**MỤC LỤC**

[CHƯƠNG 1. TỔNG QUAN VỀ XML 4](#_Toc500011279)

[1.1. Giới thiệu 4](#_Toc500011280)

[1.2. Lợi ích của XML. 5](#_Toc500011281)

[1.3. Các thành phần của tài liệu XML. 9](#_Toc500011282)

[1.5. Bộ phân tích XML. 11](#_Toc500011283)

[1.7. Kết chương 15](#_Toc500011284)

[CHƯƠNG 2. TẠO TÀI LIỆU XML HỢP KHUÔNG DẠNG 17](#_Toc500011285)

[2.1. Tài liệu XML hợp khuôn dạng 17](#_Toc500011286)

[2.2. Các thành phần của tài liệu XML 17](#_Toc500011287)

[2.2.1. Khai báo XML 20](#_Toc500011288)

[2.2.2. Chỉ thị xử lý 20](#_Toc500011289)

[2.2.3. Chú thích 20](#_Toc500011290)

[2.2.4. Thẻ và các phần tử 21](#_Toc500011291)

[2.2.5. Thuộc tính 23](#_Toc500011292)

[2.3. Xây dựng cấu trúc tài liệu hợp khuôn dạng 27](#_Toc500011293)

[2.4. Phân đoạn CDATA 28](#_Toc500011294)

[2.5. Không gian tên (namespace) của XML 29](#_Toc500011295)

[2.8. Kết chương. 31](#_Toc500011296)

[CHƯƠNG 3. ĐỊNH NGHĨA KIỂU DỮ LIỆU VÀ KIỂM TRA TÍNH HỢP LỆ CỦA XML 32](#_Toc500011297)

[3.1. Định nghĩa kiểu tư liệu 32](#_Toc500011298)

[3.2. Cấu trúc của một DTD 34](#_Toc500011302)

[3.2.1. Khai báo phần tử trong DTD 34](#_Toc500011310)

[3.2.2. Khai báo thuộc tính 36](#_Toc500011311)

[3.2.3. Chú thích 44](#_Toc500011312)

[3.3. Sử dụng DTD bên trong và bên ngoài tài liệu XML. 44](#_Toc500011313)

[3.3.1. Khai báo DTD bên trong 44](#_Toc500011314)

[3.3.2. Khai báo DTD bên ngoài 45](#_Toc500011315)

[3.4. thực thể 46](#_Toc500011316)

[3.4.1. Thực thể tổng quát nội 50](#_Toc500011317)

[3.4.2. Thực thể tổng quát ngoại 52](#_Toc500011321)

[3.4.3. Thực thể tham số nội 55](#_Toc500011322)

[3.4.4. Thực thể tham số ngoại 56](#_Toc500011323)

[CHƯƠNG 4. LƯỢC ĐỒ XML 58](#_Toc500011324)

[4.1. Giới thiệu 58](#_Toc500011325)

[4.2. So sánh DTD và lược đồ XML 58](#_Toc500011327)

[4.3. Các kiểu dữ liệu 59](#_Toc500011343)

[4.3.1. Kiểu định nghĩa sẵn 59](#_Toc500011344)

[4.3.2. Kiểu đơn giản 60](#_Toc500011366)

[4.3.3. Kiểu phức hợp 62](#_Toc500011367)

[4.4. Định nghĩa phần tử 67](#_Toc500011368)

[4.5. Các phần tử thường dùng trong lược đồ. 68](#_Toc500011369)

[CHƯƠNG 5. XPATH và XSL 70](#_Toc500011370)

[5.1. Xpath 70](#_Toc500011371)

[5.1.1. Giới thiệu 70](#_Toc500011372)

[5.1.2. Đường dẫn tuyệt đối 71](#_Toc500011373)

[5.1.3. Đường dẫn tương đối 72](#_Toc500011374)

[5.1.4. Chọn các phần tử bằng ký tự đại diện 72](#_Toc500011375)

[5.1.5. Chọn các phần tử theo điều kiện 72](#_Toc500011376)

[5.1.6. Một số hàm thường dùng 73](#_Toc500011377)

[5.1.7. Một số toán tử thường dùng 74](#_Toc500011404)

[5.2. XSL (eXtensible style sheet) 75](#_Toc500011405)

[5.2.1 XSL là gì? 75](#_Toc500011406)

[5.2.2 Qui tắc chung 75](#_Toc500011407)

[5.2. 3 Một số phần tử(element) thường dùng của XSL 76](#_Toc500011408)

[5.2.4. Cách áp dụng Style Sheet vào tài liệu XML 86](#_Toc500011409)

[CHƯƠNG 6. DOM VÀ XỬ LÝ XML VỚI JAVASCRIPT 90](#_Toc500011410)

[6.1. Mô hình DOM 90](#_Toc500011411)

[6.2. Cấu trúc cây của một tài liệu XML 91](#_Toc500011412)

[6.3. Các đối tượng trong một tài liệu XML 92](#_Toc500011413)

[6.4. Tải một tài liệu XML bằng JavaScript 92](#_Toc500011414)

[6.4.1. Nạp dữ liệu XML vào trình duyệt với lớp đối tượng Microsoft.XMLDOM 93](#_Toc500011415)

[6.4.2. Sử dụng phần tử nạp dữ liệu XML 96](#_Toc500011416)

[6.4.3. Truy xuất các nút dữ liệu dựa vào tên 99](#_Toc500011417)

[6.4.4. Truy xuất giá trị thuộc tính 102](#_Toc500011418)

[TÀI LIỆU THAM KHẢO 104](#_Toc500011419)

CHƯƠNG 1. TỔNG QUAN VỀ XML

1.1. Giới thiệu

Ngôn ngữ định dạng XML (Extensible Markup Language) là ngôn ngữ được định nghĩa bởi tổ chức mạng toàn cầu W3C (World Wide Web Consortium); đây là một tổ chức quốc tế định ra các chuẩn của Web và Internet.

XML là một ngôn ngữ tổng quát dung định nghĩa dữ liệu thong qua các thẻ. Bạn có thể tùy ý định nghĩa mọi thẻ. Trong HTML các thẻ được định nghĩa và quy định trước trong khi XML thì không. Dựa vào một số quy tắc, thế là XML tự tồn tại và phát triển tự thân nó thành các ngôn ngữ định nghĩa khác.

Thực sự thì HTML và XML quan hệ rất gần gũi với nhau; cả hai đều dựa trên chuẩn ngôn ngữ định dạng tổng quát SGML(standard Generalized Markup Language).SGML là ngôn ngữ định dạng rất tổng quát, bao hàm nhiều khả năng to lớn. Bởi vì chúng ta có thể làm được rất nhiều thứ với SGML, cho nên tuy tổng quát nhưng SGML rất khó học và thực tế thì SGMLít được dùng cho một mục đích chung nào cả.. Trong khi đó, XML là tập cpn của SGML nhưng lại dễ dùng, dễ sử dụng ở mức độ tổng quát. Có thể tham khảo thêm về quan hệ của XML và SGML tại địa chỉ [www.w3.org/TR/NOTE](http://www.w3.org/TR/NOTE) - sgml-xml.

***XML trông ra sao?***

Dưới đây là ví dụ về XML chuyển tải trang HTML :

<?xml version = “1.0” encoding = “ UTF – 8”?>

<DOCUMENT>

<GREETING>

Hello From XML

</GREETING>

<MESSAGE>

Welcome to the wild and woolly world of XML

</MESSAGE>

</DOCUMENT>

Ví dụ trên thực hiện công việc như sau:

Chúng ta bắt đầu bằng thẻ khai báo chỉ thị xử lý XML

<?xml version = “1.0” encoding = “ UTF – 8”?>

Tất cả các chỉ thị xử lý XML bắt đầu bằng< ? và kết thúc bằng ?> ; theo đó ta chỉ thị rằng ngôn ngữ XML hiện tài liệu sử dụng thoe phiên bản 1.0 và mã hóa ký tự theo UTF – 8 (phiên bản thu gọn 8 – bít của mã Unicode). Tiếp đến ta tạo thẻ mới <DOCUMENT> ; tuy nhiên bạn có thể tạo bất kỳ thẻ nào không riêng gì tên DOCUMENT. Tên đặt cho thẻ có thể bắt đầu bằng ký tự gạch chân, ký số, ký tự, dấu chấm, dấu gạch nối nhưng không được có khoảng trắng. Trong XML thẻ luôn bắt đầu bằng ký tự < và kết thúc bằng ký tự >.

## *Ngôn ngữ định dạng (Markup languages)*

Ngôn ngữ định dạng là tất cả những gì dùng để mô tả nội dung một tài liệu. Đó là cách nội dung của tài liệu được diễn dịch. Ngôn ngữ định dạng mà mọi người quen nhất là HTML. HTML là ngôn ngữ cho phép tạo các trang web.

XML và HTML rất giống nhau, cả hai đều dựa trên chuẩn ngôn ngữ định dạng tổng quát SGML (Standard Geeneralized Marup Language). SGML là ngôn ngữ định dạng tổng quát bao hàm nhiều khả năng to lớn. Bởi vì SGML làm được rất nhiều thứ, cho nên tuy tổng quát nhưng khó học và thực thế thì SGML ít được dùng. Trong khi đó XML là tập con của SGML nhưng lại dễ dùng, dễ sử dụng ở mức tổng quát.

Ngôn ngữ HTML không đủ để biểu diễn các thông tin cho mục đích riêng. Ví dụ, chúng ta muốn xây dựng một mô hình xe máy và chúng ta muốn trao đổi những đặc tả về chiếc tàu với các đồng nghiệp? HTML không hề chứa thẻ định nghĩa với các tên như <SINHVIEN>, <COLOR>, ... hay những thẻ mà muốn mô tả về xe máy. Chúng ta cần thêm các thẻ để mô tả thông tin về xe máy.

1.2. Lợi ích của XML.

XML trở lên phổ biến có rất nhiều nguyên do và chúng ta xem xét các nguyên do này trong phần tổng quan dưới đây. Xml cần gì cho chúng ta hiện tại và tương lai?

Điểm quan trọng nhất là XML cho phép dễ dàng xử lý, chuyển tải và trao đổi dữ liệu giữa rất nhiều ứng dụng và tài liệu người dùng với các định dạng khác nhau

Dễ dàng trao đổi dữ liệu:Trong XML, dữ liệu và định dạng được lưu ở dạng text và có thể dễ dàng cấu hình cũng như thay đổi chúng bằng các trình soạn thảo thông thường. Nếu thích chúng ta có thể dùng một trình soạn thảo XML chuyên nghiệp để soạn thảo nội dung XML. Tuy nhiên trường hợp đơn giản hơn là chúng ta có thể dùng ngay những trình soạn thảo kiểu notepad trong Windows để chỉnh sửa file XML. Dữ liệu và các thẻ trong XML cũng không mã hóa theo một thuật giải đòi hỏi bản quyền nào cả....

Thêm vào đó XML đã được chuẩn hóa. Mọi người đều có thể hiểu và dùng được nó.

Tùy biến ngôn ngữ định dạng:Chúng ta có thể tạo ra các ngôn ngữ định dạng tùy biến dựa trên XML, đây là một trong những khả năng mạnh nhất của XML . Hàng trăm ngôn ngữ định dạng chuyên dụng dựa trên XML đã rađời như:

+ Ngôn ngữ văn phòng về kỹ thuật nghiệm vụ ngân hàng.

+ Trao đổi dữ liệu tài chính.

+ Hệ thống thanh toán qua internet của nghiệp vụ ngân hàng

+ Định dạng trao đổi viễn thông.

+ Thư viện kinh doanh thông dụng.

+ Khởi đầu XML kinh doanh điện tử.

+ Ngôn ngữ định dạng dữ liệu sản phẩm.

+ Giao thức trao đổi thông tin tài chính.

Dữ liệu tự mô tả:Dữ liệu trong tài liệu XML tự mô tả nội dung và ý nghĩa của nó. Ví dụ:

<?xml version = “1.0” encoding = “ UTF – 8”?>

<DOCUMENT>

<GREETING>

Hello From XML

</GREETING>

<MESSAGE>

Welcome to the wild and woolly world of XML

</MESSAGE>

</DOCUMENT>

Chỉ dựa vào tên thẻ được đặt cho các phần tử XML, chúng ta có thể hình daung ra nội dung và cách dữ liệu muốn thể hiện: Tài liệu này chứa thông tin là một câu chào cùng với thông điệp là một chuỗi văn bản

Dữ liệu có cấu trúc và tích hợp:Với trang tài liệu HTML nếu bạn viết sai một thẻ mà trình duyệt không hiểu, trình duyệt sẽ bỏ qua và hiển thị các thẻ hợp lệ khác. Với XML thì không. Tài liệu được kiểm tra rất nghiêm ngặt; trình duyệt không chấp nhận tài liệu XML viết sai cúa pháp và nó sẽ không phân tích tiếp tài liệu XML cho đến khi tài liệu XML hoàn toàn đúng khuôn dạng và cú pháp. Bạn phải tự bắt lỗi tài liệu thay cho trình duyệt. Bằng cách này việc xử lý tài liệu XML của trình duyệt sẽ nhanh và hiệu quả hơn.

Trình duyệt kiểm tra tài liệu XML bằng hai cách chính: Kiểm tra tính hợp khuôn dạng của tài liệu XML và kiểm tra tính hợp lệ của tài liệu XML

## Tài liệu XML hợp khuôn dạng (well-form)

Mặc dù bạn có thể đặt ra bao nhiêu thẻ cũng được, nhưng mỗi tài liệu XML cần phải theo một số qui luật để được xem là **Well-Formed**. Nếu một tài liệu XML không Well-Formed thì coi như không sử dụng đuợc, không có chương trình xử lý nào sẽ làm việc với dữ liệu bên trong file đó. Do đó một tài liệu XML cần phải theo đúng các qui luật sau đây:

* Tài liệu XML phải bắt đầu bằng câu khai báo XML: <?xml version=“1.0”?>
* Mỗi phần tử phải nằm giữa một cặp thẻ.

<GREETING>Hello from XML</GREETING>

* Nếu thẻ nào không chứa gì ở giữa thì phải chấm dứt bằng “/>“, thí dụ như <BR/> hay <HR/>.
* Một tài liệu XML phải có một thành phần duy nhất chứa tất cả các thành phần khác. Đó là gốc của cây biểu diễn tài liệu XML (ví dụ: <DOCUMENT>).
* Các cặp thẻ không được xen kẻ nhau, ví dụ:

<NAME>John Stanmore<ADDRESS>25 King Street</NAME></ADDRESS>

là bất hợp lệ vì <ADDRESS> nằm trong cặp thẻ name).

Và thêm một vài qui luật về cách dùng các mẫu tự đặc biệt. Ngoài ra các cặp thẻ phải đánh vần đúng y như nhau kể cả chữ hoa, chữ thường, (thí dụ: <STUDent> và </STUDENT> là bất hợp lệ) và tất cả giá trị các thuộc tính đều phải nằm giữa hai ngoặc kép (thí dụ: standalone=YES là bất hợp lệ, phải dùng standalone=“yes” mới được.)

## Tài liệu XML hợp lệ (valid Document)

Hầu hết các trình duyệt trước hết kiểm tra xem tài liệu XML của bạn có hợp khuôn dạng không. Tiếp theo một vài trình duyệt có thẻ kiểm tra thêm tính hợp lệ của tài liệu.

Một tài liệu XML được gọi là hợp lệ khi nó là tài liệu được kết hợp với định nghĩa kiểu tư liệu DTD (DOCUMENT TYPE DEFINITION) và tuân theo chuẩn DTD. DTD có thể được chứa trong một file tách biệt hoặc chứa ngay trong chính tài liệu. DTD sử dụng thẻ <!DOCTYPE>. Dưới đây là một ví dụ về định nghĩa kiểu tư liệu DTD được chúng ta thêm vào file tài liệu XML ở phần trên.

<?xml version=“1.0” encoding=“UTF-8”?>

<?xml-stylesheet type=“text/css” href=“greeting.css”?>

<!DOCTYPE DOCUMENT[

<!ELEMENT DOCUMENT (GREETING, MESSAGE)>

<!ELEMENT GREETING (#PCDATA)>

<!ELEMENT MESSAGE (#PCDATA)>

]>

<DOCUMENT>

<GREETING>Hello from XML</GREETING>

<MESSAGES>Well come to XML</MESSAGES>

</DOCUMENT>

Nội dung file greeting.css:

GREETING{

display: block;

font-SIZE:36pt;

color:#FF000;

text-align:center}

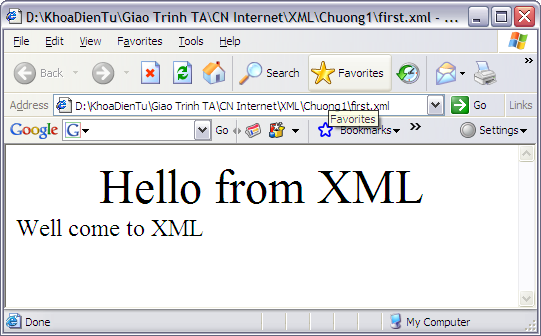
MESSAGES{

display: block;

font-SIZE:18pt;

color:#00000;}

Chúng ta thấy DTD chỉ định có hai thẻ <GREETING>, <MESSAGES> được đặt nằm trong thẻ <DOCUMENT>. Định nghĩa cho thấy <DOCUMENT> là phần tử gốc. Và <GREETING>, <MESSAGES> là các phần tử mang theo văn bản. Nội dung tài liệu greeting.xml khi hiển thị trên IE như sau:



Hình 1 Nội dung tài liệu greeting.xml khi hiển thị trên IE

1.3. Các thành phần của tài liệu XML.

Một file tài liệu XML có phần mở rộng là xml, ví dụ này ta đặt tên file là greeting.xml.

Dưới đây là ví dụ chuyển tải file HTML trong ví dụ trên:

<?xml version=“1.0” encoding=“UTF-8”?>

<DOCUMENT>

<GREETING>Hello from XML</GREETING>

<MESSAGES>Well come to XML</MESSAGES>

</DOCUMENT>

Chúng ta bắt đầu bằng thẻ khai báo chỉ thị xử lý XML

<?xml version=“1.0” encoding=“UTF-8”?>

Tất cả các chỉ thị XML bắt đầu bằng <? và kết thúc bằng ?>. Tiếp theo ta chỉ định ngôn ngữ XML hiện tài liệu sử dụng theo phiên bản 1.0 và mã hóa ký tự theo UTF-8 (phiên bản thu gọn 8-bít của bảng mã unicode).

Tiếp đến ta tạo ra thẻ mới mang tên <DOCUMENT>, có thể sử dụng bất kỳ tên gì để đặt tên thẻ. Tên đặt cho thẻ có thể bắt đầu bằng ký tự ghạch chân (\_), ký tự số, ký tự (a, b, c, …), dấu chấm, dấu ghách nối nhưng không được có khoảng trắng. Trong XML thẻ luôn bắt đầu bằng < và kết thúc bằng >.

Tài liệu XML được hình thành từ các phần tử XML. Tương tự HTML bạn tạo phần tử XML khởi đầu, ví dụ <DOCUMENT> tiếp đến là dữ liệu của các phần tử nếu có, như dữ liệu văn bản hay các phần tử XML khác cuối cùng là thẻ đóng có tên trùng với thẻ mở. Tuy nhiên thẻ đóng phải bắt đầu bằng </ (ví dụ </DOCUMENT>). Cần có thẻ đóng và thẻ mở cho toàn bộ tài liệu, phần tử này gọi là phần tử gốc. Ví dụ DOCUMENT được xem là phần tử gốc của tài liệu.

<?xml version=“1.0” encoding=“UTF-8”?>

<DOCUMENT>

...............

</DOCUMENT>

Bên trong phần tử gốc ta thêm vào thành phần thẻ khác mang tên <GREETING>. Thẻ này có chứa dữ liệu sau phần tên thẻ. Dữ liệu của thẻ là “Hello from XML”.

<?xml version=“1.0” encoding=“UTF-8”?>

<DOCUMENT>

<GREETING>Hello from XML</GREETING>

.

.

</DOCUMENT>

Kế đến ta thực hiện thêm một thẻ khác vào phần tử gốc. Thẻ mới thêm vào mang tên <MESSAGES>, tên này do ta tự đặt. Thẻ này mang theo dữ liệu thể hiện là chuỗi văn bản như sau:

<?xml version=“1.0” encoding=“UTF-8”?>

<DOCUMENT>

<GREETING>Hello from XML</GREETING>

<MESSAGES>Well come to XML</MESSAGES>

</DOCUMENT>

Cuối cùng là phần tử gốc </DOCUMENT>.

## 1.4. Tài nguyên và trình soạn thảo XML

Đặc tả về XML được định nghĩa bởi tổ chức quốc tế W3C, và đây là là nơi nên ghé thăm để biết thông tin về XML. Địa chỉ trang web của tổ chức này là: http://[www.w3c.org/XML](http://www.w3c.org/XML).

***Tìm thông tin về XML trên Web***

* http://www.xml.com – Các bài báo và tin tức về XML
* http://www.w3c.org – Tổ chức đi đầu trong tiến trình chuẩn hoá XML
* http://www.microsoft.com/xml – Web site XML của Microsoft
* http://developer.netscape.com/tech/metadata/index.html - Trung tâm phát triển meta-data và XML của Netscape
* http://www.software.ibm.com/xml – Web site XML của IBM
* http://www.xmlrepository.com – Web site sưu tập công nghệ XML, trang thử nghiệm XML và có thể tải xuống miễn phí.

Để tạo các tài liệu XML trong giáo trình này, ta chỉ sử dụng các trình soạn thảo văn bản như notepad, wordpad. Tuy nhiên có rất nhiều trình soạn thảo đặc biệt phục vụ công việc soạn thảo nội dung tài liệu XML. Với những trình soạn thảo đó cho phép tìm kiếm, tổ chức sắp xếp dữ liệu trong tài liệu XML, ... Một vài trình soạn thảo cho phép kiểm tra tính hợp lệ và hợp khuôn dạng dữ liệu. Dưới đây là một số trình soạn thảo XML nổi tiếng bạn có thể sử dụng:

* Trình soạn thảo XML Notepad của Microsoft. Phần mềm này lấy tại địa chỉ [www.microsoft.com/XML/notepad.intro.asp](http://www.microsoft.com/xml/notepad.intro.asp).

