# 

TEMAS:

* Scope
* Objetos y Clases
* Módulos
* Procesos Asíncronos y Síncronos
* Peticiones HTTP

OBJETIVO:

MARCO TEÓRICO

## JavaScript

**Autor: Javier Eguiluz**

**FUNCIONES**

Cuando se desarrolla una aplicación compleja, es muy habitual utilizar una y otra vez las mismas instrucciones. Un script para una tienda de comercio electrónico por ejemplo, tiene que calcular el precio total de los productos varias veces, para añadir los impuestos y los gastos de envío.

Cuando una serie de instrucciones se repiten una y otra vez, se complica demasiado el código fuente de la aplicación, ya que:

* El código de la aplicación es mucho más largo porque muchas instrucciones están repetidas.
* Si se quiere modificar alguna de las instrucciones repetidas, se deben hacer tantas modificaciones como veces se haya escrito esa instrucción, lo que se convierte en un trabajo muy pesado y muy propenso a cometer errores.

Las funciones son la solución a todos estos problemas, tanto en JavaScript como en el resto de lenguajes de programación. Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente.

En el siguiente ejemplo, las instrucciones que suman los dos números y muestran un mensaje con el resultado se repiten una y otra vez:

var resultado;  
   
var numero1 = 3;  
var numero2 = 5;  
   
// Se suman los números y se muestra el resultado  
resultado = numero1 + numero2;  
alert("El resultado es " + resultado);  
   
numero1 = 10;  
numero2 = 7;  
   
// Se suman los números y se muestra el resultado  
resultado = numero1 + numero2;  
alert("El resultado es " + resultado);  
   
numero1 = 5;  
numero2 = 8;  
   
// Se suman los números y se muestra el resultado  
resultado = numero1 + numero2;  
alert("El resultado es " + resultado);  
…

Aunque es un ejemplo muy sencillo, parece evidente que repetir las mismas instrucciones a lo largo de todo el código no es algo recomendable. La solución que proponen las funciones consiste en extraer las instrucciones que se repiten y sustituirlas por una instrucción del tipo "en este punto, se ejecutan las instrucciones que se han extraído":

var resultado;  
   
var numero1 = 3;  
var numero2 = 5;  
   
/\* En este punto, se llama a la función que suma  
 2 números y muestra el resultado \*/  
   
numero1 = 10;  
numero2 = 7;  
   
/\* En este punto, se llama a la función que suma  
 2 números y muestra el resultado \*/  
   
numero1 = 5;  
numero2 = 8;  
   
/\* En este punto, se llama a la función que suma  
 2 números y muestra el resultado \*/  
...

Para que la solución del ejemplo anterior sea válida, las instrucciones comunes se tienen que agrupar en una función a la que se le puedan indicar los números que debe sumar antes de mostrar el mensaje.

Por lo tanto, en primer lugar se debe crear la función básica con las instrucciones comunes. Las funciones en JavaScript se definen mediante la palabra reservada function, seguida del nombre de la función. Su definición formal es la siguiente:

function nombre\_funcion() {  
 ...  
}

El nombre de la función se utiliza para llamar a esa función cuando sea necesario. El concepto es el mismo que con las variables, a las que se les asigna un nombre único para poder utilizarlas dentro del código. Después del nombre de la función, se incluyen dos paréntesis cuyo significado se detalla más adelante. Por último, los símbolos { y } se utilizan para encerrar todas las instrucciones que pertenecen a la función (de forma similar a como se encierran las instrucciones en las estructuras if o for).

Volviendo al ejemplo anterior, se crea una función llamada suma\_y\_muestra de la siguiente forma:

function suma\_y\_muestra() {  
 resultado = numero1 + numero2;  
 alert("El resultado es " + resultado);  
}

Aunque la función anterior está correctamente creada, no funciona como debería ya que le faltan los "argumentos", que se explican en la siguiente sección. Una vez creada la función, desde cualquier punto del código se puede llamar a la función para que se ejecuten sus instrucciones (además de "llamar a la función", también se suele utilizar la expresión "invocar a la función").

