Lenguaje de Programación

JavaScript Avanzado (DOM)

Contenido

1. Árbol de nodos 3
2. Tipos de nodos 4
   2.1. Elementos vs nodos 4
3. Acceso directo a los nodos 4
   3.1. getElementsByTagName() 4
   3.2. getElementsByName() 5
   3.3. getElementById() 5
   3.4. getElementsByClassName() 6
4. Creación y eliminación de nodos 6
   4.1. Creación de elementos HTML simples 6
   4.2. createElement vs innerHTML 7
   4.3. Eliminación de nodos 8
5. Acceso directo a los atributos HTML 8
   5.1. Acceso a las propiedades CSS de un elemento. 10
6. Inserción de nodos DOM 11
   6.1. appendChild 11
   6.2. insertBefore 11
   6.3. ¿insertAfter? 12
   6.4. replaceChild 13
   6.5. remove 13
   6.6. removeChild 13
   6.7. cloneNode 14
7. Propiedades DOM 14

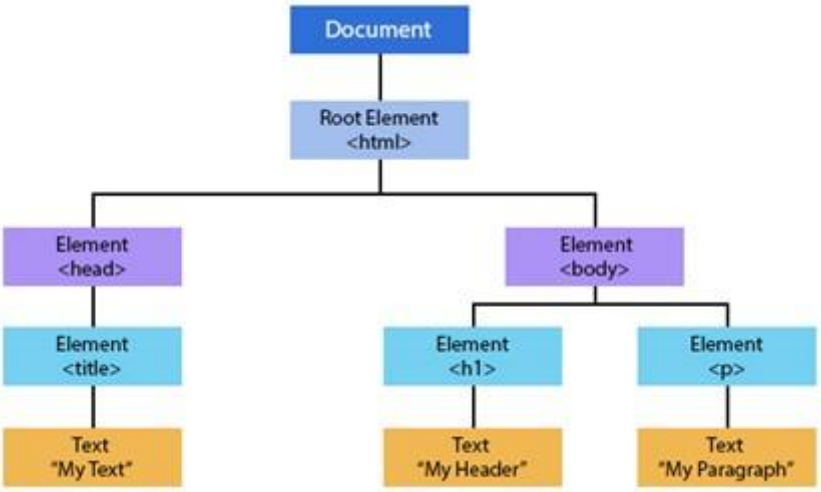
La creación del Document Object Model o DOM es una de las innovaciones que más ha influido en el desarrollo de las páginas web dinámicas y de las aplicaciones web más complejas.

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

1.Árbol de nodos

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

Para poder utilizar las utilidades de DOM, es necesario "*transformar*" la página original. Una página HTML normal no es más que una sucesión de caracteres, por lo que es un formato muy difícil de manipular. Por ello, los navegadores web transforman automáticamente todas las páginas web en una estructura más eficiente de manipular.



Esta transformación la realizan todos los navegadores de forma automática y nos permite utilizar las herramientas de DOM de forma muy sencilla. Todos los elementos de los documentos HTML se transforman en elementos llamados ***nodos***, que están interconectados y que representan los contenidos de las páginas web y las relaciones entre ellos. Por su aspecto, la unión de todos los nodos se llama *"árbol de nodos"*.

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

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

La transformación automática de la página en un árbol de nodos siempre sigue las mismas reglas:

- Las etiquetas HTML se transforman en dos nodos: el primero es la propia etiqueta y el segundo nodo es hijo del primero y consiste en el contenido textual de la etiqueta.

- Si una etiqueta HTML se encuentra dentro de otra, se sigue el mismo procedimiento anterior, pero los nodos generados serán nodos hijo de su etiqueta padre.

2. Tipos de nodos

La especificación completa de DOM define 12 tipos de nodos, aunque las páginas HTML habituales se pueden manipular manejando solamente cuatro o cinco tipos de nodos:

- *Document*, nodo raíz del que derivan todos los demás nodos del árbol.
- *Element*, representa cada una de las etiquetas HTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
- *Attr*, se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas HTML, es decir, uno por cada par *atributo = valor*.
- *Text*, nodo que contiene el texto encerrado por una etiqueta HTML.
- *Comment*, representa los comentarios incluidos en la página HTML.

Los otros tipos de nodos existentes que no se van a considerar son *DocumentType, CDataSection, DocumentFragment, Entity, EntityReference, ProcessingInstruction* y *Notation*.

