

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
KHOA CÔNG NGHỆ THÔNG TIN

LÊ TẤN HÙNG
ĐINH THỊ PHƯƠNG THU
VŨ ĐỨC VƯỢNG
HUỲNH QUYẾT THẮNG (CHỦ BIÊN)

Lập trình ứng dụng
**Web
Internet
& MẠNG KHÔNG DÂY**

INTERNET AND WIRELESS PROGRAMMING

TẬP 2

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
KHOA CÔNG NGHỆ THÔNG TIN

LÊ TẤN HÙNG
ĐINH THỊ PHƯƠNG THU
VŨ ĐỨC VƯỢNG
HUỲNH QUYẾT THẮNG (Chủ biên)

LẬP TRÌNH ỨNG DỤNG
WEB INTERNET
VÀ MẠNG KHÔNG DÂY

(INTERNET AND WIRELESS PROGRAMMING)

TẬP 2



NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
HÀ NỘI

Chịu trách nhiệm xuất bản: PGS. TS. TÔ ĐĂNG HẢI
Biên tập và sửa bài: ThS. NGUYỄN HUY TIẾN
NAM SƠN
NGỌC LINH
Trình bày bìa: HƯƠNG LAN

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
70 Trần Hưng Đạo - Hà Nội

60 – 6T7.3 150 – 183.2 – 05
KHKT – 06

LỜI NÓI ĐẦU

Ngày nay việc sử dụng Internet và các dịch vụ trên Internet là một phần tất yếu của cuộc sống. Công nghệ Internet và các ứng dụng phát triển trên nền Web và Internet đã và đang làm cho đời sống của mỗi chúng ta thay đổi. Với các cán bộ viên chức, với các nhà khoa học, với các nhà kinh doanh cho đến người nội trợ bình thường, việc sử dụng Internet, sử dụng các dịch vụ trên Internet là một công việc thường xuyên. Internet là một nguồn thông tin quý giá phục vụ cho nghiên cứu, cho kinh doanh, cho giải trí. Vì vậy ngày càng có nhiều ứng dụng trên Internet, sử dụng giao diện Web. Các công nghệ phát triển trên Internet ngày một hoàn thiện.

Trên cơ sở đó, môn học Lập trình Web, Internet đã được đưa vào giảng dạy và nghiên cứu tại các khoa, ngành tin học, toán tin, thương mại điện tử, cơ khí hay các ngành có liên quan khác trong các trường đại học. Môn học này sẽ là tiền đề cung cấp cho sinh viên và các nhà nghiên cứu các lý thuyết từ cơ sở cho đến ứng dụng nhằm xây dựng và xử lý thông tin dưới dạng các trang Web, các định dạng và đặc tả theo các chuẩn thông tin trên trang Web, nhờ đó cho phép họ tiếp cận dễ dàng hơn với các kỹ thuật lập trình hiệu quả các ứng dụng trên nền Web và Internet.

Cuốn sách này được biên soạn dựa trên giáo trình giảng dạy các môn học **Thiết kế Web và Lập trình ứng dụng Web và Internet** tại Khoa Công nghệ thông tin Trường Đại học Bách Khoa - Hà Nội. Trong cuốn sách cũng đề cập đến công nghệ mạng không dây và ngôn ngữ đặc tả WML cùng với các kỹ thuật lập trình tiêu biểu. Mục đích của cuốn sách là giúp cho sinh viên và các bạn đọc nắm được các kiến thức cơ bản trong thiết kế trang Web, lập trình các ứng dụng giao diện Web trên Internet, nắm được các kiến thức về HTML, HMTML, DHTML, WML, công cụ lập trình Javascript, công cụ thiết kế trang Web Macromedia Dreamweaver. Trong cuốn sách cũng trình bày mô hình lập trình: lập trình hướng sự kiện với các kỹ thuật tiêu biểu của mô hình này. Phần cuối của cuốn sách trình bày về WML sử dụng để xây dựng các ứng dụng sử dụng mạng không dây. Cuốn sách có thể làm tài liệu tham khảo để sử dụng làm các bài tập lớn trong các môn học như Kỹ thuật lập trình, Lập trình hướng đối tượng, v.v.

Vì khuôn khổ cuốn sách có hạn mà lĩnh vực kỹ thuật lập trình Web và Internet là lĩnh vực rộng với các định hướng kỹ thuật và phát triển ứng dụng khác nhau cho nên ngoài phần đưa ra các lý thuyết cơ sở, nội dung viết chỉ đi sâu vào khai thác các kỹ thuật tiêu biểu.

Do những hạn chế của các tác giả cho nên cuốn sách không thể không có các thiếu sót nhất định. Chúng tôi chân thành mong nhận được các ý kiến đóng góp của các đồng nghiệp và các bạn đọc nhằm giúp cho chất lượng của cuốn sách được cao hơn ở các lần tái bản sau.

Ý kiến đóng góp xin gửi về:

Trường Đại học Bách Khoa Hà Nội

Khoa Công nghệ Thông tin

Tel: 04-8692463

Người nhận:

Huỳnh Quyết Thắng

Email: thanghq@it-hut.edu.vn

Vũ Đức Vương

Email: vuongvd@it-hut.edu.vn

Đinh Thị Phương Thu

Email: thudtp@it-hut.edu.vn

Lê Tấn Hùng

Email: hunglt@it-hut.edu.vn

Trong quá trình biên soạn, chúng tôi nhận được rất nhiều những ý kiến đóng góp quý báu từ các giáo sư, bạn bè và các đồng nghiệp trong và ngoài Khoa, đặc biệt là các thầy Đỗ Văn Uy, Nguyễn Ngọc Bình, Lương Mạnh Bá.

Chúng tôi cũng xin chân thành bày tỏ sự biết ơn đến **Ban Chủ nhiệm Khoa Công nghệ Thông tin - Trường Đại học Bách Khoa Hà Nội** đã động viên và tạo mọi điều kiện để cuốn sách này sớm được ra mắt bạn đọc.

Các tác giả

PHẦN 4

JAVASCRIPT

NỘI DUNG

- Chương 8 - Giới thiệu về JavaScript**
 - Chương 9 - Câu lệnh trong JavaScript**
 - Chương 10 - Câu lệnh trong JavaScript (tiếp theo)**
 - Chương 11 - Các hàm trong JavaScript**
 - Chương 12 - Sử dụng mảng trong JavaScript**
 - Chương 13 - Các đối tượng JavaScript**
-

Chương 8

Giới thiệu về JavaScript

Nội dung của chương:

- Giới thiệu về JavaScript
- Sử dụng JavaScript để tương tác với người sử dụng
- Biến và các kiểu dữ liệu mà JavaScript hỗ trợ.

8.1. Giới thiệu

JavaScript là một ngôn ngữ kịch bản (*scripting language*) được dùng để tạo ra các *script* cho máy khách và máy chủ. Các *script* cho máy khách được thực thi trong một trình duyệt (*browser*) trong khi các *script* cho máy chủ được thực thi trên một *server*. Vậy *JavaScript* là gì?

Do Sun Microsystem và Netscape tạo ra, JavaScript được phát triển nhằm mục đích cung cấp cho những người lập trình web sức mạnh và khả năng kiểm soát các chức năng của một trang web mà không làm cho nó bị quá tải với một ngôn ngữ lập trình có cấu trúc, giúp việc tạo ra các trang web tương tác dễ dàng hơn.

JavaScript được định nghĩa như là một ngôn ngữ kịch bản dựa vào đối tượng để phát triển các ứng dụng Internet cho máy khách và máy chủ. Chúng ta có thể chèn các câu lệnh JavaScript trực tiếp vào một trang web. Khi trang web đó được hiển thị trong trình duyệt, các câu lệnh JavaScript được dịch và thực hiện bởi trình duyệt đó. JavaScript là ngôn ngữ duy nhất làm việc trực tiếp với các trình duyệt. Nó có khả năng nhận ra và đáp ứng đối với các sự kiện người dùng (như kích chuột, hay các sự kiện phát ra bởi hệ thống...). Vì vậy mà có thể bổ sung tính tương tác cho trang web.

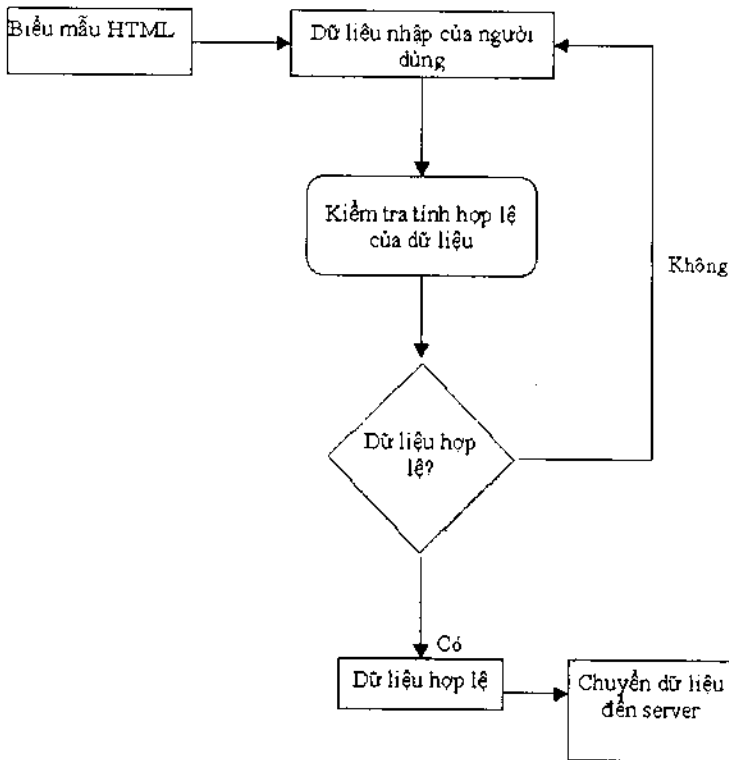
JavaScript có thể được dùng để viết các ứng dụng cả trên máy chủ và máy khách. Các ứng dụng trên máy khách chạy trong một trình duyệt, chẳng hạn như Netscape

Chương 8

Giới thiệu về JavaScript

Navigator hay Internet Explorer, và các ứng dụng chạy trên máy chủ chạy trên các Web server như Internet Information Server của Microsoft hay Netscape Enterprise Server.

- JavaScript cho máy khách bao gồm các đối tượng được định nghĩa trước chỉ có thể được dùng trong trình duyệt.
- JavaScript cho máy chủ bao gồm các đối tượng được định nghĩa trước và các chức năng có thể được dùng trong các ứng dụng dành cho máy chủ. (Hình 8.1 miêu tả cơ chế phát triển của các ứng dụng cho máy khách và máy chủ).



Hình 8.1. Phát triển ứng dụng cho máy chủ và máy khách.

Tuy nhiên bên cạnh đó JavaScript có những giới hạn:

- JavaScript không thể giao tiếp với máy chủ (chỉ làm trong phạm vi trình duyệt do đó không thể xử lý một số công việc máy chủ cần thực hiện).
- JavaScript không thể tái tạo các hình ảnh đồ họa. Nó chỉ có thể thao tác với các hình ảnh có sẵn chứ không thể tạo các hình ảnh đồ họa của riêng nó.

Chương 8

Giới thiệu về JavaScript

- JavaScript làm việc khác nhau trên các trình duyệt khác nhau. Các trình duyệt khác nhau và các phiên bản trình duyệt khác nhau hỗ trợ các phiên bản JavaScript khác nhau. Ví dụ dưới đây mô tả các phiên bản của JavaScript:

Phiên bản	Trình duyệt hỗ trợ
JavaScript 1.0	Navigator 2.0
JavaScript 1.1	Navigator 3.0
JavaScript 1.2	Navigator 4.0-4.05
JavaScript 1.3	Navigator 4.06-4.5

Có thể chèn JavaScript vào một tài liệu HTML theo nhiều cách khác nhau. Nhưng chúng chủ yếu là hai cách sau:

- Nhúng các câu lệnh trực tiếp vào tài liệu bằng thẻ `<SCRIPT>`. Bạn có thể nhúng nhiều thẻ script vào một tài liệu, khi trình duyệt gặp thẻ `<SCRIPT>`, nó đọc từng dòng nội dung cho đến khi gặp thẻ đóng `</SCRIPT>`. Sau đó kiểm tra các câu lệnh để tìm lỗi và biên dịch chúng. Ví dụ đoạn mã sử dụng thẻ `SCRIPT`

```
<Script language="JavaScript">  
  <!-- Bắt đầu JavaScript ẩn  
  Các câu lệnh JavaScript  
  // Kết thúc JavaScript ẩn - -->  
</Script>
```

- Liên kết một tệp tin nguồn JavaScript vào tài liệu HTML. Bạn cũng có thể liên kết một tệp tin bên ngoài chứa mã JavaScript với một tài liệu HTML. Thuộc tính `SRC` (nguồn) của thẻ `SCRIPT` có thể được dùng để thêm vào tệp tin bên ngoài. Bạn có thể chỉ ra các tên đường dẫn tương đối và tuyệt đối đến thuộc tính `SRC` như sau:

```
<script language="JavaScript" src="filename.js">  
</script>
```

Tập tin bên ngoài là một tệp tin văn bản chứa mã JavaScript và tên của nó kết thúc với đuôi: ".js". Không thể thêm thẻ HTML vào tệp tin bên ngoài. Ví dụ tệp tin HTML *test.htm* như sau:

Chương 8

Giới thiệu về JavaScript

```
<HTML>
<HEAD>
<script src = "test.js">
</script>
</HEAD>
<BODY>
<P>Enjoy learning experience!!!
</BODY>
</HTML>
Tập tin SRC JavaScript: (test.js)
document.write ("Hello World")
```

8.2. Chương trình JavaScript đơn giản

Chúng ta đã biết về tác dụng của JavaScript đối với trang web của mình. Đã đến lúc tạo ra một chương trình JavaScript đơn giản và chèn nó vào tài liệu HTML.

Bạn chỉ cần một trình duyệt và một chương trình soạn thảo văn bản. Bất kì một trình soạn thảo văn bản nào cũng được: *Notepad*, *Wordpad* hay *Textpad*. Hiện nay một số công cụ xây dựng web site sẽ viết JavaScript cho bạn. Thông dụng nhất là: *Dreamweaver* của Macromedia và *Frontpage* của Microsoft.

Ví dụ 8.1.

Quan sát ví dụ 8.1 sau:

```
1 <HTML>
2 <HEAD>
3 <TITLE>Chương trình JavaScript đầu tiên</TITLE>
4 <script language = "JavaScript">
5 <!--
6 document.writeln(
7   "<h1>welcome to JavaScript !</h1>" );
8   // -->
9 </script>
10 </HEAD><BODY></BODY>
11 </HTML>
```

Chương 8

Giới thiệu về JavaScript

Ví dụ trên thể hiện một khung sườn đơn giản nhất của một trang HTML với JavaScript. Ta sẽ phân tích các dòng mã trong trang HTML trên.

- Các dòng 4, 5, 6, 7, 8, 9 chính là JavaScript. Từ khóa *Language* mô tả ngôn ngữ script mà trình duyệt phải sử dụng để thông dịch script đó. Trong hầu hết các trình duyệt đều ngầm định JavaScript là ngôn ngữ mặc định, tuy nhiên có thể lựa chọn việc dùng VBScript (một ngôn ngữ kịch bản do Microsoft phát triển). Bạn cũng có thể chỉ ra phiên bản JavaScript được dùng trong từ khóa này. Ví dụ:

```
<script language="JavaScript 1.2">
```

- Để đề phòng các trình duyệt cũ không nhận ra chương trình viết JavaScript, bạn có thể thêm vào các câu lệnh JavaScript bằng thẻ ghi chú HTML (`<!-- -->`). (Trong ví dụ trên là dòng 5 và dòng 8) Điều này làm cho các trình duyệt không cung cấp JavaScript sẽ coi những dòng mã này là những chú thích và không hiển thị ra.
- Dòng 6, 7 chính là một câu lệnh của JavaScript để xuất dữ liệu ra cửa sổ trình duyệt. Để làm điều này JavaScript sử dụng phương thức: *write* và *writeln*. Sự khác nhau giữa hai phương thức là *writeln* xuất ra một ký tự xuống dòng sau khi văn bản được xuất ra.

Object.**Write** (string);

Phương thức *Write* sử dụng các hằng và biến, thậm chí có thể nối nhiều chuỗi với nhau thành một chuỗi bằng các toán tử nối. Ví dụ :

```
document.write( "<h1>chào mừng<br />đến với" +  
"<br />chương trình JavaScript!</h1>" );
```

Phương thức *writeln* tương tự như phương thức *write* ở chỗ nó được dùng để hiển thị văn bản trong cửa sổ tài liệu.

Object.**Writeln** (string) ;

Dòng 7 là dòng chứa chuỗi ký tự sẽ được đưa ra trong cửa sổ trình duyệt. Chuỗi phải bắt đầu và kết thúc bằng một dấu nháy cùng loại “ ”.

Ví dụ 8.2

Ta có thể dùng các ký tự đặc biệt chèn vào các xâu để thực hiện một nhiệm vụ cụ thể nào đó. Ví dụ:

Chương 8

Giới thiệu về JavaScript

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE = "Javascript">
document.write (" \"JavaScript\" That la đơn giản");
</SCRIPT>
</HEAD>
</HTML>
```

- Chúng ta có thể chèn ký tự khác vào một chuỗi bằng cách thêm vào trước ký tự cần chèn một ký tự "\". Bảng sau đây liệt kê các ký tự đặc biệt được sử dụng trong JavaScript và ý nghĩa của nó:

Bảng 8.1. Các ký tự đặc biệt được sử dụng trong JavaScript

Ký tự	Nghĩa
\b	Xoá lui (backspace)
\f	Form feed
\n	Dòng mới
\r	Xuống dòng
\t	Tab
\\	Ký tự vạch chéo ngược

Ví dụ 8.3

Để xuất hiện thông báo trong cửa sổ trình duyệt, bên cạnh phương thức *write* và *writeln* JavaScript còn cung cấp phương thức cảnh báo:

`object.alert ([thông báo])`

Phương thức này hiển thị một cảnh báo với một nút OK. Chúng ta không thể thay đổi thành tiêu đề của hộp thoại hiện ra.

```
<HTML>
<HEAD><TITLE> Đưa ra một cảnh báo bằng hộp thoại
</TITLE>
<script type = "text/javascript">
```

Chương 8

Giới thiệu về JavaScript

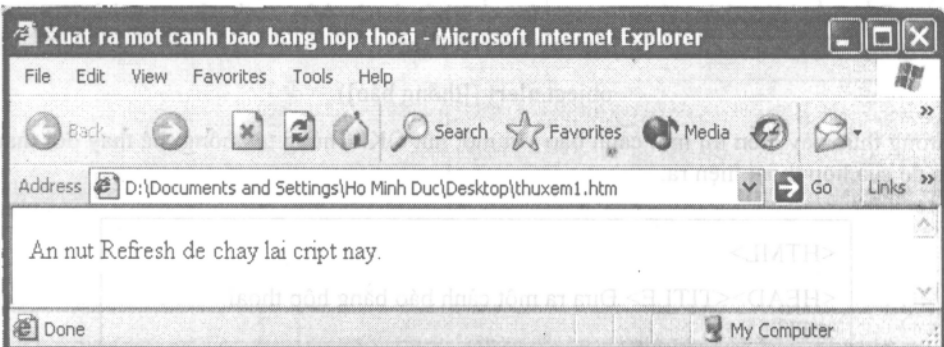
```
<!--  
window.alert( "Xin chào bạn\nđến với\nJavaScript!"); // -->  
</script>  
</HEAD>  
<BODY>  
<P>An nút Refresh để chạy lại script này.</P>  
</BODY>  
</HTML>
```

Chạy ví dụ 8.3. Cửa sổ trình duyệt sẽ xuất hiện một hộp thoại như sau:



Hình 8.2. Hộp thoại trong trình duyệt.

Sau khi bạn click vào nút OK, chương trình duyệt sẽ xuất hiện dòng chữ trong phần thân <BODY>:



Hình 8.3. Chương trình JavaScript đầu tiên.

8.3. Tương tác với người sử dụng

Một trong những điểm mạnh của JavaScript là khả năng tương tác với người sử dụng. Điều đó làm cho trang web của chúng ta trở nên sinh động hơn và có khả năng “giao tiếp” với người sử dụng. JavaScript cung cấp những phương thức sau:

Dùng Prompt:

Phương thức này dùng để hiển thị một hộp thông báo tiếp nhận dữ kiện nhập của người dùng. Chúng ta không thể thay đổi tiêu đề của hộp prompt.

object.prompt ([message[, inputdefault]]):

Trong đó *Message* là chuỗi thông báo cần thể hiện, *Inputdefault* là chuỗi hay số nguyên biểu thị giá trị mặc định của trường được nhập. Nếu bạn không chỉ ra tham số *inputdefault*, hộp thoại hiển thị giá trị *<undefined>*.

Ví dụ 8.4

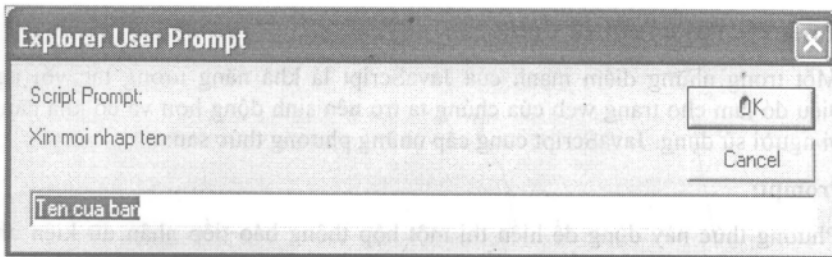
```
<HTML>
<HEAD>
<TITLE>Su dung hop Prompt va Alert</TITLE>
<script language = "JavaScript">
<!--
var name; // Chuoi ma nguoi dung nhap vao
```

```
<HTML>
<HEAD>
<TITLE>Su dung hop Prompt va Alert</TITLE>
<script language = "JavaScript">
<!--
var name; // Chuoi ma nguoi dung nhap vao
```

Trong ví dụ này, chúng ta sử dụng hộp thoại *prompt* để nhập tên của người sử dụng vào. Giao diện xuất hiện như sau:

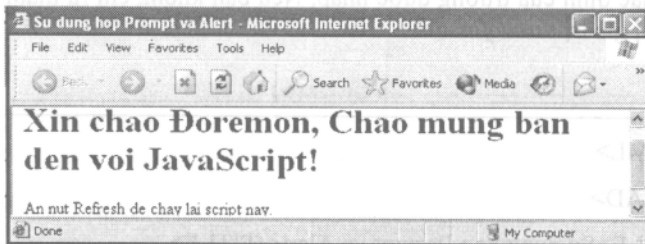
Chương 8

Giới thiệu về JavaScript



Hình 8.4. Hộp thoại prompt để nhập tên của người sử dụng.

Sau đó trình duyệt sẽ đưa ra một chuỗi nội dung chào mừng người đó. Giả sử sau khi bạn nhập tên Doremon vào hộp thoại trên, ta sẽ thấy như sau:



Hình 8.5. Tương tác với người sử dụng.

Ví dụ 8.5

JavaScript có thể trả về kết quả chính xác mà người dùng mong đợi bằng cách sử dụng *prompt* và kết hợp với các hàm cơ bản. Ví dụ sau minh họa phép cộng hai số nguyên trong JavaScript.

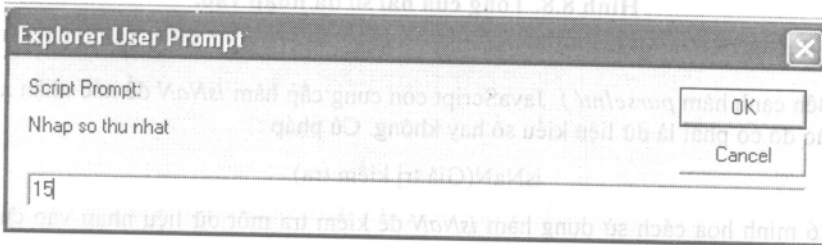
```
<HEAD> <TITLE>Tinh tong hai so</TITLE>
<script language="JavaScript">
<!--
    var so_thu_nhat, // la mot string
        so_thu_hai, // la mot string
        so1,        // so nguyen thu nhat
        so2,        // so nguyen thu hai
        tong;        // tong cua hai so
    // Nhap vao so thu nhat nhu mot string
    so_thu_nhat = window.prompt( "Nhap so thu nhat", "0" );
```

Chương 8

Giới thiệu về JavaScript

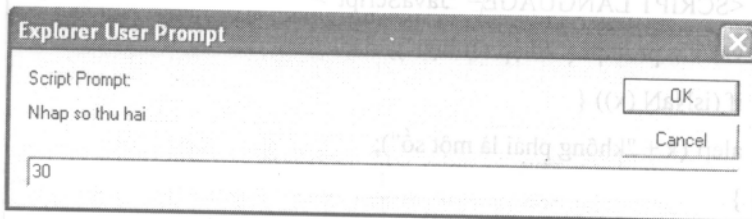
```
// Nhập vào số thứ hai như một string
so_thu_hai = window.prompt( "Nhập số thứ hai", "0" );
// chuyển các số từ string sang integer
so1 = parseInt( so_thu_nhat );
so2 = parseInt( so_thu_hai );
tong = so1 + so2;
// hiển thị kết quả
document.writeln( "<h1>Tổng của hai số : " + tong + "</h1>" );
// -->
</script>
```

Sau khi chạy, trên trình duyệt sẽ xuất hiện hộp thoại *prompt* thứ nhất cho phép chúng ta nhập số thứ nhất. Số thứ nhất bây giờ chưa phải là một số *integer* mà chỉ là một *string*.



Hình 8.6. Hộp thoại *prompt* nhập số thứ nhất.

Sau khi nhập số thứ nhất và nhấn nút OK, trình duyệt sẽ hiển thị *prompt* thứ hai cho phép chúng ta nhập số thứ hai:



Hình 8.7. Hộp thoại *prompt* nhập số thứ hai.

Đến đây số thứ hai vẫn chỉ là một *string* chứ không phải một số *integer*. Khi gặp câu lệnh

```
so1 = parseInt( so_thu_nhat );
so2 = parseInt( so_thu_hai );
```

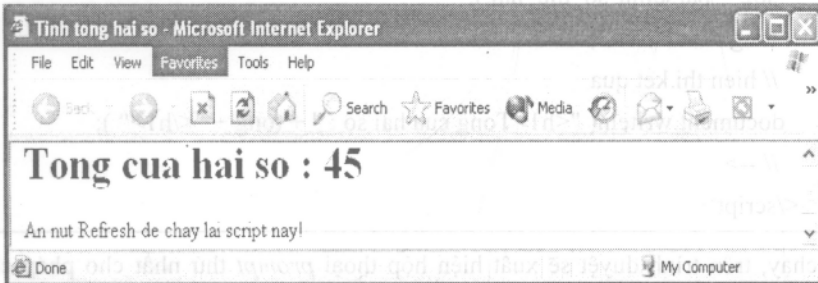
Chương 8

Giới thiệu về JavaScript

thì sẽ thực hiện chuyển đổi các chuỗi thành các số nguyên. Trong JavaScript cung cấp hàm *parseInt()* để chuyển các chuỗi thành các số. ví dụ *parseInt("15")* sẽ trả về số nguyên 15. Sau khi bấm vào nút OK, trình duyệt sẽ xuất hiện tổng của hai số nhập vào nhờ hàm :

```
document.writeln( "<h1>Tong cua hai so : " + tong + "</h1>" );
```

Khi đó trên trình duyệt xuất hiện giao diện kết quả như sau:



Hình 8.8. Tổng của hai số đã nhập vào.

Ví dụ 8.6

Bên cạnh hàm *parseInt()*, JavaScript còn cung cấp hàm *isNaN()* để xác nhận xem một giá trị nào đó có phải là dữ liệu kiểu số hay không. Cú pháp :

isNaN()(Giá trị kiểm tra).

Ví dụ 8.6 minh họa cách sử dụng hàm *isNaN()* để kiểm tra một dữ liệu nhập vào của người nhập có phải là một số không.

```
<HTML>
<SCRIPT LANGUAGE="JavaScript">
var x = prompt ("Nhập dữ liệu");
if (isNaN (x)) {
    alert (x + "không phải là một số");
}
else
    alert (x + "là một số");
</SCRIPT>
</HTML>
```


Chương 8

Giới thiệu về JavaScript

Ví dụ 8.7

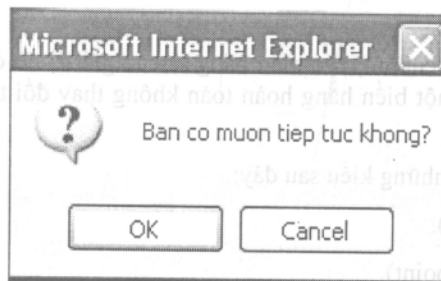
Trong quá trình làm việc đôi khi bạn cần xác định thông tin từ phía người sử dụng. Khi đó trong Javascript bạn có thể sử dụng phương thức *confirm*. Hộp thoại *confirm* sẽ hiển thị một thông báo cùng với một nút OK và một nút Cancel. Phương thức này trả về giá trị đúng nếu người dùng nhấp chuột vào nút *OK* và trả về giá trị sai nếu người dùng nhấp chuột vào nút *Cancel*. Chúng ta cũng không thể thay đổi tiêu đề của hộp thoại này. Cách gọi hộp thoại này như sau:

Object.confirm ([thông báo])

Quan sát ví dụ 3.4 sau.

```
<HTML><HEAD>
<SCRIPT LANGUAGE = "JavaScript">
confirm("Do you want to continue?");
</SCRIPT>
</HEAD>
</HTML>
```

Sau khi chạy trên trình duyệt web sẽ xuất hiện hộp thoại confirm như sau:



Hình 8.9. Hộp thoại confirm.

8.4. Những khái niệm cơ bản

Trong mục này chúng ta sẽ tìm hiểu về biến và các kiểu dữ liệu mà JavaScript hỗ trợ.

Chương 8

Giới thiệu về JavaScript

8.4.1. Biến và hằng

Biến

Biến hoạt động như là nơi lưu giữ thông tin. Có hai cách để lưu dữ liệu, hoặc là trong biến, hoặc là gõ trực tiếp vào như một hằng gọi là biến hằng (*literal*). Biến là một tham chiếu đến vị trí của bộ nhớ, nó được dùng để giữ các giá trị thay đổi trong khi thực hiện một script.

Biến trong JavaScript cũng tuân theo những quy tắc đặt tên sau:

- Tên biến phải bắt đầu bằng một chữ cái hoặc một dấu gạch dưới (" _") hay dấu (\$).
- Tên biến có thể chứa các số nguyên.
- Biến có phân biệt dạng chữ hoa và chữ thường. Điều này có nghĩa là Sum, sUM được xem như hai biến khác nhau và chúng tham chiếu tới các giá trị khác nhau.
- Tên biến bao gồm các ký tự từ "A" tới "Z" (chữ hoa) và các ký tự từ "a" đến "z" (chữ thường).
- Biến được khai báo bằng cách dùng từ khoá **var**. Biến ra đời tại lần đầu tiên dùng nó. Bạn có thể khởi tạo một biến khi khai báo hoặc sau đó trong script. Ví dụ: `var A = 10;`
- Bạn cũng có thể khai báo một biến bằng cách gán cho nó một giá trị. Ví dụ: `A = 10`
- Có thể khai báo nhiều biến trên cùng một dòng bằng cách phân biệt tên chúng bằng các dấu phẩy.

Biến hằng

Có thể thấy từ tên của nó, các biến hằng là các giá trị cố định mà bạn có thể dùng trong script. Giá trị của một biến hằng hoàn toàn không thay đổi trong khi thực hiện script đó.

Hằng có thể là một trong những kiểu sau đây:

- Số nguyên(Integer).
- Số thực (Floating point).
- Chuỗi (đặt trong 2 dấu nháy kép “ ”).
- Boolean: có hai giá trị True hoặc False.
- Null: chỉ có một giá trị là giá trị trống. Null hàm ý không có dữ liệu nào cả. Số không hay một chuỗi rỗng không thể tương đương với một giá trị Null.

Chương 8

Giới thiệu về JavaScript

8.4.2. Kiểu dữ liệu

JavaScript là một ngôn ngữ nhỏ nhưng rất mạnh, nó cung cấp cho ta tập hợp các kiểu dữ liệu sau.

Bảng 8.2. Các kiểu dữ liệu trong JavaScript

Kiểu dữ liệu	Mô tả
Số	Số nguyên (<i>integer</i>) hay số thực (<i>Floating point</i>). Ví dụ: 487 ; 3.14
Boolean	Chỉ có hai giá trị: True và False (đúng hoặc sai)
Chuỗi	Là một xâu (có thể là xâu rỗng) đặt giữa hai dấu nháy kép ""
Null	Từ khóa đặc biệt biểu thị một giá trị trống.

Ép kiểu dữ liệu:

Dữ liệu luôn thuộc một kiểu nào đó, vì vậy biến hằng cũng có kiểu. Các biến nhận kiểu dữ liệu mà chúng chứa và thay đổi dữ liệu khi dữ liệu trong chúng thay đổi. Khi ta khai báo một biến, kiểu dữ liệu của biến đó không được chỉ ra. Kiểu dữ liệu được tự động chuyển đổi khi cần thiết trong quá trình chạy. Ví dụ: `var x=10; x="JavaScript"`.

Trong JavaScript có thể kết hợp hai biến hằng khác kiểu dữ liệu với nhau. Ví dụ, nếu bạn kết hợp một giá trị kiểu chuỗi và một giá trị kiểu số, JavaScript sẽ chuyển đổi các số thành các chuỗi. Tuy nhiên bạn không thể biến một chuỗi thành một số. Nếu bạn cộng một chuỗi "12" và một biến hằng số thực (*floating-point*) 7.5 với nhau, kết quả sẽ là 127.5

8.5. Các toán tử

Các toán tử thao tác trên một hay nhiều toán hạng và trả về một giá trị mới. Trong JavaScript sử dụng cả hai loại toán tử: hai ngôi và một ngôi:

- Toán tử hai ngôi đòi hỏi phải có hai toán hạng. Ví dụ : `4 + 5`
- Toán tử một ngôi đòi hỏi phải có một toán hạng. Ví dụ : `a++`

JavaScript cung cấp cho bạn rất nhiều toán tử để thực hiện những chức năng nhất định.

8.5.1. Toán tử số học

Toán tử số học lấy các giá trị số (hoặc là hằng hoặc là biến) làm toán hạng của chúng và trả về một giá trị số. Các toán tử số học được chỉ ra trong bảng dưới đây.

Chương 8

Giới thiệu về JavaScript

Bảng 8.3. Toán tử số học trong JavaScript

Toán tử	Mô tả	Ví dụ
+	Cộng	$A = 5 + 8$
-	Trừ	$A = 8 - 5$
/	Chia	$A = 20 / 5$
%	Lấy số dư sau khi chia	$10 \% 3 = 1$
++	Tăng một ngôi. Toán tử này lấy một toán hạng. Giá trị của toán hạng đó được tăng lên một. Giá trị trả về tùy thuộc toán tử ++ được đặt trước hay sau toán hạng.	++x sẽ trả về giá trị của x sau khi tăng lên. x++ sẽ trả về giá trị của x trước khi tăng lên
--	Giảm một ngôi. Toán tử này chỉ lấy một toán hạng. Giá trị trả về tùy thuộc toán tử được đặt trước hay sau toán hạng.	--x sẽ trả về giá trị của x sau khi giảm. x-- trả về giá trị x trước khi giảm
-	Đổi dấu. Trả về giá trị toán hạng được đổi dấu.	Nếu $a = 5$ thì $a = -5$

Ví dụ 8.8

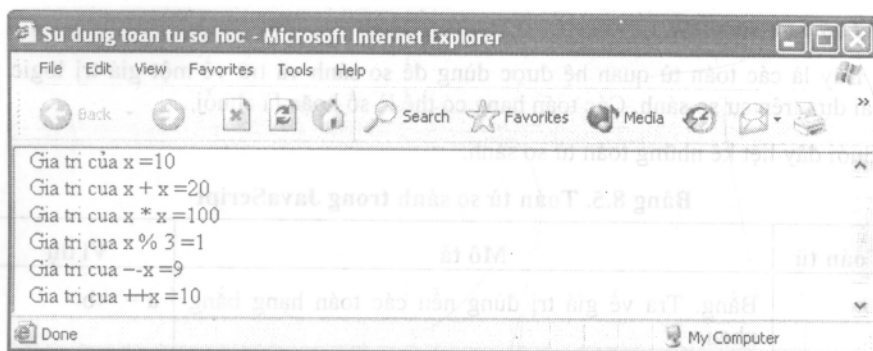
Để minh họa cho các toán tử số học, chúng ta hãy xem ví dụ sau:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
var x = 10;
document.writeln("Giá trị của x =" + x);
document.writeln("<br>Giá trị của x + x =" + (x + x));
document.writeln("<br>Giá trị của x * x =" + (x * x));
document.writeln("<br>Giá trị của x % 3 =" + (x % 3));
document.writeln("<br>Giá trị của --x =" + (--x));
document.writeln("<br>Giá trị của ++x =" + (++x));
</SCRIPT>
</HEAD>
</HTML>
```

Sau khi chạy chương trình, cửa sổ trình duyệt sẽ xuất hiện như sau:

Chương 8

Giới thiệu về JavaScript



Hình 8.10. Sử dụng toán tử số học.

8.5.2. Toán tử gán

Toán tử gán (=) được dùng để gán một giá trị cho một biến. Bạn có thể gán một giá trị tùy ý cho một biến, bao gồm cả gán một biến khác. Ví dụ:

A = 10 (Sẽ gán giá trị của toán hạng bên phải cho biến bên trái)

Total = (Mathematics + Physics + Chemistry)

JavaScript cũng như các ngôn ngữ khác như C hay java cung cấp cho chúng ta các toán tử tốc ký để thực hiện một thao tác vụn và gán kết quả của nó cho một biến trong cùng một lệnh. Bảng sau mô tả các toán tử tốc ký này.

Bảng 8.4. Toán tử gán trong JavaScript

Các toán tử tốc ký	Ý nghĩa
A += B	A = A + B
A -= B	A = A - B
A *= B	A = A * B
A /= B	A = A / B
A %= B	A = A % B
A &= B	A = A & B
A ^= B	A = A ^ B
A = B	A = A B

Chương 8

Giới thiệu về JavaScript

8.5.3. Toán tử so sánh

Đây là các toán tử quan hệ được dùng để so sánh và trả về một giá trị logic đúng hoặc sai dựa trên sự so sánh. Các toán hạng có thể là số hoặc là chuỗi.

Bảng dưới đây liệt kê những toán tử so sánh.

Bảng 8.5. Toán tử so sánh trong JavaScript

Toán tử	Mô tả	Ví dụ
<code>==</code>	Bằng. Trả về giá trị đúng nếu các toán hạng bằng nhau	<code>a == b</code>
<code>!=</code>	Không bằng. Trả về giá trị đúng nếu các toán hạng không bằng nhau	<code>Var2 != 5</code>
<code>></code>	Lớn hơn. Trả về giá trị đúng nếu toán hạng bên trái lớn hơn toán hạng bên phải	<code>Var1 > var2</code>
<code>>=</code>	Lớn hơn hoặc bằng. Trả về giá trị đúng nếu toán hạng bên trái lớn hơn hoặc bằng toán hạng bên phải	<code>Var1 >= 5</code> <code>Var1 >= var2</code>
<code><</code>	Nhỏ hơn. Trả về giá trị đúng nếu toán hạng bên trái nhỏ hơn toán hạng bên phải	<code>Var2 < var1</code>
<code><=</code>	Nhỏ hơn hoặc bằng. Trả về giá trị đúng nếu toán hạng bên trái nhỏ hơn hoặc bằng toán hạng bên phải.	<code>Var2 <= 4</code> <code>Var2 <= var1</code>

8.5.4. Toán tử điều kiện

Gán một giá trị cụ thể vào một biến nếu một điều kiện là đúng, còn nếu điều kiện là sai thì gán một giá trị thay thế. Cú pháp của nó như sau:

(điều kiện) ? trueVal: falseVal;

Chúng ta sẽ giải thích cú pháp trên qua ví dụ sau:

TinhTrang = (tuổi >= 18) ? "người lớn" : "trẻ em"

nghĩa là nếu tuổi lớn hơn hoặc bằng 18 thì biến *TinhTrang* sẽ được gán giá trị "người lớn" còn không nó sẽ mang giá trị "trẻ em".

8.5.5. Các toán tử khác

Bên cạnh những toán tử phổ biến đã viết ở trên, trong JavaScript cũng cung cấp những toán tử quen thuộc như những ngôn ngữ tin học khác.

Chương 8

Giới thiệu về JavaScript

Các toán tử theo bit (Bitwise)

Bảng 8.6. Toán tử bit trong JavaScript

Toán tử	Công Thức và miêu tả
Bitwise AND	$(x \& y)$
Bitwise OR	$(x y)$
Bitwise NOT	$\sim x$. Đảo bit của toán hạng của nó
Bitwise XOR	$(x \wedge y)$.
Dịch chuyển qua trái	$(x \ll y)$. Biến nhị phân x dịch chuyển qua trái y bit.
Dịch chuyển qua phải	$(x \gg y)$. Biểu diễn nhị phân của toán hạng x được dịch chuyển y bit qua phải. Các bit nằm ở đầu cùng bên trái sẽ bị loại.
Unsigned Right Shift	$(x \ggg y)$. Biểu diễn nhị phân của toán hạng x được dịch y bit chuyển qua phải. Các giá trị bit 0 sẽ được dịch chuyển từ bên trái

Toán tử typeof

Toán tử *typeof* cho biết kiểu dữ liệu của toán hạng. Ví dụ:

```
<HTML>
<HEAD>
<script language="JavaScript">
var x = 5;
document.write (typeof(x));
</script>
</HEAD>
<HTML>
```

Khi chạy ví dụ trên một trình duyệt web, cửa sổ trình duyệt sẽ xuất hiện sẽ cho kết quả là: number (là kiểu dữ liệu của biến x).

Chương 8

Giới thiệu về JavaScript

8.5.6. Thứ tự ưu tiên của các toán tử

Khi thao tác với các biểu thức, bạn phải tuân theo thứ tự ưu tiên của các toán tử mà JavaScript nói riêng và các ngôn ngữ lập trình nói chung. Trong JavaScript thứ tự ưu tiên của các toán tử từ thấp đến cao được chỉ ra dưới bảng sau.

Bảng 8.7. Thứ tự ưu tiên của các toán tử trong JavaScript

Kiểu toán tử	Các toán tử cụ thể
Gán	= += -= /= %= <<= >>= >>>= &= ^= =
Điều kiện	? :
Logich – or	
Logich – and	&&
Bitwise – or	
Bitwise – xor	^
Bitwise – and	&
Ngang bằng	= = !=
Quan hệ	< <= > >=
Bitwise chuyển dịch	<< >> >>>
Cộng/trừ	+ -
Nhân/chia	* / %
Phủ định/tăng	! ~ - ++ typeof(void)

8.6. Kết chương

Chương này đã cho chúng ta một khung nhìn tổng quan chung và những khái niệm cơ bản trong JavaScript: biến và hằng, kiểu dữ liệu, các toán tử,... Trong chương sau chúng ta sẽ tìm hiểu về các cấu trúc lệnh điều khiển trong các chương trình JavaScript.

Chương 9

Câu Lệnh trong JavaScript

Nội dung của chương:

- Tìm hiểu về giải thuật.
- Cấu trúc điều khiển.
- Câu lệnh điều kiện if.
- Câu lệnh điều kiện if...else.
- Câu lệnh lặp while.
- Toán tử gán.
- Toán tử tăng và giảm

9.1. Giới thiệu

Ngôn ngữ JavaScript tuy nhỏ nhưng rất mạnh cung cấp cho người lập trình web một tập lệnh khá phong phú để tạo ra các ứng dụng có hiệu quả. Trong JavaScript một câu lệnh là một đơn vị làm việc cơ sở. Một chương trình trong JavaScript được tạo nên bởi các câu lệnh. Các câu lệnh này có thể được dùng riêng rẽ hoặc chúng có thể được nhóm thành các khối lệnh. Một khối lệnh nằm trong cặp dấu ngoặc { }. Mỗi câu lệnh trong JavaScript phải được kết thúc bằng một dấu chấm phẩy. Chúng ta có thể có nhiều câu lệnh trong một dòng.

Các câu lệnh trong JavaScript có những dạng sau:

- Các câu điều kiện.
- Các vòng lặp.
- Các thao tác với các đối tượng.

Chương 9

Câu lệnh trong JavaScript

Trong đó các câu lệnh điều kiện xác định luồng chương trình cho nhiều phần khác nhau tùy thuộc vào kết quả của các kiểu kiện được chỉ ra. Các vòng lặp được dùng để lặp lại một khối lệnh cho đến khi điều kiện kiểm tra là đúng.

Khi bắt đầu viết một chương trình trong JavaScript, chúng ta phải hiểu rõ vấn đề mình cần làm để từ đó tìm ra cách tiếp cận chính xác nhất. Làm được điều đó sẽ giúp chúng ta viết chương trình một cách ngắn gọn dễ hiểu thể hiện qua những câu lệnh có chọn lọc và cấu trúc chương trình sáng sủa.

Với tập lệnh đa dạng trong JavaScript, chắc chắn sẽ giúp bạn viết nên được những hiệu ứng đẹp trong trang web của mình.

9.2. Giải thuật

Trước khi đi vào nghiên cứu cấu trúc của tập lệnh trong JavaScript, chúng ta phải hiểu rõ khái niệm “giải thuật”. *Vậy giải thuật là gì?*

Giải quyết vấn đề là một quá trình phức tạp đòi hỏi phải suy nghĩ, lên kế hoạch, lập luận chính xác, kiên trì và chú ý đến từng chi tiết. Máy tính không thể tự giải quyết vấn đề nếu lập trình viên không cung cấp giải pháp. *Giải pháp này gọi là giải thuật (hay còn gọi là thuật toán).*

Các bước để giải quyết vấn đề.

- Nghiên cứu chi tiết vấn đề.
- Tập hợp các thông tin thích hợp.
- Xử lý các thông tin.
- Đi đến kết quả.

Giải thuật là một dãy các câu lệnh (*statements*) chặt chẽ và rõ ràng xác định một trình tự các thao tác trên một số đối tượng nào đó sao cho sau một số hữu hạn bước thực hiện ta đạt được kết quả mong muốn.

Ta có thể minh họa khái niệm giải thuật thông qua ví dụ đơn giản sau.

Ví dụ 9.1

```
<HTML>
<HEAD>
  <TITLE>Minh họa một giải thuật</TITLE>
  <script language="JavaScript">
    <!--
```

Chương 9

Câu lệnh trong JavaScript

```
1   var so_dua_vao, // la mot string
2       so1;      // so nguyen
      // Nhap vao so nguyen nhu mot string
3       so_dua_vao =
      window.prompt( "Nhap mot so nguyen","0" );
      // chuyen so tu string sang integer
4       so1 = parseInt( so_dua_vao );
      // dua nhan xet ra man hinh bang cau lenh if - else
5       if(so1%2==0)
        {
6           document.writeln( "<H1>So da nhap la so chan</H1>");
        }
7       else
        {
8           document.writeln("<H1>So da nhap la so le</H1>");
9       }
      // -->
</script>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Ví dụ 9.1 miêu tả một giải thuật đơn giản là nhập vào một số nguyên, và đưa ra thông báo kiểm tra xem đó là một số chẵn hay một số lẻ. Chương trình bao gồm một tập lệnh hữu hạn sau:

- Lệnh 1 và 2: khai báo hai biến.
- Lệnh 3: nhập vào một xâu số thông qua biến *so_dua_vao*.

Chương 9

Câu lệnh trong JavaScript

- Lệnh 4: chuyển xâu số vừa nhập sang số nguyên và gán giá trị đó cho biến *sol*.
- Lệnh 5, 6, 7, 8: tạo thành một khối lệnh để kiểm tra xem *sol* là chẵn hay lẻ và đưa ra thông báo phù hợp.

Khi trình duyệt biên dịch chương trình trên, nó sẽ kiểm tra từng lệnh một từ trên xuống dưới. Nếu như gặp câu lệnh sai, sẽ đưa ra thông báo và ngừng chạy. Chỉ cần chúng ta thay đổi trật tự của các câu lệnh trên thì chắc chắn chương trình sẽ không chạy được, hoặc cho kết quả không đúng như mong muốn.

Ví dụ trên đã cho chúng ta thấy rõ tầm quan trọng của việc xác định một giải thuật phù hợp khi bắt đầu viết một chương trình.

9.3. Cấu trúc điều khiển

Một cấu trúc logic đối với một thuật toán nhằm điều hành thứ tự mà các câu lệnh của chương trình được thực hiện. Các câu lệnh điều khiển sẽ điều hành sự diễn biến của quá trình điều khiển trong chương trình bằng cách xác định sự tuần tự các bước chương trình được tiến hành.

Các cấu trúc điều khiển gồm có:

- Cấu trúc rẽ nhánh (hay còn gọi là cấu trúc chọn lựa) làm cho một loạt các chỉ lệnh cụ thể được thực hiện nếu thoả mãn một điều kiện xác định nào đó.
- Cấu trúc vòng làm cho lệnh thực hiện lặp lại mãi cho đến khi hoàn thành xong một điều kiện
- Cấu trúc thủ tục/hàm sắp xếp các hàm chức năng hoặc các thủ tục khác nhau của chương trình vào trong các mô đun riêng biệt và sẽ được chương trình chính gọi đến khi cần thiết.

Trong JavaScript chúng ta tập trung chủ yếu nghiên cứu cấu trúc rẽ nhánh và cấu trúc vòng.

Cấu trúc rẽ nhánh	Cấu trúc vòng
<i>if</i>	<i>while</i>
<i>if...else</i>	<i>do...while</i>
<i>switch</i>	<i>for</i>
	<i>for...in</i>

Lưu đồ (Flowchart)

Trong cấu trúc điều khiển, người ta dùng một công cụ để mô tả thuật toán được gọi là **lưu đồ**. Lưu đồ thực chất là một biểu đồ minh hoạ cho dãy các thao tác cần tiến hành để

Chương 9

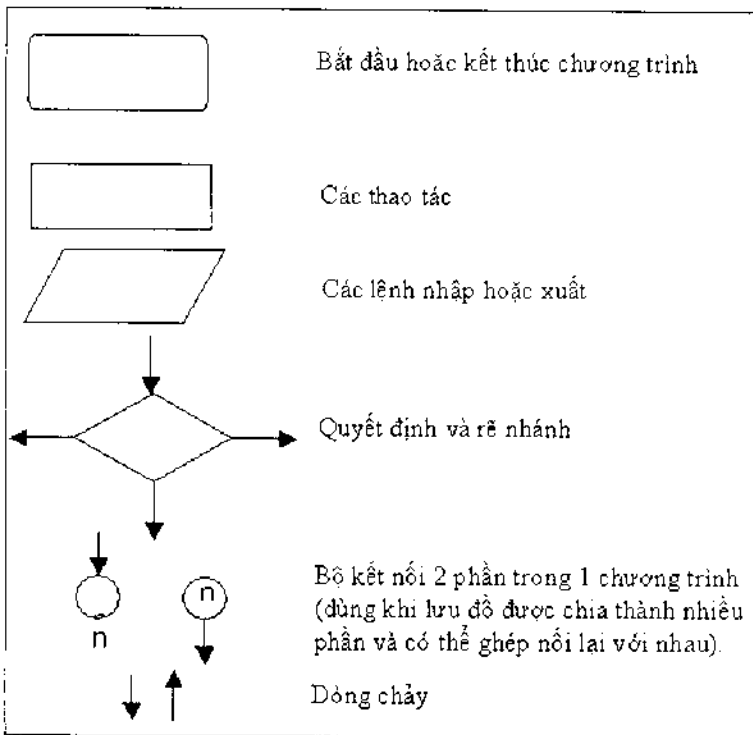
Câu lệnh trong JavaScript

giải quyết một vấn đề. Các câu lệnh được mô tả bằng các kí hiệu cụ thể. Các kí hiệu này được nối với nhau bằng các mũi tên để chỉ định rõ thứ tự thực hiện.

Lưu đồ là công cụ được sử dụng để viết chương trình và để phục vụ cho các mục đích sau:

- Mô tả bằng trực quan dễ hiểu hơn là mô tả tường thuật
- Dựa vào lưu đồ chúng ta có thể xem lại và gỡ rối chương trình một cách dễ dàng
- Cung cấp tài liệu chương trình
- Nhờ vẽ lưu đồ, dễ dàng hiểu được chương trình hoặc thảo luận giải pháp.

Các kí hiệu được sử dụng khi vẽ lưu đồ:



Hình 9.1. Các ký hiệu khi vẽ lưu đồ.

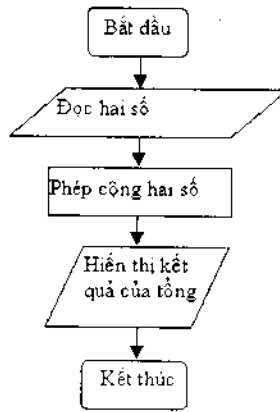
Những ví dụ sau đây giúp chúng ta hiểu rõ công dụng của từng kí hiệu được dùng trong lưu đồ

Ví dụ 9.2

Để tính tổng hai số ta có lưu đồ trong hình 9.2

Chương 9

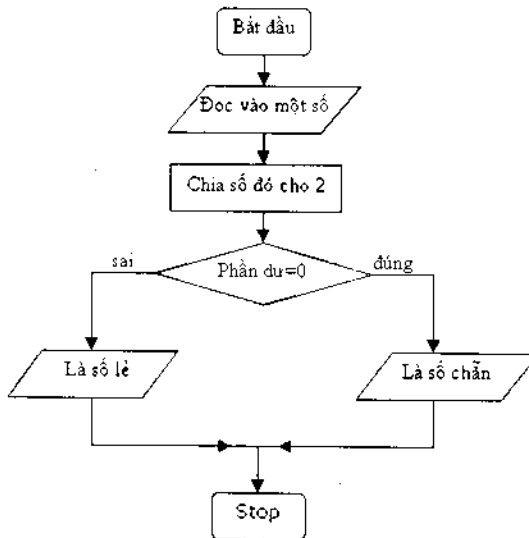
Câu lệnh trong JavaScript



Hình 9.2. Lưu đồ miêu tả quá trình thực hiện phép tính tổng hai số.

Ví dụ 9.3

Ví dụ này sẽ kiểm tra một số là số chẵn hay lẻ. Lưu đồ sử dụng hình 9.3 là lưu đồ rẽ nhánh (lưu đồ lựa chọn).



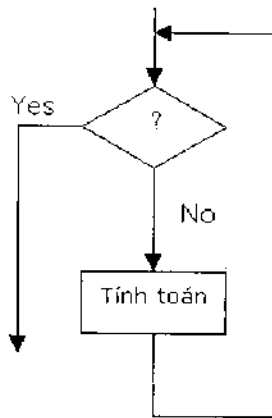
Hình 9.3. Lưu đồ kiểm tra xem một số là số chẵn hay số lẻ.

Ví dụ 9.4

Ví dụ này miêu tả lưu đồ minh họa cho một cấu trúc lặp thông thường.

Chương 9

Câu lệnh trong JavaScript



Hình 9.4. Lưu đồ của một cấu trúc lặp.

Chú ý: Cần ghi nhớ các điểm sau đây khi vẽ lưu đồ:

- Lúc đầu tập trung vào logic tổng quát của bài toán và vẽ các đường chính của lưu đồ.
- Sau khi đã hoàn tất xong các đường chính, vẽ thêm tất cả các nhánh và các vòng lặp.
- Một lưu đồ chỉ có duy nhất một điểm bắt đầu và một điểm kết thúc.
- Để cho lưu đồ có tính độc lập với chương trình, càng ít dùng các thuật ngữ liên quan với máy tính càng tốt.
- Không cần thiết phải mô tả từng bước của bài toán trong lưu đồ. Chỉ nêu các bước có ý nghĩa.
- Sử dụng các thuật ngữ mô tả thích hợp để trình bày logic của bài toán. Không sử dụng các thuật ngữ mơ hồ, khó hiểu.
- Đảm bảo người sử dụng hoặc lập trình viên khác dễ dàng hiểu được lưu đồ này.

Từ khóa

Khi làm việc với JavaScript chúng ta phải biết những từ khóa được sử dụng trong chương trình. Nhớ được những từ khóa này sẽ làm cho người lập trình không mắc những lỗi cơ bản khi viết lệnh, chẳng hạn đặt tên biến trùng với những từ khóa hoặc viết sai từ khóa,...

Bảng dưới đây chỉ ra những từ khóa mà JavaScript cung cấp và những từ khóa của các ngôn ngữ lập trình khác nhưng không được ngôn ngữ JavaScript hỗ trợ.

Chương 9

Câu lệnh trong JavaScript

Bảng 9.1. Từ khóa trong JavaScript

Từ khóa trong JavaScript.				
Break	case	continue	delete	do
Else	false	for	fuction	if
This	true	typeof	var	void
While	with			
Từ khóa không được hỗ trợ trong JavaScript.				
Catch	class	const	debugger	default
Enum	export	extends	finnaly	import
Super	try			

9.4. Câu lệnh điều kiện If

Câu lệnh *If* được dùng cho việc thực hiện chương trình dựa trên kết quả đánh giá một biểu thức. Nếu điều kiện đánh giá đúng thì một lệnh hay một khối lệnh được thực hiện.

Nếu không sử dụng câu lệnh điều kiện *if* thì chương trình không thể chuyển đến những lệnh khác mà phải thực hiện từng lệnh một theo thứ tự trên xuống. Bằng cách sử dụng biểu thức điều kiện, JavaScript có thể chuyển đến câu lệnh hay khối lệnh thỏa mãn điều kiện đưa ra.

Cấu trúc cơ bản của câu lệnh *if* trong JavaScript:

```
If < Dieukien> < Cau lenh>;  
  
//cau lenh se duoc thuc hien neu dieukien nhan gia tri true  
  
//neu dieukien nhan gia tri false thi thuc hien lenh ke tiep
```

Chú thích

Trước khi đi sâu vào hiểu rõ câu lệnh *if*, chúng ta sẽ nói về những dòng chú thích khi viết một chương trình JavaScript:

Là người lập trình, thông thường hẳn bạn đã nhận ra rằng các kết quả sinh ra thường không phù hợp với mục đích của bạn. Nói một cách đơn giản, có thể có các lỗi trong chương trình. Ngoài ra nếu bạn muốn khắc phục lỗi cho một chương trình của bạn hay của một người nào khác thì việc giải mã luồng chương trình lại tốn rất nhiều thời gian. Do vậy, một thói quen lập trình tốt là nên chèn các chú thích vào script. JavaScript sẽ bỏ qua các chú

Chương 9

Câu lệnh trong JavaScript

thích đó khi thực hiện chương trình. Ngoài ra các chú thích có thể giúp giải thích luồng chương trình, mô tả các câu lệnh, các hàm hay các biến.

- Nếu là một chú giải đơn lẻ trên một dòng thì nó sẽ nằm trong hai dấu gạch chéo lùi
// một câu chú thích trên một dòng đơn lẻ.
- Đối với các khối chú thích trên nhiều dòng, khởi đầu sẽ là /* và kết thúc với */.
/* Đây là nhưng dòng chú thích
trong JavaScript */

Ví dụ 9.5

Bây giờ chúng ta quay trở lại với câu lệnh *if* thông qua ví dụ sau:

```
<HTML>
<HEAD><TITLE>Cau lenh dieu kien if</TITLE>
<script language="JavaScript">
<!--
var day;
day=window.prompt("Nhap vao mot ngay trong tuan","Text");
if(day=="Thu bay")
    document.writeln("<H1>Hom nay la cuoi tuan</H1>");
if(day!="Thu bay")
document.writeln("<H1> Hom nay khong phai la thu bay </H1>");
// -->
</script>
</HEAD>
<BODY>
</BODY>
</HTML>
```

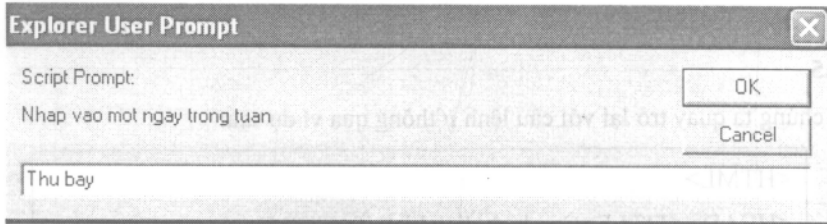
Ví dụ này cho phép người dùng nhập vào một ngày trong tuần, sau đó kiểm tra xem đó có phải là ngày cuối tuần hay không và đưa ra thông báo hợp lệ. Để làm được điều đó chương trình sử dụng hai câu lệnh điều kiện để kiểm tra một xâu nhập vào có trùng với xâu "Thu bay" không.

Chương 9

Câu lệnh trong JavaScript

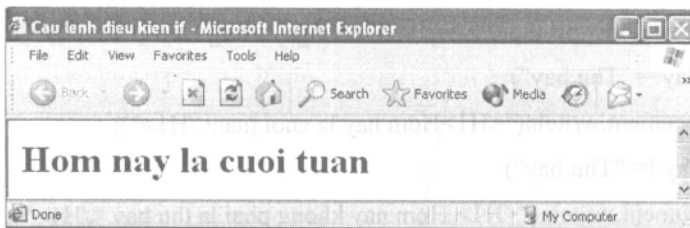
```
if(day=="Thu bay")
    document.writeln("<H1>Hom nay la cuoi tuan</H1>");
if(day != "Thu bay")
    document.writeln("<H1>Hom nay khong phai la thu bay</H1>");
```

Chạy chương trình, giao diện hộp thông báo hiện ra như hình 9.5



Hình 9.5. Giao diện hộp thông báo.

Chẳng hạn khi người dùng nhập vào hộp thoại prompt là “Thu bay”, bấm vào nút OK, trình duyệt đưa ra thông báo:



Hình 9.6. Thông báo hiện ra.

Trong cấu trúc lệnh điều kiện *if* của JavaScript, câu lệnh thực hiện có thể là nhiều lệnh khác nhau được đặt trong hai dấu { }.

```
if dieukien { khai lenh;}

//khai lenh se duoc thuc hien neu dieukien nhan gia tri true.

//neu dieukien nhan gia tri false thi thuc hien lenh ke tiep(khong lenh nao
trong khai lenh duoc thuc hien).
```

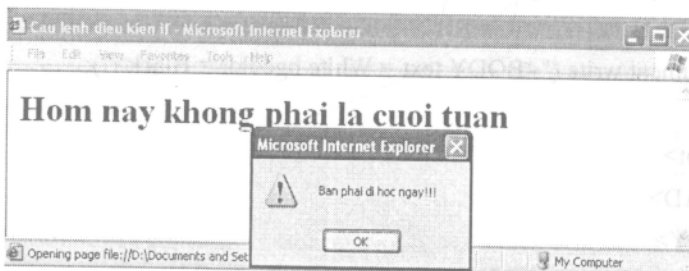
Ví dụ 9.6 : Trong ví dụ 9.6 sau đây, sau lệnh *if* là một khối lệnh.

Chương 9

Câu lệnh trong JavaScript

```
<HTML>
<HEAD><TITLE>Cau lenh dieu kien if</TITLE>
<script language="JavaScript">
<!--
var day;
day=window.prompt("Nhap vao mot ngay trong tuan","Text");
if(day!="Thu bay")
{
    document.writeln("<H1>Hom nay khong phai la cuoi tuan</H1>");
    alert("Ban phai di hoc ngay!!!");
}
// -->
</script>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Sau khi kiểm tra điều kiện, nếu điều kiện nhận giá trị true thì hai lệnh đặt trong cặp dấu ngoặc { } sẽ được thực hiện. Có nghĩa là khi bạn nhập vào trong hộp thoại xâu chữ khác với xâu “Thu bay” thì trình duyệt sẽ thực hiện đưa ra một dòng chữ và một hộp cảnh báo.



Hình 9.7. Sử dụng câu lệnh if.

9.5. Câu lệnh điều kiện if...else

Chúng ta thấy rằng, trong câu lệnh điều kiện ở mục 4 trên, các câu lệnh chỉ được thực hiện khi và chỉ khi điều kiện nhận giá trị true. Để chỉ ra lệnh hay khối lệnh cần được thực hiện nếu điều kiện đánh giá là sai, chúng ta dùng thêm mệnh đề *else*.

Cú pháp của câu lệnh như sau:

```
If (điều kiện)
{các câu lệnh; } //các câu lệnh voi điều kiện đúng.
else
{các câu lệnh2; } //các câu lệnh voi điều kiện sai.
```

Ví dụ 9.7

Trong ví dụ sau mã lệnh sẽ kiểm tra thời gian của hệ thống và tùy theo đó mà thực hiện script dưới nền xanh hay nền đen của trình duyệt:

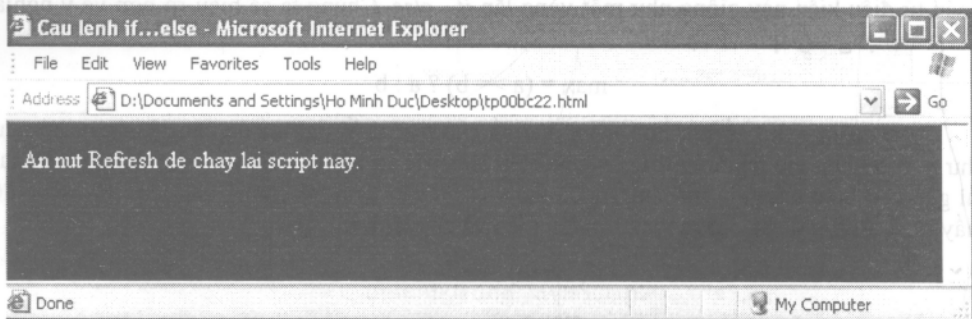
```
<HTML>
<TITLE>Câu lệnh if...else</TITLE>
<HEAD>
<script language= "JavaScript">
<!--
today = new Date( );// lấy ngày, tháng, thời gian của hệ thống.
minutes = today.getMinutes( );// hàm lấy phút
if (minutes >=0 && minutes <= 30)
    document.write ("<BODY text = White bgcolor = Blue>");
else
    document.write ("<BODY text = White bgcolor = Black>");
// -->
</script>
</HEAD>
<BODY>
<P>Ấn nút Refresh để chạy lại script này. <P>
</BODY>
</HTML>
```

Chương 9

Câu lệnh trong JavaScript

Trong ví dụ sử dụng câu lệnh *if...else* để xác định màu của trình duyệt là xanh hay đen. Khi biểu thức điều kiện (*minutes >= 0 && minutes <= 30*) nhận giá trị đúng thì màu nền của trình duyệt sẽ là màu xanh (*blue*) còn nếu biểu thức đó nhận giá trị sai thì màu nền của trình duyệt sẽ là màu đen.

Hình dưới đây minh họa khi biểu thức điều kiện nhận giá trị sai:



Hình 9.8. Sử dụng câu lệnh *if...else*.

Chú ý rằng chúng ta cũng có thể sử dụng cấu trúc lệnh *if ... else* lồng nhau.

Cú pháp như sau:

```
if (dieukien1) {  
    cac cau lenh;  
    if (dieukien2) {  
        cac cau lenh;  
    } else {  
        cac cau lenh;  
    }  
    cac cau lenh;  
} else {  
    cac cau lenh;  
}
```

Chương 9

Câu lệnh trong JavaScript

Toán tử điều kiện

Trong JavaScript cung cấp cho chúng ta một toán tử điều kiện. Ta nhắc lại cấu trúc cú pháp của toán tử này:

`(điều kiện) ? trueVal: falseVal;`

Toán tử điều kiện này giống như một vòng lặp *if...else*. Chúng ta sẽ hiểu rõ hơn về ý nghĩa của toán hạng này qua ví dụ sau:

```
max = (a >= b) ? a : b
```

Ví dụ này tìm số lớn hơn trong hai số cho trước. Ý nghĩa của câu lệnh này là: nếu như $a \geq b$ nhận giá trị đúng thì gán cho biến *max* giá trị bằng *a*, nếu $a \geq b$ nhận giá trị sai thì gán cho biến *max* giá trị bằng *b*. Có thể viết lại câu lệnh trên bằng câu lệnh *if...else* để thấy rõ sự tương đương giữa toán tử điều kiện và câu lệnh *if...else*:

```
if( a>=b)
    max = a;
else
    max =b;
```

Rõ ràng là toán tử điều kiện giúp chúng ta viết chương trình gọn gàng và nhanh hơn nhiều.

9.6. Câu lệnh lặp *while*

Câu lệnh *while* là một cấu trúc lặp khác trong JavaScript. Nó được dùng để thực hiện một khối lệnh chừng nào điều kiện còn đúng. Nếu có nhiều câu lệnh được thực hiện trong phần thân (BODY) của vòng lặp thì các đoạn của chương trình phải được bao bởi một đôi ngoặc { }.

Cú pháp của câu lệnh *while* như sau:

```
while (dieukien)
{
    cac cau lenh;
}
```

Ví dụ 9.8

Ví dụ sau đây minh họa cho một vòng lặp *while* để tính trung bình của 10 số do người dùng nhập vào.

Chương 9

Câu lệnh trong JavaScript

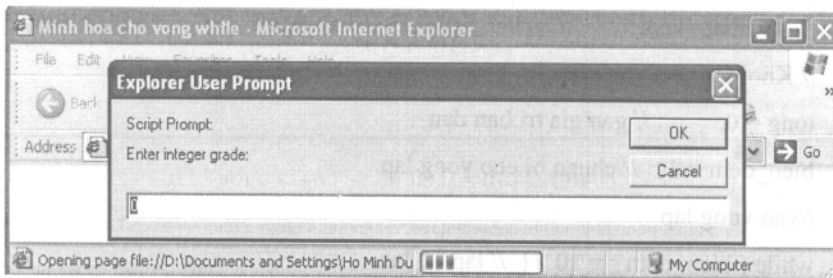
```
<HTML>
<HEAD>
<TITLE>Minh hoa cho vong while</TITLE>
<script language = "JavaScript">
  <!--
  var tong,      // tong cua cac so
      bien_dem, // dem xem nhap bao nhieu chu so
      so,        // gia tri nguyen
      trung_binh, // trung binh cua cac so
      so_nhap_vao; // string
  // Khoi tao cho vong lap
  tong = 0;      // gan gia tri ban dau
  bien_dem = 1; // chuan bi cho vong lap
  //vao vong lap
  while ( bien_dem <= 10 ) { // lap 10 lan
    // prompt de nhap so
    so_nhap_vao = window.prompt( "Enter integer grade:", "0" );
    // chuyen tu string sang integer
    so = parseInt( so_nhap_vao );
    // cong vao tong
    tong = tong + so;
    // tang bien dem len 1
    bien_dem = bien_dem + 1;
  }
  trung_binh = tong / 10; // tinh trung binh
  // hien thi gia tri trung binh
  document.writeln(
```

Chương 9

Câu lệnh trong JavaScript

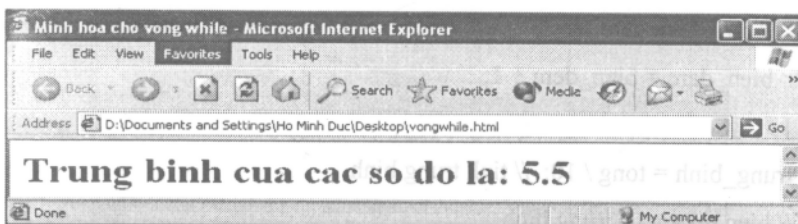
```
"<H1>Trung binh cua cac so do la: " + trung_binh + "</H1>" );  
// -->  
</script>  
</HEAD>  
<BODY>  
  <P>An nut Refresh de chay lai script nay</P>  
</BODY>  
</HTML>
```

Khi thực hiện, hộp thoại sau sẽ xuất hiện 10 lần liên tiếp cho người sử dụng nhập vào 10 số:



Hình 9.9. Kết quả sử dụng câu lệnh lặp While với vòng lặp cố định.

Chỉ khi người dùng nhập đúng 10 số vào 10 hộp thoại đó thì chương trình mới tiếp tục. Vòng lặp *while* thực hiện kiểm tra *bien_dem* lớn hơn 10 có nghĩa là giá trị của biểu thức điều kiện là *false*, ngay lập tức vòng lặp sẽ ngừng và chuyển sang lệnh tiếp theo. Nếu không sử dụng câu lệnh lặp *while* thì chúng ta phải viết tới 10 dòng lệnh tương tự nhau để nhập vào 10 số. Sau khi thoát khỏi vòng lặp *while*, chương trình tiếp tục đến câu lệnh kế tiếp như bình thường. Chẳng hạn khi người dùng nhập vào 10 số nguyên từ 1 cho đến 10, trình duyệt sẽ trả kết quả trung bình của mười số nguyên đó:



Hình 9.10. Kết quả trung bình của mười số nguyên.

Chương 9

Câu lệnh trong JavaScript

Vòng lặp while với số vòng lặp không cố định.

Ở ví dụ trên vòng lặp *while* phải thực hiện đúng 10 lần khi người dùng nhập vào đúng 10 số. Hay còn gọi là vòng lặp với số lần lặp xác định. Điều đó nhiều lúc gây phiền phức cho người sử dụng khi mà người ta không muốn tiếp tục thực hiện chương trình đó. Trong những trường hợp như vậy chúng ta có thể sử dụng *vòng lặp while với số vòng lặp không cố định*.

Vòng lặp không cố định tức là vòng lặp mà số lần lặp không được xác định trước. Người dùng chỉ có thể thoát khỏi vòng lặp khi điền vào một giá trị thích hợp thỏa mãn điều kiện của vòng lặp.

Ví dụ 9.9

Ví dụ sau sẽ minh họa cho chúng ta một vòng lặp không cố định.

```
<HEAD>
<TITLE>vong lap while</TITLE>
<script language = "JavaScript">
  <!--
    var bien_dem, // so luong so nhap vao
        so,      // gia tri nguyen
        tong,    // tong cua cac so
        trung_binh, // trung binh cac so
        so_nhap_vao; // la mot string

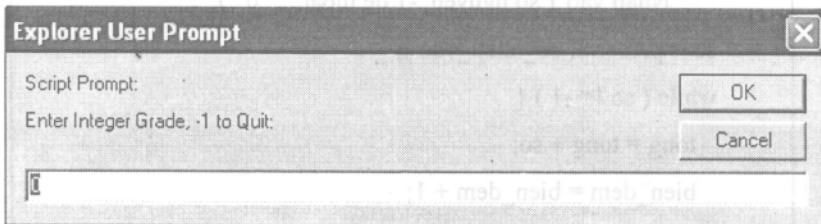
    tong = 0;
    bien_dem = 0;
    so_nhap_vao = window.prompt(
        "Nhap vao 1 so nguyen, -1 de thoát:", "0" );
    so = parseInt( so_nhap_vao );
    while ( so != -1 ) {
        tong = tong + so;
        bien_dem = bien_dem + 1;
        so_nhap_vao = window.prompt(
```

Chương 9

Câu lệnh trong JavaScript

```
"Nhập vào một số nguyên, -1 để thoát."; "0" );
so = parseInt( so_nhập_vào );
}
if ( bien_dem != 0 ) {
    trung_binh = tong / bien_dem;
    document.writeln(
        "<H1>Trung bình các số là : " + trung_binh + "</H1>" );
}
else
    document.writeln( "<P>Không có số nào được nhập
vào</P>" );
// -->
</script>
</HEAD>
<BODY>
    <P>An nút refresh để chạy lại script này</P>
</BODY>
</HTML>
```

Điểm khác biệt trong ví dụ 9.9 so với ví dụ 9.8. Đó là vòng lặp *while* không xác định số lần lặp. Có thể câu lệnh trong vòng lặp *while* không được thực hiện lần nào nếu như người dùng nhập vào hộp thoại giá trị -1, hoặc cũng có thể là không ngừng nếu nhập vào giá trị khác -1. Hộp thoại *prompt* sau sẽ xuất hiện sau mỗi lần nhập giá trị thỏa mãn điều kiện.



Hình 9.11. Hộp thoại prompt.

Chương 9

Câu lệnh trong JavaScript

Giả sử hộp thoại đó sẽ xuất hiện 4 lần khi người sử dụng nhập vào 4 giá trị theo thứ tự sau: 97, 88, 72 và -1. Vòng lặp kết thúc sau khi nhập và đưa kết quả ra:



Hình 9.12. Kết quả trung bình.

9.7. Toán tử gán

Trong phần trước chúng ta đã nhắc về toán tử gán trong JavaScript. Chúng ta sẽ nói rõ hơn về toán tử này.

Toán tử gán (=) được dùng để gán một giá trị cho một biến và có thể coi là toán tử được dùng nhiều nhất trong JavaScript. Bạn có thể gán một giá trị tùy ý cho một biến, bao gồm cả gán một biến khác (mà giá trị của biến đó sẽ được gán).

Bạn có thể dùng toán tử gán theo nhiều cách:

- $A = 10$;
- $Total = 100$

$Total = (Mathematics + Physics + Chemistry)$

Bạn có thể dùng các toán tử gán tốc ký để thực hiện một tác vụ và gán kết quả của nó cho một biến trong một bước, ví dụ: $a += 3$ cũng tương tự như $a = a + 3$. Trong JavaScript cung cấp cho ta các toán tử gán tốc ký. Các toán tử này làm cho biểu thức viết ngắn gọn và dễ hiểu hơn.

Các toán tử gán trong JavaScript được liệt kê dưới bảng sau.

Bảng 9.2. Các toán tử gán trong JavaScript

Các toán tử tốc ký	Ý nghĩa
$A += B$	$A = A + B$
$A -= B$	$A = A - B$
$A *= B$	$A = A * B$
$A /= B$	$A = A / B$

Chương 9

Câu lệnh trong JavaScript

$A \% = B$	$A = A \% B$
$A \& = B$	$A = A \& B$
$A \wedge = B$	$A = A \wedge B$
$A = B$	$A = A B$

Ví dụ 9.10

Ví dụ sau sẽ minh họa chương trình sử dụng toán hạng gán. Qua đó chúng ta thấy câu lệnh chương trình ngắn gọn hơn và súc tích hơn

```
<HTML>
<HEAD>
  <TITLE>Minh hoa toan tu gan</TITLE>
  <script language="JavaScript">
    <!--
      var dau = 0,    // so luong SV dau
      trat = 0,    // so luong trat
      so_sinh_vien = 1,
      ket_qua;
      while ( student <= 10 ) {
        ket_qua = window.prompt(
          "Nhap ket qua vao (1=qua,2=truot)", "0" );
        if ( ket_qua == "1" )
          qua += 1;
        else
          trat += 1;
        so_sinh_vien += 1;
      }
```

Chương 9

Câu lệnh trong JavaScript

```
document.writeln( "<H1>Ket qua kiem tra</H1>" );
document.writeln(
    "Qua: " + qua + "<br />Trat: " + trat );
if ( qua > 8 )
    document.writeln( "<br />Chat luong giang day tang len" );
// -->
</script>
</HEAD>
<BODY>
    <P>An nut Refresh de chay lai script nay</P>
</BODY>
</HTML>
```

Để thấy rõ ý nghĩa của các toán tử gán, bạn có thể xem xét mô tả sau:

Toán tử gán	Biến khởi tạo	Biểu thức	Giải thích	Gán
<code>+=</code>	<code>c = 3</code>	<code>c += 7</code>	<code>c = c + 7</code>	<code>c = 10</code>
<code>-=</code>	<code>d = 5</code>	<code>d -= 4</code>	<code>d = d - 4</code>	<code>d = 1</code>
<code>*=</code>	<code>e = 4</code>	<code>e *= 5</code>	<code>e = e * 4</code>	<code>e = 20</code>
<code>/=</code>	<code>f = 6</code>	<code>f /= 3</code>	<code>f = f / 3</code>	<code>f = 2</code>
<code>%=</code>	<code>g = 12</code>	<code>g *= 9</code>	<code>g = g % 9</code>	<code>g = 3</code>
Toán tử gán số học.				

9.8. Toán tử tăng và giảm

Toán tử tăng: Tăng một ngôi.

Toán tử này lấy một toán hạng, giá trị của toán hạng đó được tăng lên một. Giá trị trả về tùy thuộc toán tử `++` được đặt trước hay sau toán hạng.

- `++x` sẽ trả về giá trị của `x` sau khi tăng lên 1
- `x++` sẽ trả về giá trị của `x` trước khi tăng lên 1

Chương 9

Câu lệnh trong JavaScript

Toán tử giảm: Giảm một ngôi.

Toán tử này chỉ lấy một toán hạng. Giá trị trả về tùy thuộc toán tử được đặt trước hay sau toán hạng.

- `--x` sẽ trả về giá trị của `x` sau khi giảm 1
- `x--` sẽ trả về giá trị của `x` trước khi giảm 1

Ví dụ 9.11.

Ví dụ sau đây sẽ minh họa cho việc sử dụng toán tử tăng và giảm.

```
<HTML>
<HEAD>
  <TITLE>Minh hoa toan tu tang</TITLE>
  <script language="JavaScript">
    <!--
    var c;
    c = 5;
    document.writeln( "<h3>Postincrementing</h3>" );
    document.writeln( c );      // in ra 5
    // print 5 then increment
    document.writeln( "<br />" + c++ );
    document.writeln( "<br />" + c ); // in ra 6
    c = 5;
    document.writeln( "<h3>Preincrementing</h3>" );
    document.writeln( c );      // in 5
    // increment then print 6
    document.writeln( "<br />" + ++c );
    document.writeln( "<br />" + c ); // in 6
    // -->
  </script>
</HEAD><BODY></BODY>
</HTML>
```

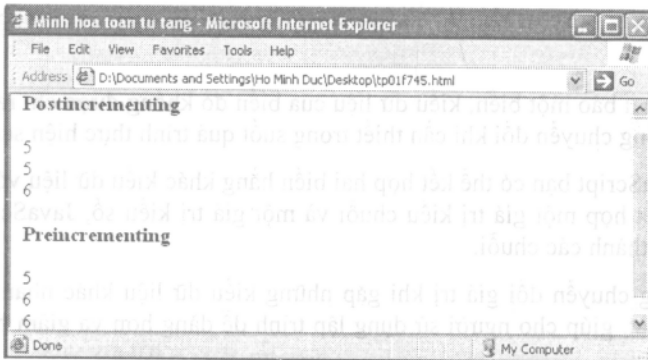
Chương 9

Câu lệnh trong JavaScript

Ví dụ này chỉ ra sự khác biệt giữa hai toán tử `c++`, `++c`.

- Giá trị khởi đầu của biến `c` là `c = 5`.
- Sau khi gặp câu lệnh `c++` thì giá trị của `c` lúc này vẫn là 5 bởi vì phép toán `c++` trả về giá trị của `c` trước khi tăng lên 1. Vì vậy câu lệnh: `document.writeln( "<br />" + c++ );` vẫn in ra giá trị 5
- Câu lệnh tiếp theo: `document.writeln( "<br />" + c );` in ra giá trị là 6.
- Chúng ta lại gán giá trị cho `c` là `c = 5`. Sau khi gặp câu lệnh `++c` thì giá trị `c` trả về sẽ là: `( c = c + 1 )` bằng 6. Vì vậy mà câu lệnh `document.writeln( "<br />" + ++c );` sẽ in kết quả ra màn hình giá trị của `c` là 6.

Kết quả của bài toán trên có thể thấy khi chạy trên trình duyệt web như sau:



Hình 9.13. Toán tử tăng và giảm.

Thứ tự ưu tiên của toán tử

Khi tính toán với một biểu thức có sử dụng toán tử tăng và giảm thì thứ tự ưu tiên để thực hiện cũng phải tuân thủ theo như những toán tử khác. Thứ tự ưu tiên của toán tử tăng, giảm so với những toán tử thông thường khác được chỉ ra trong bảng sau (theo thứ tự từ cao xuống thấp, những toán tử ở trong cùng một hàng là cùng mức độ).

Bảng 9.3. Thứ tự ưu tiên của toán tử trong JavaScript

Toán tử	Thứ tự thực hiện
<code>++ --</code>	Từ trái qua phải.
<code>* / %</code>	Từ trái qua phải
<code>+ -</code>	Từ trái qua phải
<code>< <= > >=</code>	Từ trái qua phải

Chương 9

Câu lệnh trong JavaScript

<code>== !=</code>	Từ trái qua phải
<code>?:</code>	Từ phải qua trái
<code>= += -= *= /= %=</code>	Từ phải qua trái

9.9. Chú ý về kiểu dữ liệu

Ở chương trước chúng ta đã nói về những kiểu dữ liệu trong JavaScript. Trong mục này chúng ta chỉ nhắc lại một số điểm khi thao tác với các kiểu dữ liệu để nhấn mạnh thêm sự đa dạng và mạnh mẽ khi thao tác với các kiểu dữ liệu trong JavaScript.

- Dữ liệu luôn thuộc một kiểu nào đó.
- Các biến nhận kiểu dữ liệu mà chúng chứa, và thay đổi kiểu khi dữ liệu trong chúng thay đổi.
- Khi bạn khai báo một biến, kiểu dữ liệu của biến đó không được chỉ ra. Kiểu dữ liệu được tự động chuyển đổi khi cần thiết trong suốt quá trình thực hiện script.
- Trong JavaScript bạn có thể kết hợp hai biến hằng khác kiểu dữ liệu với nhau. Ví dụ, nếu bạn kết hợp một giá trị kiểu chuỗi và một giá trị kiểu số, JavaScript sẽ chuyển đổi các số thành các chuỗi.

Sự tự động chuyển đổi giá trị khi gặp những kiểu dữ liệu khác nhau mang lại tính mềm dẻo, linh hoạt, giúp cho người sử dụng lập trình dễ dàng hơn và giảm bớt sai lầm khi viết chương trình.

9.10. Kết chương

Trong chương này chúng ta đã đi vào nghiên cứu tập lệnh của JavaScript, giải thuật, câu lệnh điều kiện *If, if... else*, toán tử gán, toán tử tăng và giảm. Trong chương sau chúng ta tiếp tục tìm hiểu về cấu trúc vòng và những câu lệnh như *for, do...while...* trong JavaScript

Chương 10

Câu Lệnh trong JavaScript (tiếp theo)

Nội dung của chương:

- Tìm hiểu về câu lệnh lặp *for*.
- Câu lệnh lặp *do...while*.
- Câu lệnh *switch*.
- Câu lệnh *break*.
- Câu lệnh *continue*.
- Toán tử logic.

10.1. Giới thiệu

Ở chương trước chúng ta đã tìm hiểu một số lệnh cơ bản dùng trong chương trình JavaScript. Ở chương này chúng ta tiếp tục nghiên cứu thêm những câu lệnh lặp trong JavaScript. Câu lệnh lặp thể hiện việc lặp đi lặp lại một đoạn mã cho đến khi biểu thức điều kiện được đánh giá là đúng. Ngoài câu lệnh *while* mà ta đã xét ở chương trước. Trong chương này còn có những câu lệnh lặp *for* và *do ...while*. Đây cũng là những câu lệnh điều khiển gần giống với câu lệnh *while*.

Bên cạnh những câu lệnh đó, trong chương này chúng ta cũng sẽ học về cấu trúc và ý nghĩa của những câu lệnh sau: *switch*, *break*, *continue*. Những câu lệnh này giúp chúng ta điều khiển được những luồng script theo một điều kiện nhất định nào đó.

Với những câu lệnh trên, chúng ta sẽ viết được những chương trình trong JavaScript ngắn gọn và dễ dàng hơn.

10.2. Bản chất của vòng lặp điều khiển đếm

Trong chương 9 ta đã tìm hiểu về cấu trúc của một vòng lặp với câu lệnh lặp *while*. Ta sẽ nói rõ về bản chất của một vòng lặp điều khiển đếm.

Chương 10

Câu lệnh trong JavaScript (tiếp theo)

Trong một vòng lặp điều khiển đếm bao giờ cũng phải gồm những yếu tố sau.

- Gán một giá trị khởi đầu cho một biến đếm, giá trị khởi đầu này có tác dụng kiểm tra điều kiện đầu tiên khi bắt đầu vào vòng lặp. Thông thường giá trị gán này nhằm mục đích để vòng lặp thực hiện ít nhất là một lần.
- Trong biểu thức điều kiện phải có một giá trị cố định để vòng lặp không lặp vô hạn. Biến đếm bao giờ cũng được kiểm tra có thỏa mãn biểu thức điều kiện hay không trước khi bắt đầu thực hiện một lần lặp tiếp.
- Giá trị biến đếm phải thay đổi trong vòng lặp. Thông thường giá trị biến thay đổi trong những biểu thức hoặc là những toán tử trong JavaScript. Diễn hình nhất và hay sử dụng nhất khi làm thay đổi giá trị của biến đếm là toán tử tăng và giảm.
- Khi đến một giá trị nào đó của biến đếm mà không thỏa mãn điều kiện của vòng lặp thì vòng lặp sẽ dừng lại. Và giá trị đó gọi là giá trị cuối cùng của biến đếm. Sau khi “thoát khỏi” vòng lặp, giá trị của biến đếm là giá trị cuối cùng.

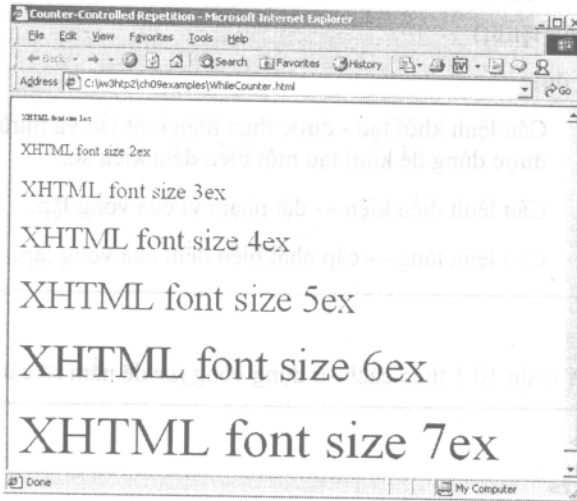
Ví dụ 10.1

Chúng ta sẽ minh họa những đặc điểm trên của vòng lặp điều khiển đếm qua ví dụ sau.

```
<HTML>
<HEAD>
  <TITLE>Counter-Controlled Repetition </TITLE>
  <script language = "JavaScript">
    <!--
      var bien_dem = 1;          // khoi tao bien dem

      while ( bien_dem <= 7 ) { // dieu kien cua vong lap
        document.writeln( "<P style = \"font-size: " +
          bien_dem + "ex\">XHTML font size " + bien_dem +
          "ex</P>" );
        ++bien_dem;             // toan tu tang
      }
    <!-->
  </script>
</HEAD><BODY></BODY>
</HTML>
```

Kết quả của ví dụ 10.1 thể hiện trong trình duyệt:



Hình 10.1. Minh họa vòng lặp điều khiển đếm.

