CĂN BẢN VỀ XML

MỤC LỤC

Chương mở đầu

Trong thời đại Công nghệ Thông tin hiện nay XML (**eXtensible Markup Language**) chiếm vị trí số một và rất quan trọng trong việc chuyển tải, trao đổi dữ liệu và liên lạc giữa các ứng dụng. Điều này càng được khẳn định khi trong các hệ điều hành từ WindowsXP trở đi, bên trong nó chứa đầy XML. Hơn nữa khi bộ .Net ra đời thì càng làm cho XML trở nên thịnh hành.

Sử dụng kỹ thuật XML không chỉ có tập đoàn Microsoft mà ngay cả Sun, IBM, Oracles điều hỗ trợ XML và dùng nó trong các ứng dụng.

1 XML là gì?

XML là ngôn ngữ xây dựng cấu trúc tài liệu văn bản, dựa theo chuẩn SGML (Standard Generalized Markup Language: siêu ngôn ngữ có khả năng sinh ngôn ngữ khác). SGML được phát triển cho việc định cấu trúc và nội dung tài liệu điện tử, do tổ chức ISO (International Organization for Standards) chuẩn hoá năm 1986.

SGML là do IBM đưa ra, song không thể không kể đến những đóng góp của các công ty khác. XML được W3C (World Wide Web Consortium: tổ chức độc lập định ra tiêu chuẩn cho trình duyệt Web, máy chủ và ngôn ngữ) phát triển, nhưng đặc tả XML lại do Netscape, Microsoft và các thành viên của dự án Text Encoding Initiative (TEI) xây dựng. Tổ chức W3C XML Special Interest Group có đại diện từ hơn 100 công ty cùng nhiều chuyên gia được mời khác.

Lý do ra đời của XML vì SGML rất rắc rối, và HTML có nhiều giới hạn nên năm 1996 tổ chức W3C thiết kế XML. XML version 1.0 được định nghĩa trong hồ sơ **February 1998 W3C Recommendation.**

Điểm quan trọng của kỹ thuật XML là nó không thuộc riêng về một công ty nào, nó là một sản phẩm mà trí tuệ của nó thuộc về cả thế giới, nó là một tiêu chuẩn được mọi người công nhận vì được soạn ra bởi World Wide Web Consortium - W3C (một ban soạn thảo với sự hiện diện của tất cả các chuyên gia Tin học) và những ý kiến đóng góp bằng cách trao đổi qua Email.

Bản thân của XML rất là đơn giản, nhưng các công cụ chuẩn được định ra để làm việc với XML như **Document Object Model - DOM, XPath, XSL**, v.v.. thì rất hữu hiệu, và chính các chuẩn này được phát triển không ngừng.

XML cũng giống như HTML đều là ngôn ngữ đánh dấu, nhưng điều cần nói ở đây là sự ra đời của XML để khắc phục cho một số yếu kém của HTML. HTML và XML đều sử dụng các tag nhưng các tag của HTML là một bộ dữ liệu tag được xây dựng và định nghĩa trước, tức là người lập trình phải tuân thủ theo các thẻ đã định nghĩa của HTML, hiện HTML có khoản hơn 400 tag, để nhớ hết 400 tag này cũng không có gì khó khăn đối với người lập trình web chuyên nghiệp nhưng thật khó đối với những người không chuyên. Hơn nữa các tag của HTML không nói lên được mô tả dữ liệu trong đó. Nhưng đối với XML thì hoàn toàn khác bởi vì tag trong XML là do người lập trình định nghĩa và mỗi tag là một mô tả dữ liệu mà người lập trình muốn truyền đạt.

2 Các nội dung sẽ trình bày

Khi XML ra đời thì có hàng loạt các ngôn ngữ chuẩn được đưa ra để làm việc với XML, nhưng trong tài liệu này tôi chỉ xin được trình bày các phần chính sau:

1. Ngôn ngữ XML(eXtensible Markup Language)

Ngôn ngữ xây dựng cấu trúc tài liệu văn bản, dựa theo chuẩn SGML (Standard Generalized Markup Language

2. Ngôn ngữ DTD (Document Type Definition)

   Ngôn ngữ dùng để định nghĩa kiểu dữ liệu cho các phần tử trong tài liệu XML

3. Ngôn ngữ Xpath (XML Path Language)

   Ngôn ngữ dung để duyệt tài liệu XML

4. XSL (eXtensible style sheet)

   Ngôn ngữ dùng để chuyển đổi tài liệu XML thành một định dạng khác

5. Ngôn ngữ Xlink(XML Link Language) và Xpointer(XML Pointer Language)

   Ngôn ngữ lien kết và định vị tài liệu

Chương 1

XML (eXtensible Markup Language)

1 Phần lý thuyết

1.1 Các quy tắc cần lưu ý

Để viết được một trang XML cũng rất đơn giãn, chúng ta chỉ cần tuân thủ những quy tắc sau:

- Phải có một Phần tử gốc duy nhất, nó chứa tất cả các Phần tử khác trong tài liệu.

```
<Catalog>
  <Product ProductID="1">Chair</Product>
  <Product ProductID="2">Desk</Product>
</Catalog>
```

- Mỗi Tag mở phải có một Tag đóng giống như nó.

```
1   <Order>
2     <OrderDate>2002-6-14</OrderDate>
3     <Customer>Helen Mooney</Customer>
4     <Item>
5       <ProductID>2</ProductID>
6       <Quantity>1</Quantity>
7     <Item>
8       <ProductID>4</ProductID>
9       <Quantity>3</Quantity>
10    </Item>
11  </Order>
```

Ví dụ trên không thỏa qui tắc 2 vì thiếu Tag đóng </Item>, ta cần thêm Tag đóng </item> vào sau dòng thứ 6

- Trong một cặp Tag đóng và mở phải được đánh vần như nhau, có nghĩa là các cặp ký tự của cặp Tag đóng mở này phải giống nhau hoàn toàn.

```
1   <Order>
2     <OrderDate>2001-01-01</Orderdate>
3     <Customer>Graeme Malcolm</Customer>
4   </Order>
```

Ví dụ này không thỏa quy tắc 3 vì Tag mở <OrderDate> và Tag đóng </Orderdate> đánh vần không giống nhau, ta phải sửa Tag đóng </Orderdate> thành </OrderDate> hoặc sửa thẻ <OrderDate> thành <Orderdate>

- Mỗi Phần tử con phải nằm trọn bên trong Phần tử cha của nó.

```
1    <Catalog>
2      <Category CategoryName="Beverages">
3        <Product ProductID="1">
4          Coca-Cola
5        </Category>
6      </Product>
7  </Catalog>
```

Ví dụ này không thỏa quy tắc 4 vì Tag đóng </Product> đặt không đúng vị trí, ta cần đổi vị trí của dòng 5 cho dòng 6

- Giá trị của thuộc tính phải được đặt trong cặp dấu nháy kép hoặc cặp dấu nháy đơn.

```
1    <Catalog>
2      <Product ProductID=1>Chair</Product>
3      <Product ProductID='2">Desk</Product>
4  </Catalog>
```

Ví dụ này không đúng quy tắc 5 vì giá trị của thuộc tính ProductID ở dòng 3 đặt trong cặp dấu nháy không đúng, ta cần sửa lại ProductID='2" thành ProductID=”2" hoặc ProductID='2’

1.2 Chỉ thị xử lý (Processing Instructions) và lời chú thích (Comments)

Chúng ta thường thấy dòng lệnh

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

nằm ở đầu file XML. Đây chính là chỉ thị xử lý, chỉ thị xử lý được đặt trong cặp Tag <? và ?>. Nó cho biết phiên bản đặc tả XML mà bộ phân tích cần làm theo, ngoài ra nó cho phép người lập trình cho biết dữ liệu trong XML dùng encoding nào, còn thuộc tính standalone sẽ cho biết tài liệu XML có cần đến một tài liệu khác không (có hai giá trị cho thuộc tính này đó là “yes” nếu không cần đến một tài liệu khác và “no” nếu cần).

```
<?xml version="1.0" encoding="utf-8"  standalone="yes"?>
        <Order>
          <OrderDate>2002-6-14</OrderDate>
          <Customer>Helen Mooney</Customer>
          <Item>
            <ProductID>1</ProductID>
            <Quantity>2</Quantity>
          </Item>
          <Item>
            <ProductID>4</ProductID>
            <Quantity>1</Quantity>
          </Item>
</Order>
```

Trong mỗi tài liệu XML có thể có hoặc không có phần này

Để file XML trở nên dễ hiểu và dễ chỉnh sửa sau này thì các dòng chú thích là không thể thiếu, các dòng chú thích được đặt trong cặp tags <!-- và -->.

1.3 Không gian tên (namespace).

XML cho phép chúng ta tự do định nghĩa các thẻ, như nó cho chúng ta dùng cùng một tên nhưng lại nói đến nhiều loại dữ liệu khác nhau trong cùng một tài liệu XML. Xem ví dụ sau:

```
<?xml version="1.0"?>
<BookOrder OrderNo="1234">
  <OrderDate>2001-01-01</OrderDate>
  <Customer>
    <Title>Mr.</Title>
    <FirstName>Graeme</FirstName>
    <LastName>Malcolm</LastName>
  </Customer>
  <Book>
    <Title>Treasure Island</Title>
    <Author>Robert Louis Stevenson</Author>
  </Book>

</BookOrder>
```

Ta thấy trong ví dụ trên có phần tử Title nói đến hai loại dữ liệu khác nhau, một nói về tên tác giả một nói về tiêu đề sách, điều này làm cho ta nhầm lẫn giữa hai loại dữ liệu. Hơn thế nữa nếu tài liệu của chúng ta được sử dụng chỉ cho một mục đích riêng rẻ thì không có vấn đề gì nhưng khi tài liệu của chúng ta kết hợp với một tài liệu khác khác thì tài liệu kết hợp này sẽ có vấn đề vì chúng ta đâu chắc chắn rằng tài liệu khác mà chúng ta muốn kết hợp không có sử dụng thẻ trùng với thẻ của chúng ta định nghĩa hay không.

Ví dụ như khi chúng ta tích hợp tài liệu XML của chúng ta với ứng dụng khác như VML hay MathML mà chẳng may giữa các tài liệu này có cùng định nghĩa thẻ NAME chẳng hạn. Lúc này trình phân tích sẽ không biết nên hiểu thẻ NAME của tài liệu của bạn hay của VML hay của MathML.

Vì vậy chúng ta cần phải khai báo không gian tên để khắc phục điều này.

1.3.1 Khai báo không gian tên (namespace)

Để khai báo một không gian tên ta chỉ cần đưa thêm thuộc tính **xmlns:prefix** vào bên trong phần tử gốc, **prefix** là tên của không gian tên, mỗi không gian tên cần mang một định danh duy nhất. Một không gian tên có thể là một địa chỉ internet hoặc một địa chỉ nào đó miễn là địa chỉ này phải duy nhất. Ví dụ sau đây sẽ tạo ra một không gian tên **hs** và áp dụng cho tất cả các phần tử con:

```
<?xml version="1.0"?>
<BookOrder  xmlns:hs="http://www.northwindtraders.com/customer">
  <hs:Customer >
    <hs:Title>Mr.</Title>
    <hs:FirstName>Graeme</FirstName>
    <hs:LastName>Malcolm</LastName>
  </hs:Customer>

</BookOrder>
```

1.3.2 Không gian tên mặc định (namespace default)

Nếu tài liệu của chúng ta các phần tử chỉ sử dụng duy nhất một không gian tên thì chúng ta có thể khai báo không gian tên mặc định cho các phần tử con của một phần tử cha bằng cách chỉ ghi thuộc tính **xmlns** và bỏ đi **prefix**

```
<?xml version="1.0"?>
<BookOrder  xmlns ="http://www.northwindtraders.com/customer">
  < Customer >
    < Title>Mr.</Title>
    < FirstName>Graeme</FirstName>
    < LastName>Malcolm</LastName>
  </ Customer>
</BookOrder>
```

Chúng ta xem tiếp ví dụ sau:

```
<?xml version="1.0"?>
<BookOrder >
  <OrderDate>2001-01-01</OrderDate>
  <Customer xmlns="http://www.northwindtraders.com/customer">
    <Title>Mr.</Title>
    <FirstName>Graeme</FirstName>
    <LastName>Malcolm</LastName>
  </Customer>
  <Book xmlns="http://www.northwindtraders.com/book">
    <Title>Treasure Island</Title>
    <Author>Robert Louis Stevenson</Author>
  </Book>
</BookOrder>
```

Ví dụ trên chúng ta thấy có hai không gian tên mặc định, một cho phần tử **Customer** và một cho phần tử **Book**. Nhưng điều đáng nói ở đây là nếu như trong tài liệu có nhiều **Customer** và nhiều **Book** thì chúng ta không thể viết đi viết lại không gian tên mãi được, rất mất thời gian.

Cách giải quyết tốt nhất là ta khai báo các không gian tên này ngay ở đầu tài liệu và mỗi không gian tên được phân biệt bởi các định danh.

```
<?xml version="1.0"?>
<BookOrder  xmlns="http://www.northwindtraders.com/order"
                xmlns:cus="http://www.northwindtraders.com/customer"
                xmlns:bok ="http://www.northwindtraders.com/book"    >
<OrderDate>2001-01-01</OrderDate>
  <cus:Customer >
    <cus:Title>Mr.</cus:Title>
    <cus:FirstName>Graeme</cus:FirstName>
    <cus:LastName>Malcolm</cus:LastName>
  </cus:Customer>
  <bok:Book >
    <bok:Title>Treasure Island</bok:Title>
    <bok:Author>Robert Louis Stevenson</bok:Author>
  </bok:Book>
</BookOrder>
```

Ví dụ trên dùng 3 không gian tên, một không gian tên mặc định và hai không gian tên có định danh là **cus** và **bok**. Trong ví dụ trên những phần tử không có định danh của không gian tên đi trước thì được hiểu là sử dụng không gian

tên mặc định http://www.northwindtraders.com/order, như phần tử **<OrderDate>2001-01-01</OrderDate>**

1.4 CDATA

Đoạn dữ liệu của CDATA là đoạn dữ liệu nằm giữa <![CDATA [và]]>. Những đoạn dữ liệu nằm trong CDATA khi đi qua trình phân tích sẽ được giữ nguyên như ban đầu, tức là khi gặp CDATA thì trình phân tích sẽ bỏ qua. Điều này rất cần thiết khi chúng ta viết những đoạn mã script trong tài liệu.

```
<script language=”javascript”>
<![CDATA[
function mag(){
   alert(“This is CDATA! ”);
}
]]
</script>
```

1.5 Thực thể định nghĩa sẵn trong XML

Trong ngôn ngữ định dạng XML có sử dụng một số ký tự định dạng đặc biệt: <, >, ‘, “, &. Vì vậy để giúp cho chúng ta thể hiện tài liệu đúng theo nguyên mẫu bằng cách định nghĩa 5 thực thể này như sau:

Thực thể	Mô tả
&apos;	Tương đương với dấu nháy đơn (‘)
&amp;	Tương đương với dấu &
&gt;	Tương đương với dấu >
&lt;	Tương đương với dấu <
&quot;	Tương đương với dấu nháy kép (“)

Ví dụ:

```
<?xml version="1.0"?>
<LINK-TO>
   &lt; a href=&quot; index.txt &quot; &gt; OPEN FILE INDEX.TXT &lt; /a &gt;
</LINK-TO>
```

(< a href=" index.txt " > OPEN FILE INDEX.TXT < /a > tương đương với OPEN FILE INDEX.TXT)

Để hiểu rõ hơn về thực thể là gì, chúng ta sẽ tìm hiểu trong chương 2.

