DOM

Ángel Villalba Fdez-Paniagua

Table of Contents

1. Document Object Model

Cuando el navegador carga la página web, crea un árbol del DOM que es un módelo de la página. Este módelo se almacena en la memoria del navegador y puede contener los siguientes tipos de nodos:

- Nodo documento: cada nodo del árbol del DOM representa un elemento, un atributo o un trozo de texto que se encuentra en la página web. El nodo del documento (`document`) representa toda la página web, por lo que es el nodo raíz del árbol y a partir del cual hay que navegar hasta llegar al resto de nodos y poder acceder a ellos.
- Nodo elemento: todos los elementos de HTML se encargan de estructurar la página web, y cada uno de ellos representa algo de esa página (una párrafo, un botón...), donde empieza un elemento y donde acaba. Para acceder a estos nodos, tendremos que usar algunos métodos que veremos más adelante. Y una vez que hayamos obtenido estos nodos, podremos acceder a sus nodos de texto y de atributos, tanto para obtener los valores, como para modificarlos.
- Nodo atributo: las etiqueta de apertura de un elemento HTML puede llevar atributos, y estos se representan como nodos en el árbol del DOM. Estos nodos no son hijos del elemento que lo contiene sino que son parte de ese elemento. Podemos obtener o modificar el valor de estos nodos usando unos métodos, una vez que ya hemos obtenido el elemento. Estos nodos se suelen modificar para cambiar el aspecto del elemento.
- Nodo texto: los nodos de texto nos devuelven el texto del elemento que los contiene. Estos nodos no pueden tener nodos hijos, por lo que si nos encontramos un nodo con texto, y parte de ese texto esta entre etiquetas HTML, ese elemento tendra un texto como hijo, y a su vez un nodo elemento como hijo (hermano del nodo texto), que tendrá el resto del texto como un nodo de texto que será hijo de este último.

Todos estos nodos se relacionan entre ellos como si fueran una familia y tuvieramos un árbol familiar. Tendremos a los padres, los hijos, los hermanos, los descendientes... Dentro del árbol del DOM, todos los nodos van a ser descendientes del nodo *document*.

