1. ＸＭＬ路徑語言

1.1. XPath 簡介

XPath (*XML Path Language*, **XML 路徑語言**) 是用來對 XML 本文進行搜尋、過濾的一種技術。XPath 不是使用 XML 的語法，取而代之的是一種簡化的路徑語法。

在 XML 家族中的許多重要的技術，例如 **XSLT** (*Extensible Stylesheet Trasnsformtion*)、**XQuery** (*XML Query Language*) 都會使用 XPath 表示式來搜尋 XML 本文中的資料。

XPath 的規格書可以在 W3C 網站中下載。

http://www.w3.org/TR/1999/REC-xpath-19991116

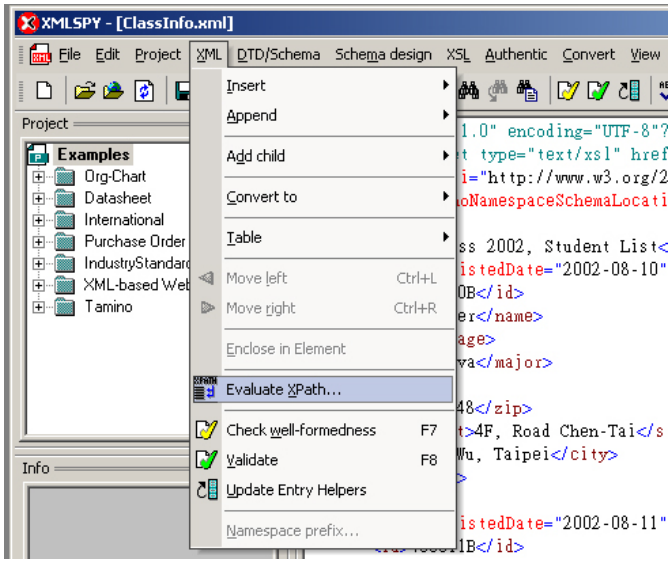
1.2. 評估 XPath 的工具

在學習寫作 XPath 的過程中可以透過 XML Spy® 中的 XPath 評估工具來檢驗結果是否正確。

若要取得最新的 XML Spy 請至下列網址下載：

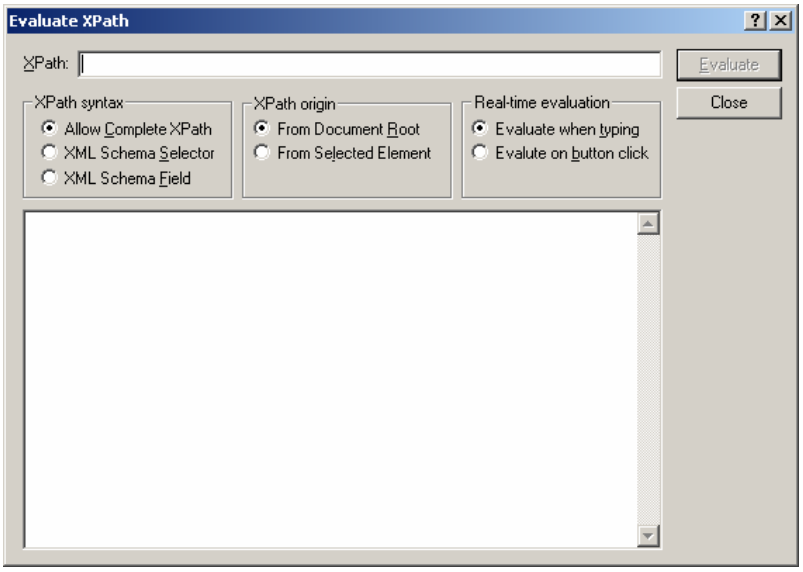
http://www.xmlspy.com

在安裝並註冊完成後，請開啓一 XML 檔案，並拉下 “XML” 功能表，選擇「Evaluate XPath ...」項目，如同圖表 1-1 一樣。



