無廢話 XML

勞虎

插圖：胭脂虎、勞虎

兩隻老虎工作室 www.2tigers.net

聯合光纖通信公司 www.ufoc.com.tw · 熱情贊助

本書由作者以 PDFLaTeX搭配 CJK 套件在 Linux 平台上製作完成。

謹將此書獻給

虎父蕭教授（麻將郎中、運動健將）

虎母蕭媽媽（鄉里名廚、女高音）

目 錄

關於作者

勞虎爲雜食動物（也吃肉），專長爲網站科技的整合運用，熟悉的項目包括 XML 、 Java 、 Perl 、資料庫、系統安全和電子商務。

留美語言學博士，玩網路純屬不務正業，曾在美擔任網管，義務翻譯 Perl FAQ ，在 Run!PC 雜誌裡寫專欄。

主修語音、文字識別的他，本行雖爲自然語言，卻深深被人工語言所吸引。他與妻子胭脂虎在 1996 年創立義務性的兩隻老虎網站，解答了世界上無數華人在網站管理和網頁設計方面的疑難雜症。

寫在前面

0.1 寫這本書的動機

現在正值 XML 科技發展的火熱階段，世界上各大軟體公司，無不卯足了勁，讓自己的產品能搶搭上這班列車。但隨著每個西方新科技的到來，背後都暗藏了一個隱憂，那就是：中文的網路大衆，往往因語言、文字的隔閡，而無法立即享受到這項新科技所帶來的便利。

XML 是一個能替未來的網路世界、乃至於人類日常生活帶來革命性改進的優良科技。這次，我眞的很不想再看到我們在這方面又 ...

因此，我起了寫這本書的念頭。我不但決定寫一本 XML 的書，讓網路上的中文人口也能不落人後，儘早享受到它的好處；更立志要寫出一本完全以中文應用爲素材、不但內容正確，而且夠新、能跟得上 XML 日新月異的發展腳步的書。

同樣重要的是，這本書必須讓所有對 XML 有志一同的網友，都能很輕易、快速地取得。要達到這個目的，這必須是一本免費下載的電子書。但免費並不表示在「質」的方面會打折扣 — 相反地，這本書以時下熱門的 PDF 格式精心製作，不管是在螢幕上閱讀，或以印表機印出，都能獲得最佳的效果，將電子書的特性，發揮到極至。

0.2 讀者應具備的基礎

在學 XML 之前，建議您最好先對 HTML 標注語有基本的認識，如果曾動手寫過 HTML 表格 (tables) 則更好。書中對所有用作範例的 HTML 碼，均不作額外解釋。

此外，本書將提到一些不屬於 XML ，但和 XML 息息相關的科技，包括 CSS 和 DOM。如果您已具備 CSS 的基本概念，則將會很有幫助。即使您從沒用過 CSS，也不用太擔心，本書將會提供一段 CSS 入門。

0.3 本書所使用的記號與標示法

- 重要的專有名詞，第一次出現時皆以綠色螢光筆 (marker) 效果表示。
- 文句中要特別強調的地方一律以*斜體*表示。
- 範例和原始碼以打字機字體（`Courier`）呈現，如：

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

此外，書中穿插了一些框框，同時配合上一些可愛的小符號，如：  、  等，用來提醒一些注意事項，或提供實用的小點子。這些框框，聰明的您到時候一看就懂，在此不需要廢話，浪費篇幅多作說明，讓書名蒙羞。

0.4 如何善用書中的各項功能

這本書在編排上，應用到許多 PDF 文件特有的便利功能，包括書籤和超連結。

書籤 PDF 中所謂的「書籤」，就是當您用 PDF 閱讀器（Acrobat Reader 等）一打開這本書的時候，在左手邊所看到的那個樹狀結構目錄[1]。本書的書籤，在中文版的 Acrobat Reader 中，可以正確地顯示。至於在其他語言的閱讀器和作業系統底下，如果配合南極星一類的軟體，應該也能正確地顯現出來。

[1]這和瀏覽器中的書籤有所不同，也不能讓使用者把自己喜歡的 PDF 頁面添加到書籤裡。

內文交叉鍊結 文中相互參考的鍊結，一律以深綠色表示。您在電腦上直接閱讀這本書的時候，可以用這些鍊結把您帶到書中其他地方。像前面總目錄中的各個章節項目、文章中的註腳，及各附表、附圖的編號，均以內部鍊結方式呈現，方便讀者查閱。

每當書中提到重要的專有名詞時，如果這個名詞，在書中其他地方有特別解釋的話，您會在緊接著該名詞之後，看到一個綠色的小鍊結，分兩種：

章節鍊結 如果一個專有名詞，是某章節的主題（例如關於 Unicode[3] 的討論），這個時候您在小鍊結中看到的，正是那個章節的編號。

頁鍊結 如果專有名詞不是某章節探討的主題，但曾在某頁提到過，譬如像統漢字[p.36] (Unihan) 這個名詞，此時在小鍊結中出現的是頁數，而不是章節編號。

這樣設計的目的，除了著眼於整體版面的美觀外，更重要的是爲了讓把書印出來、在紙上閱讀的讀者，也能享受到類似線上超連結般的便利。

外部鍊結 除了內部鍊結外，所有書裡提到的網址及網上的資源，都一律以深藍色超連結方式呈現，如： W3C XML 首頁，一切就像您在瀏覽器中閱覽網頁一樣。這個例子同時展現了本書的另一特色 — 因爲有鑑於長串的網址，穿插於字裡行間，常會影響行文流暢，進而減低閱讀速度。我決定把它們移到頁面邊緣的空白處，這也是爲了讓讀者在紙上閱讀時，也不會錯失這些相關的網址資訊。

W3C XML 首頁：
http://www.w3.org/xml/

TIP

在 Acrobat Reader 的設定中，您可以自行指定外部瀏覽器，這樣當您在按下書中的外部鍊結時，它就會去呼叫您心愛的瀏覽器，並自動替您連到該網址去。

0.5 缺憾

雖然我對這本電子書的編排及各項超連結的設計方面，煞費苦心，但礙於製作軟體的限制，目前尚無法在 PDF 閱讀器中，搜索書裡的中文字。對這個缺陷，還請您多包涵，並請多利用各項超連結功能，助您快速地跳到想找的章節和內容。

0.6 最後

本著兩隻老虎「取之於網，用之於網」的一貫的精神，在此希望能透過這本書，爲網上的中文世界，再盡一份拋磚引玉的棉薄之力。

不管您有任何批評、建議、喝采、鼓勵，或錯誤校正，都非常歡迎您寫信來給我：pailing@2Ti.com。

1 介紹 XML

XML（eXtensible Markup Language，可延伸式標注語）是網路科技中最閃亮的明日之星。的確，過去一年間，在歐美各大資訊類的雜誌、網站裡，這個字眼可說已經到達氾濫的地步。各大小資訊產品，無不爭相和它攀關係，搶搭這班最熱門的列車。押重寶在 XML 下的公司，可以說所有軟體界的龍頭全到齊了 — 微軟、Oracle 、 IBM。

1.1 爲什麼要學 XML

1.1.1 江郎才盡的 HTML

要介紹 XML ，無可避免地得先數落一下 HTML 的缺陷。但請別誤會，我不是在鼓吹說 HTML 一無是處。相反地，HTML 對整個 WWW 這幾年來的蓬勃發展、知識和資訊的流通，可說是第一功臣，更直接帶動了一波前所未有的資訊革命。不管要在網路上做生意，或要和世界網友做文件交流，人人都得學著用 HTML 寫網頁，HTML 在短短幾年內，儼然已成爲資訊交流下，通行最廣的標準格式。盡管 HTML 在人機介面方面很拿手（因爲它正是設計來將文件內容呈現在使用者眼前的），但是卻非常不利於機器之間的互相交流、傳遞資訊。我們來看個網路書店的實例。下面這段 HTML 碼，將書籍資訊以表格 (table) 方式排列：

```
<h1>推薦叢書</h1>
<table border="1" cellpadding="5">
    <tr>
        <th>名稱</th>
        <th>作者</th>
        <th>售價（新台幣）</th>
```

圖 1.1: 以 HTML 表格呈現的書籍資訊

```
    </tr>
    <tr>
        <!-- 替老婆胭脂虎的書所做的無恥宣傳 ;-) ，請勿見怪 -->
        <td>煞死你的網頁設計絕招</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>無名氏</td>
        <td align="right">120</td>
    </tr>
    <tr>
        ......
</table>
```

在瀏覽器中呈現出來，大致會像附圖 1.1 那樣。

但 HTML 的問題正出在，HTML 的標籤大多是設計來呈現文章的*格局* (layout) 和外*觀*的，譬如像上例中的 `<table>` 、 `<tr>` 、 `<td>`，還有像 `<ul>` 、 `<ol>` 、 `<li>` 、 `<font>` 等比比皆是。

今天假設我們要在網路上設立一個新興行業 — 利用機器爬蟲程式 (crawlers) 到網路上各電子商店去自動把最新的價目抓回來，讓我們的客戶可以藉此比價，從此再也不需要辛辛苦苦地一家家網站親自去看。問題在，書店甲的網頁可能使用類似上例的寫法，也就是用 HTML 表格 (table) 來呈現，但書店乙則可能選用完全不同的形式（譬如用 `<ul><li>...</ul>` 的條列方式），而書店丙可能用的又是另一套。更嚴重的是，一個 HTML 網頁中可能有一大票長得非常類似的標籤（如： `<tr>` 、 `<td>` 、 `<li>` ），有些或許是拿來標注廣告看板的，還有些是拿來標導覽連結的，到底哪幾個才是標商品名稱和價目的標籤？要如何將它們可靠地萃取出來？這就有點硬斗了[1]。如果我們要去 check 的網站隔一斷時間就拉皮翻新一次，那麼事情會更棘手。我們的爬蟲程式如果不是特別聰明，那根本別想對各色各樣的網頁去蕪存菁，找出關鍵資訊，達成任務。如果這些網頁資訊只是純供人類閱讀、消化，那沒有問題（假設一切都排列整齊）；但在資訊爆炸的網路世界，人類需要逐漸仰賴機器來替我們多分憂解勞，以 HTML 碼來標注的資訊，對機器和寫程式的人來說，不但太殘忍了點，處理起來更是極度缺乏效率。

HTML 對佈局、外觀方面很擅長，卻極度缺乏對*內容*，也就是資訊涵意的表達能力。除了少數幾個用來表達內容或文義的標籤，如 `<p>` 、 `<address>` 、 `<title>` ， `<strong>` 外，幾乎全都是用來設計網頁格局的了。所以前面才說， HTML 是設計來做人機交流用的。對上面所碰到的難題，我們最需要的，是一個能將商品價格明確標示出來的機制，譬如說，一個 `<price>` 標籤。

[1] 如果您不太能想像的話，建議您到一個電子商務網站去逛一逛，先在網頁中選定一樣商品，然後在瀏覽器中選擇顯示 HTML 原始碼，看能不能很輕易地把該項商品在原始碼中的「方位」簡單明瞭地形容出來 ;-)。

「那是不是就從改進、修訂 HTML 這方面來著手，適時按需要多加進一些新標籤？」

不行。HTML 這幾年來在兩大瀏覽器鏖戰的結果下，早已過度膨脹，肥得像個怪物了。而且，我們既不是微軟，又不是網景，或 W3C ，光是我們說希望爲線上購物制訂一個 <price> 的新標籤，難道人家就會甩我們嗎？更何況，各行各業需要用到各式各樣不同的標籤，如果統統都加進 HTML 裡，那還得了！？

其實線上商店裡的資訊，在還沒有製作成 HTML 網頁之前，可能都是一筆筆按照欄位儲存在資料庫的 tables 或物件裡的（稍具規模的電子商務網站都是這麼做），商品名稱、代號、價格 ...等，各自存在所屬的欄位、項目中，分得一清二楚；但一旦從資料庫中被調出來，再經 CGI/ASP/Cold Fusion/PHP/... scripts 搞一搞，轉成 HTML 格式後，這些重要的欄位、項目名稱全變成了毫無特定意義的 <h2> 、 <h3> 或 <th> 、 <td> ...。換句話說，原本建立在資料庫中非常寶貴的資訊*架構* (schema) ，在商品資訊轉換爲 HTML 、提供給客戶的同時，蕩然無存。試想：今天我們如果能將資料庫中的資料（如：商品形錄或訂單），保留原本完整的資料架構，一五一十地在自己和客戶的電腦之間互傳，必定大大有利雙方把資料快速地建入自己的資料庫內，進而大幅提升商務效率。這如果用 HTML 來做，恐怕很困難。

當今的網路世界，電子商務逐漸興盛，異質電腦系統之間的互動日益頻繁（商務資訊互傳），自動化程式角色也愈來愈重要（譬如前述的爬蟲程式），我們需要一個比目前 HTML 更好的方法，否則未來網路的發展將會愈走愈慢、愈累。歡迎來到 XML 的天地！

1.1.2 XML 前來解救

XML 和 HTML 的一大不同處，就在於在 XML 裡，我們可以自由定義標籤。定義出來的標籤，可以按自己的意思充分地表達文件的內容，譬如我們可以定義 <name> 、 <book_info> 這樣充分達意的標籤。在 XML 中，我們只注重內容，這和 HTML 強調

佈局的作法大不相同。至於 XML 文件的外觀呈現，可透過搭配 CSS[4.1] 或使用 XSLT[7.2] 來做 XML → HTML 或其他格式的變形。這點下面會有進一步的說明。

XML 讓已經會使用 HTML 的人輕鬆上手，因爲它的寫法和 HTML 類似，把標籤用 < 和 > 符號括起來。更酷的是，請看實例：

```
<?xml version="1.0" encoding="Big5" ?>

<推薦叢書>
    <書籍>
        <!-- 替老婆胭脂虎的書所做的無恥宣傳 ;-) ，請勿見怪 -->
        <名稱>煞死你的網頁設計絕招</名稱>
        <作者>胭脂虎</作者>
        <售價 貨幣單位="新台幣">590</售價>
    </書籍>
    <書籍>
        <名稱>如何在 7-11 白吃白喝</名稱>
        <作者>無名氏</作者>
        <售價 貨幣單位="新台幣">120</售價>
    </書籍>
</推薦叢書>
```

「哇！竟連標籤都是中文的哩，呵！」

是的，這也是 XML 的一大特性—世界通。XML 在設計之初便考慮到國際化的問題，因此打從一開始便建構在 Unicode 統一碼[3]之上，對身爲中文使用者的您我來說，眞的是非常大的福音。

XML 的名稱常造成誤導。雖然它的名字讓人感覺它和 HTML 是平行的，像是爲某項特定的應用（如呈現網頁）所設計；但事實上，它是一套無限延伸、用來設計各式各樣標注語的準則，也就是常說的 meta-language（超語言、形而上語言）。換句話說， XML 的層級比較高，它是用來「設計語言的語言」，應用範圍比狹隘的 HTML 廣多了，可隨著我們的想像空間無限延伸。

有了 XML 以後，前述的自動化比價程式，突然間變得很容易實現了。我們的機器爬蟲程式現在只要到線上書店的 XML 網頁中去找 <書籍> ... </書籍> 區塊裡的 <名稱> 和 <售價>，就可以順利達成任務，將各家書店目前所陳列的書籍和售價一一抓回，再

也不會迷失在一堆不具特定意義的 <p> 、 <td> 、 ... 裡了。不僅如此，資訊 XML 化之後，我們可以設計出更精準的搜索引擎，譬如叫它去找所有 XML 網頁裡，在 <品名> 這個標籤裡含有「MP3 隨身聽」這個字眼，而且同商品的 <價目> 標籤中所出現的數字必須低於台幣三千元。這在今日已經不是癡人說夢和天方夜譚，幾個月前推出的 Oracle8i 資料庫，配合上他們的搜索引擎元件 ConText，便已提供了這樣的功能。其他的 database 業者，如 IBM 和 Informix 等，也都在積極添加對 XML 的支援。此外，各大 database 業者還紛紛致力於讓輸出的資料可以輕易地轉成 XML，新資料更可直接以 XML 形式建入。

1.1.3 XML 的優越性

歸納上面的討論，您會發現， XML 具備幾項革命性的特質，有助於大幅改善現今的電腦、網路世界。以下針對 XML 最重要的幾項特性詳細說明。

異質系統間的資訊互通

從商業的角度來看，這點可說是 XML 最大的貢獻。當今，不要說不同的企業之間，就連許多企業內部的各個部門之間，都存在許多不同的系統。大到數百萬美元的 mainframe 老巨獸（所謂的 "legacy systems" ），小到掌中型記事簿電腦，系統與系統之間，往往因大相逕庭的平台、資料庫軟體等，造成資訊流通的困難。在這些異質系統之間做資訊交流，往往需要使用特殊的軟體，才能順利地跨越彼此的門檻、疆界，非常麻煩。如果哪個系統哪天換裝新的軟體，很可能又會牽一髮而動全身，造成一陣大忙亂。

有了 XML 後，異質系統之間，可以很方便地透過 XML 來作交流媒介。 XML 格式簡單易讀，對於各類型資料，舉凡物件、文章、RDBMS 裡的資料、圖形 ... ，不論文字檔或二元檔，都能標注。要作資訊交流的各大小系統上只需要裝有 XML 解析器[2.4]，便可解讀由別台機器所傳來的資訊，進而加以利用。 XML 解析器取得容易，有很多優

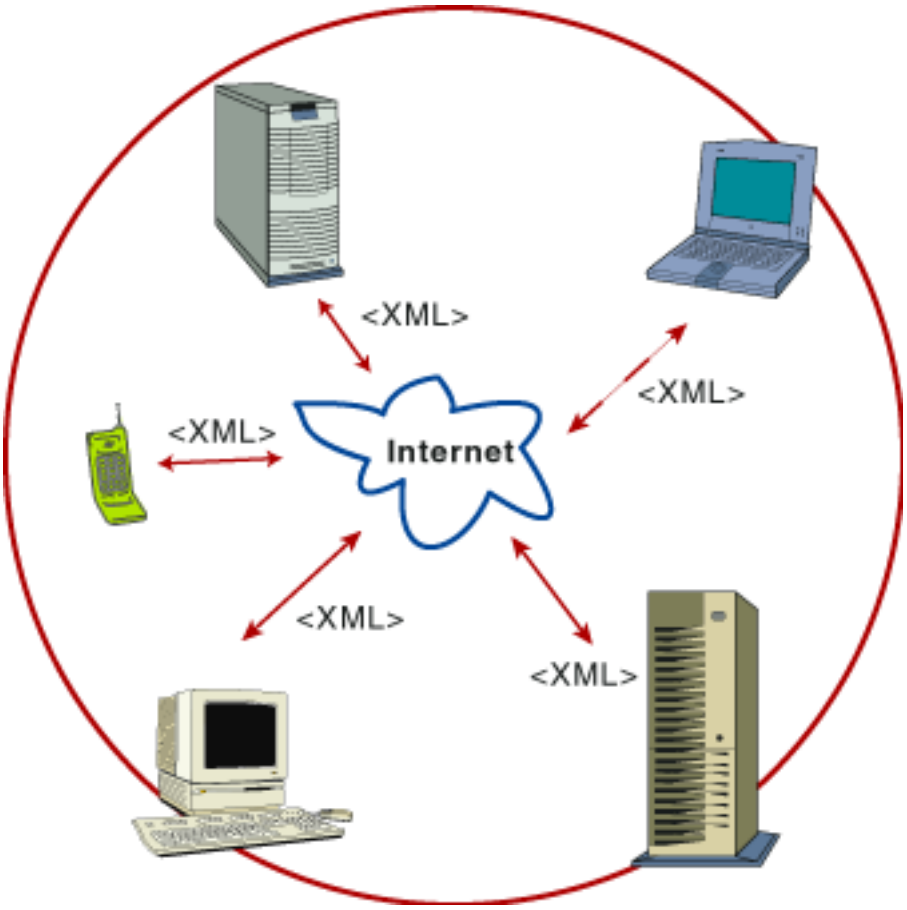

圖 1.2: XML 是非常理想的網際語言，方便各式各樣網路器具間的資訊交流

秀的軟體供人免費下載；而因爲大多數的解析器以 Java 寫成，更使得有 Java 虛擬機（解譯器）支援的平台，大到 IBM 的 AS/400 、AS/390 mainframe 級電腦，小到 Palm Pilot 掌中型記事本（還有這兩個極端之間幾乎所有的平台），都立即成爲支援 XML 的平台。異質系統之間，不用再擔心看不懂對方的資料格式，商務往來的公司之間，用不著、也不需要知道對方內部是採用何種格式儲存資料，大家都用 XML 來作中介格式即可。如此一來，某個系統內部的變更，並不會殃及和它交流往來的其他系統，因爲 XML 提供了一層理想的緩衝。

XML 這項利於資料的交換和傳遞的特性，讓很多人認爲， XML 將爲電子商務，尤其是所謂 b2b (business to business) 公司對公司的商務，帶來革命性的衝擊。譬如，一個 vertical market 可以公定一套 XML 標籤（在 XML 界的說法叫「語彙」；vocabulary），頒布遵行。這樣上下游工業、或生意夥伴間可以很方便地透過 XML 來下單、採買，或傳遞設計圖、產品明細表等。

EDI (Electronic Data Interchange) 是行之有年的電子商業文書交換標準。實踐 EDI 的公司，可以相互以電子形式交換訂單、發票，節省大量文書往返的時間和紙張的消耗。北美地區實行的 EDI 標準叫 X12 ，世界其他地區則多遵行聯合國的 EDIFACT 標準。多年以來， EDI 並沒有大量推廣開來，因爲 EDI 需要使用昂貴的軟體、聘請專業顧問、租用專屬的網路，故一般只有大型企業 (Fortune 1000 級的) 才玩得起。XML 的出現，爲負擔不起 EDI 的中小企業，提供了一道新曙光。 XML 先天上即非常適合用來作 EDI 一類的應用，可達到同樣的功能；更重要的是， XML 軟體取得方便，輕薄短小，大小機器上都能跑，又可直接利用價廉、普及的 Internet 線路來傳送（圖 1.2）。難怪不少公司已開始積極研發以 XML 爲基礎的新一代 EDI 規格。

再舉個例子：醫療健保體系，是最早被用來解說 XML 優越性的例子。使用 XML 來儲存、簡化病歷後，病人的病例可以在各醫院、診所間的大小電腦間快速遊走，因爲 XML 格式簡單（不像 HTML，有時看了眞會吐血），各醫療系統的工程師可以很輕易地寫出處理的程式，存取其他系統所傳來的 XML 資料。某病人的病歷甚至可以用 XML 儲

存在她個人的 smart card 上頭，隨身攜帶。 此外，像政府公文簡化、統一化等，XML 都是最佳候選人。

XML 是公訂、開放的標準，人人都可以拿它來開發軟體，不怕被軟體公司把持、要挾，不會有貪得無厭的公司，動不動藉著修改只有他們自己清楚的格式，來逼人花錢升級的情形出現。這正是公訂標準可貴之處，就如同今天蓬勃發展的 Internet 是建構在各項 TCP/IP 公開標準一樣。這也是爲什麼人人都應大力支援公訂標準的原因。

「有些專門設計來做連網系統間傳遞訊息的科技，譬如 CORBA-IIOP 、Java RMI ，和微軟的 DCOM 等，這些科技難道不如 XML 來得勝任嗎？」

首先， XML 和 CORBA 、DCOM 這些科技並不相衝突； XML 可以爲它們做傳遞訊息、物件的橋樑，像 XML-RPC 和微軟剛宣布的 SOAP 草案就是最好的例子。再者，以二元檔格式傳遞訊息可能要比 XML 有效率，但它們在使用簡便方面卻遠不如 XML。XML 碼都是純文字檔，而非二元檔，肉眼可輕易閱讀，更可用編輯器直接編修。純文字的特性，也讓 XML 文件可直接透過 HTTP 或 SMTP 等現成的通信協定來傳送，毋須另行編碼，或作物件序化 (serialization)[2] 、反序化處理。此外， XML 簡單易學，對已經熟悉 HTML 的人更是容易上手，沒有陡峭的學習曲線。因此和其他科技相比，它的優點大大彌補了處理起來較慢的缺點。更何況，在莫爾定律[3]歷久不衰的今日，電腦硬體的威力持續飛速成長，資料處理的速度在未來將更不是問題。

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

保值

SGML 是一套有十幾年歷史的國際標準。 HTML 便是一項 SGML 的應用實例。此外，Adobe 的文書排版工具 FrameMaker ，所用的內部格式正是 SGML。SGML 當初設計的一大目標是保值 — 它是設計來提供文件 50 年以上的壽命的。多年以來，因爲 SGML 太過龐雜，所以一直沒有在世界上大量流行起來，一般只有大型企業、或政府機

[2] XML 文件本身已算是序化了。
[3] 由英特爾 (Intel) 公司創辦人莫爾所提出，他預測科技的進展速度，將使一個晶片中所能容納的半導體總數，每年（後來修正爲每兩年）倍增一次。

構在使用。例如美國的國稅局 (IRS) 便以 SGML 來設計稅表等文件。

正因 SGML 過份複雜、難懂，且較不適用於網路，所以有識之士特別設計了 XML。也就是說， XML 是 SGML 精簡之後的網路版，而 SGML 保值的特性，在 XML 中並未消失。何謂保值？試想：不要說過去用 WordStar 1.0 格式寫成的文件，就算用今天最新的微軟 Word2000 寫出來的 `.doc` 檔案，誰敢保證 50 年後的電腦上，還找得到打開這些檔案的應用程式？50 年後的電腦上能不能跑 Word2000 、WordStar 這些早已過時的程式更成問題；而能跑這些程式的老電腦，更可能早已進垃圾堆，恐怕要到博物館才能借得到這樣的古董；如果要隨著軟體的更新和升級，不斷將大量文件配合轉換、升級到最新的格式，則又非常沒有效率。新版的文書編輯軟體，對舊版的文件中的樣式，更不見得都能一五一十地忠實重現。

