JavaScript
Diseño de Sitios Web (TUW)
Desarrollo de Páginas Web (TUR)

Capítulo 1

Introducción a JavaScript.

JavaScript, al igual que Java o VRML, es una de las formas posibles a partir de las cuales es posible extender las capacidades del lenguaje script HTML. Este lenguaje de programación no es un lenguaje propiamente dicho. Es un lenguaje script u orientado a documento, tal y como pueden ser los lenguajes de macros que tienen muchos procesadores de texto. Entonces, JavaScript tiene como función la mejora de las páginas web a partir de la introducción de revisión de formularios, mensajes en la barra de estado, así como animaciones mediante la utilización de HTML dinámico.
Existen varias versiones de JavaScript. En este curso se utiliza la versión 1.0 creada para el navegador Netscape Navigator 2.0. El Microsoft Explorer soporta el JavaScript, sólo que su nombre cambia. La versión 3.0 interpreta el JScript, que es similar al JavaScript 1.0 pero con algunas diferencias que pueden provocar ciertaas incompatibilidades. El Explorer 4 parece que sí admite JavaScript 1.1 con cierta fiabilidad, y muchas funciones de la versión 1.2.
Para seguir este curso de JavaScript es necesario conocer el lenguaje script HTML, así como poseer ciertos conocimientos de programación básicos. En el caso de no haber utilizado nunca el lenguaje de programación HTML, puede visitar el curso de HTML que se ofrece dentro de nuestras páginas.

El lenguaje JavaScript

JavaScript es un lenguaje de scripts compacto basado en objetos (y no orientado a objetos). JavaScript permite la realización de aplicaciones de propósito general a través de la WWW y aunque no está diseñado para el desarrollo de grandes aplicaciones, es suficiente para la implementación de aplicaciones WWW completas o interfaces WWW hacia otras más complejas.
Por ejemplo, una aplicación escrita en JavaScript puede ser utilizada en un documento HTML proporcionando un mecanismo para el tratamiento de eventos, como un clic del ratón o bien la validación de entrada de datos en un formulario.
Sin que exista una comunicación a través de la red una página HTML con JavaScript incrustado puede interpretar, y alertar al usuario con una ventana de diálogo, de que las entradas de los formularios no son válidas. O bien realizar algún tipo de acción como ejecutar un fichero de sonido o un applet de Java.

JavaScript y Java

Las diferencias entre Java y JavaScript son importantes, pero también comparten una serie de similitudes que se deben tener en consideración para su desarrollo en la programación.
JavaScript es un lenguaje de programación, mientras que JavaScript es un lenguaje tipo script. Este último es más sencillo de entender y usar que Java si no se tienen conocimientos previos de metodología de programación orientada a objetos. Por ello, resulta más adecuado el estudio y comprensión de este último como paso previo a un lenguaje orientado a objetos como Java.
JavaScript es mucho más modesto, pero precisamente por ello es más simple. Se basa en un modelo de instancia de objetos muy simple para el que no es necesario tener conocimientos de conceptos como herencia y polimorfismo.
Soporta un sistema en tiempo de ejecución basado en un pequeño número de tipo de datos (number, boolean y string) en el que no es necesario declarar el tipo de variables. A diferencia de esto, Java exige una gran rigidez en el tipo de datos utilizados y dispone de una amplia variedad de tipos básicos predefinidos, operadores y estructuras de control.

En Java uno de los principales bloques de programación son las clases a las que se asocian funciones específicas. Para utilizarlas es necesario instanciarlas en objetos. Los requerimientos de Java para declarar dichas clases, diseñar sus funciones, y encapsular tipos hacen que la programación en este lenguaje sea mucho más compleja que la realizada con JavaScript.
Otra diferencia bastante importante es que Java es un lenguaje bastante potente para desarrollar aplicaciones en cualquier ámbito. No es un lenguaje para programar en Internet, sinó que se trata de un lenguaje de propósito general, con el cual se puede escribir un applet para una página Web hasta una aplicación que no tenga ninguna clase de conexión a Internet.
Los requerimientos también son diferentes. Para programar en JavaScript sólo es necesario un editor de texto mientras que para programar en Java es necesario un compilador específico.
La complejidad de Java es semejante a la de un programa en C++ mientras que la de JavaScript es equivalente a la de un programa en dBase, Clipper o sh.
Por otra parte, la sintaxis de ambos lenguajes es muy similar sobre todo en lo que estructuras a control de flujo se refiere. Entonces, existen mecanismos de comunicación entre Java y JavaScript.
A continuación se presenta una tabla en la que se recogen las principales similitudes y diferencias entre los dos lenguajes.

JavaScript	**Java**
Interpretado en cliente	Compilado en el servidor antes de la ejecución en el cliente
Basado en objetos	Programación orientada a objetos
Código integrado en el HTML	Applets diferenciados del código
No es necesario declarar el tipo de variable	Necesario declarar los tipos
Enlazado dinámico. Los objetos referenciados deben existir en tiempo de ejecución (lenguaje interpretado)	Enlazado estático. Los objetos referenciados deben existir en tiempo de compilación (lenguaje compilado)

JavaScript y CGI

CGI (Common Gateaway Interfaz) es una interfaz entre programas de aplicación y servicios de información.Es decir, son un conjunto de reglas a cumplir tanto por parte del servidor como por parte del programa, pero se deja libertad al programador a la hora de escoger el lenguaje que considere más adecuado para programar la aplicación. Un programa CGI puede ser escrito en cualquier lenguaje como:C/C++, Fortran, PERL, TCL, etc.
En JavaScript no existen restricciones a cumplir en el Servidor hasta el punto que ni siquiera es necesario que éste exista.
Por otra parte y al contrario que CGI, JavaScript únicamente depende del cliente y no del sistema operativo, sólo es necesario un browser capaz de interpretarlo. Cualquier persona puede desarrollar aplicaciones escritas en JavaScript del mismo modo que realiza páginas HTML. Esto no ocurre con aplicaciones CGI que necesitan la existencia de un servidor WWW para ser ejecutadas.
Con JavaScript todo el código es trasladado al cliente y no se necesita la comunicación a través de la red cada vez que se produce un evento, como se requería en CGI.
A continuación se expone una tabla comparativa entre JavaScript y CGI en la cual se representan las características más importantes de cada uno de estos.

JavaScript	**CGI**
Es un lenguaje de programación	Es una interfaz. Da libertad de elección del lenguaje
No requiere un servidor WWW	Exige la presencia de un servidor WWW
No requiere una red, funciona en local	Requiere comunicación en red. No funciona en local
La aplicación reside en el cliente	La aplicación reside en el servidor
Requiere un cliente especial	Sirve cualquier cliente, por simple que

	sea
Pensado para aplicaciones únicas	Permite el desarrollo de aplicaciones distrinuidas, acceso concurrente y/o compartido
Tiempo de desarrollo breve	Tiempo de desarrollo medio

Primeros pasos en JavaScript.

En esta sección se tratarán las cuestiones estructurales del lenguaje de programación JavaScript, así como cada una de las acciones que se pueden llevar a cabo con este tipo de lenguaje. De esta forma, se estudiarán aspectos relacionados con la forma de utilización de acciones y métodos de JavaScript, así como la realización de un primer guión de un script.

Carga de un documento JavaScript.

Los documentos se cargan en los navegadores compatibles con JavaScript de la misma manera que lo hacen en otros navegadores. Un navegador JavaScript se comporta de la misma manera a como lo hace cualquier otro documento HTML. El navegador deberá encontrar el código JavaScript, situado el mismo de arriba a bajo en el documento de la misma manera que el código HTML. Algunas veces, JavaScript indica al navegador el situar texto o otros elementos en la páginas, de la misma manera como lo hacen las instrucciones tradicionales HTML. Otras veces, el código prepara al navegador para responder a las entradas del usuario y a las acciones que pueden venir más tarde vía script.
Esta parte consiste en el establecimiento de tablas de configuración para uso anticipado para futuros usuarios.
Una vez que los documentos se han cargado, entonces todas las partes HTML y JavaScript han sido leídas por el navegador. Salvo para los elementos de interfaz de usuario diseñado para interacciones y actualizaciones dinámicas, toda la página queda fijada en la memoria del navegador y no puede ser alterada desde el script.
Cualquier cosa que acontezca después será iniciada por la persona que usa la página. La próxima acción puede ser llevada a cabo por JavaScript, siempre que se haya diseñado la página en este sentido, o puede ser llevada a cabo por simple HTML, tal como saltar a otro URL una vez el usuario a tecleado en un enlace.

Ejecución de un script

Un script puede ejecutarse cuando el documento se carga o bien cuando el usuario interactúa en formularios en un documento. Puede haber algunas veces en las que tenga estas dos maneras dinámicas de trabajar en un documento: una parte del documento ayuda a formatear parte de la página; en otra parte responde a las acciones del usuario. Por esto, es necesario ver como cada tipo de acción está definida en un documento HTML. Para esto se describirán varios programas con cada tipo de acción a tener en consideración.

<u>Script inmediato</u>

Se utiliza el término de script inmediato para indicar las líneas de JavaScript que no sólo se ejecutan cuando el navegador carga el documento, sino que además influyen en la forma de la página. Tales scripts deben ser situados en los documentos HTML si la salida de los script reproduce los contenidos necesarios de la página.
El bloque de script debe indicar donde está por que contribuye al contenido de esa proporción de página. En este caso, y siempre que se incluya JavaScript en un documento HTML, se debe incluir esta porción del código entre las etiquetas <SCRIPT>...</SCRIPT>. Estas etiquetas alertan al navegador de que se debe empezar a interpretar todo el texto entre ellas como un script. Debe especificarse el nombre del tipo de lenguaje script en las etiquetas de la forma: <SCRIPT LANGUAGE="JavaScript">. Por tanto, cuando los navegadores reciban esta señal indicando que el script está utilizando JavaScript, se iniciará el intérprete de JavaScript para manejar el código.
En el listado siguiente se presenta un bloque de script para mostrar el contenido como parte del

cuerpo pero antes de los elementos del formulario.

Ejemplo:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
   //script that produces content for the body
</SCRIPT>
<FORM>
   <INPUT TYPE="text">
   <INPUT TYPE="button">
</FORM>
</BODY>
</HTML>
```

Script posterior

Un script posterior es visto por los navegador cuando el documento se carga, pero la sintaxis del script dice al navegador con el código nada más que estar pendiente de que existe. Estas secciones de script consisten en pequeños grupos de líneas de código que procesan información de alguna manera.
La localización recomendada para los segmentos de script diferidos es en el bloque de cabecera, según se define en el código de programa siguiente. Las secciones script pueden coexistir felizmente con los otros bloques de definición de cabezera que normalmente se situa en un documento HTML, tales como los títulos de los documentos o las especificaciones BASE FONT.
A continuación se define la zona del script donde se define la respuesta a las acciones del usuario.

Ejemplo:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
   //scripts that initializes items for user-driven
   //actions
</SCRIPT>
</HEAD>
<BODY>
<FORM>
   <INPUT TYPE="text">
   <INPUT TYPE="button">
</FORM>
</BODY>
</HTML>
```

La primera razón para la utilización de un scrpit posterior en la cabecera es que se carga en la memoria del navegador primero, incluso antes de cualquier contenido visible que aparezca en la página. Por tanto, si los usuarios interrumpen la carga de la página después de que un botón aparece en la ventana, el navegador conoce como reaccionar a la acción del botón ya que el script posterior ya está cargado. Si el script posterior se hubiera escrito al final del bloque del cuerpo, una selección prematura del botón por el usuario el navegador no sabría que ejecutar ya que el script estaría parcialmente cargado.

Script mixto

A partir de lo visto hasta ahora, se tiene que las líneas de script inmediatas ayudan a crear el

contenido de las páginas; las posteriores reaccionan a las acciones de usuarios una vez que la página está totalmente cargada. En muchos ejemplos se verá como un script posterior en el bloque del cuerpo llama a un script posterior en la cabecera. A continuación se presenta un ejemplo en el que se define un documento en que se utilizan los dos tipos de script, tanto el inmediato, definido en el cuerpo del documento, como el posterior, definido en la cabecera.

Ejemplo:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
   //scripts that initializes items for user-driven
   //actions
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
   //script that produces content for the body
</SCRIPT>
<FORM>
   <INPUT TYPE="text">
   <INPUT TYPE="button">
</FORM>
</BODY>
</HTML>
```

Acciones de usuario

Las acciones de usuarios son conocidas en JavaScript como eventos. El navegador vigila los elementos del documento diseñados para ser capaz de reaccionar a las acciones de usuario. Entonces, el programador debe indicar que elementos de interfaz de usuario deben responder a las acciones de usuario y que elementos deberían dar respuesta a una acción particular. Cada tipo de elemento responde a un limitado grupo de eventos. A continuación se presenta una lista de objetos comunes del interfaz de usuario JavaScript y los eventos a los que responden.

Elemento	Acción	Nombre del evento
Botón	Clic	Click
Casilla	Clic	Click
Enlace	Clic	Click
	Situar el puntero	MouseOver
Botón de opción	Clic	ClicK
Campo de texto	Tabulador/Clic	Focus
	Tabulador/Clic para cambio	Blur
	de texto y clic	Change

Para que un elemento de interfaz de usuario responda a un evento, debe tener además un atributo en su definición: el event handler, o manejador de eventos. Un manejador de eventos contiene instrucciones acerca de qué hacer cuando una clase particular de eventos alcanza un elemento de la ventana. Un nombre del atributo de manejador de eventos es el nombre de un evento precedido por la palabra "on"; la otra parte del atributo define qué acción se realiza. Así, se puede definir un botón que responde a un clic del ratón de la forma siguiente.

Ejemplo:

```
<INPUT TYPE="button" NAME="oneButton" VALUE="Press Me!" onClick="alert('Ok')">
```

Como puede verse, se ha añadido el manejador de eventos onClicK=. El valor del atributo consiste en las instrucciones JavaScript que se quieren ejecutar en respuesta al ratón o el nombre de un script posterior.

Manejar acciones de un script posterior

Un script posterior que se llama a partir de un script inmediato tantas veces como sea necesario para ejecutar las acciones de usuario se denomina función. La definición de un script posterior como función puede utilizarse para escribir un menor número de líneas de código, de manera que una acción se realiza tantas veces como sea necesario a partir de la llama a ese script.
De esta forma, se puede modificar el botón de alerta anterior de manera que se utilice el concepto de función, no incluyendo el comando de alerta en el botón de definición.

```
function alertUser(){
   alert('Ok')
}
```

Una vez que esta parte del documento se carga en el navegador, este sabe lo que debe hacer cuando una entidad en el documento pide que se ejecute la función alertUser(). Esta función debe definirse entre las etiquetas <SCRIPT>...</SCRIPT> de la cabecera del documento.
Abajo, en la definición del botón, el atributo del manejador de eventos onClick requiere o líneas de script o el nombre de la función en la memoria de navegador. En este caso sse especifica el nombre de la función.

```
<INPUT TYPE="button" NAME="oneButton" VALUE="Press Me!" onClick="alertUser">
```

Especificando un nombre de función en el atributo onClick=, se indica al navegador que función en concreto se ejecuta cada vez que el usuario teclea un botón.
Entonces, se puede reescribir toda la función, obteniendo el resultado siguiente.

Ejemplo:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function alertUser(){
   alert('Ok')
</SCRIPT>
</HEAD>
<BODY>
<FORM>
   <INPUT TYPE="text">
   <INPUT TYPE="button" NAME="oneButton" VALUE="Press Me!" onClick="alertUser()">
</FORM>
</BODY>
</HTML>
```

Se podría pensar el porqué de utilizar algo tan largo y complicado para construir y llamar a una función que maneja un clic de usuario. En la programación, las funciones son usualmente múltiples líneas de código que hacen las definiciones de los elementos del documento HTML largas y muy difíciles de depurar. Teniendo el código de los script posteriores convenientemente localizada en la parte de la cabecera del documento, hará que sea más fácil la búsqueda de cualquier parte que deba ser cambiada o reparada. Un script HTML grande podría tener más de una docena de definiciones de funciones en la sección de la cabecera, cada una de las cuales realiza una específica tarea para uno o más de un objeto del cuerpo del documento. Si estas funciones se situaran en el documento, podría tardar tiempo en encontrar cualquiera de ellas que necesite ajustarse.

Capítulo 2

Valores, Variables y Literales

En este capítulo se describirán los diferentes valores que JavaScript reconoce, así como también una descripción de los bloques fundamentales de trabajo de JavaScript: las variables y los literales.
Este capítulo contiene las secciones siguientes:

- Valores
- Variables
- Literales

Valores

El lenguaje de programación JavaScript reconoce los siguientes tipos de valores:

- Valores Numéricos.
- Valores lógicos.
- Valores de tipo cadena.
- Valor nulo (null).
- Valores indefinidos (undefined).

Conversión de tipos

El lenguaje de programación JavaScript es un lenguaje que realiza una conversión dinámica de los tipos; esto es, no es necesario especificar en el código fuente del programa el tipo de datos de una variable cuando esta es declarada, de manera que los tipos son convertidos de forma automática durante la ejecución del programa. De esta forma es posible asignar a una variable un valor determinado:
var variable=56
y posteriormente asignarle a esta misma variable una cadena de caracteres:
variable="como va viejo"
Entonces, debido a que JavaScript es un lenguaje de conversión dinámica de tipos, no existirá error alguno en la sentencia.
En expresiones en las que intervienen valores de tipo numérico y cadena con el operador +, JavaScript convierte los valores numéricos en cadenas de caracteres. Por ejemplo, consideremos las siguientes sentencias:
x="como va"+42 // retorna "como va 42"
y=42+"como va" // retorna "42 como va"
En las sentencias siguientes, JavaScript no realiza la conversión de valores numéricos en carácteres:
"37"-7 // retorna 30
"37"+7 // retorna 37

Variables

Las variables se pueden definir como un nombre simbólico al cual le asignamos un determinado valor en un programa.
Existen una serie de reglas que deben de cumplirse en el caso de la asignación de nombres a las variables. El identificador de variable debe empezar por una letra o bien por un caracter de subrayado (_). Algunos ejemplos de nombres válidos son los siguientes:
Número_uno, puntero_A y _variable.

Declaración de variables

La declaración de variables en JavaScript puede realizarse de dos formas diferentes:

- Mediante una simple asignación de valores. Por ejemplo: x=10.
- Mediante la sentencia var. Por ejemplo: var x=10.

Evaluación de variables

Una variable o un vector al que no le han sido asignados un valor posee un valor indefinido. El resultado de evaluar una variable indefinida depende de como haya sido declarada. Existen dos posibilidades:

- Si la variable fue declarada sin la expresión var, la evaluación provoca un error en tiempo de ejecución.
- Si la variable fue declarada a partir de la expresión var, la evaluación retorna un valor indefinido, o bien un valor de NaN si se trabaja con datos de tipo numérico.

El siguiente código demuestra la evaluación de variables indefinidas:

```
function myfuncion(){
  return y-2;
}
myfuncion()//causa un error en tiempo de ejecución.

function myfuncion2(){
  return var y-2;
}
myfuncion2()//retorna NaN
```

Es posible utilizar la sentencia undefined para determinar cuando una variable posee un determinado valor. En el siguiente código, la variable de entrada no posee un valor asignado, de manera que la estructura if se evalua como verdadero, ejecutándose la instrucción:

```
var input;
if (input===undefined){
  doThis();
}else{
  doThat();
}
```

Un valor indefinido se evalua como falso cuando se evalua como un valor booleano. Por ejemplo, el siguiente código ejecuta la función myfuncion3 debido a que el array no está definido.

```
myArray=new Array()
if (!myArray[0])
  myFuncion()
```

En el caso de evaluar una variable nula (null), el valor se define como 0 en un contexto numérico y como falso si tratamos con valores de tipo booleano. Por ejemplo

```
var n=null
n*32 //retorna 0.
```

Ámbito de las variables

En JavaScript, así como en la mayoría de lenguajes de programación, es posible encontrar dos tipos de variables: variables globales y variables locales. Las variables globales son aquellas que se definen fuera de cualquier función del programa, de manera que es posible acceder a ellas desde cualquier parte del código fuente. A diferencia de estas, las variables locales son aquellas que se definen dentro de una función, de manera que sólo es posible referirse a ellas dentro de la función en la cual han sido declaradas.
En la declaración de variables es opcional la utilización de la expresión var, aunque es

necesario su utilización si esta variable se define localmente.
Es posible acceder a una variable global declarada en una ventana o frame desde otra ventana a partir de la especificación del nombre de la ventana o frame.

