BÀI 7: XỬ LÝ SỰ KIỆN VÀ BIỂU THỨC CHÍNH QUI

Mục tiêu bài học:

Kết thúc chương này, bạn có thể:

- Làm việc với các sự kiện phổ biến trong JavaScript
- Làm việc với Biểu thức chính qui

7.1 Đối tượng Event - Khái niệm

Các chương trình JavaScript thường là hướng sự kiện. Các sự kiện là các hành động xảy ra trên trang web. Một sự kiện có thể do người dùng tạo ra – click chuột vào một button - hoặc do hệ thống tạo ra – thay đổi kích thước của trang.

Hầu hết các trình duyệt đều hỗ trợ một đối tượng Event. Mỗi sự kiện có một đối tượng Event tương ứng. Đối tượng Event cung cấp thông tin về sự kiện - loại sự kiện và vị trí của con trỏ tại thời điểm xảy ra sự kiện. Khi một sự kiện được phát sinh, nó được gửi đi như một đối số đến trình xử lý sự kiện. Dĩ nhiên, phải có một trình xử lý sự kiện trong trường hợp này.

Chẳng hạn, khi người dùng nhấp chuột, sự kiện **onmousedown** được phát sinh. Đối tượng Event chứa những thông tin sau:

- ✓ Loại sự kiện - Trong trường hợp này là nhấp chuột
- ✓ Vị trí x và y của con trỏ khi nhấp chuột
- ✓ Nút chuột nào được nhấn
- ✓ Các phím bổ trợ (Control, Alt, Meta, hoặc Shift) được nhấn vào thời điểm xảy ra sự kiện.

Đối tượng Event không thể được sử dụng trực tiếp với đối tượng window. Nó được sử dụng như một phần của trình xử lý sự kiện.

Một sự kiện bắt đầu bằng hành động hoặc điều kiện khởi tạo sự kiện và kết thúc bằng việc đáp lại của trình xử lý sự kiện. Vòng đời của một sự kiện thông thường gồm những bước sau:

1. Hành động người dùng hoặc điều kiện tương ứng với sự kiện xảy ra.
2. Đối tượng Event được cập nhật tức thì nhằm phản ánh trạng thái của sự kiện.

3. Sự kiện được kích hoạt.
4. Trình xử lý sự kiện tương ứng được gọi.
5. Trình xử lý sự kiện thực hiện hành động của nó và trả về điều khiển cho chương trình.

7.2 Các sự kiện JavaScript

Tập hợp các sự kiện tương ứng với các phần tử khác nhau trên trang là một phần của mô hình đối tượng tài liệu (Document Object Model), chứ không phải của JavaScript. Nghĩa là, các sự kiện được một phần tử nào đó hỗ trợ có thể khác nhau trên các trình duyệt.

Sau đây là một số sự kiện thường được hầu hết các đối tượng hỗ trợ:

- **onClick**

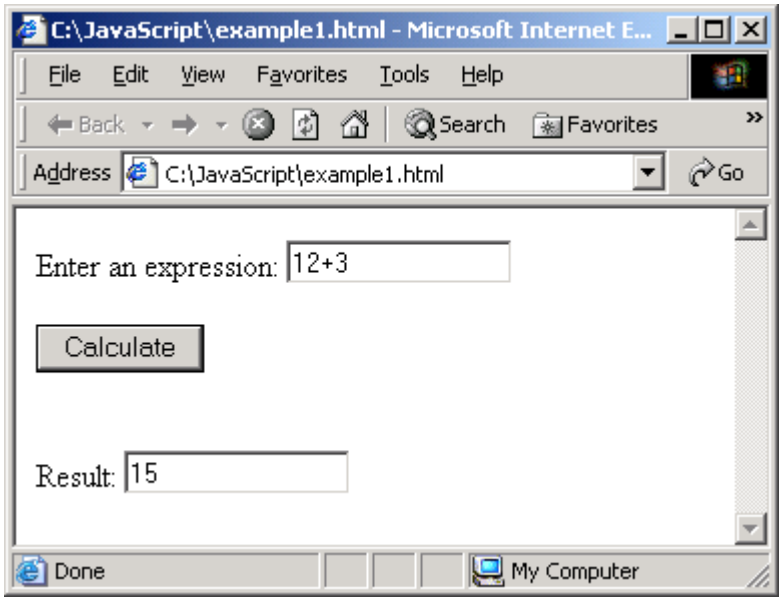
Sự kiện onClick được tạo ra bất cứ khi nào người dùng nhấp chuột lên các phần tử form nào đó (button, checkbox, radio button, và phần tử **select**), hoặc lên các hyperlink.