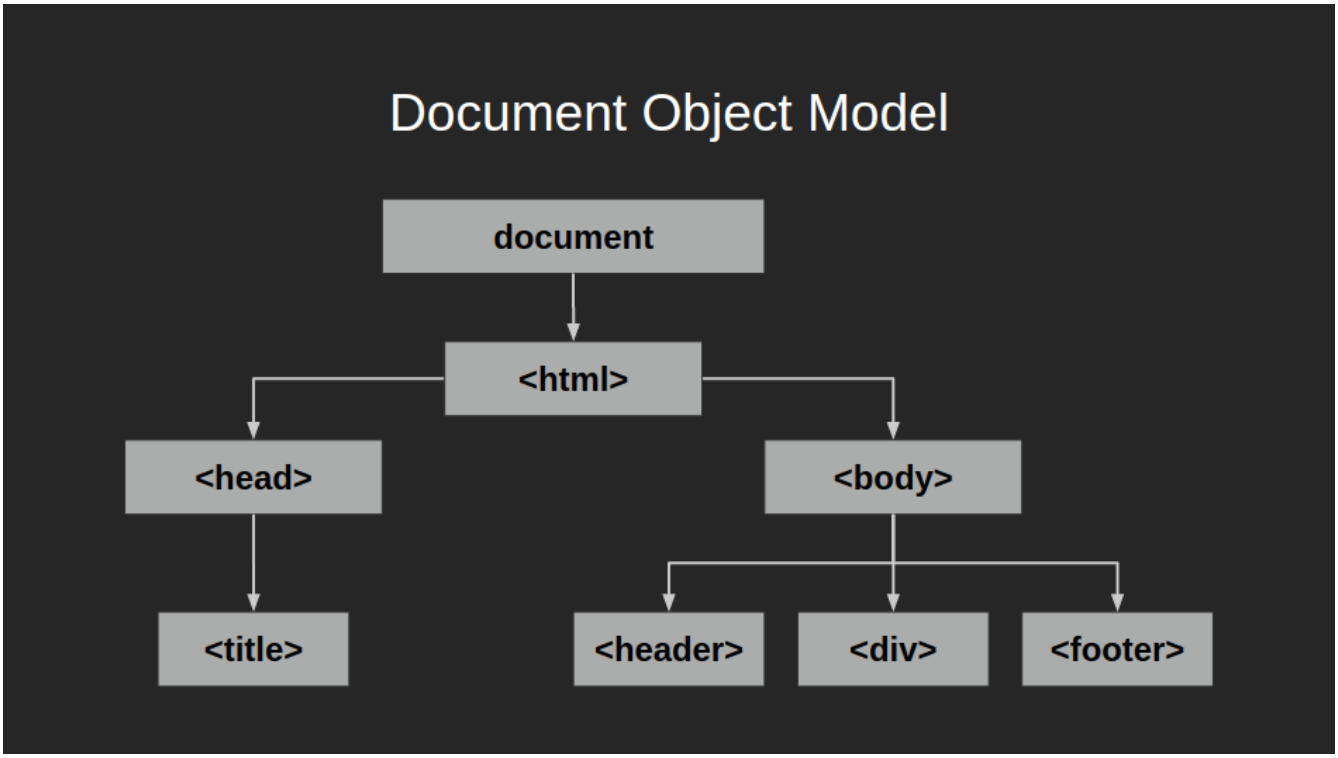


Figure 1. Document Object Model

Cada uno de estos nodos es un objeto que contiene unos métodos y unas propiedades. Nosotros podrémos modificarlos mediante los scripts, y cada cambio se va a realizar sobre el árbol del DOM, y por lo tanto se reflejará en el navegador.

Para saber de que tipo es un nodo, una vez que lo hemos obtenido, podremos acceder a su propiedad `nodeType` que nos devolverá un número. Este número se corresponde con el tipo de nodo que es, los cuales se pueden ver en la siguiente tabla:

Table 1. Tipos de Nodos

Tipo	Número
ELEMENT_NODE	1
ATTRIBUTE_NODE	2
TEXT_NODE	3
CDATA_NODE	4
ENTITY_REFERENCE_NODE	5
ENTITY_NODE	6
PROCESSING_INSTRUCTION_NODE	7
COMMENT_NODE	8
DOCUMENT_NODE	9
DOCUMENT_TYPE_NODE	10
DOCUMENT_FRAGMENT_NODE	11
NOTATION_NODE	12

2. Árbol del DOM

El árbol del DOM es el árbol que contiene la representación de la página web. Para poder acceder y modificar los elementos de este árbol, necesitamos seguir los siguientes pasos:

1. Buscamos el nodo que representa al elemento que queremos obtener para acceder a el o modificarlo.
2. Usamos su contenido (texto, nodos hijos o atributos).

Podemos acceder a los elementos de tres formas distintas:

- Seleccionando un nodo de elemento: para acceder a un nodo se puede usar `getElementById()`, `querySelector()` o los métodos que nos permiten pasar entre los nodos de elementos.
- Seleccionando múltiples nodos de elementos: para obtener múltiples nodos, se puede usar `getElementsByClassName()`, `getElementsByTagName()` y `querySelectorAll()`.
- Pasar a través de nodos de elementos: de esta forma podemos movernos entre los nodos que están relacionados, `parentNode`, previousSibling `, `nextSibling`, `firstChild` y `lastChild`.

Y una vez que tenemos los elementos, ya se puede trabajar con ellos, aunque con lo que se trabaja es con los nodos que representan a esos elementos. En este caso podemos interactuar con los siguientes casos:

- Trabajar con el contenido HTML: podemos acceder al contenido HTML (`innerHTML`), al texto del elemento (`textContent`) o incluso crear nuevos nodos (`createElement()` y `createTextNode()`), añadirlos al árbol del DOM (`appendChild()`) y borrarlos (`removeChild()`).
- Acceso o actualización de los valores de los atributos: podemos obtener/modificar las clases (`className`) o el id (`id`), obtener/modificar un atributo (`getAttribute()` o `setAttribute`), comprobar que tiene el atributo (`hasAttribute()`) y borrarlo (`removeAttribute()`).
- Acceso o actualización de los nodos de texto: podemos obtener el valor de un nodo con `nodeValue` o incluso modificar ese valor asignandoselo, pero antes hay que poder acceder a ese nodo de texto usando `firstChild`.

Si obtenemos un elemento que vamos a usar varias veces, deberíamos de guardarlo en una variable para no tener que estar buscandolo cada vez que lo necesitemos.