2 Phần ví dụ

Chương 2
DTD (Document Type Definition)

1 DTD là gì?

DTD (Document Type Definition) là kiểu tài liệu dùng để định nghĩa kiểu dữ liệu cho các phần tử trong tài liệu XML. Khi chúng ta định nghĩa các phần tử trong XML là tùy thích, miễn sao cho nó hợp quy tắc của tài liệu XML. Tuy nhiên để tường minh hơn thì ta nên định nghĩa kiểu dữ liệu cho từng phần tử trong tài liệu XML.

Trong chương trước chúng ta đã học cách viết một tài liệu hợp khuôn dạng. Tuy nhiên một tài liệu XML được xem là hợp khuôn dạng và có giá trị khi toàn bộ các phần tử trong tài liệu được được định nghĩa kiểu dữ liệu mà nó chứa. Với cách định nghĩa kiểu tư liệu (DTD) khi chúng ta đọc một tài liệu XML nào thì chỉ cần đọc phần DTD thì chúng ta sẽ biết được cấu trúc của tài liệu XML.

Trước khi đi vào phần chi tiết về cách tạo một tài liệu DTD, chúng ta hãy xem ví dụ sau:

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to      (#PCDATA)>
  <!ELEMENT from    (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body    (#PCDATA)>
]>
```

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

Xem ví dụ trên chúng ta thấy phần được bôi đen là phần DTD dùng để mô tả kiểu dữ liệu của tài liệu XML, phần có màu nhạt là các phần tử của tài liệu XML.

Phần DTD trong ví dụ này được hiểu như sau: Tài liệu XML có một phần tử gốc tên là note, phần tử gốc này có 3 phần tử con là from, heading, body và 3 phần tử con này có kiểu dữ liệu text.

2 Định nghĩa một tài liệu DTD

Để viết một tài liệu DTD cũng rất dễ, chỉ cần chúng ta tuân thủ đúng một số quy tắc của W3C là được. Đầu tiên chúng ta hãy tìm hiểu về các phần tử (emlement), thuộc tính, thực thể của DTD.

2.1 Phần tử <!DOCTYPE>

Phần tử này có chức năng dùng để khai báo bắt đầu định nghĩa kiểu tư liệu DTD.

Định nghĩa kiểu tư liệu có 2 dạng, đó là DTD tham chiếu nội và DTD tham chiếu ngoại. DTD tham chiếu nội là DTD được định nghĩa ngay trong tài liệu XML còn DTD tham chiếu ngoại là DTD được định nghĩa bên ngoài tài liệu XML. Bây giờ chúng ta sẽ tìm hiểu từng cú pháp một.

2.1.1 Định nghĩa DTD tham chiếu nội

Để bắt đầu định nghĩa kiểu tư liệu DTD tham chiếu nội chúng ta dùng cú pháp sau:

<!DOCTYPE root-element [DTD]

Trong đó root-element là phần tử gốc của tài liệu XML, DTD là các định nghĩa cho các phần tử trong tài liệu XML.

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note body>
  <!ELEMENT body    (#PCDATA)>
]>
```

```
 <note>
    <body>Don't forget me this weekend</body>
</note>
```

2.1.2 Định nghĩa DTD tham chiếu ngoại

Sử dụng định nghĩa DTD tham chiếu ngoại sẽ làm cho các ứng dụng XML của chúng ta trở nên dễ dàng chia sẽ và dùng chung với các ứng dụng khác. Có hai cách để chỉ định một DTD tham chiếu ngoại: Tham chiếu ngoại riêng và tham chiếu ngoại chung.

Những định nghĩa DTD tham chiếu ngoại riêng được sử dụng cho một nhóm người mang tính cá nhân, chúng không được dùng cho mục đích chung rộng lớn, mục đích phân phối. Còn những định nghĩa DTD tham chiếu ngoại chung sẽ mang tính cộng đồng hơn.

- Để định nghĩa một DTD tham chiếu ngoại riêng chúng ta dùng cú pháp sau:

  <!DOCTYPE root-element SYSTEM “filename”>

Trong đó root-element là tên của phần tử gốc trong tài liệu XML, filename là tên file định nghĩa kiểu tư liệu DTD

Ví dụ:

```
<?xml version="1.0"?>
 <!DOCTYPE note SYSTEM "note.dtd">
  <note>
     <to>Tove</to>
      <from>Jani</from>
      <heading>Reminder</heading>
```

```
    <body>Don't forget me this weekend!</body>
</note>
```

File note.dtd với nội dung như sau:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Địa chỉ chứa file DTD có thể một URL/URI.

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "http://www.w3schools.com/dtd/note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- Để định nghĩa một DTD tham chiếu ngoại chung chúng ta dùng cú pháp sau:

  <!DOCTYPE root-element PUBLIC “FPI” “URL”>

  Trong đó FPI (Formal Public Identifier) là một định danh chung hình thức, chúng ta cần tuân theo một số quy tắc áp dụng cho FPI sau:

  - Trường đầu tiên của một FPI là xác định kết nối của DTD đến chuẩn hình thức. Đối với các DTD chúng ta tự định nghĩa thì trường này là một dấu chấm. Đối với các chuẩn hình thức trường này sẽ tự tham chiếu đến chuẩn của nó.
  - Trường thứ hai là tên nhóm hay tên người chịu trách nhiệm bảo trì và nâng cấp các định nghĩa DTD và tên này phải mang tính duy nhất.
  - Trường thứ ba chỉ định kiểu của tài liệu được mô tả, thường thì trường này kèm theo một số định danh duy nhất nào đó (chẳn hạn như version 1.0).
  - Trương thứ ba chỉ định ngôn ngữ mà bạn định nghĩa DTD (ví dụ như ngôn ngữ Tiếng Anh - EN)
  - Mỗi trường của FPI cách nhau bởi dấu //

  Ví dụ

```
<?xml version="1.0"?>
<!DOCTYPE note PUBLIC “.//w3schools//note XML version 1.0//EN”
"http://www.w3schools.com/dtd/note.dtd">
<note>
```

```
    <to>Tove</to>
     <from>Jani</from>
     <heading>Reminder</heading>
   <body>Don't forget me this weekend!</body>
</note>
```

Còn URL là địa chỉ của file DTD.

2.2 Phần tử <!ELEMENT>

Phần tử <!ELEMENT> dùng để định nghĩa kiểu dữ liệu cho một phần tử của một tài liệu XML. Chúng ta sử dụng theo cú pháp sau:

<!ELEMENT element-name content_model>

Trong đó:

- **element_name** là tên của phần tử mà ta muốn định nghĩa
- **content_model** là kiểu của phần tử này, có thể là EMPTY, ANY, #PCDATA, các phần tử con hay trộn lẫn nhiều thành phần

Bây giờ chúng ta tìm hiểu chi tiết hơn.

- Định nghĩa một phần tử rỗng

  <!ELEMENT element_name EMPTY>

  Ví dụ: <!EMLEMENT note EMPTY>

- Định nghĩa một phần tử có chứa nhiều kiểu dữ liệu

  <!ELEMENT element_name ANY>

  Ví dụ: <!ELEMENT note ANY>

- Định nghĩa một phần tử có kiểu văn bản

  <!ELEMENT element_name (#PCDATA)>

  Ví dụ: <!ELEMENT note (#PCDATA)>

- Định nghĩa một phần tử có chứa một phần tử con

  <!ELEMENT element_name (child_element)>

  Ví dụ: <!ELEMENT note (to)>

- Định nghĩa một phần tử có chứa nhiều hơn một phần tử con, cách thứ nhất là chúng ta có thể liệt kê tất cả các phần tử con đó và mỗi phần tử con cách nhau bởi dấu phẩy.

  Ví dụ để khai báo phần tử note có 4 phần tử con là to, from, heading, body chúng ta viết như sau:

  <!ELEMENT note (to, from, heading, body)>

  Tất nhiên với cách viết như thế này thì không tối ưu, chúng ta có thể dùng cách viết thứ hai cho những phần tử có nhiều phần tử con bằng cách dùng ký tự đại diện.

  Dưới đây là một số nguyên tắc sử dụng ký tự đại diện:

  Giả sử chúng ta có phần tử ROOT, phần tử này có hai phần tử con là LIMB_A và LIMB_B, chúng ta có một số định nghĩa sau:

- <!ELEMENT ROOT (LIMB_A*)>

  Phần tử ROOT không có hoặc có nhiều phần tử LIMB_A
- <!ELEMENT ROOT (LIMB_A+)>

  Phần tử ROOT có một hoặc nhiều phần tử con LIMB_A
- <!ELEMENT ROOT (LIMB_A?)>

  Phần tử ROOT không có hoặc có một phần tử con LIMB_A
- <!ELEMENT ROOT (LIMB_A, LIMB_B)>

  Phần tử ROOT có 2 phần tử con, đầu tiên là phần tử LIMB_A tiếp đến là LIMB_B
- <!ELEMENT ROOT (LIMB_A | LIMB_B)>

  Phần tử ROOT có một phần tử con hoặc là LIMB_A hoặc là LIMB_B
- Định nghĩa một phần tử có chứa phần tử con hoặc chứa dữ liệu văn bản

<!ELEMENT LIMB_A (LIMB_A1| #PCDATA)>

2.3 Phần tử <!ATTLIST>

Phần tử <!ATTLIST> dùng để định nghĩa kiểu tư liệu của các thuộc tính cho một phần tử trong tài liệu XML. Chúng ta dùng cú pháp sau:

<!ATTLIST element-name attribute-name attribute-type default-value>

Trong đó:

- **element-name** là tên của một phần tử cần định nghĩa thuộc tính
- **attribute-name** là tên thuộc tính cần định nghĩa
- **attribute-type** kiểu của thuộc tính. Có thể nhận một tong các giá trị sau:

Kiểu	Mô tả
CDATA	Cho biết thuộc tính này chỉ có thể chứa kiểu dữ liệu ký tự
(en1|en2|..)	Danh sách các giá trị mà thuộc tính có thể được gán
ID	Cho biết thuộc tính này là một ID, tức là các giá trị của thuộc tính này không được trùng nhau và phải bắt đầu bởi một chữ cái
IDREF	Cho biết giá trị của thuộc tính này phải là một trong các giá trị của thuộc tính ID của các phần tử khác
IDREFS	Cho biết giá trị của thuộc tính này phải là các giá trị của các thuộc tính có kiểu ID
NMTOKEN	Cho biết giá trị của thuộc tính là các giá trị hợp với quy tắc đặt tên của phần tử của tài liệu XML
NMTOKENS	Cũng giống như NMTOKEN nhưng nó cho phép chứa nhiều NMTOKEN
ENTITY	Cho biết thuộc tính này nhận giá trị là một tên tham chiếu của thực thể

ENTITIES	Cho biết thuộc tính này nhận giá trị là các tên tham chiếu của thực thể và cách nhau bởi khoản trắng
NOTATION	(tôi chưa hiểu kiểu này)
xml:	(tôi chưa hiểu kiểu này)

- **default-value** thông tin về giá mặc định trị của thuộc tính này. Nó có thể nhận một trong các giá tị sau:

Giá trị	Mô tả
value	value là một giá trị mặc định nào đó cho giá trị này (ví dụ "CNTT")
#REQUIRED	Chỉ định là không có giá trị mặc định cho thuộc tính này, nhưng khi sử dụng là phải khởi tạo
#IMPLIED	Chỉ định là không có giá trị mặc định cho thuộc tính này, và thuộc tính này không cần dùng đến
#FIXED value	Chỉ định thuộc tính này chỉ mang duy nhất giá trị value này

Chúng ta có thể định nghĩa một phần tử có nhiều thuộc tính theo cú pháp sau:

<!ATTLIST element-name

attribute-name_1 attribute-type_1 default-value_1

attribute-name_2 attribute-type_2 default-value_2

...

attribute-name_n attribute-type_n default-value_n>

(Xem ví dụ1)

Ví dụ1:

Giả sử chúng ta có file att.dtd với nội dung sau:

```
<!ELEMENT attributes (#PCDATA)>
 <!ATTLIST attributes   aaa CDATA #REQUIRED  bbb CDATA #IMPLIED>
```

File XML chúng ta viết như sau:

```
<?xml version="1.0"?>
<!DOCTYPE attributes SYSTEM "att.dtd">
 <attributes aaa="#d1" bbb="*~*">Text</attributes>
```

Ví dụ2:

Giả sử chúng ta có file att.dtd với nội dung sau:

```
<!ELEMENT attributes (#PCDATA)>
    <!ATTLIST attributes   aaa CDATA #IMPLIED
```

```
                    bbb NMTOKEN #REQUIRED
                    ccc NMTOKENS #REQUIRED>
```

File XML chúng ta viết như sau:

```
<?xml version="1.0"?>
<!DOCTYPE attributes SYSTEM "att.dtd">
    <attributes aaa="#d1" bbb="a1:12" ccc=" 3.4 div   -4"/>
```

Nếu chúng ta viết như sau sẽ không hợp quy tắc vì kiểu NMTOKEN và NMTOKEN không chấp nhận ký tự # :

```
<?xml version="1.0"?>
<!DOCTYPE attributes SYSTEM "att.dtd">
 <attributes aaa="#d1" bbb="#d1" ccc="#d1"/>
```

Ví dụ3:

Giả sử chúng ta có file att.dtd với nội dung sau:

```
<!ELEMENT XXX (AAA+ , BBB+ , CCC+)>
   <!ELEMENT AAA (#PCDATA)>
  <!ELEMENT BBB (#PCDATA)>
   <!ELEMENT CCC (#PCDATA)>
     <!ATTLIST AAA  id ID #REQUIRED>
     <!ATTLIST BBB  code ID #IMPLIED
                    list NMTOKEN #IMPLIED>
     <!ATTLIST CCC X ID #REQUIRED
                   Y NMTOKEN #IMPLIED>
```

File XML chúng ta viết như sau:

```
<?xml version="1.0"?>
<!DOCTYPE XXX SYSTEM "att.dtd">
    <XXX>
      <AAA id="L12"/>
      <BBB code="QW" list="L12"/>
      <CCC X="x-0" Y="QW" />
      <CCC X="x-1" Y="QW" />
    </XXX>
```

Nếu chúng ta viết như sau sẽ không hợp quy tắc vì phần tử CCC có thuộc tính X có kiểu là ID nên phải là duy nhất.

```
<?xml version="1.0"?>
<!DOCTYPE XXX SYSTEM "att.dtd">
    <XXX>
```

```
    <AAA id="L12"/>
    <BBB code="QW" list="L12"/>
    <CCC X="ZA" Y="QW" />
    <CCC X="ZA" Y="QW" />
  </XXX>
```

Nếu chúng ta viết như sau sẽ không hợp quy tắc vì phần tử AAA và CCC có thuộc tính có kiểu là ID nên không được có giá trị giống nhau.

```
<?xml version="1.0"?>
<!DOCTYPE XXX SYSTEM "att.dtd">
  <XXX>
    <AAA id="L12"/>
    <BBB code="QW" list="L12"/>
    <CCC X="L12" Y="QW" />
  </XXX>
```

Ví dụ4

Giả sử chúng ta có file att.dtd với nội dung sau:

```
<!ELEMENT XXX (AAA+ , BBB+, CCC+, DDD+)>
  <!ELEMENT AAA (#PCDATA)>
  <!ELEMENT BBB (#PCDATA)>
  <!ELEMENT CCC (#PCDATA)>
  <!ELEMENT DDD (#PCDATA)>
  <!ATTLIST AAA
      mark ID #REQUIRED>
  <!ATTLIST BBB
      id ID #REQUIRED>
  <!ATTLIST CCC
      ref IDREF #REQUIRED>
  <!ATTLIST DDD
      ref IDREFS #REQUIRED>
```

File XML chúng ta viết như sau là hợp quy tắc:

```
<?xml version="1.0"?>
<!DOCTYPE XXX SYSTEM "att.dtd">
  < XXX>
    <AAA mark="a1"/>
    <AAA mark="a2"/>
    <AAA mark="a3"/>
    <BBB id="b001" />
```

```
    <CCC ref="a3" />
    <DDD ref="a1 b001 a2" />
  </XXX>
```

Nếu chúng ta viết như sau sẽ không hợp quy tắc vì phần tử DDD có thuộc tính ref có kiểu là IDREFS, trong khi đó chúng ta lại gán giá trị cho thuộc tính của phần tử này là ref=”a1 b001 a2” trong khi đó b001 không phải là giá trị của một ID nào cả.

```
<?xml version="1.0"?>
<!DOCTYPE XXX SYSTEM "att.dtd">
  <XXX>
    <AAA mark="a1"/>
    <AAA mark="a2"/>
    <BBB id="b01" />
    <CCC ref="a3" />
    <DDD ref="a1 b001 a2" />
  </XXX>
```

Ví dụ 5.