XML 文件不但沒有這些問題，在未來的世界，要從 XML 文件轉換成其他的格式，更是輕而易舉。仔細想想，需要保值的文件還眞不少，舉凡政府命令、公文、法律文件、史料、醫療記錄、科學研究報告，甚至於個人的日記、回憶錄等，都需要保存很長的時間，供後世子孫參考。在文書大量電子化的 21 世紀，要如何長久保存這些文件和記錄，必將是一大課題。而 SGML 和 XML 的設計，正是 20 世紀末的科技先知對此一課題的答覆。

自動化 user agent 不再是奢望

前面第 3 頁中所提到的機器爬蟲程式，是自動化 user agent （使用者代表）程式（以下簡稱機器人）的一個典型的例子。網上的各大搜索引擎站，靠的正是這些小程式，每天不斷將成千上萬的網頁內容抓回。幾年前就有人期待，未來的網路世界美景，將充滿這樣的情節：

1. 使用者最近要出一趟遠門，需要訂機票。

2. 這位使用者於是把她電腦上的機器人叫出來，把心目中理想的航空公司，及飛行

的起點、終點機場等設定好後，讓程式去網路上的旅行社站台把最新的報價抓回來。

3. 報價一一抓到齊了以後，機器人向該使用者回報，告知哪一家價格最便宜，提供作爲訂機票的重要參考。

4. 使用者根據機器人所蒐集到的情報，透過瀏覽器到最便宜的那家旅行社的站台訂位子，甚至將訂位子的大任直接賦予機器人。大功告成！

但是像這樣美好的遠景，至少到目前都還沒有實現。部分原因，正是前面所說的，HTML 文件解讀不易。這裡所謂的解讀，當然不是指看得懂那些版面、佈局的指令，這個工作瀏覽器已經做得相當好了。難是難在，軟體要具備解讀文意的能力。

XML 讓實現 smart agent 的夢想，朝實現之日邁進一大步。有了 XML 以後，我們可以自行設計達意的標籤，如：<價格> 、 <商品名> 、 <日期> ， user agent 程式可以透過這些標籤，很快地找到要找的資訊，而不會迷失在一片「HTML 汪洋」中了。

更精準的搜索

這個特性和上一個（自動化 user agent）可說是相輔相成。XML 標籤涵意豐富，明白提示所標注的內容，讓搜索引擎藉由標籤和內容之間的依存關係，準確地定位、找到目標，達成任務。舉個例子：我個人偏好那種打起來很響的鍵盤，用起來讓周遭的人感覺我正在起勁地工作（但吵得他們難以工作 ;-）。我把我的 agent 程式送到網上去找有關電腦鍵盤的資訊，好作爲我購買前的參考。但問題是，「鍵盤」兩字太過籠統，可能出現在電腦零件的網頁中（正是我想找的），如：

```
<電腦組件>
  ......
  <周邊>林口牌超耐鍵盤， PS2 介面</周邊>
  ...
  <周邊>高級雙鍵滑鼠，附智窗型滾輪，USB 介面</周邊>
</電腦組件>
```

但它們也可能出現於某合唱團的介紹中：

```
<合唱團>
  <團名>哇Zoo!</團名>
    <成員>
      <團長>黑虎 （主唱、電貝司）</團長>
      <團員>鱷神（電吉他）</團員>
      <團員>熊哥（鍵盤）</團員>
      ......
    </成員>
</合唱團>
```

甚至還可能出現於其他類型的網頁，譬如像賣鋼琴或電子琴的產品說明頁。

有了 XML ，我只要告訴我的 agent 程式，把目標鎖定在寄居於「電腦」、「周邊」這些項目中的「鍵盤」二字即可。 爬蟲、agent 設計起來不再像在 HTML 時代那麼痛苦，而且搜索起來在效率、準確性上都要高多了！

1.2 爲 XML 打扮

在資料庫之間以 XML 形式傳遞的資料，或許不需要特別妝點，但是如果一份 XML 文件要呈現在使用者面前，或者要付印時，我們可以透過 CSS[4.1] 或 XSL[7.1] style sheets （本書翻作「樣規」）來設定外觀。CSS 最早是爲 HTML 所設計的，但拿到 XML 上一樣好用。CSS 的第二代 — CSS Level2 在發展時已經將 XML 的應用涵蓋進去了。

配合樣規，我們對 XML 文件中各個標籤裡的文字內容的外觀，包括字型、大小、顏色、排列方式、位置、層面等，都可以一一指定。更好的是，同樣的 XML 文件，可依不同場合爲它設計出多套樣規。譬如一個網路書店的圖書資料，很可能既需要放在網頁中供人瀏覽，又需要把它打印在買賣的發票裡，隨書寄給顧客。這個時候，只要設計兩份樣規就可以達到這個目的， 請看附圖 1.3。

重要的是，文件的內容和外觀設計是完全分開的。外觀（樣規）變動時，XML 文件完全不受影響。

圖 1.3: 同樣一份內容，配上不同的樣規，生出截然不同的造形

1.3 XYZ 專有名詞大會串 — XML 應用實例

下面各節將介紹的應用實例，全都是直接建置於 XML 之上的語言，充分證明了 XML 無遠弗界的彈性。

1.3.1 數學 ML

MathML 藉著 XML 定義出一套能充分表達數學式子的標注語言。它已經在不久前正式成爲 W3C 的推薦標準 (recommendation) 。隨著 XML 時代的來臨，數學家們將不再遺憾 HTML 欠缺對數學符號、公式的支援了。

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

1.3.2 微笑串連

SMIL 是「多媒體同步整合語言」(Synchronized Multimedia Integration Language) 的縮寫，讀作 "smile" :-)，也是一項 XML 的應用。廣受歡迎的 RealPlayer G2 ，便採用 SMIL 來提供影 (Real Video)、音 (Real Audio) 、圖檔 (Real Pix) 、文字 (Real Text) 串連的功能，讓設計者可以自由控制多媒體出現的時間和先後順序。

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

和 MathML 一樣，SMIL 業已成爲 W3C 正式推薦的網路標準。

1.3.3 Flash 殺手？

SVG：
http://www.w3.org/TR/SVG

SVG 是 Scalable Vector Graphics 的縮寫，它是由 W3C 所研發、架構在 XML 基礎上的向量圖形格式，發展成員包括了 Adobe、HP、IBM、Macromedia、微軟、Netscape、Quark、Sun 等大公司，目前已經達最後階段，眼看即將成爲正式標準。

過去在 XML 向量圖形的發展上，有兩派人馬，互相較勁，一派以 Adobe 爲首，提倡 PGML，另一派由微軟、Macromedia 領軍，鼓吹 VML（IE5 有支援）。在這兩套提議案呈遞給 W3C 後，W3C 決定協調各路人馬，將兩套互別苗頭的格式融合在一起，促成了 SVG 的誕生。

我們知道，目前網路上最廣爲通行的圖檔格式 — GIF、JPEG 和 PNG[4]，都是屬於點陣性的格式，而點陣圖有尺寸較大，不利縮放的缺點。Macromedia Flash 所提供的向量動畫功能，適時地趁虛而入，填補了一部分這方面的空白，成功地打響了這個產品。那既然已經有 Flash，而且 Macromedia 不久前也已將 Flash 格式透明化、供人自由發展，爲什麼還要再搞出一個新的東西呢？

首先，最重要的是，SVG 建置於純文字格式的 XML 之上，直接繼承了前面提到的 XML 特性，簡化異質系統間的資訊交流，方便資料庫的存取；而且在未來，可直接融入 XML 和 XHTML[6] 網頁中，更可以直接利用其他既有的瀏覽器端科技，如 CSS[4.1]、DOM（文件物件模型；Document Object Model），和 ECMAScript（JavaScript 升格成 ECMA〔歐洲標準協會〕正式標準後的新名子），達到動畫及 DHTML 般的動態效果[5]。SVG 也支援點陣圖檔匯入，還支援目前仍在發展階段的兩個 XML 連結語言 — XLink 和 XPointer，不但可以作 HTML 式的單向連結，更提供多向連結，和許多複雜的花樣。還有，SVG 是公訂的網路標準，不受單一的公司操縱。

DOM：
http://www.w3.org/DOM

XLink：
http://www.w3.org/xlink

XPointer：
http://www.w3.org/xptr

上面多項 SVG 的優點卻正是 Flash 的缺點：Flash 必須要倚靠瀏覽器外掛程式（插

[4] PNG（讀作 "ping"）是設計來取代 GIF 的點陣圖檔格式。不但比 GIF 更爲優秀，最重要的是，它是一個公定標準，沒有專利權的糾纏。目前支援 PNG 的瀏覽器和製圖工具已有不少，而且不斷在增加中。

[5] 所謂的 DHTML，正是透過 CSS、DOM 和 JavaScript 所營造出來的動態效果的泛稱。

件；plug-ins）來播放，而且因爲 Flash 的格式爲二元檔，作品內的文字內容無法讓使用者在瀏覽器中作字串搜索（的確有人把文章做進 Flash 電影裡！），這樣的網頁，也無法讓搜索引擎站索引、登錄其中的文字，供訪客作全文檢索。此外，高互動性的多媒體動畫，往往需要靠編寫程式來幫忙（因此 Director 才有 Lingo 語言），這點是 Flash 另一個較爲不利的地方：Flash 和 JavaScript 之間的互動，只能透過比較狹窄的 FSCommand 來作橋樑；雖然 Flash 從第四版也開始加入一些簡單的程式設計功能，但仍不夠完備，和發展已趨成熟的 ECMAScript 相比還差上一截，更別說 ECMAScript 早已有廣大的使用人口。

哇，說了這麼多 Flash 的壞話，我老婆胭脂虎的頭上已經開始冒煙，看來快變成「煙之虎」了！趕快在此補充一點：儘管 Flash 有上述這些缺憾，但至少在今日，仍舊是網頁向量動畫最好的解決方案。SVG 的遠景雖美好，但目前尚仍停留在明日之星的地位，最後下場會怎樣，還很難斷定。

我們來看看 SVG 碼究竟長得什麼樣（直接取材自 SVG 標準文件）：

```
<?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 碼會畫出一個藍色的圓，紅色的外框</desc>
  <g>
    <circle style="fill: blue; stroke: red"
      cx="200" cy="200" r="100"/>
  </g>
</svg>
```

因爲 SVG 太新了，目前幾個主要的瀏覽器，包括 Netscape 、 IE 和 Opera，都還沒有支援。不過已經有 SVG 的瀏覽器，像 IBM 的 SVGView （見圖 1.4）。

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

因爲 SVG 架構在 XML 上，而 IE5 和 Netscape5 都已內建 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，所以未來的瀏覽器有可能加入對 SVG 的支援。尤其是 Netscape ，因爲它已經改採 Open Source 程式碼公開的模式來發展（Mozilla 計畫），任何熱心人

Mozilla 計畫：
http://www.mozilla.org

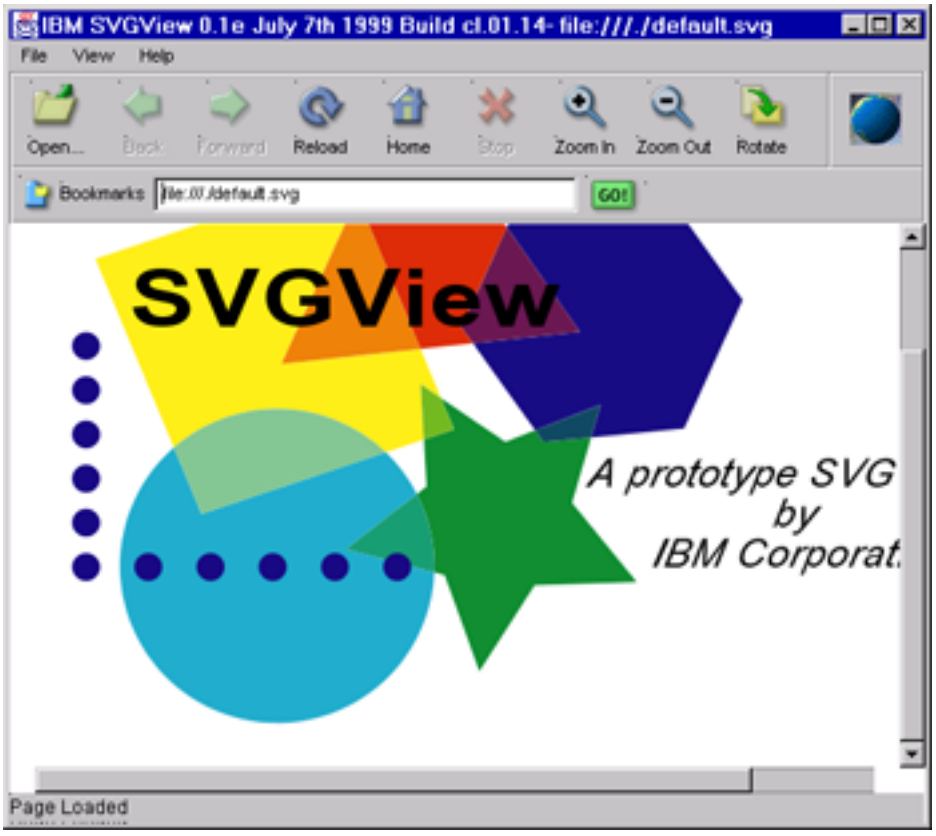

圖 1.4: IBM 的 SVG 瀏覽器

士都可以主動貢獻，SVG 的支援隨時有可能加進去。

要能讓 SVG 大量推廣開來，關鍵在要有方便好用的編輯工具出現。由於 SVG 是公訂標準，主要在幕後稱腰的又是 Adobe，而他們在 WWW 軟體市場的死敵偏偏又是 Macromedia ，加上 XML 熱潮所挾帶的優勢，從這幾方面研判，SVG 在未來不是沒有潛力取代 Flash，成爲網路上的頭號向量圖格式。如果到時候 ，「Flash 殺手」效應眞的發生，別忘了，您是先在這裡看到的哦 ;-)！

1.3.4 XSL

CSS[4.1] 之外，另一種替 XML 打扮外觀的科技是 XSL[7.1]。 XSL 所使用的方法和 CSS 大異其趣，我們在 7.1 節會詳細介紹。

1.3.5 有哪些相關的網站？

Netscape 瀏覽器 4.06 及 4.5 之後的版本，內建了一項 "What's Related" — 同類型網站推薦服務。其幕後的一大功臣，除了資料庫及搜索引擎外，應首推 RDF （資源定義架構；Resource Definition Framework）了。

RDF：
http://www.w3.org/RDF

先簡單談一下 "What's Related" 的功能是怎麼做出來的：每當使用者按下這個按鈕時，瀏覽器便將使用者正在瀏覽的網頁網址，傳送到 Netscape 公司的 server 上，server 在它的資料庫中找到這個網頁，並且將所有和這個網頁關係最密切的其他網址及相關資料，以 RDF 格式傳回給瀏覽器，最後以菜單方式呈現在使用者面前，供使用者選擇前往。

不用說，RDF 是另一個重要的 XML 語彙。它演化自 HTML 中藉著 `<meta>` 標籤來描述網路資源的做法。而所謂的「形而上資訊」(meta info; meta data)，用一句話簡單解釋[6]，就是 "data about data"。譬如圖書館的書目和作者索引，不管是較進步的電腦數位式，或是傳統的「卡片、小抽屜」式，正是典型的形而上資訊。RDF 設計的目

[6]借用 RDF 標準中的說法。

的，正是用來描述形而上資訊，不但要讓機器能讀，更要讓它能*讀得懂*（這個重要的區別，在前面討論 HTML 和 XML 的優缺點時，已經強調過）。RDF 著重於機器與機器之間的自動化交流，可以用來描述網路資源，方便搜尋，還能表達資源與資源間的相互關係。

「那 Netscape 從哪得來這麼多網頁、網站關連性的資料呢？」

這是由 Alexa 這家搜索引擎公司提供的。Alexa 的創辦人之一，正是過去名震一時的 WAIS 搜索引擎的發明人。它的搜索服務是個不落俗套、很有創意的點子：它們不接受網站來主動報名登記，運作上完全仰賴機器爬蟲大量抓回來的網頁，再對其中的超連結，作網站關連性的分析，而 "What's Related" 使用者的選擇，則作爲修正關連係數的重要參考。

反觀其他絕大多數的搜索引擎站，如 AltaVista ，因爲採用自由登記制，而且將網頁中 `<meta>` 標籤所列的關鍵字作爲搜索結果排列順序的重要參考，已經導致一些不肖網站業者，尤其是色情和詐財性質的站台，爲了吸引更多的訪客，濫用關鍵字的設計，嚴重扭曲搜索結果，無形中也降低了搜尋站的實用價值。

1.3.6 使用介面隨你設

呼之欲出的下一代 Netscape5 瀏覽器 (Mozilla) ，從裡到外，整個脫了一層皮。除了 100% 支援 W3C 的 HTML4.0 、 XML 、 CSS1 、 DOM1 等外，對 CSS2 、 DOM2[7] 也有一些支援。 Mozilla 的使用介面，也是一大創新，不但完全跨平台，而且可以讓人自由設定按鈕、菜單、指令等元件。用來設定、控制使用介面的語言 — XUL ，又是一項 XML 的應用。也就是說，熟悉 XML 的人，可以充分地用 XUL 和 CSS 來改變 Mozilla 的使用介面，瀏覽器的外觀也因而變得像系統桌面 (themes) 一樣，讓有設計巧思的人充分發揮想像的空間。就像熱門的 MP3 播放器 Sonique 和 WinAmp 所支援的 skin 功能一樣，在未來，每當我們看膩了瀏覽器的介面時，便可到網路上去下載一套更酷的皮層

HTML4.0 : `http://www.w3.org/TR/REC-html40`

XUL : `http://www.mozilla.org/xpfe/xptoolkit/xulintro.html`

[7] 仍在草案階段，即將成爲正式標準。

換上去。

1.3.7 那微軟呢？

微軟很早就開始使用 XML。事實上， IE4 瀏覽器所內建的 CDF (Channel Definition Format) 功能，提供頻道訂閱、 push 服務，堪稱爲最早的 XML 應用實例。IE5 瀏覽器對 XML 、 XSL[7.1] 、和 DOM ，也都有相當程度的支援。此外，剛出爐的 Office 2000，在儲存成 HTML 格式時，利用 `<xml> ... </xml>` 區塊（微軟的術語叫「 XML 島」），來達到雙向 (round-trip) 的格式轉換；也就是過去發表成 HTML 網頁的 Word、Excel，PowerPoint 等文件，可以重新讀入 Office 軟體中，忠實地重現爲原來的 `.doc` 、`..xls`，和 `.ppt` 格式。

微軟未來的電子商務策略中，XML 將是重頭戲。他們目前正將下一代的 COM (Component Object Model) 介面開放，讓其他系統可以透過 XML 來與之溝通。未來支援 XML 的主機軟體，更包括了 Windows2000 、 SQL Server 、 BizTalk server ，和 Commerce server 等。

1.4 先用先贏

XML 不是明日的科技，您*現在*就可以充分享受它所帶來的好處。多項重要的 XML 相關科技及標準，都已發展成熟。各類應用軟體更如雨後春筍般，不斷快速地增加。

2 XML 語法領進門

我們繼續延用第 1 章「推薦叢書」的例子（附表 2.1 ）。您可以利用其中的色塊，直接跳到說明該項語法的章節去一看究竟。

```
<?xml version="1.0" encoding="Big5" ?>A
<?xml-stylesheet href="style.css" type="text/css" ?>B
<推薦叢書>
    <書籍>
        <!-- 替老婆胭脂虎的書所做的無恥宣傳 ;-) ，請勿見怪 -->C
        <名稱>煞死你的網頁設計絕招</名稱>
        <作者>胭脂虎</作者>
        <售價 貨幣單位="新台幣">590</售價>D
    </書籍>
    <書籍>
        <名稱>如何在 7-11 白吃白喝</名稱>E
        <作者>無名氏</作者>
        <售價 貨幣單位="新台幣">120</售價>
    </書籍>
</推薦叢書>
```

表 2.1: 中文 XML 實例

2.1 前奏

在表 2.1 裡的第一行敘述中，我們看到：A

```
<?xml version="1.0" encoding="Big5" ?>
```

這叫 XML 宣告 (declaration)，更有學問的稱呼爲 前言 (prolog)。其中 `version` 這個註明版本的屬性一定要有，而 `encoding` 這個註明文字編碼的屬性則可有可無，如果省略的話，字碼必須是 Unicode[3] ，以 UTF-8 或 UTF-16[3.2] 作編碼。在這樣的情況下，甚至可以將整行 `<?xml ... ?>` 宣告一併省去，不過 XML 標準中強烈建議不要這麼做，不管用的是不是 Unicode ，最好還是養成一律寫的好習慣。

在我們的例子中，則明確地註明使用的字碼是 Big5。在 XML 文件的編碼不是 UTF-8 或 UTF-16 的情形下， `<?xml ... ?>` 宣告絕不可省，而 `encoding` 也絕不可少。

⚠ 注意

不是所有 XML 軟體都支援 Big5/GB2312。 XML 標準只強制規定所有軟體必須支援 UTF-8 和 UTF-16[3] 這兩種編碼，至於對其他國家、地區性編碼的支援，則由各軟體發展者自行決定。因此，不見得所有的 XML 軟體都能正確處理 `encoding` 標示爲 Big5/GB2312 的文件。至於爲什麼要這樣規定，我們在第 3 章討論 Unicode 時會探討這個問題。

2.2 元素與屬性

D

俗稱的「標籤」(tags)，實際上包含了「元素」(elements) 和「屬性」(attributes) 兩部分。 例如在表 2.1 中，「作者」是直屬於「推薦叢書」這個母元素底下的子元素。而「貨幣單位="..."」則是「售價」這個元素的一個屬性；我們把「貨幣單位」稱作「屬性名」，等號後面的值則稱作「屬性值」。最高層的元素（推薦叢書）稱作「根元素」(root element)。

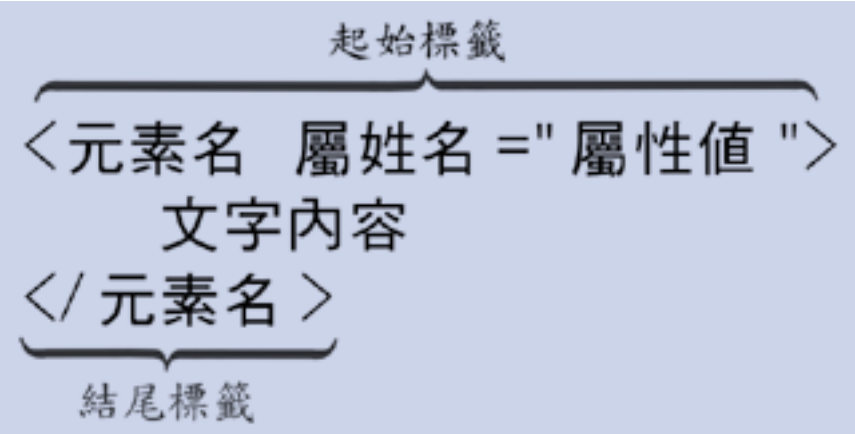

圖 2.1: XML 元素與屬性

2.3 註解

在 XML 中，註解語法和 HTML中非常相似。雖然嚴格來講，不完全一樣，但二者的差別在這裡並不重要。一如 HTML ，註解是放在 <!-- 和 --> 之間的區塊，如附表 2.1 所示。

> ⚠ 注意
>
> 附表 2.1 的 XML 碼中，用來界定元素的是 ASCII 中的小於和大於符號（< 和 > ；半形）。因此在編輯器中敲入中文的元素名時，要注意不要因疏忽而誤輸入了中文字碼中的全形符號，像〝＜〞、〝＞〞、〝〈〞、〝〉〞等。

2.4 不可或缺的解析器

在解釋 XML 嚴格的語法規定之前，讓我們先來談談 parsing 這個重要的概念。〝parse〞就是解析的意思，身爲萬物之靈的人類，我們的大腦每天都在做大量的

parsing 動作，卻往往渾然不自知 — 像您現在就正在做 parsing — 閱讀這本書。您之所以看得懂我寫的文章（希望如此 ;-），而且能聽懂周遭的人講的話，正是因爲您的腦中有一台非常厲害的語言解析器，英文叫 "parser"，隨時在高速解讀文章和話語，從斷字、斷句一直到語法、語意分析；這個解析器的複雜度和精確度是任何電腦語音輸入軟體所望塵莫及的。假設在大街上，有一個老外向您迎面走來，開口道：

「阿度仔名我叫，哪裡 bus 可以我搭？」

您心想：「哇！這是哪國的語法，也不知道這個老外是在哪裡學中文的，不過八成在學校是被當的！」

爲什麼我們一聽到那個老外的話，會馬上有這種反應？這就是因爲我們大腦中的語言解析器，也就是 parser，在舉牌抗議，讓我們知道這個句子怪裡怪氣的。

解析器是語言處理的最前線。譬如我從來沒學過泰國話，腦中因而欠缺泰語的解析器，當面對一份泰語文件時，我連最基本的字、詞、句子結構都搞不清楚，更甭提讓我進一步翻譯、或修改這篇文章了。