3. Acceso a los elementos

Los métodos que se encargan de buscar en el árbol del DOM, pueden devolvernos un elemento, una lista de nodos (**NodeList**) o una colección de elementos HTML (**HTMLCollection**).

Algunas veces querremos obtener un único elemento y otras querremos obtener varios elementos. Si usamos un método que para obtener varios elementos, siempre nos devolverá una lista, incluso en el caso en que encuentre solo un elemento que coincida con lo que se busca. Y para acceder a estos elementos que nos llegan en la lista, usaremos los índices (acceso a un array).

También hay que tener en cuenta que si queremos que nuestra página sea rápida, tenemos que usar el método de búsqueda más adecuado en cada caso, es decir, si vamos a buscar un elemento por el id, deberíamos usar el método que se encarga de realizar una búsqueda por id en el árbol.

A continuación se van a mostrar todos los métodos que se pueden usar para acceder a los elementos del árbol del DOM.

3.1. Métodos que devuelven un elemento

3.1.1. getElementById()

Este método busca un elemento por su atributo **id**. Para poder devolver este elemento, tiene que existir un elemento con ese id dentro del documento HTML. Este es el método de búsqueda más rápido y eficiente que se puede usar porque reduce la búsqueda a un solo elemento, ya que dos elementos no pueden compartir el mismo valor para su atributo id.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1 class="titulo">Un título</h1>
  <p id="parrafo1">Un párrafo</p>
  <label for="nombre">Nombre: </label>
  <input type="text" name="input" id="nombre">
  <label for="email">Email: </label>
  <input type="email" name="input" id="email">
  <h1 class="titulo">Otro título</h1>
  <p>Otro párrafo</p>
  <ul>
    <li>Uno</li>
    <li>Dos</li>
  </ul>
</body>
<script>
  var parrafo1 = document.getElementById('parrafo1');
  console.log(parrafo1);
</script>
</html>
```

3.1.2. querySelector()

Usa la sintaxis de los **selectores de css** para obtener un elemento. En caso de que haya múltiples elementos que coincidan con el selector pasado como parámetro, se devolverá solamente la primera coincidencia.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1 class="titulo">Un título</h1>
  <p id="parrafo1">Un párrafo</p>
  <label for="nombre">Nombre: </label>
  <input type="text" name="input" id="nombre">
  <label for="email">Email: </label>
  <input type="email" name="input" id="email">
  <h1 class="titulo">Otro título</h1>
  <p>Otro párrafo</p>
  <ul>
    <li>Uno</li>
    <li>Dos</li>
  </ul>
</body>
<script>
  var parrafo2 = document.querySelector('body p');
  console.log(parrafo2);
</script>
</html>
```

3.2. Métodos que devuelven varios elementos

3.2.1. getElementsByClassName()

Devuelve una lista de elementos (*HTMLCollection*) que contienen la **clase** que se le pasa como parámetro al método. Este método es más rápido que el método `querySelectorAll()`.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1 class="titulo">Un título</h1>
  <p id="parrafo1">Un párrafo</p>
  <label for="nombre">Nombre: </label>
  <input type="text" name="input" id="nombre">
  <label for="email">Email: </label>
  <input type="email" name="input" id="email">
  <h1 class="titulo">Otro título</h1>
  <p>Otro párrafo</p>
  <ul>
    <li>Uno</li>
    <li>Dos</li>
  </ul>
</body>
<script>
  var titulos = document.getElementsByClassName('titulo');
  console.log(titulos);
</script>
</html>
```

3.2.2. getElementsByTagName()

Este método realiza la búsqueda por el **nombre de la etiqueta**, por lo que devolverá todos los elementos (*HTMLCollection*) **tagName** que se encuentre en el documento HTML. Este método también es más rápido que el método `querySelectorAll()`.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1 class="titulo">Un título</h1>
  <p id="parrafo1">Un párrafo</p>
  <label for="nombre">Nombre: </label>
  <input type="text" name="input" id="nombre">
  <label for="email">Email: </label>
  <input type="email" name="input" id="email">
  <h1 class="titulo">Otro título</h1>
  <p>Otro párrafo</p>
  <ul>
    <li>Uno</li>
    <li>Dos</li>
  </ul>
</body>
<script>
  var listItems = document.getElementsByTagName('li');
  console.log(listItems);
</script>
</html>
```

3.2.3. getElementsByName()