Trình soạn thảo XML Write. Phần mềm rất mạnh, cho phép soạn thảo và xem tài liệu XML tương tự như Explorer của Microsoft. Ngoài ra còn kiểm tra tính hợp khuôn dạng (well-formed) của tài liệu và kiểm tra tính hợp lệ của định nghĩa DTD và lược đồ XML. Trình soạn thảo này có thể lấy tại: [www.xmlWriter.net](http://www.XMLWriter.net)

1.5. Bộ phân tích XML.

Bộ phân tích XML là các gói phần mềm được sử dụng chúng như một phần kèm theo của ứng dụng. Dưới đây là các trình phân tích và thư viện thường dùng nhất:

*XML for Java(XML4J):* được phát triển bởi IBM AlphaWWorks. Thư viện này được sử dụng rộng rãi trong hầu hết các ví dụ cua W3C. Địa chỉ tham khảo: [www.alphaworks.ibm.com/tech/cml4j](http://www.alphaworks.ibm.com/tech/cml4j)

*Microsoft XML parser:*Trình phân tích sử dụng bởi trình duyệt Internet Explorer cài dặt thông qua kiến trúc COM của Windows. Có thể tham khảo trên:

<http://msdn.microsoft.com>

*Java Standard Extension for XML:*Đây là gói thư viện dành cho Java xây dựng bởi SunMicrosystem. Download gói này về dùng chung với các ứng dụng Java tại địa chỉ: <http://java.sun.com/products/xml>

*Python XML Parser:* Trình phân tích XML dựa vào ngôn ngữ Python chi Linux và Unix. Địa chỉ tham khảo: <http://www.python.org/topics/xml>

Bộ kiểm tra XML

Làm cách nào biết được tài liệu XML là đúng khuôn dạng và hợp lệ? Chúng ta có thể dùng bộ kiểm tra XML. Ví dụ nếu sử dụng bộ phan tích XML4J của IBM thì có thể dùng đối tượng DOMWriter để làm bộ kiểm tra. Tuy nhiên, theo cách trên phải chạy trên Java. Bộ kiểm tra tính hợp lệ của XML có thể là một chương trình Windows thông thường tương tự như các trình soạn thảo. Dưới đây là một số bộ kiểm tra XML tham khảo:

+ XML Writer, XML

+ Tydy

+ ...

1.6. Một số ứng dụng của XML

*1.6.1. Trao đổi thông tin*

Trao đổi thông tin là xuất phát điểm cho sự hình thành của công nghệ XML, là ứng dụng chính yếu nhất của XML.

Có 2 dạng trao đổi thông tin chính: Dạng 1, trao đổi thông tin nội bộ giữa các thành phần của hệ thống tin học và dạng 2, trao đổi thông tin giữa các hệ thống tin học khác nhau. Chẳng hạn như, A có nhu cầu trao đổi thông tin với B

Dạng 1:

- A,B là 2 thành phần bên trong một hệ thống tin học(giao diện hay xử lý hay lưu trữ)

- Tài liệu XML được thiết kế cho việc sử dụng nội bộ

Dạng 2:

- A là hệ thống tin học đang xem xét.

- B là hệ thống đã có trước với khả năng chuyên biệt nào đó

- A phải sử dụng tài liệu XML có cấu trúc do B đề xuất.

Ví dụ: XML có thể được sử dụng để

- Trao đổi thông tin giữa các tầng của một ứng dụng được thiết kế theo mô hình kiến trúc đa tầng.

- Trao đổi thông tin giữa một tầng với hệ thống khác bên ngoài.

Cụ thể có thể

- Sử dụng XML trao đổi thông tin giữa hệ thống lưu trữ dữ liệu ( thông thường là hệ quản trị cơ sở dữ liệu) và tầng xử lý lưu trữ dữ liệu.

- Sử dụng XML trao đổi thông tin giữa tầng dữ liệu và tầng xử lý nghiệp vụ

- Sử dụng XML trao đổi thông tin giữa tầng xử lý nghiệp vụ và tầng thể hiện.

- Sư dụng XML trao đổi thông tin giữa các tầng xử lý nghiệp vu(khi hệ thống có nhiều tầng xử lý nghiệp vụ)

*1.6.2. Lưu trữ thông tin*

Ứng dụng Xml trong việc lưu trữ thông tin. Có 3 cách ứng dụng chính của XML để lưu trữ dữ liệu bên trong hệ thống tin học.

Cách 1 : Chỉ sử dụng các tập tin XML để lưu trữ dữ liệu.

Cách 2 : Một số dữ liệu lưu trữ dưới dạng tập tin XML, một số khác lưu trữ bên trong cơ sỡ dữ liệu.

Cách 3 : Lưu trữ toán bộ bên trong cơ sở dữ liệu, tài liệu XML khi đó đuợc nhúng vào nội dung các bảng dữ liệu.

Cách 1 :

* Ưu điểm chính :

Không cần sự hổ trợ của các hệ quản trị cơ sở dữ liệu

== > Dễ cài đặt, triễn khai

* Khuyết điểm chính :

Tính hiệu quả không cao khi khối lượng dữ liệu l

* Nhận xét :

Các phần mềm trò chơi là ứng viên tốt cho ứng dụng XMl theo cách 1 Các phần mềm quản lý không thích hợp cho cách ứng dụng này

Rất thích hợp cho các ứng dụng trên các mội trường tin học không có (hoặc chưa có ) hệ quản trị cơ sỡ dữ liệu như : điện thoại di động, Máy công cụ , v.v…

Cách 2, 3 :

* Ưu điểm chính :

Có thể kết hợp tốt ưu điểm của cả 2 mô hình lưu trữ thông tin : XML , Cơ cở dữ liệu

* Khuyết điểm chính :

Cần có sự cân nhắc và quyết định đúng đắn loại thông tin nào sẽ đùng hình thức lưu trữ nào

* Nhận xét :

Cách 2 là cách sử dụng phổ biền nhất hiện nay

Cấu hình của hệ thống tin học ( phân hệ ứng dụng ) là loại thông tin thướng được chọn để lưu trữ theo dạng tài liệu XML

***1.6.3. C****ấ****u trúc d****ữ* ***li****ệ****u***

Ứng dụng Xml với các cấu trúc dữ liệu

Với mô hình DOM ( đuợc giới thiệu tóm tắt trong phần kế tiếp và chi tiết trong chương 3 ), có thể sử dụng tài liệu XMLnhư một loại cấu trúc dữ liệu động với nhiều ưu điểm

* đọc/Ghi dễ dàng

Các cấu trúc dữ liệu động như Mảng động ( Dynamic Array), Xâu (List ), Ngăn xếp ( Stack), Hàng đợi ( Queue), Cây (Tree), … có nhiều tính năng tốt trong việc biểu diễn và xử lý thông tin trong bộ nhớ chính. Tuy nhiên việc đọc/ghi thông tin của các cấu trúc dữ liệu này từ/vào bộ nhớ phụ ( thông thướng thông qua tập tin ) là không đơn giãn và thường phải thực hiện gián tiếp với một bộ đọc ghi. Tài liệu XML có thể sử dụng để cài đặt lại hấu hết các cấu trúc dữ liệu động trên ( với một số chức năng bổ sung vào DOM qua cơ chế kế thừa hay bao bọc của hướng đối tượng !!!) và đặc biệt là việc đọc/ghi rất dễ dàng ( thông thường chỉ là một lệnh gọi hàm đơn giản )

* Khả năng truy vấn cao

Việc truy vấn các thành phần/tập họp thành phần của các cấu trúc dữ liệu động phía trên thông thường phải thông qua các vòng lặp duyệt đến từng phần tử. Với tài liệu XML, có thể sử dụng ngôn ngữ truy vần Xpath để truy xuất đến thành phần/tập họp thành phần một cách rất dễ dàng ( và thông thường cũng chỉ thông qua một lệnh gọi hàm đơn giản )

## *1.6.4. Xử lý thông tin*

Ứng dụng Xml khi xử lý thông tin

Như đã trình bày trong các phần trên , công nghệ XML đuợc xếp vào loại công nghệ biểu diễn thông tin, và như thể các hướng ứng dụng chính của XML đều nhắm vào giải quyết/ cải tiến các vấn đề về biểu diễn thông tin trên các loại hình biểu diễn khác nhau.

Tuy nhiên một trong các khả năng ứng dụng khá thú vị và có nhiều hứa hẹn sẽ phát triển mạnh trong tương lại liên quan đến công nghệ xử lý thông tin với việc đề xuất một mô hình xử lý thông tin mới theo hướng đặc tả thay vì lập trình ( chi tiết về hướng ứng dụng này đuợc trình bày chi tiết trong chương cuối )

Ý tưởng xuất phát từ việc tài liệu XML cho phép biểu diễn rất tốt các văn bản có cấu trúc. Và chương trình nguồn trong các ngôn ngữ lập trình cũng là các văn bản có cấu trúc. Có nên hay không ? tạo ra một ngôn ngữ lập trình mới tương tự như ngôn ngữ lập trình hiện nay nhưng với các từ khóa là các thẻ đánh dấu ( ví dụ <for> ….</for>, <function>….<function> )

Câu trả lời rất tiếc là không, vì lập trình trên một ngôn ngữ như thế là rất khó khăn, không tự nhiên theo các thuật giải đã đề xuất.

Ý tưởng về một ngôn ngữ lập trình mới theo định chuẩn XML không thành công, nhưng nếy thay vì sử dụng ngôn ngữ lập trình mà lại sử dụng ngôn ngữ đặc tả ( chỉ mô tả mà không đi vào chi tiết thuật giải) thì kết quả có đuợc rất khả quan. Ngôn ngữ đặc tả XSLT đã ra đời trong bối cảnh như thế.)

Với XSLT có thể xây dựng một chương trình theo hướng đặc tả với các thẻ xử lý có ngữ nghĩa rất cao (mà đặc biệt là một cơ chế mới về vòng lặp) và tính dễ mang chuyễn tốt nhất có thể có ( vì đầy là ngôn ngữ XML ). XSLT còn có nhiều đặc tính thú vị khác sẽ đuợc trình bày chi tiết trong chương cuối

1.7. Kết chương

Chương một giới thiệu tổng quan về XML, những ứng dụng mà XML đạt được ở hiện tại và sẽ phát triển trong tương lai như XHTML, VML, XUL, CML, CDF, WML, SOAP,.... Các chương sau sẽ giới thiệu chi tiết về XML, cách tạo tài liệu hợp khuôn dạng, định nghĩa kiểu tư liệu và kiểm tra tính hợp lệ của XML,...

CHƯƠNG 2. TẠO TÀI LIỆU XML HỢP KHUÔNG DẠNG

2.1. Tài liệu XML hợp khuôn dạng

Trong HTML có khoảng 100 thẻ đã được định nghĩa. Trình duyệt có thể kiểm tra các thẻ này và hiển thị nội dung tài liệu trong khuôn khổ các thẻ qui định trước. Với XML hoàn toàn có thể tự do định nghĩa các thẻ nên phải có trách nhiệm với chúng. Với XML, có thể tự định nghĩa thẻ và tùy ý quyết định sử dụng thẻ. Mặc dù được tự do định nghĩa nhưng các thẻ XML cần theo một qui tắc định nghĩa bắt buộc. Nếu không theo những qui tắc này khi định nghĩa thẻ, các bộ phân tích cú pháp XML sẽ không chịu làm việc với tài liệu XML đó.

***Một tài liệu XML ràng buộc bởi hai tính chất:***

- Hợp khuôn dạng (well-form)

- Và hợp lệ (validity).

Tài liệu được xem là đúng khuôn dạng khi theo đúng các bước thiết kế và xây dựng XML do W3C đưa ra.

Các phần của tài liệu XML được gọi là các thực thể. Một thực thể XML có thể nắm giữ dữ liệu văn bản hoặc dữ liệu nhi phân nhưng không được cả hai. Một thực thể có thể tham chiếu đến thực thể khác và do đó khiến cho thực thể tham chiếu đến được đưa vào tài liệu đích. Một thực thể có thể dùng để phân tích (parsed) hoặc làm dữ liệu thông thường không dùng cho mục đích phân tích. Nói cách khác, thuật ngữ “thực thể” chỉ là cách nói chung diễn đạt đơn vị lưu trữ dữ liệu trong XML.

Ví dụ, một nội dung tài liệu với khoảng vài thẻ trong đó được xem là một thực thể. Nhưng nó sẽ không được xem là tài liệu XML cho đến khi nào bản thân nó cho thấy rằng các thẻ trong thực thể được sắp xếp theo một trật tự và cú pháp hợp khuôn dạng.

2.2. Các thành phần của tài liệu XML

Tài liệu XML được tạo thành từ thành phần định dạng và thành phần dữ liệu ký tự, dữ liệu nhị phân chỉ mới đưa vào XML trong thời gian gần đây nhưng chưa có qui định nào cho biết cách đưa dữ liệu nhi phân vào chung với định dạng ký tự.

Phần định dạng trong tài liệu giúp tạo nên cấu trúc. Định dạng bao gồm thẻ bắt đầu và thẻ kết thúc, các phần tử thẻ rỗng, các tham chiếu thực thể, tham chiếu ký tự, lời chú thích, phân đoạn CDATA, khai báo kiểu tài liệu và chỉ thị xử lý. Tất cả các dữ liệu còn lại trong tài liệu XML không phải là định dạng đều được xem là dữ liệu ký tự.

Dưới đây một ví dụ về tài liệu XML greeting.xml.

<?xml version=“1.0” encoding=“UTF-8”?>

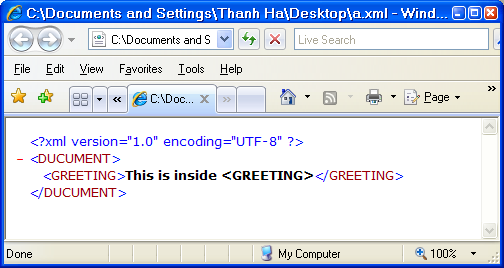
<DOCUMENT>

<GREETING>Hello from XML</GREETING>

<MESSAGES>Well come to XML</MESSAGES>

</DOCUMENT>

Thẻ hay phần tử XML bắt đầu bằng ký tự < và kết thúc bằng >, bằng qui tắc này có thể thấy trong tài liệu XML trên có các phần tử gọi là định dạng như <?xml version=“1.0” encoding=“UTF-8”?>, <DOCUMENT> hoặc </MESSAGES>, ... Còn lại chuỗi văn bản “Hello from XML”, “Well come to XML” được xem là dữ liệu ký tự.



Hình 2.1 Sử dụng thẻđịnh dạng trong XML

Mặc dù vậy phần định dạng không cần phải luôn bắt đầu bằng < và kết thúc bằng >. Trong trường hợp tham chiếu đến một thực thể thì phần định dạng có thể bắt đầu bằng & và kết thúc bằng ; (thực thể tham chiếu đến sẽ được đọc và thay thế bằng nội dung cụ thể khi trình phân tích XML diễn dịch nội dung tài liệu). Hoặc trong trường hợp tham chiếu đến một thực thể tham số phần định dạng có thể bắt đầu bằng % và kết thúc bởi ;. Một vài tham chiếu thực thể có thể chuyển sang ký tự định dạng khi trình phân tích diễn dịch (ví dụ &lt; được dịch thành < và &gt; được dịch thành >). Ví dụ:

<?xml version=“1.0” encoding=“UTF-8”?>

<DOCUMENT>

<GREETING>This is inside &lt;GREETING&gt; </GREETING>

</DOCUMENT>

Hình 2.1 thể hiện nội dung của tài liệu XML sử dụng các tham chiếu thực thể đặc biệt. Các ký tự tham chiếu giúp hiển thị và đưa vào dữ liệu ký tự những dấu hiệu đã sử dụng trong phần định dạng. Ở ví dụ trên, nếu viết “This is inside <GREETING>“, tài liệu rõ ràng không hợp khuôn dạng (vì không có thẻ </GREETING> đóng tương ứng).

<?xml version=“1.0” encoding=“UTF-8”?>

<DOCUMENT>

<GREETING>

This is inside <GREETING>

</GREETING>

</DOCUMENT>

Khoảng trắng: các ký tự như spacebar, backspace, Tab, ký tự xuống dòng, ký tự thụt vào đầu dòng được xem là khoảng trắng đối với các trình phần tích XML. Ví dụ với nội dung tài liệu XML sau:

<?xml version=“1.0” encoding=“UTF-8”?>

<DOCUMENT>

<GREETING>

Hello From XML

</GREETING>

<MESSAGES>

Wellcome to the wild and woolly world of XML

</MESSAGES>

</DOCUMENT>

Bộ phân tích không cần quan tâm đến khoảng trắng hay dấu canh lề định sẵn. Thực sự nội dung tài liệu trên hoàn toàn tương đương với nội dung tài liệu dưới đây:

<?xml version=“1.0” encoding=“UTF-8”?>

<DOCUMENT>

<GREETING>Hello From XML</GREETING>

<MESSAGES>Wellcome to the wild and woolly world of XML</MESSAGE>

</DOCUMENT>

2.2.1. Khai báo XML

Khai báo bắt đầu cho biết tài liệu được viết theo định dạng và đặc tả XML. Khai báo phải được đặt ở dòng đầu tiên của tài liệu.

*<?xml version=“1.0” encoding=“UTF-8” standalone=“yes”?>*

Trong đó:

* version : Khai báo phiên bản, khai báo này cho biết phiên bản đặc tả XML mà tài liệu sử dụng.

- encoding: Khai báo mã hóa, bộ mã sử dụng trong tài liệu XML. Mặc định là bản mã UTF-8. Có thể sử dụng mã Unicode, UCS-2, UCS-4, ... Thuộc tính này khai báo tuỳ chọn.

- standalone: Khai báo thực thể độc lập, đặt thuộc tính này là “yes” nếu tài liệu của bạn không tham chiếu đến các thực thể khác bên ngoài, nếu không đặt thuộc tính này là “no”. Thuộc tính này là tuỳ chọn.

2.2.2. Chỉ thị xử lý

Chỉ dẫn cho bộ phân tích cách xử lý tài liệu XML.Chỉ thị bắt đầu bằng <? và kết thúc bằng ?>

Ví dụ: chỉ thị yêu cầu bộ phân tích kết hợp dữ liệu của XML với bảng định kiểu XSLT như sau:

<?xml version="1.0"?>

<?xml-stylesheet type="text/xsl" href=“vd.xsl" ?>

2.2.3. Chú thích

Chú thích trong XML tương tự như chú thích trong HTML. Các chú thích để hiểu rõ hơn một phần nào đó trong cấu trúc của tài liệu XML. Khi diễn dịch nội dung tài liệu, các bộ phân tích bỏ qua những dòng chú thích.

Bắt đầu chú thích bằng chuỗi ký tự mở <!--. Nội dung chú thích được kết thúc bằng chuỗi đóng -->.

Ví dụ:

<?xml version=“1.0” encoding=“UTF-8”?>

<DOCUMENT>

<!--Đây là chú thích-->

<GREETING>

Hello From XML

</GREETING>

<MESSAGES>

Wellcome to the wild and woolly world of XML

</MESSAGES>

</DOCUMENT>

Khi đặt chú thích cần tuân theo một số qui tắc:

- Chú thích không được đặt trước các khai báo.

- Không được đặt chú thích vào bên trong phần định dạng.

- Không thể dùng chuỗi “--” bên trong chú thích.

2.2.4. Thẻ và các phần tử

Cấu trúc một tài liệu XML được tạo ra từ các thành phần định dạng. Các thành phần này bao gồm các phần tử. Mỗi phần tử thường bao gồm một cặp thẻ, thẻ bắt đầu và thẻ kết thúc. Ngoại trừ các phần tử rỗng sẽ được định nghĩa chỉ bằng một thẻ.

Thẻ bắt đầu (còn gọi là thẻ mở) bắt đầu bằng ký tự < và kết thúc bằng ký tự >. Thẻ kết thúc (còn gọi là thẻ đóng) bắt đầu bằng </ và kết thúc bằng ký tự >.

Ví dụ: Phần tử gồm: thẻ mở <DOCUMENT>, thẻ đóng </DOCUMENT>.

***Tên thẻ***

Đặc tả XML qui định rất rõ về cách đặt tên thẻ. Có thể bắt đầu tên thẻ bằng ký tự, ghạch chân (\_) hoặc dấu hai chấm (:). Các ký tự kế tiếp có thể là ký tự, ký số, ghạch chân, ghạch nối, dấu chấm, nhưng không được là khoảng trắng (nên tránh sử dụng dấu hai chấm).

Ví dụ đặt tên thẻ đúng:

<DOCUMENT>

<\_recorder123>

<\_Customer01>

.

.

</DOCUMENT>

Ví dụ đặt tên thẻ sai:

<DOCUMENT>

<123recorder123>

<.Customer01>

.

.

</DOCUMENT>

Sử dụng thẻ bắt đầu và thẻ kết thúc để tạo ra các phần tử, trong các ví dụ trên chúng ta sử dụng 3 phần tử <DOCUMENT>, <GREETING>, <MESSAGES>. Phần tử <DOCUMENT> chứa các phần tử <GREETING>, <MESSAGES>.

***Phần tử rỗng***

Một phần tử bao gồm một thẻ mở và một thẻ đóng. Giữa thẻ mở và thẻ đóng có thể chứa dữ liệu hoặc các thẻ khác. Tuy nhiên một phần tử có thể có duy nhất một thẻ. Trong XML, các thẻ chỉ đứng một mình tương trưng cho phần tử rỗng và có dấu đóng thẻ là /> Thay cho >. Ví dụ nếu chuyển dữ liệu của thẻ <GREETING> là “Hello From XML” thành thuộc tính TEXT của thẻ thì <GREETING> có thể trở thành phần tử rỗng vì không còn dữ liệu nằm giữa thẻ mở <GREETING> và thẻ đóng </GREETING>. Thẻ <GREETING> có thể được viết gọn lại như sau:

<?xml version=“1.0” encoding=“UTF-8”?>

<DOCUMENT>

<GREETING TEXT=“Hello From XML”/>

</DOCUMENT>

***Phần tử gốc***

Tất cả các tài liệu XML được coi là hợp khuôn dạng nếu chứa đựng duy nhất một phần tử gốc. Phần tử gốc này chứa đựng tất cả các phần tử và các cặp thẻ trong tài liệu. Phần tử gốc được xem là phần tử rất quan trọng trong cấu trúc XML nhất là khi chúng ta nhìn nó ở khía cạnh lập trình. Khi bạn bắt đầu phân tích nội dung tài liệu XML bạn phải khởi đầu từ một phần tử gốc, sau đó tiếp tục lần ra các phần tử chứa dữ liệu khác. Trong tài liệu order.xml trước đây, phần tử gốc chính là cặp thẻ <DOCUMENT></DOCUMENT>. Bạn có thể đặt tên bất kỳ cho phần tử gốc. Chẳng hạn order.xml có thể đổi lại tên của phần tử gốc mang ý nghĩa hơn đó là <SALES> thay cho <DOCUMENT>.