Literales

Los literales son utilizados para representar valores fijos en JavaScript. En esta sección se definen los siguientes literales.

- Literales tipo Vector (Array)
- Literales tipo Booleano (Boolean)
- Literales tipo Coma Flotante (Floating-Point)
- Literales tipo Entero (Integer)
- Literales tipo Objeto (Object)
- Literales tipo Cadena (String)

Literales tipo Array

Un literal tipo Array es una lista de cero o más expresiones, cada una de las cuales representa un elemento del array. Cuando se crea un array de tipo literal, se inicializa con los valores especificados para cada uno de los elementos, de la misma forma que su longitud depende del número de elementos.
El siguiente ejemplo crea un array de números con tres elementos y longitud de tres:

number=["one","two","three"]

Literales tipo Booleano

Un dato tipo booleano posee dos valores literales: verdadero (true) y falso (false).

Literales tipo Floating-Point

Un literal de coma flotante puede poseer las siguientes partes:

- Un entero decimal
- Un punto decimal
- Una fracción
- Un exponente

La parte exponencial se define como "e" o "E" seguido de un entero, el cual puede tener signo (precedido de "+" ó "-").Un literal de coma flotante debe poseer almenos un dígito y un punto decimal o "e" (ó "E"). Algunos ejemplos de literales de coma flotante son: 3.1415, -3.1E23, 1e27 y 1E-67.

Literales tipo Entero

En el lenguaje de programación JavaScript, los enteros pueden ser expresados en decimal (base 10), hexadecimal (base 16) y octal (base 8).Un literal decimal entero consiste en una secuencia de dígitos sin un encabezamiento de la secuencia con 0. En el caso que la secuencia comience con 0 indica que la secuencia se corresponde con un valor octal, mientras que si comienza con 0x implica un valor hexadecimal (0X).
Ejemplos de literales enteros son: 42, 0xFFF y -555.

Literales de tipo Objeto

Los literales de tipo Objeto son una lista de cero o más pares de propiedades de nombres y valores asociados de un objeto, encerrados entre llaves ({}).
Un ejemplo de objetos literales se expone a continuación. El primer elemento del literal Libro

define una propiedad, mylibro; el segundo elemento, editorial, llama a la función BookTypes("McGrawHill"); el tercer elemento, NuevaEditorial, utiliza una variable definida en el código fuente, (Venta).

```
var Venta="AddissonWesley";
function BookTypes(nombre){
        if (nombre=="McGrawHill")
            return nombre;
        else{
            return "No trabajamos con la editorial "+ nombre+".";
}
Libro={mylibro:"Arquitectura y redes de computadores", editorial:BookTypes("McGrawHill"),
Nuevaeditorial:Venta}
document.write(Libro.mylibro);//retorna Arquitectura y redes de computadores
document.write(Libro.editorial);//retorna McGrawHill
document.write(Libro.Nuevaeditorial);//retorna AddisonWesley
```

De forma adicional, es posible realizar un indexado del objeto a partir de la propiedad index, de manera que se puede acceder a cada propiedad a partir de su índice asociado. El siguiente ejemplo utiliza esta opción:

```
Libro={mylibro:{a:"McGrawHill",b:"AddissonWesley"}, c:"Paraninfo"}
document.write(Libro.mylibro.a); //McGrawHill
document.write(Libro[c]); //Paraninfo
```

Literales tipo Cadena

Un literal tipo cadena es 0 o más carácteres encerrados en dobles comillas(") o simples comillas('). Una cadena debe estar delimitada por unas marcas de acotación del mismo tipo, ya sean comillas simples o dobles. Los siguientes ejemplos son literales de tipo cadena:

- "Como va"
- 'Como va'
- "567"
- "Una línea \n otra línea"

Utilización de caracteres especiales.

A parte de los caracteres ordinarios, es posible incluir caracteres especiales en las cadenas, como se muestra en el siguiente ejemplo:
"Una línea \n otra línea"
En la siguiente tabla se muestran los caracteres especiales que puedes utilizar en las cadenas de caracteres de JavaScript.

Carácter	Significado
\b	Espacio en blanco
\f	
\n	Nueva línea
\r	Retorno de carro
\t	Tabulador
\'	Apóstrofe o comillas simples
\"	Dobles comillas
\\	Contrabarra
\XXX	
\xXX	
\uXXXX	

- Carácteres de escape

  Para carácteres no listados en la tabla anterior, una contrabarra se ignora, con excepción de una marca de acotación. Es posible insertar una marca de acotación dentro de una cadena si ésta es precedida de contrabarra. Por ejemplo, si se supone un código fuente de la forma siguiente:
  var acotación= "El estudia con el libro \"Arquitectura y Estructura de Computadores\" de W.Stallings."
  document.write(acotación)
  El resultado que se visualizaría en la pantalla sería el siguiente:
  El estudia con el libro "Arquitectura y Estructura de Computadores" de W. Stallings.
  En el caso que se desee incluir una contrabarra en una cadena de caracteres, es necesario utilizar una doble contrabarra. Por ejemplo, en el caso de asignar el path c:\temp a una cadena de caracteres, se usa la sentencia siguiente:
  var home="c:\\temp"

Capítulo 3

Expresiones y Operadores

Este capítulo describe las expresiones y operadores lógicos utilizados en el lenguaje de programación JavaScript. De la misma forma, también se incluyen operadores de asignación, comparación, matemáticos, lógicos, cadena y operadores especiales. Este capítulo contiene las siguientes secciones:

- Expresiones
- Operadores

Expresiones

Una expresión es un conjunto de literales, variables, operadores y expresiones que evaluan un único valor. El valor puede ser un número, una cadena o un valor lógico.
Conceptualmente, existen dos tipos de expresiones: aquellas que asignan un valor a una variable y las expresiones que tan sólo tienen un valor. Por ejemplo, la expresión x=7 asigna a la variable "x" el valor "7". Por otro lado, la expresión (3+4) retorna, simplemente, un valor de 7, sin que se realice ninguna asignación. JavaScript posee los siguientes tipos de expresiones:

- Expresión de tipo Aritmético: evalua un número 3.14159.
- Expresión de tipo Cadena: evalua una cadena de caracteres, por ejemplo, "Fred" o "234".
- Expresión de tipo Lógico: evalua una expresión de tipo lógico retornando un valor de verdadero o falso.

Operadores

El lenguaje de programación JavaScript poseelos siguientes tipos de operadores. Esta sección describe los operadores y contiene información hacerca de su orden de precedencia.

- Operadores de Asignación
- Operadores de Comparación
- Operadores Aritméticos
- Operadores de Bits
- Operadores Lógicos
- Operadores de Cadena

- Operadores Especiales

Operadores de Asignación

Un operador de asignación asigna un valor situado a su derecha al operando situado a su izquierda. El operador básico de asignación es igual (=). Por ejemplo, en la expresión x=y, se tiene que el valor de y es asignado a la variable x.
Los otros operadores de asignación son utilizados para operaciones básicas, como se muestra en la tabla siguiente.

Operador	Significado
x+=y	x=x+y
x-=y	x=x-y
x*=y	x=x*y
x/=y	x=x/y
x%=y	x=x%y
x<<=y	x=x<< font>
x&=y	x=x&y
x^=y	x=x^y
x|=y	x=x|y

Operadores de Comparación

Este tipo de operadores compara los operandos y retorna el valor lógico basado en la comparación de los dos elementos. Los operandos pueden ser valores numéricos o cadenas de caracteres. La siguiente tabla describe los operadores de comparación.

Operador	Descripción	Ejemplo
Igual (==)	Retorna el valor de verdadero si los dos operandos son iguales. Si los dos operandos no son del mismo tipo, JavaScript realiza la conversión de los operandos en el tipo apropiado para realizar la comparación.	3=='3'
No igual (!=)	Retorna el valor verdadero si los operandos no son iguales. En el caso que los dos operandos no sean del mismo tipo, JavaScript realiza la conversión de los operandos en el tipo apropiado para realizar la comparación.	3===4
Estrictamente Igual (===)	Retorna un valor verdadero si los operandos son iguales y del mismo tipo	3===3
Estrictamente no Igual (!==)	Retorna un valor de verdadero si el operando y/o no son del mismo tipo	3!==3
Mayor que (>)	Retorna verdadero si el operando de la izquierda es mayor que el operando de la derecha.	3>2
Mayor o igual que (>=	Retorna verdadero si el valor de la izquierda es mayor o igual que el operando de la derecha.	2>=2

Operadores Aritméticos

Un operador aritmético toma los valores numéricos (tanto literales como variables) como sus operandos, de manera que devuelve un único valor numérico. Las operaciones estándard son adición (+), sustracción (-), multiplicación (*) y división (/). Estos operadores trabajan de la misma forma que en otros lenguajes de programación, exceptuando el operador (/), que retorna un valor no truncado, como pueda ser en Java o C++.Así, por ejemplo, se tiene que:

1/2 //returns 0.5 en JavaScript.
1/2 //returns 0 en Java.

En adición, JavaScript utiliza los operadores aritméticos listados en la tabla siguiente:

Operador	Descripción	Ejemplo
% (Modulus)	Operador Binario. Retorna el entero resultante de la división de dos operandos.	12%5 returns 2
++ (Increment)	Operador Unario. Añade uno a su operando. Si se usa como prefijo, retorna el valor de su operando después de la adición.	Si x es 3, entonces ++x pone x a 4 y retorna 4. Si es x++ pone x a 4 y retorna 3.
-- (Decrement)	Operador Unario. Sustrae uno a su operando. EL valor retornado es análogo que para el operador de incremento.	Si x es 3, entonces --x pone x a 2 y retorna 2. Si es x++ pone x a 2 y retorna 3
-(Negación unaria)	Operador Unario. Retorna la negación de su operador.	Si x es 3, entonces -x retorna -3

Operadores de Bits.

Los operadores de bits tratan sus operandos como un valor en bits (ceros y unos), ya sea un número decimal, hexadecimal u octal. Por ejemplo, el número decimal 9 posee una representación 1001 en binario. Entonces, los operadores de bits realizan sus operaciones en representación binaria, retornando un valor numérico en el estándar de JavaScript. A continuación se da una tabla en la que se recogen los operadores de bits de JavaScript.

Operador	Utilización	Descripción
AND	a & b	Retorna un uno en cada posición de bit si y sólo si hay un uno en la misma posición en los dos operandos
OR	a | b	Retorna un uno en cada posición de bit si y sólo si hay un uno en una de las mismas posiciones de los dos operandos
XOR	a ^ b	Retorna un uno en cada posición de bit si y sólo si en uno de los operandos hay un uno en esa posición
NOT	a	Invierte los bits del operando
Left shift	a << b	Desplazamiento a la derecha de los bits un número de posiciones b
Right shift	a >> b	Desplazamiento a la izquierda de los bits un número de posiciones b
Zero-fill right shift	a >>> b	Desplazamiento a la derecha con ceros un número de posiciones b

Operadores Lógicos.

Los operadores lógicos son utilizados típicamente con valores booleanos (valores logicos), de manera que retornan un valor booleano. Sin embargo, los operadores && y || retornan el valor de uno de los operandos específicos, de manera que si estos operadores son utilizados con valores no booleanos, retornarán un valor no booleano. Los operadores lógicos son descritos en la siguiente tabla.

Operador	Utilización	Descripción
&&	expr1 & expr2	(AND lógico) Si se utiliza con valores booleanos, retorna true si ambos son verdaderos, si no retorna falso

||	expr1 || expr2	(OR lógico) Si se utilizan valores booleanos, retorna el valor verdadero si uno de los operandos es verdadero, si los dos son falsos retorna el valor de false
!	!expr	(NOT lógico) Retorna el valor de false si el operando es verdadero; en el caso que sea falso retorna true

A continuación se exponen los siguientes ejemplos respecto a la utilización de este tipo de operandos:

AND lógico:
a1=true && true //t && t returns true
a2=true && false //t && f returns false
a3=false && true //f && t returns false
a4=false && (3 == 4) //f && f returns false
a5="Cat" && "Dog" //t && t returns Dog
a6=false && "Cat" //f && t returns false

OR lógico:
o1=true||true //t || t returns true
o2=false || true //f || t returns true
o3=true || false //t || f returns true
o4=false || (3==4) //f || f returns false
o5="Cat" || "Dog" //t || t returns Cat
o6=false || "Cat" //f || t returns Cat
o7="Cat" || false //t || f returns Cat

NOT lógico:
n1=!true //!t returns false
n2=!false //!f returns true
n3=!"Cat" //!t returns false

Operadores de Cadena

Al igual que los operadores de comparación, que pueden ser utilizados con valores tipo cadena, el operador de concatenación (+) concatena dos valores tipo cadena, retornando una cadena que es la unión de los dos operandos anteriores. Así si se supone que se tienen dos operandos tipo cadena, la operación de concatenación se realizaría de la forma siguiente:

"my" + "string" //returns "my string"

Operadores Especiales

JavaScript posee los siguientes operadores especiales:

- Operador Condicional
- Operador Coma
- delete
- new
- this
- typeof
- void

Operador Condicional

El operador condicional es el único operador JavaScript que trabaja con tres operandos. Este operador puede tener uno de los dos valores basado sobre una condición. La sintaxis es la siguiente:

```
condition ? val1 : val2
```

Si la condición es verdadera, el operador posee el valor de val1. Si no posee el valor del operador val2. Es posible utilizar el operador condicional en cualquier lugar donde se utiliza un operador de forma habitual. Así, por ejemplo se tiene que:

```
status = (age >= 18)? "adult" : "minor"
```

Esta sentencia asigna el valor de "adult" a la variable status si age es 18 o mayor. En otro caso, asigna el valor de "minor" a status.

Operador Coma

El operador coma (,) simplemente evalua ambos operandos y retorna el valor del segundo operando. Este operador es usado dentro de un bucle for, para permitir una actualización de múltiples variables cada vez que se ejecuta el bucle.
Por ejemplo, si a es un array bidimensional (matriz) con 10 elementos, el siguiente código usa el operador coma para incrementar dos variables cada vez. El siguiente código se utiliza para acceder a los elementos situados en la diagonal del array bidimensional.

```
for (var i=0, j=9; i<=9; i++, j--)     document.writeln("a ["+i+","+j+"]="+a[i,j])
```

Operador delete

El operador delete es utilizado para eliminar un objeto, una propiedad de este objeto, o un elemento en un array indexado. La sintaxis para este operador se define como:

```
delete.objectName
delete.objectName.property
delete.objectName[index]
delete property // legal only within a with statement
```

donde objectName es el nombre del objeto, property es el nombre de una propiedad existente, y index es un entero representando la localización del elemento en el array.
Es posible la utilización del operador delete con variables declaradas de forma implícita, pero no declaradas con la sentencia var.
El operdor delete retorna un valor true si la operación de eliminación es posible; y retorna false si esta operación no es posible.

Operador new

El operador new es utilizado para la creación de objetos en JavaScript. En capítulos posteriores se tratará a fondo la utilización y sintaxis de este operador.

Operador this

El operador this se utiliza para referirse a un objeto. En general, this se utiliza en la llamada a un objeto en un método. La sintaxis utilizada es la siguiente:

```
this.[.propertyName]
```

Por ser su utilización habitual en los script, a continuación se muestran ejemplos hacerca de la utilización de este operador.

Ejemplo:

Se supone una función denominada validate que evalua la propiedad de un objeto; dando los valores bajos y altos de un objeto.

```
function validate(obj, lowval, hival){
   if ((obj.value < lowval)||(obj.value>hival))
      alert("Invalid Value!")
}
```

En este caso, es posible llamar a la función validate a partir del manejador de eventos onChange=, de manera que se utiliza el operador this para referenciar el objeto.

```
Enter a number between 18 and 99:
<INPUT TYPE="text NAME="age" SIZE=3 onChange="validate(this, 18, 99)">
```

Operador typeof

El operador typeof puede ser utilizado de las dos formas siguientes:

```
typeof operand
typeof(operand)
```

El operador typeof retorna una cadena indicando el tipo de operando. El paréntesis es opcional. Para mostrar la utilización del operando typeof, se supone el ejemplo siguiente, en el que se definen las siguientes variables:

```
var myFun = new function("5+2")
var shape = "round"
var size = 1
var today = new Date()
```

El operador typeof retorna el siguiente resultado para estas variables:

```
typeof myFun is object
typeof shape is string
typeof size is number
typeof today is object
typeof dontExist is undefined
```

Para los valores true y null, el oparador typeof retorna el resultado siguiente:

```
typeof true is boolean
typeof null is object
```

Para un tipo número o cadena,el operador typeof retorna los resultados siguientes:

```
typeof 62 is number
typeof 'Hello world' is string
```

Operador void

El operador void puede utilizarse de la forma siguiente:

```
void(expression)
void expression
```

El operador void especifica una expresión que debe ser evaluada sin retorno de ningún valor, donde expression es cualquier expresión en JavaScript que debe ser evaluada. La utilización del paréntesis es opcional. Es posible la utilización del operador void para especificar una expresión como un enlace de hipertexto. La expresión se evalua a 0, pero no tiene ningún efecto en el documento.
A continuación se presenta un enlace en el que se crea un enlace de hipertexto que no hace

nada cuando el usuario hace clic con el ratón sobre él.

Ejemplo:

<A HREF="JavaScript:void(0)">Click here to do nothing</A>
EL siguiente código crea un enlace de hipertexto que presenta un atributo form cuando el usuario hace clic con el ratón sobre él.

<A HREF="JavaScript:void(document.form.submit())">ClicK here to submit</A>

Orden de Precedencia

La precedencia de operadores determina el orden en que deben ser aplicados para evaluar una expresión. La siguiente tabla describe la precedencia de operadores, de menor a mayor precedencia.

<table>
<tr><th>Tipo de operador</th><th>Operadores Individuales</th></tr>
<tr><td>coma</td><td>,</td></tr>
<tr><td>asignación</td><td>= += -= *= /= %= <<= >>= >>>= &= ^= |=</td></tr>
<tr><td>condicional</td><td>?:</td></tr>
<tr><td>or lógico</td><td>||</td></tr>
<tr><td>and lógico</td><td>&&</td></tr>
<tr><td>or-entre bits</td><td>|</td></tr>
<tr><td>xor-entre bits</td><td>^</td></tr>
<tr><td>and-entre bits</td><td>&</td></tr>
<tr><td>igualdad</td><td>== !=</td></tr>
<tr><td>relacional</td><td>< <= > >=</td></tr>
<tr><td>rotación de bits</td><td><< >> >>></td></tr>
<tr><td>adición/sustracción</td><td>+ -</td></tr>
<tr><td>multiplicación/división</td><td>* / %</td></tr>
<tr><td>negación/incremento</td><td>! - + ++ -- typeof void delete</td></tr>
<tr><td>call</td><td>()</td></tr>
<tr><td>crear objeto</td><td>new</td></tr>
<tr><td>miembro</td><td>. []</td></tr>
</table>

Capítulo 4

Programación Orientada a Objetos

En este capítulo no se realizará un estudio en profundidad de la Programación Orientada a Objetos (OOP). No hay que ver este tipo de programación como algo muy complejo para lo que se necesite contar con una experiencia previa. Todo lo contrario, cuanta menos experiencia se tenga en programación tradicional (Basic, Pascal, Fortran, C), más rápidamente se podrán captar los beneficios de esta técnica, ya que el conocer las técnicas de programación procedural restringen a la hora de saber cómo actuar con datos y acciones.
Además, JavaScript simplifica la programación, por lo que hay que conocer pocas cosas referentes a OOP, ya que las técnicas avanzadas en OOP incluso no tienen cabida en JavaScript. Por todo esto, esta parte del curso se centrará en lo que se refiere a programación en JavaScript.

