的样式。由于现在的计算机显示器的分辨率比现在的打印机低得多，所有显示在屏幕上的字体比打印输出要大，并且选择适用于屏幕的字体，这是很重要的。

第三条规则提供适用于这两种媒体类型的样式。要为多个媒体类型同时指定样式指令，可简单地将媒体名列在 @media 规则指定符之后，并以逗号分开。

支持 CSS2 的浏览器允许文档的作者提供决定特定类型媒体如何显示文档的规则。例如，当在屏幕上显示一文档时，与把它发送到打印机相比，很可能应用不同的规则。CSS2 识别如下十种媒体类型：

1. all：所有的设备
2. aural（连续、有听）：语音合成器
3. braille（连续、可触知）：用于有视觉障碍的盲文触觉反馈设备
4. embossed（分页、可触摸）：分页盲文打印机
5. handheld（可视）：PDA（手持机）和其他手持式设备如 Windows CE 掌上型电脑、Newton 和 Palm Pilot
6. print（分页、可视）：所有的打印、不透明材料
7. projection（分页、可视）：展示和幻灯片放映，可将它们直接从计算机上投影或打印在幻灯片上
8. screen（连续、可视）：点位图彩色计算机显示器
9. tty（连续、可视）：使用位置固定的、单色字符栅格的哑终端和旧的 PC 显示器
10. tv（可听/可视）：电视类设备，如低分辨率、模拟显示器、彩色设备

浏览器软件不一定支持所有的这些类型。实际上，我知道没有任何一个设备支持所有的这些类型。但是，样式单设计者应该假设读者可能使用任何一类或所有类型的设备来浏览自己的内容。

当然，各个媒体的特性也会随着时间的流逝而改变。我的第一个打印机是 144dpi，可是，在 21 世纪，如此低分辨率的打印机就会很少见。另一方面，显示器最终也将达到 300dpi 或更高；而彩色打印机也迅速地为用户所使用。

有些属性只有特定的媒体类型才可用到。例如，pitch 属性只用于有声媒体类型。尽管 CSS2 的确为 @media 规则提供一组当前值，但它未指定所有可包括的媒体类型。给出的媒体名与大小写无关。

#### 13.2.6.4 @font-face

@font-face 规则用来描述样式单中其他地方使用的字样。可提供字体名、URL（下载字体的位置）和有关字体点阵（允许适度的精确复制来达到合成的目的）的详细信息。@font-face 规则还控制了软件如

何根据作者指定的字体来为文档选择字体。可想到的方法包括同等字体匹配、智能字体匹配、合成所需字体、下载服务器上的字体或形成字体。这些方法将在下面描述。

- **同等字体匹配 (Identical Font Matching):** 用户软件选择相同字族名的本地系统字体。相同名称的字体在外观上不一定完全一致。客户端使用的字体来源可能与服务器上的字体不同。
- **智能字体匹配 (Intelligent Font Matching):** 软件选择客户端系统上现有的字体, 并且其外观与所要求的字体最接近。不要求精确地匹配, 而应接近。这是根据字型、是否使用衬线、粗细、大写字母的高度以及其他字体的特性来匹配的。
- **字体合成 (Font Synthesis):** Web 浏览器生成与指定字体最相似的字体, 并共享其点阵。当合成一字体时, 通常它的近似程序比使用匹配找到的字体更接近。为了使所有的字体特性都能够保留, 这种合成需要精确的代换和位置信息。
- **字体下载 (Font Download):** 浏览器软件从指定的 URL 处下载字体。这个过程与下载同当前文档一起显示的影像或声音是完全一样的。下载字体的用户会经历一段等待, 这与下载影像时的情况相似。
- **字体形成 (Font Rendering):** 最后可供管理字体的选择办法是渐进形成。这是一种下载和匹配的组合, 它能使浏览器创建临时的字体, 可以一边下载原字体, 一边阅读文档内容。在“真正”的字体下载后, 用它来代替生成文档中的合成字体。为了避免文档显示两次, 字体描述必须包含描述字体的点阵信息。字体的点阵信息越完整, 那么一旦下载完成后, 文档需要重新形成的可能性就越小。

CSS2 能使文档淖髡咧付彳 诶亩料低趁挥兄付 ū 淖痔迨笔褂媚闹址椒 ä 匚缙 械幕埃┐≡褙痔滩

font-face 规则提供字体的描述。这种字体的描述是由一系列字体描述符创建的，并定义了有关页面上使用的字体的详细信息，可包括用于字体的 URL、字族名和字号。

字体描述符分为下列三类：

- 提供字体的样式单用法与其描述之间的链接。
- 提供字体的位置或其相关信息的 URL。
- 提供字体的字符信息。

@font-face 规则只应用于样式单内部指定的字体。在样式单中，对每种字体，都需一个@font-face 说明。例如：

```
@font-face { font-family: "Comic Sans";

src: url(http://metalab.unc.edu/XML/fonts/comicsans))

@font-face { font-family: "Jester"; font-weight: bold;

font-style: italic)

TITLF { font-family: "Comic Sans")

AUTHOR { font-family: "Jester", serif }
```

当软件读到此样式单时，将试图找到指定各元素应如何显示的一组规则。此样式单将所有的 TITLE 元素设置为 Comic Sans 字族，同时它又把所有的 AUTHOR 元素设置为 Jester 字体。支持 CSS1 的 Web 浏览应用程序搜索 Comic Sans 和 Jester 字族。如果找到这些字族，浏览应用程序将其缺省正文字体设为 Comic 字族，将 serif 字体指定为 Jester 字族的后略字体。@font-face 规则的字体描述符将被忽略。CSS1 软件能安全地跳过这个命令，而不会出现错误。

支持 CSS2 的应用程序将检查@font-face 规则，以试图匹配 Comic Sans 和 Jester 字体的描述。在上一例子中，浏览软件找到了 URL，从此处可下载 Comic Sans 字体。如果在客户端系统上找到 Comic Sans 字体，软件就会使用它来代替下载的字体。对于 Jester 情况，用户软件将使用匹配规则或合成规则从所提供的描述符来创建一类似的字体。如果 Web 浏览器没有为指定的字族找到相匹配的@font-face 规则的话，那它会试图使用为 CSS1 指定的规则来匹配字体。

CSS2 可以跳过浏览器不能识别或无用的任何字体描述符。这就提供了内建的增加描述符的方法以便改善字体置换、匹配或合成所使用的规则。

### 13.2.6.5 @charset

指定编写样式单的字符集有三种方式，并且以如下的顺序选择优先级：

1. Content-Type 字段中的 HTTP “charset” 参数
2. @charset 规则
3. 与文档相关联的特性和属性，如与 LINK 元素一起使用的 HTML 的 charset 特性每个样式单都包含一个@charset 规则。@charset 规则必须出现在文档的最前，前面不能有任何其他字符。使用@charset 的句法为：

@charset “character set name”

本语句中指定的 character set name（字符集名称）必须是 IANA 注册表中描述的名称。在第 7 章的表 7-7 中列出了部分字符集。要指定样式单用 Latin-1 字体来编写，可写为下列形式：

@charset “ISO 8859-1”



在第 7 章 “外国语言和非罗马文字” 中，详细地讨论了字符集。

### 13.2.7 伪元素

在 XML 文档中，伪元素在样式单中是作为元素来处理的，但不一定是特定的元素。它们是应用样式单之后显示文档的某一部分（如一段的第一行）。伪元素区分大小写，并直接出现在样式单选择符的主题之后。CSS2 引入了两个新的伪元素：:after 和:before。

:after 和:before 伪元素选择在它们之前的元素的紧前面和紧后面的位置。content 属性用来把数据放在这个位置。例如，下面的这条规则将字符串&#0;&#0;&#0;&#0;放在 STANZA 对象之间，以便将节分开。字符串文字中的\\A 为分行符的编码：

STANZA: after { content: “\\A&#0;&#0;&#0;&#0;\\A” }

除了文字字符串之外，也可使用下列四个关键字之一作为 content 属性的值：

1. open-quote
2. close-quote
3. no-open-quote

#### 4. no-close-quote

open-quote 和 close-quote 关键字为当前语言和字体插入适当的引号字符(如" 或 ')。no-open-quote 和 no-close-quote 关键字不插入任何字符,但增加嵌套的层次,就像使用引号一样。根据每个嵌套的层次,引号标记从双引号到单引号之间切换,反之亦然。

还可以使用 attr(X) 函数作为内容属性的值,将 X 特性的值插入到标识的元素之前或之后。

最后,也可以使用 counter() 或 counters() 函数,插入自动计数器的当前值。有两种截然不同的形式: counter(name) 或 counter(name, style)。其中缺省的 style 参数为十进制。

### 13.2.8 伪类

伪类选择符基于外观而不是元素的名称、特性或内容来选择元素。例如,某一伪类可以基于鼠标的位置、获得焦点的对象或是否是链接对象。当读者与文档产生交互时,元素可不断改变其伪类。某些伪类是互斥的,但大多数能同时应用于同一个元素,并能放在元素选择符内的任何地方。当伪类的确发生冲突时,级联顺序确定激活哪个规则。

#### 13.2.8.1 :first-child

:first-child 伪类选择命名元素的第一个子元素,而不管其类型。例如在清单 13-1 中,VERSE 元素的内容是 "So is it not with me as with that Muse", 此元素是 STANZA 元素的第一个子元素,可由下列规则指定:

```
STANZA: first-child { font-style: bold }
```

#### 13.2.8.2 :link、:visited、:active

在 CSS1 中,:link、:visited 和 :active 伪类是互斥的。在 CSS2 中,:link 和 :visited 也是互斥的(在逻辑上不得不如此),但可将两者中一个与 :active 一起使用。例如,下列的代码段假定 AUTHOR 元素已指定为链接,并根据此链接的当前状态改变正文的颜色。在下面的代码段中,当鼠标正放在链接的上面时,将未被访问的链接设置为红色,已访问过的链接作为灰色显示,活动链接以橙绿色显示。

```
AUTHOR: link { color: "red" }
```

```
AUTHOR: visited { color: "gray" }
```

```
AUTHOR: active { color: "lime" }
```

#### 13.2.8.3 :hover

:hover 伪类选择鼠标或其他指示设备正指着元素,但不按下鼠标键。例如,下面的这条规则是在鼠标指向 AUTHOR 元素时,将此元素变成红色。

```
AUTHOR: hover { color: "red" }
```

当鼠标不再指向 AUTHOR 元素时,此元素返回到正常颜色。

#### 13.2.8.4 :focus

:focus 伪类引用当前获得焦点的元素。当选择了某一元素,并准备接收某种文本输入时,此元素就获得了焦点。下列的规则使焦点的元素变成黑体。



```
:focus { text-style: "bold" }
```

#### 13.2.8.5 :lang()

:lang() 伪类选择使用指定语言的元素。为此，在 XML 中，一般通过 XML 声明中的 xml:lang 特性和/或 encoding 特性来实现的。下列规则改变用希伯来语编写的所有的 VERSE 元素的方向，以便从右往左阅读，而不是从左往右：

```
VERSE: lang(he) { direction: "rtl" }
```

#### 13.2.8.6 :right、:left、:first

:right、:left 和 :first 伪类只适用于 @page 规则。可用它们来为文档的第一页、文档的左（一般为偶数）页、文档的右（一般为奇数）页指定不同的样式。例如，下面的这些规则指定很大的页边距：

```
@page: right { margin-top: 5cm;
```

```
margin-bottom: 5cm;
```

```
margin-left: 7cm;
```

```
margin-right: 5cm }
```

```
@page: left { margin-top: 5cm;
```

```
margin-bottom: 5cm;
```

```
margin-left: 5cm;
```

```
margin-right: 7cm }
```

```
@page: first { margin top: 10cm;
```

```
margin-bottom: 10cm;
```

```
margin-left: 10cm;
```

```
margin-right: 10cm }
```

在用于伪类的规则中，唯一能设置的属性就是页边距属性。

## 13.3 格式化页面

@page 选择符指的是页面。可用它来设置应用于页面而不是页面上的各个 XML 元素的属性。文档的每一页都有多种属性可供应用，包括页面的大小、取向、页边距和分页符。这些属性级联于页面上的任何元素。可选的伪类可为第一页、右页和左页指定不同的属性。

CSS2 合理地假定页面是矩形的。作出这种假定之后，页面可具有框的属性，这些属性以及包括页边距和大小在内，在 CSS1 中就已熟悉了。但是页面框没有边界或贴边，因为这些已经跑到物理页面之外了。

### 13.3.1 大小属性

在 @page 规则中，size 属性指定页面的高度和宽度。可将 size 设置为 1 或 2 个绝对长度，或下列四个关键字之一：auto、portrait、landscape 或 inherit。如果只给出一个长度，则此页面将是正方形的。当给出两个尺寸时，第一个是此页面的宽度；第二个是高度。例如，

```
@page { size: 8.5in 11in }
```

auto 设置自动调整为目标屏幕或纸面的大小。landscape 强制文档格式化为适应目标页面，但长边是水平的。portrait 设置将文档格式化为适应缺省的目标页面的大小，但长边是垂直的。

### 13.3.2 页边距属性

margin 属性控制着此页的页边距，页边距为页面的四个侧边上的所有不能打印的矩形区域。此属性可作为分别设置 margin-top、margin-bottom、margin-right 和 margin-left 的简略方式。而这些属性与 CSS1 中的框的属性是相同的。例如，下面的规则描述这样的页面：长 11 英寸、宽为 8.5 英寸、所有的侧面上的页边距为 1 英寸。

```
@page { size: 8.5in 11in; margin: 1.0in }
```

### 13.3.3 标记属性

CSS2 提供了 mark 属性，以便将标记（用于描绘在何处切开纸张和如何对齐页面）放在页面上。这些标记（mark）显示在页面框之外。页面框只是文档的可视区域，它受 @page 规则的影响。如果有一张已打印的区域为 8.5 英寸×11 英寸的打印纸，那么页面框就是此张纸上可打印区域内的内容，我们通常把它认为是打印机页边距内的空间。软件控制标记的显示，它只显示于绝对的页面框上。绝对的页面框不能移动，可为页面的一般页边距所控制。相对页面框可以根据目标页面对齐，在大多数情况下，强制标记离开页面的边缘。当将相对页面框进行对齐时，实际上是用内心的眼睛在看页面，并使用 margin 和 padding 属性将此页面的打印区域在实际的纸张上移动。

mark 属性有四个值：crop、cross、inherit 和 none，只能和 @page 元素一起使用。修剪（crop）标记标识纸张的剪切边缘。交叉（cross）也称为注册标记，它用于在打印的内容之后对齐页面。如果设置为 none，文档中没有任何标记。下列的规则指定带有 crop 和 cross 标记的页面：

```
@page { mark: crop cross }
```

### 13.3.4 页面属性

除了使用 @page 选择符来指定页面属性之外，还可使用 page 属性来将页面属性与各个元素进行连接。为此，可编写指定页面属性的 @page 规则，给 @page 规则命名，然后使用这个名称作为正常元素规则的 page 属性。例如，下面两条规则说明打印 SONNET 的纸是横向放置的。

```
@page rotated { size: landscape }
```

```
SONNET { page: rotated }
```

使用 page 属性时，使用不同的同属元素指定不同的页面属性是可能的。如果是这样，将在元素之间插入一分页符。如果子元素使用了不同于父元素的页面布局，那么此子元素的版式将处于优先地位。例如，在下例中，这两个表横向显示在页面中，如果空间允许，有可能在同一页上。由于文档中的元素层次的关系，赋给 SONNET 元素的旋转页面不再有效，且不被使用。

```
@page narrow { size: 9cm 18cm }
```

```
@page rotated { size: landscaper }
```

```
STANZA { page: narrow }
```

```
SONNET { page: rotated }
```

### 13.3.5 分页符属性

page-break-after 属性强制或阻止在当前对象后插入分页符。page-break-before 属性强制或阻止在当前对象前插入分页符。page-break-inside 属性允许或阻止在当前对象内部插入分页符。这些属性可用来将相关的正文段落、标题及其主体文本、影像及其说明放在一起，或在同一页中保持表格的完整。

当将这些属性的任一个设置为 auto 时，在当前框中既不会强制，也不会禁止插入分页符。设置为 always 时，强制插入分页符。avoid 设置将阻止分页符出现。设置为 left 和 right 时，根据需要强制插入一或两个分页符，以便强制下一页成为左页或右页。这用于书中一章的结尾是很有用的，因为书中的一章通常始于右页，即使保留下空页也没关系。

下列的规则在文档中的每个 SONNET 元素前和后插入一个分页符但不在 SONNET 元素内插入分页符，以便十四行诗都出现在各自的页面上。

```
SONNET { page-break-before: always;
```

```
page-break-after: always;
```

```
page break-inside: avoid }
```

## 13.4 可视格式化

CSS2 添加了许多新的格式化功能，这些功能提供了对 XML 文档版面的更多控制。display 属性有许多新值，这些值扩充了 CSS1 基本块和内联类型。cursor 属性能够标识在对象上方显示何种光标。可控制所有对象框的高度和宽度。CSS2 还能修改文档对象的可视性、剪切大小、颜色、字体、正文阴影、对齐方式，并且如果内容超出页面，还能控制如何处理这种情况。

### 13.4.1 显示属性

CSS2 中扩充的 display 属性提供了更完整的版面选项，其中最显著的就是表格。在 CSS2 中，有 17 个显示属性值：

Inline	table-header-group
Block	table-footer-group
list-item	table-row
run-in	table-column-group
Compact	table-column
Marker	table-cell
Table	table-caption
inline-table	none
table-row-group	

块（block）元素通过在对象周围添加间距来在其内容的周围放置一个缓冲地带。内联（inline）元素不用放置边缘上的间距。表格（table）元素是各种网格。内联元素就像句子中的单词一样，其位置随着文本的增加或删除而自由移动。块对象更加固定，当在其前和后添加内容时，顶多只能上下移动，但不能左右移动。大多数显示类型只修改主块或内联类型。

#### 13.4.1.1 内联对象

内联对象（inline object）框水平放置在起始于包含它周围页面或块元素的顶端一行。在水平页边距这些框之间，可实现边界和贴边间隔。也可将这些框以各种方式（包括字符基线、框底或框顶）垂直对齐。



在 CSS1 中，block 值是所有对象的缺省显示类型，但在 CSS2 中发生了变化。现在元素自动地以 inline 显示，除非指定了其他类型。

#### 13.4.1.2 块对象

块对象（block object）一个挨一个地往下垂直分布。第一个块对象处于包含块的左上角，然后第二个块置于它之下，并齐排列在包含块的左边。各块之间的垂直距离由各块的页边距和贴边属性来定义。例如，下面的这段规则把 VERSE、STANZA 和 REFRAIN 元素当作各块来看待。图 13-1 显示的是清单 13-1 应用了此规则（并只有此规则）之后的结果。注意，AUTHOR 和 TITLE 都处于同一行，因为缺省情况下它们是内联的。但是，当块元素跟在一个内联元素之后时，就需要在块元素后面使用分行符。

```
VERSE, STANZA, REFRAIN { display: block }
```

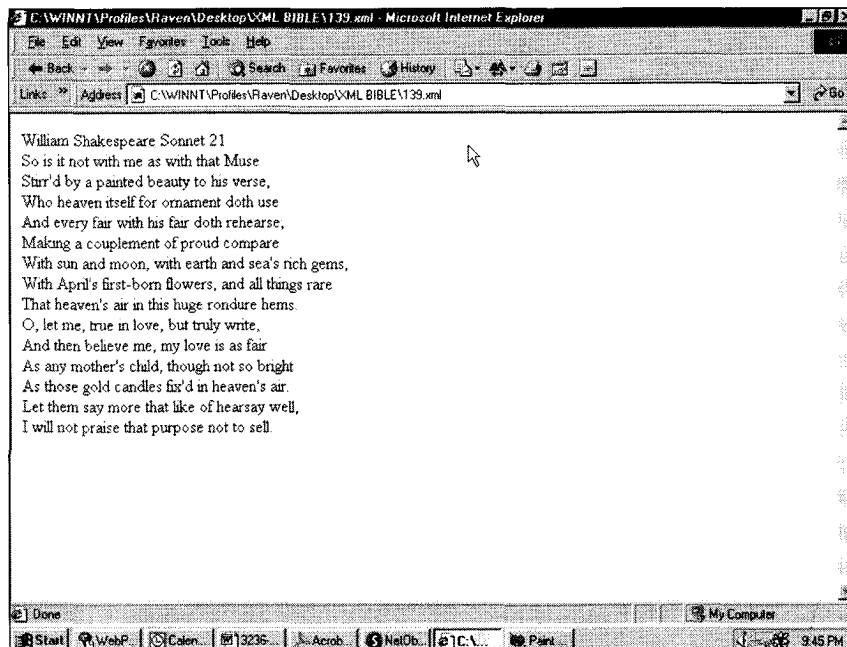


图 13-1 当作为块元素显示时, Shakespeare 的十四行诗呈现的外观更正常

#### 13.4.1.3 none 值

none 值强制此元素不产生任何类型的显示框来格式化元素的内容。另一方面, 元素对文档的版面没有任何影响。子元素和其他后代元素也不产生框, 即使它们设置了 display 属性也是如此。当 display 为 none 时, 那么此框不仅仅是不可见, 而是它确实不存在。

#### 13.4.1.4 compact 和 run-in 值

display 属性的 compact 和 run-in 值根据上下文的前后关系, 把一元素既可以标识为块, 也可以标识为内联框。根据属性的最终显示状态, 用于声明为这种类型的项目上的属性, 将是很有有效的。如果合适, compact 框将放在紧随其后的块框的页边距之内。如果跟在 compact 框之后的框不是块框, 或者 compact 框在页面距内放不下, 那么它将只作为另一块框显示。

run-in 值能够将正常的块元素格式化为代码中下一块元素的第一个内联框。如果下一个元素不是块元素, 那么 run-in 元素就能作为块元素被格式化。

#### 13.4.1.5 标记值

将 display 属性设置为 marker 值时, 可标识由样式单中生成的内容所形成的块, 而不是从 XML 文档复制来的内容所形成的块。此值只能与已和块级元素相联系的: before 和: after 伪元素一起使用。

#### 13.4.1.6 表格显示值

在 CSS2 中, 尤其对于那些经常使用标记 (这种标记与 HTML 的表格标记完全不同) 创建表格结构的 XML 开发人员来说, 最重要的新功能之一就是支持元素的表格版面。使用下列 10 个 display 属性值, 就可以使 CSS2 增加将元素格式化为表格的支持。

1. table
2. inline-table
3. table-row
4. table-column-group
5. table-row-group
6. table-row
7. table-column-group

- |                       |                   |
|-----------------------|-------------------|
| 3. table-row-group    | 8. table-column   |
| 4. table-header-group | 9. table-cell     |
| 5. table-footer-group | 10. table-caption |

例如，将 display 属性设置为 table，表示所选择的元素是块级容器，用于将较小的子元素放置在表格的单元格中。inline-table 值强制表格具有内联对象的功能，使正文能够沿着其侧边浮动，对于多个表格，并排放置。table-caption 值将元素格式化为表的标题。table-row-group、table-header-group 和 table-footer-group 值创建只起一行作用的数据单元组，就像使用 table-row 值定义的一样。table-column-group 创建充当一列的一组数据单元，可使用 table-column 值来定义。出现在表格单元中的 XML 元素应该有带有 table-cell 值的 display 属性，这已足够。

例如，如果要将十四行诗形成类似于表的结构，可将每个 STANZA 和 REFRAIN 设置为一张表，每个 VERSE 设置为表的一行。创建这种效果的样式单可能包括如下三条规则：

```
STANZA { display: table }

REFRAIN { display: table }

VERSE { display: table-row }
```

### 13.4.2 宽度和高度属性

用于显示每个元素的框的缺省高度可从元素内容的整体高度算出。每个元素框的缺省宽度可从元素内容的整体宽度算出，或从页面或屏幕的可视区域的宽度算出。内联元素和包含正文的表格元素总是具有这类自动地计算出的尺寸。但是，样式单设计者可以改变块级元素的这些缺省值，并通过指定下列的六个属性值来代替内联元素：

1. min-width
2. max-width
3. min-height
4. max-height
5. height
6. width

min-height 和 min-width 属性指定可用来显示对象的最小尺寸。最大属性是框的最大尺寸，而不管其内容的总尺寸。Web 浏览器在这些限制内可自由调节框的大小。但是，如果设置了 height 和 width，那么就可精确地确定框的大小。

```
STANZA { width: 100px;

Height: 100px }
```

### 13.4.3 overflow 属性

当使用 width 和 height 精确地指定一个框的大小时，`overflow` 属性控制如何处理超过的内容。此属性可设置为下列四个值之一：

1. auto
2. hidden
3. scroll
4. visible

如果 overflow 设置为 auto，必要时将会添加滚动条，以便用户能够看见超过的内容。如果将 overflow 设置为 hidden，超过的内容被裁去。如果将 overflow 设置为 scroll，那么无论内容是否超出显示范围，都会增加滚动条。最后，如果将 overflow 设置为 visible，将显示整个内容，如有必要，则推翻框的大小约束。

图 13-2 显示的是当使用下列规则，将 STANZA 的 overflow 属性设置为 scroll 时的十四行诗：

```
STANZA { overflow: scroll }
```

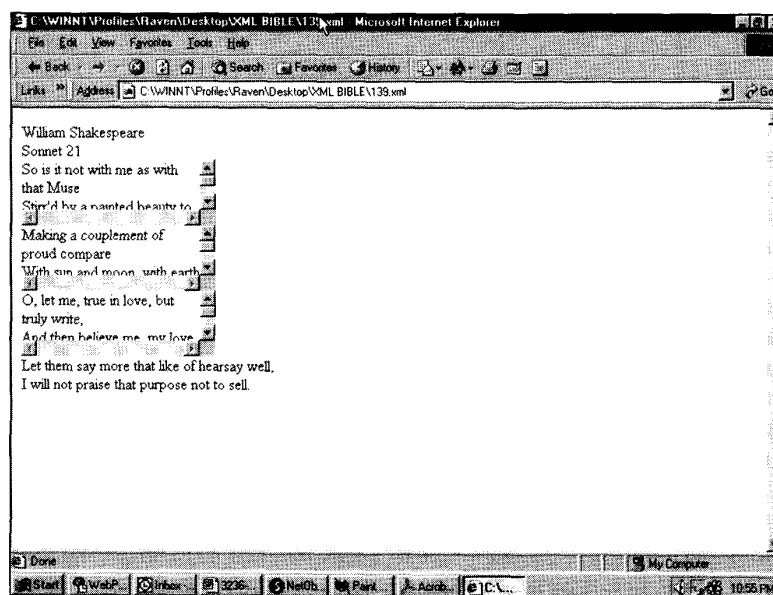


图 13-2 具有滚动条的 Shakespeare 的十四行诗的节

#### 13.4.4 clip 属性

当由用户软件显示对象内容时，clip（剪辑）属性标识可见的对象内容部分。一般地，剪辑区与元素框的外边界一致，但此区是可以改变的。此属性只适用于具有 overflow 特性（设置为非 visible 的任一值）的元素。

在 CSS2 中，只能剪辑成矩形区域。将 clip 属性设置为 rect(top, bottom, left, right)，这里的 top、bottom、left 和 right 为各侧面的偏移量。如果剪辑过的对象仍超过浏览器窗口的可视区域，那么内容将会被进一步剪辑以适应于窗口的大小。下列规则将 clip 属性用于 STANZA 块元素：

```
STANZA { clip: rect(5px, 5px, 5px, 5px);
```

```
overflow: auto }
```

#### 13.4.5 visibility 属性

visibility 属性控制元素的内容是否看得见。此属性的四个值为：

1. visible
2. hidden
3. collapse
4. inherit

如果将 `visibility` 设置为 `visible` 的话，框的内容（包括所有的边界）都会显示。如果将 `visibility` 设置为 `hidden`，那么框的内容和边界都看不见。不可见框仍占据空间，影响文档的布局。将 `visibility` 设置为 `hidden` 与将 `display` 属性设置为 `none`，是不一样的。

如果将 `visibility` 设置为 `collapse`，除表格的行或列外，对于任何对象来说，与使用 `hidden` 是一样的。但是，对于表格的行和列来说，这种设置完全隐藏（同 `display:none` 一样）行或列。

#### 13.4.6 cursor 属性

光标是箭头指针/手形/插入条/其他图标，这些图标表示鼠标箭头在屏幕上的位置。光标是显示于计算机显示器可视区上的鼠标逻辑位置的可见图像。当读者将鼠标移到特定的对象之上时，`cursor` 属性指定用户软件应该显示的光标。CSS2 有 16 个光标值：

1. `auto`：浏览器基于当前上下文关系来选择光标。此为缺省值
2. `crosshair`：一种简单的十字形光标
3. `default`：由操作系统平台决定的缺省光标，通常为箭头
4. `hand`：手形
5. `move`：交叉的箭头，表示要移动某物
6. `e-resize`：指向东方（右边）的箭头（上方是北边）
7. `ne-resize`：指向东北（右上）角的箭头
8. `nw-resize`：指向西北（左上）角的箭头
9. `n-resize`：指向北（上）方的箭头
10. `se-resize`：指向东南（右下）角的箭头
11. `sw-resize`：指向西南（左下）角的箭头
12. `s-resize`：指向南（下）方的箭头
13. `w-resize`：指向西（左）方的箭头
14. `text`：I-字型
15. `wait`：秒表、砗螺、沙漏或其他图标，表示要经历一段时间



16. help: 问号

下面的规则使用 `cursor` 属性说明当光标指针移到 VERSE 元素上时，应使用手形光标。

```
VERSE { cursor: hand }
```

也可以使用自定义的光标，该光标从给定影像的 URL 影像文件中加载。一般地，可以提供几种格式的光标，格式之间以逗号分开，最后一个应是通用光标名称。例如：

```
VERSE { cursor: url( "poetry.cur" ), url( "poetry.gif" ), text }
```

13.4.7 相关的颜色属性

CSS2 根据 Internet 普遍使用的标准缺省颜色空间 (sRGB) 中的 RGB 的值来标识颜色。表示颜色的方式随浏览器而变，但是这种规范提供了一种明确的、客观上可度量的颜色外观的定义。符合此标准的 Web 浏览器根据 CSS2 规范所确定的颜色，来执行 Gamma 修正 (Gamma Correction)。在大多数条件下，sRGB 可以标识显示器的 Gamma 值为 2.2。这意味着，对于大多数计算机硬件来讲，CSS2 属性给定的颜色不得不调整，以有效地显示 Gamma 值为 2.2。



只在 CSS2 规则中确定的颜色才受到影响。使用于影像中的颜色可传递它们自己的颜色修正信息。

13.4.7.1 颜色属性

`color` 属性指定元素文本内容的前景色。可以使用文字描述的颜色名 (如 red) 或 RGB 值 (如 #CC0000)。颜色名 (或值) 包括 aqua (浅绿)、black (黑色)、blue (蓝色)、fuchsia (紫红)、gray (灰色)、green (绿色)、lime (酸橙)、maroon (栗色)、navy (海蓝)、olive (橄榄)、purple (紫色)、red (红色)、silver (银白)、teal (深青)、white (白色) 和 yellow (黄色)。

下列的样式规则使用标识颜色的所有三种方法，将颜色应用于三个元素。将 AUTHOR 元素的 RGB 指定为十六进制值 #FF0000，所有的 TITLE 元素均显示为红色，所有的 VERSE 元素以 `rgb( 255, 0, 0)` 颜色显示。这些值都是红色：

```
AUTHOR { color: #FF0000 }
```

```
TITLE { color: red }
```

```
VERSE { color: rgb(255,0,0) }
```

Gamma 修正

从本质上来讲，Gamma 修正控制影像的明亮度，所以影像能精确地显示在计算机屏幕上。还没有经过修正的影像显得褪了色或太暗。为了使 Gamma 修正易于理解，让我们来看看计算机屏幕上显示的影像。

实际上，每个计算机显示器都具有 Gamma 值为 2.5。这意味着，光强与电压的关系是 2.5 次幂函数。如果向显示器发送指定像素的信号，获得强度为  $x$ ，那么，此像素将自动地使强度为  $x^{2.5}$  加在显示器上。由于电压的范围是在 0 到 1 之间，这意味着，像素强度比所期望的低。为了修正这个像素强度，那么显示的电压就不得不进行“Gamma 修正”。

修正这个问题的最简单的方法就是在把电压送到显示器之前提高电压。由于电压和明亮度的关系是已知的，所以可调整信号来去除显示器的 Gamma 影响。当正确地做好这一切之后，计算机显示器就能精确地反映影像导入。当然，当对影像进行 Gamma 修正时，计算机室内的光线、显示器的明亮度和对比度，以及个

人的偏爱也起一定的作用。

当试图对 Web 进行 Gamma 修正时，系统平台的特质也会开始起作用。有些 UNIX 工作站会自动地修正显示卡上的 Gamma 变化，正如 Macintosh 机一样，但大多数 PC 机则不行。这意味着，在 PC 机上看得挺好的影像，在 Mac 机上就会太浅；反过来，某些影像在 Mac 机上能很好地显示，而在 PC 机上就显示太暗。如果把彩色影像或文本放在 Internet 上，那么就不可能使所有的人都感到满意。目前，在 Web 上使用的任何图像格式都没有 Gamma 修正的编码信息。

13.4.7.2 系统颜色

CSS2 使用户通过从本地的 GUI（图形用户界面）复制颜色来指定颜色。这些系统颜色可以和相关的颜色属性一起使用。基于系统颜色的样式规则考虑了用户的偏爱，所以具备如下的一些优点：

- 1. 页面适合于用户喜爱的观感。
- 2. 用户更容易访问的页面，加入了与残疾人有关的设置项。

表 13-2 列出了 CSS2 系统颜色的关键字及其含义。任何颜色属性都可取这些值。

例如，下列的样式单设置 VERSE 的前景色和背景色为浏览器窗口所使用的前景色和背景色。

```
VERSE { color: WindowText; background-color: Window }
```

表 13-2 与所有颜色有关的属性一起使用的其他系统颜色

系统颜色关键字	含义
ActiveBorder	活动窗口边框
ActiveCaption	活动窗口标题
Appworkspace	多文档界面的背景色
Background	桌面背景色
BottonFace	三维显示元素的外观颜色

## 13.5 框

当使用 CSS 来格式化一文档及其内容时，需要用到框。框具有边界和大小，用于存放元素的内容。这些框堆叠在一起并可互相覆盖，以便根据样式表的规则，以有序的方式对齐元素的内容。CSS2 给框添加了新的轮廓（outline）属性，使框能够定位在页面、其他框或窗口上的绝对位置处。

### 13.5.1 轮廓属性

CSS2 能够将轮廓加入到对象中。轮廓很像边框，但轮廓是绘在框之上的。其宽度不加入到框宽度上。此外，如果 CSS 元素是非矩形的（不大可能），在此元素周围的轮廓也将是非矩形的。由于轮廓不必一定是矩形的，所以不能分别设置左、右、顶和底轮廓，只能一次改变整个轮廓。

#### 13.5.1.1 轮廓样式属性

outline-style 属性设置整个框的轮廓样式，它起的作用如同 CSS1 中的 border-style 属性，并且具有同样的 11 个值，其含义也相同：

1. none: 无线条
2. hidden: 使线条不可见，但仍占据空间
3. dotted: 点线
4. dashed: 虚线
5. solid: 实线
6. double: 双实线
7. grooved: 凹槽线，好像埋入页面
8. ridge: 凸纹线，好像突出页面
9. inset: 嵌入线，整个对象（不仅仅是轮廓线）像是推入到文档里面
10. outset: 外置线，整个对象（不仅仅是轮廓线）像是推出文档
11. inherit: 使用父类的值

下列三条规则设置 TITLE、AUTHOR 和 REFRAIN 元素的轮廓样式：

```
TITLE { outline-style: solid }
```

```
AUTHOR { outline-style: outset }
```

```
REFRAIN {outline-style: dashed }
```

#### 13.5.1.2 轮廓宽度属性

outline-width 属性的作用像第 12 章讨论的 margin-width 和 border-width 属性一样，可使用无符号的长度或下列三个关键字之一来设置框的轮廓宽度：

1. thin: 大约 0.5 到 0.75 磅
2. medium: 大约 1 磅
3. thick: 大约 1.5 到 2 磅

例如下面的这条规则给 STANZA 加上一条粗轮廓线，给 VERSE 加上一条细轮廓线。

```
STANZA { outline: thick }
```

```
VERSE { outline: thin }
```

#### 13.5.1.3 轮廓颜色属性

outline-color 属性设置元素框的轮廓颜色。一般地说，这种设置既可以使用颜色名（如 red），也可以使用 RGB 颜色（如 #FF0000）。但是，还有关键字值 invert，此值反转屏幕的像素颜色（黑色变成白色，反之亦然）。

```
TITLE { outline color: #FFCCCC;
```

```
outline-style: inset;
```

```
outline width: thick }
```

```
AUTHOR { outline color: #FF33CC }
```

```
VERSE { outline-color: invert }
```

#### 13.5.1.4 轮廓简略属性

outline 属性是简略属性，它设置容器框的所有四个边的轮廓宽度、颜色和样式。例如：

```
STANZA { outline: thin dashed red }
```

```
VERSE { outline: inset }
```

### 13.5.2 定位属性

CSS2 对文档中每个对象的位置提供了多种控制方法。可以把特定的对象或对象的特定类型按层放置。每层与其他层无关，可独立移动。position 属性确定对象如何排列，可使用下列四个关键字值之一：

1. static: 缺省的布局
2. relative: 对象偏移其静态位置
3. absolute: 相对于包含对象的框，将对象放置在特定的位置
4. fixed: 对象放在窗口或页面的特定位置

#### 13.5.2.1 相对定位 (relative positioning)

编排文档时，格式化标识符根据对象和文本的正常信息流（flow），选择项目的位置。实际上，这就是对象缺省的静态格式化，大多数文档创作者都使用这种方法来编排文档。编排文档完成之后，对象相对于当前位置可能会发生偏移。这种对象位置的调整就是所谓的相对定位。使用相对定位，改变对象的位置，对其后的对象没有任何影响。由于相对定位的框完全保持其正常的信息流的大小和间隔，因此，框可以交叠。

将 position 属性设置为 relative，可生成一个相对定位的对象。其偏移量可由 left、right、top 和 bottom 属性来控制。使用 JavaScript 来改变这些属性，甚至可以在文档上移动对象和层。可以使影像或文本移动、出现或消失、或中途改变。例如，下面的这条规则将 TITLE 元素从正常的位置向上移动 50 像素，向左移动 65 像素。

```
TITLE { position: relative; top: 50px; left: 65px }
```

#### 13.5.2.2 绝对定位 (Absolute Positioning)

绝对定位元素参照包含它的块来放置。它可为它包含的框建立一个新的包含块。绝对定位元素的内容不会在其他框周围流动。这样可能会使它们造成显示在文档中的其他框的内容模糊不清。绝对定位元素对其后同属的出现顺序无任何影响，所以跟在绝对定位元素后面的元素所产生的效果，就如同绝对定位元素不在此处一样。

```
AUTHOR { position: absolute; top: 60px; left: 140px }
```

#### 13.5.2.3 精确定位 (Fixed Positioning)

具有固定位置的元素相对于其显示窗口或页面的坐标放置。如果正在查看由连续媒体组成的文档，那么当此文档滚动时，被固定的框就不会移动。如果被固定的框处于分页媒体上，那它总是会出现在每页的末尾。这样就能够把页脚或页眉放在文档上，或将签名放在一系列只有一页纸的信件末尾。例如，下面的规则，以将 REFRAIN 元素的左上角放在距显示窗口（或打印的纸张）左上角往下 300 像素、往右 140 像素的地方。

```
REFRAIN { position: fixed; top: 300px; left: 140px }
```

#### 13.5.2.4 使用 z-index 属性来层叠元素

z-index 属性控制定位框的层叠顺序。要改变缺省的 z-index 值，可将 z-index 设置为整数（如 2）。有较大 z-index 值的对象放在较小 z-index 值的对象的顶部。底部的对象无论是否完全显示，它们都决定于其顶部对象的背景属性。如果背景是透明的，那么，处于下面的对象至少有一些可能完全透过。

清单 13-2 是使用绝对定位的样式单，这个样式单有一个 z-index，用它来创建 Shakespeare 十四行诗多部分的重叠效果。图 13-3 显示了此情景。这肯定不如由浏览器来编排此诗的样子好看。使用绝对定位应十分小心。我只推荐用于印刷媒体，以此媒体分发论文，而不是电子文件。

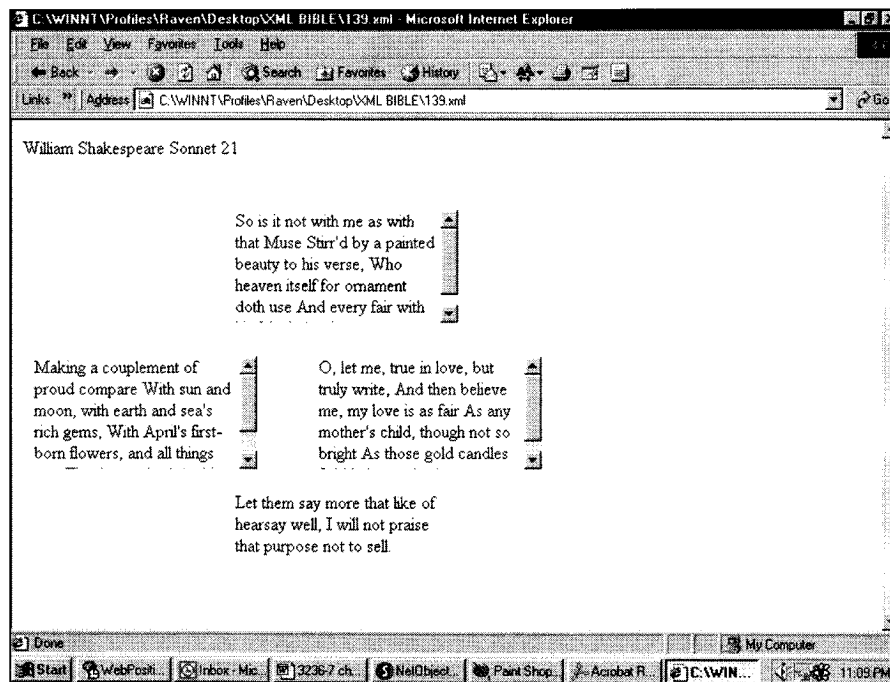


图 13-3 使用 z-index 排序的绝对定位，可以控制文本框的重叠顺序

清单 13-2: Shakespeare 十四行诗的 z-index 样式单

```
#st1 { position: absolute;
```

```
top: 160px;
```

```
left:200px;
```

```
height: 100px;
```

```
width:200px;
```

```
overflow: auto,
```

```
z-index: 2 }
```

```
#st2 { position: absolute;
```

```
top: 210px;
```

```
left:50px;
```

```
height: 100px;
```

```
width:200px;
```

```
overflow: auto;
```

```
z-index: 3 }
```

```
#st3 { position: absolute;
```

top: 210px;

left:250px;

height: 100px;

width:200px;

overflow: auto;

z-index: 4 }

REFRAIN { position: absolute;

top: 300px;

left:200px;

height: 100px;

width:200px;

overflow: auto;

z-index: 5 }

## 13.6 计数器和自动编号

CSS2 能自动地产生某些内容。例如，可使用样式单来创建这样的一个大纲：使用不同的编号系统对大纲的每个层次进行适当的缩进。

`counter-increment` 属性使计数器 (counter) 加 1。`content` 属性使用 `counter(id)` 或 `counter(id, list-style-type)` 函数作为值，插入指定计数器的当前值。最后，`counter-reset` 属性将计数器设置归 0。

例如，假定从 1 开始要对诗中的每个 VERSE 进行计数，但在每个新的 STANZA 和 REFRAIN 中重新计数。可使用下面的规则达到此目的：

```
VERSE { counter-increment: verse-num }
```

```
STANZA { counter-reset: verse-num }
```

```
REFRAIN { counter-reset: verse-num }
```

```
VERSE: before { content: counter(verse-num) }
```

