LENGUAJES DE MARCAS Y SISTEMAS DE GESTIÓN DE LA INFORMACIÓN

5. Browser Object Model (BOM)

Profesor: Alejandro Amat Reina

ÍNDICE

1. BOM (Browser Object Model)

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.

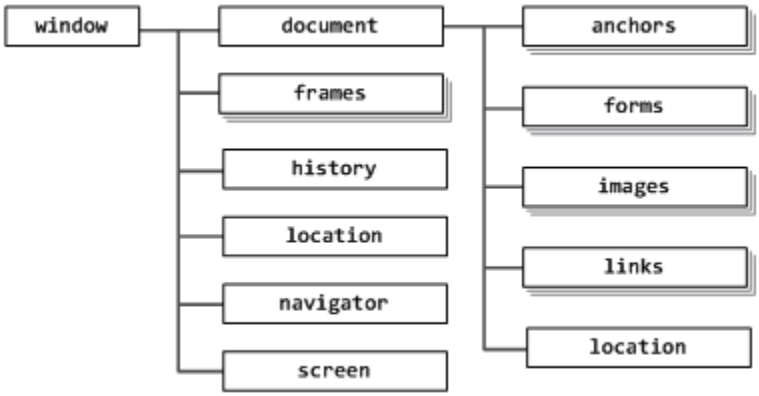
Mediante BOM, es posible redimensionar y mover la ventana del navegador, modificar el texto que se muestra en la barra de estado y realizar muchas otras manipulaciones no relacionadas con el contenido de la página HTML.

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.
- etc.

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



Jerarquía de objetos que forman el BOM

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.

1.1. El objeto window

El objeto window representa el navegador y es el objeto principal. Todo está contenido dentro de window (variables globales, el objeto document, el objeto location, el objeto navigation, etc.). El objeto window puede ser omitido accediendo a las propiedades directamente.

Ejemplos:

```
'use strict';
// Tamaño total de la ventana (excluye la barra superior del navegador)
console.log(window.outerWidth + " - " + window.outerHeight);
window.open("https://www.google.com");
// Propiedades de la pantalla
// Ancho de pantalla y alto (Resolución)
console.log(window.screen.width + " - " + window.screen.height);
// Excluyendo la barra del S.O.
console.log(window.screen.availWidth + " - " + window.screen.availHeight);
// Propiedades del navegador
// Imprime la información del navegador
console.log(window.navigator.userAgent);
window.navigator.geolocation.getCurrentPosition(function(position) {
    console.log(
        "Latitude: " + position.coords.latitude +
        ", Longitude: " + position.coords.longitude);
});
// Variables globales
let glob = "Hello";
// Imprime "Hello". Las variables globales son almacenadas en el objeto window
console.log(window.glob);
// En las variables globales podemos omitir el objeto window (está implícito)
// Número de páginas del history. Lo mismo que window.history.length
console.log(history.length);
```

1.2. Gestión de Timers

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, 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. Recibe dos argumentos:

- El primero es el código que se va a ejecutar o una referencia a la función que se debe ejecutar.
- El segundo argumento es el tiempo, en milisegundos, que se espera hasta que comienza la ejecución del código.

```
// Imprime inmediatamente la fecha actual
console.log(new Date().toString());
// Se ejecutará en 5 segundos (5000 milisegundos)
setTimeout(function() {
    console.log(new Date().toString());
}, 5000);
```

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:

```
// setTimeout devuelve un número con el id, y a partir de ahí, podremos cancelarlo
let idTime = setTimeout(function() {
    console.log(new Date().toString());
}, 5000);
// Cancela el timeout (antes de que se ejecute)
clearTimeout(idTime);
```

El uso de las funciones de timeout es muy habitual en las aplicaciones web.