<?xml version=“1.0” standalone=“yes”?>

<SALER>

<CUSTOMER>

.

.

</CUSTOMER>

.

.

</SALER>

2.2.5. Thuộc tính

Thuộc tính của các thẻ trong tài liệu XML tương tự như thuộc tính của các thẻ HTML. Thuộc tính được kết hợp theo cặp name=value. Thuộc tính cho phép bạn xác định thêm thông tin và ý nghĩa của thẻ. Thuộc tính được đặt bên trong thẻ mở và thẻ rỗng. Để gán giá trị cho thuộc tính, thường bạn dùng dấu bằng (=). Ví dụ dưới đây chúng ta gán thuộc tính STATUS cho phần tử khách hàng <CUSTOMER> trong order.xml để theo dõi tình trạng tin dụng của người mua.

<?xml version=“1.0” standalone=“yes”?>

<SALER>

<CUSTOMER STATUS=“Good credit”>

<NAME>

<LAST\_NAME>Smith</LAST\_NAME>

<FIRST\_NAME>Sam</FIRST\_NAME>

</NAME>

<DATE>12/12/2007</DATE>

<ORDERS>

<ITEM>

<PRODUCT>Tomato</PRODUCT>

<NUMBER>5</NUMBER>

<PRICE>5</PRICE>

</ITEM>

<ITEM>

<PRODUCT>Mouse</PRODUCT>

<NUMBER>10</NUMBER>

<PRICE>2</PRICE>

</ITEM>

</ORDERS>

</CUSTOMER>

<CUSTOMER STATUS=“Lousy credit”>

<NAME>

<LAST\_NAME>Nguyen Thi</LAST\_NAME>

<FIRST\_NAME>Huong</FIRST\_NAME>

</NAME>

<DATE>12/12/2007</DATE>

<ORDERS>

<ITEM>

<PRODUCT>Cup</PRODUCT>

<NUMBER>5</NUMBER>

<PRICE>5</PRICE>

</ITEM>

<ITEM>

<PRODUCT>Pen</PRODUCT>

<NUMBER>10</NUMBER>

<PRICE>2</PRICE>

</ITEM>

<ITEM>

<PRODUCT>Mouse</PRODUCT>

<NUMBER>10</NUMBER>

<PRICE>2</PRICE>

</ITEM>

</ORDERS>

</CUSTOMER>

<CUSTOMER STATUS=“Lousy credit”>

<NAME>

<LAST\_NAME>Nguyen Tuan</LAST\_NAME>

<FIRST\_NAME>Minh</FIRST\_NAME>

</NAME>

<DATE>22/11/2007</DATE>

<ORDERS>

<ITEM>

<PRODUCT>KeyBoad</PRODUCT>

<NUMBER>5</NUMBER>

<PRICE>5</PRICE>

</ITEM>

<ITEM>

<PRODUCT>Mouse</PRODUCT>

<NUMBER>10</NUMBER>

<PRICE>2</PRICE>

</ITEM>

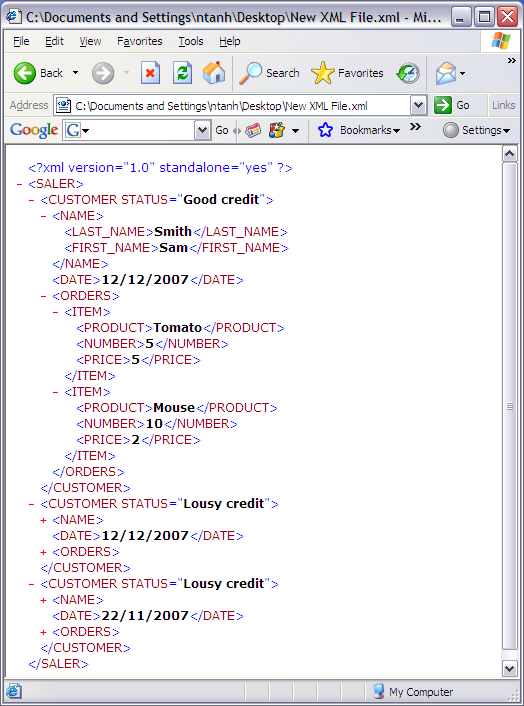
</ORDERS>

</CUSTOMER>

</SALER>

Bạn có thể thấy nội dung tài liệu XML hiển thị trong trình duyệt IE cùng với thuộc tính của thẻ như hình 2.2 .

Bộ xử lý và phân tích XML có thể đọc và lấy về giá trị của thuộc tính. Bạn sử dụng giá trị trả về của thuộc tính thẻ để phục vụ cho chương trình ứng dụng. Ví dụ như thẻ <APPLET> của HTML có rất nhiều thuộc tính. Chúng ta được dùng làm tham số truyền giá trị khởi tạo ban đầu từ môi trường ngoài vào cho Applet.



Hình 2.2 Nội dung tài liệu XML trong trình duyệt

Khi thiết lập thuộc tính cho thẻ, cần theo qui tắc đặt tên và gán giá trị cho thuộc tính như sau:

* ***Tên thuộc tính:***

Tên thuộc tính cần tuân theo qui tắc đặt tên cho thẻ. Ví dụ dưới đây cho thấy cách đặt tên thuộc tính cho thẻ hợp lệ.

<CIRCLE origin\_x=“10.0” origin\_y=“20.0” radius=“10.0”>

<image src=“image.gif”>

<BOOK page=“650”>

<pen color=“Red” width=“5”>

Tuy nhiên những thuộc tính của thẻ đặt tên như sau là không hợp lệ:

<CIRCLE 2origin\_x=“10.0” 4origin\_y=“20.0” radius=“10.0”>

<image src name=“image.gif”>

<BOOK page(some information)=“650”>

<pen color@=“Red” width=“5”>

* ***Gán giá trị cho thuộc tính:***

Phần định dạng luôn mang giá trị văn bản cho nên thuộc tính cũng mang giá trị văn bản. Ngay cả khi gán giá trị số cho thuộc tính nó cũng được biểu diễn ở dạng văn bản. Ví dụ, bạn xem số như là một chuỗi và bọc chúng trong dấu nháy kép như sau:

<CIRCLE origin\_x=“10.0” origin\_y=“20.0” radius=“10.0”>

Các giá trị được kẹp trong dấu nháy kép. Tuy nhiên khi truyền giá trị chuỗi có dấu nháy kép, chúng ta có thể sử dụng dấu nháy đơn. Ví dụ: ‘He say, “Note that” ’.

* ***Thuộc tính hữu dụng xml:lang***

Nếu sử dụng thuộc tính xml:lang, có thể tài liệu của bạn trở lên đáng giá hơn. Thuộc tính này thường được các bộ máy tìm kiếm thông tin trên mạng sử dụng như là dấu hiệu nhận dạng về ngôn ngữ của dữ liệu. Giá trị gán cho thuộc tính này thường là mã quốc gia theo chuẩn ISO 639 bao gồm hai ký tự tắt. Ví dụ để trình duyệt hiểu được nội dung đoạn văn bản thuộc ngôn ngữ Anh, bạn có thể trình bày dữ liệu bên trong thẻ <P> với thuộc tính xml:lang như sau:

<p xml:lang=“en”>Hello</P>

Bạn cũng có thể sử dụng mã quốc gia và mã vùng theo chuẩn ISO cho phân loại ngôn ngữ. Ví dụ có thể phân biệt ngôn ngữ tiếng Anh anh và Anh Mỹ như sau:

<p xml:lang=“en-GB”>Hello</P>

<p xml:lang=“en-US”>Hello</P>

2.3. Xây dựng cấu trúc tài liệu hợp khuôn dạng

Dưới đây là tổng kết qui tắc khai báo phần tử, thẻ, thuộc tính, cách đặt tên thẻ, tên thuộc tính, ... Để một tài liệu XML hợp khuôn dạng (well-formed), một tài liệu XML phải theo đúng các luật sau đây:

1. Các khai báo phải đặt ở dòng đầu tiên của tài liệu.
2. Tài liệu chỉ có một root (gốc) phần tử duy nhất, nó chứa tất cả các phần tử khác trong tài liệu.
3. Mọi phần tử khác rỗng phải bao gồm đầy đủ thẻ mở và thẻ đóng giống như nó.
4. Đóng phần tử rỗng với chuỗi đóng />.
5. Trong XML thẻ mở và thẻ đóng phải giống nhau. Ví dụ <DOCUMENT> và <DOCUMent> khác nhau. Mỗi phần tử con phải nằm trọn bên trong phần tử cha của nó.
6. Sử dụng tên thuộc tính là duy nhất. Giá trị thuộc tính trong XML phải được gói giữa một cặp ngoặc kép hay một cặp nháy đơn. Ví dụ <image name=“h.gif”>.
7. Chỉ nên sử dụng dụng 5 tham chiếu thực thể được định nghĩa trước trong XML.

**Thực thể Mô tả**

&apos; dấu (’)

&amp; dấu &

&gt; dấu lớn hơn (>)

&lt; dấu nhỏ hơn (<)

&quot; dấu ngoặc kép (“)

1. Chỉ nên sử dụng ký tự < và & đối với thẻ mở và các thực thể.

2.4. Phân đoạn CDATA

CDATA là khúc dữ liệu trong tài liệu XML nằm giữa <![CDATA[ và ]]>. Dữ liệu nằm bên trong những CDATA được cho thông qua bộ phân tích y nguyên, không bị sửa đổi. Điểm này rất quan trọng khi muốn cho vào những dữ liệu có chứa những văn bản được xem như đánh dấu. Dưới đây là các thí dụ dùng CDATA:

<![CDATA[

...place your data here...

]]>

Ví dụ:

<script>

<![CDATA[

function warning()

{

alert(“Watch out!”);

}

]]>

</SCRIPT>

2.5. Không gian tên (namespace) của XML

Không gian tên cho phép dùng cùng một tên của phần tử để nói đến hai thứ dữ liệu khác nhau trong cùng một tài liệu XML. Giống như có hai học sinh trùng tên Tuấn trong lớp học, ta phải dùng thêm họ của chúng để phân biệt, ta gọi Tuấn Trần hay Tuấn Lê. Thí dụ như có một order được đặt trong tiệm sách như 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>

Khi quan sát kỹ, ta thấy có thể có sự nhầm lẫn về cách dùng phần tử TITLE. Trong tài liệu có hai loại TITLE, một cái dùng cho khách hàng Customer nói đến danh hiệu Mr., Mrs., Dr., còn cái kia để nói đến đề tựa của một quyển sách Book.

Để tránh sự lầm lẫn, bạn có thể dùng không gian tên để nói rõ tên phần tử ấy thuộc về giòng họ nào. Giòng họ ấy là một Universal Resource Identifier (URI). Một URI có thể là một URL hay một chỗ nào định nghĩa tính cách độc đáo của nó.

Có thể khai báo không gian tên trong một phần tử bằng cách dùng thuộc tính xmlns (ns trong chữ xmlns là viết tắt cho không gian tên). Bạn cũng có thể khai báo một không gian tên mặc định để áp dụng cho những gì nằm bên trong một phần tử, nơi bạn khai báo không gian tên. Ví dụ tài liệu đặt hàng có thể được viết lại như sau:

<?xml version=“1.0”?>

<BookOrder OrderNo=“1234”>

<ORDERDATE>2001-01-01</ORDERDATE>

<Customer xmlns=“http://www.vnn.com/Customer”>

<TITLE>Mr.</TITLE>

<FirstName>Graeme</FirstName>

<LastName>Malcolm</LastName>

</CUSTOMER>

<BOOK xmlns=“http://www.vnn.com/book”>

<TITLE>Treasure Island</TITLE>

<AUTHOR>Robert Louis Stevenson</AUTHOR>

</BOOK>

</BookOrder>

Ta đã tránh được sự nhầm lẫn vì bên trong Customer thì dùng không gian tên [http://www.vnn.com/Customer](http://www.northwindtraders.com/customer) và bên trong Book thì dùng không gian tên [http://www.vnn.com/book](http://www.northwindtraders.com/book).

Tuy nhiên, ta sẽ giải quyết làm sao nếu trong order có nhiều Customer và nhiều book. Một cách giải quyết là khai báo chữ viết tắt cho các không gian tên ngay ở đầu tài liệu, trong phần tử gốc. Sau đó bên trong tài liệu ta sẽ sử dụng chữ viết tắt cho các phần tử cần xác nhận không gian tên của không gian tên nó. Thí dụ như sau:

<?xml version=“1.0”?>

<BookOrder xmlns=“http://www.vnn.com/order”

xmlns:cust=“http://www.vnn.com/Customer”

xmlns:book=“http://www.vnn.com/book” OrderNo=“1234”>

<ORDERDATE>2001-01-01</ORDERDATE>

<cust:CUSTOMER>

<cust:TITLE>Mr.</cust:TITLE>

<cust:FirstName>Graeme</cust:FirstName>

<cust:LastName>Malcolm</cust:LastName>

</cust:CUSTOMER>

<book:BOOK>

<book:TITLE>Treasure Island</book:TITLE>

<book:AUTHOR>Robert Louis Stevenson</book:AUTHOR>

</book:BOOK>

</BookOrder>

Trong tài liệu XML trên ta dùng 3 không gian tên:

* + Một không gian tên mặc định [http://www.vnn.com/order](http://www.northwindtraders.com/order)
  + Không gian tên [http://www.vnn.com/Customer](http://www.northwindtraders.com/customer) (viết tắt là cust)
  + Không gian tên [http://www.vnn.com/book](http://www.northwindtraders.com/book) (viết tắt là book).

Các phần tử và thuộc tính không có prefix (tức là không có chữ tắt đứng trước) như BookOrder, OrderNo, và OrderDate thuộc về không gian tên mặc định. Để đánh dấu một phần tử hay thuộc tính không thuộc về không gian tên mặc định, một chữ tắt, đại diện không gian tên sẽ được gắn làm prefix cho tên phần tử hay thuộc tính. Thí dụ như cust:LastName, book:TITLE.

2.8. Kết chương.

## Chương này đã nghiên cứu qua cú pháp cơ bản và cách định nghĩa các thành phần XML. Học xong chương này cần nắm vững các khái niệm chúng là nền tảng để tìm hiểu các chương tiếp theo.

CHƯƠNG 3. ĐỊNH NGHĨA KIỂU DỮ LIỆU VÀ KIỂM TRA TÍNH HỢP LỆ CỦA XML

3.1. Định nghĩa kiểu tư liệu

Ðặc tả DTD (DTD - Document Type Definition ) là bộ phận của đặc tả XML. DTD chỉ rõ những loại thẻ đánh dấu nào được sử dụng trong tài liệu XML và sự sắp xếp hợp lệ của những thẻ này. DTD cung cấp một ứng dụng không phụ thuộc vào cách chia sẻ dữ liệu. Với DTD, những nhóm người độc lập có thể dùng DTD chung để trao đổi dữ liệu. Một ứng dụng có thể dùng chuẩn DTD để kiểm tra lại dữ liệu nhận được từ bên ngoài là hợp lệ. DTD định nghĩa một cấu trúc tài liệu với một danh sách những phần tử hợp lệ. DTD có thể khai báo bên trong tài liệu XML hoặc là khai báo thành một tập tin bên ngoài tài liệu XML.

Cụ thể hơn, DTD định nghĩa các phần tử liên quan tới nhau trong một cấu trúc cây của tài liệu và nêu rõ các thuộc tính có thể được sử dụng trong các phần tử nào đó, vì vậy nó cũng bao gồm các kiểu phần tử có thể được chứa trong tài liệu. Một tài liệu XML hợp lệ là một tài liệu tuân thủ theo DTD của nó. Một DTD xuất hiện dưới dạng một tài liệu đơn giản, nó có thể được lưu trữ trong một file riêng hay là nhúng bên trong một file XML. Các tài liệu XML tham chiếu tới một DTD sẽ chứa khai báo <!DOCTYPE>, khai báo này chứa khai báo DTD hay chỉ rõ vị trí của một DTD bên ngoài.

DTD cung cấp cho các bộ phân tích cú pháp những chỉ lệnh rõ ràng về việc kiểm tra những gì khi chúng xác định tính hợp lệ của một tài liệu XML. Thành quả của DTD có thể được chia sẻ qua trang Web bằng cách định nghĩa file XML hợp lý và được lưu trữ ở những nơi khác nhau. Khi xây dựng các ứng dụng XML nên tìm kiếm một DTD đã có phù hợp với mục đích của chúng ta.

Ví dụ: tài liệu order2.xml sau đây định nghĩa kiểu tư liệu cho mọi phần tử trong tài liệu

<?xml version=“1.0” standalone=“yes”?>

<!DOCTYPE DOCUMENT[

<!ELEMENT DOCUMENT (CUSTOMER)\*>

<!ELEMENT CUSTOMER (NAME, DATE, ORDERS)>

<!ELEMENT NAME (LAST\_NAME, FIRST\_NAME)>

<!ELEMENT LAST\_NAME (#PCDATA)>

<!ELEMENT FIRST\_NAME (#PCDATA)>

<!ELEMENT DATE (#PCDATA)>

<!ELEMENT ORDERS (ITEM)\*>

<!ELEMENT PRODUCT (#PCDATA)>

<!ELEMENT NUMBER (#PCDATA)>

<!ELEMENT PRICE (#PCDATA)>

]>

<DOCUMENT>

<CUSTOMER>

<NAME>

<LAST\_NAME>Smith</LAST\_NAME>

<FIRST\_NAME>Sam</FIRST\_NAME>

</NAME>

<DATE>12/12/2007</DATE>

<ORDERS>

<ITEM>

<PRODUCT>Tomato</PRODUCT>

<NUMBER>5</NUMBER>

<PRICE>5</PRICE>

</ITEM>

<ITEM>

<PRODUCT>Mouse</PRODUCT>

<NUMBER>10</NUMBER>

<PRICE>2</PRICE>

</ITEM>

</ORDERS>

</CUSTOMER>

3.2. Cấu trúc của một DTD

Các thành phần tham gia vào viêc tạo và sử dụng 1 DTD như: Khai báo DOCTYPE chứa thông tin về vị trí của DTD, khai báo phần tử, khai báo thuộc tính, khai báo thực thể. Để khai báo các phần tử sử dụng phần tử ELEMENT, khai báo thuộc tính sử dụng phần tử ATTLIST. Khi đó cấu trúc chung của một DTD có dạng như sau:

<! DOCTYPE tên\_dtd[

<! ELEMENT tên\_phần\_tử(Kiểu\_nội\_dung\_phần\_tử)>

<! ATTLIST Tên\_phần\_tử Tên\_thuộc\_tính Kiểu\_thuộc\_tính Giá\_trị\_mặc\_định>

…….

]

DTD có thể được khai báo bên trong file XML hay có thể được lưu ở một file riêng biệt có phần mở rộng là \*.dtd

Ví dụ: **tài liệu chao.xml:**

<?xml version="1.0"?>

<!DOCTYPE DOCUMENT[

<!ELEMENT DOCUMENT (p\*)>

<!ELEMENT p (#PCDATA)>

<! ATTLIST DOCUMENT id CDATA #REQUIRED>

]>

<DOCUMENT id = “01”>

<p> chao </p>

<p> cac </p>

<p> ban </p>

</DOCUMENT

3.2.1. Khai báo phần tử trong DTD

Trong DTD, các phần tử XML được khai báo với một khai báo **ELEMENT**

Cú pháp:

<!ELEMENT tên\_phần\_tử (Kiểu-nội-dung-phần-tử)>

Ví dụ:

<!ELEMENT SV(MaSV, TenSV, Lop)>

Khai báo phần tử rỗng:

Cú pháp:

<!ELEMENT tên\_phần\_tử (EMPTY)>

Ví dụ:

<!ELEMENT SV(EMPTY)>

Khai báo phần tử chứa dữ liệu:

Cú pháp:

<!ELEMENT tên\_phần\_tử (Kiểu-nội-dung-phần-tử)>

*Kiểu-nội-dung-phần-tử có thể là:*

**#CDATA**(Character data): Phần tử chứa dữ liệu ký tự và nó không được phân tích bởi bộ phân tích cú pháp.

<!ELEMENT tên\_phần\_tử (#CDATA)>

**ANY:**Khai báo một thành phần có nội dung bất kỳ

<!ELEMENT tên\_phần\_tử (#ANY)>

**#PCDATA**(Parsed Character Data): Phần tử chứa dữ liệu được phân tích bởi bộ phân tích cú pháp, không chứa các thẻ định dạng

<!ELEMENT tên\_phần\_tử (#PCDATA)>

**Ví dụ:**

<xml version = “1.0”)

<!DOCTYPE document[

<! ELEMENT document(Sinh\_vien)\* >

<! ELEMENT Sinh\_vien (MaSV, Hoten, Lop)>

*<! ELEMENT MaSV(#PCDATA)>*

*<! ELEMENT Hoten (#PCDATA)>*

*<! ELEMENT Lop(#PCDATA)>*

]

<document>

<Sinh\_vien>

*<MaSV> DTC01</MaSV>*

*<Hoten> Nguyễn Phương Nga</Hoten>*

*<Lop>KTPM K10</Lop>*

</Sinh\_vien>

</documnet>

Khai báo phần tử có các phần tử con:

**Cú pháp:**

<!ELEMENT tên\_phần\_tử (Tên-phần-tử-con)>

Hay: <!ELEMENT tên\_phần\_tử (Danh-sách-Tên-các-phần-tử-con)>

**Ví dụ:**

<! ELEMENT Sinh\_vien (MaSV, Hoten, Lop)>

**Chú ý:**+Các phần tử con được khai báo trong một dãy và được ngăn cách nhau bởi các dấu phẩy thì chúng phải xuất hiện theo đúng thứ tự trong tài liệu.