Trong ví dụ 10.1, biến dùng để đếm là *bien_dem*, biến đếm được khởi tạo bằng 1 ở ngoài vòng lặp bằng câu lệnh:

```
var bien_dem = 1; // khởi tạo biến đếm
```

Điều kiện để kiểm tra vòng lặp là (*bien_dem* <= 7). Khi kiểm tra biến đếm mà <=7 thì vòng lặp vẫn tiếp tục. Giá trị của *bien_dem* được thay đổi bằng toán tử tăng: ++*bien_dem*. Giá trị của biến *bien_dem* sẽ tự động tăng lên 1 sau khi một vòng lặp kết thúc. Vòng lặp sẽ thực hiện lần cuối cùng khi giá trị của biến *bien_dem* bằng 7. Kết thúc lần lặp này giá trị của biến đếm sẽ là 8 và do đó không thỏa mãn điều kiện của vòng lặp. Vì vậy vòng lặp sẽ kết thúc. Chương trình sẽ tiếp tục với câu lệnh tiếp theo ngay sau vòng while.

10.3. Câu lệnh lặp for

Câu lệnh *for* là câu lệnh phức tạp và mạnh nhất trong tất cả các cấu trúc điều khiển luồng trong JavaScript. Vòng lặp *for* chạy lặp lại một khối lệnh trong khi một điều kiện kiểm tra là đúng. Thông thường, số lần lặp phụ thuộc vào một biến đếm.

Câu lệnh *for* bao gồm ba tham số được phân cách với nhau bằng các dấu chấm phẩy. Cả ba tham số này đều không bắt buộc và chúng kiểm soát việc thực hiện một vòng lặp.

Cấu trúc vòng lặp vòng *for* như sau:

Chương 10

Câu lệnh trong JavaScript (tiếp theo)

for (câu lệnh khởi tạo; điều kiện; tăng)

{các câu lệnh;}

Trong đó:

- Câu lệnh khởi tạo - được thực hiện một lần và thường được dùng để khởi tạo một biến đếm kiểu số.
- Câu lệnh điều kiện - - đặt phạm vi của vòng lặp.
- Câu lệnh tăng - - cập nhật biến đếm của vòng lặp.

Ví dụ 10.2

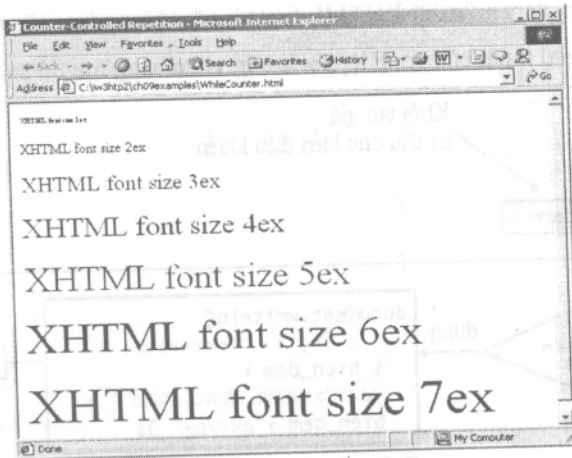
Chúng ta sẽ viết lại ví dụ 10.1 theo cách sử dụng vòng *for* để nắm rõ cấu trúc của nó.

```
<HTML>
<HEAD>
  <TITLE>Counter-Controlled Repetition</TITLE>
  <script language="JavaScript">
    <!--
    // Khởi tạo, biểu thức điều kiện và
    // toán tử tăng và lệnh trong vòng for
    for ( var bien_dem = 1; bien_dem <= 7; ++bien_dem )
      document.writeln( "<P style = \"font-size: " +
        bien_dem + "ex\">XHTML font size " + bien_dem +
        "ex</P>" );
    // -->
  </script>
</HEAD><BODY></BODY>
</HTML>
```

Kết quả trên trình duyệt web cho giá trị không khác so với ví dụ 10.1

Chương 10

Câu lệnh trong JavaScript (tiếp theo)



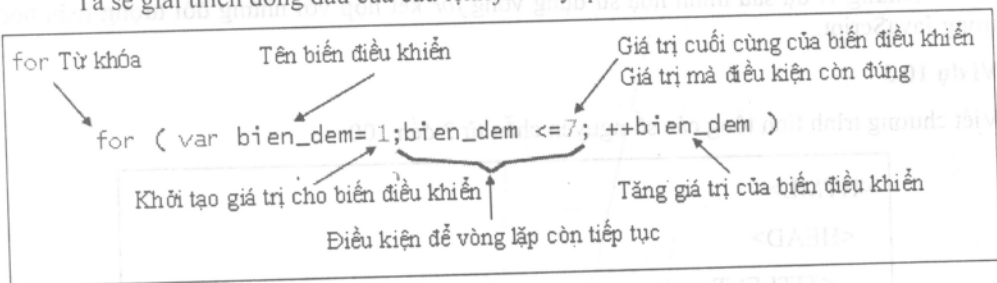
Hình 10.2. Minh họa sử dụng vòng lặp for.

Bây giờ chúng ta sẽ phân tích cú pháp của câu lệnh *for* sử dụng trong ví dụ trên.

- Dòng đầu tiên của câu lệnh *for*:

```
for ( var bien_dem = 1; bien_dem <= 7; ++bien_dem )
```

Ta sẽ giải thích dòng đầu của câu lệnh *for* này qua hình vẽ sau:



Hình 10.3. Câu lệnh for.

- Biến điều khiển được khai báo là: *bien_dem* và được khởi tạo giá trị bằng 1.
- Vòng lặp sẽ tiếp tục cho đến khi biểu thức điều kiện *bien_dem <=7*.
- Sau mỗi lần lặp giá trị của biến đếm sẽ tăng lên 1 bằng toán tử tăng *++bien_dem*.
- Khi biểu thức điều kiện nhận giá trị đúng, thì câu lệnh trong vòng *for* sẽ được thực hiện:

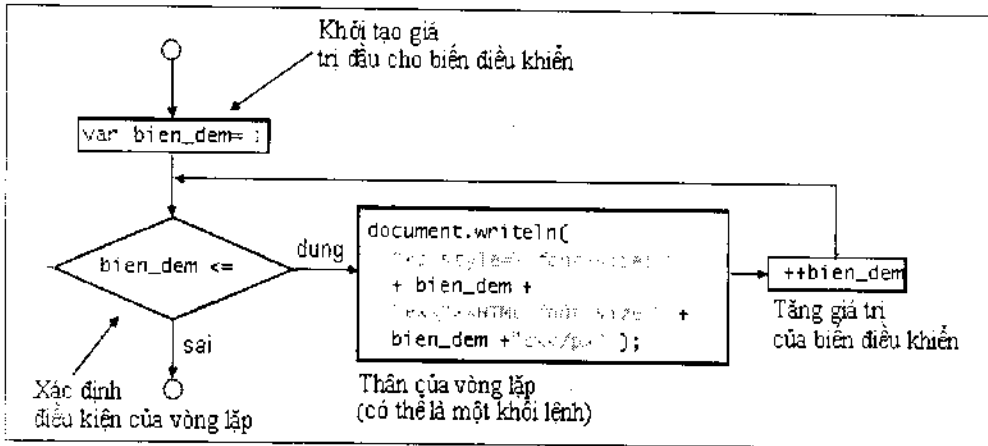
```
document.writeln( "<P style = \"font-size: " +
```

Chương 10

Câu lệnh trong JavaScript (tiếp theo)

```
bien_dem + "ex\\>XHTML font size " + bien_dem + "ex</P>" );
```

Có thể minh hoạ giải thuật của vòng lặp *for* trong ví dụ bằng lưu đồ sau:



Hình 10.4. Minh hoạ cho giải thuật cho vòng lặp *for*.

Giá trị cuối cùng của biến điều khiển *bien_dem* là bằng 8.

10.4. Những ví dụ sử dụng câu lệnh lặp *for*

Những ví dụ sau minh hoạ sử dụng vòng *for* kết hợp với những đối tượng toán học trong JavaScript.

Ví dụ 10.3

Viết chương trình tính tổng các số nguyên chẵn từ 2 đến 100.

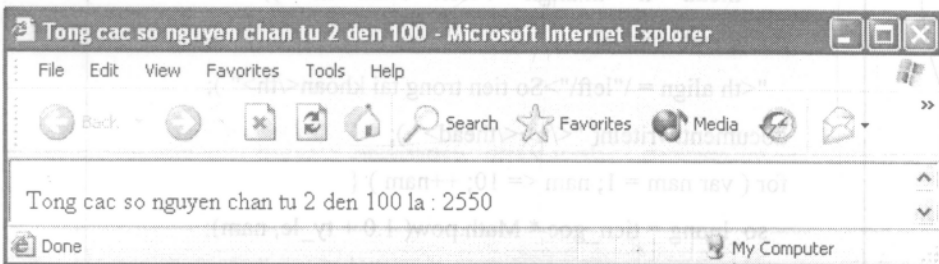
```
<HTML>
<HEAD>
  <TITLE>Tong cac so nguyen chan tu 2 den 100</TITLE>
  <script language = "JavaScript">
    <!--
    var tong = 0;
    for ( var so_nguyen = 2; so_nguyen <= 100; so_nguyen += 2 )
      tong += so_nguyen;
    document.writeln( "Tong cac so nguyen chan " +
```

Chương 10

Câu lệnh trong JavaScript (tiếp theo)

```
"tu 2 den 100 la : " + tong );  
  
// -->  
  
</script>  
  
</HEAD><BODY></BODY>  
  
</HTML>
```

Ví dụ 10.3. là một ví dụ rất điển hình của câu lệnh lặp vòng for. Kết quả đưa ra màn hình là 2550.



Hình 10.5. Kết quả sử dụng vòng lặp for.

Ví dụ 10.4

Viết chương trình tính tổng số tiền gửi tiết kiệm với một tỷ lệ gửi xác định. Chương trình sẽ in ra số tiền qua từng năm một. Thời gian gửi tiết kiệm là không quá 10 năm.

Để giải quyết bài toán này, chúng ta cần phải khởi tạo giá trị cho ba biến:

- Biến thứ nhất: *so_luong*: để ghi lại số lượng tiền sau mỗi năm gửi vào.
- Biến thứ hai: *tien_goc* = 1000.0: chỉ ra số tiền gửi vào lúc đầu.
- Biến thứ ba: *ty_le* = 0.05 chỉ lãi suất của mỗi năm.

Thực hiện tính lãi suất của từng năm, sau đó cộng với số tiền có được của năm kế trước thì ta được số tiền mà người gửi có tại thời điểm hiện tại.

```
<HTML>  
  
<HEAD>  
  
<TITLE>Tính tiền lai kep(von va lai)</TITLE>  
  
<script language ="JavaScript">  
  
<!--
```

Chương 10

Câu lệnh trong JavaScript (tiếp theo)

```
var so_luong, tien_goc = 1000.0, ty_le = .05;
document.writeln(
    "<table border = '1' width = '100%'>");
document.writeln(
    "<caption>Tính tiền lãi kép</caption>");
document.writeln(
    "<thead><tr><th align = 'left'>Nam</th>");
document.writeln(
    "<th align = 'left'>So tiền trong tài khoản</th>");
document.writeln( "</tr></thead>");
for ( var nam = 1; nam <= 10; ++nam ) {
    so_luong = tien_goc * Math.pow( 1.0 + ty_le, nam);
    document.writeln( "<tbody><tr><td>" + year +
        "</td><td>" + Math.round( so_luong * 100 ) / 100 +
        "</td></tr>" );
}
document.writeln( "</tbody></table>" );
// -->
</script>
</HEAD><BODY></BODY>
</HTML>
```

Trong ví dụ này chúng ta kết hợp sử dụng vòng *for* và đối tượng toán học. Đó là đối tượng *Math* với hai hàm *Math.pow* và *Math.round*. *Math.pow* là một hàm toán học được xây dựng trong thư viện của JavaScript.

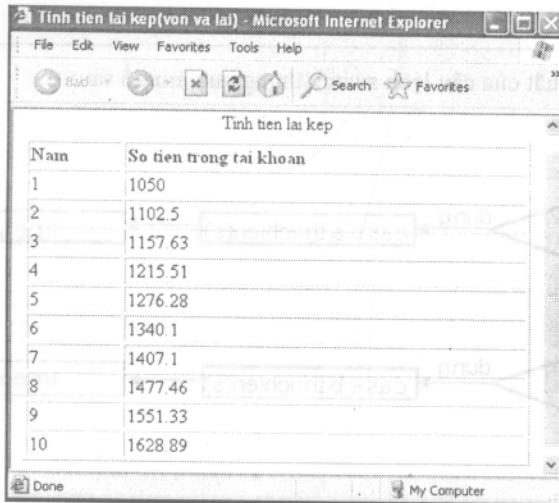
- *Math.pow (base,exponent)*: Trả về giá trị *base* lũy thừa *exponent*.
- *Math.round(number)*: trả lại giá trị của *number* làm tròn tới số nguyên gần nhất.

Chương 10

Câu lệnh trong JavaScript (tiếp theo)

Số tiền qua 10 năm gửi với lãi suất 0.05 và tiền vốn ban đầu là 1000.0 được đưa ra sau khi thực hiện chương trình như trên hình 10.6.

Trên đây chúng ta đã chỉ ra hai ví dụ rất đặc trưng sử dụng vòng lặp *for*. Vòng lặp *for* được sử dụng rất nhiều trong JavaScript.



Năm	Số tiền trong tài khoản
1	1050
2	1102.5
3	1157.63
4	1215.51
5	1276.28
6	1340.1
7	1407.1
8	1477.46
9	1551.33
10	1628.89

Hình 10.6. Sử dụng vòng lặp *for* với đối tượng toán học.

10.5. Câu lệnh *switch*

Nếu bạn có vô số tùy chọn *If ... else*, tốt hơn hết là bạn nên dùng câu lệnh *switch*. Còn gọi là câu lệnh chọn, câu lệnh *switch* thực hiện một khối lệnh trong nhiều khối lệnh tùy thuộc vào giá trị của một biểu thức. Nếu không tìm thấy giá trị phù hợp trong biểu thức thì khối lệnh mặc định được thực hiện. Câu lệnh *break*; được dùng để ngăn ngừa không thực hiện các câu lệnh nằm sau giá trị phù hợp tìm thấy.

Cấu trúc của câu lệnh *switch*:

```
Switch (biểu thức) {
```

```
case label:
```

```
  câu lệnh;
```

```
break;
```

```
case label:
```

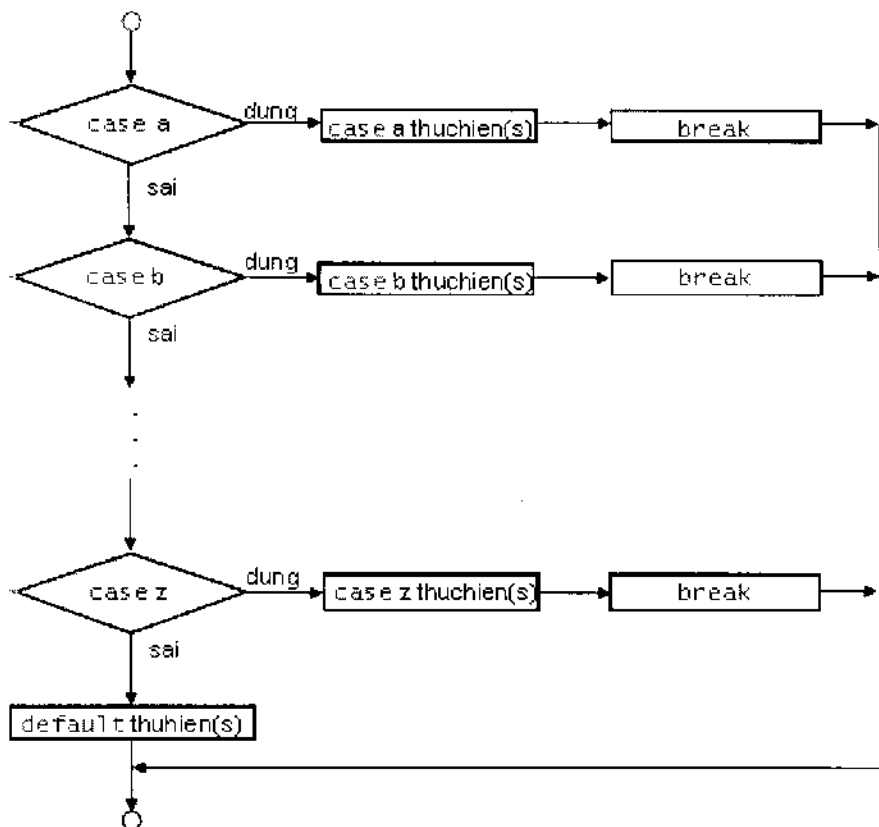
```
  câu lệnh;
```

Chương 10

Câu lệnh trong JavaScript (tiếp theo)

```
break;  
...  
default: câu lệnh;  
}
```

Có thể hiểu rõ giải thuật của câu lệnh switch thông qua lưu đồ sau:



Hình 10.7. Lưu đồ câu lệnh switch.

Chúng ta sẽ minh họa câu lệnh *switch* qua ví dụ dưới đây.

Ví dụ 10.5

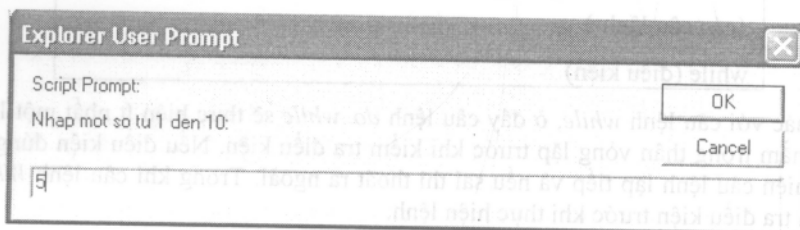
Minh họa một chương trình nhỏ, khi người dùng nhập một số từ 1 đến 10 thì ngay lập tức chương trình sẽ hiển thị số đó ra trình duyệt bằng tiếng Anh.

Chương 10

Câu lệnh trong JavaScript (tiếp theo)

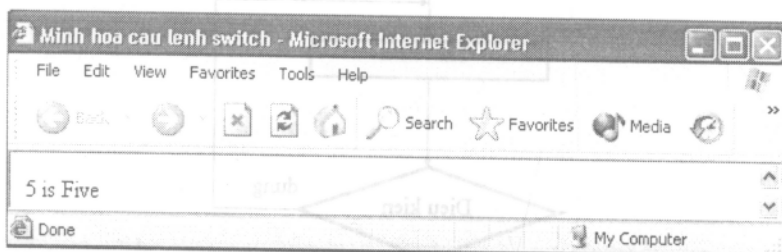
Câu lệnh *switch* sẽ nhận giá trị biến từ 1 đến 10. Ứng với mỗi giá trị của biến sẽ đưa ra thông báo thích hợp trong câu lệnh mà ta đã quy định. Nếu người sử dụng nhập giá trị sai tức là nhập số không phải là số nguyên và nằm ngoài khoảng từ 1 đến 10 thì sẽ đưa ra thông báo lỗi thông qua câu lệnh *default*.

Chẳng hạn, sau khi chúng ta nhập giá trị cho biến *bien_chon* là 5 :



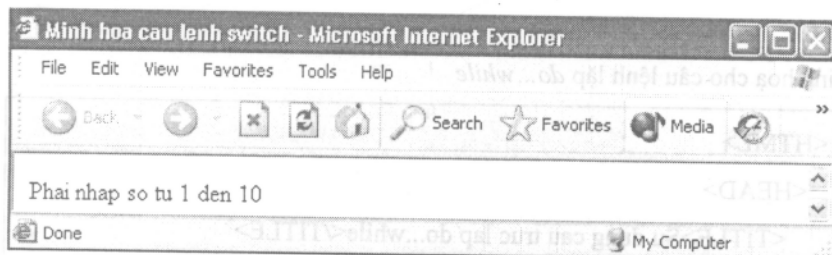
Hình 10.8. Hộp thoại nhập giá trị cho biến.

Thì trình duyệt sẽ cho chúng ta kết quả như sau:



Hình 10.9. Kết quả trên trình duyệt.

Nếu như chúng ta nhập giá trị của *bien_chon* là 11 (nằm ngoài khoảng từ 1 đến 10). Khi đó câu lệnh nằm trong *default* sẽ được thực hiện (hình 10.10).



Hình 10.10. Kết quả sử dụng câu lệnh Switch.

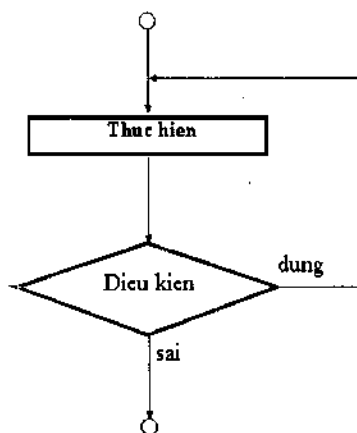
10.6. Câu lệnh lặp do...while

Câu lệnh vòng lặp *do... while* được dùng để chạy một khối lệnh cho đến khi điều kiện trở thành sai. Cú pháp của câu lệnh:

```
do  
{ các câu lệnh; }  
while (điều kiện)
```

Khác với câu lệnh *while*, ở đây câu lệnh *do...while* sẽ thực hiện ít nhất một lần khối câu lệnh nằm trong thân vòng lặp trước khi kiểm tra điều kiện. Nếu điều kiện đúng thì tiếp tục thực hiện câu lệnh lặp tiếp và nếu sai thì thoát ra ngoài. Trong khi câu lệnh *While* thực hiện kiểm tra điều kiện trước khi thực hiện lệnh.

Có thể hiểu giải thuật của câu lệnh *do...while* qua lưu đồ sau:



Hình 10.11. Câu lệnh lặp do...while.

Ví dụ 10.6

Ví dụ minh họa cho câu lệnh lặp *do...while*

```
<HTML>  
<HEAD>  
  <TITLE>Su dung cau truc lap do...while</TITLE>  
  <script language = "JavaScript">
```

Chương 10

Câu lệnh trong JavaScript (tiếp theo)

```
<!--
var bien_dem = 1;
do {
    document.writeln( "<H" + bien_dem + ">This is " +
        "an H" + bien_dem + " level HEAD" + "</H" +
        bien_dem + ">" );
    ++bien_dem;
} while ( bien_dem <= 6 );
// -->
</script>
</HEAD><BODY></BODY>
</HTML>
```

Qua ví dụ 10.6 trên ta thấy rằng câu lệnh *do...while* thực hiện rất giống với câu lệnh *while*. Điều khác biệt duy nhất là nó sẽ thực hiện ít nhất một lần trước khi kiểm tra điều kiện. Ta dùng một biến đếm để kiểm tra điều kiện của vòng lặp.

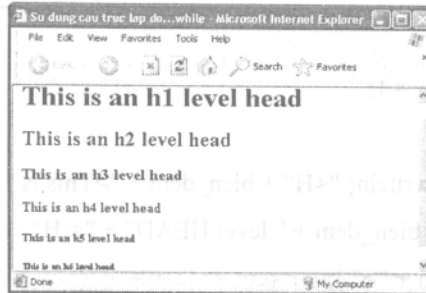
- Đầu tiên chương trình sẽ thực hiện câu lệnh:

```
document.writeln( "<H" + bien_dem + ">This is " +
    "an H" + bien_dem + " level HEAD" + "</H" +
    bien_dem + ">" );
```

- Sau khi thực hiện câu lệnh này, mới kiểm tra điều kiện của *bien_dem*. Nếu như điều kiện nhận giá trị True thì *bien_dem* tăng giá trị lên 1 và tiếp tục thực hiện câu lệnh trên. Sẽ lặp lại như vậy cho đến khi điều kiện nhận giá trị False. Như vậy ngay từ lúc đầu mà *bien_dem* nhận giá trị làm cho điều kiện nhận giá trị False thì ít nhất một câu lệnh trên đã được thực hiện.
- Kết quả chương trình trên trình duyệt như sau :

Chương 10

Câu lệnh trong JavaScript (tiếp theo)



Hình 10.12. Minh hoạ sử dụng vòng lặp do...while.

10.7. Câu lệnh break

Vòng lặp *while* và vòng lặp *if - else* kết thúc tự nhiên khi câu lệnh điều kiện trở nên sai. Bạn cũng có thể kết thúc việc thực hiện một câu lệnh khi cần bằng cách sử dụng câu lệnh *break*. Câu lệnh *break* được dùng để thoát khỏi các vòng lặp và chuyển giao việc điều khiển đến câu lệnh sau vòng lặp. Khi gặp lệnh *break*, ngay lập tức sẽ dừng vòng lặp và chuyển tới câu lệnh ngay sau vòng lặp, có nghĩa là bỏ qua cả những câu lệnh còn lại trong vòng lặp. Câu lệnh này giúp cho chúng ta điều khiển chương trình dễ dàng hơn.

Cú pháp lệnh:

```
break;
```

Ví dụ 10.7

Ví dụ sau sử dụng câu lệnh *break* trong câu lệnh lặp *for*.

```
<HTML>
<HEAD>
<TITLE> Su dung lenh break trong vong lap for </TITLE>
<script language="JavaScript">
<!--
for ( var dem = 1; dem <= 10; ++dem ) {
    if ( dem == 5 )
        break; // Thoat khoi vong lap neu dem == 5
    document.writeln( "Bien dem la : " + dem + "<br />" );
```

Chương 10

Câu lệnh trong JavaScript (tiếp theo)

```
}  
  
document.writeln(  
    "Thoat khỏi vòng lặp khi biến đếm bằng " + dem );  
  
    // -->  
  
</script>  
</HEAD>  
<BODY></BODY>  
</HTML>
```

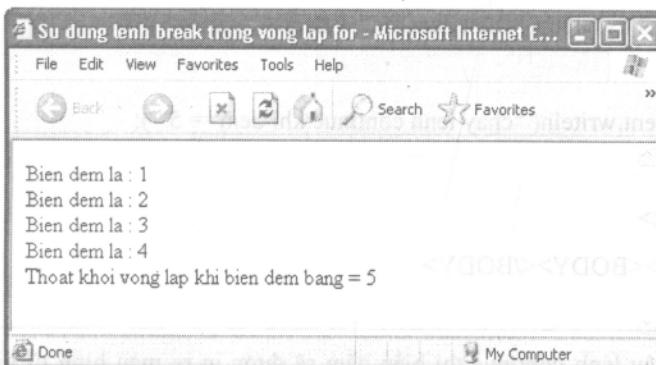
- Nếu như không có lệnh *break* thì vòng lặp sẽ thực hiện đúng 10 lần cho đến khi biến đếm nhận giá trị 11 và thoát khỏi vòng lặp.
- Nhưng trong vòng lặp *for* xuất hiện câu lệnh *break*, do đó khi gặp câu lệnh:

`if ( dem == 5 ) break;`

thì sẽ ngừng và thoát khỏi vòng lặp và thực hiện lệnh kế tiếp là:

`document.writeln("Thoat khỏi vòng lặp khi biến đếm bằng " + dem );`

- Như vậy vòng lặp chỉ xảy ra có 5 lần cho đến khi biến *dem* có giá trị bằng 5. Lúc này kết quả sẽ cho ta khác với khi không có câu lệnh *break*



Hình 10.13. Minh họa sử dụng câu lệnh *break*.

10.8. Câu lệnh *continue*

Một câu lệnh đặc biệt khác có thể được dùng bên trong vòng lặp *while* là câu lệnh *continue*. *Continue* không kết thúc vòng lặp. Khi gặp một câu lệnh *continue*, các câu lệnh

Chương 10

Câu lệnh trong JavaScript (tiếp theo)

còn lại trong vòng lặp được bỏ qua và việc điều khiển được chuyển giao trở lại cho điều kiện của vòng lặp.

Như vậy khác với câu lệnh *break*, thì câu lệnh *continue* lại không thoát khỏi vòng lặp mà quay trở lại từ đầu vòng lặp. Cú pháp lệnh:

continue;

Ví dụ 10.8

Ví dụ sau sử dụng lệnh *continue* với cấu trúc lặp *for*

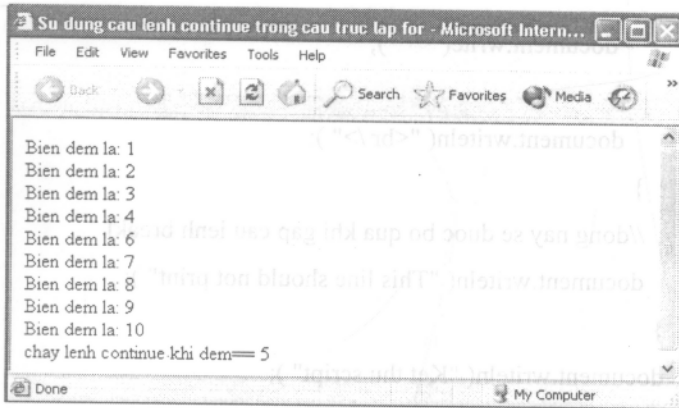
```
<HTML>
<HEAD>
  <TITLE> Su dụng cau lenh continue trong cau truc lap for </TITLE>
  <script language="JavaScript">
    <!--
    for ( var dem = 1; dem <= 10; ++dem ) {
      if ( dem == 5 )
        continue; // bỏ qua phần còn lại của vòng lặp
                  //chỉ khi biến dem == 5
      document.writeln( "Biến dem là: " + dem + "<br />" );
    }
    document.writeln( "chạy lenh continue khi dem== 5" );
    // -->
  </script>
</HEAD><BODY></BODY>
</HTML>
```

Nếu không có câu lệnh *continue* thì biến đếm sẽ được in ra màn hình từ giá trị 1 đến 10. Nhưng vì khi biến *dem* bằng 5 thì gặp câu lệnh: *if (dem == 5) continue;* nên những câu lệnh còn lại trong vòng lặp *for* sẽ bị bỏ qua và tiếp tục quay trở về đầu vòng lặp (quay trở lại kiểm tra điều kiện). Như vậy giá trị 5 của biến *dem* sẽ không được in ra.

Kết quả của ví dụ 10.8 như sau:

Chương 10

Câu lệnh trong JavaScript (tiếp theo)



Hình 10.14. Minh họa sử dụng câu lệnh continue.

Với câu lệnh *break* và *continue* cho phép chúng ta điều khiển cấu trúc của các vòng lặp linh hoạt và hiệu quả hơn.

10.9. Sử dụng nhân với câu lệnh break và continue

10.9.1. Lệnh break sử dụng nhân

Trước khi bắt đầu nói rõ về câu lệnh này chúng ta sẽ nghiên cứu ví dụ sau.

Ví dụ 10.9

Ví dụ này sẽ đặt cho một khối lệnh một nhãn và sử dụng lệnh *break*.

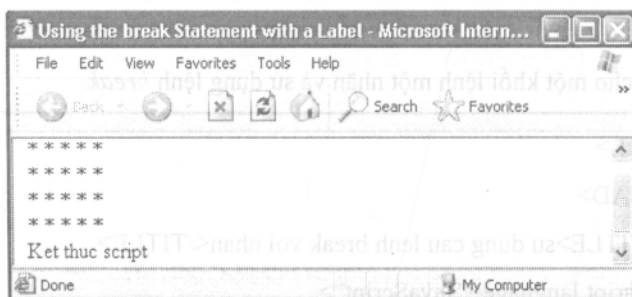
```
<HTML>
<HEAD>
<TITLE>su dung cau lenh break voi nhan</TITLE>
<script language="JavaScript">
<!--
stop: { // Nhan ket hop voi cau lenh
  for ( var hang = 1; hang <= 10; ++hang ) {
    for ( var cot = 1; cot <= 5 ; ++cot ) {
      if ( hang == 5 )
        break stop; // Nhảy tới cuối khối lệnh stop
    }
  }
}
```

Chương 10

Câu lệnh trong JavaScript (tiếp theo)

```
document.write( "*" );  
}  
  
document.writeln( "<br />" );  
}  
  
//dong nay se duoc bo qua khi gap cau lenh break  
document.writeln( "This line should not print" );  
}  
  
document.writeln( "Ket thu script" );  
  
// -->  
</script>  
</HEAD>  
<BODY></BODY>  
</HTML>
```

Kết quả của ví dụ trên như sau:



Hình 10.15. Minh hoạ lệnh break sử dụng nhãn.

Cũng giống như câu lệnh *break* dùng để thoát khỏi cấu trúc của vòng lặp, trong ví dụ này chúng ta sử dụng lệnh *break* với nhãn để thoát khỏi cấu trúc của những vòng lặp lồng nhau. Cụ thể: ngay khi bắt gặp câu lệnh *break* sử dụng nhãn *stop*, chương trình ngay lập tức thoát khỏi vòng khối lệnh có nhãn là *stop* và chuyển tới câu lệnh ngay sau khối lệnh có nhãn *stop*.

Sử dụng câu lệnh *break* với nhãn sẽ cho chúng ta điều khiển được những vòng lặp lồng nhau có cấu trúc phức tạp.

10.9.2. Lệnh *continue* sử dụng nhãn

Giống như câu lệnh *break*, câu lệnh *continue* cũng có thể sử dụng kết hợp với nhãn. Cấu trúc của nó cũng tương tự như câu lệnh *break* sử dụng nhãn.

Khi gặp câu lệnh *continue* sử dụng nhãn đã đặt, thì ngay lập tức vòng lặp sẽ bỏ qua các câu lệnh còn lại trong khối lệnh có nhãn đã đặt và quay trở lại từ đầu vòng lặp. Sử dụng câu lệnh *continue* với nhãn cho chúng ta để điều khiển những vòng lặp có cấu trúc lồng nhau.

Ví dụ 10.10

Ví dụ sau minh họa câu lệnh *continue* sử dụng với nhãn

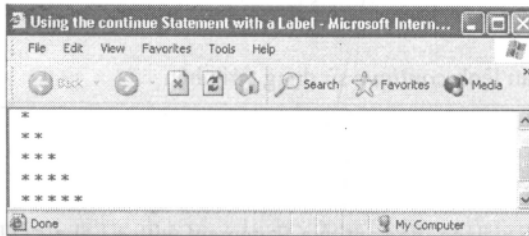
```
<HTML>
<HEAD>
  <TITLE>Using the continue Statement with a Label</TITLE>
  <script type = "text/javascript">
    <!--
    nextRow: // nhãn để sử dụng trong câu lệnh continue
    for ( var hang = 1; hang <= 5; ++hang ) {
      document.writeln( "<br />" );
      for ( var cot = 1; cot <= 10; ++cot ) {
        if ( cot > hang )
          continue nextRow; //lặp lại với nhãn lap
        document.write( "*" );
      }
    }
    // -->
  </script>
</HEAD> <BODY></BODY>
</HTML>
```

Chương 10

Câu lệnh trong JavaScript (tiếp theo)

Nhãn sử dụng trong ví dụ 10.10 này có tên là: *nextRow* đặt ở đầu khối lệnh. Câu lệnh sẽ thực hiện không có gì khác với câu lệnh lặp *for* bình thường cho đến khi gặp câu lệnh *If* (*cot > hang*) *continue nextRow*; Chương trình sẽ kiểm tra, nếu *cot > hang* nhận giá trị true thì câu lệnh *continue* sẽ được thực hiện. Lúc này nó sẽ quay trở về đầu vòng lặp của khối lệnh có nhãn là *nextRow*, đồng thời bỏ qua các câu lệnh ở dưới nó và nằm trong vòng lặp có nhãn là *nextRow*, đó là: `document.write( "*" );`

Kết quả của ví dụ này khi xuất ra trình duyệt sẽ như sau:



Hình 10.16. Minh họa lệnh *continue* sử dụng nhãn.

10.10. Toán tử logical

Sử dụng các cấu trúc lặp nhiều lúc chúng ta phải kết hợp và loại bỏ đồng thời nhiều điều kiện. Trong JavaScript giúp chúng ta điều khiển được những điều kiện này thông qua các toán tử logic.

Toán tử logic thường được dùng để nối nhiều phép so sánh vào trong một biểu thức điều kiện. Một toán tử logic lấy hai toán hạng, mỗi toán hạng mang một giá trị đúng hoặc sai, và trả về kết quả đúng hoặc sai

Có 3 toán tử logic:

- Logical AND (&&).
- Logical OR (||).
- Logical NOT (!).

Có thể miêu tả các toán tử logic qua bảng sau.

Chương 10
Câu lệnh trong JavaScript (tiếp theo)

Bảng 10.1. Toán tử logic trong JavaScript

Toán tử (kí hiệu)	Cách sử dụng	Mô tả
And (&&)	Expr1 && expr2	Trả về giá trị đúng nếu cả hai biểu thức <i>expr1</i> và <i>expr1</i> đều đúng, còn không thì trả về giá trị sai hoặc Trả về biểu thức <i>expr1</i> nếu nó chuyển qua giá trị sai. Nếu không trả về biểu thức <i>expr2</i> .
Or ()	Expr1 expr2	Trả về biểu thức <i>expr1</i> nếu biểu thức <i>expr1</i> hoặc biểu thức <i>expr2</i> đúng hoặc cả hai đều đúng, còn không thì trả về giá trị sai. hoặc Trả về biểu thức <i>expr1</i> nếu nó chuyển qua giá trị đúng. Nếu không trả về biểu thức <i>expr2</i> .
Not (!)	! expr	Trả về giá trị sai nếu biểu thức đúng và giá trị đúng nếu biểu thức sai.

Có thể hiểu rõ hơn giá trị trả về của các toán tử logic thông qua bảng sau.

Toán tử AND

Bảng 10.2. Toán tử logic AND trong JavaScript

Biểu thức 1	Biểu thức 2	Biểu thức 1 && biểu thức 2
False	False	False
False	True	False
True	False	False
True	True	True

Chương 10
Câu lệnh trong JavaScript (tiếp theo)

Toán tử OR

Bảng 10.3. Toán tử logic OR trong JavaScript

Biểu thức 1	Biểu thức 2	Biểu thức 1 Biểu thức 2
False	False	False
False	True	True
True	False	True
True	True	True

Toán tử NOT

Bảng 10.4. Toán tử logic NOT trong JavaScript

Biểu thức	!Biểu thức
False	True
True	False

Ví dụ 10.11

Ví dụ sau sẽ minh họa một chương trình sử dụng các toán tử logic.

```
<HTML>
<HEAD>
  <TITLE>Minh họa cho toán tử logic</TITLE>
  <script language="JavaScript">
    <!--
    document.writeln(
      "<table border = \"1\" width = \"100%\">" );
    document.writeln(
      "<caption>Mô tả toán tử logic " +
      "toán tử</caption> ");
    document.writeln(
      "<tr><td width = \"25%\">Logical AND (&&</td>" +
```

Chương 10

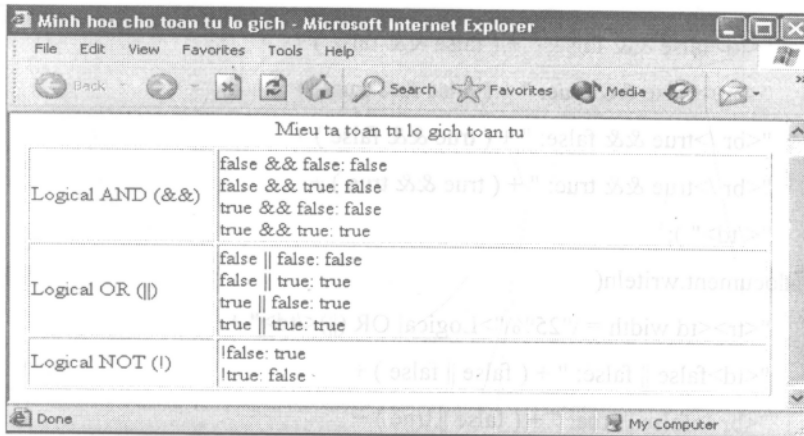
Câu lệnh trong JavaScript (tiếp theo)

```
"<td>false && false: " + ( false && false ) +  
"<br />false && true: " + ( false && true ) +  
"<br />true && false: " + ( true && false ) +  
"<br />true && true: " + ( true && true ) +  
"</td>" );  
document.writeln(  
  "<tr><td width = \"25%\">Logical OR (||)</td>" +  
  "<td>false || false: " + ( false || false ) +  
  "<br />false || true: " + ( false || true ) +  
  "<br />true || false: " + ( true || false ) +  
  "<br />true || true: " + ( true || true ) +  
  "</td>" );  
document.writeln(  
  "<tr><td width = \"25%\">Logical NOT (!)</td>" +  
  "<td>!false: " + ( !false ) +  
  "<br />!true: " + ( !true ) + "</td>" );  
document.writeln( "</table>" );  
// -->  
</script>  
</HEAD><BODY></BODY>  
</HTML>
```

Kết quả của ví dụ 10.11 sẽ như sau :

Chương 10

Câu lệnh trong JavaScript (tiếp theo)



The screenshot shows a web browser window titled "Minh họa cho toán tử lô gích - Microsoft Internet Explorer". The address bar shows a local file path. The main content area displays a table with the title "Mieu ta toán tu lô gích toán tu". The table lists the results of logical AND, OR, and NOT operations on true and false values.

Mieu ta toán tu lô gích toán tu	
Logical AND (&&)	false && false: false false && true: false true && false: false true && true: true
Logical OR ()	false false: false false true: true true false: true true true: true
Logical NOT (!)	!false: true !true: false

Hình 10.17. Minh hoạ sử dụng toán tử logic.

10.11. Kết chương

Trong chương này chúng ta đã tìm hiểu sâu thêm về các câu lệnh trong JavaScript như: bản chất của vòng lặp điều khiển đếm, câu lệnh lặp *for*, *switch*, *do...while*, câu lệnh *break* và *continue*,... Như vậy trong chương 9 và chương 10 chúng ta đã tìm hiểu khá nhiều về lập trình có cấu trúc, những toán tử và câu lệnh cơ bản trong JavaScript. Trong chương sau, chúng ta sẽ đi vào tìm hiểu về hàm trong ngôn ngữ JavaScript.

Chương 11

Các hàm trong JavaScript

Nội dung của chương:

- Tìm hiểu về cấu trúc những modul chương trình trong JavaScript.
- Tạo một hàm mới
- Cách sử dụng các hàm.
- Tạo một số ngẫu nhiên.
- Phạm vi của biến.
- Các hàm có sẵn trong JavaScript.

11.1. Giới thiệu

Khi thiết kế phần mềm bao giờ người phân tích cũng cố gắng chia phần mềm đó thành nhiều phần nhỏ. Các phần nhỏ này độc lập với nhau và có nhiệm vụ thực hiện một chức năng nào đó. Các phần nhỏ này được gọi là các modul (*modul*). Phần mềm là tập hợp của các *modul* này.

Việc chia chương trình thành các *modul* nhỏ giúp chúng ta dễ phát triển và sửa lỗi cho phần mềm, giúp xây dựng khả năng làm việc theo nhóm: mỗi một *modul* sẽ do một nhóm (hay một người) đảm nhiệm.

11.2. Modul chương trình trong JavaScript

Cũng giống như các ngôn ngữ khác, *modul* chương trình trong JavaScript là tập các hàm và các phương thức.

- **Hàm** cho phép người lập trình nhóm những đoạn mã chương trình lại với nhau để thực hiện một công việc nhất định. Hàm là một khối độc lập và được sử dụng nhiều lần trong khi viết chương trình. Hàm được định nghĩa bằng một tên và được gọi để thực hiện cũng qua tên đó.

Chương 11

Các hàm trong JavaScript

- Cũng như những **phương thức** mà ta đã biết trong JavaScript trong chương trước (như là *prompt()* và *confirm()*), hàm có thể cho ta những kết quả khác nhau thông qua các tham số và có thể trả về giá trị cụ thể.
- JavaScript cung cấp cho chúng ta những hàm cơ bản giống như những phương thức của các đối tượng. Chúng ta đã biết khi nào thì sử dụng *alert()*, *document.write()*, *parseInt()*, hoặc những phương thức khác. Tính linh động trong JavaScript là cho phép chúng ta tạo ra những hàm có ý nghĩa như vậy để có thể sử dụng nhiều lần mà không phải viết lại trong một chương trình.

Kết hợp những phương thức có sẵn và những hàm do chúng ta định nghĩa để viết một chương trình sẽ làm cho chương trình chúng ta sáng sủa và gọn gàng hơn. Ta sẽ nghiên cứu về hàm trong JavaScript trong những nội dung dưới đây.

11.2.1. Định nghĩa hàm

Trước khi gọi một hàm bạn phải định nghĩa nó trong chương trình. *Định nghĩa một hàm là quá trình nhận diện tên hàm và chỉ ra thủ tục cần tiến hành khi hàm được gọi.*

Thông thường, các hàm được định nghĩa trong phần HEAD của tài liệu. Điều này để đảm bảo rằng tất cả các hàm đều được định nghĩa trước khi trang được hiển thị. Nếu một hàm được định nghĩa sau tài liệu, và một sự kiện gọi hàm được thực hiện trước đó thì sẽ xuất hiện lỗi.

Khi xây dựng một hàm bạn phải chỉ ra các đối số có trong hàm. Một hàm cũng có thể không có một đối số nào cả.

Ví dụ 11.1. Một ví dụ về định nghĩa hàm tính tổng hai số *sum_up*.

```
<HTML>
<HEAD>
  <script language="JavaScript">
    function sum_up (one, two)
    {
      var result = one + two;
      return result;
    }
  </script>
</HEAD>
</HTML>
```

Chương 11

Các hàm trong JavaScript

11.2.2. Gọi một hàm

Để thực hiện một hàm bạn phải gọi nó. Để gọi một hàm bạn phải chỉ ra tên hàm kèm theo một danh sách các đối số nếu có.

function_name(parameters, arguments);

Những thành phần khi gọi một hàm là :

- Tên của hàm đã được định nghĩa trước (*function_name*). Chú ý rằng tên của hàm phải đặt theo đúng qui định như tên biến trong JavaScript.
- Các thông số (*parameter*) và các đối số (*arguments*) được đặt trong dấu ngoặc đơn và được cách nhau bởi dấu phẩy. Ví dụ :

total += parseFloat(inputValue);

total += parseFloat(s1 + s2);

11.3. Xây dựng một hàm do người dùng định nghĩa

11.3.1. Xây dựng hàm

Bây giờ chúng ta sẽ nghiên cứu cách thức xây dựng một hàm do chính mình tạo nên. Khi xây dựng một hàm trước tiên chúng ta cần nắm rõ những chú ý cơ bản sau:

- Hàm được chỉ ra bởi câu lệnh *function*. Câu lệnh này yêu cầu tên của hàm, danh sách các tham số và đối số.

function function_name(parameters, arguments)

```
{  
    Khoi lệnh của hàm;  
}
```

- Chúng ta cũng chú ý rằng: những câu lệnh trong hàm sẽ không được thực hiện cho đến khi nào có một đoạn script gọi hàm đó. Chỉ khi gọi hàm thì các câu lệnh trong hàm sẽ được thực hiện.
- Các giá trị của biến và hằng có thể truyền vào tham số của hàm khi chúng ta gọi một hàm. Nếu một biến được truyền vào hàm, khi chúng ta thay đổi giá trị của thông số trong hàm sẽ không làm thay đổi giá trị của biến được truyền vào hàm.
- Những tham số và đối số truyền vào hàm chỉ tồn tại trong chu trình hoạt động của hàm.

Chương 11

Các hàm trong JavaScript

Chúng ta sẽ giải thích những điều trên thông qua ví dụ sau: giả sử người dùng tạo ra một hàm *printName()* với tham số truyền vào là *name*. Hàm sau xây dựng với mục đích đưa ra màn hình tên của một người mà người lập trình sử dụng khi gọi hàm.

```
function printName(name) {  
    document.write("<HR>Your Name is <B><I>");  
    document.write(name);  
    document.write("</B></I><HR>");  
}
```

Cụ thể:

- Nếu chúng ta gọi hàm với câu lệnh *printName("Bob")*; khi chương trình thực hiện, giá trị của biến *name* là Bob.
- Khi chúng ta gọi hàm *printName()* thông qua tham số là một biến truyền vào:

```
Var user = "John";  
printName(user);
```

Lúc này biến *name* có giá trị là John.

- Nếu chúng ta thêm vào một dòng lệnh trong *printName()* để thay đổi giá trị của biến *name*:

```
name = "Mr. " + name;
```

biến *name* sẽ thay đổi giá trị, nhưng giá trị của biến *user* truyền vào đối số khi gọi hàm sẽ có giá trị không thay đổi.

Phạm vi của biến trong hàm.

- Phạm vi của biến liên quan đến nơi mà biến tồn tại.
- Bất kì biến nào được khai báo sử dụng câu lệnh *var* trong hàm sẽ có phạm vi giới hạn trong hàm đó.
- Nếu biến được khai báo ở ngoài thân của hàm, thì đó là biến có thể sử dụng trong tất cả các hàm và bất cứ ở nơi đâu trong chương trình.

Trong ví dụ *printName()*, biến *name* chỉ tồn tại trong hàm *printName()*. Nó không có tác dụng khi ở ngoài hàm đó. Nó sẽ tồn tại khi hàm được gọi và sẽ không tồn tại khi hàm kết thúc. Nếu như hàm *printName()* được gọi một lần nữa, một giá trị mới của biến *name* sẽ được khởi tạo.

Chương 11

Các hàm trong JavaScript

11.3.2. Câu lệnh trả về *Return*

Câu lệnh *return* này được dùng với ý nghĩa là một câu lệnh hàm có nhiệm vụ trả về một giá trị. Việc dùng câu lệnh trả về trong một hàm là không bắt buộc bởi vì không phải tất cả các hàm của JavaScript đều trả về một giá trị cụ thể.

Cú pháp của lệnh như sau:

```
return giá trị;  
hay  
return (giá trị);
```

Câu lệnh *return* có thể không trả về một giá trị nào hoặc trả về một giá trị.

11.3.3. Các ví dụ minh họa

Sau đây chúng ta sẽ đi vào cụ thể các ví dụ để thấy rõ hơn cách thức xây dựng và sử dụng các hàm trong một chương trình JavaScript.

Ví dụ 11.2

Viết một chương trình để tính bình phương của các số nguyên từ 1 đến 10.

```
<HTML>  
<HEAD>  
  <TITLE>Ham binh phuong</TITLE>  
  <script language="JavaScript">  
    <!--  
    document.writeln(  
      "<h1>Bình phương của các số từ 1 đến 10</h1>");  
    // bình phương của các số từ 1 đến 10  
    for ( var x = 1; x <= 10; ++x )  
      document.writeln( "Bình phương của " + x + " là " +  
        square( x ) + "<BR />" );  
    // câu lệnh trong phần thân của hàm sẽ được thực hiện  
    // Chỉ khi hàm được gọi ra  
    // định nghĩa hàm square
```

Chương 11

Các hàm trong JavaScript

```
function square( y )
{
    return y * y;
}

// -->

</script>

</HEAD><BODY></BODY>

</HTML>
```

Trong ví dụ trên chúng ta xây dựng một hàm có tên là *square*.

- Một vòng lặp *for* thực hiện từ khi biến $x = 1$ đến biến $x = 10$. Mỗi một lần lặp sẽ lấy giá trị của biến x như là một thông số của hàm *square*. Và trả về giá trị của hàm *square* thông qua thông số đó.

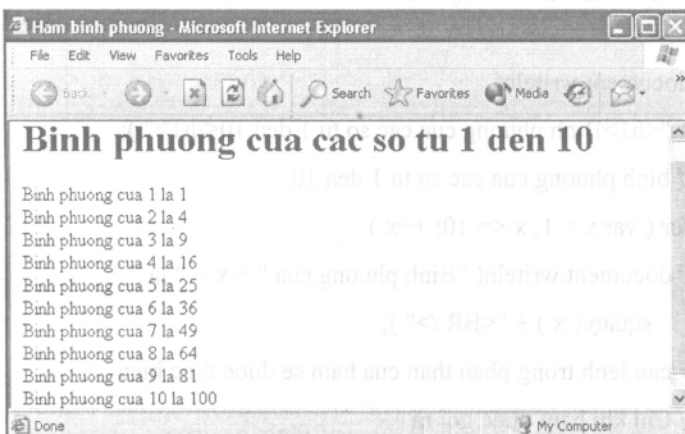
- Câu lệnh:

`document.writeln( "Bình phương của " + x + " là " + square(x) + "<BR />" );`

sẽ gọi hàm *square* và truyền giá trị của x . Hàm *square* được xây dựng với tham số là y .

- Câu lệnh `return y * y;` sẽ trả về giá trị của phép tính $y * y$ sau mỗi lần gọi hàm.

Kết quả của ví dụ trên như sau:



Hình 11.1. Sử dụng hàm để tính bình phương của một số nguyên.

Chương 11

Các hàm trong JavaScript

Ví dụ 11.3

Viết chương trình tìm số lớn nhất trong 3 số.

```
<HTML>
  <HEAD>
    <TITLE>Tìm số lớn nhất trong 3 số</TITLE>
    <script language = "JavaScript">
      <!--
        var input1 = window.prompt( "Nhập số thứ nhất", "0" );
        var input2 = window.prompt( "Nhập số thứ hai", "0" );
        var input3 = window.prompt( "Nhập số thứ ba", "0" );
        var so1 = parseFloat( input1 );
        var so2 = parseFloat( input2 );
        var so3 = parseFloat( input3 );
        var maxValue = maximum( so1, so2, so3 );
        document.writeln( "Số thứ nhất: " + so1 +
          "<BR />Số thứ hai: " + so2 +
          "<BR />Số thứ ba: " + so3 +
          "<BR />Số lớn nhất là: " + maxValue );
        function maximum( x, y, z )
        {
          return Math.max( x, Math.max( y, z ) );
        }
      // -->
    </script>
  </HEAD>
  <BODY>
    <P>Ấn nút Refresh để chạy lại script này</P>
  </BODY>
</HTML>
```

Chương 11

Các hàm trong JavaScript

Để làm được bài toán này chúng ta phải xác định rõ những việc phải làm, bao gồm:

- Đầu tiên phải nhập vào 3 số *input1*, *input2*, *input3* (sử dụng phương thức *window.prompt()*).
- Sau đó chuyển đổi 3 *string* đó sang 3 số *float*, sử dụng câu lệnh:

```
var so1 = parseFloat( input1 );  
var so2 = parseFloat( input2 );  
var so3 = parseFloat( input3 );
```

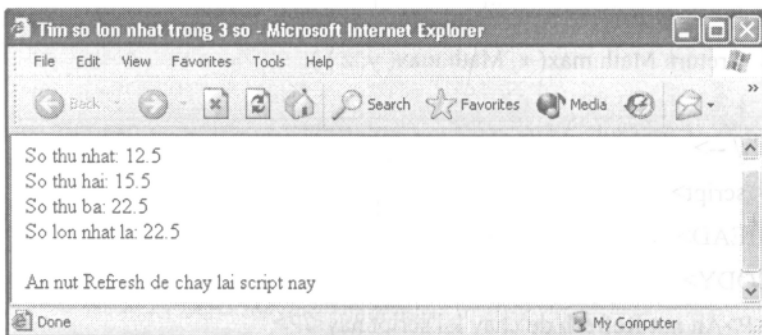
- Khi chạy chương trình này thì *so1*, *so2* và *so3* là 3 số thực float. Sau đó chúng ta gọi hàm *maximum()* với 3 thông số là: *so1*, *so2*, *so3*.
- Khi bắt gặp câu lệnh gọi hàm *maximum* với ba tham số như trên các câu lệnh trong hàm *maximum()* sẽ được thực hiện với các thông số truyền vào đó.

```
function maximum( x, y, z )  
{  
    return Math.max( x, Math.max( y, z ) );  
}
```

x, *y*, *z* lúc đó sẽ nhận 3 giá trị tương ứng là *so1*, *so2*, *so3*.

Trong câu lệnh xây dựng hàm *maximum()* chúng ta sử dụng phương thức *Math.max(x, y)*. Phương thức này sẽ trả về số có giá trị lớn hơn trong hai số *x* và *y*.

Kết quả khi chạy chương trình với ba số thực nhập vào là 12.5, 15.5, 22.5 sẽ như sau:



Hình 11.2. Sử dụng hàm tìm số lớn nhất trong 3 số.

11.4. Tạo một số ngẫu nhiên

Cũng như những ngôn ngữ lập trình, trong JavaScript cũng cung cấp cho chúng ta một phương thức để sinh ra một số thực ngẫu nhiên có giá trị nằm trong khoảng $[0,1]$. Đó chính là hàm *Math.random*. Cách gọi và sử dụng hàm này như sau:

var randomValue = Math.random();

Sử dụng các phương thức của hàm *Math.random()*:

- *Math.floor()* để trả về một giá trị nguyên tiếp theo nhỏ hơn giá trị tham số đưa vào.
Ví dụ: *Math.floor(1 + Math.random() * 6)* sẽ trả về một dãy số nguyên từ 1 đến 6
- *Math.round(number)*: trả lại giá trị của số được làm tròn tới số nguyên gần nhất.

Chúng ta sẽ minh họa một chương trình sinh ra các số ngẫu nhiên.

Ví dụ 11.4

Tạo ra các số nguyên từ 1 đến 6 và ghi vào một bảng gồm 5 cột và 4 dòng.

```
<HTML>
<HEAD>
  <TITLE>Tao ra cac so nguyen</TITLE>
  <script language = "JavaScript">
    <!--
    var value;
    document.writeln(
      "<table border = \"1\" width = \"50%\">" );
    document.writeln(
      "<caption>Cac so ngau nhien</caption><tr>" );
      for ( var i = 1; i <= 20; i++ ) {
        value = Math.floor( 1 + Math.random() * 6 );
        document.writeln( "<td>" + value + "</td>" );
        //dinh dang cho bang
        if ( i % 5 == 0 && i != 20 )
          document.writeln( "</tr><tr>" );
```

Chương 11

Các hàm trong JavaScript

```
}
document.writeln( "</tr></table>" );
// -->
</script>
</HEAD>
<BODY>
<P>An lại nút Refresh để chạy lại script này</P>
</BODY>
</HTML>
```

Để làm được điều này, chương trình đã sử dụng các câu lệnh *Math.random()* và *Math.floor(number)*.

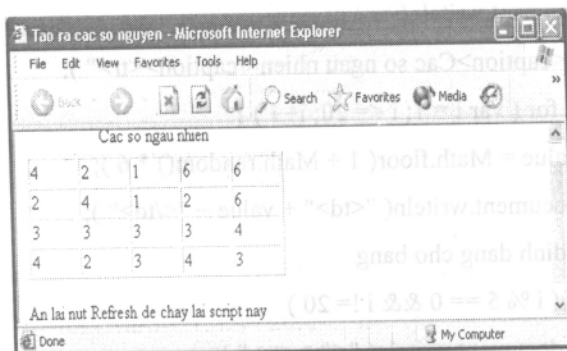
```
value = Math.floor( 1 + Math.random() * 6 );
```

Câu lệnh này sinh ra một số nguyên bất kì nằm trong khoảng từ 1 đến 6.

```
for ( var i = 1; i <= 20; i++ )
```

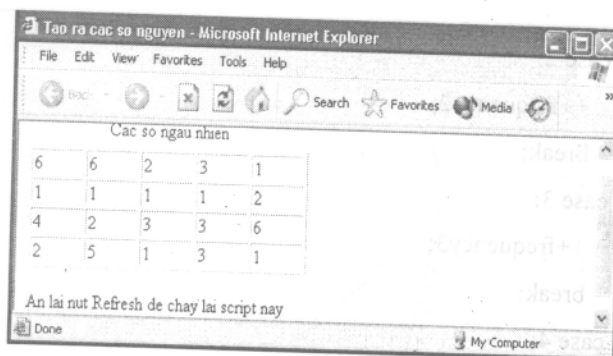
```
{
    value = Math.floor( 1 + Math.random() * 6 );
    document.writeln( "<td>" + value + "</td>" );
}
```

Vòng lặp tạo ra một bảng gồm 20 ô (4 hàng và 5 cột). Giá trị được sinh ra sẽ được đặt vào mỗi ô đó. Phương thức *floor* trả về cho chúng ta số nguyên gần nhất của tham số đưa vào và nhỏ hơn tham số đưa vào. Chạy chương trình trên 2 lần sẽ cho chúng ta kết quả như sau:



Chương 11

Các hàm trong JavaScript



Hình 11.3. Chương trình sinh ra một số ngẫu nhiên.

Ví dụ 11.5

Cũng tương tự như ví dụ 11.4, trong ví dụ 11.5 sau cũng sẽ tạo ra các số nguyên từ 1 đến 6 và ghi vào các hàng để tạo thành một chuỗi số. Bạn hãy đọc và tham khảo ví dụ này.

```
<HTML>
<HEAD>
  <TITLE>Minh họa tạo số ngẫu nhiên</TITLE>
  <script language="JavaScript">
    <!--
    var frequency1 = 0, frequency2 = 0,
        frequency3 = 0, frequency4 = 0,
        frequency5 = 0, frequency6 = 0, face;
    // 6000 lần
    for ( var roll = 1; roll <= 6000; ++roll ) {
      face = Math.floor( 1 + Math.random() * 6 );
      switch ( face ) {
        case 1:
          ++frequency1;
          break;
```

Chương 11

Các hàm trong JavaScript

```
case 2:
    ++frequency2;
    break;
case 3:
    ++frequency3;
    break;
case 4:
    ++frequency4;
    break;
case 5:
    ++frequency5;
    break;
case 6:
    ++frequency6;
    break;
}
}

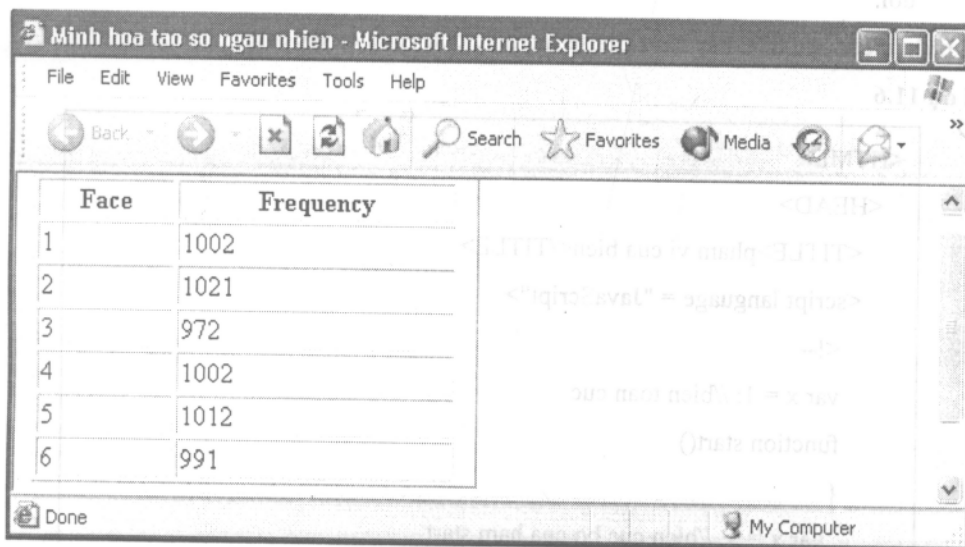
document.writeln( "<table border = \"1\" " +
    "width = \"50%\">" );
document.writeln( "<thead><th>Face</th>" +
    "<th>Frequency</th></thead>" );
document.writeln( "<tbody><tr><td>1</td><td>" +
    frequency1 + "</td></tr>" );
document.writeln( "<tr><td>2</td><td>" + frequency2 +
    "</td></tr>" );
document.writeln( "<tr><td>3</td><td>" + frequency3 +
    "</td></tr>" );
```

Chương 11

Các hàm trong JavaScript

```
document.writeln( "<tr><td>4</td><td>" + frequency4 +  
    "</td></tr>" );  
document.writeln( "<tr><td>5</td><td>" + frequency5 +  
    "</td></tr>" );  
document.writeln( "<tr><td>6</td><td>" + frequency6 +  
    "</td></tr></tbody></table>" );  
// -->  
</script>  
</HEAD>  
<BODY>  
    <P>An nút Refresh để chạy lại script này</P>  
</BODY>  
</HTML>
```

Sau khi chạy script này sẽ cho ta kết quả như sau:



Hình 11.4. Sinh ra số nguyên ngẫu nhiên.

Chương 11

Các hàm trong JavaScript

11.5. Phạm vi của biến

Trong mục này chúng ta sẽ nói rõ hơn về phạm vi của biến trong các hàm ở JavaScript.

- Bất kì biến nào được khai báo sử dụng câu lệnh *var* trong hàm sẽ chỉ có phạm vi giới hạn trong hàm đó.
- Nếu biến được khai báo ở ngoài thân của hàm, thì đó là biến có thể sử dụng trong tất cả các hàm và bất cứ ở nơi đâu trong chương trình.
- Biến mà chỉ tồn tại ở trong một hàm nhất định và khi sử dụng trong các hàm khác phải khai báo lại người ta gọi là biến cục bộ. Hay nói rõ hơn biến cục bộ chỉ tồn tại trong hàm đó mà thôi.
- Còn biến khai báo ở ngoài các hàm và được sử dụng mà không phải khai báo lại người ta gọi là biến toàn cục. Biến này tồn tại trong suốt chương trình.
- Giá trị của biến toàn cục sẽ thay đổi sau mỗi lần gọi nó. Còn giá trị của biến cục bộ là giá trị mà sau khi thực hiện lời gọi hàm có chứa nó.
- Nếu chúng ta khai báo một biến cục bộ ở trong một hàm có cùng tên với một biến toàn cục đã tồn tại trong chương trình thì khi chúng ta gọi hàm đó, nó sẽ tham chiếu đến biến cục bộ chứ không tham chiếu đến biến toàn cục. Và nếu thay đổi giá trị của biến cục bộ ở trong hàm thì giá trị của biến toàn cục có cùng tên sẽ không bị thay đổi.

Để phân biệt được điều này chúng ta sẽ tìm hiểu qua ví dụ cụ thể sau.

Ví dụ 11.6

```
<HTML>
<HEAD>
  <TITLE>phạm vi của biến</TITLE>
  <script language = "JavaScript">
    <!--
      var x = 1; //biến toàn cục

      function start()
      {
        var x = 5; //biến cục bộ của hàm start
        document.writeln(" biến cục bộ của hàm start là x =  "+x);
```

Chương 11

Các hàm trong JavaScript

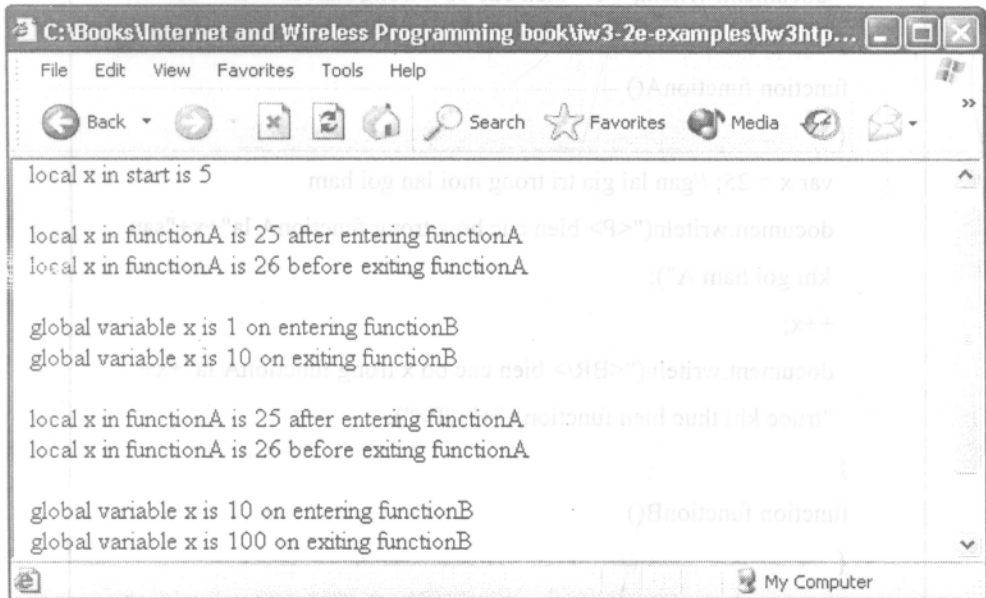
```
functionA(); //functionA() co bien cuc bo x
functionB();// functionB() su dung bien toan cuc x
functionA();// bien cuc bo bi thay doi gia tri
functionB();//bien toan cuc van khong thay doi gia tri
document.writeln("<P> bien cuc bo x trong start la "+x+"</P>");
}
function functionA()
{
    var x = 25; //gan lai gia tri trong moi lan goi ham
    documen.writeln("<P> bien cuc bo x trong functionA la"+x+"sau
    khi goi ham A");
    ++x;
    document.writeln("<BR/> bien cuc bo x trong functionA la"+x+
    "truoc khi thuc hien functionA"+"</P>");
}
function functionB()
{
    document.writeln("<P>Bien toan cuc x la "+x+" khi goi vao ham
functionB");
    x*=10;
    document.writeln("<BR/> Bien toan cuc x la "+x+
    "khi thuc hien functionB"+"</P>");
}
// -->
</script>
</HEAD>
<BODY>
```

Chương 11

Các hàm trong JavaScript

```
<P>An nút Refresh de chạy lại script này</P>  
</BODY>  
</HTML>
```

Bạn chú ý vào kết quả đầu ra của chương trình (hình 11.5) chúng ta sẽ hiểu được ý nghĩa của việc sử dụng biến trong hàm và phạm vi của chúng trong chương trình.



Hình 11.5. Minh họa phạm vi của biến.

Chú ý kết quả của chương trình ta thấy: biến *x* cục bộ trong hàm *functionA()* sẽ thay đổi giá trị sau mỗi lần gọi hàm và chỉ có thể sử dụng trong hàm *functionA()* mà thôi. Còn biến toàn cục *x* có thể có trong toàn chương trình.

Tên biến có thể trùng nhau nhưng không ảnh hưởng đến nhau. Đây cũng là một ý nghĩa quan trọng trong việc sử dụng và xây dựng hàm.

11.6. Các hàm có sẵn trong JavaScript

JavaScript có nhiều hàm định nghĩa sẵn mà bạn có thể dùng trong script của mình. Các hàm này luôn luôn sẵn sàng có thể sử dụng khi chúng ta viết chương trình mà không phải xây dựng nó. Chúng ta chỉ cần quan tâm đến giá trị mà các hàm này trả về. Hơn nữa chúng ta cũng không phải khai báo trước khi sử dụng nó như hàm do người dùng xây dựng.

Chương 11

Các hàm trong JavaScript

Nó có thể gọi bất cứ nơi đâu trong chương trình, hay thậm chí trong thân của những hàm khác.

Trong JavaScript cung cấp cho chúng ta 7 hàm sẵn, bao gồm những hàm sau:

- `escape`
- `eval`
- `isFinite`
- `isNaN`
- `parseFloat`
- `parseInt`
- `unescape`

Chúng ta sẽ nói rõ hơn về ý nghĩa và kết quả trả về của 4 hàm mà chúng ta hay dùng nhất khi viết chương trình trong JavaScript. Đó là các hàm *eval*, *isNaN*, *parseInt*, *parseFloat*.

11.6.1. Hàm *eval*

Hàm *eval* được dùng để đánh giá một chuỗi mã lệnh mà không cần tham chiếu đến bất kì đối tượng cụ thể nào. Chuỗi đó có thể là một biểu thức JavaScript, một câu lệnh, hay một nhóm các câu lệnh. Biểu thức có thể gồm cả các biến và các thuộc tính của một đối tượng.

`eval(string)`

Nếu chuỗi được chỉ ra là một biểu thức thì hàm *eval* sẽ đánh giá biểu thức đó. Nếu chuỗi đó có một hoặc nhiều hơn một câu lệnh thì hàm *eval* sẽ thực hiện các câu lệnh đó. Bạn có thể dùng hàm *eval* để tạo kết quả ra màn hình.

Ví dụ 11.7

```
<HTML>
  <SCRIPT LANGUAGE= "JavaScript">
    var x = 5;
    var z= 10;
    document.write (eval ("x + z + 5"));
  </SCRIPT>
</HTML>
```

Kết quả của ví dụ trên là 20.

Chương 11

Các hàm trong JavaScript

11.6.2. Hàm *isNaN*

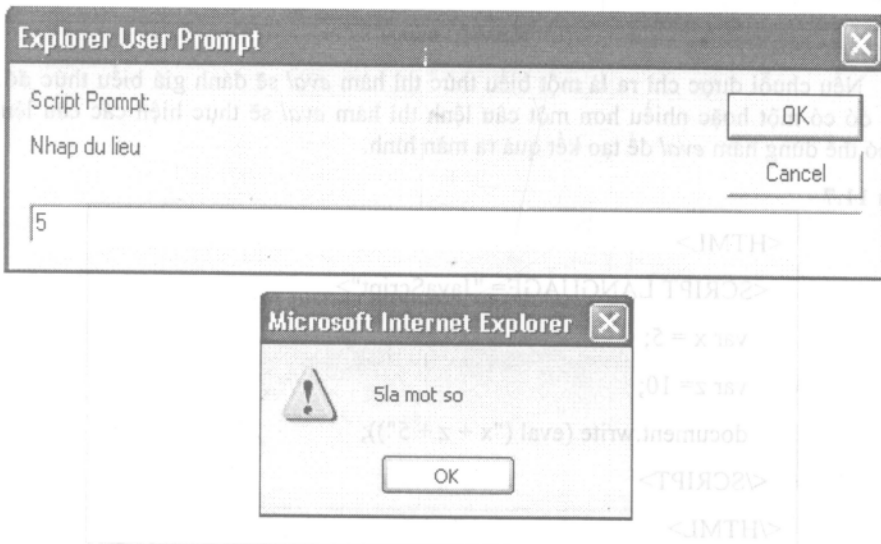
Hàm *isNaN* được dùng để xác nhận xem một đối số có phải là một số hay không. Có pháp hàm như sau:

isNaN(Giá trị kiểm tra)

Ví dụ 11.8

```
<HTML>
<SCRIPT LANGUAGE="JavaScript">
  var x = prompt ("Nhap du lieu");
  if (isNaN (x)) {
    alert (x + "khong phai la mot so");
  }
  else
    alert (x + "la mot so");
</SCRIPT>
</HTML>
```

Kết quả của chương trình trên như sau:



Hình 11.6. Minh họa sử dụng hàm *isNaN*.

Chương 11

Các hàm trong JavaScript

11.6.3. Hàm *parseInt*

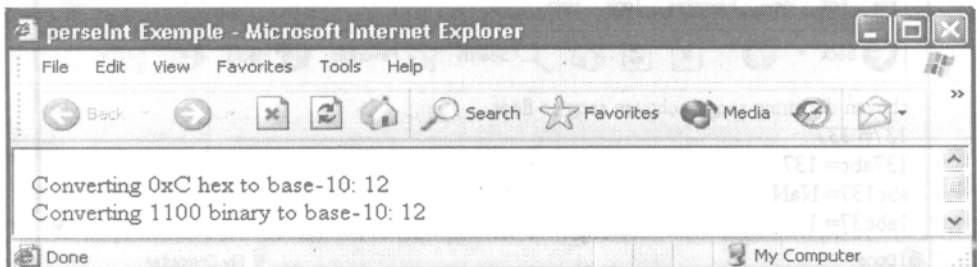
Hàm *parseInt* chuyển một chuỗi số thành số nguyên với cơ số là tham số thứ hai (tham số này không bắt buộc). Hàm này thường được sử dụng để chuyển các số nguyên sang cơ số 10 và đảm bảo rằng các dữ liệu được nhập dưới dạng ký tự được chuyển thành số trước khi tính toán. Trong trường hợp dữ liệu vào không hợp lệ, hàm *parseInt* sẽ đọc và chuyển dạng chuỗi đến vị trí nó tìm thấy ký tự không phải là số. Ngoài ra hàm này còn cắt dấu phẩy động.

Cú pháp: *parseInt* (*string*, [*radix*])

Ví dụ 11.9

```
<HTML>
<HEAD>
<TITLE> parseInt Exemple </TITLE>
<SCRIPT LANGUAGE= "JavaScript">
document.write("Converting 0xC hex to base-10: " + parseInt(0xC,10) +
"<BR>");
document.write("Converting 1100 binary to base-10: " + parseInt(1100,2) +
"<BR>");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Kết quả của chương trình này là:



Hình 11.7. Minh họa sử dụng hàm *parseInt*.

Chương 11

Các hàm trong JavaScript

11.6.4. Hàm `parseFloat`

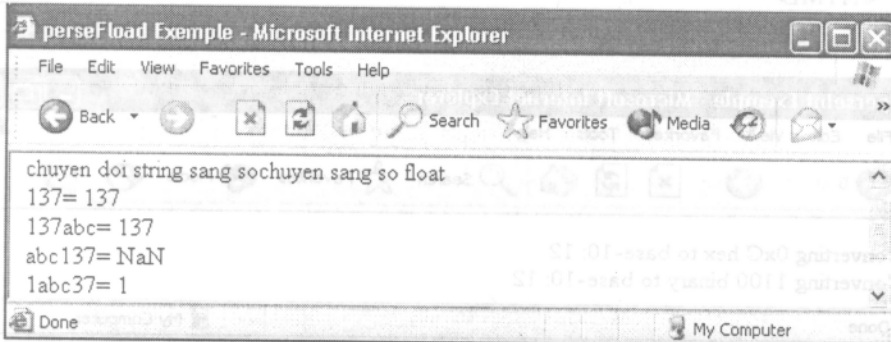
Hàm này giống hàm `parseInt` nhưng nó chuyển chuỗi thành số biểu diễn dưới dạng dấu phẩy động. Cú pháp như sau:

```
parseFloat ( string);
```

Ví dụ 11.10

```
<HTML>
<HEAD>
<TITLE> parseFloat Exemple </TITLE>
<SCRIPT LANGUAGE= "JavaScript">
    document.write("chuyen doi string sang so");
    document.write("chuyen sang so float<BR>");
    document.write("137= " + parseFloat("137") + "<BR>");
    document.write("137abc= " + parseFloat("137abc") + "<BR>");
    document.write("abc137= " + parseFloat("abc137") + "<BR>");
    document.write("1abc37= " + parseFloat("1abc37") + "<BR>");
</SCRIPT>
</HEAD>
<BODY> </BODY>
</HTML>
```

Kết quả khi chạy chương trình này là:



Hình 11.8. Minh họa hàm `parseFloat`.

Chương 11

Các hàm trong JavaScript

11.7. Kết chương

Trong chương này chúng ta đã nghiên cứu về hàm trong JavaScript. Tác dụng của hàm như chúng ta đã thấy là rất quan trọng. Nó giúp cho chúng ta tổ chức và thiết kế chương trình chặt chẽ hơn, dễ phát triển phần mềm và sửa lỗi. Việc phân chia chương trình thành các hàm và các *modul* giúp chúng ta có khả năng làm việc theo nhóm dễ dàng.

Trong chương sau chúng ta sẽ nghiên cứu về khái niệm mảng trong JavaScript. Đây cũng là một khái niệm rất chủ yếu trong JavaScript.

Chương 12

Sử dụng mảng trong JavaScript

Nội dung của chương:

- Tìm hiểu về cấu trúc của một mảng.
- Cách thức lưu trữ dữ liệu trong một mảng.
- Khai báo, khởi tạo một mảng và truy cập vào một phần tử trong mảng.
- Sử dụng mảng trong các hàm.
- Khai báo và khởi tạo mảng nhiều chiều.

12.1. Giới thiệu

Trong những phần trước, chúng ta biết rằng các biến được sử dụng để lưu giữ một thông tin riêng lẻ trong bộ nhớ. Khi có nhiều thông tin cần lưu trữ, sẽ dẫn đến những rắc rối về việc đặt tên biến. Hơn nữa, mỗi biến đều phải được khai báo ở đầu chương trình.

Khái niệm *Mảng (array)* giúp chúng ta giải quyết vấn đề này. Mảng là nhóm các phần tử có cùng kiểu dữ liệu, cùng tên và có một chỉ số dùng để phân biệt các giá trị. Trong JavaScript mảng luôn bắt đầu với chỉ số 0.

JavaScript không có một kiểu dữ liệu mảng rõ ràng nhưng nó có một đối tượng mảng xây dựng sẵn. Để thêm một mảng vào chương trình của bạn, bạn phải sử dụng đối tượng mảng và phương thức liên quan đến nó.

Mảng là phần quan trọng của đa số các ngôn ngữ lập trình. Chúng ta sẽ phân tích lợi ích của mảng qua ví dụ sau: giả sử cần viết một chương trình cộng 5 số.

Tiến hành như sau:

Chương 12

Sử dụng mảng trong JavaScript

```
Bat dau
Khai bao , so2, so3, so4, so5 và tong là một số nguyên
Nhập so1
Nhập so2
Nhập so3
Nhập so4
Nhập so5
tong = so1+so2+so3+so4+so5
Hiển thị tong
Kết thúc
```

Nhận xét: rằng trong chương trình này ta phải khai báo nhiều biến. Càng phải cộng nhiều số hơn thì chúng ta càng phải khai báo nhiều biến hơn. Để khắc phục điều này chúng ta có thể sử dụng một biến đơn và một vòng lặp như sau:

```
Bat dau
Khai báo so, tong và bien_dem là các số nguyên
tong = 0
bien_dem = 0
while bien_dem < 5
do
    Nhập giá trị so
    Cộng vào tong
    Tăng bien_dem lên 1
Kết thúc vòng lặp
Hiển thị tong
Kết thúc
```

Nhận xét: Nhược điểm của việc sử dụng chương trình trên là mỗi giá trị được người sử dụng nhập vào sẽ bị mất ngay khi nhập giá trị kế tiếp. Ví dụ, lần thứ nhất chúng ta nhập vào số 5. Số này được lưu trữ trong biến *so* và sau đó giá trị này được cộng vào *tong*. Biến đếm được tăng và giá trị tiếp theo được nhập vào. Giả sử giá trị này là 8. Giá trị này được chứa trong biến *so*. Có nghĩa là *so* có giá trị là 8. Vì thế, giá trị 5 được lưu trữ lần trước đã bị mất. Do

Chương 12

Sử dụng mảng trong JavaScript

đó nếu cần tất cả 5 số đó để thực hiện một vài toán tử khác thì cách trên không thực hiện được.

Khi đó sử dụng mảng sẽ giải quyết được vấn đề này.

12.2. Mảng trong JavaScript

Chúng ta sẽ nghiên cứu cấu trúc mảng trong JavaScript. Hình dưới đây sẽ minh họa dùng một mảng để lưu trữ 12 phần tử:

Tên của mảng (Tất cả các phần tử trong mảng có cùng tên, c)	c[0]	-45
	c[1]	6
	c[2]	0
	c[3]	72
	c[4]	1543
	c[5]	-89
	c[6]	0
	c[7]	62
	c[8]	-3
	c[9]	1
	c[10]	6453
	c[11]	78

Vị trí ở trong mảng
của mỗi phần tử

Hình 12.1. Minh họa mảng có 12 phần tử.

Qua hình vẽ minh họa trên chúng ta có thể thấy rằng khái niệm mảng trong JavaScript không khác mấy so với khái niệm mảng trong những ngôn ngữ lập trình khác.

12.2.1. Tạo mảng

Để tạo mảng trong JavaScript ta theo cấu trúc sau:

```
Ten_mang = new Array([phan_tu0, phan_tu1, ..., phan_tuN]);
```

Trong đó:

- **Ten_mang**: Là tên của đối tượng (chính là tên mảng).
- **new**: Là một toán tử trong JavaScript cung cấp bộ nhớ cho mảng mới khai báo.

Chương 12

Sử dụng mảng trong JavaScript

- *phan_tuN*: Là danh sách các giá trị cho các phần tử của mảng. Mỗi phần tử có một số chỉ mục để tham chiếu đến nó trong mảng. Trong JavaScript chỉ số bắt đầu từ 0.

Mảng được khởi tạo với các giá trị được chỉ ra như là các phần tử của nó. Thuộc tính độ dài chứa số lượng các phần tử được chỉ ra.

Thêm các phần tử vào mảng:

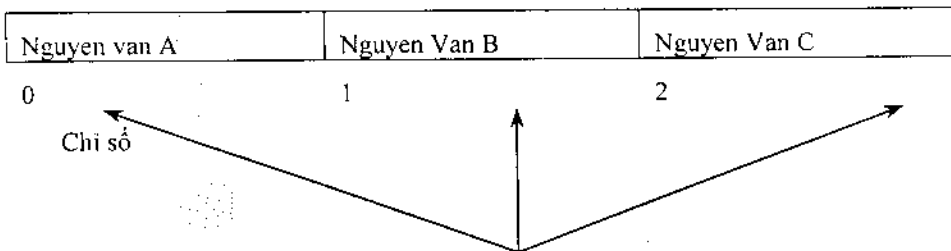
Bạn có thể thêm các phần tử vào một mảng khi bạn tạo nó. Bạn cũng có thể thêm dữ liệu vào một mảng bằng cách gán các giá trị cho các phần tử đó của nó.

Ví dụ:

```
nhan_vien [0] = "Nguyen van A"
```

```
Nhan_vien [1] = "Nguyen Van B"
```

```
Nhan_vien [2] = "Nguyen Van C"
```



Một mảng gồm các phần tử

Ví dụ 12.1: Một ví dụ miêu tả một mảng:

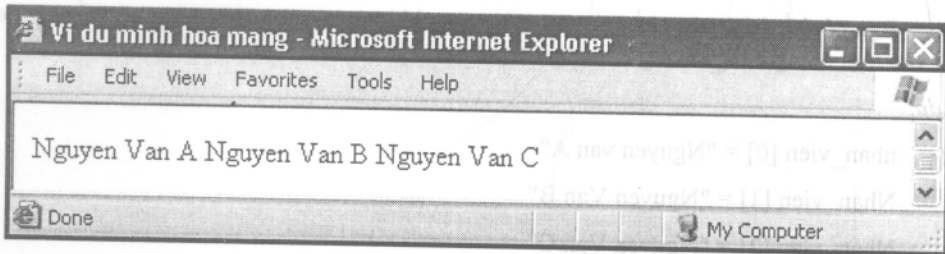
```
<HTML>
<HEAD><TITLE> Ví dụ minh họa mảng </TITLE>
<script language = "JavaScript">
    nhan_vien = new Array (3)
    nhan_vien [0] = "Nguyen Van A"
    nhan_vien [1] = "Nguyen Van B"
    nhan_vien [2] = "Nguyen Van C";
    document.writeln (nhan_vien[0]);
    document.writeln (nhan_vien[1]);
    document.writeln (nhan_vien[2]);
```

Chương 12

Sử dụng mảng trong JavaScript

```
</script>
</HEAD>
</HTML>
```

Kết quả của ví dụ 12.1 trên như sau:



Hình 12.2. Miêu tả một mảng.

Truy cập các phần tử của mảng

- Để truy cập các phần tử của mảng bạn phải chỉ ra trực tiếp tên mảng hay chỉ ra số chỉ số của phần tử đó. Ví dụ: `nhau_vien[0]` hay `nhau_vien["Nguyen Van A"]`
- Khi truy nhập vào một phần tử phải chú ý số chỉ mục của phần tử trong mảng. JavaScript bắt đầu từ 0.

12.3. Một số ví dụ sử dụng mảng

Để hiểu hơn về mảng và cách sử dụng mảng trong chương trình JavaScript chúng ta sẽ tìm hiểu bằng các ví dụ sau.

Ví dụ 12.2

Quan sát ví dụ sau

```
<HTML>
<HEAD>
<TITLE>Khoi tao mot mang</TITLE>
  <script language="JavaScript">
    <!--
      function Khoi_tao_mang()
      {
```

Chương 12

Sử dụng mảng trong JavaScript

```
var n1 = new Array( 5 );// khai bao mot mang 5 phan tu
var n2 = new Array();    // Khai bao mot mang rong
// Gan cac gia tri cho cac phan tu trong mang
for ( var i = 0; i < n1.length; ++i )
    n1[ i ] = i;
// tao va khoi tao 5 phan tu cho mang n2
for ( i = 0; i < 5; ++i )
    n2[ i ] = i;
Hien_Thi( "Phan tu cua mang n1", n1 );
Hien_Thi( "Phan tu cua mang n2", n2 );
}
function Hien_Thi( tieu_de, ten_mang )
{
    document.writeln( "<H2>" + tieu_de + "</H2>" );
    document.writeln( "<table border = \"1\" width = \"+
        \"100%\">" );
    document.writeln( "<thead><th width = \"100\" +
        \"align = \"left\">Subscript</th>" +
        "<th align = \"left\">Value</th></thead><tbody>");
    for ( var i = 0; i < ten_mang.length; i++ )
        document.writeln( "<tr><td>" + i + "</td><td>" +
            ten_mang[ i ] + "</td></tr>" );
    document.writeln( "</tbody></table>" );
}
// -->
</script>
</HEAD>
<BODY onload = "initializeArrays()"></BODY>
</HTML>
```

Chương 12

Sử dụng mảng trong JavaScript

Trong ví dụ 12.2 ở trên chúng ta đã khai báo 2 mảng *n1* và *n2* thông qua các câu lệnh sau:

```
var n1 = new Array( 5 ); // khai bao mot mang 5 phan tu
```

```
var n2 = new Array(); // Khai bao mot mang rong
```

- Mảng *n1* khi khai báo được chỉ rõ là có 5 phần tử ở trong mảng. Các phần tử lúc này trong mảng *n1* vẫn chưa được gán giá trị.
- Mảng *n2* là một mảng rỗng, chưa có phần tử nào ở trong mảng.
- Để gán giá trị cho các phần tử trong mảng *n1*, chúng ta sử dụng một vòng *for* và thực hiện gán giá trị cho các phần tử ở trong mảng *n1* như sau:

```
for ( var i = 0; i < n1.length; ++i )
```

```
    n1[ i ] = i;
```

- Để truy nhập phần tử trên mảng, chúng ta phải chỉ ra tên mảng và chỉ số của phần tử ở trong mảng. Ở ví dụ này các phần tử là *n1[0]*, *n1[1]*, *n1[2]*, *n1[3]*, *n1[4]*.
- Vì mảng *n2* rỗng nên chúng ta sử dụng một vòng *for* để khởi tạo các phần tử trong mảng đồng thời gán giá trị cụ thể cho các phần tử vừa được tạo ra trong mảng.