Ví dụ 1:

```
<HTML>
<HEAD>
 <SCRIPT LANGUAGE="JavaScript">
    function compute(form)
    {
            if (confirm("Are you sure?"))
             form.kết quả.value = eval(form.expr.value)
            else
             alert("Please come back again.")
    }
 </SCRIPT>
</HEAD>
```

```
<BODY>
 <FORM>
      Enter an expression:
      <INPUT TYPE="text" NAME="expr" SIZE=15 ><BR><BR>
      <INPUT TYPE="button" VALUE="Calculate"
      ONCLICK="compute(this.form)">
      <BR><BR><BR>
       Kết quả:
      <INPUT TYPE="text" NAME="kết quả" SIZE=15 >
      <BR>
  </FORM>
 </BODY>
 </HTML>
```

Kết quả đoạn mã trong ví dụ 1 được minh hoạ ở hình 7.1.



Hình 7.1: Kết quả của ví dụ 1

- **onChange**

  Sự kiện onChange xảy ra bất cứ khi nào một phần tử form thay đổi. Điều này có thể xảy ra khi nội dung của một trường văn bản thay đổi, hoặc khi một chọn lựa trong danh sách chọn lựa thay đổi. Tuy nhiên, sự kiện onChange không được tạo ra khi một radio button hoặc một checkbox được nhấp. Thay vào đó, sự kiện onClick sẽ được tạo ra.

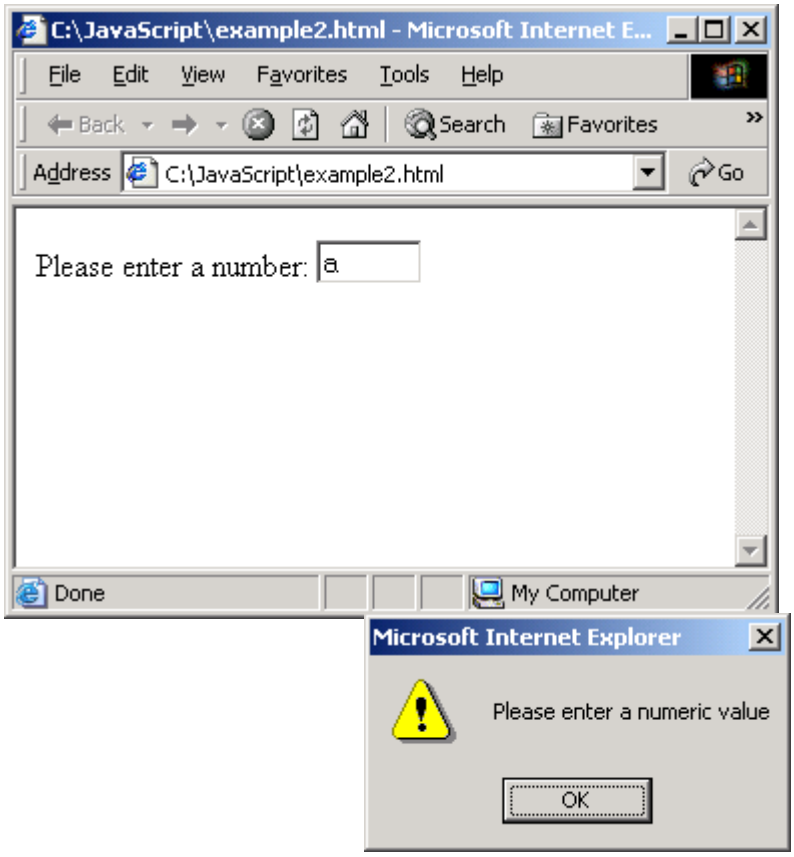
  Sự kiện onChange được gửi đi khi một phần tử hoàn tất việc thay đổi. Vì vậy, khi một textbox đang được hiệu chỉnh, sự kiện onChange chỉ được phát sinh sau khi việc hiệu chỉnh đã hoàn tất, và khi người dùng thoát khỏi textbox đó.

Ví dụ 2:

```
<HTML>
<HEAD>
   <SCRIPT LANGUAGE="JavaScript">
        <!--- hide script from old browsers
    function checkNum(num)
    {
        if (num == "")
        {
         alert("Please enter a number");
         return false;
         }
        if (isNaN (num))
        {
         alert("Please enter a numeric value");
         return false;
        }
```