同樣地，軟體也需要內建語言解析器，才能處理某種語言。XML 是一種語言，在我們能進一步利用文件的內容之前，一定要先用解析器把文件分析成一個個物件才行。還有，像網頁瀏覽器，必定具備一個 HTML 解析器，這樣它才能讀得懂各網站裡的 HTML 檔，進而將網頁佈局呈現在使用者面前。如果它讀到太誇張離譜的 HTML 文件，可能就無法按原 HTML 作者所希望表達的方式，把文章順利地在瀏覽者面前呈現出來，甚至可能一片空白，什麼都出不來。換句話說，瀏覽器內建的解析器鬥不過亂七八糟的網頁，只好認輸、放棄了。

XML 的設計者有鑑於目前網路上充斥了大量原始碼「不純淨」的網頁（多數 HTML 檔的寫法和格式嚴格講起來都不合格），因此決定這次要從一開始就嚴格執行，規定所有的 XML 文件都必須遵守幾個基本規定（下一節的主題），否則一律不留情，統統賞予那可怕的錯誤訊息。

「他們幹麼要規定得這麼嚴？」

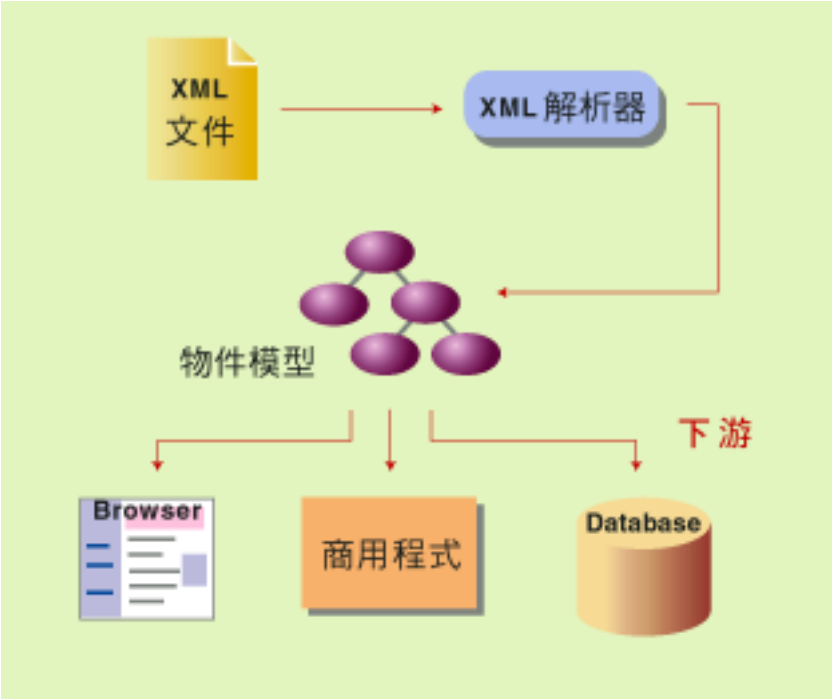

圖 2.2: 解析器是處理任何 XML 文件的第一關，分析出來的物件則交給下游的應用程式作進一步的用途

XML 簡潔的語法，及對格式的嚴格要求，大大地造福那些吐血、想破頭去發展標注語解析器、瀏覽器的可憐工程師。這個容易發展的特性，有利於 XML 的快速普及化。XML 在發展時一共訂了十條最高指導原則，這是第四條，在設計 XML 的科學家之間的打趣說法是：一個電腦科系的研究生，花頂多兩個禮拜時間，就應該能寫出一個解讀 XML 資料的程式。

要附帶一提的是，並不是用兩大瀏覽器測試過沒問題的網頁，就保證完全符合 HTML 標準。事實上，正因爲網路上有太多不合格的網頁，使得設計瀏覽器的工程師無不絞盡腦汁，讓他們的產品能夠盡量圓滑地處理各種五花八門、語法偏差的網頁。這就像前面提到的老外問路的例子，儘管我們直覺他講出來的話很奇怪，但或許我們能大致猜得出來他要說的是什麼（所以我才說我們大腦中的語言解析器眞的是很厲害）。

2.5 XML 文件必先要「及格」

「及格」在 XML 中的正式說法叫 "well-formed"，也就是格式正確。任何文件要能稱得上是 XML 文件之前，必得先「達到及格標準」。達不到標準的文件，會讓 XML 解析器噎到，解析失敗，什麼都做不成。

那麼到底要怎樣的 XML 文件才算及格呢？不難，主要有以下幾個原則：

2.5.1 所有元素都要正確地關閉

在表 2.1 中，我們看到所有的元素[2.2]，不管其中穿插了多少筆資料、或幾個子元素，到最後一定都會有一個像 HTML 中的結束標籤把這個元素「關起來」，譬如：<作者> 某人 </作者>。根據 HTML 的標準規定，有不少標籤，例如 <p>、<tr>、<td> 等，它們的結束標籤是可有可無的；但在 XML 中，結束標籤絕不可少。如前面所說，這樣嚴格的規定，大大減輕了發展 XML 解析器和其他開發工具時的負擔。

腦筋動得快的讀者可能已經在想了：「那萬一有像 HTML 裡，
、<img

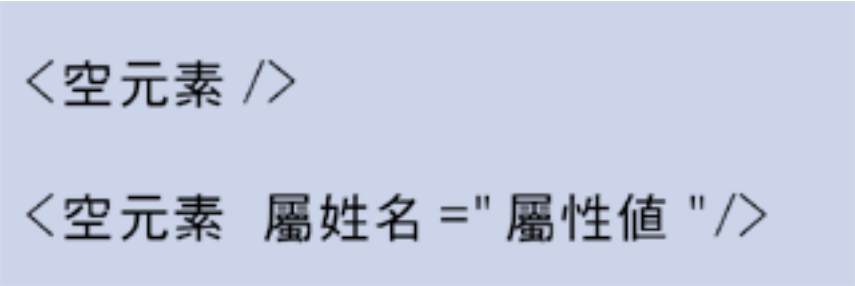

圖 2.3: XML 特有的空元素標示法

src="foo"> 這類自成一個單元的標籤怎麼辦？」

問得好！這在 XML 中叫「空元素」(empty element)，因爲這樣的元素不內含任何文字內容，只有屬性。XML 爲空元素特別發明了一種新的表示法，像這樣：

```
<元素/>
```

如果帶有屬性的話則寫成：

```
<元素 屬性甲="foo" 屬性乙="bar"/>
```

2.5.2 標籤之間不得交叉

我們繼續使用「推薦叢書」的例子。假設今天有這樣一個標籤排列：

```
<書籍>
  <名稱>如何在 7-11 白吃白喝
    <作者>
  </名稱>
    無名氏</作者>
```

這就犯了「標籤之間不得相交」的大忌，會被當掉。 XML 中規定，所有的元素排列必須是嚴謹的樹狀結構。樹狀結構的觀念對學 XML 的人非常重要，在使用 DOM 、XSLT[7.2] ，和 XPointer 來分別控制、轉換，連結 XML 文件時，都需要隨時對文件的內部結構瞭若指掌。如果您對資料結構 (data structure) 和物件導向的概念較爲單薄，而在未來可能需要寫 XML 類的程式的話，建議您找時間加強這方面的基礎。

2.5.3 所有屬性都得包上引號

在 HTML 中，我們已經被寵壞，常常忘了或懶得用引號把各標籤的屬性值包起來。網頁瀏覽器對這樣的寫法通常都能正確處理，照單全收，讓這個漏加引號的現象更加普及。甚至就連一些網頁開發、轉換工具都這麼做（譬如 Office2000 所存成的 HTML 碼）。這對 HTML 不是什麼大問題，但拿到 XML 中，卻會慘遭被 XML 解析器判出局的命運。因此，加引號的習慣，越早養成越好。

2.5.4 其他規定

其他格式正確的要求還很多，上頭提出來說明的，只是幾個大家在應用上最常碰到的。其他 XML 格式正確方面的規矩，就不在此一一列舉，因爲這本書的目標不是要把您訓練成 XML 專家，而是要讓您能很快地進入情況，開始應用。事實上，有些這方面的規定，可能只有設計 XML 解析器的軟體工程師們需要知道，光是一一解釋這些條文細節，就足足可以寫一本書（那書名大概也要改叫「愛睏 XML」了 ;-）。

2.6 以法爲證

有時候光是及格還不夠。現在假設我們將附表 2.1 中的 XML 碼加以修改，把第二筆 <書籍> 資料，改成：

```
<書籍>
    <名稱>如何在 7-11 白吃白喝</名稱>
    <作者>無名氏</作者>
    <售價 貨幣單位="新台幣">120</售價>
    <售價 貨幣單位="新台幣">90</售價>
</書籍>
```

以上的 XML 碼在格式上沒有問題，算是 well-formed 的 XML 文件（您可以用上述「及格標準」來一一印證看看）。但一本書同時有兩個價碼，而且使用的是同一種貨幣，似乎有點怪，不過或許在某些場合下是可以成立的（譬如，其中一個是團體訂購的

優待價）。但是，這是不是就是網路書店經營者心目中希望的格式呢？又假設，今天《如何在 7-11 白吃白喝》這本暢銷書冒出一個第二作者，叫「痞子」，他才是眞正幕後的「幽靈作者」(ghostwriter)，無名氏只是坐在那裡等收錢的（就像比爾大哥那兩本暢銷書 ;-），那麼我們的 XML 標籤該如何表示呢？到底是：

```
<書籍>
        <名稱>如何在 7-11 白吃白喝</名稱>
        <作者 人數="2">無名氏, 痞子</作者>
        ......
```

還是乾脆就這樣：

```
<書籍>
  <名稱>如何在 7-11 白吃白喝</名稱>
        <作者>無名氏, 痞子</作者>
        ......
```

因爲作者人數可藉由逗點的數目，讓處理資料的程式自動算出。還是：

```
<書籍>
        <名稱>如何在 7-11 白吃白喝</名稱>
        <作者>無名氏</作者>
        <作者>痞子</作者>
        ......
```

呢？很明顯地，我們現在需要定義一套法則來規範它。我們管這套法規叫 DTD[8] (Document Type Definition)，也就是對某種 XML 文件在格式上的定義。我們可以在其中規定，「作者」這個元素到底是可以出現多次、還是在任何一個 <書籍>...</書籍> 區塊中只能出現一次。還有像一個元素能包含哪些屬性、子母元素相互依存的關係、各個元素出現的順序等，都能用 DTD 一一清楚地加以定義和規範。用 DTD 定義出來的一套 XML 應用，專業術語叫「*語彙*」。

可以用 DTD 來確認其正確性的 XML 文件稱作 valid （有法可證的）XML。會根據 DTD 中的定義來確認 XML 文件正確性的解析器叫“validating parser”，沒有這種功能的解析器叫“non-validating parser”。

在使用 XML 時，我們通常會使用 DTD 已經設計好的現成 XML 語彙，譬如在第 1 章所介紹的 MathML[1.3.1] 、 SMIL[1.3.2] 等，或使用由各同業公會所制訂的專用語彙。自行設計 DTD 難度較高，但也不是那麼遙不可及。我們在第 8 章會解釋 DTD 的語法。

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

目前 W3C 正在研發、眾所企盼的 XML Schema （組織架構）標準，在未來不但可能會逐漸取代 DTD，更將提供更多的功能。更好的是， XML Schema 完全採用 XML 語法，不像 DTD 那般，有自成一格的怪異表示法[1]。

2.7 大小寫有分

XML 常見問題集 : http://www.ucc.ie/xml

XML 常見問題集 (FAQ) 中有一條，問說 XML 元素、屬性名有沒有分大小寫。答案是有。XML 在這點上和 HTML 截然不同。這個有趣的現象就好比各作業系統對檔名大小寫的態度一樣 — XML 正像 Unix 家族的作業系統，大小寫分得很嚴；而 HTML 則像 Windows 和 MacOS，檔名大小寫無所謂。這點還請讀友們特別注意。

2.8 CDATA 區

在 HTML 文件中，每當我們要舉例或附上原始碼時，我們會把它放到<pre> ... </pre> 或 <xmp> ... </xmp> 區塊中。在 XML 中，要達到這樣的功能，要用 CDATA （讀作：C data)。CDATA 中的 C 是 character （字）的縮寫。在 XML 中，所有 Unicode[3] 中定義的字碼，包括中文字，都算是合格的字，而不狹窄地局限於西文字母。那麼到底該怎麼寫呢？請看：

```
<![CDATA[
    小蜜蜂，嗡嗡嗡        &; ..... zzzzz

飛到西 <<<
                                    >>>>>> 又飛到東
]]>
```

[1] XML DTD 的語法是直接繼承自 SGML 。

CDATA 區以 <![CDATA[爲起始，]]> 爲終了。區塊內容中唯一不能包含的，正是]]> 這個終了信號。其他資料，只要是合格的 Unicode[3] 字，都可以自由放置其中。

「哇！設計成這樣，誰記得住？」

其實這正是它設計巧妙之處。就是要故意設計成這樣，才能將無心的錯誤減到最低。我們看 HTML 中的 <pre>...</pre> 區塊就很煩，在結尾的 </pre> 之前，必須自己隨時注意，把所有的 < 符號「跳脫」爲 < ，否則瀏覽器有可能把這些 < 號誤當成是一個新標籤的開始，而產生意想不到的後果。 XML 中故意設計出像]]> 這種不太可能在日常文件中出現的符號排列，讓我們在寫 CDATA 區時方便不少，不用記這個又忘那個的。

我們在前面第 2.7 節中剛談過，XML 中要區分大小寫，因此 <![CDATA[不可寫爲 <![cdata[或 <![Cdata[。

依標準規定，出現在 CDATA 區中的資料，解析器在解析時不許亂碰，而要原封不動、一五一十地交給下游的程式，嚴格的程度，就連任何在區塊起始和結尾處的空白、換行字元也無法悻免。請比較：

```
<![CDATA[我是區塊中的第一行
我是第二行
]]>
```

和：

```
<![CDATA[
我現在被擠到區塊中的第二行了 :-(
我被擠到第三行了
]]>
```

看得出其中的不同吧？結論：如果您不想在文件的每個 CDATA 區之前多一個空行的話（譬如有美觀上的考量），則應採用以上第一種寫法。

2.9 一空兩空大不同

在討論 XML 對空白字元的處理態度之前，我們最好先對「空白字元」作明確的定

義。XML 中把空白字元定義爲 space 、tab 、CR ，和 LF 這四個。對 space 和 tab 不需要多廢話，大家都知道這兩鍵。但 CR/LF 則要稍加說明一下。

CR 代表 Carriage Return，是打字機時代遺留下來的稱呼。這個 ASCII 字元通常是隱形的；它是 MacOS 平台上的換行記號。LF 是 Line Feed 的縮寫，是 Unix 上的換行記號。DOS/Windows 平台則使用一個 CR，後頭緊接著一個 LF 來標示換行。

上頭提到 CDATA 區中對空白和換行字元從嚴辦理，這是可以理解的。那麼在 CDATA 區以外，譬如在 <元素> . . . </元素> 中的文字內容呢？

在 HTML 中，文字內容中可以「一空」、「兩空」，甚至「三連空」、「多連空」，出來的結果都會一樣，因爲不管連續有幾個空白，一律都當成是一個，這是因爲 HTML 標準這樣規定。

HTML 標準：
http://www.w3.org/TR/REC-html40

但是 XML 正好相反。XML 中規定，所有位於標籤以外的空白，解析器要一個個忠實地交給下游程式作進一步處理。因爲這個限制，我們必須改變我們寫程式的習慣。在寫 HTML 標注文件時，不少人有適時換行，甚至縮排 (indent) 的好習慣，讓原始碼清楚易讀。大多數網頁開發工具，甚至還會替我們作 "pretty-print"，依文件的邏輯架構，和標籤出現的位置，作深淺不同的縮排. 但在 XML 中，如果把：

```
<作者>胭脂虎</作者>
```

寫成：

```
<作者>
        胭脂虎
</作者>
```

二者是不一樣的，因爲後者多了一個 tab 字元（「胭」字前的縮排），和兩個換行記號（分別在 <作者> 及「虎」字之後）。如果把上面第一種寫法中的「胭脂虎」前後各加上一個空白，把文字和標籤隔得開一點，即：

```
<作者> 胭脂虎 </作者>
```

，那麼解析器 parse 出來的，又會是另一種結果。

「爲什麼要這樣規定？像 HTML 那樣規定不是很好嗎？」

這是因爲考慮到，有的 XML 文件可能需要保留空白字元，譬如詩詞一類的內容，空白都有其存在的價值及特別意義。

如果我們想明確地告訴 XML 程式，不要隨便把空白去掉，而要尊重原著的精神，可以在標籤中加入 `xml:space` 這個 XML 內定的屬性，像這樣：

```
<詩句 xml:space="preserve">
小雨傘啊！          小雨傘
        一隻小雨傘
</詩句>
```

如果我們想刻意讓讀者看到這樣精雕細啄的詩句排列的話 ;-)。

2.10 PI 與樣規鍊結

XML 中還有一種標注，叫 PI，是 Processing Instruction 的縮寫。 PI 的標注是以 "<?" 開頭，"?>" 結尾（XML 宣告[2.1] 有時被人當成是一個 PI 的特例，但嚴格講起來不是）。通常 PI 是用來傳遞情報給解析器下游的程式的，譬如我們想用樣規[1.2] (style sheet) 來美化附表 2.1 中的陽春 XML 碼，不管是用 CSS[4.1] 或 XSL[7.1]，都必須有個機制，讓瀏覽器知道要到哪兒去找樣規。爲此，W3C 特別頒布了一個專爲連結樣規所設計的 PI，寫法爲：

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

"`xsl-stylesheet`" 這部分稱作 PI 的 目標 (target) 。以上的 PI 是用來告訴瀏覽器去抓一個叫 `style.css` 的 CSS 檔。如果要連結 XSL 樣規，就寫成：

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

這份鍊結樣規的標準，可自 `http://www.w3.org/TR/xml-stylesheet` 下載。

2.11 何去何從

學完了 XML 的基本語法，如果您對設計 DTD 特別感興趣的話，可以趁記憶猶新的時候，先跳到第 8 章閱讀。

3 Unicode 說分明

3.1 Unicode 簡介

Unicode （統一碼）顧名思義是一個將世界上幾十種紊亂的文字編碼整合在一起的努力。其幕後是由美國各大電腦廠商所組成的 Unicode 策進會來推動。目的在推廣一個世界通行的編碼體制，將所有世界上常用的文字都涵蓋進去，進而減少各電腦商開發國外市場時所遭遇到的問題（對我們這些常需要在各種文字碼之間作轉換的網路人，可說是感同身受）。

Unicode 策進會：http://www.unicode.org/

爲了要將成千上萬的文字統統收集到一個共通的編碼機制底下，在兼顧經濟的原則下，不管是東方或西方文字，每個字在 Unicode 中一律以兩個 bytes 來代表。這樣一來，就至少能有 $2^{16} = 65536$ 種不同的組合[1]，足以應付目前絕大多數場合的需要。

在與 Unicode 相關的各技術文件中，我們常會看到 ISO 10646 和 UCS 這兩個名詞。ISO 是位於瑞士的國際標準局的縮寫。它所頒布的第 10646 號標準叫 UCS (Universal Character Set)，也就是世界通用字集（以下簡稱「通用字集」）。UCS 通用字集的做法是用四個 bytes 來編碼，將世界上所有任何官定和商用的編碼大小通吃、一網打盡。值得慶幸的是，Unicode 策進會自 1991 年以來便和 ISO 的 UCS 小組密切配合，讓 Unicode 和 ISO 10646 保持一致。因此， Unicode 自 2.0 版開始，便和 ISO10646-1 使用完全相同的字庫和字碼。

康熙字典裡的中文字就有四萬七，如果再加上裡面沒有的簡體字，和一些筆劃寫法不同的日文字，那麼 Unicode 六萬多字的分配空間，光用來編漢字就已捉襟見

[1] 我說「至少」，是因爲 Unicode 中採用了一種機制，可以應付未來編碼空間用盡的問題。稍後會提到。

紂、不敷使用，哪還有餘地給泰文、阿拉伯文等幾十種其他文字使用？針對這個問題，Unicode （和 UCS） 採用了所謂的「中日韓文整合」(CJK Unification) 的解決方案，把中、日、韓文中筆劃近似的漢字，儘量以一個單碼來代表。例如草字頭在繁體中文裡要算成四劃（兩個十字），但在簡體中文及日文中只有三劃（中間一筆橫貫），Unicode 中忽略這種微小的差別，不再替所有帶有四劃草頭的字單獨設一個碼。但是如果一個字在簡、繁體中差別比較顯著、可能會對使用者造成文意理解上的困難或不便時，Unicode 中會爲簡、繁兩種版本個別編碼。例如言字旁在繁體中是七劃，但在簡體中只有兩劃（一點一彎勾），所有帶言字旁的字，在 Unicode 中都各有兩個不同的字碼來代表，一簡一繁。

經過「中日韓文整合」的漢字，在 Unicode 中稱做 Unihan（統漢字）。在 Unicode 2.1 及 3.0 標準中，共有二萬多個統漢字。統漢字並不包含日常生活中罕見的漢字，如康熙字典中的許多古字。

⚠ 注意

Big5 中常見的「裏」字（十六進制碼爲 `0xF9D8`），因爲沒有收錄在 Unicode (2.1, 3.0) 的標準中，因此建議您在做 Big5 和 Unicode 相互轉換之前，先把這個上下的「裏」換成左右的「裡」，否則「裏」字會變成 "?"，即 ASCII 中的問號（這是 Unicode 中的規定，找不到正確對應的字，一律換成問號）。

完整的 Unicode 3.0（最新版）統漢字資料庫可自 `ftp://ftp.unicode.org/Public/UNIDATA/Unihan.txt` 下載。要先警告您的是，這個 database *非常大*，超過 16Mb （搞不懂他們爲何不先壓縮一下）。其中不但包含了 Unicode 和 Big5、CNS、GB2312-80、GB2312-12345、SJIS . . . 等相互的對應，更連台灣電報碼、康熙字典中的位置等都詳細地收錄在裡面。除了要寫 Unicode 程式的讀者外，一般讀

者不需要下載這個 database。

3.2 字母遊戲

在我們對 Unicode 抽絲剝繭、剖析一番之前，我想把幾個常見的專有名詞及觀念先做個歸納和整理，這樣的話，讀者們在閱讀到後面的章節時，比較不會迷失方向，甚至可將下一節「血淋淋的情節」，整段跳過不讀（如果您對底下部分敘述看得有點不知所云，別擔心，只要大概看一下，有個概念即可）。

UTF 是 Unicode/UCS Transformation Format （通用字集／統一碼變換格式）的縮寫，Unicode 策進會推薦使用的是 UTF-8 和 UTF-16 這兩種格式。其中的 "8" 和 "16" 指的是 *bits* 數，而不是 *bytes*[2]。

UTF-16 基本上就是 Unicode 雙 byte 編碼的實現，再加上一個應付未來擴充需求的編碼機制（很少用；下面會談到）。

UTF-8 是一種*不等幅*的編碼方式，在 UTF-8 之下，英數字（即 ASCII 字元）保持原狀，完全不受影響（因此不需要做轉換）；但其他語文的資料則需透過程式來做轉換，而且會「變胖」，因爲每個字需要額外多用一或二個 bytes 來編碼。

UCS 通用字集中，訂有 UCS-2 和 UCS-4 等編碼方式，其中的 "2" 和 "4" 指的是 *bytes* 數，而不是 *bits*（對照： UTF-8/16）。那這兩個又和上頭的 UTF-8/16 有何不同呢？其實差別不大：

UCS-2 大體上就是 Unicode 採用的雙 byte 編碼，可以簡單地把它們想成是一樣的東西，不用爲此傷太多腦筋。

[2] 一個 byte 由 8 個 bits 所組成，相信大家都不陌生

UCS-4 是以四個 bytes 來代表一個字的編碼方式，就目前而言，在每個 UCS-2 碼之前補上兩個空白的 bytes，便可得到相對應的 UCS-4 碼。

好工具

爲了方便大家將文件轉換爲 UTF-8 格式，我寫了一個轉碼工具，叫 ccnv (enCoding CoNVerter)。這個小工具適合拿來作 UTF-8 和世界上幾十種編碼之間的轉換。最大的特色是可以一次對一個目錄底下*所有*的檔案作窮盡轉換。這是一個程式碼公開的免費程式，以 Java 寫成（目前只有指令界面），可以從兩隻老虎下載，內附 Big5 和 GB 中文的安裝方法和使用說明，在此就不再重覆。下載點：`http://2tigers.net/runpc/xml/runpc_xml_2.html`。

此外，有一個功能較多，但不是免費的 UTF-8 轉換工具，叫 uniconv ，有試用版下載：`http://rosette.basistech.com/demo.html`。

XML1.0 標準：`http://www.w3.org/TR/REC-xml`

XML1.0 標準中規定，XML 解析器至少須支援以 UTF-8 或 UTF-16 編碼的 Unicode 字串，而當 XML 宣告中沒有特別指明編碼（ `encoding` ）時，則一律以 Unicode 看待，軟體會自動偵測出文件是 UTF-8 或 UTF-16 。目前市面上幾家做得不錯的 XML 解析器，不但符合標準規定，支援 UTF-8/16，更支援 Big5、GB 等亞洲文字碼。這對我們來說，確實方便多多。