La llamada a la función se realiza simplemente indicando su nombre, incluyendo los paréntesis del final y el carácter ; para terminar la instrucción:

function suma\_y\_muestra() {  
 resultado = numero1 + numero2;  
 alert("El resultado es " + resultado);  
}  
   
var resultado;  
   
var numero1 = 3;  
var numero2 = 5;  
   
suma\_y\_muestra();  
   
numero1 = 10;  
numero2 = 7;  
   
suma\_y\_muestra();  
   
numero1 = 5;  
numero2 = 8;  
   
suma\_y\_muestra();  
…

El código del ejemplo anterior es mucho más eficiente que el primer código que se mostró, ya que no existen instrucciones repetidas. Las instrucciones que suman y muestran mensajes se han agrupado bajo una función, lo que permite ejecutarlas en cualquier punto del programa simplemente indicando el nombre de la función.

Lo único que le falta al ejemplo anterior para funcionar correctamente es poder indicar a la función los números que debe sumar. Cuando se necesitan pasar datos a una función, se utilizan los "argumentos", como se explica en la siguiente sección.

**Argumentos y valores de retorno**

Las funciones más sencillas no necesitan ninguna información para producir sus resultados. Sin embargo, la mayoría de funciones de las aplicaciones reales deben acceder al valor de algunas variables para producir sus resultados.

Las variables que necesitan las funciones se llaman argumentos. Antes de que pueda utilizarlos, la función debe indicar cuántos argumentos necesita y cuál es el nombre de cada argumento. Además, al invocar la función, se deben incluir los valores que se le van a pasar a la función. Los argumentos se indican dentro de los paréntesis que van detrás del nombre de la función y se separan con una coma (,).

Siguiendo el ejemplo anterior, la función debe indicar que necesita dos argumentos, correspondientes a los dos números que tiene que sumar:

function suma\_y\_muestra(primerNumero, segundoNumero) { ... }

A continuación, para utilizar el valor de los argumentos dentro de la función, se debe emplear el mismo nombre con el que se definieron los argumentos:

function suma\_y\_muestra(primerNumero, segundoNumero) { ... }  
 var resultado = primerNumero + segundoNumero;  
 alert("El resultado es " + resultado);  
}

Dentro de la función, el valor de la variable primerNumero será igual al primer valor que se le pase a la función y el valor de la variable segundoNumero será igual al segundo valor que se le pasa. Para pasar valores a la función, se incluyen dentro de los paréntesis utilizados al llamar a la función:

// Definición de la función  
function suma\_y\_muestra(primerNumero, segundoNumero) { ... }  
 var resultado = primerNumero + segundoNumero;  
 alert("El resultado es " + resultado);  
}  
   
// Declaración de las variables  
var numero1 = 3;  
var numero2 = 5;  
   
// Llamada a la función  
suma\_y\_muestra(numero1, numero2);

En el código anterior, se debe tener en cuenta que:

* Aunque casi siempre se utilizan variables para pasar los datos a la función, se podría haber utilizado directamente el valor de esas variables: suma\_y\_muestra(3, 5);
* El número de argumentos que se pasa a una función debería ser el mismo que el número de argumentos que ha indicado la función. No obstante, JavaScript no muestra ningún error si se pasan más o menos argumentos de los necesarios.
* El orden de los argumentos es fundamental, ya que el primer dato que se indica en la llamada, será el primer valor que espera la función; el segundo valor indicado en la llamada, es el segundo valor que espera la función y así sucesivamente.
* Se puede utilizar un número ilimitado de argumentos, aunque si su número es muy grande, se complica en exceso la llamada a la función.
* No es obligatorio que coincida el nombre de los argumentos que utiliza la función y el nombre de los argumentos que se le pasan. En el ejemplo anterior, los argumentos que se pasan son numero1y numero2 y los argumentos que utiliza la función son primerNumero y segundoNumero.