Giả sử chúng ta có file att.dtd với nội dung sau:

```
<!ELEMENT XXX (AAA+, BBB+)>
  <!ELEMENT AAA (#PCDATA)>
  <!ELEMENT BBB (#PCDATA)>
  <!ATTLIST AAA   true ( yes | no ) #REQUIRED>
  <!ATTLIST BBB  month (1|2|3|4|5|6|7|8|9|10|11|12) #IMPLIED>
```

File XML chúng ta viết như sau là hợp quy tắc:

```
<?xml version="1.0"?>
<!DOCTYPE XXX SYSTEM "att.dtd">
  <XXX>
    <AAA true="yes"/>
    <AAA true="no"/>
    <AAA true="yes"/>
    <BBB month="8" />
    <BBB month="2" />
    <BBB month="12" />
  </XXX>
```

Nếu chúng ta viết như sau sẽ không hợp quy tắc vì phần tử AAA và phần tử BBB có thuộc tính true và month có kiểu liệt kê, trong khi đó chúng ta gán giá trị cho hai thuộc tính này ngoài giá trị đã liệt kê.

```
<?xml version="1.0"?>
<!DOCTYPE XXX SYSTEM "att.dtd">
<XXX>
        <AAA true="yes"/>
        <AAA true="no"/>
        <AAA true="maybe"/>
        <BBB month="8" />
        <BBB month="2" />
        <BBB month="16" />
      </XXX>
```

2.4 Thực thể(Entity)

Như ở chương 1 đã đề cập đến thực thể nhưng đó chỉ là những thực thể đã được định nghĩa sẵn. Bây giờ chúng ta cần tìm hiểu kỹ hơn về thực thể là gì và cách định nghĩa một thực thể.

2.4.1 Thực thể là gì?

Thực thể thực chất là một cách định nghĩa một biến lưu trữ một khối dữ liệu, khi thực thể này được triệu gọi thì nó sẽ chèn nguyên khối dữ của nó vào vị tí triệu gọi. Khối dữ liệu của thực thể thường là ở dạng text, tuy nhiên nó cũng có thể là dữ liệu nhị phân, miễn là khối dữ liệu này không phá vỡ khuôn dạng của một tài liệu XML khi nó được gọi.

Có hai loại thực thể đó là thực thể tổng quát và thực thể tham số. Thực thể được khai báo trong phần định nghĩa DTD.

Để tham chiếu đến thực thể tổng quát chúng ta viết theo cú pháp:

&name_entity;

Trong đó name_entity là tên thực thể tổng quát cần tham chiếu. Lưu ý là bắt đầu bởi ký tự & và kết thúc bởi dấu chấm phẩy.

Để tham chiếu đến thực thể tham số chúng ta viết theo cú pháp:

%name_entity;

Trong đó name_entity là tên thực thể tham số cần tham chiếu. Lưu ý là bắt đầu bởi ký tự % và kết thúc bởi dấu chấp phẩy

2.4.1.1 Thực thể tổng quát

Có hai loại thực thể tổng quát đó là thực thể tổng quát nội và thực thể tổng quát ngoại.

2.4.1.1.1 Thực thể tổng quát nội

Thực thể tổng quát nội là thực thể được định nghĩa ngay trên DTD của tài liệu XML.

Chúng ta định nghĩa theo cú pháp sau:

<!ENTITY entity-name “entity-value” >

Ví dụ:

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

```
<!DOCTYPE attributes [
   <!ELEMENT attributes (#PCDATA)>
   <!ATTLIST attribute aaa CDATA  #REQUIRED>
   <!ENTITY out-text “TT CN PM”>
]>
 <attributes aaa="C" >&out-text;</attributes>
```

Đối với thực thể này chúng ta cũng có thể định nghĩa các thực thể tham chiếu lồng nhau.

Ví dụ:

<!ENTITY name “Open source software”>

<!ENTITY name-group “&name; Group”>

Tuy nhiên chúng ta không thể đảo ngược lại

<!ENTITY name-group “&name; Group”>

<!ENTITY name “Open source software”>

2.4.1.1.2 Thực thể tổng quát ngoại

Thực thể tổng quát ngoại là thực thể được định nghĩa và tham chiếu từ một nguồn bên ngoài.

Chúng ta định nghĩa định nghĩa theo 1 trong 2 cú pháp sau:

<!ENTITY entity-name SYSTEM "URI/URL">

<!ENTITY entity-name PUBLIC FPI "URI/URL">

Trong đó:

FPI đã được đề cập đến trong phần 1.2.1.2

URI/URL là địa chỉ đến nguồn dữ liệu cần gán cho entity-name

Ví dụ:

```
<?xml version="1.0"?>
<!DOCTYPE author [
   <!ELEMENT author (#PCDATA)>
   <!ATTLIST author CR CDATA  #REQUIRED>
   <!ENTITY writer   SYSTEM
"http://www.w3schools.com/entities/entities.xml">
   <!ENTITY copyright  SYSTEM "copyright.txt">
]>
 <author CR="C" >& writer; &copyright; </author>
```

Chú ý: Chúng ta không thể dùng tham chiếu thực thể tổng quát ngay trong bản thân các khai báo DTD

2.4.1.2 Thực thể tham số

Thực thể tham số khác với thực thể tổng quát ở chỗ là nó cho phép tham chiếu đến nó ngay trong bản thân các khai báo DTD và vùng hoạt động của nó chỉ nằm trong vùng khai báo các DTD.

Mục đích của đích của việc sử dụng thực thể tham số là để tránh các khai báo lặp lại khi định nghĩa DTD và giúp cho chúng ta dễ dàng thay đổi.

Tương tự như thực thể tổng quát, thực thể tham số cũng có hai loại đó là thực thể tham số ngoại và thực thể tham số nội.

2.4.1.2.1 *Thực thể tham số nội*

Thực thể tham số nội là thực thể được định nghĩa ngay trên DTD của tài liệu XML. Định nghĩa thực thể tham số chúng ta dùng cú pháp sau:

<!ENTITY % entity-name "entity-vale">

Trong đó:

% là tham số bắt buộc

entity-name là tên của thực thể tham số cần định nghĩa

entity-value là giá trị cần gán cho entity-name

Ví dụ:

Có sử dụng thực thể tham số nội	Không sử dụng thực thể tham số nội
<?xml version="1.0"?> <!DOCTYPE author [<!ENTITY name "Open source software"> <!ENTITY name-group "&name; Group"> <!ENTITY % EL "<!ELEMENT author (#PCDATA)>" > %EL;]> <author>&name-group;</author>	<?xml version="1.0"?> <!DOCTYPE author [<!ENTITY name1 "Open source software"> <!ENTITY name-group "&name1; Group"> <!ELEMENT author (#PCDATA)> >]> <author>&name-group;</author>

2.4.1.2.2 *Thực thể tham số ngoại*

Thực thể tham số ngoại là thực thể được định nghĩa và tham chiếu từ một nguồn bên ngoài. Định nghĩa thực thể tham số ngoại chúng ta viết theo một trong hai cú pháp sau:

<!ENTITY % entity-name SYSTEM "URI/URL">

<!ENTITY % entity-name PUBLIC FPI "URI/URL">

Trong đó:

Từ khóa SYSTEM cho biết đây là thực thể tham số ngoại riêng

Từ khóa PUBLIC cho biết đây là thực thể tham số ngoại chung

FPI (Formal Public Identifier) là một định danh chung hình thức (đã trình bày ở phần 1.2.1.2).

URI/URL là địa chỉ của khối giữ liệu cần gán cho entity-name

Ví dụ:

Giả sử chúng ta có file hocsinh.dtd như sau:

```
<!ELEMENT HOCSINH (HOTEN, NGAYSINH, LOP)>
<!ELEMENT HOTEN (#PCDATA)>
<!ELEMENT NGAYSINH (#PCDATA)>
<!ELEMENT LOP (#PCDATA)>
```

Bây giờ chúng ta viết file tài liệu XML có tên test.xml với thực thể tham số ngoại như sau:

```
<?xml version=”1.0”?>
<!DOCTYPE HOCSINH [
<!ENTITY % hs SYSTEM “hocsinh.dtd”>
%hs;
]>
<HOSCINH>
   <HOTEN>Le Van A</HOTEN>
   <NGAYSINH>26-06-79</NGAYSINH>
   <LOP>6A3</LOP>
</HOSCINH>
```

Viết <!ENTITY % hs SYSTEM “hocsinh.dtd”> có nghĩa là file hocsinh.dtd nằm cùng thư mục với file test.xml.

Nếu file hocsinh.dtd đặt tại địa chỉ http://hs.com.vn/hocsinh.dtd thì chúng ta viết lại dòng đó như sau: <!ENTITY % hs SYSTEM “http://hs.com.vn/hocsinh.dtd”>

Chú ý: Trước khi có được điều lưu ý thì chúng ta hãy xem ví dụ sau:

```
<!ENTITY  % mathml-colon     ">
<!ENTITY  % mathml-prefix    ">
<!ENTITY  % mathml-exp       '%mathml-prefix;%mathml-colon;exp' >
<!ENTITY  % mathml-abs       '%mathml-prefix;%mathml-colon;abs' >
<!ENTITY  % mathml-arg       '%mathml-prefix;%mathml-colon;arg' >
<!ENTITY  % mathml-real      '%mathml-prefix;%mathml-colon;real' >
<!ENTITY  % mathml-imaginary  '%mathml-prefix;%mathml-colon;imaginary'
>
<!ELEMENT %mathml-imaginary; (#PCDATA)>
```

Đây là một DTD có định nghĩa các thực thể tham số, chúng ta thấy các thực thể tham số có thể tham chiếu lẫn nhau theo một trình tự từ trên xuống và có thể được tham chiếu ngay trong một định nghĩa element. Tuy nhiên để cho các cách tham chiếu này có thể hoạt động được thì bắt buộc nó phải được định nghĩa độc lập từ một file DTD và được tham chiếu vào tài liệu XML dưới dạng DTD tham chiếu ngoại.

```
<?xml version=”1.0”>
<!DOCTYPE exp SYSTEM “exp.dtd” [
```

```
]
>
<exp> imaginary </exp>
```

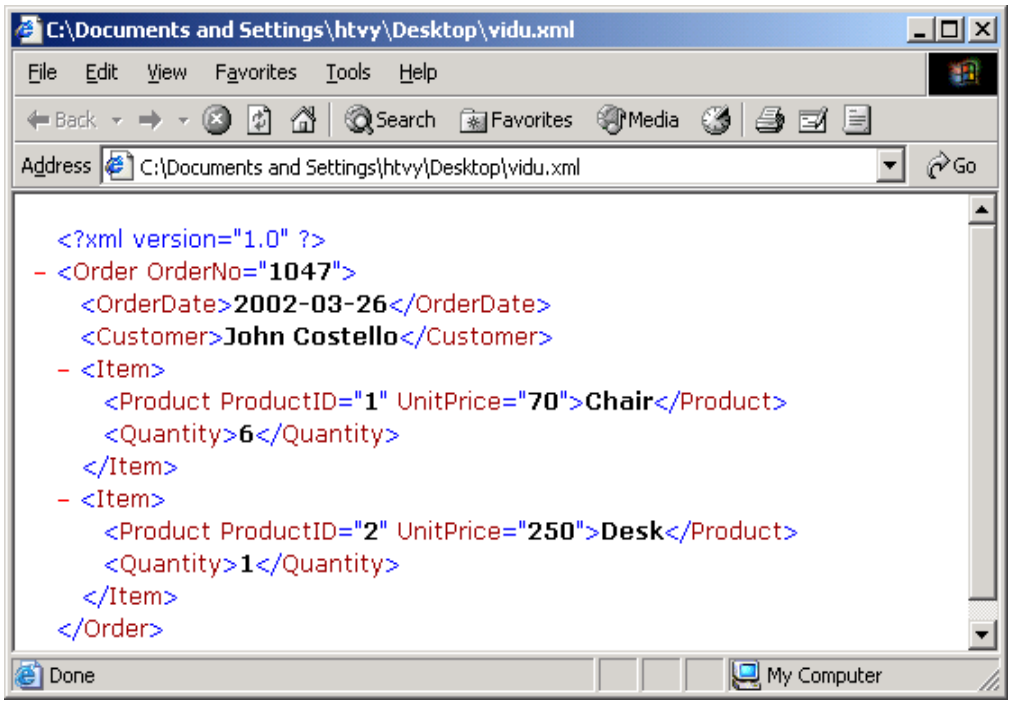
Chương 3
Xpath (XML Path Language)

1 Giới thiệu.

Trước khi đi vào phần này chúng ta hãy xem lại một ví dụ về tài liệu XML:

```
<?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>
```