可在 `counter-reset` 中的计数器名后，指定重设的整数，就可以重新将计数器设回到非 0 的数。例如，下面的规则将计数器设回到-10：

```
VERSE { counter-reset: verse-num -10 }
```

在 `counter-increment` 中的计数器名后，指定非 1 的整数作为递增量。例如，

```
VERSE { counter-increment: verse-num -1 }
```

最后，`content` 属性可不止一个计数器以及附加内容及其计数器。例如，下面的这些规则以 1.1, 1.2, 1.3, ..., 2.1, 2.2, 2.3, ... 的形式对诗进行编号，第一编号表示节，第二个编号为诗句：

```
VERSE {counter-increment: verse-num }
```

```
STANZA {counter-reset: verse-num }
```

```
STANZA {counter-increment: stanza-num }
```

```
REFRAIN {counter-reset: verse-num }
```

```
REFRAIN {counter-reset: stanza-num 0 }
```

```
VERSE:before {content:
```

```
counter(stanza-num) ". " counter(verse-num)
```

还可以使用非欧洲数字。`counter()` 函数的第二个参数可用来指定不同的数字格式。现有的格式有 `disc`、`circle`、`square`、`decimal`、`decimal-leading-zero`、`lower-roman`、`upper-roman`、`lower-greek`、`lower-alpha`、`lower-latin`、`upper-alpha`、`upper-latin`、`hebrew`、`armenian`、`georgian`、`cjk-ideographic`、`hiragana`、`katakana`、`hiragana-iroha` 和 `katakana-iroha`。例如，使用平假名的日文数字来对诗进行编号，可以这样编写：

```
VERSE: before {content: counter(stanza-num, hiragana)
```



```
". " counter(verse-num, hiragana) }
```

## 13.7 有声样式单

视觉缺陷的用户已经使用特殊的软件来阅读 Web 网页。将来，使用这种方法可能会扩大到视觉完好的人，他们一边浏览 Web，同时一边使用电话进行交谈、一边驾驶汽车、一边洗涮盘子以及进行其他活动，而在这些过程中，眼睛和手用于不同的目的。CSS2 支持一些新的特性，以便描述如何大声地读出元素，以及如何打印或在屏幕上显示。这些新特性将在下面的章节中讨论。清单 13-3 为一有声样式单，它使用指定方法，来讲出在普通与剧本有关的 XML 元素中找到的信息。

清单 13-3：用于剧本或十四行诗的有声样式单

```
TITLE, AUTHOR, ACT, SCENE {

voice-family: narrator;

stress: 20;

richness: 90;

cue-before: url("ping.au")

}

.narrator { pause: 20ms;

cue-before: url("pop.au");

cue-after: url("pop.au");

azimuth: 30deg;

elevation: above }

ACT { pause: 30ms 40ms } /* pause-before: 30ms;

pause-after: 40ms */

SCENE { pause-after: 10ms } /* pause-after: 10ms */

SCENE { cue-before: url("bell.aiff");

cue-after: url("dong.wav") }

MOOD.sad { play-during: url("violins.aiff") }

MOOD.funereal { play-during: url("harp.wav") mix }

MOOD.quiet { play-during: none }

LINE.narrator { azimuth: behind } /* 180deg */

LINE.part.romeo { voice-family: romeo, male }

LINE.part.juliet { voice-family: juliet, female }
```

```
LINE.part.hercules { azimuth: center-left }
```

```
LINE.part.richard { azimuth: right }
```

```
LINE.part.carmen { volume: x-soft }
```

```
LINE.part.musel { elevation: 60deg }
```

```
LINE.part.muse2 { elevation: 30deg }
```

```
LINE.part.muse3 { elevation: level }
```

### 13.7.1 说话属性

speak（说话）属性确定是否用声音说出文本。如果是，应如何说。如果 speak 为 normal 值，则使用最佳的语音合成来说出此段文本。如果 speak 值为 spell-out，则一个字母一个字母地拼出文本，这对于不常用或语音合成器可能无法处理的外来语来说，或许是很有用的。缺省值是 none（例如，只是视觉上显示内容，而不管语言合成）。

### 13.7.2 音量属性

volume（音量）属性控制语音合成器讲话声音的平均音量。这是模拟语音声波中的中等值，但只是平均来说。音量为 50 的高反射的语音峰值可能达到 75。音量最小值为 0，最大音量为 100。音量中，也可以使用百分比值，使用下面的六个关键字也可达到同样的目的：

1. silent：无声音
2. x-soft：0，最小的可听量
3. soft：大约为 25%
4. medium：大约为 50%
5. loud：大约为 75%
6. x-loud：100%，最大舒适度的听力级别

### 13.7.3 暂停属性

Pause（暂停）属性在听觉上等价于逗号，可用于提供戏剧色彩，或有助于将每位说话人的声音分开。在 CSS2 中，使用 pause、pause-before 和 pause-after 属性来设置暂停。

pause-before 属性指定语音合成器在说出一元素的内容之前应暂停的时间长度。pause-after 属性指定语音合成器在说出一元素的内容之后应暂停的时间长度。这些都可以用绝对时间或语速属性的百分数来设置。pause 属性是设置 pause-before 和 pause-after 属性的简略属性。当使用两个值时，第一个值用于 pause-before，第二个值用于 pause-after。当只给出一个值时，此值可应用于这两个属性。例如：

```
SCENE { pause-after: 10ms }
```

```
/* pause-before: 20ms: pause after: 20ms */
```

```
.narrator { pause: 20ms }
```

```
/* pause-before: 30ms; pause-after: 40ms */
```

```
ACT { pause: 30ms 40ms }
```

#### 13.7.4 提示属性

cue（提示）属性是听得见的线索，是用来提醒听众将要发生或刚刚发生的特定事件。每个提示属性都指定某个元素说出之前或之后将要播放的一个声音文件的 URL。cue-before 属性在读出元素之前就播放声音。cue-after 属性在读出元素之后才播放声音。

cue 属性是设置 cue-before 和 cue-after 属性的简略属性。当使用两个值时，第一个值用于 cue-before，第二个值用于 cue-after。当只给出一个值时，此值可应用于这两个属性。例如：

```
ACT, SCENE { cue-before: url( "ping.au" ) }
```

```
.narrator { cue: url( "pop.au" ) }
```

```
SCFNF { cue before: url( "bell.aiff" );
```

```
cue after: url( "dong.wav" ) }
```

#### 13.7.5 同期播放属性

play-during（同期播放）属性指定一边说出元素内容一边播放背景声音。此属性的值为声音文件的 URL。也可以把 mix 和 repeat 关键字之一或全部加入到此值中。mix 告诉语音合成器将在父元素的 play-during 声音中形成混合音。repeat 值告诉语音合成器连续不断地循环放音，直至整个元素说完为止。缺省值为 none。

#### 13.7.6 空间属性

spatial（空间）属性指定声音好像出自于何处。例如，可以是在 3 英尺外的壕沟里，或在 100 英尺外的悬崖上读出文档的。当然，这受到语音合成器和音频功能的限制。由于无法预先确定文档读者所使用的喇叭的数量和位置，所以这些属性只确定所期望的最终结果。作为文档的作者，实际上无法强制声音来自某个特定的方向，这与保证读者确有显示器是不一样的。

##### 13.7.6.1 方位角属性

azimuth（方位角）属性控制发自声音的水平角度。当使用好的立体声喇叭收听信号时，好像能听见横向声音。azimuth 属性与这类立体系统一起使用，以形成让人听到的声音的角度。当使用双耳式耳机或五喇叭家庭影院设备以形成全环绕的声音系统时，azimuth 属性就非常引人注目。

azimuth 可被指定为 $-360^{\circ}$  到  $360^{\circ}$  之间的任一角度值。0deg 值表示声音是从听众的前方直接传来的（ $-360\text{deg}$  和  $360\text{deg}$  具有同样的效果）。180deg 值表示声音是从听众的后方直接传来的（在 CSS 术语中，deg 代替了更普通的  $^{\circ}$  度符号）。角度是以听众正面的顺时针方向计算的。还可以使用下列九个关键词中的一个，来指定方位角的角度。

1. center: 0deg

2. center-right: 20deg

3. right: 40deg

4. far-right: 60deg
5. right-side: 90deg
6. left-side: 270deg
7. far-left: 300deg
8. left: 320deg
9. center-left: 340deg

可以将关键字 behind 加到这些值中的任何一个中，以设置 180deg 减正常值的位置。例如，left behind 与  $180\text{deg} - 320\text{deg} = -140\text{deg}$  或  $220\text{deg}$  的意义是一样的。

leftwards 的值将声音相对于当前角度再向左移动 20 度。可理解为将声音反时针方向旋转。所以即使声音已经处在听众的后面，也将以反时针方向继续移动到“左边”。rightwards 值将声音从当前角度向右（顺时针方向）再移动 20 度。

#### 13.7.6.2 高度属性

elevation（高度）属性控制喇叭距听众位置的表观高度。此高度可指定为  $-90^\circ$  至  $90^\circ$  的任一角度，也可为下列五个关键字之一：

1. below  $-90\text{deg}$
2. level  $0\text{deg}$
3. above  $90\text{deg}$
4. higher 处于当前高度上方  $10\text{deg}$ （这对继承是很有用的）
5. lower 处于当前高度下方  $10\text{deg}$ （这对继承是很有用的）

### 13.7.7 音质属性

通过调节语速、所使用的声系、语调（pitch）和声音的激昂程度，合成器声音的各个特性都可控制。

#### 13.7.7.1 语速属性

speech-rate（语速）属性指定语音合成器的说话速率，以每分钟平均大小的单词的近似数值来表示。可提供某个整数或下列五个关键字之一：

1. x-slow: 每分钟 80 个单词
2. slow: 每分钟 120 个单词
3. medium: 每分钟 180 至 200 个单词
4. fast: 每分钟 300 个单词

5. x-fast: 每分钟 500 个单词

还可以使用 faster 关键字, 以使父元素的速率每分钟增加 40 个单词, 或使用 slower 关键字, 以使父元素的速率每分钟减少 40 个单词。

#### 13.7.7.2 声系属性

voice-family (声系) 属性是以逗号分开的、区分优先顺序的声系名称的一组列表, 而声系名称选择用于阅读文档文本的声音。它很像第 12 章讨论过的字族属性, 但它关心的是声音而不是字样。

通常的声音值有 male、female 和 child。与字体名一样, 所指定的名称也是多种多样, 包括 Agnes、Bruce、Good News、Hysterical、Victoria、Whisper 等等。如果这些名称不符合标识符的语法规则或是由多个单词组成的话, 那么就必须用引号括起来。例如:

```
LINE.part.romeo { voice-family: Bruce, "Good News", male }
```

#### 13.7.7.3 语调属性

pitch (语调) 属性指定语音合成器用于特殊类型对象的频率。在某种程序上, 它控制所发的声音是男声还是女声。但是, 更好的方法是使用适当的声系。此值以赫兹 (hertz, 每秒周期数) 为单位。女声约为 120Hz, 而一般的男声差不多为 200Hz。也可使用下面的关键字来调节语调:

1. x-low

2. low

3. medium

4. high

5. x-high

这些关键字的精确频率狄 览涤漠没 Y 幕肪激脱=竦纳 簪? 埒牵璆-low 总是比 low 低, 而后者又总是比 medium 低, 依次类推。

#### 13.7.7.4 语调范围属性

pitch-range (语调范围) 属性指定以喇叭平均语调表示的可接受的变化范围, 其数值为 0 至 100 之间。它控制语音合成使用的声音的变形和变调。值为 0 时, 形成一平坦的单音, 而值为 50 时, 则为正常音, 值在 50 以上时, 异常兴奋的声音。

#### 13.7.7.5 重音属性

stress (重音) 属性指定断言或强调的程度, 常用于演讲, 缺省值为 50。这种特性的值和结果对正在说的各种语言具有不同的效果。当使用语言 (如英语) 来强调句子的作用时, 可选择主、次和第三强调点来控制语音的变化, 以将这种语音的变化应用于此句中的不同地方。

#### 13.7.7.6 激昂程度属性

richness (激昂程度) 属性指定语音合成器使用的声音的“鲜明度”。声音越激昂, 其传播能力就越好。语调平缓的声音不会传得很远, 这是因为其声波不如激昂的声那样深沉。此值为 1 和 100 之间的一个数

值，缺省值为 50。此值越高，所产生的声音传播得就越好；而此值越低，就会获得更柔软的声音，更容易听到。

### 13.7.8 话音属性

这些属性控制语音合成器如何解释标点符号和数字。有两个属性：speak-punctuation 和 speak-numeral。

#### 13.7.8.1 发标点符号音属性

在缺省的情况下，标点符号是逐字说出的。对于诸如“The cat, Charm, ate all of his food.”这样的句子，读作“The cat comma Charm comma ate all of his food period”。但是，可设置 speak-punctuation 属性为 none，从而不说出任何标点符号。可是将有暂停现象，就像是真实说话的声音一样。例如，“The cat <pause> Charm <pause> ate all of his food <silence>”。

#### 13.7.8.2 发数字音属性

在缺省的情况下，数字以完整的字符说出来。例如，数字 102 将读成“one hundred and two”。可是，如果将 speak-numeral 属性设置为 digits，那么，将分别说出每个数字原有格式，如 one zero two。将 speak-numeral 属性设置为 continuous，就会恢复到缺省状态。如果 speak-numeral 设置为 none，那么将不会说出数字。

## 13.8 本章小结

本章涉及到了 CSS2 的功能以及如何实现。在本章中，学习了以下内容：

- CSS2 几乎为 CSS1 的超集，尽管存在着一些差别（包括缺省的显示类型为 inline，而不是 block）。
- Internet Explorer 5 和 Mozilla 只是或多或少地执行 CSS2，所以不要指望 CSS2 的大多数功能都能完美实现。
- CSS2 已经扩充了各种选择符（其中包括通用选择符、子选择符、后代选择符以及同属选择符），通过这些选择符就能把指定的属性应用于特定的元素中。
- 已开发的新的@rules（包括@charset、@page 和@font-face），使文档的作者可对打印文档进行更多的控制。
- CSS2 有七个新的伪类（包括:first-child 和:hover），可用来选择具有共同部分的元素，但不必具有相同的类型。
- CSS2 有两个新的伪元素：:after 和:before，可使用它们向文档插入内容。
- CSS2 增加了显示属性的应用，将显示元素的值组合在一起，作为所有的表格部分来显示元素，不显示任何内容（none），也可作为 compact 或 run-in 对象来显示元素。
- 系统颜色和系统字体能够用来在 XML 应用程序中创建界面，而这种界面更加匹配各自客户计算机上全部的系统设置。
- CSS2 增加了音频属性，用以描述语音、音量、暂停、提示、发音特性以及其他声音中播放声音、及其应来自于何处的规范。

与 CSS1 一样，CSS2 仍有许多局限性，最明显的就是缺乏 Web 浏览器的充分支持，但这会随时间的推移而发生变化。XSL 到目前为止仍是用于 XML 文档的最佳的样式单语言。在下一章中，将探讨 XSL 变换，看看我们还能前进多远。



## 第 14 章 XSL 变换

可扩展的样式语言（Extensible Style Language, XSL）包括变换语言（transformation language）和格式化语言（formatting language）。每种语言都是一个 XML 应用程序。变换语言提供定义规则的元素如何将 XML 文档变换成另一个 XML 文档。被变换的 XML 文档可能使用原文档的标记和 DTD，或者使用一组完全不同的标记。特别是，可能会使用 XSL 第二部分（格式化对象）定义的标记。本章涉及到 XSL 变换语言中的部分内容。

本章的主要内容如下：

- 理解 XSL、XSL 变换和模板
- 计算节点的值
- 处理多个元素
- 用表达式选择节点
- 理解缺省的模板规则
- 确定输出要包含的内容
- 复制当前节点
- 对节点进行计数、对输出元素分类以及插入 CDATA 和 < 符号
- 设置模式特性
- 定义并创建命名模板
- 删除和保留空格
- 基于输入来改变输出
- 合并多个样式单

### 14.1 何为 XSL?

变换和格式两部分可相互独立地起作用。例如，变换语言可将 XML 文档变换成结构整洁的 HTML 文件，并且完全忽略 XSL 格式化对象。Internet Explorer 5.0 支持这种 XSL 样式，这在第 5 章已讨论过，本章着重讨论这种样式。

此外，以 XSL 格式化对象编写的文档，并非绝对要求在另一个 XML 文档上使用 XSL 变换部分才能产生。例如，很容易想象到这样的一个转换器：它是用 Java 语言写成的，可读取 TeX 或 PDF 文件，并把这些文件翻译成 XSL 格式化对象（尽管直到 1999 年夏天仍没有这样的一种转换器存在）。

实际上，XSL 是两种语言，而不是一种。第一种语言是变换语言，第二种是格式化语言。变换语言是一种很有用的语言，它与格式化语言无关。它能够把数据从一种 XML 表示移到另一种表示，这种功能，使它成为基于 XML 的电子商务、电子数据交换、元数据交换以及应用于需要在相同数据的不同 XML 表示之间进行转换的重要组成部分。由于缺乏对人们要浏览的显示器上显示数据的了解，这些用途还要结合起来使用。它们纯粹是用来将数据从一种计算机系统或程序移到另一种计算机系统或程序中。

因此，许多早期的 XSL 实现都毫无例外地将焦点集中在变换部分，而忽略了格式化对象。这些是不完善的实施方案，但仍然是很有用的。并非所有的数据最终都必须显示在计算机显示器上或打印到纸上。



第 15 章“XSL 格式化对象”将涉及 XSL 格式化语言。

有关 XSL 警告语

XSL 仍然处于开发中。XSL 语言在过去发生了根本性的变化，将来肯定会再发生变化。本章是根据 1999 年 4 月 21 日的 XSL 规范草案（第四稿）写成的。读者阅读此书时，此 XSL 草案可能已经被取代了，精确的 XSL 句法将会变化。我希望本章与实际规范不会相差太大。但是，如果的确有不一致的地方，应将本书中的例子与最新规范进行对比。

糟糕的是，仍然没有任何软件能实现 1999 年 4 月 21 日的 XSL 规范草案（第四稿）的所有内容，甚至不能实现 XSL 变换的部分。现有的所有产品只能实现当前草案的不同子集。而且，许多产品（包括 Internet Explorer 5.0 和 XT）加入的元素并没有出现在当前 XSL 草案规范中。最后一点是，大多数至少要实现部分 XSL 内容的产品在其可实现的部分中也存在着很严重的程序错误（bug）。因此，在不同的软件中，只有寥寥无几的几个例子能准确地以相同的方式工作。

当然，随着此项标准向最后版本改进时，当开发商解决了自己产品中的程序错误并实现没有被实现的内容时，以及当出版的更多软件支持 XSL 时，最终这种情况是可以得到修正的。在达到此目的之前，还得面对这样的选择：要么忍痛使用目前不完善的、未完成的 XSL，并且试图避开遇到的所有程序错误和疏忽，要么使用更确定的技术（如 CSS），直到 XSL 更加可靠为止。



## 14.2 XSL 变换概述

在 XSL 变换中，XSL 处理程序读取 XML 文档和 XSL 样式单。基于处理程序在 XSL 样式单中找到的指令，输出新的 XML 文档。

### 14.2.1 树形结构

就像第 6 章学到的那样，每个结构整洁的 XML 文档都是树形结构（tree）。树形结构是一种数据结构，它是由连接起来的节点（node）组成的，这些节点起始于一个称为根节点（root）的单节点。根节点连接它的子节点，每个子节点可以连接零个或多个它自己的子节点，依次类推。没有自己的子节点的节点称为叶节点（leave）。树形结构的图表更像家谱，列出各个先辈的后代。树形结构最有用的特征是每个节点及其子节点也会形成树形结构。因此，一个树形结构就是所有树形结构的分级结构，在此分级结构中，各树形结构都是由更小的树形结构建立的。

XML 树形结构的节点就是元素及元素的内容。但是，对于 XSL，特性、命名域（namespace）、处理指令以及注释也必须也作为节点看待。而且文档的根节点必须与根（基本）元素区别开来。因此，XSL 处理程序假定 XML 树形结构包含下列七类节点：

1. 根节点
2. 元素
3. 文本
4. 特性
5. 命名域
6. 处理指令
7. 注释

例如，对于清单 14-1 中的 XML 文档，它显示的是元素周期表，在本章我将用这个元素周期表作为例子（更恰当地说，是周期表中的前两个元素）。

完整的元素周期表放在本书所附光盘中的 examples/periodic\_table 文件夹中的 allelements.xml 文件中。

根节点 PERIODIC\_TABLE 元素包含 ATOM 子元素。每个 ATOM 元素含有各种子元素，以便提供原子序数、原子量、符号、沸点等等信息。UNITS 特性为具有单位的元素指定单位。



这里使用 ELEMENT 比 ATOM 或许更恰当。但是，写成 ELEMENT 元素难以区分化学元素和 XML 元素。因此，起码出于本章的考虑，ATOM 似乎更具可读性。

清单 14-1：氢和氦元素的 XML 周期表

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

<?xml-stylesheet type="text/xsl" href="14-2.xsl" ?>

<PERIODIC TABLE>

<ATOM STATE="GAS" >
```

```

<NAME>Hydrogen</NAME>

<SYMBOL>H</SYMBOL>

<ATOMIC_NUMBER>1</ATOMIC_NUMBER>

<ATOMIC_WLIGHT>1.00794</ATOMIC_WEIGHT>

<BOILING_POINT UNITS=" Kelvin" >20.28</BOILING_POINT>

<MELTING_POINT UNITS=" Kelvin" >13.81</MELTING_POINT>

<DENSITY UNITS=" grdMS/cubic centimeter" ><!-- At 300K -->

0.0899

</DENSITY>

</ATOM>

<ATOM STATE=" GAS" >

<NAME>Helium</NAME>

<SYMBOL>He</SYMBOL>

<ATOMIC_NUMBER>2</ATOMIC_NUMBER>

<ATOMIC_WEIGHT>4.0026</ATOMIC_WEIGHT>

<BOILING_POINT UNITS=" Kelvin" >4.216</BOILING_POINT>

<MELTING_POINT UNITS=" Kelvin" >0.95</MELTING_POINT>

<DENSITY UNITS=" grams/cubic centimeter" ><!-- At 300K -->

0.1785

</DENSITY>

</ATOM>

</PERIODIC_TABLE>

```

图 14-1 显示的是本文档作为树形结构的图解。它起始于顶端的根节点(不同于根元素!),包括两个子节点:xml-stylesheet 处理指令和根元素 PERIODIC\_TABLE。(XML 声明对 XSL 处理程序是不可见的,因而不包括在 XSL 处理程序进行操作的树形结构中)。PERIODIC\_TABLE 元素包括两个子节点,即两个 ATOM 元素。每个 ATOM 元素都有一个 STATE 特性的特性节点和各种子元素节点。每个子元素包括其内容的节点,以及任何特性节点和拥有的注释。注意,在特殊情况下,许多节点可以是除元素之外的任何内容。节点可为文本、特性、注释和处理指令。与 CSS1 不同,XSL 不限于只和全部元素一起使用,还有一个更加独特地查看文档的方法:能够根据注释、特性、处理指令等设计样式。



像 XML 声明一样，内部的 DTD 子集或 DOCTYPE 声明不是树形结构的一部分。但是，通过使用 #FIXED 或缺省特性值的 <!ATTLIST> 声明，它可能具有将特性节点添加到某些元素中的效果。

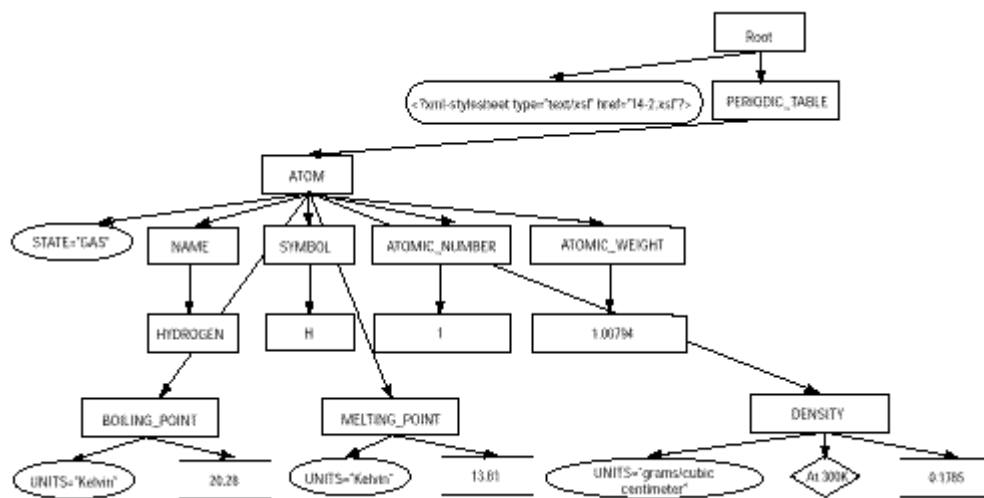


图 14-1 以树形图表示的清单 14-1

XSL 变换语言通过将 XML 树形结构变换成另一个 XML 树形结构来操作。这种语言含有操作符，此操作符用来从树形结构中选择特定节点、对节点重新排序以及输出节点。如果有一个节点是元素节点，那么它本身可能就是整个树形结构。请记住，所有的用于输入和输出的操作符都只能操作一个树形结构。它们不是用于变换任意数据的通用的正常表达语言。

### 14.2.2 XSL 样式单文档

更准确地说，当输入时，XSL 变换接受以 XML 文档表示的树形结构，而输出时，则产生也以 XML 文档来表示的新的树形结构。因此，XSL 变换部分也称为树形结构建立部分。输入和输出的内容必须是 XML 文档。不能使用 XSL 来变换成非 XML 格式（如 PDF、TeX、Microsoft Word、PostScript、MIDI 或其他）或从非 XML 格式进行变换。可使用 XSL 将 XML 变换为一中间格式（如 TeXML），然后使用另外的非 XSL 软件来将这个中间格式变换成期望的格式。HTML 和 SGML 都是介于两者之间的情况，因为它们非常接近于 XML。可使用 XSL 将符合 XML 的结构完整性规则的 HTML 和 SGML 文档变换成 XML 或者相反。但是，XSL 不能处理在大多数 Web 站点上和文档生成系统中遇到的各种各样非结构整洁的 HTML 和 SGML 文档。要牢记的首要问题是，XSL 变换语言对于 XML 到 XML 的变换是可行的，但对于其他方面则不行。

XSL 文档包含一组模板规则和其他规则。模板规则拥有模式（pattern）以及模板（template），模式用来指定模板规则所适用的树形结构，而模板是用来在与此模式匹配时进行输出。当 XSL 处理程序使用 XSL 样式单来格式化 XML 文档时，它对 XML 文档树形结构进行扫描，依次浏览每个子树形结构。当读完 XML 文档中的每个树形结构时，处理程序就把它与样式单中每个模板规则的模式进行比较。当处理程序找到与模板规则的模式相匹配的树形结构时，它就输出此规则的模板。这个模板通常包括一些标记、新的数据和从原 XML 文档的树形结构中复制来的数据。

XSL 使用 XML 来描述这些规则、模板和模式。XSL 文档本身也是 xsl:stylesheet 元素。每个模板规则都是 xsl:template 元素。规则的模式是 xsl:template 元素的 match 特性值。输出模板是 xsl:template 元素的内容。模板中所有的指令都是由一个或另一个 XSL 元素来完成的，而这些指令是用来完成某种动作，如选择输入树形结构中要包括在输出树形结构的部分。这些由元素名上的 xsl: 前缀来标识。没有 xsl: 前缀的元素为结果树部分。



更恰当地说，作为 XSL 指令的所有元素都是 xsl 命名域的一部分。有关命名域将在第 18 章“命名域”中讨论。在那以前，只要了解所有的 XSL 元素的名称都是以 xsl: 开头就可以了。

清单 14-2 显示的是一个非常简单的 XSL 样式单，它有两个模板规则。第一个模板规则与根元素 PERIODIC\_TABLE 相匹配，它使用 html 元素来代替此元素。html 元素的内容是将文档中的其他模板应用于 PERIODIC\_TABLE 元素中所获得的结果。

第二个模板与 ATOM 元素匹配，它用输出文档中的 P 元素代替输入文档中的每个 ATOM 元素。xsl:apply-templates 规则将匹配的源元素的文本插入到输出文档中。因此，P 元素的内容将是包含在相应的 ATOM 元素中的文本（但不是标记）。下面，将更进一步讨论这些元素的精确语法。

清单 14-2：有两个模板规则的周期表 XSL 样式单

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

<xsl:stylesheet

xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">

<xsl:template match="PERIODIC_TABLE">

<html>

<xsl:apply-templates/>

</html>

</xsl:template>

<xsl:template match="ATOM">

<P>

<xsl:apply templates/>

</P>

</xsl:template>

</xsl:stylesheet>
```

### 14.2.3 在何处进行 XML 变换

使用 XSL 样式单可有三种主要方式将 XML 文档变换成其他格式（如 HTML）：

1. XML 文档和相关的样式单都是用于客户端（Web 浏览器）的，然后客户端程序按照样式单中指定的格式变换文档，并将它呈现给用户。
2. 服务器将 XSL 样式单应用于 XML 文档，以便此文档能够变换成其他某种格式（通常为 HTML），并把变换后的文档发送到客户端程序（Web 浏览器）。
3. 第三个程序将原 XML 文档变换成其他某种格式（常常为 HTML）后，才把此文档放置在服务器上。服务器和客户程序只处理变换后的文档。

这三种方法尽管都使用相同的 XML 文档和 XSL 样式，但每一种都使用不同的软件。将 XML 文档发送到 Internet Explorer 5.0 的普通 Web 服务器使用的就是第一种方法。使用 IBM alphaWork 的 XML 功能与服务小程序兼容的 Web 服务器就是第二种方法的例证。使用命令行 XT 程序来将 XML 文档转换成 HTML 文档，然后将 HTML 文档放置在 Web 服务器上，采用的就是第三种方法。但是，这些方法都使用（至少在理论上是如此）相同的 XSL 语言。

本章中，我将重点介绍第三种方法，其主要原因是在撰写本书时，像 James Clark 的 XT 或 IBM 的 LotusXSL 这样的专用转换程序能够最完善、最精确地实现目前的 XSL 规范。此外，该方法提供了与先前的 Web 浏览器和服务器的最广泛的兼容性，而第一种方法要求浏览器比大多数用户使用的更新；第二种方法要求专门的 Web 服务器软件。但是，实际上，要求不同的服务器比要求特定客户来得简单。因为可以安装自己的专门服务器软件，但不能要求用户都安装特定的客户软件。

#### 14.2.4 如何使用 XT

XT 是 Java 1.1 的字符模式的应用程序。要使用它，需要安装与 Java 1.1 兼容的虚拟机，如 Sun 的 Java 开发包（Java Development Kit, JDK）或 Java 的运行时环境（Java Runtime Environment, JRE）、Apple 的 Macintosh Runtime for Java 2.1 (MRJ) 或 Microsoft 的虚拟机。还需要安装符合 SAX 的 XML 分析程序，如 James Clark 的 XP，这也是一个 Java 应用程序。



在撰写本书时，可在 <http://www.jclark.com/xml/xt.html> 站点上找到 XT 程序，而在访问 <http://www.jclark.com/xml/xp/index.html> 处找到 XP 程序。当然，这些 URL 都随时间可能发生变化。甚至无法担保在你读到此书时 XT 就能存在。但是，尽管我在本章中使用 XT，但使用任何 XSL 处理程序（执行 1999 年 4 月 21 日制定的 XSL 规范工作草案的树形结构部分）时，这些实例都能运行。另外的可能性是 IBM alphaWork 的 LotusXSL（可在 <http://www.alphaworks.ibm.com/tech/LotusXSL> 处得到）。当使用执行 XSL 近期草案标准的软件时，这些例子可能运行，也可能不运行，尽管我希望这些例子更接近于近期标准。我将在我自己的 Web 站点（<http://metalab.unc.edu/xml/books/bible/>）上发布任何更新内容。

含有 XT main 方法的 Java 类是 com.jclark.xml.sax.Driver。假设 Java 的 CLASSPATH 环境变量包括 xt.jar 和 sax.jar 文件（这两个文件在 XT 发行版中），那么在命令解释程序的提示符或 DOS 窗口中键入下面的代码，即可运行 XT：

```
C:\>java
```

```
-Dcom.jclark.xml.sax.parser=com.jclark.xml.sax.CommentDriver
```

```
com.jclark.xml.sax.Driver 14-1.xml 14-2.xsl 14-3.html
```

这一命令行运行 java 解释程序，将 com.jclark.xml.sax.parser Java 的环境变量设置为 com.jclark.xml.sax.CommentDriver，后者表示用于解析输入文档的 Java 类的完整名称。此类必须在类路径中。此处我使用 XP 语法分析器，但任何符合 SAX 的语法分析器都可以做到。接下来就是含有 XT 程序的 main() 方法的 Java 类名称（com.jclark.xml.sax.Driver）。最后，是输入 XML 文档（14-1.xml）、输入 XSL 样式单（14-2.xsl）和输出的 HTML 文件（14-3.html）的名称。如果忽略最后一个参数，那么变换后的文档将打印在控制台上。



如果正在使用 Windows，并已安装了 Microsoft Java 虚拟机，就可以使用 XT 的单机可执行版。这样，由于它包括 XP 语法分析器，并且不要求提供 CLASSPATH 环境变量，所以使用起来就稍微容易一些。对于本程序，可简单地将 xt.exe 文件放置在自己的路径中，并键入下列句子：

```
C:\> xt 14-1.xml 14-2.xsl 14-3.html
```

清单 14-2 像第 6 章讨论过的那样将输入文档转换成结构整洁的 HTML 文件。但是，只要编写的样式单支持这种变换，则可从任何 XML 应用程序变换到其他应用程序。例如，可以设想有这样的一个样式单，它把 VML 文档变换到 SVG 文档：

```
% java
```

```
-Dcom.jclark.xml.sax.parser=com.jclark.xml.sax.CommentDriver
```

```
com.jclark.xml.sax.Driver pinktriangle.vml
```

```
VmlToSVG.xml -out pinktriangle.svg
```

当然，尽管其他大多数命令行 XSL 处理程序具有不同的命令行参数和选项，但它们的表现形式相似。如果这些程序不是用 Java 来编写，由于不需要配置 CLASSPATH，使用起来可能稍微容易些。

清单 14-3 显示的是通过 XT 使用清单 14-2 中的 XSL 样式单来运行清单 14-1 时的输出结果。请注意，XT 并不简化它所产生的具有许多空白的 HTML。但这并不重要，因为最终是在 Web 浏览器中浏览此文件，而 Web 浏览器又会将空白截去。图 14-2 显示的是加载到 Netscape Navigator 4.5 中的清单 14-3。由于清单 14-3 显示标准的 HTML，所以不需要具有 XML 功能的浏览器来浏览此文档。

清单 14-3：将清单 14-2 中的样式单应用于清单 14-1 中的 XML 后产生的 HTML

```
<html>
```

```
<P>
```

```
Hydrogen
```

```
H
```

```
1
```

```
1.00794
```

```
20.28
```

```
13.81
```

```
0.0899
```

```
</P>
```

```
<P>
```

```
Helium
```

```
He
```

```
2
```

```
4.0026
```

```
4.216
```

```
0.95
```



0.1785

</P>

</html>

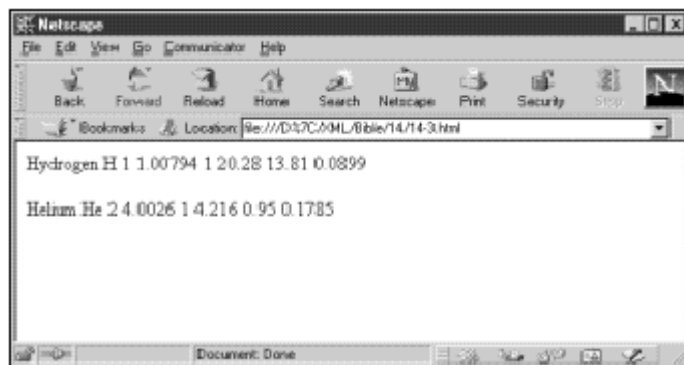


图 14-2 将清单 14-2 中的 XSL 样式单应用于清单 14-1 中的 XML 而生成的页面

### 14.2.5 直接显示带有 XSL 样式单的 XML 文件

无需预处理 XML 文件，就可以向客户端发送 XML 文件和描述如何显示此文件的 XSL 文件。客户程序负责将样式单应用于文档，并按照要求加以显示。这种情况要求客户端所做的工作更多，但服务器的负载要小得多。在这种情况下，XSL 样式单必须将文档变换成客户端能够理解的 XML 应用程序。尽管将来有些浏览器很可能能够处理 XSL 格式化对象，但 HTML 是很有希望的选择方案。

将 XSL 样式单与 XML 文档相链接是很容易的，只需要紧跟在 XML 声明之后插入序言中的 `xml-stylesheet` 处理指令。这种处理指令应有 `text/xsl` 值的 `type` 特性，以及其值为指向此样式单的 URL 的 `href` 特性。例如：

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

```
<?xml-stylesheet type="text/xsl" href="14-2.xsl"?>
```

这也是将 CSS 样式单与文档链接的方法。这里的唯一区别是 `type` 特性具有 `text/xsl` 值，而不是 `text/css` 值。

Internet Explorer 5.0 的 XSL 支持在许多方面与 1999 年 4 月 21 日制定的草案有差异。首先，它期望 XSL 元素放在 `http://www.w3.org/TR/WD-xsl` 命名域中，而不是 `http://www.w3.org/XSL/Transform/1.0` 命名域，尽管 `xsl` 前缀仍然使用。其次，当元素不与任何模板相匹配时，并不执行此元素的缺省规则。因此，在 Internet Explorer 中浏览文档时，需从根元素开始为分级结构中的每个元素提供一个模板。清单 14-4 显示了这种情况。三条规则依次与根节点、根元素 `PERIODIC_TABLE` 和 `ATOM` 相匹配。图 14-3 显示的是使用此样式单将清单 14-1 加载到 Internet Explorer 5.0 中之后的 XML 文档。

清单 14-4：将清单 14-2 调整为可在 Internet Explorer 5.0 下运行的样式单

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

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

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

```
<html>
```

```
<xsl:apply-templates/>
```

```
</html>
```

```
</xsl:template>
```

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

```
<xsl:apply-templates/>
```

```
</xsl:template>
```

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

```
<P>
```

```
<xsl:value-of select="."/></>
```

```
</P>
```

```
</xsl:template>
```

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

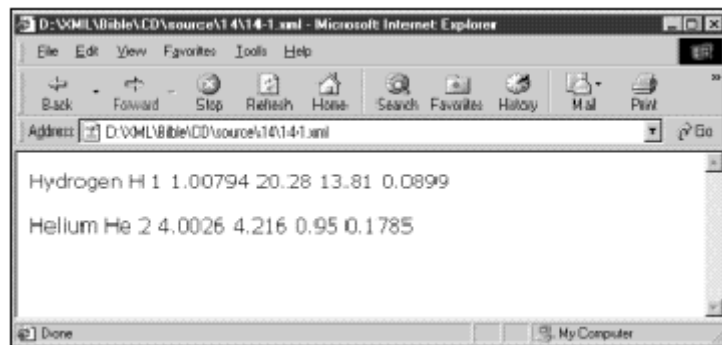


图 14-3 将清单 14-4 中调整过的 XSL 样式单应用于清单 14-1 中的 XML 文档，在 Internet Explorer 5.0 中生成的页面

理想的情况是，相同的 XML 文档既可用于直接显示也可以预处理成 HTML。不幸的是，XT 不接受 <http://www.w3.org/TR/WD-xsl> 命名域，而 IE 5 不接受 <http://www.w3.org/XSL/Transform/1.0> 命名域。由于不同的处理程序在对改进的 XSL 规范各部分的支持方面起到各有所长的作用，所以我们仍然处于这种痛苦的境地。

在本章剩下的部分，在将文档装入浏览器之前，我将把它简单地预处理成 HTML 格式。

## 14.3 XSL 模板

由 `xsl:template` 元素定义的模板规则是 XSL 样式单的最重要的部分。每个模板规则都是一个 `xsl:template` 元素。这些规则将特定的输出与特定的输入相关联。每个 `xsl:template` 元素都有一个 `match` 特性，用来指定要将此模板应用于输入文档的哪个节点。

`xsl:template` 元素的内容是要运用的实际模板。模板可能既包含逐字显示在输出文档中的文本，同时也包含从输入 XML 文档将数据复制到结果的 XSL 指令。因为所有的 XSL 指令都放在 `xsl` 命名域中（即它们都是以 `xsl:` 开头），所以要区分元素（这些元素就是复制到输出的实际数据）和 XSL 指令是很容易的。例如，下面为一个应用于输入树形结构根节点的模板：

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

```
<html>
```

```
<head>
```

```
</head>
```

```
<body>
```

```
</body>
```

```
</html>
```

```
</xsl:template>
```

当 XSL 处理程序读取此输入文档时，它所看到的第一个节点就是根节点。下面的规则与此根节点相匹配，并告诉 XSL 处理程序发送此文本：

```
<html>
```

```
<head>
```

```
</head>
```

```
<body>
```

```
</body>
```

```
</html>
```

这种文本就是结构整洁的 HTML。由于 XSL 文档本身就是 XML 文档，所以其内容（包括模板在内）也必须是结构整洁的 XML。

如果要在 XSL 样式单中使用上面的规则，并且只在 XSL 样式单中使用的话，那么输出就只限于上面的六个标记。（实际上，可压缩为四个等价的标记：`<html> <head/> <body/> </html>`）。这是由于在规则中没有任何指令告诉格式化程序沿树形结构逐级下移，以便寻找与样式单中的模板进一步的匹配。

### 14.3.1 `xsl:apply-templates` 元素

要达到根节点以外的地方，就要告诉格式化引擎处理根节点的子节点。一般来说，为了包括子节点中的内容，需递归处理整个 XML 文档中的节点。可以用来达到此目的的元素就是 `xsl:apply-templates`。把 `xsl:apply-templates` 放在输出模板中，就可以告诉格式化程序把与源元素匹配的每个子元素同样式单中的模板相比较。用于匹配节点的模板本身可能包含 `xsl:apply-templates` 元素，以便搜索与其子节点的匹配。当格式化引擎处理节点时，此节点是作为整个树形结构来看待的。这是树形结构的优点所在。每个部分都是以处理整体相同的方式来处理。例如，清单 14-5 就是使用 `xsl:apply-templates` 元素来处理子节点的 XSL 样式单。

清单 14-5：递归处理根下子节点的 XSL 样式单

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

<xsl:stylesheet

xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">

<xsl:template match="/">

<html>

<xsl:apply-templates/>

</html>

</xsl:template>

<xsl:template match="PERIODIC_TABLE">

<body>

<xsl:apply-templates/>

</body>

</xsl:template>

<xsl:template match="ATOM">

An Atom

</xsl:template>

</xsl:stylesheet>
```

当本样式单应用于清单 14-1 时，将进行以下处理：

1. 将根节点与样式单中的所有模板规则进行比较，它与第一个模板规则相匹配。
2. 写出<html>标记。
3. `xsl:apply-templates` 元素使格式化引擎处理子节点。

- A. 将根节点的第一个子节点（`xmlstylesheet` 指令）与模板规则相比较，此子节点与任何一个模板规则都不匹配，所以不产生任何输出。
- B. 将根节点的第二个子节点（根元素 `PERIODIC_TABLE`）与模板规则相比较，此子节点与第二个模板规则相匹配。
- C. 写出`<body>`标记。
- D. `body` 元素中的 `xsl:apply-templates` 元素使格式化引擎处理 `PERIODIC_TABLE` 的子节点。
  - a. 将 `PERIODIC_TABLE` 元素的第一个子元素（即氢的 `ATOM` 元素）与模板规则进行比较，此子元素与第三个模板规则相匹配。
  - b. 输出文本 `An Atom`。
  - c. 将 `PERIODIC_TABLE` 元素的第二个子元素（即氦的 `ATOM` 元素）与模板规则进行比较，此子元素与第三个模板规则相匹配。
  - d. 输出文本 `An Atom`。
- E. 写出`</body>`标记。
- 4. 写出`</html>`标记。
- 5. 处理完成。

最后的结果为：

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

