Acceso a los elementos de un documento

1. Introducción. BOM y DOM

JavaScript es un lenguaje que se ejecuta normalmente en un navegador, sobre una página web. Esta característica hace que JavaScript pueda acceder directamente a ciertas propiedades del navegador y de la propia página sobre la que se está ejecutando, lo que se conoce como BOM (*Browser Object Model*) y DOM (*Document Object Model*), respectivamente.

En esta sección veremos algunas de las propiedades más relevantes que pueden ser accedidas y utilizadas desde JavaScript en uno y otro modelo. Por ejemplo, cambiar la URL actual del navegador, o acceder a un elemento concreto de un documento para comprobar su valor o para cambiarlo.

2. Elementos principales del BOM

Dentro del conjunto de elementos definidos por el navegador o BOM, el elemento principal que lo engloba todo es el elemento `window` (ventana). A partir de este elemento podemos hacer tareas como:

- Obtener las dimensiones de la pantalla donde estamos
- Mostrar mensajes (instrucciones *alert* o *prompt* que ya hemos utilizado antes)
- Cambiar la URL de la página actual, para ir a otra página distinta
- ... etc.

Desde el propio elemento *window* podemos acceder a otros subelementos para completar estas tareas. Por ejemplo:

- La pantalla (elemento `screen`), a partir de la cual podemos acceder a ciertos datos como la anchura y altura, profundidad de color, etc.
- El navegador en sí (elemento `navigator`), donde podemos acceder al nombre del navegador, plugins instalados...
- El documento que tenemos abierto (elemento `document`), nos va a permitir acceder a la URL del documento, su título, enlaces...
- La ubicación del documento actual (elemento `location`) nos permitirá centrarnos en la URL del documento actual, pudiéndola cambiar para cargar otras páginas.
- El historial de navegación (elemento `history`) con el que podremos ir hacia adelante o atrás en el mismo.

Veamos algunos ejemplos de uso básico de algunas de estas propiedades:

```
// Sacamos por consola el ancho y alto de la pantalla:
console.log(window.screen.width + " x " + window.screen.height);

// Cargamos otra página
window.location.href = "https://iessanvicente.com";
// También sirve
location.href = "https://iessanvicente.com";

// Vamos a la página anterior del historial de navegación
window.history.back();
// También sirve
history.back();
```

2.1. Uso de *timeouts* e intervalos

Una de las aplicaciones más útiles que podemos extraer del uso del BOM es la posibilidad de definir **timeouts**, un tiempo pasado el cual queremos que se ejecute una instrucción o conjunto de instrucciones. Para ello usamos la función `timeout` indicándole qué función queremos ejecutar, y cuántos milisegundos queremos que pasen antes de ejecutarla. Por ejemplo, el siguiente código saca por consola la fecha actual pasados 3 segundos (usamos una *arrow function* para definir la función a ejecutar):

```
setTimeout(() => console.log(new Date()), 3000);
```

Una opción un poco más elaborada consist en definir **intervalos**. La diferencia con lo anterior es que la función no sólo se ejecuta una vez, sino que se ejecuta periódicamente tras cada intervalo de tiempo definido. Este código muestra la fecha por consola cada 5 segundos:

```
setInterval(() => console.log(new Date()), 5000);
```

Podemos también cancelar los intervalos que tengamos habilitados en cualquier momento. Para ello, debemos obtener el *id* del intervalo generado cuando lo creamos, y luego llamar a la función `clearInterval` con ese *id*. Por ejemplo, este código activa el intervalo anterior y, pasados 20 segundos, lo para.

```
let idIntervalo = setInterval(() => console.log(new Date()), 5000);
setTimeout(() => clearInterval(idIntervalo), 20000);
```

Tanto a *setTimeout* como a *setInterval* podemos pasarles una función ya existente, no es necesario definirla sobre la marcha con una *arrow function*. Si la función necesita parámetros adicionales, podemos pasárselos

después del tiempo de espera. El siguiente *timeout* saluda al nombre que se le pasa como parámetro, pasados 5 segundos:

```
function saludar(nombre)
{
    console.log("Hola, " + nombre);
}

setTimeout(saludar, 5000, "Nacho");
```

3. Uso del DOM

A través del modelo del documento o DOM podemos acceder a la estructura de información contenida en dicho documento: etiquetas, subetiquetas, atributos, etc. Esto nos puede servir para:

- Consultar información de ciertos elementos de la página. Por ejemplo, validar si los datos de un formulario son correctos, o cuántos elementos de un tipo hay en una página
- Modificar la información de ciertos elementos. Por ejemplo, cambiar sus propiedades CSS (color de fondo, visibilidad), o añadir/quitar elementos dinámicamente en la misma página.