XML 標準中只明訂解析器[2.4]必須支援 UTF-8/16 ，而不苛求對其他編碼方式（如 Big5 等）提供支援，是有其道理的。因爲在 UTF-8/16 和各種國家、地區性的編碼，如 Big5/GB2312/SJIS 之間做轉換是很容易的事，一旦某個軟體支援 UTF-8/16（但並不直接支援 Big5、GB），我們在資料輸入或輸出時只要配合一道 UTF-8 ↔ Big5/GB 的轉換動作即可（上頭提供的 ccnv 轉碼工具可以派上用場）。某些情況下，甚至不需要刻意

做字碼轉換，譬如 perl 的XML::Parser 模組，雖然只能輸出 UTF-8，但可接受 Big5 的輸入。又如 Netscape、IE 等瀏覽器，能直接呈現 UTF-8 格式的文件，因此當輸出的資料是打算給瀏覽器去讀的話，就不一定要再轉換（除非有其他考量），而只要在文件的宣告中正確註明是用 UTF-8 即可。

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

遺憾的是，部分 XML 解析器和不少應用軟體[3]，無法正確處理 UTF-8 編碼的字串（或許他們暫時不擔心亞洲市場吧？）。爲什麼在這裡要特別強調 UTF-8 ？因爲這是最起碼的 — 如果某 XML 軟體連 UTF-8/16 編碼的資料都無法正確處理，那麼它支援 Big5、GB2312 中文的機率只怕更加渺茫了。當我們餵它含 UTF-8/Big5/GB 碼的資料時，通常會被賞給一段莫名其妙的錯誤訊息，要不就是一堆問號、或無法預料的結果（作者刻骨銘心的體驗）。

在 UTF-8 和我們慣用的 Big5/GB 之間做轉換儘管不方便，但至少它充分支援中文字。換句話說，如果某軟體能正確支援 UTF-8，它無形中就對我們有很大的利用價值。這也正是 XML 標準強制規定解析器要支援 UTF-8 的精神所在。

3.3 血淋淋的細節

我想藉這個機會，把跟 Unicode 相關的各個重要觀念，在底下一併解釋清楚。希望能對 XML 的使用者，乃至於寫轉碼程式、或學 Java 語言的讀者等都能有所助益。

如果您認爲您學 XML 的動機只是基於好奇，或將僅限於使用現成的工具，不太可能直接和程式碼打交道，那麼上面所提供的基本概念，或許就已經足用了。避免接觸底下血淋淋的科技細節，可幫助減少失眠、頭疼等癥狀。但是，如果您平常也寫一點程式，而且打算做各種 XML 的開發、應用的話，那麼建議您把它耐心看完，尤其是 UTF-8 編碼原理的部分，非常重要。事實上，底下將介紹的知識拿到許多其他的領域裡也一樣受用，像 Java 語言、Windows NT[4]，微軟 Office 軟體等，都大大地借重到 Unicode，

[3]也就是前面提過的，對解析器輸出的資料做進一步利用的下游軟體。
[4]這也就是爲什麼有些含中文字的 Java 軟體在 NT 下可以正確顯示，但拿到 Win95/98 上就不行了。

選它做內碼。

3.3.1 Unicode 中的空間分配

以下各 Unicode 區位碼皆以十六進制表示。

不意外，Unicode 的頭 256 個字符和 ISO 8859-1 （即俗稱的 Latin-1 ，西歐字母）完全相同，其中前半段就是 ASCII。這段是從 U+0000 到 U+00FF 。當然，每個 ISO 8859-1 碼必須在前面補上一個空 byte (0x00) 後才是相對的 Unicode 碼。

和我們有切身關係的 Unihan 統漢字，在 Unicode 中主要分布在 U+3400 到 U+9FFF 之間，此外， U+F900 和 U+FAFF 之間也有一些。事實上， Big5 和 GB2312 中的漢字和符號都在U+4E00 到 U+9FFF 這塊裡面。

3.3.2 UTF-8 的編碼原理和特性

知道了西歐字母和漢字在 Unicode 中的區位後，我們可以開始談 UTF-8 。在附圖 3.1 中， 每一個小方塊代表一個 bit ，綠色方塊所代表的位元值 — 0 或 1 — 是釘死不變的。藍色方塊所代表的各個空出來的位元，則拿來存放 Unicode 內碼。一群小方塊（八個）代表一個 byte。 UTF-8 共有三種可能的排列方式，各自需要用到一、二，或三個 bytes（所以才說它是不等幅）。每個 Unicode 字符按它在標準中分到的碼位，來決定這個字符在轉換成 UTF-8 時該用哪一種排列方式、以幾個 bytes 來編排。我們在圖中左方可以看到，在 U+0000 和 U+007F 之間的字符（即所有 ASCII 碼），是以第一種方式、單 byte 來表示，但因爲第一種形式的第一個 bit 是 0，所以 ASCII 碼（從 0x00 到 0x7F）根本不需要轉換，就自然是相對的 UTF-8 碼；反過來說，只要不是ASCII ，在 UTF-8 中就一定需要兩個以上的 bytes 來編碼。

Unicode 中落在 U+0080 – U+07FF 範圍間的字符，則以第二種形式來標示。至於所有其他剩下來的字符（包括所有的中文字），則一律以第三種形式來編排。您可以自

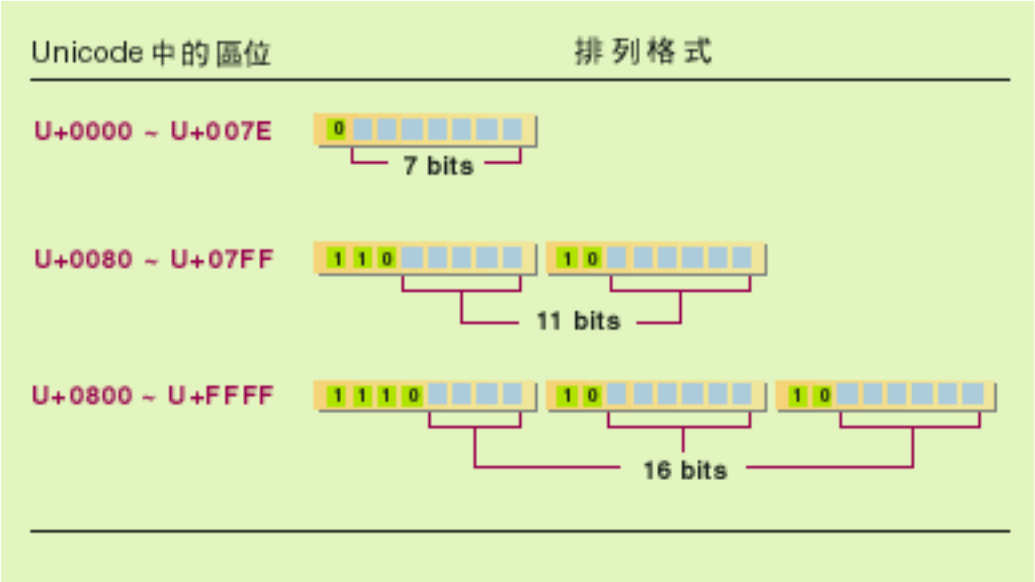

圖 3.1: UTF-8 的編碼原理

己算一算，看 UTF-8 的三種形式中各自提供的自由 bits （藍色小方塊）數，是否足以用來代表對應區位中的各 Unicode 碼，答案是剛好夠用。

那麼當程式在處理 UTF-8 編碼的文件時，要如何得知一個字符的疆界落在哪裡，還有到底它是以三種形式中的哪一種出現的呢？這就是圖 3.1 中那些綠色小方塊的作用了。每個以 UTF-8 編碼的字符，不管是以一、二，或三個 bytes 的形式出現，第一個 byte 的前端都清楚地標示了該字符的 byte 總數。譬如 `110` 中有兩個 `1` ，即代表這個字符是以第二種形式出現、由兩個 bytes 組成；而 `1110` 中有三個 `1`，暗示了這個字共佔用三個 bytes 。

每個多重 byte 的 UTF-8 碼有一個共通的特性，即其中第二和第三個 bytes ，一律以 `10` 這兩個 bits 起頭。由於其中最高的 bits[5] 總是設成 `1`，可以很容易和那些在 UTF-8 中只用到一個 byte 的 ASCII 字元分別開來，方便偵錯。

因爲上述的設計特點，UTF-8 和 Unicode 之間，可以很輕易地做*雙向*自由轉換，而

[5]又稱「最重要的 bits」；英文說 "most significant bits" 。

不會流失任何資料。事實上，這是所有 UTF 變換格式所必須滿足的基本特性，達不到這個要求的變換格式便沒資格叫 UTF。

3.3.3 UTF-8 的優點

現今大多電腦系統，仍是以 byte 爲單元來做存取。如果直接以 UTF-16（雙 byte）來存取資料，會產生許多問題。學過 C 的讀者都知道，`0x00` 這個字元是有特殊作用和意義的；而上面才剛講到，UTF-16 的頭 256 個字碼的第一個 byte，很不巧地，正是 `0x00`（第二個 byte 則是相對的 iso8859-1 字碼），因此如果用 UTF-16 來表示檔名、網址等，會惹出許多問題。而且不只是 `0x00`，其他還有幾個特殊字元，也都可能和許多現行的作業系統、函式館 (libraries) 等相衝。

此外，UTF-16 及許多亞洲地區現行的雙 byte 編碼方式，在應用上有些先天的缺陷：譬如，字與字之間的疆界不好找，程式在處理時必須從頭掃描，才能正確可靠地找出某個字的疆界，缺乏效率。此外，相信大家都有這樣的經驗：有時候中文文件中某個地方被錯殺一個 byte 時，*所有*在這個刀口之後的中文字都會壞掉，一直壞到下個英數字，或空白字元出現爲止。

以上這些問題，在 UTF-8 中都不存在。要找字與字的疆界很容易，只要看目前這個 byte 的開頭幾個 bits，看是 `0`、`110`、`1110`，還是 `10` 便可很快地找到現在這個字的疆界。如果遇到壞掉的字，也不會拖垮後頭一大票字。Unicode 標準中同時有明文規定，當 UTF-8 格式的文件中有不合理的組合出現時，該怎麼處置，例如碰到第一個 byte 是以 `1110` 起頭，但下一個 byte 卻以 `0`、而不是正確的 `10` 起頭。

3.3.4 UTF-8 的缺點

用膝蓋一想就知道，大多數文字在 UTF-8 之下都會膨脹，尤其是中、日、韓文，在最糟的情況下（即整篇文章不含任何西文時），會變成原來的 150%。對我們來說，的確有點虧。

但從另一個角度來看，如果改用 UTF-16 來做文字碼，固然不會增加亞洲文字所佔的空間（都是兩個 bytes ），但對西文來說，卻等於讓所有的文件大小，一律加倍，這比膨脹一倍半還慘。

我們設身處地站在他們的立場來想，也難怪許多西方世界的軟體業者和程式設計師對 Unicode 興趣缺缺，尤其是在過去，當記憶體和硬碟價格沒像近兩年這麼便宜的時候。

有了 UTF-8，爲數衆多的英文文件，不需任何轉換，就自然符合 UTF-8，這對向英文世界的軟體工作者促銷 Unicode 有很大的幫助，畢竟當今網路上大多數電子文件，都是英文。

因此，從讓 Unicode 在世界上快速、大量推廣的觀點來看，UTF-8 不失爲一個可行的折衷方案，它的優點似乎大過於缺點。

3.3.5 UTF-16 中的代理對

上頭提到，ISO 10646/UCS 使用四個 bytes 來編碼，因此要應付未來的擴充需求可說綽綽有餘。相對地，Unicode 只用兩個 bytes，不但空間很容易用盡，而且如果要在未來繼續和 ISO 10646/UCS 保持融通，勢必需要一種機制來彌補兩個 bytes 的不足。代理對 (surrogate pairs) 的設計便在這個背景下應運而生。

Unicode 將範圍在 U+D800 到 U+DFFF 之間的區域保留給代理對使用。這個區域又拆分成兩部分，稱做「高低部」，第一部分（高部）是從 U+D800 到 U+DBFF ，第二部分（低部）則從剩下的 U+DC00 到 U+DFFF，高低兩部各有 1024 個碼位。因此透過這個機制，Unicode 便可多容納一百多萬個字 (1024×1024)。當然，這些用代理對機制來編碼的字必須多用一個 Unicode 的基本單元（所以才叫代理對），也就是一共要四個 bytes，比較不經濟。

代理對的編碼機制，和原先不需要使用代理對的六萬三千多個基本 Unicode 碼，合起來叫 UTF-16。

您或許會奇怪，既然代理對是設計來應付未來字數擴充的需要，那爲什麼不讓高、低兩部的區位重疊？也就是讓兩部都可充分使用從 U+D800 到 U+DFFF 這整塊保留區域，爲什麼要區隔？因爲這麼一來，字的疆界很容易掌握，同時韌性高 — 即使有一、兩個 bytes 被錯殺，也不會後面跟著全死。

在現行的 Unicode 標準（2.1 和 3.0 版）中，並沒有動用到代理對來編字（接下來幾年內大概也不會用到），因爲代理對以外的六萬多區位，已經足以應付今日大多數場合的需要。儘管如此，我們可以利用其中的私用區，這是下一小節的主題。

3.3.6 私用區

Unicode 中保留了三塊私用區，寫程式的人可以拿來自由運用。其中兩塊在代理對的區域中，須配合代理對的編碼方式（即 UTF-16）來使用。如果我們的程式需要用到 Unicode 中沒有收錄的古字或特殊符號，就可以利用私用區來替那些字符做編碼。

Unicode 中第一塊私用區從 U+E000 到 U+F8FF，共 6,400 個區位。代理對中的私用區則包括 0xF0000 -- 0xFFFFD 和 0x100000 -- 0x10FFFD，約十三萬個區位。

4 XML 化妝術

4.1 CSS 入門

關於 XML + DOM + CSS ，以及 XML + CSS + 中文標籤在瀏覽器上的應用實例，請暫時先參考我在 Run!PC 讀者服務專區中的例子： http://2tigers.net/runpc/xml/runpc_xml_3.html。

此外，我在第 70 期（1999 年 11 月份）的 Run!PC 雜誌中有一篇 DOM 和 CSS 的入門指引，有興趣的讀者請參考。

Run!PC 雜誌：http://www.runpc.com.tw

4.1.1 選擇式 (selector)

待續。

5 名稱空間

名稱空間 (namespace) 的概念，對 XML 非常重要，如果少了它，XML 的應用範圍會大大地受到限制。如果您會用 Perl 或 Java ，那麼您很可能已經在使用名稱空間（而不自知？）了[1]。

名稱空間的規定，並沒有收錄在 XML 1.0 的標準中，而是透過後來一份單獨的標準來做增補，這份標準的全名叫 "Namespaces in XML" 。

XML 1.0 的標準：
http://www.w3.org/TR/REC-xml
Namespaces in XML：
http://www.w3.org/TR/REC-xml-names

5.1 爲什麼需要名稱空間

假設我們有個 XML 的電子通訊錄，專門用來存放客戶資料，記載平日來往的公司客戶：

```
<客戶名單>
  <客戶> <!-- 客戶甲 -->
    <名稱>新祥發</名稱>
    <地址>...</地址>
    <電話>...</電話>
    <fax>...</fax>
    ......
  </客戶>
  <客戶> <!-- 客戶乙 -->
    ......
</客戶名單>
```

不過有些客戶是業務專員帶進來的，通常在和這些客戶的連絡方面，都是透過各負責的業務專員。至於業務專員的連絡電話、email 等，都詳細記錄在公司的另一份 XML 文件 — 「員工名單」中：

[1] 在這兩個語言中，名稱空間是透過 package 來實現的。

```
<員工名單>
  <員工> <!-- 員工甲 -->
    <姓名>...</姓名>
    <部門>業務部</部門>
    <職位>專員</職位>
    <月薪>36000<月薪>
    <電話>
      <分機>...</分機>
      <大哥大>...</大哥大>
    </電話>
    <email>...</email>
    <fax>...</fax>
    ......
  </員工>
```

爲了連絡方便，我們想在 <客戶名單> 中，增加一項 <連絡人> ，用來存放業務專員的資料，這樣當我們要連絡由專員來負責的客戶時，便可以馬上知道該找哪位專員，打哪個電話。至於業務專員的連絡資料，則可以直接從 <員工名單> 中萃取；換句話說，就是把 <員工名單> 中的部分資料，和 <客戶名單> *結合* 在一起，像這樣：

```
<客戶名單>
  <客戶> <!-- 客戶甲 -->
    <名稱>新祥發</名稱>
    <地址>...</地址>
    <電話>...</電話>
    ......
    <連絡人>
      <姓名>...</姓名>
      <電話>
        <分機>...</分機>
        <大哥大>...</大哥大>
      </電話>
      ......
    </連絡人>
    ......
```

等等！「電話」這個元素[2.2]重複使用了！換句話說，「客戶名單」和「員工名單」這兩套 XML 語彙[p.29]，如果放到一塊用會打架。雖然對人腦來說，藉著分析上下文，可以很容易將兩個出處不同的 <電話> 區別開；但對電腦程式來說，恐怕就不見得那麼容易了。因此，在這種情況下，我們需要的是一套簡單明瞭的辦法，讓機器在處理的時

候，不用大傷腦筋。即使沒有任何名稱上的衝突，我們仍舊需要一個簡單的方法，讓電腦知道，哪些元素來自於哪個語彙，以便使用該語彙的 DTD[8] 來確認文件結構的正確性。

有的讀者可能已經想到一個可行的辦法了，那就是把 <客戶> 底下，一些太過籠統的子元素名，像「名稱」、「地址」、「電話」等，一律冠上「客戶」兩字，也就是把它們分別改成「客戶名稱」、「客戶地址」、「客戶電話」，把名子定的精確一點。此外，<員工名單> 中的一些元素名也可以照辦，譬如「姓名」和「電話」可以分別改爲「員工姓名」和「員工電話」，這樣就可以大大降低名稱衝突的機率了。

這的確是個可行的辦法，不過這麼做必須更動 DTD 裡的定義，但不見得所有的 DTD 都是由我們親手設計的[2]，不是說改就改；就算眞改了，我們依舊無法確保今後不會再遇到其他外來的同名元素。

名稱空間的設計，便在這樣的背景下應運而生。它的概念非常直接 — 如果每套 XML 語彙[p.29]，都各自用一個*獨一無二的標誌*來代表，並且在使用時，把這個標誌和語彙中的元素、屬性名連在一塊兒使用，就絕對不會和其他語彙扯不清了，因爲每個語彙中的名稱，都已經先被該語彙的獨特標示碼給修飾、限定 (qualify) 住了。其實這個解決方案，和上面提到、修改標籤名稱的做法，有些類似之處。剛才我們是把「客戶」二字加在意義含糊的名稱，如「地址」、「電話」之前，讓「客戶名單」這個語彙裡的元素名變得比較獨特，進而減少和其他語彙起衝突的機會。而名稱空間的做法，則是讓編寫 XML 文件的人替文件中所用到的語彙，自行指定標示碼。

一個獨特的標示碼，代表一套語彙；各語彙中的名稱，因而能各得其所，有自己的活動空間。「名稱空間」的命名，正是由此而來。

[2]事實上，就像軟體一樣，大多數人不會去碰 DTD 設計的東西，而會直接利用現成的 DTD 所定義出來的語彙，譬如由一個同業工會制訂出來的。

5.2 名稱空間的長像

咱們先來看看， XML 到底選用什麼來做名稱空間的標示碼。

5.2.1 標示物

上面已經提到，標示代碼最重要的特性是要能獨一無二。這可以藉由很多種方式來達到，譬如目前 Internet 上的網域名，便是獨特的，像 "2T.com" 已被人搶先一步註册走了 :-/，我就不可能再申請一樣的名子，而只得改用 "2Ti.com" 。註册固然是一種方法，但我們想想，如果 XML 名稱空間也要仿照 DNS 域名系統那樣，由專責機構來分配、管理，那不但會變得很複雜，而且會很沒效率；使用前得先註册，甚至還得付費，搞不好連大家爭著搶好名子的亂象，都會重演 ;-)。

XML 採用了一個聰明、簡便的方法來指定名稱空間：既然網域名已經是獨特的，那何不乾脆直接利用，把基礎建在它上頭，用*網址*來代表名稱空間呢[3] ？嗯，出這招太極拳，果然是高！這樣一來，責任便全「推派」給各公司、機關來負責。因爲各機關對放在自己網域底下的網址，應該有完全的主控權，所以一切名稱空間標示碼的指定工作，就統統交給編寫 XML 文件的人來決定。

至於網址的選擇問題，雖然名稱空間的原始目的，只是用來對一份文件中不同的語彙加以區隔，指定哪個網址不見得頂重要，只要能區隔開來就好，所用的網址甚至不需要存在！，但在此還是建議您養成好習慣 — 不歸我們的網域，在沒徵得人家允許之前，不要亂「借用」，譬如拿 W3C 或其他知名公司的網域名來瞎編一個網址，給自己設計的語彙當名稱空間用，就不是很好的做法[4]。大家都應該只用屬於自己 homepage 底下的網址。

[3] Java 的名稱空間也是用類似的方法，直接架構在 DNS 域名上。
[4] 筆者的確見過有人這麼做。

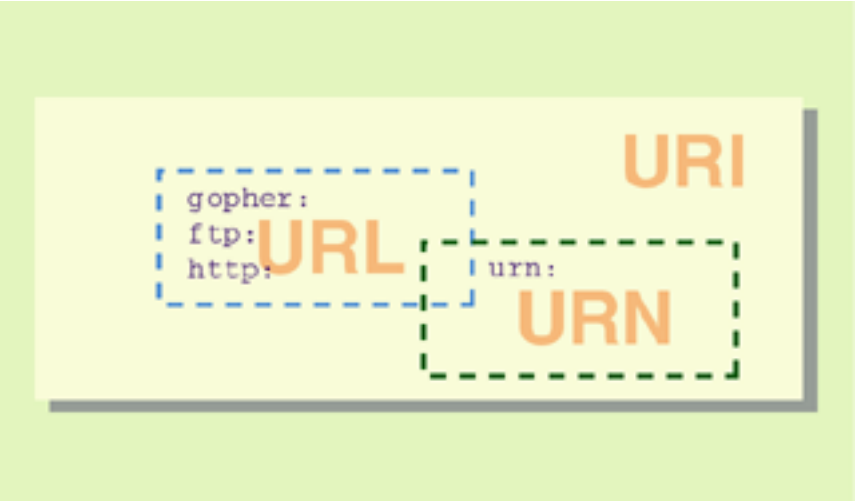

圖 5.1: URL 、 URN ，和 URI 的關係

5.2.2 URL 、 URN 、 URI — 別搞迷糊了

我們慣稱的「網址」，大致上是相當於 URL 。其實名稱空間的規定中用的是 URI ，但爲了讓讀者容易理解，我刻意在上頭使用大家比較熟悉的「網址」一詞。URI 全名叫 Uniform Resource Identifier （統一資源標示碼），這是一個 Internet 標準，記載於 RFC 2396；至於一般人較爲熟知的 URL 一詞，則是 Uniform Resource Locator （統一資源定位碼）的縮寫。那 URL 和 URI 除了名子外，還有啥不同呢？請看圖 5.1 中的集合關係。基本上，URI 比較廣義， 泛指所有以字串標示的網路資源，範圍涵蓋了 URL 和 URN。URL 指的是標有通信協定（如 HTTP 、 FTP 、 GOPHER ）的字串[5]。 URN (Uniform Resource Name) 則通常用來標示持久[6]、而且有專責機構負責的資源，譬如圖書館的圖書總目。

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

在這本書中，「網址」一詞用得比較鬆散，可能代表 URL ，也可能指 URI 。

[5]有學問的說法爲「明確標示出資源的取得途徑及管道的代碼」。
[6]與網站、網頁的壽命相比。

5.2.3 名稱空間實例

在討論名稱空間的宣告方式之前，先請您作個「成品預覽」：

```
<?xml version="1.0" encoding="UTF-8"?>
<k:客戶名單  xmlns:k="http://foo.bar.com/xml/customer.dtd"
                 xmlns:員工="http://foo.bar.com/xml/employee.dtd"> A
  <k:客戶> <!-- 客戶甲 -->
    <k:名稱>新祥發</k:名稱>
    <k:地址>...</k:地址>
    <k:電話>...</k:電話>
    ......
    <k:連絡人>
      <員工:姓名>...</員工:姓名> B
      <員工:電話>
        <員工:分機>...</員工:分機>
        <員工:大哥大>...</員工:大哥大>
      </員工:電話>
      <員工:email>...</員工:email>
    </k:連絡人>
  </k:客戶>
  <!-- 客戶乙 -->
  <客戶 xmlns="http://foo.bar.com/xml/customer.dtd"> C
    <名稱>同仁堂</名稱>
    <地址>...</地址>
    <電話>...</電話>
    <fax>...</fax>
    <連絡人>
      <員工:姓名>...</員工:姓名>
      <電話 xmlns="http://foo.bar.com/xml/employee.dtd">
        <分機>...</分機> D
        <大哥大>...</大哥大>
      </電話>
      <員工:email>...</員工:email>
    </連絡人>
  </客戶>
</k:客戶名單>
```

如同上面的例子，在標注名稱空間的 XML 文件中，您會看到一些元素、屬性名[2.2]，是用冒號把兩個名子連在一起的。

5.3 名稱空間的宣告

5.3.1 前置字串

我們已經知道，名稱空間的原理是藉著「獨特標示碼+名稱」的方式，讓每個 XML 語彙中的元素、屬性名都能有自己的小天地，而不會和其他語彙中的名子互搶地盤；換句話說，冠上了標示碼，每個名子就可變得獨一無二。