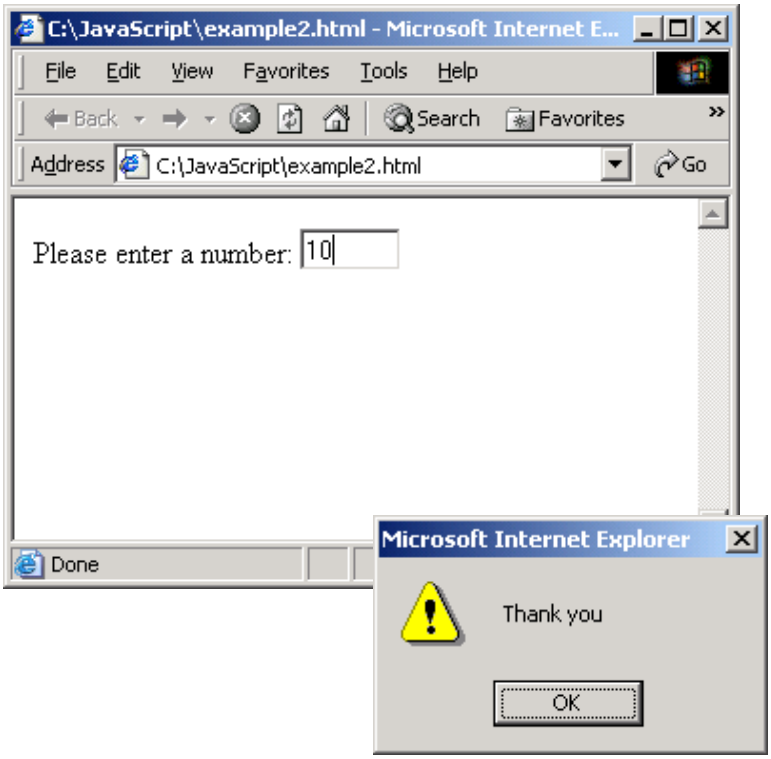
```
            else alert ("Thank you");
        }

    // end hiding from old browsers -->
  </SCRIPT>
</HEAD>
<BODY bgColor = white>
 <FORM>
    Please enter a number:
        <INPUT type = text size = 5 onChange="checkNum(this.value)">
 </FORM>
</BODY>
</HTML>
```

Hình 7.2(1) và 7.2(2) minh hoạ kết quả của đoạn mã trong ví dụ 2.



Hình 7.2: Kết quả của ví dụ 2(1)

Nếu chúng ta nhập vào một giá trị số:



Hình 7.2: Kết quả của ví dụ 2(2)

- **onFocus**
  Sự kiện onFocus được gửi đi bất cứ khi nào một phần tử form trở thành phần tử hiện thời. Chỉ khi một phần tử nhận được focus nó mới nhận được input từ người dùng. Điều này có thể xảy ra khi người dùng nhấp chuột lên phần tử, hoặc sử dụng phím Tab hoặc Shift+Tab (di chuyển tớI các phần tử trên form).

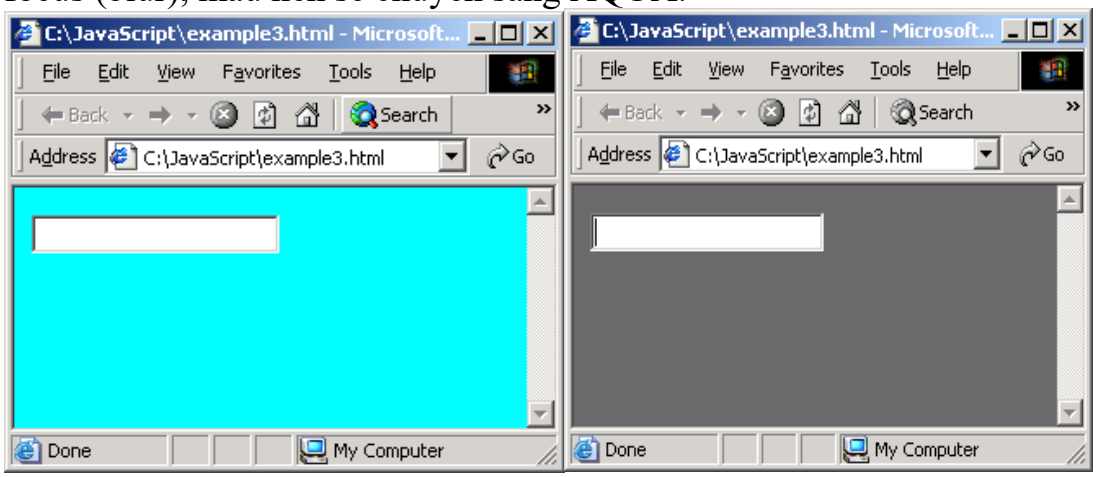
- **onBlur**

Blur ngược với focus. Khi người dùng rời khỏi một phần tử trên form, sự kiện onBlur được kích hoạt. Đối với một số phần tử, nếu nội dung của nó cũng bị thay đổi, thì sự kiện onChange cũng được kích hoạt.

Ví dụ 3:

```
<HTML>
 <BODY BGCOLOR="lavender">
    <FORM>
          <INPUT type = text name = text1
          onblur="(document.bgColor='aqua')"
          onfocus="(document.bgColor='dimgray')">
    </FORM>
 </BODY>
</HTML>
```

Khi textbox nhận được focus, màu nền sẽ chuyển sang DIMGRAY, khi mất focus (blur), màu nền sẽ chuyển sang AQUA.



Hình 7.3: Kết quả của ví dụ 3 – Blur (hình trái) and focus (hình phải)

- **onMouseOver**

Sự kiện onMouseOver được tạo ra bất cứ khi nào con trỏ chuột di chuyển lên trên một phần tử.

