

BOM I

¿Qué es?

Las versiones 3.0 de los navegadores Internet Explorer y Netscape Navigator introdujeron el concepto de **Browser Object Model o BOM**, que permite acceder y modificar las **propiedades de las ventanas del propio navegador**.

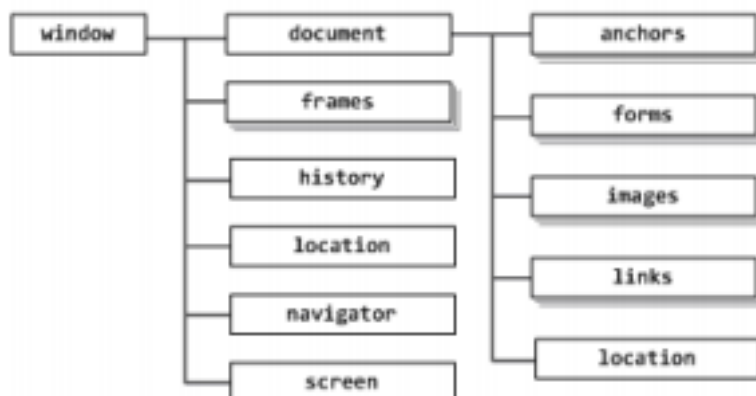
Mediante BOM, es posible **redimensionar** y **mover** la **ventana** del **navegador**, modificar el **texto** que se muestra en la **barra** de **estado** y realizar muchas otras manipulaciones no relacionadas con el contenido de la página HTML.

El mayor inconveniente de BOM es que, al contrario de lo que sucede con DOM, **ninguna entidad se encarga de estandarizarlo** o definir unos mínimos de interoperabilidad entre navegadores.

Algunos de los elementos que forman el BOM son los siguientes:

- Crear, mover, redimensionar y cerrar **ventanas** de **navegador**.
- Obtener **información** sobre el propio **navegador**.
- Propiedades de la **página actual** y de la **pantalla** del **usuario**.
- Gestión de **cookies**.
- Objetos **ActiveX** en Internet Explorer.

El **BOM** está compuesto por varios **objetos** relacionados entre sí. El siguiente esquema muestra los objetos de BOM y su relación:



En el esquema anterior, los objetos mostrados con varios **recuadros superpuestos** son **arrays**. El resto de objetos, representados por un rectángulo individual, son objetos simples. En cualquier caso, todos los objetos derivan del objeto window.

El objeto window

El **objeto window** representa la **ventana completa del navegador**. Mediante este objeto, es posible **mover**, **redimensionar** y **manipular** la **ventana** actual del **navegador**. Incluso es posible **abrir** y **cerrar nuevas ventanas** de navegador.

Si una página emplea **iframes**, cada uno de ellos se almacena en el array frames, que puede ser accedido numéricamente (window.frames[0]) o, si se ha indicado un nombre al iframe, mediante su nombre (window.frames["nombre del frame"]).

Como todos los demás objetos heredan directa o indirectamente del **objeto window**, **no es necesario indicarlo** de forma explícita en el código JavaScript. En otras palabras:

- `window.frames[0] == frames[0]`
- `window.document == document`

BOM define cuatro **métodos** para **manipular** el **tamaño** y la **posición** de la **ventana**:

- **moveBy(x,y)** **desplaza** la posición de la **ventana** x píxeles hacia la **derecha** y y píxeles hacia **abajo**. Se **permiten desplazamientos negativos** para mover la ventana hacia la izquierda o hacia arriba.
- **moveTo(x,y)** **desplaza** la **ventana** del navegador hasta que la **esquina superior izquierda** se encuentre en la **posición (x,y)** de la **pantalla** del usuario. Se permiten **desplazamientos negativos**, aunque ello suponga que parte de la ventana no se visualiza en la pantalla.
- **resizeBy(x,y)** **redimensiona** la **ventana** del navegador de forma que su **nueva anchura** sea igual a (**anchura_anterior + x**) y su **nueva altura** sea igual a (**altura_anterior + y**). Se pueden **emplear valores negativos** para reducir la anchura y/o altura de la ventana.
- **resizeTo(x,y)** **redimensiona** la **ventana** del navegador hasta que su anchura sea igual a x y su altura sea igual a y. **No se permiten valores negativos**.