您可能已經一邊抓頭，一邊在問了：「上面不是說到，XML 選用 URI 來做獨特的標示碼嗎？可是一般的 URI 、網址都蠻長的，如果直接拿來放在元素、屬性名之前使用，是不是太冗長了點？如果能用個簡短的代號來替代 URI 就好了。」

完全正確！URI 不但太長，而且裡面往往含有一些 XML 語法規定不准用作元素、屬性名的字元（譬如網址中一定都有的 "/" ），因此使用代號不僅是爲了便利，更是勢在必行。這樣的代號，在名稱空間的標準中叫「前置字串」(namespace prefix)，由編輯 XML 文件的人自由指定。前置字串只能含有 XML 標準中允許作屬性名的字，這包括了英文字母，和所有收錄在 Unicode[3] 中的漢字。還有，XML 標準把所有以 "xml" 這三個字母開頭的前置字串保留作特殊用途，所以使用者自訂的前置字串不許用這三個字母來起頭，不管是 "xml" 、 "XML" 、 "xMl" ...，什麼樣的大小寫組合都不可以。

用前置字串來代表名稱空間的標示網址，就好比我們用 Unix 上的 symlinks 、Windows 底下的 shortcuts （捷徑），或 MacOS 裡的 aliases 來代表檔案路徑一樣，只要能發揮指向的功能就好，取作什麼名字倒是其次。若是能簡短、達意當然是最好，這樣既方便書寫，又利於閱讀。事實上，XML 軟體在處理文件之前，會先將所有前置字串還原成它們所代表的 URI。

那前置字串和它所代表的名稱空間，要透過什麼樣的宣告方式，才能把它們關聯在一起呢？這得透過一種特別的屬性來達成 — 這類屬性的屬性名一律以 "`xmlns:`前置字串" 的形式出現[7]，而屬性值則正是該前置字串所要代表的名稱空間，

[7] `xmlns` 中的 `ns` 是 NameSpace 的縮寫。

也就是一個 URI 。屬性名中的 xmlns 和前置字串間，必須以英文的冒號 (:) 相連。因爲冒號已經被選作分隔記號，有特殊的功用，所以前置字串中不可以再用冒號。一個完整的名稱空間宣告，在一份 XML 文件中寫起來就像這樣：

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

> ⚠ 注意
>
> 您可能會在一些已經過時的 XML 書籍或文件中看到像 <?xml:stylesheet ... ?> 的 PI[2.10]，名稱裡面有冒號，和我們在第 2 章中介紹的 <?xml-stylesheet ... ?> （使用 - 號）有出入。這是歷史的遺跡 — stylesheet PI 之所以捨棄冒號、改用連字號，正是因爲和名稱空間的設計相衝突。

因爲名稱空間是藉著屬性來作宣告，所以必須依附在一個元素標籤裡面。至於該放在哪個元素裡，要看實際需要而定，我們稍後會針對這個問題作說明。

要在一份 XML 文件中，同時使用多種語彙（這正是名稱空間的目的），只要先將各語彙的名稱空間和前置字串定義、宣告好，然後視需要，在元素或屬性名之前，冠上適當的前置字串，並將二者以冒號隔開即可。這樣一來，不但哪個名稱來自哪個語彙，劃分得一清二楚，更不用擔心會有同名的衝突。有了這樣的機制，前面提到的「客戶名單」和「員工名單」結合使用的問題，可以圓滿地得到解決了：

```
<k:客戶名單 xmlns:k="http://foo.bar.com/xml/customer.dtd"
            xmlns:y="http://foo.bar.com/xml/employee.dtd" >
  <k:客戶>
    <k:名稱>新祥發</k:名稱>
    <k:地址>...</k:地址>
    <k:電話>...</k:電話>
    ......
```

```
    <k:連絡人>
      <y:姓名>...</y:姓名>
      <y:電話>
        <y:分機>...</y:分機>
        <y:大哥大>...</y:大哥大>
      </y:電話>
      <y:email>...</y:email>
    </k:連絡人>
  </k:客戶>
  ......
</k:客戶名單>
```

在上例中，http://foo.bar.com/xml/customer.dtd 是我們給「客戶名單」這個 XML 語彙指定的名稱空間標示碼，http://foo.bar.com/xml/employee.dtd 是「員工名單」的標示碼。爲了打字省力起見，我們用 k 這個前置字串[8]來代表「客戶名單」的標示網址 (xmlns:k="http://..."); 用 y 來代表「員工名單」(xmlns:y="...")。我們同時也看到，一個元素裡面可以放置多個 xmlns 的屬性宣告。一旦宣告好了以後，只要將各個元素名都冠上適當的前置字串，哪個元素歸屬哪個語彙，分得清清楚楚。

5.4 名稱空間的範疇

範疇 (scope) 的概念，對程式設計非常重要。假設我們把上面的例子改寫成：

```
<k:客戶名單>
  <k:客戶>
    <k:名稱>新祥發</k:名稱>
    ......
    <k:連絡人 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:電話>
        <y:分機>...</y:分機>
        <y:大哥大>...</y:大哥大>
      </y:電話>
      <y:email>...</y:email>
```

[8]字串 (string) 可以只含有一個、甚至零個字元（空字串）。

```
    </k:連絡人>
  </k:客戶>
  ......
</k:客戶名單>
```

也就是把名稱空間延後到 <連絡人> 的地方才宣告。這樣一改後，雖然仍舊可達到類似的功效，把兩套來源不同的元素區分開，但名稱空間所涵蓋的*範疇*已改變，所有紅字部分必須去掉，否則就是錯誤。爲什麼？因爲 <客戶名單> 和 <客戶> 分別是 <連絡人> 的祖母和母元素，層級要比 <連絡人> 高， <連絡人> 罩不住她們 ;-)，但是要罩層級比它低的子、孫元素，如 <姓名> 、 <大哥大> ，則不是問題。

因爲名稱空間有範疇的特性，所以我們在元素中安插名稱空間宣告時，必先將涵蓋層級的問題考慮進去。最簡單的方法，當然是在最外圍的根元素標籤裡，先把所有的名稱空間和要指定的前置字串都一一定義好。這正是前一節例子中的做法。如果要在文件中途處對某名稱空間作宣告，則最好確定所有來自這個空間的元素，都能被宣告所在的元素所涵蓋，否則無非是在編寫 XML 時，給自己多添麻煩，因爲出了這個範疇後，每當需要再用到同一個名稱空間時，又得寫一長串，再宣告一次。

5.5 預設的名稱空間

C

XML 名稱空間有預設 (default) 的概念。沒有冠上前置字串和冒號的元素、屬性名，一律視爲在預設的名稱空間底下。如同一般的名稱空間，我們可以透過宣告，把預設的名稱空間標示出來。它的寫法和前面介紹過的 `xmlns:前置字串="URI"` 很像，但少了冒號和前置字串這部分，也就是變成 `xmlns="URI"`；換句話說，我們可以把預設名稱空間的前置字串，想成是空字串。上面的範例，如果改用預設空間的做法來寫，就成了：

```
<客戶名單 xmlns="http://foo.bar.com/xml/customer.dtd"
          xmlns:y="http://foo.bar.com/xml/employee.dtd">
  <客戶>
    <名稱>新祥發</名稱>
    <地址>...</地址>
    <電話>...</電話>
```

```
    ......
    <連絡人>
      <y:姓名>...</y:姓名>
      <y:電話>
        <y:分機>...</y:分機>
        <y:大哥大>...</y:大哥大>
      </y:電話>
      <y:email>...</y:email>
    </連絡人>
  </客戶>
  ......
</客戶名單>
```

「客戶名單」這套語彙現在被指定爲預設的名稱空間，任何沒有前置字串和冒號的元素名，都會被認作是屬於這個空間。

預設的名稱空間，可以在文件中變來變去，就像一個前置字串在同一份文件中，可以先後和不同的 URI 串連一樣，並非宣告過一次之後，就不能再用，只不過後面的宣告會覆蓋前面同名的宣告，譬如：

```
<客戶名單 xmlns="http://foo.bar.com/xml/customer.dtd">
  <客戶>
    <名稱>傑兒克</名稱>
    <地址>...</地址>
    <電話>...</電話>
    <網址>...</網址>
    ......
    <附註>
      <!-- 改用 HTML 來作預設空間 -->
      <p xmlns="http://www.w3.org/TR/REC-html40">
        此客戶相當難纏，而且<b>非常搗</b>。和他們老闆應酬之前，最好先到他的
        <a href="http://jerk.com/boss/">個人網站</a>去看看他最近喜歡玩什麼。
      </p>
      <!-- 離開 HTML 名稱空間的範疇 -->
    </附註>
    ......
```

5.5.1 預設空間與範疇聯合運用

上面的範例巧妙地利用到預設空間和範疇的特性，把「客戶名單」和 HTML 兩套語彙各自的範疇，劃分的一清二楚：所有在 <附註> 裡面的部分，不包含 <附註> ，隸屬於

HTML 的名稱空間，出了 <附註> 以後，「客戶名單」的名稱空間又重新生效，恢復爲文件的預設空間。

這個範例同時展示了另一個實用的技巧 — 超連結。雖然 XML 有自己的連結方式，合稱爲 XLink/XPointer ，在功能、花樣上要比 HTML 的簡單連結模式強上十倍百倍；但是目前這兩套超連結語法都還停留在草案階段，尚未定案爲正式標準，而瀏覽器的支援也非常有限。因此借用 HTML 的 <a> 標籤來作連結，不失爲一個很好的過渡期解決方案。

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

把 HTML 標籤適時拉進 XML 這招非常好用。不只是能作簡單的超連結，還可以做表單 (form)。因爲 XML 並沒有內建表單的功能[9]，目前也沒有一個通行的 "FormML" 標準，因此要在 XML 文件中附加填表功能，並且能在瀏覽器中正確地顯現、送出，最方便的方法就是透過名稱空間，把 HTML 的 <form> 、 <input> 等元素內嵌到 XML 檔案裡頭去。IE5 和目前仍在發展的 Mozilla (Netscape 5) 都支援這樣的做法，一切就如同在 HTML 中一樣，也可以配合 ECMAScript[p.14] （即 JavaScript）來執行，作各種事件處理動作 (event handling) ，如 onClick 、 onMouseOver 等。

Mozilla : http://www.mozilla.org

下面的 XML 碼，如果要在 IE5 中正常運作，必須指定一個外部的 CSS 樣規[4.1]，這個 CSS 檔倒不見得一定要存在，但是如果沒有加上那行 PI[2.10]， IE5 會用它內建的 CSS 樣規來呈現 XML 文件，表單和鍊結的效果都會出不來。

```
<?xml version="1.0" encoding="Big5" ?>
<!-- IE5 必須配合外部 CSS -->
<?xml-stylesheet href="ie5_needs.css" type="text/css" ?>
<測試 xmlns="http://put-your-URL-here"
        xmlns:html="http://www.w3.org/TR/REC-html40">

<!-- 借用 HTML 標籤來鏈結 -->
<html:a href="http://2tigers.net">連到兩隻老虎</html:a>

<!-- 借用 HTML 表單來做按鈕 -->
<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，因爲 XML 只不過是個超語言。

```
<!-- Mozilla 也支援 XLink 舊標準中的 simple link 機制（已過時） -->
<鏈結 xml:link="simple"
        href="http://2Ti.com">這樣也可以連到 2T</鏈結>
</測試>
```

要請您注意的是，上例中用來代表 HTML 名稱空間的網址不能寫錯。

學到這裡，您應該已經能感受到名稱空間所提供的強大功能。有了名稱空間後，各元素、屬性便可跨越文件的疆界，而不再只是單純寄居在某套語彙的文件格式中；語彙也同時搖身一變，成爲全球性的*模組*，可以和其他的語彙、模組任意組合、搭配，供各式各樣的應用文件調借使用。許多重要的 XML 應用，包括 SVG*[1.3.3]* 和 XSL*[7.1]* 等，都是架構在名稱空間的基礎上。如果您用一個純文字編輯器去瞧一瞧微軟 Office 2000 所發佈出來的 HTML 碼，您會發覺它裡面的 XML 部分用名稱空間也用得很凶。

6 下一代的 HTML — XHTML

「HTML 4.0 已經推出好一段時間了。什麼時候可以看到 5.0 出來？」

不會有第 5 版了！因爲 HTML 的任務，已經交給 XHTML 來接棒了。

HTML 4.0：
http://www.w3.org/TR/REC-html40

6.1 什麼是 XHTML

XHTML 最早叫“HTML in XML”。簡單講，就是把過去以 SGML 定義的 HTML，改用 XML 來重新定義。XHTML 就像其他數不清的「X 叉叉 L」一樣，是 XML 超語言的一項應用。所有 XHTML 的標籤都摹擬既有的 HTML 4.0 標籤來定義，各元素和屬性[2.2]的名稱和用法幾乎完全不變。不過因爲變成了 XML，有些地方必須遵照 XML 的規矩，嚴格執行，不能再像過去那麼隨便。

XHTML 1.0 這套標準，在 1999 年 8 月 24 日升格爲建議標準 (proposed recommendation)。本章便是依據這份最新的標準寫成。不過成爲建議標準，並不代表就一定能順利成爲正式標準。事實上，XHTML 1.0 這份標準目前正被激烈地討論中，爭議的焦點之一是名稱空間[5]的解釋問題。暫且不管最後的結果如何，我們可以先深入了解 XHTML 發展的動機，和它試圖解決的問題。本書也會隨著最新的動態，適時更新。

XHTML 1.0：
http://www.w3.org/TR/xhtml1

6.2 XHTML 長得什麼樣子

照往例，先請您看一段 XHTML 原始碼。在腦中有個概略的輪廓就好，我們會在後面一一詳細解釋。迫不及待的讀者可以利用其中的色塊，直接跳到相關的章節去一探究竟。

```
<?xml version="1.0" encoding="Big5" ?> 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>幾隻老虎才夠看？</title> D
<meta http-equiv="Content-Type"
    content='text/html; charset="Big5"' /> 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>一個非常短的段落。</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 的各個特徵固然重要，但是先思索一個新科技是否具有實質上的意義，再去考慮學習的問題，可能會是比較理想的順序。就讓我們先來看看， XHTML 是否眞有存在的價值。

6.3 爲什麼需要 XHTML

6.3.1 HTML 的隱憂

HTML 在可預見的未來，仍將像目前一樣，在網路上扮演重要的角色。在未來的網路世界，不僅會存在大量的 HTML 文件，更會繼續不斷有新的 HTML 文件產生。但現在有一個問題存在 — 不曉得您知不知道，網路上大多數的 HTML 檔案，嚴格來講，都是不合

格的。

「哈！至少咱的一定合格，因爲我都是用那 DreamPage99 網頁編輯器做的，而且都用 Netscape 和 IE 檢查過，安啦！」

別太有把握哦！您的網頁，相信用 PC 上的瀏覽器去看是絕無問題的，但如果用明日的大哥大來看呢？

「大哥大？」

對，沒寫錯，是大哥大 — 歡迎來到*後 PC 時代*。

許多分析家預測，今後幾年，會有愈來愈多的小型器具、和自動化使用者代表程式[4]上網，爲人類做各式各樣的服務。有的甚至說，到了公元 2002 年，75% 的網頁都將透過這些非桌上型電腦來看[1]。

請別誤會，我並不是在鼓吹說， PC 將被淘汰。完全不！桌上型 PC 將繼續在未來扮演重要的角色。不同的是，將會有另一類完全不同的機器、異形，開始在網路上活耀起來。事實上，可以讓人 check email 的大哥大早就有了，而且已經有廠商宣布，將在近期推出附網頁瀏覽器的大哥大[2]。此外，從掌中型微電腦（像是最熱門的 Palm Pilot）的大興其道、寬頻時代的來臨，還有上網的電冰箱的出現，在在不難想像爲什麼分析家們會這麼預測。

HTML 瀏覽器之所以愈做愈肥，除了互拼功能外，爲了能儘量包容五花八門、邋遢含糊的 HTML 碼，也是主因。因此，用兩家瀏覽器能正確顯示的網頁並不代表它一定符合標準。更可怕的是，可能連一些網頁製作工具做出來的網頁，都不見得合格。在 PC 稱霸的今日，這不是很大的問題，因爲我們有足夠的記憶體和硬碟空間來飼養這些龐然巨物。但等不久的將來，當輕薄短小的網路工具、自動程式當家時，它們內建的網頁閱讀器、解析器是否還能像現在的大瀏覽器這樣，把不夠標準的網頁，照網頁設計者想表達的原意，順利地呈現出來呢？

[1] 直接引用自 XHTML 標準。
[2] Nokia 最近便發表了未來大哥大的雛型，不但可收看數位電視，更內建 Linux 作業系統和 Mozilla 瀏覽器，讓使用者遨遊網路。

XML 和 XHTML 前來解救。

我們知道，XML 的標準非常簡潔且嚴謹。從整份標準文件短短的三十多頁這點上，就可看得出。「簡而嚴」的特性，讓工具不必做得那麼肥，像各大公司出的 Java XML 解析器[2.4]，大小都在幾百 kByte 以下。這也是 XML 和 Java 爲什麼熱門的原因之一，因爲許多人預見網路小工具時代的來臨。而瘦瘦的 XML 解析器，要安裝在這些小工具上是再適合不過了。

6.3.2 XHTML — 既可輕薄短小，又可無限延伸

XHTML 不只提供我們一個環境，把不乾淨的 HTML 碼清一清；事實上，可大可小、能縮能脹的*彈性*，才眞正是 XHTML 的價值所在。

能縮 — 模組化

HTML 最早是個簡單的語言，設計給 CERN 高能物理中心的同仁之間，交換研究心得之用。但繼 WWW 風起雲湧、兩大瀏覽器互拼功能的連年征戰，加上網頁設計師、所見即所得編輯器爲達視覺效果，各施奇招的百般折磨之後，不管是 HTML 標準本身，或寫出來的 HTML 碼，早已被整得面目全非。雖然 HTML 在網路的發展史上，有著不可磨滅的功勞，但如果不想想辦法，後果不堪設想。

有什麼辦法可想？嗯，我們可以把龐大的 HTML 4.0 功能，按使用者需要和瀏覽器的能耐，打破、拆分成一塊塊的模組，每一組特定的功能，都由一個單獨的 DTD[8] 來負責定義。換句話說，就是把現有巨大的 HTML 4.0 DTD 模組化。每一組 DTD 只支援一部分的標籤。對不同的瀏覽工具，可以針對其個別的顯像特性、螢幕尺寸，設計出專用的 DTD ；譬如不適合顯示分割畫面 (frames) 的小電腦，就不要支援包含 frame 功能的 DTD 。這樣一來，不但大大減輕這些小工具的負擔，更讓它們能盡情發揮各自的特性。

事實上，模組化正是目前 W3C XHTML 小組的工作重點。雖然在現行的 XHTML 1.0 標準中，只照本宣科地訂出了三個 DTD （後面會談到），但是陸續發展的 XHTML 1.1

XHTML 1.1 : http://www.w3.org/TR/xhtml11

草案，則已積極開始制訂模組拼裝的原則，讓 XHTML 文件的編輯者和瀏覽器能按實際需要，套用不同組合的 DTD 。

能脹 — 擴充性

XHTML 名稱中的 X ，可不是隨隨便便冠上去的，而是眞正代表它具備「X」家族一脈相承的特性，也就是擴充性／延展性 (eXtensibility)。

過去在 HTML 時代，只有 W3C 標準會或極具影響力的瀏覽器業者（意譯：微軟和昔日的網景），才有辦法給它添加新的標籤，至於一般老百姓那想都別想。而且每定義一個新的的標籤，HTML DTD 就要更動一次，非常沒有效率。

改用 XHTML 以後，擴充困難的問題不再存在。就像所有 XML 文件一樣，XHTML 文件可以透過界定名稱空間[5]的方式，隨時按需要來徵調額外的標籤。譬如在以下的 XHTML 原始碼片段中，就借用了 MathML[1.3.1] 來表達數學式子。

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

```
<html xmlns="http://www.w3.org/TR/xhtml1">
  <head>
    <title>數學例子</title>
  </head>
  <body>
    <p>以下爲 MathML 標籤：</p>
    <!-- 開始進入 MathML 名稱空間 ... -->
    <math xmlns="http://www.w3.org/TR/REC-MathML">
      <apply> <log/>
        <logbase>
          <cn> 3 </cn>
        </logbase>
        <ci> x </ci>
      </apply>
    </math>
    <!-- 離開 MathML 名稱空間 -->
  </body>
</html>
```

薪火相傳

有的讀者可能還是會問：「XHTML 固然能縮能放，但這只不過是它從 XML 繼承的好處罷了。爲何不全面改用新的 XML 標籤，反而刻意去配合一個充滿舊包袱的 XHTML？」

因爲 XML *不會*全面取代 HTML 。有的文章，根本不需要大費周章地改用 XML 形式儲存，而可以長久保持 HTML 形式；就算 XML 在將來會完全取代 HTML ，今天佔上風的，仍是 HTML 。理由很簡單：當今大多數網頁設計者最熟悉的，仍是 HTML。此外，目前 XML 開發工具仍相當缺乏，也不夠普及（這點不禁令人回想起幾年前 HTML 工具不成熟的景象[3]），加上現今絕大多數的網頁瀏覽器，仍然只懂 HTML ，而渾然不知 XML 已誕生。至於要等新一代瀏覽器大量普及，則還要很長的一段時間。因此，不管是爲了確保明日的 HTML 文件，不會讓未來的迷你瀏覽器噎到；或是打算在這個標注語和瀏覽器新舊交替的過渡期，爲轉型到 XML 做準備，我們都可以善用 XHTML，來幫助我們達成這些目標。別忘了，XHTML 是當然的 XML。把 HTML 文件換成 XHTML 格式，便是確保未來*任何*替 XML 設計的網路器具，都能正確處理您的文件。這在網站未來化的腳步上，無疑向前邁進一大步。如果您正在設計一個新網站，更應優先考慮使用 XHTML。

6.4 XHTML 和 HTML 4.0 的差別

6.4.1 XHTML 幫您複習

在讀過前幾章介紹的 XML 語法和概念後，如果您到現在都還遲遲未動筆（或敲鍵盤），試著寫幾個 XML 標籤來實際演練一番（不應該哦！），現在正是重溫舊課的好機會。這也是爲什麼 XHTML 這個主題要刻意安插在這裡的原因。

[3] 儘管這麼比喻，但二者的本質卻不盡相同 — HTML 本身包含外觀的呈現，因此所見即所得、親善可人的編輯器就相形重要。雖然 XML 本身不考慮外觀，但如果能有好的 CSS 編修工具來配合，則也是一大福音。

有了 XML 的基礎[4]，再來瞧瞧 XHTML ，您會發覺，一切 XHTML 的規定，剎那間變得很容易理解，絲毫不感意外，因爲這些正是 XML 的規定。

好，就讓我們從一個已經認識 XML 的人的角度來想想，如果 HTML 要 XML 化，會有哪些地方需要調整。

6.4.2 格式正確原則對 HTML 的衝擊

我們在第 2.5 節曾談到，XML 文件要稱得上合格，必先格式正確 (well-formed)，也就是得滿足 XML 的基本語法。由於 HTML 標準本來在某些地方就比較自由，加上各大瀏覽器容錯度高，對 HTML 的句法不嚴格要求，造成目前的 HTML 文件中，有不少地方，在先天或後天上和 XML 的語法打架。其中最常見的包括：結尾標籤被省略、兩個標籤交叉，和屬性值沒有上引號。這些違反格式正確原則的狀況，在 XHTML 中一律不許出現，有幾項規定要強制執行：

結尾標籤不可省

HTML 標準中說，有些結尾標籤愛加不加隨便你，像 `</p>` 、 `</li>` 、 `</tr>` 、 `</th>` 、 `</td>` 、 `</dt>` 、 `</dd>` 等都是。但在 XML 和 XHTML 中，漏了結尾標籤整份文件就不及格，非寫不可，所以一切要像這樣：

```
<p>這是一個段落。
   下個段落雖空無一物，標籤仍然要成雙成對。
</p>

<p></p>
```

解開糾纏不清的標籤

標籤之間只許包含、但不許交叉是 XML 格式正確原則的另一項要求。嚴格講，這並不能算是 HTML 和 XML 相衝之處，因爲這在 HTML 中也不能算合格，只是各 HTML 瀏覽

[4]還有，假設您會用 HTML。

器對於標籤交叉的做法，常睜一隻眼閉一隻眼，導致在許多現有的 HTML 碼中，常可以見到像：

```
<i>斜體字<b>粗斜體字</i></b>
```

這樣兩個標籤糾纏在一起的邋遢寫法，結尾的 `</i>` 卻跑到 `<b>...</b>` 裡頭去了。

XML 解析器[2.4]可不吃這一套 — 要讓 HTML 碼成爲合格的 XHTML/XML 碼，我們必須反交叉、反打結：

```
<i>斜體字<b>粗斜體字</b></i>
```

把 `<b>...</b>` 完全包在 `<i>...</i>` 裡面，而不是一個在裡、一個在外。

空元素 XML 化

H

`<空元素/>`[2.5.1]標籤的寫法，是 XML 中的新發明。在 HTML 中，空元素標籤和其他標籤的長像並無不同，唯一的差別是空元素只有起始，卻沒有結尾標籤來和它配對，最常見的例子，不外乎 HTML 中的 `<br>` 和 `<img>` [5]。爲了滿足格式正確的要求，在 XHTML 中，`<br>` 這類的空元素必須改寫成 `<br/>` 或 `<br />`。

