

无废话 XML

劳虎

插图：胭脂虎、劳虎

两只老虎工作室 www.2tigers.net

联合光纤通信公司 www.ufoc.com.tw 热情赞助

Many of the designations used by the manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the author was aware of a trademark claim, the designations have been printed in initial caps or all caps.

本书内文所提及之公司及产品名称，均为各所属公司所有。

本书范例中的程序码及所有由作者提供下载之软件，均为自由／免费软件，丝毫不附帶任何保障。所有程序虽都经作者尽心尽力准备、测试，但疏漏在所难免，若因使用而直接或间接造成资料遗失、系统毁损，一切后果由使用者自行承担，作者概不负责。

本书由作者以 PDF_AT_EX 搭配 CJK 套件在 Linux 平台上制作完成。

Copyright ©1999 by Lao Hu and Two-Tigers Workshop. All rights reserved.

本书及所有自两只老虎网站下载之软件，版权均属劳虎及两只老虎工作室所有。在标明出处、保持全书完整性、不收取任何费用、并且不作商业营利用途的前提下，可自由在 Internet 上传布（含商业广告的网站和电子报在此视为营利性质）；此外，未经作者本人书面同意之前，严禁透过 Internet 以外之任何传播媒介，包括、但不仅限于光盘、书籍、报刊、杂志等，以完整或部分形式复制、转载。

读者可依个人进修或机关、学校授课需要，自由以打印机为自己或他人做少量复印，唯复印成品不得用于商业营利用途，同时不得向使用者收取任何费用。

本书自动受国际著作权公法保护。作者保留一切法律追诉权。

谨将此书献给

虎父萧教授（麻将郎中、运动健将）

虎母萧妈妈（乡里名厨、女高音）

目 录

写在前面	xiii
0.1 写这本书的动机	xiii
0.2 读者应具备的基础	xiii
0.3 本书所使用的记号与标示法	xiv
0.4 如何善用书中的各项功能	xiv
0.5 缺憾	xvi
0.6 最后	xvi
1 介绍 XML	1
1.1 为什么要学 XML	1
1.1.1 江郎才尽的 HTML	1
1.1.2 XML 前来解救	4
1.1.3 XML 的优越性	6
1.2 为 XML 打扮	12
1.3 XYZ 专有名词大会串 — XML 应用实例	13
1.3.1 数学 ML	13
1.3.2 微笑串连	13

1.3.3	Flash 杀手 ?	14
1.3.4	XSL	17
1.3.5	有哪些相关的网站 ?	17
1.3.6	使用介面随你设	18
1.3.7	那微软呢 ?	19
1.4	先用先赢	19
2	XML 语法领进门	21
2.1	前奏	21
2.2	元素与属性	22
2.3	注解	23
2.4	不可或缺的解析器	23
2.5	XML 文件必先要「及格」	26
2.5.1	所有元素都要正确地关闭	26
2.5.2	标签之间不得交叉	27
2.5.3	所有属性都得包上引号	28
2.5.4	其他规定	28
2.6	以法为証	28
2.7	大小写有分	30
2.8	CDATA 区	30
2.9	一空两空大不同	31
2.10	PI 与样规链结	33
2.11	何去何从	34
3	Unicode 说分明	35
3.1	Unicode 简介	35

3.2	字母游戏	37
3.3	血淋淋的细节	39
3.3.1	Unicode 中的空间分配	40
3.3.2	UTF-8 的编码原理和特性	40
3.3.3	UTF-8 的优点	42
3.3.4	UTF-8 的缺点	42
3.3.5	UTF-16 中的代理对	43
3.3.6	私用区	44
4	XML 化妆术	45
4.1	CSS 入门	45
4.1.1	选择式 (selector)	45
5	名称空间	47
5.1	为什么需要名称空间	47
5.2	名称空间的长像	50
5.2.1	标示物	50
5.2.2	URL、URN、URI — 别搞迷糊了	51
5.2.3	名称空间实例	52
5.3	名称空间的宣告	53
5.3.1	前置字串	53
5.4	名称空间的范畴	55
5.5	预设的名称空间	56
5.5.1	预设空间与范畴联合运用	57
6	下一代的 HTML — XHTML	61
6.1	什么是 XHTML	61

6.2	XHTML 长得什么样子	61
6.3	为什么需要 XHTML	62
6.3.1	HTML 的隐忧	62
6.3.2	XHTML — 既可轻薄短小，又可无限延伸	64
6.4	XHTML 和 HTML 4.0 的差别	66
6.4.1	XHTML 帮您复习	66
6.4.2	格式正确原则对 HTML 的冲击	67
6.4.3	检测、验证、依据	69
6.4.4	其他规定事项	72
6.5	XHTML — 到底给谁看	74
6.6	从何下手	76
6.6.1	HTML Tidy 使用方法	76
6.7	最后	77
7	XSLT — XML 专属的转换语	79
7.1	另一种样规 — XSL 简介	79
7.1.1	XSL 和 CSS 不同的地方	80
7.1.2	XSL 和 CSS 相同的地方	80
7.1.3	XSL 简史	81
7.2	XSLT 入门	82
7.2.1	XSLT 在网络上的应用模式	82
7.2.2	XSLT 的转换流程及工作原理	84
7.2.3	支援 XSL 的软件	86
7.3	细谈 XSLT	86
7.3.1	成品预览	87
7.3.2	XSL 样规与名称空间	89

7.3.3	源树分解	90
7.3.4	根元素上头还有一级	92
7.3.5	XSLT 运行细节	92
7.3.6	{ 属性值模板 }	99
7.3.7	输出文字码设定	100
7.4	XPath 路径描述语	101
7.5	换您了一 实作演练	101
7.5.1	xt	101
7.5.2	迁就 IE5	105
8	DTD — XML 语汇的定义	107
8.1	消除对 DTD 的恐惧	107
8.1.1	成品预览	108
8.2	元素类别宣告	108
8.2.1	量子学	109
8.2.2	出现次序	109
8.3	属性类别宣告	110
8.4	统统放到一起 — 文件类别宣告	111
8.5	结语	112

关于作者

劳虎为杂食动物（也吃肉），专长为网站科技的整合运用，熟悉的项目包括 XML、Java、Perl、数据库、系统安全和电子商务。

留美语言学博士，玩网络纯属不务正业，曾在美担任网管，义务翻译 Perl FAQ，在 Run!PC 杂志里写专栏。

主修语音、文字识别的他，本行虽为自然语言，却深深被人工语言所吸引。他与妻子胭脂虎在 1996 年创立义务性的两只老虎网站，解答了世界上无数华人在网站管理和网页设计方面的疑难杂症。

写在前面

0.1 写这本书的动机

现在正值 XML 科技发展的火热阶段，世界上各大软件公司，无不卯足了劲，让自己的产品能抢搭上这班列车。但随着每个西方新科技的到来，背后都暗藏了一个隐忧，那就是：中文的网络大众，往往因语言、文字的隔阂，而无法立即享受到这项新科技所带来的便利。

XML 是一个能替未来的网络世界、乃至人类日常生活带来革命性改进的优良科技。这次，我真的很不想再看到我们在这方面又 ...

因此，我起了写这本书的念头。我不但决定写一本 XML 的书，让网络上的中文人口也能不落人后，尽早享受到它的好处；更立志要写出一本完全以中文应用为素材、不但内容正确，而且够新、能跟得上 XML 日新月异的发展脚步的书。

同样重要的是，这本书必须让所有对 XML 有志一同的网友，都能很轻易、快速地取得。要达到这个目的，这必须是一本免费下载的电子书。但免费并不表示在「质」的方面会打折扣 — 相反地，这本书以时下热门的 PDF 格式精心制作，不管是在屏幕上阅读，或以打印机印出，都能获得最佳的效果，将电子书的特性，发挥到极至。

0.2 读者应具备的基础



在学 XML 之前，建议您最好先对 HTML 标注语有基本的认识，如果曾动手写过 HTML 表格 (tables) 则更好。书中对所有用作范例的 HTML 码，均不作额外解释。

此外，本书将提到一些不属于 XML，但和 XML 息息相关的科技，包括 CSS 和 DOM。如果您已具备 CSS 的基本概念，则将会很有帮助。即使您从没用过 CSS，也不用太担心，本书将会提供一段 CSS 入门。

0.3 本书所使用的记号与标示法

- 重要的专有名词，第一次出现时皆以绿色荧光笔 (marker) 效果表示。
- 文句中要特别强调的地方一律以斜体表示。
- 范例和源代码以打字机字体 (Courier) 呈现，如：

```
<?xml version="1.0" ?>
<问候>
  <hello>您好，世界！</hello>
</问候>
```

此外，书中穿插了一些框框，同时配合上一些可爱的小符号，如：、等，用来提醒一些注意事项，或提供实用的小点子。这些框框，聪明的您到时候一看就懂，在此不需要废话，浪费篇幅多作说明，让书名蒙羞。

0.4 如何善用书中的各项功能

这本书在编排上，应用到许多 PDF 文件特有的便利功能，包括书签和超连结。

书签 PDF 中所谓的「书签」，就是当您用 PDF 阅读器 (Acrobat Reader 等) 一打开这本书的时候，在左边所看到的那个树状结构目录¹。本书的书签，在中文版的 Acrobat Reader 中，可以正确地显示。至于在其他语言的阅读器和操作系统底下，如果配合南极星一类的软件，应该也能正确地显现出来。

¹这和浏览器中的书签有所不同，也不能让使用者把自己喜欢的 PDF 页面添加到书签里。

内文交叉链结 文中相互参考的链结，一律以深绿色表示。您在电脑上直接阅读这本书的时候，可以用这些链结把您带到书中其他地方。像前面总目录中的各个章节项目、文章中的注脚，及各附表、附图的编号，均以内部链结方式呈现，方便读者查阅。

每当书中提到重要的专有名词时，如果这个名词，在书中其他地方有特别解释的话，您会在紧接着该名词之后，看到一个绿色的小链结，分两种：

章节链结 如果一个专有名词，是某章节的主题（例如关于 [Unicode^{\[3\]}](#) 的讨论），这个时候您在小链结中看到的，正是那个章节的编号。

页链结 如果专有名词不是某章节探讨的主题，但曾在某页提到过，譬如像统汉字 [\[p.36\]](#) (Unihan) 这个名词，此时在小链结中出现的是页数，而不是章节编号。

这样设计的目的，除了着眼于整体版面的美观外，更重要的是为了让把书印出来、在纸上阅读的读者，也能享受到类似线上超连结般的便利。

外部链结 除了内部链结外，所有书里提到的网址及网上的资源，都一律以深蓝色超连结方式呈现，如：[W3C XML 主页](#)，一切就像您在浏览器中浏览网页一样。这个例子同时展现了本书的另一特色——因为有鉴于长串的网址，穿插于字里行间，常会影响行文流畅，进而减低阅读速度。我决定把它们移到页面边缘的空白处，这也是为了让读者在纸上阅读时，也不会错失这些相关的网址信息。

W3C XML 主页：
<http://www.w3.org/xml/>



在 Acrobat Reader 的设定中，您可以自行指定外部浏览器，这样当您在按下书中的外部链结时，它就会去呼叫您心爱的浏览器，并自动替您连到该网址去。

0.5 缺憾

虽然我对这本电子书的编排及各项超连结的设计方面，煞费苦心，但碍于制作软件的限制，目前尚无法在 PDF 阅读器中，搜索书里的中文字。对这个缺陷，还请您多包涵，并请多利用各项超连结功能，助您快速地跳到想找的章节和内容。

0.6 最后

本着两只老虎「取之于网，用之于网」的一贯的精神，在此希望能透过这本书，为网上的中文世界，再尽一份抛砖引玉的绵薄之力。

不管您有任何批评、建议、喝采、鼓励，或错误校正，都非常欢迎您写信来给我：pailing@2Ti.com。



1 介绍 XML

XML (eXtensible Markup Language, 可延伸式标注语) 是网络科技中最闪亮的明日之星。的确, 过去一年间, 在欧美各大信息类的杂志、网站里, 这个字眼可说已经到达泛滥的地步。各大小信息产品, 无不争相和它攀关系, 抢搭这班最热门的列车。押重宝在 XML 下的公司, 可以说所有软件界的龙头全到齐了——微软、Oracle、IBM。

1.1 为什么要学 XML

1.1.1 江郎才尽的 HTML

要介绍 XML, 无可避免地得先数落一下 HTML 的缺陷。但请别误会, 我不是在鼓吹说 HTML 一无是处。相反地, HTML 对整个 WWW 这几年来的蓬勃发展、知识和信息的流通, 可说是第一功臣, 更直接带动了一波前所未有的信息革命。不管要在网络上做生意, 或要和世界网友做文件交流, 人人都得学着用 HTML 写网页, HTML 在短短几年内, 俨然已成为信息交流下, 通行最广的标准格式。尽管 HTML 在人机介面方面很拿手 (因为它正是设计来将文件内容呈现在使用者眼前的), 但是却非常不利于机器之间的互相交流、传递信息。我们来看个网络书店的实例。下面这段 HTML 码, 将书籍信息以表格 (table) 方式排列:

```
<h1>推荐丛书</h1>
<table border="1" cellpadding="5">
  <tr>
    <th>名称</th>
    <th>作者</th>
    <th>售价 (新台币) </th>
```



图 1.1: 以 HTML 表格呈现的书籍信息

```
</tr>
<tr>
  <!-- 替老婆胭脂虎的书所做的无耻宣传 ;-) ，请勿见怪 -->
  <td>煞死你的网页设计绝招</td>
  <td>胭脂虎</td>
  <td align="right">590</td>
</tr>
<tr>
  <td>如何在 7-11 白吃白喝</td>
  <td>无名氏</td>
  <td align="right">120</td>
</tr>
<tr>
  .....
</table>
```

在浏览器中呈现出来，大致会像附图 1.1 那样。

但 HTML 的问题正出在，HTML 的标签大多是设计来呈现文章的格局 (layout) 和外观的，譬如像上例中的 `<table>`、`<tr>`、`<td>`，还有像 ``、``、``、`` 等比比皆是。

今天假设我们要在网络上设立一个新兴行业 — 利用机器爬虫程序 (crawlers) 到网络上各电子商店去自动把最新的价目抓回来，让我们的客户可以藉此比价，从此再也不需要辛辛苦苦地一家家网站亲自去看。问题在，书店甲的网页可能使用类似上例的写法，也就是用 HTML 表格 (table) 来呈现，但书店乙则可能选用完全不同的形式（譬如用 ` ...` 的条列方式），而书店丙可能用的又是另一套。更严重的是，一个 HTML 网页中可能有一大票长得非常类似的标签（如：`<tr>`、`<td>`、``），有些或许是拿来标注广告看板的，还有些是拿来标导览连结的，到底哪几个才是标商品名称和价目的标签？要如何将它们可靠地萃取出来？这就有点够呛了¹。如果我们要去 check 的网站隔一断时间就拉皮翻新一次，那么事情会更棘手。我们的爬虫程序如果不是特别聪明，那根本别想对各式各样的网页去芜存菁，找出关键信息，达成任务。如果这些网页信息只是纯供人类阅读、消化，那没有问题（假设一切都排列整齐）；但在信息爆炸的网络世界，人类需要逐渐仰赖机器来替我们多分忧解劳，以 HTML 码来标注的信息，对机器和编程的人来，不但太残忍了点，处理起来更是极度缺乏效率。

HTML 对布局、外观方面很擅长，却极度缺乏对内容，也就是信息涵意的表达能力。除了少数几个用来表达内容或文义的标签，如 `<p>`、`<address>`、`<title>`，`` 外，几乎全都是用来设计网页格局的了。所以前面才说，HTML 是设计来做人机交流用的。对上面所碰到的难题，我们最需要的，是一个能将商品价格明确标示出来的机制，譬如说，一个 `<price>` 标签。

¹如果您不太能想像的话，建议您到一个电子商务网站去逛一逛，先在网页中选定一样商品，然后在浏览器中选择显示 HTML 源代码，看能不能很轻易地把该项商品在源代码中的「方位」简单明了地形容出来 ;-)。

「那是不是就从改进、修订 HTML 这方面来着手，适时按需要多加进一些新标签？」

不行。HTML 这几年来在两大浏览器鏖战的结果下，早已过度膨胀，肥得像个怪物了。而且，我们既不是微软，又不是网景，或 W3C，光是我们说希望为线上购物制订一个 `<price>` 的新标签，难道人家就会甩我们吗？更何况，各行各业需要用到各式各样的标签，如果统统都加进 HTML 里，那还得了！？

其实线上商店里的信息，在还没有制作成 HTML 网页之前，可能都是一笔笔按照栏位储存在数据库的 `tables` 或物件里的（稍具规模的电子商务网站都是这么做），商品名称、代号、价格...等，各自存在所属的栏位、项目中，分得一清二楚；但一旦从数据库中被调出来，再经 CGI/ASP/Cold Fusion/PHP/... scripts 搞一搞，转成 HTML 格式后，这些重要的栏位、项目名称全变成了毫无特定意义的 `<h2>`、`<h3>` 或 `<th>`、`<td>`...。换句话说，原本建立在数据库中非常宝贵的信息架构 (schema)，在商品信息转换为 HTML、提供给客户的同时，荡然无存。试想：今天我们如果能将数据库中的资料（如：商品形录或订单），保留原本完整的资料架构，一五一十地在自己和客户的电脑之间互传，必定大大有利双方把资料快速地建入自己的数据库内，进而大幅提升商务效率。这如果用 HTML 来做，恐怕很困难。

当今的网络世界，电子商务逐渐兴盛，异质电脑系统之间的互动日益频繁（商务信息互传），自动化程序角色也愈来愈重要（譬如前述的爬虫程序），我们需要一个比目前 HTML 更好的方法，否则未来网络的发展将会愈走愈慢、愈累。欢迎来到 XML 的天地！

1.1.2 XML 前来解救

XML 和 HTML 的一大不同处，就在于在 XML 里，我们可以自由定义标签。定义出来的标签，可以按自己的意思充分地表达文件的内容，譬如我们可以定义 `<name>`、`<book_info>` 这样充分达意的标签。在 XML 中，我们只注重内容，这和 HTML 强调

布局的作法大不相同。至于 XML 文件的外观呈现，可透过搭配 CSS^[4.1] 或使用 XSLT^[7.2] 来做 XML → HTML 或其他格式的变形。这点下面会有进一步的说明。

XML 让已经会使用 HTML 的人轻松上手，因为它的写法和 HTML 类似，把标签用 < 和 > 符号括起来。更酷的是，请看实例：

```
<?xml version="1.0" encoding="GB2312" ?>

<推荐丛书>
  <书籍>
    <!-- 替老婆胭脂虎的书所做的无耻宣传 ;-) ，请勿见怪 -->
    <名称>煞死你的网页设计绝招</名称>
    <作者>胭脂虎</作者>
    <售价 货币单位="新台币">590</售价>
  </书籍>
  <书籍>
    <名称>如何在 7-11 白吃白喝</名称>
    <作者>无名氏</作者>
    <售价 货币单位="新台币">120</售价>
  </书籍>
</推荐丛书>
```

「哇！竟连标签都是中文的哩，呵！」

是的，这也是 XML 的一大特性—世界通。XML 在设计之初便考虑到国际化的问题，因此打从一开始便建构在 Unicode 统一码^[3]之上，对身为中文使用者的您我来说，真的是非常大的福音。

XML 的名称常造成误导。虽然它的名字让人感觉它和 HTML 是平行的，像是为某项特定的应用（如呈现网页）所设计；但事实上，它是一套无限延伸、用来设计各式各样标注语的准则，也就是常说的 meta-language（超语^①、形而上语^②）。换句话说，XML 的层级比较高，它是用来「设计语^③的语^④」，应用范围比狭隘的 HTML 广多了，可随着我们的想像空间无限延伸。

有了 XML 以后，前述的自动化比价程序，突然间变得很容易实现了。我们的机器爬虫程序现在只要到线上书店的 XML 网页中去找 <书籍> ... </书籍> 区块里的 <名称> 和 <售价>，就可以顺利达成任务，将各家书店目前所陈列的书籍和售价一一抓回，再

也不会迷失在一堆不具特定意义的 `<p>`、`<td>`、`` ... 里了。不仅如此，信息 XML 化之后，我们可以设计出更精准的搜索引擎，譬如叫它去找所有 XML 网页里，在 `<品名>` 这个标签里含有「MP3 随身听」这个字眼，而且同商品的 `<价目>` 标签中所出现的数字必须低于台币三千元。这在今日已经不是痴人说梦和天方夜谭，几个月前推出的 Oracle8i 数据库，配合上他们的搜索引擎元件 ConText，便已提供了这样的功能。其他的 database 业者，如 IBM 和 Informix 等，也都在积极添加对 XML 的支援。此外，各大 database 业者还纷纷致力于让输出的资料可以轻易地转成 XML，新资料更可直接以 XML 形式建入。

1.1.3 XML 的优越性

归纳上面的讨论，您会发现，XML 具备几项革命性的特质，有助于大幅改善现今的电脑、网络世界。以下针对 XML 最重要的几项特性详细说明。

异质系统间的信息互通

从商业的角度来看，这点可说是 XML 最大的贡献。当今，不要说不同的企业之间，就连许多企业内部各个部门之间，都存在许多不同的系统。大到数百万美元的 mainframe 老巨兽（所谓的“legacy systems”），小到掌中型记事簿电脑，系统与系统之间，往往因大相迳庭的平台、数据库软件等，造成信息流通的困难。在这些异质系统之间做信息交流，往往需要使用特殊的软件，才能顺利地跨越彼此的门槛、疆界，非常麻烦。如果哪个系统哪天换装新的软件，很可能又会牵一发而动全身，造成一阵大忙乱。

有了 XML 后，异质系统之间，可以很方便地透过 XML 来作交流媒介。XML 格式简单易读，对于各类型资料，举凡物件、文章、RDBMS 里的资料、图形 ...，不论文字档或二元档，都能标注。要作信息交流的各大小系统上只需要装有 XML 解析器^[2.4]，便可解读由别台机器所传来的信息，进而加以利用。XML 解析器取得容易，有很多优

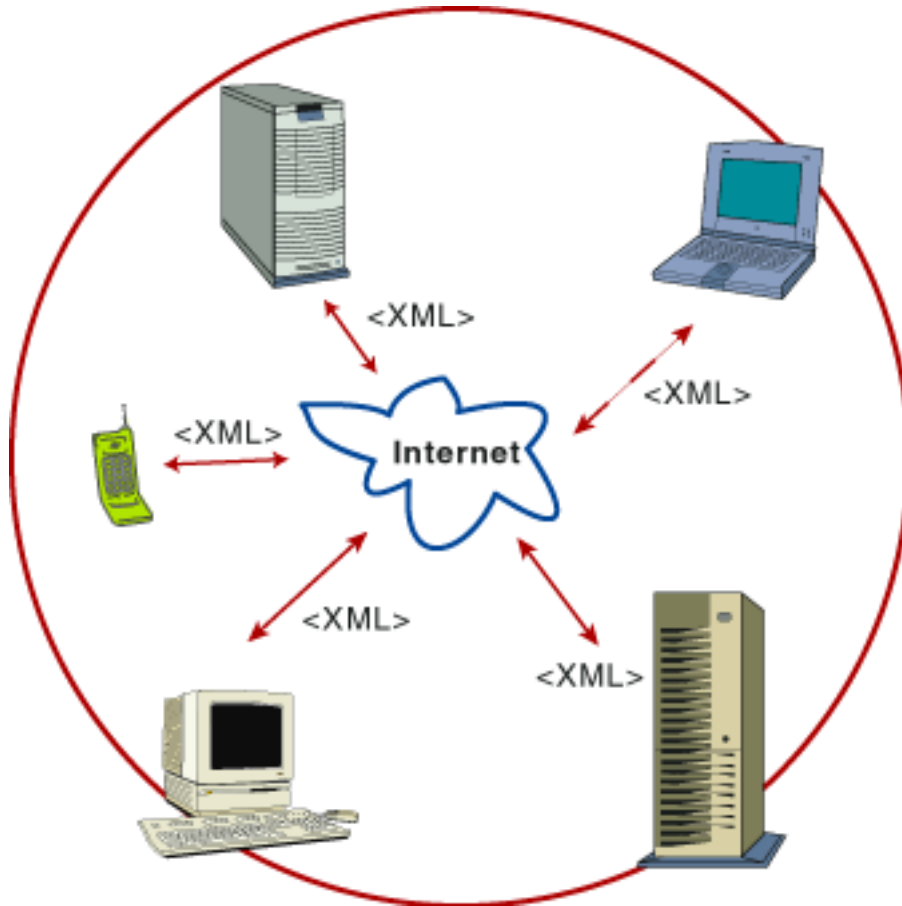


图 1.2: XML 是非常理想的网际语言，方便各式各样网络器具间的信息交流

秀的软件供人免费下载；而因为大多数的解析器以 Java 写成，更使得有 Java 虚拟机（解译器）支援的平台，大到 IBM 的 AS/400、AS/390 mainframe 级电脑，小到 Palm Pilot 掌中型记事本（还有这两个极端之间几乎所有的平台），都立即成为支援 XML 的平台。异质系统之间，不用再担心看不懂对方的资料格式，商务往来的公司之间，用不着、也不需要知道对方内部是采用何种格式储存资料，大家都用 XML 来作中介格式即可。如此一来，某个系统内部的变更，并不会殃及和它交流往来的其他系统，因为 XML 提供了一层理想的缓冲。

XML 这项利于资料的交换和传递的特性，让很多人认为，XML 将为电子商务，尤其是所谓 b2b (business to business) 公司对公司的商务，带来革命性的冲击。譬如，一个 vertical market 可以公定一套 XML 标签（在 XML 界的说法叫「语汇」；vocabulary），颁布遵行。这样上下游工业、或生意夥伴间可以很方便地透过 XML 来下单、采买，或传递设计图、产品明细表等。