2.1. Elementos vs nodos

De los dos, los elementos son los más fáciles de entender, ya que son solo elementos HTML - como una etiqueta div, span o body. Generalmente, cuando se trabaja con el DOM, estarás trabajando con elementos, ya que la mayoría de las veces se desea interactuar con elementos HTML.

Los nodos, en cambio, son la versión más genérica de un elemento. Un nodo podría ser un elemento HTML, pero también podría ser cualquier otra cosa en un documento HTML, como texto o comentarios. Esto hace que sea más difícil trabajar con los nodos, ya que la mayoría de las veces, cuando se trabaja con DOM, no nos importan cosas como los nodos de texto o los nodos de comentarios y solo nos importan los nodos de los elementos.

3. Acceso directo a los nodos

Antes de usar los distintos métodos de acceso al árbol DOM, es necesario recordar que el acceso a los nodos, su modificación y su eliminación solamente es posible cuando el árbol DOM ha sido construido completamente, es decir, después de que la página HTML se cargue por completo.

3.1. getElementsByTagName()

Obtiene todos los elementos de la página HTML cuya etiqueta sea igual que el parámetro que se le pasa a la función.

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

En este ejemplo, como se quieren obtener todos los párrafos de la página, se utilizar el valor *document* como punto de partida de la búsqueda.

El valor devuelto es un array con todos los nodos que cumplen la condición de que su etiqueta coincide con el parámetro proporcionado.

3.2. getElementsByName()

Es similar a la anterior, pero en este caso se buscan los elementos cuyo atributo *name* sea igual al parámetro proporcionado.

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

En un documento HTML pueden existir varios elementos con el mismo valor de la propiedad *name*, por ello este método devuelve un array con todos los elementos que cumplen esta condición.

En el caso de los elementos *radiobutton*, el atributo *name* es común a todos los *radiobutton* que están relacionados, por lo que la función devuelve una colección de elementos.

3.3. getElementById()

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

El método *getElementById()* devuelve el elemento HTML cuyo atribudo *id* coincide con el parámetro indiciado en la función. El atributo *id* debe ser único para cada elemento de una misma página, la función devuelve únicamente el nodo deseado.

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

3.4. getElementsByClassName()

Existe otro método para acceder a nodos DOM, aunque no es muy utilizado. Este método es ***getElementsByClassName()***. Nos permitirá seleccionar aquellos elementos cuyo contenido del atributo ***class*** es que indiquemos como argumento.

```
<!doctype html>
<head>
    <meta charset="UTF-8">
    <title>Documento HTML</title>
    <style type="text/css">
        .rojo {
            color:red;
        }
    </style>
</head>
<body>
….
<script>
    var elementos= document.getElementsByClassName("rojo");
    alert("Número de párrafos rojos: "+elementos.length);
</script>
</body>
```

4. Creación y eliminación de nodos

Las operaciones habituales son las de crear y eliminar nodos del árbol DOM, es decir, crear y eliminar "*trozos*" de la página web.

4.1. Creación de elementos HTML simples

Como se ha visto, un elemento HTML sencillo, como por ejemplo un párrafo, genera dos nodos: el primer nodo es de tipo *Element* y representa la etiqueta <p> y el segundo nodo es de tipo *Text* y representa el contenido textual de la etiqueta <p>.

Por lo tanto, para crear un nodo HTML sencilla se necesitan cuatro pasos:

1. Creación de un nodo de tipo Element que represente al elemento.
2. Creación de un nodo de tipo Text que represente el contenido del elemento.
3. Añadir el nodo Text como nodo hijo del nodo Element.
4. Añadir el nodo Element a la página, en forma de nodo hijo del nodo correspondiente al lugar en el que se quiere insertar el elemento.

Para crear un nuevo nodo usaremos el método ***createElement("tipo_elemento")*** y para crear el nodo de texto usaremos el método ***createTextNode("contenido_elemento")***. Ambos métodos heredan del objeto ***document***. Tanto para añadir el nodo de tipo texto al nodo, como para añadir el nuevo elemento al documento HTML usaremos el método ***nodo.appendChild(nodo_hijo)***.