Los **navegadores** son cada vez **menos permisivos** con la modificación mediante **JavaScript** de las propiedades de sus ventanas. De hecho, la mayoría de navegadores permite a los usuarios bloquear el uso de JavaScript para realizar cambios de este tipo. De esta forma, una aplicación nunca debe suponer que este tipo de funciones están disponibles y funcionan de forma correcta.

A continuación se muestran algunos **ejemplos** de uso de estas funciones: //

Mover la ventana 20 píxel hacia la derecha y 30 píxel hacia abajo

```
window.moveBy(20, 30);
```

```
// Redimensionar la ventana hasta un tamaño de 250 x 250
window.resizeTo(250, 250);




// Agrandar la altura de la ventana en 50 píxel
window.resizeBy(0, 50);

// Colocar la ventana en la esquina izquierda superior de la ventana
window.moveTo(0, 0);
```

Además de desplazar y redimensionar la ventana del navegador, es posible averiguar la posición y tamaño actual de la ventana. Sin embargo, la ausencia de un estándar para BOM provoca que **cada navegador implemente su propio método**:

- Para averiguar la **posición de la ventana**

- `window.screenLeft`
- `window.screenTop`

Property					
screenLeft	Yes	Yes	Not supported	Yes	Yes
screenTop	Yes	Yes	Not supported	Yes	Yes

Para Firefox utilizamos:

- `window.screenX`
- `window.screenY`

No es posible obtener el tamaño de la ventana completa, sino solamente del **área visible** de la **página** (es decir, **sin barra de estado ni menús**).

Las propiedades que proporcionan estas **dimensiones completas del navegador**

- son:
- `document.body.offsetWidth`
 - `document.body.offsetHeight`.

El **tamaño de la zona visible de la ventana** se obtiene mediante:

- `window.innerWidth`
- `window.innerHeight`

mientras que el tamaño total de la ventana se obtiene mediante:

- `window.outerWidth`
- `window.outerHeight`.

window.open()

window.open (**URL**, **nombre**, **Especificaciones**, **reemplazar**)

Abre una nueva ventana del navegador. Los valores de los parámetros y las descripciones son los siguientes:

- **URL**: (opcional) Especifica la dirección URL de la página para abrir. Si no se especifica un URL, una nueva ventana con aproximadamente: se abre en blanco
- **nombre**: (opcional) Especifica el atributo de destino o el nombre de la ventana. Los valores siguientes son compatibles:
 - **_blank** - URL se carga en una nueva ventana. Esta es por defecto
 - **_parent** - URL se carga en el marco padre
 - **_self** - URL reemplaza la página actual
 - **_top** - URL reemplaza cualquier conjuntos de marcos que pueden ser cargados
 - **nombre** - El nombre de la ventana (Nota: el nombre no especifica el título de la nueva ventana)
- **Especificaciones**: (opcionales) Una lista separada por comas de elementos. Los valores siguientes son compatibles:
 - **channelmode** = **yes | no | 1 | 0** Si se muestra o no la ventana en modo teatro. Por defecto es no. (sólo por IE)
 - **directories** = **yes | no | 1 | 0** obsoleto. Si se añaden o no botones de directorio. Por defecto es yes. (sólo por IE)
 - **fullscreen** = **yes | no | 1 | 0** Si se muestra o no el navegador en modo de pantalla completa. Por defecto es no. Una ventana en modo de pantalla completa también tiene que estar en la manera teatro. (Sólo por IE)
 - **height** = **píxeles** La altura de la ventana. Valor mínimo es 100
 - **left** = **píxeles** La posición izquierda de la ventana. Los valores negativos no son permitidos
 - **location** = **yes | no | 1 | 0** Si queréis que se muestro o no el campo de direcciones. (sólo Opera)
 - **menubar** = **yes | no | 1 | 0** Si queréis que se muestro o no la barra de menú
 - **resizable** = **yes | no | 1 | 0** Si queréis que la ventana sea o no de medida variable. (Sólo por IE)
 - **scrollbars** = **yes | no | 1 | 0** Para mostrar o no las barras de desplazamiento. (Sólo IE, Firefox y Opera)
 - **status** = **yes | no | 1 | 0** Añadir o no una barra de estado
 - **titlebar** = **yes | no | 1 | 0** Mostrar o no la barra de título. Ignorado a menos que la aplicación que la llamo sea una aplicación HTML o un cuadro de diálogo de

