JavaScript
DOM

CertiDevs

Índice de contenidos

1. Introducción al DOM

DOM Standard

A través del **Document Object Model (DOM)**, JavaScript puede acceder e interactuar con los diferentes elementos HTML.

El **DOM** se define como un árbol de objetos que el navegador crea cuando carga el navegador siguiendo la estructura de etiquetas html definida en el documento html renderizado.

El **DOM** es un estándar que sirve como interfaz para permitir a los scripts como por ejemplo de JavaScript acceder y modificar elementos HTML de forma dinámica. Cuando hablamos de modificar algo programáticamente significa que hemos programado un código que realiza esas acciones.

El Document Object Model (DOM) representa un árbol de objetos en HTML. Haciendo uso del DOM JavaScript es posible:

- Cambiar los elementos HTML de la página
- Cambiar los atributos de los elementos HTML de la página
- Cambiar los estilos CSS de los elementos HTML de la página
- Eliminar elementos / atributos HTML en la página
- Añadir nuevos elementos y atributos
- Reaccionar a posibles eventos: onclick, onchange....
- Crear nuevo eventos en la página

Ejemplo de script con función JavaScript que modifica un elemento HTML para agregar una clase CSS:

```
<script>
function hideShowImg(){
    console.log("Ejecutando la función hideShowImg");

    // Uso de DOM para acceder al elemento con id: main-img
    let imgElement = document.getElementById("main-img");

    // Compruebo si el elemento ya tiene la clase
    // Si ya la tiene entonces se la quito, si no la tiene se la añado
    if(imgElement.classList.contains("hidden-img")) {
        imgElement.classList.remove("hidden-img");
    } else{
        imgElement.classList.add("hidden-img");
    }
}
</script>
```

Document Object Model (DOM) representa todo el contenido de la página como objetos que se pueden modificar.

El objeto `document` es el principal "punto de entrada" a la página. Podemos cambiar o crear cualquier cosa en la página usándolo. Ejemplo de cómo cambiar el color de fondo:

```
// change the background color to green
document.body.style.background = "green";
```

2. Nodos

Hay 12 tipos de nodos en DOM.

https://dom.spec.whatwg.org/#node

Normalmente trabajamos con 4 tipos de nodos, pero hay 12:

- const unsigned short ELEMENT_NODE = 1;
- const unsigned short ATTRIBUTE_NODE = 2;
- const unsigned short TEXT_NODE = 3;
- const unsigned short CDATA_SECTION_NODE = 4;
- const unsigned short ENTITY_REFERENCE_NODE = 5; // legacy
- const unsigned short ENTITY_NODE = 6; // legacy
- const unsigned short PROCESSING_INSTRUCTION_NODE = 7;
- const unsigned short COMMENT_NODE = 8;
- const unsigned short DOCUMENT_NODE = 9;
- const unsigned short DOCUMENT_TYPE_NODE = 10;
- const unsigned short DOCUMENT_FRAGMENT_NODE = 11;
- const unsigned short NOTATION_NODE= 12; // legacy

Podemos saber el tipo de nodo que es un elemento gracias al atributo nodeType:

- `elem.nodeType` tiene valor 3 entonces indica que es un nodo texto.

3. Atributos DOM

Algunos de los atributos en la interfaz `Document`:

- document.documentElement
- document.nodeName
- document.childNodes
- document.body

- document.head
- document.domain
- document.forms
- document.images
- document.links
- document.scripts
- document.embeds
- document.anchors
- document.title
- document.on… para asignar eventos

Algunos de los atributos en la interfaz `HTMLElement`:

- element.innerHTML
- element.innerText
- element.nodeValue
- element.attribute. Ejemplo: `myElement.src = "logo.jpg";`
- element.style.property
- element.on(…) para asignar eventos
- element.onclick
- element.onchange
- element.onmouseover
- element.onmouseout
- element.onfocus
- element.parentNode
- element.childNodes
- element.firstChild
- element.lastChild
- element.nextSibling
- element.previousSibling

Las colecciones DOM como por ejemplo element.childNodes se mantienen actualizadas, es decir, que si alguno de los nodos sufre cambios son visibles directamente en la referencia que tengamos sin necesidad de volver a cargar de nuevo element.childNodes.