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()`:

```
let num = 1;
// Imprime un número y lo incrementa cada segundo
setInterval(function() {
    console.log(num++);
}, 1000);
```

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

```
let num = 1;
let idInterval = setInterval(function() {
    console.log(num++);
    if(num > 10) {
// Cuando imprimimos 10, paramos el timer para que no se repita más
        clearInterval(idInterval);
    }
}, 1000);
```

Tanto a `setTimeout()` como a `setInterval()` podemos pasarle un nombre de función existente. En el caso de que dicha función reciba parámetros, podemos establecerlos tras los milisegundos.

```
function multiply(num1, num2) {
    console.log(num1 * num2);
}
// Después de 3 segundos imprimirá 35 (5*7)
setTimeout(multiply, 3000, 5, 7);
```

1.3. El Objeto document

El objeto document es el único que pertenece tanto al DOM como al BOM. Desde el punto de vista del BOM, el objeto document proporciona información sobre la propia página HTML. Por ejemplo, podemos cambiar el título de la página o la url actual del navegador a través de las propiedades title y url de este objeto:

```
// modificar el título de la página
document.title = 'Nuevo título';
// llevar al usuario a otra página diferente
document.URL = 'http://nueva_pagina';
```

Además de propiedades, el objeto document contiene varios arrays con información sobre algunos elementos de la página.

Array	Descripción
`anchors`	Contiene todas las "anclas" de la página (los enlaces de tipo)
`applets`	Contiene todos los applets de la página
`embeds`	Contiene todos los objetos embebidos en la página mediante la etiqueta <embed>
`forms`	Contiene todos los formularios de la página

`images`	Contiene todas las imágenes de la página
`links`	Contiene todos los enlaces de la página (los elementos de tipo)

Los elementos de cada array del objeto document se pueden acceder mediante su índice numérico o mediante el nombre del elemento en la página HTML, aunque en la actualidad se utilizan más los métodos del DOM vistos en el tema anterior.

Una vez obtenida la referencia al elemento, se puede acceder al valor de sus atributos HTML utilizando las propiedades de DOM. De esta forma, el método del formulario se obtiene mediante document.forms["consultas"].method y la ruta de la imagen es document.images[0].src.

1.4. Otros Objetos del BOM

Además del objeto window y el objeto document, existen otros objetos que también forman parte del BOM, pero que no tienen tanta importancia:

- El objeto location: es una propiedad tanto del objeto window como del objeto document. Representa la URL de la página HTML que se muestra en la ventana del navegador y proporciona varias propiedades útiles para el manejo de la URL: host, href, port, pathname, etc. De todas las propiedades, la más utilizada es location.href, que permite obtener o establecer la dirección de la página que se muestra en la ventana del navegador. Además de las propiedades de la tabla anterior, el objeto location contiene numerosos métodos y funciones, como por ejemplo el método reload(modo). Este método recarga la página actual, si el argumento es true, se carga la página desde el servidor, si es false, se carga desde la cache del navegador.

```
// Imprime la URL actual
console.log(location.href);
// Imprime el nombre del servidor (o la IP) como "localhost" 192.168.0.34
console.log(location.host);
// Imprime el número del puerto (normalmente 80)
console.log(location.port);
// Imprime el protocolo usado (http ó https)
console.log(location.protocol);
// Recarga la página actual
location.reload();
// Carga una nueva página. El parámetro es la nueva URL
location.assign("https://google.com");
// Carga una nueva página sin guardar la actual en el objeto history
location.replace("https://google.com");
```

- El objeto navigator: es uno de los primeros objetos que incluyó el BOM y permite obtener información sobre el propio navegador. Se emplea habitualmente para detectar el tipo y/o versión del navegador en las aplicaciones cuyo código difiere para cada navegador. Además, se emplea para detectar si el navegador tiene habilitadas las cookies y Java y también para comprobar los plugins disponibles en el navegador.

```
// Propiedades del navegador
// Imprime la información del navegador
console.log(window.navigator.userAgent);
window.navigator.geolocation.getCurrentPosition(function(position) {
    console.log("Latitude: " + position.coords.latitude +
        ", Longitude: " + position.coords.longitude);
});
```

- El objeto screen: El objeto screen se utiliza para obtener información sobre la pantalla del usuario. Las propiedades más importantes de las que dispone son: availHeight (altura de pantalla disponible para las ventanas), availWidth (anchura de pantalla disponible para las ventanas), colorDepth (profundidad de color de la pantalla), height (altura total de la pantalla en píxel) y width (anchura total de la pantalla en píxel).