- **onMouseOut**

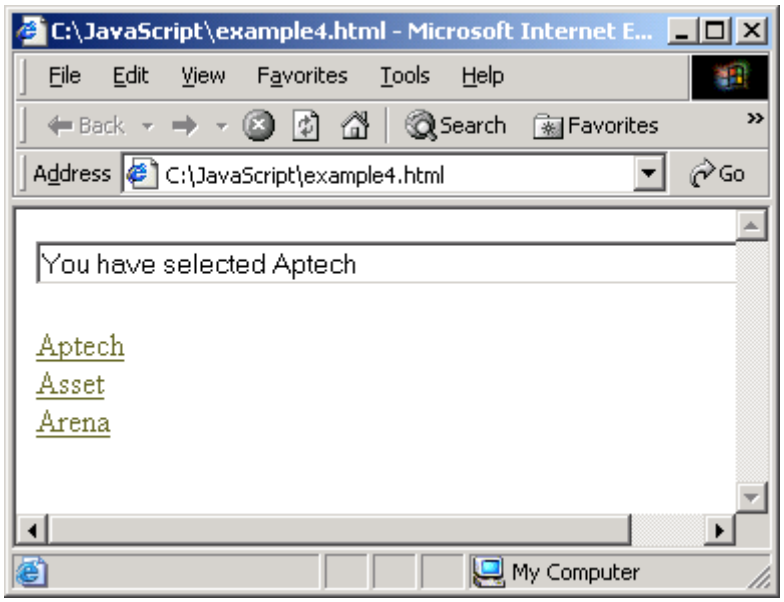
Sự kiện onMouseOut được tạo ra bất cứ khi nào con trỏ chuột di chuyển ra khỏi phần tử đó.

Ví dụ 4:

```
<html>
 <head>
    <script language = "javascript">
          var num =0
          function showLink(num)
          {
             if (num==1)
             {
                document.forms[0].elements[0].value= "You have selected Aptech";
       }
          if (num==2)
              {
                document.forms[0].elements[0].value = "You have selected Asset";
       }
          if (num==3)
              {
                document.forms[0].elements[0].value = "You have selected Arena";
       }
          }
```

```
    </script>
  </head>
  <body>
    <form>
        <input type=text size=60 >
    </form>
    <a href="#" onMouseOver="showLink(1)"document.bgcolor= “ green">Aptech</a><br>
    <a href="#" onMouseOver="showLink(2)">Asset</a><br>
    <a href="#" onMouseOver="showLink(3)">Arena</a><br>
  </body>
  </html>
```

Khi di chuyển chuột qua **Aptech**, kết quả được hiển thị như sau.



Hình 7.4: Kết quả của ví dụ 4

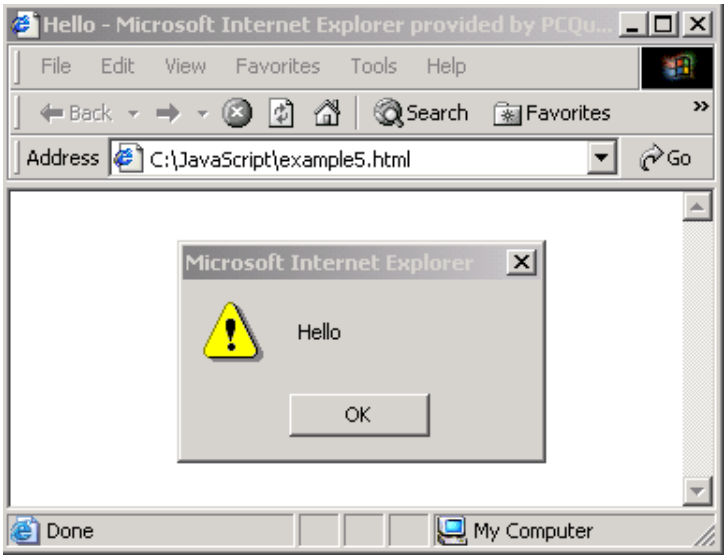
- **OnLoad**

Sự kiện onLoad (áp dụng cho đối tượng body) được phát sinh khi đã tải xong tài liệu. Nó cũng được phát sinh khi một ảnh đã tải xong.

Ví dụ 5:

```
<HTML>
<HEAD>
    <TITLE>Hello </TITLE>
 </HEAD>
 <BODY onLoad="alert('Hello')">
 </BODY>
 </HTML>
```

Kết quả:



Hình 7.5: Kết quả của ví dụ 5

- **onSubmit**

  Sự kiện onSubmit được tạo ra bất cứ khi nào người dùng truyền dữ liệu từ form đi (thường sử dụng nút Submit). Sự kiện xảy ra trước khi form thật sự được gửi đi. Thật ra, trình xử lý sự kiện tương ứng với sự kiện này có thể ngăn chặn

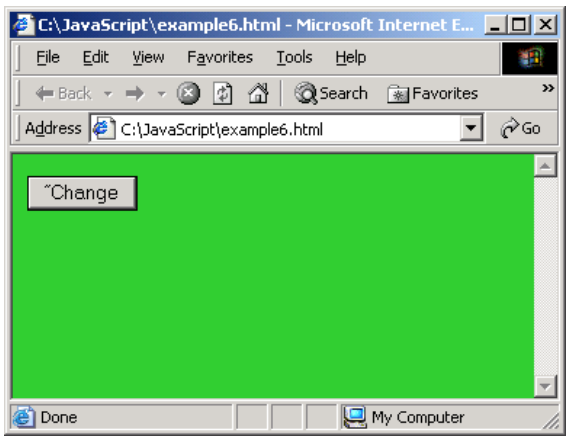
form không được gửi đi bằng cách trả về giá trị false. Cách này dùng đê kiểm tra sự hợp lệ của dữ liệu nhập vào.

