Introducción al DOM

Vicent Moncho Mas
Guillem Garcia Brustenga
Jordi Ustrell Garrigós

PID_00220487

UOC
Universitat Oberta de Catalunya

Índice

1. Nacimiento y evolución del DOM

Los modelos de objetos definen la interfaz que describe la estructura lógica de un objeto y la forma estándar que permite manipular los objetos desde el lenguaje de programación. En el caso particular de JavaScript, se suele hacer referencia a dos modelos:

- **BOM** (*browser object model*): define el acceso a las características del navegador.

BOM

Este modelo define características como la ventana, la pantalla, el historial, etc.

- **DOM** (*document object model*): define el acceso al contenido del documento que es mostrado en la ventana del navegador. De manera que contiene todos los objetos que forman parte de la página HTML.

No es sencillo delimitar la frontera entre los dos modelos anteriores, ya que aspectos que teóricamente forman parte del BOM o del DOM aparecen entremezclados y dificultan su identificación. Esto implicará que generalmente se haga referencia al DOM, incluyendo dentro de este a los objetos relacionados con el Navegador.

Por tanto, se puede definir el DOM como una **organización jerárquica** en la que un objeto depende del que se encuentra por encima, y rige a los que se encuentran en niveles inferiores. La comunicación entre objetos depende de esta jerarquía. El objeto que se encuentra en el nivel superior debe comunicarse con los del nivel inferior a través de los que se encuentran en los niveles intermedios.

A lo largo de la historia de JavaScript, han ido apareciendo hasta 4 modelos distintos de objetos:

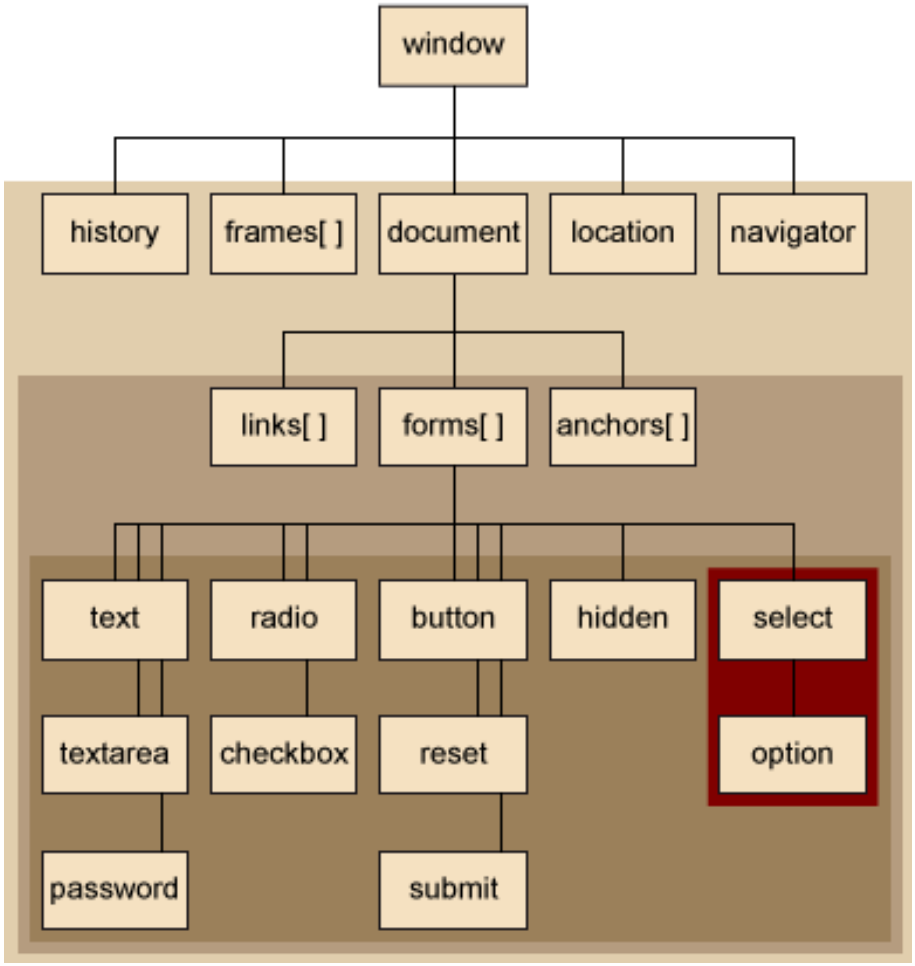
- Modelo tradicional de objetos (Netscape 2 e Internet Explorer 3).
- Modelo tradicional extendido (Netscape 3).
- Modelo de objetos de HTML dinámico (Netscape 4 e Internet Explorer 4).
- Modelo de objetos de navegador tradicional + DOM estándar (a partir de Nestcape 6 e Internet Explorer 5).

El objetivo de este primer apartado del módulo es la revisión histórica de la evolución del DOM, ya que esta visión ayudará a la mejor comprensión del DOM actual y de cómo este puede evolucionar en el futuro. No se pretende estudiar cada objeto con sus propiedades y métodos, ya que esto se planteará en el tercer apartado de esté módulo.

1.1. Modelo tradicional

Como es sabido, uno de los primeros objetivos del lenguaje JavaScript era la validación de los contenidos de los formularios. Si se tiene en cuenta lo anterior, es de esperar que el primer modelo de objetos se focalizara en la estructura de los formularios teniendo características básicas del navegador y del documento. Esto provocó que el primer modelo de objetos fuera muy similar en ambas plataformas Netscape 2 e Internet Explorer 3. La siguiente figura presenta este primer modelo:

Figura 1. Modelo de objetos tradicional (Netscape 2 e Internet Explorer 3)



En el punto más alto de la jerarquía, se encuentra el objeto Window, que representa el área de la ventana del navegador en la que aparecen los documentos HTML. El siguiente nivel en la jerarquía está formado por los siguientes objetos:

- Document: es el objeto que proporciona acceso a elementos HTML de la página como las anclas, los enlaces y los elementos de los formularios, pero además dispone de un conjunto de propiedades que definen el aspecto de la página, como el color del fondo o del texto.
- Frames[]: es un *array* que contiene los marcos de la página, de modo que cada marco, a su vez, hace referencia a un objeto Window.

- History: es un objeto que contiene la colección de direcciones URL visitadas por el usuario.
- Location: es un objeto que contiene la URL actual del navegador.
- Navigator: es un objeto que contiene las características básicas del navegador (versión, tipo, etc.).

Tal como se aprecia en la anterior figura 1, el objeto realmente importante es Document y su objeto hijo Forms[], ya que la mayoría de elementos del DOM tradicional son elementos de formulario.

Pero la simplicidad del modelo provocaba que no estuviera soportado el manejo de texto, imágenes, *applets* y objetos incrustados y que no se pudiera acceder a las propiedades de presentación de la mayoría de elementos de la página web.

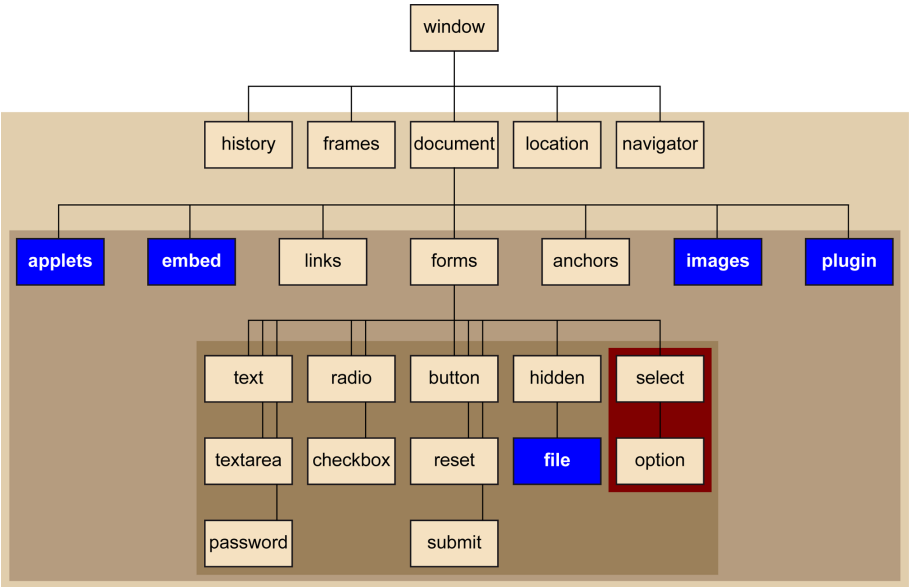
El modelo también incluía la gestión de eventos permitiéndonos de este modo ejecutar *scripts* en respuesta a ciertas acciones realizadas por el usuario en el documento HTML, pero solo a ciertos objetos.

1.2. Modelo tradicional extendido

La versión 3 de Netscape introduce un nuevo conjunto de objetos de Document, lo que proporciona capacidad de acceso a imágenes, *plugins*, *applets* y objetos incrustados en la página HTML.

La estructura y los componentes de este modelo se representan en la figura 2:

Figura 2. Modelo de objetos tradicional extendido (Netscape 3)



La novedad que tuvo más impacto fue la inclusión del *array* "images[ ]", ya que aunque la mayoría de propiedades del objeto Image era de lectura, la propiedad src, que definía la ruta de la imagen, era modificable y esto permitió cambiar dinámicamente imágenes en la página web como respuesta de eventos del usuario.

Una aplicación típica fueron los botones "rollover", que cambiaban de aspecto cuando el ratón se situaba sobre ellos.

A continuación, se muestra un ejemplo de creación de botón "rollover":

```
<a href="#"
onmouseover="document.images[0].src'/images/botonON.gif' "
onmouseout="document.images[1].src'/images/botonOFF.gif' ">
<img src="/images/botonOFF.gif">
</a>
```

Con el código anterior, cuando el puntero del ratón se sitúa sobre la imagen, el manejador de eventos "onmouseover" de la etiqueta cambia las imágenes, y cuando el puntero sale de la imagen, es el manejador "onmouseout" el que cambia de nuevo la imagen.

Aunque aparecieron más métodos y propiedades, ninguno tuvo el impacto que disfrutó el objeto Image.

1.3. Modelo de objetos de HTML dinámico

El modelo de objetos de HTML dinámico fue implementado en las versiones 4.0, tanto de Netscape Navigator como de Internet Explorer, pero cada uno de los navegadores evolucionó de manera divergente.

Esto último creó un hueco tan importante en el uso del DOM, en cada uno de los navegadores, que derivó en un serio problema para los programadores.

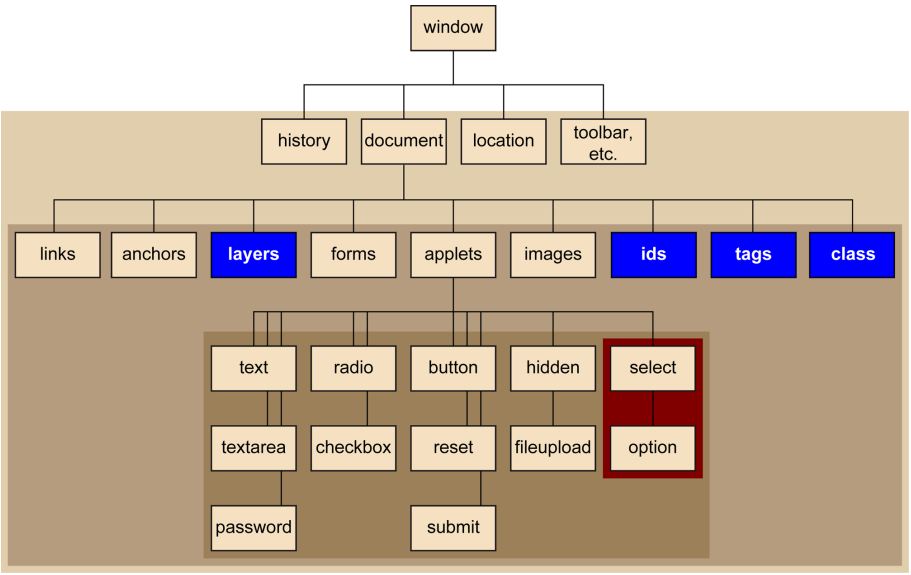
1.3.1. Netscape 4

La única animación posible hasta la incorporación de la versión 4 del navegador era la permutación de imágenes que se ha visto en el subapartado anterior; las páginas web eran completamente estáticas.

El principal cambio que aportó esta nueva versión fue el soporte para la etiqueta propietaria <layer>, algunas novedades en el modelo de gestión de eventos y el objeto Style con métodos que permitían modificar las hojas de estilo.

En la siguiente figura, se puede observar las novedades que aparecen en DOM:

Figura 3. Modelo de objetos HTML Dinámico de Netscape 4



Los nuevos objetos introducidos son:

- ids: que permite la asignación de hojas de estilo a elementos HTML con el atributo “id” establecido;
- clases: que permite la asignación de hojas de estilo a elementos HTML con el atributo “class” establecido;
- tags: que permite la asignación de hojas de estilo a elementos HTML libres, y
- layers[]: array de capas del documento.

Los tres primeros objetos permitieron que los programadores pudieran crear o manejar atributos CSS para todos los elementos del documento. Aunque fue una característica muy potente, tenía una limitación que se basaba en que solo era posible crear o modificar elementos de estilo antes de que mostrara en pantalla el contenido al que se aplicaba, ya que no era posible modificar estilos una vez que los objetos ya habían sido cargados en la página web.

La introducción de la etiqueta <layer> permitió definir áreas de contenido que se podían colocar en una posición exacta, moverlas, solaparlas y ocultarlas. Se podía escribir contenido nuevo en una capa con el método write() y se esperaba que este fuera una de las bases de las aplicaciones dinámicas, pero como se trataba de una etiqueta propietaria de Netscape que no fue incorporada al estándar HTML de W3C, no fue implementada por ningún navegador de la competencia.

También se añadió una cantidad de propiedades nuevas al objeto Window, que proporcionaban información sobre el tamaño de la pantalla, la altura y la anchura, configuración de las barras de herramienta y un conjunto nuevo de métodos.

En cuanto al modelo de eventos:

- aparecieron nuevos manejadores de eventos de ratón y de teclado como *onmouseup/down* y *onkeyup/down*;

- apareció un nuevo modelo de gestión de eventos que hace que los eventos realicen un recorrido desde los objetos de la parte superior de la jerarquía y desciendan hasta llegar al objeto en el que se ha creado el evento.

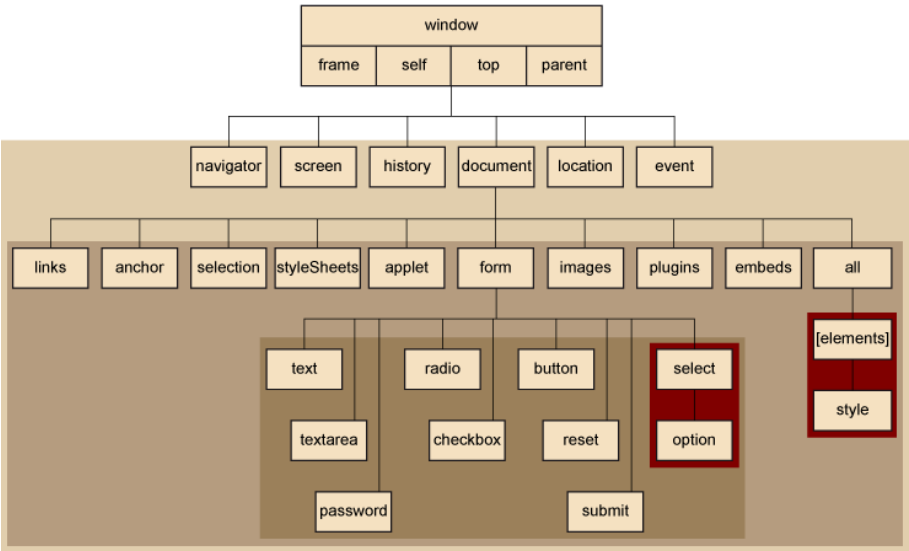
La captura de los eventos se podía realizar con el método "captureEvents()", que permite tratar al más alto nivel un evento capturado. Esta técnica podía simplificar la programación de páginas en las que un número elevado de objetos de bajo nivel, como los campos de formulario, requiere un mismo tratamiento frente a un cierto evento. De esta manera, se evitaba repetir en cada uno de los objetos el mismo manejador de eventos.

1.3.2. Internet Explorer 4

Microsoft, por su parte, lanzó el Internet Explorer 4, y fue más allá de lo proporcionado por Netscape, debido principalmente al hecho de que puso al alcance de JavaScript todos los elementos HTML, pero con el inconveniente de que este modelo era incompatible con el de Netscape.

La figura 4 representa el modelo de objetos de Internet Explorer 4:

Figura 4. Modelo de objetos HTML Dinámico de Internet Explorer 4



Las nuevas propiedades del objeto Document introducidas fueron:

- all[]: *array* que contiene todas las etiquetas HTML del documento;
- applets[]: *array* que contiene todos *applets* del documento;
- children[]: *array* de todos los elementos hijo del objeto;
- embeds[]: *array* formado por los objetos incrustados del documento;

- images[]: *array* de imágenes del documento;
- scripts[]: *array* de *scripts* del documento;
- styleSheets[]: *array* de objetos Style del documento.