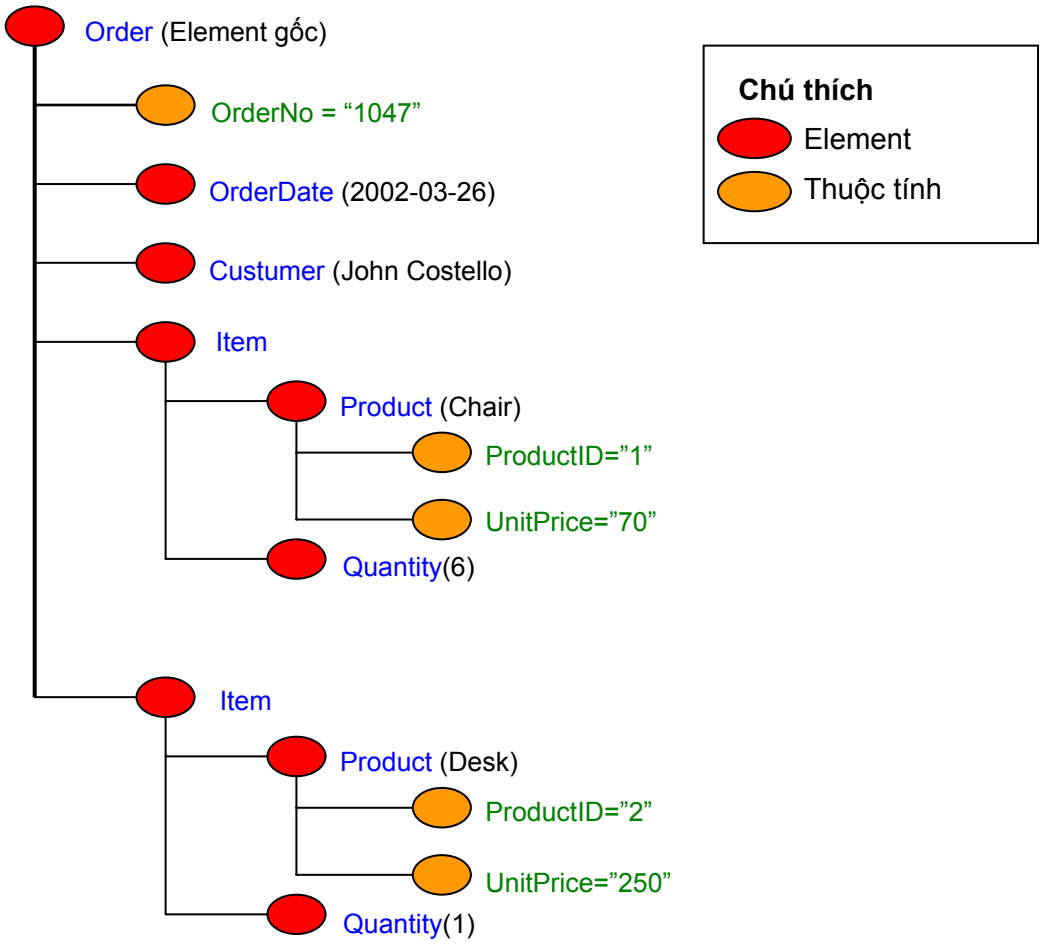
Với ví dụ này khi chúng ta mở với trình duyệt IE chúng ta sẽ được kết quả sau:



Như vậy chúng ta thấy trên trình duyệt sẽ hiển thị y nguyên tài liệu gốc. Vậy làm cách nào để chúng ta có thể đi lại trên các phần tử của tài liệu XML để trích ra những dữ liệu mà chúng ta cần thiết.

Để đáp ứng điều này người ta thiết kế ra một ngôn ngữ XPath. XPath có một vai trò quan trọng trong việc trao đổi dữ liệu giữa các máy tính hay giữa các chương trình ứng dụng vì nó cho chúng ta sàng lọc các dữ liệu mà ta mong muốn.

XPath xem XML như một cây, với ví dụ trên sẽ được biểu diễn dưới dạng cây sau:



Hình 2.2

Bây giờ chúng ta hãy học cách đi qua các nút trong tài liệu XML.

2 Cú pháp của XPath

2.1 Đường dẫn tuyệt đối

Nếu đường dẫn XPath bắt đầu bởi dấu / thì có nghĩa đây là một đường dẫn tuyệt đối bắt đầu từ phần tử gốc.

Trong hình2.2 ở trên, bây giờ chúng ta muốn chọn nút Order ta viết như sau

Cú pháp nguyên: **/child::Order**

Cú pháp tắt: **/Order**

Đi ra nhánh con **Custumer** bằng XPath như sau:

Cú pháp nguyên: **/child::Order/child::Custumer**

Cú pháp tắt: **/Order/Custumer**

Trong trường hợp muốn đi đến thuộc tính của nút thì chúng ta cần phải chỉ rõ từ khóa **Attribute** trong cú pháp nguyên hoặc **@** trong cú pháp tắt.

Để lấy thuộc tính **OrderNo** của nút **Order** ta dùng cú pháp XPath như sau:

Cú pháp nguyên: **/child::Order/Attribute::OrderNo**

Cú pháp tắt: **/Order/@OrderNo**

2.2 Đường dẫn tương đối

Khi chúng ta muốn trích một phần tử nào đó mà chúng ta chỉ biết tên của phần tử này chứ chúng ta không biết là phần tử này nằm ở vị trí nào thì chúng ta có thể dùng đường dẫn tương đối để làm điều này. Chúng ta dùng dấu **//** để chỉ cho trình phân tích biết đây là đường dẫn tương đối.

Ví dụ, để trích các phần tử có tên là **Product** chúng ta viết như sau:

Cú pháp nguyên: **//child::Product**

Cú pháp viết tắt: **//Product**

Khi chúng ta viết như thế này thì khi đi qua trình phân tích sẽ truy tìm đến các phần tử có tên là **Product**

2.3 Chọn các phần tử bằng ký tự đại diện

Để chọn tất cả các phần tử con của một phần tử nào đó chúng ta dùng ký tự đại diện ***.**

Ví dụ, để lấy tất cả các phần tử con của phần tử Order ta viết như sau:

Cú pháp nguyên: **/child::Order/child::***

Cú pháp tắt: **/Order/***

2.4 Chọn các phần tử theo điều kiện

Để lấy các phần tử theo một điều kiện nào đó chúng ta dùng dấu ngoặc vuông(**[ ]**).

Ví dụ, để lấy mọi phần tử **Product** có thuộc tính **UnitPrice > 70** ta viết như sau:

Cú pháp nguyên:

//child::Product[Attribute::UnitPrice>70]

Cú pháp tắt:

//Product[@UnitPrice>70]

Ví dụ, để lấy những phần tử **Item** có phần tử con là **Product** và có thuộc tính **ProductID=1** chúng ta viết như sau:

Cú pháp nguyên:

//child::Item[child::Product/Attribute::ProductID=1]

Cú pháp tắt:

//Item[Product/@ProductID=1]

2.5 Một số hàm thường dùng

Tên hàm	Ý nghĩa	Ví dụ
count()	Hàm lấy tổng số nút con của một phần nào đó	**//Item[count(*)=2]** Chọn tất cả các phần tử Item có số phần

		tử con là 2
name()	Lấy tên của phần tử	**/Order/*[name()='Item']** Chọn tất cả các phần tử con của Order có tên là Item
not()	Hàm phủ định	**//Item/*[not(@*)]** Chọn tất cả các phần tử con của Item không chứa thuộc tính nào
normalize-space(str)	Hàm bỏ khoảng trắng	**//Item/*[normalize-space(@ProductID)='abc']** Chọn tất cả các phần tử con của Item có thuộc tính ProductID=abc (không phân biệt khoảng trắng)
starts-with(str,substr)	Hàm kiểm tra xem chuỗi str có chứa chuỗi substr (tính từ vị trí đầu tiên) hay không	**//item/*[starts-with(name(),'P')]** Chọn tất cả các phần tử con của Item có tên bắt đầu bởi ký tự P
contains(str,substr)	Kiểm tra một chuỗi str có chứa chuỗi con substr hay không	**//item/*[contains(name(),'u')]** Chọn tất cả các phần tử con của phần tử Item mà tên của các phần tử con này có chứ ký tự u
string-length(str)	Hàm lấy chiều dài của 1 chuỗi	**//Item/*[string-length(name())=5]** Chọn tất cả các phần tử con của Item mà độ dài tên của các phần tử con này là 5
position()	Cho biết vị trí hiện tại của phần tử	**//Item[position()=5]** Chọn phần tử Item có vị trí là 5
floor()	Lấy giá trị nhỏ nhất gần với giá trị chỉ định	
ceiling()	Lấy giá trị lớn nhất gần với giá trị chỉ định	
last()	Vị trí nút cuối cùng	**//Item[last()]** Chọn phần tử Item cuối cùng

2.6 Một số toán tử thường dùng

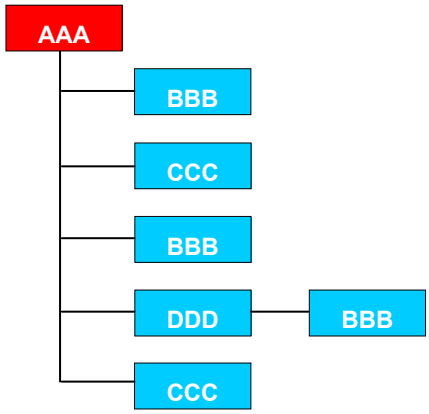
Tên toán tử	Chức năng	Ví dụ
|	Toán tử hoặc dùng để chọn ra một lần nhiều phần tử có điều kiện khác nhau	**//Item/*[starts-with(name(),'U') | starts-with(name(),'Q')]** Chọn tất cả các phần tử là con của Item có có tên bắt đầu bởi ký tự P hoặc Q
descendant	Chọn phần tử con của phần tử chỉ định	**/Order /Item/Product/descendant::*** Chọn tất cả các phần tử là con của /Order/Item/Product
ancestor	Chọn phần tử cấp trên	**/Order/Item/Product/ancestor::***

		chọn 2 phần tử Item và phần tử Order
following-sibling	Chọn phần tử cùng cấp kế tiếp	**/Order/OrderDate/following-sibling::*** chọn các phần tử Custumer và hai phần tử Item theo sau và cùng cấp với phần tử OrderDate
preceding-sibling	Chọn phần tử cùng cấp trước đó	**/Order/Custumer/preceding-sibling::*** chọn phần tử OrderDate
following	Chọn phần tử theo sau phần tử chỉ định	**/Order/OrderDate/following::*** chọn phần tử Custumer và 2 phần tử Item và các phần tử con của Item
preceding	Chọn các phần tử đứng trước phần tử chỉ định	**/Order/Custumer/preceding::*** chọn tất cả các phần tử đi trước phần tử Custumer
descendant-or-self	Chọn phần tử cấp dưới và phần tử chỉ định	**/Order/Item/descendant-or-self::*** Chọn tất cả các phần tử Item và các phần tử con của phần tử này
ancestor-or-self	Chọn phần tử cấp trên và phần tử chỉ định	**/Order/Item/product/ancestor-or-self::*** chọn 2 phần tử product, 2 phần tử Item và phần tử Order

3 Một số ví dụ

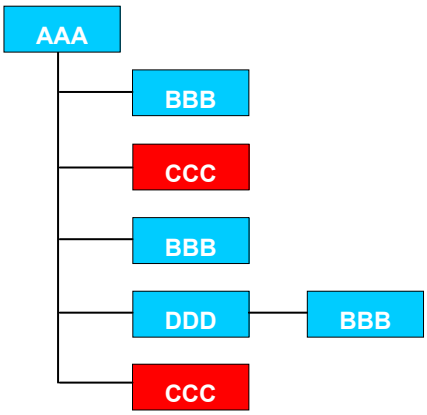
Chọn phần tử gốc AAA (/AAA)

```
<AAA>
      <BBB/>
      <CCC/>
      <BBB/>
      <BBB/>
      <DDD>
       <BBB/>
      </DDD>
      <CCC/>
</AAA>
```



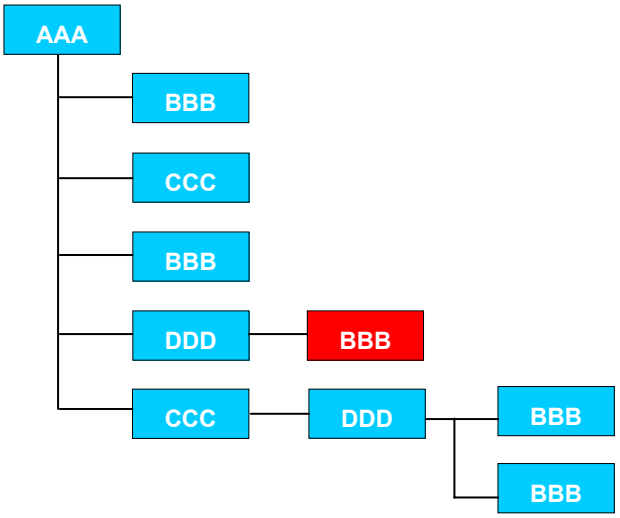
Chọn phần tử CCC là con của AAA (/AAA/CCC)

```
<AAA>
      <BBB/>
      <CCC/>
      <BBB/>
      <BBB/>
      <DDD>
      <BBB/>
      </DDD>
      <CCC/>
 </AAA>
```



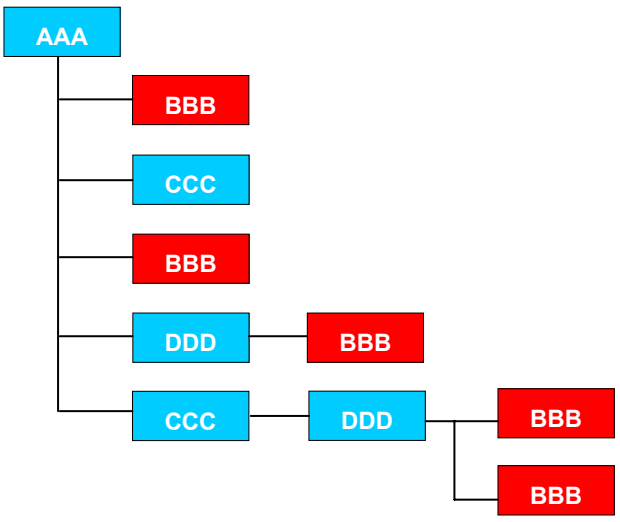
Chọn tát cả các phần tử BBB là con của DDD mà DDD là con của AAA (/AAA/DDD/BBB)

```
<AAA>
      <BBB/>
      <CCC/>
      <BBB/>
      <BBB/>
      <DDD>
         <BBB/>
      </DDD>
      <CCC/>
 </AAA>
```



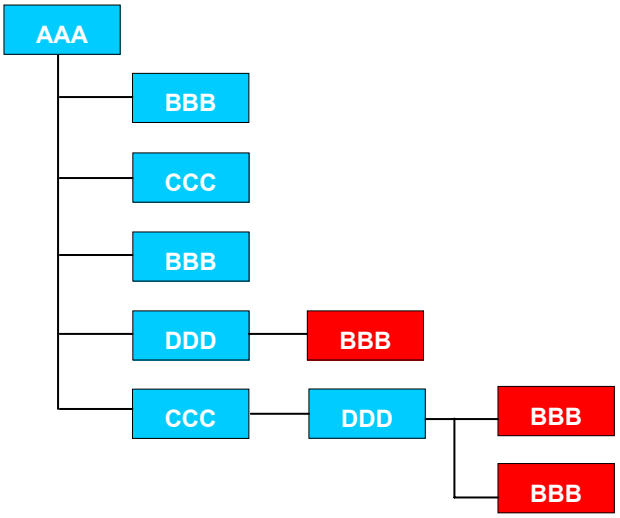
Chọn tất cả các phần tử BBB (//BBB)

```
<AAA>
      <BBB/>
      <CCC/>
      <BBB/>
      <DDD>
         <BBB/>
      </DDD>
      <CCC>
         <DDD>
            <BBB/>
            <BBB/>
         </DDD>
      </CCC>
   </AAA>
```