- **onMouseDown**
  Sự kiện này được kích hoạt khi hành động nhấp chuột xảy ra. Nghĩa là, khi người dùng nhấp chuột. Đây là trình xử lý sự kiện cho các đối tượng button, document, và link.
- **onMouseUp**
  Sự kiện này được kích hoạt khi hành động nhả chuột xảy ra. Nghĩa là, khi người dùng thả chuột. Đây là trình xử lý sự kiện cho đối tượng button, document, và link.

Ví dụ 6:

```
<HTML>
<BODY BGCOLOR="lavender">
 <FORM>
    <INPUT type = button name = text1 value=”Change Color”
    onmousedown="(document.bgColor='aqua')"
    onmouseup="(document.bgColor='limegreen')"
 </FORM>
</BODY>
</HTML>
```

Hình dưới đây hiển thị kết quả của ví dụ 6



Hình 7.6: Kết quả của ví dụ 6

- **onResize**

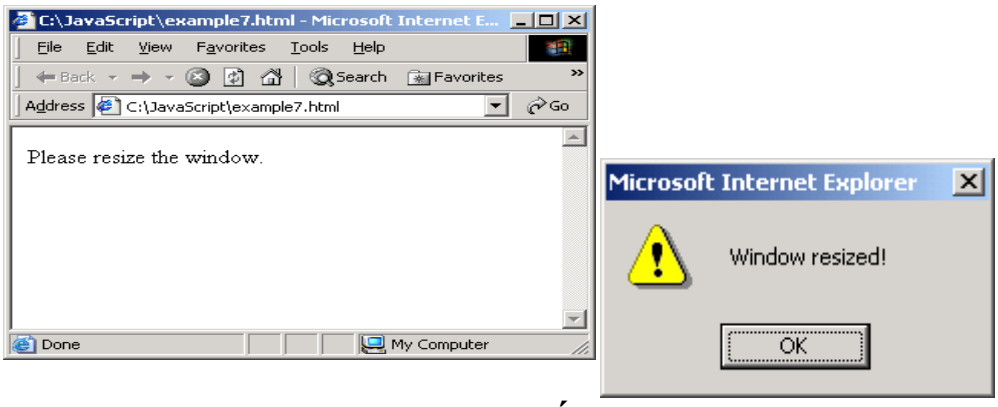
  Sự kiện này được kích hoạt khi hành động thay đổi kích thước cửa sổ xảy ra. Nghĩa là, khi người dùng hoặc một script làm thay đổi kích thước một cửa sổ hay frame. Đây là trình xử lý sự kiện cho các đối tượng Window.

Ví dụ 7

```
<html>
<head>
    <script language="JavaScript">
            window.onresize= notify;
            function notify()
            {
                    alert("Window resized!");
            }
    </script>
</head>
<body>
    Please resize the window.
</body>
```

</html>

Khi thay đổi kích thước cửa sổ, kết quả xuất hiện như sau:



Hình 7.7: Kết quả của ví dụ 7

7.3 Trình xử lý sự kiện

Khi một sự kiện được khởi tạo, chúng ta có thể tạo một đoạn mã JavaScript để đáp ứng lại sự kiện. Đoạn mã này được gọi là trình xử lý sự kiện. Trình xử lý sự kiện có thể là một câu lệnh đơn, một tập hợp các câu lệnh hoặc một hàm

```
<INPUT TYPE="button"
NAME="docode"
onClick="DoOnClick();">
```

Khi nhấp chuột vào một button, sự kiện onClick được khởi tạo. Sự kiện onClick gọi hàm DoOnClick và thực thi những câu lệnh bên trong hàm.

- **Trình xử lý sự kiện cho các thẻ HTML**
  Để khởi tạo trình xử lý sự kiện cho thẻ HTML, chúng ta phải chỉ định thẻ và thuộc tính trình xử lý sự kiện. Sau đó chúng ta gán mã JavaScript. Đoạn mã phải được đặt trong cặp dấu nháy kép.

```
<TAG eventHandler="JavaScript Code">
```

Các đối số chuỗi phải được đặt trong cặp dấu nháy đơn.

```
<INPUT TYPE="button" NAME="Button1" VALUE="Open Sesame!"
onClick="window.open('mydoc.html', 'newWin')">
```