Devuelve una lista de elementos (*NodeList*) que contienen el atributo **name** que se le pasa como parámetro al método.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1 class="titulo">Un título</h1>
  <p id="parrafo1">Un párrafo</p>
  <label for="nombre">Nombre: </label>
  <input type="text" name="input" id="nombre">
  <label for="email">Email: </label>
  <input type="email" name="input" id="email">
  <h1 class="titulo">Otro título</h1>
  <p>Otro párrafo</p>
  <ul>
    <li>Uno</li>
    <li>Dos</li>
  </ul>
</body>
<script>
  var labels = document.getElementsByName('input');
  console.log(labels);
</script>
</html>
```

3.2.4. querySelectorAll()

Usa la sintaxis de los **selectores de css** para obtener todos los elementos (*NodeList*) que coincidan con el selector pasado como parámetro en el método.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1 class="titulo">Un título</h1>
  <p id="parrafo1">Un párrafo</p>
  <label for="nombre">Nombre: </label>
  <input type="text" name="input" id="nombre">
  <label for="email">Email: </label>
  <input type="email" name="input" id="email">
  <h1 class="titulo">Otro título</h1>
  <p>Otro párrafo</p>
  <ul>
    <li>Uno</li>
    <li>Dos</li>
  </ul>
</body>
<script>
  var parrafos = document.querySelectorAll('body p');
  console.log(parrafos);
</script>
</html>
```

3.3. Métodos que devuelven varios elementos

Estos métodos que pueden devolver más de un elemento, devuelven un NodeList o HTMLCollection aunque solo haya encontrado un único elemento. El orden en que se guardan los nodos en la lista, es el mismo que el orden en el que aparecen en la página HTML.

Una vez que ya tenemos la lista de elementos, podremos acceder a uno de ellos, o recorrerla para ejecutar código JavaScript por cada uno de los elementos.

Los NodeList son como arrays, pero en realidad no lo son. Son un tipo de objeto llamado **collection**, y como cualquier otro objeto, tiene una propiedad que nos dice el número de *items* que hay en la lista (`length`), y tiene un método (`item(index)`) que nos devuelve el nodo que se encuentra en la posición que se le pasa, aunque se suele acceder a ellos como a cualquier array (`listaNodos[2]`).

Los dos métodos de acceso a los nodos necesitan saber el índice del elemento al cual queremos acceder.

Podemos usar el método `item(index)` que nos devolverá un nodo de la lista de elementos. El *index* que se le pasa como parámetro indica la posición del elemento que queremos obtener.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1 class="titulo">Un título</h1>
  <p id="parrafo1">Un párrafo</p>
  <label for="nombre">Nombre: </label>
  <input type="text" name="input" id="nombre">
  <label for="email">Email: </label>
  <input type="email" name="input" id="email">
  <h1 class="titulo">Otro título</h1>
  <p>Otro párrafo</p>
  <ul>
    <li>Uno</li>
    <li>Dos</li>
  </ul>
</body>
<script>
  var parrafos = document.querySelectorAll('body p');
  console.log('El primer párrafo es: ', parrafos.item(0));
</script>
</html>
```

O podemos usar la sintaxis de los Arrays para acceder a estos elementos. Este método de acceso es más rápido que el anterior, por lo que se recomienda usarlo. Para acceder a un elemento solo hay que poner entre corchetes la posición en la que se encuentra.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1 class="titulo">Un título</h1>
  <p id="parrafo1">Un párrafo</p>
  <label for="nombre">Nombre: </label>
  <input type="text" name="input" id="nombre">
  <label for="email">Email: </label>
  <input type="email" name="input" id="email">
  <h1 class="titulo">Otro título</h1>
  <p>Otro párrafo</p>
  <ul>
    <li>Uno</li>
    <li>Dos</li>
  </ul>
</body>
<script>
  var parrafos = document.querySelectorAll('body p');
  console.log('El segundo párrafo es: ', parrafos[1]);
</script>
</html>
```