> ⚠ 注意
>
> 不要把上面提到的 `<p>` 和 `<tr>` 等標籤和 `<br>`、`<img>` 混爲一談。`<p>` 和 `<tr>` 不是空元素，只是大家寫 HTML 時，常將結尾的 `</p>` 和 `</tr>` 標籤偷懶不寫。兩者有區別。

屬性值一律上引號

F

記得在第 2.5.3 節曾談過，不管是字串或數字，XML 文件的屬性值一律要用引號括起來。因此在 XHTML 中，`<td width=100>` 的寫法不合格，要改成 `<td width="100">` 才能過關。

[5] `<img>` 雖內附屬性（`src`、`height`...），但仍算是空元素。

屬性名不可落單

HTML 中有一種屬性，它的值只有「是」或「不是」兩種狀態[6]。通常在遇到這類的屬性時，如果我們要註明它的值是肯定的，就把它的名稱放進標籤裡，但省略後面的等號和屬性值，像以下例子中的 `checked` 、 `selected` ，和 `compact`：

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

`<input>` 和 `<option>` 這兩個標籤，對常寫 HTML 表單、 CGI 程式的讀者一定都不陌生。這種屬性值省略的做法，還特別有個術語，叫屬性最小化 (attribute minimization)。

在 XML 一板一眼、要求一律使用「屬性名="屬性值"」的做法下，上面例子中被「最小化」的屬性，必須一一改寫爲：

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

也就是把屬性名拿來當屬性值，再複頌一遍就是了。

6.4.3 檢測、驗證、依據

上面所說的各項格式正確的規定，不過是所有 XML 文件都必須共同遵守的一些最起碼的要求。我們知道，在 XML 中，有法可證[2.6] (validity) 的概念，常和格式正確相提並論。那麼，XHTML 文件除了求格式正確外，是否也要有法可證呢？

那當然！

光是格式正確的文件根本不能算是 XHTML ，充其量只是合格的 XML 文件罷了。XHTML 因爲是直接利用現成的 HTML 標籤，所以，就像我們過去寫 HTML 文件時一樣，在用 XHTML 寫東西時，也必須確定文件中使用的標籤名，都是確實存在的

[6]寫程式的讀友都知道這叫 boolean。

HTML4.0/XHTML 標籤，還有標籤之間相互的關係、出現的順序，都不能胡來。這正是有法可證的意義所在。

選定、註明 DTD

我們在第 2.6 節提過，有法可證的「法」，指的是 DTD[8] ，要讓 XML 文件能透過某套 DTD 法則加以驗證之前，我們得先指定一個合適的 DTD ，然後在 XML 宣告中明白昭告世人：這份文件選用的是那個 DTD。這樣好讓 XML 解析器有個對照、驗證的依據。

前面說到，目前的 XHTML 1.0 草案中，已有三種不同的 DTD 供人使用，未來隨著 DTD 模組化、專門化的腳步，應該會有更多 DTD 出來。目前的這三種，只是摹仿 HTML 4.0 DTD 所定義出來的，包括：

Strict（嚴格式） 最早的 HTML 比較呆板，沒有什麼 `font` 、 `color` ...等花俏的東東。那些都是 Netscape 發明的。從標注語應該專注在文意的表達，而少碰外觀呈現的觀點來看[7]，W3C 認爲 HTML 文件中動不動這裡一小撮、那裡一大駝的 `<font>` 標籤，看起來眞是噁心極了！爲了整治這個亂像，W3C 特別發明了 CSS 來跟它相抗衡。如果您的文件只用 CSS 來呈現，堅決不用 `<font>` ，那這是適合您的 DTD。

Transitional（過渡期式） `<font>` 標籤大受歡迎，W3C 在拗不過廣大世人的渴求下，只得硬著頭皮把 `<font>` 補進 HTML 4.0 的標準中，但在說明文字中，仍不忘時時告誡世人，儘量多用 CSS。這是「過渡期」一詞的由來 — 因爲他們希望，隨著有愈來愈多的瀏覽器支援 CSS ，`<font>` 標籤會逐漸消聲匿跡（嗯，恐怕沒那麼樂觀）。如果您的網站，像大多數網站一樣，使用到 `<font>` 標籤的話，則必須選擇使用 Transitional 這個爲「過渡期」設置的 DTD。

Frameset（分割畫面式） 如果您酷愛使用 `<frame>` 的話，那這是您不二的選擇 ;-)。

[7] 在這本書一開始就談到過這個問題。這是正確的觀點，更帶動了 XML 的誕生。

選好 DTD 以後，就可以開始準備宣告了。在第 8.2 頁中，我們談到用

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

這樣的方式來鍊結一個外部的 DTD，其中的 doc 是該 XML 文件根元素的名稱，而 "http://foo.bar/baz.dtd" 則是 DTD 所在的網址。「SYSTEM "網址"」這部分合起來叫「外部標示」(External ID)。

至於上面介紹的三種 XHTML DTD ，正確的 <!DOCTYPE> 宣告方式分別是：

```
<!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">
```

在這裡，我們看到另外一種格式的外部標示，這次用的是 PUBLIC 這個表示「公用」的標示，最後面接的，仍是 DTD 所在的網址。

要特別提醒您的是， <!DOCTYPE> 宣告必須出現在根元素標籤之前，否則等解吸器都已經看到根元素標籤了，卻還渾然不知這個元素是屬於哪個類型、該去參考哪個 DTD，那就來不及了。

> ⚠ 注意
>
> 以上列舉的所有 <!DOCTYPE ...> 宣告，大小寫都有特定的意義，寫錯一點點都不行。

試了才知道

很明顯地，光表面上宣告，私底下卻掛羊頭賣狗肉、沒有完全按 DTD 的規矩來做，是沒有意義的。每份 XHTML 文件，都得和它選用的 DTD 核對無誤後，才有資格算是正

W3C 的核對服務：http://validator.w3.org/

牌的 XHTML 。最簡單的核對方法，是利用 W3C 的核對服務。您只要在這個網頁中，填入您網頁的網址，送出，就會看到結果了。這個時候，如果它顯示的是："No errors found! Congratualations..." 的訊息，而不是一堆紅字，那麼恭喜恭喜！這代表您製作出來的網頁，非常的乾淨清爽。另一個核對的方法，是直接在網頁中建一個超連結，連到 `http://validator.w3.org/check/referer`。然後在瀏覽器中點擊這個連結即可。

不過現在還不到測試的時候，因爲還有幾項規定要交代一下。

6.4.4 其他規定事項

格式正確和有法可證的問題都解決後，接下來的就好辦了。

元素、屬性一律小寫

J

前面談過， XML 像 Unix 系統和許多程式語言一樣，是有區分大小寫的[2.7]。因爲這個緣故，XHTML 不能再像 HTML 那樣，標籤大小寫隨你便，必須統一作個規定，而標準決定採用小寫。因此，要轉型到 XHTML，那些過去用大寫字母來寫的元素、屬性名，都得先替換成小寫才行。

使用正確的根元素加名稱空間

C

XHTML 標準中規定，文件的根元素標籤必須是 `<html>`，而且要用 `xmlns` 這個特別的屬性來註明 XHTML 專屬的名稱空間[5]，像這樣：

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

其中 `http://www.w3.org/TR/html1` 這個網址所連到的，正是 XHTML 1.0 的標準文件（xhtml 格式，不意外）。其他標注語的名稱空間，也往往採用各標準文件所在的網址來代表。

前面談到，名稱空間的機制，是 XHTML 易於擴充[6.3.2]的主要關鍵。在根元素的標籤中標明了 XHTML 的名稱空間後，我們便可依需要使用來自其他名稱空間的標籤，而不用擔心同名的標籤會打架了。

標明哪兒是頭，哪兒是身

在 HTML 中， <head> 和 <body> 可以省略，但在 XHTML 中，一律得按規矩加在適當的位址。還有，<title> 必須是 <head> 中第一個出現的元素。 D

用 CDATA 區來包裝 style sheets 和 scripts

HTML 碼中常有 CSS 樣規[1.2] 或 JavaScript 程式直接內嵌在裡面。這些樣規和小程式分別出現在 <style> ...</style> 和 <script> ...</script> 區塊中。在 XHTML 中，因爲 style 和 script 兩個元素的內容在 DTD 中被定義爲 PCDATA[p.108] 類別，它們的內容會被解析器[2.4]當作一般文字資料來解讀。如果不巧，碰到樣規或 scripts 中含有 < 或 & 這兩個標注符號的話，解析器很可能就掛在那裡了。 E

要避免解析器去解讀文件中的某些資料，最好的辦法是把這些資料放進 CDATA 區[2.8]中。因此，當內嵌的 CSS style sheets 或 JavaScript 碼中含有這些在 XML 中具特殊意義的標注時，XHTML 標準說要用 CDATA 區把它們包起來，像這樣：

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

反應快的讀者馬上要問：「有必要那麼麻煩嗎？XML 中不是也有註解嗎？而且寫法也和在 HTML 中一樣。那爲什麼不乾脆保持過去在 HTML 中的做法，把程式放到註解中不就結了？如下」

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

想得好！不過這麼做有個缺點 :-(。XML 1.0 標準說，解析器*沒有義務*將註解[2.3]內容原封不動地交給下游的程式。換句話說，註解內容是否到得了瀏覽器手中還得看解析器高興而定。但 CDATA 區就不一樣了 — 這部分解析器不可以隨便略掉。因爲這個緣故，XHTML 1.0 才規定要用 CDATA 區，而不告訴大家可以用註解來做。

6.5 XHTML — 到底給誰看

經過檢測、驗證的 XHTML 文件便是不折不扣的 XML 文件，因此未來所有的 XML 軟體應該都讀得通。

「那過去的 HTML 瀏覽器怎麼辦？」

問得好。換一個角度來問：「那到底寫好的 XHTML 文件的副檔名要取作 `.xml` 還是 `.html`（或者是 `.xhtml`）？」

其實不只是副檔名，讀者中的站長 (webmaster) 們都知道，主機端還要有適當的 MIME `Content-Type:` 設定相配合才行；相對於 `.xml` 的 MIME type 是 `text/xml`，而 `.html` 對應的則是 `text/html`。

XHTML 文件可以按需要而設成這兩種類型其中一種，即：

- `.xml` 對應到 `text/xml`
- `.html` 對應到 `text/html`

但是如果打算設成 `.html`，而且要讓 XHTML 文件能被舊的 HTML 瀏覽器順利解讀的話，XHTML 1.0 的標準中提出以下幾個注意事項[8]：

[8] 在這裡只擇要摘錄。

- 空元素結束的地方，`/>` 之前，要多加一兩個空白，否則 HTML 瀏覽器會看不懂，也就是應該用：`<br />` 、`<img src="mypics.jpg" />` ，和 `<hr />` 這樣的寫法。

- 註明文字編碼時， XML 中的 `encoding="..."` 和 HTML 中慣用的 `<meta http-equiv ... >` 這兩種機制，都要使用，也就是：

```
<?xml version="1.0" encoding="Big5" ?>
......
<meta http-equiv="Content-Type" content='text/html; charset="Big5"' />
```

  別忘了，`<meta>` 是空元素，因此標籤結尾時要寫成像上面那樣。

- 有些瀏覽器會將 `<?xml version="1.0" encoding ... ?>` 這個 XML 專用的起始宣告[2.1]秀在網頁的頂端。如果文件中含有其他的 PI[2.10]（即 `<? ...?>` ），也會被這些瀏覽器秀出來。醜醜的，但沒有辦法。

- 如果網頁中的 CSS樣規或 script 中含有 `<` 、`&` ，或是 `]]>` 字串時，必須先把它們改存爲外部樣規或 script ，再以鍊結的方式呼叫，因爲老瀏覽器看不懂 CDATA 區[6.4.4]的寫法。

- 在 HTML 中以最小化[p.69]方式表達的屬性（如 `<input type="radio" ... checked>`），在經過 XHTML 「反最小化」[28]處理（也就是變成了 `<input type="radio" ... checked="checked">`）之後，會讓一些 HTML 瀏覽器看不懂。而在 100% 符合 HTML 4.0 的瀏覽器（如 Mozilla）中，應當不會有這種問題產生。這些最小化的屬性共計有：`compact` 、`nowrap` 、`ismap` 、`declare` 、`noshade` 、`checked` 、`disabled` 、`readonly` 、`multiple` 、`selected` 、`noresize` 、`defer` 。

Mozilla：`ftp://ftp.mozilla.org/pub/mozilla`

6.6 從何下手

讀到這裡，您可能會感嘆：「唉，這麼多項，聽起來好麻煩喔！誰有這麼多閑工夫把過去的 HTML 檔案一個個動手修改？有沒有什麼工具可以自動幫我檢查、修改的？」嘿嘿，有的。W3C 早已想到這個問題了，而且還送我們一個免費工具，它還有個可愛的名字，叫 "HTML Tidy"。

HTML Tidy ，顧名思義，是一個幫我們把不乾淨的 HTML 原始碼清一清的實用工具。Tidy 一如我在前面爲大家提供的 ccnv[p.38] 程式，是個 Unix 調調的指令介面工具。這是一個 C 程式，支援多種平台，它的 homepage 和下載點在：

`http://www.w3.org/People/Raggett/tidy/`

6.6.1 HTML Tidy 使用方法

要把中文的 HTML 檔案用 Tidy 自動轉成 XHTML ，可以下這個命令：

`tidy -raw -asxml -f` *錯誤訊息檔名* *輸入檔名* `>` *輸出檔名*

其中 `-raw` 是用來告訴 Tidy ，不要去亂動檔案中的中文碼，而 `-asxml` 是叫它把檔案轉成 XHTML 格式。如果您只是想用它來幫您清理一下 HTML 碼，而不想轉成 XHTML 的話（爲什麼不？），就不需要使用這個選項。因爲 Tidy 預設的輸出部是標準輸出 (standard output)，所以輸出檔名前的那個 > 號可別忘了，要不然輸出會全數吐到螢幕上。

如果執行了 Tidy，結果發現輸出檔是空的，這個時候很可能是您的 HTML 碼中有嚴重的錯誤，Tidy 無法處理。請您參考它的錯誤訊息，看到底是在第幾行的地方有錯，更正後再重新跑一次 Tidy 。

順利得到輸出檔，並不代表一切完全符合標準。如果您的檔案中用到了標準不支援（但某瀏覽器支援）的標籤、屬性，錯誤訊息中會一一列舉出來。不過要注意的是，您可能會碰到很多標示爲 "warning" 的訊息，這些大部分只是 Tidy 的「強烈建議」（最常見的像： `<img>` 標籤應該要附上 `alt` 屬性），並不是嚴重的錯誤。

要確定文件 100% 符合標準（也就是 DTD ），可以按照前面建議的檢測方法[p.71]，用 W3C 的核對器來查一下。

以下列出中文使用者最可能用到的幾個參數選項，給大家作參考：

參數選項	說明
`-raw`	不要去動 ASCII 以外的字碼
`-utf8`	輸入字碼爲 UTF8[3.2] 時，可用此選項來替代 `-raw`
`-asxml`	產生形式正確[6.4.2]的 XHTML 檔
`-f` *檔名*	把錯誤訊息存到指定的檔案裡
`-m` (`-modify`)	直接把修改過的內容，存回輸入檔內，取代原始內容
`-c` (`-clean`)	用 CSS 的方式改寫 `<center>` 和 `<nobr>` 這兩個沒有被標準接受的標籤
`-h` (`-help`)	列出所有支援的選項

6.7 最後

本章對 XHTML 作了非常詳盡的解說，涵蓋了 XHTML 1.0 草案中 90% 以上的要點，甚至還提供了一些草案中沒有的額外資訊。

建議您回到本章剛開始的地方，利用那裡的原始碼範例，幫助您一一回想在這章裡學到的新概念，讓印象更深刻。

7 XSLT — XML 專屬的轉換語

這章的重點是 XSLT 。要談 XSLT ，得先從 XSL 談起。

7.1 另一種樣規 — XSL 簡介

XSL (eXtensible Stylesheet Language) 是專門爲 XML 設計的樣規[1.2]語，也是在 CSS[4.1] 之外，另一個替 XML 穿戴打扮的選擇。

一如 MathML[1.3.1] 和 SVG[1.3.3] ， XSL 本身便是一項 XML 的應用，直接架構在 XML 的語法之上。它一共分作兩部分：第一部分負責將 XML 原始碼轉換爲另一種格式，而第二部分（稱作 "FO"〔Formatting Object ；打樣物件〕）則提供了大量的打樣指令，可用來配合印刷或螢幕顯像，精確地設定外觀式樣（譬如字的大小、擺放的位置等），是一種所謂 "device-independent" 的格式。第一部分的轉換語法，可以用來爲第二部分服務，將 XML 文件變形爲打樣指令。

有趣的是， XSL 的轉換語法，並不限於將 XML 轉成 FO 指令。事實上，XSL 可以輸出任何格式正確[2.5]的 XML 文件。因爲這個特性，我們可以用它來做以下幾種形式的轉換：

XML → HTML 這是最常見的一種轉換。轉換出來的 HTML 檔，在格式上因爲已達到格式正確的要求，在本質上非常接近 XHTML[6]。因爲目前大多數瀏覽器仍不支援 XML ，所以 XML → HTML 的轉換，在這個過渡時期有很大的利用價值。此外，由於 IE5 對 XML 加 CSS 的支援還不夠完善，如果在 IE5 中想把美化過的 XML 文件打印出來，唯一的方法是用 XSL 先轉 HTML 。用 CSS 來設定外觀的 XML 文件

在 IE5 中只能在螢幕上看，無法印。

XML → XML 把一種格式的 XML 文件轉換成另一種 XML 格式，有非常大的實用價值。例如：兩個機關或企業之間以 XML 來傳遞資訊，如果彼此使用的 XML 格式有出入，便可藉著 XSL 來做調整。

XSL → XSL XSL 甚至可以將一個樣規轉換成另一種形式的樣規。因爲 XSL 本身便是一項 XML 的應用，所有的 XSL 樣規自然也都是 XML 文件。因此， XSL → XSL 在本質上不過是 XML → XML 的一個特例而已。

7.1.1 XSL 和 CSS 不同的地方

XSL 採用的是轉換的方式，將一種格式的 XML ，轉換爲另一種。就好比將 Big5 碼繁體中文，轉爲 UTF-8[3.2] 碼一樣。CSS 則來自完全不同的理念： 它不含任何轉換動作，只針對 XML 文件中各個成分的外觀屬性，一一加以設定。瀏覽器便按照 CSS 樣規裡的指示，將 XML 文件呈現爲設定的式樣。整個過程中，沒有任何新碼產生。

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

XML 配上 CSS 、 ECMAScript 和 DOM 可以營造出類似 DHTML 般的動態效果。XSL 轉換則是死的，沒有互動性。

另一個區別是： XSL 樣規都是 XML 文件，完全照 XML 的語法來；相對地， CSS 在語法上自成一格，和 XML 的寫法大相逕庭。

7.1.2 XSL 和 CSS 相同的地方

XSL 和 CSS 都屬於樣規[1.2]的一種。樣規是用來設定外觀的，它並不影響原來的 XML 原始碼。XSL 雖然用的是轉換的方式，但「轉換」並不代表原始碼會遭到篡改。通常 XSL 轉換後的輸出碼，是另存到一個新的檔案、或暫存在瀏覽器的記憶體中，原來的 XML 檔案內容則保持不變。

7.1.3 XSL 簡史

XSL 歷經非常坎坷而戲劇性的發展。不但草案經過一而再、再而三的大幅修改（尚未成爲正式標準的東東難免是這樣），發展過程中更招致不少批判和反對的聲浪。 XML 專業人士之間，甚至曾經發生擁護和反對兩大陣營相互叫陣、「筆」鋒相對的熱鬧場面，足見 XSL 的爭議性。

XSL 最具爭議性的，是 FO 打樣物件的部分。反對者主要的論點是，XML 文件經過 XSL 、尤其是 FO 這麼一轉的結果，輸出的新碼中，很可能語意[1.1.2]盡失，原本一個像 <姓名> 的標籤，取而代之的將是一些冷冰冰的打印指令， XML 最重要的精神和價值所在，也隨之蕩然無存。相較之下， CSS 並沒有這樣的問題。不意外，反對 XSL 的人士多半也都爲 CSS 的擁護者。

爭議性之外，事實上，一部分也因爲 FO 的複雜度，大部分軟體，包括很早便開始支援 XSL 的 IE5 和 LotusXSL，也都不支援 FO 。爲了推廣 FO ，負責草擬的 Adobe 公司，甚至曾經使用「重賞之下，必有勇夫」的策略，懸賞兩、三萬美元，鼓勵網路上各方高手，積極開發 FO → PDF 的轉換程式。不過最後這個競賽因故取消。現在支援 FO 的軟體依舊寥寥可數。

W3C 也感受到各方對 XSL 的壓力。在 1999 年 4 月份，W3C 正式將 XSL 標準的前半部，關於轉換語法的敘述，從 XSL 中分出來，並爲它起了一個新名子，叫 XSLT ， T 代表 "Transformation" ，也就是「轉換」、「變形」的意思。自此之後， XSL 的草案只剩下 FO 的部分。 FO 的發展，已經顯得遲滯，目前動向不明，和日新又新的 XSLT ，形成強烈對比。XSLT 在 4 月到 10 月之間，又作了一些變革。在一波多折的發展、歷經 6、7 個版本的草案後，XSLT 現在總算熬到 W3C 建議標準 (proposed recommendation) 的地步（還有可能再作修改）。至於它是否能順利成爲正式標準 (recommendation)，答案很快就會揭曉，且讓我們拭目以待。本書也會跟隨最新的動態，適時更新。

⚠ 注意

由於 XSL 草案像個變色龍一樣，一改再改，導致絕大多數的教學文件和最新標準之間，出現一段落差，而在 1998 年 12 月之前寫的文章，更是嚴重脫節。有個簡單的方法可以看出 XSL 的教學文件或書籍是否已經過時 — 如果在文章中看不到 XSLT 、 XPath 這些名詞的話，那麼這個文章可能就已經有點舊了；如果連 `<xsl:apply-templates/>` 這個指令都見不到，那更是嚴重落伍了。在此建議初學者對過時（但或許仍具參考價值）的 XSL 文件，最好先盡量避免，以免在新舊標準之間，搞得頭暈腦脹、事倍功半。

7.2 XSLT 入門

我們在接下來的章節中，將把焦點集中在 XSLT 。XSL 打從一開始，就只有這部分在發燒，比較令人矚目。絕大多數軟體支援的，也只是這部分。在下面，當用到 "XSL" 這個名詞時，指的也大都是 XSLT 。

7.2.1 XSLT 在網路上的應用模式

讓我們來看看在目前的條件下， XSLT 能透過哪幾種方式來替我們的網站效勞。依 XSLT 轉換動作發生的地點來分析，可分爲：

server 端 XML 文件在下載到瀏覽器之前，先以 XSLT 轉爲 HTML；依轉換動作發生的時間，又可分成：

動態即時產生 在瀏覽器向 server 送請求的當下，以 XSLT 處理器軟體將 XML 立即轉換爲 HTML，即刻送出。在網頁中有動態性資料要即時插入、或當 XML 文件是從資料庫即時取得的場合下，最適合這種運作方式。負責輸

出 XML 的資料庫軟體，則很可能存在於另一台機器上，以 XML 來向負責 XML → HTML 轉換的 server 傳遞資料。這是現在最熱門的三層式、 n 層式 (3-tier, n-tier) 的 server 構築方式。大型資料庫網站，或大企業的 intranet 、 extranet 網站，較適合這種模式。因爲 XSLT 的運行步驟相當繁瑣（後述），所以訪客量大的網站，爲了兼顧快速轉換的要求，必須使用高檔的機器，甚至可能需要（一批）專責 XSLT 轉換的 servers。

批次產生 如果 XML 文件事先都已寫妥，一一存放在檔案裡，而非即時取得，或隨時不斷更新，我們便可用批次轉換的做法，事先將 HTML 檔準備好。轉換的工作則可依實際需要，在固定的時段自動化執行。這麼做的好處是，server 資源可獲得最大的節約，更不需要使用高性能的 server 機，任何能跑 XML 解析器和 XSLT 程式的電腦都行，甚至可以在個人家中的 PC 上來做，轉換好了以後再把 HTML 檔上傳到 server 即可。

client 端 如果瀏覽器直接支援 XML 和 XSLT ， XML 原始碼和 XSLT 樣規便可以像圖檔一樣，讓瀏覽器直接下載下去，一切轉換工作由瀏覽器代勞。這是最理想的狀況，但目前唯一支援 XSLT 的瀏覽器是 IE5 ，而不使用 IE5 的仍大有人在；更何況，IE 5 對 XSLT 的支援本來就不完整，有些部分更已過時，因此讓瀏覽器直接轉換的做法就目前而言實用性仍相當有限。不過這個美景在未來還是有希望實現的 — 微軟已經承諾將在未來補上目前缺漏的 XSL 功能；而新一代的 Netscape 瀏覽器（預計 1999 年底推出），則也可能支援 XSLT ，因爲已經有人將 XSLT 程式捐獻給 Mozilla 計畫。

雙管齊下 融合以上 server 和 client 端兩種運作模式，在瀏覽器請求網頁的同時，根據瀏覽器的身分即時決定是否將 XML 先轉換爲 HTML ，或直接交由支援 XSLT 的瀏覽器下載。 server 甚至可以更進一步，按照各個 client 軟體的特性，產生不同性質的 HTML ，讓訪客享受最佳的閱覽效果，譬如支援 CSS 的瀏覽器，便可以配合