Qué es un Objeto?

Una variable es una expresión en la que se le asigna un valor a un nombre para que este valor este representado en un programa. En realidad, una variable es simplemente un modo de

referencia a una parte de la memoria de un ordenador de tal manera que sea más entendible. Las instrucciones en los script podrían coger este valor de la memoria, realizar un cálculo y situarlo de nuevo en esa parte. Si el valor necesita más espacio después del cálculo, será problema del intérprete del lenguaje el buscar sitio en la memoria libre donde situar estos nuevos datos de manera rápida con respecto al usuario.
Entonces, se puede decir que un objeto es como una variable en algunos aspectos. Cuando un script define un objeto el navegador elige una parte en la memoria. Pero es más complejo internamente que un único valor. Su objetivo es representar una parte de la página a la que pertenece. Debido a que con JavaScript se trabaja con elementos que aparecen en ventanas, un objeto puede ser un campo de entrada, un botón o el propio documento HTML en total.
Para los usuarios de un programa que es construido usando un lenguaje de orientación a objetos, no hay forma de conocer la forma de construcción del objeto.
La programación orientada a objetos se ha empezado a utilizar en esta época de entorno gráficos de usuarios (GUI) porque, a diferencia de las pantallas de entrada de carácter, los entornos de ventanas de gráficos tales como Windows, Macintosh o X-Windows empujan a los desarrolladores a replicar en la ventana los objetos del mundo real.

Propiedades

Cada una de las características que posee un objeto dentro del sistema de la programación orientada a objetos se denomina propiedad. Cada propiedad posee un valor de algún tipo unido a él, incluso si este valor es nulo o vacío.
El valor de un objeto siempre se puede obtener en un script. Ahora bien, no todas las propiedades se pueden modificar desde un script. A continuación se expone el codigo fuente de un ejemplo que permitirá explicar esta forma de operar de las propiedades de JavaScript.

```
<HTML>
<HEAD>
<TITLE>Properties </TITLE>
</HEAD>
<H1>Let's Script...</H1>
<BODY>
<HR>
<SCRIPT LANGUAGE="JavaScript">
   document.write("Last update on " + document.lastModified + ".")
</SCRIPT>
</BODY>
</HTML>
```

En este ejemplo, el script recupera y escribe en pantalla la propiedad de un documento HTML que contiene la fecha y la hora del fichero según la última modificación del mismo. Esta fecha está impuesta en el fichero por el sistema operativo. Por motivos de seguridad, JavaScript no permite que se pueda escribir directamente en los privilegios de cualquier fichero del servidor. Por tanto, su script no puede modificar el valor de la propiedad lastModified de un documento.
Un script puede, sin embargo, modificar el valor de la propiedad de un campo de texto en un documento. Así, después de que el script registre la información que se ha escrito en los campos de texto y en la selección de botones en una parte del documento, se realizan las acciones necesarias que se hayan diseñado para transformar esta información y mostrar los resultados de la misma en algún campo. El acto de poner el texto en los campos es el resultado de actuar en las propiedades del objeto adecuado y el responsable de mostrarla.
Muchos de los objetos con los que se trabaja en JavaScriptestán predefinidos por el lenguaje. La mayor parte de los mismos representan las partes con las que se diseñan las ventanas que aparecen en los documentos. Todas las propiedades están claramente definidas, siendo parte del trabajo de los script el manipular los valores de las propiedades.
JavaScript permite, además, crear sus propios objetos. En capítulos posteriores se discutirá la creación de objetos, así como la utilización de los objetos predefinidos por JavaScript.

Los Métodos

El atributo más importante de un objeto es el que permite saber cómo se hacen las cosas. Esto es, un objeto tiene como parte de su constitución los pasos necesarios a seguir para llevar a cabo tareas específicas. Entonces, lo que es necesario conocer hacerca de un objeto es lo que permite realizar y que parámetros son necesarios para realizarlas.
Cada una de estas tareas son los métodos. Un método está relacionado con lo que en otros lenguajes de programación se denomina comando, ya que en ambos casos se produce una acción.
Se puede pensar en un método como una función y, de hecho, es como se tratan en el caso de definir un nuevo objeto en JavaScript. Como las funciones en JavaScript, un método tiene un nombre que se usa para llamarlo; algunas líneas de código se ejecutan cuando se invoca; y si está diseñado así, devolverá un valor como resultado de la acción.
El interior de los métodos de los objetos predefinidos por JavaScript va a permanecer oculto para el programador, ya que se requiere de complejas operaciones en el navegador.
En la sección anterior, se utilizó un método que pertenecía a un objeto documento de JavaScript. La línea de código era:

```
document.write("Last update on " + document.lastModified + ".")
```

El método write() requiere de un parámetro: el string que se debe escribir en el documento. Como puede verse, una llamada a un método incluye un par de paréntesis después del nombre del método. Esto recuerda a la llamada a una función. Es por esto que un método y una función son tan parecidos a la hora de trabajar en JavaScript.
Los métodos ayudan a definir el comportamiento de un objeto. Si las propiedades de un objeto son lo que un objeto es, entonces los métodos definen el comportamiento del mismo. Como puede verse, cuanto más se conozca acerca de las capacidades de un objeto, mejor se podrá planear el usar las propiedades y métodos en un script.

Manejadores de Eventos

Una de las últimas características de los objetos es los llamados manejadores de eventos. Los eventos son acciones que tienen lugar en un documento, usualmente como resultado de la actividad de un usuario, tal como hacer clic en un botón o seleccionar texto en un campo. Algunos eventos, tales como el acto de carga de un documento en un navegador, no son tan obvios.
Casi todos los objetos de JavaScript de un documento reciben eventos de una clase u otra. Lo que determina si el objeto hará algo en respuesta de un evento es un atributo extra que se incluye en la definición HTML del objeto. Este atributo consiste en un nombre de evento y el nombre del método, así como otras funciones que se quieren ejecutar en respuesta a un método. Para ver la utilización de un manejador de eventos, se puede acudir al ejemplo visto en capítulos anteriores del script de muestra una alerta cuando el usuario hace clic en ella.

```
lt;HTML>
<HEAD>
<TITLE>eventhandlers </TITLE>
<SCRIPT LANGUAGE="JavaScript">
function alertUser(alertMsg){
   alert(alertMsg)
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="button" VALUE="Click Me" onClick="alertUser('Ok')">
</FORM>
</BODY>
</HTML>
```

La definición de formato es, simplemente, una entrada estándar. Se debe presentar atención al último atributo onClick="alertUser('Ok')". Los objetos botón, como se verá en las descripciones de capítulos posteriores, reaccionan al hacer clic el ratón. Cuando un usuario hace clic en un

botón, el navegador envía un mensaje onClick al botón. En la definición de botón, el atributo dice que siempre que el botón recibe ese mensaje, debería ejecutar la función alertUser(), pasándole el texto "Ok" como parámetro.
Como muchos de los atributos de los documentos HTML, el nombre de la función se encuentra entre comillas. Si se necesitan más comillas, como en el caso del texto al ser pasado con el manejador de eventos, las comillas dobles pasan a ser comillas simples.
El argumento del manejador de eventos podría también ser un método de objeto directo como en:

onClick="alert('You pressed the button!')"

El único requisito para que este mecanismo funcione es que la función o el objeto mencionada en el atributo onClick= esté previamente cargado en la memoria del navegador. Definiendo una función en la cabecera del documento se asegura que este sea el caso.

Manejadores de Eventos y Métodos.

Algo que puede confundir es el hecho de que tanto los métodos como los eventos en cuanto al nombre se escriben de la misma forma. Un botón es un ejemplo perfecto. Tiene ambas partes: el manejador de eventos onClick y el método Click().
La distinción fundamental es que el manejador empieza a actuar cuando el usuario realiza la acción; en cambio el método es llamado sólo en la instrucción de un script. Como un método es el elemento acción-hacer, se puede decir que el método es una forma de programar las acciones de un usuario. El método y el manejador de eventos son lanzados casi desde sitios distintos: un manejador de eventos se lanza por algo que acontece en el navegador, y un método por un script.

Sintaxis del punto en JavaScript

JavaScript utiliza el punto para realizar instrucciones usando los objetos. Por ejemplo, para poner la propiedad lastModified de un documento, la sintaxis sería:

document.lastModified

El propósito de esta sintaxis es ayudar al navegador a puntualizar con precisión lo que el script busca. Una vez almacenado el documento, el navegador mantiene una lista oculta de cada objeto definido en las etiquetas HTML.
En el documento de tipo estándard HTML (sin marcos en la ventana), solamente un documento puede ser mostrado al mismo tiempo; de esta manera el objeto documento es claramente el documento que se encuentra en la ventana. Entonces, desde el documento se querra que el navegador localice una propiedad particular.
Esto se puede complicar en el caso que existan numerosos objetos de un tipo particular en un documento. Por ejemplo, se supone una plantilla consistente en varios elementos de entrada. Algunos de estos elementos son botones circulares, uno es un campo de texto, y otro es una caja de chequeo. Si se quiere conocer qué texto está en el campo, tiene que decirle al navegador exactamente que propiedadquiere y desde que objeto. Una expresión en la sintaxis con punto en JavaScript proporcionará al navegador la dirección completa del objeto.
Para especificar un elemento particular de un documento plantilla, se debe utilizar la propiedad forms del objeto documento. Como puede haber múltiples plantillas en el documento, debe ser utilizada la notación punto para señalar la plantilla que debe ser numerada. Mucha de la numeración de objetos en JavaScript empieza con cero, así que la referencia document.forms[0] significa la primera plantilla del documento. Ahora la instrucción debe meterse dentro de la plantilla para obtener un valor de un elemento de entrada particular. Para estos elementos, la dirección a seguir puede usar el nombre del objeto. A causa de que en este caso se quiere extraer el valor de la propiedad del campo de texto llamado textone, la referencia completa será:
document.forms[0].textone.value

Entonces, esta forma de direccionar ayuda al navegador a ampliar su alcance; desde el

documento hasta la plantilla primera, justo hasta el campo textone y de ahí hasta el valor de la propiedad. El listado de código siguiente aparece a continuación muestra como funciona esta construcción.

Ejemplo:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="radio" NAME="sex">Male
<INPUT TYPE="radio" NAME="sex">Female
enter birthdate: <INPUT TYPE="text" NAME="birthdate" SIZE="10">
</FORM>
<SCRIPT LANGUAGE="JavaScript">
document.writeln(document.forms[0].birthdate.value)
</SCRIPT>
</BODY>
</HTML>
```

La primera vez que se cargue este documento, será la vez en la que los elementos se carguen en la página. Después de introducir el valor en el campo y hacer clic en el botón de recarga, el valor desde el campo será mostrado en la línea de texto debajo de la plantilla. En este ejemplo se obtiene la propiedad [value] de un objeto [birthdate] de una propiedad [forms] de un objeto [document]. Como se ha visto, se ha limitado el estudio a la parte de la OOP que tiene que ver con JavaScript. En el próximo capítulo, se utilizarán los conceptos adquiridos de programación Orientada a Objetos.

Capítulo 5

Objetos en JavaScript

Un objeto es una estructura que contiene una serie de propiedades que no son más que variables de JavaScript u otros objetos, así como una serie de funciones asociadas que manipulan dichas variables, conocidas como métodos. JavaScript contiene objetos que están predefinidos en el Navegador del Cliente y del Servidor, así como también permite definir objetos al programador, para su utilización determinada dentro de la página.
En este capítulo se describe como utilizar los objetos y métodos para la manipulación de variables, así como su creación a partir de la utilización de constructores o de inicializadores.En capítulos posteriores se estudiará los objetos predefinidos más importantes utilizados por JavaScript, así como su jerarquía.
Este capítulo contiene las siguientes secciones, en las cuales se tratarán los temas anteriores:

- Objetos y Propiedades.
- Creación de Objetos.

Objetos y Propiedades.

Un objeto de JavaScript posee una serie de propiedades asociadas, a las cuales se puede acceder a partir de la notación siguiente:

```
objectName.propertyName
```

Para asignar una propiedad a un objeto es necesario asignarle un valor. Por ejemplo, se supone un objeto denominado informaticBook1, el cual ya ha sido previamente definido. Es posible atribuirle una serie de propiedades de la forma siguiente:

```
informaticBook1.title = "Programación en JavaScript"
informaticBook1.author = "Danny Goodman"
informaticBook1.year = 1997
```

En capítulos anteriores se definió el concepto de array como un conjunto de valores asociados a partir del nombre de una única variable. Entonces, mediante la estructura tipo array es posible acceder a cada una de las propiedades de un objeto.Así, por ejemplo, es posible acceder a las propiedades del objeto anterior de la forma siguiente:

```
informaticBook1["title"] = "Programación en JavaScript"
informaticBook1["author"] = "Danny Goodman"
informaticBook1["year"] = 1997
```

Este tipo de estructura se conoce con el nombre de array asociativo, ya que cada elemento índice está asociado con un valor tipo caracter. El siguiente ejemplo define una función que permite mostrar cada una de las propiedades del objeto cuando se pasa como argumento el nombre del objeto y de la propiedad que se pretende visualizar.

Ejemplo.

```
function show_props (obj, obj_name){
   var result=""
   for (var i in obj)
      result +=obj_name + "." + i + "=" +obj[i] + "\n"
   return result
}
```

Entonces, si se llama a la función como show_props(informaticBook1, "informaticBook1"), se retornan los valores siguientes:

```
informaticBook1.title = Programación en JavaScript
informaticBook1.author = Danny Goodman
informaticBook1.year = 1997
```

Creación de Objetos.

El lenguaje de programación JavaScript tiene un número de objetos predefinidos aunque, como se ha comentado anteriormente, también permite definir objetos personalizados. A partir de la versión 1.2 de JavaScript, es posible la creación de objetos a partir de la utilización de inicializadores. De forma alternativa, si el navegador no puede interpretar la versión 1.2 de JavaScript, es posible la creación de objetos mediante una función constructora e inicializar el objeto utilizando esta función y el operador new.
En esta sección se tratarán los siguientes aspectos respecto a la construcción y utilización de objetos:

- Utilización de Inicializadores.
- Utilización de Constructores.
- Indexado de las Propiedades del Objeto.
- Definición de las Propiedades de un Objeto Tipo.
- Definición de Métodos.
- Utilización de this para Referencia a Objetos.
- Eliminación de Objetos.

Utilización de Inicializadores.

La sintaxis que se utiliza para la creación de un objeto a partir de un inicializador de objetos es la siguiente:

```
objectName = {property1:value1, property2:value2,...,property n:value n}
```

donde objectName es el nombre del nuevo objeto, cada property es un identificador (ya sea un nombre, un número o un literal) y cada value es una expresión cuyo valor es asignado al identificador de la propiedad. El nombre del objeto objectName y el comando de asignación es opcional.
Cuando un objeto se crea a partir de un inicializador en la parte superior del código fuente del script, JavaScript interpreta el objeto cada vez que se evalua la expresión contenida en él.
La siguiente sentencia crea un objeto y lo asigna a la variable x solo y sólo si la expresión cond se evalua como verdadera.

Ejemplo

```
if (cond) x = {hi:"there"}
```

El siguiente ejemplo crea el objeto InformaticBook1 con tres propiedades. Notar que la propiedad index se define como un objeto con sus propiedades.

Ejemplo

```
informaticBook1 = {title:"Programación en JavaScript", author:"Danny Goodman", year:1997,
                   index:{chapter:1,section:7}}
```

Utilización de Constructores.
Es posible crear un objeto a partir de la utilización de un constructor. Un constructor es una función que inicializa un objeto. Entonces, cuando se crea un objeto del tipo que sea, se llama al constructor, pasándole cada una de las propiedades del objeto como argumentos de la función.
A partir de aquí, se tiene que para crear un objeto se debe definir una función constructora del objeto tipo, que especifique su nombre, propiedades y métodos. Una vez definida esta función, se debe crear una instancia del objeto mediante el comando new. En el siguiente ejemplo se crea el objeto informaticBook1 definido en la sección anterior a partir de un inicializador, el cual posee las propiedades de title, author y year.

Ejemplo

```
function book (title, author, year){
   this.title = title;
   this.author = author;
   this.year = year;
}
```

Notar que this se utiliza para asignar valores a las propiedades de los objetos a partir de los valores pasados a la función constructora. Entonces, es posible una vez definido el constructor, crear nuevos objetos ,llamados informaticBook1 e informaticBook2 de la forma siguiente:

```
informaticBook1 = new booK("Programación en JavaScript", "Danny Goodman", 1997)
informaticBook2 = new book("C Guía de Autoenseñanza", "Herbert Schildt", 1994)
```

Esta sentencia, como se puede ver, crea dos objetos denominados informaticBook1 e informaticBook2 y asigna a cada uno de los valores sus correspondientes propiedades.
Un caso importante dentro de la utilización de objetos se da cuando un objeto referencia a otro; es decir, cuando un objeto tiene a otro como una propiedad del primero. Para comprender realmente lo que esto implica, se supone que en los objetos definidos anteriormente es necesario introducir una nueva propiedad, una imagen de la portada de cada libro. Cada imagen tiene un URL para el fichero de imágenes y otras informaciones, tales como un número de referencia y los derechos de autor. Una forma de llevar esto a cabo es creando una definición de objeto aparte para una base de datos de imágenes. La definición podría ser la que se muestra a continuación:

```
function image (name, URL, copyright, refNum){
   this.name = name
   this.URL = URL
   this.copyright = copyright
   this.refNum = refNum
}
```

Entonces, seria necesario crear objetos de imágenes individuales para cada imagen. Una de estas definiciones sería como sigue:

```
ProgJSImage = new image ("Programación en JavaScript", "/images/ProgJS.gif",
                 "(c)1997 ANAYA MULTIMEDIA",0003)

ProgCImage = new image ("C Guía de Autoenseñanza", "/images/ProgC.gif",
                 "(c)1994 McGRAW-HILL", 0004)
```

Para adjuntar un objeto de imagen a un objeto de book es necesario modificar la definición del objeto de book, para incorporar una propiedad más. La nueva definición del objeto book sería la siguiente:

```
function book (title, author, year, image){
   this.title = title;
   this.author = author;
   this.year = year;
   this.image = image
}
```

Una vez creados los objetos de imágenes, hay que crear los objetos de books, pasando un parámetro más, un objeto de imagen que se quiere asociar a ese objeto. Esto se realiza de la forma siguiente:

```
informaticBook1 = new book("Programación en JavaScript", "Danny Goodman", 1997,
                      ProgJSImage)
informaticBook2 = new book("C Guía de Autoenseñanza", "Herbert Schildt", 1994,
                      ProgCImage)
```

Para acceder a las propiedades del objeto se debe aplicar la sintaxis siguiente:

```
informaticBook1.title
informaticBook2.title
```

En el caso que se desee acceder a las propiedades de la imagen incluida en el objeto informaticBook1, es necesario definir el código siguiente:

```
informaticBook1.image.copyright
```

También es posible definir propiedades para un objeto ya creado. Por ejemplo, es posible añadir una propiedad al objeto anterior que defina la existencia de un CD de ejemplos del libro. En este caso, se tiene que:

```
informaticBook1.CD = true
```

Esta nueva propiedad no afectará a los demás objetos que se han creado anteriormente. En el caso que se desee añadir esta propiedad a todos los objetos creados a partir del constructor, será necesario definir esta propiedad en ese mismo constructor.

Indexado de las propiedades de los objetos.

En JavaScript 1.0 es posible referirse a un objeto por el nombre asociado a cada una de las

propiedades o bien por el número de índice. Ahora bien, en JavaScript 1.1 o versiones posteriores, si se define una propiedad de un objeto a partir de un nombre, debe de referirse a esta propiedad a partir de ese nombre. De la misma forma, si se define una propiedad a partir de un índice, debe de referirse a esta propiedad a partir de ese índice.
Esta condición se aplica cuando se crea un objeto y sus propiedades a partir de un constructor, como en el ejemplo del objeto book. Así, si se definen las propiedades inicialmente con un índice, como informaticBook1[2] = "Danny Goodman", puede referirse a esa propiedad como informaticBook1[5].

Definición de las Propiedades de un Objeto Tipo.

Es posible definir una nueva propiedad a un objeto previamente creado a partir de la utilización del comando prototype. A partir de la utilización de esta expresión se puede definir una propiedad que podrá ser referenciada sobre todos los objetos de este tipo. Por ejemplo, el siguiente código incluye una segunda propiedad, numberChapters, a todos los objetos del tipo book:

```
book.prototype.numberChapters = null
```

Entonces, para referirse a esta propiedad, tan sólo es necesario ejecutar la siguiente expresión:

```
informaticBook1.numberChapters = 13
```

De esta forma se está asignando al libro Programación en JavaScript el número de capítulos que posee.

Definición de Métodos.

Un método es una función asociada con un objeto, de manera que se define de la misma forma que una función estándard. La sintaxis utilizada para asociar una función a un objeto ya existente se define de la forma siguiente:

```
object.methodname = function_name
```

donde object es el objeto definido anteriormente, methodname es el nombre que se le asigna al método y function_name es el nombre de la función que se define como método.
A partir de la definición del método, es posible realizar una llamada al método que actua sobre las propiedades del objeto de la forma siguiente:

```
object.methodname(params);
```

En la práctica, es usual definir los métodos para un tipo de objeto incluyendo el método en la función constructora del objeto. Por ejemplo, es posible definir una función que permita visualizar en pantalla el objeto predefinido anteriormente book. La función sería la siguiente:

```
function displayBook(){
   var result = "Title:"+this.title+" "this.author:"+author" " this.year:"+year".")

   pretty_print(result)
}
```

donde pretty_print es la función que permite visualizar en pantalla una cadena de caracteres.
Entonces, para añadir este método al objeto book, es necesario incluir en su definición la sentencia:

```
this.displayBook = displayBook;
```

Entonces, la definición del objeto book sería de la forma siguiente:

```
function book (title, author, year, image){
   this.title = title;
```

```
    this.author = author;
    this.year = year;
    this.image = image
    this.displayBook = displayBook
}
```

A partir de la definición del obejto, es posible llamar al método displayBook para cada uno de los objetos de la forma siguiente:

```
informaticBook1.displayBook()
informaticBook2.displayBook()
```

Utilización de this para Referencia a Objetos.

El lenguaje de programación JavaScript tiene un comando especial: this. Este comando se utiliza para en un método para referenciar un objeto. Por ejemplo, se supone una función llamada validate que evalua una propiedad numérica de un objeto, devolviendo el valor máximo y mínimo de este.

Ejemplo:

```
function validate(obj, lowval, hival){
    if ((obj.value < lowval)||(obj.value > hival))
        alert("Invalid Value!")
}
```

Entonces, es posible llamar a la función validate a partir del evento onChange, usando this para referenciar al objeto, como en el siguiente ejemplo:

```
<INPUT TYPE="text" NAME="age" SIZE=3 onChange="validate(this,18,99)">
```

Eliminación de Objetos.

En el lenguaje de programación JavaScript, es posible eliminar un objeto usando el operador delete. El siguiente código muestra la forma de eliminar un objeto:

Ejemplo:

```
myobj = new Number()
delete myobj
```

En la versión 1.1 de Javascript es posible eliminar un objeto a partir de aplicar la referencia null sobre ese objeto (siempre y cuando sea la última referencia a ese objeto). Una vez apliacada esta referencia, JavaScript omite el objeto como parte de cualquier expresión. Ahora bien, en versiones anteriores de JavaScript, como la 1.0, no es posible eliminar los objetos con un comando, de manera que la única posibilidad es eliminarlos del script.

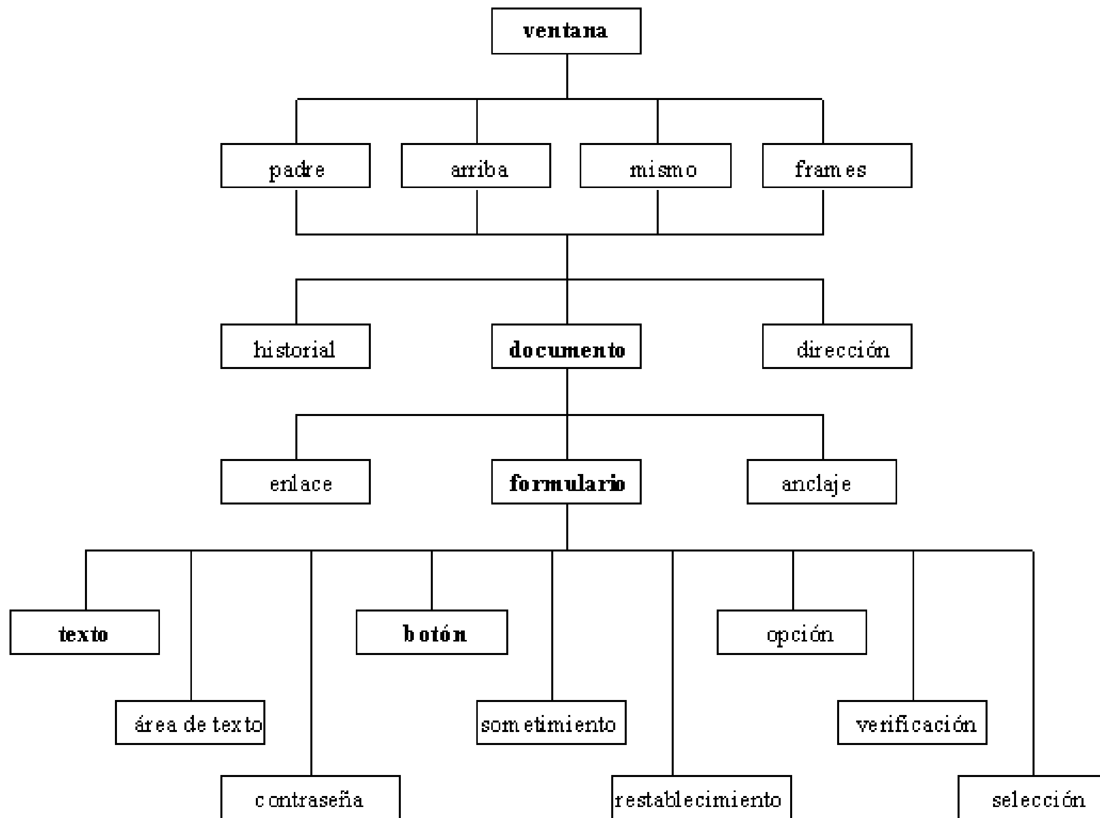
Capítulo 6

Objetos Predefinidos en JavaScript.

Jerarquía de Objetos en JavaScript.

La función principal de la jerarquía de objetos en JavaScript para un programador es un mecanismo que es dado a los scripts para poder referenciar un objeto con respecto a todos los objetos que contiene una ventana del navegador.
La figura siguiente muestra la jerarquía total de JavaScript. Como se puede ver, la ventana

objeto está en la parte más alta del esquema.Cada script que programa en JavaScript tiene una ventana del navegador, ya sea en forma de ventana o como elemento plantilla.



De todos los objetos mostrados en la figura anterior, los que están en negrita son los que se van a trabajar en mayor medida. En este capítulo, se tratarán las ventana, la localización y la historia de objetos. El objeto documento, el más importante en JavaScript, se describe en su propio capítulo (Capítulo 7).

Definición de Objetos Predefinidos.

En el siguiente capítulo y en el capítulo 7 se presenta una descripción detallada de cada uno de los objetos de las ventanas. Cada definición de objetos empieza con un sumario de la lista de sus propiedades, métodos y manejadores de eventos, de manera que se realizará una descripción de los más importantes.
Siempre que aparece una definición de sintaxis, notar las pocas convenciones usadas para designar a los elementos, junto a sus parámetros opcionales y las variables que describen que clase de datos pueden ponerse en estos puntos. Todas las definiciones de sintaxis y los ejemplos de código son HTML, por lo que a lo largo del capítulo se visualizarán muchas etiquetas del lenguaje HTML. El ejemplo siguiente muestra la definición del objeto botón.

Ejemplo

```
<INPUT TYPE="button" NAME="objectName" VALUE="buttonText" [onClick="handlerText"]>
```

Algunos atributos de parámetros (para NAME=, VALUE= y onClick=) tienen asignados nombres con significado para el script; en este caso, el nombre que va a aparecer en la etiqueta del botón y que se quiere que haga el botón cuando se haga click en él. El último atributo de la definición aparece entre paréntesis cuadrados.
Esta convención significa que el atributo es opcional. En este caso, si se omite el atributo

onClick, entonces el botón se convierte en otro botón HTML como otros que se pueden especificar en el script.
En otras definiciones, específicamente los parámetros para los métodos, los valores a ser suministrados deben ser de un tipo de datos particular (string, boolean o number). Cuando el valor debe ser de un tipo especial, se especifica en el lugar de este parámetro el tipo de dato con su nombre.
Algunos métodos devuelven valores después de ejecutarse. Por ejemplo, un método de ventana muestra una caja de diálogo para indicar al usuario que debe introducir un texto. Cuando un usuario hace clic en el botón de OK, el texto introducido en la ventana de diálogo se pasa como un valor que puede ser asignado a una variable o bien es utilizado directamente como si fuera un parámetro para otro método. Cada método listado en las siguientes definiciones de objetos indica el tipo de valor, si existe alguno, que es devuelto por el método.
El último elemento a explicar acerca de estas definiciones es el modo de manejo de los valores de las propiedades. Cada propiedad posee un valor que debe pertenecer a un tipo de datos válido por JavaScript.
Es importante para su aplicación que tipo de propiedad es devuelta (string, number, boolean). Por lo tanto se debe indicar el tipo de valor requerido por cada propiedad.
Una vez realizadas las consideraciones que se deben tener en cuenta para la explicación de cada uno de los objetos, se pasa a la descripción de los siguientes:

- El Objeto Ventana (Window)
- El Objeto de frame
- El Objeto de dirección
- El Historial de Objetos

El Objeto Ventana(window).

Propiedades	**Métodos**	**Manejador de eventos**
frames	alert	onLoad=
parent	close()	onUnload=
self	confirm()	
top	open()	
status	prompt()	
defaultStatus	setTimeout()	
window	clearTimeout()	

Sintaxis.
Para crear una ventana se utiliza la sintaxis siguiente:

windowObject = window.open ([parameters])

En el caso de ser necesario acceder a las propiedades o métodos de la ventana, se tiene el código siguiente:

window.property | method ([parameters])

self.property | ([parameters])

windowObject.property | method ([parameters])

A continuación se realiza una explicación de las propiedades más importantes del objeto window, en la cual se podrá ver la utilización de esta sintaxis a partir de numerosos ejemplos.

Información sobre el objeto

El objeto ventana posee unas características únicas, ya que es en él en el que acontecen todas las operaciones, y por ello, usualmente se omiten todas las referencias a él en los demás

objetos. Así, en el caso de que sea necesario la utilización del método write() desde una ventana que contiene el documento que ejecuta el script, la referencia utilizada puede ser window.document.write, o bien, document.write(), de manera que se asume que la ventana es parte de la referencia.
Esta situación es más compleja en ventanas con múltiples marcos y para tiempos en los que los script son creados con nuevas ventanas de navegadores.
No siempre es necesario especificar un objeto ventana en el código JavaScript. Cuando ejecutamos el navegador, usualmente se abre una ventana. Esto ocurre siempre aunque se habra una ventana vacía. Por lo tanto, cuando un usuario carga su página en un navegador, el objeto ventana del documento es creado por el script para acceder a él según este desee. Entonces, se debe pensar en el objeto ventana como una propiedad de un objeto que es este mismo objeto. Esto puede paracer confuso, pero es común en entornos orientados a objetos.

Propiedades

status

Valor: cadena de caracteres.
Obtenible: sí
Parametrizable: sí

En la parte de abajo de la ventana del navegador está la barra de estado. Entonces, es posible controlar el contenido temporal de este campo de texto asignando un string a la propiedad status del objeto ventana.
En este caso, se debe ajustar la propiedad status para responder a los eventos que tienen un efecto temporal, como pueda ser el manejador de eventos onMouseOver= del objeto de enlace.Cuando la propiedad status está disponible en esta situación, sobreescribe otras configuraciones del estado ventana. Si el usuario mueve el ratón lejos del objeto de enlace que cambia la barra de estado, la barra vuelve a tener las configuraciones por defecto.
Por ejemplo, es posible utilizar la barra de estado para definir la naturaleza del destino de un enlace, simplemente introduciendo un mensaje en la barra de estado en respuesta al manejador de eventos onMouseOver.

Ejemplo:

```
<HTML>
<HEAD>
<TITLE>window.status example</TITLE>
</HEAD>
<BODY>
<A HREF="http://www.Telyse.net" onMouseOver="window.status'Visit Telyse.net home page.';
return true"> Telyse.net</A><P>
</BODY>
<HTML>
```

defaultStatus

Valor: cadena de caracteres.
Obtenible: sí
Parametrizable: sí

Una vez que el documento se almacena en una ventana o en un marco, el campo de mensaje de la barra de estado puede mostrar un string que será visible todo el tiempo que el puntero esté encima, siempre y cuando no haya un objeto que tenga prioridad en la barra de estado. La propiedad window.defaultStatus es normalmente un string vacío, pero puede establecerse como una propiedad en respuesta a cualquier manejador de eventos. Cualquier propiedad de esta configuración sería temporalmente sobrescrita cuando un usuario mueva el puntero del ratón encima de un enlace a un objeto.
Un navegador JavaScript responde a cualquier evento onLoad cada vez que se carga el documento.En el caso de ser necesario establecer los mensajes de estado por defecto en la

ventana del navegador es necesario la utilización del manejador de eventos onLoad=. Para esto, este manejador se debe de definir dentro de la etiqueta <BODY> del script.

Ejemplo:

```
<HTML>
<HEAD>
<TITLE>window.defaultStatus example</TITLE>
</HEAD>
<BODY onLoad="window.defaultStatus='Welcome to my home page'">
<A HREF="http://www.Telyse.net" onMouseOver="window.status'Visit Telyse.net home page.';
return true"> Telyse.net</A><P>
</BODY>
<HTML>
```

parent

Valor: objeto ventana.
Obtenible: sí
Parametrizable: no

La propiedad parent se utiliza cuando un documento se muestra como parte de una ventana múltiple. Los documentos HTML que los usuarios ven en los marcos son distintos según el documento que especifica la configuración de la ventana completa. Este documento que define la configuración, mientras esté en memoria del navegador, no será visible al usuario.
En el caso que sea necesario referenciar a objetos o propiedades dentro de la ventana de establecimiento de marcos, se debe utilizar la propiedad parent. Por ejemplo, una ventana hija puede llamar a una función deifinida en la ventana padre. La referencia podría ser:

```
parent.functionName([parameters])
```

Un ejemplo típico de la utilización de la propiedad parent es cargar temporalmente datos de los scripts en los campos de texto. Entonces, para demostrar cómo varias propiedades de objetos ventanas refieren a los niveles de ventanas en un entorno de múltiples marcos, se puede definir un programa que registre y muestre con una alerta los valores de muchas propiedades de las ventanas hijas, así como también la propiedad document.title de dos diferentes referencias a ventanas. En cada uno de los marcos se situaría un botón que mostraría las propiedades de cada una de las ventanas.

Ejemplo:

```
<HTML>
<HEAD>
<TITLE>parent example</TITLE>
</HEAD>
<FRAMESET ROWS="50%,50%">
  <FRAME NAME="Child1" SRC="1st07-07.htm">
  <FRAME NAME="Child2" SRC="1st07-07.htm">
</FRAMESET>
<HTML>

<HTML>
<HEAD>
<TITLE>Windos example</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function WindowData(){
  var msg="";
  msg=msg+"window.object:"+window+"\n"
  msg=msg+"self.property:"+self+"\n"
  msg=msg+"self.document.title:"+self.document.title+"\n\n"
```

```
    msg=msg+"top.document.title:´+top.document.title+"\n"
    msg=msg+"top.document.title:"+top.document.title+"\n"
    msg=msg+"top property:"+top+"\n"
    alert(msg)
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="button" NAME="collector" VALUE="View properties of this window frame..."
onClick="WindowData()">
</FORM>
</BODY>
<HTML>
```

<u>top</u>

Valor: objeto ventana.
Obtenible: sí
Parametrizable: no

La propiedad top del objeto ventana se refiere a la ventana de más arriba en la jerarquía de JavaScript. Para una ventana con un único marco, la referencia es al mismo objeto de la ventana, por lo que no se debe incluir al objeto ventana como parte de la referencia. En el caso de una ventana de múltiples marcos, la ventana de arriba es la única que define el primer establecimiento de marcos. El usuario no ve realmente la ventana de arriba en un entorno múltiples marcos, pero el navegador lo carga como un objeto en su memoria. Esto es por lo que la ventana de arriba tiene un mapa a los otros marcos y sus marcos hijos pueden llamar hacia él. Esta referencia sería como:

top.functionName([]parameters)

<u>frames</u>

Valor: objeto ventana.
Obtenible: sí
Parametrizable: no

En una ventana de múltiples marcos,la ventana top o la padre contiene cualquier número de marcos, cada uno de los cuales actúa como una ventana objeto con todas sus características. La propiedad frameset juega un papel importante en el caso que deba referenciarse a un objeto situado en cualquier marco. Así, en el caso que un botón en un marco está programado para mostrar un documento en otro marco, el manejador de eventos del botón debe poder decir a JavaScript donde mostrar el nuevo documento HTML. La propiedad frameset debe ser utilizada en esta tarea.
Para utilizar la propiedad frames de forma adecuada se debe realizar una referencia a partir de una propiedad parent o top. Así, por ejemplo, para investigar cuantos marcos hay actualmente activos en una ventana se usa la expresión:

parent.frames.lenght

El navegador puede cargar información acerca de todos los marcos visibles en un array numerado, con el primer marco con el número 0.

parent.frames[0]

Por tanto, si la ventana muestra tres marcos (los cuales estarían indexados como: frames[0], frames[1] y frames[2], respectivamente) la referencia para recuperar la propiedad del título del segundo marco, sería la siguiente:

parent.frames[1].document.title

Cada marco, como se vio en el ejemplo anterior, tiene un nombre unido a ella. Si se especifica un nombre en la etiqueta <FRAME>, este nombre está disponible para el objeto JavaScript. Entonces, para extraer el nombre del segundo marco de la ventana sería:

parent.frames[1].name

A partir de aquí, se puede realizar el ejemplo anterior pero utilizando la propiedad frames del objeto ventana.

Ejemplo:

```
<HTML>
<HEAD>
<TITLE>parent example</TITLE>
</HEAD>
<FRAMESET ROWS="50%,50%">
   <FRAME NAME="Child1" SRC="1st07-07.htm">
   <FRAME NAME="Child2" SRC="1st07-07.htm">
</FRAMESET>
<HTML>