De las características introducidas, sin lugar a dudas la más importante es el *array* "document.all[ ]", ya que proporcionaba acceso a todos los elementos del documento HTML. Este acceso se basa en el índice, que puede ser mediante el índice del objeto o mediante el atributo "id" o "name".

```
var elemento = document.all[2];   //elemento referencia el tercer objeto del
               //documento HTML
var elemento2 = document.all["Enviar"];   //elemento2 referencia el objeto del
                  //documento HTML con id "Enviar"
```

Una característica interesante es que "document.all[]" sitúa en un mismo nivel todos los elementos del documento HTML (independientemente de su posición en la jerarquía), y de esta forma permite un acceso rápido y sencillo a cualquier parte del documento HTML.

Otra característica fundamental disponible en IE4 fue que se podía modificar el diseño de forma dinámica (después de que la página hubiera sido cargada, a diferencia de Netscape 4). Esto implicó que por primera vez desde JavaScript se podía manejar toda la estructura del documento, el contenido de las variables y la estética de todos los aspectos de la página HTML de forma dinámica.

En cuanto al modelo de eventos implementado, fue totalmente opuesto al de Netscape 4, ya que los eventos empiezan en el objeto en el que ocurren y suben por los objetos ascendientes hasta llegar al objeto Window. De esta manera, cualquier objeto por el que pase el evento en su camino hasta Window puede capturar el evento, procesarlo o cancelarlo.

1.4. Modelo de objetos del DOM estándar

A tenor de lo comentado en los subapartados anteriores, el principal problema hasta el momento es que cada proveedor decide qué características HTML presenta al programador JavaScript y cuál es la sintaxis que se debe utilizar. Esta situación, como es de esperar, provocó una divergencia muy importante en la interpretación del código entre los distintos navegadores.

El World Wide Web Consortium (W3C, organización internacional que publica recomendaciones para la WWW) publicó, en octubre de 1998, una especificación denominada "DOM Nivel 1", en la que se consideraron las características y los mecanismos de manipulación de todos los elementos existentes en los archivos HTML y XML.

En noviembre del año 2000, se publicó la especificación del DOM Nivel 2. Esta especificación incluyó la manipulación de eventos en el navegador, la capacidad de interacción con CSS y la manipulación de partes del texto en las páginas de la Web.

Para finalizar, el DOM Nivel 3 se publicó en abril del 2004 y utiliza la DTD (definición del tipo de documento) y la validación de documentos.

De esta manera, el DOM especificado por la W3C propone una interfaz de programación de aplicaciones que pone todos los elementos de una página web a disposición del lenguaje de programación, e invita a todos los proveedores a adaptarla en sus navegadores con el claro objetivo de converger.

Web recomendada

Las especificaciones completas de cada uno de los niveles del DOM publicados por el W3C son documentos públicos y se pueden consultar en línea o descargarlos.

http://www.w3.org/DOM/

Otra forma de visualizar el DOM es clasificándolo en las siguientes categorías:

- **Núcleo del DOM**: especifica un modelo de gestión de documento formado por etiquetas con una estructura jerárquica.
- **DOM HTML**: especifica una extensión del anterior para su uso con HTML, proporciona las características necesarias para gestionar documentos HTML y utiliza una sintaxis similar a los modelos de objetos tradicionales.
- **Eventos DOM**: especifica el control de eventos, tanto de eventos de interfaz de usuario como eventos del DOM que se producen al modificar partes de la estructura del documento.
- **DOM CSS**: especifica las interfaces que permiten gestionar las reglas CSS desde el lenguaje de programación.
- **DOM XML**: es el equivalente al DOM HTML pero en este caso se trata de documentos XML.

Web recomendada

Para conocer el cumplimiento del estándar en las distintas versiones de los motores de diseño, podéis consultar la Wikipedia en la que encontrareis una comparativa entre estos.

http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(Document_Object_Model)

2. El modelo estándar de objetos

En este apartado, se va a presentar el modelo estándar de objetos que están planteados en el DOM Nivel 1 y Nivel 2 del W3C. Por tanto, en los siguiente apartados se va a estudiar la estructura jerárquica definida por el estándar, la tipología de objetos que existen en un documento HTML, y el apartado finalizará con los métodos que van a permitir acceder, modificar, eliminar y, por tanto, modificar cualquier elemento de la jerarquía.

2.1. Jerarquía de elementos

A continuación, se presenta una sencilla página HTML que servirá como base del posterior análisis y la presentación de los conceptos que definen el DOM.

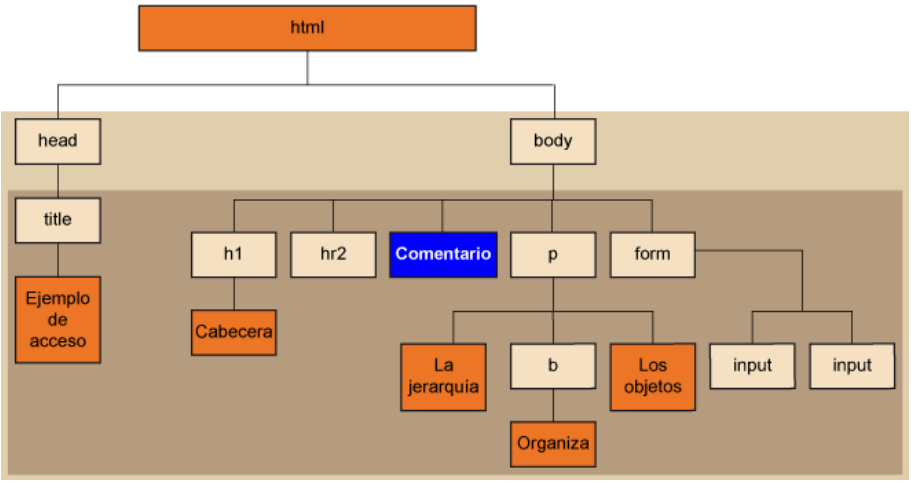
```
<html>
<head>
<!DOCTYPE HTML PUBLIC "//W3C//DTD HTML 4.01 Transitional//EN" http://www.w3.org/TR/html4/loose.dtd>
<title>Ejemplo de acceso</title>
</head>
<body>
<h1 id="cab">Cabecera</h1>
<hr>
<! - -Comentario - - >
<p id="p1">La jerarquia <b id="b1">organiza</b> los objetos</p>
<form name="form1" id="form1">
<input type="text" name="campo1" id="campo1">
<input type="button" name="ejecuta" id="ejecuta" value="Ejecuta">
</form>
</body>
</html>
```

Cuando el navegador carga un documento HTML, crea una estructura de árbol a partir de la las etiquetas HTML que definen el documento. Esta estructura jerárquica es sencilla de intuir, ya que el código HTML por sí mismo sigue una cierta estructura jerárquica, es decir:

- la etiqueta <html> contiene todas las etiquetas que definen la página web;
- la etiqueta <head> contiene la etiqueta <title>;
- la etiqueta <body> contiene a las etiquetas <h1>, <hr>, <!-->,<p> que a su vez contiene a una etiqueta <b>, y
- la etiqueta <form> contiene a dos etiquetas <input>.

Si se representa la estructura anterior en un gráfico en el que se muestran las relaciones a partir de líneas que unen las etiquetas, se obtiene la siguiente estructura:

Figura 5. Ejemplo de la estructura de un documento HTML



Cada elemento del árbol se conoce como **nodo**. Cada nodo puede contener a su vez otros nodos que pueden ser de distintos tipos (en el ejemplo se tienen 3 tipos distintos: etiquetas, comentarios y texto).

Estructura jerárquica

Una página web se organiza de manera jerárquica siguiendo la estructura definida por las etiquetas HTML.

En el estándar, el DOM establece 12 tipos de nodos, pero la mayoría solo son útiles en documentos XML, lo que reduce esta lista a la mitad en el caso de documentos HTML, tal como se puede apreciar en la siguiente tabla:

Tabla 1. Tipos de nodos del estándar DOM para HTML

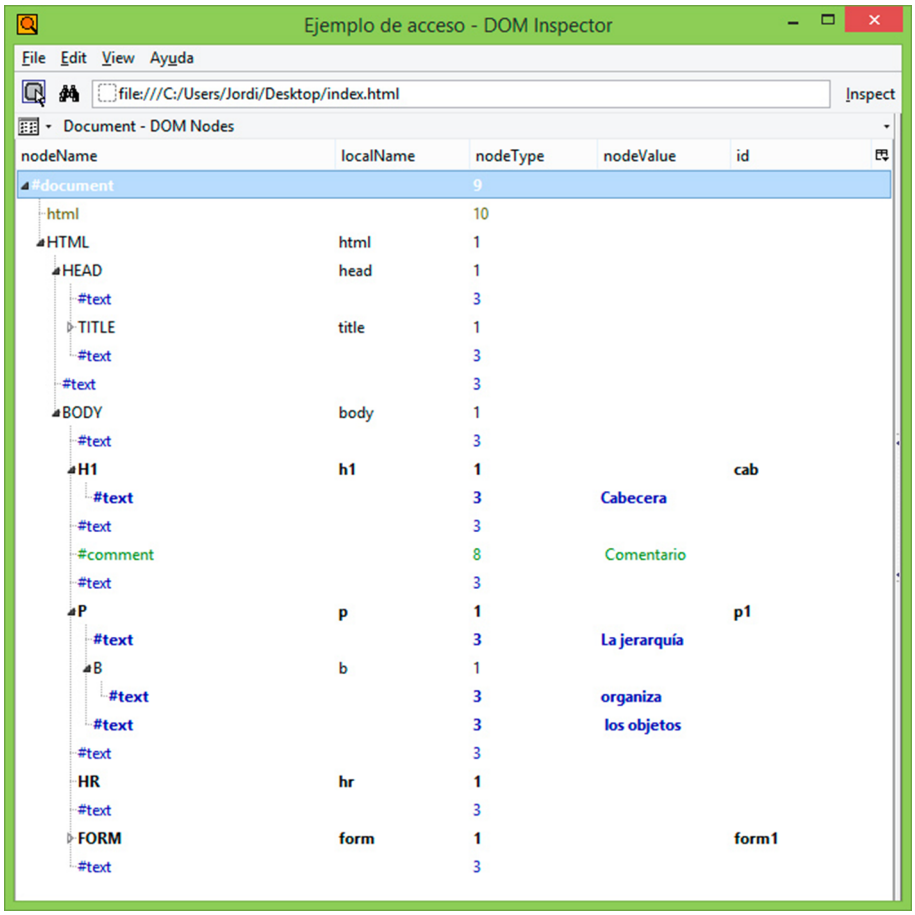
Identificador	Tipo	Descripción
1	Elemento	Una etiqueta HTML, por ejemplo <b>, <input>
2	Atributo	Un atributo de una etiqueta HTML, por ejemplo align=”center”
3	Texto	Un texto incluido en una etiqueta HTML, por ejemplo en nuestra sencilla página HTML “Ejemplo de acceso”
8	Comentario	Un comentario HTML, por ejemplo: “Comentario”
9	Documento	Documento raíz del documento, es decir la etiqueta <html>
10	Tipo documento	Una definición del tipo de documento, se trata de la etiqueta !DOCTYPE, por ejemplo: <!DOCTYPE HTML PUBLIC “//W3C//DTD HTML 4.01 Transitional//EN” http://www.w3.org/TR/html4/loose.dtd>

Estructura del DOM

Es posible analizar la estructura del documento a partir de ciertas herramientas que inspeccionan la página HTML y que muestran la jerarquía del documento desde el punto de vista del estándar de W3C. En el blog de la asignatura, encontraréis algunos ejemplos.

En el caso del ejemplo anterior, el inspector de DOM devuelve la siguiente estructura:

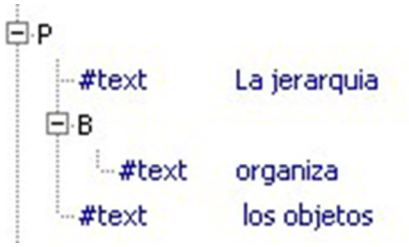
Figura 6. Ejemplo de la estructura HTML con un inspector del DOM



Como se observa, las etiquetas HTML son nodos del tipo 1, el texto del título, cabecera y del párrafo son nodos del tipo 3 y el comentario es un nodo del tipo 8. Si se hubiera definido algún atributo a una de las etiquetas, este aparecería como un nodo del tipo 2.

Las relaciones entre los nodos de la jerarquía son similares a las relaciones en los árboles genealógicos. Para explicar estas relaciones, se selecciona el subárbol del párrafo de texto. La estructura de este es la siguiente:

Figura 7. Estructura del subárbol del párrafo de texto



De este subárbol, se observan las siguientes relaciones:

- La etiqueta P es el nodo padre y dispone de tres hijos: el nodo de texto “La jerarquía”, el nodo elemento “B” y el nodo de texto “los objetos”.

- El orden de los hijos coincide con el indicado en el punto anterior, es decir, el primer hijo de “P” es el nodo de texto “La jerarquía”, el siguiente es el nodo elemento “B” y el último el nodo de texto “los objetos”.

- El nodo de texto “organiza” es hijo del nodo elemento “B” pero no del nodo “P”, sí que sería descendiente de este (nieto).

Clarificada la jerarquía de elementos o de nodos que se crean a partir de una página HTML, a continuación se van a estudiar los métodos del DOM que permitirán acceder a cada uno de los nodos.

2.2. Acceso a los nodos

Llegados a este momento, ya se dispone de suficientes conocimientos de la jerarquía que interpreta el DOM del W3C para empezar a estudiar el acceso a los elementos de esta. En primer lugar, se presenta un sencillo *script* que utiliza un primer método de acceso:

```
var elemento= document.getElementById("campo1");
var msg = "Nombre: " + elemento.nodeName + "\n";
msg += "Tipo: " + elemento.nodeType + "\n";
msg += "Valor: "+ elemento.nodeValue + "\n";
alert(msg);
```

El *script* utiliza el método del estándar “getElementById()”. A partir de un “id” que se le pasa como parámetro, este método devuelve el nodo o elemento y se asigna a la variable correspondiente. Se trata de un método con una funcionalidad similar a la que introdujo IE4 con el *array* “all”, ya que permite buscar directamente un elemento sin realizar una navegación por toda la jerarquía.

De esta manera, el *script*, una vez localizado el elemento, almacena en una cadena de texto 3 propiedades del nodo: “nodeName”, “nodeType” y “nodeValue”, y finaliza mostrando estas propiedades en una ventana “alert”. El resultado es el siguiente:

Figura 8

Como se puede observar, el nombre del nodo es “INPUT”, que es el tipo de la etiqueta; el tipo es “1”, que corresponde a un elemento HTML, y el valor es “null”. Estas propiedades ya se observaban en el “DOM inspector”.

El objeto Node dispone de un conjunto de propiedades que, además de describir sus características principales como el nombre, valor y tipo, también muestran su relación con el resto de nodos de la estructura y permiten la navegación por la estructura. La siguiente tabla muestra el conjunto de propiedades del objeto Node:

Tabla 2. Propiedades del objeto Node

Propiedades	Descripción
nodeName	Devuelve una cadena con el nombre del nodo.
nodeValue	Devuelve una cadena con el valor del nodo, solo aplicable a nodos de tipo texto.
nodeType	Devuelve un número que especifica el tipo de nodo.
parentNode	Devuelve una referencia al nodo padre.
firstChild	Devuelve una referencia al primer hijo.
lastChild	Devuelve una referencia al último hijo.
previousSibling	Devuelve una referencia al hermano mayor.
nextSibling	Devuelve una referencia al hermano menor.
childNodes	Devuelve un *array* con los nodos hijos.
attributes	Devuelve un *array* con los atributos del nodo.
ownerDocument	Devuelve el objeto Document que lo contiene.

Mediante estas propiedades es posible realizar recorridos por el árbol a partir de un nodo en concreto.