下載美美的 CSS 樣規，如果碰到不支援 CSS 的瀏覽器（或是支援得太差的，像 Netscape 4），就給它含有 <font> 標籤的網頁。如同第一種運作模式（即時轉換），這種做法需要較高的技術水準，甚至需要重金添購專業軟體，更少不了高檔的硬體作後盾。

7.2.2 XSLT 的轉換流程及工作原理

XSLT 轉換必須由特別的軟體來擔任，這樣的軟體，通稱爲 XSL 處理器 (XSL processor)。如同我們在第 24 頁中談到的，任何 XML 文件，不管要作什麼用途，必先經過解析器解析，整理出條理後，才能進一步利用。既然 XSL 樣規和它所要轉換的 XML 原始碼都屬於 XML 文件，自然也需要解析。因此， XSL 處理器在工作之前，得先借重 XML 解析器（內建或外部），替它把 XSL 樣規和 XML 文件中一個個物件和結構分析出來。請看圖 7.1。

在 XSLT 中，有兩個重要的觀念，一個是源樹 (source tree)，指的是轉換前的 XML 文件的結構；另一個是結果樹 (result tree)，代表轉換的成品，結構依然是 XML。

XSL 處理器起動時，會先叫 XML 解析器替它解析 XSL 樣規和要轉換的 XML 文件。在掌握了樣規中的各個 XSL 指令後，XSL 處理器便開始依照樣規的指示，在源樹上爬來爬去，一遇到合適的枝葉，就按樣規的設定，吐出一段新枝葉（一個 XML 片段），一直到整顆樹都走遍了爲止。有的時候，樣規會指示處理器，將枝葉先修整一下再吐出來。所有經由 XSL 處理器一路產生的新枝葉，統統加到一塊兒，就成了一株結果樹。結果樹可以輸出到檔案裡儲存起來，或由瀏覽器顯像在螢幕上。

這麼解釋，可能您對實際轉換的細節還是很糢糊。別擔心！在稍後的章節中，我將帶您去「爬樹」，實地檢視 XML 樹的枝節。目前只需要掌握一個大概的輪廓就好。

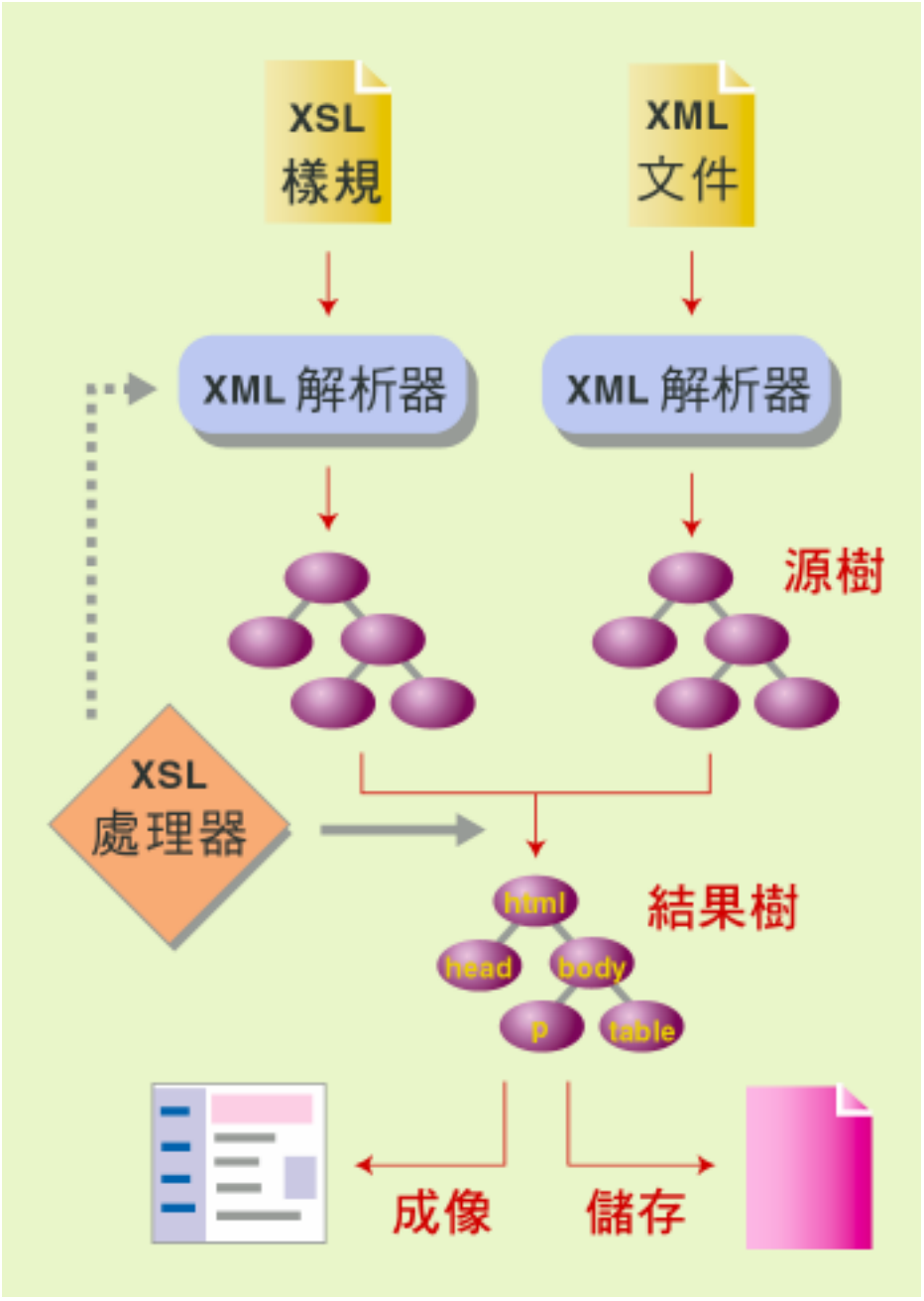

圖 7.1: XSLT 的工作流程

7.2.3 支援 XSL 的軟體

支援 XSL 的軟體，可粗分爲瀏覽器和轉換工具兩大類。目前比較熱門的瀏覽器中，只有微軟的 IE5 局部支援一個舊版的 XSL 草案（1998 年 12 月 16 日版）。在本章的結尾處，我們將會以 IE5 作實例演練。

在轉換工具方面，絕大多數的軟體都是以 Java 寫成的，包括由 IBM AlphaWorks 實驗室所出品的 LotusXSL，和 XSLT 標準的作者 James Clark[1]所寫的 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 標準。 xt 除了不斷隨著標準草案快速更新外，自 19990822 版開始支援以 Big5 、 GB2312 等亞洲文字碼直接輸出 HTML[2]。

在此要向讀者推薦，在處理 XSL 時，儘量使用 xt 作爲轉換工具。Clark 先生所寫的 XSLT 處理器和 XML 解析器[3]，在性能、速度、正確度，和對標準的支援度上，都是豎大拇指、一級棒的，和同類型的商業軟體相比，往往有過之而無不及。更可貴的是，他的軟體都是以程式碼公開 (Open Source) 的方式發行。

其他支援 XSL 的軟體還很多，無法在此一一介紹，有興趣的讀者請到 W3C 的網站去瞧瞧： http://www.w3.org/Style/XSL/。

7.3 細談 XSLT

XSLT 的語法和運行方式有些詭異，可能需要一些時間才能漸漸適應；而且必須以抽絲剝繭的方式，一小步一小步學習，如果一心求快，生吞活剝，很可能落得敗興而歸。先請您看一下 XSLT 原始碼到底長得什麼樣子，然後我再針對其中的觀念和細節，一一爲您剖析。

[1]和創建 SGI 、 Netscape 企業家 James Clark 同姓同名，但一個是英國人，一個是美國人。

[2]在舊版的 xt 裡，所有經過處理的中文字，只能以 UTF-8 形式輸出，因此如果要將輸出碼轉回 Big5/GB2312 ，需要再多跑一道轉碼手續。這個情況在輸出 XML 時依然存在，但是要 patch 不難。一等 XSLT 標準正式通過、xt 發展告一段落後，如果情況依舊，我會爲大家寫一個 patch ，讓 xt 可以直接以 Big5/GB 形式輸出。過去曾針對 1999 年 3 月 7 日版的 xt 寫過 patch ，但已過時，不能再用。

[3]包括鼎鼎大名的 Expat 解析器（以 C 寫成，速度極快），被 Mozilla 瀏覽器和 perl 、 Python 的 XML 模組選作內建解析器。

7.3.1 成品預覽

假設有一家電腦硬體製造商，在網站中提供產品搜尋的服務。他們的資料庫已經進步到能以 XML 形式輸出。以下是一筆摹擬的搜尋結果：

```
<?xml version="1.0" encoding="Big5" ?>
<?xsl-stylesheet type="text/xsl" href="search_result.xsl" ?>
<產品搜尋>
  <摘要>搜尋字串：“滑鼠 鍵盤”，共找到 2 筆</摘要>
  <產品>
    <貨號>12478943</貨號>
    <品名>手不痛健康滑鼠</品名>
    <定價>$234</定價>
    <說明頁 網址="http://foo.bar/mouse/12478943">上市發表會</說明頁>
  </產品>
  <產品>
    <貨號>83424723</貨號>
    <品名>打不響靜悄悄鍵盤</品名>
    <定價>$567</定價>
    <說明頁 網址="http://foo.bar/kbd/83424723">產品特性</說明頁>
  </產品>
</產品搜尋>
```

透過 XSLT ，我們可以將上面的 XML 碼，轉換成下面的 HTML 碼，以表格 (table) 來呈現搜索到的產品資訊：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=Big5">
<title>產品搜索結果</title>
</head>
<body>
<h1>產品搜索結果</h1>
<p>
<b>摘要：</b>搜尋字串：“滑鼠 鍵盤”，共找到 2 筆</p>
<table>
<tr>
<th>品名</th><th>定價</th><th>說明頁</th>
</tr>
<tr>
<td>手不痛健康滑鼠</td><td>$234</td>\continuearrow
  <td><a href="http://foo.bar/mouse/12478943">上市發表會</a></td>
</tr>
```

圖 7.2: XSLT 轉換出來的 HTML 檔，在瀏覽器中呈現出來的樣子

```
<tr>
<td>打不響靜悄悄鍵盤</td><td>$567</td>\continuearrow
  <td><a href="http://foo.bar/kbd/83424723">產品特性</a></td>
</tr>
</table>
</body>
</html>
```

以上的 HTML 碼爲 xt[p.86] 實際的輸出結果。用瀏覽器來看，大致會像圖 7.2 那樣。

整個轉換過程眞正的主角，是下面的 XSLT 碼。爲了方便閱讀起見，XSLT 指令的部分，一律以藍色標示。

```
<?xml version="1.0" encoding="Big5" ?>
<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="Big5"/> B
```

```
<xsl:template match="/">C D
  <html>
  <head>
  <title>產品搜尋結果</title>
  </head>
  <body>
    <h1>產品搜尋結果</h1>
    <p><b>摘要：</b><xsl:value-of select="*/摘要"/>E</p>
    <xsl:apply-templates select="產品搜尋"/>F
  </body>
  </html>
</xsl:template>

<xsl:template match="產品搜尋">D
  <table>
    <tr>
      <th>品名</th>
      <th>定價</th>
      <th>說明頁</th>
    </tr>
  <xsl:for-each select="產品">G
    <tr>
      <td><xsl:value-of select="品名"/></td>
      <td><xsl:value-of select="定價"/></td>
      <td><a href="{說明頁/@網址}">H<xsl:value-of select="說明頁"/></a></td>
      <!-- IE5 只接受 <td><a><xsl:attribute name="href">↩
              <xsl:value-of select="說明頁/@網址"/>↩
            </xsl:attribute><xsl:value-of select="說明頁"/></a></td> -->
    </tr>
  </xsl:for-each>
  </table>
</xsl:template>
</xsl:stylesheet>
```

7.3.2 XSL 樣規與名稱空間

如同其他許多 XML 的應用，XSLT 大力仰仗名稱空間[5]所提供的機制，把 XSLT 指令和其他 XML 標注隔開。樣規裡所有的 XSLT 指令，都必須標明是隸屬於 "`http://www.w3.org/1999/XSL/Transform`" 這個 XSLT 專用的名稱空間，才能正常工作。 A

在宣告 XSLT 的名稱空間時，最常見的做法是以 "xsl" 作爲前置字串[5.3.1]：

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

依規定，所有 XSLT 指令都必須置於 <xsl:stylesheet>...</xsl:stylesheet> 區塊裡。也就是說， xsl:stylesheet 這個元素界定了 XSLT 樣規的內容；因爲它是最外層的元素，名稱空間的宣告，自然也就擺在這個元素的標籤裡。 在 1999 年 10 月 8 日版的 XSLT 標準中， xsl:stylesheet 又新增加了一個不可省略的屬性： version ，用來標示樣規所遵循的 XSLT 語言版本。在先前的標準草案中，版本號碼是直接附在名稱空間網址中的。

在接下來的章節中，我們將按照慣例，所有的 XSLT 指令，一律使用 xsl 來作前置字串。但就如同我們在第 5 章所提到的，前置字串是任人自訂的，如果您不想用 xsl 這三個字母，大可在宣告中改用其他任何前置字串，包括中文，來代表 http://www.w3.org/1999/XSL/Transform 這個 XSLT 專屬的名稱空間。譬如以下的寫法，是百分之百合乎規定的，但是這樣做，有沒有實質上的意義，就由各人自由心證吧：

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

7.3.3 源樹分解

「產品搜尋」這份 XML 文件，在 XSLT 的轉換過程中，會先被 XML 解析器解析。所得出的物件結構，即爲前面所提到的源樹[7.2.2]，如圖 7.3。 圖中每個橢圓形，都代表一個 節點 (node)。就像眞實世界中的樹木一樣，枝葉是從節點處發出去的。位於末稍的物件叫 葉節點 (leaf node) ，因爲葉子上通常不會再長出枝幹。在 XML 物件樹中，所有代表文字內容的節點（那些土黃色的橢圓），一定都是葉節點。

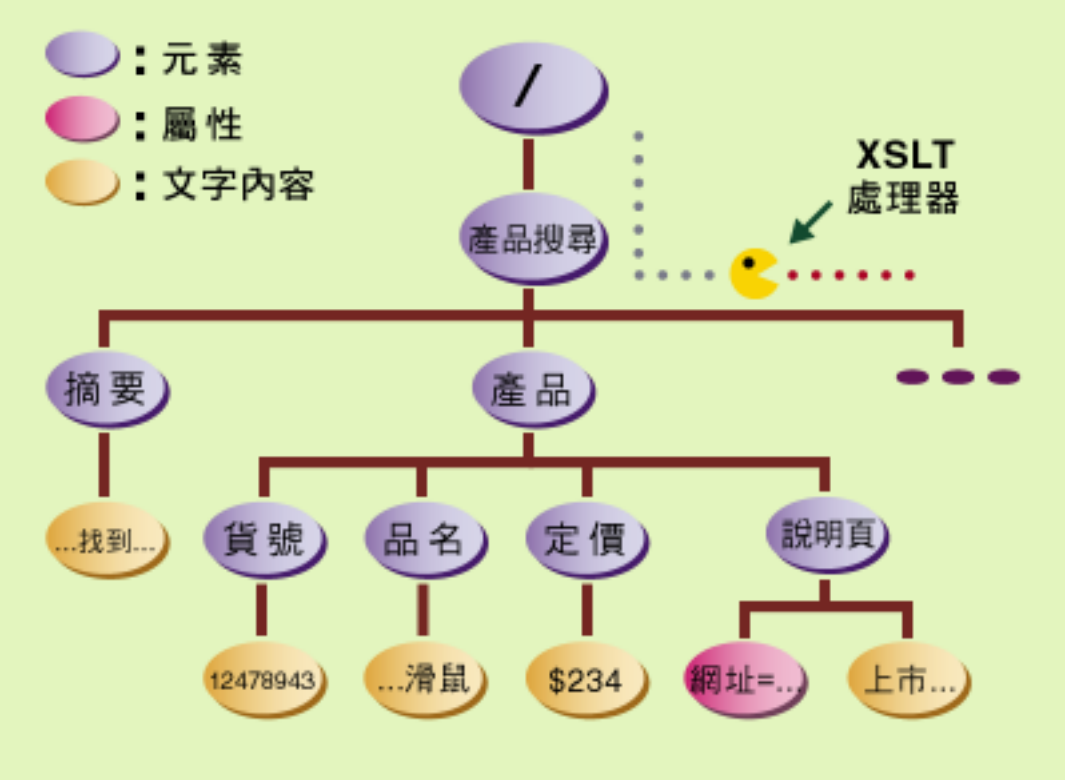

圖 7.3:「產品搜尋」這份 XML 文件，經由 XML 解析器所分析出來的物件樹，橢圓形的顏色代表 XML 的元件類型。

在接下來的各個章節中，我們將隨著 XSL 處理器的腳步，在這顆樹上爬來爬去。建議您在閱讀的時候，腦中隨時掌握這顆源樹的結構和成分，以達到最高的學習效果。您可能需要再回來多瞄這個圖幾眼。

W3C 的 DOM : http://www.w3.org/DOM

其實圖 7.3 中的樹狀結構，大致上就是按照 W3C 的 DOM（文件物件模型）所分析出來的樣子。

7.3.4 根元素上頭還有一級

C

雖然「產品搜尋」在 XML 中叫做根元素[2.2]，但人外有人，天外有天，在 XSLT 中，斜線符號 "/" 要當作是最高層，「產品搜尋」次之，如圖 7.3。熟悉 Unix 或 DOS 目錄結構的讀者應該不難想像。

7.3.5 XSLT 運行細節

模板和對應式

D

前面第 7.2.2 節只粗略地描繪了 XSLT 運作的流程。現在我們要談整個 XSLT 轉換的工作細節。前面說到，XSL 處理器按照樣規裡的設定，在源樹上找尋合適的節點[7.3.3]，並適時產生新枝葉。這個「樣規裡的設定」，在 XSLT 中的正式稱呼叫「模板式」(template rules)。XSL 處理器就是根據這些式子來尋找節點、產生新碼的。看個實例：

```
<?xml version="1.0" encoding="Big5" ?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output encoding="Big5"/>
<xsl:template match="/">
<html>
<p>一個很空洞的模板，不怎麼來勁</p>
</html>
</xsl:template>
</xsl:stylesheet>
```

這是一份完整的 XSL 樣規，裡面只有一個模板式。我們看到，模板式以 `xsl:template` 這個元素來定義，該元素所包含的內容，不意外，正是 XSLT 中所謂的「模板」（template；黑色文字部分）。 `xsl:template` 裡面有個 `match="..."` 的屬性[2.2]， XSL 樣規正是藉著這個屬性，來告訴 XSL 處理器，該去找什麼樣的節點。在這個例子中，模板式指定對應的節點是 `/` ，也就是在源樹最頂層的根元素。

XSL 處理器在起動時，會先將所有列在樣規中的模板式看過一遍，並牢記在心，然後便開始檢視源樹，而且一定先從 `/` 下手，每對應到一個節點，就把相關的模板式中的模板抄一份出來，加進要輸出的文件裡。

因爲*任何*一份 XML 文件，在分析成樹狀結構之後，「樹根」都一定是 `/` ，而上例中的 XSL 樣規，又剛好只含一個指名要對應到 `/` 的模板式，所以這份樣規不管拿來轉換什麼樣的 XML 文件，甚至像這樣一份簡陋至極的單行文件：

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

所產生的結果都必定是：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<p>一個很空洞的模板，不怎麼來勁</p>
</html>
```

事實上， `match` 的屬性值不限於使用確切的節點名（像上面的例子），其實它是一個對應式 (pattern) ，有一套完整的語法，可用來選擇樣規裡各個模板式所對應到的節點，後面會進一步介紹。

還有要請注意的一點：模板的內容，必須是格式正確[2.5]的 XML 片段。爲什麼？記得在第 7.2.2 節中曾經談到， XSL 樣規的設定內容，是先經過 XML 解析器解析，然後才交給 XSL 處理器的。因此，如果樣規裡有任何一段違背格式正確的原則，整個轉換過程會在 XML 解析器那關就宣告失敗， XSL 處理器根本連邊都摸不到。換句話說，不僅是模板式所包含的區塊，而是整份 XSL 樣規，都必須是格式正確的 XML 。如果模板裡頭含有 HTML 碼，則務必要按照第 6.4.2 節的建議，讓一切都「達到及格標準」。像常常被省略的 `</p>` （結束段落）標籤，就非添上不可。

現節點和語境

XSLT 處理器會爬樹，從這個節點爬到那個節點（至於該怎麼個爬法，我們將在第 7.3.5 討論）。每到一個新的節點，XSLT 處理器執行命令的*語境* (context) 就跟著改變。這個「新到的節點」，在 XSLT 中的術語叫 現節點 (current node)，即處理器當時所在的節點。這個概念很像在 Unix/DOS 指令列中執行 `cd` 命令來改變當時所在的目錄，新到的目錄就成了一切相對路徑依循的標準。

以第 7.3.1 節「產品搜尋」的例子而言，因爲 XSLT 處理器會從 / 根元素開始下手，而樣規中又有一個對應到 / 的模板式（`<xsl:template match="/">`），處理器於是以 / 作現節點，開始處理這個模板式中的設定。

現節點應用實例 — `xsl:value-of`

E

XSL 模板中的內容，除了能讓 XSL 處理器一五一十地照抄、輸出之外，還可以透過 XSLT 指令，對資料作排列、組合等處理（否則豈不太無趣了）。對模板的內容， XSL 處理器會先看，如果其中有隸屬於 XSLT 名稱空間[7.3.2]底下的元素，就把它當成是指令，如果它看得懂這個指令，便會做適當處置。舉例來說，在「產品搜尋」的 XSL 樣規裡有這一段：

```
<xsl:template match="/">
  <html>
  <head>
  <title>產品搜尋結果</title>
  </head>
  <body>
    <h1>產品搜尋結果</h1>
    <p><b>摘要：</b><xsl:value-of select="*/摘要"/></p>
  ......
  </body>
  </html>
</xsl:template>
```

其中的 `<xsl:value-of ... />`，就是一個常用的 XSLT 指令，它通常是用來讀取源樹中某元素所包含的文字內容，或是某個屬性的值。

⚠ 注意

<xsl:value-of ... /> 是一個空元素[2.5.1]。後面那道斜線 (/) 可別漏了。

<xsl:value-of ...> 標籤中一定要附上 select 這個屬性。如同 <xsl:template> 中的 match 屬性，在 xsl:value-of select="..." /> 裡， select 屬性後面接的，也是對應式 (pattern) ，而這次這個 "*/摘要"，看起來眞的像是個式子了。XSLT 對應式在對應的時候，以現節點作相對位置的軸點，它的語法，和 Unix shell 、 DOS 模式中的檔案、路徑表示法有些類似。在這個式子中，星號 * 會對應到位於現節點下一層的所有*元素節點*；換句話說，就是所有現節點所在元素的子元素。從圖 7.3 我們可以看出，「 / 」就只有「產品搜尋」一個子元素，因此， "*/摘要" 式子中的 * ，在現節點停留在「 / 」的語境下，只會對應到「產品搜尋」。

我們的式子中還有一條斜線 / ，這在對應式語法中是用來分隔節點層級的，就像 / 在網址和 Unix 系統中用來界定目錄一樣。合起來， "*/摘要" 會對應到一個名叫“摘要”的元素節點，它的母節點剛好是現節點的子元素節點。

在我們的例子中，因爲對應到的節點是一個元素， XSLT 處理器會將 <xsl:value-of select = "*/摘要" /> 整段指令代換爲源樹中「摘要」這個元素底下的文字內容，即「搜尋字串：“滑鼠 鍵盤”，共找到 2 筆」這段文字。因此， XSL 樣規中的這段：

```
<xsl:template match="/">
  ......
    <h1>產品搜尋結果</h1>
    <p><b>摘要：</b><xsl:value-of select="*/摘要"/></p>
  ......
```

會被 XSLT 處理器轉換成：

```
......
<h1>產品搜尋結果</h1>
<p>
```

```
<b>摘要：</b>搜尋字串：“滑鼠 鍵盤”，共找到 2 筆</p>
......
```

這樣的結果。

語境轉移

附圖 7.3 將 XSL 處理器比喻爲古老電玩中的小精靈 (pacman；我小時候流行的 game），會在源樹上各個節點間跑來跑去。但是到目前爲止，我們的語境一直停留在 /。要讓小精靈，嗯， XSL 處理器吃遍整顆樹，最常用的技巧是改變語境，讓現節點層層下移。有兩個 XSLT 指令 — `xsl:apply-templates` 和 `xsl:for-each` — 可以用來達到這個目的，轉移現節點的位置。

模板套入 — `xsl:apply-templates`

F

在我們的範例中，對應到 / 的模板裡只剩下一個指令沒交代：

```
<xsl:template match="/">
  <html>
  <head>
  <title>產品搜尋結果</title>
  </head>
  <body>
    <h1>產品搜尋結果</h1>
    <p><b>摘要：</b><xsl:value-of select="*/摘要"/></p>
    <xsl:apply-templates select="產品搜尋"/>
  </body>
  </html>
</xsl:template>
```