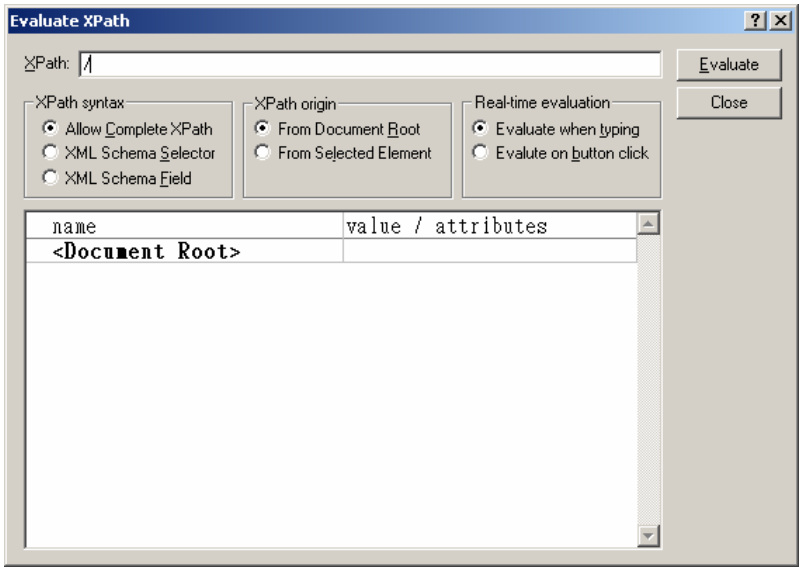
圖表 1-1

當選擇 “Evaluate XPath...” 之後會出現下面的畫面。



圖表 1-2

只要將 XPath 表示式輸入於圖表 1-2 中的 XPath 欄位中，該程式就會在輸入的同時，順便產生所 XPath 表示式的結果，例如：



圖表 1-3

1.3. XPath 資料模型

在 XPath 的**資料模型**(*data model*)中，XML 文件被視爲是由許多**節點**(*node*)所組成的**樹狀**(*tree*)結構。而在 XPath 中共有七種不同的節點，這一些節點分別爲。

節點類型	節點敘述
root node	根節點
element node	元素節點
attribute node	屬性節點
text node	文字節點
comment node	備註節點
processing instruction node	處理指令節點
namespace node	命名空間節點

圖表 1-4

在 XPath 樹中，每份 XML 文件都是由**根節點**(*root node*)開始，在根節點下可以擁有**元素節點**(*element node*)、**備註節點**(*comment node*)或是**處理指令節點**(*processing instruction node*)作爲其**子節點**(*child node*)，而不同的節點還可以擁有其子節點。

