

# 3

## 連接資源

### 章節概要：

簡介

指定資源

XPointer：XML 結構  
樹上的爬樹猴

XLink 簡介

XML 應用語言：  
XHTML

廣義而言，連結（link）是指資源和資源間的關係。資源可以是任何東西，可能是用 XML 寫的文件，可能是二進位檔，像圖形或音效檔。資源甚至可以是一種服務系統（如新聞頻道或郵件編輯器），或是可以自動產生資料的電腦程式（例如搜尋引擎或資料庫介面）。

大部份情況下，資源就是 XML 文件。例如，要把圖片引入文字中，你可以在文件中建立連結，指向存有圖片的檔案。當 XML 處理器碰上這個連結時，就會利用連結所提供的資訊，去尋找這個圖形檔並予以顯示。連結的另一個例子是把你的文件和另一個 XML 文件連接起來。像這樣的連結可以讓 XML 處理器自動顯示第二個資源的內容，或是使用者需要時再顯示。

## 簡介

你可利用連結來建立媒體互連的網站，以提高文件價值，如圖 3.1 所示。圖中的連結稱為簡單連結 (simple link)，因為連結只牽涉到兩個資源，其中一個資源一定是 XML 文件，且為單向式連接。這種連結的所有資訊都放在單一 XML 元素中，而 XML 元素所扮演的就是連結一端的角色。前面提到的例子——匯入圖片，以及連結兩份 XML 文件——都屬簡單連結。

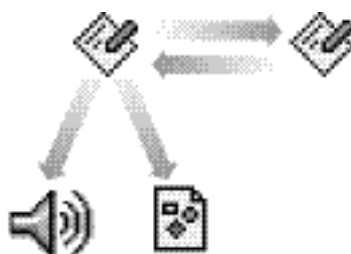


圖 3.1：由連結連接起來的資源群

更複雜的連結可以結合更多資源，而連結存放的資訊不見得有實際文件可供連結。例如，網站中也許有一頁主控頁，定義了複雜的瀏覽架構，而不是在每頁中宣告和其它網頁的連結關係。像這樣的抽象架構會讓錯綜複雜的網頁容易維護，因為所有的組態資訊都放在一個檔案？。

本書只討論簡單連結。這是因為有關複雜連結 (complex link) 的行為 (XLink 的一部份) 規格尚未定案，而且也沒有什麼 XML 處理器能夠處理。然而，在複雜連結定型之前，簡單連結還是能幫你不少忙。例如，你可以：

- 將文件分割成好幾個檔案，再用連結連接。這樣可讓好幾個人同時操作文件，且可將大型檔案分成好幾個小檔案，降低頻寬的耗用。
- 用連結建立重要地點之選單、內容表或索引，供使用者在不同文件間瀏覽。
- 引用網際網路上其它地方的文件，透過連結可提供一種方法，來取出和顯示這些文件。

- 使用連結，引入圖片、程式輸出或其它文件中的節錄，將資料或文字匯入文件內，並予以顯示。
- 提供一種媒體的表達層。你可以連結電影或音效，將其引入你的表達層中。
- 在使用者系統上觸動事件時，諸如開始寫電子郵件，打開新聞閱讀器，或打開媒體頻道，連結也許會含有該由那個應用軟體來處理某項資源的資訊，但也許不會。如果不含這類資訊，XML 處理器可根據它的喜好設定值，或是系統上對應資源類型（如 MIME 類型）的對應表，來啟動應用軟體。

圖 3.2 是簡單連結，有兩個資源，由一道箭號連接起來。本地資源（local resource）是連結的源頭，配有連結生效所需的資訊。遠端資源（remote resource）是連結的目標（target）。目標是被動的參與者，並不參與連結設定，不過，也許會有一個識別符號供連結之用。資源間的關係（relationship）稱為弦邊關係（arc），這用箭號表示，顯示一端正要連接到另一端。這也是 HTML 用來匯入影像和建立超連結時所使用的模式。

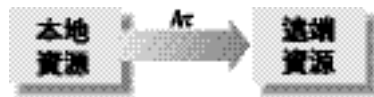


圖 3.2：簡單連結

簡單連結有三項特點：

- 連結會牽涉到兩個資源：含有連結資訊的本地資源和遠端資源。本地資源必須放在 XML 文件中。
- 連結會定義一個目標，指明遠端資源。
- 連結的行為由好幾項參數所定義，透過連結元素中的屬性來表達，稍後會再討論。參數如下所示：

- 連結啟動 (actuation) 說明連結如何被觸動。也許是自動啟動，諸如將圖形匯入文件中；也許需要與使用者互動，例如，讀者點一下超連結，告訴瀏覽器要往該連結走下去。
- 連結可用遠端資源做不同的事。可能是將遠端資源的內容，內嵌到本地文件的排版中，或是利用遠端資源實際替換本地文件。

● 連結可能還有其它相關的資訊，諸如文字標籤，或簡短的文字敘述。

讓我們來看一個例子。假設你想將圖片匯入文件中。會在某個元素中宣告該連結，通常是在你想讓圖片出現的地方。例如：

```
<image
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="simple"
  xlink:href="figs/monkey.gif"
  xlink:show="embed"
/>
```

第一個屬性建立了一個名叫 `xlink` 的名稱空間，做為該元素所有屬性的前置字。下一個屬性 `xlink:type` 宣告這個連結的類型是 `simple`，也就是告訴 XML 處理器，這個元素定義的是簡單連結。如果沒有這個屬性，其它的屬性就無法正確處理。再來是屬性 `xlink:href`，它持有圖形檔的 URL。最後是屬性 `xlink:show`，它指出 XML 處理器該如何處理這個連結。以此例言，檔案應被立刻載入，而其內容應在文件的這個地方進行排版設計。另外，請注意，該連結元素並沒有內容，因為載入這個資源並不需使用者輸入資料。

另一個連結範例如下：

```
<doclink
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="simple"
  xlink:href="anotherdoc.xml"
  xlink:show="replace"
  xlink:actuate="onRequest"
>click here</doclink>
for more info about stuff.
```

這的不同之處是，遠端資源的類型變成了 XML 文件。遠端資源不再像前例那樣自動載入，並內嵌到文字中，而是在使用者請求時，才替換當前網頁。屬性 `xlink:show` 和 `xlink:actuate` 分別控制了顯示模式和啟動方法。另一個差異處是，這個元素放有內容，可能就是用來啟動連結的方法，也許會仿照 HTML 瀏覽器處理超連結的方式：將文字標示出來，讓滑鼠可在文字範圍內點按。

## 指定資源

要建立連結來連接物件，則需指定物件。通常是透過一串字元來達成，名為統一資源標示碼 (uniform resource identifier, URI)。URI 有兩大類：第一種根據資源的位置定出唯一的名稱，第二種給予資源一個唯一的名稱，然後根據系統中的某一張表格，把名稱對應到實體位置。

URI 的開頭是協定前置字 (scheme)，協定前置字是一個短名，指出你要用什麼方式來辨識資料項，通常就是像 HTTP 或 FTP 之類的通訊協定。再來是一個冒號 (:)。然後是一串資料，是辨識資源的唯一名稱。無論協定前置字用的是什麼，都必須使資源有唯一的名稱。

以下更進一步來討論這兩種類型的 URI。

## 由位置指定資源

這種大家最熟悉的 URI 就是統一資源定位碼 (uniform resource locator, URL)：直接用位置辨識一項資源。URL 就像信封上的住址，住址有國家、省縣、街道，以及公寓的號碼。住址中每一項資訊都可縮小位置的範圍，直到指向某個地點。因此，郵政住址是優良的標示碼，不會重複。

同樣地，URL 使用電腦網路的命名法。URL 的資訊可以包括電腦的網域名稱 (domain name)、檔案系統路徑【註 1】、以及任何可以協助定出資源位置的系統特定資訊。URL 開頭是協定前置字，指出特定的定址方法或要用的通訊協定。目前定義了許多協定前置字，包括 HTTP、FTP 等等。例如，HTTP URL 是用來定出網頁文件的位置，其型式如下所示：

```
http://address/path
```

HTTP URL 的其它部份如下所示：

#### address

系統的位址。定出系統位址最常見的方法就是網域名稱，包含一系列用點號分隔的名稱，指出網路的層級。例如，`www.oreilly.com` 是 O'Reilly & Associates 網站伺服器的網域名稱。`com` 網域的伺服器是供商業網路之用的伺服器。講得更確切一點，伺服器位在 O'Reilly 網路的 `oreilly` 子網域下，一台名叫 `www` 的機器上。

#### path

資源的系統路徑。電腦系統中會有上千個的檔案。定出檔案在系統上的位置所使用的字串叫做路徑 (path)。路徑會用斜線【註 2】分隔各層目錄，連續列示出來。例如，路徑 `/documents/work/sched.html` 指的是位於 `documents` 主目錄下的 `work` 子目錄下的 `sched.html` 檔案。