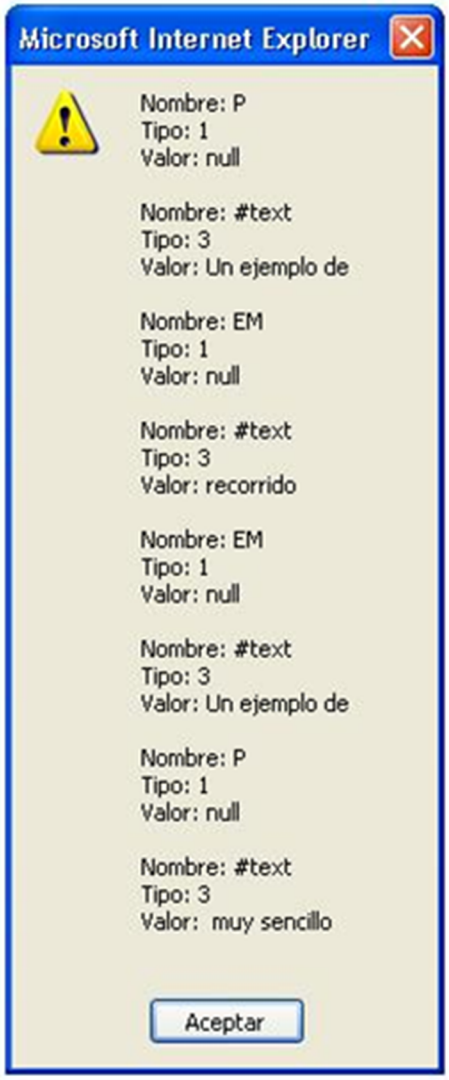
A continuación, se va a plantear un recorrido por la jerarquía a partir de un ejemplo con una estructura muy sencilla:

```
<html>
<head>
</head>
<body>
<p id="p1" align="center">Un ejemplo de <em>recorrido</em> muy sencillo</p>
<script type="text/javascript">
function tipoNodo(nodo){
   var temp = "";
   temp += "Nombre: "+nodo.nodeName+"\n";
   temp += "Tipo: "+nodo.nodeType+"\n";
   temp +="Valor: "+nodo.nodeValue+"\n\n";
```

```
    return temp;
}
var nodoActual = document.getElementById("p1");
var msg = tipoNodo(nodoActual);
nodoActual = nodoActual.firstChild;
msg += tipoNodo(nodoActual);
nodoActual = nodoActual.nextSibling;
msg += tipoNodo(nodoActual);
nodoActual = nodoActual.firstChild;
msg += tipoNodo(nodoActual);
nodoActual = nodoActual.parentNode;
msg += tipoNodo(nodoActual);
nodoActual = nodoActual.previousSibling;
msg += tipoNodo(nodoActual);
nodoActual = nodoActual.parentNode;
msg += tipoNodo(nodoActual);
nodoActual = nodoActual.lastChild;
msg += tipoNodo(nodoActual);
alert(msg);
</script>
</body>
</html>
```

El resultado del ejemplo anterior es la siguiente:

Figura 9. Un ejemplo de *recorrido* muy sencillo



Pero se ha realizado el recorrido por la jerarquía sin problemas debido a que se conocía la estructura y, por lo tanto, las relaciones entre los distintos nodos. Evidentemente, esta situación no se da en la realidad, ya que no se puede conocer a priori la estructura, ni siquiera si un nodo dispone de hijos o hermanos.

Por lo anterior, se podría realizar una navegación completa utilizando la siguiente estrategia; el recorrido empieza por un cierto nodo X, comprobamos si el nodo tiene hijos. Si los tiene, se accede a su hijo menor; si no los tiene, solo se puede o bien buscar un hermano menor o bien comprobar si tiene un nodo padre.

Para establecer un recorrido como anterior, se pueden utilizar las siguientes estructuras:

```
if (nodoActual.hasChildNodes())
   nodoActual=nodoActual.firstChild;
if (nodoActual.parentNode)
   nodoActual=nodoActual.parentNode;
```

```
if (nodoActual.nextSibling)
   nodoActual=nodoActual.nextSibling;
```

A continuación, se presenta una función que a partir de un nodo realiza un recorrido por la estructura jerárquica:

```
function mueveNodo(elementoActual, direccion){
   switch (direccion){
      case "previousSibling":
         if (elementoActual.previousSibling)
            elementoActual=elementoActual.previousSibling;
         else
            alert("No tiene hermano anterior");
      break;
      case "nextSibling":
         if (elementoActual.nextSibling)
            elementoActual=elementoActual.nextSibling;
         else
            alert("No tiene hermano posterior");
      break;
      case "parent":
         if (elementoActual.parentNode)
            elementoActual=elementoActual.parentNode;
         else
            alert("No tiene padre");
      break;
      case "firstChild":
         if (elementoActual.hasChildNodes())
            elementoActual=elementoActual.firstChild;
         else
            alert("No tiene hijos");
      break;
      case "lastChild":
         if (elementoActual.hasChildNodes())
            elementoActual=elementoActual.lastChild;
         else
            alert("No tiene hijos");
      break;
      default: alert("Llamada a dirección errónea");
   }
return (elementoActual);
}
```

La función anterior parte de dos parámetros: el primero indica el nodo inicial sobre el que se quiere hacer el recorrido, y el segundo, en qué dirección se quiere mover. La función devuelve el nodo, y en caso de no existir avisa con un mensaje.

No es común o no tiene demasiada utilidad el recorrido de una estructura para su análisis, pero es fundamental saber localizar un nodo (es sencillo con getElementById()) y poder acceder además a uno de sus "familiares directos" (padre, hijo o hermanos) de cara a modificar la estructura del documento, y para esto último la función anterior puede ser de gran ayuda.

El objeto Document no solo dispone del método getElementById(). En la siguiente tabla, se muestran los principales métodos de acceso:

Tabla 3. Principales métodos de acceso del objeto Document

Método	Descripción
getElementById()	Devuelve el objeto cuyo atributo id coincide con el pasado como parámetro.
getElementsByName()	Devuelve una lista con los objetos con el valor en el atributo name que se pasa como parámetro.
getElementsByTagName()	Devuelve una lista con los objetos cuya etiqueta HTML se pasa como parámetro.
documentElement	Devuelve el objeto raíz, es decir, la etiqueta <html>.
body	Devuelve el objeto correspondiente a la etiqueta <body>.

La diferencia entre los métodos getElementsByName() y getElementById() se basa en que el atributo "name" puede tener el mismo valor en varias etiquetas, mientras que la especificación sobre el atributo id indica que este identificador debe ser único en todo el documento. Por este motivo, se debe utilizar el atributo "id" frente al "name".

2.3. Manipulación de los nodos

Llegados a este punto, se dispone de una referencia a un nodo, y es posible moverse por la jerarquía a partir de los métodos anteriores, que permiten pasar de un nodo a su nodo padre, hermano o hijo. ¿Pero cómo se accede y modifica el contenido del documento HTML?

La modificación del contenido de una página HTML se basa en los siguientes puntos:

- **Modificación de nodos**: modificando su contenido o sus atributos.

- **Inserción o eliminación de nodos**: intercalando un nuevo nodo en una cierta posición en la jerarquía o eliminando uno de los nodos de la estructura.

2.3.1. Modificación de nodos

Para empezar, expondremos un pequeño ejemplo con un gran potencial, ya que a partir de este ejemplo se podrá modificar el contenido de cualquier página HTML.

El contenido de la información que se muestra en las páginas HTML se almacena en nodos del tipo 3, y como se observa en la figura 6, sobre el "DOM inspector" estos nodos no disponen de un valor en la columna "id", por lo que no es posible acceder a ellos utilizando el método getElementById().

Ved también

Podéis encontrar la figura 6 en el subapartado 2.1. de este módulo.

La solución es simple: se accede al nodo padre del texto que se quiere modificar, desde este se accede al nodo de texto mediante la propiedad *firstChild* y para acabar se modifica el texto utilizando la propiedad *nodeValue*.

En el siguiente ejemplo, se modifica el texto "La jerarquía" por "La estructura" del ejemplo planteado en el inicio del apartado actual:

```
var elemento= document.getElementById("p1").firstChild;
elemento.data ="La estructura";
```

De esta manera, se accede al nodo padre que tiene id "p1", de este se pasa a su primer hijo, que es el nodo de texto, y en la siguiente línea se sustituye el contenido del nodo.

Existe un conjunto de métodos que añade funcionalidad a la modificación de nodos de texto, ya que en el ejemplo anterior solo se muestra cómo sustituir todo el contenido de un nodo de texto. Los siguientes métodos permiten la modificación parcial de los nodos de texto:

Tabla 4. Métodos para modificar nodos de texto

Método	Descripción
appendData(*cadena*)	Añade la *cadena* al final del nodo de texto.
deleteData(*offset*, cantidad)	Borra la *cantidad* de caracteres empezando por el índice especificado por *offset*.
insertData(*offset*, *cadena*)	Inserta el valor de la *cadena* empezando por el índice especificado en *offset*.
replaceData(*offset*, cantidad, cadena)	Reemplaza la *cantidad* de caracteres de texto en el nodo empezando por *offset* con los caracteres especificados en la *cadena*.

Método	Descripción
splitText(*offset*)	Divide el nodo de texto en dos piezas en el índice dado en *offset*. Devuelve la parte derecha de la división en un nuevo nodo de texto y la parte izquierda, en el original.
substringData(*offset*, *cantidad*)	Devuelve una cadena correspondiente a la subcadena que empieza por el índice *offset* y sigue hasta una *cantidad* de caracteres.

Los métodos anteriores añaden la funcionalidad suficiente para que la manipulación del contenido de la página HTML se gestione de manera óptima. Una alternativa al uso de los métodos anteriores es la recuperación del contenido del nodo de texto en una cadena, la manipulación de esta cadena con los métodos existentes del objeto String utilizando expresiones regulares si fuera necesario y la asignación de nuevo del contenido de la cadena.

Pero tal como se puede intuir después de los ejemplos anteriores, en realidad no se modifican los nodos, sino los atributos de los nodos ("data" es un atributo o propiedad del nodo texto).

Por ejemplo, se quiere introducir el valor "Introduce texto..." en un cuadro de texto: para ello, se debe modificar el atributo "value" del nodo campo de texto "campo1":

```
var elemento= document.getElementById("campo1");
elemento.setAttribute("value","Introduce texto...");
```

En el ejemplo se ha utilizado el método setAttribute, que asigna un valor a un atributo del nodo sobre el que actúa. El objeto Node dispone de tres métodos que permiten trabajar con los atributos de los nodos. La siguiente tabla muestra la relación de métodos:

Tabla 5. Métodos del objeto Node para modificar nodos

Método	Descripción
getAttribute(*nombre*)	Devuelve el valor del atributo pasado como parámetro.
setAttribute(*nombre*,*valor*)	Asigna el atributo con el *nombre* y *valor* pasados como parámetros.
removeAttribute(*nombre*)	Elimina el atributo pasado como parámetro.

2.3.2. Creación e inserción de nodos

En el subapartado anterior, se han visto los métodos que permiten modificar nodos ya existentes, pero no la inserción de nuevos nodos en ciertas partes del documento. El proceso de inserción de un nodo en un documento tiene las siguientes etapas:

- creación del nuevo nodo que se quiere insertar;
- localización de la ubicación en la jerarquía en la que se va a insertar, e
- inserción del nuevo nodo en la ubicación adecuada.

Creación de nodos

La creación de un nuevo nodo es muy simple, el objeto Document dispone de métodos que permiten la creación de nodos. Por ejemplo, la siguiente línea de código crea un nodo del tipo 1 "Element", que representa una etiqueta HTML de párrafo "<p>":

```
nuevoParrafo = document.createElement("P");
```

De esta forma, la variable nuevoParrafo contiene una referencia a un nodo de tipo 1 con la etiqueta "P". De forma similar se procede a la creación de un nodo de tipo texto:

```
nuevoTexto = document.createTextNode("Nuevo nodo de texto");
```

En el anterior ejemplo, el método utilizado es createTextNode en lugar de createElement.

La siguiente tabla recopila los métodos que se utilizan en la creación de nodos:

Tabla 6. Métodos para la creación de nodos

Método	Descripción
createElement(*etiqueta*)	Crea un nodo del tipo elemento con la *etiqueta* pasada como parámetro.
createTextNode(*cadena*)	Crea un nodo de texto con la *cadena* pasada como parámetro.
createComment(*cadena*)	Crea un nodo de comentario con la *cadena* pasada como parámetro.
createAttribute(*nombre*)	Crea un atributo para un elemento especificado por la cadena *nombre*. Es raramente utilizado.

Como se observa en la tabla, existe un método de creación de nodos para cada uno de los tipos de nodos existentes, exceptuando los tipos 9 y 10 que se refieren a las etiquetas <html> y <body> respectivamente.

Una alternativa a la creación de un nuevo nodo es el **clonado** o **copia** de uno ya existente; para ello, el objeto Node dispone del método cloneNode(booleano), en el que el parámetro booleano es un argumento lógico que indica si la copia debe incluir a todos los hijos del nodo o solo el elemento en sí.

La característica anterior es muy interesante, ya que permite no solo copiar un nodo, sino también todos sus hijos.

Obsérvese el siguiente ejemplo:

```
function clonaYcopia(idNodoC,idNodoI){
   var nodoAClonar = document.getElementById(idNodoC);
   var nodoClonado = nodoAClonar.cloneNode(true);
   var puntoInsercion = document.getElementById(idNodoI);
   puntoInsercion.appendChild(nodoClonado);
}
```

La función tiene las siguientes características:

- Recibe dos parámetros, el primero es el "id" del nodo que se va a clonar, y el segundo parámetro es el "id" del nodo al que se le insertará el clon como hijo.

- La variable nodoAClonar es la referencia al nodo que se desea clonar, localizado con la función getElementById.

- La variable nodoClonado es la referencia al nuevo nodo que se ha clonado y que al pasarlo a la función cloneNode el parámetro *true* se clonará, no solo el elemento en sí, sino también todos sus hijos.

- La variable puntoInsercion es la referencia del nodo padre al que se le insertará en la siguiente línea de código el nodo clonado.

- La última línea de código muestra un nuevo método que será presentado a continuación, pero es intuitivo pensar que el método está insertando el nodo clonado como hijo del nodo puntoInsercion.

Inserción de nodos

Tal como se ha observado en el ejemplo del apartado anterior, cuando ya se dispone del nodo que se quiere añadir al documento, es necesario insertarlo en algún lugar de la jerarquía del documento.

La inserción de un nodo en la jerarquía siempre se realiza partiendo del nodo padre mediante los métodos que se muestran en la siguiente tabla:

Tabla 7. Métodos para la inserción de nodos

Método	Descripción
insertBefore(nuevoNodo, nodoHijo)	Se especifica delante de qué hijo "*nodoHijo*" se quiere insertar el nodo "*nuevoNodo*".

Método	Descripción
appendChild(*nuevoNodo*)	Añade el nodo "*nuevoNodo*" al final de la lista de los hijos del nodo que ha realizado la llamada al método.

Por lo tanto, la diferencia entre los dos métodos se basa en que el primer método inserta el nodo pasado en el parámetro "nuevoNodo", a la izquierda del hijo pasado como segundo parámetro "nodoHijo"; es decir, el nuevo nodo será el hermano mayor del que se pasa como parámetro.

Sin embargo, el método appendChild() añade el nuevo nodo al final de la lista de los hijos y, por lo tanto, será el hijo menor o situado más a la derecha de todos los nodos hijos.

Los métodos anteriores se pueden observar en el siguiente ejemplo:

```
var nuevoNodo = document.getElementById("form1").cloneNode(true);
var padre = document.getElementById("cab");
padre.appendChild(nuevoNodo);
```

donde:

- se crea un clon de un formulario identificado con el id "form1", ya que no solo se clona la etiqueta del formulario, sino todas sus hijas al pasar como parámetro del método cloneNode el valor *true*;
- se localiza el nodo padre sobre el que se insertará el formulario; este está identificado con el id "cab", y
- el código finaliza con la adición del formulario como el menor de los hijos del nodo "cab", para lo que se emplea el método appendChild.

Eliminación y remplazo de nodos

En ocasiones, es necesario poder eliminar nodos en la estructura del documento. Para ello, el objeto Node dispone del método removeChild(nodoHijo), que se utiliza para eliminar un nodo especificado por el parámetro "nodoHijo". Por ejemplo:

```
nodoActual.removeChild(nodoActual.lastChild);
```

El código anterior elimina el último hijo del nodo al que hace referencia la variable nodoActual; es importante saber que, además de eliminar, lo devuelve y puede ser asignado a una variable.

Además de eliminar un nodo, es posible reemplazar un nodo por otro utilizando el método replaceChild(hijoNuevo, hijoReemplazado), en el que el nodo "hijoNuevo" remplazará el nodo "hijoReemplazado".

3. Los objetos del BOM

En este apartado, se van a presentar los objetos junto con sus principales propiedades y métodos que forman parte de BOM, de manera que en el siguiente apartado se presentará el DOM, es decir, todo el modelo de objetos que desciende desde el objeto Document.

Web recomendada

Para información más detallada y actualizada de las propiedades y los métodos de los objetos del BOM, podéis visitar la página de w3schools.

http://www.w3schools.com/jsref/

Como es de esperar, la lista de objetos presentados es dinámica y evoluciona en paralelo a la aparición de nuevas versiones de los navegadores. La utopía de la estandarización sigue siendo un objetivo pendiente y no una realidad como sería deseable.

Es importante destacar que esta parte de BOM sufre el mayor número de divergencias, ya que este está directamente relacionado con el navegador.

Por tanto, el objetivo del apartado es la presentación de los objetos y sus componentes que, junto con algunos ejemplos, ayudarán a comprender el potencial del proceso de manipulación del DOM desde JavaScript. En ningún momento se plantea que se aprendan "de memoria" todos los objetos con sus métodos y propiedades, no tiene ningún sentido, siempre se pueden consultar en el propio apartado o en línea en internet.