Para acceder o manipular la estructura del documento utilizaremos el objeto `document`. Es un objeto global que podemos utilizar directamente, o bien desde el objeto `window` que hemos visto antes (*window.document*, aunque es mucho más habitual utilizar *document* a secas).

3.1. Instrucciones para explorar el DOM

Las siguientes instrucciones nos pueden resultar muy útiles para explorar la estructura del DOM:

- `document.documentElement` obtiene el elemento raíz (*html*)
- `document.head` obtiene la cabecera (elemento *head*)
- `document.body` obtiene el cuerpo (elemento *body*)
- `document.getElementById("id")` obtiene el elemento cuyo *id* hemos indicado como parámetro
- `document.getElementsByClassName("clase")` obtiene el conjunto de elementos de la clase indicada.
- `document.getElementsByTagName("etiqueta")` obtiene el conjunto de etiquetas del tipo indicado.
- `element.children` obtiene todas las etiquetas que cuelgan del elemento actual
- `element.parentNode` obtiene la etiqueta padre de la actual
- `element.nextSibling` obtiene el siguiente hermano del elemento actual. También tenemos disponible `element.previousSibling`. Alternativamente, tenemos `nextElementSibling` o `previousElementSibling` si sólo queremos recorrer etiquetas, y no nodos de texto.
- `element.innerHTML` obtiene el contenido HTML del elemento actual. Alternativamente, podemos emplear la propiedad `textContent` o `innerText` para mostrar el contenido textual (sin subetiquetas, ya que las renderiza) que tenga el elemento.

Podemos llamar a estas instrucciones (especialmente a las que buscan por *id*, clase o nombre de etiqueta) desde el documento en general (*document*) o desde un elemento en particular, con lo que buscará sólo en el interior de ese documento.

Veamos cómo utilizar estas instrucciones con el siguiente ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Prueba DOM</title>
</head>
<body>
    <h1>Ejemplo de prueba del DOM</h1>
    <p>Párrafo de prueba</p>
    <ul id="lista">
        <li>Uno</li>
        <li>Dos</li>
        <li>Tres</li>
    </ul>
    <img id="imagen1"
     src="imagen.png" title="Imagen de prueba">
</body>
</html>
```

Sobre este ejemplo HTML, ejecutamos las siguientes instrucciones JavaScript. Deberemos ejecutarlas *al final* del documento, cuando ya se haya cargado:

```
let lista = document.getElementById("lista");
let items = lista.getElementsByTagName("li");
alert(items.length);    // 3
for(let i = 0; i < items.length; i++)
{
    // Uno, Dos, Tres, respectivamente
    alert(items[i].textContent);
}
let imagen = document.getElementById("imagen1");
// Ruta hasta imagen "imagen.png"
alert(imagen.src);
```

> **Ejercicio 1**:
>
> Crea un documento HTML llamado **pruebaDOM.html**, con este contenido (cópialo y pégalo):

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="utf-8">
    <title>Prueba con el DOM</title>
</head>
<body>
<p>Lorem ipsum dolor sit amet, <a href="http://prueba">consectetuer
adipiscing elit</a>. Sed mattis enim vitae orci. Phasellus libero.
Maecenas nisl arcu, consequat congue, commodo nec, commodo ultricies,
turpis. Quisque sapien nunc, posuere vitae, rutrum et, luctus at,
pede. Pellentesque massa ante, ornare id, aliquam vitae, ultrices
porttitor, pede. Nullam sit amet nisl elementum elit convallis
malesuada. Phasellus magna sem, semper quis, faucibus ut, rhoncus non,
mi. <a href="http://prueba2">Fusce porta</a>.</p>

<p>Aenean at nisl. Maecenas egestas dapibus odio. Vestibulum ante ipsum
primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin
consequat auctor diam. <a href="http://prueba">Ut bibendum blandit est</a>.
Curabitur vestibulum. Nunc malesuada porttitor sapien. Aenean a lacus et
metus venenatis porta. Suspendisse cursus, sem non dapibus tincidunt,
lorem magna porttitor felis, id sodales dolor dolor sed urna. Sed rutrum
nulla vitae tellus. Sed quis eros nec lectus tempor lacinia. Aliquam nec
lectus nec neque aliquet dictum. Etiam <a href="http://prueba3">consequat
sem quis massa</a>.</p>

<p id="p3">Vestibulum aliquet, nulla sit amet imperdiet suscipit, nunc
erat laoreet est, a <a href="http://prueba">aliquam leo odio sed sem</a>.
Quisque eget eros vehicula diam euismod tristique. Proin
<a href="http://prueba4">egestas</a> adipiscing ligula. Duis iaculis
laoreet turpis. Mauris mollis est sit amet diam. Curabitur hendrerit,
eros quis malesuada tristique, ipsum odio euismod tortor, a vestibulum
nisl mi at odio. <a href="http://prueba5">Sed non lectus non est
pellentesque</a> auctor.</p>

</body>
</html>
```