<HTML>
<HEAD>
<TITLE>Frames example</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function Window2Data(){
   var msg="";
   msg=msg+"window.frames.length:"+window.frames.length+"\n"
   msg=msg+"parent.frames.length:"+window.frames.length+"\n"
   msg=msg+"window.frames:"window.frames+"\n\n"
   msg=msg+"window.frames:"+parent.frames+"\n"
   msg=msg+"parent.frames[0].name:"+parent.frames[0].name:+"\n"
   alert(msg)
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="button" NAME="collector" VALUE="View properties of this window frame..."
onClick="Window2Data()">
</FORM>
</BODY>
<HTML>
```

Métodos.

<u>alert (message)</u>

Devuelve: Nada

Un diálogo de alerta es una ventana módulo que presenta un mensaje que el usuario lo hace desaparecer con un simple botón de OK. Mientras la alerta está detenida la aplicación está detenida. El usuario debe hacer desaparecer esta ventana para continuar trabajando en el navegador o en el ordenador.

Ejemplo:

```
<HTML> <HEAD> <TITLE>window.alert()<TITLE> </HEAD>
<BODY>
```

</SCRIPT LANGUAGE="JavaScript">
alert("This document was last saved on"+ document.lastModified+".") <INPUT TYPE="button" NAME="collector" VALUE="View properties of this window frame..." </SCRIPT> </BODY> <HTML>

open("URL", "windowName")

Devuelve: un objeto ventana que representa la ventana creada recientemente o un valor nulo si el método falla.

Apartir del método window.open, el script facilita un diseñador de páginas Web que comprende una extensa serie de opciones para las distintas formas en que una ventana del navegador Web puede aparecer en la pantalla del ordenador del usuario. Así,la nueva ventana especificada por un script puede personalizar el conjunto de elementos de una ventana. A continuación se expone una tabla en la que se muestran los atributos de este método que pueden ser utilizados para controlar la apertura de una ventana.

Atributo	**Valor**	**Descripción**
toolbar	booleano	Hacia atrás, hacia adelante
location	booleano	Campo que muestra la URL actual
directories	booleano	Otros botones de la fila
status	booleano	Barra de estado en la parte inferior
menubar	booleano	Barra de menú en la parte superior
scrollbars	booleano	Muestra barras de desplazamiento
resizable	booleano	Elementos del interfaz que permiten cambiar el tamaño
width	Puntos de imagen	Ancho exterior de la ventana en puntos
height	Puntos de imagen	Altura exterior de la ventana en puntos

Entonces, en el caso que se desee crear una nueva ventana que solamente muestre la barra de herramientas y la barra de estado, y que se pueda cambiar el tamaño, el método sería el siguiente:

window.open("newURL","New Window", "toolbar,status,resizable")

close()

Devuelve: Nada.

El método window.close() cierra la ventana del navegador que el objeto de ventana de referencia. Este es el método que se utiliza con mayor probabilidad al cerrar las subventanas creadas desde una ventana del documento principal. Si la llamada para cerrar la ventana procede de una ventana distinta a la nueva subventana, es fundamental que el objeto de ventana mantenga un registro del objeto subventana. Esto se realiza almacenando el valor devuelto desde el método window.open() en una variable global que estará disponible con posterioridad para otros objetos. Si, por el contrario, se llama al método window.close() desde un objeto que está dentro de la nueva subventana, la referencia al objeto window o self será suficiente.

SetTimeout("expression",milisecondsDelay)

Devuelve: Valor ID para su uso con el método window.clearTimeout().

Un intervalo de espera es la cantidad de tiempo, en milisegundos, antes de que una expresión establecida se evalúe. Para desconectar este tiempo de espera cuando un usuario navega durante el período de tiempo asignado, cualquier botón con el que el usuario interactue necesitará llamar al método clearTimeout(), para cancelar la temporización actual.
La expresión con el primer parámetro del método window.setTimeout() puede ser una llamada a una función o a un método. La expresión se evalua después de que el método expire
Un ejemplo de este método es la visualización del tiempo actual a partir de la utilización de la función updateTime(), que muestra el formato en hh:mm am/pm en la barra de estado.

Ejemplo:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript"> <TITLE>Status Bar Clock<TITLE>
function updateTime(){
   var now =new Date()
   var theHour = now.getHours()
   var theMin=now.getMinutes()
   var theTime=" "+((theHour>12)?theHour-12: theHour)
   theTime+=((theMin<10)?":0":":")+ theMin
   theTime+=(theHour>=12)?"pm":"am"
   theTime+=((flasher)?" ": "*")
   flasher =!flasher
   window.status = theTime
   timerID = setTimeout("updateTime()",1000)
</SCRIPT> </HEAD>
<BODY onLoad="updateTime()">
</BODY>
<HTML>
```

Manejadores de sucesos

<u>onLoad=</u>

El suceso onLoad se envía a la ventana actual al final del proceso de almacenamiento del documento. En ese momento, todos los objetos y componentes del script del documento están almacenados en la memoria del navegador.
El manejador de sucesos onLoad es un atributo de una etiqueta <BODY> de un documento con un solo frame o un atributo de la etiqueta <FRAMESET> de la ventana superior de un documento con frames múltiples.
Es posible utilizar los ejemplos siguientes para insertar un manejador onLoad= en un documento.

Ejemplo:

```
<HTML>
<HEAD>
</HEAD>
<BODY [otherAtributes]onLoad="statementOrFunction">
<BODY>
</HTML>
```

Ejemplo:

```
<HTML>
<HEAD>
</HEAD>
<FRAMESET [other attributes] onLoad="statementsOrFunction">
   <FRAME> frame specifications</FRAME>
```

```
<FRAMESET>
</HTML>
```

El tipo de operaciones adecuadas para un manejador de sucesos onLoad= son aquellas que pueden ejecutarse rápidamente sin que el usuario intervenga.

onUnLoad=

Un suceso de descarga llega a la ventana actual justo antes de que se quite un documento. Los manejadores de sucesos onUnLoad= se especifican en los mismos sitios de un documento HTML que los manejadores onLoad=. Ambos manejadores de sucesos pueden aparecer sin problemas en la misma etiqueta <BODY> o <FRAMESET>. Cuando el documento está a punto de desaparecer de la ventana, el manejador de sucesos onUnLoad= apenas permanece guardado en la memoria del navegador esperando a que llegue el suceso onUnLoad=.

El objeto de frame

Propiedades	**Métodos**	**Manejador de sucesos**
frames	alert()	onLoad=
parent	close()	onUnLoad=
self	confirm()	
top	open()	
status	prompt()	
defaultStatus	setTimeout()	
window	clearTimeout()	

Sintaxis

Creación de una frame:

```
<FRAMESET>
ROWS="ValueList"
COLS="ValueList"
[onLoad="handlerTextOrFunction"]
[onUnload="handlerTextOrFunction"]>
   <FRAME SCR="locationOrURL" NAME="firstFrameName">
   ...
   <FRAME SCR="locationOrURL" NAME="lastFrameName">
</FRAMESET>
```

Acceso a las propiedades o métodos de otro frame:

```
parent.frameName.property|method([parameters])

parent.frame[i].property|method([parameters])
```

Infomación sobre el objeto

Un objeto de frame se comporta exactamente igual que un objeto de ventana, aunque haya sido creada como parte de un conjunto de frames por otro document. Un objeto de frame siempre tiene una propiedad top y una propiedad parent diferente de su propiedad self. Si se almacena un documento visualizado normalmente en una frame de una sola ventana del navegador, su ventana no es mayor que un frame.

El objeto de dirección

Propiedades	**Métodos**	**Manejador de eventos**

hash	(ninguno)	(ninguno)
host		
hostName		
href		
pathname		
port		
protocol		
search		

Sintaxis

Asignación de la ventana actual a una nueva dirección:

[window.] location="URL"

Acceso a las propiedades o métodos de la dirección:

[window.] location. property|method([parameters])

Información sobre el objeto

El objeto dirección representa la información de la URL de cualquier ventana que esté actualmente abierta. Para una ventana con un único frame, sólo hay una URL que se aplica al objeto de dirección: la URL que se muestra en el campo de Dirección, en la parte superior de la ventana del navegador.
Una ventana con frames múltiples muestra la URL de la ventana padre en el campo de Dirección. Sin embargo, si se omite el parent o top en una referencia a un objeto de dirección, JavaScript observa la URL del frame en el que se encuentra la orden. Para obtener la información de la URL de un documento que se encuentra en otro frame, la referencia al objeto de dirección debe incluir el frame de la ventana.

Propiedades

href

Valor:cadena de caracteres
Obtenible:sí
Parametrizable:sí

La propiedad location.href proporciona una cadena de entrada URL correspondiente a un objeto de la ventana. A partir de la utilización de esta propiedad, aparecerá a la izquierda del elemento asignado, un método JavaScript para abrir, en una ventana, la URL. Cualquiera de las siguientes sentencias cargará el índice de una página Web en una ventana del explorador:

window.location="http://www.telyse.net"

window.location.href="http://www.telyse.net"

Una posible utilización de esta propiedad es la extracción en una cadena de caracteres del nombre del directorio actual, para que cualquier otra sentencia pueda añadir un documento a la URL, antes de cargarlo en una ventana. Aunque otras propiedades de un objeto producen varias partes de una URL, ninguna de ellas proporciona la URL completa a la URL del directorio actual. Pero se puede valer también de las técnicas de manipulación del JavaScript para relizar esta tarea. El Ejemplo siguiente se vale de la función unescape() al principio de la cadena que captura la URL. De esta forma aparece el nombre del path en el cuadro de aviso de las versiones del explorador, en donde se suele mostrar la versión codificada en ASCII.

Ejemplo:

```
<HTML>
<HEAD>
<TITLE>Extract pathname</TITLE> <SCRIPT LANGUAGE="JavaScript"> function
getDirPath(URL){
   var result = unescape(URL.substring(0,(URL.lastIndexOf("/"))+1)
   return result
function showDirPath(URL){
   alert(getDirPath(URL))
}
<SCRIPT> </HEAD>
<BODY">
<FORM>
<INPUT TYPE="button" VALUE="View directory URL"
onClick="showDirPath(window.location.href)"> </FORM>
<BODY>
</HTML>
```

<u>hash</u>

Valor:cadena de caracteres
Obtenible:sí
Parametrizable:sí

El señalador (#) es un convenio que se utiliza para que las URL dirijan al explorador a una marca del documento. Cualquier nombre que se asigne a la marca (con la pareja de señaladores <A NAME="...">...</A>) pasará a ser parte de una URL, si figura detrás del señalador. La propiedad hash para la localización de un objeto está formada por el señalador y el nombre.
De la misma forma que es posible dirigirse a una URL a través de la propiedad window.location, puede desplazarse hasta otra marca del mismo documento. Para ello, sólo se debe ajustar la propiedad de localización hash. En el siguiente ejemplo se presenta un script que permite navegar por las marcas del documento gracias a la propiedad location.hash.

Ejemplo:

```
<HTML>
<HEAD>
<TITLE>Location.hash Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function goNextAnchor(Where){
   window.location.hash=where
}
</SCRIPT>
</HEAD>
<BODY>
<A NAME="start"><H1>Top</H1></A>
<FORM>
<INPUT TYPE="button" NAME="next" VALUE="NEXT" onClick="goNextAnchor('sec1')">
</FORM>
<HR>
<A NAME="sec1"><H1>Section 1</H1></A>
<FORM>
<INPUT TYPE="button" NAME="next" VALUE="NEXT" onClick="goNextAnchor('sec2')">
</FORM>
<HR>
<A NAME="sec2"><H1>Section 2</H1></A>
<FORM>
```

```
<INPUT TYPE="button" NAME="next" VALUE="NEXT" onClick="goNextAnchor('sec3')">
</FORM>
<HR>
<A NAME="sec3"><H1>Section 3</H1></A>
<FORM>
<INPUT TYPE="button" NAME="next" VALUE="BACK TO TOP"
onClick="goNextAnchor('start')">
</FORM>
</BODY>
</HTML>
```

host

Valor:cadena de caracteres
Obtenible:sí
Parametrizable:sí

La propiedad location.host describe tanto el nombre del servidor como el puerto de la URL. El puerto se incluye en el valor sólo cuando el puerto es una parte específica de la URL. Si se utiliza esta propiedad en una URL que no muestre el número del puerto en el campo Location del explorador, la propiedad location.host devuelve el mismo valor que la propiedad location.hostname, que será descrita a continuación.
Entonces, la propiedad location.host se utiliza para conseguir el nombre del servidor; es decir, la parte del puerto de la URL de cualquier documento cargado en el explorador. Esto, evidentemente, puede ser bastante útil en la construcción de la URL de un documento al que se desea acceder.

hostname

Valor:cadena de caracteres
Obtenible:sí
Parametrizable:sí

Esta propiedad devuelve el nombre del servidor de la URL; es decir, el nombre del ordenador que está conectado a la red y que tiene el documento que vemos en el explorador. Para la mayoría de las páginas Web, en el nombre del servidor, no sólo se incluye el nombre del dominio, sino también el prefijo "www". Sin embargo, con esta propiedad no se retorna el número del puerto, si tal número se especifica en la URL.

pathname

Valor:cadena de caracteres
Obtenible:sí
Parametrizable:sí

El componente del nombre del path de una URL consiste en la estructura de un directorio referida al directorio raíz del servidor. Entonces,se puede decir que la raíz (nombre del servidor en una conexión htpp:) no pertenece al nombre del path. Si la URL es un fichero del directorio raíz, la propiedad location.pathname retorna una barra inclinada (/). Cualquier otro parámetro que comience con la barra inclinada, indica que el directorio se encuentra dentro de la raíz. En el nombre del path también se incluye el nombre del documento.

port

Valor:cadena de caracteres
Obtenible:sí
Parametrizable:sí

ALgunas de las páginas Web necesitan que se incluya el número del puerto en la URL. En la mayoría de las URL, estos números son visibles para las sites que no tienen un nombre de

dominio asignado o utilizan protocolos poco comunes. Es posible recuperar el valor con la propiedad location.port.
Si se extrae dicho valor de una URL e intenta construir otra URL con ese componente, asegúrese de incluir la coma separadora entre la dirección IP del servidor y el número del puerto.

<u>protocol</u>

Valor:cadena de caracteres
Obtenible:sí
Parametrizable:sí

El primer componente de la URL es el protocolo que va a utilizar en la comunicación establecida. Para las páginas del World Wide Web, el Protocolo para la Transferencia de Hipertextos (http) está estandarizado. En los valores de la propiedad location.protocol se incluye el nombre del protocolo y la coma separadora. Así, para las direcciones de las típicas páginas Web, la propiedad location.protocol es:

http:

Como puede verse, las típicas barras inclinadas que suelen aparecer a continuación de los dos puntos no son parte de la propiedad location.protocol. Únicamente la propiedad location.href revela las barras inclinadas que separan el nombre del protocolo de otros componentes.

Historial de objetos

Propiedades	**Métodos**	**Manejador de eventos**
lenght	back()	(ninguno)
	forward()	
	go()	

Sintaxis

Acceder al historial de las propiedades o de los métodos:

[window.] history. property | method([parameters])

Información sobre el objeto

Cuando un usuario navega por la Red, el explorador guarda una lista de las últimas URL donde ha estado. En JavaScript, el historial de objetos es el encargado de representar esta lista, exceptuando la URL actual, las URL que se encuentran en dicha lista no pueden extraerse con una cadena.
Una aplicación para este objeto y sus métodos back() o go() puede ser el dotar a los documentos HTML de un botón equivalente al de Atrás. Este botón puede estar ligado a una cadena que vea si queda algún objeto en el historial y retroceda una página.

Propiedades
<u>length</u>
Valor:cadena de caracteres
Obtenible:sí
Parametrizable:no

Esta propiedad histori.lenght es utilizada para contar los objetos del historial.

Métodos

<u>back()</u>

Devuelve:Nada

Esté método realiza a través de una cadena la misma operación que si se hace clic sobre el botón Atrás colocado en la barra de herramientas.
Esto es utilizado para mostrar una vía de escape a los usuarios de una página Web, de manera que se proporciona una facilidad de navegación a partir de esta propiedad del JavaScript.

```
<HTML>
<HEAD>
<TITLE>Way Back Machine</TITLE>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="button" VALUE="Take me back" onClick="history.back()">
</FORM>
</BODY>
</HTML>
```

go(relativeNumber|"URLOrTitleSubstring")

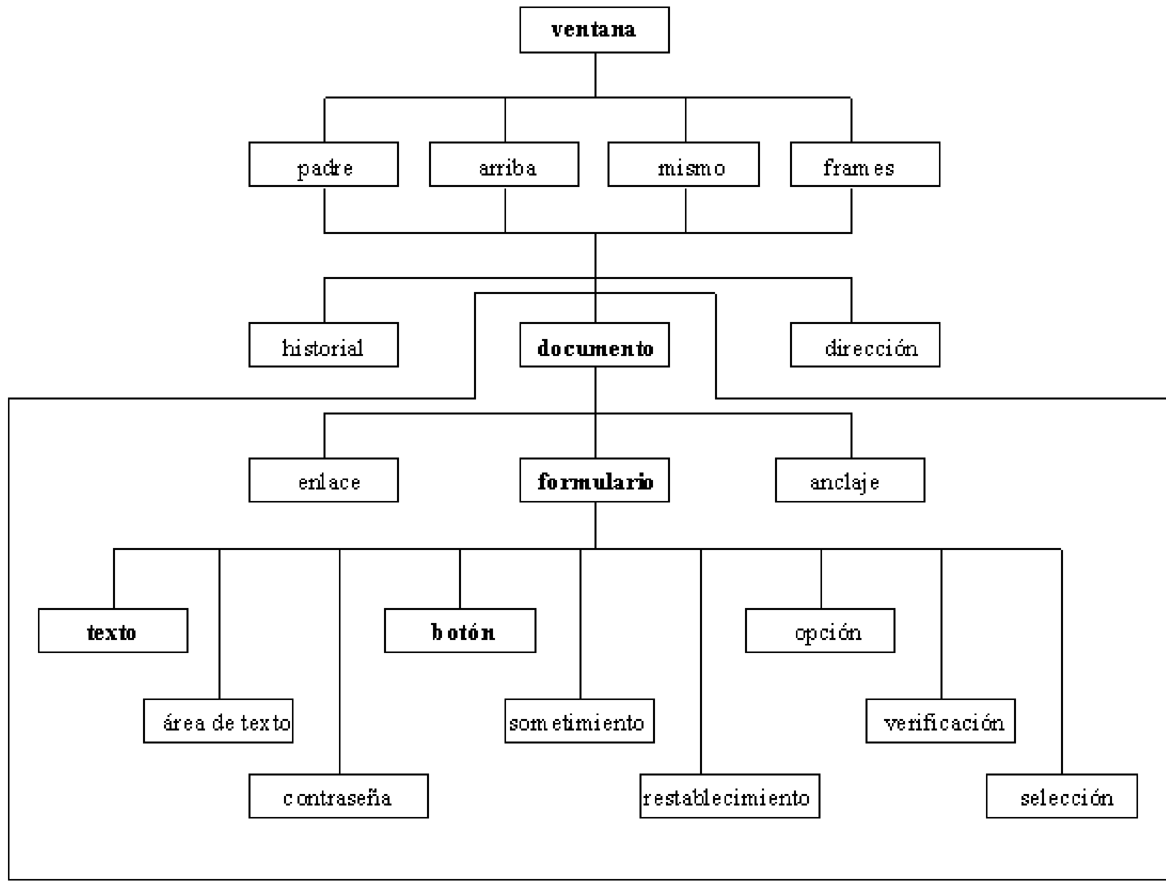
Devuelve:Nada

El método history.go() permite al usuario dirigirse donde se desee. Este comando sólo hacepta objetos que se encuentren situados en la lista del historial, y por lo tanto no puede utilizarlo en sustitución del window.location.