A continuación se muestra otro ejemplo de una función que calcula el precio total de un producto a partir de su precio básico:

// Definición de la función  
function calculaPrecioTotal(precio) {  
 var impuestos = 1.16;  
 var gastosEnvio = 10;  
 var precioTotal = ( precio \* impuestos ) + gastosEnvio;  
}  
   
// Llamada a la función  
calculaPrecioTotal(23.34);

La función anterior toma como argumento una variable llamada precio y le suma los impuestos y los gastos de envío para obtener el precio total. Al llamar a la función, se pasa directamente el valor del precio básico mediante el número 23.34.

No obstante, el código anterior no es demasiado útil, ya que lo ideal sería que la función pudiera devolver el resultado obtenido para guardarlo en otra variable y poder seguir trabajando con este precio total:

function calculaPrecioTotal(precio) {  
 var impuestos = 1.16;  
 var gastosEnvio = 10;  
 var precioTotal = ( precio \* impuestos ) + gastosEnvio;  
}  
   
// El valor devuelto por la función, se guarda en una variable  
var precioTotal = calculaPrecioTotal(23.34);  
   
// Seguir trabajando con la variable "precioTotal"

Afortunadamente, las funciones no solamente puede recibir variables y datos, sino que también pueden devolver los valores que han calculado. Para devolver valores dentro de una función, se utiliza la palabra reservada return. Aunque las funciones pueden devolver valores de cualquier tipo, solamente pueden devolver un valor cada vez que se ejecutan.

function calculaPrecioTotal(precio) {  
 var impuestos = 1.16;  
 var gastosEnvio = 10;  
 var precioTotal = ( precio \* impuestos ) + gastosEnvio;  
 return precioTotal;  
}  
   
var precioTotal = calculaPrecioTotal(23.34);  
   
// Seguir trabajando con la variable "precioTotal"

Para que la función devuelva un valor, solamente es necesario escribir la palabra reservada returnjunto con el nombre de la variable que se quiere devolver. En el ejemplo anterior, la ejecución de la función llega a la instrucción return precioTotal; y en ese momento, devuelve el valor que contenga la variable precioTotal.

Como la función devuelve un valor, en el punto en el que se realiza la llamada, debe indicarse el nombre de una variable en el que se guarda el valor devuelto:

var precioTotal = calculaPrecioTotal(23.34);

Si no se indica el nombre de ninguna variable, JavaScript no muestra ningún error y el valor devuelto por la función simplemente se pierde y por tanto, no se utilizará en el resto del programa. En este caso, tampoco es obligatorio que el nombre de la variable devuelta por la función coincida con el nombre de la variable en la que se va a almacenar ese valor.

Si la función llega a una instrucción de tipo return, se devuelve el valor indicado y finaliza la ejecución de la función. Por tanto, todas las instrucciones que se incluyen después de un return se ignoran y por ese motivo la instrucción return suele ser la última de la mayoría de funciones.

Para que el ejemplo anterior sea más completo, se puede añadir otro argumento a la función que indique el porcentaje de impuestos que se debe añadir al precio del producto. Evidentemente, el nuevo argumento se debe añadir tanto a la definición de la función como a su llamada:

function calculaPrecioTotal(precio, porcentajeImpuestos) {  
 var gastosEnvio = 10;  
 var precioConImpuestos = (1 + porcentajeImpuestos/100) \* precio;  
 var precioTotal = precioConImpuestos + gastosEnvio;  
 return precioTotal;  
}  
   
var precioTotal = calculaPrecioTotal(23.34, 16);  
var otroPrecioTotal = calculaPrecioTotal(15.20, 4);

Para terminar de completar el ejercicio anterior, se puede rendondear a dos decimales el precio total devuelto por la función:

function calculaPrecioTotal(precio, porcentajeImpuestos) {  
 var gastosEnvio = 10;  
 var precioConImpuestos = (1 + porcentajeImpuestos/100) \* precio;  
 var precioTotal = precioConImpuestos + gastosEnvio;  
 return precioTotal.toFixed(2);  
}  
   
var precioTotal = calculaPrecioTotal(23.34, 16);

ÁMBITO DE LAS VARIABLES

El ámbito de una variable (llamado "scope" en inglés) es la zona del programa en la que se define la variable. JavaScript define dos ámbitos para las variables: global y local.

