TEMA

Tema 4. Gestión del DOM

Desarrollo de aplicaciones web

Desarrollo web en entorno cliente

Autor: Cristian Catalán

Linkia FP

Tema 4: Gestión del DOM

¿Qué aprenderás?

- Reconocer el modelo de objetos de un documento web.
- Modificar la página web añadiendo estilos a los elementos HTML.
- Crear y eliminar elementos del código HTML.
- Navegar a través de los elementos del código HTML.

¿Sabías que…?

- Mozilla Firefox retorna los colores CSS obtenidos desde JavaScript en RGB en vez de hexadecimal.
- En el navegador Internet Explorer definir una variable o función global con un nombre igual al id de un elemento HTML provocaba un conflicto.
- Cada tipo de elemento HTML tiene sus propios métodos y propiedades.
- Apache Cordova permite crear aplicaciones móviles con JavaScript y HTML.

4.1. El modelo de objetos del documento

En el desarrollo de aplicaciones web, una de las acciones más comunes que querremos haces es manipular la estructura HTML de alguna forma. Ya sea creando nuevas marcas, modificando sus estilos o eliminando marcas para que el usuario las deje de ver. Todas éstas acciones las realizaremos a través del Document Object Model, es decir, el DOM.

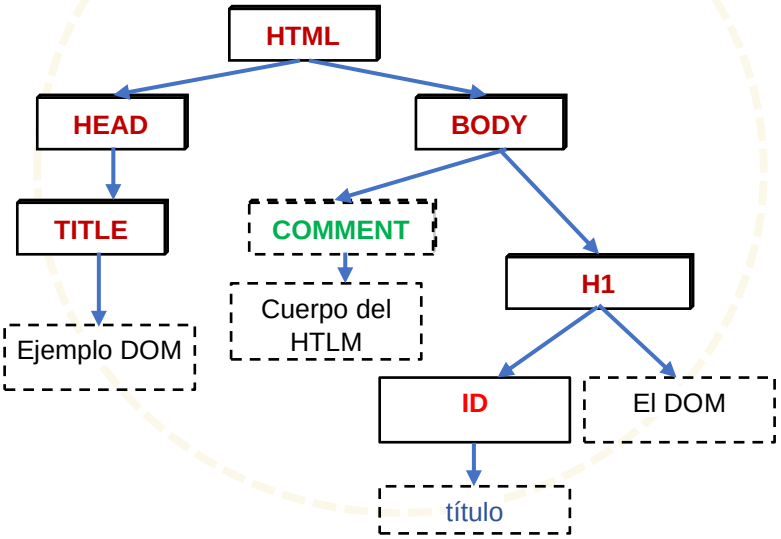
El DOM es una representación del documento HTML en el que se relacionan todos los nodos que forman el documento según su descendencia en una estructura de árbol en la que el elemento inicial es la marca HTML. Éste árbol es utilizado por el propio navegador para renderizar el contenido y aplicar estilos y puede ser utilizado por los lenguajes de programación para interpretar el documento HTML.

Los nodos que forman un documento HTML se pueden dividir en 5 categorías principales:

- **Document**: es el nodo raíz y contiene todos los nodos.
- **Los nodos de tipo Comment o comentarios**: representan los comentarios del HTML.
- **Los nodos de tipo Element o elementos**: representan las marcas que forman el HTML. Estos son los nodos con los que trabajaremos principalmente.
- **Los nodos de tipo Text o textos**: representan los textos contenidos en las marcas.
- **Los nodos de tipo Attribute o atributos**: son los atributos de las marcas.

En el siguiente ejemplo podemos ver un documento HTML y su representación del DOM.