```
An Atom
```

```
An Atom
```

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

### 14.3.2 select 特性

为了用 `ATOM` 元素的名称（由其 `NAME` 子元素给出）来代替 `An Atom` 文本，需要指定模板应用于 `ATOM` 元素的 `NAME` 子元素。为了选择一组特定的子元素，而不是所有的子元素，可向 `xsl:apply-templates` 提供 `select` 特性，用来指定要选择的子元素。见下面的例子：

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

```
<xsl:apply-templates select="NAME" />
```

```
</xsl:template>
```

select 特性使用同一类型的模式作为 xsl:template 元素的 match 特性。目前，我们坚持使用简单的元素名称；但本章后面有关匹配和选择模式的部分，将讨论 select 和 match 更多的可能用法，如果不存在 select 特性，那么选择所有的子元素。

将上面的规则加到清单 14-5 的样式单，并应用于清单 14-5，其结果如下：

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

```
Hydrogen
```

```
Helium
```

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

## 14.4 使用 xsl:value-of 来计算节点值

xsl:value-of 元素把输入文档中的节点值复制到输出文档中。xsl:value-of 元素的 select 特性指定正在获取的是哪个节点值。

例如，假设要将文字 An Atom 代替为由 NAME 子元素内容给出的 ATOM 元素的名称，可用<xsl:value-of select="NAME"/>代替 An Atom，如下所示：

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

<xsl:value of select="NAME"/>

</xsl:template>
```

然后，当将样式单应用于清单 14-1 时，产生如下文本：

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

Hydrogen

Helium

</body></html>
```

选择其值的项目（本例中的 NAME 元素）是与源节点有关的。源节点是由模板来匹配的项目（本例中的特指 ATOM 元素）。因此，当氢的 ATOM 与<xsl:template match="ATOM">相匹配，氢的 ATOM 的 NAME 元素就由 xsl:value-of 选定了。当氦的 ATOM 与<xsl:template match="ATOM">相匹配时，氦的 ATOM 的 NAME 元素就由 xsl:value-of 选定了。

节点的值总是字符串，有时可能为空字符串。此字符串的精确内容由节点的类型而定。最普通的节点类型为元素，元素节点的值特别简单，就是在元素的开始标记和结束标记之间的所有可析字符串数据（但不是标记！）。例如，清单 14-1 中的第一个 ATOM 元素如下所示：

```
<ATOM STATE="GAS">

<NAME>Hydrogen</NAME>

<SYMBOL>H</SYMBOL>

<ATOMIC_NUMBER>1</ATOMIC_NUMBER>

<ATOMIC_WEIGHT>1.00794</ATOMIC_WEIGHT>

<OXIDATION_STATES>1</OXIDATION_STATES>

<BOILING_POINT UNITS="Kelvin">20.28</BOILING_POINT>

<MELTING_POINT UNITS="Kelvin">13.81</MELTING_POINT>

<DENSITY UNITS="grams/cubic centimeter"><!-- At 300K -->

0.0899
```

</DENSITY>

</ATOM>

元素的值显示如下：

Hydrogen

H

1

1.00794

1

20.28

13.81

0.0899

我通过删除所有的标记和注释后计算出了这些值。包括空格在内的其他一切内容都完整无缺地保留着。其他六个节点类型的值也可以类似的非常明显的方式加以计算。表 14-1 总结了这些值的结果。

表 14-1 节点值

节点类型	值
根节点	根元素的值
元素	包括在元素中的所有可析的字符数据（包括元素的任何后代中的字符数据）
文本	节点的文本；实际上为节点本身
特性	标准化的特性值（详细说明见 XML 1.0 推荐的第 3.3.3 节）；主要为实体还原后的特性值，截去前导和后随的空格；不包括特性名、等号或引号
命名域	用于命名域的 URL
处理指令	处理指令的值；不包括<?或?>以及处理指令名
注释	注释文本，不包括<!--和-->



## 14.5 使用 xsl:for-each 处理多个元素

xsl:value-of 元素只用于能够不含糊地确定要获取哪个节点值的上下文中。如果有多个可能项可供选择，那么只选择第一项。例如，由于普通的 PERIODIC\_TABLE 元素包含一个以上的 ATOM，所以下列的规则较差：

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

<xsl:value-of select="ATOM"/>

</xsl:template>
```

有两种方法可依次处理多个元素。第一种方法已经看到了。只需要按下列方式与 select 特性（它选择想要包括的特定元素）一起使用 xsl:apply-templates：

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

<xsl:apply-templates select="ATOM"/>

</xsl:template>

<xsl:template match="ATOM">

<xsl:value-of select="."/>

</xsl:template>
```

第二个模板中的 select="." 告诉格式化程序取匹配的元素（本例中的 ATOM）的值。

第二种方法是使用 xsl:for-each。xsl:for-each 元素依次处理由其 select 特性选择的每个元素。不过，无需任何附加的模板。例如：

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

<xsl:for-each select="ATOM">

<xsl:value-of select="."/>

</xsl:for-each>

</xsl:template>
```

如果省略 select 特性，那么处理源节点（本例中的 PERIODIC\_TABLE）的所有子节点。

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

<xsl:for-each>

<xsl:value-of select="ATOM"/>

</xsl:for-each>
```

</xsl:template>

## 14.6 匹配节点的模式

`xsl:template` 元素的 `match` 特性支持复杂的语法，允许人们精确地表达想要和不想要与哪个节点匹配。

`xsl:apply-templates`、`xsl:value-of`、`xsl:for-each`、`xsl:copy-of` 和 `xsl:sort` 的 `select` 特性支持功能更加强大的语法的超集，允许人们精确地表达想要和不想要选择哪个节点。下面讨论匹配和选择节点的各种模式。

### 14.6.1 匹配根节点

为了使输出的文档结构整洁。从 XSL 变换的第一个输出内容应为输出文档的根元素。因此，XSL 样式单一般以应用于根节点的规则开始。要在规则中指定根节点，可将其 `match` 特性设置为合适的值。例如：

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

```
<html>
```

```
<xsl:apply-templates/>
```

```
</html>
```

```
</xsl:template>
```

本规则应用于根节点，并且只应用于输入树形结构的根节点。当读取到此根节点时，就输出`<html>`标记，处理根节点的子节点，然后输出`</html>`标记。本规则推翻了根节点的缺省规则。清单 14-6 显示了应用于根节点的带有单一规则的样式单。

清单 14-6：用于根节点的带有单一规则的 XSL 样式单

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

```
<xsl:stylesheet
```

```
xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
```

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

```
<html>
```

```
<head>
```

```
<title>Atomic Number vs. Atomic Weight</title>
```

```
</head>
```

```
<body>
```

```
<table>
```

```
Atom data will go here
```

```
< /table>
```

```
</body>
```

```
</html >
```

```
</xsl:template>
```

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

由于本样式单只为根节点提供一条规则，并且由于规则的模板未指明对子节点进行进一步的处理，因而只是按原样输出，所以在模板中所看到的所有内容都将插入到结果文档中。换句话说，将清单 14-6 中的样式单应用于清单 14-1（或其他任何结构整洁的 XML 文档）中，所获得的结果如下：

```
<html><head><title>Atomic Number vs. Atomic
```

```
Weight</title></head><body><table>
```

```
Atom data will go here
```

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

### 14.6.2 匹配元素名

正如前面介绍的那样，最基本的模式只包含一个元素名，用来匹配所有带有该名的元素。例如，下面的模板与 ATOM 元素相匹配，并将 ATOM 元素的 ATOMIC\_NUMBER 的子元素标成粗体：

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

```
<b><xsl:value-of select="ATOMIC_NUMBER" /><b>
```

```
</xsl:template>
```

清单 14-7 显示的是扩充了清单 14-6 的样式单。首先，在根节点的规则模板中包括了 xsl:apply-templates 元素。此规则使用 select 特性来确保只有 PERIODIC\_TABLE 元素获得处理。

其次，使用 match="PERIODIC\_TABLE" 语句创建了只适用于 PERIODIC\_TABLE 元素的规则。本规则设置周期表的标题，然后应用模板来从 ATOM 元素中生成周期表的主体。

最后，ATOM 规则使用<xsl:apply-templates select="NAME" />、<xsl:apply-templates select="ATOMIC\_NUMBER" />和<xsl:apply-templates select="ATOMIC\_WEIGHT" />，明确地选择 ATOM 元素的 NAME、ATOMIC\_NUMBER 和 ATOMIC\_WEIGHT 子元素。它们都包装在 HTML 的 tr 和 td 元素中，以便最终的结果是与原子量相匹配的原子序数表。图 14-4 显示将清单 14-7 中的样式单应用于整个周期表文档中的输出结果。

对本样式单需要注意的是：在输入文档中的 NAME、ATOMIC\_NUMBER 和 ATOMIC\_WEIGHT 元素的精确顺序是不重要的。它们在输出文档中以选择它们的顺序出现，也就是说首先为原子序数，然后是原子量。相反，在输入文档中，各个原子依字母顺序排序。以后，将会看到如何使用 xsl:sort 元素来改变这个顺序，以便使用更常规的原子序数的顺序来排列原子。

清单 14-7：利用 select 的施用于元素的特定类的模板

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

```
<xsl:stylesheet
```

```
xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
```

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

<html>

<head>

<title>Atomic Number vs. Atomic Weight</title>

</head>

<body>

<xsl:apply-templates select=" PERIODIC_TABLE" />

</body>

</html>

</xsl:template>

<xsl:template match=" PERIODIC_TABLE" >

<h1>Atomic Number vs. Atomic Weight</h1>

<table>

<th>Element</th>

<th>Atomic Number</th>

<th>Atomic Weight</th>

<xsl:apply-templates select=" ATOM" />

</table>

</xsl:template>

<xsl:template match=" ATOM" >

<tr>

<td><xsl:value-of select=" NAME" /></td>

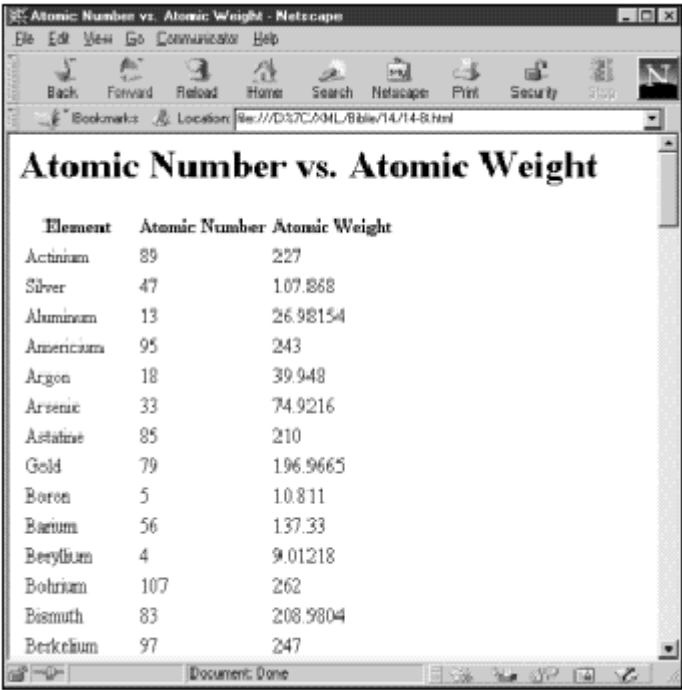
<td><xsl:value-of select=" ATOMIC_NUMBER" /></td>

<td><xsl:value-of select=" ATOMIC_WEIGHT" /></td>

</tr>

</xsl:template>
```

</xsl:stylesheet>



The screenshot shows a Netscape Navigator 4.5 window with the title 'Atomic Number vs. Atomic Weight - Netscape'. The address bar shows a local file path. The main content area displays a table with the following data:

Element	Atomic Number	Atomic Weight
Actinium	89	227
Silver	47	107.868
Aluminum	13	26.98154
Americium	95	243
Argon	18	39.948
Arsenic	33	74.9216
Astatine	85	210
Gold	79	196.9665
Boron	5	10.811
Barium	56	137.33
Beryllium	4	9.01218
Bohrium	107	262
Bismuth	83	208.9804
Berkelium	97	247

图 14-4 Netscape Navigator 4.5 中显示的原子序数与原子量的关系表

14.6.3 使用/字符匹配子节点


在 match 特性中并不局限于当前节点的子节点，可使用/符号来匹配指定的元素后代。当单独使用/符号时，它表示引用根节点。但是，在两个名称之间使用此符号时，表示第二个是第一个的子代。例如，ATOM/NAME 引用 NAME 元素，NAME 元素为 ATOM 元素的子元素。

在 xsl:template 元素中，这种方法能够用来只与某些给定类型的元素进行匹配。例如，下面的模板规则将 ATOM 子元素的 SYMBOL 元素标记为 strong。此规则与不是 ATOM 元素的直系子元素的 SYMBOL 元素无关。

<xsl:template match="ATOM/SYMBOL">

<strong><xsl:value-of select="."/"/></strong>

</xsl:template>

 请记住，本规则选择的是作为 ATOM 元素子元素的 SYMBOL 元素，而不是选择拥有 SYMBOL 子元素的 ATOM 元素。换句话说，在<xsl:value-of select="."/"/>中的. 符号引用的是 SYMBOL，而不是 ATOM。

将模式写成一行的形成，就可以指定更深层的匹配。例如，PERIODIC\_TABLE / ATOM / NAME 选择的是其父为 ATOM 元素（其父为 PERIODIC\_TABLE 元素）的 NAME 元素。

还可以使用\*通配符来代替层次结构中的任意元素名。例如，下面的模板规则应用于 PERIODIC\_TABLE 孙元素的所有 SYMBOL 元素。

<xsl:template match="PERIODIC\_TABLE/\*/SYMBOL">

<strong><xsl:value-of select="."/"/></strong>

```
</xsl:template>
```

最后一点，就如上面所看到的那样，单独的/本身，表示选择文档的根节点。例如，下面的规则应用于文档根元素的所有 PERIODIC\_TABLE 元素。

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

```
<html><xsl:apply templates/></html>
```

```
</xsl:template>
```

虽然 / 引用根节点，但/\* 则引用任意根元素。例如，

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

```
<html>
```

```
<head>
```

```
<title>Atomic Number vs. Atomic Weight</title>
```

```
</head>
```

```
<body>
```

```
<xsl:apply-templates/>
```

```
</body>
```

```
</html>
```

```
</xsl:template>
```

#### 14.6.4 使用//符号匹配子代

有时候，尤其是使用不规则的层次时，更容易的方法就是越过中间节点、只选择给定类型的所有元素而不管这些元素是不是直系子、孙、重孙或其他所有的元素。双斜杠 (//) 引用任意级别的后代元素。例如，下面的模板规则应用于 PERIODIC\_TABLE 的所有 NAME 子代，而不管它们具有何种层次的关系：

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

```
<i><xsl:value-of select="."/></i>
```

```
</xsl:template>
```

周期表实例相当简单，一看就懂，但这种技巧在更深层次，尤其是当元素包含该类的其他元素时（例如 ATOM 包含 ATOM），就显得更加重要。

模式开头的操作符选择根节点的任何子节点。例如，下面的模板规则处理所有的 ATOMIC\_NUMBER 元素，而同时完全忽略其位置：

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

```
<i><xsl:value-of select="."/></i>
```

```
</xsl:template>
```

### 14.6.5 通过 ID 匹配

有人或许想把一特定的样式应用于特定的单一元素中，而不改变该类型的所有其他元素。在 XSL 中实现此目的的最简单的方法是，将样式与元素的 ID 属性选择符（其中包括以单引号括起来的 ID 值）做到这一点。例如，下面的规则使带有 ID 值为 e47 的元素变为粗体：

```
<xsl:template match=" id( 'e47' )" >
```

```
<b><xsl:value-of select="." /></b>
```

```
</xsl:template>
```

当然，上面假设以此方式选择的元素具有在源文档的 DTD 中声明为 ID 类型的特性。但是，通常情况并非如此。首先，许多文档没有 DTD，只不过结构整洁，但不合法。即使有 DTD，也无法确保任何元素都有 ID 类型的特性。可以在样式单中使用 xsl:key 元素，用来把输入文档中的特定特性声明为应该作为 ID 来看待。

### 14.6.6 使用@来匹配特性

正如第 5 章已经看到的那样，@符号根据特性名与特性相匹配，并选择节点。方法很简单，只需在要选择的特性前加上@符号。例如，清单 14-8 显示一样式单，用它来输出一张原子序数和熔点对照的表格。不仅写出了 MELTING\_POINT 的值，而且也写出了 UNITS 特性的值。这是由于<xsl:value-of select="@UNITS"/>所获得的结果。

清单 14-8：使用@来选择 UNITS 特性的 XSL 样式单

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

```
<xsl:stylesheet
```

```
xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
```

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

```
<html>
```

```
<body>
```

```
<h1>Atomic Number vs. Melting Point</h1>
```

```
<table>
```

```
<th>Element</th>
```

```
<th>Atomic Number</th>
```

```
<th>Melting Point</th>
```



```

<xsl:apply-templates/>

</table>

</body>

</html>

</xsl:template>

<xsl:template match="ATOM">

<tr>

<td><xsl:value-of select="NAME"/></td>

<td><xsl:value-of select="ATOMIC_NUMBER"/></td>

<td><xsl:apply-templates select="MELTING_POINT"/></td>

</tr>

</xsl:template>

<xsl:template match="MELTING_POINT">

<xsl:value-of select="." />

<xsl:value-of select="@UNITS"/>

</xsl:template>

</xsl:stylesheet>

```

回想一下，特性节点的值只是此特性的字符串值。一旦应用清单 14-8 中的样式单，ATOM 元素就会格式化如下形成：

```

<tr><td>Hydrogen</td><td>1</td><td>13.81Kelvin</td></tr>

<tr><td>Helium</td><td>2</td><td>0.95Kelvin</td></tr>

```

可以使用各种层次操作符将特性与元素组合起来。例如，[BOILING\\_POINT/@UNITS](#) 引用 BOILING\_POINT 元素的 UNITS 特性。[ATOM/\\*/@UNITS](#) 就能匹配 ATOM 子元素的任何 UNITS 元素。当与模板规则中的特性匹配时，这种做法是特别有用的。必须记住，要匹配的是特性节点，而不是包含它的元素。最常见的错误是，不知不觉地将特性节点与包含它的元素节点搞混淆。例如，请看下面的规则，它试图将模板应用于具有 UNITS 特性的所有子元素：

```

<xsl:template match="ATOM">

<xsl:apply-templates select="@UNITS"/>

</xsl:template>

```

上面语句实际上做的是，将模板应用于 ATOM 元素中并不存在的 UNITS 特性。

也可以使用\*来选择元素的所有特性，例如，[BOILING\\_POINT/@\\*](#)可选择 BOILING\_POINT 元素的所有特性。

#### 14.6.7 使用 comments() 来匹配注释

大多数时候，可能应该完全忽略 XML 文档中的注释。要使注释成为文档的必不可少的部分，确实不是好主意。但是，当不得不选择注释时，XSL 确实提供了选择注释的手段。

为了选择注释，可使用 comment() 模式。尽管此模式有类似函数的圆括号，但实际上决不带任何参数。要区分不同的注释不太容易。例如，回想一下 DENSITY 元素具有如下的形式：

```
<DENSITY UNITS=" grams/cubic centimeter" ><!-- At 300K -->
```

6.51

```
</DENSITY>
```

此模板规则不仅输出密度的值和单位，而且还打印测量密度的条件：

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

```
<xsl:value-of select=" ." />
```

```
<xsl:value-of select=" @UNITS" />
```

```
<xsl:apply-templates select=" comment() " />
```

```
</xsl:template>
```

清单 14-1 使用注释而不是特性或元素来指定条件，就是为了用于本例。实际应用时，决不要将重要信息放在注释中。XSL 允许人们选择注释的唯一真实的理由是，为了用样式单把一种标记语言变换成另一种标记语言，同时又能使注释保持不变。选择注释的任何其他方面的用途都意味着原文档设计得不好。下面的规则匹配所有的注释，并使用 xsl:comment 元素将它们再次复制出来。

```
<xsl:template match=" comment() " >
```

```
<xsl:comment><xsl:value-of select=" ." /></xsl:comment>
```

```
</xsl:template>
```

可是，要注意，用于施加模板的缺省规则对注释无效。因此，遇到注释时，如果要使缺省规则起作用，需要包括 xsl:apply-templates 元素，无论注释放在何处，此元素都能选择注释。

使用层次操作符可以选择特定的注释。例如，下面的规则匹配 DENSITY 元素内部的注释：

```
<xsl:template match=" DENSITY/comment() " >
```

```
<xsl:comment><xsl:value-of select=" ." /></xsl:comment>
```

```
</xsl:template>
```

### 14.6.8 使用 pi() 来匹配处理指令

谈到编写结构化的、智能化的、可维护的 XML 时，处理指令并不比注释好。但是都有一些必需的应用，其中包括将样式单附加到文档上。

pi() 函数选择处理指令。pi() 的参数是放在引号内的字符串，表示要选择的处理指令的名称。如果没有参数，则匹配当前节点的第一个处理指令子节点。但是，可以使用层次操作符。例如，下面的规则匹配根节点的第一个处理指令子节点（很可能是 xml-stylesheet 处理指令）。xsl:pi 元素使用指定的名称和输出文档中的值来插入一个处理指令。

```
<xsl:template match="/pi() ">

<xsl:pi name="xml-stylesheet">

type="text/xsl" value="auto.xsl"

</xsl:pi>

</xsl:template/>
```

下列规则也匹配 xml-stylesheet 处理指令，但是通过其名称来匹配的：

```
<xsl:template match="pi( xml-stylesheet )">

<xsl:pi name="xml-stylesheet">

<xsl:value-of select="."/>

</xsl:pi>

</xsl:template/>
```

事实上，区分根元素和根节点的主要原因之一就是，为了读取和处理序言中的处理指令。尽管 xml-stylesheet 处理指令使用“名称=值”这样的句法，但 XSL 并不把它们当做特性看待，这是因为处理指令不是元素。处理指令的值只是跟在其名称后面的空格和结束符?>之间的所有内容。

用来施加模板的缺省规则并不匹配处理指令。因此，遇到 xml-stylesheet 处理指令时，如果要使缺省规则起作用，需要包括 xsl:apply-templates 元素，此元素在适当的地方匹配缺省规则。例如，下面这个用于根节点的模板确实将模板应用于处理指令：

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

<xsl:apply-templates select="pi()"/>

<xsl:apply-templates select="*/>

</xsl:template>
```

### 14.6.9 用 text() 来匹配文本节点

尽管文本节点的值包括在选择的元素值部分中，但它们作为节点通常被忽视。但是，text() 操作符确实能够明确选择一个元素的文本子元素。尽管这种操作符有圆括号，但不需要任何参数。例如：

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

<xsl:value-of select="text()"/>

</xsl:template>
```

此操作符存在的主要原因是为了用于缺省规则。无论作者是否指定缺省规则，XSL 处理程序必须提供下列的缺省规则：

```
<xsl:template match="text()">

<xsl:value-of select="."/>

</xsl:template>
```

这意味着无论何时将模板应用于文本节点，就会输出此节点的文本。如果并不需要这种缺省行为，可以将其推翻。例如，在样式单中，包括下列空模板规则，将会阻止输出文本节点，除非另外的规则明确地匹配。

```
<xsl:template rmatch="text()">

</xsl:template>
```

#### 14.6.10 使用“或”操作符|

竖线（|）允许一条模板规则匹配多种模式。如果节点与某种模式相匹配，则此节点将激活该模板。例如，下面模板规则与 ATOMIC\_NUMBER 和 ATOMIC\_WEIGHT 元素都匹配：

```
<xsl:template match=" ATOMIC_NUMBER|ATOMIC_WEIGHT" >

<B><xsl:apply-templates/></B>

</xsl:template>
```

也可以在|两边加入空格，这样使代码更清晰。例如：

```
<xsl:template match=" ATOMIC_NUMBER | ATOMIC_WEIGHT" >

<B><xsl:apply-templates/></B>

</xsl:template>
```

还可以顺次使用两个以上的模式。例如，下面的模板规则作用于 A

14.7 选择节点的表达式

在 `xsl:apply-templates`、`xsl:value-of`、`xsl:for-each`、`xsl:copy-of` 和 `xsl:sort` 中，可使用 `select` 特性来精确指定对哪个节点进行操作。此特性值即为表达式（expression）。表达式是前节讨论的匹配模式的超集。也就是说，所有的匹配模式都是选择表达式，但并非所有的选择表达式都是匹配模式。让我们来回顾一下，匹配模式能够使用元素名、子元素、后代以及特性来匹配节点，除此之外，还可以对这些项目进行简单的测试。选择表达式可以使用所有的这些条件来选择节点，而且还可以通过参考父元素、同属元素来选择节点，以及通过更加复杂的测试来选择节点。此外，表达式并不局限于只生成一组节点列表，而且还产生布尔值、数值和字符串。

14.7.1 节点轴

表达式并不局限于指定当前节点的子节点和后代节点。XSL 提供许多轴（axe），使用这些轴可以从相对于当前节点（通常为模板匹配的节点）的树形结构的不同部分进行选择。表 14-2 概述了这些轴及其含义。

表 14-2 表达式的轴

轴	选自于
<code>from-ancestors()</code>	当前节点的父节点、当前节点的父节点的父节点、当前节点的父节点的父节点的父节点，依次类推至根节点
<code>from-ancestors-or-self()</code>	当前节点的后代以及当前节点本身
<code>from-attributes()</code>	当前节点的特性
<code>from-children()</code>	当前节点的直系子节点
<code>from-descendants()</code>	当前节点的子节点、当前节点的子节点的子节点，依次类推
<code>from-descendants-or-self()</code>	当前节点本身及其后代节点
<code>from-following()</code>	起始于当前节点末尾之后的所有节点
<code>from-following-siblings()</code>	起始于当前节点末尾之后并且与当前节点具有同一个父节点的所有节点
<code>from-parent()</code>	当前节点的单一父节点
<code>from-preceding()</code>	当前节点开始之前开始的所有节点
<code>from-preceding-siblings()</code>	当前节点开始之前开始的所有节点并且与当前节点具有同一个父节点的所有节点
<code>from-self()</code>	当前节点



`from-following` 和 `from-preceding` 轴不可靠，可能不会包括在 XSL 的最终发布版中。如果包括在 XSL 的最终发

布版中，其准确的含义可能会改变。

这些轴的功能是选择表 14-2 第二列中列出的节点。圆括号中包括要进一步对此节点列表筛选的选择表达式。例如，可能包括由下列模板规则选择的元素名称：

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

<tr>

<td>

<xsl:value-of select="from-children(NAME)"/>

</td>

<td>

<xsl:value-of select="from-children(ATOMIC_NUMBER)"/>

</td>

<td>

<xsl:value-of select="from-children(ATOMIC_WEIGHT)"/>

</td>

</tr>

</xsl:template>
```

此模板规则匹配 ATOM 元素。当 ATOM 元素匹配时，NAME 元素、ATOMIC\_NUMBER 元素和 ATOMIC\_WEIGHT 都从此匹配的 ATOM 元素的子元素中选择，并作为表的单元格输出。（如果有多个期待的元素 &#0;&#0;例如，三个 NAME 元素 &#0;&#0;那么，只选择第一个。）

from-children() 轴不允许做单独使用元素名不能做的任何事情。实际上，select="ATOMIC\_WEIGHT" 只是 select = "from-children (ATOMIC\_WEIGHT)" 的缩写形式。但是，其他轴更令人感兴趣。

在匹配模式时引用父元素是不合法的，但在选择表达式中引用则是合法。要引用父元素，可使用 from-parent() 轴。例如，下面的规则输出具有 BOILING\_POINT 子元素的 ATOM 元素的值：

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

<P><xsl:value-of select="from-parent (ATOM)"/></P>

</xsl:template>
```

这里匹配的是 BOILING\_POINT 子元素，但输出的是 ATOM 父元素。

有些放射性原子（如钋）其半衰期是如此之短，以致无法测量重要的性质（如沸点和熔点）。所以并非所有的 ATOM 元素都必须有 BOILING\_POINT 子元素。上面的规则可用来只输出实际上有沸点的元素。清单 14-10 是此例的扩展，它匹配 BOILING\_POINT 元素，但实际上使用 from-parent (ATOM) 输出 ATOM 父元素。

清单 14-10：只输出有已知熔点的元素的样式单

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

<xsl:stylesheet

xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">

<xsl:template match="/">

<html>

<body>

<xsl:apply-templates select="PERIODIC_TABLE"/>

</body>

</html>

</xsl:template>

<xsl:template match="PERIODIC_TABLE">

<h1>Elements with known Melting Points</h1>

<xsl:apply-templates select="//MELTING_POINT"/>

</xsl:template>

<xsl:template match="MELTING_POINT">

<P>

<xsl:value-of select="from-parent (ATOM)"/>

</P>

</xsl:template>

</xsl:stylesheet>
```

偶尔，可能需要选择给定类型元素的最近的祖先。使用 from-ancestors() 就可以做到这一点。例如，下面的规则插入最近的 PERIODIC\_TABLE 元素（包含匹配的 SYMBOL 元素）的值。

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

<xsl:value-of select="from-ancestors (PERIODIC_TABLE)"/>
```

```
</xsl:template>
```

`from-ancestors-or-self()` 函数的作用与 `from-ancestors()` 函数相似，所不同的是，如果当前节点与参数类型匹配，那么它的返回值为其本身，而不是真正的祖先。例如，下面的规则匹配所有元素。如果匹配的元素是 `PERIODIC_TABLE`，那么在 `xsl:value-of` 中选择的正是 `PERIODIC_TABLE`。

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

```
<xsl:value-of select="from-ancestors-or-self(PERIODIC_TABLE)"/>
```

```
</xsl:template>
```

#### 14.7.1.1 节点类型

`from-axis()` 函数的参数，除了可以使用节点名称和通配符之外，还可以是下列四个节点类型函数之一：

- `comment()`
- `text()`
- `pi()`
- `node()`

`comment()` 节点类型选择注释节点。`text()` 节点类型选择文本节点。`pi()` 节点类型选择处理指令节点，而 `node()` 节点类型选择任何类型的节点（\*通配符只选择元素节点）。`pi()` 节点类型还有一个可选的参数，用来指定要选择的处理指令的名称。

例如，下面的规则同时使用带有 `node()` 节点类型的 `from-self()`，将匹配的 `ATOM` 元素封装在 `P` 元素中：

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

```
<P><xsl:value-of select="from-self(node())"/></P>
```

```
</xsl:template>
```

这里，选择 `from-self(node())` 与选择 `ATOM` 是不同的。下面的这个规则获取 `ATOM` 元素的 `ATOM` 子元素的值。此值不是匹配的 `ATOM` 元素的值，而是匹配 `ATOM` 元素的一个子元素的另一个 `ATOM` 元素值：

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

```
<P><xsl:value of select="ATOM"/></P>
```

```
</xsl:template>
```

#### 14.7.1.2 层次操作符

可以使用 `/` 和 `//` 操作符来将选择表达式串联在一起。例如，清单 14-11 只将有熔点的那些元素的元素名、原子序数和熔点打印成表。要实现此目的，选择 `MELTING_POINT` 元素的父元素，然后使用 `select="from-parent(*)/from-children(NAME)"` 来查找父元素的 `NAME` 和 `ATOMIC_NUMBER` 子元素。

清单 14-11：熔点与原子序数表

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



```

<xsl:stylesheet

xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">

<xsl:template match="/PERIODIC_TABLE">

<html>

<body>

<h1>Atomic Number vs. Melting Point</h1>

<table>

<th>Element</th>

<th>Atomic Number</th>

<th>Melting Point</th>

<xsl:apply-templates select="from-children(ATOM)"/>

</table>

</body>

</html>

</xsl:template>

<xsl:template match="ATOM">

<xsl:apply-templates

select="from-children(MELTING_POINT)"/>

</xsl:template>

<xsl:template match="MELTING_POINT">

<tr>

<td>

<xsl:value-of

select="from-parent(*)/from-children(NAME)"/>

</td>

<td>

```

```

<xsl:value-of

select="from-parent(*)/from-children(ATOMIC_NUMBER)"/>

</td>

<td>

<xsl:value-of select="from-self(*)"/>

<xsl:value-of select="from-attributes(UNITS)"/>

</td>

</tr>

</xsl:template>

</xsl:stylesheet>

```

这并不是解决这一问题的唯一方法。另外一种可能的方法是使用 `from-preceding-siblings()` 和 `from-following-siblings()` 轴或相对位置（前面或后面）不确定时同时使用两者。用于 `MELTING_POINT` 元素的必要模板规则如下所示：

```

<xsl:template match="MELTING_POINT">

<tr>

<td>

<xsl:value-of

select="from-preceding-siblings(NAME)

| from-following-siblings(NAME)"/>

</td>

<td>

<xsl:value-of

select="from-preceding-siblings(ATOMIC_NUMBER)

| from-following-siblings(ATOMIC_NUMBER)"/>

</td>

<td>

<xsl:value-of select="from-self(*)"/>

```

```
<xsl:value-of select="from-attributes(UNITS)"/>

</td>

</tr>

</xsl:template>
```

14.7.1.3 缩写句法

表 14-2 中的各种 from-axis() 函数对于轻松的打字工作来说过于冗长。XSL 还定义了缩写句法，以便代替最常用的轴，在实际过程中使用更广。表 14-3 显示的是完整句法形式与缩写词的对等关系。

表 14-3 选择表达式的缩写句法

缩写词	完整句法形式
.	from-self (node())
..	from-parent (node())
<i>Name</i>	from-children ( <i>name</i> )
@ <i>name</i>	from-attributes ( <i>name</i> )
//	/from-descendants-or-self (node()) /

使用缩写句法重写清单 14-11，得到清单 14-12。但这两个样式单所获得的输出结果是完全一样的。

清单 14-12：使用缩写句法获得的熔点和原子序数对照表

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

<xsl:stylesheet

xmlns:xsl="http://www.w3.org/XSL/Transform/I.0">

<xsl:template match="/PERIODIC_TABLE">

<html>

<body>

<h1>Atomic Number vs. Melting Point</h1>

<table>

<th>Element</th>

<th>Atomic Number</th>
```

```
<th>Melting Point</th>

<xsl:apply-templates select="ATOM"/>

</table>

</body>

</html>

</xsl:template>

<xsl:template match="ATOM">

<xsl:apply-templates

select="MELTING_POINT"/>

</xsl:template>

<xsl:template match="MELTING_POINT">

<tr>

<td>

<xsl:value-of

select="../NAME"/>

</td>

<td>

<xsl:value-of

select="../ATOMIC_NUMBER"/>

</td>

<td>

<xsl:value-of select="."/>

<xsl:value-of select="@UNITS"/>

</td>

</tr>

</xsl:template>
```

</xsl:stylesheet>

匹配模式可以只使用缩写句法（并非使用所有的缩写句法）。对于选择表达式，只能使用表 14-2 中的 *from-axis()* 函数的完整句法形式。


14.7.2 表达式类型

每个表达式都计算出唯一的值。例如，表达式 3+2 运算值为 5。上面所使用的表达式求出的都是节点集合。但是，在 XSL 中，有如下五种类型的表达式：

- 节点集合类型
- 布尔类型
- 数值类型
- 字符串类型
- 结果树形片段

14.7.2.1 节点集合

节点集合 (node set) 是输入文档的一组节点的列表。表 14-2 中的 *from-axis()* 函数返回包含匹配节点的节点集合。哪些节点处于某一函数返回的节点集合中，这要根据当前节点（也可以认为是上下文节点）、函数的参数而定，当然也依赖于它是哪个函数。



习惯于面向对象语言（如 Java 和 C++）的程序员可能将当前节点看作为调用函数的对象；也就是说，在 a.doSomething(b, c) 中，当前节点为 a。但是，在 XSL 中，当前节点总是明确的；也就是说，按照定义 a 类的文件所规定的形式，更可能写成 doSomething(b, c) 形式。

例如当当前节点为例 14-1 中的 PERIODIC\_TABLE 元素时，表达式 *select="from-children(ATOM)"* 返回的节点集合含有两个 ATOM 元素。当上下文节点为例 14-1 中的 PERIODIC\_TABLE 元素时，表达式 *select="from-children(ATOM)/from-children(NAME)"* 返回的节点集合含有<NAME> Hydrogen </NAME>和<NAME> Helium </NAME>两个元素节点。

上下文节点 (context node) 是上下文节点列表 (context node list) 的一个成员。上下文节点列表是同时都与同一个规则相匹配的元素集合，通常是 *xsl:apply-templates* 或 *xsl:for-each* 调用的结果。例如，当清单 14-12 应用于清单 14-1 时，ATOM 模板调用两次，第一次用于氢原子，第二次用于氦原子。第一次调用时，上下文节点就是氢的 ATOM 元素。第二次调用时，上下文节点就是氦的 ATOM 元素。但是，在这两次调用中，上下文节点列表则是包含氢和氦的 ATOM 元素的集合。

表 14-4 列举了许多既可以作为参数，也可以作为上下文节点对节点集合进行操作的函数。

表 14-4 对节点集合进行操作的函数

函数	返回值类型	返回值
<i>position()</i>	数值	上下文节点列表中上下文节点的位置。列表中的第一个节点其位置为 1
<i>last()</i>	数值	上下文节点集合中的节点数
<i>count (node-set)</i>	数值	在 <i>node-set</i> 参数指明的节点集合中的节点数
<i>id(string)</i>	节点集合	节点集合，其中只有一个元素（在同一个文档的任何位置）

		其 ID 为 <i>string</i> ; 或者空集合（如果任何元素都没有指定的 ID）
<code>idref(<i>node-set</i>)</code>	节点集合	节点集合，包括文档中的某些元素，其 ID 属性为在参数 <i>node-set</i> 中指明节点值中的（以空格分开）记号之一
<code>key(<i>string name</i>, <i>string value</i>)</code>	节点集合	节点集合，包括文档中所有具有指定值的关键字的节点。关键字是使用顶层 <code>xsl:key</code> 元素来设置的
<code>keyref(<i>string name</i>, <i>node set values</i>)</code>	节点集合	节点集合，包括文档中所有具有某种关键字节点，此关键字的值与第二个参数中的节点值相同
<code>doc(<i>string URI</i>)</code>	节点集合	文档或由 URI 引用的文档部分中的节点集合；这些节点从 URI 使用的命名的 anchor 标记或 XPointer 中选择。如果没有命名的 anchor 标记或 Xpointer，那么所指文档的根元素就存在于节点集合中。相对 URI 是相对于输入文档中的当前节点的
<code>docref(<i>node set</i>)</code>	节点集合	节点集合，包括由 URI 引用的，其值为 <i>node set</i> 参数的所有节点
<code>local-part(<i>node set</i>)</code>	字符串	<i>node set</i> 参数中第一个节点的本地部分（命名域前缀后面的所有内容）；当不使用任何参数时可用于获取上下文节点的本地部分
<code>namespace(<i>node set</i>)</code>	字符串	节点集合中第一个节点命名域的 URI；当不使用任何参数时，可用于获得上下文节点的命名域 URI；如果节点处于缺省命名域内，则返回空字符
<code>qname(<i>node set</i>)</code>	字符串	<i>node set</i> 参数中第一个节点的合法名称（可以为前缀和本地部分）；要获得上下文节点的合法名称，可不使用任何参数

<code>generate-id(<i>node set</i>)</code>	字符串	参数 <i>node set</i> 中第一个节点的唯一标识符；不带参数使用时，可生成上下文节点的 ID
---	-----	--



第 18 章“命名域”将讨论命名域 URI、前缀和本地部分。



`doc()` 和 `docref()` 函数有点模糊，特别是如果 URI 只引用非结构完整的 XML 节点或数据片段，就更是如此。细节还要留待 XSL 规范的未来版本加以澄清。

如果向这些函数传递了一个错误类型的参数，那么 XSL 试图将此参数转变成正确的类型；例如，将数字 12 转变成字符串“12”。但是，任何参数都不能转变成节点集合。

position() 函数可用来对元素进行计数。清单 14-13 是一个样式单，它使用<xsl:value-of select = "position"/>，将元素在文档中的位置作为原子名的名称的前缀。

清单 14-13：按照文档中的顺序对原子进行编号的样式单

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

<xsl:stylesheet

xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">

<xsl:template match="/PFRIODIC_TABLE">

<HTML>

<HEAD><TITLE>The Elements</TITLE></HEAD>

<BODY>

<xsl:apply-templates select="ATOM"/>

</BODY>

</HTML>

</xsl:template>

<xsl:template match="ATOM">

<P>

<xsl:value-of select="position()"/>.

<xsl:value-of select="NAME"/>

</P>

</xsl:template>

</xsl:stylesheet>
```

将此样式单应用于清单 14-1 时，其输出结果如下：

```
<HTML><HEAD><TITLE>The Elements</TITLE></HEAD><BODY><P>1.

Hydrogen</P><P>2.

Helium</P></BODY></HTML>
```

#### 14.7.2.2 布尔类型

布尔值为两个值之一：true（真）或 false（假）。XSL 允许将任何类型的数据转变成布尔值。当在可望为布尔值的地方使用了字符串或数值或节点集合时，通常就暗示需要这样做，xsl:if 元素的 test 特性中正是这样情况。根据下列这些规则，也可以使用 boolean() 函数来完成这种变换过程，此函数将任何类型（或如果不提供参数即为上下文节点）的参数转变成布尔值：

- 如果数值为零或 NaN（一种特定的符号，意为 Not a Number，即不是数字，用于表示被零除所获得的结果以及类似的非法操作），则此值为 false，否则为 true
- 空节点集合为 false；所有的其他节点集合为 true
- 空结果片段为 false；所有其他结果片段都为 true
- 零长度字符串为 false；所有其他字符串为 true

使用下列操作符所获得的表达式的结果也可以得到布尔值：

= 等于号

< 小于号（实际使用&lt;）

> 大于号

< = 小于等于（实际使用&lt;=）

> = 大于等于



在特性值中<符号是非法的。因此，必须用&lt;来代替，甚至作为小于运算符时也是如此。

这些运算符最常用于判断是否调用某个规则。选择表达式不仅含有选择某些节点的模式，而且还可以含有判断条件，从而使用此判断条件对所选的节点列表进一步筛选。例如，from-children(ATOM) 选择当前节点的所有 ATOM 子节点。但是，from-children(ATOME[position()=1]) 只选择当前节点的第一个 ATOM 子节点。[position()=1] 为一判断语句，它在节点上测试 ATOM，返回一个布尔结果值，如果当前节点的位置等于 1，则返回的结果为 true，否则为 false。对每个节点的测试都可能有任何判断值。但是，大于 1 的值则是不常见到的。

例如，下面的模板规则通过测试元素的位置是否等于 1，来决定将此规则应用于周期表中第一个 ATOM 元素，而不是后续元素。

```
<xsl:template match="PERIODIC_TABLE/ATOM[position()=1]">
```

```
<xsl:value-of select="."/>
```

```
</xsl:template>
```

下面的模板规则通过测试元素的恢复欠翊筇?，来将此规则应用于非 PERIODIC\_TABLE 第一个子元素的所有 ATOM 元素：

```
<xsl:template match="PERIODIC_TABLE/ATOM[position()>1]">
```

```
<xsl:value-of select="."/>
```

```
</xsl:template>
```



关键字 `and` 和 `or` 根据正常的逻辑规则，将两个布尔表达式进行逻辑组合。例如，假设要将模板应用于 `ATOMIC_NUMBER` 元素，它既不是其父元素的第一个子元素，也不是其父元素的最后一个子元素；也就是说，它就是父元素本身。下面的模板规则使用 `and` 来完成此项工作：

```
<xsl:template  
  
match="ATOMIC_NUMBER[position()=1 and position()=last()]">  
  
<xsl:value of select="."/>  
  
</xsl:template>
```

下面的模板规则通过匹配位置是 1 还是最后一个，应用于其父元素中的第一个和最后一个 `ATOM` 元素：

```
<xsl:template match="ATOM[position()=1 or position()=last()]">  
  
<xsl:value-of select="."/>  
  
</xsl:template>
```