confianza

- **toolbar** = **yes | no | 1 | 0** Mostrar o nola barra de herramientas del navegador. (Sólo IE y Firefox)
- **top** = **píxeles** La posición superior de la ventana. Los valores negativos no están permitidos
- **width** = **píxeles** La anchura de la ventana. Valor mínimo es 100
- **reemplazar**: (opcional) Especifica si la dirección URL crea una nueva entrada o sustituye a la entrada actual de la lista del historial. Los valores siguientes son compatibles:
 - **true** - URL reemplaza el documento actual a la lista del historial
 - **false** - URL crea una nueva entrada a la lista del historial

setTimeout(), setInterval()

Al contrario que otros lenguajes de programación, **JavaScript no incorpora** un método **wait()** que detenga la ejecución del programa durante un tiempo determinado. Sin embargo, JavaScript **proporciona** los métodos **setTimeout()** y **setInterval()** que se pueden emplear para realizar tareas similares.

El método **setTimeout()** permite **ejecutar** una **función** al **transcurrir** un **determinado periodo** de **tiempo**:

```
setTimeout("alert('Han pasado 3 segundos')", 3000);
```

El método **setTimeout()** requiere dos argumentos:

- **Código** que se va **a ejecutar** o una referencia a la **función** que se debe ejecutar.
- **Tiempo**, en milisegundos, que se espera **hasta** que **comienza** la **ejecución** del código. El ejemplo anterior se puede rehacer utilizando una función:

```
function muestraMensaje() {  
    alert("Han pasado 3 segundos");  
}  
setTimeout(muestraMensaje, 3000);
```

Como es habitual, cuando se indica la **referencia** a la **función no se incluyen los paréntesis**, ya que de otro modo, se ejecuta la función en el mismo instante en que se establece el intervalo de ejecución.

Cuando se establece una cuenta atrás, la función **setTimeout()** **devuelve** el **identificador** de esa nueva **cuenta atrás**. Empleando ese identificador y la función

`clearTimeout()` es posible **impedir** que se **ejecute** el **código pendiente**:

```
function muestraMensaje() {  
    alert("Han pasado 3 segundos");  
}  
var id = setTimeout(muestraMensaje, 3000);  
    // Antes de que transcurran 3 segundos, se decide eliminar la ejecución  
    pendiente  
clearTimeout(id);
```

Las funciones de **timeout** son imprescindibles para crear aplicaciones profesionales, ya que permiten, por ejemplo, que un script no espere infinito tiempo para obtener el resultado de una función.

En este caso, la estrategia consiste en establecer una **cuenta atrás antes** de **llamar** a la **función**. Si la función se ejecuta **correctamente**, en cuanto finalice su ejecución se **elimina** la **cuenta atrás** y continúa la ejecución normal del script. Si por cualquier motivo la función **no se ejecuta correctamente**, la **cuenta atrás se cumple** y la aplicación puede informar al **usuario** o **reintentar** la **ejecución** de la función.

Además de programar la ejecución futura de una función, JavaScript también permite establecer la **ejecución periódica** y repetitiva de una función. El método necesario es `setInterval()` y su funcionamiento es idéntico al mostrado para `setTimeout()`:

```
function muestraMensaje() {  
    alert("Este mensaje se muestra cada segundo");  
}  
setInterval(muestraMensaje, 1000);
```

De forma análoga a `clearTimeout()`, también existe un método que permite eliminar una repetición periódica y que en este caso se denomina `clearInterval()`:

```
function muestraMensaje() {  
    alert("Este mensaje se muestra cada segundo");  
}  
var id = setInterval(muestraMensaje, 1000);  
  
    // Despues de ejecutarse un determinado número de veces, se elimina el  
    intervalo  
clearInterval(id);
```