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo DOM</title>
</head>
<body>
    <!-- Cuerpo del HTML-->
    <h1 id="título"> El DOM</h1>
</body>
</html>
```



Como podemos ver en el gráfico anterior cada nodo tiene como mínimo un padre (excepto el nodo raíz). A demás en el gráfico podemos observar que los nodos de tipo texto no pueden tener hijos ni hermanos y que un nodo solo puede tener un único padre.

El DOM además de estructurar y clasificar los nodos que forman el documento HTML nos ofrece una API (un conjunto de métodos y atributos) para poder interaccionar con los nodos.

Es importante destacar que el DOM está íntimamente relacionado con el código HTML de tal forma que si el código sufre un cambio también lo sufrirá el DOM, e igualmente si el DOM sufre algún cambió éste se verá reflejado en el HTML y por lo tanto en la representación de éste. Por ejemplo, si cambiamos un estilo en el DOM automáticamente se cambiará en la visualización del HTML.

Para poder interaccionar con un nodo lo primero que tenemos que conocer es la forma de acceder a él. A continuación vamos a ver distintas formas de acceder a uno o varios nodos.

4.2. Acceso a los nodos desde document

Como “document” representa el nodo raíz del DOM, será a partir de él que podremos acceder a los distintos nodos que lo forman. Cuando accedamos a un nodo lo que obtendremos será un objeto que representará ese nodo y que estará vinculado a ése nodo. Si realizamos alguna modificación a las propiedades del objeto, el nodo se actualizará automáticamente reflejando los cambios. E igualmente si el nodo sufre de una modificación, el objeto que le haga referencia también será modificado.

El objeto global “document” contiene un conjunto de propiedades que hacen referencia al “body” , “head”, todos los links, imágenes o formularios. Accediendo a estas propiedades es una de las formas más directas que ofrece “document” para acceder a sus nodos. A continuación se muestran las propiedades principales de “document”:

propiedad	Descripción
document.body;	Retorna un objeto del que hará referencia a nodo BODY
document.head;	Retorna un objeto que hará referencia al nodo HEAD
document.links;	Retorna un array que contendrá, por cada uno de los links de la página web, un objeto representativo.
document.images;	Retorna un array que contendrá, por cada una de las imágenes de la página web, un objeto representativo.
document.forms;	Retorna un array que contendrá, por cada uno de los formularios de la página web, un objeto representativo.

Estas formas de acceder a los elementos son muy generales y en ocasiones no nos permita acceder al nodo que deseamos. Por suerte “document” también nos ofrece formas de acceso más específicas como por ejemplo acceder a un elemento a través de su atributo “id”.

4.2.1. Acceso a un nodo por id

Es muy importante recordar que en HTML el atributo “id” ha de ser único dentro de un mismo documento. Podemos acceder rápidamente a un nodo a través de su atributo “id” con el método *“getElementById()”* de “*document”* pasando por parámetro el *“id”* del *“element”* al que queremos acceder. Su sintaxis sería:

document.getElementById(“idElemento”);

Este método nos retornará un objeto que será una referencia al elemento con el "id" indicado. En el siguiente código de ejemplo obtenemos un objeto de referencia al body, un array con referencia a los links y un objeto referenciado al elemento con id "titulo":

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo DOM</title>
    <script src="dom.js" defer></script>
</head>
<body>
    <h1 id="titulo"> El DOM</h1>
    <a href="#">Link1</a>
    <a href="#">Link2</a>
</body>
</html>
```

```
//accedemos al body
let nodoBody = document.body;

//accedemos a los links
let nodosLink = document.links;

//accedemos al elemento con id
"titulo"
let nodoH1 =
document.getElementById("titulo"
);
```

ATENCION! Es muy importante destacar en éste código el uso del atributo ***defer*** que vimos al inicio del módulo. Recordar que éste atributo obliga a ejecutar el JavaScript después de haber cargado todo el documento HTML. Si no lo ponemos no podrá acceder a ningún elemento del HTML, ya que el JavaScript está situado antes del Body.

4.2.2. Acceso a múltiples nodos

Dentro del objeto "document" podemos encontrar también otros métodos para acceder a múltiples nodos del tipo "element" según marca, clase o el "name". Al seleccionar un conjunto de nodos obtendremos un objeto del tipo "NodeList" o "HTMLCollection". Ambos tipos de datos los podemos tratar como arrays formados por el conjunto de nodos que seleccionemos pudiéndolo recorrer con un FOR o con un forEach. Los principales métodos para obtener un conjunto de elementos según la marca, clase o "*name*" son:

método	Descripción
document.getElementsByTagName("a");	Retorna un NodeList formado por todos los "element" que sean del tipo indicado por parámetro. En éste caso todas las marcas <a>.
document.getElementsByName("email");	Retorna un NodeList formado por todos los "element" que sean con el atributo "name" pasado por parámetro. En este caso todos los "element" con un atributo "name" igual a "email".
document.getElementsByClassName("rojo");	Retorna un NodeList formado por todos los "element" que sean de la clase indicada por parámetro. En éste caso todos los "element" donde una de sus clases sea "rojo".