以下是 URL 的範例：

```
http://www.w3c.org/Addressing/  
ftp://ftp.fossil-hunters.org/pub/goodsites.pdf  
file://www.laffs.com/clownwigs/catalog.txt
```

---

註 1 URL 的路徑部份不見得非得是真實的檔案系統路徑不可。有些協定前置字所使用的完全是另一種路徑，像是關鍵字的階層架構。但是，在我們的範例中談的是檔案系統路徑。檔案系統路徑是到目前為止，定出檔案在系統上的位置最常見的方法。

註 2 不同的系統，有它們內部路徑的表述法，例如，MS-DOS 使用倒斜線 (\)，而 Macintosh 使用冒號 (:)。URL 的路徑分隔字元，永遠都是正斜線 (/)。

URL 可予以擴展，引入其它資訊。片段辨識碼 (fragment identifier) 可用 # 字號附在 URL 的尾端，以指涉檔案中的某個位置。這只適用於少數幾個資源類型，像 HTML 和 XML 文件。片段辨識碼必須在標的檔的一個屬性內宣告。HTML 稱為連結錨 (anchor)，使用 <a> 元素，如下所示：

```
<a name="ziggy">
```

對 XML 而言，你得在某個元素中使用 ID 屬性：

```
<section id="ziggy">
```

要連結到這些元素中的任何一個，只要將其片段辨識碼附加到 URL 尾端即可：

```
http://cartoons.net/buffoon_archetypes.htm#ziggy
```

你也可以在 URL 尾端附上問號 (?)，再接由 & 相隔的引數，將引數傳遞給程式。例如，下列的 URL 會連結程式 clock.cgi，並將 zone (時域) 和 format (輸出格式) 這兩個參數傳給 clock.cgi：

```
http://www.tictoc.org/cgi-bin/clock.cgi?zone=gmt&format=hmmss
```

目前我們所說的 URL 都是絕對 URL (absolute URL)，也就是說，URL 是全部寫出來的 URL。這種方法很囉唆，有另一種捷徑。每個絕對 URL 都有一個基底元件，包括系統和路徑資訊，本身也是 URL。例如，http://www.oreilly.com/catalog/learnxml/index.html 的基底 URL 是 http://www.oreilly.com/catalog/learnxml/。如果連結中的標的資源和本地資源共用基底 URL，你就可使用相對 URL (relative URL)。相對 URL 就是少掉開頭部份的絕對 URL。

下表是一些 URL 的範例。第一欄的 URL 和第二欄的 URL 是對等的。假設標的資源的 URL 是 http://www.oreilly.com/catalog/learnxml/index.html。

相對 URL	絕對 URL
www.oreilly.com/catalog/learnxml/desc.html	http://www.oreilly.com/catalog/learnxml/desc.html
../	http://www.oreilly.com/catalog/
errata/	http://www.oreilly.com/catalog/learnxml/errata/
/	http://www.oreilly.com/
/catalog/learnxml/desc.html	http://www.oreilly.com/catalog/learnxml/desc.html

盡可能使用相對 URL 是好事一件。不僅少打一點字，而且當你決定把彼此相連的文件群移到別的地方時，由於只有基底 URL 會改變，因此所有的連結仍然有效。

有時你會想明確設定基底 URL；也許是因為 XML 處理器不夠聰明，無法自行找出來，或是你想連結許多不同位置的檔案。屬性 `xml:base` 可在其範圍中，設定所有相對 URL 的預設基底 URL。範圍就是該元素下轄的子結構樹。例如：

```
<?xml version="1.0"?>
<html>
  <head>
    <title>Book Information</title>
  </head>
  <body>
    <ul xml:base="http://www.oreilly.com/catalog/learnxml/">
      <li><a href="index.html">Main page</a></li>
      <li><a href="desc.html">Description</a></li>
      <li><a href="errata/">Errata</a></li>
    </ul>
    <p xml:base="http://www.coolbooks.com/reviews/">
      There's also a <a href="lxml.html">review of
      the book</a> available.
    </p>
  </body>
</html>
```

無論文件在什麼地方，連結總是指向相同的位置，因為基底資訊已直接寫進文件中。

## 由名稱指定資源

資源 - 位置的方法必須仰賴資源一直駐留在同一個地方。當標的資源從一個地方移到另一個地方時，連結就斷了。糟糕的是這種事經常發生；檔案和系統常常移來移去，名稱改來改去，或被整個移除。當這種事情發生的時候，就無法再用這些資源的連結，除非更新來源文件。要減少這種問題，有人提出了另一種方法：資源名稱。

資源命名方法背後的哲學觀是，唯一的名稱永遠不變，無論資料項移到那？都一樣。例如，美國公民一般都擁有一個九位數的社會保險號碼，這個號碼會陪他一輩子。其它資料則會改變，諸如他的駕駛執照編號、住那一街、或甚至是他的姓名，但社會保險號碼則不變。無論他住在 Portland, St. Louis, 或 Walla Walla, 這個社會保險號碼所指的永遠是他。

用這種和位置無關的方法來尋找資源，可以減少連結斷裂的問題，但為什麼這種方式反而不常用？想當然爾，如果在瀏覽器中輸入一兩個關鍵字，就可把你帶到正確的地方，那的確比較方便。然而，這樣的方法仍然相當生澀，而且和過去直接定址的方法相比，並沒有完善的定義。由於每部電腦系統都用相同的方式，來處理網路位址（用 IP 位址，內建到電腦作業系統的 TCP/IP 堆疊），所以網路位址仍行得通。資源命名方法需要把唯一的名稱，對應到經常變動的位址（也許是靠組態檔），而且軟體必須知道該如何查閱這些位址。

XML 常見的一種資源命名方法是用形式公眾辨識碼（formal public identifier, FPI）【註 1】。FPI 是一種文字字串，可描述許多有關資源的特點。集合了這些特點的資訊建立了一個識別符號。FPI 通常是出現在文件類型宣告（參見第二章）和實體宣告（參見第五章）中。

FPI 的語法如圖 3.3 所示。FPI 由一個代表辨識碼之註冊狀態的符號為開頭（1）：如果有註冊且是公認的，就是加號；如果都不是，就是減號；如果屬於 ISO，則是 ISO。符號後面是兩條當分隔符號用的斜線（2），然後是擁有者的辨識碼（3）。擁有者的辨識碼是一個短字串，可辨識出 FPI 表示的這個資料項之擁有者或維護者【註 2】。再經過兩條斜線，就來到公眾文字類別（4），它描述了 FPI 表示的資源種類（例如，DTD 指的是文字類型定義）。公眾文字類別後面接一個空格，然後是對資源簡短的文字敘述（5），諸如其名稱或用途。最後，是另外兩條斜線，後面再接一個兩個字母的語言碼（language code），指出資源的語言（6）。

---

註 1      ISO 正式標準：ISO-8879。

註 2      請注意，如果擁有者之辨識碼沒有註冊，說不定也不是唯一的名稱。

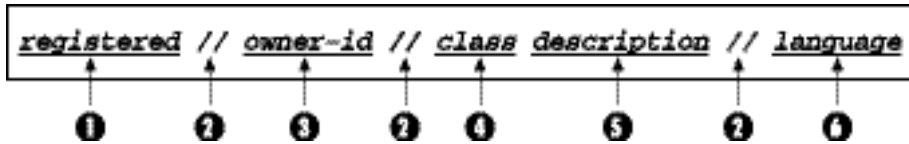


圖 3.3 : FPI 的語法

請看下列的範例，這個 FPI 屬於 ORA、未註冊、類別是 DTD、用英文所寫成：

```

1 2 3 4
-//ORA//DTD DocBook Lite XML 1.1//EN

```

- 1** 減號 ( - ) 表示這個組織贊助的 FPI，並未經公共單位（如 ISO）註冊認可。
- 2** 負責維護這份文件的單位是 ORA，也就是 O'Reilly & Associates 的簡稱。
- 3** DTD指出正在參考的文件類型是文件類型定義。後面接文字敘述 DocBook Lite XML 1.1，用簡潔的字串涵括了這個物件的名稱、版本編號以及其它資訊。
- 4** 兩個字母的語言碼 EN 表示，這份文件的主語言是英語。語言碼由 ISO-639 定義。

要完成連結，XML 處理器必須知道如何從 FPI 取得資源的實體位置，一般會牽涉到在目錄 ( catalog ) 中查閱某個名稱。目錄通常是你系統中的一個檔案，內含 FPI 欄位資料及資源的系統路徑。OASIS 團隊在技術文件 9401:1997 中，說明如何用目錄，從 FPI 上查閱位址，網址是 <http://www.oasis-open.org/html/a401.htm>。解析 FPI 的線上表單則在 <http://www.ucc.ie/cgi-bin/public> 中，可供參考。