EDI (Electronic Data Interchange) 是行之有年的电子商业文书交换标准。实践 EDI 的公司，可以相互以电子形式交换订单、发票，节省大量文书往返的时间和纸张的消耗。北美地区实行的 EDI 标准叫 X12，世界其他地区则多遵行联合国的 EDIFACT 标准。多年以来，EDI 并没有大量推广开来，因为 EDI 需要使用昂贵的软件、聘请专业顾问、租用专属的网络，故一般只有大型企业 (Fortune 1000 级的) 才玩得起。XML 的出现，为负担不起 EDI 的中小企业，提供了一道新曙光。XML 先天上即非常适合用来作 EDI 一类的应用，可达到同样的功能；更重要的是，XML 软件取得方便，轻薄短小，大小机器上都能跑，又可直接利用价廉、普及的 Internet 线路来传送（图 1.2）。难怪不少公司已开始积极研发以 XML 为基础的新一代 EDI 规格。

再举个例子：医疗健保体系，是最早被用来解说 XML 优越性的例子。使用 XML 来储存、简化病历后，病人的病例可以在各医院、诊所间的大小电脑间快速游走，因为 XML 格式简单（不像 HTML，有时看了真会吐血），各医疗系统的工程师可以很轻易地编出处理的程序，存取其他系统所传来的 XML 资料。某病人的病历甚至可以用 XML 储

存在她个人的 smart card 上头，随身携带。此外，像政府公文简化、统一化等，XML 都是最佳候选人。

XML 是公订、开放的标准，人人都可以拿它来开发软件，不怕被软件公司把持、要挟，不会有贪得无厌的公司，动不动藉着修改只有他们自己清楚的格式，来逼人花钱升级的情形出现。这正是公订标准可贵之处，就如同今天蓬勃发展的 Internet 是建构在各项 TCP/IP 公开标准一样。这也是为什么人人都应大力支援公订标准的原因。

「有些专门设计来做连网系统间传递讯息的科技，譬如 CORBA-IIOP、Java RMI，和微软的 DCOM 等，这些科技难道不如 XML 来得胜任吗？」

首先，XML 和 CORBA、DCOM 这些科技并不相冲突；XML 可以为它们做传递讯息、物件的桥梁，像 XML-RPC 和微软刚宣布的 SOAP 草案就是最好的例子。再者，以二元档格式传递讯息可能要比 XML 有效率，但它们在使用简便方面却远不如 XML。XML 码都是纯文字档，而非二元档，肉眼可轻易阅读，更可用编辑器直接编辑。纯文字的特性，也让 XML 文件可直接透过 HTTP 或 SMTP 等现成的通信协定来传递，无须另行编码，或作物件序化 (serialization)²、反序化处理。此外，XML 简单易学，对已经熟悉 HTML 的人更是容易上手，没有陡峭的学习曲线。因此和其他科技相比，它的优点大大弥补了处理起来较慢的缺点。更何况，在莫尔定律³历久不衰的今日，电脑硬件的威力持续飞速成长，资料处理的速度在未来将更不是问题。

XML-RPC :
<http://www.xmlrpc.com/spec>
SOAP :
<http://msdn.microsoft.com/workshop/xml/general/soaptemplate.asp>

保值

SGML 是一套有十几年历史的国际标准。HTML 便是一项 SGML 的应用实例。此外，Adobe 的文书排版工具 FrameMaker，所用的内部格式正是 SGML。SGML 当初设计的一大目标是保值——它是设计来提供文件 50 年以上的寿命的。多年以来，因为 SGML 太过庞杂，所以一直没有在世界上大量流行起来，一般只有大型企业、或政府机

²XML 文件本身已算是序化了。

³由英特尔 (Intel) 公司创办人莫尔所提出，他预测科技的进展速度，将使一个晶片中所能容纳的半导体总数，每年（后来修正为每两年）倍增一次。

构在使用。例如美国的国税局 (IRS) 便以 SGML 来设计税表等文件。

正因 SGML 过份复杂、难懂，且较不适用于网络，所以有识之士特别设计了 XML。也就是说，XML 是 SGML 精简之后的网络版，而 SGML 保值的特性，在 XML 中并未消失。何谓保值？试想：不要说过去用 WordStar 1.0 格式写成的文件，就算用今天最新的微软 Word2000 写出来的 .doc 档案，谁敢保证 50 年后的电脑上，还找得到打开这些档案的应用程序？50 年后的电脑上能不能跑 Word2000、WordStar 这些早已过时的程序更成问题；而能跑这些程序的老电脑，更可能早已进垃圾堆，恐怕要到博物馆才能借得到这样的古董；如果要随着软件的更新和升级，不断将大量文件配合转换、升级到最新的格式，则又非常没有效率。新版的文书编辑软件，对旧版的文件中的样式，更不见得都能一五一十地忠实重现。

XML 文件不但没有这些问题，在未来的世界，要从 XML 文件转换成其他的格式，更是轻而易举。仔细想想，需要保值的文件还真不少，举凡政府命令、公文、法律文件、史料、医疗记录、科学研究报告，甚至于个人的日记、回忆录等，都需要保存很长的时间，供后世子孙参考。在文书大量电子化的 21 世纪，要如何长久保存这些文件和记录，必将是一大课题。而 SGML 和 XML 的设计，正是 20 世纪末的科技先知对此一课题的答覆。

自动化 user agent 不再是奢望

前面第 3 页中所提到的机器爬虫程序，是自动化 user agent（使用者代表）程序（以下简称机器人）的一个典型的例子。网上的各大搜索引擎站，靠的正是这些小程序，每天不断将成千上万的网页内容抓回。几年前就有人期待，未来的网络世界美景，将充满这样的情节：

1. 使用者最近要出一趟远门，需要订机票。
2. 这位使用者于是把她电脑上的机器人叫出来，把心目中理想的航空公司，及飞行

的起点、终点机场等设定好后，让程序去网络上的旅行社站台把最新的报价抓回来。

3. 报价一一抓到齐了以后，机器人向该使用者回报，告知哪一家价格最便宜，提供作为订机票的重要参考。
4. 使用者根据机器人所搜集到的情报，透过浏览器到最便宜的那家旅行社的站台订位子，甚至将订位子的大任直接赋予机器人。大功告成！

但是像这样美好的远景，至少到目前都还没有实现。部分原因，正是前面所说的，HTML 文件解读不易。这里所谓的解读，当然不是指看得懂那些版面、布局的命令，这个工作浏览器已经做得相当好了。难是难在，软件要具备解读文意的能力。

XML 让实现 **smart agent** 的梦想，朝实现之日迈进一大步。有了 XML 以后，我们可以自行设计达意的标签，如：<价格>、<商品名>、<日期>，**user agent** 程序可以透过这些标签，很快地找到要找的信息，而不会迷失在一片「HTML 汪洋」中了。

更精准的搜索

这个特性和上一个（自动化 **user agent**）可说是相辅相成。XML 标签涵意丰富，明白提示所标注的内容，让搜索引擎藉由标签和内容之间的依存关系，准确地定位、找到目标，达成任务。举个例子：我个人偏好那种打起来很响的键盘，用起来让周遭的人感觉我正在起劲地工作（但吵得他们难以工作;-）。我把我的 **agent** 程序送到网上去找有关电脑键盘的信息，好作为我购买前的参考。但问题是，「键盘」两字太过笼统，可能出现在电脑零件的网页中（正是我想找的），如：

<电脑组件>

.....

<周边> 林口牌超耐键盘，PS2 介面 </周边>

....

<周边>高级双键滑鼠，附视窗型滚轮，USB 介面</周边>

</电脑组件>

但它们也可能出现于某合唱团的介绍中：

```
<合唱团>
  <团名>哇Zoo!</团名>
  <成员>
    <团长>黑虎（主唱、电贝司）</团长>
    <团员>鳄神（电吉他）</团员>
    <团员>熊哥（键盘）</团员>
    .....
  </成员>
</合唱团>
```

甚至还可能出现于其他类型的网页，譬如像卖钢琴或电子琴的产品说明页。

有了 XML，我只要告诉我的 agent 程序，把目标锁定在寄居于「电脑」、「周边」这些项目中的「键盘」二字即可。爬虫、agent 设计起来不再像在 HTML 时代那么痛苦，而且搜索起来在效率、准确性上都要高多了！

1.2 为 XML 打扮

在数据库之间以 XML 形式传递的资料，或许不需要特别妆点，但是如果一份 XML 文件要呈现在使用者面前，或者要付印时，我们可以透过 CSS^[4.1] 或 XSL^[7.1] style sheets（本书翻作「样规」）来设定外观。CSS 最早是为 HTML 所设计的，但拿到 XML 上一样好用。CSS 的第二代 — CSS Level2 在发展时已经将 XML 的应用涵盖进去了。

配合样规，我们对 XML 文件中各个标签里的文字内容的外观，包括字体、大小、颜色、排列方式、位置、层面等，都可以一一指定。更好的是，同样的 XML 文件，可依不同场合为它设计出多套样规。譬如一个网络书店的图书资料，很可能既需要放在网页中供人浏览，又需要把它打印在买卖的发票里，随书寄给顾客。这个时候，只要设计两份样规就可以达到这个目的，请看附图 1.3。

重要的是，文件的内容和外观设计是完全分开的。外观（样规）变动时，XML 文件完全不受影响。



图 1.3: 同样一份内容，配上不同的样规，生出截然不同的造形

1.3 XYZ 专有名词大会串 — XML 应用实例

下面各节将介绍的应用实例，全都是直接建置于 XML 之上的语汇，充分证明了 XML 无远弗界的弹性。

1.3.1 数学 ML

MathML 藉着 XML 定义出一套能充分表达数学式子的标注语汇。它已经在不久前正式成为 W3C 的推荐标准 (recommendation)。随着 XML 时代的来临，数学家们将不再遗憾 HTML 欠缺对数学符号、公式的支援了。

MathML :
<http://www.w3.org/TR/REC-MathML>

1.3.2 微笑串连

SMIL 是「多媒介同步整合语汇」(Synchronized Multimedia Integration Language) 的缩写，读作“smile” :-)，也是一项 XML 的应用。广受欢迎的 RealPlayer G2，便采用 SMIL 来提供影 (Real Video)、音 (Real Audio)、图档 (Real Pix)、文字 (Real Text) 串连的功能，让设计者可以自由选择多媒介出现的时间和先后顺序。

SMIL :
<http://www.w3.org/TR/REC-smil>

和 MathML 一样，SMIL 业已成为 W3C 正式推荐的网络标准。

1.3.3 Flash 杀手？

SVG :
[http://www.w3.org/
TR/SVG](http://www.w3.org/TR/SVG)

SVG 是 Scalable Vector Graphics 的缩写，它是由 W3C 所研发、架构在 XML 基础上的向量图形格式，发展成员包括了 Adobe、HP、IBM、Macromedia、微软、Netscape、Quark、Sun 等大公司，目前已经达最后阶段，眼看即将成为正式标准。

过去在 XML 向量图形的发展上，有两派人马，互相较劲，一派以 Adobe 为首，提倡 PGML，另一派由微软、Macromedia 领军，鼓吹 VML（IE5 有支援）。在这两套提案呈递给 W3C 后，W3C 决定协调各路人马，将两套互别苗头的格式融合在一起，促成了 SVG 的诞生。

我们知道，目前网络上最广为通行的图档格式 — GIF、JPEG 和 PNG⁴，都是属于点阵性的格式，而点阵图有尺寸较大，不利缩放的缺点。Macromedia Flash 所提供的向量动画功能，适时地趁虚而入，填补了一部分这方面的空白，成功地打响了这个产品。那既然已经有 Flash，而且 Macromedia 不久前也已将 Flash 格式透明化、供人自由发展，为什么还要再搞出一个新的东西呢？

首先，最重要的是，SVG 建置于纯文字格式的 XML 之上，直接继承了前面提到的 XML 特性，简化异质系统间的信息交流，方便数据库的存取；而且在未来，可直接融入 XML 和 XHTML^[6] 网页中，更可以直接利用其他既有的浏览器端科技，如 CSS^[4,1]、DOM（文件物件模型；Document Object Model），和 ECMAScript（JavaScript 升格成 ECMA〔欧洲标准协会〕正式标准后的新名子），达到动画及 DHTML 般的动态效果⁵。SVG 也支援点阵图档汇入，还支援目前仍在发展阶段的两个 XML 连结语汇 — XLink 和 XPointer，不但可以作 HTML 式的单向连结，更提供多向连结，和许多复杂的花样。还有，SVG 是公订的网络标准，不受单一的公司操纵。

DOM :
[http://www.w3.org/
DOM](http://www.w3.org/DOM)

XLink :
[http://www.w3.org/
xlink](http://www.w3.org/xlink)

XPointer :
[http://www.w3.org/
xptr](http://www.w3.org/xptr)

上面多项 SVG 的优点却正是 Flash 的缺点：Flash 必须要倚靠浏览器外挂程序（插

⁴PNG（读作“ping”）是设计来取代 GIF 的点阵图档格式。不但比 GIF 更为优秀，最重要的是，它是一个公定标准，没有专利权的纠缠。目前支援 PNG 的浏览器和制图工具已有不少，而且不断在增加中。

⁵所谓的 DHTML，正是透过 CSS、DOM 和 JavaScript 所营造出来的动态效果的泛称。

件；plug-ins）来播放，而且因为 Flash 的格式为二元档，作品内的文字内容无法让使用者在浏览器中作字符串搜索（的确有人把文章做进 Flash 电影里！），这样的网页，也无法让搜索引擎索引、登录其中的文字，供访客作全文检索。此外，高互动性的多媒介动画，往往需要靠程序来帮忙（因此 Director 才有 Lingo 语言），这点是 Flash 另一个较为不利的地方：Flash 和 JavaScript 之间的互动，只能透过比较狭窄的 FSCCommand 来作桥梁；虽然 Flash 从第四版也开始加入一些简单的程序设计功能，但仍不够完备，和发展已趋成熟的 ECMAScript 相比还差上一截，更别说 ECMAScript 早已有广大的使用人口。

哇，说了这么多 Flash 的坏话，我老婆胭脂虎的头上已经开始冒烟，看来快变成「烟之虎」了！赶快在此补充一点：尽管 Flash 有上述这些缺憾，但至少在今日，仍旧是网页向量动画最好的解决方案。SVG 的远景虽美好，但目前尚仍停留在明日之星的地位，最后下场会怎样，还很难断定。

我们来看看 SVG 码究竟长得什么样（直接取材自 SVG 标准文件）：

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.dtd">
<svg width="4in" height="3in">
  <desc>以下的 SVG 码会画出一个蓝色的圆，红色的外框</desc>
  <g>
    <circle style="fill: blue; stroke: red"
      cx="200" cy="200" r="100"/>
  </g>
</svg>
```

因为 SVG 太新了，目前几个主要的浏览器，包括 Netscape、IE 和 Opera，都还没有支援。不过已经有 SVG 的浏览器，像 IBM 的 SVGView（见图 1.4）。

因为 SVG 架构在 XML 上，而 IE5 和 Netscape5 都已内建 XML 解析器，充分支援 XML，而且又都支援其他 SVG 可以直接仰仗的科技，包括 CSS、DOM 和 ECMAScript，所以未来的浏览器有可能加入对 SVG 的支援。尤其是 Netscape，因为它已经改采 Open Source 程序码公开的模式来发展（Mozilla 计画），任何热心人

IBM 的 SVGView：
<http://alphaworks.ibm.com/tech/svgview>

Mozilla 计画：
<http://www.mozilla.org>

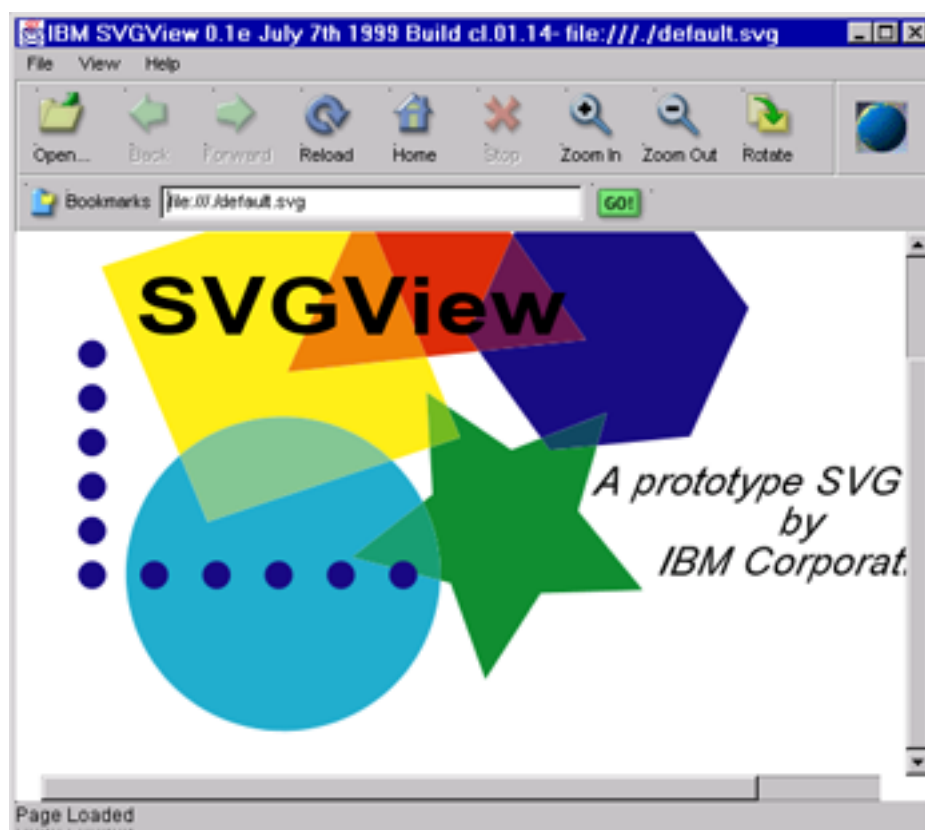


图 1.4: IBM 的 SVG 浏览器

士都可以主动贡献，SVG 的支援随时有可能加进去。

要能让 SVG 大量推广开来，关键在要有方便好用的编辑工具出现。由于 SVG 是公订标准，主要在幕后称腰的又是 Adobe，而他们在 WWW 软件市场的死敌偏偏又是 Macromedia，加上 XML 热潮所挟带的优势，从这几方面研判，SVG 在未来不是没有潜力取代 Flash，成为网络上的头号向量图格式。如果到时候，「Flash 杀手」效应真的发生，别忘了，您是先在这里看到的哦 ;-)！

1.3.4 XSL

CSS^[4.1] 之外，另一种替 XML 打扮外观的科技是 XSL^[7.1]。XSL 所使用的方法和 CSS 大异其趣，我们在 7.1 节会详细介绍。

1.3.5 有哪些相关的网站？

Netscape 浏览器 4.06 及 4.5 之后的版本，内建了一项 “What’s Related” — 同类型网站推荐服务。其幕后的一大功臣，除了数据库及搜索引擎外，应首推 RDF（资源定义架构；Resource Definition Framework）了。

RDF :
<http://www.w3.org/>
RDF

先简单谈一下 “What’s Related” 的功能是怎么做出来的：每当使用者按下这个按钮时，浏览器便将使用者正在浏览的网页网址，传送到 Netscape 公司的 server 上，server 在它的数据库中找到这个网页，并且将所有和这个网页关系最密切的其他网址及相关资料，以 RDF 格式传回给浏览器，最后以菜单方式呈现在使用者面前，供使用者选择前往。

不用说，RDF 是另一个重要的 XML 语汇。它演化自 HTML 中藉着 <meta> 标签来描述网络资源的做法。而所谓的「形而上信息」(meta info; meta data)，用一句话简单解释⁶，就是 “data about data”。譬如图书馆的书目和作者索引，不管是较进步的电脑数位式，或是传统的「卡片、小抽屉」式，正是典型的形而上信息。RDF 设计的目

⁶借用 RDF 标准中的说法。

的，正是用来描述形而上信息，不但要让机器能读，更要让它能读得懂（这个重要的区别，在前面讨论 HTML 和 XML 的优缺点时，已经强调过）。RDF 着重于机器与机器之间的自动化交流，可以用来描述网络资源，方便搜寻，还能表达资源与资源间的相互关系。

「那 Netscape 从哪得来这么多网页、网站关连性的资料呢？」

这是由 Alexa 这家搜索引擎公司提供的。Alexa 的创办人之一，正是过去名震一时的 WAIS 搜索引擎的发明人。它的搜索服务是个不落俗套、很有创意的点子：它们不接受网站来主动报名登记，运作上完全仰赖机器爬虫大量抓回来的网页，再对其中的超连结，作网站关连性的分析，而“*What's Related*”使用者的选择，则作为修正关连系数的重要参考。

反观其他绝大多数的搜索引擎站，如 AltaVista，因为采用自由登记制，而且将网页中 `<meta>` 标签所列的关键字作为搜索结果排列顺序的重要参考，已经导致一些不肖网站业者，尤其是色情和诈财性质的站台，为了吸引更多的访客，滥用关键字的设计，严重扭曲搜索结果，无形中也降低了搜寻站的实用价值。

1.3.6 使用介面随你设

呼之欲出的下一代 Netscape5 浏览器 (Mozilla)，从里到外，整个脱了一层皮。除了 100% 支援 W3C 的 HTML4.0、XML、CSS1、DOM1 等外，对 CSS2、DOM2⁷ 也有一些支援。Mozilla 的使用介面，也是一大创新，不但完全跨平台，而且可以让人自由设定按钮、菜单、命令等元件。用来设定、控制使用介面的语言 — XUL，又是一项 XML 的应用。也就是说，熟悉 XML 的人，可以充分地用 XUL 和 CSS 来改变 Mozilla 的使用介面，浏览器的外观也因而变得像系统桌面 (themes) 一样，让有设计巧思的人充分发挥想像的空间。就像热门的 MP3 播放器 Sonique 和 WinAmp 所支援的 skin 功能一样，在未来，每当我们看腻了浏览器的介面时，便可到网络上下载一套更酷的皮肤

⁷仍在草案阶段，即将成为正式标准。

HTML4.0:
[http://www.w3.org/
TR/REC-html40](http://www.w3.org/TR/REC-html40)
XUL:
[http://www.
mozilla.org/
xpfe/xp toolkit/
xulintro.html](http://www.mozilla.org/xpfe/xp toolkit/xulintro.html)

换上去。

1.3.7 那微软呢？

微软很早就开始使用 XML。事实上，IE4 浏览器所内建的 CDF (Channel Definition Format) 功能，提供频道订阅、push 服务，堪称为最早的 XML 应用实例。IE5 浏览器对 XML、XSL^[7.1]、和 DOM，也都有相当程度的支援。此外，刚出炉的 Office 2000，在储存成 HTML 格式时，利用 `<xml> ... </xml>` 区块（微软的术语叫「XML 岛」），来达到双向 (round-trip) 的格式转换；也就是过去发表成 HTML 网页的 Word、Excel、PowerPoint 等文件，可以重新读入 Office 软件中，忠实地重现为原来的 .doc、.xls，和 .ppt 格式。

微软未来的电子商务策略中，XML 将是重头戏。他们目前正将下一代的 COM (Component Object Model) 介面开放，让其他系统可以透过 XML 来与之沟通。未来支援 XML 的服务器软件，更包括了 Windows2000、SQL Server、BizTalk server，和 Commerce server 等。

1.4 先用先赢

XML 不是明日的科技，您现在就可以充分享受它所带来的好处。多项重要的 XML 相关科技及标准，都已发展成熟。各类应用软件更如雨后春笋般，不断快速地增加。



2 XML 语法领进门

我们继续延用第 1 章「推荐丛书」的例子（附表 2.1）。您可以利用其中的色块，直接跳到说明该项语法的章节去一看究竟。

```
<?xml version="1.0" encoding="GB2312" ?> A
<?xml-stylesheet href="style.css" type="text/css" ?> B
<推荐丛书>
  <书籍>
    <!-- 替老婆胭脂虎的书所做的无耻宣传 ;-) ，请勿见怪 --> C
    <名称>煞死你的网页设计绝招</名称>
    <作者>胭脂虎</作者>
    <售价 货币单位="新台币">590</售价> D
  </书籍>
  <书籍>
    <名称>如何在 7-11 白吃白喝</名称> E
    <作者>无名氏</作者>
    <售价 货币单位="新台币">120</售价>
  </书籍>
</推荐丛书>
```

表 2.1: 中文 XML 实例

2.1 前奏

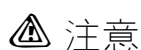
在表 2.1 里的第一行叙述中，我们看到：

A