Después, añade un bloque de código JavaScript donde, utilizando los métodos de acceso al DOM vistos aquí, muestres con mensajes (alert) la siguiente información:

- Cuántos enlaces tiene la página en total
- A qué dirección enlaza el primer enlace (deberás acceder a su propiedad *href*)
- Cuántos enlaces hay en el tercer párrafo en total

Observa también que, para que el código Javascript funcione en este ejemplo, deberemos añadirlo al final de la página, ya que de lo contrario aún no se habrán cargado los contenidos (sólo el bloque *head*), y no tendrá nada que examinar al ejecutarse.

3.2. Uso del *query selector*

Con la aparición de la librería *jQuery* en el año 2006, algunas de las operaciones anteriores se pudieron expresar de forma más simple gracias a la sintaxis incorporada por el propio lenguaje. Así, acceder a un elemento por su *id* era tan sencillo como hacer `$("#id")`. Es por ello que, con el tiempo, el propio lenguaje JavaScript incorporó el uso de esta sintaxis, a través del método `querySelector` del documento. Veamos algunos ejemplos:

- `document.querySelector("a")` obtendría el primer enlace de la página
- `document.querySelector(".miClase")` obtendría el primer elemento de clase *miClase*
- `document.querySelector("#miID")` obtendría el elemento con el id *miID*
- `document.querySelector("h1, h2")` obtendría el primer encabezado *h1* o *h2*
- `document.querySelector("div > p")` obtendría el primer párrafo que fuera hijo inmediato de un *div*
- ...

Como alternativa al método `querySelector`, también disponemos de `querySelectorAll`, que obtiene una lista (array) con todos los elementos que cumplan el patrón indicado. Por ejemplo, `document.querySelectorAll("a")` obtendrá un array con todos los enlaces del documento. Así podríamos obtenerlos y mostrar sus títulos (atributo *title*):

```
let enlaces = document.querySelectorAll("a");
enlaces.forEach(elem => console.log(elem.title));
```

> **Ejercicio 2**:
>
> Crea un documento HTML llamado **pruebaDOM2.html**, con el mismo contenido que el anterior, y trata de resolverlo usando el *query selector* esta vez.

3.2.1. Búsquedas de elementos y recorrido en bucles

A la hora de utilizar las instrucciones *getElementXXXX* o el *query selector* hay que tener en cuenta algunas diferencias importantes. Por ejemplo, el tipo de colección que se devuelve con uno u otro es distinto, y eso hace que no se pueda utilizar la instrucción `forEach` si empleamos las instrucciones *getElementXXXX*:

```
let parrafos = document.getElementsByTagName("p");
parrafos.forEach(p => { ... });     // ERROR
```

En cambio, sí podemos recorrerlo con un *for* tradicional:

```
let parrafos = document.getElementsByTagName("p");
for (let p of parrafos) { ... }     // OK
```

Si empleamos el *query selector*, podremos utilizar uno u otro tipo de recorrido indistintamente:

```
let parrafos = document.querySelectorAll("p");
parrafos.forEach(p => { ... });     // OK
for (let p of parrafos) { ... }     // OK
```

3.3. Manipulación del DOM

Una vez sabemos acceder a los elementos del documento, podemos modificar el contenido del mismo: añadir nuevos elementos, borrar elementos existentes o modificar el contenido de los que hay.