Y por último, tanto los NodeList, como los HTMLCollection se pueden recorrer para ejecutar el mismo código por cada uno de los elementos que hay en la lista. En este caso, como no son Arrays, sino que son Objetos, es mejor recorrerlas usando un bucle `for` normal.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1 class="titulo">Un título</h1>
  <p id="parrafo1">Un párrafo</p>
  <label for="nombre">Nombre: </label>
  <input type="text" name="input" id="nombre">
  <label for="email">Email: </label>
  <input type="email" name="input" id="email">
  <h1 class="titulo">Otro título</h1>
  <p>Otro párrafo</p>
  <ul>
    <li>Uno</li>
    <li>Dos</li>
  </ul>
</body>
<script>
  var parrafos = document.querySelectorAll('body p');

  for (var i = 0; i < parrafos.length; i++) {
    console.log(parrafos[i].firstChild.textContent);
  }
</script>
</html>
```

3.4. Ejercicio 1

- Ir a la página 'https://coinmarketcap.com/'
- Crear un script que obtenga todas las criptomonedas, las ordene por nombre y las muestre por consola
- Lanzar el script en la consola del navegador

3.5. Pasar a través de los nodos de elementos

Cuando ya tenemos un nodo seleccionado, se puede acceder a otro nodo que esté relacionado con el, usando alguna de las siguientes propiedades:

- **parentNode**: esta propiedad devuelve el nodo que contiene al elemento actual.
- **previousSibling**: esta propiedad devuelve el nodo siguiente al elemento actual.
- **nextSibling**: esta propiedad devuelve el nodo anterior al elemento actual.
- **firstChild**: esta propiedad devuelve el primer nodo hijo del elemento actual.

- **lastChild**: esta propiedad devuelve el último nodo hijo del elemento actual.
- **children**: esta propiedad devuelve todos los hijos del elemento actual.

Hay que tener cuidado, porque la mayor parte de los navegadores añaden un nodo de texto vacío por cada espacio en blanco o por cada retorno de carro. Antes de tratar de usar las propiedades anteriores, hay que asegurarse de que no se encuentran esos nodos vacíos.

En caso de que el elemento no tenga un elemento anterior, siguiente, o hijos, devolverán un valor **null**, y nunca nos van a servir para modificar esos nodos (son **propiedades de lectura**).

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <ul id="lista"><li id="uno">Uno</li><li id="dos">Dos</li><li id="tres">Tres</li><li
id="cuatro">Cuatro</li></ul>
</body>
<script>
  var lista = document.getElementById('lista');
  var listItems = document.getElementsByTagName('li');
  console.log('Padre: ', listItems.item(0).parentNode);
  console.log('Primer hijo: ', lista.firstChild);
  console.log('Segundo hijo: ', listItems[0].nextSibling);
  console.log('Tercer hijo: ', listItems[3].previousSibling);
  console.log('Último hijo: ', lista.lastChild);
  console.log('Todos los hijos: ', lista.children);
</script>
</html>
```

4. Obtener/Actualizar contenido de los elementos

Una vez tenemos un elemento, podemos acceder a su contenido, incluso actualizarlo. Hay que tener en cuenta el tipo del contenido del elemento para elegir la forma correcta de hacerlo.

- Podemos ir hasta los nodos de texto, y de esta forma nos aseguramos que el elemento solo contiene texto, y no otros elementos.
- Podemos trabajar con el contenido del elemento, lo que nos dará acceso a los elementos hijos y a los nodos de texto. Esta forma viene bien cuando tenemos que trabajar con un elemento que tiene nodos hijos y nodos de texto.

4.1. nodeValue

Cuando ya hemos seleccionado un nodo de texto, podemos acceder/modificar el texto usando la propiedad `nodeValue`.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
  <style>
    .ocultar {
      display: none;
    }
  </style>
</head>
<body>
  <h1 id="uno">Hola <span class="ocultar">a todos!</span>mundo</h1>
  <h1 id="dos">Hola <span class="ocultar">a todos!</span>mundo</h1>
  <h1 id="tres">Hola <span class="ocultar">a todos!</span>mundo</h1>
</body>
<script>
  var elemUno = document.getElementById('uno');
  console.log(elemUno.firstChild.nodeValue);
  elemUno.firstChild.nodeValue = 'ooo';
</script>
</html>
```

4.2. textContent

Cuando tenemos un elemento, podemos obtener todo el texto que contiene el elemento y sus hijos usando `textContent`. Esta propiedad no nos va a devolver ninguna de las etiquetas, solo devuelve el texto.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
  <style>
    .ocultar {
      display: none;
    }
  </style>
</head>
<body>
  <h1 id="uno">Hola <span class="ocultar">a todos!</span>mundo</h1>
  <h1 id="dos">Hola <span class="ocultar">a todos!</span>mundo</h1>
  <h1 id="tres">Hola <span class="ocultar">a todos!</span>mundo</h1>
</body>
<script>
  var elemDos = document.getElementById('dos');
  console.log(elemDos.textContent);
</script>
</html>
```