```
<?xml version="1.0" encoding="GB2312" ?>
```

这叫 **XML 宣告** (declaration)，更有学问的称呼为 **前记** (prolog)。其中 `version` 这个注明版本的属性肯定要有，而 `encoding` 这个注明文字编码的属性则可有可无，如果省略的话，字码必须是 **Unicode**^[3]，以 **UTF-8** 或 **UTF-16**^[3.2] 作编码。在这样的情况下，甚至可以将整行 `<?xml ... ?>` 宣告一并省去，不过 XML 标准中强烈建议不要这么做，不管用的是不是 **Unicode**，最好还是养成一律写的好习惯。

在我们的例子中，则明确地注明使用的字码是 **Big5**。在 XML 文件的编码不是 **UTF-8** 或 **UTF-16** 的情形下，`<?xml ... ?>` 宣告绝不可省，而 `encoding` 也绝不可少。



不是所有 XML 软件都支援 **Big5/GB2312**。XML 标准只强制规定所有软件必须支援 **UTF-8** 和 **UTF-16**^[3] 这两种编码，至于对其他国家、地区性编码的支援，则由各软件发展者自行决定。因此，不见得所有的 XML 软件都能正确处理 `encoding` 标示为 **Big5/GB2312** 的文件。至于为什么要这样规定，我们在第 3 章讨论 **Unicode** 时会探讨这个问题。

2.2 元素与属性

D

俗称的「标签」(tags)，实际上包含了「元素」(elements) 和「属性」(attributes) 两部分。例如在表 2.1 中，「作者」是直属于「推荐丛书」这个母元素底下的子元素。而「货币单位="..."」则是「售价」这个元素的一个属性；我们把「货币单位」称作「属性名」，等号后面的值则称作「属性值」。最高层的元素（推荐丛书）称作「根元素」(root element)。

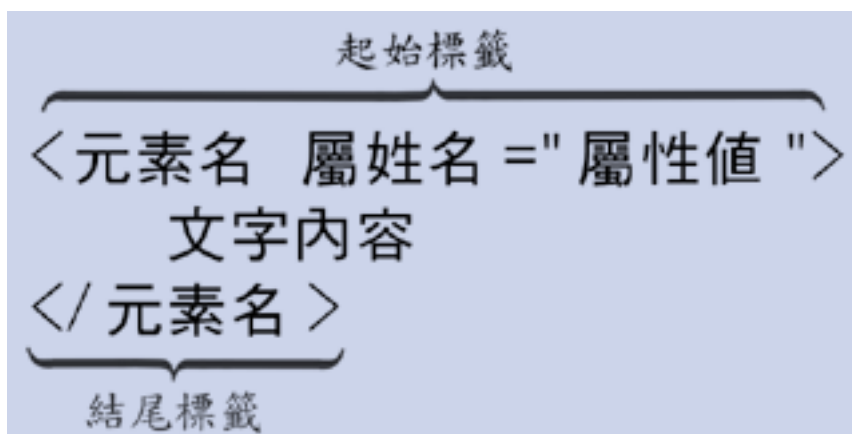


图 2.1: XML 元素与属性

2.3 注解

在 XML 中，注解语法和 HTML 中非常相似。虽然严格来讲，不完全一样，但二者的差别在这里并不重要。一如 HTML，注解是放在 `<!--` 和 `-->` 之间的区块，如附表 2.1 所示。

C

⚠ 注意

附表 2.1 的 XML 码中，用来界定元素的是 ASCII 中的小于和大于符号（`<` 和 `>`；半形）。因此在编辑器中敲入中文的元素名时，要注意不要因疏忽而误输入了中文字码中的全形符号，像“`<`”、“`>`”、“`<`”、“`>`”等。

2.4 不可或缺的解析器

在解释 XML 严格的语法规则之前，让我们先来谈谈 parsing 这个重要的概念。“parse”就是解析的意思，身为万物之灵的人类，我们的大脑每天都在做大量的

parsing 动作，卻往往浑然不自知 — 像您现在就正在做 parsing — 阅读这本书。您之所以看得懂我写的文章（希望如此;-），而且能听懂周遭的人讲的话，正是因为您的脑中有一台非常厉害的语讠解析器，英文叫“parser”，随时在高速解读文章和话语，从断字、断句一直到语法、语意分析；这个解析器的复杂度和精确度是任何电脑语音输入软件所望尘莫及的。假设在大街上，有一个老外向您迎面走来，开口道：

「阿度仔名我叫，哪里 bus 可以我搭？」

您心想：「哇！这是哪国的语法，也不知道这个老外是在哪里学中文的，不过八成在学校是被当的！」

为什么我们一听到那个老外的话，会马上有这种反应？这就是因为我们大脑中的语讠解析器，也就是 parser，在举牌抗议，让我们知道这个句子怪里怪气的。

解析器是语讠处理的最前线。譬如我从来沒学过泰国话，脑中因而欠缺泰语的解析器，当面对一份泰语文件时，我连最基本的字、词、句子结构都搞不清楚，更甬提让我进一步翻译、或修改这篇文章了。

同样地，软件也需要內建语讠解析器，才能处理某种语讠。XML 是一种语讠，在我们能进一步利用文件的内容之前，肯定要先用解析器把文件分析成一个个物件才行。还有，像网页浏览器，必定具备一个 HTML 解析器，这样它才能读得懂各网站里的 HTML 档，进而将网页布局呈现在使用者面前。如果它读到太夸张离谱的 HTML 文件，可能就无法按原 HTML 作者所希望表达的方式，把文章顺利地被浏览者面前呈现出来，甚至可能一片空白，什么都出不来。换句话说，浏览器內建的解析器斗不过乱七八糟的网页，只好认输、放弃了。

XML 的设计者有鉴于目前网络上充斥了大量源代码「不纯净」的网页（多数 HTML 档的写法和格式严格讲起来都不合格），因此决定这次要从一开始就严格执行，规定所有的 XML 文件都必须遵守几个基本规定（下一节的主题），否则一律不留情，统统赏予那可怕的错误讯息。

「他们干么要规定得这么严？」

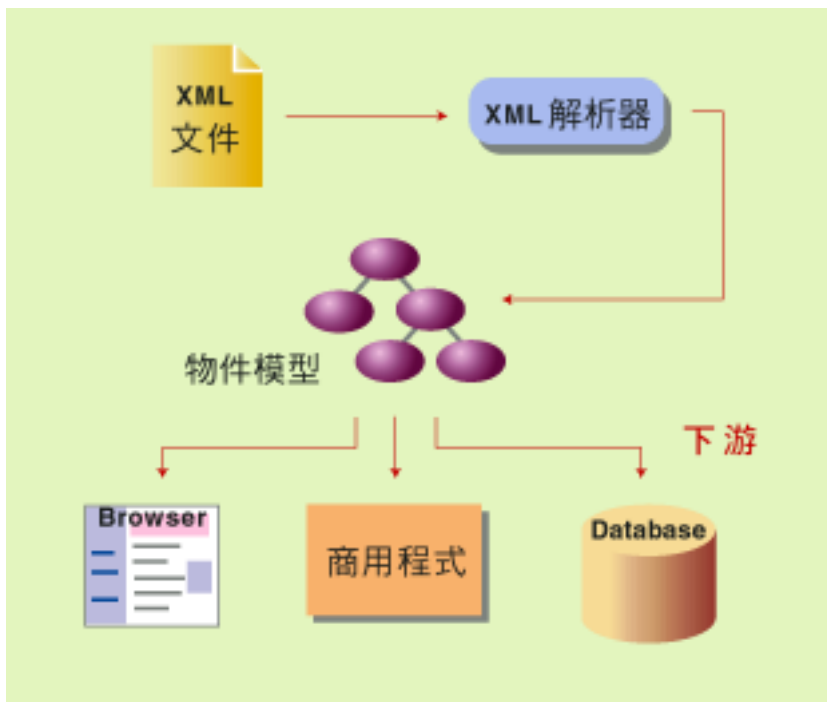


图 2.2: 解析器是处理任何 XML 文件的第一关，分析出来的物件则交给下游的应用程序作进一步的用途

XML 简洁的语法，及对格式的严格要求，大大地造福那些吐血、想破头去发展标注语解析器、浏览器的可怜工程师。这个容易发展的特性，有利于 XML 的快速普及化。XML 在发展时一共订了十条最高指导原则，这是第四条，在设计 XML 的科学家之间的打趣说法是：一个电脑科系的研究生，花顶多两个礼拜时间，就应该能编出一个解读 XML 资料的程序。

要附带一提的是，并不是用两大浏览器测试过没问题的网页，就保证完全符合 HTML 标准。事实上，正因为网络上有太多不合格的网页，使得设计浏览器的工程师无不绞尽脑汁，让他们的产品能够尽量圆滑地处理各种五花八门、语法偏差的网页。这就像前面提到的老外问路的例子，尽管我们直觉他讲出来的话很奇怪，但或许我们能大致猜得出来他要说什么（所以我说我们大脑中的语法解析器真的是很厉害）。

2.5 XML 文件必先要「及格」

「及格」在 XML 中的正式说法叫“**well-formed**”，也就是**格式正确**。任何文件要能称得上是 XML 文件之前，必得先「达到及格标准」。达不到标准的文件，会让 XML 解析器噎到，解析失败，什么都做不成。

那么到底要怎样的 XML 文件才算及格呢？不难，主要有以下几个原则：

2.5.1 所有元素都要正确地关闭

在表 2.1 中，我们看到所有的元素 [2.2]，不管其中穿插了多少笔资料、或几个子元素，到最后肯定都会有一个像 HTML 中的结束标签把这个元素「关起来」，譬如：<作者> 某人 </作者>。根据 HTML 的标准规定，有不少标签，例如 <p>、<tr>、<td> 等，它们的结束标签是可有可无的；但在 XML 中，结束标签绝不可少。如前面所说，这样严格的规定，大大减轻了发展 XML 解析器和其他开发工具时的负担。

脑筋动得快的读者可能已经在想了：「那万一有像 HTML 里，
、` 这类自成一个单元的标签怎么办？」

问得好！这在 XML 中叫「空元素」(empty element)，因为这样的元素不内含任何文字内容，只有属性。XML 为空元素特别发明了一种新的表示法，像这样：

`<元素 />`

如果带有属性的话则写成：

`<元素 属性甲="foo" 属性乙="bar" />`

2.5.2 标签之间不得交叉

我们继续使用「推荐丛书」的例子。假设今天有这样一个标签排列：

```
<书籍>
  <名称>如何在 7-11 白吃白喝
  <作者>
</名称>
  无名氏</作者>
```

这就犯了「标签之间不得相交」的大忌，会被当掉。XML 中规定，所有的元素排列必须是严谨的树状结构。树状结构的观念对学 XML 的人非常重要，在使用 DOM、XSLT^[7.2]，和 XPointer 来分别控制、转换，连结 XML 文件时，都需要随时对文件的内部结构了若指掌。如果您对资料结构 (data structure) 和物件导向的概念较为单薄，而在未来可能需要编 XML 类的程序的话，建议您找时间加强这方面的基础。

2.5.3 所有属性都得包上引号

在 HTML 中，我们已经被宠坏，常常忘了或懒得用引号把各标签的属性值包起来。网页浏览器对这样的写法通常都能正确处理，照单全收，让这个漏加引号的现象更加普及。甚至就连一些网页开发、转换工具都这么做（譬如 Office2000 所存成的 HTML 码）。这对 HTML 不是什么大问题，但拿到 XML 中，卻会惨遭被 XML 解析器判出局的命运。因此，加引号的习惯，越早养成越好。

2.5.4 其他规定

其他格式正确的要求还很多，上头提来说明的，只是几个大家在应用上最常碰到的。其他 XML 格式正确方面的规矩，就不在此一一列举，因为这本书的目标不是要把您训练成 XML 专家，而是要让您能很快地进入情况，开始应用。事实上，有些这方面的规定，可能只有设计 XML 解析器的软件工程师们需要知道，光是一一解释这些条文细节，就足足可以写一本书（那书名大概也要改叫「爱困 XML」了;-）。

2.6 以法为証

有时候光是及格还不够。现在假设我们将附表 2.1 中的 XML 码加以修改，把第二笔 <书籍> 资料，改成：

```
<书籍>
  <名称>如何在 7-11 白吃白喝</名称>
  <作者>无名氏</作者>
  <售价 货币单位="新台币">120</售价>
  <售价 货币单位="新台币">90</售价>
</书籍>
```

以上的 XML 码在格式上没有问题，算是 well-formed 的 XML 文件（您可以用上述「及格标准」来一一印証看看）。但一本书同时有两个价码，而且使用的是同一种货币，似乎有点怪，不过或许在某些场合下是可以成立的（譬如，其中一个团体订购的

优待价)。但是，这是不是就是网络书店经营者心目中希望的格式呢？又假设，今天《如何在 7-11 白吃白喝》这本畅销书冒出一个第二作者，叫「痞子」，他才是真正幕后的「幽灵作者」(ghostwriter)，无名氏只是坐在那里等收钱的（就像比尔大哥那两本畅销书;-），那么我们的 XML 标签该如何表示呢？到底是：

```
<书籍>
  <名称>如何在 7-11 白吃白喝</名称>
  <作者 人数="2">无名氏, 痞子</作者>
  .....
```

还是乾脆就这样：

```
<书籍>
  <名称>如何在 7-11 白吃白喝</名称>
  <作者>无名氏, 痞子</作者>
  .....
```

因为作者人数可藉由逗点的数目，让处理资料的程序自动算出。还是：

```
<书籍>
  <名称>如何在 7-11 白吃白喝</名称>
  <作者>无名氏</作者>
  <作者>痞子</作者>
  .....
```

呢？很明显地，我们现在需要定义一套法则来规范它。我们管这套法规叫 DTD^[8] (Document Type Definition)，也就是对某种 XML 文件在格式上的定义。我们可以在其中规定，「作者」这个元素到底是可以出现多次、还是在任何一个 <书籍> ... </书籍> 区块中只能出现一次。还有像一个元素能包含哪些属性、子母元素相互依存的关系、各个元素出现的顺序等，都能用 DTD 一一清楚地加以定义和规范。用 DTD 定义出来的一套 XML 应用，专业术语叫「语汇」。

可以用 DTD 来确认其正确性的 XML 文件称作 **valid** (**有法可証的**) XML。会根据 DTD 中的定义来确认 XML 文件正确性的解析器叫“validating parser”，没有这种功能的解析器叫“non-validating parser”。

2 XML 语法领进门

在使用 XML 时，我们通常会使用 DTD 已经设计好的现成 XML 语汇，譬如在第 1 章所介绍的 MathML^[1.3.1]、SMIL^[1.3.2] 等，或使用由各同业公会所制订的专用语汇。自行设计 DTD 难度较高，但也不是那么遥不可及。我们在第 8 章会解释 DTD 的语法。

XML Schema :
<http://www.w3.org/TR/xmlschema-1>

目前 W3C 正在研发、众所企盼的 XML Schema (组织架构) 标准，在未来不但可能会逐渐取代 DTD，更将提供更多的功能。更好的是，XML Schema 完全采用 XML 语法，不像 DTD 那般，有自成一格的怪异表示法¹。

2.7 大小写有分

XML 常见问题集：
<http://www.ucc.ie/xml>

XML 常见问题集 (FAQ) 中有一条，问说 XML 元素、属性名有没有分大小写。答案是有。XML 在这点上和 HTML 截然不同。这个有趣的现象就好比各操作系统对档名大小写的态度一样 — XML 正像 Unix 家族的操作系统，大小写分得很严；而 HTML 则像 Windows 和 MacOS，档名大小写无所谓。这点还请读友们特别注意。

2.8 CDATA 区

在 HTML 文件中，每当我们举例或附上源代码时，我们会把它放到 `<pre> ... </pre>` 或 `<xmp> ... </xmp>` 区块中。在 XML 中，要达到这样的功能，要用 CDATA (读作：C data)。CDATA 中的 C 是 character (字) 的缩写。在 XML 中，所有 Unicode^[3] 中定义的字码，包括中文字，都算是合格的字，而不狭窄地局限于西文字母。那么到底该怎么写呢？请看：

```
<![CDATA[
    小蜜蜂，嗡嗡嗡          &i; ..... zzzzz
飞到西 <<<<
                                     >>>>>> 又飞到东
]]>
```

¹XML DTD 的语法是直接继承自 SGML。

CDATA 区以 `<![CDATA[` 为起始，`]]>` 为终了。区块内容中唯一不能包含的，正是 `]]>` 这个终了信号。其他资料，只要是合格的 Unicode^[3] 字，都可以自由放置其中。

「哇！设计成这样，谁记得住？」

其实这正是它设计巧妙之处。就是要故意设计成这样，才能将无心的错误减到最低。我们看 HTML 中的 `<pre> ...</pre>` 区块就很烦，在结尾的 `</pre>` 之前，必须自己随时注意，把所有的 `<` 符号「跳脱」为 `<`，否则浏览器有可能把这些 `<` 号误当成是一个新标签的开始，而产生意想不到的后果。XML 中故意设计出像 `]]>` 这种不太可能在日常文件中出现的符号排列，让我们在写 CDATA 区时方便不少，不用记这个又忘那个的。

我们在前面第 2.7 节中刚谈过，XML 中要区分大小写，因此 `<![CDATA[` 不可写为 `<![CDATA[` 或 `<![Cdata[`。

依标准规定，出现在 CDATA 区中的资料，解析器在解析时不许乱碰，而要原封不动、一五一十地交给下游的程序，严格的程度，就连任何在区块起始和结尾处的空白、换行字元也无法倖免。请比较：

```
<![CDATA[我是区块中的第一行
我是第二行
]]>
```

和：

```
<![CDATA[
我现在被挤到区块中的第二行了 :- (
我被挤到第三行了
]]>
```

看得出其中的不同吧？结论：如果您不想在文件的每个 CDATA 区之前多一个空行的话（譬如有美观上的考量），则应采用以上第一种写法。

2.9 一空两空大不同

在讨论 XML 对空白字元的处理态度之前，我们最好先对「空白字元」作明确的定

E

2 XML 语法领进门

义。XML 中把空白字元定义为 `space`、`tab`、`CR`，和 `LF` 这四个。对 `space` 和 `tab` 不需要多废话，大家都知道这两键。但 `CR/LF` 则要稍加说明一下。

`CR` 代表 `Carriage Return`，是打字机时代遗留下来的称呼。这个 `ASCII` 字元通常是隐形的；它是 `MacOS` 平台上的换行记号。`LF` 是 `Line Feed` 的缩写，是 `Unix` 上的换行记号。`DOS/Windows` 平台则使用一个 `CR`，后头紧接着一个 `LF` 来标示换行。

上头提到 `CDATA` 区中对空白和换行字元从严办理，这是可以理解的。那么在 `CDATA` 区以外，譬如在 `<元素> ... </元素>` 中的文字内容呢？

在 `HTML` 中，文字内容中可以「一空」、「两空」，甚至「三连空」、「多连空」，出来的结果都会一样，因为不管连续有几个空白，一律都当成是一个，这是因为 `HTML 标准` 这样规定。

HTML 标准：

<http://www.w3.org/TR/REC-html40>

但是 `XML` 正好相反。`XML` 中规定，所有位于标签以外的空白，解析器要一个个忠实地交给下游程序作进一步处理。因为这个限制，我们必须改变我们编程的习惯。在写 `HTML` 标注文件时，不少人有时换行，甚至缩排 (`indent`) 的好习惯，让源代码清楚易读。大多数网页开发工具，甚至还会替我们作“pretty-print”，依文件的逻辑架构，和标签出现的位置，作深浅不同的缩排。但在 `XML` 中，如果把：

```
<作者>胭脂虎</作者>
```

写成：

```
<作者>
    胭脂虎
</作者>
```

二者是不一样的，因为后者多了一个 `tab` 字元（「胭」字前的缩排），和两个换行记号（分别在 `<作者>` 及「虎」字之后）。如果把上面第一种写法中的「胭脂虎」前后各加上一个空白，把文字和标签隔得开一点，即：

```
<作者> 胭脂虎 </作者>
```

，那么解析器 `parse` 出来的，又会是另一种结果。

「为什么要这样规定？像 `HTML` 那样规定不是很好吗？」

这是因为考虑到，有的 XML 文件可能需要保留空白字元，譬如诗词一类的内容，空白都有其存在的价值及特别意义。

如果我们想明确地告诉 XML 程序，不要随便把空白去掉，而要尊重原着的精神，可以在标签中加入 `xml:space` 这个 XML 内定的属性，像这样：

```
<诗句 xml:space="preserve" >
小雨伞啊！      小雨伞
      一只小雨伞
</诗句>
```

如果我们想刻意让读者看到这样精雕细琢的诗句排列的话 ;-)。

2.10 PI 与样规链结

XML 中还有一种标注，叫 PI，是 Processing Instruction 的缩写。PI 的标注是以 “<?” 开头，“?>” 结尾（XML 宣告 [2.1] 有时被人当成是一个 PI 的特例，但严格讲起来不是）。通常 PI 是用来传递情报给解析器下游的程序的，譬如我们想用样规 [1.2] (style sheet) 来美化附表 2.1 中的阳春 XML 码，不管是用 CSS [4.1] 或 XSL [7.1]，都必须有个机制，让浏览器知道要到哪儿去找样规。为此，W3C 特别颁布了一个专为连结样规所设计的 PI，写法为：