El siguiente ejemplo ilustra el comportamiento de los ámbitos:

function creaMensaje() {  
 var mensaje = “Mensaje de prueba”;  
}  
creaMensaje();  
alert(mensaje);

El ejemplo anterior define en primer lugar una función llamada creaMensaje que crea una variable llamada mensaje. A continuación, se ejecuta la función mediante la llamada creaMensaje(); y seguidamente, se muestra mediante la función alert() el valor de una variable llamada mensaje.

Sin embargo, al ejecutar el código anterior no se muestra ningún mensaje por pantalla. La razón es que la variable mensaje se ha definido dentro de la función creaMensaje() y por tanto, es una variable localque solamente está definida dentro de la función.

Cualquier instrucción que se encuentre dentro de la función puede hacer uso de esa variable, pero todas las instrucciones que se encuentren en otras funciones o fuera de cualquier función no tendrán definida la variable mensaje. De esta forma, para mostrar el mensaje en el código anterior, la función alert() debe llamarse desde dentro de la función creaMensaje():

function creaMensaje() {  
 var mensaje = “Mensaje de prueba”;  
 alert(mensaje);  
}  
creaMensaje();

Además de variables locales, también existe el concepto de variable global, que está definida en cualquier punto del programa (incluso dentro de cualquier función).

var mensaje = “Mensaje de prueba”;  
   
function muestraMensaje() {  
 alert(mensaje);  
}

El código anterior es el ejemplo inverso al mostrado anteriormente. Dentro de la función muestraMensaje() se quiere hacer uso de una variable llamada mensaje y que no ha sido definida dentro de la propia función. Sin embargo, si se ejecuta el código anterior, sí que se muestra el mensaje definido por la variable mensaje.

El motivo es que en el código JavaScript anterior, la variable mensaje se ha definido fuera de cualquier función. Este tipo de variables automáticamente se transforman en variables globales y están disponibles en cualquier punto del programa (incluso dentro de cualquier función).

De esta forma, aunque en el interior de la función no se ha definido ninguna variable llamada mensaje, la variable global creada anteriormente permite que la instrucción alert() dentro de la función muestre el mensaje correctamente.

Si una variable se declara fuera de cualquier función, automáticamente se transforma en variable global independientemente de si se define utilizando la palabra reservada var o no. Sin embargo, las variables definidas dentro de una función pueden ser globales o locales.

Si en el interior de una función, las variables se declaran mediante var se consideran locales y las variables que no se han declarado mediante var, se transforman automáticamente en variables globales.

Por lo tanto, se puede rehacer el código del primer ejemplo para que muestre el mensaje correctamente. Para ello, simplemente se debe definir la variable dentro de la función sin la palabra reservada var, para que se transforme en una variable global:

function creaMensaje() {  
 mensaje = "Mensaje de prueba";  
}  
   
creaMensaje();  
alert(mensaje);

¿Qué sucede si una función define una variable local con el mismo nombre que una variable global que ya existe? En este caso, las variables locales prevalecen sobre las globales, pero sólo dentro de la función:

var mensaje = "gana la de fuera";  
   
function muestraMensaje() {  
 var mensaje = "gana la de dentro";  
 alert(mensaje);  
}  
   
alert(mensaje);  
muestraMensaje();  
alert(mensaje);

El código anterior muestra por pantalla los siguientes mensajes:

**gana la de fuera  
gana la de dentro  
gana la de fuera**

Dentro de la función, la variable local llamada mensaje tiene más prioridad que la variable global del mismo nombre, pero solamente dentro de la función.