这是逻辑上的“或”，所以如果两个条件都为真，它也将匹配。也就是说，它将既与其父元素的第一个子元素 `ATOM` 元素进行匹配，也与其父元素的最后一个子元素 `ATOM` 元素进行匹配。

在 XSL 中没有 `not` 关键字，但有 `not()` 函数。将操作放在 `not()` 括号中，可实现对操作的取反。例如，下面的模板规则选择除其父元素的第一个子元素外的所有 `ATOM` 元素：

```
<xsl:template match="ATOM[not(position()=1)]">  
  
<xsl:value-of select="."/>  
  
</xsl:template>
```

下面的模板规则选择除其父元素的第一个和最后一个 `ATOM` 子元素外的所有 `ATOM` 元素：

```
<xsl:template match =  
  
"ATOM[not(position()=1 or position()=last())]">  
  
<xsl:value of select="."/>  
  
</xsl:template>
```

不存在“异或”操作符。但巧妙地使用 `not()`、`and` 和 `or` 可以形成“异或”效果。例如，下面的规则要么选择第一个子元素的 `ATOM` 元素，要么选择最后一个子元素的 `ATOM` 元素，但不会同时选择第一个和最后一个子元素的 `ATOM` 元素。

```
<xsl:template  
  
match="ATOM[(position()=1 or position()=last())  
  
and not(position()=1 and position()=last())]">
```

```
<xsl:value-of select="."/>
```

```
</xsl:template>
```

下列还有三个函数返回布尔值：

- `true()` 总是返回 `true`
- `false()` 总是返回 `false`
- `lang(code)` 如果当前节点的语言（由 `xml:lang` 特性给出）与 `code` 参数相同，则返回 `true`

### 14.7.2.3 数值

XSL 的数值为 64 位 IEEE 双精度浮点数。看起来像整数数值（如 42 或 -7000）也是以双精度保存的。非数字值（如字符串和布尔值）根据结果可转化为数字，或使用下面的规则由数值函数将非数字值转化为数字：

- 如果为 `true`，布尔值为 1；如果为 `false`，则为 0。
- 字符串去首尾空白；然后按要求转化成数字；例如，字符串 “12” 转化为数字 12。如果字符串无法作为数字表示，那么就转换为 0。
- 节点集合和结果片段转换成字符串，然后将此字符串转换成数字。

例如，下面的规则只输出自然界不存在的反铀（trans-uranium）元素以及原子序数大于 92（铀原子序数）的元素。于是，`ATOMIC_NUMBER` 产生的节点集合被隐式地转变成当前 `ATOMIC_NUMBER` 节点的字符串值。

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

```
<HTML>
```

```
<HEAD><TITLE>The TransUranium Elements</TITLE></HEAD>
```

```
<BODY>
```

```
<xsl:apply-templates select="ATOM[ATOMIC_NUMBER>92]" />
```

```
</BODY>
```

```
</HTML>
```

```
</xsl:template>
```

XSL 提供了四个标准的算术运算符：

- `+` 加法
- `-` 减法
- `*` 乘法
- `div` 除法（最通用的 `/` 在 XSL 中已用于其他目的）

例如，`<xsl:value-of select="2+2"/>` 将字符串 “4” 插入到输出文档中。这些运算更常用作测试。例如，下面的规则选择原子量大于原子序数两倍的元素：

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

<HTML>

<BODY>

<H1>High Atomic Weight to Atomic Number Ratios</H1>

<xsl:apply-templates

select="ATOM[ATOMIC\_WEIGHT > 2 \* ATOMIC\_NUMBER]"/>

</BODY>

</HTML>

</xsl:template>

下面的模板实际上打印原子量与原子序数的比值：

<xsl:template match="ATOM">

<P>

<xsl:value-of select="NAME"/>

<xsl:value-of select="ATOMIC\_WEIGHT div ATOMIC\_NUMBER"/>

</P>

</xsl:template>

XSL 还提供两个不常用的二进制运算符：

- mod：用于对两个数求余
- quo：用于两个数相除，然后截去小数部分，形成一个整数

XSL 还有对数字进行操作的四个函数：

floor() 返回比此值小的最大整数

ceiling() 返回比此值大的最小整数

round() 将数值四舍五入成最接近的整数

sum() 返回其参数的和

例如下面的模板规则将原子量（各同位素在自然界分布的中子数与质子数之和的加权平均数）减去原子序数（质子数），计算出原子中的中子数，并四舍五入成最接近的整数：

<xsl:template match="ATOM">

<P>

```
<xsl:value-of select="NAME"/>

<xsl:value-of

select="round(ATOMIC_WEIGHT - ATOMIC_NUMBER)"/>

</P>

</xsl:template>
```

下面的规则将所有的原子量相加，然后除以原子的个数，从而计算出表中所有原子的平均原子量：

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

<HTML>

<BODY>

<H1>Average Atomic Weight</H1>

<xsl:value-of

select="sum(from-descendants(ATOMIC_WEIGHT))

div count(from-descendants(ATOMIC_WEIGHT))"/>

</BODY>

</HTML>

</xsl:template>
```

14.7.2.4 字符串

字符串是 Unicode 字符序列。按照下面的准则，使用 string() 函数，就可以将其他数据类型转换成字符串类型：

- 节点集合转换的结果是将集合中的节点值连接在一起。根据表 14-1 所给出的规则，由 xsl:value-of 元素计算出集合中的节点值。
- 结果树形片段（result tree fragment）在转换时，很像是一个元素，并取此假想的元素值。而此假想的元素值是根  
据表 14-1 所给出的规则，由 xsl:value-of 元素计算出的。
- 数字转换成欧洲风格的数字字符串，如 “-12”或“3.1415292”。
- 布尔值的 false 转换成英语单词的“false”；布尔值的 true 转换成英语单词的“ true”。

除了 string( )之外，XSL 还有七个对字符串进行操作的函数。现总结于表 14-5 中。

表 14-5 对字符串进行操作的函数

函数	返回值类型	返回值
starts-with( <i>main_string</i> , <i>prefix_string</i> )	布尔	如果 <i>main_string</i> 以 <i>prefix_string</i> 开始,则为 true;

		否则为 false
Contains( <i>containing_string</i> , <i>contained_string</i> )	布尔	如果 <i>contained_string</i> 参数是 <i>containing_string</i> 参数的一部分，则为 true；否则为 false
Substring-before( <i>string</i> , <i>marker-string</i> )	字符串	从 <i>string</i> 的第一个字符直到第一次出现 <i>marker-string</i> 止（但不包括）的部分
Substring-after( <i>string</i> , <i>marker-string</i> )	字符串	从第一次出现 <i>marker-string</i> 之后到 <i>string</i> 最后一个字符为止的部分
Normalize( <i>string</i> )	字符串	截去 <i>string</i> 首尾空白后的部分，并且一连串的空白以一个空格代替；如果忽略 <i>string</i> 参数，则将上下文节点的字符串值变成为正常字符串
Translate( <i>string</i> , <i>replaced_text</i> , <i>replacement_text</i> )	字符串	返回 <i>string</i> 中由 <i>replacement_text</i> 中的相应字符来代替 <i>replaced_text</i> 中的字符后的结果
concat( <i>string1</i> , <i>string2</i> , )	字符串	将以参数形式传递的所有字符串连接起来，并返回这种连接后的字符串，其顺序为传递时的顺序
format-number( <i>number</i> , <i>format-string</i> , <i>locale-string</i> )	字符串	返回 <i>number</i> 参数格式化后的字符串形式。格式化是按照由 <i>locale-string</i> 参数指定的位置中的 <i>format-string</i> 参数所指定的格式进行的。其工作方式就好像由 Java 1.1 中的 java.text.DecimalFormat 类所进行的格式化一样（请参考 <a href="http://java.sun.com/products/jdk/1.1/docs/api/java.text.DecimalFormat.html">http://java.sun.com/products/jdk/1.1/docs/api/java.text.DecimalFormat.html</a> ）

#### 14.7.2.5 结果树形片段

结果树形片段是 XML 文档的一部分，而不是一个完整的节点或节点集合。例如，使用带有指向元素中间的 URI 的 doc() 函数，其结果可能产生一结果树形片段。有些扩展函数（专门用于特定的 XSL 实现或安装的函数）也可以返回结果树形片段。

由于结果树形片段不是结构整洁的 XML，所以不能用它们来做什么事。实际上，唯一允许的操作是分别使用 string() 和 boolean() 函数，来将它们转换成字符串值或布尔值。

## 14.8 缺省的模板规则

在 XSL 样式单中，十分小心地映射 XML 文档的层次，是很困难的。如果文档不按照固定的、可预料的顺序（如周期表）排列，而是正像许多 Web 网页那样随意地将元素放在一起，这种情况就很难映射 XML 文档的层次。在这些情况下，应有通用的规则，来查找元素并将模板应用于此元素，而不必考虑此元素究竟出现在源文档的何处。

为了使此过程更容易，XSL 定义两个缺省的模板规则，在所有的样式单中都隐性地包括这两个规则。第一个缺省规则将模板应用于所有元素的子元素，以递归的形式，降序排列元素的结构树。这种方式可确保应用于元素的所有模板规则都能够被说明。第二个缺省规则应用于下一个节点，将这些节点的值复制到输出流中。这两个规则共同使用，表示即使没有任何元素的空 XSL 样式单，仍将产生把输入的 XML 文档的原始字符数据作为输出内容的结果。

### 14.8.1 元素的缺省规则

第一个缺省规则应用于任何类型的元素节点或根节点：

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

```
<xsl:apply-templates/>
```

```
</xsl:template>
```

`*/` 是“任何元素的节点或根节点”的缩写形式。本规则的目的，就是要确保所有的元素即使没有受到隐性规则的影响，也都按递归的方式处理。也就是说，除非其他的规则覆盖了本规则（特别是对根元素就是如此），否则所有的元素节点都要处理。

但是，一旦存在任何父元素的隐性规则，那么对于子元素，除非父元素的模板规则有 `xsl:apply-templates` 子元素，否则本规则将无效。例如，按照如下方式，通过匹配根元素，并且既不应用模板，也不使用 `xsl:for-each` 来处理子元素，就可以阻止所有的处理过程：

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

```
</xsl:template>
```

### 14.8.2 文本节点的缺省规则

细心的读者或许已经注意到，有几个例子似乎已输出了有些元素的内容，但实际上没有获得输出的元素值！这些内容是由 XSL 用于以元素内容出现的文本节点的缺省规则提供的。此规则如下：

```
<xsl:template match="text()">
```

```
<xsl:value-of select="."/>
```

```
</xsl:template>
```

这一规则匹配所有的文本节点（`match="text()"`），并输出文本节点（`<xsl:value-of select="."/>`）的值。换言之，此规则将文本从输入复制到输出。

本规则确保最少输出一个元素的文本，即使没有任何规则明确地与此文本匹配。对于特定的元素（从中可或多或少获得元素的文本内容），另一个规则可以覆盖此规则。

### 14.8.3 两个缺省规则的含义

这两个缺省的规则结合在一起，意味着把只有 `xsl:stylesheet` 元素而不包括任何子元素的空样式单（如清单 14-14）应用于 XML 文档时，将把输入元素中所有的 #PCDATA 复制到输出。但是，这种方法不产生任何标记。可是这些规则的优先级很低。因此，任何其他匹配都优先于这两个规则。

清单 14-14: 空的 XML 样式单

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

```
<xsl:stylesheet
```

```
xmlns:xsl= "http://www.w3.org/XSL/Transform/1.0" >
```

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



在 Internet Explorer 5.0 中，对 XSL 产生混淆的最常见的根源之一是，没有提供任何缺省规则。要确保明确地匹配准备输出其内容（包括其后代）的任何节点。

## 14.9 决定输出要包含的内容

在未读取输入文档时，推迟决定输出何种标记往往是必要的。例如，或许想将 FILENAME 元素的内容改为 A 元素的 HREF 特性，或者根据特性的值，将输入文档中的某个元素类型用输出文档中的几个不同元素类型代替。这可以通过使用 `xsl:element`、`xsl:attribute`、`xsl:pi`、`xsl:comment` 和 `xsl:text` 来实现。在这些元素的内容中使用 XSL 指令，并在这些元素的特性值中使用特性值模板，就能改变它们的输出内容。

### 14.9.1 使用特性值模板

特性值模板将数据从输入中的元素内容复制到样式单中的特性值中。从那里，就可将其写入输出中。例如，假定根据要利用下面的基于特性的形式将周期表转换成空的 ATOM 元素：

```
<ATOM NAME=" Vanadium"

ATOMIC_WEIGHT=" 50.9415"

ATOMIC_NUMBER=" 23"

OXIDATION_STATES=" 5, 4, 3, 2"

BOILING_POINT=" 3650K"

MELTING_POINT=" 2163K"

SYMBOL=" V"

DENSITY=" 6.11 grams/cubic centimeter"

/>
```

为此，需要提取输入文档中元素的内容，并将这些内容放在输出文档的特性值中。首先，要完成下列内容：

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

<ATOM NAME=" <xsl:value-of select=' NAME' />"

ATOMIC_WEIGHT=" <xsl:value-of select=' ATOMIC_WEIGHT' />"

ATOMIC_NUMBER=" <xsl:value-of select=' ATOMIC_NUMBER' />"

/>
```

`</xsl:template>`是畸形的 XML。在特性值内部不能使用`<`字符。而且，要编写在大多数一般情况下都能解析此句的软件，是极其困难的。

取而代之的是，在特性值内部，以放在花括号 `{}` 中的数据来代替 `xsl:value-of` 元素。上面的正确编写方式如下：

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

<ATOM NAME=" {NAME}"/>
```



```
ATOMIC_WEIGHT=" {ATOMIC_WEIGHT} />"
```

```
ATOMIC_NUMBER=" {ATOMIC_NUMBER} />"
```

```
/>
```

```
</xsl:template>
```

在输出文档中，{NAME} 由当前节点的 NAME 子元素值所代替。{ATOMIC\_WEIGHT} 由当前节点的 ATOMIC\_WEIGHT 子元素值所代替。{ATOMIC\_NUMBER} 由当前节点的 ATOMIC\_NUMBER 子元素值所代替，等等。

特性值模板的模式比只是一个元素名要复杂。实际上，在特性值模板中，可使用前面讨论过的任何字符串表达式。例如，下面的模板规则以清单 14-1 中使用的形式来选择 DENSITY 元素。

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

```
<BULK_PROPERTY
```

```
NAME=" DENSITY"
```

```
ATOM=" {../NAME} "
```

```
VALUE=" {.} "
```

```
UNITS=" {@UNITS} "
```

```
/>
```

```
</xsl:template>
```

上面的模板规则将特性值模板转换成类似于如下所示的 BULK\_PROPERTY 元素：

```
< BULK_PROPERTY NAME=" DENSITY" ATOM=" Helium" VALUE="
```

```
0.1785
```

```
" UNITS=" grams/cubic centimeter" />
```

特性值并不局限于在一个特性值模板中使用。可以将特性值模板与文字数据或其他特性值模板组合起来使用。例如，下面的模板规则匹配 ATOM 元素，并且将元素名以 H.html、He.html 等格式设置成链接文件，来代替这些元素。此文件名来源于特性值模板 {SYMBOL}，而文字数据提供句号和扩展名。

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

```
<A HREF=" {SYMBOL}.html" >
```

```
<xsl:value-of select=" NAME" />
```

```
</A>
```

```
</xsl:template>
```

在特性值中，可以包含多个特性值模板。例如，下面的模板规则将密度单位作为 VALUE 特性的一部分，而不是使密度单位成为单独的特性：

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

```
<BULK_PROPERTY
```

```
NAME=" DENSITY"
```

```
ATOM=" {./NAME}"
```

```
VALUE=" {.} {@UNITS}"
```

```
/>
```

```
</xsl:template>
```

可在一个 XSL 样式单中将特性值模板用于许多特性的值中。这在 `xsl:element`、`xsl:attribute` 和 `xsl:pi` 元素中特别重要，因为在这些元素中，特性值模板允许设计者决定在读取输入文档之前，在输出文档中准确地显示何种元素、特性或处理指令。不能将特性值模板作为 `select` 或 `match` 特性的值、`xmlns` 特性、提供另一个 XSL 指令元素名的特性或顶层元素（为 `xsl:stylesheet` 直系子元素）特性来使用。



第 18 章“命名域”将讨论 `xmlns` 特性。

### 14.9.2 使用 `xsl:element` 将元素插入到输出文档中

通常，只使用文字元素本身就可以将元素插入到输出文档中。例如，要插入 P 元素，只需要在样式单的适当位置键入 `<P>` 和 `</P>`。但是，偶尔也需要使用输入文档的详细内容，来确定将哪个元素放在输出文档中。例如，当将使用特性来提供信息的源符号集变换成使用元素来提供相同信息的输出符号集时，就是这种情况。

`xsl:element` 元素将元素插入到输出文档中。元素名由 `xsl:element` 元素的 `name` 特性中的特性值模板给出。元素的内容来自于 `xsl:element` 元素的内容，此元素可能包括要插入这些项的 `xsl:attribute`、`xsl:pi` 和 `xsl:comment` 指令（下面讨论所有的指令）。

例如，假设根据 STATE 特性的值，要用 GAS、LIQUID 和 SOLID 元素来代替 ATOM 元素。使用 `xsl:element` 将 STATE 特性值转换为某个元素名，从而只需要一条规则就可以做到这一点。具体作法如下所示：

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

```
<xsl:element name=" {@STATE}" >
```

```
<NAME><xsl:value-of select=" NAME" /></NAME>
```

```
<!-- rules for other children -->
```

```
</xsl:element>
```

```
</xsl:template>
```

使用更为复杂的特性值模板，就可以实现所需的大多数运算。

### 14.9.3 使用 `xsl:attribute` 将特性插入到输出文档中

只使用文字特性，就可以将特性包括在输出文档中。例如，要插入带有 `ALIGN` 特性（其值为 `CENTER`）的 `DIV` 元素，只需在样式单的适当位置处键入 `<DIV ALIGN="CENTER">`和`</DIV>`即可。但是，为了确定特性值，有时甚至是为了确定特性名，常常不得不依赖于从输入文档中读取的数据。

例如，假设要获得一样式单，可选择原子名，并把这些原子名格式化为与 `H.html`、`He.html`、`Li.html` 等等文件的链接：

```
<LI><A HREF="H.html">Hydrogen</A></LI>
```

```
<LI><A HREF="He.html">Helium</A></LI>
```

```
<LI><A HREF="Li.html">Lithium</A></LI>
```

在输入文档中，每个不同的元素都有一个不同的 `HREF` 特性值。`xsl:attribute` 元素计算特性名和值，并将它插入到输出文档中。每个 `xsl:attribute` 元素要么是 `xsl:element` 元素的子元素，要么是文字元素。在输出中，`xsl:attribute` 计算出来的特性关联到与其父元素计算出来的元素上。特性名是由 `xsl:attribute` 元素的 `name` 特性指定的。特性值是由 `xsl:attribute` 元素的内容给出的。例如，下面的模板规则将产生上面显示的输出结果：

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

```
<LI><A>
```

```
<xsl:attribute name="HREF">
```

```
<xsl:value-of select="SYMBOL"/>.html
```

```
</xsl:attribute>
```

```
<xsl:value-of select="NAME"/>
```

```
</A></LI>
```

```
</xsl:template>
```

所有的 `xsl:attribute` 元素都必须放在其父元素的任何其他内容之前。在已经开始写出元素内容之后，就不能将特性加到元素中。例如，下面的模板是非法的：

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

```
<LI><A>
```

```
<xsl:value-of select="NAME"/>
```

```
<xsl:attribute name="HREF">
```

```
<xsl:value-of select="SYMBOL"/>.html
```

```
</xsl:attribute>
```

```
</A></LI>
```

```
</xsl:template>
```

#### 14.9.4 定义特性集合

经常需要将同一组特性应用于许多不同的元素（既可是同类的，也可以是不同类的）。例如，将样式特性应用于 HTML 表中的每个单元格。要使这一操作更加简单，可使用 `xsl:attribute-set`，在样式单的顶层定义一个或多个特性作为特性集合的成员，然后使用 `xsl:use` 将此特性集合包括在元素中。

例如，下面的 `xsl:attribute-set` 元素定义一个名为 `cellstyle` 的元素，其 `font-family` 特性为 New York、Times New Roman、Times 和 serif，其 `font-size` 特性为 12pt。

```
<xsl:attribute-set name=" cellstyle" >
```

```
<xsl:attribute name=" font-family" >
```

```
New York, Times New Roman, Times, serif
```

```
</xsl:attribute>
```

```
<xsl:attribute name=" font-size" >12pt</xsl:attribute>
```

```
</xsl:attribute-set>
```

然后，用下面的模板规则将这些特性应用于输出文档的 `td` 元素。与 `xsl:attribute` 一样，插入特性集合的 `xsl:use` 元素也必须放在作为 `td` 子元素而加入的任何内容之前。

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

```
<tr>
```

```
<td>
```

```
<xsl:use attribute-set=" cellstyle" />
```

```
<xsl:value-of select=" NAME" />
```

```
</td>
```

```
<td>
```

```
<xsl:use attribute-set=" cellstyle" />
```

```
<xsl:value-of select=" ATOMIC_NUMBER" />
```

```
</td>
```

```
</tr>
```

```
</xsl:template>
```

如果某个元素使用一个以上的特性集合，那么，就将所有集合的所有特性应用于该元素。如果一个以上的特性集合使用不同的值定义相同的特性，那么就使用较为重要集合的特性。重要性相同的多个特性集合定义相同的特性，那么此样式单就会出现错误。

#### 14.9.5 使用 `xsl:pi` 生成处理指令

`xsl:pi` 元素将指令放在输出文档中。处理指令的目标由所需的 `name` 特性指定。`xsl:pi` 元素的内容成为处理指令的内容。例如，下面的规则将 `PROGRAM` 元素用 `gcc` 处理指令代替：

```
<xsl:template select="PROGRAM">

<xsl:pi name="gcc"> -04</xsl:pi>

</xsl:template>
```

输入文档中的 `PROGRAM` 元素由输出文档中的下面的处理指令所代替：

```
<?gcc -04?>
```

若这些指令的结果为纯文本，那么 `xsl:pi` 元素的内容可包括 `xsl:value-of` 元素和 `xsl:apply-templates` 元素。例如，

```
<xsl:template select="PROGRAM">

<xsl:pi name="gcc">-04 <xsl:value-of select="NAME"/></xsl:pi>

</xsl:template>
```

`xsl:pi` 的最常用的用途之一，就是当从 XML 生成 XML 时，用来插入 XML 声明（尽管 XML 声明在技术上并不是处理指令）。例如：

```
<xsl:pi name="xml">version="1.0" standalone="yes"</xsl:pi>
```

`xsl:pi` 元素不能包括 `xsl:element` 和在结果中产生元素和特性的其他指令。此外，它还不能包括在输出文档中插入 `?>` 的任何指令和文字文本，因为这会使处理指令提前结束。

#### 14.9.6 使用 `xsl:comment` 生成注释

`xsl:comment` 元素在输出文档中插入注释。它没有特性。其内容为注释文本。例如，

```
<xsl:template select="ATOM">

<xsl:comment>There was an atom here once.</xsl:comment>

</xsl:template>
```

此规则使用下面的输出代替 `ATOM` 节点：

```
<!--There was an atom here once.-->
```

如果 `xsl:value-of` 元素和 `xsl:apply-templates` 元素指令的结果是纯文本的话，那么 `xsl:comment` 元素的内容可包括这些元素。它不能包括 `xsl:element` 以及在结果中产生元素和特性的其他指令。此外，`xsl:comment` 还不能包括在注释中插入双连字号的任何指令或文字文本。这样在输出文档中会使注释很难看，这种情况是不允许的。

#### 14.9.7 使用 `xsl:text` 生成文本

`xsl:text` 元素将其内容作为文字文本插入到输出文档中。例如，下面的规则将每个 `ATOM` 元素用字符串 “There was an atom here once” 代替。

```
<xsl:template select="ATOM">
```

```
<xsl:text>There was an atom here once.</xsl:text>
```

```
</xsl:template>
```

`xsl:text` 元素用得不多，这是因为在多数情况下，键入文本更容易。但是，`xsl:text` 的确有一个优点。它可以准确地保留空白。当处理诗句、计算机源代码或空白显示具有重要意义的其他信息时，使用 `xsl:text` 是很有用的。

## 14.10 使用 `xsl:copy` 复制当前节点

`xsl:copy` 元素将源代码复制到输出文档中。子元素、特性和其他内容不会自动复制。但是，`xsl:copy` 元素的内容也是选择要复制这些内容的 `xsl:template` 元素。当将文档从某个标记符号集转换成相同的或相近的相关标记符号集时，这种方法通常是有用的。例如，下面的模板规则删除原子的特性和子元素，并用其内容值来代替：

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

```
< xsl:copy>
```

```
<xsl:apply-templates/>
```

```
</xsl:copy>
```

```
</xsl:template>
```

`xsl:copy` 使模板具有的用途之一就是恒等转换；也就是说，可将一文档转换成本身。这种转换与下面类似：

```
<xsl:templdte match="*|@*|comment()|pi()|text()" >
```

```
< xsl:copy>
```

```
<xsl:apply-templates select="*|@*|comment()|pi()|text()" />
```

```
</xsl:copy>
```

```
</xsl:template>
```

可对恒等转换进行稍微调节，以产生相似的文档。例如，清单 14-15 是一样式单，它可去掉文档中的注释而文档的其他部分不受影响。在恒等转换中，去掉 `comment()` 节点的 `match` 和 `select` 特性值，而保留此节点的其他部分就可以产生这种结果。

清单 14-15：从文档中删除注释的 XSL 样式单

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

```
<xsl:stylesheet
```

```
xmlns:xsl="http://www.w3.org/XSL/Transform/1.0" >
```

```
<xsl:template match="* | @* | pi() | text()" >
```

```
< xsl:copy>
```

```
<xsl:apply-templates select="* | @* | pi() | text()" />
```

```
</xsl:copy>
```

```
</xsl:template>
```

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

xsl:copy 只复制源节点。使用 xsl:copy-of, 可以复制其他节点, 可能不止一个。xsl:copy-of 的 select 特性选择要复制的节点。例如, 清单 14-16 是一样式单, 它使用 xsl:copy-of, 只复制有 MELTING\_POINT 子元素的 ATOM 元素, 从而将没有熔点的元素从周期表中去掉。

清单 14-16: 只复制有 MELTING\_POINT 子元素的 ATOM 元素的样式单

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

<xsl:stylesheet

xmlns:xsl=" http://www.w3.org/XSL/Transform/1.0" >

<xsl:template match=" /PERIODIC_TABLE" >

<PERIODIC_TABLE>

<xsl:apply-templates select=" ATOM" />

</PERIODIC_TABLE>

</xsl:template>

<xsl:template match=" ATOM" >

<xsl:apply-templates

select=" MELTING_POINT" />

</xsl:template>

<xsl:template match=" MELTING_POINT" >

<xsl:copy-of select=" .." >

<xsl:apply-templates select=" *|@*|pi()|text()" />

</xsl:copy-of>

</xsl:template>

<xsl:template match=" * | @* | pi() | text()" >

<xsl:copy>

<xsl:apply-templates select=" * | @* | pi() | text()" />

</xsl:copy>

</xsl:template>

</xsl:stylesheet>
```



这是一个从源符号集到同一个符号集的 XSL 转换的例子。不像本章中的大多数例子那样，此例不转换成结构整洁的 HTML。

## 14.11 使用 xsl:number 为节点计数

xsl:number 在输出文档中插入格式化整数。由 expr 特性计算出来的数值,通过四舍五入成最接近的整数,然后根据 format 特性值,对此整数进行格式化,从而获得整数值。为这两个特性提供了恰当的缺省值。例如, 考查清单 14-17 中的 ATOM 元素的样式单。

清单 14-17: 为原子计数的 XSL 样式单

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

<xsl:stylesheet

xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">

<xsl:template match="PERIODIC_TABLE">

<html>

<head><title>The Elements</title></head>

<body>

<table>

<xsl:apply-templates select="ATOM"/>

</table>

</body>

</html>

</xsl:template>

<xsl:template match="ATOM">

<tr>

<td><xsl:number expr="position()"/></td>

<td><xsl:value-of select="NAME"/></td>

</tr>

</xsl:template>

</xsl:stylesheet>
```

当此样式单应用于清单 14-1 时, 输出类似如下显示:

```
<html><head><title>The
```

```

Elements</title></head><body><table><tr><td>1</td><td>Hydrogen<
/td></tr>

<tr><td>2</td><td>Helium</td></tr>

</table></body></html>

```

由于氢是其父元素的第一个 ATOM 元素，所以其号码为 1。由于氦是其父元素的第二个 ATOM 元素，所以其号码为 2。（这些号码对应于氢和氦的原子序数，这种对应关系是清单 14-1 的副产品，而清单 14-1 正是以原子序数的顺序进行排列的。）

### 14.11.1 缺省数值

如果使用 `expr` 特性来计算编号，那么这就是所需要的值。但是，如果省略 `expr` 特性，那么源树形结构中的当前节点位置就作为编号来使用。但是，可使用下面三个特性来调整此缺省值：

- `level`
- `count`
- `from`



这三个特性是从以前的不支持较为复杂的表达式 XSL 草案中延续下来的。如果它们完全使你混淆，那么我建议不要去考虑它们，使用 `expr` 来代替。

#### 14.11.1.1 `level` 特性

按缺省行为，当不存在 `expr` 特性时，`xsl:number` 可对源节点的同属节点加以计数。例如，如果对 `ATOMIC_NUMBER` 元素而不是 `ATOM` 元素加以编号，那么由于一个 `ATOM` 元素绝不会有多个 `ATOMIC_NUMBER` 子元素，所以任何一个编号都不会大于 1。尽管文档包含多个 `ATOMIC_NUMBER` 元素，但它们不是同属的。

将 `xsl:number` 的 `level` 特性设置成 `any`，可对与文档中当前节点同类的所有元素加以计数。此情况不仅包括与当前规则相匹配的元素，还包括类型与要求相一致的所有元素。例如，即使只选择气体的原子序数，固体和液体也仍然计数在内（即便固体和液体没有输出也是如此）。看看下面的这些规则：

```

<xsl:template match="ATOM">

<xsl:apply-templates select="NAME"/>

</xsl:template>

<xsl:template match="NAME">

<td><xsl:number level="any"/></td>

<td><xsl:value-of select="."/></td>

</xsl:template>

```

由于 `level` 设置成 `any`，上面的规则对每个新的 `NAME` 元素产生的输出不是从 1 开始，其输出结果如下：

```
<td>1</td><td>Hydrogen</td>
```

```
<td>2</td><td>Helium</td>
```

如果删除 level 特性或设置成缺省的 single 值，那么输出结果如下：

```
<td>1</td><td>Hydrogen</td>
```

```
<td>1</td><td>Helium</td>
```

另一个不大有用的方法将 xsl:number 的 level 特性设置成 multi，以便对当前节点的同属及其祖先（但不是当前节点同属的子节点）加以计数。

#### 14.11.1.2 count 特性

按缺省行为，当没有 expr 特性时，只对与当前节点元素同类的元素加以计数。但可以将 xsl:number 的 count 特性设置成选择表达式，从而指定对什么元素加以计数。例如，下面的规则对 ATOM 的所有子元素进行编号：

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

```
<td><xsl:number count="*" /></td>
```

```
<td><xsl:value-of select="." /></td>
```

```
</xsl:template>
```

应用此规则获得的输出结果如下：

```
<td>1</td><td>Helium</td>
```

```
<td>2</td><td>He</td>
```

```
<td>3</td><td>2</td>
```

```
<td>4</td><td>4.0026</td>
```

```
<td>5</td><td>1</td>
```

```
<td>6</td><td>4.216</td>
```

```
<td>7</td><td>0.95</td>
```

```
<td>8</td><td>
```

```
0.1785
```

```
</td>
```

#### 14.11.1.3 from 特性

from 特性包含 select 表达式，它指定在输入树形结构中以哪个元素开始计数。但仍可以从 1 而不从 2 或 10 或某个其他数字开始计算。

## 14.11.2 数字到字符串的变换

到目前为止，我已经含蓄地假定数值是以 1、2、3 等等表示的；也就是说，用的是以 1 开始的，并且间隔数为 1 的欧洲数字。但并非只有这种情况。例如，书的前言以及前面其他内容的页号通常是以小写罗马数字（如 i、ii、iii、iv 等等）表示的。并且，不同的国家将数字组合在一起、将实数的整数和小数分开以及使用符号来表示各种数字的习惯不同。所有这一切都可以通过下面 `xsl:number` 的五个特性来调整：

- `format`
- `letter-value`
- `digit-group-sep`
- `n-digits-per-group`
- `sequence-src`

### 14.11.2.1 `format` 特性

使用 `format` 特性，可调整 `xsl:number` 使用的编号样式。此特性通常可使用下列值之一：

- `i`：生成小写的罗马数字 i、ii、iii、iv、v、vi、 $\frac{1}{4}$  表示的序列
- `I`：生成大写的罗马数字 I、II、III、IV、V、VI、 $\frac{1}{4}$  表示的序列
- `a`：生成小写的字母 a、b、c、d、e、f、 $\frac{1}{4}$  表示的序列
- `A`：生成大写字母 A、B、C、D、E、F、 $\frac{1}{4}$  表示的序列

例如，下面的规则使用大写罗马数字对原子进行编号：

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

<P>

<xsl:number expr=" position() " format=" I" />

<xsl:value-of select=" ." />

</P>

</xsl:template>
```

改变 `format` 特性的值，可调整在哪个数字（或字母）处开始计数。例如，要在 5 处开始编号，可设置 `format="5"`。要以 iii 开始编写，可设置 `format="iii"`。

使在 `format` 特性中数字的第一位数为 0，即可指定以 0 开始的十位数编号方式。例如，设置 `format="01"`，可生成序列号为 01、02、03、04、05、06、07、08、09、10、11、12、 $\frac{1}{4}$ 。这里将数字排成一列是很有用的。

### 14.11.2.2 `letter-value` 特性

`letter-value` 特性区别是将字母翻译为数字还是翻译为字母。例如，如果要想使用 `format="I"`，获得一个 I、J、K、L、M、N、... 序列，而不是 I、II、III、IV、V、VI、... 序列，则应将 `letter-value` 特性设置为关键字 `alphabetic`。关键字 `other` 指定数字序列。例如

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

<P>
```

```

<xsl:number expr=" position() "

format=" I" letter-value=" alphabetic" />

<xsl:value-of select=" ." />

</P>

</xsl:template>

```

#### 14.11.2.3 Group Separator 特性

在美国，我们倾向于使用逗号将每三个数字作为一组，来写出大数字，如 4,567,302,000。但是，在许多语言和国家里，而是使用句号或空格来分隔各组；例如，4.567.302.000 或 4 567 302 000。而且，在有些国家，习惯将大数字分成每四个一组，而不是三个一组；例如 4,5673,0000。如果处理可能包括几千或更多项的很长序列时，就需要考虑这些问题。

digit-group-sep 特性指定用于数字组之间的分组分隔符。n-digits-per-group 特性指定每组中使用的数字个数。一般来说，应将这些特性随语言一起指定。例如：

```

<xsl:number digit-group-sep=" " />

```

#### 14.11.2.4 sequence-src 特性

最后一点，如果要使用非正常的序列（像 1-1-1999、1-2-1999、1-3-1999、... 日期字符串列表，或者像 10、20、30、40、... 间隔为 10 的列表），可以将此列表（以尖括号 4 斐诃懒 5.奈牡抵小 sequence-src 特性的值表示该文档的相对或绝对的 URL。例如：

```

<xsl:number sequence-src=" 1999.txt" />

```

## 14.12 对输出元素排序

`xsl:sort` 元素将输出元素按不同于输入文档中的顺序进行排序。`xsl:sort` 元素作为 `xsl:apply-templates` 或 `xsl:for-each` 的子元素出现。`xsl:sort` 元素的 `select` 特性定义关键字,用来按照 `xsl:apply-templates` 或 `xsl:for-each` 对元素的输出进行排序。

在缺省情况下,以关键值的字母顺序进行排序。如果在给定的 `xsl:apply-templates` 或 `xsl:for-each` 元素中,存在一个以上的 `xsl:sort` 元素,那么输出内容首先按第一个关键字进行排序,然后按第二个关键字进行排序,依次类推。如果任何元素的比较结果是一样的,那么就按源文档的顺序输出。

例如,假设在一文件中,全部都是以字母顺序排列的 `ATOM` 元素。为了要按原子序数进行排序,可使用清单 14-18 中的样式单。

清单 14-18: 按原子序数排序的 XSL 样式单

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

<xsl:stylesheet

xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">

<xsl:template match="PERIODIC_TABLE">

<html>

<head>

<title>Atomic Number vs. Atomic Weight</title>

</head>

<body>

<h1>Atomic Number vs. Atomic Weight</h1>

<table>

<th>Element</th>

<th>Atomic Number</th>

<th>Atomic Weight</th>

<xsl:apply-templates>

<xsl:sort select="ATOMIC_NUMBER"/>

</xsl:apply-templates>

</table>
```

```
</body>

</html>

</xsl:template>

<xsl:template match="ATOM">

<tr>

<td><xsl:apply-templates select="NAME"/></td>

<td><xsl:apply-templates select="ATOMIC_NUMBER"/></td>

<td><xsl:apply-templates select="ATOMIC_WEIGHT"/></td>

</tr>

</xsl:template>

</xsl:stylesheet>
```

图 14-5 显示的结果表明了以字母顺序排序的局限。原子序数为 1 的氢是第一个元素。但是第二个元素不是原子序数为 2 的氦，而是原子数为 10 的氖。尽管按数字 10 排在 9 之后，但按照字母，10 却在 2 之前。

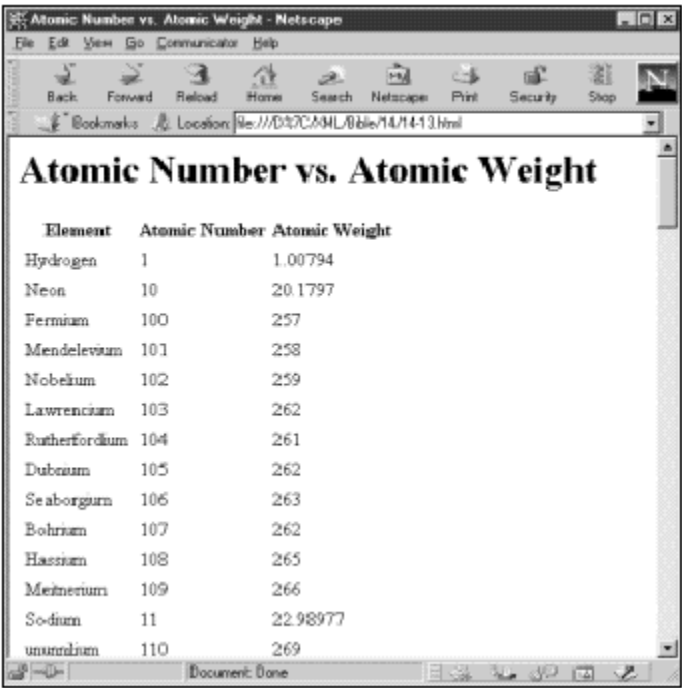


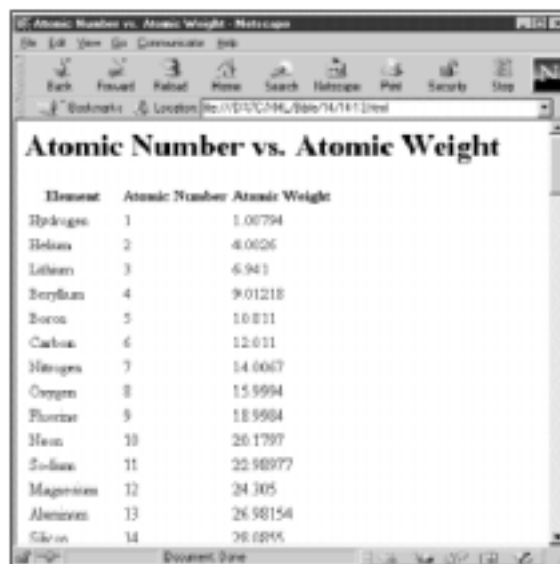
图 14-5 按原子序数的字母顺序排序的原子

但是，通过指导可选的 data-type 特性设置为 number，即可调整排列顺序。例如，

```
<xsl:sort data-type="number" select="ATOMIC_NUMBER"/>
```

图 14-6 显示了正确排序的元素。





Element	Atomic Number	Atomic Weight
Hydrogen	1	1.00794
Helium	2	4.0026
Lithium	3	6.941
Beryllium	4	9.01218
Boron	5	10.811
Carbon	6	12.011
Nitrogen	7	14.0067
Oxygen	8	15.9994
Fluorine	9	18.9984
Neon	10	20.1797
Sodium	11	22.98977
Magnesium	12	24.305
Aluminum	13	26.98154
Silicon	14	28.0855

图 14-6 以数字顺序对原子序数进行排序的原子

按照下面的方法，将 order 特性设置为 descending，即可使排列顺序从缺省的升序改为降序：

```
<xsl:sort order="descending"
```

```
sort="number"
```

```
select="ATOMIC_NUMBER"/>
```

这样就使元素从最大到最小的原子序数进行排列，所以氢现在处于表的最后。

以字母顺序进行的排序自然依赖于字母表。lang 特性可设置关键字的语言。此特性的值应是一种 ISO 639 语言码，如对于英语为 en。



这些值与 xml:lang 特性支持的值相同，这在第 10 章“DTD 中特性声明”中已经讨论过。

最后，可将 case-order 特性设置为 upper-first 或 lower-first 两个值当中的一个，以指定大写字母是排在小写字母之前，还是反过来。缺省状况依赖于语言。

## 14.13 CDATA 和<符

标准的 XSL 无法在输出文档中插入原始的、未转义的不是标记一部分的 < 符号。原始的小于号使输出文档结构混乱，这是 XSL 不允许的。作为一种替代方法，如果使用如&#60;的字符引用或实体引用&lt;来插入<符号，格式化程序将插入&lt;或可能为&#60;。

当将 JavaScript 嵌入到页面中时，由于 JavaScript 使用 < 符号表示数字的小于，而不是表示标记的开始处，这时，这种问题就变得重要。

但是，还是可在输出文档中插入原始的、未转义的 > 和 >= 符号的。因此，如果输出文档需要包含数字比较关系的 JavaScript，那么可颠倒操作数的顺序，将小于比较关系重写成大于等于比较关系。同样，可将小于等于比较关系重写成大于比较关系。例如，下面为几行 JavaScript 代码，在我的很多 Web 网页中使用了这些代码：

```
if (location.host.toLowerCase().indexOf("sunsite") < 0) {  
  
location.href="http://metalab.unc.edu/xml/";  
  
}
```

由于在前两行中使用了小于号，致使这些行结构混乱。但这些语句与下面的这些语句是完全等效的：

```
if (0 > location.host.toLowerCase().indexOf("sunsite")) {  
  
location.href="http://metalab.unc.edu/xml/";  
  
}
```

如果将布尔操作符组合起来进行多重测试，那么可能需要将逻辑“和”改为逻辑“或”。例如，下面两行 JavaScript 非常有效地用来测试页面的位置既不在 metalab 处，也不在 sunsite 处：

```
if (location.host.toLowerCase().indexOf("metalab") < 0  
  
&& location.host.toLowerCase().indexOf("sunsite") < 0) {  
  
location.href="http://metalab.unc.edu/xml/";  
  
}
```

由于在前两行中使用了小于号，致使这些语句结构混乱。但下面的这些行也是用来测试页面是在 metalab 上还是在 sunsite 上，与上面的代码行是完全等效的：

```
if (0 > location.host.toLowerCase().indexOf("metalab")  
  
|| 0 > location.host.toLowerCase().indexOf("sunsite")) {  
  
location.href="http://metalab.unc.edu/xml/";  
  
}
```