```
<?xml version="1.0" ?>
<?xml-stylesheet href="style.css" type="text/css" ?>
```

“`xml-stylesheet`” 这部分称作 PI 的 **目标 (target)**。以上的 PI 是用来告诉浏览器去抓一个叫 `style.css` 的 CSS 档。如果要连结 XSL 样规，就写成：

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

这份链结样规的标准，可自 <http://www.w3.org/TR/xml-stylesheet> 下载。

2.11 何去何从

学完了 XML 的基本语法，如果您对设计 DTD 特别感兴趣的话，可以趁记忆犹新的时候，先跳到第 8 章阅读。



3 Unicode 说分明

3.1 Unicode 简介

Unicode（统一码）顾名思义是一个将世界上几十种紊乱的文字编码整合在一起的努力。其幕后是由美国各大电脑厂商所组成的 **Unicode 策进会** 来推动。目的在推广一个世界通行的编码体制，将所有世界上常用的文字都涵盖进去，进而减少各电脑商开发国外市场时所遭遇到的问题（对我们这些常需要在各种文字码之间作转换的网络人，可说是感同身受）。

Unicode 策进会：
<http://www.unicode.org/>

为了要将成千上万的文字统统收集到一个共通的编码机制底下，在兼顾经济的原则下，不管是东方或西方文字，每个字在 **Unicode** 中一律以两个 **bytes** 来代表。这样一来，就至少能有 $2^{16} = 65536$ 种不同的组合¹，足以应付目前绝大多数场合的需要。

在与 **Unicode** 相关的各技术文件中，我们常会看到 **ISO 10646** 和 **UCS** 这两个名词。**ISO** 是位于瑞士的国际标准局的缩写。它所颁布的第 10646 号标准叫 **UCS (Universal Character Set)**，也就是世界通用字集（以下简称「**通用字集**」）。**UCS 通用字集** 的做法是用四个 **bytes** 来编码，将世界上所有任何官定和商用的编码大小通吃、一网打尽。值得庆幸的是，**Unicode** 策进会自 1991 年以来便和 **ISO** 的 **UCS** 小组密切配合，让 **Unicode** 和 **ISO 10646** 保持一致。因此，**Unicode** 自 2.0 版开始，便和 **ISO10646-1** 使用完全相同的字库和字码。

康熙字典里的中文字就有四万七，如果再加上里面没有的简体字，和一些笔划写法不同的日文字，那么 **Unicode** 六万多字的分配空间，光用来编汉字就已捉襟见

¹我说「至少」，是因为 **Unicode** 中采用了一种机制，可以应付未来编码空间用尽的问题。稍后会提到。

纒、不敷使用，哪还有馀地给泰文、阿拉伯文等几十种其他文字使用？针对这个问题，Unicode（和 UCS）采用了所谓的「中日韩文整合」(CJK Unification) 的解决方案，把中、日、韩文中笔划近似的汉字，尽量以一个单码来代表。例如草字头在繁体中文里要算成四划（两个十字），但在简体中文及日文中只有三划（中间一笔横贯），Unicode 中忽略这种微小的差别，不再替所有带有四划草头的字单独设一个码。但是如果一个字在简、繁体中差别比较显著、可能会对使用者造成文意理解上的困难或不便时，Unicode 中会为简、繁两种版本个别编码。例如讠字旁在繁体中是七划，但在简体中只有两划（一点一弯勾），所有带讠字旁的字，在 Unicode 中都各有两个不同的字码来代表，一简一繁。

经过「中日韩文整合」的汉字，在 Unicode 中称做 **Unihan（统汉字）**。在 Unicode 2.1 及 3.0 标准中，共有二万多个统汉字。统汉字并不包含日常生活中罕见的汉字，如康熙字典中的许多古字。

注意

Big5 中常见的「裏」字（十六进制码为 0xF9D8），因为没有收录在 Unicode (2.1, 3.0) 的标准中，因此建议您在做 Big5 和 Unicode 相互转换之前，先把这个上下的「裏」换成左右的「裡」，否则「裏」字会变成“?”，即 ASCII 中的问号（这是 Unicode 中的规定，找不到正确对应的字，一律换成问号）。

完整的 Unicode 3.0（最新版）统汉字数据库可自 <ftp://ftp.unicode.org/Public/UNIDATA/Unihan.txt> 下载。要先警告您的是，这个 database 非常大，超过 16Mb（搞不懂他们为何不先压缩一下）。其中不但包含了 Unicode 和 Big5、CNS、GB2312-80、GB2312-12345、SJIS... 等相互的对应，更连台湾电报码、康熙字典中的位置等都详细地收录在里面。除了要编 Unicode 程序的读者外，一般读

者不需要下载这个 database。

3.2 字母游戏

在我们对 Unicode 抽丝剥茧、剖析一番之前，我想把几个常见的专有名词及观念先做个归纳和整理，这样的话，读者们在阅读到后面的章节时，比较不会迷失方向，甚至可将下一节「血淋淋的情节」，整段跳过不读（如果您对底下部分叙述看得有点不知所云，别担心，只要大概看一下，有个概念即可）。

UTF 是 Unicode/UCS Transformation Format（通用字集／统一码变换格式）的缩写，Unicode 策进会推荐使用的是 **UTF-8** 和 **UTF-16** 这两种格式。其中的“8”和“16”指的是 *bits* 数，而不是 *bytes*²。

UTF-16 基本上就是 Unicode 双 *byte* 编码的实现，再加上一个应付未来扩充需求的编码机制（很少用；下面会谈到）。

UTF-8 是一种不等幅的编码方式，在 UTF-8 之下，英数字（即 ASCII 字元）保持原状，完全不受影响（因此不需要做转换）；但其他语文的资料则需透过程序来做转换，而且会「变胖」，因为每个字需要额外多用一或二个 *bytes* 来编码。

UCS 通用字集中，订有 **UCS-2** 和 **UCS-4** 等编码方式，其中的“2”和“4”指的是 *bytes* 数，而不是 *bits*（对照：UTF-8/16）。那这两个又和上头的 UTF-8/16 有何不同呢？其实差别不大：

UCS-2 大体上就是 Unicode 采用的双 *byte* 编码，可以简单地把它们想成是一样的东西，不用为此伤太多脑筋。

²一个 *byte* 由 8 个 *bits* 所组成，相信大家都不陌生

UCS-4 是以四个 bytes 来代表一个字的编码方式，就目前而言，在每个 UCS-2 码之前补上两个空白的 bytes，便可得到相对应的 UCS-4 码。

☰ 好工具

为了方便大家将文件转换为 UTF-8 格式，我写了一个转码工具，叫 **ccnv (enCoding CoNVerter)**。这个小工具适合拿来作 UTF-8 和世界上几十种编码之间的转换。最大的特色是可以一次对一个目录底下所有的档案作穷尽转换。这是一个程序码公开的免费程序，以 Java 写成（目前只有命令界面），可以从两只老虎下载，内附 Big5 和 GB 中文的安装方法和使用说明，在此就不再重覆。下载点：http://2tigers.net/runpc/xml/runpc_xml_2.html。

此外，有一个功能较多，但不是免费的 UTF-8 转换工具，叫 **uniconv**，有试用版下载：<http://rosette.basistech.com/demo.html>。

XML1.0 标准：
<http://www.w3.org/TR/REC-xml>

XML1.0 标准中规定，XML 解析器至少须支援以 UTF-8 或 UTF-16 编码的 Unicode 字串，而当 XML 宣告中没有特别指明编码（**encoding**）时，则一律以 Unicode 看待，软件会自动侦测出文件是 UTF-8 或 UTF-16。目前市面上几家做得不错的 XML 解析器，不但符合标准规定，支援 UTF-8/16，更支援 Big5、GB 等亚洲文字码。这对我们来说，确实方便多多。

XML 标准中只明订解析器^[2.4]必须支援 UTF-8/16，而不苛求对其他编码方式（如 Big5 等）提供支援，是有其道理的。因为在 UTF-8/16 和各种国家、地区性的编码，如 Big5/GB2312/SJIS 之间做转换是很容易的事，一旦某个软件支援 UTF-8/16（但并不直接支援 Big5、GB），我们在资料输入或输出时只要配合一道 UTF-8 ↔ Big5/GB 的转换动作即可（上头提供的 **ccnv** 转码工具可以派上用场）。某些情况下，甚至不需要刻意

做字码转换，譬如 perl 的 `XML::Parser` 模块，虽然只能输出 UTF-8，但可接受 Big5 的输入。又如 Netscape、IE 等浏览器，能直接呈现 UTF-8 格式的文件，因此当输出的资料是打算给浏览器去读的话，就不肯定要再转换（除非有其他考量），而只要在文件的宣告中正确注明是用 UTF-8 即可。

XML::Parser :
<http://www.cpan.org/authors/id/C/CO/COOPERCL/>

遗憾的是，部分 XML 解析器和不少应用软件³，无法正确处理 UTF-8 编码的字串（或许他们暂时不担心亚洲市场吧？）。为什么在这里要特别强调 UTF-8？因为这是最起码的——如果某 XML 软件连 UTF-8/16 编码的资料都无法正确处理，那么它支援 Big5、GB2312 中文的机率只怕更加渺茫了。当我们喂它含 UTF-8/Big5/GB 码的资料时，通常会被赏给一段莫名其妙的错误讯息，要不就是一堆问号、或无法预料的结果（作者刻骨铭心的体验）。

在 UTF-8 和我们惯用的 Big5/GB 之间做转换尽管不方便，但至少它充分支援中文字。换句话说，如果某软件能正确支援 UTF-8，它无形中就对我们有很大的利用价值。这也正是 XML 标准强制规定解析器要支援 UTF-8 的精神所在。

3.3 血淋淋的细节

我想藉这个机会，把跟 Unicode 相关的各个重要观念，在底下一并解释清楚。希望能对 XML 的使用者，乃至编转码程序、或学 Java 语言的读者等都能有所助益。

如果您认为您学 XML 的动机只是基于好奇，或将仅限于使用现成的工具，不太可能直接和程序码打交道，那么上面所提供的基本概念，或许就已经足用了。避免接触底下血淋淋的科技细节，可帮助减少失眠、头疼等征状。但是，如果您平常也编一点程序，而且打算做各种 XML 的开发、应用的话，那么建议您把它耐心看完，尤其是 UTF-8 编码原理的部分，非常重要。事实上，底下将介绍的知识拿到许多其他的领域里也一样受用，像 Java 语言、Windows NT⁴，微软 Office 软件等，都大大地借重到 Unicode，

³也就是前面提过的，对解析器输出的资料做进一步利用的下游软件。

⁴这也就是为什么有些含中文字的 Java 软件在 NT 下可以正确显示，但拿到 Win95/98 上就不行了。

选它做内码。

3.3.1 Unicode 中的空间分配

以下各 Unicode 区位码皆以十六进制表示。

不意外，Unicode 的头 256 个字符和 ISO 8859-1（即俗称的 Latin-1，西欧字母）完全相同，其中前半段就是 ASCII。这段是从 U+0000 到 U+00FF。当然，每个 ISO 8859-1 码必须在前面补上一个空 byte (0x00) 后才是相对的 Unicode 码。

和我们有切身关系的 Unihan 统汉字，在 Unicode 中主要分布在 U+3400 到 U+9FFF 之间，此外，U+F900 和 U+FAFF 之间也有一些。事实上，Big5 和 GB2312 中的汉字和符号都在 U+4E00 到 U+9FFF 这块里面。

3.3.2 UTF-8 的编码原理和特性

知道了西欧字母和汉字在 Unicode 中的区位后，我们可以开始谈 UTF-8。在附图 3.1 中，每一个小方块代表一个 bit，绿色方块所代表的位元值 — 0 或 1 — 是钉死不变的。蓝色方块所代表的各个空出来的位元，则拿来存放 Unicode 内码。一群小方块（八个）代表一个 byte。UTF-8 共有三种可能的排列方式，各自需要用到一、二，或三个 bytes（所以才说它是不等幅）。每个 Unicode 字符按它在标准中分到的码位，来决定这个字符在转换成 UTF-8 时该用哪一种排列方式、以几个 bytes 来编排。我们在图中左方可以看到，在 U+0000 和 U+007F 之间的字符（即所有 ASCII 码），是以第一种方式、单 byte 来表示，但因为第一种形式的第一个 bit 是 0，所以 ASCII 码（从 0x00 到 0x7F）根本不需要转换，就自然是相对的 UTF-8 码；反过来说，只要不是 ASCII，在 UTF-8 中就肯定需要两个以上的 bytes 来编码。

Unicode 中落在 U+0080 – U+07FF 范围间的字符，则以第二种形式来标示。至于所有其他剩下来的字符（包括所有的中文字），则一律以第三种形式来编排。您可以自

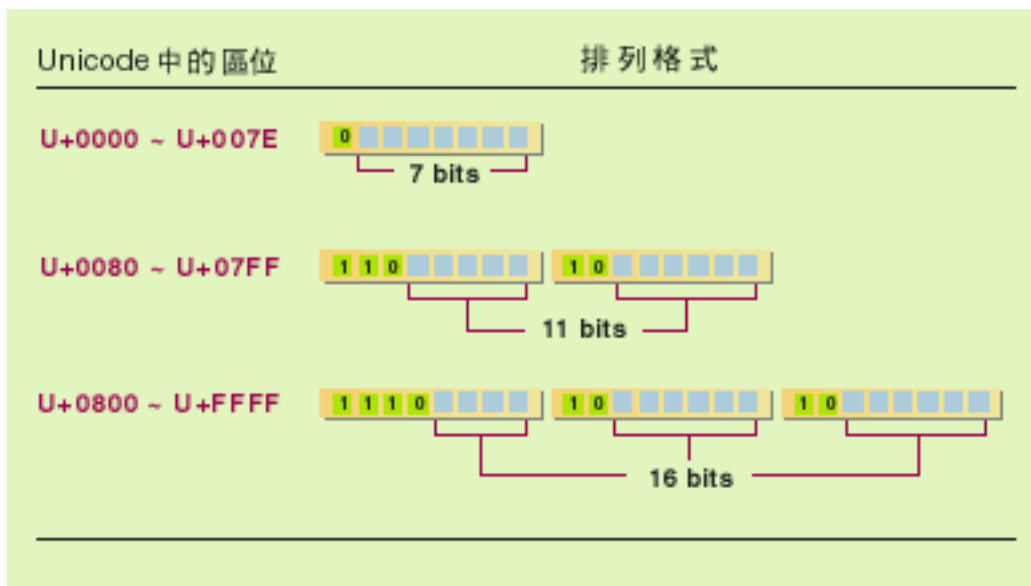


图 3.1: UTF-8 的编码原理

己算一算，看 UTF-8 的三种形式中各自提供的自由 bits（蓝色小方块）数，是否足以用来代表对应区位中的各 Unicode 码，答案是刚好够用。

那么当程序在处理 UTF-8 编码的文件时，要如何得知一个字符的疆界落在哪里，还有到底它是三种形式中的哪一种出现的呢？这就是图 3.1 中那些绿色小方块的作用了。每个以 UTF-8 编码的字符，不管是以一、二，或三个 bytes 的形式出现，第一个 byte 的前端都清楚地标示了该字符的 byte 总数。譬如 110 中有两个 1，即代表这个字符是以第二种形式出现、由两个 bytes 组成；而 1110 中有三个 1，暗示了这个字共占用三个 bytes。

每个多重 byte 的 UTF-8 码有一个共通的特性，即其中第二和第三个 bytes，一律以 10 这两个 bits 起头。由于其中最高的 bits⁵ 总是设成 1，可以很容易和那些在 UTF-8 中只用到一个 byte 的 ASCII 字元分别开来，方便侦错。

因为上述的设计特点，UTF-8 和 Unicode 之间，可以很轻易地做双向自由转换，而

⁵又称「最重要的 bits」；英文说“most significant bits”。

不会流失任何资料。事实上，这是所有 UTF 变换格式所必须满足的基本特性，达不到这个要求的变换格式便没资格叫 UTF。

3.3.3 UTF-8 的优点

现今大多电脑系统，仍是以 **byte** 为单元来做存取。如果直接以 UTF-16（双 **byte**）来存取资料，会产生许多问题。学过 C 的读者都知道，`0x00` 这个字元是有特殊作用和意义的；而上面才刚讲到，UTF-16 的头 256 个字码的第一个 **byte**，很不巧地，正是 `0x00`（第二个 **byte** 则是相对的 iso8859-1 字码），因此如果用 UTF-16 来表示档名、网址等，会惹出许多问题。而且不只是 `0x00`，其他还有几个特殊字元，也都可能和许多现行的操作系统、函式馆 (**libraries**) 等相冲。

此外，UTF-16 及许多亚洲地区现行的双 **byte** 编码方式，在应用上有些先天的缺陷：譬如，字与字之间的疆界不好找，程序在处理时必须从头扫描，才能正确可靠地找出某个字的疆界，缺乏效率。此外，相信大家都有这样的经验：有时候中文文件中某个地方被错杀一个 **byte** 时，所有在这个刀口之后的中文字都会坏掉，一直坏到下个英数字，或空白字元出现为止。

以上这些问题，在 UTF-8 中都不存在。要找字与字的疆界很容易，只要看目前这个 **byte** 的开头几个 **bits**，看是 `0`、`110`、`1110`，还是 `10` 便可很快地找到现在这个字的疆界。如果遇到坏掉的字，也不会拖垮后头一大票字。Unicode 标准中同时有明文规定，当 UTF-8 格式的文件中有不合理的组合出现时，该怎么处置，例如碰到第一个 **byte** 是以 `1110` 起头，但下一个 **byte** 却以 `0`、而不是正确的 `10` 起头。

3.3.4 UTF-8 的缺点

用膝盖一想就知道，大多数文字在 UTF-8 之下都会膨胀，尤其是中、日、韩文，在最糟的情况下（即整篇文章不含任何西文时），会变成原来的 150%。对我们来说，的确有点亏。

但从另一个角度来看，如果改用 UTF-16 来做文字码，固然不会增加亚洲文字所占的空间（都是两个 bytes），但对西文来说，却等于让所有的文件大小，一律加倍，这比膨胀一倍半还惨。

我们设身处地站在他们的立场来想，也难怪许多西方世界的软件业者和程序设计师对 Unicode 兴趣缺缺，尤其是在过去，当记忆体和硬盘价格没像近两年这么便宜的时候。

有了 UTF-8，为数众多的英文文件，不需任何转换，就自然符合 UTF-8，这对向英文世界的软件工作者促销 Unicode 有很大的帮助，毕竟当今网络上大多数电子文件，都是英文。

因此，从让 Unicode 在世界上快速、大量推广的观点来看，UTF-8 不失为一个可行的折衷方案，它的优点似乎大过于缺点。

3.3.5 UTF-16 中的代理对

上头提到，ISO 10646/UCS 使用四个 bytes 来编码，因此要应付未来的扩充需求可以说绰绰有馀。相对地，Unicode 只用两个 bytes，不但空间很容易用尽，而且如果要在未来继续和 ISO 10646/UCS 保持融通，势必需要一种机制来弥补两个 bytes 的不足。代理对 (surrogate pairs) 的设计便在这个背景下应运而生。

Unicode 将范围在 U+D800 到 U+DFFF 之间的区域保留给代理对使用。这个区域又拆分成两部分，称做「高低部」，第一部分（高部）是从 U+D800 到 U+DBFF，第二部分（低部）则从剩下的 U+DC00 到 U+DFFF，高低两部各有 1024 个码位。因此透过这个机制，Unicode 便可多容纳一百多万个字 (1024 × 1024)。当然，这些用代理对机制来编码的字必须多用一个 Unicode 的基本单元（所以才叫代理对），也就是一共要四个 bytes，比较不经济。

代理对的编码机制，和原先不需要使用代理对的六万三千多个基本 Unicode 码，合起来叫 UTF-16。

您或许会奇怪，既然代理对是设计来应付未来字数扩充的需要，那为什么不让高、低两部的区位重叠？也就是让两部都可充分使用从 **U+D800** 到 **U+DFFF** 这整块保留区域，为什么要区隔？因为这么一来，字的疆界很容易掌握，同时韧性高——即使有一、两个 **bytes** 被错杀，也不会后面跟着全死。

在现行的 **Unicode** 标准（**2.1** 和 **3.0** 版）中，并没有动用到代理对来编字（接下来几年内大概也不会用到），因为代理对以外的六万多区位，已经足以应付今日大多数场合的需要。尽管如此，我们可以利用其中的私用区，这是下一小节的主题。

3.3.6 私用区

Unicode 中保留了三块私用区，编程序的人可以拿来自由运用。其中两块在代理对的区域中，须配合代理对的编码方式（即 **UTF-16**）来使用。如果我们的程序需要用到 **Unicode** 中没有收录的古字或特殊符号，就可以利用私用区来替那些字符做编码。

Unicode 中第一块私用区从 **U+E000** 到 **U+F8FF**，共 **6,400** 个区位。代理对中的私用区则包括 **0xF0000 -- 0xFFFFFD** 和 **0x100000 -- 0x10FFFFD**，约十三万个区位。



4 XML 化妆术

4.1 CSS 入门

关于 XML + DOM + CSS，以及 XML + CSS + 中文标签在浏览器上的应用实例，请暂时先参考我在 Run!PC 读者服务专区中的例子：http://2tigers.net/runpc/xml/runpc_xml_3.html。

此外，我在第 70 期（1999 年 11 月份）的 Run!PC 杂志中有一篇 DOM 和 CSS 的入门指引，有兴趣的读者请参考。

Run!PC 杂志：
<http://www.runpc.com.tw>

4.1.1 选择式 (selector)

待续。



5 名称空间

名称空间 (namespace) 的概念，对 XML 非常重要，如果少了它，XML 的应用范围会大大地受到限制。如果您会用 Perl 或 Java，那么您很可能已经在使用名称空间（而不自知？）了¹。

名称空间的规定，并没有收录在 XML 1.0 的标准中，而是透过后来一份单独的标准来做增补，这份标准的全名叫“Namespaces in XML”。

XML 1.0 的标准：
<http://www.w3.org/TR/REC-xml>
Namespaces in XML：
<http://www.w3.org/TR/REC-xml-names>

5.1 为什么需要名称空间

假设我们有个 XML 的电子通讯录，专门用来存放客户资料，记载平日来往的公司客户：

```
<客户名单>
  <客户> <!-- 客户甲 -->
    <名称>新祥发</名称>
    <地址>...</地址>
    <电话>...</电话>
    <fax>...</fax>
    .....
  </客户>
  <客户> <!-- 客户乙 -->
    .....
</客户名单>
```

不过有些客户是业务专员带进来的，通常在和这些客户的连系方面，都是透过各负责的业务专员。至于业务专员的连系电话、email 等，都详细记录在公司的另一份 XML 文件——「职工名单」中：

¹在这两个语汇中，名称空间是透过 package 来实现的。

```
<职工名单>
  <职工> <!-- 职工甲 -->
    <姓名>...</姓名>
    <部门>业务部</部门>
    <职位>专员</职位>
    <月工资>36000</月工资>
    <电话>
      <分机>...</分机>
      <大哥大>...</大哥大>
    </电话>
    <email>...</email>
    <fax>...</fax>
    .....
  </职工>
```

为了连系方便，我们想在 <客户名单> 中，增加一项 <连系人>，用来存放业务专员的资料，这样当我们要连系由专员来负责的客户时，便可以马上知道该找哪位专员，打哪个电话。至于业务专员的连系资料，则可以直接从 <职工名单> 中萃取；换句话说，就是把 <职工名单> 中的部分资料，和 <客户名单> 结合在一起，像这样：

```
<客户名单>
  <客户> <!-- 客户甲 -->
    <名称>新祥发</名称>
    <地址>...</地址>
    <电话>...</电话>
    .....
  <连系人>
    <姓名>...</姓名>
    <电话>
      <分机>...</分机>
      <大哥大>...</大哥大>
    </电话>
    .....
  </连系人>
  .....
```

等等！「电话」这个元素 [2.2] 重复使用了！换句话说，「客户名单」和「职工名单」这两套 XML 语汇 [p.29]，如果放到一块用会打架。虽然对人脑来说，藉着分析上下文，可以很容易将两个出处不同的 <电话> 区别开；但对电脑程序来说，恐怕就不见得那么容易了。因此，在这种情况下，我们需要的是一套简单明了的办法，让机器在处理的时候

候，不用大伤脑筋。即使没有任何名称上的冲突，我们仍旧需要一个简单的方法，让电脑知道，哪些元素来自于哪个语汇，以便使用该语汇的 DTD^[8] 来确认文件结构的正确性。

有的读者可能已经想到一个可行的办法了，那就是把 <客户> 底下，一些太过笼统的子元素名，像「名称」、「地址」、「电话」等，一律冠上「客户」两字，也就是把它们分别改成「客户名称」、「客户地址」、「客户电话」，把名子定的精确一点。此外，<职工名单> 中的一些元素名也可以照办，譬如「姓名」和「电话」可以分别改为「职工姓名」和「职工电话」，这样就可以大大降低名称冲突的机率了。

这的确是个可行的办法，不过这么做必须更动 DTD 里的定义，但不见得所有的 DTD 都是由我们亲手设计的²，不是说改就改；就算真改了，我们依旧无法确保今后不会再遇到其他外来的同名元素。

名称空间的设计，便在这样的背景下应运而生。它的概念非常直接——如果每套 XML 语汇^[p.29]，都各自用一个独一无二的标志来代表，并且在使用时，把这个标志和语汇中的元素、属性名连在一块儿使用，就绝对不会和其他语汇扯不清了，因为每个语汇中的名称，都已经先被该语汇的独特标示码给修饰、限定 (qualify) 住了。其实这个解决方案，和上面提到、修改标签名称的做法，有些类似之处。刚才我们是把「客户」二字加在意义含糊的名称，如「地址」、「电话」之前，让「客户名单」这个语汇里的元素名变得比较独特，进而减少和其他语汇起冲突的机会。而名称空间的做法，则是让编写 XML 文件的人替文件中所用到的语汇，自行指定标示码。

一个独特的标示码，代表一套语汇；各语汇中的名称，因而能各得其所，有自己的活动空间。「名称空间」的命名，正是由此而来。

²事实上，就像软件一样，大多数人不会去碰 DTD 设计的東西，而会直接利用现成的 DTD 所定义出来的语汇，譬如由一个同业工会制订出来的。

5.2 名称空间的长像

咱们先来看看，XML 到底选用什么来做名称空间的标示码。

5.2.1 标示物

上面已经提到，标示代码最重要的特性是要能独一无二。这可以藉由很多种方式来达到，譬如目前 Internet 上的网域名，便是独特的，像“2T.com”已被人抢先一步注册走了 :-/，我就不可能再申请一样的名子，而只得改用“2Ti.com”。注册固然是一种方法，但我们想想，如果 XML 名称空间也要仿照 DNS 域名系统那样，由专责机构来分配、管理，那不但会变得很复杂，而且会很沒效率；使用前得先注册，甚至还得付费，搞不好连大家争着抢好名子的乱象，都会重演 :-)

XML 采用了一个聪明、简便的方法来指定名称空间：既然网域名已经是独特的，那何不乾脆直接利用，把基础建在它上头，用网址来代表名称空间呢³？嗯，出这招太极拳，果然是高！这样一来，责任便全「推派」给各公司、机关来负责。因为各机关对放在自己网域底下的网址，应该有完全的主控权，所以一切名称空间标示码的指定工作，就统统交给编写 XML 文件的人来决定。

至于网址的选择问题，虽然名称空间的原始目的，只是用来对一份文件中不同的语汇加以区隔，指定哪个网址不见得顶重要，只要能区隔开来就好，所用的网址甚至不需要存在！，但在此还是建议您养成好习惯 — 不归我们的网域，在沒徵得人家允许之前，不要乱「借用」，譬如拿 W3C 或其他知名公司的网域名来瞎编一个网址，给自己设计的语汇当名称空间用，就不是很好的做法⁴。大家都应该只用属于自己 homepage 底下的网址。

³Java 的名称空间也是用类似的方法，直接架构在 DNS 域名上。

⁴笔者的确见过有人这么做。

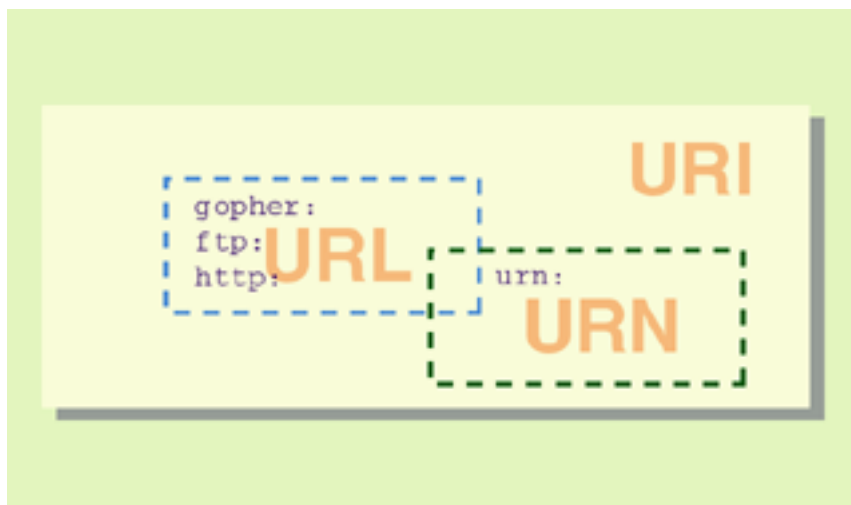


图 5.1: URL、URN，和 URI 的关系

5.2.2 URL、URN、URI — 别搞迷糊了

我们惯称的「网址」，大致上是相当于 URL。其实名称空间的规定中用的是 **URI**，但为了让读者容易理解，我刻意在上头使用大家比较熟悉的「网址」一词。URI 全名叫 Uniform Resource Identifier（统一资源标示码），这是一个 Internet 标准，记载于 [RFC 2396](#)；至于一般人较为熟知的 URL 一词，则是 Uniform Resource Locator（统一资源定位码）的缩写。那 URL 和 URI 除了名子外，还有啥不同呢？请看图 5.1 中的集合关系。基本上，URI 比较广义，泛指所有以字串标示的网络资源，范围涵盖了 URL 和 URN。URL 指的是标有通信协定（如 HTTP、FTP、GOPHER）的字串⁵。URN (Uniform Resource Name) 则通常用来标示持久⁶、而且有专责机构负责的资源，譬如图书馆的图书总目。

RFC 2396 :
<http://www.ietf.org/rfc/rfc2396.txt>

在这本书中，「网址」一词用得比较松散，可能代表 URL，也可能指 URI。

⁵有学问的说法为「明确标示出资源的取得途径及管道的代码」。

⁶与网站、网页的寿命相比。

5.2.3 名称空间实例

在讨论名称空间的宣告方式之前，先请您作个「成品预览」：