XML 不能讓你單獨在實體宣告中使用 FPI。FPI 後面必須接一個系統辨識碼 ( 關鍵字 SYSTEM，後面接系統路徑或放在引號中的 URL )。XML 的設計者覺得仰賴 XML 處理器，從公眾辨識碼中取得實體位置，太過冒險，因此要再加一個提示位置。這樣會減弱公眾辨識碼的價值，但可能是件好事，至少在 FPI 廣為使用前是如此。

## 用 ID 和 IDREF 做內部連結

到目前為止，我們已經談過如何辨識整個資源，但那抓的是整體，而你想找的可能是文件中某段資料之後的東西。如何從上千個相同類型的元素中，找出你要的元素？其中一種簡單的方式是將元素標示出來。接著要談的 ID 和 IDREF 屬性可讓你標示元素，然後以此標示符號與元素連結。

### ID：元素唯一的辨識碼

在美國常用、不會重複的辨識碼是社會保險號碼 (SSN)。在這個國家裡，不會有兩個人拿到相同九位數的社會保險號碼 (不然，就是有人在做不該做的事)。你不會用社會保險號碼，來稱呼你的好友：「喂，456-02-9211，車借我好嗎？」但對於某些單位 (像是政府或保險公司)，把這個號碼當成帳戶號碼就很方便，因為這個號碼保證不會有人相同，也就不會出錯。同樣地，XML 也提供一種特殊的元素標示件，可以保證每份文件只能比對出唯一的元素。

這個標示件的型式是屬性。屬性有不同的類型，其中一種類型是 ID。當你在 DTD 中定義一個屬性為 ID 類型時 (參照第五章)，這個屬性對 XML 解析器就有特殊的意義。屬性值會被視為唯一的辨識碼，文件中的其它 ID 屬性不能與這個字元串重複，如下所示：

```
<sandwich lbl="blt">Bacon, lettuce, tomato on rye</sandwich>
<sandwich lbl="ham-n-chs">Ham and swiss cheese on roll</sandwich>
<sandwich lbl="turkey">Turkey, stuffing,
  cranberry sauce on bulky roll</sandwich>
```

這三個元素都有 lbl 屬性，DTD 將其定義為 ID 類型。屬性值由非空格的字元所組成，且都不相同。如果有兩個或兩個以上的 lbl 屬性具有相同的值，那就錯了。事實上，即使有兩個不同名稱的屬性，同屬 ID 類型，但若其屬性值相同，一樣是錯誤。

讓我們沉思一下。看起來是很嚴格地要求 ID 值一定要不同。我們為什麼需要解析器去檢核 ID 值是否相同？原因是，當你用 ID 做為連結的終點時，可節省很多後續的麻煩。在簡單的雙邊連結中，你會指定一個目標。如有兩個或兩個以上的目標具有相同的辨識碼，就會出現歧義的情況，而無法預測連結會連到那？。

HTML 中存在著很多歧義元素的問題。要在 HTML 文件中建立標示符號，你必須有一個連結錨：`<A>` 元素配一個 `NAME` 屬性，`NAME` 的值設為某些字元字串。例如：

```
<A NAME="beginning_of_the_story">
```

現在，如果你犯了錯誤，讓兩個 `<A>` 元素擁有相同的標示符號；對 HTML 而言，這並沒有錯，瀏覽器不會抱怨什麼，且連結也會正常運作。問題是你不知道會連到那？。也許連結會連到第一個實體，也許不會。HTML 規格並未指明結果。如果你是網頁設計師，很可能會扯著你的頭髮，搞不清楚為什麼連結沒有走到你想去的地方。

所以，藉由嚴格限制，XML 解決了我們的窘境和困惑。我們知道在驗正文件時，所有的 ID 值都是唯一的，因此連結不會有問題—假設目標找得到。這就是 `IDREF` 的角色，我們稍後會再討論。

那個元素要放 ID 屬性是由你決定，但應該有所限制。雖然替每個你想連結的元素都配上 ID 值很誘人，但最好只針對主要元素來標示就好。例如，在一本書中，你可能會對章、節、圖片和表格加上 ID 值，因為它們常是文字中的參考標的，但是沒有必要連內線元素都給 ID。

你也應該謹慎使用標示符號的語法。試著用容易記住及和內文相關的名稱，例如，“vegetables-rutabaga”或“intro-chapter”。階層式的命名結構可用來比對實際的文件結構。像“k3828384”或“thingy”之類的 ID 值很不好，因為幾乎無法聯想該 ID 值代表了什麼。若可以的話，不要用數字，萬一你必須調整文件結構，那就不妙了；像“chapter-13”這樣的 ID 值並不是什麼好名稱。

## IDREF：保證不會斷的連結

XML 提供的另一個特殊屬性類型稱為 `IDREF`。由 `IDREF` 的名稱可知，`IDREF` 存放的是同一份文件中某個 `ID` 的參考值。XML 無法讓你描述被參考元素與參考該元素之元素的關係。我們能說的是有某種關係存在，但那是由樣規或應用軟體所定義。這樣看來似乎價值有限，但事實上，這樣的關係給予我們相當簡單且有效的機制，來連接兩個或兩個以上的元素，而不用訴諸於複雜的 `XLink` 結構（本章後面會討論）。

還有另一個好處。我們已經知道 ID 屬性保證都是一份文件中的唯一名稱。IDREF 屬性也有其保證：任何由 IDREF 所參考的 ID 值，一定放在同一份文件。如果有一個 ID 連結斷裂，解析器會讓你知道，且讓你在文件釋出前修正錯誤。

ID 和 IDREF 能做什麼用？以下是幾個可行的事項：

- 書中做對照參考，諸如表格、圖形、章、和附錄。
- 有許多小節次的文件可用之做索引和目錄。
- 指稱某個範圍的一群元素，可出現在另一個元素之內，諸如索引中的術語，可以橫跨好幾頁。
- 連結註釋和方塊文章。
- 物件導向資料庫中的對照參考，物件導向資料庫的實體結構，也許和邏輯結構不同。

例如，文件中可能有好幾個註釋，共用相同的文字。此例中，<footnoteref> 是一個連結到 <footnote> 元素的元素，並暗指當文件進行處理時，它將繼承標的元素的文字：

```
<para>The wumpus<footnote id="donut-warning">
Do not try to feed this animal donuts!</footnote>
lives in caves and hunts unsuspecting computer nerds. It is related
to the jabberwock<footnoteref idref="donut-warning"/>,
which prefers to hunt its prey in the open.</para>
```

使用 IDREF 的微妙之處，在於知道要參考的是什麼。例如，若你想參考某一章，目的是將該章的標題引入要顯示的文字中，此時你應該指向該章的標題，還是該章的元素本身？通常最好是指涉能符合連結意義的元素，此例就是代表章的元素。以後也許你會改變主意，決定省略標題，改為顯示章次號碼或其它屬性，但是這種事應該讓樣規去擔心，要如何找到排版所需的資訊。在標記中，你應該把心思放在意義上。

## XPointer：XML 結構樹上的爬樹猴

最後一樣資源辨識法的謎團就是 XPointer，其正式名稱是 XML 指標語言 (XML Pointer Language)。XPointer 擴充 URL，使 URL 能夠指向任何 XML 文件的深層地帶。要瞭解 XPointer 如何運作，讓我們先討論比較簡單的片段辨識碼。片段辨識碼是 HTML 連結採用的機制，可以連接 HTML 檔案中的特定點。片段辨識碼是接到 URL 的尾端，和 URL 之間隔著一個井字號 (#)：

```
<a href="http://www.someplace.com/takeme/toyour/leader.html#earthling">
```

此例中，<a> 是連結元素。# 字號右邊的字 `earthling` 延伸了 URL，使其指向 `leader.html` 檔案中的某個位置。如果 `leader.html` 檔案中含有一個如下所示的標示件，這個連結就可以找到目標：

```
<a name="earthling">
```

XML 中相當於片段辨識碼的技術就是 XPointer。W3C 有關擴充 XML 的 URL 連結的建議案，就是 XPointer 這個名稱的來源。如同片段辨識碼一般，XPointer 也是接到 URL 的右邊，中間隔一個 # 字號：

```
url#XPointer
```

最簡單的情況下，XPointer 和片段辨識碼作用相同，皆是根據 ID 屬性連結標的資源中的元素。但是，XPointer 比較有彈性，因為目標可以是任何元素。這一點和 HTML 不同，HTML 的目標永遠都是 <A> 元素，而 XPointer 的目標可以是任何元素，只要元素擁有一個 ID 類型的屬性，能讓 XPointer 比對出來即可。

這樣的 XPointer 就已經很有用了，但 XPointer 的戲還沒唱完。XPointer 建議案定義了一整套語言，可以定位文件中的任何元素，無論元素是否具有 ID 屬性。這套語言是從 XPath 所衍生出來的 (參照附錄 B)。XPath 是一般性的規格，用於描述 XML 文件中的位置，所設計的語法滿足了 URL 語法的規則。XPointer 語言的指令可一步一步地走過文件。