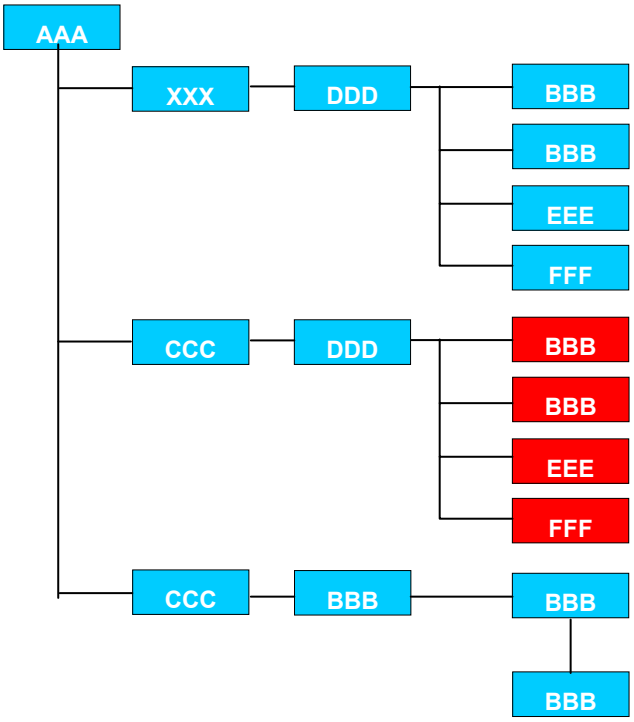
Chọn tất cả các phần tử BBB là con của DDD (//DDD/BBB)

```
<AAA>
    <BBB/>
    <CCC/>
    <BBB/>
    <DDD>
        <BBB/>
    </DDD>
    <CCC>
        <DDD>
            <BBB/>
            <BBB/>
        </DDD>
    </CCC>
</AAA>
```



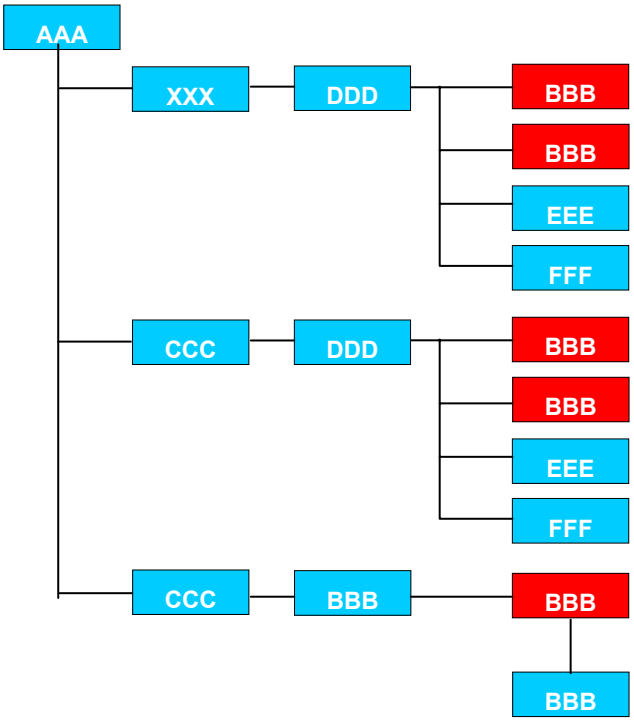
Chọn tất cả các phần tử mà dòng họ của nó là /AAA/CCC/DDD (/AAA/CCC/DDD/*)

```
<AAA>
    <XXX>
        <DDD>
            <BBB/>
            <BBB/>
            <EEE/>
            <FFF/>
        </DDD>
    </XXX>
    <CCC>
        <DDD>
            <BBB/>
            <BBB/>
            <EEE/>
            <FFF/>
        </DDD>
    </CCC>
    <CCC>
        <BBB>
            <BBB>
                <BBB/>
            </BBB>
        </BBB>
    </CCC>
</AAA>
```



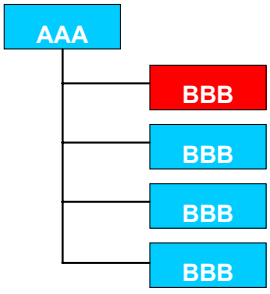
Chọn tất cả các phần tử BBB mà nó có 3 cấp cha (/*/*/*/BBB)

```
<AAA>
    <XXX>
      <DDD>
        <BBB/>
        <BBB/>
        <EEE/>
        <FFF/>
      </DDD>
    </XXX>
    <CCC>
      <DDD>
        <BBB/>
        <BBB/>
        <EEE/>
        <FFF/>
      </DDD>
    </CCC>
    <CCC>
      <BBB>
        <BBB>
          <BBB/>
        </BBB>
      </BBB>
    </CCC>
  </AAA>
```



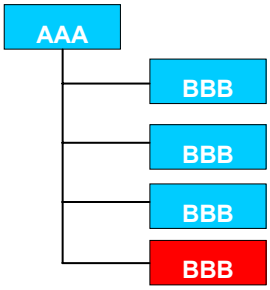
Chọn phần tử BBB đầu tiên là con của AAA (/AAA/BBB[1])

```
<AAA>
    <BBB/>
    <BBB/>
    <BBB/>
    <BBB/>
</AAA>
```



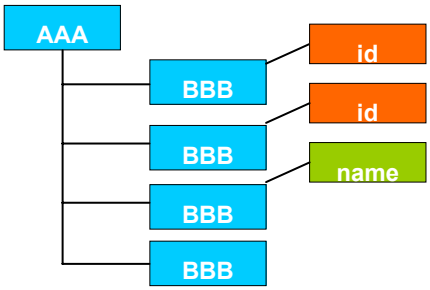
Chọn phần tử BBB cuối cùng là con của AAA (/AAA/BBB[last()])

```
<AAA>
      <BBB/>
      <BBB/>
      <BBB/>
      <BBB/>
</AAA>
```



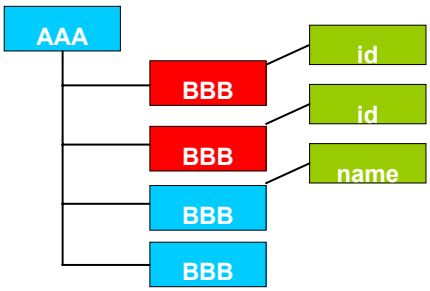
Chọn tất cả các thuộc tính có tên id (//@id)

```
<AAA>
   <BBB id = "b1"/>
   <BBB id = "b2"/>
   <BBB name = "bbb"/>
      <BBB/>
</AAA>
```



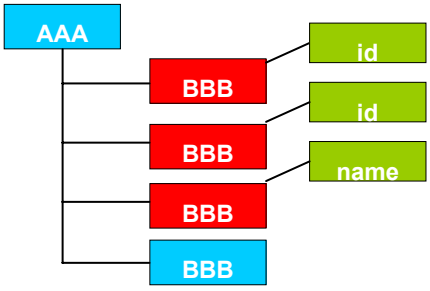
Chọn tất cả các phần tử BBB có thuộc tính tên là id (//BBB[@id])

```
<AAA>
   <BBB id = "b1"/>
   <BBB id = "b2"/>
   <BBB name = "bbb"/>
   <BBB/>
</AAA>
```



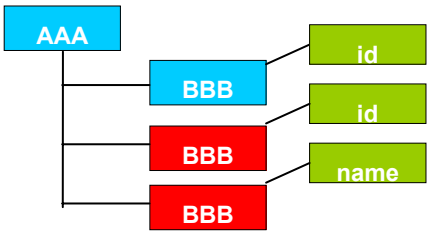
Chọn tất cả các phần tử BBB có tên thuộc tính (//BBB[@*])

```
<AAA>
   <BBB id = "b1"/>
   <BBB id = "b2"/>
   <BBB name = "bbb"/>
   <BBB/>
</AAA>
```



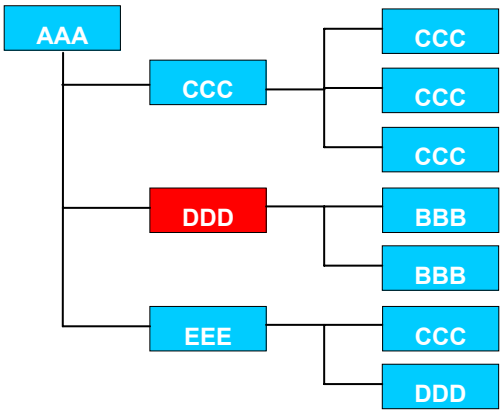
Chọn tất cả các phần tử BBB có tên thuộc tính là bbb, không phân biệt khoản trắng (//BBB[normalize-space(@name)='bbb'])

```
<AAA>
  <BBB id = "b1"/>
   <BBB name=" bbb "/>
   <BBB name = "bbb"/>
 </AAA>
```



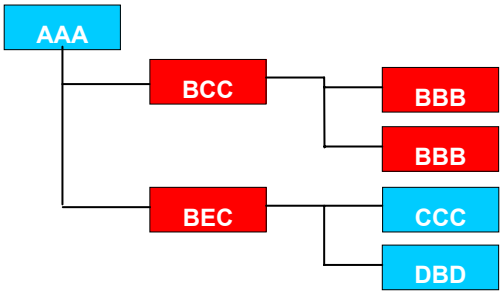
Chọn tất cả các phần tử có chứa các phần tử mà trong đó có 2 phần tử con tên là BBB (//*[count(BBB)=2])

```
<AAA>
      <CCC>
         <BBB/>
         <BBB/>
         <BBB/>
      </CCC>
      <DDD>
         <BBB/>
         <BBB/>
      </DDD>
      <EEE>
         <CCC/>
         <DDD/>
      </EEE>
   </AAA>
```



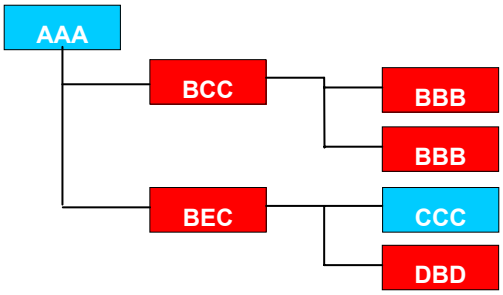
Chọn tất cả các phần tử mà tên của nó bắt đầu là ký tự B (//*[starts-with(name(),'B')])

```
<AAA>
      <BCC>
         <BBB/>
         <BBB/>
       </BCC>
       <BEC>
         <CCC/>
         <DBD/>
      </BEC>
   </AAA>
```



Chọn tất cả các phần tử mà tên của nó có chứa ký tự B(//*[contains(name(),'B')])

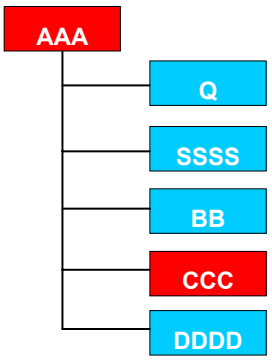
```
<AAA>
      <BCC>
         <BBB/>
         <BBB/>
       </BCC>
       <BEC>
         <CCC/>
         <DBD/>
      </BEC>
   </AAA>
```



Chọn tất cả các phần tử mà tên của nó có độ dài là 3

(//*[string-length(name())=3])

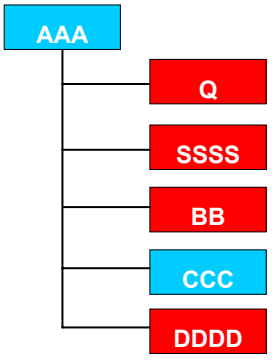
```
<AAA>
      <Q/>
      <SSSS/>
      <BB/>
      <CCC/>
      <DDDD/>
</AAA>
```



Chọn tất cả các phần tử mà tên của nó có độ dài khác 3

(//*[string-length(name())!=3])

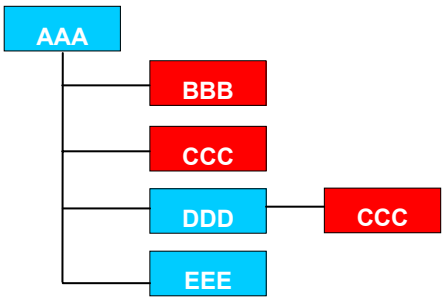
```
<AAA>
      <Q/>
      <SSSS/>
      <BB/>
      <CCC/>
      <DDDD/>
</AAA>
```



Chọn tất cả các phần tử mà tên của nó là CCC hoặc BBB

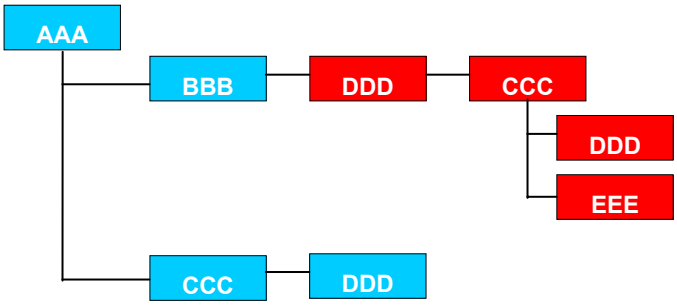
(//*[name()='CCC']| //*[name()='BBB'])

```
<AAA>
   <BBB/>
   <CCC/>
   <DDD>
      <CCC/>
   </DDD>
   <EEE/>
</AAA>
```



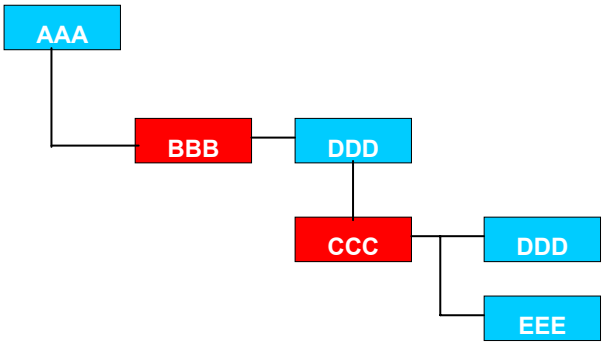
Chọn tất cả các phần tử là con của AAA/BBB (/AAA/BBB/descendant::*)

```
<AAA>
      <BBB>
         <DDD>
            <CCC>
               <DDD/>
               <EEE/>
            </CCC>
         </DDD>
      </BBB>
      <CCC>
         <DDD/>
      </CCC>
   </AAA>
```



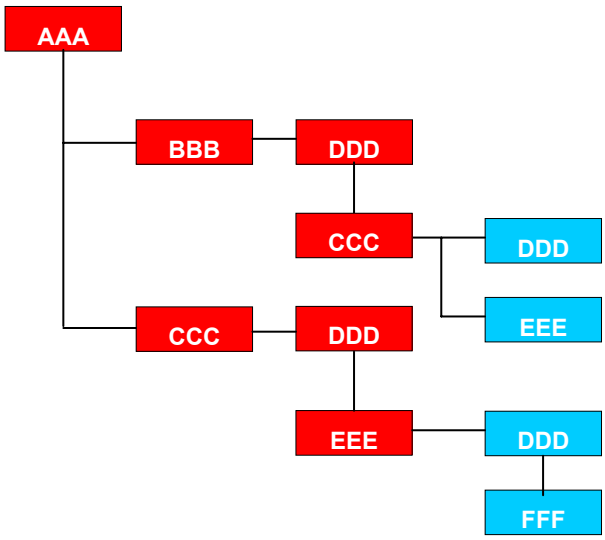
Chọn tất cả các phần tử là cha của phần tử DDD (//DDD/parent::*)

```
<AAA>
      <BBB>
         <DDD>
            <CCC>
               <DDD/>
               <EEE/>
            </CCC>
         </DDD>
      </BBB>
 </AAA>
```