+ Các phần tử con cũng có thể có những phần tử con của chúng

Khai báo số lần xuất hiện của các phần tử:

A,B là các phần tử trong tài liệu. Khi đó ta có các trường hợp sau:

**A\***: A xuất hiện ít nhất là 0 lần

**A+**: A xuất hiện ít nhất là 1 lần

**A?**: phần tử A hoặc không có phần tử nào cả

**A,B**: phần tử A tiếp đến phần tử B.

**A|B**: Phần tử A xuất hiện hoặc phần tử B xuất hiện nhưng không được cả hai cùng xuất hiện.

**Ví dụ:** <!ELEMENT ghichu(nguoinhan+, nguoigui, tieude, noidung\*)>

3.2.2. Khai báo thuộc tính

Thuộc tính để bổ sung thêm các thông tin cho phần tử. Để tài liệu XML hợp lệ phải định nghĩa tất cả các thuộc tính mà tài liệu sử dụng.

Cú pháp:

<!ATTLIST tên\_phần\_tử *Tên\_thuộc\_tính*  Kiểu\_thuộc\_tính “giá\_trị\_mặc\_định”>

Ví dụ:

<!ATTLIST Sinh\_vien Gioi\_tinh(Nam| Nữ) “Nam”>

Khi đó trong tài liệu XML ta sẽ có:

<Sinh\_vien Gioi\_tinh = “Nam”> </Sinh\_vien>

Hay: <Sinh\_vien Gioi\_tinh = “Nữ”> </Sinh\_vien>

***3.2.2.1. Kiểu thuộc tính***

|  |  |
| --- | --- |
| **Kiểu** | **Mô tả** |
| CDATA | Dữ liệu ký tự đơn giản không chứa định dạng |
| ENTITYES | Tham chiếu các thực thể (những thực thể này cũng phải được khao báo trong định nghĩa DTD) |
| ENTITY | Tên tham chiếu của thực thể (cũng phải khai báo trong DTD) |
| Enumerated | Đại điện cho một danh sách các giá trị mà thuộc tính có thể được gán |
| ID | Tên định danh duy nhất trong tài liệu XML |

* ***CDATA:*** Kiểu thuộc tính đơn giản nhất là CDATA. Đây là kiểu dữ liệu ký tự thuần văn bản và phi định dạng. Không thể sử dụng các ký tự đặc biệt <, “ hay & trong dữ liệu CDATA. Nếu muốn, có thể dùng các tham chiếu thực thể mà chúng ta đã học qua như: &gt;, &lt;. Định ghĩa kiểu CDATA cho thuộc tính tương tự như như định nghĩa kiểu PCDATA cho phần tử DTD.
* ***Kiểu liệt kê:*** Kiểu liệt kê không sử dụng từ khóa riêng. Khai báo kiểu lệt kê được qui định bằng một danh sách các giá trị có thể gán cho thuộc tính. Mỗi giá trị của thuộc tính trong danh sách phải phù hợp với cách đặt tên của đặc tả XML. Ví dụ dưới đây khai báo kiểu liêt kê cho thuộc tính mang tên CREDIT\_OK. Thuộc tính này chỉ cho phép mang hai giá trị “TRUE” hoặc “FALSE” và giá trị mặc định là “FALSE”.

<?xml version=“1.0” standalone=“yes”?>

<!DOCTYPE DOCUMENT[

<!ELEMENT DOCUMENT (CUSTOMER)\*>

<!ELEMENT CUSTOMER (NAME, DATE, ORDERS)>

<!ELEMENT NAME (LAST\_NAME, FIRST\_NAME)>

<!ELEMENT LAST\_NAME (#PCDATA)>

<!ELEMENT FIRST\_NAME (#PCDATA)>

<!ELEMENT DATE (#PCDATA)>

<!ELEMENT ORDERS (ITEM)\*>

<!ELEMENT ITEM (PRODUCT, NUMBER, PRICE)>

<!ELEMENT PRODUCT (#PCDATA)>

<!ELEMENT NUMBER (#PCDATA)>

<!ELEMENT PRICE (#PCDATA)>

***<!ATTLIST CUSTOMER***

***CREDIT\_OK (TRUE | FALSE) “FALSE”>***

]>

<DOCUMENT>

***<CUSTOMER CREDIT\_OK=“FALSE”>***

<NAME>

<LAST\_NAME>Smith</LAST\_NAME>

<FIRST\_NAME>Sam</FIRST\_NAME>

</NAME>

<DATE>&TODAY;</DATE>

<ORDERS>

.

.

</ORDERS>

.

.

</DOCUMENT>

* ***ID:*** Bạn có thể khai báo cho thuộc tính một kiểu khá quan trọng, đó là kiểu ID. XML xem kiểu định dạng ID với một ý nghĩa khá đặc biệt, giá trị này của thuộc tính dùng làm tên duy nhất để định danh một phần tử nào đó trong tài liệu XML. Trong ví dụ dưới đây chúng ta định nghĩa thuộc tính CUSTOMER\_ID mang kiểu dữ liệu ID và là định danh của thẻ <CUSTOMER>

<?xml version=“1.0” standalone=“yes”?>

<!DOCTYPE DOCUMENT[

<!ELEMENT DOCUMENT (CUSTOMER)\*>

<!ELEMENT CUSTOMER (NAME, DATE, ORDERS)>

<!ELEMENT NAME (LAST\_NAME, FIRST\_NAME)>

<!ELEMENT LAST\_NAME (#PCDATA)>

<!ELEMENT FIRST\_NAME (#PCDATA)>

<!ELEMENT DATE (#PCDATA)>

<!ELEMENT ORDERS (ITEM)\*>

<!ELEMENT ITEM (PRODUCT, NUMBER, PRICE)>

<!ELEMENT PRODUCT (#PCDATA)>

<!ELEMENT NUMBER (#PCDATA)>

<!ELEMENT PRICE (#PCDATA)>

***<!ATTLIST CUSTOMER***

***CUSTOMER\_ID ID #REQUIRED>***

]>

<DOCUMENT>

***<CUSTOMER CONTACT\_ID=“C12345”>***

……..

</CUSTOMER>

***<CUSTOMER CONTACT\_ID=“C12346”>***

………

</CUSTOMER>

</DOCUMENT>

Dữ liệu kiểu ID phải tuân theo qui tắc đặt tên của XML.

Lưu ý: Không thể sử dụng kiểu ID với khai báo #FIXED cho thuộc tính. Thay vào đó từ kháo #REQUUIRED thường được sử dụng nhiều hơn.

* ***IDREF:*** Kiểu thuộc tính IDREF cho phép xác định thông tin liên quan đến cấu trúc của tài liệu. Chính xác hơn là thông tin về quan hệ giữa các phần tử trong tài liệu. Thuộc tính có kiểu IDREF nắm giữ giá trị ID của các phần tử khác.

Ví dụ giả sử bạn muốn thiết lập quan hệ cha con giữa hai phần tử không nằm lồng nhau trong cấu trúc tài liệu.

<?xml version=“1.0” standalone=“yes”?>

<!DOCTYPE DOCUMENT[

<!ELEMENT DOCUMENT (CUSTOMER)\*>

………..

***<!ATTLIST CUSTOMER***

***CUSTOMER\_ID ID #REQUIRED EMPLOYER\_ID IDREF#IMPLIED>***

]>

<DOCUMENT>

***<CUSTOMER CUSTOMER\_ID=“c12345”>***

………

</CUSTOMER>

***<CUSTOMER CUSTOMER\_ID=“c12346” EMPLOYER\_ID=“c12345”>***

………

</CUSTOMER>

</DOCUMENT>

* **Kiểu thực thể ENTITY:** Thuộc tính được gán trị là tên của một thực thể đã được khai báo trước đó. Xét ví dụ sau:

<?xml version=“1.0” standalone=“no”?>

<!DOCTYPE DOCUMENT[

<!ELEMENT DOCUMENT (CUSTOMER)\*>

…….

***<!ATTLIST CUSTOMER***

***IMAGE ENTITY #IMPLIED>***

***<!ENTITY SNAPSHOT1 SYSTEM “image.gif”>***

]>

<DOCUMENT>

***<CUSTOMER IMAGE =“SNAPSHOT1”>***

……..

</CUSTOMER>

</DOCUMENT>

**Kiểu đa thực thể (ENTITIES):** cho phép gán nhiều thực thể vào thuộc tính . Các thực thể được tỏ chức thành một danh sách cách nhau bằng khoảng trắng.

***3.2.2.2. Các thiết lập cho giá\_trị\_mặc\_định***

|  |  |
| --- | --- |
| **Giá trị** | **Mô tả** |
| VALUE | Chuỗi giá trị văn bản thô bọc trong dấu nháy kép hoặc đơn, ví dụ “Hello” |
| #IMPLIED | Chỉ định rằng không có trị mặc định cho thuộc tính |
| #REQUIRED | Chỉ định rằng không có giá trị mặc định cho thuộc tính nhưng bạn cần phải gán giá trị cho thuộc tính khi sử dụng. |
| #FIX VALUE | Trong trường hợp này VALUE là giá trị bắt buộc và cố định. Thuộc tính luôn phải mạng giá trị VALUE này |

* **Giá trị tức thời** VALUE

Chúng ta có thể cung cấp giá trị mặc định cho thuộc tính tức thời ngay khi khai báo trong <!ATTLIST> như ví dụ trên.

<!ATTLIST CUSTOMER

OWES CDATA “0”

LAYAWAY CDATA “0”

DEFAULT CDATA “0”>

]>

.

.

Mặc dù vậy, có thể dùng thêm các từ khóa như #REQUIRED chẳng hạn:

* **#REQUIRED**

Khi sử dụng #REQUIRED để yêu cầu giá trị mặc định cho thuộc tính, có nghĩa là bạn không cung cấp trước một giá trị nào cả nhưng yêu cầu bất kỳ người nào sử dụng DTD liên quan đến thuộc tính thì phải gán giá trị cho thuộc tính trước khi sử dụng. Ví dụ chúng ta yêu cầu người sử dụng thẻ hay phần tử <CUSTOMER> đều phải cung cấp thuộc tính OWER cho thẻ.

<?xml version=“1.0” standalone=“yes”?>

<!DOCTYPE DOCUMENT[

<!ELEMENT DOCUMENT (CUSTOMER)\*>

<!ELEMENT CUSTOMER (NAME, DATE, ORDERS)>

<!ELEMENT NAME (LAST\_NAME, FIRST\_NAME)>

<!ELEMENT LAST\_NAME (#PCDATA)>

<!ELEMENT FIRST\_NAME (#PCDATA)>

<!ELEMENT DATE (#PCDATA)>

<!ELEMENT ORDERS (ITEM)\*>

<!ELEMENT ITEM (PRODUCT, NUMBER, PRICE)>

<!ELEMENT PRODUCT (#PCDATA)>

<!ELEMENT NUMBER (#PCDATA)>

<!ELEMENT PRICE (#PCDATA)>

***<!ATTLIST CUSTOMER***

***OWES CDATA #REQUIRED>***

]>

<DOCUMENT>

***<CUSTOMER OWES=“0”>***

<NAME>

<LAST\_NAME>Smith</LAST\_NAME>

<FIRST\_NAME>Sam</FIRST\_NAME>

</NAME>

<DATE>12/12/2007</DATE>

<ORDERS>

……

</ORDERS>

.

.

</DOCUMENT>

Nếu sử dụng thẻ <CUSTOMER> mà không có thuộc tính OWER, bộ kiểm tra sẽ thông báo lỗi là: *Required attribute ‘OWES’ is missing.*

Sử dụng thuộc tính bắtt buộc #REQUIRED rất tiện dụng trong trường hợp thông tin của thuộc tính bắt buộc hiện hữu khi xử lý tài liệu.

* **#IMPLIED**

Từ khóa #IMPLIED dùng để chỉ định là bạn không đặt sẵn giá trị mặc định cho thuộc tính nhưng cũng không bắt buộc người dùng phải gán giá trị cho thuộc tính như trong #REQUIRED. Người sử dụng có thể bỏ qua thuộc tính này. Xét ví dụ trên nhưng khai báo cho thuộc tính OWES trong thẻ <CUSTOMER> là #IMPLIED. Khi đó không phải mọi <CUSTOMER> đều cần đến OWES. Một số thẻ <CUSTOMER> có thể bỏ mà không cần sử dụng đến thuộc tính này.

<?xml version=“1.0” standalone=“yes”?>

<!DOCTYPE DOCUMENT[

<!ELEMENT DOCUMENT (CUSTOMER)\*>

<!ELEMENT CUSTOMER (NAME, DATE, ORDERS)>

……

***<!ATTLIST CUSTOMER***

***OWES CDATA #IMPLIED>***

]>

<DOCUMENT>

***<CUSTOMER OWES=“0”>***

<NAME>

<LAST\_NAME>Smith</LAST\_NAME>

<FIRST\_NAME>Sam</FIRST\_NAME>

</NAME>

<DATE>12/12/2007</DATE>

<ORDERS>

……

</ORDERS>

</CUSTOMER>

***<CUSTOMER >***

…….

</CUSTOMER>

</DOCUMENT>

* **#FIXED**

Bạn có thể chỉ định một giá trị cố định và bắt buộc cho thuộc tính bằng từ khóa #FIXED. Bằng cách này người sử dụng không thể gán một giá trị nào khác cho thuộc tính ngoài giá trị mà bạn chỉ định.

3.2.3. Chú thích

Chú thích trong DTD tương tự cú thích của XML. Bạn nên chú thích cho các phần tử khi định nghĩa DTD cho chúng như trong ví dụ:

<?xml version=“1.0” standalone=“yes”?>

<!DOCTYPE DOCUMENT[

<!—DOCUMENT là phần tử gốc-->

<!ELEMENT DOCUMENT (CUSTOMER)\*>

<!--CUSTOMER là phần tử chứa dữ liệu-

3.3. Sử dụng DTD bên trong và bên ngoài tài liệu XML.

3.3.1. Khai báo DTD bên trong

Cú pháp:

<!DOCTYPE Tên\_phần\_tử\_gốc[

Ví dụ 1:

<?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>All</TO>

<FROM>Teemu Rautanen</FROM>

<HEADING>Noi dung</HEADING>

<BODY>Noi dung</BODY>

</NOTE>

Ví dụ 2:

<?xml version="1.0"?>

<!DOCTYPE EMAIL [

<!ELEMENT EMAIL (TO+, FROM, CC\*, BCC\*, SUBJECT?, BODY?)>

<!ATTLIST EMAIL

LANGUAGE (Western|Greek|Latin|Universal) "Western"

ENCRYPTED CDATA #IMPLIED

PRIORITY (NORMAL|LOW|HIGH) "NORMAL">

<!ELEMENT TO (#PCDATA)>

<!ELEMENT FROM (#PCDATA)>

<!ELEMENT CC (#PCDATA)>

<!ELEMENT BCC (#PCDATA)>

<!ATTLIST BCC HIDDEN CDATA #FIXED "TRUE">

<!ELEMENT SUBJECT (#PCDATA)>

<!ELEMENT BODY (#PCDATA)>

]>

<EMAIL>

<TO>cdtp@yahoo.com</TO>

<FROM>tuananh@tnut.edu.vn</FROM>

<BCC HIDDEN="TRUE"></BCC>

<SUBJECT>Hoi tham suc khoe</SUBJECT>

<BODY>Cac ban co khoe khong?</BODY>

</EMAIL>

3.3.2. Khai báo DTD bên ngoài

***Định nghĩa DTD tham chiếu ngoại riêng***: Được sử dụng cho một nhóm ngừời mang tính cá nhân, không được dùng cho mục đích rộng lớn, mục đích phân phối.

Cú pháp:

<!DOCTYPE tên\_phần\_tử\_gốc SYSTEM “..URL..”>

Ví dụ: DTD được khai báo ở một tập tin bên ngoài có tên là “note.dtd”

<?xml version=“1.0”?>

<!DOCTYPE note SYSTEM “note.dtd”>

<note>

<TO>All</TO>

<FROM>Teemu Rautanen</FROM>

<HEADING>Noi dung</HEADING>

<BODY>Noi dung</BODY>

</NOTE>

Nội dung tập tin “note.dtd”:

<!ELEMENT NOTE (TO, FROM, HEADING, BODY)>

<!ELEMENT TO (#PCDATA)>

<!ELEMENT FROM (#PCDATA)>

<!ELEMENT HEADING (#PCDATA)>

<!ELEMENT BODY (#PCDATA)>

***Định nghĩa DTD tham chiếu ngoại chung***:mang tính cộng đồng.

Cú pháp:

<!DOCTYPE Ten\_the\_goc PUBLIC “URL”>

Ví dụ: file XML:vd.xml

<?xml version="1.0"?>

<!DOCTYPE DOCUMENT PUBLIC “http://www.w3schools.com/dtd/vd.dtd">

<DOCUMENT>

<p> chao </p>

<p> cac </p>

<p> ban </p>

</DOCUMENT>

3.4. thực thể

Có hai loại thực thể XML:Thực thể tổng quát(general entity) và thực thể tham số(parameter entity).

Vậy thực thể là gì?. Thực thể là cách XML tham chiếu đến một mục dữ liệu; thực thể thường là văn bản, tuy nhiên nó cũng có thể là dữ liệu nhị phân. Khai báo thực thể trong phần định nghĩa DTD sau đó sử dụng thực thể bằng cách tham chiếu nó đến trong tài liệu. Tham chiếu thực thể tổng quát bắt đầu bằng ký tự & và kết thúc bằng ký tự ; .Tham chiếu thực thể tham số bắt đầu bằng ký tự % và kết thúc bằng ký tự ;. Đối với thực thể văn bản, tham chiếu thực thể được thay thế bằng chính bản thân thực thể khi phân tích bởi bộ xử lý XML.

Nói cách khác, khai báo một thực thể trong DTD và tham chiếu đến chúng bằng tham chiếu thực thể, tham chiếu này có thể trong nội dung tài liệu đối với thực thể tổng quát hoặc trong DTD đối với thực thể tham số.

Thực thể có thể là nội(internal) hoặc ngoại(external).Một thực thể nội được định nghĩa hoàn toàn trong tài liệu tham chiếu đến nó.Thực thể ngoại nội dung của nó được định nghĩa bên ngoài file, và thường được tham chiếu đến bằng địa chỉ URL hoặc URI.

Thực thể có thể ở dạng phân tích hoặc không phân tích. Nội dung của một thực thể phân tích cần phải hợp khuôn dạng văn bản của XML. Nội dung thực thể không phân tích nắm giữ các dữ liệu thuần văn bản hoặc dữ liệu nhị phân mà bạn không muốn trình diễn dịch phân tích chúng.

Có thể định nghĩa thực thể bằng cách khai báo chúng trong DTD. Để khai báo thực thể, sử dụng phần tử chỉ thị <!ENTITY>. Khai báo và định nghĩa một thực thể tổng quát như sẽ có cú pháp như sau:

<!ENTITY NAME DEFINTION>

Ở đây NAME là tên của thực thể, DEFINTION là định nghĩa của thực thể. Tên của thực thể được dùng để tham chiếu đến nội dung của nó.

Nội dung định nghĩa của thực thể đơn giản nhất là các văn bản muốn thay thế khi thực thể được tham chiếu đến. Dưới đây là ví dụ minh họa, chúng ta định nghĩa một thực thể tổng quát mang tên TODAY nắm giữ ngày tháng là chuỗi “Ngay 12/12/2007”, phần khai báo thực thể đặt chung với các định nghĩa DTD.

<?xml version=“1.0” standalone=“yes”?>

<!DOCTYPE DOCUMENT[

<!ELEMENT DOCUMENT (CUSTOMER)\*>

<!ELEMENT CUSTOMER (NAME, DATE, ORDERS)>

<!ELEMENT NAME (LAST\_NAME, FIRST\_NAME)>

<!ELEMENT LAST\_NAME (#PCDATA)>

<!ELEMENT FIRST\_NAME (#PCDATA)>

<!ELEMENT DATE (#PCDATA)>

<!ELEMENT ORDERS (ITEM)\*>

<!ELEMENT PRODUCT (#PCDATA)>

<!ELEMENT NUMBER (#PCDATA)>

<!ELEMENT PRICE (#PCDATA)>

<!ENTITY TODAY ”Ngay 12/12/2007”>

]>

Lúc này trong tài liệu, chúng ta có thể tham chiếu đến nội dung thực thể TODAY theo dạng &TODAY; Ví dụ:

<?xml version=“1.0” standalone=“yes”?>

<!DOCTYPE DOCUMENT[

<!ELEMENT DOCUMENT (CUSTOMER)\*>

<!ELEMENT CUSTOMER (NAME, DATE, ORDERS)>

<!ELEMENT NAME (LAST\_NAME, FIRST\_NAME)>

<!ELEMENT LAST\_NAME (#PCDATA)>

<!ELEMENT FIRST\_NAME (#PCDATA)>

<!ELEMENT DATE (#PCDATA)>

<!ELEMENT ORDERS (ITEM)\*>

<!ELEMENT ITEM (PRODUCT, NUMBER, PRICE)>

<!ELEMENT PRODUCT (#PCDATA)>

<!ELEMENT NUMBER (#PCDATA)>

<!ELEMENT PRICE (#PCDATA)>

<!ENTITY TODAY “Ngay 12/12/2007”>

]>

<DOCUMENT>

<CUSTOMER>

<NAME>

<LAST\_NAME>Smith</LAST\_NAME>

<FIRST\_NAME>Sam</FIRST\_NAME>

</NAME>

<DATE>&TODAY;</DATE>

<ORDERS>

<ITEM>

<PRODUCT>Tomato</PRODUCT>

<NUMBER>5</NUMBER>

<PRICE>5</PRICE>

</ITEM>

<ITEM>

<PRODUCT>Mouse</PRODUCT>

<NUMBER>10</NUMBER>

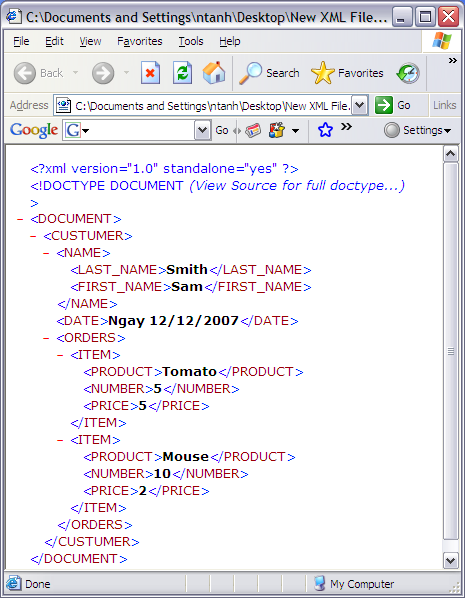
<PRICE>2</PRICE>

</ITEM>

</ORDERS>

</CUSTOMER>

</DOCUMENT>



Hình 3.4.1 Kết quả sử dụng tham chiếu thực thể tự định nghĩa.