En cuanto a esto último, y de cara a solucionar problemas con la compatibilidad del código, las siguientes referencias web muestran el modelo de objetos en cada uno de los navegadores actuales, incluyendo tanto el BOM como el DOM:

- Especificación del DOM de los navegadores basado en Mozilla: https://developer.mozilla.org/en/Gecko_DOM_Reference

- Especificación del DOM de los navegadores Internet Explorer: http://msdn.microsoft.com/en-us/default.aspx

3.1. El objeto Window

El objeto Window es el que se encuentra en el nivel más alto en la jerarquía de objetos y, por lo tanto, es el contenedor de todo aquello que se puede ver en el navegador.

Tan pronto como el navegador se ejecuta, y aunque este no visualice ningún documento, se crea una instancia de Window que es accesible desde JavaScript.

Pueden referenciarse todas las ventanas que están abiertas en la pantalla y pueden ser manipuladas por una instancia del objeto (siempre que hayan sido creadas desde la propia instancia que las intenta manipular). Tal como se verá en este subapartado, las propiedades de una instancia de Window incluyen su tamaño, componentes, posición, etc. Por su parte, los métodos van a permitir la creación y destrucción de ventanas genéricas, de ventanas de alerta o de confirmación, etc.

Debe tenerse en cuenta que el objeto Window puede representar la ventana del navegador o a un *frame* (cuando existen varios *frames* cada uno de ellos se considera un objeto Window independiente).

Así pues, un objeto Window se puede crear mediante alguna de las siguientes opciones:

- las etiquetas “body” o “frameset” de HTML, y
- el método open de JavaScript.

Por su parte, el acceso a las propiedades y los métodos del objeto Window siguen la sintaxis que se ha presentado en los apartados anteriores:

```
window.nombre_propiedad;
window.nombre_metodo([parámetros]);
```

Cuando se hace referencia a la ventana que contiene el propio documento, puede usarse la palabra *self* en sustitución de la palabra *window*. Por ejemplo, para cerrar la ventana actual se pueden usar indistintamente las formas window.close() y self.close().

De todos modos, con los métodos open() y close() es necesario usar las formas window.open() y window.close(), ya que aquellos se podrían confundir con los métodos del objeto Document (document.open() y document.close()).

La palabra *self* puede omitirse en los casos más simples y reservarse para aquellos casos que impliquen varios *frames*. Esto es debido a que, cuando se visualiza un documento, se asume el hecho de que ya existe una ventana, la que lo contiene. Por lo tanto, se puede acceder a las propiedades y los métodos de la ventana actual sin especificarla:

```
nombre_propiedad
nombre_metodo([parametros])
```

A continuación, se va a mostrar una tabla con las principales propiedades disponibles del objeto Window:

Tabla 8. Propiedades del objeto Window

Nombre	Descripción	Ejemplo

closed	Es un valor booleano que indica si una ventana ha sido cerrada. Cuando se cierra una ventana, el objeto Window continúa existiendo, pero con el valor de esta propiedad true.
	Es de solo lectura.
		`ventana = window.open( 'doc.htm','ventana1', 'width=400, height=100' )`
		`...`
		`if (ventana.closed)`
		`  alert("cerrada")`
		`else`
		`  alert("abierta")`

defaultStatus	Es el mensaje por defecto que se visualiza en la barra de estado del navegador.	**window.defaultStatus**="Página índice"
document	Hace referencia al objeto Document contenido en la ventana.	
frames	Es un *array* de objetos *frame*. Los *frames* son los generados por la etiqueta Frame del FRAMESET contenido en la página, y en el *array* están en el orden en el que se han creado en el código HTML. Es de solo lectura.	En una página que contenga dos *frames*, se puede acceder a estos de la siguiente manera: parent.frames["findice"] parent.frames["fcontenido"] o bien parent.frames[0] parent.frames[1]
history	Hace referencia al objeto History contenido en la ventana.	

length	Contiene el número de *frames* de la ventana.
	Es de solo lectura.
		`numframes = window.length;`
		`if (numframes == 0)`
		`  alert("no contiene frames");`
		`else`
		`  alert("contiene "+numframes+" frames");`

location	Hace referencia al objeto location contenido en la ventana.	
name	Cadena de texto que especifica el nombre de la ventana o marco. Es de solo lectura.	ventana = window.open('doc.htm','ventana1','width=400, height=100') alert(ventana.name)
navigator	Hace referencia al objeto Navigator contenido en la ventana.	
opener	Hace referencia al objeto Window que ha abierto la ventana actual. Esta propiedad persiste aunque el documento que ha hecho la llamada se haya descargado.	Cierra la ventana que ha abierto la ventana actual: window.opener.close() Cierra la ventana del navegador: top.opener.close()
parent	Hace referencia al objeto Window o Frame padre del marco actual. Es la ventana que contiene el *frameset*.	En una página que contenga dos *frames*, se puede acceder al segundo desde el primero con la siguiente sentencia: **parent**.frames[1]
screen	Hace referencia al objeto Screen del navegador.	
self	Es sinónimo de la ventana actual.	

status	Especifica el texto que se visualiza en la barra de estado del navegador.	`function ini_status() {`
		`  window.status = "Pulse en este link para acceder al índice"`
		`}`
		`<a href="indice.htm" onMouseOver = "ini_status()">Índice</a>`

Nombre	Descripción	Ejemplo
top	Hace referencia al objeto Window superior en la jerarquía de objetos del documento. Se utiliza para acceder al objeto Window contenedor desde un marco.	top.close()
window	Es la ventana o el *frame* actual.	

La siguiente tabla muestra los principales métodos del objeto Window. Al igual que en el caso de las propiedades, no se trata de una listado exhaustivo, sino que se presentan aquellos métodos más utilizados o considerados interesantes por el autor.

Tabla 9. Métodos del objeto Window

Nombre	Descripción	Sintaxis	Parámetros	Retorno
alert	Abre una caja de diálogo con un mensaje y el botón *Aceptar*.	alert(cadena de texto)	String (cadena de texto)	
blur	Elimina el foco de la ventana o *frame* especificados.	blur()		
clearInterval	Cancela el *timeout* que ha sido inicializado con el método setInterval.	clearInterval(intervalID)	intervalID se inicializa con la llamada al método setInterval previa.	
clearTimeout	Cancela el *timeout* que ha sido inicializado con el método setTimeout.	clearTimeout (intervalID)	intervalID se inicializa con la llamada al método setTimeout previa.	
close	Cierra la ventana especificada.	close()		
confirm	Abre una caja de diálogo con un mensaje y los botones *Aceptar* y *Cancelar*.	confirm("mensaje")	String	*True* si el usuario pulsa *Aceptar* y *false* si pulsa *Cancelar*.
focus	Asigna el foco a la ventana o *frame* especificado.	focus()		
moveBy	Mueve la ventana a la posición relativa, respecto a su posición actual, el número de píxeles especificados.	moveBy(horizontal, vertical)	Horizontal: número de píxeles para el desplazamiento horizontal. Vertical: número de píxeles para el desplazamiento vertical.	
moveTo	Mueve la esquina superior izquierda de la ventana a la posición absoluta de la pantalla especificada en píxeles.	moveTo(x, y)	x: coordenada x y: coordenada y	
open	Abre una nueva ventana del navegador.	open(URL, nombre, características)	URL: cadena de texto que especifica la URL que se va a visualizar en la nueva ventana. Nombre: texto que indica el nombre de la ventana.	
print	Hace que aparezca el diálogo de impresión.	print()		

Nombre	Descripción	Sintaxis	Parámetros	Retorno
prompt	Abre una caja de diálogo con un mensaje y un campo de entrada de texto.	prompt(mensaje, [texto])	Mensaje: texto que muestra la caja de diálogo. Texto: opcional, texto que por defecto sale en el campo de entrada de texto.	Dato introducido por el usuario.
resizeBy	Redimensiona la ventana moviendo la esquina inferior derecha según los valores relativos especificados.	resizeBy(horizontal, vertical)	Horizontal: número de píxeles que hay que sumar o restar a la dimensión horizontal. Vertical: número de píxeles que hay que sumar o restar a la dimensión vertical.	
resizeTo	Redimensiona la ventana en los valores especificados.	resizeTo(ancho, alto)	Ancho: ancho en píxeles de la ventana. Alto: altura en píxeles de la ventana.	
scrollBy	Hace un *scroll* del área de la ventana, según los valores relativos a la posición actual.	scrollBy(x,y)	x: número de píxeles que hay que sumar o restar para el *scroll* horizontal. y: número de píxeles que hay que sumar o restar para el *scroll* vertical.	
scrollTo	Hace un *scroll* del área de la ventana, según los valores absolutos indicados.	scrollTo(x, y)	x: coordenada x en píxeles, del área que se va a visualizar. y: coordenada y en píxeles, del área que se va a visualizar.	
setInterval	Ejecuta una función cada vez que transcurre el tiempo especificado y hasta que se ejecuta el método clearInterval.	setInterval(expresión, tiempo) o bien setInterval(función, tiempo, param1, ..., paramN)	Tiempo: número de milisegundos	Identificador
setTimeout	Ejecuta una función una vez transcurrido el tiempo especificado.	setTimeout(expresión, tiempo) o bien setTimeout(función, tiempo, param1, ..., paramN)	Tiempo: número de milisegundos.	Identificador

El uso de algunos de los métodos anteriores necesita de la activación o desactivación de ciertas propiedades relacionadas con la seguridad del navegador; esto significa que es posible que no funcionen debido a restricciones de seguridad.

3.1.1. El método open

Cuando se abre una ventana, se puede especificar la dirección URL, el nombre, el tamaño, los botones y todo un conjunto de atributos que definen el aspecto de la ventana.

La sintaxis del método open es:

```
open(url, nombre, características, reemplazar)
```

donde:

- url es la dirección que indica el documento que se cargará en la ventana;
- nombre es el nombre de la ventana (que se utilizará para referenciarla posteriormente);
- características es una cadena delimitada por comas que indica un conjunto de propiedades de la ventana que se va a crear, y
- reemplazar es un valor lógico y opcional que indica si la URL especificada debe reemplazar el contenido de la ventana actual.

Un primer ejemplo de este método sería:

```
ventanaUoc = open("http://www.uoc.edu", "UOC", "height=300, width=200");
```

De la misma manera que se puede abrir una ventana, es posible cerrarla utilizando el método close():

```
ventanaUoc.close();
```

Pero este método solo puede cerrar ventanas que hayan sido creadas previamente desde el mismo ámbito, y en algunos navegadores, por políticas de seguridad, no se puede cerrar la ventana principal.

La lista de **opciones** del método es muy amplia, a continuación se presentan las más "interesantes":

- **alwaysLowered**: valor booleano que indica que la nueva ventana quedará por debajo del resto de ventanas;
- **alwaysRaised**: valor booleano que indica que la nueva ventana quedará por encima del resto de ventanas;
- **dependent**: valor booleano que indica que la ventana creada es dependiente de la ventana padre. Esto implica que al cerrar una ventana se cierran todas las dependientes de esta;
- **directories**: valor booleano que indica que la nueva ventana contendrá la botones de directorio estándar;
- **height**: valor en píxeles que especifica el alto de la ventana;
- **hotkeys**: valor booleano que indica si las teclas rápidas del navegador están habilitadas en la nueva ventana;

- **innerHeight**: valor en píxeles que especifica el alto del contenido de la ventana;
- **innerWidth**: valor en píxeles que especifica el ancho del contenido de la ventana;
- **left**: valor en píxeles que especifica la coordenada x en la que aparecerá la nueva ventana;
- **location**: valor booleano que indica si la barra de direcciones debe mostrarse en la ventana;
- **menubar**: valor booleano que indica si la nueva ventana contendrá la barra de menú;
- **outerHeight**: valor en píxeles que especifica la dimensión vertical de los bordes de la nueva ventana;
- **resizable**: valor booleano que indica si el usuario podrá cambiar el tamaño de la nueva ventana;
- **scrollbars**: valor booleano que indica si la nueva ventana contendrá las barras de *scroll* cuando el documento exceda los límites de la ventana;
- **status**: valor booleano que indica si la nueva ventana contendrá la barra de estado;
- **titlebar**: valor booleano que indica si la nueva ventana contendrá la barra de título;
- **toolbar**: valor booleano que indica si la nueva ventana contendrá la barra de herramientas estándar;
- **top**: valor en píxeles que especifica la coordenada "y" en la que aparecerá la nueva ventana;
- **width**: valor en píxeles que especifica el ancho de la ventana, y
- **z-lock**: valor booleano que especifica si no se podrá cambiar el orden de pila relativo a otras ventanas incluso si esta obtiene el foco.

El siguiente ejemplo abre una ventana con las características que se usan más comúnmente:

```
function abrir(){
   url= "http://www.uoc.edu";
   caract = "left=50, top=50, status=yes, menubar=yes, toolbar=no,
```

```
    width=590, height=250, directory=no, resize=no, scrollbars=yes";
    return window.open(url,"Ejemplo apertura",caract);
}
```

3.1.2. Cajas de diálogo

Existen **tres métodos** que crean las clásicas cajas de diálogo o mensajes en ventanas modales:

- **alert()**: crea una ventana con un mensaje y el botón *Aceptar* que cierra la ventana;

- **confirm()**: crea una ventana que muestra un mensaje y espera que el usuario responda pulsando el botón *Aceptar* o *Cancelar*. Este devuelve el valor *true* si se ha pulsado el botón *Aceptar*, y *false* si se ha pulsado *Cancelar*;

- **prompt()**:crea una ventana que muestra un mensaje y solicita datos al usuario a partir de un campo de texto. Al igual que confirm(), dispone de los botones *Aceptar* y *Cancelar*; en el caso de presionar *Cancelar*, el método devolverá el valor *null*, en caso contrario, devolverá el contenido del campo de texto introducido por el usuario.

El siguiente ejemplo muestra un caso de uso de los tres métodos:

```
var entrada = prompt("Entra un dato: ", 0);
alert("El dato introducido es: " + entrada);
var respuesta = confirm("¿Desea cerrar la ventana?");
if (respuesta==true)window.close();
```

El código anterior se interpreta perfectamente si se observa el resultado en las figuras siguientes:

Figura 10. Caja de diálogo prompt

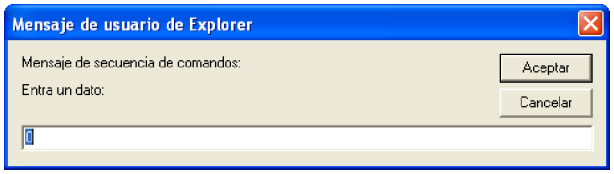


Figura 11. Caja de diálogo alert

Figura 12. Caja de diálogo confirm

3.1.3. Los métodos setTimeout y clearTimeout

En el siguiente ejemplo, se va a crear un reloj, para lo que se van a emplear los métodos setTimeout() y clearTimeout():

```
var timerID = null;
var timerRunning = false;
function stopReloj(){
   if(timerRunning)
      clearTimeout(timerID);
   timerRunning = false;
}
function startReloj(){
   stopReloj();//Hay que asegurar que el reloj está parado
   ver();
}
function ver(){
   var now = new Date();
   var hours = now.getHours();
   var minutes = now.getMinutes();
   var seconds = now.getSeconds();
   var timeValue = "" + ((hours > 12) ? hours - 12 : hours);
   timeValue += ((minutes < 10) ? ":0" : ":") + minutes;
   timeValue += ((seconds < 10) ? ":0" : ":") + seconds;
   timeValue += (hours >= 12) ? " P.M." : " A.M.";
   document.reloj.face.value = timeValue;
   timerID = setTimeout("ver()",1000);
   timerRunning = true;
}
```

El documento HTML solo necesitará implementar lo siguiente:

- Realizar una llamada a la función startReloj().
- Disponer de un formulario y un campo de texto que se llamen “reloj” y “face”, respectivamente.

La base del funcionamiento de reloj está en la llamada recursiva cada segundo de la función ver(); de esta manera, a cada segundo se va actualizando la información de la hora en el campo de texto del formulario.

3.1.4. El método moveBy

En el siguiente ejemplo, se puede observar el uso de los métodos moveBy y setInterval de manera coordinada, lo que provoca el desplazamiento por la pantalla de la ventana recién creada:

```
var v1;
function abrir(){
   var caract = "left=200, top=10, width=400, height=210, statusbar=no, menubar=no, directory=no,
   resize=no, scrollbars=no";
   v1=window.open("Movimiento.html","Ventana móvil",caract);
   ini_mover()
}

function ini_mover(){
   setInterval('mover()',500);
}

function mover(){
   v1.moveBy(5,10);
}
```

La llamada a la función abrir() debe realizarse desde la página HTML, ya que en primer lugar crea una nueva ventana y, a continuación, llama a la función que inicia el movimiento.

3.2. El objeto Location

El objeto Location proporciona acceso a la dirección URL y a porciones de la URL de manera estructurada. Asignar una cadena a un objeto Location provoca que el navegador analice la cadena como una dirección de internet, actualice las propiedades del objeto y establezca la cadena como la propiedad *href* del objeto.

