MÓDULO: PROGRAMACIÓN AVANZADA EN JAVASCRIPT

Aprendizaje Esperado

Reconocer los elementos fundamentales del Domain Object Model y los mecanismos para la manipulación de elementos en un documento html.

EVENTOS Y MANIPULACIÓN DEL DOM

Introducción a DOM

Cuando se definió el lenguaje XML, surgió la necesidad de procesar y manipular el contenido de los archivos XML mediante los lenguajes de programación tradicionales. XML es un lenguaje sencillo de escribir pero complejo para procesar y manipular de forma eficiente. Por este motivo, surgieron algunas técnicas entre las que se encuentra DOM.

DOM o *Document Object Model* es un conjunto de utilidades específicamente diseñadas para manipular documentos XML. Por extensión, DOM también se puede utilizar para manipular documentos XHTML y HTML.

Técnicamente, DOM es una API de funciones que se pueden utilizar para manipular las páginas XHTML de forma rápida y eficiente.

Antes de poder utilizar sus funciones, DOM transforma internamente el archivo XML original en una estructura más fácil de manejar formada por una jerarquía de nodos. De esta forma, DOM transforma el código XML en una serie de nodos interconectados en forma de *árbol*.

El árbol generado no sólo representa los contenidos del archivo original (mediante los nodos del árbol) sino que también representa sus relaciones (mediante las ramas del árbol que conectan los nodos).

Aunque en ocasiones DOM se asocia con la programación web y con JavaScript, la API de DOM es independiente de cualquier lenguaje de programación.

De hecho, DOM está disponible en la mayoría de lenguajes de programación comúnmente empleados.

Si se considera la siguiente página HTML 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=iso-8859-1" />
    <title>Página sencilla</title>
  </head>

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

Todo cuerpo del archivo del DOM está dividido en elementos, los cuales son los que usa Javascript para interactuar y ante todo respetando la jerarquía que tenga cada uno en el árbol estructural del DOM, tomando en cuenta el ejemplo anterior de una página HTML podemos decir que sus elementos serían HEAD, TITLE, BODY, P.

Antes de poder utilizar las funciones de DOM, los navegadores convierten automáticamente la página HTML anterior en la siguiente estructura de árbol de nodos:

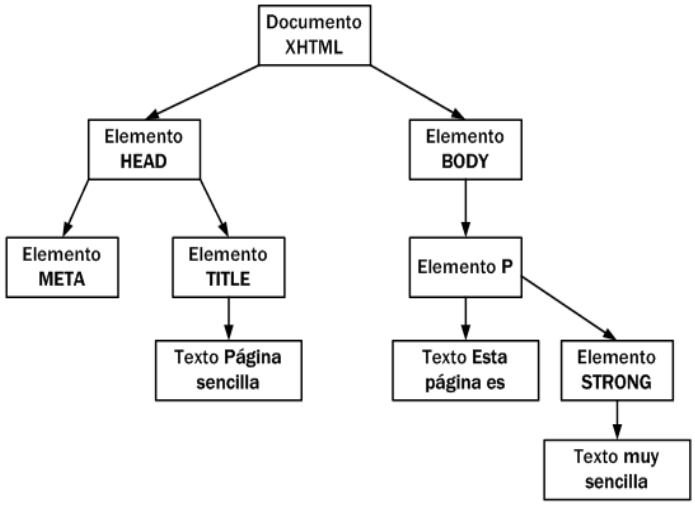

Figura de representación en forma de árbol de la página HTML de ejemplo

La página HTML se ha transformado en una jerarquía de nodos, en la que el nodo raíz es un nodo de tipo *documento HTML*. A partir de este nodo, existen dos nodos en el mismo nivel formados por las etiquetas <head> y <body>. De cada uno de los anteriores surge otro nodo (<title> y <p> respectivamente). Por último, de cada nodo anterior surge otro nodo de tipo *texto*.

Antes de poder utilizar la API de DOM, se construye de forma automática el árbol para poder ejecutar de forma eficiente todas esas funciones. De este modo, para utilizar DOM es imprescindible que la página web se haya cargado por completo, ya que de otro modo no existe el árbol de nodos y las funciones DOM no pueden funcionar correctamente.

La ventaja de emplear DOM es que permite a los programadores disponer de un control muy preciso sobre la estructura del documento HTML o XML que están manipulando. Las funciones que proporciona DOM permiten añadir, eliminar, modificar y reemplazar cualquier nodo de cualquier documento de forma sencilla.

