Đi lại trong XML bằng XPATH

Chúng ta đã thấy cấu trúc và cú pháp của XML tương đối đơn giãn. XML cho ta một cách chuẩn để trao đổi tin tức giữa các computers. Bước tiếp theo là tìm hiểu cách nào một chương trình chế biến (process) một tài liệu XML

Dĩ nhiên để chế biến một XML chương trình ứng dụng phải có cách đi lại bên trong tài liệu để lấy ra values của các Elements hay Attributes. Do đó người ta thiết kế ra ngôn ngữ **XML Path language**, mà ta gọi tắt là **XPath**. XPath đóng một vai trò quan trọng trong công tác trao đổi dữ liệu giữa các computers hay giữa các chương trình ứng dụng vì nó cho phép ta lựa chọn hay sàng lọc ra những tin tức nào mình muốn để trao đổi hay hiển thị.

Nếu khi làm việc với cơ sở dữ liệu ta dùng SQL statement **Select .. from TableXYZ WHERE ...** để trích ra một số records từ một table, thì khi làm việc với XML, một table dữ liệu nho nhỏ, XPath cho ta những expressions về criteria (điều kiện) giống giống như clause **WHERE** trong SQL.

XPath là một chuẩn để process XML, cũng giống như SQL là một chuẩn để làm việc với cơ sở dữ liệu. Tiên phuông trong việc triển khai các chương trình áp dụng XPath là công tác của các công ty phần mềm lớn như Microsoft, Oracle, Sun, IBM, v.v. Sở dĩ ta cần có một chuẩn XPath là vì nó được áp dụng trong nhiều hoàn cảnh, nên cần phải có một lý thuyết rõ ràng, chính xác.

Lý thuyết về XPath hơi khô khan nhưng nó được áp dụng trong mọi kỹ thuật của gia đình XML. Cho nên bạn hãy kiên nhẫn nắm vững những điều căn bản về nó để khi nào gặp chỗ người ta dùng XPath thì mình nhận diện và hiểu được. So với võ thuật, thì XPath trong XML giống như Tấn pháp và cách thở. Tập luyện Tấn pháp thì mỏi chân, tập thở thì nhàm chán, nhưng không có hai thứ đó thì ra chiêu không có công lực, chưa đánh đã thua rồi.