A continuación vemos un ejemplo de utilización de los métodos vistos en la tabla anterior:

```
<head>
   <title>Ejemplo DOM</title>

   <script src="dom.js" defer></script>
</head>
<body>
   <intpu name="email"   class="rojo"/>
   <a href="#" class="rojo"> Link1  </a>
   <a href="#">Link2</a>
</body>
```

```
let links = document.getElementsByTagName("a");
let inputs = document.getElementsByName("email");
let rojos = document.getElementsByClassName("rojo");

console.log(links.length); // muestra un 2
let link1 = links[0]; //accedemos al 1r link
let link2=links[1]; //accedemos al 2o link

console.log(inputs.length); // muestra un 1
let email =inputs[0]; //accedemos al input email

console.log(rojos.length);// muestra un 2
email=rojos[0]; //accedemos al input email
link1=rojos[1]; //accedemos al 1r link
```

4.2.3. Acceso mediante selectores

También podemos acceder a un elemento utilizando los selectores CSS gracias al método de document "*querySelector*()". Este método retorna el primer elemento que satisface el selector CSS empezando desde el nodo raíz.

Asimismo todos los nodos "element" tienen también éste método. Al ejecutarse retornaran el primer elemento que satisfazca el selector CSS buscando dentro del nodo actual.

Si en vez de obtener un solo nodo como respuesta queremos obtener todos los nodos que satisfazcan el selector CSS, entonces utilizaremos el método "querySelectorAll()" , tanto desde document como desde un objeto referenciado a un nodo "element". Éste método nos retornará una lista de nodos, pero cuidado porque no se actualizará si se modifica el DOM.

En el siguiente ejemplo se muestra el acceso a nodos mediante selectores. Para ilustarlo mejor se ha añadid el HTML antes y después de ejecutarse el JavaScript para mostrar claramente los efectos de JavaScript sobre el HTML.

HTML antes de ejecutar el JS

```
<body>
  <h1 id="titulo"> El DOM </h1>
  Contenido
  <a href="#" >Link1</a>
  <a href="#" >Link2</a>
</body>
```

BODY después de ejecutar el JS

```
<body
    style="background-color: orange;">
  <h1 id="titulo"> El DOM </h1>
  Contenido
  <a href="#"
        style="text-decoration:none;
          color: green;">
        Link1</a>
  <a href="#" style="color: green;">
    Link2</a>
</body>
```

Documento JavaScript dom.js

```
//seleccionamos la primera marca de nombre body
let nodoBody = document.querySelector("body");
//cambiamos su color de fondo
nodoBody.style.backgroundColor="orange";

//seleccionamos el elemento con id "titulo"
let h1 = document.querySelector("#titulo");
//cambiamos su contenido
h1.style.innerHTML="Ejemplo selectores";

//seleccionamos el segundo Link hijo de body
let segundoLink =
      document.querySelector("body>a:nth-child(2)");
//le quitamos el subrayado
segundoLink.style.textDecoration="none";

//seleccionamos todos los links dentro de body
let links=nodoBody.querySelectorAll("a");
// recorremos los links y los ponemos en verde
links.forEach(function (unLink, key) {
  unLink.style.color="green";
});
```

4.2.4. Propiedades de un nodo

Cada vez que obtenemos una referencia a un nodo podemos interactuar con sus propiedades para modificar sus estilos.

general, cualquier objeto que haga referencia a un nodo tendrá como mínimo una propiedad que se corresponda con todos los atributos naturales que puede tener el nodo como elemento HTML. Es decir, si accedemos a un nodo “A”, podremos acceder y modificar su atributo “href”, si accedemos a un nodo “INPUT”, podremos acceder y modificar su atributo “value”.

A continuación veremos las principales propiedades comunes a las que podemos acceder y/o modificar cuando accedamos a un nodo (exceptuando las de parentesco que veremos en el siguiente punto):

propiedad	Modificable	Descripción
nodeName	no	Si el nodo es un elemento retorna el nombre de la marca. Si el nodo es un atributo retorna el nombre del atributo. Para otros casos retorna distintos nombres según el tipo.
nodeType	no	Si el nodo es un elemento retorna un 1 Si el nodo es un atributo retorna un 2 Si el nodo es un texto retorna un 3 Si el nodo es un comentario retorna un 8
tagName	No	Solo disponible para nodos de tipo Element, retorna el nombre de la marca.
innerHTML	Si	Retorna el contenido HTML del nodo. Establece el contenido HTML del nodo y es interpretado de nuevo por el navegador.