讓我們寫一個 XML 文件範例，示範 XPointer 如何用於定位元素。範例 3.1 是一個簡單的人事表，顯示一家小公司的員工結構。圖 3.4 是這個文件的結構樹。



```
<name>Sarah Bellum</name>
<title>Vice President</title>
<staff>
  <employee>
    <name>Luke Bizzy</name>
    <title>Manager</title>
    <staff>
      <employee>
        <name>Eddie Puss</name>
        <title>Sales Clerk</title>
      </employee>
      <employee>
        <name>Mary Anette</name>
        <title>Sales Clerk</title>
      </employee>
    </staff>
  </employee>
  <employee>
    <name>Bubba Gumb</name>
    <title>Accounts Officer</title>
  </employee>
</staff>
</employee>
</department>

<department id="marketing">
  <employee>
    <name>Tim Burr</name>
    <title>Vice President</title>
    <staff>
      <employee>
        <name>Laurie Keet</name>
        <title>Promotions Officer</title>
      </employee>
      <employee>
        <name>Abel Boddy</name>
        <title>Advertising Officer</title>
      </employee>
    </staff>
  </employee>
</department>
```

```

        </staff>
    </employee>
</department>

</personnel>

```

XPointer `sales` 其實是 `id(sales)` 的簡稱。`id()` 是一種特殊的寫法，可跳到文件中含有 ID 屬性的元素，而且這個 ID 屬性值會與小括號中的字串相同。`id()` 也稱為絕對位置項目 (absolute location term)，因為 `id()` 能不靠其它項目的幫助，就找到唯一的元素。只有一個元素能擁有一個特定的 ID；如果真的有，`id()` 就會找到。

每個 XPointer 的開頭都是絕對項目 (absolute term)，然後可能再用相對位置項目 (relative location term) 擴充之，兩者再用點號 (.) 聯結。絕對項目會從文件中的某個點開始搜尋，然後相對項目 (relative term) 再從那兒開始搜尋，一步接著一步，直到所要的目標找到為止。每個項目都具有下列型式：

```
name(args)
```

*name* 就是項目的類型，而 *args* 是一個由逗號分隔的選項，替每個項目填入相關細節。

例如，下列的 XPointer 會從 ID 屬性值為 `marketing` 的元素開始搜尋，再移到第一個 `<employee>` 子元素，然後停在 `<employee>` 下的第一個 `<staff>` 元素：

```
id(marketing).child(1,employee).child(1,staff)
```

目標是 `<staff>` 元素，其父元素是 `<employee>`，而 `<employee>` 的父元素是 `<department>`，且 `id="marketing"`。

下一節中會更詳細地討論絕對位置項目和相對位置項目。

## 絕對位置項目

XPointer 必須以絕對位置項目為開頭，所有緊接在後的相對項目都是由絕對位置項目所提供，用來擴充的位置資訊。XPointer 所提供的絕對位置項目有四種類型，分別是 `id()`、`root()`、`origin()` 和 `html()`。

你已經看過 `id()` 的用法。`id()` 會找出文件中含有特定 ID 值的元素。如果文件經常更動，通常 ID 參考值就是最好的絕對項目之類型。即使內容重組，`XPointer` 仍可找到該元素。

絕對項目 `root()` 所指涉的是，基底 URL 所指定的整份文件。`root()` 會指向一個抽象節點（並非元素），而其子元素就是根元素。你大概不會單獨使用 `root()`，因為連結根節點通常沒什麼用處。相反地，你會在 `root()` 後面接上一連串的相對項目。例如，要到達行銷部門（marketing department），你可以用這個 `XPointer`：

```
root().child(1, personnel).child(2)
```

`id()` 需要一個引數，來設定所要搜尋的 ID，但 `root()` 不帶任何引數。`root()` 總是指向文件的頂端，因此無需引數。

`origin()` 是個絕對項目，可找出連結啟動後要連結的元素。由於 `origin()` 會自我參考（self-referential），也就是指涉文件本身，因此不能和 URL 合用。`origin()` 的一種用途是，連結原始元素和同文件中的另一個元素，以建立一個值域（range）。值域是 `XPointer` 的特殊種類，包含由兩個點號所連接的兩個位置項目，用來定出數個元素，以供一般用途之用。例如：

```
<p>Let's select <range href="root()..origin()">  
everything up to this point</range>.
```

就像 `root()` 一樣，`origin()` 不帶任何引數。

`html()` 是過渡用的絕對項目，用於 HTML 文件，可找出第一個 `name` 屬性值同於小括號內字串的 `<A>` 元素。`html()` 在比對出第一個元素後就會停止（不像 HTML 的片段辨識碼那樣，如果做複式比對，那行為將無法預測）。

## 相對位置項目

絕對項目可以帶你到文件中少數的幾個位置，不是在頂端，就是用 ID 標示。要走到其它地方，你必須採用相對位置項目。就像你告訴朋友要如何到你家一樣，相對項目也是一步一步地走過文件，直到目的地。

## 節點

回想第二章所說的，XML 文件可用家族樹來表示。相對位置項目則利用這個結構樹的模型，四處攀爬，從這梢樹頭跳到那梢樹頭，像隻訓練有素的松鼠。表 3.1 列出一些相對位置項目，你會發現和我們舉的類比頗有神似之處。請注意，現在所用的是節點 (node)，而不再是元素了。節點是 XML 的物件總稱 - 元素、處理指示或一段文字都包含在內。當前節點 (current node) 是上一個位置名稱定出來的樹點。

表 3.1 : XPointer 相對位置項目

相對項目	定位
<code>child()</code>	當前節點之下一個後繼子元素的節點。
<code>descendant()</code>	當前節點之下一個子孫元素的節點，次序採深度優先制。
<code>ancestor()</code>	當前節點之上一個祖先元素的節點，從 <code>root()</code> 開始。
<code>following()</code>	當前節點之後的一個節點。
<code>preceding()</code>	當前節點之前的一個節點。
<code>fsibling()</code>	當前節點之後同根節點中的一個節點。
<code>psibling()</code>	當前節點下同根節點中的一個節點。

這些名稱均帶有一到四個引數。引數按次序如下所示，

### 節點號碼

若位置項目所比對出的節點超過一個，就會建立一份合格的節點清單。如果你只要一個，則必須指定號碼。例如，假設該元素有三個子元素，但你只想要第二個子元素，你可以用 `child(2)` 來指定。正整數值會從合格的節點清單中往前數，而負值會往後數。如果你想找最後一個 (或倒數第二個等等)，往後數就有用處。另外，你可以用關鍵字 `all` 選取所有可用的節點。

### 節點類型

節點類型指出要比對那一種節點。如果其值是名稱，或省略引數，則要比對的類型就會被假定為元素。如果要比對其它類型，你必須使用關鍵字：

```
#text
```

比對連續的字元資料串。

```
#pi
```

比對處理指示。

```
#comment
```

比對註解。

```
#element 或 *
```

比對任何元素，無論其名稱為何。

```
#all
```

比對任一節點。

例如，`descendant(1,#all)` 會比對任一節點，無論節點是元素、正整數、註解或文字串。`descendant(1,*)` 會比對所有元素，而 `descendant(1,buttercup)` 則會比對所有 `<buttercup>` 類型的元素。

### 屬性名稱

這個引數會縮小元素的搜尋結果，因為元素必須擁有一個特別的屬性才行。屬性名稱引數只有當節點類型是元素時，才有作用。你可以指定一個名稱，要求某個屬性要存在才行，或者用星號（\*）接受任何屬性（可惜，沒有方法可指定一個以上的屬性）。如果省略屬性名稱，屬性就不會用在比對場合。這個引數必須和下一個要談的屬性值引數合用才行。

例如，`ancestor(1,grape)` 會比對任何的 `<grape>` 元素，無論其是否有屬性。`ancestor(1,grape,vine,*)` 會比對只有 `vine` 屬性的 `<grape>` 元素，而 `ancestor(1,grape,*,*)` 會比對任何至少有一個屬性的 `<grape>` 元素。

### 屬性值

這個引數設定屬性名稱引數中所指定的屬性值。你可設定特定值，使用星號接受任何值，或使用 `#IMPLIED`，表示未指定屬性值，且屬性是可有可無的。如果屬性名稱引數有用，那就不能省略這個引數。

例如，`preceding(1,fudge,tasty,yes)` 會比對出所有像這樣的元素：`<fudge tasty="yes">`。`preceding(1,fudge,tasty,*)` 會比對出有 `tasty` 屬性的 `<fudge>` 元素，且 `tasty` 屬性值可為任意值；而 `preceding(1,fudge,tasty,#IMPLIED)` 會比對出 `<fudge>` 元素，即使沒有 `tasty` 屬性也無所謂。

交待完引數後，讓我們看一下相對位置項目的細節：

`child()`

`child()` 會定出當前節點之下個子元素中的一個節點。與 `descendant()` 不同的是，`child()` 不會走超過一層深，而是在一個有限的區域內搜尋。如果找不到節點，處理器的回傳速度會比 `descendant()` 或 `forward()` 還快。