也可以将这种令人不快的 JavaScript 放在独立的文档中，并从 SCRIPT 元素的 SRC 特性中与之进行链接。但是，

这在 Internet Explorer 4 和 Netscape Navigator 3 之前的版本中是不可靠的。

出于简化的目的，在输出文档中 CDATA 部分是不允许的。CDATA 部分总是可以用带有 Unicode 转义（escape）的等价字符集合来代替出问题的 < 号和 & 号。CDATA 完全是为人类手工编写 XML 文件提供的便利。计算机程序，如 XSL 格式化程序并不需要 CDATA 部分。



为向输出文档中插入 CDATA 部分，包括在 Internet Explorer 5.0 中的 XSL 格式化程序的确支持非标准的 `xsl:cdata` 元素。但是，这一特点未必会加入到标准的 XSL 中，甚至会从将来的 Internet Explorer 版本中将此功能删除。

## 14.14 方式

有时，要在输出文档中多次地包括源文档中的相同内容。要达到此目的是很容易的：只需多次地应用模板，在每个要使数据出现的地方应用一次。但是，假如要在不同的地方对数据进行不同的格式化，那怎么办呢？这是个比较棘手的问题。

例如，若要使处理周期表的输出文档形成与 100 个更详细描述各原子信息的链接。在此情况下，输出文档的开始很可能如下：

```
<UL>

<LI><A HREF=" #Ac" >Actinium</A></LI>

<LI><A HREF=" #Al" >Aluminum</A></LI>

<LI><A HREF=" #Am" >Americium</A></LI>

<LI><A HREF=" #Sb" >Antimony</A></LI>

<LI><A HREF=" #Ar" >Argon</A></LI>
```

¼

在文档的后面，出现真正的原子的描述，格式化后与下面的类似：

```
<H3><A NAME=" Al" >Aluminum</A></H3><P>

Aluminum

26.98154

13

3

2740

933.5

Al

2.7

</P>
```

无论何时自动生成超文本的目录或索引，使用这类方法都是很普遍的。原子的 NAME 在目录中必须格式化或与文档主体中不同的格式。为此，需要在文档的不同地方将两个不同的规则应用于 ATOM 元素。此解决办法是把每个不同的规则给予 mode 特性。然后设置 xsl-apply-templates 元素的 mode 特性来选择准备应用的模板。

清单 14-19：在两个不同地方使用 mode 来对相同数据进行不同的格式化的 XSL 样式单

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

```
<xsl:stylesheet

xmlns:xsl=" http://www.w3.org/XSL/Transform/1.0" >

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

<HTML>

<HEAD><TITLE>The Elements</TITLE></HEAD>

<BODY>

<H2>Table of Contents</H2>

<UL>

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

</UL>

<H2>The Elements</H2>

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

</BODY>

</HTML>

</xsl:template>

<xsl:template match=" ATOM" mode=" toc" >

<LI><A>

<xsl:attribute name=" HREF" >#{xsl:value-of

select=" SYMBOL" /}</xsl:attribute>

<xsl:value-of select=" NAME" />

</A></LI>

</xsl:template>

<xsl:template match=" ATOM" mode=" full" >

<H3><A>

<xsl:attribute name=" NAME" >

<xsl:value-of select=" SYMBOL" />
```

</xsl:attribute>

<xsl:value-of select=" NAME" />

</A></H3>

<P>

<xsl:value-of select=" ." />

</P>

</xsl:template>

</xsl:stylesheet>

## 14.15 使用 `xsl:variable` 定义常数

命名的常数有助于代码的整洁；可以用简单的名称和引用来代替常用的样板文本；简单地改变常数定义，就能很容易地调整多处出现的样板文本。

`xsl:variable` 元素定义命名的字符串，以便借助于特性值模板用于样式单中的其他地方。`xsl:variable` 是一空元素，是 `xsl:stylesheet` 的直系子元素。它只有唯一的一个特性 `&#0;&#0;name`，此特性提供引用变量的名称。`xsl:variable` 元素的内容作为替换文本。例如，下面的 `xsl:variable` 元素定义名为 `copy99` 和值为 `Copyright 1999 Elliotte Rusty Harold` 的变量：

```
<xsl:variable name="copy99">
```

```
Copyright 1999 Elliotte Rusty Harold
```

```
</xsl:variable>
```

为了访问此变量的值，可将美元符作为前缀加到此变量名前。要在特性中插入此符号，可使用特性值模板。例如：

```
<BLOCK COPYRIGHT="{ $copy99 }">
```

```
</BLOCK >
```

还可以使用 `xsl:value-of`，将变量的替换文本以文本的形式插入到输出文档中：

```
<xsl:value-of select="$copy99"/>
```

`xsl:variable` 的内容可以含有包括其他 XSL 指令的标记。这意味着可根据其他信息（包括其他变量的值）来计算变量的值。但是，变量不能以直接或间接的方式递归地引用其自身。例如，下面的例子是错误的：

```
<xsl:variable name="GNU">
```

```
<xsl:value-of select="$GNU"/> s not Unix
```

```
</xsl:variable>
```

同样，两个变量不能像下面这样循环地相互引用：

```
<xsl:variable name="Thing1">
```

```
Thing1 loves <xsl:value-of select="$Thing2"/>
```

```
</xsl:variable>
```

```
<xsl:variable name="Thing2">
```

```
Thing2 loves <xsl:value-of select="$Thing1"/>
```

```
</xsl:variable>
```

## 14.16 命名模板

变量只限于基本的文本和标记。XSL 提供了功能更强大的宏工具，可以封装标准的标记和改变数据的文本。例如，假定要将原子的原子序数、原子量和其他关键值分别作为表的单元格，以小型的、粗体的蓝色 Times 字体来格式化。换句话说，要获得类似于下面的输入结果：

```
<td>

<font face="Times, serif" color="blue" size="2">

<b>52</b>

</font>

</td>
```

当然，还可以在模板规则中包含类似于下面的所有内容：

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

<td>

<font face="Times, serif" color="blue" size="2">

<b>

<xsl:value-of select="."/>

</b>

</font>

</td>

</xsl:template>
```

这些标记可作为其他模板，或作为其他规则中使用的模板的一部分而重复使用。当详细的标记变得更为复杂时，当标记出现于样式单中的几个不同地方时，可将它转换成命名的模板。命名的模板与变量类似，但能够包括从应用模板的位置获得的数据，而不是仅仅插入固定的文本。

xsl:template 元素有 name 特性，使用此特性，可隐性地调用该元素，甚至在非间接地应用此元素时也是如此。例如，下面显示的是用于给上面模式命名的模板：

```
<xsl:template name="ATOM_CELL">

<td>

<font face="Times, serif" color="blue" size="2">

<b>

<xsl:value-of select="."/>
```



</b>

</font>

</td>

</xsl:template>

宏中间的<xsl:value-of select="."/>元素被替换为调用此模板的当前节点的内容。

xsl:call-template 元素出现在模板规则的内容中，必须有 name 参数，用来对此元素要调用的模板进行命名。处理后，xsl:call-template 元素被它命名的 xsl:template 元素的内容所代替。例如，现在我们使用 xsl:call-template 元素来调用给模板命名的 ATOM\_CELL，那么可按下列方法重写 ATOMIC\_NUMBER 规则：

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

```
<xsl:call-template name="ATOM_CELL"/>
```

```
</xsl:template>
```

这种相当简单的例子仅省掉了几行代码，但模板越复杂，并且重复使用的次数越多，样式单的复杂程度降低得就越大。命名的模板正如变量一样，还有提取样式单中的通用模式的优点，所以可作为一个模板来编辑。例如，如果要原子序数、原子量和其他关键值的颜色由蓝色改变为红色，那么只需要在命名模板中对此改变一次即可。不必在每个分立的模板规则中单独改变此颜色。这有助于在较长的开发过程中，使样式保持更大的一致性。

#### 14.16.1 参数

对命名模板的每一次分开调用，都可将参数传递给模板，以便定制其输出内容。在 xsl:template 元素中，参数是由 xsl:param-variable 子元素来表示的。在 xsl:call-template 元素中，参数是由 xsl:param 子元素来表示的。

例如，假定要将每个原子单元格链接到一特定的文件中。其输出类似于下列情景：

<td>

<font face="Times, serif" color="blue" size="2">

<b>

<a href="atomic\_number.html">52</a>

</b>

</font>

</td>

其诀窍是，由于对模板的每次分开调用都会引起 href 特性的值发生变化，所以必须从调用模板的位置将 href 特性的值传递过去。

<td>

```
<font face=" Times, serif" color=" blue" size=" 2" >
```

```
<b>
```

```
<a href=" atomic_weight.html" >4.0026</a>
```

```
</b>
```

```
</font>
```

```
</td>
```

支持此种情况的模板与下列代码类似：

```
<xsl:template name=" ATOM_CELL" >
```

```
<xsl:param-variable name=" file" >
```

```
index.html
```

```
</xsl:param-variable>
```

```
<td>
```

```
<font face=" Times, serif" color=" blue" size=" 2" >
```

```
<b>
```

```
<a href=" {$file}" ><xsl:value-of select=" ." /></a>
```

```
</b>
```

```
</font>
```

```
</td>
```

```
</xsl:template>
```

`xsl:param-variable` 元素的 `name` 特性给参数起个名称（如果有多个参数则更为重要），如果调用过程不提供值的话，那么 `xsl:param-variable` 元素的内容就为要使用的这个参数提供一个缺省值。（这个缺省值还可以使用 `expr` 特性，以字符串表达式的形式给出，与 `xsl:variable` 完全一样。）

当调用此模板时，`xsl:call-template` 元素的 `xsl:param` 子元素使用其 `name` 特性来识别参数、使用其内容来给参数提供一个值的方法，从而提供该参数的值。例如：

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

```
<xsl:call template macro=" ATOM_CELL" >
```

```
<xsl:param name=" file" >atomic_number.html</xsl:param>
```

```
<xsl:value-of select=" ." />
```

```
</xsl:call-template>
```

```
</xsl:template>
```

这是一个相当简单的例子，但复杂得多的命名模板是存在的。例如，为了用于许多不同样式单（每种样式单一定要单独改变网页作者名字、网页标题和版权日期几个参数）的输入，很可能需要定义 Web 站点上网页的页眉和页脚宏。

## 14.17 删除和保留空白

读者可能已经注意到，到目前为止，所有输出实例的格式化方式都有点奇怪。造成这种现象的原因是，源文档需要将长行划分成多行，以便适合本书页边距的要求。不幸的是，往输入文档中增加额外的空白，就会带到输出文档中。对于计算机来说，毫无意义的空白的具体内容并不重要，但对于人来说，这些空白内容就令人困惑。

像 ATOMIC\_NUMBER 或 DENSITY 元素那样，用于文本节点的缺省行为就是保留所有的空白。常见的 DENSITY 元素看起来如下面那样：

```
<DENSITY UNITS="grams/cubic centimeter"><!-- At 300K -->
```

7.9

```
</DENSITY>
```

当取其值时，值中就会包括首、尾空白（如下所示），尽管这个空白在此处只是用来满足打印页面的要求，但没有什么实际意义：

7.9

但是，有一种例外的情况。如果文本节点只含有空白，没有其他文本，那么这个空白就认为是毫无意义，并被删除。但对此例外还有一种例外：如果文本先辈的 `xml:space` 特性保存有值，那么就不会删除此文本，除非更近的先辈的 `xml:space` 特性具有缺省值。（这种情况听起来有点复杂，但实际上很简单。所有的一切都说明，可忽略只含有空白的文本节点，除非这些文本节点明确地设置成有意义的空白。对于其他情况，空白被保留。）

如果文档中的任何元素都不保留空白，那么可设置 `xsl:stylesheet` 元素的 `default-space` 特性为 `strip`，所有的首尾空白在从文本的节点中删除之后，才输出这些节点文本。对于周期表来说，这最容易实现。例如：

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
```

```
default-space="strip">
```

如果想保留所有元素中的空白，可使用 `xsl:strip-space` 元素，用它识别输入文档中指定的元素，如果指定的元素表明文档中的空白毫无意义，就不将此空白复制到输出文档中。`element` 特性识别要截去过剩空白的元素。例如，下面的这些规则加到周期表样式单中，可避免过多的空白：

```
<xsl:strip-space element="DENSITY"/>
```

```
<xsl:strip-space element="BOILING_POINT"/>
```

```
<xsl:strip-space element="MELTING_POINT"/>
```

`xsl:preserve-space` 元素与 `xsl:strip-space` 元素相反。其 `element` 特性命名的元素表示其空白应保留。例如：

```
<xsl:preserve-space element="ATOM"/>
```

样式单内部的空白（正好与输出 XML 文档中的空白相反）是毫无意义的，在缺省情况下简化为一个空格。这种情况是可以避免的：只需将文字空白放在 `xsl:text` 元素之间。例如：

```
<xsl:template select="ATOM">
```

```
<xsl:text> This is indented exactly five spaces. </xsl:text>
```

```
</xsl:template>
```

处理空白的一个一劳永逸的方法就是将 `indent-result` 特性与根 `xsl:stylesheet` 元素相关联。如果此特性的值为 `yes`，那么就允许处理程序将多余的空白插入到（而不是删除）输出文档中，以便使输出文档看起来好看一些。这包括缩排和行分隔符。例如：

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

```
<xsl:stylesheet xmlns:xsl=http://www.w3.org/XSL/Transform/1.0
```

```
indent result="yes">
```

```
<!-- usual templates and such go here... -->
```

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

如果生成的是 HTML，指定 `indent-result="yes"` 就可使输出的文档更具可读性。`indent-result` 的缺省值为 `no`，这是由于其他非 HTML 的输出格式都可能将空白认为是有意义的。

## 14.18 选择

XSL 提供了根据输入文档来改变输出内容的两个元素。xsl:if 元素根据输入文档中存在的模式，决定是否输出给定的 XML 段。xsl:choose 元素根据输入文档中存在的模式，从几个可能的 XML 段中挑选一个。使用 xsl:if 和 xsl:choose 来完成的大部分任务也需要通过应用适当的模板来实现。但有时，使用 xsl:if 或 xsl:choose 来解决问题会更简单、更有效。

### 14.18.1 xsl:if

xsl:if 元素提供了根据模式来改变输出文档的简单途径。xsl:if 的 test 特性含有选择表达式，用来计算布尔值。如果此表达式为 true，即输出 xsl:if 元素的内容；否则，不输出 xsl:if 元素的内容。例如下面的模板取消所有 ATOM 元素的名称。除列表中的最后一个元素外，在所有的元素后加入一个逗号和一个空格。

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

<xsl:value-of select="NAME"/>

<xsl:if test="not(position()=last())">, </xsl:if>

</xsl:template>
```

本模板确保列表类似于“Hydrogen, Helium”样子，而不是“Hydrogen, Helium, ”的样子。

不存在 xsl:else 或 xsl:else-if 元素。xsl:choose 元素提供了这一功能。

### 14.18.2 xsl:choose

根据几个可能的条件，xsl:choose 元素从几个的输出结果中选择一个。xsl:when 子元素提供各种条件及其相关的输出模板。xsl:when 元素 test 特性为布尔值的选择表达式。如果多个条件都为真，那么只显示第一个为真的条件。如果 xsl:when 元素都不为真，那么显示 xsl:otherwise 子元素的内容。例如，下面的规则根据 ATOM 元素的 STATE 特性是为 SOLID、LIQUID 还是 GAS，来改变输出文档的颜色：

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

<xsl:choose>

<xsl:when test="@STATE='SOLID'">

<P style="color:black">

<xsl:value-of select="." />

</P>

</xsl:when>

<xsl:when test="@STATE='LIQUID'">

<P style="color:blue">

<xsl:value-of select="." />
```

</P>

</xsl:when>

<xsl:when test=" @STATE=' GAS" ' >

<P style=" color:red" >

<xsl:value-of select=" ." />

</P>

</xsl:when>

<xsl:other>

<P style=" color:green" >

<xsl:value-of select=" ." />

</P>

</xsl:other>

</xsl:choose>

</xsl:template>

## 14.19 合并多个样式单

单一 XML 文档可以使用在许多不同的 DTD 中描述的许多不同的标记符号集。有时希望将不同的标准样式单用于那些不同的符号集。但是，也可能还要将样式规则用于特定的文档。xsl:import 和 xsl:include 元素可用来合并多个样式单，以便组织和重新将样式单用于不同的符号集和目的。

### 14.19.1 使用 xsl:import 进行录入

xsl:import 元素为顶级元素，其 href 特性提供导入的样式单的 URI。所有的 xsl:import 元素都必须放在 xsl:stylesheet 根元素中的顶级元素中。例如，下面的这些 xsl:import 元素导入 genealogy.xml 和 standards.xml 样式单。

```
<xsl:stylesheet  
  
  xmlns:xsl=" http://www.w3.org/XSL/Transform/1.0" >  
  
  <xsl:import href=" genealogy.xml" />  
  
  <xsl:import href=" standards.xml" />  
  
  <!-- other child elements follow -->  
  
</xsl:stylesheet>
```

导入的样式单中的规则可能与执行导入的样式单中的规则发生冲突。如果真是这样，那么执行导入的样式单中的规则优先。如果不同的被导入样式单中的两个规则发生冲突，那么最后那个被导入的（如上面例子中的 standards.xml）优先。

xsl:apply-imports 元素与 xsl:apply-templates 有点差别，后者只使用被导入的规则。xsl:apply-imports 元素不使用执行导入的样式单中的任何规则。这样就可以访问被导入的规则，否则被导入的规则就会被执行导入的样式单中的规则所覆盖。除了名称不同外，xsl:apply-imports 与 xsl:apply-templates 有一样的句法，唯一的作用方式差别是它只与被导入样式单中的模板规则匹配。

### 14.19.2 使用 xsl:include 进行包括

xsl:include 元素也是顶级元素，它将另一个样式单复制到当前样式单中它所出现的位置处（更确切地说，它将远程文档中 xsl:stylesheet 元素的内容复制到当前文档中）。它的 href 特性提供要包括的样式单的 URI。xsl:include 元素可放在顶级处于最后那个 xsl:import 元素之后的任何地方。

不像 xsl:import 元素所包括的规则那样，xsl:include 元素所包括的规则与执行包括的样式单中的规则具有同样的优先级，利用这种优先级关系来决定是否从一个样式单到另一个样式单的复制和粘贴。对于格式化引擎来说，被包括的规则与实际存在的规则之间没有任何区别。

### 14.19.3 使用 xsl:stylesheet 在文档中嵌入样式单

可直接将 XSL 样式单包括在使用它的 XML 文档中。实际上，我不推荐这种方法，而且浏览器和格式化引擎也不一定支持这一作法。但是，有几个浏览器和格式化引擎却支持这一作法。为达此目的，xsl:stylesheet 元素必须以文档元素的子元素而不是根元素本身的形式出现。它可能有一个 id 特性，用来为其取唯一的名称，此 id 特性是作为 xsl:stylesheet 处理指令中的 href 特性值的形式出现的，紧跟在的 anchor（锚）标识符（#）之后。清单 14-20 演示此过程：

清单 14-20：在 XML 文档中嵌入的 XSL 样式单

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



```
<?xml-stylesheet type="text/xsl" href="#id(mystyle)"?>
```

```
<PERIODIC_TABLE>
```

```
<xsl:stylesheet
```

```
xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
```

```
id= "mystyle ">
```

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

```
<html>
```

```
<xsl:apply-templates/>
```

```
</html>
```

```
</xsl:template>
```

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

```
<xsl:apply-templates/>
```

```
</xsl:template>
```

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

```
<P>
```

```
<xsl:value-of select="."/"/>
```

```
</P>
```

```
</xsl:template>
```

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

```
<ATOM>
```

```
<NAME>Actinium</NAME>
```

```
<ATOMIC_WEIGHT>227</ATOMIC_WEIGHT>
```

```
<ATOMIC_NUMBER>89</ATOMIC_NUMBER>
```

```
<OXIDATION_STATES>3</OXIDATION_STATES>
```

```
<BOILING_POINT UNITS="Kelvin">3470</BOILING_POINT>
```

```
<MELTING_POINT UNITS="Kelvin">1324</MELTING_POINT>
```

<SYMBOL>Ac</SYMBOL>

<DENSITY\_UNITS="grams/cubic centimeter"><!-- At 300K -->

10.07

</DENSITY>

<ELECTRONEGATIVITY>1.1</ELECTRONEGATIVITY>

<ATOMIC\_RADIUS\_UNITS="Angstroms">1.88</ATOMIC\_RADIUS>

</ATOM>

</PERIODIC\_TABLE>

## 14.20 本章小结

在本章，学习了有关 XSL 变换的内容。包括如下一些内容：

- 可扩展的样式语言（Extensible Style Language，XSL）是由两个独立的 XML 应用程序（分别用于转换和格式化 XML 文档）组成。
- XSL 转换将规则应用于从 XML 文档中读入的一个树形结构中，以便将它转换成一个以 XML 文档编写的输出树形结构中。
- XSL 模板规则是一个带有 `match` 特性的 `xsl:template` 元素。输入树形结构中的节点与不同模板元素 `match` 特性的模式进行比较。当找到匹配时，即输出模板的内容。
- 节点的值是含有节点内容的纯文本（不是标记），可由 `xsl:value-of` 元素获得。
- 可以由两种方法处理多个元素：`xsl:apply-templates` 元素和 `xsl:for-each` 元素。
- `xsl:template` 元素的 `match` 特性值是匹配模式，用以指定模板与哪个节点匹配。
- 选择表达式为 `match` 特性的超集，由 `xsl:apply-templates`、`xsl:value-of`、`xsl:for-each`、`xsl:copy-of`、`xsl:sort` 以及其他各种元素的 `select` 特性所使用。
- 两个缺省的规则将模板应用于元素节点，并取文本节点的值。
- `xsl:element`、`xsl:attribute`、`xsl:pi`、`xsl:comment` 和 `xsl:text` 元素可输出元素、特性、处理指令、注释以及文本，这些输出结果都可以从输入文档中的数据进行运算获得。
- `xsl:attribute-set` 元素定义常用的一组特性，从而使用 `xsl:use` 元素，将这组特性用于不同模板中的多个元素。
- `xsl:copy` 元素将当前输入节点复制到输出文档中。
- `xsl:number` 元素使用 `format` 特性所给出的指定数字格式，将 `expr` 特性中指定的数字插入到输出文档中。
- `xsl:sort` 元素在将输入节点复制到输出文档中之前，可对输入节点重新进行排序。
- XSL 不能输出 CDATA 部分，也不能输出未转义的 `<` 符。
- 模式可从样式单中的不同位置，将不同模板应用于相同的元素。
- `xsl:variable` 元素定义命名的常数，以使代码清晰简练。
- 命名的模板有助于重新使用通用的模板代码。
- 在缺省的条件下，保留空白，除非用 `xsl:strip-space` 元素或 `xml:space` 特性说明为不保留。
- `xsl:if` 元素在当且仅当其 `test` 特性为真时，才产生输出。
- 当 `xsl:when` 子元素的 `test` 特性为真时，`xsl:choose` 元素输出其第一个 `xsl:when` 子元素的模板；或者，如果 `xsl:when` 元素都没有 `true` 的测试特性时，`xsl:choose` 元素输出其 `xsl:default` 元素的模板。
- `xsl:import` 和 `xsl:include` 元素合并不同样式单中的规则。

在下一章中，我们将继续 XSL 的另一半内容：格式化对象（formatting object）符号集。格式化对象是用来指定页面精确布局的极其强有力的手段。XSL 变换用于将 XML 文档转换成 XSL 格式化对象文档。

## 第 15 章 XSL 格式化对象

可扩展的样式语言（Extensible Style Language，XSL）的第二部分是格式化语言。这是 XML 应用程序，用来描述如何将内容显示给读者。一般地说，样式单使用 XSL 转换语言，将 XML 文档转换成使用 XSL 格式化对象符号集的新的 XML 文档。当许多人希望 Web 浏览器将来的某一天能够了解如何直接显示用 XSL 格式化对象来标记的数据时，目前就需要有其他措施，使输出文档进一步转换成其他的某个格式，如 PDE。

本章的主要内容如下：

- 理解 XSL 格式化语言
- 格式化对象及其属性
- 对页面进行格式化和设置样式

- 在文本中插入规则
- 在显示的文档中嵌入图形
- 与 URI 目标的链接
- 在文本中插入列表
- 替换字符
- 使用序列号
- 脚注
- 浮动
- 理解如何使用 XSL 格式化属性

## 15.1 XSL 格式化语言概述

XSL 格式化对象提供了比 HTML+CSS（甚至 CSS2）更为高级的可视化布局模型。XSL 格式化对象所支持但 HTML+CSS 不支持的格式化包括非西方布局、脚注、页边距注解、交叉引用中的页号等等。特别是，虽然 CSS 主要用于 Web，但 XSL 格式化对象的用途更为广泛。例如，能够编写使用格式化对象来编排整个打印稿的 XSL 样式单。不同的样式单能够将同一个 XML 文档转换到 Web 站点中。

### 有关格式化语言的警告语

XSL 仍处于开发中。过去 XSL 语言已经发生了本质上的变化，并且将来仍将发生变化。本章是根据 1999 年 4 月 21 日 XSL 规范草案（第四稿）编写的。当读者阅读本书时，XSL 的这一草案很可能已经被取代，而且 XSL 原来的句法已经改变。即便如此，本规范的格式化对象部分甚至也没有转换语言规范那样完善。如果确实遇到不能完全正常运行的情况，应将本书中提供的实例与最新的规则加以比较。

糟糕的是，仍然没有任何软件能实现 1999 年 4 月 21 日的 XSL 规范草稿的所有内容，甚至只对格式化对象这部分也没有任何软件能够实现。实际上，到目前为止，只有 James Tauber 的 FOP，才能部分地执行 XSL 格式化对象，它使用 XSL 格式化对象来将 XML 文档转换成 PDF。还没有任何 Web 浏览器可以显示用 XSL 格式化对象编写的文档。

当然，随着此项标准向最终版本改进时，当开发商实现 XSL 格式化对象时，这种情况最终是可以得到修正的。在那之前，我们不得不面对这样的选择：要么忍痛使用目前不完善的、未完成的 XSL，并且试图避开遇到的所有程序错误和疏忽，要么使用更确定的技术（如 CSS），直到 XSL 更加可靠为止。

## 15.2 格式对象及其属性

XSL 格式化对象元素正好有 51 个。在这 51 个元素当中，大多表示各种类型的矩形区域。其他的大部分都是矩形区域和空间的容器。下面以字母顺序编排，列出这些格式化对象：

- `bidi-override`
- `block`
- `character`
- `display-graphic`
- `display-included-container`
- `display-rule`
- `display-sequence`
- `first-line-marker`
- `float`
- `flow`
- `footnote`
- `footnote-citation`
- `inline graphic`
- `inline-included-container`
- `inline-rule`
- `inline-sequence`
- `layout-master-set`
- `list-block`
- `list-item`
- `list-item-body`
- `list-item-label`
- `multi-case`
- `multi-properties`
- `multi-property-set`
- `multi-switch`
- `multi-toggle`
- `page-number`
- `page-number-citation`
- `page-sequence`
- `region-after`
- `region before`
- `region-body`
- `region-end`
- `region-start`
- `root`
- `sequence-specification`
- `sequence-specifier-alternating`
- `sequence specifier repeating`
- `sequence-specifier-single`
- `simple-link`
- `simple-page-master`
- `static-content`
- `table`

- table-and-caption
- table-body
- table-caption
- table-cell
- table-column
- table-footer
- table-header
- table-row

XSL 格式化模型是基于称之为区域 (area) 的矩形框，该区域包含有文本、空格或其他格式化对象。尽管 CSS 页边距被 XSL 的缩进所代替，但正如 CSS 框一样，每个区域在其各侧都有边框和贴边。XSL 格式化程序读取格式化对象来确定将哪个区域放在页面的什么位置。许多格式化对象都会产生单一的区域（至少对大多数情况即是如此），但由于页面分隔符、单词折行、断字以及将可能存在的不确定量的文本填充到有确定区域中的其他方面的原因，一些格式化对象偶尔也确实产生多个区域。



含有间隔的框与含有空白字符的框是不一样的。含有空间的框是指页面或屏幕上的实际空的区域，例如，页面的左和右边上的页边距。这与页面上单词间的空格字符是不同的。

格式化对象主要在它们所包含的内容上有差别。例如，list-item-label 格式化对象就是一个包含项目符号、数字或放在列表项之前的其他指示符的框。list-item-body 格式化对象就是一个包含列表项的文本（无标签）的框。而 list-item 格式化对象就是一个包含 list-item-label 和 list-item 两个格式化对象的框。

格式化对象可进一步分成四类不同的矩形区域：

1. 区域容器
2. 块区域
3. 行区域
4. 内联区域

这四种类型的区域就形式了粗略的层次关系。区域容器包含其他更小的区域容器以及块区域。块区域又包含其他块区域、行区域和内容。行区域包含内联区域。内联区域包含其他内联区域和内容。所以，更具体地分为：

- 区域容器在 XSL 中是最高级别的容器。在包含它的区域内，可以精确的坐标加以定位。它既可以包含其他更小的区域容器，也可包含一系列的块区域和显示空间。可以将书的一页看作为区域容器，而这个区域容器包含五个其他区域容器：页眉、页的主体内容、页脚以及左和右页边距（在本例中，页边距区域无内容）。产生区域容器的格式化对象包括 region-body、region-before、region-after、region-start 和 region-end。
- 块区域代表块级元素，如段落或列表项。尽管块区域可能包含其他块区域，但在每个块区域的开始之前和结束之后都总是有一个换行符。块区域不能用坐标来精确定位，而是顺序地置于包含它的区域内。当在某个块区域之前或内部加入和删除其他块区域时，此块区域的位置发生移动，以便腾出空间。块区域可能含有行区域、显示空格以及连续地排列在用来包含的块区域中的其他块区域。块区域还可能包含一个图形影像。产生块区域的格式化对象包括 block、display-graphic、display-link、display-rule 和 list-block。
- 行区域表示块部分的一行文本。例如，列表项中的每个分开的行都是行区域。行区域可以包含内联区域和内联空间。对应的行区域没有格式化对象。取而代之的是，格式化引擎可计算行区域，例如确定在块区域内部如何折行。
- 内联区域是一行中的成分，如单字符、脚注引用或数学方程。内联区域可以包含其他内联区域和内联空间。产生内联的格式化对象包括 character、inline-graphic、inline-link、inline-rule、inline-sequence 和 page-number。

### 15.2.1 fo 命名域

在 XSL 样式单中，用于 XSL 格式化对象的 XML 元素放 [http://www.w3.org/XSL/Format /1.0](http://www.w3.org/XSL/Format/1.0) 命名域中，如下的声明所示：

```
<xsl:stylesheet
```

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

```
  xmlns:fo="http://www.w3.org/XSL/Format/1.0"
```

```
  result-ns="fo">
```

选择 fo 作为前缀的概率大约为 99%。因此，几乎总可以看到下列元素以 fo 作前缀：

- fo:bidirectional-override
- fo:block
- fo:character
- fo:display-graphic
- fo:display-included-container
- fo:display-rule
- fo:display-sequence
- fo:first-line-marker
- fo:float
- fo:flow
- fo:footnote
- fo:footnote-citation
- fo:inline-graphic
- fo:inline-included-container
- fo:inline-rule
- fo:inline-sequence
- fo:layout-master-set
- fo:list-block
- fo:list-item
- fo:list-item-body
- fo:list-item-label
- fo:multi-case
- fo:multi-properties
- fo:multi-property-set
- fo:multi-switch
- fo:multi-toggle
- fo:page-number
- fo:page-number-citation
- fo:page-sequence
- fo:region-after
- fo:region-before
- fo:region-body
- fo:region-end
- fo:region-start
- fo:root
- fo:sequence-specification
- fo:sequence-specifier-alternating

- fo:sequence-specifier-repeating
- fo:sequence-specifier-single
- fo:simple-link
- fo:simple-page-master
- fo:static-content
- fo:table
- fo:table-and-caption
- fo:table-body
- fo:table-caption

本章，我将使用 fo 作前缀，不再进一步说明。



命名域在第 18 章“命名域”中讨论。在那之前，私獾闹皇获 SL 格式化对象元素的名称都是以 fo: 开头。

## 15.2.2 格式化属性

总的来说，XSL 文档中的各种格式化对象都指定内容放在页面中的顺序。但是，格式化的所有详细内容（包括页的大小、元素大小、字体、颜色等等，但不局限于这些）都是由 XSL 属性指定的。这些格式化属性以各自格式化对象元素的特性来表示。

这些属性中的许多属性的细节都应该从 CSS 中了解了。下面所进行的工作是为了确保 CSS 和 XSL 使用相同的名称来表示同一个内容。例如，CSS 属性的 font-family 的含义与 XSL 的 font-family 属性是一回事；尽管在 CSS 和 XSL 中给属性赋值的句法不同，但值本身的句法是完全一样的。要说明 fo:block 元素格式化或与某种 Times 字体的近似，可以使用下面的 CSS 规则：

```
fo:block {font-family: New York, Times New Roman, Times, serif }
```

而 XSL 的等价语句可按下列方式，将 font-family 包括在 fo:block 开始标记中：

```
<fo:block
```

```
font-family=" New York,Times New Roman,Times,serif" >
```

尽管从表面上来看不同，但样式名（font-family）和样式值（New York, Times New Roman, Times, serif）却是完全一样的。CSS 的 font-family 属性指定为一组字体名，各字体名以逗号分开，而顺序是从第一选择到最后选择。XSL 的 font-family 属性指定为一组字体名，各字体名以逗号分开，选择顺序是从第一个选择到最后一个选择。CSS 和 XSL 都将关键字 serif 理解为任意的衬线字体。



由于本章是基于 XSL 草案规范的第四稿，所以无法彻底完成 CSS 和 XSL 等效属性的完全同步。在下一个草案中将解决此问题。

当然，XSL 格式化对象支持许多 CSS 中没有等效属性的属性，例如，font-size-adjust、ligature、character 和 hyphenation-keep。所以，需要学习这些属性，以便最大限度地发挥 XSL 的优势。标准的 XSL 属性有：

- auto-restore
- azimuth
- background
- background-attachment
- background-color
- background-image



- background-position
- background-repeat
- border
- border-after-color
- border-after-style
- border-after-width
- border-before-color
- border-before-style
- border-before-width
- border-bottom
- border-bottom-color
- border-bottom-style
- border-bottom-width
- border-collapse
- border-color
- border-end-color
- border-end-style
- border-end-width
- border-left
- border-left-color
- border-left-style
- border-left-width
- border-right
- border-right-color
- border-right-style
- border-right-width
- border-spacing
- border-start-color
- border-start-style
- border-start-width
- border-style
- border-top
- border-top-color
- border-top-style
- border-top-width
- border-width
- bottom
- break-after
- break-before
- caption-side
- cell-height
- character
- clear
- clip
- color
- column-count
- column-gap
- column-number
- column-width

- country
- cue
- cue-after
- cue-before
- digit-group-sep
- elevation
- empty-cells
- end-indent
- ends-row
- extent
- external-destination
- float
- flow-name
- font
- font-family
- font-height-override-after
- font-height-override-before
- font-size
- font-size-adjust
- font-stretch
- font-style
- font-variant
- font-weight
- format
- height
- href
- hyphenate
- hyphenation-char
- hyphenation-keep
- hyphenation-ladder-count
- hyphenation-push-char-count
- hyphenation-remain-char-count
- id
- indicate-destination
- inhibit-line-breaks
- initial
- initial-page-number
- internal-destination
- keep-with-next
- keep-with-previous
- language
- last-line-end-indent
- left
- length
- letter-spacing
- letter-value
- line-height
- line-height-shift-adjustment
- line-stacking-strategy

- margin
- margin-bottom
- margin-left
- margin-right
- margin-top
- max-height
- max-width
- may-break-after-row
- may-break-before-row
- min-height
- min-width
- name
- n-columns-repeated
- n-columns-spanned
- n-digits-per-group
- n-rows-spanned
- orphans
- overflow
- padding
- padding-after
- padding-before
- padding-bottom
- padding-end
- padding-left
- padding-right
- padding-start
- padding-top
- page-break-inside
- page-height
- page-master-blank-even
- page-master-even
- page-master-first
- page-master-last-even
- page-master-last-odd
- page-master-name
- page-master-odd
- page-master-repeating
- page-width
- pause
- pause-after
- pause-before
- pitch
- pitch-range
- play-during
- position
- precedence
- provisional-distance-between-starts
- provisional-label-separation
- reference-orientation

- ref-id
- richness
- right
- row-height
- rule-orientation
- rule-style
- rule-thickness
- scale
- score-spaces
- script
- sequence-src
- show-destination
- size
- space-above-destination-block
- space-above-destination-start
- space-after
- space-before
- space-between-list-rows
- space-end
- space-start
- span
- speak
- speak-header
- speak-numeral
- speak-punctuation
- speech-rate
- start-indent
- starts-row
- state
- stress
- switch-to
- table-height
- table-layout
- table-omit-middle-footer
- table-omit-middle-header
- table-width
- text-align
- text-align-last
- text-decoration
- text-indent
- text-shadow
- text-transform
- title
- top
- vertical-align
- visibility
- voice-family
- volume
- white-space-treatment

- widows
- width
- word-spacing
- wrap-option
- writing-mode
- z-index

### 15.2.3 转换成格式化对象

XSL 格式化对象是用于在页面上排列元素的一个完整的 XML 符号集。使用 XSL 格式化对象的文档只是使用此符号集的结构整洁的 XML 文档。这意味着它有 XML 声明、根元素、子元素等等。它必须遵从任何 XML 文档的所有结构整洁的规则，否则格式化程序就不接受它。出于习惯，含有 XSL 格式化对象的文件要有 .fob 这三个字符作后缀。但是，由于它还是一个结构整洁的 XML 文件，所以很可能以 .xml 作后缀。

清单 15-1 为一用 XSL 格式化对象来标记的简单文档。文档的根元素为 fo:root。此元素含有一个 fo:layout-master-set 和一个 fo:page-sequence。fo:layout-master-set 元素包含 fo:simple-page-master 子元素。每个 fo:simple-page-master 描述了用来放置内容的一类页面。其中只有一页非常简单的页面，但更复杂的文档可以有不同的主控页，用于第一页、右页、左页、正文页、封面内容、封底内容等等；每个文档又可能有一系列不同的页边距、页号以及其他特征。

使用 fo:page-sequence 可将内容放在主控页的许多副本上。fo:page-sequence 包含 fo:sequence-specification，指定不同的主控页应使用的顺序。其次，它还含有 fo:flow 子元素，此子元素保留以指定的序列放置在主控页上的实际内容。这里的内容是以两个 fo:block 子元素给出，而每个子元素的 font-size 属性值为 20 磅，font-family 属性值为 serif。

清单 15-1：使用 XSL 格式化对象符号集的简单文档

```
<fo:root xmlns:fo="http://www.w3.org/XSL/Format/1.0">

<fo:layout-master-set>

<fo:simple-page-master page-master-name="only">

<fo:region-body/>

</fo:simple-page-master>

</fo:layout master-set>

<fo:page-sequence>

<fo:sequence-specification>

<fo:sequence-specifier-single page-master-name="only"/>

</fo:sequence-specification>

<fo:flow>

<fo:block font-size="20pt" font-family="serif">
```

Hydrogen

</fo:block>

<fo:block font-size="20pt" font-family="serif">

Helium

</fo:block>

</fo:flow>

</fo:page sequence>

</fo: root>

尽管也许会像清单 15-1 中那样使用手工来编写文档，但这会失去 XML 所获得的内容格式独立性的所有优点。通常，应该编写 XSL 样式单，以便使用 XSL 转换符号集将源文档转换成格式化对象的符号集。使用清单 15-2 的 XSL 样式单，可将前一章中的清单 14-1 转换为清单 15-1。

清单 15-2：从源符号集到 XSL 格式化对象的变换

<?xml version="1.0"?>

<xsl:stylesheet

xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"

xmlns:fo="http://www.w3.org/XSL/Format/1.0"

result-ns="fo" indent-result="yes">

<xsl:template match="/">

<fo:root xmlns:fo="http://www.w3.org/XSL/Format/1.0">

<fo:layout-master-set>

<fo:simple-page-master page-master-name="only">

<fo:region-body/>

</fo:simple-page-master>

</fo:layout-master-set>

<fo:page-sequence>

<fo:sequence-specification>

<fo:sequence-specifier-single

page-master name="only"/>

```

</fo:sequence-specification>

<fo:flow>

<xsl:apply-templates select="//ATOM"/>

</fo: flow>

</fo:page-sequence>

</fo :root>

</xsl:template>

<xsl:template match="ATOM">

<fo:block font-size="20pt" font-family="serif">

<xsl:value-of select="NAME"/>

</fo:block>

</xsl:template>

</xsl:stylesheet>

```

#### 15.2.4 使用 FOP

在撰写本书时，没有任何浏览器能够直接显示转换成 XSL 格式化对象的 XML。只有一个软件可以使用以 XSL 格式化对象标记的文件，此软件即为 James Tauber 的 FOP。FOP 为免费的 Java 程序，它将 FO（格式化对象，formatting object）文档转换成 Adobe Acrobat PDF 文件。可从 <http://www.jtauber.com/fop/> 站点下载最新版的 FOP。

在撰写本书时，现有的 FOP 版本为 0.6.0，它不完全支持格式化对象的子集和 XSL 第四草案中的属性。FOP 是一 Java 程序，它可运行于适当兼容 Java 1.1 虚拟机的任何平台。要安装此程序，只需将 fop.jar 压缩文件放在 CLASSPATH 路径指明的目录中。com.jtauber.fop.FOP 类包含用于本程序的 main() 方法。在命令行中，可使用指定输入和输出文件的参数来运行本程序。例如：

```
C:\XML\BIBLE\15>java com.jtauber.fop.FOP 15-1.fob 15-1.pdf
```

```
James Tauber's FOP 0.6.0
```

```
auto page-height: using 11in
```

```
auto page-width: using 8in
```

```
successfully read and parsed 15-1.fob
```

```
laying out page 1... done page 1.
```

```
successfully wrote 15-1.pdf
```

其中 15-1.fob 是输入的 XML 文件，它使用格式化对象符号集。15-1.pdf 是输出的 PDF 文件，它能够在 Adobe Acrobat 或其他读取 PDF 文件的程序中显示和打印。

尽管 PDF 文件本身是 ASCII 文本，但本书不是有关 PostScript 的书籍，所以虽然精确地显示了上面的命令，读者却什么也得不到。如果好奇，可在任何文本编辑器程序中打开 PDF 文件。图 15-1 展示的转换文件，是使用 Acrobat 插件程序（plug-in），在 Netscape Navigator 中显示的。

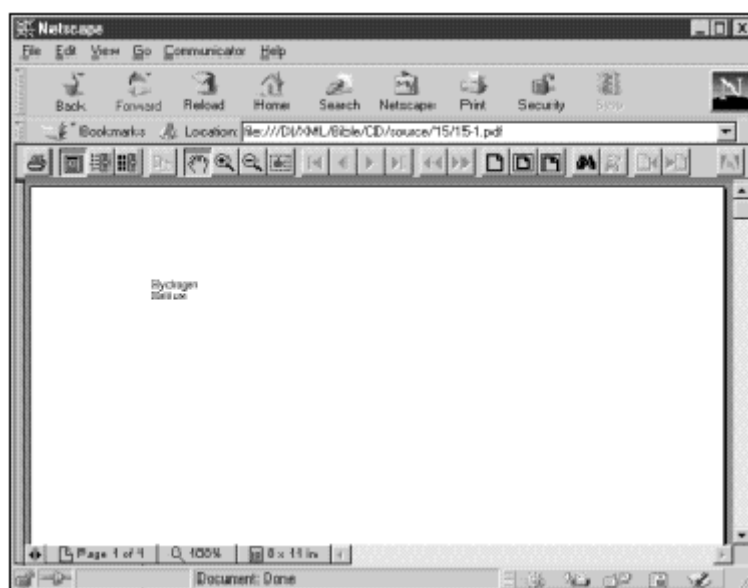


图 15-1 在 Netscape Navigator 中显示的 PDF 文件

对于使用 XSL 格式化对象来设置样式的 XML 文档来说，PDF 文件不是唯一的或是主要的最终目标格式。当然，人们希望在不远的将来 Web 浏览器能直接支持 XSL 格式化对象。就目前而言，PDF 文件是唯一可用的格式，这也正是我要在本章中说明的内容。最终，应该有更多的软件能够阅读和显示这些文件。