```
for ( i = 0; i < 5; ++i )
```

```
    n2[ i ] = i;
```

- Trong ví dụ trên chúng ta xây dựng một hàm:

```
function Hien_Thi( tieu_de, ten_mang )
```

để hiển thị các phần tử của hai mảng *n1* và *n2* trong trình duyệt web. Hàm này gồm có 2 tham số đó là :

tieu_de: để chỉ ra tiêu đề cần xuất ra trong trình duyệt.

ten_mang: để chỉ ra tên mảng cần được hiển thị.

- Trong thân chương trình, ta gọi hàm *Hien_Thi* hai lần với các tham số như sau:

```
Hien_Thi( "Phan tu cua mang n1", n1 );
```

```
Hien_Thi( "Phan tu cua mang n2", n2 );
```

Chương trình trên sẽ cho kết quả như sau trong trình duyệt web:

Chương 12

Sử dụng mảng trong JavaScript

Các phần tử của mảng n1

Subscript	Value
0	0
1	1
2	2
3	3
4	4

Các phần tử của mảng n2

Subscript	Value
0	0
1	1
2	2
3	3
4	4

Hình 12.3. Khởi tạo và gán giá trị cho các phần tử trong mảng.

Ví dụ 12.3

Ở trong ví dụ 12.2 chúng ta khởi tạo và sau đó gán giá trị cho các phần tử trong mảng. Trong ví dụ 12.3 này chúng ta miêu tả khởi tạo mảng và khởi tạo giá trị cho các phần tử trong mảng bằng cách liệt kê các giá trị của chúng.

```
<HTML>
  <HEAD>
    <TITLE>Khoi tao mot mang bang </TITLE>
    <script language = "JavaScript">
      <!--
      function Bat_dau()
      {
        // Khoi tao danh sach cac so mac dinh de
```

Chương 12.

Sử dụng mảng trong JavaScript

```
// gán cho các phần tử trong mảng
var Mau = new Array( "trắng", "vàng", "xanh", "đỏ" );
var Mang1 = [ 2, 4, 6, 8 ];
var Mang2 = [ 2, , 8 ];
Hien_thi( "Các phần tử của mảng Mau:", Mau );
Hien_thi( "Các phần tử của Mang1:", Mang1 );
Hien_thi( "Các phần tử của Mang2:", Mang2 );
}
function Hien_thi( tieu_de, ten_mang )
{
    document.writeln( "<H2>" + tieu_de + "</H2>" );
    document.writeln( "<table border = \"1\">" + "width
        = \"100%\">" );
    document.writeln( "<thead><th width = \"100\" " +
        "<th align = \"left\">Value</th></thead><tbody>" );
    for ( var i = 0; i < ten_mang.length; i++ )
        document.writeln( "<tr><td>" + i + "</td><td>" +
            ten_mang[ i ] + "</td></tr>" );
    document.writeln( "</tbody></table>" );
}
// ..>
</script>
</HEAD><BODY onload = "Bat_dau()"></BODY>
</HTML>
```

Rõ ràng ta thấy một sự khác biệt khi khai báo mảng trong ví dụ này so với ví dụ ở trên. Ở ví dụ 12.3 chúng ta có tất cả là 3 mảng:

- Với mảng *Mau* chúng ta khai báo với cách là gán ngay giá trị của phần tử trong mảng khi khai báo. Có nghĩa là:

Mau[0] = trắng, Mau[1] = vàng, Mau[2] = xanh, Mau[3] = đỏ

Chương 12

Sử dụng mảng trong JavaScript

- Với mảng *Mang1* chúng ta cũng khai báo như cách trên: sử dụng một danh sách số nguyên để khởi tạo cho các phần tử của nó. Cụ thể là có 4 phần tử:

`Mang[0] = 2, Mang[1] = 4, Mang[2] = 6, Mang[3] = 8`

- Mảng *Mang2*: xuất hiện một điểm đáng chú ý khi sử dụng phương thức khai báo này là nếu như trong mảng có một số phần tử không được chỉ ra giá trị thì nó sẽ nhận giá trị *undefined*.

`var Mang2 = [ 2, , 8 ];`

Có hai phần tử không được chỉ định giá trị đó là *Mang2[1]* và *Mang2[2]*. Lúc đó khi chúng ta hiển thị giá trị của chúng trên trình duyệt web sẽ là "undefined".

- Sau khi gọi ba lần hàm *Hien thi* với các tham số phù hợp, chương trình sẽ cho ta kết quả như sau ở trên trình duyệt:

Các phần tử của mảng Mau:	
Subscript	Value
0	trắng
1	vàng
2	xanh
3	đỏ

Các phần tử của Mang1:	
Subscript	Value
0	2
1	4
2	6
3	8

Các phần tử của Mang2:	
Subscript	Value
0	2
1	undefined
2	undefined
3	8

Hình 12.4. Khai báo một mảng.

Chương 12

Sử dụng mảng trong JavaScript

Ví dụ 12.4

Ở hai ví dụ trên chúng ta đã có những phương pháp khai báo một mảng. Trong ví dụ này chúng ta sẽ thao tác với các phần tử trong một mảng.

```
<HTML>
<HEAD>
  <TITLE>Tong cac phan tu cua mot mang</TITLE>
  <script language = "JavaScript">
    <!--
      function Bat_dau()
      {
        var MangInt = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ];
        var tong1 = 0, tong2 = 0;
        for ( var i = 0; i < MangInt.length; i++ )
          tong1 += MangInt[ i ];
        document.writeln( "Tong cua mang la: " + tong1 );
        for ( var phan_tu in MangInt )
          tong2 += MangInt[ phan_tu ];
        document.writeln( "<BR />Tong khi su dung for/in: " +
          tong2 );
      }
    // -->
  </script>
</HEAD><BODY onload = "Bat_dau()"></BODY>
</HTML>
```

Chúng ta khai báo một mảng với 10 phần tử là số nguyên. Chương trình sẽ thực hiện tính tổng của 10 phần tử trong mảng đó.

- Giá trị của các phần tử đã được gán trong khi khởi tạo:

```
var MangInt = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ];
```


Chương 12

Sử dụng mảng trong JavaScript

- Chúng ta có thể tính tổng bằng 2 cách khác nhau:

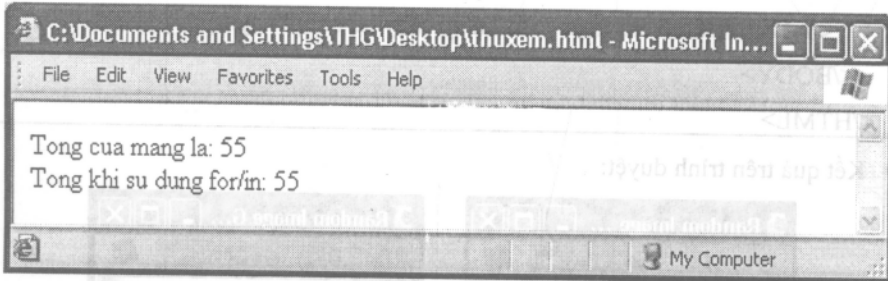
```
for ( var i = 0; i < MangInt.length; i++ )  
    tong1 += MangInt[ i ];
```

- Với vòng lặp *for* này, chúng ta duyệt từ đầu mảng đến cuối mảng. *MangInt.length* trả về cho chúng ta “độ dài” của mảng hay chính là số phần tử có trong mảng. Biến *i* chạy từ 0 đến *MangInt.length* cho phép chúng ta truy cập các phần tử của mảng và cộng dần vào biến *tong1*. Khi vòng lặp kết thúc thì giá trị của biến *tong1* chính là tổng cần tìm của các phần tử trong mảng.
- Với vòng lặp thứ hai cũng cho ta kết quả tương tự, nhưng có sự khác biệt khi truy cập phần tử:

```
for ( var phan_tu in MangInt )  
    tong2 += MangInt[ phan_tu ];
```

Chúng ta dùng một biến *phan_tu* có giá trị bắt đầu từ 0 và truy nhập các phần tử trong mảng mà không cần sử dụng *MangInt.length*. Để làm được điều này chúng ta sử dụng cú pháp *for/in*.

Kết quả của ví dụ 12.4 như sau:



Hình 12.5. Thao tác với các phần tử ở trong mảng.

Ba ví dụ ở trên đã giúp ta hiểu rõ về mảng, qua đó cho ta thấy được ý nghĩa khi sử dụng mảng.

12.4. Sinh ra một ảnh ngẫu nhiên sử dụng mảng

Trong mục này chúng ta sẽ sử dụng mảng để làm cho trang web chúng ta sinh động hơn. Đó là sinh ra một ảnh ngẫu nhiên mỗi khi người dùng duyệt web. Để thực hiện được điều này, ta sử dụng một mảng lưu trữ các ảnh. Các ảnh này được chỉ rõ đường dẫn. Sau đó ta sử dụng hàm toán học *Math.random* để sinh ra một số nguyên như là một chỉ số để truy nhập đến một phần tử của mảng. Chỉ số này ngẫu nhiên vì vậy phần tử được truy nhập trong mảng cũng là ngẫu nhiên. Do vậy sẽ cho chúng ta những hình ảnh khác nhau.

Chương 12

Sử dụng mảng trong JavaScript

Ví dụ 12.5

```
<HTML>
<HEAD>
<TITLE>Sinh ảnh ngẫu nhiên</TITLE>
<script language = "JavaScript">
<!--
var pictures= [ "CPE", "EPT", "GPP", "GUI", "PERF", "PORT", "SEO" ];
document.write ( "<img src = \"\" +
pictures[ Math.floor( Math.random() * 7 ) ] +
\".gif\" width = \"105\" height = \"100\" />\" );
// -->
</script>
</HEAD>
<BODY>
<p>Bam nút refresh hoặc load lại để chạy lại script này</p>
</BODY>
</HTML>
```

- Kết quả trên trình duyệt:



Hình 12.6. Sinh ra ảnh ngẫu nhiên sử dụng mảng.

Chương 12

Sử dụng mảng trong JavaScript

- Trong ví dụ chúng ta khai báo một mảng *picture* có 7 phần tử:

```
var pictures = [ "CPE", "EPT", "GPP", "GUI", "PERF", "PORT", "SEO" ];
```

Mỗi một phần tử là một ảnh. Trong trường hợp này các ảnh được lưu trữ cùng thư mục với trang HTML của chúng ta.

- Để sinh ra một ảnh ngẫu nhiên trong mỗi lần chạy chương trình, chúng ta sử dụng hàm toán học với câu lệnh:

```
document.write ( "<img src = \"\" +  
pictures[ Math.floor( Math.random() * 7 ) ] +  
\".gif\" width = \"105\" height = \"100\" />\" );
```

- Trong đó *Math.floor(Math.random() * 7)* sinh ra một số nguyên từ 0 đến 6. Số nguyên đó được làm chỉ mục để truy nhập đến một phần tử trong mảng *picture*. Và khi gọi đến phần tử nào trong mảng thì ảnh tương ứng sẽ được xuất hiện trong trình duyệt.

12.5. Cách dùng tham số

12.5.1. Vai trò của tham số

Cũng giống như các ngôn ngữ lập trình khác, trong JavaScript luôn sử dụng tham số. Các tham số được khai báo ở đầu các chương trình con dùng để gửi các giá trị vào để chương trình xử lý, cũng như là nơi lấy kết quả ra mà chương trình đã xử lý xong. *Có thể nói tham số là công cụ để chương trình con giao tiếp với môi trường bên ngoài.*

Điểm khác biệt giữa tham số và biến riêng của chương trình con là khi bắt đầu tham gia các câu lệnh, các tham số chỉ có tên chứ chưa có giá trị như các biến. Chỉ khi xuất hiện lời gọi chương trình con các tham số mới được truyền giá trị từ bên ngoài và chương trình con sẽ được hoạt động với bộ giá trị ấy. Điều cần chú ý là giá trị truyền vào phải tương thích kiểu với kiểu của tham số.

Trong JavaScript có hai loại tham số đó là:

- Truyền theo trị.
- Truyền theo biến.

Chú ý rằng việc truyền tham số theo các cách khác nhau nhiều khi cho ra những kết quả khác nhau, dẫn đến những lỗi logic rất khó phát hiện.

Chương 12

Sử dụng mảng trong JavaScript

12.5.2. Truyền theo trị

Việc truyền tham số theo trị được thực hiện qua bản sao. Giá trị bên ngoài (của hằng, biến, hàm hoặc biểu thức) được chép vào một vùng nhớ cấp phát tương ứng với kích thước tham số. Chương trình con sẽ làm việc với dữ liệu trong bản sao này theo những lệnh đã xây dựng cho tham số tương ứng. Vì vậy việc truyền theo trị có những đặc điểm sau:

- Nếu trong chương trình con có những lệnh làm thay đổi giá trị của tham số hình thức thì những thay đổi này không ảnh hưởng gì đến giá trị của biến được truyền ở đầu vào. Nguyên nhân là do những thay đổi này chỉ được thực hiện trên bản sao tương ứng.
- Cho phép giá trị đầu vào có thể là những giá trị của hằng, biến, hàm hoặc biểu thức. Nếu đầu vào truyền hàm hoặc biểu thức thì những giá trị này được tính trước khi truyền cho tham số.

12.5.3. Truyền theo biến

Việc truyền tham số theo biến được thực hiện vào chính địa chỉ của biến được truyền nghĩa là mọi lệnh của chương trình con đối với tham số hình thức cũng chính là các lệnh đối với biến này. Vì vậy truyền theo biến có những đặc điểm sau:

- Nếu trong chương trình con có những lệnh làm thay đổi giá trị của tham số hình thức thì những thay đổi này cũng chính là những thay đổi trên biến được truyền.
- Chỉ cho phép đầu vào là giá trị của biến.

Việc truyền tham số có rất nhiều thuận lợi khi viết chương trình. Một điều quan trọng là ta phải biết khi nào thì sử dụng truyền tham số theo biến, khi nào thì truyền theo trị.

12.6. Truyền mảng vào hàm

Đây thực ra cũng chính là một hình thức truyền tham số (theo biến). Trong đó tên của mảng là một đối số. Cũng giống như truyền tham biến mà ta nói ở trên: khi một hàm mà gọi tham số là một mảng thì mảng đó được truyền như một tham biến. Có nghĩa là khi ta thao tác trên các phần tử của mảng đó, thì các câu lệnh trong hàm sẽ thao tác trực tiếp trên các phần tử của mảng. Và nếu như có những câu lệnh trong hàm làm thay đổi giá trị của các phần tử trong mảng, thì giá trị gốc của các phần tử cũng thay đổi.

Nếu như chúng ta muốn không làm thay đổi các giá trị của các phần tử trong mảng thì chúng ta phải truyền theo trị, có nghĩa là chúng ta chỉ có thể truyền một giá trị trong mảng vào một hàm mà thôi.

Để hiểu rõ cách truyền một mảng theo tham trị và theo biến chúng ta nghiên cứu ví dụ sau.

Chương 12

Sử dụng mảng trong JavaScript

Ví dụ 12.6

```
1  <HTML>
2  <HEAD>
3    <TITLE>Truyen mang va phan tu mang toi mot ham</TITLE>
4    <script language = "JavaScript">
5      <!--
6      function bat_dau()
7      {
8        var a = [ 1, 2, 3, 4, 5 ];
9        document.writeln( "<H2>Mang a duoc truyen theo" +
10          "tham bien</H2>" );
11        hien_thi(
12          "Gia tri goc cua cac phan tu mang a ", a );
13        thay_doi( a ); // mang nhu la mot tham bien
14        hien_thi(
15          "Gia tri thay doi cua mang : ", a );
16        document.writeln( "<H2>Cac phan tu mang a " +
17          "truyen theo tri:</H2>" +
18          "a[3] truoc khi thay_doi: " + a[ 3 ] );
19        thay_doi( a[ 3 ] );
20        document.writeln(
21          "<BR />a[3] sau khi thay_doi: " + a[ 3 ] );
22      }
23      function hien_thi( tieu_de, ten_mang )
24      {
25        document.writeln(
26          tieu_de + ten_mang.join( " " ) + "<BR />" );
27      }
28    </script>
29  </HEAD>
30  <BODY>
31    bat_dau();
32  </BODY>
33  </HTML>
```

Chương 12

Sử dụng mảng trong JavaScript

```
18     function thay_doi( ten_mang )
19     {
20         for ( var j in ten_mang )
21             ten_mang[ j ] *= 2;
22     }
23     function thay_doi1( e )
24     {
25         e *= 2;
26         document.writeln( "<BR />gia tri trong
27             thay_doi1:" + e );
28     }
29     // -->
30     </script>
31 </HEAD><BODY onload = "bat_dau()"></BODY>
32 </HTML>
```

Trong ví dụ trên chúng ta đã xây dựng hai hàm:

- Hàm *thay_doi(ten_mang)*:

```
function thay_doi( ten_mang ) {
    for ( var j in ten_mang )
        ten_mang[ j ] *= 2;
}
```

Ta thấy trong hàm này, tham số được truyền vào là một mảng. Để làm được điều đó chỉ cần chỉ ra tên mảng trong đối số gọi hàm.

Câu lệnh của hàm sẽ tác động đến các phần tử của mảng có tên là *ten_mang* và nhân các phần tử của mảng với 2. Cách truyền tham số theo kiểu này là truyền tham số theo tham biến. Các giá trị của các phần tử trong mảng được truyền vào sẽ thay đổi so với giá trị gốc. Cụ thể sau khi gọi hàm này, thì các phần tử sẽ nhân với 2 và lấy giá trị mới này.

- Hàm *thay_doi1(e)*:

Chương 12

Sử dụng mảng trong JavaScript

```
function thay_doil( e ) {  
    e *= 2;  
    document.writeln( "<BR />gia tri trong thay_doil:" + e );  
}
```

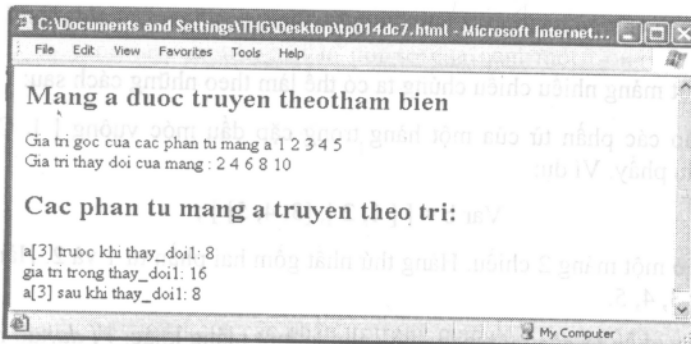
Trong hàm này, tham số truyền vào là một giá trị e . Đây là truyền theo trị. Để làm được điều này, chúng ta chỉ cần chỉ ra một biến ở trong đối số gọi hàm.

Giá trị của biến e sẽ được giữ nguyên sau khi ta thực hiện xong lệnh gọi hàm. Có nghĩa là những câu lệnh làm thay đổi giá trị biến e trong thân hàm: $e *= 2$; sẽ không làm thay đổi giá trị của biến e sau khi thực hiện xong hàm này. Để sử dụng hàm này với mảng, chúng ta phải truyền vào đối số của hàm một trong những giá trị của mảng.

Trong ví dụ 12.6:

- Đầu tiên chúng ta in ra các phần tử gốc của mảng a thông qua hàm *hien_thi* (ở dòng 10). Hàm này sẽ in ra các phần tử gốc của mảng a .
- Sau đó chúng ta gọi hàm *thay_doil(a)*; và tham số truyền vào là mảng a .
- Sau khi thực hiện, chương trình sẽ gọi lại hàm *hien_thi(tieu_de, ten_mang)*. Lúc này sẽ thực hiện in ra giá trị của các phần tử trong mảng a . Chú ý rằng các phần tử trong mảng a bây giờ không phải là những giá trị gốc, chúng đã được nhân 2.
- Ở dòng 13, ta in ra giá trị của phần tử $a[3]$ trong mảng a (sau khi đã thay đổi).
- Ở dòng 14 chúng ta gọi hàm *thay_doil(a[3])*; Lúc này $a[3]$ được truyền vào theo tham trị. Sau khi thực hiện hàm này, giá trị của $a[3]$ không thay đổi, mặc dầu trong hàm *hien_thi(tieu_de, ten_mang)* có câu lệnh làm thay đổi giá trị của $a[3]$.

Kết quả của chương trình được chỉ ra trong hình dưới đây:



Hình 12.7. Các cách truyền mảng vào hàm.

Chương 12

Sử dụng mảng trong JavaScript

Việc sử dụng thành thạo mảng như là một tham biến sẽ mang lại cho chúng ta những thuận lợi lớn khi viết chương trình. Nhưng điều quan trọng hơn là chúng ta phải xem xét khi nào thì sử dụng tham biến và khi nào thì sử dụng tham trị để tránh những sai sót khó phát hiện khi viết.

12.7. Mảng nhiều chiều

Một mảng có thể có nhiều hơn một chiều. Ví dụ, bạn có thể tạo một mảng hai chiều chứa mã nhân viên và tên của nhân viên (*Nhan_vien*) như hình vẽ sau.

		Mảng thứ hai	
		Nhân viên	Mã số
		↓	↓
		0	1
Mảng thứ nhất	0	Nguyen Van A	001
	1	Nguyen Van B	002
	2	Nguyen Van C	003
	3	Nguyen Van D	004
Chi số mảng			

Hình 12.8. Mảng nhiều chiều.

Cộng chiều

Để khai báo một mảng nhiều chiều chúng ta có thể làm theo những cách sau:

- Khai báo các phần tử của một hàng trong cặp dấu móc vuông []. Các hàng cách nhau dấu phẩy. Ví dụ:

```
Var b = [ [ 1, 2 ], [3, 4, 5] ] ;
```

- Khai báo một mảng 2 chiều. Hàng thứ nhất gồm hai phần tử 1 và 2. Hàng thứ hai có 3 phần tử 3, 4, 5.
- Khai báo phần tử của một mảng này như là một mảng khác. Ví dụ: tạo ra một mảng b gồm 2 hàng. Hàng thứ nhất với 5 cột và hàng thứ hai có 3 cột

Chương 12

Sử dụng mảng trong JavaScript

```
var b ;  
b = new Array ( 2 );  
b[0] = new Array ( 5 );  
b[1] = new Array ( 3 );
```

Một phần tử trong mảng nhiều chiều được truy cập bằng cách chỉ ra tên mảng cùng với số chỉ mục của hàng và của cột trong mảng đó.

	Cột 1	Cột 2	Cột 3	Cột 4
Hàng 1	a [0] [0]	a [0] [1]	a [0] [2]	a [0] [3]
Hàng 2	a [1] [0]	a [1] [1]	a [1] [2]	a [1] [3]
Hàng 3	a [2] [0]	a [2] [1]	a [2] [2]	a [2] [3]

Chỉ mục cột

Chỉ mục hàng

Tên mảng

Hình 12.9. Truy cập phần tử trong một mảng nhiều chiều.

Ví dụ 12.7

Ví dụ sau đây sẽ minh họa về mảng nhiều chiều trong JavaScript:

```
<HTML">  
<HEAD>  
<TITLE>Khoi tao mang nhieu chieu</TITLE>  
<script language = "JavaScript">  
<!--  
function bat_dau()  
{  
    var mang1 = [ [ 1, 2, 3 ], // hang dau tien
```

Chương 12

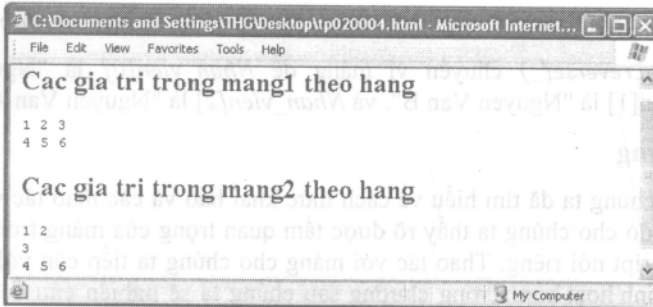
Sử dụng mảng trong JavaScript

```
        [ 4, 5, 6 ] ]; // hàng thứ hai
var mang2 = [ [ 1, 2 ],      // hàng đầu tiên
              [ 3 ],         // hàng thứ hai
              [ 4, 5, 6 ] ]; // hàng thứ 3
ham_ra( "Các giá trị trong mang1 theo hàng",
        mang1 );
ham_ra("Các giá trị trong mang2 theo hàng",mang2);
}
function ham_ra( tieu_de, ten_mang )
{
    document.writeln( "<H2>" + tieu_de + "</H2><tt>" );
    for ( var i in ten_mang ) {
        for ( var j in ten_mang[ i ] )
            document.write( ten_mang[ i ][ j ] + " " );
        document.writeln( "<BR />" );
    }
    document.writeln( "</tt>" );
}
// -->
</script>
</HEAD><BODY onload = "bat_dau()"></BODY>
</HTML>
```

Kết quả của ví dụ 12.7 được thể hiện trong trình duyệt như sau:

Chương 12

Sử dụng mảng trong JavaScript



Hình 12.10. Thao tác với mảng nhiều chiều.

12.8. Các phương thức thao tác với mảng

Để thao tác với mảng bạn sẽ phải dùng đến các phương thức của đối tượng mảng. Để kết chương này, chúng ta sẽ tìm hiểu về các phương thức hay sử dụng với mảng. Bảng dưới đây liệt kê các phương thức đó.

Bảng 12.1. Các phương thức thao tác với mảng trong JavaScript

Phương thức	Mô tả
Concat	Nối hai mảng để tạo một mảng thứ ba
Join	Kết hợp các phần tử của một mảng thành một chuỗi
Pop	Trả về phần tử cuối cùng của một mảng sau khi xóa nó từ mảng đó
push	Thêm một hay nhiều phần tử vào cuối một mảng. Nó trả về phần tử cuối cùng được cộng thêm vào
reverse	Đảo ngược các phần tử của một mảng: phần tử mảng đầu tiên trở thành phần tử cuối cùng và phần tử cuối cùng trở thành phần tử đầu tiên
shift	Xoá phần tử đầu tiên từ một mảng và trả lại phần tử đó
sort	Sắp xếp các phần tử của một mảng

Ví dụ:

Giả sử bạn xác định mảng sau đây:

```
Nhan_vien = new Array(["Nguyen Van A", "Nguyen Van B", "Nguyen Van C"]);
```

Khi đó các phương thức của mảng sẽ cho ta kết quả tương ứng như sau:

Chương 12

Sử dụng mảng trong JavaScript

- *Nhan_vien.join ()* trả về "Nguyen Van A, Nguyen Van B, Nguyen Van C"
- *Nhan_vien.reverse()* chuyển vị mảng để *Nhan_vien[0]* là "Nguyen Van C", *Nhan_vien[1]* là "Nguyen Van B", và *Nhan_vien[2]* là "Nguyen Van A"

12.9. Kết chương

Như vậy chúng ta đã tìm hiểu về cách thức khai báo và các thao tác với mảng trong JavaScript. Qua đó cho chúng ta thấy rõ được tầm quan trọng của mảng trong lập trình nói chung và JavaScript nói riêng. Thao tác với mảng cho chúng ta tiếp cận với các đối tượng dễ dàng hơn và linh hoạt hơn. Trong chương sau chúng ta sẽ nghiên cứu về các đối tượng cơ bản trong JavaScript. Đây cũng là một phần rất quan trọng của JavaScript.

Chương 13
Các đối tượng JavaScript

Nội dung của chương:

- Giới thiệu về khái niệm đối tượng (object).
- Các đối tượng của trình duyệt.
- Các đối tượng của JavaScript.
- Cách thức sử dụng các đối tượng đó.

13.1. Giới thiệu

Khi tạo một trang web bạn có thể dùng các đối tượng do trình duyệt cung cấp, các đối tượng do ngôn ngữ thao tác cung cấp và các phần tử HTML. Trong chương này bạn sẽ học về các đối tượng khác nhau mà bạn có thể chèn vào một tài liệu HTML.

Trong JavaScript nói riêng, bạn có thể thao tác với nhiều thành phần trong tài liệu XHTML thông qua một script, sử dụng những đối tượng được JavaScript cung cấp.

13.2. Các đối tượng (Object)

Một đối tượng bao gồm những đặc tính sau:

- Được định nghĩa như là một gói dữ liệu đầy đủ.
- Thuộc tính (các biến) định nghĩa đối tượng đó.
- Các phương thức tác động lên dữ liệu.
- Những thông tin về đối tượng đó.

Ví dụ: Một chiếc xe hơi là một đối tượng:

- Các thuộc tính của đối tượng xe hơi này là mẫu mã, màu sắc.
- Các phương thức của đối tượng xe hơi này là *đi()*, *phanh()*, *lùi()*.

Chương 13

Các đối tượng JavaScript

Truy cập đối tượng

- Để truy cập các thuộc tính của một đối tượng bạn phải chỉ ra tên đối tượng và thuộc tính của nó: *Tên_doi_tuong.Ten_thuoc_tinh*;

Ví dụ: *document.bgcolor* trong đó *bgcolor* là thuộc tính của đối tượng *document*

- Để truy cập các phương thức của một đối tượng bạn phải chỉ ra tên đối tượng và phương thức được yêu cầu: *Tên_doi_tuong. Phương_thức ()*;

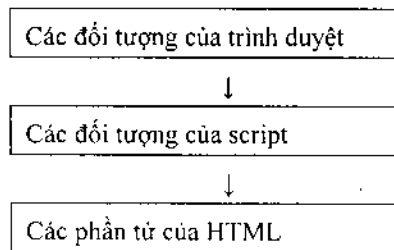
Ví dụ: Một chiếc xe hơi là một đối tượng. Các thuộc tính của đối tượng xe hơi này là mẫu mã, màu sắc. Các phương thức của đối tượng xe hơi này là *đi ()*, *phanh ()*, *lùi ()*.

Sự phân cấp đối tượng

Khi tạo ra một trang web bạn có thể chèn:

- Các đối tượng của trình duyệt.
- Các đối tượng của ngôn ngữ xây dựng sẵn.
- Các phần tử của HTML.

Bạn cũng có thể tự tạo đối tượng của riêng mình để dùng trong chương trình.



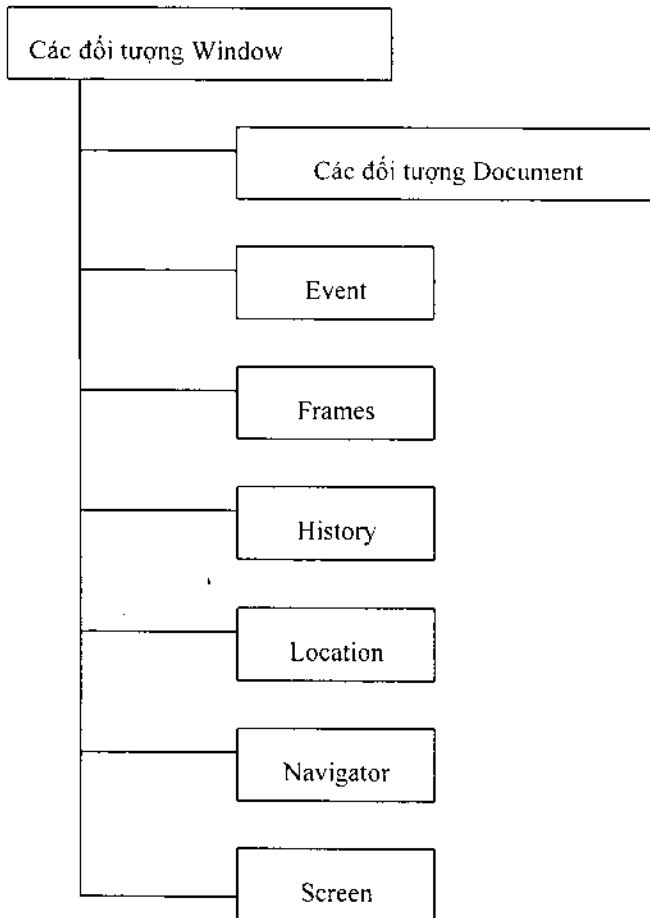
Hình 13.1. Sự phân cấp của đối tượng.

Hình vẽ trên minh họa sự phân cấp của đối tượng. Biết được sự phân cấp này là quan trọng bởi vì các đối tượng được đánh giá nhờ tham chiếu tới sự phân cấp trên. Ví dụ, để đánh giá một đối tượng FORM(biểu mẫu) bạn cần phải chỉ ra tên của biểu mẫu đó và đối tượng chứa nó, tức là tài liệu như sau: *document.form1*.

Trước khi đi vào nghiên cứu các đối tượng của JavaScript, chúng ta sẽ tìm hiểu về những đối tượng của trình duyệt.

13.3. Các đối tượng của trình duyệt

Trình duyệt là một trình ứng dụng được dùng để hiển thị nội dung của một tài liệu HTML. Các trình duyệt cung cấp một số đối tượng có thể lấy và sử dụng trong script. Hình vẽ sau miêu tả các đối tượng trình duyệt của IE (*Internet Explore*). (Trong phạm vi của giáo trình này chúng ta chỉ nghiên cứu về *Internet Explore*, hơn nữa từ khi trình duyệt *Internet Explore* ra đời, nó đã hoàn toàn như thay thế trình duyệt *Netscape*.)



Hình 13.2. Các đối tượng của trình duyệt.

Các đối tượng của trình duyệt được miêu tả trong bảng sau.

Chương 13

Các đối tượng JavaScript

Bảng 13.1. Các đối tượng của trình duyệt

Tên đối tượng	Mô tả
Window	Để truy nhập cửa sổ trình duyệt
Document	Để truy nhập tài liệu hiển thị trên trình duyệt. Nó được dùng để truy nhập nội dung được hiển thị trong phần HEAD và BODY
Event	Thể hiện trạng thái của một sự kiện chẳng hạn như phần tử xuất hiện sự kiện, trạng thái của các phím trên bàn phím,...
Frames	Để truy nhập một khung HTML
History	Lưu giữ dấu vết của URL được lấy trong cửa sổ hiện thời
Location	Biểu thị URL
Navigator	Để truy nhập thông tin về trình duyệt đang thực hiện một script
screen	Để truy nhập thông tin về kích cỡ và độ sâu về màu của một màn hình mà người dùng sử dụng

Trong các đối tượng của trình duyệt miêu tả ở trên chúng ta sẽ tìm hiểu các thuộc tính và phương thức của một số đối tượng đặc trưng.

13.3.1. Đối tượng Window

Đối tượng Window thể hiện các cửa sổ trình duyệt và có thể được dùng để lấy thông tin về trạng thái của cửa sổ. Nó được dùng để hiển thị đối tượng *document* và truy nhập các sự kiện xuất hiện trong cửa sổ, truy nhập các đặc điểm của trình duyệt có ảnh hưởng tới cửa sổ.

Các thuộc tính

Bảng 13.2. Các thuộc tính của đối tượng Window

Tên	Mô tả
Close	Nhận diện xem thử cửa sổ có đóng không
defaultStatus	Thông báo mặc định xuất hiện trên thanh trạng thái của trình duyệt
Status	Thông báo tạm thời xuất hiện trên thanh trạng thái của trình duyệt
Frames	Một mảng lưu giữ tất cả đối tượng khung trong cửa sổ
Location	Vị trí nguồn của tài liệu trong cửa sổ

Chương 13

Các đối tượng JavaScript

Name	Tên của sổ mà nó nhận diện
top	Tham chiếu tới cửa sổ bậc cao nhất
self, window	Nhận diện cửa sổ hiện thời đang được tham chiếu đến
parent	Tham chiếu tới một cửa sổ chứa một cửa sổ khác

Các phương thức của đối tượng Window

Các phương thức của đối tượng Window được mô tả trong bảng sau.

Bảng 13.3. Các phương thức của đối tượng Window

Tên	Mô tả
alert	Hiển thị một hộp chứa một thông báo do ứng dụng định nghĩa
blur	Làm cho một đối tượng mất tiêu điểm và bắt đầu sự kiện <i>onblur</i>
close	Đóng cửa sổ trình duyệt hay ứng dụng HTML hiện thời
focus	Làm cho việc điều khiển nhận tiêu điểm và thực thi mã lệnh cho sự kiện <i>onfocus</i> chỉ ra
open	Mở một cửa sổ mới và tải tài liệu được chỉ ra bởi một URL, hoặc mở một tài liệu trống nếu không có một URL nào được chỉ tới

13.3.2. Đối tượng document

Đối tượng *document* thể hiện tài liệu HTML trong một cửa sổ trình duyệt. Được dùng để lấy thông tin về tài liệu, kiểm tra, sửa đổi các phần tử HTML, văn bản trong tài liệu và để xử lý dữ kiện.

Các thuộc tính

Các thuộc tính của đối tượng *document* được mô tả trong bảng sau.

Bảng 13.4. Các thuộc tính của đối tượng document

Tên	Mô tả
alinkColor	Xác lập hay lấy màu của tất cả các liên kết động trong tài liệu
bgColor	Xác lập hay lấy màu nền của đối tượng tài liệu
BODY	Chỉ ra phần đầu và phần cuối của thân tài liệu
fgColor	Xác lập hay lấy màu (văn bản) nền trên cùng của tài liệu
linkColor	Xác lập hay lấy màu của các liên kết tài liệu

Chương 13

Các đối tượng JavaScript

location	Chứa các thông tin về URL hiện tại
TITLE	Chứa tiêu đề của tài liệu
URL	Xác lập hay lấy URL cho tài liệu hiện tại
linkColor	Xác lập hay lấy màu của các liên kết mà người dùng đã đến thăm

Các phương thức

Các phương thức của đối tượng *document* được mô tả trong bảng sau.

Bảng 13.5. Các phương thức của đối tượng document

Tên	Thuộc tính
clear	Xoá tài liệu hiện thời
close	Đóng một luồng dữ liệu xuất và buộc dữ liệu được gửi đi hiển thị
open	Mở một tài liệu để thu thập dữ liệu xuất của các phương thức <i>write</i> và <i>writeln</i>
write	Viết một hay nhiều biểu thức HTML đến một tài liệu nằm trong một cửa sổ được chỉ ra
writeln	Viết một hay nhiều biểu thức HTML được theo sau bởi một dấu xuống dòng đến một tài liệu nằm trong cửa sổ được chỉ ra

13.3.3. Đối tượng Frame

Frame (khung) chia một cửa sổ trình duyệt web thành các vùng riêng biệt, mỗi vùng có thể hiển thị một trang riêng có thanh cuộn. Mỗi *Frame* là một cửa sổ duy nhất trong cửa sổ chính.

Để truy nhập nội dung của khung sử dụng đối tượng.

Các thuộc tính

Các thuộc tính của đối tượng *frame* được mô tả trong bảng sau.

Bảng 13.6. Các thuộc tính của đối tượng Frame

Tên	Mô tả
borderColor	Xác lập hay lấy màu viền của thẻ <FRAME>
frameBorder	Xác lập hay lấy để hiển thị viền cho khung
height	Xác lập hay lấy độ cao của thẻ <FRAME>

Chương 13

Các đối tượng JavaScript

id	Lấy chuỗi nhận diện thẻ <FRAME>
marginHeight	Xác lập hay lấy chiều cao của lề trên và lề dưới trước khi hiển thị trong khung
marginWidth	Xác lập hay lấy độ rộng của lề trái và phải trước khi hiển thị văn bản trong khung
name	Xác lập hay lấy tên cửa sổ hay tên khung
scrolling	Xác lập hay lấy xem thứ khung có thể cuộn được hay không
src	Xác lập hay lấy một URL cần tải bởi thẻ <FRAME>
style	Thẻ hiển xác lập hiện thời của tất cả kiểu trên dòng (inline styles) khá dī cho một phần tử đã cho
width	Xác lập hay lấy độ rộng tính được của thẻ <FRAME>

13.4. Các đối tượng của JavaScript

JavaScript có nhiều đối tượng xây dựng sẵn mà bạn có thể dùng trong script của mình. Dĩ nhiên bạn có thể tạo cho mình những đối tượng riêng. Hai đối tượng có sẵn được dùng nhiều nhất là: *string*(chuỗi) và *date* (ngày).

13.4.1. Đối tượng toán học (Math Object)

Cũng giống như các ngôn ngữ lập trình khác, JavaScript cũng có lớp toán học. Lớp này cho phép người lập trình thực hiện được nhiều câu lệnh tính toán số học. Lớp toán học được xây dựng và cung cấp cho chúng ta nhiều phương thức và thuộc tính rất thuận lợi để xây dựng chương trình. Cụ thể là cung cấp cho người dùng những hàm số học quan trọng mà người lập trình không phải xây dựng lại khi viết chương trình.

Để truy nhập vào phương thức và thuộc tính của lớp này phải tuân theo qui tắc sau:

MathObject.method();

MathObject.property ;

Các phương thức của lớp Math

Các phương thức của lớp Math được chỉ ra trong bảng dưới đây:

Bảng 13.7. Các phương thức của lớp Math

Phương thức	Miêu tả	Ví dụ
abs(x)	Trị tuyệt đối của x	abs (7.1) = 7.1 abs (-5.6) = 5.6

Chương 13

Các đối tượng JavaScript

<code>ceil (x)</code>	Trả về số nguyên nhỏ nhất lớn hơn hoặc bằng giá trị x	<code>ceil (9.2) = 10.0</code> <code>ceil (- 9.7) = -9</code>
<code>cos (x)</code>	Giá trị $\cos(x)$ của góc x (x tính bằng radian).	<code>cos (0.0) = 1.0</code>
<code>exp (x)</code>	Trả về giá trị của phép toán : e mũ x.	<code>exp (1.0) = 2.17828</code>
<code>floor (x)</code>	Giá trị nguyên bé nhất bé hơn hoặc bằng x.	<code>floor (9.2) = 9.0</code> <code>floor (-9.8) = -10.0</code>
<code>log (x)</code>	Trả về giá trị là logarit cơ số e của x.	<code>log (2.71828) = 1.0</code>
<code>max (x , y)</code>	Trả về số lớn hơn trong hai số x và y .	<code>max (2.3, 12.7) = 12.7</code>
<code>min (x , y)</code>	Trả về số nhỏ hơn trong hai số x và y.	<code>min (2.3 , 12.7) = 2.3</code>
<code>pow (x, y)</code>	Trả về giá trị của phép toán học x mũ y.	<code>pow (2.0 , 7.0) = 128.0</code>
<code>round (x)</code>	Trả về số nguyên gần x nhất.	<code>round (9.75) = 10</code> <code>round (9.25) = 9</code>
<code>sin (x)</code>	Trả về phép toán lượng giác sin (x). x tính bằng radian.	<code>sin (0.0) = 0.0</code>
<code>sqrt (x)</code>	Trả về giá trị là căn bậc 2 của x.	<code>sqrt (9.0) = 3.0</code>
<code>tan (x)</code>	Trả về giá trị của hàm lượng giác tang (x). x bằng radian.	<code>tan (0.0) = 0.0</code>

Những thuộc tính của lớp toán học (Math)

Các thuộc tính của lớp Math được chỉ ra trong bảng dưới đây:

Bảng 13.8. Các thuộc tính của lớp Math

Hằng số	Miêu tả	Giá trị
Math.E	Cơ số tự nhiên e.	Gần bằng 2.718
Math.LN2	$\ln(2)$.	Gần bằng 0.693
Math.LN10	$\ln(10)$.	Gần bằng 2.302

Chương 13

Các đối tượng JavaScript

Math.LOG2E	Logarit cơ số 2 của E.	Gần bằng 1.442
Math.LOG10E	Logarit cơ số 10 của E.	Gần bằng 0.434
Math.PI	Giá trị của Π .	Gần bằng 3.14159265358973

13.4.2. Đối tượng String

Đối tượng *string* được dùng để xử lý và làm việc với các chuỗi văn bản. Trong một chương trình bạn có thể lấy các chuỗi con và chuyển văn bản thành chữ hoa hay chữ thường. Cú pháp tổng quát như sau:

Tên chuỗi.Tên thuộc tính;

hoặc : *Tên chuỗi.Tên phương thức;*

Có ba phương thức khác nhau để tạo các chuỗi:

- Dùng câu lệnh var để gán cho nó một giá trị (không bắt buộc). Ví dụ:

Var newstr = "Day la mot xau";

- Dùng một toán tử gán = cùng với tên biến. Ví dụ:

nwstr = "Day la mot xau";

- Dùng hàm thiết lập String (). Ví dụ:

Var newstr = String (" Day la mot xau ");

Các phương thức của lớp *String* được miêu tả trong bảng sau.

Bảng 13.9. Các phương thức của lớp String

Tên phương thức	Mô tả
CharAt (index)	Trả về một chuỗi chứa những chữ cái bắt đầu từ phân tử có chỉ số <i>index</i> . Nếu không có chữ cái nào ở vị trí <i>index</i> , <i>charAt</i> trả về một <i>string</i> rỗng. Chữ cái đầu tiên có chỉ mục là 0.
CharCodeAt(index)	Trả về giá trị unicode của chữ cái ở vị trí có chỉ mục <i>index</i> . Nếu không có chữ cái nào ở vị trí <i>index</i> thì <i>charCodeAt</i> trả về NaN (<i>Not a Number</i>).
Concat (string)	Nối một xâu vào cuối một xâu khác.

Chương 13

Các đối tượng JavaScript

IndexOf (subString, index)	Tìm xâu <i>subString</i> bắt đầu từ vị trí <i>index</i> trong xâu được gọi phương thức. Trả về vị trí bắt đầu của <i>subString</i> trong xâu nguồn hoặc bằng -1 nếu như xâu <i>subString</i> không tìm thấy. Nếu chỉ mục <i>index</i> không có khi gọi phương thức này thì sẽ bắt đầu tìm kiếm từ vị trí có chỉ mục 0 của xâu gọi phương thức này.
Slice (start , end)	Trả về một xâu con bắt đầu từ vị trí có chỉ số <i>start</i> đến vị trí có chỉ số <i>end</i> trong xâu được gọi phương thức này. Nếu chỉ số <i>end</i> không được cung cấp thì sẽ trả về một xâu từ vị trí <i>start</i> cho đến vị trí cuối cùng trong xâu đích.
Split (string)	Tách xâu nguồn thành một mảng <i>string</i> , trong đó tham số <i>string</i> làm dấu hiệu để phân tách.
lastIndexOf(subString, index)	Trả lại vị trí xuất hiện sau cùng của chuỗi con.
Substr(start , length)	Trả về một xâu con bắt đầu từ vị trí có chỉ số <i>start</i> trong xâu nguồn và có chiều dài là <i>length</i> . Nếu không có tham số <i>length</i> thì trả về xâu con từ vị trí có chỉ số <i>start</i> đến hết xâu nguồn.
ToLowerCase ()	Chuyển các chuỗi thành chữ thường.
ToUpperCase ()	Chuyển các chuỗi thành chữ hoa.
ToString ()	Trả về một xâu giống như xâu nguồn.
Big	Tăng kích thước của văn bản trong chuỗi
Blink	Làm cho văn bản trong chuỗi nhấp nháy. (Internet Explorer không hỗ trợ phương thức này.)
Bold	Làm cho văn bản trong chuỗi trở nên đậm.
Fontcolor	Xác định màu của phông chữ.
Italics	Hiện thị chuỗi dưới dạng chữ <i>ngiêng</i> .
Small	Giảm kích thước của văn bản.
Strike	Hiện thị văn bản bị gạch ngang.
Sub	Hiện thị văn bản ở dạng chỉ số dưới.
Sup	Hiện thị văn bản ở dạng chỉ số trên.

Chương 13

Các đối tượng JavaScript

Trong các phương thức ở trên, chúng ta phải để ý đến những phương thức thao tác với ký tự. Đó là những phương thức:

- *CharAt*: trả về một ký tự ở vị trí chỉ định.
- *CharCodeAt*: trả về mã unicode của ký tự ở vị trí chỉ định.
- *FromCharCode* trả về một xâu được tạo bởi một chuỗi giá trị unicode.
- *toLowerCase*: trả về một xâu với chữ thường
- *toUpperCase*: trả về một xâu với chữ hoa.

Ví dụ 13.1

Chúng ta đã tìm hiểu về các phương thức trong đối tượng String. Để hiểu hơn về chức năng của từng phương thức, chúng ta cùng tìm hiểu qua những ví dụ sau:

```
<HTML>
<HEAD>
<TITLE>Phương thức xử lý ký tự</TITLE>
<script language = "JavaScript">
  <!--
  var s = "ZEBRA";
  var s2 = "AbCdEfG";
  document.writeln("<P>Ký tự ở vị trí có chỉ số 0 trong '"
    + s + "' là '" + s.charAt( 0 ) );
  document.writeln( "<BR />Mã unicode của ký tự ở vị trí 0
    trong '" + s + "' là '" + s.charCodeAt( 0 ) - "<P>" );
  document.writeln( "<P>" -
    String.fromCharCode( 87, 79, 82, 68 ) -
    "' chưa nhưng ký tự có mã 87, 79, 82 và 68</P>" )
  document.writeln( "<P>" + s2 + "' viết thường là '" +
    s2.toLowerCase() + "'" );
  document.writeln( "<BR />" + s2 + "' viết hoa là '"
```

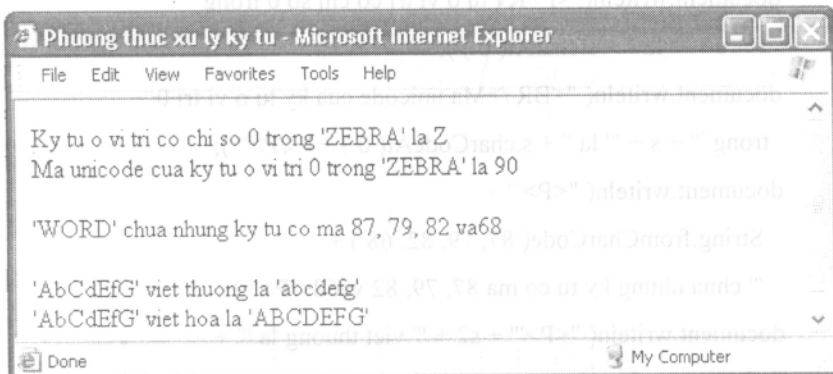
Chương 13

Các đối tượng JavaScript

```
+ s2.toUpperCase() + "</P>" );  
  
// -->  
  
</script>  
</HEAD><BODY></BODY>  
</HTML>
```

- Trong ví dụ 13.1 chúng ta khai báo 2 chuỗi *s* và *s2*. Sau đó dùng hàm *charAt* để lấy ký tự đầu tiên trong chuỗi *s*, thông qua câu lệnh sau
`document.writeln("<P>Ký tự ở vị trí có chỉ số 0 trong " + s + " là " + s.charAt(0));`
- Đồng thời trả về mã unicode của ký tự này bằng cách gọi phương thức *charCodeAt*:
`document.writeln( "<BR />Mã unicode của ký tự ở vị trí 0 trong " + s + " là " + s.charCodeAt( 0 ) + "</P>" );`
- Tiếp theo là thành lập một chuỗi từ những chữ cái có mã unicode là 87, 79, 82 và 68. Để làm được như vậy, cần gọi phương thức *fromCharCode*:
`document.writeln("<P>" + String.fromCharCode(87, 79, 82, 68) +
" chưa nhưng ký tự có mã 87, 79, 82 và 68</P>")`
- Và cuối cùng là chuyển chuỗi *s2* về chữ thường và chữ hoa thông qua hai phương thức *toLowerCase* và *toUpperCase*.

Kết quả của ví dụ trên trong trình duyệt IE như hình vẽ sau:



Hình 13.3. Sử dụng các phương thức trong lớp String.

Chương 13

Các đối tượng JavaScript

Ví dụ 13.2

```
<HTML>
<HEAD>
  <TITLE>phuong thuc split va substring</TITLE>
  <script language = "JavaScript">
    <!--
    function Ham_xu_ly()
    {
      var strings = Form.xau_nhap_vao.value.split( " " );
      Form.output.value = strings.join( "\n" );
      Form.xau_ra.value =
        Form.xau_nhap_vao.value.substring( 0, 10 );
    }
    // -->
  </script>
</HEAD>
<BODY>
  <FORM name = "Form" action = "">
    <P>Nhap vao mot xau de xu ly:<BR />
    <input name = "xau_nhap_vao" type = "text" size = "40" />
    <input name = "splitButton" type = "button" value =
      "Split" onclick = "Ham_xu_ly()" /></P>
    <P>Xau sau khi cat la:<BR />
    <textarea name = "output" rows = "8" cols = "34">
    </textarea></P>
    <P>10 chu cai dau tien o trong xau la
    <input name = "xau_ra" type = "text" size = "15" /></P>
  </FORM>
</BODY>
</HTML>
```

Chương 13

Các đối tượng JavaScript

Trong ví dụ này, chúng ta sử dụng 2 phương thức *split* và *substring* trong JavaScript. Hai phương thức này đều xử lý trên chuỗi.

- Đầu tiên chương trình xây dựng một hàm có tên là *Ham_xu_ly()* thực hiện tách chuỗi đã nhập vào thành các chuỗi con với các câu lệnh trong thân hàm như sau:

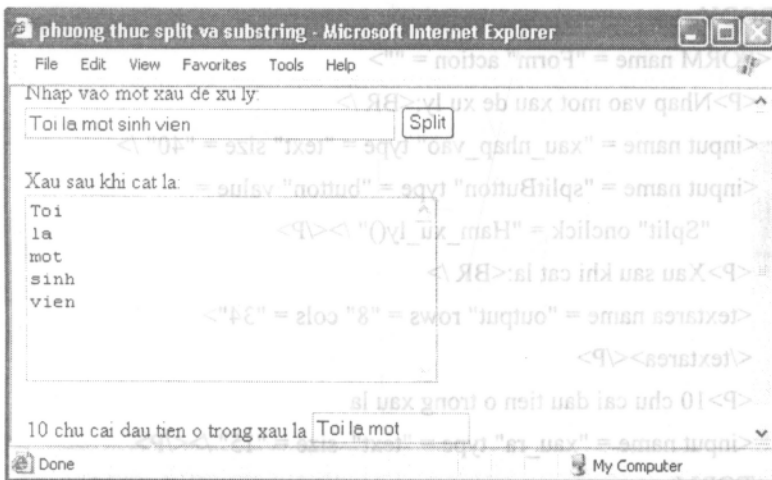
```
var strings = Form.xau_nhap_vao.value.split("");
```

- Tham số truyền vào phương thức là một *string*, ở ví dụ này là dấu cách. Điều này có nghĩa là duyệt từ đầu cho đến cuối chuỗi nhập vào, nếu gặp ký tự cách thì sẽ tách chuỗi trước nó thành một chuỗi con. Chẳng hạn như chuỗi nhập vào là “Toi la mot sinh vien” thì nó sẽ trả về các chuỗi con là: “Toi”, “la”, “mot”, “sinh”, “vien”.
- Trong câu lệnh sau, chuỗi nhập vào gọi phương thức *substring(0,10)*.

```
Form.xau_ra.value = Form.xau_nhap_vao.value.substring(0,10);
```

Hai tham số được truyền vào này có ý nghĩa như sau: bắt đầu duyệt từ vị trí đầu chuỗi mới nhập vào và tách ra một chuỗi con với chiều dài 10 ký tự. Chẳng hạn như chuỗi nhập vào là “Toi la mot sinh vien” thì sẽ trả về chuỗi có 10 ký tự đầu tiên của chuỗi đó, là chuỗi: “Toi la mot”.

Kết quả của ví dụ trên thể hiện trong trình duyệt IE như sau:



Hình 13.4. Sử dụng các phương thức trong lớp String.

Những phương thức đánh dấu XHTML

Dưới đây là một số phương thức đặc trưng sinh ra thẻ XHTML.

Chương 13
Các đối tượng JavaScript

Bảng 13.10. Những phương thức đánh dấu XHTML

Tên thẻ	Cách dùng	Ý nghĩa
Anchor	< a name = "top" > Anchor < /a >	Một điểm đánh dấu (điểm neo)
Blink	< blink > chu nhap nhay < /blink >	Hiển thị văn bản dưới dạng chữ nhấp nháy
Fixed	< tt > chu in < /tt >	Hiển thị chữ in
Strike	< strike > chu gạch dưới < /strike >	Hiển thị văn bản dưới dạng chữ gạch dưới
subscript	< sub > subscript < /sub >	Hiển thị văn bản dưới dạng chỉ số dưới
superscript	< sup > supercript < /sub>	Hiển thị văn bản dưới dạng chỉ số trên

Để minh họa cho các phương thức này chúng ta xem kết quả của ví dụ sau.

Ví dụ 13.3

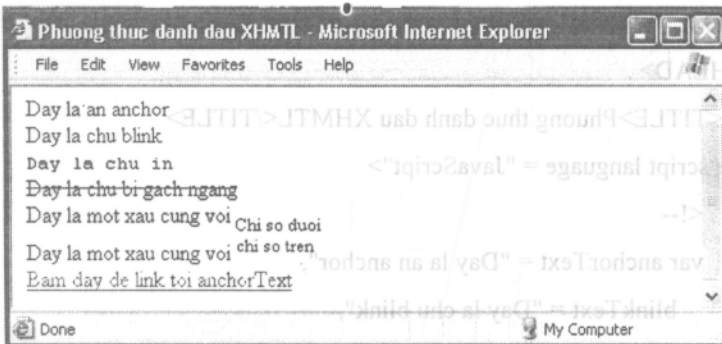
```
<HTML>
<HEAD>
  <TITLE>Phương thức đánh dấu XHTML</TITLE>
  <script language = "JavaScript">
    <!--
    var anchorText = "Đây là an anchor",
        blinkText = "Đây là chu blink",
        fixedText = "Đây là chu in",
        linkText = "Bam đây de link toi anchorText",
        strikeText = "Đây là chu bị gạch ngang",
        subText = "Chỉ số dưới",
        supText = "Chỉ số trên";
    document.writeln( anchorText.anchor( "top" ) );
    document.writeln( "<BR />" + blinkText.blink() );
```

Chương 13

Các đối tượng JavaScript

```
document.writeln( "<BR />" + fixedText.fixed() );  
document.writeln( "<BR />" + strikeText.strike() );  
document.writeln(  
    "<BR />Day la mot xau cung voi " + subText.sub() );  
document.writeln(  
    "<BR />Day la mot xau cung voi " + supText.sup() );  
document.writeln(  
    "<BR />" + linkText.link( "#top" ) );  
// -->  
</script>  
</HEAD><BODY></BODY>  
</HTML>
```

Kết quả hiển thị trên trình duyệt:



Hình 13.5. Minh họa cho phương thức đánh dấu XHTML.

13.4.3. Đối tượng Date

Đối tượng *Date* chứa thông tin cả về ngày tháng và giờ. Đối tượng ngày không có thuộc tính nào, tuy nhiên nó có một số lượng lớn các phương thức để xác lập, lấy và xử lý.

`DateObject = new Date (tham số);`

Trong đó:

- *DateObject*: là một biến dữ liệu để lưu giữ đối tượng ngày mới,

Chương 13

Các đối tượng JavaScript

▪ **Tham số:** có thể bất kì trong những giá trị sau:

- Bỏ qua - nếu ngày được chỉ ra mà không có tham số nào, nó lấy ngày và giờ hiện thời. Ví dụ, `today = new Date ( )`.
- Chuỗi - thể hiện ngày và giờ dưới dạng: "MM DD, YYYY, hh:mm:ss". Ví dụ, `tdate = new Date ("July 29, 1998, 10:30:00")`.
- Một tập hợp giá trị cho năm, tháng, ngày, giờ, phút, và giây. Ví dụ, `tdate = new Date (98, 06, 29, 10, 30, 0)`.
- Một tập hợp các giá trị số nguyên chỉ cho năm, tháng và ngày. Ví dụ, `tdate = new Date (98, 06, 29)`.

Dưới đây là các nhóm phương thức:

Bảng 13.11. Những phương thức đối tượng Date

Phương thức	Mô tả
Set	Các phương thức này được dùng để xác lập giá trị thời gian và giá trị ngày
Get	Các phương thức này được dùng để lấy giá trị thời gian và giá trị ngày
To	Các phương thức này được dùng để trả về các giá trị chuỗi từ các đối tượng ngày
Parse and UTC	Các phương thức này được dùng để phân tích các chuỗi

Các giá trị nguyên dưới đây được dùng để thể hiện các giá trị này

Bảng 13.12. Các giá trị của đối tượng Date

Các giá trị	Số nguyên
Giây và phút	0 đến 59
Giờ	0 đến 23
Ngày	0 đến 6 (ngày của tuần)
Ngày tháng năm	1 đến 31 (ngày của tháng)
Tháng	0 đến 11 (Tháng 1 đến tháng 12)
Năm	Các năm từ 1900

Chương 13

Các đối tượng JavaScript

Các phương thức Get có trong đối tượng Date

Bảng 13.13. Các phương thức Get có trong đối tượng Date

Phương thức	Mô tả
getDate	Trả về ngày của tháng từ đối tượng ngày tháng năm (1-31)
getDay	Trả về ngày của tuần từ đối tượng ngày tháng năm (0-6)
getHours	Trả về giờ từ đối tượng ngày tháng năm (0-23)
getMinutes	Trả về phút từ đối tượng ngày tháng năm (0-59)
getSeconds	Trả về giây từ đối tượng ngày tháng năm (0-59)
getMonth	Trả về tháng từ đối tượng ngày tháng năm (0-11)
getFullYear	Trả về năm từ đối tượng ngày tháng năm (Năm hiện thời (-) 1900)
getTime	Trả về số triệu giây từ Epoch (11/7/1970)
getTimeZoneOffset	Trả về sự khác nhau theo phút giữa giờ địa phương và giờ (GMT)

Các phương thức Set có trong đối tượng Date

Bảng 13.14. Các phương thức Set có trong đối tượng Date

Phương thức	Mô tả
setDate	Xác lập này của tháng cho một đối tượng ngày tháng năm
setHours	Xác lập giờ cho một đối tượng ngày tháng năm (0-23)
setMinutes	Xác lập phút cho một đối tượng ngày tháng năm (0-59)
setSeconds	Xác lập giây cho một đối tượng ngày tháng năm (0-59)
setTime	Xác lập giá trị thời gian trong một đối tượng ngày tháng năm
setMonth	Xác lập tháng trong một đối tượng ngày tháng năm (1-12)
setYear	Xác lập giá trị năm trong đối tượng ngày tháng năm. Giá trị năm phải lớn hơn 1900

Chương 13

Các đối tượng JavaScript

Các phương thức To có trong đối tượng Date

Bảng 13.15. Các phương thức To có trong đối tượng Date

Phương thức	Mô tả
ToGMTString	Chuyển một đối tượng ngày tháng năm qua thể hiện thành một chuỗi bằng cách dùng các quy ước định dạng dữ liệu giờ GMT (Greenwich Mean Time) của Internet.
ToLocaleString	Chuyển một đối tượng ngày tháng năm qua thể hiện thành một chuỗi bằng cách dùng các quy ước thời gian địa phương

Các phương thức Parse và UTC

Bảng 13.16. Các phương thức Parse và UTC trong đối tượng Date

Phương thức	Mô tả
Date.parse(chuỗi ngày tháng năm)	Số triệu giây trong một chuỗi ngày tháng năm từ ngày 1 tháng 1 năm 1970
Date.UTC(năm, tháng, ngày, giờ, phút, giây)	Số triệu giây trong đối tượng kể từ ngày 1 tháng 1 năm 1970

Sau đây là một số ví dụ sử dụng các phương thức trong đối tượng *Date*. Việc sử dụng các phương thức này cho chúng ta xử lý dễ dàng các sự kiện về ngày tháng.

Ví dụ 13.4

```
<HTML>
<HEAD>
  <TITLE>Phuong thuc trong lop Date</TITLE>
  <script type = "text/javascript">
    <!--
    var current = new Date();
    document.writeln( "<H1>Xau mieu ta va gia tri</H1>" );
    document.writeln( "toString: " + current.toString() +
      "<BR />toLocaleString: " + current.toLocaleString() +
      "<BR />toUTCString: " + current.toUTCString() +
```

Chương 13

Các đối tượng JavaScript

```
"<BR />valueOf: " + current.valueOf() );
document.writeln("<H1>Nhưng phương thức Get</H1>" );
document.writeln( "getDate: " + current.getDate() +
    "<BR />getDay: " + current.getDay() +
    "<BR />getMonth: " + current.getMonth() +
    "<BR />getFullYear: " + current.getFullYear() +
    "<BR />getTime: " + current.getTime() +
    "<BR />getHours: " + current.getHours() +
    "<BR />getMinutes: " + current.getMinutes() +
    "<BR />getSeconds: " + current.getSeconds() +
    "<BR />getMilliseconds: " +
    current.getMilliseconds() +
    "<BR />getTimezoneOffset: " +
    current.getTimezoneOffset() );

document.writeln("
    <H1>Chi ra thông số cho một đối tượng date mới</H1>" );
var anotherDate = new Date( 2001, 2, 18, 1, 5, 0, 0 );
document.writeln( "Date: " + anotherDate );
document.writeln(
    "<H1>Nhưng phương thức set</H1>" );
anotherDate.setDate( 31 );
anotherDate.setMonth( 11 );
anotherDate.setFullYear( 2001 );
anotherDate.setHours( 23 );
anotherDate.setMinutes( 59 );
anotherDate.setSeconds( 59 );
document.writeln( "Thay đổi date: " + anotherDate );
// -->
```

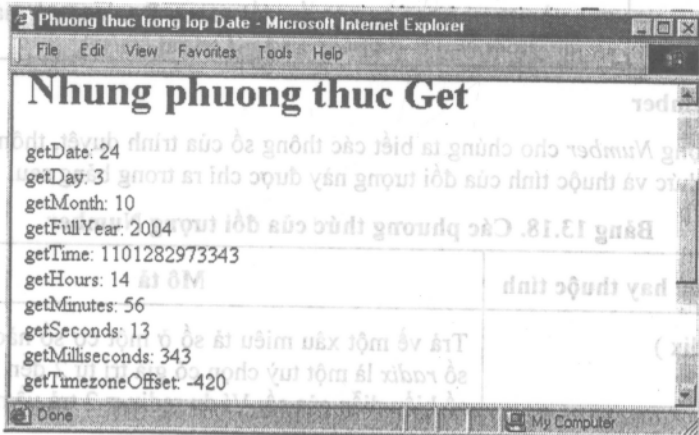

Chương 13

Các đối tượng JavaScript

```
</script>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Trong ví dụ trên, chúng ta sử dụng phương thức để lấy thời gian và ngày bằng việc gọi những phương thức *Get*. Chúng ta cũng có thể cài đặt lại thời gian cho một đối tượng *Date* bằng cách sử dụng những phương thức *Set* có trong đối tượng *Date*.

Kết quả của ví dụ trên trong trình duyệt IE như sau:



Chương 13

Các đối tượng JavaScript



Hình 13.6. Sử dụng đối tượng Date.

13.4.4. Đối tượng Boolean và số (Number)

Đối tượng Boolean

Cũng như các ngôn ngữ khác, JavaScript cho chúng ta một lớp đối tượng *Boolean*. Boolean trả về hai giá trị đúng (*true*) hoặc sai (*false*).

Các phương thức có trong đối tượng này mô tả trong bảng sau.

Bảng 13.17. Các phương thức của đối tượng Boolean

Phương thức	Mô tả
toString ()	Trả về một chuỗi "đúng" nếu giá trị của đối tượng Boolean là đúng (<i>true</i>), trong các trường hợp khác trả về giá trị là sai (<i>false</i>)
ValueOf ()	Trả về giá trị đúng (<i>true</i>) nếu đối tượng Boolean có giá trị đúng, trong các trường hợp khác trả về giá trị sai (<i>false</i>).

Đối tượng Number

Đối tượng *Number* cho chúng ta biết các thông số của trình duyệt, thông số của máy tính. Phương thức và thuộc tính của đối tượng này được chỉ ra trong bảng sau.

Bảng 13.18. Các phương thức của đối tượng Number

Phương thức hay thuộc tính	Mô tả
toString (radix)	Trả về một chuỗi miêu tả số ở một cơ số nào đó. Thông số <i>radix</i> là một tùy chọn có giá trị từ 2 đến 36 chỉ ra cơ số biểu diễn của số. Ví dụ radix = 2 trả về nhị phân của số đó.

Chương 13

Các đối tượng JavaScript

valueOf()	Trả về giá trị là số.
Number.MAX_VALUE	Thuộc tính này miêu tả giá trị lớn nhất có thể lưu trữ trong chương trình JavaScript. Giá trị này gần bằng 1.79E+308
Number.MIN_VALUE	Thuộc tính này miêu tả giá trị nhỏ nhất có thể lưu trữ trong chương trình JavaScript. Giá trị này gần bằng: 2.22E-308
Number.NaN	Thuộc tính này miêu tả một giá trị không phải là một số. Chẳng hạn: parseInt(" Hello"): không thể chuyển đổi xâu "Hello" sang một số, do đó parseInt sẽ trả về Number.NaN
Number.NEGATIVE_INFINITY	Thuộc tính này miêu tả một giá trị bé hơn giá trị Number.MIN_VALUE (Không có giá trị nào nhỏ hơn giá trị này).
Number.POSITIVE_INFINITY	Thuộc tính này miêu tả một giá trị lớn hơn giá trị Number.MAX_VALUE. (Không có giá trị nào lớn hơn giá trị này).

13.4.5. Đối tượng Document

Đối tượng *Document* thể hiện tài liệu HTML trong một cửa sổ trình duyệt và được sử dụng để lấy thông tin về tài liệu, kiểm tra, sửa đổi các phần tử HTML và văn bản trong tài liệu cũng như để xử lý các sự kiện,... Đây là đối tượng hồ sơ của cửa sổ, chứa tất cả mọi phần tử HTML của trang web trong cửa sổ đó.

Bảng 13.19. Các phương thức của đối tượng Document

Phương thức/ thuộc tính	Mô tả
Write (string)	Viết ra một chuỗi ký tự trên trang web.
Writeln (string)	Viết ra một chuỗi ký tự trên trang web. Điểm khác biệt duy nhất là <i>writeln()</i> sẽ thêm một dấu ngắt dòng ở cuối nội dung mà bạn viết.
Document.cookie	Thuộc tính này chứa tất cả các giá trị của tất cả các <i>cookie</i> được lưu giữ trong máy tính của người dùng.

Chương 13

Các đối tượng JavaScript

Clear ()	Đây là phương thức được sử dụng để xóa một văn bản.
Form []	Là mảng lưu tất cả mọi biểu mẫu của một <i>Document</i> .

13.4.6. Đối tượng Window

Đối tượng *Window* cung cấp cho chúng ta các phương thức để thao tác trên trình duyệt. Nó được dùng để hiển thị đối tượng *document* và truy nhập các sự kiện xuất hiện trong cửa sổ. Nó cũng được dùng để truy nhập tới các đặc điểm của trình duyệt có ảnh hưởng tới cửa sổ.

Trước khi đi vào tìm hiểu các phương thức của lớp này, chúng ta sẽ tham khảo ví dụ sau.

Ví dụ 13.5

Ví dụ này miêu tả các thành phần thường có của lớp *window*.

```
<HTML>
<HEAD>
<TITLE>Su dung doi tuong Window</TITLE>
<script language = "JavaScript">
  <!--
  var childWindow; // Bien dieu khiên mot cua so con
  function tao_cua_so_con()
  {
    var toolBar ; var menuBar;
    var location; var scrollBars; var status; var resizable;
    if ( toolBarCheckBox.checked )
      toolBar = "yes";
    else
      toolBar = "no";
    if ( menuBarCheckBox.checked )
      menuBar = "yes";
    else
```

Chương 13

Các đối tượng JavaScript

```
    menuBar = "no";
    if ( locationCheckBox.checked )
        location = "yes";
    else
        location = "no";
    if ( scrollBarsCheckBox.checked )
        scrollBars = "yes";
    else
        scrollBars = "no";
    if ( statusCheckBox.checked )
        status = "yes";
    else
        status = "no";
    if ( resizableCheckBox.checked )
        resizable = "yes";
    else
        resizable = "no";
    childWindow = window.open( "", "", "resizable = " + resizable +
        ",toolbar = " + toolBar + ",menubar = " + menuBar +
        ",status = " + status + ",location = " + location +
        ",scrollbars = " + scrollBars );
    closeButton.disabled = false;
    modifyButton.disabled = false;
    getURLButton.disabled = false;
    setURLButton.disabled = false;
}
function modifyChildWindow()
```

Chương 13

Các đối tượng JavaScript

```
{
    if ( childWindow.closed )
        alert( "Ban phai dong cua so" );
    else
        childWindow.document.write( textForChild.value );
}
function closeChildWindow()
{
    if ( childWindow.closed )
        alert( "Dong cua so??" );
    else
        childWindow.close();
    closeButton.disabled = true;
    modifyButton.disabled = true;
    getURLButton.disabled = true;
    setURLButton.disabled = true;
}
function getChildWindowURL()
{
    if ( childWindow.closed )
        alert( "Dong cua so" );
    else
        myChildURL.value = childWindow.location;
}
function setChildWindowURL()
{
    if ( childWindow.closed )
```

Chương 13

Các đối tượng JavaScript

```
        alert( "Dong cua so" );
    else
        childWindow.location = myChildURL.value;
    }
    //-->
</script>
</HEAD>
<BODY>
<H1>Xin chao ,Day la khung chinh</H1>
<P>Xin moi check vao nhung khung con<BR/>
    <input id = "toolBarCheckBox" type = "checkbox" value = ""
        checked = "checked" />
        <LABEL>Tool Bar</LABEL>
    <input id = "menuBarCheckBox" type = "checkbox" value = ""
        checked = "checked" />
        <LABEL>Menu Bar</LABEL>
    <input id = "locationCheckBox" type = "checkbox" value = ""
        checked = "checked" />
        <LABEL>Address Bar</LABEL><BR/>
    <input id = "scrollBarsCheckBox" type = "checkbox" value = ""
        checked = "checked" />
        <LABEL>Scroil Bars</LABEL>
    <input id = "statusCheckBox" type = "checkbox" value = ""
        checked = "checked" />
        <LABEL>Status Bar</LABEL>
    <input id = "resizableCheckBox" type = "checkbox" value = ""
        checked = "checked" />
```

Chương 13

Các đối tượng JavaScript

```
<LABEL>Resizable</LABEL><BR></P>
<P>Danh dong chu ma ban muon hien thi
trong khung con<BR>
<input id = "textForChild" type = "text"
    value = "Xin chao,toi la mot khung con</H1> <BR>">
<input id = "createButton" type = "button"
    value = "Create Child Window" onclick = "tao_cua_so_con()" />
<input id = "modifyButton" type = "button" value = "Modify Child Window"
    onclick = "modifyChildWindow()" disabled = "disabled"/>
<input id = "closeButton" type = "button" value = "Close Child Window"
    onclick = "closeChildWindow()" disabled = "disabled"/></P>
<P>The other window's URL is: <BR>
<input id = "myChildURL" type = "text" value = "/">
<input id = "setURLButton" type = "button" value = "Set Child URL"
    onclick = "setChildWindowURL()" disabled = "disabled"/>
<input id = "getURLButton" type = "button" value = "Get URL From Child"
    onclick = "getChildWindowURL()" disabled = "disabled"/></P>
</BODY>
</HTML>
```

Kết quả của ví dụ 13.5 hiển thị trên trình duyệt như hình 13.7.

Qua ví dụ trên chúng ta đã thấy được cách tổ chức các thành phần trên một trang web. Đó là tập hợp một *Frame* chính và trên nó chứa các *Frame* con. Các thao tác đều gắn với *Frame*. Khi chúng ta bấm vào nút *Create Child Window* thì sẽ xuất hiện một khung mới ở trên trình duyệt

Chương 13

Các đối tượng JavaScript



Hình 13.7. Sử dụng đối tượng Window.

Các phương thức quan trọng của đối tượng Window

Các phương thức quan trọng của đối tượng *Window* được mô tả trong bảng sau.

Bảng 13.20. Các phương thức của đối tượng Window

Phương thức hoặc thuộc tính	Mô tả
Open (url, name, option)	Tạo một cửa sổ mới với đường dẫn URL của cửa sổ là <i>url</i> , đặt tên là <i>name</i> , và một danh sách đề cập đến những thuộc tính mà cửa sổ đó có (<i>option</i>), được phân định bằng những dấu phẩy
Prompt (prompt, default)	Hiển thị một hộp thoại cho người sử dụng nhập vào và giá trị mặc định được hiển thị là giá trị <i>default</i> trong tham số.
Close ()	Đóng cửa sổ hiện tại và xóa các đối tượng của nó trong bộ nhớ.
Window.focus ()	Phương thức này quy định cách hiển thị của cửa sổ trong trình duyệt.
Window.document	Thuộc tính này chứa những đối tượng <i>document</i> của cửa sổ đang xét.

Chương 13

Các đối tượng JavaScript

Window.close	Thuộc tính này trả về giá trị kiểu <i>boolean</i> : trả về giá trị đúng (<i>true</i>) nếu cửa sổ được đóng, và giá trị sai (<i>false</i>) nếu như ngược lại.
Window.opener	Thuộc tính chứa đựng đối tượng <i>window</i> của cửa sổ đang được mở.
Window.name	Chứa tên của khung cửa sổ. Tập khung sẽ cho bạn biết tên của một khung trong đó. Tên của cửa sổ chính là tham số thứ hai trong phương thức <i>window.open()</i> .
Window.history	Là đối tượng <i>history</i> của một cửa sổ.
Window.location	Là đối tượng <i>location</i> của một cửa sổ.
Window.frame []	Là mảng của những khung có trong một cửa sổ. Mỗi khung được xem như một đối tượng <i>window</i> khác.
Window.onerror	Là tên của một hàm sẽ được kích hoạt khi có lỗi trong JavaScript. Hàm này nhận vào 3 tham số: thông báo lỗi, URL của document xuất hiện lỗi trong đó và dòng lỗi.
Window.parent	Là tham chiếu ngược trở lại cửa sổ cha của khung hiển thị hiện tại.
Window.status	Là nội dung thanh trạng thái của cửa sổ.
Window.top	Là cửa sổ trên cùng nhất trong hệ thống phân cấp cửa sổ. Thuộc tính này rất hữu ích khi JavaScript của bạn nằm trong một khung được lồng rất sâu và bạn muốn nó ảnh hưởng đến toàn bộ trang web.

13.5. Sử dụng Cookie

13.5.1. Cookie là gì?

Cookie là những mẫu thông tin mà một site để lại trên đĩa cứng của người sử dụng. Vì thông tin đó được lưu trên đĩa cứng nên nó sẽ tồn tại sau khi người sử dụng rời trang web và thậm chí sau khi họ tắt máy tính của mình. Bạn sẽ nhận thấy đặc điểm này rất hữu ích khi bạn muốn biết thông tin về người sử dụng mỗi khi họ truy cập web site của bạn.

Cookie được lưu giữ trên máy tính của bạn ở file *cookies.txt*. Bạn sẽ thấy một danh sách về tên site và mỗi tên đi kèm với một chuỗi văn bản. Có thể bạn sẽ cảm thấy văn bản

Chương 13

Các đối tượng JavaScript

rất khó hiểu vì phần lớn những site đã mã hóa *cookie* theo nhiều cách khác nhau nhằm tập hợp thông tin vào trong đó. Nhưng chúng ta nên xem qua để biết ai đã để lại thông tin này trên máy của bạn và bạn sẽ nhận ra rằng đây là điều mình nên biết.

Khi sử dụng *cookies* bạn nên quan tâm đến những chú ý sau:

- *Cookie* quan tâm nhiều vấn đề an toàn bảo mật.
- Không phải người sử dụng nào cũng có trình duyệt có khả năng hỗ trợ *Cookie*.
- Mỗi tên miền chỉ có thể có 20 *cookie* (do đó phải sử dụng một cách hợp lý nhất).
- *Cookie* không thể quá 4 Kb (tương đương với hơn 4000 ký tự).
- Một website chỉ có thể thiết lập và đọc *cookie* của chính nó.

13.5.2. Làm việc với Cookies

Trong phần này chúng ta sẽ tìm hiểu về mọi hoạt động cơ bản của *cookie* như: thiết lập, đọc, tái thiết lập và thiết lập ngày tháng kết thúc.

Thiết lập cookie

Việc thiết lập *cookie* rất cơ bản và đơn giản: bạn chỉ việc tạo một chuỗi trong biểu mẫu là: *cookie_name* = *value* và sau đó thiết lập thuộc tính *document.cookie* cho chuỗi đó. Điều duy nhất lưu ý đó là giá trị của *cookie* không được bao gồm khoảng trắng, dấu phẩy hoặc dấu chấm phẩy. Tuy nhiên, bạn không cần phải bận tâm về điều này vì hàm *escape()* và *unescape()* sẽ mã hóa và giải mã *cookie* cho bạn.

Ví dụ sau miêu tả đơn giản là lưu tên của người sử dụng trong *cookie* có tên là *username*.

Ví dụ 13.6

```
Function setCookie ( )
{
    1  var the_name = prompt( " What is your name " );
    2  var the_cookie = "username = "+ escape(the_name);
    3  document.cookie = the_cookie;
    Alert ("Thanks!");
}
```

Trong ví dụ này:

- Dòng 1: yêu cầu nhập tên của một người sử dụng.

Chương 13

Các đối tượng JavaScript

- Dòng 2: Tạo một chuỗi để lưu trong *cookie*. Hàm *escape(the_name)* nhằm biến đổi xâu đã nhập cho phù hợp với *cookie*.
- Dòng 3: sẽ thiết lập một *cookie*.

Đọc cookie

Rất dễ dàng để đọc *cookie* mà bạn đã lưu vào đĩa cứng của ai đó. Ví dụ sau cho bạn thấy có thể đọc mã *cookie* được thiết lập trong ví dụ 13.6

Ví dụ 13.7

```
Function readCookie( )
{
1  var the_cookie = document.cookie;
2  var broken_cookie = the_cookie.split("=");
3  var the_name = broken_cookie[1];
4  var the_name = unescape(the_name);
   alert("Ten cua ban la:" + the_name);
}
```

Ví dụ trên cho phép chúng ta đọc một *cookie* do chúng ta tạo ra trong ví dụ trước.

- Dòng 1: là dòng quan trọng, mỗi khi trình duyệt mở trang web, nó sẽ đọc bất kỳ *cookie* nào mà site đã lưu trên máy của bạn và tải chúng vào thuộc tính *document.cookie*.
- Điều rắc rối nhất khi đọc *cookie* là bạn sẽ chỉ có thông tin mà bạn cần từ nó.
- Dòng 2: sẽ chia *cookie* vừa được đọc thành một mảng gồm hai phần tử bằng cách sử dụng phương thức *split ()*. Phần tử đầu tiên trong mảng này là mọi thứ đứng trước dấu "=", Phần tử thứ hai trong mảng là mọi thứ đứng sau dấu "=".
- Dòng 3: sẽ nhận chuỗi này từ mảng và lưu nó trong *the_name*.
- Dòng 4: Giải mã *the_name* với hàm *unescape()* bằng cách đổi khoảng trắng bằng %20.

Tái thiết lập cookie

Để tái thiết lập (thay đổi giá trị) *cookie*, bạn chỉ việc thiết lập tên của nó bằng một giá trị khác. Ví dụ để biết thời điểm cuối cùng người sử dụng truy cập website của bạn, bạn chỉ cần thiết lập *cookie* có tên *date* mỗi lần người đó truy cập. Khi thiết lập cần để ý tới hai hàm:

Chương 13

Các đối tượng JavaScript

- *readCookie()*: hàm này kiểm tra liệu site đó đã thiết lập *cookie* hay chưa.
- *setCookie*: sẽ thiết lập một *cookie* mới. Mỗi khi *setCookie* được gọi nó sẽ thay *cookie* sau cùng bằng một *cookie* mới (điều này tương tự việc thiết lập biến).

Thiết lập thời gian tồn tại của *cookie*

Cho đến giờ, chúng ta đã tạo *cookie* có thể biến mất khi người sử dụng thoát khỏi trình duyệt. Đôi khi điều này mang lại kết quả tốt nhất. Khi biết rằng mỗi tên miền chỉ giới hạn 20 *cookie* trên máy tính người sử dụng, bạn sẽ không muốn lãng phí những khoảng trống bằng cách lưu những *cookie* không cần thiết giữa các tiến trình của trình duyệt. Tuy nhiên, nếu bạn muốn *cookie* của mình vẫn nằm trên đĩa cứng của người sử dụng sau khi người đó thoát khỏi trình duyệt thì bạn cần thiết lập thời gian kết thúc theo một định dạng đặc biệt được gọi là GMT. Ví dụ: Sun, 12 – Jan-1992 00:00:00 GMT

Định dạng GMT có thể là một công việc khó khăn. Trong JavaScript có một phương thức đó là *toGMTString* thực hiện đổi ngày tháng trong định dạng đơn giản hơn thành một định dạng GMT.

```
Var the_date = new Date("December 31,2023");
```

```
Var the_cookie = the_date.toGMTString();
```

Để thiết lập thời gian hết hạn của *cookie*, bạn cần bổ sung thời hạn kết thúc *cookie* đó. Thêm *expire = date* vào chuỗi và tách những thành phần của *cookie* ra bằng dấu chấm phẩy.

```
Cookie_name = whatever;expires = date
```

Bạn cũng có thể sử dụng thời hạn kết thúc này để loại bỏ *cookie* bằng cách thiết lập thời hạn kết thúc là một thời điểm trong quá khứ. Điều này rất cần thiết trong trường hợp bạn sử dụng *cookie* để ghi nhớ hoặc loại bỏ thông tin về người sử dụng trên site của mình. Khi người sử dụng truy cập site của bạn, hãy gửi cho họ *cookie* để họ biết những gì họ đã làm. Khi họ rời khỏi site, bạn hãy loại bỏ *cookie* đó.

Để kết thúc phần *cookie* chúng ta xem ví dụ sau.

Ví dụ 13.8

```
<HTML>
<HEAD>
  <TITLE>Su dung cookie</TITLE>
  <script language = "JavaScript">
  <!--
```

Chương 13

Các đối tượng JavaScript

```
var hien_tai = new Date(); //thoi gian va ngay hien tai
var gio = hien_tai.getHours(); // Gio o hien tai (0-23)
var name;
if ( gio < 12 ) // Neu la buoi sang
    document.write( "<H1>chao buoi sang, " );
else
{
    gio = gio - 12; // thoi gian tinh bang PM
    // neu la buoi chieu hoac buoi toi
    if ( gio < 6 )
        document.write( "<H1>Chao buoi chieu, " );
    else
        document.write( "<H1>Chao buoi toi, " );
}
// xem xet co cookie hay khong
if ( document.cookie )
{
    //giai ma cookie
    var myCookie = unescape( document.cookie );
    // Tach cookie thanh mot mang
    var cookieTokens = myCookie.split( "=" );
    // dat ten cho phan dau tu dau tien trong mang
    name = cookieTokens[ 1 ];
}
else
{
    //Neu khong co cookie thi yeu cau nguoi dung nhap ten
```

Chương 13

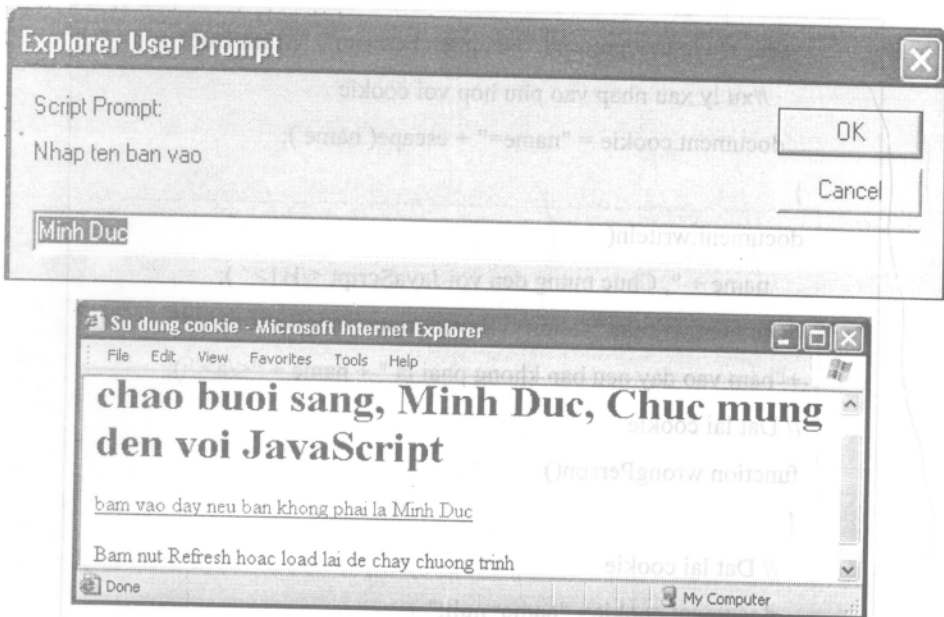
Các đối tượng JavaScript

```
name = window.prompt( "Nhap ten ban vao", "Minh Duc" );
    //xu ly xau nhap vao phu hop voi cookie
    document.cookie = "name=" + escape( name );
}
document.writeln(
    name + ", Chuc mung den voi JavaScript </H1>" );
document.writeln("<ahref=\\\"JavaScript:wrongPerson()\\\">"
+"bam vao day neu ban khong phai la " + name + "</a>" );
// Dat lai cookie
function wrongPerson()
{
    // Dat lai cookie
    document.cookie= "name=null;" +
        " expires=Thu, 01-Jan-95 00:00:01 GMT";
    // goi lai cookie moi thiet lap
    location.reload();
}
// -->
</script>
</HEAD>
<BODY>
    <P>Bam nut Refresh hoac load lai de chay chuong trinh</P>
</BODY>
</HTML>
```

Kết quả của chương trình:

Chương 13

Các đối tượng JavaScript



Hình 13.8. Sử dụng cookies.

Cookie là đặc điểm JavaScript nâng cao và có khả năng bổ sung nhiều chức năng cho web site của bạn. Trong phần này chúng ta đã nghiên cứu về cú pháp và chức năng của JavaScript, qua đó thấy được sự linh hoạt của ngôn ngữ này. JavaScript không thể thiếu được trong bất cứ trang web nào. Nắm bắt rõ về cách thức lập trình trong JavaScript sẽ cho chúng ta tiếp cận với thế giới website được dễ dàng hơn. Trước khi kết thúc chương này, bạn nên tham khảo ví dụ sau. Ví dụ này kết hợp tất cả các khái niệm cơ bản trong JavaScript, để một lần nữa giúp bạn nắm bắt được vấn đề.