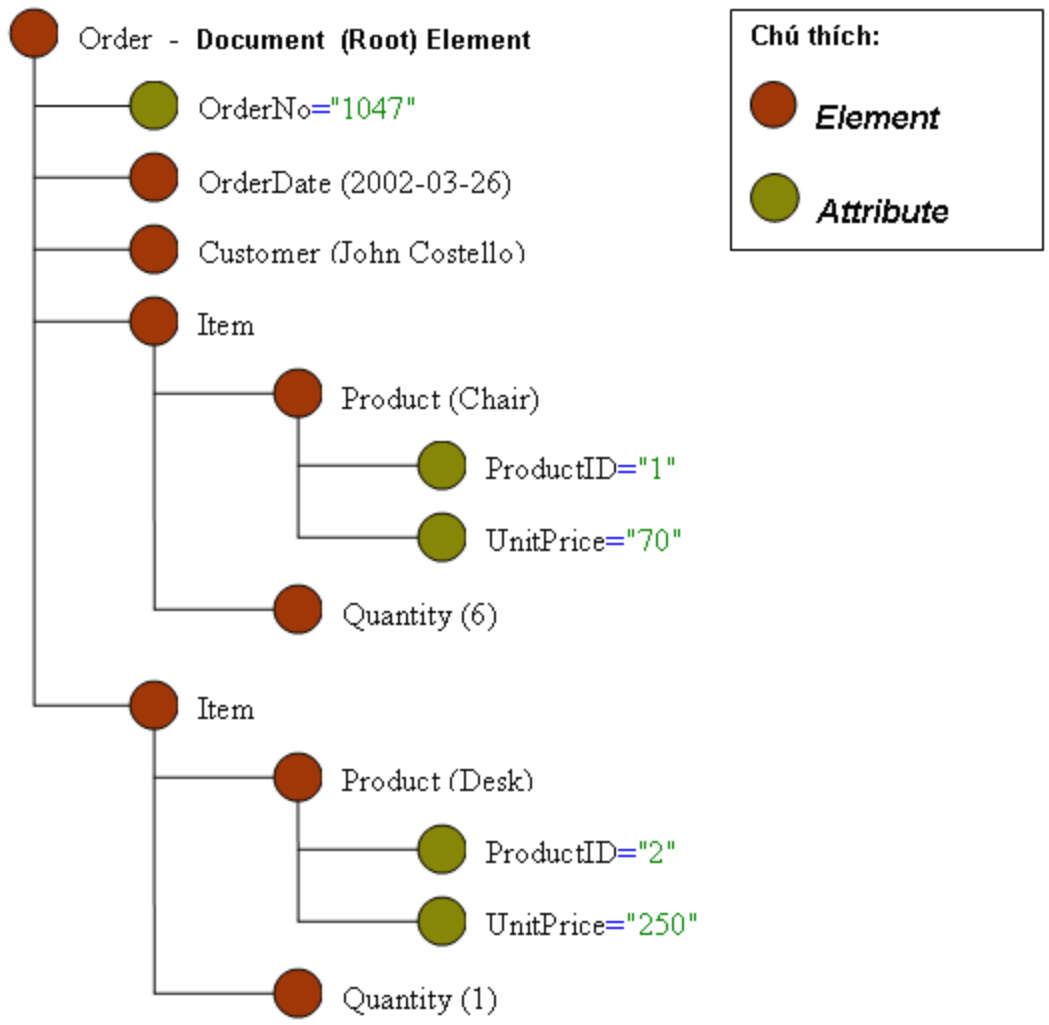
Ta sẽ chỉ học những thứ thường dùng trong XPath thôi, nếu bạn muốn có đầy đủ chi tiết về XPath thì có thể tham khão Specification của nó ở http://www.w3c.org/TR/xpath.

XML như một cây đối với XPath

XPath cho ta cú pháp để diễn tả cách đi lại trong XML. Ta coi một tài liệu XML như được đại diện bằng một tree (cây) có nhiều nodes. Mỗi Element hay Attribute là một node. Để minh họa ý niệm nầy, bạn hãy quan sát tài liệu đặt hàng (order) XML sau:

```
<?xml version="1.0"?>
<Order OrderNo="1047">
   <OrderDate>2002-03-26</OrderDate>
   <Customer>John Costello</Customer>
   <Item>
      <Product ProductID="1" UnitPrice="70">Chair</Product>
      <Quantity>6</Quantity>
   </Item>
   <Item>
      <Product ProductID="2" UnitPrice="250">Desk</Product>
      <Quantity>1</Quantity>
   </Item>
</Order>
```

Ta có thể biểu diễn XML trên bằng một Tree như dưới đây, trong đó node Element màu nâu, node Attribute màu xanh:



Chỉ định Location Path

Bạn có thể dùng XPath expression để chỉ định **Location Path** (lối đi đến vị trí) đến node nào hay trích ra (trả về) một hay nhiều nodes thỏa đúng điều kiện yêu cầu. XPath expression có thể là **tuyệt đối**, tức là lấy node gốc làm chuẩn hay **tương đối**, tức là khởi đầu từ node vừa mới được chọn. Node ấy được gọi là **context node** (node vai chính trong tình huống).

Có hai cách viết để diễn tả XPath Location, viết nguyên và viết tắt. Trong cả hai cách ta đều dùng dấu slash (**/**) để nói đến **Document Element**, tức là node gốc. Ta có thể đi lại trong các node của Tree giống giống như các node của Windows System Directory mà ta thấy trong Panel bên trái của Window Explorer. Ta cũng sẽ dùng những ký hiệu như slash **/**, một chấm **.** và hai chấm **..** của Windows System File Folder cho cách viết tắt trong XPath Location để đi xuống các nodes con, cháu, chỉ định context node, hay đi ngược lên các nodes tổ tiên.