innerText	Si	Retorna el contenido del nodo indicado (sin mostrar los < >) y todos sus descendientes excepto los ocultos, <script> y <style>. Si se modifica, se sustituirá todo el contenido del nodo por el nuevo valor indicado sin interpretarlo como contenido HTML.
textContent	Si	Retorna el contenido del nodo indicado (sin mostrar los < >) y todos sus descendientes . Si se modifica, se sustituirá todo el contenido del nodo por el nuevo valor indicado sin interpretarlo como contenido HTML.
id	Si	Establece o retorna el id que tiene el nodo.
classList	Si	Establece o retorna la lista de clases que contiene el nodo tipo element.
className	Si	Establece o retorna una string con el contenido del atributo “class”
title	si	Establece o retorna el título de un nodo tipo element.
style	Si	Establece o retorna las propiedades CSS aplicadas a un nodo tipo element. No permite modificar la hoja de estilos, sino que las modificaciones se realizan siempre sobre el atributo *style* de la marca.

4.2.4.1. Estilos de un nodo

A través de la propiedad Style podremos acceder y modificar los estilos de una marca. En general si queremos acceder a un estilo solo deberemos escribir:

.style.nombrePropiedadCSS

Teniendo presente que todas aquellas propiedades CSS que tengan guiones, en nomenclatura JavaScript se elimina el guion y se pone en mayúsculas la segunda palabra. Así por ejemplo la propiedad CSS *backgroud-color* pasa a ser *backgroundColor*.

A continuación se muestra un ejemplo de acceso y modificación de las distintas propiedades de un nodo y sus estilos.

HTML antes de ejecutar el JS

```
<html><head>
    <title>Ejemplo DOM</title>
    <script src="dom.js" defer>
    </script>
</head><body>
  <h1 id="titulo"> El DOM</h1>
    Contenido
  <a href="#" id="link">Lin</a>
</body></html>
```

BODY después de ejecutar el JS

```
<body
style="background-color:orange;
text-align: center;">
  <h1 style="font-weight:bold;
height:100px;" id="titulo">
    <ol>
    <li id="item">Link1</li>
    <li>Item2</li>
    </ol>
    </h1>
    Contenido
    <a href="#" id="link"
 style="visibility: hidden;">
      Link1</a>
</body>
```

Document JavaScript dom.js

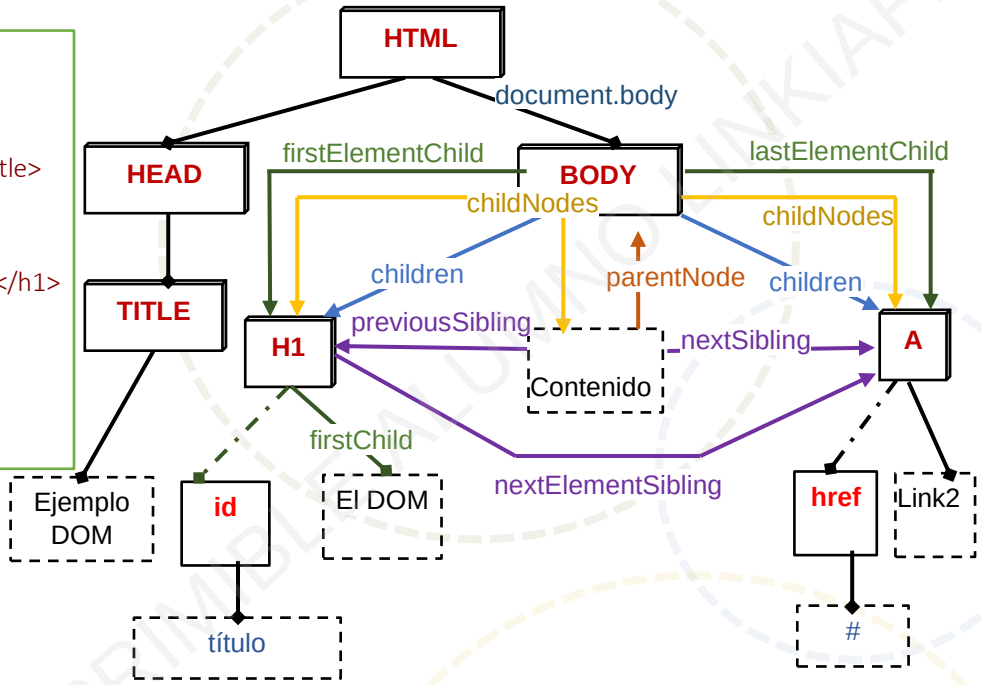
```
let nodoBody = document.body;
//accedemos al elemento con id "titulo"
let nodoH1=document.getElementById("titulo");
//accedemos al elemento con id "link"
let link = document.getElementById("link");

//ocultamo el link
link.style.visibility="hidden";
//el color de fondo de body a naranja
nodoBody.style.backgroundColor="orange";
//centramos el texto en body
nodoBody.style.textAlign="center";
//el texto de H1 en negrita
nodoH1.style.fontWeight="bold";
//la altura del a 100 pixeles
nodoH1.style.height="100px";

//canvamos el contenido de h1 por una lista
nodoH1.innerHTML=`<ol>
                    <li id="item">Item1</li>
                    <li>Item2</li>
                  </ol>`;
//dentro el primer ítem el contenido del link
document.getElementById("item")
              .innerHTML = link.innerHTML;
```