Una URL tiene la siguiente estructura:

```
protocol//host:port/pathname#hash?search
```

Por ejemplo:

```
http://www.uoc.edu:8080/assist/extensions.html#topic1?x=7&y=2
```

El significado de cada una de las partes que componen la URL es el siguiente:

- protocol: representa el inicio de la URL, es decir el protocolo web que incluye los dos puntos (http:);

- host: representa el dominio o la dirección IP del ordenador anfitrión en la red (www.uoc.edu);

- port: representa el puerto de comunicación que el servidor usa para las comunicaciones (8080);

- pathname: representa la ruta del documento (assist/extensions/);

- hash: representa el ancla en la URL, incluido el signo # (solo para http);

- search: representa la información para una búsqueda (*query*). Incluye el signo ?, que indica el inicio (solo para http). Cada elemento de la búsqueda está compuesto por el nombre de la variable y su valor, y cada elemento se separa del siguiente por el signo &.

Cuando se asigna un valor a la propiedad *location* de un objeto, JavaScript crea un objeto Location y asigna este valor a la propiedad *href* del objeto Location. Por tanto, las dos formas que se muestran a continuación son equivalentes:

```
window.location.href="http://www.uoc.edu";
window.location="http://www.uoc.edu";
```

El objeto Location hereda directamente del objeto Window. Si se hace referencia al objeto Location sin especificar una ventana, la información de este estará asociada a la ventana actual.

A continuación, se muestran en una tabla las principales propiedades del objeto Location:

Tabla 10. Propiedades del objeto Location

Nombre	Descripción	Ejemplo
hash	Nombre de ancla en la URL.	alert(window.**location.hash**)
host	Nombre del servidor o del dominio que forma parte del parámetro hostname.	window.**location.host** = "www.uoc.edu"
hostname	Contiene el nombre completo del servidor, el host y el puerto.	window.**location.host** = "www.uoc.edu:8080"
href	Especifica la URL completa.	window.**location.href**="http://www.uoc.es/"
pathname	Especifica la ruta del documento.	window.**location.pathname** = "/js/ejemplos/reloj.html"
port	Puerto de comunicaciones que usa el servidor.	window.location.port = 80
protocol	Inicio de la URL que especifica la forma de acceso.	window.location.protocol = "http:"

Nombre	Descripción	Ejemplo
search	Especifica la búsqueda en la URL.	window.**location.search** = "?cp=2&scp=4"

La siguiente tabla muestra los principales métodos del objeto Location:

Tabla 11. Métodos del objeto Location

Nombre	Descripción	Sintaxis	Parámetros
assign	Es equivalente a modificar la URL mediante location.href	assign(url)	url: cadena de texto.
reload	Actualiza el documento.	reload([forceGet])	Opcional. Si es true fuerza la carga del documento con el método GET.
replace	Carga la URL especificada en la ventana y sustituye a la que había.	replace(url)	url: cadena de texto.

Respecto a los métodos anteriores, es importante tener en cuenta que la URL de una página cargada con el método replace substituye a la anterior en el historial de páginas visitadas, por lo que al reemplazar una página la referencia a esta es substituida en el historial.

En el siguiente ejemplo, se puede observar el uso de propiedades y métodos de este objeto:

```
<html>
<head>
<title>Uso del objeto location</title>
<script type="text/javascript">
   function mostrarImagen(img) {
      document.images[0].src=img;
   }
</script>
</head>
<body>
<form name="miForm">
   <h2>Escoja un regalo:</h2>
   <p>
      <input type="radio" name="imagen" value="imagen1" checked
      onclick="mostrarImagen('img/regalo1.gif')"> Verde <br>
      <input type ="radio" name ="imagen" value ="imagen2"
      onclick="mostrarImagen('img/regalo2.gif')"> Amarillo <br>
      <input type ="radio" name ="imagen" value ="imagen3"
      onclick="mostrarImagen('img/regalo3.gif')"> Rojo
   </p>
   <p>
      <img name="mi_imagen" SRC="img/regalo1.gif" align="center">
      <input type ="button" value="Actualizar" onclick="window.location.reload()">
      <input type="button" value="Ver regalo"
```

```
        onclick="window.location.replace('regalo.htm')" name="button">
    </p>
</form>
</body>
</html>
```

En el ejemplo anterior, se ha insertado un botón en el formulario que realiza una recarga de la página HTML a partir del método reload() y un nuevo botón *Ver regalo*, que substituye la página cargada por una nueva "regalo.htm", que supuestamente mostrará el regalo elegido. También se necesita tener creada la página "regalo.htm" y las imágenes "regalo1.gif", "regalo2.gif" y "regalo3.gif" en una subcarpeta con el nombre "img".

3.3. El objeto History

El navegador almacena un *array* con las direcciones web visitadas en el objeto History, de manera que este proporciona propiedades y métodos que permitirán desde JavaScript acceder a estas direcciones y visitar estas páginas web.

Al objeto History se accede mediante la propiedad *history* de Window (window.history), ya que es un objeto que hereda directamente de Window.

Tal como se ha comentado, mantiene el historial de páginas visitadas, guardando las URL en un *array* de objetos y, por lo tanto, si se accede a una posición del *array*, por ejemplo history[2], se obtendrá la URL visitada en tercer lugar (la primera posición de un *array* en JavaScript es la 0). Se debe tener en cuenta que, por motivos de seguridad, es posible que ciertas propiedades estén restringidas por defecto en algunos navegadores.

Las principales propiedades del objeto son las siguientes:

Tabla 12. Propiedades del objeto History

Nombre	Descripción	Ejemplo
length	Número de elementos del *array*. Es de solo lectura.	alert(history.length)
state	Devuelve el valor del estado del objeto.	alert(history.state)

Los principales métodos del objeto son los siguientes:

Tabla 13. Métodos del objeto History

Nombre	Descripción	Sintaxis	Parámetros
back	Carga la anterior URL del historial.	back()	
forward	Carga la siguiente URL del historial.	forward()	

Nombre	Descripción	Sintaxis	Parámetros
go	Carga una URL del historial.	go(pos) go(url)	pos: entero que representa la posición relativa en el historial. url: string que representa una url del historial.

Respecto a los métodos anteriores, se deben tener en cuenta los siguientes detalles:

- El método back tiene el mismo efecto que usar el método go de la forma: history.go(-1).
- El método forward tiene el mismo efecto que usar el método go de la forma: history.go(1).
- La forma del método go, go(0), tiene como efecto la actualización de la página actual.

A continuación, se presenta un ejemplo en el que se puede observar el uso de los métodos del objeto:

```
<html>
<head>
</head>
<body>
<table width="100%" border="0" cellspacing="0" cellpadding="0">
<tr>
      <td bgcolor="#000000">
         <div align="center"><a href="javascript:history.back()">Anterior</a></div>
      </td>
      <td bgcolor="#000000">
         <div align="center"><a href="javascript:history.forward()">Siguiente</a></div>
      </td>
      <td bgcolor="#000000">
         <div align="center"><a href="javascript:history.go(0)">Actualizar</a></div>
      </td>
</tr>
</table>
<p align="center">P&aacute;gina 1</p>
<div align="center">
   <p><a href="http://www.uoc.edu">P&aacute;gina 2</a></p>
   <p><a href="http://www.google.com">P&aacute;gina 3</a></p>
</div>
</body>
</html>
```

3.4. El objeto Screen

El objeto Screen contiene propiedades de solo lectura, que suministran información acerca de la pantalla del usuario. Se trata de un objeto hijo de Window.

A continuación, se presentan las principales propiedades del objeto:

Tabla 14. Propiedades del objeto Screen

Nombre	Descripción
availHeight	Altura en píxeles de la pantalla sin contar las posibles utilidades que pueda mostrar el sistema en pantalla, como las barras de herramientas.
availWidth	Ancho en píxeles de la pantalla sin contar las posibles utilidades que pueda mostrar el sistema en pantalla, como las barras de herramientas.
colorDepth	Si existe una paleta en uso, indica la profundidad de color en bits por píxel; en otro caso, deriva de screen.pixelDepth.
height	Altura de la pantalla en píxeles.
pixelDepth	Resolución de la pantalla en bits por píxel.
width	Ancho de la pantalla en píxeles.

El objeto Screen no dispone de métodos.

3.5. El objeto Navigator

El objeto Navigator contiene la información del navegador que se está usando. Estas propiedades son utilizadas generalmente para la detección del navegador, aunque también proporcionan una serie de detalles sobre la configuración del usuario, como el idioma de preferencia y el sistema operativo.

Este objeto es descendiente directo del objeto Window y todas sus propiedades son de solo lectura. A continuación, se pueden consultar las principales propiedades del objeto en la tabla siguiente:

Tabla 15. Propiedades del objeto Navigator

Nombre	Descripción	Ejemplo
appCodeName	Nombre del código del navegador.	document.write("El código de su navegador es " + navigator.appCodeName)
appName	Nombre del navegador.	document.write("El nombre de su navegador es " + navigator.appName)
appVersion	Versión del navegador.	document.write("La versión de su navegador es " + navigator.appVersion)
cookieEnabled	Indica si el navegador tiene activas las cookies.	if (navigator.cookieEnabled) alert("Las cookies están activadas en el navegador")

Nombre	Descripción	Ejemplo
mimeTypes	*Array* de todos los tipos MIME soportados por el navegador.	
platform	Sistema operativo sobre el que se ejecuta el navegador.	alert(navigator.platform) Podría mostrar datos tales como: Win32, Win16, Mac68k, MacPPC, etc.
plugins	*Array* de *plugins* instalados en el navegador.	
userAgent	Valor de la cabecera user-agent enviada en el protocolo HTTP, del cliente al servidor.	document.write("La cabecera user-agent enviada en el protocolo HTTP es " + navigator.userAgent)
vendor	Cadena que contiene información de la marca comercial del navegador.	

La siguiente tabla muestra el método más usado de los disponibles en el objeto:

Tabla 16. Métodos del objeto Navigator

Nombre	Descripción	Sintaxis	Retorno
javaEnabled	Testea si la opción Java está activada.	javaEnabled()	*True* si está activo java y *false* en caso contrario.

En el siguiente ejemplo, se muestra el uso de las propiedades del objeto:

```
<html>
<head>
</head>
<body>
<h1>Propiedades del navegador</h1>
<script type="text/javascript">
   document.write("Nombre código: <b>" + navigator.appCodeName + "</b><br>")
   document.write("Nombre: <b>" + navigator.appName + "</b><br>")
   document.write("Versión: <b>" + navigator.appVersion + "</b><br>")
   document.write("Idioma: <b>" + navigator.language + "</b><br>")
   document.write("Tipos MIME: <b>" + navigator.mimeTypes + "</b><br>")
   document.write("Plataforma: <b>" + navigator.platform + "</b><br>")
   document.write("Plugins: <b>" + navigator.plugins + "</b><br>")
   document.write("Cabecera URL: <b>" + navigator.userAgent + "</b><br>")
</script>
</body>
</html>
```

3.6. El objeto MimeType

El objeto MimeType (*multipart internet mail extension*) representa el tipo MIME soportado por el cliente. Se puede acceder a este objeto mediante el *array* MimeType del objeto Navigator o desde el objeto Plugin:

```
navigator.mimeTypes[index]
```

De esta manera, el programador puede consultar si un tipo MIME en particular está soportado y, si es así, extraer información sobre el objeto Plugin que lo controla. Las propiedades de este objeto son del tipo solo lectura y se presentan en la tabla siguiente:

Tabla 17. Propiedades del objeto MimeType

Nombre	Descripción	Ejemplo
description	Descripción del tipo MIME.	navigator.mimeTypes["image/jpeg"].description El resultado sería: JPEG Image
enabledPlugin	Referencia al objeto plugin configurado por el tipo MIME. Si no hay ninguno, la propiedad vale *null*.	navigator.mimeTypes["image/jpeg"].enabledPlugins
suffixes	String que contiene la lista de extensiones aceptadas por el tipo MIME.	navigator.mimeTypes["image/jpeg"].suffixes El resultado sería: jpeg, jpg, jpe, jfif, pjpeg, pjp
type	Nombre del tipo MIME.	navigator.mimeTypes["image/jpeg"].type El resultado sería: image/jpeg

El siguiente ejemplo muestra las propiedades para cada objeto MimeType del cliente.

```
<html>
<head>
</head>
<body>
<h1>Propiedades de cada objeto mimeType del cliente:</h1>
<script type="text/javascript">
   document.writeln("<table border=1><tr valing=top>"+ "<th align=left>i"+ " <th align=left>type"+
   "<th align=left>description"+ " <th align=left>suffixes"+
   " <th align=left>enabledPlugin.name </tr>")
   for (i=0; i < navigator.mimeTypes.length; i++) {
      document.writeln("<tr valing=top><td tipus='fuoc'>"+i+" <td tipus='fuoc'>"+
      navigator.mimeTypes[i].type+ " <td tipus='fuoc'>"+
      navigator.mimeTypes[i].description+" <td tipus='fuoc'>"+navigator.mimeTypes[i].suffixes)
      if (navigator.mimeTypes[i].enabledPlugin==null) {
         document.writeln("<td tipus='fuoc'>None"+ " </tr>")
      } else {
```

```
            document.writeln("<td tipus='fuoc'>"+navigator.mimeTypes[i].enabledPlugin.name+" </tr>")
        }
    }
    document.writeln("</table>")
</script>
</body>
</html>
```

3.7. El objeto Plugin

Cada objeto Plugin corresponde a un componente instalado en el navegador. Estos objetos están disponibles mediante la propiedad *enabledPlugin* de objetos MimeType. Cada *plugin* proporciona información sobre el componente como su descripción, los tipos MIME que soporta, etc.

Este objeto se suele utilizar para determinar si el navegador soporta un componente *plugin* específico y su versión.

A continuación, se presentan las principales propiedades del objeto:

Tabla 18. Propiedades del objeto Plugin

Nombre	Descripción
description	Descripción del *plugin*. Es de solo lectura.
filename	Nombre del plugin en el disco. Es de solo lectura.
length	Número de elementos del *array* de objetos MimeType. Es de solo lectura.
name	Nombre del *plugin*. Es de solo lectura.

El siguiente ejemplo muestra las propiedades para cada *plugin* instalado en el cliente.

```
<html>
<head>
</head>
<body>
<b>Propiedades para cada conector instalado en el cliente:</b><p>
<script type="text/javascript">
   document.writeln("<table botder=1><tr valign=top>"+ "<th valign=left>i"
   + "<th valign=left>name" + "<th valign=left>filename" + "<th valign=left>description"+
   "<th valign=left># of types</tr>")
   for (i=0; i < navigator.plugins.length; i++) {
      document.writeln("<th valign=top><td tipus='fuoc'>"+i+ "<td tipus='fuoc'>",
      navigator.plugins[i].name+ "<td tipus='fuoc'>"+ navigator.plugins[i].filename+
```

```
        "<td tipus='fuoc'>"+navigator.plugins[i].description+ "<td tipus='fuoc'>"+
        navigator.plugins[i].length+ "</tr>")
    }
    document.writeln("</table>")
</script>
</body>
</html>
```

4. Los objetos de DOM

En este apartado, se va a bajar un nivel, hasta llegar al objeto Document, de manera que siguiendo la nomenclatura planteada se puede considerar que se va a estudiar el DOM. Igual que en el apartado anterior, la lista de objetos presentados es dinámica y evoluciona en paralelo a las distintas versiones de los navegadores.

Web recomendada

Para información más detallada y actualizada de las propiedades y los métodos de los objetos del DOM, podéis visitar la página de w3schools:

http://www.w3schools.com/jsref/

4.1. El objeto Document

El objeto Document proporciona acceso al contenido del documento HTML y proporciona métodos para su manipulación. Para cada página HTML, se genera un objeto Document cuando esta se carga en una ventana; de esta manera, todo objeto Window contiene un objeto Document al que puede acceder utilizando su propiedad *document*. El objeto Document se construye a partir de la etiqueta "body" de HTML.

En la siguiente tabla, se muestran las principales propiedades del objeto:

Tabla 19. Propiedades del objeto Document

Nombre	Descripción	Ejemplo
anchors	*Array* de objetos anchor del documento. Es de solo lectura.	
applets	*Array* de objetos applet del documento. Es de solo lectura.	
cookie	String que representa las cookies asociadas al documento.	
doctype	Devuelve un objeto del tipo doctype del documento HTML actual.	alert(document.doctype.name);
domain	Cadena que contiene el nombre del dominio desde el que se cargó el documento.	document.domain="mundo.com"
embeds	*Array* de todos los objetos embebidos en el documento. Es de solo lectura.	
forms	*Array* de objetos form del documento. Es de solo lectura.	document.forms[i] o lo que es equivalente: document.form_i
images	*Array* de objetos image del documento. Es de solo lectura.	document.images[i] o lo que es equivalente: document.image_i
lastModified	String que representa la fecha de la última modificación del documento. Es de solo lectura.	document.write("Esta página ha sido actualizada " + document.lastModified)
links	*Array* de objetos link del documento. Es de solo lectura.	document.links[i]

Nombre	Descripción	Ejemplo
plugins	*Array* de objetos plugin del documento. Es de solo lectura.	document.plugins[i]

referrer	URL del documento que se llama al pulsar un link.
	Es de solo lectura.
		function obtenerRef() {
		return document.referrer
		}

title	String que representa el título del documento. Es de solo lectura.	var ventana1 = window.open("http://www.uoc.es") var titulo = ventana1.document.title
URL	String que representa la URL completa del documento. Es de solo lectura.	document.write("La URL actual es " + document.URL)

Respecto a las propiedades presentadas, se deben tener en cuenta los siguientes detalles:

- El orden de los elementos en los *arrays*, para todas las propiedades que lo contienen, es el mismo orden en el que aparecen estos elementos en la página.