¿Qué sucede si dentro de una función se define una variable global con el mismo nombre que otra variable global que ya existe? En este otro caso, la variable global definida dentro de la función simplemente modifica el valor de la variable global definida anteriormente:

var mensaje = "gana la de fuera";  
function muestraMensaje() {  
 mensaje = "gana la de dentro";  
 alert(mensaje);  
}  
   
alert(mensaje);  
muestraMensaje();  
alert(mensaje);

En este caso, los mensajes mostrados son:

**gana la de fuera  
gana la de dentro  
gana la de dentro**

La recomendación general es definir como variables locales todas las variables que sean de uso exclusivo para realizar las tareas encargadas a cada función. Las variables globales se utilizan para compartir variables entre funciones de forma sencilla.

**Sentencias break y continue**

La estructura de control for es muy sencilla de utilizar, pero tiene el inconveniente de que el número de repeticiones que se realizan sólo se pueden controlar mediante las variables definidas en la zona de actualización del bucle.

Las sentencias break y continue permiten manipular el comportamiento normal de los bucles for para detener el bucle o para saltarse algunas repeticiones. Concretamente, la sentencia break permite terminar de forma abrupta un bucle y la sentencia continue permite saltarse algunas repeticiones del bucle.

El siguiente ejemplo muestra el uso de la sentencia break:

var cadena = "En un lugar de la Mancha de cuyo nombre no quiero acordarme...";

var letras = cadena.split("");

var resultado = "";

for(i in letras) {

if(letras[i] == 'a') {

break;

}

else {

resultado += letras[i];

}

}

alert(resultado);

// muestra "En un lug"

Si el programa llega a una instrucción de tipo break;, sale inmediatamente del bucle y continúa ejecutando el resto de instrucciones que se encuentran fuera del bucle for. En el ejemplo anterior, se recorren todas las letras de una cadena de texto y cuando se encuentra con la primera letra "a", se detiene la ejecución del bucle for.

La utilidad de break es terminar la ejecución del bucle cuando una variable toma un determinado valor o cuando se cumple alguna condición.

En ocasiones, lo que se desea es saltarse alguna repetición del bucle cuando se dan algunas condiciones. Siguiendo con el ejemplo anterior, ahora se desea que el texto de salida elimine todas las letras "a" de la cadena de texto original:

var cadena = "En un lugar de la Mancha de cuyo nombre no quiero acordarme...";

var letras = cadena.split("");

var resultado = "";

for(i in letras) {

if(letras[i] == 'a') {

continue;

}

else {

resultado += letras[i];

}

}

alert(resultado);

// muestra "En un lugr de l Mnch de cuyo nombre no quiero cordrme..."

En este caso, cuando se encuentra una letra "a" no se termina el bucle, sino que no se ejecutan las instrucciones de esa repetición y se pasa directamente a la siguiente repetición del bucle for.

La utilidad de continue es que permite utilizar el bucle for para filtrar los resultados en función de algunas condiciones o cuando el valor de alguna variable coincide con un valor determinado.

**DOM**

La creación del Document Object Model o DOM es una de las innovaciones que más ha influido en el desarrollo de las páginas web dinámicas y de las aplicaciones web más complejas.

DOM permite a los programadores web acceder y manipular las páginas XHTML como si fueran documentos XML. De hecho, DOM se diseñó originalmente para manipular de forma sencilla los documentos XML.

A pesar de sus orígenes, DOM se ha convertido en una utilidad disponible para la mayoría de lenguajes de programación (Java, PHP, JavaScript) y cuyas únicas diferencias se encuentran en la forma de implementarlo.

**Árbol de nodos**

Una de las tareas habituales en la programación de aplicaciones web con JavaScript consiste en la manipulación de las páginas web. De esta forma, es habitual obtener el valor almacenado por algunos elementos (por ejemplo los elementos de un formulario), crear un elemento (párrafos, <div>, etc.) de forma dinámica y añadirlo a la página, aplicar una animación a un elemento (que aparezca/desaparezca, que se desplace, etc.).