4.2.5. Recorrer los elementos del DOM por parentesco

A partir de una referencia a un nodo podemos , además de modificar sus estilos y atributos, navegar por el árbol del DOM utilizando sus propiedades. Veremos a continuación las propiedades que nos permiten obtener una referencia al resto de nodos que tengan un parentesco directo, es decir: padre, hijos y hermanos.

En el esquema del DOM del siguiente código HTML se muestran las principales propiedades que tiene cada nodo para recorrer el DOM:

```
<!DOCTYPE html>
<html>
<head>
   <title>Ejemplo DOM</title>
</head>
<body>
  <h1 id="titulo"> El DOM</h1>
          Contenido
   <a href="#">Link1</a>
</body>
</html>
```



Si nos fijamos en el gráfico veremos algunas particularidades. La diferencia entre "firstChild" y "firstElementChild" reside en que con la segunda forma estamos especificando que queremos acceder a un elemento que sea del tipo "element", es decir, que sea una marca. Esto es muy importante porque si nos fijamos en el código HTML, estrictamente el primer hijo que tiene Body es el salto de línea que hay entre las marcas <body> y <h1>, y en tal caso el firstChild de bodu será ése salto de línea. En el gráfico hemos obviado ésta posibilidad por claridad, pero no hay que olvidarla.

La propiedad "childNodes" retorna un array con todos los hijos, pero en el caso de "children" retorna un array solo con los hijos que sean "element".

También es importante remarcar que los atributos no son considerados hijos de un nodo, sino que todos los nodos de tipo "element" tienen un atributo "atributtes" que contiene un mapa con todos los atributos. No es realmente una propiedad que vayamos a utilizar mucho porque veremos formas más simples de acceder a un atributo.

A continuación se especifican las propiedades vistas en el gráfico anterior. El ejemplo de uso se basará en el HTML anterior:

propiedad	Uso	Descripción
firstElementChild	let b = document.body; let h1 = b.**firstElementChild**;	Retorna el primer hijo del tipo "element".
lastElementChild	let b = document.body; let link = b.**lastElementChild**;	Retorna el último hijo del tipo "element".

firstChild	let b = document.body;
	let h1 = b.firstElementChild;
	let elDom = h1.**firstChild**;
		Retorna el primer hijo.

nextSibling	let b = document.body;
	let h1 = b.firstElementChild;
	let contenido= h1.**nextSibling**;
		Retorna su siguiente hermano.

previousSibling	let b = document.body;
	let link = b.lastElementChild;
	
	let contenido = link.**previousSibling**;
		Retorna su anterior hermano.

nextElementSibling	let b = document.body;
	let h1 = b.firstElementChild;
	let link = h1.**nextElementSibling**;
		Retorna su siguiente hermano del tipo "element".

parentNode	let b = document.body;
	let h1 = b.firstElementChild;
	b=h1.**parentNode**;
		Retorna su padre.

childNodes	let b = document.body; let nodosHijos = b.childNodes;	Retorna un array con todos los nodos hijos.
children	let b = document.body; let marcasHijas = b.children;	Retorna un array con todos los nodos hijos que sean “element”.

En el siguiente video podemos ver cómo navegar a través del DOM utilizando el parentesco y modificar los estilos y propiedades de los elementos HTML.

Video: Navegar por el DOM y modificar estilos

4.3. Crear y añadir nodos

Utilizando el API del DOM podemos crear nuevos nodos y vincularlos con otros nodos.

4.3.1. Crear un nodo de tipo elemento

Para crear un nuevo nodo de tipo "element" utilizaremos el método "createElement()" que nos ofrece "document". La sintaxis será la siguiente:

document.createElement("TipoDeElemento");

Esta sentencia generará un nuevo "element" del tipo pasado por parámetro (DIV, SPAN, H1, etc..) y retornará una referencia a él. Pero no añadirá el "element" creado dentro de nuestro HTML. Esto lo tendremos que hacer a continuación.