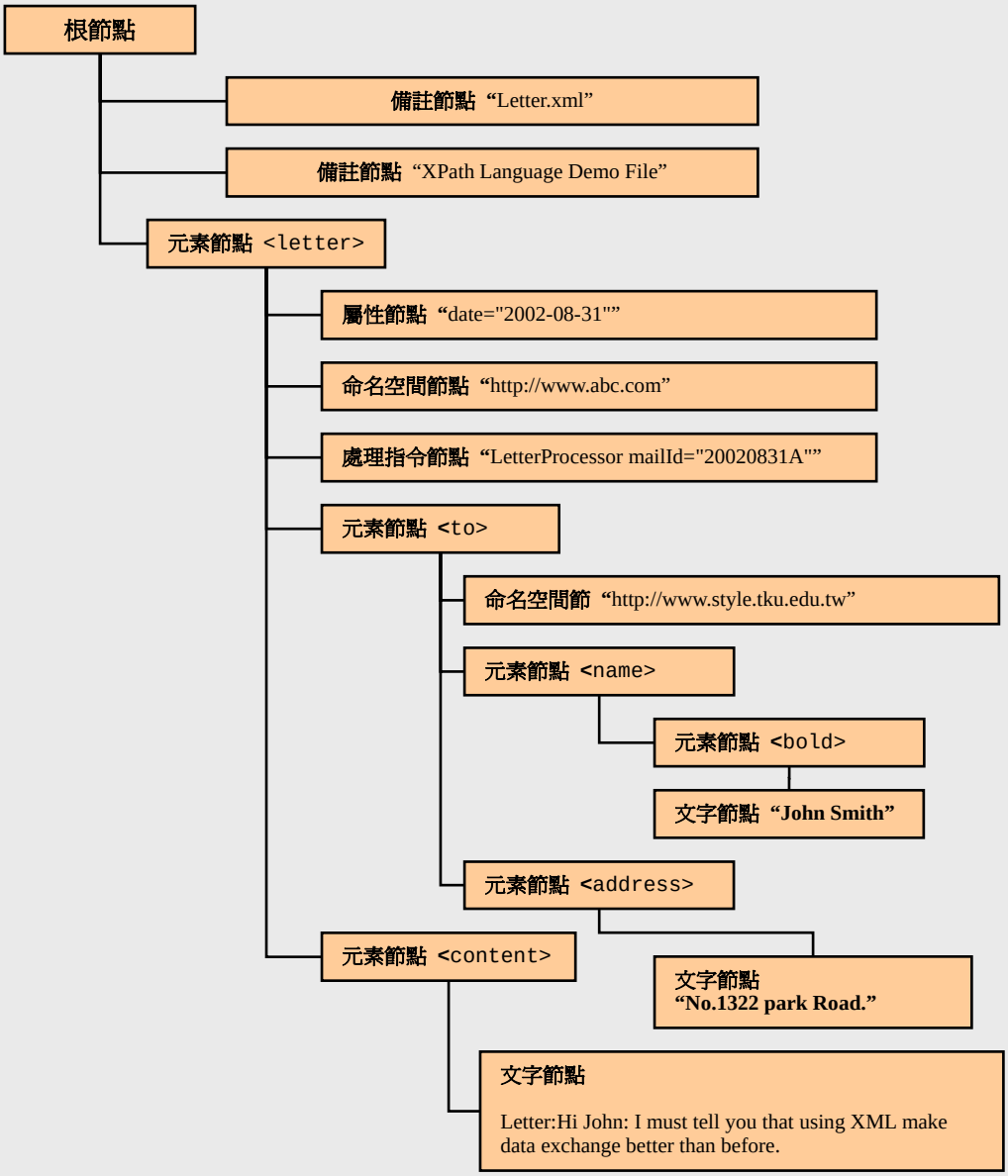
每一個 **XPath 樹** (*XPath tree*) 的節點，都會和 XML 文件中元素的**起始標籤**(*start tag*) 出現的次序相符合。這種次序被稱爲**文件次序**(*document order*)。所以根節點會是第一個節點，而元素節點出現次序一定優先於其子節點或是其屬性節點。

圖表 1-5 中的 Letter.xml 將用來說明並示範 XPath 中七種節點所在的位置與角色。

```
1  <?xml version="1.0"?>
2  <!-- Letter.xml -->
3  <!-- XPath Language Demo File -->
4  <letter xmlns = "http://www.abc.com" date="2002-08-31">
5      <?LetterProcessor mailId="20020831A"?>
6      <to xmlns:style="http://www.style.abc.com" >
7          <name>
8              <style:bold>John Smith</style:bold>
9          </name>
10         <address>No.1322 Park Road.</address>
11     </to>
12     <content>
13         <![CDATA[
14             Letter:Hi John:
15             I must tell you that using XML make data
16             exchange better than before.
17         ]]>
18     </content>
19 </letter>
```

圖表 1-5

圖表 1-6 是將圖表 1-5 的 Letter.xml 轉換成 XPath 樹。



圖表 1-6　Letter.xml 的 XPath 樹狀結構

1.3.1. 根節點

根節點(*root node*)是 XPath 件樹的**根**(*root*)。根節點可擁有的子節點分別爲**元素節點**、**處理指令節點**和**備註節點**(位於 XML 文件**前言**(*prolog*)或是**文件元素**(*document element*)尾端之後的備註)。