圖 3.5 顯示了 `child()` 向前旅行之路徑（使用正的節點編號引數）以及向後旅行之路徑（使用負的節點編號引數）。黑色節點是來源位置。

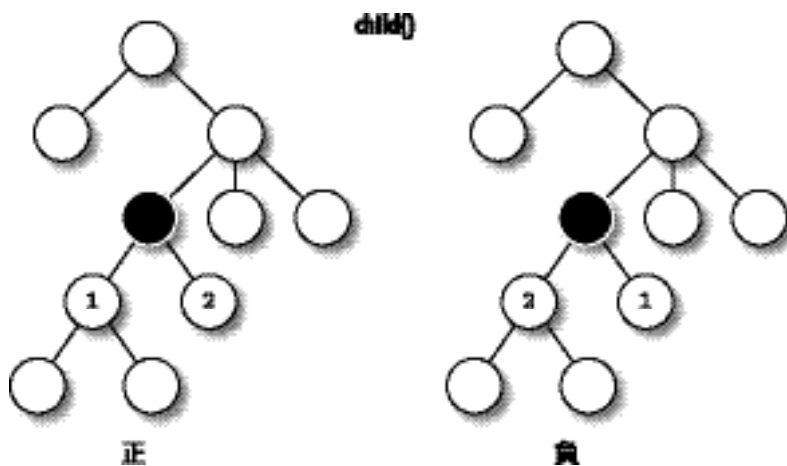


圖 3.5 : `child()` 的路徑

例如，要找出銷售部門的經理名字，你可使用以下的 XPointer：

```
id(sales).child(1,employee).child(1,name)
```

XPointer 可接受以下的語法縮寫：如果前後項目的類型相同，你可省略第二個項目。所以，此例的 XPointer 可縮寫成：

```
id(sales).child(1,employee).(1,name)
```

`descendant()`

`descendant()` 比 `child()` 走得更遠，會搜尋到任何深度的子孫元素。但是，`descendant()` 仍然只搜尋當前節點下的子結構樹。旅行的順序是深度優先 (depth-first)，也就是說會沿著一條樹枝走下去，直到碰到葉子時，再反轉往別的方向走下去，形成曲折的路徑。`descendant()` 的搜尋保證會碰到當前節點下的每一個節點。節點的搜尋次序如圖 3.6 所示，有正負兩個方向。

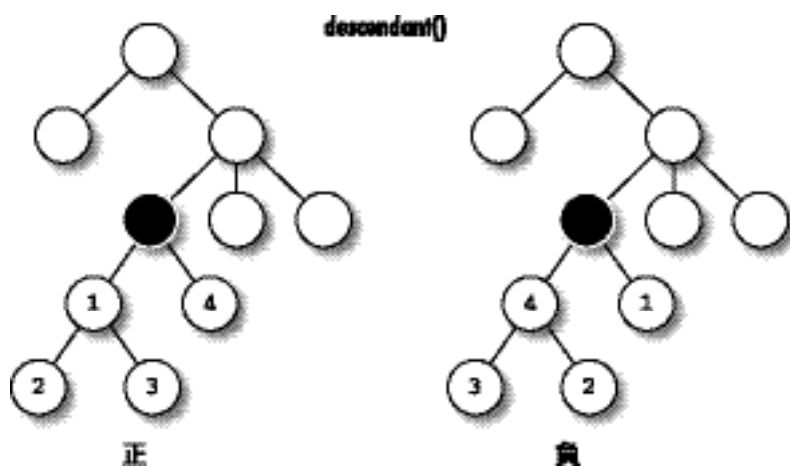


圖 3.6：descendant() 的路徑

`descendant()` 的數值引數比 `child()` 的複雜許多。用正數時，項目會從當前元素的起始標籤開始，然後向前讀取檔案，計算每個子孫元素的起始標籤，直到遇上當前節點的結尾標籤。用負值時，會從元素的結尾標籤開始計算，然後向後讀取檔案，計算每個結尾標籤。例如，`id(start).descendant(4)` 會找到 `<item2>`，因為從當前元素的起始標籤開始，在標的元素之前共有四個標籤：

```
<cont id="start">
  <sub1>
    <item1>text</item1>
    <sub2>
      <item2>more text</item2>
    </sub2>
  </sub1>
</cont>
```

前面的 `child()` 範例需有兩個相對項目，你可以簡化 `child()` 的範例，改用一個 `descendant()` 項目來代替：

```
id(sales).descendant(1,name)
```

這個範例會搜尋 `<department id="sales">` 節點下的子結構樹，找到第一個出現的 `<name>` 元素。

### `following()`

`following()` 搜尋區域的限制最鬆，搜尋的對象包括當前節點之後的所有節點。`following()` 會從當前節點開始，一個節點一個節點走過，直到發現符合的節點或碰到文件的結尾。圖 3.7 說明了兩方向的比對次序。

例如，你可以用 `following(1,employee)` 找到在 Eddie P. `<employee>` 元素下面的 Mary A. `<employee>` 元素。從同一個起始點，你可以用 `following(3,employee)` 找到 Tim B. `<employee>` 元素。

### `preceding()`

`preceding()` 類似 `following()`，但重心在文件的另一邊，從來源位置到文件開頭。搜尋方向也相反，所有正數是移到檔案的頂端，而負數是移向來源位置。圖 3.8 說明了兩方向的節點搜尋順序。

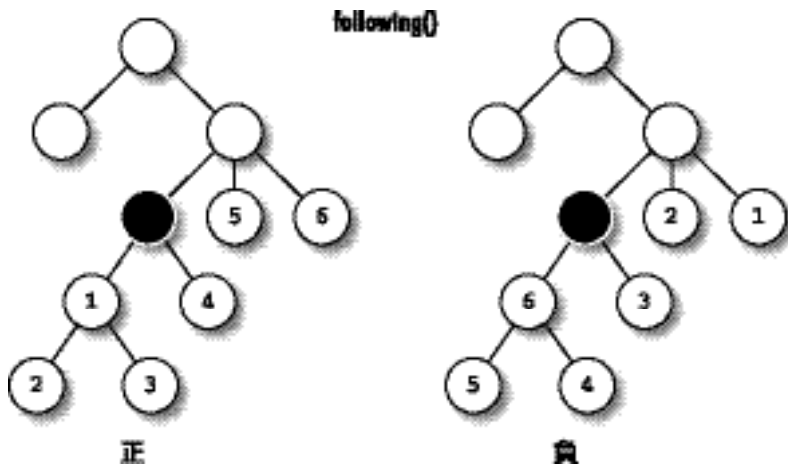


圖 3.7 : following() 的路徑

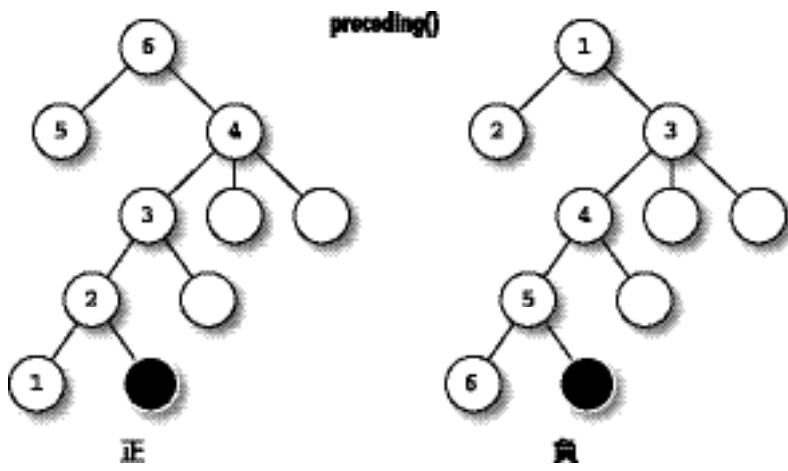


圖 3.8 : preceding() 的路徑

從任一員工的位置開始，你可以用 `preceding(1,employee)`，找出該名員工之前的員工。如果從 Laurie K. 開始，會找到 Tim B.；如果從 Abel B. 開始，會找到 Laurie K.。

### `fsibling()`

搜尋來源位置之後的同根節點。來源位置的節點可說是較年輕的同根節點。`fsibling()` 只會去找與當前節點有相同父節點的元素。就像 `child()` 那樣，`fsibling()` 的搜尋區域相當小且安全，然而代價是 `fsibling()` 需知道文件結構的某些資訊。圖 3.9 說明 `fsibling()` 的節點搜尋路徑。