Veamos un ejemplo para añadir un nuevo párrafo al body del documento HTML:

```
// Creamos el nuevo nodo:
var parr = document.createElement("p");

// Creamos el nodo de tipo texto:
var parrText = document.createTextNode("Hola Mundo!");

//  Añadir el nodo Text como hijo del nodo Element:
parr.appendChild(parrText);

// Añadimos el nodo al documento HTML:
document.body.appendChild(parr);
```

4.2. createElement vs innerHTML

Una vez vistos los pasos para crear nuevos elementos en un documento HTML, puede que se nos ocurra hacer lo mismo usando la propiedad innerHTML. Sería mucho más fácil hacer, por ejemplo:

```
document.getElementById("div1").innerHTML = "<p>Hola, soy un nuevo nodo</p>";
```

que:

```
var p = document.createElement("p");
var pText = document.createTextNode("Hola, soy un nuevo nodo");
p.appendChild(pText);
document.getElementById("div1").appendChild(p);
```

El resultado, en apariencia es exactamente el mismo, luego, ¿por qué complicarse la vida? A continuación, enumeramos las ventanas que tiene crear los nodos con createElement en lugar de con innerHTML:

1. createElement, aún **mantiene un puntero al elemento DOM** después de la inserción del elemento (la variable o constante con la que se crea). Después de que se inserta innerHTML, no hay puntero para el elemento DOM, debes volver a seleccionarlo a través de getElementById.

2. createElement puede obtener un **manejador de eventos**, y el nuevo DOM generado por innerHTML no puede obtener el manejador de eventos originales.

3. Crear un nuevo elemento y agregarlo al árbol DOM proporciona un **mejor rendimiento** que innerHTML. El uso de innerHTML hace que los navegadores web analicen y vuelvan a crear todos los nodos DOM dentro del elemento, por ejemplo, un div. Por lo tanto, es menos eficiente que crear un nuevo elemento y agregarlo al elemento padre.

4. createElement **es más seguro**. Si se incluye el contenido usando innerHTML, el código puede ser malicioso y puede inyectarse y ejecutarse.

5. Lectura y mantenimiento de createElement es mucho más clara.

4.3. Eliminación de nodos

Para eliminar un nodo solamente es necesario utilizar la función ***removeChild()***. Con *removeChild()* solo es posible eliminar nodos hijos, luego para eliminar un nodo concreto tendremos que acceder antes al padre de éste nodo y eliminar el hijo. Para acceder al nodo padre usaremos ***parentNode()***. Veamos un ejemplo:

```
var parr = document.getElementById("p");
parr.parentNode.removeChild(parr);
```

O podemos hacerlo en un solo paso así:

```
document.getElementById("p").parentNode.removeChild(document.getElementById("p"));
```

Otra forma de eliminar un nodo es con el método ***remove()***. Lo debemos ejecutar sobre el nodo a borrar, por ejemplo:

```
var el = document.getElementById('div-02');
el.remove();
```

5. Acceso directo a los atributos HTML

El siguiente paso consiste en acceder y/o modificar sus atributos y propiedades. Mediante DOM, es posible acceder de forma sencilla a todos los atributos HTML y todas sus propiedades CSS de cualquier elemento de la web.

Los atributos HTML de los elementos de la web se transforman automáticamente en propiedades de los nodos.

Para acceder a las propiedades de un nodo podremos hacerlo de dos formas:

1. Accediendo directamente al nombre de la propiedad a leer o modificar.

```
document.getElementById(“button1”).value = “Aceptar”;
```

2. A través del uso de los métodos ***setAttribute*** (atributo,valor), ***getAttribute***(atributo) y ***removeAtribute***(atributo)

```
elemento = document.getElementsByTagName("a")[0]
direccion = elemento.getAttribute("href")
```

```
elemento.setAttribute("href","http://www.google.com/")
```

3. Otro método muy usado para el acceso a los nodos DOM es mediante ***querySelector()*** y ***querySelectorAll()***:

   - **querySelector ()** → Devuelve el primer elemento que coincida con un selector CSS especificado. El selector o selectores CSS se pasan como parámetros.

     ```
     document.querySelector("h3").style.color = "pink";
     ```

     Este método es capaz de mucho más, ya que puede utilizar todo el potencial de los selectores CSS.

