Desarrollo Web en Entorno Cliente

UD. Document Object Model (DOM)

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

📖 **Importante**

! Atención

💬 **Interesante**

ÍNDICE DE CONTENIDO

UD. DOCUMENT OBJECT MODEL (DOM)

1. INTRODUCCIÓN

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

A pesar de sus orígenes, DOM se ha convertido en una utilidad disponible para la mayoría de lenguajes de programación (Java, PHP, JavaScript) y cuyas únicas diferencias se encuentran en la forma de implementarlo.

El llamado DOM (Document Object Model) es un modelo que permite tratar un documento Web XHTML como si fuera XML, navegando por los nodos existentes que forman la página, pudiendo manipular sus atributos e incluso crear nuevos elementos.

Para más información general de DOM:

- https://es.wikipedia.org/wiki/Document_Object_Model
- http://www.w3schools.com/js/js_htmldom.asp
- https://www.tutorialesprogramacionya.com/herramientas/javascript/domjs/

Usando Javascript para navegar en el DOM podemos acceder a todos los elementos XHTML de una página. Esto nos permite cambiar dinámicamente el aspecto de nuestras páginas Web.

En este tema vamos a estudiar las principales funciones de Javascript para modificar el DOM.

2. MODIFICANDO ATRIBUTOS

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.).

Todas estas tareas habituales son muy sencillas de realizar gracias a DOM. Sin embargo, 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. El motivo por el que se muestra el funcionamiento de esta transformación interna es que condiciona el comportamiento de DOM y por tanto, la forma en la que se manipulan las páginas.

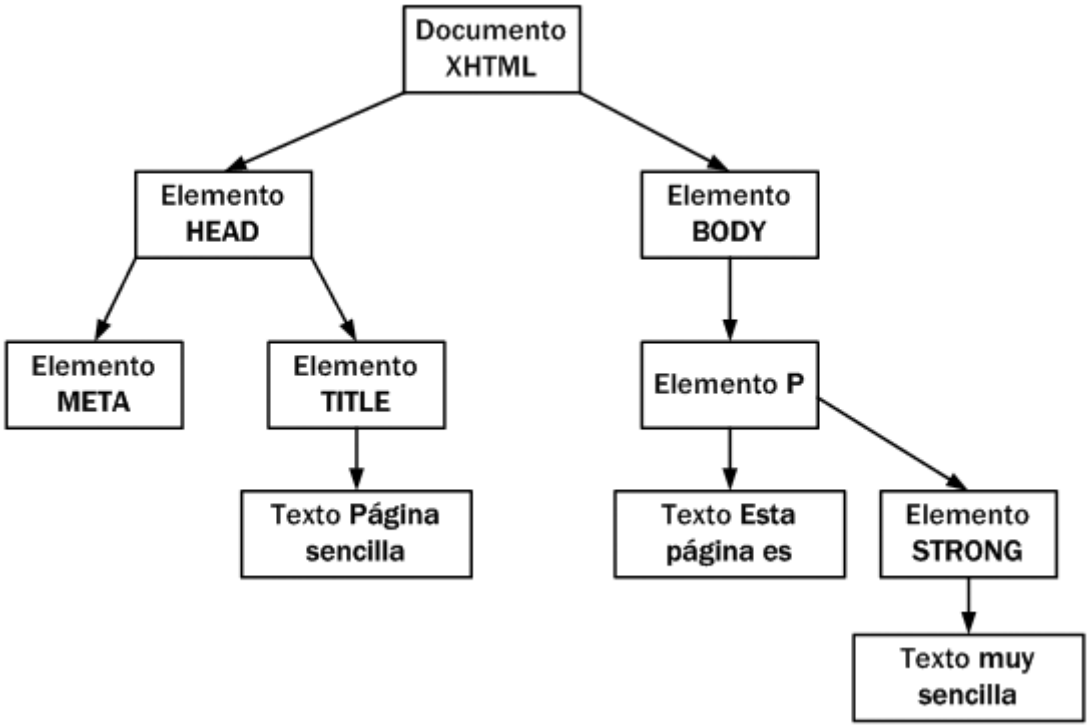
DOM transforma todos los documentos XHTML en un conjunto de elementos llamados nodos, que están interconectados y que representan los contenidos de las páginas web y las relaciones entre ellos. Por su aspecto, la unión de todos los nodos se llama "árbol de nodos".

La siguiente página XHTML sencilla:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
/>
    <title>Página sencilla</title>
</head>

<body>
    <p>Esta página es <strong>muy sencilla</strong></p>
</body>
</html>
```

Se transforma en el siguiente árbol de nodos:



En el esquema anterior, cada rectángulo representa un nodo DOM y las flechas indican las relaciones entre nodos. Dentro de cada nodo, se ha incluido su tipo (que se verá más adelante) y su contenido.

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

A partir de ese nodo raíz, cada etiqueta XHTML se transforma en un nodo de tipo "Elemento". La conversión de etiquetas en nodos se realiza de forma jerárquica. De esta forma, del nodo raíz solamente pueden derivar los nodos HEAD y BODY. A partir de esta derivación inicial, cada etiqueta XHTML se transforma en un nodo que deriva del nodo correspondiente a su "etiqueta padre".

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

Así, la siguiente etiqueta XHTML:

```
<title>Página sencilla</title>
```

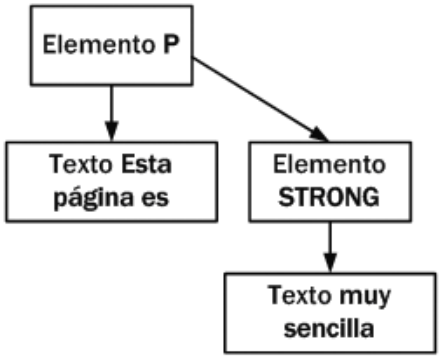
Genera los siguientes dos nodos:

De la misma forma, la siguiente etiqueta XHTML:

```
<p>Esta página es <strong>muy sencilla</strong></p>
```

Genera los siguientes nodos:

- Nodo de tipo "Elemento" correspondiente a la etiqueta <p>.
- Nodo de tipo "Texto" con el contenido textual de la etiqueta <p>.
- Como el contenido de <p> incluye en su interior otra etiqueta XHTML, la etiqueta interior se transforma en un nodo de tipo "Elemento" que representa la etiqueta y que deriva del nodo anterior.
- El contenido de la etiqueta genera a su vez otro nodo de tipo "Texto" que deriva del nodo generado por .



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

- Las etiquetas XHTML 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 XHTML se encuentra dentro de otra, se sigue el mismo procedimiento anterior, pero los nodos generados serán nodos hijo de su etiqueta padre.

Como se puede suponer, las páginas XHTML habituales producen árboles con miles de nodos. Aun así, el proceso de transformación es rápido y automático, siendo las funciones proporcionadas por DOM (que se verán más adelante) las únicas que permiten acceder a cualquier nodo de la página de forma sencilla e inmediata.

3. TIPOS DE NODOS

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

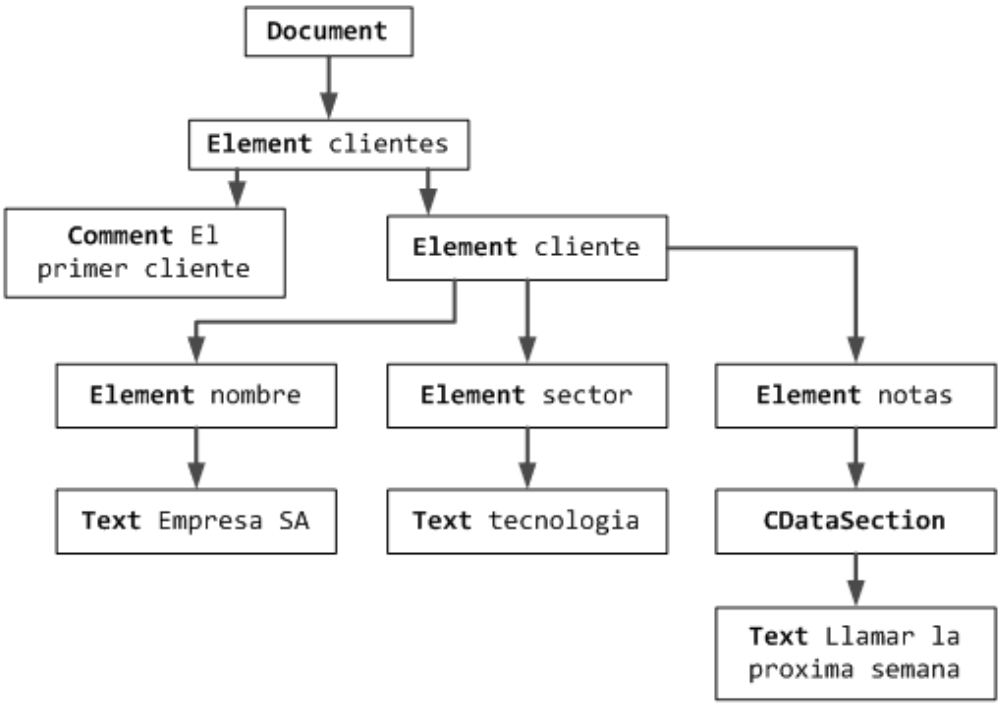
- **Document**, nodo raíz del que derivan todos los demás nodos del árbol.
- **DocumentType**, es el nodo que contiene la representación del DTD empleado en la página (indicado mediante el DOCTYPE)..
- **Element**, representa cada una de las etiquetas XHTML. Se trata del único nodo que puede contener 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 XHTML, es decir, uno por cada par atributo=valor.
- **Text**, nodo que contiene el texto encerrado por una etiqueta XHTML.
- **CdataSection**, es el nodo que representa una sección de tipo <![CDATA[ ]]>.
- **Comment**, representa los comentarios incluidos en la página XHTML.

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

El siguiente ejemplo de documento sencillo de XML muestra algunos de los nodos más habituales:

```
<?xml version="1.0"?>
<clientes>
 <!-- El primer cliente -->
 <cliente>
  <nombre>Empresa SA</nombre>
  <sector>Tecnologia</sector>
  <notas><![CDATA[
  Llamar la proxima semana
  ]]></notas>
 </cliente>
</clientes>
```

Su representación como árbol de nodos DOM es la siguiente:

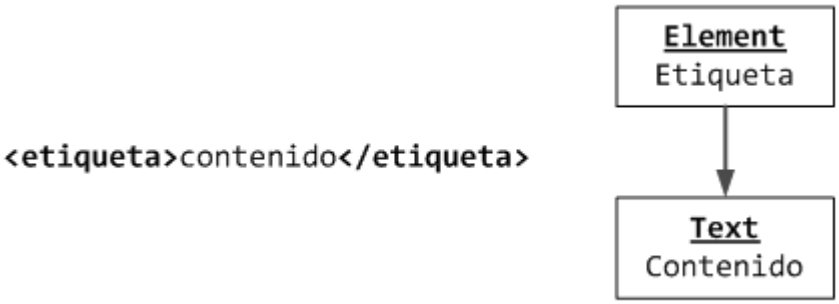


El nodo raíz siempre es el nodo de tipo Document, del que derivan todos los demás nodos del documento. Este nodo es común para todas las páginas HTML y todos los documentos XML. A continuación se incluye la etiqueta <clientes>...</clientes>. Como se trata de una etiqueta, DOM la transforma en un nodo de tipo Element. Además, como la etiqueta encierra a todos los demás elementos de la página, el nodo Clientes de tipo Element deriva directamente de Document y todos los demás nodos del documento derivan de ese nodo.

El comentario es el primer texto que se incluye dentro de la etiqueta <clientes>, por lo que se transforma en el primer subnodo del nodo clientes. Al ser un comentario de XML, se trata de un nodo de tipo Comment.

Al mismo nivel que el comentario, se encuentra la etiqueta <cliente> que define las características del primer cliente y forma el segundo subnodo del nodo clientes. Todas las demás etiquetas del documento XML se encuentran encerradas por la etiqueta <cliente>...</cliente>, por lo que todos los nodos restantes derivarán del nodo cliente.

Cada etiqueta simple de tipo <etiqueta>texto</etiqueta> se transforma en un par de nodos: el primero de tipo Element (que contiene la etiqueta en sí) y el segundo, un nodo hijo de tipo Text que contiene el contenido definido entre la etiqueta de apertura y la de cierre.



La etiqueta <notas> se transforma en tres nodos, ya que contiene una sección de tipo CData, que a su vez se transforma en un nodo del que deriva el contenido propio de la sección CData.

Un buen método para comprobar la transformación que sufren las páginas web y visualizar la jerarquía de nodos creada por DOM es utilizar la utilidad "Inspector DOM" (o "DOM Inspector") del navegador Mozilla Firefox.

La utilidad se puede encontrar en el menú Herramientas y además de mostrar visualmente la jerarquía de nodos, permite acceder fácilmente a toda la información de cada nodo y muestra en la página web el contenido al que hace referencia el nodo actual.

4. PROPIEDADES DE NAVEGACIÓN ENTRE LOS NODOS DE TIPO ELEMENTO.

Cualquier nodo de tipo elemento dispone de las siguientes propiedades para obtener la referencia de:

- **firstElementChild**: Almacena la referencia del primer nodo de tipo elemento hijo.
- **parentElement**: Almacena la referencia del nodo elemento padre.
- **lastElementChild**: Almacena la referencia del último nodo hijo.
- **nextElementSibling**: Almacena la referencia del siguiente nodo de tipo elemento (si lo tiene)
- **previousElementSibling**: Almacena la referencia del nodo de tipo elemento previo (si lo tiene)
- **children:** Almacena una colección con todos los elementos tipo nodo hijo.

5. FUNCIONES DE JAVASCRIPT PARA LOCALIZAR ELEMENTOS EN EL DOM

Una vez construido automáticamente el árbol completo de nodos DOM, ya es posible utilizar las funciones DOM para acceder de forma directa a cualquier nodo del árbol. Como acceder a un nodo del árbol es equivalente a acceder a "un trozo" de la página, una vez construido el árbol, ya es posible manipular de forma sencilla la página: acceder al valor de un elemento, establecer el valor de un elemento, mover un elemento de la página, crear y añadir nuevos elementos, etc.

DOM proporciona dos métodos alternativos para acceder a un nodo específico: acceso a través de sus nodos padre y acceso directo.

Las funciones que proporciona DOM para acceder a un nodo a través de sus nodos padre consisten en acceder al nodo raíz de la página y después a sus nodos hijos y a los nodos hijos de esos hijos y así sucesivamente hasta el último nodo de la rama terminada por el nodo buscado. Sin embargo, cuando se quiere acceder a un nodo específico, es mucho más rápido acceder directamente a ese nodo y no llegar hasta él descendiendo a través de todos sus nodos padre.

Por ese motivo, no se van a presentar las funciones necesarias para el acceso jerárquico de nodos y se muestran solamente las que permiten acceder de forma directa a los nodos.

Por último, es importante 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 XHTML se cargue por completo. Más adelante se verá cómo asegurar que un código JavaScript solamente se ejecute cuando el navegador ha cargado entera la página XHTML.
Las funciones aquí estudiadas normalmente se usan sobre el elemento “document”, ya que así se aplican a todo el documento.

Aun así, pueden usarse en cualquier nodo XHTML, entonces la búsqueda se realizaría no en todo en el documento, sino en el sub-árbol formado por el elemento en sí y sus hijos.

5.1 getElementById(identificador)

Esta función devuelve un elemento DOM del subárbol cuyo identificador sea el indicado en la cadena “identificador”.

http://www.w3schools.com/jsref/met_document_getelementbyid.asp

Ejemplo:

```
let myDiv = document.getElementById("miDiv");
console.log("El html de miDiv es "+myDiv.innerHTML);
```

5.2 getElementsByTagName(etiqueta)

Esta función devuelve un array con todos los elementos DOM del subárbol cuya etiqueta XHTML sea la indicada en la cadena "etiqueta".

http://www.w3schools.com/jsref/met_document_getelementsbytagname.asp

Ejemplo:

```
let myDiv = document.getElementById("miDiv")
let losP = myDiv.getElementsByTagName("p");
let num = losP.length;
console.log("Hay " + num + " <p> elementos en el elemento miDiv");
console.log("En el primer P el HTML asociado es "+losP[0].innerHTML);
```

5.3 getElementsByName(nombre)

Esta función devuelve un array con todos los elementos DOM del subárbol cuyo atributo name sea el indicado en la cadena "nombre".

http://www.w3schools.com/jsref/met_doc_getelementsbyname.asp

Ejemplo:

```
let elementos = document.getElementsByName("name");
let i;
// Todos los textbox que tengan de name alumnos, los marcamos
for (i = 0; i < elementos.length; i++) {
     if (elementos[i].type === "checkbox") {
           elementos[i].checked = true;
     }
}
```

5.4 querySelector(identificadorElemento)

La función querySelector(identificadoElemento) acepta como parámetro un selector que identifica el elemento (o elementos) a seleccionar. En el caso de esta función, únicamente es devuelto el primer elemento que cumple la condición. Si no existe el elemento, el valor retornado es null.

Ejemplo:

```
var logo = document.querySelector(".enlace");
```

```
<div id="cabecera">
  <a href="/" class="enlace">...</a>
</div>
<div id="cuerpo">
    <p>Loren ipsum <a href="enlace">...</a></p>
</div>
```

En este caso, a pesar de existir varios elementos de la clase enlace, únicamente es seleccionado el primero de ellos.

5.5 querySelectorAll(identificadorElemento)

La función querySelectorAll(identificadorElemetno) acepta como parámetro un selector que identifica el elemento (o elementos) a seleccionar. Esta función devuelve un objeto de tipo NodeList con los elementos que coincidan con el selector.

Ejemplo:

```
var enlaces = document.querySelectorAll(".enlace");
```

```
<div id="cabecera">
  <a href="/" class="enlace">...</a>
</div>
<div id="cuerpo">
    <p>Loren ipsum <a href="enlace">...</a></p>
</div>
```

Para acceder a los elementos almacenados en NodeList, recorremos el objeto como si de un array se tratase.

Ejemplo:

```
for (var i=0; i<enlaces.length; i++) {
    var enlaces = enlaces[i];
}
```

6. FUNCIONES PARA CREAR/ELIMINAR NODOS

En esta parte veremos las funciones básicas para crear y eliminar nodos XHTML.

6.1 removeChild(nodo)

Esta función se aplica a un nodo padre. La función recibe un nodo hijo suyo y lo borra. Es útil usarlo con el atributo "parentnode", que devuelve el nodo padre del elemento que estamos manejando.

http://www.w3schools.com/jsref/met_node_removechild.asp

Ejemplo:

```
let parrafo=document.getElementById("miParrafo");
// Obtiene la referencia del padre, y al padre le aplica la función removeChild
parrafo.parentnode.removeChild(parrafo);
```

6.2 appendChild(nodo)

Esta función se aplica a un nodo padre. La función recibe un nodo y lo incluye como nodo hijo del padre. Se puede combinar con funciones como "createElement", que permiten crear elementos XHTML.

http://www.w3schools.com/jsref/met_node_appendchild.asp

Ejemplo:

```
// Creo un nodo de tipo LI
let nuevoNodo = document.createElement("LI");
// Al nodo LI le asocio un texto (también podría asociarse XHTML con innerHTML)
let nodoTexto = document.createTextNode("Agua");
nuevoNodo.appendChild(nodoTexto);
// A miLista, lista ya existente, le añado el elemento creado
document.getElementById("miLista").appendChild(nuevoNodo);
```

6.3 Métodos createElement, createTextNode y append

El método **createElement** del objeto 'document' crea un elemento HTML indicado en el primer parámetro. De forma similar el método **creatTextNodo** crea un nodo de tipo texto que posteriormente se lo enlaza a un nodo de tipo elemento.

Una vez creado un nodo de tipo elemento lo debemos agregar al DOM de nuestra página, el método más común para añadir un elemento a otro elemento del DOM es por medio del método **append**.

Veamos como añadimos tres elementos de tipo 'li' a un elemento de tipo 'ol' que ya tiene 2 items:

```
const item3 = document.createElement('li')
const texto3 = document.createTextNode('Item 3')
item3.appendChild(texto3)
const item4 = document.createElement('li')
const texto4 = document.createTextNode('Item 4')
item4.appendChild(texto4)
const item5 = document.createElement('li')
const texto5 = document.createTextNode('Item 5')
item5.appendChild(texto5)
const lista1 = document.querySelector("#lista1")
lista1.append(item3)
lista1.append(item4)
lista1.append(item5)
```

Creamos un elemento de tipo 'li' y otro de tipo texto:

```
const item3 = document.createElement('li')
const texto3 = document.createTextNode('Item 3')
```

Añadimos el nodo de tipo texto al elemento:

```
item3.appendChild(texto3)
```

Luego de hacer lo mismo para los otros dos elementos, procedemos a obtener la referencia del elemento 'ol' y mediante el método **'append'** agregamos al final de todos los nodos hijos cada uno de los nuevos elementos:

```
const lista1 = document.querySelector("#lista1")
lista1.append(item3)
lista1.append(item4)
lista1.append(item5)
```

Veamos como mostramos la tabla de multiplicar del 5 generada en forma dinámica:

```
const tabla = document.querySelector("#tabla")
for (let x = 1; x <= 10; x++) {
  const dato = `${x} * 5 = ${x * 5}`
  const elemento = document.createElement('p')
  const texto = document.createTextNode(dato)
  elemento.appendChild(texto)
  tabla.append(elemento)
}
```

Con el método **append** podemos añadir un hijo al final de cualquier elemento HTML de la página.

7. NODOS DE TIPO ELEMENTO: PROPIEDAD INNERHTML Y OUTERHTML

La propiedad **innerHTML** solo está presente en los nodos de tipo elemento y no en los nodos de tipo texto. Si accedemos a su valor nos retorna un **string** con todos los nodos de tipo texto y nodos elementos que contiene hacia abajo en el árbol de nodos de la página web.

Por ejemplo, si accedemos a la propiedad **innerHTML** del nodo elemento **'body'** luego recuperamos un **string** con todo el contenido del **body**:

```
alert(document.body.innerHTML)
```

Como vemos nos aparece todo el contenido comprendido entre las etiquetas **<body>** y **</body>** sin incluirlas, luego de probar la salida de la función **alert**, puede borrarla para evitar que se abra dicho diálogo cada vez que se modifique el programa.

Pero no solo podemos consultar el valor almacenado en la propiedad **innerHTML**, sino que podemos asignarle un **string** con un bloque de HTML y el navegador se encarga de modificar el árbol de nodos y recrearlo con la nueva estructura.

Veamos cómo podemos cambiar el contenido de una lista desordenada (**ul**) mediante la asignación de un **string** que contienen las nuevas etiquetas a mostrar dentro de la lista:

```
setTimeout(() => {
  const lista1 = document.querySelector("#lista1")
  lista1.innerHTML = `
    <li>one</li>
    <li>two</li>
    <li>three</li>
    <li>four</li>`
}, 3000)
```

Vea luego de 3 segundo como la página web (la ventana que se encuentra abaja) se modifica su contenido (no he cambiado la página HTML original de la derecha, tener en cuenta que la propiedad **innerHTML** modifica el **DOM** en memoria y no modifica el archivo HTML físicamente)

La propiedad **outerHTML** también solo está presente en los nodos de tipo elemento y no en los nodos de tipo texto. La diferencia fundamental de la propiedad **outerHTML** es que retorna un **string** con el nodo desde donde obtenemos la referencia, es decir lo mismo que **innerHTML** más el nodo que envuelve el contenido.

Por ejemplo:

```
const lista1 = document.querySelector("#lista1")
console.log(lista1.innerHTML)
```

Retorna:

```
<li>1</li>
<li>2</li>
<li>3</li>
<li>4</li>
```

Luego con **outerHTML:**

```
const lista1 = document.querySelector("#lista1")
console.log(lista1.outerHTML)
```

Retorna:

```
<ul id="lista1">
<li>1</li>
<li>2</li>
<li>3</li>
<li>4</li>
</ul>
```

Probemos el siguiente código con el **innerHTML**:

```
setTimeout(() => {
  const lista1 = document.querySelector("#lista1")
  lista1.innerHTML = `
    <li>one</li>
    <li>two</li>
    <li>three</li>
    <li>four</li>`
}, 2000)
```

8. NODOS DE TIPO ELEMENTO: PROPIEDAD INNERHTML Y OUTERHTML

Una vez que se ha accedido a un nodo, el siguiente paso natural consiste en acceder y/o modificar sus atributos y propiedades. Mediante DOM, es posible acceder de forma sencilla a todos los atributos XHTML y todas las propiedades CSS de cualquier elemento de la página.

Los atributos XHTML de los elementos de la página se transforman automáticamente en propiedades de los nodos. Para acceder a su valor, simplemente se indica el nombre del atributo XHTML detrás del nombre del nodo.

El siguiente ejemplo obtiene de forma directa la dirección a la que enlaza el enlace:

```
var enlace = document.getElementById("enlace");
console.log(enlace.href); // muestra http://www...com
```

```
<a id="enlace" href="http://www...com">Enlace</a>
```

En el ejemplo anterior, se obtiene el nodo DOM que representa el enlace mediante la función **document.getElementById().** A continuación, se obtiene el atributo **href** del enlace mediante **enlace.href**. Para obtener por ejemplo el atributo id, se utilizaría **enlace.id**.

Las propiedades CSS requieren un paso extra para acceder a ellas. Para obtener el valor de cualquier propiedad CSS del nodo, se debe utilizar el atributo **style**. El siguiente ejemplo obtiene el valor de la propiedad **margin** de la imagen:

```
var imagen = document.getElementById("imagen");
console.log(imagen.style.margin);
```

```
<img id="imagen" style="margin:0; border:0;" src="logo.png" />
```

Si el nombre de una propiedad CSS es compuesto, se accede a su valor modificando ligeramente su nombre:

```
var parrafo = document.getElementById("parrafo");
console.log(parrafo.style.fontWeight);  // muestra "bold"
```