La primera especificación de DOM se definió en 1998 y permitió homogeneizar la implementación del DHTML o *HTML dinámico* en los diferentes navegadores, ya que permitía modificar el contenido de las páginas web sin necesidad de recargar la página entera.
Selección de un elemento del DOM

entre las sintaxis que usa Javascript para llamar a los elementos tenemos document.body, document.getElementByClassName, document.tittle, document.querySelectorAll, document.getElementById, entre otros.

El objeto document

En Javascript, la forma de acceder al DOM es a través de un objeto llamado document, que representa el árbol DOM de la página o

pestaña del navegador donde nos encontramos. En su interior pueden existir varios tipos de elementos, pero principalmente serán Elemento o Node:

- El elemento no es más que la representación genérica de una etiqueta: HTMLElement.

- Node es una unidad más básica, la cuál puede ser Elemento o un nodo de texto.

Trabajar con elementos

Crear nuevos elementos

Para crear nuevos elementos (de etiqueta) que contenga el texto (nodo tipo text), usaremos los siguientes métodos (o funciones preestablecidas):

- createElement(): Crea un elemento. En el paréntesis, y con comillas, se escribe la etiqueta del elemento a crear.
- createTextNode(): Crea un elemento de tipo text. En el paréntesis escribimos el contenido del elemento de texto. Si lo escribimos literal debemos ponerlo entre comillas. También podemos haberlo guardado previamente en una variable, en ese caso escribiremos en el paréntesis la variable sin comillas.

- appendChild(): Hace que el elemento insertado dentro del paréntesis sea hijo del elemento que debemos escribir antes del método seguido de un punto. (elemenetoPadre.appendChild(elementoHijo)).

Aca tenemos ejemplos de cómo crear elementos mediante el método createElement().

En el ejemplo crearemos un nuevo elemento de etiqueta párrafo:

```
nuevoElemento= document.createElement("p");
```

Creamos luego el elemento de texto, mediante el método createTextNode:

texto = document.createTextNode("Este es el texto del nuevo elemento");

Por último hacemos que el elemento de texto sea hijo del elemento "***nuevoElemento***" mediante el método appendChild():

nuevo = nuevoElemento.appendChild(texto)

Con esto hemos creado un nuevo elemento; ahora debemos insertarlo en la página.

Insertar elementos

Hay dos formas de insertar un elemento en la página, La primera es mediante el método appendChild, lo cual lo inserta como elemento hijo del de referencia, pero al final de los demás elementos que tenga su elemento padre. Por ejemplo, si tenemos un "div" con un "id" llamado "lateral", el nuevo elemento aparecerá al final de los demás que ya tengamos:

padre = document.getElementById("lateral")

padre.appendChild(nuevo)

Donde padre es el elemento del que va a depender, y "*nuevo*" es la variable donde hemos guardado el nodo elemento después de añadirle el elemento texto.

La segunda forma de insertar un elemento se basa en el método insertBefore(). Este método inserta el nuevo elemento delante de la etiqueta que hayamos seleccionado. para ello seguiremos los siguientes pasos:

Elegimos la etiqueta delante de la cual queremos insertar la nueva que hemos creado, en el ejemplo, sería la que lleva el "id" al que llamamos "insertar".

var referencia = document.getElementbyId("insertar");

Buscamos el elemento padre del elemento de referencia mediante la propiedad parentNode; y lo guardamos en una variable:

var padre = referencia.parentNode;

Por último desde el elemento padre, utilizamos el método InsertBefore(), el cual dentro del paréntesis tiene dos parámetros separados por una coma, el primero es el nuevo elemento que insertamos, y el segundo es el elemento de referencia, delante del cual debe insertarse.

padre.insertBefore(nuevo,referencia);

Reemplazar elemento

Una vez creado un elemento no sólo podemos insertarlo en la página como elemento nuevo, sino que podemos insertarlo reemplazando a un elemento ya existente; para ello utilizaremos el método replaceChild(). La forma de reemplazarlo es similar a insertar un elemento delante de otro, lo único que cambia es el nombre del método:

Buscamos el elemento que queremos reemplazar:

var cambiarElemento = document.getElementbyId("reemplazar");

Accedemos a su elemento padre mediante la propiedad parentNode:

var padre = cambiarElemento.parentNode;