Bên cạnh tham chiếu thực thể tổng quát, chương này cũng trình bày về tham chiếu thực thể tham số.. Thực thể này được dùng trong bản thân các định nghĩa DTD. Định nghĩa thực thể tham số có cú pháp:

<!ENTITY %NAME DEFINTION>

3.4.1. Thực thể tổng quát nội

Chúng ta xem lại ví dụ tạo thực thể tổng quát TODAY ở phần trên. Tạo thực thể tổng quát nội mang tên TODAY và tham chiếu đến thực thể này trong tài liệu bằng &TODAY:

<?xml version=“1.0” standalone=“yes”?>

<!DOCTYPE DOCUMENT[

<!ELEMENT DOCUMENT (CUSTOMER)\*>

…..

<!ELEMENT PRICE (#PCDATA)>

*<!ENTITY TODAY “Ngay 12/12/2007”>*

]>

<DOCUMENT>

<CUSTOMER>

<NAME>

…….

</NAME>

*<DATE>&TODAY;</DATE>*

………….

</DOCUMENT>

Một số lưu ý khi tạo thực thể

* Có thể tạo thực thể lồng nhau. Ví dụ:

<!ENTITY NAME=“BILL GATE”>

<!ENTITY SIGNATURE=“&NAME; Microsoft Giant”>

* Không thể định nghĩa các thực thể theo kiểu tham chiếu vòng. Ví dụ sau là không hợp lệ:

<!ENTITY NAME=“BILL GATE &SIGNATURE;”>

<!ENTITY SIGNATURE=“&NAME; Microsoft Giant”>

Trong ví dụ trên, khi trình phân tích XML muốn tham chiếu đến thực thể &NAME; nó sẽ tìm và thay thế tham chiếu thực thể &SIGNATURE; xuất hiện trong nội dung của NAME. Tuy nhiên bản thân SIGNATURE lại tham khảo ngược về NAME. Trình phân tích XML lúc này sẽ không thể nào định được nội dung cuối cùng của thực thể.

* Không được sử dụng các tham chiếu thực thể tổng quát để định nghĩa DTD. Ví dụ sau là không hợp lệ:

<!ENTITY TAGS=“NAME, DATE, ORDERS”>

<!ENTITY CUSTOMER &TAGS;”>

Tham chiếu thực thể tổng quát chỉ nên dùng trong bản thân nội dung tài liệu. Nếu muốn sử dụng tham chiếu thực thể cho định nghĩa DTD theo cách trên, chúng ta phải sử dụng tham chiếu thực thể tham số.

3.4.2. Thực thể tổng quát ngoại

Thực thể có thể định nghĩa và tham chiếu từ một nguồn bên ngoài. Chúng ta gọi đó là các thực thể ngoại. Tham chiếu đến định nghĩa của thực thể ngoại thường phải hướng dẫn cho bộ xử lý XML bằng địa chỉ URL hoặc URL. Có thể dùng tham chiếu đến thực thể ngoại để nhúng các thực thể định nghĩa sẵn vào trang tài liệu. Tham chiếu ngoại còn có thể chỉ định dữ liệu không cần phải phân tích, điều này giúp có thể sử dụng tham chiếu ngoại để trỏ đến dữ liệu dạng nhị phân sử dụng trong tài liệu.

Có thể khai báo các thực thể ngoại bằng từ khóa SYSTEM hoặc PUBLIC. Từ khóa SYSTEM dùng khai báo các thực thể dùng trong nội bộ hay tổ chức cá nhân trong khi từ khóa PULBLIC cho phép thực thể dùng chung toàn cục. Với từ khóa PUBLIC ta phải sử dụng thêm định nghĩa FPI. Khai báo thực thể ngoài bằng SYSTEM và PUBLIC có thể thực hiện như sau:

<!ENTITY NAME SYSTEM URI>

<!ENTITY NAME PUBLIC FPI URI>

Ví dụ ta muốn lưu giá trị ngày với nội dung “Ngay 12/12/2007” trong file dữ liệu date.xml. Dưới đây là cách thiết lập thực thể mang tên TODAY kết nối với phần định nghĩa trong date.xml (có thể chọn một file mở rộng khác, ví dụ .txt không nhất thiết phải là .xml).

<!ENTITY TODAY SYSTEM “date.xml”>

Tiếp theo là cách bạn có thể dùng tham chiếu đến một thực thể để chèn dữ liệu vào nội dung tài liệu (lưu ý giá trị của standalone = “yes” được chuyển sang “no” bởi vì chúng ta sử dụng thực thể ngoại).

<?xml version=“1.0” standalone=“no”?>

<!DOCTYPE DOCUMENT[

<!ELEMENT DOCUMENT (CUSTOMER)\*>

…..

<!ELEMENT PRICE (#PCDATA)>

*<!ENTITY TODAY SYSTEM “date.xml”>*

]>

<DOCUMENT>

<CUSTOMER>

<NAME>

…….

</NAME>

*<DATE>&TODAY;</DATE>*

………….

</DOCUMENT>

Cách tham chiếu này rất mạnh, nó có thể khiến tài liệu được tạo ra từ một phần của những tài liệu khác. Định nghĩa các thực thể ngoại có thể dùng chung bởi nhiều tài liệu. Ví dụ, nếu muốn dữ liệu thể hiện cho chữ ký của mình ở khắp tất cả các tài liệu, bạn chỉ cần định ghĩa thực thể này một lần, đặt thực thể chữ ký trong một file và cho tham chiếu đến thực thể từ nhiều nơi trong tài liệu.

Chú ý: Đối với các bộ phân tích chỉ kiểm tra tính hợp khuôn dạng không kiểm tra tính hợp lệ, các định nghĩa DTD có thể được bỏ qua không xử lý đến. Tuy nhiên, nếu tài liệu có tham chiếu đến một thực thể nào đó thì bộ phân tích sẽ kiểm tra phần định nghĩa của thực thể trong khai báo DTD. Vì thế chúng ta có thể chỉ xây dựng các tham chiếu thực thể mà không cần phải định nghĩa toàn bộ DTD cho các phần tử trong tài liệu.

Một trong những ứng dụng hữu ích của tham chiếu thực thể ngoại là nhúng dữ liệu từ các tài liệu khác vào tài liệu gốc. Ta hãy xét ví dụ sau:

<?xml version=“1.0” standalone=“no”?>

<!DOCTYPE DOCUMENT[

<!ELEMENT DOCUMENT (CUSTOMER)\*>

<!ELEMENT CUSTOMER (NAME, DATE, ORDERS)>

<!ELEMENT NAME (LAST\_NAME, FIRST\_NAME)>

<!ELEMENT LAST\_NAME (#PCDATA)>

<!ELEMENT FIRST\_NAME (#PCDATA)>

<!ELEMENT DATE (#PCDATA)>

<!ELEMENT ORDERS (ITEM)\*>

<!ELEMENT ITEM (PRODUCT, NUMBER, PRICE)>

<!ELEMENT PRODUCT (#PCDATA)>

<!ELEMENT NUMBER (#PCDATA)>

<!ELEMENT PRICE (#PCDATA)>

<!ENTITY data SYSTEM “data.xml”>

]>

<DOCUMENT>

&data;

</DOCUMENT>

Và nội dung file data.xml như sau

<CUSTOMER>

<NAME>

<LAST\_NAME>Smith</LAST\_NAME>

<FIRST\_NAME>Sam</FIRST\_NAME>

</NAME>

<DATE>04/9/2015</DATE>

<ORDERS>

<ITEM>

<PRODUCT>Tomato</PRODUCT>

<NUMBER>5</NUMBER>

<PRICE>5</PRICE>

</ITEM>

<ITEM>

<PRODUCT>Mouse</PRODUCT>

<NUMBER>10</NUMBER>

<PRICE>2</PRICE>

</ITEM>

</ORDERS>

</CUSTOMER>

Như vậy một tài liệu có thể là kết quả kết ghép từ nhiều phần tài liệu khác chỉ dựa vào các tham chiếu ngoại của thực thể.

3.4.3. Thực thể tham số nội

Chúng ta không thể sử dụng thực thể tham chiếu tổng quát trong bản thân các khai báo DTD. Để thật sự làm việc được với các khai báo phần tử và thuộc tính, bạn cần sử dụng những thực thể tham số. Tham chiếu đến thực thể tham số chỉ có thể sử dụng trong định nghĩa DTD. Tham chiếu đến thực thể tham số bắt đầu bằng ký tự %. Tạo thực thể tham số tương tự như tạo thực thể tổng quát ngoại trừ bạn thêm vào ký tự % trong phần chỉ thị <!ENTITY>.

<!ENTITY %NAME DEFINITION>

Có thể khai báo các thực thể tham số ngoại bằng cách thêm vào từ khóa SYSTEM hay PUBLIC cùng với định nghĩa FPI nếu cần.

<!ENTITY %NAME SYSTEM URI>

<!ENTITY %NAME PUBLIC FPI URI>

Dưới đây là ví dụ cho thấy cách sử dụng thực thể tham số khai báo nội. Ở đây ta định nghĩa thực thể mang tên BR đại điện cho chuỗi <!ELEMENT BR EMPTY> bên trong DTD.

<?xml version=“1.0” standalone=“yes”?>

<!DOCTYPE DOCUMENT[

*<!ENTITY % BR "<!ELEMENT BR EMPTY>">*

<!ELEMENT DOCUMENT (CUSTOMER)\*>

<!ELEMENT CUSTOMER (NAME, DATE, ORDERS)>

<!ELEMENT NAME (LAST\_NAME, FIRST\_NAME)>

<!ELEMENT LAST\_NAME (#PCDATA)>

<!ELEMENT FIRST\_NAME (#PCDATA)>

<!ELEMENT DATE (#PCDATA)>

<!ELEMENT ORDERS (ITEM)\*>

<!ELEMENT ITEM (PRODUCT, NUMBER, PRICE)>

<!ELEMENT PRODUCT (#PCDATA)>

<!ELEMENT NUMBER (#PCDATA)>

<!ELEMENT PRICE (#PCDATA)>

*%BR;*

]>

<DOCUMENT>

.

.

</DOCUMENT>

Bộ kiểm tra sẽ thay thế %BR bằng chuỗi nội dung thực thể mà BR được định nghĩa trong phần đầu. Tuy nhiên không thể đưa tham chiếu thực thể tham số vào khai báo DTD khác.

3.4.4. Thực thể tham số ngoại

Khi sử dụng DTD tham chiếu ngoại, các thực thể tham số có thể dùng ở bất kỳ đâu trong khai báo DTD. Trong ví dụ dưới đây chúng ta đã chuyển các khai báo DTD sang file order.dtd và thực hiện tham chiếu ngoại đến các định nghĩa phần tử trong tài liệu.

Trước hết đây là nội dung khai báo trong file order.dtd.

<!ENTITY % record "(NAME,DATE,ORDERS)">

<!ELEMENT DOCUMENT (CUSTOMER | BUYER | DISCOUNTER)\*>

<!ELEMENT CUSTOMER %record;>

<!ELEMENT BUYER %record;>

<!ELEMENT DISCOUNTER %record;>

<!ELEMENT NAME (LAST\_NAME, FIRST\_NAME)>

<!ELEMENT LAST\_NAME (#PCDATA)>

<!ELEMENT FIRST\_NAME (#PCDATA)>

<!ELEMENT DATE (#PCDATA)>

<!ELEMENT ORDERS (ITEM)\*>

<!ELEMENT ITEM (PRODUCT, NUMBER, PRICE)>

<!ELEMENT PRODUCT (#PCDATA)>

<!ELEMENT NUMBER (#PCDATA)>

<!ELEMENT PRICE (#PCDATA)>

Thêm vào định nghĩa cho phần tử <BUYER> và <DISCOUNTER>. Hai phần tử này chứa cùng nội dung như nhau <CUSTOMER>. Với lý do này, chúng ta khai báo thực thể tham chiếu %record có nội dung chung cho cả ba phần tử. Khi đưa tham chiếu ngoại của các DTD vào tài liệu, chúng hoàn toàn hợp lệ. Để sử dụng nội dung khai báo trên ta sử dụng:

<?xml version="1.0" standalone="no"?>

*<!DOCTYPE DOCUMENT SYSTEM "order.dtd">*

<DOCUMENT>

<CUSTOMER>

<NAME>

<LAST\_NAME>Nguyen Tuan</LAST\_NAME>

<FIRST\_NAME>Anh</FIRST\_NAME>

</NAME>

<DATE>12/12/2007</DATE>

<ORDERS>

<ITEM>

<PRODUCT>Keyboad</PRODUCT>

<NUMBER>5</NUMBER>

<PRICE>6</PRICE>

</ITEM>

</ORDERS>

</CUSTOMER>

</DOCUMENT>

Mục đích của việc sử dụng thực thể tham chiếu tham số là tránh các khai báo lặp lại khi định nghĩa DTD. Hơn nữa khi cần thay đổi bạn chỉ thay đổi nội dung thực thể là đủ. Các định nghĩa DTD sử dụng tham chiếu đến thực thể sẽ tự động thay đổi theo.

CHƯƠNG 4. LƯỢC ĐỒ XML

4.1. Giới thiệu

Một lược dồ định nghĩa như sau: “ Một sự biểu diễn có tính biểu đồ, một sự phác thảo hay một mô hình”. Trong ngữ cảnh phàn mềm, một lược đồ thường được hiểu một mô hình được sử dụng để mô tả cấu trúc của một cơ sở dữ liệu. Nó định nghĩa cấu trúc như bảng, trường và mối quan hệ giữa chúng. Tuy nhiên trong ngữ cảnh XML, W3C định nghĩa một lược đồ là một tập các luật để ràng buộc cấu trúc và truyền tải tập thông tin của các tài liệu XML. Một lược đồ mô tả một mô hình cho toàn bộ lớp các tài liệu, mô tả cách đánh dấu dữ liệu và cũng chỉ rõ sự sắp xếp có thể của các thẻ và văn bản trong một tài liệu hợp lệ. Một lược đồ có thể được xem như một bộ từ vựng chung để trao đổi tài liệu giữa những tooe chức khác nhau.

4.2. So sánh DTD và lược đồ XML

XML kế thừa khái niệm của DTD từ SGML. Trong khi DTD được sử dụng để định nghĩa các mô hình nội dung, thứ tự hợp lệ các thành phần lồng nhau, ở một mức độ nào đó đã định nghĩa được các kiểu dữ liệu của thuộc tính thì chúng có những hạn chế nhất định sau:

* DTD đượcviết không theo cú pháp XML.
* DTD không thể mở rộng được.
* Chúng không hỗ trợ không gian tên.
* Chúng chỉ cung cấp định kiểu dữ liệu rất hạn chê.

Các lược đồ vượt qua được những nhược điểm này và cho phép các ứng dụng Web trao đổi dữ liệu XML nhanh chóng hơn mà không dựa vào những công cụ đánh giá hợp lệ đặc biệt.

Lược đồ XML cung cấp một loạt các tính năng mới sau:

* Sử dụng cú pháp XML.
* Các lược đồ XML có thể được dịch và kiểm soát giống như bất kỳ tài liệu XML nào.
* Có nhiều kiểu dữ liệu hơn.
* Cho phép tự định nghĩa các kiểu dữ liệu từ các kiểu nguyên mẫu.
* Nhóm các thuộc tính .
* Các nguyên mẫu có thể tinh chỉnh được.
* Lược đồ XML cho phép thêm hai khả năng: mở và tinh chỉnh được.

Trong một mô hình nội dung mở, các phần tử khác với các phần tử bắt buộc có thể được thêm vào. Mô hình nội dung mở cho phép thêm vào các phần tử con và các thuộc tính trong một phần tử không được khai báo trong lược đồ của tài liệu.

4.3. Các kiểu dữ liệu

XML Schema có 3 loại kiểu chính :

- Loại 1 : Kiểu định nghĩa sẵn ( BultinType)

- Loại 2 : Kiểu đơn giản (simpleType)

- Loại 3 : Kiểu phức hợp (complexType).

Tùy thuộc vào loại thẻ cần mô tả loại kiểu tương ứng sữ được sử dụng

4.3.1. Kiểu định nghĩa sẵn

* Khái niệm: là các kiểu được xây dựng, định nghĩa sẵn trong XML Schema. Các kiểu này tương tự như các kiểu cơ sở trong ngôn ngữ lập trình.
* Một số kiểu cơ sở thông dụng:

|  |  |
| --- | --- |
| Tên kiểu | Mô tả |
| binary | Kiểu dữ liệu nhị phần |
| boolean | Kiểu logic |
| byte | Kiểu byte |
| century | Kiểu thế kỷ |
| date | Kiểu ngày |
| decimal | Kiểu thập phân |
| double | Kiểu số thực |
| ENTITY | Kiểu thực thể |
| ENTITIES | Kiểu đa thực thể |
| ID | Kiểu định danh |
| int, integer | Kiểu số nguyên |
| IDREF | Kiểu tham chiếu định danh |
| NOTATION | Kiểu ghi chú |
| NMTOKEN | Kiểu Token đơn |
| NMTOKENS | Kiểu đa Token |
| month | kiểu tháng |
| string | Kiểu chuỗi |

* **Ý nghĩa sử dụng:**

Được sử dụng để mô tả trực tiếp kiểu của các thuộc tính hay của thẻ thỏa mãn 2 điều kiện:

+ Thẻ không chứa thuộc tính

+ Không chứa thẻ khác(nội dung là chuỗi văn bản) và có miền giá trị( tập hợp giá trị có thể có) thích hợp với kiểu.

* **Cú pháp :**

Khi dùng với thẻ:

*<xs:element name=”Ten\_the” type=”Ten\_kieu\_co\_so” ... />*

Khi dùng với thuộc tính:

*<xs:attribute name=”Ten\_thuoc\_tinh” type=”Ten\_kieu\_co\_so” .. />*

* **Ví dụ:**

*<xs:element name="Ho\_ten" type="xs:string" />*

Thẻ Ho\_ten không có thuộc tính, không chứa thẻ con và có nội dung là chuỗi văn bản

*<xs:element name="Ngay\_sinh" type="xs:date" />*

Thẻ Ngay\_sinh không có thuộc tính, không chứa thẻ con và có nội dung tương ứng một ngày

*<xs:attribute name="He\_so" type="xs:float"/>*

Thuộc tính He\_so phải là số thực

*<xs:attribute name="x" type="xs:int"/>*

Thuộc tính x phải là số nguyên

*<xs:attribute name="f" type="xs:boolean"/>*

Thuộc tính f phải là giá trị logic

4.3.2. Kiểu đơn giản

* **Khái niệm** : Là các kiểu do người dùng định nghĩa dựa trên các kiểu cơ sở có sẵn trong XML Schema.
* **Ý nghĩa sử dụng :** Được sử dụng để mô tả trực tiếp kiểu của các thuộc tính hay các thẻ thỏa mãn 2 điều kiện :

Điều kiện 1 : Không có thuộc tính

Điều kiện 2 : Không chứa thẻ khác(nội dung là chuỗi văn bản) và có miền giá trị( tập hợp giá trị có thể có) thích hợp với kiểu.

* **Cú pháp:** ( dạng đơn giản và thông dụng )

*<xs:simpleType name="Ten\_kieu">*

*<xs:restriction base="Ten\_kieu\_co\_so">*

*<! --Giới hạn ( ràng buộc ) dữ liệu -->*

*</xs:restriction>*

*</xs:simpleType>*

*Ten\_kieu* : Tên của kiểu đơn giản

*Ten\_kieu\_co\_so* : Tên của kiểu cơ sở tương ứng

*Giới hạn ( ràng buộc ) dữ liệu* : Một ràng buộc định nghĩa những gì có thể xuất hiện trong tài liệu. Có 2 loại ràng buộc trong các lược đồ XML:

+ Các ràng buộc về mô hình nội dung mô tả thứ tự và dãy các phần tử.

+ Các ràng buộc về kiểu dữ liệu mô tả các kiểu dữ liệu hợp lệ. Ràng buộc này cho phép kiểm soát phạm vi và định dạng của dữ liệu có thể được lưu trữ trong một phần tử hay thuộc tính biết trước.

Để ràng buộc về kiểu dữ liệu cho phần tử ta sử dụng phần tử restriction.

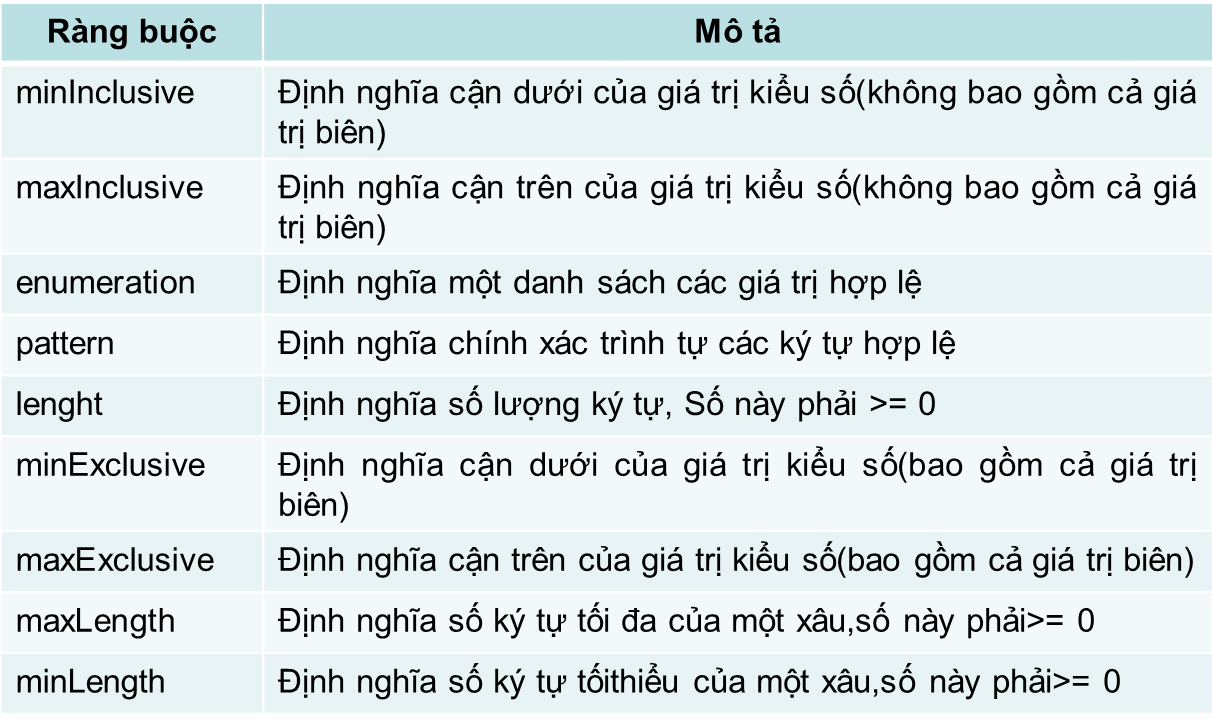
Cú pháp:

<xs: restriction base = “ tên kiểu cơ sở”

<!- -Các ràn buộc -->

</ xs: restriction>

Các ràng buộc bao gồm:



* **Ví dụ:**
* ***Restriction trên giá trị:*** sử dụng các ràng buộc: **minInclusive, maxInclusive, minExclusive, maxExclusive**
* **Ví dụ:** muốn định nghĩa một phần tử tên là AGE và giá trị của nó chỉ nằm từ 0 đến 120.