Todas estas tareas habituales son muy sencillas de realizar gracias a DOM. Sin embargo, para poder utilizar las utilidades de DOM, es necesario "transformar" la página original. Una página HTML normal no es más que una sucesión de caracteres, por lo que es un formato muy difícil de manipular. Por ello, los navegadores web transforman automáticamente todas las páginas web en una estructura más eficiente de manipular.

Esta transformación la realizan todos los navegadores de forma automática y nos permite utilizar las herramientas de DOM de forma muy sencilla. El motivo por el que se muestra el funcionamiento de esta transformación interna es que condiciona el comportamiento de DOM y por tanto, la forma en la que se manipulan las páginas.

DOM transforma todos los documentos XHTML en un conjunto de elementos llamados nodos, que están interconectados y que representan los contenidos de las páginas web y las relaciones entre ellos. Por su aspecto, la unión de todos los nodos se llama "árbol de nodos".

La siguiente página XHTML sencilla:

<html xmlns="<http://www.w3.org/1999/xhtml>">

<head>  
 <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />  
 <title>Página sencilla</title>  
 </head>  
   
 <body>  
 <p>Esta página es <strong>muy sencilla</strong></p>  
 </body>  
</html>

Se transforma en el siguiente árbol de nodos:

****

En el esquema anterior, cada rectángulo representa un nodo DOM y las flechas indican las relaciones entre nodos. Dentro de cada nodo, se ha incluido su tipo (que se verá más adelante) y su contenido.

La raíz del árbol de nodos de cualquier página XHTML siempre es la misma: un nodo de tipo especial denominado "Documento".

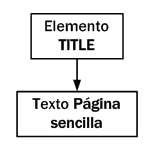
A partir de ese nodo raíz, cada etiqueta XHTML se transforma en un nodo de tipo "Elemento". La conversión de etiquetas en nodos se realiza de forma jerárquica. De esta forma, del nodo raíz solamente pueden derivar los nodos HEAD y BODY. A partir de esta derivación inicial, cada etiqueta XHTML se transforma en un nodo que deriva del nodo correspondiente a su "etiqueta padre".

La transformación de las etiquetas XHTML habituales genera dos nodos: el primero es el nodo de tipo "Elemento" (correspondiente a la propia etiqueta XHTML) y el segundo es un nodo de tipo "Texto" que contiene el texto encerrado por esa etiqueta XHTML.

Así, la siguiente etiqueta XHTML:

<title>Página sencilla</title>

Genera los siguientes dos nodos:



**Acceso directo a los nodos**

Una vez construido automáticamente el árbol completo de nodos DOM, ya es posible utilizar las funciones DOM para acceder de forma directa a cualquier nodo del árbol. Como acceder a un nodo del árbol es equivalente a acceder a "un trozo" de la página, una vez construido el árbol, ya es posible manipular de forma sencilla la página: acceder al valor de un elemento, establecer el valor de un elemento, mover un elemento de la página, crear y añadir nuevos elementos, etc.

DOM proporciona dos métodos alternativos para acceder a un nodo específico: acceso a través de sus nodos padre y acceso directo.

Las funciones que proporciona DOM para acceder a un nodo a través de sus nodos padre consisten en acceder al nodo raíz de la página y después a sus nodos hijos y a los nodos hijos de esos hijos y así sucesivamente hasta el último nodo de la rama terminada por el nodo buscado. Sin embargo, cuando se quiere acceder a un nodo específico, es mucho más rápido acceder directamente a ese nodo y no llegar hasta él descendiendo a través de todos sus nodos padre.

Por ese motivo, no se van a presentar las funciones necesarias para el acceso jerárquico de nodos y se muestran solamente las que permiten acceder de forma directa a los nodos.