4. Métodos DOM

Métodos para crear nuevos nodos:

- document.createElement(tag) – crea un elemento con la etiqueta dada
- document.createTextNode(value) – crea un nodo de texto (rara vez se usa),
- elem.cloneNode(deep) – clona el elemento. Si `deep==true` entonces con todos los descendientes.

Inserción y extracción:

- node.append(... nodos o cadenas) – insertar en el nodo, al final,
- node.prepend(... nodos o cadenas) – insertar en el nodo, al principio,
- node.before(... nodos o cadenas) –- insertar justo antes del nodo,
- nodo.after(...nodos o cadenas) –-insertar justo después del nodo,
- node.replaceWith(...nodos o cadenas) –- reemplazar nodo.
- node.remove() –- elimina el nodo.

Las cadenas de texto se insertan "como texto".

Métodos "old school":

- parent.appendChild(nodo)
- parent.insertBefore(nodo, nextSibling)
- parent.removeChild(nodo)
- parent.replaceChild(nuevoElem, nodo)

Todos estos métodos devuelven node.

Dado un html, el método `elem.insertAdjacentHTML(where, html)` lo inserta dependiendo del valor del parámetro `where`:

- "beforebegin": inserta html justo antes de elem.
- "afterbegin": inserta html en elem, al principio.
- "beforeend": inserta html en elem, al final.
- "afterend": inserta html justo después de elem.

Algunos de los métodos de `Document`:

- document.getElementById()
- document.getElementsByTagName()
- document.getElementsByClassName()
- document.querySelectorAll()
- Permite buscar por selectores CSS, el más versátil de todos.
- document.createElement()
- document.createTextNode()
- document.removeChild()

- document.appendChild()
- document.replaceChild()
- document.write()

Algunos de los métodos de `HTMLElement`:

- element.setAttribute
- element.addEventListener
- element.removeEventListener
- element.removeChild
- element.replaceChild
- element.insertBefore
- element.append
- element.prepend
- element.remove
- element.matches(css)
- element.closest(css)
- elementA.contains(elementB)
- elements.insertAdjacentHTML(where, html) donde where toma uno de los siguientes valores:
- "beforebegin"
- "afterbegin"
- "beforeend"
- "afterend"
- elements.insertAdjacentElement

5. Node vs Element

Un nodo es el nombre genérico de cualquier tipo de objeto en la jerarquía DOM. Un nodo podría ser cualquiera de los elementos DOM incorporados, como documento o `document.body`, podría ser una etiqueta HTML especificada en el HTML, como `<input>` o `<p>`, o podría ser un nodo de texto creado por el sistema para contener un bloque de texto dentro de otro elemento. Entonces, en pocas palabras, un nodo es cualquier objeto DOM.

Un elemento es un tipo específico de nodo, ya que hay muchos otros tipos de nodos (nodos de texto, nodos de comentarios, nodos de documentos, etc.).

El DOM consta de una jerarquía de nodos donde cada nodo puede tener un padre, una lista de nodos secundarios y un hermano siguiente y un hermano anterior. Esa estructura forma una jerarquía en forma de árbol. El nodo del documento tiene el nodo html como su hijo. El nodo html tiene su lista de nodos secundarios (el nodo principal y el nodo del cuerpo). El nodo del cuerpo tendría su lista de nodos secundarios (los elementos de nivel superior en su página HTML) y así

sucesivamente.

Entonces, una lista de nodos es simplemente una lista de nodos similar a una matriz.

Un elemento es un tipo específico de nodo, uno que se puede especificar directamente en el HTML con una etiqueta HTML y puede tener propiedades como una identificación o una clase. puede tener hijos, etc... Existen otros tipos de nodos como nodos de comentarios, nodos de texto, etc... con diferentes características. Cada nodo tiene una propiedad .nodeType que informa qué tipo de nodo es.

Por ejemplo, ELEMENT_NODE es un tipo particular de nodo donde la propiedad nodeType tiene un valor de 1.