*（注意！根節點不是***根元素**(*root element*)*節點）*

1.3.2. 元素節點

元素節點(*element node*)的子節點可以是**元素節點**、**備註節點**、**處理指令節**

點與**文字節點**。由於元素中可以擁有**獨一識別符**(*unique identifier*)，例如 DTD 中的 ID 型別，所以 XML 文件中不可以出現兩個有相同識別符的元素，若 XML 處理器發現有兩個以上重複的識別符時，則第二個重複的元素將視爲沒有識別符。

1.3.3. 屬性節點

元素中可攜帶屬性，而 XPath 中代表屬性的節點爲**屬性節點**(*attribute node*)。**元素**一定爲屬性節點之**父**(*parent*)，但是屬性節點並非元素之**子**(*child*)。也就是說，元素節點可包含的節點類型並沒有屬性節點類型；這是因爲父節點與子節點之關係必須是父節點包含子節點，但在屬性節點中，元素節點並沒有包含屬性節點，屬性節點只是包含了描述元素的資訊而已。所以在位置路徑(§1.4)中如果使用“**`child::`**”將不會選取到任何屬性節點。

1.3.4. 命名空間節點

元素可含有相關聯的**命名空間**(*namespace*)，而在 XPath 樹中命名空間的節點有兩種，一種是區別不同命名空間的**命名空間前置符號**(*namespace prefix*)，另外一種是元素預設命名空間節點，而且元素間不可共享命名空間節點。命名空間節點的父親必須是元素，但是命名空間節點並不是元素節點的子孫，這點和屬性節點在觀念上是相同的。

1.3.5. 處理指令節點

處理指令節點只針對那些不是屬於 DTD 宣告中的處理指令元素有效。

1.3.6. 備註節點

備註解點只針對那些不是屬於 DTD 宣告中的備註元素有效。

1.3.7. 文字節點

文字節點是將文字資料集合在一起，而文字節點盡可能的將 XML 文件中的 CDATA 區域中文字資料集合在一起，所以文字節點前後不可有其他文字節點，而且文字節點至少由一個字元資料組成。

1.4. 位置路徑

XPath 的**位置路徑**(*location path*)是一種用來尋找 XPath 樹中各種節點的**表式示**(*expression*)。位置路徑是由**位置步驟**(*location step*)所組成，每一個步驟可以是**軸線**(*axis*)、**節點測試**(*node test*) 或是**述語**(*predicate*)。

以下是一個位置路徑的範例：

```
/child::class/child::student/@registedDate
```

位置路徑表示式是由 "/" 作爲分隔不同位置步驟的符號，所以上面的範例是由三個位置步驟所組成。而位置路徑表示式所傳回的結果有以下四種：

- **節點集合** (*node set*) ，節點集合可能是一些不重複但沒有次序的節點，以上例來說，可能傳回一堆 registedDate 屬性節點。
- **布林值** (*boolean value*)，可能傳回 "TRUE" 或是 "FALSE"。
- **數字** (*number*)，例如統計文件中某個元素數量的結果。
- **字串** (*string*)，例如傳回某個文字節點的內容，可能是由 Unicode 所組成的字串。