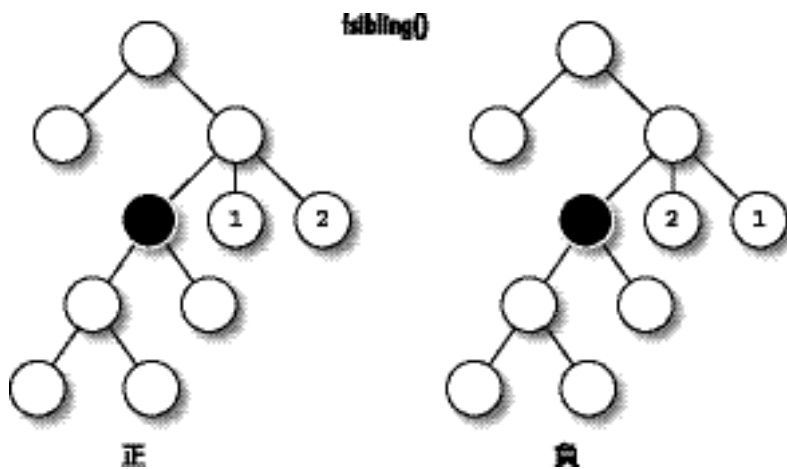


圖 3.9 : `fsibling()` 的路徑

例如，`fsibling(1)` 可以找到 Luke B. 的同事 Bubba G.，但 `fsibling(2)` 卻找不到東西。

### `psibling()`

`psibling()` 和 `fsibling()` 很像，但其搜尋的是來源位置前面的同根節點（較老的同根節點）。搜尋方向也相反。搜尋路徑如圖 3.10 所示。

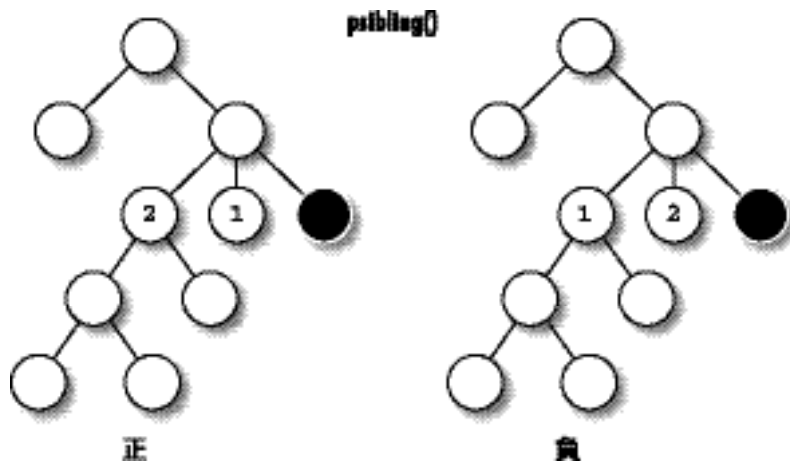


圖 3.10 : psibling() 的路徑

ancestor()

ancestor() 就像族譜學家一樣，因為 ancestor() 會追蹤一個節點的所有祖先，直到 root() 為止。用正的引數時，ancestor() 會向上搜尋，從來源位置的父節點開始，止於 root()。用負數時，從 root() 開始，止於來源位置的父節點。圖 3.11 說明了節點搜尋的順序。

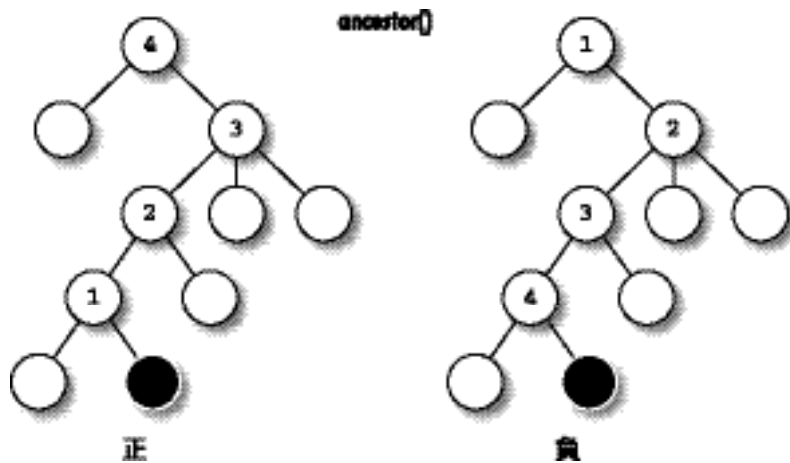


圖 3.11 : ancestor() 的路徑

例如，要找出員工所在的 `<department>`，你可以使用 `ancestor(1,department)`。要找出員工的上司（如果有的話），可以使用 `ancestor(1,employee)`。請注意，若起始點是副總裁的元素，該位置項目會因比對不出結果而失敗。

有許多方法可以走到相同的位置。要定出 Mary A. 的 `<employee>` 元素，下面的範例都做得到：

```
root().child(1,personnel).child(1).child(1).child(3).child(1).child(3).
  child(2)
root().child(1,personnel).(1).(1).(3).(1).(3).(2)
root().child(1,personnel).following(1,*,id,'marketing').
  preceding(2,employee)
id(sales).descendant(4,employee)
id(sales).descendant(-2,employee)
```

## 字串

相對項目討論到現在，都只能用於整個節點。即使是用 `#text` 關鍵字，相對項目所比對出來的文字，也是相鄰節點間的所有文字。如果我們要的是比較小的子集（像是某個字），或是某一段文字，其中還摻雜內線元素（諸如完整的一個文字段落），那麼之前所討論的相對項目就幫不了忙。此時，`string()` 就可派上用場了。

`string()` 可以帶有兩到四個引數。這幾個引數可與前面談過的相對位置項目的引數相比擬。第一個引數指定一個實體，而第二個引數是要搜尋的字串。例如，`string(2, "bubba")` 會在來源位置中找第二個“bubba”字串。`string(all, "billy")` 會找出節點中所有的“billy”。

`string()` 不僅僅侷限於字。`string(2, "B")` 可以找出“Billy-Bob”字串中的第二個“B”。比對是分大小寫的，所以，`string(2, "b")` 會找不到東西，因為“Billy-Bob”？面只有一個“b”。XML 不支援不分大小寫的比對，因為，那必須在不同的文化標準中，做邏輯判斷。例如，中文字有大小寫的問題嗎？

`string()` 另一個有用的模式是計算一般字元。空的 `string("")` 會比對任何的字元。`string(23, "")` 是來源位置中第二十三個字元。如果知道東西在那？，但不知道是什麼的時候，這種用法就很有用。

第三和第四個引數是定義所要回傳的子字串之位置和大小。例如，`string(1, "Vasco Da Gama", 6, 2)` 會搜尋字串 "Vasco Da Gama"，找到後傳回 "Da"，也就是從頭算起的第六個字元，長度共兩個字元。這個方法就像條件式敘述，先找到主字串，再來處理主字串中的子字串。

子字串不見得只限於搜尋到的字串。偏移量可以偏到字串外邊，放眼到節點中其餘的文字。用 `string(1, "Ascott", 11, 8)` 搜尋文字 "The Ascott Incident"，可以找到字串 "Incident"。

請注意，找到的物件不見得非含有字元不可，也許只是一個點而已。如果我們將前面的第四個引數設為零，那麼所找到的就是 "l" 字串前面的點。對於用滑鼠來操作的使用者而言，這恐怕不是好連結，但絕對是可行的連結標的，或是可從別的網頁中，插入一段文字的安插點。

## 測量

你要找的東西不見得都是純粹的元素或元素中的文字。因此，XPointer 給你一種方法，來找出兩個物件間的任何東西。所用到的位置項目是 `span()`，其語法如下：

```
span(XPointer,Xpointer)
```

例如，你可以指定一個值域，從強調字 "very" 橫跨到強調字 "so"，如下所示：

```
root().span(descendant(1,emph),descendant(2,emph))
```

## XLink 簡介

XML 的連結規則是由 XML 連結語言 (XML Linking Language, XLink) 規格所定義。XML 的任何元素都能作為連結元素。這是一定的，因為 XML 並沒有任何的預設元素。既然你能自訂元素，也必須能讓它們彼此連結。XLink 的語法和功能是從 HTML 的成功 (和某些失敗) 經驗而來的。XLink 和舊的 HTML 連結相容，但增加了更多的彈性和機能。

一般而言，HTML 用的是兩種連結。<A> 元素會建立一個連結，但卻無法自動載入；當使用者點選該連結時，另一端的文件才會替換當前文件。<IMG> 元素會默然地自行運作，連結到圖形資料，然後匯入文件中。

為了比較起見，讓我們看一下 XLink 如何改進 HTML 連結：

- 任何 XML 元素都可做成連結。HTML 中只有少數元素有連結功能。
- XLink 可以用 XPointer 到達文件中的任何地點。HTML 連結指定的特定文件位置，必須有專門的連結錨來接收該連結。標的文件的作者必須考慮每個可能的連結，然後加上連結錨。
- XML 可以用 XLink 匯入文字和標記。HTML 無法從標的文件中，將文字內嵌到來源文件。
- XPointer 可定義某個範圍的 XML 標記，以指涉文件中的子集。HTML 的連結只能參考單一點或整個檔案。