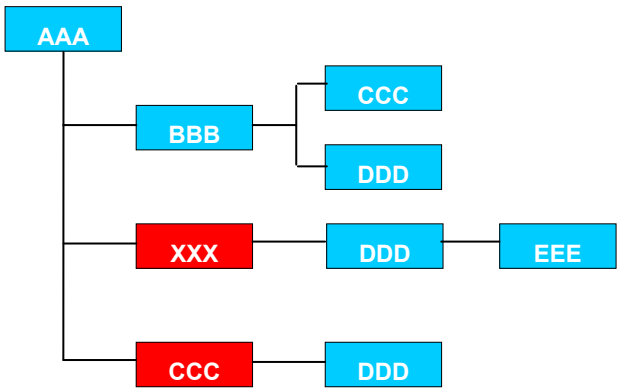
Chọn tất cả các phần tử là tổ tiên của phần tử DDD (//DDD/ancestor::*)

```
<AAA>
    <BBB>
        <DDD>
            <CCC>
                <DDD/>
                <EEE/>
            </CCC>
        </DDD>
    </BBB>
    <CCC>
        <DDD>
            <EEE>
                <DDD>
                    <FFF/>
                </DDD>
            </EEE>
        </DDD>
    </CCC>
</AAA>
```



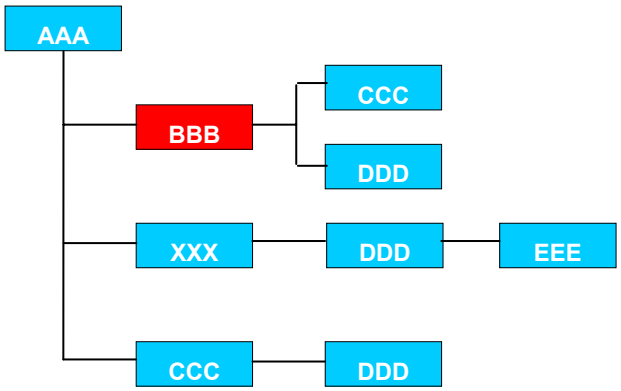
Chọn tất cả các phần tử cùng cấp đi sau phần tử BBB (//BBB/following-sibling::*)

```
<AAA>
    <BBB>
        <CCC/>
        <DDD/>
    </BBB>
    <XXX>
        <DDD>
            <EEE/>
        </DDD>
    </XXX>
    <CCC>
        <DDD/>
    </CCC>
</AAA>
```



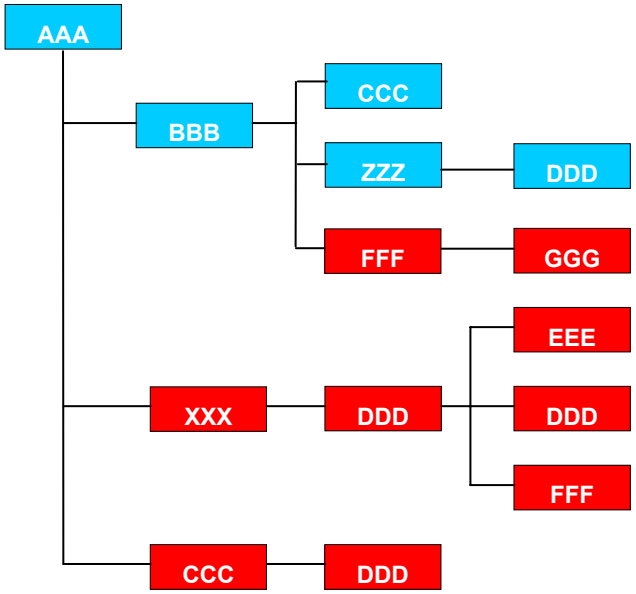
Chọn tất cả các phần tử cùng cấp đi trước phần tử XXX (//XXX/preceding-sibling::*)

```
<AAA>
    <BBB>
      <CCC/>
      <DDD/>
    </BBB>
    <XXX>
      <DDD>
        <EEE/>
      </DDD>
    </XXX>
    <CCC>
      <DDD/>
    </CCC>
  </AAA>
```



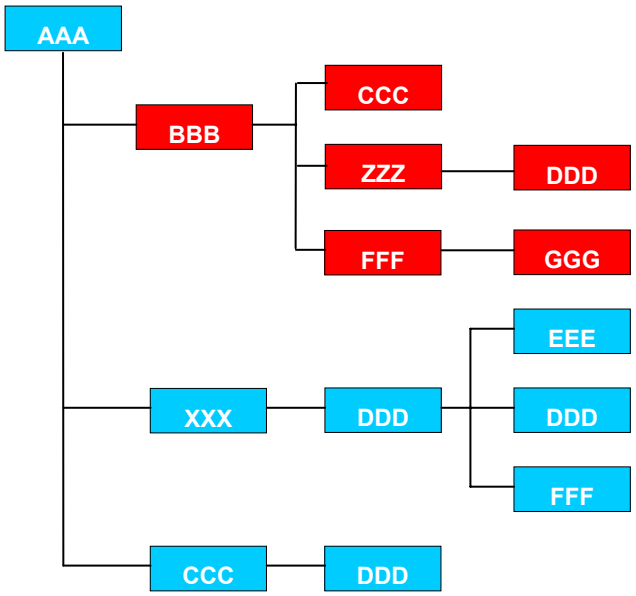
Chọn tất cả các phần tử đi sau phần tử ZZZ (//ZZZ/following::*)

```
<AAA>
    <BBB>
      <CCC/>
      <ZZZ>
        <DDD/>
      </ZZZ>
      <FFF>
        <GGG/>
      </FFF>
    </BBB>
    <XXX>
      <DDD>
        <EEE/>
        <DDD/>
        <FFF/>
      </DDD>
    </XXX>
    <CCC>
      <DDD/>
    </CCC>
  </AAA>
```



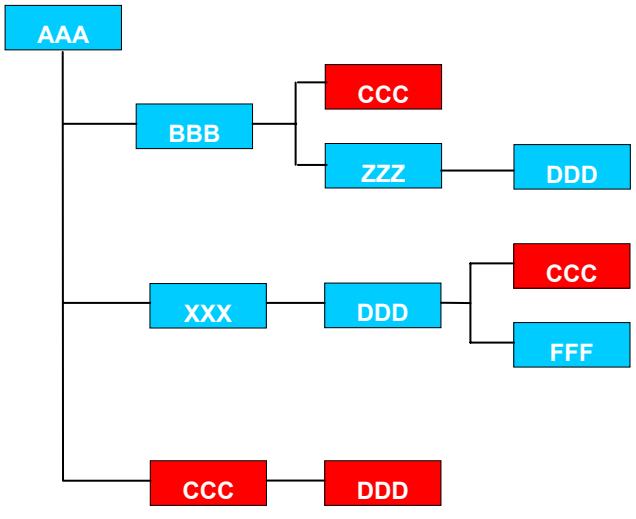
Chọn tất cả các phần tử đi trước phần tử XXX ngoại trừ những phần tử gốc (//XXX/preceding::*)

```
<AAA>
    <BBB>
      <CCC/>
      <ZZZ>
        <DDD/>
      </ZZZ>
      <FFF>
        <GGG/>
      </FFF>
    </BBB>
    <XXX>
      <DDD>
        <EEE/>
        <DDD/>
        <FFF/>
      </DDD>
    </XXX>
    <CCC>
      <DDD/>
    </CCC>
  </AAA>
```



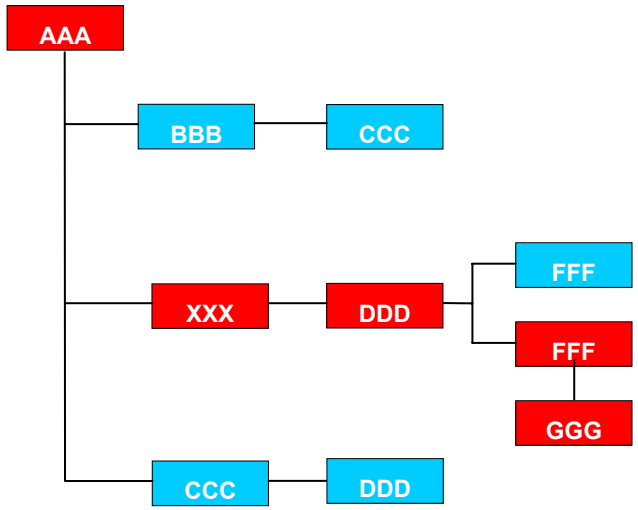
Chọn tất cả các phần tử CCC và con của nó (//CCC/descendant-or-self::*)

```
<AAA>
    <BBB>
      <CCC/>
      <ZZZ>
        <DDD/>
      </ZZZ>
    </BBB>
    <XXX>
      <DDD>
        <CCC/>
        <FFF/>
      </DDD>
    </XXX>
    <CCC>
      <DDD/>
    </CCC>
  </AAA>
```



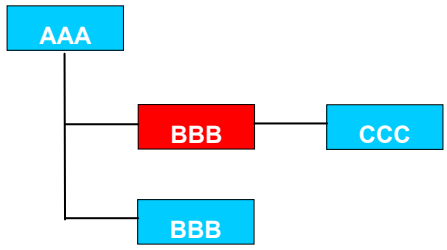
Chọn tất cả các phần tử GGG và tổ tiên của nó (//GGG/ancestor-or-self::*)

```
<AAA>
    <BBB>
       <CCC/>
    </BBB>
    <XXX>
       <DDD>
          <FFF/>
          <FFF>
             <GGG/>
          </FFF>
       </DDD>
    </XXX>
    <CCC>
       <DDD/>
    </CCC>
 </AAA>
```



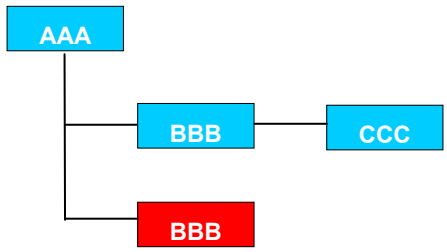
Chọn phần tử BBB đầu tiên (//BBB[floor(1.2)])

```
<AAA>
    <BBB>
       <CCC/>
    </BBB>
    <BBB/>
 </AAA>
```



Chọn phần tử BBB thứ hai (//BBB[ceiling(1.2)])

```
<AAA>
    <BBB>
       <CCC/>
    </BBB>
    <BBB/>
 </AAA>
```



Chương 4
XSL (eXtensible style sheet)

1 XSL là gì?

XSL là một ngôn ngữ chuẩn giúp chúng ta chuyển đổi tài liệu XML thành một định dạng khác như HTML, WML (Wireless (vô tuyến điện) Markup Language),... và ngay cả định dạng XML khác. Ban đầu XSL được thiết kế để sinh ra HTML những dạng khác nhau tùy theo style sheet. Nhưng bây giờ XSL rất hữu ích cho việc chuyển đổi định dạng của tài liệu XML.

Hiện tại có một phiên bản mới của XSL là XSLT(eXtensible style sheet transformations).

Trong chương trước chúng ta đã tìm hiểu về XPath, XPath giúp cho chúng ta đi lại trên các phần tử của một tài liệu XML. Nhưng để làm cho một tài liệu XML trở nên hữu ích và dễ dàng phát triển thì sự kết hợp giữu XPath và XSL là không thể thiếu

Để biết được XSL làm việc như thế nào và sự kết hợp đó như thế nào, chúng ta lần lược tìm hiểu một số cú pháp của XSL.

2 Qui tắc chung

Bản thân XSL cũng là một XML well-formed nhưng nó chứa những lệnh của chính nó và dữ liệu HTML dùng y nguyên cho dữ liệu ra. Vì vậy chúng ta phải tuân thủ mọi quy tắc của một XML well-formed.

Để trình phân tích XML nhận diện được các lệnh của XSL thì chúng ta cần phải khai báo một **namespace** trong phần tử gốc. Một style sheet thường chứa một trong hai namespace:

Namespace nguyên thủy: **http://www.w3.org/TR/WD-xsl**

Namespace của XSLT: **http://www.w3.org/1999/XSL/Transform**

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

Phần tử gốc trong tài liệu XSL thường là một phần tử **xsl:stylesheet**, nó chứa một hay nhiều phần tử **xsl:template**

Ví dụ, chúng ta có file test.xsl sau:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>Northwind Home Page</TITLE>
      </HEAD>
      <BODY>
        <P>Customer Order</P>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

Thuộc tính **match** trong phần tử **template** để chỉ ra node xuất phát.

Để tham chiếu file một tài liệu xsl vào trong tài liệu XML bằng cách thêm vào đầu tài liệu XML dòng:

<?xml-stylesheet type=”text/xsl” href=”URI/URL”?>

Trong đó URI/URL là địa chỉ của tài liệu xsl mà chúng ta muốn tham chiếu

```
<?xml-stylesheet type=”text/xsl” href=”test.xsl”?>
```

3 Một số phần tử(element) thường dùng của XSL

3.1 Phần tử value-of

Phần tử value-of có chức năng chọn giá trị của một phần tử hay một thuộc tính nào đó trong tài liệu XML để hòa nó vào tài liệu xuất. **value-of** sử dụng một thuộc tính **select** có giá trị là một biểu thức **XPath** để trích ra một phần tử. Kết quả là kết quả của việc thực hiện biểu thức **XPath**.

Ví dụ:

	Tài liệu XSL lưu với tên test.xsl	XML
1	<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" version = "1.0" >	<?xml version="1.0"?> <?xml-stylesheet type="text/xsl" href="test.xsl" ?>
2	<xsl:output method = "html" />	<AAA > <BBB>10 </BBB> <BBB>5 </BBB> <BBB>7 </BBB> </AAA>
3	<xsl:template match = "/" >	
4	<HTML> <HEAD> <TITLE>value-of</TITLE> </HEAD> <BODY>	
5	<xsl:value-of select = "//BBB[1]" /> 	Kết quả hiển thị trên trình duyệt
6	<xsl:value-of select = "//BBB[2]" /> 	10 5 7
7	<xsl:value-of select = "//BBB[3]" />	
8	</BODY> </HTML>	
9	</xsl:template>	
10	</xsl:stylesheet>	

Giải thích ví dụ:

Dòng 1: Phần tử stylesheet dùng để khai báo namespace, báo cho trình phân tích biết đây là phiên bản XSLT.

Dòng 2: Khai báo kiểu dữ liệu ra, kiểu dữ liệu ra là dưới dạng HTML

Dòng 3: Khai báo phần tử template chính và cho biết vị trí khởi đầu là phần tử gốc

Dòng 4 Các thẻ mở HTML

Dòng 5, 6, 7: Chọn nội dung của phần tử BBB thứ 1, 2, 3

Dòng 8: Các thẻ đóng HTML

Dòng 9: Thẻ đóng phần tử template chính

Dòng 10: Thẻ đóng của phần tử stylesheet

3.2 Phần tử attribute

Phần tử này giúp chúng ta đưa thêm một thuộc tính vào vào một phần tử nào đó trong hồ sơ kết quả với một trị số lấy từ tài liệu XML.