Para añadir un "element" dentro de otro, primero tenemos que acceder al padre. A partir del padre utilizar su método "*apendChild()*" pasando por parámetro el objeto con la referencia al "element" que queremos sea su hijo. La sintaxis será la siguiente:

elementPadre.appendChild(elementHijo);

En el siguiente ejemplo creamos un H1 y P, modificamos los estilos y los añadimos:

HTML antes de ejecutar el JS

```
<head>
  <script src="dom.js" defer></script>
</head>
<body>     <div id="nuevos"></div>
</body>
```

BODY después de ejecutar el JS

```
<body>
  <div id="nuevos">
    <h1 style="color: blue;">Nuevo H1</h1>
    <p> texto del párrafo</p>
</div>
</body>
```

Documento JavaScript dom.js

```
  //creamos el nuevo elemento
let elemH1 = document.createElement("H1");
  //accedemos al futuro padre
let nuevos= document.getElementById("nuevos");
  //le añadimos el nuevo elemento
nuevos.appendChild(elemH1);
  //seguimos modificando sus estilos
elemH1.innerHTML="Nuevo H1";
elemH1.style.color="blue";

  //creamos otro elemento
let elemP =document.createElement("P");
elemP.innerHTML=" texto del párrafo";
  //y lo añadimos
nuevos.appendChild(elemP);
```

A partir del ejemplo anterior podemos observar cómo podemos modificar indistintamente las propiedades de un objeto antes o después de añadirlo al HTML, aunque la recomendación sería establecer todas sus propiedades ates para evitar comportamientos extraños en su visualización.

También podemos observar como al añadir un elemento con "*appendChild()*" este nuevo elemento es añadido como último elemento dentro de su padre. Lo podemos ver claramente si observamos como el nodo P se ha añadido después de H1. Si quisiéramos añadir un elemento ántes que otro deberíamos utilizar la función "*insertBefore(x,y)*" . Para utilizar esta función necesitamos saber 3 cosas: el nodo padre, el nuevo nodo y el nodo existente delante del que queramos insertar el nuevo nodo. Su sintaxis es:

nodoPadre.insertBefore(nuevoNodo, nodoExistente);

Cabe destacar que si "*nuevoNodo*" ya está situado en algún sitio del DOM será movido hasta su nueva posición delante de "*nodoExistente*".

En el siguiente ejemplo creamos un LI y lo añadimos entre los dos existentes:

HTML antes de ejecutar el JS

```
<head>
  <script src="dom.js" defer></script>
</head>
<body>
  <ol id="lista">
    <li>Item1</li>
    <li id="item">Item2</li>
  </ol>
</body>
```

Documento JavaScript dom.js

```
//creamos el nuevo elemento
let newLI = document.createElement("LI");
newLI.innerHTML="Nuevo";

//accedemos al futuro padre
let ol = document.getElementById("lista");
//accedemos al futuro hermano
let li = document.getElementById("item");

//añadimos el nuevoitem delante su hermano
ol.insertBefore(newLI,li);
```

BODY después de ejecutar el JS

```
<body>
  <ol id="lista">
    <li>Item1</li>
    <li>Nuevo</li>
    <li id="item">Item2</li>
  </ol>
</body>
```

Como resumen los métodos principales para crear e insertar un nodo son:

método	Uso	Descripción
createElement	let n = document.createElement("H1");	Crea un nuevo "element" del tipo indicado y lo retorna. No lo añade al HTML.
appendChild	nodoPadre.appendChild(nodoNuevo);	Añade como hijo del nodo padre el nuevo nodo. Si el nuevo nodo ya existe entonces lo mueve.
insertBefore	padre.insertBefore(nuevo, hermano);	Inserta el nuevo nodo dentro del nodo padre y delante del nodo hermano. Si el nuevo nodo ya existe lo mueve de sitio.

4.4. Eliminar nodos

Cuando queramos eliminar un nodo siempre necesitaremos tener una referencia del elemento que queremos eliminar y de su padre. A partir del padre llamaremos a su método *removeChild()* pasándole como parámetro la referencia al nodo que queramos eliminar. Su sintaxis es:

nodoPadre.removeChild(nodoHijoAEliminar);

También podemos eliminar un elemento sustituyéndolo por otro. En éste caso necesitaremos también la referencia al nuevo nodo que va a sustituir el antiguo y a partir del elemento padre llamaremos a su método *replaceChild()* pasando por parámetro el nuevo nodo y el nodo que queremos eliminar para ser reemplazado por el nuevo. Su sintaxis es:

nodoPadre.replaceChild(nuevoNodo, nodoHijoAEliminar);

Si el *nuevoNodo* está situado en otro punto del HTML, entonces será movido.