Ví dụ 13.9

Ví dụ tổng quát

```
<HTML>
<HEAD>
  <TITLE>Tat ca trong mot trang web</TITLE>
  <script type = "text/javascript">
    <!--
    var now = new Date(); // thời gian hiện tại
```


Chương 13

Các đối tượng JavaScript

```
var hour = now.getHours(); // gio hien tai
//Mang luu giu ten cac anh
var pictures =
[ "CPE", "EPT", "GPP", "GUI", "PERF", "PORT", "SEO" ];
//Mang luu giu cac cau trích dẫn
var quotes = [ "Chúng ta đang học javascript<BR/>" +
"JavaScript thật hay", "Các phương thức và thuộc tính." +
" (Phải xác định được.) <BR/> Có bao nhiêu được không", "Đó là một" +
" website có bị virus tấn công không?<BR/> Phải kiểm soát được" ];
// Viet thoi gian hien tai len trang web
document.write( "<P>" + now.toLocaleString() + "<BR/></P>" );
//Neu la buoi sang
if ( hour < 12 )
    document.write( "<h2>Chao buoi sang, " );
else
{
    hour = hour - 12;
    // La buoi chieu hay buoi toi
    if ( hour < 6 )
        document.write( "<h2>Chao buoi sang, " );
    else
        document.write( "<h2>Chao buoi toi, " );
}
// xem mot cookie co ton tai khong
if ( document.cookie )
{
    var myCookie = unescape( document.cookie );
```

Chương 13

Các đối tượng JavaScript

```
var cookieTokens = myCookie.split( "=" );
name = cookieTokens[ 1 ];
}
else
{
    name = window.prompt( "Please enter your name", "GalAnt" );
    document.cookie = "name=" + escape( name );
}
// Viet loi chao
document.writeln(
    name + ", Chao mung ban den voi JavaScript!</h2>" );
// viet mot link de xoa mot cooke
document.writeln( "<a href = \" JavaScript:wrongPerson() \" > " +
"Bam vao nut nay neu ban khong phai la " + name + "</a><BR/>" );
// tao mot anh ngau nhien
document.write ( "<img src = \"\" +
    pictures[ Math.floor( Math.random() * 7 ) ] +
    ".gif\" width= \" 105 \" height= \" 100 \" /> <BR/>" );
// Viet moi loi trích ngẫu nhiên
document.write ( quotes[ Math.floor( Math.random() * 3 ) ] );
// Tao mot cua so voi tat ca cac loi trích
function allQuotes()
{
    // tao mot cua so con cho loi trích đản
quoteWindow = window.open( "", "", "resizable=yes, toolbar" +
    "=no, menubar=no, status=no, location=no," +
    " scrollBars=yes" );
```

Chương 13

Các đối tượng JavaScript

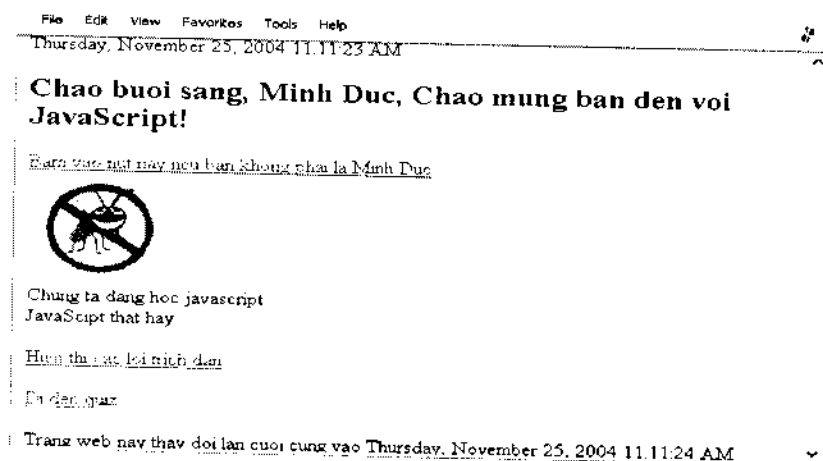
```
quoteWindow.document.write( "<P>" )
// Vong lap de ghi cac loi trích dẫn vào của số mới
for ( var i = 0; i < quotes.length; i++ )
    quoteWindow.document.write( ( i + 1 ) + ". " +
        quotes[ i ] + "<BR/><BR/>");
// link dòng cho của số mới này
quoteWindow.document.write( "</P><BR/><a href = \" " +
    "JavaScript:window.close()\">" +
    " Dòng của số này</a>" )
}
// Dặt lại cookie
function wrongPerson()
{
    // dặt lại cookie
    document.cookie= "name=null;" +
        " expires=Thu, 01-Jan-95 00:00:01 GMT";
    location.reload();
}
// mở một cửa sổ chứa trang quiz2.HTML
function openQuiz()
{
    window.open( "quiz2.HTML", "", "resizable = yes, " +
        "toolbar = no, menubar = no, status = no, " +
        "location = no, scrollBars = no");
}
// -->
</script>
```

Chương 13

Các đối tượng JavaScript

```
</HEAD>
<BODY>
<P><a href = "JavaScript:allQuotes()">Hiện thi cac loi trích dẫn</a></P>
  <P id = "quizSpot">
    <a href = "JavaScript:openQuiz()">Đi đến quiz</a></P>
  <script type = "text/javascript">
    //Bien chua thoi gian cho lan sua doi cuoi cung
    var modDate = new Date( document.lastModified );
    // Viet thoi gian do ra trang web
    document.write ( "Trang web nay thay doi lan cuoi cung vao " +
      modDate.toLocaleString() );
  </script>
</BODY>
</HTML>
```

Kết quả của ví dụ tổng quát này hiển thị trên trình duyệt như sau:

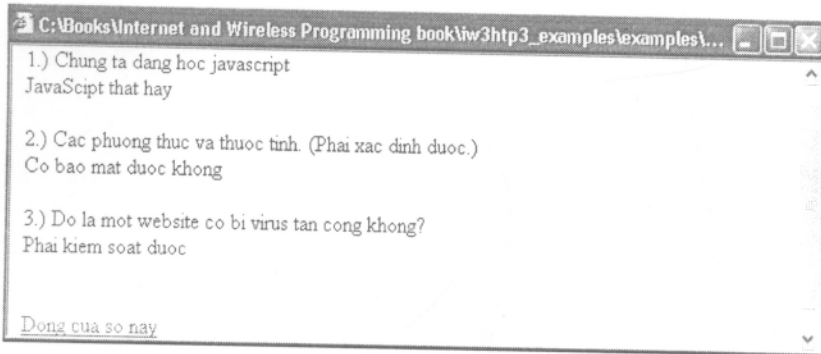


Hình 13.9. Kết quả của ví dụ tổng quát trên trình duyệt.

- Khi ta nhấp chuột vào liên kết : *Hiện thi cac loi trích dẫn* thì sẽ xuất hiện ra một khung của sổ chứa tất cả các lời trích dẫn.

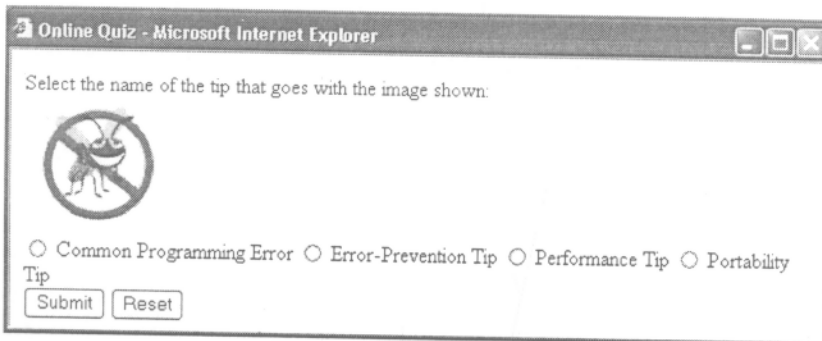
Chương 13

Các đối tượng JavaScript



Hình 13.10. Khung của sổ chứa các lời trích dẫn.

- Khi bạn nhấp vào link: *Dong cua so nay* thì cửa sổ này sẽ đóng.
- Khi bạn nhấp vào link: *Diden quiz*. (quiz là một trang web nằm cùng thư mục với trang web của mình) thì sẽ trang web này sẽ xuất hiện.



Hình 13.11. Trang web quiz.

13.6. Kết chương

Trong chương này bạn đã tìm hiểu về các đối tượng khác nhau mà bạn có thể chèn vào một tài liệu HTML bao gồm: các đối tượng của trình duyệt, các đối tượng JavaScript, cookies. Bạn có thể tham khảo thêm các tài liệu về JavaScript trên các website sau:

www.javascript.com

www.iboots.com/build/programing/js/tutorial/885.htm

www.javascriptsearch.com

www.aljavascript.com

PHẦN 5

DHTML

NỘI DUNG

Chương 14 - Giới thiệu về DHTML

Chương 15 - Mô hình sự kiện DHTML

Chương 16 - Các hiệu ứng đặc biệt

Chương 17 - Liên kết dữ liệu

Chương 14

Giới thiệu về DHTML

Nội dung của chương:

- Giới thiệu về HTML động (Dynamic HTML).
- Các đặc điểm của DHTML.
- Mô tả mô hình đối tượng DHTML.

14.1. Giới thiệu

Khi bắt đầu ra đời HTML được phát triển như một định dạng tài liệu chung dùng để trao đổi thông tin trên Internet. Việc chuyển giao dữ liệu độc lập với hệ thống nền. Tuy nhiên với sự phát triển mạnh mẽ như hiện nay, Internet đã trở thành một nguồn thông tin và giải trí lớn. Các trang web trở nên hấp dẫn và nhiều màu sắc hơn để thu hút người sử dụng. Mặc dầu vậy các style cơ bản và nội dung của trang web vẫn cố định. Những người phát triển web có rất ít hoặc hầu như không có sự kiểm soát nào đối với một trang web sau khi trang web đó đã được hiển thị trong trình duyệt. Nghĩa là HTML vẫn còn "tĩnh".

Sự ra đời của việc lập trình bằng *script* đã thêm vào tính năng động cho các trang web. Người dùng đã có thể tương tác với các trang web, tuy nhiên vẫn còn rất hạn chế. Mọi thao tác xử lý dữ liệu đều được thực hiện tại máy chủ. Để thay đổi nội dung style của một trang, thì trang đó phải được viết mới hoặc thay thế hoàn toàn. Bất kỳ tương tác nào của người dùng cũng vẫn phải đi qua máy chủ.

Để khắc phục các hạn chế đó, mỗi phiên bản trình duyệt mới ra đời lại được xây dựng thêm các đặc tính tốt hơn cho HTML. Đến nay hai trình duyệt nổi tiếng và thông dụng nhất là Internet Explorer và Netscape Navigator đều hỗ trợ một mô hình mà ở đó các phần tử HTML và các thẻ được xem như là các đối tượng. Các đối tượng có các phương thức, thuộc tính và sự kiện, chúng có thể được lập trình bằng script để điều khiển hành vi của đối tượng và được thực hiện trong trình duyệt. Điều này có nghĩa là dữ liệu có thể được thao

Chương 14

Giới thiệu về DHTML

tác, định dạng và thay đổi một cách chủ động tại máy khách mà không cần quá nhiều hỗ trợ từ máy chủ. Tương tác của người dùng giờ đây có thể được xử lý bởi chính máy khách.

Ví dụ, bạn có thể thêm vào đoạn mã để thay đổi màu của văn bản khi người dùng nhấp chuột lên nó trong thẻ xuống dòng <P> của HTML như trong ví dụ 14.1.

Ví dụ 14.1.

```
<HTML>
<HEAD>
<TITLE>Trang Web động</TITLE>
</HEAD>
<P onclick = "style.fonsize= '25'; style.color= 'red'">Click chuột vào dòng này
</HTML>
```

Lúc này HTML không còn “tĩnh”, mà đã trở thành HTML “động” và được gọi là *Dynamic HTML (DHTML)*. Vậy một cách khái quát hơn DHTML là gì?

DHTML hay HTML động có thể được mô tả như là một sự kết hợp của HTML, các *style sheet* và *script* trong một trang Web, giúp cho việc xây dựng và các thao tác trên trang Web sinh động, dễ dàng và hiệu quả hơn.

Microsoft và Netscape có cách triển khai HTML động riêng của mình. Microsoft tập trung vào dùng các *Cascading Style Sheets (CSS)*. Bạn có thể dùng script để xử lý các phần tử của CSS. Ngược lại, Netscape dựa vào phương thức dùng các lớp. Trong đó thẻ <LAYER> được dùng để cung cấp hầu hết các đặc điểm của HTML động.

14.2. Các đặc điểm của DHTML

HTML động không dừng lại ở một số tính năng mở rộng của HTML. Trong phần này bạn sẽ học về các đặc điểm của HTML động và cách thức dùng nó để thêm các đặc tính động cho một trang web.

14.2.1. Cấu trúc đối tượng

HTML động theo một cấu trúc đối tượng, theo đó mỗi phần tử được coi như là một đối tượng trong cấu trúc. Mỗi đối tượng có thể được truy cập và lập trình độc lập. Cách đơn giản nhất để tham chiếu đến một đối tượng là sử dụng thuộc tính ID của đối tượng đó. Mô hình đối tượng DHTML cho phép chúng ta thay đổi lớp thuộc tính của đối tượng và lập trình một cách dễ dàng hơn.

Chương 14

Giới thiệu về DHTML

Ví dụ 14.2.

```
<HTML>
<HEAD>
  <title>Mô hình đối tượng</title>
  <script language = "JavaScript">
    <!--
      function Bat_dau()
      {
        alert( object1.innerText );
        object1.innerText = "Cam on ban!!!.";
      }
    // -->
  </script>
</HEAD>
<body onload = "Bat_dau()">
  <p id = "object1">Chào mừng đến với trang web của chúng tôi!</p>
</body>
</HTML>
```

Trong ví dụ trên ta xây dựng một đối tượng là *object1* với *id = object1*.

- Hàm *Bat_dau()* hiển thị hộp thoại với giá trị của *object1*. Để lấy giá trị của *object1* sử dụng thuộc tính *innerText* của đối tượng theo cú pháp: *object1.innerText*.
- Ban đầu thuộc tính này sẽ tham chiếu đến dòng chữ "*chào mừng đến với trang web của chúng tôi!*". Khi người dùng click vào hộp thoại thì hàm *Bat_dau()* được gọi và thuộc tính *innerText* tham chiếu đến dòng chữ "*Cam on ban!!!*". Do đó dòng chữ "*Cam on ban!!!*" sẽ được hiển thị.

Kết quả của ví dụ được minh họa trong hình sau.

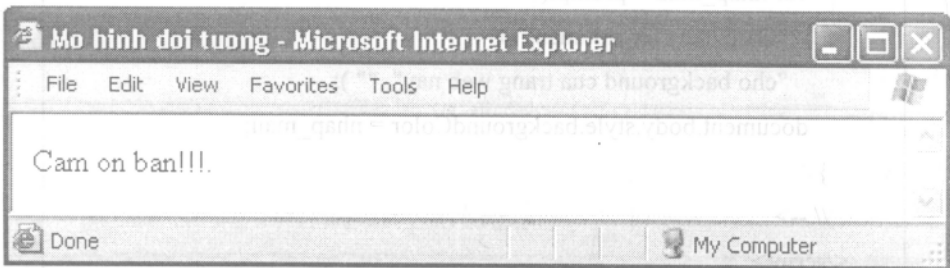
Chương 14

Giới thiệu về DHTML



Hình 14.1. Giao diện ban đầu khi hiển thị.

Sau khi người dùng click vào nút *Ok*, thì hàm *Bat_dau ()* được gọi và giá trị của *object1* được thay đổi, hộp thoại sẽ hiển thị như sau:



Hình 14.2. Giao diện khi kích vào nút OK của hộp thoại.

14.2.2. Các style động (Dynamic Styles)

Trong các phiên bản HTML trước đây, nếu bạn muốn thay đổi kiểu hay nội dung của một trang web sau khi nó được hiển thị trên trình duyệt thì toàn bộ trang đó phải được “làm tươi” (*refresh*) lại. Điều đó có nghĩa là yêu cầu phải được gửi tới máy chủ để hiển thị một trang mới. Đối với người dùng thì đây là một quá trình trong suốt. Tuy nhiên, nếu trang đó phải được làm tươi thường xuyên thì sẽ tốn nhiều thời gian và làm cho máy chủ trở nên quá tải.

Trong HTML động, tình huống tương tự sẽ không còn nữa. Bằng cách dùng các *style sheet* bạn có thể chỉ ra màu, kiểu hay kích thước của trang hiện thời. Mỗi lần thay đổi các thẻ và các thuộc tính của trang, bạn không cần phải trở về máy chủ web. Điều đó có nghĩa là bạn có thể thay đổi màu, phông, kích thước của văn bản song song với các đáp ứng lại những tương tác của người dùng. Trong HTML động, *Style* liên quan đến cách thức mọi thứ xuất hiện trên một trang hơn là về nội dung. *Style* bao gồm màu sắc, kiểu chữ, khoảng cách, sự định vị,... của văn bản khi xuất hiện trên trình duyệt.

Chương 14

Giới thiệu về DHTML

Trong ví dụ sau, chúng ta có thể thay đổi màu nền cho một trang web bằng cách sử dụng *style sheet* mà không phải quay trở về máy chủ web.

Ví dụ 14.3.

```
<HTML>
  <HEAD>
    <title>Mô hình đối tượng</title>
    <script language = "JavaScript">
      <!--
        function Bat_dau()
        {
          var nhap_mau = prompt(
            "Danh ten của một màu " +
            "cho background của trang web này", "" );
          document.body.style.backgroundColor = nhap_mau;
        }
      // -->
    </script>
  </HEAD>
  <body onload = "Bat_dau()">
    <p>Chào mừng bạn đến với trang web của chúng tôi!</p>
  </body>
</HTML>
```

- Hàm *Bat_dau ()* đưa ra một hộp thoại cho người sử dụng gõ vào tên màu muốn hiển thị trên màn hình. Sau đó thực hiện giá trị vừa nhập vào thông qua câu lệnh:

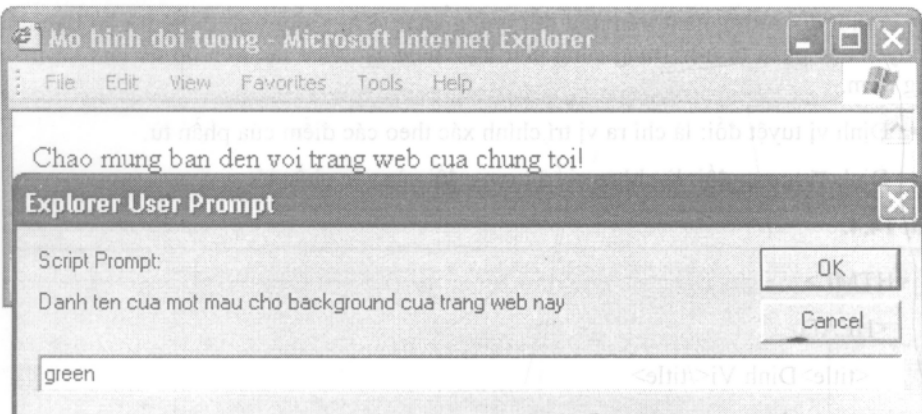
```
document.body.style.backgroundColor = nhap_mau;
```

- Sau khi thực hiện câu lệnh này, màu nền của trang web sẽ chuyển thành màu mà người sử dụng vừa nhập vào. Đây chính là điểm khác với HTML thông thường, các yêu cầu của người sử dụng sẽ được thực hiện tại trình duyệt của người sử dụng.

Kết quả của ví dụ trên được minh họa trong hình dưới đây:

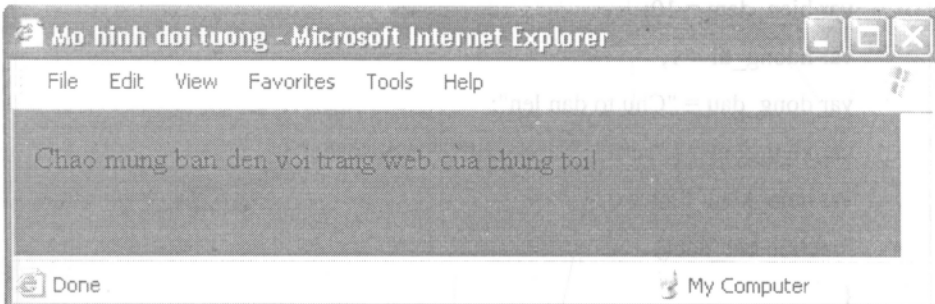
Chương 14

Giới thiệu về DHTML



Hình 14.3. Hộp thoại gõ tên màu.

Lúc đầu màu nền của trang web là màu trắng (màu mặc định). Sau khi người sử dụng nhập vào giá trị "green" thì màu nền của trang web chuyển thành màu xanh.



Hình 14.4. Chuyển màu nền.

14.2.3. Nội dung động

Tính năng này được hỗ trợ bởi Internet Explorer. Ở đây bạn có thể thay đổi chữ và hình ảnh trên một trang web sau khi nó được hiển thị. Bạn cũng có thể thay đổi nội dung của trang đáp ứng lại dữ liệu nhập vào nhập hay sự kiện của người dùng chẳng hạn như kích chuột.

14.2.4. Định vị

Các phiên bản HTML trước đây không kiểm soát được vị trí của một phần tử trên một trang web. Chúng ta chỉ có thể mô tả nội dung và vị trí tương đối của các phần tử. Trong HTML động, bạn có thể chỉ ra vị trí chính xác (tuyệt hoặc tương đối), nhờ dùng các điểm toạ độ x và y. Đặc điểm định vị cũng cho phép ta chỉ ra trật tự z-order của các phần tử,

Chương 14

Giới thiệu về DHTML

nghĩa là một đối tượng nằm trên một đối tượng khác. Khi trang web được trả lại bạn có thể di chuyển các phần tử trên trang cũng như thực hiện các thao tác trên nó để làm nó trở nên 'động' hơn.

- Định vị tuyệt đối: là chỉ ra vị trí chính xác theo các điểm của phần tử.
- Định vị tương đối: là chỉ ra vị trí tương đối của các phần tử.

Ví dụ 14.4.

```
<HTML>
<HEAD>
  <title>Định Vị</title>
  <script language = "JavaScript">
    <!--
    var toc_do = 5;
    var bien_dem = 10;
    var huong_di = 1;
    var dong_dau = "Chu to dan len";
    var kieu_font = [ "serif", "sans-serif", "monospace" ];
    var dem_kieu_font = 0;
    function bat_dau()
    {
      window.setInterval( "run()", 100 );
    }
    function run()
    {
      bien_dem += toc_do;
      if ( ( bien_dem % 200 ) == 0 ) {
        toc_do *= -1;
        huong_di = !huong_di;
        object1.style.color =
```

Chương 14

Giới thiệu về DHTML

```
( toc_do < 0 ) ? "red" : "blue" ;
dong_dau =
( toc_do < 0 ) ? "Chu be lai" : "Chu to len";
object1.style.fontFamily =
kieu_font[ ++dem_kieu_font % 3 ];
}
object1.style.fontSize = bien_dem / 3;
object1.style.left = bien_dem;
object1.innerHTML = dong_dau + "<br /> Font size: " +
bien_dem + "px";
}
// -->
</script>
</HEAD>
<body onload = "bat_dau()">
<p id = "object1" style = "position: absolute; left: 0;
font-family: serif; color: blue"> Welcome!</p>
</body>
</HTML>
```

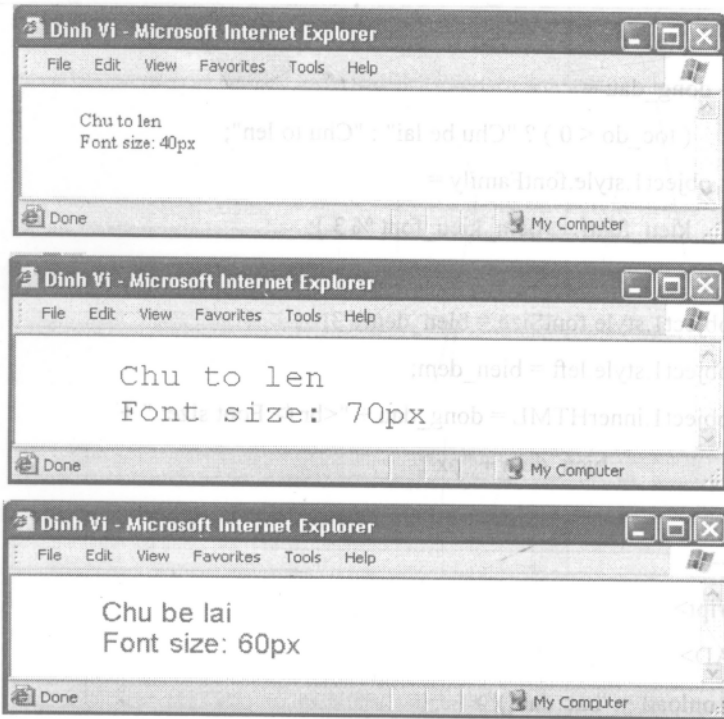
Trong ví dụ, chúng ta sử dụng Hàm *run()* dùng để tăng hay giảm kích cỡ của font chữ và xác định màu chữ là xanh hay đỏ. Vị trí xuất hiện của dòng chữ trên trang web sẽ luôn được thay đổi thông qua vị trí tuyệt đối:

```
<p id = "object1" style = "position: absolute; left: 0;
font-family: serif; color: blue"> Welcome!</p>
```

Kết quả của ví dụ này được thể hiện trong trình duyệt như sau:

Chương 14

Giới thiệu về DHTML



Hình 14.5. Tăng/giảm kích cỡ của font chữ và thay đổi màu chữ.

14.2.5. Liên kết dữ liệu

Bạn đã được giới thiệu cách dùng đối tượng bảng (*table*) trong HTML, trong HTML động bạn có thể dùng đối tượng bảng hiệu quả hơn. Được Internet Explorer hỗ trợ, chúng ta có thể liên kết bảng biểu với một trang web và một cơ sở dữ liệu. Khi trang Web hiển thị trong trình duyệt, dữ liệu từ cơ sở dữ liệu trên máy chủ sẽ được lấy về và hiển thị trong bảng biểu. Bạn cũng có thể sắp xếp và lọc dữ liệu rồi hiển thị nó cho phù hợp với yêu cầu của mình.

14.2.6. Lập trình các Script

Bạn có thể thay đổi kiểu và nội dung của trang web. Script bạn viết được chèn vào trang web. Trình duyệt thông dịch script đó mỗi khi trang đó được hiển thị.

Chương 14

Giới thiệu về DHTML

14.3. Sử dụng DHTML

Đến đây bạn đã nắm rõ hơn các đặc điểm của HTML động và sự khác biệt mà nó mang đến cho các trang web. DHTML có 3 đặc điểm sau:

- Định vị: cho phép ta chỉ ra vị trí của các phần tử HTML trong một trang theo các tọa độ x và y. Ta cũng có thể di chuyển các đối tượng sau khi trang đó được hiển thị.
- Các style: cho phép ta chỉ ra kiểu văn bản trong trang web theo màu sắc, kích thước, và kiểu chữ...
- Xử lý sự kiện: cho phép ta nắm bắt các sự kiện do người dùng hay hệ thống tạo ra và sửa đổi kiểu hay nội dung của trang web.

Chúng ta sẽ tìm hiểu cách áp dụng các đặc điểm trên của DHTML vào các trang web để làm nó trở nên “động” hơn.

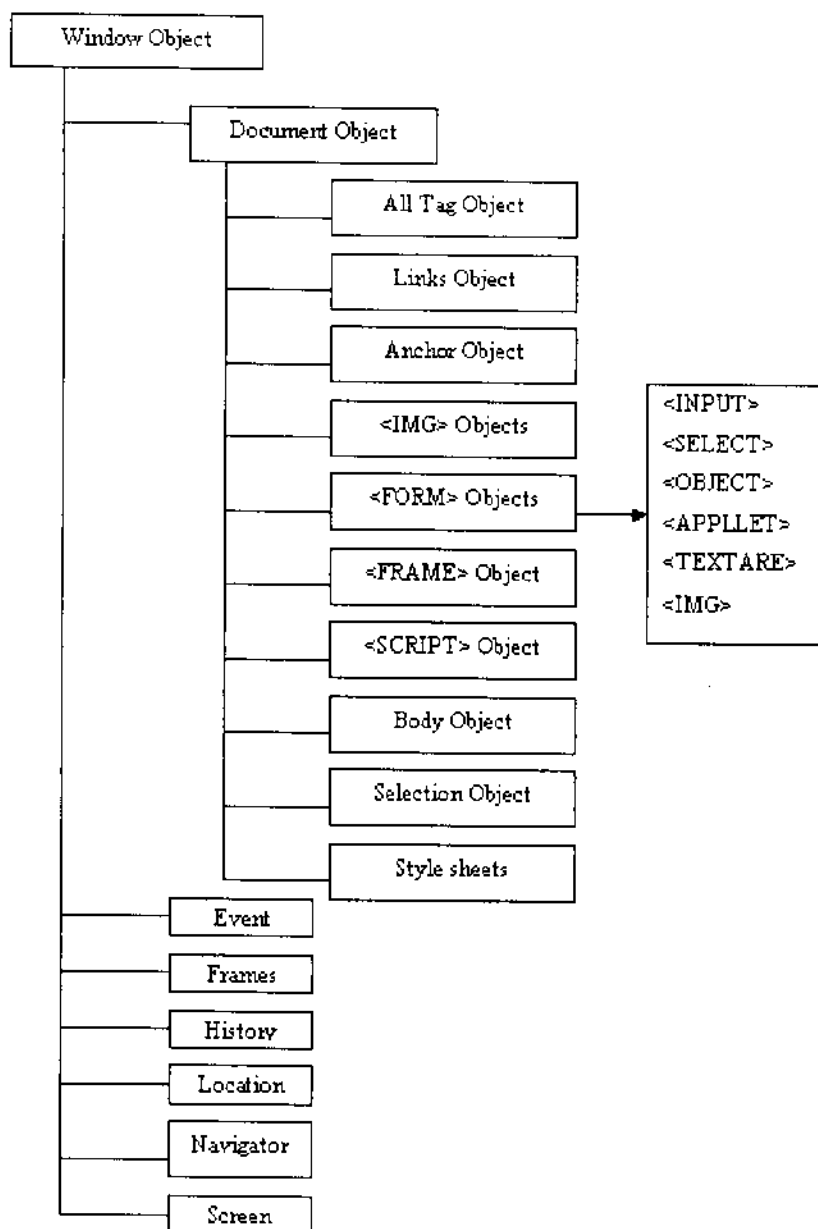
14.4. Mô hình đối tượng tài liệu

HTML động dựa trên một mô hình đối tượng ở đó mỗi phần tử được coi như một đối tượng được truy nhập và lập trình độc lập. (Bản thân HTML là không có tính động). Trong mô hình đối tượng tài liệu, các phần tử và các thẻ HTML được coi như các đối tượng. Mỗi đối tượng có tập hợp các thuộc tính, phương thức và sự kiện riêng của nó.

- Thuộc tính của một đối tượng là đặc điểm của đối tượng đó. Các thuộc tính có thể được truy cập và dùng để thao tác với các style và chức năng của đối tượng. Ví dụ, đối tượng body có thuộc tính màu nền (*bgcolor*),...
- Một phương thức là chức năng của một đối tượng. Ví dụ, đối tượng tài liệu có phương thức *write* dùng để viết một chuỗi kí tự vào nội dung tài liệu.
- Một sự kiện là một hoạt động của một đối tượng được tạo ra để đáp ứng lại tương tác của người dùng hay được tạo ra bởi hệ thống. Ví dụ, sự kiện *mouseover* (đưa chuột lên trên), sự kiện *onload* (khi trang web bắt đầu được hiển thị).

Mô hình đối tượng tài liệu được chỉ ra trong hình vẽ sau.

Chương 14
Giới thiệu về DHTML



Hình 14.6. Mô hình đối tượng tài liệu của IE.

Chương 14

Giới thiệu về DHTML

Ví dụ 14.5.

Bạn có thể chỉ ra màu nền của đối tượng *body* khi tạo trang web của bạn.

```
<HTML>
  <HEAD>
    <TITLE>Watch this space</TITLE>
  </HEAD>
  <BODY BGCOLOR=lavender TEXT=crimson>
    <P>Mau nen cua trang la lavender va mau cua chu la crison </P>
  </BODY>
</HTML>
```

Ví dụ 14.6.

Bạn có thể sử dụng phương thức *write* để hiển thị một chuỗi ký tự trong nội dung tài liệu.

```
<HTML>
  <HEAD>
    <TITLE>Using Method</TITLE>
    <SCRIPT LANGUAGE = "JavaScript">
      document.write ("Are you having fun?")
    </SCRIPT>
  </HEAD>
  <BODY>
  </BODY>
</HTML>
```

Ví dụ 14.7.

Bạn có thể sử dụng sự kiện *onload* để hiển thị hộp thoại với một câu thông báo.

Chương 14

Giới thiệu về DHTML

```
<HTML>
<HEAD>
  <TITLE>My Hopme Page</TITLE>
  <SCRIPT LANGUAGE="JavaScript">
    function home ( )
    {
      alert ("Welcome to my Home Page");
    }
  </SCRIPT>
</HEAD>
<BODY onLoad= "home()">
</BODY>
</HTML>
```

14.4.1. Đối tượng Window

Đối tượng **Window** thể hiện cửa sổ trình duyệt và được dùng để lấy thông tin về trạng thái của cửa sổ. Nó cũng được dùng để hiển thị đối tượng tài liệu và truy nhập tới các sự kiện xuất hiện trong cửa sổ.

Khi trình duyệt mở một tài liệu HTML, thông thường nó tạo ra một đối tượng *window*. Tuy nhiên nếu một tài liệu định nghĩa một hay nhiều hơn các *frame* thì trình duyệt sẽ tạo một đối tượng window cho tài liệu ban đầu và thêm một *window* nữa cho mỗi *frame*. Các đối tượng thêm vào là các cửa sổ con của cửa sổ ban đầu và có thể bị ảnh hưởng bởi các hành động xuất hiện trong cửa sổ ban đầu. Ví dụ, nếu bạn đóng cửa sổ ban đầu thì tất cả các cửa sổ con cũng sẽ bị đóng theo.

Các thuộc tính của đối tượng Window

Các thuộc tính của đối tượng *Window* được giới thiệu trong bảng sau.

Bảng 14.1. Các thuộc tính của đối tượng Window

Tên	Mô tả
document	Thể hiện tài liệu HTML trong một cửa sổ trình duyệt

Chương 14

Giới thiệu về DHTML

Event	Thể hiện trạng thái của một sự kiện, chẳng hạn trạng thái của các phím trên bàn phím, vị trí chuột và trạng thái của các nút chuột.
history	Chứa thông tin về các URL mà máy khách đã đến thăm
location	Chứa thông tin về URL hiện tại
name	Xác lập, lấy tên cửa sổ (hoặc tên khung)
navigator	Chứa thông tin về trình duyệt web
self	Lấy một tham chiếu đến cửa sổ hay khung hiện thời
status	Xác lập thông báo trong thanh trạng thái ở cuối cửa sổ
top	Lấy cửa sổ gốc, là cửa sổ cha của chính nó

Các phương thức của đối tượng Window

Các phương thức của đối tượng *Window* được giới thiệu trong bảng sau.

Bảng 14.2. Các phương thức của đối tượng Window

Tên	Mô tả
alert	Hiện thị hộp hội thoại chứa một thông báo do ứng dụng định nghĩa
blur	Làm cho một đối tượng mất tiêu điểm và bắt đầu sự kiện <i>onblur</i>
clearInterval	Hủy bỏ các ngắt quãng được thiết lập bằng phương thức <i>setInterval</i>
clearTimeout	Hủy bỏ một thời gian tạm ngưng được xác lập bằng phương thức <i>setTimeout</i>
close	Đóng cửa sổ trình duyệt hiện thời hay một ứng dụng HTML
confirm	Hiện thị một hộp hội thoại xác nhận chứa một thông báo tùy chọn cũng như các nút OK và Cancel
focus	Làm cho một lệnh điều khiển nhận tiêu điểm và thực hiện mã do sự kiện <i>onfocus</i> chỉ ra
navigate	Tải URL được chỉ ra cho cửa sổ hiện thời
open	Mở một cửa sổ mới và tải tài liệu do URL đã cho chỉ ra, hoặc mở một tài liệu trống nếu không có URL nào được cung cấp

Chương 14

Giới thiệu về DHTML

print	In tài liệu liên kết với cửa sổ
prompt	Hiển thị hộp hội thoại nhắc nhở người dùng bằng một thông báo và một ô nhập dữ liệu
setInterval	Thiết lập các ngắt quãng
setTimeout	Thiết lập thời gian tạm ngưng

14.4.2. Đối tượng Document

Đối tượng **Document** thể hiện tài liệu HTML trong một cửa sổ trình duyệt đã cho, được dùng để lấy thông tin về tài liệu đó, kiểm tra, sửa đổi, xử lý các sự kiện của các phần tử HTML và nội dung tài liệu.

Các thuộc tính của đối tượng Document

Các thuộc tính của đối tượng *Document* được giới thiệu trong bảng sau.

Bảng 14.3. Các thuộc tính của đối tượng Document

Tên	Mô tả
alink	Xác lập hay lấy màu của tất cả liên kết đang hoạt động trong tài liệu
bgColor	Xác lập hay lấy màu nền đối tượng tài liệu
body	Chỉ ra phần đầu và cuối nội dung của một tài liệu
fgColor	Xác lập hay lấy màu trên cùng của tài liệu
link	Xác lập hay lấy màu của các liên kết trong tài liệu
location	Chứa các thông tin định vị
title	Chứa tiêu đề của tài liệu
URL	Xác lập hay lấy URL cho tài liệu hiện thời
vlink	Xác lập hay lấy màu của các liên kết mà người dùng đã đến thăm

Các phương thức của đối tượng Document

Các phương thức của đối tượng *Document* được giới thiệu trong bảng sau.

Chương 14

Giới thiệu về DHTML

Bảng 14.4. Các phương thức của đối tượng Document

Tên	Mô tả
clear	Xoá tài liệu hiện thời
close	Đóng một luồng dữ liệu xuất và buộc các dữ liệu đã được gửi đi phải hiển thị
open	Mở một tài liệu để thu thập dữ liệu của phương thức <i>write / writeln</i>
write	Viết chuỗi kí tự vào một tài liệu trong cửa sổ được chỉ ra
writeln	Viết chuỗi kí tự tận cùng là một ký tự xuống dòng vào một tài liệu trong một cửa sổ được chỉ ra.

14.4.3. Phân tử và đối tượng Frame

Chỉ ra một khung với phân tử FRAMESET. Nếu một người dùng nhấp chuột hay mở một tệp tin hay thư mục trong một khung thì tệp tin hay thư mục đó chiếm toàn bộ cửa sổ. Nếu người dùng nhấp vào một tệp tin hay thư mục trong khung thứ hai thì khung đó kiểm soát toàn bộ cửa sổ gồm cả khung đầu tiên.

Các thuộc tính của đối tượng Frame

Các thuộc tính của đối tượng *Frame* được giới thiệu trong bảng sau.

Bảng 14.5. Các thuộc tính của đối tượng Frame

Tên	Mô tả
borderColor	Xác lập hay lấy màu viền của thẻ <FRAME>
frameBorder	Xác định hiển thị viền của một khung hay không
height	Xác lập hay lấy độ cao của <i>frame</i>
id	Lấy chuỗi nhận diện <i>frame</i>
marginHeight	Xác lập độ cao lề trên và dưới trước khi hiển thị văn bản trong <i>frame</i>
marginWidth	Xác lập độ rộng lề trái và phải trước khi hiển thị văn bản trong <i>frame</i>
name	Xác lập lấy tên của cửa sổ hay <i>frame</i>
scrolling	Xác lập có cho phép xuất hiện thanh cuộn hay không
src	Xác lập URL cần tải về

Chương 14

Giới thiệu về DHTML

style	Xác lập các kiểu có thể cho một phần tử
width	Xác lập hay lấy độ rộng của <i>frame</i>

Các phương thức của đối tượng Frame

Các phương thức của đối tượng *Frame* được giới thiệu trong bảng sau.

Bảng 14.6. Các phương thức của đối tượng Frame

Tên	Mô tả
blur	Làm cho một đối tượng mất tiêu điểm và bắt đầu sự kiện <i>onblur</i>
focus	Làm cho một điều khiển nhận và thực thi sự kiện <i>onfocus</i>

Để hiểu rõ hơn về khái niệm *Frame*, chúng ta cùng nghiên cứu qua ví dụ 14.8 sau:

Ví dụ 14.8.

Trong ví dụ này chúng ta sẽ chia trình duyệt thành hai *Frame* theo chiều ngang:

```
<frameset rows = "100, *">
  <frame src = "top.html" name = "upper" />
  <frame src = "" name = "lower" />
</frameset>
```

- *Frame* đầu tiên có tên là *upper* sẽ hiển thị nội dung của trang *top.htm*.
- *Frame* thứ hai có tên là *lower* được khởi tạo rỗng dùng để hiển thị lời chào với người có tên được nhập vào bởi người sử dụng.

```
<HTML>
<HEAD>
  <title>Frame</title>
</HEAD>
<frameset rows = "100, *">
  <frame src = "top.html" name = "upper" />
  <frame src = "" name = "lower" />
</frameset>
</HTML>
```

Chương 14

Giới thiệu về DHTML

Trang web *top.html* có mã như sau:

```
<HTML>
<HEAD>
  <title>Su dung Frame</title>
  <script language = "JavaScript">
    <!--
    function bat_dau()
    {
      var nhap_ten = prompt( "Ten cua ban la gi?", "" );
      parent.frames( "lower" ).document.write(
        "<h1>Xin chao, " + nhap_ten + "</h1>" );
    }
    // -->
  </script>
</head>
<body onload = "bat_dau()">
  <h1>Cross_frame!</h1>
</body>
</HTML>
```

Quan sát trong đoạn mã trên chúng ta xây dựng các hàm:

- Hàm *bat_dau()*, dùng để nhập tên người sử dụng.

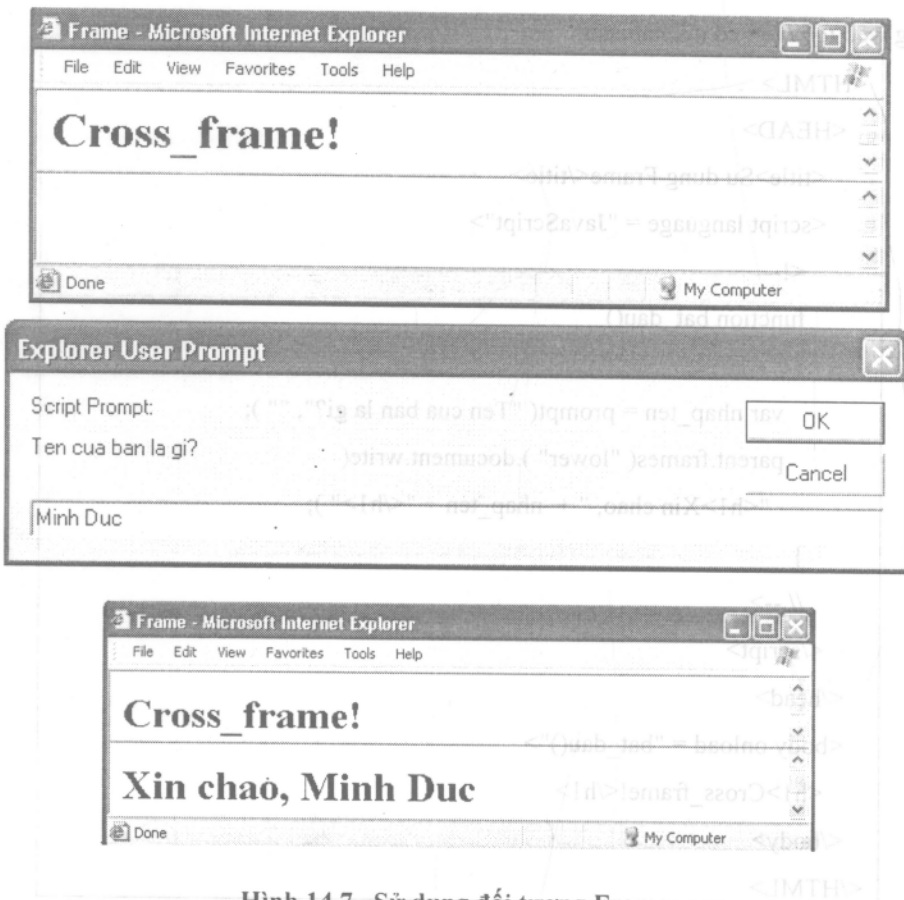
Hàm *document.write()* dùng để hiển thị nội dung lên trình duyệt. *Frame* cha của *frame* hiện tại được tham chiếu đầu tiên, sau đó là *frame lower*.

```
parent.frames( "lower" ).document.write(
  "<h1>Xin chao, " + nhap_ten + "</h1>" );
```

- Kết quả được minh họa trong trình duyệt, đầu tiên *frame* thứ nhất hiển thị nội dung của trang web *top.htm*: in ra dòng chữ *Cross_frame* và một hộp thoại cho người dùng nhập tên của mình vào. Khi người duyệt web bấm vào nút Ok, thì *frame lower* được khởi gán giá trị mới, hiển thị lời chào và tên của người mới nhập vào.

Chương 14

Giới thiệu về DHTML



Hình 14.7. Sử dụng đối tượng Frame.

Sử dụng *frame* giúp nội dung một trang web trở nên sinh động hơn. Các *frame* có thể hoạt động độc lập trên cùng một trang web cho phép chúng ta thiết kế trang web dễ dàng hơn.

14.4.4. Đối tượng Navigator

Đối tượng *Navigator* cung cấp thông tin về trình duyệt. Nó hỗ trợ một phương thức cho phép tải một URL mới, thay thế URL hiện tại.

Các thuộc tính của đối tượng Navigator

Các thuộc tính của đối tượng *Navigator* được giới thiệu trong bảng sau.

Chương 14

Giới thiệu về DHTML

Bảng 14.7. Các thuộc tính của đối tượng Navigator

Tên	Mô tả
appName	Lấy tên mã của trình duyệt
javaEnabled	Trả về kết quả xem ngôn ngữ Java có được cho phép dùng hay không
onLine	Trả về kết quả xem hệ thống đang ở chế độ offline không
platform	Lấy tên của hệ điều hành mà người dùng đang sử dụng
plugins	Lấy tập hợp tất cả các đối tượng nhúng trong tài liệu
userAgent	Lấy một chuỗi tương đương với header yêu cầu của giao thức HTTP

Các phương thức của đối tượng Navigator

Các phương thức của đối tượng *Navigator* được giới thiệu trong bảng sau.

Bảng 14.8. Các phương thức của đối tượng Navigator

Tên	Mô tả
javaEnabled	Trả về kết quả xem ngôn ngữ Java có được cho phép dùng hay không
taintEnabled	Có cho phép dữ liệu <i>tainting</i> hay không

14.4.5. All Collection

All collection trả về một tham chiếu tới tập hợp các phần tử chứa trong đối tượng. Nó có một đối tượng phần tử cho mỗi thẻ HTML hợp lệ. Nếu một thẻ hợp lệ có một thẻ kết thúc tương hợp thì cả hai thẻ đó có thể được bao gồm hoặc không bao gồm trong tài liệu.

Nếu tài liệu chứa các thẻ không xác định được hoặc không hợp lệ thì tập hợp đó gồm một đối tượng phần tử cho mỗi thẻ. Khác với các thẻ kết thúc hợp lệ, các thẻ kết thúc không hợp lệ được thể hiện bởi các đối tượng phần tử của riêng chúng.

Ví dụ sau sẽ chỉ ra các phần tử có trong một trang web.

Chương 14

Giới thiệu về DHTML

Ví dụ 14.9.

```
<HTML>
<HEAD>
  <title>Mô hình đối tượng</title>
  <script language = "JavaScript">
    <!--
      var elements = "";
      function bat_dau()
      {
        for ( var lap = 0; lap < document.all.length; ++lap )
          elements += "<br />" + document.all[ lap ].tagName;
        object1.innerHTML += elements;
        alert( elements );
      }
    // -->
  </script>
</HEAD>
<body onload = "bat_dau()">
  <p id = "object1">Các thành phần trên trang web này:</p>
</body>
</HTML>
```

Trong ví dụ trên, vòng lặp *for* duyệt các phần tử trong trang web thông qua *all collection*

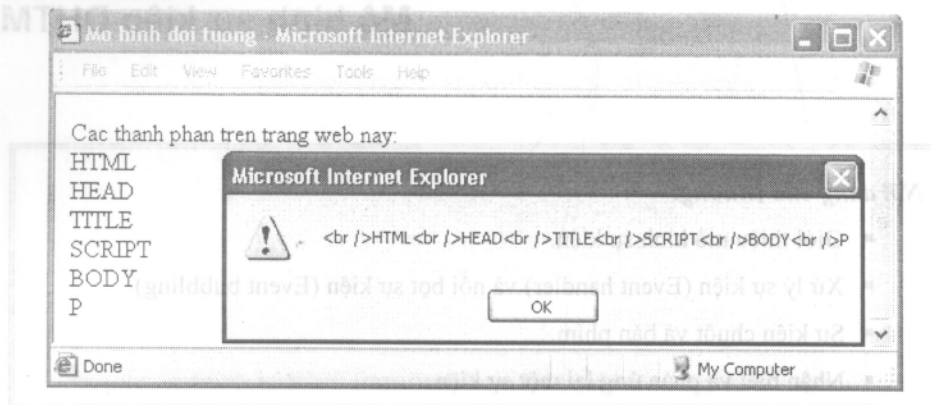
- Thuộc tính *document.all.length* chỉ ra số các phần tử trong tập hợp.
for (var lap = 0; lap < document.all.length; ++lap)
 elements += "
" + document.all[lap].tagName;
- Thuộc tính *tagName* chỉ ra tên tương ứng cho các phần tử trong *all collection*
document.all[lap].tagName

Chương 14

Giới thiệu về DHTML

- Thuộc tính *innerHTML* bao gồm cả định dạng XHTML:
`object1.innerHTML += elements;`

Kết quả của ví dụ trên trong trình duyệt như sau.



Hình 14.8. Ví dụ chỉ ra các thành phần có trên một trang Web.

14.5. Kết chương

Trong chương này chúng ta đã bước đầu tìm hiểu về DHTML – ngôn ngữ HTML động giúp cho việc xây dựng và các thao tác trên trang Web sinh động, dễ dàng và hiệu quả hơn. Sự ra đời của DHTML đã giúp việc lập trình web trở nên dễ dàng hơn nhiều và cho kết quả tốt hơn. Trong chương sau, chúng ta sẽ tìm hiểu về mô hình sự kiện trong DHTML.

Chương 15

Mô hình sự kiện DHTML

Nội dung của chương:

- Giới thiệu mô hình sự kiện.
- Xử lý sự kiện (Event handler) và nổi bọt sự kiện (Event bubbling).
- Sự kiện chuột và bàn phím.
- Nhận biết và phản ứng lại một sự kiện.

15.1. Giới thiệu

Sự kiện là thuộc tính rất quan trọng hỗ trợ sự tương tác đối với trang Web cho người sử dụng. Bạn có thể tạo ra các trình xử lý sự kiện nhằm đáp ứng lại những sự kiện do người sử dụng hoặc do hệ thống tạo ra. Sử dụng sự kiện làm trang web sinh động hơn. Trong chương này, chúng ta sẽ tìm hiểu về những sự kiện hỗ trợ HTML động, cách thay đổi nội dung và vị trí của các phần tử trên trang web theo tính động.

15.2. Đối tượng sự kiện

Mô hình sự kiện (*Event Model*) là một phần quan trọng trong mô hình đối tượng tài liệu (*Document Object Model*). Đó là thành phần chính tạo nên một trang web động và có sự tương tác với người sử dụng. Mô hình sự kiện xác định những sự kiện nào sinh ra cho một phần tử, cách thức sự kiện được gửi đến cho một trình xử lý sự kiện (*event handler*) và dữ liệu có thể cung cấp cho trình xử lý sự kiện.

Một sự kiện (*event*) là một thông báo đáp ứng lại một hành động. Sự kiện đó có thể do hệ thống tạo ra như một sự thay đổi về trạng thái hoặc sự kiện sinh ra do người sử dụng thực hiện một thao tác nào đó, chẳng hạn kích chuột hoặc ấn phím trong lúc xem tài liệu. Một bộ xử lý các sự kiện (*event handler*) là một khối các mã chương trình liên kết với nhau thực hiện điều khiển khi một sự kiện tương ứng xảy ra.

Mỗi bộ xử lý các sự kiện có một đối tượng liên kết sự kiện có nhiệm vụ cung cấp thông tin về sự kiện tương ứng: loại sự kiện và vị trí của con trỏ tại thời gian của sự kiện.

Chương 15

Mô hình sự kiện DHTML

Khi một sự kiện xảy ra, thì các thông tin đó được gửi đi như là một đối số đối với bộ xử lý sự kiện. Ví dụ khi người sử dụng ấn nút con chuột, một sự kiện tác động lên chuột được phát sinh. Những thông tin được xác định bao gồm: loại sự kiện là thao tác tác động lên chuột, các vị trí x và y của con trỏ khi ta ấn nút con chuột, nút chuột nào được sử dụng...

Chúng ta phải chú ý rằng đối tượng sự kiện không thể dùng trực tiếp với đối tượng của sô. Nó được sử dụng làm một phần của trình xử lý sự kiện.

Hiểu rõ về các sự kiện có trong DHTML cho phép chúng ta kiểm soát trang web của mình một cách dễ dàng nhất. Hơn thế nữa các sự kiện làm cho sự tương tác giữa người dùng và trang web trở nên sống động, không bị gò bó vào nội dung “tĩnh” của trang web như trước đây. Dưới đây chúng ta sẽ liệt kê các sự kiện có trong DHTML, đây là những sự kiện hay được sử dụng nhất.

15.2.1. Những sự kiện chuột (Mouse Events)

Bảng dưới đây chỉ ra các sự kiện chuột hay sử dụng.

Bảng 15.1. Các sự kiện chuột hay sử dụng

Sự kiện	Mô tả
onclick	Tạo ra khi người sử dụng kích vào một phần tử, hoặc ấn phím Enter vào phần tử đó.
ondblclick	Tạo ra khi người sử dụng kích đúp chuột lên một phần tử.
onmouseout, onmousedown onmouseup onmousemove onmouseover	Những sự kiện được tạo ra đáp ứng theo chuyển động tương ứng của chuột.

Chúng ta sẽ tìm hiểu cụ thể các sự kiện đặc trưng.

Sự kiện onclick (Event onclick)

Sự kiện này được sinh ra khi người sử dụng kích vào một phần tử hoặc ấn vào phím Enter lên phần tử đó.

Ví dụ sau đây minh hoạ một sự kiện *onclick* đơn giản. Khi người sử dụng bấm chuột vào một nút trong trình duyệt, thì ngay lập tức xuất hiện một hộp thoại hiển thị một lời chào. Nghĩa là hộp thoại này chỉ xuất hiện khi người dùng tương tác với nút.

Chương 15

Mô hình sự kiện DHTML

Ví dụ 15.1.

```
<HTML>
<HEAD>
  <TITLE>Mô hình doi tuong _onclick</ TITLE >
  <script language = "JavaScript"
    event = "onclick">
    <!--
      alert( "Xin chao" );
    // -->
  </script>
</HEAD>
<BODY>
  <P id = "para">Bam tren dong chu nay!</P>
  <!-- Chi ra su kien o dong nay -->
  <input type = "button" value = "Click Me!"
    onclick = "alert( 'Chao ban' )" />
</BODY>
</HTML>
```

Trong ví dụ trên,

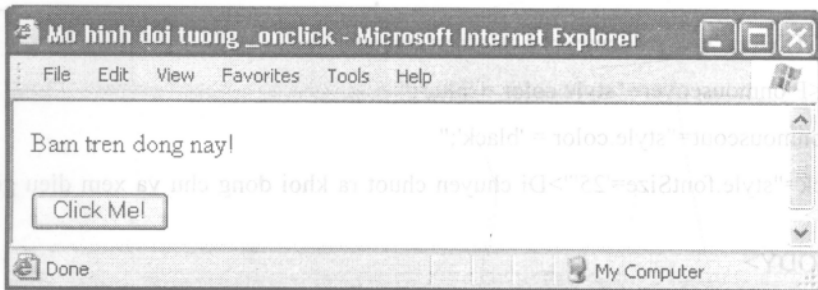
- Để có được sự kiện *onclick*, chúng ta phải chỉ ra sự kiện đó:
onclick = "alert('Chao ban')" ;
- Một hộp thoại hiển thị với lời chào khi chúng ta kích chuột vào nút này. Như vậy sự kiện *onclick* chỉ được gọi khi người sử dụng kích chuột.

Kết quả của ví dụ 15.1 được minh hoạ trong hình sau:

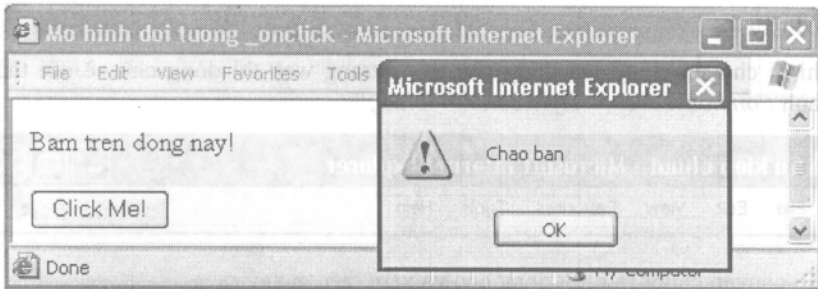
- Khi chưa bấm chuột vào nút *Click Me!*

Chương 15

Mô hình sự kiện DHTML



- Khi bấm chuột vào nút *Click Me!*



Hình 15.1. Minh hoạ cho sự kiện onclick.

Sự kiện onmouseover và sự kiện onmouseout.

Những sự kiện này đều được sinh ra nhằm đáp ứng lại sự chuyển động tương ứng của chuột. Ta phân biệt 2 sự kiện này:

- Sự kiện *onmouseover* tạo ra khi con trỏ chuột di chuyển trên một phần tử.
- Sự kiện *onmouseout* tạo ra khi con trỏ chuột rời khỏi một phần tử.

Trước khi tìm hiểu về các thuộc tính của hai sự kiện này chúng ta hãy xem ví dụ sau để hiểu rõ về tác dụng và chức năng của hai sự kiện này trong trình duyệt web. Đây là một ví dụ trong đó các sự kiện khác nhau của chuột được kiểm soát và trang web sẽ thay đổi động theo để đáp lại những sự kiện đó.

Ví dụ 15.2.

```
<HTML>
<HEAD>
  <TITLE>Su kien chuot</TITLE>
</HEAD>
```

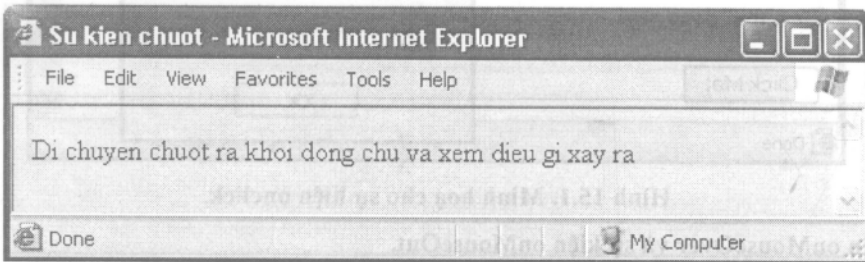

Chương 15

Mô hình sự kiện DHTML

```
<BODY onClick="document.bgColor='lavender'">
  <P onmouseover="style.color = 'blue';"
    onmouseout="style.color = 'black';"
    onclick="style.fontSize='25'">Di chuyển chuột ra khỏi dòng chữ và xem điều gì xảy
    ra</P>
</BODY>
</HTML>
```

Trong ví dụ 15.2,

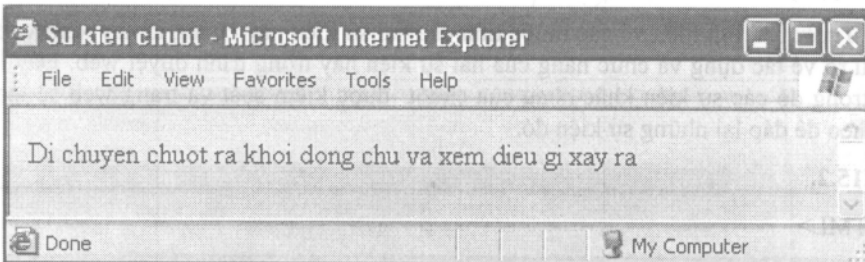
- khi di chuyển chuột lên dòng chữ trong trang web thì dòng chữ sẽ đổi thành màu xanh : `onmouseover="style.color = 'blue';"`



Hình 15.2. Minh họa cho sự kiện `onmouseover`.

- Khi di chuyển chuột ra khỏi dòng chữ đó thì dòng chữ sẽ đổi màu thành đen:

`onmouseout="style.color = 'black';"`



Hình 15.3. Minh họa cho sự kiện `onmouseout`.

Bảng sau đây liệt kê các thuộc tính của hai sự kiện chuột này

Chương 15

Mô hình sự kiện DHTML

Bảng 15.2. Các thuộc tính của sự kiện `onMouseOver` và sự kiện `onMouseOut`

Thuộc tính	Miêu tả
Altkey	Trả về giá trị <i>true</i> nếu phím <i>Alt</i> được bấm khi gọi sự kiện.
Button	Nhận dạng nút chuột được nhấn (1: chuột trái, 2: chuột phải, 3: chuột phải và chuột trái, 4: chuột giữa, 5: chuột trái và chuột giữa, 6: phải và giữa, 7:tất cả).
ClientX/ClientY	Trả về tọa độ của con chuột trong trình duyệt.
ctrlKey	Trả về giá trị <i>True</i> nếu phím <i>Ctrl</i> được bấm khi gọi một sự kiện.
PropertyName	Tên của thuộc tính được thay đổi trong sự kiện.
Recorset	Tham chiếu đến trường dữ liệu của một bản ghi.
ReturnValue	Xác định giá trị trả về của sự kiện hay của cửa sổ hội thoại.
srcElement	Một tham chiếu đến đối tượng được gọi sự kiện.
Type	Tên của sự kiện được gọi.

Ví dụ 15.3.

Trong ví dụ sau, chúng ta sẽ minh họa sự kiện `onmousemove` của chuột.

```
<HTML>
<HEAD>
  <TITLE>su kien onmousemove </TITLE>
  <script language = "JavaScript">
    <!--
    function Xac_dinh_toa_do_chuot()
    {
      toa_do.innerText = event.srcElement.tagName +
        " (" + event.offsetX + ", " + event.offsetY + ")";
    }
    // -->
  </script>
</HEAD>
<BODY style = "background-color: wheat" onmousemove =
  "Xac_dinh_toa_do_chuot()">
```

Chương 15

Mô hình sự kiện DHTML

```
<SPAN id = "toa_do">(0, 0)</SPAN><br />
<IMG src = "4s[1].JPEG" style = "position: absolute;
top: 50; left: 50" alt = "Deitel" />
</BODY>
</HTML>
```

Khi con chuột di chuyển trên nền của trang web, thì sẽ trả về cho chúng ta tọa độ tương ứng của con trỏ chuột, đồng thời xác định con trỏ đang chỉ vào phần tử nào trong trang web. Để làm được điều đó, ta phải xây dựng một hàm:

```
function Xac_dinh_toa_do_chuot()
{
    toa_do.innerText = event.srcElement.tagName +
    " (" + event.offsetX + ", " + event.offsetY + ")";
}
```

- Thuộc tính *innerText* của phần tử *toa_do* sẽ gán vào ký tự là tên của phần tử mà con trỏ đang trỏ vào.
- Thuộc tính *offsetX* và *offsetY* của đối tượng *Event* cho ta tọa độ trong trình duyệt.

Sự kiện chuột *onmousemove* sẽ được gọi mỗi lần khi người dùng di chuyển chuột

```
<BODY onmousemove = "Xac_dinh_toa_do_chuot()">
```

Kết quả minh họa trong trình duyệt:



Hình 15.4. Minh họa cho sự kiện *onmousemove*.

15.2.2. Các sự kiện *focus* và *onblur*

Sự kiện *onfocus* xuất hiện khi một phần tử nhận được tiêu điểm. Sự kiện *onblur* xuất hiện khi một điều khiển không còn tiêu điểm.

Sự kiện *onfocus*.

Sự kiện *onfocus* được gửi đi khi một phần tử trên *form* trở thành phần tử *form* hiện thời, nghĩa là được nhận tiêu điểm. Chỉ khi một phần tử đã có tiêu điểm thì nó mới có thể nhận dữ liệu nhập từ người dùng. Điều này xảy ra khi người dùng nhấp chuột vào phần tử, hay nếu người dùng gõ phím *Tab* hoặc *Shift- Tab* (để di chuyển tới phần tử trước hoặc sau trong danh sách các phần tử trên *form*).

Sự kiện *onblur*

Trái ngược với sự kiện *focus* là sự kiện *blur*. Khi người dùng rời khỏi một phần tử thì sự kiện *onBlur* được kích hoạt. Đối với một số phần tử nếu nội dung bị thay đổi thì sự kiện *onChange* được kích hoạt.

Ví dụ sau minh họa cho việc sử dụng các sự kiện *focus* và *onblur* trong một trang web.

Ví dụ 15.4.

```
<HTML>
<HEAD>
  <TITLE>su kien onfocus va onblur</TITLE>
  <script language = "JavaScript">
    <!--
    var mang_chu =
      [ "Nhập tên trong hộp này.",
        "Nhập địa chỉ email của bạn vào đây, " +
        "in the format user@domain.",
        "Đánh dấu vào đây nếu như bạn thích trang web này.",
        "trong hộp này, Viết bất cứ gì bạn muốn" +
        "cho chúng ta đọc.",
        "Nút này xác định form " +
        "server-side script",
        "Nút này để xóa các form",
```

Chương 15

Mô hình sự kiện DHTML

```
"TextArea cung cap cac ngu canh cam xuc " +
"giup do. Bam chuot hoac dung TAB " +
"Phim de biet nhieu thong tin hon ve " +
"truong vao." ];
function ham_chu( chi_muc )
{
    myForm.helpBox.value = mang_chu[ chi_muc ];
}
// -->
</script>
</HEAD>
<BODY>
    <FORM id = "myForm" action = "">
    Ten: <INPUT type = "text" name = "ten_ban"
        onfocus = "ham_chu(0)" onblur = "ham_chu(6)" /><BR />
    Email: <INPUT type = "text" name = "email"
        onfocus = "ham_chu(1)" onblur = "ham_chu(6)" /><BR />
    Bam vao day neu ban thich trang web nay
    <INPUT type = "checkbox" name = "like" onfocus =
        "ham_chu(2)" onblur = "ham_chu(6)" /><BR /><HR />
    Bat cu cau gi?<BR />
    <TEXTAREA name = "chu_thich" rows = "5" cols = "45"
        onfocus = "ham_chu(3)" onblur = "ham_chu(6)">
    </ TEXTAREA ><BR />
    <INPUT type = "submit" value = "Submit" onfocus =
        "ham_chu(4)" onblur = "ham_chu(6)" />
    <INPUT type = "reset" value = "Reset" onfocus =
```

Chương 15

Mô hình sự kiện DHTML

```
"ham_chu(5)" onblur = "ham_chu(6)" />  
< TEXTAREA name = "helpBox" style = "position: absolute;  
right: 0; top: 0" rows = "4" cols = "45">  
TextArea này cung cap cac ngu canh cam xuc. Bam chuot hoac dung TAB  
de biet thong tin cua truong vao.</ TEXTAREA >  
</FORM>  
</BODY>  
</HTML>
```

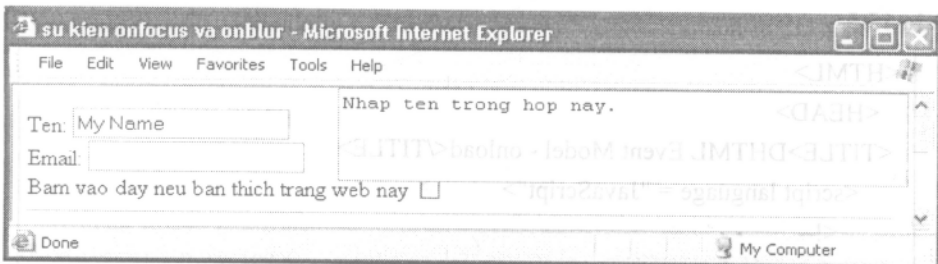
- Trong ví dụ này, chúng ta tạo ra một trang web gồm các phần tử HTML: *TextBox*, *TextArea*, *checkbox*. Tác dụng đoạn mã trên là khi chúng ta trỏ chuột đến thành phần nào trong trang thì các xâu kí tự tương ứng được chúng ta đặt trước trong một mảng sẽ xuất hiện với nội dung mô tả chức năng thành phần đó. Đây gọi là sự kiện *focus*, chẳng hạn như với hộp *TextArea*, để gọi sự kiện *focus* chúng ta dùng dòng lệnh:

```
<TEXTAREA name = "comments" rows = "5" cols = "45"  
onfocus = "ham_chu(3)" onblur = "ham_chu(6)">  
</ TEXTAREA ><BR />
```

Khi người sử dụng trỏ chuột vào *TextArea* có tên là *chu_thich* thì sự kiện *focus* xảy ra, *ham_chu(3)* sẽ được gọi và hiển thị chuỗi kí tự đã khai báo sẵn trong mảng.

- Mỗi một phần tử HTML trong ví dụ này đều có hai sự kiện *onfocus* và *onblur*. Sự kiện *onblur* xảy ra khi người dùng chuyển con trỏ chuột sang thành phần khác. Và lúc đó thành phần khác lại nhận sự kiện *onfocus*. Trong thành phần *TextArea* sự kiện *onblur* sẽ gọi một phần tử trong *mang_chu* để hiển thị khi mất sự kiện *onfocus*.

Hình sau minh họa sự kiện *onfocus* và *onblur* của thành phần *TextArea* trong trình duyệt:

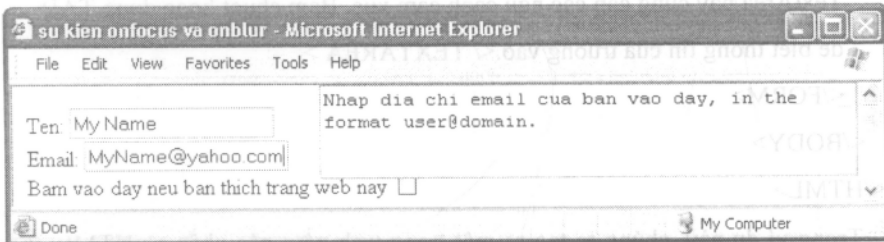


Hình 15.5. Minh họa cho sự kiện *onfocus*.

Chương 15

Mô hình sự kiện DHTML

Trong hình trên ô nhập *Ten* đang có sự kiện *focus* nên dòng chữ “*Nhap ten trong hop nay*” tương ứng xuất hiện ở trong *textarea* bên cạnh, đồng thời lúc đó các thành phần còn lại trên trang có sự kiện *onblur*. Khi người dùng trỏ chuột (hoặc dùng TAB) vào ô nhập *Email* thì ô nhập *Ten* sẽ mất sự kiện *onfocus* và xảy ra sự kiện *onblur*, trong khi ô nhập *Email* lại có sự kiện *onfocus*.



Hình 15.6. Minh hoạ cho sự kiện *onblur*.

15.2.3. Những sự kiện khác

Ngoài các sự kiện đã nêu ở trên, trong DHTML còn có các sự kiện đáng chú ý sau.

Bảng 15.3. Các sự kiện trong DHTML

Sự kiện	Mô tả
Onload	Xuất hiện khi được tải lên trang và hiển thị
onunload	Xuất hiện khi không được tải lên và người dùng chuyển qua trang khác
onreset	Xuất hiện khi người sử dụng chọn nút Reset trên một form
onsubmit	Xuất hiện khi người sử dụng chọn nút Submit trên một form

Đáng chú ý nhất trong các sự kiện này là sự kiện *Onload*. Sự kiện này xuất hiện sau khi trang và các phần tử trên trang được tải lên và hiển thị. Ví dụ sau minh hoạ cho sự kiện *onload*.

Ví dụ 15.5.

```
<HTML>
<HEAD>
<TITLE>DHTML Event Model - onload</TITLE>
<script language = "JavaScript">
  <!--
    var seconds = 0;
    function startTimer() {
```

Chương 15

Mô hình sự kiện DHTML

```
// 1000 milli giây = 1 giây
window.setInterval( "updateTime()", 1000 );
}

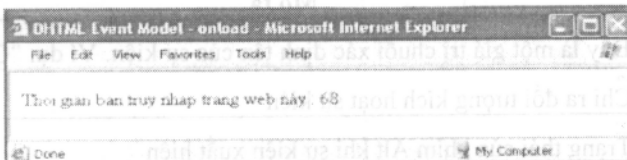
function updateTime() {
    seconds++;
    soFar.innerText = seconds;
}

// -->
</script>
</HEAD>
<BODY onload = "startTimer()">
    <P>Thời gian bạn truy nhập trang web này :
    <A id = "soFar"><STRONG>0</STRONG></A></p>
</BODY>
</HTML>
```

Ta sẽ bắt đầu phân tích ví dụ này:

- Hàm *startTimer* sẽ gọi hàm *updateTime* sau mỗi giây. Phương thức *Window.setInterval* được sử dụng để gọi hàm *updateTime* sau mỗi giây.
- Hàm *updateTime* cài đặt thuộc tính *innerText* của thành phần có ID là *soFar* để lấy giá trị thời gian trôi qua.

Sự kiện *onload* được thực thi khi một thành phần đã được nạp xong.



Trang web sẽ trở nên động để thể hiện thời gian mà trang web đã được load sau mỗi giây.

Hình 15.7. Minh họa cho sự kiện *onload*.

Chương 15

Mô hình sự kiện DHTML

15.3. Xử lý sự kiện

Một trình xử lý sự kiện là một tập hợp những câu lệnh được thực hiện để đáp ứng lại một sự kiện nào đó. Ví dụ, bạn có thể gán một hàm xử lý của mình vào sự kiện *onclick* của một nút lệnh như sau

```
<INPUT type = button onclick = "Ham_thuc_hien( )">
```

15.3.1. Định nghĩa xử lý sự kiện

Có nhiều cách để định nghĩa một hàm xử lý sự kiện. Ví dụ với sự kiện *onclick* của một nút lệnh, ta có thể định nghĩa như ví dụ trên:

```
<input type="button" name="button" value="Click Here" onClick="ham_click();">
```

Chúng ta còn có thể định nghĩa một trình xử lý sự kiện bằng thẻ `<SCRIPT>`. Nghĩa là, có thể tạo ra một script áp dụng vào một sự kiện riêng đối với một phần tử đã xác định. Các thuộc tính *FOR* và *EVENT* của thẻ `<SCRIPT>` được dùng, ví dụ.

```
<FORM name="form1">
```

```
    <input type="button" name="button1" value="click here">
```

```
</FORM>
```

```
<script for="button1" event="onclick" language="JavaScript">
```

```
    ... Code để điều khiển sự kiện
```

```
</script>
```

Bảng sau đây liệt kê một số thuộc tính của đối tượng sự kiện.

Bảng 15.4. Một số thuộc tính của đối tượng sự kiện

Thuộc tính	Mô tả
type	Đây là một giá trị chuỗi xác định tên của sự kiện. Ví dụ: "Click"
srcElement	Chỉ ra đối tượng kích hoạt sự kiện
altKey	Trạng thái của phím Alt khi sự kiện xuất hiện
ctrlKey	Trạng thái của phím Ctrl khi sự kiện xuất hiện
shiftKey	Trạng thái của phím Shift khi sự kiện xuất hiện
keyCode	Giá trị số nguyên để nhận dạng phím kích hoạt sự kiện
button	Nhận dạng nút chuột được nhấn

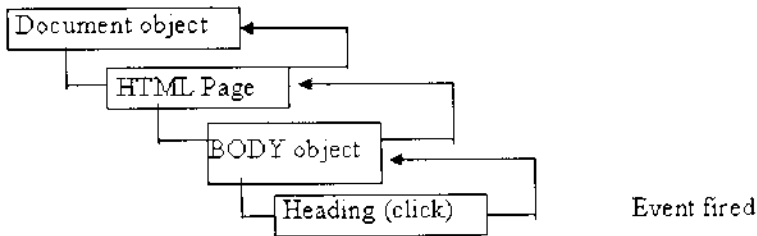
Chương 15

Mô hình sự kiện DHTML

x,y	Xác định con trỏ khi trỏ đến đối tượng
screenX, screenY	Xác định vị trí con trỏ trên màn hình desktop
returnValue	Xác định giá trị trả về của sự kiện hay của cửa sổ hội thoại
cancelBubble	Chế độ ngăn ngừa sự kiện hiện thời không thực hiện quá trình nổi bọt sự kiện

15.3.2. Nổi bọt sự kiện (Event bubbling)

Trước đây, nếu chúng ta muốn xác định được hành vi mặc định cho một sự kiện, ví dụ như *button click* thì chúng ta phải mô tả đoạn mã cho mỗi đối tượng đó. Nhưng trong HTML động chúng ta không cần thiết phải làm điều này nữa. HTML động cung cấp cho chúng ta một hệ thống phân cấp đối tượng đối với các tài liệu HTML. Nghĩa là, ta có thể viết bộ xử lý sự kiện chung cùng nhiều đối tượng và khi một sự kiện phát sinh trong một đối tượng con thì nó có thể chuyển đến các phần tử phân cấp mức cha trên trang cho tới khi tìm thấy bộ xử lý sự kiện. Quá trình này được gọi là "event bubbling" - nổi bọt sự kiện.



Hình 15.8. Quá trình nổi bọt sự kiện.

Trong sơ đồ trên, khi sự kiện kích chuột vào một liên kết xảy ra thì sự kiện đó chuyển ngược lại quá trình phân cấp cho đến khi nó tìm ra trình điều khiển sự kiện tương ứng. Sự phân cấp này được xác định bởi luồng HTML chứ không phải theo thứ tự các hạng mục (item) được hiển thị lên màn hình. Ví dụ bạn có thể định vị *heading* ở đầu trang dù sự kiện đó sẽ thực hiện quá trình nổi bọt ngược lên đến phần tử *BODY*.

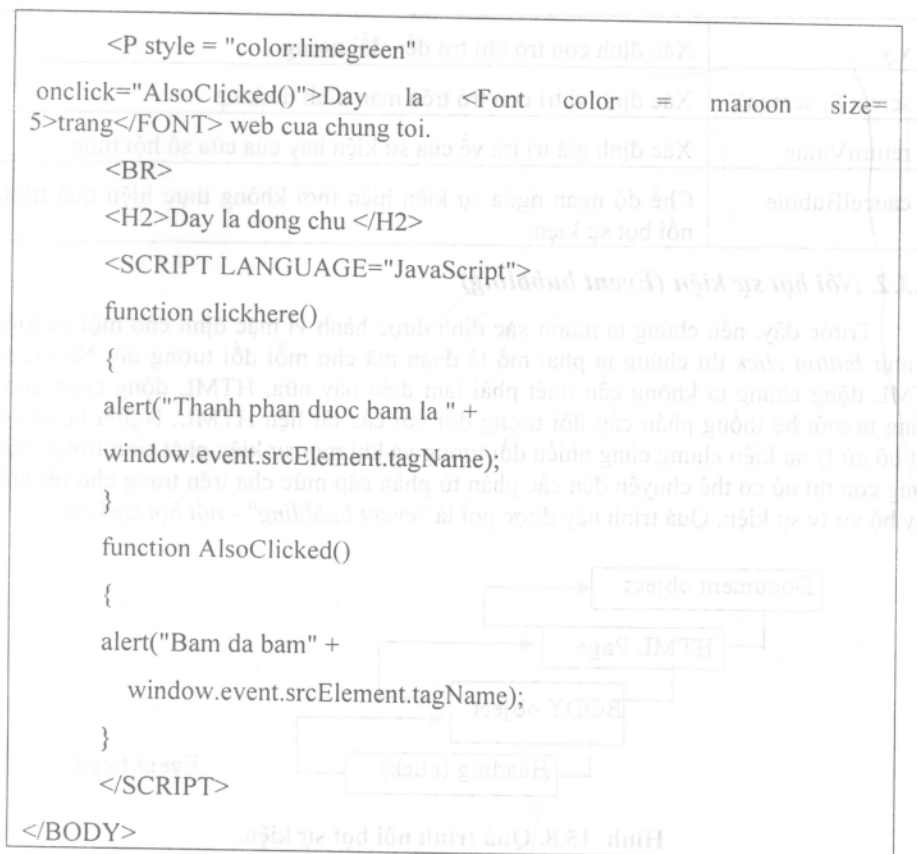
Ví dụ sau minh họa cho sự kiện nổi bọt.

Ví dụ 15.6.

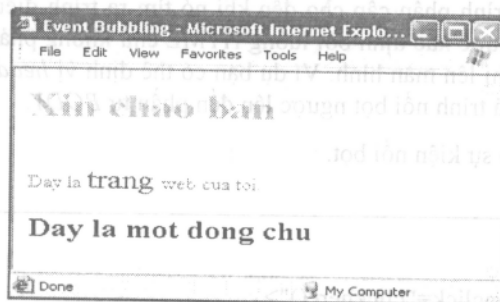
```
<HTML>
  <BODY onclick="clickhere()">
    <H1 style="color: hotpink">Xin Chao ban</H1>
    <BR>
```

Chương 15

Mô hình sự kiện DHTML



Kết quả hiển thị trên trình duyệt:



Hình 15.9. Minh họa cho sự kiện nổi bọt.

Chúng ta hãy phân tích đoạn mã trên.

Chương 15

Mô hình sự kiện DHTML

- Khi người sử dụng kích vào tiêu đề "Xin chào bạn" thì sự kiện *onclick* xảy ra. Tuy nhiên không có bộ xử lý sự kiện nào được gán cho *H1* vì vậy sự kiện *onclick* sẽ tạo ngược lại sự phân cấp đối với phần tử *BODY*. Phần tử *BODY* có một bộ xử lý sự kiện gọi là *clickhere()* đối với sự kiện *onclick*, bộ xử lý này có nhiệm vụ hiển thị dòng chữ "Thanh phan duoc bam la H1".
- Khi người dùng kích vào từ "trang", thì sự kiện *onclick* đối với phần tử *FONT* được sinh ra. Do không có trình xử lý sự kiện đối với phần tử *FONT*, nên sự kiện *onclick* tạo ngược lại sự phân cấp đối với phần tử *P*. Phần tử *P* có một bộ xử lý sự kiện *AlsoClicked()* dùng để hiển thị thông báo "Bạn đã bấm FONT", trình sự kiện lúc đó vẫn tiếp tục đường dẫn của nó ngược lên sự phân cấp đến phần tử *BODY*. Trình xử lý sự kiện *clickhere()* đã gán cho phần tử *BODY* sẽ được kích hoạt và thông báo "Bạn đã bấm FONT" được hiển thị.

Nếu bạn không muốn các sự kiện gây ngược lại sự phân cấp, thì bạn đặt thuộc tính *cancelBubble* của đối tượng sự kiện là *True* cho bất kỳ bộ xử lý sự kiện nào. Sau khi trình xử lý trả về thì sự kiện đó ngừng tạo ra chuỗi sự kiện và đi đến kết thúc ngay lập tức.

Ví dụ 15.7.

```
<HTML>
  <BODY onclick="setBodyStyle()">
    <H1>Xin chào bạn</H1>
    <P style="color:aqua" onclick="setParaStyle()">Đây là<FONT color =
maroon>trang</FONT> web của chúng tôi.
    <H2> Đây là một định kiểu H2</H2>
    <H1>Đây là một định kiểu H1</H1>
    <SCRIPT LANGUAGE="JavaScript">
      function setBodyStyle()
      {
        var head1 = document.all.tags("H1");
        for(i=0; i<head1.length;i++)
          head1.item(i).style.color="hotpink";
      }
      function setParaStyle()
      {
```

Chương 15

Mô hình sự kiện DHTML

```
var pstyle = window.event.srcElement;
while ((pstyle != null) && (pstyle.tagName != "P"))
{
    pstyle = pstyle.parentElement;
}
if(pstyle != null) pstyle.style.textDecoration = "underline";
    window.event.cancelBubble = true;
}
</SCRIPT>
</BODY>
</HTML>
```

Điểm thuận lợi trong việc sử dụng quá trình tạo thành chuỗi sự kiện là bạn không còn phải viết đoạn mã cho mỗi phần tử. Chẳng hạn, khi người dùng kích vào hộp kiểm tra (*check box*) thì điển hình là bạn sẽ phải viết mã đối với sự kiện *click* của mỗi hộp kiểm tra nhưng thay vì thế, bạn có thể viết thủ tục trên chính *form* đó. Khi một trong những hộp kiểm tra được chọn, thì sự kiện đó sẽ tự động tạo ra chuỗi sự kiện cho một *form* mẹ và xử lý ở đây.

15.3.3. Giá trị trả về

Một trình xử lý sự kiện hiển nhiên được tạo ra để thực hiện một công việc nào đó nhằm đáp ứng lại sự kiện phát sinh. Nó còn có thể trả về một giá trị cho sự kiện đó. Điều này được thực hiện bằng hai cách:

- câu lệnh *return* trong ngôn ngữ JavaScript.
- Sử dụng giá trị *return value* của đối tượng sự kiện.

Không phải tất cả các phần tử đều hỗ trợ cho tất cả các sự kiện. Bảng sau đây liệt kê một số các phần tử của mô hình đối tượng tài liệu và các sự kiện liên quan của chúng.

Bảng 15.5. Các phần tử và các sự kiện có liên quan trong IE

Sự kiện	Phần tử	Mô tả
onabort	Image	Được kích hoạt khi người sử dụng huỷ bỏ việc tải một hình ảnh bằng cách kích vào một kết nối hoặc nút Stop.

Chương 15
Mô hình sự kiện DHTML

onblur	Window, frame, all form element	Khi người sử dụng bỏ tiêu điểm ra khỏi một cửa sổ, khung hay một phần tử form.
onclick	Button, radio button, check box, submit button, reset button, link	Được kích hoạt khi người sử dụng kích trái vào nút chuột .
onchange	Text field, textarea, select list	Được kích hoạt khi người sử dụng thay đổi giá trị của phần tử.
onfocus	Window, frame, all form element	Được kích hoạt khi người sử dụng đặt tiêu điểm vào một cửa sổ, khung, hay phần tử form.
onload	Document, applet, frameset, img, link, object, script, style, window	Được kích hoạt khi trang được nạp trình duyệt.
onmousedown	Button, document, link	Được kích hoạt khi người sử dụng ấn nút chuột.
onmouseout	Area, layer, link	Được kích hoạt khi người sử dụng di chuyển con trỏ ra khỏi một phần tử.
onmouseover	Area, layer, link	Được kích hoạt khi người sử dụng di chuyển con trỏ qua một phần tử.
onmouseup	Button, document, link	Được kích hoạt khi người sử dụng thả nút chuột đã được ấn.
onreset	Form	Người sử dụng thiết lập lại form ở trạng thái ban đầu.
onresize	Window, frame	Xảy ra khi người sử dụng đặt lại kích thước một cửa sổ hoặc một khung.
onsubmit	Form	Được kích hoạt khi người sử dụng gửi đi một form.
onunload	Document, frameset, image, window	Được kích hoạt khi người sử dụng chuyển sang một trang khác.

15.3.4. Kiểm soát lỗi với sự kiện *onerror*

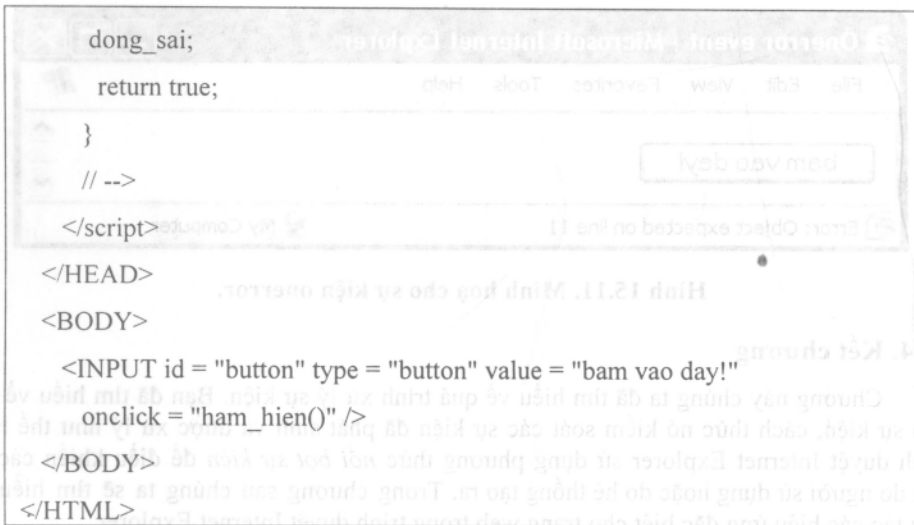
Trong quá trình lập trình, việc người lập trình viết sai chính tả của câu lệnh là điều rất thường xuyên xảy ra. Điều này có thể được trình duyệt phát hiện cho người lập trình thông qua sự kiện *onerror*. Sự kiện này được thực thi để chỉ ra lỗi xảy ra trong dòng lệnh của bạn ở trong chương trình. Nếu không dùng sự kiện *onerror* thì khi thực thi chương trình của chúng ta sẽ không được hiển thị trong trình duyệt web, còn khi dùng sự kiện này thì chương trình của chúng ta vẫn được hiển thị trong trình duyệt và gặp lỗi sai thì sự kiện *onerror* sẽ được kích hoạt để đưa ra thông báo lỗi tương ứng. Điều này không những giúp người lập trình biết được lỗi của mình mà còn giúp cho người sử dụng biết được điều gì đang xảy ra. Ví dụ sau sẽ minh họa điều đã nói trên.