```
<p id="parrafo" style="font-weight: bold;">...</p>
```

La transformación del nombre de las propiedades CSS compuestas consiste en eliminar todos los guiones medios (-) y escribir en mayúscula la letra siguiente a cada guion medio (lo que se conoce como nomenclatura lowerCamelCase). A continuación, se muestran algunos ejemplos:

- **font-weight** se transforma en **fontWeight**
- **line-height** se transforma en **lineHeight**
- **border-top-style** se transforma en **borderTopStyle**
- **list-style-image** se transforma en **listStyleImage**

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

```
var parrafo = document.getElementById("parrafo");
console.log(parrafo.class); // muestra "undefined"
console.log(parrafo.className); // muestra "normal"
```

```
<p id="parrafo" class="normal">...</p>
```

9. ATRIBUTOS HTML Y PROPIEDADES CSS EN DOM.

Los métodos presentados anteriormente para el acceso a los atributos de los elementos, son genéricos de XML. La versión de DOM específica para HTML incluye algunas propiedades y métodos aún más directos y sencillos para el acceso a los atributos de los elementos HTML y a sus propiedades CSS.

La principal ventaja del DOM para HTML es que todos los atributos de todos los elementos HTML se transforman en propiedades de los nodos. De esta forma, es posible acceder de forma directa a cualquier atributo de HTML. Si se considera el siguiente elemento <img> de HTML con sus tres atributos:

```
<img id="logo" src="logo.gif" border="0" />
```

Empleando los métodos tradicionales de DOM, se puede acceder y manipular cada atributo:

```
var laImagen = document.getElementById("logo");

// acceder a los atributos
var archivo = laImagen.getAttribute("src");
var borde = laImagen.getAttribute("border");

// modificar los atributos
laImagen.setAttribute("src", "nuevo_logo.gif");
laImagen.setAttribute("border", "1");
```

La ventaja de la especificación de DOM para HTML es que permite acceder y modificar todos los atributos de los elementos de forma directa:

```
var laImagen = document.getElementById("logo");

// acceder a los atributos
var archivo = laImagen.src;
var borde = laImagen.border;

// modificar los atributos
laImagen.src = "nuevo_logo.gif";
laImagen.border = "1";
```

Las ventajas de utilizar esta forma de acceder y modificar los atributos de los elementos es que el código resultante es más sencillo y conciso. Por otra parte, algunas versiones de Internet Explorer no implementan correctamente el método setAttribute(), lo que provoca que, en ocasiones, los cambios realizados no se reflejan en la página HTML.

La única excepción que existe en esta forma de obtener el valor de los atributos HTML es el atributo class. Como la palabra class está reservada por JavaScript para su uso futuro, no es posible utilizarla para acceder al valor del atributo class de HTML. La solución consiste en acceder a ese atributo mediante el nombre alternativo className:

```
<p id="parrafo" class="normal">...</p>
```

```
var parrafo = document.getElementById("parrafo");
alert(parrafo.class);     // muestra "undefined"
alert(parrafo.className); // muestra "normal"
```

El acceso a las propiedades CSS no es tan directo y sencillo como el acceso a los atributos HTML. En primer lugar, los estilos CSS se pueden aplicar de varias formas diferentes sobre un mismo elemento HTML. Si se establecen las propiedades CSS mediante el atributo style de HTML:

```
<p id="parrafo" style="color: #C00">...</p>
```

Para acceder al valor de una propiedad CSS, se obtiene la referencia del nodo, se accede a su propiedad style y a continuación se indica el nombre de la propiedad CSS cuyo valor se quiere obtener. Aunque parece lógico que en el ejemplo anterior el valor obtenido sea #C00, en realidad cada navegador obtiene el mismo valor de formas diferentes:

- Firefox y Safari muestran el valor rgb(204, 0, 0)
- Internet Explorer muestra el valor #c00
- Opera muestra el valor #cc0000

En el caso de las propiedades CSS con nombre compuesto (font-weight, border-top-style, list-style-type, etc.), para acceder a su valor es necesario modificar su nombre original eliminando los guiones medios y escribiendo en mayúsculas la primera letra de cada palabra que no sea la primera:

```
var parrafo = document.getElementById("parrafo");
var negrita = parrafo.style.fontWeight;
```

El nombre original de la propiedad CSS es font-weight. Para obtener su valor mediante JavaScript, se elimina el guión medio (fontweight) y se pasa a mayúsculas la primera letra de cada palabra que no sea la primera (fontWeight).

Si el nombre de la propiedad CSS está formado por tres palabras, se realiza la misma transformación. De esta forma, la propiedad border-top-style se accede en DOM mediante el nombre borderTopStyle.

Además de obtener el valor de las propiedades CSS, también es posible modificar su valor mediante JavaScript. Una vez obtenida la referencia del nodo, se puede modificar el valor de cualquier propiedad CSS accediendo a ella mediante la propiedad style:

```
<p id="parrafo">...</p>


var parrafo = document.getElementById("parrafo");
parrafo.style.margin = "10px";    // añade un margen de 10px al párrafo
parrafo.style.color = "#CCC";     // modifica el color de la letra del párrafo
```

Todos los ejemplos anteriores hacen uso de la propiedad style para acceder o establecer el valor de las propiedades CSS de los elementos. Sin embargo, esta propiedad sólo permite acceder al valor de las propiedades CSS establecidas directamente sobre el elemento HTML. En otras palabras, la propiedad style del nodo sólo contiene el valor de las propiedades CSS establecidas mediante el atributo style de HTML.

Por otra parte, los estilos CSS normalmente se aplican mediante reglas CSS incluidas en archivos externos. Si se utiliza la propiedad style de DOM para acceder al valor de una propiedad CSS establecida mediante una regla externa, el navegador no obtiene el valor correcto:

```
// Código HTML
<p id="parrafo">...</p>


// Regla CSS
#parrafo { color: #008000; }


// Código JavaScript
var parrafo = document.getElementById("parrafo");
var color = parrafo.style.color;  // color no almacena ningún valor
```

Para obtener el valor de las propiedades CSS independientemente de cómo se hayan aplicado, es necesario utilizar otras propiedades de JavaScript. Si se utiliza un navegador de la familia Internet Explorer, se hace uso de la propiedad currentStyle. Si se utiliza cualquier otro navegador, se puede emplear la función getComputedStyle().

```
// Código HTML
<p id="parrafo">...</p>

// Regla CSS
#parrafo { color: #008000; }

// Código JavaScript para Internet Explorer
var parrafo = document.getElementById("parrafo");
var color = parrafo.currentStyle['color'];

// Código JavaScript para otros navegadores
var parrafo = document.getElementById("parrafo");
var color = document.defaultView.getComputedStyle(parrafo, '').getPropertyValue('color');
```

La propiedad currentStyle requiere el nombre de las propiedades CSS según el formato de JavaScript (sin guiones medios), mientras que la función getPropertyValue() exige el uso del nombre original de la propiedad CSS. Por este motivo, la creación de aplicaciones compatibles con todos los navegadores se puede complicar en exceso.

A continuación, se muestra una función compatible con todos los navegadores, creada por el programador Robert Nyman y publicada en su blog personal:

```
function getStyle(elemento, propiedadCss) {
  var valor = "";
  if(document.defaultView && document.defaultView.getComputedStyle){
    valor = document.defaultView.getComputedStyle(elemento, '').getPropertyValue(propiedadCs
s);
  }
  else if(elemento.currentStyle) {
    propiedadCss = propiedadCss.replace(/\-(\w)/g, function (strMatch, p1) {
      return p1.toUpperCase();
    });
    valor = elemento.currentStyle[propiedadCss];
  }
  return valor;
}
```

Utilizando la función anterior, es posible rehacer el ejemplo para que funcione correctamente en cualquier navegador:

```
// Código HTML
<p id="parrafo">...</p>

// Regla CSS
#parrafo { color: #008000; }

// Código JavaScript para cualquier navegador
var parrafo = document.getElementById("parrafo");
var color = getStyle(parrafo, 'color');
```

10. TABLAS HTML EN DOM.

Las tablas son elementos muy comunes en las páginas HTML, por lo que DOM proporciona métodos específicos para trabajar con ellas. Si se utilizan los métodos tradicionales, crear una tabla es una tarea tediosa, por la gran cantidad de nodos de tipo elemento y de tipo texto que se deben crear.

Afortunadamente, la versión de DOM para HTML incluye varias propiedades y métodos para crear tablas, filas y columnas de forma sencilla.

Propiedades y métodos de <table>:

Propiedad/Método	**Descripción**
rows	Devuelve un array con las filas de la tabla
tBodies	Devuelve un array con todos los <tbody> de la tabla
insertRow(posicion)	Inserta una nueva fila en la posición indicada dentro del array de filas de la tabla
deleteRow(posicion)	Elimina la fila de la posición indicada

Propiedades y métodos de <tbody>:

Propiedad/Método	Descripción
rows	Devuelve un array con las filas del <tbody> seleccionado
insertRow(posicion)	Inserta una nueva fila en la posición indicada dentro del array de filas del <tbody>
deleteRow(posicion)	Elimina la fila de la posición indicada

Propiedades y métodos de <tr>:

Propiedad/Método	Descripción
cells	Devuelve un array con las columnas de la fila seleccionada
insertCell(posicion)	Inserta una nueva columna en la posición indicada dentro del array de columnas de la fila
deleteCell(posicion)	Elimina la columna de la posición indicada

Si se considera la siguiente tabla XHTML:

```
<table summary="Descripción de la tabla y su contenido">
<caption>Título de la tabla</caption>
<thead>
 <tr>
  <th scope="col"></th>
  <th scope="col">Cabecera columna 1</th>
  <th scope="col">Cabecera columna 2</th>
 </tr>
</thead>

<tfoot>
 <tr>
  <th scope="col"></th>
  <th scope="col">Cabecera columna 1</th>
  <th scope="col">Cabecera columna 2</th>
 </tr>
</tfoot>

<tbody>
```

```
 <tr>
  <th scope="row">Cabecera fila 1</th>
  <td>Celda 1 - 1</td>
  <td>Celda 1 - 2</td>
 </tr>
 <tr>
  <th scope="row">Cabecera fila 2</th>
  <td>Celda 2 - 1</td>
  <td>Celda 2 - 2</td>
 </tr>
</tbody>

</table>
```

A continuación, se muestran algunos ejemplos de manipulación de tablas XHTML mediante las propiedades y funciones específicas de DOM.

Obtener el número total de filas de la tabla:

```
var tabla = document.getElementById('miTabla');
var numFilas = tabla.rows.length;
```

Obtener el número total de cuerpos de la tabla (secciones <tbody>):

```
var tabla = document.getElementById('miTabla');
var numCuerpos = tabla.tBodies.length;
```

Obtener el número de filas del primer cuerpo (sección <tbody>) de la tabla:

```
var tabla = document.getElementById('miTabla');
var numFilasCuerpo = tabla.tBodies[0].rows.length;
```

Borrar la primera fila de la tabla y la primera fila del cuerpo (sección <tbody>):

```
var tabla = document.getElementById('miTabla');
tabla.deleteRow(0);
tabla.tBodies[0].deleteRow(0);
```

Borrar la primera columna de la primera fila del cuerpo (sección <tbody>):

```
var tabla = document.getElementById('miTabla');
tabla.tBodies[0].rows[0].deleteCell(0);
```

Obtener el número de columnas de la primera parte del cuerpo (sección <tbody>):

```
var tabla = document.getElementById('miTabla');
var numColumnas = tabla.rows[0].cells.length;
```

Obtener el texto de la primera columna de la primera fila del cuerpo (sección <tbody>):

```
var tabla = document.getElementById('miTabla');
var texto = tabla.tBodies[0].rows[0].cells[0].innerHTML;
```

Recorrer todas las filas y todas las columnas de la tabla:

```
var tabla = document.getElementById('miTabla');
var filas = tabla.rows;
for(var i=0; i<filas.length; i++) {
  var fila = filas[i];
  var columnas = fila.cells;
  for(var j=0; j<columnas.length; j++) {
    var columna = columnas[j];
    // ...
  }
}
```

Insertar una tercera fila al cuerpo (sección <tbody>) de la tabla:

```
var tabla = document.getElementById('miTabla');
// Insertar la tercera fila
tabla.tBodies[0].insertRow(2);

// Crear la columna de tipo <th> que hace de cabecera de fila
var cabecera = document.createElement("th");
cabecera.setAttribute('scope', 'row');
cabecera.innerHTML = 'Cabecera fila 3'
tabla.tBodies[0].rows[2].appendChild(cabecera);

// Crear las dos columnas de datos y añadirlas a la nueva fila
tabla.tBodies[0].rows[2].insertCell(1);
tabla.tBodies[0].rows[2].cells[1].innerHTML = 'Celda 3 - 1';
tabla.tBodies[0].rows[2].insertCell(2);
tabla.tBodies[0].rows[2].cells[2].innerHTML = 'Celda 3 - 2';
```

Por último, se muestra de forma resumida el código JavaScript necesario para crear la tabla XHTML del ejemplo anterior:

```
// Crear <table> y sus dos atributos
var tabla = document.createElement('table');
tabla.setAttribute('id', 'otraTabla');
tabla.setAttribute('summary', 'Descripción de la tabla y su contenido');

// Crear <caption> y añadirlo a la <table>
var caption = document.createElement('caption');
var titulo = document.createTextNode('Título de la tabla');
caption.appendChild(titulo);
tabla.appendChild(caption);

// Crear sección <thead>
var thead = document.createElement('thead');
tabla.appendChild(thead);

// Añadir una fila a la sección <thead>
thead.insertRow(0);

// Añadir las tres columnas de la fila de <thead>
var cabecera = document.createElement('th');
cabecera.innerHTML = '';
thead.rows[0].appendChild(cabecera);

cabecera = document.createElement('th');
cabecera.setAttribute('scope', 'col');
cabecera.innerHTML = 'Cabecera columna 1';
tabla.rows[0].appendChild(cabecera);

cabecera = document.createElement('th');
cabecera.setAttribute('scope', 'col');
cabecera.innerHTML = 'Cabecera columna 2';
tabla.rows[0].appendChild(cabecera);
```

```
// La sección <tfoot> se crearía de forma similar a <thead>

// Crear sección <tbody>
var tbody = document.createElement('tbody');
tabla.appendChild(tbody);

// Añadir una fila a la sección <tbody>
tbody.insertRow(0);

cabecera = document.createElement("th");
cabecera.setAttribute('scope', 'row');
cabecera.innerHTML = 'Cabecera fila 1'
tabla.tBodies[0].rows[0].appendChild(cabecera);

tbody.rows[0].insertCell(1);
tbody.rows[0].cells[1].innerHTML = 'Celda 1 - 1';
// También se podría hacer:
// tbody.rows[0].cells[0].appendChild(document.createTextNode('Celda 1 - 1'));

tbody.rows[0].insertCell(2);
tbody.rows[0].cells[2].innerHTML = 'Celda 1 - 2';

// El resto de filas del <tbody> se crearía de la misma forma

// Añadir la tabla creada al final de la página
document.body.appendChild(tabla);
```

11. BIBLIOGRAFÍA

[1] Javascript Mozilla Developer https://developer.mozilla.org/es/docs/Web/JavaScript
[2] Javascript ES6 W3C https://www.w3schools.com/js/js_es6.asp
[3] Referencia DOM
https://developer.mozilla.org/es/docs/Referencia_DOM_de_Gecko/Introducci%C3%B3n

12. AUTORES (EN ORDEN ALFABÉTICO)

A continuación, ofrecemos en orden alfabético el listado de autores que han hecho aportaciones a este documento:

- Sevillano Hernández, Juan
- García Barea, Sergio