<xs:element name="age">

<xs:simpleType>

<xs:restriction base="xs:integer">

<xs:*minInclusive* value="0"/>

<xs:*maxInclusive* value="120"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

4.3.3. Kiểu phức hợp

* **Khái niệm**: Là các kiểu do người dùng tựđịnh nghĩa cho phép mô tả nội dung và các thuộc tính của các thẻ được khai báo thuộc về kiểu đang xét.
* **Ý nghĩa sử dụng**: được sử dụng để mô tả kiểu của các thẻ thỏa mãn một trong hai điều kiện:

Điều kiện 1 : Có thuộc tính

Điều kiện 2 : Có chứa thẻ khác.

* **Dạng khai báo chung các kiểu phức hợp như sau:**

*<xs:complexType name="Ten\_kieu">*

*Dac\_ta\_cau\_truc\_noi\_dung*

*Dac\_ta\_thuoc\_tinh*

*</xs:complexType>*

**Trong đó:**

***Dac\_ta\_cau\_truc\_noi\_dung:*** XML schema cho phép mô tả cách thức tổ chức, sắp xếp các thành phần bên trong thẻ qua 3 dạng cơ sở:

+ Dạng tuần tự: mô tả thứ tự xuất hiện tuần tự các thành phần

+ Dạng tùy chọn: Mô tả việc phải sử dụng một thành phần nào đó trong tập hợp các thành phần cho trước.

+ Dạng lặp: Mô tả việc cho phép lặp lại của các thành phần

***Dac\_ta\_thuoc\_tinh:*** Cho phép mô tả hệ thống các thuộc tính của một thẻ.

***4.3.3.1. Đặc tả cấu trúc nội dung***

***1. Dạng tuần tự : Sử dụng thẻ/từ khóa sequence***

* **Cú pháp :**

*<xs:complexType name="Ten\_kieu">*

*<xs:sequence>*

*Thanh\_phan\_1*

*Thanh\_phan\_2*

*....*

*Thanh\_phan\_k*

*</xs:sequence>*

*....*

*</xs:complexType>*

* **Ý nghĩa** *:*Các thành phần Thanh\_phan\_1, Thanh\_phan\_2, …Thanh\_phan\_k phải xuất hiện duy nhât và đúng theo thứ tự trên trong thẻ tương ứng
* **Ví dụ :**

*<xs:complexType name="DIEM">*

*<xs:sequence>*

*<xs:element name="x” type="xs:float" />*

*<xs:element name="y” type="xs:float" />*

*</xs:sequence>*

*</xs:complexType>*

**2.Dạng tùy chọn** : Sử dụng thẻ/từ khóa choice

* **Cú pháp :**

<xs:complexType name="Ten\_kieu">

<xs:choice>

Thanh\_phan\_1

Thanh\_phan\_2

....

Thanh\_phan\_k

</xs:choice>

....

</xs:complexType>

* **Ý nghĩa:**Thẻ có kiểu Ten\_kieu phải sử dụng một thành phần trong số các thành phần Thanh\_phan\_1,Thanh\_phan\_2, ... Thanh\_phan\_k
* **Ví dụ:**

<xs:complexType name="X">

<xs:choice>

<xs:element name="A” type="A" />

<xs:element name="B” type="xs:string" />

</xs:choice>

</xs:complexType>

Các thẻ có khai báo kiểu X phải bao hàm bên trong một trong 2 thẻ con sau:Thẻ có tên A và có kiểu A ( cho phép tên kiểu và tên thẻ trùng nhau ).Thẻ có tên B và có kiểu là chuỗi.

**3**. **Dạng lặp**: Sử dụng thuộc tính/từ khóa minOccurs , maxOccurs

* **Cú pháp:**

<xs:complexType name="Ten\_kieu">

<xs:sequence>

...

<xs:element name="Ten\_the\_con” type="Kieu\_the\_con"

minOccurs=”So\_lan\_lap\_toi\_thieu”

maxOccurs=”So\_lan\_lap\_toi\_da” />

...

</xs:sequence>

....

</xs:complexType>

* **Ý nghĩa:**Thẻ có kiểu Ten\_kieu có chứa bên trong thẻ con có tên Ten\_the\_con với số lần lặp tối thiểu là So\_lan\_lap\_toi\_thieu và số lần lặp tối đa là So\_lan\_lap\_toi\_da.
* **Ví dụ:**

<xs:complexType name="DA\_THUC">

<xs:sequence>

<xs:element name="DON\_THUC” type="DON\_THUC"

minOccurs=”1” />

</xs:sequence>

<-- Mô t các thuoc tính -->

...

</xs:complexType>

<xs:complexType name="HOA\_DON">

<xs:sequence>

<xs:element name="CT\_HOA\_DON” type="CT\_HOA\_DON"

minOccurs=”1”

maxOccurs=”10” />

</xs:sequence>

<-- Mô t các thuoc tính -->

...

</xs:complexType>

***4.3.3.2. Đặc tả thuộc tính***

Cho phép mô tả hệ thống các thuộc tính của một thẻ

* **Cú pháp :**

<xs:complexType name="Ten\_kieu">

Đặc tả cấu trúc nội dung

....

*<xs:attribute name="Ten\_thuoc\_tinh" type="Kieu\_thuoc\_tinh"*

*Tinh\_chat\_thuoc\_tinh />*

....

</xs:complexType>

Trong đó

+ Ten\_thuoc\_tinh : tên của thuộc tính của kiểu đang xét, không cho phép 2 thuoc tính có cùng tên

+ Kieu\_thuoc\_tinh : Tên của kiểu cơ sở hay kiểu đơn giản

+ Tinh\_chat\_thuoc\_tinh : Mô tả một số tính chất của thuộc tính. XML Shema cho phép mô tả rất nhiều loại tính chất khác nhau, mỗi tính chất tương ứng với một từ khóa riêng

*<xs:attribute name="Ten\_thuoc\_tinh" type="Kieu\_thuoc\_tinh"*

*Tu\_khoa\_1=”Gia\_tri\_1”*

*Tu\_khoa\_2=”Gia\_tri\_2”*

*..*

*Tu\_khoa\_k=”Gia\_tri\_k” />*

* **Ví dụ:**

<xs:attribute name="Tu\_so" type="SO\_NGUYEN\_DUONG"

default=”1” />

<xs:attribute name="Bien\_so" type="xs:string"

fixed=”x” />

<xs:attribute name="Ten\_don\_thuc" type="xs:string"

use=”optional” />

* **Một số tính chất thông dụng**

|  |  |
| --- | --- |
| **Tính chất** | **Mô tả** |
| default | Giá trị không được gán cho thuộc tính thì thuộc tính sẽ nhận giá trị ngầm định được xác định trong lược đồ. |
| fixed | Giá trị gán cho thuộc tính là cố định và không được thay đổi |
| required | Thuộc tính phải được hiện hữu trong thẻ và phải có giá trị |
| optional | Thuộc tính có thể có, có thể không và giá trị gán cho thuộc tính là không bắt buộc |
| prohibited | Không cho phép thuộc tính xuất hiện |

Dùng từ khóa use trong khai báo<xs:element>để chị định ràng buộc cho thuộc tính theo các tùy chọn nêu trên .

Ví dụ, trong định nghĩa lược đồ dưới đây thuộc tính phone trong kiểu address là tùy ý (optional) trong khi thuộc tính name là bắt buộc (required) còn city có giá trị mặc định (default) là HCM.

<xsd:complexType name=“address”>

<xsd:element name=“name” type=“xsd:string” use=“required”/>

<xsd:element name=“street” type=“xsd:string”/>

<xsd:element name=“city” type=“xsd:string” use=“default” value=“HCM”/>

<xsd:element name=“state” type=“xsd:string”/>

<xsd:attribute name=“phone” type=“xsd:string” use=“optional”/>

</xsd:complexType>

4.4. Định nghĩa phần tử

Với XML Schema, đặc tả cấu trúc tài liệu XML tập trung vào việc đặc tả các kiểu, đặc tả các thẻ trong XML Schema rất đơn giản và chỉ nhằm vào mục tiêu chính là xác định kiểu sẽđược sử dụng của thẻ.

Các thông tin cần mô tả khi đặc tả một thẻ trong XML bao gồm:

- Tên thẻ

- Kiểu của thẻ

- Một số tính chất khác của thẻ

**Dạng khai báo chung như sau**

<xs:element name=”Ten\_the” type=”Ten\_kieu” Thuoc\_tinh\_khac />

Trong đó:

*Ten\_the* :Tên của thẻ đang xét và tuân theo cách đặt tên của định chuẩn XML

*Ten\_kieu* :Tên của kiểu tương ứng mô tả thông tin về thẻ.

*Thuoc\_tinh\_khac*: Có nhiều loại thuộc tính khác nhau cho phép mô tả các tính chất của thẻ mà trong đó thông dụng nhât là 2 thuộc tính minOccurs, maxOccurs .

Khi đặc tả thẻ vấn đề quan trọng nhất là xác định loại kiểu sẽ dùng trong thẻ. Tùy thuộc vào loại thẻ loại kiểu tương ứng sẽ được dùng.

**Phân loại thẻ:** Có rất nhiều cách phân loại thẻ, mỗi cách phục vụ cho mục tiêu khác nhau. Với mục tiêu phân loại nhằm xác định loại kiểu tương ứng được dùng, hệ thống các thẻ trong tài liệu có thể được phân loại như sau:

Thẻ bao gồm 2 nhóm chính:

+ Nhóm 1: Nhóm các thẻ có thuộc tính: nhất thiết phải sử dụng kiểu phức hợp(Khai báo kiểu phức hợp Y,sử dụng Y là kiểu của thẻ).

+ Nhóm 2: Nhóm các thẻ không có thuộc tính: bao gồm 2 nhóm

Nhóm 2.1: Nhóm các thẻ không có thuộc tính và có chứa các thẻ con bên trong: Phải sử dụng kiểu phức hợp

Nhóm 2.2: Nhóm các thẻ không có thuộc tính và không chứa các thẻ con bên trong: có thể chọn kiểu cơ sở hay kiểu đơn giản phụ thuộc vào miền giá trị của chuỗi văn bản bên trong thẻ.

4.5. Các phần tử thường dùng trong lược đồ.

**Phần tử schema**

schema là phần tử gốc của một tài liệu lược đồ XML. Phần tử schema cung cấp 2 thuộc tính là:

**+ id:** Thuộc tính định danh cho lược đồ

**+ xmlns:** Thuộc tính thiết lập không gian tên cho lược đồ

* ***Cấu trúc chung của một tài liệu lược đồ XML như sau:***

*<xs:schema id = “dinh\_danh\_của\_lược\_đồ”*

*xmlns:xs=*[*“http://www.w3.org/2001/XMLSchema*](http://www.w3.org/2001/XMLSchema)*”>*

*<!- - Phần thân của lược đồ - ->*

*</xs:schema>*

Phần tử schema trong lược đồ chứa các phần tử con thông dụng sau:

|  |  |
| --- | --- |
| all | Cho phép tất cả các phần tử trong nhóm nằm bất kỳ thứ tự nào |
| annotation | Cho phép thêm ghi chú vào giản đồ |
| any | Cho phép phần tử bất kỳ xuất hiện tuần tự hoặc theo lựa chọn |
| anyAttribute | Cho phép thuộc tính bất kỳ xuất hiện trong kiểu phức tạp chứa hoặc trong nhóm thuộc tính chứa |
| attribute | Khai báo thuộc tính |
| attributeGroup | Tạo nhóm thuộc tính |
| choice | Cho phép một và chỉ một phần tử được chọn trong phần tử chứa |
| complexType | Khai báo kiểu phức tạp |
| documentation | Chứa tài liệu được đặt trong phần tử annotation |
| simpleType | Khai báo kiểu đơn giản |
| element | Khai báo phần tử |
| field | Chứa biểu thức XML Path Language chỉ ra giá trị ràng buộc |
| group | Khai báo nhóm phần tử |
| include | Bao gồm giản đồ đã cho trong giản đồ hiện thời |
| retriction | Khai báo ràng buộc |
| sequence | Bắt buộc các phần tử đã cho phải xuất hiện theo tuần tự định trong tài liệu XML |
| key | Chỉ ra gia trị thuộc tính hoặc giá trị phần tử phải là khóa |
| keyref | Chỉ ra gia trị thuộc tính hoặc giá trị phần tử phải hợp với khóa đã cho |
| list | Khai báo phần tử kiểu đơn giản như một danh sách giá trị kiểu đã cho |
| notation | Giữ thông báo về mô tả định dạng của dữ liệu phi XML trong tfi liệu XML |

CHƯƠNG 5. XPATH và XSL

5.1. Xpath

5.1.1. Giới thiệu

Trong phần này chúng ta sẽ tìm hiểu những khái niệm cơ bản nhất về 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 một cây có nhiều nút. Mỗi phần tử hay thuộc tính là một node. Để minh họa ý niệm này, bạn hãy quan sát tài liệu đặt hàng (order.xml) 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>

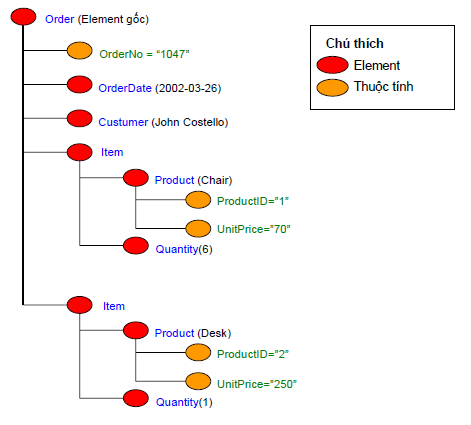
<PRODUCT ProductID=“2” UnitPrice=“250”>Desk</PRODUCT>

<QUANTITY>1</QUANTITY>

</ITEM>

</ORDER>

Với ví dụ trên sẽ được biểu diễn dưới dạng cây sau:

******

5.1.2. 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 hình 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**

5.1.3. Đường dẫn tương đối

Khi 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 điểu 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**

5.1.4. 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/\***

5.1.5. 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]**

5.1.6. 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/\*[normalizespace(@  ProductID)=’abc’]  Chọn tất cả các phần tử con của Item có  thuộc tính ProductID=abc (không phân biệt khoảng trắng) |
| startswith(  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 chuổi 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 |
| last() | Vị trí nút cuối cùng | //Item[last()]  Chọn phần tử Item cuối cùng |

5.1.7. 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’) | startswith(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-orself::\***  chọn 2 phân tử product, 2 phần tử Item và phần tử Order |

5.2. XSL (eXtensible style sheet)

5.2.1 XSL là gì?

XSL là một ngôn ngữ chuẩn giúp chuyển đổi tài liệu XML thành một địng 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 phần trước đã tìm hiểu về XPath, XPath giúp cho việc 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

XSL là ngôn ngữ định kiểu mở rộng, thực hiện nhiệm vụ biến đổi dữ liệu của XML để hiển thị tài liệu tượng tự như HTML. XSL được phân ra làm 2 nhánh:

* XSL chuyên về chuyển dịch (Tranform) còn gọi là XSLT: trích rút dữ liệu XML đưa vào khuôn dạng HTML.
* XSL chuyên về định dạng còn gọi là XSL FO (Forrmating Object): cung cấp các thuộc tính thiết lập: Font chữ, màu săc, hình ảnh,… hỗ trợ cho việc định dạng tài liệu.

5.2.2 Qui tắc chung

Bản thân XSL cũng là một XML well-formed nhưng nó chứa những lệnh của chính nó. Vì vậy phải tuân thủ mọi quy tắc của một XML well-formed. Để trình phân tích XML nhận diện được các lệnh của XSL thì 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**

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

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

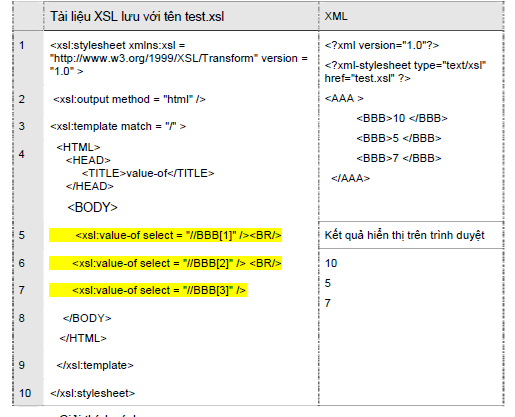
<?xml-stylesheet type=”text/xsl” href=”test.xsl”?>

5.2. 3 Một số phần tử(element) thường dùng của XSL

***5.2.3.1 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:

****

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

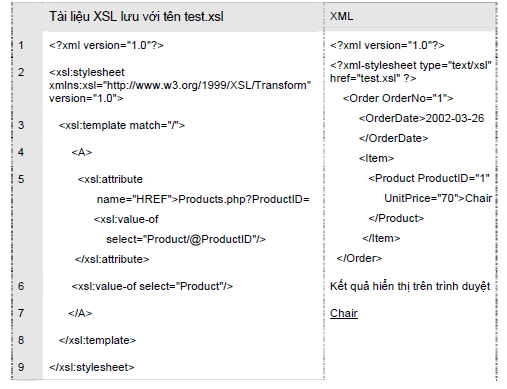
Dòng 9: Thẻ đóng phần tử template chính

Dòng 10: Thẻ đóng của phần tử stylesheet

***5.2.3.2 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:

****

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

***5.2.3.3 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**

***5.2.3.4 Phần tử element***

Phần tử này cho phép chúng ta thêm một phần tử vào tài liệu kết quả. Phần

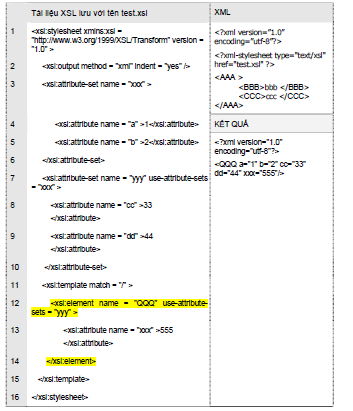
tử này có 3 thuộc tính:

• **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:



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

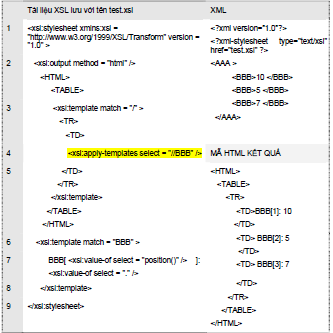
Dòng 11: Chỉ định phần tử gốc

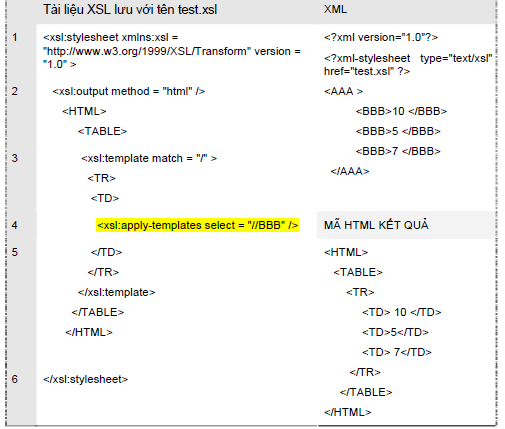
Dòng 12: Thiết lập phần tử QQQ có các thuộc tính ngoài thuộc tính được thiết lập trong dòng 13 còn sử dụng thêm tập thuộc tính yyy

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".

***5.2.3 .5 Phần tử apply-templates***

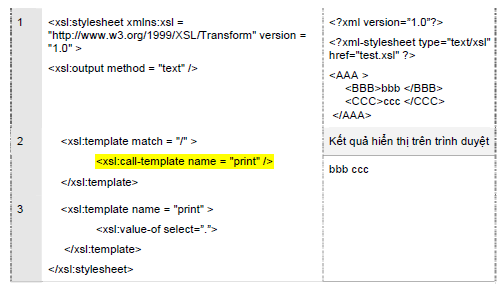
Khi một style sheet chứa nhiều phần tử template, chúng ta có thể áp dụng chúng vào một khung trình bày nào đó bằng cách sử dụng phần tử **applytemplates**. Chúng ta cần tạo ra một phần tử template để chứa phần tử applytemplates, nó sẽ lấy kết quả của các template nằm bên ngoài template chứa nó để đưa 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.



****

***5.2.3.6 Phần tử call-template***

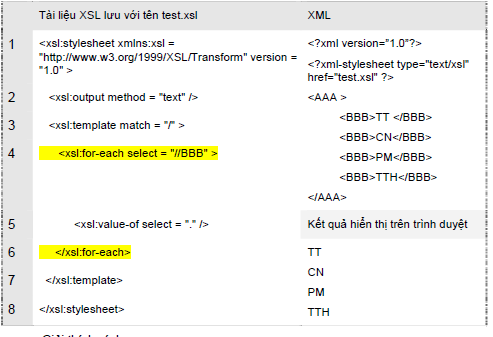
Phần tử này được dùng để triệu gọi một **xsl:template** bởi tên của **xsl:template** này.

****

***5.2. 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:

****

Giải thích ví dụ

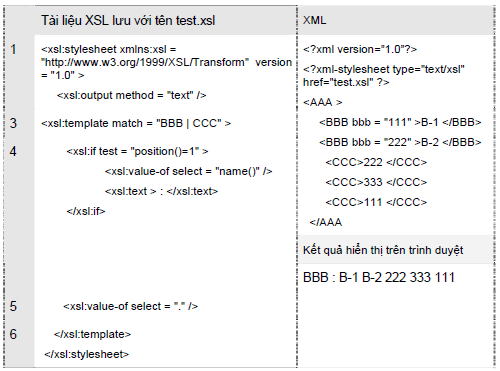
Dòng 4: Phần tử for-each sẽ cho phép duyệt qua hết tất cả các phần tử BBB

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

***5.2. 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.

Ví dụ

****

Giải thích ví dụ

Dòng 3: Dùng dể 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.