Ví dụ 15.8.

```
<HTML>
<HEAD>
<TITLE>Onerror event</TITLE>
<script language = "JavaScript">
  <!--
    //duoc chi ra neu mot su kien onerror duoc kích hoạt
    //trong window hàm soát_loi sẽ được thực hiện
    window.onerror = soát_loi;
    function hàm_hien()
    {
      alert( "xin chào bạn" ); // alert danh sai,tao nên một lỗi
    }
    // sự kiện ONERROR được truyền vào 3 tham số
    //function: tên của lỗi, địa chỉ url của file,
    //và vị trí dòng xảy ra lỗi
    function soát_loi( kiểu_lỗi, url, dòng_sai )
    {
      // viết ở thanh trạng thay phía dưới của cửa sổ trình duyệt
      window.status = "Error: " + kiểu_lỗi + " on line " +
```

Chương 15

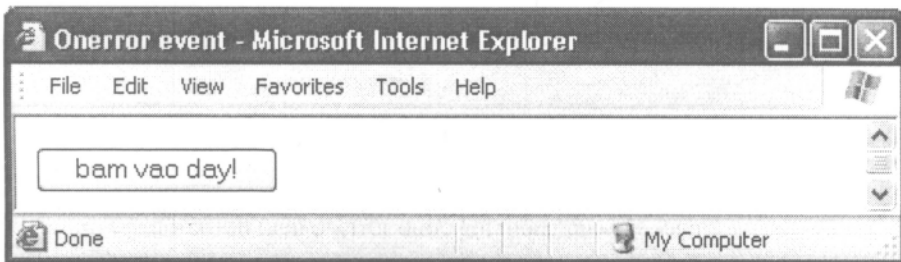
Mô hình sự kiện DHTML



- Trong ví dụ trên, sự kiện *onerror* được gọi thông qua câu lệnh:
`window.onerror = soat_loi;`
- Sự kiện *onerror* cho phép người lập trình kiểm soát lỗi một cách mềm dẻo hơn. Khi một sự kiện *onerror* được kích hoạt thì ngay lập tức hàm *ham_hien()* sẽ được gọi.
- Hàm *soat_loi(kieu_loi,url,dong_sai)* dùng để hiển thị thông báo về kiểu lỗi trên thanh trạng thái của trình duyệt. Hàm này trả về giá trị *True* nếu như lỗi đã kiểm soát thành công.

Trên trình duyệt sẽ cho ta kết quả như hình sau:

- Khi chúng ta chưa kích chuột vào nút “bam vào đây!”:

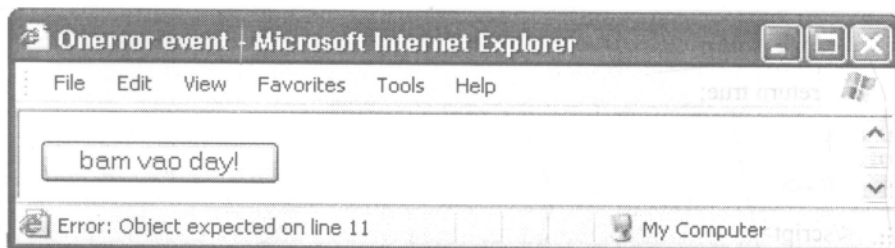


Hình 15.10. Giao diện ban đầu.

- Khi kích chuột vào nút “bam vào đây!” thì sự kiện *onerror* được kích hoạt và đưa ra thông báo lỗi nằm trên thanh *status* của cửa sổ Window.

Chương 15

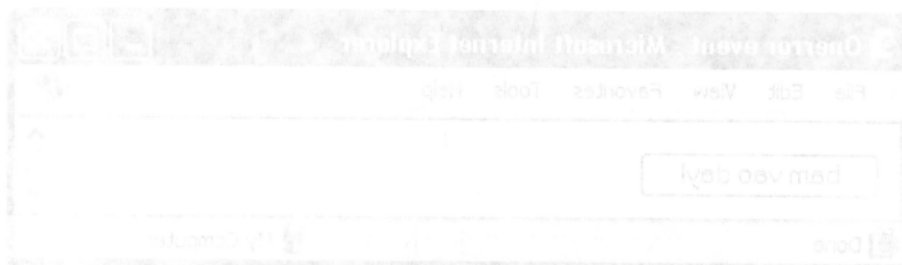
Mô hình sự kiện DHTML



Hình 15.11. Minh hoạ cho sự kiện onerror.

15.4. Kết chương

Chương này chúng ta đã tìm hiểu về quá trình xử lý sự kiện. Bạn đã tìm hiểu về mô hình sự kiện, cách thức nó kiểm soát các sự kiện đã phát sinh và được xử lý như thế nào. Trình duyệt Internet Explorer sử dụng phương thức *nổi bọt sự kiện* để điều khiển các sự kiện do người sử dụng hoặc do hệ thống tạo ra. Trong chương sau chúng ta sẽ tìm hiểu về cách tạo các hiệu ứng đặc biệt cho trang web trong trình duyệt Internet Explorer.



Chương 16

Các hiệu ứng đặc biệt

Nội dung của chương:

- Tìm hiểu về cách tạo các hiệu ứng đặc biệt cho trang web trong Internet Explorer.
- Các hiệu ứng lọc (*Filters*).
- Các hiệu ứng chuyển tiếp (*Transition*).
- Tạo ra hoạt ảnh động cho trang web.

16.1. Giới thiệu

Thuật ngữ *đa phương tiện* (*Multimedia*) nhằm nói đến việc sử dụng các phương tiện thông tin khác nhau như văn bản, âm thanh, hình ảnh,... Trong HTML động, *multimedia* ám chỉ nhiều hơn đến các hiệu ứng đặc biệt mà bạn có thể tạo ra cho một trang web. Sử dụng các hiệu ứng đặc biệt này làm cho trang web trở nên động hơn, giúp cho người sử dụng không có cảm tưởng nhàm chán khi duyệt trang web, cũng như giúp cho người thiết kế web nhấn mạnh được những điểm quan trọng trong trang web của mình...

Trong Internet Explorer hỗ trợ việc thêm các hiệu ứng đặc biệt vào trang web mà chúng ta sẽ nghiên cứu ở dưới đây.

16.2. Lọc (*Filters*)

Thuộc tính *lọc* (*filter*) là một điều khiển được áp dụng cho một phần tử. Thuộc tính này được dùng cùng với thuộc tính *style* nhằm đạt được hiệu ứng như mong muốn. Các dạng lọc cho phép bạn sắp xếp dữ liệu và trình bày thông tin dưới nhiều dạng phong phú khác nhau.

Object STYLE = "filter:filtername(tham so của filter)"

Các dạng lọc yêu cầu một không gian hình chữ nhật để làm việc và chính vì thế có thể chỉ được áp dụng đối với các phần tử sau:

Chương 16

Các hiệu ứng đặc biệt

1. <BODY>
2. <BUTTON>
3. <DIV>
4.
5. <INPUT>
6. <MARQUEE>
7.
8. <TABLE>
9. <TD>
10. <TEXTAREA>
11. <TFOOT>
12. <TH>
13. <THEAD>
14. <TR>

Chú ý: Các phần tử và <DIV> phải có chiều cao, độ rộng xác định hoặc vị trí tuyệt đối.

Có rất nhiều dạng lọc cho phép chúng ta áp dụng phù hợp trong từng hoàn cảnh khác nhau. Bảng dưới đây chỉ ra một vài dạng lọc có thể được áp dụng.

Bảng 16.1. Các dạng lọc có thể được áp dụng

Filter	Chức năng	Thuộc tính
Alpha	Quy định mức trong suốt cho một đối tượng.	Opacity, finishopacity, style, startx, starty, finishx, finishy
Blur	Làm cho đối tượng xuất hiện khi di chuyển với tốc độ cao.	Add, direction, strength
Chroma	Làm cho một màu chuyển sang trong suốt.	Color
DropShadow	Tạo ra một bóng đổ đậm cho đối tượng.	Color, offx, offy, positive

Chương 16

Các hiệu ứng đặc biệt

FlipH	Tạo ra ảnh phản chiếu theo phương ngang cho đối tượng.	None
FlipV	Tạo ra hình ảnh phản chiếu theo phương thẳng đứng cho đối tượng.	None
Glow	Tạo ra hiệu ứng phát quang xung quanh đối tượng.	Color, strength
Gray	Hiện thị đối tượng theo bóng của sắc độ xám.	None
Invert	Đảo ngược màu sắc, độ bão hoà màu, và độ sáng tối của đối tượng.	None
Light	Tạo ra hiệu ứng của một nguồn ánh sáng chiếu lên đối tượng.	addAmbient, addCone, addPoint, changeColor, ChangeStrength, clear, movelight.
Mask	Tạo ra một mặt nạ lọc trong suốt cho đối tượng.	Color
Shadow	Tạo bóng của đối tượng.	Color, direction

Chúng ta sẽ tìm hiểu kỹ một số dạng lọc hay được sử dụng.

16.2.1. Thuộc tính lọc *FlipV* và *FlipH*

Như đã mô tả hai thuộc tính này trong bảng trên:

- *FlipH*: Tạo ra một hình ảnh phản chiếu theo phương ngang cho đối tượng.
- *FlipV*: Tạo ra một hình ảnh phản chiếu theo phương thẳng đứng cho đối tượng.

Chúng ta sẽ quan sát ví dụ sau minh hoạ cho hai thuộc tính lọc này.

Ví dụ 16.1.

```
<HTML>
<HEAD>
  <TITLE>Su dụng thuộc tính lọc Flip</TITLE>
  <style type = "text/css">
    BODY { background-color: #CCFFCC }
    TABLE { font-size: 3em;
```

Chương 16

Các hiệu ứng đặc biệt

```
font-family: Arial, sans-serif;
background-color: #FFCCCC;
border-style: ridge ;
border-collapse: collapse }
TD { border-style: groove;
padding: 1ex }
</STYLE>
</HEAD>
<BODY>
<TABLE>
  <TR>
    <!--Thuộc tính lọc sử dụng được khai báo trong style -->
    <TD style = "filter: fliph">Text</TD>
    <TD>Text</TD>
  </TR>
  <TR>
    <!--Có thể có nhiều thuộc tính lọc được dùng tại một thời điểm -->
    <TD style = "filter: flipv fliph">Text</TD>
    <TD style = "filter: flipv">Text</TD>
  </TR>
</TABLE>
</BODY>
</HTML>
```

Ví dụ trên thực hiện tạo ra một bảng gồm 4 ô, trong mỗi ô có ghi chữ “Text” với những hình ảnh phản chiếu khác nhau.

- Trong ô đầu tiên việc áp dụng thuộc tính *FlipH* để tạo ra hình ảnh phản chiếu theo chiều ngang của chữ “Text”, được khai báo với câu lệnh:

```
<TD style = "filter: fliph">Text</TD>
```

Chương 16

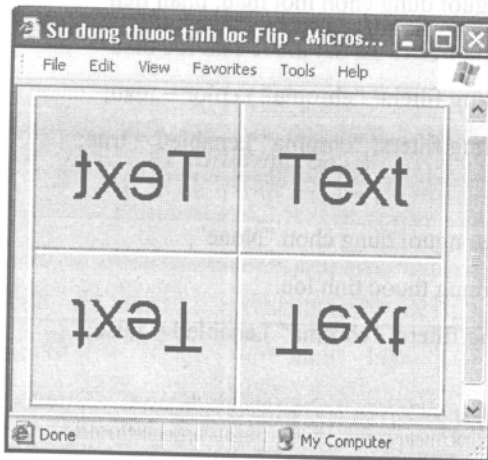
Các hiệu ứng đặc biệt

- Chúng ta cũng có thể sử dụng nhiều thuộc tính lọc tại một thời điểm cho một đối tượng. Các thuộc tính lọc khai báo cách nhau một dấu cách:

```
<TD style = "filter: flipv fliph">Text</TD>
```

- Hai thuộc tính lọc *flipV* và *flipH* được sử dụng trong ô thứ 3 của bảng. Kết quả là tạo ra một hình ảnh phản chiếu theo cả chiều ngang và chiều dọc của đối tượng đó.
- Trong ô cuối cùng, thuộc tính lọc *flipV* được áp dụng tạo ra một hình ảnh phản chiếu theo chiều dọc của chữ "Text".

Trong trình duyệt IE, kết quả của ví dụ trên như sau:



Hình 16.1. Minh hoạ cho thuộc tính lọc FlipV và FlipH.

16.2.2. Làm trong suốt cùng với thuộc tính Chroma

Thuộc tính lọc *Chroma* được dùng để làm cho màu của một đối tượng trở nên trong suốt. Một ví dụ điển hình là trong các file có đuôi *.gif* và *.png* thì màu nền được chuyển sang trong suốt để nó có thể phối hợp với màu sắc của các trang Web khác nhau tùy theo từng trang. Đặc tính này giúp chúng ta bố trí hình ảnh một cách phù hợp và tạo ra một hiệu ứng có hiệu quả.

Ví dụ sau minh hoạ cho thuộc tính lọc *Chroma*.

Ví dụ 16.2.

```
<HTML>
<HEAD>
<TITLE>Chroma Filter</TITLE>
```

Chương 16

Các hiệu ứng đặc biệt

```
<script language = "JavaScript">
  <!--
  function doi_mau( mau )
  {
    if ( mau )
    {
      // Neu nguoi dung chon mot mau,phan tich
      // sang gia tri hex va dat thuoc tinh loc mau
      chromaImg.filters( "chroma" ).color = mau;
      chromaImg.filters( "chroma" ).enabled = true;
    }
    else /// Neu nguoi dung chon "None",
    // Khong dung thuoc tinh loc.
      chromaImg.filters( "chroma" ).enabled = false;
  }
  // -->
</script>
</HEAD>
<BODY>
  <H1>Thuoc tinh loc chroma:</H1>
  <IMG id = "chromaImg" src = "trans.gif" style =
    "position: absolute; filter: chroma" alt =
    "Transparent Image" />
  <FORM action = "">
    <!-- su kien onchange duoc goi -->
    <!-- Khi mot lua chon duoc thay doi -->
    <select onchange = "doi_mau( this.value )">
```

Chương 16

Các hiệu ứng đặc biệt

```
<option value = "">None</option>
<option value = "#00FFFF">Cyan</option>
<option value = "#FFFF00">Yellow</option>
<option value = "#FF00FF">Magenta</option>
<option value = "#000000" selected = "selected">
    Black</option>
</select>
</FORM>
</BODY>
</HTML>
```

Trong ví dụ trên ta đã sử dụng thuộc tính lọc *chroma* cho một ảnh với bốn màu. Tất cả màu trong đối tượng sẽ hóa thành trong suốt.

- Dòng lệnh sau dùng để đặt thuộc tính lọc động, giá trị màu ở đây do người dùng chọn từ một danh sách xổ xuống.

```
chromaImg.filters( "chroma" ).color = mau;
```

- Dòng lệnh sau sẽ mở thuộc tính lọc *chroma*:

```
chromaImg.filters( "chroma" ).enabled = true;
```

- Nếu người sử dụng chọn giá trị *None* trong danh sách thì thuộc tính lọc *chroma* sẽ bị ẩn đi:

```
chromaImg.filters( "chroma" ).enabled = false;
```

- Như vậy chúng ta đã đặt thuộc tính lọc *chroma* cho màu tương ứng có trong bức ảnh. Để kích hoạt chúng ta sử dụng sự kiện *onchange* khi giá trị của form thay đổi do người dùng chọn:

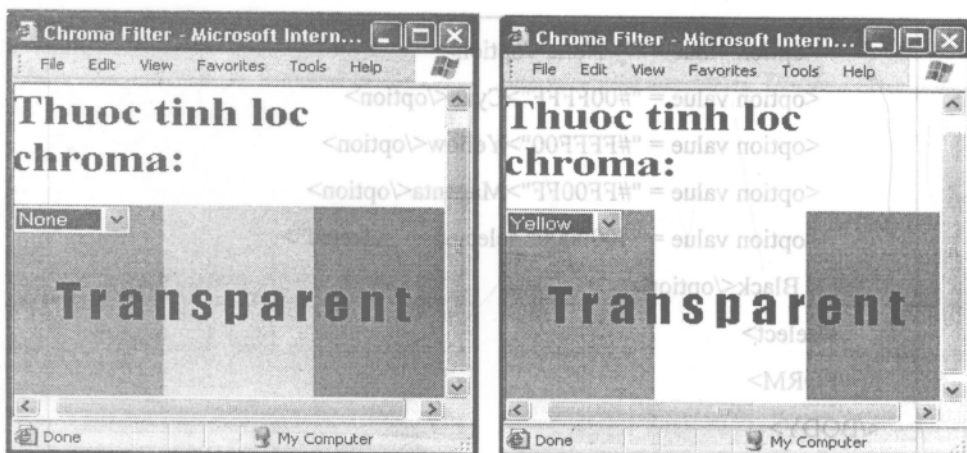
```
<select onchange = "doi_mau( this.value )">
```

Và khi đó thuộc tính lọc *chroma* thực hiện khiến tất cả các màu tương ứng trong đối tượng trùng với giá trị màu được chọn sẽ trở nên trong suốt.

Hình minh họa dưới đây chỉ ra hai trường hợp khi người dùng chọn trong danh sách màu giá trị *None* và giá trị *yellow*:

Chương 16

Các hiệu ứng đặc biệt



Hình 16.2. Minh họa cho tính lọc chroma.

Trong hình minh họa trên ta thấy rằng tất cả màu vàng trong đối tượng ảnh đầu đã trở nên trong suốt khi chúng ta chọn giá trị màu là *yellow*.

16.2.3. Hiệu ứng cùng với thuộc tính lọc Mask

Hiệu ứng này tạo ra một mặt nạ lọc trong suốt cho đối tượng. Với thuộc tính lọc này có thể cho phép chúng ta tạo ra một ảnh nền cho một đối tượng. Nền ảnh tại phần từ đó sẽ trở nên trong suốt và đường viền bao bọc xung quanh ảnh vẫn giữ nguyên. Sau đây là một ví dụ minh họa cho thuộc tính lọc *Mask*.

Ví dụ 16.3.

Trong ví dụ này chúng ta thao tác với một ảnh có tên là *gradient.gif*, khi chúng ta dùng thẻ *H1* để in một dòng chữ trên ảnh và đồng thời sử dụng thuộc tính lọc *mask* thì dòng chữ sẽ được ghi lên trên tấm ảnh và tại phần nền thể hiện dòng chữ đó sẽ trở nên trong suốt, các phần xung quanh dòng chữ sẽ không bị thay đổi tạo nên một khung “rắn” cho dòng chữ. Kết quả minh họa xem hình 16.3.

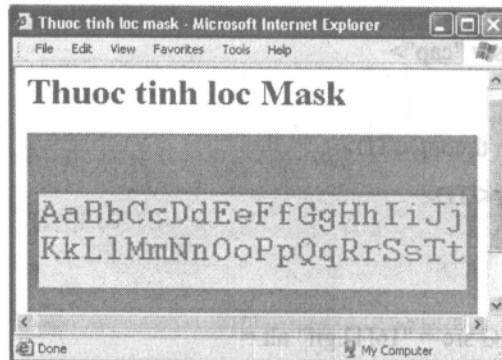
```
<HTML>
<HEAD>
<TITLE>Thuộc tính lọc Mask</TITLE>
</HEAD>
<BODY>
<H1>Thuộc tính lọc Mask</H1>
```

Chương 16

Các hiệu ứng đặc biệt

```
<DIV style = "position: absolute; top: 125; left: 20;  
filter: mask( color = #CCFFFF )" >  
  
<h1 style = "font-family: Courier, monospace">  
AaBbCcDdEeFfGgHhIiJj<br />  
KkLlMmNnOoPpQqRrSsTt  
  
</H1>  
</DIV>  
  
<IMG src = "gradient.gif" width = "400" height = "200"  
alt = "anh cung voi hieu ung" />  
  
</BODY>  
</HTML>
```

Lúc này chúng ta sẽ thấy hiệu ứng của thuộc tính lọc này như hình vẽ sau:



Hình 16.3. Minh họa cho tính lọc Mask.

16.2.4. Kết hợp các thuộc tính lọc ảnh: invert, gray và xray

Kết hợp các thuộc tính này với nhau cho ta những hiệu ứng rất đặc biệt khi thao tác với một đối tượng ảnh. Như đã giới thiệu ở trên, chức năng của từng thuộc tính lọc này như sau:

- **Invert:** Đảo ngược màu sắc, độ bão hoà màu, và độ sáng tối của đối tượng. Cụ thể là vùng sáng sẽ trở thành vùng tối và ngược lại.
- **Gray:** Hiển thị đối tượng theo bóng của sắc độ xám.

Chương 16

Các hiệu ứng đặc biệt

- *Xray*: hiển thị ngược so với Gray.

Ví dụ sau minh hoạ cho cả 3 thuộc tính trên khi thao tác với một ảnh có tên là *16[1].gif*.

Ví dụ 16.4.

```
<HTML>
<HEAD>
  <TITLE>Misc. Image filters</TITLE>
  <STYLE type = "text/css">
    .cap { font-weight: bold;
          background-color: #DDDDAA;
          text-align: center }
  </STYLE>
</HEAD>
<BODY>
  <TABLE class = "cap">
    <TR>
      <TD>Bình thường</TD>
      <TD>Gray</TD>
    </TR>
    <TR>
      <TD><IMG src = "16[1].gif" alt =
        "Hien o che do thuong" /></TD>
      <TD><IMG src = "16[1].gif" style = "filter: gray"
        alt = "dung gray"/>
      </TD>
    </TR>
    <TR>
      <TD>Xray</TD>
```

Chương 16

Các hiệu ứng đặc biệt

```
<TD>Invert</TD>
</TR>
<TR>
  <TD><IMG src = "16[1].gif" style = "filter: xray"
    alt = "dung xray"/>
  <TD><IMG src = "16[1].gif" style = "filter: invert"
    alt = "dung invert"/>
  <TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Chúng ta áp dụng từng thuộc tính lọc cho đối tượng ảnh. Tạo ra một bảng gồm 4 ô để thể hiện cho mỗi trường hợp.

- Trong ô đầu tiên, ảnh được biểu diễn ở chế độ bình thường, không sử dụng thuộc tính lọc nào:

```
<TD><IMG src = "16[1].gif" alt = "Hien o che do thuong" /></TD>
```

- Trong ô thứ hai sử dụng thuộc tính lọc *Gray*

```
<TD><IMG src="16[1].gif"style="filter: gray"alt="dung Gray"/>
```

- Trong ô thứ 3 sử dụng thuộc tính lọc *xray*:

```
<TD><IMG src="16[1].gif" style="filter: xray"alt="dung xray"/>
```

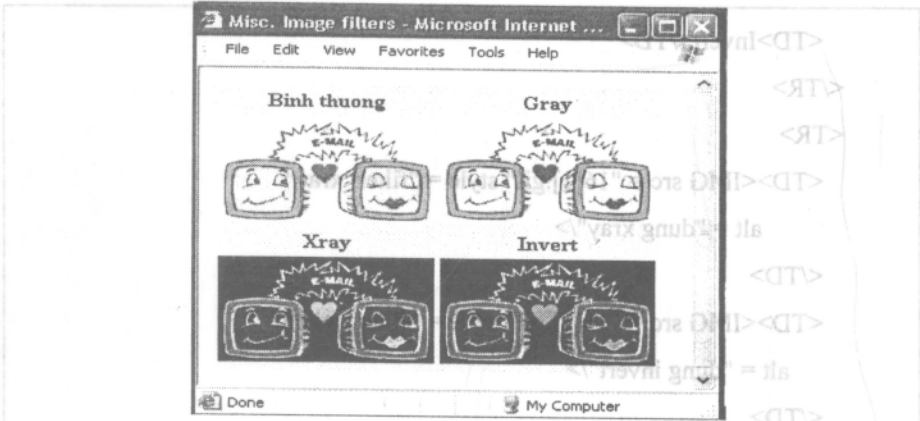
- Trong ô thứ 4 sử dụng thuộc tính *invert* cho ảnh:

```
<TD><IMG src="16[1].gif" style="filter: invert"alt = "dung invert"/>
```

Hình minh hoạ sau sẽ chỉ rõ cho chúng ta thấy hiệu ứng của các thuộc tính lọc trên:

Chương 16

Các hiệu ứng đặc biệt



Hình 16.4. Minh họa cho tính lọc invert, gray và xray.

16.2.5. Thuộc tính lọc Shadow

Sử dụng thuộc tính *Shadow* cho phép chúng ta thể hiện chữ theo không gian 3 chiều. Có hai dạng thuộc tính lọc *Shadow*:

- *Drop Shadow*: làm cho phần tử xuất hiện nổi trên trang web và tạo bóng.
- *Shadow*: tạo bóng trên cùng mặt phẳng của phần tử.

Ta sẽ minh họa cho từng dạng thuộc tính lọc *shadow* qua các hiệu ứng trong trình duyệt IE.

Ví dụ 16.5. Chúng ta sẽ in ra một dòng chữ và sau đó sử dụng thuộc tính lọc *Drop Shadow* để tạo thêm hiệu ứng cho dòng chữ.

```
<HTML>

<HEAD>

<TITLE>Thuoc tinh loc drop shadow</TITLE>

</HEAD>

<BODY>

<H1><CENTER><FONT color = red size = 5>Drop
Shadows</FONT></CENTER></H1>

<DIV STYLE = "position:absolute; top:50; left:100;
```

Chương 16

Các hiệu ứng đặc biệt

```
filter:dropshadow(color = limegreen, offx = 4, offy = 4, positive = 1)">
```

```
<BR><BR>
```

```
<P style = "font-size:20; color:hotpink">sử dụng tính drop shadows</P>
```

```
</DIV>
```

```
</BODY>
```

```
</HTML>
```

- Tính lọc *drop shadow* được khai báo ở dòng lệnh:

```
<DIV STYLE = "position:absolute; top:50; left:100; filter:dropshadow(color =  
limegreen, offx = 4, offy = 4, positive = 1)">
```

trong đó các tham số cần thiết của tính lọc này là: *color* chỉ ra màu phần đổ bóng được áp dụng cho dòng chữ; *Offx, offy*: chỉ ra vị trí sẽ đổ bóng trong trình duyệt.



Hình 16.5. Sử dụng Drop Shadow.

Ví dụ 16.6.

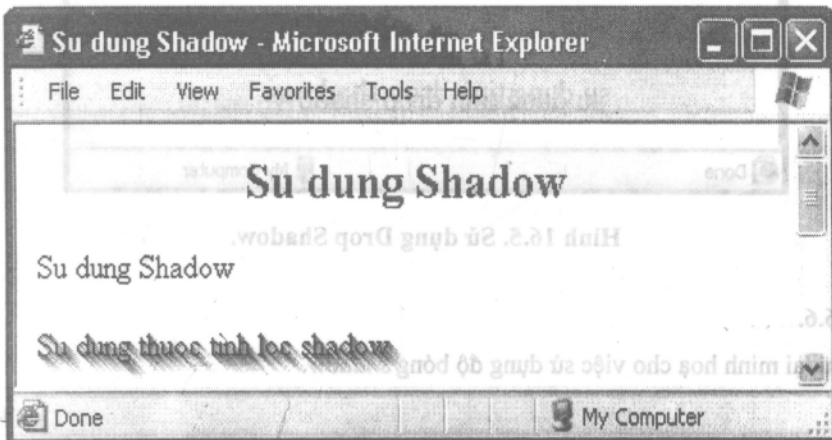
Ví dụ sau lại minh họa cho việc sử dụng độ bóng *shadow*.

Chương 16

Các hiệu ứng đặc biệt

```
<HTML>
  <HEAD>
    <TITLE>Su dung Shadow</TITLE>
  </HEAD>
  <BODY>
    <H1><CENTER><FONT color = red size = 5>Su dung
    Shadow</FONT></CENTER></H1>
    <P>Su dung Shadow </P>
    <SPAN STYLE = "width:200; height:200;
    filter:shadow(color = limegreen, direction = 135)">
      Su dung thuoc tinh loc shadow
    </SPAN>
  </BODY>
</HTML>
```

Kết quả trong trình duyệt như sau:



Hình 16.6. Sử dụng Shadow.

Chương 16

Các hiệu ứng đặc biệt

- Thuộc tính *shadow* được khai báo trong câu lệnh:

```
<SPAN STYLE = "width:200; height:200; filter:shadow(color = limegreen, direction  
= 135)">
```

Su dụng thuộc tính *shadow*

```
</SPAN>
```

- Các tham số cần chú ý của thuộc tính này là: *Color*: chỉ ra màu sẽ được đổ bóng; *Direction*: có giá trị từ 0 đến 315 xác định chiều hay phương hướng sẽ được đổ bóng của đối tượng. Ở ví dụ này, màu đổ bóng là màu xanh, và hướng hay chiều đổ bóng tương ứng với thuộc tính *direction = 135*.

16.2.6. Thuộc tính *Blur* (bóng mờ)

Những thuộc tính sau của thuộc tính *blur* được thiết lập nhằm tạo ra hiệu ứng mờ cho đối tượng:

- Strength* - xác định chiều dài của bóng mờ tính theo pixel.
- Add* - đối số này thực hiện có thêm hình ảnh gốc nằm dưới hình ảnh có bóng mờ và đã được lọc.
- Direction* - đối số này xác định hướng làm mờ. Một số giá trị cho phép là, 180 = dưới (*down*) và 90 = phải (*right*).

Để hiểu rõ về thuộc tính này chúng ta sẽ nghiên cứu ví dụ sau.

Ví dụ 16.7.

```
<HTML>  
  <HEAD>  
    <TITLE>Thuoc tinh loc Blur</TITLE>  
  </HEAD>  
  <BODY BGCOLOR=lavender TEXT=aqua>  
    <DIV STYLE = "position:absolute; top:60; left:100;  
      filter:blur(add=0,direction=270,strength=30)">  
      <IMG src="16[1].JPEG">  
    </DIV>  
  </BODY>  
</HTML>
```


Chương 16

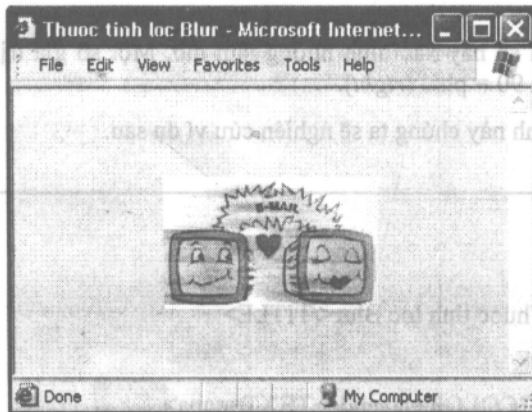
Các hiệu ứng đặc biệt

Kết quả ảnh sẽ bị làm mờ trong trình duyệt



Hình 16.7. Minh họa tính lọc Blur.

Nếu như chúng ta thay đổi giá trị của *Add* thành 1, thì ảnh sẽ được khôi phục lại:



Hình 16.8. Khôi phục lại ảnh từ thuộc tính lọc Blur.

16.2.7. Thuộc tính lọc trong suốt (alpha filter)

Thuộc tính lọc *alpha* quy định mức độ trong suốt của một đối tượng. Thuộc tính *style* của lọc *alpha* có thể tồn tại một trong bốn giá trị sau:

- 0 - Đồng dạng (*uniform*)
- 1 - Tuyến tính (*linear*)

Chương 16

Các hiệu ứng đặc biệt

- 2 - Bán kính (*radial*)
- 3 - Vuông (*rectangular*)

Trong ví dụ sau sẽ minh họa cho các thuộc tính *style* trong tính lọc alpha.

Ví dụ 16.8.

```
<HTML>
<HEAD>
  <TITLE>Thuộc tính lọc Alpha</TITLE>
  <script language= "JavaScript">
    <!--
    function run()
    {
      pic.filters( "alpha" ).opacity = opacityButton.value;
      pic.filters( "alpha" ).finishopacity = opacityButton2.value;
      pic.filters( "alpha" ).style = styleSelect.value;
    }
    // -->
  </script>
</HEAD>
<BODY>
  <DIV id = "pic"
    style = "position: absolute; left:0; top: 0;
      filter: alpha( style = 2, opacity = 100, finishopacity = 0 )">
    <IMG src = "flag.gif" alt = "Flag" />
  </DIV>
  <TABLE style = "position: absolute; top: 250; left: 0;
    background-color: #CCFFCC" border = "1">
    <TR>
```

Chương 16
Các hiệu ứng đặc biệt

```
<TD>Opacity (0-100):</TD>
<TD><input type = "text" id = "opacityButton"
      size = "3" maxlength = "3" value = "100" /></TD>
</TR>
<TR>
<TD>FinishOpacity (0-100):</TD>
<TD><input type = "text" id = "opacityButton2"
      size = "3" maxlength = "3" value = "0" /></TD>
</TR>
<TR>
<TD>Chon kieu:</TD>
<TD><select id = "styleSelect">
      <option value = "1">Tuyen tinh</option>
      <option value = "2" selected = "selected">
        Vong tron</option>
      <option value = "3">Vuong</option>
    </select></TD>
</TR>
<TR> <TD align = "center" colspan = "2">
      <input type = "button" value = "Thay doi!"
        onclick = "run()" />
    </TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Chương 16

Các hiệu ứng đặc biệt

Chúng ta hãy cùng phân tích ví dụ trên. Đầu tiên chúng ta xây dựng một hàm *run()* để đặt các tham số cho thuộc tính lọc *alpha* với đối tượng ảnh có *ID = pic*:

```
function run()
{
    pic.filters( "alpha" ).opacity = opacityButton.value;
    pic.filters( "alpha" ).finishopacity = opacityButton2.value;
    pic.filters( "alpha" ).style = styleSelect.value;
}
```

- Thuộc tính *opacity* quyết định tỷ lệ độ dốc bắt đầu. Giá trị của thuộc tính này do người dùng nhập vào từ hộp text có *id* là *opacityButton*.
- Thuộc tính *finishopacity* chỉ ra tỷ lệ độ dốc kết thúc. Giá trị của thuộc tính này do người dùng nhập vào từ hộp text có *id* là *opacityButton2*.
- Thuộc tính *style* sẽ chọn ra kiểu *style* có trong 4 kiểu trên để tạo hiệu ứng trong suốt. Style này sẽ được người dùng chọn từ danh sách có *id* là *styleSelect*.

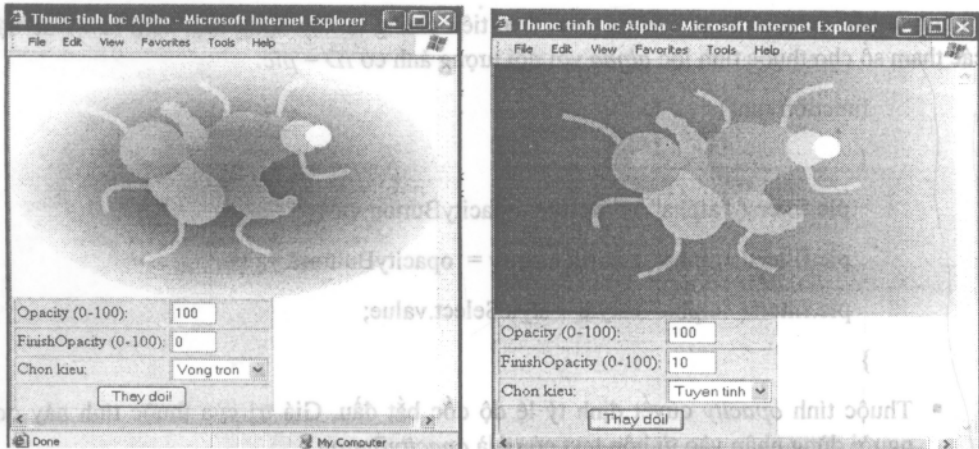
Mỗi khi người dùng nhập giá trị vào các hộp text box và nhấn vào nút “thay đổi!” thì ngay lập tức hàm *run()* sẽ được kích hoạt và các giá trị của thuộc tính lọc *alpha* sẽ được xác lập. Khi đó hiệu ứng tương ứng với các tham số đó sẽ được thể hiện. Trong ví dụ trên, ban đầu chúng ta khởi tạo các tham số cho tính lọc *alpha* như sau:

```
<DIV id = "pic"
    style = "position: absolute; left:0; top: 0;
    filter: alpha( style = 2, opacity = 100, finishopacity = 0 )">
    <IMG src = "flag.gif" alt = "Flag" />
</DIV>
```

style = 2 nghĩa là theo kiểu vòng tròn; *opacity*: độ dốc bắt đầu là 100 và kết thúc là 0. Khi thay đổi các thông số này thì sẽ thấy hiệu ứng thay đổi tương ứng trong trình duyệt IE.

Chương 16

Các hiệu ứng đặc biệt



Hình 16.9. Minh họa tính lọc alpha.

16.2.8. Thuộc tính lọc Wave

Thuộc tính lọc *wave* cho phép chúng ta tạo nên hiệu ứng cho một dòng chữ hay một đối tượng ảnh chuyển động dạng hình sóng. Các thuộc tính của tính lọc *wave* bao gồm:

- **Add**: thêm một bản sao của dòng chữ hoặc hình ảnh ở ngay phía dưới đối tượng áp dụng thuộc tính lọc. Tham số này chỉ sử dụng hiệu quả khi thao tác với đối tượng là ảnh.
- **Freq**: Xác định tần số của sóng sẽ áp dụng cho đối tượng. Có nghĩa là xác định tốc độ chuyển động của đối tượng.
- **Phase**: Chỉ ra độ lệch pha của sóng. Tăng thuộc tính này lên chỉ thay đổi về hình dạng sóng mà thôi.
- **Strength**: chỉ ra độ rộng của bước sóng được áp dụng cho đối tượng.

Ví dụ sau minh họa cho thuộc tính lọc *wave* này:

Ví dụ 16.9.

```
<HTML>
<HEAD>
  <title>Thuoc tinh loc Wave</title>
  <script language = "JavaScript">
    <!--
```

Chương 16

Các hiệu ứng đặc biệt

```
var wavePhase = 0;
function bat_dau()
{
    window.setInterval( "wave()", 5 );
}
function wave()
{
    wavePhase++;
    flag.filters( "wave" ).phase = wavePhase;
}
// -->
</script>
</HEAD>
<BODY onload = "bat_dau();">
    <SPAN id = "flag"
        style = "align: center; position: absolute;
        left: 30; padding: 15;
        filter: wave(add = 0, freq = 1, phase = 0, strength = 10); font-size: 2em">
        Chao mung ban den voi trang web cua chung toi
    </SPAN>
</BODY>
</HTML>
```

Pha của sóng sẽ được tăng lên sau 5 giây. Hình ảnh sau minh hoạ cho ví dụ trên trong trình duyệt web IE.

Chương 16

Các hiệu ứng đặc biệt



Hình 16.10. Minh họa cho tính lọc wave.

16.3. Chuyển tiếp (Transitions)

Chuyển tiếp là các dạng lọc xác định một khoảng thời gian để thay đổi từ trạng thái hiện thời sang một trạng thái khác. Có hai hiệu ứng chuyển tiếp bạn có thể sử dụng:

- Chuyển tiếp pha trộn (*blend*): được dùng để tạo ra hiệu ứng mờ dần hoặc mờ dần.
- Chuyển tiếp khám phá (*reveal*): được dùng để tạo ra các hiệu ứng như *box in*, *box out*, liên kết ngang và liên kết dọc.

Bạn có thể áp dụng một vài hiệu ứng chuyển tiếp sau:

Bảng 16.2. Các hiệu ứng chuyển tiếp

Hiệu ứng chuyển tiếp	Mã
Box in	0
Box out	1
Wipe right	6
Wipe left	7

Chương 16

Các hiệu ứng đặc biệt

Vertical blinds	8
Horizontal blinds	9
Random	23

Các thuộc tính chuyển tiếp có thể được xác định trong thẻ *<META>*. Thẻ *<META>* chỉ được hiển thị trong phần tử *<HEAD>*. Bạn phải chỉ ra thuộc tính CONTENT. Sau đó bạn có thể chỉ ra thuộc tính HTTP-EQUIV hoặc thuộc tính NAME (bạn không thể dùng hai thuộc tính này cùng một lúc).

Ví dụ sau minh họa cho một hiệu ứng chuyển tiếp.

Ví dụ 16.10.

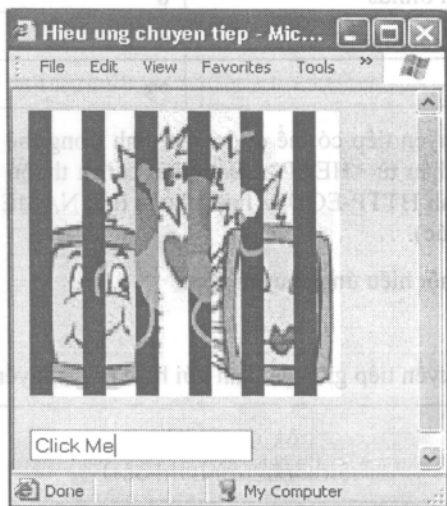
Ví dụ này sẽ cho ta sự chuyển tiếp giữa hai ảnh với hiệu ứng chuyển đổi theo chiều dọc.

```
<HTML>
  <HEAD>
    <TITLE>Hiệu ứng chuyển tiếp</TITLE>
    <SCRIPT LANGUAGE = "JavaScript">
      function an()
      {
        thePIC.filters.item(0).Apply();
        thePIC.src="flag.gif";
        thePIC.filters.item(0).Play();
      }
    </SCRIPT>
  </HEAD>
  <BODY BGCOLOR=lavender>
    <IMG ID=thePIC width=200 height=200 src="c:\forest.bmp"
      style="filter:revealTrans(Duration=4.1, Transition=8)">
    <BR><BR>
    <INPUT type = button value = "Click Me" onClick = "an()">
  </BODY>
</HTML>
```


Chương 16

Các hiệu ứng đặc biệt

Hãy chạy chương trình trên trong trình duyệt web IE và xem kết quả.



Hình 16.11. Minh họa cho một hiệu ứng chuyển tiếp.

16.4. Kết chương

Nội dung chương này giới thiệu về tác dụng của các hiệu ứng đặc biệt trong trang web. Bạn đã được giới thiệu về các thuộc tính lọc, làm mờ và chuyển tiếp. Nắm rõ được các hiệu ứng này sẽ giúp chúng ta thể hiện được ý tưởng của mình thông qua mỗi trang web làm cho trang web trở nên sống động hơn. Trong chương sau chúng ta sẽ tìm hiểu cách liên kết dữ liệu với một trang Web.

Chương 17

Liên kết dữ liệu

Nội dung của chương:

- Tìm hiểu về cách hiển thị bảng trên một trang web.
- Cách sử dụng Tabular Data Control.
- Thao tác dữ liệu trên một trang web.

17.1. Giới thiệu

Trước kia tất cả những gì bạn có thể làm chỉ là hiển thị dữ liệu trong trang web còn việc thao tác dữ liệu được điều khiển thông qua các *scripts* ở phía server. Với HTML động, mọi thứ đã thay đổi. HTML động hỗ trợ một tính năng gọi là *liên kết dữ liệu* giúp chúng ta có thể thao tác với dữ liệu tại phía Client. Liên kết dữ liệu yêu cầu hai thành phần:

- **Đối tượng nguồn dữ liệu:** cung cấp đường liên kết tới nguồn dữ liệu. Đối tượng nguồn dữ liệu được tạo ra như những thành phần ActiveX hoặc Java applets. Những đối tượng này được dùng để cung cấp nối kết giữa file dữ liệu và một khuôn mẫu trên trang web.
- **Các đuôi mở rộng HTML:** là các phần tử HTML được dùng để nối các phần tử trong trang web với đối tượng nguồn dữ liệu.

Dữ liệu được truy xuất từ file ghép với một *mẫu* để tạo ra sự hiển thị trên web. Điều này có nghĩa là dữ liệu không kết hợp với bản thân HTML. Trong chương này bạn sẽ được giới thiệu về *Tabular Data Control (TDC)*, một điều khiển được dùng để hiển thị dữ liệu từ một file dữ liệu dạng bảng. Khi dữ liệu hiển thị, bạn có thể sắp xếp và lọc nó ở phía client thay vì phải quay trở lại máy chủ. *Hạn chế duy nhất là người dùng không thể cập nhật file dữ liệu.*

Nguồn dữ liệu là file được định phân cách bằng dấu phẩy mặc định. Tuy nhiên, bạn có thể dùng các thuộc tính để quy định các dấu phân cách. Dòng đầu tiên của file là các cột

Chương 17

Liên kết dữ liệu

mặc định là Column 1 (cột 1), Column 2 (cột 2) và tiếp tục như thế. Bạn cũng có thể chỉ ra kiểu dữ liệu của trường.

TDC cung cấp một số bảo mật có sẵn. File dữ liệu và trang web muốn truy cập vào file dữ liệu cần phải nằm trên cùng một *server*. Chính vì thế, nếu *script* truy cập vào các file dữ liệu trên một *server* khác thì TDC sẽ không thực hiện chức năng của nó. Để khắc phục hạn chế này, người tạo ra file dữ liệu sẽ phải cung cấp quyền truy cập cho người sử dụng máy ở trong các tên miền (*domain*) khác. Quyền truy cập phải là dòng đầu tiên trong file dữ liệu và có cú pháp như sau: `@!allow_domains=domain names`. Các tên miền phải được ngăn cách với nhau bởi một dấu chấm phẩy (;).

Sau đây chúng ta sẽ nghiên cứu các cách để hiển thị dữ liệu lên một trang web.

17.2. Sử dụng Tabular Data Control (TDC)

Trước khi đi vào kết nối và hiển thị dữ liệu, chúng ta tìm hiểu về cấu trúc một file dữ liệu đơn giản.

17.2.1. File dữ liệu

Một file dữ liệu đơn giản bao gồm:

- Hàng tiêu đề: chỉ ra tên của các cột có trong file dữ liệu.
- Thuộc tính *TextQualifier*: dùng để chỉ ra ký tự đúng xung quanh chuỗi văn bản trong trường hợp có thể có một trường dữ liệu chứa các dấu ngoặc kép đôi thì HTML động sẽ dễ bị nhầm lẫn khi đọc nội dung của file dữ liệu. Vì vậy nếu bạn cần chỉ ra một dấu phân cách khác cho chuỗi văn bản thì bạn có thể dùng thuộc tính *TextQualifier* khi khai báo `<OBJECT>`.

`<PARAM NAME = "TextQualifier" VALUE = delimiter>`

hoặc thông qua script:

`Filename.TextQualifier = delimiter`

- Thuộc tính *FieldDelimiter*: Nếu bạn cần dùng một file dữ liệu mà trong đó các trường dữ liệu không được phân cách bởi dấu phẩy thì bạn sẽ phải chỉ ra dấu phân cách để TDC có thể định danh nó trong khi đọc file. Thuộc tính *FieldDelimiter* được dùng để thiết lập dấu phân cách trường khi khai báo `OBJECT`.

`<PARAM NAME = "FieldDelim" VALUE = delimiter character>`

Ví dụ:

`<OBJECT CLASSID = "clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"`

`ID = employees>`

Chương 17

Liên kết dữ liệu

```
<PARAM NAME = "DaTaURL" Value = "employees.csv">  
<PARAM NAME = "UseHeader" value ="True">  
<PARAM NAME = "FieldDelim" Value = "&#09;">  
</OBJECT>
```

Ví dụ sau minh hoạ cho một file dữ liệu điển hình:

```
@ColorName@|@ColorHexRGBValue@  
@aqua@|@#00FFFF@  
@black@|@#000000@  
@blue@|@#0000FF@  
@fuchsia@|@#FF00FF@  
@gray@|@#808080@  
@green@|@#008000@  
@lime@|@#00FF00@  
@maroon@|@#800000@  
@navy@|@#000080@  
@olive@|@#808000@
```

Hình 17.1. File dữ liệu điển hình.

17.2.2. Hiển thị dữ liệu

Để hiển thị dữ liệu trong một bảng trên trang web, bạn sẽ phải thực hiện những công việc dưới đây:

- Tham chiếu đối tượng nguồn dữ liệu. Phần tử **<OBJECT>** được dùng để chỉ ra ID của đối tượng nguồn dữ liệu và gán các tham số được yêu cầu để thiết lập tham chiếu đến nguồn dữ liệu.
- Định danh *Tabular Data Control*. Điều khiển này được dùng để truy cập dữ liệu từ các file phân cách bằng dấu phẩy. Lệnh CLASSID được dùng để xác định điều khiển. CLASS ID (ID của lớp) bằng 333C7BC4-460F-11D0-BC04-0080C7055A83.
- Các tham số.

Chương 17

Liên kết dữ liệu

- PARAM đầu tiên là *DataURL*. Nó chỉ ra tên và URL của file nguồn dữ liệu. Trang và nguồn dữ liệu phải nằm trên cùng một server. Nếu trị giá của *DataURL* không giống với trị giá của server chứa trang web thì TDC sẽ không thực hiện chức năng của nó.
- PARAM thứ hai là *UseHeader*. Nó xác định dòng đầu tiên của file chứa tiêu đề hay không. Tiêu đề có thể là tên của các cột dữ liệu. Giá trị mặc định của *UseHeader* là *False*. Vì vậy nếu bạn có một *header*, bạn thay đổi giá trị của nó thành *True*.

```
<OBJECT CLASSID = "clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
ID = filename>
<PARAM NAME = "DataURL" Value = "filename.csv">
<PARAM NAME = "UseHeader" value = "True">
</OBJECT>
```

- Bước cuối cùng là liên kết phần tử HTML với dữ liệu trong file. Trong trường hợp này là phần tử *<TABLE>*. Phần tử *<TABLE>* được dùng để hiển thị dữ liệu. Đầu tiên bạn liên kết phần tử *<TABLE>* bằng cách dùng thuộc tính *DATASRC*. Rồi bạn phải liên kết mỗi ô trong bảng với các trường trong nguồn dữ liệu thông qua cách dùng thuộc tính *DATASRC*. Khi bảng được hiển thị, HTML động sẽ tự động thêm một hàng vào bảng cho mỗi bản ghi truy xuất từ file dữ liệu.

Ví dụ 17.1.

```
<HTML xmlns = "http://www.w3.org/1999/xhtml">
<HEAD>
  <TITLE>Ket noi du lieu</TITLE>
  <!-- Day la doi tuong de chen mot ActiveX control -->
  <!-- de dieu khien va phan tich du lieu cua chung ta-->
  <!-- Cac the cho control cac tham so bat dau-->
  <!-- chang han nh URL-->
  <OBJECT id = "Colors"
    classid = "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
    <param name = "DataURL" value =
      "file_date.txt" />
```

Chương 17

Liên kết dữ liệu

```
<param name = "UseHeader" value = "TRUE" />
<param name = "TextQualifier" value = "@" />
<param name = "FieldDelim" value = "|" />
</OBJECT>
<script language = "JavaScript">
  <!--
  var recordSet = Colors.recordset;
  function reNumber()
  {
    if ( !recordSet.EOF )
      recordNumber.innerText =
        recordSet.absolutePosition;
    else
      recordNumber.innerText = " ";
  }
  function forward()
  {
    recordSet.MoveNext();
    if ( recordSet.EOF )
      recordSet.MoveFirst();
    Mau.style.backgroundColor =
      colorRGB.innerText;
    reNumber();
  }
  // -->
</script>
</HEAD>
```

Chương 17

Liên kết dữ liệu

```
<BODY onload = "reNumber()" onclick = "forward()">
  <H1>Bang mau XHTML</H1>
  <H3>Bam chuoat de chuyen de recordset tiep theo</H3>
  <P><STRONG>Ten mau: </STRONG>
  <SPAN id = "colorId" style = "font-family: monospace"
    datasrc = "#Colors" datafld = "ColorName"></SPAN><BR />
  <STRONG>Gia tri mau RGB: </STRONG>
  <SPAN id = "colorRGB" style = "font-family: monospace"
    datasrc = "#Colors" datafld = "ColorHexRGBValue">
  </SPAN><BR />
  Dang hien thi Record thu:
  <SPAN id = "recordNumber" style = "font-weight: 900">
  </SPAN><BR />
  <SPAN id = "Mau" style = "background-color: aqua;
    color: 888888; font-size: 30pt">Color
  </SPAN></P>
</BODY>
</HTML>
```

Trong ví dụ trên, ta đã thao tác với file dữ liệu *file_date.txt*. File này chứa 2 trường: trường thứ nhất chỉ ra tên màu, trường thứ hai chỉ ra giá trị hex của màu. Cứ mỗi cặp thuộc tính (*tên màu, giá trị màu*) tạo ra một bản ghi (*recordset*). Chúng ta sẽ hiển thị lần lượt từng *recordset* từ đầu cho đến cuối file dữ liệu.

- Đầu tiên chúng ta tạo ra một đối tượng có *Id = colors*. Đối tượng này được chèn vào *Tabular Data Control*.

```
<OBJECT id = "Colors"
  classid = "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
```

- Thuộc tính *classid* được dùng để xác định điều khiển mới được thêm vào.
- Thẻ *Param* chỉ ra tham số cho đối tượng:

Chương 17

Liên kết dữ liệu

```
<param name = "DataURL" value = "file_date.txt" />
```

```
<param name = "UseHeader" value = "TRUE" />
```

<param name = "TextQualifier" value = "@" />: dùng kí tự @ để phân biệt giữa các trường dữ liệu

<param name = "FieldDelim" value = "|" />: chỉ ra dấu phân cách giữa các trường là '| '.

- Chúng ta khai báo một biến *recordSet* = *Colors.recordset* để duyệt các bản ghi có trong file dữ liệu.
- Phương thức *MoveNext* của đối tượng *recordset* dùng để duyệt đến bản ghi tiếp sau bản ghi hiện tại. Nếu như duyệt đến bản ghi cuối cùng thì trở về bản ghi đầu tiên.

```
if ( recordSet.EOF )
```

```
recordSet.MoveFirst();
```

- Dữ liệu trong đối tượng *Colors* sẽ được kết nối với một phần tử *span*:

```
<SPAN id = "colorId" style = "font-family: monospace"
```

```
datasrc = "#Colors" datafld = "ColorName"></SPAN><BR />
```

- Khi người sử dụng kích chuột, thì hàm *forward()* sẽ được gọi và thực hiện.

Kết quả của ví dụ trên trong trình duyệt IE như sau:



Hình 17.2 Kết nối dữ liệu sử dụng TDC.

17.3. Kết nối tới bảng

Bạn đã học cách sử dụng bảng để hiển thị văn bản và hình ảnh trên một trang web. Tuy nhiên bảng có thể được đưa vào sử dụng rộng rãi hơn. Bảng có thể được dùng để hiển thị dữ liệu từ một cơ sở dữ liệu. Bạn có thể liên kết một bảng với một file nguồn, khi trang web được hiển thị, nó sẽ truy xuất dữ liệu từ file nguồn và hiển thị nó trong bảng. Các hàng và các ô được tạo ra mang tính động để đáp ứng với dữ liệu trong cơ sở dữ liệu đảm bảo cho người dùng có thể xem thông tin cập nhật.

Kết nối dữ liệu với một bảng là điều quan trọng nhất trong kết nối dữ liệu. Cách kết nối này khác với cách kết nối ở trên. Các *recordset* được hiển thị trong hàng của một bảng.

Ví dụ 17.2.

```
<HTML>
<HEAD>
  <TITLE>su dung bang</TITLE>
  <OBJECT id = "Colors"
    classid = "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
    <param name = "DataURL" value =
      "file_date.txt" />
    <param name = "UseHeader" value = "TRUE" />
    <param name = "TextQualifier" value = "@" />
    <param name = "FieldDelim" value = "|" />
  </OBJECT>
</HEAD>
<BODY style = "background-color: darkseagreen">
  <H1>Ket noi du lieu toi <code>TABLE</code></H1>
  <TABLE datasrc = "#Colors" style = "border-style: ridge;
    border-color: darkseagreen;
    background-color: lightcyan">
    <THEAD>
```

Chương 17

Liên kết dữ liệu

```
<TR style = "background-color: mediumslateblue">
  <th>Ten mau</th>
  <th>Gia tri mau RGB</th>
</TR>
</THEAD>
<TBODY>
  <TR style = "background-color: lightsteelblue">
    <TD><span datafld = "ColorName"></span></TD>
    <TD><span datafld = "ColorHexRGBValue"
      style = "font-family: monospace"></span></TD>
  </TR>
</TBODY>
</TABLE>
</BODY>
</HTML>
```

Trong ví dụ này,

- Bắt đầu kết nối với bảng bằng thuộc tính *datasrc* :

```
<TABLE datasrc = "#Colors" style = "border-style: ridge;
  border-color: darkseagreen;
  background-color: lightcyan">
```
- Kết thúc kết nối để hiển thị dữ liệu bằng cách đặt thuộc tính *datafld* của thẻ *span*:

```
<TD><span datafld = "ColorName"></span></TD>
  <TD><span datafld = "ColorHexRGBValue"
    style = "font-family: monospace"></span></TD>
```

Dữ liệu trong file *data file.txt* sẽ được hiển thị trong bảng ở trong trình duyệt như sau:



Hình 17.3. Kết nối dữ liệu tới bảng.

17.4. Kết nối tới ảnh

Có rất nhiều thành phần của XHTML có khả năng kết nối dữ liệu. Điển hình là kết nối dữ liệu tới một thành phần ảnh. Sau khi thay đổi một *recordset* và cập nhật vào thuộc tính *src* của ảnh thì ta có thể hiển thị được dữ liệu.

Ví dụ 17.3.

Chẳng hạn với file *images.txt*: image

numbers/0.gif numbers/4.gif

numbers/1.gif numbers/5.gif

numbers/2.gif numbers/6.gif

numbers/3.gif numbers/7.gif

numbers/8.gif numbers/9.gif

File này chứa tên ảnh của các số từ 0 cho đến 9. Chúng ta hãy cùng thao tác với file dữ liệu này trong ví dụ sau.

```
<HTML>
<HEAD>
  <TITLE>Ket noi toi mot anh</TITLE>
  <OBJECT id = "Images"
    classid = "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
    <param name = "DataURL" value = "images.txt" />
    <param name = "UseHeader" value = "True" />
  </OBJECT>
  <script language = "JavaScript">
    <!--
    recordSet = Images.recordset;
    function move( whereTo )
    {
      switch( whereTo ) {
        case "first":
          recordSet.MoveFirst();
          break;
        case "previous":
          recordSet.MovePrevious();
          if ( recordSet.BOF )
            recordSet.MoveLast();
          break;
        case "next":
          recordSet.MoveNext();
          if ( recordSet.EOF )
            recordSet.MoveFirst();
          break;
```

Chương 17

Liên kết dữ liệu

```
        case "last":
            recordSet.MoveLast();
            break;
    }
}
// -->
</script>
</HEAD>
<BODY>
    <IMG datasrc = "#Images" datafld = "image" alt = "Image"
        style = "position: relative; left: 45px" /><BR />
    <input type = "button" value = "First"
        onclick = "move( 'first' );" />
    <input type = "button" value = "Previous"
        onclick = "move( 'previous' );" />
    <input type = "button" value = "Next"
        onclick = "move( 'next' );" />
    <input type = "button" value = "Last"
        onclick = "move( 'last' );" />
</BODY>
</HTML>
```

- *Recordset* sử dụng trong ví dụ này là tập hợp các ảnh.

`recordSet = Images.recordset;`

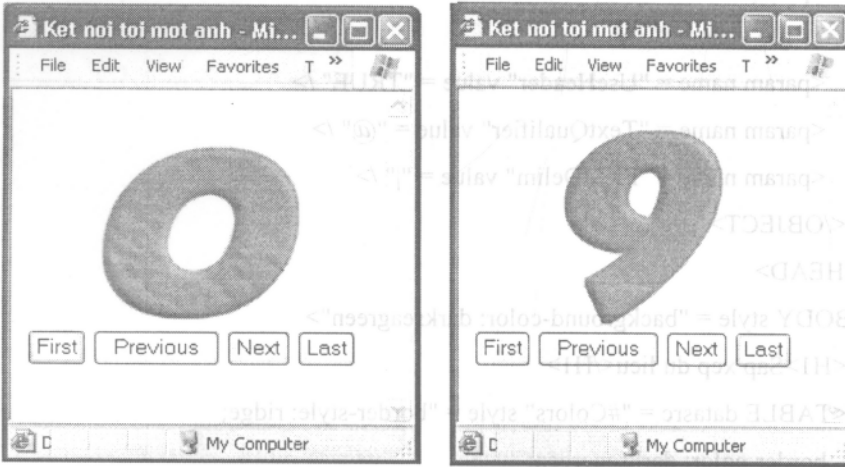
- Tương tự như ví dụ trên người dùng có thể chọn: bản ghi đầu tiên (*first*), bản ghi cuối cùng (*last*), bản ghi trước bản ghi hiện tại (*previous*), bản ghi sau bản ghi hiện tại (*next*). Hàm *move(whereto)* sẽ cho phép ta làm điều này thông qua cấu trúc *case*.
- Kết nối dữ liệu tới ảnh sẽ tự động cập nhật ảnh khi người dùng lựa chọn hiển thị một ảnh mới.

Chương 17

Liên kết dữ liệu

```
<IMG datasrc = "#Images" datafld = "image" alt = "Image" style = "position: relative; left: 45px" /><BR />
```

Kết quả hiển thị trong trình duyệt:



Hình 17.4. Minh họa cho kết nối dữ liệu tới ảnh.

17.5. Thao tác dữ liệu

Người dùng có thể muốn xem các thông tin được chọn lọc trong file dữ liệu. Một khi dữ liệu được truy xuất từ file, bạn có thể thao tác trên dữ liệu tại máy khách, sắp xếp và lọc dữ liệu trong bảng HTML trên trang web mà không cần phải quay trở lại server.

17.5.1. Sắp xếp dữ liệu

Bạn có thể sắp xếp dữ liệu ở các cột sau khi bảng đã hiển thị trong trang web. Bạn có thể sắp xếp dữ liệu theo trật tự tăng dần hoặc giảm dần. Để lọc dữ liệu trong cột theo trật tự giảm dần, bạn phải thêm một dấu trừ (-) trước tên của cột được chỉ ra.

Trong ví dụ sau, chúng ta sẽ minh họa cho các cách sắp xếp với dữ liệu lấy từ file "file_data.txt" gồm các tên màu và giá trị màu.

Ví dụ 17.4.

```
<HTML>
<HEAD>
<TITLE>Sap xep du lieu</TITLE>
```

Chương 17

Liên kết dữ liệu

```
<OBJECT id = "Colors"
  classid = "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
  <param name = "DataURL" value =
    "file_date.txt" />
  <param name = "UseHeader" value = "TRUE" />
  <param name = "TextQualifier" value = "@" />
  <param name = "FieldDelim" value = "|" />
</OBJECT>
</HEAD>
<BODY style = "background-color: darkseagreen">
  <H1>Sap xep du lieu</H1>
  <TABLE datasrc = "#Colors" style = "border-style: ridge;
    border-color: darkseagreen;
    background-color: lightcyan">
    <caption>Chon kieu sap xep:
    <select onchange = "Colors.Sort = this.value;
      Colors.Reset();">
      <option value = "ColorName">Ten mau (Tang dan)
        </option>
      <option value = "-ColorName">Ten mau (Giam dan)
        </option>
      <option value = "ColorHexRGBValue">Gia tri mau RGB
        (Tang dan)</option>
      <option value = "-ColorHexRGBValue">Gia tri mau RGB
        (Giam dan)</option>
    </select>
    </caption>
```

Chương 17

Liên kết dữ liệu

```
<THEAD>
<TR style = "background-color: mediumslateblue">
  <th>Ten mau</th>
  <th>Gia tri mau RGB</th>
</TR>
</THEAD>
<TBODY>
<TR style = "background-color: lightsteelblue">
  <TD><span datafld = "ColorName"></span></TD>
  <TD><span datafld = "ColorHexRGBValue"
    style = "font-family: monospace"></span></TD>
</TR>
</TBODY>
</TABLE>
</BODY>
</HTML>
```

Trong ví dụ này, chúng ta sắp xếp theo cả hai thứ tự tăng dần và giảm dần đối với tên màu và cả giá trị màu. Người dùng có thể chọn cách sắp xếp từ một danh sách.

- Thuộc tính *sort* sẽ nhận giá trị được lựa chọn từ *option* khi xảy ra sự kiện *onchange*
select onchange = "Colors.Sort = this.value;
- Sau khi đặt giá trị cho thuộc tính *sort*, chúng ta sẽ gọi phương thức *reset* của TDC để hiển thị dữ liệu sau khi đã được sắp xếp.

Colors.Reset();"

- Để sắp xếp theo thứ tự giảm dần, tên cột sẽ được chỉ ra cùng với ký tự "-" đứng ở phía trước tên cột:

<option value = "-ColorName">Ten mau (Giảm dần </option>

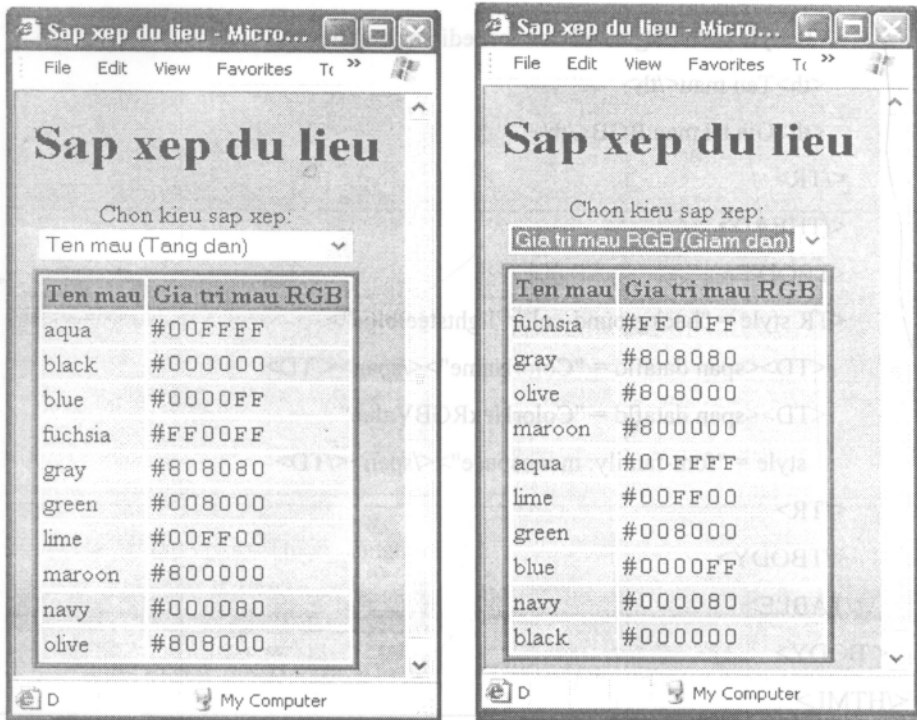
<option value = "-ColorHexRGBValue">Gia tri mau RGB(Giảm dần)</option>

Nếu không có ký tự "-" đứng trước thì giá trị sắp xếp mặc định là tăng dần.

Chương 17

Liên kết dữ liệu

Kết quả của ví dụ này được chỉ ra trong hình sau:



Theo thứ tự tăng dần tên màu Theo thứ tự giảm dần giá trị màu RGB

Hình 17.5. Minh họa cho sắp xếp dữ liệu.

17.5.2. Lọc dữ liệu

Các dạng lọc có thể được chỉ ra trong khai báo `<OBJECT>` như một giá trị tham số PARAM. Các thuộc tính linh hoạt sẽ xuất hiện khi bạn thêm các thuộc tính lọc vào script. Công việc này được làm bằng cách dùng các thuộc tính của đối tượng nguồn dữ liệu.

- **FilterColumn:** chỉ ra các cột ở đó dạng lọc sẽ có.
- **FilterValue:** chỉ ra giá trị cho dạng lọc.
- **FilterCriterion:** chỉ ra toán tử được dùng trong tiêu chí lọc.

Các toán tử **FilterCriterion** gồm:

Chương 17

Liên kết dữ liệu

Bảng 17.1. Các toán tử FilterCriterion

Toán tử	Mô tả
<	Nhỏ hơn
<=	Nhỏ hơn hoặc bằng
>	Lớn hơn
>=	Lớn hơn hoặc bằng
=	Bằng
<>	Khác

Ví dụ sau hiển thị dữ liệu được lọc theo thuộc tính *filterCriterion*. Chú ý rằng file *employees.csv* do chúng ta tạo ra và để cùng thư mục với file *.html* mà ta viết dưới đây. Cấu trúc file *employees.csv* như ở hình 17.6.

Ví dụ 17.5.

```
<HTML>
  <HEAD>
    <TITLE>Employee Data</TITLE>
    <SCRIPT LANGUAGE = "JavaScript">
      function filterdata ()
      {
        employees.FilterColumn = "Designation";
        employees.FilterValue = "Manager";
        employees.FilterCriterion = "=";
        employees.Reset();
      }
      function nofilter ()
      {
        employees.FilterColumn = "";
        employees.Reset();
      }
    </SCRIPT>
```

Chương 17

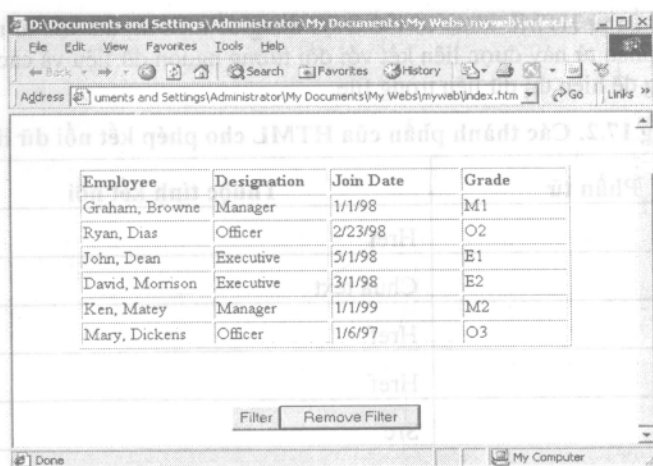
Liên kết dữ liệu

```
</HEAD>
<BODY>
<H1><CENTER><FONT size = 5 color = red>Employee
Data</FONT></CENTER></H1>
<BR>
<BR>
<OBJECT CLASSID = "clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
ID = employees>
<PARAM NAME = "DaTaURL" Value = "employees.csv">
<PARAM NAME = "UseHeader" value = "True">
</OBJECT>
<TABLE BORDER = 1 DATASRC = "#employees">
<THEAD>
<TH>Employee</TH>
<TH>Designation</TH>
<TH>Join_Date</TH>
<TH>Grade</TH>
<TBODY>
<TR>
<TD><SPAN DATAFLD = Employee_Name></SPAN>
<TD><SPAN DATAFLD = Designation></SPAN>
<TD><SPAN DATAFLD = Join_Date></SPAN>
<TD><SPAN DATAFLD = Grade></SPAN>
</TABLE>
<BR><BR>
<INPUT TYPE = button value = "Filter" onclick = "filterdata()">
<INPUT TYPE = button value = "Remove Filter" onclick = "nofilter()">
</BODY>
</HTML>
```

Chương 17

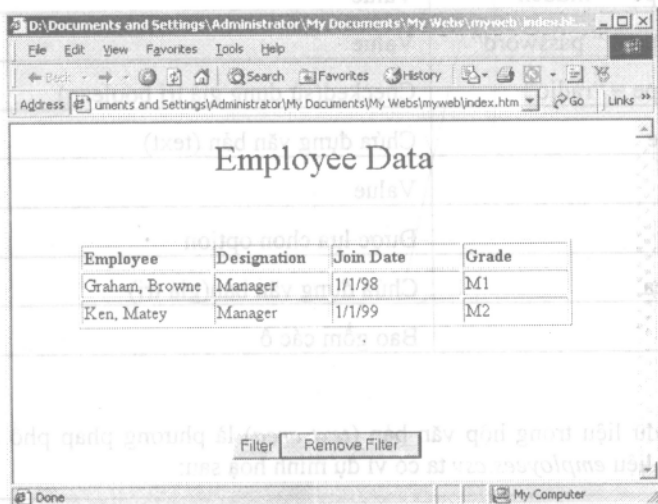
Liên kết dữ liệu

Kết quả giao diện như sau:



Hình 17.6. Hiển thị dữ liệu.

Khi bạn kích vào nút *Filter*, dữ liệu được lọc và hiển thị theo thuộc tính *FilterCriterion*.



Hình 17.7. Sau khi đã lọc dữ liệu.

17.6. Sử dụng các phần tử HTML khác

Một số phần tử HTML khác có thể được dùng để hiển thị dữ liệu từ một file dữ liệu dạng bảng. Các phần tử này được liên kết với đối tượng nguồn dữ liệu và các thuộc tính của chúng được dùng để hiển thị dữ liệu trong file.

Bảng 17.2. Các thành phần của HTML cho phép kết nối dữ liệu

Phần tử	Thuộc tính kết nối
A	Href
Div	Chứa text
Frame	Href
Iframe	Href
Img	Src
Input type = "button"	Value(button text)
Input type = "checkbox"	Checked(sử dụng giá trị boolean trong dữ liệu)
Input type = "hidden"	Value
Input type = " password"	Value
Input type = "radio"	Checked(sử dụng giá trị boolean)
Marquee	Chứa đựng văn bản (text)
Param	Value
Select	Được lựa chọn option
TextArea	Chứa đựng văn bản(giá trị)
Table	Bao gồm các ô

Ví dụ 17.6.

Hiển thị dữ liệu trong hộp văn bản (*text area*) là phương pháp phổ biến hiện nay. Cũng với file dữ liệu *employees.csv* ta có ví dụ minh hoạ sau:

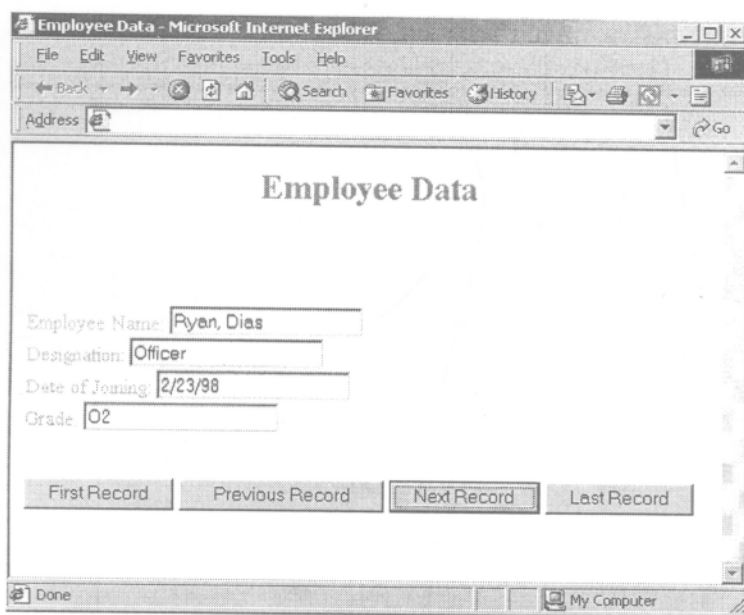
```
<HTML>
  <HEAD>
    <TITLE>Employee Data</TITLE>
```

```
<SCRIPT LANGUAGE = "JavaScript">
function first ()
{
employees.recordset.MoveFirst () ;
}
function last ()
{
employees.recordset.MoveLast () ;
}
function previous ()
{
employees.recordset.MovePrevious () ;
if (employees.recordset.BOF)
{
employees.recordset.MoveFirst () ;
}
}
function next ()
{
employees.recordset.MoveNext ();
if (employees.recordset.BOF)
{
employees.recordset.MoveLast () ;
}
}
</SCRIPT>
</HEAD>
```

```
<BODY>
<H1 style = "color:red; font-size:25; align:center"><CENTER>Employee
Data</CENTER></H1>
<BR>
<BR>
<OBJECT CLASSID = "clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
    ID = employees>
    <PARAM NAME = "DaTaURL" Value = "employees.csv">
    <PARAM NAME = "UseHeader" value ="True">
</OBJECT>
<P style = "color:hotpink; font-size:15">
    Employee Name:
    <INPUT TYPE = text DATASRC = "#employees" DATAFLD =
    Employee_Name><BR>
        Designation:
    <INPUT TYPE = text DATASRC = "#employees" DATAFLD =
    Designation><BR>
        Date of Joining:
    <INPUT TYPE = text DATASRC = "#employees" DATAFLD =
    Join_Data><BR>
        Grade:
    <INPUT TYPE = text DATASRC = "#employees" DATAFLD =
    Grade>
    <BR><BR><BR>
    <INPUT type = button value = "First Record" onclick = "first()">
    <INPUT type = button value = "Previous Record" onclick = "previous()>
    <INPUT type = button value = "Next Record" onclick = "next()>
    <INPUT type = button value = "Last Record" onclick = "last()>
</BODY>
</HTML>
```

Chương 17

Liên kết dữ liệu



Hình 17.8. Hiển thị dữ liệu trong hộp textarea.

17.7. Kết chương

Chương này giới thiệu về tính năng liên kết dữ liệu do Internet Explorer hỗ trợ. Bạn đã được giới thiệu cách tạo một file dữ liệu, hiển thị dữ liệu từ file và thao tác trên dữ liệu sau khi nó đã được hiển thị trên trang. Bạn cũng đã được giới thiệu cách liên kết dữ liệu trong một file với các phần tử HTML khác nhau. Kết thúc chương này cũng là kết thúc phần giới thiệu về DHTML. Trong phần sau bạn sẽ được tìm hiểu về ngôn ngữ *Wireless Markup Language* (WML) là một ngôn ngữ xây dựng phục vụ việc hiển thị trên thiết bị không dây.

PHẦN 6

WIRELESS MARKUP LANGUAGE

NỘI DUNG

Chương 18 - WML Cơ bản

Chương 19 - WML cơ bản - tiếp theo

Chương 20 - Tổng quan về WMLScript

Chương 21 - Hàm và Sử dụng hàm trong WMLScript

Chương 22 - Cấu trúc điều khiển trong WMLScript

**Chương 23 - Cấu trúc điều khiển trong WMLScript (tiếp
theo)**

Chương 24 - Các đối tượng trong WML

Chương 18

Wireless Markup Language (WML) Cơ bản

Nội dung của chương:

- Giới thiệu về WML.
- Thẻ, biến WML và các ngữ cảnh.
- Tác vụ và các sự kiện của WML.
- Tương tác người dùng WML.

18.1. Giới thiệu về WML

WML là gì?

Wireless Markup Language (WML) là một ngôn ngữ xây dựng phục vụ việc hiển thị trên thiết bị không dây. WML có nhiều điểm tương đồng với HTML nhưng hai ngôn ngữ này cũng có nhiều điểm khác nhau. Một trong số những điểm khác nhau này có lý do xuất phát từ tính đơn giản của WML khi so sánh với HTML. Nhiều tính năng có thể tìm thấy được ở HTML mà lại không hiện diện ở WML.

Chẳng hạn như HTML cho phép thể hiện bản chất của văn bản trong phạm vi một tài liệu, có thể thay đổi kích cỡ, xác định kiểu chữ được sử dụng, thêm các kiểu cách như là chữ đậm hay gạch chân và điều chỉnh màu sắc. So với HTML về phương diện biểu diễn tài liệu, WML chỉ cung cấp một vài kiểu hiệu chỉnh đơn giản như: chữ đậm, nghiêng, gạch chân và khả năng xác định văn bản cỡ lớn hay nhỏ hơn bình thường. Nhưng hầu hết những trình duyệt WAP hiện tại lại không hỗ trợ thậm chí cả những tính năng ít ỏi trên.

Tuy nhiên trong trường hợp này chúng ta tập trung vào nội dung của văn bản hơn là hình thức. Nếu ai đã từng thiết kế một trang web để thích hợp với nhiều người sử dụng, có nghĩa là tương thích với nhiều trình duyệt, thì chắc chắn sẽ thấy tầm quan trọng về sự ngắn gọn và đơn giản của nội dung cần thể hiện. Nguyên nhân do không phải tất cả các trình duyệt khác nhau đều hiểu hết và hiểu giống nhau về cùng một cú pháp HTML. Để thể hiện được đúng một trang web trên các trình duyệt khác nhau các bạn nên tuân theo ba nguyên tắc cơ bản:

- Tránh những cú pháp mới hoàn toàn (vừa mới được phát triển).
- Cố gắng viết các trang web có thể chạy trên bất kỳ trình duyệt nào.
- Viết nhiều phiên bản cho trang web tương ứng với các trình duyệt khác nhau và cho phép người sử dụng chọn lựa phiên bản phù hợp với trình duyệt.

Như đã nói ở trên, hiện nay hầu hết những trình duyệt WAP không hỗ trợ hình ảnh, bảng biểu, thay đổi hiệu ứng văn bản... vì thế khi viết trang WML nên lưu ý đến vấn đề tương thích với đa dạng của trình duyệt.

Trong khi WML mất đi nhiều đặc điểm của HTML thì nó lại được thêm vào một số những đặc điểm mới. Thêm nữa WML do khi thiết kế từ đầu đã có sự xem xét những vấn đề mà HTML gặp phải cho nên WML tránh được một số nhược điểm mà HTML mắc phải.

Cài đặt phần mềm giả lập thiết bị không dây Phone Simulator

Hiện nay hãng Openwave (<http://www.openwave.com>) đã cung cấp rất nhiều phiên bản giả lập thiết bị không dây : **OpenwaveSDK Phone Simulator** (miễn phí) tại địa chỉ: <http://developer.openwave.com/dvl/>. Để chạy được các trang WML đơn giản, bạn cần tải về một trong các phiên bản ở trang web trên và cài đặt trên máy tính.

Cuốn sách này sử dụng phiên bản **Openwave SDK 6.2.2** để minh họa cho nội dung phần WML, tuy nhiên sang phần WMLScript chúng ta nên sử dụng phiên bản khác (4.1) do phiên bản 6.2.2 không hỗ trợ việc cài đặt máy chủ cục bộ (*local webserver*). Phần cài đặt để phục vụ việc biên dịch WMLScript khá phức tạp và sẽ được tìm hiểu trong phần sau.

18.2. Ví dụ mở đầu

Trước khi thảo luận về WML, chúng ta hãy xem xét một ví dụ kinh điển thường gặp mỗi khi học một ngôn ngữ lập trình nào: *Hiển thị dòng chữ "Hello world"*.

Ví dụ 18.1

- Chương trình đơn giản WML *Hiển thị dòng chữ "Hello world"* như sau:

Chương 18

Wireless Markup Language (WML) Cơ bản

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC
"-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>

<card title="vi du dau tien">

<p>
Hello, World!
</p>

</card>

</wml>
```

- Kết quả hiển thị trên trình duyệt



Hình 18.1. Chương trình Hello world.

- Điều cần chú ý là phần tiêu đề gồm 4 dòng đầu tiên. Có vẻ phức tạp và rất khó hiểu song các bạn không thực sự cần phải hiểu chúng có nghĩa gì: để bắt đầu làm quen với WML chúng ta chỉ cần đặt chúng trên phần đầu các trang WML của chúng ta.
- Điều thứ hai cần chú ý là tất cả thẻ tên đều ở dạng chữ thường (các thẻ đều trong ngoặc nhọn). Không giống như HTML, nếu như `<HTML>`, `<html>`, `<Html>` và `<hTMl>` đều có cùng ý nghĩa (không phân biệt chữ hoa, chữ thường) thì trong WML, tất cả các thẻ tên WML đều là chữ thường.

Chương 18

Wireless Markup Language (WML) Cơ bản

- Phần còn lại của trang WML rất giống với trang HTML, mặc dù thẻ đầu tiên là `<wml>`, không phải là `<HTML>` và có thẻ `<card>`. Thẻ `<p>` đơn giản là bắt đầu một đoạn văn bản (trong ví dụ này, chỉ có một thẻ `<p>` duy nhất).

18.3. Các khái niệm mở đầu

18.3.1. Thẻ

Thẻ WML có cấu trúc tương tự HTML nhưng nội dung khác với HTML. Thay vì một file là một lớp dài nội dung, một file của WML biểu thị một lớp các thẻ (*deck of cards*).

- Mỗi thẻ chỉ được hiển thị tại một thời điểm, nhưng có thể liên kết từ một thẻ tới thẻ khác.
- Nhiều thẻ có thể chứa trong một *deck* trong cùng một file.
- Mỗi thẻ trên thực tế vận hành gần giống như một trang HTML, vì thế có thể xem như WML bao gồm một số lượng các trang HTML tập hợp lại cùng nhau.

Điều này mang lại nhiều thuận lợi vì WML sử dụng lớp các thẻ để nhóm thông tin liên quan lại với nhau hoặc phục vụ tương tác với người sử dụng. Tất cả các thẻ trong lớp đều được tải xuống ở cùng một thời điểm, vì vậy người sử dụng chỉ cần phải chờ một lần, những lần sau không phải chờ đợi các thẻ tải xuống khi sử dụng chúng nữa. Nhưng tuy nhiên điều này làm cho WML *lạm dụng các thẻ*. Nếu như một file chứa khoảng 30 thẻ chứa thông tin và đều cùng được tải xuống cùng một thời điểm thì tốn rất nhiều thời gian cho việc chờ đợi tất cả được tải xuống. Khó có thể đặt giới hạn cố định số thẻ trên một file, tuy nhiên có một số quy tắc chung là: Nếu như sử dụng nhiều hơn 5 hay 6 thẻ đơn trong 1 *deck*, nên nghĩ đến việc làm thế nào mà trang tài liệu có thể tải về và được nhận dạng. Thêm nữa nhiều thiết bị không thể làm việc được với các file lớn (Điện thoại Nokia 7110, thiết bị WAP thông dụng nhất ở châu Âu, gặp vấn đề nếu như một *deck* chứa nhiều hơn 1400 byte khi truyền qua *gateway*).

18.3.1.1. Thẻ phân tử rỗng

Hãy xem xét một ví dụ khác.

Ví dụ 18.2.

Cấu trúc của ví dụ 18.2 tương tự ví dụ 18.1, có xuất hiện thẻ `<img>` như trong HTML. Tuy nhiên hãy để ý phần cuối của thẻ có thêm dấu `/`.

Điều này là bắt buộc ở WML: trong trường hợp thẻ không có thẻ kết thúc tương ứng Thẻ kết thúc có dấu gạch sau ngoặc nhọn đầu tiên, giống như `</end>`). Những thẻ này vì thế được gọi là các thẻ phân tử rỗng và khá thông dụng trong WML và HTML.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC
"-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card title="mot vi du khac">
<p>

</p>
</card>
</wml>
```

18.3.1.2. Các phần tử

Một phần tử (*element*) được chứa một trong hai dạng sau:

- Chứa trong thẻ bắt đầu và thẻ kết thúc:

<tag>...</tag>

- Hoặc một thẻ phần tử rỗng (ghi nhớ kết thúc bởi dấu /):

<tag/>

Chú ý :

- Một phần tử có thể chứa các phần tử khác (không những là dạng cặp thẻ mà còn có thể là những thẻ phần tử rỗng) và có thể trộn lẫn trong văn bản.

Quan sát ví dụ ở trên, có một phần tử `<wml>` (cặp thẻ kết hợp `<wml>` `</wml>`), chứa phần tử `<card>` (cặp thẻ kết hợp `<card>` `</card>`), chứa phần tử `<p>` (cặp thẻ kết hợp, `<p>` `</p>`), và phần tử `<p>` chứa phần tử `<img>` (thẻ phần tử rỗng `<img/>`).

- Lưu ý các thẻ không cân xứng như `<x>...<y>...</x>...</y>` sẽ sinh ra lỗi cản trở việc hiển thị. Cấu trúc thẻ này phải được viết lại như sau `<x>...<y>...</y>...</x>`

Chương 18

Wireless Markup Language (WML) Cơ bản

18.3.1.3. Thuộc tính

Các thẻ bắt đầu (*begin-tag*) và các thẻ phần tử rỗng (không phải là thẻ kết thúc) đều có các thuộc tính. Các thuộc tính ảnh hưởng đến hoạt động của tất cả các phần tử, điều này cho thấy tại sao chúng không thể xuất hiện ở các thẻ kết thúc. (Mục đích duy nhất của thẻ kết thúc là đánh dấu sự kết thúc của phần tử đó).

Mỗi phần tử cung cấp các thuộc tính khác nhau, ví dụ: thuộc tính *title* trên thẻ *<card>* thiết lập tiêu đề biểu thị cùng với thẻ, trong khi thuộc tính *src* trên phần tử ** lại chỉ ra đường dẫn chứa hình ảnh sẽ được tìm thấy.

Chú ý:

Một sự khác nhau nhỏ nữa giữa WML và HTML là trong việc ghi giá trị thuộc tính. Ở HTML, giá trị thuộc tính có thể xuất hiện trong dấu ngoặc trích dẫn đơn (*attr = 'value'*), dấu ngoặc trích dẫn kép (*attr = "value"*) và hầu hết những trình duyệt web cũng cho phép chúng không cần trích dẫn (*attr = value*) thì WML không cho phép trích ghi giá trị thuộc tính mà không có trích dẫn và tất cả giá trị phải xuất hiện trong dấu ngoặc trích dẫn đơn hoặc dấu ngoặc trích dẫn đôi.

18.3.1.4. Ký hiệu cho các thuộc tính

Mỗi một thuộc tính có tên nhất định và giá trị đi kèm với các thuộc tính cũng được quy định. Dưới đây chúng ta sẽ làm quen với một số thuộc tính quen thuộc.

- **String**: Giá trị có thể là chuỗi bất kỳ nào đó.
- **url**: Giá trị nên là địa chỉ URL hợp lệ. Cũng có thể là địa chỉ URL tương đối (ở đó chỉ có một phần của URL được đưa ra) và địa chỉ URL tương đối sẽ được phân giải thành địa chỉ tuyệt đối.
- **number**: Giá trị nên là số không âm.
- **Length**: Giá trị biểu diễn độ dài của màn hình hiển thị trình duyệt. Giá trị này có thể được đo bằng pixel hoặc cũng có thể đo bằng phần trăm (%) có nghĩa là tỉ phần trăm chiều rộng, chiều cao so với chiều rộng, chiều cao của màn hình. Dù đo bằng đại lượng nào thì giá trị phải là số không âm.
- **Boolean**: Nhận một trong hai giá trị: true hoặc false.
- **Name**: Giá trị là chuỗi chứa các chữ cái, con số, dấu chấm, hai chấm, ngắt quãng và gạch dưới.
- **Variable**: Giá trị chứa các biến.
- **Optional**: Thuộc tính tùy chọn (có thể bỏ qua).

Chương 18

Wireless Markup Language (WML) Cơ bản

- **Required:** Thuộc tính phải được cung cấp giúp cho phần tử xử lý. Nhiều trình duyệt sẽ từ chối xử lý *deck* nếu như các thuộc tính yêu cầu bị bỏ qua.
- **Default:** Giá trị mặc định cho các thuộc tính. Giá trị này được sử dụng nếu như thuộc tính không được xác định rõ ràng trên thẻ. Chỉ có thuộc tính *optional* mới có thể có giá trị mặc định.