En caso de usar esta propiedad para cambiar el texto, se cambiará todo el contenido, incluyendo las etiquetas que tuviera como nodos hijos el elemento.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
  <style>
    .ocultar {
      display: none;
    }
  </style>
</head>
<body>
  <h1 id="uno">Hola <span class="ocultar">a todos!</span>mundo</h1>
  <h1 id="dos">Hola <span class="ocultar">a todos!</span>mundo</h1>
  <h1 id="tres">Hola <span class="ocultar">a todos!</span>mundo</h1>
</body>
<script>
  var elemDos = document.getElementById('dos');
  console.log(elemDos.textContent);
  elemDos.textContent = 'ooo';
</script>
</html>
```

4.2.1. Ejercicio 2

- Dado el mismo archivo *index.html* de antes, obtener el texto usando la propiedad `innerText` y ver las diferencias.

4.3. innerText

Actua como la propiedad anterior, aunque esta al no estar en ningún estandar, los navegadores no tienen porque tenerla implementada, y puede llegar a darse el caso en que falle la aplicación. Además de que ignora el texto que se encuentra en las etiquetas que se han ocultado mediante *CSS* (`display:none`) o alguna propiedad de *HTML* (`hidden`). Esta propiedad deberiamos de evitar usarla.

5. Añadir/Eliminar contenido HTML

Hemos visto como modificar los elementos que ya existen en el DOM, y como acceder a su contenido. Ahora vamos a ver como se pueden añadir nuevos elementos mediante JavaScript, y como eliminar los que ya están en el DOM. Para realizar esto tenemos dos formas de hacerlo, una es usando la propiedad `innerHTML` y la otra es usando los métodos de manipulación del DOM.

5.1. innerHTML

Esta propiedad se puede usar en cualquier nodo elemento. Puede ser usada tanto para obtener contenido, como para modificarlo y eliminarlo. Para actualizar el elemento, podemos asignarle a esta propiedad un string (que puede contener lenguaje de marcado para añadir elementos hijos) con el contenido. En cuanto a eliminar un elemento, lo que hay que hacer es igualar la propiedad a un string vacío.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
</head>
<body>
</body>
<script>
  var body = document.body;
  var contenido = '<ul id="mascotas">' +
                              '<li id="perro">Un Perro</li>' +
                              '<li id="gato">Un Gato</li>' +
                              '<li id="canario">Un Canario</li>' +
                              '</ul>';
  body.innerHTML = contenido;
</script>
</html>
```

Este método es mejor usarlo cuando vamos a cambiar fragmentos enteros de código.

5.1.1. Ejercicio 3

- Cambiar el tercer elemento de la lista para que aparezca *'Un periquito'*.
- 'periquito' tiene que aparecer en *negrita*.
- Eliminar el segundo elemento *li*.

6. Métodos de manipulación del DOM

Con estos métodos que vamos a ver, es más fácil acceder a nodos individuales para modificarlos o eliminarlos. Estos métodos son más seguros de usar que el `innerHTML` (ya que este puede dar lugar a más fallos si escribimos algo mal), aunque requiren usar mucho más código.

Con estos métodos vamos a poder crear nodos de texto, nodos de elementos, eliminarlos, añadir un nodo dentro de otro para ir creando el árbol del DOM...

Añadir elementos al DOM requiere seguir los siguientes tres pasos:

- Crear un elemento (nodo elemento)
- Crear el contenido (nodo texto)
- Añadirlo al DOM

Los métodos que vamos a usar para seguir esos tres pasos son los siguientes:

6.1. createElement()

Este método se encarga de crear un **nodo elemento**. Cuando se ha creado el nodo, todavía no se encuentra en el DOM, hasta que nosotros lo añadamos. Este método recibe como parámetro el *nombre de la etiqueta* que queremos crear.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Document</title>
</head>
<body>
</body>
<script>
  var elemH1 = document.createElement('h1');
</script>
</html>
```

6.2. createTextNode()

Este método se encarga de crear un **nodo texto**. Recibe como parámetro el *texto* que queremos mostrar dentro del elemento. En caso de querer crear un noco elemento vacío, nos podemos saltar

este paso.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Document</title>
</head>
<body>
</body>
<script>
  var elemH1 = document.createElement('h1');
  var textH1 = document.createTextNode('Un título');
</script>
</html>
```

Una vez que ya se ha creado el nodo de texto, se lo tenemos que añadir al elemento que lo va a mostrar, y para eso usaremos el método que vamos a ver a continuación.