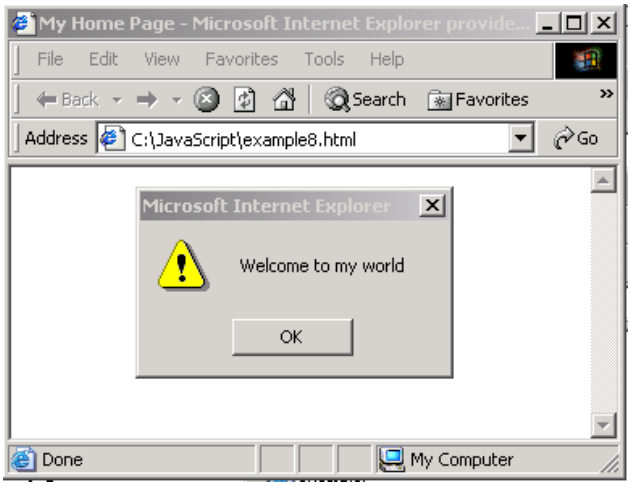
Thay vì sử dụng nhiều câu lệnh JavaScript, hàm sẽ giúp cho việc thiết kế chương trình tốt hơn. Chúng ta sẽ gọi hàm khi cần thiết. Hơn nữa các hàm đó có thể được dùng bởi các phần tử khác.

Câu lệnh này gán hàm greeting() cho trình xử lý sự kiện onLoad của window. Thuộc tính trình xử lý sự kiện được tham chiếu đến hàm greeting chứ không phảI lờI gọi đến hàm greeting()

Ví dụ 8

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

Kết quả:



Hình 7.8: Kết quả ví dụ 8

➢ **Trình xử lý sự kiện như là những thuộc tính**

Chúng ta cũng có thể gán một hàm cho một trình xử lý sự kiện của một đối tượng. Cú pháp như sau:

object.eventhandler = function;

Ví dụ,

window.onload = greeting;

Chúng ta xem lại ví dụ trên và sử dụng trình xử lý sự kiện như những thuộc tính:

Ví dụ 9

```
<HTML>
<HEAD>
 <TITLE>My Home Page</TITLE>
 <SCRIPT LANGUAGE="JavaScript">
```

```
<!--
  function greeting() {
    alert("Welcome to my world");
  }
   window.onload = greeting;


// -->
</SCRIPT>
</HEAD>
</HTML>
```

Kết quả sẽ tương tự như hình 7.8. Điểm mạnh của kĩ thuật này là tính linh hoạt. Chúng ta có thể thay đổI nhanh chóng các trình xử lý sự kiện khi được yêu cầu.

7.4 Biểu thức chính qui

Một biểu thức là một tập hợp hợp lệ gồm các literals, các biến và các toán tử để tính toán và trả về một giá trị đơn. Giá trị này có thể là một số, một chuỗi hay bất kỳ một giá trị logic nào đó.

- **Các biểu thức Chính qui**

  Biểu thức chính qui là một mẫu tìm kiếm dùng để tìm kiếm các chuỗi kí tự cùng dạng trong một xâu. Dùng Biểu thức chính qui ta có thể tìm kiếm theo mẫu trong các chuỗi kí tự do người dùng nhập vào. Ví dụ, ta tạo ra một mẫu tìm kiếm gồm từ « cat » và sẽ tìm kiếm tất cả các xuất hiện của từ này trong một xâu nào đó. Trong JavaScript, một Biểu thức chính qui cũng được xem là một đối tượng.

  Mẫu tìm kiếm của biểu thức quy tác có thể bao gồm:

  a. **Các mẫu đơn giản (Simple pattern).** Thực hiện việc tìm kiếm chính xác theo những kí tự được chỉ ra trong mẫu.Ví dụ, ta có một mẫu tìm kiếm như sau:
     /cat/

Nó sẽ tìm kiếm từ ‘cat’ trong một chuỗi. Nó sẽ trả về từ‘cat’ trong câu: “The cat sat on the mat” hoặc “Please hurry or we will not catch the train”.

Ký tự (/) chỉ ra vị trí bắt đầu và kết thúc của biểu thức cần tìm kiếm:

b. **Kết hợp các ký tự đơn giản và đặc biệt.** Được dùng khi ta muốn tìm kiếm các chuỗi cùng dang. Ví dụ, ta muốn tìm kiếm một chuỗi có một kí tự xuất hiện nhiều lần hay có nhiều kí tự trắng. Thông thường, dữ liệu do người dùng nhập vào thường có nhiều từ. Ví dụ, biểu thức /xy*z/ sẽ tìm kiếm các chuỗi bắt đầu bằng x, có thể có nhiều kí tự y hay không có bên trong nó, kết thúc bởi z.

Trong chuỗi xxzzzyzxyyyzz, biểu thức sẽ trả về chuỗi con ‘xyyyz’

Dưới đây là bảng liệt kê một số ký tự đặc biệt có thể sử dụng trong các Biểu thức chính qui .