```
<?xml version="1.0" encoding="UTF-8"?>
<k:客户名单 xmlns:k="http://foo.bar.com/xml/customer.dtd"
            xmlns:职工="http://foo.bar.com/xml/employee.dtd"> A
  <k:客户> <!-- 客户甲 -->
    <k:名称>新祥发</k:名称>
    <k:地址>...</k:地址>
    <k:电话>...</k:电话>
    .....
    <k:联系人>
      <职工:姓名>...</职工:姓名> B
      <职工:电话>
        <职工:分机>...</职工:分机>
        <职工:大哥大>...</职工:大哥大>
      </职工:电话>
      <职工:email>...</职工:email>
    </k:联系人>
  </k:客户>
  <!-- 客户乙 -->
  <客户 xmlns="http://foo.bar.com/xml/customer.dtd"> C
    <名称>同仁堂</名称>
    <地址>...</地址>
    <电话>...</电话>
    <fax>...</fax>
    <联系人>
      <职工:姓名>...</职工:姓名>
      <电话 xmlns="http://foo.bar.com/xml/employee.dtd">
        <分机>...</分机> D
        <大哥大>...</大哥大>
      </电话>
      <职工:email>...</职工:email>
    </联系人>
  </客户>
</k:客户名单>
```

如同上面的例子，在标注名称空间的 XML 文件中，您会看到一些元素、属性名 [\[2.2\]](#)，是用冒号把两个名子连在一起的。

5.3 名称空间的宣告

5.3.1 前置字串

我们已经知道，名称空间的原理是藉着「独特标示码+名称」的方式，让每个 XML 语汇中的元素、属性名都能有自己的小天地，而不会和其他语汇中的名子互抢地盘；换句话说，冠上了标示码，每个名子就可变得独一无二。

A

您可能已经一边抓头，一边在问了：「上面不是说到，XML 选用 URI 来做独特的标示码吗？可是一般的 URI、网址都蛮长的，如果直接拿来放在元素、属性名之前使用，是不是太冗长了点？如果能用个简短的代号来替代 URI 就好了。」

完全正确！URI 不但太长，而且里面往往含有一些 XML 语法规规定不准用作元素、属性名的字元（譬如网址中肯定都有的“/”），因此使用代号不仅是为了便利，更是势在必行。这样的代号，在名称空间的标准中叫「前置字串」(namespace prefix)，由编辑 XML 文件的人自由指定。前置字串只能含有 XML 标准中允许作属性名的字，这包括了英文字母，和所有收录在 Unicode^[3] 中的汉字。还有，XML 标准把所有以“xml”这三个字母开头的前置字串保留作特殊用途，所以使用者自订的前置字串不许用这三个字母来起头，不管是“xml”、“XML”、“xMI”...，什么样的大小写组合都不可以。

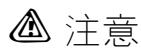
用前置字串来代表名称空间的标示网址，就好比我们用 Unix 上的 symlinks、Windows 底下的 shortcuts（捷径），或 MacOS 里的 aliases 来代表档案路径一样，只要能发挥指向的功能就好，取作什么名字倒是其次。若是能简短、达意当然是最好，这样既方便书写，又利于阅读。事实上，XML 软件在处理文件之前，会先将所有前置字串还原成它们所代表的 URI。

那前置字串和它所代表的名称空间，要透过什么样的宣告方式，才能把它们关联在一起呢？这得透过一种特别的属性来达成——这类属性的属性名一律以“xmlns:前置字串”的形式出现⁷，而属性值则正是该前置字串所要代表的名称空间，

⁷xmlns 中的 ns 是 NameSpace 的缩写。

也就是一个 URI。属性名中的 `xmlns` 和前置字符串间，必须以英文的冒号 (:) 相连。因为冒号已经被选作分隔记号，有特殊的功用，所以前置字符串中不可以再用冒号。一个完整的名称空间宣告，在一份 XML 文件中写起来就像这样：

```
<?xml version="1.0" encoding="UTF-8" ?>
.....
<某元素 xmlns:某前置字符串="http://some.uri.com/some/namespace" >
.....
</某元素>
```



注意

您可能会在一些已经过时的 XML 书籍或文件中看到像 `<?xml:stylesheet ... ?>` 的 [PI\[2.10\]](#)，名称里面有冒号，和我们在第 2 章中介绍的 `<?xml-stylesheet ... ?>`（使用 - 号）有出入。这是历史的遗迹 — `stylesheet PI` 之所以舍弃冒号、改用连字号，正是因为和名称空间的设计相冲突。

因为名称空间是藉着属性来作宣告，所以必须依附在一个元素标签里面。至于该放在哪个元素里，要看实际需要而定，我们稍后会针对这个问题作说明。

要在一份 XML 文件中，同时使用多种语汇（这正是名称空间的目的），只要先将各语汇的名称空间和前置字符串定义、宣告好，然后视需要，在元素或属性名之前，冠上适当的前置字符串，并将二者以冒号隔开即可。这样一来，不但哪个名称来自哪个语汇，划分得一清二楚，更不用担心会有同名的冲突。有了这样的机制，前面提到的「客户名单」和「职工名单」结合使用的问题，可以圆满地得到解决了：

```
<k:客户名单 xmlns:k="http://foo.bar.com/xml/customer.dtd"
             xmlns:y="http://foo.bar.com/xml/employee.dtd" >
  <k:客户>
    <k:名称>新祥发</k:名称>
    <k:地址>...</k:地址>
    <k:电话>...</k:电话>
    .....
```

```

<k:连系人>
  <y:姓名>...</y:姓名>
  <y:电话>
    <y:分机>...</y:分机>
    <y:大哥大>...</y:大哥大>
  </y:电话>
  <y:email>...</y:email>
</k:连系人>
</k:客户>
.....
</k:客户名单>

```

在上例中，<http://foo.bar.com/xml/customer.dtd> 是我们给「客户名单」这个 XML 语汇指定的名称空间标示码，<http://foo.bar.com/xml/employee.dtd> 是「职工名单」的标示码。为了打字省力起见，我们用 **k** 这个前置字串⁸来代表「客户名单」的标示网址 (`xmlns:k="http://..."`)；用 **y** 来代表「职工名单」 (`xmlns:y="..."`)。我们同时也看到，一个元素里面可以放置多个 `xmlns` 的属性宣告。一旦宣告好了以后，只要将各个元素名都冠上适当的前置字串，哪个元素归属哪个语汇，分得清清楚楚。

5.4 名称空间的范畴

范畴 (scope) 的概念，对程序设计非常重要。假设我们把上面的例子改写成：

B

```

<k:客户名单>
  <k:客户>
    <k:名称>新祥发</k:名称>
    .....
    <k:连系人 xmlns:k="http://foo.bar.com/xml/customer.dtd"
              xmlns:y="http://foo.bar.com/xml/employee.dtd" >
      <y:姓名>...</y:姓名>
      <y:电话>
        <y:分机>...</y:分机>
        <y:大哥大>...</y:大哥大>
      </y:电话>
      <y:email>...</y:email>

```

⁸字串 (string) 可以只含有一个、甚至零个字元 (空字串)。

```
    </k:联系人>
  </k:客户>
  .....
</k:客户名单>
```

也就是把名称空间延后到 <联系人> 的地方才宣告。这样一改后，虽然仍旧可达到类似的功效，把两套来源不同的元素区分开，但名称空间所涵盖的范畴已改变，所有红字部分必须去掉，否则就是错误。为什么？因为 <客户名单> 和 <客户> 分别是 <联系人> 的祖母和母元素，层级要比 <联系人> 高，<联系人> 罩不住她们 ;-)，但是要罩层级比它低的子、孙元素，如 <姓名> 、<大哥大> ，则不是问题。

因为名称空间有范畴的特性，所以我们在元素中安插名称空间宣告时，必先将涵盖层级的问题考虑进去。最简单的方法，当然是在最外围的根元素标签里，先把所有的名称空间和要指定的前置字串都一肯定义好。这正是前一节例子中的做法。如果要在文件中途处对某名称空间作宣告，则最好确定所有来自这个空间的元素，都能被宣告所在的元素所涵盖，否则无非是在编写 XML 时，给自己多添麻烦，因为出了这个范畴后，每当需要再用到同一个名称空间时，又得写一长串，再宣告一次。

5.5 预设的名称空间

C

XML 名称空间有预设 (default) 的概念。没有冠上前置字串和冒号的元素、属性名，一律视为在预设的名称空间底下。如同一般的名称空间，我们可以透过宣告，把预设的名称空间标示出来。它的写法和前面介绍过的 `xmlns:前置字串="URI"` 很像，但少了冒号和前置字串这部分，也就是变成 `xmlns="URI"`；换句话说，我们可以把预设名称空间的前置字串，想成是空字串。上面的范例，如果改用预设空间的做法来写，就成了：

```
<客户名单 xmlns="http://foo.bar.com/xml/customer.dtd"
           xmlns:y="http://foo.bar.com/xml/employee.dtd">
  <客户>
    <名称>新祥发</名称>
    <地址>...</地址>
    <电话>...</电话>
```



```

.....
<连系人>
  <y:姓名>...</y:姓名>
  <y:电话>
    <y:分机>...</y:分机>
    <y:大哥大>...</y:大哥大>
  </y:电话>
  <y:email>...</y:email>
</连系人>
</客户>
.....
</客户名单>

```

「客户名单」这套语汇现在被指定为预设的名称空间，任何没有前置字串和冒号的元素名，都会被认作是属于这个空间。

预设的名称空间，可以在文件中变来变去，就像一个前置字串在同一份文件中，可以先后和不同的 URI 串连一样，并非宣告过一次之后，就不能再用，只不过后面的宣告会覆盖前面同名的宣告，譬如：

```

<客户名单 xmlns="http://foo.bar.com/xml/customer.dtd">
  <客户>
    <名称>杰儿克</名称>
    <地址>...</地址>
    <电话>...</电话>
    <网址>...</网址>
    .....
    <附注>
      <!-- 改用 HTML 来作预设空间 -->
      <p xmlns="http://www.w3.org/TR/REC-html40">
        此客户相当难缠，而且<b>非常抠</b>。和他们老板应酬之前，最好先到他的
        <a href="http://jerk.com/boss/">个人网站</a>去看看他最近喜欢玩什么。
      </p>
      <!-- 离开 HTML 名称空间的范畴 -->
    </附注>
    .....

```

5.5.1 预设空间与范畴联合运用

上面的范例巧妙地利用到预设空间和范畴的特性，把「客户名单」和 HTML 两套语汇各自的范畴，划分的一清二楚：所有在 <附注> 里面的部分，不包含 <附注>，隶属于

HTML 的名称空间，出了 <附注> 以后，「客户名单」的名称空间又重新生效，恢复为文件的预设空间。

这个范例同时展示了另一个实用的技巧 — 超连结。虽然 XML 有自己的连结方式，合称为 **XLink/XPointer**，在功能、花样上要比 HTML 的简单连结模式强上十倍百倍；但是目前这两套超连结语法都还停留在草案阶段，尚未定案为正式标准，而浏览器的支援也非常有限。因此借用 HTML 的 <a> 标签来作连结，不失为一个很好的过渡期解决方案。

把 HTML 标签适时拉进 XML 这招非常好用。不只是能作简单的超连结，还可以做表单 (form)。因为 XML 并没有内建表单的功能⁹，目前也没有一个通行的“FormML”标准，因此要在 XML 文件中附加填表功能，并且能在浏览器中正确地显现、送出，最方便的方法就是透过名称空间，把 HTML 的 <form>、<input> 等元素内嵌到 XML 档案里头去。IE5 和目前仍在发展的 Mozilla (Netscape 5) 都支援这样的做法，一切就如同在 HTML 中一样，也可以配合 ECMAScript^[p.14]（即 JavaScript）来执行，作各种事件处理动作 (event handling)，如 onClick、onMouseOver 等。

下面的 XML 码，如果要在 IE5 中正常运作，必须指定一个外部的 CSS 样规^[4.1]，这个 CSS 档倒不见得肯定要存在，但是如果没有加上那行 PI^[2.10]，IE5 会用它内建的 CSS 样规来呈现 XML 文件，表单和连结的效果都会不出来。

```
<?xml version="1.0" encoding="GB2312" ?>
<!-- IE5 必须配合外部 CSS -->
<?xml-stylesheet href="ie5_needs.css" type="text/css" ?>
<测试 xmlns="http://put-your-URL-here"
      xmlns:html="http://www.w3.org/TR/REC-html40">

<!-- 借用 HTML 标签来连结 -->
<html:a href="http://2tigers.net">连到两只老虎</html:a>

<!-- 借用 HTML 表单来做按钮 -->
<html:form method="post" action="put-your-action-here">
  <html:input type="submit" value="按我" name="clickme" />
</html:form>
```

⁹根本不可能有，因为 XML 只不过是个超语记。

XLink :
<http://www.w3.org/xlink>
XPointer :
<http://www.w3.org/xptr>

Mozilla :
<http://www.mozilla.org>

```
<!-- Mozilla 也支援 XLink 旧标准中的 simple link 机制 (已过时) -->
<链结 xml:link="simple"
      href="http://2Ti.com">这样也可以连到 2T</链结>
</测试>
```

要请您注意的是，上例中用来代表 HTML 名称空间的网址不能写错。

学到这里，您应该已经能感受到名称空间所提供的强大功能。有了名称空间后，各元素、属性便可跨越文件的疆界，而不再只是单纯寄居在某套语汇的文件格式中；语汇也同时摇身一变，成为全球性的模块，可以和其他的语汇、模块任意组合、搭配，供各式各样的应用文件调借使用。许多重要的 XML 应用，包括 SVG^[1.3.3] 和 XSL^[7.1] 等，都是架构在名称空间的基础上。如果您用一个纯文字编辑器去瞧一瞧微软 Office 2000 所发布出来的 HTML 码，您会发觉它里面的 XML 部分用名称空间也用得很凶。



6 下一代的 HTML — XHTML

「HTML 4.0 已经推出好一段时间了。什么时候可以看到 5.0 出来？」

不会有第 5 版了！因为 HTML 的任务，已经交给 XHTML 来接棒了。

HTML 4.0:

<http://www.w3.org/>

[TR/REC-html40](http://www.w3.org/TR/REC-html40)

6.1 什么是 XHTML

XHTML 最早叫“HTML in XML”。简单讲，就是把过去以 SGML 定义的 HTML，改用 XML 来重新定义。XHTML 就像其他数不清的「X 叉叉 L」一样，是 XML 超语言的一项应用。所有 XHTML 的标签都摹拟既有的 HTML 4.0 标签来定义，各元素和属性^[2.2]的名称和用法几乎完全不变。不过因为变成了 XML，有些地方必须遵照 XML 的规矩，严格执行，不能再像过去那么随便。

XHTML 1.0 这套标准，在 1999 年 8 月 24 日升格为建议标准 (proposed recommendation)。本章便是依据这份最新的标准写成。不过成为建议标准，并不代表就肯定能顺利成为正式标准。事实上，XHTML 1.0 这份标准目前正被激烈地讨论中，争议的焦点之一是名称空间^[5]的解释问题。暂且不管最后的结果如何，我们可以先深入了解 XHTML 发展的动机，和它试图解决的问题。本书也会随着最新的动态，适时更新。

XHTML 1.0:

<http://www.w3.org/>

[TR/xhtml1](http://www.w3.org/TR/xhtml1)

6.2 XHTML 长得什么样子

照往例，先请您看一段 XHTML 源代码。在脑中有个概略的轮廓就好，我们会在后面一一详细解释。迫不及待的读者可以利用其中的色块，直接跳到相关的章节去一探究竟。

```
<?xml version="1.0" encoding="GB2312" ?> A
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
    "http://www.w3.org/TR/xhtml1/DTD/strict.dtd"> B
<html xmlns="http://www.w3.org/TR/html1"> C
<head>
<title>几只老虎才够看? </title> D
<meta http-equiv="Content-Type"
    content='text/html; charset="GB2312"' /> A
<script type="text/javascript"> E
<![CDATA[
function countTigers(n) {
    if (n < 2) return "Not enough tigers.";
    return "Enough tigers.";
}
]]>
</script>
</head>
<body background="#FFFFFF"> F
<p>一个非常短的段落。</p> G
<hr /> H
<form action="mailto:xml@2Ti.com" method="post">
<input type="checkbox" name="box" checked="checked"> I
</form>
</body> J
</html>
```

了解 XHTML 的各个特徵固然重要，但是先思索一个新科技是否具有实质上的意义，再去考虑学习的问题，可能会是比较理想的顺序。就让我们先来看看，XHTML 是否真有存在的价值。

6.3 为什么需要 XHTML

6.3.1 HTML 的隐忧

HTML 在可预见的未来，仍将像目前一样，在网络上扮演重要的角色。在未来的网络世界，不仅会存在大量的 HTML 文件，更会继续不断有新的 HTML 文件产生。但现在有一个问题存在——不晓得您知不知道，网络上大多数的 HTML 档案，严格来讲，都是不合

格的。

「哈！至少咱的肯定合格，因为我都是用那 DreamPage99 网页编辑器做的，而且都用 Netscape 和 IE 检查过，安啦！」

别太有把握哦！您的网页，相信用 PC 上的浏览器去看是绝无问题的，但如果用明日的大哥大来看呢？

「大哥大？」

对，没写错，是大哥大——欢迎来到后 PC 时代。

许多分析家预测，今后几年，会有愈来愈多的小型器具、和自动化使用者代表程序上网，为人类做各式各样的服务。有的甚至说，到了公元 2002 年，75% 的网页都将透过这些非桌上型电脑来看¹。

请别误会，我并不是在鼓吹说，PC 将被淘汰。完全不！桌上型 PC 将继续在未来扮演重要的角色。不同的是，将会有另一类完全不同的机器、异形，开始在网络上活耀起来。事实上，可以让人 check email 的大哥大早就有了，而且已经有厂商宣布，将在近期推出附网页浏览器的大哥大²。此外，从掌中型微电脑（像是最热门的 Palm Pilot）的大兴其道、宽频时代的来临，还有上网的电冰箱的出现，在在不难想像为什么分析家们会这么预测。

HTML 浏览器之所以愈做愈肥，除了互拼功能外，为了能尽量包容五花八门、邈邈含糊的 HTML 码，也是主因。因此，用两家浏览器能正确显示的网页并不代表它肯定符合标准。更可怕的是，可能连一些网页制作工具做出来的网页，都不见得合格。在 PC 称霸的今日，这不是很大的问题，因为我们有足够的记忆体和硬盘空间来饲养这些庞然巨物。但等不久的将来，当轻薄短小的网络工具、自动程序当家时，它们内建的网页阅读器、解析器是否还能像现在的大浏览器这样，把不够标准的网页，照网页设计者想表达的原意，顺利地呈现出来呢？

¹直接引用自 XHTML 标准。

²Nokia 最近便发表了未来大哥大的雏型，不但可收看数位电视，更内建 Linux 操作系统和 Mozilla 浏览器，让使用者遨游网络。

XML 和 XHTML 前来解救。

我们知道，XML 的标准非常简洁且严谨。从整份标准文件短短的三十多页这点上，就可看得出。「简而严」的特性，让工具不必做得那么肥，像各大公司出的 Java XML 解析器^[2.4]，大小都在几百 kByte 以下。这也是 XML 和 Java 为什么热门的原因之一，因为许多人预见网络小工具时代的来临。而瘦瘦的 XML 解析器，要安装在这些小工具上是再适合不过了。

6.3.2 XHTML — 既可轻薄短小，又可无限延伸

XHTML 不只提供我们一个环境，把不乾淨的 HTML 码清一清；事实上，可大可小、能缩能胀的弹性，才真正是 XHTML 的价值所在。

能缩 — 模块化

HTML 最早是个简单的语言，设计给 CERN 高能物理中心的同仁之间，交换研究心得之用。但继 WWW 风起云涌、两大浏览器互拼功能的连年征战，加上网页设计师、所见即所得编辑器为达视觉效果，各施奇招的百般折磨之后，不管是 HTML 标准本身，或写出来的 HTML 码，早已被整得面目全非。虽然 HTML 在网络的发展史上，有着不可磨灭的功劳，但如果不想想办法，后果不堪设想。

有什么办法可想？嗯，我们可以把庞大的 HTML 4.0 功能，按使用者需要和浏览器的能耐，打破、拆分成一块块的模块，每一组特定的功能，都由一个单独的 DTD^[8] 来负责定义。换句话说，就是把现有巨大的 HTML 4.0 DTD 模块化。每一组 DTD 只支援一部分的标签。对不同的浏览工具，可以针对其个别的显像特性、屏幕尺寸，设计出专用的 DTD；譬如不适合显示分割画面 (frames) 的小电脑，就不要支援包含 frame 功能的 DTD。这样一来，不但大大减轻这些小工具的负担，更让它们能尽情发挥各自的特性。

事实上，模块化正是目前 W3C XHTML 小组的工作重点。虽然在现行的 XHTML 1.0 标准中，只照本宣科地订出了三个 DTD（后面会谈到的），但是陆续发展的 XHTML 1.1

XHTML 1.1:
[http://www.w3.org/
TR/xhtml11](http://www.w3.org/TR/xhtml11)

草案，则已积极开始制订模块拼装的原则，让 XHTML 文件的编辑者和浏览器能按实际需要，套用不同组合的 DTD。

能胀 — 扩充性

XHTML 名称中的 X，可不是随随便便冠上去的，而是真正代表它具备「X」家族一脉相承的特性，也就是扩充性／延展性 (eXtensibility)。

过去在 HTML 时代，只有 W3C 标准会或极具影响力的浏览器业者（意译：微软和昔日的网景），才有办法给它添加新的标签，至于一般老百姓那想都别想。而且每定义一个新的标签，HTML DTD 就要更动一次，非常没有效率。

改用 XHTML 以后，扩充困难的问题不再存在。就像所有 XML 文件一样，XHTML 文件可以透过界定名称空间^[5]的方式，随时按需要来微调额外的标签。譬如在以下的 XHTML 源代码片段中，就借用了 [MathML^{\[1.3.1\]}](#) 来表达数学式子。

MathML :
<http://www.w3.org/TR/REC-MathML/>

```
<html xmlns="http://www.w3.org/TR/xhtml1">
  <head>
    <title>数学例子</title>
  </head>
  <body>
    <p>以下为 MathML 标签 :</p>
    <!-- 开始进入 MathML 名称空间 ... -->
    <math xmlns="http://www.w3.org/TR/REC-MathML">
      <apply> <log/>
        <logbase>
          <cn> 3 </cn>
        </logbase>
        <ci> x </ci>
      </apply>
    </math>
    <!-- 离开 MathML 名称空间 -->
  </body>
</html>
```

薪火相传

有的读者可能还是会问：「XHTML 固然能缩能放，但这只不过是它从 XML 继承的好处罢了。为何不全面改用新的 XML 标签，反而刻意去配合一个充满旧包袱的 XHTML？」

因为 XML 不会全面取代 HTML。有的文章，根本不需要大费周章地改用 XML 形式储存，而可以长久保持 HTML 形式；就算 XML 在将来会完全取代 HTML，今天占上风的，仍是 HTML。理由很简单：当今大多数网页设计者最熟悉的，仍是 HTML。此外，目前 XML 开发工具仍相当缺乏，也不够普及（这点不禁令人回想起几年前 HTML 工具不成熟的景象³），加上现今绝大多数的网页浏览器，仍然只懂 HTML，而浑然不知 XML 已诞生。至于要等新一代浏览器大量普及，则还要很长的一段时间。因此，不管是为了确保明日的 HTML 文件，不会让未来的迷你浏览器噎到；或是打算在这个标注语和浏览器新旧交替的过渡期，为转型到 XML 做准备，我们都可以善用 XHTML，来帮助我们达成这些目标。别忘了，XHTML 是当然的 XML。把 HTML 文件换成 XHTML 格式，便是确保未来任何替 XML 设计的网络器具，都能正确处理您的文件。这在网站未来化的脚步上，无疑向前迈进一大步。如果您正在设计一个新网站，更应优先考虑使用 XHTML。

6.4 XHTML 和 HTML 4.0 的差别

6.4.1 XHTML 帮您复习

在读过前几章介绍的 XML 语法和概念后，如果您到现在都还迟迟未动笔（或敲键盘），试着写几个 XML 标签来实际演练一番（不应该哦！），现在正是重温旧课的好机会。这也是为什么 XHTML 这个主题要刻意安插在这里的原因。

³尽管这么比喻，但二者的本质却不尽相同 — HTML 本身包含外观的呈现，因此所见即所得、亲善可人的编辑器就相形重要。虽然 XML 本身不考虑外观，但如果能有好的 CSS 编修工具来配合，则也是一大福音。

有了 XML 的基础⁴，再来瞧瞧 XHTML，您会发觉，一切 XHTML 的规定，刹那间变得很容易理解，丝毫不感意外，因为这些正是 XML 的规定。

好，就让我们从一个已经认识 XML 的人的角度来想想，如果 HTML 要 XML 化，会有哪些地方需要调整。

6.4.2 格式正确原则对 HTML 的冲击

我们在第 2.5 节曾谈到，XML 文件要称得上合格，必先格式正确 (well-formed)，也就是得满足 XML 的基本语法。由于 HTML 标准本来在某些地方就比较自由，加上各大浏览器容错度高，对 HTML 的句法不严格要求，造成目前的 HTML 文件中，有不少地方，在先天或后天上和 XML 的语法打架。其中最常见的包括：结尾标签被省略、两个标签交叉，和属性值没有上引号。这些违反格式正确原则的状况，在 XHTML 中一律不许出现，有几项规定要强制执行：

结尾标签不可省

HTML 标准中说，有些结尾标签爱加不加随便你，像 `</p>`、``、`</tr>`、`</th>`、`</td>`、`</dt>`、`</dd>` 等都是。但在 XML 和 XHTML 中，漏了结尾标签整份文件就不及格，非写不可，所以一切要像这样：

`<p>`这是一个段落。

下个段落虽空无一物，标签仍然要成双成对。

`</p>`

`<p></p>`

解开纠缠不清的标签

标签之间只许包含、但不许交叉是 XML 格式正确原则的另一项要求。严格讲，这并不能算是 HTML 和 XML 相冲之处，因为这在 HTML 中也不能算合格，只是各 HTML 浏览

⁴还有，假设您会用 HTML。

器对于标签交叉的做法，常睁一只眼闭一只眼，导致在许多现有的 HTML 码中，常可以见到像：