18.3.2. Các thực thể

Thực thể có tác dụng trong việc đưa các biểu tượng hay kí hiệu đặc biệt (dấu ngoặc, kí hiệu lớn hơn, nhỏ hơn và nhiều ký hiệu khác) vào trong trang HTML. Mục đích là để hiển thị những biểu tượng mà không phải lúc nào cũng có thể gõ trực tiếp vào (ví dụ không thể thấy kí hiệu của đồng bảng Anh trên bàn phím - mặc dù kí hiệu USD \$ thì có) hoặc để hiển thị những kí tự mà WML đã quy định cho nó một vai trò đặc biệt, ví dụ nếu để kí tự "<" trong văn bản bình thường, trình duyệt nghĩ rằng đó là sự bắt đầu của một thẻ trình duyệt, nó sẽ báo lỗi vì không tìm thấy thẻ kết thúc hoặc quy cách thẻ không đúng.

Bảng 18.1 biểu thị cơ cấu 3 loại thực thể trong WML. Thực thể được định tên sẵn đã được quen thuộc từ HTML. Chúng có dạng như **&** hoặc **<**; và biểu diễn những kí tự riêng biệt qua một tên dễ nhớ. Thực thể cũng có thể được đưa vào thông qua trong hai dạng số (thập phân hay hexa) cho phép đưa vào bất kỳ một kí tự Unicode trong tài liệu WML (điều này không đảm bảo rằng tất cả các trình duyệt đều có thể diễn tả được).

- Thực thể số thập phân giống như **!**; (Kí tự ! trong Unicode) hoặc **£**; (kí hiệu **bảng Anh** trong Unicode).
- Thực thể số hexa giống như **!** hoặc **£**; cho cùng hai ký tự giống nhau

Bảng 18.1 Các thực thể

Thực thể theo tên	Thực thể thập phân	Thực thể Hexa	Kí tự
"	"	"	"
&	&	&	&
'	'	'	'
<	<	<	<
>	>	>	>
 	 	 	Dấu cách
­	­	­	Gạch nối

Chú ý:

Chương 18

Wireless Markup Language (WML) Cơ bản

Lưu ý rằng tất cả các thực thể đều bắt đầu với kí tự và "&" và kết thúc với dấu chấm phẩy ";". Dấu chấm phẩy này khá quan trọng, một số trang web quên đi điều này và gây ra vấn đề cho trình duyệt.

Hầu hết những trình duyệt web đang loại bỏ những cấu trúc HTML không hợp lệ, vì thế nhiều lỗi thông thường từng bước bị bỏ qua. Trình duyệt WAP cũng rất khắt khe với những lỗi như thế này.

18.3.3. Viết chú thích

Những lời giải thích trong file WML nhằm mục đích nhắc lại các công việc đang làm tại một vị trí nào đó hoặc giúp cho người viết dễ dàng trong việc quản lý mã nguồn sau một khoảng thời gian dài sau khi viết.

Chú thích bắt đầu với bốn kí tự <!-- và kết thúc với ba kí tự -->. Mọi nội dung xuất hiện giữa hai dấu hiệu này, cho dù là các thẻ, văn bản chính yếu, các thực thể và ngắt dòng... đều bị trình duyệt bỏ qua và không ảnh hưởng tới việc hiển thị. Dưới đây là một vài ví dụ:

```
<!-- Đây là một chú thích đơn giản. -->
```

```
<!--Đây là một
```

```
chú thích
```

```
viết trên nhiều dòng.-->
```

Chú ý:

- Hai kí tự gạch nối "--" không thể xuất hiện trong thân của chú thích, chỉ ở phần cuối như là một phần của dãy kí tự kết thúc -->.
- Nếu như vì một số lý do nào đó bạn muốn chuỗi <!-- xuất hiện trong văn bản thì bạn phải viết nó cùng với một thực thể. <!--

Một câu hỏi đặt ra là việc sử dụng chú thích có thể gây một số lo ngại như có thể nó bị hiển thị ra cho người sử dụng hay việc ảnh hưởng của chúng tới băng thông truyền số liệu vốn đã hẹp? Câu trả lời là các bạn hoàn toàn có thể yên tâm vì thực ra quá trình xử lý của *WAP gateway* sẽ loại bỏ tất cả các chú thích và do đó các trình duyệt không hề biết đến sự tồn tại của chúng.

18.4. Biến WML và các ngữ cảnh

Sự khác biệt đáng kể giữa WML và HTML là WML hỗ trợ các biến. Biến WML chứa các xâu và các xâu này có thể được chèn trong phần thân văn bản của tài liệu WML hoặc trong các giá trị của một thuộc tính nào đó. Giá trị của các biến có thể được thay đổi dưới sự kiểm soát của chính WML và cũng có thể bởi các hàm của WMLScript.

Chương 18

Wireless Markup Language (WML) Cơ bản

Tên của biến WML kết hợp các chữ cái, các số và dấu gạch chân, trong đó ký tự đầu tiên của tên biến phải không được là kiểu số. Có một cách đặt tên quen thuộc trong WML (và trong các ngôn ngữ khác) đó là các tên biến nên đặt gần với nhiệm vụ của biến đó. Dưới đây là một số ví dụ của các tên biến đúng quy tắc và không đúng quy tắc.

Ví dụ 18.3.

Các tên biến theo quy tắc	Các tên biến không theo quy tắc
<code>a</code> <code>__ten_voi_daugachduoi__</code> <code>Motbiencotendai</code> <code>xy17</code>	<code>17xy</code> : (tên biến bắt đầu với một số) <code>name withspaces</code> (tên biến có chứa khoảng trống)

Mặc dù WML không áp đặt giới hạn về độ dài của tên biến, tuy nhiên bạn cần biết rằng tên biến khi xử lý được truyền qua đường truyền, sau đó mới được lưu trữ và hiển thị trong giới hạn có thể được trên trình duyệt. Như vậy việc đặt tên biến dạng ngắn được khuyến khích hơn.

18.4.1. Thay thế biến

Các biến có thể được đưa vào (hoặc được thay thế) trong thân văn bản của tài liệu WML hoặc làm giá trị thuộc tính bằng một trong ba cách sau:

- `$name`
- `$(name)`
- `$(name:conversion)`

Thông thường nên sử dụng cách thứ hai vì cách thứ nhất không chỉ là kém hơn mà còn nó còn có thể là mạo hiểm bởi khi có sự thay đổi nhỏ trong file (ví dụ việc dịch chuyển khoảng cách) có thể khiến cho trình duyệt nhầm lẫn đó là tên biến. Thêm vào đó, một vài trình duyệt hiện nay không hỗ trợ cách thay thế này.

Sự chuyển đổi với dạng thứ ba có thể mang một trong ba giá trị (e, u hoặc n), mỗi giá trị cũng có thể được viết tắt bởi ký tự đầu tiên của các dạng chuyển đổi sẽ được xem xét tiếp sau đây (thường thì không phân biệt chữ hoa chữ thường).

Các kiểu giá trị chuyển đổi

- `escape` hay `e`

Chương 18

Wireless Markup Language (WML) Cơ bản

Giá trị của biến phải qua *URL escaping* trước khi được chuyển đổi. Quá trình này lấy một chuỗi gồm một kí tự % đi kèm với hai chữ số hexa thay thế cho một kí tự và thay thế bằng kí tự + cho mỗi khoảng trống. Thêm vào đó, mỗi chỗ trống trong giá trị được thay thế bởi a +.

Ví dụ, nếu như giá trị của biến *Nicknames* là *Buz, Rocket*, thì *\$(Nicknames:e)* được thay thế bởi *Buz%2C+Rocket*. (Mã hexa cho dấu phẩy (,) có giá trị là 2C).

▪ unesc hay u

Chuyển đổi *u* này thực hiện ngược với chuyển đổi *e* ở trên, nó thay thế một chuỗi gồm một kí tự % kèm theo hai chữ số hexa cho mỗi kí tự và thay thế mỗi dấu - cho một khoảng trống.

Ví dụ, nếu như giá trị của các biến *NicknamesE* là *Buz%2C+Rocket*, sau đó *\$(NicknamesE:u)* được thay thế bởi *Buz, Rocket*.

▪ noesc hoặc n

Giá trị được thay thế trực tiếp. Không có sự chuyển đổi nào được thực hiện.

Chuyển đổi ngầm định

Nếu không có sự chuyển đổi nào được xác định thì trình duyệt sẽ tiến hành chuyển đổi ngầm định, phụ thuộc vào ngữ cảnh.

Nếu biến được thay thế vào thuộc tính của các dạng *onenterforward*, *onenterbackward*, *href* hoặc *src*, chuyển đổi ngầm định sẽ tạo sự thoát, bởi lẽ tất cả các giá trị thuộc tính này phân loại các URL thành các loại biến đổi (bạn sẽ học về những điều này ở các chương sau). Trong các ngữ cảnh khác sự ngầm định là *noesc*.

Một điểm cần ghi nhớ (sau khi bạn đã đọc chương 3 và chương 7) chính là thuộc tính của các loại *onclick* và *ontimer* ngầm định là *noesc*, mặc dù chúng ta mong rằng chúng ngầm định là *escape*. Bởi sự mâu thuẫn này mà các nhà thiết kế WAP luôn luôn xác định sự chuyển biến rõ ràng khi các biến thay thế vào các giá trị thuộc tính, đặc biệt là khi các giá trị chứa các URL.

Ví dụ 18.4.

Để minh hoạ cho sự chuyển đổi này, giả sử rằng biến *user* chứa giá trị "fred" và biến *greeting* chứa giá trị "hello, world" thì thuộc tính:

title = "welcome, \$(user)! \$(greeting)"

trở thành:

title = "welcome, fred ! Hello, world !"

Tương tự, thuộc tính:

Chương 18

Wireless Markup Language (WML) Cơ bản

href = "/cgi-bin/login?u=\$(user:e)&g=\$(greeting:e)"

trở thành:

href = "/cgi-bin/login?u=fred&g=Hello%2C+world%21"

Chú ý rằng tốt hơn hết chúng ta nên sử dụng dạng chuyển đổi rõ ràng (sử dụng :e), hơn là phụ thuộc vào việc trình duyệt thực thi sự chuyển đổi ngầm định.

18.4.1.1. Biến rỗng

Điều gì sẽ xảy ra nếu như sự cố gắng thay thế biến là không xác định? Câu trả lời đơn giản đó là biến không xác định tương đương với chuỗi rỗng nên: *foo\$(undefined)bar* trở thành *foobar* sau khi biến được thay thế. Sau đó thực hiện gán chuỗi rỗng cho biến và loại bỏ nó khỏi ngữ cảnh trình duyệt.

18.4.1.2. Quy định của biến thay thế

Nếu như \$ bắt đầu một biến, thì làm cách nào đặt ký tự \$ vào trang hoặc giá trị thuộc tính? Ví dụ khi chúng ta muốn biểu thị giá này *Price: \$17.00,...*?

Bạn sẽ trả lời là có thể sử dụng các thực thể như đặt `$` để thực hiện. Tuy nhiên khi bạn làm điều này thì không may là nó không chạy vì không phù hợp với cách mà các trình duyệt WML xử lý thay thế biến. Vậy điều gì đã xảy ra khi trình duyệt thay thế tất cả các thực thể bằng các ký tự mà các thực thể biểu diễn: nghĩa là trong khi `$` biểu diễn cho \$, thì trình duyệt sau đó lại dịch ký tự \$ này như là sự bắt đầu của một biến liên quan khác.?

Chú ý về biến WML và XML

Lý do không thể làm việc được với các biến WML cùng với các thực thể là do có mối quan hệ giữa WML và XML.

- WML được thiết kế để bộ phân tích cú pháp XML cũng có thể xử lý trang WML. Bộ xử lý chuyển tất cả các phần tử thành một cấu trúc và phần thân văn bản được đặt rải rác. Phần thân văn bản đó không chứa bất kỳ cú pháp XML đặc biệt nào, nó chỉ là một văn bản đơn thuần.
- XML không xem xét các biến WML.

Ví dụ chuỗi *\$(foo)* chỉ là chuỗi đơn thuần gồm 6 ký tự. Chỉ sau khi phân tích cấu trúc XML hoàn thành thì bộ phân tích cấu trúc WML mới tiến hành xử lý các biến WML. Đến lúc này tất cả thực thể `$` đã trở thành những ký tự \$, khiến cho bộ phân tích xem nó như là phần đầu của một biến. Tình huống này cũng giải thích tại sao không thể thay đổi cấu trúc trang bởi các biến. Do cấu trúc của trang được định nghĩa ở mức XML bằng việc sử

Chương 18

Wireless Markup Language (WML) Cơ bản

dùng các thẻ và các thuộc tính; các biến WML lại được xử lý sau đó. Nếu như bộ phân tích XML tìm ra ký tự \$ ở tên thẻ hoặc ở tên các thuộc tính, nó sẽ từ chối tài liệu vì \$ không phải là một ký tự đặt tên đúng quy định trong XML.

Thật may mắn là chúng ta có một cách để tránh được điều trên. Chuỗi \$\$ luôn luôn biểu diễn một \$ khi nó xuất hiện ở trong văn bản hoặc trong giá trị thuộc tính. Vì thế cách chính xác biểu thị chuỗi “Price: \$17.00” là “Price: \$\$17.00”. Điều này có nghĩa là:

- Chuỗi \$\$*(foo)* biểu diễn cho 6 ký tự \$(foo) và nó không là một biến nào cả.
- Chuỗi \$\$\$*(foo)* biểu thị ký tự \$ theo sau là giá trị của biến foo với sự chuyển đổi ngầm định.
- \$ đứng một mình (không theo sau bởi ký tự \$ khác, hay (, hoặc việc bắt đầu tên biến) là không đúng quy tắc.

Một điều khác cần chú ý đó là không thể thay đổi các cấu trúc thực của trang khi sử dụng biến. Có thể thay đổi văn bản ở giữa các thẻ và thay đổi một vài giá trị thuộc tính nhưng không thể thêm bất kỳ một thẻ nào hoặc thuộc tính nào cả.

Ví dụ 18.5.

Nếu biến *img* chứa giá trị `<img src = “img.wbmp”/>`, sau đó là :

some text\$(img) some more text

thì đơn giản tương đương với chuỗi:

some textsome more text

Các phần thẻ tiếp theo đây đều không hợp lệ dù giá trị của biến foo là:

`<$foo href = “img.wbmp”/>`

`<img $(foo)= “img.wbmp”/>`

Nếu như cần thay đổi cấu trúc của trang như trên, thì phải thực hiện trên phía server.

18.4.2. Thiết lập biến

Trên đây ta đã tìm hiểu về những điều cần thiết khi sử dụng các biến WML, nhưng có một vấn đề chưa rõ là làm thế nào để thiết lập giá trị cho các biến ở vị trí đầu tiên.

Để làm điều này trong thực tế có ba cách:

- Cách thông dụng nhất đó là thông qua rất nhiều phần tử giao diện với người dùng.
- Cách thứ hai là các biến cũng có thể được thiết lập từ WMLScript (sẽ được xem xét ở phần WMLScript).

- Cách thứ 3 là bằng phần tử `<setvar>`

Chúng ta sẽ xem xét cách thứ 3 là bằng phần tử `<setvar>`.

18.4.2.1. Phần tử `<setvar>`

Phần tử này phải được đặt bên trong phần tử `task`. Mục đích của nó là để gán giá trị cho một biến WML. Quá trình gán này được thực hiện khi tác vụ được thực thi.

Thuộc tính của phần tử `<setvar>`:

- *name* (yêu cầu; biến xâu): Định rõ tên của biến được thiết lập.
- *value* (yêu cầu; biến xâu): Xác định giá trị mới cho biến.

Ví dụ 18.6.

Phần tử `<setvar name = "var" value = "foo"/>` sẽ tạo biến gọi là *var*, chứa chuỗi *foo*. Nếu như biến đã tồn tại thì giá trị của nó được chuyển thành xâu *foo*.

18.4.2.2. Thứ tự của thao tác trong việc thiết lập các biến

Trước khi thực thi một hay nhiều hơn các phần tử `<setvar>` trong phạm vi một tác vụ, trình duyệt thực hiện lần lượt các phần tử `<setvar>`.

Ví dụ 18.7.

Để minh họa cho phần này, hình dung là biến *x* chứa giá trị *one*, *y* chứa *two*, và *z* chứa *three*. Bây giờ giả định rằng ba phần tử `<setvar>` như sau:

```
<setvar name="x" value="$y"/>
```

```
<setvar name="y" value="$z"/>
```

```
<setvar name="z" value="$x"/>
```

Trình duyệt đầu tiên thay thế biến:

```
<setvar name="x" value="two"/>
```

```
<setvar name="y" value="three"/>
```

```
<setvar name="z" value="one"/>
```

Cuối cùng, trình duyệt gán các giá trị mới cho ba biến đó.

Như vậy sự thay thế biến bao giờ cũng được thực hiện trước sau đó mới thực hiện phần tử `<setvar>`.

18.4.3. Các ngữ cảnh trình duyệt

Ngữ cảnh trình duyệt WML là tập hợp tất cả các biến hiện tại được thiết lập, đi kèm với danh mục tất cả các thẻ mà người sử dụng đã xem (*history stack*).

Ngữ cảnh bị trống (tất cả các biến không được thiết lập và stack cũng trống) khi mà thẻ được hiển thị với việc thiết lập thuộc tính *newcontext* là *true*. Xem chương sau để tìm hiểu kỹ hơn các thẻ và các thuộc tính của nó.

Thêm vào đó, bất kỳ khi nào người sử dụng gửi trình duyệt tới vị trí mới mà không được liên hệ trong thẻ hiện tại, ngữ cảnh mới được tạo ra. Một vài trình duyệt hỗ trợ ngữ cảnh phức tạp, chẳng hạn như sự kiện hiện tại có thể đối nghịch với ngữ cảnh tiếp theo trong sự kiện nào đó. Song thông thường người ta chỉ nên sử dụng một ngữ cảnh, nghĩa là nó tương đương với việc làm rỗng các ngữ cảnh hiện tại.

18.5. Tác vụ và các sự kiện của WML

Trong các phần chúng ta đã tìm hiểu khái niệm biến trong WML - một khái niệm không có ở HTML. Trong phần này sẽ giới thiệu thêm hai tính năng khá nổi trội của WML - *Tác vụ* và các *sự kiện* - tất nhiên hai tính năng này không xuất hiện trong HTML, tuy nhiên một số sự kiện có thể sử dụng JavaScript để tạo các hiệu ứng tương tự.

18.5.1. Các tác vụ

Tác vụ WML là một hoạt động được thi hành bởi trình duyệt. Ví dụ: hành động chuyển tới một thẻ mới được thực hiện bởi tác vụ `<go>` hay hành động quay trở lại *card* đã thăm trước đó được thực hiện bởi tác vụ `<prev>`. Các phần tử tác vụ chứa tất cả các thông tin cần thiết cho quá trình thực thi một hành động.

Tác vụ được sử dụng ở nhiều nơi trong WML. Các sự kiện đều được gắn chặt chẽ với các tác vụ và nhiều thành phần giao diện cũng sử dụng tác vụ để thực thi các hành động của mình.

Để tìm hiểu xem tác vụ được sử dụng như thế nào trong ngữ cảnh, hãy quan sát phần tử `<do>` - phần tử cho phép tương tác với người sử dụng ví dụ như nút bấm, menu trong điện thoại di động hoặc có thể thậm chí là một nút trên màn hình. Bản thân phần tử `<do>` không phải là một phần tử tác vụ mà nó chứa nhiều phần tử con nhỏ hơn là các phần tử tác vụ thực hiện hành động khi người sử dụng kích hoạt các điều khiển của phần tử `<do>`.

Ví dụ 18.8.

Ví dụ một phần tử `<do>` khi kích hoạt sẽ gán giá trị *wibble* cho biến *test* có thể được viết như sau:

```
<do type="accept">  
  <refresh>
```

Chương 18

Wireless Markup Language (WML) Cơ bản

```
<setvar name="test" value="wibble"/>
</refresh>
</do>
```

Hoặc một phần tử `<do>` khác khi kích hoạt trình duyệt sẽ gọi tới một *card* mới có tên là *card2* trong *deck* hiện thời có thể viết như sau:

```
<do type="accept">
  <go href="#card2"/>
</do>
```

Chú ý:

Nhận thấy rằng phần tử `<do>` thực chất hoạt động giống nhau trong hai ví dụ trên. Sự khác biệt duy nhất trong hai ví dụ đó là sự xuất hiện của các phần tử `<refresh>` hoặc `<go>`. Chính sự xuất hiện của chúng trong phần tử `<do>` là một lợi thế quan trọng trong sử dụng tác vụ.

Phần tử `<do>` sẽ được giải thích đầy đủ hơn ở phần sau.

18.5.1.1. Tác vụ và các biến

Tất cả các tác vụ đều có thể thay đổi biến trong ngữ cảnh của trình duyệt bằng cách sử dụng phần tử `<setvar>` đã miêu tả trong phần trước. Các ràng buộc đối với các biến mới không ảnh hưởng đến bản thân các tác vụ nhưng lại gây hiệu ứng khi tác vụ đó hoàn thành.

Ví dụ 18.9.

Giả sử ta có biến *page* chứa giá trị *login*. Tác vụ sau:

```
<go href="$(page).wml">
  <setvar name="page" value="bad"/>
</go>
```

sẽ chuyển tới *login.wml*, không phải là *bad.wml*, bởi lẽ trình duyệt đã thay thế giá trị gốc của biến *page* vào thuộc tính *href* trước khi nó gán giá trị mới cho biến.

18.5.1.2. Tác vụ `<go>`

Đúng như tên gọi của mình, tác vụ `<go>` mô tả cho hoạt động chuyển tới một *card* mới. (Nó cũng được sử dụng trong một số mục đích đặc biệt với WMLScript sẽ được nói ở phần sau). Tác vụ `<go>` mang nhiều thuộc tính khác nhau để giúp thông tin xác định các *card* mới. Thông thường, chỉ có thuộc tính *href* và *method* được sử dụng.

Chương 18

Wireless Markup Language (WML) Cơ bản

❖ Các thuộc tính của tác vụ <go>

Các thuộc tính của tác vụ <go> bao gồm:

- *href* (yêu cầu biến url): Đưa ra URL của thẻ mới. Nếu sử dụng URL tương đối thì phải liên quan đến một card hiện thời
- *method* (xâu tùy chọn, mặc định là phương thức *get*): Định rõ phương thức nên sử dụng để nhận được *deck*. Nó phải có một trong những giá trị *get* hoặc *post*, tương ứng với phương thức *GET* hoặc *POST* của HTTP.
- *sendreferer* (tùy chọn boolean, mặc định là *false*): Nếu thiết lập là *true*, trình duyệt gửi URL của *deck* hiện thời cùng với lời yêu cầu. Mục đích của việc này là cho phép server thực thi một kiểm soát truy nhập đơn giản trên *deck*, dựa trên việc *deck* nào được nối với chúng. Ví dụ nếu sử dụng HTTP thì thuộc tính được gửi này nằm trong phần header của HTTP *referer*.
- *accept-charset* (tùy chọn xâu, mặc định là *unknown*): Xác định dãy các kí tự được phân cách bởi dấu cách (hoặc dấu phẩy) có thể được mã hoá để gửi tới server trong yêu cầu POST.

❖ Thuộc tính method GET và POST

Một trong những tùy chọn thú vị của tác vụ <go> là thuộc tính *method* quy định yêu cầu được gửi đi là GET hay là POST. Tùy chọn này chỉ được sử dụng khi gửi thông tin để xử lý trên server. Nó không được sử dụng khi đơn giản chỉ là việc nhận các trang tĩnh WML. Để dàng nhận ra những thuộc tính này tương tự với thuộc tính *method* = "GET" và *method* = "POST" được đặt trong phần tử HTML <form>.

Phương thức GET

GET là một phương thức thông dụng được sử dụng trong HTTP. Tất cả các thông tin đưa tới server đều được mã hoá ở URL và server sử dụng URL này để tìm các tài nguyên trả về cho trình duyệt. Ưu điểm chính của GET là nó rất đơn giản, thậm chí có thể đặt thông tin truy vấn trực tiếp vào URL để sử dụng các biến. Nhược điểm chính của GET là nó chỉ có thể được sử dụng cho một lượng giới hạn dữ liệu. Nguyên nhân là do Web server và các chương trình khác khi xử lý đều chỉ chấp nhận một giới hạn nhất định về độ dài của một URL, điều này dẫn tới giới hạn độ dài của yêu cầu được gửi đi khi sử dụng phương thức GET.

Thêm một vấn đề không dễ mô tả với GET đó là trên thực tế tất cả các thông tin gửi đi đều trở thành một phần của URL. Nhiều trình duyệt sẽ hiển thị URL của *deck* hiện thời tại một vị trí nào đó trên màn hình (thậm chí chỉ là dạng ngắn gọn) và hầu hết các web server lưu trữ toàn bộ địa chỉ URL trong các log file, cùng với các dữ liệu thêm vào từ GET. Nếu như thông tin mang tính nhạy cảm (ví dụ như mật khẩu), nó có thể được hiển thị trên màn hình và sau đó giữ lại trên web server, gây ảnh hưởng đến tính an toàn của dữ liệu.

Chương 18

Wireless Markup Language (WML) Cơ bản

Phương thức POST

Phương thức POST là phương thức thứ hai của HTTP, nó tránh được hai vấn đề trên của phương thức GET bằng cách gửi dữ liệu tách biệt với URL. Kết quả là chúng ta có một URL ngắn gọn, thêm vào đó màn hình hiển thị của trình duyệt và các log file của web server không chứa bất kỳ một dữ liệu nào.

Trong các phiên bản mới của WML, thông tin được gửi với phương thức POST được thực hiện bởi phần tử `<postfield>` nằm trong phần tử `<go>`. Thông tin này định dạng một danh sách các cặp tên/giá trị (name/value). Mỗi phần tử `<postfield>` xác định một cặp đơn. Phần tử này đơn giản, chỉ có hai thuộc tính:

- *name* (biến xâu được yêu cầu): Tên của trường này.
- *value* (biến xâu được yêu cầu): Giá trị của trường này.

WML cho phép các phần tử `<postfield>` được sử dụng thậm chí cả với phương thức GET (hoặc có thể pha trộn cả hai). Trong sự kiện này, các trường được thêm vào cuối của phần truy vấn của URL. Sử dụng phần tử `<postfield>` theo cách này làm cho tài liệu WML rõ ràng hơn và cũng dễ dàng hơn nếu như phải sửa đổi để sử dụng POST trong tương lai.

Ví dụ 18.10

- Chúng ta sử dụng các phần tử `<postfield>` theo phương thức GET :

```
<go href="wibble" method="get">
  <postfield name="x" value="17"/>
  <postfield name="y" value="42"/>
</go>
```

- Cách viết sau cũng có kết quả tương tự:

```
<go href="wibble?x=17&y=42" method="get"/>
```

- Một cách khác viết pha trộn hai loại có kết quả giống hết hai cách trên:

```
<go href="wibble?x=17" method="get">
  <postfield name="y" value="42"/>
</go>
```

Dạng viết nhanh của các tác vụ

Thay vì việc phải bao gồm nhiều tác vụ để hoàn tất tác vụ `<go>`, WML cung cấp một dạng viết nhanh có thể sử dụng ở một vài tình huống trong đó giá trị sẽ được đặt vào thuộc tính *href* của phần tử `<go>`.

Ví dụ 18.11

Ví dụ, ràng buộc một tác vụ tới một tùy chọn trong danh mục các giá trị lựa chọn, khi đó tác vụ sẽ được thực thi khi được chọn lựa. Thông thường viết như sau:

```
<option>
<onevent type="onpick">
<go href="foo.wml"/>
</onevent>
Option text
</option>
```

Nếu sử dụng dạng viết nhanh có thể viết như sau:

```
<option onpick="foo.wml">
Option text
</option>
```

Cách viết này cũng được chấp nhận cho các thuộc tính *onenterforward*, *onenterbackward* và *ontimer* của phần tử *<card>*, thuộc tính *onpick* của phần tử *<option>* và thuộc tính *href* của phần tử *<a>*. Các phần tử này tất cả đều được xem xét cụ thể hơn ở phần sau.

18.5.1.3. Tác vụ <prev>

Tác vụ *<prev>* thực hiện việc quay trở lại *card* đã “thăm” trước đó dựa trên một ngăn xếp lưu các thông tin ghi lại quá trình làm việc của trình duyệt (*history stack*). Khi hành động này diễn ra thì phần tử đầu tiên của *stack* sẽ được lấy ra và thẻ đó sẽ được hiển thị lại lần nữa sau khi tất cả các biến đã được thiết lập giá trị bằng *<setvar>* trong tác vụ *<prev>*.

Tác vụ *<prev>* không mang thuộc tính nào. Hầu hết các ứng dụng của tác vụ *<task>* đều đơn giản. Thường thì không có biến nào được xử lý nên hầu hết các tác vụ *<prev>* đều viết đơn giản như sau: *<prev/>*.

Tác vụ *<prev>* được sử dụng thường xuyên trong mối liên kết với phần tử *<do>*. Một vài trình duyệt không cung cấp nút trở về (back button) nên tốt nhất là nên có cấu trúc WML như sau:

```
<do type = "prev">
<prev/>
</do>
```

Chương 18

Wireless Markup Language (WML) Cơ bản

Cấu trúc này đơn giản cung cấp một nút khi được kích hoạt nó sẽ gửi tới trình duyệt card trước đó.

Ví dụ 18.12

Một tình huống điển hình trong việc kết hợp các biến trong tác vụ `<prev>`, đó là đối với một trang đăng nhập yêu cầu *username* và *password*. Khi người sử dụng nhập không đúng thì yêu cầu họ nhập lại với trường *password* trống. Điều này có thể thực hiện với cấu trúc như sau:

```
<prev>
  <setvar name = "password" value = ""/>
</prev>
```

18.5.1.4. Tác vụ `<refresh>`

Tác vụ `<refresh>` là tác vụ đơn giản và có hiệu ứng ngay lập tức. Hiệu ứng của nó chỉ đơn giản là thiết lập và gán lại giá trị cho các biến được chỉ ra bởi các phần tử `<setvar>`, sau đó hiển thị lại thẻ hiện hành với các giá trị mới. Tác vụ `<go>` và `<prev>` cũng thực hiện một hành động giống nhau ngay trước khi hiển thị thẻ mới.

Tác vụ `<refresh>` không cần bất kỳ thuộc tính nào.

Ví dụ 18.13.

Tác vụ `<refresh>` thông thường được sử dụng khi muốn thực hiện một số hành động như là "reset" trên thẻ. Chẳng hạn như trong ví dụ này, phần tử `<input>` yêu cầu người dùng nhập vào các xâu và sau đó lưu trữ các xâu này vào các biến đã được xác định ở thuộc tính `<name>`. Tác vụ `<go>` sử dụng phương thức POST xác nhận các thông tin về *username* và *password* để gửi tới server xử lý.

- Trang mã WML như sau:

Chương 18

Wireless Markup Language (WML) Cơ bản

```
<wml>
<card title="Reset button example">
  <!-- Read login and password from user. -->
  <p>Login: <input name="login"/></p>
  <p>Password: <input name="password"/></p>

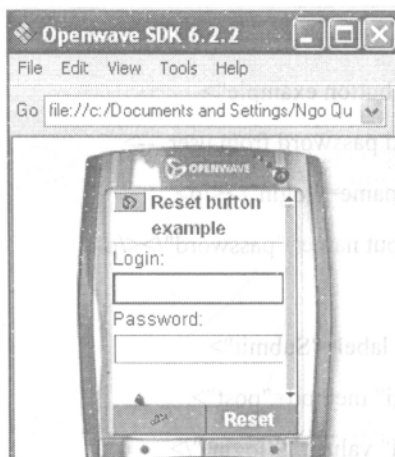
  <do type="accept" label="Submit">
    <go href="login.cgi" method="post">
      <postfield name="l" value="$(login)"/>
      <postfield name="p" value="$(password)"/>
    </go>
  </do>

  <do type="reset" label="Reset">
    <refresh>
      <setvar name="login" value=""/>
      <setvar name="password" value=""/>
    </refresh>
  </do>
</card>
</wml>
```

- Kết quả

Chương 18

Wireless Markup Language (WML) Cơ bản



Hình 18.2. Kết quả của ví dụ 18.13.

18.5.1.5. Tác vụ `<noop>`

Mục đích của tác vụ `<noop>` là không làm gì cả (không vận hành). Ứng dụng thực tế duy nhất của tác vụ này là nó liên kết với các *templates* (khuôn mẫu) (thảo luận ở phần sau). Tuy nhiên, nó có thể được sử dụng ở bất kỳ chỗ nào khi được yêu cầu.

Tác vụ `<noop>` là ngoại lệ duy nhất trong khái niệm: *các tác vụ có thể thiết lập biến*. Nghĩa là nó không thể thiết lập biến và thậm chí không thể chứa bất kỳ một phần tử `<setvar>` nào. Nếu muốn một tác vụ thiết lập các biến mà không thay đổi thể hiện tại, hãy sử dụng tác vụ `<refresh>`. Tác vụ `<noop>` cũng không cần đến bất kỳ thuộc tính nào.

Bởi vì tác vụ `<noop>` không mang thuộc tính nào và thậm chí không chứa bất kỳ một phần tử `<setvar>` nào nên nó luôn luôn ở dạng như sau:

```
<noop/>
```

18.5.2. Các sự kiện

Một sự kiện trong WML đơn giản là một điều gì đó có thể xảy ra đối với các phần tử theo thời gian. Ví dụ, truy nhập vào phần tử `<card>` sẽ kích hoạt một sự kiện trên `<card>` và việc lựa chọn một `<option>` từ danh mục chọn lựa sẽ tạo ra một sự kiện trên `<option>`.

Khai thác các sự kiện này bằng cách ràng buộc chúng vào các tác vụ. Cách thông dụng của việc làm này là sử dụng phần tử `<onevent>`.

Chương 18

Wireless Markup Language (WML) Cơ bản

Ví dụ 18.14.

Ví dụ, phần tử `<option>` khai báo một đề mục trong danh mục lựa chọn. Khi đề mục này được chọn, nó sẽ kích hoạt sự kiện *onpick* trên phần tử `<option>`. Giả dụ rằng phần tử được khai báo mà không có trình xử lý sự kiện, như sau:

```
<option>
  purple
</option>
```

Trong trường hợp này, sự kiện *onpick* được bỏ qua. Ngược lại nếu khai báo là:

```
<option>
<onevent type="onpick">
  <go href="#purple"/>
</onevent>
  Purple
</option>
```

Thì sự kiện *onpick* sẽ được xử lý bằng cách thực hiện tác vụ `<go>` và gửi tới trình duyệt một thẻ mới.

18.5.2.1. Phần tử `<onevent>`

Phần tử `<onevent>` khai báo một ràng buộc sự kiện. Nó không chứa gì ngoại trừ một phần tử tác vụ sẽ được hiển thị khi sự kiện đó xảy ra. Nó có thể xuất hiện ngay trong các phần tử `<option>` hoặc phần tử `<card>`. Trong bất kỳ trường hợp nào thì phần tử `<onevent>` phải là phần tử đầu tiên được khai báo bên trong phần tử mà nó chứa trong đó.

Phần tử `<onevent>` chỉ gồm một thuộc tính là *type* (xấu được yêu cầu): nhằm đưa ra kiểu sự kiện ràng buộc với phần tử. Ví dụ, dùng *type* = "ontimer" để định cho sự kiện *ontimer* của phần tử `<card>`.

18.5.2.2. Phần tử `card`

Phần tử này được sử dụng khi muốn thực hiện một vài sự kiện đặc biệt khi người sử dụng truy cập vào một *card* nào đó. Ví dụ như để khởi tạo và hiển thị giá trị của các biến hay là muốn xoá đi các biến khi quay trở lại một *card* nào đó trong *history stack*

Để có thể làm điều này, WML định nghĩa hai sự kiện *onenterforward* và *onenterbackward*, chúng diễn ra ở phần tử `<card>` khi người dùng truy nhập vào nó. Sự kiện diễn ra như thế nào sẽ phụ thuộc vào việc thẻ đó được đưa truy nhập vào như thế nào.

Chương 18

Wireless Markup Language (WML) Cơ bản

18.5.2.3. Sự kiện *onenterforward*

Sự kiện *onenterforward* xảy ra khi thẻ được truy nhập vào theo hướng tiến (không phải quay trở lại). Sự kiện này thông thường được sử dụng trong việc khởi tạo các giá trị trước khi *card* được hiển thị, thường thì thực hiện bằng cách sử dụng tác vụ *<refresh>* hoặc sử dụng tác vụ *<go>* để gọi script của WMLScript.

Ví dụ 18.15.

Ví dụ 18.15 cho thấy cách sử dụng sự kiện này. Khi thẻ đầu tiên được đưa vào, tác vụ *<refresh>* được thực thi, chúng khởi tạo biến *state*. Biến này sau đó có thể được cập nhật bởi các trang WML khác và được truyền tới server bằng tác vụ *<go>*.

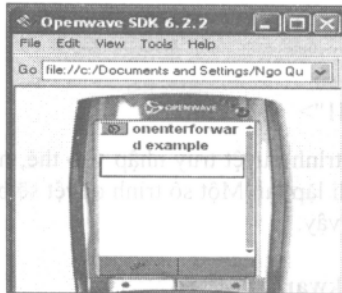
- Mã WML

```
<wml>
<card title="onenterforward example">
  <onevent type="onenterforward">
    <refresh>
      <setvar name="state" value="0"/>
    </refresh>
  </onevent>
  <p><input name="text"/></p>
  <do type="accept">
    <go href="submit.cgi">
      <postfield name="s" value="$(state)"/>
      <postfield name="t" value="$(text)"/>
    </go>
  </do>
</card>
</wml>
```

- Kết quả

Chương 18

Wireless Markup Language (WML) Cơ bản



Hình 18.3. Kết quả của ví dụ 18.15.

Nếu tác vụ ràng buộc cho sự kiện *onenterforward* là tác vụ `<go>` đơn giản mà không cần phần tử `<setvar>` hoặc `<postfield>`, thì có thể sử dụng dạng viết nhanh (đã giới thiệu trước): chỉ cần thêm thuộc tính *onenterforward* vào phần tử `<card>`. Giá trị của thuộc tính là URL đích.

Ví dụ 18.16.

Ví dụ một ràng buộc cho sự kiện *onenterforward* là tác vụ `<go>` đơn giản như sau:

```
<card title="example">
  <onevent type="onenterforward">
    <go href="#card2"/>
  </onevent>
  <!-- rest of card -->
</card>
```

Thì khi đó dùng dạng viết nhanh sẽ tương đương với dạng ngắn gọn hơn:

```
<card title="example" onenterforward="#card2">
  <!-- rest of card -->
</card>
```

Ví dụ 18.17.

Chú ý: Không phải tất cả các tác vụ ràng buộc cho sự kiện *onenterforward* đều có nghĩa. Ví dụ, sự kiện ràng buộc sau làm cho người sử dụng khi truy nhập thì thẻ trình duyệt ngay lập tức sẽ trở lại thẻ trước đó:

```
<onevent type="onenterforward"> <prev/>
</onevent>
```

Chương 18

Wireless Markup Language (WML) Cơ bản

Hãy xem thêm một đoạn mã khác dưới đây:

```
<card id="card1"
onenterforward="#card1">
```

Trong đoạn mã này ngay khi trình duyệt truy nhập vào thẻ, nó lại lập tức truy nhập lại lần nữa, quá trình này cứ thế lặp đi lặp lại. Một số trình duyệt sẽ báo lỗi nhưng không phải trình duyệt nào cũng làm được như vậy.

18.5.2.4. Sự kiện *onenterbackward*

Sự kiện *onenterbackward* cùng với sự kiện *onenterforward* là các sự kiện liên quan đến việc thẻ được quay trở lại (*backward*). Sự kiện này được chọn lựa khi thẻ bị trả lại và kết quả là tác vụ *<prev>* hoặc một số hành động khác làm quay trở lại trạng thái trước đó trong *history stack*.

Sự kiện *onenterbackward* thường được sử dụng trong việc thiết lập lại giá trị ban đầu khi quay lại phần trước đó. Ví dụ 18.18 minh hoạ điều này, nút *reset* làm cho trường *login* và *password* đều được xoá trắng khi người sử dụng quay trở lại thẻ.

Ví dụ 18.18.

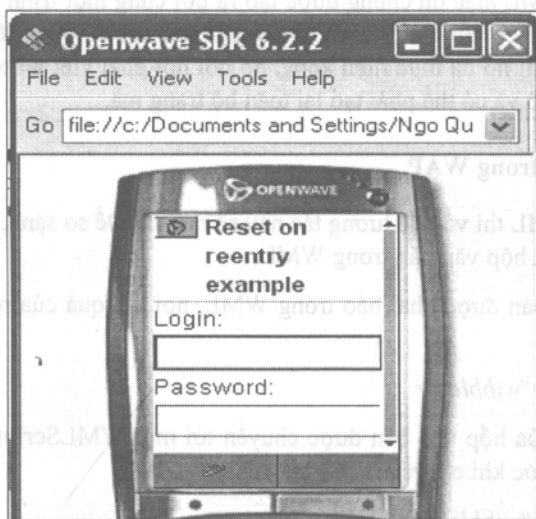
```
<wml>
<card title="Reset on reentry example">
<!-- Reset fields when entered backwards. -->
<onevent type="onenterbackward">
<refresh>
<setvar name="login" value=""/>
<setvar name="password" value=""/>
</refresh>
</onevent>
<!-- Read login and password from user. -->
<p>Login: <input name="login"/></p>
```

Chương 18

Wireless Markup Language (WML) Cơ bản

```
<p>Password: <input name="password"/></p>
<!-- Submit button sends data to server. -->
<do type="accept" label="Submit">
  <go href="login.cgi" method="post">
    <postfield name="l" value="$(login)"/>
    <postfield name="p" value="$(password)"/>
  </go>
</do>
</card>
</wml>
```

▪ Kết quả



Hình 18.4. Xóa trắng hai trường Login và Password.

18.5.3. Tương tác người dùng WML

18.5.3.1. Các vấn đề tương tác Web (Web interaction) trong HTML

Sử dụng các form HTML cho mục đích tương tác với người sử dụng đều gặp phải một vấn đề đó là quá trình xử lý phải được thực hiện trên máy chủ. Máy khách chỉ hiển thị

Chương 18

Wireless Markup Language (WML) Cơ bản

các điều khiển, tập hợp kết quả, đóng gói và gửi chúng tới máy chủ.

Có thể sử dụng Java hay Javascript để tập hợp dữ liệu và cho phép xử lý cục bộ nhưng giải pháp này cũng có nhiều vấn đề. Thêm nữa, lại có rất nhiều cách để khai báo điều khiển trong trang web. Giả sử muốn hiển thị một đoạn text trong input box. Sử dụng một form có thể sử dụng 3 cách sau:

- Cách thứ nhất: `<INPUT TYPE="TEXT" NAME="wibble">`

- Cách thứ hai là có thể sử dụng JavaScript với HTML:

`<INPUT TYPE="TEXT" NAME="wibble" ONCHANGE="wibble_chg( );">`

- Cách thứ ba là có thể sử dụng Java Applets:

`TextField wibble = new TextField ();
add (wibble);`

Mỗi một đoạn mã trong ba cách này sẽ được tham chiếu theo những cách hoàn toàn khác nhau trong trang HTML mặc dù chúng được tạo ra bởi cùng một trình duyệt và người dùng tương tác với chúng theo cùng một cách. Điều này làm cho rất khó để thay thế hành động của một điều khiển khi nó đã thực hiện xong, nó đòi hỏi phải viết lại toàn bộ những gì liên quan tới điều khiển đó và có thể phải tạo lại toàn bộ trang mã.

18.5.3.2. Tương tác trong WAP

Vậy trong WML thì vấn đề tương tác này sẽ ra sao? Để so sánh, sau đây là một ví dụ về các cách xử lý một hộp văn bản trong WML.

- Khi hộp văn bản được khai báo trong WML, nơi kết quả của nó được gửi trực tiếp tới máy chủ:

`<input name="wibble"/>`

- Khi kết quả của hộp văn bản được chuyển tới một WMLScript nào đó để kiểm tra tình hợp lệ trước khi chuyển nó tới máy chủ:

`<input name="wibble"/>`

- Và khi nó được hiển thị cho người sử dụng trong một thẻ khác để kiểm tra lại, mà không liên quan đến bất kỳ giao dịch máy chủ nào :

`<input name="wibble"/>`

Ba ví dụ này giống hệt nhau. Điều đó có nghĩa là một điều khiển luôn luôn được viết theo một cách giống nhau trong WML.

18.5.3.3. Phần tử <input>

Phần tử <input> được sử dụng khi người dùng muốn nhập một xâu hoặc một đoạn văn bản. Thông thường xâu càng ngắn càng tốt vì rất nhiều người sử dụng điện thoại để nhập văn bản. Chẳng hạn để nhập ký tự S trên bàn phím máy tính thì dễ nhưng điều này đòi hỏi 4 lần ấn phím điện thoại. Có một số ký tự thậm chí còn đòi hỏi nhiều thời gian hơn, chẳng hạn như dấu cộng “+”,..

Phần tử <input> còn được dùng để nhập *password* hay các thông tin quan trọng khác. Trong các trường hợp này, phần tử được thiết kế để không hiện đoạn văn bản hết như khi chúng được nhập vào (thông thường được thay bằng ký tự *).

Phần tử này còn được dùng để nhập các số. Trong trường hợp này miền giá trị của ký tự nhập vào được giới hạn. Để làm được điều đó, đặt định dạng chỉ cho phép nhận số.

Các thuộc tính của <input> bao gồm:

- *Name (tên điều khiển - bắt buộc)*: Đặt tên cho biến sẽ nhận đoạn văn bản từ phần tử. Khi phần tử này được thực hiện, nó được khởi tạo bằng nội dung của biến đã được đặt, nếu không là giá trị của thuộc tính *value* nếu tồn tại. Trong trường hợp thuộc tính *value* cũng không có nốt thì nó được khởi tạo là rỗng.
- *Type (Kiểu: xâu tùy chọn, mặc định là text)*: Nếu đặt kiểu là *text* thì phần tử sẽ không nhận thông tin nhạy cảm và sẽ hiển thị đúng như khi nhập. Nếu đặt kiểu là *password* thì dữ liệu nhập vào sẽ được che lấp nội dung. Điều này thông thường được thực hiện bằng cách thay các ký tự bằng một ký tự cố định nào đó.
- *Value (Biến xâu tùy chọn, mặc định là rỗng)*: Đặt giá trị khởi tạo cho đoạn text trong phần tử. Giá trị này sẽ được sử dụng khi biến gán bởi thuộc tính *Name* không được đặt giá trị. Nếu biến này chứa giá trị khác thì giá trị khởi tạo sẽ bị thay thế.
- *Format (Xâu tùy chọn, mặc định là *M)*: đặt định dạng cho giá trị:

A: Bất kỳ chữ cái hoa hay ký hiệu nào (không phải là số).

a: Bất kỳ chữ cái thường hay ký hiệu nào (không phải là số).

N: Bất kỳ 1 ký tự số nào (số hay dấu chấm thập phân).

X: Bất kỳ một ký tự hoa .

x :Bất kỳ một ký tự thường.

M: Bất kỳ một ký tự nào. Chữ hoa có thể được yêu cầu trước.

m: Bất kỳ một ký tự nào. Chữ thường có thể được yêu cầu trước.

Chương 18

Wireless Markup Language (WML) Cơ bản

- **f* : Bất kỳ một số lượng ký tự nào thỏa mãn điều kiện f (f là một trong các dạng trước: A, a, N, X, x, M, m).
- *nf* : Đầu vào gồm đúng n ký tự (n = 1...9) thỏa mãn f.
- *ic* : ký tự c được hiển thị trong trường hợp đặc biệt.
- *emptyOK* : Nếu đặt là TRUE thì một chuỗi rỗng được coi là hợp lệ.
- *Size(số tùy chọn)*.
- *Maxlength (số tùy chọn, mặc định là không giới hạn)*: Chỉ định nhiều nhất số ký tự có thể được nhập vào.
- *Title*: Đưa vào một tiêu đề cho phần tử này, một số trình duyệt có thể sẽ sử dụng trong trình chiếu.
- *Tableindex*.

Ví dụ 18.19.

Một trang đăng nhập minh họa phần tử `<input>`

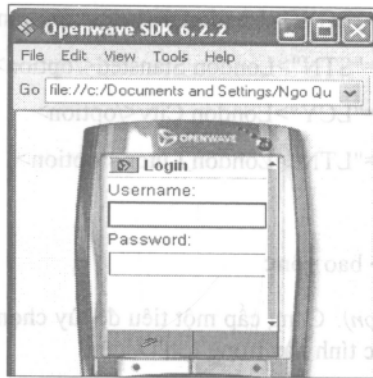
- Mã WML:

```
<wml>
<card title="Login">
  <p>Username:
  <input name="user" format="*x"/></p>
  <p>Password:
  <input name="pass" type="password"/></p>
  <do type="accept" title="Log In">
    <go href="login?u=$(user:e)&p=$(pass:e)"/>
  </do>
</card>
</wml>
```

- Kết quả:

Chương 18

Wireless Markup Language (WML) Cơ bản



Hình 18.5. Minh hoạ phần tử `<input>`.

18.5.3.4. Phần tử `<select>`

Phần tử `<select>` là một điều khiển bậc cao mà WML cung cấp. Nó là điều khiển cho phép chọn từ danh sách các thực thể. Điều khiển này thay thế cho rất nhiều điều khiển khác như *scrolling selectable lists*, *pulldown menus*, và *lists of checkboxes*.

Trong dạng đơn giản nhất `<select>` cung cấp một thuộc tính *iname*. Trong `<select>` là một danh sách các `<option>`. Chọn một *option* sẽ đặt biến *iname* bằng chỉ số của *option* đó trong `<select>`. Ví dụ :

```
<select iname="animal">
  <option>Lizard</option>
  <option>Spider</option>
  <option>Squid</option>
</select>
```

khi đó chọn *Lizard* sẽ đặt *animal* bằng 1, chọn *Spider* sẽ đặt nó bằng 2 và chọn *Squid* sẽ đặt nó bằng 3.

Trong dạng phức tạp hơn, `<select>` có thuộc tính *name* chứ không phải là *iname* và một danh sách các `<option>`, mỗi một `<option>` sẽ có một thuộc tính giá trị. Chọn một trong các *option* này sẽ đặt biến *name* bằng nội dung của thuộc tính *value*. Ví dụ một danh sách cho phép người dùng chọn sân bay ở London có thể được viết như sau :

```
<select name="airport">
  <option value="LHR">London Heathrow</option>
```

Chương 18

Wireless Markup Language (WML) Cơ bản

```
<option value="LGW">London Gatwick</option>
<option value="STN">London Stansted</option>
<option value="LCY">London City</option>
<option value="LTN">London Luton</option>
</select>
```

Các thuộc tính của `<select>` bao gồm:

- *Title (biến tùy chọn)*: Cung cấp một tiêu đề tùy chọn cho `<select>`, một vài trình duyệt sẽ sử dụng thuộc tính này trong trình chiếu.
- *iname (tên tùy chọn)*: Định rõ một biến WML được đặt bằng chỉ số của *item* được chọn. Nếu nó được định rõ và biến đã được đặt một chỉ số thì nó sẽ chọn giá trị mặc định. Nó có quyền ưu tiên cao hơn so với *ivalue*, *name* và *value* trong việc quyết định giá trị mặc định.
- *ivalue (biến số tùy chọn)*: Định rõ chỉ số của biến là giá trị mặc định nếu thuộc tính *iname* không có hoặc biến của nó không được đặt. Nó có ưu tiên cao hơn so với *name* và *value* cho việc đặt giá trị mặc định.
- *name (tên tùy chọn)*: Định rõ một biến WML sẽ được đặt bằng nội dung của biến *value* từ *option* được chọn. Nó có thể được dùng để quyết định giá trị mặc định. Nó chỉ có quyền ưu tiên cao hơn *value*.
- *Value*: Định rõ giá trị mặc định cho `<select>`. Phần tử đầu tiên có thuộc tính *value* trùng khớp sẽ được chọn. Thuộc tính này chỉ được sử dụng khi *iname*, *ivalue* hoặc *name* không quản lý việc chọn giá trị mặc định.
- *multiple (giá trị boolean tùy chọn, mặc định là FALSE)*: Nếu đặt là *TRUE* chỉ định rằng `<select>` có thể cho phép hơn một `<option>` được kích hoạt cùng lúc. Trong trường hợp này tính chất của thuộc tính *name*, *value*, *iname* và *ivalue* thay đổi nhỏ.

Đa lựa chọn

Khi `<select>` có *multiple = TRUE* nó sẽ cho phép hơn một phần tử được chọn từ danh sách cùng một lúc. Một `<select>` như vậy sẽ sử dụng các thuộc tính *name*, *value*, *iname* và *ivalue* khác so với thông thường. Quan sát ví dụ sau.

Ví dụ 18.20.

Trong ví dụ này là `<select>` chỉ cho phép chọn một phần tử trong danh sách.

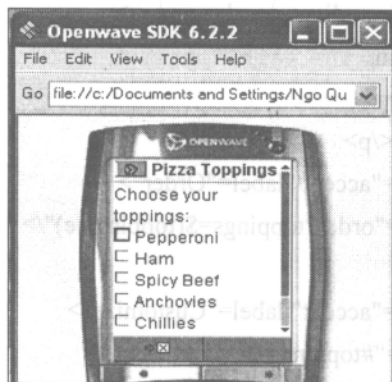
- Mã WML:

Chương 18

Wireless Markup Language (WML) Cơ bản

```
<wml>
  <card title="Pizza Toppings">
    <p>Choose your toppings:
    <select name="toppings" multiple="true">
      <option value="p">Pepperoni</option>
      <option value="h">Ham</option>
      <option value="b">Spicy Beef</option>
      <option value="a">Anchovies</option>
      <option value="c">Chillies</option>
      <option value="o">Olives</option>
      <option value="m">Mushrooms</option>
    </select></p>
    <do type="accept" label="Order">
      <go href="order?toppings=$(toppings:e)"/>
    </do>
  </card>
</wml>
```

- Kết quả:



Hình 18.6. Chọn lựa một phần tử trong danh sách.

Chương 18

Wireless Markup Language (WML) Cơ bản

Ví dụ 18.21.

Trong ví dụ này là `<select>` cho phép chọn nhiều hơn một phần tử trong danh sách.

- Mã WML:

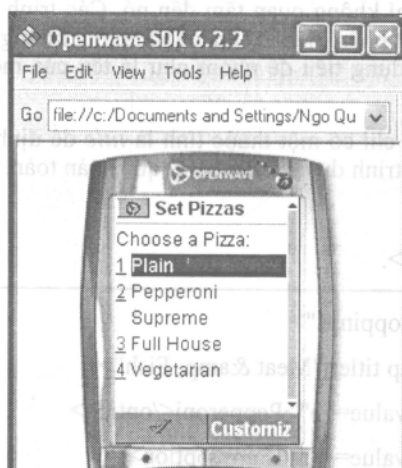
```
<wml>
  <card title="Set Pizzas" id="pizzas">
    <p>Choose a Pizza:<select name="toppings">
      <option value="">
        <!-- no toppings -->
        Plain
      </option>
      <option value="p;m;o">
        <!-- pepperoni, mushrooms, olives -->
        Pepperoni Supreme
      </option>
      <option value="p;b;h;a;m;o;c">
        <!-- everything! -->
        Full House
      </option>
      <option value="c;o;m">
        <!-- chillies, olives, mushrooms -->
        Vegetarian
      </option>
    </select></p>
    <do type="accept" label="Order">
      <go href="order?toppings=$(toppings:e)"/>
    </do>
    <do type="accept" label="Customize">
      <go href="#toppings"/>
    </do>
  </card>
```

Chương 18

Wireless Markup Language (WML) Cơ bản

```
<card title="Pizza Toppings" id="toppings">
<p>Customize your Toppings
<select name="toppings" multiple="true">
<option value="p">Pepperoni</option>
<option value="h">Ham</option>
<option value="b">Spicy Beef</option>
<option value="a">Anchovies</option>
<option value="o">Olives</option>
<option value="m">Mushrooms</option>
<option value="c">Chillies</option>
</select></p>
<do type="accept" label="Order">
<go href="order.cgi?toppings=$(toppings:e)"/>
</do>
</card>
</wml>
```

▪ Kết quả:



Hình 18.7. Đa chọn lựa.

Chương 18

Wireless Markup Language (WML) Cơ bản

18.5.3.5. Phần tử <option>

Các thuộc tính của phần tử <option>:

- *value*: Đưa ra một giá trị cho <option> và được lưu trữ trong biến *name* của <select>.
- *title*: Đưa ra một tiêu đề cho <option>. Một số trình duyệt sẽ dùng thuộc tính này để hiển thị.
- *onpick (biến URL tùy chọn)*: Cung cấp một phương thức tốc ký cho sự kiện *onpick*. Trong trường hợp bình thường nhiệm vụ chỉ đơn giản là *go*. Vì nhiệm vụ này bắt buộc phải là URL nên nó không thể chứa bất kỳ phần tử <setvar> hay <postfield> nào và phương thức của nó được giới hạn là *GET*. Trong trường hợp phức tạp, đặt phần tử <onevent="onpick"> của kiểu *onpick* bên trong phần tử <option>.

18.5.3.6. Phần tử <optgroup>

WAP không định nghĩa cách phần tử <select> được hiển thị. Nó được thực hiện theo rất nhiều cách, bao gồm cả việc sử dụng menu thả xuống, danh sách cuộn và các hàng hộp chọn trên thiết bị kiểu PDA và một vài kiểu khác của menu trên *cell phones*.

Với một màn hình hiển thị nhỏ không thể hiển thị tất cả các lựa chọn có sẵn cùng một lúc. Có một vài cách giải quyết vấn đề này, nếu các tùy chọn được hiển thị dưới dạng text bình thường thì trang web sẽ có tính năng cuộn lên, cuộn xuống. Rất nhiều *cell phones* chỉ hiển thị các lựa chọn một cách đơn giản.

Mục đích của phần tử <optgroup> là để chia một danh sách dài thành các phần. Các trình duyệt khác nhau có thể sử dụng thông tin này theo những cách khác nhau, nhiều trình duyệt đơn giản còn thậm chí không quan tâm đến nó. Các trình duyệt khác có thể hiển thị tiêu đề của nhóm như là một phần hiển thị lựa chọn nhưng không làm gì hơn với chúng. Vài trình duyệt khác có thể sử dụng tiêu đề nhóm như là tên của menu con, với nội dung của nhóm trong menu con đó.

Phần tử <optgroup> chỉ có một thuộc tính là *title* để định rõ tiêu đề của nhóm này, nó có thể được sử dụng bởi trình duyệt hoặc bị bỏ qua hoàn toàn.

Ví dụ 18.22.

Ví dụ về phần tử <optgroup>.

```
<select name="toppings">
  <optgroup title="Meat & Fish">
    <option value="p">Pepperoni</option>
    <option value="h">Ham</option>
    <option value="b">Spicy Beef</option>
    <option value="a">Anchovies</option>
```

Chương 18

Wireless Markup Language (WML) Cơ bản

```
</optgroup>
<optgroup title="Vegetables">
<option value="o">Olives</option>
<option value="m">Mushrooms</option>
<option value="c">Chillies</option>
/optgroup>
</select>
```

18.5.3.7. Phần tử <do>

Các phần tử `<select>` và `<input>` cung cấp cho người dùng các điều khiển bậc cao nhưng đôi khi bạn cần chỉ một nút hoặc một menu đơn giản. Phần tử `<do>` chính là một cách đơn giản để định rõ vài kiểu điều khiển bất kỳ mà trình duyệt tạo sẵn cho người sử dụng. Nó có thể trả lại là một nút đồ họa (như các trình duyệt PDA làm), là một item trong một menu (như hầu hết các trình duyệt trong *cell phones* làm) hoặc là bất kỳ thứ gì mà người dùng biết và tương tác.

Một phần tử `<do>` không chứa gì ngoài nhiệm vụ phải thực hiện khi phần tử được kích hoạt.

Các thuộc tính của phần tử `<do>` định rõ kiểu của phần tử `<do>` gồm một số giá trị cơ bản sau :

- *accept* : Chấp nhận cái gì đó. Cell phones thường kết nối nó với phím "yes" nếu như chỉ có một phần tử `<do>` kiểu này trong thẻ.
- *prev*: Điều chỉnh về trước.
- *help*: Giúp đỡ, có thể là giúp đỡ theo ngữ cảnh.
- *reset*: Một thao tác xoá hoặc *reset* trạng thái của tương tác.
- *options*: một yêu cầu ngữ cảnh cho những lựa chọn bổ sung. Ví dụ: một nút để đưa ra menu "*lựa chọn bổ sung*".
- *delete*: Xóa một phần tử /lựa chọn. Nếu muốn xóa nhiều phần tử thì dùng *Reset*.
- *unknown*: một phần tử tổng quát mà không hợp lý với bất kỳ lĩnh vực nào.
- *label* : Định rõ một nhãn văn bản tùy chọn cho phần tử.
- *name*: Định rõ tên của phần tử này.
- *optinal* (tùy chọn kiểu *boolean*, mặc định là *FALSE*): nếu là *TRUE* thì thông báo là trình duyệt có thể bỏ qua phần tử `<do>`.

Chương 18

Wireless Markup Language (WML) Cơ bản

18.5.3.8. Phần tử <anchor>

Phần tử <do> rất hữu ích nhưng nó không phải luôn luôn cần thiết. Nhiều *cell phones* đặt tất cả các phần tử <do> trong một thẻ trên một menu đơn, điều đó có nghĩa là không thể đảm bảo rằng nó sẽ xuất hiện khi cần thiết.

Tất cả các trang Web đều sử dụng siêu liên kết văn bản hoặc hình ảnh để kích hoạt liên kết sang một phần khác bằng thẻ HTML là <A>. Các trình duyệt Web luôn luôn hiển thị các mối liên kết này bằng cách gạch chân hoặc thể hiện bằng các màu sắc khác để người sử dụng nhận biết.

WML cũng có siêu liên kết. Siêu liên kết trong WML được đặt tới một trang sử dụng thẻ <anchor>. Nó chỉ có một thuộc tính là *Title (xâu tùy chọn)* cung cấp một tiêu đề tùy chọn cho phần tử. Như thuộc tính *label* trong phần tử <do> bạn nên chọn xâu này có tối đa 6 ký tự.

Phần tử <anchor> chứa 2 nội dung, bao gồm: nhiệm vụ mà phần tử sẽ thực hiện khi được kích hoạt và trang text/hình ảnh gắn với liên kết. Ví dụ:

```
<anchor>next page<go href="page17.wml"/></anchor>
```

Nhiệm vụ chủ yếu sử dụng phần tử <anchor> là nhiệm vụ <go>. Đối với các loại nhiệm vụ khác, thường sử dụng phần tử <do>. Ví dụ nếu muốn một điều khiển để kích hoạt <prev>, nên sử dụng phần tử <do> với type được đặt bằng *prev*.

18.5.3.9. Phần tử <a>

Nếu bạn đã từng làm nhiều về HTML thì phần tử <anchor> có vẻ sử dụng dài dòng hơn, mặc dù nó linh hoạt hơn so với thẻ <A> của HTML. Có thể lấy ví dụ sau: rõ ràng sử dụng câu lệnh HTML ngắn gọn:

```
<A HREF="somewhere">linked text</A>
```

so sánh với câu lệnh WML sử dụng phần tử <anchor> :

```
<anchor><go href="somewhere"/>linked text</anchor>
```

Những người thiết kế ra WML cũng nhận ra điều này và đã cung cấp cho WML phần tử <a>. Nó có 2 thuộc tính :

- *title (xâu tùy chọn)*: Nó có tác dụng giống như thuộc tính title trong phần tử <anchor>. Nó cung cấp một tiêu đề tùy chọn cho phần tử.
- *href*: Định rõ URL sẽ chỉ đến khi liên kết được kích hoạt.

Ví dụ câu lệnh Next Page là tương đương với:

Chương 18

Wireless Markup Language (WML) Cơ bản

```
<anchor title="Next">  
    <go href="page17.wml"/>  
    Next Page  
</anchor>
```

Form sử dụng phân tử `<a>` hiệu quả hơn so với form sử dụng `<anchor>` vì phải truyền tới trình duyệt ít hơn.

18.5.3.10. Thuộc tính `tabindex`

Các trình duyệt có thể quay vòng theo chu trình giữa `<input>` và `<select>` trong một thẻ có sử dụng các phím TAB hoặc các điều khiển tương tự. Thông thường thứ tự này được chọn bởi trình duyệt và thường là thứ tự mà các điều khiển được đưa vào thẻ. Thuộc tính `tabindex` cho phép thay đổi thứ tự này.

Không phải tất cả các trình duyệt Web đều hỗ trợ đặc tính này vì nó không thích hợp với tất cả các tương tác người sử dụng. Nếu đặc tính này được hỗ trợ, thì trình duyệt sẽ sắp xếp tất cả các phân tử `<input>` và `<select>` theo thứ tự tăng dần do `tabindex` chỉ định. Ấn phím TAB để chọn các phân tử theo thứ tự này. Phân tử nào không để `tabindex` thì sẽ được chọn sau.

Ví dụ 18.22.

- Mã WML:

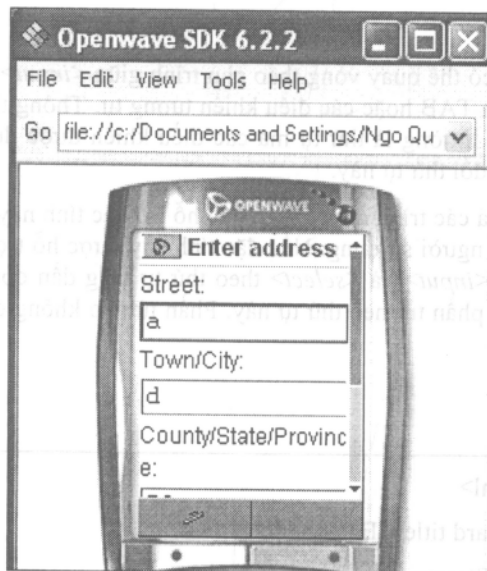
```
<wml>  
    <card title="Enter address">  
        <p>Street:  
        <input name="street" tabindex="1"/></p>  
        <p>Town/City:  
        <input name="town" tabindex="2"/></p>  
        <p>County/State/Province:  
        <input name="county"/></p>  
        <p>Postal/Zip Code:  
        <input name="code" tabindex="3"/></p>  
        <p>Nation:  
        <input name="nation" value="uk"/></p>
```

Chương 18

Wireless Markup Language (WML) Cơ bản

```
<do type="accept" label="OK">
  <go href="address.wmlsc#check()"/>
</do>
</card>
</wml>
```

- Kết quả:



Hình 18.8. Minh họa thuộc tính TabIndex.

18.6. Kết chương

Trong chương này đã giới thiệu về *Wireless Markup Language (WML)* là một ngôn ngữ xây dựng phục vụ việc hiển thị trên thiết bị không dây. Ngôn ngữ WML có nhiều điểm tương đồng với HTML nhưng hai ngôn ngữ này cũng có nhiều điểm khác nhau. Các bạn đã tìm hiểu về thẻ, biến WML và các ngữ cảnh, tác vụ và các sự kiện của WML, cũng như tương tác người dùng WML. Trong chương sau chúng ta sẽ tiếp tục tìm hiểu sâu hơn về WML.

Chương 19

WML cơ bản (tiếp theo)

Nội dung của chương:

- Tìm hiểu về các phần tử của WML.
- Văn bản trong WML và định dạng văn bản.
- Sử dụng bảng, ảnh trong WML.

19.1. Bộ định thời gian trong WML (WML Timer)

19.1.1. Khái niệm Timer trong WML

Một thẻ WML có thể có bộ định thời gian. Bất kỳ khi nào thẻ được nhập, bộ định thời gian sẽ khởi tạo tại giá trị *timeout* và bắt đầu hoạt động. Khi bộ định thời gian hoạt động, nó sẽ đếm ngược cho đến 0 hoặc là dừng lại khi tác vụ được thi hành.

- Nếu một tác vụ được thi hành, *timer* sẽ ngưng trước.
- Nếu đếm đến 0 thì sự kiện *ontimer* được kích hoạt trên thẻ chứa bộ định thời gian.

Ví dụ 19.1.

Ví dụ này sẽ hiện lên một logo, bộ định thời gian sẽ đợi một phần mười giây.

- Mã WML:

```
<wml>
  <card title="Welcome" ontimer="#main">
    <!-- Timer sẽ đợi một phần mười giây. -->
    <timer value="10"/>
```

Chương 19

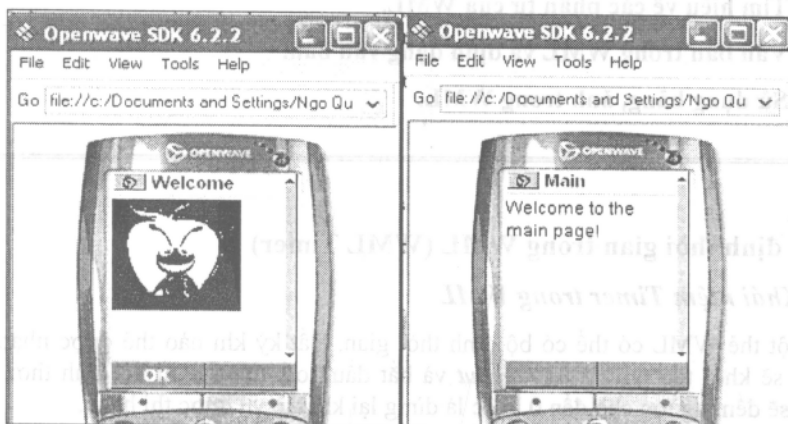
WML cơ bản (tiếp theo)

```
<p></p>
<do type="accept"><go href="#main"/></do>
</card>

<card title="Main" id="main">
  <p>Welcome to the main page!</p>
</card>

</wml>
```

- Kết quả:



Hình 19.1. Ví dụ hiện lên Logo.

Sự kiện *Ontimer* được kích hoạt khi một thẻ định thời gian đếm đến 0, nghĩa là nó sẽ không xảy ra nếu như bộ định thời được khởi tạo tại giá trị là timeout bằng 0.

Ví dụ 19.2.

Ví dụ cách để trở về thẻ trước khi bộ định thời gian hết hiệu lực như sau:

```
<onevent type="ontimer">
  <prev/>
</onevent>
```

với sự kiện *onenterforward* và *onenterbackward* form có thể viết ngắn gọn như sau:

```
<card title="simple go">
```

```
<onevent type="ontimer">
  <go href="foo.wml"/>
</onevent>
</card>
```

19.1.2. Phần tử <Timer>

Một bộ định thời gian được khai báo trong một thẻ WML với phần tử <timer>. Nó phải theo sau các phần tử <onevent>. Nếu không có phần tử <onevent> nào thì <timer> phải là phần tử đầu tiên trong <card>. Trong một thẻ chỉ có nhiều nhất một bộ định thời gian.

Phần tử <timer> có hai thuộc tính:

- *name*(Tên tùy chọn): tên biến sử dụng khi khởi tạo bộ định thời gian. Nếu biến này được thiết lập thì giá trị của biến đó sẽ được sử dụng như là *timeout*. Ngoài ra khi *timer* ngừng hoạt động, lượng thời gian còn lại sẽ được lưu trữ trong biến này. *Timeout* được đặt với đơn vị là 1/10 của 1 giây, nhưng một vài trình duyệt có thể làm tròn giá trị này. Ví dụ một trình duyệt có thể sẽ chỉ tính tổng số giây và làm tròn *timeout* gần một giây. Nếu giá trị khởi tạo của *timeout* không là một số nguyên dương thì nó sẽ bị bỏ qua.
- *value*(Bắt buộc là giá trị số): cung cấp giá trị mặc định cho *timeout*. Giá trị này được sử dụng nếu thuộc tính *name* không tồn tại hoặc tên biến dành cho *name* không được đặt. Nếu *name* tồn tại và tên biến được đặt thì thuộc tính này sẽ bị bỏ qua.

Tác vụ <refresh> có liên quan đặc biệt đến bộ định thời gian. Thực hiện tác vụ <refresh> sẽ làm ngưng bộ định thời và giá trị còn lại của nó sẽ được lưu vào biến *name* nếu tồn tại, tuy nhiên khi tác vụ <refresh> hoàn thành trình duyệt vẫn ở trạng thái đó và bộ định thời được khởi tạo lại.

Ví dụ 19.3.

Ví dụ sau chỉ ra cách một tác vụ <refresh> hiển thị ba ảnh khác nhau. Ảnh thứ nhất hiển thị trong 1/2 giây, hai ảnh sau mỗi cái một giây.

Thực hiện điều này bằng cách gán tên ảnh vào ba biến *img0*, *img1*, *img2*, và *timeouts* vào *time0*, *time1*, *time2*. *img0* và *time0* xác định ảnh hiện tại, *img1* và *time1* xác định ảnh tiếp theo. Để ý rằng, bộ định thời gian không tham chiếu biến *time0* chỉ đơn giản bằng tên thuộc tính của nó:

```
<timer name="time0"/>
<timer value="$time0"/>
```

Chương 19

WML cơ bản (tiếp theo)

Đặt $time0 = 0$ khi bộ định thời gian hết hiệu lực và hết vòng lặp

- Mã WML

```
<wml>
<card title="Three Images">
<onevent type="onenterforward">
<refresh>
<setvar name="img0" value="first.wbmp"/>
<setvar name="img1" value="second.wbmp"/>
<setvar name="img2" value="third.wbmp"/>
<setvar name="time0" value="5"/>
<setvar name="time1" value="10"/>
<setvar name="time2" value="10"/>
</refresh>
</onevent>
<onevent type="ontimer">
<refresh>
<setvar name="img0" value="$(img1)"/>
<setvar name="img1" value="$(img2)"/>
<setvar name="img2" value="$(img0)"/>
<setvar name="time0" value="$(time1)"/>
<setvar name="time1" value="$(time2)"/>
<setvar name="time2" value="$(time0)"/>
</refresh>
</onevent>
<timer value="$(time0)"/>
<p></p>
</card>
</wml>
```

Chương 19

WML cơ bản (tiếp theo)

Ví dụ 19.4.

Ví dụ dưới đây chỉ ra cách thuộc tính *name* hoạt động. Bộ định thời gian được khởi động với *timeout* là một phút. Vào bất kỳ lúc nào người dùng có thể kích hoạt phần tử `<do>` và tới thẻ thứ hai, thẻ này sẽ thông báo thời gian còn lại là bao nhiêu. Trở lại thẻ thứ nhất khởi động lại khi nó hết hiệu lực. Khi bộ định thời gian cuối cùng hết hiệu dụng, người dùng chuyển nó sang thẻ thứ ba.

- Mã WML:

```
<wml>

  <card title="First card" id="one" ontimer="#three">
    <timer name="timeout" value="600"/>
    <p>The timer is running...</p>
    <do type="" label="Check"><go href="#two"/></do>
  </card>

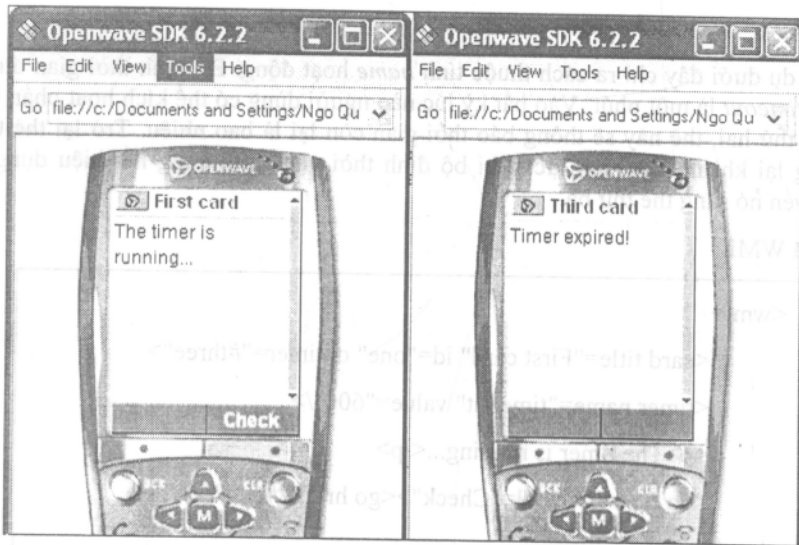
  <card title="Second card" id="two">
    <p>Time remaining:
    (timeout) tenths of a second</p>
    <do type="prev" label="Back"><prev/></do>
  </card>

  <card title="Third card" id="three">
    <p>Timer expired!</p>
  </card>
</wml>
```

- Kết quả:

Chương 19

WML cơ bản (tiếp theo)



Hình 19.2. Minh hoạ ví dụ 19.4.

19.2. Các phần tử của WML

19.2.1. Phần tử `<wml>`

Phần tử `<wml>` có vai trò giống thẻ `<HTML>` trong trang HTML. Nó bao trọn toàn bộ một tập hợp các thẻ (deck).

19.2.2. Phần tử `<head>`

Phần tử `<head>` trong WML tương tự như phần tử `<HEAD>` trong HTML, nó là dấu hiệu chỉ ra vị trí cho phần thông tin “thô” (Meta-information) của tài liệu được lưu trữ. Thông tin thô chỉ đơn thuần là thông tin về bản thân tài liệu chứ không phải là toàn bộ nội dung của nó. Nếu phần tử này xuất hiện thì nó phải được đặt làm phần tử đầu tiên trong phần tử `<wml>`.

Phần tử này không có thuộc tính nào.

19.2.3. Phần tử `<access>`

Phần tử `<access>` cung cấp một form đơn giản để kiểm soát truy nhập cho một deck, điều này cho phép một deck chỉ rõ các deck có thể liên kết tới nó (những deck này được biết tới như một URLs tham chiếu). Do sự kiểm soát này chỉ được thực thi trong trình duyệt, không phải phía server, nên nó không đảm bảo an toàn và chỉ được sử dụng rất hạn chế.

Có thể có không nhiều hơn một phần tử `<access>` trong một deck và nó phải được đặt vào vị trí đầu tiên trong phần tử `<head>`.

Chương 19

WML cơ bản (tiếp theo)

Thuộc tính của phần tử <access>

- *domain (xâu kí tự; tuỳ chọn; mặc định là domain của deck)*: Chỉ ra tên miền từ đó các *deck* được liên kết tới. Nếu thuộc tính này bị bỏ qua thì giá trị mặc định của nó hướng tới server từ một địa chỉ URL mới của *deck*. Ví dụ: Nếu thuộc tính *domain*="wml.wap.net" thì :

wml.wap.net : Tương thích, được quyền truy nhập.

www.wml.wap.net: Tương thích, được quyền truy nhập.

www.test.wml.wap.net : Tương thích, được quyền truy nhập.

wap.net: Không tương thích, không có quyền truy nhập.

otherwml.wap.net: không tương thích với wml, không có quyền truy nhập.

- *path (xâu kí tự; tuỳ chọn; mặc định là /)*: chỉ ra đường dẫn tương thích với địa chỉ URL được tham chiếu tới. Nếu thuộc tính này bị bỏ qua, nó được gán giá trị mặc định là "/" (có nghĩa là tất cả các file trên bất kì server nào cũng tương thích với thuộc tính domain). Nếu nó được nêu ra mà không có kí tự "/" bắt đầu, có nghĩa là thuộc tính này tương ứng với đường dẫn từ một URL mới của *deck*. Ví dụ như: nếu thuộc tính *path*="/foo" thì:

/foo/deck.wml: Tương thích, được phép truy nhập.

/foo/bar/deck.wml: Tương thích, được phép truy nhập.

/bar/deck.wml: Không tương thích, không có quyền truy nhập.

/foobar/deck.wmlfoobar: không tương thích với foo, không có quyền truy nhập.

19.2.4. Phần tử <meta>

Phần tử *meta* đặt phần thông tin "thô" tuỳ ý trong một WML *deck*. Mục này cấu trúc theo dạng *tên thuộc tính và giá trị*. Có thể đặt số lượng bất kì phần tử <meta> bên trong phần tử <head> như những từ khoá chỉ dẫn, lưu lại gợi ý về nội dung của *deck*, hoặc lưu bất kì thông tin nào khác. Không một thuộc tính tiêu chuẩn nào được định nghĩa cho WML, nhưng những quy ước sẽ được phát triển, giống như đã làm với HTML.

Thuộc tính của phần tử <meta>:

- *name (xâu kí tự; tuỳ chọn)*: Đưa ra tên của thuộc tính này dùng cho ứng dụng phía server, bởi vậy nó có thể loại bỏ trước khi gặp trình duyệt.
- *http-equiv (xâu kí tự; tuỳ chọn)*: Một sự lựa chọn khác cho thuộc tính *name*. Ví dụ phần tử này đề cập tới bất kì *proxy server* hoặc *gateway* không lưu *deck*.

<meta http-equiv="Cache-control" content="no-cache"/>

Chương 19

WML cơ bản (tiếp theo)

- *forua* (kiểu *boolean*; tùy chọn): nếu xuất hiện và được thiết lập bằng *true*, nó cho biết thuộc tính được dùng cho hoạt động của trình duyệt.
- *content* (xâu kí tự; tùy chọn): Chỉ ra giá trị của thuộc tính. Thuộc tính này phải được cung cấp, phần tử *<meta>* sẽ không còn ý nghĩa nếu không có thuộc tính này.
- *scheme* (xâu kí tự; tùy chọn): Có thể chỉ ra một định dạng hoặc một cấu trúc mà một số thuộc tính có thể cần thiết để hiểu giá trị của chúng. Thuộc tính này được sử dụng bởi một số thuộc tính khác.

19.2.5. Phần tử *<card>*

Đã làm quen từ ví dụ đầu tiên bắt đầu tìm hiểu về WML, phần tử *<card>* nằm trong một thẻ WML bên trong một *deck*. Văn bản và hình ảnh được nằm trong phần tử *<p>*.

Thuộc tính của phần tử *<card>*

- *title* (xâu kí tự; tùy chọn): Đưa ra một tiêu đề cho thẻ. Tiêu đề này được hiển thị theo nhiều cách khác nhau tùy thuộc vào các trình duyệt.
- *newcontext* (*boolean*; tùy chọn; mặc định là *false*): Chỉ ra rằng khi thẻ được truy nhập, ngữ cảnh trình duyệt nên được xoá, tất cả các biến liên quan được loại bỏ, *stack* cũng được xoá. Quá trình này xảy ra trước khi sự kiện *onenterforward* kích hoạt và các biến mới có thể được thiết lập.
- *ordered* (*boolean*; tùy chọn; mặc định *true*): Đưa ra những gợi ý để trình duyệt tổ chức các thẻ một cách có trật tự. Thuộc tính này sẽ được khởi gán bằng *true* nếu thẻ này chứa một số lượng các trường phân cách và các trường này có thứ tự như thứ tự của chúng trong thẻ. Giá trị được thiết lập bằng *false* nếu thẻ chứa những trường lựa chọn hoặc có thể được ghi vào mà không theo thứ tự.
- *onenterforward* (biến url; tùy chọn)
- *onenterbackward* (biến url; tùy chọn)
- *ontimer* (biến url; tùy chọn)

Ba thuộc tính cuối là phương pháp viết gọn bên trong phần tử *<card>*. Có thể sử dụng những phương pháp đó khi tác vụ được thi hành chỉ là một tác vụ *<go>* đơn giản.

19.2.6. Phần tử *<template>*

Phần tử *<template>* cung cấp một đặc điểm thú vị mà bạn sẽ không tìm thấy trong HTML. Khi viết một *deck* chứa một vài thẻ, có thể thấy những phần của cấu trúc chung cho

Chương 19

WML cơ bản (tiếp theo)

tất cả các thẻ. Ví dụ, mỗi thẻ có một điều khiển “*Back*” và một điều khiển “*home*”. Những điều khiển này thông thường được thi hành như là một phần tử `<do>`. Vì thế chúng ta nên xác định những thành phần chung này một lần cho một thẻ hơn là xác định cho tất cả các thẻ. Để làm được điều này sử dụng phần tử `<template>`.

Phần tử `<template>` có thể chứa bất kì số lượng phần tử `<do>` để cung cấp các kiểm soát một cách đa dạng. Nó cung cấp ba ràng buộc `<onevent>` với một trong số những sự kiện sau: *enterforward*, *enterbackward* và *timer*.

Shadowing (che lấp)

Nếu muốn kiểm soát hầu hết (nhưng không phải toàn bộ) các thẻ trong một *deck*, hoặc nếu muốn ràng buộc một sự kiện chỉ cho một vài thẻ thì phải làm thế nào? Bất kì thẻ nào cũng có thể thiết lập lại mức của nó trong *deck* hoặc xóa bỏ hay ràng buộc một sự kiện mới. Quá trình thiết lập lại được gọi là **shadowing (che)**.

- Để *che* đi một sự kiện ràng buộc, có nghĩa là thiết lập lại nó với một ràng buộc sự kiện khác. Việc che một sự kiện kết nối trong một thẻ rất đơn giản, chỉ cần thêm một ràng buộc mới cho sự kiện đối với thẻ và ràng buộc này mang quyền ưu tiên trên kết nối của *deck* từ phần tử `<template>`.
- Để loại bỏ một ràng buộc trong một thẻ, *che* nó với một sự kiện kết nối có chứa một tác vụ `<noop>`.

Thuộc tính của phần tử `<template>`

- *onenterforward* (biến url; tùy chọn)
- *onenterbackward* (biến url; tùy chọn)
- *ontimer* (biến url; tùy chọn)

Ba thuộc tính này giống các phần tử `<card>`, là cách viết gọn cho sự kiện ràng buộc. Có sự khác biệt giữa mức *deck* và mức thẻ ở chỗ mức *deck* cao hơn trong việc thiết lập mức.

Ví dụ 19.5.

Ví dụ sau đưa ra một template đơn giản. Thêm một điều khiển “*Back*” để điều khiển mỗi thẻ trong *deck* và cũng điều khiển một mức sự kiện kết nối của *deck*. Bởi vậy biến *secret* bị xóa tại bất kì đâu và tại bất kì thẻ nào trong *deck* được nhập trước.

- Mã WML

Chương 19

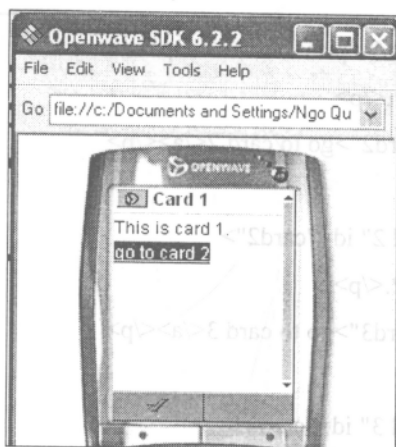
WML cơ bản (tiếp theo)

```
<wml>
<template>
<do type="prev" title="Back"><prev/></do>
<onevent type="onenterforward">
<refresh>
<setvar name="secret" value=""/>
</refresh>
</onevent>
</template>
<card title="Card 1" id="card1">
<p>This is card 1.</p>
<p><a href="#card2">go to card 2</a></p>
</card>
<card title="Card 2" id="card2">
<p>This is card 2.</p>
<p><a href="#card3">go to card 3</a></p>
</card>
<card title="Card 3" id="card3">
<p>This is card 3.</p>
<p><a href="somewhere_else.wml">
go somewhere else</a></p>
</card>
</wml>
```

- Kết quả:

Chương 19

WML cơ bản (tiếp theo)



Hình 19.3. Minh họa ví dụ 19.5.

Ví dụ 19.6.

Ví dụ 19.6 mở rộng từ ví dụ 19.5 bằng cách cùng lúc các phần tử `<do>` và các sự kiện ràng buộc tại mức *deck* có thể được *che* để ghi đè ảnh hưởng của chúng lên các thẻ cá nhân. Tại đây thẻ đầu tiên không có nút *Back* và thẻ thứ ba không xoá biến *secret* khi nhập.

- Mã WML

```
<wml>
<template>
  <do type="prev" title="Back" id="prev-ctrl">
    <prev/>
  </do>
  <onevent type="onenterforward">
    <refresh>
    <setvar name="secret" value=""/>
    </refresh>
  </onevent>
</template>
<card title="Card 1" id="card1">
```

Chương 19

WML cơ bản (tiếp theo)

```
<do type="prev" id="prev-ctrl"><noop/></do>
<p>This is card 1.</p>
<p><a href="#card2">go to card 2</a></p>
</card>
<card title="Card 2" id="card2">
<p>This is card 2.</p>
<p><a href="#card3">go to card 3</a></p>
</card>
<card title="Card 3" id="card3">
<onevent type="onenterforward"><noop/></onevent>
<p>This is card 3.</p>
<p><a href="somewhere_else.wml">
go somewhere else</a></p>
</card>
</wml>
```

19.3. Văn bản trong WML và định dạng văn bản

Trong các phần trước chúng ta đã biết cách thức tạo ra các ứng dụng tương tác trong WML, cách sử dụng biến, tác vụ và các sự kiện để xây dựng các ứng dụng. Bây giờ chúng ta sẽ quay trở lại xem xét lại một nội dung khác, đó là việc thao tác với các văn bản đơn giản và cách làm cho chúng trở nên hấp dẫn hơn. Trong lĩnh vực này thì WML không thể so sánh được với HTML - một ngôn ngữ cung cấp tất cả các tính năng giúp ta có thể hiệu chỉnh kích cỡ, kiểu, bề mặt, màu sắc của chữ, các loại khuôn dạng bảng,...

Hầu hết các trình duyệt dành cho điện thoại di động không chấp nhận các định dạng văn bản, do đó một số lớn người dùng máy điện thoại di động sẽ không nhìn thấy toàn bộ các kiểu định dạng văn bản mà bạn cung cấp cho họ. Hiệu ứng văn bản chỉ có thể sẽ được sử dụng trên PDAs (thiết bị hỗ trợ cá nhân kỹ thuật số) và các thiết bị điện thoại di động thông minh trong tương lai. Thực ra ở đây chỉ đề cập đến việc trang trí một chút cho văn bản vì với những trang trí nhỏ này thì mọi thứ vẫn có thể thể hiện tốt trên trình duyệt.

19.3.1. Phần tử <p>

Chúng ta đã biết phần tử <p> đánh dấu vị trí một đoạn văn bản trong một thẻ WML. Tất cả phần chính của văn bản, điều khiển người dùng và tất cả các ảnh phải xuất hiện trong một phần tử <p>. Hầu như các trình duyệt đều bỏ qua những thứ ở ngoài <p>, nhưng một số loại bỏ *deck* và từ chối hiển thị nó. Đặc biệt phần tử <do> không tuân theo quy luật này, nó có thể xuất hiện ở trong hay ngoài <p>.