在討論 `<xsl:apply-templates select = "..." />` 之前，先來回顧一下 XSL 處理器已經掃過的部分，以及運行的流程：

1. XSL 處理器先讀入所有的模板式，包括上面這個。

2. XSL 處理器於是開始掃瞄源樹，先從由根元素 / 看起。

3. 檢視所有的模板式，看看有沒有模板式對應到 / 。

4. 發現一個可用的模板式，開始處理其中的內容。

5. 將 <html> 一直到 摘要： 這部分照本宣科地抄到結果樹上。執行 xsl:value-of ，把 和 </p> 之間的部分以 XML 原始碼中，「摘要」一元素的文字內容填入。

接下來比較有看頭了： <xsl:apply-templates select="產品搜尋" /> 中有一個 select 對應式，整個指令是在告訴 XSL 處理器：「把 select 對應到的各個節點一一當成是現節點，並依序處理這些節點的模板」。當 <xsl:apply-templates> 中的 select="..." 屬性省略時， XSL 處理器則會以所有現節點的子元素作爲處理對象。

⚠ 注意

<xsl:apply-templates ... /> 在這裡是以空元素[2.5.1]形式出現。後面那道斜線 (/) 可別漏了。還有，apply-templates 中的 templates 是複數，後面有 s 別忘了。

<xsl:apply-templates .../> 的出現，讓轉換流程起了變化。 XSL 處理器在這裡轉彎，先去執行其他的模板式，並且將產生的結果由 <xsl:apply-templates .../> 這點插入，一直到所有對應到的模板式都執行完畢，最後才回來執行原來模板中剩下的部分。而因爲其他模板中可能含有更多的 <xsl:apply-templates .../> 指令，所以整個流程是計算機學上所謂的 recursive （遞歸？）的執行方式。如果 XSL 樣規設計得不好，則可能出現無窮執行的窘境。

了解 xsl:apply-templates 的意義之後，我們可以繼續追蹤 XSL 處理器的腳步：

6. 遇到 <xsl:apply-templates .../> ，檢視所有的模板式，看看有沒有剛好對應到「產品搜尋」的模板式可以使用。

7. 發現一個可用的模板式，如下：

```
<xsl:template match="產品搜尋">
  <table>
    <tr>
      <th>品名</th>
      <th>定價</th>
      <th>說明頁</th>
    </tr>
  <xsl:for-each select="產品">
    <tr>
      <td><xsl:value-of select="品名"/></td>
      <td><xsl:value-of select="定價"/></td>
      <td><a href="{說明頁/@網址}"><xsl:value-of select="說明頁"/>
      </a></td>
    </tr>
  </xsl:for-each>
  </table>
</xsl:template>
```

8. 處理模板的內容，將 <table> 一直到第一個 </tr> 部分照抄進結果樹，從 <xsl:apply-templates .../> 處插入。

9. 遇到 <xsl:for-each ...> 指令，暫停，且聽下節分曉 ;-)。

xsl:for-each 迴圈

xsl:for-each 是另一個會改變現節點和語境的指令。和 xsl:apply-templates 一樣， xsl:for-each 後面也有一個 select 對應式，用來選擇新的現節點。在 XSLT 中， select 對應式所對應到的節點不見得只有一個，而可能有好幾個；如果不只一個， XSL 處理器會按照它們在 XML 原始碼中的順序來一一處理。譬如前面例子中的 <xsl:for-each select="產品">，用在第 7.3.1 節的「產品搜尋」 XML 原始碼上，一共會對應到兩個「產品」元素。 XSL 處理器於是按照它們在文件中出現的順序，

先把現節點設爲第一個「產品」（滑鼠），處理完了以後再把現節點轉移到第二個「產品」（鍵盤）上。

我們現在可以繼續上一節中未完的流程，把剩下的部分交代完：

9. 遇到 `<xsl:for-each ...>` 指令，現節點轉移到第一個「產品」元素處，除了將 `<xsl:for-each ...>` ...`</xsl:for-each>` 區塊中所有寫死的 HTML 元素依序照抄進結果樹中外，每當遇到 `<xsl:value-of ... />` 時，XSL 處理器會以源樹中所對應到的文字內容 — 分別是「手不痛健康滑鼠」、「$234」，和「上市發表會」 — 來代換這些 XSLT 指令。至於模板中 `"{說明頁/@網址}"` 的部分是下一節的主題，暫時不去管它。

10. 執行完一輪 `xsl:for-each`，處理器繞回 `xsl:for-each` 起始處，把現節點轉移到第二個「產品」（鍵盤），如法泡製，依序處理其中的內容。

11. 兩個輪迴之後，要轉換的 XML 原始碼中沒有更多的「產品」元素可處理，「產品搜尋」的模板式因告執行完畢。處理器於是回到原來的切入點，也就是根元素 / 的模板式中 `<xsl:apply-templates .../>` 的地方，繼續處理後面的部分。

12. / 的模板中只剩下 `</body>` 和 `</html>` 這兩個寫死的標籤，XSL 處理器於是將它們照抄到輸出結果中，大功告成！

`xsl:for-each` 是個常用的技巧，因爲 XML 文件中，往往包含反覆規律出現的元素（譬如像「產品搜尋」中的「產品」），然而出現的次數，往往不固定（每次搜尋的結果不盡相同）。而正因爲我們在範例的樣規中用了 `xsl:for-each`，不管搜尋結果出現幾筆「產品」，我們的樣規都可以不經修改，繼續正確地運行。這個重要的特性是任何成功的 XSL 樣規所必須具備的。

7.3.6 ｛屬性值模板｝

前面說到，當樣規中的元素隸屬於 XSLT 的名稱空間時（也就是我們的例子中，以 `xsl:` H

起頭的那些元素），XSL 處理器會把這些元素當成指令去執行。除此之外，樣規裡還有一個地方會引起 XSL 處理器的注意：XSLT 名稱空間以外元素的屬性值。譬如我們的範例中有這麼一段：

```
<td><a href="{說明頁/@網址}"><xsl:value-of select="說明頁"/></a></td>
```

在這裡，<td>、<a> 這些 HTML 元素並不屬於 XSLT 的名稱空間，而當這樣的元素裡面，有被大括弧 {} 括起來的屬性值時（上例藍色部分），XSL 處理器會把大括弧中的區域，當作是屬性值模板 (attribute value templates) 來處理。我們可以把屬性值模板想成是一個迷你模板，裡面可以包含第 7.3.5 節中談到的對應式。XSL 處理器會用類似處理 <xsl:value-of select="..." /> 的手法來解讀屬性值模板中的對應式，並且作代換（如果式子對應得到東西的話）。在這個範例中，屬性值模板是「說明頁/@網址」，裡面有一個前面不曾介紹過的對應符號 — @，這個小老鼠在 XSLT 中是用來對應屬性名的，像這樣：@要對應的屬性名。因此合起來，「說明頁/@網址」會對應到現節點底下，一個叫「說明頁」子元素的「網址」屬性。

前面說過，當對應式對應出來的結果是元素的話，XSL 處理器會以該元素底下的文字內容來作代換。不過這次的式子對應到的不是元素，而是屬性，這種情況下，XSL 處理器會以該屬性的屬性值來代換。因此，

```
<td><a href="{說明頁/@網址}">.....</a></td>
```

會分別被轉換成：

```
<td><a href="http://foo.bar/mouse/12478943">上市發表會</a></td>
```

和

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

7.3.7 輸出文字碼設定

B

1999 年 8 月 13 日版的 XSLT 草案新增了一個 xsl:output 指令，專門用來設定輸出

碼的各項特性，譬如字碼、縮排 (indentation) 等。`xsl:output` 中有一個 `encoding` 的屬性，可以用來控制輸出的字碼。視各人需要，可以把它設爲 Big5 、 GB2312 ，或 UTF-8 等。

7.4 XPath 路徑描述語

我們在前面只介紹了幾個簡單的對應符號，如 * 和 / 。事實上， XSLT 對應式[7.3.5] 的語法相當繁複而完整，光是這部分的語法，便多得足以分出去，自成一份單獨的標準。而這份標準的確已經出現，它的名子叫 XPath 路徑描述語。

XPath : http://www.w3.org/TR/xpath

XPath 的目的是提供 XSLT 和 XPointer （XML 文件內部連結語）一個共同、整合的對應語法，用來對應 XML 文件的各個部位、選擇文件中的構成元件（元素、屬性、文字內容 . . . ）。表 7.1 列舉了一些常用的對應符號及應用實例， 希望您不會看得過分眼花撩亂。這些只不過是一部分而已，還有不少沒列出來。 XPath 除了提供一套對應語法之外，還包括了一些函數，提供基本的數字運算和字串處理功能。

7.5 換您了 — 實作演練

總算快輪到我休息了 :-)。看歸看，但電腦語言這東西，如果沒有自己實際操作，絕對學不札實。在此建議您以 xt 作爲您的 XSL 處理器。如果您要用 IE5 的話，請務必參考第 7.5.2 節的各注意事項。

7.5.1 xt

第 7.2.3 節中介紹過、由 XSLT 設計人 Clark 先生所寫的 xt ，無疑是目前 XSLT 軟體中的佼佼者，不但對標準支援最好、最正確，更支援 Big5/GB 中文輸出。本章中的範例，便是以 xt 爲準。以下簡介它的安裝及操作方式。

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

對應式	說明
某元素	對應到現節點下所有名爲「某元素」的子元素
*	對應到現節點下所有子元素
*/某元素	對應到自現節點開始算起、所有名爲「某元素」的孫節點
@某屬性	對應到一個附屬於現節點、名爲「某屬性」的屬性
@*	對應到所有附屬於現節點的屬性
text()	對應到現節點的子元素中所有文字節點
.	對應到現節點
..	對應到現節點上一級
某元素[1]	對應到現節點下，第一個叫做「某元素」的子元素
某元素[position()=1]	作用同上
某元素/[@某屬性="某值"]	對應到現節點下所有名爲「某元素」的子元素，這個子元素必須含有一個叫「某屬性」的屬性，其屬性值必須爲「某值」
元素甲|元素乙	對應到現節點下，所有名爲「元素甲」和「元素乙」的子元素；| 代表「或」的關係
.//後世子孫	對應到現節點底下，所有名爲「後世子孫」的元素；// 符號代表可跨越數級
祖宗//徒孫	對應到所有名爲「徒孫」的元素，它們的上級必須有一個叫「祖宗」的元素，而且「祖宗」還得是現節點的一個子元素

表 7.1: XPath 路徑描述語中的對應簡式；取材自 W3C 標準

Java 虛擬機

xt 是個 Java 程式，因此先決條件是系統上必須有執行 Java 的環境，也就是所謂的 Java 虛擬機。關於各平台 Java 虛擬機的選擇及下載點，請參考我在 ccnv[p.38] 安裝說明中的建議。

選定、下載解析器

xt 只是個 XSL 處理程式，本身未內建任何 XML 解析器，因此安裝 xt 之前，系統上必須先裝有用 Java 寫的 XML 解析器才行。選擇 XML 解析器是一項非常重要的工作。市面上的 Java XML 解析器雖不下十幾種，但眞正能兼顧正確度、性能，又支援中文的寥寥可數。在此要向您推薦 Sun 的 XML parser ，它可自 `http://developer.java.sun.com/developer/products/xml/` 免費下載（但必須先註冊爲 Java 程式發展會會員）。 Sun 的 parser 不但支援 Big5/GB ，而且據 xml.com 的測試，正確度在所有同類產品中排名最高，可說 100% 符合 XML 1.0 標準的規定。另外一個支援中文、蟲不多的 Java XML parser 是 IBM 的 xml4j (XML for Java) ，它可以從 `http://alphaworks.ibm.com/tech/xml4j` 下載（很煩，也要註冊，而且在塡表時如果用的是自己眞正的 email 地址，又沒有仔細看清楚，還會招來一堆垃圾郵件）。

下載 xt

xt 可自 `ftp://ftp.jclark.com/pub/xml/xt.zip` 下載。

安裝 — 設定類別路徑

裝過 Java 程式的讀者都知道有一個 CLASSPATH 的環境變數要設定。在下載並解壓您最中意的 XML 解析器和 xt 後，只要將其中的 `.jar` 檔（`xt.jar` 、 `xml.jar` 、

xml4j.jar ...）的安裝路徑加進您系統的 CLASSPATH 環境變數即可。

用批次檔簡化指令

xt 是個指令列的程式，參數有些冗長，如果每次執行時都打一長串，太沒效率了。這個問題可以用一個 Unix shell script （或 DOS 批次檔）輕易解決。底下以 Unix 平台的 shell script 爲例：

```
#!/bin/sh

# 設定 Java 解譯器安裝的路徑
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"
# 您也可以在此設定 CLASSPATH ：
# export CLASSPATH="$CLASSPATH:/path/to/xml.jar:/path/to/xt.jar"

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

請自行調整其中的路徑設定。如果您使用 Windows 平台，則可以寫一個像這樣的批次檔 (.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 您也可以在此設定 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

一切都設定妥當後，只要在指令列下這個命令（假設上述的批次檔檔名爲 xt ，而且安裝在系統執行檔路徑中）：

```
xt  XML原始碼檔名  XSL樣規檔名  輸出檔檔名
```

一共是三個註明檔名的參數。假設您的設定一切正確，幾秒鐘後，轉換出來的結果應該就會被存在*輸出檔檔名* 這個您所指定的新檔案裡了。如果沒得到輸出檔，卻看到錯誤訊息，那就是 xt 、 XML 解析器，或 Java 虛擬機沒有安裝、設定妥當。

您可以按需要盡情地發揮創意，將上面提供的 shell script 加以修改（譬如加入迴圈，或配合 `find` 指令作子目錄橫掃、窮盡執行），甚至配合 `cron` 來定期執行。

7.5.2 遷就 IE5

如前面第 7.2.3 節所提，因爲 IE5 是根據一份過時的 XSL 草案所設計，而且對該份草案也只有部分支援，所以爲了讓本章的範例順利在 IE5 中運行， XSL 樣規中有幾個地方必須做調整。這些地方都已經以註解標示在範例裡。以下扼要說明修改的重點。

名稱空間的差異

IE5 用的是舊標準草案中規定的名稱空間 — `http://www.w3.org/TR/WD-xsl`，但是自 XSLT 從 XSL 中獨立出來後，指定的名稱空間目前已改爲 `http://www.w3.org/1999/XSL/Transform` ，因此要讓 IE5 能正確處理樣規，必須先修改 `xsl:stylesheet` 元素中所宣告的名稱空間，使用舊的名稱空間。

IE5 不支援 `xsl:output`

`xsl:output` 這個指令是在 1999 年 8 月的草案中才加進去的，因此不僅 IE5 ，絕大多數 XSL 軟體都還不支援這個指令。要讓 IE5 讀取的 XSL 樣規不可含有 `xsl:output` ，

您得先把範例中這行刪掉或註解掉。

IE5 不支援屬性值模板

屬性值模板[7.3.6]並不是一個新功能，但很遺憾地， IE5 內建的 XSL 處理器就偏偏漏了支援這項方便的功能。在 IE5 中，要達到類似範例中 {說明頁/@網址} 所產生的效果，必須改用較冗長的寫法，以 xsl:attribute 和 xsl:value-of 相互搭配。請參考成品預覽[7.3.1]中有關這部分的註解，並將

```
<td><a href="{說明頁/@網址}"><xsl:value-of select="說明頁"/></a></td>
```

這段改換爲註解中的寫法，即（實作時應寫成一行）：

```
<td><a><xsl:attribute name="href">
    <xsl:value-of select="說明頁/@網址"/>
    </xsl:attribute><xsl:value-of select="說明頁"/></a></td>
```

8 DTD — XML 語彙的定義

8.1 消除對 DTD 的恐懼

我們在第 2 章曾談到，在有些場合裡，光用格式正確[2.5] (well-formed) 的 XML 文件還不夠，必須配合 DTD 來清楚定義文件的格式。這樣在資料自動化互傳的過程中，我方的 XML 解析器[2.4]可以根據 DTD ，即時確認所收到的資料格式是否正確無誤，如果有問題可以馬上打回去，請對方程式重傳，或請對方填資料的人修正後重送。這樣就不會因無意中輸入格式錯誤的資料，造成資料敗壞 (corrupt) ，誤犯資料庫管理上的大忌。

許多剛接觸 XML/SGML ，或者曾經讀過 W3C HTML 標準的人，在初次看到 DTD 似無條理般的語法時，心涼了半截，從此以後不想再看到這個鬼東西（ OK ，或許我誇張了點 ;- ），接下來我們要從 DTD 中理出一些頭緒，降低莫明的恐懼。

前面使用過的 <推薦書籍> 一例（附表 8.1 ）， 只是一份格式正確[2.5] 的 XML 文件，

```
<?xml version="1.0" encoding="Big5" ?>
<推薦叢書>
    <書籍>
        <名稱>煞死你的網頁設計絕招</名稱>
        <作者>胭脂虎</作者>
        <售價 貨幣單位="新台幣">590</售價>
    </書籍>
    <書籍>
        <名稱>如何在 7-11 白吃白喝</名稱>
        <作者>無名氏</作者>
        <售價 貨幣單位="新台幣">120</售價>
    </書籍>
</推薦叢書>
```

表 8.1: 線上書店 XML 實例

就讓我們拿它作練習，編個 DTD 吧！

8.1.1 成品預覽

```
<?xml version="1.0" encoding="Big5" ?>
<!ELEMENT 推薦叢書 (書籍*) >A
  <!ELEMENT 書籍 (名稱, 作者+, 售價*) >B
    <!ELEMENT 名稱 (#PCDATA)>
    <!ELEMENT 作者 (#PCDATA)>
    <!ELEMENT 售價 (#PCDATA)>
    <!ATTLIST 售價
                  貨幣單位 ( 新台幣 | 人民幣 | 港幣 | 美元 ) '新台幣'>C
```

8.2 元素類別宣告

A

我們先從文件中的最小的子元素著手，採取「由下到上」的策略來分解這份文件的結構。在附表 8.1 中，一共有 <名稱> 、 <作者> ，和 <售價> 三個不含更小的子元素的元素。用 DTD 中的類別宣告，可以把這三個元素定義爲：

```
<!ELEMENT 名稱 (#PCDATA)>
<!ELEMENT 作者 (#PCDATA)>
<!ELEMENT 售價 (#PCDATA)>
```

很明顯地，ELEMENT 之後放的是元素名，接著是它的「內容模型」，說成白話，就是用抽象的表示法，定義 <元素> xxx </元素> 之間 xxx 的區域可以出現什麼樣的內容。按規定，這個「模型」要用小括號括起來。

#PCDATA 是 XML 中預先指定好的標記，代表 Parsable Character Data，即「可解析的文字資料」。意思是說裡面的文字內容可以讓解析器 (parser) 去解讀，因此如果內容中有 < 、> ，& 等這些保留給 XML 語言使用的符號時，要把它們「跳脫」、改寫成 < 、> ， 和 & ，就像在 HTML 中一樣，否則解析器會「掛」在那兒。

以上三個元素的內容模型都不怎麼來勁，裡頭只能有單調的文字內容。事實上，XML 元素中可以包含混合式的內容 (mixed content)，也就是既可以包含子元

素，又可包含文字內容（即 #PCDATA），這種混合內容的抽象模型就比較精彩、有看頭些了。

在接著定義其他元件之前，我想先介紹幾個「量子」符號。

8.2.1 量子學

DTD ，一如在正規表現式中，有所謂的量子（quantifier；或稱「量詞」）最常用的有「?」、「*」，和「+」這三個。量子是用來修飾它前面的那個單元的。怎麼說呢？譬如我們說「黃金萬兩」，這個「萬兩」，正是一個從後面來修飾的量詞。在 DTD 中，用量子表示數量一律要使用這種形式。問號 (?) 代表「可以有零個或一個」，換句話就是頂多只能有一個；星號 (*) 代表有幾個都可以，也就是可以有零到無限多個；加號 (+) 則表示至少要有一個，但沒有上限，像星號一樣，可多到無限個。

> ⚠ 注意　DTD 中的「?」、「*」、「,」，和「+」都是 ASCII 符號（半形），不要誤用了中文的全形符號。

8.2.2 出現次序

現在回到元素定義的課題上，最小的幾個元素解決了以後，我們可以往上走，開始定義上一層的元素，也就是 <書籍>：

```
<!ELEMENT 書籍 (名稱, 作者+, 售價*) >
```

我們在這裡用逗點和量子定義了 <書籍> 下屬的三個子元素出現的順序，和容許的數量。上回我們說到，當一本書的作者有兩人以上時，該怎麼表示的問題。有了 DTD 後，我們便可對此明確加以規範。

遵照以上的元素宣告，當我們遇到 <作者> 不只一個的情形時，可用：

```
......
<作者>比爾．蓋茲</作者>
<作者>知名不俱</作者>
......
```

的方式，來標注第二、三，四 ...作者。這正是上面的宣告中，在「作者」後邊放一個加號的精髓。還有請注意，它同時規定，一本書不能沒有作者，否則就會像對「售價」這個子元素的定義一樣，用星號而不用加號。

「那爲什麼售價一項，就可有可無，而且可以容許兩種不同的售價？」

因爲一本書可能還沒正式推出，價格尚未決定；此外，一本書可以用不同的貨幣單位來分別定價。當然，這些都只是我個人的看法，您可以依照您的需求和喜好，訂定適當合理的網路書店 DTD。

我們還剩下一個元素要定義，那就是最上層的 <推薦叢書>。因爲 <推薦叢書> 中可以包含很多筆 <書籍>，也可能一筆都沒有（統統不值得推薦），所以我們就把它定爲：

```
<!ELEMENT 推薦叢書 (書籍*) >
```

8.3 屬性類別宣告

C

在附表 8.1 中，我們看到在 <售價> 一項中，有一個「貨幣單位」的屬性，在 DTD 中，屬性是用 <!ATTLIST ... > 來做宣告。ATT 就是屬性、attribute 的簡寫。整個宣告看起來是這樣的：

```
<!ATTLIST 售價
                貨幣單位 ( 新台幣 | 人民幣 | 港幣 | 美元 ) '新台幣' >
```

最重要的是「貨幣單位 ...」這行（稱作「屬性定義」）共有三個要件。很明顯地，第一個是屬性名，第二個是屬性類別，最後是預設值或預設行爲的描述。如果屬性不只一個的話，這三個要件單元可以每三個三個這樣一直重複下去。在這裡，我故意用換行

和縮排把屬性定義突顯出來。其實要統統擠到前一行也成（不建議），每個單元之間只要有一個空白字元隔開就夠了。

屬性定義有很多種花樣，在這裡，我們選用的是適合我們的條列式表示法，把所有可能用到的屬性值一一條列出來，而且不能重複。直線記號"|"代表「或」。沒有列出來的值一律不許在 XML 文件中使用。屬性值列舉完後，我們要提供一個預設的值。當我們偷懶、把「貨幣單位」這個屬性省略掉，而只寫：

```
<售價>120</售價>
```

的時候，處理 XML 資料的程式會自動把它看成是：

```
<售價 貨幣單位="新台幣" >120</售價>
```

8.4 統統放到一起 — 文件類別宣告

上面的元素、屬性宣告一一設計好後，我們可以開始把它們整理起來，做成一個完整的 DTD，並且把它和我們先前的 XML 文件（附表 8.1）連結在一起。有兩種連結方式可以使用，一是外接，一是內嵌。如果用外接式，我們只要將剛才寫好的 DTD ，存到一個副檔名爲 `.dtd` 的純文字檔中（譬如叫 `book.dtd` ），再配合：

```
<?xml version="1.0" encoding="Big5" ?>
<!DOCTYPE 推薦叢書 SYSTEM "book.dtd" >
```

這樣的鍊結方式，就可以了。我說「鍊結」，是因爲如果這個 DTD 檔不在同一台機器的同一個目錄底下，則必須明確標明網址，而不能只寫檔名、路徑，否則會找不到。

或者，我們可以把 DTD 直接內嵌在 XML 文件中。這需要用到 `<!DOCTYPE ... >`，即「文件類別宣告」的標注，寫法上有點 CDATA 區的調調，像這樣：

```
<?xml version="1.0" encoding="Big5" ?>
<!DOCTYPE 推薦叢書 [
    <!ELEMENT 推薦叢書 (書籍*) >
    ......
]>
<推薦叢書>
    ......
```

8.5 結語

DTD 的語法相當繁雜，在此無法一一詳細解說。以上的介紹不過只是淺嚐，讓您在閱讀 DTD 文件時不至於丈二金剛，摸不著腦袋。

其實光是利用格式正確[2.5]的 XML 文件，不碰 DTD ，便已經可以大量運用 XML ；譬如以 XSLT 來做格式轉換，往往並不需要 DTD。

DTD 自成一格的語法，可說是 SGML 時代的「餘毒」，而不是 XML 中的新發明。DTD 的一大致命傷，在於無法理想支援名稱空間[5]的機制。在第 2.6 節中曾提到，目前正發展得如火如荼的 XML Schema （組織架構）標準，正是設計來取代 DTD，而且將像 RDF[1.3.5]、SVG[1.3.3] 、XSLT[7.2] 等應用那樣，直接使用 XML 語法，因此直接繼承了許多 XML 文件的優勢，例如：方便搜索的特性[4]。 XML Schema 除了完美支援名稱空間，讓 XML 標注可以透過各名稱空間 URI[5.2.2] 所連結的語彙[p.29]定義來作檢測、驗證[2.6]之外，還有許多比 DTD 更強大的功能，包括資料類型 (data types) 的定義，對 SQL 資料庫軟體提供非常重要的支援。難怪 IBM 和 Oracle 等資料庫業者無不卯足了勁，全力爲 Schema 標準摧生。從目前的發展腳步看來， Schema 可望在幾個月內成爲正式標準。

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