下圖表 1-7 將用來作爲討論位置路徑的範例文件。

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <class xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3        xsi:noNamespaceSchemaLocation="ClassInfo.xsd"
4        year="2002">
5      <comment>Class 2002, Student List</comment>
6      <student registedDate="2002-08-10">
7          <id>400010B</id>
8          <name>Peter</name>
9          <age>49</age>
10         <major>Java</major>
11         <address>
12             <zip>248</zip>
13             <street>4F, Road Chen-Tai</street>
14             <city>Wu, Taipei</city>
15         </address>
16     </student>
17     <student registedDate="2002-08-11">
18         <id>400011B</id>
19         <name>John</name>
20         <age>29</age>
21         <major>C++, Java, Win32 SDK</major>
22         <address>
23             <zip>100</zip>
24             <street>2F, No.1, Ave. 4</street>
25             <city>Taipei</city>
26         </address>
27     </student>
```

圖表 1-7

1.4.1. 本文

在 XPath 樹中搜尋節點都是由**本文節點**(*context node*)開始，並以本文節點的相對路徑來尋找特定節點。XPath 中的本文通常由五個部份所組成，這五個部份分別爲：

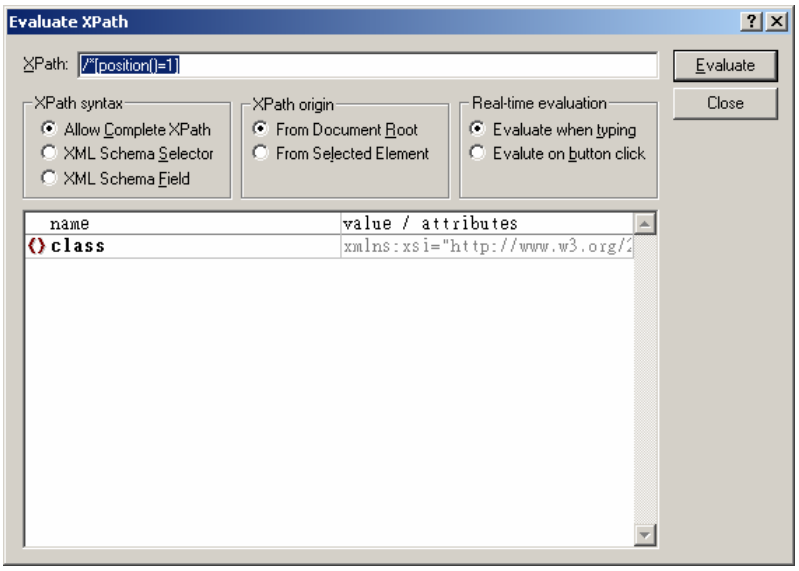
- **一個節點**。
- **一對非零之正整數**。
- **一個變數集合**。
- **一個函數**。
- **一個命名空間**。

本文位置(*context position*)必須小於等於**本文大小**(*context size*)。在 XPath 中本文位置都是由 “1” 開始計算，本文位置可以藉由 position() 函數判斷取得，例如在圖表 1-7 中，若將本文節點設定爲 <class> 元素中的 <student> 元素，則該本文節點的大小爲 “2”，所以註冊日爲 “2002-08-10” 的 <student> 元素的本文位置爲 “1” ，而註冊日爲 “2002-08-11” 的 <student> 元素的本文位置就是 “2” 了。

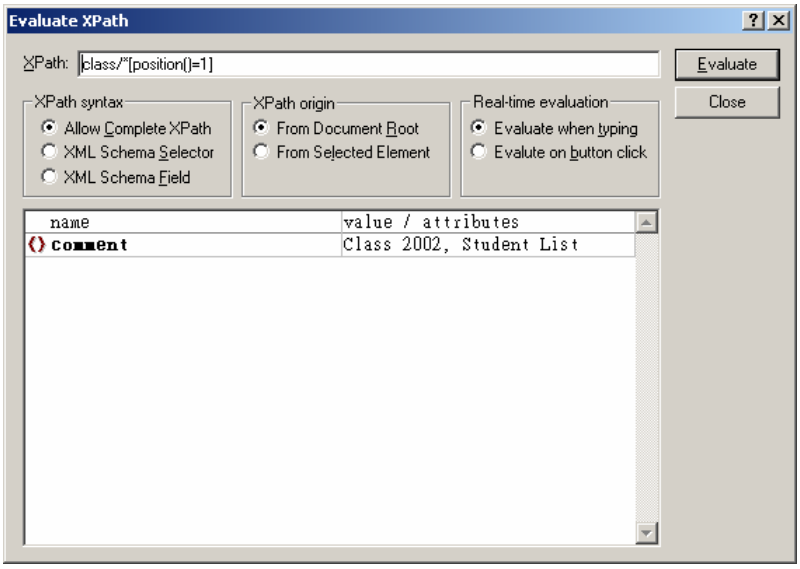
下面範例使用了 position() 函數取出第二個本文位置的 student 節點。