Por último aplicamos el método replaceChild() al elemento padre. Entre paréntesis pondremos en primer lugar el elemento nuevo que lo sustituirá, y en segundo el elemento viejo:

padre.replaceChild(nuevo,cambiarElemento)

Eliminar nodos

Para eliminar un elemento utilizaremos el método removeChild(). Dentro del paréntesis pondremos el elemento que queremos eliminar. Debemos eliminar el elemento desde su elemento padre por lo que buscamos primero su elemento padre para luego desde ahí eliminarlo como un hijo de éste: ejemplo

suprimir = document.getElementById("provisional")

suprimir.parentNode.removeChild(suprimir)

Al eliminar un elemento eliminamos también todos los elementos hijos que dependen de él, Aunque el elemento se elimina en la página, seguimos teniendo una variable que lo almacena.

Cambiar el contenido de un elemento

sí deseamos agregar o modificar el contenido de un elemento del DOM, podemos usar el comando **innerHTML**, para lograr esto primero debemos guardar en una variable el contenido nuevo que queremos que tenga el elemento, ejemplo:

var texto = "Nuevo texto para este elemento";

Después accedemos al elemento al que queremos cambiar su contenido. y lo guardamos en una variable.

var elemento = document.getElementById("nombre_id");

Por último damos al elemento el nuevo valor que hemos guardado, mediante la propiedad innerHTML:

elemento.innerHTML = texto

Ampliar el contenido de un elemento

Para ampliar el elemento añadiendo más contenido al ya existente, lo que haremos usar el mismo comando innerHTML pero de una forma que sume o concatena el contenido anterior con el nuevo:

var texto = "Añadir este texto al elemento"

var elemento = document.getElementById("nombre_id")

Hasta aquí el código empleado es el mismo que en el punto anterior, ahora en lugar de usar el signo igual (=), usaremos el operador de

asignación con suma (+=), con lo cual al contenido ya existente le añadiremos el contenido nuevo.

elemento.innerHTML += texto

Acceso a las propiedades CSS

Al igual que en lo anteriormente explicado, para modificar o agregar los parámetros de estilo a un elemento debemos primero colos el nombre de la etiqueta a la que le haremos referencia, Escribimos luego la palabra .style (precedida de un punto) y después el nombre de la propiedad (también precedida de un punto).

Los nombres de las propiedades varían ligeramente, Las propiedades que constan de más de una palabra (en CSS varias palabras separadas por guiones), pasan a escribirse sin guiones, y con la primera letra de la segunda palabra y siguientes en mayúscula. Así por ejemplo font-family pasa a llamarse fontFamily; border-right-color pasaría a ser borderRightColor, y así con las demás propiedades con más de una palabra.

En el siguiente ejemplo se accede a la propiedad text-align para ver qué alineación tiene determinado texto:

```
elemento = document.getElementById("elemento1")

 alineacion= elemento.style.textAlign
```

De esta manera guardariamos en la variable “alineacion” el valor de la propiedad text-align en ese elemento.

Pero si lo que queremos es modificar el valor de una propiedad, o añadir una propiedad nueva a un elemento, lo que debemos de hacer es simplemente asignarsela:

```
elemento = document.getElementById("cambiar");

elemento.style.textAlign = "center"
```

Para pasarle el nuevo valor debemos escribirlo entre comillas o pasarlo en una variable de texto que contenga su nombre, si el elemento tenía ya un valor para esa propiedad CSS, este se cambiará; si no tenía ninguno, al elemento se le añade la propiedad CSS con el valor que se le ha pasado.

El acceso a las propiedades CSS con javascript es una de las claves fundamentales para programar una página dinámica, ya que permite cambiar completamente el aspecto visual de la página. Cuando cambiamos las propiedades CSS, sobre todo si usamos eventos que interactúan con el usuario, y le permiten controlar los cambios, entre otras cosas podemos hacer:

- Cambiar el aspecto de los textos: tamaño, color, fuente, alineación, estilo, etc.
- Cambiar colores e imágenes de fondo.
- Cambiar bordes o márgenes de los elementos de la página.
- Cambiar el aspecto de las listas o tablas que aparecen en la página.
- Cambiar el tamaño de algunas cajas o cuadros, o imágenes, o cambiarlos de sitio (propiedades width, height, position).
- Hacer aparecer o desaparecer algunos elementos (propiedades display o visibility),

Eventos

Son las acciones o procesos que se disparan o accionan por el usuario a través del teclado o el ratón, para el que el JavaScript ejecute determinadas interacciones, entre los eventos tenemos: mover el ratón, hacer click o doble click, pulsar una tecla, entrar con el ratón en un elemento, seleccionarlo con el ratón o con la tecla TAB, etc. Incluso el hecho de acabar de cargarse la página es un evento.