```
<i>斜体字<b>粗斜体字</i></b>
```

这样两个标签纠缠在一起的邈邈写法，结尾的 `</i>` 却跑到 ` ...` 里头去了。

XML 解析器^[2.4]可不吃这一套 — 要让 HTML 码成为合格的 XHTML/XML 码，我们必须反交叉、反打结：

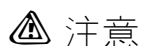
```
<i>斜体字<b>粗斜体字</b></i>
```

把 ` ...` 完全包在 `<i> ...</i>` 里面，而不是一个在里、一个在外。

空元素 XML 化

H

`<空元素 />`^[2.5.1]标签的写法，是 XML 中的新发明。在 HTML 中，空元素标签和其他标签的长像并无不同，唯一的差别是空元素只有起始，却没有结尾标签来和它配对，最常见的例子，不外乎 HTML 中的 `
` 和 ``⁵。为了满足格式正确的要求，在 XHTML 中，`
` 这类的空元素必须改写成 `
` 或 `
`。



注意

不要把上面提到的 `<p>` 和 `<tr>` 等标签和 `
`、`` 混为一谈。`<p>` 和 `<tr>` 不是空元素，只是大家写 HTML 时，常将结尾的 `</p>` 和 `</tr>` 标签偷懒不写。两者有区别。

属性值一律上引号

F

记得在第 2.5.3 节曾谈过，不管是字串或数字，XML 文件的属性值一律要用引号括起来。因此在 XHTML 中，`<td width=100>` 的写法不合格，要改成 `<td width="100">` 才能过关。

⁵`` 虽内附属性 (`src`、`height...`)，但仍算是空元素。

属性名不可落单

HTML 中有一种属性，它的值只有「是」或「不是」两种状态⁶。通常在遇到这类的属性时，如果我们要注明它的值是肯定的，就把它名称放进标签里，但省略后面的等号和属性值，像以下例子中的 `checked`、`selected`，和 `compact`：

```
<input checked>
<option selected>
<ul compact>
```

`<input>` 和 `<option>` 这两个标签，对常编 HTML 表单、CGI 程序的读者肯定都不陌生。这种属性值省略的做法，还特别有个术语，叫属性最小化 (attribute minimization)。

在 XML 一板一眼、要求一律使用「属性名="属性值"」的做法下，上面例子中被「最小化」的属性，必须一一改写为：

```
<input checked="checked">
<option selected="selected">
<ul compact="compact">
```

也就是把属性名拿来当属性值，再复颂一遍就是了。

6.4.3 检测、验证、依据

上面所说的各项格式正确的规定，不过是所有 XML 文件都必须共同遵守的一些最起码的要求。我们知道，在 XML 中，有法可证^[2.6] (validity) 的概念，常和格式正确相提并论。那么，XHTML 文件除了求格式正确外，是否也要有法可证呢？

那当然！

光是格式正确的文件根本不能算是 XHTML，充其量只是合格的 XML 文件罢了。XHTML 因为是直接利用现成的 HTML 标签，所以，就像我们过去写 HTML 文件时一样，在用 XHTML 写东西时，也必须确定文件中使用的标签名，都是确实存在的

⁶编程的读友都知道这叫 boolean。

HTML4.0/XHTML 标签，还有标签之间相互的关系、出现的顺序，都不能胡来。这正是有法可証的意义所在。

选定、注明 DTD

我们在第 2.6 节提过，有法可証的「法」，指的是 DTD^[8]，要让 XML 文件能透过某套 DTD 法则加以验证之前，我们得先指定一个合适的 DTD，然后在 XML 宣告中明白昭告世人：这份文件选用的是那个 DTD。这样好让 XML 解析器有个对照、验证的依据。

前面说到，目前的 XHTML 1.0 草案中，已有三种不同的 DTD 供人使用，未来随着 DTD 模块化、专门化的脚步，应该会有更多 DTD 出来。目前的这三种，只是摹仿 HTML 4.0 DTD 所定义出来的，包括：

Strict（严格式） 最早的 HTML 比较呆板，没有什么 `font`、`color...` 等花俏的东东。那些都是 Netscape 发明的。从标注语应该专注在文意的表达，而少碰外观呈现的观点来看⁷，W3C 认为 HTML 文件中动不动这里一小撮、那里一大坨的 `` 标签，看起来真是恶心极了！为了整治这个乱像，W3C 特别发明了 CSS 来跟它相抗衡。如果您的文件只用 CSS 来呈现，坚决不用 ``，那这是适合您的 DTD。

Transitional（过渡期式） `` 标签大受欢迎，W3C 在拗不过广大世人的渴求下，只得硬着头皮把 `` 补进 HTML 4.0 的标准中，但在说明文字中，仍不忘时时告诫世人，尽量多用 CSS。这是「过渡期」一词的由来 — 因为他们希望，随着有愈来愈多的浏览器支援 CSS，`` 标签会逐渐消声匿迹（嗯，恐怕没那么乐观）。如果您的网站，像大多数网站一样，使用到 `` 标签的话，则必须选择使用 Transitional 这个为「过渡期」设置的 DTD。

Frameset（分割画面式） 如果您酷爱使用 `<frame>` 的话，那这是您不二的选择 :-）。

⁷在这本书一开始就谈到过这个问题。这是正确的观点，更带动了 XML 的诞生。

选好 DTD 以后，就可以开始准备宣告了。在第 8.2 页中，我们谈到用

```
<!DOCTYPE doc SYSTEM "http://foo.bar/baz.dtd">
```

这样的方式来链结一个外部的 DTD，其中的 doc 是该 XML 文件根元素的名称，而 "http://foo.bar/baz.dtd" 则是 DTD 所在的网址。「SYSTEM "网址"」这部分合起来叫「外部标示」(External ID)。

至于上面介绍的三种 XHTML DTD，正确的 <!DOCTYPE> 宣告方式分别是：

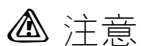
```
<!DOCTYPE
  html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/strict.dtd">

<!DOCTYPE
  html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/transitional.dtd">

<!DOCTYPE
  html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
  "http://www.w3.org/TR/xhtml1/DTD/frameset.dtd">
```

在这里，我们看到另外一种格式的外部标示，这次用的是 PUBLIC 这个表示「公用」的标示，最后面接的，仍是 DTD 所在的网址。

要特别提醒您的是，<!DOCTYPE> 宣告必须出现在根元素标签之前，否则等解吸器都已经看到根元素标签了，却还浑然不知这个元素是属于哪个类型、该去参考哪个 DTD，那就来不及了。



注意

以上列举的所有 <!DOCTYPE ...> 宣告，大小写都有特定的意义，写错一点点都不行。

试了才知道

很明显地，光表面上宣告，私底下却挂羊头卖狗肉、没有完全按 DTD 的规矩来做，是没有意义的。每份 XHTML 文件，都得和它选用的 DTD 核对无误后，才有资格算是正

W3C 的核对服务：
<http://validator.w3.org/>

牌的 XHTML。最简单的核对方法，是利用 [W3C 的核对服务](#)。您只要在这个网页中，填入您网页的网址，送出，就会看到结果了。这个时候，如果它显示的是：“No errors found! Congratulations...” 的讯息，而不是一堆红字，那么恭喜恭喜！这代表您制作出来的网页，非常的乾净清爽。另一个核对的方法，是直接在网页中建一个超连结，连到 <http://validator.w3.org/check/referer>。然后在浏览器中点击这个连结即可。

不过现在还不到测试的时候，因为还有几项规定要交代一下。

6.4.4 其他规定事项

格式正确和有法可証的问题都解决后，接下来的就好办了。

元素、属性一律小写

J

前面谈过，XML 像 Unix 系统和许多程序语言一样，是有区分大小写的 [\[2.7\]](#)。因为这个缘故，XHTML 不能再像 HTML 那样，标签大小写随你便，必须统一作个规定，而标准决定采用小写。因此，要转型到 XHTML，那些过去用大写字母来写的元素、属性名，都得先替换成小写才行。

使用正确的根元素加名称空间

C

XHTML 标准中规定，文件的根元素标签必须是 `<html>`，而且要用 `xmlns` 这个特别的属性来注明 XHTML 专属的名称空间 [\[5\]](#)，像这样：

```
<html xmlns="http://www.w3.org/TR/html1">
```

其中 <http://www.w3.org/TR/html1> 这个网址所连到的，正是 XHTML 1.0 的标准文件（`xhtml` 格式，不意外）。其他标注语的名称空间，也往往采用各标准文件所在的网址来代表。

前面谈到，名称空间的机制，是 XHTML 易于扩充^[6.3.2]的主要关键。在根元素的标签中标明了 XHTML 的名称空间后，我们便可依需要使用来自其他名称空间的标签，而不用担心同名的标签会打架了。

标明哪儿是头，哪儿是身

在 HTML 中，`<head>` 和 `<body>` 可以省略，但在 XHTML 中，一律得按规矩加在适当的位址。还有，`<title>` 必须是 `<head>` 中第一个出现的元素。

D

用 CDATA 区来包装 style sheets 和 scripts

HTML 码中常有 CSS 样规^[1.2] 或 JavaScript 程序直接内嵌在里面。这些样规和小程序分别出现在 `<style> ... </style>` 和 `<script> ... </script>` 区块中。在 XHTML 中，因为 `style` 和 `script` 两个元素的内容在 DTD 中被定义为 `PCDATA`^[p.108] 类别，它们的内容会被解析器^[2.4] 当作一般文字资料来解读。如果不巧，碰到样规或 `scripts` 中含有 `<` 或 `&` 这两个标注符号的话，解析器很可能就挂在那里了。

E

要避免解析器去解读文件中的某些资料，最好的办法是把这些资料放进 `CDATA` 区^[2.8] 中。因此，当内嵌的 CSS `style sheets` 或 JavaScript 码中含有这些在 XML 中具有特殊意义的标注时，XHTML 标准说要用 `CDATA` 区把它们包起来，像这样：

```
<script language="JavaScript">
<![CDATA[
.....
if (i < 3 && .....
]]>
</script>
```

反应快的读者马上要问：「有必要那么麻烦吗？XML 中不是也有注解吗？而且写法也和 HTML 中一样。那为什么不干脆保持过去在 HTML 中的做法，把程序放到注解中不就结了？如下」

```
<script language="JavaScript">
<!--
.....
    if (i < 3 && .....
//-->
</script>
```

想得好！不过这么做有个缺点：-(。XML 1.0 标准说，解析器没有义务将注解^[2.3]内容原封不动地交给下游的程序。换句话说，注解内容是否到得了浏览器手中还得看解析器高兴而定。但 CDATA 区就不一样了——这部分解析器不可以随便略掉。因为这个缘故，XHTML 1.0 才规定要用 CDATA 区，而不告诉大家可以用注解来做。

6.5 XHTML — 到底给谁看

经过检测、验证的 XHTML 文件便是不折不扣的 XML 文件，因此未来所有的 XML 软件应该都读得通。

「那过去的 HTML 浏览器怎么办？」

问得好。换一个角度来问：「那到底写好的 XHTML 文件的副档名要取作 .xml 还是 .html（或者是 .xhtml）？」

其实不只是副档名，读者中的站长 (webmaster) 们都知道，服务器端还要有适当的 MIME Content-Type：设定相配合才行；相对于 .xml 的 MIME type 是 text/xml，而 .html 对应的则是 text/html。

XHTML 文件可以按需要而设成这两种类型其中一种，即：

- .xml 对应到 text/xml
- .html 对应到 text/html

但是如果打算设成 .html，而且要让 XHTML 文件能被旧的 HTML 浏览器顺利解读的话，XHTML 1.0 的标准中提出以下几个注意事项⁸：

⁸在这里只择要摘录。

- 空元素结束的地方，`</>` 之前，要多加一两个空白，否则 HTML 浏览器会看不懂，也就是应该用：`
`、``，和 `<hr />` 这样的写法。

- 注明文字编码时，XML 中的 `encoding="..."` 和 HTML 中惯用的 `<meta http-equiv ...>` 这两种机制，都要使用，也就是：

```
<?xml version="1.0" encoding="GB2312" ?>
.....
<meta http-equiv="Content-Type" content='text/html; charset="GB2312"' />
```

别忘了，`<meta>` 是空元素，因此标签结尾时要写成像上面那样。

- 有些浏览器会将 `<?xml version="1.0" encoding ... ?>` 这个 XML 专用的起始宣告 [2.1] 秀在网页的顶端。如果文件中含有其他的 PI [2.10]（即 `<? ... ?>`），也会被这些浏览器秀出来。丑丑的，但没有办法。
- 如果网页中的 CSS 样规或 `<script>` 中含有 `<`、`&`，或是 `]]>` 字串时，必须先把它改存为外部样规或 `<script>`，再以链结的方式呼叫，因为老浏览器看不懂 CDATA 区 [6.4.4] 的写法。
- 在 HTML 中以最小化 [p.69] 方式表达的属性（如 `<input type="radio" ... checked>`），在经过 XHTML 「反最小化」 [28] 处理（也就是变成了 `<input type="radio" ... checked="checked">`）之后，会让一些 HTML 浏览器看不懂。而在 100% 符合 HTML 4.0 的浏览器（如 Mozilla）中，应当不会有这种问题产生。这些最小化的属性共计有：`compact`、`nowrap`、`ismap`、`declare`、`noshade`、`checked`、`disabled`、`readonly`、`multiple`、`selected`、`noresize`、`defer`。

Mozilla :
<ftp://ftp.mozilla.org/pub/mozilla>

6.6 从何下手

读到这里，您可能会感叹：「唉，这么多项，听起来好麻烦喔！谁有这么多闲工夫把过去的 HTML 档案一个个动手修改？有没有什么工具可以自动帮我检查、修改的？」嘿，有的。W3C 早已想到这个问题了，而且还送我们一个免费工具，它还有个可爱的名字，叫“HTML Tidy”。

好工具

HTML Tidy，顾名思义，是一个帮我们把不乾淨的 HTML 源代码清一清的实用工具。Tidy 一如我在前面为大家提供的 [ccnv](#) [p.38] 程序，是个 Unix 调调的命令介面工具。这是一个 C 程序，支援多种平台，它的 homepage 和下载点在：
<http://www.w3.org/People/Raggett/tidy/>

6.6.1 HTML Tidy 使用方法

要把中文的 HTML 档案用 Tidy 自动转成 XHTML，可以下这个命令：

```
tidy -raw -asxml -f 错误讯息档名 输入档名 > 输出档名
```

其中 `-raw` 是用来告诉 Tidy，不要去乱动档案中的中文码，而 `-asxml` 是叫它把档案转成 XHTML 格式。如果您只是想用它来帮您清理一下 HTML 码，而不想转成 XHTML 的话（为什么不？），就不需要使用这个选项。因为 Tidy 预设的输出部是标准输出 (standard output)，所以输出档名前的那个 `>` 号可别忘了，要不然输出会全数吐到屏幕上。

如果执行了 Tidy，结果发现输出档是空的，这个时候很可能是您的 HTML 码中有严重的错误，Tidy 无法处理。请您参考它的错误讯息，看到底是在第几行的地方有错，更正后再重新跑一次 Tidy。

顺利得到输出档，并不代表一切完全符合标准。如果您的档案中用到了标准不支援（但某浏览器支援）的标签、属性，错误讯息中会一一列举出来。不过要注意的是，您可能会碰到很多标示为“warning”的讯息，这些大部分只是 Tidy 的「强烈建议」（最常见的像：`` 标签应该要附上 `alt` 属性），并不是严重的错误。

要确定文件 100% 符合标准（也就是 DTD），可以按照前面建议的检测方法[\[p.71\]](#)，用 W3C 的核对器来查一下。

以下列出中文使用者最可能用到的几个参数选项，给大家作参考：

参数选项	说明
<code>-raw</code>	不要去动 ASCII 以外的字码
<code>-utf8</code>	输入字码为 UTF8 [3.2] 时，可用此选项来替代 <code>-raw</code>
<code>-asxml</code>	产生形式正确 [6.4.2] 的 XHTML 档
<code>-f</code> 档名	把错误讯息存到指定的档案里
<code>-m</code> (-modify)	直接把修改过的内容，存回输入档内，取代原始内容
<code>-c</code> (-clean)	用 CSS 的方式改写 <code><center></code> 和 <code><nobr></code> 这两个没有被标准接受的标签
<code>-h</code> (-help)	列出所有支援的选项

6.7 最后

本章对 XHTML 作了非常详尽的解说，涵盖了 XHTML 1.0 草案中 90% 以上的要点，甚至还提供了一些草案中没有的额外信息。

建议您回到本章刚开始的地方，利用那里的[源代码范例](#)，帮助您一一回想在这章里学到的新概念，让印象更深刻。



7 XSLT — XML 专属的转换语

本章的重点是 XSLT。要谈 XSLT，得先从 XSL 谈起。

7.1 另一种样规 — XSL 简介

XSL (eXtensible Stylesheet Language) 是专门为 XML 设计的样规^[1.2]语，也是在 CSS^[4.1] 之外，另一个替 XML 穿戴打扮的选择。

一如 MathML^[1.3.1] 和 SVG^[1.3.3]，XSL 本身便是一项 XML 的应用，直接架构在 XML 的语法之上。它一共分作两部分：第一部分负责将 XML 源代码转换为另一种格式，而第二部分（称作“FO”〔Formatting Object；打样物件〕）则提供了大量的打样命令，可用来配合印刷或屏幕显像，精确地设定外观式样（譬如字的大小、摆放的位置等），是一种所谓“device-independent”的格式。第一部分的转换语法，可以用来为第二部分服务，将 XML 文件变形为打样命令。

有趣的是，XSL 的转换语法，并不限于将 XML 转成 FO 命令。事实上，XSL 可以输出任何格式正确^[2.5]的 XML 文件。因为这个特性，我们可以用它来做以下几种形式的转换：

XML → HTML 这是最常见的一种转换。转换出来的 HTML 档，在格式上因为已达到格式正确的要求，在本质上非常接近 XHTML^[6]。因为目前大多数浏览器仍不支援 XML，所以 XML → HTML 的转换，在这个过渡时期有很大的利用价值。此外，由于 IE5 对 XML 加 CSS 的支援还不够完善，如果在 IE5 中想把美化过的 XML 文件打印出来，唯一的方法是用 XSL 先转 HTML。用 CSS 来设定外观的 XML 文件

在 IE5 中只能在屏幕上看，无法印。

XML → XML 把一种格式的 XML 文件转换成另一种 XML 格式，有非常大的实用价值。例如：两个机关或企业之间以 XML 来传递信息，如果彼此使用的 XML 格式有出入，便可藉着 XSL 来做调整。

XSL → XSL XSL 甚至可以将一个样规转换成另一种形式的样规。因为 XSL 本身便是一项 XML 的应用，所有的 XSL 样规自然也都是 XML 文件。因此，XSL → XSL 在本质上不过是 XML → XML 的一个特例而已。

7.1.1 XSL 和 CSS 不同的地方

XSL 采用的是转换的方式，将一种格式的 XML，转换为另一种。就好比将 Big5 码繁体中文，转为 UTF-8^[3.2] 码一样。CSS 则来自完全不同的理念：它不含任何转换动作，只针对 XML 文件中各个成分的外观属性，一一加以设定。浏览器便按照 CSS 样规里的指示，将 XML 文件呈现为设定的式样。整个过程中，没有任何新码产生。

DOM:

<http://www.w3.org/>

DOM

XML 配上 CSS、ECMAScript 和 DOM 可以营造出类似 DHTML 般的动态效果。XSL 转换则是死的，没有互动性。

另一个区别是：XSL 样规都是 XML 文件，完全照 XML 的语法来；相对地，CSS 在语法上自成一格，和 XML 的写法大相径庭。

7.1.2 XSL 和 CSS 相同的地方

XSL 和 CSS 都属于样规^[1.2]的一种。样规是用来设定外观的，它并不影响原来的 XML 源代码。XSL 虽然用的是转换的方式，但「转换」并不代表源代码会遭到篡改。通常 XSL 转换后的输出码，是另存到一个新的档案、或暂存在浏览器的记忆体中，原来的 XML 档案内容则保持不变。

7.1.3 XSL 简史

XSL 历经非常坎坷而戏剧性的发展。不但草案经过一而再、再而三的大幅修改（尚未成为正式标准的东东难免是这样），发展过程中更招致不少批判和反对的声浪。XML 专业人士之间，甚至曾经发生拥护和反对两大阵营相互叫阵、「笔」锋相对的热闹场面，足见 XSL 的争议性。

XSL 最具争议性的，是 FO 打样物件的部分。反对者主要的论点是，XML 文件经过 XSL、尤其是 FO 这么一转的结果，输出的新码中，很可能语意^[1.1.2]尽失，原本一个像 <姓名> 的标签，取而代之的将是一些冷冰冰的打印命令，XML 最重要的精神和价值所在，也随之荡然无存。相较之下，CSS 并没有这样的问题。不意外，反对 XSL 的人士多半也都为 CSS 的拥护者。

争议性之外，事实上，一部分也因为 FO 的复杂度，大部分软件，包括很早便开始支援 XSL 的 IE5 和 LotusXSL，也都不支援 FO。为了推广 FO，负责草拟的 Adobe 公司，甚至曾经使用「重赏之下，必有勇夫」的策略，悬赏两、三万美元，鼓励网络上各方高手，积极开发 FO → PDF 的转换程序。不过最后这个竞赛因故取消。现在支援 FO 的软件依旧寥寥可数。

W3C 也感受到各方对 XSL 的压力。在 1999 年 4 月份，W3C 正式将 XSL 标准的前半部，关于转换语法的叙述，从 XSL 中分出来，并为它起了一个新名子，叫 XSLT，T 代表“Transformation”，也就是「转换」、「变形」的意思。自此之后，XSL 的草案只剩下 FO 的部分。FO 的发展，已经显得迟滞，目前动向不明，和日新月异的 XSLT，形成强烈对比。XSLT 在 4 月到 10 月之间，又作了一些变革。在一波多折的发展、历经 6、7 个版本的草案后，XSLT 现在总算熬到 W3C 建议标准 (proposed recommendation) 的地步（还有可能再作修改）。至于它是否能顺利成为正式标准 (recommendation)，答案很快就会揭晓，且让我们拭目以待。本书也会跟随最新的动态，适时更新。

注意

由于 XSL 草案像个变色龙一样，一改再改，导致绝大多数的教学文件和最新标准之间，出现一段落差，而在 1998 年 12 月之前写的文章，更是严重脱节。有个简单的方法可以看出 XSL 的教学文件或书籍是否已经过时 — 如果在文章中看不到 XSLT、XPath 这些名词的话，那么这篇文章可能就已经有点旧了；如果连 `<xsl:apply-templates/>` 这个命令都见不到，那更是严重落伍了。在此建议初学者对过时（但或许仍具参考价值）的 XSL 文件，最好先尽量避免，以免在新旧标准之间，搞得头晕脑胀、事倍功半。

7.2 XSLT 入门

我们在接下来的章节中，将把焦点集中在 XSLT。XSL 打从一开始，就只有这部分在发烧，比较令人瞩目。绝大多数软件支援的，也只是这部分。在下面，当用到“XSL”这个名词时，指的也大都是 XSLT。

7.2.1 XSLT 在网络上的应用模式

让我们来看看在目前的条件下，XSLT 能透过哪几种方式来替我们的网站效劳。依 XSLT 转换动作发生的地点来分析，可分为：

server 端 XML 文件在下载浏览器之前，先以 XSLT 转为 HTML；依转换动作发生的时间，又可分成：

动态即时产生 在浏览器向 server 送请求的当下，以 XSLT 处理器软件将 XML 立即转换为 HTML，即刻送出。在网页中有动态性资料要即时插入、或当 XML 文件是从数据库即时取得的场合下，最适合这种运作方式。负责输

出 XML 的数据库软件，则很可能存在于另一台机器上，以 XML 来向负责 XML → HTML 转换的 server 传递资料。这是现在最热门的三层式、n 层式 (3-tier, n-tier) 的 server 构筑方式。大型数据库网站，或大企业的 intranet、extranet 网站，较适合这种模式。因为 XSLT 的运行步骤相当繁琐（后述），所以访客量大的网站，为了兼顾快速转换的要求，必须使用高档的机器，甚至可能需要（一批）专责 XSLT 转换的 servers。

批次产生 如果 XML 文件事先都已写妥，一一存放在档案里，而非即时取得，或随时不断更新，我们便可用批次转换的做法，事先将 HTML 档准备好。转换的工作则可依实际需要，在固定的时段自动化执行。这么做的好处是，server 资源可获得最大的节约，更不需要使用高性能的 server 机，任何能跑 XML 解析器和 XSLT 程序的电脑都行，甚至可以在个人家中的 PC 上来做，转换好了以后再把 HTML 档上传到 server 即可。

client 端 如果浏览器直接支援 XML 和 XSLT，XML 源代码和 XSLT 样规便可以像图档一样，让浏览器直接下载下去，一切转换工作由浏览器代劳。这是最理想的状况，但目前唯一支援 XSLT 的浏览器是 IE5，而不使用 IE5 的仍大有人在；更何况，IE 5 对 XSLT 的支援本来就不完整，有些部分更已过时，因此让浏览器直接转换的做法就目前而言实用性仍相当有限。不过这个美景在未来还是有希望实现的——微软已经承诺将在未来补上目前缺漏的 XSL 功能；而新一代的 Netscape 浏览器（预计 1999 年底推出），则也可能支援 XSLT，因为已经有人将 XSLT 程序捐献给 Mozilla 计画。

双管齐下 融合以上 server 和 client 端两种运作模式，在浏览器请求网页的同时，根据浏览器的身分即时决定是否将 XML 先转换为 HTML，或直接交由支援 XSLT 的浏览器下载。server 甚至可以更进一步，按照各个 client 软件的特性，产生不同性质的 HTML，让访客享受最佳的阅览效果，譬如支援 CSS 的浏览器，便可以配合

下载美美的 CSS 样规，如果碰到不支援 CSS 的浏览器（或是支援得太差的，像 Netscape 4），就给它含有 标签的网页。如同第一种运作模式（即时转换），这种做法需要较高的技术水平，甚至需要重金添购专业软件，更少不了高档的硬件作后盾。

7.2.2 XSLT 的转换流程及工作原理

XSLT 转换必须由特别的软件来担任，这样的软件，通称为 **XSL 处理器** (XSL processor)。如同我们在第 24 页中谈到的，任何 XML 文件，不管要作什么用途，必先经过解析器解析，整理出条理后，才能进一步利用。既然 XSL 样规和它所要转换的 XML 源代码都属于 XML 文件，自然也需要解析。因此，XSL 处理器在工作之前，得先借重 XML 解析器（内建或外部），替它把 XSL 样规和 XML 文件中一个个物件和结构分析出来。请看图 7.1。

在 XSLT 中，有两个重要的观念，一个是 **源树** (source tree)，指的是转换前的 XML 文件的结构；另一个是 **结果树** (result tree)，代表转换的成品，结构依然是 XML。

XSL 处理器起动时，会先叫 XML 解析器替它解析 XSL 样规和要转换的 XML 文件。在掌握了样规中的各个 XSL 命令后，XSL 处理器便开始依照样规的指示，在源树上爬来爬去，一遇到合适的枝叶，就按样规的设定，吐出一段新枝叶（一个 XML 片段），一直到整颗树都走遍了为止。有的时候，样规会指示处理器，将枝叶先修整一下再吐出来。所有经由 XSL 处理器一路产生的新枝叶，统统加到一块儿，就成了一株结果树。结果树可以输出到档案里储存起来，或由浏览器显像在屏幕上。

这么解释，可能您对实际转换的细节还是很模糊。别担心！在稍后的章节中，我将带您去「爬树」，实地检视 XML 树的枝节。目前只需要掌握一个大概的轮廓就好。

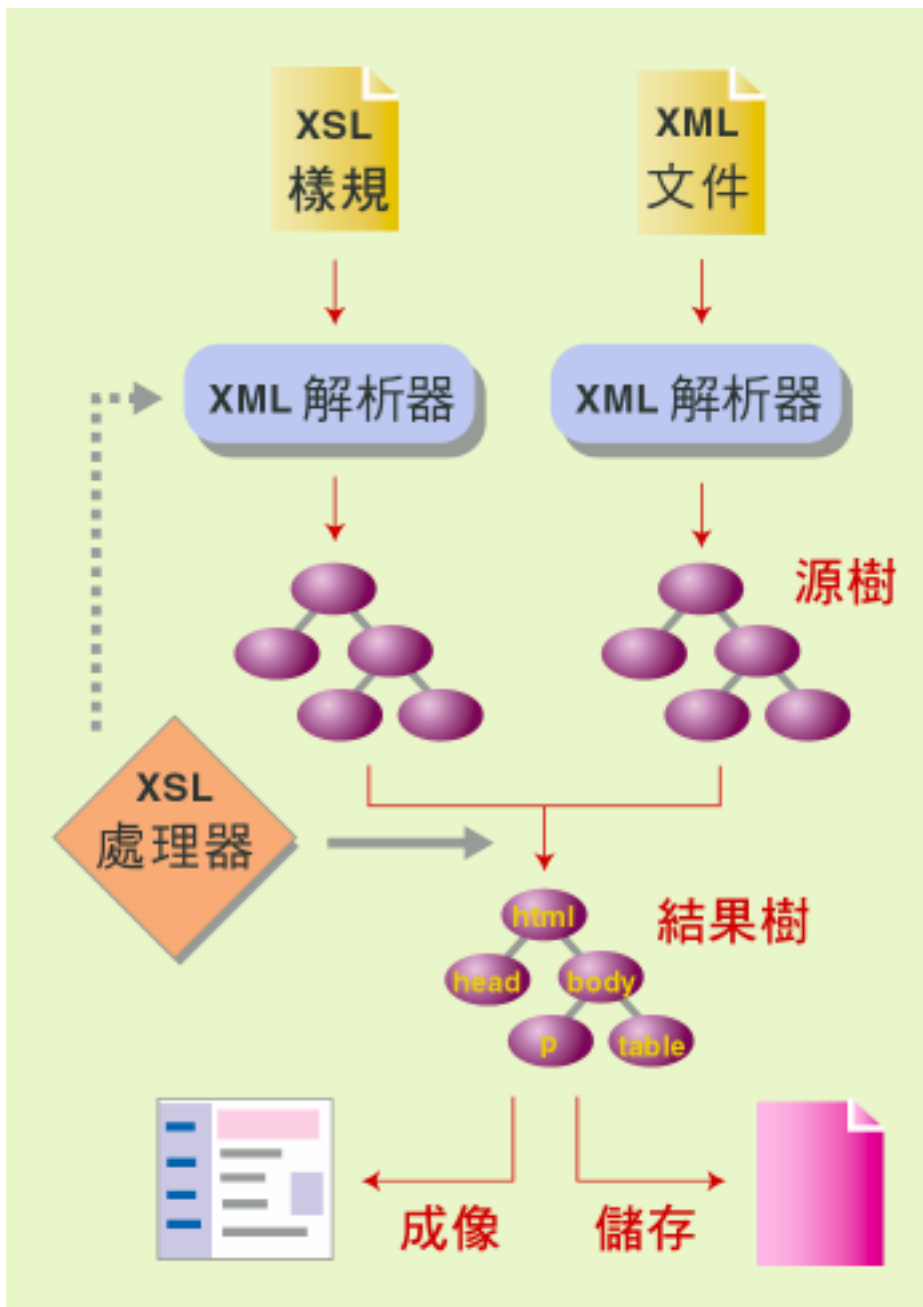


图 7.1: XSLT 的工作流程

7.2.3 支援 XSL 的软件

支援 XSL 的软件，可粗分为浏览器和转换工具两大类。目前比较热门的浏览器中，只有微软的 IE5 局部支援一个旧版的 XSL 草案（1998 年 12 月 16 日版）。在本章的结尾处，我们将会以 IE5 作实例演练。

在转换工具方面，绝大多数的软件都是以 Java 写成的，包括由 IBM AlphaWorks 实验室所出品的 LotusXSL，和 XSLT 标准的作者 James Clark¹所写的 xt。

LotusXSL：
<http://alphaworks.ibm.com/tech/LotusXSL>
xt：
<http://www.jclark.com/xml/xt.html>

LotusXSL 和 xt 都支援最新版的 XSLT 标准。xt 除了不断随着标准草案快速更新外，自 19990822 版开始支援以 Big5、GB2312 等亚洲文字码直接输出 HTML²。

在此要向读者推荐，在处理 XSL 时，尽量使用 xt 作为转换工具。Clark 先生所写的 XSLT 处理器和 XML 解析器³，在性能、速度、正确度，和对标准的支援度上，都是竖大拇指、一级棒的，和同类型的商业软件相比，往往有过之而无不及。更可贵的是，他的软件都是以程序码公开 (Open Source) 的方式发行。

其他支援 XSL 的软件还很多，无法在此一一介绍，有兴趣的读者请到 W3C 的网站去瞧瞧：<http://www.w3.org/Style/XSL/>。

7.3 细谈 XSLT

XSLT 的语法和运行方式有些诡异，可能需要一些时间才能渐渐适应；而且必须以抽丝剥茧的方式，一小步一小步学习，如果一心求快，生吞活剥，很可能落得败兴而归。先请您看一下 XSLT 源代码到底长得什么样子，然后我再针对其中的观念和细节，一一为您剖析。

¹和创建 SGI、Netscape 企业家 James Clark 同姓同名，但一个是英国人，一个是美国人。

²在旧版的 xt 里，所有经过处理的中文字，只能以 UTF-8 形式输出，因此如果要输出码转回 Big5/GB2312，需要再多跑一道转码手续。这个情况在输出 XML 时依然存在，但是要 patch 不难。一等 XSLT 标准正式通过、xt 发展告一段落后，如果情况依旧，我会为大家写一个 patch，让 xt 可以直接以 Big5/GB 形式输出。过去曾针对 1999 年 3 月 7 日版的 xt 写过 patch，但已过时，不能再用。

³包括鼎鼎大名的 Expat 解析器（以 C 写成，速度极快），被 Mozilla 浏览器和 perl、Python 的 XML 模块选作内建解析器。

7.3.1 成品预览

假设有一家电脑硬件制造商，在网站中提供产品搜寻的服务。他们的数据库已经进步到能以 XML 形式输出。以下是一笔摹拟的搜寻结果：