En el siguiente ejemplo eliminamos el Item2 y substituimos el Item4 por otro LI con el texto "Nuevo Item":

HTML antes de ejecutar el JS

```
<head>
   <script src="dom.js" defer></script>
</head>
<body>
   <ol >
      <li>Item1</li>
      <li id="item2">Item2</li>
      <li>Item3</li>
      <li id="item4">Item4</li>
   </ol>
</body>
```

Documento JavaScript dom.js

```
//accedemos al elemento a eliminar
let li2 = document.getElementById("item2");

//Accedemos a su padre
let padre = li2.parentNode;

//borramos el elemento del padre
padre.removeChild(li2);

//accedemos al elemento a eliminar
let li4 = document.getElementById("item4");
let newLi = document.createElement("LI");
newLi.innerHTML="Nuevo Item";

//sustituimos li4 por newLi
padre.replaceChild( newLi , li4 );
```

BODY después de ejecutar el JS

```
<body>
 <ol >
      <li>Item1</li>
      <li>Item3</li>
      <li >Nuevo Item</li>
  </ol>
</body>
```

Los métodos principales para eliminar un nodo son:

método	Uso	Descripción
removeChild	padre.removeChild(nodoAEliminar);	Elimina el *nodoAEliminar* del nodo *padre*.
replaceChild	padre. replaceChild (nuevo, antiguo);	Elimina el nodo *antiguo* del nodo *padre* y lo sustituye por el nodo *nuevo.*

4.5. Gestionar atributos propios

Todos los nodos HTML pueden tener atributos. Muchos de ellos son atributos “nativos”, es decir, ya especificados en el estándar HTML. Estos atributos no los deberemos crear ni eliminar nosotros. Incluso si en el código HTML hay un elemento que no tiene un atributo “nativo”, el DOM sí que lo tendrá especificado y podremos acceder a él a través de su propiedad relacionada. Por ejemplo, si queremos modificar el atributo id de un elemento (aunque inicialmente este no lo tenga) solo deberemos asignar a la propiedad id del objeto que referencie el elemento un nuevo valor.

Pero en muchas ocasiones nos puede resultar útil crear atributos propios que no existan en la especificación de HTML. Será solo en ésta ocasión cuando utilizaremos los siguientes métodos para crear, modificar o eliminar nuestro atributo.

El DOM nos ofrece muchas formas para leer los atributos propios de un nodo “element”. Y también muchas formas para modificar sus valores y crear nuevos atributos. A continuación veremos las formas más prácticas para tales efectos.

Cuando queramos leer o modificar un atributo siempre necesitaremos tener una referencia del elemento que lo tenga o lo vaya a tener.

Para leer el valor de un atributo llamaremos al método *getAttribute()* del nodo que lo contiene pasándole como parámetro el nombre del atributo que queremos leer utilizando la sintaxis:

nodoElement.getAttribute(“nombreAtributo”);

Tambien podemos utilizar el método *setAttribute()* para modificar el contenido de un atributo o crear un nuevo atributo si este ya no existía. La sintaxis utilizada es:

nodoElement.setAttribute(“nombreAtributo”,”valor del atributo”);

Hemos de tener presente que se han de cumplir las reglas HTML al establecer el nombre de un atributo. En el siguiente ejemplo accedemos a un atributo "contar", mostramos y alteamos su valor y finalmente creamos un nuevo atributo "restar" con valor "2":

HTML antes de ejecutar el JS

```
<head>
   <script src="dom.js" defer></script>
</head>
<body>
   <p id="p" contar="4">
      el atributo contar no existe en la especificación HTML  </p>
</body>
```

BODY después de ejecutar el JS

```
<body>
    <p id="p" contar="5" restar="2">      el atributo contar no existe en la especificación HTML   </p>
</body>
```

Documento JavaScript dom.js

```
let li2 = document.getElementById("p");

//obtenemos el valor del atributo contar
let valorContar = li2.getAttribute("contar");
console.log(valorContar); //4

//modifiamos el valor de contar
li2.setAttribute("contar",5);
valorContar = li2.getAttribute("contar");
console.log(valorContar); //5

//añadimos un nuevo atributo restar="2"
li2.setAttribute("restar",2);
valorContar = li2.getAttribute("restar");
console.log(valorContar); //2
```

Los métodos principales para gestionar los atributos son:

método	Uso	Descripción
getAttribute	nodo.getAttribute (nombreAtributo);	Retorna el valor del atributo *nombreAtributo* contenido en el *nodo.*
setAttribute	nodo.setAttribute (nombreAt, valor);	Crea un atributo de nombre *nombreAt* dentro del *nodo*. Si ya existe modifica su valor.
removeAttribute	nodo.removeAttribute (nombreAt);	Elimina un atributo de nombre *nombreAt* del *nodo*.