Ký tự	**Nghĩa**	**Ví dụ**
\	Giải thoát nghĩa hằng (literal) của ký tự	/b/ tương xứng với ký tự 'b'. /\b/ có thể kết hợp với bất kỳ từ nào bắt đầu bằng với b. /a*/ tương xứng với 0 hoặc nhiều kí tự a. /a*/ tương xứng với 'a*'.
^	Tương xứng ký tự đầu chuỗi nhập hay đầu dòng	/^A/ sẽ tìm kiếm "An apple."
$	Tương xứng ký tự cuối chuỗi nhập hay cuối dòng	/t$/ sẽ tìm kiếm các ký tự t trong "eat" mà không phải “eater”.
*	Tương xứng ký tự phía trước là 0 hoặc nhiều lần	/o*ch/ tương xứng 'oooouch'
+	Tương xứng ký tự phía trước là 1 hoặc nhiều lần	/r+/ tương xứng ‘r’ trong “break” và tất cả ký tự r trong “brrrrrreak”

?	Tương xứng ký tự phía trước là 0 hoặc 1 lần.	/e?le?/ tương xứng le 'el' trong "angel" và 'le' trong "bangle."
.	(Dấu chấm thập phân) tương xứng bất kỳ ký tự đơn ngoại trừ ký tự newline	/.n/ tương xứng ‘no’ và 'on' trong “There are no apples on any of the trees” nhưng không phải là ‘any’.
(x)	Tương xứng 'x' và ghi nhớ ký tự tương xứng đó.	Ví dụ, /(boo)/ tương xứng 'boo' trong "boo said Casper" và ghi nhớ ‘boo’
x|y	Tương xứng 'x' hoặc 'y'.	Ví dụ, /green|ripe/ tương xứng là 'green' trong "green mangoes" và 'ripe' trong "ripe mangoes."
{n,m}	Tương xứng ít nhất n và nhiều nhất m số lần xuất hiện của kí tự phía trước. n và m là số nguyên dương.	Ví dụ, /o{1,3}/ không tương xứng trong"cld", ký tự 'o' trong "cold," hai ký tự oo đầu tiên trong "coold," và ba ký tự o đầu tiên trong "cooooold" Mặc dù có nhiều hơn ba ký tự o xuất hiện nhưng nó chỉ trả về ba ký tự xuất hiện đầu tiên.
[xyz]	Tương xứng bất kỳ nào nằm trong dấu ngoặc. Dấu nối được dùng để xác định một khoảng	[vxyz] tương tự như [v-yz] [0-9]
[^xyz]	Tương xứng với bất kỳ ký tự nào không nằm trong dấu ngoặc. Dấu nối được dùng để xác định một khoảng	[^xyz] tương tự như [^x-z].
\d	Tương xứng ký tự số	/ \d / sẽ tương xứng 7 trong chuỗi “This is Bond7”

\s	Tương xứng với ký tự trắng đơn bao gồm space, tab, form feed, line feed. Tương tự [\f\n\r\t\v].	/\s\w*/ tương xứng ' fun' trong "good fun."
\t	Tương xứng một tab	
\w	Tương xứng ký tự alphanumberic bao gồm ký tự gạch dưới. Tương tự [A-Za-z0-9].	/\w/ tương xứng 'm' trong "mangoes", '6' trong "62.50" và '2' trong "2D."

- **Tạo một biểu thức quy tắc**

Một Biểu thức chính qui là một mẫu tìm kiếm để tìm kiếm dữ liệu cùng dạng. JavaScript xem một Biểu thức chính qui như một đối tượng. Vì vậy, chúng ta phải tạo một Biểu thức chính qui trước khi sử dụng chúng. Chúng ta có thể tạo một Biểu thức chính qui bằng một trong hai cách sau:

1. **Sử dụng khởi tạo đối tượng (Object initializer).** Cách này trước đây được gọi là tạo đối tượng bằng cách sử dụng các kí hiệu nguyên dạng. Sau đó nó được chuyển thành object initializer để giống với thuật ngữ của C++. Nếu ta muốn tạo ra một thể hiện của một đối tượng, ta phải dùng một object initializer.
   Cú pháp của việc khởi tạo đối tượng:

   Objectname = {expression}
   Trong đó **objectname** là tên của đối tượng mới.

   Expression là một khuôn mẫu để tạo đối tượng.

   Ví dụ,

   re = /xy+z/

Trong định nghĩa trên, một Biểu thức chính qui "xy+z" được tạo và gán cho đối tượng re. Bây giờ, chúng ta có thể dùng đối tượng re để tìm kiếm các mẫu theo yêu cầu.

Khi chúng ta khởi tạo đối tượng, Biểu thức chính qui được dịch khi script được đánh giá. Nếu Biểu thức chính qui không thay đổi, sử dụng khởi tạo đối tượng thì hiệu quả.

2. **Gọi hàm khởi tạo của đối tượng RegExp.** JavaScript cung cấp đối tượng biểu thức định nghĩa trước đó là, RegExp. Một hàm khởi tạo được dùng để tạo một kiểu đối tượng và định nghĩa các thuộc tính của đối tượng. Ví dụ, chúng ta có thể tạo một đối tượng gọi là "employee". Các thuộc tính của đối tượng là empID, join_dt, salary.

```
function employee(empID, join_dt, salary)
{
  this.empID = empID
  this.joint_dt = join_dt
  this.salary = salary
 }
```

Sau khi hàm được tạo, chúng ta phải dùng hàm để tạo một thể hiện của đối tượng bằng toán tử new. Ví dụ:

```
employee1=new employee("100", "11/11/02", 3000)
```

Khi chúng ta dùng hàm khởi tạo, biểu thức được dịch trong thời gian thực thi. Nếu Biểu thức chính qui thay đổi hoặc nếu nó phụ thuộc vào dữ liệu nhập vào từ người dùng, sử dụng hàm khởi tạo là hợp lý nhất.