```
<?xml version="1.0" encoding="GB2312" ?>
<?xsl-stylesheet type="text/xsl" href="search_result.xsl" ?>
<产品搜寻>
  <摘要>搜寻字符串：“滑鼠 键盘”，共找到 2 笔</摘要>
  <产品>
    <货号>12478943</货号>
    <品名>手不痛健康滑鼠</品名>
    <定价>$234</定价>
    <说明页 网址="http://foo.bar/mouse/12478943">上市发表会</说明页>
  </产品>
  <产品>
    <货号>83424723</货号>
    <品名>打不响静悄悄键盘</品名>
    <定价>$567</定价>
    <说明页 网址="http://foo.bar/kbd/83424723">产品特性</说明页>
  </产品>
</产品搜寻>
```

透过 XSLT，我们可以将上面的 XML 码，转换成下面的 HTML 码，以表格 (table) 来呈现搜索到的产品信息：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=GB2312">
<title>产品搜索结果</title>
</head>
<body>
<h1>产品搜索结果</h1>
<p>
<b>摘要：</b>搜寻字符串：“滑鼠 键盘”，共找到 2 笔</p>
<table>
<tr>
<th>品名</th><th>定价</th><th>说明页</th>
</tr>
<tr>
<td>手不痛健康滑鼠</td><td>$234</td>\continuearrow
  <td><a href="http://foo.bar/mouse/12478943">上市发表会</a></td>
</tr>
```



图 7.2: XSLT 转换出来的 HTML 档，在浏览器中呈现出来的样子

```

<tr>
<td>打不响静悄悄键盘</td><td>$567</td>\continuearrow
  <td><a href="http://foo.bar/kbd/83424723">产品特性</a></td>
</tr>
</table>
</body>
</html>

```

以上的 HTML 码为 [xt_{\[p.86\]}](#) 实际的输出结果。用浏览器来看，大致会像图 7.2 那样。

整个转换过程真正的主角，是下面的 XSLT 码。为了方便阅读起见，XSLT 命令的部分，一律以蓝色标示。

```

<?xml version="1.0" encoding="GB2312" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> A
<!-- IE5 只接受
  <xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"> -->

<!-- IE5 看不懂 xsl:output -->
<xsl:output encoding="GB2312"/> B

```



```

<xsl:template match="/"> C D
  <html>
  <head>
  <title>产品搜寻结果</title>
  </head>
  <body>
    <h1>产品搜寻结果</h1>
    <p><b>摘要 : </b><xsl:value-of select="*/摘要"/> E</p>
    <xsl:apply-templates select="产品搜寻"/> F
  </body>
  </html>
</xsl:template>

<xsl:template match="产品搜寻"> D
  <table>
  <tr>
    <th>品名</th>
    <th>定价</th>
    <th>说明页</th>
  </tr>
  <xsl:for-each select="产品"> G
  <tr>
    <td><xsl:value-of select="品名"/></td>
    <td><xsl:value-of select="定价"/></td>
    <td><a href="{说明页/@网址}"> H <xsl:value-of select="说明页"/></a></td>
    <!-- IE5 只接受 <td><a><xsl:attribute name="href">
      <xsl:value-of select="说明页/@网址"/>
    </xsl:attribute><xsl:value-of select="说明页"/></a></td> -->
  </tr>
  </xsl:for-each>
  </table>
</xsl:template>
</xsl:stylesheet>

```

7.3.2 XSL 样规与名称空间

如同其他许多 XML 的应用，XSLT 大力仰仗名称空间^[5]所提供的机制，把 XSLT 命令和其他 XML 标注隔开。样规里所有的 XSLT 命令，都必须标明是隶属于“<http://www.w3.org/1999/XSL/Transform>”这个 XSLT 专用的名称空间，才能正常工作。

7 XSLT — XML 专属的转换语

在宣告 XSLT 的名称空间时，最常见的做法是以“xsl”作为前置字串^[5.3.1]：

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
```

依规定，所有 XSLT 命令都必须置于 `<xsl:stylesheet>...</xsl:stylesheet>` 区块里。也就是说，`xsl:stylesheet` 这个元素界定了 XSLT 样规的内容；因为它是最外层的元素，名称空间的宣告，自然也就摆在这个元素的标签里。在 1999 年 10 月 8 日版的 XSLT 标准中，`xsl:stylesheet` 又新增加了一个不可省略的属性：`version`，用来标示样规所遵循的 XSLT 语汇版本。在先前的标准草案中，版本号码是直接附在名称空间网址中的。

在接下来的章节中，我们将按照惯例，所有的 XSLT 命令，一律使用 `xsl` 来作前置字串。但就如同我们在第 5 章所提到的，前置字串是任人自订的，如果您不想用 `xsl` 这三个字母，大可在宣告中改用其他任何前置字串，包括中文，来代表 `http://www.w3.org/1999/XSL/Transform` 这个 XSLT 专属的名称空间。譬如以下的写法，是百分之百合乎规定的，但是这样做，有没有实质上的意义，就由各人自由心证吧：

```
<我高兴:stylesheet xmlns:我高兴="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
  <我高兴:template match="/">
    .....
  </我高兴:stylesheet>
```

7.3.3 源树分解

「产品搜寻」这份 XML 文件，在 XSLT 的转换过程中，会先被 XML 解析器解析。所得出的物件结构，即为前面所提到的源树^[7.2.2]，如图 7.3。图中每个椭圆形，都代表一个节点 (node)。就像真实世界中的树木一样，枝叶是从节点处发出去的。位于末梢的物件叫叶节点 (leaf node)，因为叶子上通常不会再长出枝干。在 XML 物件树中，所有代表文字内容的节点（那些土黄色的椭圆），肯定都是叶节点。

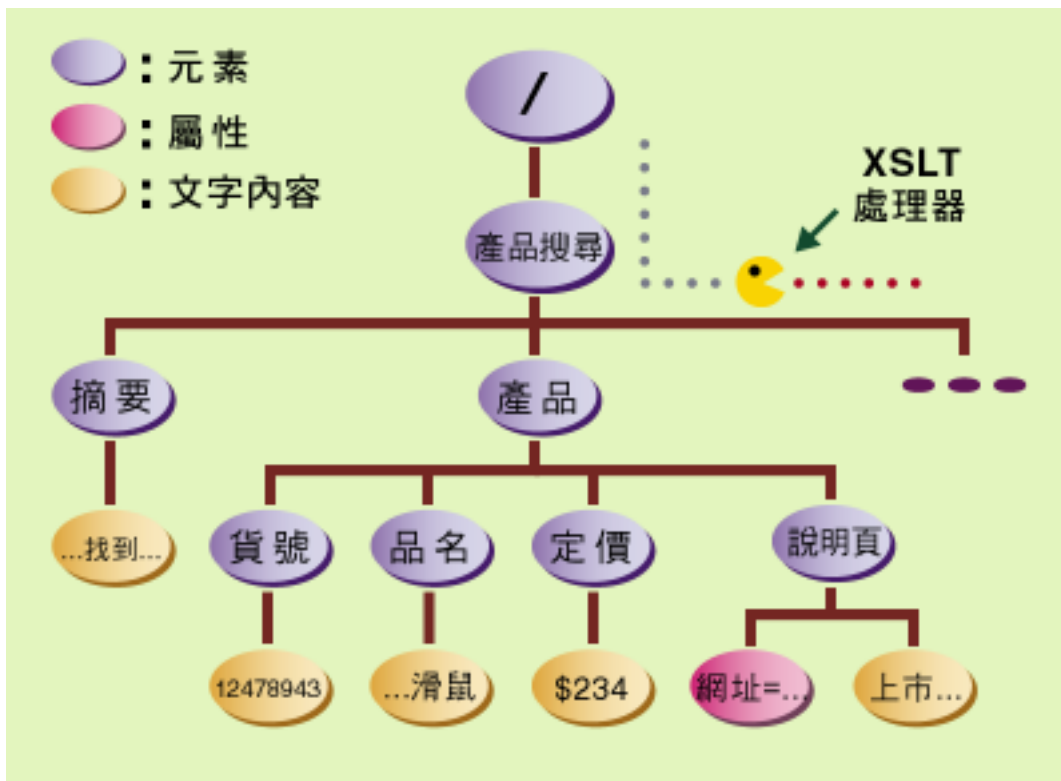


图 7.3: 「产品搜寻」这份 XML 文件，经由 XML 解析器所分析出来的物件树，椭圆形的颜色代表 XML 的元素类型。

在接下来的各个章节中，我们将随着 XSL 处理器的脚步，在这颗树上爬来爬去。建议您在阅读的时候，脑中随时掌握这颗源树的结构和成分，以达到最高的学习效果。您可能需要再回来多瞄这个图几眼。

W3C 的 DOM:

<http://www.w3.org/>
DOM

其实图 7.3 中的树状结构，大致上就是按照 W3C 的 DOM（文件物件模型）所分析出来的样子。

7.3.4 根元素上头还有一级

C

虽然「产品搜寻」在 XML 中叫做根元素^[2.2]，但人外有人，天外有天，在 XSLT 中，斜线符号“/”要当作是最高层，「产品搜寻」次之，如图 7.3。熟悉 Unix 或 DOS 目录结构的读者应该不难想像。

7.3.5 XSLT 运行细节

模板和对应式

D

前面第 7.2.2 节只粗略地描绘了 XSLT 运作的流程。现在我们要谈整个 XSLT 转换的工作细节。前面说到，XSL 处理器按照样规里的设定，在源树上找寻合适的节点^[7.3.3]，并适时产生新枝叶。这个「样规里的设定」，在 XSLT 中的正式称呼叫「**模板式**」(template rules)。XSL 处理器就是根据这些式子来寻找节点、产生新码的。看个实例：

```
<?xml version="1.0" encoding="GB2312" ?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output encoding="GB2312"/>
<xsl:template match="/">
<html>
<p>一个很空洞的模板，不怎么来劲</p>
</html>
</xsl:template>
</xsl:stylesheet>
```

这是一份完整的 XSL 样规，里面只有一个模板式。我们看到，模板式以 `xsl:template` 这个元素来定义，该元素所包含的内容，不意外，正是 XSLT 中所谓的「模板」（`template`；黑色文字部分）。`xsl:template` 里面有个 `match="..."` 的属性^[2.2]，XSL 样规正是藉着这个属性，来告诉 XSL 处理器，该去找什么样的节点。在这个例子中，模板式指定对应的节点是 `/`，也就是在源树最顶层的根元素。

XSL 处理器在起动机时，会先将所有列在样规中的模板式看过一遍，并牢记在心，然后便开始检视源树，而且肯定先从 `/` 下手，每对应到一个节点，就把相关的模板式中的模板抄一份出来，加进要输出的文件里。

因为任何一份 XML 文件，在分析成树状结构之后，「树根」都肯定是 `/`，而上例中的 XSL 样规，又刚好只含一个指名要对应到 `/` 的模板式，所以这份样规不管拿来转换什么样的 XML 文件，甚至像这样一份简陋至极的单行文件：

```
<my_xml>^_^</my_xml>
```

所产生的结果都必定是：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<p>一个很空洞的模板，不怎么来劲</p>
</html>
```

事实上，`match` 的属性值不限于使用确切的节点名（像上面的例子），其实它是一个「对应式」（`pattern`），有一套完整的语法，可用来选择样规里各个模板式所对应的节点，后面会进一步介绍。

还有要请注意的一点：模板的内容，必须是格式正确^[2.5]的 XML 片段。为什么？记得在第 7.2.2 节中曾经谈到，XSL 样规的设定内容，是先经过 XML 解析器解析，然后才交给 XSL 处理器的。因此，如果样规里有任何一段违背格式正确的原则，整个转换过程会在 XML 解析器那关就宣告失败，XSL 处理器根本连边都摸不到。换句话说，不仅是模板式所包含的区块，而是整份 XSL 样规，都必须是格式正确的 XML。如果模板里头含有 HTML 码，则务必要按照第 6.4.2 节的建议，让一切都「达到及格标准」。像常常被省略的 `</p>`（结束段落）标签，就非添上不可。

现节点和语境

XSLT 处理器会爬树，从这个节点爬到那个节点（至于该怎么个爬法，我们将在第 7.3.5 讨论）。每到一个新的节点，XSLT 处理器执行命令的语境 (context) 就跟着改变。这个「新到的节点」，在 XSLT 中的术语叫 **现节点** (current node)，即处理器当时所在的节点。这个概念很像在 Unix/DOS 命令列中执行 cd 命令来改变当时所在的目录，新到的目录就成了一切相对路径依循的标准。

以第 7.3.1 节「产品搜寻」的例子而论，因为 XSLT 处理器会从 / 根元素开始下手，而样规中又有一个对应到 / 的模板式 (`<xsl:template match="/">`)，处理器于是以 / 作现节点，开始处理这个模板式中的设定。

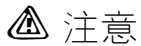
现节点应用实例 — xsl:value-of

E

XSL 模板中的内容，除了能让 XSL 处理器一五一十地照抄、输出之外，还可以透过 XSLT 命令，对资料作排列、组合等处理（否则岂不太无趣了）。对模板的内容，XSL 处理器会先看，如果其中有隶属于 XSLT 名称空间[7.3.2]底下的元素，就把它当成是命令，如果它看得懂这个命令，便会做适当处置。举例来说，在「产品搜寻」的 XSL 样规里有这一段：

```
<xsl:template match="/">
  <html>
  <head>
  <title>产品搜寻结果</title>
  </head>
  <body>
    <h1>产品搜寻结果</h1>
    <p><b>摘要 : </b><xsl:value-of select="*/摘要"/></p>
    .....
  </body>
  </html>
</xsl:template>
```

其中的 `<xsl:value-of ... />`，就是一个常用的 XSLT 命令，它通常是用来读取源树中某元素所包含的文字内容，或是某个属性的值。



注意

`<xsl:value-of ... />` 是一个空元素^[2.5.1]。后面那道斜线 (/) 可别漏了。

`<xsl:value-of ...>` 标签中肯定要附上 `select` 这个属性。如同 `<xsl:template>` 中的 `match` 属性，在 `xsl:value-of select="..." />` 里，`select` 属性后面接的，也是对应式 (pattern)，而这次这个 `*/摘要`，看起来真的像是个式子了。XSLT 对应式在对应的时候，以现节点作相对位置的轴点，它的语法，和 Unix shell、DOS 模式中的档案、路径表示法有些类似。在这个式子中，星号 `*` 会对应到位于现节点下一层的所有元素节点；换句话说，就是所有现节点所在元素的子元素。从图 7.3 我们可以看出，`/` 就只有「产品搜寻」一个子元素，因此，`*/摘要` 式子中的 `*`，在现节点停留在 `/` 的语境下，只会对应到「产品搜寻」。

我们的式子中还有一条斜线 `/`，这在对应式语法中是用来分隔节点层级的，就像 `/` 在网址和 Unix 系统中用来界定目录一样。合起来，`*/摘要` 会对应到一个名叫“摘要”的元素节点，它的母节点刚好是现节点的子元素节点。

在我们的例子中，因为对应到的节点是一个元素，XSLT 处理器会将 `<xsl:value-of select = "*/摘要" />` 整段命令代换为源树中「摘要」这个元素底下的文字内容，即「搜寻字符串：“滑鼠 键盘”，共找到 2 笔」这段文字。因此，XSL 样规中的这段：

```
<xsl:template match="/">
.....
  <h1>产品搜寻结果</h1>
  <p><b>摘要 : </b><xsl:value-of select="*/摘要" /></p>
.....
```

会被 XSLT 处理器转换成：

```
.....
<h1>产品搜寻结果</h1>
<p>
```

```
<b>摘要 : </b>搜寻字串 : “滑鼠 键盘” , 共找到 2 笔</p>
.....
```

这样的结果。

语境转移

附图 7.3 将 XSL 处理器比喻为古老电玩中的小精灵 (pacman ; 我小时候流行的 game) , 会在源树上各个节点间跑来跑去。但是到目前为止 , 我们的语境一直停留在 / 。要让小精灵 , 嗯 , XSL 处理器吃遍整颗树 , 最常用的技巧是改变语境 , 让现节点层层下移。有两个 XSLT 命令 — `xsl:apply-templates` 和 `xsl:for-each` — 可以用来达到这个目的 , 转移现节点的位置。

模板套入 — `xsl:apply-templates`

F

在我们的范例中 , 对应到 / 的模板里只剩下一个命令没交代 :


```
<xsl:template match="/">
  <html>
  <head>
  <title>产品搜寻结果</title>
  </head>
  <body>
    <h1>产品搜寻结果</h1>
    <p><b>摘要 : </b><xsl:value-of select="*/摘要" /></p>
    <xsl:apply-templates select="产品搜寻" />
  </body>
  </html>
</xsl:template>
```

在讨论 `<xsl:apply-templates select = "... " />` 之前 , 先来回顾一下 XSL 处理器已经扫过的部分 , 以及运行的流程 :