6.3. appendChild()

Este es el método que se encarga de añadir un elemento, como elemento hijo de otro. Este elemento que queremos añadir se le pasa como parámetro.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Document</title>
</head>
<body>
</body>
<script>
  var elemH1 = document.createElement('h1');
  var textH1 = document.createTextNode('Un título');
  elemH1.appendChild(textH1);
  var body = document.body;
  body.appendChild(elemH1);
</script>
</html>
```

6.4. insertBefore()

Este método añade el elemento que se le pasa como primer parámetro justo antes del elemento que se le pasa como segundo parámetro. El encargado de llamar a este método es el elemento que contiene a los otros dos (elemento padre).

6.5. replaceChild()

Este método reemplaza el elemento que se le pasa como primer parámetro por el elemento que se le pasa como segundo parámetro. El encargado de llamar a este método es el elemento que contiene a los otros dos (elemento padre).

6.6. Ejercicio 4

- Modifica la lista que aparece a continuación, para que los elementos li muestren los números del 1 al 5

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <ul id="lista">
    <li>1</li>
    <li>3</li>
    <li>6</li>
    <li>5</li>
  </ul>
</body>
</html>
```

Para eliminar un elemento del DOM vamos a seguir los siguientes pasos:

- Obtenemos el elemento que queremos eliminar.
- Obtenemos el elemento padre del elemento a eliminar.
- Eliminamos el elemento hijo del elemento padre.

El método para eliminar los elementos es el siguiente:

6.7. removeChild()