Location Path tuyệt đối

Chúng ta hãy tìm vài location paths trong cái Tree của tài liệu XML về đặt hàng nói trên. Muốn chọn cái node của Element *Order* (nó cũng là Root Element) bằng cú pháp nguyên, ta sẽ dùng XPath expression sau đây:

```
  /child::Order
```

Dịch ra cú pháp tắt, expression nầy trở nên:

```
  /Order
```

Đi ra nhánh của Tree, ta sẽ tìm được node *Customer* bằng cách dùng XPath expression sau:

```
  /child::Order/child::Customer
```

Sau đây là XPath expression viết tắt tương đương:

```
  /Order/Customer
```

Nếu bạn muốn lấy ra một node Attribute, bạn phải nói rõ điều nầy bằng cách dùng từ chìa khóa (keyword) **attribute** trong cách viết nguyên hay dùng character **@** trong cú pháp tắt. Do đó để lấy Attribute *OrderNo* của Element *Order*, ta sẽ dùng XPath expression sau:

```
  /child::Order/attribute::OrderNo
```

Cú pháp tắt cho Attribute *OrderNo* là:

```
  /Order/@OrderNo
```

Để trích ra các nodes con cháu, tức là các nodes nhánh xa hơn, ta dùng keyword **descendant** trong cú pháp nguyên hay một double slash (//) trong cú pháp tắt. Thí dụ, để lấy ra các nodes Product trong tài liệu, bạn có thể dùng expression location path sau:

```
/child::Order/descendant::Product
```

Cú pháp tắt tương đương là:

```
/Order//Product
```

Bạn cũng có thể dùng wildcards (lá bài Joker) để nói đến những nodes mà tên của chúng không thành vấn đề. Thí dụ, dấu asterisk (*) wildcard chỉ định bất cứ node tên nào. Location path sau đây chọn tất cả các nodes con của Element *Order*:

```
/child::Order/child::*
```

Cú pháp tắt tương đương là:

```
/Order/*
```

Location Path tương đối

Nhiều khi XPath location paths là tương đối với context node, trong trường hợp ấy location path diễn tả cách lấy ra một node hay một số (set of) nodes tương đối với context node. Thí dụ như, nếu Element *Item* thứ nhất trong *order* là context node, thì location path tương đối để trích ra Element con *Quantity* là:

```
child::Quantity
```

Trong cú pháp tắt, location path tương đối là:

```
Quantity
```

Tương tự như vậy, để lấy ra Attribute *ProductID* của Element con *Product*, cái location path tương đối là:

```
child::Product/attribute::ProductID
```

Expression ấy dịch ra cú pháp tắt là:

```
Product/@ProductID
```

Để đi ngược lên phía trên của Tree, ta dùng keyword **parent** (cha). Dạng tắt tương đương của keyword nầy là hai dấu chấm (**..**). Thí dụ nếu context node là Element *OrderDate*, thì Attribute *OrderNo* có thể được lấy ra từ Element *Order* bằng cách dùng location path tương đối sau:

```
parent::Order/attribute::OrderNo
```

Để ý là cú pháp nầy chỉ trả về một trị số khi node cha tên *Order*. Nếu muốn lấy ra Attribute *OrderNo* từ node cha không cần biết nó tên gì bạn phải dùng expression sau:

```
parent::*/attribute::OrderNo
```

Viết theo kiểu tắt đơn giản hơn vì bạn không cần phải cung cấp tên của node cha. Bạn có thể nói đến node cha bằng cách dùng hai dấu chấm (**..**) như sau:

```
../@OrderNo
```

Ngoài ra, bạn có thể nói đến chính context node bằng cách dùng hoặc keyword **self** hoặc một dấu chấm (**.**). Điều nầy rất tiện trong vài trường hợp, nhất là khi bạn muốn biết current context node là node nào.