1. XSL 处理器先读入所有的模板式 , 包括上面这个。
2. XSL 处理器于是开始扫描源树 , 先从由根元素 / 看起。

3. 检视所有的模板式，看看有没有模板式对应到 / 。
4. 发现一个可用的模板式，开始处理其中的内容。
5. 将 `<html>` 一直到 `摘要：` 这部分照本宣科地抄到结果树上。执行 `xsl:value-of`，把 `` 和 `</p>` 之间的部分以 XML 源代码中，「摘要」一元素的文字内容填入。

接下来比较有看头了：`<xsl:apply-templates select="产品搜寻" />` 中有一个 `select` 对应式，整个命令是在告诉 XSL 处理器：「把 `select` 对应到的各个节点——当成是现节点，并依序处理这些节点的模板」。当 `<xsl:apply-templates>` 中的 `select="..."` 属性省略时，XSL 处理器则会以所有现节点的子元素作为处理对象。

 注意

`<xsl:apply-templates ... />` 在这里是以空元素^[2.5.1]形式出现。后面那道斜线 (/) 可别漏了。还有，`apply-templates` 中的 `templates` 是复数，后面有 `s` 别忘了。

`<xsl:apply-templates .../>` 的出现，让转换流程起了变化。XSL 处理器在这里转弯，先去执行其他的模板式，并且将产生的结果由 `<xsl:apply-templates .../>` 这点插入，一直到所有对应到的模板式都执行完毕，最后才回来执行原来模板中剩下的部分。而因为其他模板中可能含有更多的 `<xsl:apply-templates .../>` 命令，所以整个流程是计算机学上所谓的 **recursive**（递归？）的执行方式。如果 XSL 样规设计得不好，则可能出现无穷执行的窘境。

了解 `xsl:apply-templates` 的意义之后，我们可以继续追踪 XSL 处理器的脚步：

6. 遇到 `<xsl:apply-templates .../>`，检视所有的模板式，看看有没有刚好对应到「产品搜寻」的模板式可以使用。

7. 发现一个可用的模板式，如下：

```
<xsl:template match="产品搜寻">
  <table>
    <tr>
      <th>品名</th>
      <th>定价</th>
      <th>说明页</th>
    </tr>
    <xsl:for-each select="产品">
      <tr>
        <td><xsl:value-of select="品名"/></td>
        <td><xsl:value-of select="定价"/></td>
        <td><a href="{说明页/@网址}"><xsl:value-of select="说明页"/></a></td>
      </tr>
    </xsl:for-each>
  </table>
</xsl:template>
```

8. 处理模板的内容，将 `<table>` 一直到第一个 `</tr>` 部分照抄进结果树，从 `<xsl:apply-templates .../>` 处插入。

9. 遇到 `<xsl:for-each ...>` 命令，暂停，且听下节分晓 ;-)。

xsl:for-each 回圈



`xsl:for-each` 是另一个会改变现节点和语境的命令。和 `xsl:apply-templates` 一样，`xsl:for-each` 后面也有一个 `select` 对应式，用来选择新的现节点。在 XSLT 中，`select` 对应式所对应到的节点不见得只有一个，而可能有好几个；如果不只一个，XSL 处理器会按照它们在 XML 源代码中的顺序来一一处理。譬如前面例子中的 `<xsl:for-each select="产品">`，用在第 7.3.1 节的「产品搜寻」XML 源代码上，一共会对应到两个「产品」元素。XSL 处理器于是按照它们在文件中出现的顺序，

先把现节点设为第一个「产品」（滑鼠），处理完了以后再把现节点转移到第二个「产品」（键盘）上。

我们现在可以继续上一节中未完的流程，把剩下的部分交代完：

9. 遇到 `<xsl:for-each ...>` 命令，现节点转移到第一个「产品」元素处，除了将 `<xsl:for-each ...> ...</xsl:for-each>` 区块中所有写死的 HTML 元素依序照抄进结果树中外，每当遇到 `<xsl:value-of ... />` 时，XSL 处理器会以源树中所对应到的文字内容——分别是「手不痛健康滑鼠」、「\$234」，和「上市发表会」——来代换这些 XSLT 命令。至于模板中 "{说明页/@网址}" 的部分是下一节的主题，暂时不去管它。
10. 执行完一轮 `xsl:for-each`，处理器绕回 `xsl:for-each` 起始处，把现节点转移到第二个「产品」（键盘），如法泡制，依序处理其中的内容。
11. 两个轮回之后，要转换的 XML 源代码中没有更多的「产品」元素可处理，「产品搜寻」的模板式因告执行完毕。处理器于是回到原来的切入点，也就是根元素 / 的模板式中 `<xsl:apply-templates .../>` 的地方，继续处理后面的部分。
12. / 的模板中只剩下 `</body>` 和 `</html>` 这两个写死的标签，XSL 处理器于是将它们照抄到输出结果中，大功告成！

`xsl:for-each` 是个常用的技巧，因为 XML 文件中，往往包含反覆规律出现的元素（譬如像「产品搜寻」中的「产品」），然而出现的次数，往往不固定（每次搜寻的结果不尽相同）。而正因为我们在范例的样规中用了 `xsl:for-each`，不管搜寻结果出现几笔「产品」，我们的样规都可以不经修改，继续正确地运行。这个重要的特性是任何成功的 XSL 样规所必须具备的。

7.3.6 { 属性值模板 }

前面说到，当样规中的元素隶属于 XSLT 的名称空间时（也就是我们的例子中，以 `xsl:` H

起头的那些元素)，XSL 处理器会把这些元素当成命令去执行。除此之外，样规里还有一个地方会引起 XSL 处理器的注意：XSLT 名称空间以外元素的属性值。譬如我们的范例中有这么一段：

```
<td><a href="{说明页/@网址}"><xsl:value-of select="说明页"/></a></td>
```

在这里，`<td>`、`<a>` 这些 HTML 元素并不属于 XSLT 的名称空间，而当这样的元素里面，有被大括弧 `{ }` 括起来的属性值时（上例蓝色部分），XSL 处理器会把大括弧中的区域，当作是 **属性值模板** (attribute value templates) 来处理。我们可以把属性值模板想成是一个迷你模板，里面可以包含第 7.3.5 节中谈到的对应式。XSL 处理器会用类似处理 `<xsl:value-of select="..." />` 的手法来解读属性值模板中的对应式，并且作代换（如果式子对应得到东西的话）。在这个范例中，属性值模板是「说明页/@网址」，里面有一个前面不曾介绍过的对应符号 — `@`，这个小老鼠在 XSLT 中是用来对应属性名的，像这样：`@`要对应的属性名。因此合起来，「说明页/@网址」会对应到现节点底下，一个叫「说明页」子元素的「网址」属性。

前面说过，当对应式对应出来的结果是元素的话，XSL 处理器会以该元素底下的文字内容来作代换。不过这次的式子对应到的不是元素，而是属性，这种情况下，XSL 处理器会以该属性的属性值来代换。因此，

```
<td><a href="{说明页/@网址}">.....</a></td>
```

会分别被转换成：

```
<td><a href="http://foo.bar/mouse/12478943">上市发表会</a></td>
```

和

```
<td><a href="http://foo.bar/kbd/83424723">产品特性</a></td>
```

7.3.7 输出文字码设定

B

1999 年 8 月 13 日版的 XSLT 草案新增了一个 `xsl:output` 命令，专门用来设定输出

码的各项特性，譬如字码、缩排 (indentation) 等。xsl:output 中有一个 encoding 的属性，可以用来控制输出的字码。视各人需要，可以把它设为 Big5、GB2312，或 UTF-8 等。

7.4 XPath 路径描述语

我们在前面只介绍了几个简单的对应符号，如 * 和 /。事实上，XSLT 对应式[7.3.5]的语法相当繁复而完整，光是这部分的语法，便多得足以分出去，自成份单独的标准。而这份标准的确已经出现，它的名子叫 XPath 路径描述语。

XPath 的目的是提供 XSLT 和 XPointer (XML 文件内部连结语) 一个共同、整合的对应语法，用来对应 XML 文件的各个部位、选择文件中的构成元件 (元素、属性、文字内容 ...)。表 7.1 列举了一些常用的对应符号及应用实例，希望您不会看得过分眼花缭乱。这些只不过是一部分而已，还有不少没列出来。XPath 除了提供一套对应语法之外，还包括了一些函数，提供基本的数字运算和字串处理功能。

XPath :

<http://www.w3.org/TR/xpath>

7.5 换您了一 实作演练

总算快轮到我休息了 :-)。看归看，但电脑语这东西，如果没有自己实际操作，绝对学不扎实。在此建议您以 xt 作为您的 XSL 处理器。如果您要用 IE5 的话，请务必参考第 7.5.2 节的各注意事项。

7.5.1 xt

第 7.2.3 节中介绍过、由 XSLT 设计人 Clark 先生所写的 xt，无疑是目前 XSLT 软件中的佼佼者，不但对标准支援最好、最正确，更支援 Big5/GB 中文输出。本章中的范例，便是以 xt 为准。以下简介它的安装及操作方式。

xt :

<http://www.jclark.com/xml/xt.html>

对应式	说明
某元素	对应到现节点下所有名为「某元素」的子元素
*	对应到现节点下所有子元素
* / 某元素	对应到自现节点开始算起、所有名为「某元素」的孙节点
@某属性	对应到一个附属于现节点、名为「某属性」的属性
@*	对应到所有附属于现节点的属性
text()	对应到现节点的子元素中所有文字节点
.	对应到现节点
..	对应到现节点上一级
某元素[1]	对应到现节点下，第一个叫做「某元素」的子元素
某元素[position()=1]	作用同上
某元素[@某属性="某值"]	对应到现节点下所有名为「某元素」的子元素，这个子元素必须含有一个叫「某属性」的属性，其属性值必须为「某值」
元素甲 元素乙	对应到现节点下，所有名为「元素甲」和「元素乙」的子元素； 代表「或」的关系
./后世子孙	对应到现节点底下，所有名为「后世子孙」的元素；//符号代表可跨越数级
祖宗//徒孙	对应到所有名为「徒孙」的元素，它们的上级必须有一个叫「祖宗」的元素，而且「祖宗」还得是现节点的一个子元素

表 7.1: XPath 路径描述语中的对应简式；取材自 W3C 标准

Java 虚拟机

xt 是个 Java 程序，因此先决条件是系统上必须有执行 Java 的环境，也就是所谓的 Java 虚拟机。关于各平台 Java 虚拟机的选择及下载点，请参考我在 [ccnv \[p.38\]](#) 安装说明中的建议。

选定、下载解析器

xt 只是个 XSL 处理程序，本身未内建任何 XML 解析器，因此安装 xt 之前，系统上必须先装有用 Java 写的 XML 解析器才行。选择 XML 解析器是一项非常重要的工作。市面上的 Java XML 解析器虽不下十几种，但真正能兼顾正确度、性能，又支援中文的寥寥可数。在此要向您推荐 Sun 的 XML parser，它可自 <http://developer.java.sun.com/developer/products/xml/> 免费下载（但必须先注册为 Java 程序发展会会员）。Sun 的 parser 不但支援 Big5/GB，而且据 xml.com 的测试，正确度在所有同类产品中排名最高，可说 100% 符合 XML 1.0 标准的规定。另外一个支援中文、虫不多的 Java XML parser 是 IBM 的 xml4j (XML for Java)，它可以从 <http://alphaworks.ibm.com/tech/xml4j> 下载（很烦，也要注册，而且在填表时如果用的是自己真正的 email 地址，又没有仔细看清楚，还会招来一堆垃圾邮件）。

下载 xt

xt 可自 <ftp://ftp.jclark.com/pub/xml/xt.zip> 下载。

安装 — 设定类别路径

装过 Java 程序的读者都知道有一个 CLASSPATH 的环境变数要设定。在下载并解压您最中意的 XML 解析器和 xt 后，只要将其中的 .jar 档 (xt.jar、xml.jar、

7 XSLT — XML 专属的转换语

xml4j.jar ...) 的安装路径加进您系统的 CLASSPATH 环境变数即可。

用批次档简化命令

xt 是个命令列的程序，叁数有些冗长，如果每次执行时都打一长串，太没效率了。这个问题可以用一个 Unix shell script (或 DOS 批次档) 轻易解决。底下以 Unix 平台的 shell script 为例：

```
#!/bin/sh

# 设定 Java 解译器安装的路径
JAVA="/usr/local/java/bin/java"
# 使用 Sun 的 XML parser :
PARSER="com.sun.xml.parser.Parser"
# 如要使用 IBM xml4j 2.x :
#PARSER="com.ibm.xml.parsers.SAXParser"
# 您也可以在此设定 CLASSPATH :
# export CLASSPATH="$CLASSPATH:/path/to/xml.jar:/path/to/xt.jar"

$JAVA -Dcom.jclark.xsl.sax.parser=$PARSER com.jclark.xsl.sax.Driver $*
```

请自行调整其中的路径设定。如果您使用 Windows 平台，则可以写一个像这样的批次档 (.bat)：

```
@ECHO OFF
SET JAVA=java

REM 使用 Sun 的 XML parser :
SET PARSER=com.sun.xml.parser.Parser
REM 如要使用 IBM xml4j 2.x :
REM SET PARSER=com.ibm.xml.parsers.SAXParser
REM 您也可以在此设定 CLASSPATH :
REM SET CLASSPATH=%CLASSPATH%;/path/to/xml.jar;/path/to/xt.jar

%JAVA% -Dcom.jclark.xsl.sax.parser=%PARSER% com.jclark.xsl.sax.Driver %1 %2 %3
```


使用方法

一切都设定妥当后，只要在命令列下这个命令（假设上述的批次档档名为 `xt`，而且安装在系统执行档路径中）：

```
xt XML源代码档名 XSL样规档名 输出档档名
```

一共是三个注明档名的参数。假设您的设定一切正确，几秒钟后，转换出来的结果应该就会被存在输出档档名 这个您所指定的新档案里了。如果没得到输出档，却看到错误讯息，那就是 `xt`、XML 解析器，或 Java 虚拟机没有安装、设定妥当。

您可以按需要尽情地发挥创意，将上面提供的 `shell script` 加以修改（譬如加入回圈，或配合 `find` 命令作子目录横扫、穷尽执行），甚至配合 `cron` 来定期执行。

7.5.2 迁就 IE5

如前面第 7.2.3 节所提，因为 IE5 是根据一份过时的 XSL 草案所设计，而且对该份草案也只有部分支援，所以为了让本章的范例顺利在 IE5 中运行，XSL 样规中有几个地方必须做调整。这些地方都已经以注解标示在范例里。以下扼要说明修改的重点。

名称空间的差异

IE5 用的是旧标准草案中规定的名称空间 — <http://www.w3.org/TR/WD-xsl>，但是自 XSLT 从 XSL 中独立出来后，指定的名称空间目前已改为 <http://www.w3.org/1999/XSL/Transform>，因此要让 IE5 能正确处理样规，必须先修改 `xsl:stylesheet` 元素中所宣告的名称空间，使用旧的名称空间。

IE5 不支援 `xsl:output`

`xsl:output` 这个命令是在 1999 年 8 月的草案中才加进去的，因此不仅 IE5，绝大多数 XSL 软件都还不支援这个命令。要让 IE5 读取的 XSL 样规不可含有 `xsl:output`，

您得先把范例中这行删掉或注解掉。

IE5 不支援属性值模板

属性值模板[7.3.6]并不是一个新功能，但很遗憾地，IE5 内建的 XSL 处理器就偏偏漏了支援这项方便的功能。在 IE5 中，要达到类似范例中 {说明页/@网址} 所产生的效果，必须改用较冗长的写法，以 `xsl:attribute` 和 `xsl:value-of` 相互搭配。请参考成品预览[7.3.1]中有关这部分的注解，并将

```
<td><a href="{说明页/@网址}"><xsl:value-of select="说明页"/></a></td>
```

这段改换为注解中的写法，即（实作时应写成一行）：

```
<td><a><xsl:attribute name="href">↵  
  <xsl:value-of select="说明页/@网址"/>↵  
</xsl:attribute><xsl:value-of select="说明页"/></a></td>
```



8 DTD — XML 语汇的定义

8.1 消除对 DTD 的恐惧

我们在第 2 章曾谈到，在有些场合里，光用格式正确^[2.5] (well-formed) 的 XML 文件还不够，必须配合 DTD 来清楚定义文件的格式。这样在资料自动化互传的过程中，我方的 XML 解析器^[2.4]可以根据 DTD，即时确认所收到的资料格式是否正确无误，如果有问题可以马上打回去，请对方程序重传，或请对方填资料的人修正后重送。这样就不会因无意中输入格式错误的资料，造成资料败坏 (corrupt)，误犯数据库管理上的大忌。

许多刚接触 XML/SGML，或者曾经读过 W3C HTML 标准的人，在初次看到 DTD 似无条理般的语法时，心凉了半截，从此以后不想再看到这个鬼东西（OK，或许我夸张了点;-），接下来我们要从 DTD 中理出一些头绪，降低莫名的恐惧。

前面使用过的 <推荐书籍> 一例（附表 8.1），只是一份格式正确^[2.5] 的 XML 文件，

```
<?xml version="1.0" encoding="GB2312" ?>
<推荐丛书>
  <书籍>
    <名称>煞死你的网页设计绝招</名称>
    <作者>胭脂虎</作者>
    <售价 货币单位="新台币">590</售价>
  </书籍>
  <书籍>
    <名称>如何在 7-11 白吃白喝</名称>
    <作者>无名氏</作者>
    <售价 货币单位="新台币">120</售价>
  </书籍>
</推荐丛书>
```

表 8.1: 线上书店 XML 实例

就让我们拿它作练习，编个 DTD 吧！

8.1.1 成品预览

```
<?xml version="1.0" encoding="GB2312" ?>
<!ELEMENT 推荐丛书 (书籍*) > A
  <!ELEMENT 书籍 (名称, 作者+, 售价*) > B
    <!ELEMENT 名称 (#PCDATA)>
    <!ELEMENT 作者 (#PCDATA)>
    <!ELEMENT 售价 (#PCDATA)>
    <!ATTLIST 售价
      货币单位 ( 新台币 | 人民币 | 港币 | 美元 ) '新台币' > C
```

8.2 元素类别宣告

A

我们先从文件中的最小的子元素着手，采取「由下到上」的策略来分解这份文件的结构。在附表 8.1 中，一共有 <名称>、<作者>，和 <售价> 三个不含更小的子元素的元素。用 DTD 中的类别宣告，可以把这三个元素定义为：

```
<!ELEMENT 名称 (#PCDATA)>
<!ELEMENT 作者 (#PCDATA)>
<!ELEMENT 售价 (#PCDATA)>
```

很明显地，ELEMENT 之后放的是元素名，接着是它的「内容模型」，说成白话，就是用抽象的表示法，定义 <元素> xxx </元素> 之间 xxx 的区域可以出现什么样的内容。按规定，这个「模型」要用小括号括起来。

#PCDATA 是 XML 中预先指定好的标记，代表 Parsable Character Data，即「可解析的文字资料」。意思是说里面的文字内容可以让解析器 (parser) 去解读，因此如果内容中有 <、>、& 等这些保留给 XML 语汇使用的符号时，要把它们「跳脱」、改写成 <、>、和 &，就像在 HTML 中一样，否则解析器会「挂」在那儿。

以上三个元素的内容模型都不怎么来劲，里头只能有单调的文字内容。事实上，XML 元素中可以包含混合式的内容 (mixed content)，也就是既可以包含子元

素，又可包含文字内容（即 #PCDATA），这种混合内容的抽象模型就比较精彩、有看头些了。

在接着定义其他元件之前，我想先介绍几个「量子」符号。

8.2.1 量子学

DTD，一如在正规表现式中，有所谓的量子（quantifier；或称「量词」）最常用的有「?」、「*」，和「+」这三个。量子是用来修饰它前面的那个单元的。怎么说呢？譬如我们说「黄车万两」，这个「万两」，正是一个从后面来修饰的量词。在 DTD 中，用量子表示数量一律要使用这种形式。问号 (?) 代表「可以有零个或一个」，换句话说就是顶多只能有一个；星号 (*) 代表有几个都可以，也就是可以有零到无限多个；加号 (+) 则表示至少要有有一个，但没有上限，像星号一样，可多到无限个。



注意

DTD 中的「?」、「*」、「,」，和「+」都是 ASCII 符号（半形），不要误用了中文的全形符号。

8.2.2 出现次序

现在回到元素定义的课题上，最小的几个元素解决了以后，我们可以往上走，开始定义上一层的元素，也就是 <书籍>：

B

```
<!ELEMENT 书籍 (名称, 作者+, 售价*) >
```

我们在这里用逗点和量子定义了 <书籍> 下属的三个子元素出现的顺序，和容许的数量。上回我们说到，当一本书的作者有两人以上时，该怎么表示的问题。有了 DTD 后，我们便可对此明确加以规范。

遵照以上的元素宣告，当我们遇到 <作者> 不只一个的情形时，可用：

```
.....  
<作者>比尔·盖兹</作者>  
<作者>知名不俱</作者>  
.....
```

的方式，来标注第二、三、四...作者。这正是上面的宣告中，在「作者」后边放一个加号的精髓。还有请注意，它同时规定，一本书不能没有作者，否则就会像对「售价」这个子元素的定义一样，用星号而不用加号。

「那为什么售价一项，就可有可无，而且可以容许两种不同的售价？」

因为一本书可能还没正式推出，价格尚未决定；此外，一本书可以用不同的货币单位来分别定价。当然，这些都只是我个人的看法，您可以依照您的需求和喜好，订定适当合理的网络书店 DTD。

我们还剩下一个元素要定义，那就是最上层的 <推荐丛书>。因为 <推荐丛书> 中可以包含很多笔 <书籍>，也可能一笔都没有（统统不值得推荐），所以我们就把它定为：

```
<!ELEMENT 推荐丛书 (书籍*) >
```

8.3 属性类别宣告

C

在附表 8.1 中，我们看到在 <售价> 一项中，有一个「货币单位」的属性，在 DTD 中，属性是用 <!ATTLIST ... > 来做宣告。ATT 就是属性、attribute 的简写。整个宣告看起来是这样的：

```
<!ATTLIST 售价  
          货币单位 ( 新台币 | 人民币 | 港币 | 美元 ) '新台币' >
```

最重要的是「货币单位 ...」这行（称作「属性定义」）共有三个要件。很明显地，第一个是属性名，第二个是属性类别，最后是预设值或预设行为的描述。如果属性不只一个的话，这三个要件单元可以每三个三个这样一直重复下去。在这里，我故意用换行

和缩排把属性定义突显出来。其实要统统挤到前一行也成（不建议），每个单元之间只要有一个空白字元隔开就够了。

属性定义有很多种花样，在这里，我们选用的是适合我们的条列式表示法，把所有可能用到的属性值——条列出来，而且不能重复。直线记号“|”代表「或」。没有列出来的值一律不许在 XML 文件中使用。属性值列举完后，我们要提供一个预设的值。当我们偷懒、把「货币单位」这个属性省略掉，而只写：

```
<售价>120</售价>
```

的时候，处理 XML 资料的程序会自动把它看成是：

```
<售价 货币单位="新台币" >120</售价>
```

8.4 统统放到一起 — 文件类别宣告

上面的元素、属性宣告——设计好后，我们可以开始把它们整理起来，做成一个完整的 DTD，并且把它和我们先前的 XML 文件（附表 8.1）连结在一起。有两种连结方式可以使用，一是外接，一是内嵌。如果用外接式，我们只要将刚才写好的 DTD，存到一个副档名为 .dtd 的纯文字档中（譬如叫 book.dtd），再配合：

```
<?xml version="1.0" encoding="GB2312" ?>
<!DOCTYPE 推荐丛书 SYSTEM "book.dtd" >
```

这样的链结方式，就可以了。我说「链结」，是因为如果这个 DTD 档不在同一台机器的同一个目录底下，则必须明确标明网址，而不能只写档名、路径，否则会找不到。

或者，我们可以把 DTD 直接内嵌在 XML 文件中。这需要用到 <!DOCTYPE ... >，即「文件类别宣告」的标注，写法上有点 CDATA 区的调调，像这样：

```
<?xml version="1.0" encoding="GB2312" ?>
<!DOCTYPE 推荐丛书 [
    <!ELEMENT 推荐丛书 (书籍*) >
    .....
]>
<推荐丛书>
    .....
```

8.5 结语

DTD 的语法相当繁杂，在此无法一一详细解说。以上的介绍不过只是浅尝，让您在阅读 DTD 文件时不至于丈二和尚，摸不着脑袋。

其实光是利用格式正确^[2.5]的 XML 文件，不碰 DTD，便已经可以大量运用 XML；譬如以 XSLT 来做格式转换，往往并不需要 DTD。

DTD 自成一格的语法，可说是 SGML 时代的「馊毒」，而不是 XML 中的新发明。DTD 的一大致命伤，在于无法理想支援名称空间^[5]的机制。在第 2.6 节中曾提到，目前正发展得如火如荼的 XML Schema（组织架构）标准，正是设计来取代 DTD，而且将像 RDF^[1.3.5]、SVG^[1.3.3]、XSLT^[7.2] 等应用那样，直接使用 XML 语法，因此直接继承了许多 XML 文件的优势，例如：方便搜索的特性^[4]。XML Schema 除了完美支援名称空间，让 XML 标注可以透过各名称空间 URI^[5.2.2] 所连结的语汇^[p.29]定义来作检测、验证^[2.6]之外，还有许多比 DTD 更强大的功能，包括资料类型 (data types) 的定义，对 SQL 数据库软件提供非常重要的支援。难怪 IBM 和 Oracle 等数据库业者无不卯足了劲，全力为 Schema 标准催生。从目前的发展脚步看来，Schema 可望在几个月内成为正式标准。

XML Schema :
<http://www.w3.org/TR/xmlschema-1>