Ví dụ:

	Tài liệu XSL lưu với tên test.xsl	XML
1	<?xml version="1.0"?>	<?xml version="1.0"?>
2	<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">	<?xml-stylesheet type="text/xsl" href="test.xsl" ?> <Order OrderNo="1"> <OrderDate>2002-03-26 </OrderDate> <Item> <Product ProductID="1" UnitPrice="70">Chair </Product> </Item> </Order>
3	<xsl:template match="/">	
4	<A>	
5	<xsl:attribute name="HREF">Products.php?ProductID= <xsl:value-of select="Product/@ProductID"/> </xsl:attribute>	
6	<xsl:value-of select="Product"/>	Kết quả hiển thị trên trình duyệt
7	</A>	Chair
8	</xsl:template>	
9	</xsl:stylesheet>	

Giải thích ví dụ:

Dòng 5: Tạo một thuộc tính có tên là HREF cho phần tử A ở dòng 4.

Kết quả sẽ cho ra từ Chair, từ này có link là Products.php?ProductID=1

3.3 Phần tử attribute-set

Phần tử này dùng để tạo ra một tập các thuộc tính. Phần tử này có hai thuộc tính:

- **name**: Tên của tập thuộc tính
- **use-attribute-sets**: Nếu thuộc tính này được sử dụng thì giá trị của nó sẽ là một tên của một tập thuộc tính khác để bổ sung vào cho tập thuộc tính này

Các phần tử con của phần tử này là các phần tử **attribute**

Ví dụ:Xem ví dụ ở mục 1.2.4.

3.4 Phần tử element

Phần tử này cho phép chúng ta thêm một phần tử vào tài liệu kết quả. Phần tử này có 3 thuộc tính:

- **name**: Giá trị là một tên của phần tử cần định nghĩa
- **namespace**: Giá trị là một không gian tên
- **use-attribute-set**: Giá trị của nó là một hoặc nhiều tên của các phần tử attribute hay attribute-set khác (có nghĩa là chúng ta muốn dùng các thuộc tính đã được định nghĩa trong các phần tử attribute).

Ví dụ:

	Tài liệu XSL lưu với tên test.xsl	XML
1	<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" version = "1.0" >	<?xml version=”1.0” encoding=”utf-8”?> <?xml-stylesheet type="text/xsl" href="test.xsl" ?> <AAA > <BBB>bbb </BBB> <CCC>ccc </CCC> </AAA>
2	<xsl:output method = "xml" indent = "yes" />	
3	<xsl:attribute-set name = "xxx" >	
4	<xsl:attribute name = "a" >1</xsl:attribute>	KẾT QUẢ
5	<xsl:attribute name = "b" >2</xsl:attribute>	<?xml version="1.0" encoding="utf-8"?> <QQQ a="1" b="2" cc="33" dd="44" xxx="555"/>
6	</xsl:attribute-set>	
7	<xsl:attribute-set name = "yyy" use-attribute-sets = "xxx" >	
8	<xsl:attribute name = "cc" >33 </xsl:attribute>	
9	<xsl:attribute name = "dd" >44 </xsl:attribute>	
10	</xsl:attribute-set>	
11	<xsl:template match = "/" >	
12	<xsl:element name = "QQQ" use-attribute-sets = "yyy" >	
13	<xsl:attribute name = "xxx" >555 </xsl:attribute>	
14	</xsl:element>	
15	</xsl:template>	
16	</xsl:stylesheet>	

Giải thích ví dụ:

Dòng 3: Thiết lập tập thuộc tính, tập có tên là xxx

Dòng 4, 5: Thiết lập hai thuộc tính a và b cho tập thuộc tính xxx

Dòng 6: Thiết lập tập thuộc tính, tập có tên là yyy, ngoài các thuộc tính được thiết lập ở dòng 8, 9 còn sử dụng thêm tập thuộc tính xxx.

Dòng 8, 9: Thiết lập 2 thuộc tính cc và dd cho tập thuộc tính yyy

Dòng 11: Chỉ định phần tử gốc

Dòng 12: Thiết lập phần tử QQQ có các thuộc tính ngoài thuộc tính được thiết lập trong dòng 13 còn sử dụng thêm tập thuộc tính yyy

Kết quả là tạo ra môt tài liệu XML, tài liệu này có một Phần tử là QQQ và có các thuộc tính là a="1" b="2" cc="33" dd="44" xxx="555".

3.5 Phần tử apply-templates

Khi một style sheet chứa nhiều phần tử template, chúng ta có thể áp dụng chúng vào một khung trình bày nào đó bằng cách sử dụng phần tử **apply-templates**. Chúng ta cần tạo ra một phần tử template để chứa phần tử apply-templates, nó sẽ lấy kết quả của các template nằm bên ngoài template chứa nó để đưa vào khung trình bày của nó. Nếu trường hợp không có template nào ngoài được áp dụng thì nó sẽ tự lấy kết quả của chính bản thân nó.

Thật là khó hiểu, để dễ hiểu hơn chúng ta xem các ví dụ sau:

	Tài liệu XSL lưu với tên test.xsl	XML
1	<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" version = "1.0" >	<?xml version=”1.0”?> <?xml-stylesheet type="text/xsl" href="test.xsl" ?>
2	<xsl:output method = "html" /> <HTML> <TABLE>	<AAA > <BBB>10 </BBB> <BBB>5 </BBB>
3	<xsl:template match = "/" > <TR> <TD>	<BBB>7 </BBB> </AAA>
4	<xsl:apply-templates select = "//BBB" />	MÃ HTML KẾT QUẢ
5	</TD> </TR> </xsl:template> </TABLE> </HTML>	<HTML> <TABLE> <TR> <TD>BBB[1]: 10
6	<xsl:template match = "BBB" >	</TD> <TD> BBB[2]: 5
7	BBB[<xsl:value-of select = "position()" />]: <xsl:value-of select = "." />	</TD> <TD> BBB[3]: 7

```
8       </xsl:template>                                  </TD>
                                                        </TR>
9   </xsl:stylesheet>                                   </TABLE>
                                                    </HTML>
```

	Tài liệu XSL lưu với tên test.xsl	XML
1	<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" version = "1.0" >	<?xml version=”1.0”?> <?xml-stylesheet type="text/xsl" href="test.xsl" ?>
2	<xsl:output method = "html" /> <HTML> <TABLE>	<AAA > <BBB>10 </BBB> <BBB>5 </BBB>
3	<xsl:template match = "/" > <TR> <TD>	<BBB>7 </BBB> </AAA>
4	<xsl:apply-templates select = "//BBB" />	MÃ HTML KẾT QUẢ
5	</TD> </TR> </xsl:template> </TABLE> </HTML>	<HTML> <TABLE> <TR> <TD> 10 </TD> <TD>5</TD> <TD> 7</TD> </TR> </TABLE> </HTML>
6	</xsl:stylesheet>	

3.6 Phần tử call-template

Phần tử này được dùng để triệu gọi một **xsl:template** bởi tên của **xsl:template** này.

Ví dụ:

	Tài liệu XSL lưu với tên test.xsl	XML

1	<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" version = "1.0" > <xsl:output method = "text" />	<?xml version=”1.0”?> <?xml-stylesheet type="text/xsl" href="test.xsl" ?> <AAA > <BBB>bbb </BBB> <CCC>ccc </CCC> </AAA>
2	<xsl:template match = "/" > <xsl:call-template name = "print" /> </xsl:template>	Kết quả hiển thị trên trình duyệt bbb ccc
3	<xsl:template name = "print" > <xsl:value-of select=”.”> </xsl:template> </xsl:stylesheet>	

3.7 Phần tử for-each

Phần tử for-each dùng để đi qua tất cả các phần tử được chỉ định ra trong thuộc tính select (for-each làm việc cũng giống như lệnh for của các ngôn ngữ lập trình).

Ví dụ:

	Tài liệu XSL lưu với tên test.xsl	XML
1	<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" version = "1.0" >	<?xml version=”1.0”?> <?xml-stylesheet type="text/xsl" href="test.xsl" ?>
2	<xsl:output method = "text" />	<AAA > <BBB>TT </BBB>
3	<xsl:template match = "/" >	<BBB>CN</BBB>
4	<xsl:for-each select = "//BBB" >	<BBB>PM</BBB> <BBB>TTH</BBB> </AAA>
5	<xsl:value-of select = "." />	Kết quả hiển thị trên trình duyệt
6	</xsl:for-each>	TT CN
7	</xsl:template>	PM
8	</xsl:stylesheet>	TTH

Giải thích ví dụ

Dòng 4: Phần tử for-each sẽ cho phép duyệt qua hết tất cả các phần tử BBB

Dòng 5: Phần tử value-of sẽ lấy nội dung của phần tử BBB hiện thời.

3.8 Phần tử if

Phần tử if là một phần tử dùng để kiểm tra điều kiện của một biểu thức logic, nếu biểu thức logic có fía trị true thì các phần tử bên trong phần tử if sẽ được thực hiện và ngược lại thì không (cách làm việc của nó cũng giống như câu lệnh if trong các ngôn ngữ lập trình khác). Phần tử này có thuộc tính tên là test thuộc tính này chức biểu thức điều kiện. Biểu thức này có thể là một biểu thức so sánh hoặc một biểu thức XPath, kết quả là true khi kết quả của biểu thức nhận một trong các giá trị sau:

- Một nút có ít nhất một nút
- Một con số khác không
- Một mảnh cây
- Một chuỗi không phải là rỗng

Ví dụ:

	Tài liệu XSL lưu với tên test.xsl	XML
1	<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" version = "1.0" >	<?xml version=”1.0”?>
	<xsl:output method = "text" />	<?xml-stylesheet type="text/xsl" href="test.xsl" ?>
		<AAA >
3	<xsl:template match = "BBB | CCC" >	<BBB bbb = "111" >B-1 </BBB>
		<BBB bbb = "222" >B-2 </BBB>
4	<xsl:if test = "position()=1" >	<CCC>222 </CCC>
	<xsl:value-of select = "name()" />	<CCC>333 </CCC>
	<xsl:text > : </xsl:text>	<CCC>111 </CCC>
	</xsl:if>	</AAA
		Kết quả hiển thị trên trình duyệt
		BBB : B-1 B-2 222 333 111
5	<xsl:value-of select = "." />	
6	</xsl:template>	
	</xsl:stylesheet>	

Giải thích ví dụ

Dòng 3: Dùng để chỉ ra node khởi đầu của quá trình trích dữ liệu là node BBB hoặc CCC

Dòng 4: Kiểm tra xem node hiện tại có phải là node thứ 1 hay không, nếu là node có vị trí 1 thì lấy tên của node này và dấu “:” và ngược lại thì không.

Dòng 5: Lấy nội dung của node hiện thời.

3.9 Phần tử điều khiển choose

Đây là phần tử điều khiển chọn lựa, nó làm việc giống như câu lệnh switch trong của một số ngôn ngữ lập trình. Các chọn lựa trong phần tử điều khiển choose là các phần tử xsl:when (giống như case trong trong câu lệnh switch

của ngôn ngữ C) và phần tử xsl:otherwise (Giống như default trong câu lệnh switch của ngôn ngữ C).

Phần tử choose không có thuộc tính, phần tử xsl:when có một thuộc tính test, giá trị của nó là một biểu thức, phần tử xsl:otherwise không có thuộc tính.

Để dễ hiểu hơn chúng ta xem ví dụ sau:

	Tài liệu XSL lưu với tên test.xsl	XML
1	<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" version = "1.0" >	<?xml version ="1.0"?>
	<xsl:output method = "text" />	<?xml-stylesheet type="text/xsl" href="test.xsl" ?>
		<AAA >
2	<xsl:template match = "BBB" >	<BBB>10 </BBB>
		<BBB>5 </BBB>
3	<xsl:choose >	<BBB>7 </BBB>
4	<xsl:when test = ".=7" >	</AAA>
	<xsl:text >test=7</xsl:text>	
	</xsl:when>	
5	<xsl:when test = ".=5" >	
	<xsl:text >test=5</xsl:text>	
	</xsl:when>	
6	<xsl:otherwise >	Kết quả hiển thị trên trình duyệt
	<xsl:text >otherwise</xsl:text>	otherwise
	</xsl:otherwise>	test=5
7	</xsl:choose>	test=7
8	</xsl:template>	
	</xsl:stylesheet>	

Giải thích ví dụ

Dòng 2: Chỉ định node bắt đầu

Dòng 3: Phần tử lựa chọn

Dòng 4: Kiểm tra xem giá trị của node hiện tại có bằng 7 hay không nếu bằng thì cho ra câu test=7

Dòng 5: Thực hiện công việc giống dòng 4 nhưng kiểm tra xem giá trị của node hiện tại có bằng 5 hay không, nếu bằng thì cho ra câu test=5

Dòng 6: Nếu hai điều kiện trên không thỏa thì cho ra câu ortherwise

Kết quả:

Lần lượt đi qua 2 node BBB, đầu tiên là node có giá trị là 10 nên cho ra câu ortherwise tiếp đến đi qua node BBB thứ hai có giá trị là 5 nên cho ra câu test=5, cuối cùng là đi qua node BBB cuối cùng có giá trị là 7 nên cho ra câu test=7.

3.10 Phần tử variable

Phần tử này dùng để khai báo một biến. Để khai báo một biến chúng ta viết theo một trong hai cách sau:

- **<xsl:variable name=”tên biến” select=”giá trị gán cho biến” />**
- **<xsl:variable name=”tên biến” >Giá trị gián cho biến</xsl:variable>**

Một biến có thể được khai báo mà không có giá trị khởi tạo

3.11 Phần tử param

Phần tử này cũng tương tự như phần tử variable là để khai báo một biến nhưng hai phần tử này có một số điểm khác nhau. Phần tử param khi chúng ta khai báo giá trị khởi gán cho nó chỉ là một giá trị default, giá trị của biến có thể được thay đổi bởi phần tử with-param (phần tử with-param dùng để gán giá trị cho biến được khai báo bởi phần tử param).