Capítulo 7

Objeto Documento.

La interacción con el usuario es una parte esencial del script de JavaScript, de manera que la mayor parte de la comunicación establecida entre el script y el usuario se establece a partir del objeto documento y sus elementos.
En la figura siguiente se establece, de la misma forma que en el capítulo anterior, la jerarquía del documeto objeto en JavaScript. Como se puede ver, cuanto más hacia abajo aparecen los objetos en la jerarquía, mayor serán las referencias utilizadas para acceder a ellos, sobre todo si se utiliza frames múltiples.Estas referencias facilitarán las direcciones para que JavaScript localice un objeto en particular, una propiedad o un método de entre los objetos almacenados en la memoria del navegador.



En este capítulo se dará una explicación de aquellos objetos etiquetados con un solo enlace, dejando para ampliaciones porteriores del curso los demás objetos. Así, las diferentes secciones que se tratan en este capítulo son las siguientes:

- El Objeto de documento
- El Objeto de formulario
- El Objeto de enlace
- El Objeto de anclaje
- El Objeto de botón

El Objeto documento.

Propiedades	**Métodos**	**Manejador de eventos**
alinkColor	clear()	(Ninguno)
anchor	close()	
bgColor	open()	
cookie	write()	
fgColor	writeln()	
forms		
lastModified		
linkColor		
links		
location		

referrer		
title		
vlinkColor		

-

Sintaxis.
Creación de un documento:

```
<BODY
 [BACKGROUND="backgroundImageURL"]
 [BGCOLOR="#backgroundColor"]
 [TEXT="foregroundColor"]
 [LINK="#unfollowedLinkColor"]
 [ALINK="#activatedLinkColor"]
 [VLINK="#followedLinkColor"]
 [onLoad="handlerTextOrFunction"]
 [onUnload="handlerTextOrFunction"]>
</BODY>
```

Acceso a las propiedades y métodos del objeto documento:

```
[window.] document.property | method([parameters])
```

Información sobre el objeto.

Un objeto de documento es la totalidad de lo que se encuentra en un área de contenido en una ventana de un navegador o en una frame de una ventana. Así, se puede decir que el documento es la combinación del contenido y los elementos de interfaz de una página que hacen que una página Web tenga sentido.
Son parte del documento el código contenido entre las etiquetas <HEAD<, <BODY> y, por su puesto, el script de las etiquetas <SCRIPT>. Al igual, también lo son las propiedades del objeto, incluida la fecha de la última modificación del documento y la URL desde el que el usuario accede al documento actual.

Propiedades

<u>forms</u>

Valor:matriz

Obtenible:sí
Parametrizable:sí

Es posible crear una referencia a un formulario de JavaScript de acuerdo con su nombre. Así, si un documento contiene la siguiente definición de formulario:

```
<FORM NAME="phoneData">
   [input item definitions] </FORM>
```

de manera que la referencia al objeto de formulario sería la siguiente:

```
document.phoneData
```

Es posible establecer los formularios como una lista numerada de formularios. Este tipo de lista se denomina matriz y consiste en una tabla con sólo una columna de datos.Cada fila de la tabla contiene una representación del correspondiente formulario del documento.

Para que JavaScript pueda determinar a cúal de las filas de la matriz quiere acceder su script, es necesario añadir un par de corchetes al nombre de la propiedad forms e insertar el número de filas entre corchetes. Este número es conocido como el índice de la matriz. Las matrices de JavaScript empiezan a numerar las filas con el cero, de manera que la referencia a la primera entrada o elementos de la matriz sería:

```
document.forms[0]
```

En este momento, se tiene disponibilidad a todas las propiedades y métodos, a partir de incluir en el código anterior el nombre correspondiente de la propiedad o método al cual se quiere acceder. Por ejemplo, la recuperación del valor introducido en un campo de texto denominado "homePhone" del segundo formulario del documento, se haría a partir de la referencia:

```
document.forms[1].homePhone.value
```

Una ventaja de esta propiedad es que es posible crear una biblioteca de script generalizables en el que se sepa como hacer un bucle con los formularios disponibles de un documento y tomar aquel que tenga una propiedad en particular. En el siguiente fragmento de script, se utliza una variable de contador del bucle (i) para que el script compruebe todos los formularios de un documento:

Ejemplo:

```
for (var i=0; i<document.forms.length; i++){
   if (document.forms[1] ...){
      statements;
   }
}
```

En este fragmento de código existe un aspecto de la propiedad document.forms a tener en consideración. Todas las matrices de JavaScript poseen una propiedad que devuelve el número de elementos de la matriz. Entonces, si la propiedad document.forms.length devuelve un valor 2, las referencias del formulario para este documento serán precisamente document.forms[0] y document.forms[1].

<u>title</u>

Valor:cadena de caracteres
Obtenible:sí
Parametrizable:no

El título del documento es el texto que aparece entre las etiquetas <TITLE>...</TITLE> en el encabecimiento de un documento HTML. El título aparece normalmente en la barra de título del navegador con una presentación de un solo frame. Solamente aparece el último documento del conjunto de frames como título de una ventana de frames múltiples, por eso, se puede disponer mediante un script, de la propiedad de un documento individual que aparezca en una frame. Por ejemplo, si hay dos frames, un script del documento que se encuentre en el frame LowerFrame podría hacer referencia a la propiedad de título del documento del otro frame de esta forma:

Ejemplo:

```
parent.UpperFrame.document.title
```

Esta propiedad no puede establecer por un script excepto cuando este construya un documento HTML completo, incluyendo las etiquetas <TITLE>.

<u>anchors</u>

Valor:matriz de objetos de anclaje

Obtenible:sí
Parametrizable:no

Los objetos de anclaje son puntos de un documento HTML que se encuentran marcados con las etiquetas <A NAME=""> y se hace referencia a ellos en los URL mediante el valor de cálculo de direccionamiento. La propiedad document.anchors entrega una matriz de anclajes indexada en un documento. Es posible utilizar las referencias de la matriz para localizar con exactitud un anclaje específico y recuperar una propiedad de anclaje.
Las matrices de anclajes empiezan a contar los índices a partir de cero: el primer anclaje de un documento tiene la referencia document.anchors[0]. Debido a que es un objeto tipo matriz, es posible hallar el número de elementos que hay en la matriz comprobando la propiedad del tamaño. Así, por ejemplo:

linkCount=documento.anchors.length

<u>links</u>

Valor:matriz de objetos de enlace
Obtenible:sí
Parametrizable:no

La propiedad document.links es bastante parecida a la propiedad document.anchors, excepto que los objetos contenidos en la matriz son objetos de enlace, elementos creados con las etiquetas <A HREF="">. Es posible utilizar las referencias de la matriz señalando un enlace concreto para la recuperación de una propiedad de enlace, como por ejemplo, la ventana de destino especificado en la definición HTML del enlace.
Las matrices de enlace empiezan a numerarse a partir de cero.el primer enlace de un documento tiene la referencia document.links[0]. Debido a que es un objeto tipo matriz, es posible hallar el número de elementos que hay en la matriz comprobando la propiedad del tamaño. Así, por ejemplo:

anchorCount=documento.links.length

<u>cookie</u>

Valor:Cadena de caracteres
Obtenible:sí
Parametrizable:no

JavaScript sólo tiene una posibilidad para almacenar la información que genera un usuario en una página HTML. El navegador de Netscape Navigator fue el primero que facilitó este tipo de almacenamiento, que en un principio se utilizaba para los programas CGI del servidor, y mediante el cual, los resultados intermedios podían ser almacenados en el ordenador personal cliente hasta que el servidor estuviese listo para aceptar toda la información de introducción. Netscape apodó este servicio como HTTP Cookies. Por lo tanto, un cookie es una cadena de información almacenada en el PC del cliente. El navegador controla totalmente el nombre del fichero y donde todos los cookies están almacenados; es decir, JavaScript no puede leer y escribir en ficheros de texto arbitrariamente -sólo puede hacerlo en el fichero cookie del navegador.
Internamente, un cookie contiene un número de informaciones aparte de la cadena de caracteres almacenada. Estas son el campo de acción del sitio del Web al que pertenece la información del cookie y la fecha de caducidad. Un script de JavaScript no puede indagar en el fichero cookie.
El nombre que se le asigne al cookie debe tener una forma parecida a la del atributo cookieName= de una etiqueta HTML. Esto facilita el manejo posterior de la información del cookie. Cuando se asignan nuevos cookies al fichero, el navegador los añade a la lista de cookies del campo de acción actual, separados por un punto y coma. Una entrada con dos cookies tiene una estructura como esta:

cookieName1=cookieData1; cookieName2=cookieData2

El script es el que se encarga de establecer la estructura del texto para el almacenamiento de informaciones complejas (como por ejemplo, los datos de varios elementos de formulario). Asímismo, el script debe analizar la información recuperada desde el cookie y extraer los datos que resultan útiles.
La especificación completa de los mecanismos de utilización del cookie de Netscape no está dentro de los objetivos de este curso, pero se puede encontrar más información acerca de este atributo en <http://home.netscape.com/newsref/std/cookie_spec.html>;

A continuación se exponen funciones básicas respecto a la utilización de cookies que pueden ser de utilidad en los scripts. Estas funciones son sencillas y permiten operar con cookies.

Ejemplo:

Para poder hacer algo con cookies se debe programar dos funciones: una que permita mandar un cookie al usuario y otra que consulte el contenido.

```
function sendCookie (name, value, expire){
   document.cookie=name+"="+escape(value)
   +((expires==null)? "": (";expires="+expires.toGMTString()))
}
```

Con esta función se manda una cookie. Como se puede ver, el valor es codificado por medio de la función escape y que la caducidad (en caso de decidir ponerla) debe ser convertida en formato GMT. Esto se hace a partir del método toGMTString() del objeto Date.

```
function readCookie(name){
   var search = name + "=";
   if (document.ccokie.lenght > 0){
      i = document.ccokie.indexOf(search);
      if (i != 1){
         i += search.lenght;           j = document.cookie.indexOf(";",i);
         if (j == -1)              j = document.cookie.lenght;
         return unescape (document.cookie.substring(i,j));
      }
   }
}
```

Se declara la variable search que contiene el nombre de la cookie que se quiere buscar más el igual que se escribe justo después de cada nombre, para que de esta forma el programa no encuentre un valor o una subcadena de otro nombre que sea igual al nombre de la cookie que se estaba buscando. Una vez hallada, se extrae la subcadena que hay entre el igual que separa el nombre y el valor y el punto y coma con que termina dicho valor.

Ejemplo: Contador Individualizado

En este ejemplo se guarda en una cookie visit el número de veces que se ha visitado una determinada página; en este caso se ha utilizado el número de veces que se ha leído este capítulo.

```
<HTML>
<HEAD>
   <SCRIPT LANGUAGE="JavaScript">
   function counter(){
      var date = new Date (2004, 12, 31)
      if (!num = referenceCookie("readChapter7")))
         num = 0;
         num ++
         sendCookie("readChapter7", num, date);
         if (num == 1)              document.write("this is the first time that you read this
```

```
chapter.");
          else{
              document.write("this is the "+num+" time that you read this chapter.");
          }
   }
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript"<
   Counter();
</SCRIPT>
</BODY>
</HTML>
```

La función definida consulta el valor de la cookie incrementándolo y, si no existe, lo pone a uno. Posteriormente, escribe en el documento el número de veces que se ha visitado el capítulo 7.

Métodos

write("string")
writeln("string")

Devuelve:valor booleano verdadero si se realiza

Estos dos métodos envían texto a un documento para que se visualice en pantalla. La única diferencia entre ellos es que document.writeln() añade un retorno de carro al final de la cadena de caracteres que envía al documento.

open("mimeType")

Devuelve:(nada)

La apertura de un documento es distinta a la apertura de una ventana. En el caso de la apertura de una ventana, se crea un nuevo objeto tanto en la pantalla como en la memoria del navegador, mientras que la apertura de un documento indica al navegador que se prepare para recibir los datos que se van a mostrar en la ventana llamada o implicada en la referencia del método document.open(). El nombre del método puede inducir a errores, ya que la propiedad document.open() no tiene nada que ver con el almacenamiento de documentos desde el servidor o el disco duro, sino que es un proceso que antecede al envío de datos a una ventana mediante el método document.write() o document.writeln().Así, se puede decir que mientras document.open() apenas abre el documento, los otros métodos envían los datos, y el método document.close() cierra el documento una vez enviado todos los datos.
Un parámetro opcional del método document.open() permite especificar la naturaleza de los datos que van a ser enviados a la ventana. Un tipo MIME es una especificación para la transferencia y representación de datos multimedia en Internet. Un tipo MIME se representa con un par de nombres de tipos de datos separados por una barra inclinada. Al especificar este tipo, se está especificando al navegador que tipo de datos está a punto de recibir para que sepa como tratarlos. En JavaScript, se tienen los siguientes tipos:

- text/html
- text/plain
- image/gif
- image/jpeg
- image/xbm
- plugIn

En caso que se omita este parámetro, se asume el más corriente por defecto, text/html. El método document.open() es opcional puesto que un método document.write() que intente escribir el documento creado quita el documento antiguo y abre uno nuevo. Se utilice o no el método document.open(), debe de asegurarse el utilizar el método document.close() después

de que se haya realizado cualquier escritura.

close()

Devuelve:(nada)

En el momento en que se abre el flujo de trazado a una ventana a través del document.open() o de uno de los métodos de escritura de documento, se debe de cerrar una vez el documento escrito.
En algunas plataformas, especialmente en las que las imágenes se trazan como parte del flujo del documento, todos o parte de los datos especificados para la ventana no se mostrarán adecuadamente hasta que se invoque el método document.close().

clear()

Devuelve:(nada)

Eliminar un documento y cerrarlo son acciones bastante diferentes. El cierre hace referencia al flujo de trazado que previamente se ha abierto a un documento. A menudo es necesario cerrar el flujo antes de que todos los datos especificados en el HTML del documento aparezcan correctamente.
La eliminación del documento hace referencia a que el HTML escrito en el documento es sacado del navegador, como lo es el modelo del objeto de ese documento. Aunque no es necesario quitar un documento antes de abrir o escribir en otro, puede especificar el método document.clear() si con ello se siente que se controla mejor el interfaz de usuario.

El Objeto Formulario.

Propiedades	**Métodos**	**Manejador de eventos**
action	submit()	onSubmit=
elements		
encoding		
method		
target		

-

Sintaxis

```
<FORM
   name="formName"
   [TARGET="windowName"]
   [ACTION="serverURL"]
   [METHOD= GET | POST]
   [ENCTYPE="MIMEType"]
   [onSubmit="handlerTextOrFunction"]>
</FORM>
```

Acceso a las propiedades y métodos del objeto de formulario:

[window.]document.formName.property|method([parameters])

[window.]document.forms[index].property|method([parameters])

Información sobre el objeto.

Los formularios y sus elementos son el acceso primario con dos direcciones entre los usuarios y los script de JavaScript. Un elemento de formulario facilita la manera en la que

un usuario puede introducir información en forma de texto o hacer una selección en conjunto predeterminado de opciones, ya sea en casillas de verificación, en un conjunto de botones de opción que se excluyen unos a otros o en la selección de una lista.
La definición de un objeto de formulario depende de como se piense utilizar la información procedente de los elementos del formulario. Si el formulario va a ser usado totalmente para los propósitos de JavaScript, los atributos ACTION=, TARGET= y METHOD= no son necesarios. Pero si su página Web va a suministrar información o a realizar peticiones o preguntas al servidor, será necesario especificar al menos los atributos ACTION= y METHOD=; al igual también que el atributo TARGET= si los datos resultantes del servidor se van a mostrar en otra ventana que no sea la que realiza la llamada. Finalmente, también debe hacerse lo mismo con el atributo ENCTYPE= si sus script de formulario dan forma a los datos ligados al servidor en un tipo MIME= que no sea un flujo ASCII.
Un documento complejo HTML puede tener múltiples objetos de formulario. Cada par de etiquetas <FORM>... </FORM> define un formulario. No ocurre nada si se reutiliza un nombre de un elemento de los formularios de un documento. Por ejemplo, si cada uno de los tres formularios tiene un grupo de botones de opción con el nombre choice, la referencia del objeto de cada botón asegura que JavaScript no los confundirá. La referencia al primer botón de cada uno de estos grupos de botones es la siguiente:

```
document.forms[0].choice[0]

document.forms[0].choice[0]

document.forms[0].choice[0]
```

Cuando un formulario o elemento de formulario contiene un manejador de sucesos que llama a una función definida en alguna parte del documento, se pueden usar un par de abreviaciones que simplifican la dirección de los objetos de la función.
La consigna de los parámetros del manejador de sucesos es this, que representa el objeto actual que contiene el atributo del manejador de sucesos. Un ejemplo de la utilización de this se presenta a continuación, en el que se comprueba el paso del objeto formulario como parámetro. El interfaz de usuario entero de este listado consiste en elementos de formulario.

Ejemplo:

```
<HTML>
<HEAD>
<TITLE> form.example</TITLE>
function processData(form){
   //process data of the forms
}
</HEAD>
<BODY>
<FORM NAME="books">
Choose your favorite editorial books:
<INPUT TYPE="radio" NAME="MathBook" VALUE="Addisson Wesley"
CHECKED="true">Addisson Wesley
<INPUT TYPE="radio" NAME="MathBook" VALUE="Paraninfo"
CHECKED="true">Paraninfo
<INPUT TYPE="radio" NAME="MathBook" VALUE="McGrawHill"
CHECKED="true">McGraw-Hill
<INPUT TYPE="radio" NAME="MathBook" VALUE="Reverte" CHECKED="true">Reverte
Enter the name of your favorite author:
<INPUT TYPE="tex" NAME="author"><P>
<INPUT TYPE="button" NAME="process" VALUE="Process Request..."
onClick="processData(this.form)">
</FORM>
</HTML>
```

Si es necesario llamar a las propiedades de los elementos de formulario para trabajar con ellas dentro de la función processData(), se puede realizar de dos formas diferentes. Una de estas formas es hacer que el manejador de sucesos onClick= simplemente llame a la función processData() y no pase ningún parámetro. Dentro de la función, todas las referencias a los objetos como los botones de opción o el campo de la canción tendrán que ser referenciadas completas como la que aparece a continuación para recuperar el valor introducido en el campo de autor:

```
document.forms[0].author.value
```

Una forma más eficaz es enviar el objeto de formulario con un parámetro con la llamada a la función. Al especificar this.form como el parámetro, lo que se dice a JavaScript es que envíe todo lo que sabe sobre el formulario desde que se llama a la función. En esta función, el objeto de formulario se asigna al nombre de una variable (form) que aparece entre paréntesis después del nombre de la función.
Una parte de la información que va con el formulario es su dirección de entre todos los objetos de JavaScript almacenados con el documento. Esto implica que mientras las órdenes se refieran a ese objeto de formulario, toda la dirección es automáticamente parte de esa referencia. De esta forma, para obtener la información del campo de autor se tiene que:

```
form.author.value
```

En el listado anterior se podría incluir un manejador de sucesos al campo de author para validar la entrada en ese campo. Entonces, se enviaría a la función el objeto de campo para que se analice. El código sería el siguiente:

```
<INPUT TYPE="text" NAME="author" onChange="verifyAuthor(this)"><P>
```

Luego, se debería crear una función para recoger esa llamada. Su estructura sería la siguiente:

```
function verifyAuthor(entryField){
    if (entryField.value !=" "){
        [statements]
    }
}
```

Propiedades.