Dùng điều kiện trong Location Path

Bạn có thể giới hạn số nodes lấy về bằng cách gắn thêm điều kiện sàng lọc vào location path. Cái điều kiện giới hạn một hay nhiều nodes được tháp vào expression bên trong một cặp ngoặc vuông (**[]**). Thí dụ, để lấy ra mọi Element *Product* có Attribute *UnitPrice* lớn hơn 70, bạn có thể dùng XPath expression sau đây:

```
/child::Order/child::Item/child::Product[attribute::UnitPrice>70]
```

Trong cú pháp tắt, nó là:

```
/Order/Item/Product[@UnitPrice>70]
```

Trong expression của điều kiện bạn cũng có thể dùng Xpath tương đối , do đó trong expression điều kiện bạn có thể dùng bất cứ node nào trong thứ bậc. Thí dụ sau đây lấy về những nodes *Item* có Element con *Product* với Attibute *ProductID* trị số bằng 1:

```
/child::Order/child::Item[child::Product/attribute::ProductID=1]
```

Dịch ra cú pháp tắt, ta có:

```
/Order/Item[Product/@ProductID=1]
```

Collections

Cái bộ (Set of) Nodes do XPath trả về được gọi là **Collection**. Thông thường trong lập trình, từ "Collection" được dùng để nói đến một tập hợp các objects đồng loại. Ta có thể lần lượt đi qua (iterate through) các objects trong một Collection nhưng không được bảo đảm thứ tự của chúng, tức là gặp object nào trước hay object nào sau.

Trái lại, trong chuẩn XPath, khi một Collection được trả về bởi một XPath Query (hỏi), nó giữ nguyên thứ tự các Nodes và cấp bậc của chúng trong tài liệu XML. Tức là nếu XPath trả về một cành các nodes thì trừ những nodes không thỏa điều kiện, các node còn lại vẫn giữ đúng vị trí trên cành.

Vì các Attributes của một Element không có thứ tự, nên chúng có thể nằm lộn xộn trong một Collection.

Indexing trong một Collection

Một Collection của Nodes được xem như một Array. Muốn nói trực tiếp đến một Node trong Collection ta có thể dùng một index trong cặp ngoặc vuông. Node thứ nhất có Index là 1.

Cặp ngoặc vuông (**[]**) có precedence cao hơn (được tính trước) dấu slash(**/**) hay hai dấu slash (**//**). Dưới đây là hai thí dụ:

Expression	Ý nghĩa
author[1]	Element **author** đầu tiên.
author[firstname][3]	Element **author** thứ ba có một Element **firstname** con.

Mối liên hệ (Axes)

Một location path dùng một **Axis** để chỉ định mối liên hệ giữa các Nodes được chọn đối với context node. Sau đây là bảng liệt kê đầy đủ các axes:

Axes	Ý nghĩa
`ancestor::`	Tổ tiên của context node. Những tổ tiên của context node gồm có cha, ông nội, ông cố .v.v., do đó `ancestor::` axis luôn luôn kể cả root node trừ khi chính context node là root node.
`ancestor-or-self::`	Chính context node và tổ tiên của nó. Cái `ancestor-or-self::` axis luôn luôn kể cả root node.
`attribute::`	Các Attributes của context node. Nếu context node không phải là một Element thì chắc chắn axis sẽ trống rỗng.
`child::`	Con cái của context node. Một con là bất cứ node nào nằm ngay dưới context node trong tree. Tuy nhiên, Attribute hay Namespace nodes không được xem là con cái của context node.
`descendant::`	Con cháu của context node. Con cháu là con, cháu, chít, .v.v., do đó `descendant::` axis không bao giờ chứa Attribute hay Namespace nodes.