Recursos y enlaces

- DOM Enlightenment: versión online y gratuita de un libro centrado en el estudio del DOM. Escrito por Cody Lindley http://domenlightenment.com

- Documentación sobre el DOM en MDN: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

- X3Dom : librería JavaScript para insertar contenido 3D utilizando HTM, CSS y JS. https://www.x3dom.org

Conceptos clave

- **DOM** (Document Object Model): es el resultado de estructurar en objetos el contenido de un documento HTML.
- Modificando el DOM con JavaScript podemos modificar el documento HTML.
- **Nodo HTML:** un nodo HTML es cualquier texto, marca, atributo o comentario que podamos encontrar en el código HTML.
- **Elemento HTML**: es el conjunto de una etiqueta inicial, su contenido y la etiqueta final.

Test de autoevaluación

¿Qué tipo de nodos pueden tener hijos?

a) Todos
b) Element
c) Ninguno
d) Text

¿Cuál es la sentencia recomendada para modificar el atributo “id” de un elemento con “id” “pelota”?.

e) document.getElementByd(“pelota”).id=”nuevo valor”;
f) document.getElementByd(“pelota”).setAttribute (“id”, “nuevo valor”);
g) document.getElementsByTagName(“pelota”).id=”nuevo valor”;
h) document.getElementsByTagName(“pelota”).setAttribute (“id”, “nuevo valor”);

¿Cómo accedemos al elemento anterior del elemento con “id” “pelota”?

i) document.getElementByd(“pelota”).previousElementSibling();
j) document.getElementByd(“pelota”).previousSibling();
k) document.getElementByd(“pelota”).previousSibling;
l) document.getElementByd(“pelota”).previousElementSibling;

Ponlo en práctica

Actividad 1

Crea un documento HTML con una lista ordenada de 7 *items*. Programa que:

1. al clicar en el primero se cambie su color e texto a verde
2. al clicar en el segundo se cambie su color de fondo a amarillo
3. al clicar en el tercero cambie el último elemento LI por el primero
4. al clicar en el cuarto se cambie el color de fondo de todos a azul
5. al clicar en el quinto, se borre a él.
6. al clicar en el sexto, se borre el *element* hermano anterior
7. al clicar en el séptimo, se añada otro LI con texto "NUEVO LI" y color de fondo rojo

SOLUCIONARIOS

Test de autoevaluación

¿Qué tipo de nodos pueden tener hijos?

m) Todos

n) Element

o) Ninguno

p) Text

¿Cuál es la sentencia recomendada para modificar el atributo “id” de un elemento con “id” “pelota”?.

q) document.getElementByd(“pelota”).id=”nuevo valor”;

r) document.getElementByd(“pelota”).setAttribute (“id”, “nuevo valor”);

s) document.getElementsByTagName(“pelota”).id=”nuevo valor”;

t) document.getElementsByTagName(“pelota”).setAttribute (“id”, “nuevo valor”);

¿Cómo accedemos al elemento anterior del elemento con “id” “pelota”?

u) document.getElementByd(“pelota”).previousElementSibling();

v) document.getElementByd(“pelota”).previousSibling();

w) document.getElementByd(“pelota”).previousSibling;

x) **document.getElementByd(“pelota”).previousElementSibling;**

Ponlo en práctica

Actividad 1

Crea un documento HTML con una lista ordenada de 7 *items*. Programa que:

1. al clicar en el primero se cambie su color e texto a verde
2. al clicar en el segundo se cambie su color de fondo a amarillo
3. al clicar en el tercero cambie el último elemento LI por el primero
4. al clicar en el cuarto se cambie el color de fondo de todos a azul
5. al clicar en el quinto, se borre a él.
6. al clicar en el sexto, se borre el *element* hermano anterior
7. al clicar en el séptimo, se añada otro LI con texto "NUEVO LI" y color de fondo rojo

Los solucionarios están disponibles en la versión interactiva del aula.