3.3.1. Crear contenidos

Para crear contenido, seguiremos estos pasos:

1. Crearemos un objeto del tipo que queramos añadir (párrafo, *h1*, *div*, etc.)
2. Crearemos el contenido de dicho objeto
3. Añadiremos el contenido al objeto
4. Añadiremos el objeto al documento, en la posición que queramos

Por ejemplo, si queremos añadir un párrafo con el texto "Hola" al final del documento, el código JavaScript sería el siguiente:

```
// 1. Creamos el párrafo
let parrafo = document.createElement("p");
// 2. Creamos su contenido
let contenido = document.createTextNode("Hola");
// 3. Añadimos el contenido
parrafo.appendChild(contenido);
// 4. Añadimos el párrafo
document.body.appendChild(parrafo);
```

En el caso de que quisiéramos añadir este mismo párrafo al final de un div con *id = div1*, primero localizaríamos este div, y luego le añadiríamos el párrafo a él, en lugar de al body:

```
// 0. Buscamos el lugar
let div = document.getElementById("div1");
// 1. Creamos el párrafo
let parrafo = document.createElement("p");
// 2. Creamos su contenido
let contenido = document.createTextNode("Hola");
// 3. Añadimos el contenido
parrafo.appendChild(contenido);
// 4. Añadimos el párrafo
div.appendChild(parrafo);
```

Alternativamente, también podemos emplear la instrucción `insertBefore` para añadir un elemento antes de otro. Por ejemplo, este código añade un nuevo item en una lista, justo delante del tercer elemento. Además, cambia el contenido de ese elemento para indicar que ahora es el cuarto:

```
<ul id="lista">
    <li>Primero</li>
    <li>Segundo</li>
    <li>Tercero</li>
</ul>
<script>
    let lista = document.getElementById("lista");
    let item3 = lista.children[2];
    let nuevoItem = document.createElement("li");
    nuevoItem.innerText = "Nuevo tercero";
    lista.insertBefore(nuevoItem, item3);
    item3.innerText = "Nuevo cuarto";
</script>
```

3.3.2. Modificar contenido existente

Si queremos modificar el contenido de un elemento existente, podemos acudir a la propiedad `innerHTML` de dicho elemento (una vez encontrado), o también la propiedad `innerText` (si sólo queremos cambiar el texto, sin subetiquetas), y le cambiamos el valor al del nuevo contenido. Por ejemplo, si queremos cambiar el contenido del párrafo con *id*="*miParrafo*":

```
let parrafo = document.getElementById("miParrafo");
parrafo.innerHTML = "Nuevo <strong>contenido</strong> del párrafo";
```

3.3.3. Borrar contenido

Para borrar un contenido de la página, deberemos localizar el elemento (párrafo, h1, etc.) que queremos borrar, acceder a su padre, y decirle que borre a ese hijo. Por ejemplo, si queremos borrar un div cuyo id es *div1*:

```
let div = document.getElementById("div1");
div.parentNode.removeChild(div);
```

> **Ejercicio 3:**
>
> Modifica el ejercicio *pruebaDOM.html* en un archivo llamado **pruebaDOM3.html**, añadiendo las siguientes funcionalidades:
>
> - Añadir un nuevo párrafo al final del documento que diga "Nuevo párrafo".
> - Borrar el segundo párrafo del documento

3.3.4. Acceder a atributos concretos

En ocasiones nos puede interesar acceder a un atributo concreto de un elemento para ver o modificar su valor. Por ejemplo, si queremos acceder al atributo *class* de un párrafo cuyo id es *p1*:

```
let parrafo = document.getElementById("p1");
let claseParrafo = parrafo.className;
```

En general, los nombres de atributos coinciden con los que hayamos puesto en el HTML (*style*, *href*...) salvo algunas excepciones como *className*. Algunos de ellos no dejan modificar su valor (sólo leerlo), pero otros sí.

Caso práctico: acceso a los estilos CSS

Una de las principales utilidades del acceso a los atributos es poder acceder y modificar algunos estilos de la página en tiempo real. Por ejemplo, cambiar la anchura del párrafo del ejemplo anterior (*width*), o la visibilidad (*display*), para mostrarlo u ocultarlo:

```
parrafo.style.width="50%";
parrafo.style.display="none";
```

Normalmente, para acceder a las propiedades CSS (como *width*, *display*...) se accede a la propiedad `style`, seguida del atributo o propiedad a modificar, que suele tener el mismo nombre que tienen en CSS; en el caso de que la propiedad tenga un nombre compuesto en CSS (*background-color*, *font-weight*...), se quita el guión y se pone la primera letra de la siguiente palabra en mayúsculas en la propiedad JavaScript (*backgroundColor*, *fontWeight*).

Este ejemplo cambia el color de fondo del párrafo anterior, y lo establece en rojo (observa que el valor de la propiedad siempre va entre comillas):

```
parrafo.style.backgroundColor="red";
```

Ejercicio 4:

Crea otra versión del ejercicio *pruebaDOM.html* anterior en un archivo **pruebaDOM4.html**. Añade estos cambios sobre la versión inicial:

- Pon el color de fondo del primer párrafo en amarillo.
- Oculta el segundo párrafo