El JavaScript puede detectar cuando se produce un determinado evento, y aplicarle un código para que al producirse el evento se ejecute ese código. De esta forma se interactúa con el usuario, el

cual al provocar el evento desencadena la ejecución del código Javascript asociado.

Entre los eventos más utilizados tenemos: onclick, onmouseover y onmouseout, pero hay más eventos que veremos a continuación.

Evento	**Descripción**	**Elementos que aplica**
onblur	Deseleccionar el elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
onclick	Pulsar y soltar el ratón	todos los elementos
ondbclick	Pulsar y soltar el ratón dos veces seguidas	todos los elementos
onfocus	Seleccionar un elemento (con el ratón o con el tabulador)	<button>, <input>, <label>, <select>, <textarea>, <body>
onkeydown	Pulsar una tecla (sin soltar)	elementos de formulario y <body>
onkeydown	Pulsar una tecla	elementos de formulario y <body>
onkeyup	Soltar una tecla pulsada	elementos de formulario y <body>
onload	La página se ha cargado completamente	<body>
onmousedown	Pulsar (sin soltar) un botón del ratón	todos los elementos
onmousemove	Mover el ratón	todos los elementos
onmouseout	El ratón "sale" del elemento (pasa por encima de otro elemento)	todos los elementos

onmouseover	El ratón "entra" en el elemento (pasa por encima del elemento)	todos los elementos
onmouseup	Soltar el botón que estaba pulsado en el ratón	todos los elementos
onreset	Inicializar el formulario (borrar todos sus datos)	<form>
onresize	Modificar el tamaño de la ventana del navegador	<body>
onselect	Seleccionar un texto	<input>, <textarea>
onsubmit	Enviar un formulario	<form>
onunload	Abandonar o cerrar la página	<body>

Ejemplos del uso de algunos eventos:

- Al hacer doble click en un párrafo se mostrará una ventana emergente con un texto:

* Haz doble clic para que se produzca el evento *

- Al hacer click en un párrafo ejecutara una función que posee varias interacciones dentro:

* Abrir dos ventanas emergentes *

- Llamar a una función que cambia una propiedad CSS al hacer colocar el mouse sobre un objeto:

```
function italica() {
        document.getElementById("texto").style.fontStyle = "italic";
}
```

document.getElementById("texto").onmouseover = italica

- Código que se ejecutara luego que el toda la página o código html se halla cargado completo:

  window.onload = function() {

  aca va el código javascript que debe ser cargado después de la página

  }

En los ejemplos anteriores modificamos un elemento diferente, pero si queremos que el evento altere o afecte al mismo elemento que lo acciono, debemos usar la palabra reservada “THIS”, y quedaría de la siguiente manera:

sí tenemos el siguiente evento

<p id="cambiar" onclick=" document.getElementById('cambiar').style.color = 'green'" </p>

lo cambiamos para que afecte al mismo elemento de la siguiente manera:

<p id="cambiar" onclick="this.style.color='green'" </p>

Entre los eventos de teclado debemos diferenciar 3 que tienden a crear confusión y son **onkeydown**, **onkeypress** y **onkeyup**. parecieran lo mismo pero no lo son, incluso podemos usar 3 funciones o acciones distintas a la misma tecla usando estos 3 métodos.

Cuando un usuario pulsa una tecla se producen los tres eventos anteriores en el siguiente orden:

- **onkeydown**: Corresponde al hecho de pulsar una tecla y no soltarla.

- **onkeypress**: Corresponde a la propia pulsación de la tecla.
- **onkeyup**: Corresponde al hecho de soltar una tecla que estaba pulsada.

REFERENCIAS:

- Introducción a DOM
  https://developer.mozilla.org/es/docs/Web/API/Document_Object_Model/Introduction

- Document Objet Model
  https://lenguajejs.com/javascript/dom/que-es/

- Elementos del DOM y trabajar con estilos
  https://www.digitalocean.com/community/tutorials/como-modificar-atributos-clases-y-estilos-en-el-dom-es