<u>elements</u>

Valor:matriz de subobjetos
Obtenible:sí
Parametrizable:no

La propiedad elements engloba a todos los elementos de interfaz de usuario que estén definidos en un formulario. Como las otras propiedades de objeto JavaScript, la propiedad de elementos es una matriz con todos los elementos definidos dentro del documento HTML actual. Por ejemplo, si un formulario define tres elementos <INPUT>, la propiedad de elementos de ese formulario es una matriz con tres entradas, una por cada elemento. Así, si el primer elemento de un formulario es un campo de texto, y se quiere extraer una cadena de caracteres introducida en él, la referencia sería:

```
document.forms[0].elements[0].value
```

Como se puede ver, esta referencia llama a dos propiedades de matriz: una, la propiedad de formulario del documento, y posteriormente la otra, la propiedad de elementos del formulario. A continuación se da un listado de código de un script que muestra un uso práctico de la propiedad elements. En este caso, se tiene un formulario que contiene cuatro

campos y algunos otros documentos. La primera parte de la función que actúa sobre estos elementos comprueba que los cuatro campos contienen datos. Usando la notación de matriz se puede hacer un ciclo para los campo a través del índice de cada uno de ellos. En el caso que un campo no tenga ningún dato, se avisa al usuario y se emplea el mismo valor de índice en el método focus() de ese campo, para situar el puntero de texto.

Ejemplo:

```
<HTML>
<HEAD>
<TITLE>Elements Array</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function doIt(){
   for (i=0; i<=3; i++){
      if (document.forms[0].elements[i].value==" "){
         alert("Please fill out all fields.")
         document.forms[0].elements[i].focus()
         break
      }
   }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
Enter your first name:<INPUT TYPE="text" NAME="firstName"><P>
Enter your last name:<INPUT TYPE="text" NAME="lastName"><P>
Enter your address:<INPUT TYPE="text" NAME="address"><P>
Enter your city:<INPUT TYPE="text" NAME="city"><P>
<INPUT TYPE="radio" NAME="gender">Male
<INPUT TYPE="radio" NAME="gender">Female<P>
<INPUT TYPE="checkbox" NAME="retired">I am retired
</FORM>
<FORM>
<INPUT TYPE="button" NAME="act" VALUE="Verify" onClick="doIt()">
</FORM>
</BODY>
</HTML>
```

<u>method</u>

Valor:"get" o "post"
Obtenible:sí
Parametrizable:sí

La propiedad método de un formulario son los valores GET o POST asignados al atributo METHOD= en una definición <FORM>.
La propiedad método es de una importancia primordial para los documentos HTML que someten los datos de un formulario a un script CGI del servidor, ya que determina el formato utilizado en la transmisión de datos. Los detalles de esta operación no están dentro de los objetivos de este manual. En el caso que sea necesario mayor información respecto a esta propiedad es posible acudir a la documentación de HTML o CGI para conocer la configuración apropiada de este atributo en el entorno de su servidor Web.

Ejemplo:

```
formMethod=document.forms[0].method
```

<u>target</u>

Valor:cadena del nombre de la ventana
Obtenible:sí
Parametrizable:sí

Cuando un documento realiza una petición o pregunta para que un servidor la procese, es muy normal que el servidor devuelva una página HTML. El objetivo del atributo TARGET= de una definición <FORM> es especificar dónde se va a mostrar la respuesta del servidor a la petición realizada. El valor de esta propiedad de destino es el nombre de la ventana o del frame. Por ejemplo, si se define un conjunto de frames con tre frames y se les asignan los nombres Frame1, Frame2 y Frame3, será necesario facilitar uno de esos nombres como parámetro del atributo TARGET= de la definición <FORM>.

Ejemplo:

```
formTarget=document.forms[0].target
```

Métodos

<u>submit</u>

Devuelve:Nada

Para un usuario, la forma más común de enviar los datos de un formulario a un programa CGI para que se procesen es hacer clic en un botón de sometimiento cuyo comportamiento está diseñado para enviar los datos de todos los elementos de un formulario de acuerdo a las especificaciones listadas en los atributos de la definición <FORM>. Pero si lo que se desea es someter automáticamente los datos de un formulario a un servidor o el uso de un botón gráfico, el sometimiento se puede realizar con el método form.submit().
La llamada a este método es similar al clic que hace un usuario en un botón de sometimiento de un formulario. Por tanto, se puede tener una imagen en una página que sea como un botón gráfico de sometimiento. Si se asocia esa imagen con un objeto de enlace, se puede capturar el clic del ratón que se hace en esa imagen y activar una función cuyos contenidos incluyan una llamada al método submit() de un formulario.
A continuación se da un ejemplo de llamada al método form.submit() desde un botón gráfico.

Ejemplo:

```
<HTML>
<HEAD>
<TITLE>Registration Form </TITLE>
<SCRIPT LANGUAGE="JavaScript">
function doIt(){
   alert(document.forms[0].submit())
}
</SCRIPT>
</HEAD>
<BODY>
<FORM METHOD=POST ACTION="http://www.u.edu/pub/cgi-bin/register">
Enter your first name:<INPUT TYPE="text" NAME="firstName"><P>
Enter your last name:<INPUT TYPE="text" NAME="lastName"><P>
Enter your address:<INPUT TYPE="text" NAME="city"><P>
Enter your city:<INPUT TYPE="text" NAME="city"><P>
<INPUT TYPE="radio" NAME="gender">Male
<INPUT TYPE="checkbox" NAME="retired">I am retired
</FORM>
<A HREF="registration.html" onClick="doIt()">
<IMG SRC="niftySubmit.gif" BORDER=0></A>
</BODY>
</HTML>
```

Como se puede ver, la acción de someter el formulario se deja a la función llamada por el manejador de sucesos onClick= del objeto de enlace. El objeto de enlace se adjunta a un fichero gráfico con la expresión .gif que tiene la apariencia de un botón de sometimiento.En la función, el método form.submit() es parte de una referencia del primer (y único) formulario de este documento. El método sigue automáticamente las instrucciones de los atributos de la definición <FORM;>.

Manejador de sucesos

onSubmit=

Esta propiedad se utiliza para realizar validaciones de datos de la introducción de un usuario, antes de que estos sean sometidos al servidor.
Cuando se define un manejador onSubmit= como un atributo de una definición <FORM>, JavaScript envía el suceso de sometimiento al formulario justo antes de que los datos salgan disparados hacia el servidor. Por tanto, cualquier script o función que sea el parámetro del atributo onSubmit= se ejecutará antes de que los datos sean realmente sometidos. Existen muchas razones por las cuales se pueda querer activar un script justo antes de querer activar un script justo antes de someter los datos a un servidor; por ejemplo, poner un mensaje en la línea de estado sobre lo que está sucediendo. También es un lugar donde se puede situar una ventana de confirmación window.confirm(), sobre todo si la acción del formulario es enviar datos por correo electrónico.
Cualquier código que se ejecute para el manejador de sucesos onSubmit= debe dar una expresión que consista en la palabra return más un valor booleano. Si el valor booleano es verdadero, el sometimiento se ejecuta normalmente; si el valor es falso, el sometimiento no se lleva a cabo.
A continuación se presenta un ejemplo en el que el manejador de eventos onSubmit= solicita confirmación antes de enviar los datos.

Ejemplo:

```
<HTML>
<HEAD>
<TITLE>onSubmit= example </TITLE>
<SCRIPT LANGUAGE="JavaScript">
function getPermission(){
   return window.confirm("Go ahead and mail this info?")
}
</SCRIPT>
</HEAD>
<BODY>
<FORM METHOD=POST ACTION="mailto:trash@dannyg.com" onSubmit="return
getPermission()">
Enter your first name:<INPUT TYPE="text" NAME="firstName"><P>
Enter your last name:<INPUT TYPE="text" NAME="lastName"><P>
Enter your address:<INPUT TYPE="text" NAME="city"><P>
Enter your city:<INPUT TYPE="text" NAME="city"><P>
<INPUT TYPE="radio" NAME="gender">Male
<INPUT TYPE="checkbox" NAME="retired">I am retired<P>
<INPUT TYPE="submit">
</FORM>
</BODY>
</HTML>
```

Para obtener permiso antes de enviar información desde este formulario, el manejador de sucesos llama a una función que muestra un cuadro de diálogo de confirmación.
Dado que el valor devuelto por este tipo de cuadros de diálogo es de verdadero o falso, el valor devuelto se añade a la orden return del manejador de sucesos para que el formulario sepa si

se debe proceder o no con el sometimiento.

El Objeto de enlace.

Propiedades	Métodos	Manejador de eventos
links[index].target	Ninguno	onClick=
length		onMouseOver
[Propiedades de los objetos de dirección]		

Sintaxis

Creación de un objeto de enlace:

```
<A HREF="locationOrURL">
   [NAME="anchorName"]
   [TARGET="windowName"]
   [onClick="handlerTextOrFunction"]
   [onMouseOver="handlerTextOrFunction"]>
   linkDisplayTextOrImage
</A>
```

Acceso a las propiedades del objeto de enlace:
[window.] document.links[index].listName.property

Información sobre el objeto.

La definición de un enlace JavaScript es la misma que la de un HTML directo -con el añadido de dos posibles manejadores de sucesos. En un entorno de frames múltiples o de ventanas múltiples o de ventanas múltiples, es importante especificar el atributo TARGET= con el nombre de la ventana o frame en el que va a aparecer el contenido del URL (importante abreviar las referencias de la ventana:_top,_parent,_self y _blank).
Es importante tener en consideración la inclusión del manejador de sucesos onMouseOver= en la definición de enlace. El uso más habitual de este manejador de sucesos es adaptar la propiedad window.status.
En el que caso que se desee que al hacer un clic en el enlace se inicie una acción, sin tener que navegar de hecho a otro URL, se puede utilizar una técnica especial de JavaScript para dirigir el URL a una función de JavaScript. Así, si se quiere que el enlace no haga nada más que cambiar la barra de estado del manejador de sucesos onMouseOver=, se debe definir una función vacía y establecer el URL a esa función de la forma siguiente:

HREF:"javascript: doNothing()"

Si no se especifica el atributo HREF= en una etiqueta de enlace, la definición se convierte en un objeto de anclaje en vez de un objeto de enlace. El atributo opcional NAME= permite también que el objeto de enlace se comporte como un objeto de anclaje, dejando que otros enlaces naveguen directamente hacia el enlace.

Propiedades.

<u>links[index].target</u>
Valor:cadena de caracteres
Obtenible:sí
Parametrizable:no
La propiedad que es única al objeto de enlace es el destino. Este valor refleja el nombre de la ventana suministrado al atributo TARGET= en la definición del enlace. Dado que los objetos de enlace se almacenan en una matriz de un documento, sólo se puede hacer referencia a la propiedad de destino de un enlace en particular mediante una referencia de enlace indexada.

Ejemplo:

windowName=document.links[3].target

length
Valor:número entero

Obtenible:sí
Parametrizable:no

La matriz de objetos de JavaScript posee la propiedad de definición de tamaño de la matriz, que puede usarse en la construcción de bucles para todas sus entradas. Así, la propiedad de tamaño revela el número de enlaces que hay en un documento. Si no existen enlaces definidos, su valor es cero.

Ejemplo:

```
linkCount = document.links.length
```

Manejador de sucesos.

onMouseOver=

Este manejador de sucesos es utilizado para mostrar el URL en la línea de estado tal y como se define en el atributo HREF= del enlace, una vez que el puntero del ratón se situa encima de un enlace.
A continuación se presenta un ejemplo de la utilización del manejador de sucesos onMouseOver para visualizar las descripciones de los enlaces de texto.

Ejemplo:

```
<HTML>
<HEAD>
<TITLE>Mousing Over Links</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setStatus(msg){
   status = msg
}
function emulate(){
   alert("Not going there in this demo.")
}
</SCRIPT>
</HEAD>
<BODY>
JavaScript is designed on a simple <A HREF="javascript:emulate()"
onMouseOver="setStatus('View chapter six'); return true">object </A>-based paradigm. An
object is a <A HREF="javascript:emulate()" onMouseOver="setStatus('learn the chapter five')
return true">construct </A> with <A HREF="javascript:emulate()"
onMouseOver="setStatus('learn chapter six'); return true">properties </A>that are JavaScript
variables or other objects.
</BODY>
</HTML>
```

onClick=

La acción que realiza un enlace cuando un usuario hace clic en él está determinada por el atributo HREF=. Pero si se necesita ejecutar un script antes de navegador a un enlace específico, puede incluir un manejador de sucesos onClick en la definición del enlace.
Cualquier orden o función llamada por el manejador de sucesos onClick= se ejecuta antes de que la navegación tenga lugar.
Un ejemplo de utilización de este tipo de manejador de sucesos es el cambio de contenidos de los frames múltiples desde un solo enlace.

Ejemplo:

```
<A HREF="http://URLOfchoice1" target="mainFrame"
onClick="parent.miniFrame.location"='http:
//URLOfChoice2'">Change display</A>
```

Este enlace usa el manejador de sucesos onClick para establecer la propiedad de dirección de un frame llamada miniframe, mientras utiliza la funcionalidad del objeto de enlace estándar para almacenar un documento diferente en el frame llamado mainFrame.

El Objeto de anclaje.

Propiedades	Métodos	Manejador de eventos
Ninguno	Ninguna	Ninguno

Sintaxis

Creación de un objeto de anclaje:

```
<A NAME="anchorName">
  anchorDisplayTextOrImage
</A>
```

Información sobre el objeto

Cuando un documento HTML se almacena en un navegador habilitado con JavaScript, el navegador crea y mantiene una lista interna con todos los anclajes que definen el documento. De manera similar a los objetos de enlace, se hace referencia a los objetos de anclaje de acuerdo con el valor indexado dentro de la propiedad document.anchors[index]. También se puede convertir un objeto de enlace en un objeto de anclaje, simplemente añadiendo un atributo NAME= a la definición de enlace como se verá en la sección siguiente.

El Objeto de botón.

Propiedades	Métodos	Manejador de eventos
name	click()	onClick=
value		

Sintaxis

Creación de un botón:

```
<FORM>
<INPUT
TYPE="button"|"submit"|"reset"
NAME="buttonName"
VALUE="contents"
[onClick="handlerTextOrFunction"]>
</FORM>
```

Acceso a las propiedades y métodos del objeto de botón:

[window.]document.formName.buttonName.property|method([parameters])

[window.]document.formName.elements.[index].property|method([parameters])

[window.]document.forms[index].buttonName.property|method([parameters])

[window.]document.forms[index].elements[index].property|method([parameters])

Información sobre estos objetos

Los objetos de botón crean en la página elementos de interfaz de usuario con forma de botón de comando estándar, dependiendo del sistema operativo en el que se ejecute un navegador en particular. La única característica de un botón que está bajo el control del autor de una página HTML es el texto que aparece sobre el botón. Esta etiqueta de texto es el parámetro para el atributo VALUE= de la definición del botón.
El único manejador de sucesos de un botón es el que responde a un clic del usuario con el puntero del ratón encima de él: el manejador de sucesos onClick=.
Existen dos variedades del objeto de botón de JavaScript: los objetos de botón de sometimiento y de restablecimiento. A continuación se da una breve descripción de cada uno de ellos:

- El botón de sometimiento tiene relación con los objetos de formulario estudiados en la sección anterior. Este botón envía los datos del mismo objeto de formulario a un URL indicado en el atributo ACTION= de la definición <FORM> con un formato controlado por el atributo METHOD=
- El botón de restablecimiento también posee características especiales y, al igual que el documento anterior, actua sobre el objeto formulario. Un clic en este tipo de botón restaura todos los elementos de un formulario a sus valores por defecto. Esto abarca a los objetos de texto, los grupos de botones de opción, las casillas de verificación y los listados de selección.

Lo que diferencia a estos tres botones entre sí en la definición de elemento <INPUT> es el parámetro de atributo TYPE=. Para aquellos botones que no estén destinados a mandar datos a un servidor; utilice el tipo "button". Se reserva los tipos "submit" y "reset" para propósitos específicos con CGI.

Propiedades

<u>name</u>

Valor:cadena de caracteres
Obtenible:sí
Parametrizable:no

El nombre de un botón se fija en el atributo NAME= de la definición <INPUT> y no se puede modificar en el script.
En el caso de ser necesario recuperar una propiedad NAME de un manejador de función de propósitos generales llamados por múltiples botones de un documento: la función puede comprobar el nombre de un botón y realizar las ordenes necesarias para ese botón.

Ejemplo:

```
buttonName=document.forms[0].elements[3].name
   //4th; element is a button
```

<u>value</u>

Valor:cadena de caracteres
Obtenible:sí
Parametrizable:no

La etiqueta visible de un botón se determina con el atributo VALUE= del elemneto <INPUT> de la definición. La propiedad del valor revela ese texto. Las palabras "Submit" y "Reset" (sometimiento y restablecimiento) se utilizan generalmente como las etiquetas de texto de sus respectivos botones. A diferencia de los nombres de botón, el atributo

VALUE= puede constar de más de una palabra.

Ejemplo:

```
buttonLabel=document.forms[0].elements[2].value
   //3th element is a button
```

Método click()

Devuelve:nada

Un método clic() de botón reproduce, mediante el script, la acción que el usuario realiza cuando hace click en un botón, excepto que no se envía ningún suceso clic al botón para activar su manejador de sucesos onClick=. Como no hay un cambio resultante en el interfaz de botón cuando se hace clic en él, sus script apenas necesitarán llamar a este método.

Ejemplo:

```
document.forms[0].sender.click()
   //sender is the name of
   //a Submit-style button
```

Manejador de sucesos

onClick=

Prácticamente, todas las acciones del botón tienen lugar como respuesta al manejador de sucesos onClick=.
El manejador de sucesos del objeto de botón de sometimiento puede ser utilizado para realizar validaciones de última hora u otras acciones del script, antes de que el formulario sea sometido. Por ejemplo, una función de manejador de sucesos onClick= puede realizar cálculos a partir de la introducción del usuario y de las selecciones hechas en un formulario de una página, y posteriormente, almacenar los resultados en un objeto de campo oculto. Cuando este manejador finaliza, el formulario se somete al CGI del servidor.
El manejador de sucesos onClick= del objeto de botón de sometimiento es totalmente independiente del manejador de sucesos onSubmit= del formulario. Por ejemplo, una función llamada por un manejador de sucesos onClick= puede establecer una variable de señalador en un valor booleano. Entonces,el manejador de sucesos onSubmit= puede utilizar perfectamente este señalador como parte de una orden solicitada return que determine si el formulario está realmente sometido o no, o si el proceso de sometimiento no debería realizarse todavía.
La variable de señalador funciona como un intermediario independiente entre los dos manejadores de sucesos.
A continuación se muestra un ejemplo del manejador de sucesos onClick=, en el que el script tiene tres botones que cada uno de ellos llama a la misma función.

Ejemplo:

```
<HTML>
<HEAD>
<TITLE> Button Click</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function displaybook(btn){
   if (btn.value=="first"){alert("Addisson-Wesley")}
   if (btn.value=="second"){alert("Paraninfo")}
   if (btn.value=="third"){alert("McGraw-Hill")}
</SCRIPT>
</HEAD>
<BODY>
```

```
Click your favorite editor about technical books:<P>
<FORM>
<INPUT TYPE="button" VALUE="Addisson-Wesley" onClick="displaybook(this)">
<INPUT TYPE="button" VALUE="Paraninfo" onClick="displaybook(this)">
<INPUT TYPE="button" VALUE="McGraw-Hill" onClick="displaybook(this)">
</FORM>
</BODY>
</HTML>
```

En este script se demuestra como se pueden extraer las propiedades de valor o de nombre de un botón en una función con propósitos generales que hace el servicio de múltiples botones.
Como se puede ver, cada botón pasa su objeto como un parámetro a la función displaybook(). La función visualiza después los resultados en un cuadro de diálogo de aviso.

Capítulo 8

Funciones

Las funciones son unos de los bloques fundamentales del lenguaje de programación JavaScript. Una función en JavaScript es un procedimiento; un conjunto de sentencias que realiza una tarea específica. Para utilizar una función, es necesario definirla, para posteriormente poder llamarla desde cualquier parte del código fuente del programa. En este capítulo se discutirán los aspectos de la programación en JavaScript que tratan la definición y la llamada a funciones en el programa.