- La propiedad *domain* solo puede modificarse de manera restringida. Inicialmente, contiene el dominio del servidor web desde el que ha sido cargada la página. Se puede modificar esta propiedad, pero solo por un dominio con el mismo sufijo. Por ejemplo, un *script* proveniente de todo.mundo.com puede cambiar el dominio por mundo.com, pero un *script* proveniente de stars.com no puede. Una vez se ha cambiado el valor de la propiedad, no se puede devolver a su valor original.

- La propiedad *lastModified* se deriva de los datos de la cabecera HTTP enviada por el servidor. El servidor normalmente obtiene este dato examinando la fecha de la última modificación del fichero. Esta información no tiene por qué existir necesariamente en la cabecera. En tal caso, JavaScript recibe un 0 que visualizaría la fecha January 1, 1970 GMT.

- La propiedad *referrer* está vacía si el servidor no provee la variable de entorno que contiene esta información.

A continuación, se muestra un subconjunto de los principales métodos disponibles en el objeto:

Tabla 20. Métodos del objeto Document

Nombre	Descripción	Sintaxis	Parámetros
close	Finaliza el flujo de salida al documento y muestra el contenido escrito.	close()	
open	Abre el documento para la escritura.	open([tipoMime, "replace"])	tipoMime: string que representa el tipo del documento. Por defecto, es text/html. "replace": si se omite esta palabra, el tipo MIME es text/html. En caso contrario, el nuevo documento no se añade al historial.

Nombre	Descripción	Sintaxis	Parámetros
write	Escribe una expresión HTML en el documento.	document.write(expr1, ..., exprN)	
writeln	Escribe una expresión HTML terminada en salto de línea.	writeln(expr1, ... exprN)	

De los métodos presentados, se deben considerar los siguientes aspectos:

- El método close finaliza la carga del documento que había empezado con el método open; cuando esto sucede, en la barra de estado del navegador aparece la frase “Listo”.

- Los posibles valores para el primer parámetro del método open son:
  - text/html: especifica que el documento contiene texto ASCII en formato HTML.
  - text/plain: especifica que el documento contiene texto ASCII plano con caracteres de fin de línea para el texto que se visualiza.
  - image/gif: especifica que el documento contiene bits codificados que constituyen cabeceras GIF y datos de píxel.
  - image/jpeg: especifica que el documento contiene bits codificados que constituyen cabeceras JPG y datos de píxel.
  - image/x-bitmap: especifica que el documento contiene bits codificados que constituyen cabeceras *bitmap* y datos de píxel.
  - Valores para *plugins* específicos, por ejemplo, “application/x-director” carga el *plugin* de Macromedia Shockwave. Estos solo son válidos si el *plugin* está instalado.

El siguiente ejemplo muestra el proceso en el que se crea una nueva ventana sobre la que se va a escribir texto a partir de los métodos de Document:

```
<html>
<head>
<script type="text/javascript">
   var miVentana
   function escribir_en_ventana() {
      var texto = "Ejemplo para el objeto Document"
      miVentana.document.open("text/html", "replace")
      miVentana.document.write("<p>" + texto)
      miVentana.document.write("<p>history.length es " + miVentana.history.length)
      miVentana.document.close()
   }
</script>
</head>
<body>
<script type="text/javascript">
   miVentana=window.open("","",'toolbar=yes,scrollbars=yes,width=400,height=300')
```

```
    escribir_en_ventana()
</script>
</body>
</html>
```

4.2. El objeto a, Anchor, Link y Area

En los modelos tradicionales, había un objeto distinto para elementos <a> que especificaban una propiedad *name* (llamada Anchor) y otro para los que especificaban una propiedad *href* (llamada Link). Esta nomenclatura es anticuada, y con el DOM estándar ya no existe ninguna distinción, ya que se fusionan Anchor y Link.

Por otra parte, el objeto Area corresponde a un elemento <area> de HTML (que representa un área de mapa de imagen); el acceso a este objeto se realiza a partir del *array* links[ ] del objeto Document.

Al tratarse de objetos que hacen referencia a una etiqueta HTML, comparten un conjunto de propiedades y métodos con el resto de etiquetas. Las siguientes dos tablas muestran las más interesantes y, como se puede intuir, forman parte de las propiedades y los métodos de todas las etiquetas HTML:

Tabla 21. Propiedades de etiquetas HTML

Nombre	Descripción	Ejemplo
attributes[]	*Array* con los atributos del elemento.	Area2.attributes[2]; muestra el tercer atributo si lo tiene.
disabled	Valor lógico que indica si el elemento está deshabilitado.	Disponible = Area2.disabled;
id	Cadena que contiene el identificador único del elemento.	Ident = Area2.id;
style	Hace referencia al objeto Style insertado en línea del elemento	
tabIndex	Valor numérico que indica el orden de tabulación del objeto.	

A las propiedades anteriores es necesario añadir todas aquellas que se han presentado en el segundo apartado de este módulo relacionadas con el DOM estándar (por ejemplo, nodeValue, nodeType, nodeName, nextSibling, etc.).

La siguiente tabla muestra los métodos más usados de los disponibles en el objeto:

Tabla 22. Métodos de etiquetas HTML

Nombre	Descripción	Sintaxis	Retorno
blur()	Quita el foco del elemento.	Area2.blur():	

Nombre	Descripción	Sintaxis	Retorno
click()	Simula un clic del ratón sobre el objeto.	Area2.click();	
focus()	Asigna el foco al elemento.	Area2.focus();	

De la misma manera que en las propiedades, a los métodos anteriores se deben añadir los métodos del DOM estándar (por ejemplo, cloneNode(), getAttribute(), hasChildNodes(), etc.).

A las propiedades y los métodos anteriores se deben añadir los siguientes, que son específicos de los objetos Anchor, Link y Area:

Tabla 23. Propiedades y métodos específicos de los objetos Anchor, Link y Area

Nombre	Descripción	Ejemplo
hash	String empezado por # que especifica un nombre de ancla en la URL.	Si la URL es: http://mundo.com/europa.htm#italia document.links[0].hash es: #italia
host	String que especifica el dominio, subdominio del servidor.	
hostname	String que contiene el nombre completo del *host* incluyendo el nombre del servidor, el subdominio, el dominio y el puerto.	
href	String que especifica la URL entera.	
pathname	String que contiene la porción de URL referente a la ruta.	
port	Sring que representa el puerto de comunicaciones que usa el servidor.	
protocol	String que contiene la porción de URL referente al protocolo.	
search	String que especifica una búsqueda contenida en la URL.	
target	Nombre de la ventana que visualizará el contenido del *hyperlink*.	document.aindice.target="ventana1"
text	(Solo para el objeto area). Texto correspondiente a la etiqueta A.	

Respecto a los métodos anteriores, cabe prestar especial atención a la hora de usar la propiedad *hash*, ya que si la URL es:

```
http://mundo.com/europa.htm#italia
```

Y se realiza la siguiente asignación:

```
h = document.links[0].hash;
document.links[0].hash = h;
```

El valor que toma la propiedad es el siguiente:

```
##italia
```

4.3. El objeto Image

Un objeto Image corresponde a una etiqueta <img> de HTML. Este objeto dispone de propiedades que permiten la manipulación dinámica de las imágenes en el documento. Se accede a un objeto Image a partir de la colección images[ ] del objeto Document.

El objeto Image se construye con los siguientes mecanismos:

- La etiqueta <img> de HTML.

- La forma: new Image(ancho, alto), en la que los parámetros son opcionales e indican el valor de ancho y alto de la imagen en píxeles.

Al tratarse de un objeto basado en una etiqueta HTML, compartirá las propiedades y los métodos presentados en el subapartado anterior y los siguientes que le son específicos:

Tabla 24. Propiedades del objeto Image

Nombre	Descripción	Ejemplo

border	String que especifica el ancho en píxeles del borde de la imagen.
	Es de solo lectura.
	(No soportado en HTML 5).
		`function bordeImg(imagen) {`
		`  if (imagen.border==0)`
		`  alert('La imagen no tiene borde')`
		`  else alert('El borde de la imagen es '`
		`  + imagen.border)`
		`}`

complete	Valor booleano que indica cuándo el navegador a terminado de cargar la imagen. Es de solo lectura.	
height	String que especifica el alto en píxeles de la imagen. Es de solo lectura.	
hspace	String que especifica el espacio en píxeles entre el borde derecho e izquierdo de la imagen y el texto que la sigue o la precede. Es de solo lectura. (No soportado en HTML 5).	
name	String que especifica el nombre de la imagen. Es de solo lectura. (No soportado en HTML 5, cabe utilizar el atributo id).	
src	String que especifica la URL de la imagen.	
vspace	String que especifica el espacio en píxeles entre el borde superior e inferior de la imagen y el texto que la sigue o la precede. Es de solo lectura. (No soportado en HTML 5).	

Nombre	Descripción	Ejemplo
width	String que especifica el ancho en píxeles de la imagen. Es de solo lectura.	

Las propiedades referentes al alto y el ancho de una imagen pueden establecerse en la creación del objeto Image, pero no pueden ser modificadas desde JavaScript.

4.4. El objeto Form

El objeto corresponde a la etiqueta <form> de HTML y es hijo del objeto Document. Este objeto es utilizado para incluir campos de texto, elementos de selección (botones de selección, listas de selección, etc.) y botones, y su objetivo final es la recopilación de información por parte del usuario para ser procesada y enviada al servidor.

Cada formulario en un documento es, en sí mismo, un objeto distinto. Para hacer referencia a los elementos de un formulario, será conveniente utilizar el atributo "id".

Igual que los objetos anteriores, se trata de un objeto relacionado con una etiqueta HTML, por lo que comparte las propiedades y los métodos de los objetos HTML; se plantean los siguientes:

Tabla 25. Propiedades del objeto Form

Nombre	Descripción	Ejemplo
action	String que especifica la URL destino cuando se envían los datos del formulario.	`document.miForm.action = " mailto://pepe@jet.es"`
elements	*Array* de objetos correspondientes a los elementos del formulario. Es de solo lectura.	
encoding	String que especifica la codificación MIME del formulario.	`function obtenerCodif() { return document.miForm.encoding }`
length	Número de elementos del formulario. Es de solo lectura.	`numele = miForm.elements.length`
method	String que especifica cómo se envía la información del formulario al servidor. Puede tener los valores *get* y *post*.	`function obtenerMetodo() { return document.miForm.method }`
name	String que especifica el nombre del formulario.	`for (var i = 0; i < document.miForm.elements.length; i++){ document.write(document.miForm.elements[i].name + " ") }`
target	String que especifica la ventana en la que se visualizará la respuesta, una vez el formulario haya sido enviado.	`document.miForm.target = "ventana1"`

Se pueden usar los siguientes mecanismos (tradicionales) para acceder a los elementos de un formulario:

```
nombre_formulario.nombre_elemento.value
nombre_formulario.elements[posición].value
```

Tabla 26. Métodos del objeto Form

Nombre	Descripción	Sintaxis
reset	Simula un clic de ratón en un botón de tipo *reset*.	reset()
submit	Envía el formulario.	submit()

Respecto a los métodos anteriores, es necesario apuntar que el método reset restaura los valores por defecto en los elementos del formulario. Para usarlo, no es necesario haber definido un botón en el formulario, se puede llamar desde una función.

A continuación, se presenta un ejemplo en el que se solicita al usuario la introducción de los valores A o B. Si el dato introducido es correcto, se muestra un mensaje que lo indica; en caso contrario, el mensaje muestra los posibles valores que hay que introducir e inicializa el formulario.

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
function verificarEntrada(letra) {
   if (letra.value == 'A' || letra.value == 'B') alert('Entrada correcta')
   else document.miForm.reset()
}
</script>
</head>
<body bgcolor="#FFFFFF">
<form name="miForm" onReset="alert('Introduzca A o B')">
Introduzca A o B:
<input type="text" name="opcion" size="5" onChange=verificarEntrada(this) maxlength="1"><P>
</form>
</body>
</html>
```

A continuación, se van a plantear todos los objetos que pertenecen o forman parte de un formulario. Al tratarse de objetos basados en etiquetas HTML, compartirán las propiedades y los métodos presentados anteriormente.

4.4.1. Los objetos Hidden, Text, Textarea, Password, URL, Search y Email

Se plantean los objetos de manera conjunta, ya que comparten propiedades y métodos; en particular, los objetos Hidden, Text, Password, URL, Search y Email corresponden a la etiqueta HTML <input>, en la que su diferencia aparece por el atributo "type". De esta manera, se tiene lo siguiente:

- El objeto Hidden es un campo más del formulario, pero no es visible cuando se visualiza la página en el navegador. Al no ser visible, el usuario no puede modificar su valor, solo se puede modificar desde la programación cambiando el valor de la propiedad *value*. Se utiliza para enviar pares de datos (nombre/valor) cuando se hace un *submit* del formulario.

- El objeto Text es un campo del formulario en el que el usuario puede introducir texto.

- Los objetos URL, Search y Email son objetos similares al objeto Text; el hecho de diferenciarlos de Text nos puede ayudar a la hora de realizar validaciones, o incluso puede hacer que el navegador trabaje con estos campos de manera distinta de como lo haría con un objeto Text, por ejemplo, en el navegador de un dispositivo móvil con objeto del tipo URL adaptará su teclado para poner más accesibles las teclas con el contenido típico de una URL, como podrían ser "/" o ".com".

- El objeto Password es un campo del formulario en el que el usuario puede introducir texto. Cada carácter introducido se visualiza en la pantalla mediante un asterisco.

- El objeto Textarea es un campo del formulario en el que el usuario puede introducir texto en varias líneas. Para indicar en el texto introducido un salto de línea, debe especificarse el carácter concreto para ello. Este carácter varía según la plataforma: en Unix es \n; en Windows es \r; y en Macintosh es \n. JavaScript comprueba estos posibles caracteres y los traslada a la plataforma del usuario. El objeto Textarea se construye mediante la etiqueta <textarea> de HTML.

Los objetos Hidden, Text, Password, URL, Search y Email se construyen mediante la etiqueta <input> de HTML, con los valores "hidden", "text", "password","url", "search" y "email" en el atributo "type".

Las siguientes tablas muestran las propiedades y los métodos específicos de estos objetos:

Tabla 27. Propiedades de los objetos Hidden, Text, Textarea, Password, URL, Search y Email

Nombre	Descripción	Ejemplo
form	Referencia al objeto *form* que lo contiene. Es de solo lectura.	<input type=hidden name=”anio” value="2000"> … <input name="b1" type="button" value="Cambiar año" onClick="this.form.anio.value = ‘2001’ ">
name	String que especifica el nombre del objeto.	document.miForm.elements[i].name
placeholder	Cadena que aparecerá en el objeto a modo de ejemplo mientras el valor del objeto esté vacío (esta propiedad no está disponible para el objeto Hidden).	document.miForm.elements[i].placeholder = “Escriba su nombre”
readOnly	Valor booleano que indica si el contenido del objeto es editable o no (esta propiedad no está disponible para el objeto Hidden).	document.miForm.elements[i].readOnly = true
type	String que especifica el tipo del objeto. Es de solo lectura.	document.miForm.elements[i].type
value	String que especifica el valor del objeto.	document.write("El valor del campo es " + document.miForm.anio.value + " ")

Tabla 28. Métodos de los objetos Hidden, Text, Textarea, Password, URL, Search y Email

Nombre	Descripción	Sintaxis
blur	Elimina el foco del campo de texto.	blur()
focus	Asigna el foco al campo de texto.	focus()
select	Selecciona el área de entrada del campo de texto.	select()

Respecto a los métodos anteriores, cabe indicar que el método select ilumina el área de texto y puede usarse para situar el cursor en el sitio en el que el usuario debe realizar la entrada de datos. Es más cómodo para el usuario cuando al empezar a introducir un dato se sustituye directamente todo el dato ya existente.

4.4.2. El objeto Range y Number

Estos objetos también se corresponden a un elemento <input> y se definen asignando con los valores “range”, “number” en atributo “type”. Estos objetos comparten las propiedades vistas en el punto anterior, exceptuando el objeto Range que no dispone de la propiedad *readOnly*.

Dentro de un formulario, un campo Number sirve para introducir valores numéricos y puede hacer que el navegador modifique su funcionalidad, por ejemplo, añadiendo al lado del campo dos botones para realizar el incremento o decremento del valor; o, en el caso de un dispositivo móvil, al intentar introducir un valor hará que este nos muestre directamente el teclado numérico. La línea para especificar un campo del tipo Number sería:

```
<input type="number" max="100" step="10">
```

Figura 13. Ejemplo de un campo Number

Los campos Range sirven para elegir un valor dentro de un rango utilizando para ello un deslizador. La línea para especificar un campo del tipo Number sería:

```
<input type="range" id="myRange" value="75">
```

La visualización de este elemento dependerá del navegador; a continuación, un ejemplo de cómo se visualiza un campo Range en Internet Explorer y en Firefox:

Figura 14. Visualización de un campo Number en Firefox e Internet Explorer

Las propiedades que añaden los objetos Range y Number son:

Tabla 29. Propiedades adicionales de Range y Number

Nombre	Descripción	Sintaxis
max	Valor numérico que sirve para indicar el valor máximo del objeto.	document.miForm.elements[i].max = 100
min	Valor numérico que sirve para indicar el valor mínimo del objeto.	document.miForm.elements[i].min = 10
step	Valor numérico que sirve para especificar los intervalos válidos. Por ejemplo si a la propiedad *min* le hemos puesto el valor de 2 y asignamos un *step* con el valor 3, el primer valor válido será el 2, el siguiente el 5 y así sucesivamente, en caso de no asignar un valor a min se tomará como valor de partida el 0.	document.miForm.elements[i].step = 5

4.4.3. El objeto FileUpload, File

Este objeto corresponde a un elemento <input> con el atributo "type" file; su misión es la inclusión de un fichero en los datos que envía el formulario al servidor.

Dentro de un formulario, la línea que especifica el campo fileUpload sería por ejemplo:

```
<input type="file" size=50 name="fileu">
```

Figura 15

De manera que, al pulsar el botón *Examinar*, se abre la ventana de diálogo que permite buscar y seleccionar el fichero que se desea adjuntar.

A continuación, se muestran las principales propiedades del objeto:

Tabla 30. Propiedades del objeto FileUpload, File

Nombre	Descripción
form	Referencia al objeto *form* que lo contiene. Es de solo lectura.
multiple	Valor booleano que indica si el usuario puede seleccionar más de un fichero o no.
name	String que especifica el nombre del objeto.
type	String que especifica el tipo del objeto. Es de solo lectura.
value	String que especifica el valor del objeto.

4.4.4. Los objetos Button, Reset y Submit

Estos objetos corresponden a un elemento HTML <input> con el atributo type "button", "reset" y "submit", respectivamente. Los tres objetos se representan como un botón y sus características son las siguientes:

- El objeto Button representa un botón del formulario que no tiene ninguna acción predefinida.

- El objeto Reset representa un botón del formulario que tiene la particularidad de restablecer los valores que por defecto tienen los elementos del formulario.

- El objeto Submit representa un botón del formulario que tiene la particularidad de enviar los datos del formulario.

A continuación, se presentan las principales propiedades y los métodos de estos objetos:

Tabla 31. Propiedades de los objetos Button, Reset y Submit

Nombre	Descripción
form	Referencia al objeto *form* que lo contiene. Es de solo lectura.
name	String que especifica el nombre del objeto.

Nombre	Descripción
type	String que especifica el tipo del objeto. Es de solo lectura.
value	String que especifica el valor del objeto.

Tabla 32. Métodos de los objetos Button, Reset y Submit

Nombre	Descripción	Sintaxis
click	Simula la pulsación del botón.	click()

4.4.5. El objeto Radio

Este objeto corresponde a un campo de entrada de formulario de <input> con el atributo type "radio". El objeto representa un botón de opción individual dentro de un grupo de botones de opción del formulario.

Para entender mejor su uso, a continuación se plantea un ejemplo en el que hay un formulario que contiene 4 objetos Radio dentro de un mismo grupo. Esto significa que el hecho de tener uno de ellos seleccionado implica que el resto no lo esté.

Para indicar que pertenecen al mismo grupo, el valor del parámetro *name* debe ser igual para todos, y para determinar qué opción estará marcada por defecto se utiliza la palabra *checked* como parámetro.

```
¿A que grupo de edad pertenece?<br><br>
<input type="radio" name="edad" value="1"> < 18
<input type="radio" name="edad" value="2"> 18 - 25
<input type="radio" name="edad" value="3" checked> 26 - 35
<input type="radio" name="edad" value="4"> > 35
```

El resultado en la ventana del navegador es el siguiente:

Figura 16

A continuación, se presentan las principales propiedades y los métodos del objeto Radio:

Tabla 33. Propiedades del objeto Radio

Nombre	Descripción	Ejemplo
checked	Valor booleano que especifica si el botón está seleccionado o no (*true* si lo está y *false* en caso contrario).	if (document.miForm.musica[0].checked == true) { alert("Está seleccionada la opción música clásica") }

Nombre	Descripción	Ejemplo
defaultChecked	Valor booleano que especifica el estado por defecto del botón (*true* si está seleccionado por defecto y *false* en caso contrario).	
form	Referencia al objeto *Form* que lo contiene. Es de solo lectura.	
length	Número de objetos radio en el grupo.	for (i=0; i<miform.edad.length; i++){ if (miform.edad[i].checked) alert(i) }
name	String que especifica el nombre del objeto.	
type	String que especifica el tipo del objeto. Es de solo lectura.	
value	String que especifica el valor asociado al botón.	

Respecto a las propiedades anteriores, cabe tener en cuenta que la propiedad *value* de un botón de opción no se visualiza en el documento, es el valor que se enviará con el resto de datos del formulario al servidor.

Tabla 34. Métodos del objeto Radio

Nombre	Descripción	Sintaxis
click	Simula la pulsación del botón.	click()

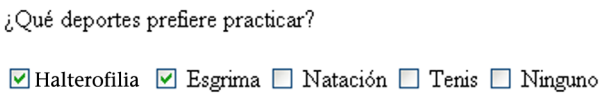
4.4.6. El objeto Checkbox

Igual que el objeto anterior, el objeto Checkbox corresponde a un elemento HTML <input>, con el atributo type en "checkbox", y se representa como una casilla que el usuario puede marcar o de la que puede quitar la marca según le interese.

En el siguiente ejemplo, el formulario contiene 5 objetos Checkbox dentro de un mismo grupo. En este caso, y a diferencia del grupo de objetos Radio, se permite seleccionar más de una opción en paralelo. Para determinar qué opción estará marcada por defecto se utiliza la palabra *checked* como parámetro.

```
¿Qué deportes prefiere practicar?<br><br>
<input type="checkbox" name="deportes" value="1" checked> Halterofilia
<input type="checkbox" name="deportes" value="2" checked> Esgrima
<input type="checkbox" name="deportes" value="3"> Natación
<input type="checkbox" name="deportes" value="4"> Tenis
<input type="checkbox" name="deportes" value="5"> Ninguno
```

Figura 17



A continuación, se presentan las principales propiedades y los métodos del objeto Checkbox:

Tabla 35. Propiedades del objeto Checkbox

Nombre	Descripción
checked	Valor booleano que especifica si el botón está seleccionado o no (*true* si lo está y *false* en caso contrario).
defaultChecked	Valor booleano que especifica el estado por defecto del botón (*true* si está seleccionado por defecto y *false* en caso contrario).
form	Referencia al objeto *form* que lo contiene. Es de solo lectura.
name	String que especifica el nombre del objeto.
type	String que especifica el tipo del objeto. Es de solo lectura.
value	String que especifica el valor asociado al botón.

Respecto a las propiedades anteriores, igual que en el objeto Radio, la propiedad *value* no se visualiza en el documento, es el valor que se enviará con el resto de datos del formulario al servidor.

Tabla 36. Métodos del objeto Checkbox

Nombre	Descripción	Sintaxis
click	Simula la pulsación del botón.	click()

4.4.7. El objeto Select

Este objeto corresponde al elemento HTML <select>. Se trata de una lista de selección del formulario en la que el usuario puede seleccionar una o más opciones en función de los atributos con los que aquel ha sido creado.

A continuación, se presentan las principales propiedades y los métodos del objeto Select:

Tabla 37. Propiedades del objeto Select

Nombre	Descripción	Ejemplo
form	Referencia al objeto *form* que lo contiene. Es de solo lectura.	
length	Número de opciones en la lista.	num = document.miForm.lista.length
name	String que especifica el nombre del objeto.	for (var i = 0; i < document.miForm.elements.length; i++) { document.write(document.miForm.elements[i].name + " ") }
options	*Array* correspondiente a las opciones de la lista ordenadas según aparecen.	miLista.options[1] = null

Nombre	Descripción	Ejemplo
selectedIndex	Número que indica la posición de la opción seleccionada.	function ObtenerIndice() { return document.miForm.lista.selectedIndex }
type	String que especifica el tipo del objeto. Es de solo lectura.	

De las propiedades anteriores, es fundamental tener en cuenta los siguientes detalles:

- Si en un objeto Select no hay ningún elemento seleccionado, el valor de la propiedad *selectedIndex* es –1.

- En el caso de poder seleccionar varios elementos de la lista, la propiedad *selectedIndex* contiene la posición del primer elemento de la lista que está seleccionado.

- En el parámetro *type*, se especifica si en la lista se podrán seleccionar varios elementos o solo uno. Los valores para indicarlo son: "select-multiple" y "select-one".

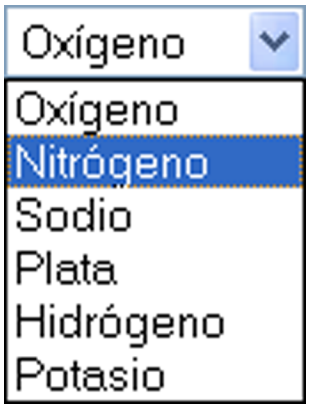
- Igual que los objetos anteriores, la propiedad *value* no se visualiza en el documento, es el valor que se envía con el resto de datos del formulario.

El siguiente ejemplo muestra cómo se realiza la construcción de un objeto Select:

```
<select name="elementos">
   <option value="O">Oxígeno
   <option value="N">Nitrógeno
   <option value="Na">Sodio
   <option value="Ag">Plata
   <option value="H">Hidrógeno
   <option value="K">Potasio
</select>
```

Como se puede observar, cada uno de los elementos de la lista es un objeto Option. El resultado en pantalla es el que se observa a continuación:

Figura 18



4.4.8. El objeto Option

Este objeto corresponde a una etiqueta HTML <option>. Representa un elemento en una lista de selección del formulario.

Los valores "option" se pueden construir utilizando HTML, o en JavaScript tal como se describe a continuación:

- Mediante la etiqueta <option> de HTML.

- Mediante el constructor new option(text, value, defaultSelected, selected), cuyos parámetros son opcionales y tienen el siguiente significado:
  - text: especifica el texto que se verá en la lista;
  - value: especifica el valor que se enviará con los datos del formulario si la opción está seleccionada;
  - defaultSelected: especifica el valor inicial de la opción, *true* si es seleccionado y *false* en caso contrario;
  - selected: especifica el estado actual de la opción, es decir, si está seleccionada o no.

A continuación, se presentan las principales propiedades del objeto Option:

Tabla 38. Propiedades del objeto Option

Nombre	Descripción	Ejemplo
defaultSelected	Valor booleano que indica si la opción está seleccionada o no por defecto. El valor *true* indica que sí y el valor *false* que no.	`function restauraLista() { for (var i = 0; i < document.miForm.lista.length;i++) { if ( document.miForm.lista.options[i].defaultSelected == true ) document.miForm.lista.options[i].selected = true else document.miForm.lista.options[i].selected = false }`

Nombre	Descripción	Ejemplo
selected	Valor booleano que indica si la opción está seleccionada. El valor *true* indica que sí y el valor *false* que no.	`function restauraLista() { for (var i = 0; i < document.miForm.lista. length; i++) { if ( document.miForm.lista.options[i]. defaultSelected == true ) document.miForm.lista.options[i]. selected = true else document.miForm.lista.options[i]. selected= false } }`
text	Texto que se muestra en la lista.	
value	Especifica el valor que se enviará con los datos del formulario si la opción está seleccionada.	

5. Gestión de eventos

Un evento es una acción que ocurre en la mayoría de las ocasiones por algo que hace el usuario, por ejemplo, la pulsación de un botón, el movimiento del ratón, etc. Por su parte, el manejador de eventos es el código JavaScript, que se asocia a un evento, de manera que cuando este se da, se ejecuta el código.

No existe un único modelo de gestión de eventos, sino básicamente dos que han evolucionado paralelamente a la evolución de los modelos de objetos de los principales navegadores.

En el modelo tradicional del modelo de objetos, ambos navegadores ofrecían un soporte muy básico para manejar eventos. Estos se basaban prácticamente en eventos generados en formularios y algunos eventos de sistema o del navegador.

A partir de la aparición de las versiones 4.x de los navegadores, el modelo de gestión de eventos se amplió considerablemente, y se añadieron nuevos eventos y una mayor funcionalidad; pero de la misma manera que los modelos de objetos divergían, también lo hicieron los modelos de eventos.

La convergencia se produce con la aparición del modelo de eventos estándar de W3C. Este modelo se puede consultar en el estándar "DOM2, modelo de eventos". Tal como se verá a continuación, el modelo estándar adquiere las ventajas de cada uno de los modelos anteriores, creando de esta manera un modelo robusto y muy completo de control de eventos.

En los siguientes subapartados se realizará una revisión histórica del modelo de eventos, desde su principio hasta la aparición del modelo estándar del DOM2 del W3C.

5.1. Modelo básico de control de eventos

El modelo básico de eventos está compuesto por un conjunto de eventos que pueden asociarse a un tipo de objetos en concreto. La captura de estos eventos se realiza a partir de la vinculación de estos eventos con los objetos en los que se producen.

La siguiente tabla muestra el conjunto básico de eventos disponible en la mayoría de los navegadores e indica los elementos que los admiten:

Tabla 39. Conjunto básico de eventos

Controlador de evento	Descripción	Objetos que lo admiten
onabort	Se produce cuando el usuario interrumpe la carga de la imagen.	<img>
onblur	Se produce cuando un elemento pierde el foco.	Todos los elementos HTML excepto <base>, <bdo>, , <head>, <html>, <iframe>, <meta>, <param>, <script>, <style> y <title>
onchange	Se produce cuando un campo de formulario ha perdido el foco y su valor ha cambiado.	<input>, <keygen>, <select> y <textarea>
onclick	Se produce cuando se pulsa sobre un objeto.	Todos los elementos visibles
ondblclick	Se produce cuando se hace una doble pulsación sobre el objeto.	Todos los elementos visibles
onfocus	Se produce cuando un objeto recibe el foco.	Todos los elementos HTML excepto <base>, <bdo>, , <head>, <html>, <iframe>, <meta>, <param>, <script>, <style> y <title>
onkeydown	Se produce cuando se pulsa una tecla.	Todos los elementos visibles
onkeypress	Se produce cuando se mantiene pulsada una tecla.	Todos los elementos visibles
onkeyup	Se produce cuando se libera una tecla pulsada.	Todos los elementos visibles
onload	Se produce cuando el navegador termina de cargar una ventana o un conjunto de marcos.	<body>, <frame>, <iframe>, <img>, <input type="image">, <link>, <script> y <style>
onmousedown	Se produce cuando se pulsa un botón del ratón.	Todos los elementos visibles
onmousemove	Se produce cuando se mueve el ratón mientras se está sobre el objeto.	Todos los elementos visibles
onmouseout	Se produce cuando el puntero abandona el área de un objeto.	Todos los elementos visibles
onmouseover	Se produce cuando el puntero entra en un área de un objeto.	Todos los elementos visibles
onmouseup	Se produce cuando se libera un botón pulsado del ratón.	Todos los elementos visibles
onreset	Se produce cuando se limpian los campos de un formulario.	<form>
onselect	Se produce cuando se selecciona el contenido de un campo de texto o de un área de texto en un formulario.	<input>, <textarea> y <keygen>
onsubmit	Se produce cuando se envía un formulario.	<form>
onunload	Se produce cuando se abandona la página.	<body>, <frameset>

Además de los anteriores eventos, los distintos fabricantes han implementado un conjunto más amplio de eventos propietarios. Evidentemente, la lista va aumentando en cada nueva versión, y en este apartado solo se están mostrando los eventos estándar o más importantes.

Internet Explorer

Por ejemplo, Internet Explorer dispone de un conjunto de eventos que le permiten capturar acciones de ratón más complejas, eventos de elementos como el movimiento del texto de la etiqueta <marquee> y eventos de vinculación de datos que permiten la carga de datos.

Como se observa en la tabla, además de que tratan acciones de ratón, existen eventos de navegador o de sistema como *load* y *unload*, dos eventos bastante útiles.

Load* y *unload

Por ejemplo, el evento *load* se utiliza para realizar una precarga de imágenes, de manera que estas quedan en la caché del navegador, lo que provoca que se presenten de manera más rápida cuando sean requeridas. Del mismo modo, el evento *unload* puede ser utilizado para realizar una limpieza de la memoria y habilitar así espacio en la caché del navegador.

La vinculación de eventos en el modelo tradicional es muy sencilla; se pueden implementar controladores de eventos en las etiquetas HTML y mediante objetos en JavaScript. En ambos casos, solo será posible implementar el controlador si la etiqueta o el objeto sobre el que se quiere aplicar admite el evento en cuestión.

5.1.1. Vinculación en etiquetas HTML

Si se aplica un evento a una etiqueta HTML, es necesario implementar un controlador de evento en la etiqueta. Para implementarlo, se debe añadir un atributo a la etiqueta que lo identificará. La sintaxis general para ello es:

```
<etiqueta controlador_de_evento="Código JavaScript">
```

En el siguiente ejemplo, se implementa un controlador para el evento *mouseover* en la etiqueta `<a>`:

```
<a href="http://www.uoc.es" onmouseover="miFuncion()">
```

En el ejemplo anterior, el paso del ratón sobre el enlace provoca la ejecución de la función miFuncion().

A continuación, se muestra el ejemplo incluyendo el código HTML:

```
<html>
<head>
<script type="text/javascript">
   function miFuncion() {
      alert("Enlace a la página principal de la UOC");
   }
</script>
</head>
<body>
```

```
<a href="http://www.uoc.es" onmouseover="miFuncion()">Sitúe el puntero sobre este enlace.</a>
</body>
</html>
```

5.1.2. Vinculación mediante objetos en JavaScript

A partir de las versiones 3 de Netscape y 4 de Internet Explorer, se implementó la posibilidad de gestionar controladores de eventos de un objeto mediante JavaScript. La principal ventaja de este mecanismo es la separación entre la estructura y la lógica del documento de su presentación.

Cuando se implementa un controlador de evento para un objeto, el objeto en cuestión adquiere una propiedad que toma el nombre del controlador del evento. Esta nueva propiedad es la que permitirá acceder al controlador del evento.