  La altura/anchura de pantalla disponible para las ventanas es menor que la altura/anchura total de la pantalla, ya que se tiene en cuenta el tamaño de los elementos del sistema operativo como por ejemplo la barra de tareas y los bordes de las ventanas del navegador.

```
// Propiedades de la pantalla
// Ancho de pantalla y alto (Resolución)
console.log(window.screen.width + " - " + window.screen.height);
// Excluyendo la barra del S.O.
console.log(window.screen.availWidth + " - " + window.screen.availHeight);
```

2. Ejercicios

2.1. Color de fondo del carrito

Cuando el carrito esté vacío, el color de fondo del mismo debe cambiar, cada segundo, de rojo a amarillo y viceversa (utiliza la función setInterval vista en el tema). Cuando se añada un artículo al carrito, el color de fondo que esté activo en ese momento se quedará fijo, y si el carrito se vuelve a quedar vacío, volverá a comenzar el parpadeo de color.

Debes mantener una variable global en la que almacenes el id del timer para poder activarlo y desactivarlo cuando corresponda.

2.2. Botones de desplazamiento

Veamos ahora como debes implementar la funcionalidad de moverte por los artículos del carrito con los botones de desplazamiento:

- Lo primero que debes tener en cuenta es que un artículo en el carrito tiene un ancho de unos 120 píxeles y que en el ancho inicial del carrito caben 4 artículos. Por lo tanto, cuando se añade un producto al carrito, debes comprobar si hay más de cuatro artículos añadidos, en cuyo caso, aumentarás el ancho del contenedor de artículos del carrito (id cart_items) en 120 píxeles.

  Para saber cuántos artículos hay en el carrito puedes obtener los hijos del mismo con la propiedad children y acceder a su propiedad length.

- Cuando se pulsa el botón anterior (id btn_prev), debes desplazar el contenedor de artículos del carrito (id cart_items) 50 píxeles a la derecha (suma 50 a la propiedad left).

- Cuando pulsas el botón siguiente (id btn_next), debes desplazar el contenedor de artículos del carrito (id cart_items) 50 píxeles a la izquierda (resta 50 a la propiedad left).

- Si el contenedor de artículos llega al borde del carrito, por la izquierda o por la derecha, no debes seguir desplazándolo aunque se siga pulsando el botón de desplazamiento. Para controlar esto guarda el rectángulo incial del carrito (`carrito.getBoundingClientRect()`[1]). Después, controla lo siguiente:

  - La posición izquierda del carrito actual después de desplazarlo no debe ser nunca mayor que la posición izquierda inicial del mismo. En caso de darse esa situación pon el carrito en su posición inicial. Para ello, haz lo siguiente:

```
carrito.style.left = '0px';
```

[1] Puedes ver el funcionamiento de la función getBoundingClientRect de un NodeElement en el siguiente enlace: *https://developer.mozilla.org/es/docs/Web/API/Element/getBoundingClientRect*

- La posición derecha actual del carrito no debe ser nunca menor que la posición derecha inicial del mismo. En caso de darse esa situación pon el borde derecho del carrito en el mismo lugar que estaba inicialmente. Para ello, haz lo siguiente:

```
carrito.style.left = -(rectCarrito.width - rectCarritoInicial.width) + 'px';
```

- Por último, al eliminar un artículo del carrito, tienes que comprobar si el número de artículos restante es mayor de 4, y, si es así, disminuir el ancho del carrito 120 píxeles. Al hacer esto también debes comprobar si la posición derecha actual del carrito es menor que la posición derecha inicial del mismo, al igual que hacías en el punto anterior.
- Debes crear constantes para los datos fijos de la aplicación, ancho artículo, máximo de artículos visibles en el carro y desplazamiento.

Importante: Cuando obtenemos desde javascript una propiedad de estilo que se ha modificado desde css, no obtendremos el valor computado de la misma. Para poder acceder al valor computado, debes utilizar el método getComputedStyle (puedes consultar la ayuda de esta función en el siguiente enlace:

https://developer.mozilla.org/es/docs/Web/API/Window/getComputedStyle.

Por ejemplo, para obtener la propiedad left computada del carrito haremos lo siguiente:

```
let carrito = document.getElementById('cart_items');
let style = window.getComputedStyle(carrito, '');
let left = parseInt(style.left);
```