# 15.3 页面布局

格式化对象的根元素是 fo:root。此元素包含一个 fo:layout-master-set 元素和零或多个 fo:page-sequence 元素。fo:root 元素通常有 xmlns:fo 特性，其特性值为 http://www.w3.org/XSL/Format/1.0，并且可能（尽管通常情况下没有）有一个 id 特性。fo:root 元素的存在只为了声明命名域和文档根元素，它对页面布局或格式化没有直接的影响。

## 15.3.1 主控页面


fo:layout-master-set 元素为一容器，用于文档使用的所有不同的主控页面。简单的页面控制与 Quark XPress 主控页面或 PowerPoint 幻灯母板的用途类似。每个都定义页面（包括此页的页边距、页眉大小、页脚、文本区域等等）的通用布局。在显示文档中的每个实际页面都基于主控页，以及从此主控页面中继承某些属性，如页边距、页编号和布局。

### 15.3.1.1 简单的页面控制

每个主控页面都是由 fo:simple-page-master 元素表示的。fo:layout-master-set 可以包含一个或多个主控页面。一个 fo:simple-page-master 元素定义页的布局，包括页前区、主体区、后区、结束区以及开始区的大小。图 15-2 显示这些部分的典型布局。正文是中间留下来的所有内容。



图 15-2 一页简单的英语文本各部分的布局

 在正常的英语文本中，结束区处于页的右侧，开始区处于页的左侧。而在希伯来语或阿拉伯语的文本中，则反过来，因为这些语言是从右往左阅读。在几乎所有的现代语言中，前区是页眉，后区则是页脚，但在以从底部往顶部书写的语言中，这种情况则相反。

设计者利用适当的区域子元素可设置正文（中间部分）区、页眉、页脚、结束区和开始区的大小以及它们之间的距离。下面就是这些区域子元素：

- fo:region-before
- fo:region-after
- fo:region-body
- fo:region-start
- fo:region-end

这五个简单页面控制中的每一个区域都可以用 fo:flow 或 fo:static-content 元素来填充。

simple-page-master 元素通常有三个主要特性：

1. page-master-name: 这一页面控制的名称，页序列使用此名来选择依赖于特定页的主控页
2. page-height: 页的高度

### 3. page-width: 页的宽度

page-height 和 page-width 可归入一个缩略属性 size 中。如果不提供这两个特性,那么格式化部分根据所使用的媒体(例如 11" 8.5") 来选择合理的缺省值。

例如,此处的 fo:layout-master-set, 含有两个 fo:simple-page-master 元素: 一个用于偶数(左)页, 一个用于奇数(右)页。它们两个都指定 11 英寸长, 8.5 英寸宽的页面大小。它们的顶和底页边距为 0.5 英寸。按照通常页面相对的情况, 每个元素的的内侧页边距为 0.5 英寸, 外侧页边距为 1 英寸,

```
<fo:layout-master-set>

<fo:simple-page-master page-master-name="even"

height="8.5in" width="11in"

margin-top="0.5in" margin-bottom="0.5in"

margin-left="1.0in" margin-right="0.5in">

<fo:region-body/>

</fo:simple-page-master>

<fo:simple-page-master page-master-name="odd"

height="8.5in" width="11in"

margin-top="0.5in" margin-bottom="0.5in"

margin-left="0.5in" margin-right="1.0in">

<fo:region-body/>

</fo:simple-page-master>

</fo:layout-master-set>
```

通常用于页面控制的其他特性包括:

- 影响页面边距的特性: margin-bottom、margin-left、margin-right、margin-top、margin
- 影响页面书写方向的特性: writing-mode、reference-orientation

#### 15.3.1.2 区域属性

五个区域(before、after、body、start、end) 共享相同的基本属性。这些属性有:

- 确定超出区域边界的内容如何处理特性: clip、over-flow
- 确定内容在栏中如何折行的特性: column-count (区域中的栏号) 以及 column-gap (栏之间的距离)
- 影响区域背景的特性: background、background-attachment、background-color、background-image、background-repeat、background-position
- 影响区域边界的特性: border-before-color、border-before-style、border-before-width、border-after-color、

border-after-style、border-after-width、border-start-color、border-start-style、border-start-width、border-end-color、border-end-style、border-end-width、border-top-color、border-top-style、border-top-width、border-bottom-color、border-bottom-style、border-bottom-width、border-left-color、border-left-style、border-left-width、border-right-color、border-right-style、border-right-width、border、border-top、border-bottom、border-left、border-right、border-color、border-style、border-width

- 影响区域贴边的特性：padding-bottom、padding-left、padding-right、padding-top、padding-bottom、padding-start、padding-end、padding-before、padding-after、padding
- 影响区域页边距的特性：margin-bottom、margin-left、margin-right、margin-top、margin、space-before、space-after、start-indent、end-indent
- 影响区域中的书写方向的特性：writing-mode、reference-orientation

大多数属性与 CSS 属性同名，从 CSS 属性中就已很熟悉了。如果不明确地设置这些属性值，则选择合理的缺省值。对它们进行调整，就可以对页面的整个布局施加影响。

此外，四个外面区域（前、后、开始以及结束但不是正文）都有一个 extent 属性，它用来确定区域的大小。在去掉这四个区域之后，中间剩下来部分，就是主体正文的大小。

例如，下面的 fo:layout-master-set 使所有的外面区域都为 1 英寸。每个区域都有两个像素的黑色边界。而且，此页在所有的侧面上都有 0.5 英寸的页边距。

```
<fo:layout-master-set>

<fo:simple-page-master page-master-name="only"

height="8.5in" width="11in"

margin-top="0.5in" margin-bottom="0.5in"

margin-left="1.0in" margin-right="0.5in">

<fo:region-start extent="1.0in"

border-color="black" border-width="2px"/>

<fo:region-before extent="1.0in"

border-color="black" border-width="2px"/>

<fo:region-body

border-color="black" border-width="2px"/>

<fo:region-end extent="1.0in"

border-color="black" border-width="2px"/>

<fo:region-after extent="1.0in"

border-color="black" border-width="2px"/>

</fo:simple-page-master>
```

</fo:layout-master-set>

基于页面控制的正文页为 5.5 英寸宽, 8 英寸高。此值可以通过此页的大小减去此页上剩下来的其他部分的大小计算获得。

### 15.3.2 页序列

除 fo:layout-master-set 外, 每个格式化对象文档通常还包含一个或多个 fo:page-sequence 元素。每个页序列按下列顺序含有三项内容:

- 一个 fo:sequence-specification 元素, 以定义主控页使用顺序
- 零个或多个 fo:static-content 元素, 其中包含每页上要放置的文本
- 一个 fo:flow 元素, 其中包含要在各页上依次放置的数据

fo:flow 和 fo:static-content 之间的主要区别是, 文本流 (flow) 中的文本并不放在多个页面上, 但静态内容则是如此。例如, 读者现在在阅读的这些行是文本流内容, 只出现在此页上, 而此页顶上的部分和章节标题则是静态内容, 它在每页都是重复的。

fo:sequence-specification 为此序列提供一组主控页。序列中的每页都有相关的页面控制, 以定义页面的外观。清单 15-1 只使用一个主控页, 但通常有多个主控页; 例如, 用于一章的第一页的主控页, 用于所有的连续的左手页的主控页以及用于所有的连续的右手页的主控页。例如, 可能有一个简单的页面控制用于目录, 另一个用于正文文本, 第三个用于索引。在这种情况下, 目录、正文文本以及索引每个都对应有一个页面序列。

fo:flow 元素按顺序包含本页上要放置的元素。当各页都以文本流的元素填充时, 就会使用序列规格中的下一个控制布局方式为文本流中存在的元素创建新页。

fo:static-content 元素包含每页上要放置的信息。例如, 可将书的标题放在每页的页眉中。静态内容可以根据主控页来调节。例如, 部分的标题可能放在左手的页上, 章节的标题放在右手页上。还可以将 fo:static-content 元素用于像页码这样的项目, 页码必须从一页到另一页不断地重复进行计算。换句话说, 静态的实质并不是文本, 而是产生文本的运算。

#### 15.3.2.1 序列规格

使用铝腥 蜚釉 机械囊恒龊蚨喔解绿 o:sequence-specification 元素可列出特定的主控页被说明的顺序:

- fo:sequence-specifier-single
- fo:sequence-specifier-alternating
- fo:sequence-specifier-repeating

每个子元素都有特性, 此特性确定何时使用哪个主控页。最简单的是 fo:sequence-specifier-single, 其 page-master-name 特性标识准备说明的主控页。例如, 下面的 fo:sequence-specification 元素说明所有的内容都必须放在名为 letter 的主控页的一个实例上。

```
<fo:sequence-specification>
```

```
<fo:sequence-specifier-single page-master-name="letter"/>
```

```
</fo:sequence-specification>
```

如果有更多的内容可以填充在一页上, 那么, 超过的内容要么切去顶端, 要么上卷, 这要根据放置此内容各区域的 clip 和 overflow 特性值而定。但是, 创建页数不会多于一页。现在来考虑一下下面的序列规格:

```
<fo:sequence-specification>
```

```
<fo:sequence-specifier-single page-master-name="letter"/>
```

```
<fo:sequence-specifier-single page-master-name="letter"/>
```

```
</fo:sequence-specification>
```

此段代码为每个基于 letter 页面控制的所有页都提供了序列规格。如果第一页填满，就创建第二页。如果那页填满，那么内容就被切去或上卷。

同样可将这一技术用来施加不同的主控页。例如，下面的序列规格将第一页以名为 letter1 的主控页为基础，第二页以名为 letter2 的主控页为基础。

```
<fo:sequence-specification>
```

```
<fo:sequence-specifier-single page-master-name="letter1"/>
```

```
<fo:sequence-specifier-single page-master-name="letter2"/>
```

```
</fo:sequence-specification>
```

当然，大多数时候不会预先精确地知道有多少页。fo:sequence-specifier-alternating 和 fo:sequence-specifier-repeating 元素可用来指定按实际需要的页数来保存内容。fo:sequence-specifier-repeating 元素为第一页指定一个主控页，为所有的后续页指定第二个主控页。fo:sequence-specifier-alternating 元素为第一页、有内容的偶数页、有内容的奇数页、空白偶数页、最后的偶数页以及最后的奇数页指定多达六个不同的主控页。

例如，下面的序列说明符表示第一页的输出应使用名为 letter\_first 的主控页，但所有的后续页都使用名为 letter 的主控页：

```
<fo:sequence-specification>
```

```
<fo:sequence-specifier-repeating
```

```
page-master-first="letter_first"
```

```
page-master-repeating="letter"
```

```
/>
```

```
</fo:sequence-specification>
```

如果整个内容都超出第一页，那么它就放在第二页上。如果超出第二页，那么就创建第三页。按照需要保存所有内容来建立页数。



在撰写本书时，仍未确定内容是否需要 page-master-first 和 page-master-repeating。但如果只有单一主控页，肯定要像下面这样将其作为值重新用于 page-master-first 和 page-master-repeating：

```
<fo:sequence-specification>
```

```
<fo:sequence-specifier-repeating
```

```
page-master-first="letter"
```

```
page-master repeating="letter"
```

```
/>
```

```
</fo:sequence-specification>
```

fo:sequence-specifier-alternating 元素多用于打印书籍的章节，习惯上书籍的第一和最后一页，以及奇偶页上的页边距、页眉和页脚都不同。这个元素的特性可用来为这些所有不同页指定主控页。例如：

```
<fo:sequence-specification>
```

```
<fo:sequence-specifier-repeating
```

```
page-master-first="chapter_first"
```

```
page-master-even="chapter_even"
```

```
page-master-blank-even="chapter_blank"
```

```
page-master-odd="chapter_odd"
```

```
page-master-last-even="chapter_last_even"
```

```
page-master-last-odd="chapter_last_odd"
```

```
page-Master-repeating="letter"
```

```
/>
```

```
</fo:sequence-specification>
```



如果上面的特性似乎不对称的话；例如没有 page-master-blank-odd 特性；这是由于传统的出版物不对称。如果仔细看看本书的所有页，并看看你所拥有的任何一本书，将会注意到编成奇数号的页总是在右边，编成偶数号的页总是在左边，并且章号总是在右手页上。章可以结束于右手（奇数）页，也可以结束于左手（偶数）页，但如果章确实结束于奇数页，那么就插入一空白的偶数页，以便下一章起始于奇数页。

### 15.3.2.2 文本流

fo:flow 对象保存放置在由序列规格指定的主控页实例上的真实内容。此内容是由一系列 fo:block、fo:display-graphic、fo:display-link、fo:display-rule 以及其他块级元素组成的。在本节，我们将集中于基本的 fo:block 元素，它大体上与 HTML 的 DIV 元素等价。在本章的后面，我们将看到文本流包含的更多的块级元素。

例如，下面的这个基本文本流，包含几个原子的名称，它们分别放在各自的块中：

```
<fo:flow name="xsl-body">
```

```
<fo:block>Actinium</fo:block>
```

```
<fo:block>Aluminum</fo:block>
```

```
<fo:block>Americium</fo:block>
```

</fo:flow>

fo:flow 的 name 特性（此处有 xsl-body 值）指定此文本流的内容将放在该页中五个区域的哪个区域上。允许值为：

- xsl-body
- xsl-after
- xsl-before
- xsl-start
- xsl-end

例如，页眉的 flow（从左到右，从上到下的英语文本）的 flow-name 具有值为 xsl-before。下面是用于页脚的文本流：

```
<fo:flow id="q2" flow-name="xsl-after">
```

```
<fo:block>
```

The XML Bible

Chapter 15: XSL Formatting Objects

```
</fo:block>
```

```
</fo:flow>
```

### 15.3.2.3 静态内容

尽管 fo:flow 元素的每条内容都出现在一页上，但 fo:static-content 元素的每条内容则出现在每页上；例如，页眉或页脚。不一定要使用 fo:static-content 元素，但如果使用，必须放在页序列中所有的 fo:flow 元素之前。

fo:static-content 元素的特性和内容与 fo:flow 的相同。但是，由于 fo:static-content 不能将自己的内容放在多个页面上，如果必须如此，那么通常其内容要比 fo:flow 少。例如，下面是用于页眉的 fo:static-content：

```
<fo:static-content id="sc2" flow-name="xsl-before">
```

```
<fo:block>
```

The XML Bible

Chapter 15: XSL Formatting Objects

```
</fo:block>
```

```
</fo:static-content>
```

### 15.3.2.4 页的编号

除了任何格式化对象所具有的常用的 id 特性外，fo:page-sequence 元素还有六个可选的特性，它们可为序列定义页编号。这六个特性为：

- initial-page-number
- format

- letter-value
- digit-group-sep
- n-digits-per-group
- sequence-src

initial-page-number 特性定义此序列中的第一页号码。此特性的最可能的值是 1，但如果前面的页存在于不同的文件中，那么此值可能比较大。剩下的五个特性的句法和含义，与作为 XSL 转换语言的 xsl:number 元素的特性使用时完全相同。



xsl:number 元素和 format、letter-value、digit-group-sep、n-digits-per-group、sequence-src 特性已在第 14 章“XSL 变换”的“数字到字符串的转换”中讨论过。

fo:page-number 格式化对象是一空的内联元素，用于插入当前页的号码。格式化程序负责确定使用什么样的号码。此元素只有唯一的一个特性 id。此外，可将 fo:page-number 包装在 fo:inline-sequence、fo:block 或类似的元素中，以便将字体属性和类似的属性应用于此元素。例如，下面的页脚使用 fo:static-content 和 fo:page-number 来将页码放在每页的底部：

```
<fo:static-content id="sc2" flow-name="xsl-after">
```

```
<fo:block>
```

```
<fo:page-number/>
```

```
</fo:block>
```

```
</fo:static-content>
```

下面的页序列指定该页码使用小罗马数字，并从 10 开始计数。

```
<fo:page-sequence initial-page-number="10" format="i">
```

```
<!-- sequence specification -->
```

```
<fo:static-content flow-name="xsl-after">
```

```
<fo:block text-align-last="centered" font-size="10pt">
```

```
<fo:page-number/>
```

```
</fo:block>
```

```
</fo:static-content>
```

```
<!-- flows -->
```

```
</fo:page-sequence>
```



## 15.4 内容

XSL 格式化对象文档的内容（与标记相反）几乎都是文本。除此之外，还可链接于外部影像，这种方式类似于 HTML 的 IMG 元素。这种内容保存存在于下列几类元素中：

- 块级格式化对象
- 内联格式化对象
- 表格格式化对象
- 外联格式化对象

所有的这些元素都是 fo:flow 或 fo:static-content 元素的后代。它们从不直接放在控制页或页序列上。

### 15.4.1 块级格式化对象

块级格式化对象以矩形区域绘制，各矩形区域以换行符分开，可能在其前或后的内容中还有多余的空白。块可能包括其他块，在此情况下，被包括的块也是通过换行符（可能使用多余的空白）与用来包括的块分开。块级格式化对象包括：

- fo:block
- fo:display-graphic
- fo:display-rule
- fo:display-included-container
- fo:display-sequence
- fo:list
- fo:list-item

fo:block 元素是 CSS 中的 display:block 或 HTML 中的 DIV 元素的 XSL 等价元素。块可能包括在 fo:flow 元素、其他 fo:block 元素以及 fo:static-content 元素中。fo:block 元素可能包括其他 fo:block 元素、其他块级元素（如 fo:display-graphic 和 fo:display-rule）以及内联元素（如 fo:inline-sequence 和 fo:page-number）。还可能包括原始文本。例如：

```
<fo:block>
```

```
<fo:inline-sequence font-style="italic">
```

```
The XML Bible
```

```
</fo:inline-sequence>
```

```
Page <fo:page-number/>
```

```
<fo:inline-sequence>
```

```
Chapter 15: XSL Formatting Objects
```

```
</fo:inline-sequence>
```

```
</fo:block>
```

fo:block 元素通常都有用于区域属性和文本格式化属性的特性。文本格式化属性可被块的任何子元素所继承，除非被覆盖。允许的属性包括：

- 对齐属性：text-align 和 text-align-last

- 听觉属性: azimuth、cue、cue-after、cue-before、elevation、pause、pause-after、pause-before、pitch、pitch-range、play-during、richness、speak、speak-header、speak-numeral、speak-punctuation、speech-rate、stress、voice-family 和 volume
- 背景属性: background、background-attachment、background-color、background-image、background-position 和 background-repeat
- 边界属性: border-before-color、border-before-style、border-before-width、border-after-color、border-after-style、border-after-width、border-start-color、border-start-style、border-start-width、border-end-color、border-end-style、border-end-width、border-top-color、border-top-style、border-top-width、border-bottom-color、border-bottom-style、border-bottom-width、border-left-color、border-left-style、border-left-width、border-right-color、border-right-style、border-right-width、border、border-top、border-bottom、border-left、border-right、border-color、border-style 以及 border-width
- 分行属性: page-break-inside、widows、orphans 和 wrap-option
- 颜色属性: color
- 栏属性: span
- 字体属性: font-family、system-font、font-size、font-size-adjust、font-stretch、font-style、font-variant、font-weight 和 font
- 断字属性: country、hyphenate、hyphenation-char、hyphenation-push-char-count、hyphenation-remain-char-count、language、script、hyphenation-keep 和 hyphenation-ladder-count
- 缩排属性: text-indent 和 last-line-end-indent
- 分层属性: z-index
- 行高属性: line-height、line-height-shift-adjustment 和 line-stacking-strategy
- 页边距属性: margin-bottom、margin-left、margin-right、margin-top、margin、space-before、space-after、start-indent 和 end-indent
- 贴边属性: padding-top、padding-bottom、padding-left、padding-right、padding-before、padding-after、padding-start 和 padding-end
- 位置属性: position、top、bottom、right 和 left
- 文本方向属性: writing-mode
- 可视属性: visibility
- 空白属性: white-space-treatment

大多数属性从 CSS 中就熟悉了，其余的将在下面讨论。其他块级元素都有很相似的属性列表。

#### 15.4.2 内联格式化对象

内联格式化对象是以矩形区域来描述的，这个区域可以包括文本或其他内联区域。内联区域几乎通常都是从左到右以行排列。当一行填满时，在前一行的下面开始新行。但是，内联元素放置的准确顺序要依赖于书写方式。例如，当使用希伯来语或阿拉伯语时，它要了解左边第一位置的内联元素的意义，然后从右边填充。内联格式化对象包括：

- fo:bidirectional-override
- fo:character
- fo:first-line-marker
- fo:inline-graphic
- fo:inline-included-container
- fo:inline-rule
- fo:inline-sequence
- fo:list-item-body
- fo:list-item-label
- fo:page-number
- fo:page-number-citation

### 15.4.3 表格格式化对象

设计的表格格式化对象是 CSS2 表格属性的 XSL 等价物。但是，表格在 XSL 中的工作方式确实比 CSS 中的稍微自然些。对于大部分内容来说，各个表都是块级对象，而表中的内容则不是真正的内联级，也不是块级。但是，整个一张表可以通过将它包装在 `fo:inline-included-container` 中，从而转换成内联对象。

有下列九个 XSL 表格格式化对象：

- `fo:table-and-caption`
- `fo:table`
- `fo:table-caption`
- `fo:table-column`
- `fo:table-header`
- `fo:table-footer`
- `fo:table-body`
- `fo:table-row`
- `fo:table-cell`

表的根元素不是 `fo:table`，而是 `fo:table-and-caption`，它包括一个 `fo:table` 和一个 `fo:caption`。`fo:table` 包含 `fo:table-header`、`fo:table-body` 和 `fo:table-footer`。表的正文包括 `fo:table-row` 元素，而此元素可划分到 `fo:table-cell` 元素中。

### 15.4.4 外联格式化对象

有下列三个外联格式化对象：

- `fo:float`
- `fo:footnote`
- `fo:footnote-citation`

外联格式化对象从现存的内联或块对象那里“借用”空间。在页面上，它们不必出现在相同的元素之间，而在输出格式化对象的 XML 树形结构中，它们则出现在相同的元素之间

## 15.5 水平线

rule 是插入到文本中的水平线。XSL 有两类水平线。fo:display-rule 格式化对象为块级元素，它创建一条水平线（如由 HTML 的<HR>标记产生的水平线）。fo:inline-rule 格式化对象元素与 fo:display-rule 元素类似。但是，顾名思义，fo:inline-rule 是内联元素，而不是块级元素。因此，它出现在一行文本的中间，并且不代表分行符。例如，下面是一显示的水平线：

但是，这个\_是内联水平线。

fo:inline-rule 和 fo:display-rule 元素有六个描述这两种水平线的主要特性：

1. length: 线条长，如 12pc 或 5in
2. rule-orientation: escapement、horizontal、line-progression 或 vertical
3. rule-style: 在编写本书时，精确值还未确定
4. rule-thickness: 线条的粗细，如 1px 或 0.1cm
5. vertical-align: baseline、bottom、middle、sub、super、text-bottom、text-top、top 或行长或行高的百分比
6. color: 线条的颜色，如 pink 或 #FFCCCC

例如，下面的这个绿色块级水平线长 7.5 英寸，粗 2 磅：

```
<fo:display-rule length="7.5in"
line-thickness="2pt" color="#00FF00"/>
```

此外，fo:display-rule 块级元素的特性中，大多数都是经常使用的，如描述页边距和贴边的特性，fo:inline-rule 也有内联元素的常用特性，如 line-height。直接与文本有关的特性则是个例外（如字族），很明显，这些特性对水平线没有意义。

## 15.6 图形

XSL 提供在显示文档中嵌入图形的两种方式。fo:display-graphic 元素插入块级图形。fo:inline-graphic 元素插入内联图形。这两个元素与 HTML 的 IMG 标签是等价的。可以使用下面六个特性来描述图像：

1. href：影像文件的 URI
2. min-height：影像的最小垂直高度
3. min-width：影像的最小水平宽度
4. max-height：影像的最大垂直高度
5. max-width：影像的最大水平宽度
6. scale：使用值 max，将图形扩充到 max-height 和 max-width 的大小；使用 max-uniform 值，在垂直和水平方向以同等数量将图形扩充到 max-height 或 max-width（使用最先出现的）的大小；使用一个实数，由此值乘以高度和宽度；使用两个实数，第一个实数乘以宽度，第二个实数乘以高度。

例如，考查下面标准 HTML 的 IMG 元素：

```
<IMG SRC="logo.gif" WIDTH="100" HEIGHT="100"  
ALIGN="right" ALT="alt text" BORDER="0">
```

下面的 fo:display-graphic 元素与上面的等效：

```
<fo:display-graphic image="logo.gif"  
height="100px" width="100px" />
```

## 15.7 链接

对于只用于在线展示而言，XSL 提供 `fo:simple-link` 元素。假定有 Web 浏览器样式的用户界面，单击链接元素内容的任何地方，可跳转到链接目标。根据此元素所包含的内容，可以充当块级或内联链接。链接行为是由下列六个特性控制的：

- `external-destination`
- `internal-destination`
- `indicate-destination`
- `show-destination`
- `space-above-destination-block`
- `space-above-destination-start`

对远程目标文档的链接通过 `external-destination` 特性的值来指定 URI。当激活链接时，应加载此 URI 上的文档。在 GUI 环境下，这种链接极可能是通过单击链接内容而被激活的。例如：

```
<fo:block> Be sure to visit the

<fo:simple-link

external-destination="http://metalab.unc.edu/xml/">

Cafe con Leche Web site!

</fo:simple-link>

</fo:block>
```

使用 `internal-destination` 特性，还可以与同一文档中的其他节点链接。此特性值不是 URI，而是链接元素的 ID。不要为一个链接同时指定内部和外部目标。

其他四个特性影响链接的外观和行为。`indicate-destination` 特性为 Boolean 值（true 或 false，缺省值为 false），它指定是否加载链接项目、何时加载，应以某种方式将它与同一文档的非链接部分区别开来。例如，如果链接于有 100 个原子的表中的一个 ATOM 元素，那么正在进行关联的指定原子可能要以粗体形式显示，而其他原子则为正常字体类型。精确的内容依赖于系统。

`show-destination` 特性有两个可能值：`replace`（缺省）和 `new`。使用 `replace` 值，下面的链接将代替同一窗口中的现有文档。使用 `new` 值时，单击链接后，目标文档在新的窗口中打开。

当浏览器跟随 HTML 链接进入文档中间时，通常指定的被链接元素定位在窗口的最顶上。`space-above-destination-start` 和 `space-above-destination-block` 特性可用来指定浏览器将链接的元素定位在窗口的下方，而在链接项目的上方留出一定数量的空间（不是空格，通常它包含链接元素前面的内容）

此外，链接也有一个常用的属性，如颜色，可被链接内容所继承。这样就可以用来格式化链接的内容，使其与链接外的内容格式不同；例如，将所有的链接加上上下划线。但是，与 CSS 和 HTML 不同，XSL 格式化对象不提供区别被访问、未被访问以及活动链接的方法。

## 15.8 列表

fo:list-block 格式化对象描述块级列表元素（没有内联列表）。一个列表可能有、也可能没有项目符号、编号、缩进或其他格式。每个 fo:list-block 元素都包含一系列的 fo:list-item 元素或 fo:list-item-label fo:list-item-body 元素对（不能同时包括两者）。fo:list-item 必须包括 fo:list-item-label 和 fo:list-item-body。fo:list-item-label 包括项目符号、编号或用于列表项的其他标签。fo:list-item-body 包括列表项的实际内容。一句话，fo:list-block 包括 fo:list-item 元素。每个 fo:list-item 包括一个 fo:list-item-label 和 fo:list-item-body。但是，fo:list-item 元素可以省略。例如：

```
<fo:list-block>

<fo:list-item>

<fo:list-item-label>*</fo:list-item-label>

<fo:list-item-body>Actinium</fo:list-item-body>

</fo:list-item>

<fo:list-item>

<fo:list-item-label>*</fo:list-item-label>

<fo:list-item-body>Aluminum</fo:list-item-body>

</fo:list-item>

</fo:list-block>
```

或者，将 fo:list-item 删除：

```
<fo:list-block>

<fo:list-item-label>*</fo:list-item-label>

<fo:list-item-body>Actinium</fo:list-item-body>

<fo:list-item-label>*</fo:list-item-label>

<fo:list-item-body>Aluminum</fo:list-item-body>

</fo:list block>
```

fo:list-block 元素有三个专用特性：

1. provisional-label-separation: 列表项标签与列表项正文之间的距离，以最大；最小；最佳这三个值来表示，如 2cm; 0.5cm; 1cm。
2. provisional-distance-between-starts: 列表项标签的开始边与列表项正文之间的开始边的距离。
3. space-between-list-rows: 连续列表项之间的垂直距离，以最大；最小；最佳这三个值来表示，如 36pt; 4pt; 12pt。

`fo:list-item` 元素的标准块级属性，用于背景、位置、音频显示、边界、贴边、页边距、线条以及分页符。



# 15.9 表格

XSL 中的基本的表格元素为 fo:table-and-caption，这是个块级对象。但将它包装到 fo:inline-included-container 中就可转变为内联对象，或将它包装到 fo:float 中就可转变为外联对象。表格模型与 HTML 的表格模型十分相近。表 15-1 显示 HTML 4.0 表元素与 XSL 格式化对象之间的等价关系：

表 15-1 HTML 表格与 XSL 格式化对象的表格的对比

HTML 元素	XSL 格式化对象元素
TABLE	Fo:table-and-caption
无对等元素	fo:table
CAPTION	fo:table-caption
COL	fo:table-column
COLGROUP	无对等元素
THEAD	fo:table-reader
TBODY	fo:table-body
TFOOT	fo:table-footer
TD	fo:table-cell
TR	fo:table-row

fo:table-and-caption 包括一个可选的 fo:caption 元素和一个 fo:table 元素。标题可包含要放在此标题中的任何块级元素。在缺省的情况下，标题放在表之前，但可以通过设置 table-and-caption 元素的 caption-side 属性为下列八个值之一而进行调整：

- before
- after
- start
- end
- top
- bottom
- left
- right

例如，下面是一个将标题放在底部的表格：

```
<fo:table-and-caption caption-side="bottom">
```

```
<fo:table-caption>
```

```
<fo:block font-weight="bold"
```

```
font-family="Helvetica, Arial, sans"
```

```
font-size="12pt">
```

Table 15-1: HTML Tables vs. XSL Formatting Object Tables

```
</fo:block>
```

```
</fo:table-caption>
```

```
<fo:table>
```

```
<!-- table contents go here -->
```

```
</fo:table>
```

```
</fo:table-and-caption>
```

fo:table 元素包括一个可选的 fo:table-column、fo:table-header、一个可选的 fo:table-footer 和一个或多个 fo:table-body 元素。fo:table-body 分成 fo:table-row 元素。每个 fo:table-row 分成 fo:table-cell 元素。fo:table-header 和 fo:table-footer 既可分成 fo:table-cell 元素，也可分成 fo:table-row 元素。例如，下面的这个简单的表，与表 15-1 的第三行相对应：

```
<fo:table>
```

```
<fo:table-header>
```

```
<fo:table-cell>
```

```
<fo:block font-family="Helvetica, Arial, sans"
```

```
font-size="11pt" font-weight="bold">
```

HTML Element

```
</fo:block>
```

```
</fo:table-cell>
```

```
<fo:table-cell>
```

```
<fo:block font-family="Helvetica, Arial, sans"
```

```
font-size="11pt" font-weight="bold">
```

XSL FO Element

```
</fo:block>
```

```
</fo:table-cell>
```

```
</fo:table-header>
```

```
<fo:table-body>
```

<fo:table-row>

<fo:table-cell>

<fo:block font-family="Courier, monospace">

TABLE

</fo:block>

</fo:table-cell>

<fo:table-cell>

<fo:block font-family="Courier, monospace">

fo:table-and-caption

</fo:block>

</fo:table-cell>

</fo:table-row>

<fo:table-row>

<fo:table-cell>

<fo:block>no equivalent</fo:block>

</fo:table-cell>

<fo:table-cell>

<fo:block font-family="Courier, monospace">

fo:table

</fo:block>

</fo:table-cell>

</fo:table-row>

</fo:table-body>

</fo:table>

设置 n-columns-spanned 和 n-rows-spanned 特性为一整数，表示要跨越的行或列数，这时，表的单元格就可以跨越多行和多列。可选的 column-number 特性可以改变从哪一列开始合并单元格，缺省值是当前列。

使用通常的边界属性（将在以后讨论），就可以在表的各部分周围绘制边框。`empty-cells` 特的值可取 `show` 或 `hide`，如果要在无内容的单元格周围绘制边框，则为 `show`；否则为 `hide`。缺省值为 `show`。

大多数表的各部分不使用标准的宽度和高度属性，而是具有等价的属性。可以省略下面当中的任何一个或所有的特性，在此情况下，格式化程序只将每个部分调整成合适的大小：

- `table`: `table-width`, `table-height`
- `table-caption`: `caption-width`, 由格式化程序自动确定高度
- `table-row`: `row-height`, 由内容确定宽度
- `table-cell`: `cell-height`, `column-number`, `column-width`, `n-columns-spanned`, `n-rows-spanned`

`fo:table-row` 元素有可选的 `may-break-after-row` 和 `may-break-before-row` 特性，其值为 `yes` 或 `no`，此值确定在行前和行后是否允许有分页符。这两个特性的缺省值都为 `yes`。

当一长表扩展到多页时，有时候在每页上重复页眉和页脚。使用 `fo:table` 元素的 `table-omit-middle-header` 和 `table-omit-middle-footer` 特性，可指定这种行为。值为 `yes` 表示页眉或页脚一页一页地重复。值为 `no` 表示页眉或页脚不是一页一页地重复。缺省值为 `no`。

可选的 `fo:table-column` 元素是一空元素，它为一特定列中的所有单元格指定值。使用此元素的单元格由 `column-number` 特性来识别。`fo:table-column` 并不真正地包含任何单元格。将 `n-columns-spanned` 属性设置成大于 1 的整数时，`fo:table-column` 可将属性应用于多个连续的列中。在 `fo:table-column` 中进行设置的最常用的属性是 `column-width`（有符号的长度），但标准的边界、贴边以及背景属性（下面讨论）也可以设置。

## 15.10 字符

fo:character 格式化对象使用输出文档中的不同字符来代替输入文档中特定字符或字符串。例如，可以使用此对象在英语小数点与法语小数逗号之间相互转换。character 特性指定使用什么字符来替换。例如，下面的模板规则将 PASSWORD 元素中的字符替换为\*：

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

```
<fo:character character="*">
```

```
<xsl:value-of select="."/>
```

```
</fo:character>
```

```
</xsl:template>
```

但是，这种用法是很少见的。fo:character 元素的主要目的是为了便于格式化引擎可将各个字符和字形作为其自身的元素来看待。如果不是编写格式化引擎，那么就可以忽略此元素。

## 15.11 序列

序列对内联或块级框的布局没有任何特别的影响。它们只是放置格式化属性（如 `font-style` 或 `text-indent`）的元素。

`fo:display-sequence` 格式化对象元素是一容器，它将块级对象成组在一起。事实上，它只能保存块级元素，如 `fo:display-graphic` 和 `fo:block`。不能包含内联元素和原文本。

`fo:inline-sequence` 格式化对象元素是将内联对象成组在一起的容器。它不能包含块级元素。例如，可以像下面这样，使用 `inline-sequence` 元素将样式加到页脚的各部分中：

```
<fo:flow id="q2" flow-name="xsl-after">

<fo:block font-style="bold" font-size="10pt"

font-family="Arial, Helvetica, sans">

<fo:inline-sequence font-style="italic"

text-align="start">

The XML Bible

</fo:inline-sequence>

<fo:inline-sequence text-align="centered">

Page <fo:page-number/>

</fo:inline-sequence>

<fo:inline-sequence text-align="right">

Chapter 15: XSL Formatting Objects

</fo:inline-sequence>

</fo:block>

</fo:flow>
```

## 15.12 脚注

fo:footnote 元素表示脚注。作者将 fo:footnote 元素放在文本流中，脚注引用（<sup>1</sup>或\*）出现的地方。fo:footnote 元素由 fo:footnote-reference 和包含脚注文本的块级元素组成。但是，只有脚注引用才是内联式插入。格式化程序将注释文本放在此页的后区（通常为页脚）中。

例如，下面的脚注使用星号作为脚注标记，并引用 *JavaBeans*, Elliotte Rusty Harold IDG Books, Foster City, 1998), p. 147。使用标准的 XSL 属性（如 font-size 和 vertical-align）按照习惯方式来格式化注释标记和文本。

```
<fo:footnote>

<fo:footnote-reference

font-size="smaller" vertical-align="super">

*

</fo:footnote reference>

<fo:block font-size="smaller">

<fo:inline-sequence

font-size="smaller" vertical-align="super">

*

</fo:inline-sequence>

<fo:inline-sequence

font-style="italic">JavaBeans</fo:inline-sequence>,

Elliotte Rusty Harold

(IDG Books, Foster City, 1998), p. 147

</fo:block>

</fo:footnote>
```



格式化对象符号集不提供自动编号和引用脚注的任何方法，但在变换样式单中巧妙地使用 xsl:number 即可做到这一点。XSL 变换同样也可以很容易地实现尾注（end note）。

## 15.13 浮动

fo:float 产生一个浮动框，定位于它所出现的区域的顶端。fo:float 最常用于图形、图表、表格，或需要出现在该页某个地方但对出现的位置无特别精确要求的其他外联内容。例如，下面的代码用于在一段落的中间嵌入带有标题的浮动图形：

```
<fo:block>
```

Although PDF files are themselves ASCII text,  
this isn't a book about PostScript, so there's  
nothing to be gained by showing you the exact  
output of the above command. If you're curious,  
open the PDF file in any text editor.

Instead, Figure 15-1

```
<fo:float>  
  
<fo:display-graphic  
image="3236-7fg1501.jpg"  
height="485px" width="623px" />  
  
<fo:block font-family="Helvetica, sans" >  
  
<fo:inline sequence font-weight="bold" >
```

Figure 15-1 :

```
</fo:inline-sequence>
```

The PDF file displayed in Netscape Navigator

```
</fo:block>
```

```
</fo:float>
```

shows the rendered file displayed in

Netscape Navigator using the Acrobat plug-in.

```
</fo:block>
```

格式化程序尽可能地将图形放在与 fo:float 周围的内容同页的某个位置上, 尽管这种情况并非永远存在, 但在此情况下, 格式化程序会将此对象移到后续页上。在这些限制范围内, 可任意将图形放在此页的任何地方。



## 15.14 XSL 格式化属性

由字面意思可知，格式化对象相对来讲完全没有说明如何格式化内容。它们只是将内容摆放在各个绝对的框中，而这些框放置在一页中的各个特定部分。各种格式化对象的特性确定如何设置这些框中内容的样式。

正像已经介绍的那样，大约有 200 个独立的格式化属性。并非所有的属性都可以与所有的元素关联。例如，指定 `fo:display-graphic` 的 `font-style` 的特点就不很多。但是，大多数属性都可以用于多种格式化对象元素（只有少数几个不能，如 `href` 和 `provisional-label-separation`，在上面已与使用它们的格式化对象一起讨论过了）。当一个属性为多个格式化对象所共有时，它的句法和含义就与这些对象相同。例如，使用相同代码来将 `fo:list-label` 格式化成 14 磅粗体 Times，就如同将 `fo:block` 格式化成 14 磅粗体 Times 那样。

许多 XSL 属性都与 CSS 属性类似。CSS 的 `font-family` 属性值与 XSL 的 `font-family` 特性值是一样的。如果已经阅读过第 12 章和 13 章，那么就已经学完了 XSL 属性一半以上的内容。

### 15.14.1 单位和数据类型

XSL 格式化属性的值可能是一关键字，如 `auto`、`italic` 或 `transparent`；或者为文字值，如 `true`、`5px`、`-5.0cm` 或 `http://www.w3.org/index.html`。在 XSL 中，文字值是以 24 个数据类型之一来表示，它们数据类型列于表 15-2 中。

表 15-2 格式化属性数据类型

数据类型	定义	实例
Name	表示 XML 名称记号	q1  copyright
ID	表示唯一的 XML 名称记号	q1  copyright
IDREF	表示与文档中元素的 ID 相匹配的名称	q1  copyright
Boolean	为字符串 <code>true</code> 或字符串 <code>false</code>	False  True
Char	单一的、无空白的 Unicode 字符	A  —
Signed Integer	一系列数字，作为选项可用加号或减号作前缀	0  - 28  +1000000000
Unsigned Integer	一系列数字	0

		28  1000000000
Positive Integer	一系列数字，包括至少有一个非零数字	28  1000000000
Signed Real	浮点数值，以符号-数字-小数点-数字格式表示。不支持指数表示法。对于正数的+为可选的	+0.879  -31.14  2.71828
Unsigned Real	非负的浮点数，其格式为数字-小数点-数字。不支持指数表示法	0.0  31.14  2.71828
Positive Real	正浮点数，其格式为数字-小数点-数字。不支持指数表示法	0.01  31.14  2.71828
Signed Length	后面带有单位的有符号整数或有符号实数	5px  -0.5in
Unsigned Length	后面带有单位的无符号整数或无符号实数	10px  0.5cm
Positive Length	后面带有单位的正整数或正实数	10px  1pc
Percent	能被 100 整除以获得其真正值的有符号实数	100.0  -43.2  0.0
Space Specifier	最小长度；最大长度；最佳长度；优先级；限制条件	0px;72px;12px; force;discard
Limit Specifier	最小长度分号最大长度	0px;72px
Color	命名颜色或#RRGGBB 形式的十六进制的三元参数	white  #FFFFFF

URI	统一源标识符；实际上为 URL	http://www.w3.org/index.html /index.html/ ../index.html
Language	ISO 639 语言码	en  la
Font Name	以实际名或符号表示的字体名	Times New Roman  serif
Font List	以逗号（并可能有空白）分隔的字体名	Times New Roman, Times, serif
Enumeration	XML 的枚举类型	(airplane   train   car   horse)
String	任何的字符序列	Fred Lucy and Ethel Castles don' t have phones.

## 15.14.2 消息属性

有两个消息属性，它们可用于任何格式化对象。但是，两者对格式化都没有直接的影响。从本质上讲，它们都是非格式化属性。

### 15.14.2.1 id 属性

第一个这样的属性就是 id。它是一个 XML 的 ID 类型的特性。所以，此属性值必须是样式单中和输出格式化对象文档内的唯一的 XML 名。由于样式单中的一个模板规则可能在输出文档中产生几百个元素，所以最后的要求就有点棘手。XSL 变换的 generate-id() 函数在此处就很有用。

### 15.14.2.2 语言属性

第二个这样的属性就是 language。它指定此元素中所包括的内容的语言。通常，此属性值是 ISO 639 语言码，如 en(English) 或 la(Latin)。它还可以为关键字 none 或 use-document。后者表示只使用 xml:lang 特性指定的输入文档的语言。例如，以凯撒的 Gallic Wars 第一首诗为例：

```
<fo:block id="versel.1.1" language="la">
```

```
Gallia est omnis divisa in partes tres,
```

```
quarum unam incolunt Belgae, aliam Aquitani,
```

```
tertiam qui ipsorum lingua Celtae, nostra Galli appellantur
```

```
</fo:block>
```

尽管 language 属性对格式化没有直接的影响，但如果格式化程序根据语言选择布局算法的话，那么它就可能直接的影响。例如，格式化程序可以将不同的缺省书写模式用于阿拉伯语和英语的文本。这就导致如何确定开始和结束的区域，以及内联的行进方向。

## 15.14.3 段落属性

- 在传统的字处理程序中，段落属性应用于整个文本块时，通常被作为样式来看待，尽管在这里作为块级文本属性或许更恰当些。例如，缩进是一种段落属性，这是因为是将一个段落进行缩进，而不能将单词独立于包含它的段落加以缩进。

#### 15.14.3.1 分隔符属性

分隔符属性指定放置分页符的位置。有五个关系不太紧密的分隔符属性：

- keep-with-next
- keep-with-previous
- break-before
- break-after
- inhibit-line-breaks

keep-with-next 和 keep-with-previous 属性都是布尔类型，它们指定格式化对象是否与下面的和前面的格式化对象分别处于相同父格式化对象中。这对保持两个格式化对象处于同一页有影响，但这种影响比那种与父格式化对象的定位关系更苛刻。

break-before 属性将一分隔符插入到格式化对象的开始之前。要进行分隔的内容可能有 column、page、odd-page 和 even-page。值也可能为 none 或 auto-page。break-after 属性将一分隔符插入到格式化对象的完成之后。可将相同的值用于 break-before。例如，下面的模板规则确保每个 SONNET 在其自身的一页中有足够小的打印尺寸。

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

<fo:block break-before="page" break-after="page">

<xsl:apply-templates/>

</fo:block>

</xsl:template>
```

最后，inhibit-line-breaks 也是布尔值，它可以设置为 true，以指示哪怕是一个换行符也不允许有，更不用提分页符了。

#### 15.14.3.2 断字属性

断字属性确定是否允许用连字符号连接以及如何使用。它只用于软或“随意”连字号，如有时在一行的结束用于分隔长单词的连字号。它不用于硬连字号，如单词 mother-in-law 中的连字号。尽管这些连字号可能影响软连字号的位置。有如下六个断字属性：

- hyphenate：只有在布尔属性的值为 true 时，才允许自动断字
- hyphenation-char：用于以连字符号连接单词的 Unicode 字符，如英语中的-
- hyphenation-keep：四个关键字之一（column、none、page、spread），指定在相对页或栏的结束处是否使用连字符号连接
- hyphenation-ladder-count：无符号整数，指定一行中连字号线条的最大个数
- hyphenation-push-char-count：无符号整数，指定必须跟在自动插入的连字符后面的最少字符个数（分开短音节显得不利）
- hyphenation-remain-char-count：无符号整数，指定必须跟在自动插入的连字符前面的最少字符个数

断字还依赖于所使用的语言和文字。因此，下面的这三个属性具有特殊的效果：

- country
- language
- script

例如：

```
<fo:block hyphenate=true

hyphenation-char="-"

hyphenation-keep="none"

hyphenation-ladder-count="2"

hyphenation-push-char-count="4"

hyphenation-remain-char-count="4" >

some content ...