Este método se encarga de eliminar el elemento hijo que se le pasa como parámetro, del elemento padre que es el que se encarga de llamar al método. Cuando eliminamos un elemento, también se eliminan todos los elementos hijos de este.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <ul id="lista">
    <li id="perro">Perro</li>
    <li id="gato">Gato</li>
    <li id="canario">Canario</li>
  </ul>
</body>
<script>
  var lista = document.getElementById('lista');
  var gato = document.getElementById('gato');
  lista.removeChild(gato);
</script>
</html>
```

7. Nodos de atributos

Los nodos de elementos también pueden contener nodos de atributos a los cuales podemos acceder para obtener, modificar o eliminar alguna de las propiedades que tiene el elemento. En este caso tenemos que seguir los siguientes pasos:

- Obtener el nodo del elemento que contiene el nodo atributo
- Realizar alguna acción sobre el nodo atributo

Los métodos que vamos a usar para realizar las acciones antes mencionadas son:

7.1. getAttribute()

Con este método vamos a poder obtener el valor del atributo que se le pasa como parámetro.

7.2. setAttribute()

Con este método vamos a poder cambiar el valor del atributo. Tanto el atributo al que se le va a cambiar el valor, como el valor nuevo, se tienen que pasar como parámetros. En caso de que el atributo que se va a cambiar no exista, se crea con el valor que se le está asignando.

7.3. hasAttribute()

Este método comprueba si el nodo del elemento tiene el atributo que se le ha pasado como parámetro.

7.4. removeAttribute()

Este método nos permite eliminar el atributo que se le pasa como parámetro del nodo del elemento.

Además de estos métodos, también podemos acceder a los atributos a través de propiedades como las que se muestran a continuación:

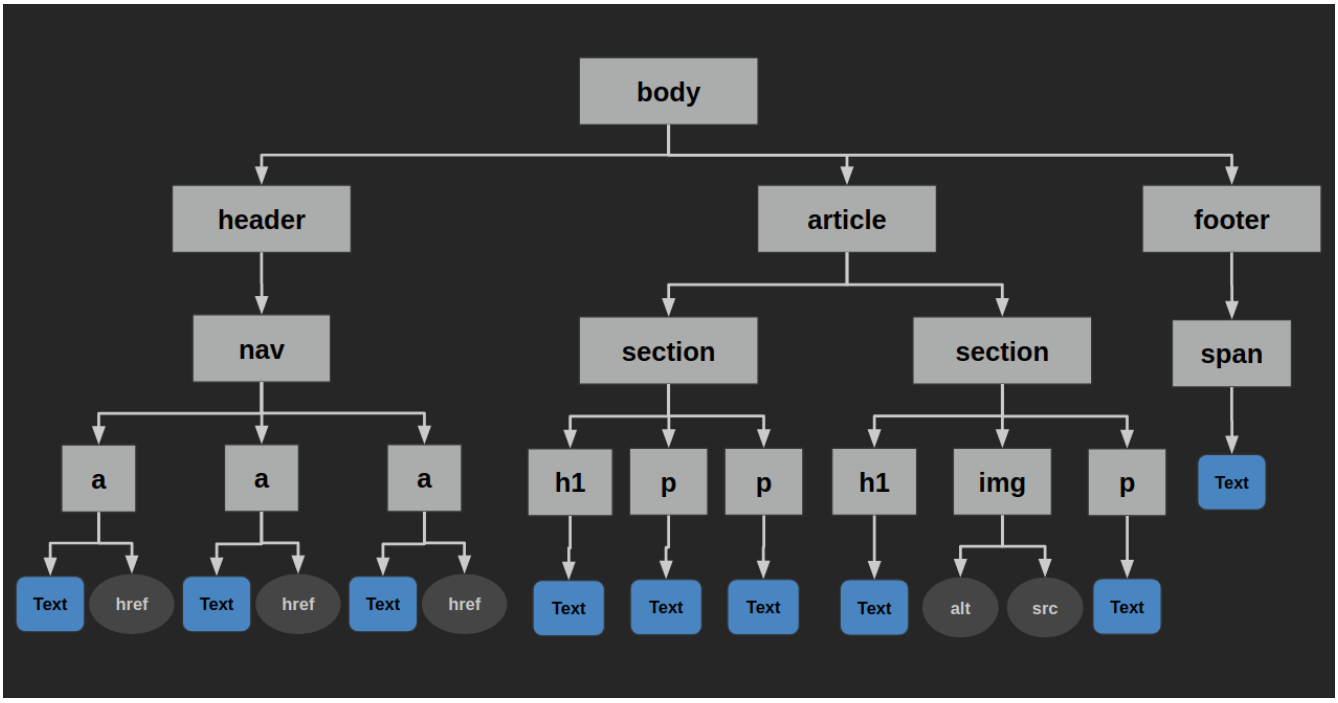
Table 2. Propiedades

Propiedad	**Descripción**
attributes	Devuelve un objeto con todos los atributos del elemento
className	Devuelve o asigna el valor al atributo *class* del elemento
id	Devuelve o asigna el valor al atributo *id* del elemento
href	Devuelve o asigna el valor al atributo *href* del elemento
src	Devuelve o asigna el valor al atributo *src* del elemento
checked	Devuelve o asigna el valor al atributo *checked* del elemento
title	Devuelve o asigna el valor al atributo *title* del elemento
type	Devuelve o asigna el valor al atributo *type* del elemento

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <style>
    .subrayado {
      text-decoration: underline;
    }
  </style>
</head>
<body>
  <h1 id="titulo">Título subrayado</h1>
  <input type="text" disabled id="btn-submit">
  <p id="parrafo">Un párrafo subrayado</p>
</body>
<script>
  var h1Elem = document.getElementById('titulo');
  console.log('El elemento h1' + (h1Elem.hasAttribute('class') ? '' : ' no') + ' tiene
un atributo clase');
  h1Elem.className = 'subrayado';
  console.log('El elemento h1' + (h1Elem.hasAttribute('class') ? '' : ' no') + ' tiene
un atributo clase');

  var pElem = document.getElementById('parrafo');
  pElem.setAttribute('class', 'subrayado');

  var inputElem =  document.getElementById('btn-submit');
  console.log('Type del input: ' + inputElem.getAttribute('type'));
  inputElem.setAttribute('type', 'submit');
  console.log('Type del input: ' + inputElem.getAttribute('type'));

  console.log('Atributos del input: ', inputElem.attributes);
  inputElem.removeAttribute('disabled');
</script>
</html>
```

7.5. Ejercicio 5

- Dado el árbol del DOM que aparece en la imagen, construyalo usando código JavaScript



7.6. Ejercicio 6

- Hacer un *carrousel* de imagenes
- La página HTML tiene que tener una seria de imagenes, y dos botones
- Al pulsar un botón se tiene que mostrar la imagen anterior, y al pulsar el otro se tiene que mostrar la imagen siguiente