```
/child::class/child::student[position()=2]
```

若令根節點爲本文節點並找出本文位置爲 “1” 的節點，則結果會挑選出元素節點 <class>，請見圖表 1-8；若令本文節點爲 <class> 則會挑出本文位置爲 “1” 的節點結果會是元素節點 <comment>，請見圖表 1-9。



圖表 1-8



圖表 1-9

1.4.2. 軸線

軸線(*axes*)是用來指出相關於本文節點位置的節點，如果搜尋本文節點以下的節點，則稱爲**順向軸線**(*forward axes*)。如果搜尋本文節點之前的節點，則稱爲**反向軸線**(*reverse axes*)。

下面表格中整理了所有軸線的定義。

軸線名稱	搜尋次序	描述	範例
self	無	本文節點自己	若本文節點是根節點，則 `self::*` 將取出文件的**根節點**(*document root*)。
parent	反向	本文節點的父親	若本文節點是 `<student>` ，則 `parent::*` 將取出元素節點 `<class>`
child	順向	本文節點的小孩	若本文節點是 `<student>`，則 `child::*` 將取出所有在 `<student>` 下的節點，例如 `<id>, <name>, <age>` 等等。
ancestor	反向	本文節點的祖先	若本文節點是 `<student>`，則 `ancestor::*` 將取出所有屬於 `<student>` 元素祖先的節點，例如 `<class>`,文件的根節點等等。
ancestor-or-self	反向	本文節點的祖先或自己	若本文節點是 `<student>`，則 `ancestor-or-self::*` 將取出自己與所有屬於 `<student>` 元素祖先的節點。
descendant	順向	本文節點的子孫	若本文節點是 `<student>` ，則 `descendant::*` 將取出所有是屬於 `<student>` 子孫的節點。
descendant-or-self	順向	本文節點的子孫或自己	若本文節點是 `<student>` ，則 `descendant-or-self::*` 將取出自己與所有屬於 `<student>` 子孫的節點，連最枝葉的文字節點都會取出。
following	順向	本文節點後的節點，不包含本文節點的子孫節點	若本文節點是 `<student>`，則 `following::*` 將會取出本文節點之後出現的所有節點，例如第二個 `<student>` 元素及其子孫節點。
following-sibling	順向	跟著本文節點後的兄弟節點	若本文節點是 `<student>` ，則 `following-sibling::*` 將會取出本文節點 `<student>` 之下一個兄弟節點 `<student>`。
preceding	反向	本文節點之前點，不包含本文節點的子孫節點	若本文節點是 `<class>` ，則 `preceding::*` 將會取出本文節點之前的節點，例如 XML 版本宣告屬性節點、處理指令節點等。
preceding-sibling	反向	本文節點之前的兄弟節點	若本文節點是 `<class>`，則 `preceding-sibling::*`將會取出本文節點之前的節點，例如 XML 宣告的處理指令節點。

attribute	順向	本文節點的屬性節點	若本文節點是 <class>，則 attribute::* 將會取出 <class> 內所有的屬性節點。
namespace	順向	本文節點的命名空間節點	若本文節點是 <class>，則 namespace::* 會取出宣告 Schema 實體文件的命名空間節點。

圖表 1-10

圖表 1-10 中的範例都是以軸線與節點測試所組成，例如：

```
attribute::node()
```

在 §1.4 中提到了，路徑位置是由一到多個路徑步驟所組成。而 attribute::node() 便是一個節點步驟。一個節點步驟必須是 **軸線::節點測試** 所組成，下一節將討論節點測試。

每一個軸線都會對應到一種**原則節點型態**(*principal node type*)，例如軸線 attribute 所跟隨的原則節點型態就是屬性節點，而軸線 namespace 對應到的原則節點型態就是命名空間節點，而其他軸線的原則節點型態則都是對應到元素節點。

1.4.3. 節點測試

透過使用 XPath 的軸線可以在 XPath 樹中選擇出零到多個節點，而節點測試可以更仔細的篩選出所需要的節點。

節點測試	描述	說明
*	選擇所有原則相同節點	例如 attribute::* 將篩選出所有原則節點型態爲屬性的節點、 child::* 將選出所有原則節點型態爲元素的節點。
node()	選擇所有節點	node() 將會選擇出所有節點，不管他們的原則節點型態爲何。
text()	選擇文字節點	
comment()	選擇備註節點	
processing-instruction()	選擇處理指令節點	
node name	選擇所有符合名稱的節點	例如 class/student 將篩選出所有在 <class> 元素中名稱爲 <student> 的節點。

圖表 1-11

1.4.4. 述語

述語(*predicate*)提供了絕對位置尋找節點的能力，這是因爲 XPath 的節點都會按照文件次序排列，所以我們可以取得特定次序的節點。