     ```
     document.querySelector("h3.clase1").style.color = "pink";
     document.querySelector("#segundo_div h3").style.color = "pink";
     ```

   - **querySelectorAll ()** → Este método funciona como querySelector() con la diferencia de que este no devuelve solamente el primer elemento que coincida sino todos ellos. La manera en que nos devuelve las coincidencias es a través de un array, así que para recuperar una de ellas deberemos hacerlo con ayuda de los corchetes [].

```
var titulo = document.querySelectorAll("h3");
titulo[1].style.color = "orange";
```

5.1. Acceso a las propiedades CSS de un elemento.

Para acceder o modificar las propiedades CSS se debe utilizar el atributo ***style***.

```
var imagenes = document.getElementsByTabName("img");
for (let i of imágenes)
      alert(i.style.margin);
```

Hay que tener en cuenta que, si el **nombre de la propiedad CSS es compuesto**, se accede a su valor modificando ligeramente el nombre en JavaScript.

```
var parr = document.getElementById("p");
alert(p.style.fontWeight);
```

La transformación consiste en eliminar todos los guiones medios (-) y escribir en mayúscula la letra siguiente a cada guion medio.

El único atributo HTML que no tiene el mismo nombre en HTML y en las propiedades DOM es el atribulo ***class***. Como la palabra *class* está reservada por JavaScript, no es posible utilizarla para acceder al atributo *class* del elemento HTML. En su lugar, DOM utiliza el nombre ***className*** para acceder al atributo *class* de HTML.

Como hemos visto, la forma más usual de acceder a las propiedades CSS de un elemento es mediante el atributo ***elemento.style.propiedadCSS***. Esta forma de acceder solo nos devolverá el valor de aquellas propiedades CSS que se hayan definido con la propiedad ***style*** del elemento HTML en el documento HTML. Habrá muchas ocasiones en las que queramos saber los valores CSS que se están aplicando a un elemento desde un fichero CSS externo, para ello tendremos que usar el siguiente método:

```
let elemento = document.getElementById("d1");
let elementStyle = window.getComputedStyle(elemento);
dim = elementStyle.getPropertyValue('width');
```

En el ejemplo anterior se accede al elemento HTML, se crea un objeto de tipo *ComputedStyle,* que contendrá **todas** las propiedades CSS que el navegador aplica a este elemento y, por último, se obtiene la propiedad *width*.

Visita esta web y prueba el método classList, puede ser muy útil.

6. Inserción de nodos DOM

Acabamos de ver cómo crear nodos en un documento HTML, pero también hemos visto que estos nodos creados se mantienen en el «limbo» del documento hasta que los incluimos en el árbol del mismo usando ***appendChild()***.

Para trabajar incorporando, modificando o eliminando nodos, contamos con otros métodos que vamos a ver a continuación.

6.1. appendChild

Por medio de *appendChild* podemos incluir en un nodo un nuevo hijo, de esta manera:

```
elemento_padre.appendChild(nuevo_nodo);
```

El nuevo nodo **se incluye inmediatamente después de los hijos ya existentes** —si hay alguno— y el nodo padre cuenta con una nueva rama.

Por ejemplo, el siguiente código:

```
var lista = document.createElement('ul');
var item = document.createElement('li');
lista.appendChild(item);
```

crea un elemento *ul* y un elemento *li*, y convierte el segundo en hijo del primero.

6.2. insertBefore

insertBefore nos permite elegir un nodo del documento e incluir otro antes que él. Su sintaxis es:

```
elemento_padre.insertBefore(nuevo_nodo,nodo_de_referencia);
```

Si tuviéramos un fragmento de un documento como éste:

```
<div id="padre">
```

```
        <p>Un párrafo.</p>
        <p>Otro párrafo.</p>
    </div>
```

y quisiéramos añadir un nuevo párrafo antes del segundo, lo haríamos así:

```
// Creamos el nuevo párrafo
var nuevo_parrafo = document.createElement('p').
nuevo_parrafo.appendChild(document.createTextNode('Nuevo párrafo.'));

// Recogemos en una variable el segundo párrafo
var segundo_p = document.getElementById('padre').getElementsByTagName('p')[1];

// Y ahora lo insertamos
document.getElementById('padre').insertBefore(nuevo_parrafo,segundo_p);
```

También existe un nuevo método Node.before(nuevo_nodo). Pruébalo.

6.3. ¿insertAfter?