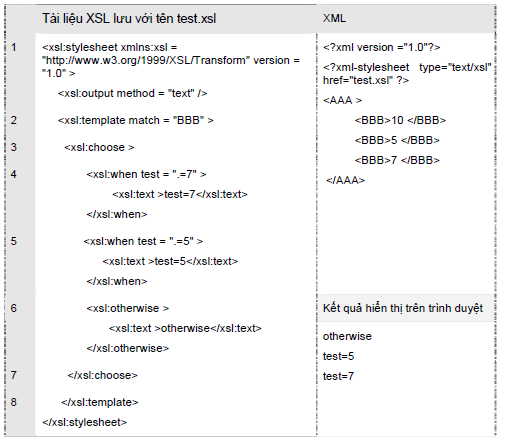
Dòng 5: Lấy nội dung của node hiện thời.

***5.2.3.9 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.

Ví dụ:

****

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.

***5.2. 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

***5.2.3.11 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).

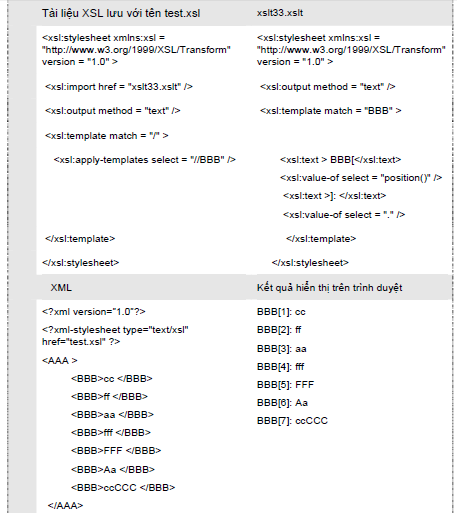
***5.2.3.12 Phần tử include***

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.

***5.2.3.13 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:



5.2.4. Cách áp dụng Style Sheet vào tài liệu XML

Áp dụng một Style Sheet là một chức năng của một bộ phân tích XML như MSXML của Internet Explorer. Bạn có thể bảo một bộ phân tích XML áp dụng một Style Sheet vào một XML bằng cách thêm chỉ thị xử lý vào đầu tài liệu XML.

***Dùng Bộ phân tích XML để hiển thị***

Nếu ta lưu trữ XSL Style Sheet của hồ sơ đặt hàng trong một file tên **Order.xsl** thì ta có thể thêm một hàng chỉ thị xử lý***xml-stylesheet*** vào đầu hồ sơ đặt hàng XML như sau:

<?xml version=“1.0”?>

<?xml-stylesheet type=“text/xsl” href=“Order.xsl”?>

<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>

Khi một Bộ phân tích XML đọc tài liệu XML này, chỉ thị xử lý *xml-stylesheet* bảo bộ phần tích áp dụng hồ sơ style sheet **Order.xsl** để biến đổi XML.

Attribute **type** cho biết loại style sheet được áp dụng, hoặc là **XSL style sheet** hoặc là **cascading style sheet (CSS)**, một loại style sheet dùng để chỉ định màu và kiểu chữ. Ở đây nó là XSL style sheet trong dạng text.

Attribute **href** cho biết tên của file dùng làm Style Sheet, đường dẫn của tên file ấy có thể là tương đối hay tuyệt đối. Ở đây tên file của style sheet là Order.xsl, không có path nên có nghĩa là nó nằm trong cùng một folder với Order.xml.

***Dùng code để transform với XSL***

Cách dùng một ngôn ngữ lập trình để bảo một bộ phân tích XML chế biến một tài liệu XML sẽ tùy thuộc vào hoàn cảnh. Nếu bạn dùng bộ phân tích XML của Microsoft, một thành phần tên MSXML, trong lập trình thì tài liệu XML sẽ được nạp vào trong một đối tượng **Document Object Model (XMLDOM)**. Kế đó bạn có thể gọi method **transformNode** để áp dụng một XSL style sheet đã được loaded trước đó vào một XMLDOM object khác để chế biến XML.

Như trong thí dụ dưới đây, ta dùng hai DOM, một cái để load tài liệu order.xml, một cái khác để load **Order.xsl** trong VBScript chạy trên **Active Server PAGES (ASP)**:

Dim objXML ' DOM for XML

Dim objXSL ' DOM for XSL

Dim strResult ' Resultant document

'Load the XML document.

Set objXML = ActiveXObject(“Microsoft.XMLDOM”)

objXML.Async = False

objXML.Load “c:\Order.xml”

'Load the XSL style sheet.

Set objXSL = ActiveXObject(“Microsoft.XMLDOM”)

objXSL.Async = False

objXSL.Load “c:\Order.xsl”

'Apply the style sheet to XML

strResult = objXML.transformNode(objXSL)

Ta cũng có thể code bằng JavaScript để chạy trong Browser, thay vì trong WebServer, như cho thấy trong trang Web dưới đây. Nó cũng cho ra cùng một kết quả như khi dùng IE để hiển thị XML trực tiếp.

<HTML>

<HEAD>

<TITLE>sample</TITLE>

<SCRIPT language=“javascript”>

function init()

{

var srcDOM = new ActiveXObject(“Msxml2.DOMDocument.4.0”);

srcDOM.async=false;

srcDOM.load(“order.xml”);

var xsltDOM= new ActiveXObject(“Msxml2.DOMDOCUMENT.4.0”);

xsltDOM.async = false;

xsltDOM.load(“order.xsl”);

resDOM.innerHTML = srcDOM.transformNode(xsltDOM);

}

</SCRIPT>

</HEAD>

<BODY onload=“init()”>

<div id=“resDOM”></div>

</BODY>

</HTML>

Có lẽ bạn hỏi tại sao ta không dùng thẳng XML như phía trên để hiển thị trang Web. Lưu ý là ta có thể dùng kỹ thuật này để Transform một XML với XSL rồi hiển thị nó bên trong một **DIV**, tức là một vùng giới hạn bên trong trang Web, chớ không chiếm cả trang Web. Tại đây khi trang Web bắt đầu load (**onload** event), IE gọi **function init()** để transform XML rồi assign kết quả vào **property innerHTML** của **DIV resDOM**.

Có một method khác ta cũng có thể dùng thay cho **transformNode** là **transformNodeToObject**. Sự khác biệt chính giữa hai methods này là:

* **transformNode:** Kết quả của method này là một tree dưới dạng text string, điển hình là một hồ sơ HTML. Ta có thể cho nó hiển thị trong một browser hay lưu trữ vào một file.
* **transformNodeToObject:** Kết quả của method này được để vào trong một object khác, rồi chính object ấy có thể sẽ được chế biến thêm.

Khi ta dùng một trong hai method nói trên, thật ra object nguồn (source object) không cần phải là một hồ sơ đầy đủ. Nó có thể chỉ là một nút của hồ sơ XML. Nếu nó chỉ là một nút thì cái **XSLT processor** xem tập hợp nút ấy, và các nút con cháu của nó như một hồ sơ đầy đủ. Tương tự như vậy, một object XSL có thể là một file XSL đầy đủ, hay chỉ là một nút bên trong một file XSL. Bạn có thể [tải về order.xml, order.xsl và trang Web có JavaScript](http://www.vovisoft.com/XML/XML05/order.zip) tại đây: http://www.vovisoft.com/XML/XML05/order.zip

CHƯƠNG 6. DOM VÀ XỬ LÝ XML VỚI JAVASCRIPT

6.1. Mô hình DOM

DOM (Document Object Model) là phương tiện cho phép truy xuất và chỉnh sửa các tài liệu XML. Thông qun DOM, XML có tính toàn càu thực sự, dùng được cho nhiều ứng dụng khác nhau và là ngôn ngữ đánh dấu độc lập với ứng dụng. DOM là tính năng hữu ích nhất của XML, việc sửu dụng DOM có thể tạo, mở, đọc, chỉnh sửa và đóng tài liệu XML.

Để xử lý, W3C định nghĩa tài liệu XML theo mô hình đối tượng tài liệu DOM (Document Object Model). Theo mô hình này, tài liệu XML là một cây bao gồm tập hợp các nút. Những nút này nội dung có thể chứa phần tử, dữ liệu, thuộc tính, … và các nút con khác.

Ví dụ, giả sử chúng ta có tài liệu XML sau:

<DOCUMENT>

<GREETING>Hello from XML</GREETING>

<MESSAGES>Well come to XML</MESSAGES>

</DOCUMENT>

Tài liệu này có thể phân theo cấu trúc hình cây bao gồm các nút đối tượng.

<DOCUMENT>

<GREETING>

< MESSAGES>

Hello from XML

Well come to XML

Ngoài chỉ thị xử lý ra, phần tử <DOCUMENT> là một nút bao gồm hai nút con là <GREETING> và <MESSAGES> lần lượt chứa một nút con khác lưu trữ dữ liệu dạng văn bản với nội dung “Hello from XML” và “Well come to XML”. Toàn bộ cấu trúc trên chính là mô hình DOM. Khi phân tích cây tài liệu DOM ta coi mỗi nút là một đối tượng. DOM cung cấp cho bạn các phương thức như nextChild, lastChild, … để lần ra toàn bộ các nút con khác.

6.2. Cấu trúc cây của một tài liệu XML

Để hiểu một DOM, trước hết ta tìm hiểu một mã của tài liệu XML và xét nó dưới góc độ DOM:

Ví dụ tài liệu cuonsachs.xml

<?xml version = “1.0” encoding = “UTF-8”?>

<khosach>

<cuonsach theloai=“cooking”>

<tieude ngonngu = “en”>everyday italian</tieude>

<tacgia>Giada De Laurentiis</tacgia>

<namxb>2005</namxb>

<gia>30.000</gia>

</cuonsach>

<cuonsach theloai=“children”>

<tieude ngonngu = “en”>Harry Potter</tieude>

<tacgia>Giada De Laurentiis</tacgia>

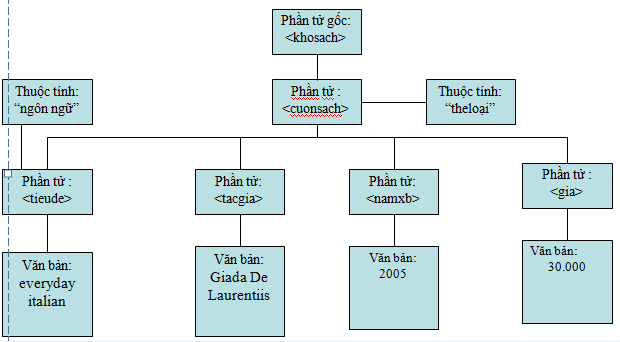
<namxb>2005</namxb>

<gia>30.000</gia>

</cuonsach>

</khosach>

DOM xem tài liệu XML hợp khuôn dạng như một cấu trúc cây. Biểu đồ đồ họa của các nút như sau:



6.3. Các đối tượng trong một tài liệu XML

Mô hình DOM xem XML như là một tập các đối tượng nút. Các nút có thể truy cập bằng JavaScript hoặc các ngôn ngữ lập trình khác. Giao diện lập trình DOM được định nghĩa bằng tập các thuộc tính và các phương thức chuẩn để xử lý với các nút trong tài liệu XML

**Các thuộc tính DOM**

**x.nodeName:** Tên của x

**x.nodeValue:** Giá trị của x

**x.parentnode**: nút cha của x

**x.childNodes**: các nút con cuả x

**x.attributes**: các thuộc tính của nút x

Chú ý: Với x là đối tượng nút

**Các phương thức**

**x.getElementByTagName(name):** trả về tất cả các phần tử được xác định bởi thẻ name.

**x.appendChild(nút):** thêm một nút con vào x

**x.removeChlid(nút):** Loại ột nút con khỏi x

6.4. Tải một tài liệu XML bằng JavaScript

Có hai cách để nạp dữ liệu XML vào trình duyệt IE bằng JavaScript. Thứ nhất là dùng lớp đối tượng Microsoft.XMLDOM và thứ hai là dùng phần tử nạp dữ liệu <XML>.

6.4.1. Nạp dữ liệu XML vào trình duyệt với lớp đối tượng Microsoft.XMLDOM

Tài liệu meetings.xml dưới đây chứa các thông tin về những người tham gia cuộc họp. Chúng ta sẽ sử dụng tài liệu này để xử lý bằng JavaScrtip theo mô hình DOM.

<?xml version=“1.0”?>

<MEETINGS>

<MEETING TYPE=“infomal”>

<MEETING\_TITLE>XML in the Real World</MEETING\_TITLE>

<MEETING\_NUMBER>12</MEETING\_NUMBER>

<SUBJECT>XML</SUBJECT>

<DATE>12/12/2007</DATE>

<PEOPLE>

<PERSON ATTENDENCE=“present”>

<FIRST\_NAME>Tuan</FIRST\_NAME>

<LAST\_NAME>Bui Anh</LAST\_NAME>

</PERSON>

<PERSON ATTENDENCE=“absent”>

<FIRST\_NAME>Binh</FIRST\_NAME>

<LAST\_NAME>Nguyen Van</LAST\_NAME>

</PERSON>

<PERSON ATTENDENCE=“present”>

<FIRST\_NAME>Hanh</FIRST\_NAME>

<LAST\_NAME>Tran Thi</LAST\_NAME>

</PERSON>

</PEOPLE>

</MEETING>

</MEETINGS>

Tạo trang tài liệu viewdata.html chứa mã JavaScript đọc và phân tích tài liệu XML trên đây như sau: trước hết cần tạo ra đối tượng xử lý DOM của Microsoft, để thực hiện điều này, bạn gọi toán tử new tạo mới đối tượng ActiveXObject với tên lớp là Microsoft.XMLDOM.

var xmldoc, meetingsNode, meetingNode, peopleNode, personNode

var first\_nameNode, last\_nameNode

xmldoc=new ActiveXObject(“Microsoft.XMLDOM”)

Tiếp theo chúng ta nạp tài liệu meetings.xml cần xử lý vào trình duyệt

xmldoc.load(“meetings.xml”)

Để duyệt qua tất cả các nút trong cây tài liệu theo mô hình DOM, ta cần xuất phát từ nút gốc. Ở đây như bạn thấy, <MEETINGS> là phần tử gốc của tài liệu. Chúng ta gọi phương thức documentElement để lấy về nút gốc của tài liệu DOM như sau:

meetingnode=xmldoc.documentElement

Bây giờ chúng ta có thể duyệt qua toàn bộ cấu trúc cây theo mô hình DOM. Sử dụng phương thức firstChild, nextSibling, priousSibling, lastSibling dùng để trả về nút cùng cấp đầu tiên, kế tiếp, trước đó hoặc nút cùng cấp sau cùng. Ví dụ như <MEETING> là nút con của nút gốc <MEETINGS> và ta có thể gọi phương thức firstChild để chuyển đến nút này như sau:

meetingNode=meetingsNode.firstChild

Chúng ta muốn tìm phần tử <PERSON> thứ ba bên trong phần tử <PEOPLE>. Do <PEOPLE> là phần tử con cuối cùng của <MEETING> nên ta có thể lần ra nút dữ liệu này như sau:

peopleNode=meetingNode.lastChild

Bởi vì phần tử <PERSON> thứ 3 mà ta muốn lấy nút con cuối cùng của <PEOPLE>, nên ta lại tiếp tục gọi phương thức lastChild của nút peopleNode.

personNode=peopleNode.lastChild

Cuối cùng ta muốn lấy thông tin về họ tên <FIRST\_NAME> và <LAST\_NAME> của phần tử <PERSON>. Bạn sử dụng phương thức firstChild và nextSibling (lấy thông tin phần tử cùng cấp tiếp theo).

first\_nameNode=personNode.firstChild

last\_nameNode=personNode.lastChild

Chúng ta vừa đi qua các nút của cây tài liệu DOM để đến được các mục thông tin cần lấy. Công việc còn lại là hiển thị thông tin lấy được ra cửa sổ trình duyệt. Trang viewdata.html hoàn chỉnh dưới đây bao gồm một nút nhấn, khi kích chuột vào nút nhấn này kết quả thông tin về người thứ 3 tham dự cuộc họp lưu trong tài liệu meetings.xml sẽ được in ra.

<HTML>

<HEAD>

<TITLE>Read XML element values</TITLE>

<script language="JavaScript">

function readXMLDocument(){

var xmldoc, meetingsNode, meetingNode, peopleNode, personNode

var first\_nameNode, last\_nameNode,outputText

xmldoc=new ActiveXObject("Microsoft.XMLDOM")

xmldoc.load("meetings.xml")

meetingsNode=xmldoc.documentElement

meetingNode=meetingsNode.firstChild

peopleNode=meetingNode.lastChild

personNode=peopleNode.lastChild

first\_nameNode=personNode.firstChild

last\_nameNode=personNode.lastChild

outputText="Third name: " + first\_nameNode.firstChild.nodeValue+last\_nameNode.firstChild.nodeValue

messageDIV.innerHTML=outputText

}

</script>

</HEAD>

<BODY>

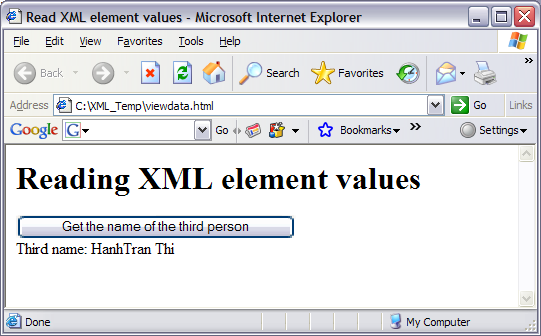
<H1>Reading XML element values</H1>

<input type="BUTTON" value="Get the name of the third person" onclick="readXMLDocument()">

<DIV ID="messageDIV"></DIV>

</BODY>

</HTML>



Hình 7.1. Hiển thị dữ liệu XML bằng JavaScript theo mô hình DOM

6.4.2. Sử dụng phần tử nạp dữ liệu XML

Với IE phiên bản 5, bạn có thể dùng thẻ <XML> nạp trực tiếp dữ liệu XML vào trang HTM để xử lý bằng JavaScript.

Khi bạn dùng phần tử XML nạp dữ liệu, nguồn dữ liệu được đặt trong thuộc tính SRC. Để truy xuất đến đối tượng dữ liệu, bạn dựa vào thuộc tính ID. Trong ví dụ sau chúng ta có một thay đổi nhỏ đối với file viewdata.html. Thay vì sử dụng ActiveXObject(“Microsoft.XMLDOM”) để nạp dữ liệu chúng ta dùng như sau:

<HTML>

<HEAD>

<TITLE>Read XML element values</TITLE>

<XML ID="meetingsXML" SRC="meetings.xml"></XML>

<script language="JavaScript">

function readXMLDocument(){

var xmldoc, meetingsNode, meetingNode, peopleNode, personNode

var first\_nameNode, last\_nameNode,outputText

xmldoc=document.all("meetingsXML").XMLDocument

meetingsNode=xmldoc.documentElement

meetingNode=meetingsNode.firstChild

peopleNode=meetingNode.lastChild

personNode=peopleNode.lastChild

first\_nameNode=personNode.firstChild

last\_nameNode=personNode.lastChild

outputText="Third name: " + first\_nameNode.firstChild.nodeValue + ’ ‘ +last\_nameNode.firstChild.nodeValue

messageDIV.innerHTML=outputText

}

</script>

</HEAD>

<BODY>

<H1>Reading XML element values</H1>

<input type="BUTTON" value="Get the name of the third person" onclick="readXMLDocument()">

<DIV ID="messageDIV"></DIV>

</BODY>

</HTML>

Kết quả vẫn không đổi.