## 設定連結元素

任何 XML 元素都可用 XLink 屬性設成連結：type、href、role、title、show 和 actuate。使用這些屬性時，你必須使用名稱空間前置字，對應到 XLink URI。XML 處理器會使用這個名稱空間，將屬性解譯成連結參數。以下是用這些屬性做出來的連結元素範例：

```
<cite
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="simple"
  xlink:href="http://www.books.org/huckfinn.xml"
  xlink:show="new"
  xlink:actuate="onRequest"
>Huckleberry Finn</cite>
<graphic
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="simple"
  xlink:href="figs/diagram39.png"
  xlink:show="embed"
```

```
        xlink:actuate="onLoad"  
    />  
    <dateref  
        xmlns:xlink="http://www.w3.org/1999/xlink"  
        xlink:type="simple"  
        xlink:href="http://dataserv.buggs.com/db.xml#entry92"  
        xlink:actuate="onLoad"  
        xlink:show="embed"  
    />
```

第一個例子是指向網站上的某一本書。第二個例子是從本地檔案中匯入圖形。第三個例子從一個檔案中接收一段資訊。應用軟體會決定要如何顯示這些連結。

任何 XLink 至少要有 `type` 屬性。這是解析器決定元素是否為連結元素的關鍵字。`type` 的值會決定 XLink 的種類：此例為 `simple` (簡單)。

簡單類型的 XLink 必須用 `href` 屬性來定義目標。`href` 是根據一個 HTML 的屬性之命名而得到的，也就是告訴 `<A>` 元素要連結到那？，才能讓 XML 相容於 HTML 文件。`href` 的值是連結另一端的 URI，可以指涉整份文件，或文件中的某個點、或某個元素。



XML 解析器沒有義務去確認你所指涉的遠端資源是否真如你所說的在那兒。URL 也許不對，但文件仍然合於結構良好的要求，且有效。這和之前提到的內部連結相反，內部連結的 `ID` 屬性必須唯一，而且 `IDREF` 屬性必須指到實有的元素。原因是因為內部連結的範圍位於同一份文件上，通常也在同一系統內。即使建立網路連線只須幾秒鐘，真的要做 URL 檢核，恐怕會使得解析變成無止盡的磨難。

其餘的屬性皆是可有可無。因為 XLink 規格還很年輕，用處還看不太出來。儘管如此，我們還是會在下面幾節中，討論幾種可能的用法。

## 行為

就像說明何謂 XLink 是很重要的事一樣，你也會想知道 XLink 是如何運作的。XML 處理器應立刻啟動該連結，還是等到使用者通知才啟動？連結應該安插文字或資料到本地文件？抑或將使用者帶到標的資源？這一節所要談的屬性，就是用來提供這類資訊的。

屬性 `actuate` 指出何時該啟動 XLink 連結。你可能想在網頁上放一些連結（如圖片和匯入某些文字），在網頁排版時，就啟動連結。此例中，遠端資源的資料會自動被 XML 處理器取出，然後由應用軟體按其需求處理，再和文件整合。`onLoad` 就是宣告連結應立刻啟動。

`onRequest` 是指把連結的啟動交由閱讀者決定。連結會維持冬眠狀態，直到使用者點取為止，而其餘的屬性就是用來決定連結的最終結果。至於使用者該如何啟動連結則沒有指定。閱讀者可能必須點應用軟體內的一個控制項（control），或在文字瀏覽器中使用鍵盤命令，或說出一道命令給聽得懂人話的瀏覽器。啟動的方法由 XML 處理器決定。

`show` 屬性是描述連結啟動後的行為（自動啟動或由使用者啟動）。此時的問題是該對標的資源的資料做何處理，一共有三種選擇：

`embed`

遠端資源的資料應該在連結元素的位置中顯示出來。

`replace`

當前文件應該被移除，由遠端文件替換。

`new`

如果可能的話，瀏覽器應該以某種方式建立新的背景。例如，瀏覽器可能打開新的視窗，來顯示遠端資源的內容，而不會移除本地資源。

以下是使用行為屬性的範例：

```
<para>The quote of the day is:</para>
<para>
  <program-call xlink:type="simple"
    xlink:href="bin/quote-o-matic.pl"
    xlink:actuate="onLoad"
    xlink:show="embed"/>
</para>
```

這個 XLink 會呼叫程式，使其傳回文字。為了方便起見，我們不去解釋資料怎麼傳回來，而是把重心放在資料傳回來之後，會發生什麼事。此例中，我們將資料內嵌到文件中，使其成為文件的一部份文字。閱讀者不知道我們呼叫了另一個程式，因為整個網頁是一次做出來的。

此例中，啟動方法設為 `onLoad`，不過，我們可以假設改用 `onRequest`。如此一來，使用者可以點取標示文字（也許是“click here”），在同一個地方帶進另一段文字。XML 並不會替你決定連結該長什麼樣子。

## 說明文字

XLink 提供了幾個地方，可讓你增加有關連結的說明文字。這種資訊可有可無，但如果閱讀者想知道他們正要觀看的東西為何，以及是否有連進去看的價值時，這樣的資訊也許就有用處。元素內容是其中一個地方。以下列的連結為例：

```
A topic related to rockets is
<related
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="simple"
  xlink:href="planes.xml"
>Airplanes</related>
```

連結元素內容的角色不限於一格。如果連結具有屬性 `actuate="onRequest"`，那連結的內容（Airplanes）可能就是用來供使用者點取、且啟動連結的標示文字。另一方面，如果有屬性 `actuate="onLoad"`，內容可能只是一個標題。通常，會自動載入標的資源的元素不會有內容。

`role` 屬性是用來說明遠端資源的性質或功能，以及遠端資源和文件間的關係。`role` 的值一定是 URI，但是就像名稱空間，只是一個唯一的標示碼，而非指向特定資源的指標。例如：

```
<image
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="simple"
  xlink:href="images/me.gif"
  xlink:role="http://www.bobsbolts.com/linkstuff/photograph"
/>
```

此例中，我們所指出標的資源是 `photograph`，可以和 `cartoon`、`diagram`、`logo` 或其它可能出現在文件中的 `<image>` 元素有所區別。做這種區別的原因之一是在樣規中，你可用 `role` 屬性，來決定每種角色該如何處理。例如，你可以給照片一個大窗格，圖片是小邊框，而商標沒有邊框。

`title` 屬性也能用來描述遠端資源，但是，`title` 的目的主要是供人閱讀，而非處理之用。在 `<image>` 的例子中，可能是圖片的說明文字：

```
<image
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="simple"
  xlink:href="images/me.gif"
  xlink:role="http://www.bobsbolts.com/linkstuff/photograph"
  xlink:activate="onLoad"
  xlink:title="A picture of me on the beach."
/>
```

如果連結是由使用者啟動再連到另一份文件，`title` 大概就是該文件的標題。XML 程式要如何使用 `title`，並沒有完善的定義，全仰賴 XML 處理器。

## XML 應用語言：XHTML

學習實務世界中連結的用法的好地方就是 HTML。HTML 是建構網頁的語言。超文字 (Hypertext) 是連接相關文件的連結文字，讓 WWW 成為今日高度成功的通訊媒體。

HTML 提供了一個簡單的架構，讓一般文件顯示於螢幕上。HTML 擁有一小組元素，扮演著設計文件結構的根本角色，而沒有花俏的技術。有一些標題元素提供標題之用（<h1>、<h2> 等等），有段落（<p>），有清單（<ul>、<ol>），有表格（<table>），簡單的內線元素（<em>、<tt>）等等。HTML 無法做細節的配置，但足以將文件顯示於螢幕上供人觀看。

我們要探討的是新版的 HTML：XHTML。XHTML 幾乎和 HTML 4 相同，但有一些限制使其滿足 XML 規則。每頁 XHTML 網頁都是完整的 XML 文件，遵隨著 XML 1.0 標準，且所有通用的 XML 工具和處理器都能處理。如果你按照下列的準則來寫 XHTML，XHTML 文件也能相容於現在的 HTML 瀏覽器。

使用 XHTML 有幾個好處：

- 由於 XHTML 遵守了 XML 標準，XHTML 文件可用任何通用的 XML 編輯器、驗正器、瀏覽器或其它專門用來處理 XML 文件的工具來處理。
- 遵照 XML 嚴格規則的文件會比較清楚且可預測，而且瀏覽器和 XML 軟體在處理時比較不會出錯。
- XML 的可擴展功能最終會讓 XHTML 受益無窮，使得 XHTML 可以輕鬆的新增元素和機能，就像宣告名稱空間或使用不同的 DTD 那樣簡單。

XHTML 目前有三種型式：嚴格式（strict）、過渡期式（transitional）和分割畫面式（frameset）。如下所述：

#### Strict XHTML

和 HTML 有清楚的界線，當初許多 HTML 用來表達版面配置的元素都被放棄了。你必須用 CSS 樣規來排版文件。Strict XHTML 最像 XML，也是這三種類型最終的依歸。