Ví dụ:

	Tài liệu XSL lưu với tên test.xsl	XML
1	<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" version = "1.0" > <xsl:output method = "text" />	<?xml version=”1.0” ?> <?xml-stylesheet type="text/xsl" href="test.xsl" ?> <AAA > <BBB>bbb </BBB> <CCC>ccc </CCC> </AAA>
2	<xsl:template match = "/" >	Kết quả hiển thị trên trình duyệt
3	<xsl:call-template name = "print" >	11 + 33 = 44
4	<xsl:with-param name = "A" >11 </xsl:with-param> <xsl:with-param name = "B" >33 </xsl:with-param>	55 + 111 = 166
5	</xsl:call-template>	
6	<xsl:call-template name = "print" >	
7	<xsl:with-param name = "A" >55 </xsl:with-param>	
8	</xsl:call-template>	
9	</xsl:template>	

```
10      <xsl:template name = "print" >
            <xsl:param name = "A" />
            <xsl:param name = "B" >111</xsl:param>
            <xsl:text ></xsl:text>
            <xsl:value-of select = "$A" />
            <xsl:text > + </xsl:text>
            <xsl:value-of select = "$B" />
            <xsl:text > = </xsl:text>
            <xsl:value-of select = "$A+$B" />
        </xsl:template>
11  </xsl:stylesheet>
```

Giải thích ví dụ:

Dòng 2: Tạo phần tử xsl:template, phần tử này có hai phần tử con là xsl:call-template

Dòng 3: Tạo phần tử xsl:call-template để triệu gọi phần tử template có tên là print, phần tử call-template có hai phần tử con xsl:param

Dòng 4: Gán giá trị cho biến A =11 và biến B=33

Dòng 7: Tương tự như dòng 3, phần tử này có một phần tử con xsl:param dùng để gán giá trị cho biến A=55

Dòng 10: Tạo phần tử xsl:template có tên là print. Phần tử này có các phần tử con thực hiện các chức năng sau:

- Khai báo biến A (không có giá trị khởi tạo)
- Khai báo biến B (với giá trị khởi tạo là 111)
- Cho ra giá trị của biến A
- Cho ra dấu '+'
- Cho ra giá trị của biến B
- Cho ra dấu '='
- Cho ra tổng của 2 biến A và B

Các bước thược hiện:

- Gọi đến phần tử template có tên là print, gán giá trị cho biến A=11, B=33 và thực hiện cộng hai biến A và B
- Gọi đến phần tử template có tên là print, gán giá trị cho biến A=55 thực hiện cộng hai biến A và B

3.12 Phần tử include

Phần tử này làm việc giống như câu lệnh include trong một số ngôn ngữ lập trình (C, PHP...), tức là phần tử này có chức năng chèn đoạn của file xsl được chỉ ra trong thuộc tình href của phần tử include vào ngay phần tử include, có nghĩa là nó thực hiện phép thế.

3.13 Phần tử import

Phần tử này làm việc cũng giống như phần tử include, nhưng chúng ta cần lưu ý là phần tử import phải là phần tử con đầu tiên của phần tử stylesheet

Ví dụ:

Tài liệu XSL lưu với tên test.xsl	xslt33.xslt
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" version = "1.0" > <xsl:import href = "xslt33.xslt" /> <xsl:output method = "text" /> <xsl:template match = "/" > <xsl:apply-templates select = "//BBB" /> </xsl:template> </xsl:stylesheet>	<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" version = "1.0" > <xsl:output method = "text" /> <xsl:template match = "BBB" > <xsl:text > BBB[</xsl:text> <xsl:value-of select = "position()" /> <xsl:text >]: </xsl:text> <xsl:value-of select = "." /> </xsl:template> </xsl:stylesheet>
XML	**Kết quả hiển thị trên trình duyệt**
<?xml version=”1.0”?> <?xml-stylesheet type="text/xsl" href="test.xsl" ?> <AAA > <BBB>cc </BBB> <BBB>ff </BBB> <BBB>aa </BBB> <BBB>fff </BBB> <BBB>FFF </BBB> <BBB>Aa </BBB> <BBB>ccCCC </BBB> </AAA>	BBB[1]: cc BBB[2]: ff BBB[3]: aa BBB[4]: fff BBB[5]: FFF BBB[6]: Aa BBB[7]: ccCCC

Chương 5
XLink và XPointer

1 XLink

1.1 XLink là gì?

Xlink (XML Linking Language) là một ngôn ngữ hỗ trợ cho liên kết tài liệu XML một cách rất tổng quát.

Siêu liên kết HTML cung cấp một số thẻ như <A>, <IMG> mới có khả năng tạo liên kết. Những liên kết này chỉ là liên kết một chiều, HTML cho phép tiến chứ không cho quay lui, tức là khi chúng ta link đến một trang nào đó thì chúng ta không thể nào đi ngược lại trang trước đó (nếu không sử dụng History của trình duyệt hay một số ngôn ngữ khác). XLink cho phép tạo liên kết đến một phần (giống như boockmark của HTML) hoặc toàn bộ tài liệu theo nhiều hình thức khác nhau. XLink cho phép liên kết một chiều hoặc nhiều chiều.

XLink cần có sự hỗ trợ của XPointer và XPath để có thể trỏ đến một cách chính xác từng vùng dữ liệu do XPointer và XPath định vị.

1.2 Cách tạo liên kết trong XLink

Không giống như HTML, XLink không quy định một phần tử liên kết nào cả, nó phụ thuộc vào thuộc tính liên kết được chỉ ra. Chúng ta cần phải định nghĩa một không gian tên cho các phần tử liên kết để trình phân tích phân biệt được đâu là XLink, khai báo không gian tên với URL: **http://www.w3.org/1999/xlink**

```
<zvon  xmlns:xlink = "http://www.w3.org/1999/xlink"
  xlink:type="simple"
  xlink:href="zvon.gif">Click here</zvon>
```

Chúng ta không nhất thiết phải lấy tiếp đầu ngữ của không gian tên XLink là xlink, chúng ta có thể dùng bất kỳ nhưng dùng tên xlink sẽ dễ phân biệt hơn.

Trong ví dụ trên chúng ta thấy có sử dụng thuộc tính **type** (xlink:type=”simple”). Đây chính la thuộc tính quy định kiểu XLink.

Có tất cả 7 kiểu XLink được định nghĩa thông qua giá trị của thuộc tính xlink:type:

Giá trị	Mô tả
simple	Liên kết đơn giản, liên kết này giống như liên kết trong HTML
extended	Liên kết mở rộng
locator	Định vị
arc	Cung liên kết
reource	Tài nguyên liên kết

Title	Tiêu đề liên kết
None	Tùy biến

1.2.1 Liên kết đơn giản (simple)

Đây chỉ là một liên kết đơn giản giống như liên lết trong HTML, kết hợp với thuộc tính xlink:type có các thuộc tính sau:

Tên thuộc tính	Giá trị	Mô tả
xlink:href	Là một địa chỉ cần link đến	Địa chỉ cần link đến
xlink:show	new, replace. embed	
xlink:actuate	onLoad, onRequest	

Khi thuộc tính xlink:show được thiết lập là new thì link này sẽ được mở ra với một cửa sổ mới. thuộc tính xlink:show có thể được kết hợp với thuộc tính xlink:actuate để làm cho liên kết đa dạng hơn:

xlink:actuate=”onLoad” thì link này sẽ tự động được gọi, tức là không cần phải click vào link này, còn nếu xlink:actuate=”onRequest” thì link sẽ không được gọi một cách tự động

```
<zvon:logo xmlns:zvon = "http://www.zvon.org"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="simple"
  xlink:href="zvon.gif"
  xlink:show="new"
  xlink:actuate="onLoad">
 </zvon:logo>
```

Khi thuộc tính xlink:show được thiết lập là replace thì nội dung của link nay sẽ được thay thế ngay trên trang hiện tại, khi kết hợp với thuộc tính xlink:actuate=”onLoad” thì link này sẽ được tự động gọi

```
<zvon:doclink xmlns:zvon = "http://www.zvon.org"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="simple"
  xlink:href="xml5_out.xml"
  xlink:show="replace"
  xlink:actuate="onLoad">  After clicking on this link  the following example will
  open in this window.
 </zvon:doclink>
```

Khi thuộc tính xlink:show được thiết lập là embed thì tài liệu link này sẽ được nhúng vào tài liệu hiện hành, chúng ta có thể hình dung nó giống như liên kết <IMG> trong HTML

```
<zvon:logo xmlns:zvon = "http://www.zvon.org"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="simple"
  xlink:href="zvon.gif"
  xlink:show="embed"
  xlink:actuate="onLoad">
 Mozilla M17 users:
  This feature is not yet implemented,
  otherwise you will see the picture here.
 </zvon:logo>
```

và thuộc tính xlink:actuate cũng được sử dụng giống như trên.

1.2.2 Liên kết mở rộng (extended)

XLink mở rộng dùng để liên kết nhiều nguồn tài liệu khác nhau từ những nguồn khác nhau. XLink mở rộng là tập các định nghĩa bao gồm quan hệ giữa tài nguyên nguồn và tài nguyên đích. Có hai loại tài nguyên liên kết mở rộng được chỉ ra bởi thuộc tính xlink:type, đó là resource (tài nguyên cục bộ) và locator (tài nguyên ở xa).

Tài nguyên cục bộ được định nghĩa trực tiếp bên trong liên kết còn tài nguyên ở xa được tham chiếu đến thông qua địa chỉ URL/URI.

```
<WEBSITE xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:type="extended">
        <NAME xlink:type="resource">Cafe au Lait</NAME>
        <HOMESITE xlink:type="locator“
    xlink:href="http://ibiblio.org/javafaq/"/>
        <MIRROR xlink:type="locator“
            xlink:href="http://sunsite.kth.se/javafaq"/>
        <MIRROR xlink:type="locator“
            xlink:href="http://sunsite.informatik.rwth-aachen.de/javafaq/"/>
        <MIRROR xlink:type="locator"
            xlink:href="http://sunsite.cnlab-switch.ch/javafaq/"/>
</WEBSITE>
```

Ví dụ trên chúng ta đã định nghĩa một WEBSITE gồm một tài nguyên cục bộ và 4 tài nguyên ở xa. Khi hiển thị trên ứng dụng hay trình duyệt thì nội dung của tài nguyên cục bộ sẽ được hiển thị và khi người dùng kích hoạt liên kết thì các địa chỉ liên kết sẽ được chọn để triệu gọi. Nhưng đây mới chỉ là cơ sở lý thuyết và chưa có trình duyệt nào hỗ trợ điều này.

Chúng ta có thể hình dung mỗi nguồn tài nguyên là một đỉnh và sự kết nối giữa một đỉnh đến một hay nhiều đỉnh khác người ta gọi là cung liên kết, mỗi đỉnh có một tên gọi được đặt bởi thuộc tính **xlink:role**. Có 3 loại cung liên kết, đó là cung kết nối, cung kết nối nhiều đỉnh, cung kết nối tổ hợp.

1.2.3 Cung liên kết

Một phần tử là cung liên kết khi thuộc tính **xlink:type** được nhận giá trị là **arc**. Bây giờ chúng ta sẽ tìm hiểu từng loại cung liên kết.

1.2.3.1 Cung kết nối

Cung kết nối là sự nối kết giữa một đỉnh tài nguyên này với một đỉnh tài nguyên khác, giữa hai đỉnh kết nối thì có một tài nguyên nguồn và một tài nguyên đích được phân biệt nhờ vào thuộc tính **xlink:from** và **xlink:to** của XLink.

```
<WEBSITE xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:type="extended">
        <NAME xlink:type="resource" xlink:role=”source”>Cafe au
    Lait</NAME>
        <HOMESITE xlink:type="locator“ xlink:href="http://ibiblio.org/javafaq/"
                        xlink:role=”ibiblio”
        />
        <MIRROR xlink:type="locator“ xlink:href="http://sunsite.kth.se/javafaq"
                      xlink:role=”sunsite-kth”
        />
        <MIRROR xlink:type="locator“
                      xlink:href="http://sunsite.informatik.rwth-aachen.de/javafaq/"
                      xlink:role=”sunsite-informatik”
         />
        <MIRROR xlink:type="locator"
                      xlink:href="http://sunsite.cnlab-switch.ch/javafaq/"
                      xlink:role=”sunsite-cnlab”
         />
        <LINK-TO >
                    xlink:type=”arc”
                    xlink:from=” source” xlink:to=” ibiblio”
                    xlink:show=”replace” xlink:actuate=”onRequest”
        </LINK-TO>
</WEBSITE>
```

1.2.3.2 Cung kết nối nhiều đỉnh

Trong trường hợp chúng ta muốn nối kết từ một đỉnh đến đòng thời nhiều đỉnh bằng cách chúng ta đặt tên cho các đỉnh muốn nối kết đến cùng một tên. Loại nối kết này gọi là cung kết nối nhiều đỉnh.

Việc xử lý các cung liên kết là do trình ứng dụng hay trình duyệt quyết định.

```
<WEBSITE xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:type="extended">
        <NAME xlink:type="resource" xlink:role=”source”>Cafe au
    Lait</NAME>
        <HOMESITE xlink:type="locator“ xlink:href="http://ibiblio.org/javafaq/"
                        xlink:role=”multi-con”
        />
        <MIRROR xlink:type="locator“ xlink:href="http://sunsite.kth.se/javafaq"
                        xlink:role=”multi-con”
        />
        <MIRROR xlink:type="locator“
                        xlink:href="http://sunsite.informatik.rwth-aachen.de/javafaq/"
                        xlink:role=”multi-con”
         />
        <MIRROR xlink:type="locator"
                        xlink:href="http://sunsite.cnlab-switch.ch/javafaq/"
                        xlink:role=”multi-con”
         />
        <LINK-TO >
                        xlink:type=”arc”
                        xlink:from=” source” xlink:to=”multi-con”
                        xlink:show=”replace” xlink:actuate=”onRequest”
        </LINK-TO>
</WEBSITE>
```

1.2.3.3 Cung kết nối tổ hợp

Nếu chúng ta không muốn chỉ ra một cung cụ thể nào thì trong định nghĩa cung chúng ta không cần sử dụng đến thuộc tính **xlink:to**, như vậy các cung liên kết là một sự tổ hợp của các đỉnh

```
    <LINK-TO >
                        xlink:type=”arc”
                        xlink:from=” source”
                        xlink:show=”replace” xlink:actuate=”onRequest”
    </LINK-TO>
```

2 XPointer(XML Pointer Language)

2.1 XPointer là gì?

Như trong chương 2 chúng ta đã tìm hiểu về XPath, XPath giúp cho chúng ta trích ra một phần tử nào đó trong tài liệu XML, XPath là một ngôn ngữ định vị nhưng nó không giúp cho chúng ta đi đi sâu vào nội dung của từng phần tử mà nó định vị được phần tử.

Ví dụ, khi chúng ta dùng cú pháp của XPath để trích ra nội dung của một phần tử B nào đó, nhưng nó không thể nào giúp cho chúng ta đi vào từng vị trí của nội dung mà nó trích lọc được .

Vì vậy sự ra đời của XPointer sẽ giúp cho chúng ta giải quyết được điều này. XPointer được xây dựng dựa trên cơ sở của XPath.

2.2 Định vị vị trí dữ liệu

XPointer định vị một vị trí dữ liệu dựa trên điểm trỏ. Có hai loại điểm trỏ đó là điểm trỏ node và điểm trỏ ký tự, vị trí điểm trỏ được bắt đầu tính từ 0.

Khi chúng ta muốn trỏ đến một phần tử trong tài liệu XML chúng ta dùng điểm trỏ node và muốn trỏ đến từng vị trí của nội dung tài liệu chúng ta dùng điểm trỏ kí tự. Các điểm trỏ dựa vào vị trí chỉ định. Nếu tập dữ liệu chúng ta chỉ định là gồm nhiều phần tử con thì chỉ số xác định vị trí điểm trỏ node còn nếu dữ liệu không chứa các phần tử thì chỉ số xác định điểm trỏ ký tự.

Chúng ta dùng hàm pointer() để định vị dữ liệu, XPointer có thể được sử dụng chung với địa chỉ URL/URI sau ký hiệu #.

Ví dụ:

```
<link xmlns:xlink="http://www.w3.org/2000/xlink" xlink:type="simple"
                xlink:href= "mydocument.xml#xpointer(//AAA/BBB[1])">
</link>
```