Thông thường, phần tử <p> được sử dụng không cần thuộc tính đi kèm, kết quả là văn bản được căn lề trái và được gói trong các dòng sao cho vừa với kích thước màn hình. Phần tử <p> cũng có thể có hai thuộc tính để điều khiển cách hiển thị của văn bản (căn lề và các trình diễn text trên dòng).

Một đoạn văn bản không chứa gì ngoài kí tự trắng được coi như không có ý nghĩa, và được trình duyệt bỏ qua. Thậm chí là toàn bộ các thuộc tính của chúng được bỏ qua (thông thường thuộc tính *mode* xác định một đoạn con đến khi gặp một thuộc tính *mode* khác, nhưng điều này không xảy ra đối với đoạn văn bản không có ý nghĩa). Một số *WAP gateway* có thể loại những đoạn văn bản không có ý nghĩa ra khỏi *deck* trước khi gửi nó tới trình duyệt.

Thuộc tính của <p>

- *align* (xâu kí tự tùy chọn; mặc định là *left*): Căn lề văn bản trong đoạn văn, dưới đây là các giá trị cho thuộc tính này:

left: Căn lề từ mép trái.

right: Căn lề từ mép phải.

center: Căn lề giữa.

Lưu ý rằng không có cách nào đưa ra một văn bản cùng lúc được căn lề trái và phải.

- *Mode* (xâu kí tự tùy chọn; mặc định là *mode* của đoạn gần nhất): Chỉ ra văn bản trong đoạn có được bao phủ hay không.

Nếu thiết lập bằng *wrap*, văn bản bị cắt thành các dòng nhỏ hơn phù hợp với màn hình điện thoại di động.

Nếu thiết lập *nowrap*, điều trên không xảy ra và người dùng có một thanh cuộn để xem hết các dòng văn bản dài vượt quá kích thước màn hình. Một số trình duyệt bỏ qua thuộc tính này, có lẽ bởi vì chúng không muốn gặp nhiều vấn đề về điều khiển thanh cuộn.

Chương 19

WML cơ bản (tiếp theo)

Nếu thuộc tính *mode* không được đưa ra, đoạn văn sẽ có thuộc tính *mode* giống với đoạn văn gần nhất với nó. Tương tự đối với *align*, trả về giá trị thiết lập bằng *left* nếu không được đưa ra.

19.3.2. Phần tử `<br>`

Phần tử `<br>` là phần tử đơn giản nhất trong WML. Nó không có thuộc tính và luôn được chỉ định như một phần tử tag rỗng `<br/>`. Nó đánh dấu một dòng bị ngắt trong một đoạn văn của văn bản. Khi trình duyệt gặp một `<br>`, nó bắt đầu một dòng mới.

Có gì khác biệt giữa sử dụng một `<br/>` để xuống dòng chẳng hạn như

```
<p>text<br/>more text</p>
```

với việc sử dụng một đoạn văn bản khác như

```
<p>text</p><p>more text</p>?
```

Câu trả lời là WAP không chỉ ra sự khác biệt. Một số trình duyệt chèn một lượng nhỏ các kí tự trống vào giữa các đoạn văn nhưng không làm như vậy ở một `<br>`. Tuy vậy trên thực tế không phải tất cả các trình duyệt đều phân biệt như thế. Lưu ý là nếu cần thay đổi kiểu căn lề hoặc bao phủ của văn bản, phải sử dụng phần tử `<p>`. Sau phần tử `<br/>` không thể chỉ ra những thuộc tính này.

Một quy tắc cần chú ý để sử dụng `<p>` là khi văn bản gốc tách ra thành từng khối, thì các khối này giống như các đoạn văn trong văn bản bình thường. Sử dụng `<br/>` tại nơi muốn ngắt dòng để phục vụ trình bày và sau đó văn bản sẽ tiếp tục với phần còn lại của những khối giống nhau.

Ví dụ 19.7.

Ví dụ khi đặt một văn bản thông thường vào trong một *deck*, sử dụng một phần tử `<p>` cho mỗi đoạn văn:

```
<p>
```

```
văn bản.
```

```
</p>
```

```
<p>
```

```
văn bản
```

```
</p>
```

Khi nhập một nội dung chẳng hạn như địa chỉ bưu điện, nội dung đó bình thường được định dạng theo các dòng nhưng vẫn tạo thành một khối logic, sử dụng `<p>` và một `<br/>` giữa các dòng:

<p>
Arnol.

101 Morris Street

Sebastopol

CA 95472

USA
</p>

19.3.3. Định dạng văn bản

Hỗ trợ định dạng kí tự trong WML khá hạn hẹp so với ngôn ngữ HTML. Không có hỗ trợ các mẫu đặc biệt, kiểu chữ, việc thay đổi kích thước bị giới hạn chỉ có mức “lớn hơn (*bigger*)” hoặc “nhỏ hơn (*smaller*)” và không có sự bảo đảm rằng tất cả các sự lựa chọn đều thực hiện đúng.

Sự hỗ trợ này được cung cấp thông qua bảy phần tử, không có thuộc tính đi kèm, tất cả các hiệu ứng áp dụng cho các văn bản nằm giữa các phần tử này. Trình duyệt hoàn toàn có thể bỏ qua một vài hay tất cả các thuộc tính phụ thuộc vào màn hình hiển thị.

19.3.3.1. Làm nổi bật (Emphasis)

Nếu muốn nội dung văn bản nổi bật, có thể sử dụng các thẻ dưới đây thay cho ****, **<i>**, **<u>** :

- **** Nổi bật.
- **** Rất nổi bật.

Một số trình duyệt có thể sử dụng các thay đổi kiểu như là chữ đậm (*boldface*) hoặc in nghiêng (*italics*) để làm nổi bật văn bản, một số có thể thay đổi màu sắc và một số hoàn toàn bỏ qua các cách làm nổi bật này. Các thẻ hầu hết là các kiểu định dạng kí tự được hỗ trợ và nên sử dụng các thẻ này trước các định dạng khác loại khác.

19.3.3.2. Thay đổi kích thước

Có hai phần tử cung cấp cách thay đổi kích thước văn bản đơn giản :

- **<big>**: Cỡ font lớn.
- **<small>**: Cỡ font nhỏ.

19.3.3.3. Đổi kiểu chữ

Có ba phần tử cung cấp cách thay đổi kiểu chữ đơn giản. Bởi vì nhiều trình duyệt không hỗ trợ các phần tử này cho nên chỉ sử dụng chúng khi thực sự cần.

- `<b>` Đậm (Boldface).
- `<i>` In nghiêng (Italics).
- `<u>` Gạch chân (Underlining).

Ví dụ 19.8.

Các phần tử đã giới thiệu ở trên sẽ được sử dụng đơn giản qua ví dụ sau:

```
<p align="center"><big>
```

Phần văn bản được căn lề giữa

```
</big></p>
```

```
<p align="left">
```

Text `<b>`Chữ đậm - bold`</b>` và `<em>` phần chữ emphasis`</em>`.

```
</p>
```

19.4. Sử dụng Bảng

Bảng là một phần tử được hỗ trợ kém nhất trong WML, rất ít khi được dùng. Nguyên nhân là bảng chứa nhiều thứ và cần nhiều khoảng trống để hiển thị mà đây lại là điểm bất lợi của điện thoại di động.

WML không cho phép giao diện người dùng xuất hiện trong bảng (trừ phần tử `<a>` và `<anchor>`). Bảng trong WML có một số lượng các hàng, mỗi hàng chứa một số ô, các ô có thể chứa nhiều dòng văn bản.

19.4.1. Phần tử `<table>`

Phần tử `<table>` trình bày một bảng trong một thẻ WML. Nó phải xuất hiện trong một đoạn văn (nói cách khác trong một `<p>`). Một bảng có thể không chứa gì nhưng phần tử `<tr>` vẫn tạo cho nó các hàng. Phần tử này có 3 thuộc tính mô tả chi tiết về sự hiển thị của bảng trên màn hình.

Thuộc tính của phần tử `<table>`

- *columns* (một giá trị số bắt buộc): Số lượng các cột trong một bảng. Nếu một hàng (phần tử `<tr>`) có số lượng ô ít hơn số lượng các cột (phần tử `<td>`) trình duyệt chèn thêm các ô trống vào cuối các hàng. Nếu nhiều hơn, các ô ở cuối các hàng được ghép lại với nhau bởi vậy không có một cột thừa nào được tạo ra.

Chương 19

WML cơ bản (tiếp theo)

- *title (xâu tùy chọn)*: Tiêu đề của bảng, trình duyệt có thể sử dụng để hiển thị bảng.
- *align (xâu tùy chọn)*: Tác dụng căn lề trong các cột của bảng, đây là một danh sách các kí tự (L, R, C). L: Căn lề trái trong cột; R: Căn lề phải trong cột; C: Căn giữa.

Nếu các kí tự trong danh sách trên ít hơn các cột thì phần thiếu sẽ được mặc định là "L". Ví dụ danh sách là LLRC cho một bảng có bảy cột thì danh sách được bổ xung là LLRCLLL.

19.4.2. Phần tử <tr>

Bên trong một phần tử <table>, các hàng của bảng được giới hạn bởi phần tử <tr>. Mỗi phần tử <tr> tương ứng với một dòng, các phần tử này không cần thuộc tính, mà chỉ cần một phần tử <td> cho các ô trong hàng. Phần tử <tr> có thể không chứa <td> hoặc rỗng. Điều này cho biết một hàng trống trong một bảng.

19.4.3. Phần tử <td>

Phần tử này bao hàm một ô đơn trong một bảng. Nó có thể chỉ xuất hiện bên trong một phần tử <tr>, không có thuộc tính đi kèm và có thể chứa những đối tượng sau:

- *flow text; * Sử dụng hình ảnh;
- *, , , <i>, <u>, <big>, <small>* Sử dụng để thay đổi kiểu văn bản;
- *<a>, <anchor>* Cố định văn bản.

Một phần tử <td> chỉ chứa các kí tự trắng vẫn được coi là hợp lệ và có nghĩa. Nếu số lượng phần tử <td> trong <tr> lớn hơn số lượng các cột của bảng thì <td> cuối là rỗng(<td/>).

Ví dụ 19.9.

Ví dụ minh họa sử dụng bảng.

- Mã WML

```
<wml>
<card title="Basket">
  <p align="center">
    <table columns="6" align="LRCRCR">
      <tr>
        <td>floppy drive</td>
        <td>|</td>
```

```
<td>@</td>
<td>$$29.95</td>
<td>=</td>
<td>$$29.95</td>
</tr>
<tr>
<td>floppy disk</td>
<td>10</td>
<td>@</td>
<td>$$0.75</td>
<td>=</td>
<td>$$7.50</td>
</tr>
<tr>
<td>TOTAL</td>
<td/><td/><td/>
<td>=</td>
<td><b>$$37.45</b></td>
</tr>
</table>
</p>
</card>
</wml>
```

19.5. Ảnh trong WML

Chủ đề ảnh trong WML là phần cuối cùng của cuốn sách này liên quan tới WML trước khi chúng ta chuyển sang *WMLScript*. Việc trình bày ảnh trong WML không phải là vấn đề quan trọng nếu so sánh với việc trình bày nội dung văn bản, do diện tích màn hình

của điện thoại di động nhỏ, không phù hợp với việc hiển thị ảnh, không những thế hầu hết các trình duyệt WAP hiện nay không hỗ trợ nhiều việc xử lý ảnh.

**19.5.1. Phần tử **

Ảnh được đưa vào trang WML thông qua phần tử . Nếu đã thành thạo với HTML thì cách thức sử dụng ảnh tương tự như trong HTML. Lưu ý rằng một số trình duyệt sắp đặt các ảnh xen lẫn trong dòng với văn bản trong khi một số khác đặt ảnh trên các dòng riêng biệt.

**Thuộc tính của phần tử **

- *src (một địa chỉ url bắt buộc)*: Đưa ra URL sẽ tải ảnh về.
- *alt (một chuỗi ký tự bắt buộc)*: Đưa ra đoạn văn bản được sử dụng để thay thế cho ảnh nếu nó không thể hiển thị do một số nguyên nhân.
- *localsrc (một chuỗi tùy chọn)*: Một số trình duyệt cung cấp ảnh lưu trữ cục bộ thay cho việc tải từ mạng. Thuộc tính này cho phép những ảnh đó được tham chiếu từ phần tử . Ví dụ một hãng cung cấp dịch vụ điện thoại di động có thể đưa ra những chiếc điện thoại có lưu trữ ảnh bên trong và lấy những bức ảnh đó sử dụng cho dịch vụ WAP.
- *align (xâu tùy chọn, mặc định là bottom)*: Chỉ ra cách ảnh được căn lề trong dòng của văn bản. Nếu thiết lập là *bottom*, đáy của ảnh được căn lề với đường chuẩn của dòng hiện thời trong văn bản. Nếu thiết lập bằng *middle*, bức ảnh được căn lề với chính giữa dòng hiện thời của văn bản. Nếu thiết lập bằng *top* đỉnh của bức ảnh được căn lề với đầu dòng của dòng hiện thời.
- *hspace (độ đo tùy chọn)*
- *vspace (độ đo tùy chọn)*: Chỉ ra khoảng diện tích (khoảng màn hình mà ảnh chiếm khi hiển thị) được chèn từ trái qua phải (*hspace*) và từ trên xuống dưới (*vspace*) của ảnh. Thay vì đưa ra 2 thuộc tính trên có thể đưa ra số phần trăm = tỷ lệ màn hình có thể sử dụng để hiển thị ảnh.
- *width (độ đo tùy chọn)*
- *height (độ đo tùy chọn)*: Cho giá trị cỡ của ảnh dưới dạng *pixels* hoặc tỉ lệ phần trăm màn hình. Nó khác so với giá trị thực của ảnh khi download, trình duyệt có thể lựa chọn bỏ qua những giá trị này hoặc xác định theo tỉ lệ cho phù hợp với màn hình hiển thị (hầu hết là kích thước đúng của ảnh). Những giá trị này là một hướng dẫn đơn giản cho phép trình duyệt dự trữ khoảng trống cho ảnh trước khi biết kích thước của chúng.

Chương 19

WML cơ bản (tiếp theo)

Ví dụ 19.11.

Ví dụ đưa ảnh lên một dòng

```
<p></p>
```

Đưa một ảnh theo chiều ngang và dọc vào giữa màn hình.

```
<p></p>
```

19.5.2. Định dạng ảnh WBMP

Tất cả các ví dụ về phân tử `<img>` trong cuốn sách này đều sử dụng file ảnh có đuôi *.wbmp*. Đây là một định dạng ảnh đặc biệt sử dụng trong WAP. Trong thực tế, nó là kiểu định dạng duy nhất mà tất cả các trình duyệt hỗ trợ. Tất cả các trình duyệt WAP hỗ trợ ảnh sẽ đều hỗ trợ *WBMP*, vì vậy để an toàn nhất khi làm việc với các ảnh trong WML chúng ta nên làm việc với định dạng *wbmp*.

WBMP là chữ viết tắt của *Wireless bitmap*, là một định dạng ảnh đơn giản, nó có một số hạn chế là không có các đặc tính về màu sắc, hình ảnh hoạt hình, nén ảnh (Các định dạng ảnh như GIF, JPEG là hai định dạng phổ biến nhất trên các trang web, chúng cho phép nén dữ liệu làm cho kích thước file nhỏ hơn và download nhanh hơn). Các file WBMP chỉ lưu trữ hai thông số cho một ảnh là kích thước và một khối các điểm ảnh, mỗi điểm ảnh là màu trắng hoặc đen. Các phiên bản trong tương lai có thể cung cấp nhiều đặc điểm hơn nhưng hiện tại với hai màu trắng và đen là tất cả những gì định dạng này có.

Để tạo ra các ảnh *.wbmp* cần một chương trình chuyển đổi để tạo ra các file WBMP, bạn có thể download các chương trình đó hoàn toàn miễn phí trên Internet. Chất lượng ảnh sau khi chuyển đổi (đối với ảnh đơn sắc) không đồng bộ, trong một ảnh đơn sắc có nhiều khối với mức độ đậm nhạt khác nhau cho kết quả khác nhau. Do đó giải pháp tốt nhất là sử dụng một phần mềm có bản quyền để chuyển ảnh sang đen trắng với định dạng WBMP.

19.6. Kết chương

Trong chương này các bạn đã được giới thiệu sâu hơn về ngôn ngữ WML. Các bạn đã biết về các phân tử của WML, văn bản trong WML, định dạng văn bản cũng như sử dụng bảng và ảnh trong WML. Trong các chương sau chúng ta sẽ tìm hiểu về WMLScript là một ngôn ngữ kịch bản của kiến trúc WAP và là sự bổ sung cho WML.

Chương 20

Tổng quan về WMLScript

Nội dung của chương:

- Tìm hiểu WMLScript là gì?
- Cài đặt phần mềm giả lập phương tiện không dây OpenwaveSDK 4.1.1.
- Biểu diễn số và xâu kí tự trong WMLScript.
- Các kiểu dữ liệu, từ khóa, toán tử và biểu thức trong WMLScript.

20.1. WMLScript là gì?

WMLScript là một ngôn ngữ kịch bản của kiến trúc WAP. Đặc biệt, WMLScript là phần bổ sung cho WML. WML là ngôn ngữ dựa trên XML xác định nội dung cho các thiết bị bằng thông hợp như điện thoại di động hay máy nhắn tin... Nội dung hiển thị có thể là text, hình ảnh, danh sách... WML cung cấp các định dạng đơn giản có thể tạo ra giao diện đồ họa thân thiện với người sử dụng miễn là các thiết bị sử dụng để hiển thị nội dung có hỗ trợ các định dạng này. Tuy nhiên tất cả các nội dung chuẩn của WML là tĩnh và không có cách nào mở rộng ngôn ngữ mà không phải sửa đổi bản thân WML. Bên cạnh đó, những hạn chế khác của WML là:

- Có sự kiểm tra tính hợp lệ dữ liệu của người sử dụng nhập vào.
- Phải truy cập các tài nguyên của thiết bị: Ví dụ đối với điện thoại di động, cho phép người lập trình có thể tạo cuộc gọi, gửi tin nhắn, truy cập thẻ SIM...
- Sinh ra các thông điệp nội bộ.

WMLScript được thiết kế để khắc phục những hạn chế trên và cung cấp các chức năng có thể lập trình được sử dụng qua đường truyền bằng thông hợp tại phía client với cơ sở hạ tầng mạng hạn chế.

Chương 20

Tổng quan về WMLScript

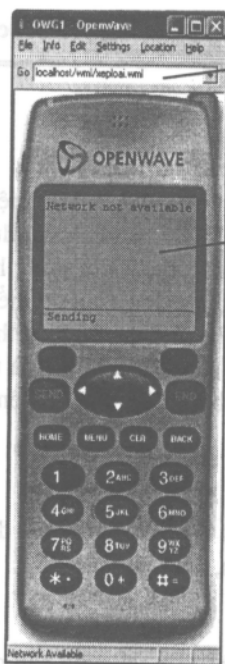
20.2. Cài đặt OpenwaveSDK 4.1.1

Để phục vụ cho việc nghiên cứu về WML và WMLScript, hãng *Openwave* cung cấp cho người sử dụng phần mềm giả lập phương tiện không dây (*Phone Simulator*) cho phép người sử dụng có thể chạy thử các tài liệu WML và các script WMLScript trên *localhost* (máy chủ nội bộ).

Hiện nay hãng Openwave đã cho ra đời nhiều phiên bản *OpenwaveSDK – Phone Simulator*, tất cả các phiên bản đều miễn phí và có sẵn tại website: <http://developer.openwave.com/dvl/>. Lưu ý rằng một số phiên bản không hỗ trợ biên dịch *wmlscript* vì thế cuốn sách này sử dụng phiên bản SDK 4.1.1 tương thích với các hệ trình duyệt Mobile 4.x có hỗ trợ WMLScript.

Một số thao tác cài đặt (trên nền Windows)

- Bạn tải phiên bản SDK 4.1.1 từ địa chỉ website ở trên. Sau đó cài đặt như các phần mềm thông thường. Sau khi cài đặt chương trình bạn có thể kích hoạt để chạy thử.



Cửa sổ địa chỉ

Màn hình hiển thị

Vùng điều khiển

Hình 20.1. Giao diện phần mềm giả lập OpenwaveSDK.

Chương 20

Tổng quan về WMLScript

- Tuy nhiên sau khi bạn cài đặt mới chỉ chạy được các tài liệu WML (.wml) chứ chưa biên dịch được WMLScript. Để chạy thử tài liệu WML, gõ đường dẫn đến tài liệu .wml/ vào cửa sổ địa chỉ (ví dụ: `file://c:\test.wml` - chú ý phần `file://`)

Để có thể biên dịch được script viết bởi WMLScript thì cần một máy chủ (*webserver*), sau đây là các bước tạo *webserver* nội bộ (*localhost*) và thiết lập cấu hình cho nó để có thể biên dịch được script.

- Tạo một thư mục bất kỳ (trong đó chứa các file .wml và .wmls sẽ được thi hành)
- Thiết lập **webserver**:
 - Kích hoạt dịch vụ mà *Microsoft* cung cấp miễn phí **Internet Information Services** (thường ở trong *Control Panel*). Nếu không tìm thấy thì bạn cài đặt lại đơn giản từ đĩa CD cài đặt Windows.
 - Tạo một **Virtual Directory** giả sử tên là **WML** với đường dẫn tới thư mục chứa các file .wml và .wmls đã tạo ra ở trên.
- Thiết lập cấu hình cho **webserver**: Bây giờ phải thiết lập các cấu hình để nhận dạng ra và dịch được WMLScript.
 - Thiết lập MIME: Chọn *Properties* cho *Web Sites* (phía dưới *localcomputer*)
 - Chọn tab *HTTP Headers*
 - Chọn *File Types*
 - Sau đó thiết lập các *MIME type* theo bảng sau

Bảng 20.1. Thiết lập các MIME

Nội dung	MIME Type	Phân mở rộng
WML	text/vnd.wap.wml	.wml
WMLScript	text/vnd.wap.wmlscript	.wmls
BMP Image	image/bmp	.bmp
WBMP Image	image/vnd.wap.wbmp	.wbmp

- Sau khi thiết lập xong bạn khởi động lại máy

Sau khi khởi động máy, bạn đã có thể kiểm tra lại xem đã cài đặt đúng chưa. Giả sử tên máy chủ nội bộ là *localhost*, *virtual directory* tạo ra ở trên là *wml*, trong thư mục chứa file *test.wml*, chạy thử bằng cách gõ vào cửa sổ địa chỉ của trình duyệt như sau: *localhost/wml/test.wml*. Lúc này các script được gọi trong file *test.wml* sẽ được biên dịch.

Chương 20

Tổng quan về WMLScript

20.3. Từ vựng và các quy ước

- Phân biệt chữ hoa, chữ thường: WMLScript là ngôn ngữ có phân biệt chữ hoa, chữ thường (*Case-Sensitive*). Tất cả các từ khoá, các tên biến, tên hàm phải tuân theo quy tắc trên.
- Khoảng trắng và xuống dòng: WMLScript bỏ qua các khoảng trắng (tab, xuống dòng) giữa các thành phần trong chương trình trừ phi các khoảng trắng đó nằm trong một chuỗi.
- Chú thích: WMLScript định nghĩa hai cấu trúc chú thích:
 - Chú thích theo dòng (trên 1 dòng): `//<nội dung chú thích>`
 - Chú thích cả khối (trên nhiều dòng): `/* <nội dung chú thích> */`

20.4. Biểu diễn số và chuỗi ký tự

20.4.1. Biểu diễn dạng số nguyên

Số nguyên được biểu diễn theo một trong những dạng sau:

- Số hệ 10: Được biểu diễn bằng một dãy chỉ gồm các chữ số từ 0 đến 9 không bắt đầu bằng chữ số 0. Ví dụ: -41
- Số Hexa (hệ 16): Được bắt đầu bằng tổ hợp 2 ký tự 0X hoặc 0x theo sau là dãy các chữ số từ 0 đến 9 và các ký tự từ a (hoặc A) đến f (hoặc F). Ví dụ: 0xF3
- Số Octa (hệ 8): Bắt đầu bằng chữ số 0 theo sau là dãy các ký tự chỉ chứa các chữ số từ 0 đến 7. Ví dụ: 032

20.4.2. Biểu diễn dạng số dấu phẩy động

Số dấu phẩy động là số có chứa phần thập phân. Biểu diễn số dấu phẩy động có thể có chứa số thập phân, phần số hay số mũ. Biểu diễn số dấu phẩy động phải có ít nhất một chữ số và dấu, phân cách phần nguyên và phần thập phân hoặc là số mũ.

Số mũ được viết theo cách là e hoặc E theo sau là 1 số nguyên. Phần mũ mô tả số dựa trên phần mũ của 10. Ví dụ: e0 là 100, e-2 là 10-2. Phần mũ có thể có dấu hoặc không dấu (đứng trước là + hoặc - để biểu thị số mũ âm hoặc dương).

Giá trị số dấu phẩy động không nằm trong phạm vi dãy giá trị giới hạn nhất định thì trình dịch sẽ sinh lỗi.

Ví dụ: Trình dịch WMLScript có thể nhận ra được các cách biểu diễn sau đây của giá trị:

3.14

3.14

3.14e0

.314e1

314e-2

20.4.3. Biểu diễn chuỗi ký tự

Chuỗi ký tự là dãy hoặc không có ký tự nào (chuỗi rỗng) hoặc là dãy các ký tự được đặt trong dấu nháy kép ("") hoặc là dãy các ký tự được đặt trong dấu nháy đơn (' ')

Chuỗi ký tự là một con trỏ trỏ tới vùng nhớ chứa dãy ký tự. Ứng dụng có thể thay đổi giá trị của dãy ký tự. WMLScript cho phép biểu diễn chuỗi ký tự bằng cả dấu nháy đơn và dấu nháy kép miễn là phải mở đầu và kết thúc chuỗi bằng cùng một loại có nghĩa là nếu bạn mở đầu chuỗi bằng nháy đơn thì bạn cũng phải kết thúc chuỗi bằng nháy đơn. Dấu nháy kép cũng tương tự như thế nếu không trình dịch sẽ sinh lỗi.

Ví dụ về chuỗi ký tự:

'Ha Noi, Viet Nam'

""Vu Kiem Van""

""Ngày 15 tháng 9 năm 2004 ""

20.4.4. Biểu diễn dạng Boolean

Trong WML Script dạng boolean cũng chứa 2 giá trị là *false* và *true*.

20.5. Biến

WMLScript chỉ hỗ trợ biến được khai báo bên trong hàm hoặc biến được truyền như là tham số của hàm.

Khai báo

WMLScript đòi hỏi người lập trình phải khai báo tên biến trước khi sử dụng. Không cần thiết phải khởi tạo giá trị cho biến trong lúc khai báo, vì các biến sẽ tự động được khởi trị bằng cách gán với chuỗi rỗng ("").

Tất cả các khai báo biến phải đứng trước bằng từ khoá *var*. Nếu khai báo nhiều hơn một biến trong cùng một câu lệnh phải phân cách các biến bằng dấu (,) và kết thúc mỗi lệnh khai báo phải có dấu (;). Không nhất thiết phải khai báo kiểu dữ liệu của biến.

Ví dụ

var x;

var x, y;

```
var size = 3;  
var name = "My Name";
```

Phạm vi và vòng đời của biến

Phạm vi của biến là đoạn chương trình mà biến có thể được tham chiếu đến. Vòng đời của biến là khoảng thời gian từ lúc khai báo biến cho đến khi kết thúc hàm (còn gọi là sự tồn tại của biến).

Phạm vi của hầu hết các biến là trong hàm nơi biến được khai báo. Tên biến trong phạm vi hàm phải là duy nhất.

Truy cập

Một biến chỉ được truy cập trong phạm vi hàm nơi biến được khai báo. Truy cập một biến bằng cách sử dụng tên của nó.

Ví dụ

```
var myAge = 20;  
var yourAge = 21;  
var ourAge = myAge + yourAge;
```

20.6. Kiểu dữ liệu

Kiểu dữ liệu là các kiểu đã được định nghĩa mà biến có thể lưu giữ. WMLScript chỉ hỗ trợ các kiểu dữ liệu nội tại. Như đã nói ở trên, khi khai báo biến, không phải xác định kiểu dữ liệu cho biến đó và bất kỳ biến nào cũng có thể có bất kỳ kiểu dữ liệu nào tùy vào từng thời điểm. WMLScript sẽ tự động chuyển đổi giữa các kiểu dữ liệu khi cần thiết.

Ví dụ

```
var flag = true;// Kiểu boolean  
var number = 12;// Kiểu số nguyên  
var point = 8.92;// Kiểu dấu phẩy động  
var century = "XXI";// Kiểu chuỗi ký tự  
var except = invalid;// invalid
```

Ngoài bốn kiểu dữ liệu chuẩn, WMLScript sử dụng thêm 1 kiểu dữ liệu thứ năm đó là *invalid*, khi các kiểu dữ liệu cần phân biệt sự khác nhau giữa các kiểu dữ liệu nội tại khác.

- Sau đây chúng ta sẽ xem xét giới hạn và cách sử dụng từng kiểu dữ liệu:
 - Kiểu số nguyên "Integer"
 - Kiểu số dấu phẩy động "Floating-Point"

Chương 20

Tổng quan về WMLScript

- Kiểu xâu ký tự “String”
- Kiểu “Boolean”

Kiểu số nguyên Integer

WMLScript hỗ trợ kiểu nguyên nằm trong dãy giá trị -2147483648 đến +2147483647. Có thể sử dụng hàm thư viện *Lang* để nhận được 2 giá trị trên

- *Lang.maxInt()*; Giá trị nguyên lớn nhất
- *Lang.minInt()*; Giá trị nguyên nhỏ nhất

Kiểu số dấu phẩy động Floating-Point

Trong WMLScript, giá trị số dấu phẩy động lớn nhất là 3.40282347E+38. Giá trị nhỏ nhất là giá trị lớn hơn 0 vào khoảng 1.17549435E-38 hoặc nhỏ hơn.

WMLScript xử lý các giá trị kiểu dấu phẩy động theo các quy tắc sau:

- Nếu kết quả của một thao tác là số dấu phẩy động nhưng không nằm trong dãy giá trị thực (không phải là số, giá trị lớn vô cực...) thì kết quả sẽ là kiểu *invalid*.
- Nếu kết quả của một thao tác là số dấu phẩy động mà giá trị nhỏ hơn giới hạn dưới (*underflow*) thì kết quả là *zero (0.0)*.
- -0 và +0 là bằng nhau và không có sự phân biệt.

Có thể sử dụng hàm thư viện *Float* để nhận được giá trị dấu phẩy động lớn nhất và nhỏ nhất: *Float.maxFloat()* và *Float.minFloat()*;

Kiểu xâu ký tự

Biến kiểu xâu ký tự chứa tạm thời các ký tự, chữ số hoặc các ký tự đặc biệt khác. Ta có thể khởi tạo cho biến kiểu xâu ký tự theo cách biểu diễn xâu ký tự đã nói ở phần trên. Ta cũng có thể thao tác với các giá trị kiểu xâu bằng các toán tử của WMLScript và các hàm nằm trong thư viện chuẩn *String*.

Ví dụ về biến kiểu xâu ký tự:

```
var msg = "Hello";  
var name = "Vu Kiem Van";  
msg = msg + name;  
var len = String.length(msg);
```

Kiểu boolean

Sử dụng giá trị kiểu *boolean* để khởi tạo hoặc gán cho biến trong câu lệnh cần thiết có tham số là kiểu *boolean*. Giá trị kiểu *boolean* có thể là *false* hoặc *true* cũng có thể là kết quả của một tính toán logic.

Chương 20

Tổng quan về WMLScript

Ví dụ về biến kiểu *boolean*:

```
var truth = true;
```

```
var lie = !true;
```

20.7. Từ khoá

Từ khoá là các từ dành riêng có ý nghĩa đặc biệt trong chương trình và do đó chúng ta không được sử dụng từ khoá để định dạng. WMLScript định nghĩa các từ khoá sau:

Bảng 20.1. Các từ khoá trong WML Script

Access	Extern	path
Agent	For	return
Break	function	typeof
Continue	Header	url
div	http	use
div=	If	user
domain	isvalid	var
else	Meta	while
equiv	Name	

Các từ khoá dưới đây đã được xếp vào danh mục để tích hợp với các phiên bản tương lai của WMLScript, tránh sử dụng các từ khoá này trong chương trình

Bảng 20.2. Các từ khoá với các phiên bản tương lai trong WML Script

case	Enum	sizeof
catch	Export	struct
class	extends	super
const	finally	switch
debugger	Import	throw
default	private	try
do	Public	

Chương 20

Tổng quan về WMLScript

Một số từ khoá thông dụng trong các ngôn ngữ khác nhưng WMLScript không định nghĩa (do đó người lập trình không được sử dụng) trong bảng 20.3.

Bảng 20.3. Các từ khóa WMLScript không định nghĩa

delete	new	void
in	null	with
lib	this	

20.8. Toán tử và biểu thức

20.8.1. Toán tử

Toán tử được xem là chức năng đã được dựng sẵn và là một phần của WMLScript. Ví dụ biểu thức $(x+y)$ được tạo bởi hai biến x và y liên kết với nhau là toán tử $+$. Ta sẽ lần lượt tìm hiểu các loại toán tử sau đây:

- Toán tử gán
- Toán tử số học
- Toán tử logic
- Toán tử chuỗi ký tự
- Toán tử so sánh
- Toán tử thao tác với mảng
- Toán tử
- Toán tử điều kiện
- Toán tử typeof
- Toán tử isvalid

Toán tử gán

Toán tử gán thực hiện việc gán một giá trị nào đó cho toán hạng. WMLScript cung cấp những loại toán tử gán sau:

Bảng 20.4. Toán tử gán

Toán tử	Thực hiện
=	Gán
+=	Thực hiện phép cộng (đối với số) hoặc ghép chuỗi sau đó gán
-=	Thực hiện phép trừ và sau đó gán
*=	Thực hiện phép nhân sau đó gán

Chương 20

Tổng quan về WMLScript

/=	Thực hiện phép chia sau đó gán
div=	Thực hiện phép chia số nguyên (lấy phần nguyên) sau đó gán
%=	Lấy số dư sau đó gán
<<=	Dịch trái bit sau đó gán
>>=	Dịch phải bit sau đó gán
&=	AND bit sau đó gán
^=	XOR bit sau đó gán
=	OR bit sau đó gán

Một số ví dụ sử dụng toán tử gán:

```
var a = "abc";// Gán giá trị "abc" cho biến a
var b = a;// Gán giá trị biến a cho biến b
var c = "def";// Gán giá trị "def" cho biến c
a += c;// Ghép giá trị của c vào giá
// trị của a, gán cho a
// sau phép toán này, a = "abcdef"
```

Toán tử số học

Toán tử số học là ký hiệu thực hiện các phép toán số học. WMLScript cung cấp 3 dạng toán tử số học sau:

Bảng 20.5. Các dạng toán tử số học cơ bản

Toán tử	Thực hiện
+	Cộng (dạng số) hoặc ghép chuỗi
-	Trừ
*	Nhân
/	Chia
Div	Chia số nguyên lấy phần nguyên

Các toán tử số học thao tác với bit

WMLScript cung cấp các toán tử số học thao tác với bit sau:

Chương 20
Tổng quan về WMLScript

Bảng 20.6. Các toán tử số học thao tác với bit

Toán tử	Thực hiện
%	Lấy số dư, dấu của số dư trùng với dấu của số bị chia
<<	Dịch trái
>>	Dịch phải
>>>	Dịch phải và chèn 0 vào phần dịch
&	AND
	OR
^	XOR

Các toán tử thông dụng khác

Bảng 20.7. Các toán tử thông dụng khác

Toán tử	Thực hiện
+	Cộng
-	Trừ
--	Giảm 1 đơn vị
++	Tăng 1 đơn vị
~	NOT (phủ định)

Các toán tử logic

Các toán tử logic sử dụng để xét quan hệ giữa các điều kiện. Các toán tử logic cơ bản gồm có:

Bảng 20.8. Các toán tử logic

Toán tử	Thực hiện
&&	Phép AND logic
	Phép OR logic
!	Phép NOT (phủ định) logic

Chương 20

Tổng quan về WMLScript

- Cách thực hiện phép AND logic là trước hết toán tử kiểm tra giá trị của toán hạng đầu tiên, nếu giá trị là *false* hoặc *invalid*, kết quả phép toán sẽ là *false* hoặc *invalid* và không cần phải kiểm tra giá trị của toán hạng thứ hai. Trường hợp toán hạng thứ nhất có giá trị *true* thì giá trị kết quả sẽ là giá trị của toán hạng thứ hai.
- Tương tự đối với phép OR logic, trước hết toán tử kiểm tra giá trị của toán hạng đầu tiên, nếu giá trị là *true* hoặc *invalid*, kết quả phép toán sẽ là *true* hoặc *invalid* và không phải kiểm tra giá trị của toán hạng thứ hai. Trường hợp toán hạng thứ nhất có giá trị *false* thì giá trị kết quả sẽ là giá trị của toán hạng thứ hai.
- Phép phủ định: Là phép toán một ngôi, phủ định giá trị của toán hạng: *!false = true*, *!true = false*.

Ví dụ:

```
weAgree = (iAmRight && youAreRight) || (!iAmRight && !youAreRight);
```

WMLScript đòi hỏi giá trị *boolean* trong các phép toán logic và hỗ trợ việc tự động chuyển đổi giá trị kiểu dữ liệu này sang giá trị kiểu dữ liệu khác.

Chú ý: Nếu giá trị của toán hạng đầu tiên của phép *&&* hoặc *||* là *invalid* thì kết quả sẽ là *invalid* và không cần phải kiểm tra giá trị toán hạng thứ 2.

Ví dụ:

```
var a = (1/0) || foo();// Kết quả: invalid
//không phải gọi hàm foo()
var b = true || (1/0); // Kết quả: true
var c = false || (1/0);// Kết quả: invalid
```

Toán tử thao tác với chuỗi ký tự

Có hai toán tử phục vụ việc ghép các chuỗi đó là *+* và *+=*. Ngoài hai toán tử trên thì trong thư viện chuẩn String của WMLScript còn hỗ trợ các thao tác làm việc với chuỗi.

Ví dụ:

```
var str = "Beginning" + "End";
var chr = String.charAt(str,10); // Lấy ký tự thứ
//10 của chuỗi str, chr = "E"
```

Toán tử so sánh

Toán tử so sánh sử dụng để so sánh giá trị của biến. Toán tử so sánh còn được gọi là toán tử quan hệ. WMLScript cung cấp các toán tử so sánh sau:

Chương 20

Tổng quan về WMLScript

Bảng 20.9. Toán tử so sánh

Toán tử	Thực hiện
<	Nhỏ hơn
<=	Nhỏ hơn hoặc bằng
==	Bằng (khác với phép gán =)
>=	Lớn hơn hoặc bằng
>	Lớn hơn
!=	Khác

Các toán tử và phép so sánh tuân theo các quy định sau:

- Kiểu *boolean*: true lớn hơn false
- Kiểu *Integer*, *Floating-Point*: Theo quy tắc tự nhiên vốn có
- Kiểu *string*: So sánh dựa trên thứ tự mã của các ký tự trong chuỗi cần so sánh. Mã của các ký tự được định nghĩa bởi tập các ký tự hỗ trợ trong trình thông dịch WMLScript.
- *Invalid*: Nếu ít nhất một toán hạng là *invalid* thì kết quả sẽ là *invalid*.

Ví dụ:

```
var res = (mypoint > yourpoint);  
var val = ((1/0) == invalid); // Kết quả: invalid
```

Bảng sau cung cấp các quy tắc chuyển đổi kiểu dữ liệu khi so sánh:

Bảng 20.10. Quy tắc chuyển đổi các kiểu dữ liệu khi so sánh

Kiểu toán hạng	Quy tắc chuyển đổi	Ví dụ
Boolean	- Nếu toán hạng là kiểu <i>boolean</i> hoặc có thể chuyển được sang kiểu <i>boolean</i> , thì hành thao tác trên kiểu <i>boolean</i> và trả về giá trị của cho thao tác	true && 3.4 => boolean 1 && 0 => boolean "A" "" =>boolean !42 =>boolean
	- Trả về kiểu <i>invalid</i>	!invalid =>invalid 3 && invalid =>invalid

Chương 20
Tổng quan về WMLScript

Integer	- Nếu toán hạng là kiểu <i>integer</i> hoặc có thể chuyển đổi được sang kiểu <i>integer</i> , thì hành thao tác trên kiểu <i>integer</i> và trả về giá trị của cho thao tác	"7" << 2 =>integer , true << 2 =>integer
	- Trả về kiểu <i>invalid</i>	2.1 div 4 =>invalid
Floating-point	Nếu toán hạng là kiểu <i>Floating-point</i> hoặc có thể chuyển đổi được sang kiểu <i>floating-point</i> , thì hành thao tác trên kiểu <i>floating-point</i> và trả về giá trị của cho thao tác	
	- Trả về kiểu <i>invalid</i>	
String	Nếu toán hạng là kiểu <i>String</i> hoặc có thể chuyển đổi được sang kiểu <i>string</i> , thì hành thao tác trên kiểu <i>string</i> và trả về giá trị của cho thao tác	
	- Trả về kiểu <i>invalid</i>	
Integer, floating-point hoặc string	Nếu có ít nhất 1 toán hạng là kiểu <i>floating-point</i> hoặc có thể chuyển đổi sang kiểu <i>floating-point</i> , chuyển đổi toán hạng còn lại sang kiểu <i>floating-point</i> , thực hiện thao tác trên kiểu <i>floating-point</i>	100/10.3 =>float 3.4*"4.3" =>float "2.3"*"3" =>float
	- Nếu toán hạng là kiểu <i>integer</i> hoặc có thể chuyển đổi được sang kiểu <i>integer</i> , thì hành thao tác trên kiểu <i>integer</i> và trả về giá trị của cho thao tác	33*44 =>integer "10"*3 =>integer "10"-"2" =>integer
	- Nếu có ít nhất 1 toán hạng là kiểu <i>string</i> hoặc có thể chuyển đổi sang kiểu <i>string</i> , chuyển đổi toán hạng còn lại sang kiểu <i>string</i> , thực hiện thao tác trên kiểu <i>string</i>	"12"+5.4 =>string "Hey"<56 =>string 2.7+"4.2" =>string
	- Trả về kiểu <i>invalid</i>	3.2*"A" =>invalid .9*"9e999" =>invalid invalid*1 =>invalid

Chương 20

Tổng quan về WMLScript

Toán tử xử lý mảng

WWMLScript không cung cấp kiểu mảng truyền thống (mà ta thường thấy ở các ngôn ngữ lập trình khác) tuy nhiên thư viện chuẩn String cung cấp những hàm giúp có thể mô phỏng các thao tác như đối với kiểu mảng truyền thống

Ví dụ về hàm trong thư viện String:

```
function dummy()  
{  
    var str = "Mary had a little lamp";  
    var word = String.elementAt(str,4,"");  
}
```

Toán tử “,”

Toán tử “,” sử dụng để kết hợp nhiều thao tác tính toán vào trong cùng một biểu thức. Kết quả của toán tử “,” là giá trị của toán hạng thứ hai.

Ví dụ:

```
for (a=1, b=10; a<10; a++, b++)  
{  
    //Mã thực hiện  
}
```

Toán tử “,” còn được sử dụng trong việc gọi hàm để ngăn cách giữa các đối số hoặc dùng để ngăn cách giữa các biến với nhau trong câu lệnh khai báo biến.

Ví dụ:

```
var a = 2;  
var b = 3, c = 3;  
myFunction("Name", 3*(b*a-c));  
//2 đối số là "Name", 9
```

Toán tử điều kiện

Toán tử điều kiện có nhiệm vụ gán giá trị cho một biểu thức dựa trên kết quả *boolean* của một lệnh trước đó. Toán tử điều kiện (?:) là một kiểu lệnh *if-then-else* cơ bản gồm có ba toán hạng được sắp xếp như sau:

$$d = \text{opr1} ? \text{opr2} : \text{opr3};$$

Chương 20

Tổng quan về WMLScript

opr1 là điều kiện sẽ được kiểm tra trước tiên, nếu điều kiện đúng, kết quả biểu thức sẽ bằng *opr2*. Nếu điều kiện là *false* hay *invalid* thì kết quả biểu thức sẽ là *opr3*

Toán tử *typeof*

Toán tử *typeof* trả về một giá trị nguyên mô tả kiểu giá trị của biểu thức. Toán tử *typeof* không chuyển đổi từ kiểu dữ liệu này sang kiểu dữ liệu khác.

Toán tử *typeof* trả về giá trị dựa trên kiểu dữ liệu theo bảng sau:

Bảng 20.11. Giá trị trả về của toán tử *typeof*

Kiểu	Mã trả về của <i>typeof</i>
Integer	0
Floating-point	1
String	2
Boolean	3
Invalid	4

Việc sử dụng toán tử *typeof* rất cần thiết vì như đã biết, WMLScript không đòi hỏi phải khai báo kiểu cho biến khi khai báo biến và tự động chuyển đổi kiểu giá trị khi cần thiết nên rất nhiều trường hợp cần toán tử này để xác định kiểu của biến đang nắm giữ là gì.

Ví dụ về toán tử *typeof*:

```
var str = "Kiểm tra toán tử typeof";  
var strType = typeof str; //strType = 2
```

Toán tử *isvalid*

Toán tử *isvalid* kiểm tra sự hợp lệ của biến hoặc biểu thức. Nó trả về giá trị kiểu *boolean*. Giá trị *false* nếu kiểu kết quả của biểu thức là *invalid* hoặc nếu biến không có kiểu hợp lệ. Nếu kiểu kết quả của biểu thức không phải là *invalid* hoặc kiểu của biến là hợp lệ thì toán tử *isvalid* sẽ trả về kết quả *true*. Toán tử *isvalid* cũng không chuyển đổi kết quả từ kiểu này sang kiểu khác

Ví dụ:

```
var str = "123";  
var ok = isvalid str; // true  
var test = isvalid(1/0); // false
```

20.8.2. Biểu thức

Biểu thức là sự kết hợp của các toán tử, các hằng, các giá trị, các hàm và các biến để thực thi một thao tác nhất định trong chương trình. WMLScript cung cấp hầu hết các loại biểu thức thông dụng thường thấy trong các ngôn ngữ lập trình khác.

Biểu thức có thể chỉ đơn giản là một giá trị của hằng hay biến như sau:

567

66.77

"This is an expression"

'Test expression'

true

Biểu thức cũng có thể phức tạp hơn với sự kết hợp của các toán tử, hằng, biến và lời gọi hàm...

MyAccount + 3;

(a + b)/3;

Value + Next(value);

Pragmas

Pragma là một chỉ dẫn sinh ra các hoạt động đặc biệt của trình dịch. Các *Pragma* xác định được đặt tại đầu mỗi *WMLScript file*, trước tất cả các khai báo hàm. Tất cả các *pragma* được bắt đầu bằng từ khoá *use* và theo sau là các thuộc tính riêng biệt của *pragma*.

File ngoài

Có thể sử dụng URL (*Uniform Resource Locator*) để truy cập file WMLScript. Sử dụng *pragma use url* để gọi hàm trong file ngoài.

Ví dụ: OtherScript "http://www.host.com/app/script";

Để gọi 1 file ngoài phải xác định:

- URL của nguồn WMLScript ("http://www.host.com/app/script")
- Tên resource (OtherScript)

Sau đó bên trong khai báo hàm, ta sử dụng resource như sau:

```
function test (par1, par2) {  
    return OtherScript#check(par1-par2);  
}
```

Chương 20

Tổng quan về WMLScript

- *Pragma* xác định URL đến file WMLScript
- Hàm được gọi nạp file sử dụng URL đã xác định ở trên
- Hàm kiểm tra lại nội dung file và thi hành 1 hàm nhất định (*check*)

Pragma use url có *namespace* riêng của nó dành cho việc tổ chức các tên của *pragma* (các tên cục bộ).

Điều khiển truy nhập

Sử dụng *pragma* điều khiển truy nhập để bảo vệ nội dung file. Bạn phải gọi *pragma* điều khiển truy nhập trước lời gọi hàm ngoài. Sử dụng nhiều hơn một *pragma* điều khiển truy nhập trong file thì trình dịch sẽ sinh lỗi.

Mỗi khi gọi một hàm ngoài, trình dịch sẽ tiến hành kiểm tra điều khiển truy nhập để quyết định file đích có cho phép truy cập từ lời gọi hay không. *Pragma* điều khiển truy nhập xác định thuộc tính *domain* (tên miền) và *path* (đường dẫn). Nếu file có thuộc tính *domain* hoặc *path* thì URL của file cần phải phù hợp với các thuộc tính trên.

Trình dịch kiểm tra sự thích hợp như sau:

- Tên miền (*domain*) truy cập phù hợp phần hậu tố (*suffix*) dựa vào đoạn tên miền của URL, tham chiếu tới.
- Đường dẫn (*path*) truy cập phù hợp với phần tiền tố (*prefix*) dựa vào đoạn đường dẫn của URL tham chiếu tới.
- Thuộc tính *domain* và *path* tuân theo quy tắc chữ hoa, chữ thường của URL
- Trình dịch thực thi việc khớp phần hậu tố tên miền bằng cách sử dụng toàn bộ các thành phần của tên miền phụ và các thành phần này phải khớp nhau. Ví dụ: *www.wapforum.org* thì khớp với *wapforum.org* nhưng không khớp với *forum.org*
- Trình dịch thực thi việc khớp phần tiền tố đường dẫn bằng cách xem xét toàn bộ các thành phần đường dẫn và mỗi thành phần phải khớp với nhau. Ví dụ: */X/Y* thì khớp với */X* không khớp với */XZ*

Thuộc tính tên miền mặc định là tên miền của file hiện thời, đường dẫn mặc định là */*. Để đơn giản hoá việc phát triển ứng dụng, có thể không cần biết đường dẫn tuyệt đối đến file hiện thời, thuộc tính *path* cũng chấp nhận URLs tương đối. *User agent* sẽ tiến hành chuyển đổi từ đường dẫn tương đối sang đường dẫn tuyệt đối và tiến hành khớp thuộc tính như trên.

Ví dụ:

Nếu thuộc tính điều khiển truy nhập cho file là:

use access domain "wapforum.org" path "/finance";

Chương 20

Tổng quan về WMLScript

thì những URLs tham chiếu sau có thể cho phép gọi hàm ngoài xác định trong các file dưới đây

<http://wapforum.org/finance/money.cgi>

<http://wapforum.org/finance/markets.cgi>

<http://wapforum.org/finance/demo/packages.cgi?x+123&y+4>

Tuy nhiên các URL dưới đây thì không cho phép gọi hàm ngoài

<http://test.net/finance>

<http://wapforum.org/internal/foo.wml>

Meta-Information

Pragma Meta-Information xác định thuộc tính tên và nội dung của file. *Pragma Information* còn xác định lược đồ file (*scheme*), dạng cấu trúc có thể được sử dụng để thông dịch các giá trị thuộc tính – các giá trị bị biến đổi phụ thuộc vào kiểu của meta-data. Các giá trị thuộc tính là các xâu ký tự.

Meta-pragma có các thuộc tính sau đây:

- “Name”
- “HTTP Equiv”
- “User Agent”

20.9. Kết chương

Trong chương này chúng ta đã tìm hiểu về ngôn ngữ kịch bản WMLScript của kiến trúc WAP. Để biên dịch các chương trình viết bằng WMLScript các bạn phải cài đặt phần mềm giả lập phương tiện không dây OpenwaveSDK. Ngoài ra các bạn cũng đã tìm hiểu cách biểu diễn số và xâu ký tự trong WMLScript, các kiểu dữ liệu, từ khóa, toán tử và biểu thức trong WMLScript. Trong chương sau các bạn sẽ tìm hiểu về hàm trong WMLScript.

Chương 21

Hàm và Sử dụng hàm

Nội dung của chương:

- Tìm hiểu về hàm: khai báo, gọi hàm, cách sử dụng hàm.
- Giới thiệu 6 thư viện chuẩn của WMLScript.

21.1. Giới thiệu

Trong chương này chúng ta sẽ tìm hiểu về hàm trong WMLScript. Các bạn sẽ biết cách khai báo, gọi và sử dụng hàm để xây dựng các ứng dụng của mình. WMLScript cũng hỗ trợ một thư viện các hàm đã được xây dựng sẵn cung cấp nhiều tính năng mà đôi khi không thể viết được bằng một chương trình WMLScript đơn thuần. Chúng ta sẽ được giới thiệu các hàm quan trọng trong thư viện hàm WMLScript này.

21.2. Khai báo

Hàm được khai báo sau các *pragmas*. Dạng thức khai báo hàm như sau:

```
function function-name(parameter-list)
```

```
{
```

```
body
```

```
}
```

hoặc

```
extern function function-name(parameter-list)
```

```
{
```

```
body
```

```
}
```


Chương 21

Hàm và Sử dụng hàm

Trong đó:

- *function*: từ khoá để khai báo hàm. Nếu hàm được khai báo bởi từ khoá *extern*, thì hàm có thể được gọi từ WML hoặc từ các hàm WMLScript khác không được khai báo trong cùng một file. Nếu hàm không được khai báo bởi từ khoá *extern* thì hàm chỉ được gọi bởi các đoạn mã WMLScript khác chứa trong cùng một file nguồn.
- *function-name*: tên hàm – do người lập trình quyết định và tuân theo quy tắc định danh trong WMLScript. Mọi tên hợp lệ đối với việc định danh biến cũng có thể sử dụng để định danh hàm. Thêm vào đó, có thể sử dụng cùng một tên cho biến và hàm, khi đó WMLScript sẽ phân biệt chúng bằng các *namespace* cho mỗi tên.
- *parameter-list*: Danh sách các tên của mỗi đối số truyền vào hàm. Các tên này cũng tuân theo nguyên tắc định danh biến. Trong thân của hàm, các đối số được xem như là các biến. Thậm chí có thể thay đổi giá trị của chúng.

Thân hàm (*body*): Là dãy các lệnh WMLScript

Ví dụ 21.1.

Hàm sau đây mang ba đối số có tên là *a*, *b* và *c*. Hàm thực hiện nhân ba đối số lại với nhau và trả về kết quả phép nhân.

```
function multiply(a, b, c)
{
    return a*b*c;
}
```

Ví dụ 21.2.

Một ví dụ phức tạp hơn, ta sẽ sử dụng hàm để tính *N* giai thừa

```
function factorial(n)
{
    if (n < 0)
        return invalid;
    var total = 1;
    for (; n>1; --n)
    {
        total *= n;
    }
    return total;
}
```

Chương 21

Hàm và Sử dụng hàm

Chú ý: cấu trúc điều khiển *for* sẽ được tìm hiểu ở phần sau.

Ví dụ 21.3.

Hàm tính giai thừa ở trên còn có thể được thực hiện bằng cấu trúc *while*

```
function factorial (n)
{
  if (n < 0)
    return invalid;
  var total = 1;
  while (n > 1)
  {
    total *= n;
    --n;
  }
  return total;
}
```

Như đã lưu ý, chúng ta không cần phải xác định kiểu dữ liệu trả về cho hàm. Trên thực tế, hàm có thể trả về bất kỳ kiểu giá trị nào (như ở các ví dụ trên, hàm có thể trả về giá trị *integer* hoặc là *invalid*).

Trong WMLScript, không có cách nào khai báo một hàm mà không trả về giá trị, có nghĩa là *tất cả các hàm đều trả về một giá trị nào đó*, tuy nhiên, trong thân hàm cũng không bắt buộc phải có câu lệnh *return* và nếu có câu lệnh *return* thì câu lệnh *return* cũng không nhất thiết phải có giá trị. Trong cả hai trường hợp này, hàm sẽ trả về một xấu rỗng.

21.3. Gọi hàm

Gọi hàm trong WMLScript giống như trong C hay Java:

function-name(argument-list)

Số lượng tham số cung cấp cho lời gọi hàm phải tương thích với số lượng đối số trong khai báo hàm. Tham số có thể là một biểu thức, tuy nhiên nếu các tham số là các biểu thức có chứa toán tử “,” thì toàn bộ biểu thức phải được đóng trong cặp ngoặc “()”, nếu không chúng sẽ bị nhầm lẫn toán tử “,” bởi dấu hiệu phân tách giữa 2 tham số.

Ví dụ 21.4.

```
factorial(5)
foo()
function_with_lots_of_arguments(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

Chú ý: Các dạng thức gọi hàm trên chỉ được áp dụng cho việc các gọi hàm nằm trong cùng một file nguồn. Để gọi hàm trong file khác phải sử dụng các cú pháp mở rộng.

21.4. Gọi hàm từ các đơn vị Script khác

Để gọi một hàm từ một file khác phải xác định file ngoài trong lời gọi

file-identifier#function-name(argument-list)

Tên hàm (*function-name*) và danh sách đối số (*argument-list*) cũng giống như gọi một hàm bình thường. Định danh file tham chiếu đến một URL qua tên được thiết lập với *url pragma*. Định danh file cũng tuân theo quy tắc định danh chung cho tên biến, tên hàm trong WMLScript.

Ví dụ 21.5.

Ví dụ về việc gọi hàm từ file ngoài:

```
mathlibrary#factorial (6)
arraylibrary#sort (array)
netlib#check_email_address (addr)
```

21.5. Url pragma

url pragma định danh tên file ngoài theo dạng thức:

use url file-identifier url-string;

file-identifier là tên được sử dụng trong lời gọi hàm ngoài, nó tuân theo quy tắc định danh chung đối với biến, hàm. Có thể định danh *file-identifier* trùng tên của biến hoặc hàm bởi vì chúng sử dụng các *namespace* khác nhau nên có thể phân biệt được.

url-string là một giá trị kiểu chuỗi ký tự bình thường trong WMLScript. Nội dung của nó là một URL có thể là tương đối hoặc tuyệt đối.

Ví dụ 21.6.

```
use url netlib "http://wap.net/lib/netlib.wmlsc";
use url test "test.wmlsc";
use url example "/script/example.wmlsc";
```

21.6. Gọi WMLScript từ WML

Một hàm có thể được gọi từ WML phải được khai báo là *extern*. Cũng như thông thường, hàm có thể nhận số lượng đối số bất kỳ.

Nhúng lời gọi hàm bằng tác vụ <go>

Gọi WMLScript từ WML bằng cách gọi tác vụ <go> với định dạng Url đặc biệt. Hàm được gọi và các đối số truyền được mã hoá sang *URL fragment* (phần nằm sau ký tự # đầu tiên). *Fragment* này giống như lời gọi hàm WMLScript thông thường.

Ví dụ 21.6.

Lời gọi hàm *test('foo')* trong file tham chiếu bởi URL tương đối *test.wmlsc* có thể được viết như sau:

```
<go href="test.wmlsc#test('foo')"/>
```

Chú ý: sử dụng dấu nháy đơn ở chuỗi 'foo'. Nó có thể viết tương đương như sau:

```
<go href="test.wmlsc#test('foo')"/>
```

Sử dụng dấu nháy đơn hoặc nháy kép trong tất cả các giá trị thuộc tính và sau đó sử dụng dạng còn lại trong lời gọi nhúng tới WMLScript. Chẳng hạn đặt dấu nháy kép (") tại các giá trị thuộc tính và sau đó sử dụng dấu nháy đơn trong chuỗi nhúng giống như ví dụ ở trên.

Tiếp sau đây, chúng ta sẽ tìm hiểu sự khác nhau giữa cách nhúng lời gọi hàm trong <go> và cách viết lời gọi trong bản thân WMLScript.

Thay thế biến

Lời gọi nhúng trong tác vụ <go> thông thường bao gồm việc thay thế biến của trình duyệt trực tiếp vào lời gọi. Ví dụ lời gọi hàm để truyền giá trị của biến *var* thay thế bởi chuỗi 'foo', viết:

```
<go href="test.wmlsc#test('$ (var:n)')"/>
```

(:n làm biến được thay thế mà không cần URL-escaping)

Chú ý: Cần đặt dấu nháy cho các chuỗi tham số bởi trình duyệt luôn luôn thực hiện việc thay thế biến trước khi gọi tới WMLScript. Nếu không đặt dấu nháy, trình duyệt sẽ cố gắng dịch giá trị theo nhiều kiểu dữ liệu khác nhau (*integer*, *boolean*, hoặc *floating point*).

Vì việc thay thế biến hoàn thành trước khi lời gọi được phân tích, cho nên thậm chí có thể thay thế toàn bộ phần lời gọi hàm của URL. Ví dụ, nếu biến *call* chứa giá trị 'foo', có thể viết:

```
<go href="test.wmlsc#$(call:n)"/>
```

Lời gọi này cũng có tác dụng giống như ví dụ đầu tiên

Chương 21

Hàm và Sử dụng hàm

Không có biểu thức nào là các đối số trong dạng viết nhanh của tác vụ <go>

Một đặc điểm khác của tác vụ <go> là bạn có thể sử dụng dạng viết nhanh được cung cấp bởi một vài cấu trúc WML như là *event bindings*. Ví dụ: bạn có một hàm được gọi khi một *WML timer* hết hiệu lực, có thể sử dụng *event binding* sau:

```
<onevent type="ontimer">
  <go href="process.wmlsc#timeout( )"/>
</onevent>
```

Có thể rút ngắn về một thuộc tính đơn bằng phần tử <card>:

```
ontimer="process.wmlsc#timeout( )"
```

Lời gọi WMLScript nhúng trong <go> không thể chứa biểu thức, tất cả các đối số phải là giá trị sau khi trình duyệt tiến hành việc thay thế biến. Có một ngoại lệ là có thể sử dụng dấu “-” để xác định số âm.

Ví dụ 21.7.

Nếu chúng ta muốn gọi hàm *foo()* trong *Ví dụ.wmlsc* tính tổng của 2 biến *var_a* và *var_b*, không được viết như sau:

```
<go href="Ví dụ.wmlsc#foo($(var_a:n)+$(var_b:n))"/>
```

bởi vì các đối số không phải là giá trị. Thay vào đó, viết lại hàm để chấp nhận hai biến trình duyệt và có một hàm WMLScript *foo()* thì hành phép cộng:

```
<go href="Ví dụ.wmlsc#foo($(var_a:n),$(var_b:n))"/>
```

Có thể thi hành phép cộng trong một hàm “gói” và sau đó truyền kết quả về nguyên bản.

```
extern function foo_wrapper(a,b)
{
  foo(a + b);
}
```

Trong trường hợp này lời gọi hàm sẽ là:

```
<go ref="Ví dụ.wmlsc#foo_wrapper($(var_a:n),$(var_b:n))"/>
```

Một số lưu ý

- Một lời gọi hàm WMLScript nhúng trong URL không được có ký tự trắng giữa tên hàm và dấu “(“ mở đầu danh sách đối số.
- Hằng xâu ký tự không được chứa các *escape sequences* như là *\n*, *\0243* hoặc *\0xAD*.

Chương 21

Hàm và Sử dụng hàm

- Lưu ý không được đặt cặp nhảy đơn vào một xâu đã được đặt trong cặp nhảy đơn khác, tương tự cũng không được đặt cặp nhảy kép vào một xâu đã được đặt trong cặp nhảy kép khác.

Ví dụ 21.8.

Một vài ví dụ gọi WMLScript từ WML

```
<go href="netlib.wmlsc#check_email('$ (email)')"/>
<go href="http://wap.net/foo.wmlsc#foo( )"/>
<go href="vidu.wmlsc#use_string('foo')"/>
<go href="vidu.wmlsc#use_integer(17)"/>
<go href="vidu.wmlsc#use_integer(0105)"/>
<go href="vidu.wmlsc#use_integer(0x3e8)"/>
<go href="vidu.wmlsc#use_float(.01)"/>
<go href="vidu.wmlsc#use_float(1e-2)"/>
<go href="vidu.wmlsc#use_boolean(true)"/>
<go href="vidu.wmlsc#use_boolean(false)"/>
<go href="vidu.wmlsc#use_invalid(invalid)"/>
```

21.7. Các thư viện chuẩn

Nếu thường xuyên sử dụng các hàm giống nhau trong các đoạn mã WMLScript khác nhau, các bạn nên đặt chúng vào các thư viện. Một cách đơn giản là đặt chúng trong cùng một file và khai báo chúng là *extern*. Sau đó, các hàm đó sẽ được truy cập từ bất cứ chỗ nào ở bên ngoài bằng cách sử dụng *URL pragma* và một lời gọi hàm ngoài (như đã giới thiệu ở phần trên).

Ngoài các thư viện do người sử dụng định nghĩa *user-defined libraries*, WMLScript cung cấp một số thư viện đã được dựng sẵn trong bản thân trình duyệt, không viết trong WMLScript. Chúng có thể thi hành những tác vụ mà không một chương trình thuần túy nào viết trong WMLScript có thể thực hiện được. Các thư viện này được gọi là các thư viện chuẩn *standard libraries*, để phân biệt chúng với thư viện do người sử dụng định nghĩa.

Do WMLScript được thiết kế với mục đích đơn giản, dễ hiểu nên có một số tính năng không trực tiếp cung cấp. Ví dụ: mặc dù có thể ghép xâu bằng việc sử dụng toán tử "+" nhưng không có cách nào để lấy xâu con, chia nhỏ thành các ký tự đơn từ một xâu hay làm một số thao tác xử lý xâu khác,... trong khi hầu hết các ngôn ngữ lập trình khác có thể thực hiện được. Tương tự, không có một cách nào để chuyển đổi giá trị giữa hai kiểu dữ liệu khác nhau mà giá trị sẽ được chuyển đổi một cách tự động nếu cần thiết.

Chương 21

Hàm và Sử dụng hàm

Tất cả những hàm như trên được cung cấp trong thư viện chuẩn của WMLScript. Do được dựng sẵn trong trình duyệt nên chúng không phải *download* thường xuyên và việc sử dụng chúng làm cho kích thước chương trình giảm đáng kể.

Gọi hàm thư viện chuẩn

Tất cả các hàm thư viện chuẩn đều được gọi theo cách giống nhau, với phần mở rộng so với cú pháp thông thường của lời gọi hàm WMLScript. Tên của thư viện và hàm được gọi đã có trước.

library-name.function-name(argument-list)

Giới thiệu chi tiết về các thư viện và các hàm liên quan.

Phần dưới đây sẽ tóm tắt 6 thư viện chuẩn và các hàm trong thư viện tương ứng.

Bảng 21.1. 6 thư viện chuẩn và các hàm

Tên thư viện	Các hàm trong thư viện
Lang Library	abort
	abs
	characterSet
	exit
	float
	isFloat
	isInt
	max
	maxInt
	min
	minInt
	parseFloat
	parseInt
	random
	seed

Chương 21
Hàm và Sử dụng hàm

Float Library	ceil
	floor
	Int
	maxFloat
	minFloat
	pow
	round
	sqrt
String Library	charAt
	compare
	elementAt
	elements
	find
	format
	insertAt
	isEmpty
	length
	removeAt
	replace
	replaceAt
	squeeze
	subString
	toString
	trim
URL Library	escapeString
	getBase
	getFragment
	getHost

Chương 21

Hàm và Sử dụng hàm

	getParameters
	getPath
	getPort
	getQuery
	getReferer
	getScheme
	isValid
	loadString
	resolve
	unescapeString
WMLBrowser Library	getCurrentCard
	getVar
	Go
	newContext
	prev
	refresh
	setVar
Dialogs Library	Alert
	Confirm
	Prompt

21.7.1. Thư viện *Lang Library*

Thực hiện việc chuyển đổi từ chuỗi ký tự sang số nguyên, số dấu phẩy động, sinh ngẫu nhiên và các thao tác liên quan tới trình thông dịch. Thư viện bao gồm một số hàm sau:

Hàm Abort

- Chức năng: Hoãn việc thi hành trình thông dịch (do có lỗi).
- Cách sử dụng: *Lang.abort (msg)* với *msg* là chuỗi ký tự.

Chương 21

Hàm và Sử dụng hàm

- Mô tả: hoàn trình thông dịch WMLScript, hiển thị thông báo *msg* và chuyển điều khiển tới nơi làm cho trình thông dịch chạy (như là trình duyệt WML). Nếu *msg* là *invalid*, nó sẽ được chuyển đổi thành xâu " *invalid*".
- Ví dụ: *Lang.abort("fatal error: bad arguments")*

Hàm abs

- Chức năng: trả về trị tuyệt đối của một số.
- Cách sử dụng: *Lang.abs (num)* với *num* : Số.
- Mô tả: Nếu *num* là một số nguyên, số dấu phẩy động (hoặc có thể chuyển đổi về dạng nguyên, dạng phẩy động) hàm sẽ trả về số nguyên dương hoặc số phẩy động dương tương ứng.
- Ví dụ:

*Lang.abs(17)*Trả về số nguyên 17

*Lang.abs("-17")*Trả về số nguyên 17

*Lang.abs(2.3)*Trả về số dấu phẩy động 2.3

*lang.abs(-2.3)*Trả về số dấu phẩy động 2.3

*Lang.abs("foo")*Trả về *invalid*

Hàm characterSet

- Chức năng: Trả về định danh cho tập các ký tự hỗ trợ bởi trình thông dịch WMLScript.
- Cách sử dụng: *Lang.characterSet()*.
- Ví dụ: *Lang.characterSet()*.

Hàm exit

- Chức năng: Dừng thi hành trình thông dịch WMLScript interpreter.
- Cách sử dụng: *Lang.exit (value)*
- Ví dụ:

Lang.exit("foo")

Lang.exit(invalid)

Lang.exit(6.7)

Chương 21

Hàm và Sử dụng hàm

Hàm float

- Chức năng: báo trình thông dịch có hỗ trợ thao tác với số dấu phẩy động hay không.
- Cách sử dụng: *Lang.float()*.
- Ví dụ: *Lang.float()*, trả về giá trị true hoặc false.

Hàm isFloat

- Chức năng: Kiểm tra xem 1 xâu có thể phân tích trở thành một số dấu phẩy động hay không
- Cách sử dụng: *Lang.isFloat (str)* với *str* là xâu ký tự
- Ví dụ:

Lang.isFloat("123.456") Trả về true

Lang.isFloat("17") Trả về true

Lang.isFloat(" -7.2 C") Trả về true

Lang.isFloat(" 55.3 mph") Trả về true

Lang.isFloat("6e+3") Trả về true

Lang.isFloat(" .3") Trả về true

Lang.isFloat("foo") Trả về false

Lang.isFloat(false) Trả về false

Lang.isFloat(invalid) Trả về invalid

Hàm isInt

- Chức năng: Kiểm tra xem 1 xâu có thể phân tích thành một số nguyên hay không.
- Cách sử dụng: *Lang.isInt (str)* với *str* là xâu ký tự.
- Ví dụ:

Lang.isInt("17") Trả về true

Lang.isInt("23.67") Trả về true

Lang.isInt(" -63 percent") Trả về true

Lang.isInt("+4inches") Trả về true

Lang.isInt(true) Trả về false

Lang.isInt(" .3") Trả về false

Chương 21

Hàm và Sử dụng hàm

`Lang.isInt("+")` Trả về false

`Lang.isInt(invalid)` Trả về invalid

Hàm max

- Chức năng: Trả giá trị lớn nhất trong 2 số.
- Cách sử dụng: `Lang.max (num1, num2)` với `num1` là số thứ nhất và `num2`: là số thứ hai.
- Ví dụ:

`Lang.max(-2, 3.4)` Trả về 3.4

`Lang.max(17, 23)` Trả về 23

`Lang.max(-3, -3.0)` Trả về -3

`Lang.max(-5, false)` Trả về false

`Lang.max("6", 6)` Trả về "6"

`Lang.max("six", 6)` Trả về invalid

Hàm maxInt

- Chức năng: Trả về giá trị nguyên cực đại.
- Cách sử dụng: `Lang.maxInt()`
- Ví dụ: `Lang.maxInt()` sẽ trả về 2147483647

Hàm min

- Chức năng: Trả về giá trị nhỏ nhất trong 2 số.
- Cách sử dụng: `Lang.min (num1, num2)` với `num1` là số thứ nhất và `num2` là số thứ hai.
- Ví dụ:

`Lang.min(1, -2.0)` Trả về -2.0

`Lang.min(17, 23)` Trả về 17

`Lang.min(4.0, 4)` Trả về 4.0

`Lang.min(true, 2)` Trả về true

`Lang.min("-1", -1)` Trả về "-1"

`Lang.min(42, "foo")` Trả về invalid

Chương 21

Hàm và Sử dụng hàm

Hàm `minInt`

- Chức năng: Trả về giá trị nguyên cực tiểu.
- Cách sử dụng: `Lang.minInt()`
- Ví dụ: `Lang.minInt()` trả về `-2147483648`

Hàm `parseFloat`

- Chức năng: Chuyển đổi 1 chuỗi ký tự sang số dấu phẩy động.
- Cách sử dụng: `Lang.parseFloat(str)` với `str` : Chuỗi ký tự.
- Ví dụ:

`Lang.parseFloat("123.456")` Trả về `123.456`

`Lang.parseFloat("17")` Trả về `17.0`

`Lang.parseFloat(" -7.2 C")` Trả về `7.2`

`Lang.parseFloat(" 55.3 mph")` Trả về `55.3`

`Lang.parseFloat("6e+3")` Trả về `6000.0`

`Lang.parseFloat(" .3")` Trả về `0.3`

`Lang.parseFloat("foo")` Trả về `invalid`

`Lang.parseFloat(false)` Trả về `invalid`

Hàm `parseInt`

- Chức năng: Chuyển đổi 1 chuỗi ký tự sang giá trị số nguyên.
- Cách sử dụng: `Lang.parseInt(str)` với `str` là chuỗi ký tự.
- Ví dụ :

`Lang.parseInt("17")` Trả về `17`

`Lang.parseInt("23.67")` Trả về `23`

`Lang.parseInt(" -63 percent")` Trả về `-63`

`Lang.parseInt("+4inches")` Trả về `4`

`Lang.parseInt(true)` Trả về `invalid`

`Lang.parseInt(".3")` Trả về `invalid`

`Lang.parseInt("+")` Trả về `invalid`

Chương 21

Hàm và Sử dụng hàm

Hàm random

- Chức năng: Sinh ngẫu nhiên một số nguyên.
- Cách sử dụng: *Lang.random (range)* với *range* là giới hạn trên.
- Ví dụ:

Lang.random(23) Trả về số nguyên nằm giữa 0 và 23

Lang.random(17.2) Trả về số nguyên nằm giữa 0 và 17

Lang.random("foo") Trả về *invalid*

Hàm seed

- Chức năng: Thiết lập số "hạt giống" cho bộ sinh ngẫu nhiên.
- Cách sử dụng: *Lang.seed (seed)* với *seed* là một số.
- Ví dụ:

Lang.seed(23) Gieo số nguyên 23 và Trả về ""

Lang.seed(1.2) Gieo số nguyên 1 và Trả về ""

Lang.seed("foo") Trả về *invalid* và không thay đổi gì cả

21.7.2. Thư viện Float Library

Thư viện chuẩn *float* chứa các hàm phục vụ các thao tác với số dấu phẩy động như hàm chuyển đổi sang số nguyên hoặc là các hàm tính bình phương... Một vài trình thông dịch WMLScript không hỗ trợ các thao tác với số dấu phẩy động, trong trường hợp này tất cả các hàm trong thư viện đều trả về giá trị *invalid*.

Hàm ceil

- Chức năng: Trả về số nguyên nhỏ nhất lớn hơn hoặc bằng đối số.
- Cách sử dụng: *Float.ceil (num)* với *num* là số dấu phẩy động.
- Ví dụ:

Float.ceil(17.0) Trả về 17

Float.ceil(17.4) Trả về 18

Float.ceil(17.5) Trả về 18

Float.ceil(-17.0) Trả về -17

Float.ceil(-17.4) Trả về -17

Chương 21

Hàm và Sử dụng hàm

`Float.ceil(-17.5)` Trả về -17

`Float.ceil(5e10)` Trả về invalid

`Float.ceil(-5e10)` Trả về invalid

`Float.ceil("foo")` Trả về invalid

Hàm floor

- Chức năng: Trả về số nguyên lớn nhất nhỏ hơn hoặc bằng đối số.
- Cách sử dụng: *Float.floor (num)* với *num* là số dấu phẩy động .
- Ví dụ

`Float.floor(17.0)` Trả về 17

`Float.floor(17.4)` Trả về 17

`Float.floor(17.5)` Trả về 17

`Float.floor(-17.0)` Trả về -17

`Float.floor(-17.4)` Trả về -18

`Float.floor(-17.5)` Trả về -18

`Float.floor(5e10)` Trả về invalid

`Float.floor(-5e10)` Trả về invalid

`Float.floor("foo")` Trả về invalid

Hàm int

- Chức năng: Cắt 1 số dấu phẩy động để lấy giá trị nguyên.
- Cách sử dụng: *Float.int (num)* với *num* là số dấu phẩy động.
- Ví dụ

`Float.int(17.0)` Trả về 17

`Float.int(17.4)` Trả về 17

`Float.int(17.5)` Trả về 17

`Float.int(-17.0)` Trả về -17

`Float.int(-17.4)` Trả về -17

`Float.int(-17.5)` Trả về -17

Chương 21

Hàm và Sử dụng hàm

`Float.int(5e10)` Trả về invalid

`Float.int(-5e10)` Trả về invalid

`Float.int("foo")` Trả về invalid

Hàm `maxFloat`

- Chức năng: Trả về số dấu phẩy động cực đại.
- Cách sử dụng: `Float.maxFloat()`
- Ví dụ: `Float.maxFloat()` có thể trả về 3.40282347e38

Hàm `minFloat`

- Chức năng: Trả về số dấu phẩy động cực tiểu.
- Cách sử dụng: `Float.minFloat()`
- Ví dụ: `Float.minFloat()` trả về số dấu phẩy động nhỏ hơn hoặc bằng 1.17549435e-38

Hàm `pow`

- Chức năng: Lấy lũy thừa của một số.
- Cách sử dụng: `Float.pow(base, power)` với *base* là số dấu phẩy động, *power* là số dấu phẩy động (số mũ).
- Ví dụ:

`Float.pow(3.0, 2.0)` Trả về 9.0

`Float.pow(9.0, 0.5)` Trả về 3.0

`Float.pow(8.0, 1/3)` Trả về 2.0

`Float.pow(-2.0, 0.0)` Trả về 1.0

`Float.pow(-2.0, 4.0)` Trả về 16.0

`Float.pow(-2.0, 5.0)` Trả về -32.0

`Float.pow(-2.0, -1.0)` Trả về -0.5

`Float.pow(-2.0, 0.5)` Trả về invalid

`Float.pow(0.0, 1.7)` Trả về 0.0

`Float.pow(0.0, 0.0)` Trả về 1.0

`Float.pow(0.0, -1.0)` Trả về invalid

Chương 21

Hàm và Sử dụng hàm

Hàm round

- Chức năng: Làm tròn số dấu phẩy động về số nguyên gần nhất.
- Cách sử dụng: *Float.round (num)* với *num* là số dấu phẩy động.
- Ví dụ

Float.round(17.0) Trả về 17

Float.round(17.4) Trả về 17

Float.round(17.5) Trả về 18

Float.round(-17.0) Trả về -17

Float.round(-17.4) Trả về -17

Float.round(-17.5) Trả về -17 (không phải -18)

Float.round(5e10) Trả về invalid

Float.round(-5e10) Trả về invalid

Float.round("foo") Trả về invalid

Hàm sqrt

- Chức năng: Trả về căn bậc 2 của số dấu phẩy động.
- Cách sử dụng: *Float.sqrt (num)* với *num* là số dấu phẩy động.
- Ví dụ

Float.sqrt(4.0) Trả về 2.0

Float.sqrt(1.0) Trả về 1.0

Float.sqrt(-1.0) Trả về invalid

21.7.3. String Library

Chứa các hàm cho phép thao tác với chuỗi ký tự, ví dụ tách chuỗi thành các ký tự đơn, tính độ dài chuỗi... Cần chú ý là tất cả các hàm không làm thay đổi giá trị của chuỗi ký tự đối số mà sẽ trả về một chuỗi khác, có nghĩa là chuỗi nguyên gốc không bị thay đổi.

Hàm charAt

- Chức năng: Trả về ký tự tại một vị trí.
- Cách sử dụng: *String.charAt (str, index)* với *str* là chuỗi ký tự và *index* là vị trí.
- Ví dụ:

Chương 21

Hàm và Sử dụng hàm

`String.charAt("Example", 0)` Trả về xâu "E"
`String.charAt("Example", 3)` Trả về xâu "m"
`String.charAt("Example", 7)` Trả về xâu ""
`String.charAt("Example", -1)` Trả về xâu ""
`String.charAt(42, 1)` Trả về xâu "2"
`String.charAt(false, 4)` Trả về xâu "e"
`String.charAt(invalid, 0)` Trả về invalid
`String.charAt("foo", "bar")` Trả về invalid

Hàm `compare`

- Chức năng: So sánh 2 xâu ký tự.
- Cách sử dụng: `String.compare(str1, str2)` với `str1` là xâu ký tự 1, `str2` là xâu ký tự 2.
- Ví dụ:

`String.compare("one", "two")` Trả về -1
`String.compare("two", "three")` Trả về 1
`String.compare("four", "four")` Trả về 0
`String.compare("six", "sixteen")` Trả về -1
`String.compare(6, 16)` Trả về 1
`String.compare(invalid, "foo")` Trả về invalid
`String.compare("foo", invalid)` Trả về invalid

Hàm `elementAt`

- Chức năng: Trả về phần tử từ 1 dãy được lưu trữ như là 1 xâu.
- Cách sử dụng: `String.elementAt(str, index, sep)` trong đó `str` là xâu ký tự, `index` là vị trí, `sep` là xâu ký tự.
- Ví dụ:

`String.elementAt("a;b;c;d", 2, ";")` Trả về xâu "c"
`String.elementAt("a;b;c;d", 23, ";")` Trả về xâu "d"
`String.elementAt("a;b;c;d", -1, ";")` Trả về xâu "a"
`String.elementAt("a;b;c;d", 1.3, ";")` Trả về xâu "b"

Chương 21

Hàm và Sử dụng hàm

`String.elementAt("", 1, "x")` Trả về `xâu ""`

`String.elementAt(Invalid, 1, "x")` Trả về `invalid`

`String.elementAt("foo", "one", "x")` Trả về `invalid`

`String.elementAt("foo", 1, Invalid)` Trả về `invalid`

`String.elementAt("foo", 1, "")` Trả về `invalid`

Hàm `elements`

- Chức năng: Trả về số phần tử trong 1 dãy được lưu trữ như 1 xâu.
- Cách sử dụng: `String.elements(str, sep)` với `str`: xâu ký tự, `sep`: xâu ký tự.
- Ví dụ:

`String.elements("1;2;3;4", ";")` Trả về `integer 4`

`String.elements("1;2;3;4", ";foo")` Trả về `integer 4`

`String.elements("aa bbb c dd e", " ")` Trả về `integer 5`

`String.elements("x,y,,,z", ",")` Trả về `integer 7`

`String.elements("", "x")` Trả về `integer 1`

`String.elements("foo", "")` Trả về `invalid`

`String.elements(Invalid, "x")` Trả về `invalid`

`String.elements("foo", Invalid)` Trả về `invalid`

Hàm `find`

- Chức năng: Tìm xâu con trong 1 xâu.
- Cách sử dụng: `String.find(str, substr)` với `str`: xâu ký tự, `substr`: xâu ký tự.
- Ví dụ

`String.find("Example", "x")` Trả về `integer 1`

`String.find("Example", "mp")` Trả về `integer 3`

`String.find("Example", "plf")` Trả về `integer -1`

`String.find("ababababab", "baba")` Trả về `integer 1`

`String.find(123456, 45)` Trả về `integer 3`

`String.find(Invalid, "foo")` Trả về `invalid`

Chương 21

Hàm và Sử dụng hàm

`String.find("foo", invalid)` Trả về `invalid`

Hàm `format`

- Chức năng: Chuyển đổi 1 giá trị về dạng chuỗi ký tự với định dạng nào đó.
- Cách sử dụng: *String.format (format, value)*, trong đó *format*: chuỗi ký tự, *value*: giá trị bất kỳ.
- Ví dụ:

`String.format("Name: %s", "Fred")` Trả về chuỗi "Name: Fred"

`String.format("Number: %d", 6)` Trả về chuỗi "Number: 6"

`String.format("Speed: %f", 23.1)` Trả về chuỗi "Speed: 23.100000"

`String.format("%7.3s", "wibble")` Trả về chuỗi " wib"

`String.format("%7.3d", 17)` Trả về chuỗi " 017"

`String.format("%7.3f", 5.4)` Trả về chuỗi " 5.400"

`String.format("%s", true)` Trả về chuỗi "true"

`String.format("%d", true)` Trả về chuỗi "1"

`String.format("%f", true)` Trả về chuỗi "1.000000"

`String.format("%s", "0042")` Trả về chuỗi "0042"

`String.format("%d", "0042")` Trả về chuỗi "42"

`String.format("%f", "0042")` Trả về chuỗi "42.000000"

`String.format("%.f", "1.2345")` Trả về chuỗi "1"

`String.format("%.2f%%", 1.3)` Trả về chuỗi "1.30%"

`String.format("%d%f%%d", 6)` Trả về chuỗi "%d%f%6"

`String.format(invalid, 0)` Trả về `invalid`

`String.format("%", 0)` Trả về `invalid`

`String.format("%x", 0)` Trả về `invalid`

`String.format("%s", invalid)` Trả về `invalid`

`String.format("%d", "two")` Trả về `invalid`

`String.format("%f", "six")` Trả về `invalid`

Chương 21

Hàm và Sử dụng hàm

Hàm `insertAt`

- Chức năng: Chèn thêm 1 phần tử vào dãy.
- Cách sử dụng: *String.insertAt (str, new, index, sep)* trong đó *str*: chuỗi ký tự, *new*: chuỗi ký tự, *index*: vị trí, *sep*: chuỗi ký tự.
- Ví dụ:

```
insertAt("a;b;c;d", "x", 2, ";") Trả về chuỗi "a;b;x;c;d"
insertAt("a;b;c;d", "x", -1.6, ";") Trả về chuỗi "x;a;b;c;d"
insertAt("a;b;c;d", "x", 17, ";") Trả về chuỗi "a;b;c;d;x"
insertAt(invalid, "x", 1, "x") Trả về invalid
insertAt("foo", invalid, 2, "x") Trả về invalid
insertAt("foo", "x", "three", "x") Trả về invalid
insertAt("foo", "x", 4, invalid) Trả về invalid
insertAt("foo", "x", 5, "") Trả về invalid
```

Hàm `isEmpty`

- Chức năng: trả lại giá trị *true* nếu là chuỗi rỗng.
- Cách sử dụng: *String.isEmpty (str)* trong đó *str*: chuỗi ký tự.
- Ví dụ

```
String.isEmpty("") Trả về boolean true
String.isEmpty("foo") Trả về boolean false
String.isEmpty(123) Trả về boolean false
String.isEmpty(true) Trả về boolean false
String.isEmpty(2.3) Trả về boolean false
String.isEmpty(invalid) Trả về invalid
```

Hàm `length`

- Chức năng: Trả về độ dài của chuỗi.
- Cách sử dụng: *String.length (str)* trong đó *str*: chuỗi ký tự.
- Ví dụ:

Chương 21

Hàm và Sử dụng hàm

`String.length("foo")` Trả về integer 3

`String.length("")` Trả về integer 0

`String.length(42)` Trả về integer 2

`String.length(true)` Trả về integer 4

`String.length(invalid)` Trả về invalid

Hàm `removeAt`

- Chức năng: Xoá phần tử của dãy tại 1 vị trí.
- Cách sử dụng: `String.removeAt (str, index, sep)` trong đó *str*: xâu ký tự, *index*: vị trí, *sep*: xâu ký tự.
- Ví dụ:

`String.removeAt("a;b;c;d", 2, ";")` Trả về xâu "a;b;d"

`String.removeAt("a;b;c;d", -1, ";")` Trả về xâu "b;c;d"

`String.removeAt("a;b;c;d", 6, ";")` Trả về xâu "a;b;c"

`String.removeAt("a;b;c;d", 1.2, ";")` Trả về xâu "a;c;d"

`String.removeAt("", 0, ";")` Trả về xâu ""

`String.removeAt("x", 0, ";")` Trả về xâu ""

`String.removeAt(invalid, 1, "x")` Trả về invalid

`String.removeAt("foo", "two", "x")` Trả về invalid

`String.removeAt("foo", 1, invalid)` Trả về invalid

`String.removeAt("foo", 1, "")` Trả về invalid

Hàm `replace`

- Chức năng: Thay thế 1 xâu con trong 1 xâu bằng xâu con khác.
- Cách sử dụng: `String.replace (str, old, new)` trong đó *str*: xâu ký tự ban đầu, *old*: xâu ký tự bị thay, *new*: xâu ký tự mới sẽ thay vào.
- Ví dụ:

`String.replace("Hello, world!", "l", "p")` Trả về xâu "Heppo, worpd!"

`String.replace("Hello, world!", "ll", "y")` Trả về xâu "Heyo, world!"

`String.replace("aaaaa", "aa", "b")` Trả về xâu "bba"

`String.replace(123456, 34, 43)` Trả về xâu "124356"

Chương 21

Hàm và Sử dụng hàm

Hàm `replaceAt`

- Chức năng: thay thế 1 phần tử tại 1 vị trí.
- Cách sử dụng: `String.replaceAt(str, new, index, sep)` trong đó *str*: chuỗi ký tự ban đầu, *new*: chuỗi ký tự, *index*: vị trí, *sep*: chuỗi ký tự.
- Ví dụ:

```
String.replaceAt("a;b;c;d", "xxx", 1, ";") Trả về chuỗi "a;xxx;c;d"
String.replaceAt("a;b;c;d", "xxx", -1, ";") Trả về chuỗi "xxx;b;c;d"
String.replaceAt("a;b;c;d", "xxx", 9.8, ";") Trả về chuỗi "a;b;c;xxx"
String.replaceAt("", "xxx", 0, ";") Trả về chuỗi "xxx"
String.replaceAt(Invalid, "x", 1, "x") Trả về Invalid
String.replaceAt("foo", Invalid, 1, "x") Trả về Invalid
String.replaceAt("foo", "x", "one", "x") Trả về Invalid
String.replaceAt("foo", "x", 1, Invalid) Trả về Invalid
String.replaceAt("foo", "x", 1, "") Trả về Invalid
```

Hàm `squeeze`

- Chức năng: Chuyển tất cả các chuỗi trắng trong 1 chuỗi thành 1 ký tự trắng.
- Cách sử dụng: `String.squeeze(str)` trong đó *str*: chuỗi ký tự.
- Ví dụ:

```
String.squeeze("foo") Trả về chuỗi "foo"
String.squeeze(" Once upon a time ") Trả về chuỗi " Once upon a time"
String.squeeze(Invalid) Trả về Invalid
```

Hàm `substring`

- Chức năng: Cắt 1 chuỗi con ra từ 1 chuỗi.
- Cách sử dụng: `String.substring(str, start, len)` trong đó *str*: chuỗi ký tự, *start*: vị trí bắt đầu cắt, *len*: độ dài chuỗi kí tự cần cắt.
- Ví dụ

```
String.substring("Example", 1, 2) Trả về chuỗi "xa"
String.substring("Example", 4, 6) Trả về chuỗi "ple"
```

Chương 21

Hàm và Sử dụng hàm

String.subString("Example", -3, 1) Trả về xâu "E"
String.subString("Example", 7, 2) Trả về xâu ""
String.subString("Example", 3, -2) Trả về xâu ""
String.subString("Example", 2.9, 3.8) Trả về xâu "amp"
String.subString(12345, 2, 2) Trả về xâu "34"
String.subString(false, 1, 3) Trả về xâu "als"
String.subString(invalid, 4, 9) Trả về invalid
String.subString("foo", "bar", 1) Trả về invalid
String.subString("foo", 1, "bar") Trả về invalid

Hàm toString

- Chức năng: Chuyển đổi bất kỳ giá trị WMLScript sang xâu.
- Cách sử dụng: *String.toString (value)* trong đó *value*: giá trị WMLScript bất kỳ.
- Ví dụ

String.toString("foo") Trả về xâu "foo"
String.toString(17) Trả về xâu "17"
String.toString(1.5) Trả về xâu "1.5"
String.toString(false) Trả về xâu "false"
String.toString(invalid) Trả về xâu "invalid"

Hàm trim

- Chức năng: Xoá bỏ ký tự trắng ở đầu và cuối xâu.
- Cách sử dụng: *String.trim (str)* trong đó *str*: xâu ký tự.
- Ví dụ:

String.trim("foo") Trả về "foo"
String.trim(" Once upon a time ") Trả về xâu "Once upon a time"
String.trim(invalid) Trả về invalid

21.7.4. URL Library

Thư viện chứa các hàm để phân tích các URLs cả URL tương đối lẫn URL tuyệt đối.

Chương 21

Hàm và Sử dụng hàm

Hàm `escapeString`

- Chức năng: Mã hóa chuỗi URL.
- Cách sử dụng: `URL.escapeString (str)`.
- Ví dụ: `URL.escapeString("/foo.cgi?foo=1&bar=2")`
Trả về xâu `"%2Ffoo.cgi%3Ffoo%3D1%26bar%3D2"`

Hàm `getBase`

- Chức năng: Trả về URL của đơn vị WMLScript hiện thời.
- Cách sử dụng: `URL.getBase()`.
- Ví dụ: `URL.getBase()` có thể trả về xâu `http://host/script.wmls`

Hàm `getFragment`

- Chức năng: Trả về *fragment* cắt từ URL.
- Cách sử dụng : `URL.getFragment (url)`.
- Ví dụ:
`URL.getFragment("#frag")` Trả về xâu "frag"
`URL.getFragment("foo.wml#card2")` Trả về xâu "card2"
`URL.getFragment("../logo.wbmp")` Trả về xâu ""
`URL.getFragment("http://www.inhand.net:xxx/")` Trả về invalid
`URL.getFragment(invalid)` Trả về invalid

Hàm `getHost`

- Chức năng: Trả về *host* cắt từ URL.
- Cách sử dụng `URL.getHost (url)`
- Ví dụ:
`URL.getHost("http://www.inhand.net/")` Trả về xâu "www.inhand.net"
`URL.getHost("../logo.wbmp")` Trả về xâu ""
`URL.getHost("http://www.inhand.net:xxx/")` Trả về invalid
`URL.getHost(invalid)` Trả về invalid

Hàm `getParameters`

- Chức năng: Trả về danh sách tham số từ URL.
- Cách sử dụng: `URL.getParameters (url)`

Chương 21

Hàm và Sử dụng hàm

- Ví dụ

URL.getParameters("http://www.inhand.net;x;y;z") Trả về xâu "x;y;z"
URL.getParameters("http://www.inhand.net/") Trả về xâu ""
URL.getParameters("../logo.wbmp") Trả về xâu ""
URL.getParameters("http://www.inhand.net:xxx/") Trả về invalid
URL.getParameters(invalid) Trả về invalid

Hàm getPath

- Chức năng: Trả về đường dẫn lấy từ URL.
- Cách sử dụng: *URL.getPath (url)*
- Ví dụ

URL.getPath("http://www.inhand.net/") Trả về xâu "/"
URL.getPath("../logo.wbmp") Trả về xâu "../logo.wbmp"
URL.getPath("test:") Trả về xâu ""
URL.getPath("http://www.inhand.net:xxx/") Trả về invalid
URL.getPath(invalid) Trả về invalid

Hàm getPort

- Chức năng: Trả về port cắt từ URL.
- Cách sử dụng: *URL.getPort (url)*
- Ví dụ

URL.getPort("http://www.inhand.net:80/") Trả về xâu "80"
URL.getPort("http://www.inhand.net/") Trả về xâu ""
URL.getPort("http://www.inhand.net:xxx/") Trả về invalid
URL.getPort(invalid) Trả về invalid

Hàm getQuery

- Chức năng: Trả về query lấy từ URL.
- Cách sử dụng: *URL.getQuery (url)*
- Ví dụ:

Chương 21

Hàm và Sử dụng hàm

URL.getQuery("http://www.inhand.net/") Trả về chuỗi ""

URL.getQuery("/cgi-bin/Ví dụ.cgi?foo=42&bar=6") Trả về "foo=42&bar=6"

URL.getQuery("http://www.inhand.net:xxx/") Trả về invalid

URL.getQuery(invalid) Trả về invalid

Hàm getReferer

- Chức năng: Trả về URL của lời gọi đơn vị WMLScript.
- Cách sử dụng: *URL.getReferer()*
- Ví dụ: URL.getReferer() có thể trả về string "../deck.wml"

Hàm getScheme

- Chức năng: Trả về *scheme* lấy từ URL.
- Cách sử dụng: *URL.getScheme(url)*
- Ví dụ

URL.getScheme("http://www.inhand.net/") Trả về chuỗi "http"

URL.getScheme("../logo.wbmp") Trả về chuỗi ""

URL.getScheme("http://www.inhand.net:xxx/") Trả về invalid

URL.getScheme(invalid) Trả về invalid

Hàm isValid

- Chức năng: Kiểm tra xem URL có hợp lệ không.
- Cách sử dụng: *URL.isValid(url)*
- Ví dụ:

URL.isValid("http://www.inhand.net/") Trả về Boolean true

URL.isValid("../logo.wbmp") Trả về Boolean true

URL.isValid("#frag") Trả về Boolean true

URL.isValid("http://www.inhand.net:xxx/") Trả về Boolean false

URL.isValid(invalid) Trả về invalid

Hàm loadString

- Chức năng: Nhận 1 chuỗi ký tự từ URL.

Chương 21

Hàm và Sử dụng hàm

- Cách sử dụng: *URL.loadString (url, type)*
- Ví dụ:

URL.loadString("http://wap.net/cgi-bin/word.cgi", "text/plain") Trả về chuỗi "Example"

URL.loadString("nonexistent", "text/plain") Trả về integer 404

URL.loadString("foo", "text/")* Trả về invalid

URL.loadString("foo", "image/wbmp") Trả về invalid

Hàm resolve

- Chức năng: Chuyển đổi URL tương đối sang URL tuyệt đối.
- Cách sử dụng: *URL.resolve(base, rel)* với *base*: URL tương đối, *rel*: URL tuyệt đối
- Ví dụ: *URL.resolve("http://www.inhand.net/wap/main.wml", "uk.wml")*
Trả về chuỗi *http://www.inhand.net/wap/uk.wml*

Hàm unescapeString

- Chức năng: loại bỏ mã hóa của một chuỗi URL.
- Cách sử dụng: *URL.unescape(str)* trong đó *str*: chuỗi ký tự
- Ví dụ: *URL.unescapeString("%2Ffoo.cgi%3Ffoo%3D1%26bar%3D2")*
Trả về chuỗi *"/foo.cgi?foo=1&bar=2"*

21.7.5. WMLBrowser Library

Thư viện chuẩn WMLBrowser cung cấp các hàm cho phép tương tác với trình duyệt.

Hàm getCurrentCard

- Chức năng: Lấy URL của *card* hiện thời của trình duyệt.
- Cách sử dụng: *WMLBrowser.getCurrentCard()*
- Ví dụ: *WMLBrowser.getCurrentCard()* có thể Trả về *"../foo.wml#card2"*

Hàm getVar

- Trả về nội dung của biến trình duyệt.
- Cách sử dụng: *WMLBrowser.getVar(name)* trong đó *name*: tên biến
- Ví dụ:

Chương 21

Hàm và Sử dụng hàm

`WMLBrowser.getVar("foo")` Trả về nội dung của foo

`WMLBrowser.getVar("foo")` Trả về chuỗi "" nếu biến foo không thiết lập

`WMLBrowser.getVar("123")` Trả về invalid

Hàm go

- Chức năng: Xác định thi hành tác vụ `<go>`
- Cách sử dụng: `WMLBrowser.go (url)` trong đó url: đường dẫn
- Ví dụ:

`WMLBrowser.go("http://inhand.net/wap/main.wml")` Trả về ""

`WMLBrowser.go(invalid)` Trả về invalid

Hàm newContext

- Chức năng: Xóa nội dung trình duyệt.
- Cách sử dụng: `WMLBrowser.newContext()`
- Ví dụ: `WMLBrowser.newContext()`

Hàm prev

- Chức năng: Xác định thi hành tác vụ `<prev>`
- Cách sử dụng: `WMLBrowser.prev()`
- Ví dụ: `WMLBrowser.prev()`

Hàm refresh

- Chức năng: Thi hành tác vụ `<refresh>`
- Cách sử dụng: `WMLBrowser.refresh()`
- Ví dụ: `WMLBrowser.refresh()`

Hàm setVar

- Chức năng: Thiết lập biến trình duyệt.
- Cách sử dụng: `WMLBrowser.setVar (name, value)` trong đó name: tên biến, value: giá trị.
- Ví dụ:

`WMLBrowser.setVar("foo", "Hello!")` Thiết lập foo là "Hello"

Chương 21

Hàm và Sử dụng hàm

WMLBrowser.setVar("123", "foo")Trả về invalid

21.7.6. Dialogs Library

Thư viện chuẩn *Dialogs* chứa các hàm sinh ra các hộp thoại giúp cho việc giao tiếp với người sử dụng.

Hàm alert

- Chức năng: Hiển thị nội dung cảnh báo tới user.
- Cách sử dụng: *Dialogs.alert(msg)* trong đó *msg*: xâu ký tự hiển thị.
- Ví dụ:

```
Dialogs.alert("Cập nhật dữ liệu thành công!")
```

```
Dialogs.alert("Bạn có muốn thực hiện điều đó không")
```

Hàm confirm

- Chức năng: yêu cầu có hoặc không có đáp ứng từ người sử dụng.
- Cách sử dụng: *Dialogs.confirm(msg, yes, no)* trong đó *msg*: xâu ký tự thông báo.
- Ví dụ:

```
Dialogs.confirm("Reboot Universe now?", "Yes", "No")
```

```
Dialogs.confirm("Really do it?", "OK", "Cancel")
```

```
Dialogs.confirm("Are you sure?", "", "")
```

Hàm prompt

- Chức năng: Lấy một xâu từ người sử dụng.
- Cách sử dụng: *Dialogs.prompt(msg, def)* trong đó *msg*: xâu ký tự hiển thị thông báo, *def*: xâu ký tự trả về từ người sử dụng.
- Ví dụ: *Dialogs.prompt("Please enter a number", "17")*

21.8. Kết chương

Trong chương này chúng ta đã tìm hiểu về về hàm trong WMLScript. Các bạn đã biết cách khai báo, gọi và sử dụng hàm để xây dựng các ứng dụng của mình. Ngoài ra, WMLScript cũng hỗ trợ một thư viện các hàm đã được xây dựng sẵn cung cấp nhiều tính năng quan trọng hỗ trợ người lập trình. Trong chương sau chúng ta sẽ tìm hiểu về cấu trúc điều khiển của WMLScript.

Chương 22

Cấu trúc điều khiển - phần I

Nội dung của chương:

- **Biểu thức và câu lệnh.**
- **Các cấu trúc điều khiển cơ bản của WMLScript.**

22.1. Giới thiệu

Chương này giới thiệu về cấu trúc điều khiển của *WMLScript*. Hầu hết các cấu trúc điều khiển trong *WMLScript* đều tương tự các ngôn ngữ lập trình khác như Java hay C, ngoại trừ việc *WMLScript* không cung cấp cấu trúc lựa chọn như là câu lệnh *switch*.

Tuy nhiên, đó không phải là vấn đề quá quan trọng. *WMLScript* được thiết kế để thực thi các thao tác đơn giản như kiểm tra dữ liệu vào có hợp lệ hay không trước khi gửi dữ liệu tới server. Với một số thao tác phức tạp hơn cần phải có thêm một số tính năng, với các tính năng này lúc đó chúng ta hoàn toàn có thể làm việc trên server.

22.2. Mã giả

Mã giả là loại mã nhân tạo dựa trên ngôn ngữ hàng ngày. Mã giả giúp người lập trình có thể phác thảo thuật giải dễ dàng, trong sáng. Mã giả không phải là một ngôn ngữ lập trình, tuy nhiên qua mã giả người lập trình có thể dễ dàng chuyển đổi sang mã của một ngôn ngữ lập trình cụ thể (ở đây là *WMLScript*).

22.3. Biểu thức và câu lệnh

Mọi biểu thức trong *WMLScript* đều có thể chuyển sang câu lệnh chi đơn giản bằng cách đặt sau biểu thức dấu “;”. Có nghĩa là khi đó biểu thức vẫn được đánh giá nhưng kết quả của nó sẽ bị bỏ qua, tuy vẫn có một số hiệu ứng khác xảy ra. Dạng thông dụng nhất của biểu thức được xem như là một câu lệnh là các phép gán.

Ví dụ: Các câu lệnh gán sau:

```
a = b;  
foo ( );  
x++;  
total += value;
```

22.4. Khối lệnh

Một dạng đơn giản của câu lệnh là một khối các câu lệnh. Điều này có nghĩa là một nhóm nhiều câu lệnh được nằm trong một khối và bao quanh bởi cặp dấu ngoặc "{" và "}".

Khối lệnh được sử dụng rộng rãi trong cấu trúc *if*, cấu trúc lặp *while* và *for*, nó cho phép phần thân của các cấu trúc trên có thể chứa nhiều hơn một câu lệnh.

Ví dụ: Một số khối lệnh:

```
{ } là một khối lệnh rỗng  
{ a = b; } chỉ chứa 1 lệnh  
{ a = b; foo (); bar ( ); } chứa 3 lệnh
```

22.5. Cấu trúc điều khiển

Các loại cấu trúc điều khiển

Các loại cấu trúc điều khiển có thể chia làm hai loại:

- Chương trình thực hiện một câu lệnh sau đó chuyển tới dòng tiếp theo: cấu trúc tuần tự.
- Thi hành câu lệnh khác thay vì câu lệnh tại dòng tiếp theo: bao gồm Cấu trúc lựa chọn và Cấu trúc lặp.

Sơ đồ khối (Flowchart)

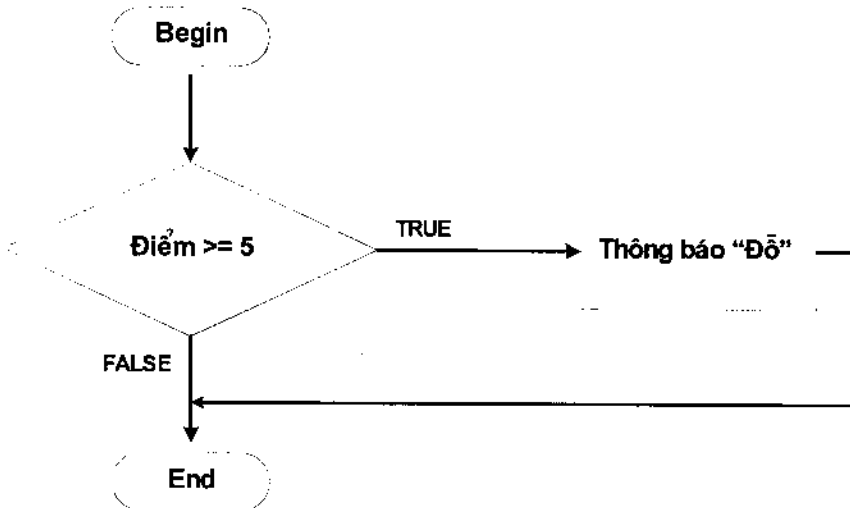
Sử dụng để phác họa chương trình và minh họa thứ tự các sự kiện có thể xảy ra. Các ký pháp trong sơ đồ khối bao gồm

- Hình chữ nhật: biểu thị hoạt động.
- Hình *oval* sử dụng để biểu thị bắt đầu và kết thúc chương trình.
- Hình tròn sử dụng như là một liên kết.
- Hình thoi biểu diễn điều kiện/quyết định.

Các cấu trúc điều khiển có thể kết hợp với nhau khối này tiếp sau khối kia, cấu trúc này có thể chứa cấu trúc khác.

22.6. Cấu trúc lựa chọn if

- Cấu trúc *if* là cấu trúc quyết định dựa trên điều kiện. Biểu thức của cấu trúc *if* được đánh giá là kiểu boolean: nếu *true* thì thi hành một hoạt động, nếu *false* thì bỏ qua hoạt động. Cấu trúc *if* chỉ có 1 đầu vào và 1 đầu ra.



Hình 22.1. Sơ đồ khối cấu trúc if.

- Cú pháp: *if (điều kiện) statement-when-true*. Trong đó *điều kiện* có thể là một biểu thức bất kỳ và giá trị của biểu thức sẽ được chuyển sang kiểu *boolean*. Nếu giá trị là *true*, phần lệnh *statement-when-true* sẽ được thi hành.
- Ví dụ:

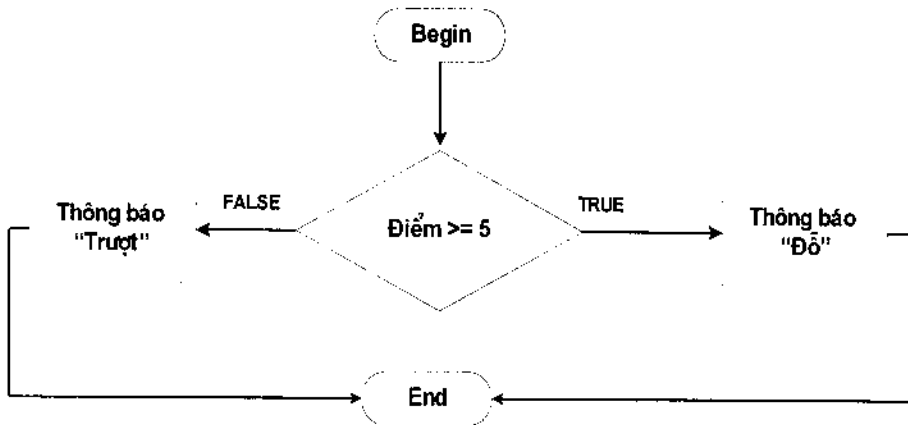
```
if (x > max)
    max = x;
```

22.7. Cấu trúc if/else

Cấu trúc *if/else*:

- Có một đầu vào và hai lựa chọn.
- Nếu biểu thức điều kiện đúng thì thực hiện một hành động.
- Khi biểu thức điều kiện sai, một hành động khác sẽ được thực hiện.

Nếu khối lệnh có nhiều lệnh, chúng ta có thể đưa các lệnh vào trong cặp dấu ngoặc "{" và "}"



Hình 22.2. Sơ đồ khối cấu trúc if/else.

Cú pháp lệnh:

```
if (condition)
    statement-when-true
else
    statement-when-false
```

Ví dụ:

```
if (number == 1)
    result = "1 apple";
else
    result = number + " apples";
```

Đôi khi, bản thân khối lệnh *statement-when-false* lại là một câu lệnh *if* khác. Lúc đó điều kiện đầu tiên sẽ được tính trước, nếu nó là *false* hoặc không thể chuyển đổi sang *boolean* thì câu lệnh *if* tiếp theo sẽ được thi hành.

Ví dụ

```
if (x > max)
    max = x;
else if (x < min)
```

Chương 22

Cấu trúc điều khiển - phần I

```
min = x;
```

Chú ý:

Có một vấn đề nảy sinh nếu khối lệnh *statement-when-true* chứa một câu lệnh *if* với mệnh đề *else*, ví dụ

```
if (x)
  if (y)
    foo ( );
else
  bar ( );
```

Ở đây dễ gây ra sự khó hiểu mệnh đề *else* thuộc về câu lệnh *if* nào.

WMLScript giải quyết vấn đề trên bằng cách quy định mệnh đề *else* được gắn với câu lệnh *if* gần nhất. Tuy nhiên để cú pháp trong sáng hơn nên đặt khối lệnh *statement-when-true* và khối lệnh *statement-when-false* bằng cặp dấu ngoặc "{" và "}".

Ví dụ

```
if (x) {
  if (y)
    foo ( );
else
  bar ( );
}
```

Một vài người lập trình có thói quen luôn luôn đặt cặp ngoặc nhọn thậm chí chỉ là một câu lệnh, điều này rất đáng khuyến khích vì nó làm cho đoạn mã trong sáng và dễ hiểu hơn.

Một số ví dụ về cấu trúc if/else

Ví dụ 21.1.

Bài toán đổi tiền

Yêu cầu đổi một lượng tiền từ một trong bốn loại: USD, Mark Đức, Franc và Lira sang một trong ba loại còn lại.

- File *converter.wmls*: Chứa các hàm WMLScript phục vụ việc chuyển đổi tiền

```
//converter.wmls
extern function convert(){
    var scur = WMLBrowser.getVar("startCur");
    var ecur = WMLBrowser.getVar("endCur");
    var cur = WMLBrowser.getVar("cur");
    var curVal = Lang.parseInt(cur);
    if(curVal){
        if(scur != ecur){
            var endval = convertVal(scur, ecur, curVal);
        }
        else { endval = cur;}
        endval = String.format("%.2f", endval);
        WMLBrowser.setVar("ans", endval);
        WMLBrowser.go("currency.wml#convD");
    }
    else{
        error(cur);
    }
}

function error(badVal){
    Dialogs.alert("Giá trị " + badVal + " không hợp lệ ");
    WMLBrowser.setVar("cur", "");
    WMLBrowser.go("#valCard");
}

function convertVal(scur, ecur, val){
    // Thiết lập tỉ giá hối đoái, lấy USD làm chuẩn
```

Chương 22

Cấu trúc điều khiển - phần I

```
var usd = 1;
var lira = 1858.22;
var dmark = 1.6535;
var franc = 5.5425;
// Chuyển hết sang USD
if (scur == "USD"){
} else if (scur == "Lira"){
    val = val / lira;
} else if (scur == "DMark"){
    val = val / dmark;
} else if (scur == "Franc"){
    val = val / franc;
}
// convert to dest currency
if (ecur == "Lira"){// Đổi sang Lira
    return(val*lira);
} else if (ecur == "DMark"){// Đổi sang Mark
    return(val*dmark);
} else if (ecur == "Franc"){// Đổi sang Franc
    return(val*franc);
} else{// Đổi sang USD
    return(val);
}
}
```

- File *currency.wml*

```
<!--currency.wml ->
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
    "http://www.phone.com/dtd/wml11.dtd" >
<wml>
<card id="first" >
<onevent type="onenterforward">
    <refresh>
        <setvar name="startCur" value=""/>
        <setvar name="endCur" value=""/>
        <setvar name="cur" value=""/>
    </refresh>
</onevent>
<p>
<do type="accept" label="conv">
    <go href="#destVal"/>
</do>
    Chon loai tien can doi?
    <select name="startCur">
        <option value="USD">USD</option>
        <option value="DMark">D-Mark</option>
        <option value="Franc">Franc</option>
        <option value="Lira">Lira</option>
    </select>
</p>
</card>
```

Chương 22

Cấu trúc điều khiển - phần I

```
<card id="destVal">
  <p>
    <do type="accept" label="dest">
      <go href="#valCard"/>
    </do>
    Đổi từ tiền $startCur sang:
    <select name="endCur">
      <option value="USD">USD</option>
      <option value="DMark">D-Mark</option>
      <option value="Franc">Franc</option>
      <option value="Lira">Lira</option>
    </select>
  </p>
</card>
<card id="valCard">
  <onevent type="onenterforward">
    <refresh>
      <setvar name="cur" value=""/>
      <setvar name="ans" value=""/>
    </refresh>
  </onevent>
  <p>
    <do type="accept" label="conv">
      <go href="converter.wmls#convert()"/>
    </do>
    Lương tiền $startCur?
```

Chương 22

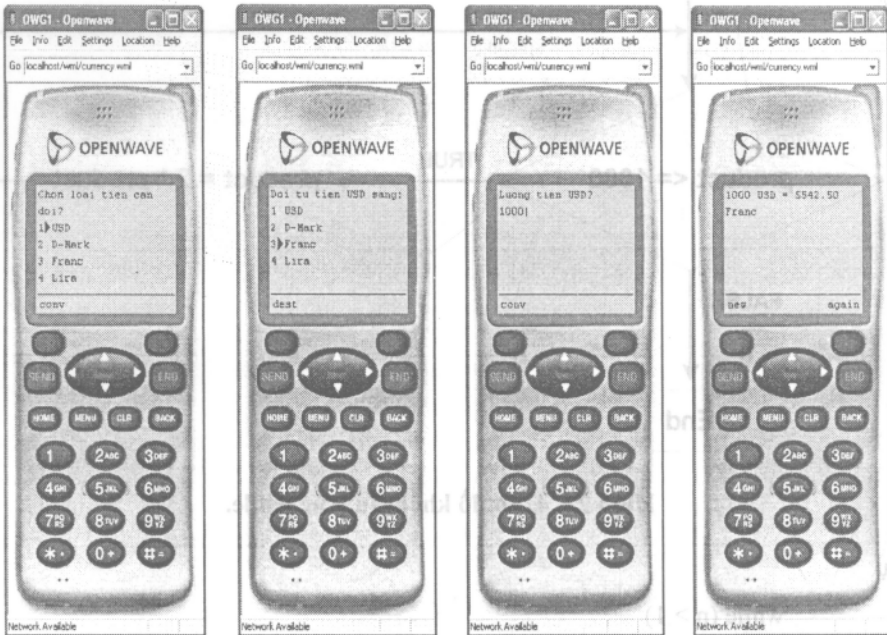
Cấu trúc điều khiển - phần I

```
<input type="text" name="cur" format="*N"/>
</p>
</card>
<card id="convD">
<p>
<do type="accept" label="new">
  <go href="#first">
    <setvar name="startCur" value=""/>
    <setvar name="endCur" value=""/>
    <setvar name="cur" value=""/>
  </go>
</do>
<do type="options" label="again">
  <go href="#valCard"/>
</do>
  $cur $startCur = $ans $endCur
</p>
</card>
</wml>
```


Chương 22

Cấu trúc điều khiển - phần I

▪ Kết quả thực hiện chương trình



Hình 22.3. Bài toán đổi tiền.

22.8. Cấu trúc while

Cấu trúc *while* là cấu trúc sử dụng để thi hành lặp lại một công việc. Công việc còn được thực hiện khi điều kiện còn đúng (*true*) và kết thúc khi điều kiện sai (*false*). Phần thân của cấu trúc *while* có thể chứa một hoặc một khối các lệnh.

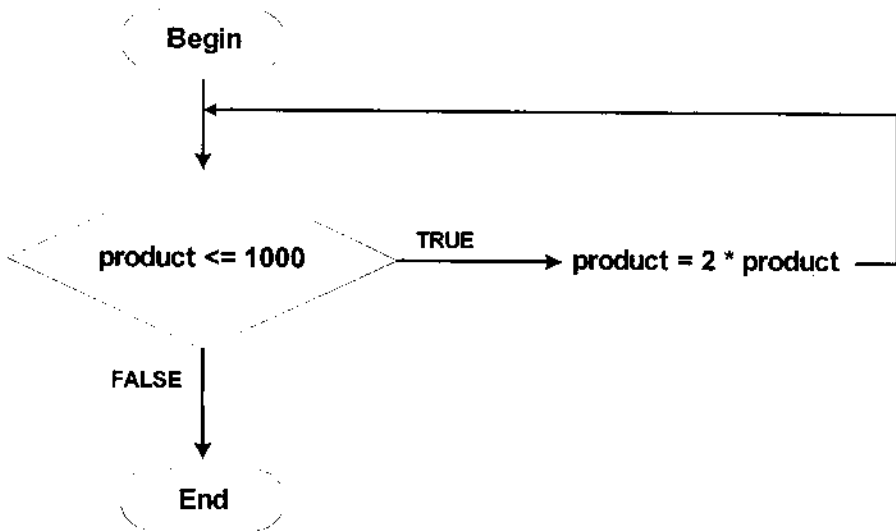
Khi sử dụng cấu trúc *while*, phải lưu ý điểm thay đổi giá trị điều kiện để tránh trường hợp vòng lặp vĩnh viễn.

Cú pháp

while (*condition*)

body

Trong đó: *condition* là điều kiện, *body*: thân câu lệnh *while* (có thể là 1 hoặc 1 khối lệnh)



Hình 22. 4. Sơ đồ khối cấu trúc **while**.

Ví dụ

```
while (n > 1)
{
    total *= n;
    --n;
}
```

Ví dụ 22.2.

Ví dụ về cấu trúc *while*: Tính trung bình cộng của N số (N tùy ý) do người sử dụng nhập vào.

Chương trình thi hành như sau: yêu cầu người sử dụng nhập số bất kỳ vào từ bàn phím. Nếu muốn ngừng nhập thì nhập giá trị (-1), sau đó chương trình sẽ tính trung bình cộng của tất cả các số đã được nhập vào

- File *trungbinh.wmls* có hàm *trungbinh()* yêu cầu người sử dụng nhập N giá trị và sau đó tính giá trị trung bình của N giá trị đó

```
//trungbinh.wmls
extern function trungbinh()
{
    var total = 0;      // tổng của N số
    var gradeCounter = 0; // chuẩn bị lặp
    var average;
    // Yêu cầu người sử dụng nhập
    var grade = Dialogs.prompt( "Nhập vào 1 số," +
        "(-1) để thoát", "" );
    // chuyển từ chuỗi sang số
    var gradeValue = Lang.parseInt( grade );
    while ( gradeValue != (-1) ) {
        // cộng vào tổng
        total = total + gradeValue;
        // tăng số lượng số đã nhập
        ++gradeCounter;
    // Yêu cầu nhập tiếp
        grade = Dialogs.prompt( "Nhập vào 1 số," +
            "(-1) để thoát", "" );
        gradeValue = Lang.parseInt( grade );
    }
    // tính trung bình
    if ( gradeCounter != 0 ) {
        average =
            String.format( "%3.2f", ( total / gradeCounter ) );
    }
}
```

Chương 22

Cấu trúc điều khiển - phần I

```
else {  
    average = "Ban khong nhap so nao"; // N=0  
}  
WMLBrowser.setVar( "classAve", average );  
WMLBrowser.go( "#classAverage" );  
}
```

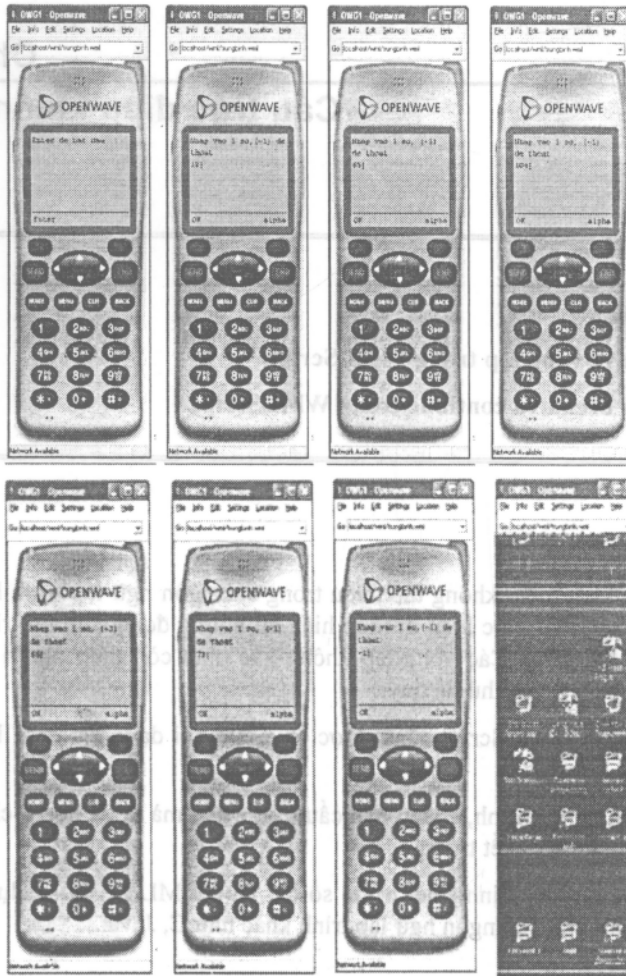
- File *trungbinh.wml* có nhiệm vụ gọi hàm *trungbinh()* của file *trungbinh.wmls*

```
<!-- trungbinh.wml -->  
<?xml version="1.0"?>  
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"  
    "http://www.phone.com/dtd/wml11.dtd" >  
  
<wml>  
    <card id = "index" title = "Class Average">  
        <!-- soft key that calls function average2 -->  
        <do type = "accept" label = "Enter">  
            <go href = "trungbinh.wmls#trungbinh()"/>  
        </do>  
        <p>  
            Enter de bat dau  
        </p>  
    </card>  
  
    <card id = "classAverage" title = "Class Average">  
        <p>  
            Trung binh cong: $classAve <!-- display result -->  
        </p>  
    </card>  
</wml>
```

Chương 22

Cấu trúc điều khiển - phần I

- Kết quả thực hiện chương trình:



Hình 22.5. Tính trung bình cộng của n số nguyên.

22.9. Kết chương

Chương này đã giới thiệu về cấu trúc điều khiển của WMLScript, bao gồm các cấu trúc lựa chọn *if*, cấu trúc *if/else*, cấu trúc *while*. Hầu hết các cấu trúc điều khiển trong WMLScript đều tương tự các ngôn ngữ lập trình khác ngoại trừ việc WMLScript không cung cấp cấu trúc lựa chọn như là câu lệnh *switch*. Trong chương sau chúng ta sẽ tìm hiểu thêm một số cấu trúc điều khiển khác trong WMLScript.

Chương 23

Cấu trúc điều khiển - phần II

Nội dung của chương:

- Tìm hiểu về vòng lặp trong WMLScript.
- Câu lệnh **break** và **continue** trong WMLScript.

23.1. Giới thiệu

Vòng lặp là khái niệm không thể thiếu trong các ngôn ngữ lập trình hiện nay bởi nó tránh cho người dùng phải thực hiện đi thực hiện lại những đoạn chương trình giống nhau, rút ngắn thời gian lập trình. Các vòng lặp không xác định còn giúp người dùng linh hoạt trong sử dụng các thuật toán, khử đệ quy....

Vòng lặp trong WMLScript cũng được chia làm hai dạng chính là lặp xác định và không xác định.

- Vòng lặp không xác định, chẳng hạn cấu trúc *while* mà ta đã học ở chương trước, là lặp với số lần không biết trước.
- Với cấu trúc lặp xác định (biết trước số lần lặp) WML Script sử dụng cấu trúc *for* cũng giống như một số ngôn ngữ lập trình khác như C, Java..., .

Cấu trúc *for*

Quan sát cấu trúc For trên hình 23.1 và 23.2.

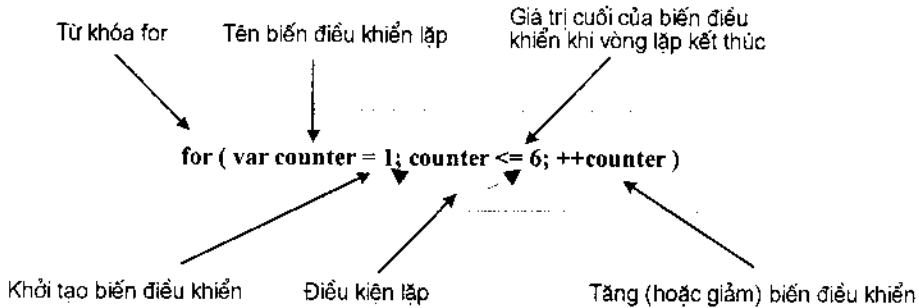
Cấu trúc *for* gồm 3 thành phần:

- Thành phần khai báo và khởi tạo biến điều khiển
- Thành phần là điều kiện lặp
- Thành phần điều khiển (tăng, giảm) biến điều khiển

Chương 23

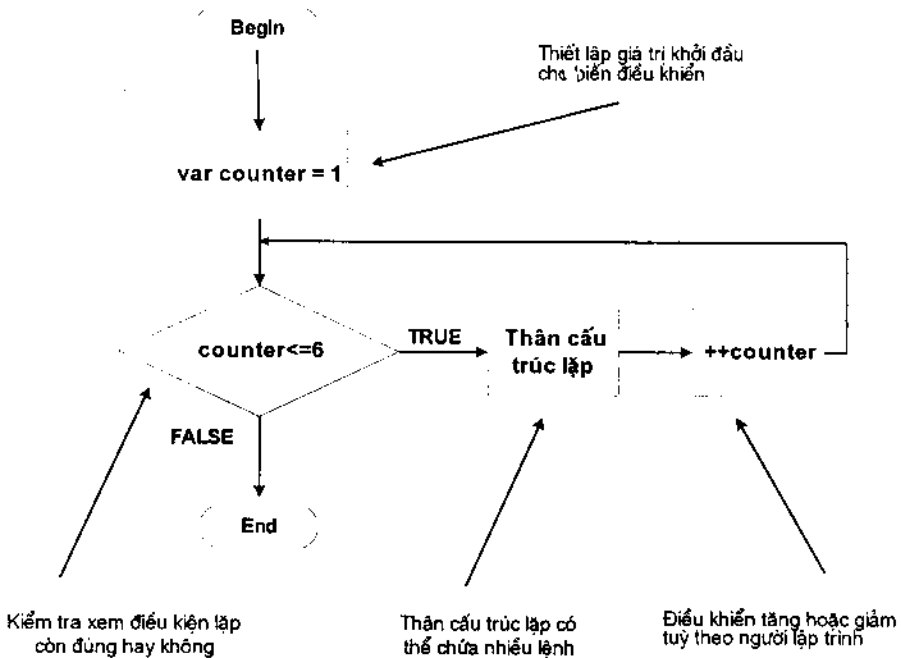
Cấu trúc điều khiển - phần II

Khi bắt đầu cấu trúc, giá trị khởi đầu của biến điều khiển được thiết lập, trước mỗi lần lặp, phần điều kiện lặp được kiểm tra, nếu điều kiện đúng (true) thì thân cấu trúc được thực hiện và biến điều khiển được tăng lên (hoặc giảm đi) ngược lại, nếu điều kiện lặp sai (false), cấu trúc lặp kết thúc.



Hình 23.1. Cấu trúc lặp for tổng quát và các thành phần.

Sơ đồ khối của cấu trúc for tổng quát:



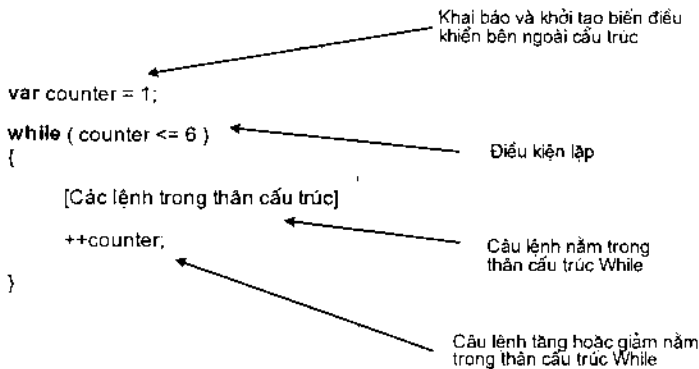
Hình 23.2. Sơ đồ khối của cấu trúc for tổng quát.

Chương 23

Cấu trúc điều khiển - phần II

Tuy nhiên nếu hiểu cấu trúc lặp xác định theo nghĩa là biết trước số lần lặp thì ta có thể có một cấu trúc lặp xác định khác dựa trên cấu trúc *while* đã học ở chương trước. Để biến cấu trúc *while* thành cấu trúc lặp xác định (biết trước số lần lặp) thì ta làm như sau:

- Phần điều kiện lặp của cấu trúc *while* giống như phần điều kiện lặp của cấu trúc *for*
- Trong thân cấu trúc *while* sẽ có 1 lệnh tương tự như phần tăng giảm biến điều khiển của cấu trúc *for*
- Biến điều khiển của cấu trúc *while* được khởi tạo ở bên ngoài cấu trúc



Hình 23. 3. Sử dụng cấu trúc *while* để thực hiện cấu trúc lặp xác định.

Ví dụ 23.1.

Ví dụ sử dụng cấu trúc *while* để lặp xác định: viết chương trình hiện sáu lần dòng "xin chào các bạn"

- File *lapWhile.wmls* chứa hàm thực hiện lặp

```
// lapWhile.wmls
// Lặp xác định sử dụng cấu trúc while
extern function repetition()
{
    var counter = 1;           // khởi tạo biến điều khiển

    while ( counter <= 6 ) {    // điều kiện lặp
```


Chương 23

Cấu trúc điều khiển - phần II

```
var result = result + "Xin chao cac ban\n";

++counter;           // tăng biến điều khiển
}
WMLBrowser.setVar( "output", result );
WMLBrowser.go( "#card2" );
}
```

- File *lapxacdinh1.wml*

```
<!--lapxacdinh1.wml      -->
<!-- Lặp có điều kiện sử dụng cấu trúc while -->
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
      "http://www.phone.com/dtd/wml11.dtd" >

<wml>
  <card id = "index" title = "Repetition">
    <do type = "accept" label = "OK">
      <go href = "lapWhile.wmls#repetition()"/>
    </do>
    <p>
      Click OK de thuc hien 6 lan lap.
    </p>
  </card>
  <card id = "card2" title = "Repetition">
    <do type = "accept" label = "Back">
      <prev />
    </do>
    <p>
```

Chương 23

Cấu trúc điều khiển - phần II

```
$output <!--Hiện thị kết quả -->  
<p>  
</card>  
</wml>
```

- Kết quả thực hiện:



Hình 23.4. Kết quả ví dụ 23.1.

Ví dụ 23. 2.

Ví dụ về thực hiện lặp xác định sử dụng cấu trúc *for*. Thực hiện sáu lần lặp, mỗi lần hiện ra thứ tự lần lặp đó

- File *lapFor.wmls* chứa hàm thi hành việc lặp bằng cấu trúc *for*

```
// lapFor.wmls  
// Lặp có điều kiện sử dụng cấu trúc for  
extern function repetition()
```

Chương 23
Cấu trúc điều khiển - phần II

```
{  
    // khởi tạo, điều kiện lặp và điều khiển biến lặp  
    // đầu được gói trong cùng 1 cấu trúc for  
    for ( var counter = 1; counter <= 6; ++counter ) {  
        var result = result + " Lan lap thu " + counter + "\n";  
    }  
    WMLBrowser.setVar( "output", result );  
    WMLBrowser.go( "#card2" );  
}
```

- File *lapxacdinh2.wml* gọi hàm thực hiện lặp của file *lapFor.wmls*

```
<!-- lapxacdinh2.wml -->  
<!-- Lặp xác định sử dụng cấu trúc for -->  
<?xml version="1.0"?>  
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"  
    "http://www.phone.com/dtd/wml11.dtd" >  
<wml>  
    <card id = "index" title = "Repetition">  
  
        <!-- Chuẩn bị gọi hàm lặp -->  
        <do type = "accept" label = "OK">  
            <go href = "lapFor.wmls#repetition()"/>  
        </do>  
  
        <p>  
            Click OK để thực hiện lặp.  
        </p>  
    </card>
```

Chương 23

Cấu trúc điều khiển - phần II

```
<card id = "card2" title = "Repetition">  
  <do type = "accept" label = "Back">  
    <prev />  
  </do>  
  <p>  
    $output <!-- display result -->  
  </p>  
</card>  
</wml>
```

- Kết quả thực hiện



Hình 23.5. Kết quả ví dụ 23.2.

23.2. Câu lệnh break và continue

Để hiểu rõ về cấu trúc *break* và *continue*, hãy xét một bài toán nhỏ sau: tìm 10 số nguyên tố từ 1 đến 10000 sao cho tổng các chữ số của nó cũng là số nguyên tố, hiện các số đó và tính tổng của chúng. Nghĩa là: giả sử số *abcd* là nguyên tố thì $a+b+c+d$ cũng là nguyên tố.

Chương 23

Cấu trúc điều khiển - phần II

Để giải quyết bài toán này một cách đơn giản, ta sẽ dùng cấu trúc lặp, giả sử đó là cấu trúc *while*. Bài toán trên được viết bằng mã giả như sau

```
var dem = 0;
while (dem <= 10000)
{
    ++dem;
    if (dem không là nguyên tố)
        continue;
    if (tổng các chữ số của dem là không nguyên tố)
        continue;
    if (đã tìm đủ 10 số)
        break;
}
```

Ta sẽ phân tích đoạn chương trình trên:

- Khi kiểm tra lần thứ nhất biến *dem* không phải là nguyên tố, ngay lập tức kết thúc lần lặp hiện thời và chuyển sang lần lặp khác (để tránh mất thời gian vào việc thực hiện các lệnh sau) bằng lệnh *continue* đầu tiên.
- Khi kiểm tra lần thứ hai nếu tổng các chữ số của *dem* không phải là nguyên tố thì cũng ngắt luôn lần lặp này để chuyển sang lần khác bằng lệnh *continue* thứ 2.
- Khi đã tìm đủ 10 kết quả, không cần phải tìm tiếp khi đó kết thúc vòng lặp.

Như vậy lợi ích của 2 lệnh *break* và *continue* rất rõ ràng, nếu không có 2 lệnh này thì khi thực hiện chương trình trên, ta sẽ phải thực hiện 10000 lần lặp, mỗi lần lặp ta phải thực hiện tất cả 3 lệnh *if* (ngoài các lệnh bắt buộc trong thân hàm). Mặt khác, lệnh *break* và *continue* làm cho chương trình linh hoạt hơn.

Câu lệnh *break*

Lệnh *break* sử dụng để thoát ra khỏi cấu trúc lặp *while* hoặc *for* gần nhất và chương trình sẽ tiếp tục thi hành tại câu lệnh ngay tiếp theo cấu trúc lặp bị ngắt.

Câu lệnh *break* được đặt trong thân của cấu trúc lặp cần ngắt.

Ta sẽ giải thích rõ hơn về “*cấu trúc lặp while hoặc for gần nhất*” mà ta đã đề cập ở trên.

Chương 23

Cấu trúc điều khiển - phần II

Xét đoạn chương trình sau

```
for (i=1; i<=1000; ++i)
{
  [Các lệnh thân cấu trúc for]
while (i là nguyên tố )
{
  [Các lệnh thân cấu trúc while]
break;
}
}
```

Như vậy khi được gọi đến câu lệnh *break* chỉ có tác dụng thoát khỏi cấu trúc *while* (cấu trúc lặp gần nhất), không thoát ra khỏi cấu trúc *for* được.

Ví dụ 23.3.

Ví dụ về sử dụng câu lệnh *break*

- File *break.wmls*

```
// break.wmls
// Minh họa lệnh break
extern function counter()
{
  for ( var count = 1; count <= 10; ++count ) {
    var newCount = newCount + "dem = " + count + " \n";
    if ( count == 3 ){
      break;          // break out of loop if count equals 3
    }
  }
  var broke = "Break vong lap tai vi tri " + count;
  WMLBrowser.setVar( "brokeVal", broke );
}
```

Chương 23

Cấu trúc điều khiển - phần II

```
WMLBrowser.setVar( "countVal", newCount );
WMLBrowser.go( "#card2" );
}
```

▪ File *break.wml*

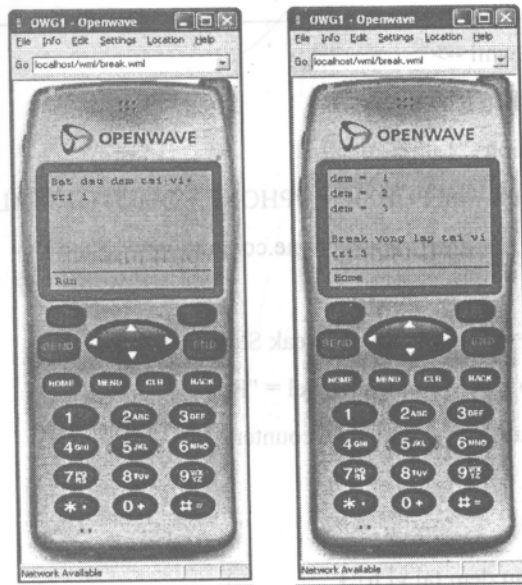
```
<!--break.wml -->
<!--Mình hoạ lệnh break -->
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
    "http://www.phone.com/dtd/wml11.dtd" >
<wml>
  <card id = "index" title = "Break Statement">
    <do type = "accept" label = "Run">
      <go href = "break.wmls#counter()" />
    </do>
    <p>
      Bắt đầu đếm tại vị trí 1
    </p>
  </card>
  <card id = "card2" title = "Break Statement">
    <do type = "accept" label = "Home">
      <prev />
    </do>
    <p>
      <!-- display results -->
      $countVal <br />
    </p>
  </card>
</wml>
```

Chương 23

Cấu trúc điều khiển - phần II

```
$brokeVal  
</p>  
</card>  
</wml>
```

- Kết quả thực hiện



Hình 23.6. Kết quả ví dụ 23.3.

Câu lệnh *continue*

Khác với câu lệnh *break*, câu lệnh *continue* không sử dụng để kết thúc toàn bộ cấu trúc lặp mà *continue* sử dụng để kết thúc một lần lặp và chuyển sang lần lặp tiếp theo.

Trong thân cấu trúc lặp, khi gặp lệnh *continue* thì:

- Đối với cấu trúc *while*, nó trở về phần điều kiện của cấu trúc.
- Đối với cấu trúc *for*, nó trở về biểu thức cập nhật biến điều khiển (thành phần thứ 3 của cấu trúc và sau đó kiểm tra điều kiện).

Chương 23

Cấu trúc điều khiển - phần II

Chú ý: Nếu sử dụng câu lệnh *break* và *continue* ở bên ngoài cấu trúc *for* hoặc *while* thì chương trình sẽ báo lỗi.

Ví dụ 23.4.

Ví dụ về sử dụng câu lệnh *continue*

- File *continue.wmls*

```
// continue.wmls
// Minh hoạ lệnh continue
extern function counter()
{
    for ( var count = 1; count <= 3; ++count ) {
        if ( count == "2" ){
            continue; // bỏ qua 2
        }
        var newCount = newCount + "dem = " + count + "\n";
    }
    var cont = "Sử dụng lệnh continue để bỏ qua vòng lặp thứ 2";
    WMLBrowser.setVar( "brokeVal", cont );
    WMLBrowser.setVar( "countVal", newCount );
    WMLBrowser.go( "#card2" );
}
```

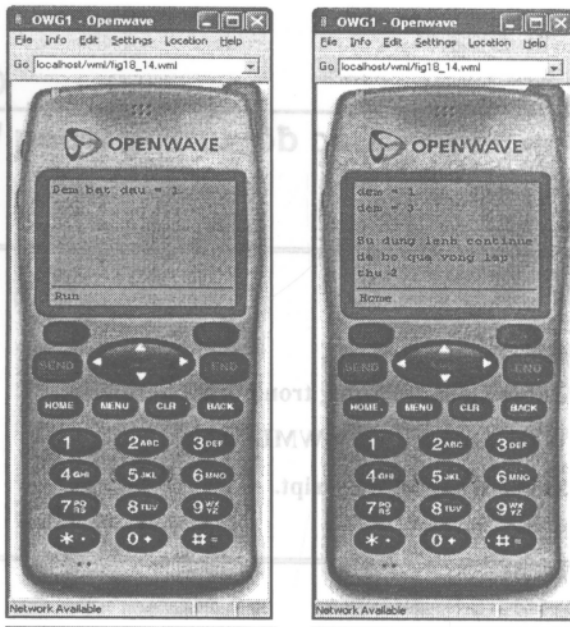
- File *continue.wml*

```
<!--continue.wml -->
<!--Minh hoạ lệnh continue -->
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
    "http://www.phone.com/dtd/wml11.dtd" >
<wml>
  <card id="index" title="Cont Statement">
    <do type="accept" label="Run">
      <go href="continue.wmls#counter()" />
    </do>
    <p>
      Dem bat dau = 1
    </p>
  </card>
  <card id="card2" title="Cont Statement">
    <do type="accept" label="Home">
      <prev />
    </do>
    <p>
      $countVal <br/>
      $brokeVal
    </p>
  </card>
</wml>
```

- Kết quả thực hiện:

Chương 23

Cấu trúc điều khiển - phần II



Hình 23.7. Kết quả của ví dụ 23.4.

23.3. Kết chương

Trong chương này chúng ta đã tìm hiểu thêm về vòng lặp trong WMLScript, câu lệnh *break* và *continue* trong WMLScript. Trong chương sau chúng ta sẽ tìm hiểu về các đối tượng trong WMLScript.

Chương 24

Các đối tượng trong WMLScript

Nội dung của chương:

- Tìm hiểu về khái niệm đối tượng trong WMLScript.
- Đối tượng WMLBrowser trong WMLScript.
- Đối tượng String trong WMLScript.

24.1. Giới thiệu

Tại chương 21 - sử dụng hàm trong WMLScript ta đã làm quen với khái niệm thư viện chuẩn WMLScript. WMLScript cung cấp sáu thư viện chuẩn:

- Lang library
- Float library
- String library
- URL library
- WMLBrowser library
- Dialogs library

Ngoài việc gọi tên là các “thư viện chuẩn”, WMLScript còn gọi chúng bằng những tên khác như là “đối tượng” (*object*). Kể từ bây giờ ta sẽ gọi chúng là các đối tượng và để gọi một hàm (phương thức) của đối tượng sử dụng cú pháp:

<tên đối tượng>. <tên hàm>

Mỗi đối tượng chứa các hàm được dựng sẵn phục vụ cho một công việc nào đấy. Khi cần sử dụng thì gọi nó. Tất cả các hàm và thủ tục của một thư viện đã được xem xét tổng thể tại chương 21. Chương này sẽ nghiên cứu kỹ thêm về hai đối tượng rất quan trọng đó là đối

Chương 24

Các đối tượng trong WMLScript

tượng *WMLBrowser* và *String*.

24.2. Đối tượng *WMLBrowser*

Trong quá trình lập trình WMLScript, thường phải gọi tới thông tin chứa trong trình duyệt bởi ngữ cảnh của trình duyệt. Ngữ cảnh của trình duyệt chứa các biến được thiết lập bởi chương trình và các biến thiết lập bởi bản thân trình duyệt.

Đối tượng *WMLBrowser* chứa các phương thức dưới đây cho phép thiết lập truy cập tới ngữ cảnh của trình duyệt.

Bảng 24.1. các phương thức của *WMLBrowser*

Phương thức	Mô tả
<code>getCurrentCard()</code>	Trả về địa chỉ URL(tương đối) của thẻ hiện thời
<code>getVar()</code>	Trả về giá trị của biến
<code>go()</code>	Chuyển tới 1 thẻ mới, xác định bởi URL và trả về xâu rỗng
<code>newContext()</code>	Xoá toàn bộ các biến và trả về xâu rỗng
<code>prev()</code>	Trình duyệt quay trở về thẻ trước đó và trả về xâu rỗng
<code>refresh()</code>	Làm tươi thẻ hiện thời và trả về xâu rỗng
<code>setvar()</code>	Thiết lập giá trị cho 1 biến và trả về giá trị true nếu giá trị mới được thiết lập thành công ngược lại trả về giá trị false

Ví dụ 24.1.

Thiết lập biến trình duyệt bằng đối tượng *WMLBrowser*

- File *setvar.wmls*

```
// setvar.wmls
extern function method()
{
    var x = "Xin chào!";
    WMLBrowser.setVar( "var1", x);
    WMLBrowser.refresh();
}
```

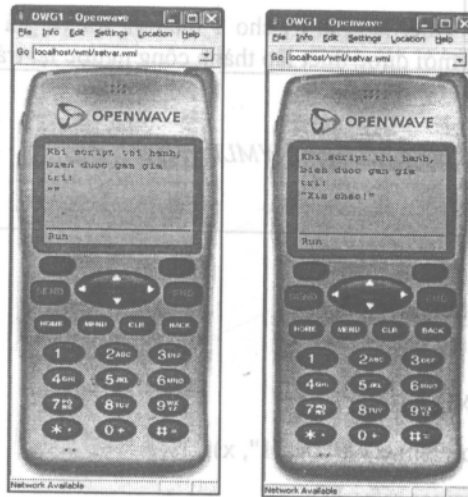
- File *setvar.wml*

Chương 24

Các đối tượng trong WMLScript

```
<!-- setvar.wml -->
<!-- Thiết lập biến trình duyệt bằng đối tượng WMLBrowser -->
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM/DTD WML 1.1/EN"
"http://www.phone.com/dtd/wml11.dtd" >
<wml>
  <card id = "index" title = "Refresh">
    <do type = "accept" label = "Run">
      <go href = "functionSet1.wmls#method()" />
    </do>
    <p>
      Khi script thi hành, biến được gán giá trị:<br />"$var1"
    </p>
  </card>
</wml>
```

■ Kết quả thực hiện:



Hình 24.1. Thiết lập biến trình duyệt.

Chương 24

Các đối tượng trong WMLScript

24.3. Đối tượng String

Đối tượng *String* chứa các phương thức cho phép thao tác với chuỗi ký tự như tách chuỗi thành các ký tự đơn, tính độ dài chuỗi... Có một điều cần chú ý là tất cả các hàm không làm thay đổi giá trị của chuỗi ký tự là đối số mà sẽ trả về một chuỗi khác, có nghĩa là chuỗi nguyên gốc không bị thay đổi.

Bảng 24.2. Các phương thức của đối tượng String

Phương thức	Mô tả
charAt(id)	Trả về ký tự tại vị trí có chỉ số <i>id</i> , nếu không có thì trả về chuỗi rỗng. lưu ý: vị trí ký tự đầu tiên trong 1 chuỗi bắt đầu từ chỉ số 0
compare(str1, str2)	So sánh 2 chuỗi <i>str1</i> và <i>str2</i> cho các kết quả sau 1, 0, -1: ▪ -1 nếu <i>str1</i> "nhỏ hơn" <i>str2</i> ▪ 0 nếu 2 chuỗi "bằng nhau" ▪ Còn lại thì bằng 1
elementAt(string, id, separator)	Tìm trong chuỗi bắt đầu từ chỉ số thứ <i>id</i> ngăn cách bởi dấu cách
elements(string, separator)	Trả về số thành phần chứa trong chuỗi <i>string</i> được ngăn cách bởi dấu cách
find(string, substring)	Trả về chỉ số của lần gặp chuỗi con <i>substring</i> đầu tiên trong chuỗi cần tìm <i>string</i>
format(format, value)	Định dạng chuỗi
insertAt(string, element, id, separator)	Trả về 1 chuỗi mới với 1 thành mới và 1 separator được chèn tại vị trí <i>id</i>
isEmpty(string)	Trả về giá trị <i>true</i> nếu độ dài chuỗi = 0 (chuỗi rỗng) ngược lại trả về giá trị <i>false</i>
length(string)	Trả về độ dài chuỗi
removeAt(string, id, separator)	Xoá chuỗi bắt đầu từ vị trí <i>id</i> cho đến khi gặp dấu cách

Chương 24

Các đối tượng trong WMLScript

<code>replace(string,old,new)</code>	Thay thế chuỗi <i>old</i> bởi chuỗi <i>new</i> trong chuỗi <i>string</i>
<code>replaceAt(string,element,id,separator)</code>	Thay thế thành phần
<code>squeeze(string)</code>	Xoá nhiều khoảng trắng trong chuỗi để trong chuỗi chỉ chứa 1 khoảng trắng
<code>substr(string,startId,length)</code>	Lấy chuỗi con từ vị trí <i>startId</i> với độ dài <i>length</i>
<code>toString(value)</code>	Chuyển đổi 1 giá trị nào đó thành chuỗi
<code>trim(string)</code>	Cắt bỏ các khoảng trắng tại đầu và cuối chuỗi

Ví dụ 24.3.

Ví dụ minh hoạ phương thức xử lý *string*: thực hiện xác định kí tự tại một vị trí trong một chuỗi.

- File *charAt.wmls*

```
// charAt.wmls
// Minh hoạ phương thức charAt
extern function methodA()
{
    var s = "HANOI-VIETNAM"; // chuỗi thao tác
    var variable1; // chuỗi kết quả
    // tạo chuỗi kết quả
    variable1 = "Kí tự tại vị trí chỉ số 4 trong chuỗi " + s + " là " +
        String.charAt( s, 4 );
    WMLBrowser.setVar( "method1", variable1 );
    WMLBrowser.go("#card2");
}
```

- File *charAt.wml*

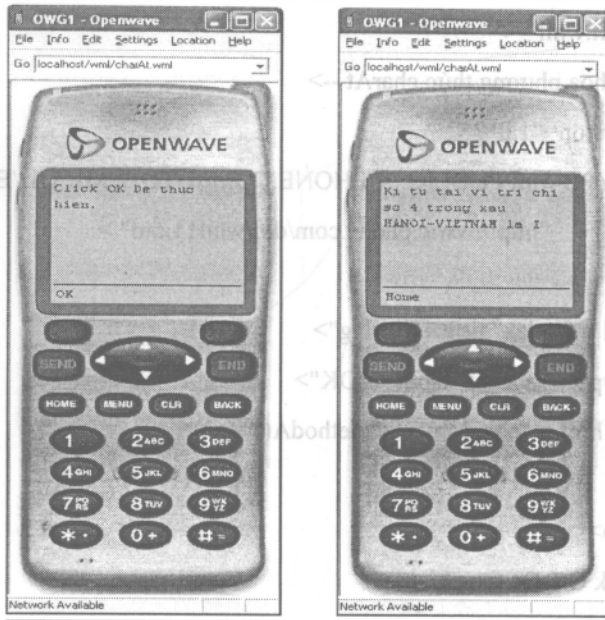

```
<!-- charAt.wml -->
<!--Minh hoạ phương thức charAt -->
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
      "http://www.phone.com/dtd/wml11.dtd" >

<wml>
  <card id = "index" title = "String">
    <do type = "accept" label = "OK">
      <go href = "charAt.wmls#methodA()" />
    </do>
    <p>
      Click OK De thuc hien.
    </p>
  </card>
  <card id = "card2" title = "String">
    <do type = "accept" label = "Home">
      <prev />
    </do>
    <p>
      $method1 <!-- hiện kết quả -->
    </p>
  </card>
</wml>
```

- Kết quả thực hiện:

Chương 24

Các đối tượng trong WMLScript



Hình 24.2. Xác định ký tự tại một vị trí trong một chuỗi.

Ví dụ 24.4.

Ví dụ minh họa phương thức so sánh chuỗi.

- File *compare.wmls*

```
// compare.wmls
// Minh họa phương thức compare
extern function compareVariables()
{
    var x1 = "hello";
    var output;
    if ( String.compare( x1, "hello" ) == 0 ) {
        output = "x1 = \"hello\"";
    }
    else {
```

Chương 24

Các đối tượng trong WMLScript

```
    output = "x1 != \"hello\"";
}
WMLBrowser.setVar( "compareVar1", output );
WMLBrowser.go( "#card2" );
}
extern function compareString()
{
    var x2 = "good bye";
    var output2;
    if ( x2 == "hello" ) {
        output2 = "x2 = \"hello\"";
    }
    else {
        output2 = "x2 != \"hello\"";
    }
    WMLBrowser.setVar( "compareVar2", output2 );
    WMLBrowser.go( "#card3" );
}
extern function compareCases()
{
    var x3 = "Happy Birthday";
    var x4 = "happy birthday";
    var output3;
    if ( String.compare( x3, x4 ) == 0 ) {
        output3 = "\"" + x3 + "\"" + " equals " +
            "\"" + x4 + "\"";
    }
}
```

```
else {
    output3 = "\"" + x3 + "\"" +
        " != " + "\"" + x4 + "\"";
}
WMLBrowser.setVar( "caseVar1", output3 );
WMLBrowser.go( "#card4" );
}
extern function greaterString()
{
    var string1 = "hello";
    var string2 = "good bye";
    var compare = "String." + "compare( x1, x2 ) = " +
        String.compare( string1, string2 );
    WMLBrowser.setVar( "result1", compare );
    WMLBrowser.go( "#card5" );
}
extern function equalString()
{
    var string1 = "hello";
    var compare2 = "String." + "compare( x1, x1 ) = " +
        String.compare( string1, string1 );
    WMLBrowser.setVar( "result2", compare2 );
    WMLBrowser.go( "#card6" );
}
extern function lessString()
{
    var string2 = "good bye";
```

Chương 24

Các đối tượng trong WMLScript

```
var string3 = "Happy Birthday";
var compare3 = "String.compare( x2, x3 ) = " +
    String.compare( string3, string2 );
WMLBrowser.setVar( "result3", compare3 );
WMLBrowser.go( "#card7" );
}
```

- File *compare.wml*

```
<!-- compare.wml -->
<--So sánh xâu -->
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
    "http://www.phone.com/dtd/wml11.dtd" >

<wml>
    <card id = "index" title = "Variables">
        <do type = "accept" label = "Run">
            <go href = "functionSet.wmls#compareVariables()" />
        </do>
        <p>
            x1 = "hello"<br />
            x2 = "good bye"<br />
            x3 = "Happy Birthday"<br />
            x4 = "happy birthday"
        </p>
    </card>
    <card id = "card2" title = "Compare Method">
        <do type = "accept" label = "OK">
```

Chương 24

Các đối tượng trong WMLScript

```
<go href = "functionSet.wmls#compareString()" />
</do>

<p>
  So sanh xau:
  ScompareVar1 <!--ket qua cua ham compareVariables -->
</p>
</card>
<card id = "card3" title = "String">
  <do type = "accept" label = "OK">
    <go href = "functionSet.wmls#compareCases()" />
  </do>
  <p>
    So sanh xau:<br />
    ScompareVar2 <!--ket qua cua ham compareString-->
  </p>
</card>
<card id = "card4" title = "Case">
  <do type = "accept" label = "OK">
    <go href = "functionSet.wmls#greaterString()" />
  </do>
  <p>
    So sanh xau:<br />
    ScaseVar1 <!--ket qua cua ham compareCases -->
  </p>
</card>
<card id = "card5" title = "Greater">
```

```
<do type = "accept" label = "OK">
  <go href = "functionSet.wmls#equalString()" />
</do>

<p>
  $result1 <!--ket qua cua ham greaterString -->
</p>
</card>
<card id = "card6" title = "Equal">
  <do type = "accept" label = "OK">
    <go href = "functionSet.wmls#lessString()" />
  </do>

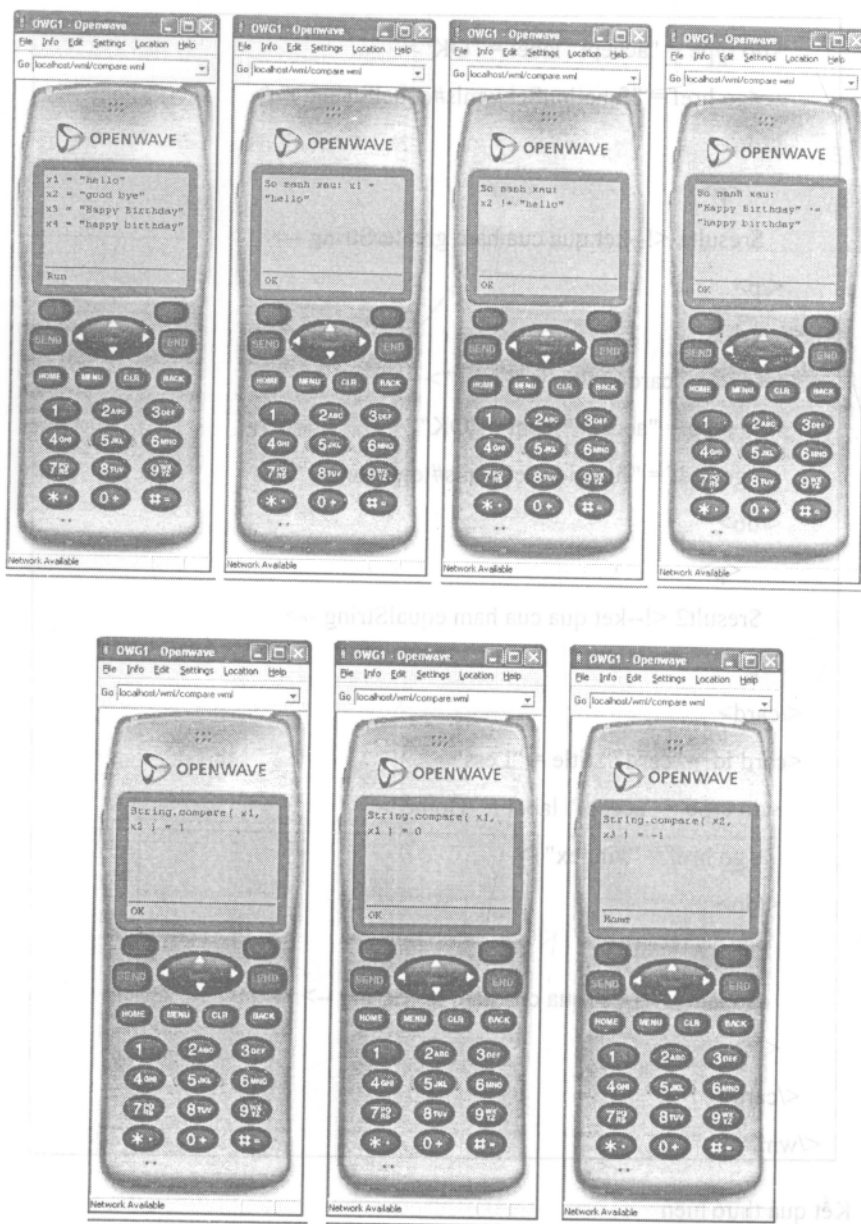
  <p>
    $result2 <!--ket qua cua ham equalString -->
  </p>
</card>
<card id = "card7" title = "Less">
  <do type = "accept" label = "Home">
    <go href = "#index" />
  </do>

  <p>
    $result3 <!--ket qua cua ham lessString -->
  </p>
</card>
</wml>
```

- Kết quả thực hiện

Chương 24

Các đối tượng trong WMLScript



Hình 24.3. Minh họa phương thức so sánh chuỗi.

Chương 24

Các đối tượng trong WMLScript

Ví dụ 24.5.

Minh hoạ phương thức *find*

- File *find.wmls*

```
// find.wmls
// Minh hoạ phương thức find
extern function findString()
{
    // Bảng chu cái
    var letters = "abcdefghijklmnopqrstuvwxyzabcdefghijklm";
    // Yêu cầu nhập vào 1 chu cái
    var string1 = Dialogs.prompt( "Nhập vào 1 xâu bất kỳ " +
        "để tìm từ trong bảng chu cái:", "" );
    // Gọi hàm tìm kí tự
    var variable1 = String.find( letters, string1 );
    WMLBrowser.setVar("result1", variable1 + 1);
    WMLBrowser.go("#card2");
}
```

- File *find.wml*

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
    "http://www.phone.com/dtd/wml11.dtd" >
<!-- find.wml -->
<!--Minh hoạ phương thức find -->
<wml>
```

Chương 24

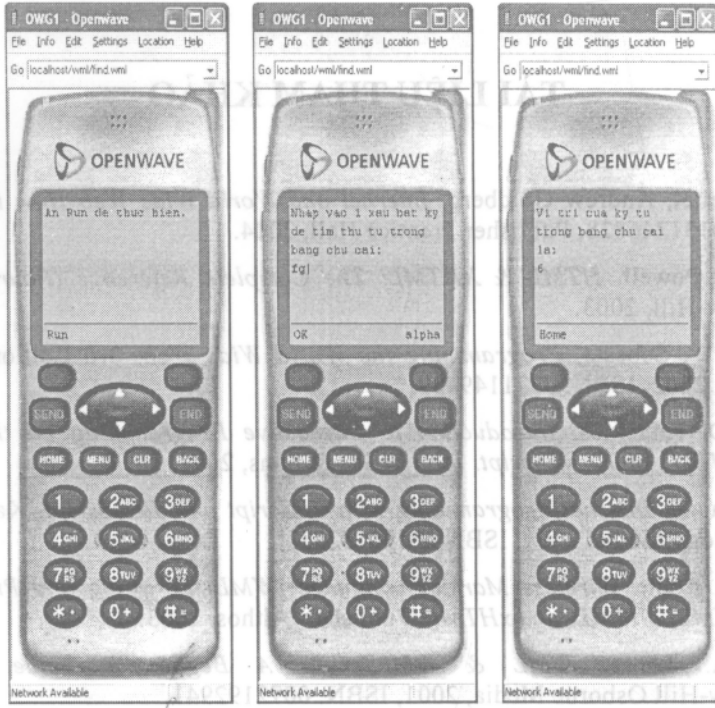
Các đối tượng trong WMLScript

```
<card id = "index" title = "String">
  <do type = "accept" label = "Run">
    <go href = "stringSearch.wmls#findString()" />
  </do>
  <p>
    An Run de thuc hien.
  </p>
</card>
<card id = "card2" title = "String">
  <do type = "accept" label = "Home">
    <prev />
  </do>
  <p>
    Vi tri cua ky tu trong bang chu cai la:<br />
    Sresult1 <!-- result of function findString -->
  </p>
</card>
</wml>
```

Chương 24

Các đối tượng trong WMLScript

▪ Kết quả thực hiện



Hình 24.4. Minh họa phương thức find.

24.4. Kết chương

Trong chương này chúng ta đã tìm hiểu về khái niệm đối tượng trong *WMLScript*, đối tượng *WMLBrowser*, đối tượng *String* trong *WMLScript*. Kết thúc phần giới thiệu về WML và *WMLScript* này chúng tôi hi vọng các bạn đã bắt đầu xây dựng được các ứng dụng của riêng mình. Chúc các bạn thành công.

TÀI LIỆU THAM KHẢO

1. Paul Deitel, Andrew Goldberg. *Internet and World Wide Web How to Program*. ISBN: 0131246828, Publisher Prentice Hall, 2004.
2. Thomas Powell. *HTML & XHTML: The Complete Reference (Fourth Edition)*. McGraw-Hill, 2003.
3. Robert W. Sebesta. *Programming the World Wide Web*. 3rd Edition, Addison-Wesley, 2002, ISBN: 0321149459.
4. Craig D. Knuckles. *Introduction to Interactive Programming on the Internet: Using HTML and JavaScript*. John Wiley & Sons, 2001.
5. Kate Kalata. *Internet Programming with VBScript and JavaScript*. Kalata Course Technology Media, 2001, ISBN 0619015233.
6. William Rott. *Wireless Markup Language (WML) Scripting and Programming using WML, cHTML, and xHTML*. Publisher: Althos, 2003.
7. Kris A. Jamsa. *WML & WMLScript: A Beginner's Guide*. Publisher: McGraw-Hill Osborne Media, 2001, ISBN: 0072192941.
8. John L. Cook. *WAP Servlets: Developing Dynamic Web Content With Java and WML (With CD-ROM)*. Publisher: John Wiley & Sons, 2000.
9. Steve McConnell. *Code Complete*. Second Edition, MicroSoft Press, 2004, ISBN: 1-55615-4844.
10. Harvey Deitel, Paul Deitel. *Simply Java Programming: An Application-Driven™ Tutorial Approach*. Prentice Hall, 2004, ISBN: 0131426486.

MỤC LỤC

Lời nói đầu	3
-------------------	---

Phần 4

JAVASCRIPT	5
-------------------------	---

Chương 8

Giới thiệu về JavaScript	6
---------------------------------------	---

8.1. Giới thiệu	6
8.2. Chương trình JavaScript đơn giản	9
8.3. Tương tác với người sử dụng	13
8.4. Những khái niệm cơ bản	17
8.4.1. Biến và hằng	18
8.4.2. Kiểu dữ liệu	19
8.5. Các toán tử	19
8.5.1. Toán tử số học	19
8.5.2. Toán tử gán	21
8.5.3. Toán tử so sánh	22
8.5.4. Toán tử điều kiện	22
8.5.5. Các toán tử khác	22
8.5.6. Thứ tự ưu tiên của các toán tử	24
8.6. Kết chương	24

Chương 9

Câu lệnh trong JavaScript	25
--	----

9.1. Giới thiệu	25
9.2. Giải thuật	26
9.3. Cấu trúc điều khiển	28
9.4. Câu lệnh điều kiện If	32
9.5. Câu lệnh điều kiện if...else	36
9.6. Câu lệnh lặp while	38

9.7. Toán tử gán.....	43
9.8. Toán tử tăng và giảm.....	45
9.9. Chú ý về kiểu dữ liệu	48
9.10. Kết chương.....	48

Chương 10

Câu lệnh trong JavaScript (tiếp theo)	49
--	----

10.1. Giới thiệu	49
10.2. Bản chất của vòng lặp điều khiển đếm	49
10.3. Câu lệnh lặp for	51
10.4. Những ví dụ sử dụng câu lệnh lặp for	54
10.5. Câu lệnh switch	57
10.6. Câu lệnh lặp do...while.....	60
10.7. Câu lệnh break.....	62
10.8. Câu lệnh continue.....	63
10.9. Sử dụng nhãn với câu lệnh break và continue	65
10.9.1. Lệnh break sử dụng nhãn	65
10.9.2. Lệnh continue sử dụng nhãn	67
10.10. Toán tử logical	68
10.11. Kết chương.....	72

Chương 11

Các hàm trong JavaScript	73
---------------------------------------	----

11.1. Giới thiệu.....	73
11.2. Modul chương trình trong JavaScript.....	73
11.2.1. Định nghĩa hàm	74
11.2.2. Gọi một hàm	75
11.3. Xây dựng một hàm do người dùng định nghĩa.....	75
11.3.1. Xây dựng hàm.....	75
11.3.2. Câu lệnh trả về Return.....	77

11.3.3. Các ví dụ minh họa.....	77
11.4. Tạo một số ngẫu nhiên	81
11.5. Phạm vi của biến	86
11.6. Các hàm có sẵn trong JavaScript.....	88
11.6.1. Hàm eval.....	89
11.6.2. Hàm isNaN	90
11.6.3. Hàm parseInt.....	91
11.6.4. Hàm parseFloat.....	92
11.7. Kết chương.....	93

Chương 12

Sử dụng mảng trong JavaScript. 94

12.1. Giới thiệu.....	94
12.2. Mảng trong JavaScript	96
12.3. Một số ví dụ sử dụng mảng	98
12.4. Sinh ra một ảnh ngẫu nhiên sử dụng mảng	105
12.5. Cách dùng tham số	107
12.5.1. Vai trò của tham số.....	107
12.5.2. Truyền theo trị	108
12.5.3. Truyền theo biến	108
12.6. Truyền mảng vào hàm.....	108
12.7. Mảng nhiều chiều	112
12.8. Các phương thức thao tác với mảng.....	115
12.9. Kết chương	116

Chương 13

Các đối tượng JavaScript. 117

13.1. Giới thiệu.....	117
13.2. Các đối tượng (Object).....	117
13.3. Các đối tượng của trình duyệt	119
13.3.1. Đối tượng Window.....	120
13.3.2. Đối tượng document	121
13.3.3. Đối tượng Frame	122

13.4. Các đối tượng của JavaScript	123
13.4.1. Đối tượng toán học (Math Object)	123
13.4.2. Đối tượng String	125
13.4.3. Đối tượng Date	132
13.4.4. Đối tượng Boolean và số (Number)	138
13.4.5. Đối tượng Document	139
13.4.6. Đối tượng Window	140
13.5. Sử dụng Cookie	146
13.5.1. Cookie là gì?	146
13.5.2. Làm việc với Cookies	147
13.6. Kết chương	157

Phần 5

DHTML	159
--------------------	-----

Chương 14

Giới thiệu về DHTML	160
----------------------------------	-----

14.1. Giới thiệu	160
14.2. Các đặc điểm của DHTML	161
14.2.1. Cấu trúc đối tượng	161
14.2.2. Các style động (Dynamic Styles)	163
14.2.3. Nội dung động	165
14.2.4. Định vị	165
14.2.5. Liên kết dữ liệu	168
14.2.6. Lập trình các Script	168
14.3. Sử dụng DHTML	169
14.4. Mô hình đối tượng tài liệu	169
14.4.1. Đối tượng Window	172
14.4.2. Đối tượng Document	174
14.4.3. Phần tử và đối tượng Frame	175
14.4.4. Đối tượng Navigator	178
14.4.5. All Collection	179
14.5. Kết chương	181

Chương 15
Mô hình sự kiện DHTML 182

15.1. Giới thiệu.....	182
15.2. Đối tượng sự kiện.....	182
15.2.1. Những sự kiện chuột (Mouse Events)	183
15.2.2. Các sự kiện focus và onblur	189
15.2.3. Những sự kiện khác	192
15.3. Xử lý sự kiện	194
15.3.1. Định nghĩa xử lý sự kiện.....	194
15.3.2. Nổi bọt sự kiện (Event bubbling)	195
15.3.3. Giá trị trả về.....	198
15.3.4. Kiểm soát lỗi với sự kiện onerror	200
15.4. Kết chương	202

Chương 16
Các hiệu ứng đặc biệt..... 203

16.1. Giới thiệu.....	203
16.2. Lọc (Filters).....	203
16.2.1. Thuộc tính lọc FlipV và FlipH.....	205
16.2.2. Làm trong suốt cùng với thuộc tính Chroma	207
16.2.3. Hiệu ứng cùng với thuộc tính lọc Mask	210
16.2.4. Kết hợp các thuộc tính lọc ảnh: invert, gray và xray	211
16.2.5. Thuộc tính lọc Shadow	214
16.2.6. Thuộc tính lọc Blur (bóng mờ)	217
16.2.7. Thuộc tính lọc trong suốt (alpha filter).....	218
16.2.8. Thuộc tính lọc Wave	222
16.3. Chuyển tiếp (Transitions).....	224
16.4. Kết chương.....	226

Chương 17
Liên kết dữ liệu 217

17.1. Giới thiệu.....	227
17.2. Sử dụng Tabular Data Control(TDC).....	228

17.2.1. File dữ liệu	228
17.2.2. Hiển thị dữ liệu	229
17.3. Kết nối tới bảng	234
17.4. Kết nối tới ảnh	236
17.5. Thao tác dữ liệu	239
17.5.1. Sắp xếp dữ liệu	239
17.5.2. Loại dữ liệu	242
17.6. Sử dụng các phần tử HTML khác	246
17.7. Kết chương	249

Phần 6

WIRELESS MARKUP LANGUAGE	251
---------------------------------------	------------

Chương 18

Wireless Markup Language (WML) Cơ bản	252
--	------------

18.1. Giới thiệu về WML	252
18.2. Ví dụ mở đầu	253
18.3. Các khái niệm mở đầu	255
18.3.1. Thẻ	255
18.3.1.1. Thẻ phần tử rỗng	255
18.3.1.2. Các phần tử	256
18.3.1.3. Thuộc tính	257
18.3.1.4. Ký hiệu cho các thuộc tính	257
18.3.2. Các thực thể	258
18.3.3. Viết chú thích	259
18.4. Biến WML và các ngữ cảnh	259
18.4.1. Thay thế biến	260
18.4.1.1. Biến rỗng	262
18.4.1.2. Quy định của biến thay thế	262
18.4.2. Thiết lập biến	263
18.4.2.1. Phần tử <setvar>	264
18.4.2.2. Thứ tự của thao tác trong việc thiết lập các biến	264
18.4.3. Các ngữ cảnh trình duyệt	265

18.5. Tác vụ và các sự kiện của WML	265
18.5.1. Các tác vụ	265
18.5.1.1. Tác vụ và các biến	266
18.5.1.2. Tác vụ <go>	266
18.5.1.3. Tác vụ <prev>	269
18.5.1.4. Tác vụ <refresh>	270
18.5.1.5. Tác vụ <noop>	272
18.5.2. Các sự kiện	272
18.5.2.1. Phần tử <onevent>	273
18.5.2.2. Phần tử card	273
18.5.2.3. Sự kiện onenterforward	274
18.5.2.4. Sự kiện onenterbackward	276
18.5.3. Tương tác người dùng WML	277
18.5.3.1. Các vấn đề tương tác Web (Web interaction) trong HTML	277
18.5.3.2. Tương tác trong WAP	278
18.5.3.3. Phần tử <input>	279
18.5.3.4. Phần tử <select>	281
18.5.3.5. Phần tử <option>	286
18.5.3.6. Phần tử <optgroup>	286
18.5.3.7. Phần tử <do>	287
18.5.3.8. Phần tử <anchor>	288
18.5.3.9. Phần tử <a>	288
18.5.3.10. Thuộc tính tabindex	289
18.6. Kết chương	290

Chương 19
WML cơ bản (tiếp theo) 175

19.1. Bộ định thời gian trong WML (WML Timer)	291
19.1.1. Khái niệm Timer trong WML	291
19.1.2. Phần tử <Timer>	293
19.2. Các phần tử của WML	296
19.2.1. Phần tử <wml>	296
19.2.2. Phần tử <head>	296

19.2.3. Phần từ <access>.....	296
19.2.4. Phần từ <meta>	297
19.2.5. Phần từ <card>.....	298
19.2.6. Phần từ <template>	298
19.3. Văn bản trong WML và định dạng văn bản	302
19.3.1. Phần từ <p>	303
19.3.2. Phần từ 	304
19.3.3. Định dạng văn bản	305
19.3.3.1. Làm nổi bật (Emphasis).....	305
19.3.3.2. Thay đổi kích thước	305
19.3.3.3. Đổi kiểu chữ	306
19.4. Sử dụng Bảng.....	306
19.4.1. Phần từ <table>	306
19.4.2. Phần từ <tr>	307
19.4.3. Phần từ <td>	307
19.5. Ảnh trong WML.....	308
19.6. Kết chương.....	310

Chương 20

Tổng quan về WMLScript..... 311

20.1. WMLScript là gì?.....	311
20.2. Cài đặt OpenwaveSDK 4.1.1	312
20.3. Từ vựng và các quy ước	314
20.4. Biểu diễn số và xâu kí tự	314
20.4.1. Biểu diễn dạng số nguyên.....	314
20.4.2. Biểu diễn dạng số dấu phẩy động.....	314
20.4.3. Biểu diễn xâu ký tự	315
20.4.4. Biểu diễn dạng Boolean.....	315
20.5. Biến	315
20.6. Kiểu dữ liệu.....	316
20.7. Từ khoá	318
20.8. Toán tử và biểu thức.....	319

20.8.1. Toán tử.....	319
20.8.2. Biểu thức.....	327
20.9. Kết chương.....	329

Chương 21

Hàm và Sử dụng hàm.....	230
--------------------------------	------------

21.1. Giới thiệu.....	330
21.2. Khai báo.....	330
21.3. Gọi hàm.....	332
21.4. Gọi hàm từ các đơn vị Script khác.....	333
21.5. Url pragma.....	333
21.6. Gọi WMLScript từ WML.....	334
21.7. Các thư viện chuẩn.....	336
21.7.1. Thư viện Lang Library.....	339
21.7.2. Thư viện Float Library.....	344
21.7.3. String Library.....	347
21.7.4. URL Library.....	354
21.7.5. WMLBrowser Library.....	358
21.7.6. Dialogs Library.....	360
21.8. Kết chương.....	360

Chương 22

Cấu trúc điều khiển - phần I.....	361
--	------------

22.1. Giới thiệu.....	361
22.2. Mã giả.....	361
22.3. Biểu thức và câu lệnh.....	361
22.4. Khối lệnh.....	362
22.5. Cấu trúc điều khiển.....	362
22.6. Cấu trúc lựa chọn if.....	363
22.7. Cấu trúc if/else.....	363

22.8. Cấu trúc while	371
22.9. Kết chương	375

Chương 23

Cấu trúc điều khiển - phần II.....	376
---	------------

23.1. Giới thiệu.....	376
23.2. Câu lệnh break và continue	382
23.3. Kết chương.....	389

Chương 24

Các đối tượng trong WMLScript.....	390
---	------------

24.1. Giới thiệu.....	390
24.2. Đối tượng WMLBrowser	391
24.3. Đối tượng String.....	393
24.4. Kết chương.....	405



ĐẠI HỌC

BÁCH KHOA

1956 - 2006

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

50 NĂM XÂY DỰNG VÀ PHÁT TRIỂN

CÁC BẠN TÌM ĐỌC CÁC SÁCH ĐÃ XUẤT BẢN:

- Huỳnh Quyết Thắng, Lê Tấn Hùng. **KỸ THUẬT ĐỒ HOẠ.**
Nhà xuất bản Khoa học và Kỹ thuật. 2004
- Lương Mạnh Bá, Nguyễn Thanh Thủy. **NHẬP MÔN XỬ LÝ ẢNH SỐ.**
Nhà xuất bản Khoa học và Kỹ thuật. 2003.
- Lê Đức Trung. **CÔNG NGHỆ PHẦN MỀM.**
Nhà xuất bản Khoa học và Kỹ thuật. 2005.
- Thạc Bình Cường. **PHÂN TÍCH THIẾT KẾ HỆ THỐNG THÔNG TIN.** Nhà xuất bản Khoa học và Kỹ thuật. 2004.
- Thạc Bình Cường. **QUẢN LÝ DỰ ÁN PHẦN MỀM.**
Nhà xuất bản Khoa học và Kỹ thuật. 2005.
- Lương Mạnh Bá. **TƯƠNG TÁC NGƯỜI MÁY.**
Nhà xuất bản Khoa học và Kỹ thuật. 2005.

lập trình ứng dụng 12 web



Giá: 60.000đ