```
//class/student[1]
```

上面的結果將會取出 <class> 元素下的第一個 <student> 節點。然而述語也可搭配節點集合函數，例如要取出 <class> 元素下最後一個 <student> 元素的 XPath 表式示應該爲：

```
//class/student[last()]
```

1.4.5. 節點集合運算子

透過**節點集合運算子**(*node-set operator*)可以將多個位置步驟集合成一個位置路徑，或是對於兩組節點進行集合運算處理。

節點集合運算子	描述
管線（|）	對兩個節點集合進行集合處理。
反斜線（/）	用來分隔不同的位置步驟。
雙反斜線（//）	軸線 /descendant-or-self/node()/ 的縮寫

圖表 1-12

1.5. XPath 函數

在 XPath 的**函數庫**(*function library*)中，所有的函數的**傳回型別**(*return type*)可以分爲下列四種。

- **節點集合**。
- **布林值**。
- **數字**。
- **字串**。

這四種型別已經在位置路徑(§1.4)中闡述過了。

1.5.1. 節點集合函數

XPath 提供對**節點集合**(*node-set*)進行處理的一些函數，透過這些函數可以

得到節點集合的資訊。

節點集合函數	說明
last()	傳回節點集合中最後一個節點。
position()	傳回目前節點所在節點集合中的位置。
count(節點集合)	傳回目前節點集合中所有節點的數量。
id(“字串”)	傳回節點名稱符合字串者，例如 id(“foo”) 則會傳回 ID 爲 “foo” 之元素。
local-name(節點集合)	傳回節點集合中第一個節點的名稱
namespace-uri(節點集合)	傳回命名空間節點名稱
name(節點集合)	傳回在節點集合中第一個節點的名稱

圖表 1-13

1.5.2. 字串函數

字串函數	說明
concat(*string, string, string)**	傳回所有字串參數的接併。例如 concat(“abc”,”def”) 會傳回字串 “abcdef”。
contains(*string,string*)	傳回第一個字串是否包含在第二個字串參數中的布林結果。例如 contains("abc","a") 的結果會傳回 “TRUE”。
normalize-space(*string?*)	將傳入的字串去掉頭尾的**空白類字元**(*white space*)。例如 normalize-space(" abc ") 的結果爲 “abc”。
starts-with(*string,string*)	若第一個字串是以第二個字串參數作爲起始，則傳回 “TRUE” 否則傳回 “FALSE”。例如 starts-with("abc","a") 的結果爲 “TRUE”。
string(*object?*)	這個函數是將所傳入的參數轉換成字串後傳回。如果傳入的參數是節點或是節點集合的話，則會將節點內容以字串型別彙整後傳回；如果是數字的話，就回傳數字；布林值則是回傳 “true” 或是 “false”。例如 string(1+2*3) 的結果爲字串 “7”。
string-length(*string?*)	傳回字串長度。

substring(*string, number, number?***)**	第一個參數是來源字串，第二個參數是切割字串的起始位置，最後一個是選擇性的參數用來指定切割長度。例如 substring("abcdefgh", 5, 2) 的結果爲 “ef”。
substring-after(*string,string***)**	傳回第一個字串中出現第二個字串以後的子字串。例如 substring-after("2003/04/21", "/") 的結果爲 “04/21”。
substring-before(*string,string***)**	傳回第一個字串中出現第二個字串以前的子字串。例如 substring-before("2003/04/21", "/") 的結果爲 “2003”。