Ví dụ, hàm getdetails() tìm kiếm một mẫu dữ liệu được nhập từ người dùng. Dữ liệu nhập bởi dùng rất đa dạng. Chọn mẫu tìm kiếm là (\w+) \s (\d+). Có nghĩa là , một hoặc nhiều ký tự bất kỳ xuất hiện theo sau một ký tự trắng hoặc xuất hiện bất kỳ một ký tự số nào. Dấu cộng (+) chỉ ra một hoặc nhiều ký tự xuất hiện Dấu sao (*) chỉ tra 0 hoặc nhiều ký tự xuất hiện.

```
function getdetails()
{
```

```
        re = /(\w+)\s(\d+)/
        re.exec();
        window.alert(RegExp.$1 + ", your age is " + RegExp.$2);
    }
```

- **Sử dụng biểu thức chính qui**

Một khi chúng ta tạo một Biểu thức chính qui , chúng ta có thể đặt nó để dùng. Một số phương thức trong bảng dưới đây có thể dùng với Biểu thức chính qui .

Phương thức	**Mô tả**
Exec	Tìm kiếm một mẫu tương xứng trong một chuỗi. Nó trả về một mảng thông tin.
Test	Kiểm tra tương xứng trong một chuỗi. Trả về giá trị đúng hoặc sai
Match	Tìm kiếm tương xứng trong một chuỗi . Trả về một mảng thông tin hoặc giá trị null nếu sai.
Search	Kiểm tra sự tương xứng trong một chuỗi. Trả về giá trị chỉ số của tương xứng nếu tồn tại, -1 nếu bị sai
Replace	Tìm kiếm sự tương xứng trong một chuỗi, và thay thế chuỗi con tìm kiếm tương xứng bằng một chuỗi con thay thế khác.
Split	Dùng để tách một chuỗi thành một mảng các chuỗi con.

Để dùng một phương thức, chúng ta phải xác định đối tượng được sử dụng. Cú pháp là:

```
objectname.method = function_name
```

Sau đó chúng ta có thể gọi phương thức trong ngữ cảnh của đối tượng. Cú pháp là:

```
objectname.methodname(parameters)
```

Chúng ta có thể dùng các cờ với Biểu thức chính qui . Hai cờ “g” và “i” được chọn tuỳ ý. Rồi chúng ta có thể dùng riêng hoặc dùng cả hai cờ. Cờ “g” được dùng để chỉ dẫn tìm kiếm toàn cục. Cờ “i” dùng để chỉ dẫn tìm kiếm có phân biệt chữ hoa và chữ thường. Ví dụ,

```
re = /\w+\s/g;   //use a global search
```

Ví dụ 13:

Đoạn mã dưới đây dùng để kiểm tra phương thức tìm kiếm một mẫu trong chuỗi. Nếu mẫu được tìm thấy, giá trị trả về "True" và ngược lại thì trả về "false".

```
<HTML>
 <HEAD>
  <SCRIPT LANGUAGE="JavaScript">
  re = /Time/
  str = re.test ("Time and Tide wait for none");
  window.alert(str);
  </SCRIPT>
 </HEAD>
</HTML>
```

Kết quả:

Hình 13.14: JavaScript: Biểu thức

Ví dụ 14:

Đoạn mã sau đây tìm kiếm sự xuất hiện của ký tự x, y hoặc z.

```
<HTML>
 <HEAD>
```

```
<SCRIPT LANGUAGE="JavaScript">
re = /[xyz]/
str = re.exec("It is very cooooooold");
window.alert(str);
</SCRIPT>
</HEAD>
</HTML>
```

Kết quả:

Figure 13.15: JavaScript: Các biểu thức (1)

Tóm tắt bài học

- Sự kiện là các hành động xảy ra trên trang Web.
- Mỗi sự kiện được gắn với một đối tượng Event.
- Một sự kiện bắt đầu bằng hành động hoặc điều kiện khởi tạo sự kiện đó và kết thúc bằng sự đáp ứng lại từ trình xử lý sự kiện.
- Tập hợp các sự kiện tương ứng với các phần tử trang khác nhau là một phần trong mô hình đối tượng tài liệu (DOM) chứ không phải của JavaScript.
- Khi một sự kiện được kích hoạt, chúng ta chỉ ra một đoạn mã JavaScript sẽ được thực hiện để đáp ứng lại sự kiện. Phần mã lệnh này được gọi là trình xử lý sự kiện.

- Một biểu thức là một tập hợp lệ các nguyên dạng, các biến, và các toán tử và trả về một giá trị đơn.