`following::`	Mọi nodes hiện ra sau context node trên tree, không kể con cháu, Attribute nodes, hay Namespace nodes.
`following-sibling::`	Mọi nodes em (nằm sau) context node. `following-sibling::` axis nói đến chỉ những Nodes con, của cùng một Node cha, nằm trên tree sau context node. Axis không kể các Nodes anh nằm trước context node. Nếu context node là Attribute hay Namespace thì `following-sibling::` axis sẽ trống rỗng.
`namespace::`	Những Namespace nodes của context node. Mỗi namespace có một namespace node trong scope (phạm vi hoạt động) của context node. Nếu context node không phải là một Element thì Axis sẽ trống rỗng.
`parent::`	Node cha của context node, nếu nó có cha. Node cha là node nằm ngay phía trên context node trên tree.
`preceding::`	Mọi nodes hiện ra trước context node trên tree, không kể các nodes tổ tiên, Attribute nodes, hay Namespace nodes. Một cách để nhận diện `preceding::` axis là mọi nodes đã kết thúc hoàn toàn trước khi context node bắt đầu.
`preceding-sibling::`	Mọi nodes anh (nằm trước) context node. `preceding-sibling::` axis nói đến chỉ những Nodes con, của cùng một Node cha, nằm trên tree trước context node. Nếu context node là Attribute hay Namespace thì `preceding-sibling::` axis sẽ trống rỗng.
`self::`	Là chính context node.

Sàng lọc (Filters)

Như ta đã thấy ở trên, để giới hạn chỉ lấy ra những Nodes thỏa đáng một điều kiện, ta gắn một **Filter** (sàng lọc) vào Collection. Filter ấy là một Clause giống giống **Clause WHERE** trong ngôn ngữ SQL của cơ sở dữ liệu.

Nếu một Collection nằm giữa một filter, nó sẽ cho kết quả TRUE nếu Collection trả về ít nhất một Node và FALSE nếu Collection trống rỗng (empty). Thí dụ expression **author/degree** có nghĩa rằng hàm ***biến đổi Collection ra trị số Boolean*** sẽ có giá trị TRUE nếu hiện hữa một Element **author** có Element con tên **degree**.

Filters luôn luôn được tính theo context của nó. Nói một cách khác, cái expression **book[author]** có nghĩa là cho mỗi Element **book** tìm thấy, nó sẽ được thử xem có chứa một Element con tên **author** không. Tương tự như vậy, **book[author = 'Brown']** có nghĩa rằng cho mỗi Element **book** tìm thấy, nó sẽ được thử xem có chứa một Element con tên **author** với trị số bằng **Brown** không.

Ta có thể dùng dấu chấm (**.**) để khám current context node. Thí dụ như, **book[. = 'Dreams']** có nghĩa rằng cho mỗi Element book tìm thấy trong current context, nó sẽ được thử xem có trị số bằng **Dreams** không. Dưới đây là một ít thí dụ:

Expression	Ý nghĩa
book[excerpt]	Mọi Element **book** có chứa ít nhất một Element **excerpt**.
book[excerpt]/title	Mọi Element **title** nằm trong những Element **book** có chứa ít nhất một Element **excerpt**.
book[excerpt]/author[degree]	Mọi Element **author** có chứa ít nhất một Element **degree** và nằm trong những Elements **book** có chứa ít nhất một Element **excerpt**.
book[author/degree]	Mọi Element **book** có chứa ít nhất một Element **author** với ít nhất một Element **degree** con.
book[excerpt][title]	Mọi Element **book** có chứa ít nhất một Element **excerpt** và ít nhất một Element **title**.

So sánh

Để so sánh hai objects trong XPath ta dùng dấu (**=**) cho **bằng nhau** và (**!=**) cho **không bằng nhau**. Mọi Element và Attributes là **string**, nhưng được Typecast (xem như) những con số khi đem ra so sánh.

Expression	Ý nghĩa
author[lastname = "Smith"]	Mọi Element **author** có chứa ít nhất một Element **lastname** với trị số bằng **Smith**.
author[lastname[1] = "Smith"]	Mọi Element **author** có Element **lastname** con đầu tiên với trị số bằng **Smith**.
author/degree[@from != "Harvard"]	Mọi Element **degree**, là con một Element **author**, và có một Attribute **from** với trị số không phải là **"Harvard"**.
author[lastname = /editor/lastname]	Mọi Element **author** có chứa một Element **lastname** bằng với Element **lastname** là con của root Element **editor**.
author[. = "John Hamilton"]	Mọi Element **author** có trị số string là **John Hamilton**.