translate(*string, string, string***)**	第一個參數是傳入的字串，第二個參數是來源模式，第三個參數是目的模式。例如：translate(“AbC”,“ABC”, “abc”) 的結果爲 “abc”。 首先 translate() 函數會去比較第二個參數中的字元在第三個字串中變化的結果，然後將第一個參數中的字串符合條件的字元轉換成爲第三個參數中所指定的結果，由於目的結果的 “abc” 皆爲小寫，所以傳入的字串 “AbC” 皆轉換成小寫。詳細請參閱 XPath 規格書。

圖表 1-14

1.5.3. 布林函數

布林函數	說明
boolean(*object***)**	● 若傳入的數字若非零與 NaN 則傳回 “TRUE”。 ● 若是節點集合則當節點集合不爲空時，傳回 “TRUE” ● 若傳入字串，則當字串長度不爲零時，傳回 “TRUE” ● 其他的物件型別則視情況轉換之。
false()	傳回 “FALSE” 。
true()	傳回 “TRUE” 。
lang(*string***)**	若文件中有設定語系屬性，例如 <para xml:lang= “en” > 則 lang(“en”) 將傳回 “TRUE” 否則傳回 “FALSE”。
not(*boolean***)**	對傳入布林型別參數進行 not 布林運算。

圖表 1-15

1.5.4. 數字函數

數字函數	說明
ceiling(*number***)**	傳回由參數取得之最大正整數。例如 `ceiling(-123.22)` 的結果爲 “-123”。
floor(*number***)**	傳回由參數取得之最小正整數。例如 `floor(-123.22)` 的結果爲 “-124”。
number(*object?***)**	● 若傳入字串不可以被剖析成數字則傳爲 “NaN”。 ● 若傳入 “TRUE” 則轉換成 “1” ， “FALSE” 則轉換成 “0”。 ● 若傳入節點集合，首先會呼叫 `string()` 函數將其內容轉換成爲字串，然後在依據字串參數剖析辨識成數字，如果字串無法被剖析成數字則傳回 “NaN”。 ● 其他物件型別的轉換則視情況不同而有所改變。
round()	將傳入的參數進行四捨五入以取得整數值。
sum(*node-set***)**	對傳入的節點集合進行加總。

圖表 1-16

1.6. 位置路徑簡寫

XPath 提供了一系列的**簡寫**(*abbreviation*)，用來簡化位置路徑表式示。下面範例是一些常用的位置路徑簡寫。

para

位置路徑 `para` 表示由本文節點中篩選出所有 `para` 元素的子節點，這個簡寫等同於 `child::para`。舉例來說：若以根節點爲本文節點，要篩選出所有名稱爲 `class` 的元素節點，可以直接使用下達 `/class` ，若不使用簡寫則爲 `/child::class`。

*****

篩選所有的本文節點。這個簡寫等同於 `child::*`。

text()

篩選所有的本文節點中所有的文字節點。這個簡寫等同於 `child::text()`。

.

“.” 代表本文節點自己。這個簡寫等同於 self::node()。

..

“..” 代表本文節點的父節點。這個簡寫等同於 parent::node()。

@number

“@number” 將由本文節點中選出屬性名稱爲 number 之屬性節點。這個簡寫等同於 attribute::number。

@*

“@*” 將選出本文節點中所有的屬性節點。這個簡寫等同於 attribute::*。

..@number

“..” 代表本文節點的父節點，所以這個簡寫範例將會選出本文節點之父節點中所有屬性名稱爲 number 的屬性節點。這個簡寫等同於 parent::node()[attribute::number]。

para[1]

“para[1]” 是指在本文節點中第一個 para 元素。這個簡寫等同於 child::para[position()=1]。

.//para

“//” 是 descendant-or-self 的簡寫。這個範例將會篩選出所有本文節點本身或是其子孫中所有名稱爲 para 的元素節點。這個簡寫等同於 self::node()/descendant-or-self::node()/child::para

其他相關的簡寫範例請參閱 XPath 規格書。