</fo:block>
```

XSL 不指定分隔音节的算法来确定在何处使用软连字符。甚至对于允许使用连字符连接的属性，仍然完全要由格式化程序来决定如何以连字符来连接各个单词。

### 15.14.3.3 垂直对齐属性

vertical-align 属性确定其行上的格式化对象的垂直位置，与同名的 CSS2 属性的行为是一样的。此属性可能有下列八个关键字的值：

1. baseline：将框的基线与行框的基线对齐
2. sub：将框的基线与行框内部的下标基线对齐
3. super：将框的基线上升到行框中的上标基线处
4. top：将框的顶端与行框的顶端对齐
5. middle：将框的中点对齐于行框的基线加上行框 x 高度的一半处
6. bottom：将框的底部与行框的底部对齐
7. text-top：将框的顶部与字体的顶部对齐
8. text-bottom：将框的底部与字体的底部对齐

还可以将 vertical-align 设置为有符号的长度，以提升或降低此框到与基线的指定距离。

### 15.14.3.4 缩进属性

缩进属性有四个：start-indent、end-indent、text-indent 和 last-line-end-indent，它们指定缩进的行离文本边有多远。start-indent 属性从开始边（英文中为左边）移动所有的行。end-indent 属性从结束边（英文中为右边）移动所有的行。text-indent 属性只从开始边移动第一行。last-line-end-indent 属性只从开始边移动最后一行。这些值是以有符号的长度来表示的。对 start-indent 取正值，text-indent 取负值，就会创建一个悬挂式缩进（hanging indent）。例如，可按照下列方式，格式化一个第一行缩进 0.5 英寸的标准段落：

```
<fo:block text-indent="0.5in">
```

The first line of this paragraph is indented

```
</fo:block>
```

按下列方式，将一个块引用格式化成左右两边所有的行都缩进 1 英寸：

```
<fo:block start-indent="1.0in" end-indent="1.0in">
```

This text is offset one inch from both edges.

```
</fo:block>
```

#### 15.14.4 字符属性

字符属性描述各个字符的性质，尽管它们可应用于包含字符的元素（如 fo:block 和 fo:list-item-body）。这些属性包括颜色、字体、样式、粗细以及类似的属性。

##### 15.14.4.1 颜色属性

color 属性设置内容的背景颜色，其句法与 CSS 的 color 属性相同。例如，下面将文本“Lions and tigers and bears, oh my!”着成粉红色：

```
<fo:inline-sequence color="#FFCCCC">
```

Lions and tigers and bears, oh my!

```
</fo:inline-sequence>
```

##### 15.14.4.2 字体属性

处理文本的任何格式化对象都有广泛的字体属性。其中的大多数都已从 CSS 中了解到，其中包括：

- font-family: 按优先级顺序排列的一组字体名
- font-size: 有符号长度
- font-size-adjust: 通常为在 x 高度和字体大小之间的比值，以无符号的实数值或 none 来表示
- font-stretch: 字体的“宽度”，以下列一个关键字来表示：condensed、expanded、extra-condensed、extra-expanded、narrower、normal、semi-condensed、semi-expanded、ultra-condensed、ultra-expanded 或 wider
- font-style: 字体的样式，指定为下列关键字之一：italic、normal、oblique、reverse-normal 或 reverse-oblique
- font-variant: 为 normal 或 small-caps
- font-weight: 绘制字体的笔划的粗细，以下列关键字之一给出：100、200、300、400、500、600、700、800、900、bold、bolder、lighter、normal

##### 15.14.4.3 text-transform 属性

text-transform 属性定义文本如何大写，与同名的 CSS 属性一致。有四个可能的值：

- none：不改变大小写（缺省值）
- capitalize：每个单词的第一个字母大写，后续的所有字母都为小写
- uppercase：所有的字母都大写
- lowercase：所有的字母都小写

此属性多少与语言有关（例如，汉语不分大小写）。当将格式化程序用于非拉丁文-1 的文本时，就会任意忽略大小写的推荐值。

#### 15.14.4.4 text-shadow 属性

text-shadow 属性在文本中使用阴影（shadow），类似于背景色，但区别是，阴影加在文本本身之上，而不是加在含有文本的框之上。

```
<fo:inline-sequence text-shadow="FFFF66">
```

This sentence is yellow.

```
</fo:inline-sequence>
```

#### 15.14.4.5 text-decoration 属性

text-decoration 属性与 CSS2 的 text-decoration 属性相同，它有下列五个可能值：

- none
- underline
- overline
- line-through
- blink

缺省值为 none。

#### 15.14.4.6 score-space 属性

划线(Scoring)是个包罗万象的词，用于下划线(underline)、删除线(line through)、双删除线(double strike-through)等等。score-space 属性确定是否在空白处划线。例如，If score-space is true, an underlined sentence looks like this.（如果 score-space 为 true，则加上下划线的句子与此类似）。If Score-space is false, an underlined sentence looks uke this.（如果 score-space 为 false，则加上下划线的句子于此类似）。

### 15.14.5 句子属性

句子属性用于字符组，也就是说，每次只对一个以上的字符才有意义，如字母或单词之间的空格。

#### 15.14.5.1 字符间距属性

文本的字距调整测定两个字符分开的空间有多大，这种测定方法需要小心对待。它不是一个绝对值。大多数格式化程序基于本地需要 #0;#0;特别是在对齐文本方面 #0;#0;来调整字母之间的空间。此外，高度品质的字体在不同的字形之间使用不同数量的空间。但是，可以使文本完全变得更宽松或更紧密。

letter-spacing 属性在每对字形之间添加额外的空间，此空间超过字距调整所获得的空间。以有符号的长度来表示，以指定加入的额外空间的数量。例如：

```
<fo:block letter-spacing="1.5px">
```

This is fairly loose text

```
</fo:block>
```

可以使长度为负值，以紧缩文本。但格式化程序通常对允许在字母之间加入或移走多少额外空间具有限制。

#### 15.14.5.2 字间距属性

word-spacing 属性调整单词之间的间隔数量。另外，它的行为很像字符间距属性。其值为一有符号的长度，表示在两个单词之间要加入的额外空间的数量。例如：

```
<fo:block word-spacing="0.3cm">
```

This is pretty loose text.

```
</fo:block>
```

#### 15.14.5.3 行距属性

XSL 格式化引擎将块区域分成行区域。不能从 XSL 中直接创建行区域。但是，使用下面的五个属性，就可以对如何在垂直方向留下间隔施加影响：

- **line-height**：一行的最小高度
- **line-height-shift-adjustment**：如果下标和上标大到一行的高度，则为 **consider-shifts**；否则为 **disregard-shifts**
- **line-stacking-strategy**：**line-height**（CSS 模型，为缺省值）；**font-height**（加上 **font-height-override-before** 和 **font-height-override-after** 之后，使行与字体一样高）；或者 **max-height**（最大的上行字母高度和最大的下行字母深度之间的距离）
- **font-height-override-after**：有符号长度，指定在每行之后添加额外的垂直空间；也可以为关键字 **use-font-metrics**（缺省值）来指示根据字体设置垂直空间距离
- **font-height-override-before**：有符号长度，指定在每行之前添加最小的额外垂直空间；也可以为关键字 **use-font-metrics**（缺省值）来指示根据字体设置垂直空间距离

在很大程度上，行高还依赖于行的字体大小。字体越大，其行自然也就越高。例如，下面的开始段落来自于 Mary Wollstonecraft 的 *Of the Rights of Woman*，实际上为二倍间隔：

```
<fo:block font-size="12pt" line-height="24pt">
```

In the present state of society it appears necessary to go

back to first principles in search of the most simple truths,

and to dispute with some prevailing prejudice every inch of

ground. To clear my way, I must be allowed to ask some plain

questions, and the answers will probably appear as



unequivocal as the axioms on which reasoning is built;

though, when entangled with various motives of action, they

are formally contradicted, either by the words or conduct

of men.

</fo:block>

#### 15.14.5.4 文本对齐属性

text-align 和 text-align-last 属性指定内联内容在其框内如何水平对齐。可有六个值：

1. start：在从右往左的文字中进行左对齐
2. centered：居中
3. end：在从右往左的文字中右对齐
4. justify：按照需要使用额外的空间对文本进行扩大，以便填满整行
5. page-inside：与页的内侧对齐，也就是说，两个面对页的左页上的右边或两个面对页的右页上的左边
6. page-outside：与页的外侧对齐，也就是说，两个面对页的左页上的左边或两个面对页的右页上的右边

text-align-last 属性能够用来为块中的最后一行指定不同的值。这对于对齐文本尤为重要，因为在文本中最后行通常单词不多，需要进行调整。其值可为 start、end、justified 和 relative。relative 值使用的值与 text-align 属性值是一样的，除非 text-align 为 justified，在这种情况下，最后行与开始边对齐。

#### 15.14.5.5 空白属性

whitespace-treatment 属性指定在最初的源文档转换成格式化对象之后，格式化引擎使用仍旧存在的空白来做什么。可以为下列三个值：

1. preserve：按原样保留空白
2. collapse：将所有的空白压缩成一个空格
3. ignore：删除首尾空白

我偏爱于保留在转换之后仍剩下的所有的空白。如果空白毫无意义，那么就很容易使用 xsl:strip-space 让转换过程删除空白。

#### 15.14.5.6 wrap-option（折行选项）属性

wrap-option 属性确定如何处理太长的无法容纳在一行中的文本。此属性有两个关键字值：

1. wrap：将文本软折行到下一行
2. no-wrap：文本不折行

## 15.14.6 区域属性

区域属性施加于框。这些属性既可以是块级也可以是内联框。每个框都有：

- 背景
- 页边距
- 边界
- 贴边
- 大小

### 15.14.6.1 背景属性

背景属性基本上与 CSS1 的背景属性相同，有五个属性：

- `background-color` 属性指定框背景的颜色。其值可以是颜色或关键字 `transparent`。
- `background-image` 属性给出用作背景影像的 URI。其值可以为关键字 `none`。
- `background-attachment` 属性指定背景影像是否加到窗口或文档上。其值为 `fixed` 或 `scroll` 关键字之一。
- `background-position` 属性指定背景影像如何放在框中。其值有 `center`、`left`、`right`、`bottom`、`middle`、`top` 或给出坐标。
- `background-repeat` 属性指定如果背景影像比其框小时，如何以及是否将背景影像平铺显示。可能值为 `repeat`、`no-repeat`、`repeat-x` 和 `repeat-y`。

下列的块表明了 `background-image`、`background-position`、`background-repeat` 和 `background-color` 的用法：

```
<fo:block background-image="/bg/paper.gif"
```

```
background-position="0,0"
```

```
background-repeat="repeat"
```

```
background-color="white">
```

```
Two strings walk into a bar ...
```

```
</fo:block>
```

### 15.14.6.2 边框属性

边框属性描述框周围边界的外观。它们几乎都与 CSS 边框属性相同。但是，除了 `border-XXX-bottom`、`border-XXX-top`、`border-XXX-left` 和 `border-XXX-right` 属性之外，XSL 版本还有 `border-XXX-before`、`border-XXX-after`、`border-XXX-start` 和 `border-XXX-end` 版本。总共有 31 个边界属性，它们是：

- 颜色： `border-color`、`border-before-color`、`border-after-color`、`border-start-color`、`border-end-color`、`border-top-color`、`border-bottom-color`、`border-left-color`、`border-right-color`。缺省边界颜色为黑色。
- 宽度： `border-width`、`border-before-width`、`border-after-width`、`border-start-width`、`border-end-width`、`border-top-width`、`border-bottom-width`、`border-left-width`、`border-right-width`。
- 样式： `border-style`、`border-before-style`、`border-after-style`、`border-start-style`、`border-end-style`、`border-top-style`、`border-bottom-style`、`border-left-style`、`border-right-style`。
- 缩略属性： `border`、`border-top`、`border-bottom`、`border-left`、`border-right`、`border-color`、`border-style`、`border-width`。

例如，下面的语句在一个块周围绘制一个 2 像素宽的蓝色框：

```
<fo:block border-before-color="blue" border-before-width="2px"
```

```
border-after-color="blue" border-after-width="2px"
```

```
border-start-color="blue" border-start-width="2px"
```

```
border-end-color="blue" border-end-width="2px">
```

Two strings walk into a bar ...

```
</fo:block>
```

### 15.14.6.3 贴边属性

贴边属性指定框的边界和框的内容之间的空间数量。框的边界线如果显示出来的话，就落在页边距和贴边之间。贴边属性绝大多数与 CSS 的贴边属性相同。但是，除了 padding-bottom、padding-top、padding-left 和 padding-right 属性之外，XSL 版本还有 padding-before、padding-after、padding-start 和 padding-end 版本。因此，总共有八个贴边属性，每个都是以有符号的长度作为值。它们是：

- padding-after
- padding-before
- padding-bottom
- padding-end
- padding-left
- padding-start
- padding-right
- padding-top

例如，下面块的所有边都有 0.5 宽的贴边：

```
<fo:block padding-before="0.5cm" padding-after="0.5cm"
```

```
padding-start="0.5cm" padding-end="0.5cm">
```

Two strings walk into a bar ...

```
</fo:block>
```

### 15.14.6.4 块的页边距属性

有五个页边距属性，每个的值都为无符号的长度，它们是：

- margin-top
- margin-bottom
- margin-left
- margin-right
- margin

但是，这些属性在此只用于与 CSS 兼容。通常，推荐使用下列属性来代替，它们能更好地满足 XSL 格式化模型：

- space-before

- space-after
- start-indent
- end-indent

space-before 和 space-after 属性分别与 margin-top 和 margin-bottom 严格地等价。start-indent 属性等于 padding-left、border-left-width 和 margin-left 之和。end-indent 属性与 padding-right、border-right-width 和 margin-right 之和相等。图 15-3 对此作了更清晰地说明。

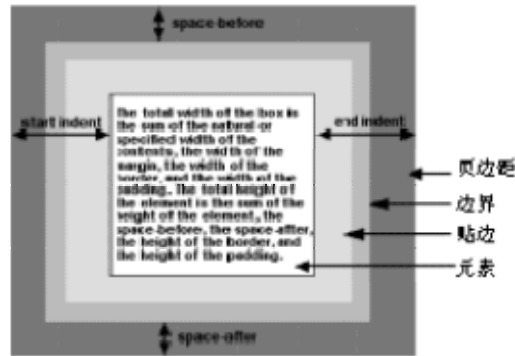


图 15-3 贴边、缩进、边界和 XSL 框之前和之后的空间

例如，下面的块在其开始和结束侧边保留 0.5 厘米的页边距：

```
<fo:block start-indent="0.5cm" end-indent="0.5cm">
```

```
Two strings walk into a bar ...
```

```
</fo:block>
```

#### 15.14.6.5 内联框页边距属性

用于内联元素的页边距属性有两个，它们是：

- space-end
- space-start

它们的值是空间说明符，以便在元素之前和之后添加额外的空间范围。实际的空间可能更小或更大。由于空间不是框本身的一部分，所以框的结束空间可以是下一框的开始空间的部分。

空间说明符的值范围有最小、最大和最佳值。格式化程序可任意选取此范围内的值，以便适应页的限制。此外，空间说明符还包括优先级和限制条件值。所有的这五个值都是用分号分开的。

优先级既可以是整数，也可以为关键字 force。优先条件确定当内联区域的 space-end 与下一个 space-start 发生冲突时，应出现什么情况。优先级更高的区域占上风。缺省的优先级为 0。

限制条件为两个关键字之一：discard 或 retain。这些关键字确定一行的结束处对额外空间产生什么影响。缺省值为放弃额外空间。

- 15.14.6.6 内容高度和宽度属性

有四个属性可用来指定框的内容区域的高度和宽度，都是无符号的长度值，它们是：

- height
- width
- max-height
- max-width

这些属性不指定框的总宽度和总高度，也不指定页边距、贴边和边界。只指定内容区域的宽度和高度。除了无符号长度之外，可以将 height 和 width 属性设置为关键字 auto，以基于框中内容的数量来选择高度和宽度。但是，在任何情况下，高度和宽度都不会比 max-height 和 max-width 属性指定的值大。例如：

```
<fo:block height="2in" width="2in">
```

```
Two strings walk into a bar ...
```

```
</fo:block>
```

#### 15.14.6.7 overflow（溢出）属性

overflow 属性确定如果内容太多无法容进指定大小的框中时要出现什么情况。可以是使用尺寸属性的显式说明，或者是基于页大小或其他限制的隐式说明。有四个可能性，每一种都使用一个关键字来表示：

1. auto：如果有超出部分，则使用滚动条；如果没有，则不使用滚动条
2. hidden：不显示到达框之外的任何内容
3. scroll：将滚动条加到框上，以便读者可以滚动其他内容
4. visible：如果需要，不考虑框的大小限制，显示完整的内容

如果 overflow 属性没有 visible 值，那么 clip 属性就指定剪切区的形状。缺省的剪切区只是框的本身。但是，可以像下面这样指定一个特定的矩形，从而改变这种情况：

```
clip=rect(top_offset right_offset bottom_offset left_offset)
```

此处的 *top\_offset*、*right\_offset*、*bottom\_offset* 和 *left\_offset* 为有符号的长度，以框的顶、右、底和左侧的剪切区的偏移量来表示。这样可以使剪切区比框的本身更小。

#### 15.14.6.8 reference-orientation（参考方位）属性

reference-orientation 属性可用来指定框的内容以相对于正常方向的 90° 增量旋转。唯一的有效值为 90° 增量，是以反时针方向计算的，也就是说可以为 0、90、180 和 270。还可以指定为 -90、-180 和 -270。例如，下面为 90° 旋转：

```
<fo:block reference-orientation="90">
```

```
Bottom to Top
```

```
</fo:block>
```

#### 15.14.6.9 书写方式属性

书写方式指定框中的内容的方向。这对框中格式化对象的排列具有重要的意义。在大多数时候，说英语和其他西方语言的人采取的都是左至右、上至下的书写方式，譬如：

A B C D E F G

H I J K L M N

O P Q R S T U

V W X Y Z

而在说希伯来语和阿拉伯语的国家里，书写顺序为右至左、上至下，如下面的方式似乎更自然：

G F E D C B A

N M L K J I H

U T S R Q P O

Z Y X W V

在台湾，由上至下、由右至左的顺序更轻松自在：

Y U Q M I E A

Z V R N J F B

W S O K G C

X T P L H D

在 XSL 格式化语言中，书写方式不只影响文本，还影响文本流或序列中对象的排列、换行等等。读者已注意到许多属性都可以用来控制开头、结尾、前和后的变化，而不是左、右、上和下的变化。根据开头、结尾、前和后，而不是左、右、上和下指定的样式规则会产生更加强健、更具有本地化特点的样式单。

writing-mode 属性指定某一地区的书写方式。此属性可取下列 14 个关键字值之一：

1. bt-lr：自下而上、从左往右
2. bt-rl：自下而上、从右往左
3. lr-alternating-rl-bt：从左往右数行交替为从右往左数行、自下而上
4. lr-alternating-rl-tb：从左往右一行，然后从右往左一行交互排列、自上而下
5. lr-bt：从左往右、自下而上
6. lr-inverting-rl-bt：从左往右，然后向上移到下一行并从右往左（即，如同自下而上的 S 那样迂回向上到达整个页面）
7. lr-inverting-rl-tb：从右往左，然后向下移到下一行并从右往左（即，如同反向的 S 那样迂回向下到达整个页面）
8. lr-tb：从左往右、自上而下
9. rl-bt：从右往左、自下而上

10. rl-tb: 从右往左、自上而下
11. tb-lr: 自上而下、从左往右
12. tb-rl: 自上而下、从右往左
13. tb-rl-in-rl-pairs: 自上而下、从右往左
14. use-page-writing-mode: 显示此对象的页使用任意一种书写方式: 此为缺省值

#### 15.14.6.10 孤行和寡行

对于排字机来说,孤行(orphan)是在页面底端段落的首行。寡行(widow)是在页面顶端段落的末行。好的排字机可以将额外的一行按要求从前页移到下一行,以避免孤行和寡行现象。可以将orphans属性设置为无符号的整数,从而调整认为是孤行的行数。可以将widows属性设置为无符号的整数,从而调整认为是寡行的行数。例如,如果想确保页尾的每个段落都至少有三行,可将orphans属性设置为3。例如:

```
<fo:simple-page-master page-master-name="even"
orphans="3" page-height="8.5in" page width="11in"
/>
```

#### 15.14.7 听觉属性

XSL 支持全部的 CSS2 音频样式单属性的集合,包括:

- azimuth
- cue
- cue-after
- cue-before
- elevation
- pause
- pause-after
- pause-before
- pitch
- pitch-range
- play-during
- richness
- speak
- speak-header
- speak-numeral
- speak-punctuation
- speech-rate
- stress
- voice-family
- volume



听觉样式单属性在第 13 章“级联样式单级别 2”的最后一节中讨论过。在 XSL 格式化对象中,这些属性的语义和

句法与 CSS2 中的完全一样。



## 15.15 本章小结

本章中，详细地学习了 XSL 格式化语言。特别是，学习了如下内容：

- XSL 变换可用来完成 XML 源文档到以 XSL 格式化对象符号集标记的新的 XML 文档的转换。
- 大多数 XSL 格式化对象生成一个或多个矩形区域。页面区域包含块区域。块区域包含块区域和行区域。行区域包含内联区域。内联区域包含其他内联区域和字符区域。
- 格式化对象文档的根元素为 `fo:root`，包含 `fo:layout-master-set` 和 `fo:page-sequence` 元素。
- `fo:layout-master-set` 元素包含一个或多个 `fo:simple-page-master` 元素，而每个元素通过将特定类型页的版面分成五个区（前、后、开头、结尾和正文），并给每个元素赋以属性，来定义页的版面。
- `fo:page-sequence` 元素包含一个 `fo:sequence-specifier` 元素、零个或多个 `fo:static-content` 元素以及一个 `fo:flow` 元素。`fo:flow` 的内容按照 `fo:sequence-specifier` 元素指定的顺序复制到主控页实例中。`fo:static-content` 元素的内容复制到它所创建的第一页上。
- `fo:display-rule` 元素生成块级水平线。`fo:inline-rule` 元素生成内联水平线。
- `fo:display-graphic` 元素从 URL 处加载影像，并显示在块中。`fo:inline-graphic` 元素从 URL 处加载影像，并以内联方式加以显示。
- `fo:simple-link` 元素创建对 URL 的超文本链接，并显示在块中。
- 列表是块级元素，由 `fo:list-block` 元素创建，包含块级 `fo:list-item` 元素。每个 `fo:list-item` 元素又包含一个 `fo:list-item-label` 和 `fo:list-item-body`。
- `fo:page-number` 元素插入当前的页码。
- `fo:character` 元素使用输出文档中的不同字符，来代替输入文档中一个特指的字符或字符串。
- `fo:display-sequence` 和 `fo:inline-sequence` 元素都是容器，用来将属性加到它们包含的文本和区域上。
- `fo:footnote` 元素将外联的脚注和内联的脚注引用插入到页中。
- `fo:float` 元素将外联的块级元素（如图或 `pullquote`）插入到页中。
- 有 200 多个独立的 XSL 格式化属性，其中有许多与同名的 CSS 属性是一样的。它们都是作为特性与 XSL 格式化对象的元素相关联。

下一章将介绍 XLink，这是标准 HTML 的 A 元素超链接或 XSL 的 `fo:display-link` 和 `fo:inline-link` 更强大的链接句法。

## 第四部分 补充技术

本部分包括以下各章：

第 16 章--XLink

第 17 章--XPointer

第 18 章--命名域

第 19 章--资源描述框架

*注意：本章内容基于 XLink 的早期草案，与现在的规范有较大出入。但为了保证完整性，我们仍将本章完整刊出。本章的最新英文版本见 <http://www.ibiblio.org/xml/books/bible/updates/16.html>，XLink 规范最新版本见 <http://www.w3.org/TR/xlink/>。*

XML 中国论坛

### 第 16 章 XLink

XLL（可扩展的链接语言，eXtensible Linking Language）分为两部分：XLink 和 XPointer。XLink（即 XML 链接语言，XML Linking Language）定义一文档如何与另一文档的链接。XPointer（即 XML 指针语言，XML Pointer Language）定义文档的各部分如何寻址。XLink 指向 URI（实际为 URL），以指定特定的资源。此 URL 可能包含 XPointer 部分，更明确地标识目标资源或文档所期望的部分或节。本章探讨 XLink，下一章将探讨 XPointer。

本章的主要内容如下：

- XLink 与 HTML 链接的对比
- 简单链接
- 扩展链接
- 外联链接
- 扩展链接组
- 如何重新命名 XLink 特性

#### 16.1 XLink 与 HTML 链接的对比

Web 征服已经建立起来的 Gopher 协议的一个主要原因是：在文档中可嵌入超文本链接。这些链接可以嵌入影像或让用户从一文档内部跳转到另一文档或同一文档的另一部分。在某种程度内，XML 可转变成其他格式以便于浏览，HTML 用于链接的句法与 XML 文档中使用的句法是一样的。使用 XSL，可将各自的句法转变成 HTML 句法，就像第 14 章的几个例子中所看到的那样。

但是，HTML 链接具有局限性。首先，URL 通常只限于指向单一文档。如果比这种要求更为详细如链接于一文档中第 17 段的第三句，就需要手工在目标文件中插入命名的定位符（锚）。如果对链接的文档没有写访问权，是不会做到这一点的。

而且，HTML 链接不保留文档之间的历史或关系内容。尽管浏览器可以跟踪浏览的一系列文档的路径，但这种跟踪是很不可靠的。从 HTML 内部，没有任何方法知道读者是从哪里来的。链接纯粹是单方向的。用来链接的文档知道它正与谁进行链接，但反过来则不行。

XLL 可获得文档间的更强有力的链接。它是专为 XML 文档设计的，但有些部分也可以与 HTML 文档一起使用。XLL 可以实现使用 HTML 的基于 URL 超文本链接和定位可获得的所有功能。但是，除此之外，它还支持多方位的链接，即以多个方向同时进行链接。任何元素都可以成为一个链接，而不仅仅是 A 元素。甚至不需要将链接保存在与链接文档相同的文件中。此外，XPointer 部分（将在下一章讨论）允许对 XML 文档中的任意位置进行链接。这些功能使 XLL 不仅更适用于新的用途，而且还适合于只使用 HTML 要花很大气力才能达到的功能，如交叉引用、脚注、尾注、互连数据等等。



请读者注意，直到编写此书时（1999 年春天），XLL 仍处于重大的开发和修改阶段。尽管正在逐渐成形，但在读者阅读本书时可能会或多或少地发生变化。

此外，到目前为止，还没有任何一个多用途的应用程序能支持任意的 XLink。这是因为 XLink 的适用性要比 HTML 链接广得多。XLink 不仅仅用于超文本的连接，还可用于在文档中嵌入影像。可被任何一个需要在文档和文档的局部之间建立连接的常用应用程序用于任何目的。因此，甚至当 XLink 在浏览器中得以完整执行时，也许并非总是单击可跳转到另一页的蓝色下划线文本。可以是那样，但也可以根据需要决定蓝色的下划线文本的多寡。

## 16.2 简单链接

在 HTML 中，链接是用<A>标记来定义的。但就像 XML 使用描述元素的标记更灵活一样，使用引用外部资源的标记也更为灵活。在 XML 中，几乎任何标记都可以是一个链接。包括链接的元素称作链接元素（linking element）。

链接元素是由值为 simple 或 extended 的 xlink:form 特性来标识的。而且，每个链接元素包含一个值为链接资源的 URI 的 href 特性。例如，下面是三个链接元素：

```
<FOOTNOTE xlink:form="simple" href="footnote7.xml">7</FOOTNOTE>
```

```
<COMPOSER xlink:form="simple" inline="true"
```

```
href="http://www.users.interport.net/~beand/">
```

```
Beth Anderson
```

```
</COMPOSER>
```

```
<IMAGE xlink:form="simple" href="logo.gif" />
```

注意，此元素具有描述它们所包含内容的语义名称，而不是这些元素如何表现。这些元素使链接的信息包含在标记的特性中。

这三个例子是简单的 XLink。简单的 XLink 类似于标准的 HTML 链接，并在更复杂（以及功能强大）的扩展链接之前很可能为应用程序的软件所支持，所以，我首先使用它们。扩展链接在下节讨论。

在上面的 FOOTNOTE 实例中，链接目标特性名为 href。其值为相对的 URL footnote7.xml。此文档的协议、主机以及路径都取自出现这种链接的文档中的协议、主机以及路径。

在上面的 COMPOSER 示例中，链接目标特性名为 href。此 href 特性值为绝对的 URL http:

//wwwusers.interport.net/~beand/。在上面的第三个示例 IMAGE 中，链接目标特性名为 href。此 href 特性值为相对的 URL logo.gif。这时同样本文档的协议、主机以及路径都取自出现这种链接的文档中的协议、主机以及路径。

如果文档有一个 DTD，那么这些特性必须和其他特性一样进行声明。例如，FOOTNOTE、COMPOSER 和 IMAGE 元素的 DTD 声明可以按下面的方式进行：

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

```
<!ATTLIST FOOTNOTE
```

```
xlink:form CDATA #FIXED "simple"
```

```
href CDATA #REQUIRED
```

```
>
```

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

```
<!ATTLIST COMPOSER
```

```
xlink:form CDATA #FIXED "simple"
```

```

href CDATA #REQUIRED

>

<!ELEMENT IMAGE EMPTY>

<IATTLIST IMAGE

xlink:form CDATA #FIXED "simple"

href CDATA #REQUIRED

>

```

使用这些声明，xlink:form 特性就有一个确定值。所以，这一特性就不需要包括在元素的实例中，现在可以将这些元素按照下列方式书写得更简洁一些：

```

<FOOTNOTE href=" footnote7.xml" >7</FOOTNOTE>

<COMPOSER href=" http://www.users.interport.net/~beand/" >

Beth Anderson

</COMPOSER>

<IMAGE href=" logo.gif" />

```

使一元素成为链接元素对元素的其他特性或内容不存在限制。链接元素可以包含任意的子元素或其他特性，当然总是受制于 DTD。例如，下面为 IMAGE 元素的更真实的声明。注意，大多数特性与链接无关。

```

<!ELEMENT IMAGE EMPTY>

<IATTLIST IMAGE

xlink:form CDATA #FIXED "simple"

href CDATA #REQUIRED

ALT CDATA #REQUIRED

HEIGHT CDATA #REQUIRED

WIDTH CDATA #REQUIRED

>

```

### 16.2.1 本地资源的描述

链接元素可以包含可选的 content-role 和 content-title 元素，这两个元素用于在链接元素出现的文档内提供附加的信息，并进一步描述此链接的目的。例如：

```

<AUTHOR href=http://www.macfaq.com/personal.html

```

```
content-title="author of the page"
```

```
content-role="whom to contact for questions about this page">
```

```
Elliotte Rusty Harold
```

```
</AUTHOR>
```

content-role 和 content-title 特性描述本地资源，即链接元素的内容（本例中的 Elliotte Rusty Harold）。但是，这些特性不描述远程的资源（如本例中为位于 <http://www.macfaq.com/personal.html> 处的文档）。因此，本例说明 Elliotte Rusty Harold 具有“author of the page”的头衔，其作用为“whom to contact for questions about this page”。本例也无需与在 <http://www.macfaq.com/personal.html> 处找到的文档有任何关系。

content-title 特性通常是由读入 XML 的应用程序所使用，以便在用户将鼠标移到链接的元素之上时，在浏览器状态条上或通过工具提示为用户显示一些附加信息。但是，应用程序不一定要为用户显示这种信息。如果此特性选择了这么做，那么它就只能如此。

content-role 特性表示文档中链接元素的目的。此特性与准备将数据传递给读入 XML 的应用程序中的处理指令相类似。可是，它的真正目的并不是作为 XML 来使用，并且应用程序可以任意忽略此特性。

像所有的其他特性一样，content-title 和 content-role 为了用于包含它们的所有元素也应在 DTD 中进行声明。例如，下面的合理声明可用于上面的 AUTHOR 元素：

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

```
<!ATTLIST AUTHOR
```

```
xlink:form CDATA #FIXED "simple"
```

```
href CDATA #REQUIRED
```

```
content-title CDATA #IMPLIED
```

```
content-role CDATA #IMPLIED
```

### 16.2.2 远程资源的描述

链接元素可以包含可选的 role 和 title 特性，用来描述远程资源，即链接所指向的文档或其他资源。例如：

```
<AUTHOR href=http://www.macfaq.com/personal.html
```

```
title="Elliotte Rusty Harold s personal home page"
```

```
role="further information about the author of this page"
```

```
content-title="author of the page"
```

```
content-role="whom to contact for questions about this page">
```

```
Elliotte Rusty Harold
```

</AUTHOR>

role 和 title 特性描述远程资源，而不是本地元素。在上面的实例中，远程资源就是 <http://www.macfaq.com/personal.html> 处的文档。因此，下面的实例说明 <http://www.macfaq.com/personal.html> 处的网页标题为“Elliott Rusty Harold's personal home page”，作用为“further information about the author of this page”。要使 title 与链接网页的 TITLE 元素内容相同是很平常的，尽管不必这样做。

读入 XML 的应用程序可以使用这两个特性来为用户显示附加的信息。但应用程序无需将这种信息显示给用户或用它来做任何事。

在链接文档（链接出发的文档）中，role 特性说明远程资源（被链接的文档）的目的。例如，可用特性来区别脚注、尾注和引文。

与所有的其他特性一样，为了用于包含它们的所有元素，应在 DTD 中声明 title 和 role 特性。例如，下面的合法声明可用于上面的 author 元素：

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

```
<!ATTLIST AUTHOR
```

```
xlink:form CDATA #FIXED "simple"
```

```
href CDATA #REQUIRED
```

```
content-title CDATA #IMPLIED
```

```
content-role CDATA #IMPLIED
```

```
title CDATA #IMPLIED
```

```
role CDATA #IMPLIED
```

```
>
```

### 16.2.3 链接行为

链接元素可以包含三个可选特性，这些特性可以建议应用程序如何将远程资源与当前页关联。下面即为这三种特性：

1. show
2. actuate
3. behavior

show 特性提示当激活链接时，应如何显示内容，例如，通过打开一个新窗口来保存内容。actuate 特性提示此链接是否可以自动切断或是否要求有明确的用户请求。behavior 特性可为应用程序提供有关如何准确地切断链接的详细信息，如在切断链接之前的一段时间迟延。但是，这些特性都是与应用程序相关的，并且应用程序可任意忽略这些提示。

#### 16.2.3.1 show 特性

show 特性有三个合法值：replace、new 和 embed。

当激活链接（通常是由单击此链接而发生的，至少在 GUI 浏览器中是如此）时使用 `replace` 值，则链接的目标代替同一个窗口中的当前文档。这是 HTML 链接的缺省行为。例如：

```
<COMPOSER href="http://www.users.interport.net/~beand/"
```

```
show="replace">
```

Beth Anderson

```
</COMPOSER>
```

使用 `new` 值时，激活链接就打开新的窗口，以显示目标资源。这种行为与 `target` 特性设置为 `_blank` 时的 HTML 链接类似。例如：

```
<WEBSITE href="http://www.quackwatch.com/" show="new">
```

Check this out, but don't leave our site completely!

```
</WEBSITE>
```



读者不希望单击链接后打开新的窗口，倒希望在单击链接后，把新页加载到当前窗口中，除非明确地要求在新窗口中打开这种链接。

有些公司相当自傲，以至他们认为任何一个用户从不会离开他们自己的站点。于是，他们就“帮助”读者打开新的窗口。在大多数时候，这只能使读者感到困惑和厌恶。如果没有一个很好的理由，就不要改变用户所期望的那种行为。让读者在站点上花费额外的两秒钟，或者多浏览一页，多看一页的广告，这种浮浅的欲望是毫无道理的。

使用 `embed` 值，激活链接将会在现有的文档中插入目标资源。其准确的含义是与应用程序相关的。但是，可以想象，此值用于 Web 页的客户端“嵌入”功能。例如，下面的这个元素（并没有直接包括家庭成员的各个元素）将家庭成员的各个元素从各自的文件 `ThomasCorwinAnderson.xml`、`LeAnahDeMintEnglish.xml`、`JohnJayAnderson.xml` 和 `SamuelEnglishAnderson.xml` 中复制出来。

```
<FAMILY ID="f732">
```

```
<HUSBAND href="ThomasCorwinAnderson.xml" show="embed"/>
```

```
<WIFE href="LeAnahDeMintEnglish.xml" show="embed"/>
```

```
<CHILD href="JohnJayAnderson.xml" show="embed"/>
```

```
<CHILD href="SamuelEnglishAnderson.xml" show="embed"/>
```

```
</FAMILY>
```

切断链接并将其内容嵌入到 `FAMILY` 元素中之后，结果如下所示：

```
<FAMILY ID="f732">
```

```
<PERSON ID="p1035" SEX="M">
```

```
<NAME>
```



<GIVEN>Thomas Corwin</GIVEN>

<SURNAME>Anderson</SURNAME>

</NAME>

<BIRTH>

<DATE>24 Aug 1845</DATE>

</BIRTH>

<DEATH>

<PLACE>Mt. Sterling, KY</PLACE>

<DATE>18 Sep 1889</DATE>

</DEATH>

</PERSON>

<PERSON ID="p1098" SEX="F">

<NAME>

<GIVEN>LeAnah (Lee Anna, Annie) DeMint</GIVEN>

<SURNAME>English</SURNAME>

</NAME>

<BIRTH>

<PLACE>Louisville, KY</PLACE>

<DATE>1 Mar 1843</DATE>

</BIRTH>

<DEATH>

<PLACE>acute Bright s disease, 504 E. Broadway</PLACE>

<DATE>31 Oct 1898</DATE>

</DEATH>

</PERSON>

<PERSON ID="p1102" SEX="M">

<NAME>

<GIVEN>John Jay (Robin Adair )</GIVEN>

<SURNAME>Anderson</SURNAME>

</NAME>

<BIRTH>

<PLACE>Sideview</PLACE>

<DATE>13 May 1873</DATE>

</BIRTH>

<DEATH>

<DATE>18 Sep 1889 </DATE>

</DEATH>

</PERSON>

<PERSON ID="p37" SEX="M">

<NAME>

<GIVEN>Samuel English</GIVEN>

<SURNAME>Anderson</SURNAME>

</NAME>

<BIRTH>

<PLACE>Sideview</PLACE>

<DATE>25 Aug 1871</DATE>

</BIRTH>

<DEATH>

<PLACE>Mt. Sterling, KY</PLACE>

<DATE>10 Nov 1919</DATE>

</DEATH>

</PERSON>

</FAMILY>

尽管每个 PERSON 元素都存在于独立的文件中，但处理全部的 FAMILY 元素就像是在一个文件中一样。

像合法文档中的所有特性一样，对于 DTD 的链接元素，show 特性必须在<!ATTLIST>声明语句加以声明。例如：

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

```
<!ATTLIST WEBSITE
```

```
xlink:form CDATA #FIXED "simple"
```

```
href CDATA #REQUIRED
```

```
show (new | replace | embed) "new"
```

```
>
```

### 16.2.3.2 actuate 特性

链接元素的 actuate 特性有两个可能的值：user 和 auto。user 值为缺省值，它指定仅当用户请求时，才切断链接。另一方面，如果链接元素的 actuate 特性设置成 auto，则在同一个链接元素的其他目标资源被切断时，都要切断此链接。

正如合法文档中的所有特性一样，对于出现链接的链接元素，actuate 特性必须在 DTD 的<!ATTLIST>声明语句中声明。例如：

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

```
<!ATTLIST WEBSITE
```

```
xlink:form CDATA #FIXED "simple"
```

```
href CDATA #REQUIRED
```

```
show (new | replace | embed) "new"
```

```
actuate (user | auto) "user"
```

```
>
```

### 16.2.3.3 behavior 特性

behavior 特性用来将任意格式的任意数据传递给读入此数据的应用程序中。应用程序使用这些数据来对如何进行链接作出附加说明。例如，如果要指定在切断链接时，播放声音文件 fanfare.au，可按下面进行编写：

```
<COMPOSER xlink:form="simple"
```

```
href="http://www.users.interport.net/~beand/"
```

```
behavior="sound: fanfare.au">
```

Beth Anderson

</COMPOSER>

用于 DTD 的快捷方式

由于特性名及类型都是标准化的，如果在一篇文档中有多个链接元素，那么将特性声明变成参数实体引用，并只在每个链接元素的声明中重复这种引用常常是方便的。例如：

```
<!ENTITY % link-attributes
"xlink:form CDATA #FIXED ¢ simple¢
href CDATA #REQUIRED
behavior CDATA #IMPLIED
content-role CDATA #IMPLIED
content-title CDATA #IMPLIED
role CDATA #IMPLIED
title CDATA #IMPLIED
show (new | replace | embed) ¢ new¢
actuate (use | auto) ¢ use¢
behavior CDATA #IMPLIED"
>

<!ELEMENT COMPOSER (#PCDATA)>

<!ATTLIST COMPOSER
%link-attributes;
>

<!ELEMENT AUTHOR (#PCDATA)>

<!ATTLIST AUTHOR
%link-attributes;
>

<!ELEMENT WEBSITE (#PCDATA)>

<!ATTLIST WEBSITE
%link-attributes;
>
```

但是，这样就要求读入 XML 文件的应用程序理解带有值为 sound:fanfare.au 的 behavior 特性即意味着当切断链接时，应播放 fanfare.au 声音文件。大多数（或许所有的）应用都不能理解这种含义。但是，它们可以将 behavior 特性当作易于使用的保存它们确实能够理解的非标准信息的地方。

正合法文档中的所有特性一样，对于出现链接的链接元素，其 behavior 特性必须在 DTD 中声明才行。例如：下面的 COMPOSER 元素可按下面方式声明：

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

```
<!ATTLIST COMPOSER
```

```
  xlink:form CDATA #FIXED "simple"
```

```
  href CDATA #REQUIRED
```

```
  behavior CDATA #IMPLIED
```

```
>
```

## 16.3 扩展链接

简单链接的效果或多或少地与 HTML 中已经熟悉了的标准链接类似。每个简单链接都包含一个本地资源和对一个远程资源的引用。本地资源为链接元素的内容，而远程资源则为链接的目标。

但是，扩展链接（Extended link）实质上超越了 HTML 链接所能达到的程度，以便在许多文档和外联链接之间包括多方向的链接。扩展链接由 xlink:form 特性来指定，其值为 extended，如：