Cuando se empieza a trabajar con el DOM, uno echa de menos un método que permita incluir un nodo detrás de otro. Como no lo hay, todo programador de JavaScript acaba creándose una función propia que haga eso exactamente. La siguiente es un ejemplo:

```
    function insertAfter(i,e){
        if(e.nextSibling){ // El siguiente hermano de e
            e.parentNode.insertBefore(i,e.nextSibling);
        } else {
            e.parentNode.appendChild(i);
        }
    }
```

Los parámetros son:

- i: el nodo que se quiere insertar.
- e: el nodo tras el que se quiere insertar otro.

También existe un nuevo método Node.after(nuevo_nodo). Pruébalo.

6.4. replaceChild

Para reemplazar un nodo por otro contamos con *replaceChild*, cuya sintaxis es:

```
elemento_padre.replaceChild(nuevo_nodo,nodo_a_reemplazar);
```

Con el mismo marcado que para el ejemplo de *insertBefore*, si quisiéramos sustituir el segundo párrafo por el que creamos, lo haríamos así:

```
document.getElementById('padre').replaceChild(nuevo_parrafo,segundo_p);
```

Se devuelve el nodo reemplazado.

6.5. remove

Dado que podemos incluir nuevos hijos en un nodo, tiene sentido que podamos eliminarlos. Ejemplo:

```
const element = document.getElementById("demo");
element.remove();
```

6.6. removeChild

Este método, a diferencia del anterior, devuelve el nodo eliminado o null si no existe el nodo a borrar.

La sintaxis es:

```
elemento_padre.removeChild(nodo_a_eliminar);
```

Con el mismo ejemplo anterior, eliminar el segundo párrafo sería algo tan sencillo como:

```
document.getElementById('padre').removeChild(segundo_p);
```

6.7. cloneNode

Por último, podemos crear un clon de un nodo por medio de *cloneNode*:

```
elemento_a_clonar.cloneNode(booleano);
```

El booleano que se pasa como parámetro define si se quiere clonar el elemento —con el valor false—, o bien si se quiere clonar con su contenido —con el valor true—, es decir, el elemento y todos sus descendientes.

Si quisiéramos clonar nuestro *div* de ejemplo con el siguiente código:

```
var clon = document.getElementById('padre').cloneNode(false);
```

clon contendría un elemento *div*, pero de esta manera:

```
var clon = document.getElementById('padre').cloneNode(true);
```

contendría un elemento div con dos párrafos que contendrían los mismos nodos de texto que el original.

7.Propiedades DOM

Propiedad	Descripción
attributes	Selecciona todos los atributos del elemento. Devuelve un array de objetos (exactamente un **NamedNodeMap**) atributos. Se puede acceder a cada atributo con ***name*** (para acceder al nombre del atributo concreto) y con ***value*** (para acceder al valor del atributo concreto). Antes de obtener los atributos, se puede preguntar si un elemento tiene alguno con: **elemento.hasAttributes()**;
childNodes	Devuelve un array con todos los elementos que cuelgan del actual.
children	Devuelve un array con todos los nodos hijo. La diferencia con ***childNodes*** es que ***children*** trabaja en elementos y ***childNodes*** en nodos que incluyen nodos que no son elementos, como nodos de texto y comentarios.
firstChild	Selecciona el primer elemento hijo del actual.

lastChild	Selecciona el último elemento hijo del actual.
firstElementChild	Devuelve el primer nodo hijo.
lastElementChild	Devuelve el último nodo hijo.
childElementCount	Devuelve el número de nodos hijo.
nextSibling **nextElementSibling**	Selecciona el siguiente hermano siguiendo el árbol DOM, si existe. nextSibling acepta como nodo comentarios o texto sin etiquetar.
previousSibling **previousElementSibling**	Obtiene el nodo del mismo nivel, anterior en el árbol DOM, si existe. previousSibling acepta como nodo comentarios o texto sin etiquetar.
nodeName	Obtiene el nombre del nodo. Si es un elemento devuelve su tipo, si es un atributo su nombre.
nodeType	Devuelve el tipo de nodo. Si lo usamos sobre un elemento devuelve 1, si es un atributo 2, si es un texto 3. Las constantes pertenecientes a los objetos de nodo: **ELEMENT_NODE, ATTRIBUTE_NODE y TEXT_NODE** están asociadas a esos valores.
parentNode	Obtiene el padre del nodo al que se aplica este método.
textContent	Obtiene el contenido (sólo el texto) del nodo, sea del tipo que sea. También permite modificarlo.

Puedes ver más métodos de acceso y modificación del DOM en:
https://www.w3schools.com/jsref/dom_obj_all.asp.