```
objeto.controlador_de_evento = función_manejadora;
```

En los siguientes ejemplos, se muestran dos maneras de responder al evento *load* de la ventana del navegador. En el primer caso, se define el controlador del evento en la etiqueta <body>, y en el segundo, en el objeto Window:

```
<html>
<head>
<script type="text/javascript">
   function direccion() {
      alert("La URL de esta página es: " + document.URL );
   }
</script>
</head>
<body onload="direccion()">
</body>
</html>
```

En el siguiente ejemplo, hay que remarcar cómo el controlador se usa como una propiedad más del objeto Window:

```
<html>
<head>
<script type="text/javascript">
   function direccion() {
      alert("La URL de esta página es: " + document.URL);
   }
   window.onload = direccion; //nombre del controlador en minúsculas
</script>
</head>
<body>
```

```
</body>
</html>
```

5.1.3. Valores de retorno

Los manejadores de eventos disponen de una característica muy útil, los valores de retorno. Los manejadores de eventos no son más que funciones JavaScript y estas pueden retornar un valor mediante la sentencia *return*.

Los valores de retorno se pueden utilizar con varios objetivos. Veamos un par de ejemplos:

- Un formulario en el que se ha definido un manejador para el evento *submit*, de manera que este manejador valida los valores introducidos en el campo y devuelve *true* o *false*, dependiendo de si estos son correctos. En el caso de que se devuelva *true*, el envío del formulario se realiza, y si se devuelve *false*, se cancela.

- Otro ejemplo sería el evento *click* sobre un enlace, de modo que si se devuelve *false*, la carga de la página se cancela.

El siguiente código es un ejemplo de lo comentado en el punto anterior:

```
<a href="http://www.uoc.edu" onclick="return confirm('¿Desea ir al Campus?')">Campus UOC</a>
```

La siguiente tabla muestra los valores de retorno de los eventos más útiles.

Tabla 40. Valores de retorno de los eventos más útiles

Evento	Retorno	Descripción
Click	false	Botón de opción y casilla de verificación: no se establece. Botón de envío: el envío del formulario se cancela. Botón de restablecer: el formulario no se restablece. Enlace: el enlace no se carga.
Keydown	false	Cancela los eventos keypress que siguen mientras el usuario mantiene pulsada la tecla.
Keypress	false	Cancela el evento keypress.
Mousedown	false	Cancela la acción predeterminada.
Submit	false	Cancela el envío del formulario.

5.2. Modelo HTML dinámico de control de eventos

Los modelos de eventos surgidos a partir de las versiones 4.x de los navegadores proporcionaron un nuevo objeto estático Event. Este objeto era creado cuando ocurría un evento y el objeto Event recogía información sobre este, como la posición del ratón, el botón del ratón pulsado, el tipo de evento y otras informaciones.

Lo realmente interesante es que este objeto se puede consultar por el manejador del evento y, de esta manera, dispone de un conjunto de información que describe el evento ocurrido.

Como es de suponer, la divergencia surgida en los modelos de objetos a partir de las versiones 4.x también afectó al modelo de eventos, por lo que los modelos entre los dos navegadores eran incompatibles.

5.2.1. Introducción al modelo de eventos de Internet Explorer 4.x y Netscape 4.x

En Netscape 4.x los eventos se propagaban mediante la jerarquía del documento, y empieza en la parte superior para descender hasta llegar al objeto en el que se había generado el evento.

De esta manera, los objetos superiores como Window, Document y Layer podían capturar los eventos antes de que estos fueran procesados por el objeto que los había generado. El objetivo era simple: unificar la gestión de eventos en un nivel superior para evitar la repetición de manejadores idénticos en niveles inferiores.

En Internet Explorer 4.x, prácticamente todos los objetos de la página podían capturar eventos. Además, se disponía de un conjunto más amplio de eventos que eran aplicables a cada objeto.

A diferencia del modelo de Netscape, en el modelo de eventos de Internet Explorer 4.x, el evento empieza a propagarse desde el objeto en el que se ha creado y va ascendiendo por la estructura de la jerarquía hasta el objeto Window.

Tanto en Internet Explorer como en Netscape, cuando ocurría un evento se creaba un objeto Event, la diferencia entre estos es que Netscape lo enviaba al manejador de eventos como parámetro, mientras que en Internet Explorer el objeto era global y, por lo tanto, accesible directamente desde cualquier código a través de *event*.

A continuación, se presenta un ejemplo de su uso en Netscape:

```
<a href="http://www.uoc.edu">Campus UOC</a>
<script type="text/javascript">
```

```
    function posicion(e){
       alert("Has hecho click en: X="+ e.screenX +" Y="+ e.screenY);
    }
document.links[0].onclick=posicion;
</script>
```

En el ejemplo vemos cómo la función manejadora necesita un parámetro de entrada que representará el objeto Event, y cómo al final del todo se indica la captura del evento *onclick* de la etiqueta <a> asignándolo al manejador la función.

Veamos cómo se realizaba la misma acción en Internet Explorer:

```
<a href="http://www.uoc.edu">Campus UOC</a>
<script type="text/javascript">
   function posicion(){
      alert("Has hecho click en: X="+ event.screenX +" Y="+ event.screenY);
   }
   document.links[0].onclick= posicion;
</script>
```

Del código anterior podemos observar cómo la función *posicion()* no recibe como parámetro el objeto del evento, en su lugar hacer referencia a un objeto global *event*.

Las diferencias entre los navegadores en el manejo de eventos, la utilización de nombres diferentes para las mismas propiedades o la existencia de ciertas propiedades solo en uno de estos obligaban a los programadores a detectar el navegador que estaba utilizando el usuario para poder actuar en consecuencia.

A continuación, un ejemplo de cómo lograr que los ejemplos anteriores funcionasen en los dos navegadores:

```
<a href="http://www.uoc.edu">Campus UOC</a>
<script type="text/javascript">
   function posicion(e){
      e=(e) ? e : event;
      alert("Has hecho click en: X="+ e.screenX +" Y="+ e.screenY);
   }
   document.links[0].onclick=posicion;
</script>
```

En el ejemplo anterior, vemos cómo en la primera línea de la función posición se mira si realmente la función ha recibido como parámetro el objeto evento que envía Netscape; en caso afirmativo, asignaremos el valor a "e" del objeto recibido como parámetro, en caso negativo, asignaremos el objeto global *event* de Internet Explorer.

5.3. Modelo de eventos del DOM estándar

El modelo de eventos del DOM2 es un híbrido entre el modelo de Netscape 4.x y el de Internet Explorer 4.x. En este modelo, los eventos empiezan en la parte superior de la jerarquía y descienden por esta hasta llegar al objeto de destino. Este proceso se conoce como **fase de captura** y es similar a lo que ocurre con los navegadores Netscape 4.

Pero la propagación no ha finalizado en el objeto de destino, sino que una vez se encuentre en este y finalice el tratamiento por el manejador (si se ha asignado) comienza la fase que se conoce como fase de ascenso, en la que el evento implementa el camino inverso hasta llegar a la parte alta de la jerarquía. De esta manera, un evento puede ser capturado y manejado por un objeto de la jerarquía, tanto en la fase de captura como de ascenso.

En el DOM estándar, el objeto Event dispone de un conjunto amplio de propiedades que se detallan en la siguiente tabla:

Tabla 41. Propiedades del objeto Event en el DOM estándar

Propiedad	Descripción
altKey	Contiene un valor booleano que indica si se ha pulsado la tecla ALT.
ctrlKey	Contiene un valor booleano que indica si se ha pulsado la tecla CTRL.
shiftKey	Contiene un valor booleano que indica si se ha pulsado la tecla SHIFT.
metaKey	Contiene un valor booleano que indica si se ha pulsado la tecla META.
bubbles	Contiene un valor booleano que indica si el evento asciende por la jerarquía.
button	Indica el botón del ratón que se ha presionado.
cancelable	Contiene un valor lógico que indica si el evento no debería ascender por la jerarquía.
clientX	Contiene la coordenada X donde ha ocurrido el evento.
clientY	Contiene la coordenada Y donde ha ocurrido el evento.
screenX	Contiene la coordenada X donde ha ocurrido el evento respecto a la pantalla completa.
screenY	Contiene la coordenada Y donde ha ocurrido el evento respecto a la pantalla completa.
target	Contiene una referencia al nodo en el que se ha producido el evento.
currentTarget	Contiene una referencia al nodo al que se le ha asignado el manejador.

Propiedad	Descripción
relatedTarget	Contiene una referencia al nodo del que ha salido el evento.
type	Contiene una cadena que contiene el tipo de evento.
eventPhase	Indica la fase del recorrido en la que se ha capturado el evento: 1 captura, 2 en el destino y 3 en el ascenso.

Por su parte, los navegadores no cumplen el estándar al 100%; en realidad, tienen ciertas diferencias y añaden propiedades o extensiones al estándar.

5.3.1. Eventos del estándar DOM

La siguiente tabla muestra un subconjunto de eventos definidos en el DOM2 que son básicamente los especificados por HTML 4 y DOM0, más un conjunto nuevo de eventos relacionados con la interfaz de usuario y la manipulación de la estructura del documento.

Tabla 42. Subconjunto de eventos definidos en el DOM2

Tipo evento	Evento	¿Asciende?	¿Cancelable?
Ratón	click	Sí	Sí
	mousedown	Sí	Sí
	mouseup	Sí	Sí
	mouseover	Sí	Sí
	mousemove	Sí	No
	mouseout	Sí	Sí
Navegador y HTML	load	No	No
	unload	No	No
	abort	Sí	No
	error	Sí	No
	select	Sí	No
	change	Sí	No
	submit	Sí	Sí
	reset	Sí	No
	focus	No	No
	blur	No	No
	resize	Sí	No
	scroll	Sí	No
De interfaz de usuario	DOMFocusIn		

Tipo evento	Evento	¿Asciende?	¿Cancelable?
	DOMFocusOut		
	DOMActivate		
De cambio jerarquía	DOMSubtreeModified	Sí	No
	DOMNodeInserted	Sí	No
	DOMNodeRemoved	Sí	No
	DOMNodeRemoved-FromDocument	No	No
	DOMNodeInsertedIn-toDocument	No	No
	DOMAttrModified	Sí	No
	DOMCharacterData-Modified	Sí	No

Como se observa en la tabla, aparece un nuevo tipo de eventos: **eventos de cambio en la jerarquía**. Este tipo de eventos, como su nombre indica, se produce cuando la estructura de la página es modificada, ya que esto es posible a partir de los métodos del DOM estándar.

La vinculación de eventos especificada en el DOM2 es también un híbrido de la vinculación especificada en los modelos propietarios de Internet Explorer y Netscape. El DOM especifica que la función manejadora recibirá un parámetro que apuntará al objeto Event (tal como ocurría en Netscape).

En el siguiente ejemplo, se observa cómo se vincula el evento *click* a una etiqueta <a>, utilizando la sintaxis del DOM en la localización del objeto:

```
<a href="http://www.uoc.edu" id="uoc">Campus UOC</a>
<script type="text/javascript">
   function maneja(e){
      alert("Has hecho click en: X="+ e.screenX +" Y="+ e.screenY);
   }
   document.getElementById("uoc").onclick=maneja;
</script>
```

Del ejemplo anterior, es necesario incidir en:

- La función manejadora recibe en su parámetro al objeto Event, que podrá consultarse en el interior de la función.

- La asignación del evento al objeto enlace se realiza utilizando las propiedades estándar del DOM.

Las versiones actuales de Internet Explorer cumplen parcialmente el estándar DOM2, por lo que pueden aparecer problemas con la gestión de eventos utilizando el estándar si se ejecutan en estos navegadores.

Por otra parte, el DOM proporciona un mecanismo muy interesante que permite la vinculación de eventos a objetos de la jerarquía. Este mecanismo aporta las siguientes ventajas:

- Permite especificar si el evento se manejará durante la fase de captura o durante la fase de ascenso, mientras que en el método presentado anteriormente el evento se captura cuando el objeto es el destino y durante la fase de ascenso, en el caso de que la haya para el evento.

- Permite vincular varios manejadores para un mismo evento y un mismo objeto.

- Permite vincular un manejador a un nodo del tipo texto.

Los métodos añaden y eliminan lo que se conoce como **oyentes de eventos**, que no son más que manejadores de eventos vinculados a un nodo determinado de la jerarquía de objetos, que es activado durante una fase específica del ciclo de vida del evento, ya sea la fase de captura o la de ascenso.

El método que añade un oyente tiene la siguiente sintaxis:

```
nodo.addEventListener(tipo,manejador,sentido);
```

donde los parámetros tienen el siguiente objetivo:

- tipo: es una cadena que indica el evento que se va a escuchar;
- manejador: es la función que va a manejar el evento;
- sentido: es un valor booleano que indica si se captura en la fase de captura o en la fase de ascenso.

Al igual que se dispone del anterior método para la creación de un oyente, la misma sintaxis es utilizada por el método removeEventListener(tipo,manejador,sentido) para eliminar un oyente añadido anteriormente.

Además, el objeto Event dispone de dos métodos muy interesantes:

- preventDefault(): que cancela el comportamiento estándar de un evento;
- stopPropagation(): que detiene el flujo del evento a través de la jerarquía.

En el siguiente ejemplo, se puede observar código que utiliza las funciones anteriores:

```
<a href="http://www.uoc.edu" id="uoc">Campus UOC</a>
<script type="text/javascript">
   function NoAltClick(e){
      if (e.altKey){
         e.preventDefault();
         e.stopPropagation();
      }
   }
   function MuestraClick(){
      alert("Se ha realizado click");
   }
   document.addEventListener("click",NoAltClick,true);
   document.getElementById("uoc").addEventListener("click",MuestraClick,true);
</script>
```

Del ejemplo, es necesario tener en cuenta el detalle siguiente: el manejador NoAltClick provoca que los eventos *click* no funcionen mientras se pulsa la tecla Alt. Esto se consigue al cancelar el comportamiento predeterminado del evento y al detener su propagación mediante la jerarquía.

Para finalizar, un último método disponible en todos los nodos del DOM: dispatchEvent(), que recibe como parámetro un objeto Event y provoca que el nodo que ha llamado al método se convierta en el nuevo destinatario del evento.

Actividades

1. Cread un *script* que, a partir de una página web formada por 2 párrafos, en cuyo texto se encuentran tres enlaces web, realice lo siguiente:

a) Calcule el número de enlaces que tiene la página web.
b) Retorne la dirección a la que dirige el último enlace.
c) Calcule el número de enlaces del primer párrafo.

2. Cread un *script* en el que, a partir de una página web con un párrafo que explica la opinión sobre una receta de cocina y un enlace, al pulsar en este último, se abra una caja de texto solicitando la opinión sobre esta al usuario y el anexo al texto anterior (similar a las opiniones que se adjuntan en ciertos periódicos web sobre las noticias publicadas).

3. Cread una página inici.html, formada por campos que contengan el nombre, apellidos, e-mail y DNI del usuario. La validación de los campos de esta habrá que programarla en un fichero valida.js.

- Los controles a validar del formulario son que el DNI es obligatorio y numérico, y que la selección de la letra ha de ser correcta. En caso de error, se deben visualizar según sea el caso los mensajes siguientes: "Completa el campo DNI", "Teclea un DNI (sin letras, solo números)" y "La letra del NIF es incorrecta."
- El algoritmo para calcular la letra del NIF es el resultado de calcular la resta de dividir el número del DNI por 23 y la letra del NIF corresponde al carácter obtenido de la cadena "TRWAGMYFP-DXBNJZSQVHLCKE" en función del valor del resto.
- Sobre la dirección de correo, habrá que controlar que exista y que los datos introducidos correspondan a una dirección de correo electrónico correcto.

4. Utilizad el *script* explicado en la página web:

http://www.desarrolloweb.com/articulos/1422.php,

para crear una página que contenga una galería de fotos que pueda mostrar de forma secuencial un conjunto de imágenes.

5. Cread una página web que muestre en un campo de texto las coordenadas del puntero de ratón de forma dinámica. Para este objetivo habrá que utilizar los eventos correspondientes.