```
<WEBSITE xlink:form="extended">
```

扩展链接的第一个作用就是指向多个目标。为此，扩展链接将目标保存在链接元素的子 locator 元素中，而不像简单链接那样保存在链接元素的唯一的 href 特性中。例如：

```
<WEBSITE xlink:form="extended">Cafe au Lait
```

```
<locator href="http://metalab.unc.edu/javafaq/">
```

North Carolina

```
</locator>
```

```
<locator
```

```
href="http://sunsite.univie.ac.at/jcca/mirrors/javafaq/">
```

Austria

```
</locator>
```

```
<locator href="http://sunsite.icm.edu.pl/java-corner/faq/">
```

Poland

```
</locator>
```

```
<locator href="http://sunsite.uakom.sk/javafaq/">
```

Slovakia

```
</locator>
```

```
<locator href="http://sunsite.cnlab-switch.ch/javafaq/">
```

Switzerland

```
</locator>
```

```
</WEBSITE>
```

本例中的链接元素 WEBSITE 本身和各 locator 子元素都可以有特性。链接元素只有适用于整个链接以及本地资源的特性，如 content-title 和 content-role。locator 元素具有应用于特定的远程资源（locator 元素链接于这些资源）的特性，如 role 和 title。例如：

```
<WEBSITE xlink:form="extended" content-title="Cafe au Lait"
content-role="Java news">

<locator href="http://metalab.unc.edu/javafaq/"
title="Cafe au Lait" role=".us"/>

<locator
href="http://sunsite.univie.ac.at/jcca/mirrors/javafaq/"
title="Cafe au Lait" role=".at"/>

<locator href="http://sunsite.icm.edu.pl/java-corner/faq/"
title="Cafe au Lait" role=".pl"/>

<locator href="http://sunsite.uakom.sk/javafaq/"
title="Cafe au Lait" role=".sk"/>

<locator href="http://sunsite.cnlab-switch.ch/javafaq/"
title="Cafe au Lait" role=".ch"/>

</WEBSITE>
```

actuate、behavior 和 show（如果存在）属于各个 locator 元素。

在有些情况下，如上面的实例所示，各定位符（locator）指向同一页面和镜像副本，对于各个 locator 元素来说，远程资源特性可以相同，都指向链接元素。在此情况下，可以在链接元素中使用远程资源特性。这些特性应用于各个 locator 子元素（对于同一个特性各子元素声明为相同的值）。例如：

```
<WEBSITE xlink:form="extended" content-title="Cafe au Lait"
content-role="Java news" title="Cafe au Lait">

<locator href="http://metalab.unc.edu/javafaq/" role=".us"/>

<locator
href="http://sunsite.univie.ac.at/icca/mirrors/javafaq/"
role=".at"/>

<locator href="http://sunsite.icm.edu.pl/java-corner/faq/"
```

```
role=".pl"/>
```

```
<locator href="http://sunsite.uakom.sk/javafaq/" role=".sk"/>
```

```
<locator href="http://sunsite.cnlab.switch.ch/javafaq/"
```

```
role=".ch"/>
```

```
</WEBSITE>
```

DTD 的另一种快捷方式

如果有多个链接和 locator 元素,那么在 DTD 的参数实体中定义通用的特性可能有好  
处,而这样又可以重新用于不同的元素。例如:

```
<!ENTITY % remote-resource-semantics.att
```

```
"role CDATA #IMPLIED
```

```
title CDATA #IMPLIED
```

```
show (embed | eplace | new) #IMPLIED ¢ replace¢
```

```
actuate (auto | use ) #IMPLIED ¢ use¢
```

```
behavior CDATA #IMPLIED"
```

```
>
```

```
<!ENTITY % local-resource-semantics.att
```

```
"content-title CDATA #IMPLIED
```

```
content-role CDATA #IMPLIED"
```

```
>
```

```
<!ENTITY % locator.att
```

```
"href CDATA #REQUIRED"
```

```
>
```

```
<!ENTITY % link-semantics.att
```

```
"inline (true | false) ¢ true¢
```

```
role CDATA #IMPLIED"
```

```
>
```

```
<!ELEMENT WEBSITE (locator*) >
```



```
<!ATTLIST WEBSITE  
  
xlink:form CDATA #FIXED "extended"  
  
%local-resource-semantics.att;  
  
>
```

```
<!ELEMENT locator EMPTY>  
  
<!ATTLIST locator  
  
xlink:form CDATA #FIXED "locator"  
  
%locator.att;  
  
%link-semantics.att;  
  
>
```

就像通常的那样，在合法的文档中，链接元素及其所有可能的特性都必须在 DTD 中声明。例如，下面声明上面实例中使用的 WEBSITE 和 locator 元素及其特性：

```
<!ELEMENT WEBSITE (locator*) >  
  
<!ATTLIST WEBSITE  
  
xlink:form CDATA #FIXED "extended"  
  
content-title CDATA #IMPLIED  
  
content-role CDATA #IMPLIED  
  
title CDATA #IMPLIED  
  
>  
  
<!ELEMENT locator EMPTY>  
  
<!ATTLIST locator  
  
xlink:form CDATA #FIXED "locator"  
  
href CDATA #REQUIRED  
  
role CDATA #IMPLIED  
  
>
```

## 16.4 外联链接

迄今为止所考虑的链接（简单和扩展）都是内联链接。内联链接（如在 HTML 中熟悉的 A 元素）使用内联元素的内容作为包含链接的文档部分。通过这种方式展示给读者。

XLink 也可以是外联方式。外联链接可能不存在于它所连接的任何文档中，而是将链接保存在各个独立的链接文档中。例如，使用这种方法来维护幻灯片放映是很有用的，因为在幻灯片放映过程中，各幻灯片都需要前后链接。改变链接文档中的幻灯片顺序，即可以改变每页上的前后链接的目标，而无需编辑幻灯片本身。

要将链接标记为外联，可将 inline 特性设置成 false 值。例如，下面的简单的外联链接使用空元素来描述 Web 站点。空元素没有任何内容；在链接的情况下，它没有本地资源。所以，它没有描述本地资源的 content-role 和 content-title 特性。但像下面的这个例子那样，可以有描述远程资源的 role 和 title 特性。

```
<WEBSITE xlink:form="simple" inline="false"

href="http://metalab.unc.edu/xml/"

title = "Cafe con Leche" role="XML News"/>
```



由于到目前为止所看到的所有链接都是内联链接，所以链接隐式地具有值为 true（缺省值）的 inline 属性。

简单的外联链接（如上面的例子）都是相当少见的。极其常用并且非常有用的是外联扩展链接，如下面所示：

```
<WEBSITE xlink:form="extended" inline="false">

<locator href="http://metalab.unc.edu/javafaq/" role=".us"/>

<locator

href="http://sunsite.univie.ac.at/jcca/mirrors/javafaq/"

role=".at"/>

<locator href "http://sunsite.icm.edu.pl/java-corner/faq/"

role=".pl"/>

<locator href="http://sunsite.uakom.sk/javafaq/" role=".sk"/>

<locator href="http://sunsite.cnlab-switch.ch/javafaq/"

role=".ch"/>

</WEBSITE>
```

有些链接（如上面的链接）可能保存在已知位置的 Web 服务器上的独立文件中，在此位置浏览器可以找到并且询问此链接，以便确定对浏览器正在寻找最近的页面镜像。但是，外联性就是该元素不出现在激活链接的文档中。

这样就将样式单的提取扩大到链接域。样式单完全与其描述的文档分离，并且提供的规则可以用来修改文档向读者显示的方式。包含外联链接的链接文档与它所连接的文档分离，但仍给读者提供必要的链接。这种方法有几个优点，其中包括可以使更多面向展示的标记保持与文档分离，以及允许只读文档的链接。



样式单链接的范围比外联链接要广得多。目前还没有如何将“链接单 (link sheet)” 加到 XML 文档中的一般性的建议，更不用说如何确定文档中的哪个元素与哪个链接相关联。

显而易见，可以将 `<?xml-linksheet?>` 处理指令加到文档的前言中，以指定在何处找到链接。链接单本身可以使用类似于 XSL 的内容来选择模式，以便将链接映射到各 XML 元素中。甚至选择符也会成为 locator 元素的 role 特性的值。

## 16.5 扩展链接组

扩展链接组（extended link group）元素包含连接一组特定文档的链接。依靠扩展链接文档元素，组中的每个文档都作为目标来定位。应用程序负责推定如何激活组成员中的连接、并怎么理解这种连接。



我不得不提醒读者，在撰写这本书时，应用程序支持链接组最多只是一种假定。尽管我可以显示如何书写这样的链接，但真正执行并支持可能还需要一段时间。有些细节无法确定，很可能以销售商指定的方式执行，至少开始就是如此。还有，这些链接能够获得比 HTML 更为复杂的链接。

### 16.5.1 一个实例

例如，我已经将我讲授的 Java 课程的注解放在我的 Web 站点上。图 16-1 显示前言页。这个特别的课程由 13 个课时组成，每个课时含有 30~60 页的注解。然后为各个课时提供一张目录。这几百页组成整个站点，其中的每一页都与前面文档、下个文档以及每周目录（顶端链接）相链接，如图 16-2 所示。把这些页放在一起，这样总计多达几千页，这些页可以在文档内相互连接。

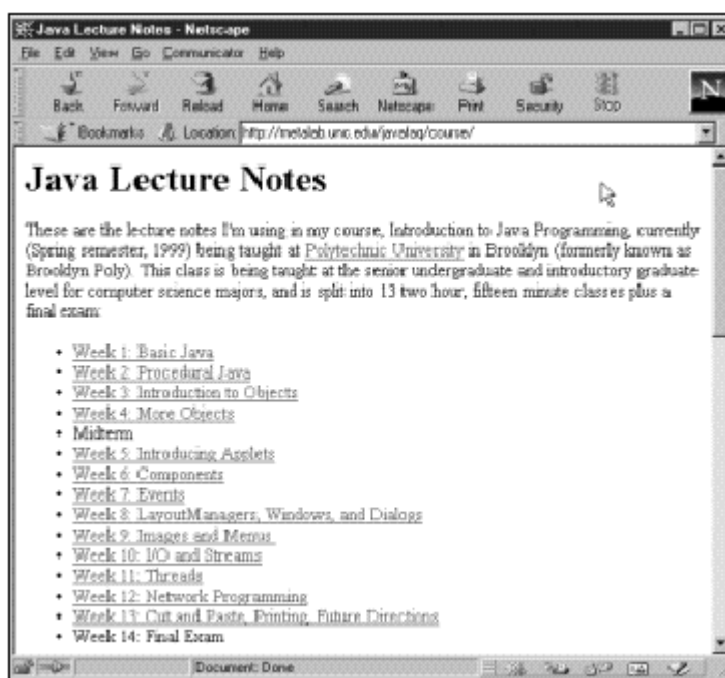


图 16-1 用于类 Web 站点的前言页显示 13 个星期的讲稿注解

可能相互连接数随着文档数量呈指数增长。每当一个文档移动、改名或分成更小的块时，就需要在页面上、在这组文档的前和其后的页面上以及每周目录上调整链接。坦率地说，这项工作比原先的更加艰苦，所以这妨碍了对课程注解的必要的修改和更新。

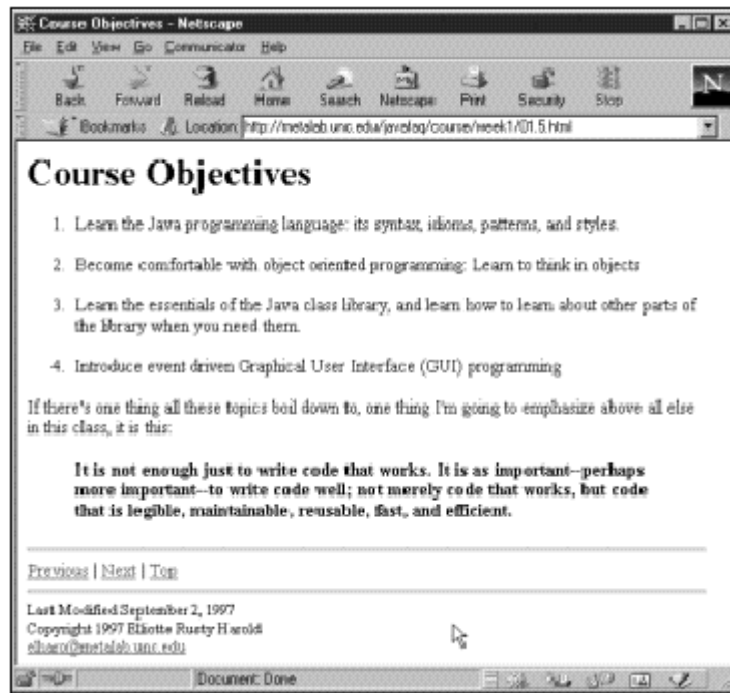


图 16-2 显示 Previous、Next 和 Top 链接的一页讲稿注解

如果 HTML 支持的话，要做的更有意义的事就是将连接保存在独立的文档中。然后编辑此文档，就可以重新组织页。HTML 链接不支持这种方式，但 XLink 却支持。不是以内联的方式将链接保存在 HTML 文件中，而是将它们通过外联的方式保存在组元素中。例如：

```
<COURSE xlink:form="group">

<CLASS xlink:form="document" href="week1/index.xml"/>

<CLASS xlink:form="document" href="week2/index.xml"/>

<CLASS xlink:form="document" href="week3/index.xml"/>

<CLASS xlink:form="document" href="week4/index.xml"/>

<CLASS xlink:form="document" href="week5/index.xml"/>

<CLASS xlink:form="document" href="week6/index.xml"/>

<CLASS xlink:form="document" href="week7/index.xml"/>

<CLASS xlink:form="document" href="week8/index.xml"/>

<CLASS xlink:form="document" href="week9/index.xml"/>

<CLASS xlink:form="document" href="week10/index.xml"/>

<CLASS xlink:form="document" href="week11/index.xml"/>

<CLASS xlink:form="document" href="week12/index.xml"/>

<CLASS xlink:form="document" href="week13/index.xml"/>
```

</COURSE>

这样就将 COURSE 元素定义成扩展链接组，此组由 13 个扩展链接文档元素（即 CLASS 元素）组成。

### 16.5.2 steps 特性

应用程序使用链接组可以做的事情之一是，预加载链接组中的所有文档。这些文档可以包含它们各自的链接组。例如，上面的每个 CLASS 元素都引用一个特定星期的站点目录，如图 16-3 所示。然后这些文档就可以加载。例如，文件 week6/index.xml 就包含这种链接组：

```
<CLASS xlink:form=" group" >

<SLIDE xlink:form=" document" href=" 01.xml" />

<SLIDE xlink:form=" document" href=" 02.html" />

<SLIDE xlink:form=" document" href=" 06.html" />

<SLIDE xlink:form=" document" href=" 12.html" />

<SLIDE xlink:form=" document" href=" 13.html" />

<SLIDE xlink:form=" document" href=" 16.html" />

<SLIDE xlink:form=" document" href=" 17.html" />

<SLIDE xlink:form=" document" href=" 19.html" />

<SLIDE xlink:form=" document" href=" 21.html" />

<SLIDE xlink:form=" document" href=" 22.html" />

<SLIDE xlink:form=" document" href=" 24.html" />

</CLASS >
```



图 16-3 显示周讲稿注解的目录页面

现在假定有一个文档反过来引用原文档。这有可能触发无限的回归，即重复加载同一个文档，直到应用程序将内存耗尽为止。为了防止这种情况的发生，组元素可以包含 `steps` 特性，用它来指定递归跟随链接组的层数。例如，要指定预加载不能达到当前文档三层以上，可以这样来编写：

```
<group xlink:form="group" steps="3">
```



坦率地说，我不敢确定 `steps` 特性有多重要。要使应用程序注意何时它已经到达某个文档，但根本不再次处理此文档并非困难。我认为，更好的方法是，由 XML 处理器而不是网页作者来防止递归。

`steps` 特性可以用来限制预发生加载的数量。例如，在课时注解实例中，尽管有可能他或她想打印或复制所有的课程注解，不可能任何人一次要阅读整个内容。在任何情况下，将 `steps` 特性设置为 1，就可以将横穿的深度限制为指定的页面而不是课程中的几百页。

就像常常要做的那样，这些元素及其特性必须在它们的任何合法文档的 DTD 中声明。实际上，`xlink:form` 是固定的，所以不需要包括在元素的实例中。例如：

```
<!ELEMENT CLASS (document*)>
```

```
<!-- ATTLIST CLASS
```

```
xlink:form CDATA #FIXED "group"
```

```
steps CDATA #IMPLIED
```

```
>
```

```
<!-- ELEMENT SLIDE EMPTY-->
```

```
<!-- ATTLIST SLIDE
```

```
xlink:form CDATA #FIXED "document"
```

```
href CDATA #REQUIRED
```

```
>
```

## 16.6 重命名 XLink 特性

XLink 有十个特性，这在前节中已经讨论过。现列于下面：

xlink:form

href

steps

title

role

content-title

content-role

show

actuate

behavior

可以想像，这些特性之一或多个已用作特定 XML 应用程序中的特性名。title 特性似乎特别要加以考虑。只有一个特性不能用于其他用途，这就是 xlink:form。

XLink 规范预料到这个问题，所以可以利用 xml:attributes 特性将 XLink 特性重新命名为更便于使用的名称。在 DTD 的 `<!ATTLIST>` 中将此特性声明为一个固定的属性，类型为 CDATA，而值为以空格分开的标准名和新名对的列表。



这种问题可以使用命名域（在第 18 章中讨论）来解决。如果在未来的 XLL 草案中将此结构整个地删除，并用简单的命名域作前缀（如 xlink:）时，我并不感到惊讶。

例如，本章展示的链接元素有点滑稽，因为标准名都是小写字母，而本书的约定都是用大写字母。按照下面的方法，使用声明语句，就很容易将 XLink 特性变成大写字母：

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

```
<!ATTLIST WEBSITE
```

```
xlink:form CDATA #FIXED "simple"
```

```
xml:attributes CDATA #FIXED
```

```
"href HREF show SHOW actuate ACTUATE"
```

```
HREF CDATA #REQUIRED
```

```
SHOW CDATA (new | replace | embed) "new"
```

```
ACTUATE CDATA (user | auto) user
```



>

现在就可以更谐调地重新编写 WEBSITE 例子：

```
<WEBSITE HREF="http://www.microsoft.com/" SHOW="new">
```

Check this out, but don't leave our site completely!

```
</WEBSITE>
```

上面的 ATTLIST 声明只改变 WEBSITE 元素的特性。如果要在其他多个例子中用同样的方式改变这些特性，最容易的方法就是使用参数实体：

```
<!ENTITY LINK_ATTS
```

```
xlink:form CDATA #FIXED "simple"
```

```
xml:attributes CDATA #FIXED
```

```
"href HREF show SHOW actuate ACTUATE"
```

```
HREF CDATA #REQUIRED
```

```
SHOW CDATA (new | replace | embed) "new"
```

```
ACTUATE CDATA (user | auto) "user" >
```

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

```
<!ATTLIST WEBSITE %LINK_ATTS;>
```

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

```
<!ATTLIST COMPOSER %LINK_ATTS;>
```

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

```
<!ATTLIST FOOTNOTE %LINK_ATTS;>
```

## 16.7 本章小结

在本章中学习了 XLink。特别是学习了如下内容：

XLink 可以做 HTML 链接能够做的任何事情，并还会更多，但不为当前的应用程序所支持。

简单链接特别类似于 HTML 链接，但不受单个<A>标记的限制。

链接元素使用 xlink:form 和 href 来指定。

链接元素使用 content-title 和 content-role 来描述本地资源。

链接元素使用 title 和 role 来描述链接的远程资源。

链接元素使用 show 特性来告诉应用程序当激活链接时应如何显示内容，如打开一个新的窗口。

链接元素使用 behavior 特性来将详细的、由应用程序决定的有关如果准确地切断此链接的信息提供给应用程序。

链接元素使用 actuate 特性来告诉应用程序没有明确的用户请求是否应切断链接。

扩展链接可以在链接元素中包括多个 URI。目前，由应用程序来决定如何在不同的可能性中挑选。

扩展链接组元素包含连接特定一组文档的链接列表。

可在 DTD 中使用 xml:attributes 特性来对标准的 XLink 特性（如 href 和 title）重新命名。

在下一章中，将会看到如何使用 XPointer 来链接远程文档，而且如何与远程文档的特指元素进行链接。

**特别提醒：我在制作这个文档的时候，因为文档的刊登站点提醒第 17 章（Xpoint）已经和现有的标准有很大的出入，所以我没有再将第 17 章提取下来，有要的朋友请自己到 <http://www.xml.net.cn/>**

**去察看** [Chicken](#)

## 第 18 章 命名域

XML 的用途不是单一的。虽然读者可能看到编写只使用一个标记符号集的文档是相当有用的（第 4 和第 5 章的棒球运动就是如此），但将不同的 XML 应用程序的标记混合，并进行匹配，甚至更为有用。例如，可将 BIOGRAPHY 元素包括在各个 PLAYER 元素中。由于传记基本上是由自由形态的、格式化的文本组成，所以，以结构整洁的 HTML 格式编写它就非常方便，而无需从零做起重新定义所有的用于段落、分行符、列表项、粗体元素等等的标记。

但是，问题是，当混杂和匹配不同的 XML 应用程序的标记时，可能会发现同一个标记已用于两个不同对象。TITLE 是指页标题还是书的标题？ADDRESS 是指公司的邮件地址还是 Web 站点管理人员的电子邮件地址？命名域（namespace）可以解决这些诸如此类的问题，它是将 URI 与各标记集相关联，并在每个元素前加上一个前缀，以表示它属于哪个标记集。于是，就可以有 BOOK:TITLE 和 HTML:TITLE 两个元素或 POSTAL:ADDRESS 和 HTML:ADDRESS 元素，而不只一类 TITLE 或 ADDRESS。本章将说明如何使用命名域。

本章的主要内容如下：

- 何为命名域？
- 命名域语法
- DTD 中的命名域

### 18.1 何为命名域

XML 能够使开发者为工程创建自己的标记语言。这些语言可以和工作于世界各地的类似工程的工作人员们共享。使用这种方式工作的典型实例之一就是 XSL。XSL 本身就是用于 XML 样式文档的一个 XML 应用程序。XSL 变换语言必须输出任意的、结构整洁的 XML，或许还包括 XSL 本身。因此，需要有明确的手段来区分何为 XSL 转换指令的 XML 元素、何为输出的 XML 元素，即便它们有相同的名称也得要区分开！

命名域就是这种解决方案。命名域允许文档中的每个元素和特性放在不同的命名域中。组成 XSL 转换指令的 XML 元素放在 <http://www.w3.org/XSL/Transform/1.0> 命名域中。成为输出部分的 XML 元素仍放在某个其他方便的命名域（如 <http://www.w3.org/TR/REC-html40> 或 <http://www.w3.org/XSL/Format/1.0>）中。只要命名域不同，那么命名域的精确性就不显得很重要。



如果熟悉 C++ 和其他程序语言命名域，那么在深入阅读本章之前，需要将以前的概念放置一边。XML 命名域与编程中使用的命名域类似，但不完全相同。特别是，XML 命名域没有必要组成一个集合（没有重名的集合）。

清单 15-2 是从源符号集转换到 XSL 格式化对象的变换，最早出现在第 15 章的“XSL 格式化对象”中。它显示了 XSL 样式单，可从输入 XML 转换成 XSL 格式化对象。格式化引擎使用命名域来区分作为 XSL 指令的元素和用于输出的文字数据。<http://www.w3.org/XSL/Transform/1.0> 命名域中的任何元素都表示一个转换指令。<http://www.w3.org/XSL/Format/1.0> 命名域中的任何元素包括输出部分。

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

```
<xsl:stylesheet
```

```
xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
```

```
xmlns:fo="http://www.w3.org/XSL/Format/1.0"
```

```
result-ns="fo" indent-result="yes">
```

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

```

<fo:root xmlns:fo="http://www.w3.org/XSL/Format/1.0">

<fo:layout-master-set>

<fo:simple-page-master page-master-name="only">

<fo:region-body/>

</fo:simple-page-master>

</fo:layout-master-set>

<fo:page-sequence>

<fo:sequence-specification>

<fo:sequence-specifier-single page-master-name="only"/>

</fo:sequence-specification>

<fo:flow>

<xsl:apply-templates select="//ATOM"/>

</fo:flow>

</fo:page-sequence>

</fo:root>

</xsl:template>

<xsl:template match="ATOM">

<fo:block font-size="20pt" font-family="serif">

<xsl:value-of select="NAME"/>

</fo:block>

</xsl:template>

</xsl:stylesheet>

```

更确切地说，下面这些元素存在于 <http://www.w3.org/XSL/Transform/1.0> 命名域中而且是 XSL 指令：

- stylesheet
- template
- apply-templates
- value-of

下面这些元素存在于 <http://www.w3.org/XSL/Format/1.0> 命名域中，是 XSL 格式化对象和输出部分：

- root
- layout-master-set
- simple-page-master
- region-body
- sequence-specification
- sequence-specifier-single
- page-sequence
- block

下面四个带有 xsl 前缀的元素使限定名具有以该前缀开始的：

- xsl:stylesheet
- xsl:template
- xsl:apply-templates
- xsl:value-of

但是，它们的完整名称使用 URL，而不是前缀：

- <http://www.w3.org/XSL/Transform/1.0:stylesheet>
- <http://www.w3.org/XSL/Transform/1.0:template>
- <http://www.w3.org/XSL/Transform/1.0:apply-templates>
- <http://www.w3.org/XSL/Transform/1.0:value-of>

实际上，由于 URL 经常包含如~、%和/这样的一些在 XML 名称中不合法的字符，所以作为别名的这种较短的限定名只用于文档内部。但是，限定名的确使文档更易于键入和阅读。



“XML 中的命名域”是正式的 W3C 标准。W3C 认为它相当完善，只是可能存在不太重要的错误和说明。但是，在 W3C 所有的 XML 规范中，正是这个命名域才最有争议。许多人非常强烈地觉得，这个标准有基本原理上的缺陷。主要的缺陷是命名域实际上与 DTD 和合法性不兼容。而我对此并没有强烈的某种看法，但我的确有这样的疑问：当人们没有达成一致意见时，发行一个标准是否明智。命名域是许多 XML 相关规范（如 XSL 和 XHTML）的至关重要的部分，所以需要人们理解。但很多开发者和读者都在他们的工作中忽略此规范。

## 18.2 命名域句法

命名域高于 XML 1.0 规范。XML 1.0 处理程序对命名域一无所知，但仍能阅读使用命名域的文档，并且不会发现任何错误。使用命名域的文档不破坏现有的 XML 分析程序（至少不进行合法性检查的分析程序是如此）；用户不必等待臭名昭著的、不准时的软件公司来发行昂贵的升级版才使用命名域。

### 18.2.1 命名域的定义

在使用命名域的有效元素上应用 `xmlns:prefix` 特性来定义命名域。*prefix* 由真正的用于命名域的前缀来代替。特性值为命名域的 URI。例如，`xsl:stylesheet` 标记将前缀 `xsl` 与 URI `http://www.w3.org/XSL/Transform/1.0` 联系在一起。

```
<xsl:stylesheet
```

```
xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
```

然后，`xsl` 前缀就可以加到 `xsl:stylesheet` 元素内的本地元素和特性名中，以便将它们标识为属于 `http://www.w3.org/XSL/Transform/1.0` 命名域。前缀通过冒号与本地名分开。清单 14-2 为用于周期表的基本的 XSL 样式单，它最初出现在第 14 章“XSL 变换”中，此清单演示了在 `stylesheet`、`template` 和 `apply-templates` 上使用 `xsl` 前缀的方法。

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

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
```

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

```
<html>
```

```
<xsl:apply-templates/>
```

```
</html>
```

```
</xsl:template>
```

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

```
<P>
```

```
<xsl:apply-templates/>
```

```
</P>
```

```
</xsl:template>
```

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

定义命名域的 URI 纯粹是形式上的，其唯一目的就是成组、并明确文档中的元素和特性。它无需指向任何对象，特别在无法确保 URI 位置上的文档描述了此文档中使用的句法；或者有用于该目的的任何文档存在于该 URI 位置。正如已说明过的，如果有一个用于特定 XML 应用程序的 URI，那么此 URI 就可以用于定义命名域。

命名域前缀可以是任何合法的 XML 名称（不能包含冒号）。回顾第 6 章的“结构整洁的 XML 文档”，合法的 XML 名必须以字母或下划线（\_）开头。名称中的后面的字符可以包括字母、数字、下划线、连字号和句点。但不能包括空白。



有两个前缀明确地不允许使用：`xml` 和 `xmlns`。`xml` 前缀是定义为用来引用 <http://www.w3.org/XML/1998/namespace> 的。`xmlns` 前缀用于将元素绑定到命名域上，所以不可用于绑定目标的前缀。

在 XML 名称中，除了不允许有冒号字符外（不包括用于分隔前缀和本地名的冒号），命名域对标准的 XML 句法没有直接的影响。使用命名域的文档必须也是结构整洁的，以便对命名域一无所知的处理程序可阅读此文档。如果文档需要检查合法性，那么它无需明确地考虑命名域就肯定能够获得通过。对于 XML 处理程序，使用命名域的文档只不过是样子古怪的文档，在此文档中有些元素和特性名可能有一个冒号。



命名域的确存在着合法性的问题。如果编写的 DTD 没有命名域前缀，那么必须使用命名域前缀来重新编写 DTD，才能用于对使用该前缀的文档进行合法性检查。例如，考虑下面的元素声明：

```
<!ELEMENT DIVISION (DIVISION_NAME, TEAM+)>
```

如果元素都是以 `bb` 作命名域前缀，就得按下面的方式重新编写：

```
<!ELEMENT bb:DIVISION (bb:DIVISION_NAME, bb:TEAM+)>
```

这意味着，不能将相同的 DTD 用于带有和不带有命名域的文档，即使这两类文档本来就使用相同的符号集也是如此。事实上，由于 DTD 受真正的前缀而不是命名域的 URI 的约束，所以甚至不能将同一个 DTD 用于使用相同的标记集和命名域、但前缀不同的文档中。

## 18.2.2 多个命名域

清单 14-2 并不真正将 HTML 元素放在命名域中，但要做到这一点则并不困难。清单 18-1 演示这种用法。正像 `xsl` 是 XSL 转换指令的惯用前缀一样，`html` 也是 HTML 元素的惯用前缀。在下面的实例中，`xsl:stylesheet` 元素声明两个不同的命名域：一个用于 XSL，另一个用于 HTML。

清单 18-1：使用 <http://www.w3.org/TR/REC-html40> 作为命名域用于输出的 XSL 样式单

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

<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
  xmlns:html="http://www.w3.org/TR/REC-html40">

  <xsl:template match="PERIODIC_TABLE">

    <html:html>

      <xsl:apply-templates/>

    </html:html>

  </xsl:template>

  <xsl:template match="ATOM">

    <html:p>
```

```
<xsl:apply-templates/>
```

```
</html:p>
```

```
</xsl:template>
```

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

虽然将 xmlns 特性放在根元素上已成为习惯，并且总的来说是很有用的，但也可以出现在其他元素上。在此情况下，命名域前缀只能在声明它的元素内才有效。考虑一下清单 18-2。html 前缀只在声明它的 xsl:template 元素中才合法。不能将其施加于其他的模板规则，除非这些模板规则分别声明 html 命名域。

清单 18-2：在模板规则中声明的带有 <http://www.w3.org/TR/REC-html40> 命名域的 XSL 样式单

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

```
<xsl:stylesheet
```

```
  xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
```

```
  <xsl:template match="PERIODIC-TABLE"
```

```
    xmlns:html="http://www.w3.org/TR/REC-html40">
```

```
      <html:html>
```

```
        <xsl:apply-templates/>
```

```
      </html:html>
```

```
    </xsl:template>
```

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

```
      <P>
```

```
        <xsl:apply-templates/>
```

```
      </P>
```

```
    </xsl:template>
```

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

可以在子元素中重新定义命名域。例如，清单 18-3 中的 XSL 样式单。此处的 xsl 前缀出现在不同的元素中，以交替引用 <http://www.w3.org/XSL/Transform/1.0> 和 <http://www.w3.org/XSL/Format/1.0>。尽管每个元素都有前缀 xsl，但由于 xsl 前缀的含义随元素而变，所以 XSL 转换指令和 XSL 格式化对象仍处于不同的命令位中。

清单 18-3：重新定义 xsl 前缀

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



```
<xsl:stylesheet

xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">

<xsl:template match="/">

<xsl:root xmlns:xsl="http://www.w3.org/XSL/Format/1.0">

<xsl:layout-master-set>

<xsl:simple-page-master page-master-name="only">

<xsl:region-body/>

</xsl:simple-page-master>

</xsl:layout-master-set>

<xsl:page-sequence>

<xsl:sequence-specification>

<xsl:sequence-specifier-single page-master-name="only"/>

</xsl:sequence-specification>

<xsl:flow>

<xsl:apply-templates select="//ATOM"/

xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"/>

</xsl:flow>

</xsl:page-sequence>

</xsl:root>

</xsl:template>

<xsl:template match="ATOM">

<xsl:block font-size="20pt" font-family="serif"

xmlns:xsl="http://www.w3.org/XSL/Format/1.0">

<xsl:value-of select="NAME"

xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"/>

</xsl:block>
```

</xsl:template>

</xsl:stylesheet>

但这样做会产生不必要的混乱，我强烈建议读者避免这样做。可供使用的前缀还有很多，几乎不需要在同一个文档中重复使用。不重复使用前缀的主要价值还在于，来自于不同作者的两个不同的文档碰巧重复使用类似的前缀，此时这两个文档就会组合在一起。这一点也是避免使用像 a、m 和 x 这样的短前缀的很好的理由，这些短前缀很可能重新用于不同的目的。

### 18.2.3 特性

由于特性属于特定元素，所以不使用命名域也可很容易地从类似的命名特性中确定出来。因此，像加到元素中那样，将命名域加到特性中几乎是没有什么必要的。例如，1999 年 4 月 21 日的 XSL 规范工作草案要求所有的 XSL 转换元素都要加入 <http://www.w3.org/XSL/Transform/1.0> 命名域，但是它不要求这些元素的特性也在任何特定命名域中（事实上，它要求元素的特性都不在任何命名域中）。但是，如果需要，可以将命名域前缀加入特性中。例如，下面的 PLAYER 元素和它所有的特性都处在 <http://metalab.unc.edu/xml/baseball> 命名域中。

```
<bb:PLAYER xmlns:bb=" http://metalab.unc.edu/xml/baseball"

bb:GIVEN_NAME=" Tom"  bb:SURNAME=" Glavine"

bb:POSITION=" Starting Pitcher"  bb:GAMES=" 33"

bb:GAMES_STARTED=" 33"  bb:WINS=" 20"  bb:LOSSES=" 6"  bb:SAVES=" 0"

bb:COMPLETE_GAMES=" 4"  bb:SHUT_OUTS=" 3"  bb:ERA=" 2.47"

bb:INNINGS=" 229.1"  bb:HOME_RUNS_AGAINST=" 13"

bb:RUNS_AGAINST=" 67"  bb:EARNED_RUNS=" 63"  bb:HIT_BATTER=" 2"

bb:WILD_PITCHES=" 3"  bb:BALK=" 0"  bb:WALKED_BATTER=" 74"

bb:STRUCK_OUT_BATTER=" 157" />
```

如果需要将两个不同 XML 应用程序中的特性组合到同一个元素中，这种方式有时或许也有用。

可以（虽然通常无意义）将同一个命名域 URI 与两个不同的前缀相关联。实在是没有道理这么做，不过需要提醒读者，我在此提出可以这样做的唯一前提是，对于带有相同名称的最多只有一个特性的一个元素来说，特性的全名必须满足 XML 规则。例如，由于 bb:GIVEN\_NAME 和 baseball:GIVEN\_NAME 是相同的，所以下面的形式是不合法的：

```
<bb:PLAYER xmlns:bb=" http://metalab.unc.edu/xml"

xmlns:baseball=" http://metalab.unc.edu/xml"

bb:GIVEN_NAME=" Hank"  bb:SURNAME=" Aaron"

baseball:GIVEN_NAME=" Henry" />
```

另一方面，URI 实际上并不领会它所指向的是什么对象。URI 的 <http://metalab.unc.edu/xml/> 和 <http://www.metalab.unc.edu/xml/指向同一页>。但下面的这种是合法的：

```

<bb:PLAYER xmlns:bb=" http://metalab.unc.edu/xml"

xmlns:baseball=" http://www.metalab.unc.edu/xml"

bb:GIVEN_NAME=" Hank"  bb:SURNAME=" Aaron"

baseball:GIVEN_NAME=" Henry"  />

```

#### 18.2.4 缺省的命名域

在有大量标记的长文档（在所有相同命名域）中，可能会发现要将前缀加到各个元素名中是很不方便的。可以使用没有前缀的 xmlns 特性，将缺省的命名域与某个元素及其子元素相关联。此元素本身（其所有的子元素也一样）可认为处于定义的命名域中，除非它们拥有明确的前缀。例如，清单 18-4 显示的 XSL 样式单就像习惯上的那样，没有使用 xsl 作为 XSL 转换元素的前缀。



特性从不处于缺省的命名域中，它们必须明确地作为加上前缀。

清单 18-4：使用缺省命名域的 XSL 样式单

```

<?xml version="1.0"?>

<stylesheet

xmlns="http://www.w3.org/XSL/Transform/1.0"

xmlns:fo="http://www.w3.org/XSL/Format/1.0"

result ns="fo">

<template match="/">

<fo:root xmlns:fo="http://www.w3.org/XSL/Format/1.0">

<fo:layout-master-set>

<fo:simple-page-master page-master-name="only">

<fo:region-body/>

</fo:simple-page-master>

</fo:layout-master-set>

<fo:page-sequence>

<fo:sequence-specification>

<fo:sequence-specifier-single page-master-name="only"/>

</fo:sequence-specification>

```

<fo:flow>

<apply-templates select="//ATOM"/>

</fo:flow>

</fo:page-sequence>

</fo:root>

</template>

<template match="ATOM">

<fo:block font-size="20pt" font-family="serif">

<value-of select="NAME"/>

</fo:block>

</template>

</stylesheet>

或许最好使用缺省的命名域来将命名域与正准备加入不同语言标记的、现有的文档中的每个元素相关联。例如，如果将某个 MathML 放在 HTML 文档中，只要将前缀加到 MathML 元素中。只需要将<html>开始标记用下面标记代替，就可以将所有的 HTML 元素放在 <http://www.w3.org/TR/REC-html40> 命名域中：

<html xmlns="http://www.w3.org/TR/REC-html40">

无需编辑文件的其余部分！插入的 MathML 标记仍需要在各自的命名域中。但是，只要这些标记不与大量的 HTML 标记相混合，就可以在 MathML 根元素上简单地声明 xmlns 特性。这样就定义用于 MathML 元素的一个缺省命名域，此元素覆盖包含 MathML 的文档的缺省命名域。如清单 18-5 所示。

清单 18-5：嵌入到使用命名域的、结构完整的 HTML 文档中的 MathML 数学元素

<?xml version="1.0"?>

<html xmlns="http://www.w3.org/TR/REC-html40">

<head>

<title>Fiat Lux</title>

<meta name="GFNFRATOR" content="amaya V1.3b" />

</head>

<body>

<P>And God said,</P>

$\text{xmlns}=\text{"http://www.w3.org/TR/REC-MathML/"}$

$\text{mrow}$

$\text{msub}$

$\text{mi}\&\#x3B4;\text{mi}$

$\text{mi}\&\#x3B1;\text{mi}$

$\text{msub}$

$\text{msup}$

$\text{mi}F\text{mi}$

$\text{mi}\&\#x3B1;\&\#x3B2;\text{mi}$

$\text{msup}$

$\text{mi}\text{mi}$

$\text{mo}=\text{mo}$

$\text{mi}\text{mi}$

$\text{mfrac}$

$\text{mrow}$

$\text{mn}4\text{mn}$

$\text{mi}\&\#x3C0;\text{mi}$

$\text{mrow}$

$\text{mi}c\text{mi}$

$\text{mfrac}$

$\text{mi}\text{mi}$

$\text{msup}$

$\text{mi}J\text{mi}$

$\text{mrow}$

$\text{mi}\&\#x3B2;\text{mi}$

$\text{mo}\text{mo}$

</mrow>

</msup>

</mrow>

</math>

<P>and there was light</P>

</body>

</html>

此处的 math、mrow、msub、mo、mi、mfrac、mn 和 msup 都在 <http://www.w3.org/TR/REC-MathML/>命名域中，尽管包含它们的文档使用 <http://www.w3.org/TR/REC-html40/> 命名域。

## 18.3 DTD 中的命名域

命名域并没有排除结构整洁和合法性的正常规则。要使带有命名域的文档合法，必须在 DTD 中声明 `xmlns` 特性，这样才能用于与这些特性相关联的元素。此外，如果文档使用 `math:subset` 元素，那么 DTD 必须声明 `math:subset` 元素，而不仅仅声明 `subset` 元素（当然，这些规定不适用于迄今讨论过的少数几个结构整洁的文档）。例如：

```
<!ELEMENT math:subset EMPTY>
```

缺省的特性值以及 `#IMPLIED` 特性在此处都有用。例如，下面的 `ATTLIST` 声明将每个 `math:subset` 元素都放在 <http://www.w3.org/TR/REC-MathML/命名域中>，除非在文档中另外指定。

```
<!ATTLIST math:subset
```

```
xmlns:math "http://www.w3.org/TR/REC-MathML/" #IMPLIED>
```

由于缺省命名域不需要在所有的元素前加上前缀，所以当处理合法的文档时，这样的命名域特别有用。给 DTD 不使用前缀的 XML 应用程序中的元素加前缀将破坏合法性。

但是，缺省命名域到底起多大作用，却有明确的范围。特别是，这些命名域不足以区分这样的两个元素：即使用的元素名相互矛盾。例如，如果一个 DTD 定义一个 `HEAD`，同时又包含一个 `TITLE` 和一个 `META` 元素，并且另一个 DTD 也定义一个 `HEAD`，同时包含 `#PCDATA`，那么就得在 DTD 和文档中使用前缀来区分这两个不同的 `HEAD` 元素。

正在进行的两种不同的开发，可能（或许不能）最终解决对来自不同领域的相互矛盾的 DTD 进行融合问题。XML 模式为 DTD 提供更加强大的替代对象；而 XML 片断能够使不同的文档与差别更大的部分结合起来。但是，这两者至今仍未完成。因此，如今，融合相互矛盾的 DTD 或许需要使用前缀来重新编写 DTD 和文档。



如果对有关使用命名域的文档是否是结构整洁或合法还不太清楚的话，请忘掉有关命名域的所有知识。只将文档作为一个正常的 XML 文档来看待，只不过这样的文档中的一些元素和特性名碰巧包含冒号罢了。这种文档也是结构整洁和合法的，就像不考虑命名域时的一样。

## 18.4 本章小结

本章解释了如何使用命名域。特别是学习了如下内容：

- 命名域区别不同 XML 应用程序中相同名称的元素和特性。
- 命名域是由 `xmlns` 特性（其值为命名域的 URI）来声明的。由该 URI 引用的文档可以不存在。
- 与命名域相关联的前缀是 `xmlns` 特性名的组成部分，而此命名域前面有一冒号；例如 `xmlns:prefix`。
- 加到所有元素和特性名中的前缀属于由前缀表征的命名域。
- 如果 `xmlns` 特性没有前缀，那么它就为元素及该元素的子元素（但不为任何特性）建立一个缺省的命名域。
- DTD 必须以这样的方式来编写，以使对命名域一无所知的处理程序仍能分析并验证此文档的合法性。

下一章将探讨资源描述框架 (Resource Description Framework, RDF)，它是个 XML 应用程序，用于编译元数据 (metadata) 和信息结构。