Operator Union | (họp lại)

Ngôn ngữ Xpath hỗ trợ Operator Union, giống như Logical **OR (hoặc là)**. Dưới đây là vài thí dụ:

Expression	Ý nghĩa
firstname | lastname	Mọi Element **firstname** và **lastname** trong current context.
(bookstore/book | bookstore/magazine)	Mọi Element **book** hay **magazine** là con một Element **bookstore**.
book | book/author	Mọi Element **book** hay Element **author** là con những Elements **book**.
(book | magazine)/price	Mọi Element **price** là con của Element **book** hay Element **magazine**.

Thử loại Node (Node Type Tests)

Để chọn những loại Node khác hơn là Element node, ta dùng **Node-Type Test**. Mục đích của việc dùng Node-Type test là để chỉ định sự lựa chọn khác thường. Thí dụ như, **descendant::text()** cho ta mọi text nodes là con cháu của context node, dù rằng loại node chính của con cháu context node là Element. Có 4 loại Node-Type tests như liệt kê dưới đây.

Node type	Trả về	Thí dụ
comment()	mọi comment node.	**following::comment()** chọn mọi comment nodes hiện ra sau context node.
node()	mọi node.	**preceding::node()** chọn mọi nodes hiện ra trước context node.
processing-instruction()	mọi processing instruction node.	**self::processing instruction()** chọn mọi processing instruction nodes trong context node.
text()	mọi text node.	**child::text()** chọn mọi text nodes là con của the context node.

Thử Node nhắm vào loại Processing Instruction

Một node test có thể chọn processing instruction thuộc loại nào, tức là chọn **mục tiêu (target)**. Cú pháp của một loại test như thế là:

```
processing-instruction("target")
```

Thí dụ node test sau đây trả về mọi processing instruction nodes có nhắc đến một XSL stylesheet trong tài liệu:

```
/child::processing-instruction("xml-stylesheet")
```

Thêm một số thí dụ Location Path

Expression	Ý nghĩa
./author	Mọi Element **author** trong current context. Expresion nầy tương đương với expression trong hàng kế.
author	Mọi Element **author** trong current context.

/bookstore	Document (Root) Element tên **bookstore** của tài liệu nầy.
//author	Mọi Element **author** trong tài liệu.
book[/bookstore/@specialty = @style]	Mọi Element **book** có Attribute **style** với value bằng value của Attribute **specialty** của Document Element **bookstore** của tài liệu.
author/firstname	Mọi Element **firstname** con của các Elements **author**.
bookstore//title	Mọi Element **title** một hay nhiều bậc thấp hơn, tức là con cháu của, Element **bookstore**. Lưu ý là expression nầy khác với expression trong hàng kế.
bookstore/*/title	Mọi Element **title** cháu của các **bookstore**.
bookstore//book/excerpt//emph	Mọi Element **emph** bất cứ nơi nào dưới **excerpt** là con của những elements **book** , bất cứ nơi nào dưới element **bookstore**.
.//title	Mọi Element **title** một hay nhiều bậc thấp hơn current context node.
author/*	Mọi Element là con của các elements con **author**.
book/*/lastname	Mọi Element **lastname** là cháu của các elements con **book**.
/	Mọi Element cháu của current context node.
*[@specialty]	Mọi Element con có Attribute **specialty**.
@style	Attribute **style** của current context node.
price/@exchange	Attribute **exchange** của những Elements **price** trong current context, tức là những Elements **price** của current context node.
price/@exchange/total	Trả về một node set trống rỗng, vì Attributes không có Element con. Expression nầy được chấp nhận trong văn phạm của XML Path Language, nhưng không thật sự hợp lệ.
book[@style]	Mọi Element **book** có Attribute **style** trong current context node. Lưu ý phần nằm trong ngoặc vuông là điều kiện của Element book
book/@style	Attribute **style** của mọi Element **book**trong current context node. Ở đây không có điều kiện như hàng trên. Ta nói đến Attribute hay Element nằm bên phải nhất.
@*	Mọi Attributes của current context node.
author[1]	Element **author** thứ nhất trong current context node.
author[firstname][3]	Element **author** thứ ba có một Element con **firstname**.
my:book	Element **book** từ namespace **my**.
my:*	Mọi Element trong namespace **my**.