Por último, es importante recordar que el acceso a los nodos, su modificación y su eliminación solamente es posible cuando el árbol DOM ha sido construido completamente, es decir, después de que la página XHTML se cargue por completo. Más adelante se verá cómo asegurar que un código JavaScript solamente se ejecute cuando el navegador ha cargado entera la página XHTML.

**getElementsByTagName()**

Como sucede con todas las funciones que proporciona DOM, la función getElementsByTagName() tiene un nombre muy largo, pero que lo hace autoexplicativo.

La función getElementsByTagName(nombreEtiqueta) obtiene todos los elementos de la página XHTML cuya etiqueta sea igual que el parámetro que se le pasa a la función.

El siguiente ejemplo muestra cómo obtener todos los párrafos de una página XHTML:

var parrafos = document.getElementsByTagName("p");

El valor que se indica delante del nombre de la función (en este caso, document) es el nodo a partir del cual se realiza la búsqueda de los elementos. En este caso, como se quieren obtener todos los párrafos de la página, se utiliza el valor document como punto de partida de la búsqueda.

El valor que devuelve la función es un array con todos los nodos que cumplen la condición de que su etiqueta coincide con el parámetro proporcionado. El valor devuelto es un array de nodos DOM, no un array de cadenas de texto o un array de objetos normales. Por lo tanto, se debe procesar cada valor del array de la forma que se muestra en las siguientes secciones.

De este modo, se puede obtener el primer párrafo de la página de la siguiente manera:

var primerParrafo = parrafos[0];

De la misma forma, se podrían recorrer todos los párrafos de la página con el siguiente código:

for(var i=0; i<parrafos.length; i++) {  
 var parrafo = parrafos[i];  
}

La función getElementsByTagName() se puede aplicar de forma recursiva sobre cada uno de los nodos devueltos por la función. En el siguiente ejemplo, se obtienen todos los enlaces del primer párrafo de la página:

var parrafos = document.getElementsByTagName("p");  
var primerParrafo = parrafos[0];  
var enlaces = primerParrafo("a");

**getElementsByName()**

La función getElementsByName() es similar a la anterior, pero en este caso se buscan los elementos cuyo atributo name sea igual al parámetro proporcionado. En el siguiente ejemplo, se obtiene directamente el único párrafo con el nombre indicado:

var parrafoEspecial = document.getElementsByName("especial");  
   
<p name="prueba">...</p>  
<p name="especial">...</p>  
<p>...</p>

Normalmente el atributo name es único para los elementos HTML que lo definen, por lo que es un método muy práctico para acceder directamente al nodo deseado. En el caso de los elementos HTML radiobutton, el atributo name es común a todos los radiobutton que están relacionados, por lo que la función devuelve una colección de elementos.

Internet Explorer 6.0 no implementa de forma correcta esta función, ya que sólo la tiene en cuenta para los elementos de tipo <input> y <img>. Además, también tiene en consideración los elementos cuyo atributo id sea igual al parámetro de la función.

**getElementById()**

La función getElementById() es la más utilizada cuando se desarrollan aplicaciones web dinámicas. Se trata de la función preferida para acceder directamente a un nodo y poder leer o modificar sus propiedades.

La función getElementById() devuelve el elemento XHTML cuyo atributo id coincide con el parámetro indicado en la función. Como el atributo id debe ser único para cada elemento de una misma página, la función devuelve únicamente el nodo deseado.

var cabecera = document.getElementById("cabecera");  
   
<div id="cabecera">  
 <a href="/" id="logo">...</a>  
</div>

La función getElementById() es tan importante y tan utilizada en todas las aplicaciones web, que casi todos los ejemplos y ejercicios que siguen la utilizan constantemente.

Internet Explorer 6.0 también interpreta incorrectamente esta función, ya que devuelve también aquellos elementos cuyo atributo name coincida con el parámetro proporcionado a la función.

Herramientas

JS, Navegador Web