Với thẻ <XML> bạn có thẻ nhúng trực tiếp toàn bộ dữ liệu XML trong tài liệu HTML như ví dụ sau:

<HTML>

<HEAD>

<TITLE>Read XML element values</TITLE>

<XML ID="meetingsXML">

<?xml version="1.0"?>

<MEETINGS>

<MEETING TYPE="infomal">

<MEETING\_TITLE>XML in the Real World</MEETING\_TITLE>

<MEETING\_NUMBER>XML in the Real World</MEETING\_NUMBER>

<SUBJECT>XML</SUBJECT>

<DATE>12/12/2007</DATE>

<PEOPLE>

<PERSON ATTENDENCE="present">

<FIRST\_NAME>Tuan</FIRST\_NAME>

<LAST\_NAME>Bui Anh</LAST\_NAME>

</PERSON>

<PERSON ATTENDENCE="absent">

<FIRST\_NAME>Binh</FIRST\_NAME>

<LAST\_NAME>Nguyen Van</LAST\_NAME>

</PERSON>

<PERSON ATTENDENCE="present">

<FIRST\_NAME>Hanh</FIRST\_NAME>

<LAST\_NAME>Tran Thi</LAST\_NAME>

</PERSON>

</PEOPLE>

</MEETING>

</MEETINGS>

</XML>

<script language="JavaScript">

function readXMLDocument(){

var xmldoc, meetingsNode, meetingNode, peopleNode, personNode, first\_nameNode, last\_nameNode,outputText

xmldoc=document.all("meetingsXML").XMLDocument

meetingsNode=xmldoc.documentElement

meetingNode=meetingsNode.firstChild

peopleNode=meetingNode.lastChild

personNode=peopleNode.lastChild

first\_nameNode=personNode.firstChild

last\_nameNode=personNode.lastChild

outputText="Third name: " + first\_nameNode.firstChild.nodeValue+last\_nameNode.firstChild.nodeValue

messageDIV.innerHTML=outputText

}

</script>

</HEAD>

<BODY>

<H1>Reading XML element values</H1>

<input type="BUTTON" value="Get the name of the third person" onclick="readXMLDocument()">

<DIV ID="messageDIV"></DIV>

</BODY>

</HTML>

Thay vì dùng lệnh

xmldoc=document.all(“meetingsXML”).xmlDocument

để lấy nút gốc trong cấu trúc DOM, bạn có thể sử dụng trực tiếp thuộc tính documentElement từ định danh của nguồn dữ liệu <XML> như sau:

xmldoc=meetingsXML.documentElement

6.4.3. Truy xuất các nút dữ liệu dựa vào tên

Bắt đầu từ nút gốc, chúng ta đã duyệt qua cấu trúc dữ liệu DOM dựa vào các phương thức như firstChild, nextSibling, ... Mặc dù vậy, bạn có thể dùng trực tiếp tên của phần tử để lấy phần tử ra danh sách các nút thỏa mãn điều kiện tìm kiếm. Chẳng hạn, chúng ta muốn tìm các thông tin về <FIRST\_NAME>, <LAST\_NAME>, ta có thể sử dụng phương thức getElementByTagName để lấy về danh sách các nút như sau:

listNodesFirstName = xmldoc.getElementsByTagName("FIRST\_NAME")

listNodesLastName = xmldoc.getElementsByTagName("LAST\_NAME")

Sau khi có danh sách các nút, bạn truy xuất đến từng nút trong sanh sách dựa vào thuộc tính ITEM. Lưu ý chỉ số các phần tử được đánh số bắt đầu từ 0. Chúng ta muốn lấy về tên của người thứ 3, cho nên cần truy xuất vào phần tử ITEM(2).

listNodesFirstName.item(2).firstChild.nodeValue

listNodesLastName.item(2).firstChild.nodeValue

File Viewdata.html được sửa lại thành như sau:

<HTML>

<HEAD>

<TITLE>Reading XML element values</TITLE>

<SCRIPT LANGUAGE="JavaScript">

function loadDocument()

{

var xmldoc, listNodesFirstName, listNodesLastName

xmldoc = new ActiveXObject("Microsoft.XMLDOM")

xmldoc.load("meetings.xml")

listNodesFirstName = xmldoc.getElementsByTagName("FIRST\_NAME")

listNodesLastName = xmldoc.getElementsByTagName("LAST\_NAME")

outputText = "Third name: " +listNodesFirstName.item(2).firstChild.nodeValue + ' '+ listNodesLastName.item(2).firstChild.nodeValue

messageDIV.innerHTML=outputText

}

</SCRIPT>

</HEAD>

<BODY>

<CENTER>

<H1>Reading XML element values</H1>

<INPUT TYPE="BUTTON" VALUE="Get the name of the third person" ONCLICK="loadDocument()">

<P>

<DIV ID="messageDIV"></DIV>

</CENTER>

</BODY>

</HTML>

Duyệt qua toàn bộ danh sách các nút:

<HTML>

<HEAD>

<TITLE>Reading XML element values</TITLE>

<SCRIPT LANGUAGE="JavaScript">

function loadDocument()

{

var xmldoc, listNodesFirstName, listNodesLastName

xmldoc = new ActiveXObject("Microsoft.XMLDOM")

xmldoc.load("meetings.xml")

listNodesFirstName = xmldoc.getElementsByTagName("FIRST\_NAME")

listNodesLastName = xmldoc.getElementsByTagName("LAST\_NAME")

outputText="Third name:"

for(var i=0;i<listNodesFirstName.length;i++){

outputText =outputText+listNodesFirstName.item(i).firstChild.nodeValue + ' '+ listNodesLastName.item(i).firstChild.nodeValue +"<br>"

}

messageDIV.innerHTML=outputText

}

</SCRIPT>

</HEAD>

<BODY>

<CENTER>

<H1>Reading XML element values</H1>

<INPUT TYPE="BUTTON" VALUE="Get the name of the third person" ONCLICK="loadDocument()">

<P>

<DIV ID="messageDIV"></DIV>

</CENTER>

</BODY>

</HTML>

6.4.4. Truy xuất giá trị thuộc tính

Chúng ta lấy được nội dung dữ liệu của một nút đại diện cho phần tử XML. Mặc dù vậy các phần tử XML có thể chứa thông tin lưu trong thuộc tính. Chẳng hạn, mỗi phần tử <PERSON> trong tài liệu meetings.xml đều có thuộc tính ATTENDANCE cho biết tình trạng hiện diện hay vắng mặt trong cuộc họp của một các nhân.

Với mỗi nút chứa phần tử, bạn lấy về danh sách các nút chưa thuộc tính. Trong ví dụ dưới đây chúng ta lấy về một sanh sách tất cả các thuộc tính của nút <PERSON> trước khi tham chiếu đến giá trị của thuộc tính ATTENDANCE.

attributes=personNode.attributes

Với danh sách các thuộc tính ta có được, ta gọi phương thức getNameItem để tham chiếu đến giá trị thuộc tính của một nút cụ thể theo tên của thuộc tính như sau:

attendencePerson=attributes.getNameItem(“ATTENDANCE”)

Sau cùng với nút thuộc tính có được, bạn gọi phương thức value để lấy về giá trị của thuộc tính.

attendencePerson.value

File viewdata.html hoàn chỉnh như sau:

<HTML>

<HEAD>

<TITLE>Read XML element values</TITLE>

<script language="JavaScript">

function readXMLDocument(){

var xmldoc, meetingsNode, meetingNode, peopleNode, personNode, first\_nameNode, last\_nameNode,outputText

xmldoc=new ActiveXObject("Microsoft.XMLDOM")

xmldoc.load("meetings.xml")

meetingsNode=xmldoc.documentElement

meetingNode=meetingsNode.firstChild

peopleNode=meetingNode.lastChild

personNode=peopleNode.lastChild

first\_nameNode=personNode.firstChild

last\_nameNode=personNode.lastChild

attributes=personNode.attributes

attendencePerson=attributes.getNamedItem("ATTENDENCE")

outputText="Third name: " + first\_nameNode.firstChild.nodeValue+last\_nameNode.firstChild.nodeValue +attendencePerson.value

messageDIV.innerHTML=outputText

}

</script>

</HEAD>

<BODY>

<H1>Reading XML element values</H1>

<input type="BUTTON" value="Get the name of the third person" onclick="readXMLDocument()">

<DIV ID="messageDIV"></DIV>

</BODY>

</HTML>

# CHƯƠNG 7. XLINK, XPOINTER

## Xlink

XLink hỗ trợ cho liên kết tài liệu một cách rất tổng quát. Trong HTML bạn phải sử dụng phần tử <A>, <IMG> để tạo liên kết. XLink không qui định phần tử liên kết, thay vào đó liên kết được chỉ ra bởi thuộc tính của phần tử. Vì lý do này bạn có thể định nghĩa mọi phần tử XML có thuộc tính liên kết tạo nên XML. Các phần tử chứa thuộc tính XLink được gọi là phần tử liên kết.

Một số kiểu XLink được định nghĩa thông qua giá trị thuộc tính Xlink:type là:

simple Liên kết đơn giản

extended Liên kết mở rộng

localor Định vị

arc Cung liên kết

resource Tài nguyên liên kết

None Tùy biến

### Xlink đơn giản

Liên kết giản đơn simple là liên kết tương tự như siêu liên kết của HTML. Ví dụ <COMPOSER> dưới đây được định nghĩa là liên kết giản đơn trỏ đến tài liệu abount.html lưu thông tin về tác giả.

<COMPOSER

xmlns:xlink=“<http://www.w3c.org/1999/xlink>“

xlink:type=“simple”

xlink:href=“http://www.tnut.edu.vn/abount.html”>

Full Name: Nguyen Van A

Address: Thai Nguyen

</COMPOSER>

Thuộc tính xlink:href là địa chỉ trỏ đến vị trí của tài nguyên cần liên kết.

Phần tử XLink có những hành vi như nạp thẳng trực tiếp tài liệu liệu liên kết vào chung với tài liệu hiện hành, mở liên kết trong một vùng cửa sổ khác, … những hành vi này được đặt trong thuộc tính Xlink:show và Xlink:actuate.

Với thuộc tính Xlink:show, bạn có thể định các giá trị như

* + “new”: mở cửa sổ mới để hiển thị tài liệu liên kết),
  + “replace”: thay thế dữ liệu hiển thị hiện hành
  + “embed”: nhúng tài liệu liên kết vào tài liệu hiện hành.
  + “undefined”: tùy phần mềm xử lý liên kết quyết định.

Các liên kết ảnh mà bạn thấy trong tài liệu HTML chính là khái niệm liên kết nhúng. Khi trình duyệt đọc đến liên kết <IMG>, toàn bộ dữ liệu ảnh sẽ được đọc và lấy về thay vào nơi đặt liên kết. Có thể biểu diễn khái niệm XLink này như sau:

<PICTURE xlink:type=“simple” Xlink:href=“log.gif” Xlink:show=“embed”/>

Thuộc tính xlink:actuate cho biết cách ứng xử của liên kết tùy vào sự kiện. Bạn có thể đặt thuộc tính này các giá trị như:

* + “onLoad”: Ứng xử khi liên kết được nạp vào lần đầu tiên.
  + “onrequest”: ứng xử khi liên kết được kích hoạt.
  + “undefined”: tùy phần mềm xử lý.

Ví dụ yêu cầu nhúng dữ liệu liên kết ảnh ngày khi trang tài liệu hiển thị dựa vào sự kiẹn “Onload” như sau:

<PICTURE xlink:type=“simple” Xlink:href=“log.gif” Xlink:show=“embed” xlink:actuate=“Onload”/>

Nếu cần sử dụng các các giá trị mặc định và làm cho tài liệu hợp lệ, bạn có thể định nghĩa tài liệu DTD cho các phần tử XLink là:

<!ELEMENT PICTURE EMPTY>

<!ATTLIST PICTURE

xmlns:xlink CDATA #FIXED “<http://www.w3c.org/1999/xlink>”

xlink:type CDATA #FIXED “simple”

xlink:href CDATA #REQUIRED

xlink:show (new | replace | embed) CDATA #IMPLIED “embed”

xlink:actuate CDATA #FIXED “onload”>

<PICTURE Xlink:href=“log.gif”/>

<PICTURE Xlink:href=“log.gif” Xlink:show=“new”/>

Trong ví dụ trên, do <PICTURE> đã được định nghĩa kiểu tư liệu DTD nên các thuộc tính có chỉ định #FIXED hoặc #IMPLED ta không cần gán giá trị cho nó.

Bạn có thể sử dụng thuộc tính xlink:title và xlink:role được dùng cung cấp thông tin về liên kết chủ yếu danh cho mục đích sử dụng của cá chương trình. Ta có thể mô tả cho liên kết COMPOSER ở trên như sau:

<COMPOSER

xmlns:Xlink:=“<http://www.w3c.org/1999/xlink>“

xlink:type=“simple”

xlink:href=“<http://www.tnut.edu.vn/abount.html>“

xlink:role=“RS”

xlink:title=“Tiêu đề”>

Full Name: Nguyen Van A

Address: Thai Nguyen

</COMPOSER>

Khi một ứng dụng sử dụng liên kết, nó dùng nội dung xlink:title hiển thị ở dạng tooltip và dùng xlink:role để phân loại cho mục đích riêng nào đó.

### Xlink mở rộng

Liên kết mở rộng được dùng để liên kết đến nhiều tài liệu đích và từ nhiều nguồn khác nhau. Đối với liên kết mở rộng, việc xác định và nguồn còn phụ thuộc vào chiều liên liên kết. XLink mở rộng là một tập các định nghĩa bao gồm quan hệ giữa tài nguyên nguồn và tài nguyên đích. Có hai loại tài nguyên liên kết được chỉ ra bởi thuộc tính xlink:type đó là tài nguyên cục bộ (tương ứng với xlink:type=“resource”) và tài nguyên ở xa (tương ứng xlink:type=“locator’).

Tài nguyên cục bộ được định nghĩa trực tiếp bên trong liên kết. Còn tài nguyên ở xa thì không có trong phần định nghĩa của liên kết mà chỉ được tham chiếu đến thông qu địa chỉ URI hoặc URL.

Ví dụ, muốn xây dựng một liên kết website cung cấp thông tin. Một trong những site mà bạn muốn liên kết đến đó là <http://www.yahoo.com>. Website này có thêm ba site phụ nằm ở ba quốc gia khác nhau. Trong HTML bạn phải tạo 2 liên kết đến ba site đó. Tuy nhiên trong XML bạn chỉ cần thiết kết một liên kết duy nhất. Trình duyệt sẽ tự động chọn địa chỉ site nào gần với người dùng nhất.

<WEBSITE xmlns:xlink=<http://www.w3c.org/199/xlink>

xlink:type=“extended”>

<NAME xlink:type=“resource”>Trường ĐHKTCN</NAME>

<HOMESITE xlink:type=“locator” xlink:href=“http://www.tnut.edu.vn”>

<MIRROR xlink:type=“locator” xlink:href=“http://www.tnut1.edu.vn”>

<MIRROR xlink:type=“locator” xlink:href=“http://www.tnut2.edu.vn”>

</WEBSITE>

Chúng ta định nghĩa WEBSITE bao gồm 3 tài nguyên ở xa (bằng chỉ định xlink:type=“locator”) và một tài nguyên cục bộ xlink:type=“resource”. Khi hiển thị, ứng dụng hoặc trình duyệt sẽ hiển thị nội dung tài nguyên cục bộ và khi người dùng kích hoạt liên kết thì các địa chỉ liên kết nào để đi đến là do trình duyệt hoặc ứng dụng, tuy nhiên điều này chỉ mới xây dựng tren lý thuyết. Hiện tại chưa có trình duyệt nào hỗ trợ khả năng này.

***Cung kết nối:***

Bạn có thể tạo kết nối giữa các đỉnh tài nguyên trong liên kết mở rộng XLink bằng cách gán xlink:type=“arcs” (cung). Một cung là đường đi hay kết nối từ đỉnh này đến đỉnh khác. Cung kết nối bao gồm tài nguyên gốc (qui định bởi xlink:from) và tài nguyên đích (qui định bở xlink:to).

Bạn đặt tên cho các đỉnh liên kết trong định nghĩa XLink mở rộng thông qua thuộc tính xlink:role và dùng tên này để gán cho xlink:from và xlink:to. Dưới đây là ví dụ minh họa:

<WEBSITE xmlns:xlink=“http://www.w3c.org/199/xlink”

xlink:type=“extended” xlink:title=“Truong DHKTCN”>

<NAME xlink:type=“resource” xlink:role=“source”>DHKTCN</NAME>

<HOMESITE xlink:type=“locator”

xlink:href=“http://www.tnut.edu.vn”

xlink:role=“VN”/>

<MIRROR xlink:type=“locator”

xlink:href=“http://www.tnut1.edu.vn”

xlink:role=“US”/>

<MIRROR xlink:type=“locator”

xlink:href=“http://www.tnut2.edu.vn”

xlink:role=“CH”/>

<CONNECTION xlink:type=“arc” xlink:from=“source”

xlink:to=“CH” xlink:show=“replace”

xlink:actuate=“onRequest”/>

<CONNECTION xlink:type=“arc” xlink:from=“source”

xlink:to=“US” xlink:show=“replace”

xlink:actuate=“onRequest”/>

<CONNECTION xlink:type=“arc” xlink:from=“source”

xlink:to=“VN” xlink:show=“replace”

xlink:actuate=“onRequest”/>

</WEBSITE>

Chúng ta có 3 cung kết nối. Cung thứ nhất bắt nguồn từ tài nguyên có tên là source đến tài nguyên có tên là “CH”. Liên kết cung này cung cấp các hành vi xlink:show =“replace” và xlink:actuate=“onRequest” yêu cầu nội dung tài liệu đích thay thế dữ liệu nguồn khi được kích hoạt

***Cung kết nối nhiều đỉnh:***

Ngoài liên kết cung từ đỉnh này đến đỉnh khác, bạn còn có thể liên kết cung từ một nguồn đến nhiều tài nguyên đích cùng tên. Trong ví dụ sau, tài nguyên nguồn “source” tạo kết nối đến ban đỉnh cùng mang tên “mirror”:

<WEBSITE xmlns:xlink=“http://www.w3c.org/199/xlink”

xlink:type=“extended” xlink:title=“DHKTCN”>

<NAME xlink:type=“resource” xlink:role=“source”>DHKTCN</NAME>

<HOMESITE xlink:type=“locator”

xlink:href=“http://www.tnut.edu.vn”

xlink:role=“home”/>

<MIRROR xlink:type=“locator”

xlink:title=“My”

xlink:role=“mirror”

xlink:href=“http://www.tnut1.edu.vn”/>

<MIRROR xlink:type=“locator”

xlink:title=“Trung Quoc”

xlink:role=“mirror”

xlink:href=“http://www.tnut2.edu.vn”/>

<CONNECTION xlink:type=“arc” xlink:from=“source”

xlink:to=“mirror” xlink:show=“replace”

xlink:actuate=“onRequest”/>

</WEBSITE>

Trong ví dụ trên, 2 định vị liên kết được đặt cùng tên là “mirror”. Chúng ta chỉ cần tạo một kết nối duy nhất từ “source” =>“mirror”. Ở đây liên kết “home” không có kết nối nào đến tài nguyên nguồn “source” cả.

***Cung kết nối tổ hợp:***

Nếu bạn không dùng thuộc tính xlink:to, các liên kết trong cung sẽ tổ hợp với mọi đỉnh có thể.

<WEBSITE xmlns:xlink=“http://www.w3c.org/199/xlink”

xlink:type=“extended” xlink:title=“DHKTCN”>

<NAME xlink:type=“resource” xlink:role=“source”>DHKTCN</NAME>

<HOMESITE xlink:type=“locator”

xlink:href=“http://www.tnut.edu.vn”

xlink:role=“home”/>

<MIRROR xlink:type=“locator”

xlink:title=“My”

xlink:role=“mirror”

xlink:href=“http://www.tnut1.edu.vn”/>

<MIRROR xlink:type=“locator”

xlink:title=“Trung Quoc”

xlink:role=“mirror”

xlink:href=“http://www.tnut2.edu.vn”/>

<CONNECTION xlink:type=“arc” xlink:from=“source”

xlink:show=“replace”

xlink:actuate=“onRequest”/>

</WEBSITE>

Với định nghĩa XLink, tài liệu của bạn có thể là tổ hợp của nhiều tài liệu khác hợp lại. Bạn xây dựng tài liệu của mình chỉ dựa vào các liên kết và định nghĩa cung.

## Xpointer

Xpointer dùng để trỏ đến một mục thông tin dữ liệu trong tài liệu XML. Xpointer được xây dựng dựa trên cơ sở XPath có khả năng định vị nhưng nó không có khả năng chỉ ra chính xác dữ liệu cần lấy. Để làm được điều này, bạn cần nhờ vào Xpointer. Để phân biệt sự khác nhau giữa XPath và Xpointer, bạn có thể hình dung để truy xuất dữ liệu chứa trong một tập tin văn bản nào đó. XPath chỉ có thể tạo đường dẫn đến vị trí của file, trong khi Xpointer có thể cho biết vị trí từng ký tự hoặc một đoạn văn bản cụ thể.

Xpointer định nghĩa một vị trí dữ liệu dựa trên điểm trỏ. Có hai loại điểm trỏ đó là điểm trỏ nút và điểm trỏ ký tự. Khi muốn định vị tài liệu dựa trên các phần tử XML bạn sử dụng điểm trỏ nút. Khi muốn định vị dữ liệu đến từng ký tự cụ thể của nội dung tài liệu, bạn dùng điểm trỏ ký tự. Các điểm trỏ dựa trên chỉ số để xác định vị trí. Nếu tập dữ liệu bạn chỉ định bao gồm nhiều nút con thì chỉ số sẽ các định vị trí điểm trỏ nút. Ngược lại nếu dữ liệu không chứa các phần tử thẻ XML thì chỉ số sẽ các định điểm trỏ ký tự. Ví dụ:

<DOCUMENT>

<GREETING>

Hello ther

</GREETING>

</DOCUMENT>

Đoạn tài liệu trên có hai điểm trỏ nút. Điểm trỏ nút với chỉ số 0 chỉ phần tử gốc <DOCUMENT>. Điểm trỏ nút với chỉ số 1 chỉ phần tử <GREETING>. Mặc dù vậy chúng ta có đến 12 điểm trỏ ký tự. Điểm trỏ ký tự với chỉ số 0 chỉ ký tự “H”, điểm trỏ với chỉ số 12 chỉ ký tự “!”.

XPointer thường được sử dụng chung với các địa chỉ cơ sở URI hay URL. Bạn sử dụng hàm xPointer() sau ký hiệu # của địa chỉ URL. Ví dụ về một địa chỉ sử dung XPointer có thể như sau:

xlink:href=“http://www.tnut.edu.vn/planets.xml#xpointer(start-point()[position()=16])

Giả sử chúng ta có tài liệu planets.xml như sau:

<?xml version=“1.0”?>

<?xml-stylesheet type=“text/xml” href=“planets.xsl”?>

<PLANETS>

<PLANET>

<NAME>Merury</NAME>

<MASS UNITS=“Earth=1”>.0553</MASS>

<DAY UNITS=“day”>58.65</DAY>

<RADIUS UNITS=“miles”>1516</RADIUS>

<DENSITY UNITS=“Earth=1”>0.983</DENSITY>

<DISTANCE UNITS=“miliion miles”>43.6</DISTANCE>

</PLANET>

</PLANETS>

Bạn dùng hàm start-point() để tạo ra điểm trỏ cùng với hàm position() để thiết lập chỉ số cần trỏ đến. Ví dụ, để tạo điểm trỏ ký tự trỏ đến ký tự “e” trong từ Mercury bạn sử dụng hàm xpointer như sau:

xpointer(/PLANETS/PLANET[1]/NAME/text()/start-point()[position()=1])

Hoặc truy xuất đến ký tự số 3 của nội dung phần tử <DAY>. XPointer có thể được định nghĩa như sau:

xpointer(/PLANETS/PLANET[1]/DAY/text()/start-point()[position()=3])

Bạn có thể định nghĩa điểm trỏ đến phần tử <RADIUS > như sau:

xpointer(/PLANETS/RADIUS[1]/start-point()[position()=3])

**TÀI LIỆU THAM KHẢO**

1. Nguyễn Phương Lan, (2001), *XML nền tảng và ứng dụng,*Nhà xuất bản Lao động- Xã hội.
2. Nguyễn Tiến, Đặng Xuân Hường, Nguyễn Văn Hoài, Nguyễn Tuấn Ngọc(2001), *Nhập môn XML thực hành và ứng dụng,* Nhà xuất bản Thống kê.
3. Nguyễn Thiên Bằng, Hoàng Đức Hải, Phương Lan,(2005) *Giáo trình nhập môn XML,* Nhà xuất bản Lao động – Xã hội.
4. Đỗ Thị Tâm, *Giáo trình Công nghệ XML,*  Nhà xuất bản giáo dục.
5. Frank Boumphrey, *XML cho mọi người,* Nhà xuất bản Thống kê
6. http://www.w3schools.com