#### Transitional XHTML

如果你想讓網頁與舊式不支援樣規的瀏覽器相容，可改用這一型的 XHTML，因為這類的 XHTML 保留了 HTML 的元素和屬性。例如，字型和顏色的設定都能使用。

## Frameset XHTML

Frameset XHTML 很像 strict XHTML，但是它加進了框架（frame）。將支援框架特徵的版本與其它版本獨立，對於不需要框架的人而言，會簡單許多。

你可以在 DTD 的文件類型宣告中，指定你要用的 XHTML 類型。下列的宣告是屬於 strict XHTML：

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

如果你在本地系統中安裝這個 DTD，你應該修改系統辨識碼，改用你自己的路徑。將這個 DTD 複製到本地系統，可以減少文件載入的時間。XHTML DTD 和資訊是由 W3C 維護的（參照附錄 B）。

讓我們看一個範例。範例 3.2 是 strict XHTML 文件。

### 範例 3.2：一份簡單的 XHTML 文件

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

<html ❸
    xmlns="http://www.w3.org/1999/xhtml" ❹
    xml:lang="en" lang="en"> ❺

    <head>
        <title>Evil Science Institute</title>
    </head>

    <body>
        <h1>Evil Science Institute</h1> ❻
        <p><em>Welcome</em> to Dr. Indigo Riceway's Institute for Evil
            Science!</p> ❼

        <h2>Table of Contents</h2>
```

## 範例 3.2：一份簡單的 XHTML 文件（續）

```
<ol>
  <li><a href="#staff">Meet Our Staff</a></li> ❶
  <li><a href="#courses">Exciting Courses</a></li>
  <li><a href="#research">Groundbreaking Research</a></li>
  <li><a href="#contact">Contact Us</a></li>
</ol>

<a name="staff" /> ❷
<h2 id="staff">Meet Our Staff</h2>
<dl>
  <dt><a href="riceway.html">Dr. Indigo Riceway</a></dt>
  <dd>
     ❸
    Founder of the institute, inventor of the moon magnet and the
    metal-eating termite, three-time winner of Most Evil Genius award.
    Teaches Death Rays 101, Physics, Astronomy, and Criminal Schemes.
  </dd>
  <dt><a href="grzinsky.html">Dr. Ruth "Ruthless" Grzinsky</a></dt>
  <dd>
     ❹
    Mastermind of the Fort Knox nano-robot heist of 2002.
    Teaches Computer Science, Nanotechnology, and Foiling Security
    Systems.
  </dd>
  <dt><a href="zucav.html">Dr. Sebastian Zucav</a></dt>
  <dd>
    
    A man of supreme mystery and devastating intellect.
    Teaches Chemistry, Poisons, Explosives, Gambling, and
    Economics of Extortion.
  </dd>
</dl>

<a name="courses" />
<h2 id="courses">Exciting Courses</h2>
<p>  Choose from such
```

```
intriguing subjects as</p>
<ul>
  <li>Training Cobras to Kill</li>
  <li>Care and Feeding of Mutant Beasts</li>
  <li>Superheros and Their Weaknesses</li>
  <li>The Wonderful World of Money</li>
  <li>Hijacking: From Studebakers to Supertankers</li>
</ul>

<a name="research" />
<h2 id="research">Groundbreaking Research</h2>
<p>Indigo's Evil Institute is a world-class research facility.
  Ongoing projects include:</p>

<h3>Blot Out The Sky</h3>
<p>A diabolical scheme to fill the sky with garish neon advertisements
  unless the governments of the world agree to pay us one hundred
  billion dollars. Mha ha ha ha ha!</p>

<h3>Killer Pigeons</h3>
<p>A merciless plan to mutate and train pigeons to become efficient
  assassins, whereby we can command huge bounties by blackmailing
  the public not to set them loose. Mha ha ha ha ha!</p>

<h3>Horror From Below</h3>
<p>A sinister plot so horrendous and terrifying, we dare not
  reveal it to any but 3rd year students and above. We shall only
  say that it will be the most evil of our projects to date!
  Mha ha ha ha ha!</p>

<a name="contact" />
<h2 id="contact">Contact Us</h2>
<p>If you think you have what it takes to be an Evil Scientist,
  including unbounded intellect, inhumane cruelty, and a sincere
  loathing of your fellow man, contact us for an application. Send
  a self-addressed, stamped envelope to:
</p>
```

## 範例 3.2：一份簡單的 XHTML 文件（續）

```
<address>The Evil Science Institute,  
Office of Admissions,  
10 Clover Lane,  
Death Island,  
Mine Infested Waters off the Coast of Sri Lanka</address>  
  
</body>  
</html>
```

說明如下：

- ❶ 雖然這個範例不需要 XML 宣告，但使用 XML 宣告是好習慣，特別是當你要用的字元集不是 UTF-8 時，就非用不可。問題是，有些舊的 HTML 瀏覽器無法正確解讀 PI，也許會把 XML 宣告印出來。
- ❷ 文件類型宣告是必要的，用於確認 XHTML 的版本和類型。請注意，你不能將內部子集的位置當成宣告，會造成許多懂 XHTML 的瀏覽器的排斥。
- ❸ 根元素永遠都是 `<html>`。請注意，XHTML 的元素都是小寫，沒有例外。這一點和 HTML 不同，因為 HTML 不分大小寫。
- ❹ 宣告預設名稱空間也是必須的。三種 XHTML 類型的名稱空間都是 `http://www.w3.org/1999/xhtml`。
- ❺ 在 transitional XHTML 文件中，你應該同時使用 `<lang>` 和 `<xml:lang>` 元素。有些瀏覽器看不懂 `<xml:lang>`，但是，看得懂 `<xml:lang>` 的瀏覽器就會以 `<xml:lang>` 的設定值為優先。
- ❻ 這個 `<h1>` 元素是節區標題的範例。DocBook 的節區完全放在特殊的元素中（如 `<sect1>`），而 XHTML 並未提供任何節區元素。相反地，只有內含標題的元素。但藉由標題的樣式，人們也可以瞭解新的節區開始了。這是版面配置資訊爬進標記中而犧牲結構的例子。
- ❼ XHTML 和舊式 HTML 一個很重要的區別在於，所有含有內容的元素現在都必須有一個結尾標籤。在以前，有時省略並不會出事。`<p>` 也必須包括起始和結尾標籤，即使標籤中不含任何內容亦然。

- ❶ 這？的 `<a>` 元素是連結同一文件中某個點的簡單連結。不陌生的 `href` 屬性含有片段辨識碼。DTD 中在定義 `<a>` 元素時，隱藏其它 XLink 屬性（第五章會學到如何把屬性變成隱性屬性）。這個連結要由使用者啟動，元素的內容會做不同排版設計，進而變成一個控制項。當使用者點取這個控制項時，就會啟動連結。啟動時，就會立刻跨進連結，替換當前文件。
- ❷ 這個 `<a>` 元素使用一個 `name` 屬性，來提供要連結的目標。你可用 XPointer，來連結任何 XML 元素。因此，使用特定元素，專門做為連結標的的技術已經落伍了；`<a>` 元素是為了和舊式瀏覽器相容才留下來的。
- ❸ 這？是空元素的範例，結尾分隔字（`/>`）遵照 XML 良好結構的規則。然而，我們在 `/>` 前面加了多餘的空格，這樣可幫助某些瀏覽器，把容器和空標籤區分開來。不能有內容的元素，就不要用容器元素的語法（例如，`<br></br>`），因為可能會出現不可預期的結果。
- ❹ `<img>` 是另一個連結元素的例子，此例是匯入並顯示圖檔。和 `<a>` 不同的是，`<img>` 的設定值是 `actuation="auto"` 和 `show="embed"`。這表示所有圖片在網頁做排版設計時，就要匯進來並當成文件的一部份，予以顯示。
- ❺ 在排版時，XHTML 的所有元素都會忽略空白，但 `<pre>` 除外。格式器會把內容前後的空格丟掉，把多餘的空格壓縮成單一空格。為了得到好看的段落，這樣做是必須的，儘管空格、跳格和換列字元都是用來排定 XML 的格式，使其更具可讀性亦然。XML 的元素都會保留空白，除非特別指定不要保留。

現在你應該瞭解，XHTML 是 HTML 演化的一大步。網頁會變得更明確，相容於更多瀏覽器，而且首次能和 XML 工具合作。將樣式設定值從標記中移走，就如 strict XHTML 所做的，會迫使作者改用樣規。要如此依賴樣規，就表示樣規的支援和豐富的版面配置，將會快速發展。

未來，XHTML 會走向模組化（modularity），也就是說，DTD 會變成由各個可以互換的部份所組成。到那個時候，HTML 文件也能混合、比對元素，使文件能適用各種用途，包括網際網路的各類應用、無線服務、合成語音客戶程式等等。

話雖如此，但 XHTML 並非各種標記問題所要的答案。XHTML 的元素也許無法達到你想要的細節，且欠缺巢狀節區結構，也是建立大型複雜文件的障礙。但就一般用途而言，如果要用來展示網頁，XHTML 就是最佳的選擇。