- Definición de Funciones
- LLamada a funciones
- Argumentos tipo Array
- Funciones Predefinidas

Definición de Funciones

Como se ha mencionado anteriormente, una función es un conjunto de instrucciones múltiples cuya ejecución es diferida hasta la llamada.
La definición de una función empieza con la palabra clave function, seguida por el nombre de la función y una lista de parámetros encerrados entre paréntesis y separados por comas. Las sentencias contenidas en la función deben estar encerradas entre corchetes ({}). Las sentencias pueden incluir llamadas a otras funciones definidas para otras aplicaciones dentro del programa. La sintaxis para definir una función se da a continuación:

```
function nameFunction(argument1,argument2,...){
   statements;
}
```

Generalmente, se define la función en la cabecera (HEAD) del documento. De esta forma, en el instante en que se carga la página, la función es la primera parte del código JavaScript que se ejecuta. Por ejemplo, el siguiente código define una función que realiza el cuadrado de un número.

Ejemplo

```
function square (number){
   return number*number
}
```

La función square toma el argumento denominado number, a partir del cual se realizan los cálculos. Entonces, esta función consiste en una única sentencia que retorna el argumento de la función multiplicado por el mismo. La sentencia return especifica que valor es devuelto por la función.

```
return number*number
```

Cuando se llama a la función, los parámetros son pasados a esta por valor, de manera que si el valor del parámetro cambia, no se refleja en el programa o en la llamada a la función. A diferencia de lo anterior, en el caso que se pase un objeto como parámetro a una función y se cambian las propiedades del objeto, este cambio será visible fuera de la función. Este concepto se puede ver más claro a partir del ejemplo siguiente:

Ejemplo:

```
function myFunction (theObject){
   theObject.leader="McGrawHill"
}
mybook={leader:"AddissonWesley", title:"Vectorial Math", year:"1994"}
x=mybook.leader       //retorna AddissonWesley
myFunction(mybook)  //pasa el objeto mybook a la función
y=mybook.leader       //retorna McGrawHill
```

LLamada a Funciones

En la sección anterior hemos estudiado la forma de definir una función, así como también las diferentes partes y comandos implicados en esta tarea .Ahora bien, la definición de una función simplemente da un nombre a esta, así como especifica cuales serán las operaciones que debe de realizar; es decir, no implica su ejecución.
La ejecución de una función se realiza a partir de su llamada, indicando el nombre de la función y, entre paréntesis, el valor de los parámetros a evaluar.Por ejemplo, si anteriormente se ha definido la función square, es posible realizar la llamada a esta función de la forma siguiente:

```
square(5)
```

de manera que se realiza una llamada a la función con un argumento igual a cinco. Entonces, la función ejecutará cada una de las sentencias asociadas devolviendo un valor de veinticinco. Los argumentos de una función no están limitados a datos de tipo cadena o numéricos si no que es posible pasar objetos como argumentos . En capítulos posteriores se estudiarán casos de este tipo.
Al igual que en la mayoría de lenguajes de programación, es posible definir en JavaScript funciones recursivas; esto es, una función que se llama a ella misma. A continuación, se presenta como ejemplo el cálculo del factorial de un número, un caso típico de utilización de una función recursiva.

Ejemplo

```
function factorial(n){
   if ((n==0)||(n==1))
       return 1
   else{
       result=(n*factorial(n-1))
   return result
   }
}
```

Argumentos tipo Array

En esta sección se estudia la forma de introducir en una función los parámetros a partir de un array de elementos. Dentro de una función, es posible referirse a los parámetros de la forma siguiente:

arguments[i]
functionName.arguments[i]

donde i es el número ordinal de el argumento, empezando en cero. Así, el primer argumento que se pasa a la función es arguments[0]. El número total de parámetros se obtiene a partir de la expresión arguments.length.
Este tipo de argumentos es utilizado en el caso que sea necesario pasar un número elevado de variables a la función definida. Entonces, es posible utilizar la expresión arguments.length para determinar el número de argumentos que pasan a la función, así como también referirse a cada uno de ellos a partir del array arguments.
Por ejemplo, supongamos una función que concatena varias cadenas de caracteres. El parámetro que se pasa por valor a la función es una cadena que contiene una serie de caracteres separados por comas. La función se define de la forma siguiente:

```
function myConcat(separator){
   result="" //inicializar lista

   for (var i=1; i < arguments.length; i++)
       result += arguments[i] + separator
   }
   return result
}
```

A partir de la función definida es posible crear una lista usando cada argumento del array:
//retorna "red,orange,blue,"
myConcat(",","red","orange","blue")

Funciones Predefinidas.

El lenguaje de programación JavaScript posee las siguientes funciones predefinidas:

- Función eval
- Función isFinite
- Función isNaN
- Función parseInt and parseFloat
- Función Number y String
- Funcion escape y unescape

Las siguientes secciones introducen estas funciones.

Función eval

La función eval ejecuta la expresión o sentencia contenida en la cadena que recibe como parámetro.La sintaxis de esta expresión se define de la forma siguiente:

eval(expr)

donde expr es la cadena a ser evaluada.
Si la cadena representa una expresión, la sentencia eval evaluará la expresión. Si el argumento representa una o más sentencias, eval las ejecutará. No se debe utilizar eval para evaluar una expresión aritmética, ya que JavaScript evalua estas expresiones de forma automática. A continuación se representa un ejemplo de la utilización de esta sentencia.

Ejemplo

```
message="Hi";
eval("alert(+message+"!");");
```

Función isFinite.

La función isFinite evalua un argumento para determinar si es un número finito y válido.
La sintaxis de isFinite es la siguiente:

```
isFinite(number)
```

donde el argumento number representa el número a evaluar.
Si el argumento es NaN, este método retorna el valor de false, mientras que si no retorna true.
A continuación se da un ejemplo de este tipo de sentencia. En este caso se utiliza la instrucción isFinite para determinar si el número de cuenta de un cliente es un número finito, de manera que no se producen errores en la entrada de datos:

Ejemplo

```
if (isFinite(Clientnumber)==ture){
    statements  /*take specific steps*/
}
```

Función isNaN

La función isNaN evalua un argumento para determinar si es NaN, de manera que retorna el valor true. La sintaxis de esta función es la siguiente:

```
isNaN(testValue)
```

donde testValue es el valor que debe ser evaluado. Las funciones parseFloat y parseInt, que serán tratadas posteriormente en este mismo capítulo retornan un valor NaN cuando evaluan un valor que no es un número. El siguiente código evalua la variable floatValue para determinar si es un número de coma flotante a partir de las funciones anteriormente mencionadas:

Ejemplo:

```
floatValue=parseFloat(toFloat)
if (isNaN(floatValue)){
   notFloat()
}else{
    isFloat()
}
```

Función parseInt y Float.
Las funciones paresInt y ParseFloat retornan un valor numérico cuando el argumento es una cadena.
La sintaxis de la función parseInt es la siguiente:

```
parseFloat(str)
```

En el caso que la cadena no pueda ser convertida se devuelve el valor NaN.
La función parseFloat convierte en un número entero la cadena que recibe, asumiendo que está en la base indicada. Si se obvia esté parámetro, se asume que se está trabajando en base 10. La sintaxis de esta función es la siguiente:

```
parseInt(str[,radix])
```

Al igual que en el caso anterior, si la cadena no puede ser convertida se devuelve un valor NaN.

Función Number y String

Las funciones Number y String convierten un objeto a un dato tipo cadena o numérico. La sintaxis de estas funciones es:
Number(objRef)

String(objRef)

donde objRef es una referencia al objeto.
A continuación se da un ejemplo de la utilización de este tipo de sentencia. Este fragmento de código convierte el objeto Date en una cadena de caracteres:

Ejemplo

```
D=new Date (430054663215)
//retorna la siguiente expresión
//"Thu Aug 18 04:37:43 GNT-0700 (Pacific Daylight Time) 1983"
x=String(D)
```

Función escape y unescape.

Las funciones escape y unescape son utilizadas para codificar y decodificar cadenas. La función escape retorna la codificación hexadecimal de un argumento codificado a partir de la codificación ISO latin. De otra forma, la función unescape retorna el carácter en código ASCII para el valor hexadecimal codificado.
La sintaxis de estas funciones son:

escape(string)
unescape(string)

Capítulo 9

Estructuras de Control.

Este capítulo contiene las siguientes secciones, a partir de las cuales se da una explicación de cada una de las sentencias utilizadas en el lenguaje JavaScript:

- Sentencias Condicionales: if...else y switch
- Sentencias en Bucle: for, while, do while, label, break y continue.
- Sentencias de manejo de objetos: for...in y width.
- Comentarios.

Sentencias Condicionales.

Una sentencia condicional es un conjunto de comandos que se ejecutan si la condición evaluada es verdadera. El lenguaje de programación JavaScript soporta dos tipos de sentencias: if...else y switch.

Sentencia If...else.

La sentencia if se define como una bifurcación condicional que permite la ejecución de una parte del programa u otra en función del resultado obtenido al evaluar una condición. Entonces, se puede decir que la sentencia if ejecutará un comando si la evaluación de la condición retorna un valor true, mientras que en el caso que se retorne un valor false es posible la utilización de la clausula else para ejecutar otro comando. La estructura de esta sentencia se da a continuación:

```
if (condition){
    statements
}
[else{
    statements
}]
```

La condición puede ser cualquier expresión de JavaScript que retorne un valor true o false. La sentencia que será ejecutada puede ser cualquier expresión de JavaScript, incluso nuevas sentencias tipo if, desarrollando así un código fuente de sentencias if...else anidadas.

Ejemplo:

```
function books(leader){
   if(leader=="McGrawHill"){
        document.write("The leader that you search is"+leader+".")
       }else{
           if(leader=="AddissonWesley"){
                document.write("The leader that you search is"+leader+".")
               }else{
                   document.write("The leader that you search is"+leader+".")
               }
   }
}
```

Sentencia switch.

La sentencia switch permite al programa evaluar una expresión y compararla con cada una de las etiquetas asociadas a la sentencia switch. Si se encuentra una igualdad, el programa pasará a evaluar la expresión asociada con esa etiqueta. La sentencia switch se define de la forma siguiente.

```
switch (expression){
       case label:
           statement;
           break;
        case label2:
           statement;
           break;
       ...
       default: statement;
}
```

El programa primero evalua la expresión contenida entre paréntesis, para posteriormente compararla con cada uno de los valores de las etiquetas. En el caso que coincidan los valores, se ejecutará la sentencia asociada a la etiqueta. Si no se produce ninguna coincidencia en los valores, el programa ejecutará la sentencia por defecto *(default: sentencia3)* y, si existe, se ejecuta la instrucción asociada a ésta.
La sentencia break asociada a cada una de las etiquetas asegura que el programa finalizará su ejecución una vez que se haya ejecutado una de las sentencias. Si se omite esta sentencia, el programa continuará su ejecución de forma habitual.

Ejemplo:

```
switch (expr){
   case "McGrawHill":
        document.write("The leader that you search is McGraw-Hill");
        break;
   case "AddissonWesley":
```

```
        document.write("The leader that you search is Addisson Wesley");
        break;
    case "Paraninfo";
        document.write("The leader that you search is Paraninfo");
        break;
    default:
        document.write("The leader that you search is Marcombo Boixareu");
}
```

Sentencias en Bucle.

Un bucle es un conjunto de comandos que se ejecuta de forma continuada hasta que se cumple una condición. El lenguaje JavaScript soporta las estructuras:do, do..while, for, y label. También, es posible utilizar las sentencias break y continue con este tipo de estructuras.

Sentencia for.

Un bucle de tipo for se ejecuta hasta que una determinada condición se evalua como falsa. La estructura de esta sentencia se da a continuación:

```
for ([initialExpression];[condition];[incrementExpression])
{
    statements
}
```

La ejecución de una sentencia for se da de la forma siguiente:

- La expresión de inicialización(expresión inicial) se ejecuta. Esta expresión suele inicializar uno o más contadores del bucle, aunque la sintaxis utilizada permite cualquier tipo de complejidad.
- Evaluación de la condición del bucle. Si el valor de la condición es verdadera, se ejecuta las sentencias contenidas en el programa; en caso contrario el bucle finaliza.
- Ejecución de las sentencias.
- Evaluación de la expresión de incremento, de manera que posteriormente se retorna al paso dos.

Ejemplo:

```
for (var i=minValue, i<= maxValue, i++){
    statements;
{
```

Sentencia do...while.

La sentencia do...while se repite hasta que una determinada condición se evalua como falsa. La estructura de esta sentencia se da a continuación:

```
do{
    statements
} while (condition)
```

El programa se ejecuta una vez antes de que se compruebe la condición de finalización. Si la condición retorna el valor verdadero, la sentencia se ejecuta otra vez. Al final de cada ejecución del bucle, se comprueba la condición, de manera que si ésta es falsa, finaliza la ejecución del programa.

Ejemplo:

```
do{
```

```
  i+=1;
  document.write(i);
}while (i<5);
```

Sentencia while.

La sentencia while se ejecuta siempre y cuando la condición de finalización retorne el valor de verdadero. La estructura de esta sentencia es la siguiente:

```
while (condition) {
  statement
}
```

La condición de finalización se evalua antes de que el programa sea ejecutado. En el caso que la condición retorne el valor de verdadero, las sentencias contenidas en el bucle se ejecutan de forma secuencial, mientras que si se retorna el valor de falso el programa finaliza su ejecución.

Ejemplo:

```
var i=0;
while (i==5){
  document.write("This is de bucle nomber"+i+".");
  i++;
}
```

Sentencia label.

La sentencia label es utilizada como un identificador al cual se puede hacer referencia desde cualquier parte del programa. De esta forma, se puede hacer referencia a un bucle a partir de una etiqueta, así como también utilizar la sentencia break o continue para indicar cuando el programa debe ser interrumpido o continuar su ejecución.
La sintaxis de la sentencia label se da a continuación:

```
label :{
  statements
}
```

El valor de la sentencia label puede ser cualquier identificador que no sea utilizado como palabra reservada en JavaScript. La sentencia que se identifica con label puede ser de cualquier tipo.

Ejemplo:

```
markloop
  while (theMark==true){
    statements;
}
```

Sentencia break.

La sentencia break ordena a JavaScript que finalice la ejecución de un bucle, una sentencia switch o label.
Algunos bucles finalizan su trabajo cuando se encuentran con cierta condición, en ese momento ya no hay necesidad de continuar el bucle con el resto de valores que le restan al contador. Un caso típico es aquel en el que se establece un bucle para un array completo en busca de una única entrada que cumpla cierto criterio. El criterio es habitual establecerlo con una construcción if dentro del bucle. En el caso que se cumpla esta condición, se interrumpe la ejecución del bucle, de manera que el script saldrá del flujo principal. Para completar esta salida del bucle, se utiliza la orden break. A continuación se establece un ejemplo de la utilización de este tipo de sentencia:

Ejemplo:

```
for (var i=1, i<=array.length, i++){
   if(array[i].editorial==McGrawHill){
        document.write("The leader "+array[i].editorial+" have a very good technical books" );
       break;
   }
}
```

La orden break ordena a JavaScript que salga del bucle for más inmediato (en caso que existan otros bucles for anidados). La ejecución del código fuente se retome de nuevo después de la última llave del bucle. El valor de la variable i mantiene el valor que tenía cuando se produjo la interrupción, de modo que se pueda utilizar esta variable posteriormente en el mismo script para, por ejemplo, acceder a este elemento de la matriz.

Sentencia continue

La sentencia continue es utilizada para saltar la ejecución de las sentencias anidadas en una estructura de control para una condición dada. Dentro de ese tipo de estructuras a las cuales se les puede aplicar la sentencia continue se tienen las estructuras en bucle, switch o label.
Al igual que para la sentencia break, un caso común suele ser aquel en el que se saltan las instrucciones a realizar para una condición determinada del bucle. Esto es, según se va realizando el bucle ciclo a ciclo, ejecutando las órdenes para cada valor del contador del bucle, puede existir un valor de ese contador para el que no se desee que se realicen las órdenes de ejecución. Para llevar a cabo esta tarea, las órdenes de ejecución necesitan incluir una condición if que detecte la presencia de ese valor. Cuando se evalua ese valor, la orden coninue implica que se salten inmediatamente el resto de órdenes, que se ejecute la orden de actualización y se vuelva al principio del bucle (saltando también la parte de la orden de condición de los parámetros del bucle). A continuación se realiza un ejemplo de utilización de la sentencia continue.

Ejemplo:

```
for (var i=1, i<=array.length, i++)
   if (array[i].editorial==AddissonWesley){
      continue;
   }
   statements;
}
```

En este ejemplo, la parte de órdenes del bucle se ejecuta para todos los elementos del array, exceptuando el referido a la editorial McGrawHill. La orden continue obliga a la ejecución a que salte la parte i++ de la estructura del bucle, incrementando el valor de i para el siguiente ciclo. En el caso de existir bucles for anidados, una orden continue afecta al bucle inmediato en el que se encuentra la construcción if.

Sentencias de manejo de objetos.

El lenguaje de programación JavaScript dispone de dos estructuras de manipulación de objetos: la sentencia for...in y la estructura width.

Sentencia for...in

La sentencia for...in es utilizada para recorrer todas las propiedades de un objeto. Para cada propiedad distinta, se ejecuta una iteración de la estructura for...in, de manera que se opera sobre cada una de las propiedades del objeto definido. La estructura de esta sentencia se expone a continuación:

```
for (var in object){
          statements
```

```
}
```

El parámetro object hace referencia al objeto en sí mismo, y no al nombre de cadena del objeto. JavaScript entrega un objeto si se le facilita el nombre del objeto como una cadena sin comillas, tales como window o document. Al usar la variable var, se puede crear un script que extraiga y muestre un conjunto de propiedades para un objeto dado.
A continuación se da un ejemplo de una función que se utiliza para la creación y depuración de errores en el diseño de sus páginas Web realizadas a partir de JavaScript.

Ejemplo:

```
function ShowProps (obj, ObjName){
   var result=""
   for(var i in obj){          result+ = objName+ "." +i + "=" + obj[i] + "\n"
   }
alert(result);
}
```

Se puede llamar a esta función desde cualquier parte del código fuente del programa, y pasa la referencia del objeto y una cadena, para ayudar a identificar el objeto cuando los resultados aparecen en un cuadro de diálogo de aviso.

Sentencia with

La sentencia with permite introducir un número de sentencias avisando a JavaScript sobre los tipos de objetos que están contenidos en el código fuente, de manera que no es necesario la utilización de direcciones completas y formales para acceder a las propiedades de un objeto.
La definición formal de la sintaxis de la orden with es la siguiente:

```
with (object) {
        statements
}
```

A continuación se expone un ejemplo de este tipo de sentencia. En este caso, se utiliza esta estructura para realizar llamadas a diferentes propiedades de un mismo objeto sin ser necesario que este sea parte de la referencia a estas propiedades.

Ejemplo:

```
function seeSection (book){
   newSection=(book.leaderList.chapter[book.leaderList.selectedSection].technical);
   return newSection;
}
```

A partir de la utlización de la sentencia with, es posible acortar la orden, accediendo a las propiedades del objeto de la forma siguiente:

```
function seeSection (book){
   with(book.leaderList){
      newSection(chapter[selectedSection].technical);
   }
   return newSection }
```

Comentarios.

Un comentario es una parte del programa que el interprete ignora, y que es utilizado por el programador para explicar el funcionamiento del programa. En JavaScript existen dos tipos de comentarios:

- Comentarios de una única línea, a partir de la utilización de la doble barra inclinada.
- Comentarios de varias líneas. Este tipo de comentario empieza por /* y terminan por */.