Entonces document.getElementById("test") solo puede devolver un nodo y se garantiza que sea un elemento (un tipo específico de nodo). Por eso, simplemente devuelve el elemento en lugar de una lista.

Dado que document.getElementsByClassName("para") puede devolver más de un objeto, los diseñadores optaron por devolver una nodeList porque ese es el tipo de datos que crearon para una lista de más de un nodo. Dado que estos solo pueden ser elementos (solo los elementos suelen tener un nombre de clase), técnicamente es una lista de nodos que solo tiene nodos de tipo elemento y los diseñadores podrían haber creado una colección con un nombre diferente que fuera una lista de elementos, pero eligieron usar solo un tipo de colección ya sea que tenga solo elementos en ella o no.

HTML5 define una HTMLCollection que es una lista de elementos HTML (no cualquier nodo, solo elementos). Varias propiedades o métodos en HTML5 ahora devuelven una HTMLCollection. Si bien es muy similar en interfaz a una lista de nodos, ahora se hace una distinción en el sentido de que solo contiene Elementos, no cualquier tipo de nodo.

La distinción entre una nodeList y una HTMLCollection tiene poco impacto en la forma en que usa, pero los diseñadores de HTML5 ahora han hecho esa distinción.

Por ejemplo, la propiedad element.children devuelve una HTMLCollection activa.

6. Eventos

Cuando usamos `element.addEventListener`, la **propagación de eventos** es una forma de definir el orden de los elementos cuando ocurre un evento. Si tiene un elemento `<p>` dentro de un elemento `<div>` y el usuario hace clic en el elemento `<p>`, ¿Qué evento de "clic" de elemento debe manejarse primero?. Dos técnicas: bubbling vs. capturing.

- **Bubbling**: el evento del elemento más interno se maneja primero y luego el externo: el evento de clic del elemento `<p>` se maneja primero, luego el evento de clic del elemento `<div>`.
- **Capturing**: el evento elemento más externo se maneja primero y luego el interno: el evento de clic del elemento `<div>` se manejará primero, luego el evento de clic del elemento `<p>`.

Por defecto `element.addEventListener(event, function, useCapture)` aplica false en el último parámetro, lo que quiere decir que por defecto la propagación es bubbling y por tanto se maneja el

elemento más interno primero.

7. HTML

Un documento HTML/XML se representa dentro del navegador como el árbol DOM.

- Las etiquetas se convierten en nodos de elementos y forman la estructura.
- El texto se convierte en nodos de texto.
- ...etc, todo en HTML tiene su lugar en DOM, incluso los comentarios.

Podemos usar herramientas de desarrollo para inspeccionar DOM y modificarlo manualmente.

Hay una documentación extensa sobre las herramientas para desarrolladores de Chrome en https://developers.google.com/web/tools/chrome-devtools para estudiar las herramientas que proporcionan los navegadores para interactuar con el DOM.

Los nodos DOM tienen propiedades y métodos que nos permiten viajar entre ellos, modificarlos, movernos por la página y más.

8. BOM

El modelo de objetos del navegador (BOM) representa objetos adicionales proporcionados por el navegador (entorno host) para trabajar con todo excepto el documento.

Por ejemplo: El objeto navegador proporciona información básica sobre el navegador y el sistema operativo. Hay muchas propiedades, pero las dos más conocidas son:
* `navigator.userAgent` sobre el navegador actual.
* `navigator.platform` sobre la plataforma (puede ayudar a diferenciar entre Windows/Linux/Mac, etc.).

El objeto `location` nos permite leer la URL actual y puede redirigir el navegador a una nueva. Ejemplo

```
alert(location.href); // current URL
if (confirm("Go to Wikipedia?")) {
  location.href = "https://wikipedia.org"; // redirect
}
```

Las funciones de `alert`/`confirm`/`promt` también forman parte de BOM: no están directamente relacionadas con el documento, sino que representan métodos del navegador para comunicarse con el usuario.

BOM se puede encontrar dentro de HMTL Standard y WHATWG Standards.